999-802-200IS

System Programmer's Guide

AT&T Personal Computer 6300

Written by
Agora Resources, Inc.
Lexington, MA

NOTICE

The information in this document is subject to change without notice. AT&T assumes no responsibility for any errors that may appear in this document.

MS™-DOS is a trademark of Microsoft Corp.

Contents

1 System Programming Concepts

Purpose of this Manual 1-2
Notation 1-3
Programming Steps 1-4

2 MS-LINK

Overview 2-2
MS-LINK File Usage 2-3
Segments, Groups, and Classes 2-7
Invoking MS-LINK 2-9
Sample MS-LINK Session 2-21
MS-LINK Error Messages 2-24

3 DEBUG

Overview 3-2
How to Invoke DEBUG 3-3
Debugging Commands 3-6
Command Parameters 3-7
DEBUG Error Messages 3-44

4 8086 Addressing Scheme

Overview 4-2
The 20-Bit Address 4-3
Aligned and Non-Aligned Words 4-5
Registers and Flags 4-6
Code, Data, and Stack Segments 4-11
Addressing Modes 4-12

5 Memory Maps Control Blocks Diskette Allocation

Overview 5-2
The Address Space 5-3
Low Memory Map 5-4
ROM BIOS Data Area 5-5
File Control Blocks 5-6
ASCIIZ Strings 5-11
Handles 5-12
Diskette Layout 5-13
Diskette Directory 5-14
File Allocation Table 5-18
Diskette Formats 5-22

6 Program File Structure and Loading

Overview 6-2
Pros and Cons for Selecting a Program Format 6-3
EXE2BIN 6-5
File Header Format 6-10
Relocation Process for .EXE Files 6-13
Program Segment Prefix 6-15
Program Loading Process 6-18

7 System Calls

Quick Reference: Functions and Interrupts 7-2
Overview 7-5
Programming Considerations 7-6
Interrupts 7-7
Functions 7-8
System Calls Description 7-10

8 ROM BIOS Service Routines

Overview 8-2
Conventions 8-3
Interrupt Vector List 8-4
Video Control 8-5
Diskette Services 8-17
Communications Services 8-19
Keyboard Handling 8-22
Printer Routines 8-29
Miscellaneous ROM-BIOS Services 8-30
Bypassing the BIOS 8-32
CONFIG.SYS 8-33
ROM BIOS Listing 8-37
ROM BIOS Change List 8-177
Notes on Enhancements in ROM BIOS 1.21 8-189
ROM Revision 1.1 to ROM Revision 1.21 Source File Differences 8-193

9 MS-DOS Device Drivers

Overview 9-2
MS-DOS Device Drivers 9-8
Asynchronous Communications Element 9-27
DMA Controller 9-36
Floppy Diskette Interface and Controller 9-45
Hard Disk Controller 9-66
Keyboard Interface 9-94
Parallel Printer Interface 9-100
Programmable Interrupt Controller 9-105
Programmable Interval Timer 9-116
Real Time Clock and Calendar 9-123
Serial Communications Controller 9-128
Speaker 9-147
Video Controller 9-150

Supplement: The Display Enhancement Board

1 System Programming Concepts

- Purpose of this Manual
- Notation
- Programming Steps

Purpose of this Manual

This guide provides you with in-depth information on the AT&T Personal Computer program development tools. The guide focuses on what you need to know to make use of the existing AT&T Personal Computer 6300 hardware and hardware interfaces.

The final chapter on programming devices assumes that you have a working knowledge of the principles of designing device drivers and need the technical details on how to program the AT&T Personal Computer.

Notation

The following syntax is used throughout this manual in descriptions of command and statement syntax:

[ ] Square brackets indicate that the enclosed entry is optional.

{ } Braces indicate a choice between two or more entries. At least one of the entries enclosed in braces must be chosen.

... Ellipses indicate that an entry may be repeated as many times as needed.

This guide contains examples of prompts and messages displayed on the screen. These system-displayed items are indented from the main body of the text so that you can easily distinguish them. For example, MS-LINK prompts:

```
OBJECT MODULES[.OBJ]
```

Descriptions or examples that show a required response are indented and presented in boldface type:

```
LINK OBJ1+OBJ2+OBJ3,MAP
```

Programming Steps

This section shows where to go for information on the task you are performing.

Running High-Level Language

If you are running a program via the BASIC interpreter, the section on "System Calls" is applicable, since you can call these functions via a BASIC program.

If you are running a compiled program, read the section on MS-LINK as well as the section on System Calls.

Writing Assembler Programs

The first eight chapters are aimed at programmers writing assembler programs. If you have not used the 8088 or 8086 assembly language, the section "8086 Addressing Scheme" gives you a good start. The sections on the linker and debugger are fundamental to writing and debugging assembler programs. Also read the sections on "System Calls" and "ROM BIOS Service Calls."

Writing Utilities

If you are writing a supplementary utility program, read the sections on assembly programs, the sections on "Memory Maps, Control Blocks, and Diskette Allocation," and "Program File Structure and Loading."

Programming Devices Directly

Every section applies to writing device drivers, especially the chapter on "MS-DOS Device Drivers."

2 MS-LINK

- Overview
- MS-LINK File Usage
- Segments, Groups, and Classes
- Invoking MS-LINK
- Sample MS-LINK Session
- MS-LINK Error Messages

Overview

MS-LINK is an executable program on your DOS Supplemental Programs diskette. MS-LINK combines object modules that are the output of the MACRO-86 assembler or a compatible compiler. It produces a relocatable run file (load module) and a list file of external references and error messages.

To run MS-LINK, you provide object, run, list, and library file parameters. You may optionally enter switches that modify the operation of MS-LINK.

"Invoking the Linker" describes the three ways to run MS-LINK: interactive entry, command line entry, and automatic response file entry. Interactive entry is used most frequently, so its section contains information common to all three methods.

If you are linking a high-level language program, the compiler determines the arrangement of your object modules in memory. If you are using assembler, however, you have more control over your program's organization. The section "Segments, Groups, and Classes" shows you how to specify the order of your object modules at run time.

MS-LINK File Usage

The link process involves the use of several files.

MS-LINK:

- Works with one or more input files
- Produces two output files
- Creates a temporary disk file if necessary
- Searches up to eight library files

The format for MS-LINK file specifications is the same as that of any disk file:

Syntax [d:][path]filename[.ext]

d: the drive designation. Permissible drive designations for MS-LINK are A: through O:.

path a path of directory names.

filename any legal filename of one to eight characters.

ext a one- to three-character extension to the filename.

If no filename extensions are given in the input (object) file specifications, MS-LINK recognizes the following extensions by default:

.OBJ Object
.LIB Library

MS-LINK appends the following default extensions to the output (Run and List) files:

.EXE Run (may not be overridden)
.MAP List (may be overridden)

VM.TMP File MS-LINK uses available memory for the link session. If an output file exceeds available memory, MS-LINK creates a temporary file, names it VM.TMP, and puts it on the disk in the default drive. If MS-LINK creates VM.TMP, it will display the message:

```
VM.TMP has been created.
Do not change diskette in drive, <d:>
```

Once this message is displayed, do not remove the diskette from the default drive until the link session ends. If the diskette is removed, the operation of MS-LINK is unpredictable and MS-LINK usually displays the error message:

```
Unexpected end of file on VM.TMP
```

MS-LINK writes the contents of VM.TMP to the file named following the Run File: prompt. VM.TMP is a working file only and is deleted at the end of the linking session.

Do not use VM.TMP as a filename for any file. If MS-LINK requires the VM.TMP file, MS-LINK deletes the VM.TMP already on disk and creates a new VM.TMP. Thus, the contents of the previous VM.TMP file are lost.

Changing diskettes

You may want to change diskettes during the link operation. If MS-LINK cannot find an object file on the specified diskette, it prompts you to change diskettes rather than aborting the session. If you enter the /PAUSE switch, MS-LINK pauses and prompts you to change diskettes before it creates the run file. You may change diskettes when prompted except in the following cases:

- the diskette you want to change has a VM.TMP file on it.
- you have requested a list file on the diskette you want to change.

Segments, Groups, and Classes

Below terms are explained to help you understand how MS-LINK works. Generally, if you are linking object modules from a high-level language compiler, you do not need to know these terms. If you are linking assembly language modules, read this section carefully.

Segment

The segment is one of the most basic units of program memory organization. A segment is a contiguous area of memory up to 64K bytes long, and may be located anywhere in RAM. The contents of a segment are addressed by a segment:offset address pair, where "segment" is the segment's base or lowest address (see "The 20-Bit Address" in chapter 4).

Each segment has a class name in addition to its segment name. All segments with the same class name are loaded into memory contiguously by the linker from the first segment of that class to the last.

Class

A class is a collection of related segments. By naming the segments of your assembly language program to classes, you control the order in which they are loaded into memory (for high level languages, the compiler does this for you).

MS-LINK loads segments into memory on a class-by-class basis. Starting with the first class encountered in the first object file, all of the segments of each class are loaded. Within each class, the linker loads the segments in the order in which it finds them in the object files. Therefore, you can control the order in which classes are loaded by the order in which segments from different classes appear in the object files.

To ensure that classes are loaded in the order you desire, you can create a dummy module to feed to the linker as the first object file. This module declares empty-segment classes in the order you want the classes loaded. For example, one such file might look like this:

```
A   SEGMENT 'CODE'
A   ENDS
B   SEGMENT 'CONST'
B   ENDS
C   SEGMENT 'DATA'
C   ENDS
D   SEGMENT STACK 'STACK'
D   ENDS
```

If this method is used, be sure to declare all the classes used in your program in the dummy module; otherwise, you lose absolute control over the ordering of classes. Also, this method should only be used when linking assembly language programs. Do not create a dummy module if linking object files for a compiler, or unpredictable results may occur. Classes may be any length.

Group

Just as classes allow you to combine segments in a way that is logical, groups combine segments in 64K byte chunks to make them easily addressable. The segments in a group need not be contiguous, but when loaded they must fit within 64K bytes. This way each segment in the group can be fully addressed by an offset to one segment address, which is the start address of the lowest segment in the group. Segments are named to groups by the assembler or compiler or, as is possible in assembly language programs, by the programmer. Note that a segment can be large enough to be an entire group by itself.

Invoking MS-LINK

Ways to Invoke MS-LINK

MS-LINK is invoked in one of three ways. The first method, interactive entry, requires you to respond to individual prompts.

For the second method, command line entry, type all commands on the same line used to start MS-LINK.

To use the third method, automatic response file entry, create a response file that contains all the necessary commands and tell MS-LINK where that file is when you run MS-LINK.

Interactive Entry	LINK
Command Line Entry	LINK filenames[/switches]
Automatic Response File Entry	LINK @filespec

Interactive Entry

To invoke MS-LINK interactively, type:

```
LINK
```

MS-LINK loads into memory, then displays four prompts, one at a time. At the end of each line, after typing your response to the prompt, you may type one or more switches preceded by a forward slash.

The command prompts are summarized below. Defaults appear in square brackets ([]) after the prompt. **Object Modules** is the only prompt that requires a response.

MS-LINK Prompts

Prompt	Responses
Object Modules[.OBJ]:	[d:][path]filename[.ext] [+[[d:][path]filename[.ext]]...]
Run File[filename.EXT]:	[d:][path][filename[.ext]]
List File[NUL.MAP]:	[d:][path][filename[.ext]]
Libraries[.LIB]:	[d:][path][filename[.ext]] [+[d:][[path]filename[.ext]]...]

Notes:

- If you enter a filename without specifying the drive, the default drive is assumed. If you enter a filename without specifying the path, the default path is assumed.The libraries prompt is an exception — if the linker looks for the libraries on the default drive and doesn't find them, it looks on the drive specified by the compiler.

- To select default responses to all remaining prompts, use a single semicolon (;) followed immediately by <return> at any time after the second prompt (Run File:).

  Once you enter the semicolon, you can no longer respond to any of the prompts for that link session. Use the <RETURN> key to skip prompts.

- Use <CONTROL-C> to abort the link session at any time.

Object Modules to be Included

Object Modules [.OBJ]:

List .OBJ files to be linked. They must be separated by blank spaces or plus signs (+). If the plus sign is the last character typed, this prompt will reappear so that you can enter more object modules.

MS-LINK assumes that object modules have the extension .OBJ unless you explicitly specify some other extension. Object filenames may not begin with the @ symbol (@ is used for specifying an automatic response file).

The order in which you key in the the object files is significant. See section on segments, groups, and classes for more information.

Load Module

Run File [Obj-file.EXE]:

Give filename for executable object code. The default is: <first-object-filename>.EXE. (You cannot change the output extension.)You can specify just the drive designation or just a path for this prompt.

Listing

List File [NUL.MAP]:

Give filename for listing (also known as a linker map). The listing is not created if you select the default. You can request a listing by entering a drive designator, path, or filename[.ext]. If you do not specify an extension, the default .MAP is used.

You can have the listing printed by specifying a print device instead of a filename or have the listing displayed on the screen by specifying CON. If you display the linker map, you can also print it by pressing Ctrl-PrtSc.

Libraries to be Searched

Libraries [.LIB]:

List filenames to be searched separated by blank spaces or plus signs (+). If a plus sign is the last character typed, the prompt will reappear.

MS-LINK searches library files in the order listed to resolve external references. When it finds the module that defines the external symbol, MS-LINK processes that module as another object module.

There is no default library search for MACRO assembler object modules. For compiled modules, if you select the default for this prompt, MS-LINK looks for the compiler package's library on the default drive. If not found there, MS-LINK looks on the drive specified by the compiler.

If MS-LINK cannot find a library file, it displays:

```
Cannot find library <library-name>
Type new drive letter:
```

Press the letter for the drive designation (for example, B).

If two libraries have the same filename, only the first in the list is searched.

MS-LINK Switches

The seven MS-LINK switches control various MS-LINK functions. Type switches at the end of a prompt response regardless of which method you use to start MS-LINK. Switches may be grouped at the end of any response, or may be scattered at the end of several. Even if you type more than one switch at the end of one response, each switch must be preceded by a forward slash (/).

All switches may be abbreviated. The only restriction is that an abbreviation must be sequential from the first letter through the last typed; no gaps or transpositions are allowed. For example:

Legal	Illegal
/D	/DSL
/DS	/DAL
/DSA	/DLC
/DSALLOCA	/DSALLOCT

/DSALLOCATE

/DSALLOCATE tells MS-LINK to load all data at the high end of the Data Segment. Otherwise, MS-LINK loads all data at the low end of the Data Segment. At runtime, the DS pointer is set to the lowest possible address to allow the entire DS segment to be used. Use of /DSALLOCATE in combination with the default load low (that is, the /HIGH switch is not used) permits the user application to dynamically allocate any available memory below the area specifically allocated within DGroup yet to remain addressable by the same DS pointer. This dynamic allocation is needed for Pascal and FORTRAN programs.

Your application program may dynamically allocate up to 64K bytes (or the actual amount of memory available) less the amount allocated within DGroup.

/HIGH

/HIGH causes MS-LINK to place the Run file as high as possible in memory. Otherwise, MS-LINK places the Run file as low as possible.

Note:
Do not use /HIGH with Pascal or FORTRAN programs.

/LINENUMBERS

/LINENUMBERS tells MS-LINK to include in the List file the line numbers and addresses of the source statements in the input modules. Otherwise, line numbers are not included in the List file.

Not all compilers produce object modules that contain line number information. In these cases, of course, MS-LINK cannot include line numbers.

/MAP

/MAP directs MS-LINK to list all public (global) symbols defined in the input modules. If /MAP is not given, MS-LINK will list only errors (including undefined globals).

The symbols are listed alphabetically. For each symbol, MS-LINK lists its value and its segment:offset location in the Run file. The symbols are listed at the end of the List file.

/PAUSE

/PAUSE causes MS-LINK to pause in the link session when the switch is encountered. Normally, MS-LINK performs the linking session from beginning to end without stopping. This switch enables you to swap the diskettes before MS-LINK outputs the Run (.EXE) file.

When MS-LINK encounters /PAUSE, it displays the message:

```
About to generate .EXE file
Change disks <hit any key>
```

MS-LINK resumes processing when you press any key.

Note
Do not remove the disk which will receive the List file, or the disk used for the VM.TMP file, if one has been created.

/STACK: <number>

Stack number represents any positive numeric value (in hexadecimal radix) up to 65536 bytes. If a value from 1 to 511 is typed, MS-LINK will use 512. If /STACK is not used for a link session, MS-LINK calculates the necessary stack size automatically.

All compilers and assemblers should provide information in the object modules that allow the linker to compute the required stack size.

At least one object (input) module must contain a stack allocation statement. If not, MS-LINK will display the following error message:

```
WARNING: NO STACK STATEMENT
```

/NO /NO is short for NODEFAULTLIBRARY-SEARCH. This switch applies only to higher level language modules. This switch tells MS-LINK not to search the default libraries in the object modules. For example, if you are linking object modules in Pascal, specifying /NO tells MS-LINK not to automatically search the library named PASCAL.LIB to resolve external references.

Command Line Entry

Purpose

You may invoke MS-LINK by typing all commands on one line. The entries following LINK are responses to the command prompts. The entry fields for the different prompts must be separated by commas. Use the following syntax:

Syntax

LINK <obj-list>,<runfile>,<listfile>,
<lib-list>[/switch...]

obj-list

a list of object modules, separated by plus signs or spaces.

runfile

name of the file to receive the executable output.

listfile

name of the file to receive the listing.

lib-list

list of library modules to be searched, separated by spaces or plus signs.

/switch

refers to optional switches which may be placed following any of the response entries (just before any of the commas or after the <lib-list>, as shown).

To select the default for a field, simply type a second comma with no spaces between the two commas.

Example:

LINK FUN+TEXT+TABLE+CARE,
FUNLIST, COBLIB.LIB

This command causes MS-LINK to load. Then the object modules FUN.OBJ, TEXT.OBJ, TABLE.OBJ and CARE.OBJ are loaded. MS-LINK links the object modules and writes the output to FUN.EXE (by default), creates a List file named FUNLIST.MAP, and searches the library file COBLIB.LIB.

Automatic Response File Entry

It is often convenient to save responses to the linker for re-use at a later time. This is especially useful when a long list of object modules needs to be specified. The use of an automatic response file allows you to do this.

Before using this option, you must create the response file. Each line of text corresponds to one MS-LINK prompt. The responses must be typed in the same order as they are when entered interactively. To continue a line, type a plus sign (+) at the end of the line.

You can enter the name of more than one automatic response file on the command line and combine response file names with additional parameters. The combined series of resulting parameters must be a valid sequence of MS-LINK prompts.

Use switches and special characters (+ and ;) in the response file the same way they are used when entered interactively.

To invoke the linker using a response file, type

LINK @ <filespec>

Filespec is the name of a response file.

When the session begins, MS-LINK displays each prompt with the corresponding response from the response file. If the response file does not contain answers for all the prompts, MS-LINK displays the prompt which does not have a response and waits for a response. When you type a legal response, MS-LINK continues the link session.

Example:

```
FUN TEXT TABLE CARE
/PAUSE /MAP
FUNLIST
COBLIB.LIB
```

This response file tells MS-LINK to load the four object modules named FUN.OBJ, TEXT.OBJ, TABLE.OBJ, and CARE.OBJ. MS-LINK pauses before producing a public symbol map to permit you to swap disks. When you press any key, the output files will be named FUN.EXE and FUNLIST.MAP. MS-LINK will search the library file COBLIB.LIB.

Sample MS-LINK Session

This sample shows you the type of information displayed during an MS-LINK session.

In response to the MS-DOS prompt, type:

```
LINK
```

The system displays the following messages and prompts:

```
        Microsoft Object Linker V2.01 (Large)
(C) Copyright 1982,1983 by Microsoft Inc.

Object Modules [.OBJ]: IO SYSINIT
Run File [IO.EXE]:
List File [NUL.MAP]: PRN /MAP /LINE
Libraries [.LIB]: ;
```

Notes:

- By specifying /MAP, you get both an alphabetic listing and a chronological listing of public symbols.
- By responding PRN to the List File: prompt, you can redirect your output to the printer.
- By specifying the /LINE switch, MS-LINK gives you a listing of all line numbers for all modules. (Note that /LINE can generate a large volume of output.)

Once MS-LINK locates all libraries, the linker map displays a list of segments in the order of their appearance within the load module. The list might look like this:

```
Start     Stop      Length    Name
00000H    009ECH    09EDH     CODE
009F0H    01166H    0777H     SYSINITSEG
```

The information in the Start and Stop columns shows the 20-bit hex address of each segment relative to location zero. Location zero is the beginning of the load module.

The addresses displayed are not the absolute addresses where these segments are loaded. See the following section on the MS-LINK DEBUG program for information on how to determine the absolute address of a segment.

Because the /MAP switch was used, MS-LINK displays the public symbols by name and value. For example:

```
ADDRESS         PUBLICS BY NAME
009F:0012       BUFFERS
009F:0005       CURRENT DOS LOCATION
009F:0011       DEFAULT DRIVE
009F:000B       DEVICE LIST
009F:0013       FILES
009F:0009       FINAL DOS LOCATION
009F:000F       MEMORY SIZE
009F:0000       SYSINIT

ADDRESS         PUBLICS BY VALUE
009F:0000       SYSINIT
009F:0005       CURRENT DOS LOCATION
009F:0009       FINAL DOS LOCATION
009F:000B       DEVICE LIST
009F:000F       MEMORY SIZE
009F:0011       DEFAULT DRIVE
009F:0012       BUFFERS
009F:0013       FILES
```

The final line in the listing file describes the program's entry point:

```
Program entry point at 0009F:0000
```

MS-LINK Error Messages

All errors, except for the two warning messages, cause the link session to abort. After the cause has been found and corrected, MS-LINK must be rerun. The following error messages are displayed by MS-LINK:

Attempt to access data outside of segment bounds, possibly bad object module

There is probably a bad object file.

Bad numeric parameter

Numeric value is not in digits.

Cannot open temporary file

MS-LINK is unable to create the file VM.TMP because the disk directory is full. Insert a new disk. Do not remove the disk that will receive the List. MAP file.

Error: dup record too complex

DUP record in assembly language module is too complex. Simplify DUP record in assembly language program.

Error: fixup offset exceeds field width

An assembly language instruction references an address with a short or near instruction instead of a long or far instruction. Edit assembly language source and reassemble.

Input file read error

There is probably a bad object file.

Invalid object module

An object module(s) is incorrectly formed or incomplete (as when assembly is stopped in the middle).

Symbol defined more than once

MS-LINK found two or more modules that define a single symbol name.

Program size or number of segments exceeds capacity of linker

The total size may not exceed 384K bytes and the number of segments may not exceed 255.

Requested stack size exceeds 64K

Specify a size less than or equal to 64K bytes with the /STACK switch.

Segment size exceeds 64K

64K bytes is the addressing system limit.

Symbol table capacity exceeded

Very many and/or very long names were typed exceeding the limit of approximately 50K bytes.

Too many external symbols in one module

The limit is 256 external symbols per module.

Too many groups

The limit is ten groups.

Too many libraries specified

The limit is 8 libraries.

Too many public symbols

The limit is 1024 public symbols.

Too many segments or classes

The limit is 256 (segments and classes together must total 256 or less).

Unresolved externals: <list>

The external symbols listed have no defining module among the modules or library files specified.

VM read error

This is a disk error; it is not caused by MS-LINK.

Warning: no stack segment

None of the object modules specified contains a statement allocating stack space.

Warning: segment of absolute or unknown type

There is a bad object module or an attempt has been made to link modules that MS-LINK cannot handle (e.g., an absolute object module).

Write error in TMP file

No more disk space remains to expand the VM.TMP file.

Write error on run file

Usually, this means there is not enough disk space for the Run file.

3 DEBUG

- Overview
- How to Invoke DEBUG
- Debugging Commands
- Command Parameters

  A Assemble
  C Compare
  D Display
  E Enter
  F Fill
  G Go
  H Hexarithmetic
  I Input
  L Load
  M Move
  N Name
  O Output
  Q Quit
  R Register
  S Search
  T Trace
  U Unassemble
  W Write

- DEBUG Error Messages

Overview

The DEBUG utility is an executable object program that resides on your MS-DOS diskette. DEBUG performs the following functions:

- Allows you to single step through a program, instruction by instruction, for testing purposes.
- Changes register and file contents during the DEBUG session so that you can test a code change without reassembling your program.
- Makes permanent changes to diskette files so you can use DEBUG to recover files that may otherwise be lost.
- Supports a disassemble command so you can translate machine code instructions into their assembly language equivalents for testing purposes.

How to Invoke DEBUG

The DEBUG program is invoked as follows:

DEBUG [filespec [,arglist]]

filespec the name of the program file to be debugged.

arglist An optional list of file name parameters and switches. These will be passed to the program specified by the filespec parameter. When the program is loaded into memory, it is loaded as if it had been invoked with the command

filespec arglist

That is, filespec indicates the file to be debugged, and arglist is the rest of the command line that is used when the file is invoked and loaded in memory via COMMAND.COM.

If you enter DEBUG without parameters, since no file name has been specified, current memory, disk blocks, or disk files can be manipulated.

Comments

On entering the DEBUG environment DEBUG responds with the hyphen (-) prompt and underline (_) cursor. You now may enter any DEBUG command.

If you include the filespec in the command line, the specified file is loaded into memory starting at location 100 (hexadecimal). However, if you specify a file with a .EXE extension, the program is relocated to the address specified in the header of the file. See the chapter on "Program Structure and Loading" for information on the format of the file header.

If the file has the HEX extension, the file is loaded beginning at the address specified in the HEX file. HEX files are in INTEL hex format and are converted to memory image format by DEBUG.

All DEBUG commands may be aborted at any time by pressing <CTRL-C>. Pressing <CTRL-S> suspends the display, so that you can read it before the output scrolls away. After suspending the display, press any key (except <CTRL-S> or <CTRL-C>) to continue scrolling.

Examples

DEBUG <CR>.

The DEBUG session begins, but without loading a file.

DEBUG b:myprog <CR>.

The DEBUG environment is entered and the file named "myprog" is loaded into memory from drive B.

When you invoke DEBUG, it sets up a program segment prefix at offset 0 in the program work area. You can overwrite this area if you enter DEBUG without parameters. Moreover, if you are debugging a file with a COM or EXE extension, do not tamper with the program header below location 5CH, or DEBUG will terminate.

Do not restart a program after a "Program terminated normally" message is displayed. You must reload the program with the N and L commands for it to run properly.

Debugging Commands

This section describes the DEBUG commands in alphabetical order for ease of reference.

- Commands can be entered in either upper or lower case.
- Command keywords and command parameters can be separated from each other by spaces or commas for readability but need not be, except where two hexadecimal numbers are entered as parameters, in which case they must be separated by a comma or space. For brevity, the syntax of this chapter will always indicate a comma where separation is obligatory, but note that a space can alternatively be used.
- Commands only become effective after entering <CR>.
- If you make a syntax error when entering a command, the message "Error" will be displayed. You must re-enter the command using the correct syntax.

Command Parameters

The following DEBUG command parameters require definition.

address a hex value in one of the following formats:

- a segment register designation and a hex offset separated from each other by a colon. For example:

  DS:0300

- a hexadecimal segment and offset separated from each other by a colon. For example:

  9D0:0100

- a hexadecimal offset value. The DEBUG command will use a default segment value from either the DS or CS registers, depending on the command. For example:

  200

byte a one or two character hexadecimal value.

drive 0, 1, or 2 depending on whether you wish to select drive A, drive B or drive C, respectively.

range a range of addresses. The range can be specified as

address L value

where address specifies the start of the range and value specifies the length of the range. For example:

DS:300L30

indicates a range of 48 locations starting at address 300 in the segment indicated by the DS register.

The specified range cannot be greater than 10000 (hexadecimal). To specify this value enter 0000 (or 0) as the value parameter.

A range can also be specified as:

address,address

where the two addresses indicate the limits of the range. A space may be used instead of a comma.

value a 1 to 4 character hexadecimal value.

A (ASSEMBLE)

Assembles 8086 mnemonics directly into memory.

Syntax

A [address]

Address is the start address into which the subsequently entered line of mnemonics is to be assembled. If this parameter is omitted, offset 100 from the segment in the CS register is assumed, if you did not enter an Assemble command previously. If you did enter Assemble previously, the code assembles into the address following the last instruction loaded by the previous Assemble command.

Comments

- After you enter the Assemble command, DEBUG displays the specified address followed by the cursor. You may then enter a line of 8086 assembler mnemonics. On terminating the line with <CR>, the line will be assembled into memory starting at the specified location. The address of the byte subsequent to the assembled code will be displayed on the next line along with the cursor to enable you to enter the next line of code. If, instead of a line of 8086 mnemonics, you simply enter <CR>, the Assemble command terminates and the DEBUG prompt reappears.
- All numeric values are hexadecimal and must be entered as 1 to 4 characters without a trailing H. Prefix mnemonics must be specified in front of the opcode to which they refer. You may also enter them on a separate line.
- The segment override mnemonics are CS:, DS:, ES: and SS:. The mnemonic for the far return is RETF. String manipulation mnemonics must explicitly state the string size. For example, use MOVSB to move byte strings.

- The Assemble command will automatically assemble short, near, or far jumps and calls, depending on byte displacement with respect to the destination address. These may be overridden with the NEAR or FAR prefix. For example:

```
0100:0500 JMP 502          ;a two-byte
                           ;short jump
0100:0502 JMP NEAR 505     ;a three-byte
                           ;near jump
0100:505 JMP FAR 50A       ;a five-byte far
                           ;jump
```

  The NEAR prefix may be abbreviated to NE, but the FAR prefix cannot be abbreviated.

- DEBUG cannot tell whether some operands refer to a word memory location or to a byte memory location. In this case the data type must be explicitly stated with the prefix "WORD PTR" or "BYTE PTR". Acceptable abbreviations are "WO" and "BY". For example:

```
NEG    BYTE PTR [128]
DEC    WO [SI]
```

- DEBUG cannot distinguish whether an operand refers to a memory location or to an immediate operand. Enclose operands that refer to memory locations in square brackets. For example:

```
MOV AX,21          ;Load AX with 21H
MOV AX,[21]        ;Load AX with the contents of
                   ;location 21H
```

- Two pseudo-instructions are available with the Assemble command. The DB opcode will assemble byte values directly into memory. The DW opcode assembles word values into memory. For example:

```
DB 1,2,3,4,"THIS IS AN EXAMPLE"
DB 'THIS IS A QUOTE:" '
DB "THIS IS A QUOTE' "
DW 1000,2000,3000,"BACH"
```

- The Assemble command supports all forms of register indirect addressing. For example:

```
ADD BX,34[BP+2]. [SI-1]
POP [BP+DI]
PUSH [SI]
```

All opcode synonyms are supported. For example:

```
LOOPZ  100
LOOPE  100
JA     200
JNBE   200
```

Example

1. Enter **A200** <**CR**>.
2. DEBUG displays **09AC:0200**_ .
3. Enter **MOV AX,[21]** <**CR**>.
4. The 8086 mnemonics are assembled starting at location 200. The byte location subsequent to the assembled code is then displayed:

   09AC:0203_

5. Enter <**CR**>.
6. The Assemble command terminates and the DEBUG prompt reappears.

C (COMPARE)

Compares the contents of two areas of memory.

Syntax **C range,address**

range the range of addresses defining the first area to be compared. If no segment is specified, then the segment specified in the DS register is assumed.

address the start of the area to be compared with the area specified by the range parameter.

Comments

- The Compare command compares the area of memory specified by the range parameter with an area of the same size starting at the location specified by the address parameter.
- If the contents of the two areas are identical, nothing is displayed. If there are differences, then the differences are displayed in the form

  <address1> <contents1> <contents2> <address2>

  <address1> indicates the address in the first area and <contents1> its contents. <address2> indicates the corresponding address in the second area and <contents2> its contents.

Example **1** Enter **C100,1FF,300 <CR>** or

C100L100, 300 <CR>.

2 The area of memory from 100 to 1FF is compared with the area of memory from 300 to 3FF.

D (DISPLAY)

Displays an area of memory.

Syntax

D [range] or
D [address]

range the range of addresses whose contents are to be displayed. If you enter only offsets, then the segment specified in the DS register is assumed.

address the address from which the display is to start. The contents of this address and the subsequent 127 locations are displayed. If only an offset is entered, then the segment specified in the DS register is assumed.

Comments

- If D is specified without parameters, then the 128 bytes following the last address to be displayed are displayed. If no location has yet been accessed, the display will start from location DS:100.
- If D and the range parameter are specified, the contents of that range of addresses are displayed. If this takes more than 24 screen lines, the display is scrolled until the contents of the final address in the range are displayed on line 24.
- The display is displayed in two portions:

  A hexadecimal display, where each byte is represented by its hexadecimal value, and an ASCII display, where the equivalent ASCII character for the byte is displayed. If there is no corresponding printable ASCII character, a period (.) is displayed.

- Each line of the display begins with an address followed by the hexadecimal contents of the 16 bytes starting from the addressed location. The eighth and ninth bytes are separated by a hyphen (-). The right-hand columns display the equivalent ASCII values. Each line of the display, except possibly the first, begins on a 16 byte boundary.

Example

1 Enter **D 100,110 <CR>**.

2 Lines 100H to 110H (inclusive) are displayed.

3 Enter **D <CR>**.

4 The 128 bytes starting from location 111H are displayed.

5 Enter **D200 <CR>**.

6 The 128 bytes starting from location 200H are displayed.

E (ENTER)

Replaces the contents of memory locations at the byte address(es) specified.

Syntax **E address[,bytevalue[,bytevalue...]]**

address the address of the location whose value is to be replaced; or the address of the first of a succession of locations whose contents are to be replaced. If only an offset is specified, then the segment indicated by the DS register is assumed.

bytevalue the value that is to replace the contents of the specified address. The first bytevalue parameter will replace the contents of the location specified by the address parameter. A second bytevalue will replace the contents of the location following that specified by the address parameter, and so on.

Comments

- If the command is entered without the byte value list, then DEBUG displays the specified address and its contents. The Enter command then waits for you to perform one of the following:

  1. Replace the displayed bytevalue by entering another value. Enter the new value after the current value. If you enter an illegal value, or if you type more than two digits, the illegal or extra character is not echoed.

  2. Advance to the next byte by pressing <SPACE>. To change the value of this byte simply enter the value as described above. If you

advance beyond an eight-byte boundary, DEBUG starts a new display line with the address displayed at the start of the line. To advance to the next byte without changing the current byte, press <SPACE> again.

3 To return to the previous byte enter hyphen (-). DEBUG then starts a new display line with the address of the byte you have returned to and its contents. You can then change the contents of this location as described above. To move back one byte further without changing this value, enter hyphen again, and another new display line will be generated.

4 Terminate the Enter command by pressing <CR>. This key may be pressed in any byte position.

- If you specify byte values in the command line, then the first of these byte values will replace the contents of the location specified by the address parameter. Subsequent entries in the list of byte values will replace subsequent bytes in memory.

Example

1 Enter **E100 <CR>**.

2 DEBUG displays something like **058D:0100 CD._**

3 Enter 26.

4 the value of location 100 is changed to 26 and DEBUG displays:

058D:0100 CD.26_

5 Enter <**SPACE**>.

6 The next byte (location 101) is displayed

```
058D:0100 CD.26 20._
```

7 Enter <**SPACE**>.

8 The next byte (location 102) is displayed

```
058D:100 CD.26 20. 00._
```

9 Enter <->.

10 The previous byte (location 101) is displayed on the next line

```
058D:0100 CD.26 20. 00.
058D:0101 20._
```

11 Enter 30 <**CR**>.

12 The contents of location 101 are changed to 30 and the Enter command is terminated.

```
058D:0100 CD.26. 20. 00.
058D:0101 20.30
-_
```

13 Enter **E 200,26,0A,19,23** <**CR**>.

14 The contents of byte locations 200, 201, 202 and 203 are changed to 26, 0A, 19 and 23, respectively.

F (FILL)

Fills an area of memory with specified byte values.

Syntax

F range,bytevalue[,bytevalue...]

range

the range of addresses whose contents are to be overwritten with the specified bytevalues. If only the offset is specified, then the segment indicated by the DS register is assumed.

bytevalue

a two digit hexadecimal value that is to overwrite the contents of the specified address(es).

Comments

- If the specified range contains more bytes than the list of byte values, then the list of byte values is repeated until the specified range is filled.
- If the list of byte values is longer than the specified range, the extra byte values are ignored.

Example

1 Enter **F04BA:100L100,42,45,48,37,20** <CR>.

2 DEBUG fills memory locations **04BA:100** to **04BA:1FF** with the byte values specified. The five values are repeated until all 256 locations are filled.

G (GO)

Executes the program currently in memory, optionally halting at specified breakpoint(s) and displaying information about the system and program environment.

Syntax

```
G [=address][,address...]
```

=address the address in memory at which program execution is to start. "=" must be entered to distinguish a start address from a breakpoint address.

address the breakpoint address. You can specify up to ten breakpoints, in any order.

Comments

If you enter G without parameters, the program currently in memory is executed starting from the address specified by the CS and IP registers.

If you specify the =address parameter, the contents of the CS and IP registers are changed to those specified by the =address parameter and the program in memory is executed, starting from the address you specified.

If you specify one or more breakpoint addresses, program execution stops at the first such address encountered and displays the contents of the registers, the state of the flags and the next instruction to be executed (see the Register command for a description of the display).

- If only an offset is entered for an address, the GO command assumes the segment in the CS register.

- If you enter more than ten breakpoints, DEBUG will display

  BP Error

- Before executing the program, the GO command replaces the contents of the breakpoint locations with an interrupt instruction (hexadecimal CC). Therefore, each breakpoint address that you specify must point to the first byte of an 8086 instruction, or unpredictable results occur.

  When program execution halts at a breakpoint DEBUG restores the original values of all the specified breakpoint locations. However, if the program terminates normally (that is, not at a specified breakpoint), the original values are not restored.

  Note: Once a program has reached completion (DEBUG has displayed "Program terminated normally") you must reload the program before you can re-execute it.

  The stack segment must have six bytes available at the stack pointer for this command, otherwise unpredictable results occur. This is because the GO command jumps into the user program with the IRET instruction. The flag, CS, and IP registers have to be pushed onto the stack in preparation for the IRET, taking up six bytes.

Example

1. Enter **G=200,1AF,141 <CR>**.
2. The program currently in memory is executed starting from location 200. Assuming location 141 is encountered before 1AF, then the program halts at location 141 and the register and flag values are displayed along with the next instruction to be executed. If neither breakpoint location is encountered, then the program terminates normally.
3. Enter **G <CR>**.
4. If, in step two, the program had halted at location 141, then program execution continues from that address.

H (HEXARITHMETIC)

Calculates and displays the sum and the difference of two hexadecimal values.

Syntax

H value_a,value_b

value_a

The first of two hexadecimal values.

value_b

The hexadecimal value that is to be added to or subtracted from value_a.

Comments

The hexadecimal values may be up to four digits long.

The Hex command displays two four-digit values:

— the first is the result of adding value_b to value_a

— the second is the result of subtracting value_b from value_a

Example

1 Enter **H19F,10A <CR>**.

2 DEBUG displays

```
02A9 0095
```

3 Enter **HFFFF,2 <CR>**.

4 DEBUG displays

```
0001 FFFD
```

I (INPUT)

Inputs and displays (in hexadecimal) one byte from the specified port.

Syntax

```
I value
```

value the address of the port that the byte is to be input from.

Comments The port address can be up to 16 bits.

Example

1 Enter `I2F8`.

2 the byte at the addressed port is input and displayed.

L (LOAD)

Loads a file or absolute disk blocks into memory.

Syntax **L [address[,drive,block,count]]**

address the address in memory at which the file or range of blocks is to be loaded. If only an offset is entered, then the segment indicated by the CS register is assumed.

drive the drive from which disk blocks are to be loaded. For drive A you must enter 0, for drive B you must enter 1, etc.

block the first of a range of blocks to be loaded from the disk specified by the drive parameter.

count the number of blocks to be loaded.

Comments

- If all parameters are specified, then DEBUG loads blocks of information from disk into memory.
- If you enter L without parameters, or with just the address parameter, the file whose file control block is correctly formatted at location CS:5C is loaded into memory. The file control block at CS:5C is set either to the filespec specified when the DEBUG command was invoked, or to the filespec specified by the most recent "Name" command.
- The default location for programs to load is at CS:100. If you specify L and the address parameter, the file is loaded at the specified

address unless it is a .EXE or .HEX file. In any case DEBUG sets the BX:CX registers to the number of bytes loaded.

- If the file has an EXE extension, then it is relocated to the load address specified in the header of the .EXE file. That is, the address parameter to the Load command is ignored. The header itself is stripped off the .EXE file before the file is loaded into memory. Thus the size of the .EXE file on disk will differ from its size in memory.
- If the file is a .HEX file, entering the Load command with no parameters causes the file to be loaded starting at the address specified within the .HEX file. If the address parameter, however, is specified, then loading starts at the address which is the sum of the address specified and the address in the .HEX file.

Examples

The following examples assume the system to be initially in MS-DOS.

1 Enter **debug <CR>**
Nb:file.com <CR>
L <CR>.

Debug is entered and the subsequent Name command sets the file control block at CS:5C to identify file "file.com" on the diskette inserted in drive B. The Load command then loads this file into memory starting at CS:100 (the default address).

2 Enter **debug b:file.com <CR>**
L300 <CR>.

file.com is loaded into memory at location CS:100 by the DEBUG command. It is then relocated to CS:300 by the Load command.

M (MOVE)

Moves the contents of a specified range of memory addresses to the locations starting at a specified address.

Syntax **M range,address**

range The area of memory whose contents are to be moved. If you only enter an offset, the segment indicated in the DS register is assumed.

address The start of the destination area. If you only enter an offset, then the segment indicated by the DS register is assumed.

Comments If the source and destination areas overlap, the move is performed without loss of data.

The contents of the source area are not changed by the move, unless the destination area overlaps it.

If you specify an address as the end of the range, you must only enter the offset. The segment specified, or defaulted to, in the start address of the range is assumed.

Example

1 Enter **MCS:100,110,CS:500 <CR>** or **MCS:100L11,CS:500 <CR>**.

2 The 17 bytes starting at location CS:100 are copied to the 17 bytes starting at location CS:500.

N (NAME)

Provides file names for the Load and Write commands or file name parameters for the program to be debugged.

Syntax

N filespec[,filespec...]

filespec

the file specifier of a file to be loaded into memory, written to diskette, or used as a file name parameter to the file currently in memory.

Comments

The Name command can be used to provide:

the name of the disk file to be loaded into memory by a subsequent Load command

the name to be assigned to the file currently in memory when the file is subsequently written to disk

file name parameters to the file in memory to be debugged.

The first case enables you to specify the file you wish to debug after entering the DEBUG environment. That is, you can enter DEBUG without specifying parameters, then use the Name command to name the disk file you wish to debug, then load the file into memory using the Load command. This has the same effect as entering the file name as the first parameter to the DEBUG command upon invocation. In either case the file control block for the file to be debugged is set up at location CS:5C and the file is loaded.

In the second case, the file is already in memory and the Name command sets up the file control block for the specified file name at location CS:5C. When a Write command is subsequently entered the file in memory is written to disk with the file name whose file control block is set up at location CS:5C.

In the third case, the Name command provides file name parameters for the program currently in memory. Whatever file control block was set at CS:5C is replaced by that of the first such parameter. If a second file parameter is specified, its file control block is set up at location CS:6C. Only two file control blocks are set up, although additional file name parameters may be included if required. All the specified — including any delimiters and switches that may have been typed — are placed in a save area at CS:81, with CS:80 containing a character count. Parameters specified in this way are analogous to file names specified in the argument list to the DEBUG command.

Examples 1 Enter **DEBUG <CR>**
Nb:file.com <CR>
L <CR>.

The system enters the DEBUG environment and FILE.COM resident on drive B has its file control block set up at location CS:5C. The Load command subsequently loads this file into memory.

This sequence has the same effect as entering

"DEBUG b:file.com"

2 Enter **Nb:newfile.com <CR>**
W <CR>

The file control block is set up at location CS:5C for the file specifier "b:newfile.com". The subsequent Write command writes the file currently in memory to drive B and names the file "newfile.com".

3 Enter **DEBUG b:file1.com <CR>**
Nfile2.dat,file3.dat <CR>
G <CR>

The DEBUG command loads the file named "file1. com" from drive B to be debugged. The Name command sets up two file control blocks at locations CS:5C and CS:6C for the file specifiers b:file2.dat and b:file3.dat, respectively. These files then become parameters to file1.COM when the subsequent GO command executes file1-.COM. Therefore, the file is executed as if the following command line had been typed:

```
b:file1  file2.dat  file3.dat
```

O (OUTPUT)

Sends a specified byte to an output port.

Syntax **O value,byte**

value the address of the output port. It must be specified in hexadecimal and can be up to 16 bits.

byte a two-digit hexadecimal value to be sent to the specified port.

Example

1 `O1E, 27 <CR>`

2 the byte value 27H is output to the port 1EH.

Q (QUIT)

Terminates the DEBUG program.

Syntax Q

Comments The Quit command terminates the debugger without saving the file you are working on. Control is returned to MS-DOS command mode.

R (REGISTER)

Displays the contents of the registers and flag settings, or displays the contents of a specified register with the option to change that value, or displays the flag settings with the option of reversing any number of those settings.

Syntax

R [register-name pipe: F]

register-name

any valid register name whose contents are to be examined and optionally changed. This may be one of:

AX	DX	SI	ES	IP
BX	SP	DI	SS	PC
CX	BP	DS	CS	

Note: IP and PC both refer to the Instruction Pointer.

F

the flag settings are to be displayed and optionally changed.

Comments

If you enter R without parameters, then the contents of all registers are displayed along with the flag settings and the next instruction to be executed. For Example:

```
AX=058D BX=0000 CS=0000 DX=0000
SP=FFF0 BP=0000 SI=0000 DI=0000 DS=058D
ES=058D CS=058D IP=013B
NV UP EI PL NZ NA PO NC
058D:013B 83D8      MOV DS,AX
```

If you enter R with a register name, then DEBUG displays the contents of that register. The command then waits for you to do one of the following:

— press <CR> to terminate the Register command without changing the value of the displayed register.

— change the value of the register by entering the four-digit hexadecimal value, then terminate the Register command by entering <CR>.

The valid flag values are shown in the following table:

Flag Name	Set	Clear
Overflow	OV (yes)	NV (no)
Direction	DN (decrement)	UP (increment)
Interrupt	EI (enabled)	DI (disabled)
Sign	NG (negative)	PL (plus)
Zero	ZR (yes)	NZ (no)
Auxiliary Carry	AC (yes)	NA (no)
Parity	PE (even)	PO (odd)
Carry	CY (yes)	NC (no)

If you enter RF, then the current flag settings are displayed. You can then either

— press <CR> to terminate the Register command without changing the flag values, or

— change the setting of one or more flags by entering the alternate value of the appropriate flags. The new values may be entered in any order, with or without delimiters.

Example

1. Enter **R <CR>**.
2. DEBUG displays the contents of all registers, flag settings and the next instruction to be executed.
3. Enter **RIP <CR>**.
4. DEBUG displays the contents of the Instruction Pointer. For example:

   ```
   IP 0139
   :—
   ```

5. Enter **0138 <CR>**.
6. the contents of the Instruction Pointer are changed to 0138.
7. Enter **RF <CR>**.
8. DEBUG displays the flag settings. For example:

   ```
   NV UP EI PL NZ NA PO NC -
   ```

9. Enter **PE ZR DI NG <CR>**.
10. The Parity flag is set to even (PE), the Zero flag is set (ZR), the Interrupt flag is cleared (DI), and the Sign flag is set (NG).
11. Enter **RF <CR>**.
12. DEBUG displays the new state of the flags

    ```
    NV UP DI NG ZR NA PE NC-
    ```

S (SEARCH)

Searches a specified range for a list of bytes.

Syntax **S range,list**

range the range of addresses within which the search is to be made. If you only enter the offset, the segment indicated by the DS register is assumed.

list the list of one or more bytes to be searched for. Bytes in the list must be separated by a space or a comma.

Comments For each occurrence of the list of bytes within the specified range, DEBUG returns the address of the first byte. If no address is returned, no match was found.

Example **1** Enter **S100L100,20** <CR> or
S100,1FF,20 <CR>.

2 DEBUG displays the address of every occurrence of byte value 20 in the address range 100 to 1FF, inclusive, for example:

```
058D: 010C
058D: 0110
058D: 0115
058D: 0118
058D: 0120
058D: 0128
058D: 01CE
```

T (TRACE)

Executes one or more instructions and displays the register contents, flag settings and the next instruction to be executed.

Syntax **T [=address][,value]**

= address DEBUG is to commence execution at this address.

value the number of instructions to be executed.

Comments If the =address parameter is not specified, execution begins at CS:IP.

If the value parameter is not specified, only one instruction is executed.

The display generated is of the same format as that of the Register command (without parameters).

Example

1. Enter **T = 200,5 <CR>**.
2. Five instructions, starting with the one at location CS:200, are executed, and the register and flag values following each instruction are displayed along with the next instruction to be executed.
3. Enter **T <CR>**.
4. The instruction pointed to by CS:IP is executed and the register and flag contents are displayed along with the next instruction to be executed.

U (UNASSEMBLE)

Disassembles strings of bytes in memory and displays them as assembler-like statements along with their corresponding addresses.

Syntax

U [range]
or
U [address]

range the range of addresses whose byte values are to be disassembled. If you do not specify the segment, then the segment indicated by the CS register is assumed.

address the start of a 32 byte area of memory to be disassembled. If you only enter an offset, then the segment indicated by the CS register is assumed.

Comments

- If neither the range nor address parameter is specified, then 32 bytes are disassembled starting at location CS:IP. If the Unassemble command is given more than once, each subsequent invocation starts at the address following the last disassembled location.
- The number of bytes disassembled may be slightly more than the number you specified. This is because instructions are not always the same length and the final address in a range will not always contain the last byte of an instruction.
- The first address of a range, or the address parameter, must always refer to the first byte of an 8086 instruction, otherwise results are unpredictable.

Example

1 Enter **U058D:204L8 <CR>**.

2 Eight bytes starting at location 058D:204 are disassembled and the result displayed:

```
058D:0204 8D16DFOD LEA  DX,[ODDF]
058D:0208 42       INC  DX
058D:0209 03DO     ADD  DX,AX
058D:020B 8916E50B MOV  [OBE5],DX
```

W (WRITE)

Writes the file being debugged to disk.

Syntax **W [address[,drive,block,count]**

address the start address of the code in memory that is to be written to disk. If you enter only an offset, then the segment indicated in the CS register is assumed.

drive the drive containing the specified blocks to which code in memory is to be written. For drive A you must enter 0, for drive B you must enter 1, etc.

block the block number on disk that is the first of a contiguous range of blocks to be overwritten with code from memory.

count the number of disk blocks to be overwritten with code from memory.

Comments

- If you enter the WRITE command without parameters, then the file is written to disk starting from memory address CS:100. If you specify the address parameter, then the file in memory, starting from the specified address, is written to disk.
- In either case, before executing the WRITE command, BX:CX must be set to the number of bytes to be written if the count parameter is not included. This value was set up correctly when the file was loaded (either by the Load command or the DEBUG command itself). However, if, since loading the file, you have executed a GO or

TRACE command, then the value of BX:CX will have been changed. Be sure this value is set up correctly.

- When the WRITE command writes a file to disk, it obtains the drive specifier and file name via the file control block set up at CS:5C. If no drive specifier is set up, then the default is assumed. This file control block is set up either by the DEBUG command (for the file you specify as a parameter to DEBUG) or by a subsequent NAME command. If it does not indicate the file specifier you require, you must set up this file control block using the NAME command. Refer to "Memory Maps, Control Blocks, and Diskette Allocation" for further details.

- When the file is written to disk it overwrites the version currently on disk unless the specified file name does not exist, in which case a new file is created.

- If all parameters are specified, then the code in memory is written to the drive specified by the parameter. The data to be written starts at the memory location specified by the address parameter, and is written to the blocks on the disk specified by the block and count parameters. Be extremely careful to correctly specify the blocks, since information stored there previously will be destroyed by this operation.

Examples **1** Enter **W <CR>**.

The file in memory, starting from location CS:100, is written to disk with the file specifier defined by the file control block set up at location CS:5C. The number of bytes written is given by BX:CX.

2 Enter **W200 <CR>**.

The file in memory, starting from location CS:200, is written to disk with the file specifier defined by the file control block set up at location CS:5C. The number of bytes written is given by BX:CX.

3 Enter **W200,1,1F,20 <CR>**.

Blocks 1F through 3F on drive B are overwritten with the data starting at memory location CS:200.

DEBUG ERROR MESSAGES

BF Bad Flag
You attempted to alter a flag, but entered some characters that are not acceptable pairs of flag values. See R (Register) command for the list of acceptable flag entries.

BP Too many Breakpoints
You specified more than ten breakpoints as parameters to the GO command. Reenter the command with ten or fewer breakpoints.

BR Bad Register
You entered the R command with an invalid register name.

DF Double Flag
You entered two values for one flag.

4 8086 Addressing Scheme

- Overview
- The 20-Bit Address
- Aligned and Non-Aligned Words
- Registers and Flags
- Code, Data, and Stack Segments
- Addressing Modes

Overview

The 8086 microprocessor has an extremely flexible addressing scheme. The 8086 uses a 16-bit word, but can address a megabyte of memory. The 8086 supports seven different addressing modes.

To take advantage of the flexibility of the 8086, so that you can write assembly language code and navigate through programs while debugging, study the addressing scheme by carefully reading this chapter.

The 20-Bit Address

The AT&T Personal Computer 6300 utilizes the full address space that is available due to the design of the 8086 microprocessor. The addresses are 20 bits long, so the address space is two to the twentieth power, 1024K, or one megabyte.

The 8086 has a 16-bit word. To convert 16-bit words to a 20-bit address, the 8086 uses "segmented addressing." A 20-bit address is created by using values from two separate registers. Two 16-bit numbers are used.

The binary representation of the first number is considered to have four binary zeroes tacked on to its end. This effectively multiplies the number by 16. This value is known as the segment portion of the address. The segment portion can point to any 16-byte segment of memory in the megabyte address space. However, with four zeroes as its least significant bits, it cannot "zero in" on individual bytes. The segment register's function is just to point to a 16-byte boundary (also known as a paragraph boundary).

Once a segment is located, the other register comes into play. This "offset register" points to the relative part of the address. The 16 bits that comprise the offset register point to an individual byte which is relative to the start of the segment.

The 8086 locates a particular address by:

1. Shifting the segment register to the left by four bits
2. Adding the contents of the offset register

The 20-bit address is conventionally expressed in special notation:

Syntax **<segment register>:<offset register>**

Example 009F:0012

Segment address	009F0
+ Relative address	0012
= Actual Address	00A02

Aligned and Non-Aligned Words

The instructions for the 8086 are made up of from one to six bytes. Instructions can start at either an even or odd address. The 8086 is capable of accessing two bytes of data in memory in a single memory cycle. When the CPU accesses a word (16 bits) located at an even address, it is accessing an "aligned" word. The word is aligned because both bytes are located at the same word address and can be accessed in a single memory cycle.

When the CPU accesses a word starting at an odd address, it is accessing a "non-aligned" word. Since the two bytes comprising the word do not occupy the same word address, two memory cycles are required to read the entire word.

The importance of aligned or non-aligned words is determined by the importance of execution speed in your application. It is good programming practice to store data starting at an even address. If your program accesses or manipulates many word quantities, this will help speed program execution. If you are writing a device driver and instruction cycle times affect the execution of your program, the impact of aligned and non-aligned words should be taken into consideration.

Registers and Flags

General Registers

There are two main groups of general registers used by the 8086: the data group and the pointer and index group. Each register is 16 bits wide.

- The data registers are AX, BX, CX, and DX. Each can be used as a single 16-bit register or as two 8-bit registers. When they are used as two 8-bit registers, they are divided into an upper(H) and lower(L) half and called AH, AL, BH, BL, CH, CL, DH, and DL.

- The pointer and index registers are 16-bit registers. They are named according to their functions: SP (stack pointer), BP (base pointer), SI (source index), and DI (destination index).

Segment Registers

There are four segment registers in the 8086. Each register is 16 bits and their names reflect their use:

- CS — Code Segment
  Always defines the current code segment.
- DS — Data Segment
  Usually defines the current data segment.
- SS — Stack Segment
  Always defines the current stack segment.
- ES — Extra Segment
  Can define an auxiliary data segment.

These registers are used in combination with other registers to form the 20-bit address. Each segment begins on a paragraph (16 byte) boundary. There are four "current" segments at any one time. The contents of each segment register is called the "segment base value". The sections on "Code, Data, and Stack Segments" and "Addressing Modes" give details on how these registers are utilized.

Instruction Pointer

The instruction pointer (IP) is used in conjunction with the Code Segment register to point to the address of the next executable instruction. The IP is also a 16-bit register.

Flags

The 8086 has nine 1-bit status or condition flags that are used to indicate the condition of the result of an arithmetic or logical operation that has just occurred. Some of the assembly language instructions use these flags to conditionally change the execution path of a program.

Flag Definitions

AF Auxiliary Carry Flag
This flag is set (i.e., equal to 1) under two conditions:

During addition there is a carry of the low nybble to the high nybble. (nybble = 4 bits)

During subtraction there is a borrow from the low nybble to the high.

CF Carry Flag
This flag is set when there has been a carry or a borrow to the high-order bit of the (8- or 16-bit) result of an operation.

OF Overflow Flag
When this flag is set, an arithmetic overflow has occurred and a significant digit has been lost.

SF Sign Flag
This flag is set when the high-order bit of the result of an operation is a logical 1. Since negative binary numbers are represented using two's complement notation, SF reflects the sign of the result: 0 indicates a positive number and 1 indicates a negative number.

PF Parity Flag
If this flag is set, the result of the operation has an even number of ones in it. Use this flag to check for data transmission errors.

ZF Zero Flag
This flag is set when the result of an operation is zero.

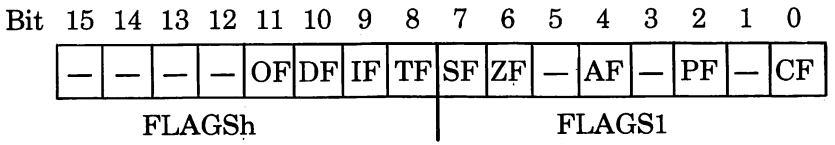
Flag Definitions (Cont'd)

TF Trap Flag
When set, the trap flag puts the system into single-step mode for the purposes of debugging. An internal interrupt is generated after each instruction so that you can inspect your program one instruction at a time.

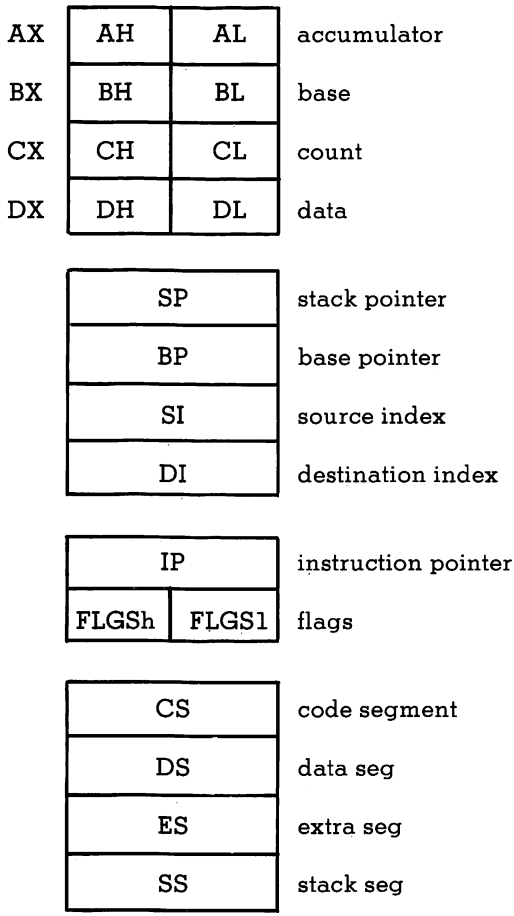
IF Interrupt-enable Flag
If this flag is set, external (maskable) interrupts are recognized by the 8086.

DF Direction Flag
This flag is set and cleared by the STD (Set Direction Flag) and CLD (Clear Direction Flag) instructions. If it has the value 1, SI and DI are decremented during string move operations. If it has the value of 0, SI and DI are incremented during string move operations. This flag is used for the following instructions: MOVS, MOVSB, MOVSW, CMPS, CMPSB, and CMPSW.

The flag register looks like this:



CPU
Registers



Code, Data, and Stack Segments

When you invoke a program, MS-DOS loads all of its segments into memory on paragraph boundaries. The segment registers are set to point to these locations. The data, code, and stack segments aren't necessarily far apart in memory; they may, in fact, overlap. Each segment may be up to 64 KB in length.

Code Segment

Programs are limited to 64K of code, unless they change the value in the CS register. If a program changes the CS register, it may address up to 1024K of code.

The CS register is modified by the FAR CALL and FAR RETurn instructions. Use these instructions to execute code that is located outside the bounds of the current segment.

Data Segment

Most programs use a maximum of 64 KB of memory for data. This includes Pascal and compiled BASIC. Assembly language programs, however, can use additional memory for data by employing the Extra Segment.

Extra Segment

The extra segment may be used in any manner you wish but is often used for transferring large blocks of data quickly in memory or as a storage area for a second stack.

Stack Segment

Stacks are used for temporarily storing register contents and other important values under these conditions:

- Interrupts
- Inter-segment calls
- One program calls another

Addressing Modes

General Comments

The flexible architecture of the 8086 supports many different memory-addressing modes. These can be broken down into six main types of addressing: immediate, register, direct, register indirect, and two kinds of calculated addressing. The following section discusses these modes and concerns the nature of the operand.

Immediate Addressing

In the immediate addressing mode, the operand appears in the instruction. For example,

```
MOV AX,333
```

moves the constant value 333 into the AX register.

Register Addressing

The register addressing mode uses the contents of one of the registers as the operand for the instruction. The instruction can specify that either 8 bits or 16 bits are to be manipulated. For example:

```
MOV AX,BX   ;moves 16 bits from BX to AX

MOV AL,BL   ;moves 8 bits from BL to AL
```

Direct Addressing

The direct addressing mode specifies a location in memory whose contents are used as the operand for the instruction. Example:

```
MOV CX,COUNT
```

This instruction uses the value found in the memory location designated by the symbol COUNT. Unless otherwise specified, COUNT is expected to be somewhere in the Data Segment. To specify that the operand is located in a segment other than the data segment, use the "segment override prefix:"

```
MOV CX,ES:COUNT
```

This syntax specifies that COUNT is located in the Extra Segment.

Register Indirect Addressing

With the register indirect addressing mode, the 16-bit offset address is contained in a base or index register. That is, the offset address resides in the BX, BP, SI, or DI register. Example:

```
MOV AX,[SI]
```

The 16-bit offset contained in the SI register is combined with the data segment register to compute the 20-bit address of the operand to move into register AX. Which segment register is used to compute the address depends on which instruction you are using (i.e., data segment or segment override for MOV, code segment for JMP or CALL, etc.).

Calculated Addressing Modes

The calculated addressing modes are like a combination of register indirect mode and direct addressing mode. There are two calculated addressing modes: single index and double index. In single index addressing, a 16-bit offset from the BX, BP, SI or DI register is added to an offset location in memory specified in the instruction. The combined value of these two items provides the offset into memory from which the operand is fetched. If the BP register is used the offset is from within the stack segment; otherwise the offset is from within the data segment. As always, use of a segment override prefix can change this. Examples:

```
MOV   AX, COUNT[DI]
MOV   AX, RECORD[BP]
```

In double index calculated addressing mode, values from two 16-bit registers are added to an optionally specified location in memory to produce the final offset. Either the BX or BP register is used for one of the register values, and the SI or DI register is used for the other one. If only two registers are given with no memory location, the memory location defaults to 0000 (start of segment). Once again, the default calculation is from the stack segment if the BP register is used; if BX is used the default is from the data segment. Examples:

```
MOV   AX, COUNT[BX+SI]
MOV   AX, RECORD[BP] [DI]
MOV   AX, [BX] [DI]
```

5 Memory Maps Control Blocks Diskette Allocation

- Overview
- The Address Space
- Low Memory Map
- ROM BIOS Data Area
- File Control Blocks
- ASCIIZ Strings
- Handles
- Diskette Layout
- Diskette Directory
- File Allocation Table
- Diskette Formats

Overview

The purpose of this chapter is to enable you to locate items in memory or on diskette for the purposes of programming and debugging.

The first portion of the chapter contains detailed memory maps of the RAM and ROM memory areas. The sections on control blocks deal with program file formats and I/O data structures. The last part of this chapter describes how data is organized on the diskette.

The Address Space

Hex	Decimal	Contents
00000	0K	Interrupt vectors (see detail in low memory map)
04000	16K	DOS software
08000	32K	Language, applications programs and data Note: There is at least 98,000 hex or 608K of address space reserved for user programs and data. To take advantage of the full amount, you must have purchased and installed the physical memory.
		• • •
A0000	640K	Reserved for extended graphics
B0000	704K	Monochrome display buffer — Not used
B8000	736K	Color/graphics display buffer(s)
C8000	800K	Fixed disk adapter's ROM (Optional)
F0000	960K	Reserved for ROM expansion
FC000	1008K	ROM BIOS

Low Memory Map

Hexadecimal addresses are in segment:offset format.

Hex	Decimal	Contents
0:0000	0	Interrupt vectors 0 - 7 8259 interrupt controller vectors (8-F)
0:0040	64	
0:0080	128	BIOS interrupt vectors 10-1F
0:0100	256	MS-DOS interrupt vectors 20-3F
0:0400	1024	Assignable interrupt vectors (40-FF) Note: These vectors may be assigned to non-Intel hardware and software products.
0:0500	1280	ROM BIOS data area (also called BIOS communications area) See map on next page.
0:0600	1536	DOS data area (also called DOS communications area)

ROM BIOS Data Area

Hex addresses are in segment:offset format.

Hex	Decimal	Contents
0:0400	1024	Hardware environment parameters (printer and RS232C device addresses, memory size, etc.)
0:0417	1047	
0:043E	1086	Keyboard buffer and status bytes
0:0449	1097	Floppy and hard disk status bytes
0:0467	1127	Video display area (current mode, color pallette, cursor position, active page numbers, etc.)
0:0471	1137	Data area for option ROM and 8253 timer chip
0:0488	1160	Fixed disk, I/O timeouts, and more keyboard status information
0:0500	1280	RESERVED
		Inter-applications communications area

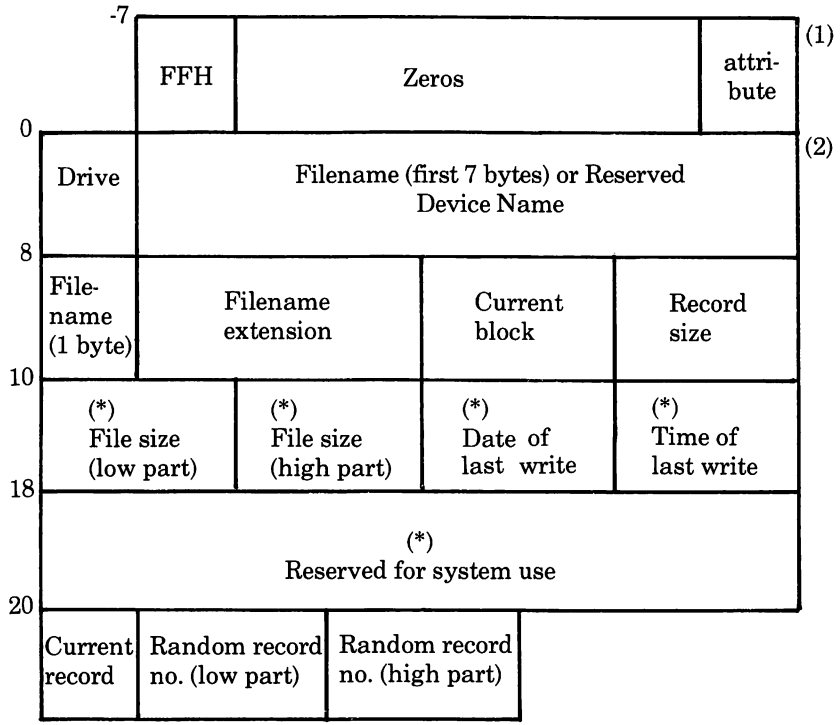
File Control Blocks

FCB Format

The standard File Control Block (FCB) contains 37 bytes of file control information. The extended File Control Block is used to create or search for files in the disk directory that have special attributes. If the extended FCB is used, it adds a 7-byte prefix to the standard FCB.

Any of the DOS functions which employ FCBs may use either an FCB or an extended FCB. (See chapter 7 for a description of each DOS function call.)

If you are using an extended FCB, set the appropriate register to the first byte of the prefix, not to the first byte of the standard FCB, before executing the function call.



(1) FCB extension
(2) Standard FCB

(*) Areas with an asterisk are filled by DOS and must not be modified. Other areas must be filled by the using program.

Offsets are in decimal.

Byte	Function
0	Drive number Before open: 0 — default drive 1 — drive A 2 — drive B etc. After open: 1 — drive A 2 — drive B etc. A 0 is replaced by the actual drive number when the file is opened.
1-8	Filename, left-justified, padded with trailing blanks. If a reserved device name is placed here (e.g., LPT1,) do not include the optional colon.
9-0B	Filename extension, left-justified, with trailing blanks (may be all blanks).
0C-0D	Current block number relative to the beginning of the file (starting at zero). A block is defined as a group of 128 records. This field is set to 0 when the file is opened. This field and the Current Record field (offset 20H) make up the record pointer that is used for sequential reads and writes.
0E-0F	Logical record size in bytes. Automatically set to 80H during open. If this is not correct, set it to the correct value.
10-13	File size in bytes. The first word of the field is the low-order part of the size.

Byte	Function
14-15	Date the file was created or last updated. The bits correspond to the date as follows: < 15 >< 14 > 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 y y y y y y y m m m m d d d d d yy = 0 - 119 (1980 to 2099) mm = 1 - 12 dd = 1 - 31
16-17	Time the file was created or last updated. The bits correspond to the time as follows: < 17 >< 16 > 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 h h h h h m m m m m m s s s s s hh = 0 - 23 mm = 0 - 59 ss = 0 - 30 (number of 2-second increments)
18-1F	Reserved for use by DOS.
20	Current relative record number (0-127) within the current block. Set this field before doing sequential read/write operations; it is not initialized by the Open File function call.
21-24	Relative record number; relative to the beginning of the file, starting with zero.

Byte	Function
	This field is not initialized by the Open File function call and must be set before doing a random read or write. If the record size is less than 64 bytes, both words of the field are used; otherwise, only the first three bytes are used. Note: If you use the FCB at offset 5CH of the Program Segment Prefix, the last byte of the Relative Record field is the first byte of the unformatted parameter area that starts at offset 80H. This is the default Disk Transfer Address.

Extended Control Block

Byte	Function
FCB-7	Contains hex FF to indicate this is extended FCB.
FCB-6 to FCB-2	Reserved
FCB-1	Attribute byte. 02 = Hidden file 04 = System file

ASCIIZ Strings

MS-DOS version 2.11 provides a set of new function calls for file I/O that are easier to use than the "traditional" calls that were used in past versions. These new calls do not utilize file control blocks. To open a diskette file, you simply provide information to identify the file in the form of an ASCIIZ string. DOS returns a numeric value, a "handle", that you use to refer to the file once you have opened it.

The older function calls that require the use of file control blocks and do not utilize ASCIIZ strings and handles are supported in MS-DOS 2.11 to provide upward compatibility. Use the newer function calls whenever possible. See Chapter 7 for details of function calls.

ASCIIZ String Format

An ASCIIZ string, also known as a pathname string, has the following format: an optional drive specifier, followed by a directory path, and where applicable, a filename. The last byte must be binary zeroes. For example:

A:\LEVELA\LEVELB\FILEA

(followed by a byte of zeroes)

Either back slash (\) or forward slash (/) are valid path-separator characters.

Handles

Several of the new function calls that support files or devices use an identifier known as a "handle" (also known as a "token"). When you create or open a file or device with these function calls, a 16-bit value is returned in register AX. Use this handle to refer to the file after it has been opened.

The following handles are pre-defined by DOS for your use. You need not open them before using them:

0000	Standard input device
0001	Standard output device
0002	Standard error output device
0003	Standard auxiliary device
0004	Standard printer device

Note:
See your MS-DOS User's Guide for information on redirecting I/O for the first two handles.

Diskette Layout

The DOS area of the diskette is formatted as follows:

Reserved Area — variable size
First copy of file allocation table — variable size
Second copy of file allocation table — variable size
Root directory — variable size
File data area

Clusters

Space for a file in the data area is not preallocated. The space is allocated one "cluster" at a time. A cluster consists of one or more consecutive sectors; all of the clusters for a file are "chained" together in the File Allocation Table (FAT). On diskettes formatted by MS-DOS 2.11, there are two copies of the FAT kept, for consistency. Should the disk develop a bad sector in the middle of the first FAT, the second is used as a backup.

Diskette Directory

The FORMAT command builds the root directory for all disks. Its location on disk and the maximum number of entries are dependent on the media.

Since directories other than the root directory are regarded as files by MS-DOS, there is no limit to the number of files they may contain.

All directory entries are 32 bytes in length, and are in the following format (byte offsets are in hexadecimal):

Directory Format

0-7 Filename. Eight characters, left aligned and padded, if necessary, with blanks. If this is not currently a file directory entry, the first byte of this field indicates the status as follows:

- 00H The directory entry has never been used. This is used to mark the end of the allocated directory and limit the length of directory searches, for performance reasons.
- 2EH The entry is for a directory (2EH is the ASCII code for the dot '.' character used to represent a directory). If the second byte is also 2EH (i.e., the entry is '..'), then the cluster field contains the cluster number of this directory's parent directory (0000H if the parent directory is the root directory). Otherwise, bytes 01H through 0AH are all spaces (i.e., the entry is '.') and the cluster field contains the cluster number of this directory.
- E5H The file was used, but it has been erased.

Any other character is the first character of a filename.

Directory Format (cont'd)

8-0A Filename extension.

0B File attribute. The attribute byte is mapped as follows (values are in hex):

- 01 File is marked read-only. An attempt to open the file for writing using the Open a File system call (Function Request 3DH) results in an error code being returned. This value can be used along with other values below. Attempts to delete the file with the Delete File system call (13H) or Delete a Directory Entry (41H) will also fail.
- 02 Hidden file. The file is excluded from normal directory searches.
- 04 System file. The file is excluded from normal directory searches.
- 08 The entry contains the volume label in the first 11 bytes. The entry contains no other usable information (except date and time of creation), and may exist only in the root directory.
- 10 The entry defines a sub-directory, and is excluded from normal directory searches.
- 20 Archive bit. The bit is set to "on" whenever the file has been written to and closed. It is used by BACKUP and RESTORE commands for determining whether or not a file has changed since its last backup. The BACKUP command clears this attribute on all files backed up.

Directory Format (cont'd)

The system files (IO.SYS and MSDOS.SYS) are marked as read-only, hidden, and system files. Files can be marked hidden when they are created. Also, the read-only, hidden, system, and archive attributes may be changed through the Change Attributes system call (Function Request 43H).

0C-15 Reserved.

16-17 Time the file was created or last updated. The hour, minutes, and seconds are mapped into two bytes as follows:

Offset 17H

H	H	H	H	H	M	M	M
7	6	5	4	3	2	1	0

Offset 16H

M	M	M	S	S	S	S	S
7	6	5	4	3	2	1	0

H is the number of hours (0-23)
M is the number of minutes (0-59)
S is the number of two-second increments

The time is stored with the least significant bit first.

Directory Format (cont'd)

18-19 Date the file was created or last updated. The year, month, and day are mapped into two bytes as follows:

Offset 19H

Y	Y	Y	Y	Y	Y	Y	M
7	6	5	4	3	2	1	0

Offset 18H

M	M	M	D	D	D	D	D
7	6	5	4	3	2	1	0

Y is the number of years since 1980, 0-119 (1980-2099)
M is the month number 1-12
D is the day of the month 1-31

The date is stored with its least significant byte first.

1A-1B The cluster number of the first cluster in the file. The first cluster for data space on all disks is cluster 002.

The cluster number is stored with the least significant byte first.

Note:

Refer to "How to Use the File Allocation Table" for details on converting cluster numbers to logical sector numbers.

1C-1F File size in bytes. The first word of this four-byte field is the low-order part of the size.

File Allocation Table (FAT)

The following information is included primarily for system programmers who are writing installable device drivers. This section explains how MS-DOS uses the File Allocation Table to convert the clusters of a file to logical sector numbers. The driver program is then responsible for locating the logical sector on disk. If you are writing a system utility, use the MS-DOS file management function calls for accessing files; programs that access the FAT directly are not guaranteed to be upwardly-compatible with future releases of MS-DOS.

FAT Entries

The File Allocation Table is an array of 12-bit entries (1.5 bytes) for each cluster on the disk.

- The first two FAT entries map a portion of the directory; these FAT entries indicate the size and format of the disk.
- The second and third bytes currently always contain FFH.
- The third FAT entry, which starts at byte offset 4, begins the mapping of the data area (cluster 002). Files in the data area are not always written sequentially on the disk. The data area is allocated one cluster at a time, skipping over clusters already allocated.The first free cluster found will be the next cluster allocated, regardless of its physical location on the disk. This permits the most efficient utilization of disk space because clusters made available by erasing files can be allocated for new files.

Each FAT entry contains three hexadecimal characters:

- 000 — If the cluster is unused and available.
- FF7 — The cluster has a bad sector in it. MS-DOS will not allocate such a cluster. CHKDSK counts the number of bad clusters for its report. These bad clusters are not part of any allocation chain.
- FF8-FFF — Indicates the last cluster of a file.
- XXX — The cluster number of the next cluster in the file. The cluster number of the first cluster in the file is kept in the file's directory entry.

The File Allocation Table always begins on the first sector after the reserved sectors. If the FAT is larger than one sector, the sectors are contiguous. Two copies of the FAT are usually written for data integrity. The FAT is read into one of the MS-DOS buffers as needed (open, read, write, etc.). For performance reasons, this buffer is given a high priority so that it stays in memory as long as possible.

How to Use the File Allocation Table

Use the directory entry to find the starting cluster of the file. Next, to locate each subsequent cluster of the file:

1. Multiply the cluster number just used by 1.5 (each FAT entry is 1.5 bytes long).
2. The whole part of the product is an offset into the FAT, pointing to the entry that maps the cluster just used. That entry contains the cluster number of the next cluster of the file.
3. Use a MOV instruction to move the word at the calculated FAT offset into a register.
4. If the last cluster used was an even number, keep the low-order 12 bits of the register by ANDing it with FFF; otherwise, keep the high-order 12 bits by shifting the register right 4 bits with a SHR instruction.
5. If the resultant 12 bits are FF8H-FFFH, the file contains no more clusters. Otherwise, the 12 bits contain the cluster number of the next cluster in the file.

To convert the cluster to a logical sector number (relative sector, such as that used by Interrupts 25H and 26H and by DEBUG):

1. Subtract 2 from the cluster number.
2. Multiply the result by the number of sectors per cluster.
3. Add to this result the logical sector number of the beginning of the data area.

Diskette Formats

On an MS-DOS disk, the clusters are arranged on diskette to minimize head movement for multi-sided media. All of the space on a track (or cylinder) is allocated before moving on to the next track. This is accomplished by using the sequential sectors on the lowest-numbered head, then all the sectors on the next head, and so on until all sectors on all heads of the track are used. The next sector to be used will be sector 1 on head 0 of the next track.

The first byte of the FAT, called the Media Descriptor Byte, can sometimes be used to determine the format of the disk. The following formats have been defined for the AT&T Personal Computer 6300, based on values of the first byte of the FAT.

MS-DOS Standard Diskette Formats

No. sides	1	1	2	2
Tracks/side	40	40	40	40
Bytes/sector	512	512	512	512
Sectors/track	8	9	8	9
Sectors/cluster	1	1	2	2
Reserved sectors	1	1	1	1
No. FATs	2	2	2	2
Root directory entries	64	64	112	112
No. sectors	320	360	640	720
Media Descriptor Byte	FE	FC	FF	FD
Sectors for 1 FAT	1	2	1	2

6 Program Structure and Loading

- Overview
- Pros and Cons for Selecting a Program File Format
- EXE2BIN
- File Header Format
- Relocation Process for .EXE Files
- Program Segment Prefix
- Program Loading Process

Overview

This chapter describes the MS-DOS program file formats and procedures for loading them into memory. MS-DOS supports two main program file formats: .EXE and .COM.

The .EXE format is the more flexible program type. An .EXE file is limited in size only by the amount of user memory installed in your system.

Programs linked by MS-LINK are output in .EXE format. .EXE files can be executed either by COMMAND.COM or by an EXEC system call (Function Request 4BH) in your program.

A .COM program file cannot exceed 64K bytes in length. However, because it does not have the same lengthy header that an .EXE file does, a .COM file takes up less diskette storage space and loads into memory more quickly than an .EXE file.

After assembling and linking your program, it must be converted to .COM format. The easiest way to do this is with the EXE2BIN utility provided on your MS-DOS Supplemental Programs diskette.

Pros and Cons for Selecting a Program File Format

This section is concerned with the pros and cons of selecting between a .EXE program format and the .COM program type.

PROS for .EXE

- Can be larger than 64 K
- Can cross segment boundaries
- Can run .EXE immediately after linking, i.e., you need not take the extra step of running EXE2BIN
- Can declare a stack segment in the assembly program

CONS for .EXE

- Disk file has large "header" containing relocation information. .EXE therefore takes more space on disk and takes longer to load into memory at execution time.

PRO for .COM

- .COM files are smaller and faster loading because .COM does not have a file header containing relocation information.

CONS for .COM

- .COM files can be no larger than one 64K segment.

- .COM is segment-relocatable; the segment can be relocated at run time. However, all of the addresses in the program must be relative to the same segment address.

EXE2BIN

EXE2BIN

EXE2BIN is an executable program available on your MS-DOS system diskette. It converts programs that are in .EXE format (as they are after having been linked) into the .COM format.

EXE2BIN can generate two types of .COM files: relocatable and non-relocatable.

Syntax

EXE2BIN <input filename> <output file name>

Both file names are in the form:

[d:][path][filename[.ext]]

Example

EXE2BIN B:PROG.EXE B:PROG.COM

Discussion

In specifying the input file, everything except the file name is optional. If you do not specify a drive, the default is used. If you do not specify a path, the default path is used. If you do not specify an extension, the default is .EXE. The input file is converted to .COM file format (memory image of the program) and placed in the output file.

You are not required to enter any part of the output file specification. If you do not specify a drive, the drive of the input file will be used. If you do not specify an output path or filename, the input path or filename will be used. If you do not specify a filename extension in the output filename, the new file will be given an extension of .BIN.

The input file must be in valid .EXE format produced by the linker. The resident, or actual code and data part of the file must be less than 64K. There must be no STACK segment.

Two kinds of conversions are possible, depending on the initial CS:IP (Code Segment: Instruction Pointer) specified in the .EXE file:

1. If CS:IP is specified as 0000:100H, it is assumed that the file is to be run as a .COM file with the location pointer set at 100H by the assembler statement ORG; the first 100H bytes of the file are deleted. No segment address fixups (that is, instructions that contain a reference to an absolute segment address) are allowed, as .COM files must be segment relocatable. Once the conversion is complete, rename the resulting file with a .COM extension. The command processor can load and execute the program in the same way as the .COM programs supplied on your MS-DOS diskettes.

2. If CS:IP is not specified in the .EXE file, a pure binary conversion is assumed. If segment fixups are necessary (i.e., the program contains instructions requiring a segment address), you are prompted for the fixup value. This value is the absolute segment at which the program is to be loaded. The resulting program is usable only when loaded at the absolute memory address specified by your application. The command processor is not capable of properly loading the program. This is the case when writing a .BIN program to use in an application such as a device driver that is always loaded at the same absolute address.

EXE2BIN Messages

Amount read less than size in header

The program portion of the file was smaller than indicated in the file's header. You should reassemble and relink your program.

File cannot be converted

CS:IP does not meet either of the criteria specified above, or it meets the .COM file criterion but has segment fixups. This message is also displayed if the file is not a valid executable file.

File creation error

EXE2BIN cannot create the output file. Run CHKDSK to determine if the directory is full or if some other condition caused the error.

File not found

The file is not on the diskette specified.

Fixups needed - base segment (hex):

The source (.EXE) file contained information indicating that a load segment is required for the file. Specify the absolute segment address at which the finished module is to be located.

Insufficient disk space

There is not enough disk space to create a new file.

Insufficient memory

There is not enough memory to run EXE2BIN.

WARNING - Read error in EXE file

Amount read less than size in header. This is a warning message only. However, it is usually a good idea to reassemble and relink your source program when this message appears.

File Header Format

The .EXE files produced by MS-LINK consist of two parts:

- Control and relocation information
- The load module

The control and relocation information is at the beginning of the file in an area called the header. The load module immediately follows the header.

Note:
.COM files do not have file headers.

The header is formatted as follows (offsets are in hexadecimal):

Offset	Contents
00-01	Must contain 4DH, 5AH.
02-03	Number of bytes contained in last page; used for reading overlays.
04-05	Size of the file in 512-byte pages, including the header.
06-07	Number of relocation entries in table.
08-09	Size of the header in 16-byte paragraphs. This is used to locate the beginning of the load module in the file.
0A-0B	Minimum number of 16-byte paragraphs required above the end of the loaded program (minalloc).
0C-0D	Maximum number of 16-byte paragraphs required above the end of the loaded program (maxalloc). If both minalloc and maxalloc are 0, then the program will be loaded as high as possible.

0E-0F	Initial value to be loaded into stack segment before starting program execution. This must be adjusted by relocation.
10-11	Value to be loaded into the SP register before starting program execution.
12-13	Negative sum of all the words in the file.
14-15	Initial value to be loaded into the IP register before starting program execution.
16-17	Initial value to be loaded into the CS register before starting program execution. This must be adjusted by relocation.
18-19	Relative byte offset from beginning of run file to relocation table.
1A-1B	The number of the overlays generated by MS-LINK.

This is followed by the relocation table. The table consists of a variable number of relocation items. Each relocation item contains two fields: a two-byte offset value, followed by a two-byte segment value. These two fields contain the offset into the load module of a word which requires modification before the module is given control.

Relocation Process for .EXE Files

The following steps describe the relocation process:

1. The formatted part of the header is read into memory. Its size is 1BH.

2. A portion of memory is allocated depending on the size of the load module and the allocation numbers (0A-0B and 0C-0D). MS-DOS attempts to allocate FFFFH paragraphs. This will always fail, returning the size of the largest free block. If this block is smaller than minalloc and loadsize, then there will be a no memory error. If this block is larger than maxalloc and loadsize, MS-DOS will allocate (maxalloc + loadsize). Otherwise, MS-DOS will allocate the largest free block of memory.

3. A Program Segment Prefix is built in the lowest part of the allocated memory.

4. The load module size is calculated by subtracting the header size from the file size. Offsets 04-05 and 08-09 can be used for this calculation. The actual size is downward-adjusted based on the contents of offsets 02-03. Based on the setting of the high/low loader switch, an appropriate segment is determined at which to load the load module. This segment is called the start segment.

5 The load module is read into memory beginning with the start segment.

6 The relocation table items are read into a work area.

7 Each relocation table item segment value is added to the start segment value. This calculated segment, plus the relocation item offset value, points to a word in the load module to which is added the start segment value. The result is placed back into the word in the load module.

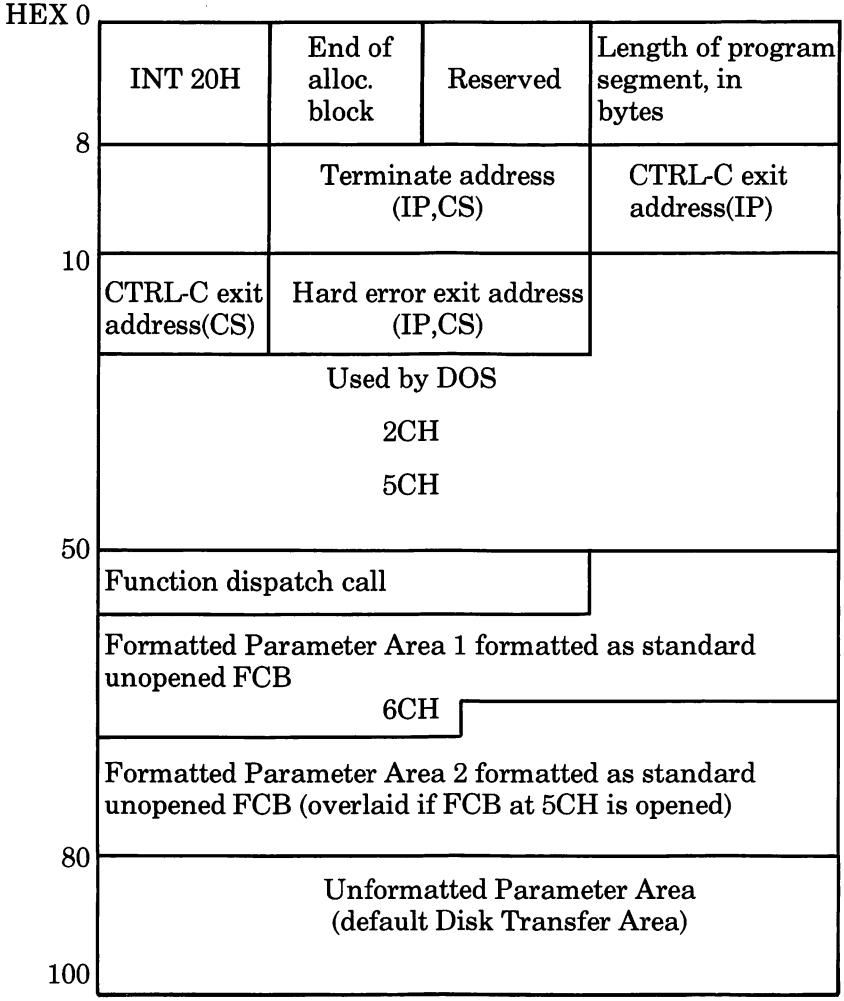
8 Once all relocation items have been processed, the SS and SP registers are set from the values in the header. Then, the start segment value is added to SS. The ES and DS registers are set to the segment address of the Program Segment Prefix. The start segment value is added to the header CS register value. The result, along with the header IP value, is the initial CS:IP to transfer to before starting execution of the program.

Program Segment Prefix

Unless you specify otherwise when linking your program, DOS loads your program in the lowest memory address available, immediately following the DOS code. This occurs whether the program loads as a result of your entering its name at the DOS prompt or through your use of the EXEC (4BH) function call. The area into which your program is loaded is called the Program Segment.

DOS requires control information for each running program: it builds a Program Segment Prefix and places it at offset 0 within the program segment. The Program Segment Prefix is hex 100 bytes long, so your program is loaded at relative address 100H.

PSP Format


HEX 0
INT 20H
End of alloc. block
Reserved
Length of program segment, in bytes
8
Terminate address (IP,CS)
CTRL-C exit address(IP)
10
CTRL-C exit address(CS)
Hard error exit address (IP,CS)
Used by DOS
2CH
5CH
50
Function dispatch call
Formatted Parameter Area 1 formatted as standard unopened FCB
6CH
Formatted Parameter Area 2 formatted as standard unopened FCB (overlaid if FCB at 5CH is opened)
80
Unformatted Parameter Area (default Disk Transfer Area)
100

HEX OFFSET	CONTENTS	
0	Return address used by interrupt hex 20	
2	Segment address of allocatable memory following this program (If this program calls a memory management function to get more memory, this is its starting address.)	
4	Reserved	
6	Number of bytes in this program segment (2 byte value)	
8	Not used	
A	Terminate address : IP	
C	Terminate address : CS	
E	Ctrl break exit : IP	
10	Ctrl break exit : CS	
12	Critical error exit : IP	
14	Critical error exit : CS	
2C	Segment address of the environment	USED by DOS
50	Code to call function dispatcher for DOS (INT 21H) interrupts	
5C	Formatted parameter area 1: formatted as standard, unopened FCB	
6C	Formatted parameter area 2: formatted as standard, unopened FCB	
80	Count of argument characters that follow the command name.	These comprise the default DTA: Disk Transfer Area(80H - FFH)
81	The argument characters themselves.	

Program Loading Process

PSP Conditions upon Program Initiation

When a program receives control, the following conditions are in effect:

- The segment address of the passed environment is contained at offset 2CH in the Program Segment Prefix. The environment is a series of ASCII strings (totaling less than 32K) in the form:

  NAME = **parameter**

  Each string is terminated by a byte of zeros, and the set of strings is terminated by another byte of zeros. The environment built by the command processor contains at least a COMSPEC= string (the parameters on COMSPEC define the path used by MS-DOS to locate COMMAND.COM on disk). The last PATH and PROMPT commands issued will also be in the environment, along with any environment strings defined with the MS-DOS SET command.

  The environment that is passed is a copy of the invoking process environment. If your application uses a "keep process" concept, be aware that the copy of the environment passed to you is static. That is, it will not change even if subsequent SET, PATH, or PROMPT commands are issued.

- Offset 50H in the Program Segment Prefix contains code to call the MS-DOS function dispatcher. After correctly loading the registers, a program can issue a far call to offset 50H to invoke an MS-DOS function, rather than issuing an Interrupt 21H. Since this is a call and not an interrupt, MS-DOS may place any code appropriate to making a system call at this position. This makes the process of calling the system portable.

- The Disk Transfer Address (DTA) is set to 80H (default DTA in the Program Segment Prefix).

- File control blocks at 5CH and 6CH are formatted from the first two parameters typed when the command was entered. If either parameter contains a pathname, then the corresponding FCB contains only the valid drive number. The filename field will not be valid.

- An unformatted parameter area at 81H contains all the characters typed after the command (including leading and imbedded delimiters), with the byte at 80H set to the number of characters. If the < or > parameters were typed on the command line, they (and the filenames associated with them) do not appear in this area or in the character count; redirection of standard input and output is transparent to applications.

- Offset 6 (one word) contains the number of bytes available in the segment.

- Register AX indicates whether or not the drive specifiers (entered with the first two parameters) are valid, as follows:

  AL=FF if the first parameter contained an invalid drive specifier (otherwise **AL=00**)

  AH=FF if the second parameter contained an invalid drive specifier (otherwise **AH=00**)

- Offset 2 (one word) contains the segment address of the first byte of unavailable memory. Programs must not modify addresses beyond this point unless they were obtained by allocating memory via the Allocate Memory system call (Function Request 48H).

Initial Conditions for .EXE Programs

- DS and ES registers are set to point to the Program Segment Prefix.
- CS, IP, SS, and SP registers are set to the values passed by MS-LINK.

Initial Conditions for .COM Programs

- All four segment registers contain the segment address of the initial allocation block that starts with the Program Segment Prefix control block.
- The Instruction Pointer (IP) is set to 100H.
- The Stack Pointer register is set to the end of the program's segment. The segment size at offset 6 is reduced by 100H to allow for a stack of that size.
- A word of zeros is placed on top of the stack. This allows your program to exit to COMMAND.COM by doing a RET instruction last. Make sure, however, to maintain your stack and code segments.

Other Uses of the Program Segment Prefix

In MS-DOS versions prior to 2.0, the PSP contained the mechanism for program termination. One of these four techniques had to be used to terminate your programs:

1. A long jump to offset 0 in the Program Segment Prefix.

2. By issuing an INT 20H with CS:0 pointing at the PSP.

3. By issuing an INT 21H with register AH = 0 and CS:0 pointing at the PSP.

4. By a long call to location 50H in the Program Segment Prefix with AH = 0 and CS:0 pointing at the PSP.

   It is the responsibility of all programs to ensure that the CS register contains the segment address of the Program Segment Prefix when terminating via any of these methods.

   However, with the 2.0 Terminate a Process system call (Function Request 4CH), the CS register need not point to the Program Segment Prefix. For this reason, Function Request 4CH is the preferred method. It may be invoked by loading the AH register with 4CH and issuing an INT 21H (or a long call to offset 50H in the Program Segment Prefix).

7 System Calls

- Quick Reference: Functions and Interrupts
- Overview
- Programming Considerations
- Interrupts
- Functions
- System Call Descriptions

Functions

Number	Function Name	Number	Function Name
00H	Terminate Program	30H	Get DOS Version Number
01H	Read Keyboard and Echo	31H	Keep Process
02H	Display Character	33H	<CTRL C> Check
03H	Auxiliary Input	35H	Get Interrupt Vector
04H	Auxiliary Output	36H	Get Disk Free Space
05H	Print Character	38H	Return Country-Dependent Info.
06H	Direct Console I/O	39H	Create Sub-Directory
07H	Direct Console Input	3AH	Remove a Directory
08H	Read Keyboard	3BH	Change the Current Directory
09H	Display String	3CH	Create a File
0AH	Buffered Keyboard Input	3DH	Open a File Handle
0BH	Check Keyboard Status	3EH	Close a File Handle
0CH	Flush Buffer, Read Keyboard	3FH	Read From File/Device
0DH	Disk Reset	40H	Write to a File/Device
0EH	Select Disk	41H	Delete a Directory Entry
0FH	Open File	42H	Move a File Pointer
10H	Close File	43H	Change Attributes
11H	Search for First Entry	44H	I/O Control for Devices
12H	Search for Next Entry	45H	Duplicate a File Handle
13H	Delete File	46H	Force a Duplicate of a Handle
14H	Sequential Read	47H	Return Name of Current Directory
15H	Sequential Write	48H	Allocate Memory
16H	Create File	49H	Free Allocated Memory
17H	Rename File	4AH	Modify Allocated Memory Blocks
19H	Current Disk	4BH	Load and Execute a Program (EXEC)
1AH	Set Disk Transfer Address	4CH	Terminate a Process
21H	Random Read	4DH	Retrieve the Return Code of a Child
22H	Random Write	4EH	Find Match File
23H	File Size	4FH	Step Through a Directory Matching Files
24H	Set Relative Record	54H	Return Current Setting of Verify
25H	Set Vector	56H	Move a Directory Entry
27H	Random Block Read	57H	Get/Set Date/Time of File
28H	Random Block Write		
29H	Parse File Name		
2AH	Get Date		
2BH	Set Date		
2CH	Get Time		
2DH	Set Time		
2EH	Set/Reset Verify Flag		
2FH	Get Disk Transfer Address		

Function Name	Number	Function Name	Number
Allocate Memory	48H	Modify Allocated Memory Blocks	4AH
Auxiliary Input	03H	Move a Directory Entry	56H
Auxiliary Output	04H	Move a File Pointer	42H
Buffered Keyboard Input	0AH	Open a File Handle	3DH
Change Attributes	43H	Open File	0FH
Change the Current Directory	3BH	Parse File Name	29H
Check Keyboard Status	0BH	Print Character	05H
Close a File Handle	3EH	Random Block Read	27H
Close File	10H	Random Block Write	28H
CTRL C Check	33H	Random Read	21H
Create a File	3CH	Random Write	22H
Create File	16H	Read From File/Device	3FH
Create Sub-Directory	39H	Read Keyboard	08H
Current Disk	19H	Read Keyboard and Echo	01H
Delete a Directory Entry	41H	Remove a Directory	3AH
Delete File	13H	Rename File	17H
Direct Console Input	07H	Retrieve the Return Code of a Child	4DH
Direct Console I/O	06H	Return Current Setting of Verify	54H
Disk Reset	0DH	Return Country-Dependent Info.	38H
Display Character	02H	Return Name of Current Directory	47H
Display String	09H	Search for First Entry	11H
Duplicate a File Handle	45H	Search for Next Entry	12H
File Size	23H	Select Disk	0EH
Find Match File	4EH	Sequential Read	14H
Flush Buffer, Read Keyboard	0CH	Sequential Write	15H
Force a Duplicate of a Handle	46H	Set Date	2BH
Free Allocated Memory	49H	Set Disk Transfer Address	1AH
Get Date	2AH	Set Relative Record	24H
Get Disk Free Space	36H	Set Time	2DH
Get Disk Transfer Address	2FH	Set Vector	25H
Get DOS Version Number	30H	Set/Reset Verify Flag	2EH
Get Interrupt Vector	35H	Step Through a Directory Matching	4FH
Get Time	2CH	Terminate a Process	4CH
Get/Set Date/Time of File	57H	Terminate Program	00H
I/O Control for Devices	44H	Write to a File/Device	40H
Keep Process	31H		
Load and Execute a Program (EXEC)	4BH		

Interrupts

Interrupts (in Numerical Order)

Interrupt (Hex)	Interrupt (Decimal)	Description
20H	32	Program Terminate
21H	33	Function Request
22H	34	Terminate Address
23H	35	<CTRL C> Exit Address
24H	36	Fatal Error Abort Address
25H	37	Absolute Disk Read
26H	38	Absolute Disk Write
27H	39	Terminate But Stay Resident
28-40H	40-64	RESERVED — DO NOT USE

Interrupts in Alphabetical Order

Description	Interrupt in Hex	Interrupt in Dec
Absolute Disk Read	25H	37
Absolute Disk Write	26H	38
<CTRL C> Exit Address	23H	35
Fatal Error Abort Address	24H	36
Function Request	21H	33
Program Terminate	20H	32
RESERVED — DO NOT USE	28-40H	40-64
Terminate Address	22H	34
Terminate But Stay Resident	27H	39

Overview

System Calls are procedures used to interface with I/O or to manage memory. They can be accessed from utility programs written in assembly language, and from some high level languages. Their use frees the programmer from having to perform primitive functions, and makes it easier to write machine-independent programs.

MS-DOS provides two types of system calls: interrupts and function requests. This chapter describes the environments from which these routines can be called, how to call them, and the processing performed by each.

Programming Considerations

System calls can be invoked from Assembly Language, from GW BASIC, or from high-level languages like PASCAL and FORTRAN. This section describes the techniques for invoking calls and for returning control to MS-DOS.

Calling from Assembly Language

The system calls can be invoked from Assembly Language simply by moving any required data into registers and issuing an interrupt. Some of the calls destroy registers, so you may have to save registers before using a system call.

Calling from GW BASIC

The BLOAD and BSAVE commands are used for loading and saving machine language programs. These are then called, using the CALL statement.

The USR function calls an indicated machine language subroutine. The starting address of the subroutine must first be specified in a DEF USR statement.

Interrupts

MS-DOS reserves interrupts 20H through 3FH for its own use. The table of interrupt routine addresses (vectors) is maintained in locations 80H-FCH. User programs should only issue Interrupts 20H, 21H, 25H, 26H, and 27H. (Functions Requests 4CH and 31H are the preferred method for Interrupts 20H and 27H for versions of MS-DOS that are 2.0 and higher.

Interrupts 22H, 23H, and 24H are not interrupts that can be issued by user programs; they are simply locations where a segment and offset address are stored. For a discussion, see the section on Address Interrupts in this chapter.

Functions

Requirements Most of the MS-DOS function calls require input to be passed to them in registers. After setting the proper register values, the function may be invoked in one of the following ways:

- Place the function number in AH and execute a long call to offset 50H in your Program Segment Prefix. Note that programs using this method will not operate correctly on versions of MS-DOS that are lower than 2.0.

- Place the function number in AH and issue Interrupt 21H. All of the examples in this chapter use this method.

- An additional method exists for programs that were written with different calling conventions. This method should be avoided for all new programs. The function number is placed in the CL register and other registers are set according to the function specification. Then, an intrasegment call is made to location 5 in the current code segment. That location contains a long call to the MS-DOS function dispatcher. Register AX is always destroyed if this method is used; otherwise, it is the same as normal function calls. Note that this method is valid only for Function Requests 00H through 024H.

This chapter provides the following type of information for each DOS interrupt and function call:

- a description of the register contents required before the system call
- a description of the register contents after the system call
- a description of the processing performed
- an example of its use.

Registers

When MS-DOS takes control after a function call, it switches to an internal stack. Registers not used to return information (except AX) are preserved. The calling program's stack must be large enough to accommodate the interrupt system — at least 128 bytes in addition to other needs.

Note

The macro definitions and extended example for MS-DOS system calls 00H through 2EH can be found at the end of this chapter.

System Call Descriptions

Interrupts

The following are not true interrupts but rather storage locations for a segment and offset address:

- Terminate Address (Interrupt 22H)
- **CTRL C** Exit Address (Interrupt 23H)
- Fatal Error Abort Address (Interrupt 24H)

The interrupts are issued by MS-DOS under the specified circumstance. You can change any of these addresses with Function Request 25H (Set Vector) if you prefer to write your own interrupt handlers.

Programming Examples

A macro is defined for most system calls, then used in some examples. In addition, a few other macros are defined for use in the examples. The use of macros allows the examples to be more complete programs, rather than isolated uses of the system calls. All macro definitions are listed at the end of the chapter.

The examples are not intended to represent good programming practice. In particular, error checking and good human interface design have been sacrificed to conserve space. You may, however, find the macros a convenient way to include system calls in your assembly language programs.

A detailed description of each system call follows. They are listed in numeric order; the interrupts are described first, then the function requests.

Note

Unless otherwise stated, all numbers in the system call descriptions, both text and code, are in hex.

20H
Program Terminate

Call

CS
Segment address of Program Segment Prefix

Return

None

Remarks

All open file handles are closed and the disk cache is cleaned. The current process is terminated and control returns to the parent process. This interrupt is almost always used in old .COM files for termination.

The CS register must contain the segment address of the Program Segment Prefix before you call this interrupt.

The following exit addresses are restored from the Program Segment Prefix:

Exit Address	Offset
Program Terminate	0AH
<CTRL C>	0EH
Critical Error	12H

All file buffers are flushed to disk.

20H
Program Terminate

Note

Close all files that have changed in length before issuing this interrupt. If a changed file is not closed, its length is not recorded correctly in the directory. See Functions 10H and 3EH for a description of the Close File system calls.

Interrupt 20H is provided for compatibility with versions of MS-DOS prior to 2.0. New programs should use Function Request 4CH, Terminate a Process.

Macro

```
terminate macro
   int   20H
   endm
```

Example

```
;CS must be equal to PSP values given at program
;start
;(ES and DS values)
   INT 20H
;There is no return from this interrupt
```

21H
Function Request

Call

AH
  Function number
Other registers as specified in
  individual function

Return

As specified in individual function

Remarks

The AH register must contain the number of the system function. See the following section on Function Requests, in this chapter, for a description of the MS-DOS system functions.

Note

No macro is defined for this interrupt, because all function request descriptions in this chapter that define a macro include Interrupt 21H.

Example

To call the Get Time function:

```
mov  ah,2CH    ;Get Time is Function 2CH
int  21H       ;THIS INTERRUPT
```

22H
Terminate Address

When a program terminates, control transfers to the address at offset 0AH of the Program Segment Prefix. This address is copied into the Program Segment Prefix, from the Interrupt 22H vector, when the segment is created. Interrupt 22H, then, is just a storage location for an address rather than a true interrupt.

23H
<CTRL C> Exit Address

If the user types **CTRL C** during keyboard input or display output, control transfers to the INT 23H vector in the interrupt table. This address is copied into the Program Segment Prefix, from the Interrupt 23H vector, when the segment is created.

If the **CTRL C** routine preserves all registers, it can end with an IRET instruction (return from interrupt) to continue program execution. When the interrupt occurs, all registers are set to the value they had when the original call to MS-DOS was made. There are no restrictions on what a **CTRL C** handler can do — including MS-DOS function calls — so long as the registers are unchanged if IRET is used.

If Function 09H or 0AH (Display String or Buffered Keyboard Input) is interrupted by **CTRL C**, the three-byte sequence 03H-0DH-0AH (ETX-CR-LF) is sent to the display and the function resumes at the beginning of the next line.

If the program creates a new segment and loads a second program that changes the **CTRL C** address, termination of the second program restores the **CTRL C** address to its value before execution of the second program.

Like INT 22H, this is really not a true interrupt, but a storage location.

24H
Fatal Error Abort Address

Call

If a fatal disk error occurs during execution of one of the disk I/O function calls, control transfers to the INT 24H vector in the vector table. This address is copied into the Program Segment Prefix, from the Interrupt 24H vector, when the segment is created.

Return

BP:SI contains the address of a Device Header Control Block from which additional information can be retrieved.

Note

Interrupt 24H is not issued if the failure occurs during execution of Interrupt 25H (Absolute Disk Read) or Interrupt 26H (Absolute Disk Write). These errors are usually handled by the MS-DOS error routine in COMMAND.COM that retries the disk operation, then gives the user the choice of aborting, retrying the operation, or ignoring the error. The following topics give you the information you need about interpreting the error codes, managing the registers and stack, and controlling the system's response to the error in order to write your own error-handling routines.

Error Codes

When an error-handling program gains control from Interrupt 24H, the AX and DI registers can contain codes that describe the error. If Bit 7 of AH is 1, the error is either a bad image of the File Allocation Table or an error which has occurred on a character device. The device header passed in BP:SI can be examined to determine which case exists. If the attribute byte high order bit indicates a block device, then the error was a bad FAT. Otherwise, the error is on a character device.

The following are error codes for Interrupt 24H:

Error Code	Description
0	Attempt to write on write-protected disk
1	Unknown unit
2	Drive not ready
3	Unknown command
4	Data error
5	Bad request structure length
6	Seek error
7	Unknown media type
8	Sector not found
9	Printer out of paper
A	Write fault
B	Read fault
C	General failure

The user stack will be in effect (the first item described below is at the top of the stack), and will contain the following from top to bottom:

IP	MS-DOS registers from
CS	issuing INT 24H
FLAGS	
AX	User registers at time of original
BX	INT 21H request
CX	
DX	
SI	
DI	
BP	
DS	
ES	
IP	From the original INT 21H
CS	from the user to MS-DOS
FLAGS	

The registers are set such that if an IRET is executed, MS-DOS will respond according to (AL) as follows:

(AL) = 0 ignore the error
= 1 retry the operation
= 2 terminate the program via INT 23H

Note

- Before giving this routine control for disk errors, MS-DOS performs five retries.
- For disk errors, this exit is taken only for errors occurring during an Interrupt 21H. It is not used for errors during Interrupts 25H or 26H.
- This routine is entered in a disabled state.
- The SS,SP,DS,ES,BX,CX, and DX registers must be preserved.
- The interrupt handler should refrain from using MS-DOS function calls. If necessary, it may use calls 01H through 0CH. Use of any other call will destroy the MS-DOS stack and will leave MS-DOS in an unpredictable state.
- The interrupt handler must not change the contents of the device header.
- If the interrupt handler will handle errors rather than returning to MS-DOS, it should restore the application program's registers from the stack, remove all but the last three words on the stack, then issue an IRET. This will return to the program immediately after the INT 21H that experienced the error. Note that if this is done, MS-DOS will be in an unstable state until a function call higher than 0CH is issued.

25H
Absolute Disk Read

Call

AL
  Drive number (0 = A, 1 = B, etc.)
DS:BX
  Disk Transfer Address
CX
  Number of sectors
DX
  Beginning relative sector

Return

Flags
  CF = 0 if successful
    = 1 if not successful
  AL
    Error code if CF = 1

Remarks

This interrupt transfers control to the MS-DOS BIOS. The number of sectors specified in CX is read from the disk to the Disk Transfer Address. Its requirements and processing are identical to Interrupt 26H, except data is read rather than written.

25H
Absolute Disk Read

Note

All registers except the segment registers are destroyed by this call. Be sure to save any registers your program uses before issuing the interrupt.

The system pushes the flags at the time of the call; they are still there upon return. (This is necessary because data is passed back in the flags). Be sure to pop the stack upon return to restore your stack pointer at the point of invocation.

If the disk operation was successful, the Carry Flag (CF) is 0. If the disk operation was not successful, CF is 1 and AL contains the MS-DOS error code (see Interrupt 24H earlier in this section for the codes and their meaning).

Macro

```
abs_disk_read macro disk,buffer,num_sectors, start
   mov   al,disk
   mov   bx,offset buffer
   mov   cx,num_sectors
   mov   dx,start
   int   25H
   endm
```

Example

The following program copies the contents of a single-sided disk in drive A: to the disk in drive B:. It uses a buffer of 32K bytes:

```
prompt    db  "Source in A, target in B",13,10
          db  "Any key to start. $"
start     dw  0
buffer    db  64 dup (512 dup (?)) ;64 sectors
          .
          .
int_25H:    display prompt ;see Function 09H
            read_kbd       ;see Function 08H

      mov cx,5                    ;copy 5 groups of
                                  ;64 sectors
copy: push cx                     ;save the loop
                                  ;counter
      abs_disk_read 0,buffer, 64,start ;THIS
                                  ;INTERRUPT
      abs_disk_write 1,buffer, 64,start ;see INT
                                  ;26H
      add start,64                ;do the next 64
                                  ;sectors
      pop cx                      ;restore the loop
                                  ;counter
      loop copy
```

26H
Absolute Disk Write

Call

AL
  Drive number (0 = A, 1 = B, etc.)
DS:BX
  Disk Transfer Address
CX
  Number of sectors
DX
  Beginning relative sector

Return

FLAGS
CF = 0 if successful
  = 1 if not successful
AL
  Error code if CF = 1

Remarks

This interrupt transfers control to the MS-DOS BIOS. The number of sectors specified in CX is written from the Disk Transfer Address to the disk. Its requirements and processing are identical to Interrupt 25H, except data is written to the disk rather than read from it.

26H
Absolute Disk Write

Note

All registers except the segment registers are destroyed by this call. Be sure to save any registers your program uses before issuing the interrupt.

The system pushes the flags at the time of the call; they are still there upon return. (This is necessary because data is passed back in the flags). Be sure to pop the stack upon return to restore your stack pointer at the point of invocation.

If the disk operation was successful, the Carry Flag (CF) is 0. If the disk operation was not successful, CF is 1 and AL contains the MS-DOS error code (see Interrupt 24H earlier in this section for the codes and their meaning).

Macro

```
abs_disk_write macro disk,buffer,num_sectors, start
   mov   al,disk
   mov   bx,offset buffer
   mov   cx,num_sectors
   mov   dx,start
   int   26H
   endm
```

Example The following program copies the contents of a single-sided disk in drive A: to the disk in drive B:, verifying each write. It uses a buffer of 32K bytes:

```
off         equ 0
on          equ 1
            .
            .
prompt      db  "Source in A, target in B", 13,10
            db  "Any key to start. $"
start       dw  0

buffer      db 64 dup (512 dup (?)) ; 64 sectors
            .
            .
int_26H:    display prompt              ;see Function 09H
            read_kbd                    ;see Function 08H
            verify on                   ;see Function 2EH
            mov cx,5                    ;copy 5 groups of
                                        ;64 sectors
copy:       push  cx                    ;save the loop
                                        ;counter
            abs_disk_read 0,buffer,  64,start  ;see INT
                                        ;25H
            abs_disk_write 1,buffer,  64,start  ;THIS
                                        ;INTERRUPT
            add start,64                ;do the next 64
                                        ;sectors
            pop cx                      ;restore the loop
                                        ;counter
            loop copy
            verify off                  ;see Function 2EH
```

27H
Terminate But Stay Resident

Call

CCS:DX
First byte following
last byte of code

Return

None

Remarks

The Terminate But Stay Resident call is used to make a piece of code remain resident in the system after its termination. Typically, this call is used in .COM files to allow some device-specific interrupt handler to remain resident to process asynchronous interrupts.

DX must contain the number of bytes in the CS segment to be reserved. When Interrupt 27H is executed, the program terminates but is treated as an extension of MS-DOS; it remains resident and is not overlaid by other programs when it terminates.

If an executable file whose extension is .COM or .EXE ends with this interrupt, it becomes a resident operating system command.

This interrupt is provided for compatibility with versions of MS-DOS prior to 2.0. New programs should use Function 31H, Keep Process.

Macro

```
stay_resident macro last_instruc
   mov   dx,offset last_instruc
   inc   dx
   int   27H
   endm
```

27H
Terminate But Stay Resident

Example

```
;CS must be equal to PSP values given at program
;start
;(ES and DS values)
;the variable Last Address must be equal
;to the offset of the last byte in the
;program.
   mov   DX,LastAddress
   inc   dx
   int   27H
;There is no return from this interrupt
```

00H
Terminate Program

Call

AH = 00H
CS
Segment address of
Program Segment Prefix

Return

None

Remarks

Function 00H is called by Interrupt 20H; it performs the same processing.

The CS register must contain the segment address of the Program Segment Prefix before you call this interrupt.

The following exit addresses are restored from the specified offsets in the Program Segment Prefix:

Program terminate	0AH
<CTRL C>	0EH
Critical error	12H

All file buffers are flushed to disk.

Warning

Close all files that have changed in length before calling this function. If a changed file is not closed, its length is not recorded correctly in the directory. See Function 10H for a description of the Close File system call.

00H
Terminate Program

Macro

```
terminate_program macro
   xor  ah,ah
   int  21H
   endm
```

Example

```
;CS must be equal to PSP values given at program start
;(ES and DS values)
   mov   ah,0
   int   21H
;There are no returns from this interrupt
```

01H
Read Keyboard and Echo

Call AH = 01H

Return AL
Character typed

Remarks Function 01H waits for a character to be typed at the keyboard, then echoes the character to the display and returns it in AL. If the character is **CTRL C,** Interrupt 23H is executed.

Macro

```
read_kdb_and_echo macro
  mov    ah,01H
  int    21H
  endm
```

Example The following program both displays and prints characters as they are typed. If **CR** is pressed, the program sends Line Feed-Carriage Return to both the display and the printer:

```
func_01H: read_kbd_and_ echo       ;THIS FUNCTION
          print_char    al         ;see Function 05H
          cmp           al,0DH     ;is it a CR?
          jne           func_01H   ;no, print it
          print_char    10         ;see Function 05H
          display_char  10         ;see Function 02H
          jmp           func_01H   ;get another character
```

02H
Display Character

Call

AH = 02H
DL
Character to be displayed

Return

None

Remarks

If **CTRL C** is typed, Interrupt 23H is issued.

Macro

```
display_char macro  character
  mov   dl,character
  mov   ah,02H
  int   21H
  endm
```

Example

The following program converts lowercase characters to uppercase before displaying them:

```
func_02H:   read_kbd            ;see FUNCTION 08H
            cmp   al,"a"
            jl    uppercase     ;don't convert
            cmp   al,"z"
            jg    uppercase     ;don't convert
            sub   al,20H        ;convert to ASCII code
                                ;for uppercase
uppercase:  display_char al     ;THIS FUNCTION
            jmp   func_02H      ;get another character
```

03H
Auxiliary Input

Call

AH = 03H

Return

AL
Character from auxiliary device

Remarks

Function 03H waits for a character from the auxiliary input device, then returns the character in AL.

This system call does not return a status or error code.

If a **CTRL C** has been typed at console input, Interrupt 23H is issued.

Macro

```
aux_input macro
  mov  ah,03H
  int  21H
  endm
```

Example

The following program prints characters as they are received from the auxiliary device. It stops printing when an end-of-file character (ASCII 26, or **CTRL Z**) is received:

```
func_03H:  aux_input                ;THIS FUNCTION
           cmp        al,1AH        ;end of file?
           je         continue      ;yes, all done
           print_char al            ;see Function 05H
           jmp        func_03H      ;get another character
continue:  .
           .
```

04H
Auxiliary Output

Call

AH = 04H
DL
Character for auxiliary device

Return

None

Remarks

This system call does not return a status or error code.

If a **CTRL C** has been typed at console input, Interrupt 23H is issued.

Macro

```
aux_output macro character
  mov  dl,character
  mov  ah,04H
  int  21H
  endm
```

Example

The following program gets a series of strings of up to 80 bytes from the keyboard, sending each to the auxiliary device. It stops when a null string (CR only) is typed:

```
string      db      82 dup(?)                ;see Function 0AH
            .
            .
func_04H:   get_string 80,string             ;see Function 0AH
            cmp     string[1],0              ;null string?
            je      continue                 ;yes, all done
            xor     ch,ch                    ;zero high byte
            mov     cl, byte ptr string[1]   ;get string length
            mov     bx,0                     ;set index to 0
send_it:    aux_output string[bx+2]          ;THIS FUNCTION
            inc     bx                       ;bump index
            loop    send_it                  ;send another character
            jmp     func_04H                 ;get another string
continue:   .
            .
```

05H
Print Character

Call

AH = 05H
DL
Character for printer

Return

None

Remarks

If **CTRL C** has been typed at console input, Interrupt 23H is issued.

Macro

```
print_char macro character
  mov  dl,character
  mov  ah,05H
  int  21H
  endm
```

Example The following program prints a walking test pattern on the printer. It stops if **CTRL C** is pressed.

```
line_num    db          0
            .
            .
func_05H:   mov         bl,33           ;first printable ASCII
                                        ;character (!)
            add         bl,line_num     ;to offset next character
            mov         cx,80           ;loop counter for line
print_it:   print_char bl               ;THIS FUNCTION
            inc         bl              ;move to next ASCII character
            cmp         bl,126          ;last printable ASCII
                                        ;character (~)
            jle         no_reset        ;not there yet
            mov         bl,33           ;start over with (!)

no_reset:   loop        print_it        ;print another character
            print_char 13               ;carriage return
            print_char 10               ;line feed
            inc         line_num        ;to offset 1st char. of line
            cmp         line_num, 93    ;end of cycle?
            jle         func_05H        ;nope, not yet
            mov         line_num, 0     ;reset char offset
            jmp         func_05H        ;continue
```

06H
Direct Console I/O

Call

AH = 06H
DL
  See Text

Return

AL
  If DL = FFH before call
    If Zero flag not set:
      Character from keyboard
    If Zero flag set:
      No character input

Remarks

Processing depends on the value in DL when the function is called:

DL is FFH.
  If a character has been typed at the keyboard, it is returned in AL and the Zero flag is 0; if a character has not been typed, the Zero flag is 1.

DL is not FFH.
  The character in DL is displayed.

This function does not check for **CTRL C.**

Macro

```
dir_console_io macro switch
   mov   dl,switch
   mov   ah,06H
   int   21H
   endm
```

Example The following program acts as a stopwatch. When a character is typed, it sets the system clock to zero and begins to continuously display the time. When a second character is typed the system stops updating the time display.

```
time        db"00:00:00.00",13,"$"
ten         db 10
            .
            .
func_06H:   dir_console_io  0FFH    ;THIS FUNCTION
            jz      func_06H        ;wait for keystroke
            set_time        0,0,0,0 ;see Function 2DH
read_clock: get_time                ;see Function 2CH
            convert ch,ten,time     ;see end of chapter
            convert cl,ten,time[3]  ;see end of chapter
            convert dh,ten,time[6]  ;see end of chapter
            convert dl,ten,time[9]  ;see end of chapter
            display time            ;see Function 09H
            dir_console_io  0FFH    ;THIS FUNCTION
            jz      read_clock      ;no char, keep updating
continue:   .
            .
            .
```

07H
Direct Console Input

Call AH = 07H

Return AL
Character from keyboard

Remarks This function does not echo the character or check for **CTRL C.** (For a keyboard input function that echoes or checks for **CTRL C,** see Functions 01H or 08H.)

Macro

```
dir_console_input macro
                 mov   ah,07H
                 int   21H
                 endm
```

Example The following program fragment prompts for a password (8 characters maximum) and places the characters into a string without echoing them:

```
password    db     8 dup(?)
prompt      db     "Password: $"     ;see Function 09H for
                                     ;explanation of $
            .
            .
func_07H:   display prompt           ;see Function 09H
            mov    cx,8              ;maximum length of password
            xor    bx,bx             ;so BL can be used as index
get_pass:   dir_console_input        ;THIS FUNCTION
            cmp    al,0DH            ;was it a CR?
            je     continue          ;yes, all done
            mov    password[bx],al   ;no, put character in string
            inc    bx                ;bump index
            loop   get_pass          ;get another character
continue:   .                        ;BX has length of password
            .
```

08H
Read Keyboard

Call AH = 08H

Return AL
Character from keyboard

Remarks If **CTRL C** is pressed, Interrupt 23H is executed. This function does not echo the character. (For a keyboard input function that echoes the character or does not check for **CTRL C**, see Functions 01H or 07H.)

Macro

```
read_kbd macro
     mov   ah,08H
     int   21H
     endm
```

Example The following program fragment prompts for a password (8 characters maximum) and places the characters into a string without echoing them:

```
password    db     8 dup(?)
prompt      db     "Password: $"     ;see Function 09H
                                     ;for explanation of $
            .
            .
func_08H:   display prompt           ;see Function 09H
            mov    cx,8              ;maximum length of password
            xor    bx,bx             ;BL can be an index
get_pass:   read_kbd;                ;THIS FUNCTION
            cmp    al,0DH            ;was it a CR?
            je     continue          ;yes, all done
            mov    password[bx],al   ;no, put char. in string
            inc    bx                ;bump index
            loop   get_pass          ;get another character
continue:   .                        ;BX has length of password
            .
```

09H
Display String

Call

AH = 09H
DS:DX
String to be displayed

Return

None

Remarks

DX must contain the offset (from the segment address in DS) of a string that ends with "$". The string is displayed (the $ is not displayed).

Macro

```
display macro string
  mov   dx,offset string
  mov   ah,09H
  int   21H
  endm
```

Example

The following program displays the hexadecimal code of the key that is typed:

```
table       db  "0123456789ABCDEF"
sixteen     db  16
result      db  "-00H",13,10,"$"        ;see text for
                                        ;explanation of $
            .
            .
func_09H:   read_kbd_and_echo           ;see Function 01H
            convert al,sixteen,result[1] ;see end of chapter
            display result              ;THIS FUNCTION
            jmp  func_09H               ;do it again
```

0AH
Buffered Keyboard Input

Call

AH = 0AH
DS:DX
Input buffer

Return

None

Remarks

DX must contain the offset (from the segment address in DS) of an input buffer of the following form:

Byte	Contents
1	Maximum number of characters in buffer, including the CR (you must set this value).
2	Actual number of characters typed, not counting the CR (the function sets this value).
3-n	Buffer; must be at least as long as the number in byte 1.

This function waits for characters to be typed. Characters are read from the keyboard and placed in the buffer beginning at the third byte until **CR** is pressed. If the buffer fills to one less than the maximum, additional characters typed are ignored and ASCII 7 (BEL) is sent to the display until **CR** is pressed. The string can be edited as it is being entered. If **CTRL C** is typed, Interrupt 23H is issued.

The second byte of the buffer is set to the number of characters entered (not counting the **CR**).

Macro

```
get_string macro limit,string
   mov   dx,offset string
   mov   string,limit
   mov   ah,0AH
   int   21H
   endm
```

Example

The following program gets a 16-byte (maximum) string from the keyboard and fills a 24-line by 80-character screen with it:

```
buffer          label     byte
max_length      db        ?                ;maximum length
chars_entered   db        ?                ;number of chars.
string          db        17 dup (?)       ;16 chars + CR
strings_per_line dw       0                ;how many strings
                                           ;fit on line
crlf            db        13,10,"$"
                .
                .
func_0AH:       get_string 17,buffer       ;THIS FUNCTION
                xor       bx,bx            ;so byte can be
                                           ;used as index
                mov       bl,chars_entered ;get string length
                mov       buffer[bx+2],"$" ;see Function 09H
                mov       al,50H           ;columns per line
                cbw
                div       chars_entered    ;times string fits
                                           ;on line
                xor       ah,ah            ;clear remainder
                mov       strings_per_line,ax ;save col. counter
                mov       cx,24            ;row counter
display_screen: push cx                    ;save it
                mov       cx,strings_per_line ;get col. counter
display_line:   display   string           ;see Function 09H
                loop      display_line
                display   crlf             ;see Function 09H
                pop       cx               ;get line counter
                loop      display_screen   ;display 1 more line
```

0BH
Check Keyboard Status

Call AH = 0BH

Return AL
FFH = characters in type-ahead buffer
0 = no characters in type-ahead buffer

Remarks Checks whether there are characters in the type-ahead buffer. If so, AL returns FFH; if not, AL returns 0. If **CTRL C** is in the buffer, Interrupt 23H is executed.

Macro

```
check_kbd_status macro
                mov   ah,0BH
                int   21H
                endm
```

Example The following program fragment continuously displays the time until any key is pressed.

```
time        db   "00:00:00.00",13,10,"$"
ten         db   10
            .
            .
func_0BH:   get_time                ;see Function 2CH
            convert ch,ten,time     ;see end of chapter
            convert cl,ten,time[3]  ;see end of chapter
            convert dh,ten,time[6]  ;see end of chapter
            convert dl,ten,time[9]  ;see end of chapter
            display time            ;see Function 09H
            check_kbd_status        ;THIS FUNCTION
            cmp   al,0FFH           ;has a key been typed
            je    all_done          ;yes, go home
            jmp   func_0BH          ;no, keep displaying
                                    ;time
all_done:   .
            .
```

0CH
Flush Buffer, Read Keyboard

Call

AH = 0CH
AL
1, 6, 7, 8, or 0AH = The corresponding function is called. Any other value = return from function.

Return

AL
0 = Type-ahead buffer was flushed; no other processing performed.

Remarks

The keyboard type-ahead buffer is emptied. Further processing depends on the value in AL when the function is called:

1, 6, 7, 8, or A:
The corresponding MS-DOS function is executed.

Any other value:
No further processing; AL returns 0.

Macro

```
flush_and_read_kbd  macro switch
                    mov   al,switch
                    mov   ah,0CH
                    int   21H
                    endm
```

0CH
Flush Buffer, Read Keyboard

Example The following program both displays and prints characters as they are typed. If **CR** is pressed, the program sends Carriage Return-Line Feed to both the display and the printer.

```
func_0CH:  flush_and_read_kbd 1      ;THIS FUNCTION
           print_char      al        ;see Function 05H
           cmp             al,0DH    ;is it a CR?
           jne             func_0CH  ;no, print it
           print_char      10        ;see Function 05H
           display_char    10        ;see Function 02H
           print_char      13        ;see Function 05H
           display_char    13        ;see Function 02H
           jmp             func_0CH  ;get another character
```

0DH
Disk Reset

Call AH = 0DH

Return None

Remarks Function 0DH is used to ensure that the internal buffer cache matches the disks in the drives. If buffers have been modified, but not yet written to disk, this function writes them out and marks all buffers in the internal cache as free.

Function 0DH flushes (frees) all file buffers. It does not update directory entries; you must close files that have changed to update their directory entries (see Function 10H, Close File). This function need not be called before a disk change if all files that changed were closed. It is generally used to force a known state of the system; **CTRL C** interrupt handlers should call this function.

Macro

```
disk_reset macro disk
  mov   ah,0DH
  int   21H
  endm
```

Example

```
  mov   ah,0DH
  int   21H
;There are no errors returned by this call.
```

0EH
Select Disk

Call

AH = 0EH
DL
  Drive number
  (0 = A:, 1 = B:, etc.)

Return

AL
  Number of logical drives

Remarks

The drive specified in DL (0 = A:, 1 = B:, etc.) is selected as the default disk. The number of drives is returned in AL.

Macro

```
select_disk macro disk
  mov  dl,disk[-65]   ;ASCII offset
  mov  ah,0EH
  int  21H
  endm
```

Example

The following program fragment selects the drive not currently selected in a 2-drive system:

```
func_0EH:   current_disk       ;see Function 19H
            cmp   al,00H       ;drive A: selected?
            je    select_b     ;yes, select B
            select_disk "A"    ;THIS FUNCTION
            jmp   continue
select_b:   select_disk "B"    ;THIS FUNCTION
continue:   .
            .
```

0FH
Open File

Call

AH = 0FH
DS:DX
  Unopened FCB

Return

AL
  0 = Directory entry found
  FFH = No directory entry found

Remarks

DX must contain the offset (from the segment address in DS) of an unopened File Control Block (FCB). The disk directory is searched for the named file.

If a directory entry for the file is found, AL returns 0 and the FCB is filled as follows:

- If the drive code was 0 (default disk), it is changed to the actual disk used (1 = A:, 2 = B:, etc.). This lets you change the default disk without interfering with subsequent operations on this file.
- The Current Block field (offset 0CH) is set to zero.
- The Record Size (offset 0EH) is set to the system default of 128.
- The File Size (offset 10H), Date of Last Write (offset 14H), and Time of Last Write (offset 16H) are set from the directory entry.

Before performing a sequential disk operation on the file, you must set the Current Record field (offset 20H). Before performing a random disk operation on the file, you must set the Relative Record field (offset 21H). If the default record size (128 bytes) is not correct, set it to the correct length.

If a directory entry for the file is not found, AL returns FFH.

Macro

```
open macro fcb
  mov   dx,offset fcb
  mov   ah,0FH
  int   21H
  endm
```

Example

The following program prints the file named TEXTFILE. ASC that is on the disk in drive B:. If a partial record is in the buffer at end-of-file, the routine that prints the partial record prints characters until it encounters an end-of-file mark (ASCII 26, or **CTRL Z**):

```
fcb             db    2,"TEXTFILEASC"
                db    25 dup (?)
buffer          db    128 dup (?)
                .
                .
func_0FH:       set_dta buffer          ;see Function 1AH
                open fcb                ;THIS FUNCTION
read_line:      read_seq fcb            ;see Function 14H
                cmp   al,01H            ;end of file?
                je    all_done          ;yes, go home
                cmp   al,00H            ;more to come?
                jg    check_more        ;no, check for partial
                                        ;record
                mov   cx,128            ;yes, print the buffer
                xor   si,si             ;set index to 0
print_it:       print_char buffer[si]   ;see Function 05H
                inc   si                ;bump index
                loop  print_it          ;print next character
                jmp   read_line         ;read another record
check_more:     cmp   al,03H            ;part. record to print?
                jne   all_done          ;no
```

```
                mov     cx,128          ;yes, print it
                xor     si,si           ;set index to 0
find_eof:       cmp     buffer[si],26   ;end-of-file mark?
                je      all_done        ;yes
                print_char buffer[si]   ;see Function 05H
                inc     si              ;bump index to next
                                        ;character
                loop    find_eof
all_done:       close   fcb             ;see Function 10H
```

10H
Close File

Call

AH = 10H
DS:DX
  Opened FCB

Return

AL
  0 = Directory entry found
  FFH = No directory entry found

Remarks

DX must contain the offset (to the segment address in DS) of an opened FCB. The disk directory is searched for the file named in the FCB. This function must be called after a file is changed to update the directory entry.

If a directory entry for the file is found, the entry is compared with the corresponding entries in the FCB. The directory entry is updated, if necessary, to match the FCB, and AL returns 0.

If a directory entry for the file is not found, AL returns FFH.

Macro

```
close macro fcb
  mov   dx,offset fcb
  mov   ah,10H
  int   21H
  endm
```

Example The following program checks the first byte of the file named MOD1.BAS in drive B: to see if it is FFH, and prints a message if it is:

```
message     db   "Not saved in ASCII format",13,10,"$"
fcb         db   2,"MOD1    BAS"
            db   25 dup (?)
buffer      db   128 dup (?)
            .
            .
func_10H:   set_dta buffer        ;see Function 1AH
            open   fcb            ;see Function 0FH
            read_seq fcb          ;see Function 14H
            cmp    buffer,0FFH    ;is first byte FFH?
            jne    all_done       ;no
            display message       ;see Function 09H
all_done    close  fcb            ;THIS FUNCTION
```

11H
Search for First Entry

Call

AH = 11H
DS:DX
  Unopened FCB

Return

AL
  0 = Directory entry found
  FFH = No directory entry found

Remarks

DX must contain the offset (from the segment address in DS) of an unopened FCB. The disk directory is searched for the first matching name. The name can have the ? wild card character to match any character. To search for hidden or system files, DX must point to the first byte of the extended FCB prefix.

If a directory entry for the filename in the FCB is found, AL returns 0 and an unopened FCB of the same type (normal or extended) is created at the Disk Transfer Address.

If a directory entry for the filename in the FCB is not found, AL returns FFH.

Note

If an extended FCB is used, the following search pattern is used:

- If the FCB attribute is zero, only normal file entries are found. Entries for volume label, subdirectories, hidden, and system files will not be returned.

- If the attribute field is set for hidden or system files, or directory entries, it is to be considered as an inclusive search. All normal file entries plus all entries matching the specified attributes are returned. To look at all directory entries except the volume label, the attribute byte may be set to hidden + system + directory (all 3 bits on).

- If the attribute field is set for the volume label, it is considered an exclusive search, and only the volume label entry is returned.

Macro

```
search_first macro fcb
  mov  dx,offset fcb
  mov  ah,11H
  int  21H
  endm
```

Example

The following program verifies the existence of a file named REPORT. ASM on the disk in drive B::

```
yes         db   "FILE EXISTS.$"
no          db   "FILE DOES NOT EXIST.$"
fcb         db   2,"REPORT  ASM"
            db   25 dup (?)
buffer      db   128 dup (?)
crlf        db   13,10 "$"
            .
func_11H:   set_dta buffer     ;see Function 1AH
            search_first fcb   ;THIS FUNCTION
            cmp  al,0FFH       ;directory entry found?
            je   not_there     ;no
            display yes        ;see Function 09H
            jmp  continue
not_there:  display no         ;see Function 09H
continue:   display crlf       ;see Function 09H
            .
            .
```

12H
Search for Next Entry

Call

AH = 12H
DS:DX
  Unopened FCB

Return

AL
  0 = Directory entry found
  FFH = No directory entry found

Remarks

DX must contain the offset (from the segment address in DS) of an FCB previously specified in a call to Function 11H (Search for First Entry). Function 12H is used after Function 11H to find additional directory entries that match a file-name that contains wild card characters. The disk directory is searched for the next matching name.

If a directory entry for the filename in the FCB is found, AL returns 0 and an unopened FCB of the same type (normal or extended) is created at the Disk Transfer Address.

If a directory entry for the filename in the FCB is not found, AL returns FFH.

Macro

```
search_next macro fcb
  mov   dx,offset fcb
  mov   ah,12H
  int   21H
  endm
```

Example The following program displays the number of files on the disk in drive B:

```
message     db   "No files",10,13,"$"
files       db   0
ten         db   10
fcb         db   2,"???????????"
            db   25dup (?)
buffer      db   128 dup (?)
            .
            .
func_12H:   set_dta buffer              ;see Function 1AH
            search_first fcb            ;see Function 11H
            cmp  al,0FFH                ;directory entry found?
            je   all_done               ;no, no files on disk
            inc  files                  ;yes, increment file
                                        ;counter
search_dir: search_next fcb             ;THIS FUNCTION
            cmp  al,0FFH                ;directory entry found?
            je   done                   ;no
            inc  files                  ;yes, increment file
                                        ;counter
            jmp  search_dir             ;check again
done:       convert files,ten,message   ;see end of chapter
all_done:   display message             ;see Function 09H
```

13H
Delete File

Call

AH = 13H
DS:DX
  Unopened FCB

Return

AL
  0 = Directory entry found
  FFH = No directory entry found

Remarks

DX must contain the offset (from the segment address in DS) of an unopened FCB. The directory is searched for a matching filename. The filename in the FCB can contain the ? wild card character to match any character.

If a matching directory entry is found, it is deleted from the directory. If the ? wild card character is used in the filename, all matching directory entries are deleted. AL returns 0.

If no matching directory entry is found, AL returns FFH.

Macro

```
delete macro fcb
  mov   dx,offset fcb
  mov   ah,13H
  int   21H
  endm
```

Example The following deletes each file on the disk in drive B: that was last written before June 30, 1984:

```
year        dw  1984
month       db  6
day         db  30
files       db  0
ten         db  10
message     db  "NO FILES DELETED.",13,10,"$"
                                    ;see Function 09H for
                                    ;explanation of $
fcb         db  2,"???????????"
            db  25 dup(?)

buffer      db  128 dup (?)
            .
            .
func_13H:   set_dta  buffer         ;see Function 1AH
            search_first fcb        ;see Function 11H
            cmp   al,0FFH           ;directory entry found?
            je    all_done          ;no, no files on disk

compare:    convert_date buffer     ;see end of chapter
            cmp   cx,year           ;next several lines
            jg    next              ;check date in directory
            cmp   dl,month          ;entry against date
            jg    next              ;above & check next file
            cmp   dh,day            ;if date in directory
            jge   next              ;entry isn't earlier.
            delete buffer           ;THIS FUNCTION
            inc   files             ;bump deleted-files
                                    ;counter
next:       search_next fcb         ;see Function 12H
            cmp   al,00H            ;directory entry found?
            je    compare           ;yes, check date
            cmp   files,0           ;any files deleted?
            je    all_done          ;no, display NO FILES
```

```
                                                ;message.
                convert files,ten,message       ;see end of chapter
all_done:       display message                 ;see Function 09H
```

14H
Sequential Read

Call

AH = 14H
DS:DX
  Opened FCB

Return

AL
  0 = Read completed successfully
  1 = EOF
  2 = DTA too small
  3 = EOF, partial record

Remarks

DX must contain the offset (from the segment address in DS) of an opened FCB. The record pointed to by the current block (offset 0CH) and Current Record (offset 20H) fields is loaded at the Disk Transfer Address, then the Current Record and, if necessary, the Current Block fields are incremented.

The record size is set to the value at offset 0EH in the FCB.

AL returns a code that describes the processing:

Code	Meaning
0	Read completed successfully.
1	End-of-file, no data in the record.
2	Not enough room at the Disk Transfer Address to read one record; read canceled.
3	End-of-file; a partial record was read and padded to the record length with zeros.

14H
Sequential Read

Macro

```
read_seq macro fcb
   mov   dx,offset fcb
   mov   ah,14H
   int   21H
   endm
```

Example

The following program displays the file named TEXTFILE.ASC that is on the disk in drive B:; its function is similar to the MS-DOS TYPE command. If a partial record is in the buffer at end of file, the routine that displays the partial record displays characters until it encounters an end-of-file mark (ASCII 26, or **CTRL Z**):

```
fcb          db   2,"TEXTFILEASC"
             db   25 dup (?)
buffer       db   128 dup (?),"$"
             .
             .
func_14H:    set_dta buffer          ;see Function 1AH
             open   fcb              ;see Function 0FH
read_line:   read_seq fcb            ;THIS FUNCTION
             cmp    al,01H           ;end-of-file?
             je     all_done         ;yes
             cmp    al,00H           ;more to come?
             jg     check_more       ;no, check for partial record
             display buffer          ;see Function 09H
             jmp    read_line        ;get another record
check_more:  cmp    al,03H           ;partial record in buffer?
             jne    all_done         ;no, go home
             xor    si,si            ;set index to 0
find_eof:    cmp    buffer[si],26    ;is character EOF?
             je     all_done         ;yes, no more to display
             display_char buffer[si] ;see Function 02H
             inc    si               ;bump index to next
```

```
                                                ;character
                jmp     find_eof                ;check next character
all_done:       close   fcb                     ;see Function 10H
```

15H
Sequential Write

Call

AH = 15H
DS:DX
  Opened FCB

Return

AL
  00H = Write completed successfully
  01H = Disk full
  02H = DTA too small

Remarks

DX must contain the offset (from the segment address in DS) of an opened FCB. The record pointed to by Current Block (offset 0CH) and Current Record (offset 20H) fields is written from the Disk Transfer Address, then the fields are incremented as necessary.

The record size is set to the value at offset 0EH in the FCB. If the Record Size is less than a sector, the data at the Disk Transfer Address is written to a buffer; the buffer is written to disk when it contains a full sector of data, or the file is closed, or a Reset Disk system call (Function 0DH) is issued.

AL returns a code that describes the processing:

Code	Meaning
0	transfer completed successfully
1	disk full; write canceled
2	write canceled; the area beginning at the Disk Transfer Address is too small to hold a record of data without overflowing or wrapping around a segment boundary.

Macro

```
write_seq macro fcb
  mov   dx,offset fcb
  mov   ah,15H
  int   21H
  endm
```

Example

The following program creates a file named DIR.TMP on the disk in drive B: that contains the disk number (0 = A:, 1 = B:, etc.) and file-name from each directory entry on the disk:

```
record_size  equ  14                    ;offset of Record Size
                                        ;field in FCB
             .
             .
fcb1         db   2,"DIR  TMP"
             db   25 dup (?)
fcb2         db   2,"???????????"
             db   25 dup (?)
buffer       db   128 dup (?)
             .
             .
func_15H:    set_dta        buffer      ;see Function 1AH
             search_first fcb2          ;see Function 11H
             cmp            al,0FFH     ;directory entry found?
             je             all_done    ;no, no files on disk
             create         fcb1        ;see Function 16H
             mov            fcb1[record_size],12
                                        ;set record size to 12
write_it:    write_seq      fcb1        ;THIS FUNCTION
             search_next    fcb2        ;see Function 12H
             cmp            al,0FFH     ;directory entry found?
             je             all_done    ;no, go home
             jmp            write_it    ;yes, write the record
all_done:    close          fcb1        ;see Function 10H
```

16H
Create File

Call

AH = 16H
DS:DX
  Unopened FCB

Return

AL
  00H = Empty directory entry found
  FFH = No empty entry directory available

Remarks

DX must contain the offset (from the segment address in DS) of an unopened FCB. The directory is searched for an empty entry or an existing entry for the specified filename.

If an empty directory entry is found, it is initialized to a zero-length file, the Open File system call (Function 0FH) is called, and AL returns 0. You can create a hidden file by using an extended FCB with the attribute byte (offset FCB-1) set to 2.

If an entry is found for the specified filename, all data in the file is released, making a zero-length file, and the Open File system call (Function 0FH) is issued for the filename (in other words, if you try to create a file that already exists, the existing file is erased, and a new, empty file is created).

If an empty directory entry is not found and there is no entry for the specified filename, AL returns FFH.

Macro

```
create macro fcb
  mov   dx,offset fcb
  mov   ah,16H
  int   21H
  endm
```

Example

The following program creates a file named DIR.TMP on the disk in drive B: that contains the disk number (0 = A:, 1 = B:, etc.) and file-name from each directory entry on the disk:

```
record_size   equ   14                    ;offset of Record Size
                                          ;field of FCB
              .
              .
fcb1          db    2,"DIR  TMP"
              db    25 dup (?)
fcb2          db    2,"???????????"
              db    25 dup (?)
buffer        db    128 dup (?)
              .
              .
func_16H:     set_dta          buffer     ;see Function 1AH
              search_first     fcb2       ;see Function 11H
              cmp              al,0FFH    ;directory entry found?
              je               all_done   ;no, no files on disk
              create           fcb1       ;THIS FUNCTION
              mov              fcb1[record_size],12
                                          ;set record size to 12
write_it:     write_seq        fcb1       ;see Function 15H
              search_next      fcb2       ;see Function 12H
              cmp              al,0FFH    ;directory entry found?
              je               all_done   ;no, go home
              jmp              write_it   ;yes, write the record
all_done:     close            fcb1       ;see Function 10H
```

17H
Rename File

Call

AH = 17H
DS:DX
  Modified FCB

Return

AL
  00H = Directory entry found
  FFH = No directory entry found or
  destination already exists

Remarks

DX must contain the offset (from the segment address in DS) of an FCB with the drive number and filename filled in, followed by a second filename at offset 11H. The disk directory is searched for an entry that matches the first filename, which can contain the ? wild card character.

If a matching directory entry is found, the filename in the directory entry is changed to match the second filename in the modified FCB (the two filenames cannot be the same name). If the ? wild card character is used in the second filename, the corresponding characters in the filename of the directory entry are not changed. AL returns 0.

If a matching directory entry is not found or an entry is found for the second filename, AL returns FFH.

Macro

```
rename macro special_fcb
  mov   dx,offset special_fcb
  mov   ah,17H
  int   21H
  endm
```

Example

The following program prompts for the name of a file and a new name, then renames the file:

```
fcb         db   37 dup (?)
prompt1     db   "Filename: $"
prompt2     db   "New name: $"
reply       db   17 dup(?)
crlf        db   13,10,"$"
            .
            .
func_17H:   display prompt1          ;see Function 09H
            get_string 15,reply      ;see Function 0AH
            display    crlf          ;see Function 09H
            parse      reply[2],fcb  ;see Function 29H
            display    prompt2       ;see Function 09H
            get_string 15,reply      ;see Function 0AH
            display    crlf          ;see Function 09H
            parse      reply[2],fcb[16]
                                     ;see Function 29H
            rename     fcb           ;THIS FUNCTION
```

19H
Current Disk

Call AH = 19H

Return AL
Currently selected drive
(0 = A:, 1 = B:, etc.)

Macro

```
current_disk macro
  mov   ah,19H
  int   21H
  endm
```

Example The following program displays the currently selected (default) drive:

```
message     db   "Current disk is $"   ;see Function 09H
                                       ;for explanation of $
crlf        db   13,10,"$"
            .
            .
func_19H:   display message            ;see Function 09H
            current_disk               ;THIS FUNCTION
            add  al, 41H               ;ASCII offset
            display_char al            ;see function 02H
            display crlf               ;see function 09H
```

1AH
Set Disk Transfer Address

Call

AH = 1AH
DS:DX
Disk Transfer Address

Return

None

Remarks

DX must contain the offset (from the segment address in DS) of the Disk Transfer Address. Disk transfers cannot wrap around from the end of the segment to the beginning, nor can they overflow into another segment.

Note:

If you do not set the Disk Transfer Address, MS-DOS defaults to offset 80H in the Program Segment Prefix.

The size of the buffer that the DTA points to must be greater than or equal to the record size at open file time.

Macro

```
set_dta macro buffer
  mov   dx,offset buffer
  mov   ah,1AH
  int   21H
  endm
```

Example The following program prompts for a letter, converts the letter to its alphabetic sequence (A = 1, B = 2, etc.), then reads and displays the corresponding record from a file named ALPHABET.DAT on the disk in drive B:. The file contains 26 records; each record is 28 bytes long:

```
record_size     equ   14          ;offset of Record Size
                                  ;field of FCB
relative_record equ   33          ;offset of Relative Record
                                  ;field of FCB
                .
                .
  fcb           db    2,"ALPHABETDAT"
                db    25 dup (?)
  buffer        db    34 dup(?),"$"
  prompt        db    "Enter letter: $"
  crlf          db    13,10,"$"
                .
                .
  func_1AH:     set_dta buffer                ;THIS FUNCTION
                open    fcb                   ;see Function 0FH
                mov     fcb[record_size],28   ;set record size
  get_char:     display prompt                ;see Function 09H
                read_kbd_and_echo             ;see Function 01H
                cmp     al,0DH                ;just a CR?
                je      all_done              ;yes, go home
                sub     al,41H                ;convert ASCII
                                              ;code to record #
                mov     fcb[relative_record],al
                                              ;set relative record
                display crlf                  ;see Function 09H
                read_ran fcb                  ;see Function 21H
                display buffer                ;see Function 09H
                display crlf                  ;see Function 09H
                jmp     get_char              ;get another character
  all_done:     close   fcb                   ;see Function 10H
```

21H
Random Read

Call

AH = 21H
DS:DX
  Opened FCB

Return

AL
  00H = Read completed successfully
  01H = EOF
  02H = DTA too small
  03H = EOF, partial record

Remarks

DX must contain the offset (from the segment address in DS) of an opened FCB. The Current Block (offset 0CH) and Current Record (offset 20H) fields are set to agree with the Relative Record field (offset 21H), then the record addressed by these fields is loaded at the Disk Transfer Address.

AL returns a code that describes the processing:

Code	Meaning
0	read completed successfully
1	End-of-file; no data in the record
2	not enough room at the Disk Transfer Address to read one record; read canceled
3	End-of-file; a partial record was read and padded to the record length with zeros.

Macro

```
read_ran macro fcb
  mov   dx,offset fcb
  mov   ah,21H
  int   21H
  endm
```

Example The following program prompts for a letter, converts the letter to its alphabetic sequence (A = 1, B = 2, etc.), then reads and displays the corresponding record from a file named ALPHABET.DAT on the disk in drive B:. The file contains 26 records; each record is 28 bytes long:

```
record_size     equ   14          ;offset of Record Size
                                  ;field of FCB
relative_record equ   33          ;offset of Relative Record
                                  ;field of FCB
                .
                .
fcb             db    2,"ALPHABETDAT"
                db    25 dup (?)
buffer          db    34 dup(?),"$"
prompt          db    "Enter letter: $"
crlf            db    13,10,"$"
                .
                .
func_21H:       set_dta buffer                ;see Function 1AH
                open  fcb                     ;see Function 0FH
                mov   fcb[record_size],28     ;set record size
get_char:       display prompt                ;see Function 09H
                read_kbd_and_echo             ;see Function 01H
                cmp   al,0DH                  ;just a CR?
                je    all_done                ;yes, go home
                sub   al,41H                  ;convert ASCII
                                              ;code to record #
                mov   fcb[relative_record],al
                                              ;set relative record
                display crlf                  ;see Function 09H
                read_ran fcb                  ;THIS FUNCTION
                display buffer                ;see Function 09H
                display crlf                  ;see Function 09H
                jmp   get_char                ;get another character
all_done:       close fcb                     ;see Function 10H
```

22H
Random Write

Call

AH = 22H
DS:DX
  Opened FCB

Return

AL
  00H = Write completed successfully
  01H = Disk full
  02H = DTA too small

Remarks

DX must contain the offset from the segment address in DS of an opened FCB. The Current Block (offset 0CH) and Current Record (offset 20H) fields are set to agree with the Relative Record field (offset 21H), then the record addressed by these fields is written from the Disk Transfer Address. If the record size is smaller than a sector (512 bytes), the records are buffered until a sector is ready to write.

AL returns a code that describes the processing:

Code	Meaning
0	Write completed successfully
1	disk is full
2	write canceled; the area beginning at the Disk Transfer Area is too small to hold a record of data without overflowing or wrapping around a segment boundary.

Macro

```
write_ran macro fcb
  mov   dx,offset fcb
  mov   ah,22H
  int   21H
  endm
```

Example The following program prompts for a letter, converts the letter to its alphabetic sequence (A = 1, B = 2, etc.), then reads and displays the corresponding record from a file named ALPHABET.DAT on the disk in drive B:. After displaying the record, it prompts the user to enter a changed record. If the user types a new record, it is written to the file; if the user just presses **CR** , the record is not replaced. The file contains 26 records; each record is 28 bytes long:

```
record_size      equ  14          ;offset of Record Size
                                  ;field of FCB
relative_record  equ  33          ;offset of Relative Record
                                  ;field of FCB
                 .
                 .
fcb              db   2,"ALPHABETDAT"
                 db   25 dup (?)
buffer           db   28 dup(?),13,10,"$"
prompt1          db   "Enter letter: $"
prompt2          db   "New record(CR for no change): $"
crlf             db   13,10,"$"
reply            db   30 dup (32)
blanks           db   28 dup (32)
                 .
                 .
func_22H:        set_dta buffer                ;see Function 1AH
                 open    fcb                   ;see Function 0FH
                 mov     fcb[record_size],28   ;set record size
get_char:        display prompt1               ;see Function 09H
                 read_kbd_and_echo             ;see Function 01H
                 cmp     al,0DH                ;just a CR?
                 je      all_done              ;yes, go home
                 sub     al,41H                ;convert ASCII
                                               ;code to record #
                 mov     fcb[relative_record],al
                                               ;set relative record
```

```
                display crlf                    ;see Function 09H
                read_ran fcb                    ;THIS FUNCTION
                display buffer                  ;see Function 09H
                display crlf                    ;see Function 09H
                display prompt2                 ;see Function 09H
                get_string 29, reply            ;see Function 09H
                display crlf                    ;see Function 09H
                cmp     reply[1],0              ;was anything typed
                                                ;besides cr?
                je      get_char                ;no
                                                ;get another character
                xor     bx,bx                   ;to load a byte
                mov     bl,reply[1]             ;use reply length as
                                                ;counter
                move_string blanks, buffer, 28
                                                ;see chapter end
                move_string reply[2], buffer, bx
                                                ;see chapter end
                write_ran fcb                   ;THIS FUNCTION
                jmp     get_char                ;get another character
all_done:       close   fcb                     ;see Function 10H
```

23H
File Size

Call

AH = 23H
DS:DX
  Unopened FCB

Return

AL
  00H = Directory entry found
  FFH = No directory entry found

Remarks

DX must contain the offset (from the segment address in DS) of an unopened FCB. You must set the Record Size field (offset 0EH) to the proper value before calling this function. The disk directory is searched for the first matching entry.

If a matching directory entry is found, the Relative Record field (offset 21H) is set to the number of records in the file, calculated from the total file size in the directory entry (offset 10H) and the Record Size field of the FCB (offset 0EH). AL returns 00.

If no matching directory entry is found, AL returns FFH.

Note

If the value of the Record Size field of the FCB (offset 0EH) doesn't match the actual number of characters in a record, this function may not return the correct file size. If the default record size (128) is not correct, you must set the Record Size field to the correct value before using this function.

Macro

```
file_size macro fcb
   mov   dx,offset fcb
   mov   ah,23H
   int   21H
   endm
```

Example

The following program prompts for the name of a file, opens the file to set the Record Size field of the FCB to 80H, issues a File Size system call, and displays the file size and number of records in hexadecimal:

```
fcb          db          37 dup(?)
prompt       db          "File name: $"
msg1         db          "Record length:  ",13,10,"$"
msg2         db          "Records:  ",13,10,"$"
crlf         db          13,10,"$"
reply        db          17 dup(?)
sixteen      db          16
             .
             .
func_23H:    display prompt              ;see Function 09H
             get_string 17,reply         ;see Function 0AH
             cmp         reply[1],0      ;just a CR?
             jne         get_length      ;no, keep going
             jmp         all_done        ;yes, go home
get_length:  display crlf                ;see Function 09H
             parse       reply[2],fcb    ;see Function 29H
             open        fcb             ;see Function 0FH
             file_size fcb               ;THIS FUNCTION
             mov         si,33           ;offset to Relative
                                         ;Record field
             mov         di,9            ;reply in msg2
convert_it:  cmp         fcb[si],0       ;digit to convert?
             je          show_it         ;no, prepare message
```

```
            convert fcb[si],sixteen,msg2[di]
            inc         si          ;bump n-o-r index
            inc         di          ;bump message index
            jmp         convert_it  ;check for a digit
show_it:    convert fcb[14],sixteen,msg1[15]
            display msg1            ;see Function 09H
            display msg2            ;see Function 09H
            jmp         func23H     ;get a filename
all_done:   close       fcb         ;see Function 10H
```

24H
Set Relative Record

Call

AH = 24H
DS:DX
  Opened FCB

Return

None

Remarks

DX must contain the offset (from the segment address in DS) of an opened FCB. The Relative Record field (offset 21H) is set to the same file address as the Current Block (offset 0CH) and Current Record (offset 20H) fields.

Macro

```
set_relative_record macro fcb
  mov   dx,offset fcb
  mov   ah,24H
  int   21H
  endm
```

Example

The following program copies a file using the Random Block Read and Random Block Write system calls. It speeds the copy by setting the record length equal to the file size and the record count to 1, and using a buffer of 32K bytes. It positions the file pointer by setting the Current Record field (offset 20H) to 0 and using Set Relative Record to make the Relative Record field (offset 21H) point to the same record as the combination of the Current Block (offset 0CH) and Current Record (offset 20H) fields:

```
current_record  equ   32                  ;offset of Current Record
                                          ;field of FCB
fsize           equ   16                  ;offset of File Size
                                          ;field of FCB
                .
                .
fcb             db    37 dup (?)
filename        db    17 dup(?)
prompt1         db    "File to copy: $"   ;see Function 09H for
prompt2         db    "Name of copy: $"   ;explanation of $
crlf            db    13,10,"$"

file_length     dw    ?
buffer          db    32767 dup(?)
                .
                .
func_24H:       set_dta       buffer      ;see Function 1AH
                display prompt1           ;see Function 09H
                get_string 15,filename    ;see Function 0AH
                display crlf              ;see Function 09H
                parse filename[2],fcb     ;see Function 29H
                open          fcb         ;see Function 0FH
                mov fcb[current_record],0 ;set Current Record
                                          ;field
                set_relative_record fcb   ;THIS FUNCTION
                mov ax,word ptr fcb[fsize] ;get file size
                mov           file_length,ax ;save it for
                                          ;ran_block_write
                ran_block_read fcb,1,ax   ;see Function 27H
                display       prompt2     ;see Function 09H
                get_string 15,filename    ;see Function 0AH
                display       crlf        ;see Function 09H
                parse filename[2],fcb     ;see Function 29H
                set_relative_record fcb   ;THIS FUNCTION
```

```
mov             ax,file_length      ;get original file
                                    ;length
ran_block_write fcb,1,ax            ;see Function 28H
close           fcb                 ;see Function 10H
```

25H
Set Vector

Call

AH = 25H
AL
  Interrupt number
DS:DX
  Interrupt-handling routine

Return

None

Remarks

Function 25H should be used to set a particular interrupt vector. The operating system can then manage the interrupts on a per-process basis. Note that programs should never set interrupt vectors by writing them directly in the low memory vector table.

DX must contain the offset (to the segment address in DS) of an interrupt-handling routine. AL must contain the number of the interrupt handled by the routine. The address in the vector table for the specified interrupt is set to DS:DX.

Macro

```
set_vector macro interrupt, seg_addr,off_addr
  push  ds
  mov   ax,seg_addr
  mov   ds,ax
  mov   dx,off_addr
  mov   al,interrupt
  mov   ah,25H
  int   21H
  pop   ds
  endm
```

Example

```
lds  dx,intvector
mov  ah,25H
mov  al,intnumber
int  21H
;There are no errors returned
```

27H
Random Block Read

Call

AH = 27H
DS:DX
  Opened FCB
CX
  Number of blocks to read

Return

AL
  00H = Read completed successfully
  01H = EOF
  02H = End of segment
  03H = EOF, partial record
CX
  Number of blocks read

Remarks

DX must contain the offset (to the segment address in DS) of an opened FCB. CX must contain the number of records to read; if it contains 0, the function returns without reading any records (no operation). The specified number of records, calculated from the Record Size field (offset 0EH), is read starting at the record specified by the Relative Record field (offset 21H). The records are placed at the Disk Transfer Address.

AL returns a code that describes the processing:

Code	Meaning
0	Read completed successfully
1	End-of-file; no data in the record
2	not enough room at the Disk Transfer Address to read one record without overflowing a segment boundary; read cancelled
3	End-of-file; a partial record was read and padded to the record length with zeros

CX returns the number of records read; the Current Block (offset 0CH), Current Record (offset 20H), and Relative Record (offset 21H) fields are set to address the next record.

Macro

```
ran_block_read macro fcb,count,rec_size
               mov   dx,offset fcb
               mov   cx,count
               mov   word ptr fcb[14],rec_size
               mov   ah,27H
               int   21H
               endm
```

Example

The following program copies a file using the Random Block Read system call. It speeds the copy by specifying a record count of 1 and a record length equal to the file size, and using a buffer of 32K bytes; the file is read as a single record (compare to the sample program for Function 28H that specifies a record length of 1 and a record count equal to the file size):

```
current_record  equ   32                ;offset of Current Record
                                        ;field
fsize           equ   16                ;offset of File Size
                                        ;field
                .
                .
fcb             db    37 dup (?)
filename        db    17 dup(?)
prompt1         db    "File to copy: $"   ;see Function 09H for
prompt2         db    "Name of copy: $"   ;explanation of $
crlf            db    13,10,"$"

file_length     dw    ?
buffer          db    32767 dup(?)
                .
                .
func_27H:       set_dta buffer            ;see Function 1AH
```

```
display prompt1                 ;see Function 09H
get_string 15,filename          ;see Function 0AH
display crlf                    ;see Function 09H
parse filename[2],fcb           ;see Function 29H
open    fcb                     ;see Function 0FH
mov fcb[current_record],0       ;set Current Record
                                ;field
set_relative_record fcb         ;see Function 24H
mov ax,word ptr fcb[fsize]      ;get file size
mov     file_length,ax          ;save it for
                                ;ran_block_write
ran_block_read fcb,1,ax         ;THIS FUNCTION
display prompt2                 ;see Function 09H
get_string 15,filename          ;see Function 0AH
display crlf                    ;see Function 09H
parse filename[2],fcb           ;see Function 29H
create  fcb                     ;see Function 16H
mov fcb[current_record],0       ;set Current Record
                                ;field
set_relative_record fcb         ;see Function 24H
mov     ax,file_length          ;get original file
                                ;size
ran_block_write fcb,1,ax        ;see Function 28H
close   fcb
```

28H
Random Block Write

Call

AH = 28H
DS:DX
  Opened FCB
CX
  Number of blocks to write
  (0 = set File Size field)

Return

AL
  00H = Write completed successfully
  01H = Disk full
  02H = End of segment
CX
  Number of blocks written

Remarks

DX must contain the offset (to the segment address in DS) of an opened FCB; CX must contain either the number of records to write or 0. The specified number of records (calculated from the Record Size field, offset 0EH) is written from the Disk Transfer Address. The records are written to the file starting at the record specified in the Relative Record field (offset 21H) of the FCB. If CX is 0, no records are written, but the File Size field of the directory entry (offset 10H) is set to the number of records specified by the Relative Record field of the FCB (offset 21H); allocation units are allocated or released, as required.

AL returns a code that describes the processing:

Code	Meaning
0	Write completed successfully
1	Disk full. No records written.
2	Not enough room at the Disk Transfer Address to write one record without overflowing a segment boundary; write canceled.

CX returns the number of records written; the current block (offset 0CH), Current Record (offset 20H), and Relative Record (offset 21H) fields are set to address the next record.

Macro

```
ran_block_write macro fcb,count,rec_size
                mov   dx,offset fcb
                mov   cx,count
                mov   word ptr fcb[14],rec_size
                mov   ah,28H
                int   21H
                endm
```

Example

The following program copies a file using the Random Block Read and Random Block Write system calls. It copies by specifying a record count equal to the file size and a record length of 1, and using a buffer of 32K bytes; the file is copied with one disk access each to read and write (compare to the sample program of Function 27H, that specifies a record count of 1 and a record length equal to file size):

```
current_record  equ  32                ;offset of Current Record
                                       ;field
fsize           equ  16                ;offset of File Size
                                       ;field
                .
                .
fcb             db   37 dup (?)
filename        db   17 dup(?)
prompt1         db   "File to copy: $"  ;see Function 09H for
prompt2         db   "Name of copy: $"  ;explanation of $
crlf            db   13,10,"$"

num_recs        dw   ?
buffer          db   32767 dup(?)
```

28H
Random Block Write

```
func_28H: set_dta      buffer                  ;see Function 1AH
          display      prompt1                 ;see Function 09H
          get_string   15,filename             ;see Function 0AH
          display      crlf                    ;see Function 09H
          parse        filename[2],fcb         ;see Function 29H
          open         fcb                     ;see Function 0FH
          mov          fcb[current_record],0   ;set Current Record
                                               ;field
          set_relative_record fcb              ;see Function 24H
          mov          ax,word ptr fcb[fsize]  ;get file size
          mov          num_recs, ax            ;save it for
                                               ;ran_block_write
          ran_block_read fcb,num_recs,1        ;see Function 27H
          display      prompt2                 ;see Function 09H
          get_string   15,filename             ;see Function 0AH
          display      crlf                    ;see Function 09H
          parse        filename[2],fcb         ;see Function 29H
          create       fcb                     ;see Function 16H
          mov          fcb[current_record],0   ;set Current Record
                                               ;field
          set_relative_record fcb              ;see Function 24H
          ran_block_write fcb,num_recs,1       ;THIS FUNCTION
          close        fcb                     ;see Function 10H
```

29H
Parse File Name

Call

AH = 29H
AL
  Controls parsing (see text)
DS:SI
  String to parse
ES:DI
  Unopened FCB

Return

AL
  00H = No wild card characters
  01H = Wild-card characters used
  FFH = Drive letter invalid
DS:SI
  First byte past string that was parsed
ES:DI
  Unopened FCB

Remarks

SI must contain the offset (to the segment address in DS) of a string (command line) to parse; DI must contain the offset (to the segment address in ES) of an unopened FCB. The string is parsed for a filename of the form d:filename.ext; if one is found, a corresponding unopened FCB is created at ES:DI.

Bits 0-3 of AL control the parsing and processing. Bits 4-7 are ignored:

Bit	Value	Meaning
0	0	All parsing stops if a file separator is encountered.
	1	Leading separators are ignored.
1	0	The drive number in the FCB is set to 0 (default drive) if the string does not contain a drive number.
	1	The drive number in the FCB is not changed if the string does not contain a drive number.
2	0	The filename in the FCB is set to 8 blanks if the string does not contain a filename.
	1	The filename in the FCB is not changed if the string does not contain a filename.
3	0	The extension in the FCB is set to 3 blanks if the string does not contain an extension.
	1	The extension in the FCB is not changed if the string does not contain an extension.

If the filename or extension includes an asterisk (*), all remaining characters in the name or extension are set to question mark (?).

Filename separators:

: . ; , = + / " [ ] \ < > | space tab

Filename terminators include all the filename separators plus any control character. A filename cannot contain a filename terminator; if one is encountered, parsing stops.

If the string contains a valid filename:

AL returns 1 if the filename or extension contains a wild card character (* or ?); AL returns 0 if neither the filename nor extension contains a wild card character.

DS:SI point to the first character following the string that was parsed.

ES:DI point to the first byte of the unopened FCB.

If the drive letter is invalid, AL returns FFH. If the string does not contain a valid filename, ES:DI+1 points to a blank (ASCII 20H).

29H
Parse File Name

Macro

```
parse macro string, fcb
        mov   si, offset string
        mov   di, offset fcb
        push  es
        push  ds
        pop   es
        mov   al,0FH  ;bits 0, 1, 2, 3 on
        mov   ah,29H
        int   21H
        pop   es
        endm
```

Example The following program verifies the existence of the file named in reply to the prompt:

```
fcb         db   37 dup (?)
prompt      db   "Filename: $"
reply       db   17 dup (?)
yes         db   "FILE EXISTS", 13,10,"$"
no          db   "FILE DOES NOT EXIST", 13,10,"$"
crlf        db   13,14,"$"
            .
            .
func_29H:   display      prompt          ;see Function 09H
            get_string   15,reply        ;see Function 0AH
            display      crlf            ;see Function 09H
            parse        reply[2],fcb    ;THIS FUNCTION
            search_first fcb             ;see Function 11H
            cmp          al,0FFH         ;dir. entry found?
            je           not_there       ;no
            display      yes             ;see Function 09H
            jmp          continue
not_there:  display      no
continue:   .
            .
```

2AH
Get Date

Call

AH = 2AH

Return

CX
  Year (1980-2099)
DH
  Month (1-12)
DL
  Day (1-31)
AL
  Day of week (0 = Sunday, 6 = Saturday)

Remarks

This function returns the current date set in the operating system as binary numbers in CX and DX:

CX	Year (1980-2099)
DH	Month (1 = January, 2 = February, etc.)
DL	Day (1-31)
AL	Day of week (0 = Sunday, 1 = Monday, etc.)

Macro

```
get_date macro
    mov   ah,2AH
    int   21H
    endm
```

Example

The following program gets the date, increments the day, increments the month or year, if necessary, and sets the new date:

```
month       db          31,28,31,30,31,30,31,31,30,31,30,31
            .
            .
func_2AH:   get_date                ;see above
            inc         dl          ;increment day
            xor         bx,bx       ;so BL can be used as index
            mov         bl,dh       ;move month to index register
            dec         bx          ;month table starts with 0
            cmp         dl,month[bx] ;past end of month?
            jle         month_ok    ;no, set the new date
            mov         dl,1        ;yes, set day to 1
            inc         dh          ;and increment month
            cmp         dh,12       ;past end of year?
            jle         month_ok    ;no, set the new date
            mov         dh,1        ;yes, set the month to 1
            inc         cx          ;increment year
month_ok:   set_date cx,dh,dl       ;THIS FUNCTION
```

2BH
Set Date

Call

AH = 2BH
CX
  Year (1980-2099)
DH
  Month (1-12)
DL
  Day (1-31)

Return

AL
  00H = Date was valid
  FFH = Date was invalid

Remarks

Registers CX and DX must contain a valid date in binary:

CX	Year (1980-2099)
DH	Month (1 = January, 2 = February, etc.)
DL	Day (1-31)

If the date is valid, the date is set and AL returns 0. If the date is not valid, the function is canceled and AL returns FFH.

Macro

```
set_date macro year,month,day
  mov   cx,year
  mov   dh,month
  mov   dl,day
  mov   ah,2BH
  int   21H
  endm
```

Example The following program gets the date, increments the day, increments the month or year, if necessary, and sets the new date:

```
month       db          31,28,31,30,31,30,31,31,30,31,30,31
            .
            .
func_2BH:   get_date                ;see Function 2AH
            inc         dl          ;increment day
            xor         bx,bx       ;so BL can be used as index
            mov         bl,dh       ;move month to index register
            dec         bx          ;month table starts with 0
            cmp         dl,month[bx] ;past end of month?
            jle         month_ok    ;no, set the new date
            mov         dl,1        ;yes, set day to 1
            inc         dh          ;and increment month
            cmp         dh,12       ;past end of year?
            jle         month_ok    ;no, set the new date
            mov         dh,1        ;yes, set the month to 1
            inc         cx          ;increment year
month_ok:   set date cx,dh,dl       ;THIS FUNCTION
```

Call AH = 2CH

Return

CH
  Hour (0-23)
CL
  Minutes (0-59)
DH
  Seconds (0-59)
DL
  Hundredths (0-99)

Remarks This function returns the current time set in the operating system as binary numbers in CX and DX:

CH Hour (0-23)
CL Minutes (0-59)
DH Seconds (0-59)
DL Hundredths of a second (0-99)

Macro

```
get_time  macro
  mov     ah,2CH
  int     21H
  endm
```

Example The following program continuously displays the time until any key is pressed:

```
time        db      "00:00:00.00",13,"$"
ten         db      10
            .
            .
func_2CH:   get_time                ;THIS FUNCTION
            convert ch,ten,time     ;see end of chapter
            convert cl,ten,time[3]  ;see end of chapter
            convert dh,ten,time[6]  ;see end of chapter
            convert dl,ten,time[9]  ;see end of chapter
            display time            ;see Function 09H
            check_kbd_status        ;see Function 0BH
            cmp     al,0FFH         ;has a key been pressed?
            je      all_done        ;yes, terminate
            jmp     func_2CH        ;no, display time
all_done:   .
            .
```

2DH
Set Time

Call

AH = 2DH
CH
  Hour (0-23)
CL
  Minutes (0-59)
DH
  Seconds (0-59)
DL
  Hundredths (0-99)

Return

AL
  00H = Time was valid
  FFH (255) = Time was invalid

Remarks

Registers CX and DX must contain a valid time in binary:

CH Hour (0-23)
CL Minutes (0-59)
DH Seconds (0-59)
DL Hundredths of a second (0-99)

If the time is valid, the time is set and AL returns 0. If the time is not valid, the function is canceled and AL returns FFH (255).

