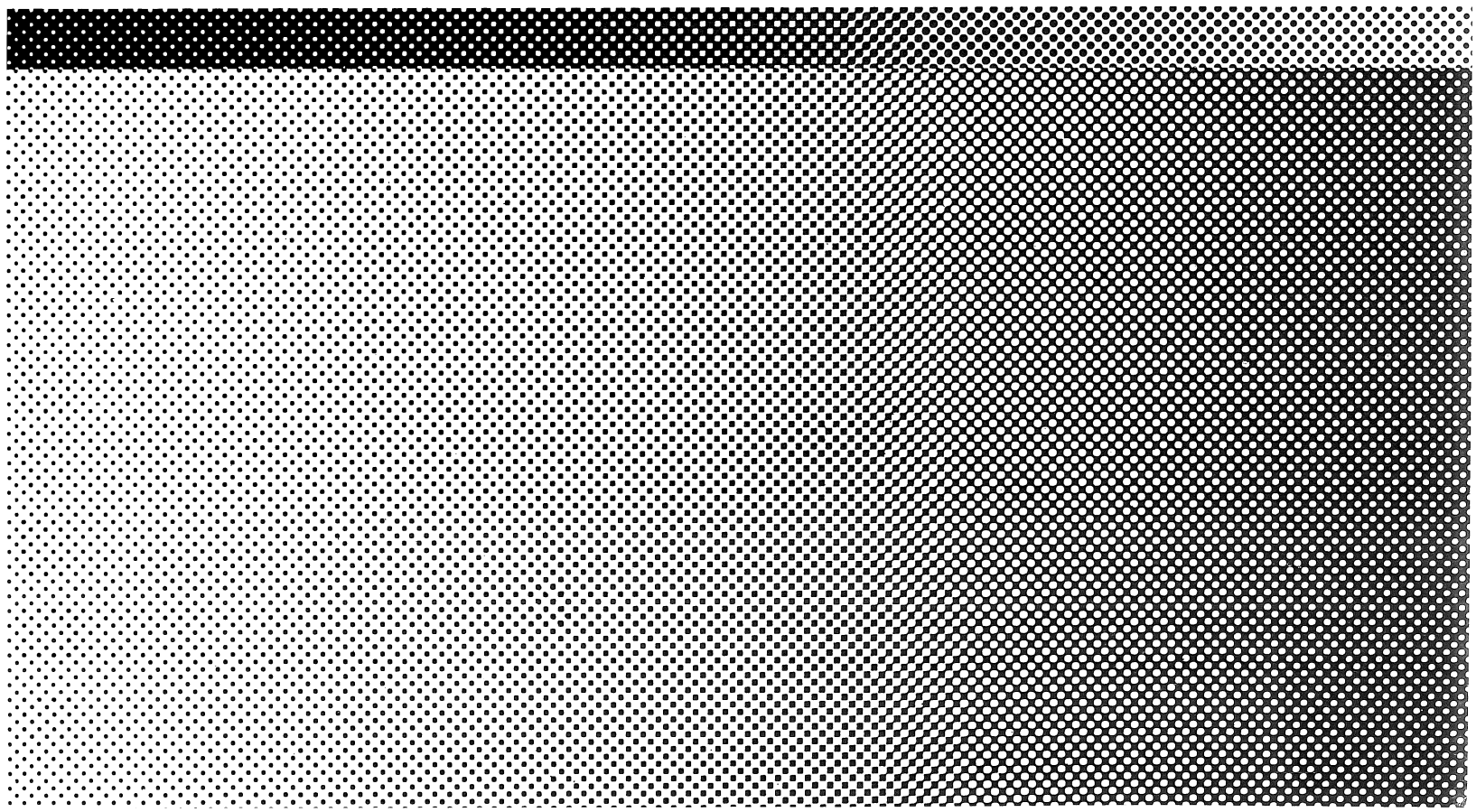




999-802-218IS
Release 2.05
Issue 1

ROM-BIOS Listing

AT&T Personal Computer
6300 PLUS



**©1986 AT&T
All Rights Reserved
Printed in USA**

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.

CDC and WREN are trademarks of Control Data Corporation
IBM is a registered trademark of International Business Machines
Micropolis is a trademark of Micropolis Corporation
MS-DOS is a registered trademark of Microsoft Corporation
Seagate is a registered trademark of Seagate Technology
UNIX is a registered trademark of AT&T

CONTENTS

ROM BIOS Listing

Index

ROM BIOS Listing

This manual contains the ROM BIOS listing for the AT&T Personal Computer 6300 PLUS. The Basic Input/Output System (BIOS) is located in the Read-Only Memory (ROM) on the PC 6300 PLUS motherboard. The ROM BIOS serves as an interface between the computer system and the input/output devices connected to the system ports. The information supplied by the ROM BIOS controls these devices. During normal operation, the ROM BIOS operates much like a driver that is resident in the PC 6300 PLUS memory space.

The index that follows the listing lets you quickly locate specific information.

```

.LIST                                ;END LISTING HERE FOR LIST 1 - START FOR LIST 1
;-----
;      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
;-----

; EQU's to include correct code for varying hardware versions%
;-----

= 0000  BETA  equ    0h                ;; non-zero for a beta test unit asm%
= 0001  G4TOD equ    01h              ; non-zero for a GEN4 TOD fix%

;-----

; EQU's to help you set rom_id variable for UNIX%
;-----

= 0000  GEN3  EQU    0                ; NORMAL GEN 3 UNIT
= 0001  TOD   EQU    1                ; GEN 4 TIME OF DAY CHIP INSTALLED
= 0002  NEWFLOP EQU  2                ; NEW FLOPPY CIRCUITRY INSTALLED
= 0004  DMACCEL EQU  4                ; DMA ACCELERATOR INSTALLED

;-----

;      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 07      rom_id     db    TOD or NEWFLOP or DMACCEL      ; ROM identifier.
          ;rom_id      db    0        ; ROM identifier.
C002 E297 R  rom_mt     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:COXX, a near call is done to the proper
          ; routine, and a far return back to the user.
          extrn bios_install:near     ; for w.d. hdu %
          extrn wx2_fmt:near          ; for w.d. format in bios %

C004 E8 E548 R call  DString           ; F000:C004      (3 bytes per near call)
C007 CB      ret                    ;                (1 byte per far return)
C008 E8 E55F R call  DCrLf            ; F000:C008

```

```

C00B CB                ret
C00C E8 E56C R        call   DColon           ; F000:C00C
C00F CB                ret
C010 E8 E578 R        call   DHexLong         ; F000:C010
C013 CB                ret
C014 E8 E582 R        call   DHexWord         ; F000:C014
C017 CB                ret
C018 E8 E589 R        call   DHexByte         ; F000:C018
C01B CB                ret
C01C E8 E596 R        call   DHexNib          ; F000:C01C
C01F CB                ret
C020 E8 E5AB R        call   DNum             ; F000:C020
C023 CB                ret
C024 E8 E5B3 R        call   DNumW            ; F000:C024
C027 CB                ret
C028 E8 E52A R        call   rom_checksum     ; F000:C028
C02B CB                ret
C02C E8 E1C0 R        call   rtc_chk          ; F000:C02C
C02F CB                ret
C030 E8 E22F R        call   memtst           ; F000:C030
C033 CB                ret
C034 E8 0000 E        call   bios_install     ; F000:C034%
C037 CB                ret
C038 E8 0000 E        call   wx2_fmt           ; F000:C038%
C03B CB                ret

C03C D912 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 ;
C ;
C ;   NAME      DATE      ACTION
C ;   ----      ----      -
C ;
C ;
C ;=====
C ;   Filename:   sysdata.src
C ;

```

```

C ;      This is the port equate and system data definition module.
C ;      (See flags.src for conditional assembly flags...)
C ;
C ;=====
C
C ;-----
C ;      Global Constants
C ;-----
C
= 0000 C abs0_seg      equ    00000h
= 0030 C stack_seg     equ    00030h
= 0040 C data_seg      equ    00040h
= B000 C para_mono     equ    0B000h
= B800 C para_graph    equ    0B800h
= F000 C code_seg      equ    0F000h
C
= 0007 C BEL           equ    007h
= 0008 C BS           equ    008h
= 000D C CR           equ    00Dh
= 000A C LF           equ    00Ah
C
= 0000 C NUL          equ    0
C
C ;-----
C ;      PC6300 PLUS Addresses
C ;-----
C
C ;-----
C ;      i8237A p_dma Controller Port Addresses
C ;-----
C
= 0000 C dma_addr_0    equ    00h    ; 16-bit address register - channel 0 - refresh
= 0001 C dma_count_0   equ    01h    ; 16-bit count register
= 0002 C dma_addr_1    equ    02h    ; 16-bit address register - channel 1 - not used
= 0003 C dma_count_1   equ    03h    ; 16-bit count register
= 0004 C dma_addr_2    equ    04h    ; 16-bit address register - channel 2 - FDU
= 0005 C dma_count_2   equ    05h    ; 16-bit count register
= 0006 C dma_addr_3    equ    06h    ; 16-bit address register - channel 3 - display
= 0007 C dma_count_3   equ    07h    ; 16-bit count register
C
= 0008 C dma_status    equ    08h    ; 8-bit read status register
= 0008 C dma_command   equ    08h    ; 8-bit write command register
= 0009 C dma_request  equ    09h    ; 4-bit write request register
= 000A C dma_mask_bit equ    0Ah    ; 4-bit (write) set/clear one mask register bit
= 000B C dma_mode     equ    0Bh    ; 6-bit write mode register
= 000C C dma_ff_clr   equ    0Ch    ; (write) clear byte pointer flip/flop
= 000D C dma_temp     equ    0Dh    ; 8-bit read temporary register
= 000D C dma_master_clr equ    0Dh    ; (write) master clear command
= 000E C dma_mask_clr  equ    0Eh    ; 4-bit (write) clear all mask register bits
= 000F C dma_mask_write equ    0Fh    ; 4-bit write all mask register bits at once
C
= 0080 C dma_seg_0     equ    080h    ; RAM refresh - 4x4-bit high nibble segment port
= 0082 C dma_seg_1     equ    082h    ; not used
= 0081 C dma_seg_2     equ    081h    ; FDU
= 0083 C dma_seg_3     equ    083h    ; display TEMP

```



```

C
C
C ;-----
C ;       i8237A p_dma controller constants
C ;-----
C
C ; dma_command port:                ; bit #0: memory-to-memory/~I/O enable
C                                     ; bit #2: controller disable
C                                     ; bit #3: compressed/~normal timing
C                                     ; bit #4: rotating/~fixed priority
C                                     ; bit #5: extended/~late write selection
C                                     ; bit #6: DREQ active low/~high
C                                     ; bit #7: DACK active high/~low
C
= 0004      C dma_cmd_disable equ    004h      ; controller disable (bit #2) command
= 0000      C dma_cmd_enable  equ    000h      ; memory-to-I/O,controller enable,normal
C                                     ; fixed priority, late write, DREQ/~DACK
C
= 0058      C dma_mode_0     equ    058h      ; channel 0, read, autoinitialize, inc-
C                                     ; rement, single mode for RAM refresh.
= 0041      C dma_mode_1     equ    041h      ; channel 1, verify, autoinit disabled,
C                                     ; increment, single mode for not used.
= 0056      C dma_mode_2     equ    056h      ; channel 2, write, autoinitialize, inc-
C                                     ; rement, single mode for FDU.
= 0043      C dma_mode_3     equ    043h      ; channel 3, verify, autoinit disabled,
C                                     ; increment, single mode for display.
C
C ; dma_mask_bit port:                ; bits #0-1: channel select
C                                     ; bit # 2: set/~clr mask bit (off/~on)
C
= 0000      C dma_unmask_0   equ    000h      ; turn on channel 0 for RAM refresh.
C
C ;-----
C ;       i8259A Programmable Interrupt Controller Port Addresses
C ;-----
C
= 0020      C pic_0         equ    020h      ; 8259A 'control' port (A0 = 0)
= 0021      C pic_1         equ    021h      ; 8259A 'data' port (A0 = 1)
C
C ;-----
C ;       i8259A Programmable Interrupt Controller Commands
C ;-----
C
= 0013      C pic_icw1     equ    013h      ; ICW1 for both master & slave pic's
C                                     ; bit #0 = 1: ICW4 to follow (w/vector base)
C                                     ; 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 4
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_neoi equ 020h ; non-specific end-of-interrupt
C
= 0060 C pic_seoi_0 equ 060h ; specific end-of-interrupt for IR0: i8254 p_timer
= 0061 C pic_seoi_1 equ 061h ; specific end-of-interrupt for IR0: i8041A kb
= 0066 C pic_seoi_6 equ 066h ; specific end-of-interrupt for IR6: fdu
C
C ;-----
C ; i8254 p_timer Port Addresses
C ;-----
C
= 0040 C p_8253_0 equ 040h ; 8254 p_timer 0 - rtc interrupt - IR0 = INT 08h
= 0041 C p_8253_1 equ 041h ; 8254 p_timer 1 - memory refresh p_dma
= 0042 C p_8253_2 equ 042h ; 8254 p_timer 2 - tone generator for speaker
= 0043 C p_8253_ctrl equ 043h ; 8254 p_timer control port
C
C ;-----
C ; i8254 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 ; i8254 p_timer Counts
C
C ; 8254 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
C
= 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 ;      (8530 not used in the 6300 PLUS)
C ;
C ;      (scc_data_x port addresses are indexed from the scc_ctl_x
C ;      port addresses. See com.scr)
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 ControlC       equ    062h    ; bit #7:   Ram parity check.
C                                     ; bit #6:   I/O channel parity check.
C                                     ; bit #1:   80287 installed
= 0065 C CommControl    equ    065h
= 0066 C sys_conf_a     equ    066h    ; bit #7:   27128/~27256 ROM's
C                                     ; bit #6 - 0 = use indiginous HDU code.
C                                     ;           1 = do not use indiginous HDU code.
C                                     ; bit #5:   not used
C                                     ; bit #4 - 0 = 80287 installed
C                                     ; bits #3 - #0: RAM configuration
= 0067 C sys_conf_b     equ    067h    ; bits #7 - #6: (number of FDU)-1
C                                     ; bits #5 - #4: reserved for monitor type
C                                     ; bit #3     Most significant bit for HDU
C                                     ;            table entry selection drive 80h
C                                     ; bit #2     Most significant bit for HDU
C                                     ;            table entry selection drive 81h
C                                     ; bit #1: - 0 = 48 tpi FDU on Drive 0
C                                     ;         - 1 = 96 tpi FDU
C                                     ; bits #0 - 0 = 48 tpi FDU on Drive 1
C                                     ;         - 1 = 96 tpi FDU
= 0080 C nmi_enable     equ    80h
= 00A0 C nmi_enable_port equ    0A0h
= 3F60 C p_trapce      equ    3F60h    ;; trapce port for power on%%
C                                     ;; reset -- above board fix%%
C

```

```

C
C ;-----
C ;      58274A Clock Calendar
C ;
C ;      (See calendar.src)
C ;-----
C
C ;-----
C ;      FDU & HDU Disk Driver Error Codes
C ;-----
C
= 0080      C time_out      equ      80h
= 0040      C seek_error    equ      40h
= 0020      C fdc_error     equ      20h
= 0010      C crc_error     equ      10h
= 0009      C dma_seg_error equ      09h
= 0008      C dma_error     equ      08h
= 0006      C media_change  equ      06h
= 0004      C sect_not_found equ     04h
= 0003      C write_protect equ     03h
= 0002      C addr_mark_error equ    02h
= 0001      C cmd_error     equ     01h
C
C ;::::::::::::::::::::::::::; FDU 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 ;-----
C ;           Game Card
C ;-----
C
= 0201      C game_card      equ    201h
C
C ;-----
C ;           Parallel Printer Interface
C ;
C ;           (prt_stat_x & prt_cmd_x port addresses are indexed from the
C ;           prt_data_x port addresses.  See prt.src.)
C ;-----
C
= 03BC      C prt_data_a     equ    03BCh
C ; prt_stat_a   equ    03BDh
C ; prt_cmd_a    equ    03BEh
C
= 0378      C prt_data_b     equ    0378h    ; on mother board
C ; prt_stat_b   equ    0379h
C ; prt_cmd_b    equ    037Ah
C
= 0278      C prt_data_c     equ    0278h
C ; prt_stat_c   equ    0279h
C ; prt_cmd_c    equ    027Ah
C
C ;-----
C ;           Color and Monochrome Video Controller
C ;
C ;           (xxxxx_data, xxxxx_mode, xxxxx_status, xxxxx_LPclear, and
C ;           xxxxx_LPPreset port addresses are indexed from the xxxxx_Pointer
C ;           port addresses for color & display.  See vid.src and graph.src.)
C ;-----
C
C ; Color Controller.
C
= 03D4      C color_pointer  equ    03D4h          ; 6845 pointer to internal regs
C
C ; Monochrome Controller.
C
= 03B4      C v_pointer      equ    03B4h          ; 6845 pointer to internal regs
C
C ;-----
C ;           INS8250 Asynchronous Communication Chip
C ;
C ;           (com_int_x, com_lctl_x, com_mctl_x, com_lstat_x, and
C ;           com_mstat_x port addresses are indexed from the com_data_x
C ;           port addresses.  See com.src.)
C ;-----
C

```

ROM BIOS Listing

```

= 03F8      C com_data_a      equ    03F8h    ; channel A 8250 data register/low byte baud
            C ; com_int_a      equ    03F9h    ; channel A 8250 high byte baud count register
= 03FA      C com_id_a        equ    03FAh    ; channel A 8250 check for presence register
            C ; com_lctl_a      equ    03FBh    ; channel A 8250 line control register
            C ; com_mctl_a      equ    03FCh    ; channel A 8250 modem control register
            C ; com_lstat_a     equ    03FDh    ; channel A 8250 line status register
            C ; com_mstat_a     equ    03FEh    ; channel A 8250 modem status register
            C
= 02F8      C com_data_b      equ    02F8h    ; channel B 8250 data register/low byte baud
            C ; com_int_b      equ    02F9h    ; channel B 8250 high byte baud count register
= 02FA      C com_id_b        equ    02FAh    ; channel B 8250 check for presence register
            C ; com_lctl_b      equ    02FBh    ; channel B 8250 line control register
            C ; com_mctl_b      equ    02FCh    ; channel B 8250 modem control register
            C ; com_lstat_b     equ    02FDh    ; channel B 8250 line status register
            C ; com_mstat_b     equ    02FEh    ; channel B 8250 modem status register
            C
            C ; -----
            C ;           Keyboard Constants
            C ; -----
            C
            C ;---- 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 scr1_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 scr1_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 scr1_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 int00locn  dd  ?          ; divide by zero
0004 C int01locn  dd  ?          ; single step trap
0008 C int02locn  dd  ?          ; nmi parity trap
000C C int03locn  dd  ?          ; break interrupt
0010 C int04locn  dd  ?          ; divide overflow
0014 C int05locn  dd  ?          ; print screen
C
0018 C int06locn  dd  ?
001C C int07locn  dd  ?
C
C ;-----
C ; i8259A Hardware Interrupt Routines
C ;-----
C
0020 C int08locn  dd  ?          ; i8254 rtc interrupt
0024 C int09locn  dd  ?          ; i8041 kb interrupt
C
0028 C int0Alocn  dd  ?
002C C int0Blocn  dd  ?
0030 C int0Clocn  dd  ?
C
0034 C int0Dlocn  dd  ?          ; hard disk interrupt
0038 C int0Elocn  dd  ?          ; floppy disk interrupt
C
003C C int0Flocn  dd  ?
C
C ;-----
C ; Software Interrupt Routines
C ;-----
C
0040 C int10locn  dd  ?          ; display request
0044 C int11locn  dd  ?          ; equipment request
0048 C int12locn  dd  ?          ; memory size request
004C C int13locn  dd  ?          ; disk I/O request
0050 C int14locn  dd  ?          ; serial communication request
0054 C int15locn  dd  ?          ; cassette request
0058 C int16locn  dd  ?          ; kb request
005C C int17locn  dd  ?          ; printer request
0060 C int18locn  dd  ?          ; cassette BASIC pointer
0064 C int19locn  dd  ?          ; boot-strap request
0068 C int1Alocn  dd  ?          ; time of day request

```

ROM BIOS Listing

```

006C ???????? C int1Blocn dd ? ; kb break pointer
0070 ???????? C int1Clocn dd ? ; p_timer break pointer
0074 ???????? C int1Dlocn dd ? ; display parameter pointer
0078 ???????? C int1Elocn dd ? ; disk parameter pointer
007C ???????? C int1Flocn 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
0008 04 [ C printer_addr dw 4 dup (?) ; 0040:0008 addresses of printers
???? ] 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 ; 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 ; Video Display Data Area
C
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 ; i8254 p_timer Data Area
C
006C ??? C t_low_order dw ? ; 0040:006C low word of i8254 p_timer count
006E ??? C t_hi_order dw ? ; 0040:006E high word of i8254 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

```

ROM BIOS Listing

```

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 ;          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
0084 ???????? C master_tbl_ptr dd    ?      ; 0040:0084 pointer to master table
C ;::::::::::;mptr dd    ?      ; 0040:0084 pointer to reseve words EGA
C
C ;-----
C ;          Reserved%
C ;-----
C
0088 ???????? C resv0       dd    ?      ; 40:88-8b %
008C ?????   C resv1       dw    ?      ; 40:8c-8d %
008E ??      C lastrate    db    ?      ; 40:8e-8f last rate used %
008F ??      C             db    ?      ; %
0090 ??      C diskstate   db    ?      ; 40:90,91 drive 0,1 media state%
0091 ??      C             db    ?      ; %
0092 ??      C             db    ?      ; 40:92,93 drive 0,1 starting state%
0093 ??      C             db    ?      ; %
C
0094 ??      C cur_cyl     db    ?      ; current cylinder for drikve 0%
0095 ??      C             db    ?      ; current cylinder for drive 1%
0096 06 [     C no_thing    dw    6 dup (?) ; 0040:0094-A1 Reserve for "compatibility"
    ?????   C
    ]      C
C
C ;-----
C ;          OS Merge Link address%
C ;-----
C
= 3FA0      C bitread     equ    3fa0h ; misc status latch%
= 0020      C pwrup1      equ    20h   ; powerup reset bit (low enable)%
C

```

```

C
00A2 ???????? C osmerge1 dd ? ; 40:A2,A3 - offset return to UNIX%
C ; 40:A4,A5 - segment return to UNIX%
C
00A6 ???????? C osmerge2 dd ? ; 40:A6,A7,A8,A9 - extension %
C
C ;-----
C ; Protected mode data space%
C ;-----
C
00AA 0C [ C gdt dw 12 dup (?) ; space for gdt AA-C1
???? ] C
C
00C2 04 [ C gdtalias dw 4 dup (?) ; gdt alias C2-C9
???? ] C
C
C
00CA ??? C seg_fail dw ? ; segment fail protected mode RAM test
C ; CA-CB
00CC ??? C off_fail dw ? ; offset fail protected mode RAM test
C ; CC-CD
00CE ??? C dwrite dw ? ; data written for p-mode RAM test
C ; CE-CF
00D0 ??? C dread dw ? ; data read for p-mode RAM test
C ; D0-D1
00D2 ?? C addr db ? ; addr space
C
00D3 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

;=====
; Filename: hdisk.asm
;
; This module includes the Western Digital Hard Disk
; controller code, wd_hdu.asm and wd_fmt.asm.
;
; Beta Release - 12/14/84 (path)
;
;=====

0000 code segment common 'ROM'
assume cs:code, ds:nothing, es:nothing, ss:nothing

.LIST

```

```

C include wd_fmt.asm
C ; SUBTTL Format Winchester Disk
C PAGE 62,132
C ; Version 07 was integrated to our working version that resides on the
C ; motherboard
C ; Version 03
C ; Bill Bailey 84/06/12 - 84/06/13
C ; Version 02
C ; Bob Hossley 84/04/10 - 84/04/12
C ; Version 01
C ; Bob Hossley 83/09/16 - 83/09/16
C ;
C ; Call: JMP WX2_FMT
C ;
C ; Purpose: Format the specified drive with the specified interleave.
C ;
C ; Entry:
C ; (AH) = Relative number of target drive. Drive 80h + d is the target.
C ; (AL) = Interleave factor.
C ;
C ; Exit: Job terminated.
C ;
C
B831 C CODE SEGMENT COMMON 'ROM'
C
C ASSUME CS:CODE,DS:CODE
C
C ; Interrupt Vectors
= 0021 C IVFC EQU 21H ;function call interrupt number
= 0013 C IVDBC EQU 13H ;Disk BIOS call
C
C ; Function call numbers
= 0001 C FCKBIN EQU 1 ;keyboard input
= 0002 C FCDISB EQU 2 ;display byte
= 0009 C FCPRSTR EQU 9 ;print string
= 004C C FCTEND EQU 04CH ;terminate
C
C ; Disk BIOS command codes
= 0007 C CCFD EQU 7 ;format drive
= 0011 C CCREC EQU 11H ;recal
= 0012 C CCRT EQU 12H ;ram test
C
B840 C ORG 0B840H
C
C PUBLIC WX2_FMT
B840 C WX2_FMT PROC NEAR
C ASSUME CS:CODE,DS:CODE
C
B840 50 C PUSH AX ;drive offset & interleave factor
B841 8C C8 C MOV AX,CS ;code segment pointer
B843 8E D8 C MOV DS,AX ;init. data segment pointer
B845 BA B8E2 R C MOV DX,OFFSET MI ;pointer to hello message
B848 B4 09 C MOV AH,FCPRSTR ;print string
B84A CD 21 C INT IVFC ;function call
C ;

```

```

B84C 58          C      POP     AX          ;Relative drive # & interleave factor
B84D 0A C0      C      OR      AL,AL        ;test for 0, change to 03H
B84F 75 02      C      JNZ     SHORT NOCHG    ;jmp if parameters specified
B851 B0 03      C      MOV     AL,3          ;drive C, interleave of 3
B853          C NOCHG:
B853 80 E4 07   C      AND     AH,07H        ;mask drive number to 0 -- 7.
B856 50         C      PUSH    AX          ;Save
C      ;;;; ADD     AH,'C'      ;generate drive letter
B857 80 C4 30   C      ADD     AH,'0'        ;generate drive letter
B85A 8A D4      C      MOV     DL,AH         ;character to display
B85C B4 02      C      MOV     AH,FCDISB     ;display byte function call #
B85E CD 21      C      INT     IVFC         ;Function call
B860 BA B9F R   C      MOV     DX,OFFSET MINT ;pointer to interleave message
B863 B4 09      C      MOV     AH,FCPRSTR    ;print string
B865 CD 21      C      INT     IVFC         ;function call
B867 58         C      POP     AX          ;Relative drive # & interleave factor
B868 50         C      PUSH    AX          ;Save
B869 E8 B8C7 R  C      CALL    DS_BY_HEX      ;display (AL) as two hex digits
C      ;Operator response
B86C B4 01      C      MOV     AH,FCKBIN     ;keyboard input function call #
B86E CD 21      C      INT     IVFC         ;function call
B870 3C 79      C      CMP     AL,'y'        ;"y" input?
B872 74 0C      C      JE      ZFMT         ;br if yes.
B874 3C 59      C      CMP     AL,'Y'        ;"Y" input?
B876 74 08      C      JE      ZFMT         ;br if yes.
B878 58         C      POP     AX          ;
B879 BA B9E0 R  C      MOV     DX,OFFSET MNOD  ;pointer to "Nothing Done Exit"
B87C 2A FF      C      SUB     BH,BH         ;no error code
B87E EB 3D      C      JMP     SHORT ZNX2    ;terminate
C      ;-----
C      ; do a ram test command to mess up the sector buffer
C      ;-----
C
B880          C ZFMT:
B880 58         C      POP     AX          ;
B881 8A D4      C      MOV     DL,AH         ;so bios will do hard disk stuff
B883 80 C2 80   C      ADD     DL,080H       ;
B886 8A E2      C      MOV     AH,DL         ;save drive number in final form
B888 50         C      PUSH    AX          ;
B889 B4 12      C      MOV     AH,CCRT        ;command
B88B CD 13      C      INT     IVDBC        ;call disk bios
B88D 8A FC      C      MOV     BH,AH         ;save error code
B88F 72 1A      C      JC      ERR          ;if error
C      ;-----
C      ; format the drive
C      ;-----
C
B891 58         C      POP     AX          ;(AL) = interleave factor
B892 50         C      PUSH    AX          ;
B893 2A F6      C      SUB     DH,DH         ;zero head number
B895 B9 0001    C      MOV     CX,1          ;sector 1, cylinder 0
B898 B4 07      C      MOV     AH,CCFD        ;format drive command code
B89A CD 13      C      INT     IVDBC        ;call disk BIOS
B89C 8A FC      C      MOV     BH,AH         ;save completion code
B89E 72 0B      C      JC      ERR          ;jump if error

```

```

C
C ;-----
C ; recal before exiting
C ;-----
B8A0 B9 0001 C      MOV    CX,1          ;sector 1, cylinder 0
B8A3 B4 11   C      MOV    AH,CCREC       ;recal command code
B8A5 CD 13   C      INT    IVDBC         ;call disk BIOS
B8A7 8A FC   C      MOV    BH,AH         ;save completion code
B8A9 73 0F   C      JNC    ZNX          ;jump if no error
B8AB 58      C ERR:  POP    AX
B8AC BA B9C5 R C      MOV    DX,OFFSET MEC   ;pointer to message
B8AF B4 09   C      MOV    AH,FCPRSTR     ;print string
B8B1 CD 21   C      INT    IVFC         ;function call
B8B3 8A C7   C      MOV    AL,BH         ;error code
B8B5 E8 B8C7 R C      CALL   DS_BY_HEX     ;display in hex
B8B8 EB 07   C      JMP    SHORT ZTEND    ;terminate
C ;-----
C
B8BA      C ZNX:
B8BA BA B9B1 R C      MOV    DX,OFFSET MSUC  ;pointer to message
B8BD      C ZNX2:
B8BD B4 09   C      MOV    AH,FCPRSTR     ;print string
B8BF CD 21   C      INT    IVFC         ;function call
B8C1      C ZTEND:
B8C1 8A C7   C      MOV    AL,BH         ;completion code=return code
B8C3 B4 4C   C      MOV    AH,FCSTEND     ;terminate
B8C5 CD 21   C      INT    IVFC         ;function call
B8C7      C WX2_FMT ENDP
C ;-----
C
C ; Call:      CALL   DS_BY_HEX
C ;           return
C ;
C ; Entry:  (AL) = byte to display
C ;
C ; Exit:  AX, CL, DL changed.
C
B8C7      C DS_BY_HEX PROC NEAR
B8C7 50     C      PUSH  AX            ;save in stack
B8C8 B1 04   C      MOV   CL,4          ;shift count
B8CA D2 E8   C      SHR   AL,CL         ;align MS 4 bits
B8CC E8 B8D3 R C      CALL  ZHEX          ;convert to hex
B8CF 58     C      POP   AX            ;byte to display
B8D0 EB 01 90 C      JMP   ZHEX          ;convert to hex, display, & exit
B8D3      C DS_BY_HEX ENDP
C
C ; Call:      CALL   ZHEX
C ;           return
C ;
C ; Entry:      (AL) bits 0 - 3 = Nibble to display
C ;
C ; Exit:  AX, DL changed.
C
B8D3      C ZHEX PROC NEAR
B8D3 24 0F   C      AND   AL,0FH        ;mask to 4 bits

```

```

B8D5 04 90          C      ADD    AL,90H
B8D7 27            C      DAA
B8D8 14 40          C      ADC    AL,40H
B8DA 27            C      DAA
B8DB 8A D0          C      MOV    DL,AL          ;hex digit to display
B8DD B4 02          C      MOV    AH,FCDISB      ;display byte function call #
B8DF CD 21          C      INT    IVFC          ;Function call
B8E1 C3            C      RET
B8E2              C  ZHEX  ENDP
C
C ;-----
B8E2 57 58 32 20 46 6F C  MI    DB    'WX2 Format Revision 3.0 (C) Copyright Western '
      72 6D 61 74 20 52 C
      65 76 69 73 69 6F C
      6E 20 33 2E 30 20 C
      28 43 29 20 43 6F C
      70 79 72 69 67 68 C
      74 20 57 65 73 74 C
      65 72 6E 20      C
B910 44 69 67 69 74 61 C      DB    'Digital Corp. 1984',0DH,0AH
      6C 20 43 6F 72 70 C
      2E 20 31 39 38 34 C
      0D 0A            C
B924 20 20 20 28 41 48 C      DB    ' (AH) = Relative drive number (0 - 7)',0DH,0AH
      29 20 3D 20 52 65 C
      6C 61 74 69 76 65 C
      20 64 72 69 76 65 C
      20 6E 75 6D 62 65 C
      72 20 28 30 20 2D C
      20 37 29 0D 0A   C
B94D 20 20 20 28 41 4C C      DB    ' (AL) = Interleave factor (3 is standard)',0DH,0AH
      29 20 3D 20 49 6E C
      74 65 72 6C 65 61 C
      76 65 20 66 61 63 C
      74 6F 72 20 28 33 C
      20 69 73 20 73 74 C
      61 6E 64 61 72 64 C
      29 0D 0A        C
B97A 50 72 65 73 73 20 C      DB    'Press "y" to begin formatting drive $'
      22 79 22 20 74 6F C
      20 62 65 67 69 6E C
      20 66 6F 72 6D 61 C
      74 74 69 6E 67 20 C
      64 72 69 76 65 20 C
      24              C
B99F 20 77 69 74 68 20 C  MINT  DB    ' with interleave $'
      69 6E 74 65 72 6C C
      65 61 76 65 20 24 C
B9B1 0D 0A 46 6F 72 6D C  MSUC  DB    0DH,0AH,'Format Successful$'
      61 74 20 53 75 63 C
      63 65 73 73 66 75 C
      6C 24            C
B9C5 0D 0A 45 72 72 6F C  MEC   DB    0DH,0AH,'Error---completion code $'
      72 2D 2D 2D 63 6F C
      6D 70 6C 65 74 69 C

```

ROM BIOS Listing

```

        6F 6E 20 63 6F 64   C
        65 20 24           C
B9E0   0D 0A 4E 6F 74 68   C MNOD   DB      0DH,0AH,'Nothing Done Exit$'
        69 6E 67 20 44 6F   C
        6E 65 20 45 78 69   C
        74 24               C
B9F4   38 35 2F 30 31 2F   C       DB      " 85/01/29"           ;release date
        32 39               C
B9FC                   C CODE   ENDS
                   C ;       END
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
C060                   fontlo16 proc near           ; System Font Table for M24
C
C060   00 00 00 00 00 00   C       DB  000h,000h,000h,000h,000h,000h,000h,000h
        00 00               C
C068   00 00 00 00 00 00   C       DB  000h,000h,000h,000h,000h,000h,000h,000h   ; 0
        00 00               C
C070   00 00 7E 81 A5 81   C       DB  000h,000h,07eh,081h,0a5h,081h,081h,0bdh
        81 BD               C
C078   99 81 7E 00 00 00   C       DB  099h,081h,07eh,000h,000h,000h,000h,000h   ; 1
        00 00               C
C080   00 00 7E FF DB FF   C       DB  000h,000h,07eh,0ffh,0dbh,0ffh,0ffh,0c3h
        FF C3               C
C088   E7 FF 7E 00 00 00   C       DB  0e7h,0ffh,07eh,000h,000h,000h,000h,000h   ; 2
        00 00               C
C090   00 00 00 36 7F 7F   C       DB  000h,000h,000h,036h,07fh,07fh,07fh,07fh
        7F 7F               C
C098   3E 1C 08 00 00 00   C       DB  03eh,01ch,008h,000h,000h,000h,000h,000h   ; 3
        00 00               C
C0A0   00 00 00 08 1C 3E   C       DB  000h,000h,000h,008h,01ch,03eh,07fh,03eh
        7F 3E               C
C0A8   1C 08 00 00 00 00   C       DB  01ch,008h,000h,000h,000h,000h,000h,000h   ; 4
        00 00               C
C0B0   00 00 18 3C 3C E7   C       DB  000h,000h,018h,03ch,03ch,0e7h,0e7h,0e7h
        E7 E7               C
C0B8   18 18 3C 00 00 00   C       DB  018h,018h,03ch,000h,000h,000h,000h,000h   ; 5
        00 00               C
C0C0   00 00 18 3C 7E FF   C       DB  000h,000h,018h,03ch,07eh,0ffh,0ffh,07eh
        FF 7E               C
C0C8   18 18 3C 00 00 00   C       DB  018h,018h,03ch,000h,000h,000h,000h,000h   ; 6
        00 00               C
C0D0   00 00 00 00 00 18   C       DB  000h,000h,000h,000h,000h,018h,03ch,03ch
        3C 3C               C
C0D8   18 00 00 00 00 00   C       DB  018h,000h,000h,000h,000h,000h,000h,000h   ; 7
        00 00               C
C0E0   FF FF FF FF FF E7   C       DB  0ffh,0ffh,0ffh,0ffh,0ffh,0e7h,0c3h,0c3h
        C3 C3               C
C0E8   E7 FF FF FF FF FF   C       DB  0e7h,0ffh,0ffh,0ffh,0ffh,0ffh,0ffh,0ffh   ; 8
        FF FF               C
C0F0   00 00 00 00 3C 24   C       DB  000h,000h,000h,000h,03ch,024h,042h,042h
        42 42               C

```


C0F8	24 3C 00 00 00 00 00 00	C C	DB 024h,03ch,000h,000h,000h,000h,000h,000h	;	9
C100	FF FF FF FF C3 DB BD BD	C C	DB 0ffh,0ffh,0ffh,0ffh,0c3h,0dbh,0bdh,0bdh		
C108	DB C3 FF FF FF FF FF FF	C C	DB 0dbh,0c3h,0ffh,0ffh,0ffh,0ffh,0ffh,0ffh	;	a
C110	00 00 1F 07 0D 19 78 CC	C C	DB 000h,000h,01fh,007h,00dh,019h,078h,0cch		
C118	CC CC 78 00 00 00 00 00	C C	DB 0cch,0cch,078h,000h,000h,000h,000h,000h	;	b
C120	00 00 3C 66 66 66 3C 18	C C	DB 000h,000h,03ch,066h,066h,066h,03ch,018h		
C128	7E 18 18 00 00 00 00 00	C C	DB 07eh,018h,018h,000h,000h,000h,000h,000h	;	c
C130	00 00 0C 0A 09 09 09 0A	C C	DB 000h,000h,00ch,00ah,009h,009h,009h,00ah		
C138	08 38 78 78 30 00 00 00	C C	DB 008h,038h,078h,078h,030h,000h,000h,000h	;	d
C140	00 00 1F 11 1F 11 11 11	C C	DB 000h,000h,01fh,011h,01fh,011h,011h,011h		
C148	13 37 77 72 20 00 00 00	C C	DB 013h,037h,077h,072h,020h,000h,000h,000h	;	e
C150	00 00 18 18 DB 3C E7 3C	C C	DB 000h,000h,018h,018h,0dbh,03ch,0e7h,03ch		
C158	DB 18 18 00 00 00 00 00	C C	DB 0dbh,018h,018h,000h,000h,000h,000h,000h	;	f
C160	00 00 40 60 70 7C 7F 7C	C C	DB 000h,000h,040h,060h,070h,07ch,07fh,07ch		
C168	70 60 40 00 00 00 00 00	C C	DB 070h,060h,040h,000h,000h,000h,000h,000h	;	10
C170	00 00 01 03 07 1F 7F 1F	C C	DB 000h,000h,001h,003h,007h,01fh,07fh,01fh		
C178	07 03 01 00 00 00 00 00	C C	DB 007h,003h,001h,000h,000h,000h,000h,000h	;	11
C180	00 00 18 3C 7E 18 18 18	C C	DB 000h,000h,018h,03ch,07eh,018h,018h,018h		
C188	7E 3C 18 00 00 00 00 00	C C	DB 07eh,03ch,018h,000h,000h,000h,000h,000h	;	12
C190	00 00 33 33 33 33 33 33	C C	DB 000h,000h,033h,033h,033h,033h,033h,033h		
C198	00 33 33 00 00 00 00 00	C C	DB 000h,033h,033h,000h,000h,000h,000h,000h	;	13
C1A0	00 00 7F DB DB DB 7B 1B	C C	DB 000h,000h,07fh,0dbh,0dbh,0dbh,07bh,01bh		
C1A8	1B 1B 1B 00 00 00 00 00	C C	DB 01bh,01bh,01bh,000h,000h,000h,000h,000h	;	14
C1B0	00 3E 63 30 1C 36 63 63	C C	DB 000h,03eh,063h,030h,01ch,036h,063h,063h		
C1B8	36 1C 06 63 3E 00 00 00	C C	DB 036h,01ch,006h,063h,03eh,000h,000h,000h	;	15
C1C0	00 00 00 00 00 00 00 00	C C	DB 000h,000h,000h,000h,000h,000h,000h,000h		
C1C8	7F 7F 7F 00 00 00 00 00	C C	DB 07fh,07fh,07fh,000h,000h,000h,000h,000h	;	16
C1D0	00 00 18 3C 7E 18	C	DB 000h,000h,018h,03ch,07eh,018h,018h,018h		

ROM BIOS Listing

18 18	C		
C1D8 7E 3C 18 FF 00 00	C	DB 07eh,03ch,018h,0ffh,000h,000h,000h,000h	; 17
00 00	C		
C1E0 00 00 18 3C 7E 18	C	DB 000h,000h,018h,03ch,07eh,018h,018h,018h	
18 18	C		
C1E8 18 18 18 00 00 00	C	DB 018h,018h,018h,000h,000h,000h,000h,000h	; 18
00 00	C		
C1F0 00 00 18 18 18 18	C	DB 000h,000h,018h,018h,018h,018h,018h,018h	
18 18	C		
C1F8 7E 3C 18 00 00 00	C	DB 07eh,03ch,018h,000h,000h,000h,000h,000h	; 19
00 00	C		
C200 00 00 00 00 0C 06	C	DB 000h,000h,000h,000h,00ch,006h,07fh,006h	
7F 06	C		
C208 0C 00 00 00 00 00	C	DB 00ch,000h,000h,000h,000h,000h,000h,000h	; 1a
00 00	C		
C210 00 00 00 00 18 30	C	DB 000h,000h,000h,000h,018h,030h,07fh,030h	
7F 30	C		
C218 18 00 00 00 00 00	C	DB 018h,000h,000h,000h,000h,000h,000h,000h	; 1b
00 00	C		
C220 00 00 00 00 60 60	C	DB 000h,000h,000h,000h,060h,060h,060h,060h	
60 60	C		
C228 7F 7F 00 00 00 00	C	DB 07fh,07fh,000h,000h,000h,000h,000h,000h	; 1c
00 00	C		
C230 00 00 00 00 24 42	C	DB 000h,000h,000h,000h,024h,042h,0ffh,042h	
FF 42	C		
C238 24 00 00 00 00 00	C	DB 024h,000h,000h,000h,000h,000h,000h,000h	; 1d
00 00	C		
C240 00 00 00 00 00 00	C	DB 000h,000h,000h,000h,000h,000h,000h,018h	
00 18	C		
C248 3C 7E FF 00 00 00	C	DB 03ch,07eh,0ffh,000h,000h,000h,000h,000h	; 1e
00 00	C		
C250 00 00 00 00 00 FF	C	DB 000h,000h,000h,000h,000h,0ffh,07eh,03ch	
7E 3C	C		
C258 18 00 00 00 00 00	C	DB 018h,000h,000h,000h,000h,000h,000h,000h	; 1f
00 00	C		
C260 00 00 00 00 00 00	C	DB 000h,000h,000h,000h,000h,000h,000h,000h	
00 00	C		
C268 00 00 00 00 00 00	C	DB 000h,000h,000h,000h,000h,000h,000h,000h	; ' ' 20
00 00	C		
C270 00 00 18 3C 3C 3C	C	DB 000h,000h,018h,03ch,03ch,03ch,018h,018h	
18 18	C		
C278 00 18 18 00 00 00	C	DB 000h,018h,018h,000h,000h,000h,000h,000h	; '! ' 21
00 00	C		
C280 00 66 66 66 24 00	C	DB 000h,066h,066h,066h,024h,000h,000h,000h	
00 00	C		
C288 00 00 00 00 00 00	C	DB 000h,000h,000h,000h,000h,000h,000h,000h	; ' ' ' 22
00 00	C		
C290 00 00 36 36 7F 36	C	DB 000h,000h,036h,036h,07fh,036h,036h,036h	
36 36	C		
C298 7F 36 36 00 00 00	C	DB 07fh,036h,036h,000h,000h,000h,000h,000h	; '# ' 23
00 00	C		
C2A0 08 08 3E 63 60 60	C	DB 008h,008h,03eh,063h,060h,060h,03eh,003h	
3E 03	C		
C2A8 03 63 3E 08 08 00	C	DB 003h,063h,03eh,008h,008h,000h,000h,000h	; '\$ ' 24
00 00	C		

C2B0	00 00 00 61 63 06	C	DB	000h,000h,000h,061h,063h,006h,00ch,018h	
	0C 18	C			
C2B8	30 63 43 00 00 00	C	DB	030h,063h,043h,000h,000h,000h,000h,000h	; '%' 25
	00 00	C			
C2C0	00 00 1C 36 36 1C	C	DB	000h,000h,01ch,036h,036h,01ch,03bh,06eh	
	3B 6E	C			
C2C8	66 66 3B 00 00 00	C	DB	066h,066h,03bh,000h,000h,000h,000h,000h	; '&' 26
	00 00	C			
C2D0	00 30 30 30 60 00	C	DB	000h,030h,030h,030h,060h,000h,000h,000h	
	00 00	C			
C2D8	00 00 00 00 00 00	C	DB	000h,000h,000h,000h,000h,000h,000h,000h	; ''' 27
	00 00	C			
C2E0	00 00 0C 18 30 30	C	DB	000h,000h,00ch,018h,030h,030h,030h,030h	
	30 30	C			
C2E8	30 18 0C 00 00 00	C	DB	030h,018h,00ch,000h,000h,000h,000h,000h	; '(' 28
	00 00	C			
C2F0	00 00 30 18 0C 0C	C	DB	000h,000h,030h,018h,00ch,00ch,00ch,00ch	
	0C 0C	C			
C2F8	0C 18 30 00 00 00	C	DB	00ch,018h,030h,000h,000h,000h,000h,000h	; ')' 29
	00 00	C			
C300	00 00 00 00 66 3C	C	DB	000h,000h,000h,000h,066h,03ch,07eh,03ch	
	7E 3C	C			
C308	66 00 00 00 00 00	C	DB	066h,000h,000h,000h,000h,000h,000h,000h	; '*' 2a
	00 00	C			
C310	00 00 00 00 18 18	C	DB	000h,000h,000h,000h,018h,018h,07eh,018h	
	7E 18	C			
C318	18 00 00 00 00 00	C	DB	018h,000h,000h,000h,000h,000h,000h,000h	; '+' 2b
	00 00	C			
C320	00 00 00 00 00 00	C	DB	000h,000h,000h,000h,000h,000h,000h,000h	
	00 00	C			
C328	18 18 18 30 00 00	C	DB	018h,018h,018h,030h,000h,000h,000h,000h	; ',' 2c
	00 00	C			
C330	00 00 00 00 00 00	C	DB	000h,000h,000h,000h,000h,000h,07eh,000h	
	7E 00	C			
C338	00 00 00 00 00 00	C	DB	000h,000h,000h,000h,000h,000h,000h,000h	; '-' 2d
	00 00	C			
C340	00 00 00 00 00 00	C	DB	000h,000h,000h,000h,000h,000h,000h,000h	
	00 00	C			
C348	00 18 18 00 00 00	C	DB	000h,018h,018h,000h,000h,000h,000h,000h	; '.' 2e
	00 00	C			
C350	00 00 01 03 06 0C	C	DB	000h,000h,001h,003h,006h,00ch,018h,030h	
	18 30	C			
C358	60 40 00 00 00 00	C	DB	060h,040h,000h,000h,000h,000h,000h,000h	; '/' 2f
	00 00	C			
C360	00 00 3E 63 67 6F	C	DB	000h,000h,03eh,063h,067h,06fh,07bh,073h	
	7B 73	C			
C368	63 63 3E 00 00 00	C	DB	063h,063h,03eh,000h,000h,000h,000h,000h	; '0' 30
	00 00	C			
C370	00 00 0C 1C 3C 0C	C	DB	000h,000h,00ch,01ch,03ch,00ch,00ch,00ch	
	0C 0C	C			
C378	0C 0C 3F 00 00 00	C	DB	00ch,00ch,03fh,000h,000h,000h,000h,000h	; '1' 31
	00 00	C			
C380	00 00 3E 63 03 06	C	DB	000h,000h,03eh,063h,003h,006h,00ch,018h	
	0C 18	C			
C388	30 63 7F 00 00 00	C	DB	030h,063h,07fh,000h,000h,000h,000h,000h	; '2' 32

ROM BIOS Listing

00 00	C		
C390 00 00 3E 63 03 03	C	DB 000h,000h,03eh,063h,003h,003h,01eh,003h	
1E 03	C		
C398 03 63 3E 00 00 00	C	DB 003h,063h,03eh,000h,000h,000h,000h,000h	; '3' 33
00 00	C		
C3A0 00 00 06 0E 1E 36	C	DB 000h,000h,006h,00eh,01eh,036h,066h,07fh	
66 7F	C		
C3A8 06 06 0F 00 00 00	C	DB 006h,006h,00fh,000h,000h,000h,000h,000h	; '4' 34
00 00	C		
C3B0 00 00 7E 60 60 60	C	DB 000h,000h,07eh,060h,060h,060h,07eh,003h	
7E 03	C		
C3B8 03 63 3E 00 00 00	C	DB 003h,063h,03eh,000h,000h,000h,000h,000h	; '5' 35
00 00	C		
C3C0 00 00 1C 30 60 60	C	DB 000h,000h,01ch,030h,060h,060h,07eh,063h	
7E 63	C		
C3C8 63 63 3E 00 00 00	C	DB 063h,063h,03eh,000h,000h,000h,000h,000h	; '6' 36
00 00	C		
C3D0 00 00 7F 63 03 06	C	DB 000h,000h,07fh,063h,003h,006h,00ch,018h	
0C 18	C		
C3D8 18 18 18 00 00 00	C	DB 018h,018h,018h,000h,000h,000h,000h,000h	; '7' 37
00 00	C		
C3E0 00 00 3E 63 63 63	C	DB 000h,000h,03eh,063h,063h,063h,03eh,063h	
3E 63	C		
C3E8 63 63 3E 00 00 00	C	DB 063h,063h,03eh,000h,000h,000h,000h,000h	; '8' 38
00 00	C		
C3F0 00 00 3E 63 63 63	C	DB 000h,000h,03eh,063h,063h,063h,03fh,003h	
3F 03	C		
C3F8 03 06 1C 00 00 00	C	DB 003h,006h,01ch,000h,000h,000h,000h,000h	; '9' 39
00 00	C		
C400 00 00 00 00 18 18	C	DB 000h,000h,000h,000h,018h,018h,000h,000h	
00 00	C		
C408 00 18 18 00 00 00	C	DB 000h,018h,018h,000h,000h,000h,000h,000h	; ':' 3a
00 00	C		
C410 00 00 00 00 18 18	C	DB 000h,000h,000h,000h,018h,018h,000h,000h	
00 00	C		
C418 00 18 18 30 00 00	C	DB 000h,018h,018h,030h,000h,000h,000h,000h	; ';' 3b
00 00	C		
C420 00 00 06 0C 18 30	C	DB 000h,000h,006h,00ch,018h,030h,060h,030h	
60 30	C		
C428 18 0C 06 00 00 00	C	DB 018h,00ch,006h,000h,000h,000h,000h,000h	; '<' 3c
00 00	C		
C430 00 00 00 00 7E 00	C	DB 000h,000h,000h,000h,07eh,000h,000h,000h	
00 00	C		
C438 7E 00 00 00 00 00	C	DB 07eh,000h,000h,000h,000h,000h,000h,000h	; '=' 3d
00 00	C		
C440 00 00 60 30 18 0C	C	DB 000h,000h,060h,030h,018h,00ch,006h,00ch	
06 0C	C		
C448 18 30 60 00 00 00	C	DB 018h,030h,060h,000h,000h,000h,000h,000h	; '>' 3e
00 00	C		
C450 00 00 3E 63 63 06	C	DB 000h,000h,03eh,063h,063h,006h,00ch,00ch	
0C 0C	C		
C458 00 0C 0C 00 00 00	C	DB 000h,00ch,00ch,000h,000h,000h,000h,000h	; '?' 3f
00 00	C		
C460 00 00 3E 63 63 6F	C	DB 000h,000h,03eh,063h,063h,06fh,06fh,06fh	
6F 6F	C		

C468	6E 60 3E 00 00 00 00 00	C C	DB 06eh,060h,03eh,000h,000h,000h,000h,000h	;'@' 40
C470	00 00 08 1C 36 63 63 7F	C C	DB 000h,000h,008h,01ch,036h,063h,063h,07fh	
C478	63 63 63 00 00 00 00 00	C C	DB 063h,063h,063h,000h,000h,000h,000h,000h	;'A' 41
C480	00 00 7E 33 33 33 3E 33	C C	DB 000h,000h,07eh,033h,033h,033h,03eh,033h	
C488	33 33 7E 00 00 00 00 00	C C	DB 033h,033h,07eh,000h,000h,000h,000h,000h	;'B' 42
C490	00 00 1E 33 60 60 60 60	C C	DB 000h,000h,01eh,033h,060h,060h,060h,060h	
C498	60 33 1E 00 00 00 00 00	C C	DB 060h,033h,01eh,000h,000h,000h,000h,000h	;'C' 43
C4A0	00 00 7C 36 33 33 33 33	C C	DB 000h,000h,07ch,036h,033h,033h,033h,033h	
C4A8	33 36 7C 00 00 00 00 00	C C	DB 033h,036h,07ch,000h,000h,000h,000h,000h	;'D' 44
C4B0	00 00 7F 33 30 34 3C 34	C C	DB 000h,000h,07fh,033h,030h,034h,03ch,034h	
C4B8	30 33 7F 00 00 00 00 00	C C	DB 030h,033h,07fh,000h,000h,000h,000h,000h	;'E' 45
C4C0	00 00 7F 33 30 34 3C 34	C C	DB 000h,000h,07fh,033h,030h,034h,03ch,034h	
C4C8	30 30 78 00 00 00 00 00	C C	DB 030h,030h,078h,000h,000h,000h,000h,000h	;'F' 46
C4D0	00 00 1E 33 60 60 60 6F	C C	DB 000h,000h,01eh,033h,060h,060h,060h,06fh	
C4D8	63 33 1D 00 00 00 00 00	C C	DB 063h,033h,01dh,000h,000h,000h,000h,000h	;'G' 47
C4E0	00 00 63 63 63 63 7F 63	C C	DB 000h,000h,063h,063h,063h,063h,07fh,063h	
C4E8	63 63 63 00 00 00 00 00	C C	DB 063h,063h,063h,000h,000h,000h,000h,000h	;'H' 48
C4F0	00 00 3C 18 18 18 18 18	C C	DB 000h,000h,03ch,018h,018h,018h,018h,018h	
C4F8	18 18 3C 00 00 00 00 00	C C	DB 018h,018h,03ch,000h,000h,000h,000h,000h	;'I' 49
C500	00 00 0F 06 06 06 06 06	C C	DB 000h,000h,00fh,006h,006h,006h,006h,006h	
C508	66 66 3C 00 00 00 00 00	C C	DB 066h,066h,03ch,000h,000h,000h,000h,000h	;'J' 4a
C510	00 00 73 33 36 36 3C 36	C C	DB 000h,000h,073h,033h,036h,036h,03ch,036h	
C518	36 33 73 00 00 00 00 00	C C	DB 036h,033h,073h,000h,000h,000h,000h,000h	;'K' 4b
C520	00 00 78 30 30 30 30 30	C C	DB 000h,000h,078h,030h,030h,030h,030h,030h	
C528	30 33 7F 00 00 00 00 00	C C	DB 030h,033h,07fh,000h,000h,000h,000h,000h	;'L' 4c
C530	00 00 63 77 7F 6B 63 63	C C	DB 000h,000h,063h,077h,07fh,06bh,063h,063h	
C538	63 63 63 00 00 00 00 00	C C	DB 063h,063h,063h,000h,000h,000h,000h,000h	;'M' 4d
C540	00 00 63 73 7B 7F	C	DB 000h,000h,063h,073h,07bh,07fh,06fh,067h	

ROM BIOS Listing

	6F 67	C		
C548	63 63 63 00 00 00 00 00	C C	DB 063h,063h,063h,000h,000h,000h,000h,000h	; 'N' 4e
C550	00 00 1C 36 63 63 63 63	C C	DB 000h,000h,01ch,036h,063h,063h,063h,063h	
C558	63 36 1C 00 00 00 00 00	C C	DB 063h,036h,01ch,000h,000h,000h,000h,000h	; 'O' 4f
C560	00 00 7E 33 33 33 3E 30	C C	DB 000h,000h,07eh,033h,033h,033h,03eh,030h	
C568	30 30 78 00 00 00 00 00	C C	DB 030h,030h,078h,000h,000h,000h,000h,000h	; 'P' 50
C570	00 00 1C 36 63 63 63 63	C C	DB 000h,000h,01ch,036h,063h,063h,063h,063h	
C578	6B 3E 1C 06 03 00 00 00	C C	DB 06bh,03eh,01ch,006h,003h,000h,000h,000h	; 'Q' 51
C580	00 00 7E 33 33 33 3E 36	C C	DB 000h,000h,07eh,033h,033h,033h,03eh,036h	
C588	33 33 73 00 00 00 00 00	C C	DB 033h,033h,073h,000h,000h,000h,000h,000h	; 'R' 52
C590	00 00 3E 63 63 30 1C 06	C C	DB 000h,000h,03eh,063h,063h,030h,01ch,006h	
C598	63 63 3E 00 00 00 00 00	C C	DB 063h,063h,03eh,000h,000h,000h,000h,000h	; 'S' 53
C5A0	00 00 7E 5A 18 18 18 18	C C	DB 000h,000h,07eh,05ah,018h,018h,018h,018h	
C5A8	18 18 3C 00 00 00 00 00	C C	DB 018h,018h,03ch,000h,000h,000h,000h,000h	; 'T' 54
C5B0	00 00 63 63 63 63 63 63	C C	DB 000h,000h,063h,063h,063h,063h,063h,063h	
C5B8	63 63 3E 00 00 00 00 00	C C	DB 063h,063h,03eh,000h,000h,000h,000h,000h	; 'U' 55
C5C0	00 00 63 63 63 63 63 63	C C	DB 000h,000h,063h,063h,063h,063h,063h,063h	
C5C8	36 1C 08 00 00 00 00 00	C C	DB 036h,01ch,008h,000h,000h,000h,000h,000h	; 'V' 56
C5D0	00 00 63 63 63 63 63 6B	C C	DB 000h,000h,063h,063h,063h,063h,063h,06bh	
C5D8	6B 7F 36 00 00 00 00 00	C C	DB 06bh,07fh,036h,000h,000h,000h,000h,000h	; 'W' 57
C5E0	00 00 63 63 63 36 1C 36	C C	DB 000h,000h,063h,063h,063h,036h,01ch,036h	
C5E8	63 63 63 00 00 00 00 00	C C	DB 063h,063h,063h,000h,000h,000h,000h,000h	; 'X' 58
C5F0	00 00 66 66 66 66 66 3C	C C	DB 000h,000h,066h,066h,066h,066h,066h,03ch	
C5F8	18 18 3C 00 00 00 00 00	C C	DB 018h,018h,03ch,000h,000h,000h,000h,000h	; 'Y' 59
C600	00 00 7F 63 06 0C 18 30	C C	DB 000h,000h,07fh,063h,006h,00ch,018h,030h	
C608	60 63 7F 00 00 00 00 00	C C	DB 060h,063h,07fh,000h,000h,000h,000h,000h	; 'Z' 5a
C610	00 00 3C 30 30 30 30 30	C C	DB 000h,000h,03ch,030h,030h,030h,030h,030h	
C618	30 30 3C 00 00 00 00 00	C C	DB 030h,030h,03ch,000h,000h,000h,000h,000h	; '[' 5b

C620	00 00 40 60 30 18	C	DB	000h,000h,040h,060h,030h,018h,00ch,006h	
	0C 06	C			
C628	03 01 00 00 00 00	C	DB	003h,001h,000h,000h,000h,000h,000h,000h	;'^' 5c
	00 00	C			
C630	00 00 3C 0C 0C 0C	C	DB	000h,000h,03ch,00ch,00ch,00ch,00ch,00ch	
	0C 0C	C			
C638	0C 0C 3C 00 00 00	C	DB	00ch,00ch,03ch,000h,000h,000h,000h,000h	;']' 5d
	00 00	C			
C640	08 1C 36 63 00 00	C	DB	008h,01ch,036h,063h,000h,000h,000h,000h	
	00 00	C			
C648	00 00 00 00 00 00	C	DB	000h,000h,000h,000h,000h,000h,000h,000h	;'^' 5e
	00 00	C			
C650	00 00 00 00 00 00	C	DB	000h,000h,000h,000h,000h,000h,000h,000h	
	00 00	C			
C658	00 00 00 00 00 00	C	DB	000h,000h,000h,000h,000h,000h,07fh,000h	;['_' 5f
	7F 00	C			
C660	18 18 0C 00 00 00	C	DB	018h,018h,00ch,000h,000h,000h,000h,000h	
	00 00	C			
C668	00 00 00 00 00 00	C	DB	000h,000h,000h,000h,000h,000h,000h,000h	;'''' 60
	00 00	C			
C670	00 00 00 00 00 3C	C	DB	000h,000h,000h,000h,000h,03ch,006h,03eh	
	06 3E	C			
C678	66 66 3B 00 00 00	C	DB	066h,066h,03bh,000h,000h,000h,000h,000h	; 'a' 61
	00 00	C			
C680	00 00 70 30 30 3E	C	DB	000h,000h,070h,030h,030h,03eh,033h,033h	
	33 33	C			
C688	33 33 6E 00 00 00	C	DB	033h,033h,06eh,000h,000h,000h,000h,000h	; 'b' 62
	00 00	C			
C690	00 00 00 00 00 3E	C	DB	000h,000h,000h,000h,000h,03eh,063h,060h	
	63 60	C			
C698	60 63 3E 00 00 00	C	DB	060h,063h,03eh,000h,000h,000h,000h,000h	; 'c' 63
	00 00	C			
C6A0	00 00 0E 06 06 3E	C	DB	000h,000h,00eh,006h,006h,03eh,066h,066h	
	66 66	C			
C6A8	66 66 3B 00 00 00	C	DB	066h,066h,03bh,000h,000h,000h,000h,000h	; 'd' 64
	00 00	C			
C6B0	00 00 00 00 00 3E	C	DB	000h,000h,000h,000h,000h,03eh,063h,07fh	
	63 7F	C			
C6B8	60 63 3E 00 00 00	C	DB	060h,063h,03eh,000h,000h,000h,000h,000h	; 'e' 65
	00 00	C			
C6C0	00 00 1E 33 30 7C	C	DB	000h,000h,01eh,033h,030h,07ch,030h,030h	
	30 30	C			
C6C8	30 30 78 00 00 00	C	DB	030h,030h,078h,000h,000h,000h,000h,000h	; 'f' 66
	00 00	C			
C6D0	00 00 00 00 00 3B	C	DB	000h,000h,000h,000h,000h,03bh,066h,066h	
	66 66	C			
C6D8	66 66 3E 06 66 3C	C	DB	066h,066h,03eh,006h,066h,03ch,000h,000h	; 'g' 67
	00 00	C			
C6E0	00 00 70 30 30 36	C	DB	000h,000h,070h,030h,030h,036h,03bh,033h	
	3B 33	C			
C6E8	33 33 73 00 00 00	C	DB	033h,033h,073h,000h,000h,000h,000h,000h	; 'h' 68
	00 00	C			
C6F0	00 00 0C 0C 00 1C	C	DB	000h,000h,00ch,00ch,000h,01ch,00ch,00ch	
	0C 0C	C			
C6F8	0C 0C 1E 00 00 00	C	DB	00ch,00ch,01eh,000h,000h,000h,000h,000h	; 'i' 69

ROM BIOS Listing

	00 00	C		
C700	00 00 0C 0C 00 1C	C	DB	000h,000h,00ch,00ch,000h,01ch,00ch,00ch
	0C 0C	C		
C708	0C 0C 0C 0C CC 78	C	DB	00ch,00ch,00ch,00ch,0cch,078h,000h,000h ; 'j' 6a
	00 00	C		
C710	00 00 70 30 30 33	C	DB	000h,000h,070h,030h,030h,033h,036h,03ch
	36 3C	C		
C718	36 33 73 00 00 00	C	DB	036h,033h,073h,000h,000h,000h,000h,000h ; 'k' 6b
	00 00	C		
C720	00 00 1C 0C 0C 0C	C	DB	000h,000h,01ch,00ch,00ch,00ch,00ch,00ch
	0C 0C	C		
C728	0C 0C 1E 00 00 00	C	DB	00ch,00ch,01eh,000h,000h,000h,000h,000h ; 'l' 6c
	00 00	C		
C730	00 00 00 00 00 66	C	DB	000h,000h,000h,000h,000h,066h,07fh,06bh
	7F 6B	C		
C738	6B 6B 6B 00 00 00	C	DB	06bh,06bh,06bh,000h,000h,000h,000h,000h ; 'm' 6d
	00 00	C		
C740	00 00 00 00 00 6E	C	DB	000h,000h,000h,000h,000h,06eh,033h,033h
	33 33	C		
C748	33 33 33 00 00 00	C	DB	033h,033h,033h,000h,000h,000h,000h,000h ; 'n' 6e
	00 00	C		
C750	00 00 00 00 00 3E	C	DB	000h,000h,000h,000h,000h,03eh,063h,063h
	63 63	C		
C758	63 63 3E 00 00 00	C	DB	063h,063h,03eh,000h,000h,000h,000h,000h ; 'o' 6f
	00 00	C		
C760	00 00 00 00 00 6E	C	DB	000h,000h,000h,000h,000h,06eh,033h,033h
	33 33	C		
C768	33 33 3E 30 30 78	C	DB	033h,033h,03eh,030h,030h,078h,000h,000h ; 'p' 70
	00 00	C		
C770	00 00 00 00 00 3B	C	DB	000h,000h,000h,000h,000h,03bh,066h,066h
	66 66	C		
C778	66 66 3E 06 06 0F	C	DB	066h,066h,03eh,006h,006h,00fh,000h,000h ; 'q' 71
	00 00	C		
C780	00 00 00 00 00 6E	C	DB	000h,000h,000h,000h,000h,06eh,033h,030h
	33 30	C		
C788	30 30 78 00 00 00	C	DB	030h,030h,078h,000h,000h,000h,000h,000h ; 'r' 72
	00 00	C		
C790	00 00 00 00 00 3E	C	DB	000h,000h,000h,000h,000h,03eh,063h,038h
	63 38	C		
C798	0E 63 3E 00 00 00	C	DB	00eh,063h,03eh,000h,000h,000h,000h,000h ; 's' 73
	00 00	C		
C7A0	00 00 00 08 18 7E	C	DB	000h,000h,000h,008h,018h,07eh,018h,018h
	18 18	C		
C7A8	18 1B 0E 00 00 00	C	DB	018h,01bh,00eh,000h,000h,000h,000h,000h ; 't' 74
	00 00	C		
C7B0	00 00 00 00 00 66	C	DB	000h,000h,000h,000h,000h,066h,066h,066h
	66 66	C		
C7B8	66 66 3B 00 00 00	C	DB	066h,066h,03bh,000h,000h,000h,000h,000h ; 'u' 75
	00 00	C		
C7C0	00 00 00 00 00 63	C	DB	000h,000h,000h,000h,000h,063h,063h,063h
	63 63	C		
C7C8	36 1C 08 00 00 00	C	DB	036h,01ch,008h,000h,000h,000h,000h,000h ; 'v' 76
	00 00	C		
C7D0	00 00 00 00 00 63	C	DB	000h,000h,000h,000h,000h,063h,063h,06bh
	63 6B	C		


```

C7D8 6B 7F 36 00 00 00 C DB 06bh,07fh,036h,000h,000h,000h,000h,000h ;'w' 77
      00 00 C
C7E0 00 00 00 00 00 63 C DB 000h,000h,000h,000h,000h,063h,036h,01ch
      36 1C C
C7E8 1C 36 63 00 00 00 C DB 01ch,036h,063h,000h,000h,000h,000h,000h ;'x' 78
      00 00 C
C7F0 00 00 00 00 00 63 C DB 000h,000h,000h,000h,000h,063h,066h,066h
      66 66 C
C7F8 66 66 3E 06 66 3C C DB 066h,066h,03eh,006h,066h,03ch,000h,000h ;'y' 79
      00 00 C
C800 00 00 00 00 00 7F C DB 000h,000h,000h,000h,000h,07fh,066h,00ch
      66 0C C
C808 18 33 7F 00 00 00 C DB 018h,033h,07fh,000h,000h,000h,000h,000h ;'z' 7a
      00 00 C
C810 00 00 0E 18 18 18 C DB 000h,000h,00eh,018h,018h,018h,070h,018h
      70 18 C
C818 18 18 0E 00 00 00 C DB 018h,018h,00eh,000h,000h,000h,000h,000h ;'{' 7b
      00 00 C
C820 00 00 18 18 18 18 C DB 000h,000h,018h,018h,018h,018h,000h,018h
      00 18 C
C828 18 18 18 00 00 00 C DB 018h,018h,018h,000h,000h,000h,000h,000h ;'|' 7c
      00 00 C
C830 00 00 70 18 18 18 C DB 000h,000h,070h,018h,018h,018h,00eh,018h
      0E 18 C
C838 18 18 70 00 00 00 C DB 018h,018h,070h,000h,000h,000h,000h,000h ;'}' 7d
      00 00 C
C840 00 00 3B 6E 00 00 C DB 000h,000h,03bh,06eh,000h,000h,000h,000h
      00 00 C
C848 00 00 00 00 00 00 C DB 000h,000h,000h,000h,000h,000h,000h,000h ;'~' 7e
      00 00 C
C850 00 00 00 00 08 1C C DB 000h,000h,000h,000h,008h,01ch,036h,063h
      36 63 C
C858 63 7F 00 00 00 00 C DB 063h,07fh,000h,000h,000h,000h,000h,000h ;'' 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
C860 fonthi8 proc near
C ; SystemFont ; <hi_mediumres> (m24) 8 x 8 font table for m24
C
C860 3C 66 60 66 3C 0C C DB 03ch,066h,060h,066h,03ch,00ch,006h,03ch ; 80
      06 3C C
C868 00 66 00 66 66 66 C DB 000h,066h,000h,066h,066h,066h,03fh,000h ; 81
      3F 00 C
C870 0E 00 3C 66 7E 60 C DB 00eh,000h,03ch,066h,07eh,060h,03ch,000h ; 82
      3C 00 C
C878 7E C3 3C 06 3E 66 C DB 07eh,0c3h,03ch,006h,03eh,066h,03fh,000h ; 83
      3F 00 C
C880 66 00 3C 06 3E 66 C DB 066h,000h,03ch,006h,03eh,066h,03fh,000h ; 84
      3F 00 C
C888 70 00 3C 06 3E 66 C DB 070h,000h,03ch,006h,03eh,066h,03fh,000h ; 85
      3F 00 C

```

ROM BIOS Listing

```

C890 18 18 3C 06 3E 66 C DB 018h,018h,03ch,006h,03eh,066h,03fh,000h ; 86
      3F 00 C
C898 00 00 3C 60 60 3C C DB 000h,000h,03ch,060h,060h,03ch,006h,01ch ; 87
      06 1C C
C8A0 7E C3 3C 66 7E 60 C DB 07eh,0c3h,03ch,066h,07eh,060h,03ch,000h ; 88
      3C 00 C
C8A8 66 00 3C 66 7E 60 C DB 066h,000h,03ch,066h,07eh,060h,03ch,000h ; 89
      3C 00 C
C8B0 70 00 3C 66 7E 60 C DB 070h,000h,03ch,066h,07eh,060h,03ch,000h ; 8a
      3C 00 C
C8B8 66 00 38 18 18 18 C DB 066h,000h,038h,018h,018h,018h,03ch,000h ; 8b
      3C 00 C
C8C0 7C C6 38 18 18 18 C DB 07ch,0c6h,038h,018h,018h,018h,03ch,000h ; 8c
      3C 00 C
C8C8 70 00 38 18 18 18 C DB 070h,000h,038h,018h,018h,018h,03ch,000h ; 8d
      3C 00 C
C8D0 63 1C 36 63 7F 63 C DB 063h,01ch,036h,063h,07fh,063h,063h,000h ; 8e
      63 00 C
C8D8 18 18 00 3C 66 7E C DB 018h,018h,000h,03ch,066h,07eh,066h,000h ; 8f
      66 00 C
C8E0 0E 00 7E 30 3C 30 C DB 00eh,000h,07eh,030h,03ch,030h,07eh,000h ; 90
      7E 00 C
C8E8 00 00 7F 0C 7F CC C DB 000h,000h,07fh,00ch,07fh,0cch,07fh,000h ; 91
      7F 00 C
C8F0 1F 36 66 7F 66 66 C DB 01fh,036h,066h,07fh,066h,066h,067h,000h ; 92
      67 00 C
C8F8 3C 66 00 3C 66 66 C DB 03ch,066h,000h,03ch,066h,066h,03ch,000h ; 93
      3C 00 C
C900 00 66 00 3C 66 66 C DB 000h,066h,000h,03ch,066h,066h,03ch,000h ; 94
      3C 00 C
C908 00 70 00 3C 66 66 C DB 000h,070h,000h,03ch,066h,066h,03ch,000h ; 95
      3C 00 C
C910 3C 66 00 66 66 66 C DB 03ch,066h,000h,066h,066h,066h,03fh,000h ; 96
      3F 00 C
C918 00 70 00 66 66 66 C DB 000h,070h,000h,066h,066h,066h,03fh,000h ; 97
      3F 00 C
C920 00 66 00 66 66 3E C DB 000h,066h,000h,066h,066h,03eh,006h,07ch ; 98
      06 7C C
C928 C3 18 3C 66 66 3C C DB 0c3h,018h,03ch,066h,066h,03ch,018h,000h ; 99
      18 00 C
C930 66 00 66 66 66 66 C DB 066h,000h,066h,066h,066h,066h,03ch,000h ; 9a
      3C 00 C
      C ;#ifdef NORDIC
      C ; DB 000h,000h,000h,03ch,06eh,076h,03ch,000h ; 9b
      C ; DB 01ch,036h,032h,078h,030h,073h,07eh,000h ; 9c
      C ; DB 07ch,0c6h,0ceh,0deh,0f6h,0e6h,07ch,000h ; 9d
      C ; DB 0f0h,060h,066h,060h,062h,066h,0feh,000h ; 9e
      C ; DB 070h,030h,030h,036h,030h,030h,078h,000h ; 9f
      C ;#else NORDIC
      C ;#ifdef PORTUGAL
      C ; DB 007h,000h,01ch,036h,063h,07fh,063h,000h ; 9b
      C ; DB 01ch,036h,032h,078h,030h,073h,07eh,000h ; 9c
      C ; DB 070h,000h,01ch,036h,063h,07fh,063h,000h ; 9d
      C ; DB 018h,024h,07eh,030h,03ch,030h,03eh,000h ; 9e
      C ; DB 03eh,063h,01ch,036h,063h,036h,01ch,000h ; 9f

```

```

C ;#else PORTUGAL
C ;      DB 018h,018h,07eh,0c0h,0c0h,07eh,018h,018h ; 9b
C ;      DB 01ch,036h,032h,078h,030h,073h,07eh,000h ; 9c
C ;      DB 066h,066h,03ch,07eh,018h,07eh,018h,018h ; 9d
C ;      DB 0f8h,0cch,0cch,0fah,0c6h,0cfh,0c6h,0c7h ; 9e
C ;      DB 00eh,01bh,018h,03ch,018h,018h,0d8h,070h ; 9f
C ;#endif PORTUGAL
C ;#endif NORDIC
C ;      DB 00eh,000h,03ch,006h,03eh,066h,03fh,000h ; a0
C ;      DB 01ch,000h,038h,018h,018h,018h,03ch,000h ; a1
C ;      DB 000h,00eh,000h,03ch,066h,066h,03ch,000h ; a2
C ;      DB 000h,00eh,000h,066h,066h,066h,03fh,000h ; a3
C ;      DB 000h,07ch,000h,07ch,066h,066h,066h,000h ; a4
C ;      DB 07eh,000h,066h,076h,07eh,06eh,066h,000h ; a5
C ;#ifdef NORDIC
C ;      DB 000h,07eh,000h,03ch,066h,066h,03ch,000h ; a6
C ;      DB 07eh,000h,01ch,036h,063h,036h,01ch,000h ; a7
C ;      DB 018h,000h,018h,030h,060h,066h,03ch,000h ; a8
C ;      DB 07eh,000h,03ch,006h,03eh,066h,03fh,000h ; a9
C ;      DB 07eh,000h,03ch,066h,07eh,066h,066h,000h ; aa
C ;      DB 010h,038h,06ch,06ch,038h,034h,058h,000h ; ab
C ;      DB 0c0h,0c0h,000h,0f8h,0cch,0cch,0cch,000h ; ac
C ;      DB 018h,018h,000h,018h,018h,018h,018h,000h ; ad
C ;      DB 07ch,018h,030h,098h,070h,000h,000h,000h ; ae
C ;      DB 000h,0c6h,07ch,0c6h,0c6h,07ch,0c6h,000h ; af
C ;#else NORDIC
C ;#ifdef PORTUGAL
C ;      DB 000h,07eh,000h,03ch,066h,066h,03ch,000h ; a6
C ;      DB 07eh,000h,01ch,036h,063h,036h,01ch,000h ; a7
C ;      DB 018h,000h,018h,030h,060h,066h,03ch,000h ; a8
C ;      DB 07eh,000h,03ch,006h,03eh,066h,03fh,000h ; a9
C ;      DB 07eh,000h,03ch,066h,07eh,066h,066h,000h ; aa
C ;      DB 007h,000h,063h,063h,063h,063h,03eh,000h ; ab
C ;      DB 00eh,000h,03ch,018h,018h,018h,03ch,000h ; ac
C ;      DB 018h,018h,000h,018h,018h,018h,018h,000h ; ad
C ;      DB 07ch,018h,030h,098h,070h,000h,000h,000h ; ae
C ;      DB 007h,000h,01ch,036h,063h,036h,01ch,000h ; af
C ;#else PORTUGAL
C ;      DB 03ch,06ch,06ch,03eh,000h,07eh,000h,000h ; a6
C ;      DB 038h,06ch,06ch,038h,000h,07ch,000h,000h ; a7
C ;      DB 018h,000h,018h,030h,060h,066h,03ch,000h ; a8
C ;      DB 000h,000h,000h,07eh,060h,060h,000h,000h ; a9
C ;      DB 000h,000h,000h,07eh,006h,006h,000h,000h ; aa
C ;      DB 0c3h,0c6h,0cch,0deh,033h,066h,0cch,00fh ; ab
C ;      DB 0c3h,0c6h,0cch,0dbh,037h,06fh,0cfh,003h ; ac
C ;      DB 018h,018h,000h,018h,018h,018h,018h,000h ; ad
C ;      DB 000h,033h,066h,0cch,066h,033h,000h,000h ; ae
C ;      DB 000h,0cch,066h,033h,066h,0cch,000h,000h ; af
C ;#endif PORTUGAL
C ;#endif NORDIC
C938 22 88 22 88 22 88 C      DB 022h,088h,022h,088h,022h,088h,022h,088h ; b0
      22 88 C
C940 55 AA 55 AA 55 AA C      DB 055h,0aah,055h,0aah,055h,0aah,055h,0aah ; b1
      55 AA C
C948 DB 77 DB EE DB 77 C      DB 0dbh,077h,0dbh,0eeh,0dbh,077h,0dbh,0eeh ; b2