Macro

```
set_time  macro  hour,minutes,seconds,hundredths
  mov    ch,hour
  mov    cl,minutes
  mov    dh,seconds
  mov    dl,hundredths
  mov    ah,2DH
  int    21H
  endm
```

Example The following program acts as a stopwatch. When a character is typed, it sets the system clock to zero and begins to continuously display the time. When a second character is typed the system stops updating the time display.

```
time        db              "00:00:00.00",13,"$"
ten         db              10
            .
            .
func_2DH:   dir_console_io  0FFH        ;see Function 06H
            jz              func_2DH    ;wait for keystroke
            set_time        0,0,0,0     ;THIS FUNCTION
read_clock: get_time                    ;see Function 2CH
            convert ch,ten,time         ;see end of chapter
            convert cl,ten,time[3]      ;see end of chapter
            convert dh,ten,time[6]      ;see end of chapter
            convert dl,ten,time[9]      ;see end of chapter
            display time                ;see Function 09H
            dir_console_io 0FFH         ;THIS FUNCTION
            jz              read_clock  ;no char, keep updating
continue:   .
            .
```

2EH
Set/Reset Verify Flag

Call

AH = 2EH
AL
  00H = Do not verify
  01H = Verify

Return

None

Remarks

AL must be either 1 (verify after each disk write) or 0 (write without verifying). MS-DOS checks this flag each time it writes to a disk.

The flag is normally off; you may wish to turn it on when writing critical data to disk. Because disk errors are rare and verification slows writing, you will probably want to leave it off at other times.

Macro

```
verify macro switch
  mov   al,switch
  mov   ah,2EH
  int   21H
  endm
```

Example

The following program copies the contents of a single-sided disk in drive A: to the disk in drive B:, verifying each write. It uses a buffer of 32K bytes:

```
on          equ     1
off         equ     0
            .
            .
prompt      db      "Source in A, target in B",13,10
            db      "Any key to start. $"
start       dw      0
buffer      db      64 dup (512 dup(?)) ;64 sectors
```

2EH
Set/Reset Verify Flag

```
func_2EH:   display prompt                      ;see Function 09H
            read_kbd                            ;see Function 08H
            verify on                           ;THIS FUNCTION
            mov     cx,5                        ;copy 64 sectors
                                                ;5 times
copy:       push    cx                          ;save counter
            abs_disk_read 0,buffer,64,start     ;see Interrupt 25H
            abs_disk_write 1,buffer,64,start    ;see Interrupt 26H
            add     start,64                    ;do next 64 sectors
            pop     cx                          ;restore counter
            loop    copy                        ;do it again
            verify off                          ;THIS FUNCTION
```

2FH
Get Disk Transfer Address

Call AH = 2FH

Return ES:BX
Points to Disk Transfer Address

Macro

```
;
get_dta macro
  mov   2h,2fh
  int   21h
  endm
```

30H
Get DOS Version Number

Call

AH = 30H

Return

AL
  Major version number
AH
  Minor version number
BH
  OEM number
BL:CX
  User number (24 bits)

Remarks

On return, AL.AH will be the two-part version designation; i.e., for MS-DOS 1.28, AL would be 1 and AH would be 28. For pre-1.28 DOS AL = 0. Note that version 1.1 is the same as 1.10, not the same as 1.01.

Macro

```
;
get_version_num macro
  mov   ah,30h
  int   21h
  endm
```

31H
Keep Process

Call

AH = 31H
AL
  Exit code
DX
  Memory size, in paragraphs

Return

None

Remarks

This call terminates the current process and attempts to set the initial allocation block to a specific size in paragraphs. It will not free up any other allocation blocks belonging to that process. The exit code passed in AX is retrievable by the parent via Function 4DH.

This method is preferred over Interrupt 27H and has the advantage of allowing more than 64K to be kept.

Macro

```
;
keep_process macro exitcode,parasize
  mov   al,exitcode
  mov   dx,parasize
  mov   ah,31h
  int   21h
  endm
```

33H
<CTRL C> Check

Call

AH = 33H
AL
  Function
  00H = Request current state
  01H = Set state
DL (if setting state)
  00H = Off
  01H = On

Return

DL
  00H = Off
  01H = On

Remarks

MS-DOS ordinarily checks for a **CTRL C** on the controlling device only when doing function call operations 01H-0CH to that device. Function 33H allows the user to expand this checking to include any system call. For example, with the **CTRL C** trapping off, all disk I/O will proceed without interruption; with **CTRL C** trapping on, the **CTRL C** interrupt is given at the system call that initiates the disk operation.

Note

Programs that wish to use calls 06H or 07H to read **CTRL C**'s as data must ensure that the **CTRL C** check is off.

Error Returns

AL = FF

The function passed in AL was not in the range 0:1.

Macro

```
;
ctrl_c_check macro switch,val
  mov   dl,val
  mov   al,switch
  mov   ah,33h
  int   21h
  endm
```

35H
Get Interrupt Vector

Call

AH = 35H
AL
Interrupt number

Return

ES:BX
Pointer to interrupt routine

Remarks

This function returns the interrupt vector associated with an interrupt. Note that programs should never get an interrupt vector by reading the low memory vector table directly.

Macro

```
;
get_vector macro interrupt
  mov   al,interrupt
  mov   ah,35h
  int   21h
  endm
```

36H
Get Disk Free Space

Call

AH = 36H
DL
Drive (0 = Default, 1 = A, etc.)

Return

AX
FFFF if drive number is invalid; otherwise sectors per cluster
BX
Available clusters
CX
Bytes per sector
DX
Clusters per drive

Remarks

This function returns free space on a disk along with additional information about the disk.

Error Returns

AX = FFFF

The drive number given in DL was invalid.

Macro

```
;
get_disk_space macro drive
        mov   dl,drive
        mov   ah,36h
        int   21h
        endm
```

38H
Return Country-Dependent Information

Call

AH = 38H
DS:DX
  Pointer to 32-byte memory area
AL
  Function code.

Return

Carry set:
AX
  2 = file not found
Carry not set:
  DX:DS filled in with country data

Remarks

The value passed in AL is either 0 (for current country) or a country code. Country codes are typically the international telephone prefix code for the country.

If DX = -1, then the call sets the current country (as returned by the AL = 0 call) to the country code in AL. If the country code is not found, the current country is not changed.

Note

Applications must assume 32 bytes of information. This means the buffer pointed to by DS:DX must be able to accommodate 32 bytes.

This function is fully supported only in versions of MS-DOS 2.01 and higher. It exists in MS-DOS 2.0, but is not fully implemented.

This function returns, in the block of memory pointed to by DS:DX, information pertinent to international applications. The contents of the block are shown in the following table.

WORD Date/time format
5 BYTE ASCIIZ string currency symbol
2 BYTE ASCIIZ string thousands separator
2 BYTE ASCIIZ string decimal separator
2 BYTE ASCIIZ string date separator
2 BYTE ASCIIZ string time separator
1 BYTE Bit field
1 BYTE Currency places
1 BYTE time format
DWORD Case Mapping call
2 BYTE ASCIIZ string data list separator

The format of most of the entries is ASCIIZ (a NUL terminated ASCII string), but a fixed size is allocated for each field for easy indexing into the table.

The date/time format (see table) has the following values:

0 — USA standard	h:m:s m/d/y
1 — Europe standard	h:m:s d/m/y
2 — Japan standard	y/m/d h:m:s

The bit field contains 8 bit values. Any bit not currently defined must be assumed to have a random value.

Bit 0 = 0 If currency symbol precedes the currency amount.

= 1 If currency symbol comes after the currency amount.

Bit 1 = 0 If the currency symbol is directly adjacent to the currency amount.

= 1 If there is a space between the currency symbol and the amount.

The time format has the following values:

0 - 12 hour time
1 - 24 hour time

The currency places field indicates the number of places which appear after the decimal point on currency amounts.

The Case Mapping call is a FAR procedure which will perform country specific lower-to-uppercase mapping on character values from 80H to FFH. It is called with the character to be mapped in AL. It returns the correct upper case code for that character, if any, in AL. AL and the FLAGS are the only registers altered. It is allowable to pass this routine codes below 80H; however nothing is done to characters in this range. In the case where there is no mapping, AL is not altered.

Error Returns

AX
2 = file not found

The country passed in AL was not found (no table for specified country).

38H
Return Country-Dependent Information

Macro

```
;
get_country_info macro buffer, country
   mov   dx,offset buffer
   mov   al,country              ;country = 0
   mov   ah,38h
   int   21h
   endm
```

39H
Create Sub-Directory

Call

AH = 39H
DS:DX
  Pointer to path name

Return

Carry set:
AX
  3 = path not found
  5 = access denied
Carry not set:
  No error

Remarks

Given a pointer to an ASCIIZ name, this function creates a new directory entry at the end.

Error Returns

AX
  3 = path not found

    The path specified was invalid or not found.

  5 = access denied

    The directory could not be created (no room in parent directory), the directory/file already existed or a device name was specified.

Macro

```
;
mkdir macro name
  mov   dx,offset name
  mov   ah,39h
  int   21h
  endm
```

3AH
Remove a Directory

Call

AH = 3AH
DS:DX
  Pointer to path name

Return

Carry set:
AX
  3 = path not found
  5 = access denied
 16 = current directory
Carry not set:
  No error

Remarks

Function 3AH is given an ASCIIZ name of a directory. That directory is removed from its parent directory.

Error Returns

AX
  3 = path not found

  The path specified was invalid or not found.

  5 = access denied

  The path specified was not empty, not a directory, the root directory, or contained invalid information.

 16 = current directory

  The path specified was the current directory on a drive.

Macro

```
;
rmdir macro name
  mov   dx,offset name
  mov   ah,3ah
  int   21h
  endm
```

3BH
Change the Current Directory

Call

AH = 3BH
DS:DX
  Pointer to path name

Return

Carry set:
AX
  3 = path not found
Carry not set:
  No error

Remarks

Function 3BH is given the ASCIIZ name of the directory which is to become the current directory. If any member of the specified pathname does not exist, then the current directory is unchanged. Otherwise, the current directory is set to the string.

Error Returns

AX
  3 = path not found

  The path specified in DS:DX either indicated a file or the path was invalid.

Macro

```
;
chdir macro name
  mov   dx,offset name
  mov   ah,3bh
  int   21h
  endm
```

3CH
Create a File

Call

AH = 3CH
DS:DX
  Pointer to path name
CX
  File attribute

Return

Carry set:
AX
  3 = path not found
  4 = too many open files
  5 = access denied
Carry not set:
  AX is handle number

Remarks

Function 3CH creates a new file or truncates an old file to zero length in preparation for writing. DS:DX must point to an ASCIIZ path to the file. If the file did not exist, then the file is created in the appropriate directory and the file is given the attribute found in CX. The given attribute byte is placed at offset 0BH in the file's directory entry. See the section on "Diskette Directory" in chapter 5 for details about the attribute byte. The file handle returned has been opened for read/write access.

Error Returns

AX

3 = path not found

The path specified was invalid.

4 = too many open files

5 = access denied

The attributes specified in CX contained one that could not be created (directory, volume ID), a file already existed with a more inclusive set of attributes, or a direc-tory existed with the same name.

The file was created with the specified attrib-utes, but there were no free handles available for the process, or the internal system tables were full.

Macro

```
;
create_file macro name,attrib
  mov   dx,offset name
  mov   cx,attrib
  mov   ah,3ch
  int   21h
  endm
```

3DH
Open a File Handle

Call

AH = 3DH
AL
  Access
  0 = file opened for reading
  1 = file opened for writing
  2 = file opened for both
    reading and writing
DS:DX
  pointer to pathname

Return

Carry set:
AX
  2 = file not found
  4 = too many open files
  5 = access denied
 12 = invalid access
Carry not set
  AX is handle number

Remarks

Function 3DH associates a 16-bit handle with a file.

The following values are allowed:

ACCESS	Function
0	opened for reading
1	opened for writing
2	opened for both reading and writing.

DS:DX point to an ASCIIZ name of the file to be opened.

The read/write pointer is set at the first byte of the file and the record size of the file is 1 byte. The returned file handle must be used for subsequent I/O to the file.

Error Returns

AX

2 = file not found

The path specified was invalid or not found.

4 = too many open files

There were no free handles available in the current process or the internal system tables were full.

5 = access denied

The user attempted to open a directory or volume-id, or open a read-only file for writing.

12 = invalid access

The access specified in AL was not in the range 0:2.

Macro

```
;
open_handle macro name, access
        mov     dx,offset name
        mov     al,access
        mov     ah,3dh
        int     21h
        endm
```

3EH
Close a File Handle

Call

AH = 3EH
BX
File handle

Return

Carry set:
AX
6 = invalid handle
Carry not set:
No error

Remarks

If BX is passed a file handle (like that returned by Functions 3CH, 3DH, or 45H), Function 3EH closes the associated file. Internal buffers are flushed to disk.

Error Returns

AX
6 = invalid handle

The handle passed in BX was not currently open.

Macro

```
;
close_handle macro handle
        mov   bx,handle
        mov   ah,3eh
        int   21h
        endm
```

3FH
Read From File/Device

Call

AH = 3FH
DS:DX
  Pointer to buffer
CX
  Bytes to read
BX
  File handle

Return

Carry set:
AX
  5 = error set:
  6 = invalid handle
Carry not set:
  AX = number of bytes read

Remarks

Function 3FH transfers a specified number of bytes from a file into a buffer location. It is not guaranteed that the number of bytes requested will be read; for example, reading from the keyboard will read at most one line of text. If the returned value is zero, then the program has tried to read from the end of file.

All I/O is done using normalized pointers; no segment wraparound will occur.

Error Returns

AX
  5 = access denied

    The handle passed in BX was opened in a mode that did not allow reading.

  6 = invalid handle

    The handle passed in BX was not currently open.

3FH
Read From File/Device

Macro

```
;
read_from_handle macro buffer,bytes,handle
                 mov   dx,offset buffer
                 mov   cx,bytes
                 mov   bx,handle
                 mov   ah,3fh
                 int   21h
                 endm
```

40H
Write to a File or Device

Call

AH = 40H
DS:DX
  Pointer to buffer
CX
  Bytes to write
BX
  File handle

Return

Carry set:
AX
  5 = access denied
  6 = invalid handle
Carry not set:
AX = number of bytes written

Remarks

Function 40H transfers a specified number of bytes from a buffer into a file. It should be regarded as an error if the number of bytes written is not the same as the number requested.

The write system call with a count of zero (CX = 0) will set the file size to the current position. Allocation units are allocated or released as required.

All I/O is done using normalized pointers; no segment wraparound will occur.

Error Returns AX

5 = access denied

The handle was not opened in a mode that allowed writing.

6 = invalid handle

The handle passed in BX was not currently open.

Macro

```
;
write_to_handle macro buffer,bytes,handle
        mov   dx,offset buffer
        mov   cx, bytes
        mov   bx,handle
        mov   ah,40h
        int   21h
        endm
```

41H
Delete a Directory Entry

Call

AH = 41H
DS:DX
  Pointer to path name

Return

Carry set:
AX
  2 = file not found
  5 = access denied
Carry not set:
  No error

Remarks

Function 41H deletes the file named in the ASCIIZ string pointed to by DS:DX.

Error Returns

AX
 2 = file not found

  The path specified was invalid or not found.

 5 = access denied

  The path specified was a directory or read-only.

Macro

```
;
erase macro name
  mov   dx,offset name
  mov   ah,41h
  int   21h
  endm
```

42H
Move File Pointer

Call

AH = 42H
CX:DX
  Distance to move, in bytes
AL
  Method of moving:
  (see text)
BX
  File handle

Return

Carry set:
AX
  1 = invalid function
  6 = invalid handle
Carry not set:
  DX:AX = new pointer location

Remarks

Function 42H moves the read/write pointer according to one of the following methods:

Method	Function
0	the pointer is moved to offset bytes from the beginning of the file
1	the pointer is moved to the current location plus offset
2	the pointer is moved to the end of file plus offset

Offset should be regarded as a 32-bit integer with CX occupying the most significant 16 bits.

Error Returns

AX

1 = invalid function

The function passed in AL was not in the range 0:2.

6 = invalid handle

The handle passed in BX was not currently open.

Macro

```
;
move_pointer macro highword,lowword,switch,
                        handle
        mov     dx,lowword
        mov     cx,highword
        mov     al,switch
        mov     bx,handle
        mov     ah,42h
        int     21h
        endm
```

43H
Change Attributes

Call

AH = 43H
DS:DX
  Pointer to path name
AL
  Function
  00 Return in CX
  01 Set to CX
CX (if AL = 01)
  Attribute to be set

Return

Carry set:
AX
  1 = invalid function
  3 = path not found
  5 = access denied
Carry not set:
  CX attributes (if AL = 00)

Remarks

Given an ASCIIZ name pointed to by DS:DX, Function 42H will set/get the attributes of the file to those given in CX. See the section on "Diskette Directory" in chapter 5 for a description of the attribute byte.

A function code is passed in AL:

AL	Function
0	return the attributes of the file in CX
1	set the attributes of the file to those in CX

Error Returns

AX

1 = invalid function

The function passed in AL was not in the range 0:1.

3 = path not found

The path specified was invalid.

5 = access denied

The attributes specified in CX contained one that could not be changed (directory, volume ID).

Macro

```
;
change_attrib macro name,attrib,switch
        mov   dx,offset name
        mov   cx,attrib
        mov   al,switch
        mov   ah,43h
        int   21h
        endm
```

44H
I/O Control for Devices

Call

AH = 44H
BX
  Handle
BL
  Drive (for function codes 4 and 5;
  0 = default, 1 = A:, etc.)
DS:DX
  Data or buffer
CX
  Bytes to read or write
AL
  Function code; see text

Return

Carry set:
AX
  1 = invalid function
  5 = access denied
  6 = invalid handle
 13 = invalid data
Carry not set:
Function Code = 2,3,4,5
  AX = Count transferred
Function Code = 6,7
  AL
    00 = Not ready
    FF = Ready

Remarks

Function 44H sets or gets device information associated with an open handle, or sends/receives a control string to a device handle or device.

The inputs to AL are function numbers, for which there are returns. The function number values and functions are discussed below.

The following values are allowed in AL as function codes:

Call	Function
0	get device information (returned in DX)
1	set device information (as determined by DX)
2	read CX number of bytes into DS:DX from device control channel
3	write CX number of bytes from DS:DX to device control channel
4	read CX number of bytes into DS:DX from disk (drive number in BL)
5	write CX number of bytes from DS:DX to disk (drive number in BL)
6	get input status
7	get output status

This function can be used to get information about device channels. Calls can be made on regular files, but only calls 0,6 and 7 are defined in that case (AL = 0,6,7). All other calls return an invalid function error.

Calls 0,1: The bits of DX are defined as follows for calls AL = 0 and AL = 1. Note that the upper byte MUST be zero on a set call.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	C							I	E	R	S	I	I	I	I
e	T							S	O	A	P	S	S	S	S
s	R			Reserved				D	F	W	E	C	N	C	C
	L							E			C	L	U	O	I
								V			L	K	L	T	N

ISDEV = 1 if this channel is a device
= 0 if this channel is a disk file
(Bits 8-15 = 0 in this case)

If ISDEV = 1

EOF = 0 if End Of File on input
RAW = 1 if this device is in Raw mode
= 0 if this device is cooked
SPECL = 1 if this device is special
ISCLK = 1 if this device is the clock device
ISNUL = 1 if this device is the null device
ISCOT = 1 if this device is the console output
ISCIN = 1 if this device is the console input

CTRL = 0 if this device cannot do control strings via calls AL = 2 and AL = 3
CTRL = 1 if this device can process control strings via calls AL = 2 and AL = 3.
NOTE that this bit cannot be set.

If ISDEV = 0

EOF = 0 if channel has been written
Bits 0-5 are the block device number for the channel (0 = A:, 1 = B:, ...)

NOTE: Bits 15,8-13,4 are reserved and should not be altered.

Calls 2..5: These four calls allow arbitrary control strings to be sent or received from a device. The call syntax is the same as the read and write system calls, except for 4 and 5, which take a drive number in BL instead of a handle in BX.

An invalid function error is returned if the CTRL bit (see above) is 0.

An access denied error is returned by calls AL = 4,5 if the drive number is invalid.

Calls 6,7: These two calls allow the user to check if a file handle is ready for input or output. Status of handles open to a device is the intended use of these calls, but status of a handle open to a disk file is allowed, and is defined as follows:

For input:

- Always ready (AL = FF) until EOF reached, then always not ready (AL = 0) unless current position changed via Function Request 42H (LSEEK).

For output:

- Always ready (even if disk full).

The status is defined at the time the system is CALLED. On future versions, by the time control is returned to the user from the system, the status returned may NOT correctly reflect the true current state of the device or file.

Error Returns

AX

1 = invalid function

The function passed in AL was not in the range 0:7.

5 = access denied (calls AL = 4,5)

6 = invalid handle

The handle passed in BX was not currently open.

13 = invalid data

Macro

```
;
io_ctrl_dev macro handle,buffer,bytes,switch
  mov   bx,handle          ;or 8-bit drive number
  mov   dx,offset buffer
  mov   cx,bytes
  mov   al,switch
  mov   2h,44h
  int   21h
  endm
```

45H
Duplicate a File Handle

Call

AH = 45H
BX
File handle

Return

Carry set:
AX
4 = too many open files
6 = invalid handle
Carry not set:
AX = new file handle

Remarks

Function 45H takes an already opened file handle and returns a new handle that refers to the same file at the same position.

Error Returns

AX
4 = too many open files

There were no free handles available in the current process or the internal system tables were full.

6 = invalid handle

The handle passed in BX was not currently open.

Example

```
mov   bx,fh
mov   ah,45H
int   21H
;ax has the returned handle
```

46H
Force a Duplicate of a Handle

Call

AH = 46H
BX
  Existing file handle
CX
  New file handle

Return

Carry set:
AX
  4 = too many open files
  6 = invalid handle
Carry not set:
  No error

Remarks

Function 46H takes an already opened file handle and returns a new handle that refers to the same file at the same position. If there was already a file open on handle CX, it is closed first.

Error Returns

AX
  4 = too many open files

    The internal system tables were full.

  6 = invalid handle

    The handle passed in BX was not currently open.

Example

```
mov     bx,fh
mov     cx,newfh
mov     ah,46H
int     21H
```

47H
Return Name of Current Directory

Call

AH = 47H
DS:SI
  Pointer to 64-byte memory area
DL
  Drive number

Return

Carry set:
AX
  15 = invalid drive
Carry not set:
  No error

Remarks

Function 47H returns an ASCIIZ string giving the name of the current directory for a particular drive. The directory is root-relative and does not contain the drive specifier or leading path separator. The drive code passed in DL is 0 = default, 1 = A:, 2 = B:, etc.

Error Returns

AX
15 = invalid drive

The drive specified in DL was invalid.

Macro

```
;
duplicate_handle macro handle
  mov   bx,handle
  mov   ah,45h
  int   21h
  endm
```

48H
Allocate Memory

Call

AH = 48H
BX
Size of memory to be allocated in paragraphs

Return

Carry set:
AX
7 = arena trashed
8 = not enough memory
BX
Maximum size that could be allocated
Carry not set:
AX:0
Pointer to the allocated memory

Remarks

Function 48H returns a pointer to a free block of memory that has the requested size in paragraphs.

Error Returns

AX
7 = arena trashed

The internal consistency of the memory arena has been destroyed. This is due to a user program changing memory that does not belong to it, thus destroying the memory manager allocation marks.

8 = not enough memory

The largest available free block is smaller than that requested or there is no free block.

Macro

```
;
force_handle macro old,new
   mov   bx,old
   mov   cx,new
   mov   ah,46h
   int   21h
   endm
```

49H
Free Allocated Memory

Call

AH = 49H
ES
Segment address of memory area to be freed

Return

Carry set:
AX
7 = arena trashed
9 = invalid block
Carry not set:
No error

Remarks

Function 49H returns a piece of previously allocated memory to the system pool.

Error Returns

AX
7 = arena trashed

The internal consistency of the memory arena has been destroyed. This is due to a user program changing memory that does not belong to it, thus destroying the memory manager allocation marks.

9 = invalid block

The block passed in ES is not one allocated via Function Request 48H.

Macro

```
;
cur_dir_name macro buffer,drive
  mov   si,offset buffer
  mov   dl,drive
  mov   ah,47h
  int   21h
  endm
```

4AH
Modify Allocated Memory Blocks

Call

AH = 4AH
ES
  Segment address of memory area
BX
  Requested memory area size

Return

Carry set:
AX
  7 = arena trashed
  8 = not enough memory
  9 = invalid block
BX
  Maximum size possible
Carry not set:
  No error

Remarks

Function 4AH will attempt to grow/shrink an allocated block of memory.

Error Returns

AX
7 = arena trashed

The internal consistency of the memory arena has been destroyed. This is due to a user program changing memory that does not belong to it, thus destroying the memory manager allocation marks.

8 = not enough memory

There was not enough free memory after the specified block to satisfy the grow request.

9 = invalid block

The block passed in ES is not one allocated via this function.

4AH
Modify Allocated Memory Blocks

Macro

```
;
alloc_mem macro size
   mov   bx,size
   mov   ah,48h
   int   21h
   endm
```

4BH
Load and Execute a Program (EXEC)

Call

AH = 4BH
DS:DX
  Pointer to pathname
ES:BX
  Pointer to parameter block
AL
  00 = Load and execute program
  03 = Load program

Return

Carry set:
AX
  1 = invalid function
  2 = file not found
  8 = not enough memory
 10 = bad environment
 11 = bad format
Carry not set:
  No error

Remarks

This function allows a program to load another program into memory and optionally begin execution of it. DS:DX points to the ASCIIZ name of the file to be loaded. ES:BX points to a parameter block for the load.

A function code is passed in AL:

AL	Function
0	load and execute the program. A program header is established for the program and the terminate and CTRL C addresses are set to the instruction after the EXEC system call.
3	load (do not create) the program header, and do not begin execution. This is useful in loading program overlays.

For each value of AL, the block has the format shown in the following table.

AL = 0 - load/execute program

WORD segment address of environment.
DWORD pointer to command line at 80H of Program Segment Prefix
DWORD pointer to default FCB to be passed at 5CH of PSP
DWORD pointer to default FCB to be passed at 6CH of PSP

AL = 3 - load overlay

WORD segment address where file will be loaded.
WORD relocation factor to be applied to the image.

Note that all open files of a process are duplicated in the child process after an EXEC. This is extremely powerful; the parent process has control over the meanings of stdin, stdout, stderr, stdaux and stdprn. The parent could, for example, write a series of records to a file, open the file as standard input, open a listing file as standard output and then EXEC a sort program that takes its input from stdin and writes to stdout.

Also inherited (or passed from the parent) is an "environment." This is a block of text strings (less than 32K bytes total) that convey various configurations parameters. The format of the environment is as follows:

(paragraph boundary)

BYTE ASCIIZ string 1
BYTE ASCIIZ string 2
...
BYTE ASCIIZ string n
BYTE of zero

Typically the environment strings have the form:

parameter = value

For example, COMMAND.COM might pass its execution search path as:

PATH=A:\BIN;B:\BASIC\LIB

A zero value of the environment address causes the child process to inherit the parent's environment unchanged.

Error Returns

AX

1 = invalid function

The function passed in AL was not 0 or 3.

2 = file not found

The path specified was invalid or not found.

8 = not enough memory

There was not enough memory for the process to be created.

10 = bad environment

The environment was larger than 32Kb.

11 = bad format

The file pointed to by DS:DX was in .EXE format and contained information that was internally inconsistent.

Macro

```
;
free_memory macro address
  mov   ax,address
  mov   es,ax
  mov   ah,49h
  int   21h
  endm
```

4CH
Terminate a Process

Call

AH = 4CH
AL = Return code

Return

None

Remarks

Function 4CH terminates the current process and transfers control to the invoking process. In addition, a return code may be sent. All files open at the time are closed.

This method is preferred over all others (Interrupt 20H, JMP 0) and has the advantage that CS:0 does not have to point to the Program Header Prefix.

Macro

```
;
modify_memory macro address,size
  mov   ax,address
  mov   es,ax
  mov   bx,size
  mov   ah,4ah
  int   21h
  endm
```

4DH
Retrieve the Return Code of a Child

Call

AH = 4DH

Return

AX
Exit code

Remarks

Function 4DH returns the Exit code specified by a child process. It returns this Exit code only once. The low byte of this code is that sent by the Exit routine. The high byte is one of the following:

0 - Terminate/abort
1 - CTRL C
2 - Hard error
3 - Terminate and stay resident

Macro

```
;
exec macro path,param,switch
  mov   dx,offset path
  mov   bx,offset param
  mov   al,switch
  mov   ah,4bh
  int   21h
  endm
```

4EH
Find Match File

Call

AH = 4EH
DS:DX
  Pointer to pathname
CX
  Search attributes

Return

Carry set:
AX
  2 = file not found
  18 = no more files
Carry not set:
  No error

Remarks

Function 4EH takes a pathname with wild card characters in the last component (passed in an ASCIIZ string pointed to by DS:DX) along with a set of attributes (passed in CX) and attempts to find all files that match the pathname and have a subset of the required attributes. A datablock at the current DTA is written that contains information in the following form:

```
find_buf_reserved DB 21 DUP (?); Reserved*
find_buf_attr     DB ? ;attribute found
find_buf_time     DW ? ;time
find_buf_date     DW ? ;date
find_buf_size_l   DW ? ;low(size)
find_buf_size_h   DW ? ;high(size)
find_buf_pname    DB 13 DUP (?) ;packed name
find_buf ENDS
```

*Reserved for MS-DOS internal use on subsequent find_nexts

To obtain the subsequent matches of the pathname, see the description of Function 4FH.

Error Returns

AX

2 = file not found

The path specified in DS:DX was an invalid path.

18 = no more files

There were no files matching this specification.

Macro

```
;
terminate_process macro code
  mov   al,code
  mov   ah,4ch
  int   21h
  endm
```

4FH
Step Through a Directory Matching Files

Call

AH = 4FH

Return

Carry set:
AX
  18 = no more files
Carry not set:
  No error

Remarks

The current DTA address must point at a block returned by Function 4EH (see Function 4EH).

Error Returns

AX
18 = no more files

  There are no more files matching this pattern.

Macro

```
;
retrieve_code macro
  mov   ah,4dh
  int   21h
  endm
```

54H
Return Current Setting of Verify After Write Flag

Call

AH = 54H

Return

AL
Current verify flag value

Remarks

The current value of the verify flag is returned in AL.

Macro

```
;
find_match macro name,attrib
  mov   dx,offset name
  mov   cx,attrib
  mov   ah,4eh
  int   21h
  endm
```

56H
Move a Directory Entry

Call

AH = 56H
DS:DX
  Pointer to pathname of existing file
ES:DI
  Pointer to new pathname

Return

Carry set:
AX
  2 = file not found
  5 = access denied
  17 = not same device
Carry not set:
  No error

Remarks

Function 56H attempts to rename a file into another path. The paths must be on the same device.

Error Returns

AX

2 = file not found

The file name specifed by DS:DX was not found.

5 = access denied

The path specified in DS:DX was a directory or the file specified by ES:DI already exists or the destination directory entry could not be created.

17 = not same device

The source and destination are on different drives.

56H
Move a Directory Entry

Macro

```
;
step_match macro
  mov   ah,4fh
  int   21h
  endm
```

57H
Get/Set Date/Time of File

Call

AH = 57H
AL
  00 = get date and time
  01 = set date and time
BX
  File handle
CX (if AL = 01)
  Time to be set
DX (if AL = 01)
  Date to be set

Return

Carry set:
AX
  1 = invalid function
  6 = invalid handle
Carry not set:
No error
CX/DX set if function 0

Remarks

Function 57H returns or sets the last-write time for a handle. These times are not recorded until the file is closed.

A function code is passed in AL:

AL	Function
0	return the time/date of the handle in CX/DX
1	set the time/date of the handle to CX/DX

The format for the date and time is the same as the date and time fields for a directory entry, except that the individual bytes in each word are reversed. The high order portion of the time is in CL, and the high order portion of the date is in DL.

57H
Get/Set Date/Time of File

Error Returns

AX

1 = invalid function

The function passed in AL was not in the range 0:1.

6 = invalid handle

The handle passed in BX was not currently open.

Macro

```
;
check_verify_flag macro
        mov   ah,54h
        int   21h
        endm
```

Macro

Note These macro definitions apply to system call examples 00H through 57H.

```
;
;*******************
; Interrupts
;*******************
;
                                        ;ABS_DISK_READ
abs_disk_read macro disk, buffer, num_sectors, first_sector
   mov     al,disk
   mov     bx,offset buffer
   mov     cx,num_sectors
   mov     dx,first_sector
   int     25H                          ;interrupt 25H
   popf
   endm
;
                                        ;ABS_DISK_WRITE
abs_disk_write macro disk, buffer, num_sectors, first_sector
   mov     al,disk
   mov     bx,offset buffer
   mov     cx,num_sectors
   mov     dx,first_sector
   int     26H                          ;interrupt 26H
   popf
   endm
;
stay_resident macro last_instruc        ;STAY_RESIDENT
   mov     dx,offset last_instruc
   inc     dx
   int     27H                          ;interrupt 27H
   endm

;*******************
; Functions
;*******************
;
```

```
;
read_kbd_and_echo macro          ;READ_KBD_AND_ECHO
   mov     ah,1                  ;function 1
   int     21H
   endm
;
display_char macro character     ;DISPLAY_CHAR
   mov     dl,character
   mov     ah,2                  ;function 2
   int     21H
   endm
;
aux_input macro                  ;AUX_INPUT
   mov     ah,3                  ;function 3
   int     21H
   endm
;
aux_output macro                 ;AUX_OUTPUT
   mov     ah,4                  ;function 4
   int     21H
   endm
;
print_char macro character       ;PRINT_CHAR
   mov     dl,character
   mov     ah,5                  ;function 5
   int     21H
   endm
dir_console_io macro switch      ;DIR_CONSOLE_IO
   mov     dl,switch
   mov     ah,6                  ;function 6
   int     21H
   endm
;
dir_console_input macro          ;DIR_CONSOLE_INPUT
   mov     ah,7                  ;function 7
   int     21H
   endm
;
read_kbd macro                   ;READ_KBD
   mov     ah,8                  ;function 8
   int     21H
   endm
```

```
;
display macro string                    ;DISPLAY
    mov     dx,offset string
    mov     ah,9                        ;function 9
    int     21H
    endm
;
get_string macro limit,string           ;GET_STRING
    mov     string,limit
    mov     dx,offset string
    mov     ah,0AH                      ;function 0AH
    int     21H
    endm
;
check_kbd_status macro                  ;CHECK_KBD_STATUS
    mov     ah,0BH                      ;function 0BH
    int     21H
    endm
;
flush_and_read_kbd macro switch ;FLUSH_AND_READ_KBD
    mov     al,switch
    mov     ah,0CH                      ;function 0CH
    int     21H
    endm
;
reset_disk macro                        ;RESET DISK
    mov     ah,0DH                      ;function 0DH
    int     21H
    endm
;
select_disk macro disk                  ;SELECT_DISK
    mov     dl,disk[-65]
    mov     ah,0EH                      ;function 0EH
    int     21H
    endm
;
open macro fcb                          ;OPEN
    mov     dx,offset fcb
    mov     ah,0FH                      ;function 0FH
    int     21H
    endm
```

```
;
close macro fcb                         ;CLOSE
    mov     dx,offset fcb
    mov     ah,10H                      ;function 10H
    int     21H
    endm
;
search_first macro fcb                  ;SEARCH_FIRST
    mov     dx,offset fcb
    mov     ah,11H                      ;function 11H
    int     21H
    endm
;
search_next macro fcb                   ;SEARCH_NEXT
    mov     dx,offset fcb
    mov     ah,12H                      ;function 12H
    int     21H
    endm
;
delete macro fcb                        ;DELETE
    mov     dx,offset fcb
    mov     ah,13H                      ;function 13H
    int     21H
    endm
;
read_seq macro fcb                      ;READ_SEQ
    mov     dx,offset fcb
    mov     ah,14H                      ;function 14H
    int     21H
    endm
;
write_seq macro fcb                     ;WRITE_SEQ
    mov     dx,offset fcb
    mov     ah,15H                      ;function 15H
    int     21H
    endm
;
create macro fcb                        ;CREATE
    mov     dx,offset fcb
    mov     ah,16H                      ;function 16H
    int     21H
    endm
```

```
;
rename macfo special_fcb          ;RENAME
   mov     dx,offset special_fcb
   mov     ah,17H                 ;function 17H
   int     21H
   endm
;
current_disk macro                ;CURRENT_DISK
   mov     ah,19H                 ;function 19H
   int     21H
   endm
;
set_dta macro buffer              ;SET_DTA
   mov     dx,offset buffer
   mov     ah,1AH                 ;function 1AH
   int     21H
   endm
;
read_ran macro fcb                ;READ_RAN
   mov     dx,offset fcb
   mov     ah,21H                 ;function 21H
   int     21H
   endm
;
write_ran macro fcb               ;WRITE_RAN
   mov     dx,offset fcb
   mov     ah,22H                 ;function 22H
   int     21H
   endm
;
file_size macro fcb               ;FILE_SIZE
   mov     dx,offset fcb
   mov     ah,23H                 ;function 23H
   int     21H
   endm
```

```
;
set_relative_record macro fcb          ;SET_RELATIVE_RECORD
   mov    dx,offset fcb
   mov    ah,24H                       ;function 24H
   int    21H
   endm
;
set_vector macro interrupt,seg_addr,off_addr ;SET_VECTOR
   push   ds
   mov    ax,seg_addr
   mov    ds,ax
   mov    dx,off_addr
   mov    al,interrupt
   mov    ah,25H                       ;function 25H
   int    21H
   endm
;
ran_block_read macro fcb,count,rec_size;RAN_BLOCK_READ
   mov    dx,offset fcb
   mov    cx,count
   mov    word ptr fcb[14], rec_size
   mov    ah,27H                       ;function 27H
   int    21H
   endm
;
ran_block_write macro fcb,count,rec_size;RAN_BLOCK_WRITE
   mov    dx,offset fcb
   mov    cx,count
   mov    word ptr fcb[14], rec_size
   mov    ah,28H                       ;function 28H
   int    21H
   endm
```

```
;
parse macro filename, fcb              ;PARSE
   mov     si,offset filename
   mov     di,offset fcb
   push    es
   push    ds
   pop     es
   mov     al,0FH
   mov     ah,29H                      ;function 29H
   int     21H
   pop     es
   endm
;
get_date macro                         ;GET_DATE
   mov     ah,2AH                      ;function 2AH
   int     21H
   endm
;
set_date macro year,month,day          ;SET_DATE
   mov     cx,year
   mov     dh,month
   mov     dl,day
   mov     ah,2BH                      ;function 2BH
   int     21H
   endm
;
get_time macro                         ;GET_TIME
   mov     ah,2CH                      ;function 2CH
   int     21H
   endm
;                                      ;SET_TIME
set_time macro hour, minutes, seconds, hundredths
   mov     ch,hour
   mov     cl,minutes
   mov     dh,seconds
   mov     dl,hundredths
   mov     ah,2DH                      ;function 2DH
   int     21H
   endm
```

```
;
verify macro switch                          ;VERIFY
   mov          al,switch
   mov          ah,2EH                       ;function 2EH
   int          21H
   endm
;
get_dta macro                                ;GET_DTA
   mov          ah,2FH
   int          21H
   endm
;
get_version_num macro                        ;GET_VERSION_NUM
   mov          ah,30H
   int          21H
   endm
;
keep_process macro exitcode,parasize         ;KEEP_PROCESS
   mov          al,exitcode
   mov          dx,parasize
   mov          ah,31H
   int          21H
   endm
;
ctrl_c_check macro switch,val                ;CTRL_C_CHECK
   mov          dl,val
   mov          al,switch
   mov          ah,33H
   int          21H
   endm
;
get_vector macro interrupt                   ;GET_VECTOR
   mov          al,interrupt
   mov          ah,35H
   int          21H
   endm
;
get_disk_space macro drive                   ;GET_DISK_SPACE
   mov          dl,drive
   mov          2h,36H
   int          21H
   endm
```

```
;
get_country_info macro buffer,country   ;GET_COUNTRY_INFO
    mov         dx,offset buffer
    mov         al,country              ;country = 0
    mov         ah,38H
    int         21H
    endm
;
mkdir macro name                        ;MKDIR
    mov         dx,offset name
    mov         ah,39H
    int         21H
    endm
;
rmdir macro name                        ;RMDIR
    mov         dx,offset name
    mov         ah,3AH
    int         21H
    endm
;
chdir macro name                        ;CHDIR
    mov         dx,offset name
    mov         ah,3BH
    int         21H
    endm
;
create_file macro name,attrib           ;CREATE_FILE
    mov         dx,offset name
    mov         cx,attrib
    mov         ah,3CH
    int         21H
    endm
;
open_handle macro name,access           ;OPEN_HANDLE
    mov         dx,offset name
    mov         al,access
    mov         2h,3DH
    int         21H
    endm
```

```
;
close_handle macro handle                    ;CLOSE_HANDLE
   mov         bx,handle
   mov         2h,3EH
   int         21H
   endm
;
read_from_handle macro buffer,bytes,handle
                                             ;READ_FROM_HANDLE
   mov         dx,offset buffer
   mov         cx,bytes
   mov         bx,handle
   mov         ah,3FH
   int         21H
   endm
;
write_to_handle macro buffer,bytes,handle
                                             ;WRITE_TO_HANDLE
   mov         dx,offset buffer
   mov         cx,bytes
   mov         bx,handle
   mov         ah,40H
   int         21H
   endm
;
erase macro name                             ;ERASE
   mov         dx,offset name
   mov         ah,41H
   int         21H
   endm
;
move_pointer macro highword,lowword,switch,handle
                                             ;MOVE_POINTER
   mov         dx,lowword
   mov         cx,highword
   mov         al,switch
   mov         bx,handle
   mov         ah,42H
   int         21H
   endm
```

```
;
change_attrib macro name,attrib,switch  ;CHANGE_ATTRIB
    mov         dx,offset name
    mov         cx,attrib
    mov         al,switch
    mov         2h,43H
    int         21H
    endm
;
io_ctrl_dev macro handle,buffer,bytes,switch
                                        ;IO_CTRL_DEV
    mov         bx,handle               ;or 8-bit drive number
    mov         dx,offset buffer
    mov         cx,bytes
    mov         al,switch
    mov         ah,44H
    int         21H
    endm
;
duplicate_handle macro handle           ;DUPLICATE_HANDLE
    mov         bx,handle
    mov         ah,45H
    int         21H
    endm
;
force_handle macro old,new              ;FORCE_HANDLE
    mov         bx,old
    mov         cx,new
    mov         ah,46H
    int         21H
    endm
;
cur_dir_name macro buffer,drive         ;CUR_DIR_NAME
    mov         si,offset buffer
    mov         dl,drive
    mov         ah,47H
    int         21H
    endm
```

```
;
alloc_mem macro size                    ;ALLOC_MEM
   mov          bx,size
   mov          ah,48H
   int          21H
   endm
;
free_memory macro address               ;FREE_MEMORY
   mov          ax,address
   mov          es,ax
   mov          ah,49H
   int          21H
   endm
;
modify_memory macro address,size        ;MODIFY_MEMORY
   mov          ax,address
   mov          es,ax
   mov          bx,size
   mov          ah,4AH
   int          21H
   endm
;
exec macro path,param,switch            ;EXEC
   mov          dx,offset path
   mov          bx,offset param
   mov          al,switch
   mov          ah,4BH
   int          21H
   endm
;
terminate_process macro code            ;TERMINATE_PROCESS
   mov          al,code
   mov          ah,4CH
   int          21H
   endm
;
retrieve_code macro                     ;RETRIEVE_CODE
   mov          ah,4DH
   int          21H
   endm
```

```
;
find_match macro name,attrib                ;FIND_MATCH
    mov         dx,offset name
    mov         cx,attrib
    mov         ah,4EH
    int         21H
    endm
;
step_match macro                            ;STEP_MATCH
    mov         ah,4FH
    int         21H
    endm
;
check_verify_flag macro                     ;CHECK_VERIFY_FLAG
    mov         ah,54H
    int         21H
    endm
;
rename macro old,new                        ;RENAME
    mov         dx,offset old
    mov         di,offset new
    mov         ah,56H
    int         21H
    endm
;
date_time_of_file macro switch,handle,date,time
                                            ;DATE_TIME_OF_FILE
    mov         al,switch
    mov         bx,handle
    mov         cx,time
    mov         dx,date
    mov         ah,57H
    int         21H
    endm
```

```
;*******************
; General
;*******************
move_string macro source, destination, num_bytes
                                                ;MOVE_STRING
    push        es
    mov         ax,ds
    mov         es,ax
    mov         si,offset source
    mov         di,offset destination
    mov         cx,num_bytes
rep movs        es:destination,source
    pop         es
    endm
;
;
convert macro value, base, destination ;CONVERT
        local   table,start
        jmp     start
table   db      "0123456789ABCDEF"
start:  mov     al,value
        xor     ah,ah
        xor     bx,bx
        div     base
        mov     bl,al
        mov     al,cs: table[bx]
        mov     destination,al
        mov     bl,ah
        mov     al,cs: table[bx]
        mov     destination[1],al
        endm
```

```
;
convert_to_binary macro string, number, value
                                        ;CONVERT_TO_BINARY
            local   ten,start,calc,mult,no_mult
            jmp     start
ten         db      10
start:      mov     value,0
            xor     cx,cx
            mov     cl,number
            xor     si,si
calc:       xor     ax,ax
            mov     al,string[si]
            sub     al,48
            cmp     cx,2
            jl      no_mult
            push    cx
            dec     cx
mult:       mul     cs:ten
            loop    mult
            pop     cx
no_mult:    add     value,ax
            inc     si
            loop    calc
            endm
;
convert_date macro dir_entry
            mov     dx,word ptr dir_entry[25]
            mov     cl,5
            shr     dl,cl
            mov     dh,dir_entry[25]
            and     dh,1fh
            xor     cx,cx
            mov     cl,dir_entry[26]
shr         cl,1
add         cx,1980
endm
;
```

8 ROM BIOS Service Routines

- Overview
- Conventions
- Interrupt Vector Listing
- Video Control
- Diskette Services
- Communications Services
- Keyboard Handling
- Printer Routines
- Miscellaneous ROM BIOS Services
- Bypassing the BIOS
- CONFIG.SYS
- ROM BIOS Listing
- ROM BIOS Change List
- Notes on Enhancements in ROM BIOS 1.21
- ROM Revision 1.1 to ROM Revision 1.21 Source File Differences

Overview

This chapter describes the ROM BIOS service routines that are provided to perform the more low-level functions that you may need in your assembly language programs. Because these are low-level routines, they provide more direct access to the hardware than the DOS routines. However, they do not provide some of the protection and conveniences that the DOS routines give. Be sure to check the chapter on "System Calls" to make your choice between similar DOS and BIOS calls.

Conventions

Access to the BIOS service routines is through the 8086 software interrupts. The routines are called with conventions that are very similar to the conventions for calling DOS routines.

To issue a BIOS interrupt, use the Interrupt statement to select the desired interrupt:

```
INT 11H
```

Some interrupts, like Interrupt 11H (Equipment List), perform only one function. Others, like Interrupt 13H (Diskette Services), have several sub-functions that you can call. To select a sub-function, move the number of the sub-function into the AH register.

This chapter describes the register usage for each of the BIOS service routines. It is usually wise to save all important registers before calling a BIOS service routine.

Interrupt Vector List

Interrupt Number (Hex)	Name
5	Print Screen
10	Video
11	Equipment Check
12	Determine Memory Size
13	Diskette
14	Communications
16	Keyboard
17	Printer
19	Bootstrap

Video Control

Introduction

The video controller on the standard AT&T Personal Computer 6300 supports both monochrome and color monitors and produces text or graphics for both color and monochrome. Interrupt 10H has the BIOS services to support all of these modes. This section describes the details that pertain to each major type of video access.

Monochrome Text Mode

The monochrome text modes are mode 0 — 40x25 characters and mode 2 — 80x25 characters. The monochrome text mode uses 32K starting at B8000H. For each screen position, there are two bytes in memory. The first byte is the ASCII code for the character to be displayed. The second byte is the "attribute" that specifies how the character is to be displayed. This attribute byte controls brightness, underlining, and blinking.

The low order nybble of the attribute byte governs the character being displayed according to the following table:

Value	Meaning
0	Character is black
1	Character is normal (white) intensity, underlined
7	Character is normal (white) intensity
F	Character is high intensity white

Any other value for the low nybble selects a particular gray character intensity.

The high order nybble of the attribute byte governs the character background and blinking. A displayed character will blink if the high order bit of its attribute byte is set. The remaining

three bits select the gray scale of the background — again, 000 is black and 111 is white. Note that inverse video can be obtained by forcing a black character on a white background, i.e. an attribute byte of 70H.

The first two bytes in the display memory control the character in the top left corner of the screen. The next two bytes control the character in the top row, in the second column position, and so on.

At the end of each line, the display memory returns to the first column of the next line. There are no gaps in the display storage, and no boundaries between one line and the next.

Eight pages of memory are used to build up to eight separate screens. Only one page is active at any time, but you can switch the active page number and thereby display screens very rapidly.

The display pages are numbered 0 - 7 for 40x25 mode and 0 - 3 for 80x25 mode. Page 0 starts at memory location B8000H. For 40 column mode, the pages occur at 2K intervals; for 80 column, at 4K intervals. A total of 32K of memory is used.

Color Text Modes

The color text modes are mode 1 — 40x25 color mode and mode 3 — 80x25 color.

Memory usage for the color text modes is similar to the method used for monochrome text. Two bytes of memory are used for each character position: the first is the ASCII code for the character and the second is the attribute byte. The attribute byte specifies blinking, brightness, and color.

The attribute bytes in color text mode operate much the same way as they do in monochrome text modes with two major differences:

- Instead of bits 0-3 and 4-7 selecting the gray scale of the foreground and background, they select foreground and background colors according to the following chart:

Bit				
d	c	b	a	Color
0	0	0	0	Black
0	0	0	1	Blue
0	0	1	0	Green
0	0	1	1	Cyan
0	1	0	0	Red
0	1	0	1	Magenta
0	1	1	0	Brown
0	1	1	1	White
1	0	0	0	Grey
1	0	0	1	Lt. blue
1	0	1	0	Lt. green
1	0	1	1	Lt. cyan
1	1	0	0	Lt. red
1	1	0	1	Lt. magenta
1	1	1	0	Yellow
1	1	1	1	High intensity white

Note that since background color is determined by a three-bit value, only the first eight colors apply to that field.

- There is no underline attribute possible in color mode. As can be seen by the chart above, attribute settings that produce an underline in monochrome mode produce a blue character in color mode.

  The display memory maps to the character positions exactly as it does in monochrome text mode.

Color Graphics Mode

There is one color graphics mode: mode 4 — medium resolution (320x200) color graphics. For any color display, you can use up to four colors. You select from one of two "palettes," each of which provides three colors. You select a "background" color to be used as the fourth color.

Palette 0 contains green, yellow, and red. Palette 1 contains cyan (light blue), magenta, and white.

320 pixels can be displayed on each of 200 lines. Each line takes 80 bytes or 640 bits of display memory. Each color pixel use two bits of memory. Since two bits give you four possible combinations, for each pixel you specify either the background color or one of the three colors in the current palette. The leftmost pixels are represented by the high order bits in the byte.

Display memory for color graphics mode starts at location B8000H and is divided into four 8K blocks. Starting at B8000, the first 8000D bytes contain the pixel data for the even scan lines on page zero. That is, the first 80 bytes describe line 0, the next 80H describe line 2, and so on through line 198. The odd lines are described in the 8K block starting at BA000. The same pattern is repeated for page one in the next 16K block, with the even lines starting at BC000 and the odd lines starting at BE000.

High Resolution Monochrome Graphics

Memory for high resolution 640x200 monochrome graphics is handled similarly to 320x200 color graphics. The only difference is that instead of memory containing two bits of color information per pixel, each pixel can only be on or off and is thus represented by one bit. In this way eight pixels can be represented in a byte instead of four, so that a scan line takes as many bytes as in color graphics mode even though it contains twice as many pixels. As in color graphics mode, the leftmost pixels are represented in the high order bits of each byte. Line mapping is exactly as described above for color graphics mode. In high resolution monochrome graphics the background color is always black and the foreground color is chosen by bits 0-3 of the color select register.

Super High Resolution Monochrome Graphics

Super high resolution 640x400 monochrome graphics mode maps one bit per pixel with the leftmost pixel represented at the high end of the byte, just like high resolution 640x200 mode. Also like high resolution mode, super high mode maps onto a black background with a foreground color chosen by the color select register. The memory mapping, however, takes up all 32K of display memory for a single page. Memory is broken up into four 8K segments, with each segment containing the data for every fourth scan line. Thus display memory looks like this:

Memory location	Contains pixels for line numbers
B8000	0, 4, 8, ... 396
B9F3F	Not used.
BA000	1, 5, 9, ... 397
BBF3F	Not used.
BC000	2, 6, 10, ... 398
BDF3F	Not used.
BE000	3, 7, 11, ... 399
BFF3F	Not used.

Set Mode and Clear Screen

Input:
(AH) = 0
(AL) contains the CRT mode value

Text Modes:
(AL) = 0 40x25 monochrome
(AL) = 1 40x25 color
(AL) = 2 80x25 monochrome
(AL) = 3 80x25 color

Graphics modes:
(AL) = 4 320x200(medium resolution), color
(AL) = 5 320x200(medium resolution), monochrome
(AL) = 6 640x200 black/white (high resolution)
(AL) = 40H graphics 640x400 monochrome super high resolution
(AL) = 48H graphics 640x400 monochrome tiny text (80x50 text)

Set Cursor Type

Input:
(AH) = 1
Low order 5 bits of (CH) = start line for cursor.

Note
Do not set the high bits of CH: unpredictable results will occur.

Low order 5 bits of (CL) = end line for cursor.

Set Cursor Position

Input:
(AH) = 2
(DH,DL) = Row,Column (Position 0,0 is upper left.)
(BH) = page number (must be 0 for super-res graphics mode.)

Read Cursor Position

Input:
(AH) = 3
(BH) = page number (must be 0 for super-res graphics mode.)

Output:
(DH,DL) = row, column of current cursor
(CH,CL) = current cursor start and end lines

Read Light Pen Position

Input:
(AH) = 4

Output:
(AH) = 0 light pen switch not triggered
(AH) = 1 valid light pen value obtained:
(DH,DL) = row, column of character light pen position
(CH) = raster line (0-199)
(BX) = pixel column (0-319 for medium resolution, 0-639 for high resolution.)

Select Active Page Number

Valid only for modes (0 - 6)

Input:
(AH) = 5
(AL) = 0-15 for modes 0, 1
= 0-7 for modes 2, 3
= 0-1 for modes 4, 6

Scroll Active Page up

Input:
(AH) = 6
(AL) = number of lines blanked at bottom of window by scrolling up. AL = 0 means blank entire window.
(CH,CL) = row, column of upper left corner of scroll
(DH,DL) = row, column of lower right corner of scroll
(BH) = attribute to be used on blank line(s).

Scroll Active Page Down

Input:
(AH) = 7
(AL) = number of lines blanked at top of window by scrolling down. AL = 0 means blank entire window.
(CH,CL) = row, column of upper left corner of scroll
(DH,DL) = row, column of lower right corner of scroll
(BH) = attribute to be used on blank line(s).

Character Handling

The next three video services perform character input/output for the CRT. If your program displays characters to the screen while in graphics modes, the characters are formed from a character generator image that is maintained in the ROM. However, only the first 128 characters are encoded there. If you want to create your own characters, either for the purposes of doing character graphics or implementing a foreign language alphabet, you must set up a table of code points for 128 new characters and initialize the pointer at interrupt 1F (address 0007CH) to point to the 1K table. These codes can then be accessed by referring to characters 128-255.

When you write characters to the screen in text mode, if you send more characters to be written than will fit on one line, the extra characters automatically wrap around to the beginning to the next line. In graphics mode, the character handling routines only produce correct results for characters contained on the same row (continuation to succeeding lines does not work.)

Read Attribute or Character at Current Cursor Position

Input:
(AH) = 8
(BH) = current display page

Output:
(AL) = character read
(AH) = attribute of character read

Write Attribute and Character at Current Cursor Position

Input:
(AH) = 9
(BH) = current display page
(CX) = count of characters to write
(AL) = character to write
(BL) = attribute of character (if text mode)
= color of character (if graphics mode)

Note
If bit 7 of BL = 1, the color value is exclusive OR'd with the current contents of the dot.

Write Character Only at Current Cursor Position

Input:
(AH) = OAH
(BH) = current display page
(CX) = count of characters to write
(AL) = character to write

Set Color Pallette

Input:
(AH) = OBH
(BH) = color palette ID (0-127)
(BL) = color value to be used with that color ID

Color ID = 0 selects the background color (0-15)
Color ID = 1 selects the palette to be used:
0 = green/red/yellow
1 = cyan/magenta/white

Write Dot

Input:
(AH) = OCH
(DX) = row number
(CX) = column number
(AL) = color value. If bit 7 of AL = 1, the color value is exclusive OR'd with the current contents of the dot.

Read Dot

Input:
(AH) = ODH
(DX) = row number
(CX) = column number

Output:
(AL) = the dot read

Write Teletype

This routine is used by the "TYPE" command and other DOS commands to display data on the screen.

Input:
(AH) = OEH
(AL) = character to write
(BL) = foreground color in graphics mode

Note
Screen width is controlled by previous mode set.

Current Video State

Input:
(AH) = OFH

Output:
(AL) = current mode
(AH) = number of character columns on screen
(BH) = current active display page

Diskette Services

Introduction Interrupt 13H is the BIOS routine for diskette services. There are six services provided by INT 13H.

Input

(AH) = 0 Reset Diskette System
(AH) = 1 Read status of diskette system into AL
(AH) = 2 Read sectors into memory
(AH) = 3 Write sectors from memory to diskette
(AH) = 4 Verify the specified sectors
(AH) = 5 Format a track

Additional settings for read, write, verify, and format:

(DL) = drive number (0 - 3 allowed, value checked)
(DH) = head number (0 - 1 allowed, value not checked)
(CH) = track number (0-39 allowed, value not checked)

Additional settings for read, write, and verify:

(CL) = sector number (1-9, value not checked)
(AL) = number of sectors (max = 9, value not checked)

ES:BX = address of buffer (not required for verify) For the format operation, ES:BX points to the collection of address fields for the track. There must be one of these fields for every sector on the track. Each field has four bytes:

Offset 0 = track number
1 = head number
2 = sector number
3 = number of bytes/ sector
(00 = 128, 01 = 256, 02 = 512, 03 = 1024)

Output

(AH) = Status of operation:

01 bad command
02 address mark not found
03 write was requested on write-protected disk
04 requested sector not found
08 DMA overrun
09 DMA transfer crossed a 64K boundary
10 read data error detected by CRC
20 diskette controller chip failed
40 seek to desired track failed
80 device timeout

(CY) = 0 successful operation
(CY) = 1 unsuccessful operation (AH has details)

For read, write, and verify these registers are preserved: DS, BX, DX, CH, and CL.

(AL) = number of sectors read; this value may be incorrect if a timeout occurred.

NOTE

If an error is reported by the diskette, reset the diskette, then retry the operation. On read operations, no motor start delay is taken, so your code should retry three times to make sure that a read error is not caused by motor start-up.

Communications Services

Introduction This set of routines performs serial, RS232C communications through the communications port. You should use a polling technique in your communications; this is not interrupt-driven I/O. All functions are accessed through BIOS interrupt 14H.

Initialize the Communications Port

Input:
(AH) = 0
(DX) = selection of RS-232 channel (0 or 1)
(AL) = parameters for initialization in the following form:

7 6 5	4 3	2	1 0
—Baud Rate—	—Parity—	Stopbit	—Word length—
000 - 110 baud	00 - None	0 - 1	10 - 7 bits
001 - 150	01 - Odd	1 - 2	11 - 8 bits
010 - 300	11 - Even		
011 - 600			
100 - 1200			
101 - 2400			
110 - 4800			
111 - 9600			

Output:
Condition is set according to the same conventions as in "Get Comm Port Status" (see below).

Send Character

Input:
(AH) = 1
(DX) = RS232 channel to be used (0 or 1)
(AL) = the character to be sent.

Output:
(AL) is preserved.
(AH) — if the operation was unsuccessful, bit 7 is set. The other bits in (AH) are set as they are in "Get Comm Port Status" if the operation was successful.

Receive Character

Input:
(AH) = 2
(DX) = RS232 channel to be used (0 or 1)

Output:
(AL) = the received character.
(AH) = status of operation, if (AH) = 0, the operation was successful. If the high order bit of (AH) is set, a timeout error aborted the operation and the rest of (AH) can be ignored. Any other setting of (AH) indicates errors in the receive character operation.

Get Comm Port Status

Input:
(AH) = 3
(DX) = RS232 channel to be used (0 or 1)

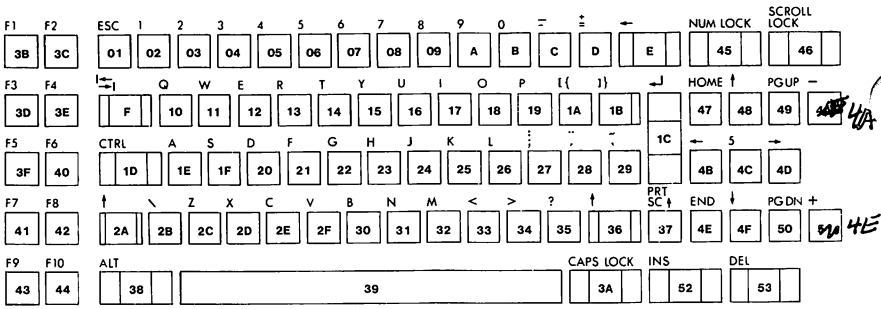
Output
(AX) = status:
(AH) = line control status
- bit 7 = timeout
- bit 6 = transmission shift reg. empty
- bit 5 = transmission holding reg. empty
- bit 4 = break detect
- bit 3 = framing error
- bit 2 = parity error
- bit 1 = overrun error
- bit 0 = data ready

(AL) = modem status
- bit 7 = received line signal detect
- bit 6 = ringing detect
- bit 5 = data set ready
- bit 4 = clear to send
- bit 3 = delta receive line signal detect
- bit 2 = trailing edge ring detected
- bit 1 = delta data set ready
- bit 0 = delta clear to send

Keyboard Handling

Introduction

Interrupt 16H provides the keyboard handling functions through three sub-functions. Most keys return two values: a scan code and a character code. The scan code is the same as the key number (see diagram below), and the character code is the ASCII superset interpretation of the key (including coincident SHIFTs or CTRLs). Check the section on "DOS Interrupts and Function Calls" to select either the BIOS keyboard routines or the DOS routines.



CHARACTER CODES

ASCII Decimal	Value Hex	Character	Control Character	ASCII Decimal	Value Hex	Character
000	00	(null)	NUL	032	20	(space)
001	01	☺	SOH	033	21	!
002	02	☻	STX	034	22	"
003	03	♥	ETX	035	23	#
004	04	♦	EOT	036	24	$
005	05	♣	ENQ	037	25	%
006	06	♠	ACK	038	26	&
007	07	(beep)	BEL	039	27	'
008	08	◘	BS	040	28	(
009	09	(tab)	HT	041	29	)
010	0A	(line feed)	LF	042	2A	*
011	0B	(home)	VT	043	2B	+
012	0C	(form feed)	FF	044	2C	,
013	0D	(carriage return)	CR	045	2D	-
014	0E	♫	SO	046	2E	.
015	0F	☼	SI	047	2F	/
016	10	►	DLE	048	30	0
017	11	◄	DC1	049	31	1
018	12	↕	DC2	050	32	2
019	13	!!	DC3	051	33	3
020	14	¶	DC4	052	34	4
021	15	§	NAK	053	35	5
022	16	▬	SYN	054	36	6
023	17	↨	ETB	055	37	7
024	18	↑	CAN	056	38	8
025	19	↓	EM	057	39	9
026	1A	→	SUB	058	3A	:
027	1B	←	ESC	059	3B	;
028	1C	(cursor right)	FS	060	3C	<
029	1D	(cursor left)	GS	061	3D	=
030	1E	(cursor up)	RS	062	3E	>
031	1F	(cursor down)	US	063	3F	?

CHARACTER CODES (Cont'd)

ASCII Decimal	Value Hex	Character	Control Character	ASCII Decimal	Value Hex	Character
064	40	@		096	60	©
065	41	A		097	61	a
066	42	B		098	62	b
067	43	C		099	63	c
068	44	D		100	64	d
069	45	E		101	65	e
070	46	F		102	66	f
071	47	G		103	67	g
072	48	H		104	68	h
073	49	I		105	69	i
074	4A	J		106	6A	j
075	4B	K		107	6B	k
076	4C	L		108	6C	l
077	4D	M		109	6D	m
078	4E	N		110	6E	n
079	4F	O		111	6F	o
080	50	P		112	70	p
081	51	Q		113	71	q
082	52	R		114	72	r
083	53	S		115	73	s
084	54	T		116	74	t
085	55	U		117	75	u
086	56	V		118	76	v
087	57	W		119	77	w
088	58	X		120	78	x
089	59	Y		121	79	y
090	5A	Z		122	7A	z
091	5B	[		123	7B	{
092	5C	\		124	7C	¦
093	5D	]		125	7D	}
094	5E	^		126	7E	~
095	5F	—		127	7F	⌂

CHARACTER CODES (Cont'd)

ASCII Decimal	Value Hex	Character	ASCII Decimal	Value Hex	Character
128	80	Ç	160	A0	á
129	81	ü	161	A1	í
130	82	é	162	A2	ó
131	83	â	163	A3	ú
132	84	ä	164	A4	ñ
133	85	à	165	A5	Ñ
134	86	å	166	A6	ª
135	87	ç	167	A7	º
136	88	ê	168	A8	¿
137	89	ë	169	A9	⌐
138	8A	è	170	AA	¬
139	8B	ï	171	AB	½
140	8C	î	172	AC	¼
141	8D	ì	173	AD	¡
142	8E	Ä	174	AE	«
143	8F	Å	175	AF	»
144	90	É	176	B0	▒
145	91	æ	177	B1	▒
146	92	Æ	178	B2	▓
147	93	ô	179	B3	│
148	94	ö	180	B4	┤
149	95	ò	181	B5	╡
150	96	û	182	B6	╢
151	97	ù	183	B7	╖
152	98	ÿ	184	B8	╕
153	99	Ö	185	B9	╣
154	9A	Ü	186	BA	║
155	9B	¢	187	BB	╗
156	9C	£	188	BC	╝
157	9D	¥	189	BD	╜
158	9E	Pt	190	BE	╛
159	9F	ƒ	191	BF	┐

CHARACTER CODES (Cont'd)

ASCII Decimal	Value Hex	Character	ASCII Decimal	Value Hex	Character
192	C0	└	224	E0	α
193	C1	┴	225	E1	ß
194	C2	┬	226	E2	Γ
195	C3	├	227	E3	π
196	C4	─	228	E4	Σ
197	C5	┼	229	E5	σ
198	C6	╞	230	E6	µ
199	C7	╟	231	E7	τ
200	C8	╚	232	E8	Φ
201	C9	╔	233	E9	Θ
202	CA	╩	234	EA	Ω
203	CB	╦	235	EB	δ
204	CC	╠	236	EC	∞
205	CD	═	237	ED	ø
206	CE	╬	238	EE	ε
207	CF	╧	239	EF	∩
208	D0	╨	240	F0	≡
209	D1	╤	241	F1	±
210	D2	╥	242	F2	≥
211	D3	╙	243	F3	≤
212	D4	╘	244	F4	⌠
213	D5	╒	245	F5	⌡
214	D6	╓	246	F6	÷
215	D7	╫	247	F7	≈
216	D8	╪	248	F8	°
217	D9	┘	249	F9	∙
218	DA	┌	250	FA	·
219	DB	▉	251	FB	√
220	DC	▄	252	FC	n
221	DD	▌	253	FD	2
222	DE	▐	254	FE	■
223	DF	▀	255	FF	(blank)

Read Next ASCII Character

Input:
(AH) = 0

Output:
(AL) = character code
(AH) = scan code

Note

This routine will not return execution to the calling program until it has a keystroke to report.

Check if Keystroke Available

This routine is used to check to see if a keystroke has been entered. Use this function if you want to continue processing whether or not a key has been pressed.

Input:
(AH) = 1

Output:
Z flag = 1 — no code available
Z flag = 0 — code is available
If a character is available, it is stored in AX in the same format as for the "Read Next ASCII Character" call. However, the code also remains in the keyboard buffer, so that a "Read Next ASCII Character" call returns this character's code value again.

Get Current Shift Status

Input:
(AH) = 2

Output:
(AL) = current shift status

Shift States

Bit	Subject matter	Meaning, when bit is 1
7	Insert	state active
6	Caps-Lock	state active
5	Num-Lock	state active
4	Scroll-Lock	state active
3	Alt shift	key depressed
2	Ctrl shift	key depressed
1	left-hand shift	key depressed
0	right-hand shift	key depressed

Printer Routines

This set of BIOS routines communicates with the printer through interrupt 17H.

Print a Character

Input:
(AH) = 0
(AL) = the character to be printed
(DX) = printer port number (0-3)

Output:
(AH) = 1 if character not printed due to timeout. Otherwise, bits are set as they are in "Get Printer Status" call.

Initialize Printer Port

Input:
(AH) = 1
(DX) = printer port number (0-3)

Output:
(AH) = printer status (see below)

Get Printer Status

Input:
(AH) = 2
(DX) = printer port number (0-3)

Output:

Bit	Meaning if Set (equal to 1)
7	not busy
6	acknowledge
5	out of paper
4	selected
3	I/O error
2	not used
1	not used
0	timeout (set by software)

If a printer is connected, 10H and 90H are healthy statuses. Otherwise 30H is healthy.

Miscellaneous ROM BIOS Services

System Reset

If you issue issue interrupt 19H, the system bootstraps itself in much the same way as it does with the Ctrl-Alt-Delete key sequence. The only difference is that Ctrl-Alt-Delete causes diagnostics to be run, whereas Interrupt 19H causes an immediate system load.

Print Screen

To obtain a printed copy of what is on the screen, issue a request for interrupt 5H. This produces exactly the same result as pressing the shift and PrtSc keys. This routine works in either text or graphics modes. Unrecognizable characters are printed as blanks.

Equipment List

You can use this routine to obtain a list of the optional equipment attached to your system. Simply issue an interrupt 11H; no register set-up is necessary. Output (starting with the most significant bit of (AX)) is as follows:

	Bit	Meaning
AH	7-6	number of printer adapters (0-3)
	5	not used
	4	game adapter attached, or not
	3-1	number of communications adapters (0-7)
	0	not used
AL	7-6	number of diskette drives minus 1, if bit 0 is set.
	5-4	starting video mode: 01 — graphics card display,40 columns,b/w 10 — graphics card display,80 columns,b/w 11 —monochrome card display
	3-2	amount of memory on system board: 00 — 16KB Base 01 — 32KB Base 10 — 48KB Base 11 — 64KB Base
	1	not used
	0	diskettes are attached; refer to bits 7 and 6.

Determine Memory Size

Interrupt 12H gives the total amount of memory in the address space, up to a megabyte. No register set-up is required. The BIOS reads the switches on the system board and adds the amount of memory on a memory expansion board and returns the total in (AX). The amount of memory is expressed as the number of 1K blocks.

Bypassing the BIOS

This section explains how you can either replace one of the BIOS service routines with a program of your own or add a "front-end" so that you perform some preprocessing immediately prior to using a particular BIOS routine.

When the system is powered on, low memory is initialized with the addresses of all of the BIOS interrupt routines. To replace a BIOS routine, change the address in the interrupt table to the address of the code which you want to execute in place of the BIOS code. To perform preprocessing before handing execution on to the BIOS code:

1. replace the address of the BIOS routine with the address of your program
2. transfer execution to the BIOS routine at the end of your program.

Be sure to use the "Set Vector" system call (Function Request 25H) to replace the BIOS routine addresses instead of writing directly to low memory.

CONFIG.SYS

The special file CONFIG.SYS is processed automatically when DOS starts. As with the batch file AUTOEXEC.BAT, this processing is automatic. DOS will simply look at the root directory to see if the file is there.

CONFIG.SYS is a text file that can be edited by EDLIN or any other text editor that produces ASCII files. Five commands can be used in the CONFIG.SYS file. Each command changes a system parameter.

- **BREAK ON/OFF**
  Changes the way DOS checks for a CTRL/BREAK

- **BUFFERS=xx**
  Sets the number of data buffers that DOS uses.

- **DEVICE=[d:] [path] filename [.ext]**
  Adds a nonstandard device driver to DOS.

- **FILES=xx**
  Sets the number of files that can use ASCIIZ strings.

- **SHELL=[d:] [path] filename [.ext] [d:] [path] /P**
  Specifies an alternate command processor.

Break

Normally, DOS checks for a CTRL/BREAK only when it is doing input or output. Some programs do very little (if any) input or output for long periods of time. The BREAK ON command sets DOS to check for a CTRL/BREAK when any DOS function is called.

BREAK OFF resets the default so that DOS only checks for a CTRL/BREAK during input and output. This command can be used to override a BREAK ON that was set by CONFIG.-SYS.

Buffers

The number of buffers has an effect on both the speed of disk I/O and available memory. A larger number of buffers allocates more memory to the system for operations. This is highly desirable for systems with large amounts of RAM that will be used for database applications. It is highly undesirable for systems with minimal RAM that do little work with disk files.

The system default is BUFFERS=2, which is adequate for most purposes and requires only 1K of RAM. If your system will be doing a significant amount of data handling, especially on a hard disk, BUFFERS=4 would improve access times at a cost of only 1K of internal memory.

Device

When DOS is first started, it loads all of the standard device drivers for the keyboard, screen and so on. If your system requires a special device driver, this command tells DOS where to find it. The device driver will then be loaded as an extension to DOS.

Device drivers are .COM files with a specific structure described in Chapter 9. The command DEVICE=ANSI.SYS causes DOS to replace the standard display and keyboard device drivers with the extended screen and keyboard support that the extended functions require.

If you wish to load several special device drivers, you must use a DEVICE command for each one.

Files

Opening a file with an ASCIIZ string eliminates the traditional direct handling of a File Control Block (FCB). DOS will handle all these things for you by creating and maintaining FCBs internally. To do this, it needs memory. Each file requires 39 bytes of memory.

The FILES command sets aside memory for this operation. The default is FILES=8. The maximum is FILES=99. This is the limit for the entire system. A particular process (or program) can still have only 20 files open at once.

Shell

The COMMAND.COM file that DOS uses as its "front end" processor can be replaced by another command processor. The SHELL command specifies the file to be used and the default path for processing commands. The command processor must be able to read and execute commands, and handle interrupts 22H, 23H, and 24H.

Since COMMAND.COM handles internal commands, .BAT file execution, and .EXE file loading, these functions will be unavailable unless the new command processor duplicates them.

(This page intentionally left blank)

ROM BIOS Listing

```
                        ;BOOT ROM - AT&T PERSONAL COMPUTER 6300
                        ;Rev. 1.0 - MAY 1984

                        page 60,132
                        ;===============================================================================
                        ;       Filename:       flags.src
                        ;
                        ;       This file contains the temporary conditional assembly flag
                        ;       definitions and includes of the other assembly modules.
                        ;
                        ;===============================================================================

                        ;-------------------------------------------------------------------------------
                        ;       Macro Definitions
                        ;-------------------------------------------------------------------------------

                        ; MASM does not let you code a `jump intersegment direct' instruction, so
                        ; this macro simulates that instruction.

                        jmpf    macro   arg1,arg2               ;; USAGE:       jmpf    seg,off
                                db      0EAh
                                dw      arg2
                                dw      arg1
                                endm

                        ;-------------------------------------------------------------------------------
                        ;       Code Declaration
                        ;-------------------------------------------------------------------------------

0000                    code    segment public 'ROM'    ; link code segments first
                                assume  cs:code, ds:nothing, es:nothing, ss:nothing

C000                            ORG     0C000h

C000                    flags_data1     proc

C000  00                chk_lo          db      0       ; space for checksum of F000:C000 to F000:DFFF
C001  00                rom_id          db      0       ; ROM identifier.
C002  E2CE R            rom_at          dw      mastab  ; offset of mastab in ROM.

C004                    flags_data1     endp

C004                    far_calls       proc    far     ; far call table: the user does a far call to
                                                        ; F000:C0XX, a near call is done to the proper
                                                        ; routine, and a far return back to the user.

C004  E8 E602 R                 call    DString                 ; F000:C004     (3 bytes per near call)
C007  CB                        ret                             ;               (1 byte per far return)
C008  E8 E619 R                 call    DCrLf                   ; F000:C008
C00B  CB                        ret
C00C  E8 E626 R                 call    DColon                  ; F000:C00C
C00F  CB                        ret
C010  E8 E632 R                 call    DHexLong                ; F000:C010
C013  CB                        ret
C014  E8 E63C R                 call    DHexWord                ; F000:C014
C017  CB                        ret
C018  E8 E643 R                 call    DHexByte                ; F000:C018
C01B  CB                        ret
C01C  E8 E650 R                 call    DHexNib                 ; F000:C01C
C01F  CB                        ret
C020  E8 E665 R                 call    DNum                    ; F000:C020
C023  CB                        ret
C024  E8 E66D R                 call    DNumW                   ; F000:C024
C027  CB                        ret
C028  E8 E5E5 R                 call    rom_checksum            ; F000:C028
C02B  CB                        ret
C02C  E8 E1F6 R                 call    rtc_chk                 ; F000:C02C
C02F  CB                        ret
C030  E8 E266 R                 call    memtst                  ; F000:C030
C033  CB                        ret
C034  E8 CF9F R                 call    h_init                  ; F000:C034
C037  CB                        ret
C038  E8 D640 R                 call    h_fmt                   ; F000:C038
C03B  CB                        ret

C03C  DB8C R                    dw      offset banner_m         ; pointer to banner
C03E  0000                      dw      0                       ; For aligning the copyright message.
C040  43 4F 50 59 52 49         db      'COPYRIGHT (C)  '
      47 48 54 20 28 43
      29 20 20 20
C050  4F 4C 49 56 45 54         db      'OLIVETTI 1984  '
      54 49 20 31 39 38
      34 20 20 20

C060                    far_calls       endp

C060                    code    ends

                        ;-------------------------------------------------------------------------------
                        ;       Includes of Assembly Modules
                        ;-------------------------------------------------------------------------------

                        ;include flags.asm                              (this module)
                      C include sysdata.asm
                      C
                      C ;===============================================================================
                      C ;       Filename:       sysdata.src
                      C ;
                      C ;       This is the port equate and system data definition module.
```

```
                        ;       (See flags.src for conditional assembly flags...)
                        ;
                        ;=============================================================================

                        ;-----------------------------------------------------------------------
                        ;       Global Constants
                        ;-----------------------------------------------------------------------

= 0000                  abs0_seg        equ     00000h
= 0030                  stack_seg       equ     00030h
= 0040                  data_seg        equ     00040h
= B000                  para_mono       equ     0B000h
= B800                  para_graph      equ     0B800h
= F000                  code_seg        equ     0F000h

= 0007                  BEL             equ     007h
= 0008                  BS              equ     008h
= 000D                  CR              equ     00Dh
= 000A                  LF              equ     00Ah

= 0000                  NUL             equ     0

                        ;-----------------------------------------------------------------------
                        ;       AT&T PERSONAL COMPUTER Addresses
                        ;-----------------------------------------------------------------------

                        ;-----------------------------------------------------------------------
                        ;       i8237A p_dma Controller Port Addresses
                        ;-----------------------------------------------------------------------

= 0000                  dma_addr_0      equ     00h     ; 16-bit address register - channel 0 - refresh
= 0001                  dma_count_0     equ     01h     ; 16-bit count register
= 0002                  dma_addr_1      equ     02h     ; 16-bit address register - channel 1 - not used
= 0003                  dma_count_1     equ     03h     ; 16-bit count register
= 0004                  dma_addr_2      equ     04h     ; 16-bit address register - channel 2 - FDU
= 0005                  dma_count_2     equ     05h     ; 16-bit count register
= 0006                  dma_addr_3      equ     06h     ; 16-bit address register - channel 3 - display
= 0007                  dma_count_3     equ     07h     ; 16-bit count register

= 0008                  dma_status      equ     08h     ;  8-bit read status register
= 0008                  dma_command     equ     08h     ;  8-bit write command register
= 0009                  dma_request     equ     09h     ;  4-bit write request register
= 000A                  dma_mask_bit    equ     0Ah     ;  4-bit (write) set/clear one mask register bit
= 000B                  dma_mode        equ     0Bh     ;  6-bit write mode register
= 000C                  dma_ff_clr      equ     0Ch     ;       (write) clear byte pointer flip/flop
= 000D                  dma_temp        equ     0Dh     ;  8-bit read temporary register
= 000D                  dma_master_clr  equ     0Dh     ;       (write) master clear command
= 000E                  dma_mask_clr    equ     0Eh     ;  4-bit (write) clear all mask register bits
= 000F                  dma_mask_write  equ     0Fh     ;  4-bit write all mask register bits at once

= 0080                  dma_segm_0      equ     080h    ; RAM refresh - 4x4-bit high nibble segment port
= 0082                  dma_segm_1      equ     082h    ; not used
= 0081                  dma_segm_2      equ     081h    ; FDU
= 0083                  dma_segm_3      equ     083h    ; display                       TEMP


                        ;-----------------------------------------------------------------------
                        ;       i8237A p_dma controller constants
                        ;-----------------------------------------------------------------------

                        ; dma_command port:                    ; bit #0: memory-to-memory/~I/O enable
                                                               ; bit #2: controller disable
                                                               ; bit #3: compressed/normal timing
                                                               ; bit #4: rotating/~fixed priority
                                                               ; bit #5: extended/~late write selection
                                                               ; bit #6: DREQ active low/~high
                                                               ; bit #7: DACK active high/~low

= 0004                  dma_cmd_disable equ     004h           ; controller disable (bit #2) command
= 0000                  dma_cmd_enable  equ     000h           ; memory-to-I/O,controller enable,normal
                                                               ; fixed priority, late write, DREQ/~DACK

= 0058                  dma_mode_0      equ     058h           ; channel 0, read, autoinitialize, inc-
                                                               ; rement, single mode for RAM refresh.
= 0041                  dma_mode_1      equ     041h           ; channel 1, verify, autoinit disabled,
                                                               ; increment, single mode for not used.
= 0056                  dma_mode_2      equ     056h           ; channel 2, write, autoinitialize, inc-
                                                               ; rement, single mode for FDU.
= 0043                  dma_mode_3      equ     043h           ; channel 3, verify, autoinit disabled,
                                                               ; increment, single mode for display.

                        ; dma_mask_bit port:                   ; bits #0-1: channel select
                                                               ; bit  #  2: set/~clr mask bit (off/~on)

= 0000                  dma_unmask_0    equ     000h           ; turn on channel 0 for RAM refresh.

                        ;-----------------------------------------------------------------------
                        ;       i8259A Programmable Interrupt Controller Port Addresses
                        ;-----------------------------------------------------------------------

= CCCC                  pic_0           equ     020h    ; 8259A 'control' port (A0 = 0)
= 0021                  pic_1           equ     021h    ; 8259A 'data'    port (A0 = 1)

                        ;-----------------------------------------------------------------------
                        ;       i8259A Programmable Interrupt Controller Commands
                        ;-----------------------------------------------------------------------

= 0013                  pic_icw1        equ     013h    ; ICW1 for both master & slave pic's
                                                        ; bit #0 = 1: ICW4 to follow (w/vector base)
                                                        ; bit #1 = 1: single mode (no slaves or icw3)
```

```
                        C                                       ; bit #2 = 0: call address interval of 8 bytes
                        C                                       ; (don't care if 8086 mode -- always vectors &
                        C                                       ; byte interval)
                        C                                       ; bit #3 = 0: edge triggered
= 0008                  C  pic_icw2         equ     008h        ; interrupt vector base address (INTs 08h - 0Fh)
= 0008                  C  pic_icw3         equ     008h        ; if cascade mode , and IR3 is a
                        C                                       ; slave, 8259A is reprogrammed including icw3
= 000D                  C  pic_icw4         equ     00Dh        ; bit #0 = 1: 8086 mode
                        C                                       ; bit #1 = 0: normal end_of_int
                        C                                       ; bit #2 = 1: specify master for buffered mode
                        C                                       ; (  specifies slave for buffered mode )
                        C                                       ; bit #3 = 1: buffered mode
                        C                                       ; bit #4 = 0: not special fully nested
= 00FF                  C  pic_off_msk      equ     0FFh        ; pic interrupt mask bits (all interrupts off)
                        C
= 0020                  C  pic_eoi          equ     020h        ; non-specific end-of-interrupt
                        C
= 0060                  C  pic_eoi_0        equ     060h        ; specific end-of-interrupt for IR0: i8253 p_timer
= 0061                  C  pic_eoi_1        equ     061h        ; specific end-of-interrupt for IR0: i8041A kb
= 0066                  C  pic_eoi_6        equ     066h        ; specific end-of-interrupt for IR6: fdu
                        C
                        C  ;-------------------------------------------------------------------------
                        C  ;       i8253 p_timer Port Addresses
                        C  ;-------------------------------------------------------------------------
                        C
= 0040                  C  p_8253_0         equ     040h        ; 8253 p_timer 0 - rtc interrupt - IRQ = INT 08h
= 0041                  C  p_8253_1         equ     041h        ; 8253 p_timer 1 - memory refresh p_dma
= 0042                  C  p_8253_2         equ     042h        ; 8253 p_timer 2 - tone generator for speaker
= 0043                  C  p_8253_ctrl      equ     043h        ; 8253 p_timer control port
                        C
                        C  ;-------------------------------------------------------------------------
                        C  ;       i8253 p_timer Control Bytes
                        C
                        C  ; bit  #0        -> Binary Code Decimal (BCD) Enable
                        C  ; bits #1-3      -> Mode (0-5)   000     Mode 0: Interrupt on Terminal Count
                        C  ;                                001     Mode 1: Programmable One-Shot
                        C  ;                                x10     Mode 2: Rate Generator
                        C  ;                                x11     Mode 3: Square Wave Rate Generator
                        C  ;                                100     Mode 4: Software Triggered Strobe
                        C  ;                                101     Mode 5: Hardware Triggered Strobe
                        C  ; bits #4-5      -> Read/Load Instruction (0-3)
                        C  ; bits #6-7      -> Select Counter (0-2)
                        C
                        C  ;-------------------------------------------------------------------------
= 0036                  C  t0cmd            equ     036h        ; 00 11 011 0 -> p_8253_0, lsb 1st, mode 3, no BCD
= 0074                  C  t1cmd            equ     074h        ; 01 11 010 0 -> p_8253_1, lsb 1st, mode 2, no BCD
= 00B6                  C  t2cmd            equ     0B6h        ; 10 11 011 0 -> p_8253_2, lsb 1st, mode 3, no BCD
                        C
                        C  ;-------------------------------------------------------------------------
                        C  ;       i8253 p_timer Counts
                        C
                        C  ; 8253 input is 1.2288 MHz (3.6864/3) or a period of 813.8 nsec = 0.814 usec
                        C  ; Note:      PC input is 1.19318 MHz or a period of 838.1 nsec = 0.838 usec
                        C
                        C  ;-------------------------------------------------------------------------
= 0000                  C  t0count equ      0           ; = 65,536  -> (1,228,800 Hz)/(65,536) = 18.75 ints/sec
                        C                               ;           -> (1,193,180 Hz)/(65,536) = 18.21 ints/sec
                        C
                        C  ;t1count equ     9           ; OLD refresh cycle =  9*(813.8 nsec) = 7.32 usec
= 0013                  C  t1count equ      19          ; REAL refresh cycle = 19*(813.8 nsec) = 15.5 usec
                        C                               ; < 15.625 usec minimum required. (  is 18 - safety??)
                        C
= 0266                  C  t2count equ      614         ; (1.2288 MHz)/(2*614) = 1.00 kHz tone
                        C
                        C  ;-------------------------------------------------------------------------
                        C  ;       Z8530 Serial Communication Controller
                        C  ;
                        C  ;       (scc_data_x port addresses are indexed from the scc_ctl_x
                        C  ;       port addresses.  See com.src.)
                        C  ;-------------------------------------------------------------------------
                        C
= 0050                  C  scc_ctl_a        equ     050h        ; write to SCC pointer register (0-Fh),
= 0052                  C  scc_ctl_b        equ     052h        ; then read or write from selected register.
                        C
                        C  ;-------------------------------------------------------------------------
                        C  ;       8041 Keyboard Controller
                        C  ;-------------------------------------------------------------------------
                        C
= 0060                  C  p_kscan          equ     060h
= 0061                  C  p_kctrl          equ     061h        ; bit  #7:      reset interrupt pending
                        C                                       ; bit  #6:      kb clock reset
                        C                                       ; bit  #5:      I/O channel (NMI) enable
                        C                                       ; bit  #4:      RAM parity (NMI) enable
                        C                                       ; bits #3 & #2: not used
                        C                                       ; bit  #1:      speaker data
                        C                                       ; bit  #0:      speaker gate to p_8253_2
= 0064                  C  kb_status        equ     064h        ; bit  #1:      input buffer (ok to write byte)
                        C                                       ; bit  #0:      output buffer (byte to be read)
                        C
                        C  ;-------------------------------------------------------------------------
                        C  ;       General Control Ports
                        C  ;-------------------------------------------------------------------------
                        C
= 0062                  C  Control8         equ     062h        ; 8087, etc.
= 0065                  C  CommControl      equ     065h
= 0066                  C  sys_conf_a       equ     066h        ; bit  #7:      2764/*2732 ROM's
                        C                                       ; bit  #6:      not used
                        C                                       ; bit  #5:      SCC 8530 chip installed
                        C                                       ; bit  #4:      8087 installed
```

```
                                                                ; bit  #3:      256k x 1 RAM used
                                                                ; bits #2 - #0: RAM configuration
= 0067                  sys_conf_b      equ     067h    ; bits #7 - #6: (number of FDUs)-1
                                                                ; bits #5 - #4: reserved for monitor type
                                                                ; bits #3 - #2: reserved for HDU type
                                                                ; bit  #1:      "fast" FDU start up
                                                                ; bits #0:      96 tpi FDU

                        ;-----------------------------------------------------------------------
                        ;       58174A Clock Calendar
                        ;
                        ;       (See calendar.src)
                        ;-----------------------------------------------------------------------

                        ;-----------------------------------------------------------------------
                        ;       FDU & HDU Disk Driver Error Codes
                        ;-----------------------------------------------------------------------

= 0080                  time_out        equ     80h
= 0040                  seek_error      equ     40h
= 0020                  fdc_error       equ     20h
= 0010                  crc_error       equ     10h
= 0009                  dma_seg_error   equ     09h
= 0008                  dma_error       equ     08h
= 0004                  sect_not_found  equ     04h
= 0003                  write_protect   equ     03h
= 0002                  addr_mark_error equ     02h
= 0001                  cmd_error       equ     01h

                        ;-----------------------------------------------------------------------
                        ;       Game Card
                        ;-----------------------------------------------------------------------

= 0201                  game_card       equ     201h

                        ;-----------------------------------------------------------------------
                        ;       Parallel Printer Interface
                        ;
                        ;       (prt_stat_x & prt_cmd_x port addresses are indexed from the
                        ;       prt_data_x port addresses.  See prt.src.)
                        ;-----------------------------------------------------------------------

= 03BC                  prt_data_a      equ     03BCh
                        ; prt_stat_a    equ     03BDh
                        ; prt_cmd_a     equ     03BEh

= 0378                  prt_data_b      equ     0378h   ; on mother board
                        ; prt_stat_b    equ     0379h
                        ; prt_cmd_b     equ     037Ah

= 0278                  prt_data_c      equ     0278h
                        ; prt_stat_c    equ     0279h
                        ; prt_cmd_c     equ     027Ah

                        ;-----------------------------------------------------------------------
                        ;       Color and Monochrome Video Controller

                        ;       (xxxxx_data, xxxxx_mode, xxxxx_status, xxxxx_LPolear, and
                        ;       xxxxx_LPPreset port addresses are indexed from the xxxxx_Pointer
                        ;       port addresses for color & display.  See vid.src and graph.src.)
                        ;-----------------------------------------------------------------------

                        ; Color Controller.

= 03D4                  color_pointer   equ     03D4h           ; 6845 pointer to internal regs

                        ; Monochrome Controller.

= 03B4                  v_pointer       equ     03B4h           ; 6845 pointer to internal regs

                        ;-----------------------------------------------------------------------
                        ;       INS8250 Asynchronous Communication Chip
                        ;
                        ;       (com_int_x, com_lctl_x, com_mctl_x, com_lstat_x, and
                        ;       com_mstat_x port addresses are indexed from the com_data_x
                        ;       port addresses.  See com.src.)
                        ;-----------------------------------------------------------------------

= 03F8                  com_data_a      equ     03F8h   ; channel A 8250 data register/low byte baud
                        ; com_int_a     equ     03F9h   ; channel A 8250 high byte baud count register
= 03FA                  com_id_a        equ     03FAh   ; channel A 8250 check for presence register
                        ; com_lctl_a    equ     03FBh   ; channel A 8250 line control register
                        ; com_mctl_a    equ     03FCh   ; channel A 8250 modem control register
                        ; com_lstat_a   equ     03FDh   ; channel A 8250 line status register
                        ; com_mstat_a   equ     03FEh   ; channel A 8250 modem status register

= 02F8                  com_data_b      equ     02F8h   ; channel B 8250 data register/low byte baud
                        ; com_int_b     equ     02F9h   ; channel B 8250 high byte baud count register
= 02FA                  com_id_b        equ     02FAh   ; channel B 8250 check for presence register
                        ; com_lctl_b    equ     02FBh   ; channel B 8250 line control register
                        ; com_mctl_b    equ     02FCh   ; channel B 8250 modem control register
                        ; com_lstat_b   equ     02FDh   ; channel B 8250 line status register
                        ; com_mstat_b   equ     02FEh   ; channel B 8250 modem status register

                        ;-----------------------------------------------------------------------
                        ;       Keyboard Constants
                        ;-----------------------------------------------------------------------

                        ; --- shift flag equates within kb_flag
```

```
                                C
= 0080                          C  insert_mode      equ     80h     ; insert state in action
= 0040                          C  caps_lock_mode   equ     40h     ; caps lock state toggled
= 0020                          C  num_lock_mode    equ     20h     ; num lock state toggled
= 0010                          C  scrl_lock_mode   equ     10h     ; scroll lock state toggled
= 0008                          C  pause_mode       equ     08h     ; pause toggled
= 0001                          C  dlx_kb           equ     01h     ; deluxe keyboard
                                C
                                C
                                C  ;---- shift flag equates within kb_flag_1
                                C
= 0080                          C  insert_shift     equ     80h     ; insert key depressed
= 0040                          C  caps_lock_shift  equ     40h     ; caps lock key depressed
= 0020                          C  num_lock_shift   equ     20h     ; num lock key depressed
= 0010                          C  scrl_lock_shift  equ     10h     ; scroll lock key depressed
= 0008                          C  alt_shift        equ     08h     ; alternate shift key depressed
= 0004                          C  cntrl_shift      equ     04h     ; control shift key depressed
= 0002                          C  left_shift       equ     02h     ; left shift key depressed
= 0001                          C  right_shift      equ     01h     ; right shift key depressed
                                C
                                C
                                C  ;---- Scan codes for special function keys
                                C
= 001D                          C  cntrl_key        equ     29      ; control key scan code
= 002A                          C  left_shift_key   equ     42      ; left shift scan code
= 0036                          C  right_shift_key  equ     54      ; right shift scan code
= 0038                          C  alt_key          equ     56      ; alt shift key scan code
= 003A                          C  caps_lock_key    equ     58      ; shift lock scan code
= 0045                          C  num_lock_key     equ     69      ; number lock scan code
= 0046                          C  scrl_lock_key    equ     70      ; scroll lock key scan code
= 0052                          C  insert_key       equ     82      ; insert key scan code
= 0053                          C  delete_key       equ     83      ; delete key scan code
                                C
                                C  ;--------------------------------------------------------------------------
                                C  ;       Data Declarations
                                C  ;--------------------------------------------------------------------------
                                C
                                C  ;--------------------------------------------------------------------------------
                                C  ;       Interrupt Locations (dummy data segment to define constant offsets)
                                C  ;--------------------------------------------------------------------------------
                                C
0000                            C  abs0     segment public 'RAM'            ; at abs0_seg
                                C          assume  cs:nothing, ds:nothing, es:nothing, ss:nothing
                                C
                                C  ;--------------------------------------------------------------------------------
                                C  ; CPU Interrupt Routines
                                C  ;--------------------------------------------------------------------------------
                                C
0000  ????????                  C  int00loc         dd      ?               ; divide by zero
0004  ????????                  C  int01loc         dd      ?               ; single step trap
0008  ????????                  C  int02loc         dd      ?               ; nmi parity trap
000C  ????????                  C  int03loc         dd      ?               ; break interrupt
0010  ????????                  C  int04loc         dd      ?               ; divide overflow
0014  ????????                  C  int05loc         dd      ?               ; print screen
                                C
0018  ????????                  C  int06loc         dd      ?
001C  ????????                  C  int07loc         dd      ?
                                C
                                C  ;--------------------------------------------------------------------------------
                                C  ; 18259A Hardware Interrupt Routines
                                C  ;--------------------------------------------------------------------------------
                                C
0020  ????????                  C  int08loc         dd      ?               ; 18253 rtc interrupt
0024  ????????                  C  int09loc         dd      ?               ; 18041 kb interrupt
                                C
0028  ????????                  C  int0Aloc         dd      ?
002C  ????????                  C  int0Bloc         dd      ?
0030  ????????                  C  int0Cloc         dd      ?
                                C
0034  ????????                  C  int0Dloc         dd      ?               ; hard disk interrupt
0038  ????????                  C  int0Eloc         dd      ?               ; floppy disk interrupt
                                C
003C  ????????                  C  int0Floc         dd      ?
                                C
                                C  ;--------------------------------------------------------------------------------
                                C  ; Software Interrupt Routines
                                C  ;--------------------------------------------------------------------------------
                                C
0040  ????????                  C  int10loc         dd      ?               ; display request
0044  ????????                  C  int11loc         dd      ?               ; equipment request
0048  ????????                  C  int12loc         dd      ?               ; memory size request
004C  ????????                  C  int13loc         dd      ?               ; disk I/O request
0050  ????????                  C  int14loc         dd      ?               ; serial communication request
0054  ????????                  C  int15loc         dd      ?               ; cassette request
0058  ????????                  C  int16loc         dd      ?               ; kb request
005C  ????????                  C  int17loc         dd      ?               ; printer request
0060  ????????                  C  int18loc         dd      ?               ; cassette BASIC pointer
0064  ????????                  C  int19loc         dd      ?               ; boot-strap request
0068  ????????                  C  int1Aloc         dd      ?               ; time of day request
006C  ????????                  C  int1Bloc         dd      ?               ; kb break pointer
0070  ????????                  C  int1Cloc         dd      ?               ; p_timer break pointer
0074  ????????                  C  int1Dloc         dd      ?               ; display parameter pointer
0078  ????????                  C  int1Eloc         dd      ?               ; disk parameter pointer
007C  ????????                  C  int1Floc         dd      ?               ; graphics character extensions pointer
                                C
0080                            C  abs0     ends
                                C
                                C  ;--------------------------------------------------------------------------
                                C  ;       RAM stack
                                C  ;--------------------------------------------------------------------------
                                C
```

```
0000                    C  stack_ram        segment public 'RAM'     ; at stack_seg
                        C
0000                    C  stack_ram        ends
                        C
                        C  ;--------------------------------------------------------------------
                        C  ;       System Data Area
                        C  ;--------------------------------------------------------------------
                        C
0000                    C  data     segment public 'RAM'             ; at data_seg
                        C          assume  cs:nothing, ds:nothing, es:nothing, ss:nothing
                        C
                        C  ;--------------------------------------------------------------------
                        C  ;       Data Area
                        C  ;--------------------------------------------------------------------
                        C
                        C  ;       ROM Bios Data Area
                        C
0000     04 [           C  rs232_addr       dw      4 dup (?)  ; 0040:0000  addresses of rs232 adapters
              ????      C
                   ]    C
                        C
0008     04 [           C  printer_addr     dw      4 dup (?)  ; 0040:0008  addresses of printers
              ????      C
                   ]    C
                        C
0010  ????              C  switch_bits      dw      ?          ; 0040:0010  state of DIP switches
0012  ??                C  mfg_tst          db      ?          ; 0040:0012  initialization flag
0013  ????              C  memory_size      dw      ?          ; 0040:0013  memory size in kbytes
0015     02 [           C  mfg_err_flag     db      2 dup (?)  ; 0040:0015  error codes for manufacturing
              ??        C
                   ]    C
                        C
                        C  ;       Keyboard Data Area
                        C
0017  ??                C  kb_flag          db      ?          ; 0040:0017  keyboard shift flag status byte
0018  ??                C  kb_flag_1        db      ?          ; 0040:0018  second byte of keyboard status
0019  ??                C  alt_input        db      ?          ; 0040:0019  alternate keypad entry
001A  ????              C  buffer_head      dw      ?          ; 0040:001A  keyboard output pointer offset
001C  ????              C  buffer_tail      dw      ?          ; 0040:001C  keyboard input pointer offset
                        C
001E     10 [           C  kb_buffer        dw      16 dup (?) ; 0040:001E  room for 15 entries: head =
              ????      C
                   ]    C
                        C
                        C                                      ;              tail implies buffer is empty
                        C
                        C  ;       Floppy Diskette Data Area
                        C
003E  ??                C  seek_status      db      ?          ; 0040:003E  floppy disk restore status bits
003F  ??                C  motor_status     db      ?          ; 0040:003F  floppy disk motor status bits
0040  ??                C  motor_count      db      ?          ; 0040:0040  floppy disk turn off counter
0041  ??                C  diskette_status  db      ?          ; 0040:0041  floppy disk driver status byte
                        C
0042                    C  cmd_block        label   byte       ; 0040:0042  HDU command block buffer
0042                    C  hd_error         label   byte       ; 0040:0042  HDU sense byte buffer
0042     07 [           C  nec_status       db      7 dup (?)  ; 0040:0042  status bytes from NEC controller
              ??        C
                   ]    C
                        C
                        C  ;       Video Display Data Area
                        C
0049  ??                C  v_mode           db      ?       ; 0040:0049     CRT mode
004A  ????              C  v_width          dw      ?       ; 0040:004A     CRT number of columns (often db)
004C  ????              C  v_height         dw      ?       ; 0040:004C     CRT length of video ram in bytes
004E  ????              C  v_top            dw      ?       ; 0040:004E     CRT video ram buffer address
0050     08 [           C  v_curpos         dw      8 dup (?)  ; 0040:0050  cursor for each of up to 8 pages
              ????      C
                   ]    C
                        C
0060  ????              C  v_cursize        dw      ?       ; 0040:0060     v_curs_type sets cursor value
0062  ??                C  v_apage          db      ?       ; 0040:0062
0063  ????              C  v_base6845       dw      ?       ; 0040:0063
0065  ??                C  v_3x8            db      ?       ; 0040:0065
0066  ??                C  v_colorpal       db      ?       ; 0040:0066
                        C
                        C  ;       Optional Post Data Area
                        C
0067  ????              C  io_rom_init      dw      ?          ; 0040:0067  option ROM init routine offset
0069  ????              C  io_rom_seg       dw      ?          ; 0040:0069  option ROM init routine segment
006B  ??                C  intr_flag        db      ?          ; 0040:006B  occurrence of interrupt flag
                        C
                        C  ;       i8253 p_timer Data Area
                        C
006C  ????              C  t_low_order      dw      ?          ; 0040:006C  low word of i8253 p_timer count
006E  ????              C  t_hi_order       dw      ?          ; 0040:006E  high word of i8253 p_timer count
0070  ??                C  t_overflow       db      ?          ; 0040:0070  time rolled over flag
                        C
                        C  ;       System Data Area
                        C
0071  ??                C  bios_break       db      ?          ; 0040:0071  bit #7 set if break key hit
0072  ????              C  reset_flag       dw      ?          ; 0040:0072  = 1234h if keyboard reset hit
                        C
                        C  ;       Fixed Disk Data Area
                        C
0074  ??                C  disk_status      db      ?          ; 0040:0074  fixed disk driver status byte
0075  ??                C  hf_num           db      ?          ; 0040:0075  fixed disk drive count
```

```
0076  ??                    C  control_byte    db      ?          ; 0040:0076  fixed disk control byte options
0077  ??                    C  port_off        db      ?          ; 0040:0077  fixed disk port offset
                            C
                            C  ;        Printer & RS-232 Time-Out Data Areas
                            C
0078      04 [              C  printer_t_out   db      4 dup (?)  ; 0040:0078  printer time-out variables
                ??          C
                    ]       C
                            C
007C      04 [              C  serial_t_out    db      4 dup (?)  ; 0040:007C  RS-232 time-out variables
                ??          C
                    ]       C
                            C
                            C
                            C  ;        Additional Keyboard Data Area
                            C
0080  ????                  C  buffer_start    dw      ?          ; 0040:0080  offset of kb_buffer     = 001E
0082  ????                  C  buffer_end      dw      ?          ; 0040:0082  offset of kb_buffer_end = 003E
                            C
                            C  ;--------------------------------------------------------------------------
                            C  ;       Data Area
                            C  ;--------------------------------------------------------------------------
                            C
                            C  ;       Master Table Pointer.
                            C
0084  ????????              C  master_tbl_ptr  dd      ?          ; 0040:0084  pointer to master table
                            C
0088                        C  data    ends
                            C
                            C  ;--------------------------------------------------------------------------
                            C  ;       Video RAM
                            C  ;--------------------------------------------------------------------------
                            C
0000                        C  v_ram   segment public 'RAM'    ; at para_mono
                            C
0000                        C  v_ram   ends
C060                           code    segment public 'ROM'
                               assume  cs:code, ds:nothing, es:nothing, ss:nothing

C060                           font_lo_8x16    label   byte                    ; 2048 bytes
                            C  include fontlo16.asm
                            C
C060                        C  fontlo16 proc near       ;   System Font Table for M24
                            C
C060  00 00 00 00 00 00     C          DB  00h,00h,00h,00h,00h,00h,00h,00h
      00 00
C068  00 00 00 00 00 00     C          DB  00h,00h,00h,00h,00h,00h,00h,00h     ;     0
      00 00
C070  00 00 7E 81 A5 81     C          DB  00h,00h,07eh,081h,0a5h,081h,081h,0bdh
      81 BD
C078  99 81 7E 00 00 00     C          DB  099h,081h,07eh,00h,00h,00h,00h,00h  ;     1
      00 00
C080  00 00 7E FF DB FF     C          DB  00h,00h,07eh,0ffh,0dbh,0ffh,0ffh,0c3h
      FF C3
C088  E7 E7 FF 7E 00 00     C          DB  0e7h,0e7h,0ffh,07eh,00h,00h,00h,00h ;     2
      00 00
C090  00 00 00 36 7F 7F     C          DB  00h,00h,00h,036h,07fh,07fh,07fh,07fh
      7F 7F
C098  3E 1C 08 00 00 00     C          DB  03eh,01ch,08h,00h,00h,00h,00h,00h   ;     3
      00 00
C0A0  00 00 00 08 1C 3E     C          DB  00h,00h,00h,08h,01ch,03eh,07fh,03eh
      7F 3E
C0A8  1C 08 00 00 00 00     C          DB  01ch,08h,00h,00h,00h,00h,00h,00h    ;     4
      00 00
C0B0  00 00 18 3C 3C E7     C          DB  00h,00h,018h,03ch,03ch,0e7h,0e7h,0e7h
      E7 E7
C0B8  18 18 3C 00 00 00     C          DB  018h,018h,03ch,00h,00h,00h,00h,00h  ;     5
      00 00
C0C0  00 00 18 3C 7E FF     C          DB  00h,00h,018h,03ch,07eh,0ffh,0ffh,07eh
      FF 7E
C0C8  18 18 3C 00 00 00     C          DB  018h,018h,03ch,00h,00h,00h,00h,00h  ;     6
      00 00
C0D0  00 00 00 00 00 18     C          DB  00h,00h,00h,00h,00h,018h,03ch,03ch
      3C 3C
C0D8  18 00 00 00 00 00     C          DB  018h,00h,00h,00h,00h,00h,00h,00h    ;     7
      00 00
C0E0  FF FF FF FF FF E7     C          DB  0ffh,0ffh,0ffh,0ffh,0ffh,0e7h,0c3h,0c3h
      C3 C3
C0E8  C3 E7 FF FF FF FF     C          DB  0c3h,0e7h,0ffh,0ffh,0ffh,0ffh,0ffh,0ffh     ;     8
      FF FF
C0F0  00 00 00 00 3C 24     C          DB  00h,00h,00h,00h,03ch,024h,042h,042h
      42 42
C0F8  24 3C 00 00 00 00     C          DB  024h,03ch,00h,00h,00h,00h,00h,00h   ;     9
      00 00
C100  FF FF FF FF C3 99     C          DB  0ffh,0ffh,0ffh,0ffh,0c3h,099h,0bdh,0bdh
      BD BD
C108  BD 99 C3 FF FF FF     C          DB  0bdh,099h,0c3h,0ffh,0ffh,0ffh,0ffh,0ffh     ;     a
      FF FF
C110  00 00 1F 07 0D 19     C          DB  00h,00h,01fh,07h,0dh,019h,078h,0cch
      78 CC
C118  CC CC 78 00 00 00     C          DB  0cch,0cch,078h,00h,00h,00h,00h,00h  ;     b
      00 00
C120  00 00 3C 66 66 66     C          DB  00h,00h,03ch,066h,066h,066h,03ch,018h
      3C 18
C128  7E 18 18 00 00 00     C          DB  07eh,018h,018h,00h,00h,00h,00h,00h  ;     c
      00 00
C130  00 00 18 14 12 12     C          DB  00h,00h,018h,014h,012h,012h,012h,014h
      12 14
C138  10 70 F0 F0 60 00     C          DB  010h,070h,0f0h,0f0h,060h,00h,00h,00h        ;     d
      00 00
C
```

```
C140  00 00 1F 11 1F 11      C          DB  00h,00h,01fh,011h,01fh,011h,011h,011h
      11 11                  C
C148  13 37 77 72 20 00      C          DB  013h,037h,077h,072h,020h,00h,00h,00h          ;      e
      00 00                  C
C150  00 00 18 18 DB 3C      C          DB  00h,00h,018h,018h,0dbh,03ch,0e7h,03ch
      E7 3C                  C
C158  DB 18 18 00 00 00      C          DB  0dbh,018h,018h,00h,00h,00h,00h,00h  ;      f
      00 00                  C
C160  00 00 40 60 70 7C      C          DB  00h,00h,040h,060h,070h,07ch,07fh,07ch
      7F 7C                  C
C168  70 60 40 00 00 00      C          DB  070h,060h,040h,00h,00h,00h,00h,00h  ;      10
      00 00                  C
C170  00 00 01 03 07 1F      C          DB  00h,00h,01h,03h,07h,01fh,07fh,01fh
      7F 1F                  C
C178  07 03 01 00 00 00      C          DB  07h,03h,01h,00h,00h,00h,00h,00h     ;      11
      00 00                  C
C180  00 00 18 3C 7E 18      C          DB  00h,00h,018h,03ch,07eh,018h,018h,018h
      18 18                  C
C188  7E 3C 18 00 00 00      C          DB  07eh,03ch,018h,00h,00h,00h,00h,00h  ;      12
      00 00                  C
C190  00 00 33 33 33 33      C          DB  00h,00h,033h,033h,033h,033h,033h,033h
      33 33                  C
C198  00 00 33 00 00 00      C          DB  00h,00h,033h,00h,00h,00h,00h,00h    ;      13
      00 00                  C
C1A0  00 00 7F DB DB DB      C          DB  00h,00h,07fh,0dbh,0dbh,0dbh,07bh,01bh
      7B 1B                  C
C1A8  1B 1B 1B 00 00 00      C          DB  01bh,01bh,01bh,00h,00h,00h,00h,00h  ;      14
      00 00                  C
C1B0  00 3E 63 30 1C 36      C          DB  00h,03eh,063h,030h,01ch,036h,063h,063h
      63 63                  C
C1B8  36 1C 06 63 3E 00      C          DB  036h,01ch,06h,063h,03eh,00h,00h,00h ;      15
      00 00                  C
C1C0  00 00 00 00 00 00      C          DB  00h,00h,00h,00h,00h,00h,00h,00h
      00 00                  C
C1C8  7E 7E 7E 00 00 00      C          DB  07eh,07eh,07eh,00h,00h,00h,00h,00h  ;      16
      00 00                  C
C1D0  00 00 18 3C 7E 18      C          DB  00h,00h,018h,03ch,07eh,018h,018h,018h
      18 18                  C
C1D8  7E 3C 18 FF 00 00      C          DB  07eh,03ch,018h,0ffh,00h,00h,00h,00h ;      17
      00 00                  C
C1E0  00 00 18 3C 7E 18      C          DB  00h,00h,018h,03ch,07eh,018h,018h,018h
      18 18                  C
C1E8  18 18 18 00 00 00      C          DB  018h,018h,018h,00h,00h,00h,00h,00h  ;      18
      00 00                  C
C1F0  00 00 18 18 18 18      C          DB  00h,00h,018h,018h,018h,018h,018h,018h
      18 18                  C
C1F8  7E 3C 18 00 00 00      C          DB  07eh,03ch,018h,00h,00h,00h,00h,00h  ;      19
      00 00                  C
C200  00 00 00 00 0C 06      C          DB  00h,00h,00h,00h,0ch,06h,07fh,06h
      7F 06                  C
C208  0C 00 00 00 00 00      C          DB  0ch,00h,00h,00h,00h,00h,00h,00h     ;      1a
      00 00                  C
C210  00 00 00 00 18 30      C          DB  00h,00h,00h,00h,018h,030h,07fh,030h
      7F 30                  C
C218  18 00 00 00 00 00      C          DB  018h,00h,00h,00h,00h,00h,00h,00h    ;      1b
      00 00                  C
C220  00 00 00 00 60 60      C          DB  00h,00h,00h,00h,060h,060h,060h,060h
      60 60                  C
C228  7F 7F 00 00 00 00      C          DB  07fh,07fh,00h,00h,00h,00h,00h,00h   ;      1c
      00 00                  C
C230  00 00 00 00 24 42      C          DB  00h,00h,00h,00h,024h,042h,0ffh,042h
      FF 42                  C
C238  24 00 00 00 00 00      C          DB  024h,00h,00h,00h,00h,00h,00h,00h    ;      1d
      00 00                  C
C240  00 00 00 00 00 00      C          DB  00h,00h,00h,00h,00h,00h,00h,018h
      00 18                  C
C248  3C 7E FF 00 00 00      C          DB  03ch,07eh,0ffh,00h,00h,00h,00h,00h  ;      1e
      00 00                  C
C250  00 00 00 00 00 FF      C          DB  00h,00h,00h,00h,00h,0ffh,07eh,03ch
      7E 3C                  C
C258  18 00 00 00 00 00      C          DB  018h,00h,00h,00h,00h,00h,00h,00h    ;      1f
      00 00                  C
C260  00 00 00 00 00 00      C          DB  00h,00h,00h,00h,00h,00h,00h,00h
      00 00                  C
C268  00 00 00 00 00 00      C          DB  00h,00h,00h,00h,00h,00h,00h,00h     ;' ' 20
      00 00                  C
C270  00 00 18 3C 3C 3C      C          DB  00h,00h,018h,03ch,03ch,03ch,018h,018h
      18 18                  C
C278  00 18 18 00 00 00      C          DB  00h,018h,018h,00h,00h,00h,00h,00h   ;'!' 21
      00 00                  C
C280  00 00 66 66 24 00      C          DB  00h,00h,066h,066h,024h,00h,00h,00h
      00 00                  C
C288  00 00 00 00 00 00      C          DB  00h,00h,00h,00h,00h,00h,00h,00h     ;'"' 22
      00 00                  C
C290  00 00 36 36 7F 36      C          DB  00h,00h,036h,036h,07fh,036h,036h,036h
      36 36                  C
C298  7F 36 36 00 00 00      C          DB  07fh,036h,036h,00h,00h,00h,00h,00h  ;'#' 23
      00 00                  C
C2A0  08 08 3E 63 60 60      C          DB  08h,08h,03eh,063h,060h,060h,03eh,03h
      3E 03                  C
C2A8  03 63 3E 08 08 00      C          DB  03h,063h,03eh,08h,08h,00h,00h,00h   ;'$' 24
      00 00                  C
C2B0  00 00 00 61 63 06      C          DB  00h,00h,00h,061h,063h,06h,0ch,018h
      0C 18                  C
C2B8  30 63 43 00 00 00      C          DB  030h,063h,043h,00h,00h,00h,00h,00h  ;'%' 25
      00 00                  C
C2C0  00 00 1C 36 36 1C      C          DB  00h,00h,01ch,036h,036h,01ch,03bh,06eh
      3B 6E                  C
C2C8  66 66 3B 00 00 00      C          DB  066h,066h,03bh,00h,00h,00h,00h,00h  ;'&' 26
      00 00                  C
```

```
C2D0  00 00 30 30 30 60      C          DB  00h,00h,030h,030h,030h,060h,00h,00h
      00 00                  C
C2D8  00 00 00 00 00 00      C          DB  00h,00h,00h,00h,00h,00h,00h,00h     ;''' 27
      00 00                  C
C2E0  00 00 0C 18 30 30      C          DB  00h,00h,0ch,018h,030h,030h,030h,030h
      30 30                  C
C2E8  30 18 0C 00 00 00      C          DB  030h,018h,0ch,00h,00h,00h,00h,00h   ;'(' 28
      00 00                  C
C2F0  00 00 30 18 0C 0C      C          DB  00h,00h,030h,018h,0ch,0ch,0ch,0ch
      0C 0C                  C
C2F8  0C 18 30 00 00 00      C          DB  0ch,018h,030h,00h,00h,00h,00h,00h   ;')' 29
      00 00                  C
C300  00 00 00 00 66 3C      C          DB  00h,00h,00h,00h,066h,03ch,0ffh,03ch
      FF 3C                  C
C308  66 00 00 00 00 00      C          DB  066h,00h,00h,00h,00h,00h,00h,00h    ;'*' 2a
      00 00                  C
C310  00 00 00 00 18 18      C          DB  00h,00h,00h,00h,018h,018h,07eh,018h
      7E 18                  C
C318  18 00 00 00 00 00      C          DB  018h,00h,00h,00h,00h,00h,00h,00h    ;'+' 2b
      00 00                  C
C320  00 00 00 00 00 00      C          DB  00h,00h,00h,00h,00h,00h,00h,00h
      00 00                  C
C328  18 18 18 30 00 00      C          DB  018h,018h,018h,030h,00h,00h,00h,00h ;',' 2c
      00 00                  C
C330  00 00 00 00 00 00      C          DB  00h,00h,00h,00h,00h,00h,07eh,00h
      7E 00                  C
C338  00 00 00 00 00 00      C          DB  00h,00h,00h,00h,00h,00h,00h,00h     ;'-' 2d
      00 00                  C
C340  00 00 00 00 00 00      C          DB  00h,00h,00h,00h,00h,00h,00h,00h
      00 00                  C
C348  00 18 18 00 00 00      C          DB  00h,018h,018h,00h,00h,00h,00h,00h   ;'.' 2e
      00 00                  C
C350  00 00 01 03 06 0C      C          DB  00h,00h,01h,03h,06h,0ch,018h,030h
      18 30                  C
C358  60 40 00 00 00 00      C          DB  060h,040h,00h,00h,00h,00h,00h,00h   ;'/' 2f
      00 00                  C
C360  00 00 3E 63 67 6F      C          DB  00h,00h,03eh,063h,067h,06fh,07bh,073h
      7B 73                  C
C368  63 63 3E 00 00 00      C          DB  063h,063h,03eh,00h,00h,00h,00h,00h  ;'0' 30
      00 00                  C
C370  00 00 0C 1C 3C 0C      C          DB  00h,00h,0ch,01ch,03ch,0ch,0ch,0ch
      0C 0C                  C
C378  0C 0C 3F 00 00 00      C          DB  0ch,0ch,03fh,00h,00h,00h,00h,00h    ;'1' 31
      00 00                  C
C380  00 00 3E 63 03 06      C          DB  00h,00h,03eh,063h,03h,06h,0ch,018h
      0C 18                  C
C388  30 63 7F 00 00 00      C          DB  030h,063h,07fh,00h,00h,00h,00h,00h  ;'2' 32
      00 00                  C
C390  00 00 3E 63 03 03      C          DB  00h,00h,03eh,063h,03h,03h,03eh,03h
      3E 03                  C
C398  03 63 3E 00 00 00      C          DB  03h,063h,03eh,00h,00h,00h,00h,00h   ;'3' 33
      00 00                  C
C3A0  00 00 06 0E 1E 36      C          DB  00h,00h,06h,0eh,01eh,036h,066h,07fh
      66 7F                  C
C3A8  06 06 0F 00 00 00      C          DB  06h,06h,0fh,00h,00h,00h,00h,00h     ;'4' 34
      00 00                  C
C3B0  00 00 7F 60 60 60      C          DB  00h,00h,07fh,060h,060h,060h,07eh,03h
      7E 03                  C
C3B8  03 63 3E 00 00 00      C          DB  03h,063h,03eh,00h,00h,00h,00h,00h   ;'5' 35
      00 00                  C
C3C0  00 00 1C 30 60 60      C          DB  00h,00h,01ch,030h,060h,060h,07eh,063h
      7E 63                  C
C3C8  63 63 3E 00 00 00      C          DB  063h,063h,03eh,00h,00h,00h,00h,00h  ;'6' 36
      00 00                  C
C3D0  00 00 7F 63 03 06      C          DB  00h,00h,07fh,063h,03h,06h,0ch,018h
      0C 18                  C
C3D8  18 18 18 00 00 00      C          DB  018h,018h,018h,00h,00h,00h,00h,00h  ;'7' 37
      00 00                  C
C3E0  00 00 3E 63 63 63      C          DB  00h,00h,03eh,063h,063h,063h,03eh,063h
      3E 63                  C
C3E8  63 63 3E 00 00 00      C          DB  063h,063h,03eh,00h,00h,00h,00h,00h  ;'8' 38
      00 00                  C
C3F0  00 00 3E 63 63 63      C          DB  00h,00h,03eh,063h,063h,063h,03fh,03h
      3F 03                  C
C3F8  03 06 1C 00 00 00      C          DB  03h,06h,01ch,00h,00h,00h,00h,00h    ;'9' 39
      00 00                  C
C400  00 00 00 18 18 00      C          DB  00h,00h,00h,018h,018h,00h,00h,00h
      00 00                  C
C408  18 18 00 00 00 00      C          DB  018h,018h,00h,00h,00h,00h,00h,00h   ;':' 3a
      00 00                  C
C410  00 00 00 18 18 00      C          DB  00h,00h,00h,018h,018h,00h,00h,00h
      00 00                  C
C418  18 18 30 00 00 00      C          DB  018h,018h,030h,00h,00h,00h,00h,00h  ;';' 3b
      00 00                  C
C420  00 00 06 0C 18 30      C          DB  00h,00h,06h,0ch,018h,030h,060h,030h
      60 30                  C
C428  18 0C 06 00 00 00      C          DB  018h,0ch,06h,00h,00h,00h,00h,00h    ;'<' 3c
      00 00                  C
C430  00 00 00 00 7E 00      C          DB  00h,00h,00h,00h,07eh,00h,00h,00h
      00 00                  C
C438  7E 00 00 00 00 00      C          DB  07eh,00h,00h,00h,00h,00h,00h,00h    ;'=' 3d
      00 00                  C
C440  00 00 60 30 18 0C      C          DB  00h,00h,060h,030h,018h,0ch,06h,0ch
      06 0C                  C
C448  18 30 60 00 00 00      C          DB  018h,030h,060h,00h,00h,00h,00h,00h  ;'>' 3e
      00 00                  C
C450  00 00 3E 63 63 06      C          DB  00h,00h,03eh,063h,063h,06h,0ch,0ch
      0C 0C                  C
C458  00 0C 0C 00 00 00      C          DB  00h,0ch,0ch,00h,00h,00h,00h,00h     ;'?' 3f
      00 00                  C
```

```
C460  00 00 3E 63 63 6F      C          DB  00h,00h,03eh,063h,063h,06fh,06fh,06fh
      6F 6F                  C
C468  6E 60 3E 00 00 00      C          DB  06eh,060h,03eh,00h,00h,00h,00h,00h  ;'@' 40
      00 00                  C
C470  00 00 08 1C 36 63      C          DB  00h,00h,08h,01ch,036h,063h,063h,07fh
      63 7F                  C
C478  63 63 63 00 00 00      C          DB  063h,063h,063h,00h,00h,00h,00h,00h  ;'A' 41
      00 00                  C
C480  00 00 7E 33 33 32      C          DB  00h,00h,07eh,033h,033h,032h,03eh,033h
      3E 33                  C
C488  33 33 7E 00 00 00      C          DB  033h,033h,07eh,00h,00h,00h,00h,00h  ;'B' 42
      00 00                  C
C490  00 00 1E 33 60 60      C          DB  00h,00h,01eh,033h,060h,060h,060h,060h
      60 60                  C
C498  60 33 1E 00 00 00      C          DB  060h,033h,01eh,00h,00h,00h,00h,00h  ;'C' 43
      00 00                  C
C4A0  00 00 7C 36 33 33      C          DB  00h,00h,07ch,036h,033h,033h,033h,033h
      33 33                  C
C4A8  33 36 7C 00 00 00      C          DB  033h,036h,07ch,00h,00h,00h,00h,00h  ;'D' 44
      00 00                  C
C4B0  00 00 7F 33 30 34      C          DB  00h,00h,07fh,033h,030h,034h,03eh,034h
      3C 34                  C
C4B8  30 33 7F 00 00 00      C          DB  030h,033h,07fh,00h,00h,00h,00h,00h  ;'E' 45
      00 00                  C
C4C0  00 00 7F 31 34 3C      C          DB  00h,00h,07fh,031h,034h,03eh,03eh,034h
      3C 34                  C
C4C8  30 30 78 00 00 00      C          DB  030h,030h,078h,00h,00h,00h,00h,00h  ;'F' 46
      00 00                  C
C4D0  00 00 1E 33 60 60      C          DB  00h,00h,01eh,033h,060h,060h,060h,06fh
      60 6F                  C
C4D8  63 33 1D 00 00 00      C          DB  063h,033h,01dh,00h,00h,00h,00h,00h  ;'G' 47
      00 00                  C
C4E0  00 00 63 63 63 63      C          DB  00h,00h,063h,063h,063h,063h,07fh,063h
      7F 63                  C
C4E8  63 63 63 00 00 00      C          DB  063h,063h,063h,00h,00h,00h,00h,00h  ;'H' 48
      00 00                  C
C4F0  00 00 3C 18 18 18      C          DB  00h,00h,03eh,018h,018h,018h,018h,018h
      18 18                  C
C4F8  18 18 3C 00 00 00      C          DB  018h,018h,03eh,00h,00h,00h,00h,00h  ;'I' 49
      00 00                  C
C500  00 00 0F 06 06 06      C          DB  00h,00h,0fh,06h,06h,06h,06h,06h
      06 06                  C
C508  66 66 3C 00 00 00      C          DB  066h,066h,03eh,00h,00h,00h,00h,00h  ;'J' 4a
      00 00                  C
C510  00 00 73 33 36 36      C          DB  00h,00h,073h,033h,036h,036h,03eh,036h
      3C 36                  C
C518  36 33 73 00 00 00      C          DB  036h,033h,073h,00h,00h,00h,00h,00h  ;'K' 4b
      00 00                  C
C520  00 00 78 30 30 30      C          DB  00h,00h,078h,030h,030h,030h,030h,030h
      30 30                  C
C528  30 33 7F 00 00 00      C          DB  030h,033h,07fh,00h,00h,00h,00h,00h  ;'L' 4c
      00 00                  C
C530  00 00 63 77 7F 6B      C          DB  00h,00h,063h,077h,07fh,06bh,063h,063h
      63 63                  C
C538  63 63 63 00 00 00      C          DB  063h,063h,063h,00h,00h,00h,00h,00h  ;'M' 4d
      00 00                  C
C540  00 00 63 73 7B 7F      C          DB  00h,00h,063h,073h,07bh,07fh,06fh,067h
      6F 67                  C
C548  63 63 63 00 00 00      C          DB  063h,063h,063h,00h,00h,00h,00h,00h  ;'N' 4e
      00 00                  C
C550  00 00 1C 36 63 63      C          DB  00h,00h,01ch,036h,063h,063h,063h,063h
      63 63                  C
C558  63 36 1C 00 00 00      C          DB  063h,036h,01ch,00h,00h,00h,00h,00h  ;'O' 4f
      00 00                  C
C560  00 00 7E 33 33 33      C          DB  00h,00h,07eh,033h,033h,033h,03eh,030h
      3E 30                  C
C568  30 30 78 00 00 00      C          DB  030h,030h,078h,00h,00h,00h,00h,00h  ;'P' 50
      00 00                  C
C570  00 00 1C 36 63 63      C          DB  00h,00h,01ch,036h,063h,063h,063h,063h
      63 63                  C
C578  6B 3E 1C 06 03 00      C          DB  06bh,03eh,01ch,06h,03h,00h,00h,00h  ;'Q' 51
      00 00                  C
C580  00 00 7E 33 33 33      C          DB  00h,00h,07eh,033h,033h,033h,03eh,036h
      3E 36                  C
C588  33 33 33 00 00 00      C          DB  033h,033h,033h,00h,00h,00h,00h,00h  ;'R' 52
      00 00                  C
C590  00 00 3E 63 63 30      C          DB  00h,00h,03eh,063h,063h,030h,01ch,06h
      1C 06                  C
C598  63 63 3E 00 00 00      C          DB  063h,063h,03eh,00h,00h,00h,00h,00h  ;'S' 53
      00 00                  C
C5A0  00 00 7E 5A 18 18      C          DB  00h,00h,07eh,05ah,018h,018h,018h,018h
      18 18                  C
C5A8  18 18 3C 00 00 00      C          DB  018h,018h,03eh,00h,00h,00h,00h,00h  ;'T' 54
      00 00                  C
C5B0  00 00 63 63 63 63      C          DB  00h,00h,063h,063h,063h,063h,063h,063h
      63 63                  C
C5B8  63 63 3E 00 00 00      C          DB  063h,063h,03eh,00h,00h,00h,00h,00h  ;'U' 55
      00 00                  C
C5C0  00 00 63 63 63 63      C          DB  00h,00h,063h,063h,063h,063h,063h,063h
      63 63                  C
C5C8  36 1C 08 00 00 00      C          DB  036h,01ch,08h,00h,00h,00h,00h,00h   ;'V' 56
      00 00                  C
C5D0  00 00 63 63 63 63      C          DB  00h,00h,063h,063h,063h,063h,063h,06bh
      63 6B                  C
C5D8  6B 7F 36 00 00 00      C          DB  06bh,07fh,036h,00h,00h,00h,00h,00h  ;'W' 57
      00 00                  C
C5E0  00 00 63 63 63 36      C          DB  00h,00h,063h,063h,063h,036h,01ch,036h
      1C 36                  C
C5E8  63 63 63 00 00 00      C          DB  063h,063h,063h,00h,00h,00h,00h,00h  ;'X' 58
      00 00                  C
```

```
C5F0  00 00 66 66 66 66      C          DB  00h,00h,066h,066h,066h,066h,066h,03ch
      66 3C                  C
C5F8  18 18 3C 00 00 00      C          DB  018h,018h,03ch,00h,00h,00h,00h,00h  ;'Y' 59
      00 00                  C
C600  00 00 7F 63 06 0C      C          DB  00h,00h,07fh,063h,06h,0ch,018h,030h
      18 30                  C
C608  60 63 7F 00 00 00      C          DB  060h,063h,07fh,00h,00h,00h,00h,00h  ;'Z' 5a
      00 00                  C
C610  00 00 3C 30 30 30      C          DB  00h,00h,03ch,030h,030h,030h,030h,030h
      30 30                  C
C618  30 30 3C 00 00 00      C          DB  030h,030h,03ch,00h,00h,00h,00h,00h  ;'[' 5b
      00 00                  C
C620  00 00 40 60 30 18      C          DB  00h,00h,040h,060h,030h,018h,0ch,06h
      0C 06                  C
C628  03 01 00 00 00 00      C          DB  03h,01h,00h,00h,00h,00h,00h,00h     ;'\' 5c
      00 00                  C
C630  00 00 3C 0C 0C 0C      C          DB  00h,00h,03ch,0ch,0ch,0ch,0ch,0ch
      0C 0C                  C
C638  0C 0C 3C 00 00 00      C          DB  0ch,0ch,03ch,00h,00h,00h,00h,00h    ;']' 5d
      00 00                  C
C640  08 1C 36 63 00 00      C          DB  08h,01ch,036h,063h,00h,00h,00h,00h
      00 00                  C
C648  00 00 00 00 00 00      C          DB  00h,00h,00h,00h,00h,00h,00h,00h     ;'^' 5e
      00 00                  C
C650  00 00 00 00 00 00      C          DB  00h,00h,00h,00h,00h,00h,00h,00h
      00 00                  C
C658  00 00 00 00 00 00      C          DB  00h,00h,00h,00h,00h,00h,07fh,00h    ;'_' 5f
      7F 00                  C
C660  18 18 0C 00 00 00      C          DB  018h,018h,0ch,00h,00h,00h,00h,00h
      00 00                  C
C668  00 00 00 00 00 00      C          DB  00h,00h,00h,00h,00h,00h,00h,00h     ;'`' 60
      00 00                  C
C670  00 00 00 00 00 3C      C          DB  00h,00h,00h,00h,00h,03ch,06h,03eh
      06 3E                  C
C678  66 66 3B 00 00 00      C          DB  066h,066h,03bh,00h,00h,00h,00h,00h  ;'a' 61
      00 00                  C
C680  00 00 70 30 30 3E      C          DB  00h,00h,070h,030h,030h,03eh,033h,033h
      33 33                  C
C688  33 33 6E 00 00 00      C          DB  033h,033h,06eh,00h,00h,00h,00h,00h  ;'b' 62
      00 00                  C
C690  00 00 00 00 00 3E      C          DB  00h,00h,00h,00h,00h,03eh,063h,060h
      63 60                  C
C698  60 63 3E 00 00 00      C          DB  060h,063h,03eh,00h,00h,00h,00h,00h  ;'o' 63
      00 00                  C
C6A0  00 00 0E 06 06 3E      C          DB  00h,00h,0eh,06h,06h,03eh,066h,066h
      66 66                  C
C6A8  66 66 3B 00 00 00      C          DB  066h,066h,03bh,00h,00h,00h,00h,00h  ;'d' 64
      00 00                  C
C6B0  00 00 00 00 00 3E      C          DB  00h,00h,00h,00h,00h,03eh,063h,07fh
      63 7F                  C
C6B8  60 63 3E 00 00 00      C          DB  060h,063h,03eh,00h,00h,00h,00h,00h  ;'e' 65
      00 00                  C
C6C0  00 00 1E 33 30 30      C          DB  00h,00h.01eh,033h,030h,030h,07eh,030h
      7C 30                  C
C6C8  30 30 78 00 00 00      C          DB  030h,030h,078h,00h,00h,00h,00h,00h  ;'f' 66
      00 00                  C
C6D0  00 00 00 00 00 3B      C          DB  00h,00h,00h,00h,00h,03bh,066h,066h
      66 66                  C
C6D8  66 66 3E 06 66 3C      C          DB  066h,066h,03eh,06h,066h,03eh,00h,00h        ;'g' 67
      00 00                  C
C6E0  00 00 70 30 36 3B      C          DB  00h,00h,070h,030h,036h,03bh,033h,033h
      33 33                  C
C6E8  33 33 73 00 00 00      C          DB  033h,033h,073h,00h,00h,00h,00h,00h  ;'h' 68
      00 00                  C
C6F0  00 00 0C 0C 00 1C      C          DB  00h,00h,0ch,0ch,00h,01ch,0ch,0ch
      0C 0C                  C
C6F8  0C 0C 1E 00 00 00      C          DB  0ch,0ch,01eh,00h,00h,00h,00h,00h    ;'i' 69
      00 00                  C
C700  00 00 0C 0C 00 1C      C          DB  00h,00h,0eh,0ch,00h,01ch,0ch,0ch
      0C 0C                  C
C708  0C 0C 0C 0C CC 78      C          DB  0ch,0ch,0ch,0ch,0cch,078h,00h,00h   ;'j' 6a
      00 00                  C
C710  00 00 70 30 30 33      C          DB  00h,00h,070h,030h,030h,033h,036h,03ch
      36 3C                  C
C718  36 33 73 00 00 00      C          DB  036h,033h,073h,00h,00h,00h,00h,00h  ;'k' 6b
      00 00                  C
C720  00 00 1C 0C 0C 0C      C          DB  00h,00h,01ch,0ch,0ch,0ch,0ch,0ch
      0C 0C                  C
C728  0C 0C 1E 00 00 00      C          DB  0ch,0ch,01eh,00h,00h,00h,00h,00h    ;'l' 6c
      00 00                  C
C730  00 00 00 00 00 66      C          DB  00h,00h,00h,00h,00h,066h,07fh,06bh
      7F 6B                  C
C738  6B 6B 6B 00 00 00      C          DB  06bh,06bh,06bh,00h,00h,00h,00h,00h  ;'m' 6d
      00 00                  C
C740  00 00 00 00 00 6E      C          DB  00h,00h,00h,00h,00h,06eh,033h,033h
      33 33                  C
C748  33 33 33 00 00 00      C          DB  033h,033h,033h,00h,00h,00h,00h,00h  ;'n' 6e
      00 00                  C
C750  00 00 00 00 00 3E      C          DB  00h,00h,00h,00h,00h,03eh,063h,063h
      63 63                  C
C758  63 63 3E 00 00 00      C          DB  063h,063h,03eh,00h,00h,00h,00h,00h  ;'o' 6f
      00 00                  C
C760  00 00 00 00 00 6E      C          DB  00h,00h,00h,00h,00h,06eh,033h,033h
      33 33                  C
C768  33 33 3E 30 30 78      C          DB  033h,033h,03eh,030h,030h,078h,00h,00h       ;'p' 70
      00 00                  C
C770  00 00 00 00 00 3B      C          DB  00h,00h,00h,00h,00h,03bh,066h,066h
      66 66                  C
C778  66 66 3E 06 06 0F      C          DB  066h,066h,03eh,06h,06h,0fh,00h,00h  ;'q' 71
      00 00                  C
```

```
C780  00 00 00 00 00 6E      C          DB  00h,00h,00h,00h,00h,06eh,03bh,030h
      3B 30                  C
C788  30 30 78 00 00 00      C          DB  030h,030h,078h,00h,00h,00h,00h,00h  ;'r' 72
      00 00                  C
C790  00 00 00 00 00 3E      C          DB  00h,00h,00h,00h,00h,03eh,063h,038h
      63 38                  C
C798  0E 63 3E 00 00 00      C          DB  0eh,063h,03eh,00h,00h,00h,00h,00h   ;'s' 73
      00 00                  C
C7A0  00 00 00 08 18 7E      C          DB  00h,00h,00h,08h,018h,07eh,018h,018h
      18 18                  C
C7A8  18 1B 0E 00 00 00      C          DB  018h,01bh,0eh,00h,00h,00h,00h,00h   ;'t' 74
      00 00                  C
C7B0  00 00 00 00 00 66      C          DB  00h,00h,00h,00h,00h,066h,066h,066h
      66 66                  C
C7B8  66 66 3B 00 00 00      C          DB  066h,066h,03bh,00h,00h,00h,00h,00h  ;'u' 75
      00 00                  C
C7C0  00 00 00 00 00 63      C          DB  00h,00h,00h,00h,00h,063h,063h,063h
      63 63                  C
C7C8  36 1C 08 00 00 00      C          DB  036h,01ch,08h,00h,00h,00h,00h,00h   ;'v' 76
      00 00                  C
C7D0  00 00 00 00 00 63      C          DB  00h,00h,00h,00h,00h,063h,06bh,06bh
      6B 6B                  C
C7D8  6B 7F 36 00 00 00      C          DB  06bh,07fh,036h,00h,00h,00h,00h,00h  ;'w' 77
      00 00                  C
C7E0  00 00 00 00 00 63      C          DB  00h,00h,00h,00h,00h,063h,036h,01ch
      36 1C                  C
C7E8  1C 36 63 00 00 00      C          DB  01ch,036h,063h,00h,00h,00h,00h,00h  ;'x' 78
      00 00                  C
C7F0  00 00 00 00 00 63      C          DB  00h,00h,00h,00h,00h,063h,066h,066h
      66 66                  C
C7F8  66 66 3E 06 66 3C      C          DB  066h,066h,03eh,06h,066h,03ch,00h,00h        ;'y' 79
      00 00                  C
C800  00 00 00 00 00 7F      C          DB  00h,00h,00h,00h,00h,07fh,066h,0ch
      66 0C                  C
C808  18 33 7F 00 00 00      C          DB  018h,033h,07fh,00h,00h,00h,00h,00h  ;'z' 7a
      00 00                  C
C810  00 00 0E 18 18 18      C          DB  00h,00h,0eh,018h,018h,018h,070h,018h
      70 18                  C
C818  18 18 0E 00 00 00      C          DB  018h,018h,0eh,00h,00h,00h,00h,00h   ;'{' 7b
      00 00                  C
C820  00 00 18 18 18 18      C          DB  00h,00h,018h,018h,018h,018h,00h,018h
      00 18                  C
C828  18 18 18 00 00 00      C          DB  018h,018h,018h,00h,00h,00h,00h,00h  ;'|' 7c
      00 00                  C
C830  00 00 70 18 18 18      C          DB  00h,00h,070h,018h,018h,018h,0eh,018h
      0E 18                  C
C838  18 18 70 00 00 00      C          DB  018h,018h,070h,00h,00h,00h,00h,00h  ;'}' 7d
      00 00                  C
C840  00 00 3B 6E 00 00      C          DB  00h,00h,03bh,06eh,00h,00h,00h,00h
      00 00                  C
C848  00 00 00 00 00 00      C          DB  00h,00h,00h,00h,00h,00h,00h,00h     ;'~' 7e
      00 00                  C
C850  00 00 00 00 08 1C      C          DB  00h,00h,00h,00h,08h,01eh,036h,063h
      36 63                  C
C858  63 7F 00 00 00 00      C          DB  063h,07fh,00h,00h,00h,00h,00h,00h   ;'~' 7f
      00 00                  C
                             C  ;End of font matrix
                             C
C860                         C  fontlo16 endp

C860                            font_hi_8x8     label   byte                    ; 1024 bytes
                             C  include fonthi8.asm
                             C
C860                         C  fonthi8 proc near
                             C  ;       SystemFont      ; <hi_mediumres> (M24) 8 x 8 font table for M24
                             C
C860  1E 33 60 33            C          DB  01eh,033h,060h,033h
C864  1E 08 04 08            C          DB  01eh,08h,04h,08h    ;     0
C868  66 00 66 66            C          DB  066h,00h,066h,066h
C86C  66 66 66 3B            C          DB  066h,066h,066h,03bh ;     1
C870  06 0C 00 1E            C          DB  06h,0ch,00h,01eh
C874  33 3F 30 1E            C          DB  033h,03fh,030h,01eh ;     2
C878  08 14 00 3C            C          DB  08h,014h,00h,03ch
C87C  06 3E 66 3B            C          DB  06h,03eh,066h,03bh  ;     3
C880  00 36 00 3C            C          DB  00h,036h,00h,03ch
C884  06 3E 66 3B            C          DB  06h,03eh,066h,03bh  ;     4
C888  30 18 00 3C            C          DB  030h,018h,00h,03ch
C88C  06 3E 66 3B            C          DB  06h,03eh,066h,03bh  ;     5
C890  08 14 08 3C            C          DB  08h,014h,08h,03ch
C894  06 3E 66 3B            C          DB  06h,03eh,066h,03bh  ;     6
C898  3C 66 60 66            C          DB  03ch,066h,060h,066h
C89C  3C 08 04 08            C          DB  03ch,08h,04h,08h    ;     7
C8A0  18 34 00 3E            C          DB  018h,034h,00h,03eh
C8A4  63 7F 60 3E            C          DB  063h,07fh,060h,03eh ;     8
C8A8  00 36 00 3E            C          DB  00h,036h,00h,03eh
C8AC  63 7F 60 3E            C          DB  063h,07fh,060h,03eh ;     9
C8B0  30 18 00 3C            C          DB  030h,018h,00h,03ch
C8B4  66 7E 60 3C            C          DB  066h,07eh,060h,03ch ;     a
C8B8  00 36 00 1C            C          DB  00h,036h,00h,01ch
C8BC  0C 0C 0C 1E            C          DB  0ch,0ch,0ch,01eh    ;     b
C8C0  08 14 00 1C            C          DB  08h,014h,00h,01ch
C8C4  0C 0C 0C 1E            C          DB  0ch,0ch,0ch,01eh    ;     c
C8C8  30 18 00 1C            C          DB  030h,018h,00h,01ch
C8CC  0C 0C 0C 1E            C          DB  0ch,0ch,0ch,01eh    ;     d
C8D0  66 00 18 3C            C          DB  066h,00h,018h,03ch
C8D4  66 7E 66 66            C          DB  066h,07eh,066h,066h ;     e
C8D8  18 24 18 3C            C          DB  018h,024h,018h,03ch
C8DC  66 7E 66 66            C          DB  066h,07eh,066h,066h ;     f
C8E0  0C 18 7F 31            C          DB  0ch,018h,07fh,031h
C8E4  3C 30 31 7F            C          DB  03ch,030h,031h,07fh ;     10
```

```
C8E8  00 36 4B 0F          C       DB  00h,036h,04bh,0fh
C8EC  38 58 5D 36          C       DB  038h,058h,05dh,036h ;       11
C8F0  00 1F 28 49          C       DB  00h,01fh,028h,049h
C8F4  AF 79 A8 AF          C       DB  0afh,079h,0A8h,0AFh ;       12
C8F8  08 14 00 1C          C       DB  08h,014h,00h,01ch
C8FC  36 63 36 1C          C       DB  036h,063h,036h,01ch ;       13
C900  00 36 00 1C          C       DB  00h,036h,00h,01ch
C904  36 63 36 1C          C       DB  036h,063h,036h,01ch ;       14
C908  30 18 00 1C          C       DB  030h,018h,00h,01ch
C90C  36 63 36 1C          C       DB  036h,063h,036h,01ch ;       15
C910  08 14 00 66          C       DB  08h,014h,00h,066h
C914  66 66 66 3B          C       DB  066h,066h,066h,03bh ;       16
C918  30 18 00 66          C       DB  030h,018h,00h,066h
C91C  66 66 66 3B          C       DB  066h,066h,066h,03bh ;       17
C920  66 00 66 66          C       DB  066h,00h,066h,066h
C924  66 3E 06 78          C       DB  066h,03eh,06h,078h  ;       18
C928  00 36 00 1C          C       DB  00h,036h,00h,01ch
C92C  36 63 36 1C          C       DB  036h,063h,036h,01ch ;       19
C930  00 63 00 63          C       DB  00h,063h,00h,063h
C934  63 63 63 3E          C       DB  063h,063h,063h,03eh ;       1a
C938  08 08 3E 40          C       DB  08h,08h,03eh,040h
C93C  40 3E 08 08          C       DB  040h,03eh,08h,08h   ;       1b
C940  0E 1B 18 3C          C       DB  0eh,01bh,018h,03ch
C944  18 18 39 7E          C       DB  018h,018h,039h,07eh ;       1c
C948  22 14 08 3E          C       DB  022h,014h,08h,03eh
C94C  08 3E 08 08          C       DB  08h,03eh,08h,08h    ;       1d
C950  70 48 72 42          C       DB  070h,048h,072h,042h
C954  47 42 42 01          C       DB  047h,042h,042h,01h  ;       1e
C958  0C 12 10 38          C       DB  0ch,012h,010h,038h
C95C  10 10 50 20          C       DB  010h,010h,050h,020h ;       1f
C960  06 0C 00 3C          C       DB  06h,0ch,00h,03ch
C964  06 3E 66 3B          C       DB  06h,03eh,066h,03bh  ;' ' 20
C968  06 0C 00 1C          C       DB  06h,0ch,00h,01ch
C96C  0C 0C 0C 18          C       DB  0ch,0ch,0ch,01eh    ;'!' 21
C970  06 0C 00 1C          C       DB  06h,0ch,00h,01ch
C974  36 63 36 1C          C       DB  036h,063h,036h,01ch ;'"' 22
C978  06 0C 00 66          C       DB  06h,0ch,00h,066h
C97C  66 66 66 3B          C       DB  066h,066h,066h,03bh ;'#' 23
C980  33 CC 00 7C          C       DB  033h,0cch,00h,07ch
C984  66 66 66 66          C       DB  066h,066h,066h,066h ;'$' 24
C988  33 CC 63 73          C       DB  033h,0cch,063h,073h
C98C  7B 6F 67 63          C       DB  07bh,06fh,067h,063h ;'%' 25
C990  00 3C 06 3E          C       DB  00h,03ch,06h,03eh
C994  66 3B 00 7F          C       DB  066h,03bh,00h,07fh  ;'&' 26
C998  00 1C 36 63          C       DB  00h,01ch,036h,063h
C99C  36 1C 00 7F          C       DB  036h,01ch,00h,07fh  ;''' 27
C9A0  18 00 18 18          C       DB  018h,00h,018h,018h
C9A4  30 46 3C 00          C       DB  030h,046h,03ch,00h  ;'(' 28
C9A8  00 00 7F 60          C       DB  00h,00h,07fh,060h
C9AC  60 00 00 00          C       DB  060h,00h,00h,00h    ;')' 29
C9B0  00 00 7F 03          C       DB  00h,00h,07fh,03h
C9B4  03 00 00 00          C       DB  03h,00h,00h,00h     ;'*' 2a
C9B8  20 62 24 28          C       DB  020h,062h,024h,028h
C9BC  16 21 42 07          C       DB  016h,021h,042h,07h  ;'+' 2b
C9C0  20 62 24 2A          C       DB  020h,062h,024h,02ah
C9C4  16 2A 4F 02          C       DB  016h,02ah,04fh,02h  ;',' 2c
C9C8  18 18 00 18          C       DB  018h,018h,00h,018h
C9CC  18 3C 3C 18          C       DB  018h,03ch,03ch,018h ;'-' 2d
C9D0  00 1B 36 6C          C       DB  00h,01bh,036h,06ch
C9D4  36 1B 00 00          C       DB  036h,01bh,00h,00h   ;'.' 2e
C9D8  00 6C 36 1B          C       DB  00h,06ch,036h,01bh
C9DC  36 6C 00 00          C       DB  036h,06ch,00h,00h   ;'/' 2f
C9E0  22 88 22 88          C       DB  022h,088h,022h,088h
C9E4  22 88 22 88          C       DB  022h,088h,022h,088h ;'0' 30
C9E8  55 AA 55 AA          C       DB  055h,0aah,055h,0aah
C9EC  55 AA 55 AA          C       DB  055h,0aah,055h,0aah ;'1' 31
C9F0  CE 77 CE 77          C       DB  0ceh,077h,0ceh,077h
C9F4  CE 77 CE 77          C       DB  0ceh,077h,0ceh,077h ;'2' 32
C9F8  18 18 18 18          C       DB  018h,018h,018h,018h
C9FC  18 18 18 18          C       DB  018h,018h,018h,018h ;'3' 33
CA00  18 18 18 F8          C       DB  018h,018h,018h,0f8h
CA04  18 18 18 18          C       DB  018h,018h,018h,018h ;'4' 34
CA08  18 18 F8 18          C       DB  018h,018h,0f8h,018h
CA0C  F8 18 18 18          C       DB  0f8h,018h,018h,018h ;'5' 35
CA10  36 36 36 F6          C       DB  036h,036h,036h,0f6h
CA14  36 36 36 36          C       DB  036h,036h,036h,036h ;'6' 36
CA18  00 00 00 FE          C       DB  00h,00h,00h,0feh
CA1C  36 36 36 36          C       DB  036h,036h,036h,036h ;'7' 37
CA20  00 00 F8 18          C       DB  00h,00h,0f8h,018h
CA24  F8 18 18 18          C       DB  0f8h,018h,018h,018h ;'8' 38
CA28  36 36 F6 36          C       DB  036h,036h,0f6h,036h
CA2C  F6 36 36 36          C       DB  0f6h,036h,036h,036h ;'9' 39
CA30  36 36 36 36          C       DB  036h,036h,036h,036h
CA34  36 36 36 36          C       DB  036h,036h,036h,036h ;':' 3a
CA38  00 00 FE 06          C       DB  00h,00h,0feh,06h
CA3C  F6 36 36 36          C       DB  0f6h,036h,036h,036h ;';' 3b
CA40  36 36 F6 06          C       DB  036h,036h,0f6h,06h
CA44  FE 00 00 00          C       DB  0feh,00h,00h,00h    ;'<' 3c
CA48  36 36 36 FE          C       DB  036h,036h,036h,0feh
CA4C  00 00 00 00          C       DB  00h,00h,00h,00h     ;'=' 3d
CA50  18 18 F8 18          C       DB  018h,018h,0f8h,018h
CA54  F8 00 00 00          C       DB  0f8h,00h,00h,00h    ;'>' 3e
CA58  00 00 00 F8          C       DB  00h,00h,00h,0f8h
CA5C  18 18 18 18          C       DB  018h,018h,018h,018h ;'?' 3f
CA60  18 18 18 1F          C       DB  018h,018h,018h,01fh
CA64  00 00 00 00          C       DB  00h,00h,00h,00h     ;'@' 40
CA68  18 18 18 FF          C       DB  018h,018h,018h,0ffh
CA6C  00 00 00 00          C       DB  00h,00h,00h,00h     ;'A' 41
CA70  00 00 00 FF          C       DB  00h,00h,00h,0ffh
CA74  18 18 18 18          C       DB  018h,018h,018h,018h ;'B' 42
```

```
CA78  18 18 18 1F          C          DB  018h,018h,018h,01fh
CA7C  18 18 18 18          C          DB  018h,018h,018h,018h ;'C' 43
CA80  00 00 00 FF          C          DB  00h,00h,00h,0ffh
CA84  00 00 00 00          C          DB  00h,00h,00h,00h     ;'D' 44
CA88  18 18 18 FF          C          DB  018h,018h,018h,0ffh
CA8C  18 18 18 18          C          DB  018h,018h,018h,018h ;'E' 45
CA90  18 18 1F 18          C          DB  018h,018h,01fh,018h
CA94  1F 18 18 18          C          DB  01fh,018h,018h,018h ;'F' 46
CA98  36 36 36 37          C          DB  036h,036h,036h,037h
CA9C  36 36 36 36          C          DB  036h,036h,036h,036h ;'G' 47
CAA0  36 36 37 30          C          DB  036h,036h,037h,030h
CAA4  3F 00 00 00          C          DB  03fh,00h,00h,00h    ;'H' 48
CAA8  00 00 3F 30          C          DB  00h,00h,03fh,030h
CAAC  37 36 36 36          C          DB  037h,036h,036h,036h ;'I' 49
CAB0  36 36 F7 00          C          DB  036h,036h,0f7h,00h
CAB4  FF 00 00 00          C          DB  0ffh,00h,00h,00h    ;'J' 4a
CAB8  00 00 FF 00          C          DB  00h,00h,0ffh,00h
CABC  F7 36 36 36          C          DB  0f7h,036h,036h,036h ;'K' 4b
CAC0  36 36 37 30          C          DB  036h,036h,037h,030h
CAC4  37 34 34 34          C          DB  037h,034h,034h,034h ;'L' 4c
CAC8  00 00 FF 00          C          DB  00h,00h,0ffh,00h
CACC  FF 00 00 00          C          DB  0ffh,00h,00h,00h    ;'M' 4d
CAD0  36 36 F7 00          C          DB  036h,036h,0f7h,00h
CAD4  F7 36 36 36          C          DB  0f7h,036h,036h,036h ;'N' 4e
CAD8  18 18 FF 00          C          DB  018h,018h,0ffh,00h
CADC  FF 00 00 00          C          DB  0ffh,00h,00h,00h    ;'O' 4f
CAE0  36 36 36 FF          C          DB  036h,036h,036h,0ffh
CAE4  00 00 00 00          C          DB  00h,00h,00h,00h     ;'P' 50
CAE8  00 00 FF 00          C          DB  00h,00h,0ffh,00h
CAEC  FF 18 18 18          C          DB  0ffh,018h,018h,018h ;'Q' 51
CAF0  00 00 00 FF          C          DB  00h,00h,00h,0ffh
CAF4  36 36 36 36          C          DB  036h,036h,036h,036h ;'R' 52
CAF8  36 36 36 3F          C          DB  036h,036h,036h,03fh
CAFC  00 00 00 00          C          DB  00h,00h,00h,00h     ;'S' 53
CB00  18 18 1F 18          C          DB  018h,018h,01fh,018h
CB04  1F 00 00 00          C          DB  01fh,00h,00h,00h    ;'T' 54
CB08  00 00 1F 18          C          DB  00h,00h,01fh,018h
CB0C  1F 18 18 18          C          DB  01fh,018h,018h,018h ;'U' 55
CB10  00 00 00 3F          C          DB  00h,00h,00h,03fh
CB14  36 36 36 36          C          DB  036h,036h,036h,036h ;'V' 56
CB18  36 36 36 FF          C          DB  036h,036h,036h,0ffh
CB1C  36 36 36 36          C          DB  036h,036h,036h,036h ;'W' 57
CB20  18 18 FF 18          C          DB  018h,018h,0ffh,018h
CB24  FF 18 18 18          C          DB  0ffh,018h,018h,018h ;'X' 58
CB28  18 18 18 F8          C          DB  018h,018h,018h,0f8h
CB2C  00 00 00 00          C          DB  00h,00h,00h,00h     ;'Y' 59
CB30  00 00 00 1F          C          DB  00h,00h,00h,01fh
CB34  18 18 18 18          C          DB  018h,018h,018h,018h ;'Z' 5a
CB38  FF FF FF FF          C          DB  0ffh,0ffh,0ffh,0ffh
CB3C  FF FF FF FF          C          DB  0ffh,0ffh,0ffh,0ffh ;'[' 5b
CB40  00 00 00 00          C          DB  00h,00h,00h,00h
CB44  FF FF FF FF          C          DB  0ffh,0ffh,0ffh,0ffh ;'\' 5c
CB48  F0 F0 F0 F0          C          DB  0f0h,0f0h,0f0h,0f0h
CB4C  F0 F0 F0 F0          C          DB  0f0h,0f0h,0f0h,0f0h ;']' 5d
CB50  0F 0F 0F 0F          C          DB  0fh,0fh,0fh,0fh
CB54  0F 0F 0F 0F          C          DB  0fh,0fh,0fh,0fh     ;'^' 5e
CB58  FF FF FF FF          C          DB  0ffh,0ffh,0ffh,0ffh
CB5C  00 00 00 00          C          DB  00h,00h,00h,00h     ;'_' 5f
CB60  00 00 3B 6E          C          DB  00h,00h,03bh,06eh
CB64  6C 6C 6E 3B          C          DB  06ch,06ch,06eh,03bh ;'`' 60
CB68  3E 63 7E 63          C          DB  03eh,063h,07eh,063h
CB6C  63 7E 60 60          C          DB  063h,07eh,060h,060h ;'a' 61
CB70  00 7E 32 30          C          DB  00h,07eh,032h,030h
CB74  30 30 30 78          C          DB  030h,030h,030h,078h ;'b' 62
CB78  00 01 7F 54          C          DB  00h,01h,07fh,054h
CB7C  14 14 14 14          C          DB  014h,014h,014h,014h ;'c' 63
CB80  00 7F 61 30          C          DB  00h,07fh,061h,030h
CB84  18 30 61 7F          C          DB  018h,030h,061h,07fh ;'d' 64
CB88  00 00 1F 34          C          DB  00h,00h,01fh,034h
CB8C  62 62 34 18          C          DB  062h,062h,034h,018h ;'e' 65
CB90  00 36 36 36          C          DB  00h,036h,036h,036h
CB94  3C 30 30 60          C          DB  03ch,030h,030h,060h ;'f' 66
CB98  00 3B 6E 0C          C          DB  00h,03bh,06eh,0ch
CB9C  0C 0C 00 00          C          DB  0ch,0ch,00h,00h     ;'g' 67
CBA0  66 3C 18 3C          C          DB  066h,03ch,018h,03ch ;'h' 68
CBA4  00 1C 36 63          C          DB  00h,01ch,036h,063h
CBA8  7F 63 36 1C          C          DB  07fh,063h,036h,01ch ;'i' 69
CBAC  00 1C 36 63          C          DB  00h,01ch,036h,063h
CBB0  63 36 36 77          C          DB  063h,036h,036h,077h ;'j' 6a
CBB4  1E 30 18 0C          C          DB  01eh,030h,018h,0ch
CBB8  3E 66 66 3C          C          DB  03eh,066h,066h,03ch ;'k' 6b
CBBC  00 00 76 DB          C          DB  00h,00h,076h,0dbh
CBC0  DB 6E 00 00          C          DB  0dbh,06eh,00h,00h   ;'l' 6c
CBC4  03 06 7E DB          C          DB  03h,06h,07eh,0dbh
CBC8  DB 7E 60 C0          C          DB  0dbh,07eh,060h,0c0h ;'m' 6d
CBCC  00 1C 30 60          C          DB  00h,01ch,030h,060h
CBD0  7C 60 30 1C          C          DB  07ch,060h,030h,01ch ;'n' 6e
CBD4  00 00 3E 63          C          DB  00h,00h,03eh,063h
CBD8  63 63 63 63          C          DB  063h,063h,063h,063h ;'o' 6f
CBDC  00 00 3E 00          C          DB  00h,00h,03eh,00h
CBE0  3E 00 3E 00          C          DB  03eh,00h,03eh,00h   ;'p' 70
CBE4  18 18 7E 18          C          DB  018h,018h,07eh,018h
CBE8  18 00 7E 00          C          DB  018h,00h,07eh,00h   ;'q' 71
CBEC  18 0C 06 0C          C          DB  018h,0ch,06h,0ch
CBF0  18 00 3E 00          C          DB  018h,00h,03eh,00h   ;'r' 72
CBF4  0C 18 30 18          C          DB  0ch,018h,030h,018h
CBF8  0C 00 3E 00          C          DB  0ch,00h,03eh,00h    ;'s' 73
CBFC  0E 1B 1B 18          C          DB  0eh,01bh,01bh,018h
CC00  18 18 18 18          C          DB  018h,018h,018h,018h ;'t' 74
CC04  18 18 18 18          C          DB  018h,018h,018h,018h
```

```
CC0C  18 D8 D8 70          C          DB   018h,0d8h,0d8h,070h ;'u' 75
CC10  00 18 18 00          C          DB   00h,018h,018h,00h
CC14  3E 00 18 18          C          DB   03eh,00h,018h,018h  ;'v' 76
CC18  00 00 3B 6E          C          DB   00h,00h,03bh,06eh
CC1C  00 3B 6E 00          C          DB   00h,03bh,06eh,00h   ;'w' 77
CC20  00 38 6C 6C          C          DB   00h,038h,06ch,06ch
CC24  38 00 00 00          C          DB   038h,00h,00h,00h    ;'x' 78
CC28  00 18 3C 3C          C          DB   00h,018h,03ch,03ch
CC2C  18 00 00 00          C          DB   018h,00h,00h,00h    ;'y' 79
CC30  00 00 00 18          C          DB   00h,00h,00h,018h
CC34  18 00 00 00          C          DB   018h,00h,00h,00h    ;'z' 7a
CC38  0F 0C 0C 0C          C          DB   0fh,0ch,0ch,0ch
CC3C  0C 6C 3C 1C          C          DB   0ch,06ch,03ch,01ch  ;'{' 7b
CC40  6C 36 36 36          C          DB   06ch,036h,036h,036h
CC44  36 00 00 00          C          DB   036h,00h,00h,00h    ;'|' 7c
CC48  08 14 04 08          C          DB   08h,014h,04h,08h
CC4C  10 1C 00 00          C          DB   010h,01ch,00h,00h   ;'}' 7d
CC50  00 1C 1C 1C          C          DB   00h,01ch,01ch,01ch
CC54  1C 1C 00 00          C          DB   01ch,01ch,00h,00h   ;'~' 7e
CC58  00 00 00 00          C          DB   00h,00h,00h,00h
CC5C  00 00 00 00          C          DB   00h,00h,00h,00h     ;     7f
                           C  ;End of font matrix
                           C
CC60                       C  fonth18 endp

CC60                          code    ends

                           C  include kbdata.asm
                           C
                           C  ;=============================================================================
                           C  ;       Filename:       kb.data:        USA-ASCII
                           C  ;
                           C  ;       This module includes the keyboard scan code translation data
                           C  ;       for different keyboards.
                           C  ;
                           C  ;=============================================================================
                           C
CC60                       C  code    segment public 'ROM'
                           C          assume  cs:code, ds:nothing, es:nothing, ss:nothing
                           C
CC60                       C  kb_data1        proc
                           C
                           C  ;-----------------------------------------------------------------------
                           C  ;       special_cases
                           C  ;-----------------------------------------------------------------------
                           C
= 00C0                     C  kbins   equ     0C0h    ; kb_insert_lock        (min_case)
= 00C1                     C  kbcap   equ     0C1h    ; kb_caps_lock
= 00C2                     C  kbnum   equ     0C2h    ; kb_num_lock
= 00C3                     C  kbscr   equ     0C3h    ; kb_scroll_lock
= 00C4                     C  kbalt   equ     0C4h    ; kb_alt_lock
= 00C5                     C  kbctl   equ     0C5h    ; kb_control_lock
= 00C6                     C  kblsh   equ     0C6h    ; kb_l_shift_lock
= 00C7                     C  kbrsh   equ     0C7h    ; kb_r_shift_lock
                           C
= 00C8                     C  kbres   equ     0C8h    ; kb_reset              (mid_case)
= 00C9                     C  kbbrk   equ     0C9h    ; kb_break
= 00CA                     C  pause   equ     0CAh    ; kb_pause
= 00CB                     C  kbprt   equ     0CBh    ; kb_print_screen
= 00CC                     C  kbnul   equ     0CCh    ; kb_null
= 00CD                     C  kNONE   equ     0CDh    ; kb_none
                           C
= 00CE                     C  kdec9   equ     0CEh    ; kb_alt_dec_9
= 00CF                     C  kdec8   equ     0CFh    ; kb_alt_dec_8
= 00D0                     C  kdec7   equ     0D0h    ; kb_alt_dec_7
= 00D1                     C  kdec6   equ     0D1h    ; kb_alt_dec_6
= 00D2                     C  kdec5   equ     0D2h    ; kb_alt_dec_5
= 00D3                     C  kdec4   equ     0D3h    ; kb_alt_dec_4
= 00D4                     C  kdec3   equ     0D4h    ; kb_alt_dec_3
= 00D5                     C  kdec2   equ     0D5h    ; kb_alt_dec_2
= 00D6                     C  kdec1   equ     0D6h    ; kb_alt_dec_1
= 00D7                     C  kdec0   equ     0D7h    ; kb_alt_dec_0
                           C
= 00D8                     C  kdbl0   equ     0D8h    ; kb_double_zero        (max_case)
                           C
                           C  ;-----------------------------------------------------------------------
                           C  ;       7 CapLk Bytes
                           C  ;-----------------------------------------------------------------------
                           C
CC60                       C  kb_cap_flags    label   byte
                           C
CC60  00                   C    db      00000000b       ; scancode 00 (00h) - 07 (07h)  ESC        to '6'
CC61  00                   C    db      00000000b       ; scancode 08 (08h) - 15 (0Fh)  '7'        to HT
CC62  FF                   C    db      11111111b       ; scancode 16 (10h) - 23 (17h)  'q'        to 'i'
CC63  C3                   C    db      11000011b       ; scancode 24 (18h) - 31 (1Fh)  'o' & 'p'  to 'a' & 's'
CC64  FE                   C    db      11111110b       ; scancode 32 (20h) - 39 (27h)  'd'        to 'l' & ';'
CC65  0F                   C    db      00001111b       ; scancode 40 (28h) - 47 (2Fh)  ':' to '\' to 'z' to 'v'
CC66  E0                   C    db      11100000b       ; scancode 48 (30h) - 55 (37h)  'b' to 'm' to ',' to '*'
                           C
                           C  ;-----------------------------------------------------------------------
                           C  ;       Alphabetic (Migratory)                  |Olivetti|Other |
                           C  ;                                               | KB     | KBs  |
                           C  ;-----------------------------------------------------------------------
                           C
CC67                       C  kb_data_table   label   byte
                           C
CC67  1B1B                 C    dw      (1Bh) * 100h + (1Bh)    ;  01 01h       ESC     ESC     (BASE)
CC69  1B1B                 C    dw      (1Bh) * 100h + (1Bh)    ;                  ESC     ESC     (SHIFT)
CC6B  1B1B                 C    dw      (1Bh) * 100h + (1Bh)    ;                  ESC     ESC     (CTL)
CC6D  CDCD                 C    dw      kNONE * 100h + kNONE    ;                  None    None    (ALT)
```

```
CC6F  3131                C   dw    (31h) * 100h + (31h)    ;  02 02h        1       1
CC71  2121                C   dw    (21h) * 100h + (21h)    ;                !       !
CC73  CDCD                C   dw    kNONE * 100h + kNONE    ;                None    None
CC75  7800                C   dw                7800h       ;                X120
CC77  3232                C   dw    (32h) * 100h + (32h)    ;  03 03h        2       2
CC79  2240                C   dw    (22h) * 100h + (40h)    ;                "       @
CC7B  CDCC                C   dw    kNONE * 100h + kbnul    ;                None    NUL=X03(^@)
CC7D  7900                C   dw                7900h       ;                X121
CC7F  3333                C   dw    (33h) * 100h + (33h)    ;  04 04h        3       3
CC81  2323                C   dw    (23h) * 100h + (23h)    ;                #       #
CC83  CDCD                C   dw    kNONE * 100h + kNONE    ;                None    None
CC85  7A00                C   dw                7A00h       ;                X122
CC87  3434                C   dw    (34h) * 100h + (34h)    ;  05 05h        4       4
CC89  2424                C   dw    (24h) * 100h + (24h)    ;                $       $
CC8B  CDCD                C   dw    kNONE * 100h + kNONE    ;                None    None
CC8D  7B00                C   dw                7B00h       ;                X123
CC8F  3535                C   dw    (35h) * 100h + (35h)    ;  06 06h        5       5
CC91  2525                C   dw    (25h) * 100h + (25h)    ;                %       %
CC93  CDCD                C   dw    kNONE * 100h + kNONE    ;                None    None
CC95  7C00                C   dw                7C00h       ;                X124
CC97  3636                C   dw    (36h) * 100h + (36h)    ;  07 07h        6       6
CC99  265E                C   dw    (26h) * 100h + (5Eh)    ;                &       ^
CC9B  CD1E                C   dw    kNONE * 100h + (1Eh)    ;                None    RS (^^)
CC9D  7D00                C   dw                7D00h       ;                X125
CC9F  3737                C   dw    (37h) * 100h + (37h)    ;  08 08h        7       7
CCA1  2726                C   dw    (27h) * 100h + (26h)    ;                '       &
CCA3  CDCD                C   dw    kNONE * 100h + kNONE    ;                None    None
CCA5  7E00                C   dw                7E00h       ;                X126
CCA7  3838                C   dw    (38h) * 100h + (38h)    ;  09 09h        8       8
CCA9  282A                C   dw    (28h) * 100h + (2Ah)    ;                (       *
CCAB  CDCD                C   dw    kNONE * 100h + kNONE    ;                None    None
CCAD  7F00                C   dw                7F00h       ;                X127
CCAF  3939                C   dw    (39h) * 100h + (39h)    ;  10 0Ah        9       9
CCB1  2928                C   dw    (29h) * 100h + (28h)    ;                )       (
CCB3  CDCD                C   dw    kNONE * 100h + kNONE    ;                None    None
CCB5  8000                C   dw                8000h       ;                X128
CCB7  3030                C   dw    (30h) * 100h + (30h)    ;  11 0Bh        0       0
CCB9  5F29                C   dw    (5Fh) * 100h + (29h)    ;                _       )
CCBB  1FCD                C   dw    (1Fh) * 100h + kNONE    ;                US (^_) None
CCBD  8100                C   dw                8100h       ;                X129
CCBF  2D2D                C   dw    (2Dh) * 100h + (2Dh)    ;  12 0Ch        -       -
CCC1  3D5F                C   dw    (3Dh) * 100h + (5Fh)    ;                =       _
CCC3  CD1F                C   dw    kNONE * 100h + (1Fh)    ;                None    US (^_)
CCC5  8200                C   dw                8200h       ;                X130
CCC7  583D                C   dw    (5Eh) * 100h + (3Dh)    ;  13 0Dh        ^       =
CCC9  7E2B                C   dw    (7Eh) * 100h + (2Bh)    ;                ~       +
CCCB  1ECD                C   dw    (1Eh) * 100h + kNONE    ;                RS (^^) None
CCCD  8300                C   dw                8300h       ;                X131
CCCF  0808                C   dw    (08h) * 100h + (08h)    ;  14 0Eh        BS      BS
CCD1  0808                C   dw    (08h) * 100h + (08h)    ;                BS      BS
CCD3  7F7F                C   dw    (7Fh) * 100h + (7Fh)    ;                DEL     DEL
CCD5  CDCD                C   dw    kNONE * 100h + kNONE    ;                None    None
CCD7  0909                C   dw    (09h) * 100h + (09h)    ;  15 0Fh        HT      HT
CCD9  0F00                C   dw                0F00h       ;                RHT=X15
CCDB  CDCD                C   dw    kNONE * 100h + kNONE    ;                None    None
CCDD  CDCD                C   dw    kNONE * 100h + kNONE    ;                None    None
CCDF  7171                C   dw    (71h) * 100h + (71h)    ;  16 10h        q       q
CCE1  5151                C   dw    (51h) * 100h + (51h)    ;                Q       Q
CCE3  1111                C   dw    (11h) * 100h + (11h)    ;                DC1(^Q) DC1(^Q)
CCE5  1000                C   dw                1000h       ;                X16
CCE7  7777                C   dw    (77h) * 100h + (77h)    ;  17 11h        w       w
CCE9  5757                C   dw    (57h) * 100h + (57h)    ;                W       W
CCEB  1717                C   dw    (17h) * 100h + (17h)    ;                ETB(^W) ETB(^W)
CCED  1100                C   dw                1100h       ;                X17
CCEF  6565                C   dw    (65h) * 100h + (65h)    ;  18 12h        e       e
CCF1  4545                C   dw    (45h) * 100h + (45h)    ;                E       E
CCF3  0505                C   dw    (05h) * 100h + (05h)    ;                ENQ(^E) ENQ(^E)
CCF5  1200                C   dw                1200h       ;                X18
CCF7  7272                C   dw    (72h) * 100h + (72h)    ;  19 13h        r       r
CCF9  5252                C   dw    (52h) * 100h + (52h)    ;                R       R
CCFB  1212                C   dw    (12h) * 100h + (12h)    ;                DC2(^R) DC2(^R)
CCFD  1300                C   dw                1300h       ;                X19
CCFF  7474                C   dw    (74h) * 100h + (74h)    ;  20 14h        t       t
CD01  5454                C   dw    (54h) * 100h + (54h)    ;                T       T
CD03  1414                C   dw    (14h) * 100h + (14h)    ;                DC4(^T) DC4(^T)
CD05  1400                C   dw                1400h       ;                X20
CD07  7979                C   dw    (79h) * 100h + (79h)    ;  21 15h        y       y
CD09  5959                C   dw    (59h) * 100h + (59h)    ;                Y       Y
CD0B  1919                C   dw    (19h) * 100h + (19h)    ;                EM (^Y) EM (^Y)
CD0D  1500                C   dw                1500h       ;                X21
CD0F  7575                C   dw    (75h) * 100h + (75h)    ;  22 16h        u       u
CD11  5555                C   dw    (55h) * 100h + (55h)    ;                U       U
CD13  1515                C   dw    (15h) * 100h + (15h)    ;                NAK(^U) NAK(^U)
CD15  1600                C   dw                1600h       ;                X22
CD17  6969                C   dw    (69h) * 100h + (69h)    ;  23 17h        i       i
CD19  4949                C   dw    (49h) * 100h + (49h)    ;                I       I
CD1B  0909                C   dw    (09h) * 100h + (09h)    ;                HT (^I) HT (^I)
CD1D  1700                C   dw                1700h       ;                X23
CD1F  6F6F                C   dw    (6Fh) * 100h + (6Fh)    ;  24 18h        o       o
CD21  4F4F                C   dw    (4Fh) * 100h + (4Fh)    ;                O       O
CD23  0F0F                C   dw    (0Fh) * 100h + (0Fh)    ;                SI (^O) SI (^O)
CD25  1800                C   dw                1800h       ;                X24
CD27  7070                C   dw    (70h) * 100h + (70h)    ;  25 19h        p       p
CD29  5050                C   dw    (50h) * 100h + (50h)    ;                P       P
CD2B  1010                C   dw    (10h) * 100h + (10h)    ;                DLE(^P) DLE(^P)
CD2D  1900                C   dw                1900h       ;                X25
CD2F  405B                C   dw    (40h) * 100h + (5Bh)    ;  26 1Ah        @       [
CD31  607B                C   dw    (60h) * 100h + (7Bh)    ;                `       {
CD33  CC1B                C   dw    kbnul * 100h + (1Bh)    ;          NUL=X03(^@) ESC(^[)
CD35  CDCD                C   dw    kNONE * 100h + kNONE    ;                None    None
```

```
CD37  5B5D          C   dw      (5Bh) * 100h + (5Dh)    ;  27 1Bh       [       ]
CD39  7B7D          C   dw      (7Bh) * 100h + (7Dh)    ;               {       }
CD3B  1B1D          C   dw      (1Bh) * 100h + (1Dh)    ;               ESC(^[) GS (^])
CD3D  CDCD          C   dw      kNONE * 100h + kNONE    ;               None    None
CD3F  0D0D          C   dw      (0Dh) * 100h + (0Dh)    ;  28 1Ch       CR      CR
CD41  0D0D          C   dw      (0Dh) * 100h + (0Dh)    ;               CR      CR
CD43  0A0A          C   dw      (0Ah) * 100h + (0Ah)    ;               LF      LF
CD45  CDCD          C   dw      kNONE * 100h + kNONE    ;               None    None
CD47  C5C5          C   dw      kbctl * 100h + kbctl    ;  29 1Dh       Ctrl    Ctrl
CD49  C5C5          C   dw      kbctl * 100h + kbctl    ;               Ctrl    Ctrl
CD4B  C5C5          C   dw      kbctl * 100h + kbctl    ;               Ctrl    Ctrl
CD4D  C5C5          C   dw      kbctl * 100h + kbctl    ;               Ctrl    Ctrl
CD4F  6161          C   dw      (61h) * 100h + (61h)    ;  30 1Eh       a       a
CD51  4141          C   dw      (41h) * 100h + (41h)    ;               A       A
CD53  0101          C   dw      (01h) * 100h + (01h)    ;               SOH(^A) SOH(^A)
CD55  1E00          C   dw                1E00h         ;               X30
CD57  7373          C   dw      (73h) * 100h + (73h)    ;  31 1Fh       s       s
CD59  5353          C   dw      (53h) * 100h + (53h)    ;               S       S
CD5B  1313          C   dw      (13h) * 100h + (13h)    ;               DC3(^S) DC3(^S)
CD5D  1F00          C   dw                1F00h         ;               X31
CD5F  6464          C   dw      (64h) * 100h + (64h)    ;  32 20h       d       d
CD61  4444          C   dw      (44h) * 100h + (44h)    ;               D       D
CD63  0404          C   dw      (04h) * 100h + (04h)    ;               EOT(^D) EOT(^D)
CD65  2000          C   dw                2000h         ;               X32
CD67  6666          C   dw      (66h) * 100h + (66h)    ;  33 21h       f       f
CD69  4646          C   dw      (46h) * 100h + (46h)    ;               F       F
CD6B  0606          C   dw      (06h) * 100h + (06h)    ;               ACK(^F) ACK(^F)
CD6D  2100          C   dw                2100h         ;               X33
CD6F  6767          C   dw      (67h) * 100h + (67h)    ;  34 22h       g       g
CD71  4747          C   dw      (47h) * 100h + (47h)    ;               G       G
CD73  0707          C   dw      (07h) * 100h + (07h)    ;               BEL(^G) BEL(^G)
CD75  2200          C   dw                2200h         ;               X34
CD77  6868          C   dw      (68h) * 100h + (68h)    ;  35 23h       h       h
CD79  4848          C   dw      (48h) * 100h + (48h)    ;               H       H
CD7B  0808          C   dw      (08h) * 100h + (08h)    ;               BS (^H) BS (^H)
CD7D  2300          C   dw                2300h         ;               X35
CD7F  6A6A          C   dw      (6Ah) * 100h + (6Ah)    ;  36 24h       j       j
CD81  4A4A          C   dw      (4Ah) * 100h + (4Ah)    ;               J       J
CD83  0A0A          C   dw      (0Ah) * 100h + (0Ah)    ;               LF (^J) LF (^J)
CD85  2400          C   dw                2400h         ;               X36
CD87  6B6B          C   dw      (6Bh) * 100h + (6Bh)    ;  37 25h       k       k
CD89  4B4B          C   dw      (4Bh) * 100h + (4Bh)    ;               K       K
CD8B  0B0B          C   dw      (0Bh) * 100h + (0Bh)    ;               VT (^K) VT (^K)
CD8D  2500          C   dw                2500h         ;               X37
CD8F  6C6C          C   dw      (6Ch) * 100h + (6Ch)    ;  38 26h       l       l
CD91  4C4C          C   dw      (4Ch) * 100h + (4Ch)    ;               L       L
CD93  0C0C          C   dw      (0Ch) * 100h + (0Ch)    ;               FF (^L) FF (^L)
CD95  2600          C   dw                2600h         ;               X38
CD97  3B3B          C   dw      (3Bh) * 100h + (3Bh)    ;  39 27h       ;       ;
CD99  2B3A          C   dw      (2Bh) * 100h + (3Ah)    ;               +       :
CD9B  CDCD          C   dw      kNONE * 100h + kNONE    ;               None    None
CD9D  CDCD          C   dw      kNONE * 100h + kNONE    ;               None    None
CD9F  3A27          C   dw      (3Ah) * 100h + (27h)    ;  40 28h       :       '
CDA1  2A22          C   dw      (2Ah) * 100h + (22h)    ;               *       "
CDA3  CDCD          C   dw      kNONE * 100h + kNONE    ;               None    None
CDA5  CDCD          C   dw      kNONE * 100h + kNONE    ;               None    None
CDA7  5D60          C   dw      (5Dh) * 100h + (60h)    ;  41 29h       ]       `
CDA9  7D7E          C   dw      (7Dh) * 100h + (7Eh)    ;               }       ~
CDAB  1DCD          C   dw      (1Dh) * 100h + kNONE    ;               GS (^])  None
CDAD  CDCD          C   dw      kNONE * 100h + kNONE    ;               None    None
CDAF  C6C6          C   dw      kblsh * 100h + kblsh    ;  42 2Ah       LShft   LShft
CDB1  C6C6          C   dw      kblsh * 100h + kblsh    ;               LShft   LShft
CDB3  C6C6          C   dw      kblsh * 100h + kblsh    ;               LShft   LShft
CDB5  C6C6          C   dw      kblsh * 100h + kblsh    ;               LShft   LShft
CDB7  5C5C          C   dw      (5Ch) * 100h + (5Ch)    ;  43 2Bh       \       \
CDB9  7C7C          C   dw      (7Ch) * 100h + (7Ch)    ;               |       |
CDBB  1C1C          C   dw      (1Ch) * 100h + (1Ch)    ;               FS (^\) FS (^\)
CDBD  CDCD          C   dw      kNONE * 100h + kNONE    ;               None    None
CDBF  7A7A          C   dw      (7Ah) * 100h + (7Ah)    ;  44 2Ch       z       z
CDC1  5A5A          C   dw      (5Ah) * 100h + (5Ah)    ;               Z       Z
CDC3  1A1A          C   dw      (1Ah) * 100h + (1Ah)    ;               SUB(^Z) SUB(^Z)
CDC5  2C00          C   dw                2C00h         ;               X44
CDC7  7878          C   dw      (78h) * 100h + (78h)    ;  45 2Dh       x       x
CDC9  5858          C   dw      (58h) * 100h + (58h)    ;               X       X
CDCB  1818          C   dw      (18h) * 100h + (18h)    ;               CAN(^X) CAN(^X)
CDCD  2D00          C   dw                2D00h         ;               X45
CDCF  6363          C   dw      (63h) * 100h + (63h)    ;  46 2Eh       c       c
CDD1  4343          C   dw      (43h) * 100h + (43h)    ;               C       C
CDD3  0303          C   dw      (03h) * 100h + (03h)    ;               ETX(^C) ETX(^C)
CDD5  2E00          C   dw                2E00h         ;               X46
CDD7  7676          C   dw      (76h) * 100h + (76h)    ;  47 2Fh       v       v
CDD9  5656          C   dw      (56h) * 100h + (56h)    ;               V       V
CDDB  1616          C   dw      (16h) * 100h + (16h)    ;               SYN(^V) SYN(^V)
CDDD  2F00          C   dw                2F00h         ;               X47
CDDF  6262          C   dw      (62h) * 100h + (62h)    ;  48 30h       b       b
CDE1  4242          C   dw      (42h) * 100h + (42h)    ;               B       B
CDE3  0202          C   dw      (02h) * 100h + (02h)    ;               STX(^B) STX(^B)
CDE5  3000          C   dw                3000h         ;               X48
CDE7  6E6E          C   dw      (6Eh) * 100h + (6Eh)    ;  49 31h       n       n
CDE9  4E4E          C   dw      (4Eh) * 100h + (4Eh)    ;               N       N
CDEB  0E0E          C   dw      (0Eh) * 100h + (0Eh)    ;               SO (^N) SO (^N)
CDED  3100          C   dw                3100h         ;               X49
CDEF  6D6D          C   dw      (6Dh) * 100h + (6Dh)    ;  50 32h       m       m
CDF1  4D4D          C   dw      (4Dh) * 100h + (4Dh)    ;               M       M
CDF3  0D0D          C   dw      (0Dh) * 100h + (0Dh)    ;               CR (^M) CR (^M)
CDF5  3200          C   dw                3200h         ;               X50
CDF7  2C2C          C   dw      (2Ch) * 100h + (2Ch)    ;  51 33h       ,       ,
CDF9  3C3C          C   dw      (3Ch) * 100h + (3Ch)    ;               <       <
CDFB  CDCD          C   dw      kNONE * 100h + kNONE    ;               None    None
CDFD  CDCD          C   dw      kNONE * 100h + kNONE    ;               None    None
```

```
CDFF  2E2E                C   dw      (2Eh) * 100h + (2Eh)    ;  52 34h        .       .
CE01  3E3E                C   dw      (3Eh) * 100h + (3Eh)    ;                >       >
CE03  CDCD                C   dw      kNONE * 100h + kNONE    ;                None    None
CE05  CDCD                C   dw      kNONE * 100h + kNONE    ;                None    None
CE07  2F2F                C   dw      (2Fh) * 100h + (2Fh)    ;  53 35h        /       /
CE09  3F3F                C   dw      (3Fh) * 100h + (3Fh)    ;                ?       ?
CE0B  CDCD                C   dw      kNONE * 100h + kNONE    ;                None    None
CE0D  CDCD                C   dw      kNONE * 100h + kNONE    ;                None    None
                          C
                          C ;-----------------------------------------------------------------------
                          C ;       Alphabetic (Non-Migratory)                |Olivetti|Other |
                          C ;                                                 |  KB    | KBs  |
                          C ;-----------------------------------------------------------------------
                          C
CE0F  C7C7                C   dw      kbrsh * 100h + kbrsh    ;  54 36h        RShft   RShft
CE11  C7C7                C   dw      kbrsh * 100h + kbrsh    ;                RShft   RShft
CE13  C7C7                C   dw      kbrsh * 100h + kbrsh    ;                RShft   RShft
CE15  C7C7                C   dw      kbrsh * 100h + kbrsh    ;                RShft   RShft
CE17  2A2A                C   dw      (2Ah) * 100h + (2Ah)    ;  55 37h        *       *
CE19  CBCB                C   dw      kbprt * 100h + kbprt    ;                PrtSc   PrtSc
CE1B  7200                C   dw              7200h           ;                X114
CE1D  CDCD                C   dw      kNONE * 100h + kNONE    ;                None    None
CE1F  C4C4                C   dw      kbalt * 100h + kbalt    ;  56 38h        ALT     ALT
CE21  C4C4                C   dw      kbalt * 100h + kbalt    ;                ALT     ALT
CE23  C4C4                C   dw      kbalt * 100h + kbalt    ;                ALT     ALT
CE25  C4C4                C   dw      kbalt * 100h + kbalt    ;                ALT     ALT
CE27  2020                C   dw      (20h) * 100h + (20h)    ;  57 39h        SP      SP
CE29  2020                C   dw      (20h) * 100h + (20h)    ;                SP      SP
CE2B  2020                C   dw      (20h) * 100h + (20h)    ;                SP      SP
CE2D  2020                C   dw      (20h) * 100h + (20h)    ;                SP      SP
CE2F  C1C1                C   dw      kbcap * 100h + kbcap    ;  58 3Ah        CapLk   CapLk
CE31  C1C1                C   dw      kbcap * 100h + kbcap    ;                CapLk   CapLk
CE33  C1C1                C   dw      kbcap * 100h + kbcap    ;                CapLk   CapLk
CE35  C1C1                C   dw      kbcap * 100h + kbcap    ;                CapLk   CapLk
CE37  3B00                C   dw              3B00h           ;  59 3Bh        F01=X59
CE39  5400                C   dw              5400h           ;                F11=X84
CE3B  5E00                C   dw              5E00h           ;                F21=X94
CE3D  6800                C   dw              6800h           ;                F31=X104
CE3F  3C00                C   dw              3C00h           ;  60 3Ch        F02=X60
CE41  5500                C   dw              5500h           ;                F12=X85
CE43  5F00                C   dw              5F00h           ;                F22=X95
CE45  6900                C   dw              6900h           ;                F32=X105
CE47  3D00                C   dw              3D00h           ;  61 3Dh        F03=X61
CE49  5600                C   dw              5600h           ;                F13=X86
CE4B  6000                C   dw              6000h           ;                F23=X96
CE4D  6A00                C   dw              6A00h           ;                F33=X106
CE4F  3E00                C   dw              3E00h           ;  62 3Eh        F04=X62
CE51  5700                C   dw              5700h           ;                F14=X87
CE53  6100                C   dw              6100h           ;                F24=X97
CE55  6B00                C   dw              6B00h           ;                F34=X107
CE57  3F00                C   dw              3F00h           ;  63 3Fh        F05=X63
CE59  5800                C   dw              5800h           ;                F15=X88
CE5B  6200                C   dw              6200h           ;                F25=X98
CE5D  6C00                C   dw              6C00h           ;                F35=X108
CE5F  4000                C   dw              4000h           ;  64 40h        F06=X64
CE61  5900                C   dw              5900h           ;                F16=X89
CE63  6300                C   dw              6300h           ;                F26=X99
CE65  6D00                C   dw              6D00h           ;                F36=X109
CE67  4100                C   dw              4100h           ;  65 41h        F07=X65
CE69  5A00                C   dw              5A00h           ;                F17=X90
CE6B  6400                C   dw              6400h           ;                F27=X100
CE6D  6E00                C   dw              6E00h           ;                F37=X110
CE6F  4200                C   dw              4200h           ;  66 42h        F08=X66
CE71  5B00                C   dw              5B00h           ;                F18=X91
CE73  6500                C   dw              6500h           ;                F28=X101
CE75  6F00                C   dw              6F00h           ;                F38=X111
CE77  4300                C   dw              4300h           ;  67 43h        F09=X67
CE79  5C00                C   dw              5C00h           ;                F19=X92
CE7B  6600                C   dw              6600h           ;                F29=X102
CE7D  7000                C   dw              7000h           ;                F39=X112
CE7F  4400                C   dw              4400h           ;  68 44h        F10=X68
CE81  5D00                C   dw              5D00h           ;                F20=X93
CE83  6700                C   dw              6700h           ;                F30=X103
CE85  7100                C   dw              7100h           ;                F40=X113
CE87  C2C2                C   dw      kbnum * 100h + kbnum    ;  69 45h        NumLk   NumLk
CE89  C2C2                C   dw      kbnum * 100h + kbnum    ;                NumLk   NumLk
CE8B  CACA                C   dw      pause * 100h + pause    ;                Pause   Pause
CE8D  C2C2                C   dw      kbnum * 100h + kbnum    ;                NumLk   NumLk
CE8F  C3C3                C   dw      kbscr * 100h + kbscr    ;  70 46h        ScrLk   ScrLk
CE91  C3C3                C   dw      kbscr * 100h + kbscr    ;                ScrLk   ScrLk
CE93  C9C9                C   dw      kbbrk * 100h + kbbrk    ;                Break   Break
CE95  C3C3                C   dw      kbscr * 100h + kbscr    ;                ScrLk   ScrLk
                          C
                          C ;-----------------------------------------------------------------------
                          C ;       Numeric Keypad                            |Olivetti|Other |
                          C ;                                                 |  KB    | KBs  |
                          C ;-----------------------------------------------------------------------
                          C
CE97  4700                C   dw              4700h           ;  71 47h        Home=X71
CE99  3737                C   dw      (37h) * 100h + (37h)    ;                7       7
CE9B  7700                C   dw              7700h           ;                X119
CE9D  D0D0                C   dw      kdec7 * 100h + kdec7    ;                XDec7   XDec7
CE9F  4800                C   dw              4800h           ;  72 48h        Up      =X72
CEA1  3838                C   dw      (38h) * 100h + (38h)    ;                8       8
CEA3  CDCD                C   dw      kNONE * 100h + kNONE    ;                None    None
CEA5  CFCF                C   dw      kdec8 * 100h + kdec8    ;                XDec8   XDec8
CEA7  4900                C   dw              4900h           ;  73 49h        PgUp=X73
CEA9  3939                C   dw      (39h) * 100h + (39h)    ;                9       9
CEAB  8400                C   dw              8400h           ;                X132
CEAD  CECE                C   dw      kdec9 * 100h + kdec9    ;                XDec9   XDec9
```

```
CEAF  2D2D              C   dw      (2Dh) * 100h + (2Dh)    ;  74 4Ah       -       -
CEB1  2D2D              C   dw      (2Dh) * 100h + (2Dh)    ;               -       -
CEB3  CDCD              C   dw      kNONE * 100h + kNONE    ;               None    None
CEB5  CDCD              C   dw      kNONE * 100h + kNONE    ;               None    None
CEB7  4B00              C   dw                4B00h         ;  75 4Bh       Left=X75
CEB9  3434              C   dw      (34h) * 100h + (34h)    ;               4       4
CEBB  7300              C   dw                7300h         ;               X115
CEBD  D3D3              C   dw      kdec4 * 100h + kdec4    ;  76 4Ch       XDec4   XDec4
CEBF  CDCD              C   dw      kNONE * 100h + kNONE    ;               None    None
CEC1  3535              C   dw      (35h) * 100h + (35h)    ;               5       5
CEC3  CDCD              C   dw      kNONE * 100h + kNONE    ;               None    None
CEC5  D2D2              C   dw      kdec5 * 100h + kdec5    ;               XDec5   XDec5
CEC7  4D00              C   dw                4D00h         ;  77 4Dh       Rght=X77
CEC9  3636              C   dw      (36h) * 100h + (36h)    ;               6       6
CECB  7400              C   dw                7400h         ;               X116
CECD  D1D1              C   dw      kdec6 * 100h + kdec6    ;               XDec6   XDec6
CECF  2B2B              C   dw      (2Bh) * 100h + (2Bh)    ;  78 4Eh       +       +
CED1  2B2B              C   dw      (2Bh) * 100h + (2Bh)    ;               +       +
CED3  CDCD              C   dw      kNONE * 100h + kNONE    ;               None    None
CED5  CDCD              C   dw      kNONE * 100h + kNONE    ;               None    None
CED7  4F00              C   dw                4F00h         ;  79 4Fh       End =X79
CED9  3131              C   dw      (31h) * 100h + (31h)    ;               1       1
CEDB  7500              C   dw                7500h         ;               X117
CEDD  D6D6              C   dw      kdec1 * 100h + kdec1    ;               XDec1   XDec1
CEDF  5000              C   dw                5000h         ;  80 50h       Down=X80
CEE1  3232              C   dw      (32h) * 100h + (32h)    ;               2       2
CEE3  CDCD              C   dw      kNONE * 100h + kNONE    ;               None    None
CEE5  D5D5              C   dw      kdec2 * 100h + kdec2    ;               XDec2   XDec2
CEE7  5100              C   dw                5100h         ;  81 51h       PgDn=X81
CEE9  3333              C   dw      (33h) * 100h + (33h)    ;               3       3
CEEB  7600              C   dw                7600h         ;               X118
CEED  D4D4              C   dw      kdec3 * 100h + kdec3    ;               XDec3   XDec3
CEEF  C0C0              C   dw      kbins * 100h + kbins    ;  82 52h       INS=X82 INS=X82
CEF1  3030              C   dw      (30h) * 100h + (30h)    ;               0       0
CEF3  CDCD              C   dw      kNONE * 100h + kNONE    ;               None    None
CEF5  D7D7              C   dw      kdec0 * 100h + kdec0    ;               XDec0   XDec0
CEF7  5300              C   dw                5300h         ;  83 53h       DEL =X83
CEF9  2E2E              C   dw      (2Eh) * 100h + (2Eh)    ;               .       .
CEFB  C8C8              C   dw      kbres * 100h + kbres    ;               Reset   Reset
CEFD  C8C8              C   dw      kbres * 100h + kbres    ;               Reset   Reset
                        C
                        C   ;-----------------------------------------------------------------------
                        C   ;       Function Keypad                         |Olivetti|Other|
                        C   ;                                               | KB     | KBs |
                        C   ;-----------------------------------------------------------------------
                        C
CEFF  D8D8              C   dw      kdbl0 * 100h + kdbl0    ;  84 54h       00      00
CF01  D8D8              C   dw      kdbl0 * 100h + kdbl0    ;               00      00
CF03  CDCD              C   dw      kNONE * 100h + kNONE    ;               None    None
CF05  CDCD              C   dw      kNONE * 100h + kNONE    ;               None    None
CF07  CBCB              C   dw      kbprt * 100h + kbprt    ;  85 55h       PrtSc   PrtSc
CF09  CBCB              C   dw      kbprt * 100h + kbprt    ;               PrtSc   PrtSc
CF0B  7200              C   dw                7200h         ;               X114
CF0D  CDCD              C   dw      kNONE * 100h + kNONE    ;               None    None
CF0F  CACA              C   dw      pause * 100h + pause    ;  86 56h       Pause   Pause
CF11  CACA              C   dw      pause * 100h + pause    ;               Pause   Pause
CF13  CACA              C   dw      pause * 100h + pause    ;               Pause   Pause
CF15  CACA              C   dw      pause * 100h + pause    ;               Pause   Pause
CF17  0D0D              C   dw      (0Dh) * 100h + (0Dh)    ;  87 57h       CR      CR
CF19  0D0D              C   dw      (0Dh) * 100h + (0Dh)    ;               CR      CR
CF1B  0A0A              C   dw      (0Ah) * 100h + (0Ah)    ;               LF      LF
CF1D  CDCD              C   dw      kNONE * 100h + kNONE    ;               None    None
CF1F  4B00              C   dw                4B00h         ;  88 58h       Left=X75
CF21  7300              C   dw                7300h         ;               Rev Word = X115
CF23  7300              C   dw                7300h         ;               Rev Word = X115
CF25  7300              C   dw                7300h         ;               Rev Word = X115
CF27  5000              C   dw                5000h         ;  89 59h       Down=X80
CF29  5000              C   dw                5000h         ;               Down=X80
CF2B  5000              C   dw                5000h         ;               Down=X80
CF2D  5000              C   dw                5000h         ;               Down=X80
CF2F  4D00              C   dw                4D00h         ;  90 5Ah       Rght=X77
CF31  7400              C   dw                7400h         ;               Adv Word = X116
CF33  7400              C   dw                7400h         ;               Adv Word = X116
CF35  7400              C   dw                7400h         ;               Adv Word = X116
CF37  4800              C   dw                4800h         ;  91 5Bh       Up  =X72
CF39  4700              C   dw                4700h         ;               Home=X71
CF3B  4700              C   dw                4700h         ;               Home=X71
CF3D  4700              C   dw                4700h         ;               Home=X71
CF3F  C9C9              C   dw      kbbrk * 100h + kbbrk    ;  92 5Ch       Break   Break
CF41  C9C9              C   dw      kbbrk * 100h + kbbrk    ;               Break   Break
CF43  C9C9              C   dw      kbbrk * 100h + kbbrk    ;               Break   Break
CF45  C9C9              C   dw      kbbrk * 100h + kbbrk    ;               Break   Break
CF47  C9C9              C   dw      kbbrk * 100h + kbbrk    ;  93 5Dh       Break   Break
CF49  C9C9              C   dw      kbbrk * 100h + kbbrk    ;               Break   Break
CF4B  C9C9              C   dw      kbbrk * 100h + kbbrk    ;               Break   Break
CF4D  C9C9              C   dw      kbbrk * 100h + kbbrk    ;               Break   Break
CF4F  C2C2              C   dw      kbnum * 100h + kbnum    ;  94 5Eh       NumLk   NumLk
CF51  C2C2              C   dw      kbnum * 100h + kbnum    ;               NumLk   NumLk
CF53  CACA              C   dw      pause * 100h + pause    ;               Pause   Pause
CF55  C2C2              C   dw      kbnum * 100h + kbnum    ;               NumLk   NumLk
CF57  2F2F              C   dw      (2Fh) * 100h + (2Fh)    ;  95 5Fh       /       /
CF59  2F2F              C   dw      (2Fh) * 100h + (2Fh)    ;               /       /
CF5B  CDCD              C   dw      kNONE * 100h + kNONE    ;               None    None
CF5D  CDCD              C   dw      kNONE * 100h + kNONE    ;               None    None
CF5F  5400              C   dw                5400h         ;  96 60h       F11=X84
CF61  5400              C   dw                5400h         ;               F11=X84
CF63  5400              C   dw                5400h         ;               F11=X84
CF65  5400              C   dw                5400h         ;               F11=X84
CF67  5500              C   dw                5500h         ;  97 61h       F12=X85
CF69  5500              C   dw                5500h         ;               F12=X85
```

```
CF6B  5500                C   dw              5500h           ;               F12=X85
CF6D  5500                C   dw              5500h           ;               F12=X85
CF6F  5600                C   dw              5600h           ;  98 62h       F13=X86
CF71  5600                C   dw              5600h           ;               F13=X86
CF73  5600                C   dw              5600h           ;               F13=X86
CF75  5600                C   dw              5600h           ;               F13=X86
CF77  5700                C   dw              5700h           ;  99 63h       F14=X87
CF79  5700                C   dw              5700h           ;               F14=X87
CF7B  5700                C   dw              5700h           ;               F14=X87
CF7D  5700                C   dw              5700h           ;               F14=X87
CF7F  5800                C   dw              5800h           ; 100 64h       F15=X88
CF81  5800                C   dw              5800h           ;               F15=X88
CF83  5800                C   dw              5800h           ;               F15=X88
CF85  5800                C   dw              5800h           ;               F15=X88
CF87  5900                C   dw              5900h           ; 101 65h       F16=X89
CF89  5900                C   dw              5900h           ;               F16=X89
CF8B  5900                C   dw              5900h           ;               F16=X89
CF8D  5900                C   dw              5900h           ;               F16=X89
CF8F  5A00                C   dw              5A00h           ; 102 66h       F17=X90
CF91  5A00                C   dw              5A00h           ;               F17=X90
CF93  5A00                C   dw              5A00h           ;               F17=X90
CF95  5A00                C   dw              5A00h           ;               F17=X90
CF97  5B00                C   dw              5B00h           ; 103 67h       F18=X91
CF99  5B00                C   dw              5B00h           ;               F18=X91
CF9B  5B00                C   dw              5B00h           ;               F18=X91
CF9D  5B00                C   dw              5B00h           ;               F18=X91
                          C
CF9F                      C   kb_data1        endp
                          C
CF9F                      C   code    ends
                          C   include hdu.asm
                          C
                          C
                          C   ;=============================================================================
                          C   ;       Filename:       hdu.src
                          C   ;
                          C   ;       This module includes HDU support.
                          C   ;
                          C   ;=============================================================================
                          C
CF9F                      C   code    segment public 'ROM'
                          C           assume  cs:code, ds:nothing, es:nothing, ss:nothing
                          C
                          C   ; ######## COMMANDS AVAILABLE FROM INTERRUPT 13 ########
                          C
= 0000                    C   reset_int               equ     00h     ;reset disks (according to drive#) and
                          C                                           ;   assign_param
= 0001                    C   status_int              equ     01h     ;return status of last operation
                          C                                           ;   (disk_status) in al
= 0002                    C   read_int                equ     02h     ;read sectors from disk
                          C   ;write_int              equ     03h     ;write sectors to disk
                          C   ;chktrk_int             equ     04h     ;check track
                          C   ;fmttrk_int             equ     05h     ;format track
                          C   ;fmtbad_int             equ     06h     ;format bad track
= 0007                    C   fmtdrv_int              equ     07h     ;format drive starting at the
                          C                                           ;   indicated track
                          C
                          C   ;read_param_int         equ     08h     ;read the current drive parameters
= 0009                    C   assign_param_int        equ     09h     ;assign parameters to drives
                          C   ;rlong_int              equ     0ah     ;read long (512 data + 4 ecc)
                          C   ;wlong_int              equ     0bh     ;write long
                          C   ;seek_int               equ     0ch     ;seek cylinder
                          C   ;reset2_int             equ     0dh     ;same as 00h
                          C   ;rbuff_int              equ     0eh     ;read sector buffer
= 000F                    C   wbuff_int               equ     0fh     ;write sector buffer
                          C
= 0010                    C   tst_drv_rdy_int         equ     10h     ;test drive ready
= 0011                    C   recal_int               equ     11h     ;recalibrate drive (seek track 00)
= 0012                    C   ramdiag_int             equ     12h     ;controller ram diagnostic
                          C   ;drvdiag_int            equ     13h     ;drive diagnostic
= 0014                    C   bxdiag_int              equ     14h     ;controller internal diagnostic
                          C
= 0014                    C   max_int                 equ     14h     ;maximum interrupt command number
                          C
                          C   ; ######## INT13 STATUS CODES ########
                          C
                          C   ; These are put into the disk_status byte in data_seg when there is an error.
                          C
                          C   ;reset_err              equ     05h     ;disk reset (int13-00) failed
= 0007                    C   init_err                equ     07h     ;assign parameter (int13-09) failed
= 000B                    C   badtrack_err            equ     0bh     ;bad track
= 0011                    C   ecc_used_err            equ     11h     ;ecc used to correct data
= 00FF                    C   sense_err               equ     0ffh    ;request sense failed
                          C
                          C   ; ######## CONTROLLER INFORMATION ########
                          C
                          C   ; drives_per_bx         equ      2      ;max number of drives on a controller
                          C   ; drives_per_pc         equ      8      ;max number of drives on a pc
                          C   ; bxes_per_pc           equ      4      ;max number of bxes on a pc
                          C   ; ports_per_bx          equ      4      ;number of i/o ports for a controller
= 0011                    C   sectors_per_track       equ     17      ;compatible fixed format
                          C
                          C   ; ######## CONTROLLER PORTS ########
                          C
= 0320                    C   read_port       equ     320h            ;read data port
= 0320                    C   write_port      equ     320h            ;write data port
= 0321                    C   status_port     equ     321h            ;read controller hardware status
= 0321                    C   reset_port      equ     321h            ;write to reset controller hardware
= 0322                    C   switch_port     equ     322h            ;read the drivetype switches
= 0322                    C   select_port     equ     322h            ;write to select controller
```

```
= 0323                  C  mask_port       equ     323h            ;write to mask p_dma and interrupts
                        C
                        C  ; ######## CONTROLLER REGISTER BITS ########
                        C
                        C  ; This is the completion status byte received in the read_port after
                        C  ; command completion.
                        C
                        C  ;completion_status:
                        C  ;                h_c_drive       equ     0E0h    ; bits #7 to 5
= 0002                  C                   h_c_err         equ     002h    ; bit  #1
                        C
                        C  ; This is the controller hardware status read from status_port.
                        C
                        C  ;hdu_status:
= 0020                  C                   hdu_irq5        equ     020h    ; bit  #5
                        C  ;                hdu_dreq3       equ     010h    ; bit  #4
= 0008                  C                   hdu_bsy         equ     008h    ; bit  #3
= 0004                  C                   hdu_cd          equ     004h    ; bit  #2
= 0002                  C                   hdu_io          equ     002h    ; bit  #1
= 0001                  C                   hdu_req         equ     001h    ; bit  #0
                        C
                        C  ; This is the controller p_dma and interrupt mask written to mask_port.
                        C
                        C  ;mask_port_cmd: mask_port_bank  equ     008h    ; bit  #3
= 0002                  C                   mask_port_inte  equ     002h    ; bit  #1
= 0001                  C                   mask_port_dmae  equ     001h    ; bit  #0
                        C
                        C  ; These are the bits for a drive in the switch.
                        C
                        C  ; switch_mask            equ     00110011b
                        C
                        C  ; ######## CONTROLLER COMMANDS ########
                        C
                        C  ; The first byte of the cmd_block (CDB) is the controller command.
                        C  ; The upper 3 bits are the class (either 0 or 7, all reset or all set).
                        C  ; The lower 5 bits are the opcode (0 to 31).
                        C
                        C  ; odb__0         record  class:3, opcode:5
                        C
                        C  ; Class 0 Controller Commands:  000?????b.
                        C
= 0000                  C  tst_drv_rdy_bx  equ      00h
= 0001                  C  recal_bx        equ      01h
                        C  ; reserved               02h
= 0003                  C  sense_bx        equ      03h
= 0004                  C  fmtdrv_bx       equ      04h
= 0005                  C  chktrk_bx       equ      05h
= 0006                  C  fmttrk_bx       equ      06h
= 0007                  C  fmtbad_bx       equ      07h
= 0008                  C  read_bx         equ      08h
                        C  ; reserved               09h
= 000A                  C  write_bx        equ      0Ah
= 000B                  C  seek_bx         equ      0Bh
= 000C                  C  assign_param_bx equ      0Ch
= 000D                  C  read_ecc_bx     equ      0Dh
= 000E                  C  rbuff_bx        equ      0Eh
= 000F                  C  wbuff_bx        equ      0Fh
                        C
                        C  ; Class 7 Controller Commands:  111?????b.
                        C
= 00E0                  C  ramdiag_bx      equ      0E0h
                        C  ; reserved               0E1h
                        C  ; reserved               0E2h
= 00E3                  C  drvdiag_bx      equ      0E3h
= 00E4                  C  bxdiag_bx       equ      0E4h
= 00E5                  C  rlong_bx        equ      0E5h
= 00E6                  C  wlong_bx        equ      0E6h
                        C
                        C  ; ########  CONTROLLER CDB BYTES ########
                        C
                        C  ; The remainder of the cmd_block bytes (CDB) are defined as follows:
                        C
                        C  odb__1   record  zzp:2, drivenum:1, headnum:5
                        C
                        C  ; odb__2         record  cylhi:2, secnum:6
                        C
                        C  ; odb__3         record  cyllow:8
                        C
                        C  ; odb__4         record  zzf:3, interleave:5
                        C
                        C  ; odb__5         record  retry:1, use_ecc:1, zzg:3, stepcode:3
                        C
                        C  ; ######## SENSE BYTES FOR REQUEST SENSE COMMAND ########
                        C
                        C  ; Bytes returned when a request sense command is issued in response to an error.
                        C
                        C  ; sense0:        sense0_val_addr equ     080h    ; bit  #7
                        C  ;                sense0_type     equ     030h    ; bits #5 & 4
                        C  ;                sense0_code     equ     00Fh    ; bits #3 to 0
= 003F                  C                   sense0_msk      equ     03Fh    ; sense0_type+sense0_code
                        C
                        C  ; sense1:        sense1_drive    equ     020h    ; bit  #5
                        C  ;                sense1_head     equ     01Fh    ; bits #4 to 0
                        C
                        C  ; sense2:        sense2_cylhi    equ     0C0h    ; bits #7 & 6
                        C  ;                sense2_sector   equ     03Fh    ; bit  #5 to 0
                        C
                        C  ; sense3:        sense2_cyllow   equ     0FFh    ; bits #7 to 0
                        C
                        C  ; ######## HARD DISK PARAMETER TABLE ########
```

```
                                C
                                C  ; This is a table which is indexed by the switch settings to determine
                                C  ; the parameters for each drive.
                                C
                                C  parameter_table struc
                                C
0000  0132                      C  p_cyls           dw      306d    ;number of cylinders
0002  04                        C  p_heads          db        4d    ;number of heads
0003  0132                      C  p_write_cur      dw      306d    ;reduced write current
0005  0000                      C  p_precomp        dw        0d    ;write precompensation
0007  08                        C  p_ecc_len        db       11d    ;maximum ecc burst length
0008  05                        C  p_control_byte   db        5d    ;enable retry, enable ecc, 70usec steps
0009  0C                        C  p_timeout        db       0ch    ;standard timeout
000A  B4                        C  p_fmt_timeout    db      0b4h    ;timeout for format drive
000B  28                        C  p_drvdiag_timeout db     028h    ;timeout for test drive ready
000C  00 00 00 00               C  p_zzj            db      0,0,0,0
                                C
0010                            C  parameter_table ends
                                C
                                C  ; ####### p_dma SYSTEM (8237) #######
                                C
= 0047                          C  read_mode        equ     01000111b               ;write to dma_mode
                                C                           ;7:6    single mode
                                C                           ;5:5    address increment
                                C                           ;4:4    autoinit disable
                                C                           ;3:2    read
                                C                           ;1:0    channel 3
                                C
= 004B                          C  write_mode       equ     01001011b               ;write to dma_mode
                                C                           ;7:6    single mode
                                C                           ;5:5    address increment
                                C                           ;4:4    autoinit disable
                                C                           ;3:2    write
                                C                           ;1:0    channel 3
                                C
                                C
= 0003                          C  allow_dma3       equ     00000011b               ;write to dma_mask_bit to allow
= 0007                          C  disallow_dma3    equ     00000111b               ;or disallow dma3 interrupt
                                C
= 0082                          C  hdu_dma_segm     equ     082h    ;port sets top 4 bits of 20-bit p_dma address
                                C
                                C
                                C  ; ####### MACROS #######
                                C
                                C  ; Input to a register from a port.
                                C  ; Won't work when the index has to cause a carry to high byte.
                                C
                                C  inport  macro   inrg,inpt
                                C          mov     dx,inpt         ;get the port number for the base controller
                                C          add     dl,byte ptr ds:[port_off]       ;add port_off for the correct bx
                                C          in      inrg,dx
                                C          endm
                                C
                                C  ; Output from a register to a port.
                                C  ; Ditto.
                                C
                                C  outport macro   outpt,outrg
                                C          mov     dx,outpt        ;get the port number for the base controller
                                C          add     dl,byte ptr ds:[port_off]       ;add port_off for the correct bx
                                C          out     dx,outrg
                                C          endm
                                C
                                C  ;======= hd2.asm       ================================================
                                C  ;       Power on self test
                                C  ;=============================================================================
                                C
                                C          assume  cs:code, ds:nothing, es:nothing, ss:nothing
                                C
CF9F                            C  h_init  proc    near
                                C
                                C  ; Install vectors.
                                C
                                C          assume  cs:code, ds:abs0, es:abs0, ss:nothing
                                C
CF9F  33 C0                     C          xor     ax,ax                           ; satisfy assumptions
CFA1  8E D8                     C          mov     ds,ax                           ; ds = es = ax = abs0_seg
CFA3  8E C0                     C          mov     es,ax
                                C
CFA5  FA                        C          cli                                     ; disable during critical code.
                                C
                                C  ; Install HDU hardware interrupt service routine.
                                C
CFA6  C7 06 0034 R D62F R       C          mov     word ptr ds:[int0Dloc+0000h],cs:(offset h_int)
CFAC  8C 0E 0036 R              C          mov     word ptr ds:[int0Dloc+0002h],cs
                                C
                                C  ; Transfer old FDU int 13 vector to the new int 40 FDU location.
                                C
CFB0  A1 004C R                 C          mov     ax,word ptr ds:[int13loc+0000h]
CFB3  A3 0100                   C          mov     word ptr ds:[(4*40h)+0000h],ax
CFB6  A1 004E R                 C          mov     ax,word ptr ds:[int13loc+0002h]
CFB9  A3 0102                   C          mov     word ptr ds:[(4*40h)+0002h],ax
                                C
                                C  ; Install new HDU request int 13 vector.
                                C
CFBC  C7 06 004C R D1AB R       C          mov     word ptr ds:[int13loc+0000h],cs:(offset h_io)
CFC2  8C 0E 004E R              C          mov     word ptr ds:[int13loc+0002h],cs
                                C
                                C  ; Install new HDU boot-strap int 19 vector
                                C
CFC6  C7 06 0064 R D0E9 R       C          mov     word ptr ds:[int19loc+0000h],cs:(offset h_boot)
```

```
CFCC  8C 0E 0066 R          C          mov     word ptr ds:[int19locn+0002h],cs
                            C
                            C  ; Install new int 41 HDU parameter table pointer
                            C
CFD0  C7 06 0104 D14B R     C          mov     word ptr ds:[(4*41h)+0000h],cs:(offset hdu_parm_tbl)
CFD6  8C 0E 0106            C          mov     word ptr ds:[(4*41h)+0002h],cs
                            C
CFDA  FB                    C          sti                                         ; exit critical code.
                            C
                            C
                            C  ; Initialize stuff in low memory.
                            C
                            C          assume  cs:code, ds:data, es:abs0, ss:nothing
                            C
CFDB  B8 0040               C          mov     ax,data_seg                         ; satisfy assumptions.
CFDE  8E D8                 C          mov     ds,ax
                            C
CFE0  C6 06 0074 R 00       C          mov     byte ptr ds:[disk_status],0         ; clear errors.
CFE5  C6 06 0075 R 01       C          mov     byte ptr ds:[hf_num],1              ; assume 1 hard disk.
                            C
                            C  ; Reset the controller.
                            C
CFEA  B9 000A               C          mov     cx,10                   ;try to reset controller up to 10 times
                            C
                            C  init_controller_lp:
                            C
CFED  8A 16 0075 R          C          mov     dl,byte ptr ds:[hf_num]         ;get the current hd number
CFF1  D0 E2                 C          shl     dl,1                            ;make it look like
CFF3  80 E2 FC              C          and     dl,11111100b                    ;an index
CFF6  88 16 0077 R          C          mov     byte ptr ds:[port_off],dl       ;fake for reset_hx
CFFA  E8 D5B1 R             C          call    reset_hx                        ;reset that controller
                            C
                            C  ; Set up dl to call int13.
                            C
CFFD  B6 00                 C          mov     dh,0                    ;clear dh
CFFF  8A 16 0075 R          C          mov     dl,byte ptr ds:[hf_num] ;number of HDUs (incremented in this
                            C                                          ;   loop)
D003  FE CA                 C          dec     dl                      ;compensate for zero-origin
D005  80 CA 80              C          or      dl,80h                  ;flag as a hard disk number
                            C
                            C  ; Test the controller.
                            C
D008  B4 12                 C          mov     ah,ramdiag_int          ;set up for int13 - controller ram test
D00A  CD 13                 C          int     13h
D00C  72 5C                 C          jc      post_no_more_drives     ;if error returned
                            C
D00E  B4 14                 C          mov     ah,bxdiag_int           ;set up for int13 - controller internal test
                            C                                          ;   test
D010  CD 13                 C          int     13h
D012  72 56                 C          jc      post_no_more_drives     ;if error returned
                            C
                            C
                            C  ; Set up p_timer for timeout of tst_rdy.
                            C
D014  C7 06 006C R 0000     C          mov     word ptr ds:[t_low_order],0*18  ; start counting at zero seconds
                            C                                                  ; t_low_order counts
                            C                                                  ;   18times/sec.
                            C  ; Test system reset_flag.
                            C
D01A  81 3E 0072 R 1234     C          cmp     word ptr ds:[reset_flag],01234h ; test flag for reset function.
D020  75 06                 C          jne     init_start_timer                ; skip if not CTL ALT DEL reboot
                            C
D022                        C  init_drive_lp:
                            C
D022  C7 06 006C R 0132     C          mov     word ptr ds:[t_low_order],17*18 ; only give 3 seconds if
                            C                                                  ; warm boot or not first disk
                            C                                                  ; t_low_order counts
                            C                                                  ;   18times/sec.
D028                        C  init_start_timer:
                            C
D028  FA                    C          cli
D029  E4 21                 C          in      al,pic_1                        ; get interrupt mask
D02B  24 FE                 C          and     al,0FEh                         ; un-mask p_timer interrupt IR0
D02D  E6 21                 C          out     pic_1,al                        ; set the controller
D02F  FB                    C          sti
                            C
D030  B4 09                 C          mov     ah,assign_param_int     ;assign drive parameters according to
D032  CD 13                 C          int     13h                     ;switch settings on bx board
D034  72 3B                 C          jc      post_lose
                            C
D036                        C  init_try_drive:
D036  B4 10                 C          mov     ah,tst_drv_rdy_int      ;test drive ready
D038  CD 13                 C          int     13h
D03A  73 0B                 C          jno     init_drive_win
D03C  81 3E 006C R 0168     C          cmp     word ptr ds:[t_low_order],20*18 ; t_low_order counts
                            C                                                  ;   18times/sec.
D042  72 F2                 C          jb      init_try_drive          ;if not given enough time yet, go back
D044  EB 24 90              C          jmp     post_no_more_drives     ;if given enough time and still losing
                            C
D047                        C  init_drive_win:
D047  B4 11                 C          mov     ah,recal_int            ;recalibrate drive
D049  CD 13                 C          int     13h
D04B  72 24                 C          jc      post_lose
                            C
                            C          assume  es:code
                            C
D04D  8C C8                 C          mov     ax,cs                   ;satisfy assumption
D04F  8E C0                 C          mov     es,ax
D051  33 DB                 C          xor     bx,bx                   ;initialize offset (es:bx)
                            C
```

```
0053  B4 0F                    C          mov     ah,wbuff_int            ;write sector buffer as a test
0055  B0 01                    C          mov     al,01                   ;one block length
0057  CD 13                    C          int     13h
0059  72 16                    C          jc      post_lose
                               C
                               C
                               C  ; Looks like we got another hard disk!
                               C
005B  FE 06 0075 R             C          inc     byte ptr ds:[hf_num]    ;point to the next one
005F  FE C2                    C          inc     dl                      ;register for calling int13
0061  F6 06 0075 R 01          C          test    byte ptr ds:[hf_num],01h ;have we proceeded to a new controller?
0066  74 BA                    C          jz      init_drive_lp           ;no, go init second drive on this one
0068  EB 83                    C          jmp     init_controller_lp      ;and go back .
                               C
                               C  ; Come here when an attempt to initialize a drive or controller fails.
                               C  ; If (at least) one good controller is found, then continue.
                               C
006A                           C  post_no_more_drives:
006A  FE 0E 0075 R             C          dec     byte ptr ds:[hf_num]            ;restore hf_num
                               C
006E  EB 13 90                 C          jmp     post_win                        ; could be FDU system only
                               C
                               C  ; Come here if there was clearly an error during initialization.
                               C
0071                           C  post_lose:
0071  FE 0E 0075 R             C          dec     byte ptr ds:[hf_num]            ;anticipation disproved
                               C
                               C  ; Output an error message
                               C
                               C  ;post_message:
0075  BD 000F                  C          mov     bp,0fh                          ;error flag
0078  BE D0A2 R                C          mov     si,cs:(offset h_bad_m)          ;string for error '1701' (HDU)
007B  E8 E5FA R                C          call    DRomString                      ;display message on screen
007E  E8 D0BB R                C          call    show_sense                      ;display the sense bits
0081  EB 18                    C          jmp     short p_disp
                               C
0083                           C  post_win:
0083  A0 0075 R                C          mov     al,byte ptr ds:[hf_num]         ; number of hard disks.
0086  0A C0                    C          or      al,al                           ; are there any fixed disks.
0088  75 08                    C          jnz     p_some
                               C
008A  BE D0AE R                C          mov     si,cs:(offset h_abs_m)          ; message for no HDU
008D  E8 E5FA R                C          call    DRomString
0090  EB 09                    C          jmp     short p_disp
                               C
0092                           C  p_some:
0092  E8 E650 R                C          call    DHexNib                         ; display the number
                               C
0095  BE D0A6 R                C          mov     si,cs:(offset h_good_m)         ; message for 'HDU'
0098  E8 E5FA R                C          call    DRomString
                               C
                               C
009B  E8 E619 R                C  p_disp: call    DCrLf
                               C
009E  E8 D5FA R                C          call    disable_disk_interrupts
00A1  C3                       C          ret
                               C
                               C
00A2  20 4E 6F 74              C  h_bad_m         db      ' Not'                  ; purposely no NUL!!!!!
00A6  20 52 65 61 64 79        C  h_good_m        db      ' Ready ',NUL
      20 00
00AE  20 4E 6F 74 20 50        C  h_abs_m         db      ' Not Present',NUL
      72 65 73 65 6E 74
      00
                               C
                               C
                               C
00BB                           C  h_init  endp
                               C
00BB                           C  show_sense      proc    near
                               C
                               C  ; Displays the error sense code as (ab), where a is the type and b is the code.
                               C
00BB  B8 0E28                  C          mov     ax,(0Eh*100h)+'('               ;display a '('
00BE  CD 10                    C          INT     10h
                               C
00C0  A0 0042 R                C          mov     al,byte ptr ds:[hd_error]       ;get the error code
00C3  24 3F                    C          and     al,sense0_msk                   ;isolate the type and code bits
00C5  E8 E643 R                C          call    DHexByte                        ;display error type and code
                               C
00C8  B8 0E29                  C          mov     ax,(0Eh*100h)+')'               ;display a ')'
00CB  CD 10                    C          INT     10h
                               C
00CD  C3                       C          ret
                               C
00CE                           C  show_sense      endp
                               C
00CE                           C  maybe_show_sense        proc    near
                               C
                               C  ; Show sense information if switch 8 is zero (closed) and 6 is one (open).
                               C  ; To be used for field maintenance.
                               C
00CE  50                       C          push    ax
00CF  80 3E 0074 R 00          C          cmp     byte ptr ds:[disk_status],0
00D4  74 11                    C          je      m_exit                          ;if no error, do not display
                               C          inport  al,switch_port                  ;read the switches
00D6  BA 0322                  C+         mov     dx,switch_port          ;get the port number for the base controller
00D9  02 16 0077 R             C+         add     dl,byte ptr ds:[port_off]       ;add port_off for the correct bx
00DD  EC                       C+         in      al,dx
00DE  24 A0                    C          and     al,0A0h                         ;mask switches 8 and 6
00E0  3C 80                    C          cmp     al,080h                         ;are they =10b?
```

```
D0E2  75 03                   C          jne     m_exit
                              C
D0E4  E8 D0BB R               C          call    show_sense                    ;if so, display sense
                              C
D0E7                          C m_exit:
D0E7  58                      C          pop     ax                            ;back to caller
D0E8  C3                      C          ret
D0E9                          C maybe_show_sense        endp
                              C
                              C ;====== bd3.asm        ==========================================
                              C ; This is called from the ROM Bios after the hard disk has been initialized.
                              C ; It attempts to read and initiate a master boot block from logical sector 0 of
                              C ; the floppy disk, and then from the hard disk.
                              C ;=============================================================================
                              C
                              C          assume  cs:code, ds:nothing, es:nothing, ss:nothing
                              C
D0E9                          C h_boot  proc    far
                              C
                              C ; Install vectors.
                              C
                              C          assume  cs:code, ds:abs0, es:abs0, ss:nothing
                              C
D0E9  33 C0                   C          xor     ax,ax                         ; satisfy assumptions
D0EB  8E D8                   C          mov     ds,ax                         ; ds = es = ax = abs0_seg
D0ED  8E C0                   C          mov     es,ax
                              C
D0EF  FA                      C          cli                                   ; disable during critical code.
                              C
                              C ; Install new int 41 HDU parameter table pointer.
                              C
D0F0  C7 06 0104 D14B R       C          mov     word ptr ds:[(4*41h)+0000h],cs:(offset hdu_parm_tbl)
D0F6  8C 0E 0106              C          mov     word ptr ds:[(4*41h)+0002h],cs
                              C
                              C ; Install new FDU parameter table pointer.
                              C
                              C ;        mov     word ptr ds:[int1Eloc+0000h],cs:(offset fd_parms)
                              C ;        mov     word ptr ds:[int1Eloc+0002h],cs
                              C
D0FA  FB                      C          sti                                   ; exit critical code.
                              C
                              C ; Attempt to reset the diskette. Motor startup time requires 3 tries.
                              C
                              C          assume  cs:code, ds:data, es:abs0, ss:nothing
                              C
D0FB  B8 0040                 C          mov     ax,data_seg                   ; satisfy assumptions.
D0FE  8E D8                   C          mov     ds,ax
                              C
D100  B9 0003                 C          mov     cx,3            ;3 retries
D103  B2 00                   C          mov     dl,0            ;drive number zero (soft disk A:)
D105  E8 D12A R               C          call    boot_a_disk     ;try to boot from floppy
D108  72 02                   C          jc      b_try_hdu       ;failed - try the hard disk
D10A  EB 13                   C          jmp     short b_ok      ;jump to the boot code if we succeeded
                              C
                              C ; else boot from the hard disk.
                              C
D10C                          C b_try_hdu:
D10C  B9 0003                 C          mov     cx,3            ;3 retries
D10F  B2 80                   C          mov     dl,80h          ;drive number 80h (hard disk C:)
D111  E8 D12A R               C          call    boot_a_disk     ;try to boot from hard disk
D114  72 0E                   C          jc      b_failed        ;if this failed too, then exit.
                              C
D116  26: 81 3E 7DFE AA55     C          cmp     word ptr es:[07C00h+510d],0AA55h      ;is it the code?
D11D  75 05                   C          jne     b_failed                              ;code tester
                              C
D11F                          C b_ok:
D11F  2E: FF 2E D126 R        C          jmp     dword ptr cs:[boot_indirect]  ;jump to boot code if correct
                              C
D124                          C b_failed:
D124  CD 18                   C          int     18h                     ; initiate reboot through ROM BASIC INT.
                              C
D126  7C00                    C boot_indirect   dw      07C00h          ; jump indirect to boot code at 0:7C00
D128  0000                    C                 dw      abs0_seg
                              C
D12A                          C h_boot  endp
                              C
                              C
D12A                          C boot_a_disk     proc
                              C
                              C ; Set dl=disk number and call this proc.
                              C ; It will reset and try to read cx times (in case there is a read error).
                              C ; If it succeeds, it will return with CY=0. If it fails, it will leave CY=1.
                              C
D12A                          C b_lp:
D12A  B4 00                   C          mov     ah,reset_int    ;reset the disk
D12C  CD 13                   C          int     13h
                              C
D12E  72 14                   C          jc      b_lost          ;if the operation lost
                              C
D130  B4 02                   C          mov     ah,read_int     ;if it worked, read from the disk
D132  B0 01                   C          mov     al,1            ;one sector
                              C
D134  BB 7C00                 C          mov     bx,07C00h       ;offset for the boot_area.
                              C
D137  51                      C          push    cx              ;save the retry count
                              C
D138  B5 00                   C          mov     ch,0            ;cylinder 0
D13A  B1 01                   C          mov     cl,1            ;sector 1 (lowest sector)
D13C  B6 00                   C          mov     dh,0            ;head 0
                              C
```

```
D13E  CD 13                   C          int     13h             ;attempt read
                              C
D140  59                      C          pop     cx              ;restore retry count
                              C
D141  72 01                   C          jc      b_lost          ;if it worked, return to caller
                              C
D143  C3                      C          ret                     ;return with no carry
                              C
D144                          C  b_lost:
D144  80 FC 80                C          cmp     ah,time_out     ;if it was a timeout,
D147  E0 E1                   C          loopne  b_lp            ;otherwise, another chance
D149  F9                      C          stc                     ;dropped through loop
D14A  C3                      C          ret                     ;return with carry
                              C
D14B                          C  boot_a_disk     endp
                              C
                              C
                              C
D14B                          C  hdu_parm_tbl:
                              C
                              C  ; The next six are the supported drives:
                              C
D14B  0132                    C                  parameter_table <,2d,,,,00h>          ;type 0: 5mb drive
D14D  02                      C
D14E  0132                    C
D150  0000                    C
D152  0B                      C
D153  00                      C
D154  0C                      C
D155  BA                      C
D156  28                      C
D157  00 00 00 00             C
                              C
D15B  0177                    C                  parameter_table <375d,8d,375d>        ;type 1: 24mb drive
D15D  08                      C
D15E  0177                    C
D160  0000                    C
D162  0B                      C
D163  05                      C
D164  0C                      C
D165  BA                      C
D166  28                      C
D167  00 00 00 00             C
                              C
D16B  0132                    C                  parameter_table <,6d,128d,256d>       ;type 2: 15mb drive
D16D  06                      C
D16E  0080                    C
D170  0100                    C
D172  0B                      C
D173  05                      C
D174  0C                      C
D175  BA                      C
D176  28                      C
D177  00 00 00 00             C
                              C
D17B  0132                    C                  parameter_table <>                    ;type 3: 10mb drive
D17D  04                      C
D17E  0132                    C
D180  0000                    C
D182  0B                      C
D183  05                      C
D184  0C                      C
D185  BA                      C
D186  28                      C
D187  00 00 00 00             C
                              C                                                        ;   (default)
D18B  0132                    C                  parameter_table <,2d,128d,306d>       ;type 4: Syquest SQ306
D18D  02                      C
D18E  0080                    C
D190  0132                    C
D192  0B                      C
D193  05                      C
D194  0C                      C
D195  BA                      C
D196  28                      C
D197  00 00 00 00             C
                              C
D19B  0284                    C                  parameter_table <644d,5d,128d,128d>   ;type 5: CDC Wren
D19D  05                      C
D19E  0080                    C
D1A0  0080                    C
D1A2  0B                      C
D1A3  05                      C
D1A4  0C                      C
D1A5  BA                      C
D1A6  28                      C
D1A7  00 00 00 00             C
                              C
                              C
                              C  ; ...End of hard disk parameters.
                              C
                              C  ;======= hd4.asm        ================================================
                              C  ;       Interrupt 13h - HDU I/O request
                              C  ;=============================================================================
                              C
D1AB                          C  h_io    proc    far
                              C          assume  cs:code, ds:nothing, es:nothing, ss:nothing
                              C
                              C  ; Debugging code to display all calls to this procedure.
```

```
                                C
                                C  ;       push    ax
                                C  ;       mov     al,ah
                                C  ;       call    DHexByte
                                C  ;       mov     al,dl
                                C  ;       call    DHexNib
                                C  ;       call    DColon
                                C  ;       pop     ax
                                C
                                C
D1AB  F6 C2 80                  C          test    dl,080h                 ; test HDU flag bit #7.
D1AE  75 05                     C          jnz     hard_disk_int13         ; is this an HDU command?
D1B0  CD 40                     C          int     40h                     ; if not, pass to the FDU driver
                                C
D1B2                            C  ret_from_40:
D1B2  CA 0002                   C          ret     2                       ; and then return, saving flags
                                C
D1B5                            C  hard_disk_int13:
D1B5  FB                        C          sti                             ; enable interrupts
D1B6  80 FC 00                  C          cmp     ah,reset_int            ; is this a reset command?
D1B9  75 09                     C          jne     h_save_regs
D1BB  CD 40                     C          int     40h                     ; if so, must reset FDU also
                                C
D1BD  80 FA 87                  C          cmp     dl,87h                  ; is the number absurd?
D1C0  77 F0                     C          ja      ret_from_40             ; if so, return without HDU
                                C
D1C2  B4 00                     C          mov     ah,reset_int            ; perform a hard disk reset
                                C
D1C4                            C  h_save_regs:
D1C4  57                        C          push    di
D1C5  56                        C          push    si
D1C6  06                        C          push    es
D1C7  1E                        C          push    ds
D1C8  53                        C          push    bx
D1C9  51                        C          push    cx
D1CA  52                        C          push    dx
                                C
                                C  ; Set up ds: segment.
                                C
                                C          assume  cs:code, ds:data, es:nothing, ss:nothing
                                C
D1CB  50                        C          push    ax
D1CC  B8 0040                   C          mov     ax,data_seg             ; satisfy assumptions.
D1CF  8E D8                     C          mov     ds,ax
D1D1  58                        C          pop     ax
                                C
D1D2  88 16 0076 R              C          mov     byte ptr ds:[control_byte],dl   ;retain drive specifier
                                C
D1D6  80 FC 01                  C          cmp     ah,status_int           ;is it a status request?
D1D9  75 03                     C          jne     h_make_cdb
D1DB  E9 D263 R                 C          jmp     status                  ;treat as a special case
                                C
                                C
                                C  ; Set up some of the CDB.
                                C
D1DE                            C  h_make_cdb:
D1DE  FE C9                     C          dec     cl                      ;bx sectors are 0-16, not 1-17
D1E0  88 0E 0044 R              C          mov     byte ptr ds:[cmd_block+2],cl    ;cylinder-hi and sector
D1E4  88 2E 0045 R              C          mov     byte ptr ds:[cmd_block+3],ch    ;cylinder-low
D1E8  A2 0046 R                 C          mov     byte ptr ds:[cmd_block+4],al    ;interleave or block count
                                C
D1EB  06                        C          push    es
D1EC  53                        C          push    bx
D1ED  E8 D5C1 R                 C          call    get_drivetype_vector    ;point to parms for this drive
D1F0  26: 8A 47 08              C          mov     al,es:[bx+p_control_byte]       ;get the control byte
D1F4  5B                        C          pop     bx
D1F5  07                        C          pop     es
D1F6  A2 0047 R                 C          mov     byte ptr ds:[cmd_block+5],al    ;put it in the cdb
                                C
                                C  ; Set up the port_off with the port offset for the drive.
                                C
D1F9  8A C2                     C          mov     al,dl                   ; HDU drive number
D1FB  24 7F                     C          and     al,07Fh                 ; mask off HDU flag bit #7
D1FD  3A 06 0075 R              C          cmp     al,byte ptr ds:[hf_num] ; count of HDU's
D201  73 57                     C          jae     h_bad_command           ; error if too high
                                C
D203  50                        C          push    ax
D204  D0 E0                     C          shl     al,1                    ;multiply by 2
D206  24 FC                     C          and     al,11111100b            ;provide port offset
D208  A2 0077 R                 C          mov     byte ptr ds:[port_off],al       ;store for use by port routine
D20B  58                        C          pop     ax
                                C
D20C  B1 05                     C          mov     cl,drivenum             ;shift for cdb drive#
D20E  D2 E0                     C          shl     al,cl
D210  80 E6 1F                  C          and     dh,(mask headnum)       ;get the bits for head number
D213  0A C6                     C          or      al,dh                   ;share drive# with head number
D215  A2 0043 R                 C          mov     byte ptr ds:[cmd_block+1],al    ;store into the drive/head byte
                                C
                                C
                                C  ; Set up an index to use the h_cmd_table.
                                C
D218  80 FC 14                  C          cmp     ah,max_int              ;highest-numbered valid int13 command
D21B  77 3D                     C          ja      h_bad_command           ;error if out of range
                                C
D21D  8A C4                     C          mov     al,ah                   ;get the command code
D21F  D0 E0                     C          shl     al,1                    ;multiply command by 4...
D221  D0 E0                     C          shl     al,1                    ;for indexing of table
D223  B4 00                     C          mov     ah,0                    ;zero top 8 bits
D225  05 D26D R                 C          add     ax,offset h_cmd_table   ;now we point to the command in the table
D228  8B F0                     C          mov     si,ax                   ;we'll use cs:[si] to index the table
```

```
                        C
                        C
                        C  ; Now we can finish setup of the command.
                        C  ; If it's a p_dma operation, the command subroutine will be
                        C  ; called with al=p_dma mode byte for 8237.
                        C
D22A  2E: 8A A6 0000    C          mov     ah,cs:[si].bx_opcode            ;get the opcode for the cdb
D22F  88 26 0042 R      C          mov     byte ptr ds:[cmd_block+0],ah    ;and put it in place
                        C
D233  2E: 8A 84 0003    C          mov     al,cs:[si].h_cmd_mode_dma       ;p_dma mode byte (read/write)
                        C
D238  C6 06 0074 R 00   C          mov     byte ptr ds:[disk_status],0     ; clear errors.
D23D  2E: FF 94 0001    C          call    cs:[si].h_cmd_subr      ;call subroutine to execute command
                        C
D242                    C  h_finish:
D242  E8 D5FA R         C          call    disable_disk_interrupts
D245  E8 D0CE R         C          call    maybe_show_sense
                        C
D248  8A 26 0074 R      C          mov     ah,byte ptr ds:[disk_status]    ; get the status code from oper.
D24C  80 FC 01          C          cmp     ah,1                            ; if nonzero, clear CF
D24F  F5                C          cmc                                     ; now CF reflects error status.
                        C
D250  5A                C          pop     dx
D251  59                C          pop     cx
D252  5B                C          pop     bx
D253  1F                C          pop     ds
D254  07                C          pop     es
D255  5E                C          pop     si
D256  5F                C          pop     di
                        C
D257  CA 0002           C          ret     2               ;return, preserving flags
                        C
                        C
                        C
                        C
D25A                    C  h_bad_command:
D25A  C6 06 0074 R 01   C          mov     byte ptr ds:[disk_status],cmd_error     ;bad command to int13
D25F  B0 00             C          mov     al,0                                    ;clear burst length
D261  EB DF             C          jmp     h_finish                                ;and exit
                        C
                        C  ; Special code for return_status.
                        C
D263                    C  status:
D263  A0 0074 R         C          mov     al,byte ptr ds:[disk_status]    ;return status from last command
D266  C6 06 0074 R 00   C          mov     byte ptr ds:[disk_status],0     ;this command was successful!
D26B  EB D5             C          jmp     h_finish                        ;and exit
                        C
                        C
                        C  ; This is the structure for the table which allows command dispatch.
                        C
                        C  h_cmd_struc      struc
0000  00                C  bx_opcode        db      0h      ;opcode for the bx (cdb byte 0)
0001  0000              C  h_cmd_subr       dw      0h      ;address of command routine
0003  00                C  h_cmd_mode_dma   db      0h      ;code for 8237 read/write
0004                    C  h_cmd_struc      ends
                        C
                        C  ; This is the table itself. Each entry corresponds to a command.
                        C
D26D                    C  h_cmd_table:
                        C
D26D  0C                C  h_cmd_struc      <assign_param_bx,assign_param>          ;00 reset
D26E  D504 R            C
D270  00                C
                        C
D271  00                C  h_cmd_struc      <0h,            0h>                    ;01 status
D272  0000              C
D274  00                C
                        C
D275  08                C  h_cmd_struc      <read_bx,       h_dma,  read_mode>     ;02 read
D276  D2C1 R            C
D278  47                C
                        C
D279  0A                C  h_cmd_struc      <write_bx,      h_dma,  write_mode>    ;03 write
D27A  D2C1 R            C
D27C  4B                C
                        C
D27D  05                C  h_cmd_struc      <chktrk_bx,     h_ndma>                ;04 chktrk
D27E  D33E R            C
D280  00                C
                        C
D281  06                C  h_cmd_struc      <fmttrk_bx,     h_ndma>                ;05 fmttrk
D282  D33E R            C
D284  00                C
                        C
D285  07                C  h_cmd_struc      <fmtbad_bx,     h_ndma>                ;06 fmtbad
D286  D33E R            C
D288  00                C
                        C
D289  04                C  h_cmd_struc      <fmtdrv_bx,     h_ndma>                ;07 fmtdrv
D28A  D33E R            C
D28C  00                C
                        C
D28D  00                C  h_cmd_struc      <0h,            read_param>            ;08 read_param
D28E  D4DF R            C
D290  00                C
                        C
D291  0C                C  h_cmd_struc      <assign_param_bx,assign_param>          ;09 assign_param
D292  D504 R            C
D294  00                C
                        C
```

```
D295  E5                  C  h_cmd_struc     <rlong_bx,       h_dma,   read_mode>      ;0a rlong
D296  D2C1 R              C
D298  47                  C
                          C
D299  E6                  C  h_cmd_struc     <wlong_bx,       h_dma,   write_mode>     ;0b wlong
D29A  D2C1 R              C
D29C  4B                  C
                          C
D29D  0B                  C  h_cmd_struc     <seek_bx,        h_ndma>                  ;0c seek
D29E  D33E R              C
D2A0  00                  C
                          C
D2A1  0C                  C  h_cmd_struc     <assign_param_bx,assign_param>            ;0d reset
D2A2  D504 R              C
D2A4  00                  C
                          C
D2A5  0E                  C  h_cmd_struc     <rbuff_bx,       h_dma,   read_mode>      ;0e rbuff
D2A6  D2C1 R              C
D2A8  47                  C
                          C
D2A9  0F                  C  h_cmd_struc     <wbuff_bx,       h_dma,   write_mode>     ;0f wbuff
D2AA  D2C1 R              C
D2AC  4B                  C
                          C
D2AD  00                  C  h_cmd_struc     <tst_drv_rdy_bx,h_ndma>                   ;10 tst_drv_rdy
D2AE  D33E R              C
D2B0  00                  C
                          C
D2B1  01                  C  h_cmd_struc     <recal_bx,       h_ndma>                  ;11 recal
D2B2  D33E R              C
D2B4  00                  C
                          C
D2B5  E0                  C  h_cmd_struc     <ramdiag_bx,     h_ndma>                  ;12 ramdiag
D2B6  D33E R              C
D2B8  00                  C
                          C
D2B9  E3                  C  h_cmd_struc     <drvdiag_bx,     h_ndma>                  ;13 drvgdiag
D2BA  D33E R              C
D2BC  00                  C
                          C
D2BD  E4                  C  h_cmd_struc     <bxdiag_bx,      h_ndma>                  ;14 bxdiag
D2BE  D33E R              C
D2C0  00                  C
                          C
                          C
                          C
                          C
D2C1                      C  h_io    endp
                          C
                          C  ; This procedure is called by int13 when a command requires a p_dma operation.
                          C  ; The al register should contain the h_cmd_mode_dma.
                          C
D2C1                      C  h_dma   proc
                          C
D2C1  FA                  C          cli                                      ; disable interrupts
D2C2  E6 0B               C          out     dma_mode,al                      ; set the p_dma mode
                          C
                          C  ; Convert es:bx to an absolute 20-bit address for data.
                          C
D2C4  8C C0               C          mov     ax,es                            ; get es
D2C6  B1 04               C          mov     cl,4                             ; rotate 4 bits (*16)
D2C8  D3 C0               C          rol     ax,cl                            ; now it's aligned with bx
D2CA  8A E8               C          mov     ch,al                            ; save the top nibble of address
D2CC  24 F0               C          and     al,0f0h                          ; clear low nibble
                          C
D2CE  03 D8               C          add     bx,ax                            ; add the offset of the address
D2D0  80 D5 00            C          adc     ch,0                             ; propagate CF (20-bit addition)
                          C                                                   ; ch = 20-bit address hi nibble
                          C                                                   ; bx = 20-bit offset of address
                          C
D2D3  E6 0C               C          out     dma_ff_clr,al                    ; clear the first/last ff
                          C
D2D5  8A C3               C          mov     al,bl
D2D7  E6 06               C          out     dma_addr_3,al                    ; output low byte of address
                          C
D2D9  8A C7               C          mov     al,bh
D2DB  E6 06               C          out     dma_addr_3,al                    ; output high byte of address
                          C
D2DD  8A C5               C          mov     al,ch                            ; restore top nibble of address
D2DF  24 0F               C          and     al,0Fh                           ; mask the nibble
D2E1  E6 82               C          out     hdu_dma_segm,al                  ; to bank select register
                          C
D2E3  A0 0042 R           C          mov     al,byte ptr ds:[cmd_block+0]     ; get command from cmd_block
                          C
                          C  ; If it is a rlong or wlong, it's limited to one block (516 chars)
                          C  ; Note that if rlong or wlong is requested, the length is not checked.
                          C
D2E6  3C E5               C          cmp     al,rlong_bx                      ; is command rlong_bx?
D2E8  74 38               C          je      h_dma_long
D2EA  3C E6               C          cmp     al,wlong_bx                      ; is command wlong_bx?
D2EC  74 34               C          je      h_dma_long
                          C
                          C  ; If it is a rbuff or wbuff, it's forced to one block (512 chars)
                          C  ; Note that if rbuff or wbuff is requested, the length is not checked.
                          C
D2EE  3C 0E               C          cmp     al,rbuff_bx                      ; is command rbuff_bx?
D2F0  74 3A               C          je      h_dma_buff
D2F2  3C 0F               C          cmp     al,wbuff_bx                      ; is command wbuff_bx?
D2F4  74 36               C          je      h_dma_buff
                          C
                          C  ; Set up the number of bytes for a normal command.
```

```
                                C
D2F6  80 3E 0046 R 80           C          cmp     byte ptr ds:[cmd_block+4],080h  ; too many blocks requested?
D2FB  77 39                     C          ja      h_dma_err                       ; (64k RAM/512 bytes per sector)
                                C
                                C  ; Determine number of bytes to transfer.
                                C
D2FD  B1 09                     C          mov     cl,9h                           ; multiply 512 times
D2FF  A0 0046 R                 C          mov     al,byte ptr ds:[cmd_block+4]    ; ...block count
D302  D3 E0                     C          shl     ax,cl                           ; to make byte count
                                C
D304                            C  h_dma_len:
D304  48                        C          dec     ax                              ; ax = byte count - 1
                                C
D305  E6 07                     C          out     dma_count_3,al                  ; output low byte of count
D307  86 C4                     C          xchg    al,ah
D309  E6 07                     C          out     dma_count_3,al                  ; output high byte of count
D30B  86 C4                     C          xchg    al,ah                           ; ax = byte count - 1
                                C
D30D  FB                        C          sti                                     ; critical code is over
                                C
D30E  03 C3                     C          add     ax,bx                           ; compute last byte to transfer
D310  72 24                     C          jc      h_dma_err                       ; if carry flag set, overflow!!
                                C
                                C  ; Enable and allow interrupts from p_dma, and issue command.
                                C
D312  B0 03                     C          mov     al,(mask_port_inte+mask_port_dmae)
D314  E8 D3DD R                 C          call    command                         ; send the cdb out to the bx
D317  72 31                     C          jc      h_err_chk                       ; return error
                                C
D319  B0 03                     C          mov     al,allow_dma3                   ; allow interrupts
D31B  E6 0A                     C          out     dma_mask_bit,al                 ; from the p_dma channel
                                C
D31D  E8 D43B R                 C          call    h_cmd_wait                      ; wait for command completion
D320  EB 28                     C          jmp     short h_err_chk                 ; check for possible error
                                C
                                C
                                C
                                C  ; For rlong and wlong commands:
                                C
D322                            C  h_dma_long:
D322  C6 06 0046 R 01           C          mov     byte ptr ds:[cmd_block+4],01h   ; one block maximum
D327  B8 0204                   C          mov     ax,512d + 4d                    ; length of data + ecc
D32A  EB D8                     C          jmp     h_dma_len
                                C
                                C  ; For rbuff and wbuff commands:
                                C
D32C                            C  h_dma_buff:
D32C  C6 06 0046 R 01           C          mov     byte ptr ds:[cmd_block+4],01h   ; one block maximum
D331  B8 0200                   C          mov     ax,512d                         ; length of data
D334  EB CE                     C          jmp     h_dma_len
                                C
                                C
D336                            C  h_dma_err:
D336  FB                        C          sti                                     ; enable interrupts
D337  F9                        C          stc                                     ; set carry flag.
D338  C6 06 0074 R 09           C          mov     byte ptr ds:[disk_status],dma_seg_error
D33D  C3                        C          ret
                                C
D33E                            C  h_dma   endp
                                C
                                C
                                C  ; This procedure is called by int13 when a non-p_dma command is requested.
                                C
D33E                            C  h_ndma  proc
                                C
D33E  B0 02                     C          mov     al,mask_port_inte       ; allow bx interrupt at cmd completion
D340  E8 D3DD R                 C          call    command                 ; send the cdb out to the bx
D343  72 05                     C          jc      h_err_chk               ; return error
                                C
D345  E8 D43B R                 C          call    h_cmd_wait              ; wait for the command to complete
D348  EB 00                     C          jmp     short h_err_chk         ; check for possible error
                                C
D34A                            C  h_ndma  endp
                                C
                                C
D34A                            C  h_err_chk       proc
                                C
D34A  80 3E 0074 R 00           C          cmp     byte ptr ds:[disk_statu~],0     ; check status flag from command
D34F  75 01                     C          jne     h_had_err
D351  C3                        C          ret                                     ; no error - looking good
                                C
                                C  ; Perform request sense command same drive number.
                                C
D352                            C  h_had_err:
D352  C6 06 0042 R 03           C          mov     byte ptr ds:[cmd_block+0],sense_bx
D357  B0 00                     C          mov     al,0                            ; interrupt and p_dma for bx
D359  E8 D3DD R                 C          call    command                         ; send out the command
D35C  72 75                     C          jc      e_lose                          ; sense command lost
                                C
D35E  B9 0004                   C          mov     cx,4                            ; accept 4 data + 1 status byte
D361  B8 0000                   C          mov     ax,0
D364  8B F8                     C          mov     di,ax                           ; used as index of array
                                C
D366                            C  e_read_sense:
D366  B4 0B                     C          mov     ah,(hdu_bsy+hdu_io+hdu_req)     ; bsy-cd+io+req
D368  E8 D418 R                 C          call    wait_for_status                 ; wait for status byte ready
D36B  72 66                     C          jc      e_lose                          ; jump if error
                                C          inport  al,read_port                    ; read data
D36D  BA 0320                   C+         mov     dx,read_port            ;get the port number for the base controller
D370  02 16 0077 R              C+         add     dl,byte ptr ds:[port_off]       ;add port_off for the correct bx
```

```
D374  EC                      C+          in      al,dx
D375  88 85 0042 R            C           mov     byte ptr ds:[hd_error+di],al    ; store in array
D379  47                      C           inc     di
D37A  E2 EA                   C           loop    e_read_sense
                              C
D37C  BA 0F                   C           mov     ah,(hdu_bsy+hdu_cd+hdu_io+hdu_req)      ;bsy+cd+io+req
D37E  E8 D418 R               C           call    wait_for_status                 ; wait for status byte ready
D381  72 50                   C           jc      e_lose                          ; jump if error
                              C           inport  al,read_port                    ; read the status byte
D383  BA 0320                 C+          mov     dx,read_port            ;get the port number for the base controller
D386  02 16 0077 R            C+          add     dl,byte ptr ds:[port_off]       ;add port_off for the correct bx
D38A  EC                      C+          in      al,dx
                              C
D38B  A8 02                   C           test    al,h_c_err                      ; check status byte
D38D  75 44                   C           jnz     e_lose                          ; leave if we had sense error
                              C
D38F  A0 0042 R               C           mov     al,byte ptr ds:[hd_error+0]     ; get first sense byte
D392  24 3F                   C           and     al,sense0_msk                   ; isolate the type and code bits
D394  BB D49F R               C           mov     bx,offset bx_error_table
D397  2E: D7                  C           xlat    cs:byte ptr bx_error_table      ; get the dos error code
D399  A2 0074 R               C           mov     byte ptr ds:[disk_status],al    ; store the status
D39C  3C 11                   C           cmp     al,ecc_used_err                 ; was it a correctable ecc err?
D39E  75 32                   C           jne     h_err_ret                       ; if not, return
                              C
                              C   ; Read ecc burst length.
                              C
D3A0  C6 06 0042 R 0D         C           mov     byte ptr ds:[cmd_block+0],read_ecc_bx
D3A5  B0 00                   C           mov     al,0                            ; interrupt mask = no interrupts
D3A7  E8 D3DD R               C           call    command
D3AA  72 27                   C           jc      e_lose
                              C
D3AC  B4 0B                   C           mov     ah,(hdu_bsy+hdu_io+hdu_req)     ; bsy-cd+io+req
D3AE  E8 D418 R               C           call    wait_for_status
D3B1  72 20                   C           jc      e_lose
                              C           inport  al,read_port                    ; read the burst length
D3B3  BA 0320                 C+          mov     dx,read_port            ;get the port number for the base controller
D3B6  02 16 0077 R            C+          add     dl,byte ptr ds:[port_off]       ;add port_off for the correct bx
D3BA  EC                      C+          in      al,dx
D3BB  8A C8                   C           mov     cl,al                           ; save it for a moment
                              C
D3BD  B4 0F                   C           mov     ah,(hdu_bsy+hdu_cd+hdu_io+hdu_req)      ;bsy+cd+io+req
D3BF  E8 D418 R               C           call    wait_for_status                 ; read the status byte
D3C2  72 0F                   C           jc      e_lose
                              C           inport  al,read_port
D3C4  BA 0320                 C+          mov     dx,read_port            ;get the port number for the base controller
D3C7  02 16 0077 R            C+          add     dl,byte ptr ds:[port_off]       ;add port_off for the correct bx
D3CB  EC                      C+          in      al,dx
D3CC  A8 02                   C           test    al,h_c_err                      ; check completion status
D3CE  75 03                   C           jnz     e_lose
                              C
D3D0  8A C1                   C           mov     al,cl                           ; get burst length back
D3D2                          C   h_err_ret:
D3D2  C3                      C           ret
                              C
                              C   ; Sense failed. Try to recover.
                              C
D3D3                          C   e_lose:
D3D3  E8 D504 R               C           call    assign_param            ;reset controller and assign parameters
D3D6  C6 06 0074 R FF         C           mov     byte ptr ds:[disk_status],sense_err     ;sense failed
D3DB  F9                      C           stc
D3DC  C3                      C           ret
                              C
D3DD                          C   h_err_chk       endp
                              C
                              C   ; Send a CDB to the bx.
                              C   ; The al register should be set to a mask_port_cmd.
                              C
D3DD                          C   command         proc
                              C
                              C           outport select_port,al          ;select controller
D3DD  BA 0322                 C+          mov     dx,select_port  ;get the port number for the base controller
D3E0  02 16 0077 R            C+          add     dl,byte ptr ds:[port_off]       ;add port_off for the correct bx
D3E4  EE                      C+          out     dx,al
                              C
                              C           outport mask_port,al            ;allow interrupts and maybe p_dma
D3E5  BA 0323                 C+          mov     dx,mask_port    ;get the port number for the base controller
D3E8  02 16 0077 R            C+          add     dl,byte ptr ds:[port_off]       ;add port_off for the correct bx
D3EC  EE                      C+          out     dx,al
                              C
D3ED  B4 0D                   C           mov     ah,(hdu_bsy+hdu_cd+hdu_req)     ;bsy+cd-io+req
D3EF  E8 D418 R               C           call    wait_for_status         ;wait for controller to setup for cmd
D3F2  72 23                   C           jc      c_finish                ;leave if timed out
D3F4  FC                      C           cld                             ;clear direction: count up with [si]
D3F5  BE 0042 R               C           mov     si,ds:(offset cmd_block)                ;use si as index
                              C
D3F8                          C   send_cdb_byte:
D3F8  AC                      C           lodsb                           ;get a byte of cdb
                              C           outport write_port,al           ;send it to bx
D3F9  BA 0320                 C+          mov     dx,write_port   ;get the port number for the base controller
D3FC  02 16 0077 R            C+          add     dl,byte ptr ds:[port_off]       ;add port_off for the correct bx
D400  EE                      C+          out     dx,al
D401  B9 0000                 C           mov     cx,0                    ;prepare for loop
                              C
D404                          C   c_wait:
                              C           inport  al,status_port          ;check status
D404  BA 0321                 C+          mov     dx,status_port          ;get the port number for the base controller
D407  02 16 0077 R            C+          add     dl,byte ptr ds:[port_off]       ;add port_off for the correct bx
D40B  EC                      C+          in      al,dx
D40C  A8 01                   C           test    al,hdu_req              ;is status valid?
D40E  E1 F4                   C           loopz   c_wait                  ;wait for valid or timeout
```

```
0410  24 07                C           and     al,07h                  ;mask low 3 bits
0412  3C 05                C           cmp     al,(hdu_cd+hdu_req)     ;is it still looking for cd+out?
0414  74 E2                C           je      send_cdb_byte           ;yes - go send another byte
0416  F8                   C           clc
0417                       C  o_finish:
0417  C3                   C           ret                             ;done sending all bytes, so return
                           C
0418                       C  command endp
                           C
                           C  ; Wait for status from bx (specified in ah), or else time out.
                           C
0418                       C  wait_for_status proc
                           C
0418  50                   C           push    ax
0419  51                   C           push    cx
041A  B9 0000              C           mov     cx,0
041D                       C  r_waiting:
                           C           inport  al,status_port              ;read bx status
041D  BA 0321              C+          mov     dx,status_port          ;get the port number for the base controller
0420  02 16 0077 R          C+          add     dl,byte ptr ds:[port_off]       ;add port_off for the correct bx
0424  EC                   C+          in      al,dx
0425  A8 01                C           test    al,hdu_req              ;look for validity of status
0427  E1 F4                C           loopz   r_waiting               ;loop if not valid yet
0429  24 0F                C           and     al,0fh                  ;get low 4 bits
042B  3A C4                C           cmp     al,ah                   ;test for the specified status
042D  74 09                C           je      bx_ready
042F  C6 06 0074 R 80      C           mov     byte ptr ds:[disk_status],time_out      ;took too long to select
0434  F9                   C           stc                                             ;flag error
0435                       C  r_finish:
0435  59                   C           pop     cx
0436  58                   C           pop     ax
0437  C3                   C           ret
                           C
0438                       C  bx_ready:
0438  F8                   C           clc                             ;got the req
0439  EB FA                C           jmp     r_finish
043B                       C  wait_for_status endp
                           C
                           C  ; Wait for a command to complete.
                           C
043B                       C  h_cmd_wait      proc
                           C
043B  FA                   C           cli
043C  E4 21                C           in      al,pic_1                ; get interrupt mask
043E  24 DF                C           and     al,0DFh                 ; un-mask HDU interrupt IR5
0440  E6 21                C           out     pic_1,al                ; set the controller
0442  FB                   C           sti
                           C
                           C           assume  es:abs0
                           C
0443  33 C0                C           xor     ax,ax                   ; es = ax = abs0_seg
0445  8E C0                C           mov     es,ax                   ; es points to abs0_seg segment
0447  26: C4 36 0104       C           les     si,dword ptr es:[(4*41h)]       ; es:si points to hdu_parm_tbl
                           C
                           C           assume  es:nothing
                           C
                           C  ; Determine timeout count.
                           C
044C  B7 00                C           mov     bh,0                            ;zero top byte of count
044E  26: 8A 5C 09         C           mov     bl,es:byte ptr [si].p_timeout   ;standard timeout
0452  80 3E 0042 R 04      C           cmp     byte ptr ds:[cmd_block+0],fmtdrv_bx     ;but is it a format
                           C                                                   ;drive command?
0457  75 07                C           jne     w1
0459  26: 8A 5C 0A         C           mov     bl,es:byte ptr [si].p_fmt_timeout       ;fmtdrv timeout
045D  EB 0C 90             C           jmp     w2
0460                       C  w1:
0460  80 3E 0042 R E3      C           cmp     byte ptr ds:[cmd_block+0],drvdiag_bx    ;but is it a drive
                           C                                                   ;diagnostic?
0465  75 08                C           jne     w2
0467  26: 8A 5C 0B         C           mov     bl,es:byte ptr [si].p_drvdiag_timeout   ;drvdiag timeout
                           C
                           C  ; Now we can wait for the interrupt.
                           C
046B                       C  w2:
046B  D1 E3                C           shl     bx,1                            ;double the timeout for the
                           C                                                   ;8086.
                           C
046D                       C  wait_for_irq5:
                           C           inport  al,status_port
046D  BA 0321              C+          mov     dx,status_port          ;get the port number for the base controller
0470  02 16 0077 R         C+          add     dl,byte ptr ds:[port_off]       ;add port_off for the correct bx
0474  EC                   C+          in      al,dx
0475  A8 20                C           test    al,hdu_irq5                     ;was there an interrupt yet?
0477  75 0D                C           jnz     w_its_done
0479  E2 F2                C           loop    wait_for_irq5
                           C
047B  4B                   C           dec     bx                              ;outer timing loop
047C  75 EF                C           jnz     wait_for_irq5
                           C
047E  C6 06 0074 R 80      C           mov     byte ptr ds:[disk_status],time_out      ;we fell through
0483  EB 0F 90             C           jmp     w_finish
                           C
0486                       C  w_its_done:
                           C           inport  al,read_port                    ;read status byte
0486  BA 0320              C+          mov     dx,read_port            ;get the port number for the base controller
0489  02 16 0077 R         C+          add     dl,byte ptr ds:[port_off]       ;add port_off for the correct bx
048D  EC                   C+          in      al,dx
048E  24 02                C           and     al,h_c_err                      ;get completion error bit
0490  08 06 0074 R          C           or      byte ptr ds:[disk_status],al    ;h_err_chk will look at it
                           C
```

```
D494                            C  w_finish:
D494  B0 00                     C          mov     al,0                            ;turn off controller interrupts
                                C          outport mask_port,al
D496  BA 0323                   C+         mov     dx,mask_port    ;get the port number for the base controller
D499  02 16 0077 R              C+         add     dl,byte ptr ds:[port_off]       ;add port_off for the correct bx
D49D  EE                        C+         out     dx,al
                                C
D49E  C3                        C          ret
                                C
D49F                            C  h_cmd_wait       endp
                                C
D49F                            C  h_data1 proc
                                C
D49F                            C  bx_error_table:
                                C  ; Type 0 errors
D49F  00 20 40 20               C          db      0, fdc_error, seek_error, fdc_error
D4A3  80 00 20 00               C          db      time_out, 0, fdc_error, 0
D4A7  40 00 00 00               C          db      seek_error, 0, 0, 0
D4AB  00 00 00 00               C          db      0, 0, 0, 0
                                C  ;Type 1 errors
D4AF  10 10 02 00               C          db      crc_error, crc_error, addr_mark_error, 0
D4B3  04 40 00 00               C          db      sect_not_found, seek_error, 0, 0
D4B7  11 0B 00 00               C          db      ecc_used_err, badtrack_err, 0, 0
D4BB  00 00 00 00               C          db      0, 0, 0, 0
                                C  ;Type 2 errors
D4BF  01 02 00 00               C          db      cmd_error, addr_mark_error, 0, 0
D4C3      0C [                  C          db      12 dup(0)
                00
                    ]           C
                                C
                                C  ;Type 3 errors
D4CF  20 20 10 00               C          db      fdc_error, fdc_error, crc_error, 0
D4D3      0C [                  C          db      12 dup(0)
                00
                    ]           C
                                C
                                C
D4DF                            C  h_data1 endp
                                C
                                C  ;======= hd5.asm        ================================================
                                C
D4DF                            C  read_param       proc
D4DF  06                        C          push    es                      ;save caller's es: register
D4E0  E8 D5C1 R                 C          call    get_drivetype_vector    ;loads es with pointer & bx with offset
                                C
D4E3  26: 8B 0F                 C          mov     cx,es:[bx].p_cyls       ;get number of cylinders
D4E6  83 E9 02                  C          sub     cx,2d                   ;subtract 2: zero origin and diag track
D4E9  86 E9                     C          xchg    ch,cl                   ;put low-order in ch
D4EB  D0 C9                     C          ror     cl,1                    ;put high-order 2 bits
D4ED  D0 C9                     C          ror     cl,1                    ;in top of cl
D4EF  80 C9 11                  C          or      cl,sectors_per_track    ;fixed, not variable
                                C
D4F2  26: 8A 77 02              C          mov     dh,es:[bx].p_heads      ;number of heads
D4F6  FE CE                     C          dec     dh                      ;zero origin
                                C
D4F8  8A 16 0075 R              C          mov     dl,byte ptr ds:[hf_num] ;count of hard disks
                                C
D4FC  07                        C          pop     es
                                C
                                C
D4FD  58                        C          pop     ax                      ;save the return address
D4FE  5B                        C          pop     bx                      ;trash the dx
D4FF  5B                        C          pop     bx                      ;and cx from the caller
                                C
D500  51                        C          push    cx                      ;push new parameters
D501  52                        C          push    dx                      ;for caller to receive
D502  50                        C          push    ax                      ;replace return address
                                C
D503  C3                        C          ret
D504                            C  read_param       endp
                                C
                                C  ; Perform a reset and assign parameters for the drives.
                                C  ; Note: parameters must be assigned for BOTH drives on the chosen controller.
                                C
D504                            C  assign_param     proc
                                C
D504  E8 D5B1 R                 C          call    reset_bx
                                C
D507  06                        C          push    es
                                C
D508  C6 06 0042 R 0C           C          mov     byte ptr ds:[cmd_block+0],assign_param_bx       ;set cdb
D50D  C6 06 0043 R 00           C          mov     byte ptr ds:[cmd_block+1],0                     ;assume drive 0
D512  80 26 0076 R FE           C          and     byte ptr ds:[control_byte],0feh
D517  E8 D532 R                 C          call    assign_param_1                                  ;assign params
D51A  72 14                     C          jc      ap_finish                                       ;jump if OK
                                C
D51C  07                        C          pop     es
D51D  06                        C          push    es
                                C
D51E  C6 06 0042 R 0C           C          mov     byte ptr ds:[cmd_block+0],assign_param_bx       ;set cdb
D523  C6 06 0043 R 20           C          mov     byte ptr ds:[cmd_block+1],(mask drivenum)       ;assume drive 1
D528  80 0E 0076 R 01           C          or      byte ptr ds:[control_byte],01h
D52D  E8 D532 R                 C          call    assign_param_1                                  ;assign params
                                C
D530                            C  ap_finish:
D530  07                        C          pop     es
D531  C3                        C          ret
                                C
D532                            C  assign_param     endp
                                C
```

```
                                C
D532                            C  assign_param_1  proc
D532  B0 00                     C          mov     al,0                    ;no interrupts desired
D534  E8 D3DD R                 C          call    command                 ;send the cdb
D537  72 63                     C          jc      a_finish
                                C
                                C  ; Send the 8 bytes of drive parameters as PIO.
                                C
D539  E8 D5C1 R                 C          call    get_drivetype_vector    ;snare pointer to the table
                                C
                                C  ; send msb before lsb
                                C
D53C  26: 8A 47 01              C          mov     al,es:byte ptr [bx].p_cyls+1
D540  E8 D59D R                 C          call    data_out
D543  72 57                     C          jc      a_finish
                                C
D545  26: 8A 07                 C          mov     al,es:byte ptr [bx].p_cyls
D548  E8 D59D R                 C          call    data_out
D54B  72 4F                     C          jc      a_finish
                                C
D54D  26: 8A 47 02              C          mov     al,es:[bx].p_heads
D551  E8 D59D R                 C          call    data_out
D554  72 46                     C          jc      a_finish
                                C
D556  26: 8A 47 04              C          mov     al,es:byte ptr [bx].p_write_cur+1
D55A  E8 D59D R                 C          call    data_out
D55D  72 3D                     C          jc      a_finish
                                C
D55F  26: 8A 47 03              C          mov     al,es:byte ptr [bx].p_write_cur
D563  E8 D59D R                 C          call    data_out
D566  72 34                     C          jc      a_finish
                                C
D568  26: 8A 47 06              C          mov     al,es:byte ptr [bx].p_precomp+1
D56C  E8 D59D R                 C          call    data_out
D56F  72 2B                     C          jc      a_finish
                                C
D571  26: 8A 47 05              C          mov     al,es:byte ptr [bx].p_precomp
D575  E8 D59D R                 C          call    data_out
D578  72 22                     C          jc      a_finish
                                C
D57A  26: 8A 47 07              C          mov     al,es:[bx].p_ecc_len
D57E  E8 D59D R                 C          call    data_out
D581  72 19                     C          jc      a_finish
                                C
                                C  ; Note that control byte is handled differently... See near h_make_cdb.
                                C
D583  B4 0F                     C          mov     ah,(hdu_bsy+hdu_cd+hdu_io+hdu_req)      ;bsy+cd+io+req
D585  E8 D418 R                 C          call    wait_for_status
D588  72 12                     C          jc      a_finish
                                C
                                C          inport  al,read_port                            ;read the status byte
D58A  BA 0320                   C+         mov     dx,read_port            ;get the port number for the base controller
D58D  02 16 0077 R              C+         add     dl,byte ptr ds:[port_off]       ;add port_off for the correct bx
D591  EC                        C+         in      al,dx
D592  A8 02                     C          test    al,h_c_err                      ;did we do ok?
D594  74 06                     C          jz      a_finish
D596  C6 06 0074 R 07           C          mov     byte ptr ds:[disk_status],init_err      ;no
D59B  F9                        C          stc
                                C
D59C                            C  a_finish:
D59C  C3                        C          ret
                                C
D59D                            C  assign_param_1  endp
                                C
                                C  ; Send a byte of data out the write_port.
                                C
D59D                            C  data_out        proc
                                C
D59D  50                        C          push    ax
D59E  B4 09                     C          mov     ah,(hdu_bsy+hdu_req)    ;bsy-cd-io+req
D5A0  E8 D418 R                 C          call    wait_for_status
D5A3  58                        C          pop     ax
D5A4  73 01                     C          jnc     do_ready        ;if it didn't time out, output the data
D5A6  C3                        C          ret
                                C
D5A7                            C  do_ready:
                                C          outport write_port,al           ;write it out
D5A7  BA 0320                   C+         mov     dx,write_port   ;get the port number for the base controller
D5AA  02 16 0077 R              C+         add     dl,byte ptr ds:[port_off]       ;add port_off for the correct bx
D5AE  EE                        C+         out     dx,al
D5AF  F8                        C          clc                                     ;no problem
D5B0  C3                        C          ret
                                C
D5B1                            C  data_out        endp
                                C
D5B1                            C  reset_bx        proc
                                C
                                C          outport reset_port,al           ;reset controller
D5B1  BA 0321                   C+         mov     dx,reset_port   ;get the port number for the base controller
D5B4  02 16 0077 R              C+         add     dl,byte ptr ds:[port_off]       ;add port_off for the correct bx
D5B8  EE                        C+         out     dx,al
D5B9  51                        C          push    cx
D5BA  B9 0100                   C          mov     cx,100h                 ;wait in a loop
D5BD  E2 FE                     C          loop    $
D5BF  59                        C          pop     cx
D5C0  C3                        C          ret
D5C1                            C  reset_bx        endp
                                C
                                C  ; Returns with es:[bx] pointing to the drivetype vector for the selected drive.
                                C  ;
```

```
                        ;       The bits are selected from the switch as follows:
                        ;
                        ;       switch          | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
                        ;       bit               0   1   2   3   4   5   6   7
                        ;       drive 0           0   1           2   3
                        ;       drive 1                   0   1           2   3
                        ;
D5C1                    get_drivetype_vector    proc

D5C1  50                        push    ax

                                assume  es:abs0

D5C2  33 C0                     xor     ax,ax                           ; es = ax = abs0_seg
D5C4  8E C0                     mov     es,ax                           ; es points to abs0_seg segment
D5C6  26: C4 1E 0104            les     bx,dword ptr es:[(4*41h)]       ; es:bx points to hdu_parm_tbl

                                assume  es:nothing

D5CB  52                        push    dx
                                inport  al,switch_port          ;read switches on controller board
D5CC  BA 0322                   mov     dx,switch_port          ;get the port number for the base controller
D5CF  02 16 0077 R              add     dl,byte ptr ds:[port_off]       ;add port_off for the correct bx
D5D3  EC                        in      al,dx
D5D4  5A                        pop     dx
D5D5  24 7F                     and     al,07fh                 ;ignore top switch!

D5D7  F6 06 0076 R 01           test    byte ptr ds:[control_byte],01b  ;was this controller drive one?
D5DC  75 04                     jnz     g_mask
D5DE  D0 E8                     shr     al,1                    ;to get drive zero,
D5E0  D0 E8                     shr     al,1                    ;shift twice right

D5E2                    g_mask:
D5E2  24 33                     and     al,00110011b            ;get switch_mask bits for this drivetype
D5E4  8A E0                     mov     ah,al                   ;get top two bits from bits 4&5
D5E6  D0 EC                     shr     ah,1                    ;..into bits 2&3
D5E8  D0 EC                     shr     ah,1
D5EA  0A C4                     or      al,ah                   ;combine with low 2 bits

D5EC  D0 E0                     shl     al,1                    ;multiply by 16 for offset in table
D5EE  D0 E0                     shl     al,1
D5F0  D0 E0                     shl     al,1
D5F2  D0 E0                     shl     al,1
D5F4  B4 00                     mov     ah,0
D5F6  03 D8                     add     bx,ax                   ;add offset to base of param_table

D5F8  58                        pop     ax
D5F9  C3                        ret
D5FA                    get_drivetype_vector    endp


D5FA                    disable_disk_interrupts proc

                        ; Disables interrupts from all bx controllers on the system.

D5FA  52                        push    dx
D5FB  51                        push    cx
D5FC  50                        push    ax
D5FD  1E                        push    ds

                        ; Set up ds: segment.

                                assume  ds:data

D5FE  B8 0040                   mov     ax,data_seg                     ; satisfy assumptions.
D601  8E D8                     mov     ds,ax

D603  FA                        cli
D604  E4 21                     in      al,pic_1                        ; get interrupt mask
D606  0C 20                     or      al,020h                         ; turn off IR5
D608  E6 21                     out     pic_1,al                        ; and set the new mask
D60A  FB                        sti

D60B  B5 00                     mov     ch,0
D60D  8A 0E 0075 R              mov     cl,byte ptr ds:[hf_num]         ;number of drives
D611  41                        inc     cx                              ;make it into
D612  D1 E9                     shr     cx,1                            ;...number of controllers
D614  74 10                     jz      v_disallow                      ;if zero, disallow

D616                    v_reset_bx:

                        ; Reset controller cx by sending mask byte.

D616  B0 00                     mov     al,0                            ;mask to impose on bx
D618  8B D1                     mov     dx,cx                           ;get controller number
D61A  4A                        dec     dx                              ;we go from 0-3, not 1-4
D61B  D1 E2                     shl     dx,1                            ;make it an offset (4 ports/bx)
D61D  D1 E2
D61F  81 C2 0323                add     dx,mask_port                    ;now bx-> controller
D623  EE                        out     dx,al

D624  E2 F0                     loop    v_reset_bx

D626                    v_disallow:
D626  B0 07                     mov     al,disallow_dma3        ;command to turn off dma3 interrupts
D628  E6 0A                     out     dma_mask_bit,al         ;send it

                                assume  ds:nothing
D62A  1F                        pop     ds
```

```
D62B  58                C           pop     ax
D62C  59                C           pop     cx
D62D  5A                C           pop     dx
D62E  C3                C           ret
                        C
D62F                    C   disable_disk_interrupts endp
                        C
                        C   ;--------------------------------------------------------------------------
                        C   ;       Interrupt 0Dh - handle interrupt from controller
                        C   ;--------------------------------------------------------------------------
                        C
                        C   ; This is run when the hard disk causes an interrupt due to command completion.
                        C
D62F                    C   h_int   proc
                        C
D62F  50                C           push    ax
                        C
D630  B0 20             C           mov     al,pic_eoi                      ;command for 8259
D632  E6 20             C           out     pic_0,al
                        C
D634  B0 07             C           mov     al,disallow_dma3                ;command for 8237
D636  E6 0A             C           out     dma_mask_bit,al
                        C
D638  E4 21             C           in      al,pic_1                        ; get interrupt mask
D63A  0C 20             C           or      al,020h                         ; turn off IR5
D63C  E6 21             C           out     pic_1,al                        ; and set the new mask
                        C
D63E  58                C           pop     ax
D63F  CF                C           iret
                        C
D640                    C   h_int   endp
                        C
                        C   ;------- dtcfmt.asm     ---------------------------------------------
                        C   ;       HDU Physical Formatting Utility
                        C   ;
                        C   ; Formatting utility to perform physical format of hard disk drive
                        C   ; using DTC-5150BX controller.  Calls int 13h to provide disk services.
                        C   ;
                        C   ; Use this program when you first use a new disk. Then use FDISK and
                        C   ; FORMAT programs to finish the job.
                        C   ;
                        C   ; The user can supply an argument number for any hard disk on the system.
                        C   ; ex:  1=drive c, 2=drive d.
                        C   ;--------------------------------------------------------------------------
                        C
                        C           assume  cs:code, ds:nothing, es:nothing, ss:nothing
                        C
D640                    C   h_fmt   proc    near
                        C
                        C   ; Write greeting and prompt for drive number.
                        C   ; Drives are numbered the same way as in the industry standard "FDISK" utility.
                        C
D640  BE D6AC R         C           mov     si,cs:(offset h_intro_m)        ;greeting string
D643  E8 E5FA R         C           call    DRomString
                        C
D646  32 E4             C           xor     ah,ah                   ; ah = 0.
D648  CD 16             C           INT     16h                     ; get a keystroke.
                        C
D64A  3C 71             C           cmp     al,'q'                  ;check for quit code
D64C  74 45             C           je      finished                ;eject if user typed a Q
D64E  3C 51             C           cmp     al,'Q'
D650  74 41             C           je      finished
                        C
D652  E8 E619 R         C           call    DCrLf
                        C
D655  2C 31             C           sub     al,'1'                  ; range should be '1' to '8'
D657  72 3E             C           jc      no_hard_disk            ; if negative, incorrect number
D659  0C 80             C           or      al,080h                 ; set HDU flag bit #7.
D65B  8A D0             C           mov     dl,al                   ; prepare drive code for int13
                        C
                        C   ; Create a 512 byte sector buffer of 6Ch in RAM (on the stack!).
                        C
D65D  FC                C           cld                             ; clear direction flag.
D65E  06                C           push    es                      ; save registers.
D65F  16                C           push    ss
D660  07                C           pop     es                      ; es gets ss.
D661  81 EC 0200        C           sub     sp,512                  ; make room on stack!
                        C
D665  8B FC             C           mov     di,sp                   ; es:di points to stack buffer.
D667  B8 6C6C           C           mov     ax,06C6Ch               ; word fill pattern.
D66A  B9 0100           C           mov     cx,(512/2)              ; number of words to fill.
D66D  F3/ AB            C           rep     stosw                   ; fill stack buffer.
                        C                                           ; di saves original stack pointer
                        C
                        C   ; Write the stack sector buffer with the 06Ch code to fill data fields on drive.
                        C
D66F  B8 0F01           C           mov     ax,(wbuff_int*100h)+1   ; prepare for write sector of 1 buffer.
D672  8B DC             C           mov     bx,sp                   ; es:bx points to stack buffer (of 06Ch)
                        C                                           ; cx is zero.
D674  B6 00             C           mov     dh,0                    ; clear head number.
                        C
D676  CD 13             C           INT     13h                     ; call for write sector buffer
                        C
D678  8B E7             C           mov     sp,di                   ; restore original stack pointer.
D67A  07                C           pop     es                      ; restore registers.
                        C
D67B  72 1A             C           jc      no_hard_disk            ; error writing sector buffer
                        C
                        C   ; Print message that we are formatting the drive.
                        C
```

```
D67D  BE D733 R            C          mov     si,cs:(offset h_fmt_m)  ;text for informational message
D680  E8 E5FA R            C          call    DRomString
                           C
                           C ; Format the drive.
                           C
D683  B8 0706              C          mov     ax,(fmtdrv_int*100h)+6  ; format drive with interleave of 6
D686  B9 0001              C          mov     cx,01h                  ; start with sector zero
                           C
                           C ; Note that dx is retained from above.
                           C
D689  CD 13                C          INT     13h                     ; format the drive
D68B  72 12                C          jc      format_error            ; problem during format
                           C
D68D  BE D750 R            C          mov     si,cs:(offset h_pass_m)
D690  E8 E5FA R            C          call    DRomString
                           C
D693                       C finished:
D693  E8 E619 R            C          call    DCrLf
D696  C3                   C          ret                                             ; return to caller
                           C
                           C
D697                       C no_hard_disk:
D697  BE D784 R            C          mov     si,cs:(offset h_none_m)         ;give error message
D69A  E8 E5FA R            C          call    DRomString
D69D  EB F4                C          jmp     short finished                  ;exit
                           C
                           C
D69F                       C format_error:
D69F  BE D7B7 R            C          mov     si,cs:(offset h_err_m)          ;give error message
D6A2  E8 E5FA R            C          call    DRomString
                           C
D6A5  8A C4                C          mov     al,ah                           ; print error code in ah
D6A7  E8 E643 R            C          call    DHexByte
                           C
D6AA  EB E7                C          jmp     short finished
                           C
D6AC                       C h_fmt    endp
                           C
D6AC                       C h_data2 proc
                           C
D6AC  46 69 78 65 64 20    C h_intro_m        db      'Fixed Disk Formatting Utility',CR,LF
      44 69 73 6B 20 46    C
      6F 72 6D 61 74 74    C
      69 6E 67 20 55 74    C
      69 6C 69 74 79 0D    C
      0A                   C
D6CB  41 6C 6C 20 64 61    C                  db      'All data on the specified fixed disk will be erased.'
      74 61 20 6F 6E 20    C
      74 68 65 20 73 70    C
      65 63 69 66 69 65    C
      64 20 66 69 78 65    C
      64 20 64 69 73 6B    C
      20 77 69 6C 6C 20    C
      62 65 20 65 72 61    C
      73 65 64 2E          C
D6FF  0D 0A                C                  db      CR,LF
D701  45 6E 74 65 72 20    C                  db      'Enter fixed disk number (1 to 8) or "Q" to quit: ',NUL
      66 69 78 65 64 20    C
      64 69 73 6B 20 6E    C
      75 6D 62 65 72 20    C
      28 31 20 74 6F 20    C
      38 29 20 6F 72 20    C
      22 51 22 20 74 6F    C
      20 71 75 69 74 3A    C
      20 00                C
                           C
D733  0D 0A 46 6F 72 6D    C h_fmt_m          db      CR,LF,'Formatting Fixed Disk...',CR,LF,NUL
      61 74 74 69 6E 67    C
      20 46 69 78 65 64    C
      20 44 69 73 6B 2E    C
      2E 2E 0D 0A 00       C
                           C
D750  46 6F 72 6D 61 74    C h_pass_m         db      'Format is complete.',CR,LF
      20 69 73 20 63 6F    C
      6D 70 6C 65 74 65    C
      2E 0D 0A             C
D765  50 72 6F 63 65 65    C                  db      'Proceed with FDISK and FORMAT.',NUL
      64 20 77 69 74 68    C
      20 46 44 49 53 4B    C
      20 61 6E 64 20 46    C
      4F 52 4D 41 54 2E    C
      00                   C
                           C
D784  45 72 72 6F 72 3A    C h_none_m         db      'Error: No fixed disk drive exists for this number.',NUL
      20 4E 6F 20 66 69    C
      78 65 64 20 64 69    C
      73 6B 20 64 72 69    C
      76 65 20 65 78 69    C
      73 74 73 20 66 6F    C
      72 20 74 68 69 73    C
      20 6E 75 6D 62 65    C
      72 2E 00             C
                           C
D7B7  46 6F 72 6D 61 74    C h_err_m          db      'Format Error.  Code: ',NUL
      20 45 72 72 6F 72    C
      2E 20 20 43 6F 64    C
      65 3A 20 00          C
                           C
D7CD                       C h_data2 endp
                           C
```

```
D7CD                    C  code    ends
                        C  include graph.asm
                        C
                        C  ;===========================================================================
                        C  ;       Filename:       graph.src
                        C  ;
                        C  ;       This module includes the four graphics functions for INT 10h.
                        C  ;
                        C  ;
                        C  ;===========================================================================
                        C
                        C  page
                        C  ;---------------------------------------------------------------------------
                        C  ;       INT 10h Graphics Support
                        C  ;
                        C  ;       Must preserve bx, cx, dx and return values in ax
                        C  ;
                        C  ;       All other registers are saved and restored by video dispatcher.
                        C  ;
                        C  ;       Entered with the following in registers:
                        C  ;               al, bx, cx, dx intact (set by INT 10 invoker)
                        C  ;               ah = v_mode
                        C  ;---------------------------------------------------------------------------
                        C
D7CD                    C  code    segment public 'ROM'
                        C          assume  cs:code, ds:data, es:v_ram, ss:nothing
                        C
                        C  ;===========================================================================
                        C  ;
                        C  ;       Read Light Pen                  function code = 04h
                        C  ;
                        C  ;       Input:  None.
                        C  ;       Output: ah      = 0 light pen switch not down/not triggered
                        C  ;               ah      = 1 implies:
                        C  ;               (dh,dl) = (row,col) of character light pen
                        C  ;                         position from (0,0)
                        C  ;                       ch      = raster line (0-199)
                        C  ;                       bx      = pixel column (0-319,0-639)
                        C  ;
                        C  ;       Trash:  None.   ???
                        C  ;
                        C  ;---------------------------------------------------------------------------
                        C
D7CD                    C  grf_light_pen   proc    near
                        C
D7CD  32 E4             C          xor     ah,ah                   ; return ah = 0 for now (al intact)...
D7CF  C3                C          ret
                        C
D7D0                    C  grf_light_pen   endp
                        C
                        C  page
                        C  ;===========================================================================
                        C  ;
                        C  ;       Read Dot                        function code = 0Dh
                        C  ;
                        C  ;       reads a pixel from the indicated location returning its value.
                        C  ;       valid in graphics modes only.
                        C  ;
                        C  ;       Input:
                        C  ;               AH = CRT Mode
                        C  ;               DX = row (0..199 in Med. & Hi-Res., 0..399 in Ultra Res Mode)
                        C  ;               CX = column (0..319 in Med, 0..639 in Hi & Ultra Res. Modes)
                        C  ;
                        C  ;       Output:
                        C  ;               AL = dot value read
                        C  ;               AH = mask of pixel in byte (from g_addr)
                        C  ;
                        C  ;       Saved:  BX, CX, DX (Video dispatcher saves the rest)
                        C  ;
                        C  ;---------------------------------------------------------------------------
                        C
D7D0                    C  grf_read_dot    proc    near
                        C
D7D0  52                C          push    dx                      ; save registers
D7D1  51                C          push    cx
                        C
D7D2  50                C          push    ax                      ; save crt mode (high byte)
D7D3  E8 D815 R         C          call    g_addr                  ; compute address & mask
D7D6  84 C4             C          mov     al,ah                   ; copy mask
D7D8  26: 22 05         C          and     al,byte ptr es:[di]     ; AND with byte containing pixel
D7DB  5A                C          pop     dx                      ; dh = crt mode
D7DC  FE C1             C          inc     cl                      ; prep for wraparound left (B & W)
D7DE  80 FE 05          C          cmp     dh,5                    ; color?
D7E1  7F 02             C          jg      g_jsfy_dot              ; jump if not
D7E3  FE C1             C          inc     cl                      ; prep for wraparound left (color)
D7E5                    C  g_jsfy_dot:
D7E5  D2 C0             C          rol     al,cl                   ; right justify pixel
                        C
D7E7  59                C          pop     cx                      ; restore registers
D7E8  5A                C          pop     dx
D7E9  C3                C          ret
                        C
D7EA                    C  grf_read_dot    endp
                        C
                        C  page
                        C  ;===========================================================================
                        C  ;
                        C  ;       Write Dot                       function code = 0Ch
                        C  ;
                        C  ;       writes a pixel, with the indicated value, at the indicated location.
```

```
                        C  ;       valid in graphics modes only.
                        C  ;
                        C  ;       Input:
                        C  ;               AL = dot value to write (1 or 2 bits depending on mode,
                        C  ;                    right justified).  Bit 7 = 1 specifies XOR the value
                        C  ;                    with the pixel at DX, CX
                        C  ;               AH = CRT Mode
                        C  ;               DX = row (0-399)      (the actual value depends on the mode)
                        C  ;               CX = column (0-639)   (the values are not range checked)
                        C  ;
                        C  ;       Saved:  AX, BX, CX, DX (Video dispatcher saves the rest)
                        C  ;
                        C  ;--------------------------------------------------------------------------
                        C
D7EA                    C  grf_write_dot   proc    near
                        C
D7EA 52                 C          push    dx              ; preserve working registers
D7EB 51                 C          push    cx
D7EC 50                 C          push    ax
                        C
D7ED E8 D815 R          C          call    g_addr          ; compute address & mask
D7F0 26: 8A 05          C          mov     al,byte ptr es:[di] ; fetch byte from video memory
D7F3 5A                 C          pop     dx              ; get v_mode & dot value to write
D7F4 52                 C          push    dx              ; resave
D7F5 0A D2              C          or      dl,dl           ; is the XOR bit set?
D7F7 78 06              C          js      g_xorbit        ; jump if yes
D7F9 F6 D4              C          not     ah              ; invert mask
D7FB 22 C4              C          and     al,ah           ; clear bits for new pixel
D7FD F6 D4              C          not     ah              ; restore mask for later
D7FF                    C  g_xorbit:
D7FF FE C1              C          inc     cl              ; prep for wraparound right (b&w)
D801 80 FE 05           C          cmp     dh,5            ; color ?
D804 7F 02              C          jg      g_align_dot     ; jump if no
D806 FE C1              C          inc     cl              ; prep for wraparound right (color)
D808                    C  g_align_dot:
D808 D2 CA              C          ror     dl,cl           ; align new pixel
D80A 22 D4              C          and     dl,ah           ; strip off non-pixel bits (xor bit, etc)
D80C 32 C2              C          xor     al,dl           ; OR or XOR in new pixel depending on xor bit
D80E 26: 88 05          C          mov     byte ptr es:[di],al ; update video memory
                        C
D811 58                 C          pop     ax              ; restore registers
D812 59                 C          pop     cx
D813 5A                 C          pop     dx
D814 C3                 C          ret
                        C
D815                    C  grf_write_dot   endp
                        C
                        C  page
                        C  ;==========================================================================
                        C  ;
                        C  ;       This subroutine determines the video RAM byte location
                        C  ;       of the indicated row column value.  The current graphics mode
                        C  ;       is taken into account.
                        C  ;
                        C  ;       INPUT:
                        C  ;               AH = current graphics mode
                        C  ;               DX = row value (0-399)
                        C  ;               CX = column value (0-639)
                        C  ;
                        C  ;       OUTPUT:
                        C  ;               DI = byte address of pixel location in video ram
                        C  ;               AH = mask of pixel in byte
                        C  ;               CL = # of bits from left end of byte to leftmost bit in mask
                        C  ;
                        C  ;       DESTROYED:
                        C  ;               AL, CX, DX, BP
                        C  ;
                        C  ;--------------------------------------------------------------------------
                        C
D815                    C  g_addr          proc    near
                        C
                        C  ; convert row-count to a "mod 4" value;
                        C  ;   multiply it by 2000H for offset to start of v_ram subarea
                        C
D815 51                 C          push    cx                      ; save column count
D816 BD 0001            C          mov     bp,1                    ; shift count; assume not 640x400
D819 8B CD              C          mov     cx,bp                   ; multiplier; assume not 640x400
D81B 80 FC 40           C          cmp     ah,64                   ; video mode 64 or 72?
D81E 72 06              C          jb      g_skp_1                 ;       -no: jmp
D820 BD 0002            C          mov     bp,2                    ;       -yes; change shift count
D823 B9 0003            C          mov     cx,3                    ;       -yes: change multiplier
                        C
D826                    C  g_skp_1:
D826 33 FF              C          xor     di,di                   ; init offset amount
D828 23 CA              C          and     cx,dx                   ; get least sig. bit(s) from row count
D82A 74 06              C          jz      g_skp_3                 ;       jmp if nothing to add
D82C                    C  g_skp_2:                                ; mini-multiply loop
D82C 81 C7 2000         C          add     di,2000H                ;       add v_ram sub-area size
D830 E2 FA              C          loop    g_skp_2                 ;       do it again (maybe)
                        C
                        C  ; add bytes for the row coordinate to the offset into the v_ram subarea
                        C
D832                    C  g_skp_3:
D832 57                 C          push    di                      ; temp. store offset
D833 8B FA              C          mov     di,dx                   ; DX<--row count
D835 8B CD              C          mov     cx,bp                   ; CX<--mode-related shift val
D837 D3 EF              C          shr     di,cl                   ; get #rows into a v_ram subarea
                        C
D839 8B D7              C          mov     dx,di                   ; save a copy for a moment
D83B B9 0406            C          mov     cx,0406H                ; mult. di by 80 [rows-->byte offset]
```

```
D83E  D3 E7              C          sal     di,cl                    ;       [fast multiply-
D840  8A CD              C          mov     cl,ch                    ;           by-80]
D842  D3 E2              C          sal     dx,cl
D844  03 FA              C          add     di,dx                    ; #subarea byte offset for this row
                         C
D846  59                 C          pop     cx                       ; retrieve subarea beginning's offset
D847  03 F9              C          add     di,cx                    ; offset from start of v_ram
D849  5A                 C          pop     dx                       ; restore column count
                         C
                         C  ; find column offset
                         C
D84A  B9 0703            C          mov     cx,703H                  ; initialize for black & white
D84D  80 FC 05           C          cmp     ah,5                     ; color ?
D850  7F 03              C          jg      g_skp_4                  ; CX = 703H (black & white)
D852  B9 0302            C          mov     cx,302H                  ; CX = 302H (color)
D855                     C  g_skp_4:
                         C
                         C  ; CL = shift count to divide column by pixels per byte ...
                         C  ;      3 for b&w (divide by 8), 2 for color (divide by 4)
                         C  ; CH = remainder's mask during division (7 for b&w , 3 for color)
                         C  ; DX = column
                         C  ; DI = address to column 0 of requested row
                         C
D855  22 EA              C          and     ch,dl                    ; get division's remainder in CH
D857  D3 EA              C          shr     dx,cl                    ; perform division
                         C
                         C  ; DX = quotient = column's byte offset from start of row
                         C  ; CH = remainder (= pixel's offset into byte)
                         C
D859  03 FA              C          add     di,dx                    ; DI = pixel's byte address
                         C
                         C  ;       NOW:
                         C  ;            DI = address of byte containing the pixel
                         C  ;            CH = pixel offset into byte (remainder)
                         C  ;            CL = 3 (black & white) or 2 (color)
                         C
D85B  80 F9 03           C          cmp     cl,3                     ; black & white?
D85E  74 02              C          je      g_bitmask                ; jump if yes
D860  D0 E5              C          shl     ch,1                     ; color: convert to bit offset
                         C
D862                     C  g_bitmask:
D862  86 CD              C          xchg    cl,ch                    ; load CL, preserve "mode"
D864  B4 80              C          mov     ah,80H                   ; set high bit in byte
D866  D2 EC              C          shr     ah,cl                    ; mask pixel's leftmost bit
D868  80 FD 03           C          cmp     ch,3                     ; black & white?
D86B  74 06              C          je      g_skp_5                  ; jump if yes
D86D  8A C4              C          mov     al,ah                    ; color: create 2-bit mask
D86F  D0 E8              C          shr     al,1
D871  0A E0              C          or      ah,al
                         C
D873                     C  g_skp_5:                                 ; DI = byte address, AH = pixel mask,
D873  C3                 C          ret                              ; CL = bit offset: pixel's leftmost bit
                         C
D874                     C  g_addr  endp
                         C
                         C  page
                         C  ;===============================================================================
                         C  ;
                         C  ; Scroll Up In Graphics Mode
                         C  ;
                         C  ;       Scroll up the number of lines specified within the specified screen
                         C  ;       area (window).
                         C  ;
                         C  ;       Input:
                         C  ;            AL =    number of lines to be scrolled up ( zero
                         C  ;                    means clear the window)
                         C  ;            BH =    fill pattern to be used
                         C  ;            CH,CL = upper left corner of window in which to scroll
                         C  ;            DH,DL = lower right corner of window in which to scroll
                         C  ;            DS =    data segment
                         C  ;            ES =    graphics refresh ram segment
                         C  ;
                         C  ;       Saved:  BX, CX, DX (Video dispatcher saves the rest)
                         C  ;
                         C  ;-------------------------------------------------------------------------------
                         C
D874                     C  grf_graphics_up proc    near
                         C
D874  53                 C          push    bx                       ; save
D875  51                 C          push    cx                       ;       the
D876  52                 C          push    dx                       ;           registers
                         C
                         C  ;       old                              ;dir. flag = increment (from v_scrl_up)
D877  BD 0050            C          mov     bp,80                    ;offset to next scanline (CLD => +80)
                         C
D87A  50                 C          push    ax                       ;save mode, # rows to scroll
D87B  8B C1              C          mov     ax,cx                    ;compute address of window's
D87D  E8 DB5F R          C          call    g_curs_off               ;   upper left corner
D880  8B F8              C          mov     di,ax                    ;save in DI for string instructions
                         C
D882  06                 C          push    es                       ;set DS to video ram for string inst.
D883  1F                 C          pop     ds
                         C
D884  2B D1              C          sub     dx,cx                    ;compute window's dimensions
D886  81 C2 0101         C          add     dx,101H                  ;   DH = height,  DL = width
                         C
D88A  58                 C          pop     ax                       ;AH = mode,  AL = # rows to scroll
                         C
D88B  B3 02              C          mov     bl,2                     ;# interlace areas = 2 for modes 4,5,6
D88D  8A CB              C          mov     cl,bl                    ;# scanlines per i.a. = 4 for modes 4-72
```

```
D88F  80 FC 40          C           cmp     ah,64
D892  72 06             C           jb      g_tst_mod               ;jump if mode = 4,5,6
D894  B3 04             C           mov     bl,4                    ;# interlace areas = 4 for modes 64 & 72
D896  74 02             C           je      g_tst_mod               ;jump if mode = 64
D898  D0 F9             C           sar     cl,1                    ;# scanlines per i.a. = 2 for mode 72
                        C
D89A                    C g_tst_mod:
D89A  D2 E0             C           sal     al,cl                   ;convert number of rows to number of
D89C  D2 E6             C           sal     dh,cl                   ;   scanlines per interlace area
D89E  8B C8             C           mov     cx,ax                   ;CH = mode, CL = # scanlines to scroll
D8A0  80 FD 06          C           cmp     ch,6                    ;are we in a medium resolution mode ?
D8A3  7D 04             C           jge     g_set_up                ;jump if no
D8A5  D1 E7             C           sal     di,1                    ;double number of bytes per character
D8A7  D0 E2             C           sal     dl,1
                        C
D8A9                    C g_set_up:         ;get address of lines to scroll in refresh ram memory
D8A9  81 E1 00FF        C           and     cx,00FFH                ;CX = # of scanlines to scroll per i.a.
D8AD  74 3C             C           jz      g_filler                ;if zero, go fill all of window
                        C
D8AF  8B C1             C           mov     ax,cx                   ;make DH = # scanlines to fill per i.a.
D8B1  8A E9             C           mov     ch,cl                   ;      AX = # scanlines to scroll *  *
D8B3  86 F5             C           xchg    dh,ch                   ;      CH = # scanlines in window *  *
                        C
D8B5  8B F7             C           mov     si,di                   ;compute address of scanline to be
D8B7  B1 04             C           mov     cl,4                    ;   scrolled to top of window:
D8B9  D3 E0             C           sal     ax,cl                   ;   <window's address> +
D8BB  03 F0             C           add     si,ax                   ;   (<# scanlines to scroll per i.a.> *
D8BD  D1 E0             C           sal     ax,1                    ;    <# bytes per scanline = 80>)
D8BF  D1 E0             C           sal     ax,1
D8C1  03 F0             C           add     si,ax
                        C
D8C3  8A E5             C           mov     ah,ch                   ;compute # of scanlines per i.a.
D8C5  2A E6             C           sub     ah,dh                   ;   to be moved
                        C
D8C7  33 C9             C           xor     cx,cx                   ;CH = 0 for REP counter
                        C
                        C ;         jmp short g_scroller            ;go scroll up and fill
                        C
D8C9                    C grf_graphics_up endp
                        C
                        C page
                        C ;===============================================================================
                        C ;
                        C ;       Scroll rows in graphics refresh memory
                        C ;
                        C ;       Input:
                        C ;               AH =    Number of scanlines per interlace area to be moved
                        C ;               BL =    Number of interlace areas
                        C ;               CH =    0
                        C ;               DI =    Destination scanline address of first byte to fill
                        C ;               SI =    Source scanline address of first byte to fill
                        C ;               DL =    Number of bytes to fill in each scanline
                        C ;               BP =    offset to next scanline (+/- 80)
                        C ;
                        C ;       Output:
                        C ;               DI =    address of first byte to be filled
                        C ;               AH =    0
                        C ;
                        C ;       BL,BH,CH,DL,DH,BP preserved
                        C ;
                        C ;-------------------------------------------------------------------------------
                        C
D8C9                    C g_scroller      proc    near
                        C
D8C9  56                C           push    si                      ;save original source
D8CA  57                C           push    di                      ;  and destination addresses
D8CB  53                C           push    bx                      ;save interlace areas count (BL)
                        C
D8CC                    C g_m_area:       ;Move Interlace Areas Loop
D8CC  57                C           push    di                      ;save interlace area
D8CD  56                C           push    si                      ;   addresses
D8CE  8A CA             C           mov     cl,dl                   ;count of bytes to be moved
D8D0  F3/ A4            C           rep     movsb                   ;move the scanline
D8D2  5E                C           pop     si                      ;restore interlace area
D8D3  5F                C           pop     di                      ;   addresses
D8D4  81 C7 2000        C           add     di,2000H                ;next interlace area
D8D8  81 C6 2000        C           add     si,2000H                ;   addresses
D8DC  FE CB             C           dec     bl                      ;loop to move one scanline in
D8DE  75 EC             C           jnz     g_m_area                ;   each interlace area
                        C
D8E0  5B                C           pop     bx                      ;restore interlace areas count (BL)
D8E1  5F                C           pop     di                      ;restore original source
D8E2  5E                C           pop     si                      ;   and destination addresses
D8E3  03 FD             C           add     di,bp                   ;address next scanline in
D8E5  03 F5             C           add     si,bp                   ;   each interlace area
D8E7  FE CC             C           dec     ah                      ;loop to move "all" scanlines in
D8E9  75 DE             C           jnz     g_scroller              ;   each interlace area
                        C
                        C ;         jmp short g_filler              ;now fill in the gap
                        C
D8EB                    C g_scroller      endp
                        C
                        C page
                        C ;===============================================================================
                        C ;
                        C ; Fill rows in graphics refresh memory with the fill pattern
                        C ;
                        C ;
                        C ;       Input:
                        C ;               BL =    Number of interlace areas
```

```
                        C  ;               BH =    Fill pattern to be used
                        C  ;               CH =    0
                        C  ;               DL =    Number of bytes to fill in each scanline
                        C  ;               DH =    Number of scanlines to fill in each interlace area
                        C  ;               DI =    Destination scanline address of first byte to fill
                        C  ;               BP =    offset to next scanline (+/- 80)
                        C  ;
                        C  ;       Output:
                        C  ;               AL =    fill pattern
                        C  ;               AH =    0
                        C  ;
                        C  ;-------------------------------------------------------------------------------
                        C
D8EB                    C  g_filler        proc    near
                        C
D8EB  8A C7             C          mov     al,bh                   ;AL = fill pattern for STOSB instruction
                        C
D8ED                    C  g_f_i_lp:       ;Fill Interlace Areas Loop
D8ED  57                C          push    di                      ;save interlace area address
D8EE  52                C          push    dx                      ;save scanlines count (DH)
                        C
D8EF                    C  g_f_s_lp:       ;Fill Scanlines Loop
D8EF  57                C          push    di                      ;save scanline address
D8F0  8A CA             C          mov     cl,dl                   ;count of bytes to fill in scanline
D8F2  F3/ AA            C          rep     stosb                   ;fill scanline
D8F4  5F                C          pop     di                      ;restore scanline address
D8F5  03 FD             C          add     di,bp                   ;next scanline in interlace area
D8F7  FE CE             C          dec     dh                      ;fill an interlace area
D8F9  75 F4             C          jnz     g_f_s_lp
                        C
D8FB  5A                C          pop     dx                      ;restore scanlines count (DH)
D8FC  5F                C          pop     di                      ;restore interlace area address
D8FD  81 C7 2000        C          add     di,2000H                ;next interlace area address
D901  FE CB             C          dec     bl                      ;fill next interlace area
D903  75 E8             C          jnz     g_f_i_lp
                        C
D905  5A                C          pop     dx                      ;restore
D906  59                C          pop     cx                      ;       the
D907  5B                C          pop     bx                      ;               registers
D908  C3                C          ret                             ;exit scroll routine
                        C
D909                    C  g_filler        endp
                        C
                        C  page
                        C  ;===============================================================================
                        C  ;
                        C  ; Scroll Down In Graphics Mode
                        C  ;
                        C  ;       Scroll down the number of lines specified within the specified screen
                        C  ;       area (window).
                        C  ;
                        C  ;       Input:
                        C  ;               AL     = number of lines to be scrolled up ( zero
                        C  ;                        means clear the window)
                        C  ;               BH     = fill pattern to be used
                        C  ;               CH,CL  = upper left corner of window in which to scroll
                        C  ;               DH,DL  = lower right corner of window in which to scroll
                        C  ;               DS     = data segment
                        C  ;               ES     = graphics refresh ram segment
                        C  ;
                        C  ;       Saved:  BX, CX, DX (Video dispatcher saves the rest)
                        C  ;
                        C  ;-------------------------------------------------------------------------------
                        C
D909                    C  grf_graphics_down       proc near
                        C
D909  53                C          push    bx                      ; save
D90A  51                C          push    cx                      ;       the
D90B  52                C          push    dx                      ;               registers
                        C
                        C  ;       std                             ;dir. flag = decrement (from v_scrl_dn)
D90C  BD FFB0           C          mov     bp,-80                  ;offset to next scanline (STD => -80)
                        C
D90F  50                C          push    ax                      ;save mode, # rows to scroll
D910  8B C2             C          mov     ax,dx                   ;compute address of window's
D912  E8 DB5F R         C          call    g_curs_off              ;   lower right corner
D915  8B F8             C          mov     di,ax                   ;save in DI for string instructions
                        C
D917  06                C          push    es                      ;set DS to video ram for string inst.
D918  1F                C          pop     ds
                        C
D919  2B D1             C          sub     dx,cx                   ;compute window's dimensions
D91B  81 C2 0101        C          add     dx,101H                 ;   DH = height,  DL = width
                        C
D91F  58                C          pop     ax                      ;AH = mode,  AL = # rows to scroll
                        C
D920  B3 02             C          mov     bl,2                    ;# interlace areas = 2 for modes 4,5,6
D922  8A CB             C          mov     cl,bl                   ;# scanlines per i.a. = 4 for modes 72
D924  80 FC 40          C          cmp     ah,64
D927  72 06             C          jb      g_cmp_mod               ;jump if mode = 4,5,6
D929  B3 04             C          mov     bl,4                    ;# interlace areas = 4 for modes 64 & 72
D92B  74 02             C          je      g_cmp_mod               ;jump if mode = 64
D92D  D0 F9             C          sar     cl,1                    ;# scanlines per i.a. = 2 for mode 72
                        C
D92F                    C  g_cmp_mod:
D92F  D2 E0             C          sal     al,cl                   ;convert number of rows to number of
D931  D2 E6             C          sal     dh,cl                   ;   scanlines per interlace area
D933  80 FC 06          C          cmp     ah,6                    ;are we in a medium resolution mode ?
D936  7D 05             C          jge     g_setdown               ;jump if no
D938  D1 E7             C          sal     di,1                    ;double number of bytes per character
```

```
D93A  D0 E2                C           sal     dl,1
D93C  47                   C           inc     di                      ;address last byte in bottom row
                           C
D93D                       C   g_setdown:     ;get address of lines to scroll in refresh RAM memory
D93D  BE 0050              C           mov     si,80                   ;address bottom scanline in i.a.
D940  D3 E6                C           sal     si,cl
D942  83 EE 50             C           sub     si,80
D945  03 FE                C           add     di,si
D947  8B C8                C           mov     cx,ax                   ;CH = mode, CL = # scanlines to scroll
D949  81 E1 00FF           C           and     cx,00FFH                ;CL = # of scanlines to scroll per i.a.
D94D  74 9C                C           jz      g_filler                ;if zero, go fill all of window
                           C
D94F  8B C1                C           mov     ax,cx                   ;make DH = # scanlines to fill per i.a.
D951  8A E9                C           mov     ch,cl                   ;     AX = # scanlines to scroll "  "
D953  86 F5                C           xchg    dh,ch                   ;     CH = # scanlines in window "  "
                           C
D955  8B F7                C           mov     si,di                   ;compute address of scanline to be
D957  B1 04                C           mov     cl,4                    ;   scrolled to top of window:
D959  D3 E0                C           sal     ax,cl                   ;   <window's address> -
D95B  2B F0                C           sub     si,ax                   ;   ((# scanlines to scroll per i.a.> *
D95D  D1 E0                C           sal     ax,1                    ;     (# bytes per scanline = 80))
D95F  D1 E0                C           sal     ax,1
D961  2B F0                C           sub     si,ax
                           C
D963  8A E5                C           mov     ah,ch                   ;compute # of scanlines per i.a.
D965  2A E6                C           sub     ah,dh                   ;   to be moved
                           C
D967  33 C9                C           xor     cx,cx                   ;CH = 0 for REP counter
                           C
D969  E9 D8C9 R            C           jmp     g_scroller              ;go scroll down and fill
                           C
D96C                       C   grf_graphics_down       endp
                           C
                           C   page
                           C   ;===========================================================================
                           C   ;
                           C   ;       graphics_read - read the character at the current cursor
                           C   ;                       position on the screen, or zero.
                           C   ;       Input:  none
                           C   ;
                           C   ;       Output: AL =    character at the current cursor position or zero
                           C   ;
                           C   ;       Saved:  BX, CX, DX (video dispatcher saves the rest)
                           C   ;
                           C   ;---------------------------------------------------------------------------
                           C
D96C                       C   grf_graphics_read       proc    near
                           C
D96C  53                   C           push    bx                      ; save
D96D  51                   C           push    cx                      ;       the
D96E  52                   C           push    dx                      ;           registers
                           C
D96F  8A DC                C           mov     bl,ah                   ;BL saves v_mode
D971  A1 0050 R            C           mov     ax,word ptr ds:[v_curpos]
D974  E8 DB5F R            C           call    g_curs_off              ;get address of cursor position
D977  8B F0                C           mov     si,ax                   ;save as a pointer for later
                           C
D979  8A C3                C           mov     al,bl                   ;AL saves v_mode
D97B  B9 0004              C           mov     cx,4                    ;# scanlines per i.a. = 4 for modes 72
D97E  33 DB                C           xor     bx,bx                   ;make BX=0 for modes 4,5,6
D980  3C 40                C           cmp     al,64
D982  72 08                C           jb      g_lds_r                 ;jump if mode = 4,5,6
D984  B3 04                C           mov     bl,4                    ;make BX=4 for mode 64
D986  74 04                C           je      g_lds_r                 ;jump if mode = 64
D988  D1 F9                C           sar     cx,1                    ;# scanlines per i.a. = 2 for mode 72
D98A  33 DB                C           xor     bx,bx                   ;make BX=0 for mode 72
                           C
D98C                       C   g_lds_r:                ;(BX= font pointer offset in master table)
D98C  C5 3E 0084 R         C           lds     di,dword ptr ds:[master_tbl_ptr] ;get pointer to master table
D990  C5 79 06             C           lds     di,dword ptr ds:[di+6][bx] ;get pointer to font's 1st 128 chars
D993  1E                   C           push    ds                      ;XCHG DS,ES
D994  06                   C           push    es
D995  1F                   C           pop     ds                      ;DS:SI -> grafix ram read location  ???
D996  07                   C           pop     es                      ;ES:DI -> font's 1st 128 characters
                           C
D997  D1 E3                C           sal     bx,1                    ;make BX=8 for modes 4,5,6,72
D999  83 C3 08             C           add     bx,8                    ;     BX=16 for mode 64
                           C                   ;(BX= number of font bytes per character)
                           C
D99C  2B E3                C           sub     sp,bx                   ;get stack space for font bytes
D99E  8B EC                C           mov     bp,sp                   ;save pointer to stack space
D9A0  BA 0002              C           mov     dx,2                    ;number of interlace areas (modes 4,5,6)
D9A3  3C 06                C           cmp     al,6                    ;check graphics mode
D9A5  7C 33                C           jl      g_rd_med                ;jump if in medium resolution mode (4,5)
D9A7  74 03                C           je      g_rdloop                ;jump if in 640x200 resolution mode (6)
                           C
                           C   ; we're in super resolution (640x400) mode (64 & 72)
D9A9  BA 0004              C           mov     dx,4                    ;number of interlace areas
                           C
D9AC                       C   g_rdloop:               ;Read Scanlines Loop for both 640x200 and 640x400
D9AC  51                   C           push    cx                      ;save # of scanlines per interlace area
D9AD  56                   C           push    si                      ;save current addr. in interlace area #1
D9AE  8B CA                C           mov     cx,dx                   ;init. counter: # of interlace areas
                           C
D9B0                       C   g_rd_ia:                ;Read Interlace Area Loop
D9B0  8A 04                C           mov     al,[si]                 ;get byte from grafix ram
D9B2  88 46 00             C           mov     [bp],al                 ;save on reserved stack
D9B5  45                   C           inc     bp                      ;bump reserved stack address
D9B6  81 C6 2000           C           add     si,2000H                ;address next interlace area
D9BA  E2 F4                C           loop    g_rd_ia
```

```
                                C
D9BC  5E                        C          pop     si                      ;restore interlace area #1 address
D9BD  83 C6 50                  C          add     si,80                   ;address next scanline in each i.a.
D9C0  59                        C          pop     cx                      ;restore # of scanlines counter
D9C1  E2 E9                     C          loop    g_rdloop
                                C
D9C3  83 FA 04                  C          cmp     dx,4                    ;are we in mode 64 or 72 (640x400)?
D9C6  75 47                     C          jne     g_matchb                ;jump if no (reverse video not allowed)
D9C8  8B F5                     C          mov     si,bp                   ;point to first byte in stack save area
D9CA  2B F3                     C          sub     si,bx
D9CC  F6 04 80                  C          test    byte ptr [si],80H       ;is upper left bit of char = 0 or 1 ?
D9CF  74 3E                     C          jz      g_matchb                ;jump if 0 (not reversed video)
D9D1  8B CB                     C          mov     cx,bx                   ;number of char bytes counter
D9D3                            C g_unreverse_video_loop:
D9D3  F6 14                     C          not     byte ptr [si]           ;reverse the reversed byte for matching
D9D5  46                        C          inc     si                      ;address next char byte
D9D6  E2 FB                     C          loop    g_unreverse_video_loop
D9D8  EB 35                     C          jmp short g_matchb              ;now find the char in the font table
                                C
D9DA                            C g_rd_med:             ;Read Medium Resolution
D9DA  D1 E6                     C          sal     si,1                    ;double graf ram pointer (2 bytes/char)
                                C
D9DC                            C g_medget:             ;Get font bytes in medium resolution (320 X 200) mode
D9DC  51                        C          push    cx                      ;save # scanlines per i.a. counter
D9DD  56                        C          push    si                      ;save current scanline address
D9DE  B9 0002                   C          mov     cx,2                    ;init. # interlace areas counter
D9E1                            C g_med_ia:
D9E1  51                        C          push    cx                      ;save i.a. counter
D9E2  8B 04                     C          mov     ax,[si]                 ;get 2 bytes of 1 char from video memory
D9E4  86 E0                     C          xchg    ah,al                   ;order them logically
                                C
D9E6  F7 D0                     C          not     ax                      ;map background pixels to 0,
D9E8  8B D0                     C          mov     dx,ax                   ;    foreground pixels to 1
D9EA  D1 E2                     C          shl     dx,1
D9EC  23 D0                     C          and     dx,ax
D9EE  F7 D2                     C          not     dx
                                C
D9F0  32 C0                     C          xor     al,al                   ;clear result accumulator
D9F2  B9 0008                   C          mov     cx,8                    ;prepare to process 8 bits
D9F5                            C g_med_bit:
D9F5  D1 EA                     C          shr     dx,1                    ;ignore unused bit
D9F7  D1 EA                     C          shr     dx,1                    ;load carry with mapped pixel value
D9F9  D0 D8                     C          rcr     al,1                    ;rotate it into result accumulator
D9FB  E2 F8                     C          loop    g_med_bit               ;process next bit of character
                                C
D9FD  88 46 00                  C          mov     [bp],al                 ;save font byte on reserved stack
DA00  45                        C          inc     bp                      ;bump reserved stack pointer
DA01  81 C6 2000                C          add     si,2000H                ;address next interlace area
DA05  59                        C          pop     cx                      ;restore i.a. counter
DA06  E2 D9                     C          loop    g_med_ia                ;process next i.a. of character
                                C
DA08  5E                        C          pop     si                      ;restore scanline address
DA09  83 C6 50                  C          add     si,80                   ;address next scanline
DA0C  59                        C          pop     cx                      ;restore scanlines counter
DA0D  E2 CD                     C          loop    g_medget                ;process next scanline of character
                                C
DA0F                            C g_matchb:             ;Match Font Byte: find character
                                C
DA0F  2B EB                     C          sub     bp,bx                   ;point to first byte in stack save area
DA11  8B F5                     C          mov     si,bp                   ;pointer to font bytes from graf ram
DA13  32 C0                     C          xor     al,al                   ;index to font bytes (start with 0)
DA15                            C g_f_cont:             ; Get Font Byte Match Control
DA15  16                        C          push    ss                      ;setup string compare registers
DA16  1F                        C          pop     ds                      ;DS:SI -> stack area w/grafix ram bytes
DA17  B9 0080                   C          mov     cx,128                  ;loop control = 1st 128 ascii chars.
                                C
DA1A                            C g_f_mach:             ;Get Font Byte Match Loop
DA1A  51                        C          push    cx                      ;save loop counter
DA1B  8B CB                     C          mov     cx,bx                   ;counter for string compare
DA1D  57                        C          push    di                      ;save font pointer
DA1E  56                        C          push    si                      ;save pointer to stack save area
DA1F  F3/ A6                    C          repe    cmpsb                   ;screen bytes match font bytes ?
DA21  5E                        C          pop     si                      ;retrieve pointer to stack save area
DA22  5F                        C          pop     di                      ;retrieve pointer to fomat byte table
DA23  59                        C          pop     cx                      ;restore loop counter
DA24  74 2D                     C          je      g_f_exit                ;if match go to exit code
DA26  03 FB                     C          add     di,bx                   ;address next font in table
DA28  FE C0                     C          inc     al                      ;bump ascii index
DA2A  E2 EE                     C          loop    g_f_mach                ;go back if more chars to search for
                                C
                                C ; no match in first 128 ascii character set - look for user's second set
                                C
DA2C  0A C0                     C          or      al,al                   ;have we scanned both 128 char 1/2s ?
DA2E  74 23                     C          jz      g_f_exit                ;jump if yes
DA30  83 FB 10                  C          cmp     bx,16                   ;are we in mode 64 ?
DA33  74 08                     C          je      g_8x16_2                ;jump if yes
                                C
DA35  8E D9                     C          mov     ds,cx                   ;move zero to segment register
                                C          assume  ds:abs0
DA37  C4 3E 007C R              C          les     di,dword ptr ds:[int1Floc] ;get pointer to 2nd half of 8x8 font
                                C          assume  ds:data
DA3B  EB 0C                     C          jmp short g_test_addr           ;see if font is really there
                                C
DA3D                            C g_8x16_2:                             ;get 2nd half of 8x16 font
DA3D  2E: 8E 1E E5F2 R          C          mov     ds,word ptr cs:[set_ds_word]    ;set DS to data segment
DA42  C5 3E 0084 R              C          lds     di,dword ptr ds:[master_tbl_ptr] ;get pointer to master table
DA46  C4 7D 0E                  C          les     di,dword ptr ds:[di+14] ;get pointer to 2nd half of 8x16 font
                                C
DA49                            C g_test_addr:
DA49  8C C0                     C          mov     ax,es                   ;check if font table is set up
```

```
DA4B  0B C7                C          or      ax,di                   ;if zeros, then no user font table
DA4D  74 04                C          jz      g_f_exit                ;no table, just go to exit
DA4F  B0 80                C          mov     al,128                  ;offset to 2nd 1/2 of ascii set
DA51  EB C2                C          jmp short g_f_cont              ;go back to try rest of ascii set
                           C
DA53                       C g_f_exit:                ;either the character is found, or al = 0
DA53  03 E3                C          add     sp,bx                   ;restore stack pointer
DA55  5A                   C          pop     dx                      ;restore
DA56  59                   C          pop     cx                      ;       the
DA57  5B                   C          pop     bx                      ;           registers
DA58  C3                   C          ret
                           C
DA59                       C grf_graphics_read        endp
                           C
                           C page
                           C ;===========================================================================
                           C ;
                           C ;       graphics_write - write a character to the screen
                           C ;
                           C ;       Input:  AL =    character to write
                           C ;               BL =    Foreground color attribute
                           C ;                       bit 7 = 1: xor character with graphics ram
                           C ;               CX =    number of characters to write
                           C ;               DS =    data segment
                           C ;               ES =    graphics ram segment
                           C ;
                           C ;       Saved:  BX, CX, DX (video dispatcher saves the rest)
                           C ;
                           C ;---------------------------------------------------------------------------
                           C
DA59                       C grf_graphics_write       proc    near
                           C
DA59  53                   C          push    bx                      ;save
DA5A  51                   C          push    cx                      ;    the
DA5B  52                   C          push    dx                      ;        registers
                           C
DA5C  8B D0                C          mov     dx,ax                   ;DH= ort mode, DL= char to write
                           C ; locate beginning of character in graphics ram
DA5E  A1 0050 R            C          mov     ax,word ptr ds:[v_curpos]
DA61  E8 DB5F R            C          call    g_curs_off              ;get address of cursor position
DA64  8B F8                C          mov     di,ax                   ; pointer to graphics location
                           C ; determine if character is from 1st or 2nd half of table
DA66  80 FA 80             C          cmp     dl,128                  ;is it in first 1/2 of ASCII set ?
DA69  72 26                C          jb      g_selfont               ;jump if in 1st 1/2 (0 -> 127)
                           C
                           C ; character (128 -> 255) is in 2nd 1/2 of font table
                           C
DA6B  80 EA 80             C          sub     dl,128                  ;make zero origin for font table lookup
DA6E  80 FE 40             C          cmp     dh,64                   ;are we in mode 64 ?
DA71  75 09                C          jne     g_8x8_2                 ;jump if no
                           C
DA73  C5 36 0084 R         C          lds     si,dword ptr ds:[master_tbl_ptr] ;get pointer to master table
DA77  C5 74 0E             C          lds     si,dword ptr ds:[si+14] ;get pointer to 2nd half of 8x16 font
                           C
DA7A  EB 08                C          jmp short g_addr_test           ;see if font table is really there
                           C
DA7C                       C g_8x8_2:                         ;get 2nd half of 8x8 font table
DA7C  33 F6                C          xor     si,si                   ;move zero to segment register
DA7E  8E DE                C          mov     ds,si
                           C          assume  ds:abs0
DA80  C5 36 007C R         C          lds     si,dword ptr ds:[int1Floc] ;get pointer to 2nd half of 8x8 font
                           C          assume  ds:data
                           C
DA84                       C g_addr_test:
DA84  8C D8                C          mov     ax,ds                   ;check if font table is set up
DA86  0B C6                C          or      ax,si                   ;if zeros, then no 2nd half of table
DA88  75 1D                C          jnz     g_detmode               ;continue if font table is present
DA8A  32 D2                C          xor     dl,dl                   ;substitute null character
DA8C  2E: 8E 1E E5F2 R     C          mov     ds,word ptr cs:[set_ds_word]    ;restore data segment register
                           C ;        jmp short g_selfont             ;and continue in 1st half of font table
                           C
                           C ; character (0 -> 127) is in 1st 1/2 of font table
                           C
DA91                       C g_selfont:
DA91  53                   C          push    bx                      ;preserve register
                           C
DA92  33 DB                C          xor     bx,bx                   ;make BX=0 for modes 4,5,6
DA94  80 FE 40             C          cmp     dh,64
DA97  72 06                C          jb      g_lds_w                 ;jump if mode = 4,5,6
DA99  B3 04                C          mov     bl,4                    ;make BX=4 for mode 64
DA9B  74 02                C          je      g_lds_w                 ;jump if mode = 64
DA9D  33 DB                C          xor     bx,bx                   ;make BX=0 for mode 72
                           C
DA9F                       C g_lds_w:                 ;(BX= font pointer offset in master table)
DA9F  C5 36 0084 R         C          lds     si,dword ptr ds:[master_tbl_ptr] ;get pointer to master table
DAA3  C5 70 06             C          lds     si,dword ptr ds:[si+6][bx] ;get pointer to font's 1st 128 chars
                           C
DAA6  5B                   C          pop     bx                      ;restore register
                           C
DAA7                       C g_detmode:                       ;determine graphics mode
DAA7  51                   C          push    cx
DAA8  33 C0                C          xor     ax,ax                   ;get ascii code in AX
DAAA  8A C2                C          mov     al,dl
DAAC  B1 03                C          mov     cl,3                    ; to multiply by
DAAE  D3 E0                C          sal     ax,cl                   ;    8 (font bytes per character)
DAB0  59                   C          pop     cx
DAB1  03 F0                C          add     si,ax                   ;and add to address of font table
                           C
DAB3  B2 04                C          mov     dl,4                    ;# scanlines per i.a. (modes 4,5,6,64)
                           C
```

```
DAB5  80 FE 06          C       cmp     dh,6                    ;which resolution are we using?
DAB8  7C 49             C       jl      g_med_wr                ;jump if medium resolution (modes 4 & 5)
DABA  74 0F             C       je      g_hi_wr                 ;jump if 640x200 resolution (mode 6)
                        C
                        C                                       ;we're in 640x400 resolution
DABC  80 FE 48          C       cmp     dh,72                   ;mode 72?
DABF  B6 04             C       mov     dh,4                    ;# interlace areas = 4 for modes 64 & 72
DAC1  75 04             C       jne     g_super_wr
                        C
DAC3                    C g_tinytext:                           ;640x400 resolution (mode 72)
DAC3  B2 02             C       mov     dl,2                    ;# scanlines per i.a. = 2 for mode 72
DAC5  EB 09             C       jmp short g_repchar
                        C
DAC7                    C g_super_wr:                           ;640x400 resolution (mode 64)
DAC7  03 F0             C       add     si,ax                   ;multiply ascii code by 16 bytes/char
DAC9  EB 05             C       jmp short g_repchar
                        C
DACB                    C g_hi_wr:                              ;Hi-resolution Character Write
DACB  B6 02             C       mov     dh,2                    ;interlace areas count (even/odd)
DACD  80 CB 01          C       or      bl,1                    ;mode 6 doesn't allow reverse video
                        C
DAD0                    C g_repchar:                            ;Repeat Character Loop
DAD0  51                C       push    cx                      ;save character repeat count
DAD1  56                C       push    si                      ;save source address (font table)
DAD2  57                C       push    di                      ;save destination addr. (grafix ram)
DAD3  33 C9             C       xor     cx,cx                   ;prepare loop counter
DAD5  8A CA             C       mov     cl,dl                   ;scanlines per interlace area counter
                        C
DAD7                    C g_linelp:                             ;Scanline Loop
DAD7  51                C       push    cx                      ;save scanlines per i.a. counter
DAD8  57                C       push    di                      ;save interlace area #1 address
DAD9  8A CE             C       mov     cl,dh                   ;init. interlace areas counter (2 or 4)
                        C
DADB                    C g_i_a_lp:                             ;Interlace Area Loop
DADB  AC                C       lodsb                           ;get byte from font table
DADC  F6 C3 01          C       test    bl,1                    ;reverse video?
DADF  75 02             C       jnz     g_t_xor                 ;jump if no
DAE1  F6 D0             C       not     al                      ;reverse video
DAE3                    C g_t_xor:                              ;Test For XOR
DAE3  0A DB             C       or      bl,bl                   ;XOR the char. with grafix ram?
DAE5  79 03             C       jns     g_w_byte                ;jump if no
DAE7  26: 32 05         C       xor     al,es:[di]              ;XOR with grafix ram
DAEA                    C g_w_byte:                             ;Write Byte
DAEA  26: 88 05         C       mov     es:[di],al              ;write byte in grafix ram
DAED  81 C7 2000        C       add     di,2000H                ;address next interlace area
DAF1  E2 E8             C       loop    g_i_a_lp
                        C
DAF3  5F                C       pop     di                      ;restore interlace area #1 address
DAF4  83 C7 50          C       add     di,80                   ;address next scanline in each i.a.
DAF7  59                C       pop     cx                      ;restore scanlines per i.a. counter
DAF8  E2 DD             C       loop    g_linelp
                        C
DAFA  5F                C       pop     di                      ;restore char's grafix ram address
DAFB  47                C       inc     di                      ;address next character in grafix ram
DAFC  5E                C       pop     si                      ;restore char's font table address
DAFD  59                C       pop     cx                      ;restore character repeat count
DAFE  E2 D0             C       loop    g_repchar               ;repeat the character
DB00  EB 55 90          C       jmp     g_return                ;exit
                        C
DB03                    C g_med_wr:                             ;Medium Resolution Character Write
DB03  D1 E7             C       sal     di,1                    ;double graf ram pointer (2 bytes/char)
DB05  8A D3             C       mov     dl,bl                   ;DL saves XOR bit input param (bit 7)
DB07  81 E3 0003        C       and     bx,0003H                ;BX= foreground color (& table offset)
DB0B  2E: 8A 9F DB5B R  C       mov     bl,cs:g_color_table[bx] ;propagate color through byte
DB10  8A FB             C       mov     bh,bl                   ;propagate color through word
DB12  8B EB             C       mov     bp,bx                   ;BP saves word of color masks
                        C
DB14                    C g_char_lp:                            ;Repeat Character Loop
DB14  51                C       push    cx                      ;save character repeat counter
DB15  56                C       push    si                      ;save source address (font table)
DB16  57                C       push    di                      ;save destination addr. (grafix ram)
DB17  B9 0004           C       mov     cx,4                    ;init. scanlines per i.a. counter
                        C
DB1A                    C g_scan_lp:                            ;Scanline Loop
DB1A  51                C       push    cx                      ;save scanlines per i.a. counter
DB1B  57                C       push    di                      ;save interlace area #1 address
DB1C  B9 0002           C       mov     cx,2                    ;init. interlace areas counter
                        C
DB1F                    C g_ia_lp:                              ;Interlace Area Loop
DB1F  51                C       push    cx                      ;save i.a. counter
DB20  AC                C       lodsb                           ;get


DB1F  51                C       push    cx                      ;save i.a. counter
DB20  AC                C       lodsb                           ;get a byte from the font table
DB21  8A E0             C       mov     ah,al                   ;copy it
DB23  B9 0008           C       mov     cx,8                    ;init. loop counter (8 bits/byte)
                        C
DB26                    C g_exp_byt:                            ;Expand Byte Loop
DB26  D0 EC             C       shr     ah,1                    ;load carry with font byte bit
DB28  D1 DB             C       rcr     bx,1                    ;rotate it into expansion accumulator
DB2A  D0 E8             C       shr     al,1                    ;load carry with same bit as before
DB2C  D1 DB             C       rcr     bx,1                    ;double the bit
DB2E  E2 F6             C       loop    g_exp_byt               ;expand font byte bits
                        C
DB30  23 DD             C       and     bx,bp                   ;color pixels with foreground color
DB32  86 FB             C       xchg    bh,bl                   ;reorder the bytes for grafix ram
DB34  0A D2             C       or      dl,dl                   ;is the XOR bit set ?
DB36  79 03             C       jns     g_med_store             ;jump if no
DB38  26: 33 1D         C       xor     bx,es:[di]              ;XOR with grafix ram
```

```
DB3B                    C  g_mod_store:
DB3B  26: 89 1D         C          mov     es:[di],bx              ;update grafix ram
DB3E  81 C7 2000        C          add     di,2000H                ;address next interlace area
DB42  59                C          pop     cx                      ;restore i.a. loop counter
DB43  E2 DA             C          loop    g_ia_lp                 ;next interlace area
                        C
DB45  5F                C          pop     di                      ;restore interlace area #1 address
DB46  83 C7 50          C          add     di,80                   ;address next scanline in each i.a.
DB49  59                C          pop     cx                      ;restore scanline loop counter
DB4A  E2 CE             C          loop    g_scan_lp               ;next scanline
                        C
DB4C  5F                C          pop     di                      ;restore char's grafix ram address
DB4D  47                C          inc     di                      ;address next character in grafix ram
DB4E  47                C          inc     di
DB4F  5E                C          pop     si                      ;restore char's font table address
DB50  59                C          pop     cx                      ;restore character repeat count
DB51  E2 C1             C          loop    g_char_lp               ;repeat the character
                        C
DB53  8A E3             C          mov     ah,bl                   ;return AX with last word written
DB55  8A C7             C          mov     al,bh
                        C
DB57                    C  g_return:                    ;Return from Write Char
DB57  5A                C          pop     dx                      ;restore
DB58  59                C          pop     cx                      ;       the
DB59  5B                C          pop     bx                      ;          registers
DB5A  C3                C          ret
                        C
DB5B                    C  g_color_table    label   byte    ;Table of foreground colors extended to byte
                        C
DB5B  00                C          db      00000000B               ;color 0        (bit pattern: 00)
DB5C  55                C          db      01010101B               ;color 1        (bit pattern: 10)
DB5D  AA                C          db      10101010B               ;color 2        (bit pattern: 01)
DB5E  FF                C          db      11111111B               ;color 3        (bit pattern: 11)
                        C
DB5F                    C  grf_graphics_write       endp
                        C
                        C  page
                        C  ;===============================================================================
                                C     ;===============================================================================
                        C  ;
                        C  ;       Get offset into graphics ram refresh memory which corresponds to
                        C  ;       the current cursor position (or any arbitrary character position).
                        C  ;
                        C  ;       Input:  AX  = current cursor position (AL = Column #, AH = Row #)
                        C  ;
                        C  ;       Output: AX =  offset into graphics ram
                        C  ;
                        C  ;-------------------------------------------------------------------------------
                        C
DB5F                    C  g_curs_off       proc    near
```

```
DB5F  51                      C          push    cx                          ;save work register
DB60  8A E8                   C          mov     ch,al                       ;hold column number
DB62  8A C4                   C          mov     al,ah                       ;row number to al
                              C
DB64  B1 01                   C          mov     cl,1                        ;mode 72 shift count (multiply * 2)
DB66  80 3E 0049 R 48         C          cmp     byte ptr ds:[v_mode],72
DB6B  74 02                   C          je      g_72                        ;jump if mode 72
DB6D  FE C1                   C          inc     cl                          ;mode ~72 shift count (multiply * 4)
DB6F                          C g_72:
DB6F  D2 E0                   C          shl     al,cl                       ;multiply row # by rows per byte
DB71  32 C9                   C          xor     cl,cl                       ;zero out the shift count
DB73  86 E9                   C          xchg    ch,cl                       ;move column number for add
DB75  F6 26 004A R            C          mul     byte ptr ds:[v_width]       ;multiply by bytes per columnn
DB79  03 C1                   C          add     ax,cx                       ;compute offset into refresh ram
DB7B  59                      C          pop     cx                          ;restore register
DB7C  C3                      C          ret                                 ;and return to caller
                              C
DB7D                          C g_curs_off      endp
                              C
DB7D                          C code    ends
                              C include pwrup1.asm
                              C
                              C
                              C ;===========================================================================
                              C ;       Filename:       pwrup1.src
                              C ;
                              C ;       This module includes CPU, ROM, 8253 p_dma p_timer, & 8237 p_dma
                              C ;       Controller tests.
                              C ;
                              C ;===========================================================================
                              C
DB7D                          C code    segment public 'ROM'
                              C         assume  cs:code, ds:nothing, es:nothing, ss:nothing
                              C
DB7D                          C p1_data1        proc    near
                              C
DB7D  90                      C even                                              ; word-align stack_rom
                              C
DB7E  DDB2 R                  C stack_rom       dw      i_rom                     ; return from i_cpu
DB80  DDB8 R                  C                 dw      i_rom_ret                 ; return from rom_checksum
DB82  DDC7 R                  C                 dw      i_dmat                    ; return from i_rom
DB84  DDD3 R                  C                 dw      i_dmat_ret                ; return from rto_chk
DB86  DDE2 R                  C                 dw      i_dmac                    ; return from i_dmat
DB88  DE51 R                  C                 dw      i_dmac_ret                ; return from memtst
DB8A  DE6B R                  C                 dw      i_pic                     ; return from i_dmac
                              C
DB8C  52 65 73 69 64 65       C banner_m        db      'Resident Diagnostics',CR,LF
      6E 74 20 44 69 61       C
      67 6E 6F 73 74 69       C
      63 73 0D 0A             C
DBA2  52 65 76 20 31 2E       C                 db      'Rev 1.0  May 1984',CR,LF,LF,NUL
      30 20 20 4D 61 79       C
      20 31 39 38 34 0D       C
      0A 0A 00                C
                              C
DBB7  0D 0A 50 72 69 6D       C bt_m            db      CR,LF,'Primary Boot-Strap...',CR,LF,NUL
      61 72 79 20 42 6F       C
      6F 74 2D 53 74 72       C
      61 70 2E 2E 2E 0D       C
      0A 00                   C
DBD1  50 72 69 6D 61 72       C bt_merr         db      'Primary Boot-Strap DISK READ ERROR.',CR,NUL
      79 20 42 6F 6F 74       C
      2D 53 74 72 61 70       C
      20 44 49 53 4B 20       C
      52 45 41 44 20 45       C
      52 52 4F 52 2E 0D       C
      00                      C
                              C ; This line must have same number of blanks as the preceding has characters:
DBF6  20 20 20 20 20 20       C bt_spaces       db      '                                   ',CR,NUL
      20 20 20 20 20 20       C
      20 20 20 20 20 20       C
      20 20 20 20 20 20       C
      20 20 20 20 20 20       C
      20 20 20 20 20 0D       C
      00                      C
                              C
DC1B  2A 20 49 6C 6C 65       C ill_m1          db      '* Illegal Interrupt No. ',NUL
      67 61 6C 20 49 6E       C
      74 65 72 72 75 70       C
      74 20 4E 6F 2E 20       C
      00                      C
DC34  68 20 61 74 20 00       C ill_m2          db                          'h at ',NUL
DC3A  20 2A 00                C ill_m3          db                                  ' *',NUL
                              C
DC3D  20 50 61 73 73 0D       C pass_m          db      ' Pass',CR,LF,NUL
      0A 00                   C
DC45  20 46 61 69 6C 00       C fail_m          db      ' Fail',NUL
                              C
DC4B  43 50 55 20 28 69       C i_cpu_m         db      'CPU (i8086) ',NUL      ; Pass/Fail
      38 30 38 36 29 20       C
      00                      C
DC58  52 4F 4D 20 4D 6F       C i_rom_m         db      'ROM Module  ',NUL      ; Pass/Fail
      64 75 6C 65 20 20       C
      00                      C
DC65  44 4D 41 20 54 69       C i_dmat_m        db      'DMA Timer   ',NUL      ; Pass/Fail
      6D 65 72 20 20 20       C
      00                      C
DC72  44 4D 41 20 43 6F       C i_dmac_m        db      'DMA Control ',NUL      ; Pass/Fail
      6E 74 72 6F 6C 20       C
      00                      C
```

```
DC7F  49 6E 74 65 72 72      C  i_pic_m          db      'Interrupts  ',NUL      ; Pass/Fail/Fail:Hx
      75 70 74 73 20 20      C
      00                     C
DC8C  56 69 64 65 6F 20      C  i_d_m            db      'Video Board ',NUL      ; Pass/Fail
      42 6F 61 72 64 20      C
      00                     C
DC99  4E 50 55 20 28 69      C  i_npu_m          db      'NPU (18087) ',NUL      ; Pass/Fail
      38 30 38 37 29 20      C
      00                     C
DCA6  52 54 20 43 6C 6F      C  i_rtc_m          db      'RT Clock    ',NUL      ; Pass/Fail/Fail:LO,HI,NR
      63 6B 20 20 20 20      C
      00                     C
DCB3  3A 4C 4F 00            C  i_rtc_lo_m       db      ':LO',NUL               ; Error #1      (must remain in
DCB7  3A 48 49 00            C  i_rtc_hi_m       db      ':HI',NUL               ; Error #2      order for addr.
DCBB  3A 4E 52 00            C  i_rtc_nr_m       db      ':NR',NUL               ; Error #3      calculation!)
DCBF  4B 65 79 62 6F 61      C  i_kb_m           db      'Keyboard    ',NUL      ; Pass/Fail/Fail:ST
      72 64 20 20 20 20      C
      00                     C
DCCC  3A 53 54 00            C  i_kb_st_m        db      ':ST',NUL
DCD0  50 72 69 6E 74 65      C  i_prt_m          db      'Printer Port',NUL      ; Pass/Fail:xx
      72 20 50 6F 72 74      C
      00                     C
DCDD  53 65 72 69 61 6C      C  i_com_m          db      'Serial Comm.',NUL      ; Pass/Fail:xx
      20 43 6F 6D 6D 2E      C
      00                     C
DCEA  20 6B 62 20 52 41      C  i_RAM_m          db         ' kb RAM  ',NUL    ; Pass/Fail:cc:y000:zzzz:wwww:rrrr
      4D 20 20 00            C
DCF8  4F 70 74 69 6F 6E      C  i_optROM_m       db      'Optional ROM',NUL      ; Pass/Fail:xxxx
      61 6C 20 52 4F 4D      C
      00                     C
DD01  46 6C 6F 70 70 79      C  i_fduA_m         db      'Floppy (A:) ',NUL      ; Ready/Not Ready/Fail:xx
      20 28 41 3A 29 20      C
      00                     C
DD0E  46 6C 6F 70 70 79      C  i_fduB_m         db      'Floppy (B:) ',NUL
      20 28 42 3A 29 20      C
      00                     C
DD1B  20 4E 6F 74            C  i_fdu_not_m      db      ' Not'                  ; purposely no NUL!!!!!
DD1F  20 52 65 61 64 79      C  i_fdu_rdy_m      db      ' Ready',CR,LF,NUL
      0D 0A 00               C
DD28  46 69 78 65 64 20      C  i_hdu_m          db      'Fixed Disk  ',NUL      ; Pass/Fail:xx
      44 69 73 6B 20 20      C
      00                     C
DD35  53 65 6C 65 63 74      C  i_alt_select_m   db      'Select Alternate CPU (y/n)?',NUL
      20 41 6C 74 65 72      C
      6E 61 74 65 20 43      C
      50 55 20 28 79 2F      C
      6E 29 3F 00            C
                             C
                             C
                             C  ;                db      'Disk Diagnostics xxxxx'
                             C  ;                db      'Loop Diagnostics xxxxx'
                             C  ;                db      'Primary BootStrap Floppy (A:)  Not Ready
                             C  ;                db      'Insert system disk and type any key.'
                             C  ;                db      'Primary BootStrap Floppy (A:)  Fail:xx'
                             C  ;                db      'Primary BootStrap Floppy (B:)  Fail:xx'
                             C  ;                db      'Primary BootStrap Fixed Disk   Fail:xx'
                             C  ;                db      'Select Operating System?
                             C  ;                db      'Serial BootStrap'
                             C
DD51  0F                     C  i_cal_val        db      0Fh     ; port 074h = units of minutes  (0-9)
DD52  07                     C                   db      07h     ; port 075h = tens of minutes   (0-5)
DD53  0F                     C                   db      0Fh     ; port 076h = units of hours    (0-9)
DD54  03                     C                   db      03h     ; port 077h = tens of hours     (0-2)
DD55  0F                     C                   db      0Fh     ; port 078h = units of days     (0-9)
DD56  03                     C                   db      03h     ; port 079h = tens of days      (0-3)
DD57  07                     C                   db      07h     ; port 07Ah = day of week       (0-7)
DD58  0F                     C                   db      0Fh     ; port 07Bh = tens of months    (0-9)
DD59  01                     C                   db      01h     ; port 07Ch = units of months   (0-1)
                             C
DD5A                         C  p1_data1         endp
                             C
                             C  ;=========================================================================
DD5A                         C  diagnostics_1    proc    near
                             C                   assume  cs:code, ds:nothing, es:nothing, ss:nothing
                             C
DD5A  FA                     C          cli                          ; disable interrupts
DD5B  B0 40                  C          mov     al,40h               ; Check Point #0
DD5D  BA 0378                C          mov     dx,378h              ; parallel port data port address
DD60  EE                     C          out     dx,al                ; output "Running- Checkpoint 0"
DD61  FC                     C          cld                          ; clear string direction flag
                             C
                             C  ;-------------------------------------------------------------------------
                             C  ;       8086 CPU Test
                             C  ;-------------------------------------------------------------------------
                             C
DD62                         C  i_cpu:
                             C
                             C  ; Flags Test (All Set):  SF, ZF, AF, PF, CF & OF.
                             C  ; (Exercises flags and accumulator only.)
                             C
DD62  B8 FF00                C          mov     ax,0FF00h            ; ah = all 1's; al = all 0's
DD65  9E                     C          sahf                         ; set SF, ZF, AF, PF, & CF
DD66  79 25                  C          jns     i_cpu_err            ; SF set?  if not, abort
DD68  75 23                  C          jnz     i_cpu_err            ; ZF set?  if not, abort
DD6A  7B 21                  C          jnp     i_cpu_err            ; PF set?  if not, abort
DD6C  73 1F                  C          jnb     i_cpu_err            ; CF set?  if not, abort
                             C
DD6E  37                     C          aaa                          ; to test if AF is set, al must be <= 9
                             C                                       ; if AF set, then:  (ah +=1) == 0;
                             C                                       ; al = ((al+6) & 0Fh)== 6; CF = AF == 1
```

```
DD6F  73 1C              C          jnb     i_cpu_err                ; CF = AF set?  if not, abort
DD71  0A E4              C          or      ah,ah                    ; ah = 0?  if not, abort
DD73  75 18              C          jnz     i_cpu_err
                         C
DD75  B0 40              C          mov     al,40h                   ; ah = 0
DD77  02 C0              C          add     al,al                    ; al = 40h + 40h = 80h = -128
DD79  71 12              C          jno     i_cpu_err                ; OF set?  if not, abort
                         C
                         C  ; Flags Test (All Reset):  SF, ZF, AF, PF, CF & OF.
                         C  ; (Exercises flags and accumulator only.)
                         C
DD7B  33 C0              C          xor     ax,ax                    ; ax = 0
DD7D  9E                 C          sahf                             ; reset SF, ZF, AF, PF, & CF
DD7E  78 0D              C          js      i_cpu_err                ; SF reset?  if not, abort
DD80  76 0B              C          jbe     i_cpu_err                ; ZF or CF reset?  if not, abort
DD82  7A 09              C          jp      i_cpu_err                ; PF reset?  if not, abort
                         C
DD84  37                 C          aaa                              ; to test if AF reset, al must be <= 9
                         C                                           ; if AF reset, then:  ah unchanged;
                         C                                           ; al = (al & 0Fh) >= 0; CF = AF == 0
DD85  72 06              C          jb      i_cpu_err                ; CF = AF reset?  if not, abort
DD87  03 C0              C          add     ax,ax                    ; ax = 0?  if not, abort
DD89  75 02              C          jnz     i_cpu_err
                         C                                           ; ax = 0 + 0 = 0, so should be no OF.
DD8B  71 11              C          jno     i_cpu_ok                 ; OF reset?  if not, abort
                         C
                         C
                         C                  assume  cs:code, ds:code, es:abs0, ss:code
                         C
DD8D                     C  i_cpu_err:
DD8D  8C C8              C          mov     ax,cs                    ; satisfy assumptions
DD8F  8E D8              C          mov     ds,ax
DD91  8E D0              C          mov     ss,ax                    ; use ROM 'stack'
DD93  BC DB7E R          C          mov     sp,cs:(offset stack_rom)
                         C
DD96  BE DC4B R          C          mov     si,cs:(offset i_cpu_m)
DD99  32 E4              C          xor     ah,ah                    ; clear ah (no error number to report)
DD9B  E9 E0E9 R          C          jmp     i_fatal                  ; i_fatal will `ret' to i_rom
                         C
DD9E                     C  i_cpu_ok:
DD9E  8C C8              C          mov     ax,cs                    ; satisfy assumptions
DDA0  8E D8              C          mov     ds,ax
DDA2  8E D0              C          mov     ss,ax                    ; use ROM 'stack'
DDA4  BC DB7E R          C          mov     sp,cs:(offset stack_rom)
                         C
DDA7  B0 81              C          mov     al,81h                   ; Check Point #1
DDA9  BA 0378            C          mov     dx,378h                  ; parallel port data port address
DDAC  EE                 C          out     dx,al                    ; output "Running- Checkpoint 1"
                         C  ; Reset the keyboard
                         C
DDAD  B0 00              C          mov     al,0
DDAF  E6 61              C          out     p_kotrl,al
DDB1  C3                 C          ret                              ; will `ret' to i_rom
                         C
                         C  ;--------------------------------------------------------------------------
                         C  ;       ROM Module Test
                         C  ;--------------------------------------------------------------------------
                         C
                         C
                         C                  assume  cs:code, ds:code, es:abs0, ss:code
DDB2                     C  i_rom:
                         C
                         C  ; Calculate Checksum of ROM.
                         C
DDB2  BE E000            C          mov     si,0E000h                ; ROM starts at ds:si = F000:E000
DDB5  E9 E5E5 R          C          jmp     rom_checksum             ; `call' rom_checksum
DDB8                     C  i_rom_ret:                               ; will `ret' here
DDB8  74 06              C          jz      i_rom_ok
                         C
                         C
DDBA                     C  i_rom_err:
DDBA  BE DC58 R          C          mov     si,cs:(offset i_rom_m)
DDBD  E9 E0E9 R          C          jmp     i_fatal                  ; ah has illegal checksum
                         C                                           ; i_fatal will `ret' to i_dmat
                         C
DDC0                     C  i_rom_ok:
DDC0  B0 82              C          mov     al,82h                   ; Check Point #2
DDC2  BA 0378            C          mov     dx,378h                  ; parallel port data port address
DDC5  EE                 C          out     dx,al                    ; output "Running- Checkpoint 2"
DDC6  C3                 C          ret                              ; will `ret' to i_dmat
                         C
                         C  ;--------------------------------------------------------------------------
                         C  ;       8253 p_dma p_timer Test
                         C  ;--------------------------------------------------------------------------
                         C
                         C                  assume  cs:code, ds:code, es:abs0, ss:code
                         C
DDC7                     C  i_dmat:
                         C
                         C  ; Disable 8237A p_dma Controller before the testing of the 8253 p_dma p_timer channel.
                         C
DDC7  B0 04              C          mov     al,dma_cmd_disable       ; disable p_dma controller command
DDC9  E6 08              C          out     dma_command,al
                         C
                         C  ; Proceed with the testing of the 8253 p_dma p_timer channel ()p_8253_1.
                         C
DDCB  B0 74              C          mov     al,074h          ; 01 11 010 0 -> p_8253_1, lsb 1st, mode 2, no BCD
DDCD  BA 0041            C          mov     dx,p_8253_1              ; select p_dma refresh counter
DDD0  E9 E1F6 R          C          jmp     rtc_chk                  ; `call' rtc_chk
DDD3                     C  i_dmat_ret:                              ; will `ret' here
DDD3  74 06              C          jz      i_dmat_ok
```

```
                                C
                                C
DDD5                            C i_dmat_err:
DDD5  BE DC65 R                 C       mov     si,cs:(offset i_dmat_m)
DDD8  E9 E0E9 R                 C       jmp     i_fatal                 ; ah has error code to report.
                                C                                       ; i_fatal will `ret' to i_dmac
                                C
DDDB                            C i_dmat_ok:
DDDB  B0 43                     C       mov     al,43h                  ; Check Point #3
DDDD  BA 0378                   C       mov     dx,378h                 ; parallel port data port address
DDE0  EE                        C       out     dx,al                   ; output "Running- Checkpoint 3"
DDE1  C3                        C       ret                             ; will `ret' to i_dmac
                                C
                                C ;-------------------------------------------------------------------------
                                C ;       8237 p_dma Controller Test -- Test Chip's Operation & Channel Registers
                                C ;
                                C ;       The 8237A p_dma Controller was disabled before the testing of the
                                C ;       8253 p_dma p_timer channel.
                                C ;-------------------------------------------------------------------------
                                C
                                C               assume  cs:code, ds:code, es:abs0, ss:code
                                C
DDE2                            C i_dmac:
                                C
                                C ; Send a `master clear' to 8237 p_dma Controller.
                                C
DDE2  E6 0D                     C       out     dma_master_clr,al       ; send master clear to  port
                                C
                                C ; The dma_command, dma_status, dma_request, dma_temp, and dma_ff registers
                                C ; are cleared, and the dma_mask register is set (all off).
                                C ; Test readable control registers:  dma_status & dma_temp)
                                C
DDE4  B4 01                     C       mov     ah,1                    ; TEMP Error #1
                                C
DDE6  E4 0D                     C       in      al,dma_temp
DDE8  0A C0                     C       or      al,al                   ; al = 0?
DDEA  75 72                     C       jnz     i_dmac_err              ; if not, abort
                                C
                                C ; Test all 8 16-bit readable/writeable channel registers (address and count
                                C ; registers for all 4 channels, i.e., ports 0 through 7) with register bit test:
                                C ; (dma_addr_x & dma_count_x are tested with OFFFFh and then 0h for x = 0 to 3.)
                                C
DDEC  BB FFFF                   C       mov     bx,0FFFFh               ; bx = 0 all bits reset
                                C
DDEF                            C i_dmac_pass2:                         ; outer loop
                                C                                       ; if 1st pass, bx = 0FFFFh
                                C                                       ; if 2nd pass, bx = 0h
DDEF  BA 0007                   C       mov     dx,7                    ; loop counter and port address!!!
                                C
DDF2                            C i_dmac_lp:                            ; inner loop
DDF2  8B C3                     C       mov     ax,bx                   ; get bit test pattern
DDF4  EE                        C       out     dx,al                   ; write low byte of address/count
DDF5  EE                        C       out     dx,al                   ; write high byte of address/count
DDF6  EC                        C       in      al,dx                   ; read low byte of address/count
DDF7  8A E0                     C       mov     ah,al                   ; save low byte in ah
DDF9  EC                        C       in      al,dx                   ; read high byte of address/count
                                C
DDFA  3B C3                     C       cmp     ax,bx                   ; does what's been read = test pattern?
DDFC  B4 02                     C       mov     ah,2                    ; TEMP Error #2
DDFE  75 5E                     C       jnz     i_dmac_err              ; if not, abort
                                C
DE00  4A                        C       dec     dx                      ; did we decrement past port address 0?
DE01  79 EF                     C       jns     i_dmac_lp               ; if not, continue same pass for all 8.
                                C
DE03  43                        C       inc     bx                      ; 1st pass?  if so, bx = 0FFFFh & loop
DE04  74 E9                     C       jz      i_dmac_pass2            ; 2nd pass?  if so, bx = 0 & continue
                                C
                                C ; We are done testing all 8 16-bit readable/writeable channel registers (address
                                C ; and count registers for all 4 channels) with the following results:  All the
                                C ; address registers (dma_addr_x) and count registers (dma_count_x) have been
                                C ; initialized to zero.
                                C
                                C ; Load 64k (0FFFFh+1) count for RAM refresh p_dma controller channel.
                                C
DE06  B0 FF                     C       mov     al,0FFh
DE08  E6 01                     C       out     dma_count_0,al          ; low byte of count for 64k RAM refresh
DE0A  E6 01                     C       out     dma_count_0,al          ; high byte of count for 64k RAM refresh
                                C
                                C ; Load mode for RAM refresh p_dma controller channel:  channel 0, read, auto-
                                C ; initialize, increment, single mode.
                                C
DE0C  B0 58                     C       mov     al,dma_mode_0           ; mode for RAM refresh
DE0E  E6 0B                     C       out     dma_mode,al
                                C
                                C ; Enable p_dma controller:  memory-to-I/O, controller enable, normal, fixed
                                C ; priority, late write, and DREQ/~DACK.
                                C
DE10  B0 00                     C       mov     al,dma_cmd_enable       ; enable p_dma controller
DE12  E6 08                     C       out     dma_command,al
                                C
                                C ; The master clear command above has masked off all channels. Now, we `unmask'
                                C ; the RAM refresh dma_mask bit.  p_dma RAM refresh begins for the first time!
                                C
DE14  B0 00                     C       mov     al,dma_unmask_0         ; turn on RAM refresh channel 0
DE16  E6 0A                     C       out     dma_mask_bit,al
                                C
                                C ; Program p_8253_1 of i8253 p_timer to proper value for RAM refresh.
                                C
DE18  B0 74                     C       mov     al,t1cmd                ; select p_dma refresh counter
DE1A  E6 43                     C       out     p_8253_ctrl,al
```

```
DE1C  B8 0013                   C               mov     ax,tlcount              ; load p_dma refresh count
DE1F  E6 41                     C               out     p_8253_1,al
DE21  8A C4                     C               mov     al,ah
DE23  E6 41                     C               out     p_8253_1,al
                                C
                                C       ; Check dma_status for `hot' p_dma request from p_8253_1.
                                C
DE25  B4 03                     C               mov     ah,3                    ; TEMP Error #3
DE27  E4 08                     C               in      al,dma_status           ; test for RAM refresh request in status
DE29  A8 10                     C               test    al,010h                 ; bit #4 -> channel 0 request
DE2B  75 31                     C               jnz     i_dmac_err              ; if `hot' p_dma request is there, abort
                                C
                                C       ; Initialize other p_dma counters and modes.
                                C
DE2D  BA 000B                   C               mov     dx,dma_mode
                                C
                                C       ; Initialize p_dma channel 1 not used.
                                C
DE30  B0 41                     C               mov     al,dma_mode_1           ; mode for not used
DE32  EE                        C               out     dx,al
                                C
                                C       ; Initialize p_dma channel 2 FDU.
                                C
DE33  B0 56                     C               mov     al,dma_mode_2           ; mode for FDU
DE35  EE                        C               out     dx,al
                                C
                                C       ; Initialize p_dma channel 3 display.
                                C
DE36  B0 43                     C               mov     al,dma_mode_3           ; mode for display
DE38  EE                        C               out     dx,al
                                C
                                C       ; Initialize p_dma Segment Nibble Latches to zero.
                                C       ; (dma_segm_x for x = 0 to 3 is port addresses 80h to 83h).
                                C
DE39  32 C0                     C               xor     al,al                   ; al = 0
DE3B  BA 0083                   C               mov     dx,083h                 ; loop counter and port address!
                                C                                               ; dx = dma_segm_3 = 083h
DE3E                            C       i_dmac_nib:
DE3E  EE                        C               out     dx,al
DE3F  FE CA                     C               dec     dl                      ; when dl goes from 80h (-128) to
DE41  78 FB                     C               js      i_dmac_nib              ; 07Fh (+127) we will exit
                                C
                                C       ;-----------------------------------------------------------------------
                                C       ;       8237 p_dma Controller Test -- Test Lowest 64k bank of RAM
                                C       ;-----------------------------------------------------------------------
                                C
                                C                       assume  cs:code, ds:data, es:abs0, ss:code
                                C
DE43  2E: 8E 1E E5F2 R          C               mov     ds,word ptr cs:[set_ds_word]    ; satisfy assumptions
DE48  8B 36 0072 R              C               mov     si,word ptr ds:[reset_flag]     ; save reset_flag
                                C
DE4C  33 D2                     C               xor     dx,dx                   ; dx = 0; test 0000:0 to 0000:FFFF
DE4E  E9 E266 R                 C               jmp     memtst                  ; `call' memtst
DE51                            C       i_dmac_ret:                             ; will `ret' here
                                C
DE51  2E: 8E 1E E5F2 R          C               mov     ds,word ptr cs:[set_ds_word]    ; satisfy assumptions
DE56  89 36 0072 R              C               mov     word ptr ds:[reset_flag],si     ; restore reset_flag
                                C
DE5A  B4 04                     C               mov     ah,4                    ; TEMP Error #4
DE5C  74 06                     C               jz      i_dmac_ok
                                C
                                C
DE5E                            C       i_dmac_err:
DE5E  BE DC72 R                 C               mov     si,cs:(offset i_dmac_m)
DE61  E9 E0E9 R                 C               jmp     i_fatal                 ; ah has error code to report.
                                C                                               ; i_fatal will `ret' to i_pic
                                C
DE64                            C       i_dmac_ok:
DE64  B0 44                     C               mov     al,44h                  ; Check Point #4
DE66  BA 0378                   C               mov     dx,378h                 ; parallel port data port address
DE69  EE                        C               out     dx,al                   ; output "Running- Checkpoint 4"
DE6A  C3                        C               ret                             ; will `ret' to_pic
                                C
                                C       ;-----------------------------------------------------------------------
                                C       ;       8259A Programmable Interrupt Controller Test.
                                C       ;-----------------------------------------------------------------------
                                C
                                C                       assume  cs:code, ds:data, es:abs0, ss:stack_ram
                                C
DE6B                            C       i_pic:
DE6B  B8 0030                   C               mov     ax,stack_seg            ;Initialize RAM Stack
DE6E  8E D0                     C               mov     ss,ax                   ;  on lower tested memory
DE70  BC 0100                   C               mov     sp,100h
                                C
                                C       ;       Initialize & Disable 8259A Programmable Interrupt Controller.
                                C
DE73  E8 E1DC R                 C               call    i_pic_init
                                C
                                C       ; Install Interrupt Vectors for diagnostics.
                                C
                                C       ; Install unexpected diagnostic interrupt vectors.
                                C
DE76  33 F6                     C               xor     si,si                           ; es:si = abs0_seg:int00loc
DE78  8B FE                     C               mov     di,si                           ; es:di = abs0_seg:int00loc
DE7A  B9 01FE                   C               mov     cx,(0400h-0004h)/2              ; words from 0:0004h to 0:0400h
                                C
DE7D  B8 DEDC R                 C               mov     ax,cs:(offset i_pic_err)        ; store offset i_pic_err
DE80  AB                        C               stosw
DE81  8C C8                     C               mov     ax,cs                           ; store segment address
```

```
DE83  AB                      C           stosw                                   ; es:di = abs0_seg:int00locn + 4
DE84  F3/ 26: A5              C           rep     movs    word ptr es:0004h,word ptr es:0000h   ; replicate vector
                              C
                              C
                              C ; Install software diagnostic interrupt vectors.
                              C
DE87  B8 DEA9 R               C           mov     ax,cs:(offset i_pic_0_ok)       ; ax = offset i_pic_0_ok
DE8A  33 FF                   C           xor     di,di                           ; es:di = abs0_seg:int00locn
DE8C  B1 05                   C           mov     cl,5                            ; load INT's 0h through 4h.
                              C
DE8E                          C i_pic_soft:                                       ; ax = (4*x)+(i_pic_0_ok)
DE8E  AB                      C           stosw                                   ; es:di gets offset i_pic_x_ok
DE8F  47                      C           inc     di                              ; skip segment (already = cs)
DE90  47                      C           inc     di
DE91  05 0004                 C           add     ax,4                            ; i_pic_x_ok are 4 bytes apart
DE94  E2 F8                   C           loop    i_pic_soft                      ; until cx = 0.
                              C
                              C ; Install hardware diagnostic interrupt vectors.
                              C
DE96  B8 FF23 R               C           mov     ax,cs:(offset ill_int)          ; ax = offset ill_int
DE99  BF 0020 R               C           mov     di,es:(offset int08locn)        ; es:di = abs0_seg:int08locn
DE9C  B1 08                   C           mov     cl,8                            ; load INT's 8h through Fh.
                              C
DE9E                          C i_pic_hard:
DE9E  AB                      C           stosw                                   ; es:di gets offset ill_int
DE9F  47                      C           inc     di                              ; skip segment (already = cs)
DEA0  47                      C           inc     di
DEA1  E2 FB                   C           loop    i_pic_hard                      ; until cx = 0.
                              C
                              C ; Test software interrupts first (cl = 0 if error).
                              C
DEA3  B7 09                   C           mov     bh,09h                  ; bx has its 8th and 11th bits set.
                              C
                              C                                           ; cx = 0 from loop above.
DEA5  F6 F5                   C           div     ch                      ; generate a divide-by-zero INT 00h
DEA7  EB 33                   C           jmp     short i_pic_err
DEA9                          C i_pic_0_ok:
                              C                                           ; bh = 09h. set trap & OF flags in bx
                              C                                           ; (bits 8 and 11 of flags).
DEA9  53                      C           push    bx                      ; put trap flags on stack
DEAA  9D                      C           popf                            ; generate a single-step trap INT 01h
DEAB  EB 2F                   C           jmp     short i_pic_err         ; must be 4 bytes long!
DEAD                          C i_pic_1_ok:
                              C
DEAD  CD 02                   C           INT     02h                     ; generate a software interrupt INT 02h
DEAF  EB 2B                   C           jmp     short i_pic_err         ; must be 4 bytes long!
DEB1                          C i_pic_2_ok:
                              C
DEB1  CC                      C           INT     03h                     ; generate a 1-byte break-point INT 03h
DEB2  90                      C           nop                             ; must be 4 bytes long!
DEB3  EB 27                   C           jmp     short i_pic_err
DEB5                          C i_pic_3_ok:
                              C                                           ; OF overflow flag is still set.
DEB5  CE                      C           INTO                            ; generate an overflow interrupt INT 04h
DEB6  90                      C           nop                             ; must be 4 bytes long!
DEB7  EB 23                   C           jmp     short i_pic_err
DEB9                          C i_pic_4_ok:
                              C
                              C ; Test hardware interrupts second.
                              C
                              C ; Test 8259A PIC interrupt mask with test patterns (cl = 0 if error).
                              C
DEB9  B0 01                   C           mov     al,1                    ; initialize mask value  = 1
                              C
DEBB                          C i_pic_test:
DEBB  E8 E1ED R               C           call    i_out_mask              ; output pattern, test input
DEBE  75 1C                   C           jne     i_pic_err               ; if not same pattern, abort
DEC0  D0 D0                   C           rcl     al,1                    ; rotate test pattern
DEC2  73 F7                   C           jnc     i_pic_test              ; test again, if not finished
                              C
DEC4  B0 FF                   C           mov     al,0FFh                 ; test pattern of all ones
DEC6  E8 E1ED R               C           call    i_out_mask              ; output pattern, test input
DEC9  75 11                   C           jne     i_pic_err               ; if not same pattern, abort
                              C
                              C ; Look for `hot' (active though masked off) PIC interrupts (cl = IR# if error).
                              C
                              C ; Enable Interrupts for the very first time!
                              C
DECB  FB                      C           sti                             ; enable interrupts
DECC  33 C9                   C           xor     cx,cx                   ; delay awhile, waiting for
DECE  E2 FE                   C           loop    $                       ; a `hot' interrupt.
                              C
DED0  A0 006B R               C           mov     al,byte ptr ds:[intr_flag]      ; get the flag from ill_int
DED3  0A C0                   C           or      al,al                   ; intr_flag = 0?
DED5  74 34                   C           jz      i_pic_ok                ; if so, we're all done.
                              C
                              C ; Convert `hot' interrupt mask (bit pattern) to IR# (1 to 8 error code).
                              C
DED7                          C i_pic_hot:                                ; cx = 0 from loop above.
DED7  41                      C           inc     cx                      ; increment cx (IR#+1).
DED8  D0 D8                   C           rcr     al,1                    ; mask's least significant bit.
DEDA  73 FB                   C           jnb     i_pic_hot               ; if not set, continue.
                              C                                           ; (exit with cl = 1 to 8.)
                              C
DEDC                          C i_pic_err:                                ; cl = 0 or failing PIC `hot' active IR#
DEDC  BC 0100                 C           mov     sp,100h                 ; re-initialize stack
                              C
DEDF  51                      C           push    cx                      ; save error code.
                              C
                              C ; Install Vector Table.                   ; set int10locn = code_seg:v_io, and
```

```
                                C                                       ; set int1Dloon = code_seg:v_parms.
DEE0  E8 E1A0 R                 C               call    i_vector
                                C
                                C ; Initialize Video.
                                C
DEE3  E8 E148 R                 C               call    i_d_init
                                C
                                C ; Display error message.
                                C
DEE6  BE DC7F R                 C               mov     si,cs:(offset i_pic_m)
DEE9  E8 E5FA R                 C               call    DRomString              ; display failing test message.
                                C
DEEC  BE DC45 R                 C               mov     si,cs:(offset fail_m)
DEEF  E8 E5FA R                 C               call    DRomString              ; display fail message.
                                C
DEF2  59                        C               pop     cx                      ; restore error code.
DEF3  0A C9                     C               or      cl,cl                   ; cl = 0?
DEF5  74 0E                     C               jz      i_pic_no_hot            ; if so, we're done.
                                C
                                C ; Display `hot' interrupt number :Hx. (where x is the IR# from 0 to 7)
                                C
DEF7  E8 E626 R                 C               call    DColon                  ; display a colon.
DEFA  B8 0E48                   C               mov     ax,(0Eh*100h)+'H'       ; display `hot' interrupt symbol.
DEFD  CD 10                     C               int     10h
DEFF  8A C1                     C               mov     al,cl                   ; transfer error code.
DF01  48                        C               dec     ax                      ; error code (1 to 8) to (0 to 7) IR#.
DF02  E8 E650 R                 C               call    DHexNib                 ; display lowest nibble.
                                C
DF05                            C i_pic_no_hot:
DF05  E8 E619 R                 C               call    DCrLf
DF08  EB 07                     C               jmp     short i_pic_end
                                C
DF0A  F4                        C               hlt
                                C
DF0B                            C i_pic_ok:
DF0B  B0 45                     C               mov     al,45h                  ; Check Point #5
DF0D  BA 0378                   C               mov     dx,378h                 ; parallel port data port address
DF10  EE                        C               out     dx,al                   ; output "Running- Checkpoint 5"
                                C
DF11                            C i_pic_end:
                                C
                                C ;-------------------------------------------------------------------------------
                                C ;       Install Vector Table.
                                C ;-------------------------------------------------------------------------------
                                C
DF11  BC 0100                   C               mov     sp,100h                 ; re-initialize stack
DF14  E8 E1A0 R                 C               call    i_vector
                                C
                                C ;-------------------------------------------------------------------------------
                                C ;       Determine System Configuration from Switches and Initialize Video.
                                C ;-------------------------------------------------------------------------------
                                C
DF17  E8 E148 R                 C               call    i_d_init
                                C
                                C ;-------------------------------------------------------------------------------
                                C ;       Display Passing Error Messages
                                C ;-------------------------------------------------------------------------------
                                C
DF1A                            C disp_pass:
DF1A  BE DB8C R                 C               mov     si,cs:(offset banner_m)
DF1D  E8 E5FA R                 C               call    DRomString
                                C
DF20  BE DC4B R                 C               mov     si,cs:(offset i_cpu_m)
DF23  E8 E5FA R                 C               call    DRomString
DF26  BE DC3D R                 C               mov     si,cs:(offset pass_m)
DF29  E8 E5FA R                 C               call    DRomString
                                C
DF2C  BE DC58 R                 C               mov     si,cs:(offset i_rom_m)
DF2F  E8 E5FA R                 C               call    DRomString
DF32  BE DC3D R                 C               mov     si,cs:(offset pass_m)
DF35  E8 E5FA R                 C               call    DRomString
                                C
DF38  BE DC65 R                 C               mov     si,cs:(offset i_dmat_m)
DF3B  E8 E5FA R                 C               call    DRomString
DF3E  BE DC3D R                 C               mov     si,cs:(offset pass_m)
DF41  E8 E5FA R                 C               call    DRomString
                                C
DF44  BE DC72 R                 C               mov     si,cs:(offset i_dmac_m)
DF47  E8 E5FA R                 C               call    DRomString
DF4A  BE DC3D R                 C               mov     si,cs:(offset pass_m)
DF4D  E8 E5FA R                 C               call    DRomString
                                C
DF50  BE DC7F R                 C               mov     si,cs:(offset i_pic_m)
DF53  E8 E5FA R                 C               call    DRomString
DF56  BE DC3D R                 C               mov     si,cs:(offset pass_m)
DF59  E8 E5FA R                 C               call    DRomString
                                C
                                C ;-------------------------------------------------------------------------------
                                C ;       Size & clear RAM at every 64k byte bank past the lowest 64k.
                                C ;-------------------------------------------------------------------------------
                                C
DF5C                            C RAM_size_tst:
                                C                       assume  cs:code, ds:data, es:abs0, ss:stack_ram
                                C
DF5C  BD 0040                   C               mov     bp,64                           ;initialize memory count
                                C
DF5F  8B 36 0072 R              C               mov     si,word ptr ds:[reset_flag]     ; get warm bootflag
DF63  81 EE 1234                C               sub     si,01234h               ; si=0 iff CTL ALT DEL sequence.
DF67  33 FF                     C               xor     di,di                   ; offset = 0000h
                                C
```

```
DF69  BA 1000                C          mov     dx,1000h                ; start at 1000:0000 (dx keeps segment)
DF6C                         C  RAM_size_lp:
DF6C  8E C2                  C          mov     es,dx                   ; get segment
                             C
DF6E  26: 8B 05              C          mov     ax,word ptr es:[di]     ; read existing ram value
DF71  F7 D0                  C          not     ax                      ; complement it
DF73  26: 89 05              C          mov     word ptr es:[di],ax     ; write complement back to RAM
                             C
DF76  26: 8B 1D              C          mov     bx,word ptr es:[di]     ; read back from RAM
                             C
DF79  3B C3                  C          cmp     ax,bx                   ; verify to test for end of RAM
DF7B  F7 D0                  C          not     ax                      ; recreate original value
DF7D  75 2C                  C          jne     RAM_size_end            ; if verify fails, at end of RAM
                             C
DF7F  26: 89 05              C          mov     word ptr es:[di],ax     ; restore original value back to RAM
DF82  0B F6                  C          or      si,si                   ; test warm boot flag
DF84  74 1D                  C          jz      RAM_size_nxt            ; if CTL ALT DEL sequence,
                             C                                          ; don't clear memory
DF86  E8 E266 R              C          call    memtst                  ; test and clear memory
DF89  75 3E                  C          jnz     RAM_error               ; test flag from storage test
                             C
DF8B  83 C5 40               C          add     bp,64                   ; increment size
DF8E  56                     C          push    si                      ; save Warm Boot Flag
DF8F  B8 0E0D                C          mov     ax,0E0Dh                ; put out a CR
DF92  CD 10                  C          INT     10H
                             C
DF94  8B C5                  C          mov     ax,bp                   ; display tested RAM
DF96  BB 0003                C          mov     bx,3
DF99  E8 E66D R              C          call    DNumW
DF9C  BE DCEA R              C          mov     si,cs:(offset i_RAM_m)
DF9F  E8 E5FA R              C          call    DRomString
                             C
DFA2  5E                     C          pop     si                      ; retrieve Warm Boot Flag
                             C
DFA3                         C  RAM_size_nxt:
DFA3  80 C6 10               C          add     dh,10h                  ; maximum RAM = 640k = 10 * 64k
DFA6  80 FE A0               C          cmp     dh,0A0h                 ; next segment
DFA9  72 C1                  C          jb      RAM_size_lp             ; top of RAM yet (A000:0000)?
                             C                                          ; if not, continue.
                             C
DFAB                         C                  assume  cs:code, ds:data, es:abs0, ss:stack_ram
DFAB  0B F6                  C  RAM_size_end:
DFAD  74 06                  C          or      si,si                   ; test warm boot flag
DFAF  BE DC3D R              C          jz      RAM_size_end_1          ; if CTL ALT DEL sequence,
DFB2  E8 E5FA R              C          mov     si,cs:(offset pass_m)   ;Display OK
                             C          call    DRomString
DFB5                         C
DFB5  33 C0                  C  RAM_size_end_1:                         ; bx = RAM size/16
DFB7  8E C0                  C          xor     ax,ax                   ; ax = 0
DFB9  8A C6                  C          mov     es,ax                   ; satisfy assumptions es = 0 = abs0_seg
DFBB  D1 E0                  C          mov     al,dh                   ; ax = (RAM size/16)/256 = RAM size/4k
DFBD  D1 E0                  C          shl     ax,1                    ; ax = (RAM size/4k) * 2 = RAM size/2k
                             C          shl     ax,1                    ; ax = (RAM size/2k) * 2 = RAM size/1k
DFBF  2E: 8E 1E E5F2 R       C
DFC4  A3 0013 R              C          mov     ds,word ptr cs:[set_ds_word]    ; restore data segment pointer
                             C          mov     word ptr ds:[memory_size],ax
                             C
                             C
                             C  ;       GoTo Display Passing Messages
                             C
DFC7  EB 30                  C          jmp short i_cal                 ;Go check clock calendar
                             C
                             C
DFC9                         C  RAM_error:              ; Error message looks like: Fail:cc:y000:zzzz:wwww:rrrr
                             C                          ; where:   cc = RAM configuration number
                             C                          ;       y000 = Segment of failure       = dx = es
                             C                          ;       zzzz = Offset of failure        = di
                             C                          ;       wwww = Data that was written    = ax
                             C                          ;       rrrr = Data that was read       = bx
                             C
DFC9  52                     C          push    dx                      ;save failing segment
DFCA  1E                     C          push    ds                      ;save ds
DFCB  50                     C          push    ax                      ; save failing test pattern.
                             C
DFCC  E8 E619 R              C          call    DCrLf                   ;Carriage Return, Line Feed
DFCF  BE DC45 R              C          mov     si,cs:(offset fail_m)
DFD2  E8 E5FA R              C          call    DRomString              ; display fail message.
                             C
DFD5  E8 E626 R              C          call    DColon                  ; display a colon
                             C
DFD8  E4 66                  C          in      al,sys_conf_a           ; get RAM configuration.
DFDA  24 0F                  C          and     al,0Fh                  ; mask valid bits.
DFDC  E8 E643 R              C          call    DHexByte                ; display RAM configuration.
                             C
DFDF  E8 E626 R              C          call    DColon                  ; display a colon
                             C
DFE2  8E DA                  C          mov     ds,dx                   ; ds = failing segment = dx
DFE4  8B C7                  C          mov     ax,di                   ; ax = failing segment = di
DFE6  E8 E632 R              C          call    DHexLong                ; display ds:ax
                             C
DFE9  E8 E626 R              C          call    DColon                  ; display a colon
                             C
DFEC  1F                     C          pop     ds                      ; ds = failing test pattern = on stack
DFED  8B C3                  C          mov     ax,bx                   ; ax = what was read = bx
DFEF  E8 E632 R              C          call    DHexLong                ; display ds:ax
DFF2  E8 E619 R              C          call    DCrLf
                             C
DFF5  1F                     C          pop     ds                      ;restore ds
DFF6  5A                     C          pop     dx                      ;restore failing segment
DFF7  EB BC                  C          jmp     short RAM_size_end_1
                             C
```

```
                        C  ;--------------------------------------------------------------------------
                        C  ;       MM58174 Clock Calendar Device Test
                        C  ;--------------------------------------------------------------------------
                        C
                        C                       assume  cs:code, ds:data, es:abs0, ss:stack_ram
                        C
DFF9                    C  i_cal:
                        C
                        C  ; Read Clock Calendar Device.                ; bx = day (from 1-1 of leap year)
                        C                                               ; ch = hour
DFF9  B4 FE             C          mov     ah,-2                        ; cl = minutes
DFFB  CD 1A             C          int     1Ah                          ; dh = seconds
                        C                                               ; dl = hundredths of seconds
                        C  ; Check time & date read.
                        C
                        C  ; Hundredths of Seconds.
                        C
DFFD  B8 000A           C          mov     ax,10                        ; ax = 10; dl = hundredths
E000  86 C2             C          xchg    al,dl                        ; ax = hundredths; dl = 10
E002  F6 F2             C          div     dl                           ; ah = remainder, al = quotient
                        C                                               ; (can only read tenths of seconds.)
E004  3D 000A           C          cmp     ax,10                        ; ah should = 0, and al should be < 10.
E007  73 15             C          jae     i_cal_1_1_80                 ; if not, test chip & write 1-1-80.
                        C
                        C  ; Seconds.
                        C
E009  80 FE 3C          C          cmp     dh,60                        ; dh = seconds should be < 60.
E00C  73 10             C          jae     i_cal_1_1_80                 ; if not, test chip & write 1-1-80.
                        C
                        C
                        C  ; Minutes.
                        C
E00E  80 F9 3C          C          cmp     cl,60                        ; cl = minutes should be < 60.
E011  73 0B             C          jae     i_cal_1_1_80                 ; if not, test chip & write 1-1-80.
                        C
                        C  ; Hours.
                        C
E013  80 FD 18          C          cmp     ch,24                        ; ch = hour should be < 24.
E016  73 06             C          jae     i_cal_1_1_80                 ; if not, test chip & write 1-1-80.
                        C
                        C  ; Days.
                        C
E018  81 FB 0B6A        C          cmp     bx,(2*366)+(6*365)           ; bx = day from leap year mod 8 should
                        C                                               ;    be < (0-2921) = (0-0B69h)
E01C  72 5D             C          jb      i_cal_end                    ; if so, valid time & day, skip test.
                        C
E01E                    C  i_cal_1_1_80:                                ; else invalid time & day, write 1-1-80.
                        C
                        C  ; Initialize and Stop Clock.
                        C
E01E  33 C0             C          xor     ax,ax                        ; ax = 0
E020  E6 70             C          out     70h,al                       ; test only port = out of test mode
E022  E6 7E             C          out     7Eh,al                       ; stop/start port = stop clock
                        C
                        C  ; Output test pattern of maximum value with all bits set to read/writable ports.
                        C
E024  BE DD51 R         C          mov     si,cs:(offset i_cal_val)
E027  B9 0009           C          mov     cx,9                         ; ch keeps 0; cx = 9
E02A  BA 0074           C          mov     dx,0074h                     ; dh keeps 0; dx = 74h
                        C
E02D                    C  i_cal_max:
E02D  2E: AC            C          lods    byte ptr cs:[si]             ; al get cs:si (ds overridden!).
E02F  8A E0             C          mov     ah,al                        ; save maximum value.
E031  EE                C          out     dx,al                        ; ports 74 through 7C (units of minutes
                        C                                               ; to tens of months) get max value.
E032  EC                C          in      al,dx                        ; read it back.
E033  22 C4             C          and     al,ah                        ; mask valid bits.
E035  3A C4             C          cmp     al,ah                        ; is it equal to the value written?
E037  75 28             C          jnz     i_cal_err                    ; if not, abort.
                        C
E039  42                C          inc     dx                           ; increment to next port
E03A  E2 F1             C          loop    i_cal_max
                        C
E03C  B0 07             C          mov     al,07h
E03E  B2 7F             C          mov     dl,7Fh                       ; dh kept 0; dx = 7Fh
E040  EE                C          out     dx,al                        ; interrupt (year mod 8) gets 0Fh.
E041  EC                C          in      al,dx                        ; must do 'in' from this port 3 times.
E042  EC                C          in      al,dx
E043  EC                C          in      al,dx
E044  24 07             C          and     al,07h                       ; mask valid bits.
E046  2C 07             C          sub     al,07h                       ; is it equal to the value written?
E048  75 17             C          jnz     i_cal_err                    ; if not, abort.
                        C
                        C  ; Write out 0h (bits of lower nibble reset) test pattern to read/writable ports.
                        C
                        C                                               ; al kept 0h.
E04A  B1 09             C          mov     cl,9                         ; ch kept 0; cx = 9
E04C  B2 74             C          mov     dl,74h                       ; dh kept 0; dx = 74h
E04E                    C  i_cal_0:
E04E  EE                C          out     dx,al                        ; ports 74 through 7C (units of minutes
                        C                                               ; to tens of months) get 0h.
E04F  EC                C          in      al,dx                        ; read it back.
E050  24 0F             C          and     al,0Fh                       ; mask valid bits (lower nibble) = 0h?
E052  75 0D             C          jnz     i_cal_err                    ; if not, abort.
                        C
E054  42                C          inc     dx                           ; increment to next port
E055  E2 F7             C          loop    i_cal_0
                        C
E057  B2 7F             C          mov     dl,7Fh                       ; dh kept 0; dx = 7Fh
E059  EE                C          out     dx,al                        ; interrupt (year mod 8) gets 0Fh.
```

```
E05A  EC                C          in      al,dx                   ; must do `in' from this port 3 times.
E05B  EC                C          in      al,dx
E05C  EC                C          in      al,dx
E05D  24 0F             C          and     al,0Fh                  ; mask valid bits (lower nibble) = 0h?
E05F  74 12             C          jz      i_cal_ok                ; if so, we're all done.
                        C                                          ; else, abort.
E061                    C  i_cal_err:
E061  BE DCA6 R         C          mov     si,cs:(offset i_rtc_m)
E064  E8 E5FA R         C          call    DRomString
E067  BE DCA5 R         C          mov     si,cs:(offset fail_m)
E06A  E8 E5FA R         C          call    DRomString              ; display fail message.
E06D  E8 E619 R         C          call    DCrLf
E070  EB 01             C          jmp     short i_cal_ok          ; try to write 1-1-80, regardless...
                        C
E072  F4                C          hlt
                        C
E073                    C  i_cal_ok:
                        C
E073  33 DB             C          xor     bx,bx                   ; 1-1-80 is day 0.
E075  33 C9             C          xor     cx,cx                   ; hours & minutes = 0.
                        C
                        C  ; Write & Start Clock Calendar Device.
                        C
E077  B4 FF             C          mov     ah,-1                   ; bx = day (from 1-1 of leap year)
E079  CD 1A             C          int     1Ah                     ; ch = hour
                        C                                          ; cl = minutes
                        C                                          ; Output: ah = -1 implies date/time err.
                        C                                          ;         ah = 0  implies date/time OK.
                        C
E07B                    C  i_cal_end:
                        C
                        C  ;---------------------------------------------------------------------------
                        C  ;       18253 Real-Time Time Clock Test (p_8253_1 tested in i_dmat)
                        C  ;---------------------------------------------------------------------------
                        C
                        C                  assume  cs:code, ds:data, es:abs0, ss:stack_ram
E07B                    C  i_rtc:
                        C
                        C  ; String to be displayed regardless of results.
                        C
E07B  BE DCA6 R         C          mov     si,cs:(offset i_rtc_m)
E07E  E8 E5FA R         C          call    DRomString
                        C
                        C  ; Test 18253 real-time clock interrupt p_timer counter (p_8253_0).
                        C
E081  B0 34             C          mov     al,034h         ; 00 11 010 0 -> p_8253_0, lsb 1st, mode 2, no BCD
E083  BA 0040           C          mov     dx,p_8253_0     ; select real-time clock counter
E086  E8 E1F6 R         C          call    rtc_chk
E089  75 2E             C          jnz     i_rtc_err       ; if nz, ah has error code to report.
                        C
                        C  ; Test 18253 tone generator p_timer counter (p_8253_2).
                        C
E08B  B0 01             C          mov     al,1            ; clear kb interrupts, reset kb, disable parity
E08D  E6 61             C          out     p_kotrl,al      ; turn off speaker data -- bit #1
                        C                                  ; turn on  speaker gate to p_8253_2 -- bit #0
                        C
E08F  B0 B4             C          mov     al,0B4h         ; 10 11 010 0 -> p_8253_2, lsb 1st, mode 2, no BCD
E091  BA 0042           C          mov     dx,p_8253_2     ; select tone generator counter
E094  E8 E1F6 R         C          call    rtc_chk
                        C
E097  B0 00             C          mov     al,0            ; clear kb interrupts, reset kb, disable parity
E099  E6 61             C          out     p_kotrl,al      ; turn off speaker data & gate -- bits #1 & #0
                        C
E09B  75 1C             C          jnz     i_rtc_err       ; if nz, ah has error code to report.
                        C
                        C  ; Initialize 18253 real-time clock interrupt p_timer counter (p_8253_0).
                        C
E09D  B0 36             C          mov     al,t0cmd                ; select real time clock counter
E09F  E6 43             C          out     p_8253_ctrl,al
                        C
E0A1  B8 0000           C          mov     ax,t0count              ; load real time clock count
E0A4  E6 40             C          out     p_8253_0,al
E0A6  8A C4             C          mov     al,ah
E0A8  E6 40             C          out     p_8253_0,al
                        C
                        C  ; Initialize 18253 tone generator p_timer counter (p_8253_2).
                        C
E0AA  B0 B6             C          mov     al,t2cmd                ; select tone generator counter
E0AC  E6 43             C          out     p_8253_ctrl,al
                        C
E0AE  B8 0266           C          mov     ax,t2count              ; load tone generator count
E0B1  E6 42             C          out     p_8253_2,al
E0B3  8A C4             C          mov     al,ah
E0B5  E6 42             C          out     p_8253_2,al
                        C
E0B7  EB 1C             C          jmp     short i_rtc_ok
                        C
                        C
E0B9                    C  i_rtc_err:                              ; ah has error code to report.
E0B9  BE DC45 R         C          mov     si,cs:(offset fail_m)
E0BC  E8 E5FA R         C          call    DRomString              ; display fail message.
                        C
E0BF  BE DCAF R         C          mov     si,cs:(offset i_rtc_lo_m-(4*1))
E0C2  8A C4             C          mov     al,ah                   ; al = error code = (1, 2 or, 3)
E0C4  32 E4             C          xor     ah,ah                   ; ax = error code = (1, 2 or, 3)
E0C6  D1 E0             C          shl     ax,1                    ; ax = 2*(error code) = (2, 4, or 6)
E0C8  D1 E0             C          shl     ax,1                    ; ax = 4*(error code) = (4, 8, or 12)
E0CA  03 F0             C          add     si,ax                   ; index to (LO, HI, or NR message.)
E0CC  E8 E5FA R         C          call    DRomString              ; display failing mode.
E0CF  E8 E619 R         C          call    DCrLf
```

```
                        C
E0D2  EB 0D             C               jmp     short i_rtc_end
                        C
E0D4  F4                C               hlt
                        C
E0D5                    C   i_rtc_ok:
E0D5  BE DC3D R         C               mov     si,cs:(offset pass_m)
E0D8  E8 E5FA R         C               call    DRomString
                        C
E0DB  B0 48             C               mov     al,48h                  ; Check Point #8
E0DD  BA 0378           C               mov     dx,378h                 ; parallel port data port address
E0E0  EE                C               out     dx,al                   ; output "Running- Checkpoint 8"
                        C
E0E1                    C   i_rtc_end:
E0E1  B8 0E07           C               mov     ax,(0Eh*100h)+BEL       ; beep keyboard
E0E4  CD 10             C               int     10h
                        C
                        C                       assume  cs:code, ds:nothing, es:nothing, ss:stack_ram
                        C
E0E6  E9 E2E4 R         C               jmp     poinit
                        C
E0E9                    C   diagnostics_1   endp
                        C
                        C   ;--------------------------------------------------------------------------
                        C   ;       Fatal Error Routine.
                        C   ;
                        C   ;       Input:  cs:si   = points to offset of failing error message
                        C   ;               if ah <> 0, do DHexByte of ah.
                        C   ;               if ah =  0, do nothing (just print error).
                        C   ;       Output: None.
                        C   ;
                        C   ;       Trash:  al, dx, & si destroyed.
                        C   ;--------------------------------------------------------------------------
                        C
E0E9                    C   i_fatal proc    near
                        C                   assume  cs:code, ds:nothing, es:nothing, ss:nothing
                        C
                        C   ; Disable 8237A p_dma Controller.
                        C
E0E9  B0 04             C               mov     al,dma_cmd_disable      ; disable p_dma controller command
E0EB  E6 08             C               out     dma_command,al
                        C
                        C   ; Send a `master clear' to 8237 p_dma Controller.
                        C
E0ED  E6 0D             C               out     dma_master_clr,al       ; send master clear port any garbage
                        C
                        C   ; Load 64k (OFFFFh+1) count for RAM refresh p_dma controller channel.
                        C
E0EF  B0 FF             C               mov     al,0FFh
E0F1  E6 01             C               out     dma_count_0,al          ; low byte of count for 64k RAM refresh
E0F3  E6 01             C               out     dma_count_0,al          ; high byte of count for 64k RAM refresh
                        C
                        C   ; Load mode for RAM refresh p_dma controller channel:  channel 0, read, auto-
                        C   ; initialize, increment, single mode.
                        C
E0F5  B0 58             C               mov     al,dma_mode_0           ; mode for RAM refresh
E0F7  E6 0B             C               out     dma_mode,al
                        C
                        C   ; Enable p_dma controller:  memory-to-I/O, controller enable, normal, fixed
                        C   ; priority, late write, and DREQ/~DACK.
                        C
E0F9  B0 00             C               mov     al,dma_cmd_enable       ; enable p_dma controller
E0FB  E6 08             C               out     dma_command,al
                        C
                        C   ; The master clear command above has masked off all channels. Now, we `unmask'
                        C   ; the RAM refresh dma_mask bit.  p_dma RAM refresh begins for the first time!
                        C
E0FD  B0 00             C               mov     al,dma_unmask_0         ; turn on RAM refresh channel 0
E0FF  E6 0A             C               out     dma_mask_bit,al
                        C
                        C   ; Program p_8253_1 of i8253 p_timer to proper value for RAM refresh.
                        C
E101  B0 74             C               mov     al,t1cmd                ; select p_dma refresh counter
E103  E6 43             C               out     p_8253_ctrl,al
                        C
E105  B0 13             C               mov     al,t1count              ; load p_dma refresh count
E107  E6 41             C               out     p_8253_1,al
E109  32 C0             C               xor     al,al
E10B  E6 41             C               out     p_8253_1,al
                        C
                        C                       assume  cs:code, ds:nothing, es:nothing, ss:stack_ram
                        C
E10D  8C D7             C               mov     di,ss                   ; save stack pointer
E10F  8B EC             C               mov     bp,sp
E111  BA 0030           C               mov     dx,stack_seg
E114  8E D2             C               mov     ss,dx
E116  BC 0100           C               mov     sp,100h
                        C
E119  50                C               push    ax                      ; save error code
                        C
                        C   ; Initialize & Disable 8259A Programmable Interrupt Controller.
                        C
E11A  E8 E1DC R         C               call    i_pic_init
                        C
                        C   ; Install Vector Table.                     ; set int10locn = code_seg:v_io, and
                        C                                               ; set int1Dlocn = code_seg:v_parms.
E11D  E8 E1A0 R         C               call    i_vector
                        C
                        C   ; Initialize Video.
                        C
```

```
E120  E8 E148 R           C          call    i_d_init
                          C
                          C ; Display error message.
                          C
E123  58                  C          pop     ax                      ; restore error code
                          C
E124  E8 E5FA R           C          call    DRomString              ; display string at cs:si.
                          C
E127  BE DC45 R           C          mov     si,cs:(offset fail_m)
E12A  E8 E5FA R           C          call    DRomString              ; display fail message.
                          C
E12D  0A E4               C          or      ah,ah                   ; ah = 0?
E12F  74 08               C          jz      i_fatal_ret             ; if so, no arguments
                          C
E131  E8 E626 R           C          call    DColon                  ; display a colon
                          C
E134  8A C4               C          mov     al,ah                   ; display error code
E136  E8 E643 R           C          call    DHexByte
                          C
E139                      C i_fatal_ret:
E139  E8 E619 R           C          call    DCrLf
                          C
                          C ;Output fatal error status for manufacturing tests
                          C
E13C  BA 0378             C          mov     dx,378h                 ; parallel port address
E13F  EC                  C          in      al,dx                   ; read last checkpoint value
E140  34 3F               C          xor     al,03fh                 ; extract checkpoint number from status
E142  EE                  C          out     dx,al                   ; output "Not OK - number"
                          C
                          C                  assume  cs:code, ds:nothing, es:nothing, ss:nothing
                          C
E143  8E D7               C          mov     ss,di                   ; restore stack pointer
E145  8B E5               C          mov     sp,bp
                          C ;        ret
                          C
E147  F4                  C          hlt
                          C
E148                      C i_fatal endp
                          C
                          C ;--------------------------------------------------------------------
                          C ;        Determine System Configuration from Switches and Enable Video.
                          C ;
                          C ;        Input:  None.
                          C ;        Output: None.
                          C ;
                          C ;        Trash:  ax & cx destroyed.
                          C ;--------------------------------------------------------------------
                          C
E148                      C i_d_init         proc    near
                          C                  assume  cs:code, ds:nothing, es:nothing, ss:stack_ram
                          C
E148  1E                  C          push    ds                              ; save registers
                          C
                          C ; Initialize both boards.
                          C
                          C                  assume  cs:code, ds:data, es:nothing, ss:stack_ram
                          C
E149  B8 0040             C          mov     ax,data_seg                     ; ds = ax = data_seg = 0040h.
E14C  8E D8               C          mov     ds,ax                           ; (ah = 0.)
                          C
                          C ; Initialize monochrome board.
                          C
E14E  B0 30               C          mov     al,30h                          ; switch_bits for monochrome.
E150  A3 0010 R           C          mov     word ptr ds:[switch_bits],ax    ; set data for monochrome.
E153  CD 10               C          INT     10h                             ; ah = 0 = v_set_mode.
                          C
                          C ; Initialize color board.
                          C
E155  B8 0003             C          mov     ax,0003h                        ; switch_bits for not monochrome.
E158  A3 0010 R           C          mov     word ptr ds:[switch_bits],ax    ; set data for color.
E15B  CD 10               C          INT     10h                             ; ah = 0 = v_set_mode.
                          C
                          C ; Determine system configuration from switches (low byte of switch_bits).
                          C
E15D  E4 67               C          in      al,sys_conf_b           ; read high nibble of
                          C                                          ; system configuration switches.
                          C                                          ; bits #7 - #6: (number of FDU's)-1
                          C                                          ; bits #5 - #4: monitor type
E15F  24 F0               C          and     al,0F0h                 ; mask off low nibble (keep high nibble)
                          C                                          ; of low byte; clear high byte.
E161  0C 0D               C          or      al,00Dh                 ; ALWAYS 64k planar RAM and >= 1 FDU!
                          C
E163  8A C8               C          mov     cl,al                   ; cl is ok, excepts bits #5 & #4.
E165  B5 03               C          mov     ch,03h                  ; initialize display to mode #3 (default).
                          C
E167  24 30               C          and     al,030h                 ; isolate display switches (bits #5 & #4).
E169  74 1C               C          jz      i_d_80x25               ; if zero, default to 80x25 color.
E16B  3C 30               C          cmp     al,030h                 ; is it the monochrome board?
E16D  75 1E               C          jnz     i_d_ok          ; if not, 40x25 or 80x25 color ok.
                          C
                          C                  assume  cs:code, ds:v_ram, es:nothing, ss:stack_ram
                          C
E16F  1E                  C          push    ds                              ; save ds = data_seg = 0040h.
E170  B8 B000             C          mov     ax,para_mono                    ; satisfy assumptions
E173  8E D8               C          mov     ds,ax
E175  B0 A5               C          mov     al,0A5h                         ; test pattern
E177  A2 0000             C          mov     byte ptr ds:[0000h],al          ; if so, is monochrome there?
E17A  8A 26 0000          C          mov     ah,byte ptr ds:[0000h]          ; read monochrome RAM
E17E  1F                  C          pop     ds                              ; restore ds = data_seg = 0040h.
                          C
```

```
                        C                       assume  cs:code, ds:data, es:nothing, ss:stack_ram
                        C
E17F  3A C4             C               cmp     al,ah                           ; if monochrome RAM is there,
E181  75 04             C               jnz     i_d_80x25                       ; then the board must be there!
                        C                                                       ; if not, default to 80x25 color
E183  B5 07             C               mov     ch,07h                          ; if there, we believe switches,
E185  EB 06             C               jmp     short i_d_ok                    ; initialize display to mode #7.
                        C
E187                    C       i_d_80x25:
E187  80 E1 EF          C               and     cl,0EFh                         ; reset bit #4 for 80x25 color.
E18A  80 C9 20          C               or      cl,020h                         ;   set bit #5 for 80x25 color.
                        C
E18D                    C       i_d_ok:
                        C
                        C       ; Set system configuration (switch_bits) from switches.
                        C
E18D  32 E4             C               xor     ah,ah                           ; ah = 0.
E18F  8A C1             C               mov     al,cl                           ; get data from switches.
E191  A3 0010 R         C               mov     word ptr ds:[switch_bits],ax    ; save data from switches
                        C
                        C       ; Determine mode to initialize display monitor (from switches).
                        C
E194  A8 20             C               test    al,020h                         ; does user want 40x25 color?
E196  75 02             C               jnz     i_d_mode
E198  B5 01             C               mov     ch,01h                          ; if so, initialize mode #1.
E19A                    C       i_d_mode:
                        C
                        C       ; Initialize desire board (from switches).
                        C
E19A  8A C5             C               mov     al,ch                           ; transfer display mode to al.
E19C  CD 10             C               INT     10h                             ; ah = 0 = v_set_mode.
                        C
E19E  1F                C               pop     ds                              ; restore registers
E19F  C3                C               ret
                        C
E1A0                    C       i_d_init        endp
                        C
                        C       ;-----------------------------------------------------------------------------
                        C       ;       Install Vector Table
                        C       ;
                        C       ;       Input:  None.
                        C       ;       Output: None.
                        C       ;
                        C       ;       Trash:  ax = cx = 0 destroyed.
                        C       ;-----------------------------------------------------------------------------
                        C
E1A0                    C       i_vector        proc    near
                        C                       assume  cs:code, ds:nothing, es:nothing, ss:stack_ram
                        C
E1A0  1E                C               push    ds                              ; save registers
E1A1  06                C               push    es
E1A2  57                C               push    di
E1A3  56                C               push    si
                        C
                        C       ; Initialize Interrupt Vectors 00h through 07h to known routines.
                        C
                        C                       assume  cs:code, ds:abs0, es:abs0, ss:stack_ram
                        C
E1A4  33 FF             C               xor     di,di                           ; satisfy assumptions
E1A6  8E DF             C               mov     ds,di                           ; ds = es = ax = abs0_seg = 0
E1A8  8E C7             C               mov     es,di                           ; es:di = abs0_seg:int00locn
E1AA  B8 FF23 R         C               mov     ax,cs:(offset ill_int)          ; ax = offset ill_int
E1AD  B9 0008           C               mov     cx,(07h-00h)+1                  ; load INT's 00h through 07h.
                        C
E1B0  AB                C       i_vec0: stosw                                   ; es:di++ gets offset ill_int
E1B1  8C 0D             C               mov     word ptr ds:[di],cs             ; es:di   gets cs
E1B3  47                C               inc     di                              ; di++
E1B4  47                C               inc     di
E1B5  E2 F9             C               loop    i_vec0                          ; until cx = 0.
                        C
                        C                                                       ; load INT's 02h and 05h
E1B7  C7 06 0014 R FF54 R C             mov     word ptr ds:[int05locn],cs:(offset s_int)
E1BD  C7 06 0008 R F85F R C             mov     word ptr ds:[int02locn],cs:(offset n_int)
                        C
                        C       ; Initialize Interrupt Vectors 08h through 1Eh to known routines.
                        C
                        C                       assume  cs:code, ds:code, es:abs0, ss:stack_ram
                        C
E1C3  8C C8             C               mov     ax,cs                           ; satisfy assumptions
E1C5  8E D8             C               mov     ds,ax                           ; ds = ax = cs
E1C7  BE FEF3 R         C               mov     si,cs:(offset i_vec_tbl)        ; ds:si = code_seg:i_vec_tbl
                        C                                                       ; es:di = abs0_seg:int08locn
E1CA  B1 18             C               mov     cl,(1Fh-08h)+1                  ; load INT's 08h through 1Fh.
                        C
E1CC  A5                C       i_vec8: movsw                                   ; es:di++ gets ds:si   (offset)
E1CD  AB                C               stosw                                   ; es:di++ gets ax = cs (segment)
E1CE  E2 FC             C               loop    i_vec8                          ; until cx = 0.
                        C
                        C       ; Initialize Interrupt Vectors 20h and above to zero.
                        C
E1D0  33 C0             C               xor     ax,ax                           ; ax = 0
                        C                                                       ; es:di = abs0_seg:int20locn
E1D2  B9 01B8           C               mov     cx,((03F0h-0080h)/2)            ; clear 0:0080h to 0:03F0h
                        C                                                       ; don't blow away stack!
E1D5  F3/ AB            C               rep     stosw                           ; es:di++ gets 0
                        C
E1D7  5E                C               pop     si                              ; restore registers
E1D8  5F                C               pop     di
E1D9  07                C               pop     es
E1DA  1F                C               pop     ds
```

```
E1DB  C3                C          ret
C
E1DC                    C  i_vector        endp
C
C  ;--------------------------------------------------------------------------
C  ;       Initialize & Disable 8259A Programmable Interrupt Controller.
C  ;
C  ;       Input:  None.
C  ;       Output: None.
C  ;
C  ;       Trash:  al & dx destroyed.
C  ;--------------------------------------------------------------------------
C
E1DC                    C  i_pic_init      proc    near
C                               assume  cs:code, ds:nothing, es:nothing, ss:stack_ram
C
E1DC  BA 0020           C          mov     dx,pic_0                ; dx = pic_0 (8259A `control' port)
E1DF  B0 13             C          mov     al,pic_icw1             ; edge triggered, single, icw4 to follow
E1E1  EE                C          out     dx,al
C
E1E2  42                C          inc     dx                      ; dx = pic_1 (8259A `data' port)
E1E3  B0 08             C          mov     al,pic_icw2             ; interrupt vector base address
E1E5  EE                C          out     dx,al
C                                                  ; since we are single mode (no slave), skip icw3
C
C                                                  ; dx = pic_1 (8259A `data' port)
E1E6  B0 0D             C          mov     al,pic_icw4             ; not special fully nested, buffered,
E1E8  EE                C          out     dx,al                   ; master, normal end_of_int, 8086 mode
C
E1E9  B0 FF             C          mov     al,pic_off_msk          ; mask all interrupts off for now
E1EB  EE                C          out     dx,al                   ; dx = pic_1 (8259A `data' port)
E1EC  C3                C          ret
C
E1ED                    C  i_pic_init      endp
C
C  ;--------------------------------------------------------------------------
C  ;       Output Mask to 8259A Programmable Interrupt Controller.
C  ;
C  ;       Input:  AL = mask pattern
C  ;
C  ;       Output: Flags
C  ;
C  ;       Trash:  ah destroyed.
C  ;--------------------------------------------------------------------------
C
E1ED                    C  i_out_mask      proc    near
C                               assume  cs:code, ds:nothing, es:nothing, ss:stack_ram
C
E1ED  E6 21             C          out     pic_1,al                ;output interrupt mask pattern
E1EF  8A E0             C          mov     ah,al                   ;save pattern for compar
E1F1  E4 21             C          in      al,pic_1                ;get mask from 8259
E1F3  3A E0             C          cmp     ah,al                   ;the same ?
E1F5  C3                C          ret                             ;return flags = result of compare
C
E1F6                    C  i_out_mask      endp
C
C
C  ;--------------------------------------------------------------------
C  ;       8253 p_timer test for one p_timer counter channel
C  ;
C  ;       Input:  al      = 8253 p_timer control byte
C  ;               dx      = port address of 8253 p_timer data (counter)
C  ;
C  ;       Output: zf      = set (z status) if no error; reset (nz status) if error
C  ;               ah      = Error codes:  0 -> No Error!
C  ;                                       1 -> Low below time interval window.
C  ;                                       2 -> High above time interval window.
C  ;                                       3 -> No Response.
C  ;
C  ;       Trash:  al, bx & cx destroyed.
C  ;--------------------------------------------------------------------
C
E1F6                    C  rtc_chk proc    near
C                               assume  cs:code, ds:nothing, es:nothing, ss:nothing
C
E1F6  8A E0             C          mov     ah,al                   ; save control byte for later.
E1F8  B9 FFFF           C          mov     cx,0FFFFh               ; time out for both Register Bit Tests.
C
C  ; Register Bit Test (All Reset):  Count down from 100h until all bits reset.
C
E1FB  8B D9             C          mov     bx,cx                   ; bx gets all its bits set.
C
E1FD  E6 43             C          out     p_8253_ctrl,al          ; send i8253 p_timer control byte.
C
E1FF  32 C0             C          xor     al,al                   ; al = 00h
E201  EE                C          out     dx,al                   ; load low byte of p_timer count.
E202  FE C0             C          inc     al                      ; al = 01h
E204  EE                C          out     dx,al                   ; load high byte of p_timer count.
C
E205                    C  rtc_chk_reset_lp:
E205  8A C4             C          mov     al,ah                   ; get control byte for read.
E207  24 C0             C          and     al,0C0h                 ; mask off all but top 2 bits.
E209  E6 43             C          out     p_8253_ctrl,al          ; send latching control byte for read.
C
E20B  EC                C          in      al,dx                   ; get low byte of p_timer count.
E20C  22 D8             C          and     bl,al                   ; `and' low byte.
E20E  EC                C          in      al,dx                   ; get high byte of p_timer count.
E20F  22 F8             C          and     bh,al                   ; `and' high byte.
C
E211  0B DB             C          or      bx,bx                   ; is bx = 0?
```

```
E213  74 05                C          jz      rtc_chk_reset_ok        ; if so, we're done.
E215  E2 EE                C          loop    rtc_chk_reset_lp        ; if not, continue reading.
                           C                                          ; (Note: loops less than 16 times.)
                           C
E217                       C  rtc_chk_reset_err:                      ; time out.
E217  B4 03                C          mov     ah,3                    ; Error #3. (No Response.)
E219  C3                   C          ret                             ; return nz status (loop leaves zf ok).
                           C
E21A                       C  rtc_chk_reset_ok:
                           C
                           C  ; Register Bit Test (All Set):  Count down from 0h (FFFFh+1) until all bits set.
                           C
E21A  33 DB                C          xor     bx,bx                   ; bx gets all its bits reset.
                           C
E21C  8A C4                C          mov     al,ah                   ; get control byte for load.
E21E  E6 43                C          out     p_8253_ctrl,al          ; send 18253 p_timer control byte.
                           C
E220  32 C0                C          xor     al,al                   ; al = 00h
E222  E8                   C          out     dx,al                   ; load low byte of p_timer count.
E223  E8                   C          out     dx,al                   ; load high byte of p_timer count.
                           C
E224                       C  rtc_chk_set_lp:
E224  8A C4                C          mov     al,ah                   ; get control byte for read.
E226  24 C0                C          and     al,0C0h                 ; mask off all but top 2 bits.
E228  E6 43                C          out     p_8253_ctrl,al          ; send latching control byte for read.
                           C
E22A  EC                   C          in      al,dx                   ; get low byte of p_timer count.
E22B  0A D8                C          or      bl,al                   ; `or' low byte.
E22D  EC                   C          in      al,dx                   ; get high byte of p_timer count.
E22E  0A F8                C          or      bh,al                   ; `or' high byte.
                           C
E230  81 FB FFFF           C          cmp     bx,0FFFFh               ; is bx = 0FFFFh?
E234  74 05                C          jz      rtc_chk_set_ok          ; if so, we're done.
E236  E2 EC                C          loop    rtc_chk_set_lp          ; if not, continue reading.
                           C                                          ; (Note: loops less than 16 times.)
                           C
E238                       C  rtc_chk_set_err:                        ; time out.
E238  B4 03                C          mov     ah,3                    ; Error #3. (No Response.)
E23A  C3                   C          ret                             ; return nz status (loop leaves zf ok).
                           C
E23B                       C  rtc_chk_set_ok:
                           C
                           C  ; p_timer Time Window Test:  Test p_timer versus CPU & see if it falls within spec.
                           C
E23B  8A C4                C          mov     al,ah                   ; get control byte for read.
E23D  24 C0                C          and     al,0C0h                 ; mask off all but top 2 bits.
E23F  E6 43                C          out     p_8253_ctrl,al          ; send latching control byte for read.
                           C
E241  EC                   C          in      al,dx                   ; get low byte of p_timer count.
E242  8A D8                C          mov     bl,al                   ; save low byte.
E244  EC                   C          in      al,dx                   ; get high byte of p_timer count.
E245  8A F8                C          mov     bh,al                   ; save high byte.
                           C
E247  8A C4                C          mov     al,ah                   ; get control byte for read.
E249  24 C0                C          and     al,0C0h                 ; mask off all but top 2 bits.
E24B  E6 43                C          out     p_8253_ctrl,al          ; send latching control byte for read.
                           C
E24D  EC                   C          in      al,dx                   ; get low byte of p_timer count.
E24E  8A C8                C          mov     cl,al                   ; save low byte.
E250  EC                   C          in      al,dx                   ; get high byte of p_timer count.
E251  8A E8                C          mov     ch,al                   ; save high byte.
                           C
E253  2B D9                C          sub     bx,cx                   ; calculate time difference.
                           C
                           C  ; Do Time Range Checking (4 <= bx <= 14).
                           C
E255  B4 02                C          mov     ah,2                    ; Error #2. (High above time window.)
E257  83 FB 0E             C          cmp     bx,14
E25A  77 09                C          ja      rtc_chk_high            ; return nz status. (ja has zf reset.)
                           C
E25C  FE CC                C          dec     ah                      ; Error #1. (Low below time window.)
E25E  83 FB 04             C          cmp     bx,4
E261  72 02                C          jb      rtc_chk_low             ; return nz status. (jb has zf reset.)
                           C
E263  FE CC                C          dec     ah                      ; Error #0. (No Error!) return z status.
                           C
E265                       C  rtc_chk_high:
E265                       C  rtc_chk_low:
E265  C3                   C          ret
                           C
E266                       C  rtc_chk endp
                           C
                           C  ;---------------------------------------------------------------------------
                           C  ;       RAM (64k) Storage Test.
                           C  ;
                           C  ;       Input:  dx      = segment of RAM to be tested
                           C  ;
                           C  ;       Output: zf      = set (z status) if no error; reset (nz status) if error
                           C  ;               es:di   = dx:di = failing RAM location if error; else di = 0.
                           C  ;               ax      = test pattern (what was written).
                           C  ;               bx      = if error, what was read.
                           C  ;               cx      = number left to test if error; else cx = 0.
                           C  ;
                           C  ;       Trash:  None.
                           C  ;---------------------------------------------------------------------------
                           C
E266                       C  memtst          proc    near
                           C                  assume  cs:code, ds:nothing, es:nothing, ss:nothing
                           C
E266  B9 8000              C          mov     cx,08000h               ; get word count
```

```
                                ;       (64k = 32k * 2 bytes/word)
E269  8E C2                     mov     es,dx
E26B  33 FF                     xor     di,di           ; es:di = address
E26D                    memtst_w1:
E26D  8B C7                     mov     ax,di           ;data = offset
E26F  AB                        stosw
E270  E2 FB                     loop    memtst_w1
E272  8E DA                     mov     ds,dx
E274  33 DB                     xor     bx,bx           ;ds:bx = address
E276  B9 8000                   mov     cx,08000h       ; word count
E279                    memtst_r1:
E279  8B 07                     mov     ax,[bx]         ;read data
E27B  3B C3                     cmp     ax,bx           ;verify data
E27D  75 2C                     jne     memtst_err
E27F  43                        inc     bx
E280  43                        inc     bx              ;next address
E281  E2 F6                     loop    memtst_r1

E283  B9 8000                   mov     cx,08000h       ; word count
E286                    memtst_w2:                      ; address is already ok
E286  8B C7                     mov     ax,di           ; data = offset
E288  F7 D0                     not     ax              ; complement it
E28A  AB                        stosw                   ;fill memory
E28B  E2 F9                     loop    memtst_w2

E28D  B9 8000                   mov     cx,08000h       ; word count

E290                    memtst_r2:
E290  8B 07                     mov     ax,[bx]         ; read data
E292  F7 D0                     not     ax              ; complement
E294  3B C3                     cmp     ax,bx           ; verify
E296  75 0F                     jne     memtst_err_c

E298  43                        inc     bx              ; update address
E299  43                        inc     bx
E29A  E2 F4                     loop    memtst_r2
E29C  B8 0000                   mov     ax,0            ; to clear memory
E29F  B9 8000                   mov     cx,08000h       ; word count
E2A2  F3/ AB                    rep     stosw
E2A4  0B C0                     or      ax,ax           ;set ZF
E2A6  C3                        ret

E2A7                    memtst_err_c:
E2A7  F7 D0                     not     ax              ; error during complemented
E2A9  F7 D3                     not     bx              ;  address test

E2AB                    memtst_err:
E2AB  93                        xchg    ax,bx           ;return registers as specified
E2AC  C3                        ret

E2AD                    memtst          endp

E2AD                    code    ends
                        include pwrup0.asm


                        ;=============================================================================
                        ;       Filename:       pwrup0.src
                        ;
                        ;       This module includes temporary hardware initialization.
                        ;
                        ;       Includes        Diagnostics
                        ;                       Cold Boot
                        ;                       Device Drivers
                        ;
                        ;=============================================================================

E2AD                    code    segment public 'ROM'
                        assume  cs:code, ds:nothing, es:nothing, ss:nothing


E2AD                    p0_data1        proc    near

E2AD  4F 70 74 69 6F 6E opt_ROM_m       db      'Optional ROM at ',NUL
      61 6C 20 52 4F 4D
      20 61 74 20 00

E2BE  03BC              p_tbl   dw      prt_data_a              ; printer in   port address space
E2C0  0378                      dw      prt_data_b              ; always on mother board.
E2C2  0278                      dw      prt_data_c              ; printer in   port address space
E2C4  0000                      dw      0                       ; no printer

E2C6  0052              scc_tbl dw      scc_ctl_b               ; rs232 SCC channel B
E2C8  0050                      dw      scc_ctl_a               ; rs232 SCC channel A

E2CA  E545 R            alt_ret dw      i_alt_restart           ; 00h Z8000 restart sequence offset
E2CC  F000                      dw      code_seg                ; 04h Z8000 restart sequence segment

E2CE  0016              mastab  dw      ((mt_end)-(mastab))     ; 00h master table byte length

E2D0  CC67 R                    dw      kb_data_table           ; 02h kb xlation table offset
E2D2  F000                      dw      code_seg                ; 04h kb xlation table segment

E2D4  FA6E R                    dw      font_lo_8x8             ; 06h 1st 128 char.s 8x8 font offset
E2D6  F000                      dw      code_seg                ; 08h 1st 128 char.s 8x8 font segment

E2D8  C060 R                    dw      font_lo_8x16            ; 0Ah 1st 128 char.s 8x16 font offset
E2DA  F000                      dw      code_seg                ; 0Ch 1st 128 char.s 8x16 font segment

E2DC  0000                      dw      0                       ; 0Eh 2nd 128 char.s 8x16 font offset
```

```
E2DE  0000                C          dw      0                       ; 10h 2nd 128 char.s 8x16 font segment
                          C
E2E0  0000                C          dw      0                       ; 12h soft font utility offset
E2E2  0000                C          dw      0                       ; 14h soft font utility segment
                          C
                          C                                          ; 16h etc...
E2E4                      C  mt_end  label   word
                          C
E2E4                      C  p0_data1        endp
                          C
                          C  ;--------------------------------------------------------------------------
                          C  ;       Initialize the basic hardware,
                          C  ;       Set up the interrupt pointers,
                          C  ;       Initialize all RAM variables,
                          C  ;       Clear the screen,
                          C  ;       Initialize the disk drivers,
                          C  ;       and perform the cold boot.
                          C  ;--------------------------------------------------------------------------
                          C
E2E4                      C  poinit  proc    near
                          C
                          C          assume  cs:code, ds:data, es:data, ss:stack_ram
                          C
E2E4  B8 0040             C          mov     ax,data_seg             ; satisfy assumptions
E2E7  8E D8               C          mov     ds,ax
E2E9  8E C0               C          mov     es,ax
                          C
                          C  ; Initialize Keyboard Controller.
                          C
E2EB  9C                  C          pushf                           ; save flags and
E2EC  FA                  C          cli                             ; disable interrupts
                          C
E2ED  BA 0061             C          mov     dx,p_kctrl              ; dx = p_kctrl
E2F0  B0 40               C          mov     al,40h                  ; remove keyboard reset
E2F2  EE                  C          out     dx,al
                          C
E2F3  33 C9               C          xor     cx,cx                   ; delay
E2F5  E2 FE               C          loop    $
                          C
E2F7  B4 01               C          mov     ah,1                    ; enable self test
E2F9  E8 E581 R           C          call    kb_cmd_send
E2FC  89 0E 0072 R        C          mov     word ptr ds:[reset_flag],cx     ;
                          C
E300  33 C9               C          xor     cx,cx                   ; delay
E302  E2 FE               C          loop    $
                          C  ; Flush any keyboard scan code and store AAh if we get it.
                          C
E304                      C  kb_flush:
E304  E4 64               C          in      al,kb_status            ; get 8041 status
E306  A8 01               C          test    al,1                    ; test output buffer bit
E308  74 09               C          jz      kb_flush_back           ; jump if no character pending
E30A  E4 60               C          in      al,p_kscan              ; get scan code from data port
E30C  3C AA               C          cmp     al,0AAh                 ; verify keyboard present
E30E  75 F4               C          jne     kb_flush
E310  A3 0072 R           C          mov     word ptr ds:[reset_flag],ax     ; keyboard present
E313                      C  kb_flush_back:
E313  E2 EF               C          loop    kb_flush                ;loop if zero
                          C
E315  33 C9               C          xor     cx,cx                   ; delay
E317  E2 FE               C          loop    $
                          C
                          C  ; Initialize System Variables.
                          C
E319  C7 06 0084 R E2CE R C          mov     word ptr ds:[master_tbl_ptr+0000h],cs:(offset mastab)
E31F  8C 0E 0086 R        C          mov     word ptr ds:[master_tbl_ptr+0002h],cs
                          C
                          C  ; Initialize Keyboard Driver Variables.
                          C
E323  B8 001E R           C          mov     ax,ds:(offset kb_buffer)        ; pointer to beginning of buffer
E326  A3 001A R           C          mov     word ptr ds:[buffer_head],ax    ; keyboard output pointer offset
E329  A3 001C R           C          mov     word ptr ds:[buffer_tail],ax    ; keyboard input pointer offset
E32C  A3 0080 R           C          mov     word ptr ds:[buffer_start],ax   ; keeps beginning of buffer
                          C
E32F  C7 06 0082 R 003E R C          mov     word ptr ds:[buffer_end],ds:(offset kb_buffer)+(size kb_buffer)
                          C
                          C          ; Assume first not Deluxe Keyboard
                          C
E335  80 26 0017 R DF     C          and     byte ptr ds:[kb_flag],(not num_lock_mode)
E33A  80 26 0018 R FE     C          and     byte ptr ds:[kb_flag_1],(not dlx_kb)
                          C
                          C          ; Send command to request ID code from keyboard.
                          C
E33F  B4 05               C          mov     ah,05H                  ; Read keyboard type
E341  E8 E581 R           C          call    kb_cmd_send             ; -- send command.
                          C
E344  33 C9               C          xor     cx,cx                   ; set up timeout count
E346                      C  kb_type_wait:
E346  E4 64               C          in      al,kb_status            ; get port status
E348  A8 01               C          test    al,1                    ; data byte available?
E34A  75 05               C          jnz     kb_type_read            ; if so, go read it ..
E34C  E2 F8               C           loop   kb_type_wait            ; else wait awhile longer.
E34E  EB 11 90            C          jmp     kb_not_dlx              ; timeout, default to non-dlx
                          C
E351                      C  kb_type_read:
E351  E4 60               C          in      al,p_kscan              ; read ID byte..
E353  A8 01               C          test    al,01H                  ; deluxe kbd. bit set?
E355  74 0A               C          jz      kb_not_dlx
                          C
                          C          ; 01H bit set, so initialize to Deluxe Keyboard.
                          C
```

```
E357  80 0E 0017 R 20     C          or      byte ptr ds:[kb_flag],num_lock_mode
E35C  80 0E 0018 R 01     C          or      byte ptr ds:[kb_flag_1],dlx_kb
                          C
E361                      C  kb_not_dlx:
                          C
E361  9D                  C          popf                                    ; restore interrupt-
                          C                                                  ; enable state.
                          C
                          C  ; Initialize Printer & Communication (RS-232) Driver Variables.
                          C
E362  B0 14               C          mov     al,14h                          ; printer default timeout = 20
E364  BF 0078 R           C          mov     di,ds:(offset printer_t_out)
E367  B9 0004             C          mov     cx,4
E36A  F3/ AA              C          rep     stosb                           ; es:di gets al
                          C
E36C  B0 01               C          mov     al,01h                          ; printer default timeout = 01
E36E  BF 007C R           C          mov     di,ds:(offset serial_t_out)
E371  B9 0004             C          mov     cx,4
E374  F3/ AA              C          rep     stosb                           ; es:di gets al
                          C
                          C  ; Determine Parallel Port Configuration.
                          C
                          C          assume  cs:code, ds:code, es:data, ss:stack_ram
                          C
E376  8C C8               C          mov     ax,cs
E378  8E D8               C          mov     ds,ax                           ; satisfy assumptions
                          C
E37A  32 DB               C          xor     bl,bl                           ; clear high byte of switch_bits
                          C
E37C  BF 0008 R           C          mov     di,es:(offset printer_addr)     ; es:di points at printer_base
E37F  BE E2BE R           C          mov     si,ds:(offset p_tbl)            ; addresses of printer ports
                          C
E382                      C  i_prt_loop:
E382  AD                  C          lodsw                                   ; ax gets ds:si port address.
E383  0B C0               C          or      ax,ax                           ; valid port address?
E385  74 14               C          jz      i_prt_exit                      ; exit if invalid port address
                          C
E387  8B D0               C          mov     dx,ax                           ; transfer to data register
E389  B0 A5               C          mov     al,0A5h                         ; load test pattern
E38B  EE                  C          out     dx,al                           ; output test pattern.
E38C  86 C4               C          xchg    al,ah                           ; mov ah,al; trash al; & delay.
E38E  EC                  C          in      al,dx                           ; input test pattern back.
E38F  3A C4               C          cmp     al,ah                           ; what we read = test pattern ?
E391  75 EF               C          jnz     i_prt_loop                      ; if not, loop as port is absent
                          C                                                  ; else, printer port is present
E393  8B C2               C          mov     ax,dx                           ; retrieve port address
E395  AB                  C          stosw                                   ; es:di gets ax.
                          C
E396  80 C3 40            C          add     bl,040h                         ; add to high byte of switch_bits
                          C
E399  EB E7               C          jmp     i_prt_loop                      ; will go around loop 3 times
                          C
E39B                      C  i_prt_exit:
                          C
                          C  ; Determine Communication (RS-232) Configuration (SCC Z8530 & INS8250's).
                          C
E39B  BF 0000 R           C          mov     di,es:(offset rs232_addr)       ; es:di points at rs232_base
                          C
E39E  BA 03FA             C          mov     dx,com_id_a                     ; read interrupt I.D. register
E3A1  EC                  C          in      al,dx                           ; for first 8250 port.
E3A2  A8 F8               C          test    al,0F8h                         ; bits #3-7 are always low if
E3A4  75 07               C          jnz     i_no_com_a                      ; installed.
                          C
E3A6  B8 03F8             C          mov     ax,com_data_a                   ; if present, load address of
E3A9  AB                  C          stosw                                   ; first 8250 data port.
                          C                                                  ; es:di gets ax.
E3AA  80 C3 02            C          add     bl,002h                         ; add to high byte of switch_bits
                          C
E3AD                      C  i_no_com_a:                                     ; es:di points next empty word
                          C
E3AD  BA 02FA             C          mov     dx,com_id_b                     ; read interrupt I.D. register
E3B0  EC                  C          in      al,dx                           ; for second 8250 port.
E3B1  A8 F8               C          test    al,0F8h                         ; bits #3-7 are always low if
E3B3  75 07               C          jnz     i_no_com_b                      ; installed.
                          C
E3B5  B8 02F8             C          mov     ax,com_data_b                   ; if present, load address of
E3B8  AB                  C          stosw                                   ; second 8250 data port.
                          C                                                  ; es:di gets ax.
E3B9  80 C3 02            C          add     bl,002h                         ; add to high byte of switch_bits
                          C
E3BC                      C  i_no_com_b:                                     ; read switch settings to test
E3BC  E4 66               C          in      al,sys_conf_a                   ; for SCC Z8530 chip
E3BE  A8 20               C          test    al,020h                         ; bit #5: SCC chip installed
E3C0  74 08               C          jz      i_no_scc                        ; if not, don't load SCC table
                          C
E3C2  BE E2C6 R           C          mov     si,ds:(offset scc_tbl)          ; SCC Z8530 control ports
E3C5  A5                  C          movsw                                   ; always 2 ports
E3C6  A5                  C          movsw                                   ; es:di gets ds:si (twice)
                          C
E3C7  80 C3 04            C          add     bl,2*(002h)                     ; add to high byte of switch_bits
E3CA                      C  i_no_scc:                                       ; es:di points next empty word
                          C  ; Determine Game Card Configuration.
                          C
E3CA  BA 0201             C          mov     dx,game_card                    ; get game card address.
E3CD  EC                  C          in      al,dx                           ; bits #0-3 are low if installed
E3CE  A8 0F               C          test    al,0Fh
E3D0  75 03               C          jnz     i_no_game_card                  ; skip, if not present
                          C
E3D2  80 C3 10            C          add     bl,010h                         ; add to high byte of switch_bits
                          C
```

```
E3D5                            C  i_no_game_card:
                                C
                                C  ; Initialize High Byte of switch_bits.
                                C
E3D5  26: 88 1E 0011 R          C          mov     byte ptr es:[switch_bits+1],bl  ; save high byte of switch_bits
                                C
                                C  ; Initialize i8259A PIC with appropriate interrupt mask and enable interrupts.
                                C
                                C  ;       mov     al,10111100b                    ; p_timer & kb & dsk at this point
E3DA  B0 FC                     C          mov     al,11111100b                    ; p_timer & kb only at this point.
E3DC  BA 0021                   C          mov     dx,pic_1                        ; now set proper interrupt mask
E3DF  EE                        C          out     dx,al
                                C
                                C  ; Send specific end of interrupt (SEOI) to pic `command' port for keyboard.
                                C
E3E0  B0 61                     C          mov     al,pic_seoi_1                   ; specific end of interrupt
E3E2  BA 0020                   C          mov     dx,pic_0                        ; to pic `command' port.
E3E5  EE                        C          out     dx,al
E3E6  FB                        C          sti                                     ; enable interrupts
                                C
                                C  ; Initialize Parallel Printer Interface.
                                C
E3E7  B4 01                     C          mov     ah,1                            ; initialize printer...
E3E9  33 D2                     C          xor     dx,dx                           ; ...port 0
E3EB  CD 17                     C          INT     17h
                                C
                                C  ; Initialize all 4 (2) Z8530 Serial Communication Controller.
                                C  ; NOTE: Special function code (FF) for power up ONLY initialization of 8530
E3ED  B9 0004                   C          mov     cx,4
E3F0                            C  rs_init:
E3F0  B8 FFE3                   C          mov     ax,1111111111100011b            ; initialize SCC RS-232
                                C                                                  ; 9600 baud,none,1 stop & 8 data
E3F3  8B D1                     C          mov     dx,cx                           ; port number = loop - 1
E3F5  4A                        C          dec     dx
E3F6  CD 14                     C          INT     14h
E3F8  E2 F6                     C          loop    rs_init
                                C
                                C  ;---------------------------------------------------------------------------
                                C  ;Test for and Initialize optional ROMs
                                C  ;---------------------------------------------------------------------------
                                C
                                C          assume  cs:code, ds:nothing, es:nothing, ss:nothing
                                C
E3FA  BB C800                   C          mov     bx,0C800h                       ; load starting segment
                                C
E3FD                            C  rom_scan_loop:
E3FD  8E DB                     C          mov     ds,bx                           ; bx has pending segment
E3FF  33 F6                     C          xor     si,si                           ; offset 0000h
                                C
E401  81 3C AA55                C          cmp     word ptr ds:[si],0AA55h
E405  75 43                     C          jne     rom_scan_next
                                C
E407  BE E2AD R                 C          mov     si,cs:(offset opt_ROM_m)        ; indicate ROM detected
E40A  E8 E5FA R                 C          call    DRomString
                                C
E40D  33 C0                     C          xor     ax,ax
E40F  E8 E632 R                 C          call    DHexLong                        ; ds:ax points at ROM
                                C
E412  B8 0E20                   C          mov     ax,(0Eh*100h)+' '               ; put out SPACE
E415  CD 10                     C          INT     10h
                                C
E417  B8 0040                   C          mov     ax,data_seg                     ; satisfy assumptions
E41A  8E C0                     C          mov     es,ax                           ; for es in rom_check
E41C  33 F6                     C          xor     si,si                           ; ds:si points to ROM to check
                                C
                                C  ;       Now:    ds:si   = pointer to ROM to be tested
                                C  ;               bx = ds = pending segment of ROM under test
                                C  ;               es:     = data segment
                                C
                                C          assume  cs:code, ds:nothing, es:data, ss:nothing
                                C
E41E  33 C0                     C          xor     ax,ax                           ; clear al
E420  8A 64 02                  C          mov     ah,byte ptr ds:[si+2]           ; ax = (ROM length/512) * 256
E423  D1 E0                     C          shl     ax,1                            ; ax = (ROM length/512) * 512
                                C                                                  ; ax = ROM length in bytes
E425  50                        C          push    ax                              ; save ROM length
E426  B1 04                     C          mov     cl,4
E428  D3 E8                     C          shr     ax,cl                           ; advance segment for next ROM
E42A  03 D8                     C          add     bx,ax                           ; by the number of paragraphs
E42C  59                        C          pop     cx                              ; restore ROM length in cx
                                C
E42D  E8 E5E8 R                 C          call    rom_checksum_cnt                ; get the checksum of the cx-
                                C                                                  ; byte ROM.
E430  74 03                     C          jz      rom_chksum_ok                   ; OK if the checksum was zero
E432  E9 E5D5 R                 C          jmp     rom_err                         ; error the checksum wasn't zero
                                C
E435                            C  rom_chksum_ok:
E435  53                        C          push    bx                              ; save the segment for next ROM
                                C
E436  26: C7 06 0067 R 0003     C          mov     word ptr es:[io_rom_init],0003h
E43D  26: 8C 1E 0069 R          C          mov     word ptr es:[io_rom_seg],ds
E442  26: FF 1E 0067 R          C          call    dword ptr es:[io_rom_init]      ; initialize the ROM
                                C
E447  5B                        C          pop     bx                              ; restore segment for next ROM
                                C
E448  EB 04                     C          jmp     short rom_scan_exit
                                C
E44A                            C  rom_scan_next:
E44A  81 C3 0080                C          add     bx,(800h/10h)                   ; add 2k to the pending segment
                                C
```

```
E44E                    C rom_scan_exit:
E44E  81 FB F600        C         cmp     bx,0F600h                       ; are we done?
E452  7C A9             C         jnge    rom_scan_loop                   ; if not, continue
                        C
                        C
                        C ;--------------------------------------------------------------------------
                        C ;       HDU Test
                        C ;--------------------------------------------------------------------------
                        C
                        C         assume  cs:code, ds:abs0, es:nothing, ss:stack_ram
                        C
E454  33 C0             C         xor     ax,ax                   ; satisfy assumptions
E456  8E D8             C         mov     ds,ax
                        C
                        C ; Check int 41h to see if any one installed a HDU parameter table pointer.
                        C
E458  A1 0104           C         mov     ax,word ptr ds:[(4*41h)+0000h]
E45B  0B 06 0106        C         or      ax,word ptr ds:[(4*41h)+0002h]
E45F  75 1C             C         jnz     i_hdu_ok                        ; if so, let them be...
                        C
                        C ; If not, call HDU initialization routine.
                        C
E461  BE DD28 R         C         mov     si,cs:(offset i_hdu_m)
E464  E8 E5FA R         C         call    DRomString                      ; print test message
                        C
E467  E8 CP9F R         C         call    h_init
                        C
                        C         assume  cs:code, ds:data, es:nothing, ss:stack_ram
                        C
E46A  2E: 8E 1E E5F2 R  C         mov     ds,word ptr cs:[set_ds_word]    ; satisfy assumptions
E46F  80 3E 0075 R 00   C         cmp     byte ptr ds:[hf_num],0          ; number of hard disks.
E474  75 07             C         jnz     i_hdu_ok                        ; if ok, leave everything alone.
                        C
E476  BC 0100           C         mov     sp,100h                         ; re-initialize stack
E479  FA                C         cli                                     ; disable interrupts
E47A  E8 E1A0 R         C         call    i_vector                        ; re-install old vectors
                        C
E47D                    C i_hdu_ok:
                        C
                        C ; Clean Up after Option ROM's
                        C
E47D  FA                C         cli                             ; disable interrupts
                        C
E47E  BA 0021           C         mov     dx,pic_1                ; get current interrupt mask
E481  EC                C         in      al,dx
                        C ;       and     al,10111100b            ; p_timer & kb & dsk at this point.
E482  24 FC             C         and     al,11111100b            ; p_timer & kb must be on at this point.
E484  EE                C         out     dx,al
                        C
                        C         assume  cs:code, ds:abs0, es:nothing, ss:stack_ram
                        C
E485  33 C0             C         xor     ax,ax
E487  8E D8             C         mov     ds,ax
E489  C7 06 0060 R F860 R C       mov     word ptr ds:[int18loc+0000h],cs:(offset bt_int)
E48F  8C 0E 0062 R      C         mov     word ptr ds:[int18loc+0002h],cs    ; (ROM BASIC not available)
                        C
E493  FB                C         sti                             ; enable interrupts
                        C
                        C ;--------------------------------------------------------------------------
                        C ;       FDU Test
                        C ;--------------------------------------------------------------------------
                        C
                        C ; Initialize Floppy Disk Controller and related Driver Variables
                        C
E494  33 C0             C         xor     ax,ax                   ; initialize the disk routines
E496  33 DB             C         xor     bx,bx
E498  33 C9             C         xor     cx,cx
E49A  33 D2             C         xor     dx,dx
E49C  CD 13             C         INT     13h
                        C
                        C ; Dummy Disk Attachment Test to Spin Up Drive for INT 19h (boot-strap).
                        C
E49E  BE DD01 R         C         mov     si,cs:(offset i_fduA_m)
E4A1  E8 E5FA R         C         call    DRomString                      ; print test message
                        C
E4A4  BD 0003           C         mov     bp,3                            ; loop counter
E4A7                    C i_fdu_lp:
E4A7  B8 0201           C         mov     ax,0201h                        ; read one sector
                        C
E4AA  33 DB             C         xor     bx,bx
E4AC  8E DB             C         mov     ds,bx
E4AE  8E C3             C         mov     es,bx                           ; xfer_segment
E4B0  BB 7C00           C         mov     bx,7C00h                        ; xfer_offset
                        C
E4B3  B9 0001           C         mov     cx,0001h                        ; track 0; sector 1
E4B6  33 D2             C         xor     dx,dx                           ; head  0; drive  0
                        C
E4B8  55                C         push    bp                              ; save retry count
E4B9  50                C         push    ax                              ; save return registers
E4BA  06                C         push    es
E4BB  CD 13             C         INT     13h                             ; bx, cx, dx, & ds preserved
E4BD  07                C         pop     es                              ; restore return registers
E4BE  58                C         pop     ax
E4BF  5D                C         pop     bp                              ; restore retry count
                        C
E4C0  73 08             C         jnc     i_fdu_ok                        ; error during read?
E4C2  4D                C         dec     bp                              ; if so, decrement retry count
E4C3  75 E2             C         jnz     i_fdu_lp                        ; and try again
                        C
E4C5  BE DD1B R         C         mov     si,cs:(offset i_fdu_not_m)      ; drive not ready message.
```

```
E4C8  EB 03                   C          jmp     short i_fdu_end
                              C
E4CA                          C  i_fdu_ok:
E4CA  BE DD1F R               C          mov     si,cs:(offset i_fdu_rdy_m)      ; drive ready message.
                              C
E4CD                          C  i_fdu_end:
E4CD  E8 E5FA R               C          call    DRomString
                              C
                              C  ; Initialize & enable NMI's (parity register).
                              C
E4D0  BA 0061                 C          mov     dx,p_kctrl
E4D3  EC                      C          in      al,dx
E4D4  0C 30                   C          or      al,030h                         ; enable bits #5 & #4
E4D6  EE                      C          out     dx,al
                              C
                              C  ; If alternate processor is available, let user select which one to use.
                              C
E4D7                          C  i_alt_cpu:
E4D7  9C                      C          pushf                                   ; preserve state for int 19h
E4D8  50                      C          push    ax
E4D9  53                      C          push    bx
E4DA  51                      C          push    cx
E4DB  52                      C          push    dx
E4DC  55                      C          push    bp
E4DD  57                      C          push    di
E4DE  56                      C          push    si
E4DF  1E                      C          push    ds
E4E0  06                      C          push    es
                              C
                              C          assume  cs:code, ds:data, es:nothing, ss:nothing
                              C
E4E1  2E: 8E 1E E5F2 R        C          mov     ds,word ptr cs:[set_ds_word]    ; satisfy assumptions
E4E6  A1 0072 R               C          mov     ax,word ptr ds:[reset_flag]     ; get result of keyboard test
E4E9  3C AA                   C          cmp     al,0AAh                         ; is keyboard attached?
E4EB  74 03                   C          je      i_alt_cont                      ; jump if yes (continue)
E4ED  EB 7D 90                C          jmp     i_init_end                      ; jump if no (in mfg)
                              C
E4F0                          C  i_alt_cont:
                              C
E4F0  FA                      C          cli                                     ; disable interupts
                              C
                              C          assume  cs:code, ds:abs0, es:nothing, ss:nothing
                              C
E4F1  33 C0                   C          xor     ax,ax                           ; set up absolute zero address
E4F3  8E D8                   C          mov     ds,ax                           ;    for alternate cpu semaphore
E4F5  8B 16 0000              C          mov     dx,word ptr ds:0                ; save word at 0:0
E4F9  52                      C          push    dx
E4FA  A2 0000                 C          mov     byte ptr ds:0,al                ; request alt cpu id (0:0 = 0)
                              C
E4FD  BA 80C1                 C          mov     dx,80C1h                        ; Z8000 liaison port
E500  B0 01                   C          mov     al,1                            ; AL=1 starts Z8000
E502  EE                      C          out     dx,al                           ; try to boot Z8000
E503  B9 00FF                 C          mov     cx,0FFh                         ; loop counter
                              C
E506                          C  i_alt_test:
E506  80 3E 0000 01           C          cmp     byte ptr ds:0,01h               ; has the semaphore changed?
E50B  74 04                   C          je      i_alt_found                     ; jump if yes (Z8000 id = 1)
E50D  E2 F7                   C          loop    i_alt_test                      ; wait for Z8000 to gain control
E50F  EB 56                   C          jmp short i_alt_end                     ; jump: no alternate cpu
                              C
E511                          C  i_alt_found:
E511  58                      C          pop     ax                              ; restore word at 0:0
E512  A3 0000                 C          mov     word ptr ds:0,ax
E515  50                      C          push    ax                              ; resave for future pop
E516  FB                      C          sti                                     ; enable interupts
E517  E8 E619 R               C          call    DCrLf
E51A  BE DD35 R               C          mov     si,cs:(offset i_alt_select_m)   ; ask user if alt. cpu used
E51D  E8 E5FA R               C          call    DRomString
E520  33 D2                   C          xor     dx,dx                           ; for int 14h
                              C
E522                          C  i_alt_inq:
E522  B4 01                   C          mov     ah,1                            ; has a key been struck?
E524  CD 16                   C          INT     16h
E526  74 FA                   C          jz      i_alt_inq                       ; if not, stay in loop
                              C
E528  32 E4                   C          xor     ah,ah                           ; if so, get the keystroke
E52A  CD 16                   C          INT     16h
E52C  0C 20                   C          or      al,00100000b                    ; force lower case
E52E  3C 79                   C          cmp     al,'y'                          ; did user mean "yes"
E530  74 02                   C          je      i_alt_echo                      ; jump if yes
E532  B0 6E                   C          mov     al,'n'                          ; force "no"
                              C
E534                          C  i_alt_echo:
E534  B4 0E                   C          mov     ah,0Eh                          ; echo "y" or "n"
E536  CD 10                   C          INT     10h
E538  3C 79                   C          cmp     al,'y'                          ; was answer "yes"
E53A  75 2B                   C          jne     i_alt_end                       ; jump if no
E53C  E8 E619 R               C          call    DCrLf
                              C
E53F  FA                      C          cli                                     ; disable interupts
E540  C6 06 0000 0F           C          mov     byte ptr ds:0,0Fh               ; tell Z8000 to take over
                              C
E545                          C  i_alt_restart:                 ; entry for re-entering Z8000 (assume 0:0 setup)
E545  BA 80C1                 C          mov     dx,80C1h                        ; Z8000 liaison port
E548  B0 01                   C          mov     al,1                            ; AL=1 starts Z8000
E54A  EE                      C          out     dx,al                           ; pass control to Z8000
                              C
E54B  B9 00FF                 C          mov     cx,0FFh                         ; loop counter
E54E  E2 FE                   C          loop    $                               ; wait for Z8000 to gain control
                              C
```

```
                                                                        ; we're here only if Z8000 released control
E550  33 C0                     xor     ax,ax                           ; reset segment register
E552  8E D8                     mov     ds,ax                           ;    to absolute zero
E554  80 3E 0000 10             cmp     byte ptr ds:0,10h               ; is a jump requested?
E559  72 0C                     jb      i_alt_end                       ; no, try to boot (or crash)
E55B  8C C8                     mov     ax,cs                           ; pass return registers so that:
E55D  8E C0                     mov     es,ax                           ;       jmp es:[bp]
E55F  BD E2CA R                 mov     bp,offset alt_ret               ;    restarts Z8000 as desired
                                jmpf    0,0                             ; intersegment direct jmp to 0:0
E562  EA                        db      0EAh
E563  0000                      dw      0
E565  0000                      dw      0

                        ; End of Initialization.

E567                    i_alt_end:
E567  58                        pop     ax                              ; restore word at 0:0
E568  A3 0000                   mov     word ptr ds:0,ax
E56B  FB                        sti                                     ; enable interupts

E56C                    i_init_end:
E56C  BA 0378                   mov     dx,0378h                        ; printer port
E56F  B0 80                     mov     al,80h                          ; OK status
E571  EE                        out     dx,al                           ; tell mfg tester

E572  07                        pop     es                              ; restore state for boot
E573  1F                        pop     ds
E574  5E                        pop     si
E575  5F                        pop     di
E576  5D                        pop     bp
E577  5A                        pop     dx
E578  59                        pop     cx
E579  5B                        pop     bx
E57A  58                        pop     ax
E57B  9D                        popf

E57C  E8 E619 R                 call    DCrLf
E57F  CD 19                     INT     19h                             ; go to boot-strap routine

E581                    poinit  endp

                        ; Send command in AH to keyboard interface processor.  AX is used.

E581                    kb_cmd_send proc near

E581                    kb_cmd_wlup:
E581  E4 64                     in      al,kb_status                    ; get 8041 port status
E583  A8 02                     test    al,10b                          ; ready to receive?
E585  75 FA                     jnz     kb_cmd_wlup

E587  8A C4                     mov     al,ah                           ; ready, send command
E589  E6 60                     out     p_kscan,al
E58B  C3                        ret

E58C                    kb_cmd_send endp

E58C                    code    ends
                        include pwrup2.asm

                        ;=============================================================================
                        ;       Filename:       pwrup2.src
                        ;
                        ;       This module includes 8259 Interrupt, Video Controller, 8087
                        ;       NPU, and 8253 & MM58174 Clock tests.
                        ;
                        ;=============================================================================

E58C                    code    segment public 'ROM'
                                assume  cs:code, ds:nothing, es:nothing, ss:nothing

                        ;=============================================================================
                        ;       Note:   We are called from ill_int ONLY (see vector.src), and
                        ;               stack looks like this:
                        ;                                                       High Address
                        ;                               |------------------| <-- sp before ill_int trap
                        ;               (10)            | return fsw flags |
                        ;                               |------------------|
                        ;               (0E)            | return cs segment|
                        ;                               |------------------|
                        ;               (0C)            | return ip offset |
                        ;                               |------------------| <-- sp after ill_int trap
                        ;               (0A)            |        ax        |
                        ;                               |------------------|
                        ;               (08)            |        ds        |
                        ;                               |------------------| <-- sp after ill_int pushes
                        ;               (06)            |  near call here  |
                        ;                               |------------------| <-- sp after ill_int calls ill_trap
                        ;               (04)            |        ax        |
                        ;                               |------------------|
                        ;               (02)            |        dx        |
                        ;                               |------------------|
                        ;               (00)            |        si        |
                        ;                               |------------------| <-- sp after ill_trap pushes
                        ;                                                       Low Address
                        ;=============================================================================

                                assume  cs:code, ds:nothing, es:nothing, ss:nothing

E58C                    ill_trap        proc    near
```

```
                                C  ; Turn off floppy disk drives.
                                C
E58C  50                        C          push    ax                              ; save registers
E58D  52                        C          push    dx
E58E  56                        C          push    si
                                C
E58F  E8 EF4F R                 C          call    stop_disk                       ; destroys ax & dx
                                C
E592  E8 E5C4 R                 C          call    ill_ln
E595  BE DC1B R                 C          mov     si,cs:(offset ill_m1)           ; part 1 of message
E598  E8 E5FA R                 C          call    DRomString
                                C
E59B  8B F4                     C          mov     si,sp
E59D  36: 8E 5C 0E              C          mov     ds,word ptr ss:[si+0Eh]         ; cs past si,dx,ax,ret,ds,ax,ip
E5A1  36: 8B 74 0C              C          mov     si,word ptr ss:[si+0Ch]         ; ip past si,dx,ax,ret,ds,ax
                                C
E5A5  4E                        C          dec     si                              ; si points to interrupt number
E5A6  8A 04                     C          mov     al,byte ptr ds:[si]             ; print illegal interrupt number
E5A8  E8 E643 R                 C          call    DHexByte
E5AB  4E                        C          dec     si                              ; si points to interrupt instr.
E5AC  8B C6                     C          mov     ax,si                           ; save pointer
                                C
E5AE  BE DC34 R                 C          mov     si,cs:(offset ill_m2)           ; part 2 of message
E5B1  E8 E5FA R                 C          call    DRomString
                                C
E5B4  E8 E632 R                 C          call    DHexLong                        ; print illegal cs:ip = ds:ax
                                C
E5B7  BE DC3A R                 C          mov     si,cs:(offset ill_m3)           ; part 3 of message
E5BA  E8 E5FA R                 C          call    DRomString
E5BD  E8 E5C4 R                 C          call    ill_ln
                                C
E5C0  5E                        C          pop     si                              ; restore registers
E5C1  5A                        C          pop     dx
E5C2  58                        C          pop     ax
E5C3  C3                        C          ret
                                C
                                C
E5C4  E8 E619 R                 C  ill_ln: call    DCrLf                           ; prints a line of '*'s
E5C7  B2 2A                     C          mov     dl,'*'
                                C
E5C9  B8 0E2A                   C  ill_lp: mov     ax,(0Eh*100h)+('*')
E5CC  CD 10                     C          INT     10h
E5CE  FE CA                     C          dec     dl
E5D0  75 F7                     C          jnz     ill_lp
E5D2  EB 45 90                  C          jmp     DCrLf
                                C
E5D5                            C  ill_trap        endp
                                C
E5D5                            C  code    ends
                                C  include pwrup3.asm
                                C
                                C  ;=========================================================================
                                C  ;       Filename:       pwrup3.src
                                C  ;
                                C  ;       This module includes 8041 keyboard, communication LSI, RAM, and
                                C  ;       optional ROM tests.
                                C  ;
                                C  ;=========================================================================
                                C
E5D5                            C  code    segment public  'ROM'
                                C
                                C          assume  cs:code, ds:nothing, es:data, ss:nothing
                                C
                                C  ;-------------------------------------------------------------------
                                C  ;       Input:  ds      = segment of ROM under test
                                C  ;               es      = firmware data segment
                                C  ;
                                C  ;       Trash:  All other registers except bx destroyed (in general).
                                C  ;-------------------------------------------------------------------
                                C
E5D5                            C  rom_err         proc    near
                                C                  assume  cs:code, ds:nothing, es:data, ss:nothing
                                C
E5D5  8C D8                     C          mov     ax,ds
E5D7  26: 88 26 0015 R          C          mov     byte ptr es:[mfg_err_flag],ah   ; high byte of ROM address
                                C
E5DC  BE DC45 R                 C          mov     si,cs:(offset fail_m)           ; indicate ROM failed
E5DF  E8 E5FA R                 C          call    DRomString
E5E2  EB 35 90                  C          jmp     DCrLf
                                C
E5E5                            C  rom_err         endp
                                C
                                C
                                C
                                C          assume  cs:code, ds:nothing, es:nothing, ss:nothing
                                C
                                C  ;-------------------------------------------------------------------
                                C  ;       Input:  ds:si   = pointer to ROM to be tested
                                C  ;
                                C  ;       Output: ah      = checksum for the ROM
                                C  ;               cx      = 0
                                C  ;               si      = pointer to byte past ROM
                                C  ;               zf      = state of checksum for the ROM
                                C  ;
                                C  ;       Trash:  al destroyed.
                                C  ;-------------------------------------------------------------------
                                C
E5E5                            C  rom_checksum    proc    near
                                C                  assume  cs:code, ds:nothing, es:nothing, ss:nothing
                                C
```

```
E5E5  B9 2000              C          mov     cx,2000h
                           C
E5E8                       C rom_checksum_cnt:
E5E8  33 C0                C          xor     ax,ax                          ; clear ah
                           C
E5EA                       C rom_checksum_loop:
E5EA  AC                   C          lodsb                                  ; 12 al gets ds:si
E5EB  02 E0                C          add     ah,al                          ;  3
E5ED  E2 FB                C          loop    rom_checksum_loop              ; 17
                           C
E5EF  0A E4                C          or      ah,ah
E5F1  C3                   C          ret
                           C
E5F2                       C rom_checksum     endp
                           C
E5F2                       C code     ends
                           C include pwrup4.asm
                           C
                           C ;=============================================================================
                           C ;       Filename:       pwrup4.src
                           C ;
                           C ;       This module includes disk drive tests, system initialization,
                           C ;       keyboard boot-strap options, and message routines.
                           C ;
                           C ;=============================================================================
                           C
                           C
E5F2                       C code     segment public 'ROM'
                           C          assume  cs:code, ds:nothing, es:nothing, ss:nothing
                           C
                           C ;=============================================================================
                           C ;       Utility Routines:
                           C ;
                           C ;       DRomString      DString         DCrLf           DColon
                           C ;       DHexLong        DHexWord        DHexByte        DHexNib
                           C ;       DNum            DNumW
                           C ;=============================================================================
                           C
E5F2                       C p4_data1         proc    near
                           C
                           C even
                           C
E5F2  0040                 C set_ds_word      dw      data_seg                ; 2 bytes       =  0 clocks
                           C
E5F4                       C p4_data1         endp
                           C
                           C
E5F4                       C set_ds           proc    near            ; set ds to firmware data segment
                           C                  assume  cs:code, ds:nothing, es:nothing, ss:nothing
                           C
E5F4  2E: 8E 1E E5F2 R     C          mov     ds,word ptr cs:[set_ds_word]    ; 5 bytes 2+9+6 = 17 clocks
E5F9  C3                   C          ret                                     ; 1 byte        =  8 clocks
                           C                                                  ; -------       ---------
E5FA                       C set_ds           endp
                           C ;=============================================================================
                           C ;       Display ASCII String Utilities
                           C ;=============================================================================
                           C
E5FA                       C DRomString       proc    near    ; Displays NUL terminated string at cs:si
                           C                  assume  cs:code, ds:nothing, es:nothing, ss:nothing
                           C
E5FA  1E                   C          push    ds              ; all registers saved
E5FB  0E                   C          push    cs              ; ds gets cs
E5FC  1F                   C          pop     ds
E5FD  E8 E602 R            C          call    DString
E600  1F                   C          pop     ds              ; restore ds
E601  C3                   C          ret
E602                       C DRomString       endp
                           C
E602                       C DString          proc    near    ; Displays NUL terminated string at ds:si
                           C                  assume  cs:code, ds:nothing, es:nothing, ss:nothing
                           C
E602  50                   C          push    ax              ; all registers & flags saved
E603  53                   C          push    bx
E604  56                   C          push    si
E605  9C                   C          pushf
E606  FC                   C          cld                     ; auto increment
E607  B3 01                C          mov     bl,1            ; select foreground color for grafix modes
E609  AC                   C DS_lp:   lodsb                   ; al gets ds:si and si++
E60A  0A C0                C          or      al,al           ; NUL ?
E60C  74 06                C          je      DS_ret
E60E  B4 0E                C          mov     ah,0Eh          ; tty emulator
E610  CD 10                C          INT     10h
E612  EB F5                C          jmp     short DS_lp
E614                       C DS_ret:
E614  9D                   C          popf                    ; restore registers & flags
E615  5E                   C          pop     si
E616  5B                   C          pop     bx
E617  58                   C          pop     ax
E618  C3                   C          ret
E619                       C DString          endp
                           C
E619                       C DCrLf            proc    near    ; Displays a CR & LF.
                           C                  assume  cs:code, ds:nothing, es:nothing, ss:nothing
E619  50                   C          push    ax              ; all registers preserved
E61A  B8 0E0D              C          mov     ax,(0Eh*100h)+CR
E61D  CD 10                C          INT     10h             ; tty emulator
E61F  B8 0E0A              C          mov     ax,(0Eh*100h)+LF
E622  CD 10                C          INT     10h             ; tty emulator
E624  58                   C          pop     ax              ; restore ax
```

```
E625  C3                  C              ret
E626                      C  DCrLf       endp
                          C
E626                      C  DColon      proc    near    ; Displays a ':'.
                          C              assume  cs:code, ds:nothing, es:nothing, ss:nothing
E626  50                  C              push    ax              ; all registers preserved
E627  53                  C              push    bx
E628  B3 01               C              mov     bl,1            ; select foreground color for grafix modes
E62A  B8 0E3A             C              mov     ax,(0Eh*100h)+':'
E62D  CD 10               C              INT     10h             ; tty emulator
E62F  5B                  C              pop     bx              ; restore registers
E630  58                  C              pop     ax
E631  C3                  C              ret
E632                      C  DColon      endp
                          C
                          C  ;====================================   ==============================
                          C  ;       Display Hexadecimal Number in   CII Utilities
                          C  ;====================================  ==============================
                          C
E632                      C  DHexLong    proc    near    ; Displays ds:ax in ASCII
                          C              assume  cs:code, ds:nothing, es:nothing, ss:nothing
E632  50                  C              push    ax              ; all registers preserved
E633  8C D8               C              mov     ax,ds           ; display segment first
E635  E8 E63C R           C              call    DHexWord
                          C
E638  E8 E626 R           C              call    DColon          ; display a colon
                          C
E63B  58                  C              pop     ax              ; restore ax
                          C  ;           jmp     short DHexWord  ; fall through:  display offset second
                          C
E63C                      C  DHexLong    endp
                          C
E63C                      C  DHexWord    proc    near    ; Displays ax in ASCII
                          C              assume  cs:code, ds:nothing, es:nothing, ss:nothing
E63C  50                  C              push    ax              ; all registers preserved
E63D  8A C4               C              mov     al,ah
E63F  E8 E643 R           C              call    DHexByte        ; display high byte first
E642  58                  C              pop     ax              ; restore ax
                          C  ;           jmp     short DHexByte  ; fall through:  display low byte second
                          C
E643                      C  DHexWord    endp
                          C
E643                      C  DHexByte    proc    near    ; Displays al in ASCII
                          C              assume  cs:code, ds:nothing, es:nothing, ss:nothing
E643  50                  C              push    ax              ; all registers preserved
E644  D0 C8               C              ror     al,1
E646  D0 C8               C              ror     al,1
E648  D0 C8               C              ror     al,1
E64A  D0 C8               C              ror     al,1            ; move high nibble to low nibble
E64C  E8 E650 R           C              call    DHexNib         ; display high nibble in ASCII
E64F  58                  C              pop     ax              ; restore ax
                          C  ;           jmp     short DHexNib   ; fall through:  display low nibble in ASCII
                          C
E650                      C  DHexByte    endp
                          C
E650                      C  DHexNib     proc    near    ; Displays low nibble of al in ASCII
                          C              assume  cs:code, ds:nothing, es:nothing, ss:nothing
E650  50                  C              push    ax              ; all registers preserved
E651  53                  C              push    bx
E652  B3 01               C              mov     bl,1            ; select foreground color for grafix modes
E654  24 0F               C              and     al,0fh          ; clear high nibble
E656  04 30               C              add     al,'0'
E658  3C 39               C              cmp     al,'9'
E65A  76 02               C              jbe     NibOk           ; '0' <= al <= '9' ?
E65C  04 07               C              add     al,'A'-'0'-10
E65E  B4 0E               C  NibOk:      mov     ah,0Eh          ; tty emulator
E660  CD 10               C              INT     10h
E662  5B                  C              pop     bx              ; restore registers
E663  58                  C              pop     ax
E664  C3                  C              ret
E665                      C  DHexNib     endp
                          C
                          C  ;=========================================================================
                          C  ;       Display Decimal Number in ASCII Utilities
                          C  ;=========================================================================
                          C
E665                      C  DNum        proc    near    ; Displays decimal of ax in ASCII in min width
                          C              assume  cs:code, ds:nothing, es:nothing, ss:nothing
E665  53                  C              push    bx              ; all registers preserved
E666  33 DB               C              xor     bx,bx           ; minimum width
E668  E8 E66D R           C              call    DNumW           ; display ax
E66B  5B                  C              pop     bx              ; restore bx
E66C  C3                  C              ret
E66D                      C  DNum        endp
                          C
E66D                      C  DNumW       proc    near    ; Displays decimal of ax in ASCII of width bx
                          C              assume  cs:code, ds:nothing, es:nothing, ss:nothing
                          C
E66D  50                  C              push    ax                      ; all registers preserved
E66E  53                  C              push    bx
E66F  51                  C              push    cx
E670  52                  C              push    dx
E671  56                  C              push    si
                          C
E672  BE 000A             C              mov     si,10                   ; decimal modulus
E675  33 C9               C              xor     cx,cx                   ; clear digit counter
E677                      C  DNumW_loop:
E677  33 D2               C              xor     dx,dx                   ; dx:ax = decimal number
E679  F7 F6               C              div     si                      ; dh = 0
                          C                                              ; dl = remainder = (0-9)
```

```
                        C
                        C                               ; ax = quotient  = higher order digits
E67B  52                C               push    dx      ; save the digit on stack
E67C  41                C               inc     cx      ; increment count of what's on stack
E67D  0B C0             C               or      ax,ax   ; are we done?
E67F  75 F6             C               jnz     DNumW_loop
                        C
E681  2B D9             C               sub     bx,cx   ; subtract digit count from width
E683  76 08             C               jbe     DNumW_skip      ; skip spaces if bx is not > cx
                        C
E685                    C DNumW_spaces:
E685  B8 0E20           C               mov     ax,(0Eh*100h)+' '       ; display a space
E688  CD 10             C               INT     10h
E68A  4B                C               dec     bx      ; decrement count of spaces
E68B  75 F8             C               jnz     DNumW_spaces    ; keep going
                        C
E68D                    C DNumW_skip:
E68D  B3 01             C               mov     bl,1    ; foreground color for grafix modes
E68F  58                C               pop     ax      ; remove digit from stack
E690  04 30             C               add     al,'0'  ; convert to ASCII
E692  B4 0E             C               mov     ah,0Eh  ; display the digit
E694  CD 10             C               INT     10h
E696  E2 F5             C               loop    DNumW_skip
                        C
E698  5E                C               pop     si      ; restore registers
E699  5A                C               pop     dx
E69A  59                C               pop     cx
E69B  5B                C               pop     bx
E69C  58                C               pop     ax
E69D  C3                C               ret
E69E                    C DNumW         endp
                        C
E69E                    C code          ends

                        C include boot1.asm
                        C
                        C ;=============================================================================
                        C ;       Filename:       boot1.src
                        C ;
                        C ;       This module includes the ORG'd jump to INT 19h (boot2.src)
                        C ;
                        C ;=============================================================================
                        C
E69E                    C code    segment public 'ROM'
                        C         assume  cs:code, ds:nothing, es:nothing, ss:nothing
                        C
E6F2                    C         ORG     0E6F2h
                        C
E6F2                    C bt_jmp  proc    near
                        C
E6F2  E9 F860 R         C         jmp     bt_int                ; necessary jump for ORG
                        C
E6F5                    C bt_jmp  endp
                        C
E6F5                    C code ends
                        C include com1.asm
                        C
                        C ;=============================================================================
                        C ;       Filename:       com1.src
                        C ;
                        C ;       This module, com2, and com3 supply INT 14h.
                        C ;
                        C ;=============================================================================
                        C
E6F5                    C code    segment public 'ROM'
                        C         assume  cs:code, ds:nothing, es:nothing, ss:nothing
                        C
                        C
                        C ;-----------------------------------------------------------------------
                        C ;       INS8250 Compatible Line Status Bits (ah) for Z8530 SCC Re-Mapping
                        C ;-----------------------------------------------------------------------
                        C
= 0080                  C com_te                  equ     80h             ; time out error (bit #7)
= 0020                  C com_txd                 equ     20h             ; transmit ready (bit #5)
= 0008                  C com_fe                  equ     08h             ; framing  error (bit #3)
= 0004                  C com_pe                  equ     04h             ; parity   error (bit #2)
= 0002                  C com_oe                  equ     02h             ; overrun  error (bit #1)
= 0001                  C com_rxd                 equ     01h             ; receive  ready (bit #0)
                        C
                        C ;-----------------------------------------------------------------------
                        C ;       INS8250 Compatible Modem Status Bits (al) for Z8530 SCC Re-Mapping
                        C ;-----------------------------------------------------------------------
                        C
= 0020                  C com_dsr         equ     20h                     ; data set ready (bit #5)
= 0010                  C com_cts         equ     10h                     ; clear to send  (bit #4)
                        C
                        C ;-----------------------------------------------------------------------
                        C ;       INS8250 Compatible Modem Control Bits.
                        C ;-----------------------------------------------------------------------
                        C
= 0002                  C com_rts         equ     02h                     ; request to send     (bit #1)
= 0001                  C com_dtr         equ     01h                     ; data terminal ready (bit #0)
                        C
                        C ;-----------------------------------------------------------------------
                        C ;       Z8530 SCC Status Register (Read Register #0)
                        C ;-----------------------------------------------------------------------
                        C
= 0004                  C scc_txd                 equ     04h             ; transmit ready (bit #2)
= 0001                  C scc_rxd                 equ     01h             ; receive  ready (bit #0)
                        C
                        C ;-----------------------------------------------------------------------
                        C ;       Z8530 SCC Error Register (Read Register #1)
```

```
                        C  ;--------------------------------------------------------------------
                        C
= 0040                  C  scc_fe               equ     40h             ; framing  error (bit #6)
= 0020                  C  scc_oe               equ     20h             ; overrun  error (bit #5)
= 0010                  C  scc_pe               equ     10h             ; parity   error (bit #4)
                        C
                        C  ;--------------------------------------------------------------------
                        C  ;       INS8250 Asynchronous Communication Chip Baud Rate Time Constants
                        C  ;       (baud rate generator signal is 3.6864 MHz put through a
                        C  ;       divide-by-2 circuit.
                        C  ;
                        C  ;                       ((3,686,400 Hz)/2) = Input Freq.
                        C  ;       Time Constant = ---------------------------------
                        C  ;                               (16)*(baud rate)
                        C  ;--------------------------------------------------------------------
                        C
E729                    C                       ORG     0E729h
                        C
E729                    C  com_data1            proc
                        C
E729  0417              C  com_baud             dw      1047    ;   110 baud    (0)
E72B  0300              C                       dw      768     ;   150 baud    (1)
E72D  0180              C                       dw      384     ;   300 baud    (2)
E72F  00C0              C                       dw      192     ;   600 baud    (3)
E731  0060              C                       dw      96      ;  1200 baud    (4)
E733  0030              C                       dw      48      ;  2400 baud    (5)
E735  0018              C                       dw      24      ;  4800 baud    (6)
E737  000C              C                       dw      12      ;  9600 baud    (7)
                        C  .list
                        C
E739                    C  com_data1            endp
                        C
                        C  ;--------------------------------------------------------------------
                        C  ;       Z8530 Serial Communication Controller Baud Rate Time Constants
                        C  ;           (baud rate generator signal is 3.6864 MHz)
                        C  ;           (NO divide-by-2 circuit!!!!)
                        C  ;
                        C  ;                       (3,686,400 Hz) = Input Freq.
                        C  ;       Time Constant = --------------------------  - 2
                        C  ;                           (16)*(2)*(baud rate)
                        C  ;
                        C  ;       NOTE: These values are the SAME as the above EXCEPT for the - 2!!!!
                        C  ;--------------------------------------------------------------------
                        C  ;
                        C  ;scc_baud            dw      1045    ;   110 baud    (0)
                        C  ;                    dw      766     ;   150 baud    (1)
                        C  ;                    dw      382     ;   300 baud    (2)
                        C  ;                    dw      190     ;   600 baud    (3)
                        C  ;                    dw      94      ;  1200 baud    (4)
                        C  ;                    dw      46      ;  2400 baud    (5)
                        C  ;                    dw      22      ;  4800 baud    (6)
                        C  ;                    dw      10      ;  9600 baud    (7)
                        C  .list
                        C
                        C  ;====================================================================
                        C  ;       INT 14h -- RS-232 Software Interrupt Request Routine
                        C  ;
                        C  ;       Assumes:        INS8250 port addresses are > 256.  That is, the
                        C  ;                       high byte of the port address is nonzero, if and
                        C  ;                       only if, the port is a INS8250. (e.g. com_a ports
                        C  ;                       are 03F8h - 03FFh & com_b ports are 02F8h - 02FFh.)
                        C  ;
                        C  ;       Similarly:      Z8530 port addresses are < 256.  That is, the
                        C  ;                       high byte of the port address is zero, if and
                        C  ;                       only if, the port is a Z8530. (e.g. scc_a ports
                        C  ;                       are 0050h - 0051h & scc_b ports are 0052h - 0053h.)
                        C  ;
                        C  ;       Z8530 Note:     For the reset during power-up, DTR and RTS must be
                        C  ;                       set low which is the only difference from a normal
                        C  ;                       reset (AH=0).  This is accomplished by a special
                        C  ;                       function code (AH=0FFh).
                        C  ;====================================================================
                        C
E739                    C          ORG     0E739h
                        C
E739                    C  serial_io            proc    near
                        C          assume  cs:code, ds:nothing, es:nothing, ss:nothing
                        C
E739  FB                C          sti                                     ; enable interrupts
                        C
E73A  55                C          push    bp                              ; save register
                        C
E73B  83 FA 04          C          cmp     dx,4                            ; 4 RS-232 channels allowed max
E73E  73 3D             C          jae     rs_nop
                        C
E740  8B E8             C          mov     bp,ax                           ; save original function code
E742  80 FC FF          C          cmp     ah,0FFh                         ; power-up reset?
E745  75 02             C          jne     rs_norm                         ; jump if no
E747  32 E4             C          xor     ah,ah                           ; same as reset, BP remembers FF
                        C
E749                    C  rs_norm:
E749  80 FC 03          C          cmp     ah,03h                          ; input out of range?
E74C  77 2F             C          ja      rs_nop
                        C
                        C                  assume  cs:code, ds:data, es:nothing, ss:nothing
                        C
E74E  52                C          push    dx                              ; save registers
E74F  51                C          push    cx
E750  53                C          push    bx
E751  1E                C          push    ds                              ; save ds
```

```
E752  2E: 8E 1E E5F2 R      C          mov     ds,word ptr cs:[set_ds_word]    ; satisfy assumptions
                            C
E757  8B DA                 C          mov     bx,dx                                   ; get port number (0-3)
E759  33 C9                 C          xor     cx,cx                                   ; clear ch
E75B  8A 8F 007C R          C          mov     cl,byte ptr ds:[bx+serial_t_out]        ; get RS-232 time-out
E75F  D1 E3                 C          shl     bx,1                                    ; make word index
E761  8B 97 0000 R          C          mov     dx,word ptr ds:[bx+rs232_addr]          ; get address of RS-232
                            C                                                          ; data port
E765  1F                    C          pop     ds                                      ; restore ds
                            C
                            C                  assume  cs:code, ds:nothing, es:nothing, ss:nothing
                            C
E766  0B D2                 C          or      dx,dx                           ; RS-232 port present?
E768  74 10                 C          jz      rs_ret                          ; if not, leave
                            C
E76A  0A F6                 C          or      dh,dh                           ; are we a INS8250 chip?
E76C  75 03                 C          jnz     rs_ok                           ; if so, take jump
                            C
E76E  80 C4 04              C          add     ah,4                            ; if SCC Z8530 add 4 to function
                            C
E771                        C rs_ok:
E771  8A DC                 C          mov     bl,ah                           ; bx = function number
E773  D1 E3                 C          shl     bx,1                            ; bx = 2*(function number)
E775  2E: FF 97 E77F R      C          call    cs:[bx+(offset rs_tbl)]         ; perform rs232 function
                            C
E77A                        C rs_ret:
E77A  5B                    C          pop     bx                              ; restore registers
E77B  59                    C          pop     cx
E77C  5A                    C          pop     dx
E77D                        C rs_nop:
E77D  5D                    C          pop     bp
E77E  CF                    C          iret
                            C
                            C ;--------------------------------------------------------------------
                            C ;       INT 14h Jump Table
                            C ;--------------------------------------------------------------------
                            C
E77F  E78F R                C rs_tbl   dw      com_init        ; ah = 00h for INS8250
E781  E8EC R                C          dw      com_pb          ; ah = 01h for INS8250
E783  E92A R                C          dw      com_gb          ; ah = 02h for INS8250
E785  E87F R                C          dw      com_stat        ; ah = 03h for INS8250
E787  F573 R                C          dw      scc_init        ; ah = 00h for SCC Z8530
E789  E91B R                C          dw      scc_pb          ; ah = 01h for SCC Z8530
E78B  E94D R                C          dw      scc_gb          ; ah = 02h for SCC Z8530
E78D  E88A R                C          dw      scc_stat        ; ah = 03h for SCC Z8530
                            C
E78F                        C serial_io        endp
                            C
                            C
                            C ;--------------------------------------------------------------------
                            C ;       Initialize RS-232 Interface.
                            C ;
                            C ;       Input:  al =    input parameters
                            C ;               dx =    address of RS-232 channel
                            C ;       Output: ax =    RS-232 channel status
                            C ;
                            C ;               al initializes port with: bit # 7 6 5 4 3 2 1 0
                            C ;                                              +-+-+-+-+-+-+-+-+
                            C ;                                              |B|B|B|P|P|S|D|D|
                            C ;                                              +-+-+-+-+-+-+-+-+
                            C ;       Baud (BBB):             Parity (PP):    Stop Bits (S):  Data Bits (DD):
                            C ;       0 = 110  4 = 1200       x0 = None       0 = 1           10 = 7
                            C ;       1 = 150  5 = 2400       01 = Odd        1 = 2           11 = 8
                            C ;       2 = 300  6 = 4800       11 = Even                       (00 = 5?)
                            C ;       3 = 600  7 = 9600                                       (01 = 6?)
                            C ;
                            C ;       Assumes:        com_int_x  = com_data_x + 1 = dx + 1
                            C ;                       com_lctl_x = com_data_x + 3 = dx + 3
                            C ;--------------------------------------------------------------------
                            C
E78F                        C com_init         proc    near                    ; ah = 00h
                            C                  assume  cs:code, ds:nothing, es:nothing, ss:nothing
                            C
E78F  52                    C          push    dx                              ; save dx = com_data_x
                            C
E790  8A E8                 C          mov     ch,al                           ; save input parameters.
                            C
E792  B0 80                 C          mov     al,080h                         ; access divisor latch of
                            C                                                  ; baud count register.
E794  83 C2 03              C          add     dx,3                            ; dx = com_lctl_x
E797  EE                    C          out     dx,al                           ; write to line control register
E798  E8 E8E2 R             C          call    rs_dly
                            C
E79B  8A DD                 C          mov     bl,ch                           ; get input parameters.
E79D  81 E3 00E0            C          and     bx,111100000b                   ; get bits #5, 6, & 7 (clear bh)
E7A1  B1 04                 C          mov     cl,4
E7A3  D2 EB                 C          shr     bl,cl                           ; move to bits #1,2,& 3
                            C                                                  ; bx is word index
E7A5  2E: 8B 87 E729 R      C          mov     ax,word ptr cs:[bx+com_baud]    ; get 8250 baud count
                            C
E7AA  5A                    C          pop     dx                              ; restore dx = com_data_x
E7AB  EE                    C          out     dx,al                           ; output low byte of baud rate
E7AC  E8 E8E2 R             C          call    rs_dly
                            C
E7AF  8A C4                 C          mov     al,ah                           ; transfer high byte to low byte
E7B1  42                    C          inc     dx                              ; dx = com_int_x
E7B2  EE                    C          out     dx,al                           ; output high byte of baud rate
E7B3  E8 E8E2 R             C          call    rs_dly
                            C
E7B6  8A C5                 C          mov     al,ch                           ; get input parameters.
```

```
E7B8  24 1F                C          and     al,00011111b                    ; get bits #0 thru #4
E7BA  42                   C          inc     dx
E7BB  42                   C          inc     dx                              ; dx = com_lotl_x
E7BC  EE                   C          out     dx,al                           ; write to line control register
E7BD  E8 E8E2 R            C          call    rs_dly                          ; disable access divisor latch,
                           C                                                  ; set data & stop bits, & parity
                           C
E7C0  32 C0                C          xor     al,al                           ; disable all interrupts!!
E7C2  4A                   C          dec     dx
E7C3  4A                   C          dec     dx                              ; dx = com_int_x
E7C4  EE                   C          out     dx,al                           ; write to interrupt ID register
                           C
E7C5  4A                   C          dec     dx                              ; dx = com_data_x
E7C6  E9 E87F R            C          jmp     com_stat                        ; return status
                           C
E7C9                       C  com_init        endp
                           C
E7C9                       C  code    ends
                           C  include kb1.asm
                           C
                           C  ;=============================================================================
                           C  ;       Filename:       kb1.src
                           C  ;
                           C  ;       This module includes INT 09h & 16h.
                           C  ;
                           C  ;=============================================================================
                           C
E7C9                       C  code    segment public 'ROM'
                           C          assume  cs:code, ds:nothing, es:nothing, ss:nothing
                           C
                           C  ;=============================================================================
                           C  ;       INT 16h -- i8041A Keyboard Software Interrupt Request Routine
                           C  ;=============================================================================
                           C  ;
                           C  ;       Input:  ah      = function number (00h <= ah <= 03h)
                           C  ;       Output: ah      = (ah - 2) if ah >= 2
                           C  ;
                           C  ;       Trash:  None. (bx & ds if ROM stack)
                           C  ;
                           C  ;       Note:   The stack never gets deeper than 6 words!!!
                           C  ;                                                  High Address
                           C  ;                               |------------------| <-- sp (at entry & exit)
                           C  ;                               | return fpw flags |
                           C  ;                               |------------------|
                           C  ;                               | return cs segment|
                           C  ;                               |------------------|
                           C  ;                               | return ip offset |
                           C  ;                               |------------------| <-- sp after kb trap
                           C  ;                               |        ds        |
                           C  ;                               |------------------|
                           C  ;                               |        bx        |
                           C  ;                               |------------------|
                           C  ;                               |    1 near call   |
                           C  ;                               |------------------| <-- sp (at its deepest!!)
                           C  ;                                                     Low Address
                           C  ;
                           C  ;=============================================================================
                           C
E82E                       C          ORG     0E82Eh
                           C
E82E                       C  k_io    proc    near
                           C          assume  cs:code, ds:nothing, es:nothing, ss:nothing
                           C
E82E  FB                   C          sti                                     ; enable interrupts!!
E82F  1E                   C          push    ds                              ; save ds
E830  2E: 8E 1E E5F2 R     C          mov     ds,word ptr cs:[set_ds_word]    ; avoid potential stack problems
E835  53                   C          push    bx                              ; save bx
                           C
                           C          assume  cs:code, ds:data, es:nothing, ss:nothing
                           C
E836  80 FC 01             C          cmp     ah,1                            ; ah <= 1 ?
E839  72 0A                C          jb      k_read                          ; jump if ah = 0
E83B  74 23                C          je      k_look                          ; jump if ah = 1
E83D  80 FC 02             C          cmp     ah,2                            ; ah=2 ?
E840  74 27                C          je      k_stat
                           C
E842  5B                   C  k_ret:  pop     bx                              ; restore registers
E843  1F                   C          pop     ds
E844  CF                   C          iret
                           C
E845                       C  k_io    endp
                           C
                           C  ;-----------------------------------------------------------------------
                           C  ;       Wait for key and extract if from the keyboard buffer.
                           C  ;
                           C  ;       Output: ah = raw scan code
                           C  ;               al = ASCII translated key
                           C  ;                    or
                           C  ;               ah =    translated key
                           C  ;               al = 00h
                           C  ;
                           C  ;       Trash:  None.
                           C  ;-----------------------------------------------------------------------
                           C
E845                       C  k_read  proc    near
                           C          assume  cs:code, ds:data, es:nothing, ss:nothing
                           C
E845  FB                   C          sti                                     ; enable interrupts (again)
E846  E8 E854 R            C          call    k_see                           ; is there a character present?
```

```
                                C
E849  74 FA                     C           jz      k_read                          ; interrupts come back disabled!
E84B  E8 E86E R                 C           call    k_adv_ptr                       ; loop until something in buffer
E84E  89 1E 001A R              C           mov     word ptr ds:[buffer_head],bx    ; move pointer to next position
E852  EB EE                     C           jmp     short k_ret                     ; store value in variable
                                C
                                C
E854  FA                        C k_see:    cli                                     ; disable interrupts!!
E855  8B 1E 001A R              C           mov     bx,word ptr ds:[buffer_head]    ; get pointer to head of buffer
E859  3B 1E 001C R              C           cmp     bx,word ptr ds:[buffer_tail]    ; if equal, then nothing there
E85D  8B 07                     C           mov     ax,word ptr ds:[bx]             ; get scan code and ascii code
E85F  C3                        C           ret
                                C
E860                            C k_read    endp
                                C
                                C ;--------------------------------------------------------------------------
                                C ;       Checks for key in keyboard buffer, but does not extract it.
                                C ;
                                C ;       Output: if key is in buffer, then:
                                C ;                       zf = 0 (nz = reset)
                                C ;                       ah = raw scan code
                                C ;                       al = ASCII translated key
                                C ;                            or
                                C ;                       ah =   translated key
                                C ;                       al = 00h
                                C ;               else:
                                C ;                       zf = 1 (z = set)
                                C ;                       ax = 16th previous key
                                C ;
                                C ;       Trash:  ax is trashed if keyboard buffer is empty.
                                C ;--------------------------------------------------------------------------
                                C
E860                            C k_look    proc    far     ; must be far!!!!
                                C           assume  cs:code, ds:data, es:nothing, ss:nothing
                                C
E860  E8 E854 R                 C           call    k_see                           ; is there a character present?
                                C                                                   ; interrupts come back disabled!
E863  FB                        C           sti                                     ; must return interrupts enabled
E864  5B                        C           pop     bx                              ; restore registers
E865  1F                        C           pop     ds                              ; blow away flags returning:
E866  CA 0002                   C           ret     2                               ; zf & ax = k_see output, & sti
                                C
E869                            C k_look    endp
                                C
                                C ;--------------------------------------------------------------------------
                                C ;       Returns keyboard shift state kb_flag in al.
                                C ;
                                C ;       Output: ah = 0.
                                C ;               al = kb_flag
                                C ;       Trash:  None.
                                C ;--------------------------------------------------------------------------
                                C
E869                            C k_stat    proc    near
                                C           assume  cs:code, ds:data, es:nothing, ss:nothing
                                C
E869  A0 0017 R                 C           mov     al,byte ptr ds:[kb_flag]        ; get the shift status flags
E86C  EB D4                     C           jmp     short k_ret
                                C
E86E                            C k_stat    endp
                                C
                                C ;--------------------------------------------------------------------------
                                C ;       Advances kb_buffer ring buffer pointer.
                                C ;
                                C ;       Input:  bx
                                C ;       Output: bx
                                C ;       Trash:  None.
                                C ;--------------------------------------------------------------------------
                                C
E86E                            C k_adv_ptr         proc    near
                                C                   assume  cs:code, ds:data, es:nothing, ss:nothing
                                C
E86E  43                        C           inc     bx                              ; move to next word in list
E86F  43                        C           inc     bx
E870  3B 1E 0082 R              C           cmp     bx,word ptr ds:[buffer_end]     ; end of buffer ?
E874  75 04                     C           jne     k_adv_end                       ; no, continue
E876  8B 1E 0080 R              C           mov     bx,word ptr ds:[buffer_start]   ; yes, buffer to beginning
E87A                            C k_adv_end:
E87A  C3                        C           ret
                                C
E87B                            C k_adv_ptr         endp
                                C
E87B                            C code      ends

                                C include com2.asm
                                C
                                C ;==========================================================================
                                C ;       Filename:       com2.src
                                C ;
                                C ;==========================================================================
                                C
E87B                            C code      segment public 'ROM'
                                C           assume  cs:code, ds:nothing, es:nothing, ss:nothing
                                C
                                C ;--------------------------------------------------------------------------
                                C ;       Read Status of RS-232 Interface.  (rs_stat)
                                C ;
                                C ;       Input:  dx =    if dh =  0, then address of Z8530 channel (scc_ctl_x).
                                C ;                       if dh <> 0, then address of 8250 data port (com_data_x).
                                C ;       Output: ax =    RS-232 (INS8250-compatible) channel status.
                                C ;       Trash:  None.
```

```
                        C ;
                        C ;       Assumes:        com_lstat_x = com_data_x + 5 = dx + 5 (line status)
                        C ;                       com_mstat_x = com_data_x + 6 = dx + 6 (modem status)
                        C ;---------------------------------------------------------------------
                        C
E87B                    C rs_stat proc    near                    ; ah = 03h
                        C         assume  cs:code, ds:nothing, es:nothing, ss:nothing
                        C
E87B  0A F6             C         or      dh,dh                   ; are we a SCC Z8530 chip?
E87D  74 0B             C         jz      scc_stat                ; if so, take jump
                        C
                        C
E87F                    C com_stat:                               ; INS8250 read status routine.
                        C
                        C ; Get Line Status.
                        C
E87F  52                C         push    dx                      ; save dx = com_data_x
E880  83 C2 05          C         add     dx,5                    ; dx = com_lstat_x
E883  EC                C         in      al,dx                   ; get line status
E884  8A E0             C         mov     ah,al                   ; line status comes back in high byte
                        C
                        C ; Get Modem Status.
                        C
E886  42                C         inc     dx                      ; dx = com_mstat_x
E887  EC                C         in      al,dx                   ; get modem status in low byte
E888  5A                C         pop     dx                      ; restore dx = com_data_x
E889  C3                C         ret
                        C
                        C
E88A                    C scc_stat:                               ; SCC Z8530 read status routine.
                        C
                        C ; Get Channel Status.
                        C
E88A  33 C0             C         xor     ax,ax                   ; al = selects 0 ; ah = no errors
E88C  EE                C         out     dx,al                   ; dx = scc_ctl_x
E88D  E8 E8E2 R         C         call    rs_dly
E890  EC                C         in      al,dx                   ; get channel status
                        C
E891  A8 04             C         test    al,scc_txd              ; test for transmit ready
E893  74 03             C         jz      scc_no_txd
                        C
E895  80 CC 20          C         or      ah,com_txd              ; if so, set INS8250-compatible bit.
E898                    C scc_no_txd:
                        C
E898  A8 01             C         test    al,scc_rxd              ; test for receive ready
E89A  74 03             C         jz      scc_no_rxd
                        C
E89C  80 CC 01          C         or      ah,com_rxd              ; if so, set INS8250-compatible bit.
E89F                    C scc_no_rxd:
                        C
                        C ; Get Error Status.
                        C
E89F  B0 01             C         mov     al,1                    ; get errors
E8A1  EE                C         out     dx,al                   ; dx = scc_ctl_x
E8A2  E8 E8E2 R         C         call    rs_dly
E8A5  EC                C         in      al,dx                   ; get error status
                        C
E8A6  A8 40             C         test    al,scc_fe               ; test for framing error
E8A8  74 03             C         jz      scc_no_fe
                        C
E8AA  80 CC 08          C         or      ah,com_fe               ; if so, set INS8250-compatible bit.
E8AD                    C scc_no_fe:
                        C
E8AD  A8 10             C         test    al,scc_pe               ; test for parity error
E8AF  74 03             C         jz      scc_no_pe
                        C
E8B1  80 CC 04          C         or      ah,com_pe               ; if so, set INS8250-compatible bit.
E8B4                    C scc_no_pe:
                        C
E8B4  A8 20             C         test    al,scc_oe               ; test for overrun error
E8B6  74 03             C         jz      scc_no_oe
                        C
E8B8  80 CC 02          C         or      ah,com_oe               ; if so, set INS8250-compatible bit.
E8BB                    C scc_no_oe:
                        C
                        C ; Set Modem Status.
                        C
E8BB  B0 30             C         mov     al,(com_dsr+com_cts)    ; al = DSR and CTS
E8BD  C3                C         ret
                        C
E8BE                    C rs_stat endp
                        C
                        C ;---------------------------------------------------------------------
                        C ;       Wait for Status of RS-232 Interface.  (rs_ws)
                        C ;
                        C ;       Input:  ah =    RS-232 channel status for which to wait
                        C ;               cx =    RS-232 time-out
                        C ;               dx =    if dh =  0, then address of Z8530 channel (scc_ctl_x).
                        C ;                       if dh <> 0, then address of 8250 data port to poll.
                        C ;
                        C ;       Output: AH =    Status.
                        C ;               ZF =    set,   if status matches.
                        C ;                       reset, if time-out.
                        C ;
                        C ;       Trash:  al & bx destroyed.
                        C ;
                        C ;       Assumes:        com_lstat_x = com_data_x + 5 = dx + 5 (line status)
                        C ;                       com_mstat_x = com_data_x + 6 = dx + 6 (modem status)
                        C ;---------------------------------------------------------------------
                        C
```

```
E8BE                    C  rs_ws    proc    near
                        C          assume  cs:code, ds:nothing, es:nothing, ss:nothing
                        C
E8BE  51                C          push    cx                      ; save time-out
E8BF  33 DB             C          xor     bx,bx                   ; clear bx
E8C1  87 CB             C          xchg    cx,bx                   ; BL now has rs232_time_out.
                        C
E8C3                    C  rs_ws_lp:
E8C3  0A F6             C          or      dh,dh                   ; are we an INS8250 chip?
E8C5  75 06             C          jnz     rs_ws_com               ; if so, take jump
                        C
E8C7  32 C0             C          xor     al,al                   ; al = selects 0 on SCC Z8530
E8C9  EE                C          out     dx,al                   ; dx = scc_ctl_x
E8CA  E8 E8E2 R         C          call    rs_dly
                        C
E8CD                    C  rs_ws_com:
E8CD  EC                C          in      al,dx                   ; get channel status
E8CE  8A F8             C          mov     bh,al                   ; save status in BH.
E8D0  22 C4             C          and     al,ah                   ; mask bits we're waiting for
E8D2  3A C4             C          cmp     al,ah                   ; are they all on?
                        C
E8D4  74 08             C          jz      rs_ws_exit              ; if so, exit with zf set
                        C
E8D6  E2 EB             C          loop    rs_ws_lp                ; inner loop
E8D8  FE CB             C          dec     bl
E8DA  75 E7             C          jnz     rs_ws_lp                ; outer loop
                        C
E8DC  0A E4             C          or      ah,ah                   ; time-out, exit with zf reset
E8DE                    C  rs_ws_exit:
E8DE  8A E7             C          mov     ah,bh                   ; move status to AH.
E8E0  59                C          pop     cx                      ; restore time-out
E8E1  C3                C          ret
                        C
E8E2                    C  rs_ws    endp
                        C
                        C
E8E2                    C  rs_dly   proc    near
                        C          assume  cs:code, ds:nothing, es:nothing, ss:nothing
                        C
E8E2  9C                C          pushf
E8E3  51                C          push    cx
E8E4  B9 0008           C          mov     cx,8
E8E7  E2 FE             C  rs_lp:  loop    rs_lp
E8E9  59                C          pop     cx
E8EA  9D                C          popf
E8EB  C3                C          ret
                        C
E8EC                    C  rs_dly   endp
                        C
                        C          assume  cs:code, ds:nothing, es:nothing, ss:nothing
                        C
                        C
                        C  ;---------------------------------------------------------------------------
                        C  ;       INS8250 Put Byte (com_pb) & SCC Z8530 Put Byte (scc_pb)
                        C  ;
                        C  ;       Transmit Character to RS-232 Interface.
                        C  ;
                        C  ;       Input:  al =    character to transmit
                        C  ;               cx =    RS-232 time-out
                        C  ;               dx =    if dh =  0, then address of Z8530 channel (scc_ctl_x).
                        C  ;                       if dh <> 0, then address of 8250 data port (com_data_x).
                        C  ;       Output: ah =    line status
                        C  ;       Trash:  bx destroyed.
                        C  ;
                        C  ;       Assumes:        com_mctl_x  = com_data_x + 4 = dx + 4
                        C  ;                       com_lstat_x = com_data_x + 5 = dx + 5
                        C  ;                       com_mstat_x = com_data_x + 6 = dx + 6
                        C  ;
                        C  ;                       scc_data_x = scc_ctl_x + 1 = dx + 1
                        C  ;---------------------------------------------------------------------------
                        C
E8EC                    C  com_pb   proc    near                    ; ah = 01h
                        C          assume  cs:code, ds:nothing, es:nothing, ss:nothing
                        C
E8EC  52                C          push    dx                      ; save dx = com_data_x
E8ED  50                C          push    ax                      ; save character to output in al
                        C
E8EE  B0 03             C          mov     al,(com_rts+com_dtr)    ; signal RTS & DTR
E8F0  83 C2 04          C          add     dx,4                    ; dx = com_mctl_x
E8F3  EE                C          out     dx,al                   ; send to modem control register
                        C
E8F4  B4 30             C          mov     ah,(com_dsr+com_cts)    ; wait for DSR & CTS
E8F6  42                C          inc     dx
E8F7  42                C          inc     dx                      ; dx = com_mstat_x
E8F8  E8 E8BE R         C          call    rs_ws                   ; wait for modem status register
E8FB  75 13             C          jnz     rs_pbe                  ; if time-out, take jump
                        C
E8FD  B4 20             C          mov     ah,com_txd              ; wait for transmit ready
E8FF  4A                C          dec     dx                      ; dx = com_lstat_x
E900  E8 E8BE R         C          call    rs_ws                   ; wait for line status register
E903  75 0B             C          jnz     rs_pbe                  ; if time-out, take jump
                        C
E905  58                C          pop     ax                      ; restore character to output in al
E906  8A D8             C          mov     bl,al                   ; save character input/output in bl
E908  E8 E87B R         C          call    rs_stat                 ; get return status
E90B  8A C3             C          mov     al,bl                   ; restore character input/output in al
E90D  5A                C          pop     dx                      ; restore dx = com_data_x
E90E  EE                C          out     dx,al                   ; else, output the character
                        C
                        C
                        C                                          ; exit for put and get byte
```

```
E90F                    C  rs_pb_gb:                           ; if SCC Z8530, dx = scc_ctl_x
                        C
E90F  C3                C          ret
                        C
E910                    C  rs_pbe:                             ; exit for put byte error
E910  5B                C          pop     bx                  ; restore character to output in bl
E911  5A                C          pop     dx                  ; if SCC Z8530, restore dx = scc_ctl_x
                        C                                      ; else INS8250, restore dx = com_data_x
                        C
E912  E8 E87B R         C          call    rs_stat             ; get return status
E915  8A C3             C          mov     al,bl               ; restore character to output in al
E917  80 CC 80          C          or      ah,com_te           ; indicate timeout error
E91A  C3                C          ret
                        C
E91B                    C  com_pb  endp
                        C
                        C
E91B                    C  scc_pb  proc    near                ; ah = 01h
                        C          assume  cs:code, ds:nothing, es:nothing, ss:nothing
                        C
E91B  52                C          push    dx                  ; save dx = scc_ctl_x
E91C  50                C          push    ax                  ; save character to output in al
                        C
E91D  B4 04             C          mov     ah,scc_txd          ; wait for transmit ready
E91F  E8 E8BE R         C          call    rs_ws
E922  75 EC             C          jnz     rs_pbe              ; if time-out, take jump
                        C
E924  58                C          pop     ax                  ; restore character to output in al
E925  42                C          inc     dx                  ; dx = scc_data_x
E926  EE                C          out     dx,al               ; else, output the character
                        C
E927  5A                C          pop     dx                  ; restore dx = scc_ctl_x
E928  EB E5             C          jmp     short rs_pb_gb
                        C
E92A                    C  scc_pb  endp
                        C
                        C  ;---------------------------------------------------------------------
                        C  ;
                        C  ;       INS8250 Get Byte (com_gb) & SCC Z8530 Get Byte (scc_gb)
                        C  ;
                        C  ;       Receive Character to RS-232 Interface.
                        C  ;
                        C  ;       Input:  cx =    RS-232 time-out
                        C  ;               dx =    if dh =  0, then address of Z8530 channel (scc_ctl_x).
                        C  ;                       if dh <> 0, then address of 8250 data port (com_data_x).
                        C  ;       Output: al =    character received
                        C  ;               ah =    line status
                        C  ;       Trash:  bx destroyed.
                        C  ;
                        C  ;       Assumes:        com_mctl_x  = com_data_x + 4 = dx + 4
                        C  ;                       com_lstat_x = com_data_x + 5 = dx + 5
                        C  ;                       com_mstat_x = com_data_x + 6 = dx + 6
                        C  ;
                        C  ;                       scc_data_x = scc_ctl_x + 1 = dx + 1
                        C  ;---------------------------------------------------------------------
                        C
E92A                    C  com_gb  proc    near                ; ah = 02h
                        C          assume  cs:code, ds:nothing, es:nothing, ss:nothing
                        C
E92A  52                C          push    dx                  ; save dx = com_data_x
E92B  B0 01             C          mov     al,com_dtr          ; signal DTR
E92D  83 C2 04          C          add     dx,4                ; dx = com_mctl_x
E930  EE                C          out     dx,al               ; send to modem control register
                        C
E931  B4 20             C          mov     ah,com_dsr          ; wait for DSR
E933  42                C          inc     dx
E934  42                C          inc     dx                  ; dx = com_mstat_x
E935  E8 E8BE R         C          call    rs_ws               ; wait for modem status register
E938  75 0E             C          jnz     rs_gbe              ; if time-out, take jump
                        C
E93A  B4 01             C          mov     ah,com_rxd          ; wait for receive ready
E93C  4A                C          dec     dx                  ; dx = com_lstat_x
E93D  E8 E8BE R         C          call    rs_ws               ; wait for line status register
E940  75 06             C          jnz     rs_gbe              ; if time-out, take jump
                        C
E942  80 E4 0E          C          and     ah,0Eh              ; Only interested on low nibble.
E945  5A                C          pop     dx                  ; restore dx = com_data_x
E946  EC                C          in      al,dx               ; else get character
E947  C3                C          ret
                        C
                        C
E948                    C  rs_gbe:                             ; exit for get byte error
E948  5A                C          pop     dx                  ; if SCC Z8530, restore dx = scc_ctl_x
                        C                                      ; else INS8250, restore dx = com_data_x
E949  80 CC 80          C          or      ah,com_te           ; indicate timeout error
E94C  C3                C          ret
                        C
E94D                    C  com_gb  endp
                        C
                        C
E94D                    C  scc_gb  proc    near                ; ah = 02h
                        C          assume  cs:code, ds:nothing, es:nothing, ss:nothing
                        C
E94D  52                C          push    dx                  ; save dx = scc_ctl_x
                        C
E94E  B4 01             C          mov     ah,scc_rxd          ; wait for receive ready
E950  E8 E8BE R         C          call    rs_ws
E953  75 F3             C          jnz     rs_gbe              ; if time-out, take jump
                        C
E955  42                C          inc     dx                  ; dx = scc_data_x
```

```
E956  EC                      C          in      al,dx                   ; else get character
                              C
E957  5A                      C          pop     dx                      ; restore dx = scc_ctl_x
E958  EB B5                   C          jmp     short rs_pb_gb
                              C
E95A                          C scc_gb   endp
E95A                          C code     ends
                              C include kb2.asm
                              C
                              C
                              C ;===========================================================================
                              C ;       Filename:       kb2.src
                              C ;
                              C ;       This module includes INT 09h & 16h.
                              C ;
                              C ;===========================================================================
                              C
E95A                          C code     segment public 'ROM'
                              C          assume  cs:code, ds:nothing, es:nothing, ss:nothing
                              C
E987                          C          ORG     0E987h
                              C
                              C
                              C ;===========================================================================
                              C ;       INT 09h -- i8041A Keyboard Hardware Interrupt Service Routine
                              C ;
                              C ;       Note:   `make'  -> key is depressed -> 00h + key scan code
                              C ;               `break' -> key is released  -> 80h + key scan code
                              C ;===========================================================================
                              C
                              C
E987                          C k_int    proc    near
                              C          assume  cs:code, ds:nothing, es:nothing, ss:nothing
                              C
E987  FB                      C          sti                                     ; re-enable interrupts
E988  FC                      C          cld                                     ; clear direction flag
                              C
E989  50                      C          push    ax
E98A  53                      C          push    bx
E98B  51                      C          push    cx
E98C  52                      C          push    dx
E98D  56                      C          push    si
E98E  57                      C          push    di
E98F  1E                      C          push    ds
E990  2E: 8E 1E E5F2 R        C          mov     ds,word ptr cs:[set_ds_word]    ; avoid potential stack problems
E995  06                      C          push    es
                              C
                              C                  assume  cs:code, ds:data, es:nothing, ss:nothing
                              C
                              C ; Get the scan code.
                              C
E996  E4 60                   C          in      al,p_kscan                      ; get scan code from data port
E998  8A E0                   C          mov     ah,al                           ; save scan code in ah
                              C
                              C ; Reset the keyboard.
                              C
E99A  BA 0061                 C          mov     dx,p_kctrl                      ; get control port address
E99D  EC                      C          in      al,dx                           ; get the status
E99E  8A D8                   C          mov     bl,al                           ; save the status in bl
                              C
E9A0  0C 80                   C          or      al,080h                         ; set bit #7 -- reset
E9A2  EE                      C          out     dx,al                           ; reset keyboard
                              C
E9A3  8A C3                   C          mov     al,bl                           ; retrieve original status
E9A5  EE                      C          out     dx,al                           ; send it back to keyboard
                              C
                              C ; Retrieve the scan code in both al & ah.
                              C
E9A6  8A C4                   C          mov     al,ah                           ; scan code in both registers
                              C
                              C ; Test for overrun scan code from keyboard = 0FFh.
                              C ; Note:  20 key scan codes can be buffered up by the   keyboard.
                              C
E9A8  3C FF                   C          cmp     al,0FFh                         ; an overrun scan code?
E9AA  75 06                   C          jnz     k_ok                            ; if not, continue
                              C
E9AC  E8 EBC3 R               C          call    k_beep                          ; beep the speaker
E9AF  E9 EA5E R               C          jmp     k_nop
E9B2                          C k_ok:
                              C
                              C ;---------------------------------------------------------------------------
                              C ;       ah      = scan code (make/break)        al = scan code (make/break)
                              C ;       bx      = ?
                              C ;       cx      = ?
                              C ;       dx      = ?
                              C ;       es:di   = ?
                              C ;---------------------------------------------------------------------------
                              C
E9B2  24 7F                   C          and     al,07Fh                         ; al = scan code make
                              C
E9B4  C4 3E 0084 R            C          les     di,dword ptr ds:[master_tbl_ptr] ; es:di points to master table
E9B8  26: C4 7D 02            C          les     di,dword ptr es:[di+0002h]       ; es:di points to p_kscan table
                              C
E9BC  33 DB                   C          xor     bx,bx                           ; clear bh
E9BE  33 C9                   C          xor     cx,cx                           ; clear cl
E9C0  8A 2E 0017 R            C          mov     ch,byte ptr ds:[kb_flag]
                              C
E9C4  8A D8                   C          mov     bl,al                           ; bx = scan code make
E9C6  D1 E3                   C          shl     bx,1                            ; bx = 2*(scan code make)
E9C8  D1 E3                   C          shl     bx,1                            ; bx = 4*(scan code make)
```

```
                                C
                                C ;--------------------------------------------------------------------
                                C ;       ah      = scan code (make/break)        al = scan code (make)
                                C ;       bx      = 4*(scan code make) = 4*al
                                C ;       ch      = kb_flag                       cl = 0
                                C ;       dx      = ?
                                C ;       es:di   = p_kscan base
                                C ;--------------------------------------------------------------------
                                C
E9CA  4B                        C         dec     bx                              ; bx = 4*(scan code make)-1
                                C
E9CB  F6 C5 08                  C         test    ch,alt_shift                    ; alt state?
E9CE  75 32                     C         jnz     k_ix                            ; if so, has highest priority
                                C
E9D0  4B                        C         dec     bx                              ; bx = 4*(scan code make)-2
                                C
E9D1  F6 C5 04                  C         test    ch,cntrl_shift                  ; control state?
E9D4  75 2C                     C         jnz     k_ix                            ; if so, next highest priority
                                C
E9D6  4B                        C         dec     bx                              ; bx = 4*(scan code make)-3
                                C
                                C ; Handle CapLk Case.
                                C
E9D7  3C 37                     C         cmp     al,55                           ; 0 <= scan code <= (7*8)-1
E9D9  77 0C                     C         ja      k_no_cap                        ; test NumLk case.
                                C
E9DB  F6 C5 40                  C         test    ch,caps_lock_mode                       ; caplock state?
E9DE  74 1C                     C         jz      k_no_lock                       ; if not, test shift states
                                C
E9E0  E8 EB85 R                 C         call    k_bit                           ; get kb_cap_flags bit in cf
E9E3  73 17                     C         jnb     k_no_lock                       ; (jnc) swap if cf set
                                C
E9E5  EB 0D                     C         jmp     short k_lock                    ; honor caps lock.
                                C
E9E7                            C k_no_cap:
                                C
                                C ; Handle NumLk Case.
                                C
E9E7  3C 47                     C         cmp     al,71                           ; scan code >= 71?
E9E9  72 11                     C         jb      k_no_lock
E9EB  3C 53                     C         cmp     al,83                           ; scan code <= 83?
E9ED  77 0D                     C         ja      k_no_lock
E9EF  F6 C5 20                  C         test    ch,num_lock_mode                        ; numlock state?
E9F2  74 08                     C         jz      k_no_lock                       ; if not, test shift states
                                C
E9F4                            C k_lock:                                         ; either caps or num lock.
E9F4  F6 C5 03                  C         test    ch,(left_shift+right_shift)     ; if so, and shift state
E9F7  74 09                     C         jz      k_ix                            ; also true
                                C
E9F9  4B                        C         dec     bx                              ; bx = 4*(scan code make)-4
E9FA  EB 06                     C         jmp     short k_ix                      ; (base case)
                                C
E9FC                            C k_no_lock:                                      ; neither caps or num lock.
E9FC  F6 C5 03                  C         test    ch,(left_shift+right_shift)     ; (shift state)
E9FF  75 01                     C         jnz     k_ix
EA01  4B                        C         dec     bx                              ; bx = 4*(scan code make)-4
                                C ;       jmp     short k_ix                      ; (base case) fall through
                                C
                                C
EA02                            C k_ix:                                           ; bx = index into p_kscan table
EA02  D1 E3                     C         shl     bx,1                            ; bx = p_kscan word index
EA04  03 FB                     C         add     di,bx                           ; add p_kscan base & index
                                C
EA06  26: 8B 15                 C         mov     dx,word ptr es:[di]             ; get data word from table
                                C
EA09  33 DB                     C         xor     bx,bx                           ; clear bh
EA0B  8A DA                     C         mov     bl,dl                           ; move translated   key to bx
EA0D  F6 06 0018 R 01           C         test    byte ptr ds:[kb_flag_1],dlx_kb  ; are we a deluxe keyboard
EA12  74 02                     C         jz      k_xlat                          ;   key
                                C
EA14  8A DE                     C         mov     bl,dh                           ; move translated deluxe
                                C                                                 ;   key to key
EA16                            C k_xlat:                                         ; bx has translated byte (bh=0)
                                C
                                C ;--------------------------------------------------------------------
                                C ;       ah      = scan code (make/break)        al = scan code (make)
                                C ;       bx      = deluxe keyboard translated byte (bh = 0)
                                C ;       ch      = kb_flag                       cl = 0
                                C ;       dx      = es:[di] (could be deluxe key code)
                                C ;       es:di   = p_kscan base + p_kscan scan code word index
                                C ;--------------------------------------------------------------------
                                C
                                C ; Registers are all loaded up.  Start going through the cases.
                                C
EA16  0A D2                     C         or      dl,dl                           ; deluxe scan codes are
EA18  74 1C                     C         jz      k_no_case                       ; NEVER special cases!!!!
                                C
                                C ; Test for special cases.
                                C
EA1A  80 FB C0                  C         cmp     bl,0C0h                         ; xlated byte special case?
EA1D  72 17                     C         jb      k_no_case                       ; if not, handle unspecial case
EA1F  80 FB D8                  C         cmp     bl,0D8h                         ; 0C0h <= xlated byte <= 0D8h
EA22  77 12                     C         ja      k_no_case                       ; if so, do the special function
                                C
                                C ; Test for `break' of special case.
                                C
EA24  80 FB C8                  C         cmp     bl,0C8h                         ; is xlated byte a shift key?
EA27  72 04                     C         jb      k_jmp                           ; if so, do `break' of shift key
                                C                                                 ; 0C0h <= xlated byte < 0C8h
EA29  0A E4                     C         or      ah,ah
```

```
EA2B  78 2E                   C          js      k_none                          ; nothing for `break' of others.
                              C
EA2D                          C  k_jmp:                                          ; jump to special case routine.
EA2D  8B F3                   C          mov     si,bx                           ; if so, si gets special index
EA2F  D1 E6                   C          shl     si,1                            ; make it a word index
EA31  2E: FF A4 EA66 R        C          jmp     cs:[si+((offset k_case)-(2*0C0h))]
                              C
EA36                          C  k_no_case:
                              C
                              C  ; Test for `break' of non special case.
                              C
EA36  0A E4                   C          or      ah,ah
EA38  78 21                   C          js      k_none                          ; do nothing if `break' of key.
                              C
                              C  ; Test for `unpause' case.
                              C
EA3A  F6 06 0018 R 08         C          test    byte ptr ds:[kb_flag_1],pause_mode      ; are we in hold state?
EA3F  74 0B                   C          jz      k_no_hold                       ; if not, continue.
EA41  3C 45                   C          cmp     al,num_lock_key                 ; don't clear hold state
EA43  74 16                   C          je      k_none                          ; on num_lock_key (Pause).
EA45  80 26 0018 R F7         C          and     byte ptr ds:[kb_flag_1],not pause_mode  ; reset hold state bit &
EA4A  EB 0F                   C          jmp     short k_none                    ; ignore the key.
                              C
EA4C                          C  k_no_hold:
                              C
                              C  ; Test for deluxe scan code.
                              C
EA4C  0A D2                   C          or      dl,dl
EA4E  75 04                   C          jnz     k_no_xcode
                              C
EA50  8B C2                   C          mov     ax,dx                           ; move deluxe key to scan code
EA52  EB 02                   C          jmp     short k_buf                     ; put ax into the buffer
                              C
EA54                          C  k_no_xcode:
EA54  8A C3                   C          mov     al,bl                           ; move translated key to key
                              C
EA56                          C  k_buf:                                          ; put ax into kb_buffer.
                              C                                                  ; try to put ax into kb_buffer.
EA56  E8 EBAD R               C          call    k_try                           ; zf set (z)   if buffer is full
EA59  74 03                   C          jz      k_nop                           ; zf reset (nz) if buffer got ax
                              C
EA5B                          C  k_none:                                         ; scan code translate to nothing
                              C
                              C  ; Send specific end of interrupt (SEOI) to pic `command' port.
                              C
EA5B  E8 EBA4 R               C          call    k_eoi                           ; send specific end of interrupt
                              C
EA5E  07                      C  k_nop:  pop     es                      ; restore registers without issuing SEOI
EA5F  1F                      C          pop     ds
EA60  5F                      C          pop     di
EA61  5E                      C          pop     si
EA62  5A                      C          pop     dx
EA63  59                      C          pop     cx
EA64  5B                      C          pop     bx
EA65  58                      C          pop     ax
EA66  CF                      C          iret
                              C
                              C  ;--------------------------------------------------------------------
                              C  ;       Test for system reset sequence.  `make' only.
                              C  ;--------------------------------------------------------------------
                              C
EA67  80 E5 0C                C  k_res:  and     ch,(alt_shift+cntrl_shift)
EA6A  80 FD 0C                C          cmp     ch,(alt_shift+cntrl_shift)      ; is it CTL ALT shift?
EA6D  75 EC                   C          jne     k_none
                              C
                              C  ; CTL ALT DEL system reset.
                              C
EA6F  C7 06 0072 R 1234       C          mov     word ptr ds:[reset_flag],01234h ; set flag for warm boot
EA75  E9 DD5A R               C          jmp     diagnostics_1                   ; re-boot
                              C
                              C  ;--------------------------------------------------------------------
                              C  ;       Pause waiting for another key.  `make' only.
                              C  ;--------------------------------------------------------------------
                              C
EA78                          C  k_pause:
EA78  80 0E 0018 R 08         C          or      byte ptr ds:[kb_flag_1],pause_mode      ; set the pause bit.
                              C
EA7D  E8 EBA4 R               C          call    k_eoi                           ; send specific end of interrupt
                              C
                              C  ; Note: Video not disabled during vertical retrace.
                              C
EA80  80 3E 0049 R 07         C          cmp     byte ptr ds:[v_mode],7  ; never on a monochrome card
EA85  74 0B                   C          je      k_hold
                              C
EA87  8B 16 0063 R            C          mov     dx,word ptr ds:[v_base6845]     ; get 6845 pointer register
EA8B  83 C2 04                C          add     dx,4                            ; get 6845 mode control register
EA8E  A0 0065 R               C          mov     al,byte ptr ds:[v_3x8]  ; get the video mode last sent
EA91  EE                      C          out     dx,al                           ; enable video
                              C
EA92  F6 06 0018 R 08         C  k_hold: test    byte ptr ds:[kb_flag_1],pause_mode      ; test the pause bit.
EA97  75 F9                   C          jnz     k_hold                          ; loop until pause bit cleared
EA99  EB C3                   C          jmp     short k_nop
                              C
                              C  ;--------------------------------------------------------------------
                              C  ;       Print Screen sequence.  `make' only.
                              C  ;--------------------------------------------------------------------
                              C
EA9B  E8 EBA4 R               C  k_prt:  call    k_eoi                           ; send specific end of interrupt
EA9E  CD 05                   C          INT     5h                              ; issue print screen interrupt
EAA0  EB BC                   C          jmp     short k_nop
```

```
                                C
                                C  ;--------------------------------------------------------------------
                                C  ;       Deluxe code put NUL into kb_buffer.  `make' only.
                                C  ;--------------------------------------------------------------------
                                C
EAA2  B8 0300                   C  k_nul:  mov     ax,0300h                    ; NUL=X03
EAA5  EB AF                     C          jmp     short k_buf                 ; put ax into the buffer
                                C
                                C  ;--------------------------------------------------------------------
                                C  ; Four state shifts. `make' & `break' in kb_flag_1 plus history in kb_flag.
                                C  ;--------------------------------------------------------------------
                                C
EAA7  B0 80                     C  k_ins:  mov     al,insert_shift             ; INS toggle lock
EAA9  E8 EAD4 R                 C          call    k_4tog                      ; toggle 4 state
                                C
EAAC  0A E4                     C          or      ah,ah                       ; is INS toggle `make' ?
EAAE  78 AB                     C          js      k_none                      ; if not, exit
EAB0  B8 5200                   C          mov     ax,(insert_key*100h)+00h            ; else, ax gets deluxe INS key
EAB3  EB A1                     C          jmp     k_buf                       ; put ax into the buffer
                                C
                                C
EAB5  B0 40                     C  k_cap:  mov     al,caps_lock_shift                  ; CAPS LOCK toggle lock
EAB7  E8 EAD4 R                 C          call    k_4tog                      ; toggle 4 state
EABA  B1 01                     C          mov     cl,00000001b                ; CAPS LOCK LED is bit #0.
EABC  E8 EB63 R                 C          call    k_LED_cap                   ; send LED information
EABF  EB 9A                     C  k_non1: jmp     short k_none
                                C
EAC1  B0 20                     C  k_num:  mov     al,num_lock_shift                   ; NUM LOCK toggle lock
EAC3  E8 EAD4 R                 C          call    k_4tog                      ; toggle 4 state
EAC6  B1 02                     C          mov     cl,00000010b                ; NUM LOCK LED is bit #1.
EAC8  E8 EB59 R                 C          call    k_LED_num                   ; send LED information
EACB  EB F2                     C          jmp     short k_non1
                                C
                                C
EACD  B0 10                     C  k_scr:  mov     al,scrl_lock_shift                  ; SCROLL LOCK toggle lock
EACF  E8 EAD4 R                 C          call    k_4tog                      ; toggle 4 state
EAD2  EB EB                     C          jmp     short k_non1
                                C
                                C
EAD4  0A E4                     C  k_4tog: or      ah,ah                       ; is toggle shift `make' ?
EAD6  78 0F                     C          js      k_4res                      ; if `break' reset kb_flag_1
                                C
EAD8  84 06 0018 R              C          test    byte ptr ds:[kb_flag_1],al  ; if `make', test bit.
EADC  75 08                     C          jnz     k_4ret                      ; return if already pressed
                                C
EADE  08 06 0018 R              C          or      byte ptr ds:[kb_flag_1],al  ; set the bit.
EAE2  30 06 0017 R              C          xor     byte ptr ds:[kb_flag],al    ; toggle kb_flag history.
EAE6  C3                        C  k_4ret: ret
                                C
EAE7  F6 D0                     C  k_4res: not     al
EAE9  20 06 0018 R              C          and     byte ptr ds:[kb_flag_1],al  ; if `break', reset bit only.
EAED  C3                        C          ret
                                C
                                C  ;--------------------------------------------------------------------
                                C  ;       Two state shifts. `make' & `break' in kb_flag only.
                                C  ;--------------------------------------------------------------------
                                C
EAEE  B0 08                     C  k_alt:  mov     al,alt_shift                ; ALT set/reset kb_flag
EAF0  E8 EB0F R                 C          call    k_2tog                      ; toggle 2 state
                                C
EAF3  33 C0                     C          xor     ax,ax                       ; scan code of ah = 0.
EAF5  86 06 0019 R              C          xchg    al,byte ptr ds:[alt_input]  ; alt_input gets 0.
EAF9  0A C0                     C          or      al,al                       ; was alt_input 0?
EAFB  74 C2                     C          je      k_non1                      ; if so, do nothing.
EAFD  E9 EA56 R                 C          jmp     k_buf                       ; else, put it into buffer.
                                C
                                C
EB00  B0 04                     C  k_ctl:  mov     al,cntrl_shift              ; CTL set/reset kb_flag
EB02  EB 06                     C          jmp     short k_2ret                ; toggle 2 state and return
                                C
                                C
EB04  B0 02                     C  k_lsh:  mov     al,left_shift               ; LEFT SHIFT set/reset kb_flag
EB06  EB 02                     C          jmp     short k_2ret                ; toggle 2 state and return
                                C
                                C
EB08  B0 01                     C  k_rsh:  mov     al,right_shift              ; RIGHT SHIFT set/reset kb_flag
                                C  ;       jmp     short k_2ret                ; fall through
                                C
                                C
EB0A                            C  k_2ret:
EB0A  E8 EB0F R                 C          call    k_2tog                      ; toggle 2 state
EB0D  EB B0                     C          jmp     short k_non1
                                C
EB0F  0A E4                     C  k_2tog: or      ah,ah                       ; is set/reset shift `make' ?
EB11  78 05                     C          js      k_2res                      ; if `break', reset bit only.
                                C
EB13  08 06 0017 R              C          or      byte ptr ds:[kb_flag],al    ; if `make', set bit only.
EB17  C3                        C          ret
                                C
EB18  F6 D0                     C  k_2res: not     al
EB1A  20 06 0017 R              C          and     byte ptr ds:[kb_flag],al    ; if `break', reset only.
EB1E  C3                        C          ret
                                C
                                C  ;--------------------------------------------------------------------
                                C  ;       Alternate Numeric Keypad.  `make' only.
                                C  ;--------------------------------------------------------------------
                                C
EB1F  41                        C  k_alt9: inc     cx                          ; Alternate Numeric Keypad #9
EB20  41                        C  k_alt8: inc     cx                          ; Alternate Numeric Keypad #8
```

```
EB21  41                C  k_alt7: inc     cx                              ; Alternate Numeric Keypad #7
EB22  41                C  k_alt6: inc     cx                              ; Alternate Numeric Keypad #6
EB23  41                C  k_alt5: inc     cx                              ; Alternate Numeric Keypad #5
EB24  41                C  k_alt4: inc     cx                              ; Alternate Numeric Keypad #4
EB25  41                C  k_alt3: inc     cx                              ; Alternate Numeric Keypad #3
EB26  41                C  k_alt2: inc     cx                              ; Alternate Numeric Keypad #2
EB27  41                C  k_alt1: inc     cx                              ; Alternate Numeric Keypad #1
EB28                    C  k_alt0:                                         ; Alternate Numeric Keypad #0
                        C
EB28  B0 0A             C          mov     al,10
EB2A  F6 26 0019 R      C          mul     byte ptr ds:[alt_input]         ; alt_input = (10*alt_input)+cl
EB2E  02 C1             C          add     al,cl
EB30  A2 0019 R         C          mov     byte ptr ds:[alt_input],al
EB33  EB 8A             C          jmp     short k_non1
                        C
                        C  ;---------------------------------------------------------------------
                        C  ;       Double Zero on Keypad.  `make' only.
                        C  ;---------------------------------------------------------------------
                        C
EB35  B0 30             C  k_00:   mov     al,'0'                          ; try to put ax into kb_buffer.
EB37  E8 EBAD R         C          call    k_try                           ; zf set (z)   if buffer is full
EB3A  74 03             C          jz      k_nop1                          ; zf reset (nz) if buffer got ax
EB3C  E9 EA56 R         C          jmp     k_buf                           ; put it into buffer, again.
                        C
EB3F  E9 EA5E R         C  k_nop1: jmp     k_nop
                        C
                        C  ;---------------------------------------------------------------------
                        C  ;       Break key sequence.  `make' only.
                        C  ;---------------------------------------------------------------------
                        C
EB42  BB 001E R         C  k_brk:  mov     bx,ds:(offset kb_buffer)
EB45  89 1E 001A R      C          mov     word ptr ds:[buffer_head],bx    ; reset buffer to empty
EB49  89 1E 001C R      C          mov     word ptr ds:[buffer_tail],bx
EB4D  C6 06 0071 R 80   C          mov     byte ptr ds:[bios_break],80h    ; turn on bios_break bit
EB52  CD 1B             C          INT     1Bh                             ; break interrupt vector
EB54  33 C0             C          xor     ax,ax                           ; ax gets deluxe 00h
EB56  E9 EA56 R         C          jmp     k_buf                           ; put ax into the buffer
                        C
EB59                    C  k_int   endp
                        C
                        C  ;---------------------------------------------------------------------
                        C  ;       Puts keyboard LED's in correct state after CAPS/NUM LOCK.
                        C  ;
                        C  ;       Input:  ah = scan code (make or break)
                        C  ;               al = kb_flag bit for CAPS/NUM LOCK (caps_lock_shift or num_lock_shift)
                        C  ;               cl = 00000001b for CAPS LOCK LED or 00000010b for NUM LOCK LED.
                        C  ;       Output: None.
                        C  ;
                        C  ;       Trash:  al & cl destroyed.
                        C  ;---------------------------------------------------------------------
                        C
EB59                    C  k_LED_num       proc    near
                        C                  assume  cs:code, ds:data, es:nothing, ss:nothing
                        C
EB59  F6 06 0018 R 01   C          test    byte ptr ds:[kb_flag_1],d1x_kb  ; are we a deluxe keyboard
EB5E  74 03             C          jz      k_LED_cap                       ; if   kb, LED is num_lock
EB60  80 F1 80          C          xor     cl,10000000b                    ; if deluxe kb, LED is
                        C                                                  ;   ~num_lock
                        C
EB63                    C  k_LED_cap:
EB63  0A E4             C          or      ah,ah                           ; is CAPS/NUM LOCK `make' ?
EB65  78 1D             C          js      k_LED_ret                       ; if not, exit
                        C
EB67  84 06 0017 R      C          test    byte ptr ds:[kb_flag],al        ; is CAPS/NUM LOCK kb_flag set ?
EB6B  74 03             C          jz      k_LED_cmd                       ; if not, LED data is ok.
EB6D  80 F1 80          C          xor     cl,10000000b                    ; else, flip sense of LED data.
                        C
EB70                    C  k_LED_cmd:                                      ; polling loop to send command.
EB70  E4 64             C          in      al,kb_status                    ; get 8041 status
EB72  A8 02             C          test    al,00000010b                    ; test input buffer bit
EB74  75 FA             C          jnz     k_LED_cmd                       ; if not ok to write cmd, loop.
                        C
EB76  B0 13             C          mov     al,013h                         ; keyboard `LED' command.
EB78  E6 60             C          out     p_kscan,al                      ; send keyboard `LED' command.
                        C
EB7A                    C  k_LED_dat:                                      ; polling loop to send data.
EB7A  E4 64             C          in      al,kb_status                    ; get 8041 status
EB7C  A8 02             C          test    al,00000010b                    ; test input buffer bit
EB7E  75 FA             C          jnz     k_LED_dat                       ; if not ok to write data, loop.
                        C
EB80  8A C1             C          mov     al,cl                           ; retrieve keyboard `LED' data.
EB82  E6 60             C          out     p_kscan,al                      ; send keyboard `LED' data.
EB84                    C  k_LED_ret:
EB84  C3                C          ret
                        C
EB85                    C  k_LED_num       endp
                        C
                        C  ;---------------------------------------------------------------------
                        C  ;       Get kb_cap_flags bit into the carry flag (cf).
                        C  ;
                        C  ;       Input:  al = scan code (make)
                        C  ;               cl = 0
                        C  ;               es:di   = p_kscan base
                        C  ;       Output: cf set if kb_cap_flags bit
                        C  ;
                        C  ;       Trash:  si destroyed.
                        C  ;---------------------------------------------------------------------
                        C
EB85                    C  k_bit   proc    near
                        C          assume  cs:code, ds:data, es:nothing, ss:nothing
```

```
                        C
EB85  8B F3             C          mov     si,bx                        ; save bx
                        C
EB87  33 DB             C          xor     bx,bx                        ; clear bh
EB89  8A D8             C          mov     bl,al                        ; bx = scan code (make) index
                        C                                               ; bx = 00000000 00xxyyy
                        C
                        C
EB8B  B1 03             C          mov     cl,3                         ; rotate right bx 3
EB8D  D3 CB             C          ror     bx,cl                        ; bx = yyy00000 00000xxx
                        C
EB8F  B1 03             C          mov     cl,3                         ; rotate left bh 3
EB91  D2 C7             C          rol     bh,cl                        ; bx = 00000yyy 00000xxx
                        C                                               ; bh = remainder = (0-7)
                        C                                               ; bl = quotient  = (0-6)
                        C
                        C
EB93  8A CF             C          mov     cl,bh                        ; cl = remainder = (0-7)
EB95  32 FF             C          xor     bh,bh                        ; bx = quotient  = (0-6)
                        C
                        C
EB97  26: 8A 59 F9      C          mov     bl,byte ptr es:[di+bx-7]     ; bl = proper cap_flags byte
                        C
EB9B  D2 D3             C          rcl     bl,cl                        ; rotate (0-7) times into bit #7
EB9D  32 C9             C          xor     cl,cl                        ; cl = 0
EB9F  D0 D3             C          rcl     bl,1                         ; rotate into cf bit #7
                        C
EBA1  8B DE             C          mov     bx,si                        ; restore bx
EBA3  C3                C          ret
                        C
EBA4                    C  k_bit   endp
                        C
                        C  ;---------------------------------------------------------------------
                        C  ;       Input:  None.
                        C  ;       Output: None.
                        C  ;
                        C  ;       Trash:  al & dx destroyed.
                        C  ;---------------------------------------------------------------------
                        C
EBA4                    C  k_eoi   proc    near
                        C
                        C  ; Send specific end of interrupt (SEOI) to pic `command' port.
                        C
EBA4  FA                C          cli                                  ; disable interrupts
EBA5  B0 61             C          mov     al,pic_seoi_1                ; specific end of interrupt
EBA7  BA 0020           C          mov     dx,pic_0                     ; command to pic `command' port.
EBAA  EE                C          out     dx,al
EBAB  FB                C          sti                                  ; enable interrupts
EBAC  C3                C          ret
                        C
EBAD                    C  k_eoi   endp
                        C
                        C  ;---------------------------------------------------------------------
                        C  ;       Try to put ax into the kb_buffer.
                        C  ;
                        C  ;       Input:  ax      = word to put in kb_buffer.
                        C  ;       Output: zf set   (z)  if buffer is full. (ax trashed)
                        C  ;               zf reset (nz) if buffer got ax.  (ax saved)
                        C  ;
                        C  ;       Trash:  bx, cx, dx, & si destroyed (in general).
                        C  ;---------------------------------------------------------------------
                        C
EBAD                    C  k_try   proc    near
                        C          assume  cs:code, ds:data, es:nothing, ss:nothing
                        C
EBAD  8B 1E 001C R      C          mov     bx,word ptr ds:[buffer_tail]    ; get buffer end pointer
EBB1  8B F3             C          mov     si,bx                           ; save the value
EBB3  E8 E86E R         C          call    k_adv_ptr                       ; advance the tail
                        C
EBB6  3B 1E 001A R      C          cmp     bx,word ptr ds:[buffer_head]    ; has the buffer wrapped around
EBBA  74 07             C          je      k_beep
                        C                                                  ; zf is reset (nz)
EBBC  89 04             C          mov     word ptr ds:[si],ax             ; store the value
EBBE  89 1E 001C R      C          mov     word ptr ds:[buffer_tail],bx    ; move the pointer up
EBC2  C3                C          ret                                     ; return zf reset (nz)
                        C
EBC3                    C  k_try   endp
                        C
                        C  ;---------------------------------------------------------------------
                        C  ;       Input:  None.
                        C  ;       Output: zf set   (z)  always.
                        C  ;
                        C  ;       Trash:  ax, bl, cx, & dx destroyed.
                        C  ;---------------------------------------------------------------------
                        C
EBC3                    C  k_beep  proc    near
                        C          assume  cs:code, ds:data, es:nothing, ss:nothing
                        C
EBC3  E8 EBA4 R         C          call    k_eoi                           ; send specific end of interrupt
                        C
EBC6  BA 0061           C          mov     dx,p_kctrl                      ; get kb control port address
EBC9  EC                C          in      al,dx                           ; get control data
EBCA  8A E0             C          mov     ah,al                           ; save control data
                        C
EBCC  B3 80             C          mov     bl,80h                          ; outer loop counter
                        C
EBCE  24 FC             C  k_lp:   and     al,0FCh                         ; turn off speaker data
EBD0  EE                C          out     dx,al
                        C
EBD1  B9 0048           C          mov     cx,48h                          ; set up count
EBD4  E2 FE             C          loop    $                               ; delay awhile
                        C
```

```
EBD6  0C 02            C          or      al,02h                          ; turn on speaker
EBD8  EE               C          out     dx,al
                       C
EBD9  B9 0048          C          mov     cx,48h                          ; set up count
EBDC  E2 FE            C          loop    $                               ; delay awhile
                       C
EBDE  FE CB            C          dec     bl                              ; decrement outer loop counter
EBE0  75 EC            C          jnz     k_lp
                       C                                                  ; zf is set (z)
EBE2  8A C4            C          mov     al,ah                           ; restore control data
EBE4  EE               C          out     dx,al
EBE5  C3               C          ret                                     ; return zf set (z)
                       C
EBE6                   C  k_beep          endp
                       C
                       C
EBE6                   C  k_data1 proc
                       C
EBE6  EAA7 R           C  k_case  dw      k_ins          ; kbins (0C0h)          ^
EBE8  EAB5 R           C          dw      k_cap          ; kbcap                 |
EBEA  EAC1 R           C          dw      k_num          ; kbnum                 |
EBEC  EACD R           C          dw      k_scr          ; kbscr                 |
EBEE  EAEE R           C          dw      k_alt          ; kbalt        `make' & `break'
EBF0  EB00 R           C          dw      k_ctl          ; kbctl                 |
EBF2  EB04 R           C          dw      k_lsh          ; kblsh                 |
EBF4  EB08 R           C          dw      k_rsh          ; kbrsh                 v
                       C
EBF6  EA67 R           C          dw      k_res          ; kbres (0C8h)          ^
EBF8  EB42 R           C          dw      k_brk          ; kbbrk                 |
EBFA  EA78 R           C          dw      k_pause        ; pause                 |
EBFC  EA9B R           C          dw      k_prt          ; kbprt                 |
EBFE  EAA2 R           C          dw      k_nul          ; kbnul                 |
EC00  EA5B R           C          dw      k_none         ; NONE                  |
                       C                                 ;                       |
EC02  EB1F R           C          dw      k_alt9         ; kdec9                 |
EC04  EB20 R           C          dw      k_alt8         ; kdec8                 |
EC06  EB21 R           C          dw      k_alt7         ; kdec7          `make' only
EC08  EB22 R           C          dw      k_alt6         ; kdec6                 |
EC0A  EB23 R           C          dw      k_alt5         ; kdec5                 |
EC0C  EB24 R           C          dw      k_alt4         ; kdec4                 |
EC0E  EB25 R           C          dw      k_alt3         ; kdec3                 |
EC10  EB26 R           C          dw      k_alt2         ; kdec2                 |
EC12  EB27 R           C          dw      k_alt1         ; kdec1                 |
EC14  EB28 R           C          dw      k_alt0         ; kdec0                 |
                       C                                 ;                       |
EC16  EB35 R           C          dw      k_00           ; kdbl0 (0D8h)          v
                       C
EC18                   C  k_data1 endp
                       C
EC18                   C  code    ends
                       C  include fdu4.asm
                       C
                       C  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                       C
EC18                   C  code    segment public  'ROM'
                       C          assume  cs:code, ds:data, es:nothing, ss:nothing
                       C
EC18                   C  f_nec_reset     proc    near
                       C
EC18  BA 03F4 90       C          mov     dx,f_nec_status                 ; NEC status port
EC1C  EC               C          in      al,dx
EC1D  3C 10            C          cmp     al,10h                          ; NEC busy.
EC1F  75 0E            C          jne     f_nec_reset_ret                 ; no.
EC21  A0 0041 R        C          mov     al,diskette_status              ; save from previous operation.
EC24  50               C          push    ax
EC25  33 C0            C          xor     ax,ax                           ; reset call.
EC27  8B D0            C          mov     dx,ax
EC29  CD 13            C          int     13h
EC2B  58               C          pop     ax
EC2C  A2 0041 R        C          mov     diskette_status,al              ; restore from previous op.
EC2F                   C  f_nec_reset_ret:
EC2F  C3               C          ret
                       C
EC30                   C  f_nec_reset     endp
                       C
EC30                   C  code    ends

                       C  include fdu1.asm
                       C
                       C  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;  EQUATES  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                       C  ;                   Controller constants
                       C
= 000C                 C  f_srt_48                equ     1100b   ; 48TPI Step Rate Time (6 ms @ 4 Mhz)
= 000E                 C  f_srt_96                equ     1110b   ; 96TPI Step Rate Time (4 ms @ 4 Mhz)
= 000F                 C  f_hut                   equ     1111b   ; Head Unload Time (480 ms @ 4 Mhz)
= 0001                 C  f_hlt                   equ     1       ; Head Load Time (4 ms @ 4 Mhz)
= 0000                 C  f_ndma                  equ     0       ; Not DMA bit (0 = dma on)
                       C
= 0025                 C  f_motor_wait            equ     37      ; no. of RTC ticks before turning
                       C                                          ; motor off. (f_motor_wait x 55ms)
                       C
=                      C  f_drive         equ     [bp+0]                  ; byte pointer.
=                      C  f_head          equ     [bp+1]                  ; byte pointer.
=                      C  f_numsecs       equ     [bp+2]                  ; byte pointer.
=                      C  f_command       equ     [bp+3]                  ; byte pointer.
=                      C  f_bufoff        equ     [bp+4]                  ; word pointer.
=                      C  f_secnum        equ     [bp+6]                  ; byte pointer.
=                      C  f_cyl           equ     [bp+7]                  ; byte pointer.
=                      C  f_real_drive    equ     [bp+8]                  ; byte pointer.
                       C
```

```
                               C ;          Floppy Disk port addresses
                               C
 = 03F2                        C f_motor_port           equ     03F2h   ; drive select port
 = 03F4                        C f_nec_status           equ     03F4h   ; disk controller status port
 = 03F5                        C f_nec_data             equ     03F5h   ; disk controller data port
                               C
                               C ;          Floppy Disk commands
                               C
 = 00E6                        C f_read_cmd             equ     0E6h    ; read data
 = 00C5                        C f_write_cmd            equ     0C5h    ; write data
 = 004D                        C f_format_cmd           equ     04Dh    ; format
 = 0007                        C f_recal_cmd            equ     007h    ; recalibrate
 = 0008                        C f_snsint_cmd           equ     008h    ; sense interrupt
 = 0004                        C f_snsdrv_cmd           equ     004h    ; sense drive
 = 0003                        C f_specify_cmd          equ     003h    ; specify
 = 000F                        C f_seek_cmd             equ     00Fh    ; seek
                               C
                               C ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;; CODE ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;,;;;;;;;;
                               C
 EC30                          C code    segment public  'ROM'
                               C         assume  cs:code, ds:data, es:nothing, ss:nothing
                               C
                               C
                               C ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                               C
                               C
 EC59                          C         ORG     0EC59h
 EC59                          C fd_io   proc    far
                               C
 EC59  FB                      C         sti                                    ; enable interupts
 EC5A  55                      C         push    bp
 EC5B  1E                      C         push    ds
 EC5C  56                      C         push    si
 EC5D  57                      C         push    di
 EC5E  52                      C         push    dx                             ; head & drive#
 EC5F  51                      C         push    cx                             ; cyl. & sec#
 EC60  53                      C         push    bx                             ; buffer offset
 EC61  50                      C         push    ax                             ; command & #secs
 EC62  52                      C         push    dx                             ; this one gets modified(96 TPI)
 EC63  8B EC                   C         mov     bp,sp                          ; BP preserves SP throughout
                               C
                               C ; test command code & use jump table to jump to appropriate routine
                               C
 EC65  2E: 8E 1E E5F2 R        C         mov     ds,word ptr cs:[set_ds_word]   ; DS = data_seg (40h)
 EC6A  80 26 003F R 0F         C         and     motor_status,0Fh               ; preserve motor on bits.
 EC6F  80 FA 01                C         cmp     dl,1                           ; max. drives
 EC72  77 05                   C         ja      f_io1                          ; drive out of range.
 EC74  80 FC 05                C         cmp     ah,5                           ; max. command.
 EC77  76 07                   C         jbe     diskette_io1                   ; command in range
 EC79                          C f_io1:
 EC79  C6 06 0041 R 01         C         mov     diskette_status,cmd_error              ; 01h
 EC7E  EB 1B                   C         jmp     short f_io_ret                 ; quick return
 EC80                          C diskette_io1:
 EC80  FC                      C         cld                                    ; Autoincrement for strings.
 EC81  E8 EFAD R               C         call    f_check_valid                  ; Will not return if error.
 EC84  32 FF                   C         xor     bh,bh                          ; clear high byte
 EC86  8A DC                   C         mov     bl,ah                          ; move selection into low byte
 EC88  D1 E3                   C         shl     bx,1                           ; multiply by 2
 EC8A  2E: FF A7 EC8F R        C         jmp     cs:[f_table.bx]
                               C
                               C ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                               C
 EC8F                          C f_table label   word
                               C
 EC8F  ECBF R                  C         dw      f_reset                        ; AH = 0 (reset)
 EC91  EC9B R                  C         dw      f_io_ret                       ; AH = 1 (status)
 EC93  ED65 R                  C         dw      f_rdata                        ; AH = 2 (read)
 EC95  ED39 R                  C         dw      f_wdata                        ; AH = 3 (write)
 EC97  ED65 R                  C         dw      f_rdata                        ; AH = 4 (verify)
 EC99  ED39 R                  C         dw      f_wdata                        ; AH = 5 (format)
                               C
                               C ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                               C
 EC9B                          C f_io_ret:
                               C ; f_nec_reset is located in the module called special.src.
 EC9B  E8 EC18 R               C         call    f_nec_reset                    ; reset if necessary.
 EC9E  BB 0002                 C         mov     bx,2                           ; motor_wait parameter.
 ECA1  E8 EDFF R               C         call    f_get_var                      ; Returns 0 in AH.
 ECA4  A2 0040 R               C         mov     motor_count,al                 ; motor shut off value.
                               C
 ECA7  32 C0                   C         xor     al,al                          ; zero AL.
 ECA9  8A 26 0041 R            C         mov     ah,diskette_status
                               C
 ECAD  80 FC 01                C         cmp     ah,1
 ECB0  F5                      C         cmc
                               C
 ECB1  8B E5                   C         mov     sp,bp                          ; for safety sake.
 ECB3  5B                      C         pop     bx                             ; discard DX.
 ECB4  5B                      C         pop     bx                             ; discard AX.
 ECB5  5B                      C         pop     bx
 ECB6  59                      C         pop     cx
 ECB7  5A                      C         pop     dx
 ECB8  5F                      C         pop     di
 ECB9  5E                      C         pop     si
 ECBA  1F                      C         pop     ds
 ECBB  5D                      C         pop     bp
                               C
 ECBC  CA 0002                 C         ret     2
                               C
 ECBF                          C fd_io   endp
```

```
                              C
                              C ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                              C ;
                              C ;       Reset and reprogram the FDC without turning the motors off.
                              C ;
                              C ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                              C
 ECBF                         C f_reset proc    near
                              C
                              C
 ECBF  FA                     C         cli                             ; disable interrupts
                              C
 ECC0  B0 66                  C         mov     al,pic_seoi_6           ; specific end of interrupt
 ECC2  BA 0020                C         mov     dx,pic_0                ; to pic `command' port.
 ECC5  EE                     C         out     dx,al
                              C
 ECC6  42                     C         inc     dx                      ; pic `data' port.
 ECC7  EC                     C         in      al,dx                   ; read mask.
 ECC8  24 BF                  C         and     al,10111111b            ; enable IR6.
 ECCA  EE                     C         out     dx,al
                              C
                              C ; Develop mask for motor control port.
                              C
 ECCB  A0 003F R              C         mov     al,motor_status         ; which motor is running?
 ECCE  24 0F                  C         and     al,0Fh                  ; blow off high 4 bits.
 ECD0  74 0E                  C         jz      f_r2                    ; no motors running.
 ECD2  8A E0                  C         mov     ah,al
 ECD4  B1 04                  C         mov     cl,4
 ECD6  D2 E0                  C         shl     al,cl                   ; move to high nib.
                              C
                              C ; A motor is on.  Find out which one.
                              C
 ECD8                         C f_r1:
 ECD8  D0 EC                  C         shr     ah,1                    ; determine drive select
 ECDA  72 04                  C         jc      f_r2                    ; from motor enable bit.
 ECDC  FE C0                  C         inc     al
 ECDE  EB F8                  C         jmp     short f_r1
 ECE0                         C f_r2:
                              C
                              C ; Reset signal has to be maintained for at least 14 clocks.
                              C
 ECE0  C6 06 0041 R 00        C         mov     diskette_status,0
 ECE5  0C 08                  C         or      al,8                    ; set bit 3.
 ECE7  BA 03F2                C         mov     dx,f_motor_port
 ECEA  EE                     C         out     dx,al                   ; send reset signal.
 ECEB  0C 04                  C         or      al,4                    ; set bit 2.
 ECED  C6 06 003E R 00        C         mov     seek_status,0
 ECF2  EE                     C         out     dx,al                   ; clear reset.
 ECF3  FB                     C         sti
 ECF4  E8 EF82 R              C         call    f_sis                   ; sense int. status
                              C
 ECF7  B4 03                  C         mov     ah,f_specify_cmd
 ECF9  E8 EE0C R              C         call    f_put_byte
 ECFC  BB 0000                C         mov     bx,0                    ; 1st specify byte.
 ECFF  E8 EDFF R              C         call    f_get_var
 ED02  8A E0                  C         mov     ah,al
 ED04  E8 EE0C R              C         call    f_put_byte
 ED07  BB 0001                C         mov     bx,1                    ; 2nd specify byte.
 ED0A  E8 EDFF R              C         call    f_get_var
 ED0D  8A E0                  C         mov     ah,al
 ED0F  E8 EE0C R              C         call    f_put_byte
                              C
 ED12  EB 87                  C         jmp     f_io_ret
                              C
 ED14                         C f_reset endp
                              C
                              C ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                              C ;
                              C ;       (f_motor_on)
                              C ;
                              C ;       INPUT:          none
                              C ;
                              C ;       OUTPUT:         Carry set if motor was off, cleared otherwise.
                              C ;
                              C ;       DESTROYS:       AX, CX
                              C ;
                              C ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                              C
 ED14                         C f_motor_on      proc    near
                              C
                              C
 ED14  C6 06 0040 R FF        C         mov     motor_count,0FFh        ; max. time for motor.
 ED19  8A 4E 00               C         mov     cl,f_drive              ; drive.
 ED1C  B0 01                  C         mov     al,1
 ED1E  D2 E0                  C         shl     al,cl                   ; mask for motor status.
 ED20  84 06 003F R           C         test    al,motor_status         ; test always clears carry.
 ED24  75 12                  C         jnz     f_mo_ret                ; already running.
 ED26  A2 003F R              C         mov     motor_status,al         ; set correct bit.
 ED29  8A E1                  C         mov     ah,cl                   ; drive into AH.
 ED2B  B1 04                  C         mov     cl,4
 ED2D  D2 E0                  C         shl     al,cl                   ; motor on bit to high nibble.
 ED2F  0C 0C                  C         or      al,0Ch                  ; set bits 2 & 3 (0000 1100).
 ED31  0A C4                  C         or      al,ah                   ; drive bits ( 0 & 1).
 ED33  BA 03F2                C         mov     dx,f_motor_port
 ED36  EE                     C         out     dx,al                   ; turn on the motor.
 ED37  F9                     C         stc                             ; motor was off indicator.
                              C
 ED38                         C f_mo_ret:
 ED38  C3                     C         ret
                              C
```

```
ED39                          C  f_motor_on       endp
                              C
                              C  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                              C
ED39                          C  f_wdata proc     near
                              C
                              C
ED39  E8 ED14 R               C          call     f_motor_on                ; sets carry if motor was off.
ED3C  73 1E                   C          jnc      f_wd1                     ; motor was on, skip delay.
                              C
ED3E  E4 67                   C          in       al,sys_conf_b             ; read switch.
ED40  F6 D0                   C          not      al                        ; toggle the sense.
ED42  24 02                   C          and      al,2                      ; isolate slow motor bit.
ED44  8A C8                   C          mov      cl,al
ED46  BA 007D                 C          mov      dx,125                    ; 125 ms delay to start with.
ED49  D3 E2                   C          shl      dx,cl                     ; 125 x 4
ED4B  BB 000A                 C          mov      bx,10                     ; motor start delay parameter.
ED4E  E8 EDFF R               C          call     f_get_var                 ; returns param. in AL.
ED51  32 E4                   C          xor      ah,ah                     ; for good measure.
ED53  F7 E2                   C          mul      dx                        ; AX has total delay
ED55  8B C8                   C          mov      cx,ax
ED57                          C  f_wd_loop:
ED57  E8 EF47 R               C          call     f_wait_one_ms
ED5A  E2 FB                   C          loop     f_wd_loop
                              C
ED5C                          C  f_wd1:
ED5C  80 0E 003F R 80         C          or       motor_status,080h         ; set high bit, indicate write.
ED61  B0 4A                   C          mov      al,04Ah                   ; DMA mode byte: channel 2,
                              C                                             ; single mode, read transfer
ED63  EB 22                   C          jmp      short f_rw_common
ED65                          C  f_wdata endp
                              C
                              C  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                              C
ED65                          C  f_rdata proc     near
                              C
                              C
ED65  80 26 003F R 7F         C          and      motor_status,07Fh         ; clear high bit, indicate read
ED6A  E8 ED14 R               C          call     f_motor_on
ED6D  73 0E                   C          jnc      f_rd1                     ; motor was on, no delay.
                              C
                              C  ; slow motor delay loop
                              C
ED6F  E4 67                   C          in       al,sys_conf_b             ; read configuration switch(7W)
ED71  A8 02                   C          test     al,2                      ; bit 1 = slow drive bit
ED73  75 08                   C          jnz      f_rd1                     ; 1 = 250 ms motors, no delay
ED75  B9 01F4                 C          mov      cx,500                    : approx. 500 ms delay for
                              C                                             ; slow motors
ED78                          C  f_rd_loop:
ED78  E8 EF47 R               C          call     f_wait_one_ms
ED7B  E2 FB                   C          loop     f_rd_loop
                              C
ED7D                          C  f_rd1:
ED7D  B0 46                   C          mov      al,046h                   ; DMA mode byte: channel 2,
                              C                                             ; single mode, write transfer.
ED7F  80 7E 03 04             C          cmp      byte ptr f_command,4      ; Is it a verify command?
ED83  75 02                   C          jne      f_rw_common               ; No, must have been a read.
ED85  B0 42                   C          mov      al,042h                   ; DMA mode byte: channel 2,
                              C                                             ; single mode, verify transfer.
                              C
ED87                          C  f_rdata endp
                              C
                              C  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                              C  ;
                              C  ;       Common  (f_rw_common)
                              C  ;
                              C  ;       INPUT:          AL       dma mode byte.
                              C  ;
                              C  ;       OUTPUT:
                              C  ;
                              C  ;       DESTROYS:
                              C  ;
                              C  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                              C
ED87                          C  f_rw_common      proc     near
                              C
                              C
ED87  C6 06 0040 R FF         C          mov      motor_count,0FFh          ; long wait.
ED8C  E8 EE73 R               C          call     f_set_dma                 ; pass mode byte on through.
                              C
                              C  ; Clear out status from previous operation.
                              C
ED8F  06                      C          push     es
ED90  1E                      C          push     ds
ED91  07                      C          pop      es
ED92  32 C0                   C          xor      al,al
ED94  B9 0007                 C          mov      cx,7
ED97  BF 0042 R               C          mov      di,offset nec_status
ED9A  F3/ AA                  C          rep      stosb
ED9C  07                      C          pop      es
                              C
ED9D  E8 EECD R               C          call     f_seek                    ; On return, f_drive has
                              C                                             ; head bit or'd in.
                              C
                              C
EDA0  8A 56 03                C          mov      dl,byte ptr f_command     ; get command
EDA3  B4 C5                   C          mov      ah,f_write_cmd
EDA5  80 FA 03                C          cmp      dl,3                      ; Is it a write command?
EDA8  74 09                   C          je       f_rw1                     ; yes
EDAA  B4 4D                   C          mov      ah,f_format_cmd
EDAC  80 FA 05                C          cmp      dl,5                      ; Is it a format command?
```

```
EDAF  74 02               C          je      f_rw1                           ; yes
EDB1  B4 E6               C          mov     ah,f_read_cmd                   ; must be read or verify.
EDB3                      C  f_rw1:
EDB3  E8 EE0C R           C          call    f_put_byte                      ; send command.
EDB6  8A 66 00            C          mov     ah,f_drive                      ; has head and drive bits.
EDB9  E8 EE0C R           C          call    f_put_byte
EDBC  80 7E 03 05         C          cmp     byte ptr f_command,5            ; was it a format command?
EDC0  74 15               C          je      f_rw_skip                       ; yes, skip next 3 params.
EDC2  8A 66 07            C          mov     ah,f_cyl
EDC5  E8 EE0C R           C          call    f_put_byte
EDC8  8A 66 01            C          mov     ah,f_head
EDCB  80 E4 01            C          and     ah,1                            ; blow off high 7 bits.
EDCE  E8 EE0C R           C          call    f_put_byte
EDD1  8A 66 06            C          mov     ah,f_secnum
EDD4  E8 EE0C R           C          call    f_put_byte
                          C
                          C  ; Get bytes 3,4,5,6 from table.
                          C  ; If we are formatting then we need bytes 3,4,7,8 from table.
                          C
EDD7                      C  f_rw_skip:
EDD7  B9 0004             C          mov     cx,4
EDDA  BB 0003             C          mov     bx,3                            ; no. bytes per sector.
EDDD                      C  f_rw2:
EDDD  E8 EDFF R           C          call    f_get_var
EDE0  8A E0               C          mov     ah,al
EDE2  E8 EE0C R           C          call    f_put_byte
EDE5  43                  C          inc     bx
EDE6  83 FB 05            C          cmp     bx,5                            ; time to check for format?
EDE9  75 09               C          jne     f_rw3                           ; No.
EDEB  80 7E 03 05         C          cmp     byte ptr f_command,5            ; was it a format command?
EDEF  75 03               C          jne     f_rw3                           ; no.
EDF1  BB 0007             C          mov     bx,7                            ; 7th parameter in table.
EDF4                      C  f_rw3:
EDF4  E2 E7               C          loop    f_rw2
                          C
EDF6  E8 EF8D R           C          call    f_wait_for_nec
EDF9  E8 EE29 R           C          call    f_get_byte                      ; get the results.
                          C
EDFC                      C  f_rw_ret:
EDFC  E9 EC9B R           C          jmp     f_io_ret
                          C
EDFF                      C  f_rw_common     endp
                          C
                          C  ;--------------------------------------------------------------------------------
                          C  ;
                          C  ;       Get specified byte from fdu parameter table  (f_get_var)
                          C  ;
                          C  ;       INPUT:          BX      parameter number (0 - 10)
                          C  ;
                          C  ;       OUTPUT:         AL      The requested byte.
                          C  ;
                          C  ;       DESTROYS:       AH
                          C  ;
                          C  ;--------------------------------------------------------------------------------
                          C
EDFF                      C  f_get_var       proc    near
                          C
EDFF  1E                  C          push    ds
EE00  33 C0               C          xor     ax,ax
EE02  8E D8               C          mov     ds,ax                           ; segment 0
                          C
                          C          assume  ds:abs0                         ; tell assembler seg 0:
                          C
EE04  C5 36 0078 R        C          lds     si,dword ptr [int1Eloc]         ; DS:SI points to table
EE08  8A 00               C          mov     al,[bx+si]
EE0A  1F                  C          pop     ds
                          C
                          C          assume  ds:data                         ; tell assembler seg 40:
EE0B  C3                  C          ret
                          C
EE0C                      C  f_get_var       endp
                          C
                          C  ;--------------------------------------------------------------------------------
                          C  ;
                          C  ;       Send a byte to the NEC controller  (f_put_byte)
                          C  ;
                          C  ;       INPUT:          AH      byte to output.
                          C  ;
                          C  ;       OUTPUT:
                          C  ;
                          C  ;--------------------------------------------------------------------------------
                          C
EE0C                      C  f_put_byte      proc    near
                          C
EE0C  E8 EF6B R           C          call    f_nec_rdy                       ; returns MSR byte in AL.
EE0F  A8 40               C          test    al,40h                          ; direction bit.
                          C  .list
EE11  75 05               C          jnz     f_pb_erret                      ; wrong direction.
EE13  42                  C          inc     dx                              ; NEC data port 3F5h
EE14  8A C4               C          mov     al,ah
EE16  EE                  C          out     dx,al
                          C
EE17                      C  f_pb_ret:
EE17  C3                  C          ret
                          C
EE18                      C  f_pb_erret:
EE18  C6 06 0041 R 20     C          mov     diskette_status,fdc_error
EE1D  E9 EC9B R           C          jmp     f_io_ret
                          C
EE20                      C  f_put_byte      endp
```

```
                        C
                        C ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                        C ;
                        C ;      Read the results bytes from the NEC controller (fdu_data1)
                        C ;
                        C ;      INPUT:          none
                        C ;
                        C ;      OUTPUT:
                        C ;
                        C ;
                        C ;      DESTROYS:       DX, SI
                        C ;
                        C ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                        C
EE20                    C fdu_data1       proc
                        C
EE20                    C f_gb_table      label   byte
EE20  04                C         db      4                       ; record not found.
EE21  00                C         db      0                       ; dummy
EE22  10                C         db      10h                     ; crc error.
EE23  08                C         db      8                       ; dma error.
EE24  00                C         db      0                       ; dummy.
EE25  04                C         db      4                       ; sector not found.
EE26  03                C         db      3                       ; write protect.
EE27  02                C         db      2                       ; address mark error.
EE28  20                C         db      20h                     ; fdc error.
                        C
EE29                    C fdu_data1       endp
                        C
EE29                    C f_get_byte      proc    near
                        C
EE29  06                C         push    es
EE2A  1E                C         push    ds
EE2B  07                C         pop     es
EE2C  BF 0042 R         C         mov     di,offset nec_status    ; ES:DI is now ready for stosb.
EE2F  8B F7             C         mov     si,di                   ; offset of nec_status in SI.
EE31  B9 0007           C         mov     cx,7                    ; maximum no. of bytes.
EE34                    C f_gb_loop:
EE34  E8 EF6B R         C         call    f_nec_rdy               ; Returns MSR byte in AL.
                        C                                         ; will not return if error.
EE37  A8 10             C         test    al,10h                  ; busy bit.
EE39  74 36             C         jz      f_gb_ret                ; done.
EE3B  A8 40             C         test    al,40h                  ; direction bit.
EE3D  74 05             C         jz      f_gb_out                ; wrong direction.
EE3F  42                C         inc     dx                      ; point to nec data port (3F5h)
EE40  EC                C         in      al,dx
EE41  AA                C         stosb
EE42  E2 F0             C         loop    f_gb_loop
EE44                    C f_gb_out:
EE44  80 7E 03 00       C         cmp     byte ptr f_command,0    ; was it reset command?
EE48  74 27             C         je      f_gb_ret                ; yes.
EE4A  AC                C         lodsb                           ; get ST0 in AL.
EE4B  A8 20             C         test    al,20h                  ; seek end?
EE4D  75 22             C         jnz     f_gb_ret                ; yes.
EE4F  A8 C0             C         test    al,0C0h
EE51  74 1A             C         jz      f_gb_jmp
                        C
EE53  C6 06 0041 R 20   C         mov     diskette_status,fdc_error
EE58  AC                C         lodsb                           ; get ST1 in AL.
EE59  B9 0008           C         mov     cx,8
EE5C  33 DB             C         xor     bx,bx
EE5E                    C f_gb_loop1:
EE5E  D0 C0             C         rol     al,1
EE60  72 03             C         jc      f_gb_decode
EE62  43                C         inc     bx
EE63  E2 F9             C         loop    f_gb_loop1
EE65                    C f_gb_decode:
EE65  2E: 8A 87 EE20 R  C         mov     al,cs:f_gb_table[bx]
EE6A  A2 0041 R         C         mov     diskette_status,al
EE6D                    C f_gb_jmp:
EE6D  07                C         pop     es
EE6E  E9 EC9B R         C         jmp     f_io_ret
                        C
EE71                    C f_gb_ret:
EE71  07                C         pop     es
EE72  C3                C         ret
                        C
EE73                    C f_get_byte      endp
                        C
                        C ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                        C ;
                        C ;      Test for boundary crossing case then program the
                        C ;      DMA controller (f_set_dma).
                        C ;
                        C ;      INPUT:          AL      mode byte
                        C ;                      ES      segment of user buffer.
                        C ;
                        C ;      OUTPUT:
                        C ;
                        C ;      DESTROYS:       AX, BX, CX, DX
                        C ;
                        C ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                        C
EE73                    C f_set_dma       proc    near
                        C
EE73  C6 06 0041 R 09   C         mov     diskette_status,dma_seg_error
EE78  E6 0C             C         out     dma_ff_clr,al           ; set to known state.
EE7A  E6 0B             C         out     dma_mode,al             ; send mode byte.
EE7C  8B 5E 04          C         mov     bx,word ptr f_bufoff    ; offset of user buffer.
EE7F  8C C0             C         mov     ax,es                   ; offset of user buffer.
```

```
EE81  B1 04              C          mov     cl,4                        ; shift count
EE83  D3 C0              C          rol     ax,cl                       ; move high nib around.
EE85  8A E8              C          mov     ch,al                       ; save high nib.
EE87  80 E5 0F           C          and     ch,0Fh                      ; isolate high nib
EE8A  24 F0              C          and     al,0F0h                     ; prepare for offset calculation
EE8C  03 D8              C          add     bx,ax                       ; 20 bit calculation.
EE8E  73 02              C          jnc     f_sd1
EE90  FE C5              C          inc     ch                          ; high nib
EE92                     C  f_sd1:
EE92  8A C5              C          mov     al,ch
EE94  E6 81              C          out     dma_segm_2,al               ; high nib. to latch
EE96  B0 06              C          mov     al,6
EE98  E6 0A              C          out     dma_mask_bit,al             ; disable channel 2.
EE9A  8A C3              C          mov     al,bl
EE9C  E6 04              C          out     dma_addr_2,al               ; low byte of address.
EE9E  8A C7              C          mov     al,bh
EEA0  E6 04              C          out     dma_addr_2,al               ; high byte of address.
EEA2  8B D3              C          mov     dx,bx                       ; save buffer offset.
EEA4  BB 0003            C          mov     bx,3                        ; bytes/sector param
EEA7  E8 EDFF R          C          call    f_get_var                   ; get param from table
EEAA  8A C8              C          mov     cl,al                       ; save for shift count
EEAC  8A 66 02           C          mov     ah,f_numsecs                ; get #sectors.
EEAF  32 C0              C          xor     al,al                       ; AX = #sectors x 256
EEB1  D1 E8              C          shr     ax,1                        ; AX = #sectors x 128
EEB3  D3 E0              C          shl     ax,cl                       ; multiply by bytes/sector.
EEB5  48                 C          dec     ax                          ; DMAC counts zero.
EEB6  03 D0              C          add     dx,ax                       ; add count to offset
EEB8  73 03              C          jnc     f_sd2
EEBA  E9 EC9B R          C          jmp     f_io_ret                    ; exit boundary crossing.
EEBD                     C  f_sd2:
EEBD  E6 05              C          out     dma_count_2,al              ; low byte of count.
EEBF  8A C4              C          mov     al,ah
EEC1  E6 05              C          out     dma_count_2,al              ; high byte of count.
EEC3  B0 02              C          mov     al,2
EEC5  E6 0A              C          out     dma_mask_bit,al             ; enable channel 2
EEC7  C6 06 0041 R 00    C          mov     diskette_status,0
EECC                     C  f_sd_ret:
EECC  C3                 C          ret
EECD                     C  f_set_dma       endp
                         C
                         C  ;--------------------------------------------------------------------------------
                         C  ;
                         C  ;       Seek  (f_seek)
                         C  ;
                         C  ;       INPUT:
                         C  ;
                         C  ;       OUTPUT:
                         C  ;
                         C  ;       DESTROYS:
                         C  ;
                         C  ;--------------------------------------------------------------------------------
                         C
EECD                     C  f_seek  proc    near
                         C
EECD  8A 4E 00           C          mov     cl,f_drive                  ; get drive# as shift count.
EED0  B0 01              C          mov     al,1
EED2  D2 E0              C          shl     al,cl                       ; mask for recal.
EED4  84 06 003E R       C          test    seek_status,al
EED8  75 28              C          jnz     f_s1                        ; no recal. required.
EEDA  08 06 003E R       C          or      seek_status,al              ; set the corresponding bit.
                         C
                         C  ; Two recalibrate commands required in the case of 96 TPI drives.
                         C
EEDE  B9 0002            C          mov     cx,2                        ; loop count for 96 TPI
EEE1                     C  f_s_recal:
EEE1  B4 07              C          mov     ah,f_recal_cmd              ; recal. command
EEE3  51                 C          push    cx                          ; save it.
EEE4  E8 EE0C R          C          call    f_put_byte
EEE7  8A 66 00           C          mov     ah,f_drive                  ; drive bit.
EEEA  E8 EE0C R          C          call    f_put_byte                  ; end of command phase.
EEED  E8 EF82 R          C          call    f_sis                       ; Sense Interrupt Status
EEF0  59                 C          pop     cx                          ; restore it.
                         C
                         C  ; possibly test bit 4 as well (equipment check...track 0 not reached)
EEF1  F6 06 0042 R C0    C          test    nec_status,0C0h
EEF6  74 0A              C          jz      f_s1                        ; equipment OK so restore.
EEF8  E2 E7              C          loop    f_s_recal
EEFA  C6 06 0041 R 40    C          mov     diskette_status,seek_error
EEFF  E9 EC9B R          C          jmp     f_io_ret
                         C
EF02                     C  f_s1:
EF02  B4 0F              C          mov     ah,f_seek_cmd
EF04  E8 EE0C R          C          call    f_put_byte
                         C  ; prepare second byte of seek command.
EF07  8A 46 01           C          mov     al,f_head
EF0A  24 01              C          and     al,1                        ; blow off high 7 bits.
EF0C  D0 E0              C          shl     al,1                        ; move head to bit 2.
EF0E  D0 E0              C          shl     al,1
EF10  0A 46 00           C          or      al,f_drive                  ; combine drive bit with head
EF13  88 46 00           C          mov     f_drive,al                  ; and save it.
EF16  8A E0              C          mov     ah,al
EF18  E8 EE0C R          C          call    f_put_byte
EF1B  8A 66 07           C          mov     ah,f_cyl
EF1E  F6 46 01 80        C          test    byte ptr f_head,80h         ; test 80 track bit.
EF22  74 02              C          jz      f_cont
EF24  D0 E4              C          shl     ah,1                        ; cyl x 2
EF26                     C  f_cont:
EF26  E8 EE0C R          C          call    f_put_byte                  ; end of seek command.
EF29  E8 EF82 R          C          call    f_sis                       ; Sense Interrupt Status
                         C
```

```
EF2C  F6 06 0042 R C0      C          test    nec_status,0C0h           ; test for seek end.
                           C
EF31  75 13                C          jnz     f_s_ret
EF33  BB 0009              C          mov     bx,9                      ; head settle parameter.
EF36  E8 EDFF R            C          call    f_get_var
EF39  0A C0                C          or      al,al
EF3B  74 09                C          jz      f_s_ret                   ; head settle = 0.
EF3D  8A C8                C          mov     cl,al                     ; use settle parm. al loop index
EF3F  32 ED                C          xor     ch,ch
EF41                       C  f_head_settle:
EF41  E8 EF47 R            C          call    f_wait_one_ms
EF44  E2 FB                C          loop    f_head_settle
EF46                       C  f_s_ret:
EF46  C3                   C          ret
                           C
EF47                       C  f_seek  endp
                           C
                           C  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                           C  ;
                           C  ;       One millisecond delay loop (approximately)
                           C  ;
                           C  ;               The CALL is 19 clocks.
                           C  ;               The callers LOOP statement is 17 clocks.
                           C  ;               No flags affected.
                           C  ;
                           C  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                           C
EF47                       C  f_wait_one_ms   proc    near
EF47  51                   C          push    cx                        ; 11 clocks
EF48  B9 0176              C          mov     cx,374                    ; 4 clocks
EF4B  E2 FE                C  w_one:  loop    w_one                     ; (17 x CX) + 5 clocks
EF4D  59                   C          pop     cx                        ; 8 clocks
EF4E  C3                   C          ret                               ; 16 clocks
EF4F                       C  f_wait_one_ms   endp
                           C
                           C  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                           C  ;
                           C  ; This routine is called by the timer_int routine when motor_count = 0
                           C  ;
                           C  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                           C
EF4F                       C  stop_disk       proc    near
                           C
EF4F  B0 0C                C          mov     al,0Ch
EF51  BA 03F2              C          mov     dx,f_motor_port
EF54  EE                   C          out     dx,al
EF55  C3                   C          ret
EF56                       C  stop_disk       endp
                           C
                           C  include fdu2.asm
                           C
                           C  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                           C  ;
                           C  ;
                           C  ;       INPUT:          none
                           C  ;
                           C  ;       OUTPUT:         MSB of seek_status is set if NEC interrupts.
                           C  ;
                           C  ;       DESTROYS:       nothing
                           C  ;
                           C  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