```

ROM BIOS Listing

	DB EE	C			
C950	18 18 18 18 18 18 18 18	C C	DB	018h,018h,018h,018h,018h,018h,018h,018h	; b3
C958	18 18 18 18 F8 18 18 18	C C	DB	018h,018h,018h,018h,0f8h,018h,018h,018h	; b4
C960	18 18 F8 18 F8 18 18 18	C C	DB	018h,018h,0f8h,018h,0f8h,018h,018h,018h	; b5
C968	36 36 36 36 F6 36 36 36	C C	DB	036h,036h,036h,036h,0f6h,036h,036h,036h	; b6
C970	00 00 00 00 FE 36 36 36	C C	DB	000h,000h,000h,000h,0feh,036h,036h,036h	; b7
C978	00 00 F8 18 F8 18 18 18	C C	DB	000h,000h,0f8h,018h,0f8h,018h,018h,018h	; b8
C980	36 36 F6 06 F6 36 36 36	C C	DB	036h,036h,0f6h,006h,0f6h,036h,036h,036h	; b9
C988	36 36 36 36 36 36 36 36	C C	DB	036h,036h,036h,036h,036h,036h,036h,036h	; ba
C990	00 00 FE 06 F6 36 36 36	C C	DB	000h,000h,0feh,006h,0f6h,036h,036h,036h	; bb
C998	36 36 F6 06 FE 00 00 00	C C	DB	036h,036h,0f6h,006h,0feh,000h,000h,000h	; bc
C9A0	36 36 36 36 FE 00 00 00	C C	DB	036h,036h,036h,036h,0feh,000h,000h,000h	; bd
C9A8	18 18 F8 18 F8 00 00 00	C C	DB	018h,018h,0f8h,018h,0f8h,000h,000h,000h	; be
C9B0	00 00 00 00 F8 18 18 18	C C	DB	000h,000h,000h,000h,0f8h,018h,018h,018h	; bf
C9B8	18 18 18 18 1F 00 00 00	C C	DB	018h,018h,018h,018h,01fh,000h,000h,000h	; c0
C9C0	18 18 18 18 FF 00 00 00	C C	DB	018h,018h,018h,018h,0ffh,000h,000h,000h	; c1
C9C8	00 00 00 00 FF 18 18 18	C C	DB	000h,000h,000h,000h,0ffh,018h,018h,018h	; c2
C9D0	18 18 18 18 1F 18 18 18	C C	DB	018h,018h,018h,018h,01fh,018h,018h,018h	; c3
C9D8	00 00 00 00 FF 00 00 00	C C	DB	000h,000h,000h,000h,0ffh,000h,000h,000h	; c4
C9E0	18 18 18 18 FF 18 18 18	C C	DB	018h,018h,018h,018h,0ffh,018h,018h,018h	; c5
C9E8	18 18 1F 18 1F 18 18 18	C C	DB	018h,018h,01fh,018h,01fh,018h,018h,018h	; c6
C9F0	36 36 36 36 37 36 36 36	C C	DB	036h,036h,036h,036h,037h,036h,036h,036h	; c7
C9F8	36 36 37 30 3F 00 00 00	C C	DB	036h,036h,037h,030h,03fh,000h,000h,000h	; c8
CA00	00 00 3F 30 37 36 36 36	C C	DB	000h,000h,03fh,030h,037h,036h,036h,036h	; c9
CA08	36 36 F7 00 FF 00 00 00	C C	DB	036h,036h,0f7h,000h,0ffh,000h,000h,000h	; ca
CA10	00 00 FF 00 F7 36 36 36	C C	DB	000h,000h,0ffh,000h,0f7h,036h,036h,036h	; cb
CA18	36 36 37 30 37 36 36 36	C C	DB	036h,036h,037h,030h,037h,036h,036h,036h	; cc
CA20	00 00 FF 00 FF 00 00 00	C C	DB	000h,000h,0ffh,000h,0ffh,000h,000h,000h	; cd

CA28	36 36 F7 00 F7 36	C	DB	036h,036h,0f7h,000h,0f7h,036h,036h,036h	:	ce
	36 36	C				
CA30	18 18 FF 00 FF 00	C	DB	018h,018h,0ffh,000h,0ffh,000h,000h,000h	:	cf
	00 00	C				
CA38	36 36 36 36 FF 00	C	DB	036h,036h,036h,036h,0ffh,000h,000h,000h	:	d0
	00 00	C				
CA40	00 00 FF 00 FF 18	C	DB	000h,000h,0ffh,000h,0ffh,018h,018h,018h	:	d1
	18 18	C				
CA48	00 00 00 00 FF 36	C	DB	000h,000h,000h,000h,0ffh,036h,036h,036h	:	d2
	36 36	C				
CA50	36 36 36 36 3F 00	C	DB	036h,036h,036h,036h,03fh,000h,000h,000h	:	d3
	00 00	C				
CA58	18 18 1F 18 1F 00	C	DB	018h,018h,01fh,018h,01fh,000h,000h,000h	:	d4
	00 00	C				
CA60	00 00 1F 18 1F 18	C	DB	000h,000h,01fh,018h,01fh,018h,018h,018h	:	d5
	18 18	C				
CA68	00 00 00 00 3F 36	C	DB	000h,000h,000h,000h,03fh,036h,036h,036h	:	d6
	36 36	C				
CA70	36 36 36 36 FF 36	C	DB	036h,036h,036h,036h,0ffh,036h,036h,036h	:	d7
	36 36	C				
CA78	18 18 FF 18 FF 18	C	DB	018h,018h,0ffh,018h,0ffh,018h,018h,018h	:	d8
	18 18	C				
CA80	18 18 18 18 F8 00	C	DB	018h,018h,018h,018h,0f8h,000h,000h,000h	:	d9
	00 00	C				
CA88	00 00 00 00 1F 18	C	DB	000h,000h,000h,000h,01fh,018h,018h,018h	:	da
	18 18	C				
CA90	FF FF FF FF FF FF	C	DB	0ffh,0ffh,0ffh,0ffh,0ffh,0ffh,0ffh,0ffh	:	db
	FF FF	C				
CA98	00 00 00 00 FF FF	C	DB	000h,000h,000h,000h,0ffh,0ffh,0ffh,0ffh	:	dc
	FF FF	C				
CAA0	F0 F0 F0 F0 F0 F0	C	DB	0f0h,0f0h,0f0h,0f0h,0f0h,0f0h,0f0h,0f0h	:	dd
	F0 F0	C				
CAA8	0F 0F 0F 0F 0F 0F	C	DB	00fh,00fh,00fh,00fh,00fh,00fh,00fh,00fh	:	de
	0F 0F	C				
CAB0	FF FF FF FF 00 00	C	DB	0ffh,0ffh,0ffh,0ffh,000h,000h,000h,000h	:	df
	00 00	C				
CAB8	00 00 3B 6E 64 6E	C	DB	000h,000h,03bh,06eh,064h,06eh,03bh,000h	:	e0
	3B 00	C				
CAC0	00 3C 66 7C 66 7C	C	DB	000h,03ch,066h,07ch,066h,07ch,060h,060h	:	e1
	60 60	C				
CAC8	00 7E 66 60 60 60	C	DB	000h,07eh,066h,060h,060h,060h,060h,000h	:	e2
	60 00	C				
CAD0	00 7F 36 36 36 36	C	DB	000h,07fh,036h,036h,036h,036h,036h,000h	:	e3
	36 00	C				
CAD8	7E 66 30 18 30 66	C	DB	07eh,066h,030h,018h,030h,066h,07eh,000h	:	e4
	7E 00	C				
CAE0	00 00 3F 6C 6C 6C	C	DB	000h,000h,03fh,06ch,06ch,06ch,038h,000h	:	e5
	38 00	C				
CAE8	00 33 33 33 33 3E	C	DB	000h,033h,033h,033h,033h,03eh,030h,060h	:	e6
	30 60	C				
CAF0	00 3B 6E 0C 0C 0C	C	DB	000h,03bh,06eh,00ch,00ch,00ch,00ch,000h	:	e7
	0C 00	C				
CAF8	7E 18 3C 66 66 3C	C	DB	07eh,018h,03ch,066h,066h,03ch,018h,07eh	:	e8
	18 7E	C				
CB00	1C 36 63 7F 63 36	C	DB	01ch,036h,063h,07fh,063h,036h,01ch,000h	:	e9

ROM BIOS Listing

```

1C 00 C
CB08 1C 36 63 63 36 36 C DB 01ch,036h,063h,063h,036h,036h,077h,000h ; ea
77 00 C
CB10 0E 18 0C 3E 66 66 C DB 00eh,018h,00ch,03eh,066h,066h,03ch,000h ; eb
3C 00 C
CB18 00 00 7E DB DB 7E C DB 000h,000h,07eh,0dbh,0dbh,07eh,000h,000h ; ec
00 00 C
CB20 06 0C 7E DB DB 7E C DB 006h,00ch,07eh,0dbh,0dbh,07eh,060h,0c0h ; ed
60 C0 C
CB28 1C 60 C0 FC C0 60 C DB 01ch,060h,0c0h,0fch,0c0h,060h,01ch,000h ; ee
1C 00 C
CB30 3C 66 66 66 66 66 C DB 03ch,066h,066h,066h,066h,066h,066h,000h ; ef
66 00 C
CB38 00 7E 00 7E 00 7E C DB 000h,07eh,000h,07eh,000h,07eh,000h,000h ; f0
00 00 C
CB40 18 18 7E 18 18 00 C DB 018h,018h,07eh,018h,018h,000h,07eh,000h ; f1
7E 00 C
CB48 30 18 0C 18 30 00 C DB 030h,018h,00ch,018h,030h,000h,07eh,000h ; f2
7E 00 C
CB50 0C 18 30 18 0C 00 C DB 00ch,018h,030h,018h,00ch,000h,07eh,000h ; f3
7E 00 C
CB58 0E 1B 1B 18 18 18 C DB 00eh,01bh,01bh,018h,018h,018h,018h,018h ; f4
18 18 C
CB60 18 18 18 18 18 D8 C DB 018h,018h,018h,018h,018h,0d8h,0d8h,070h ; f5
D8 70 C
CB68 18 18 00 7E 00 18 C DB 018h,018h,000h,07eh,000h,018h,018h,000h ; f6
18 00 C
CB70 00 76 DC 00 76 DC C DB 000h,076h,0dch,000h,076h,0dch,000h,000h ; f7
00 00 C
CB78 38 6C 6C 38 00 00 C DB 038h,06ch,06ch,038h,000h,000h,000h,000h ; f8
00 00 C
CB80 00 00 00 18 18 00 C DB 000h,000h,000h,018h,018h,000h,000h,000h ; f9
00 00 C
CB88 00 00 00 00 18 00 C DB 000h,000h,000h,000h,018h,000h,000h,000h ; fa
00 00 C
CB90 0F 0C 0C 0C EC 6C C DB 00fh,00ch,00ch,00ch,0ech,06ch,03ch,01ch ; fb
3C 1C C
CB98 78 6C 6C 6C 6C 00 C DB 078h,06ch,06ch,06ch,06ch,000h,000h,000h ; fc
00 00 C
CBA0 70 18 30 60 78 00 C DB 070h,018h,030h,060h,078h,000h,000h,000h ; fd
00 00 C
CBA8 00 00 3C 3C 3C 3C C DB 000h,000h,03ch,03ch,03ch,03ch,000h,000h ; fe
00 00 C
CBB0 00 00 00 00 00 00 C DB 000h,000h,000h,000h,000h,000h,000h,000h ; ff
00 00 C
C ; End of font matrix
C
CBB8 C fonthi8 endp

CBB8 code ends
C include kbdata.asm
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
CBB8 C code segment public 'ROM'
C      assume cs:code, ds:nothing, es:nothing, ss:nothing
C
CBB8 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 kbaln 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 kbnu1 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 kdb10 equ 0D8h ; kb_double_zero      (max_case)
C
C ;-----
C ;      7 CapLk Bytes
C ;-----
C
CBB8 C kb_cap_flags label byte
C
CBB8 00 C db 00000000b ; scancode 00 (00h) - 07 (07h) ESC to '6'
CBB9 00 C db 00000000b ; scancode 08 (08h) - 15 (0Fh) '7' to HT
CBBA FF C db 11111111b ; scancode 16 (10h) - 23 (17h) 'q' to 'i'
CBBB C3 C db 11000011b ; scancode 24 (18h) - 31 (1Fh) 'o' & 'p' to 'a' & 's'
CBBC FE C db 11111110b ; scancode 32 (20h) - 39 (27h) 'd' to 'l' & ';'
CBBD 0F C db 00001111b ; scancode 40 (28h) - 47 (2Fh) ':' to '~' to 'z' to 'v'
CBBE E0 C db 11100000b ; scancode 48 (30h) - 55 (37h) 'b' to 'm' to ',' to '*'

```

```

C
C ;-----
C ;      Alphabetic (Migratory)          | AT&T  |Other |
C ;                                     | KB    | KBs   |
C ;-----
C
CBBF      kb_data_table  label  byte
C
CBBF 1B1B  C dw (1Bh) * 100h + (1Bh) ; 01 01h  ESC  ESC  (BASE)
CBC1 1B1B  C dw (1Bh) * 100h + (1Bh) ;          ESC  ESC  (SHIFT)
CBC3 1B1B  C dw (1Bh) * 100h + (1Bh) ;          ESC  ESC  (CTL)
CBC5 CDCD  C dw kNONE * 100h + kNONE ;          None None  (ALT)
CBC7 3131  C dw (31h) * 100h + (31h) ; 02 02h  1    1
CBC9 2121  C dw (21h) * 100h + (21h) ;          !    !
CBCB CDCD  C dw kNONE * 100h + kNONE ;          None None
CBCD 7800  C dw 7800h ;          X120
CBCF 3232  C dw (32h) * 100h + (32h) ; 03 03h  2    2
CBD1 2240  C dw (22h) * 100h + (40h) ;          "    @
CBD3 CDCD  C dw kNONE * 100h + kbnu1 ;          None NUL=X03(^@)
CBD5 7900  C dw 7900h ;          X121
CBD7 3333  C dw (33h) * 100h + (33h) ; 04 04h  3    3
CBD9 2323  C dw (23h) * 100h + (23h) ;          #    #
CBDB CDCD  C dw kNONE * 100h + kNONE ;          None None
CBDD 7A00  C dw 7A00h ;          X122
CBDF 3434  C dw (34h) * 100h + (34h) ; 05 05h  4    4
CBE1 2424  C dw (24h) * 100h + (24h) ;          $    $
CBE3 CDCD  C dw kNONE * 100h + kNONE ;          None None
CBE5 7B00  C dw 7B00h ;          X123
CBE7 3535  C dw (35h) * 100h + (35h) ; 06 06h  5    5
CBE9 2525  C dw (25h) * 100h + (25h) ;          %    %
CBEB CDCD  C dw kNONE * 100h + kNONE ;          None None
CBED 7C00  C dw 7C00h ;          X124
CBEF 3636  C dw (36h) * 100h + (36h) ; 07 07h  6    6
CBF1 265E  C dw (26h) * 100h + (5Eh) ;          &    ^
CBF3 CD1E  C dw kNONE * 100h + (1Eh) ;          None RS (^^)
CBF5 7D00  C dw 7D00h ;          X125
CBF7 3737  C dw (37h) * 100h + (37h) ; 08 08h  7    7
CBF9 2726  C dw (27h) * 100h + (26h) ;          '    &
CBFB CDCD  C dw kNONE * 100h + kNONE ;          None None
CBFD 7E00  C dw 7E00h ;          X126
CBFF 3838  C dw (38h) * 100h + (38h) ; 09 09h  8    8
CC01 282A  C dw (28h) * 100h + (2Ah) ;          (    *
CC03 CDCD  C dw kNONE * 100h + kNONE ;          None None
CC05 7F00  C dw 7F00h ;          X127
CC07 3939  C dw (39h) * 100h + (39h) ; 10 0Ah  9    9
CC09 2928  C dw (29h) * 100h + (28h) ;          )    (
CC0B CDCD  C dw kNONE * 100h + kNONE ;          None None
CC0D 8000  C dw 8000h ;          X128
CC0F 3030  C dw (30h) * 100h + (30h) ; 11 0Bh  0    0
CC11 5F29  C dw (5Fh) * 100h + (29h) ;          _    )
CC13 1FCD  C dw (1Fh) * 100h + kNONE ;          US (^_) None
CC15 8100  C dw 8100h ;          X129
CC17 2D2D  C dw (2Dh) * 100h + (2Dh) ; 12 0Ch  -    -
CC19 3D5F  C dw (3Dh) * 100h + (5Fh) ;          =
CC1B CD1F  C dw kNONE * 100h + (1Fh) ;          None US (^_)

```


CC1D	8200	C	dw	8200h	;		X130
CC1F	5E3D	C	dw	(5Eh) * 100h + (3Dh)	;	13 0Dh	^ =
CC21	7E2B	C	dw	(7Eh) * 100h + (2Bh)	;		~ +
CC23	1ECD	C	dw	(1Eh) * 100h + kNONE	;		RS (^) None
CC25	8300	C	dw	8300h	;		X131
CC27	0808	C	dw	(08h) * 100h + (08h)	;	14 0Eh	BS BS
CC29	0808	C	dw	(08h) * 100h + (08h)	;		BS BS
CC2B	7F7F	C	dw	(7Fh) * 100h + (7Fh)	;		DEL DEL
CC2D	CDCD	C	dw	kNONE * 100h + kNONE	;		None None
CC2F	0909	C	dw	(09h) * 100h + (09h)	;	15 0Fh	HT HT
CC31	0F00	C	dw	0F00h	;		RHT=X15
CC33	CDCD	C	dw	kNONE * 100h + kNONE	;		None None
CC35	CDCD	C	dw	kNONE * 100h + kNONE	;		None None
CC37	7171	C	dw	(71h) * 100h + (71h)	;	16 10h	q q
CC39	5151	C	dw	(51h) * 100h + (51h)	;		Q Q
CC3B	1111	C	dw	(11h) * 100h + (11h)	;		DC1(^Q) DC1(^Q)
CC3D	1000	C	dw	1000h	;		X16
CC3F	7777	C	dw	(77h) * 100h + (77h)	;	17 11h	w w
CC41	5757	C	dw	(57h) * 100h + (57h)	;		W W
CC43	1717	C	dw	(17h) * 100h + (17h)	;		ETB(^W) ETB(^W)
CC45	1100	C	dw	1100h	;		X17
CC47	6565	C	dw	(65h) * 100h + (65h)	;	18 12h	e e
CC49	4545	C	dw	(45h) * 100h + (45h)	;		E E
CC4B	0505	C	dw	(05h) * 100h + (05h)	;		ENQ(^E) ENQ(^E)
CC4D	1200	C	dw	1200h	;		X18
CC4F	7272	C	dw	(72h) * 100h + (72h)	;	19 13h	r r
CC51	5252	C	dw	(52h) * 100h + (52h)	;		R R
CC53	1212	C	dw	(12h) * 100h + (12h)	;		DC2(^R) DC2(^R)
CC55	1300	C	dw	1300h	;		X19
CC57	7474	C	dw	(74h) * 100h + (74h)	;	20 14h	t t
CC59	5454	C	dw	(54h) * 100h + (54h)	;		T T
CC5B	1414	C	dw	(14h) * 100h + (14h)	;		DC4(^T) DC4(^T)
CC5D	1400	C	dw	1400h	;		X20
CC5F	7979	C	dw	(79h) * 100h + (79h)	;	21 15h	y y
CC61	5959	C	dw	(59h) * 100h + (59h)	;		Y Y
CC63	1919	C	dw	(19h) * 100h + (19h)	;		EM (^Y) EM (^Y)
CC65	1500	C	dw	1500h	;		X21
CC67	7575	C	dw	(75h) * 100h + (75h)	;	22 16h	u u
CC69	5555	C	dw	(55h) * 100h + (55h)	;		U U
CC6B	1515	C	dw	(15h) * 100h + (15h)	;		NAK(^U) NAK(^U)
CC6D	1600	C	dw	1600h	;		X22
CC6F	6969	C	dw	(69h) * 100h + (69h)	;	23 17h	i i
CC71	4949	C	dw	(49h) * 100h + (49h)	;		I I
CC73	0909	C	dw	(09h) * 100h + (09h)	;		HT (^I) HT (^I)
CC75	1700	C	dw	1700h	;		X23
CC77	6F6F	C	dw	(6Fh) * 100h + (6Fh)	;	24 18h	o o
CC79	4F4F	C	dw	(4Fh) * 100h + (4Fh)	;		O O
CC7B	0F0F	C	dw	(0Fh) * 100h + (0Fh)	;		SI (^O) SI (^O)
CC7D	1800	C	dw	1800h	;		X24
CC7F	7070	C	dw	(70h) * 100h + (70h)	;	25 19h	p p
CC81	5050	C	dw	(50h) * 100h + (50h)	;		P P
CC83	1010	C	dw	(10h) * 100h + (10h)	;		DLE(^P) DLE(^P)
CC85	1900	C	dw	1900h	;		X25
CC87	405B	C	dw	(40h) * 100h + (5Bh)	;	26 1Ah	@ [
CC89	607B	C	dw	(60h) * 100h + (7Bh)	;		· {

ROM BIOS Listing

CC8B	CC1B	C	dw	kbnu1 * 100h + (1Bh)	;	NUL=X03(^@)	ESC(^[)
CC8D	CDCD	C	dw	kNONE * 100h + kNONE	;	None	None
CC8F	5B5D	C	dw	(5Bh) * 100h + (5Dh)	;	27	1Bh []
CC91	7B7D	C	dw	(7Bh) * 100h + (7Dh)	;	{ }	
CC93	1B1D	C	dw	(1Bh) * 100h + (1Dh)	;	ESC(^[)	GS (^[)
CC95	CDCD	C	dw	kNONE * 100h + kNONE	;	None	None
CC97	0D0D	C	dw	(0Dh) * 100h + (0Dh)	;	28	1Ch CR CR
CC99	0D0D	C	dw	(0Dh) * 100h + (0Dh)	;	CR	CR
CC9B	0A0A	C	dw	(0Ah) * 100h + (0Ah)	;	LF	LF
CC9D	CDCD	C	dw	kNONE * 100h + kNONE	;	None	None
CC9F	C5C5	C	dw	kbctl * 100h + kbctl	;	29	1Dh Ctrl Ctrl
CCA1	C5C5	C	dw	kbctl * 100h + kbctl	;	Ctrl	Ctrl
CCA3	C5C5	C	dw	kbctl * 100h + kbctl	;	Ctrl	Ctrl
CCA5	C5C5	C	dw	kbctl * 100h + kbctl	;	Ctrl	Ctrl
CCA7	6161	C	dw	(61h) * 100h + (61h)	;	30	1Eh a a
CCA9	4141	C	dw	(41h) * 100h + (41h)	;	A	A
CCAB	0101	C	dw	(01h) * 100h + (01h)	;	SOH(^A)	SOH(^A)
CCAD	1E00	C	dw	1E00h	;	X30	
CCAF	7373	C	dw	(73h) * 100h + (73h)	;	31	1Fh s s
CCB1	5353	C	dw	(53h) * 100h + (53h)	;	S	S
CCB3	1313	C	dw	(13h) * 100h + (13h)	;	DC3(^S)	DC3(^S)
CCB5	1F00	C	dw	1F00h	;	X31	
CCB7	6464	C	dw	(64h) * 100h + (64h)	;	32	20h d d
CCB9	4444	C	dw	(44h) * 100h + (44h)	;	D	D
CCBB	0404	C	dw	(04h) * 100h + (04h)	;	EOT(^D)	EOT(^D)
CCBD	2000	C	dw	2000h	;	X32	
CCBF	6666	C	dw	(66h) * 100h + (66h)	;	33	21h f f
CCC1	4646	C	dw	(46h) * 100h + (46h)	;	F	F
CCC3	0606	C	dw	(06h) * 100h + (06h)	;	ACK(^F)	ACK(^F)
CCC5	2100	C	dw	2100h	;	X33	
CCC7	6767	C	dw	(67h) * 100h + (67h)	;	34	22h g g
CCC9	4747	C	dw	(47h) * 100h + (47h)	;	G	G
CCCB	0707	C	dw	(07h) * 100h + (07h)	;	BEL(^G)	BEL(^G)
CCCD	2200	C	dw	2200h	;	X34	
CCCF	6868	C	dw	(68h) * 100h + (68h)	;	35	23h h h
CCD1	4848	C	dw	(48h) * 100h + (48h)	;	H	H
CCD3	0808	C	dw	(08h) * 100h + (08h)	;	BS (^H)	BS (^H)
CCD5	2300	C	dw	2300h	;	X35	
CCD7	6A6A	C	dw	(6Ah) * 100h + (6Ah)	;	36	24h j j
CCD9	4A4A	C	dw	(4Ah) * 100h + (4Ah)	;	J	J
CCDB	0A0A	C	dw	(0Ah) * 100h + (0Ah)	;	LF (^J)	LF (^J)
CCDD	2400	C	dw	2400h	;	X36	
CCDF	6B6B	C	dw	(6Bh) * 100h + (6Bh)	;	37	25h k k
CCE1	4B4B	C	dw	(4Bh) * 100h + (4Bh)	;	K	K
CCE3	0B0B	C	dw	(0Bh) * 100h + (0Bh)	;	VT (^K)	VT (^K)
CCE5	2500	C	dw	2500h	;	X37	
CCE7	6C6C	C	dw	(6Ch) * 100h + (6Ch)	;	38	26h l l
CCE9	4C4C	C	dw	(4Ch) * 100h + (4Ch)	;	L	L
CCEB	0C0C	C	dw	(0Ch) * 100h + (0Ch)	;	FF (^L)	FF (^L)
CCEd	2600	C	dw	2600h	;	X38	
CCEF	3B3B	C	dw	(3Bh) * 100h + (3Bh)	;	39	27h ; ;
CCF1	2B3A	C	dw	(2Bh) * 100h + (3Ah)	;	+	:
CCF3	CDCD	C	dw	kNONE * 100h + kNONE	;	None	None
CCF5	CDCD	C	dw	kNONE * 100h + kNONE	;	None	None
CCF7	3A27	C	dw	(3Ah) * 100h + (27h)	;	40	28h : '

CCF9	2A22	C	dw	(2Ah) * 100h + (22h)	;	*	"
CCFB	CDCD	C	dw	kNONE * 100h + kNONE	;	None	None
CCFD	CDCD	C	dw	kNONE * 100h + kNONE	;	None	None
CCFF	5D60	C	dw	(5Dh) * 100h + (60h)	;	41 29h] ' ~
CD01	7D7E	C	dw	(7Dh) * 100h + (7Eh)	;	}	}
CD03	1DCD	C	dw	(1Dh) * 100h + kNONE	;	GS (^)]	None
CD05	CDCD	C	dw	kNONE * 100h + kNONE	;	None	None
CD07	C6C6	C	dw	kb1sh * 100h + kb1sh	;	42 2Ah	LShft LShft
CD09	C6C6	C	dw	kb1sh * 100h + kb1sh	;		LShft LShft
CD0B	C6C6	C	dw	kb1sh * 100h + kb1sh	;		LShft LShft
CD0D	C6C6	C	dw	kb1sh * 100h + kb1sh	;		LShft LShft
CD0F	5C5C	C	dw	(5Ch) * 100h + (5Ch)	;	43 2Bh	\ \
CD11	7C7C	C	dw	(7Ch) * 100h + (7Ch)	;		
CD13	1C1C	C	dw	(1Ch) * 100h + (1Ch)	;	FS (^)	FS (^)
CD15	CDCD	C	dw	kNONE * 100h + kNONE	;	None	None
CD17	7A7A	C	dw	(7Ah) * 100h + (7Ah)	;	44 2Ch	z z
CD19	5A5A	C	dw	(5Ah) * 100h + (5Ah)	;	Z Z	Z Z
CD1B	1A1A	C	dw	(1Ah) * 100h + (1Ah)	;	SUB(^Z)	SUB(^Z)
CD1D	2C00	C	dw	2C00h	;		X44
CD1F	7878	C	dw	(78h) * 100h + (78h)	;	45 2Dh	x x
CD21	5858	C	dw	(58h) * 100h + (58h)	;	X X	X X
CD23	1818	C	dw	(18h) * 100h + (18h)	;	CAN(^X)	CAN(^X)
CD25	2D00	C	dw	2D00h	;		X45
CD27	6363	C	dw	(63h) * 100h + (63h)	;	46 2Eh	c c
CD29	4343	C	dw	(43h) * 100h + (43h)	;	C C	C C
CD2B	0303	C	dw	(03h) * 100h + (03h)	;	ETX(^C)	ETX(^C)
CD2D	2E00	C	dw	2E00h	;		X46
CD2F	7676	C	dw	(76h) * 100h + (76h)	;	47 2Fh	v v
CD31	5656	C	dw	(56h) * 100h + (56h)	;	V V	V V
CD33	1616	C	dw	(16h) * 100h + (16h)	;	SYN(^V)	SYN(^V)
CD35	2F00	C	dw	2F00h	;		X47
CD37	6262	C	dw	(62h) * 100h + (62h)	;	48 30h	b b
CD39	4242	C	dw	(42h) * 100h + (42h)	;	B B	B B
CD3B	0202	C	dw	(02h) * 100h + (02h)	;	STX(^B)	STX(^B)
CD3D	3000	C	dw	3000h	;		X48
CD3F	6E6E	C	dw	(6Eh) * 100h + (6Eh)	;	49 31h	n n
CD41	4E4E	C	dw	(4Eh) * 100h + (4Eh)	;	N N	N N
CD43	0E0E	C	dw	(0Eh) * 100h + (0Eh)	;	SO (^N)	SO (^N)
CD45	3100	C	dw	3100h	;		X49
CD47	6D6D	C	dw	(6Dh) * 100h + (6Dh)	;	50 32h	m m
CD49	4D4D	C	dw	(4Dh) * 100h + (4Dh)	;	M M	M M
CD4B	0D0D	C	dw	(0Dh) * 100h + (0Dh)	;	CR (^M)	CR (^M)
CD4D	3200	C	dw	3200h	;		X50
CD4F	2C2C	C	dw	(2Ch) * 100h + (2Ch)	;	51 33h	, ,
CD51	3C3C	C	dw	(3Ch) * 100h + (3Ch)	;	< <	< <
CD53	CDCD	C	dw	kNONE * 100h + kNONE	;	None	None
CD55	CDCD	C	dw	kNONE * 100h + kNONE	;	None	None
CD57	2E2E	C	dw	(2Eh) * 100h + (2Eh)	;	52 34h	. .
CD59	3E3E	C	dw	(3Eh) * 100h + (3Eh)	;	> >	> >
CD5B	CDCD	C	dw	kNONE * 100h + kNONE	;	None	None
CD5D	CDCD	C	dw	kNONE * 100h + kNONE	;	None	None
CD5F	2F2F	C	dw	(2Fh) * 100h + (2Fh)	;	53 35h	/ /
CD61	3F3F	C	dw	(3Fh) * 100h + (3Fh)	;	? ?	? ?
CD63	CDCD	C	dw	kNONE * 100h + kNONE	;	None	None
CD65	CDCD	C	dw	kNONE * 100h + kNONE	;	None	None

		C	-----			
		C	Alphabetic (Non-Migratory)		AT&T	Other
		C			KB	KBs
		C	-----			
		C				
CD67	C7C7	C	dw	kbrsh * 100h + kbrsh	; 54 36h	RShft RShft
CD69	C7C7	C	dw	kbrsh * 100h + kbrsh	;	RShft RShft
CD6B	C7C7	C	dw	kbrsh * 100h + kbrsh	;	RShft RShft
CD6D	C7C7	C	dw	kbrsh * 100h + kbrsh	;	RShft RShft
CD6F	2A2A	C	dw	(2Ah) * 100h + (2Ah)	; 55 37h	* *
CD71	CBCB	C	dw	kbprt * 100h + kbprt	;	PrtSc PrtSc
CD73	7200	C	dw	7200h	;	X114
CD75	CDCD	C	dw	kNONE * 100h + kNONE	;	None None
CD77	C4C4	C	dw	kbalt * 100h + kbalt	; 56 38h	ALT ALT
CD79	C4C4	C	dw	kbalt * 100h + kbalt	;	ALT ALT
CD7B	C4C4	C	dw	kbalt * 100h + kbalt	;	ALT ALT
CD7D	C4C4	C	dw	kbalt * 100h + kbalt	;	ALT ALT
CD7F	2020	C	dw	(20h) * 100h + (20h)	; 57 39h	SP SP
CD81	2020	C	dw	(20h) * 100h + (20h)	;	SP SP
CD83	2020	C	dw	(20h) * 100h + (20h)	;	SP SP
CD85	2020	C	dw	(20h) * 100h + (20h)	;	SP SP
CD87	C1C1	C	dw	kbcap * 100h + kbcap	; 58 3Ah	CapLk CapLk
CD89	C1C1	C	dw	kbcap * 100h + kbcap	;	CapLk CapLk
CD8B	C1C1	C	dw	kbcap * 100h + kbcap	;	CapLk CapLk
CD8D	C1C1	C	dw	kbcap * 100h + kbcap	;	CapLk CapLk
CD8F	3B00	C	dw	3B00h	; 59 3Bh	F01=X59
CD91	5400	C	dw	5400h	;	F11=X84
CD93	5E00	C	dw	5E00h	;	F21=X94
CD95	6800	C	dw	6800h	;	F31=X104
CD97	3C00	C	dw	3C00h	; 60 3Ch	F02=X60
CD99	5500	C	dw	5500h	;	F12=X85
CD9B	5F00	C	dw	5F00h	;	F22=X95
CD9D	6900	C	dw	6900h	;	F32=X105
CD9F	3D00	C	dw	3D00h	; 61 3Dh	F03=X61
CDA1	5600	C	dw	5600h	;	F13=X86
CDA3	6000	C	dw	6000h	;	F23=X96
CDA5	6A00	C	dw	6A00h	;	F33=X106
CDA7	3E00	C	dw	3E00h	; 62 3Eh	F04=X62
CDA9	5700	C	dw	5700h	;	F14=X87
CDAB	6100	C	dw	6100h	;	F24=X97
CDAD	6B00	C	dw	6B00h	;	F34=X107
CDAF	3F00	C	dw	3F00h	; 63 3Fh	F05=X63
CDB1	5800	C	dw	5800h	;	F15=X88
CDB3	6200	C	dw	6200h	;	F25=X98
CDB5	6C00	C	dw	6C00h	;	F35=X108
CDB7	4000	C	dw	4000h	; 64 40h	F06=X64
CDB9	5900	C	dw	5900h	;	F16=X89
CDBB	6300	C	dw	6300h	;	F26=X99
CDBD	6D00	C	dw	6D00h	;	F36=X109
CDBF	4100	C	dw	4100h	; 65 41h	F07=X65
CDC1	5A00	C	dw	5A00h	;	F17=X90
CDC3	6400	C	dw	6400h	;	F27=X100
CDC5	6E00	C	dw	6E00h	;	F37=X110
CDC7	4200	C	dw	4200h	; 66 42h	F08=X66

```

CDC9 5B00      C dw          5B00h      ;           F18=X91
CDCB 6500      C dw          6500h      ;           F28=X101
CDCD 6F00      C dw          6F00h      ;           F38=X111
CDCF 4300      C dw          4300h      ; 67 43h    F09=X67
CDD1 5C00      C dw          5C00h      ;           F19=X92
CDD3 6600      C dw          6600h      ;           F29=X102
CDD5 7000      C dw          7000h      ;           F39=X112
CDD7 4400      C dw          4400h      ; 68 44h    F10=X68
CDD9 5D00      C dw          5D00h      ;           F20=X93
CDDB 6700      C dw          6700h      ;           F30=X103
CDDD 7100      C dw          7100h      ;           F40=X113
CDDF C2C2      C dw    kbnun * 100h + kbnun ; 69 45h    NumLk  NumLk
CDE1 C2C2      C dw    kbnun * 100h + kbnun ;           NumLk  NumLk
CDE3 CACA      C dw    pause * 100h + pause ;           Pause  Pause
CDE5 C2C2      C dw    kbnun * 100h + kbnun ;           NumLk  NumLk
CDE7 C3C3      C dw    kbscr * 100h + kbscr ; 70 46h    ScrLk  ScrLk
CDE9 C3C3      C dw    kbscr * 100h + kbscr ;           ScrLk  ScrLk
CDEB C9C9      C dw    kbbrk * 100h + kbbrk ;           Break  Break
CDED C3C3      C dw    kbscr * 100h + kbscr ;           ScrLk  ScrLk
C
C ; -----
C ;           Numeric Keypad           | AT&T  |Other |
C ; ;                               | KB    | KBs  |
C ; -----
C
CDEF 4700      C dw          4700h      ; 71 47h    Home=X71
CDF1 3737      C dw    (37h) * 100h + (37h) ;           7      7
CDF3 7700      C dw          7700h      ;           X119
CDF5 D0D0      C dw    kdec7 * 100h + kdec7 ;           XDec7  XDec7
CDF7 4800      C dw          4800h      ; 72 48h    Up =X72
CDF9 3838      C dw    (38h) * 100h + (38h) ;           8      8
CDFB CDCD      C dw    kNONE * 100h + kNONE ;           None   None
CDFD CFCF      C dw    kdec8 * 100h + kdec8 ;           XDec8  XDec8
CDFF 4900      C dw          4900h      ; 73 49h    PgUp=X73
CE01 3939      C dw    (39h) * 100h + (39h) ;           9      9
CE03 8400      C dw          8400h      ;           X132
CE05 CECE      C dw    kdec9 * 100h + kdec9 ;           XDec9  XDec9
CE07 2D2D      C dw    (2Dh) * 100h + (2Dh) ; 74 4Ah    -      -
CE09 2D2D      C dw    (2Dh) * 100h + (2Dh) ;           -      -
CE0B CDCD      C dw    kNONE * 100h + kNONE ;           None   None
CE0D CDCD      C dw    kNONE * 100h + kNONE ;           None   None
CE0F 4B00      C dw          4B00h      ; 75 4Bh    Left=X75
CE11 3434      C dw    (34h) * 100h + (34h) ;           4      4
CE13 7300      C dw          7300h      ;           X115
CE15 D3D3      C dw    kdec4 * 100h + kdec4 ;           XDec4  XDec4
CE17 CDCD      C dw    kNONE * 100h + kNONE ; 76 4Ch    None   None
CE19 3535      C dw    (35h) * 100h + (35h) ;           5      5
CE1B CDCD      C dw    kNONE * 100h + kNONE ;           None   None
CE1D D2D2      C dw    kdec5 * 100h + kdec5 ;           XDec5  XDec5
CE1F 4D00      C dw          4D00h      ; 77 4Dh    Rght=X77
CE21 3636      C dw    (36h) * 100h + (36h) ;           6      6
CE23 7400      C dw          7400h      ;           X116
CE25 D1D1      C dw    kdec6 * 100h + kdec6 ;           XDec6  XDec6
CE27 2B2B      C dw    (2Bh) * 100h + (2Bh) ; 78 4Eh    +      +
CE29 2B2B      C dw    (2Bh) * 100h + (2Bh) ;           +      +

```

ROM BIOS Listing

```

CE2B CDCD      C dw  kNONE * 100h + kNONE ;           None None
CE2D CDCD      C dw  kNONE * 100h + kNONE ;           None None
CE2F 4F00      C dw  4F00h ; 79 4Fh End =X79
CE31 3131      C dw  (31h) * 100h + (31h) ;           1 1
CE33 7500      C dw  7500h ;           X117
CE35 D6D6      C dw  kdec1 * 100h + kdec1 ;           XDec1 XDec1
CE37 5000      C dw  5000h ; 80 50h Down=X80
CE39 3232      C dw  (32h) * 100h + (32h) ;           2 2
CE3B CDCD      C dw  kNONE * 100h + kNONE ;           None None
CE3D D5D5      C dw  kdec2 * 100h + kdec2 ;           XDec2 XDec2
CE3F 5100      C dw  5100h ; 81 51h PgDn=X81
CE41 3333      C dw  (33h) * 100h + (33h) ;           3 3
CE43 7600      C dw  7600h ;           X118
CE45 D4D4      C dw  kdec3 * 100h + kdec3 ;           XDec3 XDec3
CE47 C0C0      C dw  kbins * 100h + kbins ; 82 52h INS=X82 INS=X82
CE49 3030      C dw  (30h) * 100h + (30h) ;           0 0
CE4B CDCD      C dw  kNONE * 100h + kNONE ;           None None
CE4D D7D7      C dw  kdec0 * 100h + kdec0 ;           XDec0 XDec0
CE4F 5300      C dw  5300h ; 83 53h DEL =X83
CE51 2E2E      C dw  (2Eh) * 100h + (2Eh) ;           . .
CE53 C8C8      C dw  kbres * 100h + kbres ;           Reset Reset
CE55 C8C8      C dw  kbres * 100h + kbres ;           Reset Reset
C
C ; -----
C ;           Function Keypad           | AT&T | Other |
C ; ;                                     | KB  | KBs |
C ; -----
C
CE57 D8D8      C dw  kdb10 * 100h + kdb10 ; 84 54h 00 00
CE59 D8D8      C dw  kdb10 * 100h + kdb10 ;           00 00
CE5B CDCD      C dw  kNONE * 100h + kNONE ;           None None
CE5D CDCD      C dw  kNONE * 100h + kNONE ;           None None
CE5F CBCB      C dw  kbprt * 100h + kbprt ; 85 55h PrtSc PrtSc
CE61 CBCB      C dw  kbprt * 100h + kbprt ;           PrtSc PrtSc
CE63 7200      C dw  7200h ;           X114
CE65 CDCD      C dw  kNONE * 100h + kNONE ;           None None
CE67 CACA      C dw  pause * 100h + pause ; 86 56h Pause Pause
CE69 CACA      C dw  pause * 100h + pause ;           Pause Pause
CE6B CACA      C dw  pause * 100h + pause ;           Pause Pause
CE6D CACA      C dw  pause * 100h + pause ;           Pause Pause
CE6F OD0D      C dw  (0Dh) * 100h + (0Dh) ; 87 57h CR CR
CE71 OD0D      C dw  (0Dh) * 100h + (0Dh) ;           CR CR
CE73 0A0A      C dw  (0Ah) * 100h + (0Ah) ;           LF LF
CE75 CDCD      C dw  kNONE * 100h + kNONE ;           None None
CE77 4B00      C dw  4B00h ; 88 58h Left=X75
CE79 7300      C dw  7300h ;           Rev Word = X115
CE7B 7300      C dw  7300h ;           Rev Word = X115
CE7D 7300      C dw  7300h ;           Rev Word = X115
CE7F 5000      C dw  5000h ; 89 59h Down=X80
CE81 5000      C dw  5000h ;           Down=X80
CE83 5000      C dw  5000h ;           Down=X80
CE85 5000      C dw  5000h ;           Down=X80
CE87 4D00      C dw  4D00h ; 90 5Ah Rght=X77
CE89 7400      C dw  7400h ;           Adv Word = X116
CE8B 7400      C dw  7400h ;           Adv Word = X116

```

```

CE8D 7400          C dw          7400h          ;          Adv Word = X116
CE8F 4800          C dw          4800h          ; 91 5Bh   Up =X72
CE91 4700          C dw          4700h          ;          Home=X71
CE93 4700          C dw          4700h          ;          Home=X71
CE95 4700          C dw          4700h          ;          Home=X71
CE97 C9C9          C dw    kbrk * 100h + kbrk ; 92 5Ch   Break Break
CE99 C9C9          C dw    kbrk * 100h + kbrk ;          Break Break
CE9B C9C9          C dw    kbrk * 100h + kbrk ;          Break Break
CE9D C9C9          C dw    kbrk * 100h + kbrk ;          Break Break
CE9F C9C9          C dw    kbrk * 100h + kbrk ; 93 5Dh   Break Break
CEA1 C9C9          C dw    kbrk * 100h + kbrk ;          Break Break
CEA3 C9C9          C dw    kbrk * 100h + kbrk ;          Break Break
CEA5 C9C9          C dw    kbrk * 100h + kbrk ;          Break Break
CEA7 C2C2          C dw    kbnm * 100h + kbnm ; 94 5Eh   NumLk NumLk
CEA9 C2C2          C dw    kbnm * 100h + kbnm ;          NumLk NumLk
CEAB CACA          C dw    pause * 100h + pause ;          Pause Pause
CEAD C2C2          C dw    kbnm * 100h + kbnm ;          NumLk NumLk
CEAF 2F2F          C dw    (2Fh) * 100h + (2Fh) ; 95 5Fh   / /
CEB1 2F2F          C dw    (2Fh) * 100h + (2Fh) ;          / /
CEB3 CDCD          C dw    kNONE * 100h + kNONE ;          None None
CEB5 CDCD          C dw    kNONE * 100h + kNONE ;          None None
CEB7 5400          C dw          5400h          ; 96 60h   F11=X84
CEB9 5400          C dw          5400h          ;          F11=X84
CEBB 5400          C dw          5400h          ;          F11=X84
CEBD 5400          C dw          5400h          ;          F11=X84
CEBF 5500          C dw          5500h          ; 97 61h   F12=X85
CEC1 5500          C dw          5500h          ;          F12=X85
CEC3 5500          C dw          5500h          ;          F12=X85
CEC5 5500          C dw          5500h          ;          F12=X85
CEC7 5600          C dw          5600h          ; 98 62h   F13=X86
CEC9 5600          C dw          5600h          ;          F13=X86
CECB 5600          C dw          5600h          ;          F13=X86
CECD 5600          C dw          5600h          ;          F13=X86
CECF 5700          C dw          5700h          ; 99 63h   F14=X87
CED1 5700          C dw          5700h          ;          F14=X87
CED3 5700          C dw          5700h          ;          F14=X87
CED5 5700          C dw          5700h          ;          F14=X87
CED7 5800          C dw          5800h          ; 100 64h  F15=X88
CED9 5800          C dw          5800h          ;          F15=X88
CEDB 5800          C dw          5800h          ;          F15=X88
CEDD 5800          C dw          5800h          ;          F15=X88
CEDF 5900          C dw          5900h          ; 101 65h  F16=X89
CEE1 5900          C dw          5900h          ;          F16=X89
CEE3 5900          C dw          5900h          ;          F16=X89
CEE5 5900          C dw          5900h          ;          F16=X89
CEE7 5A00          C dw          5A00h          ; 102 66h  F17=X90
CEE9 5A00          C dw          5A00h          ;          F17=X90
CEEB 5A00          C dw          5A00h          ;          F17=X90
CEED 5A00          C dw          5A00h          ;          F17=X90
CEEF 5B00          C dw          5B00h          ; 103 67h  F18=X91
CEF1 5B00          C dw          5B00h          ;          F18=X91
CEF3 5B00          C dw          5B00h          ;          F18=X91
CEF5 5B00          C dw          5B00h          ;          F18=X91
CEF7          C
CEF7          C kb_data1    endp

```

```

C
C include wd_hdu.asm
C
C ; TITLE WX2BIOS Version 8 85/04/02
C ; SUBTTL BIOS for Western Digital Winchester Controllers
C PAGE 62,132
C ; Integrated Verion F to our existing BIOS. 04/07/85
C ; From Dave Joan, Western Digital.
C ; (1) BIOS polling status at end of rest pulse caused 5 to 9 us pulses
C ; on all drive lines due to 1015 ports not lbeing set up. A delay of
C ; at least 3 milliseconds between reset and polling was added to the
C ; reset command routine.
C ; (2) Install sets up registers AL,DH,CX in case System BIOS did not.
C ;
C ; Ported Version E to our version of the WX2BIOS. I have added enhancements
C ; like: 1) deglitching the interrupt service routine; 2) inserting time outs
C ; around the wait for interrupt loop
C ; Version E
C ; Dave Joan 85/02/07
C ; Version 8 (begun from NLSBIOS version 1)
C ; Bob Hossley 83/10/10 - 83/10/10
C ; Exact copy of NLSBIOS version 1
C ; Version 1
C ; Bob Hossley 83/10/06 - 83/10/06
C ; 83/10/06 Reset code refined work with SHD artwork revision 1, 2, or 3
C ; Version 0 (begun from WX2BIOS version 6)
C ; Bob Hossley 83/10/05 - 83/10/06
C ; Bill Bailey 83/10/04 - 83/10/04
C
C ;-----
C ;
C ; Basic (i.e. Fundamental) Input/Output System for the Western Digital ;
C ; WX2 Winchester Controller board ;
C ;
C ; This BIOS provides access to 5 1/4 inch fixed disks via a ;
C ; controller compatible with the IBM fixed disk controller. ;
C ;
C ; The BIOS routines are meant to be entered only via software ;
C ; interrupts. Addresses in these listings should never be referenced. ;
C ; Applications which reference absolute addresses in the code segment ;
C ; violate the structure and design of this BIOS. ;
C ;-----
C
B9FC C code segment common 'ROM'
C assume cs:code, ds:nothing, es:nothing, ss:nothing
C public bios_install
C public i13_ih
CEF8 C ORG OCEF8H
C
= 0001 C DMACC EQU 01h ; non zero for dma accelerator code
C ; Error Codes Returned by BIOS
C
= 00FF C ec_stat EQU OFFh ;Read status failed
= 00BB C ec_undef EQU 0BBh ;Undefined error

```



```

= 0080      C  ec_time      EQU    080h      ;No response from device
= 0040      C  ec_seek      EQU    040h      ;Seek failed
= 0020      C  ec_cntlr    EQU    020h      ;Controller failed self test
= 0011      C  ec_ecc_cor    EQU    011h      ;ECC corrected data error
= 0010      C  ec_ecc_un    EQU    010h      ;Uncorrectable data error
= 000B      C  ec_bad_trk    EQU    00Bh      ;Bad track
= 0009      C  ec_dma_64k    EQU    009h      ;Attempt to DMA across 64K boundary
= 0007      C  ec_init      EQU    007h      ;Initialize drive failed
= 0005      C  ec_reset     EQU    005h      ;Reset failed
= 0004      C  ec_sec_not_fnd EQU    004h      ;Sector not found
= 0002      C  ec_addr_mark EQU    002h      ;Address mark not found
= 0001      C  ec_bc         EQU    001h      ;Bad command
= 0000      C  ec_no_err     equ    000      ;no error
C
C ; Error codes returned by the wx2
C
= 0018      C  erc_corr      equ    18h      ;ECC error corrected
C
C ; BIOS Command Codes
C
= 0000      C  bc_reset     EQU    0          ;reset controller
= 0001      C  bc_cc        EQU    1          ;return last completion code
= 0002      C  bc_rd        EQU    2          ;read sectors
= 0003      C  bc_wr        EQU    3          ;write sectors
= 0004      C  bc_vr        EQU    4          ;verify sectors
= 0005      C  bc_ft        EQU    5          ;format track
= 0006      C  bc_fbt       EQU    6          ;format bad track
= 0007      C  bc_fd        EQU    7          ;format drive
= 0008      C  bc_par_rd    EQU    8          ;read parameters
= 0009      C  bc_par_set   equ    9          ;set parameters
= 000A      C  bc_rdl       equ    10         ;read long
= 000B      C  bc_wrl       equ    11         ;write long
= 000C      C  bc_seek     equ    12         ;seek
= 000D      C  bc_reset_1   equ    13         ;reset controller
= 000E      C  bc_buff_rd    equ    14         ;read sector buffer
= 000F      C  bc_buff_wr    equ    15         ;write sector buffer
= 0010      C  bc_tst_rdy   equ    16         ;test drive ready
= 0011      C  bc_recal    equ    17         ;recalibrate
= 0012      C  bc_diag_ram  equ    18         ;ram diagnostic
= 0013      C  bc_diag_drv  equ    19         ;drive diagnostic
= 0014      C  bc_diag_ctlr  equ    20        ;controller diagnostic
= 0015      C  bc_dasd     equ    21         ; new cmd 15H ; read DASD type%
C
C ; Screen BIOS command codes
= 000E      C  bc_v_w        equ    14          ;write character to screen
C
C ; Disk Controller Command Codes
C
= 0000      C  dc_tst_rdy    EQU    0          ;TEST READY
= 0001      C  dc_recal    EQU    1          ;RECALIBRATE
= 0003      C  dc_stat_rd   EQU    3          ;READ STATUS
= 0004      C  dc_fd        EQU    4          ;FORMAT DRIVE
= 0005      C  dc_vr        EQU    5          ;verify sectors
= 0006      C  dc_ft        EQU    6          ;format track
= 0007      C  dc_fbt       EQU    7          ;FORMAT BAD TRACK

```

```

= 0008      C dc_rd      EQU      8          ;READ SECTORS
= 000A      C dc_wr      EQU      0AH        ;WRITE SECTORS
= 000B      C dc_seek    EQU      0BH        ;SEEK
= 000C      C dc_par_set  EQU      0CH        ;SET DRIVE PARAMETERS
= 000D      C dc_ecc_rd   EQU      0DH        ;READ ERROR BURST LENGTH
= 000E      C dc_buff_rd  EQU      0EH        ;READ SECTOR BUFFER
= 000F      C dc_buff_wr  EQU      0FH        ;WRITE SECTOR BUFFER
= 00E0      C dc_diag_ram EQU      0E0H       ;RAM DIAGNOSTIC
= 00E3      C dc_diag_drv EQU      0E3H       ;DRIVE DIAGNOSTIC
= 00E4      C dc_diag_ctlr EQU     0E4H       ;CONTROLLER DIAGNOSTIC
= 00E5      C dc_rdl     EQU      0E5H       ;READ LONG
= 00E6      C dc_wrl     EQU      0E6H       ;WRITE LONG

C
C ; Command Control Block offsets
= 0000      C ccb_cmd    equ      0          ;command code
= 0004      C ccb_blks   equ      4          ;number of blocks or interleave factor
= 0005      C ccb_opt    equ      5          ;option byte

C
C ; Bits in Byte 1 of the ccb
C
= 0020      C ccb_drv_b   equ      20h        ;drive bit

C
C ; CCB option byte
C ; Bit
C ; 0, 1, 2 Step option
C ; 3, 4, 5 Reserved for future use
C ; 6 0 --> Stable ECC required before correction
C ; 1 --> Immediate ECC correction
C ; 7 0 --> Retries allowed
C ; 1 --> No retries allowed

C
C ; Command Completion byte
C
= 0002      C cc_er      equ      2          ;error flag bit: 0 --> no error

C
C ; offsets in a winchester parameter subtable
= 0000      C wst_cyl    equ      0          ;2-byte number of cylinders
= 0002      C wst_heads  equ      2          ;1-byte number of heads
= 0003      C wst_re_wr   equ      3          ;2-byte starting reduced write cylinder
= 0005      C wst_wr_pre  equ      5          ;2-byte starting write pre-comp cyl
= 0007      C wst_er_bur  equ      7          ;1-byte max. error burst length
= 0008      C wst_opt     equ      8          ;1-byte option byte for CCB
= 0009      C wst_sto     equ      9          ;1-byte standard time out parameter%
= 000A      C wst_fto     equ      10         ;1-byte format drive time out parameter%
= 000B      C wst_ddto    equ      11         ;1-byte drive diagnostic time out parameter%

C
C ; Activation Record Descriptions
C
= 0000      C a1_es      equ      0
= 0002      C a1_ds      equ      2
= 0004      C a1_si      equ      4
= 0006      C a1_di      equ      6
= 0008      C a1_bp      equ      8
= 000A      C a1_dx      equ      10
= 000C      C a1_cx      equ      12

```

```

= 000E      C a1_bx      equ    14
C
C ; Interrupt vector numbers
= 0010      C ivn_dis_char equ    10h      ;call to screen bios
= 0013      C ivn_bc      equ    13h      ;call to diskette or winchester bios
= 0018      C ivn_basic   equ    18h      ;enter ROM resident BASIC
= 0040      C ivn_bc_dette equ    40h
C
C ; Timer Values
= 0165      C ti_0_1     equ    446d-89d    ;time to wait after drive 0 is found
C                                           ;usable = 89 * .056 = 5 seconds
= 019A      C ti_kb_reset equ    446d-36d    ;initial timer counter for kb reset
C                                           ; 36 * .056 = 2 seconds
= 01BE      C ti_fin     equ    446d      ;final timer counter
C                                           ;.056 seconds per count
= 0584      C ti_bc_reset equ    1412d      ;delay loop count for 3 ms minimum.
C                                           ; based on 17 cyc X .125 us = 1 loop
C
C ; Miscellaneous symbols
C
= 0004      C ctlr_mx    equ    4          ;maximum number of wx2 controllers
= 0002      C drv_ctlr   EQU    2          ;number of drives per controller
= 0008      C drv_total  EQU    ctlr_mx*drv_ctlr ;total number of drives possible
= 0080      C dnwin     equ    80h        ;least significant winchester drive #
= 0200      C sec_size   equ    512       ;number of bytes per sector
C
C ; Hardware specific values
C ; Port Addresses
= 0000      C p_dma     EQU    0          ;base address for DMA chip registers
= 0020      C p_int     EQU    20H        ;8259 control port
= 0082      C p_dma_latch EQU    082H      ;4-bit latch for bits 16 - 19 of DMA
C                                           ;address
= 0063      C p_dmacc   EQU    63H        ;port for dma accelerator %
C                                           ;predictor sector size and sample enable%
= 0320      C p_wx2     EQU    0320H      ;Base address for WX2 ports
C
= 0004      C wx2_l     equ    4          ;number of ports per WX2 controller
C
= 0067      C sw_b      equ    67H        ;mother board switch%
C
C ; Offsets from base address for one WX2 controller
C ; Read Use
= 0000      C wx2_r_data equ    0          ;data from WX2 to host
= 0001      C wx2_r_status equ    1      ;WX2 status
C                                           ;Caution: subroutine wx2_req requires that
C                                           ; wx2_r_data + 1 = wx2_r_status
C                                           ;Caution: subroutine ccb_send requires that
C                                           ;wx2_r_stat = wx2_w_select -1
= 0002      C wx2_r_config equ    2      ;configuration switches
C
C ; Write Use
= 0000      C wx2_w_data equ    0          ;data from host to WX2
= 0001      C wx2_w_reset equ    1      ;reset WX2 controller
= 0002      C wx2_w_select equ    2      ;select WX2 controller
C                                           ;Caution: subroutine ccb_send requires that

```

```
C ; wx2_w_msk = wx2_w_select +1
= 0003 C wx2_w_msk equ 3 ;mask register
= 000C C wx2_lrg_offset equ 12 ; largest port offset for hard disk controller%
C
C ; Bits in wx2_r_status
= 0008 C wx2_stat_busy EQU 00001000B ; Busy
= 0004 C wx2_stat_cd EQU 00000100B ; command/data
= 0002 C wx2_stat_io EQU 00000010B ; input/output
= 0001 C wx2_stat_req EQU 00000001B ; request (start state machine)
= 0020 C wx2_stat_int equ 00100000b ;interrupt
= 0010 C wx2_stat_drq equ 00010000b ;dma data request
C
C ; Bits in wx2_r_config
= 000C C wx2_config_0 equ 00001100b ;switch field for drive 0
= 0003 C wx2_config_1 equ 00000011b ;switch field for drive 1
C
C ; Bits in wx2_w_msk (bits 2 - 7 are don't cares)
= 0001 C wx2_msk_dma equ 01b ;dma mask bit
C ; 0 --> dma to host disabled
C ; 1 --> dma to host enabled
= 0002 C wx2_msk_int equ 10b ;interrupt mask bit
C ; 0 --> interrupt to host disabled
C ; 1 --> interrupt to host enabled
C
C ; Offsets from the DMA base address
C ; Write
C ; For each channel there is a 16-bit address register and a 16-bit
C ; byte count register. For channel i (i = 0, 1, 2, 3) the address
C ; register is at address i*2 and the byte count register is at address
C ; (i*2) + 1.
= 0006 C dma_w_addr equ 3*2 ;channel 3 address register (16-bits)
= 0007 C dma_w_cnt equ 3*2+1 ;channel 3 byte count register (16-bits)
= 0008 C dma_w_cmd equ 8 ;command register (8 bits)
= 000B C dma_w_mode equ 11 ;mode register (6 bits)
= 0009 C dma_w_req equ 9 ;request register (4 bits)
= 000A C dma_w_mask_b equ 10 ;write single mask register bit
= 000F C dma_w_mask equ 15 ;mask register (4 bits)
= 000C C dma_w_byte equ 12 ;clear pointer to register byte
C ;0 --> bits 0 - 7 of register
C ;1 --> bits 8 - 15 of register
= 000D C dma_w_clr equ 13 ;master clear
C ; Read
= 0008 C dma_r_status equ 8 ;status register (8 bits)
C
C ; Commands for DMA controller
C ; Commands for mode register
C ENDEF
C IF DMACC
= 0007 C dma_mode_wr EQU 00000111B ;bits 6, 7 = 01 --> DEMAND mode select%
C ;bits 2, 3 = 01 --> write transfer%
C ;bits 0, 1 = 11 --> Channel 3 select%
= 000B C dma_mode_rd EQU 00001011B ;bits 6, 7 = 01 --> DEMAND mode select%
C ;bits 2, 3 = 10 --> read transfer%
C ;bits 0, 1 = 11 --> Channel 3 select%
```

```

C   ENDIF
C                                     ;bits 0, 1 = 11 --> Channel 3 select
C ;   Commands for mask register bit change register
= 0003 C dma_mask_b_3   equ    11b           ;bits 0, 1 = 3 --> channel 3 mask bit
= 0004 C dma_mask_b_s   equ    100b         ;bit 2 = 1 --> set mask bit
C
C IF DMACC
= 0000 C dmanorm       equ    0             ;normal dma flag%
= 0001 C dmalong       equ    1             ;long dma flag%
C   ENDIF
C
C ; Offsets from p_int
C ;   write
= 0000 C int_w_ocw2    equ    0             ;Operation Control Word 0:
C                                     ; rotate & end of interrupt control
C ;   Read or Write
= 0001 C int_ocw1      equ    1             ;Operation Control Word 1: set
C                                     ; the interrupt mask register
C ; Bits in int_ocw1
= 0001 C int_ocw1_m0   equ    00000001b    ;channel zero mask
C                                     ; The clock used channel 0.
= 0020 C int_ocw1_m5   equ    00100000b    ;channel 5 mask
C                                     ; The WX2 uses channel 5.
C
C ; Commands for ocw2
= 0020 C int_ocw2_eoi  EQU    20H          ;End of interrupt command
C ;
C ;-----:
C ;           Interrupt Vector Area           :
C ;-----:
C
0000 C intvec  SEGMENT AT 0
C
0034 C         ORG    4*0dh              ;winchester interrupt
0034 C iv_int   LABEL  DWORD
C
004C C         ORG    4*ivn_bc          ;call to diskette or winchester BIOS
004C C iv_bc    LABEL  DWORD
C
0064 C         ORG    4*19h            ;call to winchester BIOS to boot system
0064 C iv_boot  LABEL  DWORD
C
0078 C         ORG    1EH*4            ;Diskette parameter table
0078 C iv_p_tbl_dett LABEL  DWORD
C
0100 C         ORG    4*ivn_bc_dette   ;call to diskette BIOS
0100 C iv_bc_dett LABEL  DWORD
C
0104 C         ORG    041H*4          ;winchester parameter table
0104 C iv_p_tbl_win LABEL  DWORD
C
7C00 C         ORG    7C00H           ;address of buffer for disk resident
C                                     ;boot code
7C00 C iv_boot_buf LABEL  FAR         ;A jump to this label begins execution
C                                     ;of the disk resident boot code

```

```

C
7C00 C intvec ENDS
C
C
C ;-----:
C ; RAM Workspace :
C ;-----:
C
= 0040 C W2RAM EQU 40H ; needed for masm%
C
0000 C wdram SEGMENT AT 40H ;%
C
C
0042 C ORG 42H
0042 C ram_ccb LABEL BYTE ;6-byte Command Control Block
0042 C ram_stat LABEL BYTE ;4-bytes of controller status
C
006C C ORG 06CH
006C ??? C ram_time DW ? ;Timer low word
0072 C ORG 72H
0072 ??? C ram_kb_reset DW ? ;1234H if keyboard reset underway
0074 C ORG 74H
0074 ?? C ram_cc DB ? ;completion code
0075 ?? C ram_drv_cnt DB ? ;Number of drives on WX2 controller 0
0076 ?? C ram_opt DB ? ;spare - was temp for option byte
0077 ?? C ram_po DB ? ;spare - port offset temp
0078 C wdram ENDS
C
C ;*****
C
C ;code segment
C ;
C ; org 0
C ; db 55h,0aah ;BIOS header
C ; db 16 ;number of 512 blocks in the block of
C ; ;host memory allocated to this BIOS.
C ; ;If the BIOS ROM used is smaller, then
C ; ;it appears several times in the host
C ; ;memory
C ;=====
C ;
C ; Call: call ROM-base-address + 3
C ; return
C ;
C ; Install Winchester BIOS into the interrupt vector system.
C ;
C ; Entry: No entry parameters.
C ;
C ; Exit: No exit parameters.
C ;
C ; The message " Not Ready" is displayed if there are no usable winchester
C ; disks.
C
C assume cs:code ;call to initialize is required to
C ;establish cs

```

```

CEFB  E9 CFB4 R      C          jmp      near ptr bios_install ;entry for power-up initialize
C
C ;=====
C ;
C ; Call          JMP ROM_base_address +5
C ; Purpose:     format the specified drive with the specified interleave.
C ;
C ; Entry:
C ;      (AH) = Relative number of target drive. Drive 80h + d is the target.
C ;      (AL) = Interleave factor.
C ;
C ; Exit: Job terminated.
C
C ;          jmp      wx2_fmt          ;format entry
C ;-----
C
CEFB  28 43 29 20 43 6F C          db      '(C) Copyright 1983 Western Digital Corporation'
      70 79 72 69 67 68 C
      74 20 31 39 38 33 C
      20 57 65 73 74 65 C
      72 6E 20 44 69 67 C
      69 74 61 6C 20 43 C
      6F 72 70 6F 72 61 C
      74 69 6F 6E      C
C
CF29  C          p_tbl_dett:
CF29  CF          C          DB      11001111B          ;srt=C, HD unload = F
CF2A  02          C          DB      2              ;HD load = 1, mode = DMA
CF2B  25          C          DB      25h           ;wait after open til motor off
CF2C  02          C          DB      2              ;512 bytes per sector
CF2D  08          C          DB      8              ;EOT
CF2E  2A          C          DB      02AH          ;gap length
CF2F  FF          C          DB      0FFH          ;DTL
CF30  50          C          DB      050H          ;gap length for format
CF31  F6          C          DB      0F6H          ;fill byte for format
CF32  19          C          DB      25            ;head settle time in milliseconds
CF33  04          C          DB      4            ;motor start time (1/8 seconds)
C
C ; Winchester Parameter Table
C ;
C ; Offset
C ; 0 - 15      subtable 0
C ; 16 - 31     subtable 1
C ; 32 - 47     subtable 2
C ; 48 - 63     subtable 3
C ; 64 - 79     subtable 4%
C ; 80 - 95     subtable 5%
C ; 96 - 111    subtable 6%
C ; 112 - 127   subtable 7%
C ;
C ; The configuration switches accessible via port wx2_r_config specify
C ; the subtable to be used for each drive. Each drive has two switches
C ; which determine a 2-bit field in port wx2_r_config which specifies
C ; the subtable number for the drive.
C ; Two auxiliary switches have been added to expand the parameter table.%

```

```

C ; Bits 3 and 2 (0CH) on the mother board (port 067H) are read for one%
C ; extra bit per drive; bit 3 for drive 0, bit 2 for drive 1.%
C ;
C ; Winchester Parameter Subtable
C ;
C ; Offset      Byte Length
C ; 0           2           Number of cylinders
C ; 2           1           Number of heads
C ; 3           2           Starting reduced write current cylinder
C ; 5           2           Starting write precompensation cylinder
C ; 7           1           Max. correctable error burst length
C ; 8           1           CCB option byte
C ; 9           1           Standard time out value
C ; 10          1           Format time out value
C ; 11          1           Check drive time out value
C ; 12          4           Reserved for future use
C
CF34          C p_tbl_win:
C ; Table Entry 0 - 10Mb
C ; This is the 10Mb drive supported as the default drive in the PC6300
C
CF34 0132    C          DW      0306D
CF36 04      C          DB      04
CF37 0132    C          DW      0306D
CF39 0000    C          DW      0
CF3B 0B      C          DB      0BH
CF3C 05      C          DB      5
CF3D 0C      C          DB      0CH
CF3E B4      C          DB      0B4H
CF3F 28      C          DB      28H
CF40 00 00 00 00 C          DB      0,0,0,0
C
C ; Table Entry 1 - 30Mb
C ; CDC Wren Hard Disk Drive
C
CF44 02B9    C          DW      0697D
CF46 05      C          DB      05
CF47 02B9    C          DW      0697D
CF49 0000    C          DW      0
CF4B 0B      C          DB      0BH
CF4C 05      C          DB      5
CF4D 10      C          DB      10H
CF4E D0      C          DB      0D0H
CF4F 60      C          DB      60H
CF50 00 00 00 00 C          DB      0,0,0,0
C
C ; Table Entry 2 - 20Mb
C ; CMI CM6426 Hard Disk Drive
C
CF54 0280    C          DW      0640D
CF56 04      C          DB      04
CF57 0100    C          DW      0256D
CF59 0100    C          DW      0256D
CF5B 0B      C          DB      0BH
CF5C 05      C          DB      5

```

CF5D	0C	C	DB	0CH
CF5E	B4	C	DB	0B4H
CF5F	28	C	DB	28H
CF60	00 00 00 00	C	DB	0,0,0,0
		C		
		C		; Table Entry 3 - 40Mb
		C		; Tandon Hard Disk Drive
		C		
CF64	03D5	C	DW	0981D
CF66	05	C	DB	05
CF67	03D6	C	DW	0982D
CF69	FFFF	C	DW	0FFFFH
CF6B	0B	C	DB	0BH
CF6C	05	C	DB	5
CF6D	18	C	DB	18H
CF6E	FF	C	DB	0FFH
CF6F	90	C	DB	90H
CF70	00 00 00 00	C	DB	0,0,0,0
		C		
		C		; Table Entry 4 - 40Mb
		C		; Seagate Hard Disk Drive ST4051
		C		
CF74	03D1	C	DW	0977D
CF76	05	C	DB	05
CF77	FFFF	C	DW	0FFFFH
CF79	01F4	C	DW	0500D
CF7B	0B	C	DB	0BH
CF7C	05	C	DB	5
CF7D	18	C	DB	18H
CF7E	FF	C	DB	0FFH
CF7F	90	C	DB	090H
CF80	00 00 00 00	C	DB	0,0,0,0
		C		
		C		; Table Entry 5 - 80 Mb
		C		; Miniscribe Hard Disk Drive Model 6086
		C		
CF84	0400	C	DW	1024D
CF86	08	C	DB	08
CF87	0401	C	DW	1025D
CF89	0200	C	DW	512D
CF8B	0B	C	DB	0BH
CF8C	05	C	DB	5
CF8D	30	C	DB	30H
CF8E	FF	C	DB	0FFH
CF8F	BB	C	DB	0BBH
CF90	00 00 00 00	C	DB	0,0,0,0
		C		
		C		; Table Entry 6 - 67 Mb
		C		; Micropolis Hard Disk Drive Model 1325
		C		
CF94	0400	C	DW	1024D
CF96	08	C	DB	08
CF97	0401	C	DW	1025D
CF99	0400	C	DW	1024D
CF9B	0B	C	DB	0BH

ROM BIOS Listing

```

CF9C 05          C          DB          5
CF9D 25          C          DB          25H
CF9E FF          C          DB          OFFH
CF9F A0          C          DB          0A0H
CFA0 00 00 00 00 C          DB          0,0,0,0
C
C ; Default table 7 - 20 Mb
C ; Seagate ST225 Hard Disk Drive or Miniscribe 3425 Hard Disk Drive
C
CFA4 0264        C          DW          0612D
CFA6 04          C          DB          04
CFA7 0265        C          DW          0613D
CFA9 0100        C          DW          256D
CFAB 0B          C          DB          0BH
CFAC 05          C          DB          5
CFAD 0C          C          DB          0CH
CFAE B4          C          DB          0B4H
CFAF 28          C          DB          28H
CFB0 00 00 00 00 C          DB          0,0,0,0
C
C ;-----
C
CFB4             C bios_install proc near
CFB4 33 C0        C          xor          ax,ax          ;zero
CFB6 8E D8        C          mov          ds,ax          ;initialize data segment register
C          assume ds:intvec
C
C ; Set interrupt vectors to install BIOS into operating system
C
CFB8 FA          C          cli          ;disable interrupts
CFB9 C4 06 004C R C          les          ax,iv_bc          ;20-bit pointer to entry for
C          ;call to diskette bios
CFBD A3 0100 R    C          mov          word ptr iv_bc_dett,ax ;initialize new pointer
CFC0 8C 06 0102 R C          mov          word ptr iv_bc_dett+2,es
CFC4 C7 06 004C R D17D R C          mov          word ptr iv_bc,offset i13_ih ;20-bit pointer to entry for
CFCA 8C 0E 004E R C          mov          word ptr iv_bc+2,cs          ;call to winchester BIOS
CFCE C7 06 0064 R D0ED R C          mov          word ptr iv_boot,offset i19_boot_sys ;20-bit pointer to boot system code
CFD4 8C 0E 0066 R C          mov          word ptr iv_boot+2,cs
CFD8 C7 06 0034 R D161 R C          mov          word ptr iv_int,offset id_ih ;20-bit pointer to disk interrupt
CFDE 8C 0E 0036 R C          mov          word ptr iv_int+2,cs          ;handler
CFE2 C7 06 0104 R CF34 R C          mov          word ptr iv_p_tbl_win,offset p_tbl_win ;20-bit pointer to winchester
CFE8 8C 0E 0106 R C          mov          word ptr iv_p_tbl_win+2,cs ;parameter table
CFEC FB          C          sti          ;enable interrupts
C
C          assume ds:wdrum
CFED B8 0040        C          mov          ax,w2RAM          ;ram segment address
CFF0 8E D8        C          mov          ds,ax          ;establish segment
C
C ; If a power-up boot is in progress, then wait up to 25 seconds for the
C ; first winchester drive to spin up.
C
CFF2 C6 06 0075 R 00 C          mov          ram_drv_cnt,00          ;clear number of drives
CFF7 B8 019A        C          mov          ax,ti_kb_reset          ;initial timer value for keyboard reset
CFFA 81 3E 0072 R 1234 C          cmp          ram_kb_reset,1234h          ;keyboard reset?
D000 74 02        C          je          kb_reset          ;jump if yes

```

```

D002 33 C0      C      xor    ax,ax          ;clear for long delay while disk
C                                     ;spins up after power on
D004           C  kb_reset:
D004 A3 006C R  C      mov    ram_time,ax      ;initialize timer counter
C
D007 FA        C      cli                    ;disable interrupts
D008 E4 21     C      in     al,p_int+int_ocw1    ;read interrupt mask register
D00A 24 FE     C      and    al,not int_ocw1_m0    ;enable interrupts on channel 0
D00C E6 21     C      out    p_int+int_ocw1,al    ;set interrupt mask register
C                                     ;This starts the timer.
D00E FB        C      sti                    ;enable interrupts
C
C      ; Count the usable winchester drives. To speed this counting, if a
C      ; controller is unusable, then testing of its second drive is skipped.
C      ; A drive is usable if and only if its controller resets, runs its
C      ; controller diagnostic, recalibrates the drive, and executes test
C      ; drive ready all without error.
C
D00F B2 80     C      mov    dl,dnwin          ;base winchester drive number
C
D011           C  ctrl_init:
C      ; These 3 lines set up regs for Test Drive Ready in case host didn't. V.F.
D011 B9 0001   C      mov    cx,1             ; cylinder 0, sector 1
D014 8A F5     C      mov    dh,ch            ;head # of zero
D016 8A C1     C      mov    al,cl            ;sector count of 1
C
D018 B4 00     C      mov    ah,bc_reset      ;bios call code
D01A CD 13     C      int    ivn_bc           ;call disk bios
D01C 72 2F     C      jc     nxt_ctrl        ;jump if error
C
D01E B4 14     C      mov    ah,bc_diag_ctrl  ;bios call code
D020 CD 13     C      int    ivn_bc           ;call disk bios
D022 72 29     C      jc     nxt_ctrl        ;jump if error
C
D024           C  trdy0:
D024 B4 10     C      mov    ah,bc_tst_rdy    ;bios call code%
D026 CD 13     C      int    ivn_bc           ;call disk bios%
D028 73 08     C      jnc    drv              ;jump if drive ready%
D02A 81 3E 006C R 0222 C      cmp    ram_time,222h    ; 30 sec Time-out?%
D030 72 F2     C      jb     trdy0           ;jump if no time-out yet%
C
D032           C  drv:
D032 33 C0     C      xor    ax,ax
D034 A3 006C R  C      mov    ram_time,ax      ;clear timer value%
D037 B4 11     C      mov    ah,bc_recal      ;bios call code
D039 CD 13     C      int    ivn_bc           ;call disk bios
D03B 72 50     C      jc     nxt_drv        ;jump if error
C
D03D           C  tst_drv_rdy:
D03D B4 10     C      mov    ah,bc_tst_rdy    ;bios call code
D03F CD 13     C      int    ivn_bc           ;call disk bios
D041 73 33     C      jnc    drv_rdy         ;jump if drive ready
D043 81 3E 006C R 01BE C      cmp    ram_time,ti_fin    ;Time-out?
D049 72 F2     C      jb     tst_drv_rdy     ;jump if no time-out yet
C

```

ROM BIOS Listing

```

D04B EB 40          C          jmp      short nxt_drv      ;go to next drive
C ;-----
C
D04D              C  nxt_ctlr:
D04D FE C2          C          inc      dl                  ;skip drive 1 on unusable controller
D04F EB 3C          C          jmp      short nxt_drv
C ;-----
C
D051 20 20 4E 6F 74 20 C  hbad:      db      ' Not Ready',0Ah,0Dh
      52 65 61 64 79 0A C
      0D          C
= 000D            C  hbad$      equ      $-hbad
D05E 20 20 52 65 61 64 C  hgood:     db      ' Ready',0Ah,0Dh
      79 0A 0D    C
= 0009            C  hgood$     equ      $-hgood
D067 20 20 4E 6F 74 20 C  habs:      db      ' Not Present',0Ah,0Dh
      50 72 65 73 65 6E C
      74 0A 0D    C
= 000F            C  habss$     equ      $-habs
C
C
C
D076              C  drv_rdy:
D076 FE 06 0075 R    C          inc      ram_drv_cnt      ;update drive count
C
D07A 80 FA 80          C          cmp      dl,dnwin            ; 1st drive?
D07D 75 0E            C          jne      nxt_drv          ;jump if no
D07F 81 3E 0072 R 1234 C          cmp      ram_kb_reset,1234h ;keyboard reset?
D085 74 06            C          je       nxt_drv          ;jump if yes
D087 C7 06 006C R 0165 C          mov      ram_time,ti_0_1      ;1st drive usable --> only wait
C ;up to 5 seconds more for other
C ;drives
D08D              C  nxt_drv:
D08D FE C2          C          inc      dl                  ;update drive number
D08F F6 C2 01          C          test     dl,1                ;Drive 1 on target controller?
D092 75 9E            C          jnz      drv                  ;jump if yes.%
D094 80 FA 88          C          cmp      dl,dnwin+drv_total ;more wx2's to test?
C ;          jb      ctlr_init        ;jump if yes.
D097 73 03            C          jae      usable              ; continue if usable
D099 E9 D011 R          C          jmp      ctlr_init              ; go to ctlr_init if not
C
D09C              C  usable:
D09C 80 3E 0075 R 00    C          cmp      ram_drv_cnt,0        ;Any usable winchesters?
D0A1 75 17            C          jne      wins_usable         ;jump if yes.
C
C          ; Because no winchester drives are usable, display error message " Not Ready"
C          ; and set the error exit flag.
C
D0A3 BE D051 R          C          mov      si,offset hbad      ;pointer to message string
D0A6 FC              C          cld                          ;clear direction --> inc si
D0A7 B9 000D           C          mov      cx,hbad$              ;number of bytes in message
D0AA B3 01            C          mov      bl,1                    ;choose foreground color%
C
D0AC              C  dis_char:
D0AC 2E: AC           C          lods     byte ptr hbad          ;load message byte
D0AE B4 0E            C          mov      ah,bc_v_w                ;bios call code

```

```

DOB0 CD 10          C      int   ivn_dis_char      ;call screen bios
DOB2 E2 F8          C      loop  dis_char        ;do next character
C
DOB4 BD 000F        C      mov   bp,15d          ;set error exit flag
DOB7 EB 2B 90       C      jmp   h_exit          ;jmp to exit%
C
DOBA                C      wins_usable:
DOBA A0 0075 R      C      mov   al,byte ptr ds:ram_drv_cnt ;hold num of drives%
DOBD 1E             C      push  ds
DOBE 0E             C      push  cs
DOBF 1F             C      pop   ds          ;ds for string prints%
DOC0 FC             C      cld
DOC1 B3 01          C      mov   bl,1          ;choose foregnd color%
DOC3 0A C0          C      or    al,al          ;set flags%
DOC5 75 09          C      jnz   nzdrvs         ;cont if 0 drives%
DOC7 B9 000F        C      mov   cx,habss        ;msg size%
DOCA BE D067 R      C      mov   si,offset habs   ;msg ptr%
DOCD EB 0D 90       C      jmp   hmsg
DOD0                C      nzdrvs:
DOD0 04 30          C      add   al,'0'          ; HEX drives = ASCII number%
DOD2 B4 0E          C      mov   ah,bc_v_w       ;video cmd tty%
DOD4 CD 10          C      int   ivn_dis_char    ;print number of drives%
DOD6 B9 0009        C      mov   cx,hgoods        ;msg size%
DOD9 BE D05E R      C      mov   si,offset hgood  ;msg ptr%
DODC                C      hmsg:
DODC AC             C      lodsb          ;al=ds:si si++ %
DODD B4 0E          C      mov   ah,bc_v_w       ;video cmd,tty %
DODF CD 10          C      int   ivn_dis_char    ;print char%
DOE1 E2 F9          C      loop  hmsg           ;print string%
DOE3 1F             C      pop   ds          ;restore DS %
DOE4                C      h_exit:
C
DOE4 FA             C      cli          ;disable interrupts
DOE5 E4 21          C      in    al,p_int+int_ocw1 ;read interrupt mask register
DOE7 0C 01          C      or    al,int_ocw1_m0    ;disable interrupts on channel 0
DOE9 E6 21          C      out   p_int+int_ocw1,al ;set interrupt mask register
C                  ;This stops the timer.
DOEB FB             C      sti          ;enable interrupts
DOEC C3             C      ret          ;exit to system bios initialize
C ;-----
DOED                C      bios_install  endp
C ;      page +
C ;=====
C ;
C ; Call:   int 19h
C ;
C ; Boot the operating system from diskette A or from the first usable
C ; winchester drive.
C ;
C ; Entry:  No entry parameters.
C ;
C ; Normal Exit:  Jump to the start of the bootstrap sector read from disk.
C ;
C ; Error Exit:  int 18h -- ROM resident BASIC entered.
C ;      The ROM resident BASIC is entered if the system can't be booted.