```
                               C
EF57                           C                ORG     0EF57h
                               C
EF57                           C  fd_int  proc    near
                               C
                               C
EF57  FB                       C          sti                               ; enable interupts
EF58  50                       C          push    ax
EF59  1E                       C          push    ds
EF5A  B0 66                    C          mov     al,pic_seoi_6             ; specific end of interupt
EF5C  E6 20                    C          out     pic_0,al                  ; send to 8259
EF5E  2E: 8E 1E E5F2 R         C          mov     ds,word ptr cs:[set_ds_word]  ; set DS to segment 40h
EF63  80 0E 003E R 80          C          or      seek_status,80h           ; interupt indicator
EF68  1F                       C          pop     ds
EF69  58                       C          pop     ax
EF6A  CF                       C          iret
EF6B                           C  fd_int  endp
                               C
                               C  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                               C  ;
                               C  ;       NEC ready (f_nec_rdy)
                               C  ;
                               C  ;       INPUT:          none
                               C  ;
                               C  ;       OUTPUT:         AL          Main Status Register byte.
                               C  ;                       DX          Points to port 3F4h
                               C  ;
                               C  ;       DESTROYS:
                               C  ;
                               C  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                               C
EF6B                           C  f_nec_rdy       proc    near
                               C
EF6B  51                       C          push    cx
EF6C  33 C9                    C          xor     cx,cx
EF6E                           C  f_nr1:
EF6E  BA 03F4                  C          mov     dx,f_nec_status           ; Main status register(3F4h)
EF71  EC                       C          in      al,dx
EF72  A8 80                    C          test    al,080h                   ; RQM
EF74  75 0A                    C          jnz     f_nr_ret                  ; NEC is ready.
EF76  E2 F6                    C          loop    f_nr1
EF78  C6 06 0041 R 80          C          mov     diskette_status,time_out
EF7D  E9 EC9B R                C          jmp     f_io_ret                  ; Took too long to respond.
EF80                           C  f_nr_ret:
EF80  59                       C          pop     cx
EF81  C3                       C          ret
                               C
EF82                           C  f_nec_rdy       endp
                               C
                               C  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                               C  ;
                               C  ;       Sense Interrupt Status  (f_sis).
                               C  ;
                               C  ;       INPUT:          none
                               C  ;
                               C  ;       OUTPUT:
                               C  ;
                               C  ;       DESTROYS:
                               C  ;
                               C  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                               C
EF82                           C  f_sis   proc    near
                               C
EF82  E8 EF8D R                C          call    f_wait_for_nec            ; no return on error.
EF85  B4 08                    C          mov     ah,f_snsint_cmd           ; end of command phase.
EF87  E8 EE0C R                C          call    f_put_byte                ; no return on error.
EF8A  E9 EE29 R                C          jmp     f_get_byte
                               C
EF8D                           C  f_sis   endp
                               C
                               C  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                               C  ;
                               C  ;       Wait for NEC to interupt on completion of execution phase or
                               C  ;       time out if NEC never interupts (f_wait_for_nec).
                               C  ;
                               C  ;       INPUTS:         none
                               C  ;
                               C  ;       OUTPUTS:
                               C  ;
                               C  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                               C
EF8D                           C  f_wait_for_nec  proc    near
                               C
EF8D  FB                       C          sti                               ; enable interupts
EF8E  51                       C          push    cx
EF8F  B9 03E8                  C          mov     cx,1000                   ; wait a while
EF92                           C  w_nec:
EF92  F6 06 003E R 80          C          test    seek_status,80h           ; test MSB (int_flag)
EF97  75 0D                    C          jnz     w_nec_ret
EF99  E8 EF47 R                C          call    f_wait_one_ms
EF9C  E2 F4                    C          loop    w_nec
EF9E  C6 06 0041 R 80          C          mov     diskette_status,time_out
EFA3  E9 EC9B R                C          jmp     f_io_ret
EFA6                           C  w_nec_ret:
EFA6  80 26 003E R 7F          C          and     seek_status,07Fh          ; clear MSB.
EFAB  59                       C          pop     cx
EFAC  C3                       C          ret
                               C
EFAD                           C  f_wait_for_nec  endp
```

```
                         C
                         C  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                         C  ;
                         C  ;       Check Valid (f_check_valid)
                         C  ;
                         C  ;       This routine determines the validity of the request based on
                         C  ;       drive type (40 or 80 trk) and media type (40 or 80 trk)
                         C  ;
                         C  ;       Bit 7 of sys_conf_b port:       0 = 40 trk drive, 1 = 80 trk drive
                         C  ;       Bit 7 of head parameter(DH):    0 = 40 trk media, 1 = 80 trk media
                         C  ;
                         C  ;                                  drive
                         C  ;                             |-----------|
                         C  ;                             |  0  |  1  |
                         C  ;                          ---|-----------|
                         C  ;                          | 0 | r/w | r/w |
                         C  ;                  media    ---|-----------|
                         C  ;                          | 1 | n/a | r/w |
                         C  ;                          ---|-----------|
                         C  ;
                         C  ;       OUTPUT:         The MSB of the drive parameter at bp+1 is set
                         C  ;                       if the drive is 80 track and the media is 40 track.
                         C  ;                       This is used by seek to determine if double stepping
                         C  ;                       is necessary.
                         C  ;
                         C  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                         C
EFAD                     C  f_check_valid    proc    near
                         C
EFAD  50                 C          push    ax
EFAE  E4 67              C          in      al,sys_conf_b           ; switch port
EFB0  24 01              C          and     al,1                    ; blow off high 7 bits
EFB2  D0 C8              C          ror     al,1                    ; move it into the high bit
EFB4  30 46 01           C          xor     f_head,al               ; head param. (xor clears carry)
EFB7  79 0C              C          jns     f_cv_ret                ; media & drive agree
EFB9  A8 80              C          test    al,80h                  ; Is drive 80 track.
EFBB  75 08              C          jnz     f_cv_ret                ; It was.
EFBD  C6 06 0041 R 01    C          mov     diskette_status,cmd_error
EFC2  E9 EC9B R          C          jmp     f_io_ret
EFC5                     C  f_cv_ret:
EFC5  58                 C          pop     ax
EFC6  C3                 C          ret
                         C
EFC7                     C  f_check_valid    endp
                         C
                         C  include fdu3.asm
                         C
                         C  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                         C  ; fd_parms
                         C  ;       This is the set of parameters required for diskette operation.
                         C  ;       They are pointed to by interrupt vector 1Eh (0:78h). To modify
                         C  ;       the parameters, build another parameter block and point disk_pointer
                         C  ;       to it.
                         C  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                         C
EFC7                     C          ORG     0EFC7h
                         C
EFC7                     C  fdu_data2       proc
                         C
EFC7                     C  fd_parms        label   byte
                         C
EFC7  CF                 C          db      (f_srt_48 shl 4) + f_hut        ; SRT + HUT
EFC8  02                 C          db      (f_hlt shl 1) + f_ndma          ; HLT + DMA mode
EFC9  25                 C          db      f_motor_wait                    ; motor off delay
EFCA  02                 C          db      2                               ; 512 bytes per sector
EFCB  08                 C          db      8                               ; EOT last sector on cyl
EFCC  2A                 C          db      02Ah                            ; gap length
EFCD  FF                 C          db      0FFh                            ; DTL
EFCE  50                 C          db      050h                            ; gap length for format
EFCF  F6                 C          db      0F6h                            ; fill byte for format
EFD0  19                 C          db      25                              ; head settle time (ms)
EFD1  04                 C          db      4                               ; motor start time
                         C
EFD2                     C  fdu_data2       endp
                         C
EFD2                     C  code    ends
                         C  include prt.asm
                         C
                         C  ;=====================================================================
                         C  ;       Filename:       prt.src
                         C  ;
                         C  ;       This module includes INT 17h.
                         C  ;
                         C  ;=====================================================================
                         C
EFD2                     C  code    segment public 'ROM'
                         C          assume  cs:code, ds:nothing, es:nothing, ss:nothing
                         C
                         C  ;=====================================================================
                         C  ;       INT 17h -- Printer Software Interrupt Request Routine
                         C  ;
                         C  ;       Input:  ah = 0  print character
                         C  ;               al =    character to print
                         C  ;               dx =    printer port number (0,1,2,3)
                         C  ;       Output: ah =    status of printer: bit #0 set if time out
                         C  ;               al =    character to print (preserved)
                         C  ;
                         C  ;       Input:  ah = 1  initialize the printer port
                         C  ;               dx =    printer port number (0,1,2,3)
                         C  ;       Output: ah =    status of printer
```

```
C  ;               al      preserved
C  ;
C  ;       Input:  ah = 2  read the printer status
C  ;               dx =    printer port number (0,1,2,3)
C  ;       Output: ah =    status of printer
C  ;
C  ;       Trash:  ah =    (ah - 2) if ah > 2
C  ;               al      preserved
C  ;
C  ;       Assumes:        prt_stat_x = prt_data_x + 1 = dx + 1
C  ;                       prt_cmd_x  = prt_data_x + 2 = dx + 2
C  ;
C  ; Printer Status Byte from prt_stat_x:
C  ;
C  ;       bit     #0 =    time-out on printing character (p_out).
C  ;                       set by software; no hardware significance.
C  ;       bits  #1-2 =    no significance (always cleared).
C  ;       bit     #3 =    hardware: I/O error.
C  ;       bit     #4 =    hardware: selected.
C  ;       bit     #5 =    hardware: out of paper.
C  ;       bit     #6 =    hardware: acknowledge.
C  ;       bit     #7 =    hardware: not busy.
C  ;
C  ; "Good" Statuses are:  90h & 10h if a printer is connected.
C  ;                       30h if printer is disconnected.
C  ;=============================================================================
C
EFD2                    C          ORG     0EFD2h
C
EFD2                    C  p_io    proc    near
C                       C          assume  cs:code, ds:nothing, es:nothing, ss:nothing
C
EFD2  FB                C          sti                                         ; enable interrupts
C
EFD3  83 FA 04          C          cmp     dx,4                                ; 4 printers allowed max
EFD6  73 32             C          jae     p_nop
C
C                       C          assume  cs:code, ds:data, es:nothing, ss:nothing
C
EFD8  51                C          push    cx                                  ; save registers
EFD9  52                C          push    dx
EFDA  57                C          push    di
EFDB  86 C4             C          xchg    al,ah                               ; reverse al & ah
C                       C                                                      ; ah saves al throughout
EFDD  1E                C          push    ds                                  ; save ds
EFDE  2E: 8E 1E E5F2 R  C          mov     ds,word ptr cs:[set_ds_word]        ; satisfy assumptions
C
EFE3  8B FA             C          mov     di,dx                               ; get port number (0-3)
EFE5  33 C9             C          xor     cx,cx                               ; clear ch
EFE7  8A 8D 0078 R      C          mov     cl,byte ptr ds:[di+printer_t_out]   ; get printer time-out
EFEB  D1 E1             C          sal     cx,1                                ; double it 4 faster cpu
EFED  03 FF             C          add     di,di                               ; make word index
EFEF  8B 95 0008 R      C          mov     dx,word ptr ds:[di+printer_addr]    ; get address of printer
C                       C                                                      ; data port
EFF3  1F                C          pop     ds                                  ; restore ds
C
C                       C          assume  cs:code, ds:nothing, es:nothing, ss:nothing
C
EFF4  0B D2             C          or      dx,dx                               ; is a printer there?
EFF6  74 0D             C          jz      p_ret
C
EFF8  33 FF             C          xor     di,di                               ; clear di
C
EFFA  0A C0             C          or      al,al                               ; al = 0?
EFFC  74 0D             C          jz      p_out                               ; dx = prt_data_x
C
EFFE  42                C          inc     dx                                  ; dx = prt_stat_x
C
EFFF  2C 02             C          sub     al,2                                ; al = 1 < 2?
F001  72 24             C          jb      p_init                              ; dx = prt_stat_x
C                                                                              ; al = 2?
F003  74 2F             C          je      p_stat                              ; dx = prt_stat_x
C
F005  86 C4             C  p_ret:  xchg    al,ah                               ; reverse al & ah back
C                       C                                                      ; ah saved al throughout
F007  5F                C          pop     di                                  ; restore registers
F008  5A                C          pop     dx
F009  59                C          pop     cx
F00A  CF                C  p_nop:  iret
C
C  ;-----------------------------------------------------------------------
C  ;       Print Character to Parallel Printer Interface.
C  ;
C  ;       Input:  ah =    character to print
C  ;               cx =    printer time-out
C  ;               dx =    address of printer data port (prt_data_x).
C  ;               di =    0
C  ;       Output: ah =    character to print
C  ;               al =    status of printer: bit #0 set if time out
C  ;               dx =    address of printer status port (prt_stat_x).
C  ;       Trash:  cx & di destroyed.
C  ;
C  ;       Assumes:        prt_stat_x = prt_data_x + 1 = dx + 1
C  ;                       prt_cmd_x  = prt_data_x + 2 = dx + 2
C  ;-----------------------------------------------------------------------
C
C                       C          assume  cs:code, ds:nothing, es:nothing, ss:nothing
C
F00B  8A C4             C  p_out:  mov     al,ah                               ; get character to print
F00D  EE                C          out     dx,al                               ; output character
```

```
                                C
F00E  42                        C          inc     dx                           ; dx = prt_stat_x
                                C
F00F  EC                        C  p_lp:   in      al,dx                        ; get printer status
F010  24 F8                     C          and     al,0F8h                      ; clear bogus printer bits
F012  34 49                     C          xor     al,049h                      ; flip acknowledge & I/O err bit
                                C                                               ; & set printer time-out bit #0
F014  78 07                     C          js      p_ok                         ; wait for not busy bit #7 set
                                C
F016  4F                        C          dec     di                           ; inner loop counter
F017  75 F6                     C          jnz     p_lp                         ; inner loop
F019  E2 F4                     C          loop    p_lp                         ; outer loop
F01B  EB E8                     C          jmp     short p_ret                  ; return status with time-out
                                C
F01D  42                        C  p_ok:   inc     dx                           ; dx = prt_cmd_x
F01E  B0 0D                     C          mov     al,0Dh                       ; set strobe high (al = 0Dh)
F020  EE                        C          out     dx,al
F021  90                        C          nop
F022  48                        C          dec     ax                           ; set strobe low (al = 0Ch)
F023  EE                        C          out     dx,al
F024  4A                        C          dec     dx                           ; dx = prt_stat_x
                                C
F025  EB 0D                     C          jmp     shcrt p_stat
                                C
                                C  ;---------------------------------------------------------------------
                                C  ;       Initialize Parallel Printer Interface.
                                C  ;
                                C  ;       Input:  ah =    byte to return in al.
                                C  ;               dx =    address of printer status port (prt_stat_x).
                                C  ;       Output: al =    status of printer
                                C  ;               dx =    address of printer status port (prt_stat_x).
                                C  ;       Trash:  cx =    0 destroyed.
                                C  ;
                                C  ;       Assumes:        prt_stat_x = prt_data_x + 1 = dx + 1
                                C  ;                       prt_cmd_x  = prt_data_x + 2 = dx + 2
                                C  ;---------------------------------------------------------------------
                                C
                                C          assume  cs:code, ds:nothing, es:nothing, ss:nothing
                                C
F027  B0 08                     C  p_init: mov     al,08h                       ; request init (hold line low)
F029  42                        C          inc     dx                           ; dx = prt_cmd_x
F02A  EE                        C          out     dx,al
                                C
F02B  B5 05                     C          mov     ch,05h                       ; delay awhile (cx = 05??h)
F02D  E2 FE                     C          loop    $
                                C
F02F  B0 0C                     C          mov     al,0Ch                       ; disable interupts, manual lf
F031  EE                        C          out     dx,al                        ; (init done - line set high)
F032  90                        C          nop
                                C
F033  4A                        C          dec     dx                           ; dx = prt_stat_x
                                C  ;       jmp     short p_stat                 ; fall through
                                C
                                C  ;---------------------------------------------------------------------
                                C  ;       Read Status of Parallel Printer Interface.
                                C  ;
                                C  ;       Input:  dx =    address of printer status port (prt_stat_x).
                                C  ;
                                C  ;       Output: al =    status of printer
                                C  ;               dx =    address of printer status port (prt_stat_x).
                                C  ;
                                C  ;       Assumes:        prt_stat_x = prt_data_x + 1 = dx + 1
                                C  ;---------------------------------------------------------------------
                                C
                                C          assume  cs:code, ds:nothing, es:nothing, ss:nothing
                                C
F034  EC                        C  p_stat: in      al,dx                        ; get printer status
F035  24 F8                     C          and     al,0F8h                      ; clear bogus printer bits
F037  34 48                     C          xor     al,048h                      ; flip acknowledge & I/O err bit
F039  EB CA                     C          jmp     short p_ret                  ; exit
                                C
F03B                            C  p_io    endp
                                C
F03B                            C  code    ends
                                C  include vid.asm
                                C
                                C  ;===========================================================================
                                C  ;       Filename:       vid.src
                                C  ;
                                C  ;       This module includes INT 10h, the display routines.
                                C  ;
                                C  ;===========================================================================
                                C
                                C  ; These constants must be defined (amount to scroll/clear during vert retrace):
= 0004                          C          V_KSCROLL       equ     4       ; 4 rows, plus
= 0007                          C          V_KCLEAR        equ     7
                                C
F03B                            C  code    segment public 'ROM'
                                C          assume  cs:code, ds:nothing, es:nothing, ss:nothing
                                C
                                C  ;---------------------------------------------------------------------
                                C  ;       ROM data
                                C  ;---------------------------------------------------------------------
                                C
F045                            C          ORG     0F045h
                                C
F045                            C  v_data1 proc    near
                                C
F045  F0FC R                    C  v_tbl   dw      v_set_mode              ; ah = 00h
F047  F1E9 R                    C          dw      v_curs_type             ; ah = 01h
```

```
F049  F1F7 R                C           dw      v_curs_pos              ; ah = 02h
F04B  F215 R                C           dw      v_r_curs_pos            ; ah = 03h
F04D  D7CD R                C           dw      grf_light_pen           ; ah = 04h      (see graph.src)
F04F  F22C R                C           dw      v_page                  ; ah = 05h
F051  F27F R                C           dw      v_scrl_up               ; ah = 06h
F053  F35B R                C           dw      v_scrl_dn               ; ah = 07h
F055  F37C R                C           dw      v_rac                   ; ah = 08h
F057  F3A6 R                C           dw      v_wac                   ; ah = 09h
F059  F3DF R                C           dw      v_wc                    ; ah = 0Ah
F05B  F41A R                C           dw      v_col                   ; ah = 0Bh
F05D  D7EA R                C           dw      grf_write_dot           ; ah = 0Ch      (see graph.src)
F05F  D7D0 R                C           dw      grf_read_dot            ; ah = 0Dh      (see graph.src)
F061  F446 R                C           dw      v_terminal              ; ah = 0Eh
F063  F4CF R                C           dw      v_stat                  ; ah = 0Fh
                            C
F065                        C   v_data1 endp
                            C
                            C   ;=========================================================================
                            C   ;       INT 10h --  Video Interrupt Service Routine.
                            C   ;=========================================================================
                            C   ;       -- Set CPU flags.
                            C   ;       -- Segment registers properly loaded.
                            C   ;       -- CALLs routines with: -- al, bx, cx, dx intact
                            C   ;                               -- ah = v_mode
                            C   ;                               -- si = 2 * (function that was in ah)
                            C   ;                               -- di = bits #4 & 5 of switch_bits
                            C   ;                               -- bp = value of ax to be returned
                            C   ;
                            C   ;       Input:  ah      = function number (00h <= ah <= 0Fh)
                            C   ;
                            C   ;       Trash:  None. (bp, si, di, ds, & es if ROM stack)
                            C   ;=========================================================================
                            C
F065                        C           ORG     0F065h
                            C
F065                        C   v_io    proc    near
                            C
                            C           assume  cs:code, ds:nothing, es:nothing, ss:nothing
                            C
F065  FB                    C           sti                                     ; enable interrupts
F066  80 FC 0F              C           cmp     ah,0Fh                          ; input out of range?
F069  77 38                 C           ja      v_nop
                            C
F06B  06                    C           push    es                              ; save 'trashable' registers
F06C  1E                    C           push    ds
F06D  2E: 8E 1E E5F2 R      C           mov     ds,word ptr cs:[set_ds_word]    ; avoid potential stack problems
F072  57                    C           push    di
F073  56                    C           push    si
F074  55                    C           push    bp
                            C
                            C           assume  cs:code, ds:data, es:nothing, ss:nothing
                            C
F075  50                    C           push    ax                              ; save ax
                            C
F076  8A C4                 C           mov     al,ah                           ; al = function number
F078  02 C4                 C           add     al,ah                           ; * 2
F07A  98                    C           cbw
F07B  50                    C           push    ax                              ; save for CALL later.
                            C
F07C  BF B800               C           mov     di,para_graph                   ; get screen segment ..
F07F  A0 0010 R             C           mov     al,byte ptr ds:[switch_bits]    ; .. check switch bits ..
F082  A8 20                 C           test    al,20h
F084  74 08                 C           jz      v_colour                        ; .. if either is 0, it's
F086  A8 10                 C           test    al,10h                          ; .. color display board
F088  74 04                 C           jz      v_colour
F08A  81 C7 F800            C           add     di,para_mono-para_graph         ; .. it's a monochrome board
F08E                        C   v_colour:
F08E  8E C7                 C           mov     es,di                           ; set es = video ram
                            C
F090  5F                    C           pop     di                              ; Get index to call table.
                            C
F091  58                    C           pop     ax                              ; restore ax
F092  8A 26 0049 R          C           mov     ah,byte ptr ds:[v_mode] ; get display driver mode
F096  8B E8                 C           mov     bp,ax                           ; bp saves ax throughout
                            C                                                   ; (return ah = v_mode
                            C                                                   ;  unless specific ret. value)
                            C
F098  FC                    C           cld                                     ; String ops move UP, mostly.
F099  2E: FF 95 F045 R      C           call    cs:[di+(offset v_tbl)]          ; perform display function
                            C
F09E  5D                    C           pop     bp                              ; restore 'trashed' registers
F09F  5E                    C           pop     si                              ; (destroyed if ROM stack)
F0A0  5F                    C           pop     di
F0A1  1F                    C           pop     ds
F0A2  07                    C           pop     es
F0A3                        C   v_nop:
F0A3  CF                    C           iret
                            C
F0A4                        C   v_io    endp
                            C
                            C   ;=========================================================================
                            C
                            C
F0A4                        C           ORG     0F0A4h
                            C
F0A4                        C   v_data2 proc    near
                            C
F0A4                        C   v_parms label   byte
                            C
                            C   ; 6845 Parameters except register 3h (Horizontal Synch Width).
                            C
```

```
F0A4  38 28 2D 06          C  v_md_40          db      38h,28h,2Dh,06h ; text  40 x 25
F0A8  1F 06 19 1C          C                   db      1Fh,06h,19h,1Ch ; mode 0 -> monochrome
F0AC  02 07 06 07          C                   db      02h,07h,06h,07h ; mode 1 -> color
F0B0  00 00 00 00          C                   db      00h,00h,00h,00h
                           C
F0B4  71 50 5A 0C          C  v_md_80          db      71h,50h,5Ah,0Ch ; text  80 x 25
F0B8  1F 06 19 1C          C                   db      1Fh,06h,19h,1Ch ; mode 2 -> monochrome
F0BC  02 07 06 07          C                   db      02h,07h,06h,07h ; mode 3 -> color
F0C0  00 00 00 00          C                   db      00h,00h,00h,00h
                           C
F0C4  38 28 2D 06          C  v_md_graph       db      38h,28h,2Dh,06h ; graphics
F0C8  7F 06 64 70          C                   db      7Fh,06h,64h,70h ; mode 4 -> 320 x 200 color
F0CC  02 01 06 07          C                   db      02h,01h,06h,07h ; mode 5 -> 320 x 200 monochrome
F0D0  00 00 00 00          C                   db      00h,00h,00h,00h ; mode 6 -> 640 x 200 monochrome
                           C
F0D4  61 50 52 0F          C  v_md_mono        db      61h,50h,52h,0Fh ; monochrome card 80 x 25
F0D8  19 06 19 19          C                   db      19h,06h,19h,19h ; mode 7 -> monochrome card
F0DC  02 0D 0B 0C          C                   db      02h,0Dh,0Bh,0Ch ;
F0E0  00 00 00 00          C                   db      00h,00h,00h,00h
                           C  .list
                           C
F0E4  0800                 C  v_md_len         dw      2048            ; 40x25         modes 0 & 1
F0E6  1000                 C                   dw      4096            ; 80x25         modes 2 & 3
F0E8  4000                 C                   dw      16384           ; graphics      modes 4 & 5
F0EA  4000                 C                   dw      16384           ;               mode  6
                           C  ;only in 1050    dw      32768           ;               modes 64 & 72
                           C
F0EC  28                   C  v_md_wid         db      40              ; 0 -> text      40x 25 monochrome
F0ED  28                   C                   db      40              ; 1 -> text      40x 25 color
F0EE  50                   C                   db      80              ; 2 -> text      80x 25 monochrome
F0EF  50                   C                   db      80              ; 3 -> text      80x 25 color
F0F0  28                   C                   db      40              ; 4 -> graphics 320x200 color
F0F1  28                   C                   db      40              ; 5 -> graphics 320x200 monochrome
F0F2  50                   C                   db      80              ; 6 -> graphics 640x200 monochrome
F0F3  50                   C                   db      80              ; 7 -> text      80x 25 monochrome card
                           C  .list
F0F4  2C                   C  v_md_enable      db      2Ch             ; 0 -> text      40x 25 monochrome
F0F5  28                   C                   db      28h             ; 1 -> text      40x 25 color
F0F6  2D                   C                   db      2Dh             ; 2 -> text      80x 25 monochrome
F0F7  29                   C                   db      29h             ; 3 -> text      80x 25 color
F0F8  2A                   C                   db      2Ah             ; 4 -> graphics 320x200 color
F0F9  2E                   C                   db      2Eh             ; 5 -> graphics 320x200 monochrome
F0FA  1E                   C                   db      1Eh             ; 6 -> graphics 640x200 monochrome
F0FB  29                   C                   db      29h             ; 7 -> text      80x 25 monochrome card
                           C  .list
F0FC                       C  v_data2 endp
                           C
                           C  ;===========================================================================
                           C
                           C  ;---------------------------------------------------------------------------
                           C  ;       Set Mode & Clear Screen         ah = 00h
                           C  ;
                           C  ;       Input:  al      = mode  = 0 -> text      40x 25 monochrome
                           C  ;                              = 1 -> text      40x 25 color
                           C  ;                              = 2 -> text      80x 25 monochrome
                           C  ;                              = 3 -> text      80x 25 color
                           C  ;                              = 4 -> graphics 320x200 color
                           C  ;                              = 5 -> graphics 320x200 monochrome
                           C  ;                              = 6 -> graphics 640x200 monochrome
                           C  ;                              = 7 -> text      80x 25 monochrome card
                           C  ;                              = 64 -> graphics 640x400 monochrome
                           C  ;                              = 72 -> graphics 640x400 monochrome tinytext
                           C  ;
                           C  ;       Output: ah      = 00h
                           C  ;               al      = v_colorpal
                           C  ;
                           C  ;       Assume:  contents of v_base6845     = pointer register          (3D4h)
                           C  ;               (contents of v_base6845)+1  = data register             (3D5h)
                           C  ;               (contents of v_base6845)+4  = mode control register     (3D8h)
                           C  ;               (contents of v_base6845)+5  = overscan register         (3D9h)
                           C  ;               (contents of v_base6845)+6  = status register           (3DAh)
                           C  ;               (contents of v_base6845)+10 = mode control register #2  (3DEh)
                           C  ;
                           C  ;       Trash:  si & di destroyed.
                           C  ;---------------------------------------------------------------------------
                           C
F0FC                       C  v_set_mode       proc    near
                           C
                           C                   assume  cs:code, ds:data, es:v_ram, ss:nothing
                           C
F0FC  52                   C          push     dx                              ; save dx
F0FD  51                   C          push     cx                              ; save cx
                           C
                           C  ; Get Color/Monochrome dependent stuff.
                           C
F0FE  32 E4                C          xor      ah,ah                           ; AH = mode ctrl., color board
F100  BA 03D4              C          mov      dx,color_pointer                ; color 6845 pointer register.
                           C
F103  F6 06 0010 R 10      C          test     byte ptr ds:[switch_bits],10h   ; monochrome board?
F108  74 0D                C          jz       v_set_mode_color                ; if not, skip monochrome stuff.
F10A  F6 06 0010 R 20      C          test     byte ptr ds:[switch_bits],20h
F10F  74 06                C          jz       v_set_mode_color
                           C                                                   ; It's a monochrome board, so..
F111  B8 0107              C          mov      ax,0107h                        ; (AH,AL) = (overwrite mode
                           C                                                   ; for monochrome, mono. mode)
F114  83 C2 E0             C          add      dx,v_pointer-color_pointer; monochrome pointer register.
                           C
F117                       C  v_set_mode_color:
                           C
                           C  ; Save CRT Mode & 6845 address, and reset display monitor with mode control.
```

```
                        C
F117  A2 0049 R         C          mov     byte ptr ds:[v_mode],al ; save mode.
F11A  89 16 0063 R      C          mov     word ptr ds:[v_base6845],dx     ; save 6845 pointer register.
F11E  86 E0             C          xchg    ah,al                           ; AH=mode, AL=mode ctrl.
F120  83 C2 04          C          add     dx,4                            ; get 6845 mode control register
F123  EE                C          out     dx,al                           ; reset display
F124  83 EA 04          C          sub     dx,4                            ; restore 6845 address.
F127  8A C4             C          mov     al,ah                           ; restore mode in al
                        C
                        C  ; Get pointer to display parameters.
                        C
F129  1E                C          push    ds                              ; save ds = data_seg
                        C
                        C                  assume  cs:code, ds:abs0, es:v_ram, ss:nothing
                        C
F12A  33 F6             C          xor     si,si
F12C  8E DE             C          mov     ds,si                           ; satisfy assumptions
F12E  C5 36 0074 R      C          lds     si,dword ptr ds:[int1Dlocn]     ; display parameter pointer
                        C
                        C  ; ds:si in effect points to cs:v_parms.
                        C
                        C                  assume  cs:code, ds:code, es:v_ram, ss:nothing
                        C
                        C  ; Determine which set of parameters to use from mode.
                        C
F132  B9 0010           C          mov     cx,16                           ; count of parameters
F135  3C 02             C          cmp     al,2                            ; 40x25 mode?  (0 & 1?)
F137  72 0E             C          jb      v_set_mode_lp                   ; if so, we're done
F139  03 F1             C          add     si,cx                           ; next set of parameters
F13B  3C 04             C          cmp     al,4                            ; 80x25 mode?  (2 & 3?)
F13D  72 08             C          jb      v_set_mode_lp                   ; if so, we're done
F13F  03 F1             C          add     si,cx                           ; next set of parameters
F141  3C 07             C          cmp     al,7                            ; graphics mode?  (4,5,6,64,72?)
F143  75 02             C          jne     v_set_mode_lp                   ; if so, we're done
F145  03 F1             C          add     si,cx                           ; else, monochrome card (7)
                        C
                        C  ; Loop through 6845 initialization table outputting register number and data
                        C
F147                    C  v_set_mode_lp:
                        C
F147  B0 10             C          mov     al,16                           ; get 6845 register number =
                        C                                                  ;  = (16 - cl) (unscrambled)
F149  2A C1             C          sub     al,cl
F14B  EE                C          out     dx,al                           ; output to register port
F14C  42                C          inc     dx                              ; point to 6845 data register
F14D  AC                C          lodsb                                   ; get parm value:  al gets ds:si
F14E  EE                C          out     dx,al                           ; output to data port
F14F  4A                C          dec     dx                              ; point back to pointer register
F150  E2 F5             C          loop    v_set_mode_lp                   ; next register
                        C                                                  ; dx = pointer register
                        C
                        C                  assume  cs:code, ds:data, es:v_ram, ss:nothing
                        C
F152  1F                C          pop     ds                              ; restore ds = data_seg
                        C
F153  8A C4             C          mov     al,ah                           ; save mode in ah & al
F155  32 E4             C          xor     ah,ah                           ; ax = mode
F157  8B F0             C          mov     si,ax                           ; si = mode hence...
                        C
                        C  ; Clear the screen.
                        C
F159  B9 2000           C          mov     cx,2000h                        ; assume 8k words to clear
                        C
F15C  3C 04             C          cmp     al,4                            ; 40x25 or 80x25 text modes 0-3?
F15E  72 0E             C          jb      v_md_clr_8k                     ; if so, clear 8k words.
F160  3C 07             C          cmp     al,7                            ; monochrome card mode 7?
F162  74 08             C          je      v_md_clr_2k                     ; if so, clear 2k words.
F164  72 02             C          jb      v_md_clr_graphics               ; graphics mode 4-6 clear 8k wds
F166  D1 E1             C          shl     cx,1                            ;   mode 64 & 72 clear 16k wds
F168                    C  v_md_clr_graphics:
F168  33 C0             C          xor     ax,ax                           ; graphics mode
F16A  EB 05             C          jmp     short v_md_clr                  ; clear screen with zeroes
                        C
F16C                    C  v_md_clr_2k:
F16C  B5 08             C          mov     ch,08h                          ; cx = 0800h
F16E                    C  v_md_clr_8k:
F16E  B8 0720           C          mov     ax,(7*100h)+(' ')               ; clear with attribute & space
F171                    C  v_md_clr:
F171  33 FF             C          xor     di,di
F173  F3/ AB            C          rep     stosw                           ; es:di gets ax
                        C
                        C  ; Set mode control register #2
                        C
                        C  ; Handle underline on shades-of-gray monitor.
                        C
                        C                                                  ; dx = pointer register
F175  83 C2 06          C          add     dx,6                            ; get 6845 status register
F178  EC                C          in      al,dx                           ; get CRT status
F179  24 10             C          and     al,010h                         ; isolate color/shades bit #4
F17B  D0 E0             C          shl     al,1                            ; move it to underline bit #6
F17D  D0 E0             C          shl     al,1
                        C
                        C  ; Handle double scan line modes 64 & 72.
                        C                                                  ; dx = 6845 status register
F17F  83 FE 40          C          cmp     si,64                           ; modes 0 through 7?
F182  72 04             C          jb      v_md_dbl                        ; if so, single scan line mode
F184  40                C          inc     ax                              ; else mode 64 or 72, set bit #0
F185  BE 0006           C          mov     si,6                            ; modes 64 & 72 look like mode 6
                        C                                                  ;    from now on
                        C
F188                    C  v_md_dbl:                                       ; double scan line mode.
```

```
F188  83 C2 04              C           add     dx,4                            ; get mode control register #2
F18B  EE                    C           out     dx,al
                            C
                            C   ; Enable display monitor with mode control.
                            C
F18C  2E: 8A 84 F0F4 R      C           mov     al,byte ptr cs:[si+v_md_enable]
F191  A2 0065 R             C           mov     byte ptr ds:[v_3x8],al          ; save the value for later
                            C                                                   ; dx = mode control register #2
F194  83 EA 06              C           sub     dx,6                            ; get 6845 mode control register
F197  EE                    C           out     dx,al                           ; enable display
                            C
                            C   ; Determine width & length of screen.
                            C
F198  33 C0                 C           xor     ax,ax                           ; clear ah
F19A  2E: 8A 84 F0EC R      C           mov     al,byte ptr cs:[si+v_md_wid]
F19F  A3 004A R             C           mov     word ptr ds:[v_width],ax
                            C
F1A2  81 E6 000E            C           and     si,0Eh                          ; make word index divided by 2
F1A6  2E: 8B 84 F0E4 R      C           mov     ax,word ptr cs:[si+v_md_len]
F1AB  A3 004C R             C           mov     word ptr ds:[v_height],ax
                            C
                            C   ; Set up overscan register & v_colorpal.
                            C
F1AE  B0 30                 C           mov     al,030h                         ; v_colorpal for modes 0-5 & 7
F1B0  8A 26 0049 R          C           mov     ah,byte ptr ds:[v_mode] ; retrieve v_mode
F1B4  80 FC 06              C           cmp     ah,6                            ; modes 0-5 ?
F1B7  72 0D                 C           jb      v_ovr_ok                        ; if so, we're ok.
                            C
F1B9  74 09                 C           je      v_ovr_not_ok                    ; 640x200 graphics mode 6 ?
                            C                                                   ; if so, change v_colorpal
                            C
F1BB  80 FC 40              C           cmp     ah,64                           ; 640x400 graphics mode 64, 72 ?
F1BE  72 06                 C           jb      v_ovr_ok                        ; if not, we're ok.
                            C
                            C                                                   ; if so we set v_height wrong,
F1C0  D1 26 004C R          C           shl     word ptr ds:[v_height],1        ; double v_height from 16k to 32k
                            C
F1C4                        C   v_ovr_not_ok:
F1C4  B0 3F                 C           mov     al,03Fh                         ; v_colorpal for modes 6,64,72
                            C
F1C6                        C   v_ovr_ok:                                       ; dx = 6845 mode control register
F1C6  A2 0066 R             C           mov     byte ptr ds:[v_colorpal],al     ; save the value for later
F1C9  42                    C           inc     dx                              ; get 6845 overscan register
F1CA  EE                    C           out     dx,al
                            C
                            C   ; Clear all cursor positions.
                            C
                            C                   assume  cs:code, ds:data, es:data, ss:nothing
                            C
F1CB  1E                    C           push    ds                              ; set es = firmware data area
F1CC  07                    C           pop     es
                            C
F1CD  BF 0050 R             C           mov     di,ds:(offset v_curpos) ; get address [defined by DB]
F1D0  B9 0008               C           mov     cx,8
F1D3  33 C0                 C           xor     ax,ax                           ; ax = 0
F1D5  F3/ AB                C           rep     stosw                           ; es:di gets ax = 0
                            C
                            C   ; Clear other firmware data variables (ax = 0).
                            C
F1D7  A3 004E R             C           mov     word ptr ds:[v_top],ax  ; set starting offset to 0
F1DA  A2 0062 R             C           mov     byte ptr ds:[v_apage],al        ; set current active page to 0
F1DD  C7 06 0060 R 0607     C           mov     word ptr ds:[v_cursize],0607h   ; set cursor mode
                            C
                            C   ; Clean up.
                            C
F1E3  A0 0066 R             C           mov     al,byte ptr ds:[v_colorpal]
F1E6  59                    C           pop     cx
F1E7  5A                    C           pop     dx                              ; restore cx
F1E8  C3                    C           ret                                     ; restore dx
                            C
F1E9                        C   v_set_mode      endp
                            C
                            C   ;-----------------------------------------------------------------------
                            C   ;       Set Cursor Value                ah = 01h
                            C   ;
                            C   ;       Input:  ch      = bits #0-4 = starting line for cursor
                            C   ;               cl      = bits #0-4 = ending line for cursor
                            C   ;       Output: ah      = 10
                            C   ;               al      = cl
                            C   ;
                            C   ;       Trash:  si & di destroyed. (si = dx; di = cx)
                            C   ;-----------------------------------------------------------------------
                            C
F1E9                        C   v_curs_type     proc    near
                            C           assume  cs:code, ds:data, es:v_ram, ss:nothing
                            C
F1E9  89 0E 0060 R          C           mov     word ptr ds:[v_cursize],cx      ; save the cursor value
                            C
F1ED  8B F9                 C           mov     di,cx                           ; di saves cx
                            C
F1EF  B4 0A                 C           mov     ah,10                           ; 6845 cursor set register = 10
F1F1  E8 F262 R             C           call    v_6845                          ; set cursor; ah 6845 gets cx
                            C                                                   ;  si = dx destroyed
                            C                                                   ;  ah = preserved = 10; al = cl
                            C                                                   ;  di restores cx
F1F4  8A C1                 C           mov     al,cl                           ; restore al with original cl
F1F6  C3                    C           ret
F1F7                        C   v_curs_type     endp
                            C
                            C   ;-----------------------------------------------------------------------
```

```
                        ;       Set Cursor Position             ah = 02h
                        ;
                        ;       Input:  bh      = page number (0-7)
                        ;               (dh,dl) = (row,col) of current cursor from (0,0)
                        ;       Output: if bh = v_apage,        ah = 14
                        ;                                       al = low byte of cursor position
                        ;               else,                   ah = v_mode
                        ;                                       al = preserved
                        ;
                        ;       Trash:  si & di destroyed. (si = dx, di = cx)
                        ;-----------------------------------------------------------------------

F1F7                    v_curs_pos      proc    near

                                assume  cs:code, ds:data, es:v_ram, ss:nothing

F1F7  8B F9                     mov     di,cx                           ; save cx

F1F9  8A CF                     mov     cl,bh
F1FB  8B F1                     mov     si,cx
F1FD  81 E6 0007                and     si,7                            ; mask to 8 pages
F201  03 F6                     add     si,si                           ; *2 => word index

F203  89 94 0050 R              mov     word ptr ds:[si+v_curpos],dx    ; save the cursor position

F207  3A 3E 0062 R              cmp     bh,byte ptr ds:[v_apage]        ; if active page, put the cursor
F20B  74 03                     je      v_set_curs                      ; on the screen.
                                                                        ; not active page, so just
F20D  8B CF                     mov     cx,di                           ;  restore cx
F20F  C3                        ret                                     ;  and exit

F210                    v_set_curs:                                     ; active page, so set cursor..
F210  8B C2                     mov     ax,dx                           ; ax gets cursor position
F212  EB 42 90                  jmp     v_set_cur_pos                   ; set cursor; ah 6845 gets cx
                                                                        ;  si = dx destroyed
                                                                        ;  ah = preserved = 14
                                                                        ;  al = low byte of cursor posn

F215                    v_curs_pos      endp

                        ;-----------------------------------------------------------------------
                        ;       Read Cursor                     ah = 03h
                        ;
                        ;       Input:  bh      = page number (0-7)
                        ;       Output: (dh,dl) = (row,col) of current cursor from (0,0)
                        ;               (ch,cl) = current cursor mode setting
                        ;               ax      = dx
                        ;
                        ;       Trash:  None.
                        ;-----------------------------------------------------------------------

F215                    v_r_curs_pos    proc    near

                                assume  cs:code, ds:data, es:v_ram, ss:nothing

F215  8B C2                     mov     ax,dx
F217  8B CB                     mov     cx,bx                           ; save bx
F219  8A DF                     mov     bl,bh
F21B  81 E3 0007                and     bx,07h                          ; page number mod 8
F21F  D1 E3                     shl     bx,1                            ; page number mod 8 word index
F221  8B 97 0050 R              mov     dx,word ptr ds:[bx+v_curpos]
F225  8B D9                     mov     bx,cx                           ; restore bx
F227  8B 0E 0060 R              mov     cx,word ptr ds:[v_cursize]
F22B  C3                        ret

F22C                    v_r_curs_pos    endp

                        ;-----------------------------------------------------------------------
                        ;       Read Light Pen (see graph.src)  ah = 04h
                        ;-----------------------------------------------------------------------

                        ;-----------------------------------------------------------------------
                        ;       Set Active Display Page         ah = 05h
                        ;
                        ;       Input:  al      = new page number (0-7 for modes 0-1; 0-3 for 2-3)
                        ;       Output: 6845 is reset to display the new active page
                        ;               ah      = 14
                        ;               al      = low byte of cursor position
                        ;
                        ;       Trash:  bp, si & di destroyed. (si = dx, di = cx)
                        ;-----------------------------------------------------------------------

F22C                    v_page  proc    near

                                assume  cs:code, ds:data, es:v_ram, ss:nothing

F22C  25 0007                   and     ax,07h                          ; page number mod 8
F22F  8B E8                     mov     bp,ax                           ; save ax = page number mod 8

F231  A2 0062 R                 mov     byte ptr ds:[v_apage],al        ; save active page number (0-7)
F234  74 08                     jz      v_page_0                        ; page number = 0?

F236  8B FA                     mov     di,dx                           ; save dx
F238  F7 26 004C R              mul     word ptr ds:[v_height]          ; dx:ax = (page number)*v_height
F23C  8B D7                     mov     dx,di                           ; restore dx

F23E                    v_page_0:
F23E  A3 004E R                 mov     word ptr ds:[v_top],ax          ; save starting address of page
F241  D1 F8                     sar     ax,1                            ; divide by 2 for byte count
```

```
                                C
F243  8B F9                     C          mov     di,cx                        ; save cx
                                C
F245  8B C8                     C          mov     cx,ax                        ; cx gets offset of active page
F247  B4 0C                     C          mov     ah,12                        ; 6845 cursor set address = 12
F249  E8 F262 R                 C          call    v_6845                       ; set cursor; ah 6845 gets cx
                                C                                               ;  si = dx destroyed
                                C                                               ;  ah = preserved = 12; al = cl
                                C                                               ;  di restores cx
                                C
F24C  8B C5                     C          mov     ax,bp                        ; restore ax = page number mod 8
F24E  D1 E0                     C          shl     ax,1                         ; page number mod 8 word index
F250  8B F0                     C          mov     si,ax
F252  8B 84 0050 R              C          mov     ax,word ptr ds:[si+v_curpos] ; get page's cursor position
                                C
F256                            C  v_set_cur_pos:                               ;(ah,al) = (row,col) cursor pos.
                                C                                               ; di     = value to return in cx
F256  E8 F52B R                 C          call    v_posn                       ; (ah,al) -> ax offset; si trash
F259  03 06 004E R              C          add     ax,word ptr ds:[v_top]       ; add offset of active page
F25D  D1 F8                     C          sar     ax,1                         ; divide by 2 for byte count
                                C
F25F  B5 0E                     C          mov     ch,14                        ; 6845 cursor pos register = 14
F261  91                        C          xchg    cx,ax
                                C
                                C
                                C  ; Now:  ah      = 6845 register selection
                                C  ;       ch      = first data byte to (ah) 6845 internal register
                                C  ;       cl      = second data byte to (ah+1) 6845 internal register
                                C  ;       di      = value to return in cx
                                C
                                C
F262                            C  v_6845:         ; program 6845 cursor:
                                C
F262  8B F2                     C          mov     si,dx                        ; save dx
F264  E8 F273 R                 C          call    v_out_byte                   ; output to 6845
F267  FE C4                     C          inc     ah                           ; next 6845 register
F269  8A E9                     C          mov     ch,cl                        ; get the register input
F26B  E8 F273 R                 C          call    v_out_byte                   ; output to 6845
F26E  8B D6                     C          mov     dx,si                        ; restore dx value
F270  8B CF                     C          mov     cx,di                        ; set up return value
F272  C3                        C          ret
F273                            C  v_page  endp
                                C
                                C  ;-----------------------------------------------------------------------------
                                C  ;       Output two byte to the selected 6845 registers
                                C  ;
                                C  ;       Input:  ah      = 6845 register selection
                                C  ;               ch      = first data byte to (ah) 6845 internal register
                                C  ;               cl      = second data byte to (ah+1) 6845 internal register
                                C  ;               di      = value to return in cx
                                C  ;
                                C  ;       Assume: contents of v_base6845     = pointer register
                                C  ;              (contents of v_base6845)+1        = data register
                                C  ;
                                C  ;       Output: ah      = 6845 register selection
                                C  ;               al      = second data byte
                                C  ;               cx      = di
                                C  ;-----------------------------------------------------------------------------
                                C
F273                            C  v_out_byte      proc    near
                                C
                                C          assume  cs:code, ds:data, es:v_ram, ss:nothing
                                C
F273  8B 16 0063 R              C          mov     dx,word ptr ds:[v_base6845]  ; get 6845 pointer register
F277  8A C4                     C          mov     al,ah                        ; get the register address
F279  EE                        C          out     dx,al                        ; select the data register
                                C
F27A  42                        C          inc     dx                           ; next register
F27B  8A C5                     C          mov     al,ch                        ; get second data byte
F27D  EE                        C          out     dx,al                        ; output a data byte to 6845
F27E  C3                        C          ret
                                C
F27F                            C  v_out_byte      endp
                                C
                                C  ;-----------------------------------------------------------------
                                C  ;       Scroll Active Page Up           ah = 06h
                                C  ;
                                C  ;       Input:  if al = 0, then clear entire window with attribute in bh
                                C  ;               else,   al = number of rows to `scroll' up
                                C  ;                       = number of rows to clear at bottom of window
                                C  ;               bh      = attribute to be used on blank row(s)
                                C  ;               (ch,cl) = (row,col) of upper left corner of window from (0,0)
                                C  ;               (dh,dl) = (row,col) of lower right corner of window from (0,0)
                                C  ;       Output: ah      = attribute to be used on blank row(s)
                                C  ;               if v_mode = 7,  al = 20h = space
                                C  ;               else                    al = v_3x8
                                C  ;
                                C  ;       Assume: (contents of v_base6845)+6  = status register
                                C  ;
                                C  ;       Trash:  bp, si, & di destroyed. (bx thru dx destroyed if ROM stack)
                                C  ;-----------------------------------------------------------------
                                C
F27F                            C  v_scrl_up       proc    near
                                C
                                C          assume  cs:code, ds:data, es:v_ram, ss:nothing
                                C
F27F  E8 F53C R                 C          call    v_txt_md                ; all registers preserved
F282  72 03                     C          jb      v_txt_up
F284  E9 D874 R                 C          jmp     grf_graphics_up         ; jump if graphics
F287                            C  v_txt_up:
```

```
                        C
F287  52                C          push    dx                      ; save registers
F288  51                C          push    cx
F289  53                C          push    bx
                        C
F28A  8A D8             C          mov     bl,al                   ; save line count
F28C  8B C1             C          mov     ax,cx                   ; pass upper left coordinates....
F28E  E8 FADE R         C          call    v_scrl_pos              ; to common scroll positioning routine
F291  74 62             C          jz      v_clr_bot               ; clear rows if nothing to move
                        C
                        C ; Scroll cl rows up.
                        C
                        C ;v_mv_up:
F293  03 F0             C          add     si,ax                   ; add (bytes/row)*(rows to scroll) to
                        C                                          ; 'from' address for scroll
                        C
                        C ; Scroll cl rows up/down (based on bp & direction flag DF).
                        C
F295                    C v_mv:
F295  8A E1             C          mov     ah,cl                   ; ah gets number of rows to move = cl
                        C
F297  80 3E 0049 R 07   C          cmp     byte ptr ds:[v_mode],7
F29C  74 49             C          je      v_mv_flp                ; jump to fast loop
                        C
F29E  8A CA             C          mov     cl,dl                   ; no. of columns to move up/down per row
                        C
F2A0                    C v_mv_lp:
F2A0  52                C          push    dx                              ; save dx
F2A1  8B 16 0063 R      C          mov     dx,word ptr ds:[v_base6845]     ; get 6845 pointer register
F2A5  83 C2 06          C          add     dx,6                            ; get 6845 status register
                        C
                        C                  assume  cs:code, ds:v_ram, es:v_ram, ss:nothing
                        C
F2A8  06                C          push    es                      ; satisfy assumptions
F2A9  1F                C          pop     ds                      ; ds gets es
                        C
                        C ; Wait for horizontal retrace...
                        C
F2AA                    C v_mv_hi:
F2AA  EC                C          in      al,dx                   ; get CRT status
F2AB  D0 D8             C          rcr     al,1                    ; test display enable (bit #0)
F2AD  72 FB             C          jc      v_mv_hi                 ; wait for display enable low (cf)
                        C                                          ; interupts are disabled ..
                        C
F2AF  FA                C          cli                             ; disable now
                        C
F2B0                    C v_mv_lo:
F2B0  EC                C          in      al,dx                   ; 08    get CRT status
F2B1  D0 D8             C          rcr     al,1                    ; 02    test display enable (bit #0)
F2B3  73 FB             C          jnc     v_mv_lo                 ; 16/04 wait for display enable hi (cf)
                        C .list
F2B5  A5                C          movsw                           ; 18    move word (es:di gets ds:si)
                        C
                        C ; Worst case: (8+2+16)+(8+2+4)+(18) = 58 cycles = 72.5% of 80 cycles
                        C
F2B6  EC                C          in      al,dx                   ; 08    get CRT status (vert synch slow)
                        C
                        C                  assume  cs:code, ds:data, es:v_ram, ss:nothing
                        C
F2B7  BA 0040           C          mov     dx,data_seg             ; 04    satisfy assumptions
F2BA  8E DA             C          mov     ds,dx                   ; 02
F2BC  5A                C          pop     dx                      ; 08    restore dx
                        C
F2BD  49                C          dec     cx                      ; 03    decrement columns to move
F2BE  75 0A             C          jnz     v_mv_nxt                ; 04/16 did we move the last column?
                        C
F2C0  03 F5             C          add     si,bp                   ; 03    add/subtract number of bytes
                        C                                          ;       to skip in row to/from 'from'
                        C                                          ;       address for scroll up/down
F2C2  03 FD             C          add     di,bp                   ; 03    add/subtract number of bytes
                        C                                          ;       to skip in row to/from 'to'
                        C                                          ;       address for scroll up/down
                        C
F2C4  FE CC             C          dec     ah                      ; 03    decrement rows to move
F2C6  74 2C             C          jz      v_mv_end                ; 4/16 did we move the last row?
                        C
F2C8  8A CA             C          mov     cl,dl                   ; 02    number of columns to move
                        C                                          ;       up/down per row
                        C
                        C ; Worst case: [58]+(8+4+2+8)+(3+4)+(3+3)+(3+4+2) = 102 cycles = 12.75 usec
                        C
F2CA                    C v_mv_nxt:
                        C
F2CA  FB                C          sti                             ; enable interrupts immediately
F2CB  A8 08             C          test    al,08h                  ; 04    check for vertical retrace
F2CD  74 16             C          jz      v_mv_past               ; 16/04 jump past vertical retrace code
                        C
                        C ; We get here if vertical retrace has started...
                        C
                        C                                          ; NOTE: interrupts disabled for 1.1ms!
F2CF  B0 04             C          mov     al,V_KSCROLL            ; 04    at most V_KSCROLL rows per
                        C                                          ;       vertical retrace ...
F2D1                    C v_mv_row:
F2D1  1E                C          push    ds
F2D2  06                C          push    es
F2D3  1F                C          pop     ds
F2D4  F3/ A5            C          rep     movsw                   ; 11+17*80 move row (es:di gets es:si)
F2D6  1F                C          pop     ds
                        C
F2D7  03 F5             C          add     si,bp                   ; 03    add/subtract number of bytes
```

```
                                C                                       ;       to skip in row to/from `from'
                                C                                       ;       address for scroll up/down
F2D9  03 FD                     C               add     di,bp           ; 03    add/subtract number of bytes
                                C                                       ;       to skip in row to/from `to'
                                C                                       ;       address for scroll up/down
                                C
F2DB  FE CC                     C               dec     ah              ; 03    decrement rows to move
F2DD  74 15                     C               jz      v_mv_end        ; 04/16 did we move the last row
                                C
F2DF  8A CA                     C               mov     cl,dl           ; 02    number of columns to move
                                C                                       ;       up/down per row
F2E1  FE C8                     C               dec     al              ; 03
F2E3  75 EC                     C               jnz     v_mv_row        ; 04/16
                                C
                                C ; Worst case:[98]+(4+4+4)+[6*(1369+3+3+3+4+2+3)]+[5*16]+(4)
                                C ; = 98+12+8322+80+4 = 8516 cycles = 88.71% * 9600 cycles = 1.065ms < 1.2ms
                                C
F2E5                            C v_mv_past:
F2E5  EB B9                     C               jmp     short v_mv_lp
                                C
                                C ;---------------------------------------------------------------------------
                                C
F2E7                            C v_mv_flp:                             ; ah has number of rows to move
F2E7  8A CA                     C               mov     cl,dl           ; number of columns to move
F2E9  F3/ 26: A5                C               rep     movs    word ptr es:[di],word ptr es:[si]
                                C                                       ; move the row (es:di gets es:si)
F2EC  03 F5                     C               add     si,bp           ; add number of bytes to skip in row to
                                C                                       ; `from' address for scroll
F2EE  03 FD                     C               add     di,bp           ; add number of bytes to skip in row to
                                C                                       ; `to' address for scroll
F2F0  FE CC                     C               dec     ah
F2F2  75 F3                     C               jnz     v_mv_flp
                                C
                                C ;---------------------------------------------------------------------------
                                C
F2F4                            C v_mv_end:
F2F4  FB                        C               sti                     ; enable interrupts immediately
                                C
                                C ; Clear bl rows below.
                                C
F2F5                            C v_clr_bot:
                                C
                                C ; Clear bl rows below/above if scroll up/down (based on bp & direction flag DF).
                                C
F2F5                            C v_clr:
                                C
F2F5  8A E7                     C               mov     ah,bh
F2F7  80 3E 0049 R 07           C               cmp     byte ptr ds:[v_mode],7
F2FC  74 41                     C               je      v_clr_fast      ; jump to fast loop
                                C
F2FE  8A CA                     C               mov     cl,dl           ; cl = no. of columns to clear per row
                                C                                       ; (bl = number of rows to clear)
F300                            C v_clr_lp:
F300  B7 20                     C               mov     bh,' '          ; bh = blanking character
                                C
F302  52                        C               push    dx                      ; save dx
F303  8B 16 0063 R              C               mov     dx,word ptr ds:[v_base6845]     ; get 6845 pointer register
F307  83 C2 06                  C               add     dx,6                    ; get 6845 status register
                                C
                                C ; Wait for horizontal retrace...
                                C
F30A                            C v_clr_hi:                             ; wait till we're in a scanline
F30A  EC                        C               in      al,dx           ; get CRT status
F30B  D0 D8                     C               rcr     al,1            ; test display enable (bit #0)
F30D  72 FB                     C               jc      v_clr_hi        ; wait for display enable low (cf)
F30F  FA                        C               cli                     ; disable ints first.
                                C
F310                            C v_clr_lo:                             ; wait till start of horiz. blanking
F310  EC                        C               in      al,dx           ; 08    get CRT status
F311  D0 D8                     C               rcr     al,1            ; 02    test display enable (bit #0)
F313  73 FB                     C               jnc     v_clr_lo        ; 16/04  wait for display enable hi (cf)
                                C
F315  8A C7                     C               mov     al,bh           ; 02    al= bh= ' ' = blanking character
                                C .list
F317  AB                        C               stosw                   ; 11    clear word (es:di gets ax)
                                C
                                C ; Worst case: (8+2+16)+(8+2+4)+(2+11) = 53 cycles = 66.25% of 80 cycles
                                C
F318  EC                        C               in      al,dx           ; 08    get CRT status (vert synch slow)
F319  5A                        C               pop     dx              ; 08    restore dx
                                C
F31A  49                        C               dec     cx              ; 03    decrement columns to clear
F31B  75 08                     C               jnz     v_clr_nxt       ; 04/16 did we move the last column?
                                C
F31D  03 FD                     C               add     di,bp           ; 03    add/subtract number of bytes
                                C                                       ;       to skip in row to/from `to'
                                C                                       ;       address for clear for up/down
                                C
F31F  FE CB                     C               dec     bl              ; 03    decrement rows to clear
F321  74 28                     C               jz      v_scrl_ret       ; 04/16 did we clear the last row?
                                C
F323  8A CA                     C               mov     cl,dl           ; 02    number of columns to clear
                                C                                       ;       per row
                                C
                                C ; Worst case: [53]+(8+8)+(3+4)+(3)+(3+4+2) = 88 cycles = 11.00 usec
                                C
F325                            C v_clr_nxt:
                                C
F325  FB                        C               sti
F326  A8 08                     C               test    al,08h          ; 04    check for vertical retrace
```

```
F328  74 12            C        jz      v_clr_past          ; 16/04 jump past vertical retrace code
                       C
                       C  ; We get here if vertical retrace has started...
                       C
                       C                                    ; NOTE: interrupts disabled for 1.1ms!
                       C
F32A  8A C7            C        mov     al,bh               ; 02    al= bh= ' ' = blanking character
                       C
F32C  B7 07            C        mov     bh,V_KCLEAR         ; 04    at most 7 (not 10) rows per pass
                       C
F32E                   C  v_clr_row:
F32E  F3/ AB           C        rep     stosw               ; 09+10*80 clear the row (es:di gets ax)
                       C
F330  03 FD            C        add     di,bp               ; 03    add/subtract number of bytes
                       C                                    ;       to skip in row to/from `to'
                       C                                    ;       address for clear for up/down
                       C
F332  FE CB            C        dec     bl                  ; 03    decrement rows to clear
F334  74 15            C        jz      v_scrl_ret           ; 04/16 did we clear the last row?
                       C
F336  8A CA            C        mov     cl,dl               ; 02    number of columns to clear
                       C                                    ;       up/down per row
F338  FE CF            C        dec     bh                  ; 03    still working on 10 scan lines??
F33A  75 F2            C        jnz     v_clr_row           ; 04/16
                       C
                       C  ; Worst case:[88]+(4+4+2+4)+[10*(809+3+3+4+2+3)]+[9*16]+(4)
                       C  ; = 88+14+8240+144+4 = 8490 cycles = 88.44% * 9600 cycles = 1.061ms < 1.2ms
                       C
F33C                   C  v_clr_past:
F33C  FB               C        sti                         ; enable interrupts immediately
F33D  EB C1            C        jmp     short v_clr_lp
                       C
                       C  ;----------------------------------------------------------------
                       C
F33F                   C  v_clr_fast:                       ;  ah has attribute for blank line(s)
F33F  B0 20            C        mov     al,' '              ;  al = blanking character
                       C                                    ; (bl = number of rows to clear)
F341                   C  v_clr_flp:
F341  8A CA            C        mov     cl,dl               ; number of columns to clear
F343  F3/ AB           C        rep     stosw               ; clear the row (es:di gets ax)
F345  03 FD            C        add     di,bp               ; add number of bytes to skip in row to
                       C                                    ; `to' address for scroll
F347  FE CB            C        dec     bl
F349  75 F6            C        jnz     v_clr_flp
                       C
                       C  ;----------------------------------------------------------------
                       C
F34B                   C  v_scrl_ret:                       ; common clean up routine
F34B  FB               C        sti                         ; enable interrupts immediately
                       C
F34C  80 3E 0049 R 07  C        cmp     byte ptr ds:[v_mode],7
F351  74 03            C        je      v_scrl_mode_7
F353  A0 0065 R        C        mov     al,byte ptr ds:[v_3x8]  ;
F356                   C  v_scrl_mode_7:
                       C
                       C  ; We did't disable display during vertical retrace...
                       C
                       C  ; Clean up.
                       C
F356  FC               C        cld                         ; in case we are `call'ed & scroll down
F357  5B               C        pop     bx                  ; restore registers
F358  59               C        pop     cx
F359  5A               C        pop     dx
F35A  C3               C        ret
                       C
F35B                   C  v_scrl_up       endp
                       C
                       C  ;----------------------------------------------------------------
                       C  ;       Scroll Active Page Down         ah = 07h
                       C  ;
                       C  ;       Input:  if al = 0, then clear entire window with attribute in bh
                       C  ;               else,   al = number of rows to `scroll' down
                       C  ;                          = number of rows to clear at top of window
                       C  ;               bh      = attribute to be used on blank row(s)
                       C  ;               (ch,cl) = (row,col) of upper left corner of window from (0,0)
                       C  ;               (dh,dl) = (row,col) of lower right corner of window from (0,0)
                       C  ;       Output: ah      = attribute to be used on blank row(s)
                       C  ;               if v_mode = 7,  al = 20h = space
                       C  ;               else                al = v_3x8
                       C  ;
                       C  ;       Assume: (contents of v_base6845)+6  = status register
                       C  ;
                       C  ;       Trash:  bp, si, & di destroyed. (bx thru dx destroyed if ROM stack)
                       C  ;----------------------------------------------------------------
                       C
F35B                   C  v_scrl_dn       proc    near
                       C
                       C                  assume  cs:code, ds:data, es:v_ram, ss:nothing
                       C
F35B  FD               C        std                         ; NOTE: scroll down everything backwards
                       C
F35C  E8 F53C R        C        call    v_txt_md            ; all registers preserved
F35F  72 03            C        jb      v_txt_dn
F361  E9 D909 R        C        jmp     grf_graphics_down   ; jump if graphics
F364                   C  v_txt_dn:
                       C
F364  52               C        push    dx                  ; save registers
F365  51               C        push    cx
F366  53               C        push    bx
                       C
```

```
F367  8A D8               C          mov     bl,al                   ; save line count
F369  8B C2               C          mov     ax,dx                   ; pass lower right coordinates....
F36B  E8 F4DE R           C          call    v_scrl_pos              ; to common scroll positioning routine
F36E  74 07               C          jz      v_clr_top               ; clear rows if nothing to move
                          C
                          C ; Scroll cl rows down.
                          C
F370                      C v_mv_dn:
F370  2B F0               C          sub     si,ax                   ; subtract (bytes/row)*(rows to scroll)
                          C                                          ; from 'from' address for scroll
F372  F7 DD               C          neg     bp                      ; negate number of bytes to skip per row
F374  E9 F295 R           C          jmp     v_mv                    ; now identical to scroll up!
                          C
                          C ; Clear bl rows above.
                          C
F377                      C v_clr_top:
F377  F7 DD               C          neg     bp                      ; negate number of bytes to skip per row
F379  E9 F2F5 R           C          jmp     v_clr                   ; now identical to v_clr_bot!
                          C
F37C                      C v_scrl_dn       endp
                          C
                          C ;---------------------------------------------------------------------
                          C ;       Read Attribute & Character at Cursor    ah = 08h
                          C ;
                          C ;       Input:  bh      = current active display page (0-7)
                          C ;       Output: al      = character read
                          C ;               ah      = attribute of character read
                          C ;
                          C ;       Assume: (contents of v_base6845)+6  = status register
                          C ;
                          C ;       Trash:  si & di destroyed. (si = dx; di = bx)
                          C ;---------------------------------------------------------------------
                          C
F37C                      C v_rac   proc    near
                          C
                          C         assume  cs:code, ds:data, es:v_ram, ss:nothing
                          C
F37C  E8 F53C R           C          call    v_txt_md                ; all registers preserved
F37F  72 03               C          jb      v_txt_rac
F381  E9 D96C R           C          jmp     grf_graphics_read       ; jump if graphics
F384                      C v_txt_rac:
                          C
F384  8B FB               C          mov     di,bx                   ; save bx
F386  E8 F50D R           C          call    v_fpos                  ; ax & si destroyed.
                          C                                          ; bx = offset into current page
                          C
F389  8B F2               C          mov     si,dx                           ; save dx
F38B  BA 0006             C          mov     dx,6                            ; get 6845 status reg. offset
F38E  03 16 0063 R        C          add     dx,word ptr ds:[v_base6845]     ; add 6845 pointer
                          C
                          C ; Wait for horizontal retrace... we can't read the screen during a trace
                          C ; without disturbing the screen image.
                          C
                          C
F392                      C v_rac_inline:                           ; make sure we're in a scanline
F392  EC                  C          in      al,dx                   ; get horiz. retrace blanking status
F393  D0 D8               C          rcr     al,1                    ; test for horiz. retrace
F395  72 FB               C          jc      v_rac_inline            ; wait for display enable low (cf)
F397  FA                  C          cli                             ; disable ints FIRST
                          C
                          C                                          ; wait for blanking:
F398                      C v_rac_inblank:
F398  EC                  C          in      al,dx                   ; get retrace blanking status
F399  D0 D8               C          rcr     al,1                    ; test display enable (bit #0)
F39B  73 FB               C          jnc     v_rac_inblank           ; try again if still in scanline.
                          C
F39D  26: 8B 07           C          mov     ax,es:[bx]              ; char. and attr.  now in AX
F3A0  FB                  C          sti                             ; enable interrupts immediately
                          C
F3A1  8B D6               C          mov     dx,si                   ; restore dx
F3A3  8B DF               C          mov     bx,di                   ; restore bx
F3A5  C3                  C          ret
                          C
F3A6                      C v_rac   endp
                          C
                          C ;---------------------------------------------------------------------
                          C ;       Write Attribute & Character at Cursor   ah = 09h
                          C ;
                          C ;       Input:  al      = character to write
                          C ;               bh      = current active display page (0-7)
                          C ;               bl      = attribute of character to write
                          C ;               cx      = counter of characters to write
                          C ;               bp      = value to return in ax
                          C ;       Output: al      = character to write
                          C ;               ah      = attribute of character to write
                          C ;
                          C ;       Assume: (contents of v_base6845)+6  = status register
                          C ;
                          C ;       Trash:  bp, si & di destroyed. (si = cx; dx if ROM stack)
                          C ;---------------------------------------------------------------------
                          C
F3A6                      C v_wac   proc    near
                          C
                          C         assume  cs:code, ds:data, es:v_ram, ss:nothing
                          C
F3A6  E8 F53C R           C          call    v_txt_md                ; all registers preserved
F3A9  72 03               C          jb      v_txt_wac
F3AB  E9 DA59 R           C          jmp     grf_graphics_write      ; jump if graphics
F3AE                      C v_txt_wac:
                          C
```

```
F3AE  8B FB                   C           mov     di,bx                   ; save bx
                              C
F3B0  E8 F50D R               C           call    v_fpos                  ; ax & si destroyed.
                              C                                           ; bx = offset into current page
F3B3  87 DF                   C           xchg    bx,di                   ; restore bx; di = transfer offset
                              C
F3B5  8B C5                   C           mov     ax,bp                   ; restore ax
F3B7  8A E3                   C           mov     ah,bl                   ; transfer attribute byte to ah
F3B9  8B E8                   C           mov     bp,ax                   ; save attribute & character in bp
                              C
F3BB  8B F1                   C           mov     si,cx                           ; save cx
F3BD  52                      C           push    dx                              ; save dx
F3BE  8B 16 0063 R            C           mov     dx,word ptr ds:[v_base6845]     ; get 6845 pointer register
F3C2  83 C2 06                C           add     dx,6                            ; get 6845 status register
                              C
                              C   ; Wait for horizontal retrace blank interval ...
                              C
                              C
F3C5                          C   v_wac_hi:                               ; wait till we're in a scanline..
F3C5  EC                      C           in      al,dx                   ; get CRT status
F3C6  D0 D8                   C           rcr     al,1                    ; test display enable (bit #0)
F3C8  72 FB                   C           jc      v_wac_hi                ; wait for display enable low (cf)
F3CA  FA                      C           cli                             ; disable ints FIRST
                              C
F3CB                          C   v_wac_lo:                               ; now wait till start of blanking..
F3CB  EC                      C           in      al,dx                   ; 08    get CRT status
F3CC  D0 D8                   C           rcr     al,1                    ; 02    test display enable (bit #0)
F3CE  73 FB                   C           jnc     v_wac_lo                ; 16/04 wait for display enable hi (cf)
                              C
F3D0  8B C5                   C           mov     ax,bp                   ; 02    restore ax
F3D2  AB                      C           stosw                           ; 11    es:di gets ax (attribute & char)
                              C
F3D3  49                      C          dec     cx                      ; 02
                              C   .list
F3D4  74 04                   C           jz      v_wac_end               ; 04/16
F3D6  AB                      C           stosw                           ; 11    es:di gets ax (attribute & char)
                              C
                              C   ; Worst case: (8+2+16)+(8+2+4)+(2+11)+(2+4+11) = 70 cycles = 87.5% of 80 cycles
                              C
F3D7  FB                      C           sti                             ; enable interrupts immediately
F3D8  E2 EB                   C           loop    v_wac_hi                ; do it cx times
                              C
F3DA                          C   v_wac_end:
F3DA  FB                      C           sti                             ; enable interrupts immediately
                              C
F3DB  5A                      C           pop     dx                      ; restore dx
F3DC  8B CE                   C           mov     cx,si                   ; restore cx
F3DE  C3                      C           ret
                              C
F3DF                          C   v_wac   endp
                              C
                              C   ;---------------------------------------------------------------------
                              C   ;       Write Character at Cursor Position      ah = 0Ah
                              C   ;
                              C   ;       Input:  al      = character to write
                              C   ;               bh      = current active display page (0-7)
                              C   ;               cx      = counter of characters to write
                              C   ;               bp      = value to return in ax
                              C   ;       Output: al      = character to write
                              C   ;               ah      = top byte of offset of character in page 0
                              C   ;
                              C   ;       Assume: (contents of v_base6845)+6  = status register
                              C   ;
                              C   ;       Trash:  si & di destroyed. (si = cx; dx if ROM stack)
                              C   ;---------------------------------------------------------------------
                              C
F3DF                          C   v_wc    proc    near
                              C
                              C           assume  cs:code, ds:data, es:v_ram, ss:nothing
                              C
F3DF  E8 F53C R               C           call    v_txt_md                ; all registers preserved
F3E2  72 03                   C           jb      v_txt_wc
F3E4  E9 DA59 R               C           jmp     grf_graphics_write      ; jump if graphics
F3E7                          C   v_txt_wc:
                              C
F3E7  8B FB                   C           mov     di,bx                   ; save bx
                              C
F3E9  E8 F50D R               C           call    v_fpos                  ; si destroyed.
                              C                                           ; ax = offset into page 0
                              C                                           ; bx = offset into current page
F3EC  87 DF                   C           xchg    bx,di                   ; restore bx; di = transfer offset
                              C
                              C
F3EE  8B F1                   C           mov     si,cx                           ; save cx
F3F0  52                      C           push    dx                              ; save dx
                              C
F3F1  8B D5                   C           mov     dx,bp                           ; retrieve character in dl
F3F3  8A C2                   C           mov     al,dl                           ; get char in al (ah = attr.)
F3F5  8B E8                   C           mov     bp,ax                           ; bp = (attr. char)
                              C
F3F7  BA 0006                 C           mov     dx,6                            ; get 6845 status register
F3FA  03 16 0063 R            C           add     dx,word ptr ds:[v_base6845]     ; get 6845 pointer register
                              C
                              C   ; Wait for horizontal retrace...
                              C
F3FE                          C   v_wc_next:
                              C
F3FE                          C   v_wc_hi:
F3FE  EC                      C           in      al,dx                   ; get CRT status
F3FF  D0 D8                   C           rcr     al,1                    ; test display enable (bit #0)
```

```
F401  72 FB                   C           jc      v_wc_hi                 ; wait for display enable low (cf)
F403  FA                      C           cli                             ; disable ints FIRST
                              C
F404                          C   v_wc_lo:
F404  EC                      C           in      al,dx                   ; 08    get CRT status
F405  D0 D8                   C           rcr     al,1                    ; 02    test display enable (bit #0)
F407  73 FB                   C           jnc     v_wc_lo                 ; 16/04 wait for display enable hi (cf)
                              C
F409  8B C5                   C           mov     ax,bp                   ; 02    restore ax = (attr, char)
F40B  AA                      C           stosb                           ; 11    es:di gets al (character)
                              C
F40C  49                      C           dec     cx                      ; 02
                              C   .list
F40D  74 06                   C           jz      v_wc_end                ; 04/16
F40F  47                      C           inc     di                      ; 02    skip past attribute byte
                              C
F410  AA                      C           stosb                           ; 11    es:di gets al (character)
                              C
                              C   ; Worst case: (8+2+16)+(8+2+4)+(2+11)+(2+4+2+11) = 72 cycles = 90% of 80 cycles
                              C
F411  FB                      C           sti                             ; enable interrupts immediately
F412  47                      C           inc     di                      ; skip past attribute byte
F413  E2 E9                   C           loop    v_wc_next               ; do it cx times
                              C
F415                          C   v_wc_end:
F415  FB                      C           sti                             ; enable interrupts immediately
                              C
F416  5A                      C           pop     dx                      ; restore dx
F417  8B CE                   C           mov     cx,si                   ; restore cx
F419  C3                      C           ret
                              C
F41A                          C   v_wc    endp
                              C
                              C   ;---------------------------------------------------------------------
                              C   ;       Set Overscan, Back, & Foreground Colors ah = 0Bh
                              C   ;
                              C   ;       Input:  bh      = palette color ID to set (0-127)
                              C   ;               bl      = color value to be used with that color ID
                              C   ;       Output: ah      = v_mode
                              C   ;               al      = new v_colorpal
                              C   ;
                              C   ;       Assume: (contents of v_base6845)+5  = overscan register
                              C   ;
                              C   ;       Trash:  si & di destroyed. (si = bx; di = dx)
                              C   ;---------------------------------------------------------------------
                              C
F41A                          C   v_col   proc    near
                              C
                              C           assume  cs:code, ds:data, es:v_ram, ss:nothing
                              C
F41A  A0 0066 R               C           mov     al,byte ptr ds:[v_colorpal]     ; get current palette
                              C
F41D  8B FA                   C           mov     di,dx                           ; save dx
F41F  8B F3                   C           mov     si,bx                           ; save bx
                              C
F421  0A FF                   C           or      bh,bh                           ; palette color ID = 0?
F423  74 0A                   C           jz      v_col_0                         ; handle color ID 0
                              C
F425  24 DF                   C           and     al,0DFh                         ; clear palette select bit #5
F427  D0 DB                   C           rcr     bl,1                            ; test new color (bit #0)
F429  73 0B                   C           jnb     v_col_1                         ; if bit #0 set, all done
F42B  0C 20                   C           or      al,20h                          ; else set palette select bit #5
F42D  EB 07                   C           jmp     short v_col_1
                              C
F42F                          C   v_col_0:
F42F  80 E3 1F                C           and     bl,01Fh                         ; save bits #0-4 of new color
F432  24 E0                   C           and     al,0E0h                         ; clear bits #0-4 of old color
F434  0A C3                   C           or      al,bl                           ; and combine the two.
                              C
F436                          C   v_col_1:
F436  BA 0005                 C           mov     dx,5                            ; get 6845 overscan reg. offset
F439  03 16 0063 R            C           add     dx,word ptr ds:[v_base6845]     ; add 6845 pointer register
F43D  EE                      C           out     dx,al                           ; output selection
                              C
F43E  8B DE                   C           mov     bx,si                           ; restore bx
F440  8B D7                   C           mov     dx,di                           ; restore dx
                              C
F442  A2 0066 R               C           mov     byte ptr ds:[v_colorpal],al     ; save the value for later
F445  C3                      C           ret
                              C
F446                          C   v_col   endp
                              C
                              C   ;---------------------------------------------------------------------
                              C   ;       Write Dot (see graph.src)       ah = 0Ch
                              C   ;---------------------------------------------------------------------
                              C
                              C   ;---------------------------------------------------------------------
                              C   ;       Read Dot (see graph.src)        ah = 0Dh
                              C   ;---------------------------------------------------------------------
                              C
                              C   ;---------------------------------------------------------------------
                              C   ;       Terminal Emulator to active page        ah = 0Eh
                              C   ;
                              C   ;       Input:  al      = character to write
                              C   ;               bl      = foreground color in graphics mode
                              C   ;               bp      = value to return in ax
                              C   ;       Output: All registers saved.
                              C   ;
                              C   ;       Trash:  si & di destroyed. (si = cx; di = bx; dx if ROM stack)
                              C   ;---------------------------------------------------------------------
```

```
                                C
F446                            C  v_terminal proc near
                                C
                                C          assume  cs:code, ds:data, es:v_ram, ss:nothing
                                C
F446  3C 07                     C          cmp     al,BEL                          ; is it bell character?
F448  75 03                     C          jne     v_term_nobell
                                C
F44A  E9 F54E R                 C          jmp     v_bell
                                C
F44D                            C  v_term_nobell:
F44D  52                        C          push    dx                              ; save dx
F44E  8B F1                     C          mov     si,cx                           ; save cx
F450  8B FB                     C          mov     di,bx                           ; save bx
                                C
                                C  ; Get cursor position in active page.
                                C
F452  B7 07                     C          mov     bh,07h                          ;mask for page number, MOD 8
F454  22 3E 0062 R              C          and     bh,byte ptr ds:[v_apage]        ; get active page number (0-7)
                                C
F458  B4 03                     C          mov     ah,03h                          ; call v_r_curs_pos
F45A  CD 10                     C          INT     10h                             ; (dh,dl) = (row,col) of cursor
                                C                                                  ; (ch,cl) = cursor mode setting
                                C
F45C  8B C5                     C          mov     ax,bp                           ; restore ax
F45E  B9 0001                   C          mov     cx,1                            ; character count for write char
F461  B4 0A                     C          mov     ah,0Ah                          ; function code for write char
                                C
                                C  ; Handle special cases: dx has (row,col) of current cursor position.
                                C
F463  3C 0A                     C          cmp     al,LF                           ; is it a line feed?
F465  74 14                     C          je      v_lf
F467  3C 0D                     C          cmp     al,CR                           ; is it a carriage return?
F469  74 60                     C          je      v_cr
F46B  3C 08                     C          cmp     al,BS                           ; is it a backspace?
F46D  74 54                     C          je      v_bs
                                C
                                C  ; Normal Case: write the character
                                C
F46F  CD 10                     C          INT     10h                             ; to write the character
                                C
F471  FE C2                     C          inc     dl                              ; increment the column
F473  3A 16 004A R              C          cmp     dl,byte ptr ds:[v_width]        ; column overflow?
F477  72 13                     C          jb      v_set_new_cur                   ; set new cursor position
                                C
F479  32 D2                     C          xor     dl,dl                           ; carriage return cursor
                                C
F47B  80 3E 0049 R 48           C  v_lf:   cmp     byte ptr ds:[v_mode],72         ; is this mode 72 ?
F480  B4 31                     C          mov     ah,49                           ; mode 72 has 50 rows
F482  74 02                     C          je      v_lrow                          ; jump if mode 72
F484  B4 18                     C          mov     ah,24                           ; modes 4,5,6,64 have 25 rows
F486  3A F4                     C  v_lrow: cmp     dh,ah                           ; are we at last row yet?
F488  74 07                     C          je      v_scrl_tty                      ; if yes,  go scroll the screen
F48A  FE C6                     C          inc     dh                              ; otherwise, inc to next row
F48C                            C  v_set_new_cur:
F48C  B4 02                     C          mov     ah,02h                          ; call v_curs_pos to set new
F48E  EB 29 90                  C          jmp     v_term_ret                      ; cursor position
                                C
F491                            C  v_scrl_tty:                         ; (dh,dl) = (row,col) = (24,0) or (49,0)
F491  B4 02                     C          mov     ah,02h                          ; call v_curs_pos to set cursor
F493  CD 10                     C          INT     10h                             ; and so that we can read back
                                C                                                  ; the proper attribute byte
                                C
F495  32 E4                     C          xor     ah,ah                           ; ah = 0 for graphics
F497  E8 F53C R                 C          call    v_txt_md                        ; are we text mode?
F49A  73 04                     C          jnb     v_scrl_tty_graphics             ; jump if graphics
                                C
F49C  B4 08                     C          mov     ah,08h                          ; call v_rac
F49E  CD 10                     C          INT     10h                             ; to get attribute byte in ah
                                C
F4A0                            C  v_scrl_tty_graphics:
F4A0  33 C9                     C          xor     cx,cx                           ; (ch,cl)= upper left (row,col)
                                C                                                  ;        = (0,0)
F4A2  8A FC                     C          mov     bh,ah                           ; store attribute in bh
F4A4  B8 0601                   C          mov     ax,0601h                        ; call v_scrl_up to scroll
                                C                                                  ; one line with attribute bh
F4A7  8A 16 004A R              C          mov     dl,byte ptr ds:[v_width]        ; (dh,dl)= lower right (row,col)
F4AB  80 EA 01                  C          sub     dl,1                            ; column = v_width-1
F4AE  80 3E 0049 R 48           C          cmp     byte ptr ds:[v_mode],72         ; is this mode 72 ?
F4B3  B6 31                     C          mov     dh,49                           ; if yes then row = 49
F4B5  74 02                     C          je      v_term_ret                      ; jump if mode = 72
F4B7  B6 18                     C          mov     dh,24                           ; if not mode 72 then row = 24
                                C
F4B9                            C  v_term_ret:
F4B9  CD 10                     C          INT     10h
F4BB                            C  v_term_nop:
                                C
                                C  ; Clean up.
                                C
F4BB  8B C5                     C          mov     ax,bp                           ; restore ax
F4BD  8B DF                     C          mov     bx,di                           ; restore bx
F4BF  8B CE                     C          mov     cx,si                           ; restore cx
F4C1  5A                        C          pop     dx                              ; restore dx
F4C2  C3                        C          ret
                                C
                                C
F4C3  0A D2                     C  v_bs:   or      dl,dl                           ; back space -- column = 0 ?
F4C5  74 F4                     C          jz      v_term_nop                      ; don't change cursor position
F4C7  FE CA                     C          dec     dl
F4C9  EB C1                     C          jmp     v_set_new_cur
```

```
                                C
                                C
F4CB  32 D2                     C  v_cr:    xor     dl,dl                           ; carriage return
F4CD  EB BD                     C           jmp     v_set_new_cur
                                C
F4CF                            C  v_terminal endp
                                C
                                C  ;--------------------------------------------------------------------
                                C  ;       Read Current Video Status       ah = 0Fh
                                C  ;
                                C  ;       Input:  None.
                                C  ;       Output: ah      = number of character columns on screen
                                C  ;               al      = display mode currently set
                                C  ;               bh      = current active display page (0-7)
                                C  ;
                                C  ;       Trash:  None.
                                C  ;--------------------------------------------------------------------
                                C
F4CF                            C  v_stat   proc    near
                                C
                                C           assume  cs:code, ds:data, es:v_ram, ss:nothing
                                C
F4CF  8A 26 004A R              C           mov     ah,byte ptr ds:[v_width]
F4D3  A0 0049 R                 C           mov     al,byte ptr ds:[v_mode]
F4D6  8A 3E 0062 R              C           mov     bh,byte ptr ds:[v_apage]
F4DA  80 E7 07                  C           and     bh,07h                          ; page number mod 8
F4DD  C3                        C           ret
                                C
F4DE                            C  v_stat   endp
                                C           page
                                C  ;===============================================================================
                                C
                                C  ;--------------------------------------------------------------------
                                C  ;       Common scroll positioning and register initialization routine.
                                C  ;
                                C  ;       Input:  (ah,al) = starting (row,col) position from (0,0)
                                C  ;               = (ch,cl) for up / (dh,dl) for down
                                C  ;               bh      = attribute to be used on blank row(s)
                                C  ;               if bl = 0, then clear entire window with attribute in bh
                                C  ;               else,   bl = number of rows to `scroll' up / down
                                C  ;                       = number of rows to clear at top / bottom of window
                                C  ;               (ch,cl) = (row,col) of upper left corner of window from (0,0)
                                C  ;               (dh,dl) = (row,col) of lower right corner of window from (0,0)
                                C  ;
                                C  ;       es:     +-----------------------------------------+
                                C  ;               |    (ch,cl)                              |
                                C  ;               |       +---------------------------+     |
                                C  ;               |       |                           |     |
                                C  ;               |       |                           |     |
                                C  ;               |       |                           |     |
                                C  ;               |       |                           |     |
                                C  ;               |     ^ +---------------------------+     |
                                C  ;               |  bl | |                           |     |
                                C  ;               |     v +---------------------------+     |
                                C  ;               |                               (dh,dl)   |
                                C  ;               +-----------------------------------------+
                                C  ;
                                C  ;       Output: zf      = state of bl at entry (nz if scroll; z if clear only)
                                C  ;               ax      = (number of bytes/row) * (number of rows to scroll)
                                C  ;               = (2 * bl * v_width)
                                C  ;               bh      = attribute to be used on blank line(s)
                                C  ;               if bl = 0 at entry, then clear entire window with bl attribute
                                C  ;                       bl = dh = window height to clear
                                C  ;               else,   bl = number of rows to `scroll' up / down
                                C  ;                          = number of rows to clear at top / bottom of window
                                C  ;               ch      = zero
                                C  ;               cl      = number of rows to move
                                C  ;                       = delta of upper and lower coordinates-bl = (dh-bl)
                                C  ;               (dh,dl) = delta of upper left and lower right coordinates
                                C  ;                       = window height and width
                                C  ;               bp      = number of bytes NOT to move per row (2*(v_width -dl))
                                C  ;               si      = `from' address for scroll
                                C  ;               di      = `to' address for scroll
                                C  ;               ds      = data_seg (preserved)
                                C  ;               es      = para_mono
                                C  ;
                                C  ;                       <-- bp ->                       <-- bp ->
                                C  ;       es:     +-----------------------------------------+
                                C  ;               | si, di offset if up                     |
                                C  ;               |       +-------------------------+ ^     |
                                C  ;               |       |<---------- dl ---------->| |     |
                                C  ;               |       |                         | |     |
                                C  ;               |       |                         | | dh  |
                                C  ;               |     ^ +-------------------------+ |     |
                                C  ;               |  bl | |                         | |     |
                                C  ;               |     v +-------------------------+ v     |
                                C  ;               |                       si, di offset if down |
                                C  ;               +-----------------------------------------+
                                C  ;
                                C  ;--------------------------------------------------------------------
                                C           page
F4DE                            C  v_scrl_pos       proc    near
                                C
                                C                   assume  cs:code, ds:data, es:v_ram, ss:nothing
                                C
F4DE  E8 F52B R                 C           call    v_posn                          ; (ah,al) -> ax offset; si trash
                                C
F4E1  2A F5                     C           sub     dh,ch                           ; (dh,dl) gets delta (drow,dcol)
F4E3  FE C6                     C           inc     dh                              ; dh = number or rows
F4E5  2A D1                     C           sub     dl,cl
F4E7  FE C2                     C           inc     dl                              ; dl = number or columns
```

```
                        C
F4E9  8B 36 004E R      C          mov     si,word ptr ds:[v_top]  ; get offset of active page
F4ED  03 F0             C          add     si,ax                           ;  add offset in page => 'from'
F4EF  8B FE             C          mov     di,si                           ;  , 'to' addresses.
                        C
                        C
F4F1  33 C9             C          xor     cx,cx                           ; init count register to zero.
F4F3  8A CA             C          mov     cl,dl                           ; cx = number of columns = dl
F4F5  A1 004A R         C          mov     ax,word ptr ds:[v_width]        ; get screen width
F4F8  8B E8             C          mov     bp,ax                           ; bp = v_width
F4FA  2B E9             C          sub     bp,cx                           ; bp = (v_width - dl)
F4FC  D1 E5             C          shl     bp,1                            ; bp = 2 * (v_width - dl)
                        C
F4FE  D0 E0             C          shl     al,1                            ; al = 2 * v_width
F500  F6 E3             C          mul     bl                              ; ax = 2*v_width*no. of rows
                        C
F502  8A CE             C          mov     cl,dh
F504  2A CB             C          sub     cl,bl                           ; cl <= number of rows to move
                        C
F506  0A DB             C          or      bl,bl                           ; if rows to scroll,
F508  75 02             C          jnz     v_scrl_mv_and_clr       ; then, move & clear row, return nz.
                        C
F50A  8A DE             C          mov     bl,dh                   ; else clear dh rows only, return z.
                        C
F50C                    C v_scrl_mv_and_clr:
F50C  C3                C          ret                             ; ZF indicates state of BL at entry
                        C
F50D                    C v_scrl_pos    endp
                        C          page
                        C ;---------------------------------------------------------------------------
                        C ;       Calculates video ram buffer offset of a character in text mode
                        C ;
                        C ;       Input:  bh      = current active display page (0-7)
                        C ;       Output: bx      = offset of character in text mode at display page
                        C ;                       = (page number)*(v_height)+offset of v_curpos(bh)
                        C ;               ax      = offset of character in text mode from page 0
                        C ;
                        C ;       Trash:  si = destroyed.
                        C ;---------------------------------------------------------------------------
                        C
F50D                    C v_fpos  proc    near
                        C
                        C         assume  cs:code, ds:data, es:v_ram, ss:nothing
                        C
F50D  8A C7             C          mov     al,bh                           ; al gets page number
F50F  33 DB             C          xor     bx,bx                           ; bx = 0
F511  25 0007           C          and     ax,07h                          ; ax = page number mod 8
F514  8B F0             C          mov     si,ax                           ; si keeps page number mod 8
F516  74 07             C          jz      v_fpos_0                        ; page number = 07
                        C
F518                    C v_fpos_lp:
F518  03 1E 004C R      C          add     bx,word ptr ds:[v_height]       ; optimization: word multipli-
F51C  48                C          dec     ax                              ; cation by less than 8 without
F51D  75 F9             C          jnz     v_fpos_lp                       ; destroying dx (or cx).
                        C
F51F                    C v_fpos_0:                                       ; bx = (page number)*(v_height)
F51F  D1 E6             C          shl     si,1                            ; page number mod 8 word index
F521  8B 84 0050 R      C          mov     ax,word ptr ds:[si+v_curpos]
F525  E8 F52B R         C          call    v_posn                          ; (ah,al) -> ax offset; si trash
F528  03 DB             C          add     bx,ax                           ; bx = (page)*(v_height)+offset
F52A  C3                C          ret
                        C
F52B                    C v_fpos  endp
                        C
                        C ;---------------------------------------------------------------------------
                        C ;       Calculates video ram buffer offset of a character in text mode
                        C ;
                        C ;       Input:  (ah,al) = (row,col) position
                        C ;       Output: ax      = offset of character in text mode.
                        C ;
                        C ;       Trash:  si destroyed.
                        C ;---------------------------------------------------------------------------
                        C
F52B                    C v_posn          proc    near
                        C
                        C                 assume  cs:code, ds:data, es:v_ram, ss:nothing
                        C
F52B  8B F0             C         mov     si,ax
F52D  81 E6 00FF        C         and     si,0FFh                 ; si keeps column (al)
F531  8A C4             C         mov     al,ah                   ; al gets row (ah)
F533  F6 26 004A R      C         mul     byte ptr ds:[v_width]   ; ax gets (row * v_width)
F537  03 C6             C         add     ax,si                   ; ax gets (row * v_width)+ column
F539  D1 E0             C         shl     ax,1                    ; ax gets 2*((row * v_width)+column)
F53B  C3                C         ret
                        C
F53C                    C v_posn          endp
                        C
                        C
                        C ;---------------------------------------------------------------------------
                        C ;       Is v_mode text or graphics or black/white card?
                        C ;
                        C ;       Input:  None.
                        C ;       Output: carry flag (cf) set if text.  carry flag cleared if graphics.
                        C ;               Text Modes:     0 to 3 and 7
                        C ;               Graphics Modes: 4 to 6, 64, and 72
                        C ;
                        C ;       Trash:  None.
                        C ;---------------------------------------------------------------------------
                        C
F53C                    C v_txt_md        proc    near
```

```
                                C
                                C                       assume  cs:code, ds:data, es:v_ram, ss:nothing
                                C
F53C  80 3E 0049 R 04           C               cmp     byte ptr ds:[v_mode],4
F541  72 09                     C               jb      v_txt_ok
                                C
F543  80 3E 0049 R 07           C               cmp     byte ptr ds:[v_mode],7
F548  74 02                     C               je      v_txt_ok
F54A  F8                        C               clc                                     ; graphics mode (CF = 0)
F54B  C3                        C               ret                                     ; modes 4 to 6, 64, and 72
                                C
F54C                            C       v_txt_ok:
F54C  F9                        C               stc                                     ; text mode (CF = 1)
F54D  C3                        C               ret                                     ; modes 0 to 3 and 7
                                C
F54E                            C       v_txt_md        endp
                                C
                                C       ;-----------------------------------------------------------------------
                                C       ;       Handle BEL character: Beeps the speaker.
                                C       ;
                                C       ;       No parameters.
                                C       ;
                                C       ;-----------------------------------------------------------------------
                                C
F54E                            C       v_bell proc near
                                C
                                C                       assume  cs:code, ds:data, es:v_ram, ss:nothing
                                C
F54E  50                        C               push    ax
                                C
F54F  B0 B6                     C               mov     al,t2cmd                        ; p_8253_2, lsb 1st, mode 3,no BCD
F551  E6 43                     C               out     p_8253_ctrl,al
F553  B0 00                     C               mov     al,00h                          ; p_timer count
F555  E6 42                     C               out     p_8253_2,al                     ; least significant byte
F557  B0 06                     C               mov     al,06h
F559  E6 42                     C               out     p_8253_2,al                     ; most significant byte
                                C
F55B  E4 61                     C               in      al,p_kctrl                      ; get control data
F55D  8A E0                     C               mov     ah,al                           ; save control status
F55F  0C 03                     C               or      al,03h                          ; turn speaker on
F561  E6 61                     C               out     p_kctrl,al
                                C
F563  51                        C               push    cx
F564  B9 00C8                   C               mov     cx,200                          ;512 msec
F567                            C       bell_wait:
F567  E8 EFA7 R                 C               call    F_wait_one_ms                   ;wait for 1 ms
F56A  E2 FB                     C               loop    bell_wait
F56C  59                        C               pop     cx
                                C
F56D  8A C4                     C               mov     al,ah                           ; restore control status
F56F  E6 61                     C               out     p_kctrl,al
                                C
F571  58                        C               pop     ax
F572  C3                        C               ret                                     ; return from v_term
                                C
F573                            C       v_bell endp
                                C
F573                            C       code    ends
                                C
                                C       include com3.asm
                                C
                                C
                                C       ;=======================================================================
                                C       ;       Filename:       com3.src
                                C       ;=======================================================================
                                C
F573                            C       code    segment public 'ROM'
                                C               assume  cs:code, ds:nothing, es:nothing, ss:nothing
                                C
                                C
                                C       ;-----------------------------------------------------------------------
                                C       ;       INS8250 Asynchronous Communication Chip Baud Rate Time Constants
                                C       ;       (baud rate generator signal is 3.6864 MHz put through a
                                C       ;       divide-by-2 circuit).
                                C       ;
                                C       ;                       ((3,686,400 Hz)/2) = Input Freq.
                                C       ;       Time Constant = -------------------------------
                                C       ;                               (16)*(baud rate)
                                C       ;-----------------------------------------------------------------------
                                C
F573                            C       scc_init        proc    near                    ; ah = 00h
                                C                       assume  cs:code, ds:nothing, es:nothing, ss:nothing
                                C
F573  8A E8                     C               mov     ch,al                           ; save input parameters.
                                C
F575  32 C0                     C               xor     al,al
F577  EE                        C               out     dx,al                           ; dummy read on scc_ctl_x
F578  E8 E8E2 R                 C               call    rs_dly
F57B  EC                        C               in      al,dx                           ; insures proper register addr.
F57C  E8 E8E2 R                 C               call    rs_dly
                                C
F57F  B0 09                     C               mov     al,9                            ; select write register #9
F581  EE                        C               out     dx,al                           ; reset and disable int
F582  E8 E8E2 R                 C               call    rs_dly
F585  B0 C1                     C               mov     al,11000001b                    ; hw reset, st=0, mie=dis, nv=1
F587  EE                        C               out     dx,al
F588  E8 E8E2 R                 C               call    rs_dly
                                C
F58B  B0 04                     C               mov     al,4                            ; select write register #4
F58D  EE                        C               out     dx,al                           ; transfer parameters
```

```
F58E  E8 E8E2 R           C          call    rs_dly
                          C
F591  B0 4E               C          mov     al,01000100b                ; 16x, 8-bit synch, 1 or 2 stop
                          C
F593  8A E5               C          mov     ah,ch                       ; get input parameters.
F595  80 E4 04            C          and     ah,00000100b                ; get bit #2
F598  D0 E4               C          shl     ah,1                        ; move to bit #3
F59A  0A C4               C          or      al,ah                       ; 'or' the bit
                          C
F59C  8A E5               C          mov     ah,ch                       ; get input parameters.
F59E  80 E4 18            C          and     ah,00011000b                ; get bits #3 & 4
F5A1  B1 03               C          mov     cl,3
F5A3  D2 EC               C          shr     ah,cl                       ; move to bits #0 & 1
F5A5  0A C4               C          or      al,ah                       ; 'or' the bits
                          C
F5A7  EE                  C          out     dx,al                       ; all done!!!
F5A8  E8 E8E2 R           C          call    rs_dly
                          C
F5AB  B0 0B               C          mov     al,11                       ; select write register #11
F5AD  EE                  C          out     dx,al                       ; clock mode control
F5AE  E8 E8E2 R           C          call    rs_dly
F5B1  B0 55               C          mov     al,01010101b                ; no x,r=br,t=br,txc o,tx ck
F5B3  EE                  C          out     dx,al
F5B4  E8 E8E2 R           C          call    rs_dly
                          C
F5B7  B0 0C               C          mov     al,12                       ; select write register #12
F5B9  EE                  C          out     dx,al                       ; low byte of baud rate const
F5BA  E8 E8E2 R           C          call    rs_dly
                          C
F5BD  8A DD               C          mov     bl,ch                       ; get input parameters.
F5BF  81 E3 00E0          C          and     bx,111100000b               ; get bits #5, 6, & 7 (clear bh)
F5C3  B1 04               C          mov     cl,4
F5C5  D2 EB               C          shr     bl,cl                       ; move to bits #1,2,& 3
                          C                                              ; bx is word index
                          C
                          C  ; NOTE: These values are the SAME as the com_baud EXCEPT for the - 2!!!!
                          C
                          C  ;       mov     ax,word ptr cs:[bx+scc_baud]    ; get 8530 baud count
                          C
F5C7  2E: 8B 87 E729 R    C          mov     ax,word ptr cs:[bx+com_baud]    ; get 8250 baud count
F5CC  48                  C          dec     ax                          ; and subtract 2!!!!!
F5CD  48                  C          dec     ax
                          C
F5CE  EE                  C          out     dx,al                       ; output low byte of baud rate
F5CF  E8 E8E2 R           C          call    rs_dly
                          C
F5D2  B0 0D               C          mov     al,13                       ; select write register #13
F5D4  EE                  C          out     dx,al                       ; high byte of baud rate const
F5D5  E8 E8E2 R           C          call    rs_dly
                          C
F5D8  8A C4               C          mov     al,ah                       ; output high byte of baud rate
F5DA  EE                  C          out     dx,al
F5DB  E8 E8E2 R           C          call    rs_dly
                          C
F5DE  B0 0E               C          mov     al,14                       ; select write register #14
F5E0  EE                  C          out     dx,al                       ; baud rate generator enable
F5E1  E8 E8E2 R           C          call    rs_dly
F5E4  B0 03               C          mov     al,00000011b                ; baud generator enable & source
F5E6  EE                  C          out     dx,al
F5E7  E8 E8E2 R           C          call    rs_dly
                          C
F5EA  B0 01               C          mov     al,1                        ; select write register #1
F5EC  EE                  C          out     dx,al                       ; data xfer mode definition
F5ED  E8 E8E2 R           C          call    rs_dly
F5F0  32 C0               C          xor     al,al                       ; rx dis, pty no spec, tx dis
F5F2  EE                  C          out     dx,al                       ; ext dis
F5F3  E8 E8E2 R           C          call    rs_dly
                          C
F5F6  B0 03               C          mov     al,3                        ; select write register #3
F5F8  EE                  C          out     dx,al                       ; RxD parameters and control
F5F9  E8 E8E2 R           C          call    rs_dly
                          C
F5FC  B0 01               C          mov     al,00000001b
                          C                                              ; RxD enable
F5FE  8A DD               C          mov     bl,ch                       ; get input parameters.
F600  81 E3 0003          C          and     bx,00000011b                ; get bits #0 & 1 (clear bh)
F604  2E: 8A A7 F62E R    C          mov     ah,byte ptr cs:[bx+scc_dbit]
F609  B1 06               C          mov     cl,6
F60B  D2 E4               C          shl     ah,cl                       ; move data bits to #6 & 7
F60D  0A C4               C          or      al,ah                       ; 'or' the data bits
                          C
F60F  EE                  C          out     dx,al
F610  E8 E8E2 R           C          call    rs_dly
                          C
F613  B0 05               C          mov     al,5                        ; select write register #5
F615  EE                  C          out     dx,al                       ; TxD parameters and control
F616  E8 E8E2 R           C          call    rs_dly
                          C
F619  B0 08               C          mov     al,00001000b                ; TxD enabled. (power-up value)
F61B  95                  C          xchg    ax,bp                       ; get original function code
F61C  0A E4               C          or      ah,ah                       ; is it 0 or FF?
F61E  95                  C          xchg    ax,bp                       ; restore AX
F61F  75 02               C          jnz     scc_pwrup                   ; jump if original AH=0FFh
F621  B0 8A               C          mov     al,10001010b                ; TxD,DTR,RTS enabled. (normal)
                          C
F623                      C  scc_pwrup:
F623  D0 EC               C          shr     ah,1                        ; move bits #6 & 7 to #5 & 6
F625  0A C4               C          or      al,ah                       ; 'or' the bits
F627  EE                  C          out     dx,al
F628  E8 E8E2 R           C          call    rs_dly
```

```
                                C
F62B  E9 E88A R                 C               jmp     scc_stat                        ; return status
                                C
                                C ;-----------------------------------------------------------------------
                                C ;       Z8530 Compatible Data-Bit Definitions for Mapping   bits
                                C ;-----------------------------------------------------------------------
                                C
F62E  00                        C scc_dbit              db      00b     ;       5 data bits (0) (non- )
F62F  02                        C                       db      10b     ;       6 data bits (1) (non- )
F630  01                        C                       db      01b     ;       7 data bits (2)
F631  03                        C                       db      11b     ;       8 data bits (3)
                                C
F632                            C scc_init              endp
                                C
F632                            C code    ends

                                C include mem.asm
                                C
                                C ;=======================================================================
                                C ;       Filename:       mem.src
                                C ;
                                C ;       This module includes INT 12h, 11h, & 15h.
                                C ;
                                C ;=======================================================================
                                C
F632                            C code    segment public 'ROM'
                                C         assume  cs:code, ds:nothing, es:nothing, ss:nothing
                                C
                                C ;-----------------------------------------------------------------------
                                C ;       INT 12h -- memory size detect
                                C ;-----------------------------------------------------------------------
                                C
F841                            C         ORG     0F841h
                                C
F841                            C m_size  proc    near
                                C         assume  cs:code, ds:nothing, es:nothing, ss:nothing
                                C
F841  FB                        C         sti
F842  1E                        C         push    ds
F843  B8 0040                   C         mov     ax,data_seg
F846  8E D8                     C         mov     ds,ax
                                C
                                C         assume  cs:code, ds:data, es:nothing, ss:nothing
                                C
F848  A1 0013 R                 C         mov     ax,word ptr ds:[memory_size]
F84B  1F                        C         pop     ds
                                C
F84C  CF                        C         iret
                                C
F84D                            C m_size  endp
                                C
                                C ;-----------------------------------------------------------------------
                                C ;       INT 11h -- equipment check
                                C ;-----------------------------------------------------------------------
                                C
F84D                            C         ORG     0F84Dh
                                C
F84D                            C m_equip proc    near
                                C         assume  cs:code, ds:nothing, es:nothing, ss:nothing
                                C
F84D  FB                        C         sti
F84E  1E                        C         push    ds
F84F  B8 0040                   C         mov     ax,data_seg
F852  8E D8                     C         mov     ds,ax
                                C
                                C         assume  cs:code, ds:data, es:nothing, ss:nothing
                                C
F854  A1 0010 R                 C         mov     ax,word ptr ds:[switch_bits]
F857  1F                        C         pop     ds
                                C
F858  CF                        C         iret
                                C
F859                            C m_equip endp
                                C
                                C ;-----------------------------------------------------------------------
                                C ;       INT 15h -- cassette I/O
                                C ;-----------------------------------------------------------------------
                                C
F859                            C         ORG     0F859h
                                C
F859                            C m_cass  proc    near
                                C         assume  cs:code, ds:nothing, es:nothing, ss:nothing
                                C
F859  F9                        C         stc                             ; error
F85A  B4 86                     C         mov     ah,86h
F85C  C2 0002                   C         ret     2
                                C ;       iret
                                C
F85F                            C m_cass  endp
                                C
F85F                            C code    ends
                                C include nmi.asm
                                C
                                C ;=======================================================================
                                C ;       Filename:       nmi.src
                                C ;
                                C ;       This module includes INT 02h.
                                C ;
                                C ;=======================================================================
                                C
```

```
F85F                    C  code    segment public 'ROM'
                        C          assume  cs:code, ds:nothing, es:nothing, ss:nothing
                        C
                        C  ;--------------------------------------------------------------------------
                        C  ;       INT 02h
                        C  ;--------------------------------------------------------------------------
                        C
F85F                    C          ORG     0F85Fh
                        C
F85F                    C  n_int   proc    near
                        C          assume  cs:code, ds:nothing, es:nothing, ss:nothing
                        C
F85F  CF                C          iret
                        C
F860                    C  n_int   endp
                        C
F860                    C  code    ends
                        C
                        C  include boot2.asm
                        C
                        C  ;==========================================================================
                        C  ;       Filename:       boot.src
                        C  ;
                        C  ;       This module includes INT 19h.
                        C  ;
                        C  ;==========================================================================
                        C
F860                    C  code    segment public 'ROM'
                        C          assume  cs:code, ds:nothing, es:nothing, ss:nothing
                        C
                        C  ;==========================================================================
                        C  ;       INT 19h -- Cold boot routine:
                        C  ;
                        C  ;       This code reads Track 0, Side 0, Sector 0 into memory at
                        C  ;       0000:7C00 and iret's into the secondary boot-strap loader
                        C  ;==========================================================================
                        C  ;       Note:   The stack looks like this at exit!!!!
                        C  ;
                        C  ;                                                   High Address
                        C  ;                               ¦------------------¦ <-- sp (at entry & exit)
                        C  ;                               ¦ return fsw flags ¦
                        C  ;                               ¦------------------¦
                        C  ;                               ¦ return cs segment¦
                        C  ;                               ¦------------------¦
                        C  ;                               ¦ return ip offset ¦
                        C  ;                               ¦------------------¦ <-- sp after INT 19h trap
                        C  ;                               ¦    saved si, bp  ¦
                        C  ;                               ¦------------------¦ <-- sp at loading of return address
                        C  ;                                                     Low Address
                        C  ;
                        C  ;       Output:
                        C  ;       (DL)     = Driver Number            [ 00h -> Floppy Drive (A:);
                        C  ;                                             01h -> Floppy Drive (B:);
                        C  ;                                             80h -> Fixed Disk   (C:). ]
                        C  ;
                        C  ;       (DH)     = Head Number              [ These three parameters will
                        C  ;       (CH)     = Cylinder Number            specify the booted partition
                        C  ;       (CL)     = Sector Number              in the case of the fixed disk. ]
                        C  ;
                        C  ;       (AH)     = Successful status = 0.
                        C  ;       (AL)     = Number of Sectors read  [in order to read 512 bytes. ]
                        C  ;
                        C  ;       (ES:BX) = Address of the transfer [0000:7C00]
                        C  ;       (DS:BX)
                        C  ;
                        C  ;       (CS:IP) = Address of entry point  [0000:7C00]
                        C  ;
                        C  ;       (SS:SP) = Stack Segment and Pointer are left intact from the INT
                        C  ;                   19h invocation for multi-tasking environments.
                        C  ;
                        C  ;       (IF)    = The interrupt enable flag is left intact from the INT
                        C  ;                   19h invocation for multi-tasking environments.
                        C  ;
                        C  ;       Trash:  bp destoryed. (si, di, &  bp preserved.)
                        C  ;==========================================================================
                        C
F860                    C  bt_int  proc    near
                        C          assume  cs:code, ds:nothing, es:nothing, ss:nothing
                        C
F860  FB                C          sti                                     ; enable interrupts
F861  55                C          push    bp                              ; save BP & SI
F862  56                C          push    si
                        C
                        C          assume  ds:data                         ; reset master table ptr to ROM
F863  2E: 8E 1E E5F2 R  C          mov     ds,word ptr cs:[set_ds_word]    ; satisfy assumption
F868  C7 06 0084 R E2CE R C        mov     word ptr ds:[master_tbl_ptr+0000h],cs:(offset mastab)
F86E  8C 0E 0086 R      C          mov     word ptr ds:[master_tbl_ptr+0002h],cs
                        C
F872  BE DBB7 R         C          mov     si,cs:(offset bt_m)             ; boot strap message
F875  E8 E5FA R         C          call    DRomString                      ; print banner
                        C
F878  32 DB             C          xor     bl,bl                           ; disable error message blinking
F87A  53                C          push    bx                              ; save blink status
                        C
                        C          assume  ds:abs0, es:abs0
F87B                    C  bt_o:                                           ; boot strap outer loop
F87B  33 C0             C          xor     ax,ax                           ; AX = abs0_seg.
F87D  8E D8             C          mov     ds,ax                           ; satisfy assumptions
F87F  8E C0             C          mov     es,ax
                        C
```

```
                        C  ; Reset fd_parms table vector.
                        C
F881  C7 06 0078 R EFC7 R C          mov     word ptr ds:[int1Eloc+0],cs:(offset fd_parms)
F887  8C 0E 007A R        C          mov     word ptr ds:[int1Eloc+2],cs
                        C
                        C  ; Initialize retry loop.
                        C
F88B  BD 0003           C          mov     bp,3                        ; retry counter
F88E                    C  bt_1:                                       ; boot retry inner loop
                        C
                        C  ; Initialize the driver.
                        C
F88E  33 C0             C          xor     ax,ax                       ; AX = 0.
F890  8B D8             C          mov     bx,ax                       ; BX = 0.
F892  8B C8             C          mov     cx,ax                       ; CX = 0.
F894  8B D0             C          mov     dx,ax                       ; DX = 0.
F896  CD 13             C          INT     13h
F898  72 0A             C          jc      bt_nxt                      ; try again, if error
                        C
                        C  ; Read the boot sector.
                        C
F89A  B8 0201           C          mov     ax,0201h                    ; read one sector
                        C                                              ; bl = 0.
F89D  B7 7C             C          mov     bh,7Ch                      ; xfer address = ES:BX = 0:7C00
                        C                                              ; cx = 0.
F89F  41                C          inc     cx                          ; track 0; sector 1
                        C                                              ; dx = 0.
                        C                                              ; head  0; drive  0
                        C
F8A0  06                C          push    es                          ; save return registers
F8A1  CD 13             C          INT     13h                         ; BX,CX,DX,SI,DI,BP, & DS saved
F8A3  07                C          pop     es                          ; restore return registers
                        C
F8A4                    C  bt_nxt:
F8A4  73 1E             C          jnc     bt_ok                       ; jump if no error during read
                        C
F8A6  5B                C          pop     bx                          ; get blink status
F8A7  0A DB             C          or      bl,bl                       ; have 3 retries been completed?
F8A9  74 0E             C          jz      bt_dec                      ; jump if no
                        C
F8AB  BE DBD1 R         C          mov     si,cs:(offset bt_merr)      ; blink error message on
F8AE  78 03             C          js      bt_blnk                     ; blink state from BL above
F8B0  BE DBF6 R         C          mov     si,cs:(offset bt_spaces)    ; blink error message off
                        C
F8B3                    C  bt_blnk:
F8B3  E8 E5FA R         C          call    DRomString                  ; blink error message
F8B6  80 F3 80          C          xor     bl,10000000b                ; toggle blink state
                        C
F8B9                    C  bt_dec:
F8B9  53                C          push    bx                          ; resave blink status
F8BA  4D                C          dec     bp                          ; decrement retry count
F8BB  75 D1             C          jnz     bt_1                        ; and, try again
                        C
F8BD  5B                C          pop     bx                          ; get blink status
F8BE  80 CB 01          C          or      bl,1                        ; enable error message blinking
F8C1  53                C          push    bx                          ; save new status
                        C
F8C2  EB B7             C          jmp     bt_o                        ; and try again, for now.
                        C
F8C4                    C  bt_ok:
F8C4  BE DBF6 R         C          mov     si,cs:(offset bt_spaces)    ; blink error message off
F8C7  E8 E5FA R         C          call    DRomString
F8CA  5E                C          pop     si                          ; discard blink status
F8CB  5E                C          pop     si                          ; restore SI
F8CC  8B EC             C          mov     bp,sp
F8CE  89 5E 02          C          mov     word ptr ss:[bp+2],bx       ; return IP = BX = 7C00h
F8D1  8C 46 04          C          mov     word ptr ss:[bp+4],es       ; return CS = ES = 0000h
F8D4  5D                C          pop     bp                          ; restore BP
F8D5  CF                C          iret                                ; return flags
                        C
F8D6                    C  bt_int  endp
                        C
F8D6                    C  code    ends
                        C  include calendar.asm
                        C
                        C  ;===========================================================================
                        C  ;       Filename:       cal.src
                        C  ;
                        C  ;       This module includes c_read and c_write of INT 1Ah.
                        C  ;
                        C  ;===========================================================================
                        C
F8D6                    C  code    segment public 'ROM'
                        C          assume  cs:code, ds:nothing, es:nothing, ss:nothing
                        C
F8D6                    C  c_data1 proc
                        C
                        C  ; Days per year.
                        C
F8D6  0000              C  c_dy_yr dw      (0*366)+(0*365) ; year 0 = leap year + 0
F8D8  016E              C          dw      (1*366)+(0*365) ; year 1 = leap year + 1
F8DA  02DB              C          dw      (1*366)+(1*365) ; year 2 = leap year + 2
F8DC  0448              C          dw      (1*366)+(2*365) ; year 3 = leap year + 3
F8DE  05B5              C          dw      (1*366)+(3*365) ; year 4 = leap year + 0
F8E0  0723              C          dw      (2*366)+(3*365) ; year 5 = leap year + 1
F8E2  0890              C          dw      (2*366)+(4*365) ; year 6 = leap year + 2
F8E4  09FD              C          dw      (2*366)+(5*365) ; year 7 = leap year + 3
                        C
                        C  ; Days per month.
                        C
```

```
F8E6  1F                C  c_dy_mo db      31              ; month 0 = Jan
F8E7  1C                C             db      28              ; month 1 = Feb
F8E8  1F                C             db      31              ; month 2 = Mar
F8E9  1E                C             db      30              ; month 3 = Apr
F8EA  1F                C             db      31              ; month 4 = May
F8EB  1E                C             db      30              ; month 5 = Jun
F8EC  1F                C             db      31              ; month 6 = Jul
F8ED  1F                C             db      31              ; month 7 = Aug
F8EE  1E                C             db      30              ; month 8 = Sep
F8EF  1F                C             db      31              ; month 9 = Oct
F8F0  1E                C             db      30              ; month A = Nov
F8F1  1F                C             db      31              ; month B = Dec
                        C
F8F2                    C  c_data1 endp
                        C
                        C  ;==========================================================================
                        C  ;   Read or Write Clock Calendar Device  (c_read)
                        C  ;
                        C  ;
                        C  ;       Input:  ah = -1 Write Clock Calendar Device, then:
                        C  ;               bx =    day (from 1-1 of leap year up to 12-31 of leap year+7)
                        C  ;                  =    (0-2921) = (0-B69h)
                        C  ;               ch =    hour    (0-23)
                        C  ;               cl =    minutes (0-59)
                        C  ;       Output: ah = -1 implies date/time error
                        C  ;               ah = 0  implies date/time OK
                        C  ;
                        C  ;       Input:  ah = -2 Read Clock Calendar Device, then:
                        C  ;       Output: bx =    day (from 1-1 of leap year up to 12-31 of leap year+7)
                        C  ;               ch =    hour
                        C  ;               cl =    minutes
                        C  ;               dh =    seconds
                        C  ;               dl =    hundredths of seconds
                        C  ;
                        C  ;       Trash:  None.
                        C  ;==========================================================================
                        C
F8F2                    C  c_read  proc    near
                        C          assume  cs:code, ds:nothing, es:nothing, ss:nothing
                        C
                        C  ; Save registers.
                        C
F8F2  50                C          push    ax
                        C
                        C  ; Years.
                        C
F8F3  BA 007F           C          mov     dx,7Fh                  ; interrupts (years mod 8)
F8F6  EC                C          in      al,dx
F8F7  EC                C          in      al,dx
F8F8  E8 F945 R         C          call    c_rBCD                  ; al = years
                        C
F8FB  8A E8             C          mov     ch,al                   ; ch = saves year mod 8
                        C
                        C  ; Months.
                        C
F8FD  B2 7C             C          mov     dl,07Ch                 ; dl = tens of months port = 7Ch
F8FF  E8 F958 R         C          call    c_rhex                  ; Input:  dl = tens of mon.s port = 7Ch
                        C                                          ; Output: ax = hex of months (1-12)
                        C                                          ;         dx = day of week port = 7Ah
F902  48                C          dec     ax                      ; ax = map month (1-12) to month (0-11)
                        C
F903  8B D8             C          mov     bx,ax                   ; bx = saves month (0-11)
                        C
                        C  ; Days.
                        C
F905  4A                C          dec     dx                      ; dl = tens of days port = 79h
F906  E8 F958 R         C          call    c_rhex                  ; Input:  dl = tens of days port = 79h
                        C                                          ; Output: ax = hex of days (1-?)
                        C                                          ;         dx = tens of hours port = 77h
F909  48                C          dec     ax                      ; ax = map days (1-?) to days (0-?)
                        C
                        C  ; Calculate Day (ax has day).
                        C
F90A  8B D0             C          mov     dx,ax                   ; dx = day
                        C
                        C  ; Calculate Month (bx has month).
                        C
F90C  4B                C          dec     bx                      ; previous month
F90D  78 08             C          js      c_rm0                   ; jump if it was zero
                        C
F90F                    C  c_rmlp:
F90F  E8 F9FD R         C          call    c_gdays                 ; get days per month
F912  03 D0             C          add     dx,ax                   ; dx = day + current month
F914  4B                C          dec     bx                      ; previous month
F915  79 F8             C          jns     c_rmlp
                        C
F917                    C  c_rm0:                                  ; zero case
                        C                                          ; dx = day + month
                        C  ; Calculate Year (ch has month).
                        C
F917  33 DB             C          xor     bx,bx                   ; clear bh
F919  8A DD             C          mov     bl,ch                   ; get year mod 8
F91B  D1 E3             C          shl     bx,1                    ; make word index
F91D  2E: 03 97 F8D6 R  C          add     dx,word ptr cs:[bx+c_dy_yr]
                        C
F922  8B DA             C          mov     bx,dx                   ; bx = day + month + year
                        C
                        C  ; Hours.
                        C
F924  B2 77             C          mov     dl,077h                 ; dl = tens of hours port = 77h
```