```

```

C ;
C ;       Attempt to reset drive 0 (a diskette) and then to read sector 1,
C ; cylinder 0, head 0.  If this read is successful, then jump to the start
C ; of this sector in memory.  Retry the preceding three times.
C ;
C ;       Beginning with drive 80h (the first winchester) and ending with
C ; drive 87h (the last possible winchester) do the following:  Attempt to
C ; reset the drive and then to read sector 1, cylinder 0, head 0.  If this
C ; read is successful and the last word of this sector contains AA55h, then
C ; jump to the start of this sector in memory.
C ;
C ;       Enter the ROM resident BASIC by executing int 18h.
C
DOED          C i19_boot_sys:
DOED 33 C0    C     xor     ax,ax             ;zero
DOEF 8E D8    C     mov     ds,ax          ;establish pointer to intvec segment
DOF1 8E C0    C     mov     es,ax          ;establish pointer for disk buffer
C             C     assume ds:intvec,es:intvec
C
C             C ; Install pointers to the default diskette parameter table and the
C             C ; default winchester parameter table.
C
DOF3 FA      C     cli                     ;disable interrupts
DOF4 C7 06 0078 R CF29 R C     mov     word ptr iv_p_tbl dett,offset p_tbl dett ;20-bit pointer to diskette
DOFA 8C 0E 007A R C     mov     word ptr iv_p_tbl dett+2,cs      ;parameter table
DOFE C7 06 0104 R CF34 R C     mov     word ptr iv_p_tbl win,offset p_tbl win   ;20-bit pointer to winchester
D104 8C 0E 0106 R C     mov     word ptr iv_p_tbl win+2,cs      ;parameter table
D108 FB      C     sti                     ;enable interrupts
C
C             C ; Attempt to boot from diskette 0 up to 3 times.  A time-out error
C             C ; immediately ends the attempt to boot from diskette.
C
D109 B9 0003 C     mov     cx,3             ;number of times to retry diskette
D10C 33 D2    C     xor     dx,dx            ;zero drive & head numbers
C
D10E          C dett_boot:
D10E B8 0000 C     mov     ax,bc_reset*256        ;(AH) = bios call #, (AL) = block count
D111 CD 40    C     int     ivn_bc_dette          ;call diskette bios
D113 72 0F    C     jc     dett_boot_nxt        ;jump if error
C
D115 BB 7C00 R C     mov     bx,offset iv_boot_buf   ;address of sector buffer
D118 B8 0201 C     mov     ax,bc_rd*256+1        ;read one sector
D11B 51      C     push    cx                 ;save retry count
D11C B9 0001 C     mov     cx,1                 ;sector number
D11F CD 40    C     int     ivn_bc_dette          ;call diskette bios
D121 59      C     pop     cx                 ;restore retry count
D122 73 34    C     jnc    boot_succ           ;jump if no error
C
D124          C dett_boot_nxt:
D124 80 FC 80 C     cmp     ah,ec_time            ;time-out?
D127 74 02    C     je     dett_boot_end        ;jump if yes.
D129 E2 E3    C     loop   dett_boot           ;jump if try again
C
D12B          C dett_boot_end:
D12B B8 0000 C     mov     ax,bc_reset*256        ;(AH) = bios call #, (AL) = block count

```

```

D12E CD 40          C      int    ivn_bc_dette      ;call diskette bios
C                                     ;ignore error if any
C
C      ; Boot from the first possible winchester drive. Only one attempt is made
C      ; to boot from each drive. A valid boot sector must have AA55 in its last
C      ; word.
C
D130 B2 80          C      mov    dl,dnwin          ;least significant winchester drive #
D132 B9 0008        C      mov    cx,drv_total      ;max. number of winchester drives
C
D135                C win_boot:
D135 B4 00          C      mov    ah,bc_reset      ;bios command code
D137 CD 13          C      int    ivn_bc           ;call disk bios
D139 72 17          C      jc     win_boot_nxt     ;jump if error
C
D13B BB 7C00 R      C      mov    bx,offset iv_boot_buf ;address of sector buffer
D13E B8 0201        C      mov    ax,bc_rd*256+1    ;read one sector
D141 51             C      push   cx               ;save drive count
D142 B9 0001        C      mov    cx,1              ;sector number
D145 CD 13          C      int    ivn_bc           ;call disk bios
D147 59             C      pop    cx               ;restore drive count
D148 72 08          C      jc     win_boot_nxt     ;jump if error
C
D14A 81 3E 7DFE R AA55 C      cmp    word ptr iv_boot_buf+sec_size-2,0aa55h ;valid boot sector?
D150 74 06          C      je     boot_succ        ;jump if yes.
C
D152                C win_boot_nxt:
D152 FE C2          C      inc    dl               ;update drive number
D154 E2 DF          C      loop  win_boot         ;jump if more drives to try
C
D156 CD 18          C      int    ivn_basic        ;can't boot from disk, enter
C                                     ;ROM resident BASIC
C      ;-----
C
D158                C boot_succ:
C      ;      jmp    iv_boot_buf ;jump to boot sector read from disk
D158 2E: FF 2E D15D R C      jmp    dword ptr cs:[brec] ;jump to boot sector read from disk%
D15D                C brec:
D15D 7C00 R          C      dw    offset iv_boot_buf ;%
D15F 0000           C      dw    0000              ;%
C
C      ;=====
C      ;
C      ; Call:      int    0dh
C      ;           return
C      ;
C      ; Winchester interrupt handler
C      ;
C      ; Entry:   No entry parameters
C      ;
C      ; Exit:    No registers changed.
C      ;
C
D161                C id_ih:

```

```
D161 50          C      push   ax                ;save register
C
C
D162 80 0B      C      mov    al,0bh                ; set up to
D164 E6 20      C      out   p_int+int_w_ocw2,al    ; read the in service register of 8259
D166 90         C      nop                       ; seperate by nop
D167 E4 20      C      in    al,p_int+int_w_ocw2    ; read in service register
D169 A8 20      C      test   al,20h                ; if hard disk not in service zf=1
D16B 74 0E      C      jz    bogus                  ; if zf=1 then bogus interrupt(exception)
C
C
D16D 80 20      C      mov    al,int_ocw2_eoi        ;end of interrupt bit
D16F E6 20      C      out   p_int+int_w_ocw2,al    ;end of interrupt to 8259
C
D171 80 07      C      mov    al,dma_mask_b_s or dma_mask_b_3 ;set channel 3 mask bit
D173 E6 0A      C      out   p_dma+dma_w_mask_b,al  ;set bit to disable channel 3
C
D175 E4 21      C      in    al,p_int+int_ocw1      ;read interrupt mask
D177 0C 20      C      or    al,int_ocw1_m5         ;set mask register bit to disable
D179 E6 21      C      out   p_int+int_ocw1,al      ;channel 5 which is used by the wx2
D17B           C      bogus:
D17B 58         C      pop    ax                    ;restore register
D17C CF         C      iret                       ;long return & restore flags
C
C ;=====
C ;
C ; Call:      int    13h
C ;           return
C ;
C ; Function call to disk BIOS.
C ;
C ; Entry:      (AH) = Command code
C ;           (AL) = Sector count or interleave factor
C ;           (ES:BX) = Address of buffer
C ;           (CL) bits 0 - 5 = Sector number: 1, 2, 3, . . .
C ;           (CL) bits 6, 7 = Bits 8, 9 of cylinder number
C ;           (CH) = Bits 0 - 7 of cylinder number
C ;           (DL) = Drive number. 0 - 7f are diskettes. 80h and up are winchesters
C ;           (DH) = Head number
C ;
C ; Exit:  (CF) = 0 --> no error
C ;       1 --> error
C ;       (AH) = Completion code. 0 --> no error
C ;       (AL) = Error burst length if (AH) = 11h. Else, changed.
C ;
C ; Read parameters:
C ;       (DL) = Number of usable drives at install time
C ;       (DH) = Maximum head number
C ;       (CL) Bits 0 - 5 = 17d --- Maximum sector number
C ;       (CL) bits 6, 7 = Bits 8, 9 of Maximum cylinder number
C ;       (CH) = Bits 0 - 7 of maximum cylinder number
C
D17D           C      i13_inh proc far
C
C      assume ds:nothing,es:nothing
C
D17D FB         C      sti                          ;enable interrupts
```



```

C
D17E 80 FA 80      C      cmp     dl,dnwin      ;winchester drive number?
D181 73 05        C      jae     win_bc         ;jump if yes.
C
D183 CD 40        C      int     ivn_bc_dette ;call diskette bios
D185 CA 0002      C      ret     2           ;exit from call to disk bios
C ;-----
C
C      ; Target drive is a winchester.
C
D188              C win_bc:
D188 80 FC 00      C      cmp     ah,bc_reset ;reset?
D18B 75 04        C      jne     no_reset    ;jump if not reset
C
C                                     ;diskette bios ignores (DL) on reset call
D18D CD 40        C      int     ivn_bc_dette ;reset diskette controller
C                                     ;ignore error if any
D18F B4 00        C      mov     ah,bc_reset ;reset command code
D191              C no_reset:
C
D191 53           C      push    bx          ;save registers to be preserved & used
D192 51           C      push    cx
D193 52           C      push    dx
D194 55           C      push    bp
D195 57           C      push    di
D196 56           C      push    si
D197 1E           C      push    ds
D198 06           C      push    es
D199 8B EC        C      mov     bp,sp       ;initialize activation record pointer
C
C      assume ds:wdrum
D19B 50           C      push    ax          ;temp save
D19C B8 0040      C      mov     ax,W2RAM    ;segment address
D19F 8E D8        C      mov     ds,ax       ;make ram reachable
D1A1 58           C      pop     ax          ;restore ax
C
D1A2 80 FC 15      C      cmp     ah,bc_dasd   ;command code in range? (new cmd)%
D1A5 75 05        C      jnz     win_cont    ; cont if not equal%
D1A7 B4 03        C      mov     ah,3         ; hard disk parm%
D1A9 EB 2E 90      C      jmp     hd_quit     ; avoid everything%
D1AC              C win_cont:
D1AC 77 05        C      ja     bc_bad      ;jump if out of range
D1AE 80 FA 88      C      cmp     dl,dnwin+drv_total ;drive number in range?
D1B1 72 04        C      jb     drvn_ok     ;jump if in range.
D1B3              C bc_bad:
D1B3 B4 01        C      mov     ah,ec_bc   ;completion code
D1B5 EB 03        C      jmp     short cmd_done
C ;-----
C
D1B7              C drvn_ok:
D1B7 E8 D1E4 R     C      call   command_br  ;branch through command table
D1BA              C cmd_done:
D1BA 50           C      push    ax          ;save output parameter
D1BB 88 26 0074 R C      mov     ram_cc,ah   ;save completion code
C

```

```

D1BF E8 D51E R      C      call    wx2_msk          ;calculate port address
D1C2 B0 FC          C      mov     al,not (wx2_msk_dma or wx2_msk_int)
D1C4 EE            C      out    dx,al           ;disable dma & interrupts by wx2
D1C5 B0 07          C      mov     al,dma_mask_b_3 or dma_mask_b_s
D1C7 E6 0A          C      out    p_dma+dma_w_mask_b,al   ;set bit 3 to disable channel 3
D1C9 FA            C      cli                    ;disable interrupts
D1CA E4 21          C      in     al,p_int+int_ocw1       ;read mask register
D1CC 0C 20          C      or     al,int_ocw1_m5         ;set bit 5 to disable channel 5
D1CE E6 21          C      out    p_int+int_ocw1,al      ;set mask register
D1D0 FB            C      sti                    ;enable interrupts
C
D1D1 80 C4 FF          C      add    ah,0ffh           ;set CF to indicate error/no error
C                                     ;;; PCNET and Bernoulli fix
D1D4 B0 01          C      mov     al,dmalong          ;;; disable dmacc
D1D6 E6 63          C      out    p_dmacc,al          ;;; again
D1D8 58            C      pop    ax                ;restore output parameter
C
D1D9              C  hd_quit:                  ;%
D1D9 07            C      pop    es                ;restore registers
D1DA 1F            C      pop    ds
D1DB 5E            C      pop    si
D1DC 5F            C      pop    di
D1DD 5D            C      pop    bp
D1DE 5A            C      pop    dx
D1DF 59            C      pop    cx
D1E0 5B            C      pop    bx
D1E1 CA 0002        C      ret     2                ;exit & throw away flags
D1E4              C  i13_ih endp
C
C  ;=====
C  ;
C  ; Call:          CALL  command_br
C  ;               return
C  ;
C  ; Branch through the bios command table
C  ;
C  ; Entry:  Same as i13_ih.
C  ;         (BP) = Pointer to activation record described by symbols a1_XXXX
C  ;
C  ; Jump to command specific routine:
C  ;         (DS) = ram
C  ;         (BP) = Pointer to activation record described by symbols a1_XXXX
C  ;         (si) = offset to port base for target controller
C  ;
C  ; Exit:   (AH) = Completion code
C  ;         (DS) = ram
C
D1E4              C  command_br  proc  near
D1E4 A2 0046 R      C      mov     ram_ccb+ccb_blks,al   ;sector count or interleave factor
C
D1E7 FE C9          C      dec    cl                ;controller expects sector to be
C                                     ;numbered 0, 1, 2, . . .
D1E9 89 0E 0044 R  C      mov     word ptr ram_ccb+2,cx ;set cylinder & sector numbers
C                                     ; into command control block
C

```

```

D1ED 8A EA          C      mov     ch,dl          ;drive #
D1EF 80 E5 01      C      and     ch,01           ;mask to bit 0
D1F2 B1 05          C      mov     cl,5             ;shift count
D1F4 D2 E5          C      shl     ch,cl           ;align drive bit for ccb
D1F6 0A EE          C      or      ch,dh           ;combine with head number
D1F8 88 2E 0043 R   C      mov     ram_ccb+1,ch     ;set into command control block
C
C                                     ;Calculate the offset from p_wx2 for drive:
C                                     ; Drive #      Offset
C                                     ; 80, 81       0
C                                     ; 82, 83       4
C                                     ; 84, 85       8
C                                     ; 86, 87      12d
D1FC 80 EA 80      C      sub     dl,dhwin         ;origin to 0
D1FF 81 E2 00FE    C      and     dx,0FEh
D203 D0 E2          C      shl     dl,1
D205 8B F2          C      mov     si,dx           ;save controller port offset
C
C ; *** D version added next 7 lines to set correct step rate ***
D207 50            C      push    ax             ;save command & # sectors
D208 06            C      push    es             ;save buffer address
D209 E8 D4DA R     C      call   subtable         ;determine subtable to use
D20C 26: 8A 47 08  C      mov     al,es:wst_opt[bx] ;option byte from subtable
D210 A2 0047 R     C      mov     ram_ccb+ccb_opt,al ;set into ccb, sets step rate
D213 07            C      pop     es
D214 58            C      pop     ax
C
C ; Table lookups indexed by the function call code
C
D215 8A C4          C      mov     al,ah           ;BIOS command code
D217 BB D254 R     C      mov     bx,offset dc_tbl ;pointer to table of controller
C                                     ;command codes
D21A 2E: D7          C      xlat   cs:dc_tbl       ;translate bios command code
D21C A2 0042 R     C      mov     ram_ccb+ccb_cmd,al ;command code into ccb
C
D21F 8A DC          C      mov     bl,ah           ;bios command code
D221 32 FF          C      xor     bh,bh           ;zero high byte
D223 D0 E3          C      sal     bl,1           ;multiply by 2
D225 2E: FF A7 D22A R C      jmp     cs:br_tbl[bx]    ;branch to command specific routine
C ;-----
C
D22A                C br_tbl      label word
D22A D269 R          C      dw     i13_reset
D22C D312 R          C      dw     i13_cc
D22E D33A R          C      dw     i13_rd
D230 D317 R          C      dw     i13_wr
D232 D399 R          C      dw     dma_no          ;4 --- verify
D234 D399 R          C      dw     dma_no          ;5 --- format track
D236 D399 R          C      dw     dma_no          ;6 --- format bad track
D238 D399 R          C      dw     dma_no          ;7 --- format drive
D23A D2EB R          C      dw     i13_par_rd
D23C D28B R          C      dw     i13_par_wr
D23E D31B R          C      dw     i13_rdl
D240 D32C R          C      dw     i13_wrl
D242 D399 R          C      dw     dma_no          ;12 -- seek

```

```

D244 D269 R      C      dw      i13_reset          ;13 -- same as 0
D246 D330 R      C      dw      i13_buff_rd
D248 D336 R      C      dw      i13_buff_wr
D24A D399 R      C      dw      dma_no           ;16 -- test drive ready
D24C D399 R      C      dw      dma_no           ;17 -- recalibrate
D24E D399 R      C      dw      dma_no           ;18 -- RAM diagnostic
D250 D399 R      C      dw      dma_no           ;19 -- drive diagnostic
D252 D399 R      C      dw      dma_no           ;20 -- controller diagnostic
C
D254            C      dc_tbl      label      byte
D254 0C          C          db      dc_par_set      ;reset sets parameters
D255 00          C          db      0              ;return completion code
D256 08          C          db      dc_rd
D257 0A          C          db      dc_wr
D258 05          C          db      dc_vr
D259 06          C          db      dc_ft
D25A 07          C          db      dc_fbt
D25B 04          C          db      dc_fd
D25C 00          C          db      0              ;read parameters
D25D 0C          C          db      dc_par_set
D25E E5          C          db      dc_rdl
D25F E6          C          db      dc_wrl
D260 0B          C          db      dc_seek
D261 0C          C          db      dc_par_set      ;reset also sets parameters
D262 0E          C          db      dc_buff_rd
D263 0F          C          db      dc_buff_wr
D264 00          C          db      dc_tst_rdy
D265 01          C          db      dc_recal
D266 E0          C          db      dc_diag_ram
D267 E3          C          db      dc_diag_drv
D268 E4          C          db      dc_diag_ctlr
C ;-----
C ;      page
C ; Call:      call   i13_reset
C ;           return
C ;
C ; Purpose:  Reset the target winchester controller.
C ;
C ; Entry:    (si) = Offset to port base for target controller
C ;
C ; Exit:     (ah) = Completion code
C
C
D269            C      i13_reset:
D269 E8 D519 R    C          call  wx2_reset      ;calculate address of reset port to use
D26C EE          C          out   dx,al          ;reset wx2 controller
D26D B9 0584      C          mov   cx,ti_bc_reset  ;loop count for 3 ms on fastest 80286 V.F.
D270            C      re_dly:
D270 E2 FE          C          loop  re_dly          ;wait longer then 1.4 ms reset pulse
D272 B4 0A          C          mov   ah,10          ;msb of loop count
D274            C      re_w:
D274 42            C          inc   dx              ;control port
D275 EE            C          out   dx,al          ;select the controller
D276 4A            C          dec   dx              ;status port
D277 EC            C          in   al,dx          ;read status

```

```

D278 A8 30          C      test    al,wx2_stat_int or wx2_stat_drq
D27A 75 0C          C      jnz     ctlr_missing      ;jump if the controller is missing
D27C 24 0D          C      and     al,wx2_stat_busy or wx2_stat_cd or wx2_stat_req
D27E 34 0D          C      xor     al,wx2_stat_busy or wx2_stat_cd or wx2_stat_req
D280 74 09          C      jz      i13_par_wr        ;jump if reset successful and the
C                                     ;controller is trying to read ccb
D282 E2 F0          C      loop   re_w              ;jump if no rollover
D284 FE CC          C      dec     ah                ;update msb of counter
D286 75 EC          C      jnz     re_w              ;jump if success still possible
C
D288                C      ctlr_missing:
D288 B4 05          C      mov     ah,ec_reset      ;error code
D28A                C      ret_near:
D28A C3              C      ret                       ;exit
C      ;-----
C
C      ; Call:      call    i13_par_wr
C      ;              return
C      ;
C      ; Purpose:   Set parameters
C      ;
C      ; Entry:    (si) = Offset to port base for target controller
C      ;
C      ; Exit:     (ah) = Completion code
C      ; *** As of A3 version, step rate selection is not part of Set Parameters
C      ; *** command. Step rate is moved to CCB from subtable on every command.
C
D28B                C      i13_par_wr:
D28B C6 06 0043 R 00 C      mov     ram_ccb+1,0      ;drive 0 is 1st target
D290 E8 D29A R      C      call   par_wr            ;set parameters for 1 drive
D293 72 F5          C      jc      ret_near        ;jump if error
C
D295 C6 06 0043 R 20 C      mov     ram_ccb+1,ccb_drv_b ;set drive bit
C
D29A                C      par_wr:
D29A B0 FC          C      mov     al,not(wx2_msk_dma or wx2_msk_int) ;no dma & no interrupt
D29C E8 D47A R      C      call   ccb_send         ;send CCB to target controller
D29F 72 E9          C      jc      ret_near        ;jump if error
C
D2A1 E8 D4DA R      C      call   subtable         ;determine subtable to use
C
C      ; Send 8-byte parameter block to the target controller.
C      ; Note that the bytes are sent MSB first, then LSB
C
D2A4 BF 0001        C      mov     di,wst_cyl+1      ;offset to byte to send
D2A7 E8 D2DE R      C      call   send_byte        ;send msb of number of cylinders
C
D2AA BF 0000        C      mov     di,wst_cyl+0      ;LSB of number of cylinders
D2AD E8 D2DE R      C      call   send_byte        ;send 1 byte
C
D2B0 BF 0002        C      mov     di,wst_heads       ;number of heads
D2B3 E8 D2DE R      C      call   send_byte        ;send 1 byte
C
D2B6 BF 0004        C      mov     di,wst_re_wr+1     ;MSB of reduced write current cyl
D2B9 E8 D2DE R      C      call   send_byte        ;send 1 byte

```

```

C
D2BC BF 0003 C      mov    di,wst_re_wr+0      ;LSB of reduced write current cyl
D2BF E8 D2DE R C      call   send_byte          ;send 1 byte
C
D2C2 BF 0006 C      mov    di,wst_wr_pre+1     ;MSB of write precomp. cyl
D2C5 E8 D2DE R C      call   send_byte          ;send 1 byte
C
D2C8 BF 0005 C      mov    di,wst_wr_pre+0     ;LSB of write precomp cyl
D2CB E8 D2DE R C      call   send_byte          ;send 1 byte
C
D2CE BF 0007 C      mov    di,wst_er_bur      ;maximum burst length
D2D1 E8 D2DE R C      call   send_byte          ;send 1 byte
C
D2D4 E8 D4AC R C      call   wx2_cc             ;receive command completion byte
D2D7 72 02 C      jc     par_wr_erx         ;jump if error prevented reception
D2D9 74 02 C      jz     ret_near_1         ;jump if no error on command
C
D2DB C      par_wr_erx:
D2DB B4 07 C      mov    ah,ec_init         ;error code
D2DD C      ret_near_1:
D2DD C3 C      ret                       ;exit
C ;-----
C
D2DE- C      send_byte:
D2DE E8 D4C9 R C      call   wx2_req            ;wait for data request
D2E1 72 05 C      jc     send_err           ;jump if error
D2E3 26: 8A 01 C      mov    al,es:[bx+di]     ;byte to send
D2E6 EE C      out   dx,al              ;send to wx2
D2E7 C3 C      ret                       ;exit
C ;-----
C
D2E8 C      send_err:
D2E8 58 C      pop   ax                  ;throw away near return
D2E9 EB F0 C      jmp   par_wr_erx         ;error exit from write parameters
C
C ;-----
C ;
C ; Call:      call   i13_par_rd
C ;           return
C ;
C ; Read parameters
C ;
C ; Entry:      (si) = offset to port base for target controller
C ;
C ; Exit: (DL) = Number of usable drives at install time
C ;        (DH) = Maximum head number
C ;        (CL) Bits 0 - 5 = 17d --- Maximum sector number
C ;        (CL) bits 6, 7 = Bits 8, 9 of Maximum cylinder number
C ;        (CH) = Bits 0 - 7 of maximum cylinder number
C
D2EB C      i13_par_rd:
D2EB E8 D4DA R C      call   subtable          ;determine subtable to use
D2EE 26: 8B 07 C      mov    ax,es:wst_cyl[bx] ;number of cylinders
D2F1 2D 0002 C      sub   ax,2                ;max. cylinder number
C ;The last cylinder is reserved.

```

```

D2F4 8A E8      C      mov     ch,al           ;temp save
D2F6 D1 E8      C      shr     ax,1             ;align bits 8 & 9 of cylinder #
D2F8 D1 E8      C      shr     ax,1
D2FA 24 C0      C      and     al,11000000b     ;mask to bits 8 & 9 of cylinder #
D2FC 0C 11      C      or      al,17           ;maximum sector number
D2FE 8A E5      C      mov     ah,ch           ;bits 0 - 7 of cylinder #
D300 89 46 0C   C      mov     a1_cx[bp],ax     ;set into activation record
C
D303 26: 8A 67 02 C      mov     ah,es:wst_heads[bx] ;number of heads
D307 FE CC      C      dec     ah             ;maximum head number
D309 A0 0075 R   C      mov     al,ram_drv_cnt    ;number of usable drives at install
D30C 89 46 0A   C      mov     a1_dx[bp],ax     ;set into activation record
D30F           C      ret_no_err:
D30F B4 00      C      mov     ah,ec_no_err     ;no error
D311 C3           C      ret                    ;exit
C
C ;=====
C ;
C ; Call:      call   i13_cc
C ;           return
C ;
C ; Return the completion code for the previous command to any winchester drive.
C ;
C ; Entry: (ram_cc) = Completion code for last command to any winchester
C ;
C ; Exit: (al) = completion code for last command
C
C i13_cc:
D312           C
D312 A0 0074 R   C      mov     al,ram_cc      ;completion code for last command
D315 EB F8      C      jmp     ret_no_err       ;no error exit
C ;           page
C ;=====
C ;
C ; Call:      call   i13_wr
C ;           return
C ;
C ; Purpose: Write sectors
C ;
C ; Entry: (si) = Offset to port base for target controller
C ;
C ; Exit: (ah) = Completion code
C
C i13_wr:
D317           C
D317 B0 0B      C      mov     al,dma_mode_rd ;byte to set dma chip to read from host
D319 EB 21      C      jmp     short io_norm
C ;=====
C ;
C ; Call:      call   i13_rdl
C ;           return
C ;
C ; Purpose: Read sector and ECC bytes
C ;
C ; Entry: (si) = Offset to port base for target controller
C ;
C ; Exit: (ah) = Completion code

```

```
C
D31B          C i13_rdl:
D31B B0 07    C      mov     al,dma_mode_wr      ;byte to set dma chip to write to host
D31D          C io_long:
C          C IF DMACC
D31D 50       C      push    ax
D31E B0 01    C      mov     al,dmalong          ;long sector dma%
D320 E6 63    C      out     p_dmacc,al         ;tell dma acc that this is a long sector%
D322 58       C      pop     ax
C          C ENDIF
D323 8A 16 0046 R C      mov     dl,ram_ccb+ccb_blks ;number of blocks to transfer
D327 BF 0204   C      mov     di,sec_size+4       ;number of bytes per sector + 4 ECC
D32A EB 1D    C      jmp     short dma_start
C          C ;=====
C          C ; Call:      call    i13_wrl
C          C ;           return
C          C ;
C          C ; Purpose:  Write sector and ECC bytes
C          C ;
C          C ; Entry:    (si) = Offset to port base for target controller
C          C ;
C          C ; Exit:     (ah) = Completion code
C          C
D32C          C i13_wrl:
D32C B0 0B    C      mov     al,dma_mode_rd      ;byte to set dma chip to read from host
D32E EB ED    C      jmp     short io_long
C          C ;=====
C          C ; Call:      call    i13_buff_rd
C          C ;           return
C          C ;
C          C ; Purpose:  Read sector buffer
C          C ;
C          C ; Entry:    (si) = Offset to port base for target controller
C          C ;
C          C ; Exit:     (ah) = Completion code
C          C
D330          C i13_buff_rd:
D330 B0 07    C      mov     al,dma_mode_wr      ;byte to set dma chip to write to host
D332          C buff_io:
D332 B2 01    C      mov     dl,1                ;transfer one block
D334 EB 0A    C      jmp     short sec_size_norm
C          C ;=====
C          C
D336          C i13_buff_wr:
D336 B0 0B    C      mov     al,dma_mode_rd      ;byte to set dma chip to read from host
D338 EB F8    C      jmp     short buff_io
C          C ;=====
C          C ;
C          C ; Call:      call    i13_rd
C          C ;           return
C          C ;
C          C ; Purpose:  Read sectors
C          C ;
```



```

C ; Entry: (si) = Offset to port base for target controller
C ;
C ; Exit: (ah) = Completion code
C
D33A C i13_rd:
D33A B0 07 C mov al,dma_mode_wr ;byte to set dma chip to write to host
D33C C io_norm:
D33C 8A 16 0046 R C mov dl,ram_ccb+ccb_blks ;number of blocks to transfer
D340 C sec_size_norm:
C IF DMACC
D340 50 C push ax
D341 B0 00 C mov al,dmanorm ;norm sector dma%
D343 E6 63 C out p_dmacc,al ;tell dma acc that this is a norm sector%
D345 58 C pop ax
C ENDF
D346 BF 0200 C mov di,sec_size ;numbr of bytes per sector
C
C ; Prepare the DMA chip
C
D349 C dma_start:
D349 FA C cli ;disable interrupts
D34A E6 0B C out p_dma+dma_w_mode,al ;set read or write to host
D34C 90 C nop ; Can't handle back to%
D34D 90 C nop ; back writes to dma controller %
D34E E6 0C C out p_dma+dma_w_byte,al ;clear pointer to register byte
C
C ; Calculate the 20-bit address of the DMA base address
C
D350 8C C0 C mov ax,es ;segment reg for i/o buffer
D352 B1 04 C mov cl,4 ;count for shift
D354 D3 C0 C rol ax,cl ;multiply by 16
D356 8A E8 C mov ch,al ;bits 16 - 19 in bits 0 - 3
D358 24 F0 C and al,0f0h ;clear bits 0 - 3 of address
D35A 03 46 0E C add ax,a1_bx[bp] ;combine 16-bit offset with seg base
D35D 80 D5 00 C adc ch,0 ;add carry into MS bits
D360 E6 06 C out p_dma+dma_w_addr,al ;set bits 0 -7 of dma address
D362 86 E0 C xchg ah,al ;bits 8 - 15 to al
D364 E6 06 C out p_dma+dma_w_addr,al ;set bits 8 - 15 of dma address
D366 86 C5 C xchg al,ch ;bits 16 - 19 of dma address to al
D368 8A CC C mov cl,ah ;bits 0 - 15 of addr in cx now
D36A 24 0F C and al,0fh ;mask to 4 bits
D36C E6 82 C out p_dma_latch,al ;set 4-bit latch for highest address
C ;bits
C
C ; Calculate the number of bytes to be DMAed minus 1.
C
D36E 8B C7 C mov ax,di ;block size in bytes
D370 32 F6 C xor dh,dh ;clear upper byte
D372 F7 E2 C mul dx ;number of bytes to dma: (ax) LSW
C ;(dx) is MSW
D374 2D 0001 C sub ax,1 ;byte offset to last byte to be trans
D377 80 DA 00 C sbb dl,0 ;bits 16 - 23 of byte offset.
C ; and set ZF
D37A E6 07 C out p_dma+dma_w_cnt,al ;set bits 0 - 7 of byte count
D37C 86 C4 C xchg al,ah ;bits 8 - 15 of count to al

```

ROM BIOS Listing

```

D37E E6 07      C      out    p_dma+dma_w_cnt,al      ;set bits 8 - 15 of byte count
C
D380 FB        C      sti                          ;enable interrupts
C
C      ; Check for DMA spanning 64k absolute address boundary
C      ; Are bits 16 - 23 of byte offset 0?
C      ; note: If the block count is 0,
C      ; then (dl) = ff.
D381 75 13      C      jnz    dma_64k                ;jump if dma more than 64kb
D383 86 E0      C      xchg   ah,al                  ;byte offset to last byte transferred
D385 03 C1      C      add    ax,cx                  ;lsw of address of last byte
C      ; transferred.
D387 72 0D      C      jc     dma_64k                ;jump if dma crosses absolute 64k
C      ; boundary
C
C      ; Start winchester controller and DMA controller.
C
D389 B0 03      C      mov    al,wx2_msk_dma or wx2_msk_int ;dma & interrupt enable
D38B E8 D47A R   C      call   ccb_send                ;send ccb to wx2
D38E 72 08      C      jc     ret_near_3              ;jump if error
C
D390 B0 03      C      mov    al,dma_mask_b_3        ;clear bit 3 of mask register
D392 E6 0A      C      out    p_dma+dma_w_mask_b,al   ;enable channel 3
D394 EB 0A      C      jmp    short wx2_wait
C      ;-----
C
D396           C dma_64k:
D396 B4 09      C      mov    ah,ec_dma_64k          ;completion error code
D398           C ret_near_3:
D398 C3          C      ret                          ;exit
C      ; page
C      ;=====
C
C ; Call:      call   i13_dma_no
C ;           return
C ;
C ; Purpose:   Execute a command involving the winchester controller
C ;           but not involving DMA.
C ;
C ; Entry:    (si) = Offset to port base for target controller
C ;
C ; Exit:     (ah) = Completion code
C
D399           C dma_no:
D399 B0 02      C      mov    al,wx2_msk_int          ;enable interrupt by wx2
D39B E8 D47A R   C      call   ccb_send                ;send ccb to wx2
D39E 72 F8      C      jc     ret_near_3              ;jump if error
C
C      ; Wait for the winchester interrupt in a polling loop
C
D3A0           C wx2_wait:
D3A0 FA        C      cli                          ;disable interrupts
D3A1 E4 21      C      in    al,p_int+int_ocw1        ;read interrupt mask
D3A3 24 DF      C      and   al,not int_ocw1_m5       ;enable channel 5 interrupts
D3A5 E6 21      C      out    p_int+int_ocw1,al       ;set interrupt mask

```

```

D3A7 FB          C      sti                ;enable interrupts
C
D3A8 53          C      push   bx                ;%
D3A9 51          C      push   cx                ;save these registers %
D3AA 06          C      push   es                ;just in case %
C
D3AB E8 D4DA R   C      call   subtable           ; which parameter table is being used%
D3AE 33 C0       C      xor    ax,ax              ; clear ax%
D3B0 A0 0042 R   C      mov    al,ram_ccb+ccb_cmd ; which command was it?%
D3B3 3D 00E3     C      cmp    ax,dc_diag_drv      ; was it a drive diagnostic command?%
D3B6 74 0C       C      jz    drv_diag           ; if so jump %
C
D3B8 3D 0004     C      cmp    ax,dc_fd          ; was it a format drive command?%
D3BB 74 0E       C      jz    drv_format        ; if so jump %
C
D3BD 26: 8A 47 09 C      mov    al,es:[bx+wst_sto] ; if we're here so it must be %
C ; a standard command%
D3C1 EB 0C 90     C      jmp    continue         ; continue with this dirty deed%
C
D3C4             C      drv_diag:                ;%
D3C4 26: 8A 47 0B C      mov    al,es:[bx+wst_ddto] ; get the drive diagnostic time out value%
D3C8 EB 05 90     C      jmp    continue         ; get on with it%
D3CB             C      drv_format:              ;%
D3CB 26: 8A 47 0A C      mov    al,es:[bx+wst_fto]   ; get the format drive time out value%
C
D3CF             C      continue:                ;%
D3CF 33 DB       C      xor    bx,bx              ; clear bx%
D3D1 8A D8       C      mov    bl,al              ; put time out value into bx%
D3D3 B9 0002     C      mov    cx,2               ;;;;changed%%%%
D3D6 D3 E3       C      shl    bx,c1              ; we are soooo fast we should%
C ; make this number bigger, MUCH%
C
C ;      mov    ram_cc,0         ; clear ram_cc %
D3D8             C      wait_more:                ;%
D3D8 33 C9       C      xor    cx,cx              ;clear cx%
D3DA E8 D519 R   C      call   wx2_stat           ;calc. address of target port
D3DD             C      int_wait:                ;%
D3DD EC         C      in    al,dx              ;read wx2 status
D3DE A8 20       C      test   al,wx2_stat_int     ;interrupt?
D3E0 75 0F       C      jnz   wx2_int            ;jump if yes.
D3E2 A8 08       C      test   al,wx2_stat_busy    ;busy?
D3E4 74 05       C      jz    ret_time_k         ;jump if not busy.%
C
D3E6             C      fall:                    ;%
D3E6 E2 F5       C      loop  int_wait           ;%
D3E8 4B          C      dec   bx                  ;%
D3E9 75 ED       C      jnz   wait_more          ;%
D3EB             C      ret_time_k:              ;%
D3EB 07          C      pop   es                  ;%
D3EC 59          C      pop   cx                  ;%
D3ED 5B          C      pop   bx                  ;%
D3EE             C      ret_time_j:              ;%
D3EE E9 D48B R   C      jmp    ret_time           ;return time-out error
C ;-----
C

```

```
D3F1          C wx2_int:
D3F1 07      C      pop     es             ;%
D3F2 59      C      pop     cx             ;%
D3F3 5B      C      pop     bx             ;%
C
C
D3F4 E8 D51E R C      call    wx2_msk             ;calc. address of target msk
D3F7 B0 FC      C      mov     al,not (wx2_msk_dma or wx2_msk_int) ;disable dma &
D3F9 EE      C      out     dx,al             ;interrupts
D3FA E8 D4AC R C      call    wx2_cc             ;read completion code from wx2
D3FD 72 EF      C      jc     ret_time_j             ;jump if error prevented cc reception
D3FF 74 97      C      jz     ret_near_3            ;jump if no error
C
C      ; The controller reported an error. Read status to obtain more error
C      ; information.
C
D401 C6 06 0042 R 03 C      mov     ram_ccb+ccb_cmd,dc_stat_rd ;Read status command
D406 B0 FC      C      mov     al,not (wx2_msk_dma or wx2_msk_int) ;no dma or interrupt
C      ;by wx2
D408 E8 D47A R C      call    ccb_send             ;send command to wx2
D40B 72 6A      C      jc     stat_err             ;jmp if error
C
C      ; Read the 4-bytes of error status from the controller.
C
D40D BF 0042 R C      mov     di,offset ram_stat ;offset to 4-byte area for status
D410 8C D8      C      mov     ax,ds             ;ram segment
D412 8E C0      C      mov     es,ax             ;set es for stosb instruction
D414 B9 0004      C      mov     cx,4             ;number of status bytes
D417 FC      C      cld                     ;clear direction --> inc di
C
D418          C stat_loop:
D418 E8 D4C9 R C      call    wx2_req             ;wait for request
D41B 72 5A      C      jc     stat_err             ;jump if error
D41D EC      C      in     al,dx             ;read status byte
D41E AA      C      stosb                    ;save status byte
D41F E2 F7      C      loop   stat_loop          ;jump if more status to read
C
D421 E8 D4AC R C      call    wx2_cc             ;receive command completion byte
D424 72 51      C      jc     stat_err             ;jump if error prevented reception
D426 75 4F      C      jnz    stat_err             ;jump if error on command
C
C      ; Translate the controller error code into a BIOS error code
C
D428 8A 2E 0042 R C      mov     ch,ram_stat ;error code
D42C 8A DD      C      mov     bl,ch
D42E 81 E3 0030 C      and     bx,30h             ;mask to error type field
D432 B1 03      C      mov     cl,3             ;shift count
D434 D2 EB      C      shr     bl,cl             ;type times 2
D436 8A E5      C      mov     ah,ch             ;error code
D438 80 E4 0F      C      and     ah,0fh           ;mask to error in type field
D43B 2E: 3A A7 D523 R C      cmp     ah,cs:[bx]+offset er_master_tbl ;number of error codes in type
D440 73 32      C      jnc     undef             ;jump if out of range
D442 43      C      inc     bx             ;update table address
D443 2E: 8A 9F D523 R C      mov     bl,cs:[bx]+offset er_master_tbl ;offset to subtable
D448 02 DC      C      add     bl,ah             ;error code subfield
```

```

D44A 2E: 8A A7 D523 R   C      mov     ah,cs:[bx]+offset er_master_tbl ;completion code
D44F 80 FD 18          C      cmp     ch,erc_corr      ;correctable ECC error?
D452 75 22            C      jne     ret_near_5      ;jump if no
C
C      ; ECC error corrected ---> Read the error burst length
C
D454 8A FC            C      mov     bh,ah          ;save completion code
D456 C6 06 0042 R 0D   C      mov     ram_ccb+ccb_cmd,dc_ecc_rd      ;read error burst length
D45B B0 FC            C      mov     al,not (wx2_msk_dma or wx2_msk_int) ;no dma or interrupt
C
C      ;by wx2
D45D E8 D47A R        C      call    ccb_send        ;send command to wx2
D460 72 15            C      jc     stat_err        ;jmp if error
D462 E8 D4C9 R        C      call    wx2_req        ;wait for request
D465 72 10            C      jc     stat_err        ;jump if error
D467 EC              C      in     al,dx          ;read error burst length
D468 8A D8            C      mov     bl,al         ;save
D46A E8 D4AC R        C      call    wx2_cc        ;receive completion byte
D46D 72 08            C      jc     stat_err        ;jump if error prevented reception
D46F 75 06            C      jnz    stat_err        ;jump if error on command
D471 8B C3            C      mov     ax,bx         ;completion code & error burst length
D473 C3                C      ret                    ;exit
C ;-----
C
D474                C undef:
D474 B4 BB            C      mov     ah,ec_undef    ;undefined error code
D476                C ret_near_5:
D476 C3                C      ret                    ;exit
C
C ;-----
C
D477                C stat_err:
D477 B4 FF            C      mov     ah,ec_stat    ;read status failed
D479 C3                C      ret                    ;exit
C
C ;=====
C ;
C ; Call:      call    ccb_send
C ;           return
C ;
C ; Send the Command Control Block to the wx2 controller
C ;
C ; Entry: (al) = byte to set into wx2 mask port
C ;       (ram_ccb + 0, 1, 2, 3, 4, 5) = ccb to send to wx2
C ;       (si) = offset to port base for target controller
C ;
C ; Exit: (CF) = 0 --> no error. 1 --> error
C ;       (AH) = Error completion code, if error
C ; Changed: ax, cx, dx, di
C ; Unchanged: bx, bp, si
C
D47A                C ccb_send:
D47A E8 D51E R        C      call    wx2_msk        ;calculate address of target mask port
D47D EE              C      out    dx,al         ;set wx2 mask port
D47E 4A              C      dec    dx            ;address of target select port
D47F EE              C      out    dx,al         ;select wx2

```

```
D480 4A          C      dec    dx          ;address of target status port
D481 B9 0258    C      mov    cx,600       ;loop counter(value doubled)%
C ;          mov    cx,300       ;loop counter
C
D484          C busy_wait:
D484 EC          C      in     al,dx        ;read wx2 status
D485 A8 08      C      test   al,wx2_stat_busy ;busy?
D487 75 06      C      jnz   busy          ;jump if yes.
D489 E2 F9      C      loop  busy_wait     ;loop if success still possible
C
D48B          C ret_time:
D48B B4 80      C      mov    ah,ec_time    ;completion code
D48D F9          C      stc                ;set error flag
D48E          C ret_near_2:
D48E C3          C      ret                ;exit
C ;-----
C
D48F          C busy:
D48F BF 0042 R   C      mov    di,offset ram_ccb ;pointer to ccb
D492 B9 0006    C      mov    cx,6           ;number of bytes in a ccb
D495 FC          C      cld                ;clear direction --> inc si
C
D496          C ccb_byte:
D496 E8 D4C9 R   C      call   wx2_req        ;wait for data request
D499 72 F3      C      jc     ret_near_2     ;jump if error
C
D49B 24 0E      C      and   al,wx2_stat_busy or wx2_stat_cd or wx2_stat_io
D49D 34 0C      C      xor   al,wx2_stat_busy or wx2_stat_cd ;command byte requested?
D49F 75 EA      C      jnz   ret_time        ;jump if no.
C
D4A1 87 FE      C      xchg  di,si          ;ccb pointer to si & save port offset
D4A3 AC          C      lodsb                ;ccb byte to al
D4A4 87 FE      C      xchg  di,si          ;restore port offset to si & save ccb
D4A6 EE          C      out   dx,al          ;send to wx2
D4A7 E2 ED      C      loop  ccb_byte        ;loop if more ccb bytes
C
D4A9 E9 D30F R   C      jmp    ret_no_err
C ;=====
C ;
C ; Call:      call   wx2_cc
C ;           return
C ;
C ; Read command completion byte
C ;
C ; Entry: (si) = offset to port base for target controller
C ;
C ; Exit: (cf) = 0 --> no error in reading completion byte
C ;       1 --> error, completion byte not read
C ;       (zf) = Valid only if (cf) = 0
C ;       1 --> error bit in completion byte not set
C ;       0 --> error bit in completion byte set
C ;       (ah) = 0
C ; changed: ax, dx
C ; unchanged: bx, cx, bp, si, di
C
```

```

D4AC          C wx2_cc:
D4AC E8 D4C9 R C      call  wx2_req          ;wait for request
D4AF B4 00    C      mov   ah,ec_no_err      ;no error completion code
D4B1 72 15    C      jc    ret_near_4        ;jump if error
D4B3 24 0E    C      and   al,wx2_stat_busy or wx2_stat_cd or wx2_stat_io
D4B5 3C 0E    C      cmp   al,wx2_stat_busy or wx2_stat_cd or wx2_stat_io
D4B7 75 0E    C      jnz   ret_stc          ;jump if not cc byte
D4B9 EC      C      in    al,dx            ;read command completion byte
C
D4BA 8A E0    C      mov   ah,al            ;save completion byte
D4BC 42      C      inc   dx              ;address of status port
C
D4BD          C cc_busy:
D4BD EC      C      in    al,dx            ;read status
D4BE 24 08    C      and   al,wx2_stat_busy    ;busy?
D4C0 75 FB    C      jnz   cc_busy          ;jump if busy
C
D4C2 86 C4    C      xchg  al,ah            ;clear ah & obtain completion byte
D4C4 A8 02    C      test  al,cc_er          ;error?
D4C6 C3      C      ret                    ;exit
C ;-----
C
D4C7          C ret_stc:
D4C7 F9      C      stc                    ;set carry
D4C8          C ret_near_4:
D4C8 C3      C      ret                    ;exit
C
C ;=====
C ;
C ; Call:      call  wx2_req
C ;           return
C ;
C ; Wait for request signal from target WX2 controller
C ;
C ; Entry:    (SI) = Offset to port base for target WX2 controller
C ;
C ; Exit:     (CF) = 0 --> no error. 1 --> error
C ;          (al) = Last status read
C ;          (ah) = completion code if error
C ;          (DX) = Address of port wx2_r/w_data for target wx2 controller
C ; unchanged: bx, cx, bp, si, di
C
D4C9          C wx2_req:
D4C9 E8 D519 R C      call  wx2_stat          ;calc. status port of target controller
C
D4CC          C req_1:
D4CC EC      C      in    al,dx            ;read status
D4CD A8 01    C      test  al,wx2_stat_req    ;data request? (& clear CF)
D4CF 75 07    C      jnz   req_succ          ;jump if yes.
D4D1 A8 08    C      test  al,wx2_stat_busy    ;busy?
D4D3 75 F7    C      jnz   req_1            ;jump if yes
C
D4D5 F9      C      stc                    ;set carry to indicate error
D4D6 B4 80    C      mov   ah,ec_time        ;error code
C

```

```
C
D4D8      C req_succ:
D4D8  4A      C      dec    dx                ;port address for data i/o to target
C                ;dec does not affect CF
D4D9  C3      C      ret                    ;exit
C
C ;=====
C ;
C ; Call:      call  subtable
C ;           return
C ;
C ; Calculate the address of the subtable to be used for the target drive
C ;
C ; Entry: (si) = Offset to port for target wx2
C ;       (ram_ccb+1) and ccb_drv_b = Drive bit
C ;
C ; Exit: (ES:BX) = Address of parameter subtable for target drive
C
D4DA      C subtable:
D4DA  33 C0    C      xor    ax,ax            ;zero
D4DC  8E C0    C      mov    es,ax            ;set pointer to segment intvec
C          assume es:intvec
C
D4DE  26: C4 1E 0104 R C      les    bx,iv_p_tbl_win    ;20-bit pointer to winchester parm. tbl
C
D4E3  E8 D514 R C      call   wx2_config           ;calculate port address
D4E6  EC      C      in     al,dx              ;read configuration switches
D4E7  F6 06 0043 R 20 C      test   ram_ccb+1,ccb_drv_b  ;Is drive bit set?
D4EC  75 04    C      jnz   drv_1              ;jump if yes.
D4EE  D0 E8    C      shr   al,1              ;align switches for drive 0
D4F0  D0 E8    C      shr   al,1
D4F2      C drv_1:
D4F2  25 0003  C      and   ax,011b           ;mask to switches for target drive
C
D4F5  8A E0    C      mov   ah,al            ;store HD switch info%
D4F7  E4 67    C      in   al,sw_b           ;auxiliary HD switch info%
D4F9  F6 06 0043 R 20 C      test   ram_ccb+1,ccb_drv_b  ;Is drive bit set?%
D4FE  75 07    C      jnz   mdrv_1           ;jump if yes%
D500  24 08    C      and   al,08H           ;isolate drive 0 bit%
D502  D0 E8    C      shr   al,1            ;align for offset%
D504  EB 03 90  C      jmp   mseltbl
D507      C mdrv_1:
D507  24 04    C      and   al,04H           ;isolate drive 1 bit%
D509      C mseltbl:
D509  0A C4    C      or    ah,ah            ;retrieve full offset%
D50B  32 E4    C      xor   ah,ah            ;clear upper byte%
C
D50D  B1 04    C      mov   cl,4              ;shift count
D50F  D3 E0    C      shl   ax,cl            ;multiply by 16
D511  03 D8    C      add   bx,ax              ;address of subtable into bx
D513  C3      C      ret                    ;exit
C ;-----
C      assume es:nothing
C
C ; Call:      call  wx2_config
```



```

C ;          return
C ;
C ; Purpose:  Calculate the address of the configuration port for the
C ;          target controller.
C ;
C ; Entry:   (si) = offset to port base for target controller
C ;
C ; Exit:    (dx) = Address of configuration port for target controller
C
D514          C wx2_config:
D514 8D 94 0322 C      lea    dx,p_wx2+wx2_r_config[si] ;load effective address of port
D518 C3        C      ret                                ;exit
C ;-----
D519          C wx2_stat:
D519          C wx2_reset:
D519 8D 94 0321 C      lea    dx,p_wx2+wx2_w_reset[si] ;load effective address of port
D51D C3        C      ret
C ;-----
C
D51E          C wx2_msk:
D51E 8D 94 0323 C      lea    dx,p_wx2+wx2_w_msk[si] ;load effective address of port
D522 C3        C      ret                                ;exit
C ;-----
C
D523          C command_br    endp
C
D523          C er_master_tbl label byte
D523 09        C      db     t01                ;Number of errors in type class
D524 08        C      db     t0_tbl-er_master_tbl ;byte offset to sub-table
C
D525 0A        C      db     t11                ;Number of errors in type class
D526 11        C      db     t1_tbl-er_master_tbl ;byte offset to sub-table
C
D527 02        C      db     t21                ;Number of errors in type class
D528 1B        C      db     t2_tbl-er_master_tbl ;byte offset to sub-table
C
D529 03        C      db     t31                ;Number of errors in type class
D52A 1D        C      db     t3_tbl-er_master_tbl ;byte offset to sub-table
C
D52B          C t0_tbl:
D52B 00        C      db     ec_no_err          ;no error
D52C 20        C      db     ec_cntlr          ;no index pulse
D52D 40        C      db     ec_seek           ;no seek complete
D52E 20        C      db     ec_cntlr          ;write fault
D52F 80        C      db     ec_time           ;drive not ready
D530 00        C      db     ec_no_err          ;not used
D531 20        C      db     ec_cntlr          ;track 0 not found
D532 00        C      db     ec_no_err          ;not used
D533 40        C      db     ec_seek           ;buffered seek in progress
= 0009        C t01    equ    $-t0_tbl
C
D534          C t1_tbl:
D534 10        C      db     ec_ecc_un          ;ECC error in id field
D535 10        C      db     ec_ecc_un          ;uncorrectable ecc error
D536 02        C      db     ec_addr_mark       ;address mark not found

```

ROM BIOS Listing

```

D537 00      C      db      ec_no_err      ;not used
D538 04      C      db      ec_sec_not_fnd  ;sector not found
D539 40      C      db      ec_seek      ;seek error
D53A 00      C      db      ec_no_err      ;not used
D53B 00      C      db      ec_no_err      ;not used
D53C 11      C      db      ec_ecc_cor      ;ecc error corrected
D53D 0B      C      db      ec_bad_trk      ;bad track
= 000A      C      t11     equ      $-t1_tbl
C
D53E      C      t2_tbl:
D53E 01      C      db      ec_bc      ;invalid command
D53F 02      C      db      ec_addr_mark      ;illegal disk address
= 0002      C      t21     equ      $-t2_tbl
C
D540      C      t3_tbl:
D540 20      C      db      ec_cntlr      ;RAM failure
D541 20      C      db      ec_cntlr      ;ROM checksum error
D542 10      C      db      ec_ecc_un      ;ECC subsystem failure
= 0003      C      t31     equ      $-t3_tbl
C
D543 38 35 2F 30 31 2F  C      db      " 85/01/25"      ;release date
      32 35      C
D54B      C      code     ends
C      end
CEF7      C      code     ends
C      include graph.asm
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      ;          ah, bx, cx, dx intact (set by INT 10 invoker)
C      ;          ah = v_mode
C      ;-----
C
CEF7      C      code     segment public 'ROM'
C      assume  cs:code, ds:data, es:v_ram, ss:nothing
C
D550      C      ORG      0D550H
C      ;Attempt to force a vacant space in the code for the hdu %
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
D550      C  grf_read_dot  proc    near
C
D550  52      C          push  dx          ; save registers
D551  51      C          push  cx
C
D552  50      C          push  ax          ; save crt mode (high byte)
D553  E8 D595 R  C          call  g_addr      ; compute address & mask
D556  8A C4      C          mov   al,ah        ; copy mask
D558  26: 22 05  C          and  al,byte ptr es:[di] ; AND with byte containing pixel
D55B  5A        C          pop   dx          ; dh = crt mode
D55C  FE C1      C          inc  cl          ; prep for wraparound left (B & W)
D55E  80 FE 05   C          cmp  dh,5         ; color?
D561  7F 02      C          jg   g_jsfy_dot    ; jump if not
D563  FE C1      C          inc  cl          ; prep for wraparound left (color)
D565      C  g_jsfy_dot:
D565  D2 C0      C          rol  al,cl        ; right justify pixel
C
D567  59        C          pop  cx          ; restore registers
D568  5A        C          pop  dx
D569  C3        C          ret
C
D56A      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
D56A          C grf_write_dot  proc    near
C
D56A 52      C          push    dx          ; preserve working registers
D56B 51      C          push    cx
D56C 50      C          push    ax
C
D56D E8 D595 R C          call    g_addr      ; compute address & mask
D570 26: 8A 05 C          mov     al,byte ptr es:[di] ; fetch byte from video memory
D573 5A      C          pop     dx          ; get v mode & dot value to write
D574 52      C          push   dx          ; resave
D575 0A D2   C          or     dl,dl       ; is the XOR bit set?
D577 78 06   C          js     g_xorbit    ; jump if yes
D579 F6 D4   C          not    ah          ; invert mask
D57B 22 C4   C          and    al,ah       ; clear bits for new pixel
D57D F6 D4   C          not    ah          ; restore mask for later
D57F          C g_xorbit:
D57F FE C1   C          inc     cl          ; prep for wraparound right (b&w)
D581 80 FE 05 C          cmp     dh,5          ; color ?
D584 7F 02   C          jg     g_align_dot  ; jump if no
D586 FE C1   C          inc     cl          ; prep for wraparound right (color)
D588          C g_align_dot:
D588 D2 CA    C          ror     dl,cl       ; align new pixel
D58A 22 D4   C          and    dl,ah       ; strip off non-pixel bits (xor bit, etc)
D58C 32 C2   C          xor     al,dl       ; OR or XOR in new pixel depending on xor bit
D58E 26: 88 05 C          mov     byte ptr es:[di],al ; update video memory
C
D591 58      C          pop     ax          ; restore registers
D592 59      C          pop     cx
D593 5A      C          pop     dx
D594 C3      C          ret
C
D595          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
D595          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
D595 51          C          push    cx          ; save column count
D596 BD 0001     C          mov     bp,1          ; shift count; assume not 640x400
D599 8B CD       C          mov     cx,bp          ; multiplier; assume not 640x400
D59B 80 FC 40    C          cmp     ah,64          ; video mode 64 or 72?
D59E 72 06       C          jb     g_skp_1        ; -no: jmp
D5A0 BD 0002     C          mov     bp,2          ; -yes; change shift count
D5A3 B9 0003     C          mov     cx,3          ; -yes: change multiplier
C
D5A6          C  g_skp_1:
D5A6 33 FF       C          xor     di,di          ; init offset amount
D5A8 23 CA       C          and     cx,dx          ; get least sig. bit(s) from row count
D5AA 74 06       C          jz     g_skp_3        ; jmp if nothing to add
D5AC          C  g_skp_2:          ; mini-multiply loop
D5AC 81 C7 2000  C          add     di,2000H        ; add v_ram sub-area size
D5B0 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
D5B2          C  g_skp_3:
D5B2 57          C          push   di          ; temp. store offset
D5B3 8B FA       C          mov     di,dx          ; DX<--row count
D5B5 8B CD       C          mov     cx,bp          ; CX<--mode-related shift val
D5B7 D3 EF       C          shr     di,c1          ; get #rows into a v_ram subarea
C
D5B9 8B D7       C          mov     dx,di          ; save a copy for a moment
D5BB B9 0406     C          mov     cx,0406H        ; mult. di by 80 [rows-->byte offset]
D5BE D3 E7       C          sal     di,c1          ; [fast multiply-
D5C0 8A CD       C          mov     c1,ch          ; by-80]
D5C2 D3 E2       C          sal     dx,c1
D5C4 03 FA       C          add     di,dx          ; #subarea byte offset for this row
C
D5C6 59          C          pop     cx          ; retrieve subarea beginning's offset
D5C7 03 F9       C          add     di,cx          ; offset from start of v_ram
D5C9 5A          C          pop     dx          ; restore column count
C
C ; find column offset
C
D5CA B9 0703     C          mov     cx,703H        ; initialize for black & white
D5CD 80 FC 05     C          cmp     ah,5          ; color ?
D5D0 7F 03       C          jg     g_skp_4          ; CX = 703H (black & white)
D5D2 B9 0302     C          mov     cx,302H        ; CX = 302H (color)
D5D5          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
D5D5 22 EA C      and   ch,d1           ; get division's remainder in CH
D5D7 03 EA C      shr   dx,c1           ; perform division
C
C ; DX = quotient = column's byte offset from start of row
C ; CH = remainder (= pixel's offset into byte)
C
D5D9 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
D5DB 80 F9 03 C      cmp   cl,3           ; black & white?
D5DE 74 02 C      je    g_bitmask       ; jump if yes
D5E0 00 E5 C      shl   ch,1           ; color: convert to bit offset
C
D5E2      C g_bitmask:
D5E2 86 CD C      xchg  cl,ch           ; load CL, preserve "mode"
D5E4 B4 80 C      mov   ah,80H         ; set high bit in byte
D5E6 D2 EC C      shr   ah,cl         ; mask pixel's leftmost bit
D5E8 80 FD 03 C      cmp   ch,3           ; black & white?
D5EB 74 06 C      je    g_skp_5       ; jump if yes
D5ED 8A C4 C      mov   al,ah         ; color: create 2-bit mask
D5EF 00 E8 C      shr   al,1
D5F1 0A E0 C      or    ah,al
C
D5F3      C g_skp_5:           ; DI = byte address, AH = pixel mask,
D5F3 C3 C      ret                    ; CL = bit offset: pixel's leftmost bit
C
D5F4      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
D5F4 C grf_graphics_up proc near
C
D5F4 53 C push bx ; save
D5F5 51 C push cx ; the
D5F6 52 C push dx ; registers
C
C ; cld ;dir. flag = increment (from v_scr1 up)
D5F7 BD 0050 C mov bp,80 ;offset to next scanline (CLD => +80)
C
D5FA 50 C push ax ;save mode, # rows to scroll
D5FB 8B C1 C mov ax,cx ;compute address of window's
D5FD E8 D8E1 R C call g_curs_off ; upper left corner
D600 8B F8 C mov di,ax ;save in DI for string instructions
C
D602 06 C push es ;set DS to video ram for string inst.
D603 1F C pop ds
C
D604 2B D1 C sub dx,cx ;compute window's dimensions
D606 81 C2 0101 C add dx,101H ; DH = height, DL = width
C
D60A 58 C pop ax ;AH = mode, AL = # rows to scroll
C
D60B B3 02 C mov bl,2 ;# interlace areas = 2 for modes 4,5,6
D60D 8A CB C mov cl,bl ;# scanlines per i.a. = 4 for modes ~72
D60F 80 FC 40 C cmp ah,64
D612 72 06 C jb g_tst_mod ;jump if mode = 4,5,6
D614 B3 04 C mov bl,4 ;# interlace areas = 4 for modes 64 & 72
D616 74 02 C je g_tst_mod ;jump if mode = 64
D618 D0 F9 C sar cl,1 ;# scanlines per i.a. = 2 for mode 72
C
D61A C g_tst_mod:
D61A D2 E0 C sal al,cl ;convert number of rows to number of
D61C D2 E6 C sal dh,cl ; scanlines per interlace area
D61E 8B C8 C mov cx,ax ;CH = mode, CL = # scanlines to scroll
D620 80 FD 06 C cmp ch,6 ;are we in a medium resolution mode ?
D623 7D 04 C jge g_set_up ;jump if no
D625 D1 E7 C sal di,1 ;double number of bytes per character
D627 D0 E2 C sal dl,1
C
D629 C g_set_up: ;get address of lines to scroll in refresh ram memory
D629 81 E1 00FF C and cx,00FFH ;CX = # of scanlines to scroll per i.a.
D62D 74 3C C jz g_filler ;if zero, go fill all of window
C
D62F 8B C1 C mov ax,cx ;make DH = # scanlines to fill per i.a.
D631 8A E9 C mov ch,cl ; AX = # scanlines to scroll " "
D633 86 F5 C xchg dh,ch ; CH = # scanlines in window " "
C
D635 8B F7 C mov si,di ;compute address of scanline to be
D637 B1 04 C mov cl,4 ; scrolled to top of window:
D639 D3 E0 C sal ax,cl ; <window's address> +
D63B 03 F0 C add si,ax ; (<# scanlines to scroll per i.a.> *

```

ROM BIOS Listing

```

D63D D1 E0      C      sal    ax,1          ;   <# bytes per scanline = 80>)
D63F D1 E0      C      sal    ax,1
D641 03 F0      C      add    si,ax
C
D643 8A E5      C      mov    ah,ch          ;compute # of scanlines per i.a.
D645 2A E6      C      sub    ah,dh          ;   to be moved
C
D647 33 C9      C      xor    cx,cx          ;CH = 0 for REP counter
C
C ;      jmp short g_scroller      ;go scroll up and fill
C
D649           C      grp_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
D649           C      g_scroller      proc   near
C
D649 56          C      push   si          ;save original source
D64A 57          C      push   di          ; and destination addresses
D64B 53          C      push   bx          ;save interlace areas count (BL)
C
D64C           C      g_m_area:      ;Move Interlace Areas Loop
D64C 57          C      push   di          ;save interlace area
D64D 56          C      push   si          ; addresses
D64E 8A CA      C      mov    cl,dl        ;count of bytes to be moved
D650 F3/ A4      C      rep   movsb        ;move the scanline
D652 5E          C      pop    si          ;restore interlace area
D653 5F          C      pop    di          ; addresses
D654 81 C7 2000 C      add    di,2000H       ;next interlace area
D658 81 C6 2000 C      add    si,2000H       ; addresses
D65C FE CB      C      dec    bl          ;loop to move one scanline in
D65E 75 EC      C      jnz    g_m_area    ; each interlace area
C
D660 5B          C      pop    bx          ;restore interlace areas count (BL)
D661 5F          C      pop    di          ;restore original source

```



```

D662 5E          C      pop     si          ; and destination addresses
D663 03 FD      C      add     di,bp        ;address next scanline in
D665 03 F5      C      add     si,bp        ; each interlace area
D667 FE CC      C      dec     ah          ;loop to move "all" scanlines in
D669 75 DE      C      jnz     g_scroller   ; each interlace area
C
C ;          jmp short g_filler      ;now fill in the gap
C
D66B          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
D66B          C      g_filler   proc   near
C
D66B 8A C7      C      mov     al,bh        ;AL = fill pattern for STOSB instruction
C
D66D          C      g_f_i_lp:   ;Fill Interlace Areas Loop
D66D 57          C      push    di          ;save interlace area address
D66E 52          C      push    dx          ;save scanlines count (DH)
C
D66F          C      g_f_s_lp:   ;Fill Scanlines Loop
D66F 57          C      push    di          ;save scanline address
D670 8A CA      C      mov     cl,dl        ;count of bytes to fill in scanline
D672 F3/ AA     C      rep     stosb       ;fill scanline
D674 5F          C      pop     di          ;restore scanline address
D675 03 FD      C      add     di,bp        ;next scanline in interlace area
D677 FE CE      C      dec     dh          ;fill an interlace area
D679 75 F4      C      jnz     g_f_s_lp
C
D67B 5A          C      pop     dx          ;restore scanlines count (DH)
D67C 5F          C      pop     di          ;restore interlace area address
D67D 81 C7 2000 C      add     di,2000H      ;next interlace area address
D681 FE CB      C      dec     bl          ;fill next interlace area
D683 75 E8      C      jnz     g_f_i_lp
C
D685 5A          C      pop     dx          ;restore

```

ROM BIOS Listing

```

D686 59      C      pop    cx          ;      the
D687 5B      C      pop    bx          ;          registers
D688 C3      C      ret                ;exit scroll routine
C
D689      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
D689      C  grf_graphics_down  proc near
C
D689 53      C      push   bx          ; save
D68A 51      C      push   cx          ;      the
D68B 52      C      push   dx          ;          registers
C
C ;      std                ;dir. flag = decrement (from v_scr1_dn)
D68C BD FF B0 C      mov     bp,-80        ;offset to next scanline (STD => -80)
C
D68F 50      C      push   ax          ;save mode, # rows to scroll
D690 8B C2   C      mov     ax,dx        ;compute address of window's
D692 E8 D8 E1 R C      call    g_curs_off   ; lower right corner
D695 8B F8   C      mov     di,ax        ;save in DI for string instructions
C
D697 06      C      push   es          ;set DS to video ram for string inst.
D698 1F      C      pop     ds
C
D699 2B D1   C      sub     dx,cx        ;compute window's dimensions
D69B 81 C2 0101 C      add     dx,101H      ; DH = height, DL = width
C
D69F 58      C      pop     ax          ;AH = mode, AL = # rows to scroll
C
D6A0 B3 02   C      mov     bl,2         ;# interlace areas = 2 for modes 4,5,6
D6A2 8A CB   C      mov     cl,bl        ;# scanlines per i.a. = 4 for modes ~72
D6A4 80 FC 40 C      cmp     ah,64
D6A7 72 06   C      jb     g_cmp_mod     ;jump if mode = 4,5,6
D6A9 B3 04   C      mov     bl,4         ;# interlace areas = 4 for modes 64 & 72
D6AB 74 02   C      je     g_cmp_mod     ;jump if mode = 64

```

```

D6AD D0 F9          C      sar    c1,1          ;# scanlines per i.a. = 2 for mode 72
C
D6AF              C  g_cmp_mod:
D6AF D2 E0          C      sal    al,c1          ;convert number of rows to number of
D6B1 D2 E6          C      sal    dh,c1          ; scanlines per interlace area
D6B3 80 FC 06      C      cmp    ah,6          ;are we in a medium resolution mode ?
D6B6 7D 05          C      jge    g_setdown      ;jump if no
D6B8 D1 E7          C      sal    di,1          ;double number of bytes per character
D6BA D0 E2          C      sal    dl,1
D6BC 47            C      inc    di          ;address last byte in bottom row
C
D6BD              C  g_setdown: ;get address of lines to scroll in refresh RAM memory
D6BD BE 0050        C      mov    si,80          ;address bottom scanline in i.a.
D6C0 D3 E6          C      sal    si,c1
D6C2 83 EE 50      C      sub    si,80
D6C5 03 FE          C      add    di,si
D6C7 8B C8          C      mov    cx,ax          ;CH = mode, CL = # scanlines to scroll
D6C9 81 E1 00FF    C      and    cx,00FFH       ;CL = # of scanlines to scroll per i.a.
D6CD 74 9C          C      jz     g_filler       ;if zero, go fill all of window
C
D6CF 8B C1          C      mov    ax,cx          ;make DH = # scanlines to fill per i.a.
D6D1 8A E9          C      mov    ch,c1          ; AX = # scanlines to scroll " "
D6D3 86 F5          C      xchg   dh,ch          ; CH = # scanlines in window " "
C
D6D5 8B F7          C      mov    si,di          ;compute address of scanline to be
D6D7 B1 04          C      mov    cl,4          ; scrolled to top of window:
D6D9 D3 E0          C      sal    ax,c1          ; <window's address> -
D6DB 2B F0          C      sub    si,ax          ; (<# scanlines to scroll per i.a.> *
D6DD D1 E0          C      sal    ax,1          ; <# bytes per scanline = 80>)
D6DF D1 E0          C      sal    ax,1
D6E1 2B F0          C      sub    si,ax
C
D6E3 8A E5          C      mov    ah,ch          ;compute # of scanlines per i.a.
D6E5 2A E6          C      sub    ah,dh          ; to be moved
C
D6E7 33 C9          C      xor    cx,cx          ;CH = 0 for REP counter
C
D6E9 E9 D649 R      C      jmp    g_scroller      ;go scroll down and fill
C
D6EC              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
D6EC              C  grf_graphics_read    proc    near

```

```
~
D6EC 53 C push bx ; save
D6ED 51 C push cx ; the
D6EE 52 C push dx ; registers
C
D6EF 8A DC C mov bl,ah ;BL saves v_mode
D6F1 A1 0050 R C mov ax,word ptr ds:[v_curpos]
D6F4 E8 D8E1 R C call g_curs_off ;get address of cursor position
D6F7 8B F0 C mov si,ax ;save as a pointer for later
C
D6F9 8A C3 C mov al,bl ;AL saves v_mode
D6FB B9 0004 C mov cx,4 ;# scanlines per i.a. = 4 for modes ~72
D6FE 33 DB C xor bx,bx ;make BX=0 for modes 4,5,6
D700 3C 40 C cmp al,64
D702 72 08 C jb g_lds_r ;jump if mode = 4,5,6
D704 B3 04 C mov bl,4 ;make BX=4 for mode 64
D706 74 04 C je g_lds_r ;jump if mode = 64
D708 D1 F9 C sar cx,1 ;# scanlines per i.a. = 2 for mode 72
D70A 33 DB C xor bx,bx ;make BX=0 for mode 72
C
D70C C g_lds_r: ;(BX= font pointer offset in master table)
D70C C5 3E 0084 R C lds di,dword ptr ds:[master_tbl_ptr] ;get pointer to master table
D710 C5 79 06 C lds di,dword ptr ds:[di+6][bx] ;get pointer to font's 1st 128 chars
D713 1E C push ds ;XCHG DS,ES
D714 06 C push es
D715 1F C pop ds ;DS:SI -> grafix ram read location ???
D716 07 C pop es ;ES:DI -> font's 1st 128 characters
C
D717 D1 E3 C sal bx,1 ;make BX=8 for modes 4,5,6,72
D719 83 C3 08 C add bx,8 ; BX=16 for mode 64
C ;(BX= number of font bytes per character)
C
D71C 2B E3 C sub sp,bx ;get stack space for font bytes
D71E 8B EC C mov bp,sp ;save pointer to stack space
D720 BA 0002 C mov dx,2 ;number of interlace areas (modes 4,5,6)
D723 3C 06 C cmp al,6 ;check graphics mode
D725 7C 35 C jl g_rd_med ;jump if in medium resolution mode (4,5)
D727 74 03 C je g_rdloop ;jump if in 640x200 resolution mode (6)
C
C ; we're in super resolution (640x400) mode (64 & 72)
D729 BA 0004 C mov dx,4 ;number of interlace areas
C
D72C C g_rdloop: ;Read Scanlines Loop for both 640x200 and 640x400
D72C 51 C push cx ;save # of scanlines per interlace area
D72D 56 C push si ;save current addr. in interlace area #1
D72E 8B CA C mov cx,dx ;init. counter: # of interlace areas
C
D730 C g_rd_ia: ;Read Interlace Area Loop
D730 8A 04 C mov al,[si] ;get byte from grafix ram
D732 88 46 00 C mov [bp],al ;save on reserved stack
D735 45 C inc bp ;bump reserved stack address
D736 81 C6 2000 C add si,2000H ;address next interlace area
D73A E2 F4 C loop g_rd_ia
C
D73C 5E C pop si ;restore interlace area #1 address
```

```

D73D 83 C6 50      C      add    si,80          ;address next scanline in each i.a.
D740 59           C      pop    cx            ;restore # of scanlines counter
D741 E2 E9        C      loop   g_rdloop
C
D743 83 FA 04      C      cmp    dx,4          ;are we in mode 64 or 72 (640x400)?
D746 75 49        C      jne    g_matchb     ;jump if no (reverse video not allowed)
D748 8B F5        C      mov    si,bp        ;point to first byte in stack save area
D74A 2B F3        C      sub    si,bx
D74C 36: F6 04 80  C      test   byte ptr ss:[si],80H ;is upper left bit of char = 0 or 1 ?
D750 74 3F        C      jz     g_matchb     ;jump if 0 (not reversed video)
D752 8B CB        C      mov    cx,bx        ;number of char bytes counter
D754             C      g_unreverse_video_loop:
D754 36: F6 14      C      not   byte ptr ss:[si] ;reverse the reversed byte for matching
D757 46           C      inc    si            ;address next char byte
D758 E2 FA        C      loop   g_unreverse_video_loop
D75A EB 35        C      jmp short g_matchb   ;now find the char in the font table
C
D75C             C      g_rd_med:          ;Read Medium Resolution
D75C D1 E6        C      sal    si,1          ;double graf ram pointer (2 bytes/char)
C
D75E             C      g_medget:         ;Get font bytes in medium resolution (320 X 200) mode
D75E 51           C      push   cx            ;save # scanlines per i.a. counter
D75F 56           C      push   si            ;save current scanline address
D760 B9 0002      C      mov    cx,2          ;init. # interlace areas counter
D763             C      g_med_ia:
D763 51           C      push   cx            ;save i.a. counter
D764 8B 04        C      mov    ax,[si]       ;get 2 bytes of 1 char from video memory
D766 86 E0        C      xchg   ah,al        ;order them logically
C
D768 F7 D0        C      not    ax            ;map background pixels to 0,
D76A 8B D0        C      mov    dx,ax        ; foreground pixels to 1
D76C D1 E2        C      shl    dx,1
D76E 23 D0        C      and    dx,ax
D770 F7 D2        C      not    dx
C
D772 32 C0        C      xor    al,al        ;clear result accumulator
D774 B9 0008      C      mov    cx,8          ;prepare to process 8 bits
D777             C      g_med_bit:
D777 D1 EA        C      shr    dx,1          ;ignore unused bit
D779 D1 EA        C      shr    dx,1          ;load carry with mapped pixel value
D77B D0 D8        C      rcr    al,1          ;rotate it into result accumulator
D77D E2 F8        C      loop   g_med_bit    ;process next bit of character
C
D77F 88 46 00      C      mov    [bp],al       ;save font byte on reserved stack
D782 45           C      inc    bp            ;bump reserved stack pointer
D783 81 C6 2000    C      add    si,2000H      ;address next interlace area
D787 59           C      pop    cx            ;restore i.a. counter
D788 E2 D9        C      loop   g_med_ia     ;process next i.a. of character
C
D78A 5E           C      pop    si            ;restore scanline address
D78B 83 C6 50      C      add    si,80          ;address next scanline
D78E 59           C      pop    cx            ;restore scanlines counter
D78F E2 CD        C      loop   g_medget     ;process next scanline of character
C
D791             C      g_matchb:         ;Match Font Byte: find character

```

```

C
D791 2B EB C sub bp,bx ;point to first byte in stack save area
D793 8B F5 C mov si,bp ;pointer to font bytes from graf ram
D795 32 C0 C xor al,al ;index to font bytes (start with 0)
D797 C g_f_cont: ; Get Font Byte Match Control
D797 16 C push ss ;setup string compare registers
D798 1F C pop ds ;DS:SI -> stack area w/grafix ram bytes
D799 B9 0080 C mov cx,128 ;loop control = 1st 128 ascii chars.
C
D79C C g_f_mach: ;Get Font Byte Match Loop
D79C 51 C push cx ;save loop counter
D79D 8B CB C mov cx,bx ;counter for string compare
D79F 57 C push di ;save font pointer
D7A0 56 C push si ;save pointer to stack save area
D7A1 F3/ A6 C repe cmpsb ;screen bytes match font bytes ?
D7A3 5E C pop si ;retrieve pointer to stack save area
D7A4 5F C pop di ;retrieve pointer to fonmt byte table
D7A5 59 C pop cx ;restore loop counter
D7A6 74 2D C je g_f_exit ;if match go to exit code
D7A8 03 FB C add di,bx ;address next font in table
D7AA FE C0 C inc al ;bump ascii index
D7AC 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
D7AE 0A C0 C or al,al ;have we scanned both 128 char 1/2s ?
D7B0 74 23 C jz g_f_exit ;jump if yes
D7B2 83 FB 10 C cmp bx,16 ;are we in mode 64 ?
D7B5 74 08 C je g_8x16_2 ;jump if yes
C
D7B7 8E D9 C mov ds,cx ;move zero to segment register
C assume ds:abs0
D7B9 C4 3E 007C R C les di,dword ptr ds:[int1Flocn] ;get pointer to 2nd half of 8x8 font
C assume ds:data
D7BD EB 0C C jmp short g_test_addr ;see if font is really there
C
D7BF C g_8x16_2: ;get 2nd half of 8x16 font
D7BF 2E: 8E 1E E538 R C mov ds,word ptr cs:[set_ds_word] ;set DS to data segment
D7C4 C5 3E 0084 R C lds di,dword ptr ds:[master_tbl_ptr] ;get pointer to master table
D7C8 C4 7D 0E C les di,dword ptr ds:[di+14] ;get pointer to 2nd half of 8x16 font
C
D7CB C g_test_addr:
D7CB 8C C0 C mov ax,es ;check if font table is set up
D7CD 0B C7 C or ax,di ;if zeros, then no user font table
D7CF 74 04 C jz g_f_exit ;no table, just go to exit
D7D1 B0 80 C mov al,128 ;offset to 2nd 1/2 of ascii set
D7D3 EB C2 C jmp short g_f_cont ;go back to try rest of ascii set
C
D7D5 C g_f_exit: ;either the character is found, or al = 0
D7D5 03 E3 C add sp,bx ;restore stack pointer
D7D7 5A C pop dx ;restore
D7D8 59 C pop cx ; the
D7D9 5B C pop bx ; registers
D7DA C3 C ret
C