```
F926  E8 F958 R           C          call    c_rhex                  ; Input:  dl = tens of hours port = 77h
                          C                                          ; Output: ax = hexadecimal of hours
                          C                                          ;         dx = tens of min.s port = 75h
F929  8A E8               C          mov     ch,al                   ; ch = hours
                          C
                          C  ; Minutes.
                          C                                          ; dl = tens of minutes port = 75h
F92B  E8 F958 R           C          call    c_rhex                  ; Input:  dl = tens of min.s port = 75h
                          C                                          ; Output: ax = hexadecimal of minutes
                          C                                          ;         dx = tens of sec.s port = 73h
F92E  8A C8               C          mov     cl,al                   ; cl = minutes
                          C
                          C  ; Seconds.
                          C                                          ; dl = tens of seconds port = 73h
F930  E8 F958 R           C          call    c_rhex                  ; Input:  dl = tens of sec.s port = 73h
                          C                                          ; Output: ax = hexadecimal of seconds
                          C                                          ;         dx = tenths of secs port = 71h
F933  8A F0               C          mov     dh,al                   ; dh = seconds
                          C
F935  52                  C          push    dx                      ; save seconds (dh)
                          C
                          C
                          C  ; Hundredths of Seconds.
                          C
                          C                                          ; dl = tenths of seconds port = 71h
F936  E8 F945 R           C          call    c_rBCD                  ; al = tenths of seconds
F939  8A E0               C          mov     ah,al                   ; move tenths of seconds to high byte
F93B  32 C0               C          xor     al,al                   ; ax = BCD of hundredths of seconds
F93D  E8 F962 R           C          call    c_BCD2hex               ; ax = hex of hundredths of seconds
                          C
F940  5A                  C          pop     dx                      ; restore seconds (dh)
F941  8A D0               C          mov     dl,al                   ; dl = hex of hundredths of seconds
                          C
                          C  ; Restore registers.
                          C
F943  58                  C          pop     ax
F944  C3                  C          ret
                          C
F945  51                  C  c_rBCD: push    cx                      ; save cx
F946  B9 0003             C          mov     cx,3                    ; try 3 times only!!!
F949  32 F6               C          xor     dh,dh                   ; clear dh
                          C
F94B  EC                  C  c_rBlp: in      al,dx                   ; get the byte
F94C  24 0F               C          and     al,0fh                  ; clear high nibble
F94E  3C 0A               C          cmp     al,10                   ; is it less than 10?
F950  72 04               C          jb      c_rBret                 ; if so, return
                          C
F952  E2 F7               C          loop    c_rBlp                  ; else, try again
F954  B0 01               C          mov     al,1                    ; if timeout, return one.
                          C
F956                      C  c_rBret:
F956  59                  C          pop     cx                      ; restore cx
F957  C3                  C          ret
                          C
F958                      C  c_read  endp
                          C
                          C  ;---------------------------------------------------------------------
                          C  ;       Convert to Hex  (c_rhex)
                          C  ;
                          C  ;       Inputs both BCD bytes and converts to hexdecimal word.
                          C  ;
                          C  ;       Input:  dl = pointer to tens of whatever port
                          C  ;       Output: ax = hexadecimal word (ah = 0)
                          C  ;               dx = pointer to tens of previous port (dh = 0)
                          C  ;
                          C  ;       Trash:  None.
                          C  ;---------------------------------------------------------------------
                          C
F958                      C  c_rhex  proc    near
                          C          assume  cs:code, ds:nothing, es:nothing, ss:nothing
                          C
F958  E8 F945 R           C          call    c_rBCD                  ; in from tens of whatever
F95B  8A E0               C          mov     ah,al                   ; move tens of whatever to high byte
F95D  4A                  C          dec     dx                      ; dx points to units of whatever port
F95E  E8 F945 R           C          call    c_rBCD                  ; in from units of whatever
F961  4A                  C          dec     dx                      ; dx points to tens of previous port
                          C
                          C  ;       jmp     short c_BCD2hex         ; fall through
                          C
F962                      C  c_rhex  endp
                          C
                          C  ;---------------------------------------------------------------------
                          C  ;
                          C  ;       BCD to Hexadecimal (c_BCD2hex)
                          C  ;
                          C  ;       Input:  ah = high BCD digit
                          C  ;               al = low BCD digit
                          C  ;       Output: ax = hexadecimal byte (ah = 0)
                          C  ;               dh = 0
                          C  ;       Trash:  None.
                          C  ;---------------------------------------------------------------------
                          C
F962                      C  c_BCD2hex       proc    near
                          C                  assume  cs:code, ds:nothing, es:nothing, ss:nothing
                          C
F962  8A F4               C          mov     dh,ah                   ; dh =     hi BCD digit
F964  D0 E6               C          shl     dh,1                    ; dh =  2*(hi BCD digit)
F966  D0 E6               C          shl     dh,1                    ; dh =  4*(hi BCD digit)
F968  02 F4               C          add     dh,ah                   ; dh =  5*(hi BCD digit)
F96A  D0 E6               C          shl     dh,1                    ; dh = 10*(hi BCD digit)
```

```
F96C  02 C6                   c          add     al,dh                   ; al = 10*(hi BCD digit)+(low BCD digit)
F96E  32 E4                   c          xor     ah,ah                   ; ax = 10*(hi BCD digit)+(low BCD digit)
F970  32 F6                   c          xor     dh,dh                   ; dh = 0
F972  C3                      c          ret
                              c
F973                          c  o_BCD2hex       endp
                              c
                              c  ;---------------------------------------------------------------------
                              c  ;       Write Clock Calendar Device  (c_write)
                              c  ;
                              c  ;       Input:  ah = -1
                              c  ;               bx =    day (from 1-1 of leap year up to 12-31 of leap year+7)
                              c  ;                  =    (0-2921) = (0-B69h)
                              c  ;               ch =    hour    (0-23)
                              c  ;               cl =    minutes (0-59)
                              c  ;       Output: ah = -1 implies date/time error
                              c  ;               ah = 0  implies date/time OK
                              c  ;       Trash:  None.
                              c  ;---------------------------------------------------------------------
                              c
F973                          c  c_write proc    near
                              c          assume  cs:code, ds:nothing, es:nothing, ss:nothing
                              c
                              c  ; Check for errors.
                              c
F973  80 F9 3C                c          cmp     cl,60                   ; cl = minutes (0-59)
F976  73 73                   c          jae     c_werr
F978  80 FD 18                c          cmp     ch,24                   ; ch = hour    (0-23)
F97B  73 6E                   c          jae     c_werr
F97D  81 FB 0B6A              c          cmp     bx,(2*366)+(6*365)      ; bx = day from leap year mod 8
F981  73 68                   c          jae     c_werr                  ;    = (0-2921) = (0-0B69h)
                              c
                              c  ; Save registers.
                              c
F983  50                      c          push    ax
F984  53                      c          push    bx
F985  51                      c          push    cx
F986  52                      c          push    dx
                              c
                              c  ; Initialize and Stop Clock.
                              c
F987  33 C0                   c          xor     ax,ax                   ; ax = 0
F989  E6 70                   c          out     70h,al                  ; test only port = out of test mode
F98B  E6 7E                   c          out     7Eh,al                  ; stop/start port = stop clock
                              c
                              c  ; Minutes.
                              c
F98D  B2 74                   c          mov     dl,074h                 ; dl = units of minutes port = 74h
F98F  8A C1                   c          mov     al,cl                   ; al = minutes (0-59)
F991  E8 F9EC R               c          call    c_whex                  ; Input:  al = hexadecimal of minutes
                              c                                          ;         dl = units of min.s port = 74h
                              c                                          ; Output: ax = trash
                              c                                          ;         dx = units of hours port = 76h
                              c  ; Hours.
                              c                                          ; dl = units of hours port = 76h
F994  8A C5                   c          mov     al,ch                   ; al = hours (0-23)
F996  E8 F9EC R               c          call    c_whex                  ; Input:  al = hexadecimal of hours
                              c                                          ;         dl = units of hours port = 76h
                              c                                          ; Output: ax = trash
                              c                                          ;         dx = units of days port = 78h
                              c  ; Calculate Year.
                              c
F999  8B D3                   c          mov     dx,bx                   ; dx = day from leap year mod 8
F99B  BB 0010                 c          mov     bx,(8*2)                ; word index of year
F99E                          c  c_wylp:
F99E  4B                      c          dec     bx
F99F  4B                      c          dec     bx
F9A0  2E: 3B 97 F8D6 R        c          cmp     dx,word ptr cs:[bx+c_dy_yr]
F9A5  72 F7                   c          jb      c_wylp
                              c
F9A7  2E: 2B 97 F8D6 R        c          sub     dx,word ptr cs:[bx+c_dy_yr]     ; dx = saves day of year
F9AC  D1 EB                   c          shr     bx,1                    ; bl = year mod 8
F9AE  8A EB                   c          mov     ch,bl                   ; ch = saves year mod 8
                              c
                              c  ; Calculate Days & Months.
                              c
F9B0  BB FFFF                 c          mov     bx,-1                   ; start at January
F9B3                          c  c_wmlp:
F9B3  43                      c          inc     bx                      ; next month
F9B4  E8 F9FD R               c          call    c_gdays                 ; get days per month
F9B7  2B D0                   c          sub     dx,ax
F9B9  73 F8                   c          jae     c_wmlp
                              c
F9BB  03 C2                   c          add     ax,dx                   ; bx = month (0-11)
                              c                                          ; ax = day   (0-?)
                              c  ; Days.
                              c
F9BD  B2 78                   c          mov     dl,078h                 ; dl = units of days port = 78h
F9BF  40                      c          inc     ax                      ; al = map days (0-?) to days (1-?)
F9C0  E8 F9EC R               c          call    c_whex                  ; Input:  al = hexadecimal of days
                              c                                          ;         dl = units of days port = 78h
                              c                                          ; Output: ax = trash
                              c                                          ;         dx = day of week port = 7Ah
                              c  ; Months.
                              c
F9C3  42                      c          inc     dx                      ; dl = units of months port = 7Bh
F9C4  8B C3                   c          mov     ax,bx                   ; al = month (0-11)
F9C6  40                      c          inc     ax                      ; al = map month (0-11) to month (1-12)
F9C7  E8 F9EC R               c          call    c_whex                  ; Input:  al = hexadecimal of months
                              c                                          ;         dl = units of mon.s port = 7Bh
```

```
                        C                                       ; Output: ax = trash
                        C                                       ;         dx = leap year port = 7Dh
                        C ; Leap Years.
                        C
                        C                                       ; dx = leap year port = 7Dh
F9CA  B0 08             C       mov     al,08h                  ; set leap year bit
F9CC  8A CD             C       mov     cl,ch                   ; get year mod 8
F9CE  80 E1 03          C       and     cl,03h                  ; get year mod 4
F9D1  D2 E8             C       shr     al,cl                   ; shift leap year bit into position
F9D3  EE                C       out     dx,al
                        C
                        C ; Years.
                        C
F9D4  42                C       inc     dx                      ; dx = stop/start = 7Eh
F9D5  42                C       inc     dx                      ; dx = interrupt  = 7Fh
F9D6  8A C5             C       mov     al,ch                   ; get year mod 8
F9D8  0C 08             C       or      al,08h                  ; set 'repeated interrupt' bit
F9DA  EE                C       out     dx,al
F9DB  90                C       nop
F9DC  EC                C       in      al,dx
F9DD  90                C       nop
F9DE  EC                C       in      al,dx
F9DF  90                C       nop
F9E0  EC                C       in      al,dx
                        C
                        C ; Start Clock.
                        C
F9E1  B0 FF             C       mov     al,0FFh                 ; al = 0FFh
F9E3  4A                C       dec     dx                      ; dx = stop/start = 7Eh
F9E4  EE                C       out     dx,al                   ; start clock
                        C
                        C ; Restore registers.
                        C
F9E5  5A                C       pop     dx
F9E6  59                C       pop     cx
F9E7  5B                C       pop     bx
F9E8  58                C       pop     ax
F9E9  32 E4             C       xor     ah,ah                   ; ah = 0        no error
                        C
F9EB                    C c_werr:                               ; ah = -1       error
F9EB  C3                C       ret
                        C
F9EC                    C c_write endp
                        C
                        C ;---------------------------------------------------------------------------
                        C ;       Converts hexadecimal byte to BCD and outputs both bytes. (c_whex)
                        C ;
                        C ;       Input:  al = hexadecimal byte
                        C ;               dl = pointer to units of whatever port
                        C ;       Output: dx = pointer to units of next port (dh = 0)
                        C ;
                        C ;       Trash:  ax destroyed.
                        C ;---------------------------------------------------------------------------
                        C
F9EC                    C c_whex  proc    near
                        C         assume  cs:code, ds:nothing, es:nothing, ss:nothing
                        C
F9EC  32 E4             C       xor     ah,ah                   ; ax = hexadecimal byte
F9EE  B6 0A             C       mov     dh,10                   ; dh = divisor
F9F0  F6 F6             C       div     dh                      ; ah = remainder = low BCD digit (0-9)
                        C                                       ; al = quotient  = high BCD digit
F9F2  86 C4             C       xchg    al,ah                   ; ah = quotient  = high BCD digit
                        C                                       ; al = remainder = low BCD digit (0-9)
                        C
F9F4  32 F6             C       xor     dh,dh                   ; dx points to units of whatever port
F9F6  EE                C       out     dx,al                   ; out to units of whatever
F9F7  8A C4             C       mov     al,ah                   ; move tens of whatever to low byte
F9F9  42                C       inc     dx                      ; dx points to tens of whatever port
F9FA  EE                C       out     dx,al                   ; out to tens of whatever
F9FB  42                C       inc     dx                      ; dx points to units of next port
F9FC  C3                C       ret
                        C
F9FD                    C c_whex  endp
                        C
                        C ;---------------------------------------------------------------------------
                        C ;
                        C ;       Get Days per Month. (c_gdays)
                        C ;
                        C ;       This routine calculates the number of days
                        C ;       per month based on the year without checking validity of month.
                        C ;
                        C ;       Input:  bx      = month (assumes bx < 12)
                        C ;               ch      = year
                        C ;       Output: ax      = days in month, if month valid; else garbage
                        C ;
                        C ;       Trash:  None.
                        C ;---------------------------------------------------------------------------
                        C
F9FD                    C c_gdays proc    near
                        C         assume  cs:code, ds:nothing, es:nothing, ss:nothing
                        C
F9FD  33 C0             C       xor     ax,ax                   ; clear ah
F9FF  2E: 8A 87 F8E6 R  C       mov     al,cs:[bx+c_dy_mo]
                        C
FA04  80 FB 01          C       cmp     bl,1                    ; is is February?
FA07  75 06             C       jnz     c_gret                  ; if not February, return
                        C
FA09  F6 C5 03          C       test    ch,03h                  ; if year = 0 mod 4, leap year
FA0C  75 01             C       jnz     c_gret                  ; if not leap year, ax = 28th, so return
                        C
FA0E  40                C       inc     ax                      ; load ax with February 29th
```

```
FA0F  C3                  C  c_gret: ret
                          C
FA10                      C  c_gdays endp
                          C
FA10                      C  code    ends
                             ;--------------------------------------------------------------------------
                             ;         ORG'd Font Tables
                             ;--------------------------------------------------------------------------

FA10                         code    segment public 'ROM'
                                     assume  cs:code, ds:nothing, es:nothing, ss:nothing

FA6E                                 ORG     0FA6Eh
FA6E                         font_lo_8x8     label   byte
                          C  include font108.asm
                          C
FA6E                      C  font108 proc near         ;   System Font Table for M24
                          C
FA6E  00 00 00 00         C          DB  00h,00h,00h,00h
FA72  00 00 00 00         C          DB  00h,00h,00h,00h      ;       0
FA76  7E 81 A5 81         C          DB  07eh,081h,0a5h,081h
FA7A  BD 99 81 7E         C          DB  0bdh,099h,081h,07eh  ;       1
FA7E  7E FF DB FF         C          DB  07eh,0ffh,0dbh,0ffh
FA82  C3 E7 FF 7E         C          DB  0c3h,0e7h,0ffh,07eh  ;       2
FA86  36 7F 7F 7F         C          DB  036h,07fh,07fh,07fh
FA8A  3E 1C 08 00         C          DB  03eh,01ch,08h,00h    ;       3
FA8E  08 1C 3E 7F         C          DB  08h,01ch,03eh,07fh
FA92  3E 1C 08 00         C          DB  03eh,01ch,08h,00h    ;       4
FA96  18 3C E7 E7         C          DB  018h,03ch,0e7h,0e7h
FA9A  66 18 18 3C         C          DB  066h,018h,018h,03ch  ;       5
FA9E  18 3C 7E FF         C          DB  018h,03ch,07eh,0ffh
FAA2  FF 7E 18 3C         C          DB  0ffh,07eh,018h,03ch  ;       6
FAA6  00 00 18 3C         C          DB  00h,00h,018h,03ch
FAAA  3C 18 00 00         C          DB  03ch,018h,00h,00h    ;       7
FAAE  FF FF E7 C3         C          DB  0ffh,0ffh,0e7h,0c3h
FAB2  C3 E7 FF FF         C          DB  0c3h,0e7h,0ffh,0ffh  ;       8
FAB6  00 3C 24 42         C          DB  00h,03ch,024h,042h
FABA  42 24 3C 00         C          DB  042h,024h,03ch,00h   ;       9
FABE  FF C3 99 BD         C          DB  0ffh,0c3h,099h,0bdh
FAC2  BD 99 C3 FF         C          DB  0bdh,099h,0c3h,0ffh  ;       a
FAC6  1F 07 0D 19         C          DB  01fh,07h,0dh,019h
FACA  78 CC CC 78         C          DB  078h,0cch,0cch,078h  ;       b
FACE  3C 66 66 3C         C          DB  03ch,066h,066h,03ch
FAD2  18 7E 18 18         C          DB  018h,07eh,018h,018h  ;       c
FAD6  18 18 12 12         C          DB  018h,018h,012h,012h
FADA  14 70 F0 60         C          DB  014h,070h,0f0h,060h  ;       d
FADE  1F 11 1F 11         C          DB  01fh,011h,01fh,011h
FAE2  13 37 72 20         C          DB  013h,037h,072h,020h  ;       e
FAE6  18 DB 3C E7         C          DB  018h,0dbh,03ch,0e7h
FAEA  3C DB 18 00         C          DB  03ch,0dbh,018h,00h   ;       f
FAEE  40 70 7C 7F         C          DB  040h,070h,07ch,07fh
FAF2  7C 70 40 00         C          DB  07ch,070h,040h,00h   ;       10
FAF6  01 07 1F 7F         C          DB  01h,07h,01fh,07fh
FAFA  1F 07 01 00         C          DB  01fh,07h,01h,00h     ;       11
FAFE  18 3C 7E 18         C          DB  018h,03ch,07eh,018h
FB02  18 7E 3C 18         C          DB  018h,07eh,03ch,018h  ;       12
FB06  00 33 33 33         C          DB  00h,033h,033h,033h
FB0A  33 00 33 00         C          DB  033h,00h,033h,00h    ;       13
FB0E  7F DB DB 7B         C          DB  07fh,0dbh,0dbh,07bh
FB12  1B 1B 1B 00         C          DB  01bh,01bh,01bh,00h   ;       14
FB16  3E 61 3C 66         C          DB  03eh,061h,03ch,066h
FB1A  66 3C 86 7C         C          DB  066h,03ch,086h,07ch  ;       15
FB1E  00 00 00 00         C          DB  00h,00h,00h,00h
FB22  7E 7E 7E 00         C          DB  07eh,07eh,00h        ;       16
FB26  18 3C 7E 18         C          DB  018h,03ch,07eh,018h
FB2A  7E 3C 18 7E         C          DB  07eh,03ch,018h,07eh  ;       17
FB2E  18 3C 7E 18         C          DB  018h,03ch,07eh,018h
FB32  18 18 18 18         C          DB  018h,018h,018h,018h  ;       18
FB36  18 18 18 18         C          DB  018h,018h,018h,018h
FB3A  18 7E 3C 18         C          DB  018h,07eh,03ch,018h  ;       19
FB3E  0C 06 7F 06         C          DB  0ch,06h,07fh,06h
FB42  0C 00 00 00         C          DB  0ch,00h,00h,00h      ;       1a
FB46  18 30 7F 30         C          DB  018h,030h,07fh,030h
FB4A  18 00 00 00         C          DB  018h,00h,00h,00h     ;       1b
FB4E  00 00 60 60         C          DB  00h,00h,060h,060h
FB52  60 7F 7F 00         C          DB  060h,07fh,07fh,00h   ;       1c
FB56  00 24 66 FF         C          DB  00h,024h,066h,0ffh
FB5A  66 24 00 00         C          DB  066h,024h,00h,00h    ;       1d
FB5E  00 00 18 3C         C          DB  00h,00h,018h,03ch
FB62  7E FF 00 00         C          DB  07eh,0ffh,00h,00h    ;       1e
FB66  00 FF 7E 3C         C          DB  00h,0ffh,07eh,03ch
FB6A  18 00 00 00         C          DB  018h,00h,00h,00h     ;       1f
FB6E  00 00 00 00         C          DB  00h,00h,00h,00h
FB72  00 00 00 00         C          DB  00h,00h,00h,00h      ;' ' 20
FB76  18 3C 3C 18         C          DB  018h,03ch,03ch,018h
FB7A  18 00 18 00         C          DB  018h,00h,018h,00h    ;'!' 21
FB7E  36 36 14 00         C          DB  036h,036h,014h,00h
FB82  00 00 00 00         C          DB  00h,00h,00h,00h      ;'"' 22
FB86  36 36 7F 36         C          DB  036h,036h,07fh,036h
FB8A  7F 36 36 00         C          DB  07fh,036h,036h,00h   ;'#' 23
FB8E  18 3E 60 3C         C          DB  018h,03eh,060h,03ch
FB92  06 7C 18 00         C          DB  06h,07ch,018h,00h    ;'$' 24
FB96  00 63 66 0C         C          DB  00h,063h,066h,0ch
FB9A  18 33 33 00         C          DB  018h,033h,033h,00h   ;'%' 25
FB9E  1C 36 1C 3B         C          DB  01ch,036h,01ch,03bh
FBA2  6E 66 3B 00         C          DB  06eh,066h,03bh,00h   ;'&' 26
FBA6  18 18 30 00         C          DB  018h,018h,030h,00h
FBAA  00 00 00 00         C          DB  00h,00h,00h,00h      ;''' 27
FBAE  0C 18 30 30         C          DB  0ch,018h,030h,030h
FBB2  30 30 18 0C         C          DB  030h,030h,018h,0ch   ;'(' 28
```

```
FBB6  30 18 0C 0C          C          DB  030h,018h,0ch,0ch
FBBA  0C 0C 18 30          C          DB  0ch,0ch,018h,030h    ;')' 29
FBBE  00 66 3C FF          C          DB  00h,066h,03ch,0ffh
FBC2  3C 66 00 00          C          DB  03ch,066h,00h,00h    ;'*' 2a
FBC6  00 18 18 7E          C          DB  00h,018h,018h,07eh
FBCA  18 18 00 00          C          DB  018h,018h,00h,00h    ;'+' 2b
FBCE  00 00 00 00          C          DB  00h,00h,00h,00h
FBD2  00 18 18 30          C          DB  00h,018h,018h,030h   ;',' 2c
FBD6  00 00 00 7E          C          DB  00h,00h,00h,07eh
FBDA  00 00 00 00          C          DB  00h,00h,00h,00h      ;'-' 2d
FBDE  00 00 00 00          C          DB  00h,00h,00h,00h
FBE2  00 18 18 00          C          DB  00h,018h,018h,00h    ;'.' 2e
FBE6  03 06 0C 18          C          DB  03h,06h,0ch,018h
FBEA  30 60 40 00          C          DB  030h,060h,040h,00h   ;'/' 2f
FBEE  3E 63 67 6F          C          DB  03eh,063h,067h,06fh
FBF2  7B 73 3E 00          C          DB  07bh,073h,03eh,00h   ;'0' 30
FBF6  0C 1C 0C 0C          C          DB  0ch,01ch,0ch,0ch
FBFA  0C 0C 3F 00          C          DB  0ch,0ch,03fh,00h     ;'1' 31
FBFE  1F 33 03 0E          C          DB  01fh,033h,03h,0eh
FC02  18 33 3F 00          C          DB  018h,033h,03fh,00h   ;'2' 32
FC06  1E 33 03 0E          C          DB  01eh,033h,03h,0eh
FC0A  03 33 1E 00          C          DB  03h,033h,01eh,00h    ;'3' 33
FC0E  0E 1E 36 66          C          DB  0eh,01eh,036h,066h
FC12  7F 06 0F 00          C          DB  07fh,06h,0fh,00h     ;'4' 34
FC16  3F 30 3E 03          C          DB  03fh,030h,03eh,03h
FC1A  03 33 1E 00          C          DB  03h,033h,01eh,00h    ;'5' 35
FC1E  0E 18 30 3E          C          DB  0eh,018h,030h,03eh
FC22  33 33 1E 00          C          DB  033h,033h,01eh,00h   ;'6' 36
FC26  3F 33 03 06          C          DB  03fh,033h,03h,06h
FC2A  0C 0C 0C 00          C          DB  0ch,0ch,0ch,00h      ;'7' 37
FC2E  1E 33 33 1E          C          DB  01eh,033h,033h,01eh
FC32  33 33 1E 00          C          DB  033h,033h,01eh,00h   ;'8' 38
FC36  1E 33 31 1F          C          DB  01eh,033h,031h,01fh
FC3A  03 06 1C 00          C          DB  03h,06h,01ch,00h     ;'9' 39
FC3E  00 18 18 00          C          DB  00h,018h,018h,00h
FC42  00 18 18 00          C          DB  00h,018h,018h,00h    ;':' 3a
FC46  00 18 18 00          C          DB  00h,018h,018h,00h
FC4A  00 18 18 30          C          DB  00h,018h,018h,030h   ;';' 3b
FC4E  06 0C 18 30          C          DB  06h,0ch,018h,030h
FC52  18 0C 06 00          C          DB  018h,0ch,06h,00h     ;'<' 3c
FC56  00 00 7E 00          C          DB  00h,00h,07eh,00h
FC5A  00 7E 00 00          C          DB  00h,07eh,00h,00h     ;'=' 3d
FC5E  30 18 0C 06          C          DB  030h,018h,0ch,06h
FC62  0C 18 30 00          C          DB  0ch,018h,030h,00h    ;'>' 3e
FC66  3E 63 03 06          C          DB  03eh,063h,03h,06h
FC6A  0C 00 0C 00          C          DB  0ch,00h,0ch,00h      ;'?' 3f
FC6E  3E 63 6F 6F          C          DB  03eh,063h,06fh,06fh
FC72  6E 60 3E 00          C          DB  06eh,060h,03eh,00h   ;'@' 40
FC76  08 1C 36 63          C          DB  08h,01ch,036h,063h
FC7A  7F 63 63 00          C          DB  07fh,063h,063h,00h   ;'A' 41
FC7E  7E 33 33 3E          C          DB  07eh,033h,033h,03eh
FC82  33 33 7E 00          C          DB  033h,033h,07eh,00h   ;'B' 42
FC86  1E 33 60 60          C          DB  01eh,033h,060h,060h
FC8A  61 33 1E 00          C          DB  061h,033h,01eh,00h   ;'C' 43
FC8E  7C 36 33 33          C          DB  07ch,036h,033h,033h
FC92  33 36 7C 00          C          DB  033h,036h,07ch,00h   ;'D' 44
FC96  7F 31 34 3C          C          DB  07fh,031h,034h,03ch
FC9A  34 31 7F 00          C          DB  034h,031h,07fh,00h   ;'E' 45
FC9E  7F 31 34 3C          C          DB  07fh,031h,034h,03eh
FCA2  34 30 78 00          C          DB  034h,030h,078h,00h   ;'F' 46
FCA6  1E 33 60 60          C          DB  01eh,033h,060h,060h
FCAA  67 33 1D 00          C          DB  067h,033h,01dh,00h   ;'G' 47
FCAE  63 63 63 7F          C          DB  063h,063h,063h,07fh
FCB2  63 63 63 00          C          DB  063h,063h,063h,00h   ;'H' 48
FCB6  3C 18 18 18          C          DB  03ch,018h,018h,018h
FCBA  18 18 3C 00          C          DB  018h,018h,03ch,00h   ;'I' 49
FCBE  0F 06 06 06          C          DB  0fh,06h,06h,06h
FCC2  66 66 3C 00          C          DB  066h,066h,03ch,00h   ;'J' 4a
FCC6  73 36 3C 38          C          DB  073h,036h,03eh,038h
FCCA  3C 36 73 00          C          DB  03ch,036h,073h,00h   ;'K' 4b
FCCE  78 30 30 30          C          DB  078h,030h,030h,030h
FCD2  30 33 7F 00          C          DB  030h,033h,07fh,00h   ;'L' 4c
FCD6  63 77 7F 6B          C          DB  063h,077h,07fh,06bh
FCDA  63 63 63 00          C          DB  063h,063h,063h,00h   ;'M' 4d
FCDE  63 73 7B 6F          C          DB  063h,073h,07bh,06fh
FCE2  67 63 63 00          C          DB  067h,063h,063h,00h   ;'N' 4e
FCE6  1C 36 63 63          C          DB  01ch,036h,063h,063h
FCEA  63 36 1C 00          C          DB  063h,036h,01ch,00h   ;'O' 4f
FCEE  7E 33 33 3E          C          DB  07eh,033h,033h,03eh
FCF2  30 30 78 00          C          DB  030h,030h,078h,00h   ;'P' 50
FCF6  1C 36 63 63          C          DB  01ch,036h,063h,063h
FCFA  63 36 1C 07          C          DB  063h,036h,01ch,07h   ;'Q' 51
FCFE  7E 33 33 3E          C          DB  07eh,033h,033h,03eh
FD02  36 33 33 00          C          DB  036h,033h,033h,00h   ;'R' 52
FD06  3E 63 30 18          C          DB  03eh,063h,030h,018h
FD0A  06 63 3E 00          C          DB  06h,063h,03eh,00h    ;'S' 53
FD0E  7E 5A 18 18          C          DB  07eh,05ah,018h,018h
FD12  18 18 3C 00          C          DB  018h,018h,03ch,00h   ;'T' 54
FD16  63 63 63 63          C          DB  063h,063h,063h,063h
FD1A  63 63 3E 00          C          DB  063h,063h,03eh,00h   ;'U' 55
FD1E  63 63 63 63          C          DB  063h,063h,063h,063h
FD22  36 1C 08 00          C          DB  036h,01ch,08h,00h    ;'V' 56
FD26  63 63 63 6B          C          DB  063h,063h,063h,06bh
FD2A  6B 7F 36 00          C          DB  06bh,07fh,036h,00h   ;'W' 57
FD2E  63 63 36 1C          C          DB  063h,063h,036h,01ch
FD32  36 63 63 00          C          DB  036h,063h,063h,00h   ;'X' 58
FD36  66 66 66 3C          C          DB  066h,066h,066h,03ch
FD3A  18 18 3C 00          C          DB  018h,018h,03ch,00h   ;'Y' 59
FD3E  7F 63 06 0C          C          DB  07fh,063h,06h,0ch
FD42  18 33 7F 00          C          DB  018h,033h,07fh,00h   ;'Z' 5a
```

```
FD46  3C 30 30 30        C            DB  03ch,030h,030h,030h
FD4A  30 30 3C 00        C            DB  030h,030h,03ch,00h   ;'[' 5b
FD4E  20 30 18 0C        C            DB  020h,030h,018h,0ch
FD52  06 03 01 00        C            DB  06h,03h,01h,00h      ;'\' 5c
FD56  3C 0C 0C 0C        C            DB  03ch,0ch,0ch,0ch
FD5A  0C 0C 3C 00        C            DB  0ch,0ch,03ch,00h     ;']' 5d
FD5E  08 1C 36 63        C            DB  08h,01ch,036h,063h
FD62  00 00 00 00        C            DB  00h,00h,00h,00h      ;'^' 5e
FD66  00 00 00 00        C            DB  00h,00h,00h,00h
FD6A  00 00 7F 00        C            DB  00h,00h,07fh,00h     ;'_' 5f
FD6E  18 18 0C 00        C            DB  018h,018h,0ch,00h
FD72  00 00 00 00        C            DB  00h,00h,00h,00h      ;'`' 60
FD76  00 00 3C 06        C            DB  00h,00h,03ch,06h
FD7A  3E 66 3B 00        C            DB  03eh,066h,03bh,00h   ;'a' 61
FD7E  70 30 3E 33        C            DB  070h,030h,03eh,033h
FD82  33 33 6E 00        C            DB  033h,033h,06eh,00h   ;'b' 62
FD86  00 00 3E 61        C            DB  00h,00h,03eh,061h
FD8A  60 61 3E 00        C            DB  060h,061h,03eh,00h   ;'c' 63
FD8E  0E 06 3E 66        C            DB  0eh,06h,03eh,066h
FD92  66 66 3B 00        C            DB  066h,066h,03bh,00h   ;'d' 64
FD96  00 00 3E 63        C            DB  00h,00h,03eh,063h
FD9A  7F 60 3F 00        C            DB  07fh,060h,03fh,00h   ;'e' 65
FD9E  1E 33 30 7C        C            DB  01eh,033h,030h,07ch
FDA2  30 30 78 00        C            DB  030h,030h,078h,00h   ;'f' 66
FDA6  00 00 3B 66        C            DB  00h,00h,03bh,066h
FDAA  66 3E 46 3C        C            DB  066h,03eh,046h,03ch  ;'g' 67
FDAE  70 30 36 3F        C            DB  070h,030h,036h,03fh
FDB2  33 33 73 00        C            DB  033h,033h,073h,00h   ;'h' 68
FDB6  0C 00 1C 0C        C            DB  0ch,00h,01ch,0ch
FDBA  0C 0C 1E 00        C            DB  0ch,0ch,01eh,00h     ;'i' 69
FDBE  0C 00 1E 0C        C            DB  0ch,00h,01eh,0ch
FDC2  0C CC CC 78        C            DB  0ch,0cch,0cch,078h   ;'j' 6a
FDC6  70 30 33 36        C            DB  070h,030h,033h,036h
FDCA  3C 36 73 00        C            DB  03ch,036h,073h,00h   ;'k' 6b
FDCE  1C 0C 0C 0C        C            DB  01ch,0ch,0ch,0ch
FDD2  0C 0C 1E 00        C            DB  0ch,0ch,01eh,00h     ;'l' 6c
FDD6  00 00 66 7F        C            DB  00h,00h,066h,07fh
FDDA  6B 6B 6B 00        C            DB  06bh,06bh,06bh,00h   ;'m' 6d
FDDE  00 00 6E 33        C            DB  00h,00h,06eh,033h
FDE2  33 33 33 00        C            DB  033h,033h,033h,00h   ;'n' 6e
FDE6  00 00 1E 33        C            DB  00h,00h,01eh,033h
FDEA  33 33 1E 00        C            DB  033h,033h,01eh,00h   ;'o' 6f
FDEE  00 00 6E 33        C            DB  00h,00h,06eh,033h
FDF2  33 3E 30 78        C            DB  033h,03eh,030h,078h  ;'p' 70
FDF6  00 00 3B 66        C            DB  00h,00h,03bh,066h
FDFA  66 3E 06 0F        C            DB  066h,03eh,06h,0fh    ;'q' 71
FDFE  00 00 6E 3B        C            DB  00h,00h,06eh,03bh
FE02  30 30 78 00        C            DB  030h,030h,078h,00h   ;'r' 72
FE06  00 00 3F 60        C            DB  00h,00h,03fh,060h
FE0A  3C 03 7E 00        C            DB  03ch,03h,07eh,00h    ;'s' 73
FE0E  08 18 3E 18        C            DB  08h,018h,03eh,018h
FE12  18 1B 0E 00        C            DB  018h,01bh,0eh,00h    ;'t' 74
FE16  00 00 66 66        C            DB  00h,00h,066h,066h
FE1A  66 66 3B 00        C            DB  066h,066h,03bh,00h   ;'u' 75
FE1E  00 00 63 63        C            DB  00h,00h,063h,063h
FE22  36 1C 08 00        C            DB  036h,01ch,08h,00h    ;'v' 76
FE26  00 00 63 6B        C            DB  00h,00h,063h,06bh
FE2A  6B 7F 36 00        C            DB  06bh,07fh,036h,00h   ;'w' 77
FE2E  00 00 63 36        C            DB  00h,00h,063h,036h
FE32  1C 36 63 00        C            DB  01ch,036h,063h,00h   ;'x' 78
FE36  00 00 66 66        C            DB  00h,00h,066h,066h
FE3A  66 3E 06 7C        C            DB  066h,03eh,06h,07ch   ;'y' 79
FE3E  00 00 7E 4C        C            DB  00h,00h,07eh,04ch
FE42  18 32 7E 00        C            DB  018h,032h,07eh,00h   ;'z' 7a
FE46  0E 18 18 70        C            DB  0eh,018h,018h,070h
FE4A  18 18 0E 00        C            DB  018h,018h,0eh,00h    ;'{' 7b
FE4E  18 18 18 00        C            DB  018h,018h,018h,00h
FE52  18 18 18 00        C            DB  018h,018h,018h,00h   ;'|' 7c
FE56  70 18 18 0E        C            DB  070h,018h,018h,0eh
FE5A  18 18 70 00        C            DB  018h,018h,070h,00h   ;'}' 7d
FE5E  3B 6E 00 00        C            DB  03bh,06eh,00h,00h
FE62  00 00 00 00        C            DB  00h,00h,00h,00h      ;'~' 7e
FE66  00 08 1C 36        C            DB  00h,08h,01ch,036h
FE6A  63 63 7F 00        C            DB  063h,063h,07fh,00h   ;'~' 7f
                         C   ;End of font matrix
                         C
FE6E                     C   fontlo8 endp

FE6E                         code    ends


                         C   include rtc.asm
                         C
                         C   ;===============================================================
                         C   ;       Filename:       rtc.src
                         C   ;
                         C   ;       This module includes INT 08h & 1Ah.
                         C   ;
                         C   ;===============================================================
                         C
FE6E                     C   code    segment public 'ROM'
                         C           assume  cs:code, ds:nothing, es:nothing, ss:nothing
                         C
                         C   ;===============================================================
                         C   ;       INT 1Ah -- Time of Day Software Interrupt Request Routine
                         C   ;
                         C   ;       Input:  ah = 0  Read the Clock, then:
                         C   ;       Output: cx =    High Portion of Clock (t_hi_order)
                         C   ;               dx =    Low Portion of Clock (t_low_order)
                         C   ;               al =    1 if 24 hours have elapsed (t_overflow); 0 otherwise
```

```
                        C  ;
                        C  ;       Input:  ah = 1  Set the Clock, then:
                        C  ;               cx =    High Portion of Clock (t_hi_order)
                        C  ;               dx =    Low Portion of Clock (t_low_order)
                        C  ;
                        C  ;       Trash:  ah =    (ah - 1) if ah <> 0,-1, or -2
                        C  ;===========================================================================
                        C  ;       Input:  ah = -1 Write Clock Calendar Device, then:
                        C  ;               bx =    day (from 1-1 of leap year up to 12-31 of leap year+7)
                        C  ;                  =    (0-2921) = (0-B69h)
                        C  ;               ch =    hour    (0-23)
                        C  ;               cl =    minutes (0-59)
                        C  ;       Output: ah = -1 implies date/time error
                        C  ;               ah = 0  implies date/time OK
                        C  ;
                        C  ;       Input:  ah = -2 Read Clock Calendar Device, then:
                        C  ;       Output: bx =    day (from 1-1 of leap year up to 12-31 of leap year+7)
                        C  ;               ch =    hour
                        C  ;               cl =    minutes
                        C  ;               dh =    seconds
                        C  ;               dl =    hundredths of seconds
                        C  ;
                        C  ;       Trash:  None.
                        C  ;===========================================================================
                        C
FE6E                    C          ORG     0FE6Eh
                        C
FE6E                    C  t_day   proc    near
                        C          assume  cs:code, ds:nothing, es:nothing, ss:nothing
                        C
FE6E  FB                C          sti                                     ; enable interrupts
                        C
FE6F  80 FC FE          C          cmp     ah,0FEh                         ; ZF set if FEh, CF reset if FFh
FE72  75 04             C          jne     t_nFE                           ; ah = -2 = 0FEh ?
                        C
FE74  E8 F8F2 R         C          call    c_read                          ; read calendar chip
FE77  CF                C          iret
FE78                    C  t_nFE:
                        C
FE78  72 04             C          jb      t_nFF                           ; ah = -1 = 0FFh > 0FEh ?
                        C
FE7A  E8 F973 R         C          call    c_write                         ; set calendar chip
FE7D  CF                C          iret
FE7E                    C  t_nFF:
                        C
                        C          assume  cs:code, ds:data, es:nothing, ss:nothing
                        C
FE7E  1E                C          push    ds                              ; save registers
FE7F  E8 E5F4 R         C          call    set_ds                          ; satisfy assumptions
                        C
FE82  FA                C          cli                                     ; interrupts off!
                        C                                                  ; (shared variables)
                        C
FE83  80 EC 01          C          sub     ah,1                            ; DON'T DECREMENT (CF needed!)
FE86  73 0D             C          jae     t_set                           ; ah = 0 < 1?
                        C
                        C  ; Read Time of Day.
                        C
FE88  32 E4             C          xor     ah,ah                           ; ah = 0 & ZF set!
FE8A  8B 0E 006E R      C          mov     cx,word ptr ds:[t_hi_order]
FE8E  8B 16 006C R      C          mov     dx,word ptr ds:[t_low_order]
FE92  A0 0070 R         C          mov     al,byte ptr ds:[t_overflow]     ; t_overflow = 0 by setting time!
                        C                                                  ; fall through (ah = 0 & ZF set)
                        C
FE95  75 0C             C  t_set:  jnz     t_end                           ; was ah = 1? (is ah = 0 now)?
                        C
                        C  ; Set Time of Day.
                        C
FE97  89 0E 006E R      C          mov     word ptr ds:[t_hi_order],cx     ; it's ok, if we fell through.
FE9B  89 16 006C R      C          mov     word ptr ds:[t_low_order],dx    ; (a bit slower, but smaller!)
FE9F  88 26 0070 R      C          mov     byte ptr ds:[t_overflow],ah     ; ah = 0  (in all cases...)
                        C
FEA3  1F                C  t_end:  pop     ds                              ; restore registers
FEA4  CF                C          iret
                        C
FEA5                    C  t_day   endp
                        C
                        C  ;===========================================================================
                        C  ;       INT 08h -- i8253 p_timer Hardware Interrupt Service Routine
                        C  ;===========================================================================
                        C
FEA5                    C          ORG     0FEA5h
                        C
FEA5                    C  t_int   proc    near
                        C          assume  cs:code, ds:nothing, es:nothing, ss:nothing
                        C
                        C                                                  ; interrupts off!
                        C                                                  ; (shared variables)
                        C
FEA5  50                C          push    ax                              ; preserve registers
FEA6  52                C          push    dx
FEA7  1E                C          push    ds
                        C
                        C          assume  cs:code, ds:data, es:nothing, ss:nothing
                        C
FEA8  2E: 8E 1E E5F2 R  C          mov     ds,word ptr cs:[set_ds_word]    ; satisfy assumption
                        C
                        C  ; Handle turning off floppy disk drive motor.
                        C
FEAD  FE 0E 0040 R      C          dec     byte ptr ds:[motor_count]       ; decrement motor on count
```

```
FEB1  75 08                   C          jnz     t_inc                           ; should we turn off drive?
                              C
FEB3  E8 EF4F R               C          call    stop_disk                       ; if so, stop disk motor
FEB6  80 26 003F R F0         C          and     byte ptr ds:[motor_status],0F0h ; clear low nibble of status
                              C
FEBB                          C  t_inc:
                              C
                              C  ; Increment long p_timer count
                              C
FEBB  FF 06 006C R            C          inc     word ptr ds:[t_low_order]               ; increment low byte of counter
FEBF  75 04                   C          jnz     t_hi                            ; skip t_hi_order
                              C
FEC1  FF 06 006E R            C          inc     word ptr ds:[t_hi_order]        ; increment high byte of counter
FEC5                          C  t_hi:
                              C
                              C  ; Handle 24 hour overflow situation
                              C
FEC5  81 3E 006C R 00B0       C          cmp     word ptr ds:[t_low_order],00B0h ; has 24 hours elapsed?
FECB  75 14                   C          jne     t_ofl                           ; if not, skip t_overflow
                              C
FECD  83 3E 006E R 18         C          cmp     word ptr ds:[t_hi_order],24     ; has 24 hours elapsed?
FED2  75 0D                   C          jne     t_ofl                           ; if not, skip t_overflow
                              C
FED4  C6 06 0070 R 01         C          mov     byte ptr ds:[t_overflow],01h
FED9  33 C0                   C          xor     ax,ax
FEDB  A3 006C R               C          mov     word ptr ds:[t_low_order],ax
FEDE  A3 006E R               C          mov     word ptr ds:[t_hi_order],ax
FEE1                          C  t_ofl:
                              C
FEE1  FB                      C          sti                                     ; enable interrupts
                              C                                                  ; (no more shared variables)
                              C  ; Invoke any user p_timer break routine.
                              C
FEE2  CD 1C                   C          INT     1Ch
                              C
                              C  ; Send specific end of interrupt (SEOI) to pic `command' port AFTER p_timer
                              C  ; break, because user may be out-to-lunch for quite awhile....
                              C
FEE4  B0 60                   C          mov     al,pic_seoi_0           ; specific end of interrupt command
FEE6  E6 20                   C          out     pic_0,al                ; to pic `command port.
                              C
FEE8  1F                      C          pop     ds                      ; restore registers
FEE9  5A                      C          pop     dx
FEEA  58                      C          pop     ax
FEEB  CF                      C          iret
                              C
FEEC                          C  t_int   endp
                              C
FEEC                          C  code    ends
                              C  include vector.asm
                              C
                              C  ;=============================================================================
                              C  ;       Filename:       vector.src
                              C  ;
                              C  ;       This module includes the table of ROM interrupt vectors & ill_int
                              C  ;       hardware diagnostic & illegal software interrupt service routine.
                              C  ;
                              C  ;=============================================================================
                                                05-11-84

                              C
FEEC                          C  code    segment public 'ROM'
                              C          assume  cs:code, ds:nothing, es:nothing, ss:nothing
                              C
FEF3                          C          ORG     0FEF3h
                              C
FEF3                          C  i_vec_tbl       proc    near
                              C
FEF3  FEA5 R                  C          dw      t_int                   ; int08locn     see rtc.src
FEF5  E987 R                  C          dw      k_int                   ; int09locn     see kb.src
FEF7  FF23 R                  C          dw      ill_int                 ; int0Aloc
FEF9  FF23 R                  C          dw      ill_int                 ; int0Bloc
FEFB  FF23 R                  C          dw      ill_int                 ; int0Cloc
FEFD  FF23 R                  C          dw      ill_int                 ; int0Dloc
FEFF  EF57 R                  C          dw      fd_int                  ; int0Eloc      see dsk.src
FF01  FF23 R                  C          dw      ill_int                 ; int0Floc
                              C
FF03  F065 R                  C          dw      v_io                    ; int10loc      see vid.src
FF05  F84D R                  C          dw      m_equip                 ; int11loc      see mem.src
FF07  F841 R                  C          dw      m_size                  ; int12loc      see mem.src
FF09  EC59 R                  C          dw      fd_io                   ; int13loc      see dsk.src
FF0B  E739 R                  C          dw      serial_io               ; int14loc      see com.src
FF0D  F859 R                  C          dw      m_cass                  ; int15loc      see mem.src
FF0F  E82E R                  C          dw      k_io                    ; int16loc      see kb.src
FF11  EFD2 R                  C          dw      p_io                    ; int17loc      see prn.src
                              C
FF13  F860 R                  C          dw      bt_int                  ; int18loc      (We Don't Have BASIC!)
FF15  F860 R                  C          dw      bt_int                  ; int19loc      see boot.src
FF17  FE6E R                  C          dw      t_day                   ; int1Aloc      see rtc.src
FF19  FF4B R                  C          dw      dummy_iret              ; int1Bloc      see kb.src
FF1B  FF4B R                  C          dw      dummy_iret              ; int1Cloc      see rtc.src
FF1D  F0A4 R                  C          dw      v_parms                 ; int1Dloc      see vid.src
FF1F  EFC7 R                  C          dw      fd_parms                ; int1Eloc      see dsk.src
FF21  C860 R                  C          dw      font_hi_8x8             ; int1Floc      see graph.src
                              C
FF23                          C  i_vec_tbl       endp
                              C
                              C  ;-----------------------------------------------------------------------------
                              C  ;       Interrupt Routine for Unused Hardware & Illegal Software Interrupts
```

```
                        C ;--------------------------------------------------------------------------
                        C
                        C         assume  cs:code, ds:nothing, es:nothing, ss:nothing
                        C
FF23                    C         ORG     0FF23h
                        C
FF23                    C ill_int proc    near                          ; trap for illegal interrupts
                        C
FF23  50                C         push    ax                            ; save registers
                        C                                               ; ah = -1; illegal software trap
FF24  B8 FF0B           C         mov     ax,(0FFh*100h)+0Bh            ; al = OCW3 -- read PIC's
FF27  E6 20             C         out     pic_0,al                      ; in-service register
                        C
FF29  1E                C         push    ds                            ; save registers & delay
                        C
                        C ; Determine whether it is a hardware or software interrupt.
                        C
FF2A  E4 20             C         in      al,pic_0                      ; get active PIC IR#
FF2C  0A C0             C         or      al,al                         ; are any active?
FF2E  74 0E             C         jz      ill_sw                        ; if not, illegal software trap.
                        C
                        C ; If hardware interrupt, disable the 8259 PIC from further interrupts.
                        C
FF30  8A E0             C         mov     ah,al                         ; return active PIC IR#
FF32  E4 21             C         in      al,pic_1                      ; OCW1 -- get PIC interrupt mask
FF34  0A C4             C         or      al,ah                         ; shut off (set) IR# bit.
FF36  E6 21             C         out     pic_1,al                      ; send PIC new mask.
                        C
FF38  B0 20             C         mov     al,pic_neoi                   ; OCW2 -- send PIC a
FF3A  E6 20             C         out     pic_0,al                      ; nonspecific end_of_int
                        C
                        C ; Return ah = active PIC Interrupt Number in intr_flag.
                        C
FF3C  EB 03             C         jmp     short ill_flg
                        C
FF3E                    C ill_sw:                                       ; illegal software trap; ah = -1
                        C
                        C ; Turn off floppy disk drives and notify user
                        C
FF3E  E8 E58C R         C         call    ill_trap                      ; every register but ds saved!
                        C .list
                        C         assume  cs:code, ds:nothing, es:nothing, ss:nothing
                        C
                        C
FF41                    C ill_flg:                                      ; set illegal trap flag.
                        C                                               ; illegal software trap; ah = -1
                        C
                        C         assume  cs:code, ds:data, es:nothing, ss:nothing
                        C
FF41  E8 E5F4 R         C         call    set_ds                        ; satisfy assumptions.
FF44  88 26 006B R      C         mov     byte ptr ds:[intr_flag],ah    ; return interrupt flag.
                        C
FF48  1F                C         pop     ds                            ; restore registers.
FF49  58                C         pop     ax                            ; give user a second chance
FF4A  CF                C         iret
                        C
FF4B                    C ill_int endp
                        C
                        C         assume  cs:code, ds:nothing, es:nothing, ss:nothing
                        C
FF4B                    C         ORG     0FF4Bh                        ; ORG 0FF4Bh through 0FF53h
                        C
FF4B                    C dummy_iret      proc    near                  ; `BREAK' key interrupt (1Bh)
FF4B  CF                C         iret                                  ; p_timer break interrupt (1Ch)
                        C
FF4C  CF                C         iret                                  ; 0FF4Ch -- in case someone is
FF4D  CF                C         iret                                  ; 0FF4Dh
FF4E  CF                C         iret                                  ; 0FF4Eh
FF4F  CF                C         iret                                  ; 0FF4Fh
FF50  CF                C         iret                                  ; 0FF50h
FF51  CF                C         iret                                  ; 0FF51h
FF52  CF                C         iret                                  ; 0FF52h
FF53  CF                C         iret                                  ; 0FF53h
                        C
FF54                    C dummy_iret      endp
                        C
FF54                    C code    ends
                        C include prnscr.asm
                        C
                        C ;==========================================================================
                        C ;       Filename:       prnscr.src
                        C ;
                        C ;       This module includes INT 05h.
                        C ;
                        C ;==========================================================================
                        C
                        C
FF54                    C code    segment public 'ROM'
                        C         assume  cs:code, ds:nothing, es:nothing, ss:nothing
                        C
                        C ;--------------------------------------------------------------------------
                        C ;       INT 05h -- Print Screen
                        C ;
                        C ;       Input:  None.
                        C ;       Output: None.
                        C ;       Trash:  None. (Uses byte at 50:0 = 40:0100 as monitor lock:
                        C ;                                  0 indicates monitor not locked.
                        C ;                                  1 indicates monitor locked.
                        C ;                                 -1 indicates printer error.
                        C ;--------------------------------------------------------------------------
```

```
                                C
                                C
FF54                            C          ORG     0FF54h
                                C
FF54                            C  s_int   proc    near
                                C          assume  cs:code, ds:nothing, es:nothing, ss:nothing
                                C
FF54  1E                        C          push    ds                      ; save ds
                                C
                                C
                                C          assume  cs:code, ds:data, es:nothing, ss:nothing
                                C
FF55  2E: 8E 1E E5F2 R          C          mov     ds,word ptr cs:[set_ds_word] ; satisfy assumptions
                                C
FF5A  52                        C          push    dx
FF5B  B2 01                     C          mov     dl,1                    ; 1 = locked
FF5D  F0/ 86 16 0100            C  lock    xchg    byte ptr ds:[100h],dl   ; check monitor lock
FF62  FE CA                     C          dec     dl                      ; already locked ?
FF64  74 4C                     C          jz      s_nop                   ; if set, do nothing
FF66  FB                        C          sti                             ; enable interrupts after
                                C                                          ; monitor lock code!!!!
                                C
FF67  51                        C          push    cx                      ; save registers
FF68  53                        C          push    bx
FF69  50                        C          push    ax
                                C
                                C  ; Get Current Video Width.
                                C
FF6A  B4 0F                     C          mov     ah,0Fh                  ; call v_video_state
FF6C  CD 10                     C          INT     10h                     ; Output: ah = crt_cols
                                C                                          ;         al = crt_mode
                                C                                          ;         bh = active_page
                                C
FF6E  8A DC                     C          mov     bl,ah                   ; save ah = crt_cols
                                C
                                C  ; Get Current Cursor Position.
                                C
FF70  B4 03                     C          mov     ah,03h                  ; call v_read_cursor
FF72  CD 10                     C          INT     10h                     ; Input:  bh = active_page
                                C                                          ; Output:
                                C                                          ; (dh,dl) = (row,col) of cursor
                                C                                          ; (ch,cl) = cursor mode setting
                                C
FF74  8A EB                     C          mov     ch,bl                   ;get crt_cols
FF76  52                        C          push    dx                      ; save row, col of cursor
FF77  B1 FF                     C          mov     cl,-1                   ; initialize cl = printer error
                                C
                                C  ; Loop Through the Screen.
                                C
FF79  BA 0000                   C          mov     dx,0                    ; (dh,dl) = (row,col) of origin
                                C
FF7C  E8 FFB5 R                 C          call    s_eol                   ; print a new line
FF7F  75 24                     C          jnz     s_err                   ; any errors?
                                C
FF81  E8 FFC9 R                 C  s_lp:   call    s_get                   ; get next character from screen
                                C
                                C  ; Map Invalid Characters to Space.
                                C
FF84  3C 00                     C          cmp     al,0                    ; check validity of character.
FF86  75 02                     C          jne     s_ok                    ; if valid, we're ok.
FF88  B0 20                     C          mov     al,' '                  ; if invalid, print a space.
FF8A                            C  s_ok:
                                C
                                C  ; Print the Character.
                                C
FF8A  E8 FFBC R                 C          call    s_out
FF8D  75 16                     C          jnz     s_err                   ; any errors?
                                C
                                C  ; Advance to Next Character.
                                C
FF8F  FE C2                     C          inc     dl                      ; advance column (1-crt_cols)
FF91  3A D5                     C          cmp     dl,ch                   ; dl < ch = crt_cols?
FF93  7C EC                     C          jl      s_lp                    ; if so, continue
                                C
FF95  E8 FFB5 R                 C          call    s_eol                   ; else print a new line
FF98  75 0B                     C          jnz     s_err                   ; any errors?
                                C
FF9A  32 D2                     C          xor     dl,dl                   ; move column back to 0
FF9C  FE C6                     C          inc     dh                      ; advance row (1-25)
FF9E  80 FE 19                  C          cmp     dh,25                   ; dh < 25
FFA1  7C DE                     C          jl      s_lp                    ; if so, continue
                                C
FFA3  33 C9                     C          xor     cx,cx                   ; set cl = printer no error
                                C  ;       jmp     short s_err             ; fall through
                                C
FFA5  5A                        C  s_err:  pop     dx                      ; restore dx=(row,col) of cursor
FFA6  B4 02                     C          mov     ah,02h                  ; call v_set_cpos
FFA8  CD 10                     C          INT     10h                     ; Input: bh = active_page
                                C                                          ;        dx =(row,col) of cursor
                                C
FFAA  FB                        C          sti                             ; disable interrupts during
                                C                                          ; monitor lock code!!!!
FFAB  88 0E 0100                C          mov     byte ptr ds:[100h],cl   ; reset monitor lock
                                C
FFAF  58                        C          pop     ax                      ; restore registers
FFB0  5B                        C          pop     bx
FFB1  59                        C          pop     cx
FFB2  5A                        C  s_nop:  pop     dx
FFB3  1F                        C          pop     ds
FFB4  CF                        C          iret
                                C
                                C
```

```
FFB5  B0 0A                    C  s_eol:  mov     al,LF                ; print LF & CR
FFB7  E8 FFBC R                C          call    s_out
FFBA  B0 0D                    C          mov     al,CR                ; print CR
                               C  ;       jmp     short s_out          ; fall through
                               C
FFBC                           C  s_out:                               ; prints out byte in al
FFBC  52                       C          push    dx                   ; save dx
FFBD  BA 0000                  C          mov     dx,0                 ; address printer port 0.
FFC0  B4 00                    C          mov     ah,0                 ; write byte to port 0
FFC2  CD 17                    C          INT     17h
FFC4  5A                       C          pop     dx                   ; restore dx
                               C  ;       test    ah,025h              ; test for any errors?
FFC5  F6 C4 01                 C          test    ah,001h              ; test for time out?
FFC8  C3                       C          ret
FFC9                           C  s_get:          ; Set Cursor Position and get character @ curs. position
                               C
FFC9  B4 02                    C          mov     ah,02h               ; call v_set_cpos
FFCB  CD 10                    C          INT     10h                  ; Input: bh = active_page
                               C                                       ;        dx =(row,col) of cursor
                               C  ; Read Character at Cursor Position.
                               C
FFCD  B4 08                    C          mov     ah,08h               ; call v_read_ac_current
FFCF  CD 10                    C          INT     10h                  ; Input:  bh = active_page
                               C                                       ; Output: al = character read
FFD1  C3                       C          ret                          ;         ah = attribute
                               C
FFD2                           C  s_int   endp
                               C
FFD2                           C  code    ends

                                  ;--------------------------------------------------------------------------
                                  ;       CPU System Reset Vector
                                  ;--------------------------------------------------------------------------

FFD2                              code    segment public 'ROM'
                                          assume  cs:code, ds:nothing, es:nothing, ss:nothing

FFF0                                      ORG     0FFF0h              ; F000:FFF0 = FFFF0 = FFFF:0000

FFF0                              vector          proc    near

FFF0  EA                                  db      0EAh                ; jmp intersegment  F000:(offset diagnostics_1)
FFF1  DD5A R                              dw      diagnostics_1       ; instruction pointer cs:(offset diagnostics_1)
FFF3  F000                                dw      code_seg            ; code segment          F000h

FFF5  30 35 2F 30 33 2F                   db      '05/03/84'          ; release marker (exactly 8 bytes!!!!)
      38 34

FFFD  00                          chk_hi  db      0                   ; space for checksum of F000:E000 to F000:FFFF

FFFE                              vector          endp    near

FFFE                              code    ends

                                          end

Macros:

                N a m e                   Length

INPORT . . . . . . . . . . . . .          0005
JMPF . . . . . . . . . . . . . .          0001
OUTPORT. . . . . . . . . . . . .          0005

Structures and records:

                N a m e                   Width   # fields
                                          Shift   Width   Mask    Initial

CDB__1 . . . . . . . . . . . . .          0008    0003
  ZZP. . . . . . . . . . . . . .          0006    0002    00C0    0000
  DRIVENUM . . . . . . . . . . .          0005    0001    0020    0000
  HEADNUM. . . . . . . . . . . .          0000    0005    001F    0000
H_CMD_STRUC. . . . . . . . . . .          0004    0003
  BX_OPCODE. . . . . . . . . . .          0000
  H_CMD_SUBR . . . . . . . . . .          0001
  H_CMD_MODE_DMA . . . . . . . .          0003
PARAMETER_TABLE. . . . . . . . .          0010    000A
  P_CYLS . . . . . . . . . . . .          0000
  P_HEADS. . . . . . . . . . . .          0002
  P_WRITE_CUR. . . . . . . . . .          0003
  P_PRECOMP. . . . . . . . . . .          0005
  P_ECC_LEN. . . . . . . . . . .          0007
  P_CONTROL_BYTE . . . . . . . .          0008
  P_TIMEOUT. . . . . . . . . . .          0009
  P_FMT_TIMEOUT. . . . . . . . .          000A
  P_DRVDIAG_TIMEOUT. . . . . . .          000B
  P_ZZJ. . . . . . . . . . . . .          000C

Segments and groups:

                N a m e                   Size    align   combine class

ABS0 . . . . . . . . . . . . .            0080    PARA    PUBLIC  'RAM'
CODE . . . . . . . . . . . . .            FFFE    PARA    PUBLIC  'ROM'
DATA . . . . . . . . . . . . .            0088    PARA    PUBLIC  'RAM'
STACK_RAM. . . . . . . . . . .            0000    PARA    PUBLIC  'RAM'
V_RAM. . . . . . . . . . . . .            0000    PARA    PUBLIC  'RAM'

Symbols:
```

Name	Type	Value	Attr	
ABSO_SEG	Number	0000		
ADDR_MARK_ERROR	Number	0002		
ALLOW_DMA3	Number	0003		
ALT_INPUT	L BYTE	0019	DATA	
ALT_KEY	Number	0038		
ALT_RET	L WORD	E2CA	CODE	
ALT_SHIFT	Number	0008		
AP_FINISH	L NEAR	D530	CODE	
ASSIGN_PARAM	N PROC	D504	CODE	Length =002E
ASSIGN_PARAM_1	N PROC	D532	CODE	Length =006B
ASSIGN_PARAM_BX	Number	000C		
ASSIGN_PARAM_INT	Number	0009		
A_FINISH	L NEAR	D59C	CODE	
BADTRACK_ERR	Number	000B		
BANNER_M	L BYTE	DB8C	CODE	
BEL	Number	0007		
BELL_WAIT	L NEAR	F567	CODE	
BIOS_BREAK	L BYTE	0071	DATA	
BOOT_A_DISK	N PROC	D12A	CODE	Length =0021
BOOT_INDIRECT	L WORD	D126	CODE	
BS	Number	0008		
BT_BLNK	L NEAR	F8B3	CODE	
BT_DEC	L NEAR	F8B9	CODE	
BT_I	L NEAR	F88E	CODE	
BT_INT	N PROC	F860	CODE	Length =0076
BT_JMP	N PROC	E6F2	CODE	Length =0003
BT_M	L BYTE	DBB7	CODE	
BT_MERR	L BYTE	DBD1	CODE	
BT_NXT	L NEAR	F8AA	CODE	
BT_O	L NEAR	F87B	CODE	
BT_OK	L NEAR	F8C4	CODE	
BT_SPACES	L BYTE	DBF6	CODE	
BUFFER_END	L WORD	0082	DATA	
BUFFER_HEAD	L WORD	001A	DATA	
BUFFER_START	L WORD	0080	DATA	
BUFFER_TAIL	L WORD	001C	DATA	
BXDIAG_BX	Number	00E4		
BXDIAG_INT	Number	0014		
BX_ERROR_TABLE	L NEAR	D49F	CODE	
BX_READY	L NEAR	D438	CODE	
B_FAILED	L NEAR	D12A	CODE	
B_LOST	L NEAR	D144	CODE	
B_LP	L NEAR	D12A	CODE	
B_OK	L NEAR	D11F	CODE	
B_TRY_HDU	L NEAR	D10C	CODE	
CAPS_LOCK_KEY	Number	003A		
CAPS_LOCK_MODE	Number	0040		
CAPS_LOCK_SHIFT	Number	0040		
CHKTRK_BX	Number	0005		
CHK_HI	L BYTE	FFFD	CODE	
CHK_LO	L BYTE	C000	CODE	
CMD_BLOCK	L BYTE	0042	DATA	
CMD_ERROR	Number	0001		
CNTRL_KEY	Number	001D		
CNTRL_SHIFT	Number	0004		
CODE_SEG	Number	F000		
COLOR_POINTER	Number	03D4		
COMMAND	N PROC	D3DD	CODE	Length =003B
COMMCONTROL	Number	0065		
COM_BAUD	L WORD	E729	CODE	
COM_CTS	Number	0010		
COM_DATA1	N PROC	E729	CODE	Length =0010
COM_DATA_A	Number	03F8		
COM_DATA_B	Number	02F8		
COM_DSR	Number	0020		
COM_DTR	Number	0001		
COM_FE	Number	0008		
COM_GB	N PROC	E92A	CODE	Length =0023
COM_ID_A	Number	03FA		
COM_ID_B	Number	02FA		
COM_INIT	N PROC	E78F	CODE	Length =003A
COM_OE	Number	0002		
COM_PB	N PROC	E8EC	CODE	Length =002F
COM_PE	Number	0004		
COM_RTS	Number	0002		
COM_RXD	Number	0001		
COM_STAT	L NEAR	E87F	CODE	
COM_TE	Number	0080		
COM_TXD	Number	0020		
CONTROLB	Number	0062		
CONTROL_BYTE	L BYTE	0076	DATA	
CR	Number	000D		
CRC_ERROR	Number	0010		
C_BCD2HEX	N PROC	F962	CODE	Length =0011
C_DATA1	N PROC	F8D6	CODE	Length =001C
C_DY_MO	L BYTE	F8E6	CODE	
C_DY_YR	L WORD	F8D6	CODE	
C_FINISH	L NEAR	DA17	CODE	
C_GDAYS	N PROC	F9FD	CODE	Length =0013
C_GRET	L NEAR	FA0F	CODE	
C_RBCD	L NEAR	F945	CODE	
C_RBLP	L NEAR	F94B	CODE	
C_RBRET	L NEAR	F956	CODE	
C_READ	N PROC	F8F2	CODE	Length =0066
C_RHEX	N PROC	F958	CODE	Length =000A
C_RMO	L NEAR	F917	CODE	
C_RMLP	L NEAR	F90F	CODE	

```
C_WAIT . . . . . L NEAR  D404   CODE
C_WERR . . . . . L NEAR  F9EB   CODE
C_WHEX . . . . . N PROC  F9EC   CODE    Length =0011
C_WMLP . . . . . L NEAR  F9B3   CODE
C_WRITE. . . . . N PROC  F973   CODE    Length =0079
C_WYLP . . . . . L NEAR  F99E   CODE
DATA_OUT . . . . . N PROC  D59D   CODE    Length =0014
DATA_SEG . . . . . Number 0040
DCOLON . . . . . N PROC  E626   CODE    Length =000C
DCRLF. . . . . N PROC  E619   CODE    Length =000D
DELETE_KEY . . . . . Number 0053
DHEXBYTE . . . . . N PROC  E643   CODE    Length =000D
DHEXLONG . . . . . N PROC  E632   CODE    Length =000A
DHEXNIB. . . . . N PROC  E650   CODE    Length =0015
DHEXWORD . . . . . N PROC  E63C   CODE    Length =0007
DIAGNOSTICS_1. . . . . N PROC  DD5A   CODE    Length =038F
DISABLE_DISK_INTERRUPTS. . . . . N PROC  D5FA   CODE    Length =0035
DISALLOW_DMA3. . . . . Number 0007
DISKETTE_IO1 . . . . . L NEAR  EC80   CODE
DISKETTE_STATUS. . . . . L BYTE  0041   DATA
DISK_STATUS. . . . . L BYTE  0074   DATA
DISP_PASS. . . . . L NEAR  DF1A   CODE
DLX_KB . . . . . Number 0001
DMA_ADDR_0 . . . . . Number 0000
DMA_ADDR_1 . . . . . Number 0002
DMA_ADDR_2 . . . . . Number 0004
DMA_ADDR_3 . . . . . Number 0006
DMA_CMD_DISABLE. . . . . Number 0004
DMA_CMD_ENABLE . . . . . Number 0000
DMA_COMMAND. . . . . Number 0008
DMA_COUNT_0. . . . . Number 0001
DMA_COUNT_1. . . . . Number 0003
DMA_COUNT_2. . . . . Number 0005
DMA_COUNT_3. . . . . Number 0007
DMA_ERROR. . . . . Number 0008
DMA_FF_CLR . . . . . Number 000C
DMA_MASK_BIT . . . . . Number 000A
DMA_MASK_CLR . . . . . Number 000E
DMA_MASK_WRITE . . . . . Number 000F
DMA_MASTER_CLR . . . . . Number 000D
DMA_MODE . . . . . Number 000B
DMA_MODE_0 . . . . . Number 0058
DMA_MODE_1 . . . . . Number 0041
DMA_MODE_2 . . . . . Number 0056
DMA_MODE_3 . . . . . Number 0043
DMA_REQUEST. . . . . Number 0009
DMA_SEGM_0 . . . . . Number 0080
DMA_SEGM_1 . . . . . Number 0082
DMA_SEGM_2 . . . . . Number 0081
DMA_SEGM_3 . . . . . Number 0083
DMA_SEG_ERROR. . . . . Number 0009
DMA_STATUS . . . . . Number 0008
DMA_TEMP . . . . . Number 000D
DMA_UNMASK_0 . . . . . Number 0000
DNUM . . . . . N PROC  E665   CODE    Length =0008
DNUMW. . . . . N PROC  E66D   CODE    Length =0031
DNUMW_LOOP . . . . . L NEAR  E677   CODE
DNUMW_SKIP . . . . . L NEAR  E68D   CODE
DNUMW_SPACES . . . . . L NEAR  E685   CODE
DO_READY . . . . . L NEAR  D5A7   CODE
DROMSTRING . . . . . N PROC  E5FA   CODE    Length =0008
DRVDIAG_BX . . . . . Number 00E3
DSTRING. . . . . N PROC  E602   CODE    Length =0017
DS_LP. . . . . L NEAR  E609   CODE
DS_RET . . . . . L NEAR  E614   CODE
DUMMY_IRET . . . . . N PROC  FF4B   CODE    Length =0009
ECC_USED_ERR . . . . . Number 0011
E_LOSE . . . . . L NEAR  D3D3   CODE
E_READ_SENSE . . . . . L NEAR  D366   CODE
FAIL_M . . . . . L BYTE  DC45   CODE
FAR_CALLS. . . . . F PROC  C004   CODE    Length =005C
FDC_ERROR. . . . . Number 0020
FDU_DATA1. . . . . N PROC  EE20   CODE    Length =0009
FDU_DATA2. . . . . N PROC  EFC7   CODE    Length =000B
FD_INT . . . . . N PROC  EF57   CODE    Length =0014
FD_IO. . . . . F PROC  EC59   CODE    Length =0066
FD_PARMS . . . . . L BYTE  EFC7   CODE
FINISHED . . . . . L NEAR  D693   CODE
FLAGS_DATA1. . . . . N PROC  C000   CODE    Length =0004
FMTBAD_BX. . . . . Number 0007
FMTDRV_BX. . . . . Number 0004
FMTDRV_INT . . . . . Number 0007
FMTTRK_BX. . . . . Number 0006
FONTH16. . . . . N PROC  C860   CODE    Length =0400
FONTL016 . . . . . N PROC  C060   CODE    Length =0800
FONTL08. . . . . N PROC  FA6E   CODE    Length =0400
FONT_HI_8X8. . . . . L BYTE  C860   CODE
FONT_LO_8X16 . . . . . L BYTE  C060   CODE
FONT_LO_8X8. . . . . L BYTE  FA6E   CODE
FORMAT_ERROR . . . . . L NEAR  D69F   CODE
F_BUFOFF . . . . . Text    [bp+8]
F_CHECK_VALID. . . . . N PROC  EFAD   CODE    Length =001A
F_COMMAND. . . . . Text    [bp+3]
F_CONT . . . . . L NEAR  EF26   CODE
F_CV_RET . . . . . L NEAR  EFC5   CODE
F_CYL. . . . . Text    [bp+7]
F_DRIVE. . . . . Text    [bp+0]
F_FORMAT_CMD . . . . . Number 004D
F_GB_DECODE. . . . . L NEAR  EE65   CODE
F_GB_JMP . . . . . L NEAR  EE6D   CODE
```

```
F_GB_LOOP. . . . . .    L NEAR  EE34    CODE
F_GB_LOOP1 . . . . .    L NEAR  EE5E    CODE
F_GB_OUT . . . . . .    L NEAR  EE44    CODE
F_GB_RET . . . . . .    L NEAR  EE71    CODE
F_GB_TABLE . . . . .    L BYTE  EE20    CODE
F_GET_BYTE . . . . .    N PROC  EE29    CODE    Length =004A
F_GET_VAR. . . . . .    N PROC  EDFF    CODE    Length =000D
F_HEAD . . . . . .      Text    [bp+1]
F_HEAD_SETTLE. . . .    L NEAR  EF41    CODE
F_HLT. . . . . .        Number  0001
F_HUT. . . . . .        Number  000F
F_IO1. . . . . .        L NEAR  EC79    CODE
F_IO_RET . . . . . .    L NEAR  EC9B    CODE
F_MOTOR_ON . . . . .    N PROC  ED14    CODE    Length =0025
F_MOTOR_PORT . . . .    Number  03F2
F_MOTOR_WAIT . . . .    Number  0025
F_MO_RET . . . . . .    L NEAR  ED38    CODE
F_NDMA . . . . . .      Number  0000
F_NEC_DATA . . . . .    Number  03F5
F_NEC_RDY. . . . . .    N PROC  EF6B    CODE    Length =0017
F_NEC_RESET. . . . .    N PROC  EC18    CODE    Length =0018
F_NEC_RESET_RET. . .    L NEAR  EC2F    CODE
F_NEC_STATUS . . . .    Number  03F4
F_NR1. . . . . .        L NEAR  EF6E    CODE
F_NR_RET . . . . . .    L NEAR  EF80    CODE
F_NUMSECS. . . . . .    Text    [bp+2]
F_PB_ERRET . . . . .    L NEAR  EE18    CODE
F_PB_RET . . . . . .    L NEAR  EE17    CODE
F_PUT_BYTE . . . . .    N PROC  EE0C    CODE    Length =0014
F_R1 . . . . . .        L NEAR  ECD8    CODE
F_R2 . . . . . .        L NEAR  ECE0    CODE
F_RD1. . . . . .        L NEAR  ED7D    CODE
F_RDATA. . . . . .      N PROC  ED65    CODE    Length =0022
F_RD_LOOP. . . . . .    L NEAR  ED78    CODE
F_READ_CMD . . . . .    Number  00E6
F_REAL_DRIVE . . . .    Text    [bp+8]
F_RECAL_CMD. . . . .    Number  0007
F_RESET. . . . . .      N PROC  ECBF    CODE    Length =0055
F_RW1. . . . . .        L NEAR  EDB3    CODE
F_RW2. . . . . .        L NEAR  EDDD    CODE
F_RW3. . . . . .        L NEAR  EDF4    CODE
F_RW_COMMON. . . . .    N PROC  ED87    CODE    Length =0078
F_RW_RET . . . . . .    L NEAR  EDFC    CODE
F_RW_SKIP. . . . . .    L NEAR  EDD7    CODE
F_S1 . . . . . .        L NEAR  EF02    CODE
F_SD1. . . . . .        L NEAR  EE92    CODE
F_SD2. . . . . .        L NEAR  EEBD    CODE
F_SD_RET . . . . . .    L NEAR  EECC    CODE
F_SECNUM . . . . . .    Text    [bp+6]
F_SEEK . . . . . .      N PROC  EECD    CODE    Length =007A
F_SEEK_CMD . . . . .    Number  000F
F_SET_DMA. . . . . .    N PROC  EE73    CODE    Length =005A
F_SIS. . . . . .        N PROC  EF82    CODE    Length =000B
F_SNSDRV_CMD . . . .    Number  0004
F_SNSINT_CMD . . . .    Number  0008
F_SPECIFY_CMD. . . .    Number  0003
F_SRT_HB . . . . . .    Number  000C
F_SRT_96 . . . . . .    Number  000E
F_S_RECAL. . . . . .    L NEAR  EEE1    CODE
F_S_RET. . . . . .      L NEAR  EF46    CODE
F_TABLE. . . . . .      L WORD  EC8F    CODE
F_WAIT_FOR_NEC . . .    N PROC  EF8D    CODE    Length =0020
F_WAIT_ONE_MS. . . .    N PROC  EF97    CODE    Length =0008
F_WD1. . . . . .        L NEAR  ED5C    CODE
F_WDATA. . . . . .      N PROC  ED39    CODE    Length =002C
F_WD_LOOP. . . . . .    L NEAR  ED57    CODE
F_WRITE_CMD. . . . .    Number  00C5
GAME_CARD. . . . . .    Number  0201
GET_DRIVETYPE_VECTOR .  N PROC  D5C1    CODE    Length =0039
GRF_GRAPHICS_DOWN. .    N PROC  D909    CODE    Length =0063
GRF_GRAPHICS_READ. .    N PROC  D96C    CODE    Length =00ED
GRF_GRAPHICS_UP. . .    N PROC  D874    CODE    Length =0055
GRF_GRAPHICS_WRITE .    N PROC  DA59    CODE    Length =0106
GRF_LIGHT_PEN. . . .    N PROC  D7CD    CODE    Length =0003
GRF_READ_DOT . . . .    N PROC  D7D0    CODE    Length =001A
GRF_WRITE_DOT. . . .    N PROC  D7EA    CODE    Length =002B
G_72 . . . . . .        L NEAR  DB6F    CODE
G_8X16_2 . . . . . .    L NEAR  DA3D    CODE
G_8X8_2. . . . . .      L NEAR  DA7C    CODE
G_ADDR . . . . . .      N PROC  D815    CODE    Length =005F
G_ADDR_TEST. . . . .    L NEAR  DA84    CODE
G_ALIGN_DOT. . . . .    L NEAR  D808    CODE
G_BITMASK. . . . . .    L NEAR  DB62    CODE
G_CHAR_LP. . . . . .    L NEAR  DB14    CODE
G_CMP_MOD. . . . . .    L NEAR  D92F    CODE
G_COLOR_TABLE. . . .    L BYTE  DB5B    CODE
G_CURS_OFF . . . . .    N PROC  DB5F    CODE    Length =001E
G_DETMODE. . . . . .    L NEAR  DAA7    CODE
G_EXP_BYT. . . . . .    L NEAR  DB26    CODE
G_FILLER . . . . . .    N PROC  D8EB    CODE    Length =001E
G_F_CONT . . . . . .    L NEAR  DA15    CODE
G_F_EXIT . . . . . .    L NEAR  DA53    CODE
G_F_I_LP . . . . . .    L NEAR  D8ED    CODE
G_F_MACH . . . . . .    L NEAR  DA1A    CODE
G_F_S_LP . . . . . .    L NEAR  D8EF    CODE
G_HI_WR. . . . . .      L NEAR  DACB    CODE
G_IA_LP. . . . . .      L NEAR  DB1F    CODE
G_I_A_LP . . . . . .    L NEAR  DADB    CODE
G_JSFT_DOT . . . . .    L NEAR  D7E5    CODE
G_LDS_R. . . . . .      L NEAR  D98C    CODE
```

```
G_LDS_W. . . . . .          L NEAR  DA9F    CODE
G_LINELP . . . . . .        L NEAR  DAD7    CODE
G_MASK . . . . . .          L NEAR  D5E2    CODE
G_MATCHB . . . . . .        L NEAR  DA0F    CODE
G_MEDGET . . . . . .        L NEAR  D9DC    CODE
G_MED_BIT. . . . . .        L NEAR  D9F5    CODE
G_MED_IA . . . . . .        L NEAR  D9E1    CODE
G_MED_STORE. . . . .        L NEAR  DB3B    CODE
G_MED_WR . . . . . .        L NEAR  DB03    CODE
G_M_AREA . . . . . .        L NEAR  D8CC    CODE
G_RDLOOP . . . . . .        L NEAR  D9AC    CODE
G_RD_IA. . . . . .          L NEAR  D9B0    CODE
G_RD_MED . . . . . .        L NEAR  D9DA    CODE
G_REPCHAR. . . . . .        L NEAR  DAD0    CODE
G_RETURN . . . . . .        L NEAR  DB57    CODE
G_SCAN_LP. . . . . .        L NEAR  DB1A    CODE
G_SCROLLER . . . . .        N PROC  D8C9    CODE    Length =0022
G_SELFONT. . . . . .        L NEAR  DA91    CODE
G_SETDOWN. . . . . .        L NEAR  D93D    CODE
G_SET_UP . . . . . .        L NEAR  D8A9    CODE
G_SKP_1. . . . . .          L NEAR  D826    CODE
G_SKP_2. . . . . .          L NEAR  D82C    CODE
G_SKP_3. . . . . .          L NEAR  D832    CODE
G_SKP_4. . . . . .          L NEAR  D855    CODE
G_SKP_5. . . . . .          L NEAR  D873    CODE
G_SUPER_WR . . . . .        L NEAR  DAC7    CODE
G_TEST_ADDR. . . . .        L NEAR  DA49    CODE
G_TINYTEXT . . . . .        L NEAR  DAC3    CODE
G_TST_MOD. . . . . .        L NEAR  D89A    CODE
G_T_XOR. . . . . .          L NEAR  DAE3    CODE
G_UNREVERSE_VIDEO_LOOP .    L NEAR  D9D3    CODE
G_W_BYTE . . . . . .        L NEAR  DAEA    CODE
G_XORBIT . . . . . .        L NEAR  D7FF    CODE
HARD_DISK_INT13. . . .      L NEAR  D1B5    CODE
HDU_BSY. . . . . .          Number  0008
HDU_CD . . . . . .          Number  0004
HDU_DMA_SEGM . . . . .      Number  0082
HDU_IO . . . . . .          Number  0002
HDU_IRQ5 . . . . . .        Number  0020
HDU_PARM_TBL . . . . .      L NEAR  D14B    CODE
HDU_REQ. . . . . .          Number  0001
HD_ERROR . . . . . .        L BYTE  0042    DATA
HF_NUM . . . . . .          L BYTE  0075    DATA
H_ABS_M. . . . . .          L BYTE  D0AE    CODE
H_BAD_COMMAND. . . . .      L NEAR  D25A    CODE
H_BAD_M. . . . . .          L BYTE  D0A2    CODE
H_BOOT . . . . . .          F PROC  D0E9    CODE    Length =0041
H_CMD_TABLE. . . . .        L NEAR  D26D    CODE
H_CMD_WAIT . . . . . .      N PROC  D43B    CODE    Length =0064
H_C_ERR. . . . . .          Number  0002
H_DATA1. . . . . .          N PROC  D49F    CODE    Length =0040
H_DATA2. . . . . .          N PROC  D6AC    CODE    Length =0121
H_DMA. . . . . .            N PROC  D2C1    CODE    Length =007D
H_DMA_BUFF . . . . . .      L NEAR  D32C    CODE
H_DMA_ERR. . . . . .        L NEAR  D336    CODE
H_DMA_LEN. . . . . .        L NEAR  D304    CODE
H_DMA_LONG . . . . . .      L NEAR  D322    CODE
H_ERR_CHK. . . . . .        N PROC  D3AA    CODE    Length =0093
H_ERR_M. . . . . .          L BYTE  D7B7    CODE
H_ERR_RET. . . . . .        L NEAR  D3D2    CODE
H_FINISH . . . . . .        L NEAR  D2A2    CODE
H_FMT. . . . . .            N PROC  D640    CODE    Length =006C
H_FMT_M. . . . . .          L BYTE  D733    CODE
H_GOOD_M . . . . . .        L BYTE  D0A6    CODE
H_HAD_ERR. . . . . .        L NEAR  D352    CODE
H_INIT . . . . . .          N PROC  CF9F    CODE    Length =011C
H_INT. . . . . .            N PROC  D62F    CODE    Length =0011
H_INTRO_M. . . . . .        L BYTE  D6AC    CODE
H_IO . . . . . .            F PROC  D1AB    CODE    Length =0116
H_MAKE_CDB . . . . . .      L NEAR  D1DE    CODE
H_NDMA . . . . . .          N PROC  D33E    CODE    Length =000C
H_NONE_M . . . . . .        L BYTE  D784    CODE
H_PASS_M . . . . . .        L BYTE  D750    CODE
H_SAVE_REGS. . . . . .      L NEAR  D1C4    CODE
ILL_FLG. . . . . .          L NEAR  FFA1    CODE
ILL_INT. . . . . .          N PROC  FF23    CODE    Length =0028
ILL_LN . . . . . .          L NEAR  E5C4    CODE
ILL_LP . . . . . .          L NEAR  E5C9    CODE
ILL_M1 . . . . . .          L BYTE  DC1B    CODE
ILL_M2 . . . . . .          L BYTE  DC34    CODE
ILL_M3 . . . . . .          L BYTE  DC3A    CODE
ILL_SW . . . . . .          L NEAR  FF3E    CODE
ILL_TRAP . . . . . .        N PROC  E58C    CODE    Length =0049
INIT_CONTROLLER_LP . .      L NEAR  CFED    CODE
INIT_DRIVE_LP. . . . .      L NEAR  D022    CODE
INIT_DRIVE_WIN . . . .      L NEAR  D047    CODE
INIT_ERR . . . . . .        Number  0007
INIT_START_TIMER . . .      L NEAR  D028    CODE
INIT_TRY_DRIVE . . . .      L NEAR  D036    CODE
INSERT_KEY . . . . . .      Number  0052
INSERT_MODE. . . . . .      Number  0080
INSERT_SHIFT . . . . .      Number  0080
INT00LOCN. . . . . .        L DWORD 0000    ABS0
INT01LOCN. . . . . .        L DWORD 0004    ABS0
INT02LOCN. . . . . .        L DWORD 0008    ABS0
INT03LOCN. . . . . .        L DWORD 000C    ABS0
INT04LOCN. . . . . .        L DWORD 0010    ABS0
INT05LOCN. . . . . .        L DWORD 0014    ABS0
INT06LOCN. . . . . .        L DWORD 0018    ABS0
INT07LOCN. . . . . .        L DWORD 001C    ABS0
```

```
INT08LOCN. . . . . .  L DWORD 0020    ABS0
INT09LOCN. . . . . .  L DWORD 0024    ABS0
INT0ALOCN. . . . . .  L DWORD 0028    ABS0
INT0BLOCN. . . . . .  L DWORD 002C    ABS0
INT0CLOCN. . . . . .  L DWORD 0030    ABS0
INT0DLOCN. . . . . .  L DWORD 0034    ABS0
INT0ELOCN. . . . . .  L DWORD 0038    ABS0
INT0FLOCN. . . . . .  L DWORD 003C    ABS0
INT10LOCN. . . . . .  L DWORD 0040    ABS0
INT11LOCN. . . . . .  L DWORD 0044    ABS0
INT12LOCN. . . . . .  L DWORD 0048    ABS0
INT13LOCN. . . . . .  L DWORD 004C    ABS0
INT14LOCN. . . . . .  L DWORD 0050    ABS0
INT15LOCN. . . . . .  L DWORD 0054    ABS0
INT16LOCN. . . . . .  L DWORD 0058    ABS0
INT17LOCN. . . . . .  L DWORD 005C    ABS0
INT18LOCN. . . . . .  L DWORD 0060    ABS0
INT19LOCN. . . . . .  L DWORD 0064    ABS0
INT1ALOCN. . . . . .  L DWORD 0068    ABS0
INT1BLOCN. . . . . .  L DWORD 006C    ABS0
INT1CLOCN. . . . . .  L DWORD 0070    ABS0
INT1DLOCN. . . . . .  L DWORD 0074    ABS0
INT1ELOCN. . . . . .  L DWORD 0078    ABS0
INT1FLOCN. . . . . .  L DWORD 007C    ABS0
INTR_FLAG. . . . . .  L BYTE  006B    DATA
IO_ROM_INIT. . . . .  L WORD  0067    DATA
IO_ROM_SEG . . . . .  L WORD  0069    DATA
I_ALT_CONT . . . . .  L NEAR  E4F0    CODE
I_ALT_CPU. . . . . .  L NEAR  E4D7    CODE
I_ALT_ECHO . . . . .  L NEAR  E534    CODE
I_ALT_END. . . . . .  L NEAR  E567    CODE
I_ALT_FOUND. . . . .  L NEAR  E511    CODE
I_ALT_INQ. . . . . .  L NEAR  E522    CODE
I_ALT_RESTART. . . .  L NEAR  E585    CODE
I_ALT_SELECT_M . . .  L BYTE  DD35    CODE
I_ALT_TEST . . . . .  L NEAR  E506    CODE
I_CAL. . . . . .      L NEAR  DFF9    CODE
I_CAL_0. . . . . .    L NEAR  E04E    CODE
I_CAL_1_1_80 . . . .  L NEAR  E01E    CODE
I_CAL_END. . . . . .  L NEAR  E07B    CODE
I_CAL_ERR. . . . . .  L NEAR  E061    CODE
I_CAL_MAX. . . . . .  L NEAR  E02D    CODE
I_CAL_OK . . . . . .  L NEAR  E073    CODE
I_CAL_VAL. . . . . .  L BYTE  DD51    CODE
I_COM_M. . . . . .    L BYTE  DCDD    CODE
I_CPU. . . . . .      L NEAR  DD62    CODE
I_CPU_ERR. . . . . .  L NEAR  DD8D    CODE
I_CPU_M. . . . . .    L BYTE  DC4B    CODE
I_CPU_OK . . . . . .  L NEAR  DD9E    CODE
I_DMAC . . . . . .    L NEAR  DDB2    CODE
I_DMAC_ERR . . . . .  L NEAR  DE5E    CODE
I_DMAC_LP. . . . . .  L NEAR  DDF2    CODE
I_DMAC_M . . . . . .  L BYTE  DC72    CODE
I_DMAC_NIB . . . . .  L NEAR  DE3E    CODE
I_DMAC_OK. . . . . .  L NEAR  DE64    CODE
I_DMAC_PASS2 . . . .  L NEAR  DDEF    CODE
I_DMAC_RET . . . . .  L NEAR  DE51    CODE
I_DMAT . . . . . .    L NEAR  DDC7    CODE
I_DMAT_ERR . . . . .  L NEAR  DDD5    CODE
I_DMAT_M . . . . . .  L BYTE  DC65    CODE
I_DMAT_OK. . . . . .  L NEAR  DDDB    CODE
I_DMAT_RET . . . . .  L NEAR  DDD3    CODE
I_D_80X25. . . . . .  L NEAR  E187    CODE
I_D_INIT . . . . . .  N PROC  E148    CODE    Length =0058
I_D_M. . . . . .      L BYTE  DC8C    CODE
I_D_MODE . . . . . .  L NEAR  E19A    CODE
I_D_OK . . . . . .    L NEAR  E18D    CODE
I_FATAL. . . . . .    N PROC  E0E9    CODE    Length =005F
I_FATAL_RET. . . . .  L NEAR  E139    CODE
I_FDUA_M . . . . . .  L BYTE  DD01    CODE
I_FDUB_M . . . . . .  L BYTE  DD0E    CODE
I_FDU_END. . . . . .  L NEAR  E4CD    CODE
I_FDU_LP . . . . . .  L NEAR  E4A7    CODE
I_FDU_NOT_M. . . . .  L BYTE  DD1B    CODE
I_FDU_OK . . . . . .  L NEAR  E4CA    CODE
I_FDU_RDY_M. . . . .  L BYTE  DD1F    CODE
I_HDU_M. . . . . .    L BYTE  DD28    CODE
I_HDU_OK . . . . . .  L NEAR  E47D    CODE
I_INIT_END . . . . .  L NEAR  E56C    CODE
I_KB_M . . . . . .    L BYTE  DCBF    CODE
I_KB_ST_M. . . . . .  L BYTE  DCCC    CODE
I_NO_COM_A . . . . .  L NEAR  E3AD    CODE
I_NO_COM_B . . . . .  L NEAR  E3BC    CODE
I_NO_GAME_CARD . . .  L NEAR  E3D5    CODE
I_NO_SCCS. . . . . .  L NEAR  E3CA    CODE
I_NPU_M. . . . . .    L BYTE  DC99    CODE
I_OPTROM_M . . . . .  L BYTE  DCF4    CODE
I_OUT_MASK . . . . .  N PROC  E1ED    CODE    Length =0009
I_PIC. . . . . .      L NEAR  DE6B    CODE
I_PIC_0_OK . . . . .  L NEAR  DEA9    CODE
I_PIC_1_OK . . . . .  L NEAR  DEAD    CODE
I_PIC_2_OK . . . . .  L NEAR  DEB1    CODE
I_PIC_3_OK . . . . .  L NEAR  DEB5    CODE
I_PIC_4_OK . . . . .  L NEAR  DEB9    CODE
I_PIC_END. . . . . .  L NEAR  DF11    CODE
I_PIC_ERR. . . . . .  L NEAR  DEDC    CODE
I_PIC_HARD . . . . .  L NEAR  DE9E    CODE
I_PIC_HOT. . . . . .  L NEAR  DED7    CODE
I_PIC_INIT . . . . .  N PROC  E1DC    CODE    Length =0011
I_PIC_M. . . . . .    L BYTE  DC7F    CODE
```

```
I_PIC_NO_HOT . . . . . L NEAR  DF05  CODE
I_PIC_OK . . . . . L NEAR  DF0B  CODE
I_PIC_SOFT . . . . . L NEAR  DE8E  CODE
I_PIC_TEST . . . . . L NEAR  DEBB  CODE
I_PRT_EXIT . . . . . L NEAR  E39B  CODE
I_PRT_LOOP . . . . . L NEAR  E382  CODE
I_PRT_M. . . . . L BYTE  DCD0  CODE
I_RAM_M. . . . . L BYTE  DCEA  CODE
I_ROM. . . . . L NEAR  DDB2  CODE
I_ROM_ERR. . . . . L NEAR  DDBA  CODE
I_ROM_M. . . . . L BYTE  DC58  CODE
I_ROM_OK . . . . . L NEAR  DDC0  CODE
I_ROM_RET. . . . . L NEAR  DDB8  CODE
I_RTC. . . . . L NEAR  E07B  CODE
I_RTC_END. . . . . L NEAR  E0E1  CODE
I_RTC_ERR. . . . . L NEAR  E0B9  CODE
I_RTC_HI_M . . . . . L BYTE  DCB7  CODE
I_RTC_LO_M . . . . . L BYTE  DCB3  CODE
I_RTC_M. . . . . L BYTE  DCA6  CODE
I_RTC_NR_M . . . . . L BYTE  DCBB  CODE
I_RTC_OK . . . . . L NEAR  E0D5  CODE
I_VEC0 . . . . . L NEAR  E1B0  CODE
I_VEC8 . . . . . L NEAR  E1CC  CODE
I_VECTOR . . . . . N PROC  E1A0  CODE    Length =003C
I_VEC_TBL. . . . . N PROC  FEF3  CODE    Length =0030
KBALT. . . . . Number  00C4
KBBRK. . . . . Number  00C9
KBCAP. . . . . Number  00C1
KBCTL. . . . . Number  00C5
KBINS. . . . . Number  00C0
KBLSH. . . . . Number  00C6
KBNUL. . . . . Number  00CC
KBNUM. . . . . Number  00C2
KBPRT. . . . . Number  00CB
KBRES. . . . . Number  00CA
KBRSH. . . . . Number  00C7
KBSCR. . . . . Number  00C3
KB_BUFFER. . . . . L WORD  001E  DATA    Length =0010
KB_CAP_FLAGS . . . . . L BYTE  CC60  CODE
KB_CMD_SEND. . . . . N PROC  E581  CODE    Length =000B
KB_CMD_WLUP. . . . . L NEAR  E581  CODE
KB_DATA1 . . . . . N PROC  CC60  CODE    Length =033F
KB_DATA_TABLE. . . . . L BYTE  CC67  CODE
KB_FLAG. . . . . L BYTE  0017  DATA
KB_FLAG_1. . . . . L BYTE  0018  DATA
KB_FLUSH . . . . . L NEAR  E304  CODE
KB_FLUSH_BACK. . . . . L NEAR  E313  CODE
KB_NOT_DLX . . . . . L NEAR  E361  CODE
KB_STATUS. . . . . Number  0064
KB_TYPE_READ . . . . . L NEAR  E351  CODE
KB_TYPE_WAIT . . . . . L NEAR  E346  CODE
KDBLO. . . . . Number  00D8
KDEC0. . . . . Number  00D7
KDEC1. . . . . Number  00D6
KDEC2. . . . . Number  00D5
KDEC3. . . . . Number  00D4
KDEC4. . . . . Number  00D3
KDEC5. . . . . Number  00D2
KDEC6. . . . . Number  00D1
KDEC7. . . . . Number  00D0
KDEC8. . . . . Number  00CF
KDEC9. . . . . Number  00CE
KNONE. . . . . Number  00CD
K_00 . . . . . L NEAR  EB35  CODE
K_2RES . . . . . L NEAR  EB18  CODE
K_2RET . . . . . L NEAR  EB0A  CODE
K_2TOG . . . . . L NEAR  EB0F  CODE
K_4RES . . . . . L NEAR  EAE7  CODE
K_4RET . . . . . L NEAR  EAE6  CODE
K_4TOG . . . . . L NEAR  EAD4  CODE
K_ADV_END. . . . . L NEAR  EB7A  CODE
K_ADV_PTR. . . . . N PROC  E86E  CODE    Length =000D
K_ALT. . . . . L NEAR  EABE  CODE
K_ALT0 . . . . . L NEAR  EB28  CODE
K_ALT1 . . . . . L NEAR  EB27  CODE
K_ALT2 . . . . . L NEAR  EB26  CODE
K_ALT3 . . . . . L NEAR  EB25  CODE
K_ALT4 . . . . . L NEAR  EB24  CODE
K_ALT5 . . . . . L NEAR  EB23  CODE
K_ALT6 . . . . . L NEAR  EB22  CODE
K_ALT7 . . . . . L NEAR  EB21  CODE
K_ALT8 . . . . . L NEAR  EB20  CODE
K_ALT9 . . . . . L NEAR  EB1F  CODE
K_BEEP . . . . . N PROC  EBC3  CODE    Length =0023
K_BIT. . . . . N PROC  EB85  CODE    Length =001F
K_BRK. . . . . L NEAR  EBA2  CODE
K_BUF. . . . . L NEAR  EA56  CODE
K_CAP. . . . . L NEAR  EAB5  CODE
K_CASE . . . . . L WORD  EBE6  CODE
K_CTL. . . . . L NEAR  EB00  CODE
K_DATA1. . . . . N PROC  EBE6  CODE    Length =0032
K_EOI. . . . . N PROC  EBA4  CODE    Length =0009
K_HOLD . . . . . L NEAR  EA92  CODE
K_INS. . . . . L NEAR  EAA7  CODE
K_INT. . . . . N PROC  E987  CODE    Length =01D2
K_IO . . . . . N PROC  E82E  CODE    Length =0017
K_IX . . . . . L NEAR  EA02  CODE
K_JMP. . . . . L NEAR  EA2D  CODE
K_LED_CAP. . . . . L NEAR  EB63  CODE
K_LED_CMD. . . . . L NEAR  EB70  CODE
```

```
K_LED_DAT. . . . . .    L NEAR  EB7A    CODE
K_LED_NUM. . . . . .    N PROC  EB59    CODE    Length =002C
K_LED_RET. . . . . .    L NEAR  EB84    CODE
K_LOCK . . . . . .      L NEAR  E9F4    CODE
K_LOOK . . . . . .      F PROC  E860    CODE    Length =0009
K_LF . . . . . .        L NEAR  EBCE    CODE
K_LSH. . . . . .        L NEAR  EB04    CODE
K_NON1 . . . . . .      L NEAR  EABF    CODE
K_NONE . . . . . .      L NEAR  EA5B    CODE
K_NOP. . . . . .        L NEAR  EA5E    CODE
K_NOP1 . . . . . .      L NEAR  EB3F    CODE
K_NO_CAP . . . . . .    L NEAR  E9E7    CODE
K_NO_CASE. . . . . .    L NEAR  EA36    CODE
K_NO_HOLD. . . . . .    L NEAR  EA4C    CODE
K_NO_LOCK. . . . . .    L NEAR  E9FC    CODE
K_NO_XCODE . . . . .    L NEAR  EA54    CODE
K_NUL. . . . . .        L NEAR  EAA2    CODE
K_NUM. . . . . .        L NEAR  EAC1    CODE
K_OR . . . . . .        L NEAR  E9B2    CODE
K_PAUSE. . . . . .      L NEAR  EA78    CODE
K_PRT. . . . . .        L NEAR  EA9B    CODE
K_READ . . . . . .      N PROC  E845    CODE    Length =001B
K_RES. . . . . .        L NEAR  EA67    CODE
K_RET. . . . . .        L NEAR  E842    CODE
K_RSH. . . . . .        L NEAR  EB08    CODE
K_SCR. . . . . .        L NEAR  EACD    CODE
K_SEE. . . . . .        L NEAR  E854    CODE
K_STAT . . . . . .      N PROC  E869    CODE    Length =0005
K_TRY. . . . . .        N PROC  EBAD    CODE    Length =0016
K_XLAT . . . . . .      L NEAR  EA16    CODE
LEFT_SHIFT . . . . . .  Number  0002
LEFT_SHIFT_KEY . . . .  Number  002A
LP . . . . . .          Number  000A
MASK_PORT. . . . . .    Number  0323
MASK_PORT_DMAE . . . .  Number  0001
MASK_PORT_INTE . . . .  Number  0002
MASTAB . . . . . .      L WORD  E2CE    CODE
MASTER_TBL_PTR . . . .  L DWORD 0084    DATA
MAX_INT. . . . . .      Number  0014
MAYBE_SHOW_SENSE . . .  N PROC  D0CE    CODE    Length =001B
MEMORY_SIZE. . . . . .  L WORD  0013    DATA
MEMTST . . . . . .      N PROC  E266    CODE    Length =0047
MEMTST_ERR . . . . . .  L NEAR  E2AB    CODE
MEMTST_ERR_C . . . . .  L NEAR  E2A7    CODE
MEMTST_R1. . . . . .    L NEAR  E279    CODE
MEMTST_R2. . . . . .    L NEAR  E290    CODE
MEMTST_W1. . . . . .    L NEAR  E26D    CODE
MEMTST_W2. . . . . .    L NEAR  E286    CODE
MFG_ERR_FLAG . . . . .  L BYTE  0015    DATA    Length =0002
MFG_TST. . . . . .      L BYTE  0012    DATA
MOTOR_COUNT. . . . . .  L BYTE  0040    DATA
MOTOR_STATUS . . . . .  L BYTE  003F    DATA
MT_END . . . . . .      L WORD  E2E4    CODE
M_CASS . . . . . .      N PROC  F859    CODE    Length =0006
M_EQUIP. . . . . .      N PROC  F84D    CODE    Length =000C
M_EXIT . . . . . .      L NEAR  D0E7    CODE
M_SIZE . . . . . .      N PROC  F841    CODE    Length =000C
NEC_STATUS . . . . . .  L BYTE  0042    DATA    Length =0007
NIBOK. . . . . .        L NEAR  E65E    CODE
NO_HARD_DISK . . . . .  L NEAR  D697    CODE
NUL. . . . . .          Number  0000
NUM_LOCK_KEY . . . . .  Number  0045
NUM_LOCK_MODE. . . . .  Number  0020
NUM_LOCK_SHIFT . . . .  Number  0020
N_INT. . . . . .        N PROC  F85F    CODE    Length =0001
OPT_ROM_M. . . . . .    L BYTE  E2AD    CODE
PD_DATA1 . . . . . .    N PROC  E2AD    CODE    Length =003F
P1_DATA1 . . . . . .    N PROC  DB7D    CODE    Length =01DD
P4_DATA1 . . . . . .    N PROC  E5F2    CODE    Length =0002
PARA_GRAPH . . . . . .  Number  B800
PARA_MONO. . . . . .    Number  B000
PASS_M . . . . . .      L BYTE  DC3D    CODE
PAUSE. . . . . .        Number  00CA
PAUSE_MODE . . . . . .  Number  0008
PCINIT . . . . . .      N PROC  E2E4    CODE    Length =029D
PIC_0. . . . . .        Number  0020
PIC_1. . . . . .        Number  0021
PIC_ICW1 . . . . . .    Number  0013
PIC_ICW2 . . . . . .    Number  0008
PIC_ICW3 . . . . . .    Number  0008
PIC_ICW4 . . . . . .    Number  000D
PIC_NEOI . . . . . .    Number  0020
PIC_OFF_MSK. . . . . .  Number  00FF
PIC_SEOI_0 . . . . . .  Number  0060
PIC_SEOI_1 . . . . . .  Number  0061
PIC_SEOI_6 . . . . . .  Number  0066
PORT_OFF . . . . . .    L BYTE  0077    DATA
POST_LOSE. . . . . .    L NEAR  D071    CODE
POST_NO_MORE_DRIVES. .  L NEAR  D06A    CODE
POST_WIN . . . . . .    L NEAR  D083    CODE
PRINTER_ADDR . . . . .  L WORD  0008    DATA    Length =0004
PRINTER_T_OUT. . . . .  L BYTE  0078    DATA    Length =0004
PRT_DATA_A . . . . . .  Number  03BC
PRT_DATA_B . . . . . .  Number  0378
PRT_DATA_C . . . . . .  Number  0278
P_8253_0 . . . . . .    Number  0040
P_8253_1 . . . . . .    Number  0041
P_8253_2 . . . . . .    Number  0042
P_8253_CTRL. . . . . .  Number  0043
P_DISP . . . . . .      L NEAR  D09B    CODE
```

```
P_INIT . . . . . L NEAR  F027   CODE
P_IO . . . . . N PROC  EFD2   CODE    Length =0069
P_KCTRL. . . . . Number  0061
P_KSCAN. . . . . Number  0060
P_LP . . . . . L NEAR  F00F   CODE
P_NOP. . . . . L NEAR  F00A   CODE
P_OK . . . . . L NEAR  F01D   CODE
P_OUT. . . . . L NEAR  F00B   CODE
P_RET. . . . . L NEAR  F005   CODE
P_SOME . . . . . L NEAR  D092   CODE
P_STAT . . . . . L NEAR  F034   CODE
P_TBL. . . . . L WORD  E2BE   CODE
RAMDIAG_BX . . . . . Number  00E0
RAMDIAG_INT. . . . . Number  0012
RAM_ERROR. . . . . L NEAR  DFC9   CODE
RAM_SIZE_END . . . . . L NEAR  DFAB   CODE
RAM_SIZE_END_1 . . . . . L NEAR  DFB5   CODE
RAM_SIZE_LP. . . . . L NEAR  DF6C   CODE
RAM_SIZE_NXT . . . . . L NEAR  DFA3   CODE
RAM_SIZE_TST . . . . . L NEAR  DF5C   CODE
RBUFF_BX . . . . . Number  000E
READ_BX. . . . . Number  0008
READ_ECC_BX. . . . . Number  000D
READ_INT . . . . . Number  0002
READ_MODE. . . . . Number  0047
READ_PARAM . . . . . N PROC  DADF   CODE    Length =0025
READ_PORT. . . . . Number  0320
RECAL_BX . . . . . Number  0001
RECAL_INT. . . . . Number  0011
RESET_BX . . . . . N PROC  D5B1   CODE    Length =0010
RESET_FLAG . . . . . L WORD  0072   DATA
RESET_INT. . . . . Number  0000
RESET_PORT . . . . . Number  0321
RET_FROM_NO. . . . . L NEAR  D1B2   CODE
RIGHT_SHIFT. . . . . Number  0001
RIGHT_SHIFT_KEY. . . . . Number  0036
RLONG_BX . . . . . Number  00E5
ROM_CHECKSUM . . . . . N PROC  E5E5   CODE    Length =000D
ROM_CHECKSUM_CNT . . . . . L NEAR  E5E8   CODE
ROM_CHECKSUM_LOOP. . . . . L NEAR  E5EA   CODE
ROM_CHKSUM_OK. . . . . L NEAR  E435   CODE
ROM_ERR. . . . . N PROC  E5D5   CODE    Length =0010
ROM_ID . . . . . L BYTE  C001   CODE
ROM_MT . . . . . L WORD  C002   CODE
ROM_SCAN_EXIT. . . . . L NEAR  E44E   CODE
ROM_SCAN_LOOP. . . . . L NEAR  E3FD   CODE
ROM_SCAN_NEXT. . . . . L NEAR  E44A   CODE
RS232_ADDR . . . . . L WORD  0000   DATA    Length =0008
RS_DLY . . . . . N PROC  E8E2   CODE    Length =000A
RS_GBE . . . . . L NEAR  E948   CODE
RS_INIT. . . . . L NEAR  E3F0   CODE
RS_LP. . . . . L NEAR  E8E7   CODE
RS_NOP . . . . . L NEAR  E77D   CODE
RS_NORM. . . . . L NEAR  E749   CODE
RS_OK. . . . . L NEAR  E771   CODE
RS_PBE . . . . . L NEAR  E910   CODE
RS_PB_GB . . . . . L NEAR  E90F   CODE
RS_RET . . . . . L NEAR  E77A   CODE
RS_STAT. . . . . N PROC  E87B   CODE    Length =0043
RS_TBL . . . . . L WORD  E77F   CODE
RS_WS. . . . . N PROC  E8BE   CODE    Length =0024
RS_WS_COM. . . . . L NEAR  E8CD   CODE
RS_WS_EXIT . . . . . L NEAR  E8DE   CODE
RS_WS_LP . . . . . L NEAR  E8C3   CODE
RTC_CHK. . . . . N PROC  E1F6   CODE    Length =0070
RTC_CHK_HIGH . . . . . L NEAR  E265   CODE
RTC_CHK_LOW. . . . . L NEAR  E265   CODE
RTC_CHK_RESET_ERR. . . . . L NEAR  E217   CODE
RTC_CHK_RESET_LP . . . . . L NEAR  E205   CODE
RTC_CHK_RESET_OK . . . . . L NEAR  E21A   CODE
RTC_CHK_SET_ERR. . . . . L NEAR  E238   CODE
RTC_CHK_SET_LP . . . . . L NEAR  E224   CODE
RTC_CHK_SET_OK . . . . . L NEAR  E23B   CODE
R_FINISH . . . . . L NEAR  D435   CODE
R_WAITING. . . . . L NEAR  D41D   CODE
SCC_CTL_A. . . . . Number  0050
SCC_CTL_B. . . . . Number  0052
SCC_DBIT . . . . . L BYTE  F62E   CODE
SCC_FE . . . . . Number  0040
SCC_GB . . . . . N PROC  E94D   CODE    Length =000D
SCC_INIT . . . . . N PROC  F573   CODE    Length =00BF
SCC_NO_FE. . . . . L NEAR  E8AD   CODE
SCC_NO_OE. . . . . L NEAR  E8BB   CODE
SCC_NO_PE. . . . . L NEAR  E8B9   CODE
SCC_NO_RXD . . . . . L NEAR  E89F   CODE
SCC_NO_TXD . . . . . L NEAR  E898   CODE
SCC_OE . . . . . Number  0020
SCC_PB . . . . . N PROC  E91B   CODE    Length =000F
SCC_PE . . . . . Number  0010
SCC_PWRUP. . . . . L NEAR  F623   CODE
SCC_RXD. . . . . Number  0001
SCC_STAT . . . . . L NEAR  E88A   CODE
SCC_TBL. . . . . L WORD  E2C6   CODE
SCC_TXD. . . . . Number  0004
SCRL_LOCK_KEY. . . . . Number  0046
SCRL_LOCK_MODE . . . . . Number  0010
SCRL_LOCK_SHIFT. . . . . Number  0010
SECTORS_PER_TRACK. . . . . Number  0011
SECT_NOT_FOUND . . . . . Number  0004
SEEK_BX. . . . . Number  000B
```

```
SEEK_ERROR . . . . . .  Number  0040
SEEK_STATUS. . . . . .  L BYTE  003E    DATA
SELECT_PORT. . . . . .  Number  0322
SEND_CDB_BYTE. . . . .  L NEAR  D3F8    CODE
SENSE0_MSK . . . . . .  Number  003F
SENSE_BX . . . . . .  Number  0003
SENSE_ERR. . . . . .  Number  00FF
SERIAL_IO. . . . . .  N PROC  E739    CODE    Length =0056
SERIAL_T_OUT . . . . .  L BYTE  007C    DATA    Length =0004
SET_DS . . . . . .  N PROC  E5F4    CODE    Length =0006
SET_DS_WORD. . . . . .  L WORD  E5F2    CODE
SHOW_SENSE . . . . . .  N PROC  D0BB    CODE    Length =0013
STACK_ROM. . . . . .  L WORD  DB7E    CODE
STACK_SEG. . . . . .  Number  0030
STATUS . . . . . .  L NEAR  D263    CODE
STATUS_INT . . . . . .  Number  0001
STATUS_PORT. . . . . .  Number  0321
STOP_DISK. . . . . .  N PROC  EF4F    CODE    Length =0007
SWITCH_BITS. . . . . .  L WORD  0010    DATA
SWITCH_PORT. . . . . .  Number  0322
SYS_CONF_A . . . . . .  Number  0066
SYS_CONF_B . . . . . .  Number  0067
S_EOL. . . . . .  L NEAR  FFB5    CODE
S_ERR. . . . . .  L NEAR  FFA5    CODE
S_GET. . . . . .  L NEAR  FFC9    CODE
S_INT. . . . . .  N PROC  FF54    CODE    Length =007E
S_LP . . . . . .  L NEAR  FF81    CODE
S_NOP. . . . . .  L NEAR  FFB2    CODE
S_OK . . . . . .  L NEAR  FF8A    CODE
S_OUT. . . . . .  L NEAR  FFBC    CODE
T0CMD. . . . . .  Number  0036
T0COUNT. . . . . .  Number  0000
T1CMD. . . . . .  Number  0074
T1COUNT. . . . . .  Number  0013
T2CMD. . . . . .  Number  00B6
T2COUNT. . . . . .  Number  0266
TIME_OUT . . . . . .  Number  0080
TST_DRV_RDY_BX . . . . .  Number  0000
TST_DRV_RDY_INT. . . . .  Number  0010
T_DAY. . . . . .  N PROC  FE6E    CODE    Length =0037
T_END. . . . . .  L NEAR  FEA3    CODE
T_HI . . . . . .  L NEAR  FEC5    CODE
T_HI_ORDER . . . . . .  L WORD  006E    DATA
T_INC. . . . . .  L NEAR  FEBB    CODE
T_INT. . . . . .  N PROC  FE45    CODE    Length =0047
T_LOW_ORDER. . . . . .  L WORD  006C    DATA
T_NFE. . . . . .  L NEAR  FE78    CODE
T_NFF. . . . . .  L NEAR  FE7E    CODE
T_OFL. . . . . .  L NEAR  FEE1    CODE
T_OVERFLOW . . . . . .  L BYTE  0070    DATA
T_SET. . . . . .  L NEAR  FE95    CODE
VECTOR . . . . . .  N PROC  FFF0    CODE    Length =000E
V_3X8. . . . . .  L BYTE  0065    DATA
V_6845 . . . . . .  L NEAR  F262    CODE
V_APAGE. . . . . .  L BYTE  0062    DATA
V_BASE6845 . . . . . .  L WORD  0063    DATA
V_BELL . . . . . .  N PROC  F54E    CODE    Length =0025
V_BS . . . . . .  L NEAR  F4C3    CODE
V_CLR. . . . . .  L NEAR  F2F5    CODE
V_CLR_BOT. . . . . .  L NEAR  F2F5    CODE
V_CLR_FAST . . . . . .  L NEAR  F33F    CODE
V_CLR_FLP. . . . . .  L NEAR  F341    CODE
V_CLR_HI . . . . . .  L NEAR  F30A    CODE
V_CLR_LO . . . . . .  L NEAR  F310    CODE
V_CLR_LP . . . . . .  L NEAR  F300    CODE
V_CLR_NXT. . . . . .  L NEAR  F325    CODE
V_CLR_PAST . . . . . .  L NEAR  F33C    CODE
V_CLR_ROW. . . . . .  L NEAR  F32E    CODE
V_CLR_TOP. . . . . .  L NEAR  F377    CODE
V_COL. . . . . .  N PROC  F41A    CODE    Length =002C
V_COLORPAL . . . . . .  L BYTE  0066    DATA
V_COLOUR . . . . . .  L NEAR  F08E    CODE
V_COL_0. . . . . .  L NEAR  F42F    CODE
V_COL_1. . . . . .  L NEAR  F436    CODE
V_CR . . . . . .  L NEAR  F4CB    CODE
V_CURPOS . . . . . .  L WORD  0050    DATA    Length =0008
V_CURSIZE. . . . . .  L WORD  0060    DATA
V_CURS_POS . . . . . .  N PROC  F1F7    CODE    Length =001E
V_CURS_TYPE. . . . . .  N PROC  F1E9    CODE    Length =000E
V_DATA1. . . . . .  N PROC  F045    CODE    Length =0020
V_DATA2. . . . . .  N PROC  F0A8    CODE    Length =0058
V_DISALLOW . . . . . .  L NEAR  D626    CODE
V_FPOS . . . . . .  N PROC  F50D    CODE    Length =001E
V_FPOS_0 . . . . . .  L NEAR  F51F    CODE
V_FPOS_LP. . . . . .  L NEAR  F518    CODE
V_HEIGHT . . . . . .  L WORD  004C    DATA
V_IO . . . . . .  N PROC  F065    CODE    Length =003F
V_KCLEAR . . . . . .  Number  0007
V_KSCROLL. . . . . .  Number  0004
V_LF . . . . . .  L NEAR  F47B    CODE
V_LROW . . . . . .  L NEAR  F486    CODE
V_MD_40. . . . . .  L BYTE  F0A8    CODE
V_MD_80. . . . . .  L BYTE  F0B8    CODE
V_MD_CLR . . . . . .  L NEAR  F171    CODE
V_MD_CLR_2K. . . . . .  L NEAR  F16C    CODE
V_MD_CLR_8K. . . . . .  L NEAR  F16E    CODE
V_MD_CLR_GRAPHICS. . . . .  L NEAR  F168    CODE
V_MD_DBL . . . . . .  L NEAR  F188    CODE
V_MD_ENABLE. . . . . .  L BYTE  F0F4    CODE
V_MD_GRAPH . . . . . .  L BYTE  F0C4    CODE
```

```
V_MD_LEN . . . . . L WORD  F05A  CODE
V_MD_MONO. . . . . L BYTE  F0DA  CODE
V_MD_WID . . . . . L BYTE  F0EC  CODE
V_MODE . . . . . L BYTE  0049  DATA
V_MV . . . . . L NEAR  F295  CODE
V_MV_DN. . . . . L NEAR  F370  CODE
V_MV_END . . . . . L NEAR  F2FA  CODE
V_MV_FLP . . . . . L NEAR  F2E7  CODE
V_MV_HI. . . . . L NEAR  F2AA  CODE
V_MV_LO. . . . . L NEAR  F2B0  CODE
V_MV_LP. . . . . L NEAR  F2A0  CODE
V_MV_NXT . . . . . L NEAR  F2CA  CODE
V_MV_PAST. . . . . L NEAR  F2E5  CODE
V_MV_ROW . . . . . L NEAR  F2D1  CODE
V_NOP. . . . . L NEAR  F0A3  CODE
V_OUT_BYTE . . . . . N PROC  F273  CODE    Length =000C
V_OVR_NOT_OK . . . . . L NEAR  F1C8  CODE
V_OVR_OK . . . . . L NEAR  F1D6  CODE
V_PAGE . . . . . N PROC  F22C  CODE    Length =0047
V_PAGE_O . . . . . L NEAR  F23E  CODE
V_PARMS. . . . . L BYTE  F0A8  CODE
V_POINTER. . . . . Number 03B4
V_POSN . . . . . N PROC  F52B  CODE    Length =0011
V_RAC. . . . . N PROC  F37C  CODE    Length =002A
V_RAC_INBLANK. . . . . L NEAR  F398  CODE
V_RAC_INLINE . . . . . L NEAR  F392  CODE
V_RESET_BX . . . . . L NEAR  D616  CODE
V_R_CURS_POS . . . . . N PROC  F215  CODE    Length =0017
V_SCRL_DN. . . . . N PROC  F35B  CODE    Length =0021
V_SCRL_MODE_7. . . . . L NEAR  F356  CODE
V_SCRL_MV_AND_CLR. . . . . L NEAR  F50C  CODE
V_SCRL_POS . . . . . N PROC  F4DE  CODE    Length =002F
V_SCRL_RET . . . . . L NEAR  F34B  CODE
V_SCRL_TTY . . . . . L NEAR  F491  CODE
V_SCRL_TTY_GRAPHICS. . . . . L NEAR  F4A0  CODE
V_SCRL_UP. . . . . N PROC  F27F  CODE    Length =00DC
V_SET_CURS . . . . . L NEAR  F210  CODE
V_SET_CUR_POS. . . . . L NEAR  F256  CODE
V_SET_MODE . . . . . N PROC  F0FC  CODE    Length =00ED
V_SET_MODE_COLOR . . . . . L NEAR  F117  CODE
V_SET_MODE_LP. . . . . L NEAR  F147  CODE
V_SET_NEW_CUR. . . . . L NEAR  F48C  CODE
V_STAT . . . . . N PROC  F4CF  CODE    Length =000F
V_TBL. . . . . L WORD  F045  CODE
V_TERMINAL . . . . . N PROC  F446  CODE    Length =0089
V_TERM_NOBELL. . . . . L NEAR  F44D  CODE
V_TERM_NOP . . . . . L NEAR  F4BB  CODE
V_TERM_RET . . . . . L NEAR  F4B9  CODE
V_TOP. . . . . L WORD  004E  DATA
V_TXT_DN . . . . . L NEAR  F364  CODE
V_TXT_MD . . . . . N PROC  F53C  CODE    Length =0012
V_TXT_OK . . . . . L NEAR  F5AC  CODE
V_TXT_RAC. . . . . L NEAR  F384  CODE
V_TXT_UP . . . . . L NEAR  F287  CODE
V_TXT_WAC. . . . . L NEAR  F3AE  CODE
V_TXT_WC . . . . . L NEAR  F3E7  CODE
V_WAC. . . . . N PROC  F3A6  CODE    Length =0039
V_WAC_END. . . . . L NEAR  F3DA  CODE
V_WAC_HI . . . . . L NEAR  F3C5  CODE
V_WAC_LO . . . . . L NEAR  F3CB  CODE
V_WC . . . . . N PROC  F3DF  CODE    Length =003B
V_WC_END . . . . . L NEAR  F415  CODE
V_WC_HI. . . . . L NEAR  F3FE  CODE
V_WC_LO. . . . . L NEAR  F404  CODE
V_WC_NEXT. . . . . L NEAR  F3FE  CODE
V_WIDTH. . . . . L WORD  004A  DATA
W1 . . . . . L NEAR  D460  CODE
W2 . . . . . L NEAR  D46B  CODE
WAIT_FOR_IRQ5. . . . . L NEAR  D46D  CODE
WAIT_FOR_STATUS. . . . . N PROC  D418  CODE    Length =0023
WBUFF_BX . . . . . Number 000F
WBUFF_INT. . . . . Number 000F
WLONG_BX . . . . . Number 00E6
WRITE_BX . . . . . Number 000A
WRITE_MODE . . . . . Number 004B
WRITE_PORT . . . . . Number 0320
WRITE_PROTECT. . . . . Number 0003
W_FINISH . . . . . L NEAR  D498  CODE
W_ITS_DONE . . . . . L NEAR  D486  CODE
W_NEC. . . . . L NEAR  EF92  CODE
W_NEC_RET. . . . . L NEAR  EFA6  CODE
W_ONE. . . . . L NEAR  EF4B  CODE

Warning Severe
Errors  Errors
0       0
```

ROM BIOS Change List

ROM Rev 1.1 August 10, 1984
Aug 8 16:40 1984 fdu1.src:
Fix random value in DX during reset causing problem

```
Line 73   Add
  push    es
  cmp     ah, 0           ;Is it a reset command?
  jz      diskette_i01    ;Yes, so ignore drive
                          ;param.

Line 123  Add
  pop     es
```

Aug 8 16:40 1984 fdu1.src:
Allow verify operation without valid buffer pointer (buffer not required)

```
  xor     bx,bx           ;Fool the DMAC into
  mov     es,bx           ;thinking there is a full
                          ;segment to play with.
```

Do not range check head number

```
Change
  and     ah,1            ;blow off high 7 bits.
to
  and     ah,07Fh         ;blow off bit 7.
```

Aug 8 16:41 1984 flags.src:
Change date of release in ROM

```
  db      '05/03/84'      ;release marker (exactly
                          ;8 bytes!!!!)
to
  db      '08/10/84'      ;release marker (exactly
                          ;8 bytes!!!!)
```

Aug 8 16:43 1984 fontl08.src:
Change 8 X 8 font to improve italics as displayed by 3rd party software

Changed

```
        DB   00h,00h,00h,00h
        DB   00h,00h,00h,00h        ;   0
        DB   07eh,081h,0a5h,081h
        DB   0bdh,099h,081h,07eh    ;   1
        DB   07eh,0ffh,0dbh,0ffh
        DB   0c3h,0e7h,0ffh,07eh    ;   2
        DB   036h,07fh,07fh,07fh
        DB   03eh,01ch,08h,00h      ;   3
        DB   08h,01ch,03eh,07fh
        DB   03eh,01ch,08h,00h      ;   4
        DB   018h,03ch,0e7h,0e7h
        DB   066h,018h,018h,03ch    ;   5
        DB   018h,03ch,07eh,0ffh
        DB   0ffh,07eh,018h,03ch    ;   6
        DB   00h,00h,018h,03ch
        DB   03ch,018h,00h,00h      ;   7
        DB   0ffh,0ffh,0e7h,0c3h
        DB   0c3h,0e7h,0ffh,0ffh    ;   8
        DB   00h,03ch,024h,042h
        DB   042h,024h,03ch,00h     ;   9
        DB   0ffh,0c3h,099h,0bdh
        DB   0bdh,099h,0c3h,0ffh    ;   a
        DB   01fh,07h,0dh,019h
        DB   078h,0cch,0cch,078h    ;   b
        DB   03ch,066h,066h,03ch
        DB   018h,07eh,018h,018h    ;   c
        DB   018h,014h,012h,012h
        DB   014h,070h,0f0h,060h    ;   d
        DB   01fh,011h,01fh,011h
        DB   013h,037h,072h,020h    ;   e
        DB   018h,0dbh,03ch,0e7h
        DB   03ch,0dbh,018h,00h     ;   f
        DB   040h,070h,07ch,07fh
        DB   07ch,070h,040h,00h     ;   10
        DB   01h,07h,01fh,07fh
        DB   01fh,07h,01h,00h       ;   11
        DB   018h,03ch,07eh,018h
        DB   018h,07eh,03ch,018h    ;   12
```

```
DB   00h,033h,033h,033h
DB   033h,00h,033h,00h          ;    13
DB   07fh,0dbh,0dbh,07bh
DB   01bh,01bh,01bh,00h         ;    14
DB   03eh,061h,03ch,066h
DB   066h,03ch,086h,07ch        ;    15
DB   00h,00h,00h,00h
DB   07eh,07eh,07eh,00h         ;    16
DB   018h,03ch,07eh,018h
DB   07eh,03ch,018h,07eh        ;    17
DB   018h,03ch,07eh,018h
DB   018h,018h,018h,018h        ;    18
DB   018h,018h,018h,018h
DB   018h,07eh,03ch,018h        ;    19
DB   0ch,06h,07fh,06h
DB   0ch,00h,00h,00h            ;    1a
DB   018h,030h,07fh,030h
DB   018h,00h,00h,00h           ;    1b
DB   00h,00h,060h,060h
DB   060h,07fh,07fh,00h         ;    1c
DB   00h,024h,066h,0ffh
DB   066h,024h,00h,00h          ;    1d
DB   00h,00h,018h,03ch
DB   07eh,0ffh,00h,00h          ;    1e
DB   00h,0ffh,07eh,03ch
DB   018h,00h,00h,00h           ;    1f
DB   00h,00h,00h,00h
DB   00h,00h,00h,00h            ;    ' '  20
DB   018h,03ch,03ch,018h
DB   018h,00h,018h,00h          ;    '!'  21
DB   036h,036h,014h,00h
DB   00h,00h,00h,00h            ;    '"'  22
DB   036h,036h,07fh,036h
DB   07fh,036h,036h,00h         ;    '#'  23
DB   018h,03eh,060h,03ch
DB   06h,07ch,018h,00h          ;    '$'  24
DB   00h,063h,066h,0ch
DB   018h,033h,033h,00h         ;    '%'  25
```

```
DB   01ch,036h,01ch,03bh
DB   06eh,066h,03bh,00h      ;    '&'  26
DB   018h,018h,030h,00h
DB   00h,00h,00h,00h         ;    ' '  27
DB   0ch,018h,030h,030h
DB   030h,030h,018h,0ch      ;    '('  28
DB   030h,018h,0ch,0ch
DB   0ch,0ch,018h,030h       ;    ')'  29
DB   00h,066h,03ch,0ffh
DB   03ch,066h,00h,00h       ;    '*'  2a
DB   00h,018h,018h,07eh
DB   018h,018h,00h,00h       ;    '+'  2b
DB   00h,00h,00h,00h
DB   00h,018h,018h,030h      ;    ','  2c
DB   00h,00h,00h,07eh
DB   00h,00h,00h,00h         ;    '—'  2d
DB   00h,00h,00h,00h
DB   00h,018h,018h,00h       ;    '-'  2e
DB   03h,06h,0ch,018h
DB   030h,060h,040h,00h      ;    '/'  2f
DB   03eh,063h,067h,06fh
DB   07bh,073h,03eh,00h      ;    '0'  30
DB   0ch,01ch,0ch,0ch
DB   0ch,0ch,03fh,00h        ;    '1'  31
DB   01fh,033h,03h,0eh
DB   018h,033h,03fh,00h      ;    '2'  32
DB   01eh,033h,03h,0eh
DB   03h,033h,01eh,00h       ;    '3'  33
DB   0eh,01eh,036h,066h
DB   07fh,06h,0fh,00h        ;    '4'  34
DB   03fh,030h,03eh,03h
DB   03h,033h,01eh,00h       ;    '5'  35
DB   0eh,018h,030h,03eh
DB   033h,033h,01eh,00h      ;    '6'  36
DB   03fh,033h,03h,06h
DB   0ch,0ch,0ch,00h         ;    '7'  37
DB   01eh,033h,033h,01eh
DB   033h,033h,01eh,00h      ;    '8'  38
```

```
DB   01eh,033h,031h,01fh
DB   03h,06h,01ch,00h          ;    '9'  39
DB   00h,018h,018h,00h
DB   00h,018h,018h,00h         ;    ':'  3a
DB   00h,018h,018h,00h
DB   00h,018h,018h,030h        ;    ';'  3b
DB   06h,0ch,018h,030h
DB   018h,0ch,06h,00h          ;    '<'  3c
DB   00h,00h,07eh,00h
DB   00h,07eh,00h,00h          ;    '='  3d
DB   030h,018h,0ch,06h
DB   0ch,018h,030h,00h         ;    '>'  3e
DB   03eh,063h,03h,06h
DB   0ch,00h,0ch,00h           ;    '?'  3f
DB   03eh,063h,06fh,06fh
DB   06eh,060h,03eh,00h        ;    '@'  40
DB   08h,01ch,036h,063h
DB   07fh,063h,063h,00h        ;    'A'  41
DB   07eh,033h,033h,03eh
DB   033h,033h,07eh,00h        ;    'B'  42
DB   01eh,033h,060h,060h
DB   061h,033h,01eh,00h        ;    'C'  43
DB   07ch,036h,033h,033h
DB   033h,036h,07ch,00h        ;    'D'  44
DB   07fh,031h,034h,03ch
DB   034h,031h,07fh,00h        ;    'E'  45
DB   07fh,031h,034h,03ch
DB   034h,030h,078h,00h        ;    'F'  46
DB   01eh,033h,060h,060h
DB   067h,033h,01dh,00h        ;    'G'  47
DB   063h,063h,063h,07fh
DB   063h,063h,063h,00h        ;    'H'  48
DB   03ch,018h,018h,018h
DB   018h,018h,03ch,00h        ;    'I'  49
DB   0fh,06h,06h,06h
DB   066h,066h,03ch,00h        ;    'J'  4a
DB   073h,036h,03ch,038h
DB   03ch,036h,073h,00h        ;    'K'  4b
```

```
DB   078h,030h,030h,030h
DB   030h,033h,07fh,00h      ;   'L'   4c
DB   063h,077h,07fh,06bh
DB   063h,063h,063h,00h      ;   'M'   4d
DB   063h,073h,07bh,06fh
DB   067h,063h,063h,00h      ;   'N'   4e
DB   01ch,036ch,063h,063h
DB   063h,036h,01ch,00h      ;   'O'   4f
DB   07eh,033h,033h,03eh
DB   030h,030h,078h,00h      ;   'P'   50
DB   01ch,036h,063h,063h
DB   063h,036h,01ch,07h      ;   'Q'   51
DB   07eh,033h,033h,03eh
DB   036h,033h,033h,00h      ;   'R'   52
DB   03eh,063h,030h,018h
DB   06h,063h,03eh,00h       ;   'S'   53
DB   07eh,05ah,018h,018h
DB   018h,018h,03ch,00h      ;   'T'   54
DB   063h,063h,063h,063h
DB   063h,063h,03eh,00h      ;   'U'   55
DB   063h,063h,063h,063h
DB   036h,01ch,08h,00h       ;   'V'   56
DB   063h,063h,063h,06bh
DB   06bh,07fh,036h,00h      ;   'W'   57
DB   063h,063h,036h,01ch
DB   036h,063h,063h,00h      ;   'X'   58
DB   066h,066h,066h,03ch
DB   018h,018h,03ch,00h      ;   'Y'   59
DB   07fn,063h,06h,0ch
DB   018h,033h,07fh,00h      ;   'Z'   5a
DB   03ch,030h,030h,030h
DB   030h,030h,03ch,00h      ;   '['   5b
DB   020h,030h,018h,0ch
DB   06h,03h,01h,00h         ;   '\'   5c
DB   03ch,0ch,0ch,0ch
DB   0ch,0ch,03ch,00h        ;   ']'   5d
DB   08h,01ch,036h,063h
DB   00h,00h,00h,00h         ;   ' '   5e
```

```
DB    00h,00h,00h,00h
DB    00h,00h,07fh,00h          ;    '—'  5f
DB    018h,018h,0ch,00h
DB    00h,00h,00h,00h           ;    ' '  60
DB    00h,00h,03ch,06h
DB    03eh,066h,03bh,00h        ;    'a'  61
DB    070h,030h,03eh,033h
DB    033h,033h,06eh,00h        ;    'b'  62
DB    00h,00h,03eh,061h
DB    060h,061h,03eh,00h        ;    'c'  63
DB    0eh,06h,03eh,066h
DB    066h,066h,03bh,00h        ;    'd'  64
DB    00h,00h,03eh,063h
DB    07fh,060h,03fh,00h        ;    'e'  65
DB    01eh,033h,030h,07ch
DB    030h,030h,078h,00h        ;    'f'  66
DB    00h,00h,03bh,066h
DB    066h,03eh,046h,03ch       ;    'g'  67
DB    070h,030h,036h,03fh
DB    033h,033h,073h,00h        ;    'h'  68
DB    0ch,00h,01ch,0ch
DB    0ch,0ch,01eh,00h          ;    'i'  69
DB    0ch,00h,01eh,0ch
DB    0ch,0cch,0cch,078h        ;    'j'  6a
DB    070h,030h,033h,036h
DB    03ch,036h,073h,00h        ;    'k'  6b
DB    01ch,0ch,0ch,0ch
DB    0ch,0ch,01eh,00h          ;    'l'  6c
DB    00h,00h,066h,07fh
DB    06bh,06bh,06bh,00h        ;    'm'  6d
DB    00h,00h,06eh,033h
DB    033h,033h,033h,00h        ;    'n'  6e
DB    00h,00h,01eh,033h
DB    033h,033h,01eh,00h        ;    'o'  6f
DB    00h,00h,06eh,033h
DB    033h,03eh,030h,078h       ;    'p'  70
DB    00h,00h,03bh,066h
DB    066h,03eh,06h,0fh         ;    'q'  71
DB    00h,00h,06eh,03bh
```

```
        DB   030h,030h,078h,00h        ;    'r'  72
        DB   00h,00h,03fh,060h
        DB   03ch,03h,07eh,00h         ;    's'  73
        DB   08h,018h,03eh,018h
        DB   018h,01bh,0eh,00h         ;    't'  74
        DB   00h,00h,066h,066h
        DB   066h,066h,03bh,00h        ;    'u'  75
        DB   00h,00h,063h,063h
        DB   036h,01ch,08h,00h         ;    'v'  76
        DB   00h,00h,063h,06bh
        DB   06bh,07fh,036h,00h        ;    'w'  77
        DB   00h,00h,063h,036h
        DB   01ch,036h,063h,00h        ;    'x'  78
        DB   00h,00h,066h,066h
        DB   066h,03eh,06h,07ch        ;    'y'  79
        DB   00h,00h,07eh,04ch
        DB   018h,032h,07eh,00h        ;    'z'  7a
        DB   0eh,018h,018h,070h
        DB   018h,018h,0eh,00h         ;    '{'  7b
        DB   0ch,0ch,00h,00h
        DB   0ch,0ch,0ch,00h           ;    '|'  7c
        DB   070h,018h,018h,0eh
        DB   018h,018h,070h,00h        ;    '}'  7d
        DB   03bh,06eh,00h,00h
        DB   00h,00h,00h,00h           ;    '~'  7e
        DB   00h,08h,01ch,036h
        DB   063h,063h,07fh,00h        ;         7f
```

to

```
DB   00h,00h,00h,00h,00h,00h,00h,00h      ;    0
DB   7eh,81h,0a5h,81h,0bdh,99h,81h,7eh    ;    1
DB   7eh,0ffh,0dbh,0ffh,0c3h,0e7h,0ffh,7eh ;   2
DB   6ch,0feh,0feh,0feh,7ch,38h,10h,00h   ;    3
DB   10h,38h,7ch,0feh,7ch,38h,10h,00h     ;    4
DB   38h,7ch,38h,0feh,0feh,7ch,38h,7ch    ;    5
DB   10h,10h,38h,7ch,0feh,7ch,38h,7ch     ;    6
DB   00h,00h,18h,3ch,3ch,18h,00h,00h      ;    7
DB   0ffh,0ffh,0e7h,0c3h,0c3h,0e7h,0ffh,0ffh ; 8
DB   00h,3ch,66h,42h,42h,66h,3ch,00h      ;    9
DB   0ffh,0c3h,99h,0bdh,0bdh,99h,0c3h,0ffh ;   a
```

```
DB    0fh,07h,0fh,7dh,0cch,0cch,0cch,78h      ;      b
DB    3ch,66h,66h,66h,3ch,18h,7eh,18h         ;      c
DB    3fh,33h,3fh,30h,30h,70h,0f0h,0e0h       ;      d
DB    7fh,63h,7fh,63h,63h,67h,0e6h,0c0h       ;      e
DB    99h,5ah,3ch,0e7h,0e7h,3ch,5ah,99h       ;      f
DB    80h,0e0h,0f8h,0feh,0f8h,0e0h,80h,00h    ;      10
DB    02h,0eh,3eh,0feh,3eh,0eh,02h,00h        ;      11
DB    18h,3ch,7eh,18h,18h,7eh,3ch,18h         ;      12
DB    66h,66h,66h,66h,66h,00h,66h,00h         ;      13
DB    7fh,0dbh,0dbh,7bh,1bh,1bh,1bh,00h       ;      14
DB    3eh,63h,38h,6ch,6ch,38h,0cch,78h        ;      15
DB    00h,00h,00h,00h,7eh,7eh,7eh,00h         ;      16
DB    18h,3ch,7eh,18h,7eh,3ch,18h,0ffh        ;      17
DB    18h,3ch,7eh,18h,18h,18h,18h,00h         ;      18
DB    18h,18h,18h,18h,7eh,3ch,18h,00h         ;      19
DB    00h,18h,0ch,0feh,0ch,18h,00h,00h        ;      1a
DB    00h,30h,60h,0feh,60h,30h,00h,00h        ;      1b
DB    00h,00h,0c0h,0c0h,0c0h,0feh,00h,00h     ;      1c
DB    00h,24h,66h,0ffh,66h,24h,00h,00h        ;      1d
DB    00h,18h,3ch,7eh,0ffh,0ffh,00h,00h       ;      1e
DB    00h,0ffh,0ffh,7eh,3ch,18h,00h,00h       ;      1f
DB    00h,00h,00h,00h,00h,00h,00h,00h         ;' '   20
DB    30h,78h,78h,30h,30h,00h,30h,00h         ;'!'   21
DB    6ch,6ch,6ch,00h,00h,00h,00h,00h         ;'"'   22
DB    6ch,6ch,0feh,6ch,0feh,6ch,6ch,00h       ;'#'   23
DB    30h,7ch,0c0h,78h,0ch,0f8h,30h,00h       ;'$'   24
DB    00h,0c6h,0cch,18h,30h,66h,0c6h,00h      ;'%'   25
DB    38h,6ch,38h,76h,0dch,0cch,76h,00h       ;'&'   26
DB    60h,60h,0c0h,00h,00h,00h,00h,00h        ;''    27
DB    18h,30h,60h,60h,60h,30h,18h,00h         ;'('   28
DB    60h,30h,18h,18h,18h,30h,60h,00h         :')'   29
DB    00h,66h,3ch,0ffh,3ch,66h,00h,00h        :'*'   2a
DB    00h,30h,30h,0fch,30h,30h,00h,00h        ;'+'   2b
DB    00h,00h,00h,00h,00h,30h,30h,60h         ;','   2c
DB    00h,00h,00h,0fch,00h,00h,00h,00h        ;'—'   2d
DB    00h,00h,00h,00h,00h,30h,30h,00h         ;'-'   2e
DB    06h,0ch,18h,30h,60h,0c0h,80h,00h        ;'/'   2f
DB    7ch,0c6h,0ceh,0deh,0f6h,0e6h,7ch,00h    ;'0'   30
```

```
DB   30h,70h,30h,30h,30h,30h,0fch,00h      ;'1'  31
DB   78h,0cch,0ch,38h,60h,0cch,0fch,00h    ;'2'  32
DB   78h,0cch,0ch,38h,0ch,0cch,78h,00h     ;'3'  33
DB   1ch,3ch,6ch,0cch,0feh,0ch,1eh,00h     ;'4'  34
DB   0fch,0c0h,0f8h,0ch,0ch,0cch,78h,00h   ;'5'  35
DB   38h,60h,0c0h,0f8h,0cch,0cch,78h,00h   ;'6'  36
DB   0fch,0cch,0ch,18h,30h,30h,30h,00h     ;'7'  37
DB   78h,0cch,0cch,78h,0cch,0cch,78h,00h   ;'8'  38
DB   78h,0cch,0cch,7ch,0ch,18h,70h,00h     ;'9'  39
DB   00h,30h,30h,00h,00h,30h,30h,00h       ;':'  3a
DB   00h,30h,30h,00h,00h,30h,30h,60h       ;';'  3b
DB   18h,30h,60h,0c0h,60h,30h,18h,00h      ;'<'  3c
DB   00h,00h,0fch,00h,00h,0fch,00h,00h     ;'='  3d
DB   60h,30h,18h,0ch,18h,30h,60h,00h       ;'>'  3e
DB   78h,0cch,0ch,18h,30h,00h,30h,00h      ;'?'  3f
DB   7ch,0c6h,0deh,0deh,0deh,0c0h,78h,00h  ;'@'  40
DB   30h,78h,0cch,0cch,0fch,0cch,0cch,00h  ;'A'  41
DB   0fch,66h,66h,7ch,66h,66h,0fch,00h     ;'B'  42
DB   3ch,66h,0c0h,0c0h,0c0h,66h,3ch,00h    ;'C'  43
DB   0f8h,6ch,66h,66h,66h,6ch,0f8h,00h     ;'D'  44
DB   0feh,62h,68h,78h,68h,62h,0feh,00h     ;'E'  45
DB   0feh,62h,68h,78h,68h,60h,0f0h,00h     ;'F'  46
DB   3ch,66h,0c0h,0c0h,0ceh,66h,3eh,00h    ;'G'  47
DB   0cch,0cch,0cch,0fch,0cch,0cch,0cch,00h ;'H'  48
DB   78h,30h,30h,30h,30h,30h,78h,00h       ;'I'  49
DB   1eh,0ch,0ch,0ch,0cch,0cch,78h,00h     ;'J'  4a
DB   0e6h,66h,6ch,78h,6ch,66h,0e6h,00h     ;'K'  4b
DB   0f0h,60h,60h,60h,62h,66h,0feh,00h     ;'L'  4c
DB   0c6h,0eeh,0feh,0feh,0d6h,0c6h,0c6h,00h ;'M'  4d
DB   0c6h,0e6h,0f6h,0deh,0ceh,0c6h,0c6h,00h ;'N'  4e
DB   38h,6ch,0c6h,0c6h,0c6h,6ch,38h,00h    ;'O'  4f
DB   0fch,66h,66h,7ch,60h,60h,0f0h,00h     ;'P'  50
DB   78h,0cch,0cch,0cch,0dch,78h,1ch,00h   ;'Q'  51
DB   0fch,66h,66h,7ch,6ch,66h,0e6h,00h     ;'R'  52
DB   78h,0cch,0e0h,70h,1ch,0cch,78h,00h    ;'S'  53
DB   0fch,0b4h,30h,30h,30h,30h,78h,00h     ;'T'  54
DB   0cch,0cch,0cch,0cch,0cch,0cch,0fch,00h ;'U'  55
DB   0cch,0cch,0cch,0cch,0cch,78h,30h,00h  ;'V'  56
```

```
DB  0c6h,0c6h,0c6h,0d6h,0feh,0eeh,0c6h,00h;'W' 57
DB  0c6h,0c6h,6ch,38h,38h,6ch,0c6h,00h    ;'X' 58
DB  0cch,0cch,0cch,78h,30h,30h,78h,00h    ;'Y' 59
DB  0feh,0c6h,8ch,18h,32h,66h,0feh,00h    ;'Z' 5a
DB  78h,60h,60h,60h,60h,60h,78h,00h       ;'[' 5b
DB  0c0h,60h,30h,18h,0ch,06h,02h,00h      ;'\' 5c
DB  78h,18h,18h,18h,18h,18h,78h,00h       ;']' 5d
DB  10h,38h,6ch,0c6h,00h,00h,00h,00h      ;' ' 5e
DB  00h,00h,00h,00h,00h,00h,00h,0ffh      ;'—' 5f
DB  30h,30h,18h,00h,00h,00h,00h,00h       ;'"' 60
DB  00h,00h,78h,0ch,7ch,0cch,76h,00h      ;'a' 61
DB  0e0h,60h,60h,7ch,66h,66h,0dch,00h     ;'b' 62
DB  00h,00h,78h,0cch,0c0h,0cch,78h,00h    ;'c' 63
DB  1ch,0ch,0ch,7ch,0cch,0cch,76h,00h     ;'d' 64
DB  00h,00h,78h,0cch,0fch,0c0h,78h,00h    ;'e' 65
DB  38h,6ch,60h,0f0h,60h,60h,0f0h,00h     ;'f' 66
DB  00h,00h,76h,0cch,0cch,7ch,0ch,0f8h    ;'g' 67
DB  0e0h,60h,6ch,76h,66h,66h,0e6h,00h     ;'h' 68
DB  30h,00h,70h,30h,30h,30h,78h,00h       ;'i' 69
DB  0ch,00h,0ch,0ch,0ch,0cch,0cch,78h     ;'j' 6a
DB  0e0h,60h,66h,6ch,78h,6ch,0e6h,00h     ;'k' 6b
DB  70h,30h,30h,30h,30h,30h,78h,00h       ;'l' 6c
DB  00h,00h,0cch,0feh,0feh,0d6h,0c6h,00h  ;'m' 6d
DB  00h,00h,0f8h,0cch,0cch,0cch,0cch,00h  ;'n' 6e
DB  00h,00h,78h,0cch,0cch,0cch,78h,00h    ;'o' 6f
DB  00h,00h,0dch,66h,66h,7ch,60h,0f0h     ;'p' 70
DB  00h,00h,76h,0cch,0cch,7ch,0ch,1eh     ;'q' 71
DB  00h,00h,0dch,76h,66h,60h,0f0h,00h     ;'r' 72
DB  00h,00h,7ch,0c0h,78h,0ch,0f8h,00h     ;'s' 73
DB  10h,30h,7ch,30h,30h,34h,18h,00h       ;'t' 74
DB  00h,00h,0cch,0cch,0cch,0cch,76h,00h   ;'u' 75
DB  00h,00h,0cch,0cch,0cch,78h,30h,00h    ;'v' 76
DB  00h,00h,0c6h,0d6h,0feh,0feh,6ch,00h   ;'w' 77
DB  00h,00h,0c6h,6ch,38h,6ch,0c6h,00h     ;'x' 78
DB  00h,00h,0cch,0cch,0cch,7ch,0ch,0f8h   ;'y' 79
DB  00h,00h,0fch,98h,30h,64h,0fch,00h     ;'z' 7a
DB  1ch,30h,30h,0e0h,30h,30h,1ch,00h      ;'{' 7b
DB  18h,18h,18h,00h,18h,18h,18h,00h       ;'|' 7c
```

```
DB   0e0h,30h,30h,1ch,30h,30h,0e0h,00h      ;'}'   7d
DB   76h,0dch,00h,00h,00h,00h,00h,00h       ;'~'   7e
DB   00h,10h,38h,6ch,0c6h,0c6h,0feh,00h     ;' '   7f
```

Aug 8 16:44 1984 graph.src:
Solved 640 X 400 inverse video mode problem

Changed

```
  test  byte ptr [si],80H          ;is upper left bit of
                                   ;char = 0 or 1?
```

to

```
  test  byte ptr ss:[si],80H       ;is upper left bit of
                                   ;char = 0 or 1?
```

Changed

```
  not   byte ptr [si]              ;reverse the reversed
                                   ;byte for matching
```

to

```
  not   byte ptr ss:[si]           ;reverse the reversed
                                   ;byte for matching
```

Aug 8 16:48 1984 pwrup1.src:
Changed release date and display on boot

Changed

```
banner_m db      "Resident Diagnostics",CR,LF
         db      "Rev 1.0 May 1984",CR,LF,LF,
                 NUL
```

to

```
         db      "Rev 1.1 Aug 1984",CR,LF,LF
banner_m db      "Resident Diagnostics",CR,LF,
                 LF
         db      NUL
```

Notes on Enhancements in ROM BIOS 1.21

Introduction

These notes describe the differences between the 1.0, 1.1 and 1.21 ROM BIOS for the AT&T PC 6300. The notes first describe the differences between the 1.0 and 1.1 ROMs, then describe the differences between the 1.1 and 1.21 ROMs.

Differences Between ROM 1.0 and ROM 1.1

The differences between the 1.0 and 1.1 ROM are:

- In the floppy disk utility, the random value of the DX register is fixed.
- In the floppy disk utility, the verify operation without a valid buffer pointer is allowed.
- In the floppy disk utility, the value checking for head number has been eliminated.
- The ROM release date has been changed to 08/10/84.
- The 8x8 font has been changed to improve italics displayed by some third-party software.
- The 640x400 inverse video mode now works properly.
- The displayed release marker has been changed to "Rev 1.1."
- LOTUS™ Symphony runs on the 1.1 ROM.

These changes are explained in detail in the "System Programmer's Guide" on pages 8-177 to 8-188.

Differences Between ROM 1.1 and ROM 1.21

ROM BIOS 1.21 was introduced to solve problems with the use of external vendor's hard disk expansion boxes, to support the 20 megabyte hard disk drives, and to better initialize the second communications port.

The differences between ROM 1.1 and 1.21 are:

- The code sent to the parallel port during power up diagnostics has been changed from 80H to 3fH.
- In INT 15H, a near return has been changed to a far return.
- INT 18H is enhanced to display a message "ROM BASIC not present, Press RESET to reboot ..." and wait for a key press before re-booting.
- Code has been added to reset the Display Enhancement Board during a warm boot.
- The reset code has been changed to origin at 0E05Bh.
- The floppy disk drive motor delay has been changed from a range of 0-500 milliseconds to 250-750 milliseconds.
- The order of executing HDU code and external ROM code has been reversed to execute the internal HDU code, then check for optional ROMs.

- In order to simplify support for external hard disk drives with controllers other than the DTC controller, a check of DIPSW-1 position 3 on the motherboard (at location 7W) has been added. Position 3 of DIPSW-1 has been defined as:

  ON = use indigenous hard disk code
  OFF = use external hard disk code

- A test has been added to check for the presence of the 8530 alternate communications chip.

- The initialization of COM2: has been improved.

- Code has been added in an INT 10H routine that enables the video after scroll up is performed.

- An incompatibility that prevented the Softech P-System from booting has been corrected.

- The high 128 characters of the 8x8 font and the low 128 characters of the 8x16 font have been changed to provide better alignment of the characters within the 8x8 and 8x16 fonts.

- Support for 20 megabyte disk drives has been added to the disk parameter table. Drive type 6 is a Seagate ST225, drive type 7 is a Miniscribe 3425, and drive type 8 is a CMI 6426.

- The disk parameters for the CDC WREN drive have been changed to 697 cylinders to allow for the full capacity of the drive.
- Support in INT 11H was added for 8087 switch setting.
- The ROM release date has been changed to 02/28/85.
- The displayed release marker has been changed to "Rev 1.21."

ROM Revision 1.1 to ROM Revision 1.21 Source File Differences

```
   8/10/84              2/14/85
   < = rom1.1           > = rom1.20

Source Module:       comm2.src

<    call rs_stat                ; get return status
---
> ;; call rs_stat          *E912*     ; get return status

Source Module:       fdu1.src

>  * mikef      9/18/84         Changed motor delay routine.
>  * mikef      01/11/85  If reset cmd don't do check valid.
>  * mikef      02/14/85  Made the reset code callable.

<    jz    diskette_io1                ; Yes, so ignore drive parm.
---
>    jz    diskette_io2   *EC73*          ; Yes, so ignore drive parm.

<    cld                         ; Autoincrement for strings.


<    jnz   f_rd1                       ; 1 = 250 ms motors, no delay
---
>    mov   cx,250           *ED60*           ;
>    jnz   f_rd_loop *ED63*           ; 1 = 500 ms motors.

Source Module:       flags.src

>  * mikef      9/18/84         Changed reset pointer to point to org'ed
>  *                 location.
>  * mikef      10/02/84  Included hdu1.asm

> include pwrup1a.asm

> include hdu1.asm
>

>
> include int18.asm
>

<    dw    diagnostics_1  ; instruction pointer cs:(offset diagnostics_1)
---
>    dw    i_hard_reset   ; instruction pointer cs:(offset i_hard_reset)

<    db    '08/10/84'     ; release marker (exactly 8 bytes!!!!)
---
>    db    '02/14/85'     ; release marker (exactly 8 bytes!!!!)
```

```
Source Module:      fonthi8.src

<

<  * NAME DATE      ACTION
<  * ---- ----      ------
<  * robert    4/30/84        Corrected '^'
<  * robert    5/01/84        Corrected char 182
---
>  * name      date      action
>  * ----      ----      ------
>  * seagrave  12/13/84  new font table
< fonthi8 endp
---
> fonthi8 endp

Source Module:      fontlo16.src

<  * NAME DATE      ACTION
<  * ---- ----      ------
<  * robert    4/30/84        corrected 'p'.
<  * robert    5/4/84         corrected '!'
---
>  * NAME      DATE      ACTION
>  * ----      ----      ------
>  * seagrave  12/14/84  new font table
>  * yin       2/5/85         new font table

Source Module:      graph.src

>  * mikef          2/13/85        Moved grf_light_pen to vid.src.

<
<
;===========================================================================
< ;
< ;  Read Light Pen            function code = 04h
< ;
< ;  Input:    None.
< ;  Output:   ah    = 0 light pen switch not down/not triggered
< ;       ah   = 1 implies:
< ;            (dh,dl)    = (row,col) of character light pen
< ;                  position from (0,0)
< ;            ch    = raster line (0-199)
< ;            bx    = pixel column (0-319,0-639)
< ;
< ;  Trash:    None.   ???
< ;
< ;---------------------------------------------------------------------------
<
< grf_light_pen     proc near
<
<    xor  ah,ah                ; return ah = 0 for now (al intact)...
```

```
<    ret
<
< grf_light_pen     endp

Source Module:      hdu.src

< /*
<  *
<  * NAME DATE      ACTION
---
> /* NAME DATE      ACTION

>  * mikef     10/02/84  Moved h_data2 to hdu1.src to make rom in
>  *                 low rom for an ORG.
>  * mikef     10/23/84  Changed documentation describing dip
>  *                 switches.
>  * mikef     11/02/84  Fixed bug in reset call.
>  * mikef     12/06/84  Corrected no. of cyls for CDC drive.
>  * mikef     12/18/84  Added type 6 and type 7 drives.
>  * mikef     12/20/84  Corrected write precomp for Seagate ST225.
>  * mikef     01/08/84  Changed int 19 to jmp bt_int.
>  * mikef     01/21/84  Added type 8 in parameter_table and changed
>  *                 some other parameters per Olivetti memo.

<    int     18h             ; initiate reboot through ROM BASIC INT.
---
>    jmp  bt_int        *D124*     ; initiate reboot through floppy.
> ;  int     19h                     ; initiate reboot through ROM BASIC
INT.

< hdu_parm_tbl:
---
> hdu_parm_tbl proc near

< ; The next six are the supported drives:
---
> ; The next nine are the supported drives:

<         parameter_table <>                        ;type 3: 10mb drive
---
>    *D17C*    parameter_table <,,,128>      ;type 3: 10mb drive

<         parameter_table     <644d,5d,128d,128d> ;type 5: CDC Wren
---
>         parameter_table     <697d,5d,697d,0>    ;type 5: CDC Wren
>         parameter_table <612,4,256,256,,3> ;type 6: Seagate ST225.
>         parameter_table <612,4,128,128,,3> ;type 7: Miniscribe 3425
>                                 ; and CMI CM 4426.
>    *D1CC*    parameter_table <640,4,256,256,,3> ;type 8: CMI CM6426.

> hdu_parm_tbl endp
>

<    int  40h                ; if not, pass to the FDU driver
```

```
<
---
> ;; int  40h                    ; if not, pass to the FDU driver
>    jmp  near ptr fd_io          *D1E1*    ;; rom floppy driver.
>                                 ;; will not return.

<    int  40h                    ; if so, must reset FDU also
---
> ;; int  40h                    ; if so, must reset FDU also
>    pushf                            ;; simulate an int because
>                                 ;; fd_io simulates an iret.
>    push cs                 *D1EE*    ;; Make it look like a far call.
>    call near ptr fd_io          *D1EF*    ;; rom floppy driver.

> ; Test to see if the vector at location 0:100 points to me. If it does it
> ; means that an optional rom thinks I'm the floppy driver so I'll just
return.
>

< ;  drive 0          0   1           2   3
< ;  drive 1                  0   1           2   3
---
> ;; drive 0                  0   1           2   3
> ;; drive 1          0   1           2   3

>

<    and  al,07fh                ;ignore top switch!
---
> ;; and  al,07fh                ;ignore top switch!

< h_data2 proc
<
< h_intro_m     db   'Fixed Disk Formatting Utility',CR,LF
<         db   'All data on the specified fixed disk will be erased.'
<         db   CR,LF
<         db   'Enter fixed disk number (1 to 8) or "Q" to quit: ',NUL
<
< h_fmt_m       db   CR,LF,'Formatting Fixed Disk...',CR,LF,NUL
<
< h_pass_m      db   'Format is complete.',CR,LF
<         db   'Proceed with FDISK and FORMAT.',NUL
<
< h_none_m      db   'Error: No fixed disk drive exists for this
number.',NUL
<
< h_err_m       db   'Format Error.  Code: ',NUL
<
< h_data2 endp
```

```
Source Module:      mem.src

< /*
<  *
<  * NAME DATE      ACTION
---
> /* NAME DATE      ACTION

< *
---
>  * mikef     9/18/84        Changed m_cass proc to far.

>

< m_cass  proc near
---
> m_cass  proc far  *F8F9*

Source Module:      nmi.src

< /*
<  *
<  * NAME DATE      ACTION
---
> /* NAME DATE      ACTION

>  * mikef     12/06/84  Added code to inform user that a parity
>  *                    error happend.
> ;  And ENABLE_PARITY
> ;

<     assume    cs:code, ds:nothing, es:nothing, ss:nothing

>     push ax              *F85F*
>     in   al,ControlC               ; High two bits indicate parity.
>     and  al,0C0h                   ; Mask of low 6 bits.
>     jz   n_out                     ; It wasn't a parity interrupt!
>     mov  si,offset parity1_m       ; System board message.
>     rol  al,1
>     jc   n_1
>     mov  si,offset parity2_m       ; Expansion board message.
> n_1:
>     call DRomString
>     hlt
> n_out:
>     pop  ax              *F874*

Source Module:      pwrup0.src

<
<
< /*
<  *
<  * NAME DATE      ACTION
---
> /* NAME DATE      ACTION

>  * mikef     9/18/84        Changed OK code to parallel port for mfg.
```

```
>  * mikef      10/02/84  Changed int 18 trap to go to 'basic_trap'.
>  * mikef      10/11/84  Added test to see if 8530 is really there.
>  * mikef      11/20/84  Now executes internal HDU code before
>  *                optional ROM checking.
>  * mikef      12/07/84  Changed OK code to '3Fh' for mfg.
>  * mikef      12/13/84  Added call to enable parity interrupt.
>  * mikef      12/17/84  Added switch reading for indiginous HDU code.

>
>    mov  al,0Fh               *E2FB*    ;;
>    out  scc_ctl_a,al              ;; read register 15.
>    in   al,scc_ctl_a              ;;
>    test al,1                 ;;
>    jnz  i_no_sccs       *E303*    ;; LSB of rr15 is always 0.
>

> ; Call internal HDU init code.
> ;---------------------------------------------------------------------------
--
>
>
>    assume    cs:code, ds:abs0, es:nothing, ss:stack_ram
>
>    xor  ax,ax                *E33D*    ; satisfy assumptions
>    mov  ds,ax
>
>    in   al,sys_conf_b             ;; port 67h.
>    test al,4                 ;; test switch bit 2
>    jnz  i_hdu_ok             ;; if set, skip init
>
>    mov  si,cs:(offset i_hdu_m)
>    call DRomString                ; print test message
>
>    call h_init
>
>    assume    cs:code, ds:data, es:nothing, ss:stack_ram
>
>    mov  ds,word ptr cs:[set_ds_word]  ; satisfy assumptions
>    cmp  byte ptr ds:[hf_num],0        ; number of hard disks.
>    jnz  i_hdu_ok             ; if ok, leave everything alone.
>
>    mov  sp,100h                   ; re-initialize stack
>    cli                       ; disable interrupts
>    call i_vector             ; re-install old vectors
>    sti                  *E363*
> i_hdu_ok:
>
> ;---------------------------------------------------------------------------
<
< ;---------------------------------------------------------------------------
< ;  HDU Test
< ;---------------------------------------------------------------------------
<
<    assume    cs:code, ds:abs0, es:nothing, ss:stack_ram
```

```
<
<     xor  ax,ax                ; satisfy assumptions
<     mov  ds,ax
<
< ; Check int 41h to see if any one installed a HDU parameter table
pointer.
<
<     mov  ax,word ptr ds:[(4*41h)+0000h]
<     or   ax,word ptr ds:[(4*41h)+0002h]
<     jnz  i_hdu_ok             ; if so, let them be...
<
< ; If not, call HDU initialization routine.
<
<     mov  si,cs:(offset i_hdu_m)
<     call DRomString                 ; print test message
<
<     call h_init
<
<     assume    cs:code, ds:data, es:nothing, ss:stack_ram
<
<     mov  ds,word ptr cs:[set_ds_word]  ; satisfy assumptions
<     cmp  byte ptr ds:[hf_num],0        ; number of hard disks.
<     jnz  i_hdu_ok             ; if ok, leave everything alone.
<
<     mov  sp,100h                    ; re-initialize stack
<     cli                       ; disable interrupts
<     call i_vector             ; re-install old vectors
<
< i_hdu_ok:
<

>     call DCrLf                *E3BE*    ;;

<

< ;   and  al,10111100b         ; p_timer & kb & dsk at this point.

<
<     assume    cs:code, ds:abs0, es:nothing, ss:stack_ram
<
<     xor  ax,ax
<     mov  ds,ax
<     mov  word ptr ds:[int18locn+0000h],cs:(offset bt_int)
<     mov     word ptr ds:[int18locn+0002h],cs     ; (ROM BASIC not
available)
<

< ; Initialize & enable NMI's (parity register).
<
<     mov  dx,p_kctrl
<     in   al,dx
<     or   al,030h                    ; enable bits #5 & #4
<     out  dx,al
<
```

```
<     mov  al,80h                    ; OK status
---
>     mov  al,3Fh          *E49E*          ; OK status

Source Module:       pwrup1.src

>   * mikef      09/18/84  Put in org for reset vector.
>   * mikef      12/10/84  Added parity stuff.
>   * mikef      01/08/85  Clear the screen before printing messages.

<          db    'Rev 1.1',CR,LF,LF
---
>          db    'Rev 1.20',CR,LF,LF

>     or   cx,cx             *DD68*     ;; if zero then it was a parity error.
>     jnz  i_d_e                              ;;
>     mov  si,cs:(offset parity1_m) *DD6C*     ;;
> i_d_e:                          ;;

>     mov  ax,3        *DD28*          ; mode co80
>     int  10h         *DD2B*          ; Clear screen.
>

> skip_parity:

<          assume    cs:code, ds:data, es:abs0, ss:stack_ram
<

>     ORG  0E05Bh                ;;
>
> i_hard_reset proc            ;;
>     jmp  diagnostics_1       ;;
> i_hard_reset endp            ;;
>

<     push ds                    ; save registers
---
> ;; Here is the code for putting the DEB in Transparent Mode.

>     push ax          *E05E*
>     push dx
>     mov  dx,03DDh         ;; DEB I/O Address Register port address
>     mov  al,1             ;; select Mode Control Register
>     out  dx,al
>
>     inc  dx               ;; DEB Mode Control Register address
>     inc  dx
>     dec  al               ;; set DEB Transparent Mode
>     out  dx,al                 ;;
>     pop  dx
>     pop  ax          *E06C*
>
```

```
>    push ds                    ; save registers
>

<    mov  al,cl                      ; get data from switches.
---
>    in   al,sys_conf_a  *E0B4*          ; Read port 66h.
>    and  al,010h                    ; Keep 8087 bit only.
>    shr  al,1                  ; Move to bit one.
>    shr  al,1
>    shr  al,1
>    or   al,cl          *E0BE*          ; get data from switches.

> ;  NOTE:          If CX is zero and ZF is nz then parity error occured.

>
> ;; Toggle parity latch
>    in   al,p_kctrl     *E1A1*          ;; read B port. (61h)
>    or   al,30h                     ;; toggle bits 4 & 5.
>    out  p_kctrl,al                 ;;
>    and  al,0CFh                    ;;
>    out  p_kctrl,al     *E1A9*          ;;
>
```

Source Module: pwrup4.src

```
< /*
<  *
<  * NAME DATE      ACTION
---
> /* NAME DATE      ACTION

>  * mikef     12/06/84  Added 'enable_parity'

>


>
> enable_parity     proc                ;;
>
>    push ax             *E5C5*
>    in   al,p_kctrl                 ;; read B port. (61h)
>    or   al,30h                     ;; enable bits 4 & 5.
>    out  p_kctrl,al                 ;;
>    and  al,0CFh                    ;;
>    out  p_kctrl,al                 ;;
>    mov  al,nmi_enable              ;; 80h.
>    out  nmi_enable_port,al         ;; defined in sysdata.src (A0h)
>    pop  ax                    ;;
>    ret                        ;;
>
> parity1_m    db   'Parity error on system board',NUL ;;
> parity2_m    db   'Parity error on expansion board',NUL   ;;
>                   *E61C*
> enable_parity     endp                ;;
```

```
Source Module:      sysdata.src

< ControlB     equ  062h ; 8087, etc.
---
> ControlC     equ  062h ; bit  #7:     Ram parity check.
>                     ; bit  #6:     I/O channel parity check.
>                     ; bit  #1:     8087 installed

<                     ; bits #3 - #2:     reserved for HDU type
---
>                     ; bit  #3 not defined.
>                     ; bit  #2 - 0 = use indiginous HDU code.
>                     ;       1 = do not use indiginous HDU code.

> nmi_enable   equ  80h
> nmi_enable_port   equ  0A0h

Source Module:      vector.src

<    dw   bt_int              ; int18locn    (We Don't Have BASIC!)
---
>    dw   basic_trap          ; int18locn    see int18.src

Source Module:      vid.src

< /*
<  *
<  * NAME DATE       ACTION
---
> /* NAME DATE       ACTION

>  * mikef     10/25/84  Added code in scroll up to enable video.
>  * mikef     02/13/85  Moved grf_light_pen from graph.src to here.

>    mov  dx,03D8h       *F356*    ;; video enable register.
>    out  dx,al               *F359*    ;; enable video.

< ; We did't disable display during vertical retrace...
---
> ; We did't disable display during vertical retrace...but enable it
anyway.
>
>
;=============================================================================
> ;
> ;  Read Light Pen           function code = 04h
> ;
> ;  Input:    None.
> ;  Output:   ah   = 0 light pen switch not down/not triggered
> ;       ah   = 1 implies:
> ;            (dh,dl)   = (row,col) of character light pen
> ;                  position from (0,0)
> ;            ch   = raster line (0-199)
> ;            bx   = pixel column (0-319,0-639)
```

```
> ;
> ;  Trash:    None.   ???
> ;
> ;--------------------------------------------------------------------------
>
> grf_light_pen     proc near
>
>    xor  ah,ah                ; return ah = 0 for now (al intact)...
>    ret
>
> grf_light_pen     endp
```

9 MS-DOS Device Drivers

- Overview
- MS-DOS Device Drivers
- Asynchronous Communications Element
- DMA Controller
- Floppy Diskette Interface and Controller
- Hard Disk Controller
- Keyboard Interface
- Parellel Printer Interface
- Programmable Interrupt Controller
- Programmable Interval Timer
- Real Time Clock and Calendar
- Serial Communications Controller
- Speaker
- Video Controller

Overview

Interested Audience

This section contains information on how to write device drivers. You may not use it often since many input and output capabilities are implemented by the BIOS routines discussed in Section 6. BIOS routines allow you to do general input and output without a detailed understanding of the hardware and shield your program from hardware changes.

However, there are times when BIOS routines do not perform the necessary function or do so in an inefficient fashion. Then your own driver is necessary. Implementation of operating systems, of high speed graphics packages, and of unusual keyboard mapping are examples of software which require specialized drivers.

Programmable Devices

This is a list of programmable devices.

INS	8350B	Asynchronous Communications Element
INTEL	8237A	DMA Controller
NECE	uPD765	Floppy Diskette Controller
DTC	5150BX	Hard Disk Controller
INTEL	8041	Keyboard Interface
INTEL	8259A	Programmable Interrupt Controller
INTEL	8253	Programmable Interval Timer
	58174A	Real Time Clock and Calendar
AMD	Z8530	Serial Communications Controller
		Speaker Interface
HD	6845	Video Controller

Port Addresses

A port is a place to read or write information to an I/O device. An I/O device is hardware which the CPU controls. It is both input and output peripherals as well as hardware such as the DMA controller, the Interrupt controller and the Interval Timer. Each device needs different information, but they all need some combination of control, status, and data.

Each port has an address. Sixteen of the twenty possible address lines are available for I/O addressing. This means there are 65,535 possible port addresses.

I/O Instructions

IN and OUT instructions distinguish an I/O access from a memory access. These instructions translate into control signals which define the direction and path of the data.

The port address can be specified in one of two ways — fixed or variable. In the fixed method, the absolute port address is specified in the instruction. The following instruction is an example of fixed port addressing:

```
OUT 020H, AL
```

Only 8-bit port addresses can be used in this format.

The variable method allows 16-bit port addresses to be specified. In this case, the port address is loaded into the DX register and then the DX register is used in the I/O instruction. An example of variable port addressing follows:

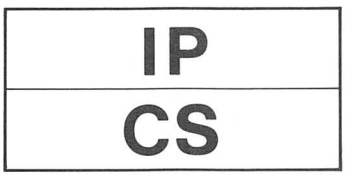
```
MOV DX,03F2H
OUT DX,AL
```

Interrupts

External devices do not need the CPU's attention all the time. When they need servicing, they ask for it either by interrupting or by setting a polled flag. External interrupts are ignored when the CLI instruction has cleared the interrupt enable flag and are recognized when the STI instruction has set the flag.

The first 1024 bytes of memory contain an interrupt table. This table has 255 interrupt pointers defining the start address of interrupt service routines. This is the pointer format.

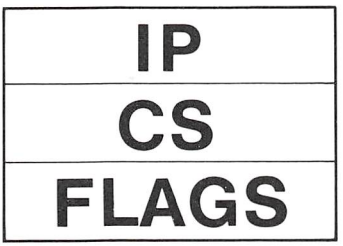
INTERRUPT POINTER



When the CPU recognizes the interrupt, an 8-bit interrupt type identifies the device. The interrupt type is an index into the table of pointers. To obtain the interrupt pointer address, the type is multiplied by four. The CPU saves the flag register on the stack, disables interrupts and single step mode, and saves the CS and IP registers on the stack. Then the interrupt pointer is loaded into IP and CS, and control is transferred to the interrupt service routine. The stack looks like this when the service routine gets control.

Interrupt Devices

SYSTEM STACK



When the service routine begins, interrupts are disabled. Depending on the nature of the application, interrupts can be enabled immediately or just prior to releasing control.

The service routine must preserve the value of all internal registers. Therefore it saves the registers it uses on the stack. Before returning, it restores these registers from the stack.

To return control to the interrupt program, the service routine executes IRET. All the information necessary to do this was carefully placed on the stack

Interrupt Devices

INS	8250B	Asynchronous Communications Element
INTEL	8237A	DMA Controller
NEC	uPD765	Floppy Diskette Controller
DTC	5150BX	Hard Disk Controller
INTEL	8041	Keyboard Interface
INTEL	8259A	Programmable Interrupt Controller
INTEL	8253	Programmable Interval Timer
	58174A	Real Time Clock and Calendar
AMD	Z8530	Serial Communications Controller

Block Diagrams

Block diagrams are a pictorial description of the electrical connection between the CPU, the interface, and the external device. In general, the CPU's bus signals appear on the left. The middle of the diagram describes the internals of the interface. The right side of the diagram describes the electrical interface. This physically connects the interface to the external device. The connection is often through a dual inline package (DIP) of pins. Each pin carries one signal.

Individual signals are represented by lines. Busses are shown as double lines. Each of these have arrows which indicate the direction of the data. Often they are bidirectional.

There are several common CPU control bus signals. There mnemonics and definitions follow:

- A0-A19
  These are the lines used to transmit the address of memory or the address of an I/O port. Only A0-A15 are used for I/O addressing.

- CS
  This signal selects the chip. No reading or writing will occur unless the device is selected.

- D0-D7
  These are the bidirectional data lines used to exchange information with a memory location or an I/O port. D7 is the most significant bit.

- INT0-INT7
  These are the priority interrupt request lines. See the description of the Interrupt Controller.

- IORD
  This signal indicates that an input port address has been placed on the address bus. The data at the specified port is to be placed on the data bus.

- IOWR
  This signal indicates that an output port address has been placed on the address bus and the data has been placed on the data bus to be output to the specified port.

- RESET
  This signal resets the system to a predetermined state.

MS-DOS Device Drivers

What Is a Device Driver?

A device driver is binary code which manipulates hardware in the MS-DOS environment. A special header at the beginning identifies it as a driver, defines the strategy and interrupt entry points, and describes various attributes of the device. The file must have an origin of zero.

There are two kinds of devices:

- Character devices
- Block devices

Character devices perform serial character I/O like CONSOLE, AUXILIARY, and PRINTER. These devices are named and users open channels to do I/O to them.

Block devices are the disk drivers on the system. They perform random I/O in pieces called blocks. This is usually the physical sector size. These devices are not named as the character devices are, and cannot be opened directly. Instead they are identified by drive letters (A:,B:,C:, etc.).

Drive letters are assigned to device drivers based on their ordering in the CONFIG.SYS file. Starting with the letter 'A', each device driver is assigned as many consecutive alphabetic characters as the driver has units. The theoretical limit is 63, but after 26 the drive letters are non-alphabetic (such as] and ^).

Character devices cannot define multiple units because they have only one name.

Device Headers

A device header is required at the beginning of a device driver. A device header looks like this:

DWORD pointer to next device (Must be set to -1)
WORD attributes Bit 15 = 1 if char device, 0 if blk if bit 15 is 1 Bit 0 = 1 if current sti device Bit 1 = 1 if current sto device Bit 2 = 1 if current NUL device Bit 3 = 1 if current CLOCK dev Bit 4 = 1 if special Bits 5-12 Reserved; must be set to 0 Bit 14 is the IOCTL bit Bit 13 is the NON IBM FORMAT bit
WORD pointer to device strategy entry point
WORD pointer to device interrupt entry point
8-BYTE character device name field Character devices set a device name. For block devices the first byte is the number of units

The strategy and interrupt routines are in the same segment as this device header.

Pointer to Next Device Field

The pointer to the next device header field is a double word field, offset followed by segment. MS-DOS chains the device headers together using this field. If you have a single device header in your driver, initialize this field to -1. If you have more than one device header, the first word of the double word pointer is the offset of the next driver's Device Header.

Attribute Field

The attribute field is used to tell the system whether this device is a block or character device (Bit 15). Most other bits are used to give selected character devices special treatment and are meaningless on a block device. For example, assume that you have a new standard input and output device driver. Besides installing the driver, you must tell MS-DOS that you want this new driver to override the current standard input and standard output device. This is accomplished by setting the attributes to the desired characteristics, so you set Bits 0 and 1 to 1. Similarly, a new CLOCK device could be installed by setting the appropriate attribute. Although there is a NUL device attribute, it is reserved for MS-DOS.

The NON PC-DOS FORMAT bit applies only to block devices and affects the operation of the BUILD BPB (Bios Parameter Block) device call. The implementation of all block devices is PC-DOS software and hardware compatible.

The IOCTL bit is meaningful for both types of devices. This bit tells MS-DOS whether the device can handle control strings with the IOCTL system call, Function 44H.

If a driver cannot process control strings, this bit is 0. MS-DOS returns an error if an attempt is made to handle control strings. A device which can process control strings sets the IOCTL bit to 1. For drivers of this type, MS-DOS calls IOCTL INPUT and OUTPUT device functions to send and receive IOCTL strings.

The IOCTL functions allow data outside of the normal user's reads and writes to be sent to the driver. The interpretation of this information is up to the driver.

Strategy and Interrupt Routines

These two fields are the entry points of the strategy and interrupt routines. They are word values and they must be in the same segment as the Device Header.

Name Field

This 8-byte field contains the name of a character device or the number of units of a block device. If it is a block device, the number of units can be put in the first byte. This is optional because MS-DOS fills in this location with the value returned by the driver's INIT code.

How to Create a Device Driver

To create a device driver, write a binary file with a Device Header at the beginning of the file. The code originates at 0.

MS-DOS always processes installable device drivers before handling the default devices. To install a new CON device, simply name the device CON. Remember to set the standard input device and standard output device bits in the attribute word on the new CON device. The scan of the device list stops on the first match, so the installable device driver takes precedence.

MS-DOS installs the driver anywhere in memory; therefore, be careful with memory references. Do not expect the driver to be loaded in the same place.

Installation of Device Drivers

MS-DOS allows new device drivers to be installed dynamically at boot time. This is accomplished by INIT code in the BIOS which processes the CONFIG.SYS file.

At load time, DOS searches the root directory for a file named CONFIG.SYS. Declare the files containing your device drivers using the DEVICE command.

DEVICE = [C:] [path] filename [.ext]

DOS loads your drivers as an extension of itself. Include a separate DEVICE command for each driver to be loaded.

Request Header

When MS-DOS calls a device driver to perform a function, it passes a Request Header in ES:BX to the strategy entry point. This is a fixed length header followed by data pertinent to the operation being performed. It is the device driver's responsibility to preserve the machine state. For example, save all registers on entry and restore them on exit. There is enough room on the stack when strategy or interrupt is called to do about 20 pushes. If more stack is needed, the driver sets up its own stack.

The following figure illustrates a Request Header.

REQUEST HEADER ->

0	BYTE length of record (the length in bytes of pertinent data, plus the length of this Request Header)
1	BYTE unit code The subunit the operation is for (minor device) (no meaning on character devices)
2	BYTE command code
3	WORD status
5	8 bytes RESERVED

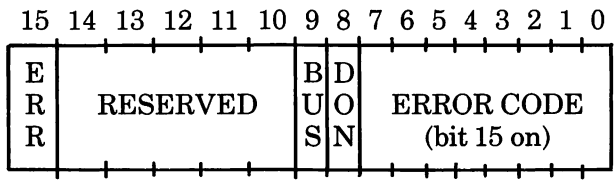
Unit Code

If your device driver has three units, then the possible values of the unit code field are 0, 1, and 2.

Command Code Field The command code field in the Request Header can have the following values:

Command Code	Function
0	INIT
1	MEDIA CHECK (Block only, NOP for character)
2	BUILD BPB "
3	IOCTL INPUT (Only called if device has IOCTL)
4	INPUT (Read)
5	NON-DESTRUCTIVE INPUT NO WAIT (Char devices only)
6	INPUT STATUS "
7	INPUT FLUSH "
8	OUTPUT (Write)
9	OUTPUT (Write) with verify
10	OUTPUT STATUS "
11	OUTPUT FLUSH "
12	IOCTL OUTPUT (Only called if device has IOCTL)

Status Word The following figure illustrates the status word in the Request Header.



The status word is set by the driver interrupt routine.

Bit 8 is the done bit. When set, it means the operation is complete.

Bit 15 is the error bit. If it is set, then the low 8 bits indicate the error. The errors are:

0 Write protect violation
1 Unknown Unit
2 Drive not ready
3 Unknown command
4 CRC error
5 Bad drive request structure length
6 Seek error
7 Unknown media
8 Sector not found
9 Printer out of paper
A Write fault
B Read Fault
C General failure

Bit 9 is the busy bit which is set only by status calls.

- For output on character devices:
  If bit 9 is 1 on return, a write request waits for completion of a current request. If it is 0, there is no current request and a write request starts immediately.

- For input on character devices with a buffer:
  If bit 9 is 1 on return, a read request goes to the physical device. If it is 0 on return, then there are characters in the device buffer and a read returns quickly. MS-DOS assumes all character devices have an input type-ahead buffer. Devices that do not have a type-ahead buffer return busy=0 so that MS-DOS does not wait for non-existent buffer input.

Media Check and Build BPB

MEDIA CHECK and BUILD BPB are used with block devices only. MS-DOS calls MEDIA CHECK first for a drive unit and passes its current media descriptor byte. MEDIA CHECK returns one of the following results:

- Media Not Changed — current DPB and media byte are OK.
- Media Changed — Current DPB and media are wrong. MS-DOS invalidates buffers for this unit and calls the device driver to build the BPB.
- Not Sure — If there are dirty buffers for this unit, MS-DOS assumes the DPB and media byte are OK. If nothing is dirty, MS-DOS assumes the media has changed. It invalidates buffers for the unit and calls the device driver to build the BPB.
- Error — If an error occurs, MS-DOS sets the error code.

MS-DOS calls BUILD BPB under the following conditions:

- If Media Changed is returned
- If Not Sure is returned and there are no dirty buffers

Init Routine

The Init Routine is called only once when the device is installed. It returns a location DS:DX which is a pointer to the first free byte of memory after the device driver. To save space, this pointer method can be used to delete initialization code that is only used once.

Additional information that block drivers return is:

- The number of units
- A pointer to a BPB
- The media decriptor

The number of units determines the logical device names. This mapping is determined by the position of the driver in the device list and by the number of units on the device.

BPB blocks are used to build an internal MS-DOS data structure for each of the units. The driver passes MS-DOS a pointer to an array of n word BPB pointers, where n is the number of units. If all units are the same, they can share a BPB to save space. This array must be before the free space pointer since MS-DOS builds an internal DOS structure starting at this free byte. The defined sector size must be less than or equal to the maximum sector size defined at INIT time; otherwise, the install fails.

The media descriptor byte means nothing to MS-DOS. It is passed to drivers so that they know what parameters MS-DOS is currently using for a drive unit.

Block devices are either dumb or smart. A dumb device defines a unit and an internal DOS structure for each possible media drive combination. For example, unit 0 = drive 0 single side, unit 1 = drive 0 double side. In this case, media descriptor bytes mean nothing.

A smart device allows multiple media per unit. In this case, the BPB table returned by INIT defines space large enough to accommodate the largest possible media supported. Smart drivers use the media descriptor byte to pass information about the media currently in a unit.

Function Call Parameters

Strategy routines are called with ES:BX pointing to the Request Header. The interrupt routines get the pointers to the Request Header from the queue that the strategy routines store them in. The command code in the Request Header tells the driver which function to perform.

All DWORD pointers are stored offset first, then segment.

INIT

Command code = 0

INIT - ES:BX ->

	13-BYTE Request Header
0D 13	BYTE # of units
0E 14	DWORD break address
10 16	DWORD pointer to BPB array (Not set by character devices)

The number of units, break address, and BPB pointer are set by the driver. On entry, the DWORD points to the character after the '=' on the line in CONFIG.SYS. This allows drivers to scan the CONFIG.SYS invocation line for arguments.

If there are multiple device drivers in a single .COM file, the ending address returned by the last INIT is the one MS-DOS uses. All of the device drivers in a single .COM file return the same ending address.

Media Check

Command Code = 1

MEDIA CHECK - ES:BX ->

13-BYTE Request Header
BYTE media descriptor from DPB
BYTE returned

In addition to setting the status word, the driver sets the return byte to one of the following:

-1 Media has been changed
0 Don't know if media has been changed
1 Media has not been changed

If the driver can return -1 or 1 because it has a door-lock or other interlock mechanism, MS-DOS performance is enhanced because MS-DOS does not need to reread the FAT for each directory access.

Build BPB (BIOS Parameter Block)

Command code = 2

BUILD BPB - ES:BX ->

13-BYTE Request Header
BYTE media descriptor from DPB
DWORD transfer address (Points to one sector worth of scratch space or first sector of FAT depending on the value of the NON PC-DOS FORMAT bit)
DWORD pointer to BPB

If the NON PC-DOS FORMAT bit is 1, then the DWORD transfer address points to a sector scratch buffer.

If the NON PC-DOS FORMAT bit is 0, then this buffer contains the first sector of the first FAT and the driver must not alter this buffer.

The first sector of the first FAT must be located in the same sector for all media. This is because the FAT sector is read BEFORE the media is actually determined. Use this mode to read the FAT ID byte.

In addition to setting status word, the driver must set the pointer to the BPB on return.

To allow different OEMs to read each other's disks, the information relating to the BPB for the media is kept in the boot sector of the media. The format of the boot sector is:

	3 BYTE near JUMP to boot code
	8 BYTES OEM name and version
BPB	WORD bytes per sector
BPB	BYTE sectors per allocation unit
BPB	WORD reserved sectors
BPB	BYTE number of FATs
BPB	WORD number of root dir entries
BPB	WORD number of sectors in logical image
BPB	BYTE media descriptor
BPB	WORD number of FAT sectors
	WORD sectors per track
	WORD number of heads
	WORD number of hidden sectors

Sectors per track, number of heads, and number of hidden sectors are optional. They are intended to help the BIOS understand the media. Sectors per track may be redundant since it can be calculated from total size of the disk. Number of heads supports different multi-head drives with the same storage capacity but a different number of surfaces. Number of hidden sectors supports drive-partitioning schemes.

Media Descriptor Byte

The last two digits of the FAT ID byte are called the media descriptor byte. Currently, the media descriptor byte has been defined for a few media types.

Bit	Meaning	
0	1 = 2 sided;	0 = not 2 sided
1	1 = 8 sector;	0 = not 8 sector
2	1 = removable;	0 = not removable
3-6	must be set to 1	
7	1 = not 80 track;	0 = 80 track

5 1/4" disks:

FEh	160K	
FCh	180K	
FFh	320K	formatted single sided
FDh	360K	formatted single sided
7Dh	720K	

HDU: F8h

Although these media bytes map directly to FAT ID bytes which must be F8-FF, media bytes can, in general, be any value in the range 0-FF.

Read or Write

Command codes = 3,4,8,9, and 12

READ or WRITE - ES:BX (Including IOCTL) ->

13-BYTE Request Header
BYTE media descriptor from DPB
DWORD transfer address
WORD byte/sector count
WORD starting sector number (Ignored on character devices)

In addition to setting the status word, the driver must set the sector count to the actual number of sectors (or bytes) transferred. No error check is performed on an IOCTL I/O call. The driver must set the return sector (byte) count to the actual number of bytes transferred.

A user program can not request an I/O of more than FFFFH bytes and cannot wrap around in the transfer segment.

Non Destructive Read No Wait

Command code = 5

NON DESTRUCTIVE READ NO WAIT -
ES:BX ->

13-BYTE Request Header
BYTE read from device

If the character device returns busy bit = 0 (characters in buffer), then the next character that would be read is returned. This character is not removed from the input buffer, hence the term Non Destructive Read. Basically, this call allows MS-DOS to look ahead one input character.

Status

Command codes = 6 and 10

STATUS Calls - ES:BX ->

13-BYTE Request Header

The driver sets the status word and the busy bit as follows:

- For output on character devices:
  If bit 9 is 1 on return, a write request waits for completion of a current request. If it is 0, there is no current request and a write request starts immediately.

- For input on character devices with a buffer:
  A return of 1 means a read request goes to the physical device. If it is 0 on return, then there are characters in the device's buffer and a read returns quickly. A return of 0 also indicates that the user has typed something. MS-DOS assumes that all character devices have an input type-ahead buffer. Devices that do not have a type-ahead buffer return busy = 0 so that the DOS does not wait for something to get into a non-existent buffer.

Flush

Command codes = 7 and 11

FLUSH Calls - ES:BX ->

13-BYTE Request Header

The FLUSH call tells the driver to terminate all pending requests. This call is used to flush the input queue on character devices.

Clock Device

One of the special enhancements for the Safari-3 is the battery-backed 58174A clock-calender chip and related driver. This chip is integrated into the system as the CLOCK device and is accessed with the TIME and DATE command.

This CLOCK device defines and performs functions like any other character device. When a read or write to this device occurs, exactly 6 bytes are transferred. The first two bytes are the count of days since 1-1-80. The third byte is minutes, the fourth, hours, the fifth, hundredths of seconds, and the sixth, seconds.

Reading the CLOCK device gets the date and time; writing to it sets the date and time.

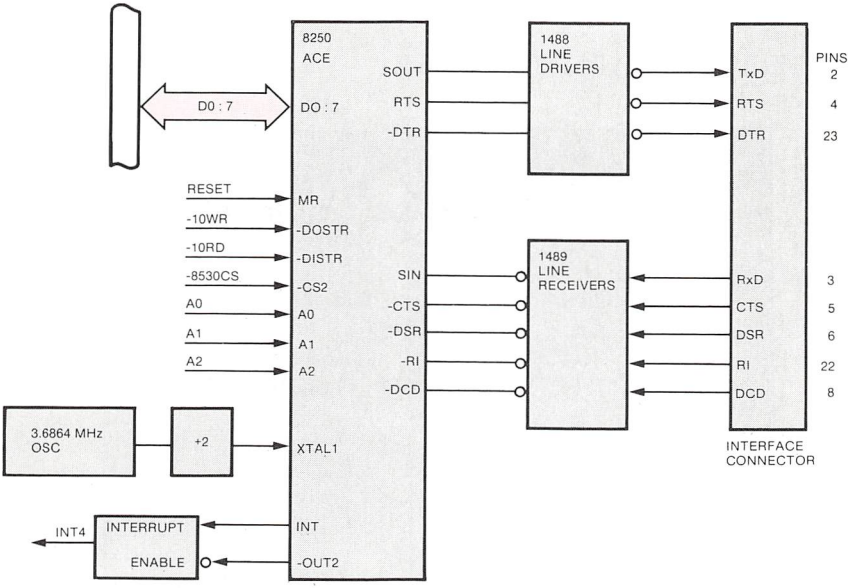
Asynchronous Communications Element

Functional Description

The Asynchronous Communications Element (ACE) performs serial-to-parallel conversion on input data characters received from a modem and parallel-to-serial conversion on output data characters received from the CPU. You can read the status of transfer operations at any time. This device gives you modem control capability.

The baud rate and serial interface characteristics are programmable. The ACE has a software-tailored interrupt system whose interrupt request is on INT4.

Block Diagram

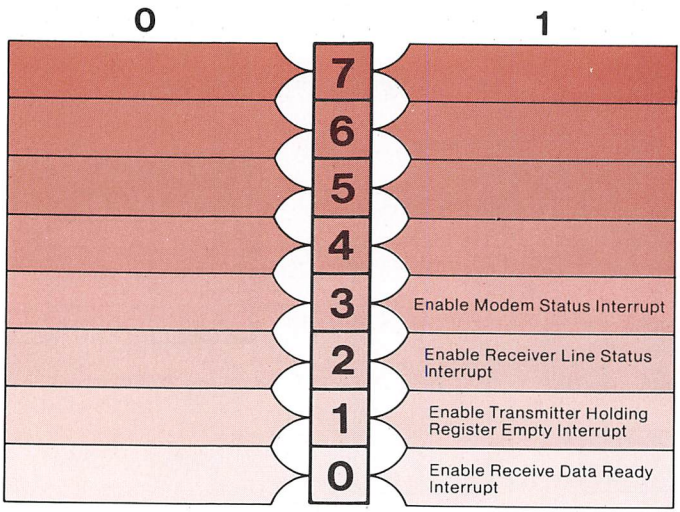


Registers

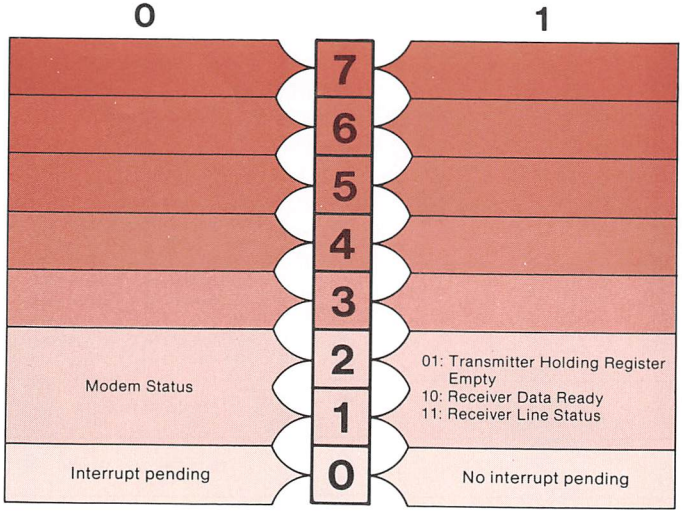
PORT A	PORT B	NAME	READ/ WRITE	DESCRIPTION
3F8	2F8	DATA	R/W W	Receive Buffer, Transmitter Holding Register
3F9	2F9	INTERRUPT ENABLE	W	See layout
3FA	2FA	INTERRUPT IDENTIFICATION	R	See layout
3FB	2FB	LINE CONTROL	R/W	See layout
3FC	2FC	MODEM CONTROL	R/W	See layout
3FD	2FD	LINE STATUS	R	See layout
3FE	2FE	MODEM STATUS	R/W	See layout
3FF	2FF	SCRATCH	R/W	Scratch pad register
3F8	2F8	DIVISOR LATCH	W	Least significant byte
3F9	2F9	DIVISOR LATCH	W	Most significant byte

Layout

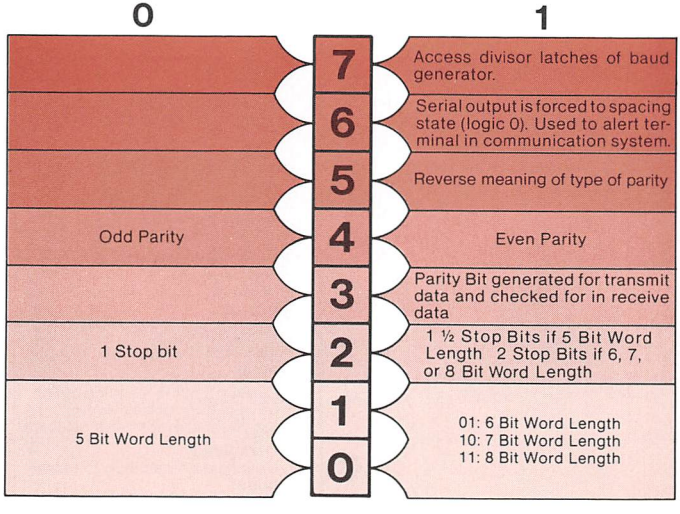
INTERRUPT ENABLE REGISTER



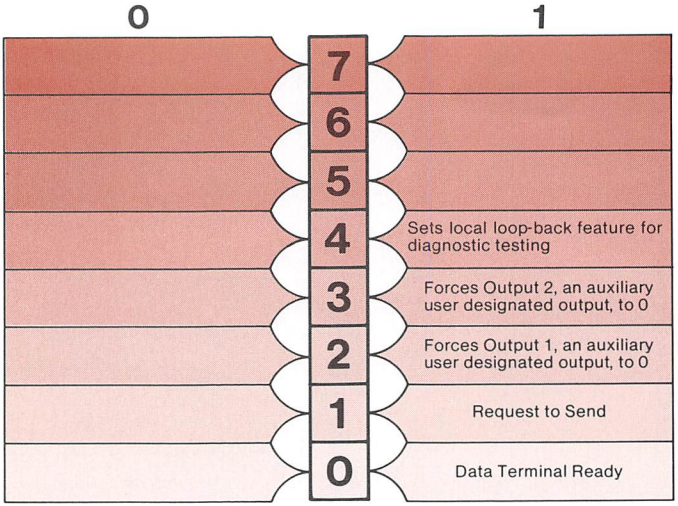
INTERRUPT IDENTIFICATION REGISTER



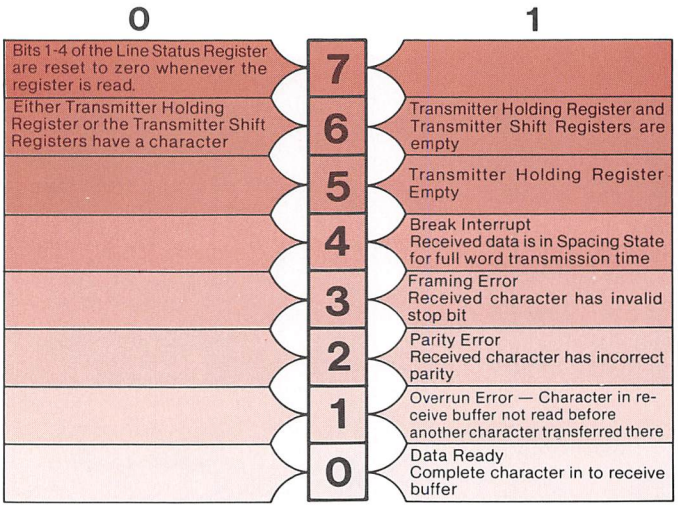
LINE CONTROL REGISTER



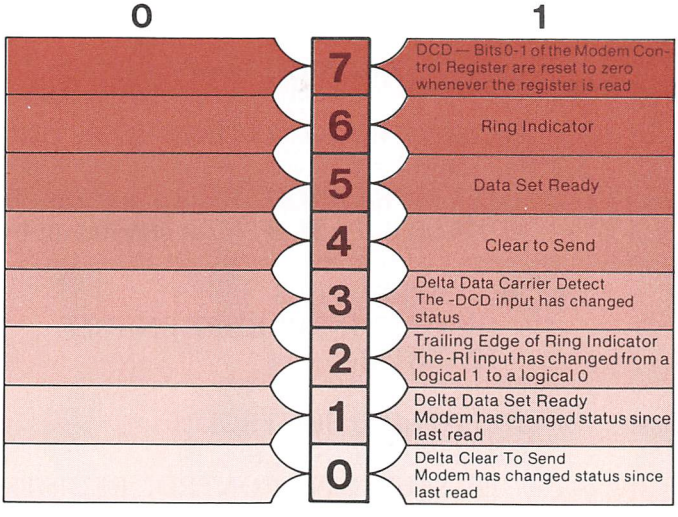
MODEM CONTROL REGISTER



LINE STATUS



MODEM STATUS REGISTER



Functions

- IDENTIFY INTERRUPTS
  INPUT: INTERRUPT IDENTIFICATION REGISTER
- READ LINE STATUS
  INPUT: LINE STATUS REGISTER
- READ MODEM STATUS
  INPUT: MODEM STATUS REGISTER
- RECEIVE CHARACTER
  INPUT: DATA
- SEND CHARACTER
  OUTPUT: DATA

- SET BAUD RATE
  To set the baud rate, you load the divisor which yields the correct rate (16 x divisor = clock frequency (1.8432 MHz)/baud rate x 16)

  OUTPUT: LINE CONTROL REGISTER
  BIT 7 = 1
  DIVISOR LATCH - Least significant byte
  DIVISOR LATCH - Most significant byte

- SET INTERRUPTS
  OUTPUT: INTERRUPT ENABLE REGISTER

- WRITE LINE CONTROL CHARACTERISTICS
  OUTPUT: LINE CONTROL REGISTER

- WRITE MODEM CONTROL CHARACTERISTICS
  OUTPUT: MODEM CONTROL REGISTER

Sequencing and Timing

To transmit a character, first issue a Request to Send and Data Terminal Ready to the Modem Control Register. Then wait for the Modem Status to have Data Set Ready and Clear to Send set. When the Transmitter Holding Register is empty as indicated in the Line Status Register, write the character to the data register.

To receive a character, set Data Terminal Ready in the Modem Control Register. Then wait for Data Set Ready in the Modem Status Register. When Data Ready in the Line Status Register is set, input the character from the data register.

The following table states the divisors to use to obtain a given baud rate.

BAUD RATES USING 1.8432 MHz CLOCK

BAUD RATE	DIVISOR
110	1047
150	768
300	384
600	192
1200	96
1800	64
2000	58
2400	48
3600	32
4800	24
7200	16
9600	12

If you wish to use the break control feature to alert a terminal in a communication system, the following sequence assures that no erroneous or extraneous characters are transmitted.

- Load an all 0's pad character into the transmitter holding register.
- Set break when the transmitter holding register is empty.
- Wait for transmitter empty (Bit 6 = 1 in Line Status Register) and clear break.

Note that the transmitter operates normally during a break sequence and can be used as a character timer to establish an accurate break length.

If the ACE is programmmed to interrupt, the interrupt is on INT4. The ACE acknowledges the highest priority interrupt as indicated in this chart. The Interrupt Identification Register states which interrupt is pending.

TYPE	SOURCE — RESET
Receiver Line Status	Overrun Error — Read Line Status Register
	Parity Error
	Framing Error
	Break Interrupt
Received Data Ready	Receive Data Ready — Read data
Transmitter Holding Register Empty	Transmitter Holding — Write data Register Empty
Modem Status	Clear To Send — Read Modem Status Register
	Data Set Ready
	Ring Indicator
	Data Carrier Detect

Sample Program

```
;This program sets the baud rate to 1200 baud
LINE_CTL  EQU 3FB
DVSR_L  EQU 3F8
DVSR_M  EQU 3F9

SET_BAUD:
   MOV AL,080H          ;access divisor
   MOV DX,LINE_CTL      ;latch
   OUT DX,AL

   MOV AX,096           ;divisor for 1200 baud
   MOV DX,DVSR_L
   OUT DX,AL            ;least significant byte

   MOV DX,DVSR_M
   MOV AL,AH
   OUT DX,AL            ;most significant byte

   XOR AL,AL            ;turn off access to latch
   MOV DX,LINE_CTL
   OUT DX,AL
   RET
```

DMA Controller

Functional Description

The DMA controller allows devices to transfer data directly to and from memory without CPU involvement. It has four channels.

Channel 0 has the highest priority and is used to refresh memory. The Interval Timer is programmed to periodically request a dummy DMA transfer. This creates a memory read cycle which refreshes memory.

Channel 1 is available on the I/O expansion bus to support high speed transfer between I/O devices and memory. Channel 3 has the lowest priority.

Channel 3 is dedicated to the hard disk controller. Channel 2 is dedicated to the floppy disk controller.

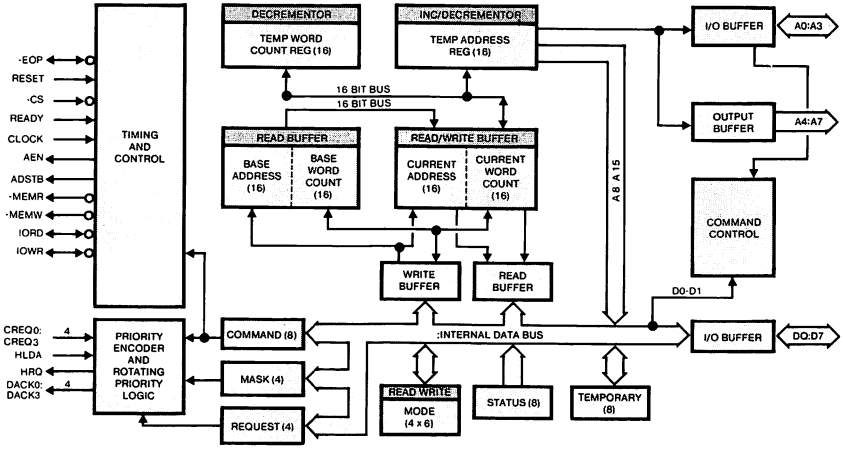
The DMA controller has four transfer modes. Single transfer mode makes only one transfer. Block transfer mode continues transferring until the count goes from 0 to FFFFH. Demand transfer allows transfers to continue until the I/O device has exhausted its capacity. The cascade mode allows more than one DMA controller to be used and is not applicable in the AT&T Personal Computer 6300.

When autoinitialize is requested, the original values of the Current Address and Current Count registers are restored at the end of the operation.

The DMA controller has two types of priority schemes. The fixed scheme bases the priority on the descending value of their numbers. In this scheme, Channel 3 has the lowest priority. The second scheme is rotating priority. The last channel to get service becomes the lowest priority channel.

Compressed Timing allows greater throughput by compressing the transfer time into two clock cycles.

Block Diagram



Registers

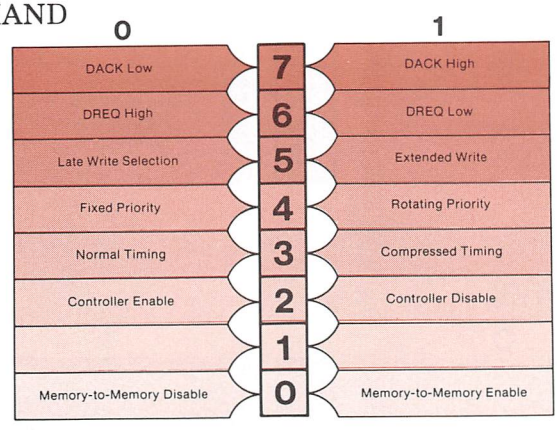
PORT	NAME	READ/ WRITE	DESCRIPTION
0	CHANNEL 0 ADDRESS	W	16 bit address
1	CHANNEL 0 COUNT	W	1's complement of # of bytes to transfer
0	CHANNEL 0 CURRENT ADDRESS	R	16 bit address
1	CHANNEL 0 CURRENT COUNT	R	1's complement of # of bytes to transfer
2	CHANNEL 1 ADDRESS	W	16 bit address
3	CHANNEL 1 COUNT	W	1's complement of # of bytes to transfer
2	CHANNEL 1 CURRENT ADDRESS	R	16 bit address
3	CHANNEL 1 CURRENT COUNT	R	1's complement of # of bytes to transfer
4	CHANNEL 2 ADDRESS	W	16 bit address
5	CHANNEL 2 COUNT	W	1's complement of # of bytes to transfer
4	CHANNEL 2 CURRENT ADDRESS	R	16 bit address
5	CHANNEL 2 CURRENT COUNT	R	1's complement of # of bytes to transfer
6	CHANNEL 3 ADDRESS	W	16 bit address
7	CHANNEL 3 COUNT	W	1's complement of # of bytes to transfer
6	CHANNEL 3 CURRENT ADDRESS	R	16 bit address
7	CHANNEL 3 CURRENT COUNT	R	1's complement of # of bytes to transfer
8	STATUS	R	See layout
8	COMMAND	W	See layout
9	REQUEST	W	See layout
A	SINGLE MASK	W	See layout
B	MODE	W	See layout
C	CLEAR FLIP/FLOP	W	Execute prior to read or write of address or count
D	TEMPORARY	R	Contains last byte of memory-to-memory transfer

Registers

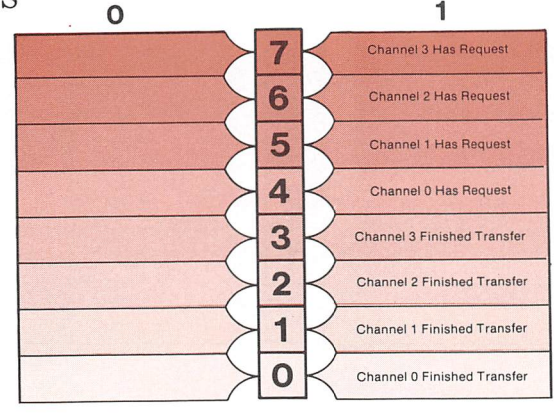
D	MASTER CLEAR	W	Any write clears controller
E	CLEAR MASK	W	Clear mask, all channels accept DMA commands
F	WRITE ALL MASK	W	See layout
80	CHANNEL 0 SEGMENT	W	Address segment nybble
82	CHANNEL 1 SEGMENT	W	Address segment nybble
81	CHANNEL 2 SEGMENT	W	Address segment nybble
83	CHANNEL 3 SEGMENT	W	Address segment nybble

Layout

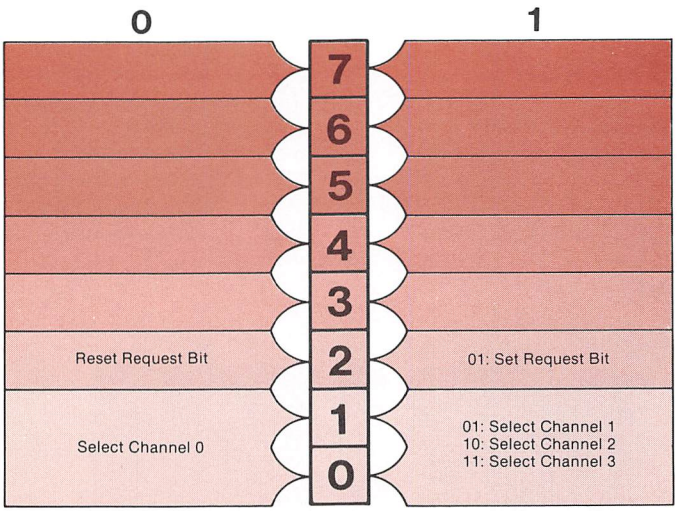
COMMAND



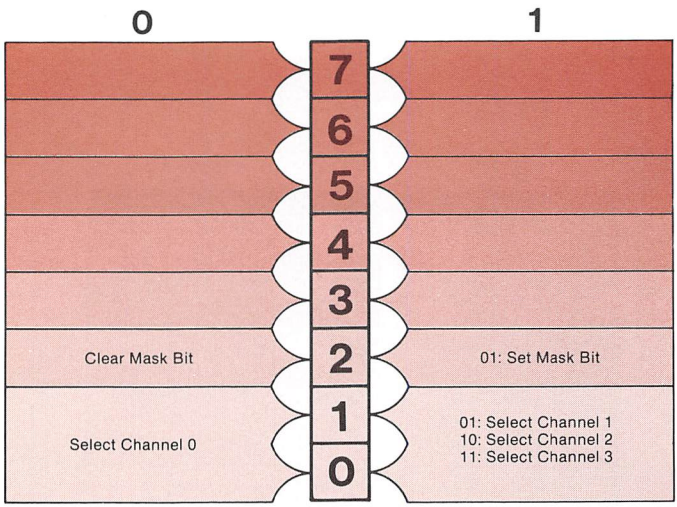
STATUS



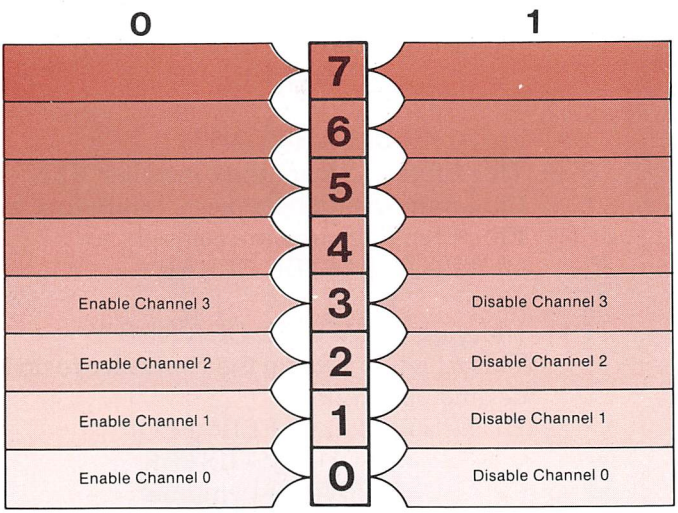
REQUEST REGISTER



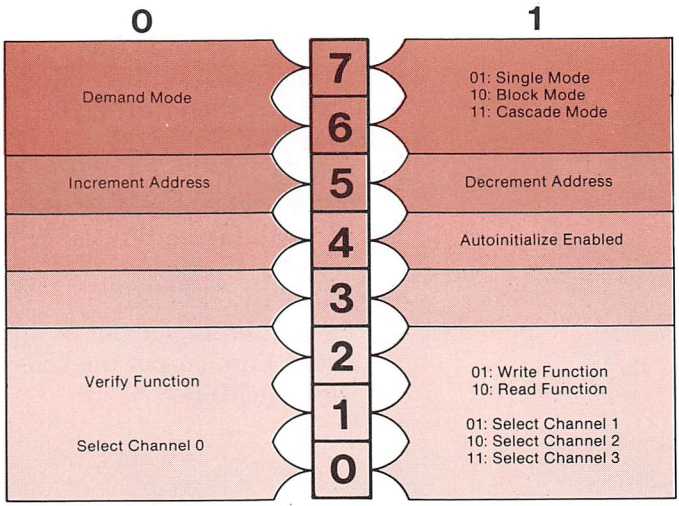
MARK REGISTER



WRITE ALL MASK REGISTER



MODE



Functions

- DISABLE CONTROLLER
  This function disables the controller.
  OUTPUT: COMMAND REGISTER
  BIT 2 = 1

- MASTER CLEAR CONTROLLER
  This function clears the controller. The Command, Status, Request, Temporary and Flip/Flop Registers are cleared.
  OUTPUT: MASTER CLEAR

- DMA READ, WRITE OR VERIFY
  This function sets up the controller to do the desired operation.
  OUTPUT: CLEAR FLIP/FLOP
  MODE REGISTER
  BITS 0-1 channel
  BITS 2-3 function
  BIT 4 autoinitialize
  BIT 5 address increment or decrement
  BITS 6-7 mode
  ADDRESS REGISTER
  16 BIT address, first output the LSB and then the MSB
  SEGMENT NYBBLE
  One nybble. The significant nybble of the segment register is in bits 12-15. Rotate to Bits 0-3 before output.
  COUNT REGISTER
  16 bit 1's complement of the # of bytes. First output the LSB and then the MSB.

- REQUEST DMA SERVICE
  This is a software request for DMA services.
  OUTPUT: REQUEST REGISTER

- READ STATUS
  This function reads the channel status.
  INPUT: STATUS REGISTER

- WRITE COMMAND REGISTER
  This function controls the operation of the DMA controller.
  OUTPUT: COMMAND REGISTER

- WRITE ALL MASK REGISTER
  This function enables and disables automatic DMA transfer for all channels.
  OUTPUT: WRITE ALL MASKS REGISTER

- WRITE MASK REGISTER
  This function enables and disables automatic DMA transfers for a channel.
  OUTPUT: MASK REGISTER

- WRITE MODE REGISTER
  This function specifies the mode for the specified channel.
  OUTPUT: MODE REGISTER

- CLEAR MASKS
  This function clears all the masks so that all channels accept DMA commands.
  OUTPUT: CLEAR MASK REGISTER

Sequencing and Timing

When the system is powered-up, it is recommended that all mode registers be set with valid data even if the channel is not used.

Before loading the address and count registers, disable the controller (BIT 2 of COMMAND REGISTER) or mask the channel. This prevents erroneous transfers before a complete address is loaded.

A write to the Clear Flip/Flop sets the controller so that an access to an address or count is to the upper and lower byte in the correct sequence.

Sample Program

```
;This program initializes an unused channel
;at start-up
COMMAND     EQU 8
MODE        EQU B
INIT_CHAN:
  MOV AL,04H          ;disable controller
  OUT COMMAND,AL
  MOV AL,041H         ;channel 1, verify, inc.
                      ;addr
  OUT MODE,AL         ;single mode
  RET                 ;now setup other
                      ;channels
```

Floppy Diskette Interface and Controller

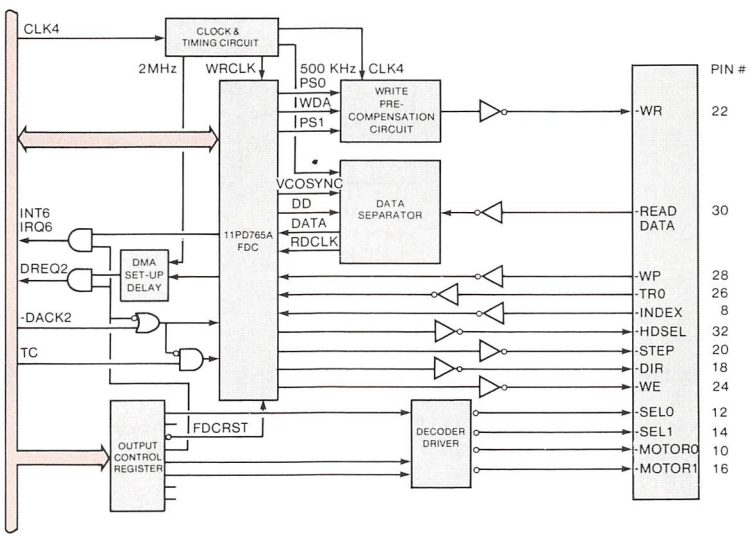
Functional Description

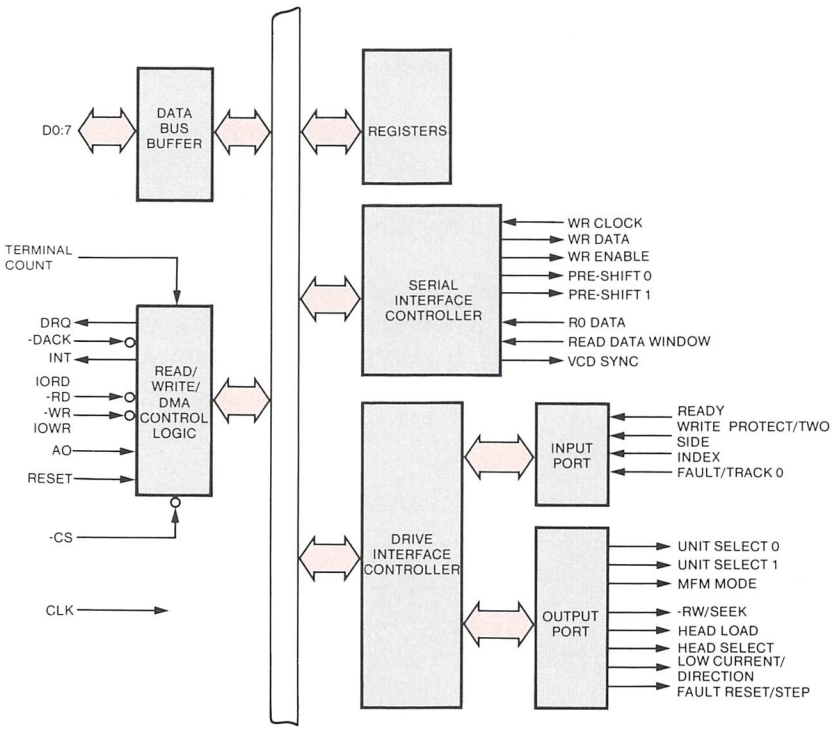
The diskette interface and NEC uPD765 controller read and write 5 1/4 inch diskettes on as many as two drives. Single density (FM) or double density (MFM) formats are supported. Single density diskettes contain 163,840 bytes and double density diskettes contain 327,680 bytes.

Each sector on the diskette contains an ID field and the Data field. The ID field contains the cylinder number, the head number, the sector number and the number of bytes per sector.

The diskette controller performs 15 separate functions. It operates in either DMA or non-DMA mode. Interrupts can be enabled on INT6.

Block Diagrams



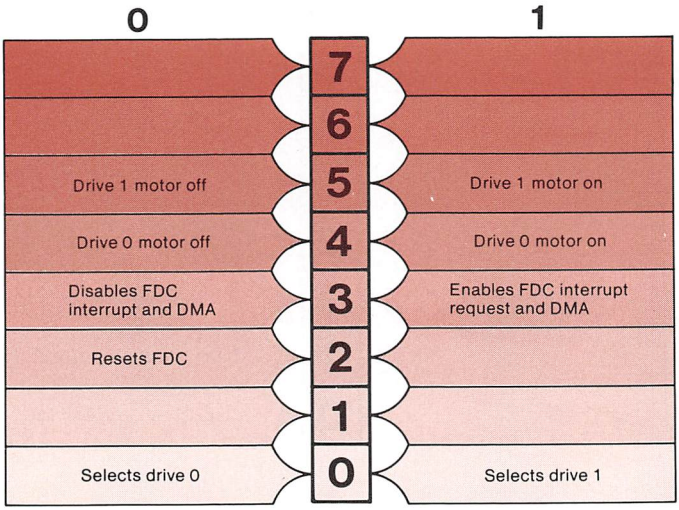


Registers

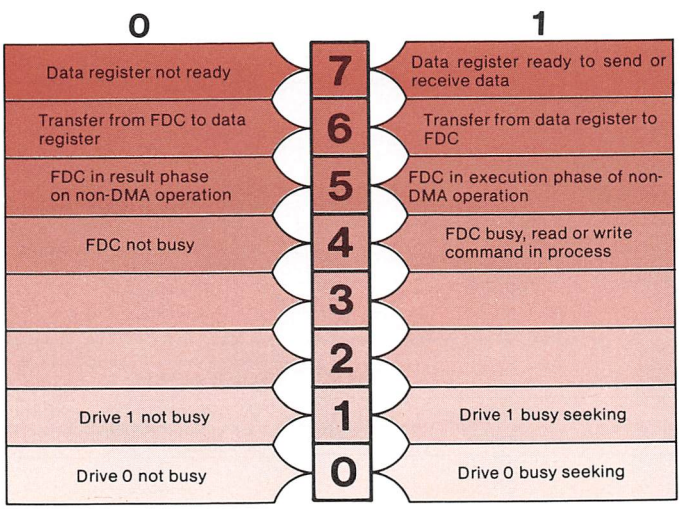
PORT	NAME	READ/ WRITE	DESCRIPTION
3F2	INTERFACE OUTPUT CONTROL	W	See layout
3F4	FDC MAIN STATUS REGISTER	R	See layout
3F5	FDC DATA	R/W	Transfers data, commands, parameters, and status

Layout

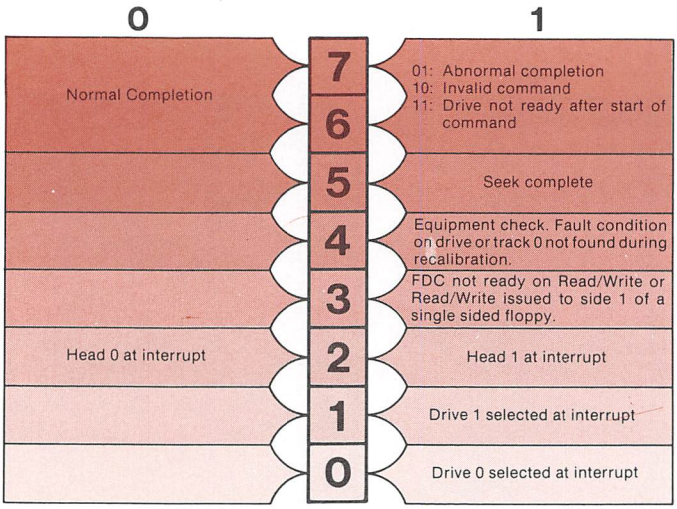
INTERFACE OUTPUT CONTROL



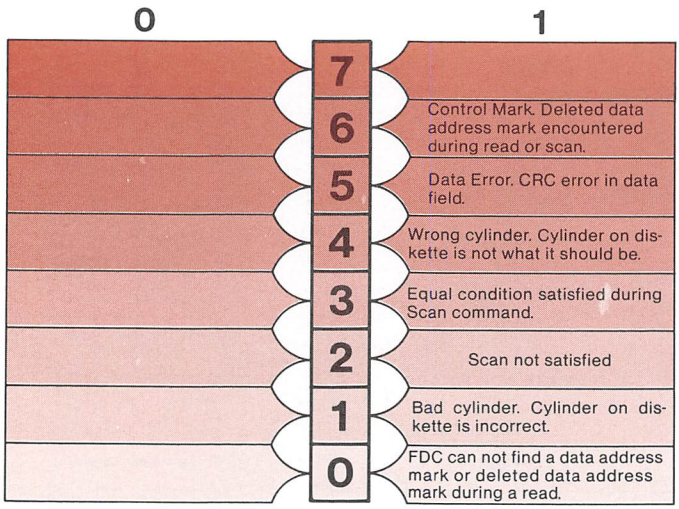
FDC MAIN STATUS REGISTER (MSR)



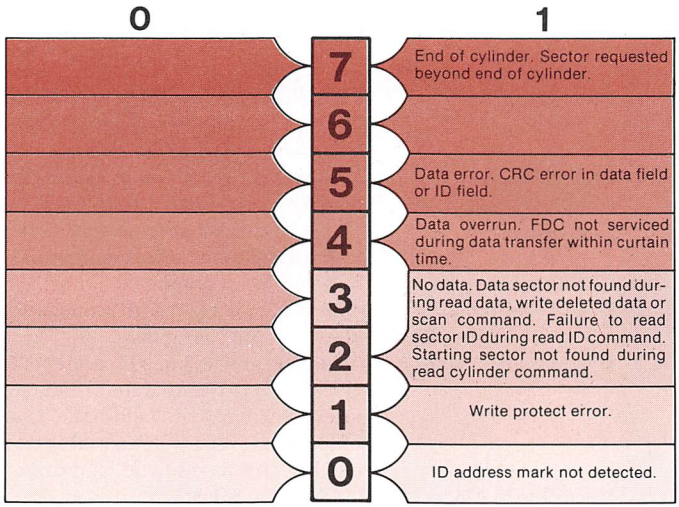
STATUS REGISTER 0



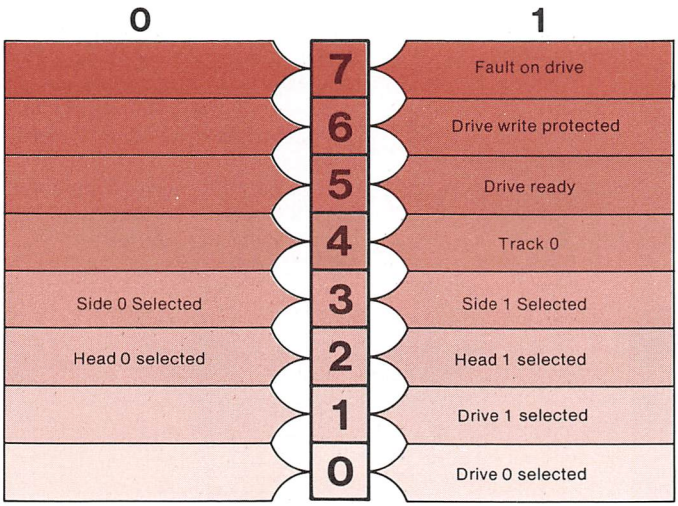
STATUS REGISTER 1



STATUS REGISTER 2



STATUS REGISTER 3



Parameters

SYMBOL	NAME	DESCRIPTION
DTL	Data Length	Only applies when there are 128 bytes per sector. If so, number of bytes to read or write. Otherwise, DTL = FF.
EOT	End of Track	Last sector number on cylinder. If there are 8 sectors per cylinder, then EOT = 8.
GPL	Gap Length	Gap 3 length between sectors. Different for format and read/write commands.
HD	Head	Selected Head number.
HLT	Head Load Time	4ms to 508ms in 4ms increments for 8Mhz clock. In this case, 4ms.
HUT	Head Unload Time	0ms to 480ms in 32ms increments for 8Mhz clock. This is the amount of time to wait after a read or write before the heads are unloaded. If a new command is issued quickly, this saves head load time.
MF	MF or MFM Mode	0 = MF, 1 = MFM
MT	Multi-track	MT = 1, multi-track operation. After completing an operation on side 0, the FDC continues on side 1.
N	Number of bytes/ sector	0 = 128 bytes, 1 = 256 bytes, 2 = 512 bytes 3 = 1024 bytes
ND	Non-DMA	0 = DMA mode, 1 = non-DMA mode
SRT	Step Rate Time	32ms to 2ms in 2 ms increments for 8Mhz clock. This is the amount of time to move the head from track to track. At 48 TPI, SRT = 6 ms. At 96 TPI, SRT = 4ms.
ST0-ST3	Status Registers	See layout
STP	Scan Test Flag	STP = 1, sector by sector compare STP = 2, alternate sectors
US0,US1	Unit Select	USx = 0, drive not selected USx = 1, drive selected

Functions

- FORMAT A TRACK
  This function formats an entire track. The ID Field for each sector is supplied by the programmer during the execution phase.

OUTPUT:

0	MF	0	0	1	1	0	1

0	0	0	0	0	HD	US1	US0

Number of bytes/sector
Sector/track
Gap Length
Filler Byte

EXECUTION: Sector ID Field transfer

INPUT: Status Register 0
Status Register 1
Status Register 2
Cylinder
Head
Sector
Number of bytes/sector

POSSIBLE ERRORS: Equipment check, and not ready

- READ DATA
  This function reads data from the diskette.

OUTPUT:

MT	MF	SK	0	0	1	1	0
0	0	0	0	0	HD	US1	US0

Cylinder
Head
Sector
Number of bytes/sector
End of Track
Gap Length
Data Length

EXECUTION: Data Transfer

INPUT:
Status Register 0
Status Register 1
Status Register 2
Cylinder
Head
Sector
Number of bytes/sector

POSSIBLE ERRORS: No data, data error, data error in data field, and control mark

- READ DELETED DATA
  This function reads deleted data.

OUTPUT:

MT	MF	SK	0	1	1	0	0
0	0	0	0	0	HD	US1	US0

Cylinder
Head
Sector
Number of bytes/sector
End of Track
Gap Length
Data Length

EXECUTION: Data Transfer

INPUT: Status Register 0
Status Register 1
Status Register 2
Cylinder
Head
Sector
Number of bytes/sector

- READ ID
  This function reads the first correct sector ID field.

OUTPUT:

0	MF	0	0	1	0	1	0
0	0	0	0	0	HD	US1	US0

EXECUTION:

INPUT: Status Register 0
Status Register 1
Status Register 2
Cylinder
Head

Sector
Number of bytes/sector

POSSIBLE ERRORS: Missing address mark and no data

- READ TRACK
  This function reads all data fields from the index hole to EOT. The FDC continues reading even if it finds a CRC error in the ID or data fields.

OUTPUT:

0	MF	SK	0	0	0	1	0
0	0	0	0	0	HD	US1	US0

Cylinder
Head
Sector
Number of bytes/sector
End of Track
Gap Length
Data Length

EXECUTION: Data transfer

INPUT: Status Register 0
Status Register 1
Status Register 2
Cylinder
Head
Sector
Number of bytes/sector

POSSIBLE ERRORS: No data, data error and missing address mark

- RECALIBRATE
  This function positions the head to head 0, cylinder or track 0.

OUTPUT:

0	0	0	0	0	1	1	1
0	0	0	0	0	0	US1	US0

EXECUTION: Head repositioned

POSSIBLE ERRORS: Equipment check

- SCAN EQUAL
  This function scans for an equal data compare.

OUTPUT:

MT	MF	SK	1	0	0	0	1
0	0	0	0	0	HD	US1	US0

Cylinder
Head
Sector
Number of bytes/sector
End of track
Gap Length
Contiguous or alternate sectors

EXECUTION: Data transfer

INPUT: Status Register 0
Status Register 1
Status Register 2
Cylinder
Head
Sector
Number of bytes/sector

- SCAN LOW OR EQUAL
  This function scans for a low or equal data compare.

OUTPUT:

MT	MF	SK	1	1	0	0	1
0	0	0	0	0	HD	US1	US0

Cylinder
Head
Sector
Number of bytes/sector
End of track
Gap Length
Contiguous or alternate sectors

EXECUTION: Data transfer

INPUT: Status Register 0
Status Register 1
Status Register 2
Cylinder
Head
Sector
Number of bytes/sector

- SCAN HIGH OR EQUAL
  This function scans for a high or equal data compare.

OUTPUT:

MT	MF	SK	1	1	1	0	1
0	0	0	0	0	HD	US1	US0

Cylinder
Head
Sector
Number of bytes/sector
End of track
Gap Length
Contiguous or alternate sectors

EXECUTION: Data transfer

INPUT: Status Register 0
Status Register 1
Status Register 2
Cylinder
Head
Sector
Number of bytes/sector

- SEEK
  This function positions the head at the requested cylinder.

OUTPUT:

0	0	0	0	1	1	1	1
0	0	0	0	0	HD	US1	US0

New cylinder number

EXECUTION: Heads repositioned

- SENSE DRIVE STATUS
  This function obtains the current drive status.

OUTPUT:

0	0	0	0	0	1	0	0
0	0	0	0	0	HD	US1	US0

INPUT: Status Register 0

- SPECIFY
  This function defines the drive parameters.

OUTPUT:

0	0	0	0	0	0	1	1
SRT				HUT			
HLT							ND

- WRITE DATA
  This function writes data.

OUTPUT:

MT	MF	0	0	0	1	0	1
0	0	0	0	0	HD	US1	US0

Cylinder
Head
Sector
Number of bytes/sector
End of Track
Gap Length
Data Length

EXECUTION: Data Transfer

INPUT: Status Register 0
Status Register 1
Status Register 2
Cylinder
Head
Sector
Number of bytes/sector

- WRITE DELETE DATA
  This function writes deleted data.

OUTPUT:

MT	MF	0	0	1	0	0	1
0	0	0	0	0	HD	US1	US0

Cylinder
Head
Sector
Number of bytes/sector
End of Track
Gap Length
Data Length

EXECUTION: Data Transfer

INPUT: Status Register 0
Status Register 1
Status Register 2
Cylinder
Head
Sector
Number of bytes/sector

Sequencing and Timing

There are three phases to each function:

- command - The programmer writes the required information to the FDC.
- execution - The FDC performs the operation.
- result - The programmer reads the FDC's status.

Before any data can be read or written to the FDC the Main Status Register (MSR) must be read to determine the status of Bit 6 and Bit 7. In the command phase, Bit 6 must be 0 and Bit 7 must be 1. In the result phase both bits must be 1. You must wait 12 usec after a data read or write before reading the MSR.

In the command phases, all output must be written. The same is true in the result stage. All status information must be read.

During the execution phase, the FDC operates in DMA mode or non-DMA mode. In DMA mode, there is one interrupt at the end of the phase. In non-DMA mode, there is an interrupt after the transfer of each byte. In the format command, the ID field information for all the sectors in a track is sent to the FDC (cylinder, head, sector and bytes/sector). In DMA mode, 4 DMA requests per sector are issued. In non-DMA mode, there are 4 interrupts per sector. If interrupts cannot be handled every 13 ms in MFM mode or every 27 ms in FM then the FDC is polled. When polling, Bit 7 in the MSR functions just like the interrupt.

When it is not executing a command, the FDC polls the drives looking for a change in drive ready. If there is a change, the FDC interrupts. You can determine the cause of the unexpected interrupt with the Sense Drive Status function.

The drive motors should be off when the drives are not in use. However, they must be on prior to a drive select.

During the execution phase of read and write commands, the following occurs:

- The heads are loaded if unloaded.
- The FDC waits for the head settle time to elapse.
- The FDC begins reading the ID address marks and ID field.
- When the requested sector number compares with the one on the diskette, the transfer begins.
- After completion of the transfer, the FDC waits the head unload time before unloading the heads.

The amount of data that can be transferred in one instruction depends on MT, MF, and N.

Multi-Track MT	MFM/FM MF	Bytes/Sector N	Maximum Transfer (Bytes/Sector)(Number of Sectors)
0	0	00	128*26 = 3,328
0	1	01	256*26 = 6,656
1	0	00	128*52 = 6,656
1	1	01	256*52 = 13,312
0	0	01	256*15 = 3,840
0	1	02	512*15 = 7,680
1	0	01	256*30 = 7,680
1	1	02	512*30 = 15,360
0	0	02	512*8 = 4,096
0	1	03	1024*8 = 8,192
1	0	02	512*16 = 8,192
1	1	03	1024*16 = 16,384

If a read or write terminates on error, then the values for cylinder, head, sector, and number of bytes per cylinder depends on the state of MT and EOT.

MT	HD	LAST SECTOR TRANSFERRED EOT	ID INFORMATION IN RESULTS C	H	S	N
0	0	Less than EOT	NC	NC	S+1	NC
0	0	Equal to EOT	C+1	NC	S=1	NC
0	1	Less than EOT	NC	NC	S+1	NC
0	1	Equal to EOT	C+1	NC	S=1	NC
1	0	Less than EOT	NC	NC	S+1	NC
1	0	Equal to EOT	NC	LSB	S=1	NC
1	1	Less than EOT	NC	NC	S+1	NC
1	1	Equal to EOT	C+1	LSB	S=1	NC

NC = No Change
LSB = Least Significant Bit

The Write Deleted Data is the same as Write Data except that the FDC writes a Deleted Data Address mark at the beginning of the Data Field instead of the normal Data Address Mark. When reading deleted data, the FDC sets the CM error in Status Register 2 and reads the data. A Read Data would not read the data. If SK = 1, then the FDC skips the sector with the Deleted Data Address mark and reads the next one.

The Gap Length is different for read, write, and format commands. This table suggests appropriate values.

FORMAT	SECTOR SIZE	N	SC	GPL(1)	GPL(2)
FM	128	00	12	07	09
FM	128	00	10	10	19
FM	256	01	08	18	30
FM	512	02	04	46	87
FM	1024	03	02	C8	FF
FM	2048	04	01	C8	FF
MFM	256	01	12	0A	0C
MFM	256	01	10	20	32
MFM	512	02	08	2A	50
MFM	1024	03	04	80	F0
MFM	2048	04	02	C8	FF
MFM	4096	05	01	C8	FF

GPL(1) - Suggested GPL in read and write commands
GPL(2) - Suggested GPL in format commands

The scan commands terminate when a scan condition is met, last sector on the track is reached, or a terminal count is received. The DMA issues the terminal count when it has no more data to send. This chart determines the result of the scan.

COMMAND	STATUS REGISTER 2 BIT 2	BIT 3	COMMENT
SCAN EQUAL	0	1	DISKETTE DATA = PROCESSOR DATA
SCAN EQUAL	1	0	DISKETTE DATA >< PROCESSOR DATA
SCAN LOW OR EQUAL	0	1	DISKETTE DATA = PROCESSOR DATA
SCAN LOW OR EQUAL	0	0	DISKETTE DATA < PROCESSOR DATA
SCAN LOW OR EQUAL	1	0	DISKETTE DATA > PROCESSOR DATA
SCAN HIGH OR EQUAL	0	1	DISKETTE DATA = PROCESSOR DATA
SCAN HIGH OR EQUAL	0	0	DISKETTE DATA > PROCESSOR DATA
SCAN HIGH OR EQUAL	1	0	DISKETTE DATA < PROCESSOR DATA

Scans allow the compare to be on contiguous sectors (STP = 1) or alternate sectors (STP = 2). However, for normal termination of the command the last sector on the track must be compared.

When a seek is requested, the FDC checks its current position and decides in which direction to move. Then step pulses are issued to move the heads. The speed of the pulse is controlled by the Step Rate Time in the Specify function. While the drive is seeking, the seek bit in the MSR is set. It must be cleared by Sense Interrupt Status at the completion interrupt. While a drive is seeking, the FDC is not busy. Another seek command to the other drive can be requested.

Interrupts occur as the result of:

1) Entering Result Phase of:
   Read Data
   Read Track
   Read Deleted Data
   Write Data
   Write Deleted Data
   Format Track
   Scans
2) The execution phase in non-DMA mode
3) The Drive Ready line changing state
4) The end of Seek or Recalibrate

When the latter two occur, a Sense Drive Status determines the cause of the interrupt. It is mandatory to follow Seek and Recalibrate functions with a Sense Drive Status. This chart shows how to interpret the results of a Sense Drive Status.

STATUS REGISTER 0			CAUSE
BIT 5	BIT 6	BIT 7	
0	1	1	Ready line changed state
1	0	0	Normal termination of Seek or Recalibrate
1	1	0	Abnormal termination of Seek or Recalibrate

The Specify command defines internal timers. Head Unload Time (HUT) is programmable from 32ms to 480ms in increments of 32ms. Therefore 1 = 32ms, 2 = 64ms, and F = 480ms. The Step Rate Time is programmable from 2ms to 32ms in increments of 2ms. In this case, F = 2ms, E = 4ms, and 1 = 32ms. The Head Load Time is programmable from 4ms to 508ms in increments of 4ms. In this case, 1 = 4ms, 2 = 8ms, 7F = 508ms.

Sample Program

```
;
; GET_RESULTS
;       This subroutine obtains a variable amount
;       of status information in the result phase.
; INPUT:  ES:DI points to the area that receives
;       the status bytes
;
NEC_STATUS      EQU 3F4

GET_RESULTS:
   MOV CX,7                     ;max. bytes in status

GET1:
   MOV DX, NEC_STATUS           ;port address of MSR
   IN AL,DX                     ;get MSR
   TEST AL,080H                 ;Data register ready to
                                ;send or receive
   JZ GET1                      ;jump if not ready yet
   TEST AL,40H                  ;direction bit
   JZ GET2                      ;jump if wrong
                                ;direction
   INC DX                       ;port addr of data
                                ;register
   IN AL,DX                     ;get one byte of status
   STOSB                        ;move it to status area
   DEC CX                       ;maximum number
   JNZ GET1                     ;jump if not max yet

GET2:RET
```

Hard Disk Controller

Functional Description

The DTC-5150BX hard disk controller reads and writes to a maximum of two standard 5 1/4" Winchester disk drives. A sector size of 256, 512, or 1024 bytes is selectable. The sectors can be interleaved in 16 different ways.

The hard disk controller operates in DMA or non-DMA mode. Interrupts can be enabled on INT5.

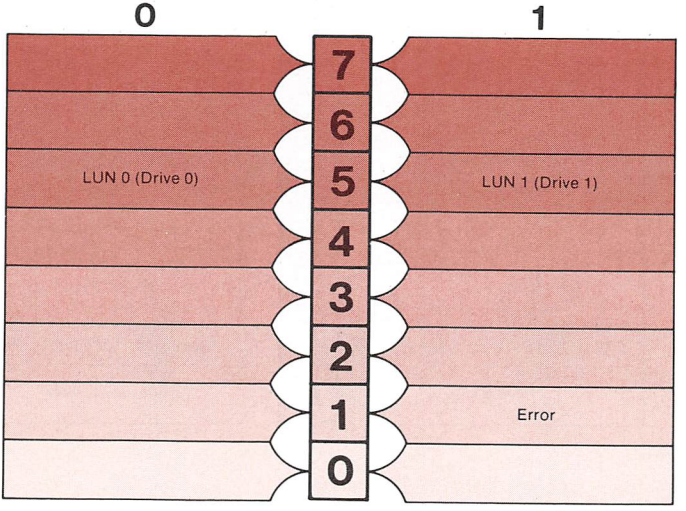
Extensive diagnostics are implemented. If a correctable data error is discovered, the error is automatically corrected using ECC.

Registers

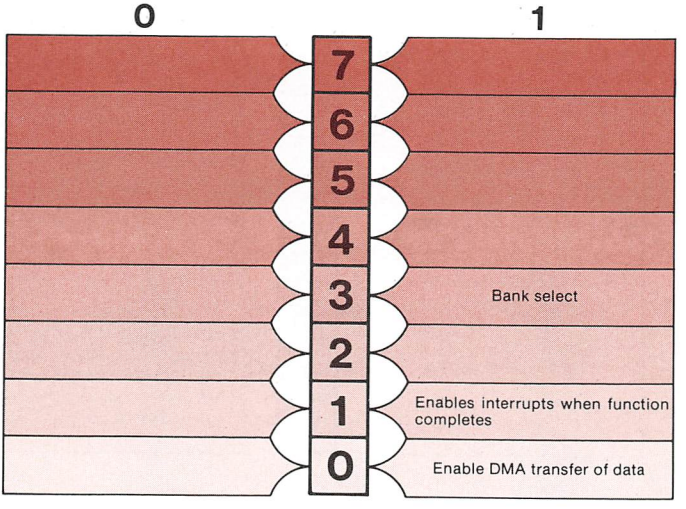
PORT #	NAME	READ/ WRITE	DESCRIPTION
320	COMPLETION STATUS REGISTER	R	See layout
320	DATA	R/W	Transfers data, function bytes, and controller sense bytes. See layout.
321	RESET CONTROLLER	W	Initialize Controller
321	STATUS	R	See layout
322	SELECT CONTROLLER	W	Select Controller
322	DRIVE TYPE	R	See layout
323	CONTROL REGISTER	W	See layout

Layout

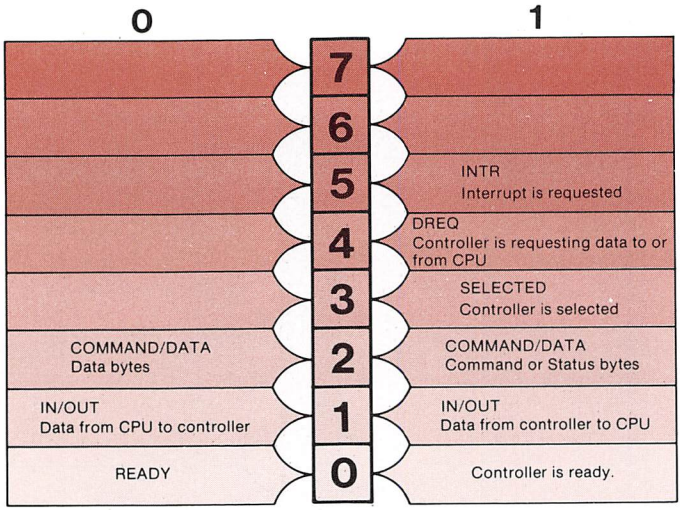
COMPLETION STATUS REGISTER



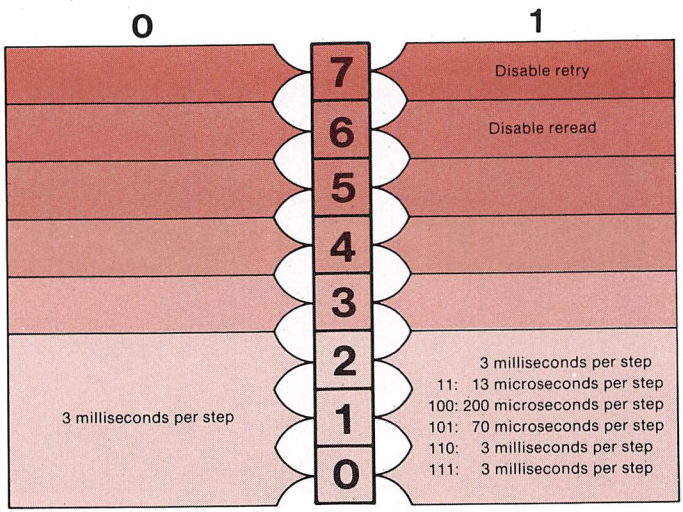
CONTROL REGISTER



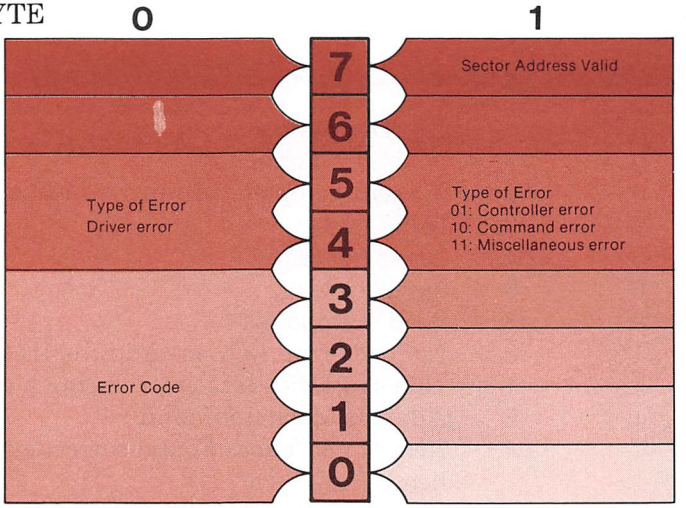
STATUS REGISTER



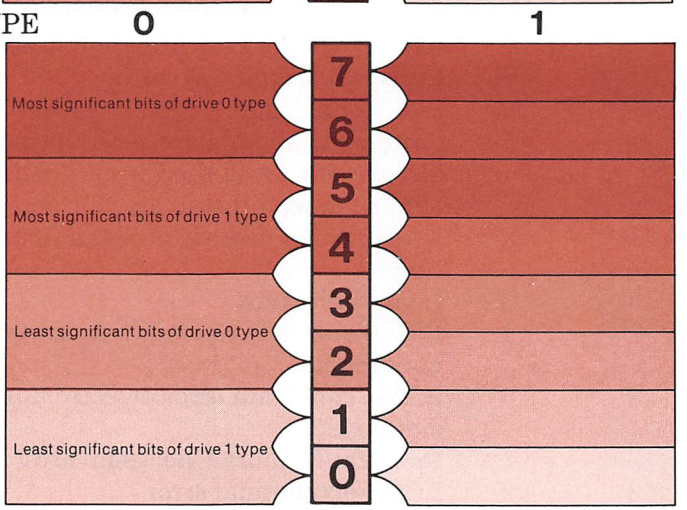
CONTROL COMMAND



SENSE BYTE



DRIVE TYPE



NUMBER	DRIVE TYPE
0	5 MB
1	24 MB
2	15 MB
3	10 MB
4	SQ306
5	CDC Wren

Error Codes

TYPE	CODE	DESCRIPTION
0	0	No error status
0	1	No index signal
0	2	No seek complete within 1.0 seconds
0	3	Write fault
0	4	Drive not ready
0	6	No track 0
0	8	Seek in progress
1	0	ID read error. ECC error in the ID field.
1	1	Uncorrectable data error during a read
1	2	Address Mark not found
1	4	Record not found. Found correct cylinder and head.
1	5	Seek error. Read/Write head positioned on wrong cylinder and/or wrong head selected.
1	8	Correctable data field error
1	9	Bad sector found
1	A	Format error. An unexpected format discovered during the Check Track function.
1	C	Unable to read the alternate track address
1	E	Attempted to directly access and alternate track
2	0	Invalid function
2	1	Illegal disk address. Address is beyond maximum.
3	0	RAM error. Data error detected during RAM diagnostic
3	1	Program Memory Checksum error
3	2	ECC Polynominal error

Function Parameters

NAME	DESCRIPTION
CONTROL COMMAND	Tells controller how to react to an error condition and defines step mode. See layout.
CYLINDER LOW	Eight least significant bits of the cylinder number
CYLINDER HIGH	Two most significant bits of the cylinder number
ECC0,ECC1,ECC2	ECC bytes of sector. ECC0 is least significant byte.
HEAD #	Head number
LUN #	Logical Unit Number. Winchester Drive 1 = Lun 0, Winchester Drive 2 = Lun 1.
SECTOR #	Sector number
SENSE BYTE	Gives detailed error information. See layout.
TYPE	0 = good track, 1 = alternate track, 2 = bad track, 3 = alternate bad track

Functions

- ASSIGN ALTERNATE TRACK
  This function formats the primary track specified in the function block with the alternated and bad track flags set in the ID fields and with the track address of the alternate track written in the data fields. The data field is written with the data in the sector buffer.

  Future read/write accesses to the primary track cause the drive to seek to the alternate track and to perform the operation there. This is transparent to the software. Alternate tracks can be assigned once. An alternate track cannot point to another alternate track.

OUTPUT:

7	6	5	4	3	2	1	0
11							
LUN			PRIMARY HEAD #				
CYL HI	0						
PRIMARY CYLINDER LOW							
INTERLEAVE							
CONTROL							

DATA
OUTPUT:

7	6	5	4	3	2	1	0
0	0	0	SECONDARY HEAD #				
CYL HI	0						
SECONDARY CYLINDER LOW							
0							

- CHECK TRACK
  This function checks the track format on the specified track for the correctness of the ID fields and the interleave of the sectors. It does not read the data.

OUTPUT:

7	6	5	4	3	2	1	0
							10
LUN				HEAD #			
CYL HI				0			
CYLINDER LOW							
INTERLEAVE							
CONTROL							

- CONTROLLER INTERNAL DIAGNOSTICS
  This function performs the controller internal diagnostics. The controller checks the internal processor, data buffer, ECC circuit and the checksum.

OUTPUT:

7	6	5	4	3	2	1	0
							E4
0							
0							
0							
0							
0							

- COPY
  This function transfers the data blocks from the source unit to the destination unit. The number of sectors to copy is specified in the number of blocks field. If the field is zero, 15,777,216 sectors are copied.

OUTPUT:

7	6	5	4	3	2	1	0
							A0
LUN/S				HEAD #/S			
CYL HI/S				SECTOR #/S			
CYLINDER LOW/S							
LUN/D				HEAD #/D			
CYL HI/D				SECTOR #/D			

CYLINDER LOW/D
OF BLOCKS 2
OF BLOCKS 1
OF BLOCKS 0
0
CONTROL

S = Source D = Destination

- DRIVE DIAGNOSTIC
  This function performs a diagnostic on the specified unit. It reads sector 0 on sequential tracks and then reads sector 0 on 256 random tracks.

OUTPUT:

7	6	5	4	3	2	1	0
							E3
LUN				0			
0							
0							
0							
CONTROL							

- FORMAT BAD TRACK
  This function formats the track with the bad block flag set in all ID fields. It fills the data field with the data pattern in the sector buffer. The interleave must be the same for the entire drive.

OUTPUT:

7	6	5	4	3	2	1	0
							7
LUN				HEAD #			
CYL HI				0			
CYLINDER LOW							
INTERLEAVE							
CONTROL							

- FORMAT TRACK
  This function formats the specified track with no flags set in the ID fields. It fills the data field with the data pattern in the sector buffer. The interleave must be the same for the entire drive.

OUTPUT:

7	6	5	4	3	2	1	0
6							
LUN				HEAD #			
CYL HI				0			
CYLINDER LOW							
INTERLEAVE							
CONTROL							

- FORMAT DRIVE
  This function formats all of the tracks starting with the one specified in the function block to the end of the drive. The selected track format is used. The sectors are placed on the tracks according to the interleave code. The data fields are filled with the data pattern from the sector buffer.

OUTPUT:

7	6	5	4	3	2	1	0
4							
LUN				HEAD #			
CYL HI				0			
CYLINDER LOW							
INTERLEAVE							
CONTROL							

- INITIALIZE DRIVE CHARACTERISTICS
  This function sets up the drive with different capacities and characteristics.

OUTPUT:

7	6	5	4	3	2	1	0
							C
LUN			0				
0							
0							
0							
0							

DATA OUTPUT:

MAX # OF CYLINDERS HIGH
MAX # OF CYLINDERS LOW
MAX # OF HEADS
REDUCED WR. CUR. CYLINDER HIGH
REDUCED WR. CUR. CYLINDER LOW
WRITE PRECOMP. CYLINDER HIGH
WRITE PRECOMP. CYLINDER LOW
MAX ECC DATA BURST LENGTH

- RAM DIAGNOSTIC
  This function performs a data pattern test on the controller RAM.

  OUTPUT:

7	6	5	4	3	2	1	0
							E0
				0			
				0			
				0			
				0			
				0			

- READ
  This function reads the specified number of blocks. The function specifies the initial sector address. The data is transferred to the CPU.

  OUTPUT:

7	6	5	4	3	2	1	0
							8
LUN				HEAD #			
CYL HI				SECTOR #			
CYLINDER LOW							
# OF BLOCKS							
CONTROL							

- READ ECC BURST ERROR LENGTH
  This function transfers one byte of data to the CPU. This byte contains the ECC burst length that the controller detected for the correctable ECC data error during the last read function.

  OUTPUT:

7	6	5	4	3	2	1	0
							D
0							
0							
0							
0							
0							

  DATA
  INPUT:

ECC BURST LENGTH

- READ ID
  This function reads three bytes and three ECC bytes from the specified sector address given in the function block and transfers them to the CPU.

OUTPUT:

7	6	5	4	3	2	1	0
							E2
LUN				HEAD #			
CYL HI				SECTOR #			
CYLINDER LOW							
INTERLEAVE							
CONTROL							

DATA
INPUT:

7	6	5	4	3	2	1	0
CYLINDER LOW							
TYPE	CYL HI				HEAD #		
SECTOR #							
ECC 2							
ECC 1							
ECC 0							

- READ LONG
  This function reads sectors of data and ECC bytes from the disk and transfers them to the CPU. If an ECC error occurs during the read, the controller does not attempt to correct the data.

  OUTPUT:

7	6	5	4	3	2	1	0
							E5
LUN				HEAD #			
CYL HI				SECTOR #			
CYLINDER LOW							
# OF BLOCKS							
CONTROL							

  DATA
  INPUT:

256/512/1024 BYTES OF DATA	E C 2	E C 1	E C 0	0 0

- READ SECTOR BUFFER
  This function reads one sector from the controller sector buffer. No data transfer occurs between the controller and the drives.

OUTPUT:

7	6	5	4	3	2	1	0
							E
0							
0							
0							
0							
0							

- READ VERIFY
  This function reads the specified number of blocks but does not transfer the data to the CPU. The function specifies the sector number where verification begins.

  OUTPUT:

7	6	5	4	3	2	1	0
							5
LUN				HEAD #			
CYL HI				SECTOR #			
CYLINDER LOW							
# OF BLOCKS							
CONTROL							

- RECALIBRATE
  This function positions the read/write arm at track 0 and clears errors in the drive.

OUTPUT:

7	6	5	4	3	2	1	0
							1
LUN			0				
0							
0							
0							
CONTROL							

- REQUEST LOGOUT
  This function retrieves the four bytes of error log for the specified unit. Each device has its own error log which is incremented every time certain errors occur and is cleared after this function is executed.

OUTPUT:

7	6	5	4	3	2	1	0
							E7
LUN			0				
							0
							0
							0
							0

DATA
INPUT:

RETRY COUNT HIGH
RETRY COUNT LOW
PERMANENT ERROR HIGH
PERMANENT ERROR LOW

- REQUEST SENSE
  This function sends the four Sense Bytes to the CPU as data.

OUTPUT:

7	6	5	4	3	2	1	0
							3
LUN				0			
							0
							0
							0
							0

DATA
INPUT:

7	6	5	4	3	2	1	0
SENSE BYTE							
LUN			HEAD #				
CYL HI			SECTOR #				
CYLINDER LOW							

- REQUEST SYNDROME
  This function returns the four bytes of the ECC syndrome to the CPU as data.

OUTPUT:

7	6	5	4	3	2	1	0
2							
LUN			0				
0							
0							
0							
0							

DATA
INPUT:

7	6	5	4	3	2	1	0
MSB bit offset							
LSB bit offset							
0							
0				MASK			

- SEEK
  This function seeks to the cylinder of the specified block. For Winchester drives capable of overlap seeks, this function returns completion status before the seek is complete.

  OUTPUT:

7	6	5	4	3	2	1	0
							B
LUN				HEAD #			
CYL HI				0			
CYLINDER LOW							
0							
CONTROL							

- TEST DRIVE READY
  This function selects the specified drive and verifies the drive is ready, the seek is complete, and there are no drive faults.

  OUTPUT:

7	6	5	4	3	2	1	0
							0
LUN				0			
							0
							0
							0
							0

- WRITE
  This function writes the data starting at the initial block address given in the function.

OUTPUT:

7	6	5	4	3	2	1	0
							A
LUN				HEAD #			
CYL HI				SECTOR #			
CYLINDER LOW							
# OF BLOCKS							
CONTROL							

- WRITE LONG
  This function writes blocks of data and ECC bytes from the CPU to the disk without generating ECC for the data.

OUTPUT:

7	6	5	4	3	2	1	0
							E6
LUN				HEAD #			
CYL HI				SECTOR #			
CYLINDER LOW							
# OF BLOCKS							
CONTROL							

DATA OUTPUT:

256/512/1024 BYTES OF DATA	ECC2	ECC1	ECC0	00

- WRITE SECTOR BUFFER
  This function writes one sector's worth of data to the controller sector buffer. No data transfer occurs between the controller and the drives.

  OUTPUT:

7	6	5	4	3	2	1	0
							F
							0
							0
							0
							0

Sequencing and Timing

There are three phases to each function:

- function initiation
- function execution
- function results

To initiate a function, you select the controller with the Select Controller Register. Then you wait for Ready in the Status Register to be set. The In/Out bit and the Command/Data bit should indicate function transfer to the controller. You then write six function bytes to the Data Register.

If the Ready is set after this transfer, either there was an error in the function bytes or the controller is ready to receive another group of six function bytes and/or data.

Data can be transferred in DMA or non-DMA mode. If the transfer is in DMA mode, the DMA Controller is programmed in Single Transfer mode (See DMA Controller). The count word is set to:

(number of sectors to transfer)(bytes/sector) - 1

If data is transferred in non-DMA, you use Ready, In/Out, Command/Data and Interrupt Request to time the transfer during execution.

Execution begins when the last function byte is received. In data transfer functions, the controller temporarily stores the data in the sector buffer. This prevents data overruns. When the function completes and the Completion Status byte is loaded, the controller issues interrupts if requested.

You clear the Interrupt Enable and the DMA enable bits in the Control Register after reading the Completion Status. This allows the controller to clear Interrupt Request and Data Request in the Status Register. It also clears the Selected bit.

The controller does extensive error recovery. If an error is found, four retries are attempted. If a retry is successful, the error is not reported; however, the retry count is incremented.

The following errors result in a retry:

- Seek error
- Sector not found
- Uncorrectable data error

- Correctable data error
- No data address mark
- No ID address mark
- ECC error in ID field

On a seek error, a recalibrate and reseek is done by the controller.

The following errors are accumulated in the log:

- ECC error in ID field
- Correctable error in data field
- Uncorrectable error in data field
- No ID address mark
- No data address mark
- Seek error
- Record not found

If rereads are disabled, the controller does not reread before applying the ECC correction.

When a reset is done, the controller defaults to the following characteristics:

- Maximum number of cylinders = 306
- Maximum number of heads = 4
- Starting reduced write current cylinder = 306
- Starting write precompensation cylinder = 0
- Maximum ECC data burst length = 4 bits

The interleave factor states how many physical sectors logical sectors are apart. For example, if the Interleave Factor is 6 and there are 16 sectors in a track, then a sector looks like this:

Physical Sector	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Logical Sector	0	3	6	9	12	15	1	4	7	10	13	16	2	5	8	11	14

The track layout for 256 bytes per sector, 33 sectors per track is:

13 bytes 00's	am	FE	cyl	hd	sec	ecc	00	00	13 bytes 00's	am	F8	256 bytes data	ecc	00	00	10 bytes 4E's

am, FE, cyl, hd, sec, 00, F8 = 1 byte
ecc = 3 bytes

Track capacity = 10416

```
   16 = Index Gap (4E)
10197 = 33 sectors @ 309 bytes/sector
  203 = Speed Tolerance Gap (4E)
-----
10416
```

309 bytes/sector including ID and overhead

The track layout for 512 bytes per sector, 17 sectors per track is:

13 bytes 00's	am	FE	cyl	hd	sec	ecc	00	00	13 bytes 00's	am	F8	512 bytes data	ecc	00	00	37 bytes 4E's

am, FE, cyl, hd, sec, 00, F8 = 1 byte
ecc = 3 bytes

Track capacity = 10416

```
    16 = Index Gap (4E)
 10064 = 17 sectors @ 592 bytes/sector
   336 = Speed Tolerance Gap (4E)
 10416
```

592 bytes/sector including ID and overhead

The track layout for 1024 bytes per sector, 9 sectors per track is:

13 bytes 00's	am	FE	cyl	hd	sec	ecc	00	00	13 bytes 00's	am	F8	1024 bytes data	ecc	00	00	58 bytes 4E's

am, FE, cyl, hd, sec, 00, F8 = 1 byte
ecc = 3 bytes

Track capacity = 10416

```
    16 = Index Gap (4E)
 10125 = 9 sectors @ 1125 bytes/sector
   275 = Speed Tolerance Gap (4E)
 10416
```

1125 bytes/sector including ID and overhead

Sample Program

```
; INIT_CTLR
;       This routine prepares the controller to
;       receive a function.
;           OUTPUT: Carry set if error
SELECT          EQU 322
STATUS          EQU 321
CONTROL         EQU 323
INIT-CTLR:
   MOV DX,SELECT        ;Select Port Address
   OUT DX,AL            ;Output anything
   MOV DX,CONTROL       ;Control Port Address
   MOV AL,3H            ;Enable interrupts and
                        ;DMA
   OUT DX,AL
   MOV DX,STATUS        ;Status Port Address
INIT1:
   IN AL,DX             ;Get status
   TEST AL,1H           ;Is it ready?
   LOOPZ INIT1          ;Loop if not ready
   CMP AL,DH            ;Jump if ready for a
   JE INIT2             ;function and selected
   STC                  ;Flag error
   RET
INIT2:
   CLC
   RET
```

Keyboard Interface

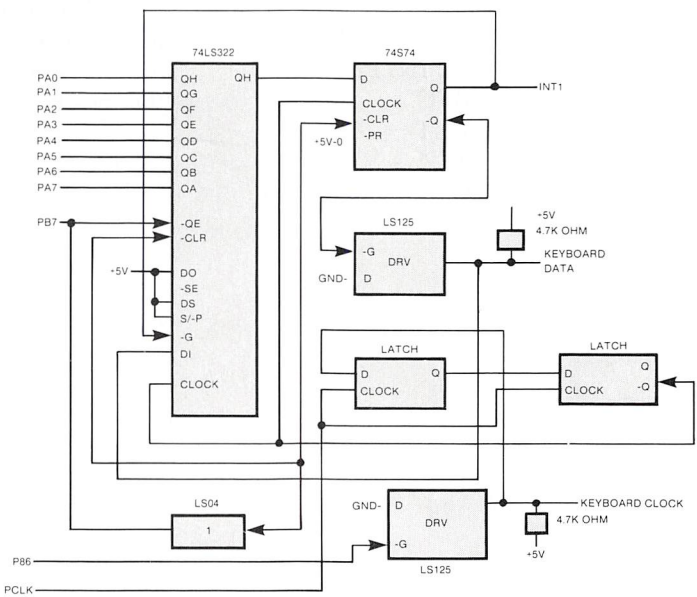
Functional Description

The keyboard interface converts the parallel data into serial data for transmission to and from the keyboard.

To provide maximum flexibility in defining keyboard operations, the keyboard uses scan codes rather than ASCII codes. In addition, all keys generate a make scan code when pressed and a break scan code when released. The break scan code is 80H plus the make scan code.

The keyboard is responsible for keeping track of the amount of time a key is depressed and for generating the repeat key signal. All the keys have this repeat function.

Block Diagram

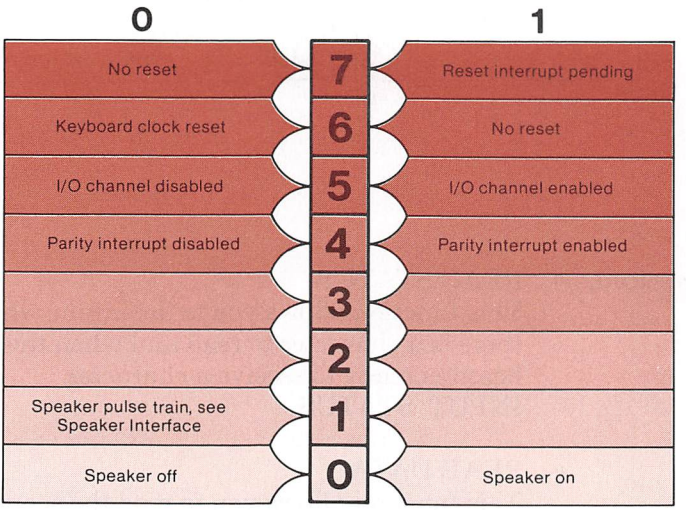


Registers

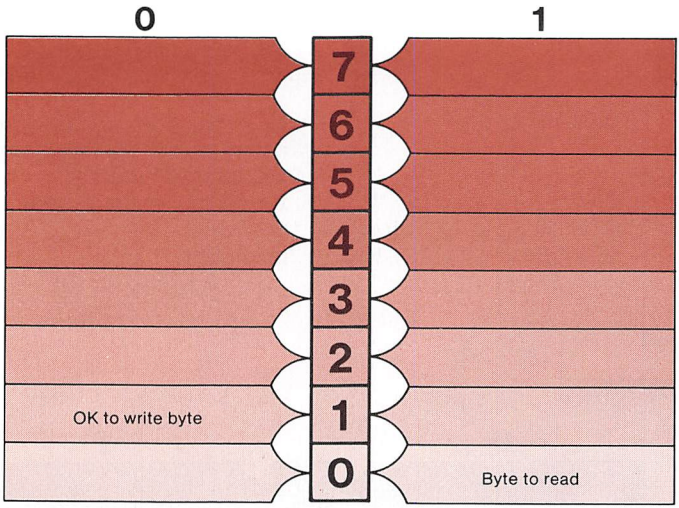
PORT	NAME	READ/ WRITE	DESCRIPTION
60	DATA	R	8 bit scan code when pressed, 8 bit scan code plus 80H when released See scan code chart.
60	DATA	W	Keyboard command codes
60	CONTROL	R/W	See layout
64	STATUS	R	See layout

Layout

CONTROL



STATUS



Functions

- READ STATUS
  This function allows you to determine when there is a character to read and when the keyboard is ready to receive a character.
  INPUT: STATUS

- READ DATA
  This function allows you to read the scan code of the key that was pressed or released.
  INPUT: DATA

- WRITE CONTROL
  This function allows you to issue control commands to the keyboard.
  OUTPUT: CONTROL

- GET KEYBOARD TYPE
  This function determines if the keyboard is an Olivetti M20 type keyboard.
  OUTPUT: DATA REGISTER
  5
  INPUT: DATA REGISTER
  1 = Olivetti M20 keyboard

- KEYBOARD LEDS
  This function controls the illuminations of the CAPS LOCK and NUM LOCK LEDS.
  OUTPUT: DATA REGISTER
  13H

Value	Operation
00h	No operation
01h	Cap lock LED OFF
02h	Num lock LED OFF
03h	Both LEDS OFF
80h	No operation
81h	Caps lock LED ON
82h	Num lock LED ON
83h	Both LEDS ON

Any other value will cause one of the above operations.

Scan Codes

KEY NO	SCAN CODE	KEY NO	SCAN CODE	KEY NO	SCAN CODE
1	01H	36	24H	71	47H
2	02H	37	25H	72	48H
3	03H	38	26H	73	49H
4	04H	39	27H	74	4AH
5	05H	40	28H	75	4BH
6	06H	41	29H	76	4CH
7	07H	42	2AH	77	4DH
8	08H	43	2BH	78	4EH
9	09H	44	2CH	79	4FH
10	0AH	45	2DH	80	50H
11	0BH	46	2EH	81	51H
12	0CH	47	2FH	82	52H
13	0DH	48	30H	83	53H
14	0EH	49	31H	84	54H
15	0FH	50	32H	85	55H
16	10H	51	33H	86	56H
17	11H	52	34H	87	57H
18	12H	53	35H	88	58H
19	13H	54	36H	89	59H
20	14H	55	37H	90	5AH
21	15H	56	38H	91	5BH
22	16H	57	39H	92	5CH
23	17H	58	3AH	93	5DH
24	18H	59	3BH	94	5EH
25	19H	60	3CH	95	5FH
26	1AH	61	3DH	96	60H
27	1BH	62	3EH	97	61H
28	1CH	63	3FH	98	62H
29	1DH	64	40H	99	63H
30	1EH	65	41H	100	64H
31	1FH	66	42H	101	65H
32	20H	67	43H	102	66H
33	21H	68	44H	103	67H
34	22H	69	45H	104	68H
35	23H	70	46H	-	-

Sequencing and Timing

To send a character to the keyboard, wait for Bit 1 of the Status Register to be set and write the byte to the data register.

To receive a character, wait for Bit 0 of the Status Register to be set and read the character. The keyboard interface can be programmed to interrupt on INT1 when there is a character to read.

Sample Program

```
;This program sets the CAPS LOCK LED
STATUS      EQU 64
DATA        EQU 60
SET_LED:
   IN AL,STATUS      ;read status
   TEST AL,2         ;keyboard ready to receive
   JNZ SET_LED       ;input? Jump if ready.
   MOV AL,013H       ;keyboard LED command
   OUT DATA,AL

SET1:
   IN AL,STATUS      ;read status
   TEST AL,2         ;ready to receive input?
   JNZ SET1          ;jump if not
   MOV AL,81         ;CAPS LOCK
   OUT DATA,AL       ;write out code
   RET
```

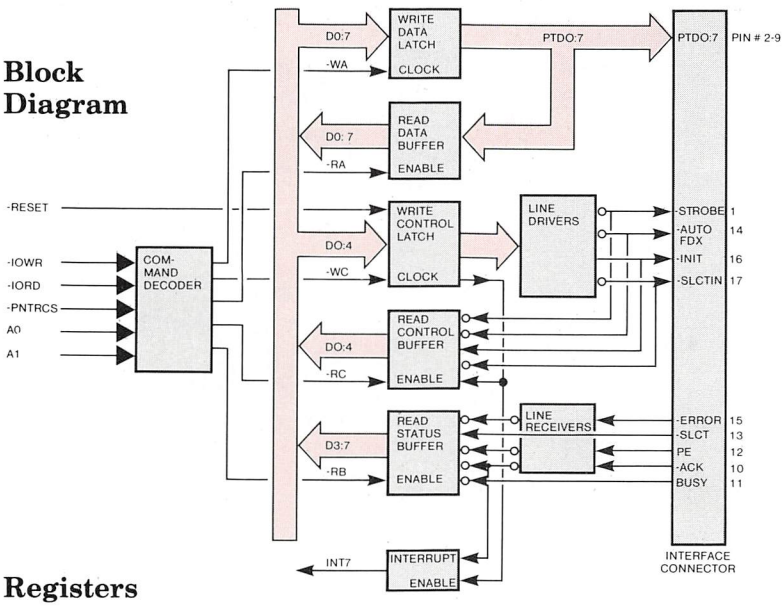
Parallel Printer Interface

Functional Description

The parallel printer interface connects to printers with a Centronics-like parallel interface or any other device with identical interface characteristics. The input and output signals are presented to the external device through a 25-pin "D" type connector.

The interface has 5 buffered outputs — data, strobe, initialize printer, automatic linefeed, and select. These can be read and written. In addition, the interface has five inputs — acknowledge, busy, paper out, error and select. An interrupt can be enabled on INT7.

Block Diagram

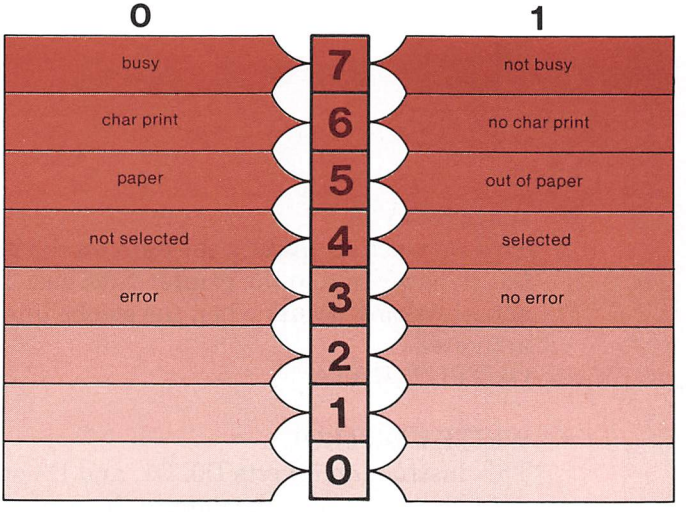


Registers

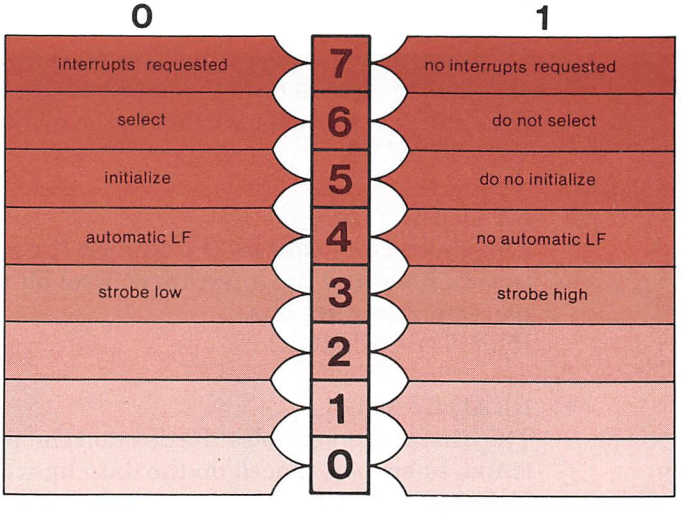
PORT #	NAME	READ/ WRITE	DESCRIPTION
378	DATA	R/W	Print character
379	STATUS	R	See layout
37A	CONTROL	R/W	See layout

Layout

STATUS



CONTROL



Functions

- RESET
  After a power on or a hardware reset, the data buffer is cleared and the control register is initialized to:

  bit 0: 0
  bit 1: 0
  bit 2: 0
  bit 3: 0
  bit 4: 0

- WRITE DATA
  This instruction enables the data on the data bus to be written to the printer data bus. The actual writing occurs when the strobe line is activated.
  OUTPUT: DATA

- WRITE CONTROL
  This instruction inverts D0, D1, and D3 on the data bus and writes the data to the control register. If D4 is a 1 then interrupts are requested.
  OUTPUT: CONTROL

- READ DATA
  This instruction enables the data on the printer data bus to be read onto the data bus. It normally is the last character written to the printer.
  INPUT: DATA

- READ CONTROL
  This instruction enables the data on the printer control lines and the interrupt control bit to be placed on the data bus.
  INPUT: CONTROL

- READ STATUS
  This instruction enables the data on the printer status lines to be placed on the data bus.
  INPUT: STATUS

Sequencing and Timing

To send a character to the printer, the character is put on the data bus. When the printer is not busy, it is ready to accept the next character. The character must be strobed into the printer by setting the strobe bit to 1 for at least 5 μ-seconds and then resetting it.

Interrupt can be enabled on INT7 by writing D4 = 1 in the control register. An interrupt will be triggered everytime Bit 6 of the status register goes from 0 to 1 (end of the acknowledge cycle from the printer).

To initialize the printer, first select it. Then issue the initialize command setting the automatic line feed and interrupt parameters.

Sample Program

```
; PRINT_CHAR
;       Send character to printer and get status
;       INPUT  AL - character to print
;       OUTPUT:  AL - status
;
DATA                EQU 378H
PRINT_CHAR:
   MOV DX,DATA              ;get data port
   OUT DX,AL                ;put char on data
                            ;line
   INC DX                   ;get status port
   IN AL,DX                 ;read in status
   TEST AL,080H             ;is the printer busy?
   JNZ PRINT_NOT_BUSY       ;jump if not busy
   .
   .
   .
PRINT_NOT_BUSY:
   MOV AL,0DH               ;strobe high
   INC DX                   ;get control port
   OUT DX,AL                ;to control register
   NOP                      ;wait
   NOP                      ;wait
   MOV AL,0CH               ;strobe low
   OUT DX,AL                ;to control register
   DEC DX                   ;get status port
   IN AL,DX                 ;read in status
   RET
```

Programmable Interrupt Controller

Functional Description

The Intel 8259A Programmable Interrupt Controller (PIC) manages external interrupts. It receives requests from peripheral equipment, decides which request has the highest priority, and issues an interrupt to the CPU. Each PIC handles 8 maskable priority interrupts. PIC's can be cascaded allowing up to 64 priority interrupts. However, on the AT&T Personal Computer 6300 this is not done.

Each interrupt device runs to one of eight interrupt lines (INT0-INT7). If more than one device interrupts at once, the PIC decides which device to service according to one of several schemes.

The following schemes apply to a single PIC:

- Fully Nested Mode
  This is the default mode. The interrupt requests have an ordered priority from 0 (highest) to 7 (lowest). The highest priority is acknowledged first and those of lower priority are inhibited.

- Special Mask Mode
  This mode is similar to fully nested mode except that the interrupt mask register (IMR) determines which interrupts are disabled.

- Polled
  This mode allows the CPU to poll the devices. It is selected by disabling interrupts with the CLI instruction. Periodically the CPU polls the PIC to receive the interrupt type of the highest priority device requesting service.

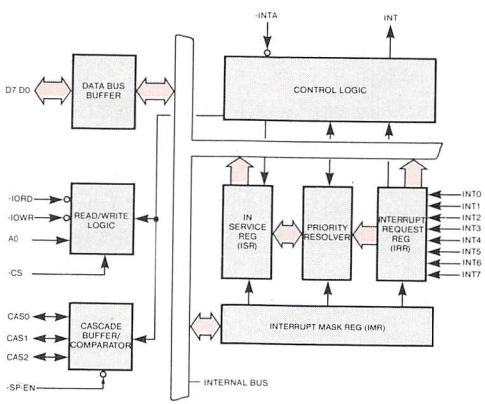
- Automatic Rotation
  In this mode, a device receives the lowest priority after it is serviced. All other devices have their priorities adjusted accordingly. The next highest interrupt line receives the highest priority.

- Programmable Rotation
  In this mode, the programmer declares the lowest priority device.

The PIC keeps track of devices that are waiting for service in the interrupt request register (IRR). If not in polled mode, the PIC notifies the CPU of the pending interrupt. When it receives an interrupt acknowlege from the CPU (INTA), it sends the interrupt type of the device to the CPU. The device is then in-service. This is noted in the in-service register (ISR). The type of INT0 is programmable. It must be a multiple of 8.

After the interrupt service routine services the interrupt, it notifies the PIC of end of interrupt (EOI). The device is then removed from the in-service register.

Block Diagram



- ROTATION OF PRIORITIES INDEPENDENT OF EOI
  This function requests an immediate change in priorities.
  OUTPUT: OCW2 BITS 5-7 = 6
  OCW2 BITS 0-2 = level of lowest priority

- SPECIFIC END OF INTERRUPT (SEOI)
  This function is issued in an interrupt service routine to declare end of interrupt service for the specified level.
  OUTPUT: OCW2 BITS 5-7 = 3
  BITS 0-2 = interrupt level

- NON-SPECIFIC END OF INTERRUPT (EOI)
  When the PIC is operating in Fully Nested Mode, it can determine which interrupt is completing. This function signals completion of an interrupt service routine.
  OUTPUT: OCW2 BITS 5-7 = 1

- AUTOMATIC END OF INTERRUPT
  This function requests the PIC to declare end of interrupt automatically after delivering the interrupt to the CPU.
  OUTPUT: ICW4 BIT 1 = 1

- READ IRR
  This function reads the devices requesting service.
  OUTPUT: OCW3 BITS 0-1 = 2
  INPUT: IRR

- READ ISR
  This function reads the in-service register.
  OUTPUT: OCW3 BITS 0-1 = 3
  INPUT: ISR

- READ IMR
  This function reads the interrupt mask register.
  INPUT: OCW1

Sequencing and Timing

When the ICW1 command is issued, the initialization process begins. The following automatically occurs:

- IMR is cleared.
- INT7 is assigned the lowest priority.
- Single mode is assumed.
- Special Mask Mode is cleared.
- A status read fetches IRR.
- If Bit 0 equals zero, then ICW4 functions are set to zero.

  Next ICW2 is output. ICW3 is skipped in all single PIC systems. If the ICW4 was requested by ICW1 then it is output. This completes initialization.

  Once the initialization process is complete, the PIC is ready to accept interrupts. Any of the functions to change the priority scheme can be executed. In addition, the IRR, IMR, and ISR can be read.

  If automatic EOI is not specified then the interrupt service routine must declare EOI. Either specific or non-specific EOI can be used depending on the priority scheme in use.

When a write command is issued to the PIC, 480 nanoseconds must elapse before another command is issued. In the read case, 395 nanoseconds must elapse.

Sample Program

```
;
; SEND_SEOI
;       Send end of INT1 service at the end of the
;       interrupt service routine.
;
OCW2            EQU 20   ;port address of OCW2
COMMAND         EQU 61   ;6 = SEOI, 1 = INT1

SEND_SEOI:
   MOV AL,COMMAND        ;set AL = command
   MOV DX,OCW2           ;set DX = port address
   OUT DX,AL             ;send command
   STI                   ;enable external interrupts
   RET                   ;return to service routine
```

Programmable Interval Timer

Functional Description

The Intel 8253 Interval Timer has three identical, 16-bit, settable, decrementing counters. Each counter is totally independent. The counters have either a BCD or binary value and operate in one of five modes:

- Interrupt on Terminal Count
  The output remains low after the mode is set. It continues low after the counter is loaded until the counter counts down to zero. Then the output goes high. It remains high until a new mode is selected or a new count is loaded.

- Programmable One-Shot
  The output goes low one count following the rising edge of the gate input. The output goes high on the terminal count.

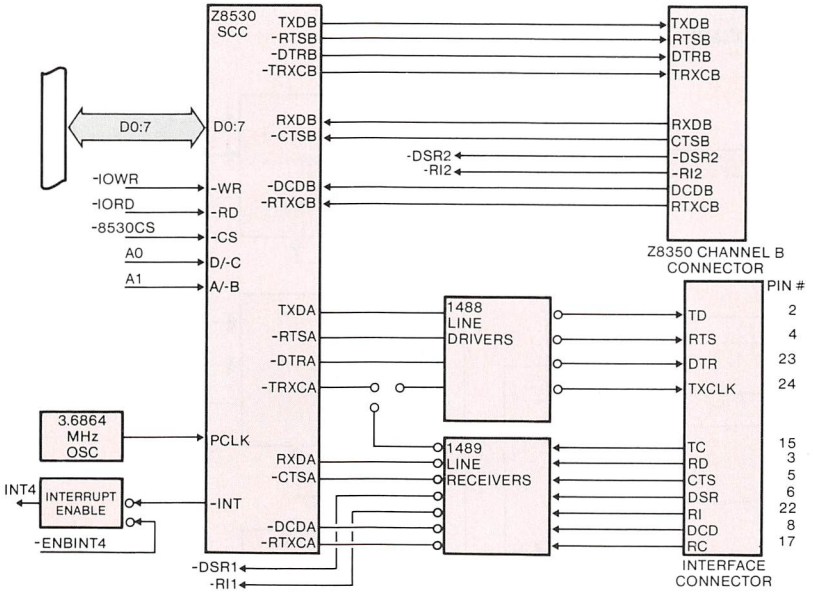
- Rate Generator
  The output is low for one period of the input clock. The period from one output pulse to the next equals the number of input counts in the count register.

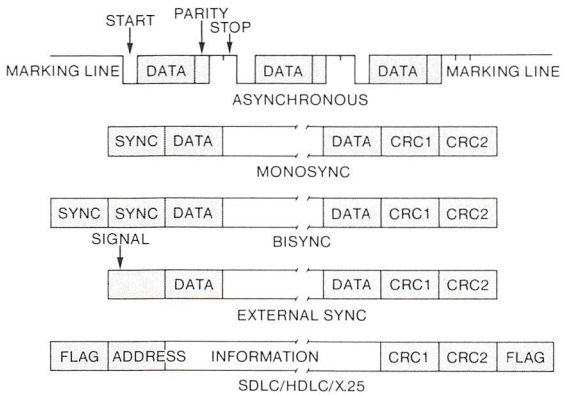
- Square Wave Rate Generator
  The output remains low for one period of the input clock. The output remains high until one half the count has elapsed. If the count is odd, the output is high for (n+1)/2 and low for (n-1)/2.

- Software Triggered Strobe
  After the mode is set, the output will be high. When the count is loaded, the counter begins counting. On the terminal count,the output goes low for one clock period and then goes high.

- Hardware Triggered Strobe
  The counter starts counting after the rising edge of the trigger input and goes low for one clock period when terminal count is reached.

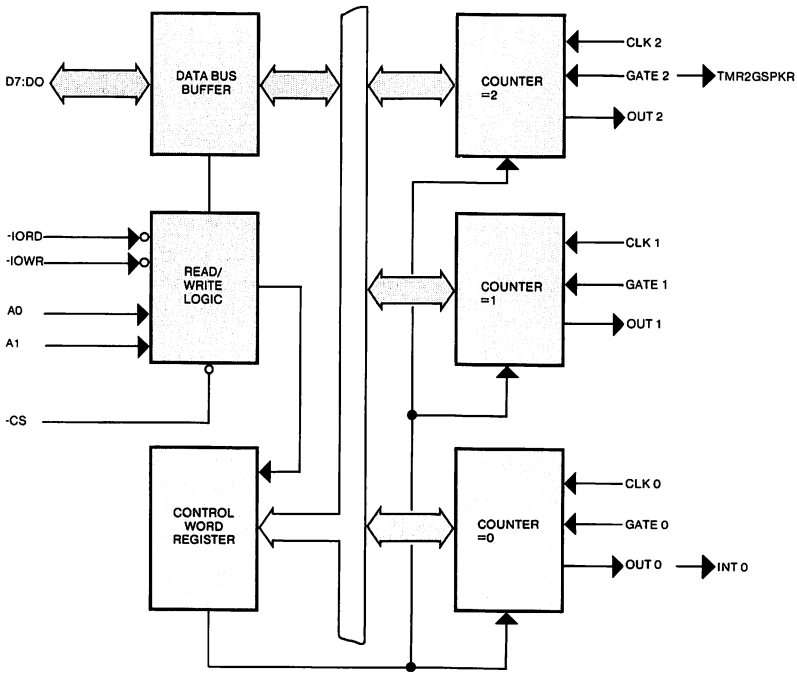
These timing diagrams illustrate the different modes.





In the AT&T Personal Computer 6300 system, Counter 0 provides real time interrupts on INT0, counter 1 requests memory refreshes, and counter 2 generates a pulse train for the audio speaker.

Block Diagram

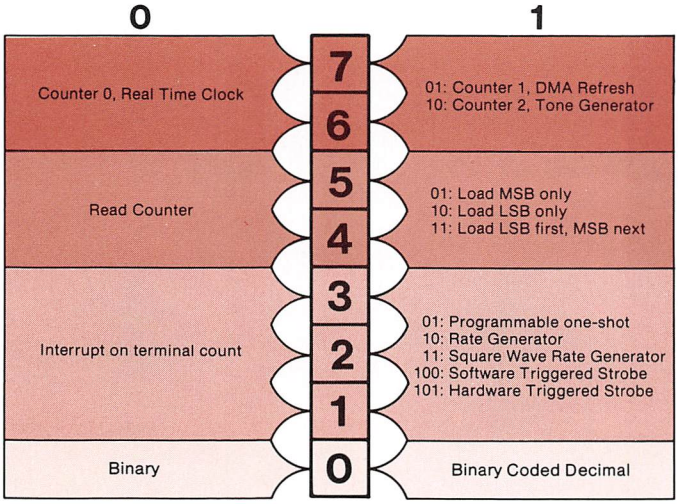


Registers

PORT #	NAME	READ/ WRITE	DESCRIPTION
40	COUNTER 0	R/W	Provides real time interrupt INT0
41	COUNTER 1	R/W	Provides signals to refresh memory
42	COUNTER 2	R/W	Generate pulse train for the audio speaker
43	CONTROL	W	See Layout

Layout

CONTROL



Functions

- LOAD COUNTER
  This function allows you to set a specific counter.
  OUTPUT: CONTROL REGISTER
  BITS 0 binary or BCD
  BITS 1-3 mode
  BITS 4-5 1,2 or 3
  BITS 6-7 counter
  8 or 16 bit count value

- READ COUNTER
  This function allows you to read a specific counter.
  OUTPUT: CONTROL REGISTER
  BITS 4-5 0
  BITS 6-7 counter

Sequencing and Timing

All counters must be initialized with the control register. The control register specifies the number of bytes which must be loaded.

Whenever a read or a load command is issued, the requested counter bytes must be read or written. In the read case, two reads are necessary, the first for the least significant byte (LSB) and the last for the most significant byte (MSB). In the write case, the control register specifies the byte to write.

1 microsecond recovery time is required between a read or a load and any other control signal.

Input to the timer is 1.2288MHz. Therefore, there are 18.75 interrupts per second. To generate a 1.00 KHz tone with the audio speaker, a square wave rate generator is used with a count of 614 (1.2288MHz/2*614 = 1KHz).

Sample Program

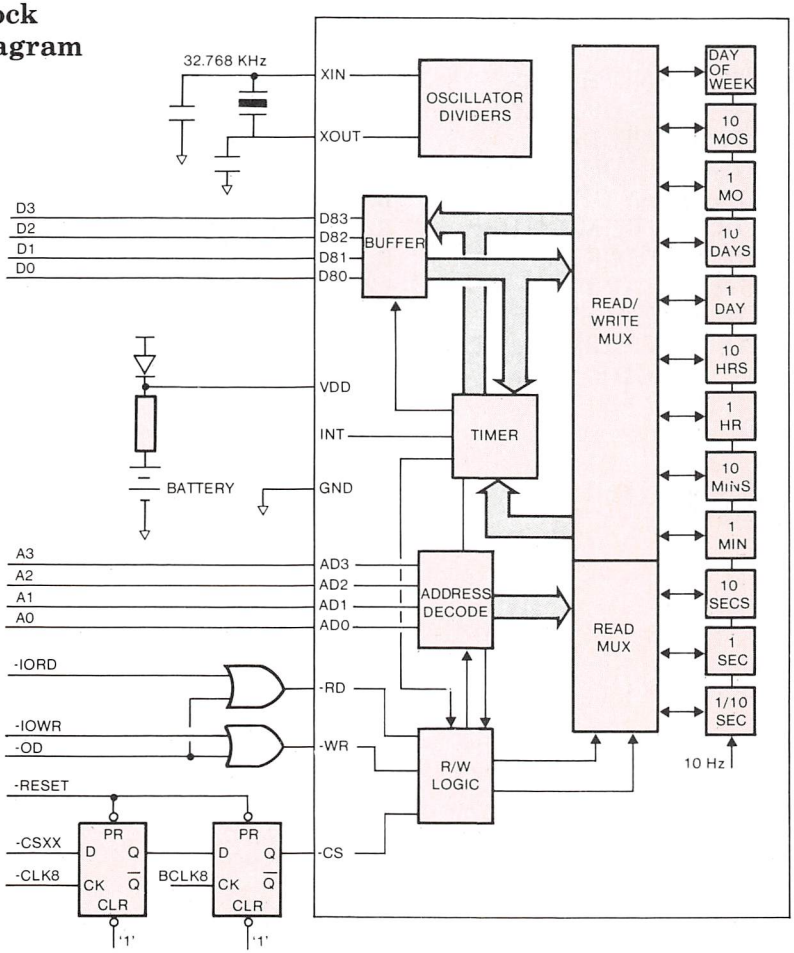
```
;
; ASK_FOR_INTR
;       This program requests an interrupt in
;       approximately 10 usec (9765.6 nsec)
;
TIMER_CONTROL   EQU 43
TIMER0          EQU 40

INT_MASK        EQU 21

ASK_FOR_INTR:
  MOV AL,0FEH               ;allow only INT0
                            ;interrupt
  OUT INT_MASK,AL           ;send mask
  MOV AL,00110000B          ;binary counter,
                            ;interrupt on terminal
                            ;count, set counter 0
  OUT TIME_CONTROL,AL
  MOV AX,12                 ;12 * 813.8 nsec
  OUT TIMER0,AL
  MOV AL,AH                 ;output counter, LSB
                            ;then MSB
  OUT TIMER0,AL
  RET
```

Real Time Clock and Calendar

Functional Description

The real time clock and calendar keep the current data and time. All of the date and time fields can be read but the second fields cannot be written. The calendar keeps up to eight years. A rechargeable battery keeps the unit running even when the computer is turned off.

Block Diagram

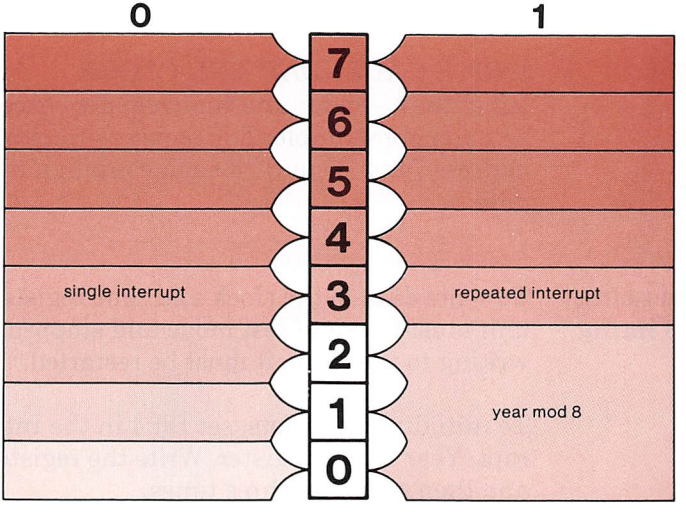


Registers

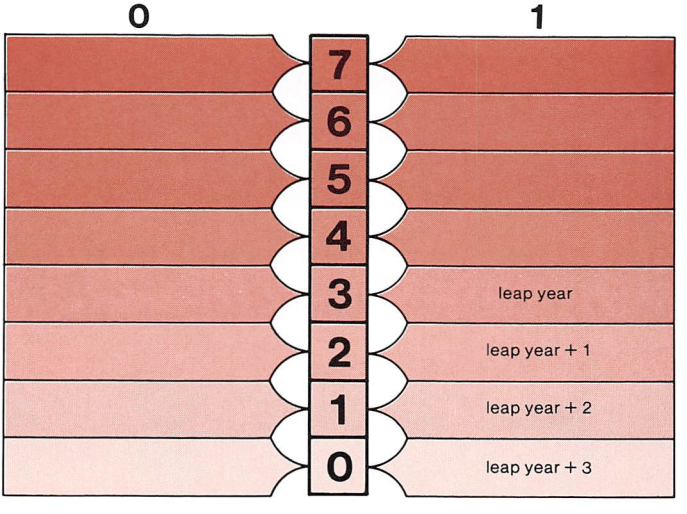
PORT #	NAME	READ/ WRITE	DESCRIPTION
70	TEST PORT	W	0=not test mode, 1=test mode
71	1/10 OF A SECOND	R	nybble, 0-9
72	UNIT SECONDS	R	nybble, 0-9
73	10'S OF SECONDS	R	nybble, 0-5
74	UNIT MINUTES	R/W	nybble, 0-9
75	10'S OF MINUTES	R/W	nybble, 0-5
76	UNIT HOURS	R/W	nybble, 0-9
77	10'S OF HOURS	R/W	nybble, 0-1
78	UNIT DAYS	R/W	nybble, 0-9
79	10'S OF DAYS	R/W	nybble, 0-3
7A	DAY OF WEEK	R/W	nybble, 1-7
7B	UNIT MONTHS	R/W	nybble, 0-9
7C	10'S OF MONTHS	R/W	nybble, 0-1
7D	LEAP YEAR	W	See layout
7E	STOP/START	W	0 = stop, FF = start
7F	INTERRUPT/YEAR MOD 8	R/W	See layout

Layout

INTERRUPT/YEAR MOD 8



LEAP YEAR



Functions

- READ CALENDAR AND CLOCK
  All of the calendar and time registers are readable. Those from port address 71 to 7D and 7F contain a nibble of data.

- WRITE CALENDAR AND CLOCK
  All of the calendar and time registers except seconds are writable. The registers from port address 74 to 7D and 7F each contain a nibble of data.

Sequencing and Timing

To write data to the clock and time registers, the unit must be out of test mode and stopped. After writing to the clock, it must be restarted.

To initialize interrupts, set Bit 4 in the Interrupt/Year mod 8 register. Write the register once and then read it in three times.

If an update occurs while reading a register, the illegal code of F is returned.

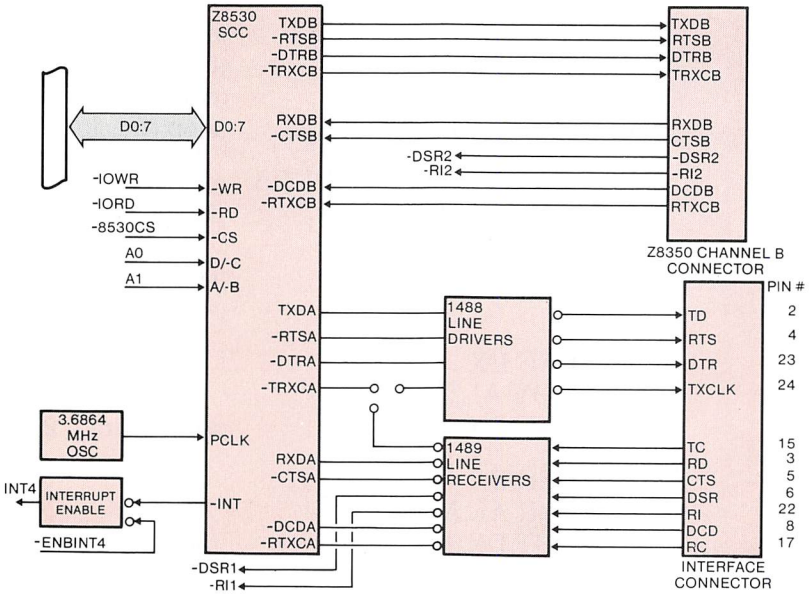
Sample Program

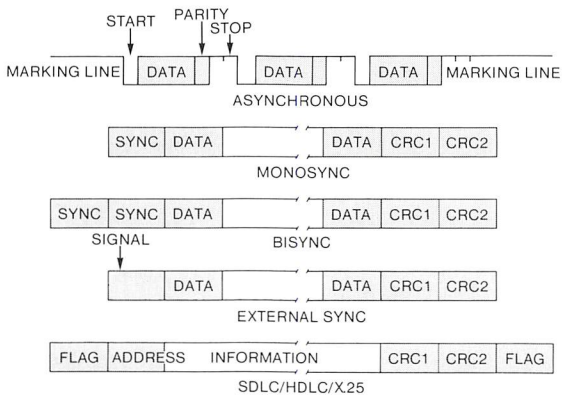
```
; SET TIME
;       This routine sets the time to 12:00 noon
;
TEST_PORT       EQU 70
STOP_START      EQU 7E
TENS_HOURS      EQU 77
WRITE_TIME:
   XOR AX,AX                    ;AX = 0
   MOV DX,TEST_PORT             ;setup dx
   OUT DX,AL                    ;take out of
                                ;test mode
   OUT STOP_START,AL            ;stop the clock

   MOV DX,TENS_HOURS            ;port addr of
                                ;10's of hours
   MOV AL,1
   OUT DX,AL                    ;ten's of hours
                                ;= 1

   DEC DX
   MOV AL,2
   OUT DX,AL                    ;unit hours = 2

   DEC DX
   XOR AL,AL
   OUT DX,AL                    ;tens of minutes
                                ;= 0

   DEC DX
   OUT DX,AL                    ;minutes = 0

   MOV AL,0FFH
   OUT STOP_START,AL            ;start the clock

   RET
```

Serial Communications Controller

Functional Description

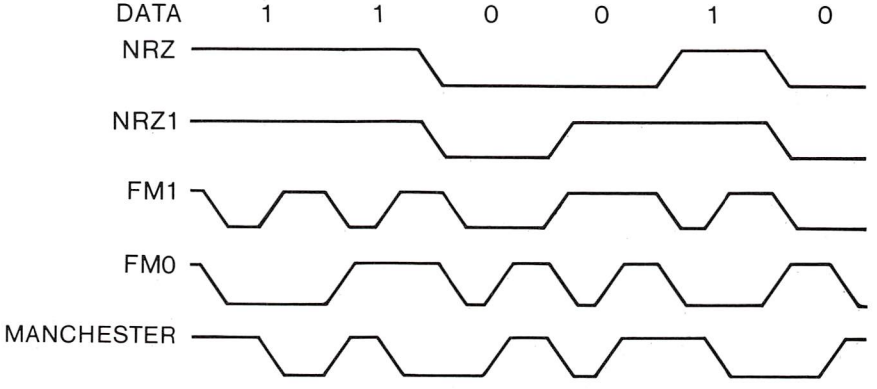
Z8530 Serial Communications Controller (SCC) performs serial-to-parallel conversion on input data characters received from a modem and parallel-to-serial conversion on output data characters received from the CPU. It supports the following common asynchronous and synchronous data communication protocols.





In addition, the SCC supports five encoding methods — NRZ, NRZI, FM1 (bi-phase mark), FM0 (bi-phase space), and Manchester (bi-phase level).

Figure 20



The SCC has the following capabilities:

Asynchronous

- 5, 6, 7, or 8 bits per character
- 1, 1 1/2, or 2 stop bits
- Odd or even parity
- Times 1, 16, 32, or 64 clock modes
- Break generation and detection
- Parity, overrun and framing error detection

Byte-oriented synchronous

- Internal or external character synchronization
- 1 or 2 sync characters in separate registers
- Automatic sync character insertion and deletion
- Cyclic redundancy check (CRC) generation/detection
- 6- or 8-bit sync character

SDLC/HDLC

- Abort sequence generation and checking
- Automatic zero insertion and deletion
- Automatic flag insertion between messages
- Address field recognition
- I-field residue handling
- CRC generation/detection
- SDLC loop mode with EOP recognition/loop entry and exit

The baud rate is programmable for both channels.

Registers

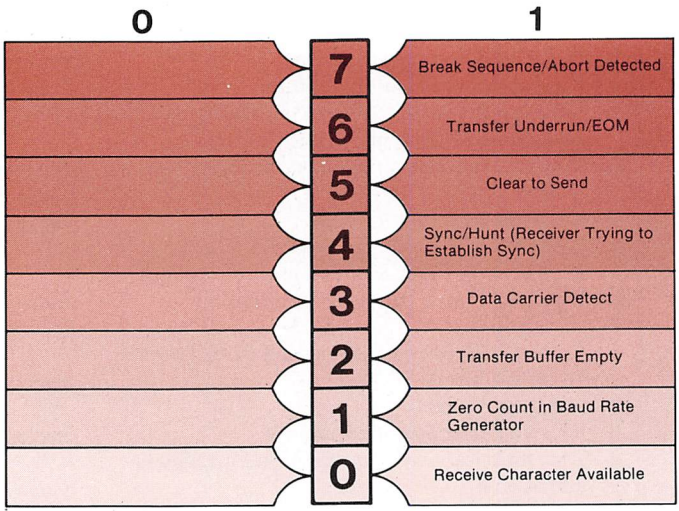
PORT # A	B	NAME	READ/ WRITE	DESCRIPTION
51	53	DATA	R/W	Transfer data
50	52	SCC REGISTER POINTER	R/W	Transfer SCC register number and SCC register data

SCC Registers

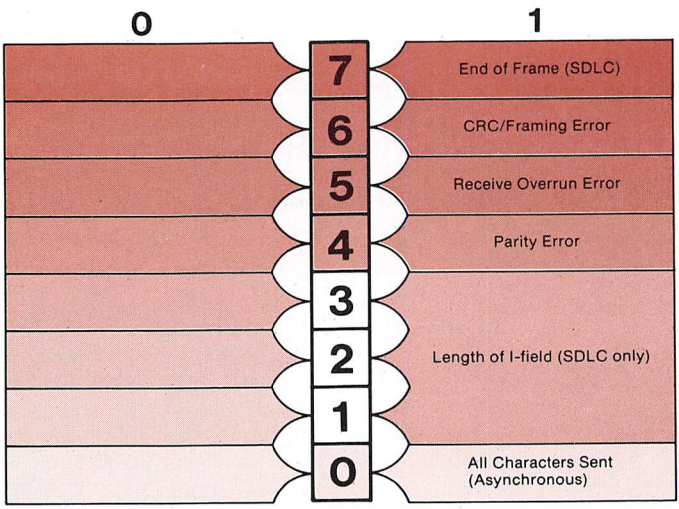
READ/ WRITE	NO	NAME	DESCRIPTION
R	0	Buffer and External Status	See layout
R	1	Special Receive Condition Status	See layout
R	2	Modified Interrupt Vector (Channel B) Unmodified Interrupt Vector (Channel A)	
R	3	Interrupt Pending Bits (Channel A)	See layout
R	8	Receive Buffer	See layout
R	10	Miscellaneous Status	See layout
R	12	Low Byte of Baud Rate Generator Constant	
R	13	High Byte of Baud Rate Generator Constant	
R	15	External/Status Interrupt Information	See layout
W	0	CRC Initialize	See layout
W	1	Transmit/Receive Interrupt and Data Transfer Mode Definition	See layout
W	2	Interrupt Vector	
W	3	Receive Parameters and Control	See layout
W	4	Transmit/Receive Miscellaneous Parameters and Modes	See layout
W	5	Transmit Parameters and Control	See layout
W	6	Sync Characters or SDLC Address Field	See layout
W	7	Sync Characters or SDLC Flag	See layout
W	8	Transmit Buffer	See layout
W	9	Master Interrupt Control and Reset	See layout
W	10	Miscellaneous Transmit/Receive Control Bits	See layout
W	11	Clock Mode Control	See layout
W	12	Low Byte of Baud Rate Generator Constant	
W	13	High Byte of Baud Rate Generator Constant	
W	14	Miscellaneous Control Bits	See layout
W	15	External/Status Interrupt Control	See layout

Layout

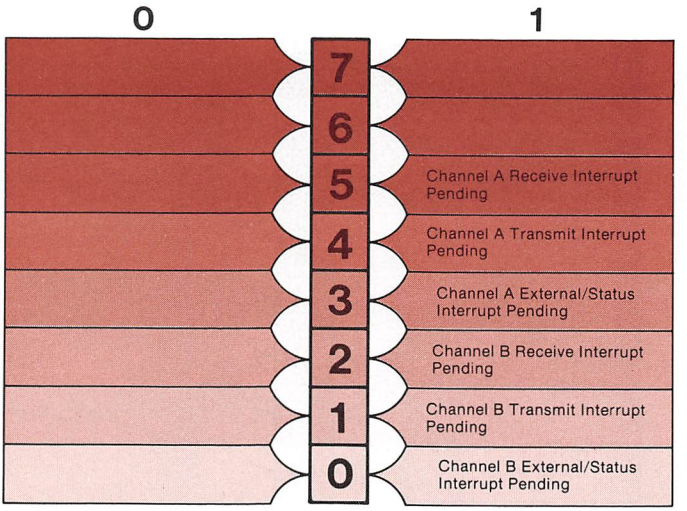
READ REGISTER 0



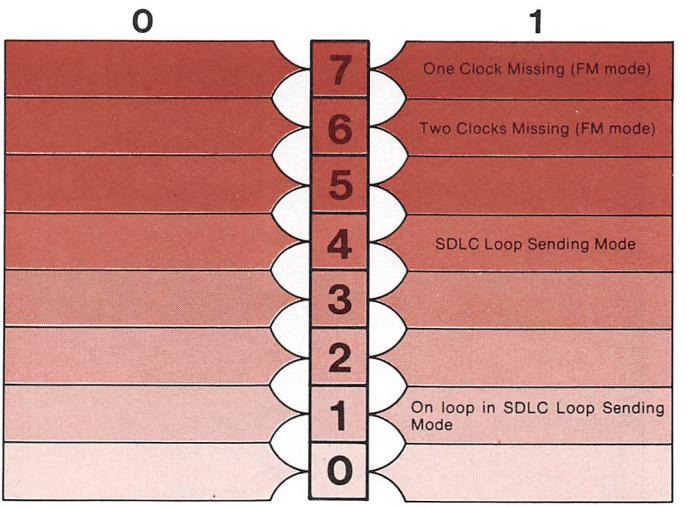
READ REGISTER 1



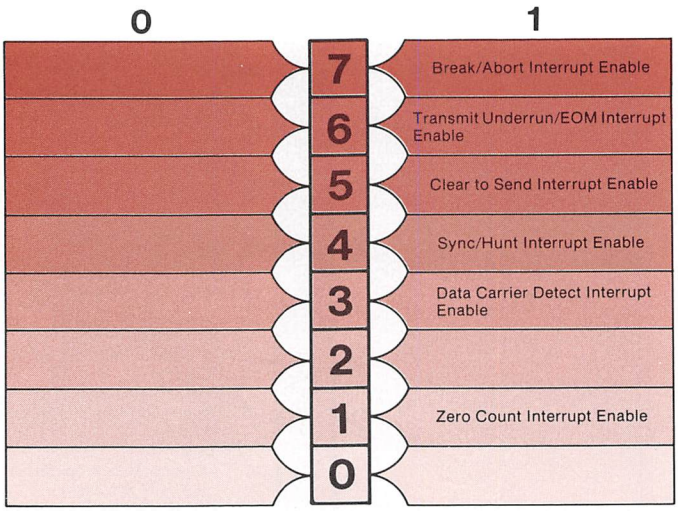
READ REGISTER 3



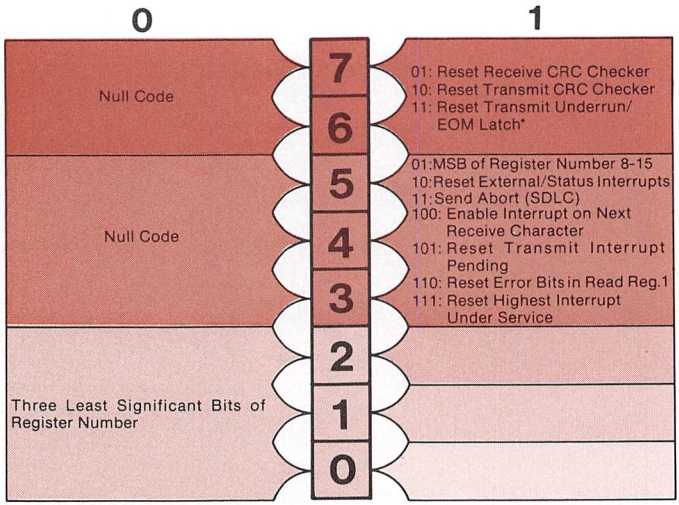
READ REGISTER 10



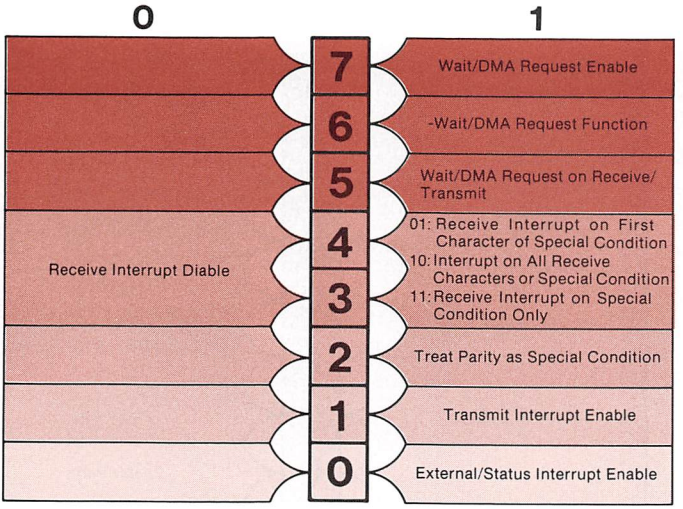
READ REGISTER 15



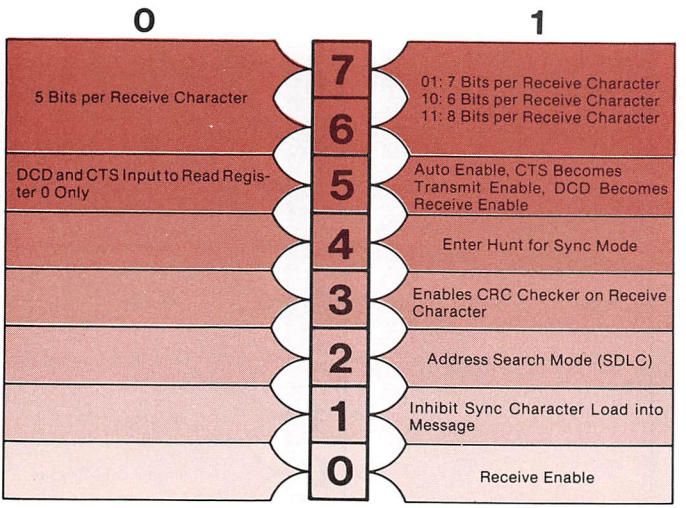
WRITE REGISTER 0



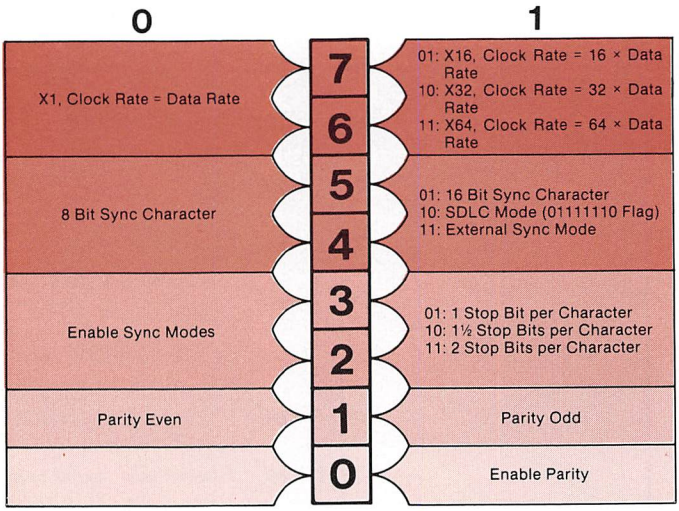
WRITE REGISTER 1



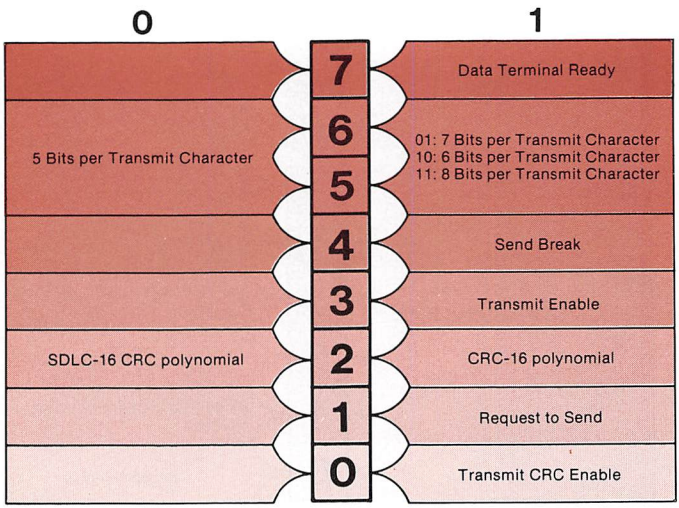
WRITE REGISTER 3



WRITE REGISTER 4



WRITE REGISTER 5



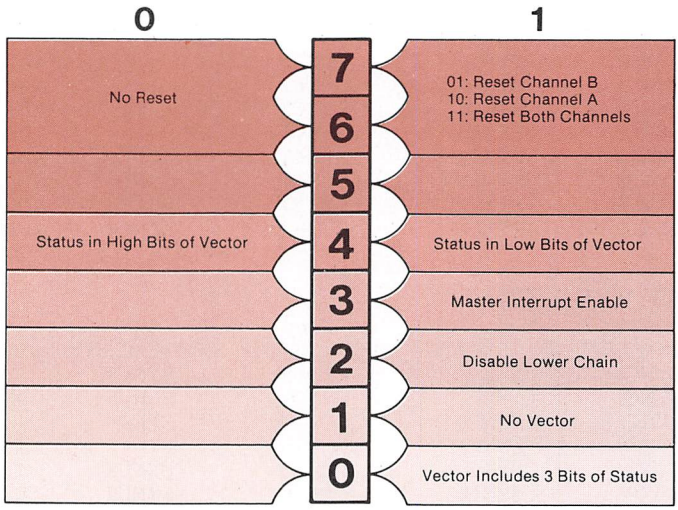
WRITE REGISTER 6

	MONO,8 BITS	MONO,16 BITS	BISYNC,16 BITS	BISYNC,12 BITS	SDLC	SDLC
0	SYNC0	SYNC0	SYNC0	1	ADR0	X
1	SYNC1	SYNC1	SYNC1	1	ADR1	X
2	SYNC2	SYNC2	SYNC2	1	ADR2	X
3	SYNC3	SYNC3	SYNC3	1	ADR3	X
4	SYNC4	SYNC4	SYNC4	SYNC0	ADR4	ADR4
5	SYNC5	SYNC5	SYNC5	SYNC1	ADR5	ADR5
6	SYNC6	SYNC0	SYNC6	SYNC2	ADR6	ADR6
7	SYNC7	SYNC1	SYNC7	SYNC3	ADR7	ADR7

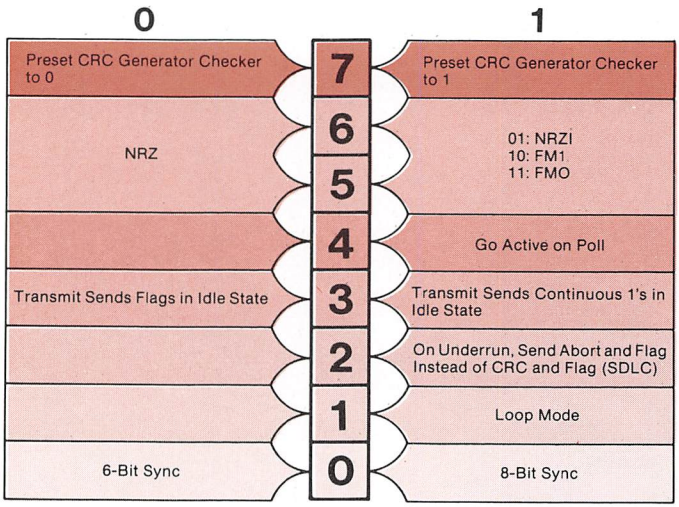
WRITE REGISTER 7

	MONO,8 BITS	MONO,16 BITS	BISYNC,16 BITS	BISYNC,12 BITS	SDLC
0	SYNC0	X	SYNC8	SYNC4	0
1	SYNC1	X	SYNC9	SYNC5	1
2	SYNC2	SYNC0	SYNC10	SYNC6	1
3	SYNC3	SYNC1	SYNC11	SYNC7	1
4	SYNC4	SYNC2	SYNC12	SYNC8	1
5	SYNC5	SYNC3	SYNC13	SYNC9	1
6	SYNC6	SYNC4	SYNC14	SYNC10	1
7	SYNC7	SYNC5	SYNC15	SYNC11	0

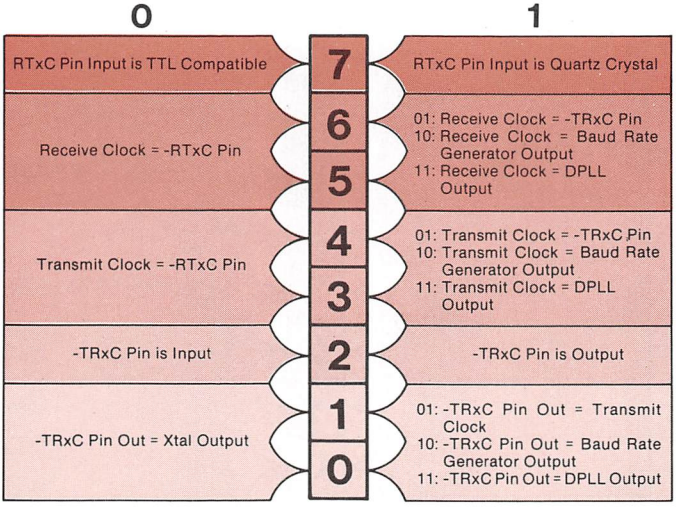
WRITE REGISTER 9



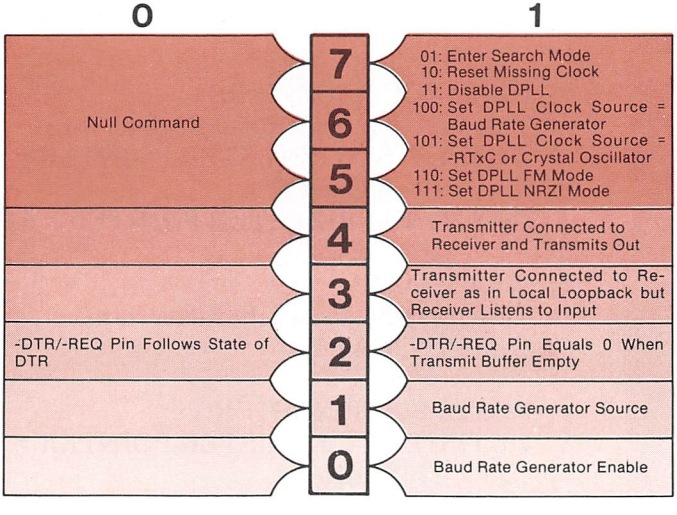
WRITE REGISTER 10



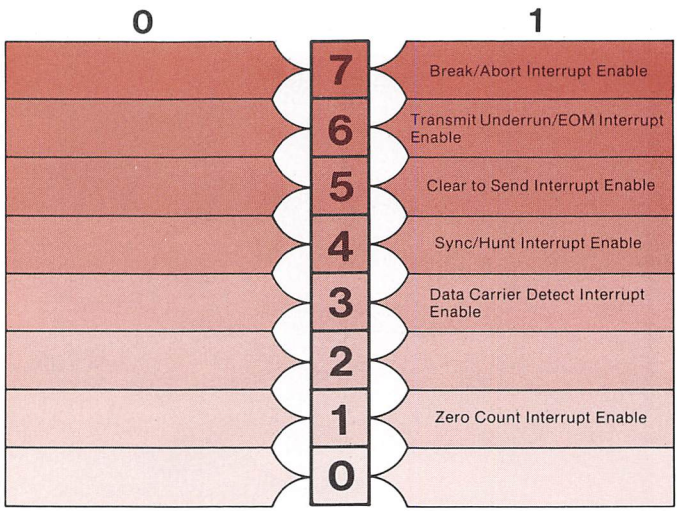
WRITE REGISTER 11



WRITE REGISTER 14



WRITE REGISTER 15



Functions

- READ DATA
  This function reads character data.
  INPUT: DATA

- READ SCC REGISTERS
  This function reads Read Register 0-15.
  OUTPUT: SCC REGISTER POINTER =
  register #
  INPUT: SCC REGISTER POINTER = data

- WRITE DATA
  This function writes character data.
  OUTPUT: DATA

- WRITE SCC REGISTER
  OUTPUT: SCC REGISTER POINTER =
  register #
  OUTPUT: SCC REGISTER POINTER = data

Sequencing and Timing

The SCC has direct addressing for the data register only. To access the other SCC registers, first write the register number to the SCC Register Pointer and then read or write the register using the SCC Register Pointer.

The SCC can operate in three basic modes to transfer data, status and control information:

- polling
- interrupt
- block

The block mode is not used on the AT&T Personal Computer 6300.

In the polling mode, Receive Character Available and Transmit Buffer Empty in Read Register 0 are examined before receiving and sending a character. All interrupt functions must be disable. To do this, clear the Master Bit Enable and set the No Vector bit in Write Register 9. Then clear Write Register 1. This disables specific types of interrupts.

For interrupt mode, the Master Interrupt Enable in Write Register 9 must be set. In addition, bits 3-4 in Write Register 1 specify interrupts on receive character conditions. Bit 1 in Write Register 1 enables interrupts on Transmit Buffer Empty. Bit 0 enables External/Status interrupt. This interrupt is caused by transmit underrun condition, a zero count in the baud rate generator, a break detection (Asynchronous Mode), Abort (SDLC Mode), or EOP (SDLC Loop Mode). Write Register 15 enables or disables more specific types of interrupts.

Interrupt sources have the following priority:

Receive Channel A
Transmit Channel A
External/Status Channel A
Receive Channel B
Transmit Channel B
Receive Channel B

A bit in Read Register 3 is set to indicate the highest priority device needing service. You can read the vector address of the interrupt service routine in Read Register 2 if it was programmed. If this vector is read on Channel B, it includes status bits. Vectors are initialized with Write Register 2. The interrupt service routine resets the Highest Interrupt Under Service in Write Register 0. Other interrupts are reset in Write Register 0.

To set the baud rate, first clear bits 0-1 in Write Register 14. Then load Write Register 12 and 13 with the time constant. Last set bit 0-1 in Write Register 14 to enable the baud rate generator. To use the baud rate, set the transmit and receive clocks in Write Register 11 to the baud rate generator.

To determine the time constant to use for a given baud rate, use this formula:

$$\frac{3{,}686{,}400}{(16)(2)(\text{Baud Rate})} - 2 = \text{Time Constant}$$

The following table states the divisors to use to obtain a given baud rate.

BAUD RATES USING 3.6864 MHZ CLOCK

BAUD RATE	TIME CONSTANT
110	1045
150	766
300	382
600	190
1200	94
2400	46
4800	22
9600	10

In Asynchronous mode, you initialize:

- Write Register 1 to disable DMA transfers and to enable or disable interrupts,
- Write Register 3 to set Receive Enable and the number of bits per receive character, to disable synchronous functions, and to enable or disable Auto Enable,
- Write Register 4 to set parity, stop bits, and data rate, and to disable synchronous mode,
- Write Register 5 to set Transmit Enable and the number of bits per transmit character, to enable or disable Request to Send, Data Terminal Ready and Send Break, and disable synchronous functions,
- Write Register 9 to force a reset and set interrupt parameters,
- Write Register 10 to choose an encoding method,
- Write Register 11 to select clock sources,
- Write Register 12 and 13 to set the baud rate time constant, and
- Write Register 15 to enable the baud rate generator.

To transmit a character, wait for Transmit Buffer Empty to be set in Read Register 1 of the SCC. Then write the character to the Data Register.

To receive a character, wait for Receive Character Ready in the Read Register 1 of the SCC. Then read the character from the Data Register.

In synchronous mode, you must transfer the data using interrupts. The External/Status interrupt is used to monitor the status of Clear to Send and Transmit Underrun/EOM latch.

In bisynchronous mode, you initialize:

- Write Register 0 to reset Transmit Underrun/EOM Latch, receive CRC checker, external/status interrupts, and enable interrupts on next receive character,
- Write Register 3 to enable the receiver, and to program Sync Character Load Inhibit, Enter Hunt For Sync Character, and number of bits per receive character,
- Write Register 4 to set parity, enable sync modes, number of bits per sync character, and clock mode,
- Write Register 5 to enable Transmit CRC, to request 16-bit CRC polynomial, enable transmit, transmit 8 bits per character, and to set Data Terminal Ready and Request to Send,
- Write Register 6 and 7 to set the sync bytes,
- Write Register 9 first to reset the hardware and later to set interrupts and vector variables,

- Write Register 10 to set length of sync character, the encoding method, to preset the CRC generator and to set the loop mode and go active on poll if wanted,
- Write Register 11 to set the clock sources, and
- Write Register 15 to set interrupt enable conditions.

The monosync transmitter is initialized as a bisynchronous transmitter with two exceptions:

- Only one sync character is written, and
- The 6-bit or 8-bit selection in Write Register 10 must be made.

In SDLC mode, you initialize:

- Write Register 0 to reset the transmit CRC generator after transmit enable has been done and to enable interrupts,
- Write Register 1 to enable interrupts,
- Write Register 3 to select bits per receive character, to set address search mode, to enable CRC receiver and receive enable,
- Write Register 4 to set SDLC mode before anything else is initialized and later to set clock mode,
- Write Register 5 to select the SDLC-CRC polynomial, to set Request to Send, Data Terminal Ready, transmit character length, transmit enable, and transmit CRC enable
- Write Register 6 to contain the secondary address field,

- Write Register 7 to contain flag character 01111110,
- Write Register 9 to reset the hardware and to set interrupt parameters,
- Write Register 10 to set loop mode, Go Active on Poll, Mark/Idle Flag, Abort on Underrun, the CRC preset condition, and the encoding mode,
- Write Register 11 to set clock sources,
- Write Register 14 to set the clock source for the DPLL, and
- Write Register 15 to set interrupt enable condition.

Sample Program

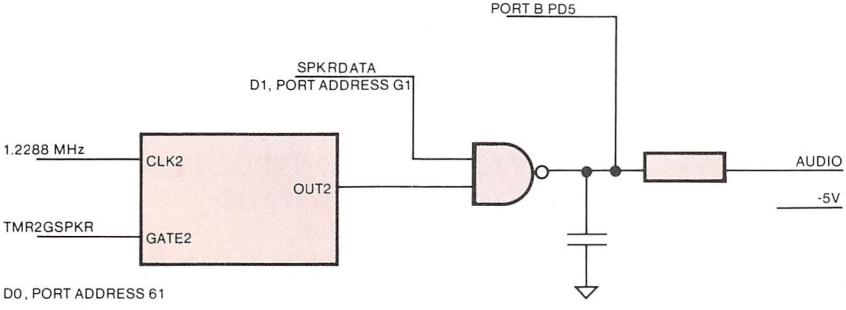
```
; RECEIVE
;       This routine receives one character
;
PTER        EQU 50
DATA        EQU 51
RECEIVE:
   MOV DX,PTER     ;Pointer Port Address
   XOR AL,AL       ;AL = 0
   OUT DX,AL       ;Select Read Register 0
REC1:
   IN AL,DX        ;Get Read Register 0
   TEST AL,1       ;Receive character available
   JZ REC1         ;Jump if no character
   MOV DX,DATA     ;Data Port Address
   IN AL,DX        ;Read character
   RET
```

Speaker

Functional Description

The speaker uses a permanent magnet speaker which is driven by one of two sources. Counter 2 of the Interval Timer can be programmed to automatically generate a pulse train. A bit in the Keyboard Control Register controls this pulse train. A bit in the Keyboard Control Register can also be programmed to manually generate a pulse train.

Block Diagram



Registers

PORT #	NAME	READ/ WRITE	DESCRIPTION
61	KEYBOARD CONTROL	W	Contains speaker enable and manual pulse train bits. See Keyboard documentation.
42	COUNTER 2	R/W	Counter for audio speaker tone generation. See Timer documentation.
43	TIMER CONTROL	W	Control register for Interval Timer. See Interval Timer documentation.

Functions

- AUTOMATIC PULSE TRAIN
  This function automatically generates an audible sound.
  OUTPUT:
  INTERVAL TIMER CONTROL REGISTER
  BITS 1-3 = 3, square wave rate generator
  BITS 4-5 = 3, set counter
  BITS 6-7 = 2, generator
  INTERVAL TIME COUNTER 2
  KEYBOARD CONTROL
  BIT 0= 1, turns on speaker

- MANUAL PULSE TRAIN
  This function manually generates an audible sound.
  OUTPUT: KEYBOARD CONTROL
  BIT 1 This bit is set and cleared to generate a pulse train.

Sequencing and Timing

Input to the timer is 1.2288MHz. To generate a 1.00 KHz tone with the audio speaker, a square wave rate generator is used with a count of 614 (1.2288MHz/2*614 = 1KHz).

Sample Program

```
;
BEEP
: This program sounds the beep manually
;
KEY_CONTROL          EQU 61
TIMER_CONTROL        EQU 43
BEEP:
  MOV DX, TIMER_CONTROL ;port address
  MOV AL, B8H           ;of timer
  OUT DX, AL            ;set channel 2
                        ;in mode 4
  MOV DX, KEY_CONTROL
  IN AL,DX
  MOV AH,AL
  OR AH, 01H            ;turn on Gate
  MOV BL, 80H
BEEP1:

  MOV AL, AH            ;restore value
  AND AL, OFDH
  OUT DX, AL
  MOV CX, 48H
  LOOP $
  MOV AL, AH
  OR AL, 02H
  OUT DX, AL
  MOVE CX, 48H
  LOOP $
  DEC BL
  JNZ BEEP1
  MOV AL,AH
  OUT DX,AL
  RET
```

Video Controller

Functional Description

The AT&T Personal Computer 6300 Display Controller interfaces the CPU to either monochrome or color displays. It uses a HD6845 CRT Controller. The Display Controller operates in two basic modes — text or all points addressable (APR) graphics. Several resolutions are available depending on the mode and display.

RESOLUTION	PC COMPATIBLE	GRAPHIC/ TEXT	COLOR/ MONOCHROME
80X25	YES	T	C/M
40X25	YES	T	C/M
640X400	NO	G	M
640X200	YES	G	M
320X200	YES	G	C

In text mode, character attributes include reverse video, blinking, highlight, hide and underline. In color mode if blinking is not requested, one of 16 colors can be chosen. Otherwise, one of 8 colors can be chosen.

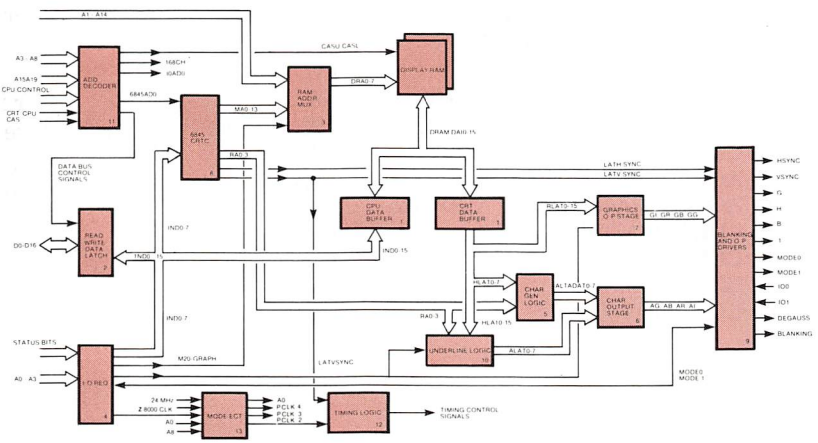
In graphic mode, each pixel on a color monitor is one of four selected colors. These four colors are from a choice of 16. In a monochrome monitor, these 16 colors are shades of gray from black to white.

The Display Controller has 32K of RAM to refresh one screen page.

The Display Controller can be upgraded with an optional board. This board gives you the following features:

- Up to three additional screen pages in RAM
- Software controlled look-up table for an international character set
- High resolution characters (16 X 16 pixels instead of 8 X 16 pixels)
- Hardware smooth scroll
- The ability to mix text and graphic modes simultaneously
- Up to 16 colors can be displayed at once

Block Diagram



Registers

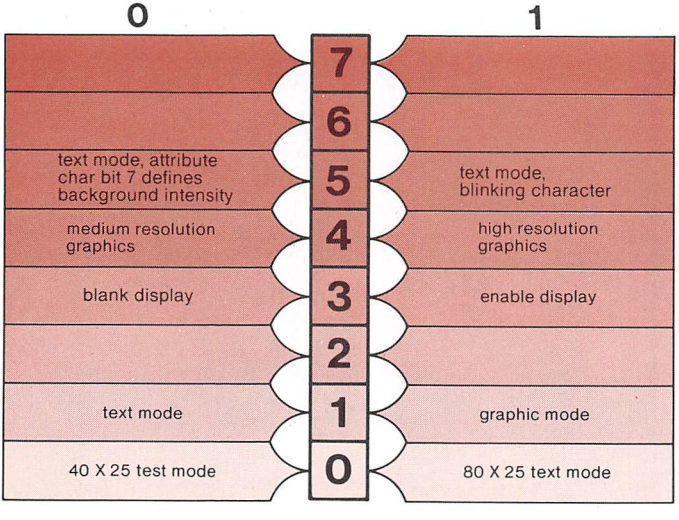
PORT #	NAME	READ/ WRITE	DESCRIPTION
3D8	MODE SELECT REGISTER 1	W	See layout
3D9	COLOR SELECT REGISTER	W	See layout
3DA	STATUS REGISTER	R	See layout
3DE	MODE SELECT REGISTER 2	W	See layout
3D4	POINTER TO HD6845 REGISTER	W	
3D5	HD6845 DATA REGISTER	R/W	

HD6845 Registers

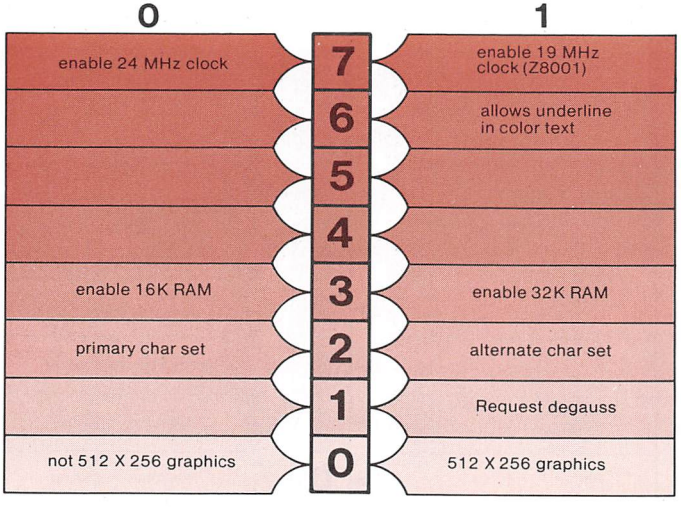
NO	NAME	READ/ WRITE	INITIALIZATION VALUE 40X25	80X25	GRAPHIC
0	HORIZONTAL TOTAL	W	38	71	38
1	HORIZONTAL DISPLAYED	W	28	50	28
2	HORIZONTAL SYNC POSITION	W	2D	5A	2D
3	HORIZONTAL SYNC WIDTH	W	06	0C	06
4	VERTICAL TOTAL	W	1F	1F	7F
5	VERTICAL TOTAL ADJUST	W	06	06	06
6	VERTICAL DISPLAYED	W	19	19	64
7	VERTICAL SYNC POSITION	W	1C	1C	70
8	INTERLACE MODE	W	02	02	02
9	MAX. SCAN LINE ADDRESS	W	07	07	01
A	CURSOR START LINE (SIZE)	W	06	06	06
B	CURSOR END LINE	W	07	07	07
C	ACTIVE PAGE START ADDR (H)	W	00	00	00
D	ACTIVE PAGE START ADDR (L)	W	00	00	00
E	CURSOR ADDRESS (H)	R/W			
F	CURSOR ADDRESS (L)	R/W			
10	LIGHT PEN (H)	R			
11	LIGHT PEN (L)	R			

Layout

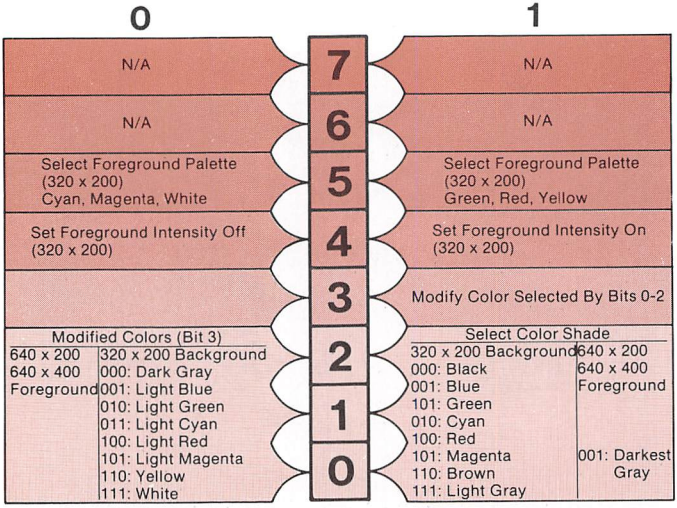
MODE SELECT REGISTER 1



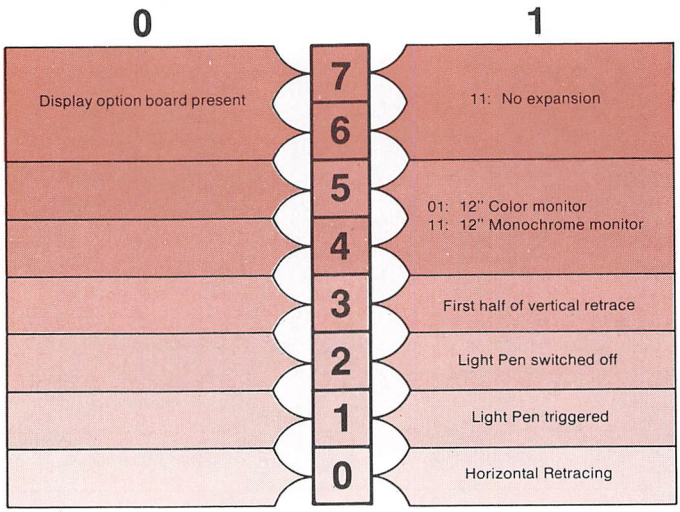
MODE SELECT REGISTER 2



COLOR SELECT REGISTER 1 (Graphics Mode Only)



STATUS REGISTER

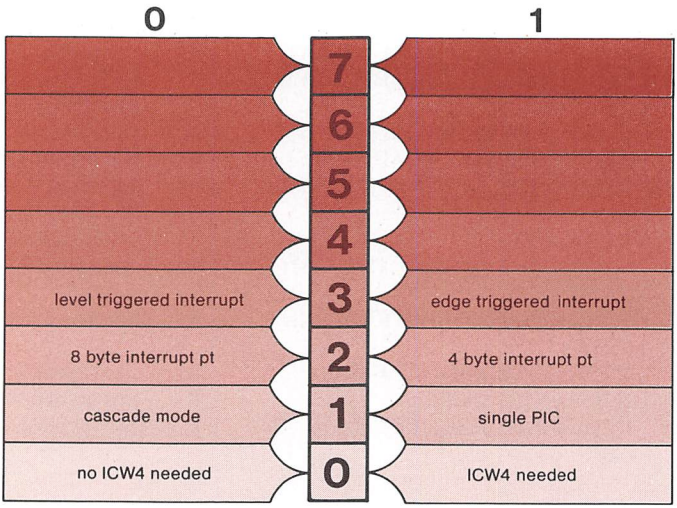


Registers

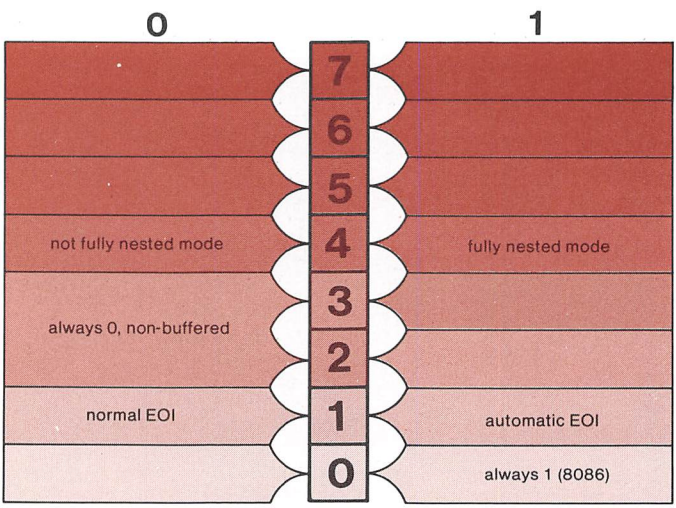
PORT #	NAME	READ/ WRITE	DESCRIPTION
20	INIT COMMAND WORD 1 (ICW1)	W	See layout
21	INIT COMMAND WORD 2 (ICW2)	W	INT0 interrupt type, multiple of 8
21	INIT COMMAND WORD 3 (ICW3)	W	Cascade mode only
21	INIT COMMAND WORD 4 (ICW4)	W	See layout
21	OPERATION COMMAND WORD 1 (OCW1, IMR)	R/W	Interrupt Mask Register See layout
20	OPERATION COMMAND WORD 2 (OCW2)	W	See layout
20	OPERATION COMMAND WORD 3 (OCW3)	W	See layout
20	IN-SERVICE REGISTER (ISR)	R	See layout
21	INTERRUPT LEVEL	R	See layout
20	INTERRUPT REQUEST REGISTER (IRR)	R	See layout

Layout

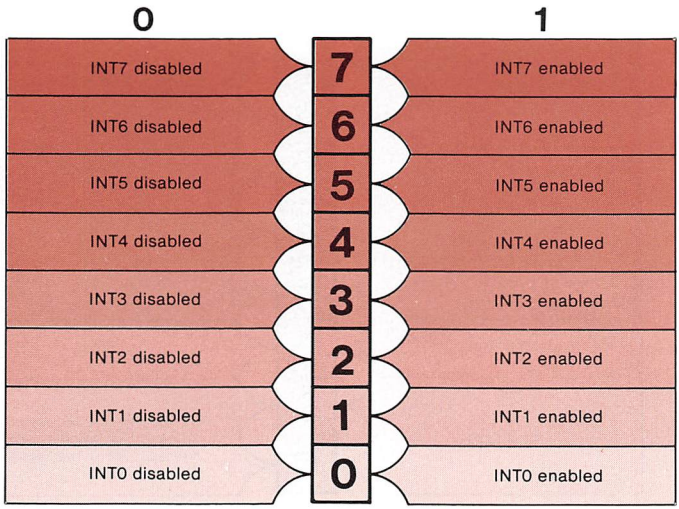
ICW1



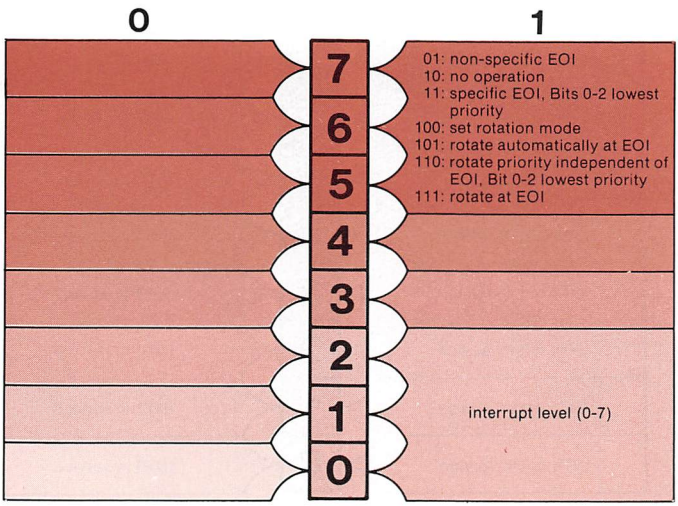
ICW4



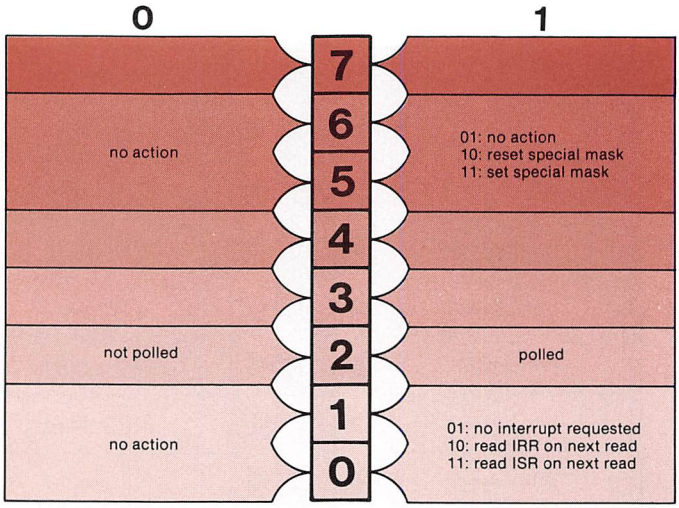
OCW1 or INTERRUPT MASK REGISTER



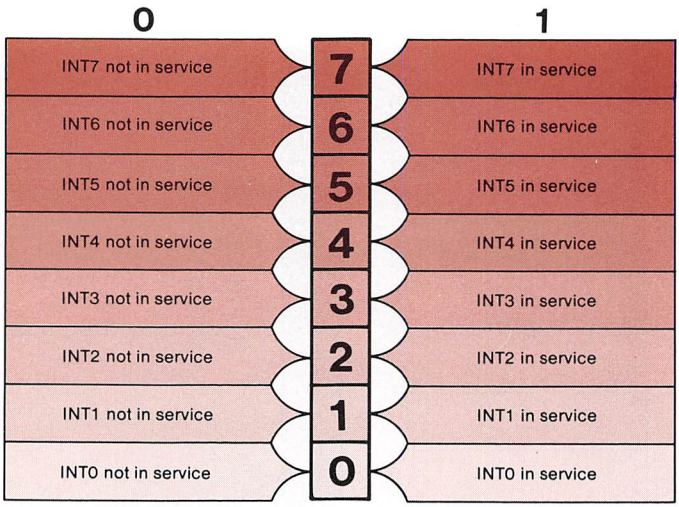
OCW2



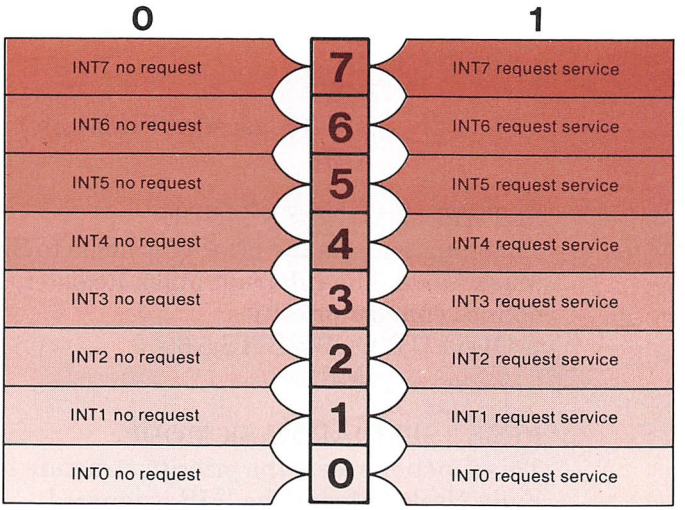
OCW3



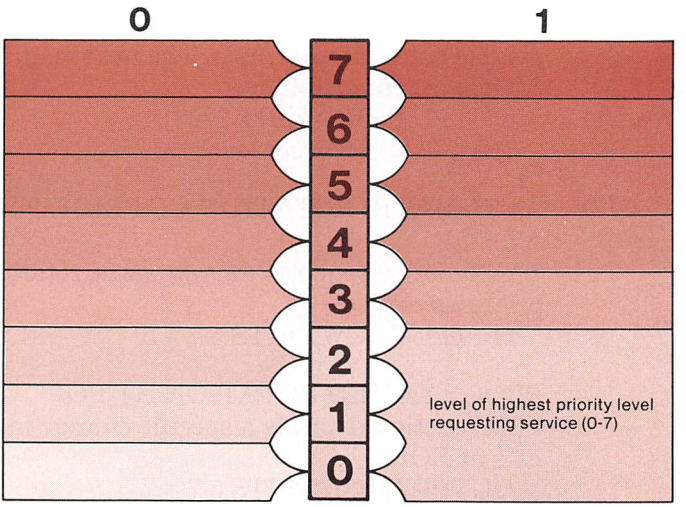
IN SERVICE REGISTER



INTERRUPT REQUEST REGISTER



INTERRUPT LEVEL



Functions

- INITIALIZATION
  This function prepares the PIC to accept interrupts by setting it to an initial state.
  OUTPUT: ICW1
  ICW2
  ICW3 (Cascaded PIC's only)
  ICW4

- SET SPECIAL MASK MODE
  This function sets the priority scheme to Special Mask Mode. The Interrupt Mask Register (IMR) defines enabled priorities.
  OUTPUT: OCW3 BITS 5-6 = 3
  IMR

- RESET SPECIAL MASK MODE
  This function resets the priority structure to the Fully Nested Mode. The IMR is ignored.
  OUTPUT: OCW3 BITS 5-6 = 1

- SET POLLED MODE
  This function sets the priority scheme to Polled Mode. The CLI instruction must be executed to disable external interrupts. The next read fetches the interrupt level.
  OUTPUT: OCW3 BIT 2 = 1
  INPUT: interrupt level

- AUTOMATIC ROTATION OF PRIORITIES AT EOI
  This function requests an automatic rotation of priorities at EOI.
  OUTPUT: ICW2 BITS 5-7 = 5

- PROGRAMMED ROTATION AT EOI
  This function requests a specific change in priorities at EOI.
  OUTPUT: ICW2 BITS 5-7 = 7
  ICW2 BITS 0-2 = level of lowest priority

Text Mode

Every character position is defined by two bytes:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
I	R	G	B	I	R	G	B	ASCII CHAR CODE							

Background Foreground
Attribute Byte

If neither the underline or blinking capabilities are specified, the color choice for foreground and background with a color monitor is:

Black	Red
Blue	Magenta
Green	Brown
Cyan	Light Gray
Dark Gray	Light Red
Light Blue	Light Magenta
Light Green	Yellow
Light Cyan	White

With a monochrome monitor, the color choice is:

Black
Darkest Gray
.
Lightest Gray
.
White

The codes for common monochrome choices follow:

normal	00001111
reverse video	11110000
non-display black	00000000
non-display white	11111111

When the blinking capability is specified in Mode Select 1, then Bit 15 of the attribute byte specifies whether the character blinks.

If the underline capability is specified in the Mode Select Register 2, then Bit 11 specifies whether the character is underlined.

The first position in the left-hand corner of the screen is defined in the first two bytes of memory starting at B0000. The next position, one column to the right, is defined in the next two bytes of memory at B0002. The first character in the next row follows immediately after the definition for the last character in the first row. For 80 column X 25 rows, memory looks like this:

B0000	Row 1
B00A0	Row 2
B0140	Row 3
	.
	.
	.
	.
B0E60	Row 24
B0F00	Row 25

The 80 column display uses 4K of RAM and the 40 column display uses 2K of RAM. The rest of the 32K is used for multiple screen images called pages. There are either 16 or eight pages available.

Graphics Mode

In graphics mode, the display screen is a grid of pixels, the smallest displayable unit on a video monitor. In medium resolution, there are 640 across and either 200 or 400 down.

In high resolution, each pixel is defined by one bit. Bit 7 of each byte defines the first pixel and bit 0 defines the last pixel to be displayed. The background color is always black and is displayed when the pixel is off. When the pixel is on, the foreground color is one of 16 shades of gray as defined in the Color Select Register Bits 0-2 and 3.

In medium resolution, each pixel is defined by two bits:

7 6	5 4	3 2	1 0	
1st	2nd	3rd	4th	PIXELS

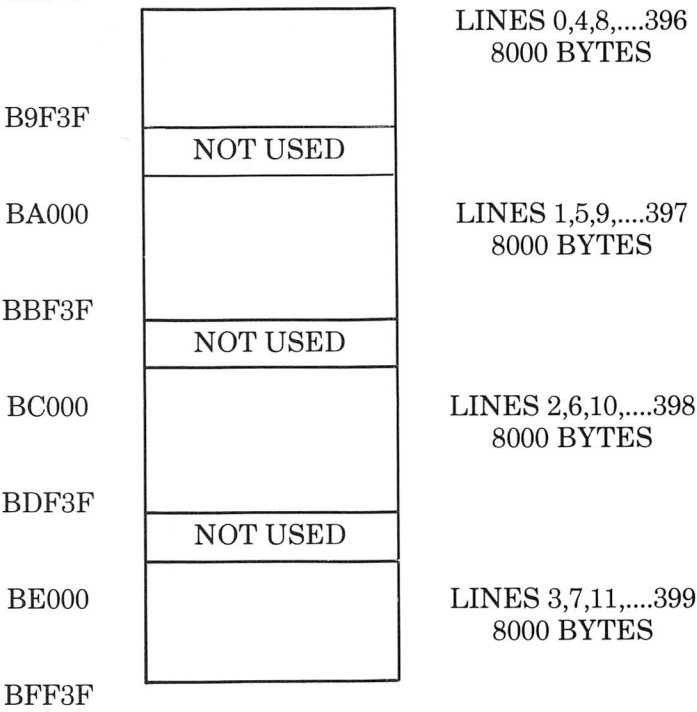
The value of two bits define one of four preselected colors.

0 — background color
1 — color 1
2 — color 2
3 — color 3

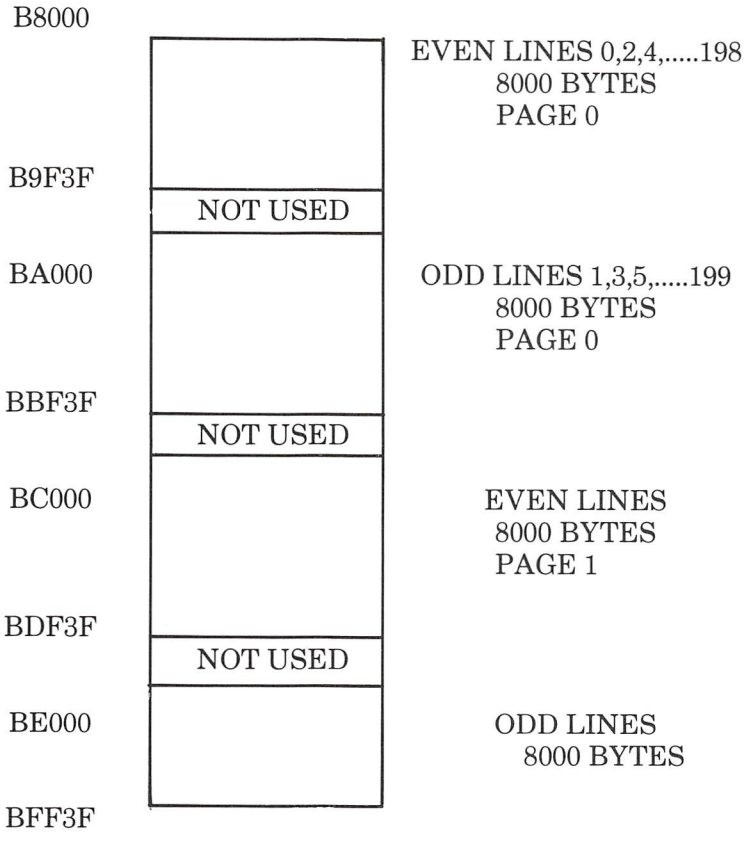
The background color is defined by Bits 0-3 in the Color Select Register. Colors 1, 2, and 3 are cyan, magenta and white if Bit 5 of the Color Select Register is zero and are green, red, and yellow if Bit 5 is one.

Unlike the text mode, rows of pixels do not follow one after another in memory. The following memory maps illustrate the layouts.

MEMORY MAP (640 X 400 GRAPHICS MODE)



MEMORY MAP (320 X 200 GRAPHICS MODE)



Functions

- INITIALIZE HD6845
  This function initializes the 16 registers of the HD6845 with predetermined values.
  OUTPUT: POINTER TO HD6845 REGISTER
  Number of HD6845 register
  HD6845 DATA REGISTER
  Value of HD6845 register
  (Repeat 16 times for each register)

- SET MODE
  Set different mode characteristics such as text or graphics, type of graphics, blinking character, etc.
  OUTPUT: MODE SELECT 1
  MODE SELECT 2

- SET COLOR TYPE
  Choose the different color or shades of gray to display.
  OUTPUT: COLOR SELECT REGISTER

- SET CURSOR SIZE
  Set starting and ending line for cursor.
  OUTPUT: POINTER TO HD6845 REGISTER
  0AH
  HD6845 DATA REGISTER
  start line
  POINTER TO HD6845 REGISTER
  0BH
  HD6845 DATA REGISTER
  end line

- SET CURSOR POSITION
  Set cursor to location in memory.
  OUTPUT: POINTER TO HD6845 REGISTER
  0EH
  HD6845 DATA REGISTER
  most significant byte of address
  POINTER TO H36845 REGISTER
  0FH
  HD6845 DATA REGISTER
  least significant byte of address

- READ CURSOR POSITION
  Read the current position of the cursor.
  INPUT: POINTER TO HD6845 REGISTER
  0EH
  HD6845 DATA REGISTER
  most significant byte of address
  POINTER TO H36845 REGISTER
  0FH
  HD6845 DATA REGISTER
  least significant byte of address

- SET ACTIVE PAGE
  Set the address of the current page to display.
  OUTPUT: POINTER TO HD6845 REGISTER
  0CH
  HD6845 DATA REGISTER
  most significant byte of address
  POINTER TO H36845 REGISTER
  0DH
  HD6845 DATA REGISTER
  least significant byte of address

Sequencing and Timing

There are two methods of communicating with the video display. One is with I/O commands. This method is used to set the modes of operation, the cursor position, the cursor size or the current active page.

Scrambler ROM

For programs which do not use the regular BIOS calls, the 6845 CRT controller requires some additional circuitry to translate the data sent to it by application programs into the appropriate data for the display controller. This circuitry (called the scrambler circuitry) converts the IBM6845 register values to the values which the display controller requires. A listing of these conversions follows. For additional information see the AT&T Personal Computer 6300 reference manual.