```

```

D7DB          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
D7DB          C  grf_graphics_write      proc      near
C
D7DB 53       C      push    bx                ;save
D7DC 51       C      push    cx                ; the
D7DD 52       C      push    dx                ; registers
C
D7DE 8B D0    C      mov     dx,ax                ;DH= crt mode, DL= char to write
C ; locate beginning of character in graphics ram
D7E0 A1 0050 R C      mov     ax,word ptr ds:[v_curpos]
D7E3 E8 D8E1 R C      call    g_curs_off            ;get address of cursor position
D7E6 8B F8    C      mov     di,ax                ; pointer to graphics location
C ; determine if character is from 1st or 2nd half of table
D7E8 80 FA 80 C      cmp     dl,128                ;is it in first 1/2 of ASCII set ?
D7EB 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
D7ED 80 EA 80 C      sub     dl,128                ;make zero origin for font table lookup
D7F0 80 FE 40 C      cmp     dh,64                ;are we in mode 64 ?
D7F3 75 09    C      jne     g_8x8_2            ;jump if no
C
D7F5 C5 36 0084 R C      lds    si,dword ptr ds:[master_tbl_ptr] ;get pointer to master table
D7F9 C5 74 0E  C      lds    si,dword ptr ds:[si+14] ;get pointer to 2nd half of 8x16 font
C
D7FC EB 08    C      jmp    short g_addr_test        ;see if font table is really there
C
D7FE          C  g_8x8_2:                ;get 2nd half of 8x8 font table
D7FE 33 F6    C      xor     si,si                ;move zero to segment register
D800 8E DE    C      mov     ds,si
C      assume ds:abs0
D802 C5 36 007C R C      lds    si,dword ptr ds:[int1Flocn] ;get pointer to 2nd half of 8x8 font
C      assume ds:data
C
D806          C  g_addr_test:
D806 8C D8    C      mov     ax,ds                ;check if font table is set up
D808 0B C6    C      or     ax,si                ;if zeros, then no 2nd half of table
D80A 75 1D    C      jnz    g_detmode            ;continue if font table is present

```

ROM BIOS Listing

```

D80C 32 D2          C      xor     dl,dl          ;substitute null character
D80E 2E: 8E 1E E538 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
D813              C      g_selfont:
D813 53            C      push    bx              ;preserve register
C
D814 33 DB          C      xor     bx,bx          ;make BX=0 for modes 4,5,6
D816 80 FE 40       C      cmp     dh,64          ;
D819 72 06          C      jb     g_lds_w          ;jump if mode = 4,5,6
D81B B3 04          C      mov     bl,4           ;make BX=4 for mode 64
D81D 74 02          C      je     g_lds_w          ;jump if mode = 64
D81F 33 DB          C      xor     bx,bx          ;make BX=0 for mode 72
C
D821              C      g_lds_w:              ;(BX= font pointer offset in master table)
D821 C5 36 0084 R    C      lds   si,dword ptr ds:[master_tbl_ptr] ;get pointer to master table
D825 C5 70 06       C      lds   si,dword ptr ds:[si+6][bx] ;get pointer to font's 1st 128 chars
C
D828 5B            C      pop     bx              ;restore register
C
D829              C      g_detmode:          ;determine graphics mode
D829 51            C      push   cx              ;
D82A 33 C0          C      xor     ax,ax          ;get ascii code in AX
D82C 8A C2          C      mov     al,dl          ;
D82E B1 03          C      mov     cl,3           ; to multiply by
D830 D3 E0          C      sal     ax,cl          ; 8 (font bytes per character)
D832 59            C      pop     cx              ;
D833 03 F0          C      add     si,ax          ;and add to address of font table
C
D835 B2 04          C      mov     dl,4           ;# scanlines per i.a. (modes 4,5,6,64)
C
D837 80 FE 06       C      cmp     dh,6           ;which resolution are we using?
D83A 7C 49          C      jl     g_med_wr        ;jump if medium resolution (modes 4 & 5)
D83C 74 0F          C      je     g_hi_wr        ;jump if 640x200 resolution (mode 6)
C
C      ;we're in 640x400 resolution
D83E 80 FE 48       C      cmp     dh,72          ;mode 72?
D841 B6 04          C      mov     dh,4           ;# interlace areas = 4 for modes 64 & 72
D843 75 04          C      jne    g_super_wr
C
D845              C      g_tinytext:        ;640x400 resolution (mode 72)
D845 B2 02          C      mov     dl,2           ;# scanlines per i.a. = 2 for mode 72
D847 EB 09          C      jmp short g_repchar
C
D849              C      g_super_wr:        ;640x400 resolution (mode 64)
D849 03 F0          C      add     si,ax          ;multiply ascii code by 16 bytes/char
D84B EB 05          C      jmp short g_repchar
C
D84D              C      g_hi_wr:           ;Hi-resolution Character Write
D84D B6 02          C      mov     dh,2           ;interlace areas count (even/odd)
D84F 80 CB 01       C      or     bl,1           ;mode 6 doesn't allow reverse video
C
D852              C      g_repchar:        ;Repeat Character Loop

```



```

D852 51          C      push  cx          ;save character repeat count
D853 56          C      push  si          ;save source address (font table)
D854 57          C      push  di          ;save destination addr. (grafix ram)
D855 33 C9       C      xor   cx,cx       ;prepare loop counter
D857 8A CA       C      mov   cl,dl       ;scanlines per interlace area counter
C
D859            C      g_line1p:          ;Scanline Loop
D859 51          C      push  cx          ;save scanlines per i.a. counter
D85A 57          C      push  di          ;save interlace area #1 address
D85B 8A CE       C      mov   cl,dh       ;init. interlace areas counter (2 or 4)
C
D85D            C      g_i_a_lp:          ;Interlace Area Loop
D85D AC          C      lodsb          ;get byte from font table
D85E F6 C3 01    C      test  bl,1         ;reverse video?
D861 75 02       C      jnz  g_t_xor       ;jump if no
D863 F6 D0       C      not  al           ;reverse video
D865            C      g_t_xor:          ;Test For XOR
D865 0A DB       C      or   bl,bl        ;XOR the char. with grafix ram?
D867 79 03       C      jns  g_w_byte       ;jump if no
D869 26: 32 05   C      xor   al,es:[di]    ;XOR with grafix ram
D86C            C      g_w_byte:          ;Write Byte
D86C 26: 88 05   C      mov   es:[di],al    ;write byte in grafix ram
D86F 81 C7 2000  C      add  di,2000H        ;address next interlace area
D873 E2 E8       C      loop  g_i_a_lp
C
D875 5F          C      pop  di          ;restore interlace area #1 address
D876 83 C7 50    C      add  di,80          ;address next scanline in each i.a.
D879 59          C      pop  cx          ;restore scanlines per i.a. counter
D87A E2 DD       C      loop  g_line1p
C
D87C 5F          C      pop  di          ;restore char's grafix ram address
D87D 47          C      inc  di          ;address next character in grafix ram
D87E 5E          C      pop  si          ;restore char's font table address
D87F 59          C      pop  cx          ;restore character repeat count
D880 E2 D0       C      loop  g_repchar     ;repeat the character
D882 EB 55 90     C      jmp  g_return       ;exit
C
D885            C      g_med_wr:          ;Medium Resolution Character Write
D885 D1 E7       C      sal  di,1         ;double graf ram pointer (2 bytes/char)
D887 8A D3       C      mov  dl,bl         ;DL saves XOR bit input param (bit 7)
D889 81 E3 0003  C      and  bx,0003H       ;BX= foreground color (& table offset)
D88D 2E: 8A 9F D8DD R C      mov  bl,cs:g_color_table[bx] ;propagate color through byte
D892 8A FB       C      mov  bh,bl         ;propagate color through word
D894 8B EB       C      mov  bp,bx         ;BP saves word of color masks
C
D896            C      g_char_lp:          ;Repeat Character Loop
D896 51          C      push  cx          ;save character repeat counter
D897 56          C      push  si          ;save source address (font table)
D898 57          C      push  di          ;save destination addr. (grafix ram)
D899 B9 0004     C      mov  cx,4          ;init. scanlines per i.a. counter
C
D89C            C      g_scan_lp:          ;Scanline Loop
D89C 51          C      push  cx          ;save scanlines per i.a. counter
D89D 57          C      push  di          ;save interlace area #1 address
D89E B9 0002     C      mov  cx,2          ;init. interlace areas counter

```

```

C
D8A1          C   g_ia_lp:                ;Interlace Area Loop
D8A1  51      C           push   cx                ;save i.a. counter
D8A2  AC      C           lodsb                ;get a byte from the font table
D8A3  8A E0   C           mov     ah,al          ;copy it
D8A5  B9 0008 C           mov     cx,8            ;init. loop counter (8 bits/byte)
C
D8A8          C   g_exp_byt:              ;Expand Byte Loop
D8A8  D0 EC   C           shr     ah,1            ;load carry with font byte bit
D8AA  D1 DB   C           rcr     bx,1            ;rotate it into expansion accumulator
D8AC  D0 E8   C           shr     al,1            ;load carry with same bit as before
D8AE  D1 DB   C           rcr     bx,1            ;double the bit
D8B0  E2 F6   C           loop    g_exp_byt        ;expand font byte bits
C
D8B2  23 DD   C           and     bx,bp          ;color pixels with foreground color
D8B4  86 FB   C           xchg   bh,bl          ;reorder the bytes for grafix ram
D8B6  0A D2   C           or      dl,dl          ;is the XOR bit set ?
D8B8  79 03   C           jns    g_med_store      ;jump if no
D8BA  26: 33 1D C           xor     bx,es:[di]       ;XOR with grafix ram
D8BD          C   g_med_store:
D8BD  26: 89 1D C           mov     es:[di],bx       ;update grafix ram
D8C0  81 C7 2000 C          add     di,2000H         ;address next interlace area
D8C4  59      C           pop     cx                ;restore i.a. loop counter
D8C5  E2 DA   C           loop    g_ia_lp          ;next interlace area
C
D8C7  5F      C           pop     di                ;restore interlace area #1 address
D8C8  83 C7 50   C          add     di,80            ;address next scanline in each i.a.
D8CB  59      C           pop     cx                ;restore scanline loop counter
D8CC  E2 CE   C           loop    g_scan_lp        ;next scanline
C
D8CE  5F      C           pop     di                ;restore char's grafix ram address
D8CF  47      C           inc     di                ;address next character in grafix ram
D8D0  47      C           inc     di
D8D1  5E      C           pop     si                ;restore char's font table address
D8D2  59      C           pop     cx                ;restore character repeat count
D8D3  E2 C1   C           loop    g_char_lp        ;repeat the character
C
D8D5  8A E3   C           mov     ah,bl            ;return AX with last word written
D8D7  8A C7   C           mov     al,bh
C
D8D9          C   g_return:                ;Return from Write Char
D8D9  5A      C           pop     dx                ;restore
D8DA  59      C           pop     cx                ;      the
D8DB  5B      C           pop     bx                ;      registers
D8DC  C3      C           ret
C
D8DD          C   g_color_table  label  byte ;Table of foreground colors extended to byte
C
D8DD  00      C           db     00000000B         ;color 0      (bit pattern: 00)
D8DE  55      C           db     01010101B         ;color 1      (bit pattern: 10)
D8DF  AA      C           db     10101010B         ;color 2      (bit pattern: 01)
D8E0  FF      C           db     11111111B         ;color 3      (bit pattern: 11)
C
D8E1          C   grf_graphics_write  endp
C

```

```

C page
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
D8E1      C  g_curs_off      proc      near
C
D8E1  51      C          push      cx          ;save work register
D8E2  8A E8   C          mov       ch,al       ;hold column number
D8E4  8A C4   C          mov       al,ah       ;row number to al
C
D8E6  B1 01   C          mov       cl,1         ;mode 72 shift count (multiply * 2)
D8E8  80 3E 0049 R 48 C          cmp       byte ptr ds:[v_mode],72
D8ED  74 02   C          je        g_72         ;jump if mode 72
D8EF  FE C1   C          inc       cl          ;mode ~72 shift count (multiply * 4)
D8F1      C  g_72:
D8F1  D2 E0   C          shl       al,cl         ;multiply row # by rows per byte
D8F3  32 C9   C          xor       cl,cl         ;zero out the shift count
D8F5  86 E9   C          xchg      ch,cl       ;move column number for add
D8F7  F6 26 004A R C          mul       byte ptr ds:[v_width] ;multiply by bytes per columnn
D8FB  03 C1   C          add       ax,cx         ;compute offset into refresh ram
D8FD  59      C          pop       cx          ;restore register
D8FE  C3      C          ret              ;and return to caller
C
D8FF      C  g_curs_off      endp
C
D8FF      C  code      ends
C          .LIST                      ; start list 2
C          include pwrup1.asm
C
C ;-----
C ;      Filename:      pwrup1.src
C ;
C ;      This module includes CPU, ROM, 8254 p_dma p_timer, & 8237 p_dma
C ;      Controller tests.
C ;
C ;-----
C
D8FF      C  code      segment public 'ROM'
C          assume cs:code, ds:nothing, es:nothing, ss:nothing
C
= 0000    C  PARITY = 0                      ;; CONDITIONAL ASSEMBLY
C
D8FF      C  p1_data1      proc      near
C
D8FF  90      C  even                      ; word-align stack_rom
C
D900  DB48 R   C  stack_rom      dw       i_rom      ; return from i_cpu

```

ROM BIOS Listing

```

D902 DB4E R C dw i_rom_ret1 ; return from rom_checksum
D904 DB56 R C dw i_rom_ret2
D906 DB5E R C dw i_rom_ret3
D908 DB6D R C dw i_dmat ; return from i_rom
D90A DB79 R C dw i_dmat_ret ; return from rtc_chk
D90C DB88 R C dw i_dmac ; return from i_dmat
D90E DBFA R C dw i_dmac_ret ; return from memtst
D910 DC1B R C dw i_pic ; return from i_dmac
C
D912 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
D928 56 65 72 73 20 32 C db 'Vers 2.02',CR,LF,LF
2E 30 32 0D 0A 0A C
D934 00 C db NUL
C
D935 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
D94F 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:
D974 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
D999 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
D9B2 68 20 61 74 20 00 C ill_m2 db 'h at ',NUL
D9B8 20 2A 00 C ill_m3 db ' ',NUL
C
D9BB 20 20 50 61 73 73 C pass_m db ' Pass',CR,LF,NUL
0D 0A 00 C
D9C4 20 50 61 73 73 0D C spass_m db ' Pass',CR,LF,NUL
0A 00 C
D9CC 20 20 46 61 69 6C C fail_m db ' Fail',NUL
00 C
C
D9D3 43 50 55 20 28 69 C i_cpu_m db 'CPU (i286) ',NUL ; Pass/Fail
32 38 36 29 20 20 C

```

```

00 C
D9E0 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
D9ED 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
D9FA 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
DA07 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
DA14 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
DA21 4E 50 55 20 28 69 C i_npu_m db 'NPU (i287) ',NUL ; Pass/Fail
32 38 37 29 20 20 C
00 C
DA2E 43 61 6C 65 6E 64 C i_calr_m db 'Calendar Clk',NUL ; Fail only%
61 72 20 43 6C 6B C
00 C
DA3B 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
DA48 3A 4C 4F 00 C i_rtc_lo_m db ':LO',NUL ; Error #1 (must remain in
DA4C 3A 48 49 00 C i_rtc_hi_m db ':HI',NUL ; Error #2 order for addr.
DA50 3A 4E 52 00 C i_rtc_nr_m db ':NR',NUL ; Error #3 calculation!)
DA54 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
DA61 3A 53 54 00 C i_kb_st_m db ':ST',NUL
DA65 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
DA72 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
DA7F 20 6B 62 20 52 41 C i_RAM_m db ' kb RAM ',NUL ; Pass/Fail:cc:y000:zzzz:www:rrrr
4D 20 20 00 C
DA89 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
DA96 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
DAA3 46 6C 6F 70 70 79 C i_fduB_m db 'Floppy (B:)',NUL
20 28 42 3A 29 20 C
00 C
DAB0 20 4E 6F 74 C i_fdu_not_m db ' Not' ; purposeely no NUL!!!!
DAB4 20 52 65 61 64 79 C i_fdu_rdy_m db ' Ready',CR,LF,NUL
0D 0A 00 C
DABD 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

```

```

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
DACA 0F      C i_cal_val    db      0Fh      ; port 074h = units of minutes (0-9)
DACB 07      C              db      07h      ; port 075h = tens of minutes (0-5)
DACC 0F      C              db      0Fh      ; port 076h = units of hours (0-9)
DACD 03      C              db      03h      ; port 077h = tens of hours (0-2)
DACE 0F      C              db      0Fh      ; port 078h = units of days (0-9)
DACF 03      C              db      03h      ; port 079h = tens of days (0-3)
DAD0 07      C              db      07h      ; port 07Ah = day of week (0-7)
DAD1 0F      C              db      0Fh      ; port 07Bh = tens of months (0-9)
DAD2 01      C              db      01h      ; port 07Ch = units of months (0-1)
C
DAD3          C p1_data1    endp
C
C ;-----
C ; DO NOT change any code between here and the first "out" statement unless
C ; you know what you are doing ... which is doubtful. You could break
C ; merge.
C
DAD3          C diagnostics_1  proc   near
C              assume  cs:code, ds:nothing, es:nothing, ss:nothing
C
DAD3 BA 3FA0  C          mov     dx,bitread      ; misc status latch%
DAD6 EC      C          in     al,dx            ; decide why resetting%
DAD7 A8 20   C          test    al,pwrupl       ; is it a powerup-up?%
DAD9 74 09   C          jz     i_pwrup         ; yes, do diagnostics%
DADB 2E: 8E 1E E538 R C          mov     ds,word ptr cs:[set_ds_word] ; get resumption offset%
DAE0 FF 2E 00A2 R C          jmp     dword ptr ds:[osmerge1]      ; go to UNIX%
C
DAE4          C i_pwrup:
DAE4 FA      C          cli                     ; disable interrupts
DAE5 B0 40   C          mov     al,40h          ; Check Point #0
DAE7 BA 0378 C          mov     dx,378h         ; parallel port data port address
DAEA EE      C          out     dx,al          ; output "Running- Checkpoint 0"
DAEB FC      C          cld                     ; clear string direction flag
C
DAEC BA 3F40 C          mov     dx,3f40h         ; CTC timer port,access to clr intr%
DAEF B0 03   C          mov     al,3           ; value to stop timers%
DAF1 EE      C          out     dx,al          ; stop CTC timer and clear intr%
DAF2 FE C2   C          inc     dl             ; do port 3f41%
DAF4 EE      C          out     dx,al          ; stop timer%
DAF5 FE C2   C          inc     dl             ; do port 3f42%
DAF7 EE      C          out     dx,al          ; stop timer%
C
C ;-----
C ;          80286 CPU Test

```

```

C ;-----
C
DAF8      C i_cpu:
C
C ; Flags Test (All Set): SF, ZF, AF, PF, CF & OF.
C ; (Exercises flags and accumulator only.)
C
DAF8 B8 FF00 C      mov    ax,0FF00h      ; ah = all 1's; al = all 0's
DAFB 9E      C      sahf           ; set SF, ZF, AF, PF, & CF
DAFC 79 25   C      jns    i_cpu_err    ; SF set? if not, abort
DAFE 75 23   C      jnz    i_cpu_err    ; ZF set? if not, abort
DB00 7B 21   C      jnp    i_cpu_err    ; PF set? if not, abort
DB02 73 1F   C      jnb    i_cpu_err    ; CF set? if not, abort
C
DB04 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
DB05 73 1C   C      jnb    i_cpu_err    ; CF = AF set? if not, abort
DB07 0A E4   C      or     ah,ah        ; ah = 0? if not, abort
DB09 75 18   C      jnz    i_cpu_err
C
DB0B B0 40   C      mov    al,40h       ; ah = 0
DB0D 02 C0   C      add    al,al        ; al = 40h + 40h = 80h = -128
DB0F 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
DB11 33 C0   C      xor    ax,ax        ; ax = 0
DB13 9E      C      sahf           ; reset SF, ZF, AF, PF, & CF
DB14 78 0D   C      js     i_cpu_err    ; SF reset? if not, abort
DB16 76 0B   C      jbe    i_cpu_err    ; ZF or CF reset? if not, abort
DB18 7A 09   C      jp     i_cpu_err    ; PF reset? if not, abort
C
DB1A 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
DB1B 72 06   C      jb     i_cpu_err    ; CF = AF reset? if not, abort
DB1D 03 C0   C      add    ax,ax        ; ax = 0? if not, abort
DB1F 75 02   C      jnz    i_cpu_err
C
C                        ; ax = 0 + 0 = 0, so should be no OF.
DB21 71 11   C      jno    i_cpu_ok     ; OF reset? if not, abort
C
C      assume cs:code, ds:code, es:abs0, ss:code
C
DB23      C i_cpu_err:
DB23 8C C8   C      mov    ax,cs        ; satisfy assumptions
DB25 8E D8   C      mov    ds,ax
DB27 8E D0   C      mov    ss,ax        ; use ROM 'stack'
DB29 BC D900 R C      mov    sp,cs:(offset stack_rom)
C
DB2C BE D9D3 R C      mov    si,cs:(offset i_cpu_m)
DB2F 32 E4   C      xor    ah,ah        ; clear ah (no error number to report)
DB31 E9 F720 R C      jmp    i_fatal        ; i_fatal will 'ret' to i_rom
C

```

```

DB34          C  i_cpu_ok:
DB34 8C C8    C      mov    ax,cs                ; satisfy assumptions
DB36 8E D8    C      mov    ds,ax
DB38 8E D0    C      mov    ss,ax                ; use ROM 'stack'
DB3A BC D900 R C      mov    sp,cs:(offset stack_rom)
C
DB3D B0 41    C      mov    al,41h                ; Check Point #1
DB3F BA 0378  C      mov    dx,378h                ; parallel port data port address
DB42 EE      C      out    dx,al                ; output "Running- Checkpoint 1"
C ; Reset the keyboard
C
DB43 B0 00    C      mov    al,0
DB45 E6 61    C      out    p_kctrl,al
DB47 C3      C      ret                    ; will 'ret' to i_rom
C
C ;-----
C ;       ROM Module Test
C ;-----
C
C          assume cs:code, ds:code, es:abs0, ss:code
C
DB48          C  i_rom:
C
C ; Calculate Checksum of ROM.
C
DB48 BE E000  C      mov    si,0E000h            ; ROM starts at ds:si = F000:E000
DB4B E9 E52A R C      jmp    rom_checksum        ; 'call' rom_checksum
DB4E          C  i_rom_ret1:
C          jnz    i_rom_err      ; will 'ret' here
DB4E 75 10    C      jnz    i_rom_err
DB50 BE C000  C      mov    si,0C000h            ; ROM starts at ds:si = F000:C000
DB53 E9 E52A R C      jmp    rom_checksum        ; 'call' rom_checksum
DB56          C  i_rom_ret2:
DB56 75 08    C      jnz    i_rom_err
DB58 BE A000  C      mov    si,0A000h            ; ROM starts at ds:si = F000:A000
DB5B E9 E52A R C      jmp    rom_checksum        ; 'call' rom_checksum
DB5E          C  i_rom_ret3:
DB5E 74 06    C      jz     i_rom_ok            ; will 'ret' here
C
C
DB60          C  i_rom_err:
DB60 BE D9E0 R C      mov    si,cs:(offset i_rom_m)
DB63 E9 F720 R C      jmp    i_fatal                ; ah has illegal checksum
C          ; i_fatal will 'ret' to i_dmat
C
DB66          C  i_rom_ok:
DB66 B0 42    C      mov    al,42h                ; Check Point #2
DB68 BA 0378  C      mov    dx,378h                ; parallel port data port address
DB6B EE      C      out    dx,al                ; output "Running- Checkpoint 2"
DB6C C3      C      ret                    ; will 'ret' to i_dmat
C
C ;-----
C ;       8254 p_dma p_timer Test
C ;-----
C
C          assume cs:code, ds:code, es:abs0, ss:code

```



```

C
DB6D          C i_dmat:
C
C ; Disable 8237A p_dma Controller before the testing of the 8254 p_dma p_timer channel.
C
DB6D B0 04    C      mov    al,dma_cmd_disable    ; disable p_dma controller command
DB6F E6 08    C      out    dma_command,al
C
C ; Proceed with the testing of the 8254 p_dma p_timer channel ()p_8253_1.
C
DB71 B0 74    C      mov    al,074h                ; 01 11 010 0 -> p_8253_1, lsb 1st, mode 2, no BCD
DB73 BA 0041  C      mov    dx,p_8253_1            ; select p_dma refresh counter
DB76 E9 E1C0 R C      jmp    rtc_chk                ; 'call' rtc_chk
DB79          C i_dmat_ret:                      ; will 'ret' here
DB79 74 06    C      jz     i_dmat_ok
C
C
C
DB7B          C i_dmat_err:
DB7B BE D9ED R C      mov    si,cs:(offset i_dmat_m)
DB7E E9 F720 R C      jmp    i_fatal                ; ah has error code to report.
C                                          ; i_fatal will 'ret' to i_dmac
C
C
DB81          C i_dmat_ok:
DB81 B0 43    C      mov    al,43h                 ; Check Point #3
DB83 BA 0378  C      mov    dx,378h                ; parallel port data port address
DB86 EE       C      out    dx,al                 ; output "Running- Checkpoint 3"
DB87 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 ;      8254 p_dma p_timer channel.
C ;-----
C
C      assume cs:code, ds:code, es:abs0, ss:code
C
DB88          C i_dmac:
C
C ; Send a 'master clear' to 8237 p_dma Controller.
C
DB88 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
DB8A B4 01    C      mov    ah,1                   ; TEMP Error #1
C
DB8C E4 0D    C      in     al,dma_temp
DB8E 0A C0    C      or     al,al                  ; al = 0?
DB90 75 75    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 0FFFFh and then 0h for x = 0 to 3.)
C
DB92 BB FFFF C      mov    bx,0FFFFh      ; bx = 0 all bits reset
C
DB95          C i_dmac_pass2:          ; outer loop
C              ; if 1st pass, bx = 0FFFFh
C              ; if 2nd pass, bx = 0h
DB95 BA 0007 C      mov    dx,7              ; loop counter and port address!!!
C
DB98          C i_dmac_lp:              ; inner loop
DB98 8B C3 C      mov    ax,bx          ; get bit test pattern
DB9A EE       C      out    dx,al        ; write low byte of address/count
DB9B 90       C      nop                    ; no successive in's or outs to dma chip
DB9C EE       C      out    dx,al        ; write high byte of address/count
DB9D 90       C      nop                    ;
DB9E EC       C      in     al,dx         ; read low byte of address/count
DB9F 8A E0    C      mov    ah,al        ; save low byte in ah
DBA1 EC       C      in     al,dx         ; read high byte of address/count
C
DBA2 3B C3    C      cmp    ax,bx          ; does what's been read = test pattern?
DBA4 B4 02    C      mov    ah,2          ; TEMP Error #2
DBA6 75 5F    C      jnz    i_dmac_err        ; if not, abort
C
DBA8 4A       C      dec    dx              ; did we decrement past port address 0?
DBA9 79 ED    C      jns    i_dmac_lp        ; if not, continue same pass for all 8.
C
DBAB 43       C      inc    bx              ; 1st pass? if so, bx = 0FFFFh & loop
DBAC 74 E7    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
DBAE B0 FF    C      mov    al,0FFh          ;
DBB0 E6 01    C      out    dma_count_0,al    ; low byte of count for 64k RAM refresh
DBB2 90       C      nop                    ; chip needs time%
DBB3 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
DBB5 B0 58    C      mov    al,dma_mode_0      ; mode for RAM refresh
DBB7 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
DBB9 B0 00    C      mov    al,dma_cmd_enable  ; enable p_dma controller
DBBB 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
DBBD B0 00 C      mov     al,dma_unmask_0      ; turn on RAM refresh channel 0
DBBF E6 0A C      out     dma_mask_bit,al
C
C ; Program p_8253_1 of i8254 p_timer to proper value for RAM refresh.
C
DBC1 B0 74 C      mov     al,t1cmd          ; select p_dma refresh counter
DBC3 E6 43 C      out     p_8253_ctrl,al
C
DBC5 B8 0013 C     mov     ax,t1count        ; load p_dma refresh count
DBC8 E6 41 C      out     p_8253_1,al
DBCA 8A C4 C      mov     al,ah
DBCC E6 41 C      out     p_8253_1,al
C
C ; Check dma_status for 'hot' p_dma request from p_8253_1.
C
DBCE B4 03 C      mov     ah,3              ; TEMP Error #3
DBD0 E4 08 C      in      al,dma_status      ; test for RAM refresh request in status
DBD2 A8 10 C      test    al,010h           ; bit #4 -> channel 0 request
DBD4 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
DBD6 BA 000B C     mov     dx,dma_mode
C
C ; Initialize p_dma channel 1 not used.
C
DBD9 B0 41 C      mov     al,dma_mode_1      ; mode for not used
DBDB EE C      out     dx,al
C
C ; Initialize p_dma channel 2 FDU.
C
DBDC B0 56 C      mov     al,dma_mode_2      ; mode for FDU
DBDE EE C      out     dx,al
C
C ; Initialize p_dma channel 3 display.
C
DBDF B0 43 C      mov     al,dma_mode_3      ; mode for display
DBE1 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
DBE2 32 C0 C      xor     al,al              ; al = 0
DBE4 BA 0083 C     mov     dx,083h          ; loop counter and port address!
C ; dx = dma_segm_3 = 083h
DBE7 C      i_dmac_nib:
DBE7 EE C      out     dx,al
DBE8 FE CA C      dec     dl                ; when dl goes from 80h (-128) to
DBEA 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
DBEC  2E: 8E 1E E538 R  C          mov     ds,word ptr cs:[set_ds_word]    ; satisfy assumptions
DBF1  8B 36 0072 R      C          mov     si,word ptr ds:[reset_flag]         ; save reset_flag
C
DBF5  33 D2             C          xor     dx,dx                               ; dx = 0; test 0000:0 to 0000:FFFF
DBF7  E9 E22F R        C          jmp     memtst                                  ; 'call' memtst
DBFA                                     C i_dmac_ret:                                   ; will 'ret' here
C
DBFA  2E: 8E 1E E538 R  C          mov     ds,word ptr cs:[set_ds_word]    ; satisfy assumptions
DBFF  89 36 0072 R      C          mov     word ptr ds:[reset_flag],si      ; restore reset_flag
C
DC03  B4 04             C          mov     ah,4                                   ; TEMP Error #4
DC05  74 0D             C          jz     i_dmac_ok
C
C
DC07                                     C i_dmac_err:
DC07  BE D9FA R        C          mov     si,cs:(offset i_dmac_m)
DC0A  0B C9             C          or     cx,cx                               ;; if zero then it was a parity error.
DC0C  75 03             C          jnz    i_d_e                               ;;
DC0E  BE E5FB R        C          mov     si,cs:(offset parity1_m)::
DC11                                     C i_d_e:                                       ;;
DC11  E9 F720 R        C          jmp     i_fatal                             ; ah has error code to report.
C                                          ; i_fatal will 'ret' to i_pic
C
DC14                                     C i_dmac_ok:
DC14  B0 44             C          mov     al,44h                             ; Check Point #4
DC16  BA 0378          C          mov     dx,378h                             ; parallel port data port address
DC19  EE               C          out    dx,al                             ; output "Running- Checkpoint 4"
DC1A  C3               C          ret                                     ; will 'ret' to i_pic
C
C ;-----
C ;          8259A Programmable Interrupt Controller Test.
C ;-----
C
C          assume cs:code, ds:data, es:abs0, ss:stack_ram
C
DC1B                                     C i_pic:
DC1B  B8 0030          C          mov     ax,stack_seg                       ;Initialize RAM Stack
DC1E  8E D0             C          mov     ss,ax                             ; on lower tested memory
DC20  BC 0100          C          mov     sp,100h
C
C ;          Initialize & Disable 8259A Programmable Interrupt Controller.
C
DC23  E8 E1A6 R        C          call    i_pic_init
C
C ; Install Interrupt Vectors for diagnostics.
C
C ; Install unexpected diagnostic interrupt vectors.
C
DC26  33 F6             C          xor     si,si                               ; es:si = abs0_seg:int00locn
DC28  8B FE             C          mov     di,si                               ; es:di = abs0_seg:int00locn
DC2A  B9 01FE          C          mov     cx,(0400h-0004h)/2                ; words from 0:0004h to 0:0400h
C
DC2D  B8 DC8C R        C          mov     ax,cs:(offset i_pic_err)           ; store offset i_pic_err
```

```

DC30 AB          C      stosw
DC31 8C C8       C      mov    ax,cs          ; store segment address
DC33 AB          C      stosw          ; es:di = abs0_seg:int00locn + 4
DC34 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
DC37 B8 DC59 R   C      mov    ax,cs:(offset i_pic_0_ok) ; ax = offset i_pic_0_ok
DC3A 33 FF       C      xor    di,di          ; es:di = abs0_seg:int00locn
DC3C B1 05       C      mov    cl,5          ; load INT's 0h through 4h.
C
DC3E             C      i_pic_soft:          ; ax = (4*x)+(i_pic_0_ok)
DC3E AB          C      stosw          ; es:di gets offset i_pic_x_ok
DC3F 47          C      inc    di          ; skip segment (already = cs)
DC40 47          C      inc    di
DC41 05 0004     C      add    ax,4          ; i_pic_x_ok are 4 bytes apart
DC44 E2 F8       C      loop   i_pic_soft   ; until cx = 0.
C
C ; Install hardware diagnostic interrupt vectors.
C
DC46 B8 FF23 R   C      mov    ax,cs:(offset ill_int)    ; ax = offset ill_int
DC49 BF 0020 R   C      mov    di,es:(offset int08locn) ; es:di = abs0_seg:int08locn
DC4C B1 08       C      mov    cl,8          ; load INT's 8h through Fh.
C
DC4E             C      i_pic_hard:
DC4E AB          C      stosw          ; es:di gets offset ill_int
DC4F 47          C      inc    di          ; skip segment (already = cs)
DC50 47          C      inc    di
DC51 E2 FB       C      loop   i_pic_hard   ; until cx = 0.
C
C ; Test software interrupts first (cl = 0 if error).
C
DC53 B7 09       C      mov    bh,09h        ; bx has its 8th and 11th bits set.
C
C ; cx = 0 from loop above.
DC55 F6 F5       C      div    ch          ; generate a divide-by-zero INT 00h
DC57 EB 33       C      jmp    short i_pic_err
DC59             C      i_pic_0_ok:
C ; bh = 09h. set trap & OF flags in bx
C ; (bits 8 and 11 of flags).
DC59 53          C      push   bx          ; put trap flags on stack
DC5A 9D          C      popf          ; generate a single-step trap INT 01h
DC5B EB 2F       C      jmp    short i_pic_err ; must be 4 bytes long!
DC5D             C      i_pic_1_ok:
C
DC5D CD 02       C      INT    02h        ; generate a software interrupt INT 02h
DC5F EB 2B       C      jmp    short i_pic_err ; must be 4 bytes long!
DC61             C      i_pic_2_ok:
C
DC61 CC          C      INT    03h        ; generate a 1-byte break-point INT 03h
DC62 90          C      nop              ; must be 4 bytes long!
DC63 EB 27       C      jmp    short i_pic_err
DC65             C      i_pic_3_ok:
C ; OF overflow flag is still set.

```

ROM BIOS Listing

```

DC65 CE          C          INTO          ; generate an overflow interrupt INT 04h
DC66 90          C          nop          ; must be 4 bytes long!
DC67 EB 23       C          jmp      short i_pic_err
DC69             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
DC69 B0 01       C          mov     al,1          ; initialize mask value = 1
C
DC6B             C i_pic_test:
DC6B E8 E1B7 R   C          call    i_out_mask      ; output pattern, test input
DC6E 75 1C       C          jne     i_pic_err      ; if not same pattern, abort
DC70 D0 D0       C          rcl     al,1          ; rotate test pattern
DC72 73 F7       C          jnc     i_pic_test      ; test again, if not finished
C
DC74 B0 FF       C          mov     al,0FFh         ; test pattern of all ones
DC76 E8 E1B7 R   C          call    i_out_mask      ; output pattern, test input
DC79 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
DC7B FB          C          sti          ; enable interrupts
DC7C 33 C9       C          xor     cx,cx          ; delay awhile, waiting for
DC7E E2 FE       C          loop   $            ; a 'hot' interrupt.
C
DC80 A0 006B R   C          mov     al,byte ptr ds:[intr_flag] ; get the flag from ill_int
DC83 0A C0       C          or      al,al          ; intr_flag = 0?
DC85 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
DC87             C i_pic_hot:
DC87 41          C          inc     cx            ; cx = 0 from loop above.
DC88 D0 D8       C          rcr     al,1          ; increment cx (IR#+1).
DC8A 73 FB       C          jnb     i_pic_hot      ; mask's least significant bit.
C          ; if not set, continue.
C          ; (exit with cl = 1 to 8.)
C
DC8C             C i_pic_err:
DC8C BC 0100     C          mov     sp,100h       ; cl = 0 or failing PIC 'hot' active IR#
C          ; re-initialize stack
C
DC8F 51          C          push    cx            ; save error code.
C
C          ; Install Vector Table.
C          ; set int10locn = code_seg:v_io, and
C          ; set int1Dlocn = code_seg:v_parms.
DC90 E8 E164 R   C          call    i_vector
C
C          ; Initialize Video.
C
DC93 E8 E0A0 R   C          call    i_d_init
C
C          ; Display error message.

```

```

C
DC96 BE DA07 R C      mov     si,cs:(offset i_pic_m)
DC99 E8 E540 R C      call    DRomString      ; display failing test message.
C
DC9C BE D9CC R C      mov     si,cs:(offset fail_m)
DC9F E8 E540 R C      call    DRomString      ; display fail message.
C
DCA2 59 C      pop     cx                ; restore error code.
DCA3 0A C9 C      or      cl,cl            ; cl = 0?
DCA5 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
DCA7 E8 E56C R C      call    DColon          ; display a colon.
DCAA B8 0E48 C      mov     ax,(0Eh*100h)+'H' ; display 'hot' interrupt symbol.
DCAD CD 10 C      int     10h
DCAF 8A C1 C      mov     al,cl           ; transfer error code.
DCB1 48 C      dec     ax              ; error code (1 to 8) to (0 to 7) IR#.
DCB2 E8 E596 R C      call    DHexNib        ; display lowest nibble.
C
DCB5 C      i_pic_no_hot:
DCB5 E8 E55F R C      call    DCrLf
DCB8 EB 07 C      jmp     short i_pic_end
C
DCBA F4 C      hlt
C
DCBB C      i_pic_ok:
DCBB B0 45 C      mov     al,45h          ; Check Point #5
DCBD BA 0378 C      mov     dx,378h         ; parallel port data port address
DCC0 EE C      out     dx,al           ; output "Running- Checkpoint 5"
C
DCC1 C      i_pic_end:
C
C ;-----
C ;      Install Vector Table.
C ;-----
C
DCC1 BC 0100 C      mov     sp,100h         ; re-initialize stack
DCC4 E8 E164 R C      call    i_vector
C
C ;-----
C ;      Determine System Configuration from Switches and Initialize Video.
C ;-----
C
DCC7 E8 E0A0 R C      call    i_d_init
C
C ;-----
C ;      Display Passing Error Messages
C ;-----
C
DCCA C      disp_pass:
DCCA B8 0003 C      mov     ax,3            ; mode co80
DCCD CD 10 C      int     10h            ; Clear screen.
C
DCCF BE D912 R C      mov     si,cs:(offset banner_m)

```

ROM BIOS Listing

```

DCD2 E8 E540 R      C      call   DRomString
C
DCD5 BE D9D3 R      C      mov     si,cs:(offset i_cpu_m)
DCD8 E8 E540 R      C      call   DRomString
DCDB BE D9BB R      C      mov     si,cs:(offset pass_m)
DCDE E8 E540 R      C      call   DRomString
C
DCE1 BE D9E0 R      C      mov     si,cs:(offset i_rom_m)
DCE4 E8 E540 R      C      call   DRomString
DCE7 BE D9BB R      C      mov     si,cs:(offset pass_m)
DCEA E8 E540 R      C      call   DRomString
C
DCED BE D9ED R      C      mov     si,cs:(offset i_dmat_m)
DCF0 E8 E540 R      C      call   DRomString
DCF3 BE D9BB R      C      mov     si,cs:(offset pass_m)
DCF6 E8 E540 R      C      call   DRomString
C
DCF9 BE D9FA R      C      mov     si,cs:(offset i_dmac_m)
DCFC E8 E540 R      C      call   DRomString
DCFF BE D9BB R      C      mov     si,cs:(offset pass_m)
DD02 E8 E540 R      C      call   DRomString
C
DD05 BE DA07 R      C      mov     si,cs:(offset i_pic_m)
DD08 E8 E540 R      C      call   DRomString
DD0B BE D9BB R      C      mov     si,cs:(offset pass_m)
DD0E E8 E540 R      C      call   DRomString
C
DD11 E8 E5E4 R      C      call   enable_parity           ; enable parity routine
C ; -----
C ;      Size & clear RAM at every 64k byte bank past the lowest 64k.
C ; -----
C
DD14                C      RAM_size_tst:
C                      assume cs:code, ds:data, es:abs0, ss:stack_ram
C
DD14 BD 0040         C      mov     bp,64                 ;initialize memory count
C
DD17 8B 36 0072 R   C      mov     si,word ptr ds:[reset_flag] ; get warm bootflag
DD1B 81 EE 1234     C      sub     si,01234h           ; si=0 iff CTL ALT DEL sequence.
DD1F 33 FF          C      xor     di,di              ; offset = 0000h
C
DD21 BA 1000        C      mov     dx,1000h           ; start at 1000:0000 (dx keeps segment)
DD24                C      RAM_size_lp:
DD24 8E C2          C      mov     es,dx              ; get segment
DD26 0B F6         C      or     si,si              ; if not a warm boot then save contents%
DD28 74 03         C      jz     save_ram           ; of RAM%
DD2A 26: 89 05     C      mov     word ptr es:[di],ax ; write something to RAM%
C                      ; to initialize parity correctly%
DD2D                C      save_ram:
DD2D 26: 8B 05     C      mov     ax,word ptr es:[di] ; read existing ram value
DD30 F7 D0         C      not    ax                 ; complement it
DD32 26: 89 05     C      mov     word ptr es:[di],ax ; write complement back to RAM
C
DD35 26: 8B 1D     C      mov     bx,word ptr es:[di] ; read back from RAM
C

```



```

DD38 3B C3          C      cmp     ax,bx          ; verify to test for end of RAM
DD3A F7 D0          C      not     ax            ; recreate original value
DD3C 75 41          C      jne     pmemcnt       ; if verify fails, go see protected mode
C                                     ; RAM
C ;      jne     RAM_size_end ; if verify fails, at end of RAM
C ;      je      go_on1       ; RAM_size_end out of range %
C ;      jmp     RAM_size_end
C
DD3E               C      go_on1:
DD3E 26 89 05       C      mov     word ptr es:[di],ax ; restore original value back to RAM
DD41 0B F6          C      or      si,si         ; test warm boot flag
DD43 74 20          C      jz      RAM_size_nxt   ; if CTL ALT DEL sequence,
C                                     ; don't clear memory
DD45 E8 E2F R       C      call    memtst        ; test and clear memory
C ;      jnz     RAM_error     ; test flag from storage test
DD48 74 03          C      jz      go_on2       ; RAM_error out of range%
DD4A E9 DF05 R     C      jmp     RAM_error
C
DD4D               C      go_on2:
DD4D 83 C5 40       C      add     bp,64          ; increment size
DD50 56             C      push    si            ; save Warm Boot Flag
DD51 B8 0E0D        C      mov     ax,0E0Dh       ; put out a CR
DD54 CD 10          C      INT     10H
C
DD56 8B C5          C      mov     ax,bp          ; display tested RAM
DD58 BB 0003        C      mov     bx,3
DD5B E8 E5B3 R     C      call    DNumW
DD5E BE DA7F R     C      mov     si,cs:(offset i_RAM_m)
DD61 E8 E540 R     C      call    DRomString
C
DD64 5E             C      pop     si            ; retrieve Warm Boot Flag
C
DD65               C      RAM_size_nxt:        ; maximum RAM = 640k = 10 * 64k
DD65 80 C6 10       C      add     dh,10h         ; next segment
DD68 33 C0          C      xor     ax,ax
DD6A 8A C6          C      mov     al,dh            ; ax = (RAM size/16)/256 = RAM size/4k
DD6C D1 E0          C      shl     ax,1            ; ax = (RAM size/4k) * 2 = RAM size/2k
DD6E D1 E0          C      shl     ax,1            ; ax = (RAM size/2k) * 2 = RAM size/1k
DD70 1E             C      push    ds            ; save ds%
DD71 2E: 8E 1E E538 R C      mov     ds,word ptr cs:[set_ds word] ; restore data segment pointer%
DD76 A3 0013 R     C      mov     word ptr ds:[memory_size],ax ; new size into memory_size%
DD79 1F             C      pop     ds            ; restore ds%
DD7A 80 FE A0       C      cmp     dh,0A0h         ; top of RAM yet (A000:0000)?
DD7D 72 A5          C      jb      RAM_size_lp     ; if not, continue.
C ;
C ;
C ; jump into protected mode and test memory above 640k bytes%
C ; This is new code added for 6300 Plus
C ;
C
DD7F               C      pmemcnt:
C      assume cs:code, ds:data
DD7F B8 0040        C      mov     ax,data_seg
DD82 8E D8          C      mov     ds,ax          ; satisfy assumptions
DD84 8B 2E 0013 R   C      mov     bp,word ptr ds:[memory_size] ; bp has real mode memory

```

ROM BIOS Listing

```

C                                     ; size at this point
C
DD88 FF 36 00A8 R C      push   word ptr ds:[osmerge2+2]      ; EGA -- GWBASIC fix
DD8C FF 36 00AA R C      push   word ptr ds:[osmerge2+4]      ; save locations 4a8 and 4aa on
C                                     ; stack an restore when memory
C                                     ; count is complete
C
DD90 E8 E05E R   C      call    i_gdt                      ; initialize standard gdt entries
C
DD93 B8 DE3C R   C      mov     ax,word ptr cs:[offset ptestaddr]      ; initialize gdt code segment%
DD96 A3 00B4 R   C      mov     word ptr ds:[gdt+8+2],ax
DD99 B0 FA       C      mov     al,0fah
DD9B A2 00D2 R   C      mov     ds:[addr],al                ; start counting at 15M+640k%
C
DD9E             C      innerloop:
DD9E B8 DDD5 R   C      mov     ax,cs:[offset wereback]
DDA1 A3 00A2 R   C      mov     word ptr [osmerge1],ax          ;return location in memory%
DDA4 B8 F000     C      mov     ax,0f000h
DDA7 A3 00A4 R   C      mov     word ptr ds:[osmerge1+2],ax          ;%
C
DDAA 60          C      ;      pusha                      ; save the world%
DDAB 1E          C      db      60h
DDAC 06          C      push   ds
DDAD FA          C      push   es
DDAE A0 00D2 R   C      cli                      ; stop all interrupts%
DDB1 A2 00BE R   C      mov     al,ds:[addr]
C      mov     byte ptr ds:[gdt+10h+4],al      ; high byte into proper location%
C      ;      ; of gdt%
DDB4 BB 00C2 R   C      mov     bx,ds:[offset gdtalias]
C      ;      lgdt    [bx]                      ; load up the gdt%
DDB7 0F 01 17   C      db      0fh,01h,17h
C
DDBA BA 3F20     C      mov     dx,3f20h          ; ff port%
DDBD B0 90       C      mov     al,90h           ; enable upper 4 data lines%
C      ;      ; and return to wereback upon
C      ;      ; reset
DDBF EE         C      out     dx,al           ; set the ff
C      ;      smsw   ax            ; machine status word into ax
DDC0 0F 01 E0   C      db      0fh,01h,0e0h
DDC3 0D 0001     C      or      ax,1            ; set Protection Enable bit
C      ;      lmsw   ax            ; enable protection
DDC6 0F 01 F0   C      db      0fh,01h,0f0h
DDC9 EB 01 90     C      jmp     foo              ; clear prefetch que
DDCC             C      foo:
DDCC 2E: FF 2E DDD1 R C      jmp     dword ptr cs:[holdon]      ; indirect jump to 8:0
DDD1             C      holdon:
DDD1 0000        C      dw     offset 0          ; engage warp drives, scotty
DDD3 0008        C      dw     8
C
DDD5             C      wereback:
DDD5 B8 0030     C      mov     ax,stack_seg
DDD8 8E D0       C      mov     ss,ax           ; restore stack segment
DDDA 07          C      pop     es              ; restore yourself%
DDDB 1F          C      pop     ds
C      ;      popa

```

```

DDDC 61          C      db      61h
DDD4 BA 3F20     C      mov     dx,3f20h          ; ff port%
DDE0 B0 00       C      mov     al,0h           ; make memory writable again%
C                                     ; and return to wereback upon
C                                     ; reset
DDE2 EE         C      out     dx,al           ; set the ff
DDE3 FB         C      sti     ; enable interrupts%
DDE4 8A 1E 00A8 R C      mov     bl,byte ptr ds:[osmerge2+2] ; load error code into bl%
DDE8 80 FB 00    C      cmp     bl,0h           ; test for error%
DDEB 75 34       C      jnz     stop           ; stop if some type of error%
DDED 83 C5 40    C      add     bp,64          ; otherwise increment size%
DDF0 0B F6       C      or     si,si           ; if warm boot%
DDF2 74 15       C      jz     noprint        ; don't print messages%
C
DDF4 56          C      push    si           ; save warm boot flag%
DDF5 B8 0E0D     C      mov     ax,0E0Dh        ; put out CR LF just in case%
DDF8 CD 10       C      int     10h           ; call screen display%
DDFA 8B C5       C      mov     ax,bp           ; display tested RAM%
DDFC BB 0003     C      mov     bx,3
DDFF E8 E5B3 R   C      call   DNumW
DE02 BE DA7F R   C      mov     si,cs:(offset i_RAM_m)
DE05 E8 E540 R   C      call   DROMString
DE08 5E         C      pop     si           ; restore warm boot flag%
C
DE09            C      noprint:
DE09 A0 00D2 R   C      mov     al,ds:[addr]      ; current 64k chunk into al%
DE0C 3C FF       C      cmp     al,0ffh          ; are we at 16Mbytes?%
DE0E 75 02       C      jne     cont_cnt        ; if not continue count%
DE10 B0 0F       C      mov     al,0fh           ; if we hit 16MB then jump%
C                                     ; back down to 1MB boundary%
C                                     ; al gets incremented below
C
DE12            C      cont_cnt:
DE12 3C F9       C      cmp     al,0f9h          ; are we at 15M+640k boundary?%
C      ;      je     RAM_size_end ; if so we are at max ram%
C                                     ; so stop%
DE14 75 03       C      jne     go_on3
DE16 E9 DEC6 R   C      jmp     RAM_size_end
C
DE19            C      go_on3:
DE19 FE C0       C      inc     al           ; otherwise increment address%
DE1B A2 00D2 R   C      mov     ds:[addr],al
C      ;      loop   innerloop ; continue to test the memory%
C      ;      jcxz  stop
DE1E E9 DD9E R   C      jmp     innerloop
DE21            C      stop:
DE21 80 FB 01     C      cmp     bl,1           ; an error was reported%
C      ;      je     RAM_size_end ; it was end of memory%
DE24 75 02       C      jne     go_on4
C      ;      jmp     RAM_size_end
DE26 EB E1       C      jmp     noprint        ; we're gonna look at every
C                                     ; segment boundary in protected
C                                     ; mode to see if there is hole
C
DE28            C      go_on4:

```

ROM BIOS Listing

```

DE28 8B 16 00CA R      C      mov     dx,ds:[seg_fail]      ; setup registers expected by
                                C                                ; RAM error
DE2C 8E C2             C      mov     es,dx
DE2E 8B 3E 00CC R      C      mov     di,ds:[off_fail]
DE32 A1 00CE R        C      mov     ax,ds:dwrite
DE35 8B 1E 00D0 R      C      mov     bx,ds:dread
                                C
DE39 E9 DF05 R        C      jmp     RAM_error              ; it was an error%
                                C
                                C .....
                                C ;
                                C ; ptestaddr is the code that runs in protected mode to test memory above
                                C ; 640k bytes. It is patterned CLOSELY after the memory test code that
                                C ; runs in real mode.
                                C ;
                                C .....
DE3C                                C ptestaddr:
DE3C B8 0010          C      mov     ax,10h
DE3F 8E C0            C      mov     es,ax                ; set up es selector%
DE41 33 FF            C      xor     di,di                 ; clear di%
DE43 26: 89 05        C      mov     word ptr es:[di],ax   ; initialize es:[di]%
                                C                                ; first to set parity bit %
DE46 26: 8B 05        C      mov     ax,word ptr es:[di]   ; mov es:[di] into ax%
DE49 F7 D0            C      not    ax                    ; complement ax%
DE4B 26: 89 05        C      mov     word ptr es:[di],ax   ; mov complement into es:[di]%
DE4E 26: 8B 1D        C      mov     bx, word ptr es:[di]  ; mov es:[di] into bx%
DE51 3B C3            C      cmp     ax,bx                ; is ax = bx?%
DE53 75 49            C      jne    endofram             ; if not we passed end of ram%
DE55 B9 8000          C      mov     cx,08000h           ; load cx with word count for%
                                C                                ; 64k bytes%
DE58 FC              C      cld                        ; increment direction%
DE59                                C pmemtst_w1:
DE59 8B C7            C      mov     ax,di                ; ax = location%
DE5B AB              C      stosw                       ; write location to memory%
DE5C E2 FB            C      loop   pmemtst_w1           ; write all 64k%
DE5E B9 8000          C      mov     cx,08000h           ; reset loop counter%
DE61 33 FF            C      xor     di,di                ; clear di%
                                C
DE63                                C pmemtst_r1:
DE63 8B C7            C      mov     ax,di                ; set ax%
DE65 26: 8B 1D        C      mov     bx,word ptr es:[di]  ; read memory location into bx%
DE68 3B C3            C      cmp     ax,bx                ; compare location with contents%
DE6A 75 37            C      jne    ptsterr              ; if not equal report error%
DE6C 47              C      inc     di                    ; increment di twice%
DE6D 47              C      inc     di
DE6E E2 F3            C      loop   pmemtst_r1
DE70 B9 8000          C      mov     cx,08000h           ; reset loop counter%
DE73 33 FF            C      xor     di,di                ; clear di%
                                C
DE75                                C pmemtst_w2:
DE75 8B C7            C      mov     ax,di                ; ax = location%
DE77 F7 D0            C      not    ax                    ; complement location%
DE79 AB              C      stosw                       ; write complement location to memory
DE7A E2 F9            C      loop   pmemtst_w2           ; write all 64k%
DE7C B9 8000          C      mov     cx,08000h           ; reset loop counter%

```

```

DE7F 33 FF          C      xor     di,di          ; clear di%
C
DE81              C      pmentst_r2:
DE81 8B C7          C      mov     ax,di          ; set ax%
DE83 F7 D0          C      not     ax            ; complement location%
DE85 26: 8B 1D      C      mov     bx,word ptr es:[di] ; read memory location into bx%
DE88 3B C3          C      cmp     ax,bx          ; compare location with contents%
DE8A 75 17          C      jne     ptsterr        ; if not equal report error%
DE8C 47             C      inc     di            ; increment di twice%
DE8D 47             C      inc     di
DE8E E2 F1          C      loop   pmentst_r2
C
DE90 B9 8000        C      mov     cx,08000h       ; reset loop counter%
DE93 33 FF          C      xor     di,di          ; clear di%
DE95 8B C7          C      mov     ax,di          ; clear ax%
DE97 F3/ AB         C      rep     stosw         ; zero memory%
DE99 B0 00          C      mov     al,0           ; no error
DE9B EB 1D 90       C      jmp     history        ; return to real mode%
C
DE9E              C      endofram:
DE9E B0 01          C      mov     al,1           ; end of memory code
DEA0 EB 18 90       C      jmp     history        ; return to real mode%
C
DEA3              C      ptsterr:
DEA3 32 D2          C      xor     dl,dl
DEA5 8A 36 00D2 R   C      mov     dh,ds:[addr]
DEA9 89 16 00CA R   C      mov     ds:[seg_fail],dx ; save error codes in
DEAD 89 3E 00CC R   C      mov     ds:[off_fail],di ; a memory location
DEB1 A3 00CE R       C      mov     ds:[dwrite],ax  ; so that REAL MODE can
DEB4 89 1E 00D0 R   C      mov     ds:[dread],bx  ; print it
DEB8 B0 02          C      mov     al,2           ; error code
C
DEBA              C      history:
C ; in location osmerge2+2%
DEBA A2 00A8 R      C      mov     byte ptr ds:[osmerge2+2],al ; error code to report%
DEBD BA 3F00        C      mov     dx,3f00h       ; reset port%
DECO EC           C      in     al,dx          ; bang the port%
C ; take the mains offline
C ; scotty
DEC1 90           C      nop
DEC2 90           C      nop ; incase prefetch que%
DEC3 90           C      nop ; has fetched ahead%
DEC4 FA           C      cli ; should never get%
DEC5 F4           C      hlt ; to this point%
C
C ; REAL MODE AGAIN
C      assume cs:code, ds:data, es:abs0, ss:stack_ram
DEC6              C      RAM_size_end:
DEC6 0B F6          C      or     si,si          ; test warm boot flag
DEC8 74 11          C      jz     RAM_size_end_1 ; if CTL ALT DEL sequence,
DECA 81 FD 0400    C      cmp     bp,400h       ; if RAM >999 then print different pass
C ; message Dawson MADE me do it!!
DECE 72 05          C      jb     rpass
DEDO BE D9C4 R     C      mov     si,cs:(offset spass_m)

```

ROM BIOS Listing

```

DED3 EB 03          C      jmp      short ppass
DED5               C rpass:
DED5 BE D9BB R     C      mov     si,cs:(offset pass_m) ;Display OK
DED8               C ppass:
DED8 E8 E540 R     C      call    DRomString
                  C
DEDB               C RAM_size_end_1:                ; bx = RAM size/16
DEDB 33 C0         C      xor     ax,ax                  ; ax = 0
DEDD 8E C0         C      mov     es,ax                ; satisfy assumptions es = 0 = abs0_seg
DEDF 2E: 8E 1E E538 R C      mov     ds,word ptr cs:[set_ds_word]
DEE4 A1 0013 R     C      mov     ax,ds:[memory_size]   ; see if real memory full%
                  C ; cmp     ax,280h                ; if memory > 640k DOS won't load%
DEE7 3B C5         C      cmp     ax,bp                  ; if total memory > real memory%
DEE9 72 08         C      jb     mor_mem                ; see if memory above 640k%
DEEB C7 06 00CA R 0000 C      mov     ds:[seg_fail],0       ; zero out p-mode mem size%
DEF1 EB 08         C      jmp     short no_pmem          ; dos is ok%
                  C
DEF3               C mor_mem:
DEF3 8B DD         C      mov     bx,bp                ; bp has total mem size at this point
DEF5 2B D8         C      sub     bx,ax                ; subtract real mode memory from total%
                  C ; memory to get p-mode memory in ax%
DEF7 89 1E 00CA R  C      mov     ds:[seg_fail],bx       ; mov p-mode mem size into seg_fail location%
DEFB -            C no_pmem:
                  C
DEFB 8F 06 00AA R  C      pop     word ptr ds:[osmerge2+4] ; EGA -- GWBASIC fix
DEFF 8F 06 00A8 R  C      pop     word ptr ds:[osmerge2+2] ; restore locations 4aa and 4a8 from
                  C ; stack an restore when memory
                  C ; count is complete
                  C
                  C ; GoTo Display Passing Messages
                  C
DF03 EB 30         C      jmp     short i_cal            ;Go check clock calendar
                  C
DF05               C RAM_error:                    ; Error message looks like: Fail:cc:y000:zzzz:www: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
DF05 52           C      push    dx                  ;save failing segment
DF06 1E           C      push    ds                  ;save ds
DF07 50           C      push    ax                  ; save failing test pattern.
                  C
DF08 E8 E55F R     C      call    DCrLf                ;Carriage Return, Line Feed
DF0B BE D9CC R     C      mov     si,cs:(offset fail_m)
DF0E E8 E540 R     C      call    DRomString            ; display fail message.
                  C
DF11 E8 E56C R     C      call    DColon                ; display a colon
                  C
DF14 E4 66         C      in     al,sys_conf_a        ; get RAM configuration.
DF16 24 0F         C      and    al,0Fh                ; mask valid bits.
DF18 E8 E589 R     C      call    DHexByte              ; display RAM configuration.
                  C

```

```

DF1B E8 E56C R      C      call   DColon          ; display a colon
C
DF1E 8E DA          C      mov    ds,dx          ; ds = failing segment = dx
DF20 8B C7          C      mov    ax,di         ; ax = failing segment = di
DF22 E8 E578 R      C      call   DHexLong       ; display ds:ax
C
DF25 E8 E56C R      C      call   DColon          ; display a colon
C
DF28 1F             C      pop    ds             ; ds = failing test pattern = on stack
DF29 8B C3          C      mov    ax,bx         ; ax = what was read = bx
DF2B E8 E578 R      C      call   DHexLong       ; display ds:ax
DF2E E8 E55F R      C      call   DCrLf
C
DF31 1F             C      pop    ds             ;restore ds
DF32 5A             C      pop    dx             ;restore failing segment
DF33 EB A6          C      jmp    short RAM_size_end_1
C
C ;-----
C ;      MM58274 Clock Calendar Device Test
C ;-----
C
DF35                C i_cal:
C
C IF 64TOD
DF35 B0 01          C      mov    al,01h       ; This should fix the factory
DF37 BA 0070        C      mov    dx,70h       ; initial powerup problem
DF3A EE            C      out   dx,al        ; al=1 implies Normal mode, Clock run
C ; Clock setting reg, interrupt stop
C ENDF
C
C ; Read Clock Calendar Device.          ; bx = day (from 1-1 of leap year)
C ; ch = hour
DF3B B4 FE          C      mov    ah,-2       ; cl = minutes
DF3D CD 1A          C      int   1Ah          ; dh = seconds
C ; dl = hundredths of seconds
C ; Check time & date read.
C
C ; Hundredths of Seconds.
C
DF3F B8 000A        C      mov    ax,10        ; ax = 10; dl = hundredths
DF42 86 C2          C      xchg  al,dl         ; ax = hundredths; dl = 10
DF44 F6 F2          C      div   dl           ; ah = remainder, al = quotient
C ; (can only read tenths of seconds.)
DF46 3D 000A        C      cmp    ax,10        ; ah should = 0, and al should be < 10.
DF49 73 15          C      jae   i_cal_1_1_80 ; if not, test chip & write 1-1-80.
C
C ; Seconds.
C
DF4B 80 FE 3C       C      cmp    dh,60        ; dh = seconds should be < 60.
DF4E 73 10          C      jae   i_cal_1_1_80 ; if not, test chip & write 1-1-80.
C
C ; Minutes.
C
DF50 80 F9 3C       C      cmp    cl,60        ; cl = minutes should be < 60.

```

```
DF53 73 0B      C      jae    i_cal_1_1_80      ; if not, test chip & write 1-1-80.
C
C      ; Hours.
C
DF55 80 FD 18   C      cmp     ch,24                ; ch = hour should be < 24.
DF58 73 06      C      jae    i_cal_1_1_80      ; if not, test chip & write 1-1-80.
C
C      ; Days.
C
DF5A 81 FB 0B6A C      cmp     bx,(2*366)+(6*365) ; bx = day from leap year mod 8 should
C                                     ; be < (0-2921) = (0-0B69h)
DF5E 72 52      C      jb     i_cal_end          ; if so, valid time & day, skip test.
C
DF60            C      i_cal_1_1_80:          ; else invalid time & day, write 1-1-80.
C
C      ; Initialize and Stop Clock.
C      ENDIF
C      IF G4TOD
DF60 B8 0015    C      mov     ax,15h          ; interrupt stop, clock stop%
DF63 E6 70      C      out     70h,al          ;%
DF65 33 C0      C      xor     ax,ax           ;%
DF67 E6 7F      C      out     7Fh,al          ; no interrupts programmed%
DF69 B8 0005    C      mov     ax,5            ;%
DF6C E6 70      C      out     70h,al          ; clock is out of test mode, halted%
C      ENDIF
C                                     ; clock setting register is selected%
C      ; Output test pattern of maximum value with all bits set to read/writable ports.
C
DF6E BE DACA R  C      mov     si,cs:(offset i_cal_val)
DF71 B9 0009    C      mov     cx,9            ; ch keeps 0; cx = 9
DF74 BA 0074    C      mov     dx,0074h        ; dh keeps 0; dx = 74h
C
DF77            C      i_cal_max:
DF77 2E AC      C      lods   byte ptr cs:[si] ; al get cs:si (ds overridden!).
DF79 8A E0      C      mov     ah,al          ; save maximum value.
DF7B EE        C      out     dx,al          ; ports 74 through 7C (units of minutes
C                                     ; to tens of months) get max value.
DF7C EC        C      in     al,dx           ; read it back.
DF7D 22 C4      C      and     al,ah          ; mask valid bits.
DF7F 3A C4      C      cmp     al,ah          ; is it equal to the value written?
DF81 75 10      C      jnz    i_cal_err      ; if not, abort.
C
DF83 42        C      inc     dx             ; increment to next port
DF84 E2 F1      C      loop   i_cal_max
C
C      ENDIF
C
C      ; Write out 0h (bits of lower nibble reset) test pattern to read/writable ports.
C
C                                     ; al kept 0h.
DF86 B1 09      C      mov     cl,9           ; ch kept 0; cx = 9
DF88 B2 74      C      mov     dl,74h         ; dh kept 0; dx = 74h
DF8A            C      i_cal_0:
DF8A EE        C      out     dx,al          ; ports 74 through 7C (units of minutes
C                                     ; to tens of months) get 0h.
```



```

DF8B EC          C      in    al,dx          ; read it back.
DF8C 24 0F      C      and    al,0Fh        ; mask valid bits (lower nibble) = 0h?
DF8E 75 03      C      jnz    i_cal_err    ; if not, abort.
C
DF90 42          C      inc    dx            ; increment to next port
DF91 E2 F7      C      loop   i_cal_0
C      ENDIF
C
C      ; else, abort.
DF93          C      i_cal_err:
DF93 BE DA2E R   C      mov    si,cs:(offset i_calr_m)
DF96 E8 E540 R   C      call   DRomString
DF99 BE D9CC R   C      mov    si,cs:(offset fail_m)
DF9C E8 E540 R   C      call   DRomString      ; display fail message.
DF9F E8 E55F R   C      call   DCrLf
DFA2 EB 01      C      jmp    short i_cal_ok  ; try to write 1-1-80, regardless...
C
DFA4 F4          C      hlt
C
DFA5          C      i_cal_ok:
C
C      IF G4TOD
DFA5 B8 0001     C      mov    ax,1          ;%
DFA8 E6 7F      C      out    7Fh,al        ; select 24 hour mode%
C      ENDIF
DFAA 33 DB      C      xor    bx,bx          ; 1-1-80 is day 0.
DFAC 33 C9      C      xor    cx,cx          ; hours & minutes = 0.
C
C      ; Write & Start Clock Calendar Device.
C
C      ; bx = day (from 1-1 of leap year)
DFAE B4 FF      C      mov    ah,-1          ; ch = hour
DFB0 CD 1A      C      int    1Ah           ; cl = minutes
C
C      ; Output: ah = -1 implies date/time err.
C      ;          ah = 0 implies date/time OK.
C
DFB2          C      i_cal_end:
C
C      ;-----
C      ;          i8254 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
C
DFB2          C      i_rtc:
C
C      ; String to be displayed regardless of results.
C
DFB2 BE DA3B R   C      mov    si,cs:(offset i_rtc_m)
DFB5 E8 E540 R   C      call   DRomString
C
C      ; Test i8254 real-time clock interrupt p_timer counter (p_8253_0).
C
DFB8 B0 34      C      mov    al,034h        ; 00 11 010 0 -> p_8253_0, lsb 1st, mode 2, no BCD
DFBA BA 0040     C      mov    dx,p_8253_0    ; select real-time clock counter
DFBD E8 E1C0 R   C      call   rtc_chk
DFC0 75 2E      C      jnz    i_rtc_err    ; if nz, ah has error code to report.

```

```
C
C ; Test i8254 tone generator p_timer counter (p_8253_2).
C
DFC2 B0 01 C      mov     al,1           ; clear kb interrupts, reset kb, disable parity
DFC4 E6 61 C      out     p_kctrl,al      ; turn off speaker data -- bit #1
C                                     ; turn on speaker gate to p_8253_2 -- bit #0
C
DFC6 B0 B4 C      mov     al,0B4h        ; 10 11 010 0 -> p_8253_2, lsb 1st, mode 2, no BCD
DFC8 BA 0042 C      mov     dx,p_8253_2      ; select tone generator counter
DFCB E8 E1C0 R C      call    rtc_chk
C
DFCE B0 00 C      mov     al,0           ; clear kb interrupts, reset kb, disable parity
DFD0 E6 61 C      out     p_kctrl,al      ; turn off speaker data & gate -- bits #1 & #0
C
DFD2 75 1C C      jnz     i_rtc_err      ; if nz, ah has error code to report.
C
C ; Initialize i8254 real-time clock interrupt p_timer counter (p_8253_0).
C
DFD4 B0 36 C      mov     al,t0cmd          ; select real time clock counter
DFD6 E6 43 C      out     p_8253_ctrl,al
C
DFD8 B8 0000 C      mov     ax,t0count        ; load real time clock count
DFDB E6 40 C      out     p_8253_0,al
DFDD 8A C4 C      mov     al,ah
DFDF E6 40 C      out     p_8253_0,al
C
C ; Initialize i8254 tone generator p_timer counter (p_8253_2).
C
DFE1 B0 B6 C      mov     al,t2cmd          ; select tone generator counter
DFE3 E6 43 C      out     p_8253_ctrl,al
C
DFE5 B8 0266 C      mov     ax,t2count        ; load tone generator count
DFE8 E6 42 C      out     p_8253_2,al
DFEA 8A C4 C      mov     al,ah
DFEC E6 42 C      out     p_8253_2,al
C
DFEE EB 1C C      jmp     short i_rtc_ok
C
C
DFF0 C      i_rtc_err:          ; ah has error code to report.
DFF0 BE D9CC R C      mov     si,cs:(offset fail_m)
DFF3 E8 E540 R C      call    DRomString          ; display fail message.
C
DFF6 BE DA44 R C      mov     si,cs:(offset i_rtc_lo_m-(4*1))
DFF9 8A C4 C      mov     al,ah          ; al = error code = (1, 2 or, 3)
DFFB 32 E4 C      xor     ah,ah          ; ax = error code = (1, 2 or, 3)
DFFD D1 E0 C      shl     ax,1          ; ax = 2*(error code) = (2, 4, or 6)
DFFF D1 E0 C      shl     ax,1          ; ax = 4*(error code) = (4, 8, or 12)
E001 03 F0 C      add     si,ax          ; index to (LO, HI, or NR message.)
E003 E8 E540 R C      call    DRomString          ; display failing mode.
E006 E8 E55F R C      call    DCrLf
C
E009 EB 0D C      jmp     short i_rtc_end
C
E00B F4 C      hlt
```

```

C
E00C          C i_rtc_ok:
E00C BE D9BB R C      mov     si,cs:(offset pass_m)
E00F E8 E540 R C      call    DRomString
C
E012 B0 48    C      mov     al,48h           ; Check Point #8
E014 BA 0378  C      mov     dx,378h        ; parallel port data port address
E017 EE       C      out     dx,al         ; output " Running- Checkpoint 8"
C
E018          C i_rtc_end:
E018 B8 0E07  C      mov     ax,(0Eh*100h)+BEL ; beep keyboard
E01B CD 10    C      int     10h
C
C      assume cs:code, ds:nothing, es:nothing, ss:stack_ram
C
E01D E9 E2AD R C      jmp     pcinit
C
E020          C diagnostics_1 endp
C
E020          C code ends
C      include pwrup1a.asm
C
C      ;=====
C      ;      Filename:      pwrup1a.src
C      ;
C      ;=====
C
E020          C code      segment public 'ROM'
C      assume cs:code, ds:nothing, es:nothing, ss:nothing
C
E05B          C      ORG     0E05Bh           ;;
C
E05B          C i_hard_reset proc           ;;
E05B E9 DAD3 R C      jmp     diagnostics_1    ;;
E05E          C i_hard_reset endp         ;;
C
C      ;-----
C      ;      Install GDT ENTRIES
C      ;
C      ;      Input:  None.
C      ;      Output: None.
C      ;
C      ;      Trash: ax = cx = 0 destroyed.
C      ;-----
C
E05E          C i_gdt   proc   near
C      assume cs:code, ds:nothing, es:nothing, ss:stack_ram
C
E05E 1E       C      push  ds           ; save registers
E05F 06       C      push  es
E060 57       C      push  di
E061 56       C      push  si
C
C      ; Initialize the gdt with gdt_ent.