```
SCRAM7 (USED ON REV. P1-P2A)
Unrecognized command
*DIS ..350
0000  39 39 39 39 39 39 39 39  39 39 39 39 39 39 39 39  9999999999999999
0010  39 39 39 39 39 39 39 39  39 39 39 39 39 39 39 39  9999999999999999
0020  39 39 39 39 39 39 39 39  39 39 39 39 39 39 39 39  9999999999999999
0030  39 39 39 39 39 39 39 39  39 39 39 39 39 39 39 39  9999999999999999
0040  39 39 39 39 39 39 39 39  39 39 39 39 39 39 39 39  9999999999999999
0050  39 39 39 39 39 39 73 73  73 73 73 73 73 73 73 73  999999ssssssssss
0060  73 73 73 73 73 73 73 73  73 73 73 73 73 73 73 73  ssssssssssssssss
0070  73 73 73 73 73 73 73 73  73 73 73 73 73 73 73 73  ssssssssssssssss
0080  00 01 02 03 04 05 06 07  08 09 0A 0B 0C 0D 0E 0F  ................
0090  10 11 12 13 14 15 16 17  18 19 1A 1B 1C 1D 1E 1F  ................
00A0  20 21 22 23 24 25 26 27  28 29 2A 2B 2C 2D 2E 2F   !"#$%&'()*+,-./
00B0  30 31 32 33 34 35 36 37  38 39 3A 3B 3C 3D 3E 3F  0123456789:;<=>?
00C0  40 41 42 43 44 45 46 47  48 49 4A 4B 4C 4D 4E 4F  @ABCDEFGHIJKLMNO
00D0  50 51 52 53 54 55 56 57  58 59 5A 5B 5C 5D 5E 5F  PQRSTUVWXYZ[\]^_
00E0  60 61 62 63 64 65 66 67  68 69 6A 6B 6C 6D 6E 6F  `abcdefghijklmno
00F0  70 71 72 73 74 75 76 77  78 79 7A 7B 7C 7D 7E 7F  pqrstuvwxyz{|}~.
0100  00 01 02 03 04 05 06 07  08 09 0A 0B 0C 0D 0E 0F  ................
0110  10 11 12 13 14 15 16 17  18 19 1A 1B 1C 1D 1E 1F  ................
0120  20 21 22 23 24 25 26 27  28 29 2A 2B 2C 2D 2E 2F   !"#$%&'()*+,-./
0130  30 31 32 33 34 35 36 37  38 39 3A 3B 3C 3D 3E 3F  0123456789:;<=>?
0140  40 41 42 43 44 45 46 47  48 49 4A 4B 4C 4D 4E 4F  @ABCDEFGHIJKLMNO
0150  50 51 52 53 54 55 56 57  58 59 5A 5B 5C 5D 5E 5F  PQRSTUVWXYZ[\]^_
0160  60 61 62 63 64 65 66 67  68 69 6A 6B 6C 6D 6E 6F  `abcdefghijklmno
0170  70 71 72 73 74 75 76 77  78 79 7A 7B 7C 7D 7E 7F  pqrstuvwxyz{|}~.
0180  00 01 02 03 04 05 06 07  08 09 0A 0B 0C 0D 0E 0F  ................
0190  10 11 12 13 14 15 16 17  18 19 1A 1B 1C 1D 1E 1F  ................
01A0  20 21 22 23 24 25 26 27  28 29 2A 2B 2C 2D 2E 2F   !"#$%&'()*+,-./
01B0  30 31 32 33 34 35 36 37  38 39 3A 3B 3C 3D 3E 3F  0123456789:;<=>?
01C0  40 41 42 43 44 45 46 47  48 49 4A 4B 4C 4D 4E 4F  @ABCDEFGHIJKLMNO
01D0  50 51 52 53 54 55 56 57  58 59 5A 5B 5C 5D 5E 5F  PQRSTUVWXYZ[\]^_
01E0  60 61 62 63 64 65 66 67  68 69 6A 6B 6C 6D 6E 6F  `abcdefghijklmno
01F0  70 07 72 73 74 75 76 77  78 79 7A 7B 7C 7D 7E 7F  p.rstuvwxyz{|}~.
```

```
0200  02 03 04 05 06 07 08 09  0A 0B 0C 0D 0E 0F 10 11  ................
0210  12 13 14 15 16 17 18 19  1A 1B 1C 1D 1E 1F 20 21  .............. !
0220  22 23 24 25 26 27 28 29  2A 2B 2C 2D 2E 2F 30 31  "#$%&'()*+,-./01
0230  32 33 34 35 36 37 38 39  3A 3B 3C 3D 3E 3F 40 41  23456789:;<=>?@A
0240  42 43 44 45 46 47 48 49  4A 4B 4C 4D 4E 4F 50 51  BCDEFGHIJKLMNOPQ
0250  52 53 54 55 56 57 58 59  5A 5B 5C 5D 5E 5F 60 61  RSTUVWXYZ[\]^_`a
0260  62 63 64 65 66 67 68 69  6A 6B 6C 6D 6E 6F 70 71  bcdefghijklmnopq
0270  72 73 74 75 76 77 78 79  7A 7B 7C 7D 7E 7F 80 81  rstuvwxyz{|}~...
0280  00 01 02 03 04 05 06 07  08 09 0A 0B 0C 0D 0E 0F  ................
0290  10 11 12 13 14 15 16 17  18 19 1A 1B 1C 1D 1E 1F  ................
02A0  20 21 22 23 24 25 26 27  28 29 2A 2B 2C 2D 2E 2F   !"#$%&'()*+,-./
02B0  30 31 32 33 34 35 36 37  38 39 3A 3B 3C 3D 3E 3F  0123456789:;<=>?
02C0  40 41 42 43 44 45 46 47  48 49 4A 4B 4C 4D 4E 4F  @ABCDEFGHIJKLMNO
02D0  50 51 52 53 54 55 56 57  58 59 5A 5B 5C 5D 5E 5F  PQRSTUVWXYZ[\]^_
02E0  60 61 62 63 64 65 66 67  68 69 6A 6B 6C 6D 6E 6F  `abcdefghijklmno
02F0  70 71 72 73 74 75 76 77  78 79 7A 7B 7C 7D 7E 7F  pqrstuvwxyz{|}~.
0300  06 06 06 06 06 06 06 06  06 06 09 06 0C 0C 0C 0C  ................
0310  09 09 09 09 09 09 09 09  09 09 09 09 09 09 09 09  ................
0320  09 09 09 09 09 09 09 09  09 09 09 09 09 09 09 09  ................
0330  09 09 09 09 09 09 09 09  09 09 09 09 09 09 09 09  ................
0340  09 09 09 09 09 09 09 09  09 09 09 09 09 09 09 09  ................
0350  09 09 09 09 09 09 09 09  09 09 09 09 09 09 09 09  ................
0360  09 09 09 09 09 09 09 09  09 09 09 09 09 09 09 09  ................
0370  09 09 09 09 09 09 09 09  09 09 09 09 09 09 09 09  ................
0380  09 09 09 09 09 09 09 09  09 09 09 09 09 09 09 09  ................
0390  09 09 09 09 09 09 09 09  09 09 09 09 09 09 09 09  ................
03A0  09 09 09 09 09 09 09 09  09 09 09 09 09 09 09 09  ................
03B0  09 09 09 09 09 09 09 09  09 09 09 09 09 09 09 09  ................
03C0  09 09 09 09 09 09 09 09  09 09 09 09 09 09 09 09  ................
03D0  09 09 09 09 09 09 09 09  09 09 09 09 09 09 09 09  ................
03E0  09 09 09 09 09 09 09 09  09 09 09 09 09 09 09 09  ................
03F0  09 09 09 09 09 09 09 09  09 09 09 09 09 09 09 06  ................
0400  1A 1A 1A 1A 1A 1A 1A 1A  1A 1A 1A 1A 1A 1A 1A 1A  ................
0410  1A 1A 1A 1A 1A 1A 1A 1A  1A 1A 1A 1A 1A 1A 1A 1A  ................
0420  1A 1A 1A 1A 1A 1A 1A 1A  1A 1A 1A 1A 1A 1A 1A 1A  ................
0430  1A 1A 1A 1A 1A 1A 1A 1A  1A 1A 1A 1A 1A 1A 1A 1A  ................
0440  1A 6B 6B 6B 6B 6B 6B 6B  6B 6B 6B 6B 6B 6B 6B 6B  .kkkkkkkkkkkkkkk
0450  6B 6B 6B 6B 6B 6B 6B 6B  6B 6B 6B 6B 6B 6B 6B 6B  kkkkkkkkkkkkkkkk
0460  6B 6B 6B 6B 6B 6B 6B 6B  6B 6B 6B 6B 6B 6B 6B 6B  kkkkkkkkkkkkkkkk
0470  6B 6B 6B 6B 6B 6B 6B 6B  6B 6B 6B 6B 6B 6B 6B 6B  kkkkkkkkkkkkkkkk
0480  6B 6B 6B 6B 6B 6B 6B 6B  6B 6B 6B 6B 6B 6B 6B 6B  kkkkkkkkkkkkkkkk
0490  6B 6B 6B 6B 6B 6B 6B 6B  6B 6B 6B 6B 6B 6B 6B 6B  kkkkkkkkkkkkkkkk
04A0  6B 6B 6B 6B 6B 6B 6B 6B  6B 6B 6B 6B 6B 6B 6B 6B  kkkkkkkkkkkkkkkk
04B0  6B 6B 6B 6B 6B 6B 6B 6B  6B 6B 6B 6B 6B 6B 6B 6B  kkkkkkkkkkkkkkkk
04C0  6B 6B 6B 6B 6B 6B 6B 6B  6B 6B 6B 6B 6B 6B 6B 6B  kkkkkkkkkkkkkkkk
04D0  6B 6B 6B 6B 6B 6B 6B 6B  6B 6B 6B 6B 6B 6B 6B 6B  kkkkkkkkkkkkkkkk
04E0  6B 6B 6B 6B 6B 6B 6B 6B  6B 6B 6B 6B 6B 6B 6B 6B  kkkkkkkkkkkkkkkk
04F0  6B 6B 6B 6B 6B 6B 6B 6B  6B 6B 6B 6B 6B 6B 6B 6B  kkkkkkkkkkkkkkkk
0500  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
0510  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
0520  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
0530  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
```

```
0540  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
0550  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
0560  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
0570  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
0580  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
0590  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
05A0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
05B0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
05C0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
05D0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
05E0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
05F0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
0600  00 01 02 03 04 05 06 07  08 09 0A 0B 0C 0D 0E 0F  ................
0610  10 11 12 13 14 15 16 17  18 19 1A 1B 1C 1D 1E 1F  ................
0620  20 21 22 23 24 25 26 27  28 29 2A 2B 2C 2D 2E 2F   !"#$%&'()*+,-./
0630  30 31 32 33 34 35 36 37  38 39 3A 3B 3C 3D 3E 3F  0123456789:;<=>?
0640  40 41 42 43 44 45 46 47  48 49 4A 4B 4C 4D 4E 4F  @ABCDEFGHIJKLMNO
0650  50 51 52 53 54 55 56 57  58 59 5A 5B 5C 5D 5E 5F  PQRSTUVWXYZ[\]^_
0660  60 61 62 63 64 65 66 67  68 69 6A 6B 6C 6D 6E 6F  'abcdefghijklmno
0670  70 71 72 73 74 75 76 77  78 79 7A 7B 7C 7D 7E 7F  pqrstuvwxyz{|}~.
0680  80 81 82 83 84 85 86 87  88 89 8A 8B 8C 8D 8E 8F  ................
0690  90 91 92 93 94 95 96 97  98 99 9A 9B 9C 9D 9E 9F  ................
06A0  A0 A1 A2 A3 A4 A5 A6 A7  A8 A9 AA AB AC AD AE AF  ................
06B0  B0 B1 B2 B3 B4 B5 B6 B7  B8 B9 BA BB BC BD BE BF  ................
06C0  C0 C1 C2 C3 C4 C5 C6 C7  C8 C9 CA CB CC CD CE CF  ................
06D0  D0 D1 D2 D3 D4 D5 D6 D7  D8 D9 DA DB DC DD DE DF  ................
06E0  E0 E1 E2 E3 E4 E5 E6 E7  E8 E9 EA EB EC ED EE EF  ................
06F0  F0 F1 F2 F3 F4 F5 F6 F7  F8 F9 FA FB FC FD FE FF  ................
0700  19 19 19 19 19 19 19 19  19 19 19 19 19 19 19 19  ................
0710  19 19 19 19 19 19 19 19  19 19 19 19 19 19 19 19  ................
0720  19 19 19 19 19 19 19 19  19 19 19 19 19 19 19 19  ................
0730  19 19 19 19 19 19 19 19  19 19 19 19 19 19 19 19  ................
0740  19 64 64 64 64 64 64 64  64 64 64 64 64 64 64 64  .dddddddddddddd
0750  64 64 64 64 64 64 64 64  64 64 64 64 64 64 64 64  dddddddddddddddd
0760  64 64 64 64 64 64 64 64  64 64 64 64 64 64 64 64  dddddddddddddddd
0770  64 64 64 64 64 64 64 64  64 64 64 64 64 64 64 64  dddddddddddddddd
0780  64 64 64 64 64 64 64 64  64 64 64 64 64 64 64 64  dddddddddddddddd
0790  64 64 64 64 64 64 64 64  64 64 64 64 64 64 64 64  dddddddddddddddd
07A0  64 64 64 64 64 64 64 64  64 64 64 64 64 64 64 64  dddddddddddddddd
07B0  64 64 64 64 64 64 64 64  64 64 64 64 64 64 64 64  dddddddddddddddd
07C0  64 64 64 64 64 64 64 64  64 64 64 64 64 64 64 64  dddddddddddddddd
07D0  64 64 64 64 64 64 64 64  64 64 64 64 64 64 64 64  dddddddddddddddd
07E0  64 64 64 64 64 64 64 64  64 64 64 64 64 64 64 64  dddddddddddddddd
07F0  64 64 64 64 64 64 64 64  64 64 64 64 64 64 64 64  dddddddddddddddd
0800  00 00 00 03 00 00 00 00  00 00 00 00 00 00 00 00  ................
0810  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
0820  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
0830  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
0840  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
0850  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
0860  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
0870  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
```

```
0880  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
0890  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
08A0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
08B0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
08C0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
08D0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
08E0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
08F0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
0900  01 03 05 03 09 0B 0D 0F  11 13 15 17 19 1B 1D 0F  ................
0910  21 23 25 27 29 2B 2D 2F  31 33 35 37 39 3B 3D 3F  !#%')+-/13579;=?
0920  41 43 45 47 49 4B 4D 4F  51 53 55 57 59 5B 5D 5F  ACEGIKMOQSUWY[]_
0930  61 63 65 67 69 6B 6D 6F  71 73 75 77 79 7B 7D 7F  acegikmoqsuwy{}.
0940  81 83 85 87 89 8B 8D 8F  91 93 95 97 99 9B 9D 9F  ................
0950  A1 A3 A5 A7 A9 AB AD AF  B1 B3 B5 B7 B9 BB BD BF  ................
0960  C1 C3 C5 C7 C9 CB CD CF  D1 D3 D5 D7 D9 DB DD DF  ................
0970  E1 E3 E5 E7 E9 EB ED EF  F1 F3 F5 F7 F9 FB FD FF  ................
0980  01 03 05 07 09 0B 0D 0F  11 13 15 17 19 1B 1D 1F  ................
0990  21 23 25 27 29 2B 2D 2F  31 33 35 37 39 3B 3D 3F  !#%')+-/13579;=?
09A0  41 43 45 47 49 4B 4D 4F  51 53 55 57 59 5B 5D 5F  ACEGIKMOQSUWY[]_
09B0  61 63 65 67 69 6B 6D 6F  71 73 75 77 79 7B 7D 7F  acegikmoqsuwy{}.
09C0  81 83 85 87 89 8B 8D 8F  91 93 95 97 99 9B 9D 9F  ................
09D0  A1 A3 A5 A7 A9 AB AD AF  B1 B3 B5 B7 B9 BB BD BF  ................
09E0  C1 C3 C5 C7 C9 CB CD CF  D1 D3 D5 D7 D9 DB DD DF  ................
09F0  E1 E3 E5 E7 E9 EB ED EF  F1 F3 F5 F7 F9 FB FD FF  ................
0A00  00 02 04 06 08 0A 0C 0E  10 10 10 10 10 10 10 10  ................
0A10  10 10 10 10 10 10 10 10  10 10 10 10 10 10 10 10  ................
0A20  20 22 24 26 28 2A 2C 2E  30 30 30 30 30 30 30 30   "$&(*,.00000000
0A30  30 30 30 30 30 30 30 30  30 30 30 30 30 30 30 30  0000000000000000
0A40  40 42 44 46 48 4A 4C 4E  50 50 50 50 50 50 50 50  @BDFHJLNPPPPPPPP
0A50  50 50 50 50 50 50 50 50  50 50 50 50 50 50 50 50  PPPPPPPPPPPPPPPP
0A60  60 62 64 66 68 6A 6C 6E  70 70 70 70 70 70 70 70  'bdfhjlnpppppppp
0A70  70 70 70 70 70 70 70 70  70 70 70 70 70 70 70 70  pppppppppppppppp
0A80  00 02 04 06 08 0A 0C 0E  10 10 10 10 10 10 10 10  ................
0A90  10 10 10 10 10 10 10 10  10 10 10 10 10 10 10 10  ................
0AA0  20 22 24 26 28 2A 2C 2E  30 30 30 30 30 30 30 30   "$&(*,.00000000
0AB0  30 30 30 30 30 30 30 30  30 30 30 30 30 30 30 30  0000000000000000
0AC0  40 42 44 46 48 4A 4C 4E  50 50 50 50 50 50 50 50  @BDFHJLNPPPPPPPP
0AD0  50 50 50 50 50 50 50 50  50 50 50 50 50 50 50 50  PPPPPPPPPPPPPPPP
0AE0  60 62 64 66 68 6A 6C 6E  70 70 70 70 70 70 70 70  'bdfhjlnpppppppp
0AF0  70 70 70 70 70 70 70 70  70 70 70 70 70 70 70 70  pppppppppppppppp
0B00  00 02 04 06 08 0A 0C 0E  10 12 14 16 18 1A 1C 1E  ................
0B10  20 22 24 26 28 2A 2C 2E  30 32 34 36 38 3A 3C 3E   "$&(*,.02468:< >
0B20  40 42 44 46 48 4A 4C 4E  50 52 54 56 58 5A 5C 5E  @BDFHJLNPRTVXZ\
0B30  60 62 64 66 68 6A 6C 6E  70 72 74 76 78 7A 7C 7E  'bdfhjlnprtvxz ~
0B40  80 82 84 86 88 8A 8C 8E  90 92 94 96 98 9A 9C 9E  ................
0B50  A0 A2 A4 A6 A8 AA AC AE  B0 B2 B4 B6 B8 BA BC BE  ................
0B60  C0 C2 C4 C6 C8 CA CC CE  D0 D2 D4 D6 D8 DA DC DE  ................
0B70  E0 E2 E4 E6 E8 EA EC EE  F0 F2 F4 F6 F8 FA FC FE  ................
0B80  00 02 04 06 08 0A 0C 0E  10 12 14 16 18 1A 1C 1E  ................
0B90  20 22 24 26 28 2A 2C 2E  30 32 34 36 38 3A 3C 3E   "$&(*,.02468:< >
0BA0  40 42 44 46 48 4A 4C 4E  50 52 54 56 58 5A 5C 5E  @BDFHJLNPRTVXZ\
0BB0  60 62 64 66 68 6A 6C 6E  70 72 74 76 78 7A 7B 7E  'bdfhjlnprtvxz ~
```

```
0BC0  80 82 84 86 88 8A 8C 8E  90 92 94 96 98 9A 9C 9E  ................
0BD0  A0 A2 A4 A6 A8 AA AC AE  B0 B2 B4 B6 B8 BA BC BE  ................
0BE0  C0 C2 C4 C6 C8 CA CC CE  D0 D2 D4 D6 D8 DA DC DE  ................
0BF0  E0 E2 E4 E6 E8 EA EC EE  F0 F2 F4 F6 F8 FA FC FE  ................
0C00  00 01 02 03 04 05 06 07  08 09 0A 0B 0C 0D 0E 0F  ................
0C10  10 11 12 13 14 15 16 17  18 19 1A 1B 1C 1D 1E 1F  ................
0C20  20 21 22 23 24 25 26 27  28 29 2A 2B 2C 2D 2E 2F   !"#$%&'()*+,-./
0C30  30 31 32 33 34 35 36 37  38 39 3A 3B 3C 3D 3E 3F  0123456789:;<=>?
0C40  40 41 42 43 44 45 46 47  48 49 4A 4B 4C 4D 4E 4F  @ABCDEFGHIJKLMNO
0C50  50 51 52 53 54 55 56 57  58 59 5A 5B 5C 5D 5E 5F  PQRSTUVWXYZ[\]^_
0C60  60 61 62 63 64 65 66 67  68 69 6A 6B 6C 6D 6E 6F  'abcdefghijklmno
0C70  70 71 72 73 74 75 76 77  78 79 7A 7B 7C 7D 7E 7F  pqrstuvwxyz{|}~.
0C80  80 81 82 83 84 85 86 87  88 89 8A 8B 8C 8D 8E 8F  ................
0C90  90 91 92 93 94 95 96 97  98 99 9A 9B 9C 9D 9E 9F  ................
0CA0  A0 A1 A2 A3 A4 A5 A6 A7  A8 A9 AA AB AC AD AE AF  ................
0CB0  B0 B1 B2 B3 B4 B5 B6 B7  B8 B9 BA BB BC BD BE BF  ................
0CC0  C0 C1 C2 C3 C4 C5 C6 C7  C8 C9 CA CB CC CD CE CF  ................
0CD0  D0 D1 D2 D3 D4 D5 D6 D7  D8 D9 DA DB DC DD DE DF  ................
0CE0  E0 E1 E2 E3 E4 E5 E6 E7  E8 E9 EA EB EC ED EE EF  ................
0CF0  F0 F1 F2 F3 F4 F5 F6 F7  F8 F9 FA FB FC FD FE FF  ................
0D00  00 01 02 03 04 05 06 07  08 09 0A 0B 0C 0D 0E 0F  ................
0D10  10 11 12 13 14 15 16 17  18 19 1A 1B 1C 1D 1E 1F  ................
0D20  20 21 22 23 24 25 26 27  28 29 2A 2B 2C 2D 2E 2F   !"#$%&'()*+,-./
0D30  30 31 32 33 34 35 36 37  38 39 3A 3B 3C 3D 3E 3F  0123456789:;<=>?
0D40  40 41 42 43 44 45 46 47  48 49 4A 4B 4C 4D 4E 4F  @ABCDEFGHIJKLMNO
0D50  50 51 52 53 54 55 56 57  58 59 5A 5B 5C 5D 5E 5F  PQRSTUVWXYZ[\]^_
0D60  60 61 62 63 64 65 66 67  68 69 6A 6B 6C 6D 6E 6F  'abcdefghijklmno
0D70  70 71 72 73 74 75 76 77  78 79 7A 7B 7C 7D 7E 7F  pqrstuvwxyz{|}~.
0D80  80 81 82 83 84 85 86 87  88 89 8A 8B 8C 8D 8E 8F  ................
0D90  90 91 92 93 94 95 96 97  98 99 9A 9B 9C 9D 9E 9F  ................
0DA0  A0 A1 A2 A3 A4 A5 A6 A7  A8 A9 AA AB AC AD AE AF  ................
0DB0  B0 B1 B2 B3 B4 B5 B6 B7  B8 B9 BA BB BC BD BE BF  ................
0DC0  C0 C1 C2 C3 C4 C5 C6 C7  C8 C9 CA CB CC CD CE CF  ................
0DD0  D0 D1 D2 D3 D4 D5 D6 D7  D8 D9 DA DB DC DD DE DF  ................
0DE0  E0 E1 E2 E3 E4 E5 E6 E7  E8 E9 EA EB EC ED EE EF  ................
0DF0  F0 F1 F2 F3 F4 F5 F6 F7  F8 F9 FA FB FC FD FE FF  ................
0E00  00 01 02 03 04 05 06 07  08 09 0A 0B 0C 0D 0E 0F  ................
0E10  10 11 12 13 14 15 16 17  18 19 1A 1B 1C 1D 1E 1F  ................
0E20  20 21 22 23 24 25 26 27  28 29 2A 2B 2C 2D 2E 2F   !"#$%&'()*+,-./
0E30  30 31 32 33 34 35 36 37  38 39 3A 3B 3C 3D 3E 3F  0123456789:;<=>?
0E40  40 41 42 43 44 45 46 47  48 49 4A 4B 4C 4D 4E 4F  @ABCDEFGHIJKLMNO
0E50  50 51 52 53 54 55 56 57  58 59 5A 5B 5C 5D 5E 5F  PQRSTUVWXYZ[\]^_
0E60  60 61 62 63 64 65 66 67  68 69 6A 6B 6C 6D 6E 6F  'abcdefghijklmno
0E70  70 71 72 73 74 75 76 77  78 79 7A 7B 7C 7D 7E 7F  pqrstuvwxyz{|}~.
0E80  80 81 82 83 84 85 86 87  88 89 8A 8B 8C 8D 8E 8F  ................
0E90  90 91 92 93 94 95 96 97  98 99 9A 9B 9C 9D 9E 9F  ................
0EA0  A0 A1 A2 A3 A4 A5 A6 A7  A8 A9 AA AB AC AD AE AF  ................
0EB0  B0 B1 B2 B3 B4 B5 B6 B7  B8 B9 BA BB BC BD BE BF  ................
0EC0  C0 C1 C2 C3 C4 C5 C6 C7  C8 C9 CA CB CC CD CE CF  ................
0ED0  D0 D1 D2 D3 D4 D5 D6 D7  D8 D9 DA DB DC DD DE DF  ................
0EE0  E0 E1 E2 E3 E4 E5 E6 E7  E8 E9 EA EB EC ED EE EF  ................
0EF0  F0 F1 F2 F3 F4 F5 F6 F7  F8 F9 FA FB FC FD FE FF  ................
```

```
0F00   00 01 02 03 04 05 06 07   08 09 0A 0B 0C 0D 0E 0F   ................
0F10   10 11 12 13 14 15 16 17   18 19 1A 1B 1C 1D 1E 1F   ................
0F20   20 21 22 23 24 25 26 27   28 29 2A 2B 2C 2D 2E 2F    !"#$%&'()*+,-./
0F30   30 31 32 33 34 35 36 37   38 39 3A 3B 3C 3D 3E 3F   0123456789:;<=>?
0F40   40 41 42 43 44 45 46 47   48 49 4A 4B 4C 4D 4E 4F   @ABCDEFGHIJKLMNO
0F50   50 51 52 53 54 55 56 57   58 59 5A 5B 5C 5D 5E 5F   PQRSTUVWXYZ[\]^_
0F60   60 61 62 63 64 65 66 67   68 69 6A 6B 6C 6D 6E 6F   'abcdefghijklmno
0F70   70 71 72 73 74 75 76 77   78 79 7A 7B 7C 7D 7E 7F   pqrstuvwxyz{|}~.
0F80   80 81 82 83 84 85 86 87   88 89 8A 8B 8C 8D 8E 8F   ................
0F90   90 91 92 93 94 95 96 97   98 99 9A 9B 9C 9D 9E 9F   ................
0FA0   A0 A1 A2 A3 A4 A5 A6 A7   A8 A9 AA AB AC AD AE AF   ................
0FB0   B0 B1 B2 B3 B4 B5 B6 B7   B8 B9 BA BB BC BD BE BF   ................
0FC0   C0 C1 C2 C3 C4 C5 C6 C7   C8 C9 CA CB CC CD CE CF   ................
0FD0   D0 D1 D2 D3 D4 D5 D6 D7   D8 D9 DA DB DC DD DE DF   ................
0FE0   E0 E1 E2 E3 E4 E5 E6 E7   E8 E9 EA EB EC ED EE EF   ................
0FF0   F0 F1 F2 F3 F4 F5 F6 F7   F8 F9 FA FB FC FD FE FF   ................
At end of current range
*
```

```
SCRAM8B (USED ON REV. P3 AND LATER)
Unrecognized command
*DIS ..350
0000   39 39 39 39 39 39 39 39   39 39 39 39 39 39 39 39   9999999999999999
0010   39 39 39 39 39 39 39 39   39 39 39 39 39 39 39 39   9999999999999999
0020   39 39 39 39 39 39 39 39   39 39 39 39 39 39 39 39   9999999999999999
0030   39 39 39 39 39 39 39 39   39 39 39 39 39 39 39 39   9999999999999999
0040   39 39 39 39 39 39 39 39   39 39 39 39 39 39 39 39   9999999999999999
0050   39 39 39 39 39 39 73 73   73 73 73 73 73 73 73 73   999999ssssssssss
0060   73 73 73 73 73 73 73 73   73 73 73 73 73 73 73 73   ssssssssssssssss
0070   73 73 73 73 73 73 73 73   73 73 73 73 73 73 73 73   ssssssssssssssss
0080   00 01 02 03 04 05 06 07   08 09 0A 0B 0C 0D 0E 0F   ................
0090   10 11 12 13 14 15 16 17   18 19 1A 1B 1C 1D 1E 1F   ................
00A0   20 21 22 23 24 25 26 27   28 29 2A 2B 2C 2D 2E 2F    !"#$%&'()*+,-./
00B0   30 31 32 33 34 35 36 37   38 39 3A 3B 3C 3D 3E 3F   0123456789:;<=>?
00C0   40 41 42 43 44 45 46 47   48 49 4A 4B 4C 4D 4E 4F   @ABCDEFGHIJKLMNO
00D0   50 51 52 53 54 55 56 57   58 59 5A 5B 5C 5D 5E 5F   PQRSTUVWXYZ[\]^_
00E0   60 61 62 63 64 65 66 67   68 69 6A 6B 6C 6D 6E 6F   'abcdefghijklmno
00F0   70 71 72 73 74 75 76 77   78 79 7A 7B 7C 7D 7E 7F   pqrstuvwxyz{|}~.
0100   00 01 02 03 04 05 06 07   08 09 0A 0B 0C 0D 0E 0F   ................
0110   10 11 12 13 14 15 16 17   18 19 1A 1B 1C 1D 1E 1F   ................
0120   20 21 22 23 24 25 26 27   28 29 2A 2B 2C 2D 2E 2F    !"#$%&'()*+,-./
0130   30 31 32 33 34 35 36 37   38 39 3A 3B 3C 3D 3E 3F   0123456789:;<=>?
0140   40 41 42 43 44 45 46 47   48 49 4A 4B 4C 4D 4E 4F   @ABCDEFGHIJKLMNO
0150   50 51 52 53 54 55 56 57   58 59 5A 5B 5C 5D 5E 5F   PQRSTUVWXYZ[\]^_
0160   60 61 62 63 64 65 66 67   68 69 6A 6B 6C 6D 6E 6F   'abcdefghijklmno
0170   70 71 72 73 74 75 76 77   78 79 7A 7B 7C 7D 7E 7F   pqrstuvwxyz{|}~.
0180   80 81 82 83 84 85 86 87   88 89 8A 8B 8C 8D 8E 8F   ................
0190   90 91 92 93 94 95 96 97   98 99 9A 9B 9C 9D 9E 9F   ................
01A0   A0 A1 A2 A3 A4 A5 A6 A7   A8 A9 AA AB AC AD AE AF   ................
01B0   B0 B1 B2 B3 B4 B5 B6 B7   B8 B9 BA BB BC BD BE BF   ................
```

```
01C0  C0 C1 C2 C3 C4 C5 C6 C7  C8 C9 CA CB CC CD CE CF  ................
01D0  D0 D1 D2 D3 D4 D5 D6 D7  D8 D9 DA DB DC DD DE DF  ................
01E0  E0 E1 E2 E3 E4 E5 E6 E7  E8 E9 EA EB EC ED EE EF  ................
01F0  F0 F1 F2 F3 F4 F5 F6 F7  F8 F9 FA FB FC FD FE FF  ................
0200  02 03 04 05 06 07 08 09  0A 0B 0C 0D 0E 0F 10 11  ................
0210  12 13 14 15 16 17 18 19  1A 1B 1C 1D 1E 1F 20 21  ..............  !
0220  22 23 24 25 26 27 28 29  2A 2B 2C 2D 2E 2F 30 31  "#$%&'()*+,-./01
0230  32 33 34 35 36 37 38 39  3A 3B 3C 3D 3E 3F 40 41  23456789:;<=>?@A
0240  42 43 44 45 46 47 48 49  4A 4B 4C 4D 4E 4F 50 51  BCDEFGHIJKLMNOPQ
0250  52 53 54 55 56 57 58 59  5A 5B 5C 5D 5E 5F 60 61  RSTUVWXYZ[\]^_'a
0260  62 63 64 65 66 67 68 69  6A 6B 6C 6D 6E 6F 70 71  bcdefghijklmnopq
0270  72 73 74 75 76 77 78 79  7A 7B 7C 7D 7E 7F 80 81  rstuvuxyz{|}~...
0280  00 01 02 03 04 05 06 07  08 09 0A 0B 0C 0D 0E 0F  ................
0290  10 11 12 13 14 15 16 17  18 19 1A 1B 1C 1D 1E 1F  ................
02A0  20 21 22 23 24 25 26 27  28 29 2A 2B 2C 2D 2E 2F   !"#$%&'()*+,-./
02B0  30 31 32 33 34 35 36 37  38 39 3A 3B 3C 3D 3E 3F  0123456789:;<=>?
02C0  40 41 42 43 44 45 46 47  48 49 4A 4B 4C 4D 4E 4F  @ABCDEFGHIJKLMNO
02D0  50 51 52 53 54 55 56 57  58 59 5A 5B 5C 5D 5E 5F  PQRSTUVWXYZ[\]^_
02E0  60 61 62 63 64 65 66 67  68 69 6A 6B 6C 6D 6E 6F  'abcdefghijklmno
02F0  70 71 72 73 74 75 76 77  78 79 7A 7B 7C 7D 7E 7F  pqrstuvuxyz{|}~.
0300  0E 0E 0E 0E 0E 0E 0E 0E  0E 0E 0E 0E 0E 0E 0E 0E  ................
0310  0E 0E 0E 0E 0E 0E 0E 0E  0E 0E 0E 0E 0E 0E 0E 0E  ................
0320  0E 0E 0E 0E 0E 0E 0E 0E  0E 0E 0E 0E 0E 0E 0E 0E  ................
0330  0E 0E 0E 0E 0E 0E 0E 0E  0E 0E 0E 0E 0E 0E 0E 0E  ................
0340  0E 0E 0E 0E 0E 0E 0E 0E  0E 0E 0E 0E 0E 0E 0E 0E  ................
0350  0E 0E 0E 0E 0E 0E 0E 0E  0E 0E 0E 0E 0E 0E 0E 0E  ................
0360  0E 0E 0E 0E 0E 0E 0E 0E  0E 0E 0E 0E 0E 0E 0E 0E  ................
0370  0E 0E 0E 0E 0E 0E 0E 0E  0E 0E 0E 0E 0E 0E 0E 0E  ................
0380  0E 0E 0E 0E 0E 0E 0E 0E  0E 0E 0E 0E 0E 0E 0E 0E  ................
0390  0E 0E 0E 0E 0E 0E 0E 0E  0E 0E 0E 0E 0E 0E 0E 0E  ................
03A0  0E 0E 0E 0E 0E 0E 0E 0E  0E 0E 0E 0E 0E 0E 0E 0E  ................
03B0  0E 0E 0E 0E 0E 0E 0E 0E  0E 0E 0E 0E 0E 0E 0E 0E  ................
03C0  0E 0E 0E 0E 0E 0E 0E 0E  0E 0E 0E 0E 0E 0E 0E 0E  ................
03D0  0E 0E 0E 0E 0E 0E 0E 0E  0E 0E 0E 0E 0E 0E 0E 0E  ................
03E0  0E 0E 0E 0E 0E 0E 0E 0E  0E 0E 0E 0E 0E 0E 0E 0E  ................
03F0  0E 0E 0E 0E 0E 0E 0E 0E  0E 0E 0E 0E 0E 0E 0E 0E  ................
0400  1A 1A 1A 1A 1A 1A 1A 1A  1A 1A 1A 1A 1A 1A 1A 1A  ................
0410  1A 1A 1A 1A 1A 1A 1A 1A  1A 1A 1A 1A 1A 1A 1A 1A  ................
0420  1A 1A 1A 1A 1A 1A 1A 1A  1A 1A 1A 1A 1A 1A 1A 1A  ................
0430  1A 1A 1A 1A 1A 1A 1A 1A  1A 1A 1A 1A 1A 1A 1A 1A  ................
0440  1A 6B 6B 6B 6B 6B 6B 6B  6B 6B 6B 6B 6B 6B 6B 6B  .kkkkkkkkkkkkkkk
0450  6B 6B 6B 6B 6B 6B 6B 6B  6B 6B 6B 6B 6B 6B 6B 6B  kkkkkkkkkkkkkkkk
0460  6B 6B 6B 6B 6B 6B 6B 6B  6B 6B 6B 6B 6B 6B 6B 6B  kkkkkkkkkkkkkkkk
0470  6B 6B 6B 6B 6B 6B 6B 6B  6B 6B 6B 6B 6B 6B 6B 6B  kkkkkkkkkkkkkkkk
0480  00 01 02 03 04 05 06 07  08 09 0A 0B 0C 0D 0E 0F  ................
0490  10 11 12 13 14 15 16 17  18 19 1A 1B 1C 1D 1E 1F  ................
04A0  20 21 22 23 24 25 26 27  28 29 2A 2B 2C 2D 2E 2F   !"#$%&'()*+,-./
04B0  30 31 32 33 34 35 36 37  38 39 3A 3B 3C 3D 3E 3F  0123456789:;<=>?
04C0  40 41 42 43 44 45 46 47  48 49 4A 4B 4C 4D 4E 4F  @ABCDEFGHIJKLMNO
04D0  50 51 52 53 54 55 56 57  58 59 5A 5B 5C 5D 5E 5F  PQRSTUVWXYZ[\]^_
04E0  60 61 62 63 64 65 66 67  68 69 6A 6B 6C 6D 6E 6F  'abcdefghijklmno
04F0  70 71 72 73 74 75 76 77  78 79 7A 7B 7C 7D 7E 7F  pqrstuvuxyz{|}~.
```

```
0500  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
0510  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
0520  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
0530  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
0540  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
0550  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
0560  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
0570  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
0580  00 01 02 03 04 05 06 07  08 09 0A 0B 0C 0D 0E 0F  ................
0590  10 11 12 13 14 15 16 17  18 19 1A 1B 1C 1D 1E 1F  ................
05A0  20 21 22 23 24 25 26 27  28 29 2A 2B 2C 2D 2E 2F   !"#$%&'()*+,-./
05B0  30 31 32 33 34 35 36 37  38 39 3A 3B 3C 3D 3E 3F  0123456789:;<=>?
05C0  40 41 42 43 44 45 46 47  48 49 4A 4B 4C 4D 4E 4F  @ABCDEFGHIJKLMNO
05D0  50 51 52 53 54 55 56 57  58 59 5A 5B 5C 5D 5E 5F  PQRSTUVWXYZ[\]^_
05E0  60 61 62 63 64 65 66 67  68 69 6A 6B 6C 6D 6E 6F  `abcdefghijklmno
05F0  70 71 72 73 74 75 76 77  78 79 7A 7B 7C 7D 7E 7F  pqrstuvwxyz{|}~.
0600  00 01 02 03 04 05 06 07  08 09 0A 0B 0C 0D 0E 0F  ................
0610  10 11 12 13 14 15 16 17  18 19 1A 1B 1C 1D 1E 1F  ................
0620  20 21 22 23 24 25 26 27  28 29 2A 2B 2C 2D 2E 2F   !"#$%&'()*+,-./
0630  30 31 32 33 34 35 36 37  38 39 3A 3B 3C 3D 3E 3F  0123456789:;<=>?
0640  40 41 42 43 44 45 46 47  48 49 4A 4B 4C 4D 4E 4F  @ABCDEFGHIJKLMNO
0650  50 51 52 53 54 55 56 57  58 59 5A 5B 5C 5D 5E 5F  PQRSTUVWXYZ[\]^_
0660  60 61 62 63 64 65 66 67  68 69 6A 6B 6C 6D 6E 6F  `abcdefghijklmno
0670  70 71 72 73 74 75 76 77  78 79 7A 7B 7C 7D 7E 7F  pqrstuvwxyz{|}~.
0680  80 81 82 83 84 85 86 87  88 89 8A 8B 8C 8D 8E 8F  ................
0690  90 91 92 93 94 95 96 97  98 99 9A 9B 9C 9D 9E 9F  ................
06A0  A0 A1 A2 A3 A4 A5 A6 A7  A8 A9 AA AB AC AD AE AF  ................
06B0  B0 B1 B2 B3 B4 B5 B6 B7  B8 B9 BA BB BC BD BE BF  ................
06C0  C0 C1 C2 C3 C4 C5 C6 C7  C8 C9 CA CB CC CD CE CF  ................
06D0  D0 D1 D2 D3 D4 D5 D6 D7  D8 D9 DA DB DC DD DE DF  ................
06E0  E0 E1 E2 E3 E4 E5 E6 E7  E8 E9 EA EB EC ED EE EF  ................
06F0  F0 F1 F2 F3 F4 F5 F6 F7  F8 F9 FA FB FC FD FE FF  ................
0700  19 19 19 19 19 19 19 19  19 19 19 19 19 19 19 19  ................
0710  19 19 19 19 19 19 19 19  19 19 19 19 19 19 19 19  ................
0720  19 19 19 19 19 19 19 19  19 19 19 19 19 19 19 19  ................
0730  19 19 19 19 19 19 19 19  19 19 19 19 19 19 19 19  ................
0740  19 64 64 64 64 64 64 64  64 64 64 64 64 64 64 64  .dddddddddddddd
0750  64 64 64 64 64 64 64 64  64 64 64 64 64 64 64 64  dddddddddddddddd
0760  64 64 64 64 64 64 64 64  64 64 64 64 64 64 64 64  dddddddddddddddd
0770  64 64 64 64 64 64 64 64  64 64 64 64 64 64 64 64  dddddddddddddddd
0780  00 01 02 03 04 05 06 07  08 09 0A 0B 0C 0D 0E 0F  ................
0790  10 11 12 13 14 15 16 17  18 19 1A 1B 1C 1D 1E 1F  ................
07A0  20 21 22 23 24 25 26 27  28 29 2A 2B 2C 2D 2E 2F   !"#$%&'()*+,-./
07B0  30 31 32 33 34 35 36 37  38 39 3A 3B 3C 3D 3E 3F  0123456789:;<=>?
07C0  40 41 42 43 44 45 46 47  48 49 4A 4B 4C 4D 4E 4F  @ABCDEFGHIJKLMNO
07D0  50 51 52 53 54 55 56 57  58 59 5A 5B 5C 5D 5E 5F  PQRSTUVWXYZ[\]^_
07E0  60 61 62 63 64 65 66 67  68 69 6A 6B 6C 6D 6E 6F  `abcdefghijklmno
07F0  70 71 72 73 74 75 76 77  78 79 7A 7B 7C 7D 7E 7F  pqrstuvwxyz{|}~.
0800  00 00 00 03 00 00 00 00  00 00 00 00 00 00 00 00  ................
0810  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
0820  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
0830  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
```

```
0840  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
0850  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
0860  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
0870  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
0880  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
0890  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
08A0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
08B0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
08C0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
08D0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
08E0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
08F0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
0900  01 03 05 03 09 0B 0D 0F  11 13 15 17 19 1B 1D 0F  ................
0910  21 23 25 27 29 2B 2D 2F  31 33 35 37 39 3B 3D 3F  !#%')+-/13579;=?
0920  41 43 45 47 49 4B 4D 4F  51 53 55 57 59 5B 5D 5F  ACEGIKMOQSUWY[]_
0930  61 63 65 67 69 6B 6D 6F  71 73 75 77 79 7B 7D 7F  acegikmoqsuwy{}.
0940  81 83 85 87 89 8B 8D 8F  91 93 95 97 99 9B 9D 9F  ................
0950  A1 A3 A5 A7 A9 AB AD AF  B1 B3 B5 B7 B9 BB BD BF  ................
0960  C1 C3 C5 C7 C9 CB CD CF  D1 D3 D5 D7 D9 DB DD DF  ................
0970  E1 E3 E5 E7 E9 EB ED EF  F1 F3 F5 F7 F9 FB FD FF  ................
0980  00 01 02 03 04 05 06 07  08 09 0A 0B 0C 0D 0E 0F  ................
0990  10 11 12 13 14 15 16 17  18 19 1A 1B 1C 1D 1E 1F  ................
09A0  20 21 22 23 24 25 26 27  28 29 2A 2B 2C 2D 2E 2F   !"#$%&'()*+,-./
09B0  30 31 32 33 34 35 36 37  38 39 3A 3B 3C 3D 3E 3F  0123456789:;<=>?
09C0  40 41 42 43 44 45 46 47  48 49 4A 4B 4C 4D 4E 4F  @ABCDEFGHIJKLMNO
09D0  50 51 52 53 54 55 56 57  58 59 5A 5B 5C 5D 5E 5F  PQRSTUVWXYZ[\]^_
09E0  60 61 62 63 64 65 66 67  68 69 6A 6B 6C 6D 6E 6F  'abcdefghijklmno
09F0  70 71 72 73 74 75 76 77  78 79 7A 7B 7C 7D 7E 7F  pqrstuvwxyz{|}~.
0A00  00 02 04 06 08 0A 0C 0E  10 10 10 10 10 10 10 10  ................
0A10  10 10 10 10 10 10 10 10  10 10 10 10 10 10 10 10  ................
0A20  20 22 24 26 28 2A 2C 2E  30 30 30 30 30 30 30 30   "$&(*,.00000000
0A30  30 30 30 30 30 30 30 30  30 30 30 30 30 30 30 30  0000000000000000
0A40  40 42 44 46 48 4A 4C 4E  50 50 50 50 50 50 50 50  @BDFHJLNPPPPPPPP
0A50  50 50 50 50 50 50 50 50  50 50 50 50 50 50 50 50  PPPPPPPPPPPPPPPP
0A60  60 62 64 66 68 6A 6C 6E  70 70 70 70 70 70 70 70  'bdfhjlnpppppppp
0A70  70 70 70 70 70 70 70 70  70 70 70 70 70 70 70 70  pppppppppppppppp
0A80  00 01 02 03 04 05 06 07  08 09 0A 0B 0C 0D 0E 0F  ................
0A90  10 11 12 13 14 15 16 17  18 19 1A 1B 1C 1D 1E 1F  ................
0AA0  20 21 22 23 24 25 26 27  28 29 2A 2B 2C 2D 2E 2F   !"#$%&'()*+,-./
0AB0  30 31 32 33 34 35 36 37  38 39 3A 3B 3C 3D 3E 3F  0123456789:;<=>?
0AC0  40 41 42 43 44 45 46 47  48 49 4A 4B 4C 4D 4E 4F  @ABCDEFGHIJKLMNO
0AD0  50 51 52 53 54 55 56 57  58 59 5A 5B 5C 5D 5E 5F  PQRSTUVWXYZ[\]^_
0AE0  60 61 62 63 64 65 66 67  68 69 6A 6B 6C 6D 6E 6F  'abcdefghijklmno
0AF0  70 71 72 73 74 75 76 77  78 79 7A 7B 7C 7D 7E 7F  pqrstuvwxyz{|}~.
0B00  00 02 04 06 08 0A 0C 0E  10 12 14 16 18 1A 1C 1E  ................
0B10  20 22 24 26 28 2A 2C 2E  30 32 34 36 38 3A 3C 3E   "$&(*,.02468:<>
0B20  40 42 44 46 48 4A 4C 4E  50 52 54 56 58 5A 5C 5E  @BDFHJLNPRTVXZ\
0B30  60 62 64 66 68 6A 6C 6E  70 72 74 76 78 7A 7B 7E  'bdfhjlnprtvxz {~
0B40  80 82 84 86 88 8A 8C 8E  90 92 94 96 98 9A 9C 9E  ................
0B50  A0 A2 A4 A6 A8 AA AC AE  B0 B2 B4 B6 B8 BA BC BE  ................
0B60  C0 C2 C4 C6 C8 CA CC CE  D0 D2 D4 D6 D8 DA DC DE  ................
0B70  E0 E2 E4 E6 E8 EA EC EE  F0 F2 F4 F6 F8 FA FC FE  ................
```

```
0B80  00 01 02 03 04 05 06 07  08 09 0A 0B 0C 0D 0E 0F  ................
0B90  10 11 12 13 14 15 16 17  18 19 1A 1B 1C 1D 1E 1F  ................
0BA0  20 21 22 23 24 25 26 27  28 29 2A 2B 2C 2D 2E 2F   !"#$%&'()*+,-./
0BB0  30 31 32 33 34 35 36 37  38 39 3A 3B 3C 3D 3E 3F  0123456789:;<=>?
0BC0  40 41 42 43 44 45 46 47  48 49 4A 4B 4C 4D 4E 4F  @ABCDEFGHIJKLMNO
0BD0  50 51 52 53 54 55 56 57  58 59 5A 5B 5C 5D 5E 5F  PQRSTUVWXYZ[\]^_
0BE0  60 61 62 63 64 65 66 67  68 69 6A 6B 6C 6D 6E 6F  'abcdefghijklmno
0BF0  70 71 72 73 74 75 76 77  78 79 7A 7B 7C 7D 7E 7F  pqrstuvwxyz{|}~.
0C00  00 01 02 03 04 05 06 07  08 09 0A 0B 0C 0D 0E 0F  ................
0C10  10 11 12 13 14 15 16 17  18 19 1A 1B 1C 1D 1E 1F  ................
0C20  20 21 22 23 24 25 26 27  28 29 2A 2B 2C 2D 2E 2F   !"#$%&'()*+,-./
0C30  30 31 32 33 34 35 36 37  38 39 3A 3B 3C 3D 3E 3F  0123456789:;<=>?
0C40  40 41 42 43 44 45 46 47  48 49 4A 4B 4C 4D 4E 4F  @ABCDEFGHIJKLMNO
0C50  50 51 52 53 54 55 56 57  58 59 5A 5B 5C 5D 5E 5F  PQRSTUVWXYZ[\]^_
0C60  60 61 62 63 64 65 66 67  68 69 6A 6B 6C 6D 6E 6F  'abcdefghijklmno
0C70  70 71 72 73 74 75 76 77  78 79 7A 7B 7C 7D 7E 7F  pqrstuvwxyz{|}~.
0C80  80 81 82 83 84 85 86 87  88 89 8A 8B 8C 8D 8E 8F  ................
0C90  90 91 92 93 94 95 96 97  98 99 9A 9B 9C 9D 9E 9F  ................
0CA0  A0 A1 A2 A3 A4 A5 A6 A7  A8 A9 AA AB AC AD AE AF  ................
0CB0  B0 B1 B2 B3 B4 B5 B6 B7  B8 B9 BA BB BC BD BE BF  ................
0CC0  C0 C1 C2 C3 C4 C5 C6 C7  C8 C9 CA CB CC CD CE CF  ................
0CD0  D0 D1 D2 D3 D4 D5 D6 D7  D8 D9 DA DB DC DD DE DF  ................
0CE0  E0 E1 E2 E3 E4 E5 E6 E7  E8 E9 EA EB EC ED EE EF  ................
0CF0  F0 F1 F2 F3 F4 F5 F6 F7  F8 F9 FA FB FC FD FE FF  ................
0D00  00 01 02 03 04 05 06 07  08 09 0A 0B 0C 0D 0E 0F  ................
0D10  10 11 12 13 14 15 16 17  18 19 1A 1B 1C 1D 1E 1F  ................
0D20  20 21 22 23 24 25 26 27  28 29 2A 2B 2C 2D 2E 2F   !"#$%&'()*+,-./
0D30  30 31 32 33 34 35 36 37  38 39 3A 3B 3C 3D 3E 3F  0123456789:;<=>?
0D40  40 41 42 43 44 45 46 47  48 49 4A 4B 4C 4D 4E 4F  @ABCDEFGHIJKLMNO
0D50  50 51 52 53 54 55 56 57  58 59 5A 5B 5C 5D 5E 5F  PQRSTUVWXYZ[\]^_
0D60  60 61 62 63 64 65 66 67  68 69 6A 6B 6C 6D 6E 6F  'abcdefghijklmno
0D70  70 71 72 73 74 75 76 77  78 79 7A 7B 7C 7D 7E 7F  pqrstuvwxyz{|}~.
0D80  80 81 82 83 84 85 86 87  88 89 8A 8B 8C 8D 8E 8F  ................
0D90  90 91 92 93 94 95 96 97  98 99 9A 9B 9C 9D 9E 9F  ................
0DA0  A0 A1 A2 A3 A4 A5 A6 A7  A8 A9 AA AB AC AD AE AF  ................
0DB0  B0 B1 B2 B3 B4 B5 B6 B7  B8 B9 BA BB BC BD BE BF  ................
0DC0  C0 C1 C2 C3 C4 C5 C6 C7  C8 C9 CA CB CC CD CE CF  ................
0DD0  D0 D1 D2 D3 D4 D5 D6 D7  D8 D9 DA DB DC DD DE DF  ................
0DE0  E0 E1 E2 E3 E4 E5 E6 E7  E8 E9 EA EB EC ED EE EF  ................
0DF0  F0 F1 F2 F3 F4 F5 F6 F7  F8 F9 FA FB FC FD FE FF  ................
0E00  00 01 02 03 04 05 06 07  08 09 0A 0B 0C 0D 0E 0F  ................
0E10  10 11 12 13 14 15 16 17  18 19 1A 1B 1C 1D 1E 1F  ................
0E20  20 21 22 23 24 25 26 27  28 29 2A 2B 2C 2D 2E 2F   !"#$%&'()*+,-./
0E30  30 31 32 33 34 35 36 37  38 39 3A 3B 3C 3D 3E 3F  0123456789:;<=>?
0E40  40 41 42 43 44 45 46 47  48 49 4A 4B 4C 4D 4E 4F  @ABCDEFGHIJKLMNO
0E50  50 51 52 53 54 55 56 57  58 59 5A 5B 5C 5D 5E 5F  PQRSTUVWXYZ[\]^_
0E60  60 61 62 63 64 65 66 67  68 69 6A 6B 6C 6D 6E 6F  'abcdefghijklmno
0E70  70 71 72 73 74 75 76 77  78 79 7A 7B 7C 7D 7E 7F  pqrstuvwxyz{|}~.
0E80  80 81 82 83 84 85 86 87  88 89 8A 8B 8C 8D 8E 8F  ................
0E90  90 91 92 93 94 95 96 97  98 99 9A 9B 9C 9D 9E 9F  ................
0EA0  A0 A1 A2 A3 A4 A5 A6 A7  A8 A9 AA AB AC AD AE AF  ................
0EB0  B0 B1 B2 B3 B4 B5 B6 B7  B8 B9 BA BB BC BD BE BF  ................
```

```
0EC0   C0 C1 C2 C3 C4 C5 C6 C7   C8 C9 CA CB CC CD CE CF   ................
0ED0   D0 D1 D2 D3 D4 D5 D6 D7   D8 D9 DA DB DC DD DE DF   ................
0EE0   E0 E1 E2 E3 E4 E5 E6 E7   E8 E9 EA EB EC ED EE EF   ................
0EF0   F0 F1 F2 F3 F4 F5 F6 F7   F8 F9 FA FB FC FD FE FF   ................
0F00   00 01 02 03 04 05 06 07   08 09 0A 0B 0C 0D 0E 0F   ................
0F10   10 11 12 13 14 15 16 17   18 19 1A 1B 1C 1D 1E 1F   ................
0F20   20 21 22 23 24 25 26 27   28 29 2A 2B 2C 2D 2E 2F    !"#$%&'()*+,-./
0F30   30 31 32 33 34 35 36 37   38 39 3A 3B 3C 3D 3E 3F   0123456789:;<=>?
0F40   40 41 42 43 44 45 46 47   48 49 4A 4B 4C 4D 4E 4F   @ABCDEFGHIJKLMNO
0F50   50 51 52 53 54 55 56 57   58 59 5A 5B 5C 5D 5E 5F   PQRSTUVWXYZ[\]^_
0F60   60 61 62 63 64 65 66 67   68 69 6A 6B 6C 6D 6E 6F   `abcdefghijklmno
0F70   70 71 72 73 74 75 76 77   78 79 7A 7B 7C 7D 7E 7F   pqrstuvwxyz{|}~.
0F80   80 81 82 83 84 85 86 87   88 89 8A 8B 8C 8D 8E 8F   ................
0F90   90 91 92 93 94 95 96 97   98 99 9A 9B 9C 9D 9E 9F   ................
0FA0   A0 A1 A2 A3 A4 A5 A6 A7   A8 A9 AA AB AC AD AE AF   ................
0FB0   B0 B1 B2 B3 B4 B5 B6 B7   B8 B9 BA BB BC BD BE BF   ................
0FC0   C0 C1 C2 C3 C4 C5 C6 C7   C8 C9 CA CB CC CD CE CF   ................
0FD0   D0 D1 D2 D3 D4 D5 D6 D7   D8 D9 DA DB DC DD DE DF   ................
0FE0   E0 E1 E2 E3 E4 E5 E6 E7   E8 E9 EA EB EC ED EE EF   ................
0FF0   F0 F1 F2 F3 F4 F5 F6 F7   F8 F9 FA FB FC FD FE FF   ................
At end of current range
*
```

Index

A

A (assemble) DEBUG command 3-9 to 3-12
Absolute disk read (INT 25H) 7-19 to 7-21
Absolute disk write (INT 26H) 7-22 to 7-24
Addressing 4-1 to 4-14
Aligned words 4-5
Allocate memory (Function 48H) 7-140 to 7-141
ASCII 3-15, 8-22 to 8-27
ASCIIZ strings 5-11 to 5-12
Assembler 1-4
  modules 2-7 to 2-8, 2-12
  in DEBUG 3-9 to 3-12
  DEBUG list 3-39 to 3-40
  using system calls 7-6
Attribute (see File attribute and Video control)
ARF (see Automatic response file)
Asynchronous communications (see communications)
Automatic response file 2-9, 2-19 to 2-20
Auxiliary input (Function 03H) 7-31
Auxiliary output (Function 04H) 7-32

B

BASIC 1-4, 7-6
BIOS
  Memory usage 5-4, 5-5
  Service routines 8-1 to 8-44
Block devices 9-8
Bootstrap loader (INT 19H) 8-4, 8-30
Break ON/OFF Command 8-33
Breakpoints 3-21 to 3-23
Buffered keyboard input (Function 0AH) 7-40 to 7-41
Buffers Command 8-34
Bus lines 9-6 to 9-7

C

C (compare) DEBUG command 3-13 to 3-14
Calendar (see Clock)
Calculated addressing 4-14
Change file attributes (Function 43H) 7-130 to 7-131
Change current directory (Function 3BH) 7-117
Character devices 9-8
Control-C check (Function 33H) 7-106 to 7-107
Check keyboard status (Function 0BH) 7-42
Class 2-7
CLOCK
  device 9-26, 9-123 to 9-127
  (also see Time and Date)
Close file
  (Function 10H) 7-50 to 7-51
  (Function 3EH) 7-122
Clusters 5-13 to 5-23, 7-109
COM files 3-5, 3-26, 3-30, 6-1 to 6-7, 6-10, 6-21
Command line
  MS-LINK 2-9, 2-18 to 2-19
  DEBUG 3-3, 3-7 to 3-8
  direct commands 6-18

COMMUNICATIONS
BIOS routines (INT 14H) 8-4, 8-19 to 8-21
device drivers 9-27 to 9-35, 9-128 to 9-146
Compilers 1-4
COMSPEC 6-18
CON device (see Console)
CONFIG.SYS
Break ON/OFF 8-33
Buffer 8-34
Device 8-34
Files 8-35
Shell 8-35
Console 2-12
Direct I/O 7-35 to 7-36
Direct input 7-37
(also see Keyboard and Video)
Control blocks (see File control block)
Control-C check (Function 33H) 7-106 to 7-107
Country-dependent information 7-110 to 7-114
Create file
(Function 16H) 7-64 to 7-65
(Function 3CH) 7-118 to 7-119
Create sub-directory (Function 39H) 7-115
Current disk (Function 19H) 7-68

D

D (display) DEBUG command 3-15 to 3-16
Date
read 7-93 to 7-94, 7-157 to 7-158
set 7-95 to 7-96, 7-157 to 7-158
DEBUG 3-1 to 3-44
Commands:
A (assemble) 3-9 to 3-12
C (compare) 3-13 to 3-14
D (display) 3-15 to 3-16
E (enter) 3-17 to 3-19
F (fill) 3-20
G (go) 3-21 to 3-23
H (hexarithmetic) 3-24
I (input) 3-25
L (load) 3-26
M (move) 3-28
N (name) 3-5, 3-29 to 3-31
O (output) 3-32
Q (quit) 3-33
R (register) 3-34 to 3-36
S (search) 3-37
T (trace) 3-38
U (unassemble) 3-39 to 3-40
W (write) 3-41 to 3-43
Delete directory entry (Function 41H) 7-127
Delete file (Function 13H) 7-56 to 7-58
Determine memory size (INT 12H) 8-4, 8-31
DEVICES
character 9-8
drivers 1-4, 9-4 to 9-161
installation 9-12
read 7-123 to 7-124
Device header 9-9 to 9-12
Direct addressing 4-13

Direct console I/O (Function 06H) 7-35 to 7-36
Direct console input (Function 07H) 7-37
Directory 5-14 to 5-17, 5-23
  Change 7-117
  Create sub-directory 7-115
  Delete entry 7-127
  Delete file 7-56 to 7-58
  Move entry 7-115 to 7-156
  Remove directory 7-116
  Return current 7-139
  Step through 7-153
Diskette
  allocation 5-13 to 5-23
  BIOS routines (INT 13H) 8-4, 8-17 to 8-18
Disk operations:
  (also see Files)
  Create sub-directory 7-115
  Current disk 7-68
  Select 7-46
  Current directory 7-139
  Delete directory entry 7-127
  Duplicate file handle 7-137
  Move directory entry 7-155 to 7-156
  Read absolute 7-19 to 7-21
  Remove directory 7-116
  Reset 7-45
  Space 7-109
  Step through directory 7-153
  Transfer address 7-69 to 7-70, 7-103
  Verify flag 7-101 to 7-102, 7-154
  Write absolute 7-22 to 7-24
Display character (Function 02H) 7-30
Display string (Function 09H) 7-39
DMA controller 9-36 to 9-44
DOS version number 7-104
DSALLOCATE link switch 2-14
Duplicate file handle (Function 45H) 7-137

E

E (enter) DEBUG command 3-17 to 3-19
Equipment check (INT 11H) 8-4, 8-31
Error messages
  MS-LINK 2-24 to 2-27
  DEBUG 3-22, 3-44
  EXE2BIN 6-8 to 6-9
EXE files (see Run files)
EXE2BIN 6-2, 6-5 to 6-9
EXEC (see Load and execute program)
Exit address (INT 23H) 7-15

F

F (fill) DEBUG command 3-20
FAT (see File Allocation Table)
Fatal error abort address (INT 24H) 7-16 to 7-18
FCB (see File control block)
FILES
  (also see Disk operations)
  allocation table 5-13, 5-18 to 5-21, 5-22, 5-23
  attribute 5-15, 7-53

change 7-130 to 7-131
close file 7-50 to 7-51, 7-122
control block 3-30, 5-6 to 5-12, 6-16 to 6-19
create file 7-64 to 7-65, 7-118 to 7-119
delete file 7-56 to 7-58
disk transfer address 7-69 to 7-70, 7-103
duplicate file handle 7-137
forced 7-138
find 7-151 to 7-152, 7-52 to 7-53, 7-54 to 7-55
header (Run files) 6-10 to 6-14
load and execute program file 7-145 to 7-148
names 7-89 to 7-92, 7-66 to 7-67
open file 7-47 to 7-49, 7-120 to 7-121
pointer 7-128 to 7-129
random operations
block read 7-83 to 7-85
block write 7-86 to 7-88
record read 7-71 to 7-72
record write 7-73 to 7-75
set relative record 7-79 to 7-81
read 7-123 to 7-124
sequential operations
read 7-59 to 7-61
write 7-62 to 7-63
size 7-76 to 7-78
usage 2-3 to 2-6
write 7-125 to 7-126
Files Command 8-35
Find file (Function 4EH) 7-151 to 7-152
Flags 4-8 to 4-9
Floppy diskette interface controller 9-45 to 9-65
Flush buffer, read keyboard (Function 0CH) 7-43 to 7-44
Force duplicate of handle (Function 46H) 7-138
Free allocated memory (Function 49H) 7-142
Function request (INT 21H) 7-13, 7-29 to 7-173
Functions (DOS) 7-2 to 7-3, 7-8 to 7-9, 7-13, 7-29 to 7-173, 7-160 to 7-173
FORTRAN 2-14, 7-6

G

G (go) DEBUG command 3-21 to 3-23
Get date
(Function 2AH) 7-93 to 7-94
(Function 57H) 7-157 to 7-158
Get disk free space
(Function 36H) 7-109
Get disk transfer address
(Function 2FH) 7-103
Get DOS version number
(Function 30H) 7-104
Get interrupt vector
(Function 35H) 7-108
Get time
(Function 2CH) 7-97 to 7-98
(Function 57H) 7-157 to 7-158

Global symbols (see Public symbols)
Group 2-8, 2-14

H

H (hexarithmetic) DEBUG command 3-24
Handles 5-12
Hard disk controller 9-66 to 9-93
HEX files 3-4, 3-27
HIGH Link switch 2-14

I

I (input) DEBUG command 3-25
I/O control for devices (Function 44H) 7-132 to 7-136
I/O ports 9-3
Immediate addressing 4-12
Indirect addressing 4-13
INPUT
  (also see files)
  auxiliary device 7-31
  control devices 7-132 to 7-136
  file or device 7-123 to 7-124
  keyboard
    buffered 7-40 to 7-41
    direct 7-38
    echo 7-29
    flush buffer 7-43 to 7-44
Installing a device driver 9-12
Interrupts
  In general 5-3, 5-4, 7-4, 7-7, 7-10 to 7-28, 7-159, 8-1 to 8-32, 9-4 to 9-5
  Get vector 7-108
  Set vector 7-82
Interrupt controller 9-105 to 9-115
Interval timer 9-116 to 9-122

K

Keep process (Function 31H) 7-105
KEYBOARD
  (also see Console)
  BIOS routines 8-4, 8-22 to 8-28
  Buffered input 7-40 to 7-41
  Control-C check 7-106 to 7-107
  Device driver 9-94 to 9-99
  Flush buffer read 7-43 to 7-44
  Status check 7-42

L

L (load) DEBUG command 3-5
LIB files (see Libraries)
Libraries 2-4, 2-10, 2-12, 2-17, 2-18, 2-19, 2-22
LINENUMBERS link switch 2-15, 2-21
Linking object modules (see MS-LINK)
List files (see MAP files)
Load module (see Run files)
Loading programs 6-18 to 6-22, 7-145 to 7-148
Load and execute program (Function 4BH) 7-145 to 7-148
Logic lines (see Bus lines)

M

M (move) DEBUG command 3-28
Macro assembler (see Assembler)
MAP files 2-4, 2-10, 2-11, 2-15, 2-16, 2-18, 2-19, 2-23
MAP link switch 2-15, 2-21, 2-23
Media check 2-19, 9-19
Memory
  addressing 4-1 to 4-14
  allocation 7-142 to 7-144
  maps 5-1 to 5-5
  size 8-4, 8-31
Messages (also see Error messages) 1-3, 3-5, 3-6, 3-35
Modifications to BIOS 8-32 to 8-44
Modify allocated memory blocks (Function 4AH) 7-143 to 7-144
Move directory entry (Function 56H) 7-155 to 7-156
Move file pointer (Function 42H) 7-128 to 7-129
MS-LINK 1-4, 2-1 to 2-27

N

N (name) DEBUG command 3-5, 3-29 to 3-31
NO link switch 2-17
Non-aligned words 4-5
Notation 1-3

O

O (output) DEBUG command 3-32
OBJ files (see Object modules)
Object modules 2-2, 2-4, 2-10, 2-11, 2-18, 2-19, 2-20
Open file
  (Function 0FH) 7-47 to 7-49
  (Function 3DH) 7-120 to 7-121
OUTPUT
  Auxiliary device 7-32
  Character 7-30, 7-33 to 7-34
  Control devices 7-132 to 7-136
  File or device 7-125 to 7-126
  Printer BIOS routines 8-4, 8-29
  Screen dump 8-4, 8-30
  String 7-39

P

Parallel printer interface 9-100 to 9-104
Parse file name (Function 29H) 7-89 to 7-92
Pascal 2-14, 2-17, 7-6
PATH command 6-18
Pathname string (see ASCIIZ strings)
PAUSE link switch 2-15 to 2-16, 2-20
Print character (Function 05H) 7-33 to 7-34
Print screen (INT 05H) 8-4, 8-30
Printer routines (INT 17H) 8-4, 8-29

Printer 2-12, 2-21, 8-4
(also see Output)
PRN device (see Printer)
Process control (also see
Memory allocation)
  Keep process 7-105
  Load and execute program
  7-145 to 7-148
  Return code 7-150
  Terminate 7-11 to 7-12,
  7-25 to 7-28, 7-149
Program files (also see Run
files and COM files)
  Load and execute 7-145 to
  7-148
  Structure 6-1 to 6-22
Program segment prefix 3-5,
6-15 to 6-17, 6-19, 6-21, 7-11
Program terminate (INT 20H)
7-11 to 7-12
PROMPT command 6-18
Prompts 1-3, 2-10, 2-12, 3-4, 3-12
Public symbols 2-15, 2-20, 2-23

Q

Q (quit) DEBUG command
3-33

R

R (register) DEBUG command
3-34 to 3-36
Random block read (Function
27H) 7-83 to 7-85
Random block write (Function
28H) 7-86 to 7-88
Random Read (Function 21H)
7-71 to 7-72
Random Write (Function 22H)
7-73 to 7-75
Read from file or device
(Function 3FH) 7-123 to 7-124
Read keyboard (Function 08H)
7-38
Read keyboard and echo
(Function 01H) 7-29
Registers 4-6 to 4-10, 7-9
Register addressing 4-12
Remove directory (Function
3AH) 7-116
Rename file (Function 17H)
7-66 to 7-67
Request header 9-13 to 9-15,
9-18 to 9-28
Reset disks (Function 0DH)
7-45
Reset verify flag (Function
2EH) 7-101 to 7-102
Retrieve return code of child
(Function 4DH) 7-150
Return country-dependent info.
(Function 38H) 7-110 to 7-114
Return current verify flag
(Function 54H) 7-154
Return current directory
(Function 47H) 7-139
Run files 2-4, 2-11, 2-14, 2-15,
2-18, 2-19, 2-21, 3-4, 3-5, 3-27, 6-1
to 6-3, 6-5 to 6-7, 6-13 to 6-14,
6-20

S

S (search) DEBUG command
3-37
Scrambler ROM 9-162

Search for first entry (Function 11H) 7-52 to 7-53
Search for next entry (Function 12H) 7-54 to 7-55
Sector 5-21, 5-22, 5-23, 7-109, 8-17
Segment 2-7, 2-14, 2-21, 3-7, 3-8, 3-10, 3-21, 3-28, 3-37, 4-3 to 4-4, 4-7, 4-11, 6-6, 6-7, 6-16, 6-18, 6-21
Select disk (Function 0EH) 7-46
Sequential read (Function 14H) 7-59 to 7-61
Sequential write (Function 15H) 7-62 to 7-63
SET command 6-18
Set date
  (Function 2BH) 7-95 to 7-96
  (Function 57H) 7-157 to 7-158
Set time
  (Function 2DH) 7-99 to 7-100
  (Function 57H) 7-157 to 7-158
Set disk transfer address (Function 1AH) 7-69 to 7-70
Set relative record (Function 24H) 7-79 to 7-81
Set interrupt vector (Function 25H) 7-82
Set verify flag (Function 2EH) 7-101 to 7-102
Shell command 8-35
Speaker 9-147 to 9-149
STACK link switch 2-16
Status byte 5-5
Step through directory files (Function 4FH) 7-153
Switches 2-13 to 2-17, 2-19
Syntax (general) 1-3
System calls 7-1 to 7-173

T

T (trace) DEBUG command 3-38
Terminate address (INT 22H) 7-14
Terminate but stay resident (INT 27H) 7-25 to 7-26
Terminate process (Function 4CH) 7-149
Terminate program (Function 00H) 7-27 to 7-28
Time
  get 7-97 to 7-98, 7-157 to 7-158
  set 7-99 to 7-100, 7-157 to 7-158
Token (see Handle)

U

U (unassemble) DEBUG command 3-39 to 3-40
Utility programs 1-4

V

Video control (also see Console and Output)
  BIOS routine (INT 10H) 8-4, 8-5 to 8-16
Video controller 9-150 to 9-161
VM.TMP file 2-5, 2-6, 2-16

W

W (write) DEBUG command 3-41 to 3-43
Write to file or device (Function 41H) 7-125 to 7-126

The Display Enhancement Board

Table of Contents

1 **DEB Capabilities**

Introduction 1-2
The DEB Driver 1-4
16-Color Graphics 1-5
Look-Up Table (LUT) 1-7
Overlay Modes 1-8

2 **Programming Tips**

Presence of Hardware/Software 2-2
Hardware/Software Compatibility 2-3
Setup 2-4

3 **How to Program the DEB**

Overview 3-2
Mode Setting 3-3
Setting Colors and Effects 3-5
Displaying Graphics Images 3-6

4 Interrupt 10H Functions

Introduction 4-2
Functions 4-4

5 Programming the LUT

Overview 5-2
16-Color Graphics LUT Programming 5-3
Overlay Modes LUT Programming 5-23
Programming the Bit Planes 5-34

1 DEB Capabilities

- Introduction
- The DEB Driver
- 16-Color Graphics
- Look-Up Table (LUT)
- Overlay Modes

Introduction

The Display Enhancement Board (DEB) option adds improved color and graphics functionality to your AT&T PC 6300. When you use the DEB with the PC 6300 color monitor, you can display graphics in up to 16 color combinations simultaneously or treat the screen as two screens in one and overlay one screen treatment on top of the other. When you use the DEB with the PC 6300 monochrome monitor, you have the same capabilities you have with the color monitor, except that colors are displayed as "shades of green."

The DEB is compatible with existing software, so all the programs you have already can be used now as if the DEB were not installed. Of course, these programs may not have access to any of the new capabilities.

This supplement describes the functionality of the DEB device driver. Although it is not necessary to use the driver in order to use the DEB, the driver is designed to work with MS-DOS, GWBASIC, and other AT&T software products. If you wish to program the DEB hardware directly, you must consult the *AT&T Technical Reference Manual.* Such programming is considered a circumvention of the AT&T operating system and we advise against it.

This supplement assumes that you are familiar with video programming through the Interrupt 10H interface and with INTEL® 8086 assembler programming. Information on the Interrupt 10H interface can be found in the *System Programmer's Guide,* in the section on the ROM BIOS Service Routines.

Before you begin writing programs for the DEB, follow the procedures in the DEB Installation Manual for installing the DEB hardware and device driver software.

The DEB is an optional hardware component for the AT&T PC 6300 that works in conjunction with the PC 6300's built-in Video Display Controller (VDC) to provide improved color and graphics functionality.

The built-in VDC contains circuitry and memory that support either 4 color medium resolution (320 × 200 pixels) graphics, 1 color high resolution (640 × 200 pixels) graphics, or 1 color super resolution (640 × 400 pixels) graphics.

The DEB contains additional circuitry and memory that can be combined with the capabilities of the built-in VDC to produce up to 16 color combinations in either high or super resolution. You can also program the VDC and DEB separately, treating them as two separate images that are combined on one screen to produce an overlaying effect. The overlay modes let you use up to 8 colors on the DEB screen and up to 16 colors on the VDC screen.

The DEB Driver

You load the DEB device driver by entering a "DEVICE" statement in the CONFIG.SYS file (see **Chapter 2, Programming Tips**). The driver installs an Interrupt 10H "filter" during the loading process.

When you are using the DEB and are running some programs that use the DEB and some that do not, the "filter" provides video support for both kinds of programs. For programs that do not use the DEB, the filter passes control to the standard Interrupt 10H ROM BIOS routine.

The DEB driver installs a filter for Interrupt 9H. This filter resets the DEB to transparent mode whenever you warmstart the system through **CTRL/ALT/DEL**. The filter controls scrolling when you press **CTRL/NUMLOCK**.

16-Color Graphics

This feature lets you display 16 color combinations in either high resolution (640 × 200) or super resolution (640 × 400). Not only can you use the standard 16 colors, you can also combine colors to form new colors and cause pixels to blink from one color to another.

The DEB provides 5 palettes for you to use when programming in color. At any point in your program, you select one of the palettes as the "active" palette. The color combinations contained in that palette determine what colors and effects show on the screen.

Each of the first 4 palettes contains a default set of 16 color combinations, but to suit the needs of your program you can change the contents of the palette to any one of the following:

- any of the 16 standard colors with which you are already familiar from the standard applications. The standard colors are:

0 = black	8 = gray
1 = blue	9 = light blue
2 = green	10 = light green
3 = cyan	11 = light cyan
4 = red	12 = light red
5 = magenta	13 = light magenta
6 = brown	14 = yellow
7 = white	15 = high intensity white

- a mixture, or "dithering," of any 2 of the 16 standard colors

- an alternation, or blinking, between any 2 of the standard 16 colors

The last palette contains no default combinations. You program the fifth palette by loading color values into a 256-byte array. The DEB device driver uses this special palette to program the DEB's color Look-Up Table (LUT). By using the LUT you can add the capability of dithering or blinking between any four colors.

Look-Up Table (LUT)

The LUT resides in RAM on the DEB board, and is accessed through write-only hardware registers. The device driver keeps a copy of the register values in the LUT. The register values are accessible to software applications through the device driver. The LUT contains 256 values that determine the colors, blinking, and dithering that appear on the screen. Whether you need to learn about the use and layout of the LUT depends on the application you are writing.

If you use the standard palettes, you need not be concerned with the LUT. The DEB device driver automatically programs the LUT to correspond to the way you set up the palettes. If you program a custom LUT, you greatly increase the color combinations and blinking effects available to you.

Overlay Modes

The overlay modes let you use the screen to display two images at once, independently. For example, you can display a high resolution color graphics image with its own foreground and background. Then, on "top" of that image, you can display a box of text and scroll the text without affecting the graphics image.

The overlay modes use the DEB to control one image and use the standard controller board to control the other image. You can select from many combinations of graphics, text, color, and high or super resolution in designing the two images.

The overlay modes offer 5 palettes. Each of the first 4 palettes has 8 positions. These four palettes have default colors that you can change to suit your needs. You can choose 8 color combinations from any of the 16 standard colors, or blink between 2 of the standard colors. The dithering combinations of the 16-color graphics modes are not available. You can also use the last palette to custom program the LUT.

2 Programming Tips

- **Presence of Hardware/Software**
- **Hardware/Software Compatibility**
- **Setup**

Presence of Hardware/Software

Whenever you plan an application, it is important to use the DEB device driver to test for the presence of both the DEB and the associated driver. Test for the presence of the hardware by checking for DEB video memory. This is accomplished by writing and reading back data patterns into memory, in the range A000H:0H to B800H:0H. Test for the software device driver by issuing a function call to open the device called "DEBDRIVE," then immediately issuing a call to close "DEBDRIVE." If the open fails (carry set on return from Interrupt 21H) then the driver is not present. No functions are implemented in the driver, which is used only to detect the presence of the software.

Hardware/Software Compatibility

The driver software has been designed to fit into the structure of MS-DOS programs. The DEB hardware uses the same range of addresses as the standard video ports on any compatible machine. If your application uses a light pen, consult the DEB supplement in the *AT&T Personal Computer Technical Reference Guide.*

The DEB driver makes minor modifications to the ROM BIOS video interrupt. Mode setting and color selection offer additional functionality. Be careful when you use the following functions.

- SET MODE — uses an additional register BL
- SCROLLING — uses an additional register BH
- STATUS — returns an additional register pair ES:DI. No application should count on ES:DI not changing.

Setup

Install the DEB driver just as you would install any device driver. Be sure the CONFIG.SYS file is in the root directory. Put the line DEVICE = DEDRIVER.DEV in CONFIG.SYS. This line puts the DEB driver in the device driver chain. The driver makes patches in INT 10H and INT 9H to add the new functionality. The driver has two features:

- the INIT function, which deallocates itself after it runs
- chaining, which allows you to test for the driver's presence by issuing an open function call

3 How to Program the DEB

- Overview
- Mode Setting
- Setting Colors and Effects
- Displaying Graphics Images

Overview

There are three steps for video programming that apply whether or not you are using the DEB capability:

1 Set the hardware's mode. You also must set the active page if you are in an overlay mode and want to select the DEB screen.

2 Select the color combinations and effects you want to use.

3 Construct the graphics images you want to display.

This chapter describes each of these steps in detail. This chapter does **not** describe how to program the LUT directly (see **Chapter 5, "Programming the LUT"**).

Mode Setting

The DEB is controlled by invoking one of the DEB video modes through the Set Mode function (INT 10H, function 0H). The Set Mode function establishes the mode for both the DEB and the VDC. These modes fall into four categories: 16-color graphics, overlay, transparent, and disabled.

16-Color Graphics Modes

There are two DEB modes that provide 16-color graphics: high resolution and super resolution. Both these modes let you use 5 palettes and display up to 16 color combinations simultaneously.

Overlay Modes

When overlaying the VDC on the DEB output, you specify one of the modes for the VDC and one mode for the DEB. The VDC modes are a subset of the modes for non-DEB graphics: 80 × 25 text mode, high and super resolution modes. The DEB modes are both graphics modes: high and super resolution.

If you are using one of the four standard palettes, the VDC's output takes precedence over the output of the DEB, so that if each board writes a pixel to the same screen location, the pixel sent by the VDC is displayed. This precedence is programmed into the LUT. If you want to have the DEB take precedence over the VDC, you must change the values in the LUT. (For more information, see **Chapter 5, "Programming the LUT."**)

Transparent Mode

The non-DEB modes, modes 0-40H and mode 48H, work exactly as they work without the DEB device driver installed.

Disabled Mode

In the disabled mode, you can cause the output of the VDC, the DEB, or both to be blacked out. This allows you to draw a graphics image or to fill a screen with text and not have them displayed while you are building them. You can then have the image "pop up" by taking VDC or DEB out of the disabled mode. You can also achieve this result by using the programmable palettes and the LUT.

Setting Colors and Effects

Colors and effects are controlled by the Set Color Palette command, (INT 10H function 0BH). Use this function to set color values in one of the four palettes, to switch between palettes, or to reset palettes to their default values. You also use Set Color Palette to program the LUT directly.

Displaying Graphics Images

There are two methods for displaying graphics images using the DEB: writing dots at screen locations or directly programming the VDC and DEB memory.

To write dots (pixels) to the screen, use the Write Dot function (INT 10H, function 0CH). Write Dot requires that you specify the display page, the row and column where you want the dot to appear, and the color or pattern for the dot.

If you want to program the VDC and DEB graphics memory directly , you need to learn the details of how the LUT is structured and how LUT addresses are formed (see the section on "Programming the Bit Planes" in Chapter 5).

4 Interrupt 10H Functions

- Introduction
- Functions

Introduction

The following section describes the DEB device driver software functions. This interface is an extension of the INT 10H software function to the PC6300 ROM BIOS that controls the VDC. The ROM BIOS screen driver has 16 functions:

- 0H set the display mode
- 2H set the cursor position
- 3H read the cursor position
- 5H select the active display page
- 6H scroll the active page up
- 7H scroll the active page down
- 8H read character/attribute at the current cursor position
- 9H write character/attribute at cursor position
- AH write only the character at current cursor position
- BH change the color palette
- CH write a point on the screen

DH read a point on the screen

EH write in teletype style to the active page

FH return information about the active video state

Not all these functions are applicable to the DEB. The filter receives the Interrupt 10H function call, filters the functions that are applicable to the DEB and performs them. The functions that are not applicable to the DEB are passed on to the ROM BIOS INT 10H routine or to a previously installed filter or driver routine. The following section describes the functions which are processed by the DEB Interrupt 10H filter.

Functions

Set Mode

The function establishes the mode for both the DEB and the VDC. If you select a non-DEB related mode, control is passed to the ROM resident Set Mode function. Set Mode initializes palette 0 as the active palette.

Input

(AH) = 0H function number for Set Mode
(AL) = new mode
(BL) = optional overlay mode

Setting AL bit 7 = 0 puts you in either the DEB transparent mode or 16-color graphics mode:

(AL) = 0H 40 × 25 monochrome, text
(AL) = 1H 40 × 25 color, text
(AL) = 2H 80 × 25 monochrome, text
(AL) = 3H 80 × 25 color, text
(AL) = 4H 320 × 200 color
(AL) = 5H 320 × 200 monochrome
(AL) = 6H 640 × 200 color
(AL) = 40H 640 × 400 with 2-position programmable palette, defaulting to black and white
(AL) = 41H 640 × 200 16-color graphics with four palettes
(AL) = 42H 640 × 400 16-color graphics with four palettes
(AL) = 44H Disable mode (disables both DEB and VDC output)

Setting AL bit 7 = 1 puts you in overlay mode. The following values are only used in overlay mode. AL contains the setting for the VDC; BL contains the mode setting for the DEB. In overlay modes, the active page defaults to zero.

VDC Settings

(AL) = 82H	80 × 25 monochrome, text
(AL) = 83H	80 × 25 color, text
(AL) = 86H	640 × 200 color graphics
(AL) = 0C0H	640 × 400 color graphics
(AL) = 0C4H	Disable mode. Disables only the VDC.

DEB settings

(BL) = 6H	640 × 200 graphics with four 8-position palettes.
(BL) = 40H	640 × 400 graphics with four 8-position palettes.
(BL) = 44H	Disable mode. Disables only the DEB.

Output

Contents of all registers are preserved.

Example

```
MOV AH,0        ; Select Set Mode
MOV AL,41H      ; Select 16 color graphics
INT 10H         ; Change the mode
```

Set Cursor Position

This function sets the cursor position for either the DEB, the VDC, or both.

Input

(AH) = 2H Function number for Set Cursor Position
(DH,DL) = row, column of new position
(BH) = page number
Valid page numbers for DEB modes are 0 for the VDC and 80H for the DEB in overlay mode. Row values are 0 thru 23, column values are 0 thru 79, in DEB modes.

Output

Contents of all registers are preserved.

Example

```
MOV AH,2   ; SCP function
MOV DH,ROW
MOV DL,COL
MOV BH,PAGE
INT 10H    ; Moves cursor to position defined in
             above variables.
```

Read Cursor Position

This function returns the position of the cursor for the DEB, VDC, or both.

Input

(AH) = 3H Function number for Read Cursor Position

(BH) = page number
Valid page numbers for DEB modes are 0 for VDC and 80H for the DEB in overlay mode. Row values are 0 thru 23, column values are 0 thru 79, in DEB modes.

Output

(DH,DL) = row, column of current position.

Contents of all other registers are preserved.

Example

```
MOV AH,3
MOV BH,PAGE
INT 10H
MOV ROW,DH
MOV COL,DL
```

Select Active Display

This command allows selection of the DEB display in overlay mode.

Input

(AH) = 5H Function number for Select Action Display
(AL) = active page
Values for the active page in DEB modes are, 0 for the VDC and 80H for the DEB.

Output

Contents of all registers are preserved.

Example

```
MOV AH,5
MOV AL,PAGE
INT 10H
```

Input

(AH) = 0BH Function number for Set Color Palette
(AL) = palette function selector
(BH) = positional pointer
(BL) = color value

For simple palette programming functions, use the following

(AL) = 0
(BH) = palette color ID

BH = FFH	switches to the palette specified in BL, without changing to the default palettes unless there is a change in palette type (e.g., change from a 16-position palette to an 8-position palette).
BH = 80H	switches to the palette specified in BL and resets the palette to its default.
BH = 0-16	sets this palette position to the color or attribute in BL.

(BL) = actual color value or code for blinking and dithering

The special settings for using a customized LUT in Set Color Palette are as follows:

Input

(AL) = non-zero (a zero here selects a standard palette)

AL bit 0 = 1 means use ES:SI to program the palette and registers BH and BL to indicate an offset and length into the LUT. ES:SI points to the LUT table (in the above example, LUT-STRING).

In this case, BH = offset into LUT and BL = length of portion of LUT to be changed. If you are loading an entire new table, set BH and BL to 0.

AL bit 1 = 1 means use BH and BL to program the LUT one location at a time.

In this case, BH = position in LUT and BL = the value to put in that position.

AL bit 2 = 1 means use the short LUT addressing mode. (Only uses the first 16 LUT entries).

The DEB driver lets you automatically load your customized LUT and use it in place of one of the standard palettes.

The steps for loading and using the customized LUT are:

1 Define the table with DB (Define Byte) statements.

2 Load the table in by using the Set Color Palette command.

3 Use the Read Dot and Write Dot commands to access the LUT (see **Chapter 4, "Interrupt 10H Functions"**).

The code for defining the table would be similar to this:

```
LUT-STRING   DB  4          ! Signifies active palette 4
             DB  1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,
             DB  1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,
             DB  1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,
             DB  1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,
             .
             .
             .
             DB  1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,
```

To load a new table of values into the LUT, where the table in your program is named LUT-STRING, you can use these statements:

```
PUSH DS          ! save the data segment address
POP ES
MOV  S1,LUT-STRING
MOV  AL,1
MOV  AH,11
XOR  BH,BX       ! Sets BH=BL=0
INT  10
```

The defaults for each of the four palettes are:

Default Palettes

Palette Number 0

Position	Color
0	0 = black
1	2 = green
2	4 = red
3	6 = brown
4	1 = blue
5	3 = cyan
6	5 = magenta
7	7 = white
8	8 = gray
9	9 = light blue
10	10 = light green
11	11 = light cyan
12	12 = light red
13	13 = light magenta
14	14 = yellow
15	15 = high-intensity white

Palette Number 1

Position	Color
0	0 = black
1	3 = cyan
2	5 = magenta
3	7 = white
4	1 = blue
5	2 = green
6	4 = red
7	6 = brown
8	8 = gray
9	9 = light blue
10	10 = light green
11	11 = light cyan
12	12 = light red
13	13 = light magenta
14	14 = yellow
15	15 = high-intensity white

Palettes 2 and 3 are the same, and they contain the standard colors in numerical order.

Palette Number 2 and Palette Number 3

Position	Color
0	0 = black
1	1 = blue
2	2 = green
3	3 = cyan
4	4 = red
5	5 = magenta
6	6 = brown
7	7 = white
8	8 = gray
9	9 = light blue
10	10 = light green
11	11 = light cyan
12	12 = light red
13	13 = light magenta
14	14 = yellow
15	15 = high-intensity white

DITHER COMBINATIONS FOR DEB PALETTES 0-3

Color combinations 136-255 have been pre-assigned to allow you easy access to dithering effects while using the standard palettes. The following table describes the available combinations.

A ↓ / B →	black	blue	green	cyan	red	magenta	brown	white	gray	light blue	light green	light cyan	light red	light magenta	yellow
black															
blue	136														
green	137	138													
cyan	139	140	141												
red	142	143	144	145											
magenta	146	147	148	149	150										
brown	151	152	153	154	155	156									
white	157	158	159	160	161	162	163								
gray	164	165	166	167	168	169	170	171							
light blue	172	173	174	175	176	177	178	179	180						
light green	181	182	183	184	185	186	187	188	189	190					
light cyan	191	192	193	194	195	196	197	198	199	200	201				
light red	202	203	204	205	206	207	208	209	210	211	212	213			
light magenta	214	215	216	217	218	219	220	221	222	223	224	225	226		
yellow	227	228	229	230	231	232	233	234	235	236	237	238	239	240	
high-intensity white	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255

NOTE: To select a value that combines colors A and B to create a new color, find the number at the intersection of row A and column B.

BLINKING COLOR EFFECTS FOR DEB PALETTES 0-3

Color combinations 16-135 have been pre-assigned to allow you easy access to blinking effects while using the standard palettes. The following table describes the available combinations.

A ↓ / B →	blue	green	cyan	red	magenta	brown	white	gray	light blue	light green	light cyan	light red	light magenta	yellow	high-intensity white
black	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
blue		31	32	33	34	35	36	37	38	39	40	41	42	43	44
green			45	46	47	48	49	50	51	52	53	54	55	56	57
cyan				58	59	60	61	62	63	64	65	66	67	68	69
red					70	71	72	73	74	75	76	77	78	79	80
magenta						81	82	83	84	85	86	87	88	89	90
brown							91	92	93	94	95	96	97	98	99
white								100	101	102	103	104	105	106	107
gray									108	109	110	111	112	113	114
light blue										115	116	117	118	119	120
light green											121	122	123	124	125
light cyan												126	127	128	129
light red													130	131	132
light magenta														133	134
yellow															135

NOTE: To select a value that will cause blinking between colors A and B, find the number at the intersection of row A and column B.

Write Dot

The Write function writes a pixel to the location on the screen that you specify. If the screen is in the DEB mode, Write Dot may also write a pattern.

Input

(AH) = 0CH, Function number for Write Dot
(AL) = Palette position to be written.
(BH) = display page designator (bit 7 = 1 selects the DEB)
(CX) = column number
(DX) = row number

Output

Contents of all registers are preserved.

Example

```
MOV AH,0CH
MOV AL,PALPOS
MOV BH,PAGE
MOV CX,COL
MOV DX,ROW
INT  ;Write the Dot
```

Read Dot

This function reads a dot from the screen. If the screen is in the DEB mode this function returns the value in the LUT that corresponds to this dot. (For more information, see **Chapter 5, "Programming the LUT."**)

Input

(AH) = 0DH Function number for Read Dot
(BH) = display page designator (bit 7 = 1 selects the DEB)
(CX) = column number
(DX) = row number

Output

(AH) = VDC value or DEB palette position
Contents of all other registers are preserved.

Example

```
MOV AH,0DH
MOV BH,PAGE
MOV CX,COL
MOV DX,ROW
INT
MOV DOTCOL,AH   ;Save the Dot
```

Input

(AH) = 0EH, Function number for Write Teletype
(AL) = character to write
(BL) = foreground color (in graphics modes) If bit 7 = 1, color is XOR'd to current contents.

Output

Contents of all registers are preserved.

Example

```
MOV AH,0EH
MOV AL,CHAR
MOV BL,FGCOL
INT 10H
```

Read Current Video State

This function returns the current video state. It indicates whether the DEB or VDC is active in the overlay mode and returns the number of the active palette.

Input

(AH) = 0FH Function number for Read Current Video State

Output

(BH) = display page designator
(AL) = mode currently set
(ES:DI) = pointer to a copy of the current LUT

Example

```
MOV AH,0FH
INT 10H
```

5 Programming the LUT

- Overview
- 16-Color Graphics LUT Programming
- Overlay Modes LUT Programming
- Programming the Bit Planes

Overview

This chapter describes programming the DEB Look-Up Table (LUT). By programming the LUT yourself, you can create color patterns that are not available when you use standard palettes. You need not read this chapter if you do not want to use this extended functionality.

The hardware uses the LUT to translate the contents of video memory patterns into graphics effects. In the standard palettes, INT 10H filter programs the LUT for you and thereby provides the preassigned color combinations and effects as described in previous chapters.

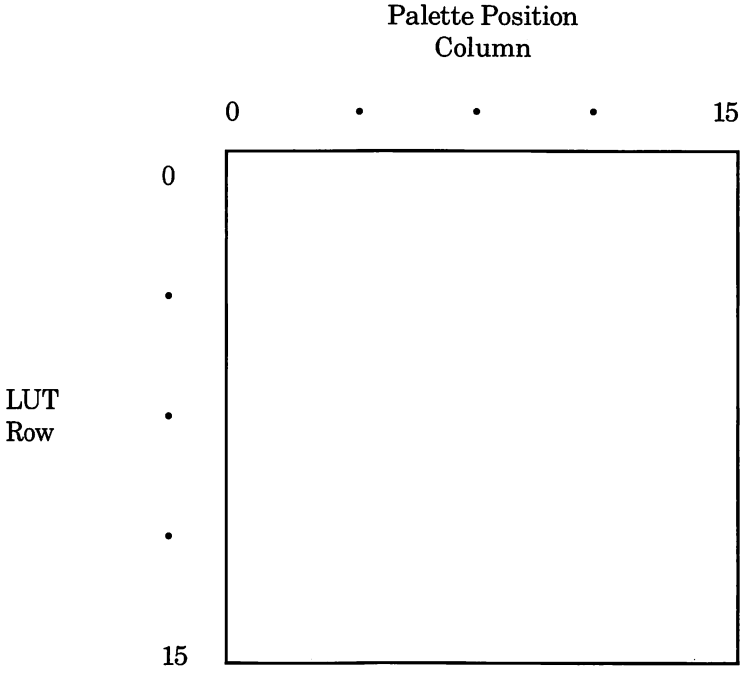
To program the LUT directly, you select Palette 4 in Set Color Palette function. Palette 4, also called the "LUT palette," has a minimum of 256 positions. Each palette position contains a value between 0 and 15. These values map into the LUT locations on the DEB. The 256 locations on the DEB collectively determine the color and special effects displayed when you specify a particular palette position. The color and special effect for each pixel on the screen are determined by:

- the palette position you specify
- the values in the LUT
- the active mode

There are some differences in the way the LUT is structured for 16-color graphics modes and overlay modes. This chapter describes LUT operation for 16-color graphics modes and overlay modes separately.

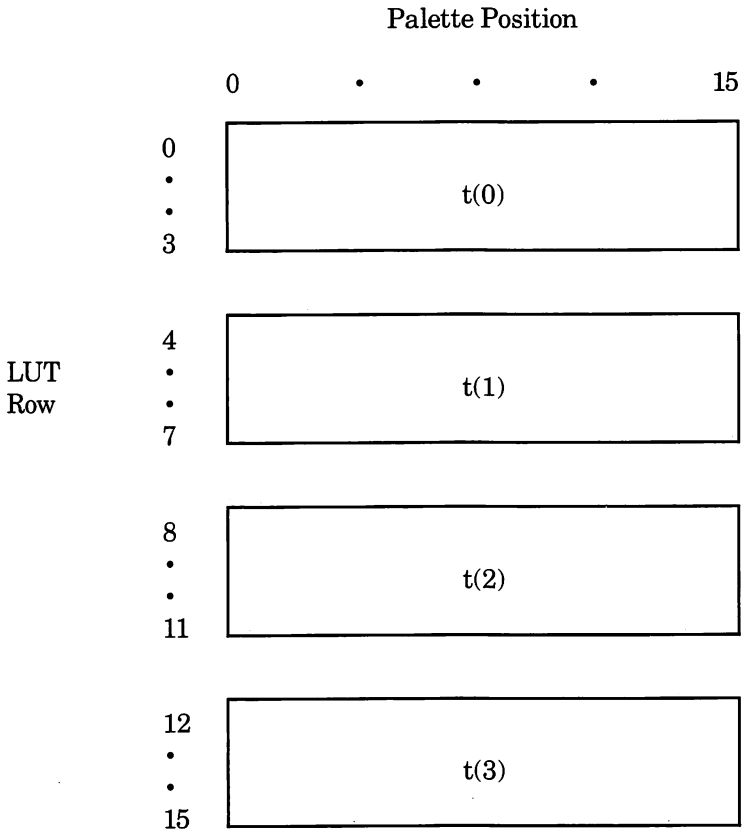
16-Color Graphics Lut Programming

In these modes, the LUT can be viewed as a two-dimensional array (16 × 16). Each location contains one of the standard 16 colors.



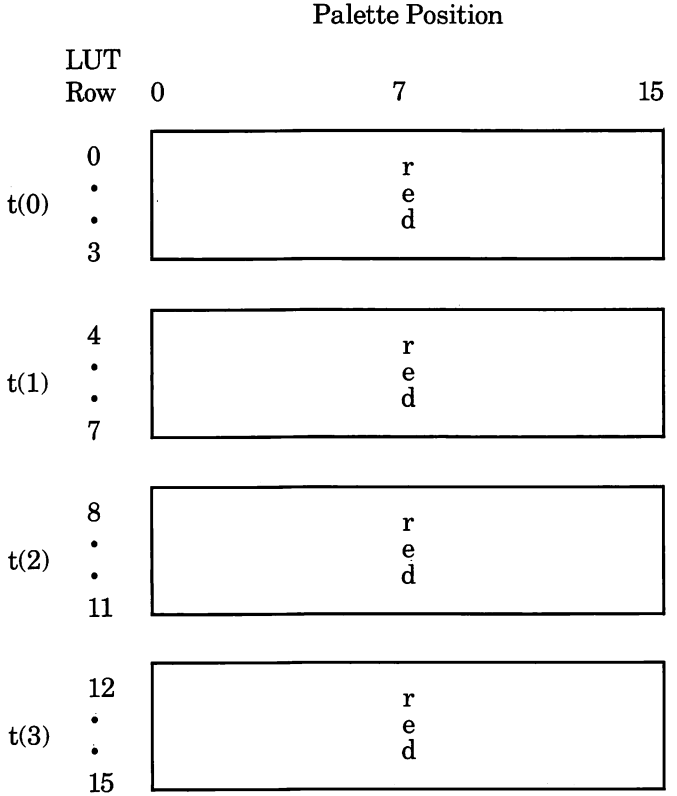
The locations in the LUT are numbered consecutively from left to right and top to bottom. Thus, location 17 corresponds to Row 1, palette position 1.

In the 16-color graphics mode, the LUT is divided into four "time states." At any one time, only one quarter of the LUT determines the display on the screen.

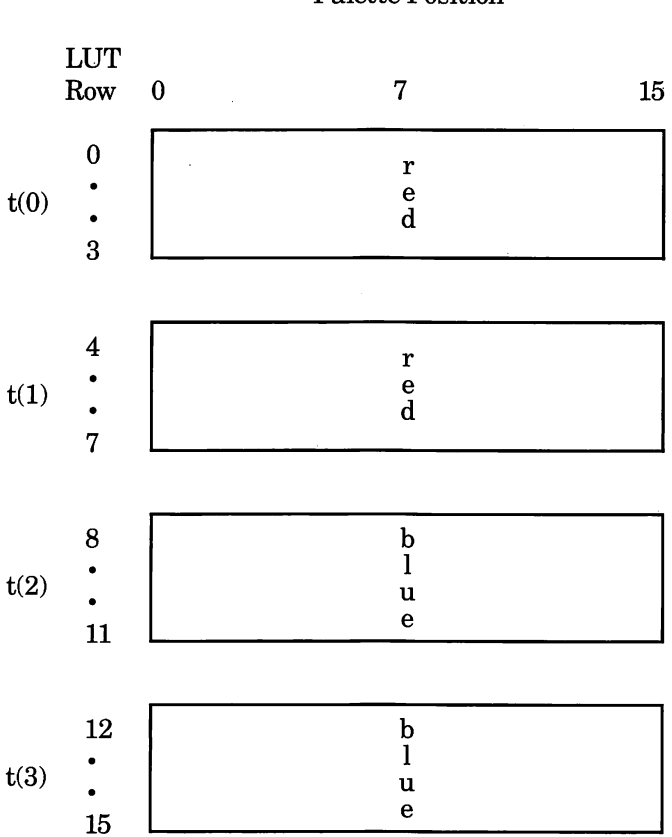


The hardware cycles through the LUT every second, so each quarter of the LUT is active for ¼ of each second. The cycling mechanism produces blinking. The following examples show the details of how you can produce several different blinking effects by setting different values in the LUT.

In this example, the Write Dot or Write Character functions specify palette position 7 and the LUT is set up as shown. Pixels are displayed as a solid red color. In the first ¼ second, the DEB displays the color in the first quarter of the LUT, which in this case is red. In the second, third, and fourth ¼ seconds, the DEB displays the color in the second, third, and fourth quarters of the LUT, respectively. In this example, the DEB keeps finding the color value for red, so what you see on the screen is a solid (non-blinking) red color.

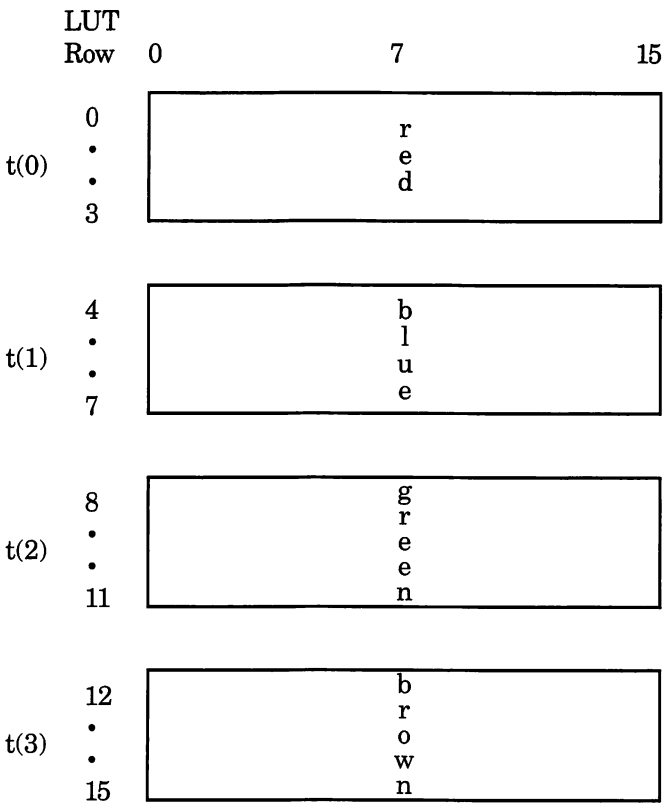


In this example, any item displayed on the screen with palette position 7 blinks between red and blue. For the first two ¼ seconds, the DEB picks up the color value for red from the first and second quarters of the LUT. For the second two ¼ seconds, the DEB obtains the color value of blue from the LUT. The net effect is a slow blink between red and blue.



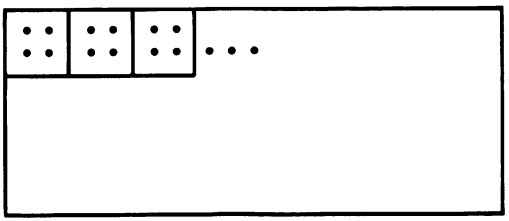
Slow Blink

In this example, any item displayed using palette position 7 blinks rapidly between red, blue, green, and brown.



4-Color Fast Blink

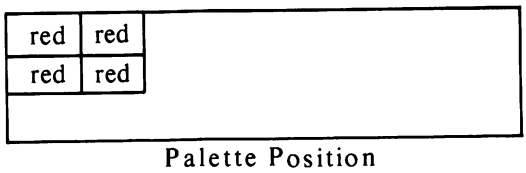
For dithering colors, the DEB uses a scheme similar to the blinking scheme. Dithering is accomplished by manipulating groups of 4 adjacent pixels. The screen is divided into blocks of 4 pixels.

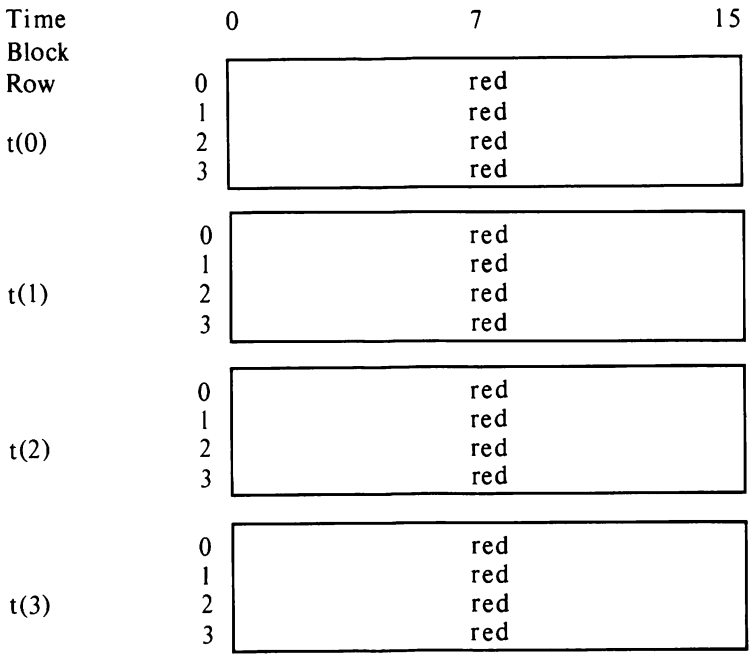


Each of the 4 time states is divided into four dither states that determine the dithering effect. The rows of the time state blocks correspond to the 4-pixel blocks on the screen in the following way:

LUT Row	Time Block Row	Palette Position 0 . . . 15
0	0	t(0)
1	1	
2	2	
3	3	
4	0	t(1)
5	1	
6	2	
7	3	
8	0	t(2)
9	1	
10	2	
11	3	
12	0	t(3)
13	1	
14	2	
15	3	

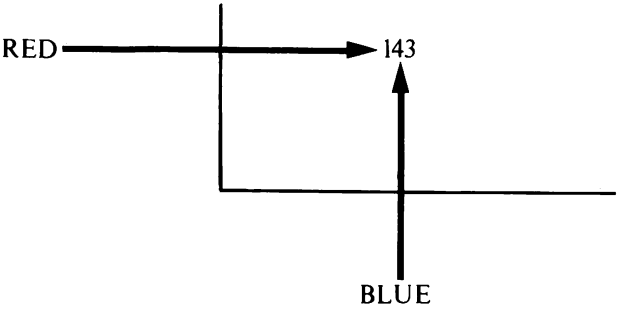
The pixels in the pixel blocks are so close together that our eyes cannot perceive them as separate. If each of the pixels in a pixel block is a different color, our eyes perceive the pixel block as one color — a combination of the color of the individual pixels. If the adjacent pixels are the same color, our eyes see just that one color.



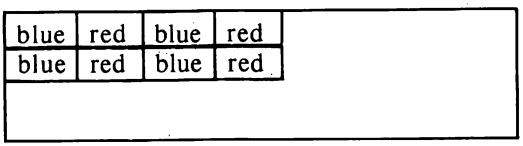


"Solid" Dither showing correspondence between pixel positions in a pixel block and time state rows

Remember the table of "pre-assigned" dithered colors. To combine colors, you check the table for the color number for a particular dither effect. For example, you would choose this number to produce a dither between red and blue.



If you want to program the LUT directly to dither red and blue together, the LUT would look like this:



	Time Block Row	Palette Position 0	7	15
t(0)	0	blue		
	1	red		
	2	blue		
	3	red		
t(1)	0	blue		
	1	red		
	2	blue		
	3	red		
t(2)	0	blue		
	1	red		
	2	blue		
	3	red		
t(3)	0	blue		
	1	red		
	2	blue		
	3	red		

2-Color Dither

You can set up the LUT to dither two, three, or four colors together.

Palette Position

	Time Block Row	0	7	15
t(0)	0	red		
	1	blue		
	2	green		
	3	brown		
t(1)	0	red		
	1	blue		
	2	green		
	3	brown		
t(2)	0	red		
	1	blue		
	2	green		
	3	brown		
t(3)	0	red		
	1	blue		
	2	green		
	3	brown		

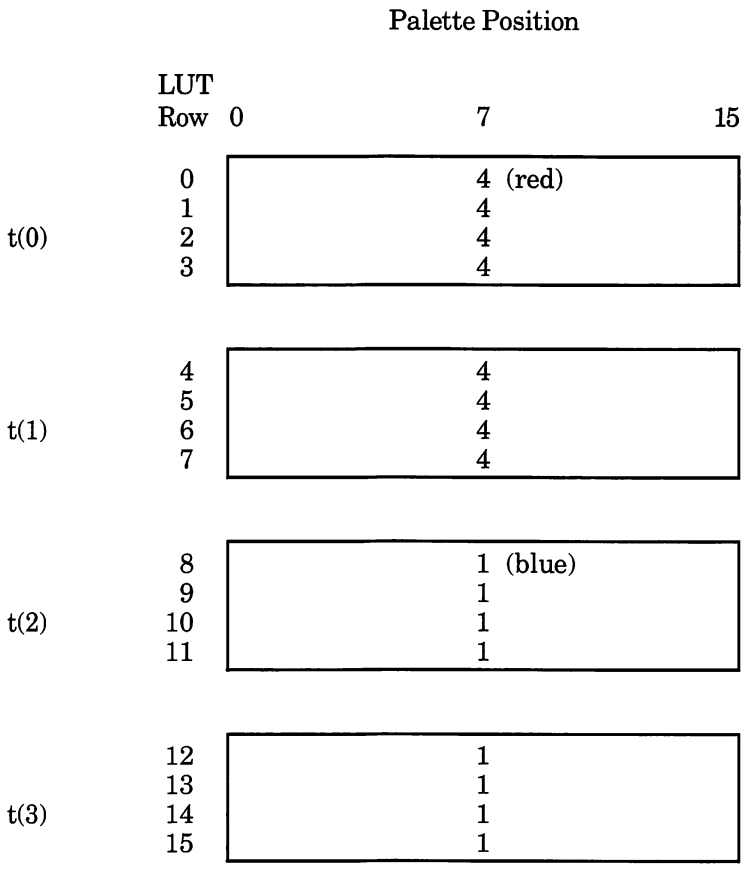
4-Color Dither

The following examples show the actual LUT values for each of the previous cases of blinking and dithering.

Palette Position

	LUT Row	0	7	15
	0		4 (red)	
	1		4	
t(0)	2		4	
	3		4	
	4		4	
	5		4	
t(1)	6		4	
	7		4	
	8		4	
	9		4	
t(2)	10		4	
	11		4	
	12		4	
	13		4	
t(3)	14		4	
	15		4	

Palette Position 7 programmed for Non-Blinking Red

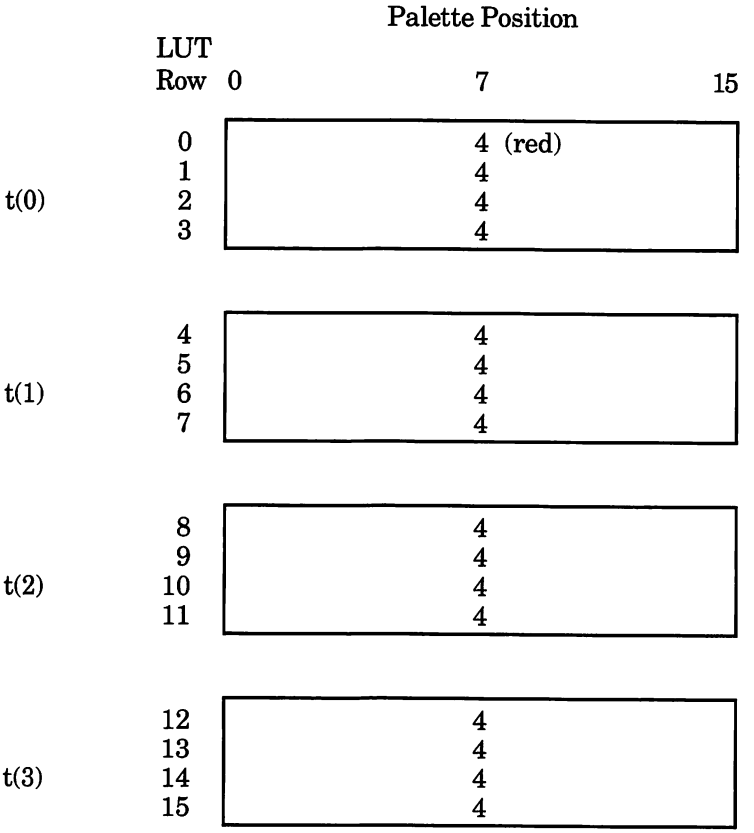


Palette Position 7 programmed to blink slowly between red and blue.

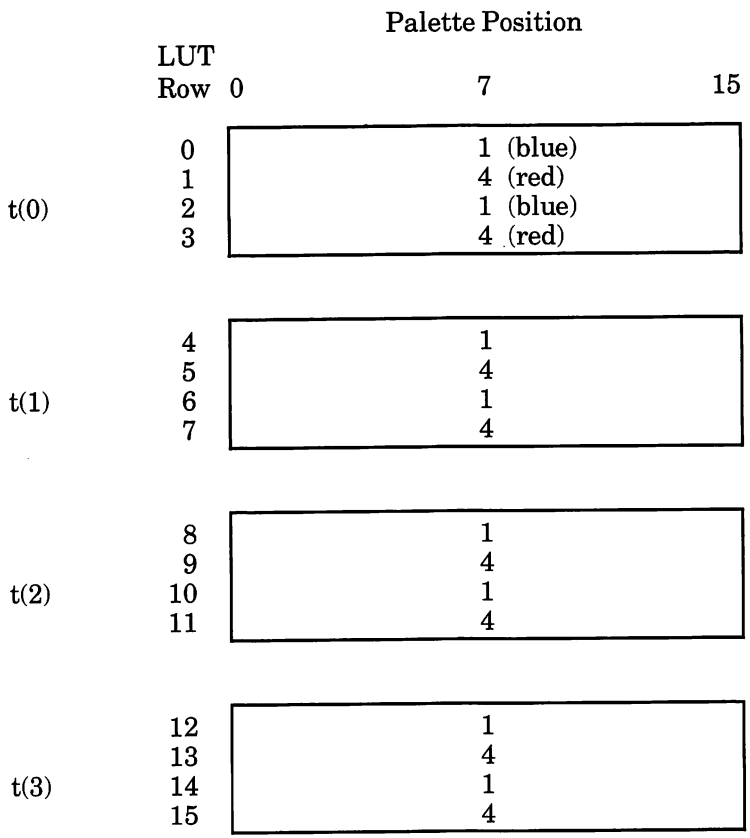
Palette Position

	LUT Row	0	7	15
	0		4 (red)	
	1		4	
t(0)	2		4	
	3		4	
	4		1 (blue)	
	5		1	
t(1)	6		1	
	7		1	
	8		2 (green)	
	9		2	
t(2)	10		2	
	11		2	
	12		6 (brown)	
	13		6	
t(3)	14		6	
	15		6	

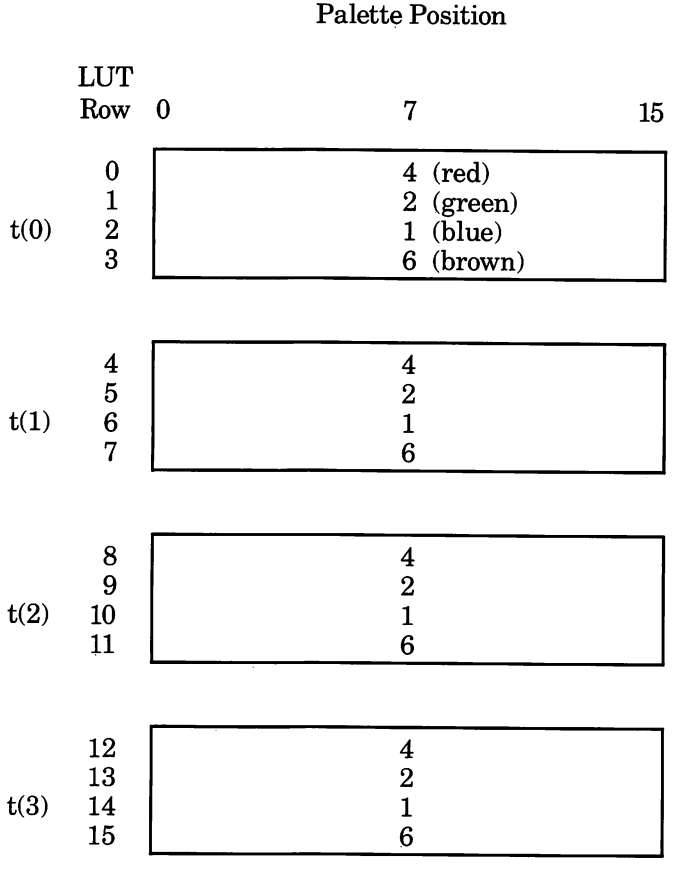
4-Color Fast Blink



Solid Red Dither



2-Color Dither: Red and Blue



4-Color Dither Between Red, Green, Blue, and Brown

The following is an example that combines blinking and dithering:

	LUT Row	Palette Position 0	7	15
	0		1 (blue)	
	1		4 (red)	
t(0)	2		1	
	3		4	
	4		1	
	5		4	
t(1)	6		1	
	7		4	
	8		2 (green)	
	9		6 (brown)	
t(2)	10		2	
	11		6	
	12		2	
	13		6	
t(3)	14		2	
	15		6	

The following table of values can be used to program the LUT for normal 16-color graphics.

Palette Position

	LUT Row	0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
	0	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,
	1	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,
t(o)	2	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,
	3	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,
	4	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,
	5	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,
t(1)	6	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,
	7	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,
	8	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,
	9	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,
t(2)	10	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,
	11	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,
	12	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,
	13	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,
t(3)	14	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,
	15	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,

Non-Blinking Standard Colors

Note that palette position 7 in the first two time states has been programmed to show white and in the second two time states to show red.

Palette Position

	LUT Row	0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
	0	0, 1, 2, 3, 4, 5, 6, **7,** 8, 9, 10, 11, 12, 13, 14, 15,
	1	0, 1, 2, 3, 4, 5, 6, **7,** 8, 9, 10, 11, 12, 13, 14, 15,
t(o)	2	0, 1, 2, 3, 4, 5, 6, **7,** 8, 9, 10, 11, 12, 13, 14, 15,
	3	0, 1, 2, 3, 4, 5, 6, **7,** 8, 9, 10, 11, 12, 13, 14, 15,
	4	0, 1, 2, 3, 4, 5, 6, **7,** 8, 9, 10, 11, 12, 13, 14, 15,
	5	0, 1, 2, 3, 4, 5, 6, **7,** 8, 9, 10, 11, 12, 13, 14, 15,
t(1)	6	0, 1, 2, 3, 4, 5, 6, **7,** 8, 9, 10, 11, 12, 13, 14, 15,
	7	0, 1, 2, 3, 4, 5, 6, **7,** 8, 9, 10, 11, 12, 13, 14, 15,
	8	0, 1, 2, 3, 4, 5, 6, **4,** 8, 9, 10, 11, 12, 13, 14, 15,
	9	0, 1, 2, 3, 4, 5, 6, **4,** 8, 9, 10, 11, 12, 13, 14, 15,
t(2)	10	0, 1, 2, 3, 4, 5, 6, **4,** 8, 9, 10, 11, 12, 13, 14, 15,
	11	0, 1, 2, 3, 4, 5, 6, **4,** 8, 9, 10, 11, 12, 13, 14, 15,
	12	0, 1, 2, 3, 4, 5, 6, **4,** 8, 9, 10, 11, 12, 13, 14, 15,
	13	0, 1, 2, 3, 4, 5, 6, **4,** 8, 9, 10, 11, 12, 13, 14, 15,
t(3)	14	0, 1, 2, 3, 4, 5, 6, **4,** 8, 9, 10, 11, 12, 13, 14, 15,
	15	0, 1, 2, 3, 4, 5, 6, **4,** 8, 9, 10, 11, 12, 13, 14, 15,

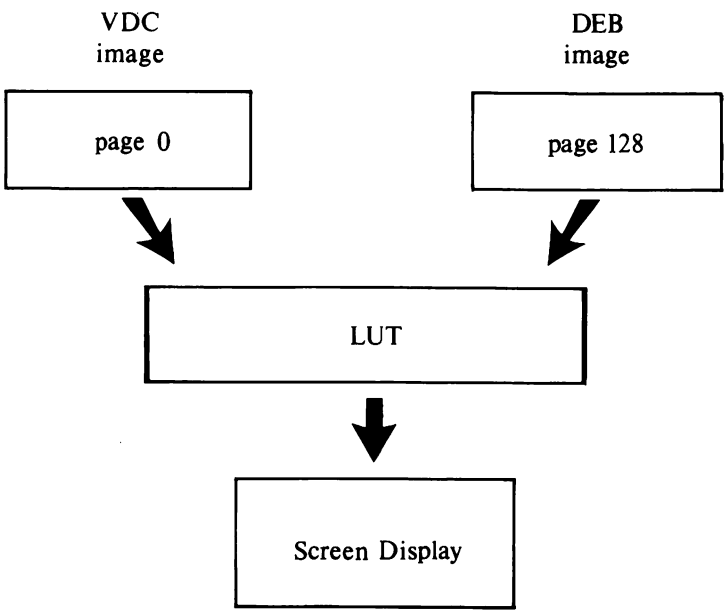
LUT for Blinking Between White and Red in Palette Position 7

Overlay Modes LUT Programming

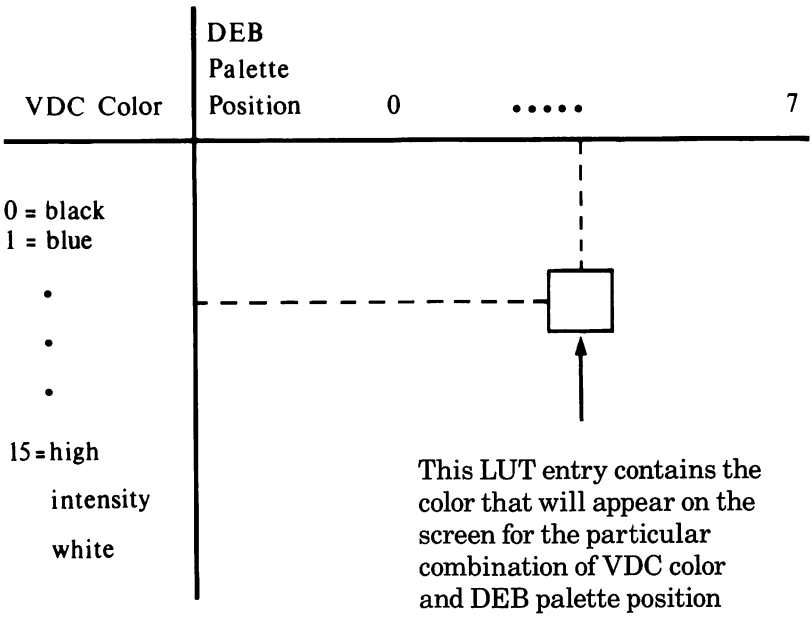
Overlay Modes LUT Programming

When the LUT is used in the overlay modes it can be viewed as a two-dimensional array with 8 columns and 32 rows. The column values are DEB palette positions. The row values are VDC color values.

In overlay modes, there are 2 separately controlled images: the VDC image and the DEB image. The 2 images are combined on the display screen. Each pixel on the screen has 2 values associated with it: the VDC color and the DEB palette position. The LUT is used to resolve contention between the 2 values associated with each pixel.



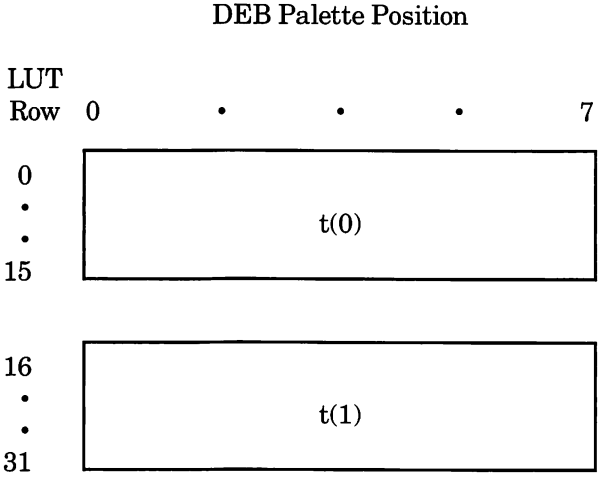
The LUT for overlay modes looks like this:



As in the 16-color graphics modes, the locations in the LUT are numbered consecutively from left to right and top to bottom. For example, location 17 corresponds to Row 2, Palette Position 0.

In the overlay modes, as in the 16-color graphics mode, the LUT is divided into time states that control blinking effects. However, in the overlay modes, the LUT is only divided into two time states. Half of the LUT determines what is being displayed at any time. The top half is used for the first ½ of each second and the bottom half is used for the second ½ of each second.

Using the overlay modes, you create blinking by making the values in the top half of the table different from the corresponding values in the bottom half of the table.



The following example shows the LUT values for standard Palette 2 of an overlay mode. The LUT is programmed so that the DEB image is displayed only if the VDC color is 0 (black). If the VDC requests any other color, then that color is displayed no matter what the DEB requests. This has the effect of overlaying the VDC image "on top" of the DEB image.

t(0)

VDC Color Values	DEB Palette Position 0	1	2	3	4	5	6	7
0	**0,**	**1,**	**2,**	**3,**	**4,**	**5,**	**6,**	**7,**
1	1,	1,	1,	1,	1,	1,	1,	1,
2	2,	2,	2,	2,	2,	2,	2,	2,
3	3,	3,	3,	3,	3,	3,	3,	3,
4	4,	4,	4,	4,	4,	4,	4,	4,
5	5,	5,	5,	5,	5,	5,	5,	5,
6	6,	6,	6,	6,	6,	6,	6,	6,
7	7,	7,	7,	7,	7,	7,	7,	7,
8	8,	8,	8,	8,	8,	8,	8,	8,
9	9,	9,	9,	9,	9,	9,	9,	9,
10	10,	10,	10,	10,	10,	10,	10,	10,
11	11,	11,	11,	11,	11,	11,	11,	11,
12	12,	12,	12,	12,	12,	12,	12,	12,
13	13,	13,	13,	13,	13,	13,	13,	13,
14	14,	14,	14,	14,	14,	14,	14,	14,
15	15,	15,	15,	15,	15,	15,	15,	15,

DEB Palette Position

	VDC Color Values	0	1	2	3	4	5	6	7
	0	**0,**	**1,**	**2,**	**3,**	**4,**	**5,**	**6,**	**7,**
	1	1,	1,	1,	1,	1,	1,	1,	1,
	2	2,	2,	2,	2,	2,	2,	2,	2,
	3	3,	3,	3,	3,	3,	3,	3,	3,
t(1)	4	4,	4,	4,	4,	4,	4,	4,	4,
	5	5,	5,	5,	5,	5,	5,	5,	5,
	6	6,	6,	6,	6,	6,	6,	6,	6,
	7	7,	7,	7,	7,	7,	7,	7,	7,
	8	8,	8,	8,	8,	8,	8,	8,	8,
	9	9,	9,	9,	9,	9,	9,	9,	9,
	10	10,	10,	10,	10,	10,	10,	10,	10,
	11	11,	11,	11,	11,	11,	11,	11,	11,
	12	12,	12,	12,	12,	12,	12,	12,	12,
	13	13,	13,	13,	13,	13,	13,	13,	13,
	14	14,	14,	14,	14,	14,	14,	14,	14,
	15	15,	15,	15,	15,	15,	15,	15,	15,

In this example, the standard Palette 2 is modified so that position 2 is a blinking between blue (color 1) and red (color 4).

	VDC Color Values	DEB Palette Position 0	1	2	3	4	5	6	7
	0	0,	1,	**1,**	3,	4,	5,	6,	7,
	1	1,	1,	1,	1,	1,	1,	1,	1,
	2	2,	2,	2,	2,	2,	2,	2,	2,
	3	3,	3,	3,	3,	3,	3,	3,	3,
t(0)	4	4,	4,	4,	4,	4,	4,	4,	4,
	5	5,	5,	5,	5,	5,	5,	5,	5,
	6	6,	6,	6,	6,	6,	6,	6,	6,
	7	7,	7,	7,	7,	7,	7,	7,	7,
	8	8,	8,	8,	8,	8,	8,	8,	8,
	9	9,	9,	9,	9,	9,	9,	9,	9,
	10	10,	10,	10,	10,	10,	10,	10,	10,
	11	11,	11,	11,	11,	11,	11,	11,	11,
	12	12,	12,	12,	12,	12,	12,	12,	12,
	13	13,	13,	13,	13,	13,	13,	13,	13,
	14	14,	14,	14,	14,	14,	14,	14,	14,
	15	15,	15,	15,	15,	15,	15,	15,	15,

DEB Palette Position

	VDC Color Values	0	1	2	3	4	5	6	7
t(1)	0	0,	1,	**4,**	3,	4,	5,	6,	7,
	1	1,	1,	1,	1,	1,	1,	1,	1,
	2	2,	2,	2,	2,	2,	2,	2,	2,
	3	3,	3,	3,	3,	3,	3,	3,	3,
	4	4,	4,	4,	4,	4,	4,	4,	4,
	5	5,	5,	5,	5,	5,	5,	5,	5,
	6	6,	6,	6,	6,	6,	6,	6,	6,
	7	7,	7,	7,	7,	7,	7,	7,	7,
	8	8,	8,	8,	8,	8,	8,	8,	8,
	9	9,	9,	9,	9,	9,	9,	9,	9,
	10	10,	10,	10,	10,	10,	10,	10,	10,
	11	11,	11,	11,	11,	11,	11,	11,	11,
	12	12,	12,	12,	12,	12,	12,	12,	12,
	13	13,	13,	13,	13,	13,	13,	13,	13,
	14	14,	14,	14,	14,	14,	14,	14,	14,
	15	15,	15,	15,	15,	15,	15,	15,	15,

In this example, values in the LUT cause the DEB's output to take precedence over the VDC's output. The VDC's output is only displayed when you specify DEB palette position 0 in a graphics statement.

	VDC Color Values	DEB Palette Positions 0 1 2 3 4 5 6 7
	0	**0,** 1, 2, 3, 4, 5, 6, 7,
	1	**1,** 1, 2, 3, 4, 5, 6, 7,
	2	**2,** 1, 2, 3, 4, 5, 6, 7,
	3	**3,** 1, 2, 3, 4, 5, 6, 7,
	4	**4,** 1, 2, 3, 4, 5, 6, 7,
	5	**5,** 1, 2, 3, 4, 5, 6, 7,
	6	**6,** 1, 2, 3, 4, 5, 6, 7,
t(0)	7	**7,** 1, 2, 3, 4, 5, 6, 7,
	8	**8,** 1, 2, 3, 4, 5, 6, 7,
	9	**9,** 1, 2, 3, 4, 5, 6, 7,
	10	**10,** 1, 2, 3, 4, 5, 6, 7,
	11	**11,** 1, 2, 3, 4, 5, 6, 7,
	12	**12,** 1, 2, 3, 4, 5, 6, 7,
	13	**13,** 1, 2, 3, 4, 5, 6, 7,
	14	**14,** 1, 2, 3, 4, 5, 6, 7,
	15	**15,** 1, 2, 3, 4, 5, 6, 7,

	VDC Color Values	DEB Palette Positions 0 1 2 3 4 5 6 7
	1	**0,** 1, 2, 3, 4, 5, 6, 7,
	1	**0,** 1, 2, 3, 4, 5, 6, 7,
	2	**2,** 1, 2, 3, 4, 5, 6, 7,
	3	**3,** 1, 2, 3, 4, 5, 6, 7,
	4	**4,** 1, 2, 3, 4, 5, 6, 7,
t(1)	5	**5,** 1, 2, 3, 4, 5, 6, 7,
	6	**6,** 1, 2, 3, 4, 5, 6, 7,
	7	**7,** 1, 2, 3, 4, 5, 6, 7,
	8	**8,** 1, 2, 3, 4, 5, 6, 7,
	9	**9,** 1, 2, 3, 4, 5, 6, 7,
	10	**10,** 1, 2, 3, 4, 5, 6, 7,
	11	**11,** 1, 2, 3, 4, 5, 6, 7,
	12	**12,** 1, 2, 3, 4, 5, 6, 7,
	13	**13,** 1, 2, 3, 4, 5, 6, 7,
	14	**14,** 1, 2, 3, 4, 5, 6, 7,
	15	**15,** 1, 2, 3, 4, 5, 6, 7,

The following LUT entirely blocks out VDC output:

	VDC Color Values	DEB Palette Positions 0 1 2 3 4 5 6 7
t(0)	0	0, 1, 2, 3, 4, 5, 6, 7,
	1	0, 1, 2, 3, 4, 5, 6, 7,
	2	0, 1, 2, 3, 4, 5, 6, 7,
	3	0, 1, 2, 3, 4, 5, 6, 7,
	4	0, 1, 2, 3, 4, 5, 6, 7,
	5	0, 1, 2, 3, 4, 5, 6, 7,
	6	0, 1, 2, 3, 4, 5, 6, 7,
	7	0, 1, 2, 3, 4, 5, 6, 7,
	8	0, 1, 2, 3, 4, 5, 6, 7,
	9	0, 1, 2, 3, 4, 5, 6, 7,
	10	0, 1, 2, 3, 4, 5, 6, 7,
	11	0, 1, 2, 3, 4, 5, 6, 7,
	12	0, 1, 2, 3, 4, 5, 6, 7,
	13	0, 1, 2, 3, 4, 5, 6, 7,
	14	0, 1, 2, 3, 4, 5, 6, 7,
	15	0, 1, 2, 3, 4, 5, 6, 7,

DEB Palette Positions

	VDC Color Values	0 1 2 3 4 5 6 7
t(1)	0	0, 1, 2, 3, 4, 5, 6, 7,
	1	0, 1, 2, 3, 4, 5, 6, 7,
	2	0, 1, 2, 3, 4, 5, 6, 7,
	3	0, 1, 2, 3, 4, 5, 6, 7,
	4	0, 1, 2, 3, 4, 5, 6, 7,
	5	0, 1, 2, 3, 4, 5, 6, 7,
	6	0, 1, 2, 3, 4, 5, 6, 7,
	7	0, 1, 2, 3, 4, 5, 6, 7,
	8	0, 1, 2, 3, 4, 5, 6, 7,
	9	0, 1, 2, 3, 4, 5, 6, 7,
	10	0, 1, 2, 3, 4, 5, 6, 7,
	11	0, 1, 2, 3, 4, 5, 6, 7,
	12	0, 1, 2, 3, 4, 5, 6, 7,
	13	0, 1, 2, 3, 4, 5, 6, 7,
	14	0, 1, 2, 3, 4, 5, 6, 7,
	15	0, 1, 2, 3, 4, 5, 6, 7,

Programming the Bit Planes

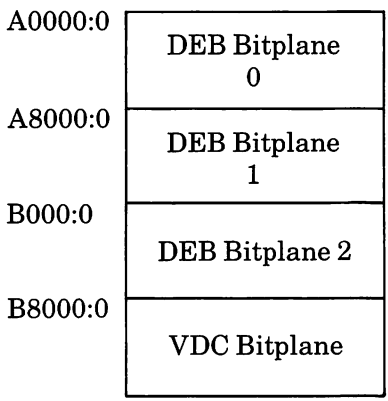
Introduction

Once you have learned to program the LUT directly using the Set Color Palette command, you can make further use of the LUT's capabilities by programming the VDC and DEB video memory directly.

By directly programming the video memory of the VDC and DEB boards, you can increase the graphics display speed. The values you load into the video memory planes determine how the LUT is accessed. This section assumes that you have read and understood how to program the LUT directly.

In the 16-color graphics modes, the device driver combines the 3 bit planes of the DEB with one bit plane from the VDC to create the four bit planes necessary for 16-color graphics.

In the overlay modes, the device driver uses the 3 DEB bit planes for 8-color graphics output and uses the VDC board separately for either text or graphics output.



Memory Map

LUT Addressing

A LUT address is an 8-bit value that points to one of the 256 locations within the LUT. The method of address formation depends on the current video mode.

For transparent and disabled modes, LUT addressing is irrelevant. In the transparent mode, VDC color values bypass the LUT processing and go directly to the monitor output. In the disabled mode, all output from the LUT is forced to the value of zero.

For the 16-color graphics and overlay modes, the LUT address is composed of bits from the DEB video bit planes, the VDC's video output, and DEB timing bits.

Timing Bits

The timing bits are called BLINK1, BLINK2, PAT1, and PAT2. BLINK1 and BLINK2 effect blinking; PAT1 and PAT2 effect patterning (dithering).

All of the timing bits are applicable in the 16-color graphics mode; only BLINK1 is part of the address formation in the overlay mode. Therefore, you have fewer options for blinking and no ability to dither in the overlay mode.

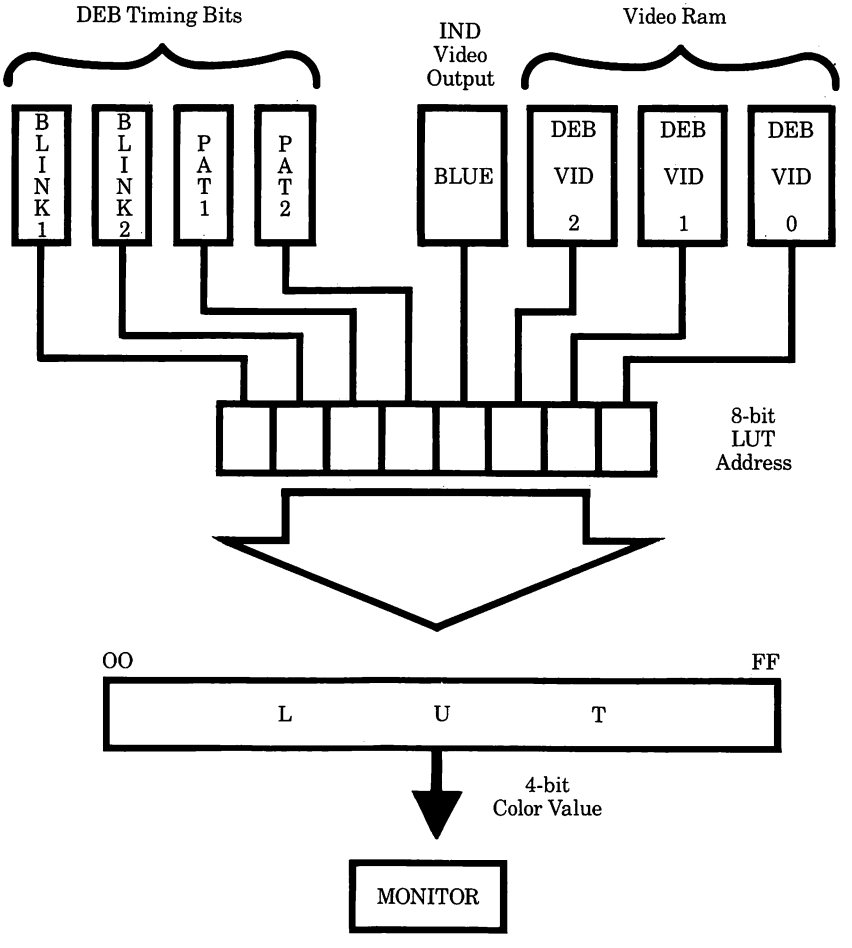
The operation of the timing bits is very fundamental to creation of special effects. The bits always cycle on and off, each at a different rate. BLINK1 cycles on and off each 1/4 second. BLINK2 cycles on and off each 1/8 second.

PAT1 and PAT2 cycle on and off so fast that the eye cannot perceive a blink (PAT1 is the fastest). A dithered color is really 2-4 separate colors that are changing so rapidly that the eye perceives them as one solid color.

PAT1 changes value at the same rate that the monitor's cathode ray moves from one pixel to the next. PAT1's effect on LUT addressing is that it switches the address by 16 LUT entries — in the previous table, between pairs of rows. PAT2 changes value at the same rate that the cathode ray changes scanlines — in the previous table, between one pair of rows and the next pair of rows.

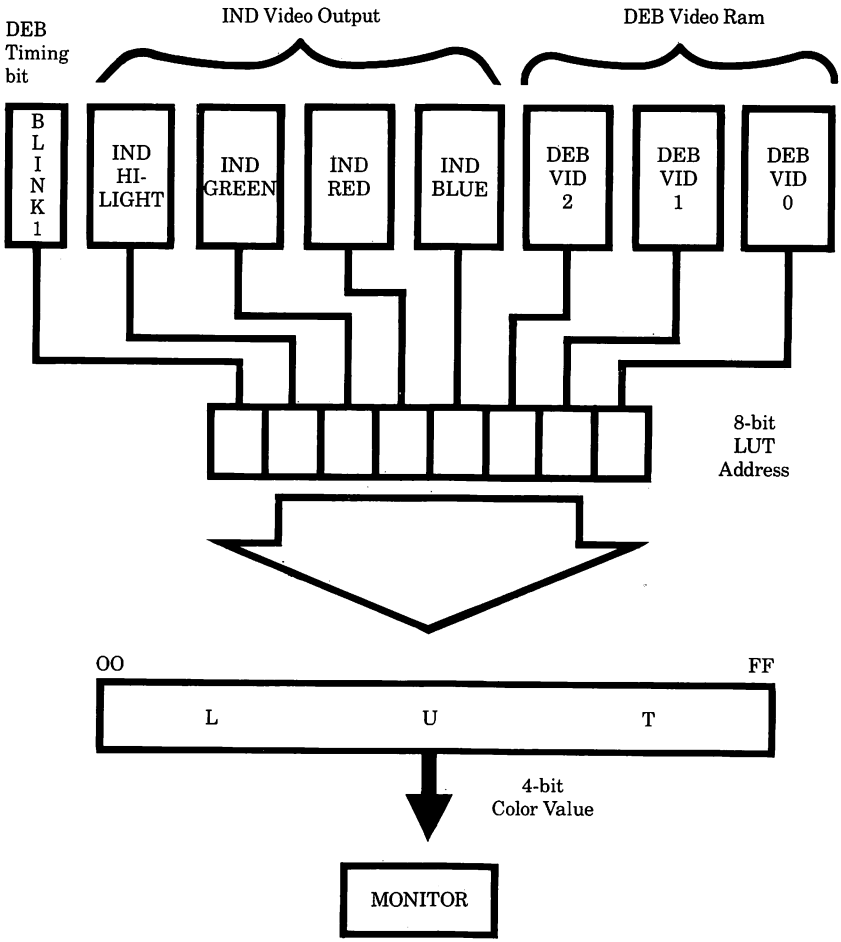
PAT2	PAT1	Portion of LUT
0	0	1st 16 entries of each quarter
0	1	2nd 16 entries of each quarter
1	0	3rd 16 entries of each quarter
1	1	4th 16 entries of each quarter

1. 16-color Graphics Mode



To output a color to the monitor, the DEB concatenates the DEB timing bits BLINK1, BLINK2, PAT1, PAT2, the BLUE output bit from the VDC, and a bit from corresponding locations on each of the three DEB bit planes.

2. Overlay Mode



To output a color to the monitor, the DEB concatenates the following bits: BLINK1, the HILIGHT, GREEN, RED, and BLUE output bits from the VDC, and a bit from corresponding locations on each of the three DEB bit planes.

Write Character Only at Current Cursor Position

In DEB modes, this function is the same as "Write Character and Attribute."

Set Color Palette

This function is used to set color values in one of the four palettes, to switch between palettes, or to reset palettes to their default values.

In the overlay modes, the Set Color Palette function works on the active page. If the active page is set to display to the VDC board, this function works the same as the standard ROM BIOS INT 10H (function 0BH).

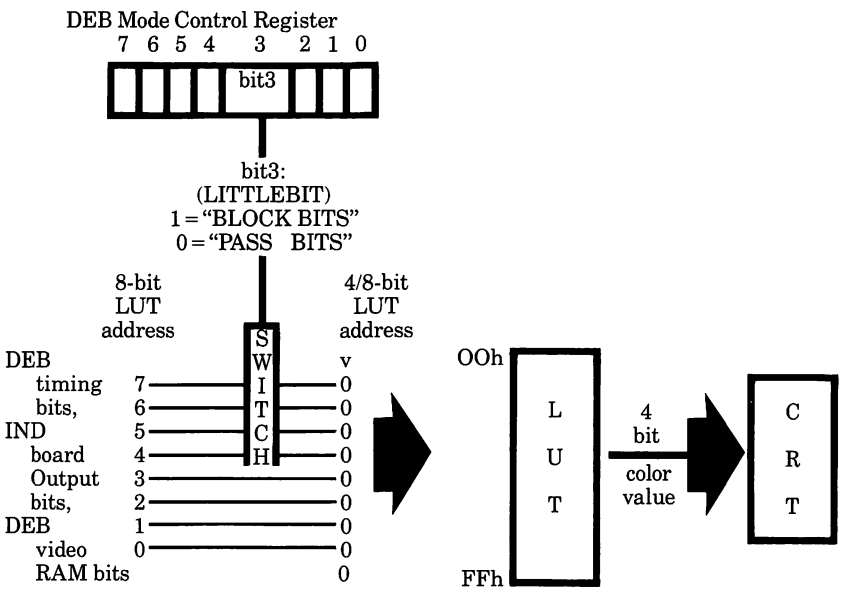
If you specify a palette position greater than the value allowed for the mode in which you are working, the value you specify will be put in that palette's highest position. For example, if you attempted to set palette position 13 to red when working in overlay mode, which has 8-position palettes, the 8th palette position would be set to red.

Note:
The following discussion covers the use of the simple palette programming functions. You can also use "Set Color Palette" to program the LUT. (For more information, see **Chapter 5, "Programming the LUT"**).

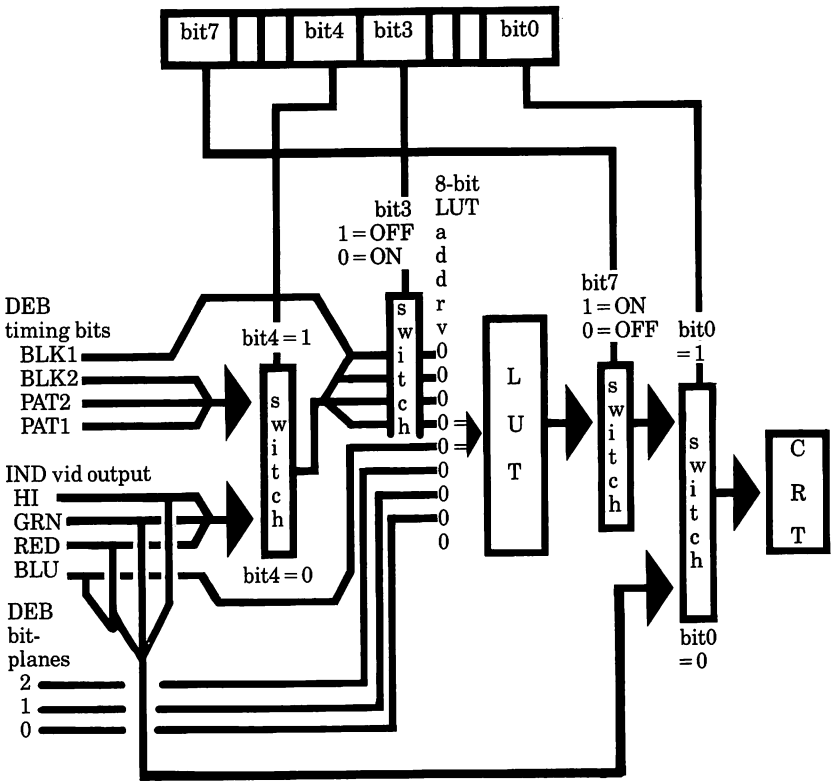
Short LUT Addresses

The DEB supports a method for you to access only the first sixteen LUT locations. This lets you use normal 16-color graphics without needing to manage all of the 256 LUT locations. You invoke this short addressing mode by a setting bit 2 in AL in the "Set Color Palette" command.

3. Short LUT Addresses



4. Modes, Address Formation, & DEB Mode Control Register



Scroll Active Page Up

This function defines a pattern that is to be displayed on the blank lines as the screen scrolls. The pattern consists of ones and zeros. Zeros are interpreted as the background color (palette position zero). Ones are interpreted as the foreground color, which is defined in BL. Care should be taken when scrolling in DEB modes, to insure that all applications set the additional argument in BH correctly.

Input

(AH)	= 6H function number for Scroll Active Page Up
(AL)	= number of lines to scroll
(CH,CL)	= row, column of upper left corner to scroll
(DH,DL)	= row, column of lower right corner to scroll
(BH)	= pattern to be used on blank lines
(BL)	= foreground color

The range of lines to be scrolled is 0 thru 23 (where 0 specifies clear screen).

Row values are 0 thru 23, column values are 0 thru 79, in DEB modes.

Valid foreground colors are specified by palette position 0-FH for 16-color graphics, and 0-7H for 8-color graphics.

Output

Contents of all registers are preserved.

Example

```
MOV AH,6   ;Scroll Active Page Up
MOV AL,LINES
MOV CH,UPROW
MOV CL,UPCOL
MOV DH,LOWROW
MOV DL,LOWCOL
MOV BH,0
MOV BL,FGCOLOR
INT  10H
```

Scroll Active Page Down

This function permits you to define a pattern that is to be displayed on the blank lines as the screen scrolls downward. The pattern consists of ones and zeros. Zeros are interpreted as the background color (palette position zero). Ones are interpreted as the foreground color, which is defined in BL. Care should be taken when scrolling in DEB modes, to insure that all applications set the additional argument in BH correctly.

Input

(AH) = 7H function number for Scroll Active Page Down
(AL) = number of lines to scroll
(CH,CL) = row, column of upper left corner to scroll
(DH,DL) = row, column of lower right corner to scroll
(BH) = pattern to be used on blank lines
(BL) = foreground color

The range of lines to be scrolled is 0 thru 23 (where 0 specifies clear screen).

Row values are 0 thru 23, column values are 0 thru 79, in DEB modes.

Valid foreground colors are specified by palette position 0-FH for 16-color graphics, and 0-7H for 8-color graphics.

Output

Contents of all registers are preserved.

Example

```
MOV AH,7  ; Scroll Active Page Down
MOV AL,LINES
MOV CH,UPROW
MOV CL,UPCOL
MOV DH,LOWROW
MOV DL,LOWCOL
MOV BH,0
MOV BL,FGCOLOR
INT  10H
```

Read Character and Attribute at Current Cursor Position

This function returns the value of the character at the current cursor position. The value of the character's foreground color is returned in AH.

Input

(AH) = 8 Function number for Read Character and Attribute at Current Cursor Position.

(BH) = Valid page numbers for DEB modes are 0 for the VDC and 80H for the DEB in overlay mode.

Output

(AL) = ASCII character code

(AH) = foreground palette position or VDC attribute

Contents of all other registers are preserved.

Example

```
MOV AH,8  ;Read CHR function
MOV BH,PAGE
INT  10H
MOV CHAR,AL ;Save CHAR/COLOR
MOV CURCOLOR,AH
```

Write Character and Attribute at Current Cursor Position

This function displays the character whose ASCII code is in register AL. The character is displayed according to the color values in BL.

Input

(AH) = 9H Write Character function
(AL) = ASCII character code
(BL) = foreground color
(BH) = page
(CX) = count of characters to write

If bit 7 of BL = 1, the color value is XOR'd with the current dots in that location. Valid page numbers for DEB modes are 0 for the VDC and 80H for the DEB in overlay mode.

Valid foreground colors are specified by palette position 0-FH for 16-color graphics, and 0-7H for 8-color graphics.

Output

Contents of all registers are preserved.

Example

```
MOV AH,9
MOV AL,CHAR
MOV BL,CURCOLOR
MOV BH,PAGE
MOV CX,1
INT 10H
```