```

```

C
C          assume  cs:code, ds:code, es:data, ss:stack_ram
C
E062  8C C8      C      mov     ax,cs                ; satisfy assumptions
E064  8E D8      C      mov     ds,ax                ;
E066  B8 0040    C      mov     ax,data_seg            ;
E069  8E C0      C      mov     es,ax                ; es:di = data_seg:gdt
E06B  B8 00AA R   C      mov     ax,es:[offset gdt]
E06E  8B F8      C      mov     di,ax
E070  B8 E080 R   C      mov     ax,cs:(offset gdt_ent) ; ax = offset gdt_ent
E073  8B F0      C      mov     si,ax                ; si = offset gdt_ent
E075  B9 0010    C      mov     cx,16                ; loop 16 times for 16 words
C
E078  A5        C      i_gdt0: movsw             ; es:di++ gets ds:si
E079  E2 FD      C      loop   i_gdt0             ; until cx = 0.
C
E07B  5E        C      pop     si                ; restore registers
E07C  5F        C      pop     di
E07D  07        C      pop     es
E07E  1F        C      pop     ds
E07F  C3        C      ret
C
E080  0000 0000 0000 0000 C      gdt_ent    dw     0,0,0,0        ; first entry all 0's 0
E088  FFFF      C      dw     0ffffh            ; next entry 8h
E08A  0000      C      dw     0                ; to be filled in
E08C  0F        C      db     0fh              ; segment 0f000h
E08D  9A        C      db     9ah              ; kernel code protection
E08E  0000      C      dw     0                ; always 0
E090  FFFF      C      dw     0ffffh            ; next entry 10h
E092  0000      C      dw     0                ; offset is 0
E094  00        C      db     0                ; filled in by addr in program
E095  92        C      db     92h              ; kernel data
E096  0000      C      dw     0                ; always 0
E098  0020      C      dw     32               ; limit of gdt
E09A  04AA      C      dw     4aah            ; offset of gdt
E09C  00        C      db     0                ; segment of gdt
E09D  92        C      db     92h              ; kdata
E09E  0000      C      dw     0                ; always 0
E0A0                                C      i_gdt    endp
C
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
E0A0                                C      i_d_init  proc  near
C                                C      assume  cs:code, ds:nothing, es:nothing, ss:stack_ram
C
C ;: Here is the code for putting the DEB in Transparent Mode.
C

```

```

EOA0 50          C      push  ax
EOA1 52          C      push  dx
EOA2 BA 03DD    C      mov   dx,03DDh          ;; DEB I/O Address Register port address
EOA5 B0 01      C      mov   al,1             ;; select Mode Control Register
EOA7 EE          C      out   dx,al
C
EOA8 42          C      inc   dx             ;; DEB Mode Control Register address
EOA9 42          C      inc   dx
EOAA FE C8      C      dec   al             ;; set DEB Transparent Mode
EOAC EE          C      out   dx,al          ;;
EOAD 5A          C      pop   dx
EOAE 58          C      pop   ax
C
C
C
C ; Initialize both boards.
C
C      assume cs:code, ds:data, es:nothing, ss:stack_ram
C
EOAF 1E          C      push  ds             ; save registers
C
EOB0 B8 0040    C      mov   ax,data_seg      ; ds = ax = data_seg = 0040h.
EOB3 8E D8      C      mov   ds,ax          ; (ah = 0.)
C
C ; Initialize monochrome board.
C
EOB5 B0 30      C      mov   al,30h         ; switch_bits for monochrome.
EOB7 A3 0010 R  C      mov   word ptr ds:[switch_bits],ax ; set data for monochrome.
EOBA CD 10      C      INT   10h           ; ah = 0 = v_set_mode.
C
C ; Initialize color board.
C
EOBC B8 0003    C      mov   ax,0003h        ;switch_bits for not monochrome.
EOBF A3 0010 R  C      mov   word ptr ds:[switch_bits],ax ; set data for color.
EOC2 CD 10      C      INT   10h           ; ah = 0 = v_set_mode.
C
C ; Determine system configuration from switches (low byte of switch_bits).
C
EOC4 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
EOC6 24 F0      C      and   al,0F0h       ; mask off low nibble (keep high nibble)
C                                     ; of low byte; clear high byte.
EOC8 0C 0D      C      or    al,00Dh       ; ALWAYS 64k planar RAM and >= 1 FDU!
C
EOCA 8A C8      C      mov   cl,al          ; cl is ok, excepts bits #5 & #4.
EOCC B5 03      C      mov   ch,03h         ; initialize display to mode 3 (default).
C
EOCE 24 30      C      and   al,030h       ; isolate display switches(bits #5 & #4).
EOD0 74 2C      C      jz    i_d_ok         ; if zero, EGA adapter present.
C
C ; Initialize System Variables.
C ;; You don't have an EGA card so you can initialize the master table pointer
C

```

```
E0D2 C7 06 0084 R E297 R C      mov     word ptr ds:[master_tbl_ptr+0000h],cs:(offset mastab)
E0D8 8C 0E 0086 R      C      mov     word ptr ds:[master_tbl_ptr+0002h],cs
C
C
C
E0DC 3C 30      C      cmp     al,030h          ; is it the monochrome board?
E0DE 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
E0E0 1E      C      push   ds              ; save ds = data_seg = 0040h.
E0E1 B8 B000      C      mov     ax,para_mono   ; satisfy assumptions
E0E4 8E D8      C      mov     ds,ax
E0E6 B0 A5      C      mov     al,0A5h        ; test pattern
E0E8 A2 0000      C      mov     byte ptr ds:[0000h],al ; if so, is monochrome there?
E0EB 8A 26 0000      C      mov     ah,byte ptr ds:[0000h] ; read monochrome RAM
E0EF 1F      C      pop     ds              ; restore ds = data_seg = 0040h.
C
C      assume cs:code, ds:data, es:nothing, ss:stack_ram
C
E0F0 3A C4      C      cmp     al,ah          ; if monochrome RAM is there,
E0F2 75 04      C      jnz    i_d_80x25     ; then the board must be there!
C      ; if not, default to 80x25 color
E0F4 B5 07      C      mov     ch,07h        ; if there, we believe switches,
E0F6 EB 06      C      jmp    short i_d_ok   ; initialize display to mode #7.
C
E0F8      C      i_d_80x25:
E0F8 80 E1 EF      C      and     c1,0EFh      ; reset bit #4 for 80x25 color.
E0FB 80 C9 20      C      or      c1,020h      ; set bit #5 for 80x25 color.
C
E0FE      C      i_d_ok:
C
C      ; Set system configuration (switch_bits) from switches.
C
E0FE E4 66      C      in     al,sys_conf_a ; get switch info%
E100 A8 10      C      test    al,10h        ; 80287 present?%
E102 75 03      C      jnz    i_d_sw        ; no, move on%
E104 80 C9 02      C      or      c1,02        ; yes,record it%
E107      C      i_d_sw:
E107 32 E4      C      xor     ah,ah         ; ah = 0.
E109 8A C1      C      mov     al,c1         ; get data from switches.
E10B A3 0010 R      C      mov     word ptr ds:[switch_bits],ax ; save data from switches
C
C      ;-----
C      ;Test for and Initialize optional video ROMs
C      ;-----
C
C      assume cs:code, ds:nothing, es:nothing, ss:nothing
C
E10E BB C000      C      mov     bx,0C000h    ; load starting segment
C
E111      C      vrom_scan_loop:
E111 8E DB      C      mov     ds,bx        ; bx has pending segment
E113 33 F6      C      xor     si,si        ; offset 0000h
```

```

C
E115 81 3C AA55 C      cmp     word ptr ds:[si],0AA55h
E119 75 33      C      jne     vrom_scan_next
C
E11B B8 0040     C      mov     ax,data_seg           ; satisfy assumptions
E11E 8E C0      C      mov     es,ax               ; for es in rom_check
E120 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
E122 33 C0      C      xor     ax,ax                ; clear al
E124 8A 64 02   C      mov     ah,byte ptr ds:[si+2] ; ax = (ROM length/512) * 256
E127 D1 E0      C      shl     ax,1                 ; ax = (ROM length/512) * 512
C          ; ax = ROM length in bytes
E129 50          C      push    ax                  ; save ROM length
E12A B1 04      C      mov     cl,4
E12C D3 E8      C      shr     ax,cl                ; advance segment for next ROM
E12E 03 D8      C      add     bx,ax                 ; by the number of paragraphs
E130 59          C      pop     cx                  ; restore ROM length in cx
C
E131 E8 E52D R   C      call   rom_checksum_cnt      ; get the checksum of the cx-
C          ; byte ROM.
E134 74 03      C      jz     vrom_chksum_ok       ; OK if the checksum was zero
E136 EB 16 90    C      jmp    vrom_err               ; error the checksum wasn't zero
C
E139          C vrom_chksum_ok:
E139 53          C      push   bx                  ; save the segment for next ROM
C
E13A 26: C7 06 0067 R 0003 C      mov     word ptr es:[io_rom_init],0003h
E141 26: 8C 1E 0069 R   C      mov     word ptr es:[io_rom_seg],ds
E146 26: FF 1E 0067 R   C      call   dword ptr es:[io_rom_init] ; initialize the ROM
C
E14B 5B          C      pop     bx                  ; restore segment for next ROM
C
E14C EB 04      C      jmp    short vrom_scan_exit
C
E14E          C vrom_err:
E14E          C vrom_scan_next:
E14E 81 C3 0080     C      add     bx,(800h/10h)       ; add 2k to the pending segment
C
E152          C vrom_scan_exit:
E152 81 FB C800     C      cmp     bx,0C800h           ; are we done?
E156 7C B9      C      jnge   vrom_scan_loop       ; if not, continue
C
C ; Determine mode to initialize display monitor (from switches).
C
E158 A8 20      C      test    al,020h              ; does user want 40x25 color?
E15A 75 02      C      jnz    i_d_mode
E15C B5 01      C      mov     ch,01h               ; if so, initialize mode #1.
E15E          C i_d_mode:

```

```

C
C ; Initialize desire board (from switches).
C
E15E 8A C5 C      mov     al,ch                ; transfer display mode to al.
E160 CD 10 C      INT     10h                ; ah = 0 = v_set_mode.
C
E162 1F C      pop     ds                ; restore registers
E163 C3 C      ret
C
E164 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
E164 C      i_vector      proc    near
C                      assume  cs:code, ds:nothing, es:nothing, ss:stack_ram
C
E164 1E C      push    ds                ; save registers
E165 06 C      push    es
E166 57 C      push    di
E167 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
E168 33 FF C      xor     di,di                ; satisfy assumptions
E16A 8E DF C      mov     ds,di                ; ds = es = ax = abs0_seg = 0
E16C 8E C7 C      mov     es,di                ; es:di = abs0_seg:int00locn
E16E B8 FF23 R C      mov     ax,cs:(offset ill_int) ; ax = offset ill_int
E171 B9 0008 C      mov     cx,(07h-00h)+1 ; load INT's 00h through 07h.
C
E174 AB C      i_vec0: stosw                ; es:di++ gets offset ill_int
E175 8C 0D C      mov     word ptr ds:[di],cs ; es:di gets cs
E177 47 C      inc     di                ; di++
E178 47 C      inc     di
E179 E2 F9 C      loop    i_vec0                ; until cx = 0.
C
C                      ; load INT's 02h and 05h
E17B C7 06 0014 R FF54 R C      mov     word ptr ds:[int05locn],cs:(offset s_int)
E181 C7 06 0008 R F85F R C      mov     word ptr ds:[int02locn],cs:(offset n_int)
E187 C7 06 0018 R E638 R C      mov     word ptr ds:[int06locn],cs:(offset op_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
E18D 8C C8 C      mov     ax,cs                ; satisfy assumptions
E18F 8E D8 C      mov     ds,ax                ; ds = ax = cs

```



```

E191 BE FEF3 R      C      mov     si,cs:(offset i_vec_tbl)      ; ds:si = code_seg:i_vec_tbl
C                                                           ; es:di = abs0_seg:int08locn
E194 B1 18         C      mov     cl,(1Fh-08h)+1      ; load INT's 08h through 1Fh.
C
E196 A5           C      i_vec8: movsw      ; es:di++ gets ds:si (offset)
E197 AB           C      stosw      ; es:di++ gets ax = cs (segment)
E198 E2 FC         C      loop    i_vec8      ; until cx = 0.
C
C      ; Initialize Interrupt Vectors 20h and above to zero.
C
E19A 33 C0         C      xor     ax,ax      ; ax = 0
C                                                           ; es:di = abs0_seg:int20locn
E19C B9 01B8       C      mov     cx,((03F0h-0080h)/2)      ; clear 0:0080h to 0:03F0h
C                                                           ; don't blow away stack!
E19F F3/ AB        C      rep     stosw      ; es:di++ gets 0
C
E1A1 5E           C      pop     si      ; restore registers
E1A2 5F           C      pop     di
E1A3 07           C      pop     es
E1A4 1F           C      pop     ds
E1A5 C3           C      ret
C
E1A6              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
E1A6              C      i_pic_init    proc    near
C                                                           assume cs:code, ds:nothing, es:nothing, ss:stack_ram
C
E1A6 BA 0020       C      mov     dx,pic_0      ; dx = pic_0 (8259A 'control' port)
E1A9 B0 13         C      mov     al,pic_icw1    ; edge triggered, single, icw4 to follow
E1AB EE           C      out     dx,al
C
E1AC 42           C      inc     dx      ; dx = pic_1 (8259A 'data' port)
E1AD B0 08         C      mov     al,pic_icw2    ; interrupt vector base address
E1AF EE           C      out     dx,al
C                                                           ; since we are single mode (no slave), skip icw3
C
C                                                           ; dx = pic_1 (8259A 'data' port)
E1B0 B0 0D         C      mov     al,pic_icw4    ; not special fully nested, buffered,
E1B2 EE           C      out     dx,al      ; master, normal end_of_int, 8086 mode
C
E1B3 B0 FF         C      mov     al,pic_off_msk ; mask all interrupts off for now
E1B5 EE           C      out     dx,al      ; dx = pic_1 (8259A 'data' port)
E1B6 C3           C      ret
C
E1B7              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
E1B7 C i_out_mask      proc      near
C          assume  cs:code, ds:nothing, es:nothing, ss:stack_ram
C
E1B7 E6 21 C          out      pic_1,al          ;output interrupt mask pattern
E1B9 8A E0 C          mov      ah,al            ;save pattern for compar
E1BB E4 21 C          in       al,pic_1          ;get mask from 8259
E1BD 3A E0 C          cmp      ah,al            ;the same ?
E1BF C3 C          ret                          ;return flags = result of compare
C
E1C0 C i_out_mask      endp
C
C ;-----
C ;      8254 p_timer test for one p_timer counter channel
C ;
C ;      Input:  al      = 8254 p_timer control byte
C ;             dx      = port address of 8254 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
E1C0 C rtc_chk proc      near
C          assume  cs:code, ds:nothing, es:nothing, ss:nothing
C
E1C0 8A E0 C          mov      ah,al            ; save control byte for later.
E1C2 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
E1C5 8B D9 C          mov      bx,cx            ; bx gets all its bits set.
C
E1C7 E6 43 C          out      p_8253_ctrl,al        ; send i8254 p_timer control byte.
C
E1C9 32 C0 C          xor      al,al            ; al = 00h
E1CB EE C          out      dx,al            ; load low byte of p_timer count.
E1CC FE C0 C          inc      al              ; al = 01h
E1CE EE C          out      dx,al            ; load high byte of p_timer count.
C
E1CF C rtc_chk_reset_lp:
```

```

E1CF 8A C4      C      mov     al,ah           ; get control byte for read.
E1D1 24 C0      C      and     al,0C0h         ; mask off all but top 2 bits.
E1D3 E6 43      C      out     p_8253_ctrl,al ; send latching control byte for read.
C
E1D5 EC         C      in     al,dx           ; get low byte of p_timer count.
E1D6 22 D8      C      and     bl,al          ; 'and' low byte.
E1D8 EC         C      in     al,dx           ; get high byte of p_timer count.
E1D9 22 F8      C      and     bh,al          ; 'and' high byte.
C
E1DB 0B DB      C      or     bx,bx          ; is bx = 0?
E1DD 74 05      C      jz     rtc_chk_reset_ok ; if so, we're done.
E1DF E2 EE      C      loop   rtc_chk_reset_lp ; if not, continue reading.
C
C ; (Note: loops less than 16 times.)
C
E1E1           C      rtc_chk_reset_err:   ; time out.
E1E1 B4 03      C      mov     ah,3          ; Error #3. (No Response.)
E1E3 C3          C      ret                    ; return nz status (loop leaves zf ok).
C
E1E4           C      rtc_chk_reset_ok:   ;
C
C ; Register Bit Test (All Set): Count down from 0h (FFFFh+1) until all bits set.
C
E1E4 33 DB      C      xor     bx,bx          ; bx gets all its bits reset.
C
E1E6 8A C4      C      mov     al,ah           ; get control byte for load.
E1E8 E6 43      C      out     p_8253_ctrl,al ; send i8254 p_timer control byte.
C
E1EA 32 C0      C      xor     al,al          ; al = 00h
E1EC EE         C      out     dx,al          ; load low byte of p_timer count.
E1ED EE         C      out     dx,al          ; load high byte of p_timer count.
C
E1EE           C      rtc_chk_set_lp:     ;
E1EE 8A C4      C      mov     al,ah           ; get control byte for read.
E1F0 24 C0      C      and     al,0C0h         ; mask off all but top 2 bits.
E1F2 E6 43      C      out     p_8253_ctrl,al ; send latching control byte for read.
C
E1F4 EC         C      in     al,dx           ; get low byte of p_timer count.
E1F5 0A D8      C      or     bl,al          ; 'or' low byte.
E1F7 EC         C      in     al,dx           ; get high byte of p_timer count.
E1F8 0A F8      C      or     bh,al          ; 'or' high byte.
C
E1FA 83 FB FF   C      cmp     bx,0FFFFh        ; is bx = 0FFFFh?
E1FD 74 05      C      jz     rtc_chk_set_ok   ; if so, we're done.
E1FF E2 ED      C      loop   rtc_chk_set_lp   ; if not, continue reading.
C
C ; (Note: loops less than 16 times.)
C
E201           C      rtc_chk_set_err:     ; time out.
E201 B4 03      C      mov     ah,3          ; Error #3. (No Response.)
E203 C3          C      ret                    ; return nz status (loop leaves zf ok).
C
E204           C      rtc_chk_set_ok:     ;
C
C ; p_timer Time Window Test: Test p_timer versus CPU & see if it falls within spec.
C
E204 8A C4      C      mov     al,ah           ; get control byte for read.

```

ROM BIOS Listing

```

E206 24 C0      C      and    al,0C0h      ; mask off all but top 2 bits.
E208 E6 43      C      out    p_8253_ctrl,al  ; send latching control byte for read.
C
E20A EC         C      in     al,dx         ; get low byte of p_timer count.
E20B 8A D8      C      mov    bl,al         ; save low byte.
E20D EC         C      in     al,dx         ; get high byte of p_timer count.
E20E 8A F8      C      mov    bh,al         ; save high byte.
C
E210 8A C4      C      mov    al,ah         ; get control byte for read.
E212 24 C0      C      and    al,0C0h      ; mask off all but top 2 bits.
E214 E6 43      C      out    p_8253_ctrl,al  ; send latching control byte for read.
C
E216 EC         C      in     al,dx         ; get low byte of p_timer count.
E217 8A C8      C      mov    cl,al         ; save low byte.
E219 EC         C      in     al,dx         ; get high byte of p_timer count.
E21A 8A E8      C      mov    ch,al         ; save high byte.
C
E21C 2B D9      C      sub    bx,cx         ; calculate time difference.
C
C ; Do Time Range Checking (4 <= bx <= 14).
C
E21E B4 02      C      mov    ah,2          ; Error #2. (High above time window.)
E220 83 FB 0E   C      cmp    bx,14
E223 77 09      C      ja     rtc_chk_high   ; return nz status. (ja has zf reset.)
C
E225 FE CC      C      dec    ah            ; Error #1. (Low below time window.)
E227 83 FB 04   C      cmp    bx,4
E22A 72 02      C      jb     rtc_chk_low    ; return nz status. (jb has zf reset.)
C
E22C FE CC      C      dec    ah            ; Error #0. (No Error!) return z status.
C
E22E           C      rtc_chk_high:
E22E           C      rtc_chk_low:
E22E C3         C      ret
C
E22F           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 ;       NOTE:      If CX is zero and ZF is nz then parity error occurred.
C ;
C ;       Trash:  None.
C ; -----
C
E22F           C      memtst      proc    near
C ;                assume cs:code, ds:nothing, es:nothing, ss:nothing
C

```

```

E22F B9 8000      C      mov     cx,08000h      ; get word count
C                                     ;      (64k = 32k * 2 bytes/word)
E232 8E C2      C      mov     es,dx
E234 33 FF      C      xor     di,di           ; es:di = address
E236             C      memtst_w1:
E236 8B C7      C      mov     ax,di           ;data = offset
E238 AB         C      stosw
E239 E2 FB      C      loop    memtst_w1
C
E23B 8E DA      C      mov     ds,dx
E23D 33 DB      C      xor     bx,bx           ;ds:bx = address
E23F B9 8000      C      mov     cx,08000h      ; word count
E242             C      memtst_r1:
E242 8B 07      C      mov     ax,[bx]         ;read data
E244 3B C3      C      cmp     ax,bx           ;verify data
E246 75 2C      C      jne     memtst_err
E248 43         C      inc     bx
E249 43         C      inc     bx           ;next address
E24A E2 F6      C      loop    memtst_r1
C
E24C B9 8000      C      mov     cx,08000h      ; word count
E24F             C      memtst_w2:           ; address is already ok
E24F 8B C7      C      mov     ax,di           ; data = offset
E251 F7 D0      C      not     ax             ; complement it
E253 AB         C      stosw           ;fill memory
E254 E2 F9      C      loop    memtst_w2
C
E256 B9 8000      C      mov     cx,08000h      ; word count
C
E259             C      memtst_r2:
E259 8B 07      C      mov     ax,[bx]         ; read data
E25B F7 D0      C      not     ax             ; complement
E25D 3B C3      C      cmp     ax,bx           ; verify
E25F 75 0F      C      jne     memtst_err_c
C
E261 43         C      inc     bx           ; update address
E262 43         C      inc     bx
E263 E2 F4      C      loop    memtst_r2
E265 B8 0000      C      mov     ax,0           ; to clear memory
E268 B9 8000      C      mov     cx,08000h      ; word count
E26B F3/ AB      C      rep     stosw
E26D 0B C0      C      or      ax,ax           ;set ZF
C      ENDIF
E26F C3         C      ret
C
E270             C      memtst_err_c:
E270 F7 D0      C      not     ax             ; error during complemented
E272 F7 D3      C      not     bx             ; address test
C
E274             C      memtst_err:
E274 93         C      xchg    ax,bx           ;return registers as specified
E275 C3         C      ret
C
E276             C      memtst      endp
C

```

```
E276          C code ends
              C include pwrup0.asm
              C
              C ;=====
              C ;      Filename:      pwrup0.src
              C ;
              C ;      This module includes temporary hardware initialization.
              C ;
              C ;      includes      Diagnostics
              C ;                      Cold Boot
              C ;                      Device Drivers
              C ;
              C ;=====
              C
E276          C code segment public 'ROM'
              C extrn i13_ih:word ;%
              C assume cs:code, ds:nothing, es:nothing, ss:nothing
              C
              C
E276          C p0_data1 proc near
              C
E276 4F 70 74 69 6F 6E C opt_ROM_m db 'Optional ROM at ',NUL
          61 6C 20 52 4F 4D C
          20 61 74 20 00 C
              C
E287 03BC     C p_tbl dw prt_data_a ; printer in port address space
E289 0378     C dw prt_data_b ; always on mother board.
E28B 0278     C dw prt_data_c ; printer in port address space
E28D 0000     C dw 0 ; no printer
              C
E28F 0052     C scc_tbl dw scc_ctl_b ; rs232 SCC channel B
E291 0050     C dw scc_ctl_a ; rs232 SCC channel A
              C
              C ; Z8000 and Z8530
              C ; NOT USED in 6300 PLUS!!
E293 DAD3 R   C alt_ret dw diagnostics_1 ; 00h Z8000 restart sequence offset
E295 F000     C dw code_seg ; 04h Z8000 restart sequence segment
              C
E297 0016     C mastab dw ((mt_end)-(mastab)) ; 00h master table byte length
              C
E299 CBBF R   C dw kb_data_table ; 02h kb xlation table offset
E29B F000     C dw code_seg ; 04h kb xlation table segment
              C
E29D FA6E R   C dw font_lo_8x8 ; 06h 1st 128 char.s 8x8 font offset
E29F F000     C dw code_seg ; 08h 1st 128 char.s 8x8 font segment
              C
E2A1 C060 R   C dw font_lo_8x16 ; 0Ah 1st 128 char.s 8x16 font offset
E2A3 F000     C dw code_seg ; 0Ch 1st 128 char.s 8x16 font segment
              C
E2A5 0000     C dw 0 ; 0Eh 2nd 128 char.s 8x16 font offset
E2A7 0000     C dw 0 ; 10h 2nd 128 char.s 8x16 font segment
              C
E2A9 0000     C dw 0 ; 12h soft font utility offset
E2AB 0000     C dw 0 ; 14h soft font utility segment
              C
```

```

C                                     ; 16h etc...
E2AD C mt_end label word
C
E2AD 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
E2AD C pcinit proc near
C
C assume cs:code, ds:data, es:data, ss:stack_ram
C extrn bios_install:near
C
E2AD B8 0040 C mov ax,data_seg ; satisfy assumptions
E2B0 8E D8 C mov ds,ax
E2B2 8E C0 C mov es,ax
C
C ; Initialize Keyboard Controller.
C
E2B4 9C C pushf ; save flags and
E2B5 FA C cli ; disable interrupts
C
E2B6 BA 0061 C mov dx,p_kctrl ; dx = p_kctrl
E2B9 B0 40 C mov al,40h ; remove keyboard reset
E2BB EE C out dx,al
C
E2BC 33 C9 C xor cx,cx ; delay
E2BE E2 FE C loop $
C
E2C0 B4 01 C mov ah,1 ; enable self test
E2C2 E8 E4B3 R C call kb_cmd_send
E2C5 89 0E 0072 R C mov word ptr ds:[reset_flag],cx ;
C
E2C9 33 C9 C xor cx,cx ; delay
E2CB E2 FE C loop $
C ; Flush any keyboard scan code and store AAh if we get it.
C
E2CD C kb_flush:
E2CD E4 64 C in al,kb_status ; get 8041 status
E2CF A8 01 C test al,1 ; test output buffer bit
E2D1 74 09 C jz kb_flush_back ; jump if no character pending
E2D3 E4 60 C in al,p_kscan ; get scan code from data port
E2D5 3C AA C cmp al,0AAh ; verify keyboard present
E2D7 75 F4 C jne kb_flush
E2D9 A3 0072 R C mov word ptr ds:[reset_flag],ax ; keyboard present
E2DC C kb_flush_back:
E2DC E2 EF C loop kb_flush ;loop if zero
C
E2DE 33 C9 C xor cx,cx ; delay

```

```
E2E0 E2 FE          C      loop    $
C
C      ; Initialize System Variables.
C
C      ;;      mov     word ptr ds:[master_tbl_ptr+0000h],cs:(offset mastab)
C      ;;      mov     word ptr ds:[master_tbl_ptr+0002h],cs
C
C      ; Initialize Keyboard Driver Variables.
C
E2E2 B8 001E R      C      mov     ax,ds:(offset kb_buffer)      ; pointer to beginning of buffer
E2E5 A3 001A R      C      mov     word ptr ds:[buffer_head],ax  ; keyboard output pointer offset
E2E8 A3 001C R      C      mov     word ptr ds:[buffer_tail],ax  ; keyboard input pointer offset
E2EB A3 0080 R      C      mov     word ptr ds:[buffer_start],ax ; keeps beginning of buffer
C
E2EE 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
E2F4 80 26 0017 R DF C      and     byte ptr ds:[kb_flag],(not num_lock_mode)
E2F9 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
E2FE B4 05          C      mov     ah,05H                      ; Read keyboard type
E300 E8 E4B3 R      C      call    kb_cmd_send                 ; -- send command.
C
E303 33 C9          C      xor     cx,cx                        ; set up timeout count
E305          C      kb_type_wait:
E305 E4 64          C      in     al,kb_status                 ; get port status
E307 A8 01          C      test    al,1                         ; data byte available?
E309 75 05          C      jnz    kb_type_read                 ; if so, go read it ..
E30B E2 F8          C      loop   kb_type_wait                 ; else wait awhile longer.
E30D EB 11 90        C      jmp    kb_not_dlx                   ; timeout, default to non-dlx
C
E310          C      kb_type_read:
E310 E4 60          C      in     al,p_kscan                   ; read ID byte..
E312 A8 01          C      test    al,01H                       ; deluxe kbd. bit set?
E314 74 0A          C      jz     kb_not_dlx
C
C      ; 01H bit set, so initialize to Deluxe Keyboard.
C
E316 80 0E 0017 R 20 C      or     byte ptr ds:[kb_flag].num_lock_mode
E31B 80 0E 0018 R 01 C      or     byte ptr ds:[kb_flag_1],dlx_kb
C
E320          C      kb_not_dlx:
C
E320 9D            C      popf                                ; restore interrupt-
C                                          ; enable state.
C
C      ; Initialize Printer & Communication (RS-232) Driver Variables.
C
E321 B0 14          C      mov     al,14h                       ; printer default timeout = 20
E323 BF 0078 R      C      mov     di,ds:(offset printer_t_out)
E326 B9 0004        C      mov     cx,4
E329 F3/ AA         C      rep     stosb                        ; es:di gets al
```



```

C
E32B B0 01 C mov al,01h ; printer default timeout = 01
E32D BF 007C R C mov di,ds:(offset serial_t_out)
E330 B9 0004 C mov cx,4
E333 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
E335 8C C8 C mov ax,cs
E337 8E D8 C mov ds,ax ; satisfy assumptions
C
E339 32 DB C xor bl,bl ; clear high byte of switch_bits
C
E33B BF 0008 R C mov di,es:(offset printer_addr) ; es:di points at printer_base
E33E BE E287 R C mov si,ds:(offset p_tbl) ; addresses of printer ports
C
E341 C i_prt_loop:
E341 AD C lodsw ; ax gets ds:si port address.
E342 0B C0 C or ax,ax ; valid port address?
E344 74 14 C jz i_prt_exit ; exit if invalid port address
C
E346 8B D0 C mov dx,ax ; transfer to data register
E348 B0 A5 C mov al,0A5h ; load test pattern
E34A EE C out dx,al ; output test pattern.
E34B 86 C4 C xchg al,ah ; mov ah,al; trash al; & delay.
E34D EC C in al,dx ; input test pattern back.
E34E 3A C4 C cmp al,ah ; what we read = test pattern ?
E350 75 EF C jnz i_prt_loop ; if not, loop as port is absent
C ; else, printer port is present
E352 8B C2 C mov ax,dx ; retrieve port address
E354 AB C stosw ; es:di gets ax.
C
E355 80 C3 40 C add bl,040h ;add to high byte of switch_bits
C
E358 EB E7 C jmp i_prt_loop ; will go around loop 3 times
C
E35A C i_prt_exit:
C
C ; Determine Communication (RS-232) Configuration (INS8250's).
C
E35A BF 0000 R C mov di,es:(offset rs232_addr) ; es:di points at rs232_base
C
E35D BA 03FA C mov dx,com_id_a ; read interrupt I.D. register
E360 EC C in al,dx ; for first 8250 port.
E361 A8 F8 C test al,0F8h ; bits #3-7 are always low if
E363 75 07 C jnz i_no_com_a ; installed.
C
E365 B8 03F8 C mov ax,com_data_a ; if present, load address of
E368 AB C stosw ; first 8250 data port.
C ; es:di gets ax.
E369 80 C3 02 C add bl,002h ;add to high byte of switch_bits
C
E36C C i_no_com_a: ; es:di points next empty word

```

```
C
E36C BA 02FA C      mov    dx,com_id_b      ; read interrupt I.D. register
E36F EC      C      in     al,dx          ; for second 8250 port.
E370 A8 F8   C      test   al,0F8h         ; bits #3-7 are always low if
C ;         jnz   i_no_com_b     ; installed.%
E372 75 07   C      jnz   i_no_sccs       ; if not, don't load SCC table%
C
E374 B8 02F8 C      mov    ax,com_data_b     ; if present, load address of
E377 AB      C      stosw          ; second 8250 data port.
C          ; es:di gets ax.
E378 80 C3 02 C      add    bl,002h         ;add to high byte of switch_bits
C
E37B        C  i_no_sccs:              ; es:di points next empty word
C ; Determine Game Card Configuration.
C
E37B BA 0201 C      mov    dx,game_card     ; get game card address.
E37E EC      C      in     al,dx          ; bits #0-3 are low if installed
E37F A8 0F   C      test   al,0Fh         ;
E381 75 03   C      jnz   i_no_game_card  ; skip, if not present
C
E383 80 C3 10 C      add    bl,010h         ;add to high byte of switch_bits
C
E386        C  i_no_game_card:
C
C ; Initialize High Byte of switch_bits.
C
E386 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
E38B B0 FC   C      mov    al,11111100b    ; p_timer & kb only at this point.
E38D BA 0021 C      mov    dx,pic_1         ; now set proper interrupt mask
E390 EE      C      out    dx,al
C
C ; Send specific end of interrupt (SEOI) to pic 'command' port for keyboard.
C
E391 B0 61   C      mov    al,pic_seoi_1   ; specific end of interrupt
E393 BA 0020 C      mov    dx,pic_0         ; to pic 'command' port.
E396 EE      C      out    dx,al
E397 FB      C      sti                    ; enable interrupts
C
C ; Initialize Parallel Printer Interface.
C
E398 B4 01   C      mov    ah,1            ; initialize printer...
E39A 33 D2   C      xor    dx,dx          ; ...port 0
E39C CD 17   C      INT    17h
C
C ; (Z8530 not used in the 6300 PLUS)
C ; Initialize all 4 (2) Z8530 Serial Communication Controller.
C ; NOTE: Special function code (FF) for power up ONLY initialization of 8530
E39E B9 0004 C      mov    cx,4
E3A1        C  rs_init:
E3A1 B8 FFE3 C      mov    ax,111111111100011b ; initialize SCC RS-232 (FFE3h)
C          ; 9600 baud,none,1 stop & 8 data
```

```

E3A4 8B D1      C      mov     dx,cx                ; port number = loop - 1
E3A6 4A        C      dec     dx
E3A7 CD 14      C      INT     14h
E3A9 E2 F6      C      loop   rs_init
C
E3AB          C      dis_dmacc:
E3AB B0 01      C      mov     al,1
E3AD BA 0063    C      mov     dx,63h
E3B0 EE        C      out    dx,al
C
C ;-----
C ; Call internal HDU init code.
C ;-----
C
C      assume cs:code, ds:abs0, es:nothing, ss:stack_ram
C
E3B1 33 C0      C      xor     ax,ax                ; satisfy assumptions
E3B3 8E D8      C      mov     ds,ax
C
E3B5 E4 66      C      in     al,sys_conf_a        ;; port 66h.%
E3B7 A8 40      C      test    al,64                ;; test switch bit 7%(1-8 no 0-7)
E3B9 75 4A      C      jnz    i_hdu_ok            ;; if set, skip init
C
E3BB BE DABD R   C      mov     si,cs:(offset i_hdu_m)
E3BE E8 E540 R   C      call   DRomString           ; print test message
C
E3C1 E8 0000 E   C      call   bios_install         ; calls w.d. bios %
C
C      assume cs:code, ds:data, es:abs0, ss:stack_ram
C
E3C4 2E: 8E 1E E538 R C      mov     ds,word ptr cs:[set_ds_word] ; satisfy assumptions
E3C9 80 3E 0075 R 00 C      cmp     byte ptr ds:[hf_num],0      ; number of hard disks.
E3CE 75 35      C      jnz    i_hdu_ok            ; if ok, leave everything alone.
C
E3D0 BC 0100    C      mov     sp,100h             ; re-initialize stack
E3D3 FA        C      cli                       ; disable interrupts
C ;;: call i_vector           ; re-install old vectors for
E3D4 33 C0      C      xor     ax,ax
E3D6 8E C0      C      mov     es,ax
E3D8 26: C7 06 0034 R FF23 C      mov     word ptr es:[int00locn],cs:(offset ill_int) ; hard disk and
E3DF 26: 8C 0E 0036 R      C      mov     word ptr es:[int00locn+2],cs
E3E4 26: C7 06 004C R EC59 C      mov     word ptr es:[int13locn],cs:(offset fd_io) ; floppy disk and
E3EB 26: 8C 0E 004E R      C      mov     word ptr es:[int13locn+2],cs ; floppy disk and
C ; int 40h
E3F0 26: C7 06 0064 R F876 C      mov     word ptr es:[int19locn],cs:(offset bt_int) ; and the boot
E3F7 26: 8C 0E 0066 R      C      mov     word ptr es:[int19locn+2],cs ; and the boot
C ; interrupt 19h
C ; Initialize Interrupt Vectors 20h and above to zero.
E3FC 26: A3 0100    C      mov     word ptr es:[(4*40h)+0000h],ax ; ROM initialization messed up%
E400 26: A3 0102    C      mov     word ptr es:[(4*40h)+0002h],ax ; es:di = abs0_seg.int40locn
E404 FB        C      sti
E405          C      i_hdu_ok:
C
C

```

```

C ;-----
C ;Test for and Initialize optional ROMs
C ;-----
C
C         assume  cs:code, ds:nothing, es:nothing, ss:nothing
C
E405 BB C800 C         mov     bx,0C800h           ; load starting segment
C
E408                C rom_scan_loop:
E408 8E DB C         mov     ds,bx           ; bx has pending segment
E40A 33 F6 C         xor     si,si           ; offset 0000h
C
E40C 81 3C AA55 C         cmp     word ptr ds:[si],0AA55h
E410 75 43 C         jne     rom_scan_next
C
E412 BE E276 R C         mov     si,cs:(offset opt_ROM_m) ; indicate ROM detected
E415 E8 E540 R C         call    DRomString
C
E418 33 C0 C         xor     ax,ax
E41A E8 E578 R C         call    DHexLong           ; ds:ax points at ROM
C
E41D B8 0E20 C         mov     ax,(0Eh*100h)+' ' ; put out SPACE
E420 CD 10 C         INT     10h
C
E422 B8 0040 C         mov     ax,data_seg        ; satisfy assumptions
E425 8E C0 C         mov     es,ax             ; for es in rom_check
E427 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
E429 33 C0 C         xor     ax,ax             ; clear al
E42B 8A 64 02 C         mov     ah,byte ptr ds:[si+2] ; ax = (ROM length/512) * 256
E42E D1 E0 C         shl     ax,1             ; ax = (ROM length/512) * 512
C                                     ; ax = ROM length in bytes
E430 50 C         push    ax               ; save ROM length
E431 B1 04 C         mov     cl,4
E433 D3 E8 C         shr     ax,cl            ; advance segment for next ROM
E435 03 D8 C         add     bx,ax            ; by the number of paragraphs
E437 59 C         pop     cx              ; restore ROM length in cx
C
E438 E8 E52D R C         call    rom_checksum_cnt ; get the checksum of the cx-
C                                     ; byte ROM.
E43B 74 03 C         jz     rom_chksum_ok    ; OK if the checksum was zero
E43D E9 E51A R C         jmp    rom_err         ; error the checksum wasn't zero
C
E440                C rom_chksum_ok:
E440 53 C         push    bx              ; save the segment for next ROM
C
E441 26: C7 06 0067 R 0003 C         mov     word ptr es:[io_rom_init],0003h
E448 26: 8C 1E 0069 R C         mov     word ptr es:[io_rom_seg],ds
E44D 26: FF 1E 0067 R C         call    dword ptr es:[io_rom_init] ; initialize the ROM

```

```

C
E452 5B C pop bx ; restore segment for next ROM
C
E453 EB 04 C jmp short rom_scan_exit
C
E455 C rom_scan_next:
E455 81 C3 0080 C add bx,(800h/10h) ; add 2k to the pending segment
C
E459 C rom_scan_exit:
E459 81 FB F600 C cmp bx,0F600h ; are we done?
E45D 7C A9 C jnge rom_scan_loop ; if not, continue
C
C ; Clean Up after Option ROM's
C
E45F E8 E55F R C call DCrLf ;;
E462 FA C cli ; disable interrupts
E463 BA 0021 C mov dx,pic_1 ; get current interrupt mask
E466 EC C in al,dx
E467 24 FC C and al,11111100b ; p_timer & kb must be on at this point.
E469 EE C out dx,al
E46A FB C sti ; enable interrupts
C
C ;-----
C ; FDU Test
C ;-----
C
C ; Initialize Floppy Disk Controller and related Driver Variables
C
C assume cs:code, ds:abs0, es:nothing, ss:stack_ram
C
E46B 33 C0 C xor ax,ax ; initialize the disk routines
E46D 33 DB C xor bx,bx
E46F 33 C9 C xor cx,cx
E471 33 D2 C xor dx,dx
E473 CD 13 C INT 13h
C
C ; Dummy Disk Attachment Test to Spin Up Drive for INT 19h (boot-strap).
C
E475 BE DA96 R C mov si,cs:(offset i_fduA_m)
E478 E8 E540 R C call DRomString ; print test message
C
E47B BD 0003 C mov bp,3 ; loop counter
E47E C i_fdu_lp:
E47E B8 0201 C mov ax,0201h ; read one sector
C
E481 33 DB C xor bx,bx
E483 8E DB C mov ds,bx
E485 8E C3 C mov es,bx ; xfer_segment
E487 BB 7C00 C mov bx,7C00h ; xfer_offset
C
E48A B9 0001 C mov cx,0001h ; track 0; sector 1
E48D 33 D2 C xor dx,dx ; head 0; drive 0
C
E48F 55 C push bp ; save retry count
E490 50 C push ax ; save return registers

```

ROM BIOS Listing

```

E491 06          C      push   es
E492 CD 13      C      INT    13h          ; bx, cx, dx, & ds preserved
E494 07          C      pop    es          ; restore return registers
E495 58          C      pop    ax
E496 5D          C      pop    bp          ; restore retry count
C
E497 73 08      C      jnc    i_fdu_ok      ; error during read?
E499 4D          C      dec    bp          ; if so, decrement retry count
E49A 75 E2      C      jnz    i_fdu_lp      ; and try again
C
E49C BE DAB0 R  C      mov    si,cs:(offset i_fdu_not_m) ; drive not ready message.
E49F EB 03      C      jmp    short i_fdu_end
C
E4A1          C      i_fdu_ok:
E4A1 BE DAB4 R  C      mov    si,cs:(offset i_fdu_rdy_m) ; drive ready message.
C
E4A4          C      i_fdu_end:
E4A4 E8 E540 R  C      call   DRomString
C
E4A7 FB          C      sti
E4A8          C      i_init_end:
E4A8 BA 0378    C      mov    dx,0378h      ; printer port
E4AB B0 3F      C      mov    al,3Fh        ; OK status
E4AD EE          C      out    dx,al        ; tell mfg tester
C
E4AE E8 E55F R  C      call   DCrLf
C
E4B1 CD 19      C      INT    19h          ; go to boot-strap routine
C
E4B3          C      pcinit endp
C
C      ; Send command in AH to keyboard interface processor.  AX is used.
C
E4B3          C      kb_cmd_send proc near
C
E4B3          C      kb_cmd_wlup:
E4B3 E4 64      C      in    al,kb_status ; get 8041 port status
E4B5 A8 02      C      test   al,10b        ; ready to receive?
E4B7 75 FA      C      jnz    kb_cmd_wlup
C
E4B9 8A C4      C      mov    al,ah          ; ready, send command
E4BB E6 60      C      out    p_kscan,al
E4BD C3          C      ret
C
E4BE          C      kb_cmd_send endp
C
E4BE          C      code    ends
C      include pwrup2.asm
C
C      ;=====
C      ;      Filename:      pwrup2.src
C      ;
C      ;      This module includes 8259 Interrupt, Video Controller, 82087
C      ;      NPU, and 8254 & MM58274 Clock tests.
C      ;

```

```

C ;=====
C
E4BE C code segment public 'ROM'
C      assume cs:code, ds:nothing, es:nothing, ss:nothing
C
C ;=====
C ;      Note: We are called from ill_int ONLY (see vector.src), and
C ;            stack looks like this:
C ;
C ;                                     High Address
C ;      |-----| <-- sp before ill_int trap
C ;      (10) | return fsw flags |
C ;      |-----|
C ;      (0E) | return cs segment|
C ;      |-----|
C ;      (0C) | return ip offset |
C ;      |-----| <-- sp after ill_int trap
C ;      (0A) |      ax      |
C ;      |-----|
C ;      (08) |      ds      |
C ;      |-----| <-- sp after ill_int pushes
C ;      (06) | near call here |
C ;      |-----| <-- sp after ill_int calls ill_trap
C ;      (04) |      ax      |
C ;      |-----|
C ;      (02) |      dx      |
C ;      |-----|
C ;      (00) |      si      |
C ;      |-----| <-- sp after ill_trap pushes
C ;
C ;                                     Low Address
C ;=====
C
C      assume cs:code, ds:nothing, es:nothing, ss:nothing
C
E4BE C ill_trap      proc      near
C
C ; Turn off floppy disk drives.
C
E4BE 50 C      push ax ; save registers
E4BF 52 C      push dx
E4C0 56 C      push si
C
E4C1 8B F4 C      mov si,sp ; setup for test%
E4C3 36: 8E 5C 0E C      mov ds,word ptr ss:[si+0Eh] ; cs past si,dx,ax,ret,ds,ax,ip%
E4C7 36: 8B 74 0C C      mov si,word ptr ss:[si+0Ch] ; ip past si,dx,ax,ret,ds,ax%
E4CB 83 EE 02 C      sub si,2 ; point to possible INT instr%
E4CE 8A 04 C      mov al,byte ptr ds:[si] ; get illegal opcode number%
E4D0 3C CD C      cmp al,0CDh ; compare to sw intr opcode%
E4D2 75 31 C      jnz ill_tend ; exit if not a sw intr%
C
E4D4 E8 ED50 R C      call stop_disk ; destroys ax & dx
C
E4D7 E8 E509 R C      call ill_ln
E4DA BE D999 R C      mov si,cs:(offset ill_m1) ; part 1 of message
E4DD E8 E540 R C      call DRomString
C

```

ROM BIOS Listing

```

E4E0 8B F4          C      mov     si,sp
E4E2 36: 8E 5C 0E    C      mov     ds,word ptr ss:[si+0Eh]      ; cs past si,dx,ax,ret,ds,ax,ip
E4E6 36: 8B 74 0C    C      mov     si,word ptr ss:[si+0Ch]      ; ip past si,dx,ax,ret,ds,ax
C
E4EA 4E             C      dec     si                          ; si points to interrupt number
E4EB 8A 04          C      mov     al,byte ptr ds:[si]          ; print illegal interrupt number
E4ED E8 E589 R       C      call    DHexByte
E4F0 4E             C      dec     si                          ; si points to interrupt instr.
E4F1 8B C6          C      mov     ax,si                        ; save pointer
C
E4F3 BE D9B2 R     C      mov     si,cs:(offset ill_m2)        ; part 2 of message
E4F6 E8 E540 R     C      call    DRomString
C
E4F9 E8 E578 R     C      call    DHexLong                    ; print illegal cs:ip = ds:ax
C
E4FC BE D9B8 R     C      mov     si,cs:(offset ill_m3)        ; part 3 of message
E4FF E8 E540 R     C      call    DRomString
E502 E8 E509 R     C      call    ill_ln
C
E505              C      ill_tend:
E505 5E             C      pop     si                          ; restore registers
E506 5A             C      pop     dx
E507 58             C      pop     ax
E508 C3             C      ret
C
C
E509 E8 E55F R     C      ill_ln: call    DCrLf                    ; prints a line of '*'s
E50C B2 2A          C      mov     dl,42
C
E50E B8 0E2A        C      ill_lp: mov     ax,(0Eh*100h)+('*')
E511 CD 10          C      INT     10h
E513 FE CA          C      dec     dl
E515 75 F7          C      jnz     ill_lp
E517 EB 46 90        C      jmp     DCrLf
C
E51A              C      ill_trap      endp
C
E51A              C      code      ends
C      include pwrap3.asm
C
C      ;=====
C      ;      Filename:      pwrap3.src
C      ;
C      ;      This module includes 8041 keyboard, communication LSI, RAM, and
C      ;      optional ROM tests.
C      ;      This module also includes disk drive tests, system initialization,
C      ;      keyboard boot-strap options, and message routines.
C      ;
C      ;=====
C
E51A              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
E51A          C rom_err      proc      near
C              assume cs:code, ds:nothing, es:data, ss:nothing
C
E51A  8C D8          C          mov     ax,ds
E51C  26: 88 26 0015 R C          mov     byte ptr es:[mfg_err_flag],ah ; high byte of ROM address
C
E521  BE D9CC R      C          mov     si,cs:(offset fail_m) ; indicate ROM failed
E524  E8 E540 R      C          call    DRomString
E527  EB 36 90          C          jmp     DCrLf
C
E52A          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
E52A          C rom_checksum proc      near
C              assume cs:code, ds:nothing, es:nothing, ss:nothing
C
E52A  B9 2000          C          mov     cx,2000h
C
E52D          C rom_checksum_cnt:
E52D  33 C0          C          xor     ax,ax ; clear ah
C
E52F          C rom_checksum_loop:
E52F  AC          C          lodsb ; 12 al gets ds:si
E530  02 E0          C          add     ah,al ; 3
E532  E2 FB          C          loop  rom_checksum_loop ; 17
C
E534  0A E4          C          or     ah,ah
E536  C3          C          ret
C
E537          C rom_checksum endp
C
C
C ;=====
C ;      Utility Routines:
C ;

```

```

C ;      DRomString      DString      DCrLf      DColon
C ;      DHexLong       DHexWord     DHexByte    DHexNib
C ;      DNum           DNumW
C ;=====
C
E537      C p4_data1      proc      near
C
E537 90    C even
C
E538 0040  C set_ds_word    dw      data_seg      ; 2 bytes      = 0 clocks
C
E53A      C p4_data1      endp
C
C
E53A      C set_ds        proc      near      ; set ds to firmware data segment
C          assume cs:code, ds:nothing, es:nothing, ss:nothing
C
E53A 2E: 8E 1E E538 R  C          mov      ds,word ptr cs:[set_ds_word]    ; 5 bytes 2+9+6 = 17 clocks
E53F C3          C          ret                                ; 1 byte      = 8 clocks
C          ; -----
E540      C set_ds        endp
C ;=====
C ;      Display ASCII String Utilities
C ;=====
C
E540      C DRomString    proc      near      ; Displays NUL terminated string at cs:si
C          assume cs:code, ds:nothing, es:nothing, ss:nothing
C
E540 1E      C          push   ds      ; all registers saved
E541 0E      C          push   cs      ; ds gets cs
E542 1F      C          pop    ds
E543 E8 E548 R  C          call   DString
E546 1F      C          pop    ds      ; restore ds
E547 C3          C          ret
E548      C DRomString    endp
C
E548      C DString       proc      near      ; Displays NUL terminated string at ds:si
C          assume cs:code, ds:nothing, es:nothing, ss:nothing
C
E548 50      C          push   ax      ; all registers & flags saved
E549 53      C          push   bx
E54A 56      C          push   si
E54B 9C      C          pushf
E54C FC      C          cld          ; auto increment
E54D B3 01    C          mov     bl,1      ; select foreground color for grafix modes
E54F AC      C DS_lp: lodsb          ; al gets ds:si and si++
E550 0A C0    C          or     al,al      ; NUL ?
E552 74 06    C          je     DS_ret
E554 B4 0E      C          mov     ah,0Eh     ; tty emulator
E556 CD 10    C          INT    10h
E558 EB F5    C          jmp     short DS_lp
E55A      C DS_ret:
E55A 9D      C          popf          ; restore registers & flags
E55B 5E      C          pop    si
E55C 5B      C          pop    bx

```

```

E55D 58          C      pop    ax
E55E C3          C      ret
E55F            C      DString    endp
C
E55F            C      DCrLf      proc    near    ; Displays a CR & LF.
C                    assume  cs:code, ds:nothing, es:nothing, ss:nothing
E55F 50          C      push   ax            ; all registers preserved
E560 B8 0E0D     C      mov    ax,(0Eh*100h)+CR
E563 CD 10      C      INT    10h           ; tty emulator
E565 B8 0E0A     C      mov    ax,(0Eh*100h)+LF
E568 CD 10      C      INT    10h           ; tty emulator
E56A 58          C      pop    ax            ; restore ax
E56B C3          C      ret
E56C            C      DCrLf      endp
C
E56C            C      DColon     proc    near    ; Displays a ':'.
C                    assume  cs:code, ds:nothing, es:nothing, ss:nothing
E56C 50          C      push   ax            ; all registers preserved
E56D 53          C      push   bx
E56E B3 01      C      mov    bl,1           ; select foreground color for grafix modes
E570 B8 0E3A     C      mov    ax,(0Eh*100h)+':'
E573 CD 10      C      INT    10h           ; tty emulator
E575 5B          C      pop    bx            ; restore registers
E576 58          C      pop    ax
E577 C3          C      ret
E578            C      DColon     endp
C
C ;=====
C ;      Display Hexadecimal Number in ASCII Utilities
C ;=====
C
E578            C      DHexLong   proc    near    ; Displays ds:ax in ASCII
C                    assume  cs:code, ds:nothing, es:nothing, ss:nothing
E578 50          C      push   ax            ; all registers preserved
E579 8C D8      C      mov    ax,ds         ; display segment first
E57B E8 E582 R  C      call   DHexWord
C
E57E E8 E56C R  C      call   DColon       ; display a colon
C
E581 58          C      pop    ax            ; restore ax
C ;      jmp    short DHexWord ; fall through:  display offset second
C
E582            C      DHexLong   endp
C
E582            C      DHexWord   proc    near    ; Displays ax in ASCII
C                    assume  cs:code, ds:nothing, es:nothing, ss:nothing
E582 50          C      push   ax            ; all registers preserved
E583 8A C4      C      mov    al,ah
E585 E8 E589 R  C      call   DHexByte     ; display high byte first
E588 58          C      pop    ax            ; restore ax
C ;      jmp    short DHexByte ; fall through:  display low byte second
C
E589            C      DHexWord   endp
C
E589            C      DHexByte   proc    near    ; Displays al in ASCII

```

ROM BIOS Listing

```

C          assume cs:code, ds:nothing, es:nothing, ss:nothing
E589 50    C          push ax          ; all registers preserved
E58A D0 C8 C          ror  al,1
E58C D0 C8 C          ror  al,1
E58E D0 C8 C          ror  al,1
E590 D0 C8 C          ror  al,1          ; move high nibble to low nibble
E592 E8 E596 R C          call DHexNib          ; display high nibble in ASCII
E595 58    C          pop  ax          ; restore ax
C          ; jmp  short DHexNib      ; fall through: display low nibble in ASCII
C
E596      C DHexByte      endp
C
E596      C DHexNib        proc  near      ; Displays low nibble of al in ASCII
C          assume cs:code, ds:nothing, es:nothing, ss:nothing
E596 50    C          push ax          ; all registers preserved
E597 53    C          push bx
E598 B3 01  C          mov  bl,1          ; select foreground color for grafix modes
E59A 24 0F  C          and  al,0fh          ; clear high nibble
E59C 04 30  C          add  al,'0'
E59E 3C 39  C          cmp  al,'9'
E5A0 76 02  C          jbe  NibOk          ; '0' <= al <= '9' ?
E5A2 04 07  C          add  al,'A'-'0'-10
E5A4 B4 0E  C NibOk:  mov  ah,0Eh          ; tty emulator
E5A6 CD 10  C          INT  10h
E5A8 5B    C          pop  bx          ; restore registers
E5A9 58    C          pop  ax
E5AA C3    C          ret
E5AB      C DHexNib        endp
C
C          ;=====
C          ;          Display Decimal Number in ASCII Utilities
C          ;=====
C
E5AB      C DNum          proc  near      ; Displays decimal of ax in ASCII in min width
C          assume cs:code, ds:nothing, es:nothing, ss:nothing
E5AB 53    C          push bx          ; all registers preserved
E5AC 33 DB  C          xor  bx,bx          ; minimum width
E5AE E8 E5B3 R C          call DNumW          ; display ax
E5B1 5B    C          pop  bx          ; restore bx
E5B2 C3    C          ret
E5B3      C DNum          endp
C
E5B3      C DNumW         proc  near      ; Displays decimal of ax in ASCII of width bx
C          assume cs:code, ds:nothing, es:nothing, ss:nothing
C
E5B3 50    C          push ax          ; all registers preserved
E5B4 53    C          push bx
E5B5 51    C          push cx
E5B6 52    C          push dx
E5B7 56    C          push si
C
E5B8 BE 000A C          mov  si,10          ; decimal modulus
E5BB 33 C9  C          xor  cx,cx          ; clear digit counter
E5BD      C DNumW loop:
E5BD 33 D2  C          xor  dx,dx          ; dx:ax = decimal number

```

```

E5BF F7 F6      C      div    si          ; dh = 0
C              C              ; dl = remainder = (0-9)
C              C              ; ax = quotient = higher order digits
E5C1 52        C      push   dx          ; save the digit on stack
E5C2 41        C      inc    cx          ; increment count of what's on stack
E5C3 0B C0     C      or     ax,ax       ; are we done?
E5C5 75 F6     C      jnz   DNumW_loop
C
E5C7 2B D9     C      sub    bx,cx       ; subtract digit count from width
E5C9 76 08     C      jbe   DNumW_skip  ; skip spaces if bx is not > cx
C
E5CB          C      DNumW_spaces:
E5CB B8 0E20   C      mov    ax,(0Eh*100h)+' ' ; display a space
E5CE CD 10     C      INT   10h
E5D0 4B        C      dec    bx          ; decrement count of spaces
E5D1 75 F8     C      jnz   DNumW_spaces ; keep going
C
E5D3          C      DNumW_skip:
E5D3 B3 01     C      mov    bl,1        ; foreground color for grafix modes
E5D5 58        C      pop    ax          ; remove digit from stack
E5D6 04 30     C      add    al,'0'       ; convert to ASCII
E5D8 B4 0E     C      mov    ah,0Eh      ; display the digit
E5DA CD 10     C      INT   10h
E5DC E2 F5     C      loop  DNumW_skip
C
E5DE 5E        C      pop    si          ; restore registers
E5DF 5A        C      pop    dx
E5E0 59        C      pop    cx
E5E1 5B        C      pop    bx
E5E2 58        C      pop    ax
E5E3 C3        C      ret
E5E4          C      DNumW  endp
C
E5E4          C      enable_parity proc          ;;
C
E5E4 50        C      push   ax
E5E5 52        C      push   dx
E5E6 BA 3F60   C      mov    dx,p_trapce
E5E9 EE        C      out    dx,al          ;;; reset trapce upon powerup %%
C              C              ;;; this also needed as part%%
C              C              ;;; of the Above Board fix%%
E5EA E4 61     C      in     al,p_kctrl   ;; read B port. (61h)
E5EC 0C 30     C      or     al,30h       ;; enable bits 4 & 5.
E5EE E6 61     C      out    p_kctrl,al   ;;
E5F0 24 CF     C      and    al,0CFh      ;;
E5F2 E6 61     C      out    p_kctrl,al   ;;
E5F4 B0 80     C      mov    al,nmi_enable ;; 80h.
E5F6 E6 A0     C      out    nmi_enable_port,al ;; defined in sysdata.src (A0h)
E5F8 5A        C      pop    dx
E5F9 58        C      pop    ax
E5FA C3        C      ret
C
E5FB 50 61 72 69 74 79 C parity1_m db 'Parity error on system board',NUL ;;
      20 65 72 72 6F 72 C
      20 6F 6E 20 73 79 C

```

ROM BIOS Listing

```

73 74 65 6D 20 62    C
6F 61 72 64 00      C
E618 50 61 72 69 74 79 C parity2_m      db      'Parity error on expansion board',NUL  ;;
20 65 72 72 6F 72    C
20 6F 6E 20 65 78    C
70 61 6E 73 69 6F    C
6E 20 62 6F 61 72    C
64 00                C
C
E638                  C enable_parity  endp                ;;
C
C ;=====
C ;      INT 06H -- Illegal Opcode Interrupt Routine%
C ;
C ;      Purpose:      To intercept certain (well-used) illegal%
C ;                    opcodes. Note that this vector is installed%
C ;                    i_vector.%
C ;
C ;=====
C
E638                  C OP_INT  PROC    NEAR
C
E638 FA                C      cli                ; stop intrs%
C                        ; This routine checks for MOV CS<-AX and emulates it.%
E639 50                C      push  ax           ;%
E63A 56                C      push  si           ;%
E63B 06                C      push  es           ;%
E63C 83 C4 06          C      add   sp, 6        ;%
C
E63F 5E                C      pop   si           ;%
E640 07                C      pop   es           ;%
E641 26: 8B 04          C      mov   ax, es:[si]  ;%
E644 83 EC 0A          C      sub   sp, 0ah      ;%
E647 07                C      pop   es           ;%
E648 5E                C      pop   si           ;%
C
C                        ; Compare offending instruction with known opcode%
E649 2D C88E          C      sub   ax, 0c88eH   ;%
E64C 58                C      pop   ax           ;%
E64D 74 01            C      jz   movaxcs      ;%
C
E64F CF                C      iret              ;%
C
C
E650                  C movaxcs:
C                        ;replace cs on stack with ax%
E650 83 C4 04          C      ADD   SP, 4        ;%
E653 50                C      PUSH  AX           ;%
E654 83 EC 02          C      SUB   SP, 2        ;%
C
E657 50                C      PUSH  AX           ;%
C
C                        ;increment return address by two %
E658 83 C4 02          C      ADD   SP,2        ;%
E65B 58                C      POP   AX           ;%

```

```

E65C 05 0002      C      ADD    AX,2      ;%
E65F 50          C      PUSH   AX        ;%
E660 83 EC 02    C      SUB    SP,2      ;%
C
C
C      ;retore AX register%
E663 58          C      POP    AX        ;%
C
E664 CF          C      IRET   ;%
E665             C      OP_INT ENDP
C
E665             C      code  ends
C      include fdu6.asm
C
E665             C      code  segment public 'ROM'
C      assume cs:code, ds:data, es:nothing, ss:nothing
C      ;::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
C      ;
C      ;      file fdu6.asm
C      ;
C      ;::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
C
C
C      ;::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
C      ;
C      ;      Test for Retry
C      ;
C      ;      Routine checks for certain commands and error conditions
C      ;      If no retry is needed, the disk-state is made 'established'
C      ;      When a retry is needed, disk-state and f_head are changed.
C      ;      CY is set/cleared as appropriate
C      ;
C      ;      note: state transitions are odd.
C      ;
C      ;::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
C
E665             C      f_tstretry    proc    near
E665 50          C      push   ax        ; savefunction info%
E666 E8 F03 R    C      call   f_getdrv   ; get drive number in bl%
E669 8A 46 03   C      mov    al,f_command ; get original cmd%
E66C 3C 01      C      cmp    al,1       ; reset/status cmd?%
E66E 76 1F      C      jbe   f_jmpnr    ; yes, exit%
E670 3C 06      C      cmp    al,6       ; special cmds?%
E672 73 1B      C      jae   f_jmpnr    ; yes, no special handling here%
E674 A0 0041 R   C      mov    al,diskette_status ; hold status info%
E677 0A C0      C      or    al,al      ; test error msg%
E679 75 17      C      jnz   f_dskerr   ; jmp if error%
E67B 8A 87 0090 R C      mov    al,diskstate[bx] ; no errors, massage state%
E67F A8 10 90 90  C      test   al,ESTAB     ; is it established??%
E683 75 5C      C      jnz   f_noretry  ; yes it is, jump out%
E685 0C 10 90 90 C      or    al,ESTAB     ; no, state is now confirmed%
E689 04 03      C      add    al,3       ; state part was unestab, add w/o fear%
E68B 88 87 0090 R C      mov    diskstate[bx],al ; record new state%
E68F             C      f_jmpnr:      ; added for 127-byte jump to f_noretry %
E68F EB 50 90     C      jmp    f_noretry ; done
E692             C      f_dskerr:

```

```

C          ; work with error code here
C ; try using ESTABs states only (don't make an unESTAB state upon error%)
C ;     mov     al,diskstate[bx]      ; hold lastrate info%
C ;     test    al,ESTAB              ; was state established?%
C ;     jz     f_nopass              ; no, don't try again%
C ;     sub     al,3                  ; make it un-established%
C ;     and     al,NOT(ESTAB)        ; clr estab bit%
C ;     mov     diskstate[bx],al     ; store 'new' state%
C ;     jmp     f_pass2              ; run 2nd try at same rate%
E692      C f_nopass:
E692 E8 F5F6 R      C     call    f_drvswitch          ; disk error,nonspecial cmd,nonestab%
E695 0A C0          C     or     al,al                ; check for low density drive%
E697 74 48          C     jz     f_noretry            ; finished if 48tpi%
E699 E8 F672 R      C     call    chkspeed            ; chk speed of multi-speed drive%
E69C 73 43          C     jnc    f_noretry            ; exit if non-speed related error%
E69E 8A 87 0090 R   C     mov     al,diskstate[bx]    ; prepare to change state/speed%
E6A2 3C 74 90 90   C     cmp     al,E48M12D          ; test state%%
E6A6 74 06          C     jz     f_mk1212            ; next state%
E6A8 B0 74 90      C     mov     al,E48M12D          ; next state%
E6AB EB 04 90      C     jmp     f_trcont           ; message new state%
E6AE      C f_mk1212:
E6AE B0 15 90      C     mov     al,E12M12D          ; next state%
E6B1      C f_trcont:
E6B1 A8 20 90 90   C     test    al,DOUBLE           ; chk for double step state%
E6B5 74 07          C     jz     f_clrhead           ; jmp for 1-step%
E6B7 80 4E 01 80   C     or     byte ptr f_head,80h ; internal parm for double step%
E6BB EB 05 90      C     jmp     f_trcont0          ; cont%
E6BE      C f_clrhead:
E6BE 80 66 01 7F   C     and     byte ptr f_head,07fh ; clr MSB for single stepping%
E6C2      C f_trcont0:
E6C2 88 87 0090 R   C     mov     diskstate[bx],al    ; hold next state before cont%
E6C6 8A 87 0092 R   C     mov     al,diskstate[bx+2]  ; hold orig state%
E6CA 3A 87 0090 R   C     cmp     al,diskstate[bx]    ; test for last attempt%
E6CE 74 11          C     jz     f_noretry            ; if equal, operation dead, try no more%
E6D0 2A 87 0090 R   C     sub     al,diskstate[bx]    ; check for estab/unestab cmp%
E6D4 3C 13          C     cmp     al,(ESTAB OR 03h)  ; sub result for failure%
E6D6 74 09          C     jz     f_noretry            ; zero -> failure%
E6D8 80 26 003E R 80 C     and     seek_status,80h    ; clear drive bits to force seek%
C
E6DD      C f_pass2:
E6DD F9            C     stc     ; try again, flag failure%
E6DE EB 02 90      C     jmp     f_trdone           ; exit%
E6E1      C f_noretry:
C          ; work with error code here
E6E1 F8            C     clc     ; flag success %
E6E2      C f_trdone:
E6E2 58            C     pop     ax                 ; restore%
E6E3 C3              C     ret
E6E4      C f_tstretry    endp
C
E6E4      C code     ends

.LIST ;number 3 start
C include boot1.asm
C ;=====

```



```

C ;      Filename:      boot1.src
C ;
C ;      This module includes the ORG'd jump to INT 19h (boot2.src)
C ;
C ;=====
C
E6E4      C code   segment public 'ROM'
C          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 F876 R      C       jmp    bt_int                ; necessary jump for ORG
C
E6F5      C bt_jmp  endp
C
E6F5      C code ends
C          C include memx.asm
E6F5      C code   segment public 'ROM'
C          C       assume cs:code, ds:nothing, es:nothing, ss:nothing
C
E6F5      C add_mem_code  proc   far
C          C       assume cs:code, ds:nothing, es:nothing, ss:nothing
C
E6F5  80 FC 88      C       cmp    ah,88h                ; see if request for mem above 1 Mb
E6F8  75 0B          C       jne    cass
C          C       assume ds:data
E6FA  1E            C       push   ds
E6FB  2E: 8E 1E E538 R      C       mov    ds,word ptr cs:[set_ds_word] ; satisfy assumptions
E700  A1 00CA R      C       mov    ax,ds:[seg_fail]
E703  1F            C       pop    ds
E704  CF            C       iret
C
E705      C cass:
E705  F9            C       stc                        ; error
E706  B4 86          C       mov    ah,86h
E708  CA 0002        C       ret    2
C
E70B      C add_mem_code  endp
C
C
C
C
E70B      C code   ends
C          C include comm1.asm
C ;=====
C ;      Filename:      com1.src
C ;
C ;      This module, com2, and com3 supply INT 14h.
C ;
C ;=====
C
E70B      C code   segment public 'ROM'
C          C       assume cs:code, ds:nothing, es:nothing, ss:nothing

```

```
C
C ;-----
C ;     INS8250 Compatible Line Status Bits (ah) for Z8530 SCC Re-Mapping
C ;     (8530 not used on the 6300 PLUS)
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 ;     (8530 not used on the 6300 PLUS)
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 ;     (8530 not used on the 6300 PLUS)
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 ;     (8530 not used on the 6300 PLUS)
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 ;
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
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 ;      (8530 not used on the 6300 PLUS)
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
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). (8530 not used on
C ;                   the 6300 PLUS).

```

ROM BIOS Listing

```

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 E538 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 and 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      ired
C
C ;-----
C ;      INT 14h Jump Table
C ;-----
C
E77F E787 R      C  rs_tbl dw    com_init      ; ah = 00h for INS8250
E781 E8B6 R      C      dw    com_pb        ; ah = 01h for INS8250
E783 E8E2 R      C      dw    com_gb        ; ah = 02h for INS8250
E785 E87D R      C      dw    com_stat      ; ah = 03h for INS8250
C ;      dw    scc_init     ; ah = 00h for SCC Z8530%
C ;      dw    scc_pb      ; ah = 01h for SCC Z8530%
C ;      dw    scc_gb      ; ah = 02h for SCC Z8530%
C ;      dw    scc_stat    ; ah = 03h for SCC Z8530%
C
E787            C  serial_io    endp
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 ;                                     +-----+
C ;                                     IBIBIBIPIPISIDIDI
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
E787            C  com_init    proc    near                ; ah = 00h
C                   assume cs:code, ds:nothing, es:nothing, ss:nothing
C
E787 52          C      push   dx                ; save dx = com_data_x
C
E788 8A E8      C      mov    ch,al            ; save input parameters.
C
E78A B0 80      C      mov    al,080h         ; access divisor latch of
C                                     ; baud count register.
E78C 83 C2 03   C      add    dx,3            ; dx = com_lctl_x
E78F EE          C      out   dx,al           ; write to line control register

```

ROM BIOS Listing

```

E790 E8 E8AC R      C      call    rs_dly
C
E793 8A DD          C      mov     bl,ch                ; get input parameters.
E795 81 E3 00E0     C      and    bx,11100000b        ; get bits #5, 6, & 7 (clear bh)
E799 B1 04           C      mov     cl,4
E79B D2 EB          C      shr    bl,cl                ; move to bits #1,2,& 3
C                                     ; bx is word index
E79D 2E: 8B 87 E729 R C      mov     ax,word ptr cs:[bx+com_baud] ; get 8250 baud count
C
E7A2 5A             C      pop     dx                ; restore dx = com_data_x
E7A3 EE            C      out    dx,al              ; output low byte of baud rate
E7A4 E8 E8AC R      C      call    rs_dly
C
E7A7 8A C4         C      mov     al,ah              ; transfer high byte to low byte
E7A9 42             C      inc    dx                ; dx = com_int_x
E7AA EE            C      out    dx,al              ; output high byte of baud rate
E7AB E8 E8AC R      C      call    rs_dly
C
E7AE 8A C5         C      mov     al,ch                ; get input parameters.
E7B0 24 1F         C      and    al,00011111b        ; get bits #0 thru #4
E7B2 42             C      inc    dx
E7B3 42             C      inc    dx                ; dx = com_lctl_x
E7B4 EE            C      out    dx,al              ; write to line control register
E7B5 E8 E8AC R      C      call    rs_dly                ; disable access divisor latch,
C                                     ; set data & stop bits, & parity
C
E7B8 32 C0         C      xor    al,al                ; disable all interrupts!!
E7BA 4A             C      dec    dx
E7BB 4A             C      dec    dx                ; dx = com_int_x
E7BC EE            C      out    dx,al              ; write to interrupt ID register
C
E7BD 4A             C      dec    dx                ; dx = com_data_x
E7BE E9 E87D R      C      jmp     com_stat            ; return status
C
E7C1               C com_init      endp
C
E7C1               C code      ends
C include kb1.asm
C ;=====
C ;      Filename:      kb1.src
C ;
C ;      This module includes INT 09h & 16h.
C ;
C ;=====
C
E7C1               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.
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          ; interrupts come back disabled!
E849 74 FA      C      jz      k_read            ; loop until something in buffer
E84B E8 E86E R C      call   k_adv_ptr        ; move pointer to next position
E84E 89 1E 001A R C      mov    word ptr ds:[buffer_head],bx ; store value in variable
E852 EB EE      C      jmp     short k_ret
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 comm2.asm
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 ;      (8530 not used on the 6300 PLUS)
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?
C ; jz scc_stat ; if so, take jump%
C
E87D C com_stat: ; INS8250 read status routine.
C
C ; Get Line Status.
C
E87D 52 C push dx ; save dx = com_data_x
E87E 83 C2 05 C add dx,5 ; dx = com_lstat_x
E881 EC C in al,dx ; get line status
E882 8A E0 C mov ah,al ; line status comes back in high byte
C
C ; Get Modem Status.
C
E884 42 C inc dx ; dx = com_mstat_x
E885 EC C in al,dx ; get modem status in low byte
E886 5A C pop dx ; restore dx = com_data_x
E887 C3 C ret
C
C
C ;scc_stat: ; SCC Z8530 read status routine.
C ;
C ; Get Channel Status.
C
C ; xor ax,ax ; al = selects 0 ; ah = no errors
C ; out dx,al ; dx = scc_ctl_x
C ; call rs_dly
C ; in al,dx ; get channel status
C ;
C ; test al,scc_txd ; test for transmit ready
C ; jz scc_no_txd
C ;
C ; or ah,com_txd ; if so, set INS8250-compatible bit.
C ;scc_no_txd:
C ;
C ; test al,scc_rxd ; test for receive ready
C ; jz scc_no_rxd
C ;
C ; or ah,com_rxd ; if so, set INS8250-compatible bit.
C ;scc_no_rxd:
C ;
C ;; Get Error Status.
C ;
C ; mov al,1 ; get errors
C ; out dx,al ; dx = scc_ctl_x
C ; call rs_dly
C ; in al,dx ; get error status
```

```

C ;
C ;     test    al,scc_fe           ; test for framing error
C ;     jz     scc_no_fe
C ;
C ;     or     ah,com_fe           ; if so, set INS8250-compatible bit.
C ;scc_no_fe:
C ;
C ;     test    al,scc_pe           ; test for parity error
C ;     jz     scc_no_pe
C ;
C ;     or     ah,com_pe           ; if so, set INS8250-compatible bit.
C ;scc_no_pe:
C ;
C ;     test    al,scc_oe           ; test for overrun error
C ;     jz     scc_no_oe
C ;
C ;     or     ah,com_oe           ; if so, set INS8250-compatible bit.
C ;scc_no_oe:
C ;
C ;;; Set Modem Status.
C ;
C ;     mov     al,(com_dsr+com_cts) ; al = DSR and CTS
C ;     ret
C
E888 C rs_stat endp
C
C ;-----
C ;     Wait for Status of RS-232 Interface. (rs_ws)
C ;     (8530 not used on the 6300 PLUS)
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
E888 C rs_ws proc near
C ;     assume cs:code, ds:nothing, es:nothing, ss:nothing
C
E888 51 C     push    cx                ; save time-out
E889 33 DB C     xor     bx,bx          ; clear bx
E88B 87 CB C     xchg   cx,bx            ; BL now has rs232_time_out.
C
E88D C rs_ws_lp:
E88D 0A F6 C     or     dh,dh           ; are we an INS8250 chip?
E88F 75 06 C     jnz   rs_ws_com          ; if so, take jump

```

```
C
E891 32 C0      C      xor    al,al          ; al = selects 0 on SCC Z8530
E893 EE        C      out    dx,al          ; dx = scc_ctl_x
E894 E8 E8AC R  C      call   rs_dly
C
E897           C      rs_ws_com:
E897 EC        C      in    al,dx          ; get channel status
E898 8A F8      C      mov    bh,al        ; save status in BH.
E89A 22 C4      C      and    al,ah        ; mask bits we're waiting for
E89C 3A C4      C      cmp    al,ah        ; are they all on?
C
E89E 74 08      C      jz    rs_ws_exit    ; if so, exit with zf set
C
E8A0 E2 EB      C      loop  rs_ws_lp      ; inner loop
E8A2 FE CB      C      dec    bl
E8A4 75 E7      C      jnz  rs_ws_lp      ; outer loop
C
E8A6 0A E4      C      or    ah,ah        ; time-out, exit with zf reset
E8A8           C      rs_ws_exit:
E8A8 8A E7      C      mov    ah,bh        ; move status to AH.
E8AA 59        C      pop    cx          ; restore time-out
E8AB C3         C      ret
C
E8AC           C      rs_ws  endp
C
C
E8AC           C      rs_dly  proc  near
C                  assume  cs:code, ds:nothing, es:nothing, ss:nothing
C
E8AC 9C        C      pushf
E8AD 51        C      push  cx
E8AE B9 0008   C      mov   cx,8
E8B1 E2 FE     C      rs_lp: loop  rs_lp
E8B3 59        C      pop  cx
E8B4 9D        C      popf
E8B5 C3         C      ret
C
E8B6           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 ;      (8530 not used on the 6300 PLUS)
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
E8B6 C com_pb proc near ; ah = 01h
C assume cs:code, ds:nothing, es:nothing, ss:nothing
C
E8B6 52 C push dx ; save dx = com_data_x
E8B7 50 C push ax ; save character to output in al
C
E8B8 B0 03 C mov al,(com_rts+com_dtr) ; signal RTS & DTR
E8BA 83 C2 04 C add dx,4 ; dx = com_mctl_x
E8BD EE C out dx,al ; send to modem control register
C
E8BE B4 30 C mov ah,(com_dsr+com_cts) ; wait for DSR & CTS
E8C0 42 C inc dx
E8C1 42 C inc dx ; dx = com_mstat_x
E8C2 E8 E888 R C call rs_ws ; wait for modem status register
E8C5 75 13 C jnz rs_pbe ; if time-out, take jump
C
E8C7 B4 20 C mov ah,com_txd ; wait for transmit ready
E8C9 4A C dec dx ; dx = com_lstat_x
E8CA E8 E888 R C call rs_ws ; wait for line status register
E8CD 75 0B C jnz rs_pbe ; if time-out, take jump
C
E8CF 58 C pop ax ; restore character to output in al
E8D0 5A C pop dx ; restore dx = com_data_x
E8D1 8A D8 C mov bl,al ; save character input/output in bl
E8D3 E8 E87B R C call rs_stat ; get return status
E8D6 8A C3 C mov al,bl ; restore character input/output in al
E8D8 EE C out dx,al ; else, output the character
C
C ; exit for put and get byte
E8D9 C rs_pb_gb: ; if SCC Z8530, dx = scc_ctl_x
C
E8D9 C3 C ret
C
E8DA C rs_pbe: ; exit for put byte error
E8DA 5B C pop bx ; restore character to output in bl
E8DB 5A C pop dx ; if SCC Z8530, restore dx = scc_ctl_x
C ; else INS8250, restore dx = com_data_x
C
C ;; call rs_stat ; get return status
E8DC 8A C3 C mov al,bl ; restore character to output in al
E8DE 80 CC 80 C or ah,com_te ; indicate timeout error
E8E1 C3 C ret
C
E8E2 C com_pb endp
C
C ;scc_pb proc near ; ah = 01h
C ; assume cs:code, ds:nothing, es:nothing, ss:nothing

```

```

C ;
C ;   push   dx           ; save dx = scc_ctl_x
C ;   push   ax           ; save character to output in al
C ;
C ;   mov    ah,scc_txd    ; wait for transmit ready
C ;   call   rs_ws
C ;   jnz   rs_pbe        ; if time-out, take jump
C ;
C ;   pop    ax           ; restore character to output in al
C ;   inc   dx           ; dx = scc_data_x
C ;   out   dx,al        ; else, output the character
C ;
C ;   pop    dx           ; restore dx = scc_ctl_x
C ;   jmp   short rs_pb_gb
C ;
C ; scc_pb endp
C ;
C ; -----
C ;
C ;   INS8250 Get Byte (com_gb) & SCC Z8530 Get Byte (scc_gb)
C ;   (8530 not used on the 6300 PLUS)
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 ;
E8E2  C com_gb proc near           ; ah = 02h
      C   assume cs:code, ds:nothing, es:nothing, ss:nothing
      C
E8E2  52      C   push   dx           ; save dx = com_data_x
E8E3  B0 01   C   mov    al,com_dtr        ; signal DTR
E8E5  83 C2 04 C   add   dx,4           ; dx = com_mctl_x
E8E8  EE      C   out   dx,al        ; send to modem control register
      C
E8E9  B4 20   C   mov    ah,com_dsr        ; wait for DSR
E8EB  42      C   inc   dx
E8EC  42      C   inc   dx           ; dx = com_mstat_x
E8ED  E8 E888 R C   call  rs_ws            ; wait for modem status register
E8F0  75 0E   C   jnz   rs_gbe        ; if time-out, take jump
      C
E8F2  B4 01   C   mov    ah,com_rxd        ; wait for receive ready
E8F4  4A      C   dec   dx           ; dx = com_lstat_x
E8F5  E8 E888 R C   call  rs_ws            ; wait for line status register

```

```

E8F8 75 06      C      jnz    rs_gbe          ; if time-out, take jump
C
E8FA 80 E4 0E   C      and    ah,0Eh         ; Only interested on low nibble.
E8FD 5A         C      pop    dx             ; restore dx = com_data_x
E8FE EC        C      in     al,dx          ; else get character
E8FF C3         C      ret
C
C
E900           C      rs_gbe:             ; exit for get byte error
E900 5A         C      pop    dx             ; if SCC Z8530, restore dx = scc_ctl_x
C                                     ; else INS8250, restore dx = com_data_x
E901 80 CC 80   C      or     ah,com_te      ; indicate timeout error
E904 C3         C      ret
C
E905           C      com_gb  endp
C
C
C ;scc_gb proc  near          ; ah = 02h
C ;      assume cs:code, ds:nothing, es:nothing, ss:nothing
C
C ;      push  dx             ; save dx = scc_ctl_x
C
C ;      mov   ah,scc_rxd      ; wait for receive ready
C ;      call  rs_ws
C ;      jnz  rs_gbe          ; if time-out, take jump
C
C ;      inc  dx             ; dx = scc_data_x
C ;      in   al,dx           ; else get character
C ;
C ;      pop  dx             ; restore dx = scc_ctl_x
C ;      jmp  short rs_pb_gb
C ;
C ;scc_gb endp
E905           C      code  ends
C      include fdu7.asm
C ;::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
C ;
C ;      fdu7.asm
C ;
C ;::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
E905           C      code  segment public 'ROM'
C      assume cs:code, ds:data, es:nothing, ss:nothing
C ;::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
C ;
C ;      Set Format transfer rate
C ;
C ;      Routine sets the disk-state variable to an Established state.
C ;      CAUTION: routine sets the rate on the pertain drive. If this is
C ;      the current drive, and the rate is differnet than the current
C ;      rate, the existing media will be inaccessible!
C ;
C ;::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
C
E905           C      f_setfrmt  proc  near
E905 E8 F603 R   C      call  f_getdrv          ; get drive number in bl%

```



```

E95B          C f5_tmp:
E95B E8 EF4C R C      call  f_wait_one_ms
E95E E2 FB     C      loop  f5_tmp
E960 59       C      pop   cx
C
E961 8A 4E 00 C      mov   cl,f_drive      ; drive.%
E964 B0 01     C      mov   al,1           ;%
E966 D2 E0     C      shl   al,cl         ; mask for motor status.%
E968 8A E1     C      mov   ah,cl         ; drive number in ah.%
E96A B1 04     C      mov   cl,4           ;%
E96C D2 E0     C      shl   al,cl         ; motor on bit to high nibble.%
E96E 0C 0C     C      or    al,0Ch        ; set bits 2 & 3 (0000 1100).%
E970 0A C4     C      or    al,ah         ; drive bits ( 0 & 1).
E972 34 01     C      xor   al,01h        ; toggle drive bits ( 0 & 1).%
E974 BA 03F2   C      mov   dx,f_motor_port ;%
E977 EE       C      out   dx,al         ; turn on the motor.%
C
C              ; write a 1d to deselect a%
C              ; or 2c to deselect b%
E978 34 01     C      xor   al,01h        ; toggle drive bits ( 0 & 1).%
E97A BA 03F2   C      mov   dx,f_motor_port ;%
E97D 90       C      nop                ; for good measure%
E97E EE       C      out   dx,al         ; turn on the motor.%
C
C      ;end of tmp fix
C ENDIF
C
E97F          C f_ratedone:
E97F C3       C      ret
E980          C f_setrate   endp
C
E980          C code ends
C include kb2.asm
C
C ;=====
C ;      Filename:      kb2.src
C ;
C ;      This module includes INT 09h & 16h.
C ;
C ;=====
C
E980          C code      segment public 'ROM'
C              assume  cs:code, ds:nothing, es:nothing, ss:nothing
C
E987          C          ORG    0E987h
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

```

ROM BIOS Listing

```

C
E987 FB C sti ; re-enable interrupts
C
E988 50 C push ax
E989 53 C push bx
E98A 51 C push cx
E98B 52 C push dx
E98C 56 C push si
E98D 57 C push di
E98E 1E C push ds
E98F FC C cld ; clear direction flag
C ;; prokey fix?
E990 2E: 8E 1E E538 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 EBC9 R C call k_beep ; beep the speaker
E9AF E9 EA63 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
C ;;;      les    di,dword ptr ds:[master_tbl_ptr] ; es:di points to master table
E9B4 2E 8E 06 F000 C      mov    es,word ptr cs:[code_seg]      ;; ega2 fix
E9B9 BB F000      C      mov    bx,code_seg
E9BC 8E C3        C      mov    es,bx
E9BE BF CBBF R    C      mov    di,offset [kb_data_table]      ; es:di points to p_kscan table
C
E9C1 33 DB        C      xor    bx,bx                      ; clear bh
E9C3 33 C9        C      xor    cx,cx                      ; clear cl
E9C5 8A 2E 0017 R C      mov    ch,byte ptr ds:[kb_flag]
C
E9C9 8A D8        C      mov    bl,al                      ; bx = scan code make
E9CB D1 E3        C      shl    bx,1                       ; bx = 2*(scan code make)
E9CD 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
E9CF 4B          C      dec    bx                        ; bx = 4*(scan code make)-1
C
E9D0 F6 C5 08     C      test   ch,alt_shift              ; alt state?
E9D3 75 32       C      jnz    k_ix                      ; if so, has highest priority
C
E9D5 4B          C      dec    bx                        ; bx = 4*(scan code make)-2
C
E9D6 F6 C5 04     C      test   ch,cntrl_shift           ; control state?
E9D9 75 2C       C      jnz    k_ix                      ; if so, next highest priority
C
E9DB 4B          C      dec    bx                        ; bx = 4*(scan code make)-3
C
C ; Handle CapLk Case.
C
E9DC 3C 37       C      cmp    al,55                     ; 0 <= scan code <= (7*8)-1
E9DE 77 0C       C      ja    k_no_cap                  ; test NumLk case.
C
E9E0 F6 C5 40     C      test   ch,caps_lock_mode        ; caplock state?
E9E3 74 1C       C      jz    k_no_lock                 ; if not, test shift states
C
E9E5 E8 EB8C R    C      call   k_bit                     ; get kb_cap_flags bit in cf
E9E8 73 17       C      jnb   k_no_lock                 ; (jnc) swap if cf set
C
E9EA EB 0D        C      jmp    short k_lock              ; honor caps lock.
C
E9EC           C      k_no_cap:
C
C ; Handle NumLk Case.
C
E9EC 3C 47       C      cmp    al,71                    ; scan code >= 71?

```

ROM BIOS Listing

```

E9EE 72 11      C      jb      k_no_lock
E9F0 3C 53      C      cmp      al,83          ; scan code <= 83?
E9F2 77 0D      C      ja      k_no_lock
E9F4 F6 C5 20    C      test     ch,num_lock_mode          ; numlock state?
E9F7 74 08      C      jz      k_no_lock          ; if not, test shift states
C
E9F9           C      k_lock:          ; either caps or num lock.
E9F9 F6 C5 03    C      test     ch,(left_shift+right_shift) ; if so, and shift state
E9FC 74 09      C      jz      k_ix          ; also true
C
E9FE 4B         C      dec      bx          ; bx = 4*(scan code make)-4
E9FF EB 06      C      jmp      short k_ix          ; (base case)
C
EA01           C      k_no_lock:      ; neither caps or num lock.
EA01 F6 C5 03    C      test     ch,(left_shift+right_shift) ; (shift state)
EA04 75 01      C      jnz     k_ix
EA06 4B         C      dec      bx          ; bx = 4*(scan code make)-4
C      ;      jmp      short k_ix          ; (base case) fall through
C
EA07           C      k_ix:          ; bx = index into p_kscan table
EA07 D1 E3      C      shl      bx,1          ; bx = p_kscan word index
EA09 03 FB      C      add      di,bx          ; add p_kscan base & index
C
EA0B 26: 8B 15   C      mov      dx,word ptr es:[di]          ; get data word from table
C
EA0E 33 DB      C      xor      bx,bx          ; clear bh
EA10 8A DA      C      mov      bl,d1          ; move translated key to bx
EA12 F6 06 0018 R 01 C      test     byte ptr ds:[kb_flag_1],dlx_kb ; are we a deluxe keyboard
EA17 74 02      C      jz      k_xlat          ; key
C
EA19 8A DE      C      mov      bl,dh          ; move translated deluxe
C      ;      key to key
EA1B           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
EA1B 0A D2      C      or      dl,d1          ; deluxe scan codes are
EA1D 74 1C      C      jz      k_no_case          ; NEVER special cases!!!!
C
C      ; Test for special cases.
C
EA1F 80 FB C0    C      cmp      bl,0C0h          ; xlated byte special case?
EA22 72 17      C      jb      k_no_case          ; if not, handle unspecial case
EA24 80 FB D8    C      cmp      bl,0D8h          ; 0C0h <= xlated byte <= 0D8h
EA27 77 12      C      ja      k_no_case          ; if so, do the special function
C

```

```

C ; Test for 'break' of special case.
C
EA29 80 FB C8 C      cmp     b1,0C8h      ; is xlated byte a shift key?
EA2C 72 04 C      jb     k_jump      ; if so, do 'break' of shift key
C ; 0C0h <= xlated byte < 0C8h
EA2E 0A E4 C      or     ah,ah
EA30 78 2E C      js     k_none      ; nothing for 'break' of others.
C
EA32 C      k_jump:      ; jump to special case routine.
EA32 8B F3 C      mov     si,bx      ; if so, si gets special index
EA34 D1 E6 C      shl     si,1      ; make it a word index
EA36 2E: FF A4 EA6C R C      jmp     cs:[si+((offset k_case)-(2*0C0h))]
C
EA3B C      k_no_case:
C
C ; Test for 'break' of non special case.
C
EA3B 0A E4 C      or     ah,ah
EA3D 78 21 C      js     k_none      ; do nothing if 'break' of key.
C
C ; Test for 'unpause' case.
C
EA3F F6 06 0018 R 08 C      test    byte ptr ds:[kb_flag_1],pause_mode ; are we in hold state?
EA44 74 0B C      jz     k_no_hold   ; if not, continue.
EA46 3C 45 C      cmp     al,num_lock_key ; don't clear hold state
EA48 74 16 C      je     k_none      ; on num_lock_key (Pause).
EA4A 80 26 0018 R F7 C      and     byte ptr ds:[kb_flag_1],not pause_mode ; reset hold state bit &
EA4F EB 0F C      jmp     short k_none ; ignore the key.
C
EA51 C      k_no_hold:
C
C ; Test for deluxe scan code.
C
EA51 0A D2 C      or     dl,dl
EA53 75 04 C      jnz    k_no_xcode
C
EA55 8B C2 C      mov     ax,dx      ; move deluxe key to scan code
EA57 EB 02 C      jmp     short k_buf ; put ax into the buffer
C
EA59 C      k_no_xcode:
EA59 8A C3 C      mov     al,b1      ; move translated key to key
C
EA5B C      k_buf:      ; put ax into kb_buffer.
C ; try to put ax into kb_buffer.
EA5B E8 EBB3 R C      call    k_try      ; zf set (z) if buffer is full
EA5E 74 03 C      jz     k_nop      ; zf reset (nz) if buffer got ax
C
EA60 C      k_none:      ; scan code translate to nothing
C
C ; Send specific end of interrupt (SEOI) to pic 'command' port.
C
EA60 E8 EBAB R C      call    k_eoi      ; send specific end of interrupt
C
EA63 07 C      k_nop: pop     es      ; restore registers without issuing SEOI
EA64 1F C      pop     ds

```

ROM BIOS Listing

```

EA65 5F          C      pop    di
EA66 5E          C      pop    si
EA67 5A          C      pop    dx
EA68 59          C      pop    cx
EA69 5B          C      pop    bx
EA6A 58          C      pop    ax
EA6B CF          C      iret
C
C ;-----
C ;      Test for system reset sequence. 'make' only.
C ;-----
C
EA6C 80 E5 0C    C k_res: and    ch,(alt_shift+cntrl_shift)
EA6F 80 FD 0C    C      cmp    ch,(alt_shift+cntrl_shift)      ; is it CTL ALT shift?
EA72 75 EC          C      jne    k_none
C
C ; CTL ALT DEL system reset.
C
EA74 C7 06 0072 R 1234 C      mov    word ptr ds:[reset_flag],01234h      ; set flag for warm boot
EA7A E9 DAD3 R    C      jmp    diagnostics_1      ; re-boot
C
C ;-----
C ;      Pause waiting for another key. 'make' only.
C ;-----
C
EA7D          C k_pause:
EA7D 80 0E 0018 R 08 C      or     byte ptr ds:[kb_flag_1],pause_mode      ; set the pause bit.
C
EA82 E8 EBAB R    C      call   k_eoi      ; send specific end of interrupt
EA85 FB          C      sti     ; k_eoi clears interrupts
C ; we must undo that here
C
C ; Note: Video not disabled during vertical retrace.
C
EA86 80 3E 0049 R 07 C      cmp    byte ptr ds:[v_mode],7 ; never on a monochrome card
EA8B 74 0B          C      je     k_hold
C
EA8D 8B 16 0063 R    C      mov    dx,word ptr ds:[v_base6845] ; get 6845 pointer register
EA91 83 C2 04          C      add    dx,4 ; get 6845 mode control register
EA94 A0 0065 R    C      mov    al,byte ptr ds:[v_3x8] ; get the video mode last sent
EA97 EE          C      out   dx,al ; enable video
C
EA98 F6 06 0018 R 08 C k_hold: test byte ptr ds:[kb_flag_1],pause_mode ; test the pause bit.
EA9D 75 F9          C      jnz   k_hold ; loop until pause bit cleared
EA9F EB C2          C      jmp   short k_nop
C
C ;-----
C ;      Print Screen sequence. 'make' only.
C ;-----
C
EAA1 E8 EBAB R    C k_prt: call   k_eoi      ; send specific end of interrupt
EAA4 FB          C      sti     ; enable interrupts%
EAA5 CD 05          C      INT    5h ; issue print screen interrupt
EAA7 EB BA          C      jmp   short k_nop
C

```

```

C ;-----
C ;      Deluxe code put NUL into kb_buffer. 'make' only.
C ;-----
C
EAA9 B8 0300 C k_nul: mov    ax,0300h                ; NUL=X03
EAAC EB AD    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
EAAE B0 80    C k_ins: mov    al,insert_shift          ; INS toggle lock
EAB0 E8 EADB R C          call   k_4tog                    ; toggle 4 state
C
EAB3 0A E4    C          or     ah,ah                  ; is INS toggle 'make' ?
EAB5 78 A9    C          js     k_none                    ; if not, exit
EAB7 B8 5200  C          mov    ax,(insert_key*100h)+00h ; else, ax gets deluxe INS key
EABA EB 9F    C          jmp    k_buf                    ; put ax into the buffer
C
C
EABC B0 40    C k_cap: mov    al,caps_lock_shift        ; CAPS LOCK toggle lock
EABE E8 EADB R C          call   k_4tog                    ; toggle 4 state
EAC1 B1 01    C          mov    cl,00000001b            ; CAPS LOCK LED is bit #0.
EAC3 E8 EB6A R C          call   k_LED_cap                  ; send LED information
EAC6 EB 98    C k_non1: jmp    short k_none
C
C
EAC8 B0 20    C k_num: mov    al,num_lock_shift         ; NUM LOCK toggle lock
EACA E8 EADB R C          call   k_4tog                    ; toggle 4 state
EACD B1 02    C          mov    cl,00000010b           ; NUM LOCK LED is bit #1.
EACF E8 EB60 R C          call   k_LED_num                    ; send LED information
EAD2 EB F2    C          jmp    short k_non1
C
C
EAD4 B0 10    C k_scr: mov    al,scr1_lock_shift        ; SCROLL LOCK toggle lock
EAD6 E8 EADB R C          call   k_4tog                    ; toggle 4 state
EAD9 EB EB    C          jmp    short k_non1
C
C
EADB 0A E4    C k_4tog: or     ah,ah                  ; is toggle shift 'make' ?
EADD 78 0F    C          js     k_4res                    ; if 'break' reset kb_flag_1
C
EADF 84 06 0018 R C          test   byte ptr ds:[kb_flag_1],al      ; if 'make', test bit.
EAE3 75 08    C          jnz   k_4ret                    ; return if already pressed
C
EAE5 08 06 0018 R C          or     byte ptr ds:[kb_flag_1],al  ; set the bit.
EAE9 30 06 0017 R C          xor    byte ptr ds:[kb_flag],al      ; toggle kb_flag history.
EAED C3        C k_4ret: ret
C
EAEF F6 D0    C k_4res: not   al
EAF0 20 06 0018 R C          and   byte ptr ds:[kb_flag_1],al      ; if 'break', reset bit only.
EAF4 C3        C          ret
C
C ;-----
C ;      Two state shifts. 'make' & 'break' in kb_flag only.

```

```

C ;-----
C
EAF5 B0 08 C k_alt: mov al,alt_shift ; ALT set/reset kb_flag
EAF7 E8 EB16 R C call k_2tog ; toggle 2 state
C
EAF8 33 C0 C xor ax,ax ; scan code of ah = 0.
EAF9 86 06 0019 R C xchg al,byte ptr ds:[alt_input] ; alt_input gets 0.
EBA0 0A C0 C or al,al ; was alt_input 0?
EBA1 74 C2 C je k_non1 ; if so, do nothing.
EBA2 E9 EA5B R C jmp k_buf ; else, put it into buffer.
C
C
EBA7 B0 04 C k_ctl: mov al,ctrl_shift ; CTL set/reset kb_flag
EBA8 EB 06 C jmp short k_2ret ; toggle 2 state and return
C
C
EBA9 B0 02 C k_lsh: mov al,left_shift ; LEFT SHIFT set/reset kb_flag
EBA0 EB 02 C jmp short k_2ret ; toggle 2 state and return
C
C
EBAF B0 01 C k_rsh: mov al,right_shift ; RIGHT SHIFT set/reset kb_flag
C ; jmp short k_2ret ; fall through
C
C
EBA11 C k_2ret:
EBA11 E8 EB16 R C call k_2tog ; toggle 2 state
EBA14 EB B0 C jmp short k_non1
C
EBA16 0A E4 C k_2tog: or ah,ah ; is set/reset shift 'make' ?
EBA18 78 05 C js k_2res ; if 'break', reset bit only.
C
EBA1A 08 06 0017 R C or byte ptr ds:[kb_flag],al ; if 'make', set bit only.
EBA1E C3 C ret
C
EBA1F F6 D0 C k_2res: not al
EBA21 20 06 0017 R C and byte ptr ds:[kb_flag],al ; if 'break', reset only.
EBA25 C3 C ret
C
C ;-----
C ; Alternate Numeric Keypad. 'make' only.
C ;-----
C
EBA26 41 C k_alt9: inc cx ; Alternate Numeric Keypad #9
EBA27 41 C k_alt8: inc cx ; Alternate Numeric Keypad #8
EBA28 41 C k_alt7: inc cx ; Alternate Numeric Keypad #7
EBA29 41 C k_alt6: inc cx ; Alternate Numeric Keypad #6
EBA2A 41 C k_alt5: inc cx ; Alternate Numeric Keypad #5
EBA2B 41 C k_alt4: inc cx ; Alternate Numeric Keypad #4
EBA2C 41 C k_alt3: inc cx ; Alternate Numeric Keypad #3
EBA2D 41 C k_alt2: inc cx ; Alternate Numeric Keypad #2
EBA2E 41 C k_alt1: inc cx ; Alternate Numeric Keypad #1
EBA2F C k_alt0: ; Alternate Numeric Keypad #0
C
EBA2F B0 0A C mov al,10
EBA31 F6 26 0019 R C mul byte ptr ds:[alt_input] ; alt_input = (10*alt_input)+c1

```



```

EB35 02 C1      C      add    al,c1
EB37 A2 0019 R  C      mov    byte ptr ds:[alt_input],al
EB3A EB 8A      C      jmp    short k_non1
C
C ;-----
C ;      Double Zero on Keypad. 'make' only.
C ;-----
C
EB3C B0 30      C k_00:  mov    al,'0'                ; try to put ax into kb_buffer.
EB3E E8 EBB3 R  C      call   k_try                    ; zf set (z)  if buffer is full
EB41 74 03      C      jz     k_nop1                  ; zf reset (nz) if buffer got ax
EB43 E9 EA5B R  C      jmp    k_buf                    ; put it into buffer, again.
C
EB46 E9 EA63 R  C k_nop1: jmp    k_nop
C
C ;-----
C ;      Break key sequence. 'make' only.
C ;-----
C
EB49 BB 001E R  C k_brk:  mov    bx,ds:(offset kb_buffer)
EB4C 89 1E 001A R C      mov    word ptr ds:[buffer_head],bx ; reset buffer to empty
EB50 89 1E 001C R C      mov    word ptr ds:[buffer_tail],bx
EB54 C6 06 0071 R 80 C      mov    byte ptr ds:[bios_break],80h ; turn on bios_break bit
EB59 CD 1B      C      INT    1Bh                    ; break interrupt vector
EB5B 33 C0      C      xor    ax,ax                  ; ax gets deluxe 00h
EB5D E9 EA5B R  C      jmp    k_buf                    ; put ax into the buffer
C
EB60           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 ;             c1 = 00000001b for CAPS LOCK LED or 00000010b for NUM LOCK LED.
C ;      Output: None.
C ;
C ;      Trash: al & c1 destroyed.
C ;-----
C
EB60           C k_LED_num  proc  near
C                   assume  cs:code, ds:data, es:nothing, ss:nothing
C
EB60 F6 06 0018 R 01 C      test   byte ptr ds:[kb_flag_1],dlx_kb ; are we a deluxe keyboard
EB65 74 03      C      jz     k_LED_cap            ; if kb, LED is num_lock
EB67 80 F1 80      C      xor    c1,10000000b          ; if deluxe kb, LED is
C                                     ; ~num_lock
C
EB6A           C k_LED_cap:
EB6A 0A E4      C      or     ah,ah                ; is CAPS/NUM LOCK 'make' ?
EB6C 78 1D      C      js     k_LED_ret            ; if not, exit
C
EB6E 84 06 0017 R  C      test   byte ptr ds:[kb_flag],al    ; is CAPS/NUM LOCK kb_flag set ?
EB72 74 03      C      jz     k_LED_cmd            ; if not, LED data is ok.
EB74 80 F1 80      C      xor    c1,10000000b          ; else, flip sense of LED data.

```

```

C
EB77          C k_LED_cmd:                ; polling loop to send command.
EB77 E4 64    C      in      al,kb_status        ; get 8041 status
EB79 A8 02    C      test     al,00000010b         ; test input buffer bit
EB7B 75 FA    C      jnz     k_LED_cmd            ; if not ok to write cmd, loop.
C
EB7D B0 13    C      mov     al,013h                ; keyboard 'LED' command.
EB7F E6 60    C      out     p_kscan,al           ; send keyboard 'LED' command.
C
EB81          C k_LED_dat:                ; polling loop to send data.
EB81 E4 64    C      in      al,kb_status        ; get 8041 status
EB83 A8 02    C      test     al,00000010b         ; test input buffer bit
EB85 75 FA    C      jnz     k_LED_dat            ; if not ok to write data, loop.
C
EB87 8A C1    C      mov     al,cl                ; retrieve keyboard 'LED' data.
EB89 E6 60    C      out     p_kscan,al           ; send keyboard 'LED' data.
EB8B          C k_LED_ret:
EB8B C3       C      ret
C
EB8C          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
EB8C          C k_bit  proc  near
C             assume cs:code, ds:data, es:nothing, ss:nothing
C
EB8C 8B F3    C      mov     si,bx                ; save bx
C
EB8E 33 DB    C      xor     bx,bx                ; clear bh
EB90 8A D8    C      mov     bl,al                ; bx = scan code (make) index
C                                     ; bx = 00000000 00xxxxyy
C
EB92 B1 03    C      mov     cl,3                ; rotate right bx 3
EB94 D3 CB    C      ror     bx,cl                ; bx = yyy00000 00000xxx
C
EB96 B1 03    C      mov     cl,3                ; rotate left bh 3
EB98 D2 C7    C      rol     bh,cl                ; bx = 00000yyy 00000xxx
C                                     ; bh = remainder = (0-7)
C                                     ; bl = quotient = (0-6)
C
EB9A 8A CF    C      mov     cl,bh                ; cl = remainder = (0-7)
EB9C 32 FF    C      xor     bh,bh                ; bx = quotient = (0-6)
C
EB9E 26: 8A 59 F9 C      mov     bl,byte ptr es:[di+bx-7] ; bl = proper cap_flags byte
C

```

```

EBA2 D2 D3      C      rcl    bl,cl          ; rotate (0-7) times into bit #7
EBA4 32 C9      C      xor    cl,cl          ; cl = 0
EBA6 D0 D3      C      rcl    bl,1          ; rotate into cf bit #7
                C
EBA8 8B DE      C      mov    bx,si          ; restore bx
EBAA C3         C      ret
                C
EBAB            C      k_bit  endp
                C
                C ;-----
                C ;      Input:  None.
                C ;      Output: None.
                C ;
                C ;      Trash:  al & dx destroyed.
                C ;-----
                C
EBAB            C      k_eoi  proc   near
                C
                C ; Send specific end of interrupt (SEOI) to pic 'command' port.
                C
EBAB FA         C      cli                    ; disable interrupts
EBAC B0 61      C      mov    al,pic_seoi_1      ; specific end of interrupt
EBAE BA 0020    C      mov    dx,pic_0          ; command to pic 'command' port.
EBB1 EE         C      out    dx,al
                C ;      sti                    ; enable interrupts
EBB2 C3         C      ret
                C
EBB3            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
EBB3            C      k_try  proc   near
                C      assume cs:code, ds:data, es:nothing, ss:nothing
                C
EBB3 8B 1E 001C R C      mov    bx,word ptr ds:[buffer_tail] ; get buffer end pointer
EBB7 8B F3      C      mov    si,bx          ; save the value
EBB9 E8 E86E R  C      call   k_adv_ptr        ; advance the tail
                C
EBBC 3B 1E 001A R C      cmp    bx,word ptr ds:[buffer_head] ; has the buffer wrapped around
EBC0 74 07      C      je    k_beep
                C ;      zf is reset (nz)
EBC2 89 04      C      mov    word ptr ds:[si],ax      ; store the value
EBC4 89 1E 001C R C      mov    word ptr ds:[buffer_tail],bx ; move the pointer up
EBC8 C3         C      ret                    ; return zf reset (nz)
                C
EBC9            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
EBC9      C k_beep  proc   near
          C         assume cs:code, ds:data, es:nothing, ss:nothing
          C
          C ;      call   k_eoi                ; send specific end of interrupt
          C
EBC9 BA 0061 C         mov    dx,p_kctrl          ; get kb control port address
EBCC EC      C         in     al,dx          ; get control data
EBCD 8A E0   C         mov    ah,al            ; save control data
          C
EBCF B3 80   C         mov    bl,80h            ; outer loop counter
          C
EBD1 24 FC   C k_lp:   and    al,0FCh          ; turn off speaker data
EBD3 EE      C         out   dx,al
          C
EBD4 B9 0048 C         mov    cx,48h            ; set up count
EBD7 E2 FE   C         loop   $                ; delay awhile
          C
EBD9 0C 02   C         or     al,02h          ; turn on speaker
EBDB EE      C         out   dx,al
          C
EBDC B9 0048 C         mov    cx,48h            ; set up count
EBDF E2 FE   C         loop   $                ; delay awhile
          C
EBE1 FE CB   C         dec    bl                ; decrement outer loop counter
EBE3 75 EC   C         jnz   k_lp
          C
          C ;      zf is set (z)
EBE5 8A C4   C         mov    al,ah            ; restore control data
EBE7 EE      C         out   dx,al
EBE8 E8 EBAB R C         call   k_eoi                ; send specific end of interrupt
EBEB C3      C         ret                    ; return zf set (z)
          C
EBEC      C k_beep   endp
          C
          C
EBEC      C k_data1 proc
          C
EBEC EAAE R   C k_case  dw    k_ins          ; kbins (0C0h)      ^
EBEE EABC R   C         dw    k_cap          ; kbcap           |
EBF0 EAC8 R   C         dw    k_num          ; kbnum           |
EBF2 EAD4 R   C         dw    k_scr          ; kbscr           |
EBF4 EAF5 R   C         dw    k_alt          ; kbalt          'make' & 'break'
EBF6 EB07 R   C         dw    k_ctl          ; kbctl           |
EBF8 EB0B R   C         dw    k_lsh          ; kblsh           |
EBFA EB0F R   C         dw    k_rsh          ; kbrsh           v
          C
EBFC EA6C R   C         dw    k_res          ; kbres (0C8h)   ^
EBFE EB49 R   C         dw    k_brk          ; kbrk            |
EC00 EA7D R   C         dw    k_pause        ; pause           |

```

```

EC02 EAA1 R      C      dw      k_prt      ; kbprt      |
EC04 EAA9 R      C      dw      k_nul      ; kbnu1      |
EC06 EA60 R      C      dw      k_none     ; NONE       |
C                                     ;           |
EC08 EB26 R      C      dw      k_alt9     ; kdec9      |
EC0A EB27 R      C      dw      k_alt8     ; kdec8      |
EC0C EB28 R      C      dw      k_alt7     ; kdec7      'make' only
EC0E EB29 R      C      dw      k_alt6     ; kdec6      |
EC10 EB2A R      C      dw      k_alt5     ; kdec5      |
EC12 EB2B R      C      dw      k_alt4     ; kdec4      |
EC14 EB2C R      C      dw      k_alt3     ; kdec3      |
EC16 EB2D R      C      dw      k_alt2     ; kdec2      |
EC18 EB2E R      C      dw      k_alt1     ; kdec1      |
EC1A EB2F R      C      dw      k_alt0     ; kdec0      |
C                                     ;           |
EC1C EB3C R      C      dw      k_00      ; kdb10 (0D8h) v
C
EC1E             C      k_data1 endp
C
EC1E             C      code      ends
C             C      include fdu1.asm
C
C
C      :::::::::::::::::::::::::::: CODE ::::::::::::::::::::::::::::
C
EC1E             C      code      segment public 'ROM'
C             C      assume cs:code, ds:data, es:nothing, ss:nothing
C
C      ::::::::::::::::::::::::::::
C      ;
C      ;      defines to support 1.2Mb floppy%
= 0080           C      U48M48D equ      80h
= 0061           C      U48M12D equ      61h
= 0002           C      U12M12D equ      02h
= 0093           C      E48M48D equ      93h
= 0074           C      E48M12D equ      74h
= 0015           C      E12M12D equ      15h
C      ;      f_check_valid uses 0ffh as an illegal rate%%
C      ;      established state only used in format cmd and at end of%
C      ;      successful run%
= 0020           C      DOUBLE equ      20h
= 0010           C      ESTAB equ      10h
= 0000           C      HIRATE equ      00h
= 0040           C      MEDRATE equ     40h
= 0080           C      LORATE equ      80h
C
C      ::::::::::::::::::::::::::::
C
EC59             C      ORG      0EC59h
EC59             C      fd_io   proc   far
C
EC59 FB         C      sti                       ; enable interrupts
EC5A 55         C      push    bp
EC5B 06         C      push    es

```

ROM BIOS Listing

```

EC5C 1E          C      push  ds
EC5D 56          C      push  si
EC5E 57          C      push  di
EC5F 52          C      push  dx          ; head & drive#
EC60 51          C      push  cx          ; cyl. & sec#
EC61 53          C      push  bx          ; buffer offset
EC62 50          C      push  ax          ; command & #secs
EC63 52          C      push  dx          ; this one gets modified(96 TPI)
EC64 8B EC      C      mov   bp,sp       ; BP preserves SP throughout
C
C ; test command code & use jump table to jump to appropriate routine
C
EC66 2E: 8E 1E E538 R C      mov   ds,word ptr cs:[set_ds_word] ; DS = data seg (40h)
EC6B 80 26 003F R 0F C      and   motor_status,0Fh           ; preserve motor on bits.
EC70 80 FC 00      C      cmp   ah,0                   ; Is it a reset command?
EC73 74 24        C      jz   diskette_io1          ; Yes, so ignore drive param.
EC75 80 FA 01      C      cmp   dl,1                 ; max. drives
EC78 77 18        C      ja   f_io1                ; drive out of range.
EC7A 80 FC 05      C      cmp   ah,5                 ; max. command.
EC7D 76 1A        C      jbe  diskette_io1          ; command in range
EC7F 80 FC 14      C      cmp   ah,14h              ; range check%
EC82 76 0E        C      jbe  f_io1                ; error if 6<=cmd<=14%
EC84 80 FC 18      C      cmp   ah,18h              ; range check%
EC87 73 09        C      jae  f_io1                ; error if >=18h%
EC89 80 EC 0F      C      sub   ah,0fh              ; put new cmd in range%
EC8C 88 66 03      C      mov   f_command,ah         ; store new command%
EC8F EB 08 90      C      jmp   diskette_io1          ; cont%
EC92          C f_io1:
EC92 C6 06 0041 R 01 C      mov   diskette_status,cmd_error ; 01h
EC97 EB 30          C      jmp   short f_io_quit        ; quick return%
EC99          C diskette_io1:
EC99 FC          C      cld                       ; Autoincrement for strings.
EC9A E8 EF6B R      C      call  f_check_valid         ; Returns with CY set if err &%
C ; sets current & orignl states%
EC9D          C f_retry:
EC9D E8 F60C R      C      call  f_nustate             ; decide new state, set speed%
ECA0 80 66 00 01 C      and   byte ptr f_drive,1      ; use only dirve LSB for retry%
C ; parm changed in f_seek%
ECA4 32 FF        C      xor   bh,bh                 ; clear high byte
ECA6 8A 5E 03      C      mov   bl,f_command         ; move selection into low byte%
ECA9 D1 E3          C      shl   bx,1                 ; multiply by 2
ECAB 2E: FF A7 EC80 R C      jmp   cs:[f_table.bx]
C
C ;::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
C
ECB0          C f_table label word
C
ECB0 ECFA R      C      dw   f_reset              ; AH = 0 (reset)
ECB2 ECC2 R      C      dw   f_io_ret             ; AH = 1 (status)
ECB4 ED57 R      C      dw   f_rdata              ; AH = 2 (read)
ECB6 F691 R      C      dw   f_wdata              ; AH = 3 (write)
ECB8 ED57 R      C      dw   f_rdata              ; AH = 4 (verify)
ECBA F691 R      C      dw   f_wdata              ; AH = 5 (format)
ECBC F5EB R      C      dw   f_dtype              ; AH'= 6(15H) (read DASD type)%
ECBE EFB3 R      C      dw   f_chngln            ; AH'= 7(16H) (chk change-line status)%

```

```

ECC0 E905 R      C      dw      f_setfrmt      ; AH'= 8(17H) (set format type/speed)%
C
C ;::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
C
ECC2            C      f_io_ret:
ECC2 8B E5      C      mov      sp,bp      ; for safety sake.(several
C                                     ; routines jump here if error)%
ECC4 E8 E665 R  C      call     f_tstretry    ; chk if retry needed%
ECC7 72 D4      C      jc       f_retry      ; do retry%
ECC9            C      f_io_quit:      ;%
C      ; quit processing interrupt%
ECC9 50         C      push     ax      ; hold returns for cmd 15h%
ECCA E8 F5AB R  C      call     f_nec_reset    ; reset if necessary.
C                                     ; f_nec_reset is in fdu4.asm.%
ECCD BB 0002    C      mov      bx,2      ; motor_wait parameter.
ECD0 E8 F62E R  C      call     f_get_var      ; Returns 0 in AH.
ECD3 A2 0040 R  C      mov      motor_count,al ; motor shut off value.
C
ECD6 58         C      pop      ax      ; restore returns%
ECD7 80 7E 03 06 C      cmp      byte ptr f_command,6 ; message return info of cmd15h%
ECDB 75 06      C      jnz     f_retstat    ; else put diskette_status in AH%
ECDD 86 E0      C      xchg    ah,al      ; put info in ah%
ECDF F8         C      cld      ; clr CY, no error%
ECE0 EB 0B 90    C      jmp     f_io_exit      ; done%
ECE3            C      f_retstat:      ;%
ECE3 32 C0      C      xor      al,al      ; zero AL.
ECE5 8A 26 0041 R C      mov      ah,diskette_status
C
ECE9 80 FC 01    C      cmp      ah,1
ECEC F5         C      cmc
C
ECED            C      f_io_exit:      ;%
C                                     ; sp restored above%
ECED 5B         C      pop      bx      ; discard DX.
ECEE 5B         C      pop      bx      ; discard AX.
ECE7 5B         C      pop      bx
ECF0 59         C      pop      cx
ECF1 5A         C      pop      dx
ECF2 5F         C      pop      di
ECF3 5E         C      pop      si
ECF4 1F         C      pop      ds
ECF5 07         C      pop      es
ECF6 5D         C      pop      bp
C
ECF7 CA 0002    C      ret     2
C
ECFA            C      fd_io  endp
C
C ;::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
C ;
C ;      Reset and reprogram the FDC without turning the motors off.
C ;      (Motor turned off for gen 3 reset within f_nurate)%
C ;
C ;::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
C

```

```
ECFA          C   f_reset proc   near
              C
              C
ECFA FA       C       cli                               ; disable interrupts
              C
ECFB B0 66    C       mov     al,pic_seoi_6           ; specific end of interrupt
ECFD BA 0020  C       mov     dx,pic_0               ; to pic 'command' port.
ED00 EE       C       out     dx,al
              C
ED01 42       C       inc     dx                       ; pic 'data' port.
ED02 EC       C       in      al,dx                   ; read mask.
ED03 24 BF    C       and     al,10111111b           ; enable IR6.
ED05 EE       C       out     dx,al
              C
              C ; Develop mask for motor control port.
              C
ED06 A0 003F R C       mov     al,motor_status         ; which motor is running?
ED09 24 0F    C       and     al,0Fh                 ; blow off high 4 bits.
ED0B 74 0E    C       jz      f_r2                   ; no motors running.
ED0D 8A E0    C       mov     ah,al
ED0F B1 04    C       mov     cl,4
ED11 D2 E0    C       shl     al,cl                       ; move to high nib.
              C
              C ; A motor is on. Find out which one.
              C
ED13          C   f_r1:
ED13 D0 EC    C       shr     ah,1                       ; determine drive select
ED15 72 04    C       jc      f_r2                   ; from motor enable bit.
ED17 FE C0    C       inc     al
ED19 EB F8    C       jmp     short f_r1
ED1B          C   f_r2:
              C
              C ; Reset signal has to be maintained for at least 14 clocks.
              C
ED1B C6 06 0041 R 00 C       mov     diskette_status,0
ED20 0C 08    C       or      al,8                       ; set bit 3.
ED22 BA 03F2  C       mov     dx,f_motor_port
ED25 EE       C       out     dx,al                       ; send reset signal.
ED26 0C 04    C       or      al,4                       ; set bit 2.
ED28 C6 06 003E R 00 C       mov     seek_status,0
ED2D EE       C       out     dx,al                       ; clear reset.
ED2E FB       C       sti
ED2F E8 F686 R C       call    f_sis                       ; sense int. status
              C
ED32 B4 03    C       mov     ah,f_specify_cmd
ED34 E8 F6C5 R C       call    f_put_byte
ED37 BB 0000  C       mov     bx,0                       ; 1st specify byte.
ED3A E8 F62E R C       call    f_get_var
ED3D 8A E0    C       mov     ah,al
ED3F E8 F6C5 R C       call    f_put_byte
ED42 BB 0001  C       mov     bx,1                       ; 2nd specify byte.
ED45 E8 F62E R C       call    f_get_var
ED48 8A E0    C       mov     ah,al
ED4A E8 F6C5 R C       call    f_put_byte
              C
```



```

ED4D E9 ECC2 R      C          jmp    f_io_ret
C
C
ED50                C f_reset endp
C
C
C ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
C ;
C ; This routine is called by the timer_int routine when motor_count = 0
C ;
C ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
C
ED50                C stop_disk    proc    near
C
ED50 B0 0C          C          mov    al,0Ch
ED52 BA 03F2       C          mov    dx,f_motor_port
ED55 EE           C          out    dx,al
ED56 C3           C          ret
ED57                C stop_disk    endp
C
C ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
C ;
C ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
C
ED57                C f_rdata proc    near
C
C
C
ED57 80 26 003F R 7F C          and    motor_status,07Fh      ; clear high bit, indicate read
ED5C E8 F5C6 R      C          call   f_motor_on
ED5F 73 08          C          jnc    f_rd1              ; motor was on, no delay.
C
C ; slow motor delay loop
C
C ; slow motors
ED61 B9 01F4       C          mov    cx,500                ; approx. 500 ms delay for
C
ED64                C f_rd_loop:
ED64 E8 EF4C R      C          call   f_wait_one_ms
ED67 E2 FB          C          loop   f_rd_loop
C
ED69                C f_rd1:
ED69 B0 46          C          mov    al,046h              ; DMA mode byte: channel 2,
C ; single mode, write transfer.
ED6B 80 7E 03 04   C          cmp    byte ptr f_command,4      ; Is it a verify command?
ED6F 75 06          C          jne    f_rw_common          ; No, must have been a read.
ED71 B0 42          C          mov    al,042h              ; DMA mode byte: channel 2,
C ; single mode, verify transfer.
ED73 33 DB          C          xor    bx,bx                    ; Fool the DMAC into thinking
ED75 8E C3          C          mov    es,bx                    ; there is a full segment to
C ; play with.
ED77                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
ED77 C f_rw_common proc near
C
C
ED77 C6 06 0040 R FF C mov motor_count,0FFh ; long wait.
ED7C E8 EE49 R C call f_set_dma ; pass mode byte on through.
C
C ; Clear out status from previous operation.
C
ED7F 06 C push es
ED80 1E C push ds
ED81 07 C pop es
ED82 32 C0 C xor al,al
ED84 B9 0007 C mov cx,7
ED87 BF 0042 R C mov di,offset nec_status
ED8A F3/ AA C rep stosb
ED8C 07 C pop es
C
ED8D E8 EEAA R C call f_seek ; On return, f_drive has
C ; head bit or'd in.
C
ED90 8A 56 03 C mov dl,byte ptr f_command ; get command
ED93 B4 C5 C mov ah,f_write_cmd
ED95 80 FA 03 C cmp dl,3 ; Is it a write command?
ED98 74 09 C je f_rw1 ; yes
ED9A B4 4D C mov ah,f_format_cmd
ED9C 80 FA 05 C cmp dl,5 ; Is it a format command?
ED9F 74 02 C je f_rw1 ; yes
EDA1 B4 E6 C mov ah,f_read_cmd ; must be read or verify.
EDA3 C f_rw1:
EDA3 E8 F6C5 R C call f_put_byte ; send command.
EDA6 8A 66 00 C mov ah,f_drive ; has head and drive bits.
EDA9 E8 F6C5 R C call f_put_byte
EDAC 80 7E 03 05 C cmp byte ptr f_command,5 ; was it a format command?
EDB0 74 15 C je f_rw_skip ; yes, skip next 3 params.
EDB2 8A 66 07 C mov ah,f_cyl
EDB5 E8 F6C5 R C call f_put_byte
EDB8 8A 66 01 C mov ah,f_head
EDBB 80 E4 7F C and ah,07Fh ; blow off bit 7.
EDBE E8 F6C5 R C call f_put_byte
EDC1 8A 66 06 C mov ah,f_secnum
EDC4 E8 F6C5 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
EDC7 C f_rw_skip:
EDC7 B9 0004 C mov cx,4

```

```

EDCA BB 0003      C      mov     bx,3                ; no. bytes per sector.
EDCD             C      f_rw2:
EDCD E8 F62E R   C      call    f_get_var
EDD0 8A E0       C      mov     ah,al
EDD2 E8 F6C5 R   C      call    f_put_byte
EDD5 43          C      inc     bx
EDD6 83 FB 05    C      cmp     bx,5                ; time to check for format?
EDD9 75 09       C      jne    f_rw3                ; No.
EDDB 80 7E 03 05 C      cmp     byte ptr f_command,5    ; was it a format command?
EDDF 75 03       C      jne    f_rw3                ; no.
EDE1 BB 0007     C      mov     bx,7                ; 7th parameter in table.
EDE4             C      f_rw3:
EDE4 E2 E7       C      loop   f_rw2
EDE4             C
EDE6 E8 F652 R   C      call    f_wait_for_nec
EDE9 E8 EDF8 R   C      call    f_get_byte           ; get the results.
EDE9             C
EDEC             C      f_rw_ret:
EDEC E9 ECC2 R   C      jmp     f_io_ret
EDEC             C
EDEF             C      f_rw_common   endp
EDEF             C
EDEF             C      ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
EDEF             C      ;
EDEF             C      ;      Read the results bytes from the NEC controller (fdu_data1)
EDEF             C      ;
EDEF             C      ;      INPUT:          none
EDEF             C      ;
EDEF             C      ;      OUTPUT:
EDEF             C      ;
EDEF             C      ;
EDEF             C      ;      DESTROYS:      DX, SI
EDEF             C      ;
EDEF             C      ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
EDEF             C
EDEF             C      fdu_data1   proc
EDEF             C
EDEF             C      f_gb_table   label   byte
EDEF 04          C      db     4                ; record not found.
EDF0 00          C      db     0                ; dummy
EDF1 10          C      db    10h               ; crc error.
EDF2 08          C      db     8                ; dma error.
EDF3 00          C      db     0                ; dummy.
EDF4 04          C      db     4                ; sector not found.
EDF5 03          C      db     3                ; write protect.
EDF6 02          C      db     2                ; address mark error.
EDF7 20          C      db    20h               ; fdc error.
EDF7             C
EDF8             C      fdu_data1   endp
EDF8             C
EDF8             C      f_get_byte   proc   near
EDF8             C
EDF8 06          C      push    es
EDF9 1E          C      push    ds
EDFA 07          C      pop     es

```

ROM BIOS Listing

```

EDFB BF 0042 R      C      mov     di,offset nec_status      ; ES:DI is now ready for stosb.
EDFE 8B F7         C      mov     si,di                          ; offset of nec_status in SI.
EE00 B9 0007       C      mov     cx,7                          ; maximum no. of bytes.
EE03              C      f_gb_loop:
EE03 E8 F63B R     C      call    f_nec_rdy                      ; Returns MSR byte in AL.
                                           C      ; will not return if error.
EE06 A8 10         C      test    al,10h                        ; busy bit.
EE08 74 3D         C      jz     f_gb_ret                        ; done.
EE0A A8 40         C      test    al,40h                        ; direction bit.
EE0C 74 0C         C      jz     f_gb_out                        ; wrong direction.
EE0E 42            C      inc     dx                            ; point to nec data port (3F5h)
EE0F EC           C      in     al,dx
EE10 AA           C      stosb
EE11 51           C      push   cx                            ; save cx %
EE12 B9 0002       C      mov     cx,2                          ; need to insure atleast%
                                           C      ; 12 microseconds between %
EE15              C      wasteg:                             ; this out and next in done%
EE15 E2 FE         C      loop   wasteg                       ; by f_nec_rdy, so waste some time%
EE17 59           C      pop    cx                            ; restore cx %
EE18 E2 E9         C      loop   f_gb_loop
EE1A              C      f_gb_out:
EE1A 80 7E 03 00   C      cmp     byte ptr f_command,0         ; was it reset command?
EE1E 74 27         C      je     f_gb_ret                        ; yes.
EE20 AC           C      lodsb                                ; get ST0 in AL.
EE21 A8 20         C      test    al,20h                        ; seek end?
EE23 75 22         C      jnz    f_gb_ret                        ; yes.
EE25 A8 C0         C      test    al,0C0h
EE27 74 1A         C      jz     f_gb_jump
                                           C
EE29 C6 06 0041 R 20 C      mov     diskette_status,fdc_error
EE2E AC           C      lodsb                                ; get ST1 in AL.
EE2F B9 0008       C      mov     cx,8
EE32 33 DB         C      xor     bx,bx
EE34              C      f_gb_loop1:
EE34 D0 C0         C      rol     al,1
EE36 72 03         C      jc     f_gb_decode
EE38 43           C      inc     bx
EE39 E2 F9         C      loop   f_gb_loop1
EE3B              C      f_gb_decode:
EE3B 2E: 8A 87 EDEF R C      mov     al,cs:f_gb_table[bx]
EE40 A2 0041 R     C      mov     diskette_status,al
EE43              C      f_gb_jump:
EE43 07           C      pop    es
EE44 E9 ECC2 R     C      jmp     f_io_ret
EE47              C      f_gb_ret:
EE47 07           C      pop    es
EE48 C3           C      ret
                                           C
EE49              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
EE49 C f_set_dma proc near
C
EE49 C6 06 0041 R 09 C mov diskette_status,dma_seg_error
EE4E E6 0B C out dma_mode,al ; send mode byte.
EE50 8B 5E 04 C mov bx,word ptr f_bufoff ; offset of user buffer.
EE53 8C C0 C mov ax,es ; offset of user buffer.
EE55 B1 04 C mov cl,4 ; shift count
EE57 D3 C0 C rol ax,cl ; move high nib around.
EE59 8A E8 C mov ch,al ; save high nib.
EE5B 80 E5 0F C and ch,0Fh ; isolate high nib
EE5E 24 F0 C and al,0F0h ; prepare for offset calculation
EE60 03 D8 C add bx,ax ; 20 bit calculation.
EE62 73 02 C jnc f_sd1
EE64 FE C5 C inc ch ; high nib
EE66 C f_sd1:
EE66 8A C5 C mov al,ch
EE68 E6 81 C out dma_segm_2,al ; high nib. to latch
EE6A B0 06 C mov al,6
EE6C E6 0A C out dma_mask_bit,al ; disable channel 2.
EE6E FA C cli ;rsc/drb disable interrupt %
EE6F E6 0C C out dma_ff_clr,al ; set to known state. %
EE71 8A C3 C mov al,bl
EE73 E6 04 C out dma_addr_2,al ; low byte of address.
EE75 8A C7 C mov al,bh
EE77 E6 04 C out dma_addr_2,al ; high byte of address.
EE79 FB C sti ; enable interrupts rsc/drb %
EE7A 8B D3 C mov dx,bx ; save buffer offset.
EE7C BB 0003 C mov bx,3 ; bytes/sector param
EE7F E8 F62E R C call f_get_var ; get param from table
EE82 8A C8 C mov cl,al ; save for shift count
EE84 8A 66 02 C mov ah,f_numsecs ; get #sectors.
EE87 32 C0 C xor al,al ; AX = #sectors x 256
EE89 D1 E8 C shr ax,1 ; AX = #sectors x 128
EE8B D3 E0 C shl ax,cl ; multiply by bytes/sector.
EE8D 48 C dec ax ; DMAC counts zero.
EE8E 03 D0 C add dx,ax ; add count to offset
EE90 73 03 C jnc f_sd2
EE92 E9 ECC2 R C jmp f_io_ret ; exit boundary crossing.
EE95 C f_sd2:
EE95 FA C cli ;rsc/drb disable interrupt %
EE96 E6 0C C out dma_ff_clr,al ; set to known state. %
EE98 90 C nop ; no back to back i/o %
EE99 E6 05 C out dma_count_2,al ; low byte of count.
EE9B 8A C4 C mov al,ah
EE9D E6 05 C out dma_count_2,al ; high byte of count.
EE9F FB C sti ;enable interrupt rsc/drb%

```

ROM BIOS Listing

```

EEA0 B0 02      C      mov     al,2
EEA2 E6 0A      C      out     dma_mask_bit,al      ; enable channel 2
EEA4 C6 06 0041 R 00  C      mov     diskette_status,0
EEA9           C      f_sd_ret:
EEA9 C3          C      ret
EEAA           C      f_set_dma      endp
C
C      ;::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
C      ;
C      ;      Seek  (f_seek)
C      ;
C      ;      INPUT:
C      ;
C      ;      OUTPUT:
C      ;
C      ;      DESTROYS:
C      ;
C      ;::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
C
EEAA           C      f_seek  proc   near
C
EEAA 8A 4E 00    C      mov     cl,f_drive      ; get drive# as shift count.
EEAD B0. 01      C      mov     al,1
EEAF D2 E0      C      shl     al,cl      ; mask for recal.
EEB1 84 06 003E R  C      test    seek_status,al
EEB5 75 35      C      jnz     f_s1      ; no recal. required.
C
EEB7 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
EEBB B9 0002    C      mov     cx,2      ; loop count for 96 TPI
EEBE           C      f_s_recal:
EEBE B4 07      C      mov     ah,f_recal_cmd      ; recal. command
EEC0 51          C      push    cx      ; save it.
EEC1 E8 F6C5 R  C      call    f_put_byte
EEC4 8A 66 00    C      mov     ah,f_drive      ; drive .bit.
EEC7 E8 F6C5 R  C      call    f_put_byte      ; end of command phase.
EECA E8 F686 R  C      call    f_sis      ; Sense Interrupt Status
EECD 59          C      pop     cx      ; restore it.
C
C      ; possibly test bit 4 as well (equipment check...track 0 not reached)
EECE E8 F603 R  C      call    f_getdrv      ; current drive
C
EED1 F6 06 0042 R C0  C      test    nec_status,0C0h
EED6 74 0F      C      jz      f_s2      ; equipment OK so restore.
EED8 E2 E4      C      loop   f_s_recal
EEDA C6 87 0094 R FF  C      mov     cur_cyl[bx],0ffh      ; error so set cur_cyl to a
C      ; wacky value
EEDF C6 06 0041 R 40  C      mov     diskette_status,seek_error
EEE4 E9 ECC2 R    C      jmp     f_io_ret
C
EEE7           C      f_s2:
EEE7 C6 87 0094 R 00  C      mov     cur_cyl[bx],0      ; recal'd to cylinder 0
EEEC           C      f_s1:

```

```

EEEC E8 F603 R      C      call    f_getdrv
EEEF 8A 46 01      C      mov     al,f_head      ; this code moved %%%/%
EEF2 24 01        C      and     al,1           ; blow off high 7 bits.
EEF4 D0 E0        C      shl    al,1           ; move head to bit 2.
EEF6 D0 E0        C      shl    al,1
EEF8 0A 46 00      C      or     al,f_drive     ; combine drive bit with head
EEFB 88 46 00      C      mov     f_drive,al    ; and save it.
EEFE 8A 87 0094 R  C      mov     al,cur_cyl[bx] ; get cur_cyl value in al%
EF02 8A 66 07      C      mov     ah,f_cyl      ; desired cylinder #%
EF05 F6 46 01 80  C      test   byte ptr f_head,80h ; test 80 track bit.
EF09 74 02        C      jz     no_dbl_step    ; see if double step is
EF0B D0 E4        C      shl    ah,1           ; necessary
EF0D              C no_dbl_step:
EF0D 88 A7 0094 R  C      mov     cur_cyl[bx],ah
EF11 3A C4        C      cmp     al,ah          ; how do they compare?%
EF13 74 36        C      je     f_s_ret        ; if they were equal don't seek%
C
EF15 B4 0F        C      mov     ah,f_seek_cmd
EF17 E8 F6C5 R    C      call   f_put_byte
C ; prepare second byte of seek command.
EF1A 8A 66 00      C      mov     ah,f_drive
EF1D E8 F6C5 R    C      call   f_put_byte
EF20 8A 66 07      C      mov     ah,f_cyl
EF23 F6 46 01 80  C      test   byte ptr f_head,80h ; test 80 track bit.
EF27 74 02        C      jz     f_cont
EF29 D0 E4        C      shl    ah,1           ; cyl x 2
EF2B              C f_cont:
EF2B E8 F6C5 R    C      call   f_put_byte     ; end of seek command.
EF2E E8 F686 R    C      call   f_sis          ; Sense Interrupt Status
C
EF31 F6 06 0042 R C0 C      test   nec_status,0C0h ; test for seek end.
C
EF36 75 13        C      jnz    f_s_ret
EF38 BB 0009        C      mov     bx,9           ; head settle parameter.
EF3B E8 F62E R    C      call   f_get_var
EF3E 0A C0        C      or     al,al
EF40 74 09        C      jz     f_s_ret        ; head settle = 0.
EF42 8A C8        C      mov     cl,al          ; use settle parm. al loop index
EF44 32 ED        C      xor    ch,ch
EF46              C f_head_settle:
EF46 E8 EF4C R    C      call   f_wait_one_ms
EF49 E2 FB        C      loop  f_head_settle
EF4B              C f_s_ret:
EF4B C3            C      ret
C
EF4C              C f_seek  endp
C
C ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
C ;
C ;      One millisecond delay loop (approximately)
C ;
C ;      The CALL is 7 clocks.
C ;      The callers LOOP statement is 8 clocks.
C ;      No flags affected.
C ;

```

```
C ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
C
EF4C C f_wait_one_ms proc near
EF4C 51 C push cx ; 3 clocks
C ; mov cx,374 ; 2 clocks
EF4D B9 02E4 C mov cx,740 ; 2 clocks
EF50 E2 FE C w_one: loop w_one ; (8 x CX) clocks
EF52 59 C pop cx ; 5 clocks
EF53 C3 C ret ; 11+ clocks
EF54 C f_wait_one_ms endp
C
C include fdu2.asm
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 interrupts
EF58 50 C push ax
EF59 1E C push ds
EF5A B0 66 C mov al,pic_seoi_6 ; specific end of interrupt
EF5C E6 20 C out pic_0,al ; send to 8259
EF5E 2E: 8E 1E E538 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 ; interrupt indicator
EF68 1F C pop ds
EF69 58 C pop ax
EF6A CF C iret
EF6B C fd_int endp
C
C
C ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
C ;
C ; Check Valid (f_check_valid)
C ;
C ; This routine sets the starting diskstate. If the last operation%
C ; passed (was ESTABLISHED), the state is held. Otherwise, the %
C ; default state indicated by the motherboard switches is used%
C ;
C ; Drive 1:%
C ; Bit 0 of sys_conf_b port: 0 = 40 trk drive, 1 = 80 trk drive
C ;
C ; Drive 0:%
C ; Bit 1 of sys_conf_b port: 0 = 40 trk drive, 1 = 80 trk drive%
C ;
```



```
C
EFAD          C f_cv_ret:
EFAD E8 F6BA R C      call   f_setff          ; set rate flip-flops al=state%
EFB0 5B       C      pop    bx          ; restore regs%
EFB1 58       C      pop    ax          ; restore regs%
C
EFB2 C3       C      ret
C
EFB3          C f_check_valid  endp
C
C ;::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
C ;
C ;      Change-line status
C ;
C ;      Routine records a media change.
C ;
C ;::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
C
EFB3          C f_chngln   proc   near
EFB3 C6 06 0041 R 06 C      mov    diskette_status,media_change ;status %
EFB8 E9 ECC2 R   C      jmp    f_io_ret          ; cmd over%
EFBB          C f_chngln   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
C ;fdu_data3   proc
C ;commented out because DOS[ibmbios] handles parms internally%
C ;
C ;fd_12parms  label  byte
C ;
C ;      db    (f_srt_96 shl 4) + f_hut          ; SRT + HUT%
C ;      db    (f_hlt shl 1) + f_ndma          ; HLT + DMA mode
C ;      db    f_motor_wait                    ; motor off delay
C ;      db    2                               ; 512 bytes per sector
C ;      db    0fh                             ; EOT last sector on cyl%
C ;      db    01bh                            ; gap length(23h-48m12d?)%
C ;      db    0FFh                            ; DTL
C ;      db    054h                            ; gap length for format%
C ;      db    0F6h                            ; fill byte for format
C ;      db    20                              ; head settle time (ms)%
C ;      db    4                               ; motor start time
C ;
C ;fdu_data3   endp
C
EFC7          C      ORG    0EFC7h
C
EFC7          C fdu_data2   proc
```

```

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
EFCF 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 fd_data2      endp
C
EFD2          C code      ends
C          .LIST                                          ;number 4 start

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 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 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
EFD8 86 C4 C xchg al,ah ; reverse al & ah
C ; ah saves al throughout
EFD8 1E C push ds ; save ds
EFD8 2E: 8E 1E E538 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 ; data port
EFF3 1F C pop ds ; restore ds
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                                     ; 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      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      short 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 interrupts, 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 ;/*      NAME      DATE          ACTION
```

```

C ;* ---- ---- -----
C ;* mikef 10/25/84 Added code in scroll up to enable video.
C ;* mikef 02/13/85 Moved grf_light_pen from graph.src to here.
C ;* joe 03/13/85 Changed ORG to ORG
C ;* joe 03/25/85 Replaced scroll code in text mode for speed
C ;* joe 04/04/85 Fixed Mode 7 scroll to use ds
C ;* joe 04/18/85 Added code to check for dip switch indication
C ;* of non-Olivetti video controller board
C ;* joe 06/11/85 Revived code to wait for Hercules card during
C ;* slow scrolling code.
C ;* joe 06/13/85 Changed fourth parameter in 6845 parameter table
C ;* from 06h to 0Ah, for v_md_40 and v_md_graph.
C ;* Removed extraneous parameter table stuff.
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):
C ;; V_KSCROLL1 equ 224 ; 314 chars to move during vert. retrace
= 00E0 C V_KSCROLL1 equ 224 ; 324 chars to move during vert. retrace
= 0162 C V_KSCROLL2 equ 354 ; 354 chars to move during vert. retrace
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 F5A8 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 F390 R C dw v_scrl_dn ; ah = 07h
F055 F3B1 R C dw v_rac ; ah = 08h
F057 F3DB R C dw v_wac ; ah = 09h
F059 F414 R C dw v_wc ; ah = 0Ah
F05B F44F R C dw v_col ; ah = 0Bh
F05D D56A R C dw grf_write_dot ; ah = 0Ch (see graph.src)
F05F D550 R C dw grf_read_dot ; ah = 0Dh (see graph.src)
F061 F47B R C dw v_terminal ; ah = 0Eh
F063 F504 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 ;:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
C ;
C ;      Osmerge code relies on the fact that v_io is org'ed at 0f065h and
C ;      that the first instruction is "sti".  It is also important that
C ;      this is the ONLY sti that is incurred in the path to/in the cursor
C ;      postioning code.  Osmerge contains an elegant hack that speeds up
C ;      video access if the stated condition is met.  For further information
C ;      see memo "OSMERGE Constraints on ROM BIOS" .
C ;
C ;:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
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 E538 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
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)
FOA0 5F             C      pop    di
FOA1 1F             C      pop    ds
FOA2 07             C      pop    es
FOA3                C      v_nop:
FOA3 CF            C      iret
C
FOA4                C      v_io   endp
C
C                  ;=====
C
FOA4                C      ORG    0F0A4h
C
FOA4                C      v_data2 proc  near
C
FOA4                C      v_parms label byte
C
C                  ; 6845 Parameters except register 3h (Horizontal Synch Width).
C
FOA4 38 28 2D 0A    C      v_md_40      db    38h,28h,2Dh,0Ah ; text 40 x 25
FOA8 1F 06 19 1C    C                  db    1Fh,06h,19h,1Ch ; mode 0 -> monochrome
FOAC 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
FOB4 71 50 5A 0C    C      v_md_80      db    71h,50h,5Ah,0Ch ; text 80 x 25
FOB8 1F 06 19 1C    C                  db    1Fh,06h,19h,1Ch ; mode 2 -> monochrome
FOBC 02 07 06 07    C                  db    02h,07h,06h,07h ; mode 3 -> color
FOC0 00 00 00 00    C                  db    00h,00h,00h,00h
C
FOC4 38 28 2D 0A    C      v_md_graph   db    38h,28h,2Dh,0Ah ; graphics
FOC8 7F 06 64 70    C                  db    7Fh,06h,64h,70h ; mode 4 -> 320 x 200 color
FOCC 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

```

ROM BIOS Listing

```

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
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      C          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
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
F0FC                 C  v_data2 endp
C
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
FOFE 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
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:[int1D10cn] ; 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 ; 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                    ; restore cx
F1E7 5A        C      pop     dx                    ; restore dx
F1E8 C3        C      ret
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 ;-----
C ;      Set Cursor Position          ah = 02h
C ;
C ;      Input:  bh      = page number (0-7)
C ;              (dh,d1) = (row,col) of current cursor from (0,0)
C ;      Output: if bh = v_apage,     ah = 14
C ;                  al = low byte of cursor position
C ;              else,               ah = v_mode
C ;                  al = preserved
C ;
C ;      Trash:  si & di destroyed. (si = dx, di = cx)
C ;-----
C
C
F1F7 C v_curs_pos  proc   near
C
C      assume cs:code, ds:data, es:v_ram, ss:nothing
C
F1F7 8B F9 C      mov     di,cx          ; save cx
C
F1F9 8A CF C      mov     cl,bh
F1FB 8B F1 C      mov     si,cx
F1FD 81 E6 0007 C     and     si,7          ; mask to 8 pages
F201 03 F6 C      add     si,si        ; *2 => word index
C
F203 89 94 0050 R C     mov     word ptr ds:[si+v_curpos],dx ; save the cursor position
C
F207 3A 3E 0062 R C     cmp     bh,byte ptr ds:[v_apage] ; if active page, put the cursor
F20B 74 03 C     je      v_set_curs          ; on the screen.
C                                     ; not active page, so just
F20D 8B CF C     mov     cx,di          ; restore cx
F20F C3 C     ret                    ; and exit
C
F210 C v_set_curs: ; active page, so set cursor..
F210 8B C2 C     mov     ax,dx          ; ax gets cursor position
F212 EB 42 90 C     jmp     v_set_cur_pos        ; set cursor; ah 6845 gets cx
C                                     ; si = dx destroyed
C                                     ; ah = preserved = 14
C                                     ; al = low byte of cursor posn
C
F215 C v_curs_pos  endp
C
C ;-----
C ;      Read Cursor                  ah = 03h
C ;
C ;      Input:  bh      = page number (0-7)
C ;      Output: (dh,d1) = (row,col) of current cursor from (0,0)
C ;              (ch,c1) = current cursor mode setting
C ;              ax      = dx
C ;

```

```

C ;      Trash: None.
C ;-----
C
F215 C v_r_curs_pos  proc  near
C
C          assume cs:code, ds:data, es:v_ram, ss:nothing
C
F215 8B C2 C      mov  ax,dx
F217 8B CB C      mov  cx,bx          ; save bx
F219 8A DF C      mov  bl,bh
F21B 81 E3 0007 C      and  bx,07h          ; page number mod 8
F21F D1 E3 C      shl  bx,1          ; page number mod 8 word index
F221 8B 97 0050 R C      mov  dx,word ptr ds:[bx+v_curpos]
F225 8B D9 C      mov  bx,cx          ; restore bx
F227 8B 0E 0060 R C      mov  cx,word ptr ds:[v_cursize]
F22B C3 C      ret
C
F22C C v_r_curs_pos  endp
C
C ;-----
C ;      Read Light Pen (see graph.src) ah = 04h
C ;-----
C
C ;-----
C ;      Set Active Display Page          ah = 05h
C ;
C ;      Input:  al      = new page number (0-7 for modes 0-1; 0-3 for 2-3)
C ;      Output: 6845 is reset to display the new active page
C ;      ah      = 14
C ;      al      = low byte of cursor position
C ;
C ;      Trash:  bp, si & di destroyed. (si = dx, di = cx)
C ;-----
C
F22C C v_page  proc  near
C
C          assume cs:code, ds:data, es:v_ram, ss:nothing
C
F22C 25 0007 C      and  ax,07h          ; page number mod 8
F22F 8B E8 C      mov  bp,ax          ; save ax = page number mod 8
C
F231 A2 0062 R C      mov  byte ptr ds:[v_apage],al  ; save active page number (0-7)
F234 74 08 C      jz   v_page_0          ; page number = 0?
C
F236 8B FA C      mov  di,dx          ; save dx
F238 F7 26 004C R C      mul  word ptr ds:[v_height]  ; dx:ax = (page number)*v_height
F23C 8B D7 C      mov  dx,di          ; restore dx
C
F23E C v_page_0:
F23E A3 004E R C      mov  word ptr ds:[v_top],ax  ; save starting address of page
F241 D1 F8 C      sar  ax,1          ; divide by 2 for byte count
C
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 F560 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 ; 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
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_scr1_up   proc   near
C
C          assume cs:code, ds:data, es:v_ram, ss:nothing
C
F27F E8 F571 R  C      call    v_txt_md                      ; all registers preserved
F282 72 03      C      jb     v_txt_up
F284 E9 D5F4 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 F513 R  C      call    v_scr1_pos                    ; to common scroll positioning routine
F291 74 25      C      jz     v_clr                          ; 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 <- number of rows to scroll
C
F297 80 3E 0049 R 07 C      cmp    byte ptr ds:[v_mode],7
F29C 74 0A      C      je     v_mv_fast           ; jump to fast loop
C
F29E 53         C      push   bx                  ; save parameters of "clear"
C
C ; If a non-Olivetti Video Controller Board, we'll do a slow scroll
C
F29F BB F322 R   C      mov    bx,offset v_1        ; set up to move one word per scan line
C ;;; in    al,sys_conf_b         ; read port 67h (switches at 7w)
C ;;; test  al,NON_OLI_VID        ; non Olivetti video controller ?
C ;;; jnz   v_mv2                 ; jump if not Olivetti Video Controller
C
C ; The dip-switches say it's Olivett. are we runing on a Turbo (M24SP) ?
C
C ;;; call  v_8253                 ; latch and read counter 0
C ;;; mov   bh,b1                  ; bh <- lsb of latched count
C ;;; mov   cl,2                   ; execute enough instructions to
C ;;; v_mv1: loop v_mv1            ; distinguish 8 MHz from 10 MHz
C ;;; call  v_8253                 ; latch and read counter 0 again
C ;;; sub   bh,b1                  ; compute the difference
C ;;; cmp   bh,23h                 ; max val is 20h on 10 MHz 8086
C ;;; mov   bx,offset v_2          ; set up to move two words per scan line
C ;;; jb    v_mv2                 ; jump if 10 MHz cpu clock
C ;;; mov   bx,offset v_1          ; set up to move one word per scan line
F2A2           C v_mv2:
C
C ; Perform the scroll
C
F2A2 E8 F2EC R   C      call   v_scroll_or_clear
F2A5 5B         C      pop    bx                  ; recover the "clear" parameters
F2A6 EB 10      C      jmp    short v_clr           ; go clear rows at top/bottom
C
C
C ;-----
C
F2A8           C v_mv_fast:
F2A8 1E         C      push   ds                  ; save data segment
F2A9 06         C      push   es                  ; use video RAM address in es for move
F2AA 1F         C      pop    ds                  ; set up ds to address video RAM
F2AB           C v_mv_flp:
F2AB 8A CA      C      mov    cl,d1                ; number of columns to move
F2AD F3/ A5     C      rep    movsw               ; move the row (es:di gets ds:si)
F2AF 03 F5      C      add    si,bp                ; add number of bytes to skip in row to
C                                          ; 'from' address for scroll
F2B1 03 FD      C      add    di,bp                ; add number of bytes to skip in row to
C                                          ; 'to' address for scroll
F2B3 FE CC      C      dec    ah                  ; one less row to be moved

```

```
C
F2B5 75 F4      C      jnz     v_mv_flp          ; go back if not finished
C
F2B7 1F        C      pop     ds              ;restore data segment
C ;-----
C
C ;      Set up to clear one or more rows
C
F2B8          C      v_clr:
F2B8 8A E7      C      mov     ah,bh          ; ah <- attribute of line to clear
F2BA B0 20      C      mov     al,' '        ; al <- " space"
F2BC 80 3E 0049 R 07 C      cmp     byte ptr ds:[v_mode],7
F2C1 74 0C      C      je      v_clr_fast     ; fast clear if monochrome board
F2C3 8B F0      C      mov     si,ax          ; si <- attr:space
F2C5 8A E3      C      mov     ah,bl          ; ah <- number of rows to clear
F2C7 BB F30D R   C      mov     bx,offset v_0     ; set up for clear
C
C ;      Perform the clear
C
F2CA E8 F2EC R   C      call    v_scroll_or_clear
C
F2CD EB 0A        C      jmp     short v_clr_fin
C
F2CF          C      v_clr_fast:
F2CF 8A CA      C      mov     cl,dl          ;number of columns to clear
F2D1 F3/ AB      C      rep stosw             ;clear the row
F2D3 03 FD      C      add     di,bp            ;add number of bytes to skip in row
F2D5 FE CB      C      dec     bl              ;decrement row counter
F2D7 75 F6      C      jnz     v_clr_fast     ;repeat if not finished
C
F2D9          C      v_clr_fin:
F2D9 80 3E 0049 R 07 C      cmp     byte ptr ds:[v_mode],7
F2DE 74 07      C      je      v_scr1_mode_7
F2E0 A0 0065 R   C      mov     al,byte ptr ds:[v_3x8] ;
F2E3 BA 03D8      C      mov     dx,03D8h         ;; video enable register.
F2E6 EE          C      out     dx,al          ;; enable video.
F2E7          C      v_scr1_mode_7:
C
C ;      Restore the registers and return
C
F2E7 FC        C      cld                    ; restore the direction flag to UP
F2E8 5B        C      pop     bx
F2E9 59        C      pop     cx
F2EA 5A        C      pop     dx
F2EB C3        C      ret
C
C ;      Scroll or clear the requested number of rows
C
F2EC          C      v_scroll_or_clear:
C
F2EC 8A CA      C      mov     cl,dl          ; cx <- number of cols to move per row
F2EE 52        C      push    dx              ; save dl for subsequent "clear" call
F2EF 8B 16 0063 R C      mov     dx,[v_base6845]
F2F3 83 C2 06      C      add     dx,6              ; dx <- addr(video status register)
F2F6 1E        C      push    ds              ; save ds
```

```

F2F7 06      C      push  es
F2F8 1F      C      pop   ds          ; ds <- v_ram segment
C           C           assume ds:v_ram
C           C           ;
C           C           Inhibit the timer interrupt
C           C           ;
F2F9 FA      C      cli
F2FA E4 21   C      in   al,pic_1      ; 8259 mask register
F2FC 0C 01   C      or   al,01h      ; mask IRQ0
F2FE E6 21   C      out  pic_1,al
F300 FB      C      sti
C           C           ;
C           C           Wait for the end of horizontal retrace
C           C           ;
F301 E8 F385 R C      call  v_sync
C           C           ;
C           C           Loop for each row to scroll or clear
C           C           ;
F304         C      v_rows:
F304 51      C      push  cx          ; save the column counter on the stack
C           C           ;
C           C           Waste some time to be sure retrace is ended
C           C           ;
F305         C      v_cols:
F305 51      C      push  cx          ;This section of code is
F306 B1 02   C      mov   cl,2          ; necessary for the
F308 E2 FE   C      v_c2: loop  v_c2      ; Hercules Graphics Card
F30A 59      C      pop   cx
C           C           ;
C           C           If clearing rows
C           C           ;
F30B FF E3   C      jmp   bx          ; select case (v_0, v_1 or v_2)
C           C           ;
C           C           Loop for all columns of the current row
C           C           Wait for the beginning of horizontal retrace
C           C           ;
F30D         C      v_0:
F30D EC      C      in   al,dx
F30E D0 D8   C      rcr  al,1
F310 73 FB   C      jnc  v_0
C           C           ;
C           C           Clear one word of the current row
C           C           ;
F312 96      C      xchg  ax,si      ; ax <- attrib:space
F313 AB      C      stosw
C           C           ;
C           C           If there are more columns to clear in the current row
C           C           ;
F314 49      C      dec  cx
F315 74 03   C      jz   v_01
C           C           ;
C           C           Clear a second word
C           C           ;
F317 AB      C      stosw
F318 EB 01   C      jmp  short v_02

```

ROM BIOS Listing

```

F31A          C v_01:
F31A 41      C      inc    cx                ; adjust the column counter
F31B          C v_02:
F31B 96      C      xchg   ax,si            ; restore ah = row counter
C
C ;          Repeat for all columns of the current row
C
F31C E2 E7   C      loop   v_cols
F31E EB 4A   C      jmp    short v_31
C
C ;          Else scrolling rows
C ;          Loop for all columns in the current row
C ;          If only one word can be moved at each horizontal retrace
C ;          Wait for the beginning of horizontal retrace
C
F320 00E0    C      dw     V_KSCROLL1          ; # of words to move at vert. retrace
F322          C v_1:
F322 EC      C      in     al,dx
F323 D0 D8   C      rcr   al,1
F325 73 FB   C      jnc   v_1
C
C ;          Move one word of the current row
C
F327 A5      C      movsw
C
C ;          If vertical retrace has started
C ;          If scrolling the full screen width
C ;          Move a fixed number of words or finish the scroll
C
F328 EB 0E   C      jmp    short v_v
C
C ;          Else two words can be moved at each horizontal retrace
C ;          If there is only one column remaining in this row
C
F32A 0162    C      dw     V_KSCROLL2          ; # of words to move at vert. retrace
F32C          C v_2:
F32C E2 03   C      loop   v_21
C
C ;          Move the last word of the current row
C
F32E 41      C      inc    cx                ; adjust the column counter
F32F EB F1   C      jmp    v_1
C
C ;          Else
C ;          Wait for the beginning of horizontal retrace
C
F331          C v_21:
F331 EC      C      in     al,dx
F332 D0 D8   C      rcr   al,1
F334 73 FB   C      jnc   v_21
C
C ;          Move two words of the current row
C
F336 A5      C      movsw
F337 A5      C      movsw

```

```

C
C ;           If vertical retrace has started
C
F338          C v_v:
F338 A8 04    C      test  a1,04h           ; has vertical retrace started ?
F33A 74 2A    C      jz     v_22           ; jump if not
C
C ;           If scrolling the full screen width
C
F33C 0B ED    C      or     bp,bp
F33E 75 26    C      jnz   v_22
C
C ;           Move a fixed number of words or finish the scroll
C
F340 8B EA    C      mov   bp,dx           ; save the video status reg. addr
F342 5A      C      pop   dx           ; dx <- number of columns per row
F343 52      C      push  dx           ; save it for the next row pass
F344 49      C      dec   cx           ; cx = cols remaining to move, this row
F345 8A C4    C      mov   al,ah          ; al <- remaining row count
F347 FE C8    C      dec   al           ; don't want to consider the curr. row
F349 F6 E2    C      mul   dl           ; ax <- cols remaining on subseq. rows
F34B 03 C1    C      add   ax,cx          ; ax <- total columns remaining to move
F34D 2E: 8B 4F FE C      mov   cx,cs:[bx-2]       ; set up to move V_KSCROLLn words
F351 2B C1    C      sub   ax,cx          ; ax <- cols left after move V_KSCROLLn
C ;           ; fewer than V_KSCROLLn remaining ?
F353 72 1E    C      jb    v_v2           ; jump if so, exit the row loop
F355 F3/ A5    C      rep  movsw          ; move V_KSCROLLn words
F357 F6 F2    C      div   dl           ; al <- full rows remaining to be moved
C ;           ; ah <- columns left to move, curr. row
F359 8A CC    C      mov   c1,ah          ; restore column counter in cx
F35B 41      C      inc   cx           ; adjust column counter
F35C 40      C      inc   ax           ; adjust row counter
F35D 8A E0    C      mov   ah,al          ; restore row counter in ah
F35F 8B D5    C      mov   dx,bp          ; restore video status reg addr
F361 33 ED    C      xor   bp,bp          ; restore bp register
F363 E8 F385 R C      call  v_sync          ; wait for the end of hor. retrace
C
C ;           Repeat for all columns in the current row
C
F366          C v_22:
F366 E2 9D    C      loop  v_cols
C
C ;           Repeat while more rows remain to be scrolled
C
F368          C v_3:
F368 03 F5    C      add   si,bp          ; advance source pointer to next row
F36A          C v_31:
F36A 03 FD    C      add   di,bp          ; advance dest. pointer to next row
F36C 59      C      pop   cx           ; reinitialize the column counter
F36D FE CC    C      dec   ah           ; decrement the row counter
F36F 74 09    C      jz     v_4           ; jump if all rows finished
F371 EB 91    C      jmp   v_rows          ; continue with the next row
C
C ;           Move any remaining words during the current vertical retrace interval
C

```

```
F373          C v_v2:
F373 03 C8    C      add    cx,ax
F375 F3/ A5   C      rep movsw      ; move the remaining words
F377 59      C      pop     cx        ; reinitialize the column counter
F378 33 ED   C      xor     bp,bp      ; restore bp register
C
C ;          Reenable the timer interrupt and return
C
F37A          C v_4:
F37A FA      C      cli
F37B E4 21   C      in     al,pic_1      ; 8259 mask register
F37D 24 FE   C      and    al,not 01h    ; unmask IRQ0
F37F E6 21   C      out    pic_1,al
F381 FB      C      sti
F382 1F      C      pop     ds          ; restore ds
C
C      assume ds:data
F383 5A      C      pop     dx          ; restore dl for subsequent "clear" call
F384 C3      C      ret
C
C ;          Latch and read counter 0 of the 8253
C
C ;;:v_8253:
C ;;:      mov     al,0      ; "latch counter" command
C ;;:      out    43h,al     ; latch 8253 counter 0
C ;;:      in     al,40h     ; lsb of latched count
C ;;:      mov     bl,al     ; save it for comparison
C ;;:      in     al,40h     ; msb of latched count
C ;;:      ret
C
C ;          Wait for the end of horizontal retrace
C
F385          C v_sync:
F385 EC      C      in     al,dx
F386 D0 D8   C      rcr    al,1
F388 73 FB   C      jnc    v_sync
F38A          C v_sync2:
F38A EC      C      in     al,dx
F38B D0 D8   C      rcr    al,1
F38D 72 FB   C      jc     v_sync2
F38F C3      C      ret
F390          C v_scr1_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
F390 C v_scr1_dn proc near
C
C assume cs:code, ds:data, es:v_ram, ss:nothing
C
F390 FD C std ; NOTE: scroll down everything backwards
C
F391 E8 F571 R C call v_txt_md ; all registers preserved
F394 72 03 C jb v_txt_dn
F396 E9 D689 R C jmp grf_graphics_down ; jump if graphics
F399 C v_txt_dn:
C
F399 52 C push dx ; save registers
F39A 51 C push cx
F39B 53 C push bx
C
F39C 8A D8 C mov bl,al ; save line count
F39E 8B C2 C mov ax,dx ; pass lower right coordinates...
F3A0 E8 F513 R C call v_scr1_pos ; to common scroll positioning routine
F3A3 74 07 C jz v_clr_top ; clear rows if nothing to move
C
C ; Scroll cl rows down.
C
F3A5 C v_mv_dn:
F3A5 2B F0 C sub si,ax ; subtract (bytes/row)*(rows to scroll)
C ; from 'from' address for scroll
F3A7 F7 DD C neg bp ; negate number of bytes to skip per row
F3A9 E9 F295 R C jmp v_mv ; now identical to scroll up!
C
C ; Clear bl rows above.
C
F3AC C v_clr_top:
F3AC F7 DD C neg bp ; negate number of bytes to skip per row
F3AE E9 F2B8 R C jmp v_clr ; now identical to v_clr_bot!
C
F3B1 C v_scr1_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
F3B1 C v_rac proc near
C

```

```
C      assume cs:code, ds:data, es:v_ram, ss:nothing
C
F3B1 E8 F571 R C      call v_txt_md      ; all registers preserved
F3B4 72 03     C      jb v_txt_rac
F3B6 E9 D6EC R C      jmp grf_graphics_read ; jump if graphics
F3B9                                     C v_txt_rac:
C
F3B9 8B FB     C      mov di,bx          ; save bx
F3BB E8 F542 R C      call v_fpos          ; ax & si destroyed.
C                                          ; bx = offset into current page
C
F3BE 8B F2     C      mov si,dx          ; save dx
F3C0 BA 0006   C      mov dx,6           ; get 6845 status reg. offset
F3C3 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
C
F3C7                                     C v_rac_inline:      ; make sure we're in a scanline
F3C7 EC       C      in al,dx        ; get horiz. retrace blanking status
F3C8 .D0 D8   C      rcr al,1        ; test for horiz. retrace
F3CA 72 FB    C      jc v_rac_inline ; wait for display enable low (cf)
F3CC FA       C      cli            ; disable ints FIRST
C
C                                          ; wait for blanking:
C
F3CD                                     C v_rac_inblank:
F3CD EC       C      in al,dx        ; get retrace blanking status
F3CE D0 D8   C      rcr al,1        ; test display enable (bit #0)
F3D0 73 FB    C      jnc v_rac_inblank ; try again if still in scanline.
C
F3D2 26: 8B 07 C      mov ax,es:[bx]       ; char. and attr. now in AX
F3D5 FB       C      sti            ; enable interrupts immediately
C
F3D6 8B D6    C      mov dx,si        ; restore dx
F3D8 8B DF    C      mov bx,di        ; restore bx
F3DA C3       C      ret
C
F3DB                                     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
F3DB          C v_wac  proc   near
C
C          assume cs:code, ds:data, es:v_ram, ss:nothing
C
F3DB E8 F571 R  C      call   v_txt_md           ; all registers preserved
F3DE 72 03      C      jb    v_txt_wac
F3E0 E9 D7DB R  C      jmp   grf_graphics_write    ; jump if graphics
F3E3          C v_txt_wac:
C
F3E3 8B FB      C      mov   di,bx                ; save bx
C
F3E5 E8 F542 R  C      call   v_fpos                ; ax & si destroyed.
C          ; bx = offset into current page
F3E8 87 DF      C      xchg  bx,di                ; restore bx; di = transfer offset
C
F3EA 8B C5      C      mov   ax,bp                ; restore ax
F3EC 8A E3      C      mov   ah,bl                ; transfer attribute byte to ah
F3EE 8B E8      C      mov   bp,ax                ; save attribute & character in bp
C
F3F0 8B F1      C      mov   si,cx                ; save cx
F3F2 52          C      push  dx                    ; save dx
F3F3 8B 16 0063 R C      mov   dx,word ptr ds:[v_base6845] ; get 6845 pointer register
F3F7 83 C2 06    C      add   dx,6                    ; get 6845 status register
C
C ; Wait for horizontal retrace blank interval ...
C
C
C
F3FA          C v_wac_hi:                ; wait till we're in a scanline..
F3FA EC          C      in   al,dx                ; get CRT status
F3FB D0 D8      C      rcr  al,1                  ; test display enable (bit #0)
F3FD 72 FB      C      jc   v_wac_lo                ; wait for display enable low (cf)
F3FF FA          C      cli                          ; disable ints FIRST
C
F400          C v_wac_lo:                ; now wait till start of blanking..
F400 EC          C      in   al,dx                ; 08 get CRT status
F401 D0 D8      C      rcr  al,1                  ; 02 test display enable (bit #0)
F403 73 FB      C      jnc  v_wac_lo                ; 16/04 wait for display enable hi (cf)
C
F405 8B C5      C      mov   ax,bp                ; 02 restore ax
F407 AB          C      stosw                       ; 11 es:di gets ax (attribute & char)
C
F408 49          C      dec   cx                    ; 02
C          ; we can do 2 words in 10 us
F409 74 04      C      jz   v_wac_end                ; 04/16
F40B 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
F40C FB          C      sti                          ; enable interrupts immediately
F40D E2 EB      C      loop  v_wac_hi                ; do it cx times
C
F40F          C v_wac_end:
F40F FB          C      sti                          ; enable interrupts immediately
C

```

ROM BIOS Listing

```

F410 5A          C      pop    dx                ; restore dx
F411 8B CE       C      mov    cx,si              ; restore cx
F413 C3          C      ret

F414            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      ;-----
F414            C      v_wc   proc   near
C
C      assume cs:code, ds:data, es:v_ram, ss:nothing
C
F414 E8 F571 R    C      call   v_txt_md            ; all registers preserved
F417 72 03        C      jb    v_txt_wc
F419 E9 D7DB R    C      jmp    grf_graphics_write    ; jump if graphics
F41C            C      v_txt_wc:
C
F41C 8B FB        C      mov    di,bx                ; save bx
C
F41E E8 F542 R    C      call   v_fpos                ; si destroyed.
C                                  ; ax = offset into page 0
C                                  ; bx = offset into current page
F421 87 DF        C      xchg   bx,di                ; restore bx; di = transfer offset
C
F423 8B F1        C      mov    si,cx                ; save cx
F425 52          C      push   dx                ; save dx
C
F426 8B D5        C      mov    dx,bp                ; retrieve character in dl
F428 8A C2        C      mov    al,dl                ; get char in al (ah = attr.)
F42A 8B E8        C      mov    bp,ax                ; bp = (attr. char)
C
F42C BA 0006      C      mov    dx,6                ; get 6845 status register
F42F 03 16 0063 R C      add    dx,word ptr ds:[v_base6845] ; get 6845 pointer register
C
C      ; Wait for horizontal retrace...
C
F433            C      v_wc_next:
C
F433            C      v_wc_hi:
F433 EC          C      in    al,dx                ; get CRT status
F434 D0 D8        C      rcr   al,1                ; test display enable (bit #0)

```

```

F436 72 FB      C      jc      v_wc_hi      ; wait for display enable low (cf)
F438 FA        C      cli      ; disable ints FIRST
C
F439          C      v_wc_lo:
F439 EC        C      in      al,dx      ; 08  get CRT status
F43A D0 D8     C      rcr      al,1      ; 02  test display enable (bit #0)
F43C 73 FB     C      jnc      v_wc_lo      ; 16/04 wait for display enable hi (cf)
C
F43E 8B C5     C      mov      ax,bp      ; 02  restore ax = (attr, char)
F440 AA        C      stosb     ; 11  es:di gets al (character)
C
F441 49        C      dec      cx      ; 02
C      ;we can do 2 bytes in 10 us
F442 74 06     C      jz      v_wc_end      ; 04/16
F444 47        C      inc      di      ; 02  skip past attribute byte
C
F445 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
F446 FB        C      sti      ; enable interrupts immediately
F447 47        C      inc      di      ; skip past attribute byte
F448 E2 E9     C      loop     v_wc_next      ; do it cx times
C
F44A          C      v_wc_end:
F44A FB        C      sti      ; enable interrupts immediately
C
F44B 5A        C      pop      dx      ; restore dx
F44C 8B CE     C      mov      cx,si      ; restore cx
F44E C3        C      ret
C
F44F          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
F44F          C      v_col   proc   near
C
C      assume  cs:code, ds:data, es:v_ram, ss:nothing
C
F44F A0 0066 R  C      mov      al,byte ptr ds:[v_colorpal] ; get current palette
C
F452 8B FA     C      mov      di,dx      ; save dx
F454 8B F3     C      mov      si,bx      ; save bx
C

```

ROM BIOS Listing

```

F456 0A FF      C      or      bh,bh          ; palette color ID = 0?
F458 74 0A      C      jz      v_col_0        ; handle color ID 0
C
F45A 24 DF      C      and     a1,0DFh        ; clear palette select bit #5
F45C D0 DB      C      rcr     b1,1          ; test new color (bit #0)
F45E 73 0B      C      jnb     v_col_1        ; if bit #0 set, all done
F460 0C 20      C      or      a1,20h        ; else set palette select bit #5
F462 EB 07      C      jmp     short v_col_1
C
F464            C      v_col_0:
F464 80 E3 1F    C      and     b1,01Fh        ; save bits #0-4 of new color
F467 24 E0      C      and     a1,0E0h        ; clear bits #0-4 of old color
F469 0A C3      C      or      a1,b1          ; and combine the two.
C
F46B            C      v_col_1:
F46B BA 0005     C      mov     dx,5            ; get 6845 overscan reg. offset
F46E 03 16 0063 R C      add     dx,word ptr ds:[v_base6845] ; add 6845 pointer register
F472 EE          C      out     dx,a1          ; output selection
C
F473 8B DE      C      mov     bx,si          ; restore bx
F475 8B D7      C      mov     dx,di          ; restore dx
C
F477 A2 0066 R   C      mov     byte ptr ds:[v_colorpal],a1 ; save the value for later
F47A C3          C      ret
C
F47B            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:  a1      = character to write
C ;              b1      = 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
F47B            C      v_terminal proc near
C
C      assume cs:code, ds:data, es:v_ram, ss:nothing
C
F47B 3C 07      C      cmp     a1,BEL          ; is it bell character?
F47D 75 03      C      jne     v_term_nobell
C
F47F E9 F583 R   C      jmp     v_bell
C

```

```

F482          C v_term_nobell:
F482 52       C          push  dx          ; save dx
F483 8B F1    C          mov   si,cx        ; save cx
F485 8B FB    C          mov   di,bx        ; save bx
C
C          ; Get cursor position in active page.
C
F487 B7 07    C          mov   bh,07h        ;mask for page number, MOD 8
F489 22 3E 0062 R C          and   bh,byte ptr ds:[v_apage] ; get active page number (0-7)
C
F48D B4 03    C          mov   ah,03h        ; call v_r_curs_pos
F48F CD 10    C          INT   10h          ; (dh,dI) = (row,col) of cursor
C
C          ; (ch,cI) = cursor mode setting
C
F491 8B C5    C          mov   ax,bp          ; restore ax
F493 B9 0001  C          mov   cx,1          ; character count for write char
F496 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
F498 3C 0A    C          cmp   al,LF          ; is it a line feed?
F49A 74 14    C          je    v_lf            ;
F49C 3C 0D    C          cmp   al,CR          ; is it a carriage return?
F49E 74 60    C          je    v_cr            ;
F4A0 3C 08    C          cmp   al,BS          ; is it a backspace?
F4A2 74 54    C          je    v_bs            ;
C
C          ; Normal Case: write the character
C
F4A4 CD 10    C          INT   10h          ; to write the character
C
F4A6 FE C2    C          inc   dl            ; increment the column
F4A8 3A 16 004A R C          cmp   dl,byte ptr ds:[v_width] ; column overflow?
F4AC 72 13    C          jb   v_set_new_cur     ; set new cursor position
C
F4AE 32 D2    C          xor   dl,dl           ; carriage return cursor
C
F4B0 80 3E 0049 R 48 C v_lf: cmp   byte ptr ds:[v_mode],72 ; is this mode 72 ?
F4B5 B4 31    C          mov   ah,49          ; mode 72 has 50 rows
F4B7 74 02    C          je    v_lrow         ; jump if mode 72
F4B9 B4 18    C          mov   ah,24          ; modes 4,5,6,64 have 25 rows
F4BB 3A F4    C v_lrow: cmp   dh,ah        ; are we at last row yet?
F4BD 74 07    C          je    v_scri_tty     ; if yes, go scroll the screen
F4BF FE C6    C          inc   dh            ; otherwise, inc to next row
F4C1          C v_set_new_cur:
F4C1 B4 02    C          mov   ah,02h        ; call v_curs_pos to set new
F4C3 EB 29 90  C          jmp   v_term_ret     ; cursor position
C
F4C6          C v_scri_tty:          ; (dh,dI) = (row,col) = (24,0) or (49,0)
F4C6 B4 02    C          mov   ah,02h        ; call v_curs_pos to set cursor
F4C8 CD 10    C          INT   10h          ; and so that we can read back
C
C          ; the proper attribute byte
C
F4CA 32 E4    C          xor   ah,ah          ; ah = 0 for graphics
F4CC E8 F571 R C          call  v_txt_md       ; are we text mode?

```

```

F4CF 73 04      C      jnb     v_scr1_tty_graphics      ; jump if graphics
C
F4D1 B4 08      C      mov     ah,08h                    ; call v_rac
F4D3 CD 10      C      INT     10h                      ; to get attribute byte in ah
C
F4D5           C      v_scr1_tty_graphics:
F4D5 33 C9      C      xor     cx,cx                    ; (ch,cl)= upper left (row,col)
C                                           ;      = (0,0)
F4D7 8A FC      C      mov     bh,ah                    ; store attribute in bh
F4D9 B8 0601     C      mov     ax,0601h                 ; call v_scr1_up to scroll
C                                           ; one line with attribute bh
F4DC 8A 16 004A R C      mov     dl,byte ptr ds:[v_width]     ; (dh,dl)= lower right (row,col)
F4E0 80 EA 01     C      sub     dl,1                      ; column = v_width-1
F4E3 80 3E 0049 R 48 C      cmp     byte ptr ds:[v_mode],72     ; is this mode 72 ?
F4E8 B6 31      C      mov     dh,49                    ; if yes then row = 49
F4EA 74 02      C      je      v_term_ret                ; jump if mode = 72
F4EC B6 18      C      mov     dh,24                    ; if not mode 72 then row = 24
C
F4EE           C      v_term_ret:
F4EE CD 10      C      INT     10h
F4F0           C      v_term_nop:
C
C      ; Clean up.
C
F4F0 8B C5      C      mov     ax,bp                    ; restore ax
F4F2 8B DF      C      mov     bx,di                    ; restore bx
F4F4 8B CE      C      mov     cx,si                    ; restore cx
F4F6 5A      C      pop     dx                    ; restore dx
F4F7 C3      C      ret
C
C
F4F8 0A D2      C      v_bs:  or     dl,dl                ; back space -- column = 0 ?
F4FA 74 F4      C      jz     v_term_nop                ; don't change cursor position
F4FC FE CA      C      dec     dl
F4FE EB C1      C      jmp    v_set_new_cur
C
C
F500 32 D2      C      v_cr:  xor     dl,dl                ; carriage return
F502 EB BD      C      jmp    v_set_new_cur
C
F504           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
F504           C      v_stat proc  near
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
F513      C v_scr1_pos      proc      near
C
C          assume cs:code, ds:data, es:v_ram, ss:nothing
C
F513      E8 F560 R      C      call      v_posn          ; (ah,al) -> ax offset; si trash
C
F516      2A F5          C      sub      dh, ch          ; (dh,dl) gets delta (drow,dcol)
F518      FE C6          C      inc      dh          ; dh = number of rows
F51A      2A D1          C      sub      dl, cl
F51C      FE C2          C      inc      dl          ; dl = number of columns
C
F51E      8B 36 004E R   C      mov     si, word ptr ds:[v_top] ; get offset of active page
F522      03 F0          C      add     si, ax          ; add offset in page => 'from'
F524      8B FE          C      mov     di, si          ; , 'to' addresses.
C
C
F526      33 C9          C      xor     cx, cx          ; init count register to zero.
F528      8A CA          C      mov     cl, dl          ; cx = number of columns = dl
F52A      A1 004A R     C      mov     ax, word ptr ds:[v_width] ; get screen width
F52D      8B E8          C      mov     bp, ax          ; bp = v_width
F52F      2B E9          C      sub     bp, cx          ; bp = (v_width - dl)
F531      D1 E5          C      shl     bp, 1          ; bp = 2 * (v_width - dl)
C
F533      D0 E0          C      shl     al, 1          ; al = 2 * v_width
F535      F6 E3          C      mul     bl          ; ax = 2*v_width*no. of rows
C
F537      8A CE          C      mov     cl, dh
F539      2A CB          C      sub     cl, bl          ; cl <= number of rows to move
C
F53B      0A DB          C      or     bl, bl          ; if rows to scroll,
F53D      75 02          C      jnz     v_scr1_mv_and_clr ; then, move & clear row, return nz.
C
F53F      8A DE          C      mov     bl, dh          ; else clear dh rows only, return z.
C
F541      C v_scr1_mv_and_clr:
F541      C3          C      ret          ; ZF indicates state of BL at entry
C
F542      C v_scr1_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
F542      C v_fpos proc near
C
C          assume cs:code, ds:data, es:v_ram, ss:nothing
C
F542 8A C7      C      mov     al,bh                ; al gets page number
F544 33 DB      C      xor     bx,bx                ; bx = 0
F546 25 0007    C      and     ax,07h            ; ax = page number mod 8
F549 8B F0      C      mov     si,ax                ; si keeps page number mod 8
F54B 74 07      C      jz     v_fpos_0            ; page number = 0?
C
F54D      C v_fpos_lp:
F54D 03 1E 004C R C      add     bx,word ptr ds:[v_height] ; optimization: word multipli-
F551 48          C      dec     ax                ; cation by less than 8 without
F552 75 F9      C      jnz    v_fpos_lp            ; destroying dx (or cx).
C
F554      C v_fpos_0:
F554 D1 E6      C      shl     si,1                ; bx = (page number)*(v_height)
F556 8B 84 0050 R C      mov     ax,word ptr ds:[si+v_curpos] ; page number mod 8 word index
F55A E8 F560 R   C      call    v_posn                ; (ah,al) -> ax offset; si trash
F55D 03 D8      C      add     bx,ax                ; bx = (page)*(v_height)+offset
F55F C3          C      ret
C
F560      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
F560      C v_posn      proc near
C
C          assume cs:code, ds:data, es:v_ram, ss:nothing
C
F560 8B F0      C      mov     si,ax
F562 81 E6 00FF C      and     si,0FFh            ; si keeps column (al)
F566 8A C4      C      mov     al,ah                ; al gets row (ah)
F568 F6 26 004A R C      mul     byte ptr ds:[v_width] ; ax gets (row * v_width)
F56C 03 C6      C      add     ax,si                ; ax gets (row * v_width)+ column
F56E D1 E0      C      shl     ax,1                ; ax gets 2*((row * v_width)+column)
F570 C3          C      ret
C

```

```

F571          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
F571          C  v_txt_md      proc   near
C
C          assume  cs:code, ds:data, es:v_ram, ss:nothing
C
F571 80 3E 0049 R 04  C      cmp      byte ptr ds:[v_mode],4
F576 72 09          C      jb      v_txt_ok
C
F578 80 3E 0049 R 07  C      cmp      byte ptr ds:[v_mode],7
F57D 74 02          C      je      v_txt_ok
F57F F8            C      clc
F580 C3            C      ret
C
F581          C  v_txt_ok:
F581 F9            C      stc
F582 C3            C      ret
C
F583          C  v_txt_md      endp
C
C ;-----
C ;          Handle BEL character:  Beeps the speaker.
C ;
C ;          No parameters.
C ;
C ;-----
C
F583          C  v_bell proc near
C
C          assume  cs:code, ds:data, es:v_ram, ss:nothing
C
F583 50          C      push   ax
C
F584 B0 B6          C      mov     al,t2cmd
F586 E6 43          C      out    p_8253_ctrl1,al
F588 B0 00          C      mov     al,00h
F58A E6 42          C      out    p_8253_2,al
F58C B0 06          C      mov     al,06h
F58E E6 42          C      out    p_8253_2,al
C
F590 E4 61          C      in     al,p_kctrl
F592 8A E0          C      mov     ah,al
F594 0C 03          C      or     al,03h

```

```

F596 E6 61      C      out    p_kctrl,al
C
F598 51        C      push   cx
F599 B9 00C8    C      mov     cx,200                      ;512 msec
F59C           C      bell_wait:
F59C E8 EF4C R   C      call   f_wait_one_ms             ;wait for 1 ms
F59F E2 FB     C      loop   bell_wait
F5A1 59        C      pop     cx
C
F5A2 8A C4     C      mov     al,ah                      ; restore control status
F5A4 E6 61     C      out    p_kctrl,al
C
F5A6 58        C      pop     ax
F5A7 C3        C      ret                                ; return from v_term
C
F5A8           C      v_bell endp
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,d1) = (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
F5A8           C      grf_light_pen  proc  near
C
F5A8 32 E4     C      xor     ah,ah                      ; return ah = 0 for now (al intact)...
F5AA C3        C      ret
C
F5AB           C      grf_light_pen  endp
C
F5AB           C      code  ends
C      include fdu4.asm
C      ;::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
C
F5AB           C      code  segment public 'ROM'
C      assume cs:code, ds:data, es:nothing, ss:nothing
C
F5AB           C      f_nec_reset  proc  near
C
F5AB BA 03F4   C      mov     dx,f_nec_status           ; NEC status port
F5AE EC        C      in     al,dx
F5AF 3C 10     C      cmp     al,10h                     ; NEC busy.
F5B1 75 12     C      jne     f_nec_reset_ret           ; no.
F5B3 A0 0041 R C      mov     al,diskette_status         ; save from previous operation.
F5B6 50        C      push   ax

```

ROM BIOS Listing

```

F5B7 33 C0          C      xor    ax,ax                ; reset call.
F5B9 8B D0          C      mov    dx,ax
F5BB 9C             C      pushf
F5BC 9A             C      db     9ah
F5BD EC59 R         C      dw    fd_io                ; this is call far 0f000:fd_io
F5BF F000           C      dw    code_seg
C
C
C ;      int    13h
F5C1 58             C      pop    ax
F5C2 A2 0041 R      C      mov    diskette_status,al    ; restore from previous op.
F5C5             C      f_nec_reset_ret:
F5C5 C3             C      ret
C
F5C6             C      f_nec_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
F5C6             C      f_motor_on    proc    near
C
F5C6 C6 06 0040 R FF C      mov    motor_count,0FFh      ; max. time for motor.
F5CB 8A 4E 00       C      mov    c1,f_drive            ; drive.
F5CE B0 01           C      mov    al,1
F5D0 D2 E0         C      shl    al,c1                ; mask for motor status.
F5D2 84 06 003F R  C      test   al,motor_status        ; test always clears carry.
F5D6 75 12         C      jnz    f_mo_ret            ; already running.
F5D8 A2 003F R      C      mov    motor_status,al     ; set correct bit.
F5DB 8A E1         C      mov    ah,c1                ; drive into AH.
F5DD B1 04         C      mov    c1,4
F5DF D2 E0         C      shl    al,c1                ; motor on bit to high nibble.
F5E1 0C 0C         C      or    al,0Ch              ; set bits 2 & 3 (0000 1100).
F5E3 0A C4         C      or    al,ah                ; drive bits ( 0 & 1).
F5E5 BA 03F2       C      mov    dx,f_motor_port
F5E8 EE           C      out   dx,al                ; turn on the motor.
F5E9 F9           C      stc                       ; motor was off indicator.
C
F5EA             C      f_mo_ret:
F5EA C3             C      ret
F5EB             C      f_motor_on    endp
C
C
C ;::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
C ;
C ;      Read DASD type
C ;

```

```

C ;      This routine reads the motherboard switches to determine
C ;      what type of drive is installed. (value returned from this
C ;      routine in AL, exchange to AH made after f_io_ret).
C ;
C ;      Returns 0 - not used
C ;          1 - low density drive (48tpi)
C ;          2 - high density drive (1.2Mb)
C ;
C ;::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
C
F5EB      C f_dtype proc    near
F5EB E8 F603 R      C      call    f_getdrv          ; get drive bit in bl%
F5EE E8 F5F6 R      C      call    f_drvswitch       ; get switch info%
F5F1 FE C0          C      inc     al              ; get return info%
F5F3 E9 ECC2 R      C      jmp     f_io_ret          ; cmd over%
F5F6      C f_dtype endp
C
C
C ;::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
C ;
C ;      Routine checks the hardware switches for the type of drive
C ;      being used. The fdu-parameter table (INT1ELOCN) is changed
C ;      as is appropriate.
C ; (commented out because DOS[ibmbios] adjusts the table internally)
C ;::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
C ;
C ;f_setparms      proc    near
C ;      call    f_drvswitch       ; get drive type in al%
C ;      mov     bl,f_head         ; get info b4 change segment%
C ;      push   ds
C ;      push   ax
C ;      xor    ax,ax
C ;      mov    ds,ax
C ;
C ;assume ds:abs0
C ;
C ;      pop    ax
C ;      test   bl,80h            ; test media/drive mismatch?%
C ;      jnz   f_setslow         ; yes,use 320kb(48tpi) info%
C ;      or    al,al              ; else use drive(same as media) info%
C ;      jz    f_setslow         ; jmp if 48tpi drive%
C ;;same parms for now%
C ;      mov   word ptr INT1ELOCN,offset fd_12parms ; 12Mb drive%
C ;      jmp   f_setdone
C ;f_setslow:
C ;      mov   word ptr INT1ELOCN,offset fd_parms   ; 48 tpi drive%
C ;f_setdone:
C ;assume ds:data
C ;      pop   ds
C ;      ret
C ;f_setparms   endp
C ;::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
C ;
C ;      get the configuration switch for the fdu and
C ;      return it in the LSB of al.

```

```

C ;
C ;          0 = 48 tpi
C ;          1 = 96 tpi / 1.2Mb
C ;
C ;::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
C
F5F6      C f_drvswitch   proc   near
F5F6 E4 67      C         in     al,sys_conf_b           ; switch port
F5F8 F6 46 00 01  C         test   byte ptr f_drive,1           ; select drive bit%
F5FC 75 02      C         jnz    f_cv_dr           ; jmp if drive 1%
F5FE D0 C8      C         ror    al,1           ; shift Dr0 into LSB%
F600      C f_cv_dr:
F600 24 01      C         and    al,1           ; blow off high 7 bits
F602 C3         C         ret
F603      C f_drvswitch   endp
C
C ;::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
C ;
C ;          f_getdrv
C ;
C ;          Routine gets the input drive parm and returns with
C ;          the LSB (drive number) in bl.
C ;
C ;::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
C
F603      C f_getdrv     proc   near
F603 32 FF      C         xor    bh,bh
F605 8A 5E 00   C         mov    bl,f_drive
F608 80 E3 01   C         and    bl,1           ; get drive bit
F60B C3         C         ret
F60C      C f_getdrv     endp
C
C
C ;::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
C ;
C ;          f_nustate
C ;
C ;          Routine sets the original state if this is the first attempt
C ;          at an operation. The new transfer rate is set. A delay is taken
C ;          if the motor is already on. ( A test for track number <40 could
C ;          be made here [f_head setting also needed if so].
C ;
C ;::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
C
F60C      C f_nustate   proc   near
F60C 8A 46 03   C         mov    al,f_command           ; hold present cmd%
F60F 3C 01      C         cmp    al,1           ; test for status cmd%
F611 74 05      C         jz     f_notstat           ; skip for status%
F613 C6 06 0041 R 00  C         mov    diskette_status,0       ; prepare for new attempt%
F618      C f_notstat:
F618 E8 F603 R    C         call   f_getdrv           ; get drive number in bl%
F61B 80 BF 0092 R 00  C         cmp    diskstate[bx+2],0       ; test for 1st attempt%
F620 75 08      C         jnz    f_nu_cont           ; jmp if its a retry%
F622 8A 87 0090 R    C         mov    al,diskstate[bx]
F626 88 87 0092 R    C         mov    diskstate[bx+2],al       ; store original state%

```



```

F62A          C f_nu_cont:
F62A E8 E93A R C      call    f_setrate          ; set transfer rate%
C
F62D C3       C      ret
F62E          C f_nustate    endp
C
C
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
F62E          C f_get_var    proc    near
C
C ;      mov     al,f_head          ;get media info%
C ;      and     al,80h
C ;      jz      f_gvok            ; media-drive match%
C ;      cmp     bl,5              ; test gap parm%
C ;      jnz     f_gvok
C ;      mov     al,23h            ; return it%
C ;      jmp     f_gvdone
F62E          C f_gvok:
F62E 1E       C      push   ds
F62F 33 C0    C      xor     ax,ax
F631 8E D8    C      mov     ds,ax              ; segment 0
C
C      assume ds:abs0            ; tell assembler seg 0:
C
F633 C5 36 0078 R C      lds     si,dword ptr [int1Elocn] ; DS:SI points to table
F637 8A 00    C      mov     al,[bx+si]
F639 1F       C      pop     ds
C
C      assume ds:data            ; tell assembler seg 40:
F63A          C f_gvdone:          ;%
F63A C3       C      ret
C
F63B          C f_get_var    endp
C
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
F63B C f_nec_rdy proc near
C
F63B 51 C push cx
F63C 33 C9 C xor cx,cx
F63E C f_nr1:
F63E BA 03F4 C mov dx,f_nec_status ; Main status register(3F4h)
F641 EC C in al,dx
F642 A8 80 C test al,080h ; RQM
F644 75 0A C jnz f_nr_ret ; NEC is ready.
F646 E2 F6 C loop f_nr1
F648 C6 06 0041 R 80 C mov diskette_status,time_out
F64D E9 ECC2 R C jmp f_io_ret ; Took too long to respond.
F650 C f_nr_ret:
F650 59 C pop cx
F651 C3 C ret
C
F652 C f_nec_rdy endp
C
C ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
C ;
C ; Wait for NEC to interrupt on completion of execution phase or
C ; time out if NEC never interrupts (f_wait_for_nec).
C ;
C ; INPUTS: none
C ;
C ; OUTPUTS:
C ;
C ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
C
F652 C f_wait_for_nec proc near
C
F652 FB C sti ; enable interrupts
F653 51 C push cx
F654 B9 03E8 C mov cx,1000 ; wait a while
F657 C w_nec:
F657 F6 06 003E R 80 C test seek_status,80h ; test MSB (int_flag)
F65C 75 0D C jnz w_nec_ret
F65E E8 EF4C R C call f_wait_one_ms
F661 E2 F4 C loop w_nec
F663 C6 06 0041 R 80 C mov diskette_status,time_out
F668 E9 ECC2 R C jmp f_io_ret
F66B C w_nec_ret:
F66B 80 26 003E R 7F C and seek_status,07Fh ; clear MSB.
F670 59 C pop cx
F671 C3 C ret
C
F672 C f_wait_for_nec endp
C
C ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
C ;
C ; chkspeed
```

```

C ;
C ;   this routine checks the error status associated with the
C ;   previous operation. If the bad-addr-mark error was found
C ;   a speed change will be made. Otherwise the error is taken
C ;   as valid and not retry is made.
C ;
C ;   OUTPUT: CY=0, no error-> no retry, speed ok.
C ;           CY=1, speed error -> retry at other speed
C ;
C ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
C
F672          C  chkspeed      proc  near
F672  2E: 8E 1E E538 R  C      mov    ds,word ptr cs:[set_ds_word]    ;get segment%
F677  BE 0042 R      C      mov    si,offset nec_status                ; get buffer%
F67A  46              C      inc    si                          ; point to ST1 of error info%
F67B  AC              C      lodsb                       ; get error info from DS:SI%
F67C  A8 01          C      test   al,01h                          ; check for addr-mark error%
F67E  74 04          C      jz    f_speedok                       ; zeros for no error%
F680  F9              C      stc                               ; flag error%
F681  EB 02 90       C      jmp    f_cs_out                       ; done%
F684          C  f_speedok:
F684  F8              C      cllc                               ; no error%
F685          C  f_cs_out:
F685  C3              C      ret
F686          C  chkspeed      endp
C
C ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
C ;
C ;   Sense Interrupt Status (f_sis).
C ;
C ;   INPUT:      none
C ;
C ;   OUTPUT:
C ;
C ;   DESTROYS:
C ;
C ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
C
F686          C  f_sis  proc   near
C
F686  E8 F652 R      C      call   f_wait_for_nec                    ; no return on error.
F689  B4 08          C      mov    ah,f_snsint_cmd                ; end of command phase.
F68B  E8 F6C5 R      C      call   f_put_byte                      ; no return on error.
F68E  E9 EDF8 R      C      jmp    f_get_byte
C
F691          C  f_sis  endp
C
F691          C  f_wdata proc  near
C
C
F691  E8 F5C6 R      C      call   f_motor_on                      ; sets carry if motor was off.
F694  73 1A          C      jnc    f_wd1                          ; motor was on, skip delay.
C
F696  B0 02          C      mov    al,2                          ; assume slow motor bit%
F698  8A C8          C      mov    cl,al

```

ROM BIOS Listing

```

F69A BA 007D      C      mov     dx,125                ; 125 ms delay to start with.
F69D D3 E2       C      shl     dx,c1                ; 125 x 4
F69F BB 000A     C      mov     bx,10                ; motor start delay parameter.
F6A2 E8 F62E R   C      call    f_get_var            ; returns param. in AL.
F6A5 32 E4       C      xor     ah,ah                ; for good measure.
F6A7 F7 E2       C      mul     dx                    ; AX has total delay
F6A9 8B C8       C      mov     cx,ax
F6AB             C      f_wd_loop:
F6AB E8 EF4C R   C      call    f_wait_one_ms
F6AE E2 FB       C      loop   f_wd_loop
C
F6B0             C      f_wd1:
F6B0 80 0E 003F R 80 C      or      motor_status,080h      ; set high bit, indicate write.
F6B5 B0 4A       C      mov     al,04Ah              ; DMA mode byte: channel 2,
C                                          ; single mode, read transfer
F6B7 E9 ED77 R   C      jmp     f_rw_common
F6BA             C      f_wdata endp
C
C      ;::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
C      ;
C      ;      f_setff
C      ;
C      ;      routine writes to the fdu rate flip-flops
C      ;      (called in setrate and f_check_valid.)
C      ;
C      ;      INPUT: AL - parm to determine rate: eg. diskstate[],lastrate[]
C      ;
C      ;::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
C
F6BA             C      f_setff proc    near
F6BA D0 C0       C      rol     al,1                ; move onto low bits%
F6BC D0 C0       C      rol     al,1                ; move%
F6BE 24 03       C      and     al,3                ; hold relevant bits%
C      ENDIF    ; non-beta units use 2 bits for data transfer rate%
F6C0             C      f_dorate:
F6C0 BA 0065     C      mov     dx,65h              ; rate port%
F6C3 EE         C      out     dx,al              ; set rate%
F6C4 C3         C      ret
F6C5             C      f_setff 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
F6C5             C      f_put_byte    proc    near
C
F6C5 E8 F63B R   C      call    f_nec_rdy            ; returns MSR byte in AL.
F6C8 A8 40       C      test    al,40h              ; direction bit.
C      ; We could put a little more intelligence here to determine why the NEC balked.

```

```

F6CA 75 0C          C      jnz     f_pb_erret      ; wrong direction.
F6CC 42            C      inc     dx              ; NEC data port 3F5h
F6CD 8A C4        C      mov     al,ah
F6CF EE          C      out     dx,al
F6D0 51          C      push    cx              ; save cx %
F6D1 B9 0002      C      mov     cx,2            ; need to insure atleast%
                                C                        ; 12 microseconds between %
F6D4            C      waste:                  ; this out and next in done%
F6D4 E2 FE        C      loop    waste           ; by f_nec_rdy, so waste some time%
F6D6 59          C      pop     cx              ; restore cx %
                                C
F6D7            C      f_pb_ret:
F6D7 C3          C      ret
                                C
F6D8            C      f_pb_erret:
F6D8 C6 06 0041 R 20 C      mov     diskette_status,fdc_error
F6DD E9 ECC2 R   C      jmp     f_io_ret
                                C
F6E0            C      f_put_byte   endp
                                C
F6E0            C      code     ends
                                C      include int18.asm
                                C
                                C      ;=====
                                C      ;      Filename:      int18.src
                                C      ; .
                                C      ;=====
                                C
F6E0            C      code     segment public 'ROM'
                                C      assume  cs:code, ds:nothing, es:nothing, ss:nothing
                                C
F6E0            C      basic_trap   proc   near
                                C
F6E0 BE F6E8 R    C      mov     si,offset trap_mess
F6E3 E8 E540 R   C      call    DRomString      ; CS:SI points to string.
F6E6 EB FE          C      jmp     $              ; loop forever.
                                C
F6E8 52 6F 6D 20 42 41 C      trap_mess   db      'Rom BASIC not available, ',CR,LF
          53 49 43 20 6E 6F C
          74 20 61 76 61 69 C
          6C 61 62 6C 65 2C C
          20 0D 0A          C
F703 50 72 65 73 73 20 C                        db      'Press reset to re-boot...',CR,LF,LF,NUL
          72 65 73 65 74 20 C
          74 6F 20 72 65 2D C
          62 6F 6F 74 2E 2E C
          2E 0D 0A 0A 00   C
                                C
F720            C      basic_trap   endp
                                C
F720            C      code     ends
                                C      include pwrup4.asm
                                C
                                C
                                C      ;=====

```

```
C ;      Filename:      pwrup4.src
C ;
C ;      This module contains the i_fatal routine for powerup
C ;      diagnostics.
C ;
C ;=====
F720 C code segment public 'ROM'
C      assume cs:code, ds:nothing, es:nothing, ss:nothing
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 ;-----
F720 C i_fatal proc near
C      assume cs:code, ds:nothing, es:nothing, ss:nothing
C ; Disable 8237A p_dma Controller.
C
F720 B0 04 C      mov     al,dma_cmd_disable      ; disable p_dma controller command
F722 E6 08 C      out     dma_command,al
F724 90 C      nop                               ; chip needs time%
C
C ; Send a 'master clear' to 8237 p_dma Controller.
C
F725 E6 0D C      out     dma_master_clr,al      ; send master clear port any garbage
C
C ; Load 64k (0FFFFh+1) count for RAM refresh p_dma controller channel.
C
F727 B0 FF C      mov     al,0FFh
F729 E6 01 C      out     dma_count_0,al      ; low byte of count for 64k RAM refresh
F72B 90 C      nop                               ; chip needs time%
F72C 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
F72E B0 58 C      mov     al,dma_mode_0      ; mode for RAM refresh
F730 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
F732 B0 00 C      mov     al,dma_cmd_enable      ; enable p_dma controller
F734 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
F736 B0 00 C      mov     al,dma_unmask_0      ; turn on RAM refresh channel 0
F738 E6 0A C      out     dma_mask_bit,al
C
C ; Program p_8253_1 of i8254 p_timer to proper value for RAM refresh.
C
F73A B0 74 C      mov     al,t1cmd              ; select p_dma refresh counter
F73C E6 43 C      out     p_8253_ctrl,al
C
F73E B0 13 C      mov     al,t1count           ; load p_dma refresh count
F740 E6 41 C      out     p_8253_1,al
F742 32 C0 C      xor     al,al
F744 E6 41 C      out     p_8253_1,al
C
C      assume cs:code, ds:nothing, es:nothing, ss:stack_ram
C
F746 8C D7 C      mov     di,ss                ; save stack pointer
F748 8B EC C      mov     bp,sp
F74A BA 0030 C     mov     dx,stack_seg
F74D 8E D2 C      mov     ss,dx
F74F BC 0100 C     mov     sp,100h
C
F752 50 C      push    ax                  ; save error code
C
C ; Initialize & Disable 8259A Programmable Interrupt Controller.
C
F753 E8 E1A6 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.
F756 E8 E164 R C     call    i_vector
C
C ; Initialize Video.
C
F759 E8 E0A0 R C     call    i_d_init
C
C ; Display error message.
C
F75C 58 C      pop     ax                  ; restore error code
C
F75D E8 E540 R C     call    DRomString          ; display string at cs:si.
C
F760 BE D9CC R C     mov     si,cs:(offset fail_m)
F763 E8 E540 R C     call    DRomString          ; display fail message.
C
F766 0A E4 C      or      ah,ah                ; ah = 0?
F768 74 08 C      jz     i_fatal_ret          ; if so, no arguments
C
F76A E8 E56C R C     call    DColon              ; display a colon
C
F76D 8A C4 C      mov     al,ah                ; display error code
F76F E8 E589 R C     call    DHexByte
C
F772 C      i_fatal_ret:
F772 E8 E55F R C     call    DCrLf

```

```
C
C ;Output fatal error status for manufacturing tests
C
F775 BA 0378 C      mov    dx,378h          ; parallel port address
F778 EC      C      in     al,dx           ; read last checkpoint value
F779 34 3F   C      xor    al,03fh         ; extract checkpoint number from status
F77B EE      C      out   dx,al          ; output " Not OK - number"
C
C          assume cs:code, ds:nothing, es:nothing, ss:nothing
C
F77C 8E D7   C      mov    ss,di           ; restore stack pointer
F77E 8B E5   C      mov    sp,bp
C ;      ret
C
F780 F4      C      hlt
C
F781        C      i_fatal endp
C
F781        C      code  ends
C      include mem.asm
C
C
C ;=====
C ;      Filename:      mem.src
C ;
C ;      This module includes INT 12h, 11h, & 15h.
C ;
C ;=====
C
F781        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
F859      C m_cass proc far
C          assume cs:code, ds:nothing, es:nothing, ss:nothing
C
C ;      stc                                ; error
C ;      mov ah,86h
C ;      ret 2
C ;;     iret
C ;      jmp word ptr cs:add_mem_code
F859 EA   C      db 0eah
F85A E6F5 R C      dw add_mem_code
F85C F000  C      dw code_seg
F85E      C m_cass endp
C
F85E      C code ends
C include nmi.asm
C
C ;=====
C ;      Filename:      nmi.src
C ;
C ;      This module includes INT 02h.
C ;
C ;      And ENABLE_PARITY
C ;
C ;=====
C
F85E      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
F85F 50   C      push ax
F860 E4 62 C      in al,ControlC ; High two bits indicate parity.
F862 24 C0 C      and al,0C0h ; Mask of low 6 bits.
F864 74 0E C      jz n_out ; It wasn't a parity interrupt!
F866 BE E5FB R C      mov si,offset parity1_m ; System board message.
F869 D0 C0 C      rol al,1
F86B 72 03 C      jc n_1
F86D BE E618 R C      mov si,offset parity2_m ; Expansion board message.
F870      C n_1:
F870 E8 E540 R C      call DRomString
F873 F4   C      hlt
F874      C n_out:
F874 58   C      pop ax
F875 CF   C      iret
C
F876      C n_int endp
C
F876      C code ends

```



```
C ;=====
C
F876          C bt_int proc near
C              assume cs:code, ds:nothing, es:nothing, ss:nothing
C
F876 FB       C          sti                ; enable interrupts
F877 55       C          push bp           ; save BP & SI
F878 56       C          push si
C
C          assume ds:data                ; reset master table ptr to ROM
F879 2E: 8E 1E E538 R C          mov ds,word ptr cs:[set_ds_word] ; satisfy assumption
C          ;;EGA2 fix
C          ;;          mov word ptr ds:[master_tbl_ptr+0000h],cs:(offset mastab)
C          ;;          mov word ptr ds:[master_tbl_ptr+0002h],cs
C
F87E BE D935 R C          mov si,cs:(offset bt_m)        ; boot strap message
F881 E8 E540 R C          call DRomString              ; print banner
C
F884 32 DB    C          xor bl,bl                ; disable error message blinking
F886 53       C          push bx                    ; save blink status
C
C          assume ds:abs0, es:abs0
F887          C bt_o:                          ; boot strap outer loop
F887 33 C0    C          xor ax,ax                ; AX = abs0_seg.
F889 8E D8    C          mov ds,ax                ; satisfy assumptions
F88B 8E C0    C          mov es,ax
C
C          ; Reset fd_parms table vector.
C
F88D C7 06 0078 R EFC7 R C          mov word ptr ds:[int1Elocn+0],cs:(offset fd_parms)
F893 8C 0E 007A R C          mov word ptr ds:[int1Elocn+2],cs
C
C          ; Initialize retry loop.
C
F897 BD 0003   C          mov bp,3                ; retry counter
F89A          C bt_i:                          ; boot retry inner loop
C
C          ; Initialize the drive.
C
F89A 33 C0    C          xor ax,ax                ; AX = 0.
F89C 8B D8    C          mov bx,ax                ; BX = 0.
F89E 8B C8    C          mov cx,ax                ; CX = 0.
F8A0 8B D0    C          mov dx,ax                ; DX = 0.
F8A2 CD 13    C          INT 13h
F8A4 72 0A    C          jc bt_nxt                ; try again, if error
C
C          ; Read the boot sector.
C
F8A6 B8 0201   C          mov ax,0201h            ; read one sector
C          ; b1 = 0.
F8A9 B7 7C    C          mov bh,7Ch              ; xfer address = ES:BX = 0:7C00
C          ; cx = 0.
F8AB 41       C          inc cx                    ; track 0; sector 1
C          ; dx = 0.
C          ; head 0; drive 0
```

```

C
F8AC 06 C push es ; save return registers
F8AD CD 13 C INT 13h ; BX,CX,DX,SI,DI,BP, & DS saved
F8AF 07 C pop es ; restore return registers
C
F8B0 C bt_nxt:
F8B0 73 2E C jnc bt_ok ; jump if no error during read
C
F8B2 5B C pop bx ; get blink status
F8B3 0A DB C or bl,bl ; have 3 retries been completed?
F8B5 74 0E C jz bt_dec ; jump if no
C
F8B7 BE D94F R C mov si,cs:(offset bt_merr) ; blink error message on
F8BA 78 03 C js bt_blnk ; blink state from BL above
F8BC BE D974 R C mov si,cs:(offset bt_spaces) ; blink error message off
C
F8BF C bt_blnk:
F8BF E8 E540 R C call DRomString ; blink error message
F8C2 80 F3 80 C xor bl,10000000b ; toggle blink state
C
F8C5 C bt_dec:
F8C5 53 C push bx ; resave blink status
F8C6 4D C dec bp ; decrement retry count
F8C7 75 D1 C jnz bt_i ; and, try again
C
F8C9 5B C pop bx ; get blink status
F8CA 80 CB 01 C or bl,1 ; enable error message blinking
F8CD 53 C push bx ; save new status
F8CE A1 0060 R C mov ax,word ptr [int18locn]
F8D1 8D 1E F6E0 R C lea bx,cs:basic_trap
F8D5 3B D8 C cmp bx,ax
F8D7 5B C pop bx
F8D8 74 04 C je bt_again
F8DA CD 18 C int 18h
F8DC EB 02 C jmp short bt_ok
C
F8DE C bt_again:
F8DE EB A7 C jmp bt_o ; and try again, for now.
C
F8E0 C bt_ok:
F8E0 BE D974 R C mov si,cs:(offset bt_spaces) ; blink error message off
F8E3 E8 E540 R C call DRomString
F8E6 5E C pop si ; discard blink status
F8E7 5E C pop si ; restore SI
F8E8 8B EC C mov bp,sp
F8EA 89 5E 02 C mov word ptr ss:[bp+2],bx ; return IP = BX = 7C00h
F8ED 8C 46 04 C mov word ptr ss:[bp+4],es ; return CS = ES = 0000h
F8F0 5D C pop bp ; restore BP
F8F1 CF C iret ; return flags
C
F8F2 C bt_int endp
C
F8F2 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
F8F2 C code segment public 'ROM'
C      assume cs:code, ds:nothing, es:nothing, ss:nothing
C
F8F2 C c_data1 proc
C
C ; Days per year.
C
F8F2 0000 C c_dy_yr dw (0*366)+(0*365) ; year 0 = leap year + 0
F8F4 016E C      dw (1*366)+(0*365) ; year 1 = leap year + 1
F8F6 02DB C      dw (1*366)+(1*365) ; year 2 = leap year + 2
F8F8 0448 C      dw (1*366)+(2*365) ; year 3 = leap year + 3
F8FA 05B5 C      dw (1*366)+(3*365) ; year 4 = leap year + 0
F8FC 0723 C      dw (2*366)+(3*365) ; year 5 = leap year + 1
F8FE 0890 C      dw (2*366)+(4*365) ; year 6 = leap year + 2
F900 09FD C      dw (2*366)+(5*365) ; year 7 = leap year + 3
C
C ; Days per month.
C
F902 1F C c_dy_mo db 31 ; month 0 = Jan
F903 1C C      db 28 ; month 1 = Feb
F904 1F C      db 31 ; month 2 = Mar
F905 1E C      db 30 ; month 3 = Apr
F906 1F C      db 31 ; month 4 = May
F907 1E C      db 30 ; month 5 = Jun
F908 1F C      db 31 ; month 6 = Jul
F909 1F C      db 31 ; month 7 = Aug
F90A 1E C      db 30 ; month 8 = Sep
F90B 1F C      db 31 ; month 9 = Oct
F90C 1E C      db 30 ; month A = Nov
F90D 1F C      db 31 ; month B = Dec
C
F90E C c_data1 endp
C
C IF G4TOD
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
F90E          C  c_read  proc   near
C              assume  cs:code, ds:nothing, es:nothing, ss:nothing
C
C ; Save registers.
C
F90E  50      C          push   ax
C
C ; Years.
C
F90F  BA 0070 C          mov    dx,70h          ;; clear data changed flag
F912  EC      C          in     al,dx          ;;
C
F913          C  years:
C ;          mov    dx,7Fh          ; interrupts (years mod 8)%
F913  BA 007C C          mov    dx,7Ch          ; units year port%
C ;          in     al,dx          ;%
C ;          in     al,dx          ;%
F916  E8 F96C R C          call   c_rBCD          ; al = years
C
F919  8A E8   C          mov    ch,al          ; ch = saves year mod 8
C ; Months.
C
C ; 7Ch port = tens of mths for MM58274A chip%
C ;mov    dl,07Ch          ; dl = tens of months port = 7Ch%
F91B  B2 7B   C          mov    dl,07Bh          ; dl = tens of mths port, MM58274A chip%
F91D  E8 F97F 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
F920  48      C          dec    ax          ; ax = map month (1-12) to month (0-11)
C
F921  8B D8   C          mov    bx,ax          ; bx = saves month (0-11)
C
C ; Days.
C
C ;          dec    dx          ; dl = tens of days port = 79h
C ; dx was dec in c_rhex to 79 already, hence comment out original dec dx inst%
F923  E8 F97F 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
F926  48      C          dec    ax          ; ax = map days (1-?) to days (0-?)
C
C ; Calculate Day (ax has day).
C
F927  8B D0   C          mov    dx,ax          ; dx = day
C
C ; Calculate Month (bx has month).
C

```

ROM BIOS Listing

```

F929 4B          C      dec    bx          ; previous month
F92A 78 08       C      js     c_rm0        ; jump if it was zero
C
F92C           C      c_rmlp:
F92C E8 FA41 R   C      call   c_gdays       ; get days per month
F92F 03 D0       C      add    dx,ax          ; dx = day + current month
F931 4B          C      dec    bx          ; previous month
F932 79 F8       C      jns    c_rmlp
C
F934           C      c_rm0:              ; zero case
C                                     ; dx = day + month
C      ; Calculate Year (ch has month).
C
F934 33 DB       C      xor    bx,bx         ; clear bh
F936 8A DD       C      mov    bl,ch         ; get year mod 8
F938 D1 E3       C      shl   bx,1          ; make word index
F93A 2E: 03 97 F8F2 R C      add    dx,word ptr cs:[bx+c_dy_yr]
C
F93F 8B DA       C      mov    bx,dx         ; bx = day + month + year
C
C      ; Hours.
C
F941 B2 77       C      mov    dl,077h       ; dl = tens of hours port = 77h
F943 E8 F97F 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
F946 8A E8       C      mov    ch,al         ; ch = hours
C
C      ; Minutes.
C                                     ; dl = tens of minutes port = 75h
F948 E8 F97F 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
F94B 8A C8       C      mov    cl,al         ; cl = minutes
C
C      ; Seconds.
C                                     ; dl = tens of seconds port = 73h
F94D E8 F97F 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
F950 8A F0       C      mov    dh,al         ; dh = seconds
C
F952 52          C      push   dx           ; save seconds (dh)
C
C      ; Hundredths of Seconds.
C
C                                     ; dl = tenths of seconds port = 71h
F953 E8 F96C R   C      call   c_rBCD         ; al = tenths of seconds
F956 8A E0       C      mov    ah,al         ; move tenths of seconds to high byte
F958 32 C0       C      xor    al,al         ; ax = BCD of hundredths of seconds
F95A E8 F989 R   C      call   c_BCD2hex       ; ax = hex of hundredths of seconds
C
F95D 50          C      push   ax           ;;;; save ax
F95E BA 0070     C      mov    dx,70h        ;;;; read the data changed flag

```



```

F961 EC          C      in    al,dx          ;;; again
F962 A8 08       C      test   al,8h          ;;; if bit is set then reread the chip
F964 58          C      pop    ax            ;;; reset ax
F965 5A          C      pop    dx            ; restore seconds (dh)
F966 75 AB       C      jnz    years
C
F968 8A D0       C      mov    dl,al          ; dl = hex of hundredths of seconds
C
C      ; Restore registers.
C
C
F96A 58          C      pop    ax
F96B C3          C      ret
C
F96C 51          C c_rBCD: push  cx          ; save cx
F96D B9 0003     C      mov    cx,3          ; try 3 times only!!!
F970 32 F6       C      xor    dh,dh          ; clear dh
C
F972 EC          C c_rB1p: in    al,dx          ; get the byte
F973 24 0F       C      and    al,0Fh         ; clear high nibble
F975 3C 0A       C      cmp    al,10          ; is it less than 10?
F977 72 04       C      jb    c_rBret         ; if so, return
C
F979 E2 F7       C      loop   c_rB1p         ; else, try again
F97B B0 01       C      mov    al,1           ; if timeout, return one.
C
F97D             C c_rBret:
F97D 59          C      pop    cx            ; restore cx
F97E C3          C      ret
C
F97F             C c_read  endp
C
C      ENDIF
C      ENDIF
C      ;-----
C      ;      Convert to Hex  (c_rhex)
C      ;
C      ;      Inputs both BCD bytes and converts to hexadecimal 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
F97F             C c_rhex  proc  near
C      assume  cs:code, ds:nothing, es:nothing, ss:nothing
C
F97F E8 F96C R     C      call   c_rBCD          ; in from tens of whatever
F982 8A E0       C      mov    ah,al          ; move tens of whatever to high byte
F984 4A          C      dec    dx            ; dx points to units of whatever port
F985 E8 F96C R     C      call   c_rBCD          ; in from units of whatever
F988 4A          C      dec    dx            ; dx points to tens of previous port
C

```

```

C ;      jmp      short c_BCD2hex      ; fall through
C
F989      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
F989      C c_BCD2hex      proc      near
C
C             assume  cs:code, ds:nothing, es:nothing, ss:nothing
C
F989 8A F4      C      mov      dh,ah          ; dh = hi BCD digit
F98B D0 E6      C      shl      dh,1          ; dh = 2*(hi BCD digit)
F98D D0 E6      C      shl      dh,1          ; dh = 4*(hi BCD digit)
F98F 02 F4      C      add      dh,ah          ; dh = 5*(hi BCD digit)
F991 D0 E6      C      shl      dh,1          ; dh = 10*(hi BCD digit)
F993 02 C6      C      add      al,dh          ; al = 10*(hi BCD digit)+(low BCD digit)
F995 32 E4      C      xor      ah,ah          ; ax = 10*(hi BCD digit)+(low BCD digit)
F997 32 F6      C      xor      dh,dh          ; dh = 0
F999 C3          C      ret
C
F99A      C c_BCD2hex      endp
C
C IF G4TOD
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
F99A      C c_write proc      near
C
C             assume  cs:code, ds:nothing, es:nothing, ss:nothing
C
C ; Check for errors.
C
F99A 80 F9 3C      C      cmp      cl,60          ; cl = minutes (0-59)
F99D 72 01          C      jb      label1         ; jif within the range%
F99F C3            C      ret                    ; return (c_werr too far) %
F9A0          C label1:
F9A0 80 FD 18      C      cmp      ch,24          ; ch = hour (0-23)
F9A3 72 01          C      jb      label2         ; jif within the range%

```

```

F9A5 C3          C          ret                ; return (c_werr too far)  %
F9A6            C label2:                ;%
F9A6 81 FB 0B6A C          cmp      bx,(2*366)+(6*365) ; bx = day from leap year mod 8
F9AA 72 01       C          jb      label3                ;%
F9AC C3          C          ret                ; return (c_werr too far)  %
C                                     ;   = (0-2921) = (0-0B69h)
F9AD            C label3:                ;%
C
C ; Save registers.
C
F9AD 50          C          push   ax
F9AE 53          C          push   bx
F9AF 51          C          push   cx
F9B0 52          C          push   dx
C
C ; Initialize and Stop Clock.
C ; *****%
C ;      xor    ax,ax                ; ax = 0%
C ;      out   70h,al                ; test only port = out of test mode%
C ;      out   7Eh,al                ; stop/start port = stop clock%
C ; *****%
F9B1 B8 0005     C          mov     ax,5                ;%
F9B4 E6 70       C          out    70h,al                ; interrupt stop, clock stop%
F9B6 33 C0       C          xor    ax,ax                ;%
F9B8 E6 7F       C          out    7Fh,al                ; no interrupts programmed%
F9BA B8 0005     C          mov     ax,5                ;%
F9BD E6 70       C          out    70h,al                ; clock is out of test mode, halted,%
C                                     ; clock setting register is selected%
F9BF E6 7F       C          out    7Fh,al                ; select 24 hour mode, and LEAP YEAR%
C                                     ; counter 1 just for now, the LEAP YEAR%
C                                     ; counter gets set properly when year%
C                                     ; is calculated (see c_wypl: label)%
C ; Seconds
F9C1 B2 72       C          mov     dl,072h                ; dl= units of seconds port = 72h
F9C3 B0 00       C          mov     al,0                ; al=seconds
F9C5 E8 FA30 R   C          call   c_whex
C
C ; Minutes.
C
C ;;;; mov     dl,074h                ; dl = units of minutes port = 74h
F9C8 8A C1       C          mov     al,c1                ; al = minutes (0-59)
F9CA E8 FA30 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
F9CD 8A C5       C          mov     al,ch                ; al = hours (0-23)
F9CF E8 FA30 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
F9D2 8B D3       C          mov     dx,bx                ; dx = day from leap year mod 8

```

```
F9D4 BB 0010      C      mov     bx,(8*2)           ; word index of year
F9D7              C      c_wy1p:
F9D7 4B          C      dec     bx
F9D8 4B          C      dec     bx
F9D9 2E: 3B 97 F8F2 R  C      cmp     dx,word ptr cs:[bx+c_dy_yr]
F9DE 72 F7       C      jb     c_wy1p
C
F9E0 2E: 2B 97 F8F2 R  C      sub     dx,word ptr cs:[bx+c_dy_yr] ; dx = saves day of year
F9E5 D1 EB       C      shr     bx,1                ; bl = year mod 8
F9E7 8A EB       C      mov     ch,bl                ; ch = saves year mod 8
C
C ; Calculate LEAP YEAR counter for clock setting register%
F9E9 B8 0004      C      mov     ax,4                ;%
F9EC 3B C3       C      cmp     ax,bx                ; check if year is > 4%
F9EE 72 04       C      jb     leap1                ; jif less than 4 since 1984%
F9F0 2B C3       C      sub     ax,bx                ; ax = year since 1984 modulo 4%
F9F2 8B D8       C      mov     bx,ax                ; so that bx = modulo 4 LEAP%
C ; YEAR counter since 1984%
F9F4              C      leap1:                    ;%
F9F4 8B C3       C      mov     ax,bx                ;%
F9F6 D1 E0       C      shl     ax,1                ; Shift LEAP COUNTER into pro-%
F9F8 D1 E0       C      shl     ax,1                ;%
C ; bit position for clock reg%
F9FA 0C 01       C      or     al,1                ; set 24-hour mode%
F9FC E6 7F       C      out    7Fh,al              ; set LEAP counter & 24-hour%
C
C ; Calculate Days & Months.
C
F9FE BB FFFF      C      mov     bx,-1                ; start at January
FA01              C      c_wmlp:
FA01 43          C      inc     bx                    ; next month
FA02 E8 FA41 R    C      call    c_gdays              ; get days per month
FA05 2B D0       C      sub     dx,ax
FA07 73 F8       C      jae     c_wmlp
C ; bx = month (0-11)
FA09 03 C2       C      add     ax,dx                ; ax = day (0-?)
C
C ; Days.
C
FA0B B2 78       C      mov     dl,078h              ; dl = units of days port = 78h
FA0D 40          C      inc     ax                    ; al = map days (0-?) to days (1-?)
FA0E E8 FA30 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 ; For MM58274 chip 7Ah is Units Months port
C ; Months.
C
C ; inc dx ; dl = units of months port = 7Bh%
C ; comment out the above because 7Ah is already units of months%
FA11 8B C3       C      mov     ax,bx                ; al = month (0-11)
FA13 40          C      inc     ax                    ; al = map month (0-11) to month (1-12)
FA14 E8 FA30 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, MM58274A%
C ; dx = leap year port = 7Dh%
C ; mov al,08h ; set leap year bit%
C ; mov cl,ch ; get year mod 8%
C ; and cl,03h ; get year mod 4%
C ; shr al,cl ; shift leap year bit into position%
C ; out dx,al ;%
C ; *****%
C ; dx = leap year port = 7Dh, MM58274A%
C
C ; Years.
C
C ; inc dx ; dx = stop/start = 7Eh%
C ; inc dx ; dx = interrupt = 7Fh%
C ; mov al,ch ; get year mod 8%
C ; or al,08h ; set 'repeated interrupt' bit%
C ; out dx,al%
C ; nop%
C ; in al,dx%
C ; nop%
C ; in al,dx%
C ; nop%
C ; in al,dx%
FA17 8A C5 C mov al,ch ; get year since beginning of time %
FA19 E6 7C C out 7Ch,al ; units year port%
C
C ; Start Clock.
C
C ; mov al,0FFh ; al = 0FFh%
C ; dec dx ; dx = stop/start = 7Eh%
C ; out dx,al ; start clock%
FA1B B8 0002 C mov ax,2 ;%
FA1E E6 70 C out 70h,al ; select the interrupt register%
FA20 B8 000B C mov ax,0Bh ; repeated interrupts, every second%
FA23 E6 7F C out 7Fh,al ;%
FA25 33 C0 C xor ax,ax ;%
FA27 E6 70 C out 70h,al ; start the clock%
C
C ; Restore registers.
C
FA29 5A C pop dx
FA2A 59 C pop cx
FA2B 5B C pop bx
FA2C 58 C pop ax
FA2D 32 E4 C xor ah,ah ; ah = 0 no error
C
FA2F C3 C c_werr: ret
C
FA30 C c_write endp
C ENDIF
C

```

```

C   ENDIF
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
FA30      C   c_whex proc   near
C           assume  cs:code, ds:nothing, es:nothing, ss:nothing
C
FA30 32 E4      C       xor    ah,ah           ; ax = hexadecimal byte
FA32 B6 0A      C       mov    dh,10          ; dh = divisor
FA34 F6 F6      C       div    dh             ; ah = remainder = low BCD digit (0-9)
C                                   ; al = quotient = high BCD digit
FA36 86 C4      C       xchg   al,ah          ; ah = quotient = high BCD digit
C                                   ; al = remainder = low BCD digit (0-9)
C
FA38 32 F6      C       xor    dh,dh          ; dx points to units of whatever port
FA3A EE         C       out    dx,al          ; out to units of whatever
FA3B 8A C4      C       mov    al,ah          ; move tens of whatever to low byte
FA3D 42         C       inc    dx             ; dx points to tens of whatever port
FA3E EE         C       out    dx,al          ; out to tens of whatever
FA3F 42         C       inc    dx             ; dx points to units of next port
FA40 C3         C       ret
C
FA41      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
FA41      C   c_gdays proc   near
C           assume  cs:code, ds:nothing, es:nothing, ss:nothing
C
FA41 33 C0      C       xor    ax,ax           ; clear ah
FA43 2E: 8A 87 F902 R  C       mov    al,cs:[bx+c_dy_mo]
C
FA48 80 FB 01      C       cmp    bl,1           ; is is February?
FA4B 75 06         C       jnz   c_gret          ; if not February, return
C
FA4D F6 C5 03      C       test   ch,03h         ; if year = 0 mod 4, leap year

```


ROM BIOS Listing

FAF6	02 0E 3E FE 3E 0E 02 00	C C	DB	02h,0eh,3eh,0feh,3eh,0eh,02h,00h	; 11
FAFE	18 3C 7E 18 18 7E 3C 18	C C	DB	18h,3ch,7eh,18h,18h,7eh,3ch,18h	; 12
FB06	66 66 66 66 66 00 66 00	C C	DB	66h,66h,66h,66h,66h,00h,66h,00h	; 13
FB0E	7F DB DB 7B 1B 1B 1B 00	C C	DB	7fh,0dbh,0dbh,7bh,1bh,1bh,1bh,00h	; 14
FB16	3E 63 38 6C 6C 38 CC 78	C C	DB	3eh,63h,38h,6ch,6ch,38h,0cch,78h	; 15
FB1E	00 00 00 00 7E 7E 7E 00	C C	DB	00h,00h,00h,00h,7eh,7eh,7eh,00h	; 16
FB26	18 3C 7E 18 7E 3C 18 FF	C C	DB	18h,3ch,7eh,18h,7eh,3ch,18h,0ffh	; 17
FB2E	18 3C 7E 18 18 18 18 00	C C	DB	18h,3ch,7eh,18h,18h,18h,18h,00h	; 18
FB36	18 18 18 18 7E 3C 18 00	C C	DB	18h,18h,18h,18h,7eh,3ch,18h,00h	; 19
FB3E	00 18 0C FE 0C 18 00 00	C C	DB	00h,18h,0ch,0feh,0ch,18h,00h,00h	; 1a
FB46	00 30 60 FE 60 30 00 00	C C	DB	00h,30h,60h,0feh,60h,30h,00h,00h	; 1b
FB4E	00 00 C0 C0 C0 FE 00 00	C C	DB	00h,00h,0c0h,0c0h,0c0h,0feh,00h,00h	; 1c
FB56	00 24 66 FF 66 24 00 00	C C	DB	00h,24h,66h,0ffh,66h,24h,00h,00h	; 1d
FB5E	00 18 3C 7E FF FF 00 00	C C	DB	00h,18h,3ch,7eh,0ffh,0ffh,00h,00h	; 1e
FB66	00 FF FF 7E 3C 18 00 00	C C	DB	00h,0ffh,0ffh,7eh,3ch,18h,00h,00h	; 1f
FB6E	00 00 00 00 00 00 00 00	C C	DB	00h,00h,00h,00h,00h,00h,00h,00h	; ' ' 20
FB76	30 78 78 30 30 00 30 00	C C	DB	30h,78h,78h,30h,30h,00h,30h,00h	; '! ' 21
FB7E	6C 6C 6C 00 00 00 00 00	C C	DB	6ch,6ch,6ch,00h,00h,00h,00h,00h	; '" ' 22
FB86	6C 6C FE 6C FE 6C 6C 00	C C	DB	6ch,6ch,0feh,6ch,0feh,6ch,6ch,00h	; '# ' 23
FB8E	30 7C C0 78 0C F8 30 00	C C	DB	30h,7ch,0c0h,78h,0ch,0f8h,30h,00h	; '\$ ' 24
FB96	00 C6 CC 18 30 66 C6 00	C C	DB	00h,0c6h,0cch,18h,30h,66h,0c6h,00h	; '% ' 25
FB9E	38 6C 38 76 DC CC 76 00	C C	DB	38h,6ch,38h,76h,0dch,0cch,76h,00h	; '& ' 26
FBA6	60 60 C0 00 00 00 00 00	C C	DB	60h,60h,0c0h,00h,00h,00h,00h,00h	; ' ' ' 27
FBAE	18 30 60 60 60 30 18 00	C C	DB	18h,30h,60h,60h,60h,30h,18h,00h	; '(' 28
FBB6	60 30 18 18 18 30 60 00	C C	DB	60h,30h,18h,18h,18h,30h,60h,00h	; ') ' 29
FBBE	00 66 3C FF 3C 66 00 00	C C	DB	00h,66h,3ch,0ffh,3ch,66h,00h,00h	; '* ' 2a
FBC6	00 30 30 FC 30 30 00 00	C C	DB	00h,30h,30h,0fch,30h,30h,00h,00h	; '+ ' 2b
FBCE	00 00 00 00 00 30	C	DB	00h,00h,00h,00h,00h,30h,30h,60h	; ', ' 2c

	30 60	C				
FBD6	00 00 00 FC 00 00 00 00	C C	DB	00h,00h,00h,0fch,00h,00h,00h,00h	; '-'	2d
FBDE	00 00 00 00 00 30 30 00	C C	DB	00h,00h,00h,00h,00h,30h,30h,00h	; '.'	2e
FBE6	06 0C 18 30 60 C0 80 00	C C	DB	06h,0ch,18h,30h,60h,0c0h,80h,00h	; '/'	2f
FBEE	7C C6 CE DE F6 E6 7C 00	C C	DB	7ch,0c6h,0ceh,0deh,0f6h,0e6h,7ch,00h	; '0'	30
FBF6	30 70 30 30 30 30 FC 00	C C	DB	30h,70h,30h,30h,30h,30h,0fch,00h	; '1'	31
FBFE	78 CC 0C 38 60 CC FC 00	C C	DB	78h,0cch,0ch,38h,60h,0cch,0fch,00h	; '2'	32
FC06	78 CC 0C 38 0C CC 78 00	C C	DB	78h,0cch,0ch,38h,0ch,0cch,78h,00h	; '3'	33
FC0E	1C 3C 6C CC FE 0C 1E 00	C C	DB	1ch,3ch,6ch,0cch,0feh,0ch,1eh,00h	; '4'	34
FC16	FC C0 F8 0C 0C CC 78 00	C C	DB	0fch,0c0h,0f8h,0ch,0ch,0cch,78h,00h	; '5'	35
FC1E	38 60 C0 F8 CC CC 78 00	C C	DB	38h,60h,0c0h,0f8h,0cch,0cch,78h,00h	; '6'	36
FC26	FC CC 0C 18 30 30 30 00	C C	DB	0fch,0cch,0ch,18h,30h,30h,30h,00h	; '7'	37
FC2E	78 CC CC 78 CC CC 78 00	C C	DB	78h,0cch,0cch,78h,0cch,0cch,78h,00h	; '8'	38
FC36	78 CC CC 7C 0C 18 70 00	C C	DB	78h,0cch,0cch,7ch,0ch,18h,70h,00h	; '9'	39
FC3E	00 30 30 00 00 30 30 00	C C	DB	00h,30h,30h,00h,00h,30h,30h,00h	; ':'	3a
FC46	00 30 30 00 00 30 30 60	C C	DB	00h,30h,30h,00h,00h,30h,30h,60h	; ';'	3b
FC4E	18 30 60 C0 60 30 18 00	C C	DB	18h,30h,60h,0c0h,60h,30h,18h,00h	; '<'	3c
FC56	00 00 FC 00 00 FC 00 00	C C	DB	00h,00h,0fch,00h,00h,0fch,00h,00h	; '='	3d
FC5E	60 30 18 0C 18 30 60 00	C C	DB	60h,30h,18h,0ch,18h,30h,60h,00h	; '>'	3e
FC66	78 CC 0C 18 30 00 30 00	C C	DB	78h,0cch,0ch,18h,30h,00h,30h,00h	; '?'	3f
FC6E	7C C6 DE DE DE C0 78 00	C C	DB	7ch,0c6h,0deh,0deh,0deh,0c0h,78h,00h	; '@'	40
FC76	30 78 CC CC FC CC CC 00	C C	DB	30h,78h,0cch,0cch,0fch,0cch,0cch,00h	; 'A'	41
FC7E	FC 66 66 7C 66 66 FC 00	C C	DB	0fch,66h,66h,7ch,66h,66h,0fch,00h	; 'B'	42
FC86	3C 66 C0 C0 C0 66 3C 00	C C	DB	3ch,66h,0c0h,0c0h,0c0h,66h,3ch,00h	; 'C'	43
FC8E	F8 6C 66 66 66 6C F8 00	C C	DB	0f8h,6ch,66h,66h,66h,6ch,0f8h,00h	; 'D'	44
FC96	FE 62 68 78 68 62 FE 00	C C	DB	0feh,62h,68h,78h,68h,62h,0feh,00h	; 'E'	45
FC9E	FE 62 68 78 68 60 F0 00	C C	DB	0feh,62h,68h,78h,68h,60h,0f0h,00h	; 'F'	46
FCA6	3C 66 C0 C0 CE 66 3E 00	C C	DB	3ch,66h,0c0h,0c0h,0ceh,66h,3eh,00h	; 'G'	47

ROM BIOS Listing

FCAE	CC CC CC FC CC CC CC 00	C C	DB	0cch,0cch,0cch,0fch,0cch,0cch,0cch,00h	;'H' 48
FCB6	78 30 30 30 30 30 78 00	C C	DB	78h,30h,30h,30h,30h,30h,78h,00h	;'I' 49
FCBE	1E 0C 0C 0C CC CC 78 00	C C	DB	1eh,0ch,0ch,0ch,0cch,0cch,78h,00h	;'J' 4a
FCC6	E6 66 6C 78 6C 66 E6 00	C C	DB	0e6h,66h,6ch,78h,6ch,66h,0e6h,00h	;'K' 4b
FCCE	F0 60 60 60 62 66 FE 00	C C	DB	0f0h,60h,60h,60h,62h,66h,0feh,00h	;'L' 4c
FCD6	C6 EE FE FE D6 C6 C6 00	C C	DB	0c6h,0eeh,0feh,0feh,0d6h,0c6h,0c6h,00h	;'M' 4d
FCDE	C6 E6 F6 DE CE C6 C6 00	C C	DB	0c6h,0e6h,0f6h,0deh,0ceh,0c6h,0c6h,00h	;'N' 4e
FCE6	38 6C C6 C6 C6 6C 38 00	C C	DB	38h,6ch,0c6h,0c6h,0c6h,6ch,38h,00h	;'O' 4f
FCEE	FC 66 66 7C 60 60 F0 00	C C	DB	0fch,66h,66h,7ch,60h,60h,0f0h,00h	;'P' 50
FCF6	78 CC CC CC DC 78 1C 00	C C	DB	78h,0cch,0cch,0cch,0dch,78h,1ch,00h	;'Q' 51
FCFE	FC 66 66 7C 6C 66 E6 00	C C	DB	0fch,66h,66h,7ch,6ch,66h,0e6h,00h	;'R' 52
FD06	78 CC E0 70 1C CC 78 00	C C	DB	78h,0cch,0e0h,70h,1ch,0cch,78h,00h	;'S' 53
FD0E	FC B4 30 30 30 30 78 00	C C	DB	0fch,0b4h,30h,30h,30h,30h,78h,00h	;'T' 54
FD16	CC CC CC CC CC CC FC 00	C C	DB	0cch,0cch,0cch,0cch,0cch,0cch,0fch,00h	;'U' 55
FD1E	CC CC CC CC CC 78 30 00	C C	DB	0cch,0cch,0cch,0cch,0cch,78h,30h,00h	;'V' 56
FD26	C6 C6 C6 D6 FE EE C6 00	C C	DB	0c6h,0c6h,0c6h,0d6h,0feh,0eeh,0c6h,00h	;'W' 57
FD2E	C6 C6 6C 38 38 6C C6 00	C C	DB	0c6h,0c6h,6ch,38h,38h,6ch,0c6h,00h	;'X' 58
FD36	CC CC CC 78 30 30 78 00	C C	DB	0cch,0cch,0cch,78h,30h,30h,78h,00h	;'Y' 59
FD3E	FE C6 8C 18 32 66 FE 00	C C	DB	0feh,0c6h,8ch,18h,32h,66h,0feh,00h	;'Z' 5a
FD46	78 60 60 60 60 60 78 00	C C	DB	78h,60h,60h,60h,60h,78h,00h	;'[' 5b
FD4E	C0 60 30 18 0C 06 02 00	C C	DB	0c0h,60h,30h,18h,0ch,06h,02h,00h	;'^' 5c
FD56	78 18 18 18 18 18 78 00	C C	DB	78h,18h,18h,18h,18h,78h,00h	;']' 5d
FD5E	10 38 6C C6 00 00 00 00	C C	DB	10h,38h,6ch,0c6h,00h,00h,00h,00h	;'^' 5e
FD66	00 00 00 00 00 00 00 FF	C C	DB	00h,00h,00h,00h,00h,00h,00h,0ffh	;'_' 5f
FD6E	30 30 18 00 00 00 00 00	C C	DB	30h,30h,18h,00h,00h,00h,00h,00h	;''' 60
FD76	00 00 78 0C 7C CC 76 00	C C	DB	00h,00h,78h,0ch,7ch,0cch,76h,00h	;'a' 61
FD7E	E0 60 60 7C 66 66 DC 00	C C	DB	0e0h,60h,60h,7ch,66h,66h,0dch,00h	;'b' 62
FD86	00 00 78 CC C0 CC C	C	DB	00h,00h,78h,0cch,0c0h,0cch,78h,00h	;'c' 63

	78 00	C				
FD8E	1C 0C 0C 7C CC CC	C	DB	1ch,0ch,0ch,7ch,0cch,0cch,76h,00h	;	'd' 64
	76 00	C				
FD96	00 00 78 CC FC C0	C	DB	00h,00h,78h,0cch,0fch,0c0h,78h,00h	;	'e' 65
	78 00	C				
FD9E	38 6C 60 F0 60 60	C	DB	38h,6ch,60h,0f0h,60h,60h,0f0h,00h	;	'f' 66
	F0 00	C				
FDA6	00 00 76 CC CC 7C	C	DB	00h,00h,76h,0cch,0cch,7ch,0ch,0f8h	;	'g' 67
	0C F8	C				
FDAE	E0 60 6C 76 66 66	C	DB	0e0h,60h,6ch,76h,66h,66h,0e6h,00h	;	'h' 68
	E6 00	C				
FDB6	30 00 70 30 30 30	C	DB	30h,00h,70h,30h,30h,30h,78h,00h	;	'i' 69
	78 00	C				
FDBE	0C 00 0C 0C 0C CC	C	DB	0ch,00h,0ch,0ch,0ch,0cch,0cch,78h	;	'j' 6a
	CC 78	C				
FDC6	E0 60 66 6C 78 6C	C	DB	0e0h,60h,66h,6ch,78h,6ch,0e6h,00h	;	'k' 6b
	E6 00	C				
FDCE	70 30 30 30 30 30	C	DB	70h,30h,30h,30h,30h,30h,78h,00h	;	'l' 6c
	78 00	C				
FDD6	00 00 CC FE FE D6	C	DB	00h,00h,0cch,0feh,0feh,0d6h,0c6h,00h	;	'm' 6d
	C6 00	C				
FDDE	00 00 F8 CC CC CC	C	DB	00h,00h,0f8h,0cch,0cch,0cch,0cch,00h	;	'n' 6e
	CC 00	C				
FDE6	00 00 78 CC CC CC	C	DB	00h,00h,78h,0cch,0cch,0cch,78h,00h	;	'o' 6f
	78 00	C				
FDEE	00 00 DC 66 66 7C	C	DB	00h,00h,0dch,66h,66h,7ch,60h,0f0h	;	'p' 70
	60 F0	C				
FDF6	00 00 76 CC CC 7C	C	DB	00h,00h,76h,0cch,0cch,7ch,0ch,1eh	;	'q' 71
	0C 1E	C				
FDFE	00 00 DC 76 66 60	C	DB	00h,00h,0dch,76h,66h,60h,0f0h,00h	;	'r' 72
	F0 00	C				
FE06	00 00 7C C0 78 0C	C	DB	00h,00h,7ch,0c0h,78h,0ch,0f8h,00h	;	's' 73
	F8 00	C				
FE0E	10 30 7C 30 30 34	C	DB	10h,30h,7ch,30h,30h,34h,18h,00h	;	't' 74
	18 00	C				
FE16	00 00 CC CC CC CC	C	DB	00h,00h,0cch,0cch,0cch,0cch,76h,00h	;	'u' 75
	76 00	C				
FE1E	00 00 CC CC CC 78	C	DB	00h,00h,0cch,0cch,0cch,78h,30h,00h	;	'v' 76
	30 00	C				
FE26	00 00 C6 D6 FE FE	C	DB	00h,00h,0c6h,0d6h,0feh,0feh,6ch,00h	;	'w' 77
	6C 00	C				
FE2E	00 00 C6 6C 38 6C	C	DB	00h,00h,0c6h,6ch,38h,6ch,0c6h,00h	;	'x' 78
	C6 00	C				
FE36	00 00 CC CC CC 7C	C	DB	00h,00h,0cch,0cch,0cch,7ch,0ch,0f8h	;	'y' 79
	0C F8	C				
FE3E	00 00 FC 98 30 64	C	DB	00h,00h,0fch,98h,30h,64h,0fch,00h	;	'z' 7a
	FC 00	C				
FE46	1C 30 30 E0 30 30	C	DB	1ch,30h,30h,0e0h,30h,30h,1ch,00h	;	'{' 7b
	1C 00	C				
FE4E	18 18 18 00 18 18	C	DB	18h,18h,18h,00h,18h,18h,18h,00h	;	' ' 7c
	18 00	C				
FE56	E0 30 30 1C 30 30	C	DB	0e0h,30h,30h,1ch,30h,30h,0e0h,00h	;	'}' 7d
	E0 00	C				
FE5E	76 DC 00 00 00 00	C	DB	76h,0dch,00h,00h,00h,00h,00h	;	'~' 7e
	00 00	C				

ROM BIOS Listing

```

FE66 00 10 38 6C C6 C6   C          DB      00h,10h,38h,6ch,0c6h,0c6h,0feh,00h   ;'' 7f
      FE 00              C
                        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 F90E 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 F99A 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 E53A 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 -- i8254 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              assume cs:code, ds:data, es:nothing, ss:nothing
C
FEA8 2E: 8E 1E E538 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 ED50 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
C      ;;;; sti ; enable interrupts
C              ; (no more shared variables)
C
C      ; Invoke any user p_timer break routine.
C
FEE1 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
```

```

FEE3 B0 60      C      mov     al,pic_seoi_0      ; specific end of interrupt command
FEE5 E6 20      C      out     pic_0,al          ; to pic 'command port.
FEE7 EB 01 90   C      .jmp    drb
FEEA           C      drb:
FEEA FB        C      sti
C
FEEB 1F        C      pop     ds          ; restore registers
FEEC 5A        C      pop     dx
FEED 58        C      pop     ax
FEEE CF        C      iret
C
FEEF           C      t_int  endp
C
FEEF           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      ;=====
C
FEEF           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     ; int0Alocn
FEF9 FF23 R     C      dw     ill_int     ; int0Blocn
FEFB FF23 R     C      dw     ill_int     ; int0Clocn
FEFD FF23 R     C      dw     ill_int     ; int0Dlocn
FEFF EF57 R     C      dw     fd_int     ; int0Elocn  see dsk.src
FF01 FF23 R     C      dw     ill_int     ; int0Flocn
C
FF03 F065 R     C      dw     v_io       ; int10locn  see vid.src
FF05 F84D R     C      dw     m_equip     ; int11locn  see mem.src
FF07 F841 R     C      dw     m_size     ; int12locn  see mem.src
FF09 EC59 R     C      dw     fd_io       ; int13locn  see dsk.src
FF0B E739 R     C      dw     serial_io   ; int14locn  see com.src
FF0D F859 R     C      dw     m_cass     ; int15locn  see mem.src
FF0F E82E R     C      dw     k_io       ; int16locn  see kb.src
FF11 EFD2 R     C      dw     p_io       ; int17locn  see prn.src
C
FF13 F6E0 R     C      dw     basic_trap ; int18locn  see int18.src
FF15 F876 R     C      dw     bt_int     ; int19locn  see boot.src
FF17 FE6E R     C      dw     t_day      ; int1Alocn  see rtc.src
FF19 FF4B R     C      dw     dummy_iret ; int1Blocn  see kb.src
FF1B FF4B R     C      dw     dummy_iret ; int1Clocn  see rtc.src
FF1D F0A4 R     C      dw     v_parms    ; int1Dlocn  see vid.src

```

ROM BIOS Listing

```

FF1F EFC7 R      C      dw      fd_parms          ; int1Elocn   see dsk.src
FF21 C860 R      C      dw      font_hi_8x8        ; int1Flocn   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 = 0CW3 -- 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                ; 0CW1 -- 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             ; 0CW2 -- 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 E4BE R    C      call    ill_trap                ; every register but ds saved!
C
C      assume cs:code, ds:nothing, es:nothing, ss:nothing
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 E53A 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
FF54             C      ORG     0FF54h
C
FF54             C      s_int   proc   near

```

```

C      assume cs:code, ds:nothing, es:nothing, ss:nothing
C ;=====
C ;      INT 05H -- Check for Bound instruction%
C ;
C ;      Purpose:      To intercept the interrupt generated by the %
C ;                   bound instruction if the bound test fails.%
C ;                   If the last instruction executed was a bound %
C ;                   then iret, else print the screen.%
C ;
C ;=====
C
FF54 FA      C      cli          ; stop intrs%
C                   ; This front end routine checks for bound instruction.%
FF55 50      C      push   ax          ;%
FF56 56      C      push   si          ;going to trash these registers%
FF57 06      C      push   es          ;%
FF58 83 C4 06 C      add    sp, 6          ;go back to old sp%
C
FF5B 5E      C      pop    si          ;old ip%
FF5C 07      C      pop    es          ;old cs%
FF5D 26 8B 04 C      mov    ax, es:[si]    ;ax has desired opcode %
FF60 83 EC 0A C      sub    sp, 0ah        ;normal sp%
FF63 07      C      pop    es          ;original values of es and si%
FF64 5E      C      pop    si          ;%
C
C                   ; Compare offending instruction with known opcode%
FF65 2C 62   C      sub    al, 062H      ; 62 /r is the opcode for bound %
FF67 58      C      pop    ax          ;restore ax%
FF68 75 02   C      jnz   notbound     ;if not a bound instruction jmp to notbound%
FF6A FB      C      sti          ; reenable interrupts%
C
FF6B CF      C      iret         ;%
C
FF6C        C      notbound:      ; print screen routine %
C
FF6C 1E      C      push   ds          ; save ds
C
C      assume cs:code, ds:data, es:nothing, ss:nothing
C
FF6D 2E: 8E 1E E538 R C      mov    ds,word ptr cs:[set_ds_word] ; satisfy assumptions
C
FF72 52      C      push   dx
FF73 B2 01   C      mov    dl,1          ; 1 = locked
FF75 F0/ 86 16 0100 C lock  xchg  byte ptr ds:[100h],dl    ; check monitor lock
FF7A FE CA   C      dec    dl          ; already locked ?
FF7C 74 4C   C      jz    s_nop        ; if set, do nothing
FF7E FB      C      sti          ; enable interrupts after
C                   ; monitor lock code!!!!
C
FF7F 51      C      push   cx          ; save registers
FF80 53      C      push   bx
FF81 50      C      push   ax
C
C ; Get Current Video Width.
C

```

```

FF82 B4 0F      C      mov     ah,0Fh      ; call v_video_state
FF84 CD 10      C      INT     10h          ; Output: ah = crt_cols
                  C                      ;         al = crt_mode
                  C                      ;         bh = active_page
                  C
FF86 8A DC      C      mov     bl,ah          ; save ah = crt_cols
                  C
                  C ; Get Current Cursor Position.
                  C
FF88 B4 03      C      mov     ah,03h        ; call v_read_cursor
FF8A CD 10      C      INT     10h          ; Input:  bh = active_page
                  C                      ; Output:
                  C                      ; (dh,d1) = (row,col) of cursor
                  C                      ; (ch,c1) = cursor mode setting
                  C
FF8C 8A EB      C      mov     ch,bl          ;get crt_cols
FF8E 52         C      push   dx            ; save row, col of cursor
FF8F B1 FF      C      mov     cl,-1         ; initialize cl = printer error
                  C
                  C ; Loop Through the Screen.
                  C
FF91 BA 0000    C      mov     dx,0           ; (dh,d1) = (row,col) of origin
                  C
FF94 E8 FFCD R  C      call   s_eol           ; print a new line
FF97 75 24     C      jnz    s_err          ; any errors?
                  C
FF99 E8 FFE1 R  C s_lp: call   s_get       ; get next character from screen
                  C
                  C ; Map Invalid Characters to Space.
                  C
FF9C 3C 00     C      cmp     al,0           ; check validity of character.
FF9E 75 02     C      jne    s_ok           ; if valid, we're ok.
FFA0 B0 20     C      mov     al,' '        ; if invalid, print a space.
FFA2         C s_ok:
                  C
                  C ; Print the Character.
                  C
FFA2 E8 FFD4 R  C      call   s_out         ;
FFA5 75 16     C      jnz    s_err          ; any errors?
                  C
                  C ; Advance to Next Character.
                  C
FFA7 FE C2     C      inc     dl            ; advance column (1-crt_cols)
FFA9 3A D5     C      cmp     dl,ch          ; dl < ch = crt_cols?
FFAB 7C EC     C      jl     s_lp           ; if so, continue
                  C
FFAD E8 FFCD R  C      call   s_eol           ; else print a new line
FFB0 75 0B     C      jnz    s_err          ; any errors?
                  C
FFB2 32 D2     C      xor     dl,dl          ; move column back to 0
FFB4 FE C6     C      inc     dh            ; advance row (1-25)
FFB6 80 FE 19  C      cmp     dh,25          ; dh < 25
FFB9 7C DE     C      jl     s_lp           ; if so, continue
                  C
FFBB 33 C9     C      xor     cx,cx          ; set cl = printer no error

```



```

; it's data area, i.e. osmerge1 and osmerge2. Do NOT change change
; the code to jump elsewhere as this will break OSMERGE
;::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

FFEA          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 DAD3 R   dw      diagnostics_1 ; instruction pointer
FFF3 F000     dw      code_seg       ; code segment          F000h

FFF5 31 32 2F 31 35 2F   db      '12/15/85' ; release marker (exactly 8 bytes!!!!)
      38 35

FFFD 00       chk_hi db      0           ;
FFFE         ORG     0FFFEh
FFFE FE       db      0FEh           ; for "compatibility"

FFFF          vector   endp    near

FFFF          code    ends

              end

```

Macros:

Name	Length
JMPF	0001

Segments and Groups:

Name	Size	Align	Combine	Class
ABSO	0080	PARA	PUBLIC	'RAM'
CODE	FFFF	PARA	PUBLIC	'ROM'
DATA	00D3	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	L BYTE	00D2	DATA
ADDR_MARK_ERROR.	Number	0002	
ADD_MEM_CODE	F PROC	E6F5	CODE Length =0016
ALT_INPUT.	L BYTE	0019	DATA
ALT_KEY.	Number	0038	

ROM BIOS Listing

ALT_RET	L WORD	E293	CODE	
ALT_SHIFT	Number	0008		
BANNER_M	L BYTE	D912	CODE	
BASIC_TRAP	N PROC	F6E0	CODE	Length =0040
BEL	Number	0007		
BELL_WAIT	L NEAR	F59C	CODE	
BETA	Number	0000		
BIOS_BREAK	L BYTE	0071	DATA	
BIOS_INSTALL	L NEAR	0000	CODE	External
BITREAD	Number	3FA0		
BS	Number	0008		
BT_AGAIN	L NEAR	F8DE	CODE	
BT_BLNK	L NEAR	F8BF	CODE	
BT_DEC	L NEAR	F8C5	CODE	
BT_I	L NEAR	F89A	CODE	
BT_INT	N PROC	F876	CODE	Length =007C
BT_JMP	N PROC	E6F2	CODE	Length =0003
BT_M	L BYTE	D935	CODE	
BT_MERR	L BYTE	D94F	CODE	
BT_NXT	L NEAR	F8B0	CODE	
BT_O	L NEAR	F887	CODE	
BT_OK	L NEAR	F8E0	CODE	
BT_SPACES	L BYTE	D974	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	
CAPS_LOCK_KEY	Number	003A		
CAPS_LOCK_MODE	Number	0040		
CAPS_LOCK_SHIFT	Number	0040		
CASS	L NEAR	E705	CODE	
CHKSPEED	N PROC	F672	CODE	Length =0014
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		
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	E8E2	CODE	Length =0023
COM_ID_A	Number	03FA		
COM_ID_B	Number	02FA		
COM_INIT	N PROC	E787	CODE	Length =003A
COM_OE	Number	0002		
COM_PB	N PROC	E8B6	CODE	Length =002C

COM_PE	Number	0004		
COM_RTS.	Number	0002		
COM_RXD.	Number	0001		
COM_STAT	L NEAR	E87D	CODE	
COM_TE	Number	0080		
COM_TXD.	Number	0020		
CONTROLC	Number	0062		
CONTROL_BYTE	L BYTE	0076	DATA	
CONT_CNT	L NEAR	DE12	CODE	
CR	Number	000D		
CRC_ERROR.	Number	0010		
CUR_CYL.	L BYTE	0094	DATA	
C_BCD2HEX.	N PROC	F989	CODE	Length =0011
C_DATA1.	N PROC	F8F2	CODE	Length =001C
C_DY_MO.	L BYTE	F902	CODE	
C_DY_YR.	L WORD	F8F2	CODE	
C_GDAYS.	N PROC	FA41	CODE	Length =0013
C_GRET	L NEAR	FA53	CODE	
C_RBCD	L NEAR	F96C	CODE	
C_RBLP	L NEAR	F972	CODE	
C_RBRET.	L NEAR	F97D	CODE	
C_READ	N PROC	F90E	CODE	Length =0071
C_RHEX	N PROC	F97F	CODE	Length =000A
C_RMO.	L NEAR	F934	CODE	
C_RMLP	L NEAR	F92C	CODE	
C_WERR	L NEAR	FA2F	CODE	
C_WHEX	N PROC	FA30	CODE	Length =0011
C_WMLP	L NEAR	FA01	CODE	
C_WRITE.	N PROC	F99A	CODE	Length =0096
C_WYLP	L NEAR	F9D7	CODE	
DATA_SEG	Number	0040		
DCOLON	N PROC	E56C	CODE	Length =000C
DCRLF.	N PROC	E55F	CODE	Length =000D
DELETE_KEY	Number	0053		
DHEXBYTE	N PROC	E589	CODE	Length =000D
DHEXLONG	N PROC	E578	CODE	Length =000A
DHEXNIB.	N PROC	E596	CODE	Length =0015
DHEXWORD	N PROC	E582	CODE	Length =0007
DIAGNOSTICS_1.	N PROC	DAD3	CODE	Length =054D
DISKETTE_IO1	L NEAR	EC99	CODE	
DISKETTE_STATUS.	L BYTE	0041	DATA	
DISKSTATE.	L BYTE	0090	DATA	
DISK_STATUS.	L BYTE	0074	DATA	
DISP_PASS.	L NEAR	DCCA	CODE	
DIS_DMACC.	L NEAR	E3AB	CODE	
DLX_KB	Number	0001		
DMACCEL.	Number	0004		
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		

ROM BIOS Listing

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	E5AB	CODE	Length =0008
DNUMW	N PROC	E5B3	CODE	Length =0031
DNUMW_LOOP	L NEAR	E5BD	CODE	
DNUMW_SKIP	L NEAR	E5D3	CODE	
DNUMW_SPACES	L NEAR	E5CB	CODE	
DOUBLE	Number	0020		
DRB	L NEAR	FEEA	CODE	
DREAD	L WORD	00D0	DATA	
DROMSTRING	N PROC	E540	CODE	Length =0008
DSTRING	N PROC	E548	CODE	Length =0017
DS_LP	L NEAR	E54F	CODE	
DS_RET	L NEAR	E55A	CODE	
DUMMY IRET	N PROC	FF4B	CODE	Length =0009
DWRITE	L WORD	00CE	DATA	
E12M12D	Number	0015		
E48M12D	Number	0074		
E48M48D	Number	0093		
ENABLE_PARITY	N PROC	E5E4	CODE	Length =0054
ENDOFRAM	L NEAR	DE9E	CODE	
ESTAB	Number	0010		
F5_TMP	L NEAR	E95B	CODE	
FAIL_M	L BYTE	D9CC	CODE	
FAR_CALLS	F PROC	C004	CODE	Length =005C
FDC_ERROR	Number	0020		
FDU_DATA1	N PROC	EDEF	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 =00A1
FD_PARMS	L BYTE	EFC7	CODE	
FLAGS_DATA1	N PROC	C000	CODE	Length =0004
FONTHI8	N PROC	C860	CODE	Length =0358
FONTLO16	N PROC	C060	CODE	Length =0800

FONTLO8.	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	
F00.	L NEAR	DDCC	CODE	
F_BUFOFF	Text	[bp+4]		
F_CHECK_VALID.	N PROC	EF6B	CODE	Length =0048
F_CHGNLN	N PROC	EFB3	CODE	Length =0008
F_CLRHEAD.	L NEAR	E6BE	CODE	
F_COMMAND.	Text	[bp+3]		
F_CONT	L NEAR	EF2B	CODE	
F_CS_OUT	L NEAR	F685	CODE	
F_CV_DOUBLE.	L NEAR	EFA9	CODE	
F_CV_DR.	L NEAR	F600	CODE	
F_CV_HI.	L NEAR	EF94	CODE	
F_CV_PRE2.	L NEAR	EF96	CODE	
F_CV_RET	L NEAR	EFAD	CODE	
F_CV_TST2.	L NEAR	EF9A	CODE	
F_CYL.	Text	[bp+7]		
F_DOFMT.	L NEAR	E92C	CODE	
F_DORATE	L NEAR	F6C0	CODE	
F_DRIVE.	Text	[bp+0]		
F_DRVSWITCH.	N PROC	F5F6	CODE	Length =000D
F_DSKERR	L NEAR	E692	CODE	
F_DTYPE.	N PROC	F5EB	CODE	Length =000B
F_FMTDONE.	L NEAR	E937	CODE	
F_FORMAT_CMD	Number	004D		
F_GB_DECODE.	L NEAR	EE3B	CODE	
F_GB_JMP	L NEAR	EE43	CODE	
F_GB_LOOP.	L NEAR	EE03	CODE	
F_GB_LOOP1	L NEAR	EE34	CODE	
F_GB_OUT	L NEAR	EE1A	CODE	
F_GB_RET	L NEAR	EE47	CODE	
F_GB_TABLE	L BYTE	EDEF	CODE	
F_GETDRV	N PROC	F603	CODE	Length =0009
F_GET_BYTE	N PROC	EDF8	CODE	Length =0051
F_GET_VAR.	N PROC	F62E	CODE	Length =000D
F_GVDONE	L NEAR	F63A	CODE	
F_GVOK	L NEAR	F62E	CODE	
F_HEAD	Text	[bp+1]		
F_HEAD_SETTLE.	L NEAR	EF46	CODE	
F_HLT.	Number	0001		
F_HUT.	Number	000F		
F_IO1.	L NEAR	EC92	CODE	
F_IO_EXIT.	L NEAR	ECED	CODE	
F_IO_QUIT.	L NEAR	ECC9	CODE	
F_IO_RET	L NEAR	ECC2	CODE	
F_JMPNR.	L NEAR	E68F	CODE	
F_MK1212	L NEAR	E6AE	CODE	
F_MOTOR_ON	N PROC	F5C6	CODE	Length =0025
F_MOTOR_PORT	Number	03F2		
F_MOTOR_WAIT	Number	0025		
F_MO_RET	L NEAR	F5EA	CODE	
F_NDMA	Number	0000		
F_NEC_DATA	Number	03F5		

ROM BIOS Listing

F_NEC_RDY.	N PROC	F63B	CODE	Length =0017
F_NEC_RESET.	N PROC	F5AB	CODE	Length =001B
F_NEC_RESET_RET.	L NEAR	F5C5	CODE	
F_NEC_STATUS.	Number	03F4		
F_NOLOFMT.	L NEAR	E917	CODE	
F_NOMDFMT.	L NEAR	E923	CODE	
F_NOPASS.	L NEAR	E692	CODE	
F_NORETRY.	L NEAR	E6E1	CODE	
F_NOTSTAT.	L NEAR	F618	CODE	
F_NR1.	L NEAR	F63E	CODE	
F_NR_RET.	L NEAR	F650	CODE	
F_NUMSECS.	Text	[bp+2]		
F_NURATE.	L NEAR	E954	CODE	
F_NUSTATE.	N PROC	F60C	CODE	Length =0022
F_NU_CONT.	L NEAR	F62A	CODE	
F_OPSPFMT.	L NEAR	E932	CODE	
F_PASS2.	L NEAR	E6DD	CODE	
F_PB_ERRET.	L NEAR	F6D8	CODE	
F_PB_RET.	L NEAR	F6D7	CODE	
F_PUT_BYTE.	N PROC	F6C5	CODE	Length =001B
F_R1.	L NEAR	ED13	CODE	
F_R2.	L NEAR	ED1B	CODE	
F_RATEDONE.	L NEAR	E97F	CODE	
F_RD1.	L NEAR	ED69	CODE	
F_RDATA.	N PROC	ED57	CODE	Length =0020
F_RD_LOOP.	L NEAR	ED64	CODE	
F_READ_CMD.	Number	00E6		
F_REAL_DRIVE.	Text	[bp+8]		
F_RECAL_CMD.	Number	0007		
F_RESET.	N PROC	ECFA	CODE	Length =0056
F_RETRY.	L NEAR	EC9D	CODE	
F_RETSTAT.	L NEAR	ECE3	CODE	
F_RW1.	L NEAR	EDA3	CODE	
F_RW2.	L NEAR	EDCD	CODE	
F_RW3.	L NEAR	EDE4	CODE	
F_RW_COMMON.	N PROC	ED77	CODE	Length =0078
F_RW_RET.	L NEAR	EDEC	CODE	
F_RW_SKIP.	L NEAR	EDC7	CODE	
F_S1.	L NEAR	EEEC	CODE	
F_S2.	L NEAR	EEE7	CODE	
F_SD1.	L NEAR	EE66	CODE	
F_SD2.	L NEAR	EE95	CODE	
F_SD_RET.	L NEAR	EEA9	CODE	
F_SECNUM.	Text	[bp+6]		
F_SEEK.	N PROC	EEAA	CODE	Length =00A2
F_SEEK_CMD.	Number	000F		
F_SETFF.	N PROC	F6BA	CODE	Length =000B
F_SETFRMT.	N PROC	E905	CODE	Length =0035
F_SETRATE.	N PROC	E93A	CODE	Length =0046
F_SET_DMA.	N PROC	EE49	CODE	Length =0061
F_SIS.	N PROC	F686	CODE	Length =000B
F_SNSDRV_CMD.	Number	0004		
F_SNSINT_CMD.	Number	0008		
F_SPECIFY_CMD.	Number	0003		
F_SPEEDOK.	L NEAR	F684	CODE	

F_SRT_48	Number	000C		
F_SRT_96	Number	000E		
F_S_RECAL	L NEAR	EEBE	CODE	
F_S_RET	L NEAR	EF4B	CODE	
F_TABLE	L WORD	ECB0	CODE	
F_TRCONT	L NEAR	E6B1	CODE	
F_TRCONTO	L NEAR	E6C2	CODE	
F_TRDONE	L NEAR	E6E2	CODE	
F_TSTRETRY	N PROC	E665	CODE	Length =007F
F_WAIT_FOR_NEC	N PROC	F652	CODE	Length =0020
F_WAIT_ONE_MS	N PROC	EF4C	CODE	Length =0008
F_WD1	L NEAR	F6B0	CODE	
F_WDATA	N PROC	F691	CODE	Length =0029
F_WD_LOOP	L NEAR	F6AB	CODE	
F_WRITE_CMD	Number	00C5		
G4TOD	Number	0001		
GAME_CARD	Number	0201		
GDT	L WORD	00AA	DATA	Length =000C
GDTALIAS	L WORD	00C2	DATA	Length =0004
GDT_ENT	L WORD	E080	CODE	
GEN3	Number	0000		
GO_ON1	L NEAR	DD3E	CODE	
GO_ON2	L NEAR	DD4D	CODE	
GO_ON3	L NEAR	DE19	CODE	
GO_ON4	L NEAR	DE28	CODE	
GRF_GRAPHICS_DOWN	N PROC	D689	CODE	Length =0063
GRF_GRAPHICS_READ	N PROC	D6EC	CODE	Length =00EF
GRF_GRAPHICS_UP	N PROC	D5F4	CODE	Length =0055
GRF_GRAPHICS_WRITE	N PROC	D7DB	CODE	Length =0106
GRF_LIGHT_PEN	N PROC	F5A8	CODE	Length =0003
GRF_READ_DOT	N PROC	D550	CODE	Length =001A
GRF_WRITE_DOT	N PROC	D56A	CODE	Length =002B
G_72	L NEAR	D8F1	CODE	
G_8X16_2	L NEAR	D7BF	CODE	
G_8X8_2	L NEAR	D7FE	CODE	
G_ADDR	N PROC	D595	CODE	Length =005F
G_ADDR_TEST	L NEAR	D806	CODE	
G_ALIGN_DOT	L NEAR	D588	CODE	
G_BITMASK	L NEAR	D5E2	CODE	
G_CHAR_LP	L NEAR	D896	CODE	
G_CMP_MOD	L NEAR	D6AF	CODE	
G_COLOR_TABLE	L BYTE	D8DD	CODE	
G_CURS_OFF	N PROC	D8E1	CODE	Length =001E
G_DETMODE	L NEAR	D829	CODE	
G_EXP_BYT	L NEAR	D8A8	CODE	
G_FILLER	N PROC	D66B	CODE	Length =001E
G_F_CONT	L NEAR	D797	CODE	
G_F_EXIT	L NEAR	D7D5	CODE	
G_F_I_LP	L NEAR	D66D	CODE	
G_F_MACH	L NEAR	D79C	CODE	
G_F_S_LP	L NEAR	D66F	CODE	
G_HI_WR	L NEAR	D84D	CODE	
G_IA_LP	L NEAR	D8A1	CODE	
G_I_A_LP	L NEAR	D85D	CODE	
G_JSFY_DOT	L NEAR	D565	CODE	

ROM BIOS Listing

G_LDS_R	L NEAR D70C	CODE	
G_LDS_W	L NEAR D821	CODE	
G_LINELP	L NEAR D859	CODE	
G_MATCHB	L NEAR D791	CODE	
G_MEDGET	L NEAR D75E	CODE	
G_MED_BIT	L NEAR D777	CODE	
G_MED_IA	L NEAR D763	CODE	
G_MED_STORE	L NEAR D8BD	CODE	
G_MED_WR	L NEAR D885	CODE	
G_M_AREA	L NEAR D64C	CODE	
G_RDLOOP	L NEAR D72C	CODE	
G_RD_IA	L NEAR D730	CODE	
G_RD_MED	L NEAR D75C	CODE	
G_REPCHAR	L NEAR D852	CODE	
G_RETURN	L NEAR D8D9	CODE	
G_SCAN_LP	L NEAR D89C	CODE	
G_SCROLLER	N PROC D649	CODE	Length =0022
G_SELFONT	L NEAR D813	CODE	
G_SETDOWN	L NEAR D6BD	CODE	
G_SET_UP	L NEAR D629	CODE	
G_SKP_1	L NEAR D5A6	CODE	
G_SKP_2	L NEAR D5AC	CODE	
G_SKP_3	L NEAR D5B2	CODE	
G_SKP_4	L NEAR D5D5	CODE	
G_SKP_5	L NEAR D5F3	CODE	
G_SUPER_WR	L NEAR D849	CODE	
G_TEST_ADDR	L NEAR D7CB	CODE	
G_TINYTEXT	L NEAR D845	CODE	
G_TST_MOD	L NEAR D61A	CODE	
G_T_XOR	L NEAR D865	CODE	
G_UNREVERSE_VIDEO_LOOP	L NEAR D754	CODE	
G_W_BYTE	L NEAR D86C	CODE	
G_XORBIT	L NEAR D57F	CODE	
HD_ERROR	L BYTE 0042	DATA	
HF_NUM	L BYTE 0075	DATA	
HIRATE	Number 0000		
HISTORY	L NEAR DEBA	CODE	
HOLDON	L NEAR DDD1	CODE	
I13_IH	V WORD 0000	CODE	External
ILL_FLG	L NEAR FF41	CODE	
ILL_INT	N PROC FF23	CODE	Length =0028
ILL_LN	L NEAR E509	CODE	
ILL_LP	L NEAR E50E	CODE	
ILL_M1	L BYTE D999	CODE	
ILL_M2	L BYTE D9B2	CODE	
ILL_M3	L BYTE D9B8	CODE	
ILL_SW	L NEAR FF3E	CODE	
ILL_TEND	L NEAR E505	CODE	
ILL_TRAP	N PROC E4BE	CODE	Length =005C
INNERLOOP	L NEAR DD9E	CODE	
INSERT_KEY	Number 0052		
INSERT_MODE	Number 0080		
INSERT_SHIFT	Number 0080		
INT00LOCN	L DWORD 0000	ABSO	
INT01LOCN	L DWORD 0004	ABSO	

INT02LOCN.	L DWORD 0008	ABSO
INT03LOCN.	L DWORD 000C	ABSO
INT04LOCN.	L DWORD 0010	ABSO
INT05LOCN.	L DWORD 0014	ABSO
INT06LOCN.	L DWORD 0018	ABSO
INT07LOCN.	L DWORD 001C	ABSO
INT08LOCN.	L DWORD 0020	ABSO
INT09LOCN.	L DWORD 0024	ABSO
INT0ALOCN.	L DWORD 0028	ABSO
INT0BLOCN.	L DWORD 002C	ABSO
INT0CLOCN.	L DWORD 0030	ABSO
INT0DLOCN.	L DWORD 0034	ABSO
INT0ELOCN.	L DWORD 0038	ABSO
INT0FLOCN.	L DWORD 003C	ABSO
INT10LOCN.	L DWORD 0040	ABSO
INT11LOCN.	L DWORD 0044	ABSO
INT12LOCN.	L DWORD 0048	ABSO
INT13LOCN.	L DWORD 004C	ABSO
INT14LOCN.	L DWORD 0050	ABSO
INT15LOCN.	L DWORD 0054	ABSO
INT16LOCN.	L DWORD 0058	ABSO
INT17LOCN.	L DWORD 005C	ABSO
INT18LOCN.	L DWORD 0060	ABSO
INT19LOCN.	L DWORD 0064	ABSO
INT1ALOCN.	L DWORD 0068	ABSO
INT1BLOCN.	L DWORD 006C	ABSO
INT1CLOCN.	L DWORD 0070	ABSO
INT1DLOCN.	L DWORD 0074	ABSO
INT1ELOCN.	L DWORD 0078	ABSO
INT1FLOCN.	L DWORD 007C	ABSO
INTR_FLAG.	L BYTE 006B	DATA
IO_ROM_INIT.	L WORD 0067	DATA
IO_ROM_SEG.	L WORD 0069	DATA
I_CAL.	L NEAR DF35	CODE
I_CALR_M.	L BYTE DA2E	CODE
I_CAL_O.	L NEAR DF8A	CODE
I_CAL_1_1_80.	L NEAR DF60	CODE
I_CAL_END.	L NEAR DFB2	CODE
I_CAL_ERR.	L NEAR DF93	CODE
I_CAL_MAX.	L NEAR DF77	CODE
I_CAL_OK.	L NEAR DFA5	CODE
I_CAL_VAL.	L BYTE DACA	CODE
I_COM_M.	L BYTE DA72	CODE
I_CPU.	L NEAR DAF8	CODE
I_CPU_ERR.	L NEAR DB23	CODE
I_CPU_M.	L BYTE D9D3	CODE
I_CPU_OK.	L NEAR DB34	CODE
I_DMAC.	L NEAR DB88	CODE
I_DMAC_ERR.	L NEAR DC07	CODE
I_DMAC_LP.	L NEAR DB98	CODE
I_DMAC_M.	L BYTE D9FA	CODE
I_DMAC_NIB.	L NEAR DBE7	CODE
I_DMAC_OK.	L NEAR DC14	CODE
I_DMAC_PASS2.	L NEAR DB95	CODE
I_DMAC_RET.	L NEAR DBFA	CODE

ROM BIOS Listing

I_DMAT	L NEAR DB6D	CODE	
I_DMAT_ERR	L NEAR DB7B	CODE	
I_DMAT_M	L BYTE D9ED	CODE	
I_DMAT_OK	L NEAR DB81	CODE	
I_DMAT_RET	L NEAR DB79	CODE	
I_D_80X25	L NEAR EOF8	CODE	
I_D_E	L NEAR DC11	CODE	
I_D_INIT	N PROC E0A0	CODE	Length =00C4
I_D_M	L BYTE DA14	CODE	
I_D_MODE	L NEAR E15E	CODE	
I_D_OK	L NEAR E0FE	CODE	
I_D_SW	L NEAR E107	CODE	
I_FATAL	N PROC F720	CODE	Length =0061
I_FATAL_RET	L NEAR F772	CODE	
I_FDU_M	L BYTE DA96	CODE	
I_FDUB_M	L BYTE DAA3	CODE	
I_FDU_END	L NEAR E4A4	CODE	
I_FDU_LP	L NEAR E47E	CODE	
I_FDU_NOT_M	L BYTE DAB0	CODE	
I_FDU_OK	L NEAR E4A1	CODE	
I_FDU_RDY_M	L BYTE DAB4	CODE	
I_GDT	N PROC E05E	CODE	Length =0042
I_GDT0	L NEAR E078	CODE	
I_HARD_RESET	N PROC E05B	CODE	Length =0003
I_HDU_M	L BYTE DABD	CODE	
I_HDU_OK	L NEAR E405	CODE	
I_INIT_END	L NEAR E4A8	CODE	
I_KB_M	L BYTE DA54	CODE	
I_KB_ST_M	L BYTE DA61	CODE	
I_NO_COM_A	L NEAR E36C	CODE	
I_NO_GAME_CARD	L NEAR E386	CODE	
I_NO_SCCS	L NEAR E37B	CODE	
I_NPU_M	L BYTE DA21	CODE	
I_OPTROM_M	L BYTE DA89	CODE	
I_OUT_MASK	N PROC E1B7	CODE	Length =0009
I_PIC	L NEAR DC1B	CODE	
I_PIC_0_OK	L NEAR DC59	CODE	
I_PIC_1_OK	L NEAR DC5D	CODE	
I_PIC_2_OK	L NEAR DC61	CODE	
I_PIC_3_OK	L NEAR DC65	CODE	
I_PIC_4_OK	L NEAR DC69	CODE	
I_PIC_END	L NEAR DCC1	CODE	
I_PIC_ERR	L NEAR DC8C	CODE	
I_PIC_HARD	L NEAR DC4E	CODE	
I_PIC_HOT	L NEAR DC87	CODE	
I_PIC_INIT	N PROC E1A6	CODE	Length =0011
I_PIC_M	L BYTE DA07	CODE	
I_PIC_NO_HOT	L NEAR DCB5	CODE	
I_PIC_OK	L NEAR DCBB	CODE	
I_PIC_SOFT	L NEAR DC3E	CODE	
I_PIC_TEST	L NEAR DC6B	CODE	
I_PRT_EXIT	L NEAR E35A	CODE	
I_PRT_LOOP	L NEAR E341	CODE	
I_PRT_M	L BYTE DA65	CODE	
I_PWRUP	L NEAR DAE4	CODE	

I_RAM_M	L BYTE	DA7F	CODE	
I_ROM	L NEAR	DB48	CODE	
I_ROM_ERR	L NEAR	DB60	CODE	
I_ROM_M	L BYTE	D9E0	CODE	
I_ROM_OK	L NEAR	DB66	CODE	
I_ROM_RET1	L NEAR	DB4E	CODE	
I_ROM_RET2	L NEAR	DB56	CODE	
I_ROM_RET3	L NEAR	DB5E	CODE	
I_RTC	L NEAR	DFB2	CODE	
I_RTC_END	L NEAR	E018	CODE	
I_RTC_ERR	L NEAR	DFF0	CODE	
I_RTC_HI_M	L BYTE	DA4C	CODE	
I_RTC_LO_M	L BYTE	DA48	CODE	
I_RTC_M	L BYTE	DA3B	CODE	
I_RTC_NR_M	L BYTE	DA50	CODE	
I_RTC_OK	L NEAR	E00C	CODE	
I_VEC0	L NEAR	E174	CODE	
I_VEC8	L NEAR	E196	CODE	
I_VECTOR	N PROC	E164	CODE	Length =0042
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	00C8		
KBRSH	Number	00C7		
KBSCR	Number	00C3		
KB_BUFFER	L WORD	001E	DATA	Length =0010
KB_CAP_FLAGS	L BYTE	CBB8	CODE	
KB_CMD_SEND	N PROC	E4B3	CODE	Length =000B
KB_CMD_WLUP	L NEAR	E4B3	CODE	
KB_DATA1	N PROC	CBB8	CODE	Length =033F
KB_DATA_TABLE	L BYTE	CBBF	CODE	
KB_FLAG	L BYTE	0017	DATA	
KB_FLAG_1	L BYTE	0018	DATA	
KB_FLUSH	L NEAR	E2CD	CODE	
KB_FLUSH_BACK	L NEAR	E2DC	CODE	
KB_NOT_DLX	L NEAR	E320	CODE	
KB_STATUS	Number	0064		
KB_TYPE_READ	L NEAR	E310	CODE	
KB_TYPE_WAIT	L NEAR	E305	CODE	
KDBL0	Number	00D8		
KDECO	Number	00D7		
KDEC1	Number	00D6		
KDEC2	Number	00D5		
KDEC3	Number	00D4		
KDEC4	Number	00D3		
KDEC5	Number	00D2		
KDEC6	Number	00D1		
KDEC7	Number	00D0		

ROM BIOS Listing

KDEC8.	Number	00CF	
KDEC9.	Number	00CE	
KNONE.	Number	00CD	
K_00	L NEAR	EB3C	CODE
K_2RES	L NEAR	EB1F	CODE
K_2RET	L NEAR	EB11	CODE
K_2TOG	L NEAR	EB16	CODE
K_4RES	L NEAR	EAEE	CODE
K_4RET	L NEAR	EAED	CODE
K_4TOG	L NEAR	EADB	CODE
K_ADV_END.	L NEAR	E87A	CODE
K_ADV_PTR.	N PROC	E86E	CODE Length =000D
K_ALT.	L NEAR	EAF5	CODE
K_ALTO	L NEAR	EB2F	CODE
K_ALT1	L NEAR	EB2E	CODE
K_ALT2	L NEAR	EB2D	CODE
K_ALT3	L NEAR	EB2C	CODE
K_ALT4	L NEAR	EB2B	CODE
K_ALT5	L NEAR	EB2A	CODE
K_ALT6	L NEAR	EB29	CODE
K_ALT7	L NEAR	EB28	CODE
K_ALT8	L NEAR	EB27	CODE
K_ALT9	L NEAR	EB26	CODE
K_BEEP	N PROC	EBC9	CODE Length =0023
K_BIT.	N PROC	EB8C	CODE Length =001F
K_BRK.	L NEAR	EB49	CODE
K_BUF.	L NEAR	EA5B	CODE
K_CAP.	L NEAR	EABC	CODE
K_CASE	L WORD	EBEC	CODE
K_CTL.	L NEAR	EB07	CODE
K_DATA1.	N PROC	EBEC	CODE Length =0032
K_EOI.	N PROC	EBAB	CODE Length =0008
K_HOLD	L NEAR	EA98	CODE
K_INS.	L NEAR	EAAE	CODE
K_INT.	N PROC	E987	CODE Length =01D9
K_IO	N PROC	E82E	CODE Length =0017
K_IX	L NEAR	EA07	CODE
K_JMP.	L NEAR	EA32	CODE
K_LED_CAP.	L NEAR	EB6A	CODE
K_LED_CMD.	L NEAR	EB77	CODE
K_LED_DAT.	L NEAR	EB81	CODE
K_LED_NUM.	N PROC	EB60	CODE Length =002C
K_LED_RET.	L NEAR	EB8B	CODE
K_LOCK	L NEAR	E9F9	CODE
K_LOOK	F PROC	E860	CODE Length =0009
K_LP	L NEAR	EBD1	CODE
K_LSH.	L NEAR	EB0B	CODE
K_NON1	L NEAR	EAC6	CODE
K_NONE	L NEAR	EA60	CODE
K_NOP.	L NEAR	EA63	CODE
K_NOP1	L NEAR	EB46	CODE
K_NO_CAP	L NEAR	E9EC	CODE
K_NO_CASE.	L NEAR	EA3B	CODE
K_NO_HOLD.	L NEAR	EA51	CODE
K_NO_LOCK.	L NEAR	EA01	CODE

K_NO_XCODE	L NEAR EA59	CODE	
K_NUL	L NEAR EAA9	CODE	
K_NUM	L NEAR EAC8	CODE	
K_OK	L NEAR E9B2	CODE	
K_PAUSE	L NEAR EA7D	CODE	
K_PRT	L NEAR EAA1	CODE	
K_READ	N PROC E845	CODE	Length =001B
K_RES	L NEAR EA6C	CODE	
K_RET	L NEAR E842	CODE	
K_RSH	L NEAR EBOF	CODE	
K_SCR	L NEAR EAD4	CODE	
K_SEE	L NEAR E854	CODE	
K_STAT	N PROC E869	CODE	Length =0005
K_TRY	N PROC EBB3	CODE	Length =0016
K_XLAT	L NEAR EA1B	CODE	
LABEL1	L NEAR F9A0	CODE	
LABEL2	L NEAR F9A6	CODE	
LABEL3	L NEAR F9AD	CODE	
LAstrate	L BYTE 008E	DATA	
LEAP1	L NEAR F9F4	CODE	
LEFT_SHIFT	Number 0002		
LEFT_SHIFT_KEY	Number 002A		
LF	Number 000A		
LORATE	Number 0080		
MASTAB	L WORD E297	CODE	
MASTER_TBL_PTR	L DWORD 0084	DATA	
MEDIA_CHANGE	Number 0006		
MEDRATE	Number 0040		
MEMORY_SIZE	L WORD 0013	DATA	
MEMTST	N PROC E22F	CODE	Length =0047
MEMTST_ERR	L NEAR E274	CODE	
MEMTST_ERR_C	L NEAR E270	CODE	
MEMTST_R1	L NEAR E242	CODE	
MEMTST_R2	L NEAR E259	CODE	
MEMTST_W1	L NEAR E236	CODE	
MEMTST_W2	L NEAR E24F	CODE	
MFG_ERR_FLAG	L BYTE 0015	DATA	Length =0002
MFG_TST	L BYTE 0012	DATA	
MOR_MEM	L NEAR DEF3	CODE	
MOTOR_COUNT	L BYTE 0040	DATA	
MOTOR_STATUS	L BYTE 003F	DATA	
MOVAXCS	L NEAR E650	CODE	
MT_END	L WORD E2AD	CODE	
M_CASS	F PROC F859	CODE	Length =0005
M_EQUIP	N PROC F84D	CODE	Length =000C
M_SIZE	N PROC F841	CODE	Length =000C
NEC_STATUS	L BYTE 0042	DATA	Length =0007
NEWFLOP	Number 0002		
NIBOK	L NEAR E5A4	CODE	
NMI_ENABLE	Number 0080		
NMI_ENABLE_PORT	Number 00A0		
NOPRINT	L NEAR DE09	CODE	
NOTBOUND	L NEAR FF6C	CODE	
NO_DBL_STEP	L NEAR EF0D	CODE	
NO_PMEM	L NEAR DEFB	CODE	

ROM BIOS Listing

NO_THING	L WORD	0096	DATA	Length =0006
NUL.	Number	0000		
NUM_LOCK_KEY	Number	0045		
NUM_LOCK_MODE.	Number	0020		
NUM_LOCK_SHIFT	Number	0020		
N_1.	L NEAR	F870	CODE	
N_INT.	N PROC	F85F	CODE	Length =0017
N_OUT.	L NEAR	F874	CODE	
OFF_FAIL	L WORD	00CC	DATA	
OPT_ROM_M.	L BYTE	E276	CODE	
OP_INT	N PROC	E638	CODE	Length =002D
OSMERGE1	L DWORD	00A2	DATA	
OSMERGE2	L DWORD	00A6	DATA	
PO_DATA1	N PROC	E276	CODE	Length =0037
P1_DATA1	N PROC	D8FF	CODE	Length =01D4
P4_DATA1	N PROC	E537	CODE	Length =0003
PARA_GRAPH	Number	B800		
PARA_MONO.	Number	B000		
PARITY	Number	0000		
PARITY1_M.	L BYTE	E5FB	CODE	
PARITY2_M.	L BYTE	E618	CODE	
PASS_M	L BYTE	D9BB	CODE	
PAUSE.	Number	00CA		
PAUSE_MODE	Number	0008		
PCINIT	N PROC	E2AD	CODE	Length =0206
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_NE01	Number	0020		
PIC_OFF_MSK.	Number	00FF		
PIC_SE01_0	Number	0060		
PIC_SE01_1	Number	0061		
PIC_SE01_6	Number	0066		
PMEMCNT.	L NEAR	DD7F	CODE	
PMENTST_R1	L NEAR	DE63	CODE	
PMENTST_R2	L NEAR	DE81	CODE	
PMENTST_W1	L NEAR	DE59	CODE	
PMENTST_W2	L NEAR	DE75	CODE	
PORT_OFF	L BYTE	0077	DATA	
PPASS.	L NEAR	DED8	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		
PTSTADDR.	L NEAR	DE3C	CODE	
PTSTERR.	L NEAR	DEA3	CODE	
PWRUPL	Number	0020		
P_8253_0	Number	0040		
P_8253_1	Number	0041		
P_8253_2	Number	0042		
P_8253_CTRL.	Number	0043		

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_STAT	L NEAR	F034	CODE	
P_TBL.	L WORD	E287	CODE	
P_TRAPCE	Number	3F60		
RAM_ERROR.	L NEAR	DF05	CODE	
RAM_SIZE_END	L NEAR	DEC6	CODE	
RAM_SIZE_END_1	L NEAR	DEDB	CODE	
RAM_SIZE_LP.	L NEAR	DD24	CODE	
RAM_SIZE_NXT	L NEAR	DD65	CODE	
RAM_SIZE_TST	L NEAR	DD14	CODE	
RESET_FLAG	L WORD	0072	DATA	
RESVO.	L DWORD	0088	DATA	
RESV1.	L WORD	008C	DATA	
RIGHT_SHIFT.	Number	0001		
RIGHT_SHIFT_KEY.	Number	0036		
ROM_CHECKSUM	N PROC	E52A	CODE	Length =000D
ROM_CHECKSUM_CNT	L NEAR	E52D	CODE	
ROM_CHECKSUM_LOOP.	L NEAR	E52F	CODE	
ROM_CHKSUM_OK.	L NEAR	E440	CODE	
ROM_ERR.	N PROC	E51A	CODE	Length =0010
ROM_ID	L BYTE	C001	CODE	
ROM_MT	L WORD	C002	CODE	
ROM_SCAN_EXIT.	L NEAR	E459	CODE	
ROM_SCAN_LOOP.	L NEAR	E408	CODE	
ROM_SCAN_NEXT.	L NEAR	E455	CODE	
RPASS.	L NEAR	DED5	CODE	
RS232_ADDR	L WORD	0000	DATA	Length =0004
RS_DLY	N PROC	E8AC	CODE	Length =000A
RS_GBE	L NEAR	E900	CODE	
RS_INIT.	L NEAR	E3A1	CODE	
RS_LP.	L NEAR	E8B1	CODE	
RS_NOP	L NEAR	E77D	CODE	
RS_NORM.	L NEAR	E749	CODE	
RS_OK.	L NEAR	E771	CODE	
RS_PBE	L NEAR	E8DA	CODE	
RS_PB_GB	L NEAR	E8D9	CODE	
RS_RET	L NEAR	E77A	CODE	
RS_STAT.	N PROC	E87B	CODE	Length =000D
RS_TBL	L WORD	E77F	CODE	
RS_WS.	N PROC	E888	CODE	Length =0024
RS_WS_COM.	L NEAR	E897	CODE	
RS_WS_EXIT	L NEAR	E8A8	CODE	
RS_WS_LP	L NEAR	E88D	CODE	
RTC_CHK.	N PROC	E1C0	CODE	Length =006F
RTC_CHK_HIGH	L NEAR	E22E	CODE	
RTC_CHK_LOW.	L NEAR	E22E	CODE	
RTC_CHK_RESET_ERR.	L NEAR	E1E1	CODE	

ROM BIOS Listing

RTC_CHK_RESET_LP	L NEAR	E1CF	CODE	
RTC_CHK_RESET_OK	L NEAR	E1E4	CODE	
RTC_CHK_SET_ERR.	L NEAR	E201	CODE	
RTC_CHK_SET_LP	L NEAR	E1EE	CODE	
RTC_CHK_SET_OK	L NEAR	E204	CODE	
SAVE_RAM	L NEAR	DD2D	CODE	
SCC_CTL_A.	Number	0050		
SCC_CTL_B.	Number	0052		
SCC_FE	Number	0040		
SCC_OE	Number	0020		
SCC_PE	Number	0010		
SCC_RXD.	Number	0001		
SCC_TBL.	L WORD	E28F	CODE	
SCC_TXD.	Number	0004		
SCRL_LOCK_KEY.	Number	0046		
SCRL_LOCK_MODE	Number	0010		
SCRL_LOCK_SHIFT.	Number	0010		
SECT_NOT_FOUND	Number	0004		
SEEK_ERROR	Number	0040		
SEEK_STATUS.	L BYTE	003E	DATA	
SEG_FAIL	L WORD	00CA	DATA	
SERIAL_IO.	N PROC	E739	CODE	Length =004E
SERIAL_T_OUT	L BYTE	007C	DATA	Length =0004
SET_DS	N PROC	E53A	CODE	Length =0006
SET_DS_WORD.	L WORD	E538	CODE	
SPASS_M.	L BYTE	D9C4	CODE	
STACK_ROM.	L WORD	D900	CODE	
STACK_SEG.	Number	0030		
STOP	L NEAR	DE21	CODE	
STOP_DISK.	N PROC	ED50	CODE	Length =0007
SWITCH_BITS.	L WORD	0010	DATA	
SYS_CONF_A	Number	0066		
SYS_CONF_B	Number	0067		
S_EOL.	L NEAR	FFCD	CODE	
S_ERR.	L NEAR	FFBD	CODE	
S_GET.	L NEAR	FFE1	CODE	
S_INT.	N PROC	FF54	CODE	Length =0096
S_LP	L NEAR	FF99	CODE	
S_NOP.	L NEAR	FFCA	CODE	
S_OK	L NEAR	FFA2	CODE	
S_OUT.	L NEAR	FFD4	CODE	
T0CMD.	Number	0036		
T0COUNT.	Number	0000		
T1CMD.	Number	0074		
T1COUNT.	Number	0013		
T2CMD.	Number	00B6		
T2COUNT.	Number	0266		
TIME_OUT	Number	0080		
TOD.	Number	0001		
TRAP_MESS.	L BYTE	F6E8	CODE	
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	FEA5	CODE	Length =004A
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	
U12M12D	Number	0002		
U48M12D	Number	0061		
U48M48D	Number	0080		
VECTOR	N PROC	FFF0	CODE	Length =000F
VROM_CHKSUM_OK	L NEAR	E139	CODE	
VROM_ERR	L NEAR	E14E	CODE	
VROM_SCAN_EXIT	L NEAR	E152	CODE	
VROM_SCAN_LOOP	L NEAR	E111	CODE	
VROM_SCAN_NEXT	L NEAR	E14E	CODE	
V_0	L NEAR	F30D	CODE	
V_01	L NEAR	F31A	CODE	
V_02	L NEAR	F31B	CODE	
V_1	L NEAR	F322	CODE	
V_2	L NEAR	F32C	CODE	
V_21	L NEAR	F331	CODE	
V_22	L NEAR	F366	CODE	
V_3	L NEAR	F368	CODE	
V_31	L NEAR	F36A	CODE	
V_3X8	L BYTE	0065	DATA	
V_4	L NEAR	F37A	CODE	
V_6845	L NEAR	F262	CODE	
V_APAGE	L BYTE	0062	DATA	
V_BASE6845	L WORD	0063	DATA	
V_BELL	N PROC	F583	CODE	Length =0025
V_BS	L NEAR	F4F8	CODE	
V_C2	L NEAR	F308	CODE	
V_CLR	L NEAR	F2B8	CODE	
V_CLR_FAST	L NEAR	F2CF	CODE	
V_CLR_FIN	L NEAR	F2D9	CODE	
V_CLR_TOP	L NEAR	F3AC	CODE	
V_COL	N PROC	F44F	CODE	Length =002C
V_COLORPAL	L BYTE	0066	DATA	
V_COLOUR	L NEAR	F08E	CODE	
V_COLS	L NEAR	F305	CODE	
V_COL_0	L NEAR	F464	CODE	
V_COL_1	L NEAR	F46B	CODE	
V_CR	L NEAR	F500	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	F0A4	CODE	Length =0058
V_FPOS	N PROC	F542	CODE	Length =001E
V_FPOS_0	L NEAR	F554	CODE	
V_FPOS_LP	L NEAR	F54D	CODE	
V_HEIGHT	L WORD	004C	DATA	
V_IO	N PROC	F065	CODE	Length =003F

ROM BIOS Listing

V_KSCROLL1	Number	00E0	
V_KSCROLL2	Number	0162	
V_LF	L NEAR	F4B0	CODE
V_LROW	L NEAR	F4BB	CODE
V_MD_40.	L BYTE	F0A4	CODE
V_MD_80.	L BYTE	F0B4	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	F0E4	CODE
V_MD_MONO.	L BYTE	F0D4	CODE
V_MD_WID	L BYTE	F0EC	CODE
V_MODE	L BYTE	0049	DATA
V_MV	L NEAR	F295	CODE
V_MV2.	L NEAR	F2A2	CODE
V_MV_DN.	L NEAR	F3A5	CODE
V_MV_FAST.	L NEAR	F2A8	CODE
V_MV_FLP	L NEAR	F2AB	CODE
V_NOP.	L NEAR	F0A3	CODE
V_OUT_BYTE	N PROC	F273	CODE Length =000C
V_OVR_NOT_OK	L NEAR	F1C4	CODE
V_OVR_OK	L NEAR	F1C6	CODE
V_PAGE	N PROC	F22C	CODE Length =0047
V_PAGE_0	L NEAR	F23E	CODE
V_PARS.	L BYTE	F0A4	CODE
V_POINTER.	Number	03B4	
V_POSN	N PROC	F560	CODE Length =0011
V_RAC.	N PROC	F3B1	CODE Length =002A
V_RAC_INBLANK.	L NEAR	F3CD	CODE
V_RAC_INLINE	L NEAR	F3C7	CODE
V_ROWS	L NEAR	F304	CODE
V_R_CURS_POS	N PROC	F215	CODE Length =0017
V_SCRL_DN.	N PROC	F390	CODE Length =0021
V_SCRL_MODE_7.	L NEAR	F2E7	CODE
V_SCRL_MV_AND_CLR.	L NEAR	F541	CODE
V_SCRL_POS	N PROC	F513	CODE Length =002F
V_SCRL_TTY	L NEAR	F4C6	CODE
V_SCRL_TTY_GRAPHICS.	L NEAR	F4D5	CODE
V_SCRL_UP.	N PROC	F27F	CODE Length =0111
V_SCROLL_OR_CLEAR.	L NEAR	F2EC	CODE
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	F4C1	CODE
V_STAT	N PROC	F504	CODE Length =000F
V_SYNC	L NEAR	F385	CODE
V_SYNC2.	L NEAR	F38A	CODE
V_TBL.	L WORD	F045	CODE
V_TERMINAL	N PROC	F47B	CODE Length =0089

V_TERM_NOBELL	L NEAR	F482	CODE	
V_TERM_NOP	L NEAR	F4F0	CODE	
V_TERM_RET	L NEAR	F4EE	CODE	
V_TOP	L WORD	004E	DATA	
V_TXT_DN	L NEAR	F399	CODE	
V_TXT_MD	N PROC	F571	CODE	Length =0012
V_TXT_OK	L NEAR	F581	CODE	
V_TXT_RAC	L NEAR	F3B9	CODE	
V_TXT_UP	L NEAR	F287	CODE	
V_TXT_WAC	L NEAR	F3E3	CODE	
V_TXT_WC	L NEAR	F41C	CODE	
V_V	L NEAR	F338	CODE	
V_V2	L NEAR	F373	CODE	
V_WAC	N PROC	F3DB	CODE	Length =0039
V_WAC_END	L NEAR	F40F	CODE	
V_WAC_HI	L NEAR	F3FA	CODE	
V_WAC_LO	L NEAR	F400	CODE	
V_WC	N PROC	F414	CODE	Length =003B
V_WC_END	L NEAR	F44A	CODE	
V_WC_HI	L NEAR	F433	CODE	
V_WC_LO	L NEAR	F439	CODE	
V_WC_NEXT	L NEAR	F433	CODE	
V_WIDTH	L WORD	004A	DATA	
WASTE	L NEAR	F6D4	CODE	
WASTEG	L NEAR	EE15	CODE	
WEREBACK	L NEAR	DDD5	CODE	
WRITE_PROTECT	Number	0003		
WX2_FMT	L NEAR	0000	CODE	External
W_NEC	L NEAR	F657	CODE	
W_NEC_RET	L NEAR	F66B	CODE	
W_ONE	L NEAR	EF50	CODE	
YEARS	L NEAR	F913	CODE	

ROM BIOS Listing

Segments and Groups:

Name	Size	Align	Combine	Class
CODE	D54B	PARA	COMMON	'ROM'
INTVEC	7C00	AT	0000	
WDRAM	0078	AT	0040	

Symbols:

Name	Type	Value	Attr
A1_BP	Number	0008	
A1_BX	Number	000E	
A1_CX	Number	000C	
A1_DI	Number	0006	
A1_DS	Number	0002	
A1_DX	Number	000A	
A1_ES	Number	0000	
AGAIN0	L NEAR	B01C	CODE
AGAIN1	L NEAR	B034	CODE
AGAIN2	L NEAR	B04A	CODE
AGAIN3	L NEAR	B060	CODE
AGAIN4	L NEAR	B076	CODE
AGAIN5	L NEAR	B08C	CODE
AGAIN6	L NEAR	B0A2	CODE
AL_SI	Number	0004	
ANOTHER	L NEAR	B015	CODE
BAD	N PROC	B000	CODE Length =0831
BBB	L NEAR	B000	CODE
BC_BAD	L NEAR	D1B3	CODE
BC_BUFF_RD	Number	000E	
BC_BUFF_WR	Number	000F	
BC_CC	Number	0001	
BC_DASD	Number	0015	
BC_DIAG_CTLR	Number	0014	
BC_DIAG_DRV	Number	0013	
BC_DIAG_RAM	Number	0012	
BC_FBT	Number	0006	
BC_FD	Number	0007	
BC_FT	Number	0005	
BC_PAR_RD	Number	0008	
BC_PAR_SET	Number	0009	
BC_RD	Number	0002	
BC_RDL	Number	000A	
BC_RECAL	Number	0011	
BC_RESET	Number	0000	
BC_RESET_1	Number	0000	
BC_SEEK	Number	000C	
BC_TST_RDY	Number	0010	
BC_VR	Number	0004	
BC_V_W	Number	000E	
BC_WR	Number	0003	
BC_WRL	Number	000B	
BIOS_INSTALL	N PROC	CFB4	CODE Global Length =0139

BOGUS	L NEAR D17B	CODE	
BOOT_SUCC.	L NEAR D158	CODE	
BREC	L NEAR D15D	CODE	
BR_TBL	L WORD D22A	CODE	
BUFF_IO.	L NEAR D332	CODE	
BUSY	L NEAR D48F	CODE	
BUSY_WAIT.	L NEAR D484	CODE	
CCB_BLKs	Number 0004		
CCB_BYTE	L NEAR D496	CODE	
CCB_CMD.	Number 0000		
CCB_DRV_B.	Number 0020		
CCB_OPT.	Number 0005		
CCB_SEND	L NEAR D47A	CODE	
CCFD	Number 0007		
CCREC.	Number 0011		
CCRT	Number 0012		
CC_BUSY.	L NEAR D4BD	CODE	
CC_ER.	Number 0002		
CHK_1.	L BYTE A000	CODE	
CMD_DONE	L NEAR D1BA	CODE	
COMMAND_BR	N PROC D1E4	CODE	Length =033F
CONTINUE	L NEAR D3CF	CODE	
CTLR_INIT.	L NEAR D011	CODE	
CTLR_MISSING	L NEAR D288	CODE	
CTLR_MX.	Number 0004		
DC_BUFF_RD	Number 000E		
DC_BUFF_WR	Number 000F		
DC_DIAG_CTLR	Number 00E4		
DC_DIAG_DRV.	Number 00E3		
DC_DIAG_RAM.	Number 00E0		
DC_ECC_RD.	Number 000D		
DC_FBT	Number 0007		
DC_FD.	Number 0004		
DC_FT.	Number 0006		
DC_PAR_SET	Number 000C		
DC_RD.	Number 0008		
DC_RDL	Number 00E5		
DC_RECAL	Number 0001		
DC_SEEK.	Number 000B		
DC_STAT_RD	Number 0003		
DC_TBL	L BYTE D254	CODE	
DC_TST_RDY	Number 0000		
DC_VR.	Number 0005		
DC_WR.	Number 000A		
DC_WRL	Number 00E6		
DETT_BOOT.	L NEAR D10E	CODE	
DETT_BOOT_END.	L NEAR D12B	CODE	
DETT_BOOT_NXT.	L NEAR D124	CODE	
DIS_CHAR	L NEAR D0AC	CODE	
DMACC.	Number 0001		
DMALONG.	Number 0001		
DMANORM.	Number 0000		
DMA_64K.	L NEAR D396	CODE	
DMA_MASK_B_3	Number 0003		
DMA_MASK_B_S	Number 0004		

ROM BIOS Listing

DMA_MODE_RD	Number	000B		
DMA_MODE_WR	Number	0007		
DMA_NO	L NEAR	D399	CODE	
DMA_R_STATUS	Number	0008		
DMA_START	L NEAR	D349	CODE	
DMA_W_ADDR	Number	0006		
DMA_W_BYTE	Number	000C		
DMA_W_CLR	Number	000D		
DMA_W_CMD	Number	0008		
DMA_W_CNT	Number	0007		
DMA_W_MASK	Number	000F		
DMA_W_MASK_B	Number	000A		
DMA_W_MODE	Number	000B		
DMA_W_REQ	Number	0009		
DNWIN	Number	0080		
DRV	L NEAR	D032	CODE	
DRVN_OK	L NEAR	D1B7	CODE	
DRV_1	L NEAR	D4F2	CODE	
DRV_CTLR	Number	0002		
DRV_DIAG	L NEAR	D3C4	CODE	
DRV_FORMAT	L NEAR	D3CB	CODE	
DRV_RDY	L NEAR	D076	CODE	
DRV_TOTAL	Number	0008		
DS_BY_HEX	N PROC	B8C7	CODE	Length =000C
EC_ADDR_MARK	Number	0002		
EC_BAD_TRK	Number	000B		
EC_BC	Number	0001		
EC_CNTLRLR	Number	0020		
EC_DMA_64K	Number	0009		
EC_ECC_COR	Number	0011		
EC_ECC_UN	Number	0010		
EC_INIT	Number	0007		
EC_NO_ERR	Number	0000		
EC_RESET	Number	0005		
EC_SEC_NOT_FND	Number	0004		
EC_SEEK	Number	0040		
EC_STAT	Number	00FF		
EC_TIME	Number	0080		
EC_UNDEF	Number	00BB		
ERC_CORR	Number	0018		
ERR	L NEAR	B8AB	CODE	
ER_MASTER_TBL	L BYTE	D523	CODE	
FALL	L NEAR	D3E6	CODE	
FCDISB	Number	0002		
FCKBIN	Number	0001		
FCPRSTR	Number	0009		
FCTEND	Number	004C		
FILL	L BYTE	A001	CODE	Length =7FFF
HABS	L NEAR	D067	CODE	
HABSS	Number	000F		
HBAD	L NEAR	D051	CODE	
HBADS	Number	000D		
HD_QUIT	L NEAR	D1D9	CODE	
HGOOD	L NEAR	D05E	CODE	
HGOODS	Number	0009		

HMSG	L NEAR D0DC	CODE	
H_EXIT	L NEAR D0E4	CODE	
I13_BUFF_RD.	L NEAR D330	CODE	
I13_BUFF_WR.	L NEAR D336	CODE	
I13_CC	L NEAR D312	CODE	
I13_IH	F PROC D17D	CODE	Global Length =0067
I13_PAR_RD	L NEAR D2EB	CODE	
I13_PAR_WR	L NEAR D28B	CODE	
I13_RD	L NEAR D33A	CODE	
I13_RDL.	L NEAR D31B	CODE	
I13_RESET.	L NEAR D269	CODE	
I13_WR	L NEAR D317	CODE	
I13_WRL.	L NEAR D32C	CODE	
I19_BOOT_SYS	L NEAR D0ED	CODE	
ID_IH.	L NEAR D161	CODE	
INT_OCW1	Number 0001		
INT_OCW1_M0.	Number 0001		
INT_OCW1_M5.	Number 0020		
INT_OCW2_EOI	Number 0020		
INT_WAIT	L NEAR D3DD	CODE	
INT_W_OCW2	Number 0000		
IO_LONG.	L NEAR D31D	CODE	
IO_NORM.	L NEAR D33C	CODE	
IVDBC.	Number 0013		
IVFC	Number 0021		
IVN_BASIC.	Number 0018		
IVN_BC	Number 0013		
IVN_BC_DETTE	Number 0040		
IVN_DIS_CHAR	Number 0010		
IV_BC.	L DWORD 004C	INTVEC	
IV_BC_DETT	L DWORD 0100	INTVEC	
IV_BOOT.	L DWORD 0064	INTVEC	
IV_BOOT_BUF.	L FAR 7C00	INTVEC	
IV_INT	L DWORD 0034	INTVEC	
IV_P_TBL_DETT.	L DWORD 0078	INTVEC	
IV_P_TBL_WIN	L DWORD 0104	INTVEC	
KB_RESET	L NEAR D004	CODE	
MDRV_1	L NEAR D507	CODE	
MEC.	L BYTE B9C5	CODE	
MI	L BYTE B8E2	CODE	
MINT	L BYTE B99F	CODE	
MNOD	L BYTE B9E0	CODE	
MSELTBL.	L NEAR D509	CODE	
MSUC	L BYTE B9B1	CODE	
NOCHG.	L NEAR B853	CODE	
NO_RESET	L NEAR D191	CODE	
NXT_CTLR	L NEAR D04D	CODE	
NXT_DRV.	L NEAR D08D	CODE	
NZDRVS	L NEAR D0D0	CODE	
PAR_WR	L NEAR D29A	CODE	
PAR_WR_ERX	L NEAR D2DB	CODE	
P_DMA.	Number 0000		
P_DMACC.	Number 0063		
P_DMA_LATCH.	Number 0082		
P_INT.	Number 0020		

ROM BIOS Listing

P_TBL_DETT	L NEAR	CF29	CODE
P_TBL_WIN	L NEAR	CF34	CODE
P_WX2	Number	0320	
RAM_CC	L BYTE	0074	WDRAM
RAM_CCB	L BYTE	0042	WDRAM
RAM_DRV_CNT	L BYTE	0075	WDRAM
RAM_KB_RESET	L WORD	0072	WDRAM
RAM_OPT	L BYTE	0076	WDRAM
RAM_PO	L BYTE	0077	WDRAM
RAM_STAT	L BYTE	0042	WDRAM
RAM_TIME	L WORD	006C	WDRAM
REPEAT	L NEAR	B027	CODE
REQ_L	L NEAR	D4CC	CODE
REQ_SUCC	L NEAR	D4D8	CODE
RET_NEAR	L NEAR	D28A	CODE
RET_NEAR_1	L NEAR	D2DD	CODE
RET_NEAR_2	L NEAR	D48E	CODE
RET_NEAR_3	L NEAR	D398	CODE
RET_NEAR_4	L NEAR	D4C8	CODE
RET_NEAR_5	L NEAR	D476	CODE
RET_NO_ERR	L NEAR	D30F	CODE
RET_STC	L NEAR	D4C7	CODE
RET_TIME	L NEAR	D48B	CODE
RET_TIME_J	L NEAR	D3EE	CODE
RET_TIME_K	L NEAR	D3EB	CODE
RE_DLY	L NEAR	D270	CODE
RE_W	L NEAR	D274	CODE
ROW1	L NEAR	B02D	CODE
ROW2	L NEAR	B043	CODE
ROW3	L NEAR	B059	CODE
ROW4	L NEAR	B06F	CODE
ROW5	L NEAR	B085	CODE
ROW6	L NEAR	B09B	CODE
SEC_SIZE	Number	0200	
SEC_SIZE_NORM	L NEAR	D340	CODE
SEND_BYTE	L NEAR	D2DE	CODE
SEND_ERR	L NEAR	D2E8	CODE
STAT_ERR	L NEAR	D477	CODE
STAT_LOOP	L NEAR	D418	CODE
SUBTABLE	L NEAR	D4DA	CODE
SW_B	Number	0067	
T0L	Number	0009	
T0_TBL	L NEAR	D52B	CODE
T1L	Number	000A	
T1_TBL	L NEAR	D534	CODE
T2L	Number	0002	
T2_TBL	L NEAR	D53E	CODE
T3L	Number	0003	
T3_TBL	L NEAR	D540	CODE
TI_0_1	Number	0165	
TI_BC_RESET	Number	0584	
TI_FIN	Number	01BE	
TI_KB_RESET	Number	019A	
TRDY0	L NEAR	D024	CODE
TST_DRV_RDY	L NEAR	D03D	CODE

UNDEF	L NEAR	D474	CODE	
USABLE	L NEAR	D09C	CODE	
W2RAM	Number	0040		
WAIT_MORE	L NEAR	D3D8	CODE	
WASTE0	L NEAR	B01F	CODE	
WASTE1	L NEAR	B037	CODE	
WASTE2	L NEAR	B04D	CODE	
WASTE3	L NEAR	B063	CODE	
WASTE4	L NEAR	B079	CODE	
WASTE5	L NEAR	B08F	CODE	
WASTE6	L NEAR	B0A5	CODE	
WE_BAD	L NEAR	B0B1	CODE	
WINS_USABLE	L NEAR	D0BA	CODE	
WIN_BC	L NEAR	D188	CODE	
WIN_BOOT	L NEAR	D135	CODE	
WIN_BOOT_NXT	L NEAR	D152	CODE	
WIN_CONT	L NEAR	D1AC	CODE	
WST_CYL	Number	0000		
WST_DDTO	Number	000B		
WST_ER_BUR	Number	0007		
WST_FTO	Number	000A		
WST_HEADS	Number	0002		
WST_OPT	Number	0008		
WST_RE_WR	Number	0003		
WST_STO	Number	0009		
WST_WR_PRE	Number	0005		
WX2_CC	L NEAR	D4AC	CODE	
WX2_CONFIG	L NEAR	D514	CODE	
WX2_CONFIG_0	Number	000C		
WX2_CONFIG_1	Number	0003		
WX2_FMT	N PROC	B840	CODE	Global Length =0087
WX2_INT	L NEAR	D3F1	CODE	
WX2_L	Number	0004		
WX2_LRG_OFFSET	Number	000C		
WX2_MSK	L NEAR	D51E	CODE	
WX2_MSK_DMA	Number	0001		
WX2_MSK_INT	Number	0002		
WX2_REQ	L NEAR	D4C9	CODE	
WX2_RESET	L NEAR	D519	CODE	
WX2_R_CONFIG	Number	0002		
WX2_R_DATA	Number	0000		
WX2_R_STATUS	Number	0001		
WX2_STAT	L NEAR	D519	CODE	
WX2_STAT_BUSY	Number	0008		
WX2_STAT_CD	Number	0004		
WX2_STAT_DRQ	Number	0010		
WX2_STAT_INT	Number	0020		
WX2_STAT_IO	Number	0002		
WX2_STAT_REQ	Number	0001		
WX2_WAIT	L NEAR	D3A0	CODE	
WX2_W_DATA	Number	0000		
WX2_W_MSK	Number	0003		
WX2_W_RESET	Number	0001		
WX2_W_SELECT	Number	0002		
ZFMT	L NEAR	B880	CODE	

ROM BIOS Listing

ZHEX	N PROC	B8D3	CODE	Length =000F
ZNX.	L NEAR	B8BA	CODE	
ZNX2	L NEAR	B8BD	CODE	
ZTEND.	L NEAR	B8C1	CODE	

A

ACSII String Utilities—Display, 1-142
 Additional Keyboard Data Area, 1-14
 Addresses, 1-4
 Floppy Disk Port, 1-8
 PC 6300 PLUS, 1-4
 i8237A p_dma Controller Port, 1-4
 i8254 p_timer Port, 1-6
 i8259A Programmable Interrupt Controller Port, 1-5
 Alphabetic (Migratory), 1-36
 Alphabetic (Non-Migratory), 1-40
 Alternate Numeric Keypad, 1-172
 Assembly Modules, 1-3
 Asynchronous Communication Chip (INS8250), 1-9

B

Background Colors—Set, 1-217
 BEL Character, 1-224
 Bound Instruction—Check, 1-262
 Break Key Sequence, 1-173
 Byte Location—Video RAM, 1-80

C

Call Internal HDU Init Code, 1-135
 CapLk Bytes, 1-35
 Carry Flag, 1-174
 Cassette I/O, 1-237
 Change-Line Status, 1-190
 Check Error Status, 1-231
 Check Hardware Switches for Drive Type, 1-227
 Check Valid, 1-188
 Check for Bound Instruction, 1-262
 Clear Screen, 1-131
 Clock Calendar, 1-8
 58274A, 1-8
 MM58274 Device Test, 1-115
 Read or Write, 1-242
 Write, 1-246
 Clock Test (8254), 1-138
 Clock Test (MM58274), 1-138
 Code Declaration, 1-2
 Cold Boot, 1-131
 Cold Boot Routine, 1-239
 Color Video Controller, 1-9
 Colors—Set Background and Foreground, 1-217
 Commands, 1-5
 Floppy Disk, 1-8
 i8259A Programmable Interrupt Controller, 1-5

Communication LSI Test, 1-140
 Constants, 1-4
 Global, 1-4
 Keyboard, 1-10
 i8237A p_dma Controller, 1-5
 Control Bytes—i8254 p_timer, 1-6
 Control Ports (General), 1-7
 Controller Tests, 1-95
 8237 p_dma, 1-95, 1-101, 1-103
 8254 p_dma p_timer, 1-95
 8259A Programmable Interrupt Controller, 1-104
 CPU, 1-95
 ROM, 1-95
 Convert BCD to Hexadecimal, 1-245, 1-246
 Convert Hexadecimal to BCD, 1-250
 CPU Interrupt Routines, 1-11
 CPU System Reset Vector, 1-264
 Cursor, 1-202
 Read, 1-203
 Read Attribute and Character, 1-213
 Set Position, 1-203
 Set Value, 1-202
 Write Attribute and Character, 1-214
 Write Character, 1-216

D

Data Area, 1-12
 Additional Keyboard, 1-14
 Fixed Disk, 1-13
 Floppy Disk, 1-13
 Keyboard, 1-12
 Optional Post, 1-13
 Printer and RS-232 Time-Out, 1-14
 ROM BIOS, 1-12
 System, 1-13
 Video Display, 1-13
 i8254 p_timer, 1-13
 Data Declarations, 1-11
 Days Per Month, 1-250
 Decimal Number—Display in ASCII Utilities, 1-144
 Delay Loop (One Millisecond), 1-187
 Deluxe Code Put NUL into kb_buffer, 1-171
 Disable 8259A Programmable Interrupt Controller, 1-125
 Disk Drive Tests, 1-140
 Disk Driver Error Codes (FDU and HDU), 1-8
 Disk Drivers—Initialize, 1-131
 Diskstate, 1-188
 Display ACSII String Utilities, 1-142
 Display Decimal Number in ASCII Utilities, 1-144
 Display Hexadecimal Number in ASCII Utilities, 1-143
 Display Page (Active), 1-204
 Scroll Down, 1-212
 Scroll Up, 1-206
 Set, 1-204
 Terminal Emulator, 1-218

Display Routines, 1-195
DMA Controller, 1-184
Double Zero on Keypad, 1-173

E

Enable Video, 1-120
Equipment Check, 1-237
EQUs to Include Correct Code for Varying Hardware Versions, 1-2
EQUs to Set rom_id Variable for UNIX, 1-2
Error Routine—Fatal, 1-234
Error Status—Check, 1-231

F

Fatal Error Routine, 1-234
FDC—Reset and Reprogram, 1-179
FDU, 1-8
 Controller Constants, 1-8
 Disk Driver Error Codes, 1-8
 Equates, 1-8
 Floppy Disk Commands, 1-8
 Floppy Disk Port Addresses, 1-8
 Parameter Table, 1-229
 Rate Flip-Flops—Write, 1-232
 Test, 1-137
FDU and HDU Disk Driver Error Codes, 1-8
Filename, 1-15
 boot.src, 1-239
 boot1.src, 1-149
 cal.src, 1-242
 com1.src, 1-149
 com2.src, 1-157
 fd6.asm, 1-147
 fd7.asm, 1-163
 graph.src, 1-78
 hdisk.asm, 1-15
 int18.src, 1-233
 kb.data, 1-34
 kb1.src, 1-154
 kb2.src, 1-165
 kb_buffer, 1-171, 1-175
 kb_flag, 1-171
 mem.src, 1-236
 nmi.src, 1-238
 prnscr.src, 1-261
 prt.src, 1-191
 pwrap0.src, 1-130
 pwrap1.src, 1-95
 pwrap1a.src, 1-119
 pwrap2.src, 1-138
 pwrap3.src, 1-140
 pwrap4.src, 1-234
 rtc.src, 1-256
 vector.src, 1-259
 vid.src, 1-195
Fixed Disk Data Area, 1-13

Floppy Disk, 1-8
 Commands, 1-8
 Data Area, 1-13
 Port Addresses, 1-8
Foreground Colors—Set, 1-217
Format Drive, 1-17
Function Keypad, 1-42

G

Game Card, 1-9
General Control Ports, 1-7
Get Byte From FDU Parameter Table, 1-229
Global Constants, 1-4
Graphics Mode, 1-82
 Read Character, 1-87
 Scroll Down, 1-86
 Scroll Up, 1-82
 Write Character, 1-91
Graphics Refresh Memory, 1-84
 Fill Rows With Fill Pattern, 1-85
 Get Offset, 1-95
 Scroll Rows, 1-84
Graphics Support, 1-78
Groups, 1-265, 1-284

H

Hard Disk (Western Digital), 1-15
Hardware Switches—Check for Drive Type, 1-227
HDU Init Code—Call, 1-135
Hexadecimal Number—Display in ASCII Utilities, 1-143

I

i8041A Keyboard Hardware Interrupt Service Routine, 1-165
i8041A Keyboard Software Interrupt Request Routine, 1-154
i8237A p_dma, 1-4
 Controller Constants, 1-5
 Controller Port Addresses, 1-4
i8254 Real-Time Time Clock Test, 1-117
i8254 p_timer, 1-6
 Control Bytes, 1-6
 Counts, 1-6
 Data Area, 1-13
 Hardware Interrupt Service Routine, 1-257
 Port Addresses, 1-6
i8259A Programmable Interrupt Controller, 1-5
 Commands, 1-5
 Hardware Interrupt Routines, 1-11
 Port Addresses, 1-5
Illegal Opcode Interrupt Routine, 1-146
Includes of Assembly Modules, 1-3

Initialize 8259A Programmable Interrupt Controller, 1-125
 Initialize All RAM Variables, 1-131
 Initialize Basic Hardware, 1-131
 Initialize Disk Drivers, 1-131
 Initialize Optional ROMs, 1-136
 Initialize Parallel Printer Interface, 1-194
 Initialize RS-232 Interface, 1-153
 Initialize Video, 1-107
 Input Drive Parameter, 1-228
 INS8250, 1-9
 Asynchronous Communication Chip, 1-9
 Baud Rate Time Constants, 1-150
 Compatible Modem Control Bits, 1-150
 Get Byte, 1-162
 Put Byte, 1-160
 Install Vector Table, 1-124
 Interrupt Locations, 1-11
 Interrupt Pointers—Set Up, 1-131
 Interrupt Routines, 1-11
 CPU, 1-11
 Illegal Opcode, 1-146
 Printer Software, 1-191
 RS-232 Software, 1-151
 Software, 1-11
 Time of Day, 1-256
 Unused Hardware and Illegal Software
 Interrupts, 1-260
 Video, 1-196
 i8041A Keyboard Software, 1-154
 i8254 p_timer Hardware, 1-257
 i8259A Programmable Interrupt Controller, 1-11
 Interrupt Vector Area, 1-49
 Interrupt Vectors, 1-259

J

Jump Table, 1-153

K

Keyboard, 1-7
 Boot-Strap Options, 1-140
 Constants, 1-10
 Controller (8041), 1-7
 Data Area, 1-12
 LEDs, 1-173
 Scan Code Translation Data, 1-34
 Test (8041), 1-140
 i8041A Hardware Interrupt Service Routine, 1-165
 i8041A Software Interrupt Request Routine, 1-154
 Keypad, 1-41
 Alternate, 1-172
 Double Zero, 1-173
 Function, 1-42
 Numeric, 1-41

L

Light Pen—Read, 1-204, 1-225

M

Macro Definitions, 1-2
 Macros, 1-265
 Media Change, 1-190
 Memory Size Detect, 1-236
 Message Routines, 1-140
 MM58274 Clock Calendar Device Test, 1-115
 MM58274 Clock Test, 1-138
 Monochrome Video Controller, 1-9
 Motherboard Switches—Read, 1-227

N

NEC, 1-183
 Controller—Read Result Bytes, 1-183
 Controller—Send Byte, 1-232
 Interrupt, 1-230
 Ready, 1-229
 Numeric Keypad, 1-41

O

Optional Post Data Area, 1-13
 ORG'd Font Tables, 1-251
 OS Merge Link Address, 1-14
 Overscan—Set, 1-217

P

Parallel Printer Interface, 1-9
 Initialize, 1-194
 Print Character, 1-193
 Read Status, 1-194
 Parameters for Diskette Operation, 1-190
 Passing Error Messages—Display, 1-107
 Pause Waiting for Another Key, 1-170
 PC 6300 PLUS Addresses, 1-4
 Port Addresses, 1-4
 Floppy Disk, 1-8
 i8237A p_dma Controller, 1-4
 i8254 p_timer, 1-6
 i8259A Programmable Interrupt Controller, 1-5
 Print Character to Parallel Printer Interface, 1-193
 Print Screen, 1-261
 Print Screen Sequence, 1-170
 Printer Software Interrupt Request Routine, 1-191
 Printer Time-Out Data Area, 1-14

Programmable Interrupt Controller, 1-125
 Disable, 1-125
 Initialize, 1-125
 Output Mask, 1-126
Protected Mode Data Space, 1-15

R

RAM, 1-12
 Initialize All Variables, 1-131
 Size and Clear, 1-108
 Stack, 1-12
 Storage Test, 1-128
 Test, 1-17, 1-140
 Workspace, 1-50
Read Attribute and Character at Cursor, 1-213
Read Character in Graphics Mode, 1-87
Read Clock Calendar Device, 1-242
Read Current Video Status, 1-220
Read Cursor, 1-203
Read DASD Type, 1-226
Read Dot, 1-79, 1-218
Read Light Pen, 1-204, 1-225
Read Motherboard Switches, 1-227
Read Result Bytes From NEC Controller, 1-183
Read Status of Parallel Printer Interface, 1-194
Read Status of RS-232 Interface, 1-157
Register Initialization Routine, 1-221
Registers (6845)—Output Two Bytes, 1-205
Reserved, 1-14
Reset and Reprogram FDC, 1-179
ROM, 1-100
 Data, 1-195
 Module Test, 1-100
 Test, 1-140
ROM BIOS Data Area, 1-12
ROMs—Test for and Initialize, 1-136
RS-232 Interface, 1-14
 Initialize, 1-153
 Read Status, 1-157
 Software Interrupt Request Routine, 1-151
 Time-Out Data Area, 1-14
 Wait for Status, 1-159

S

Scan Code Translation Data (Keyboard), 1-34
Screen Sequence—Print, 1-170
Screen—Print, 1-261
Screen—Set Mode and Clear, 1-198
Scroll Active Page Down, 1-212
Scroll Active Page Up, 1-206
Scroll Down in Graphics Mode, 1-86
Scroll Positioning and Register Initialization Routine, 1-221

Scroll Rows in Graphics Refresh Memory, 1-84
Scroll Up in Graphics Mode, 1-82
Sector Buffer, 1-17
Seek, 1-186
Segments, 1-265, 1-284
Send Byte to NEC Controller, 1-232
Sense Interrupt Status, 1-231
Set Active Display Page, 1-204
Set Cursor Position, 1-203
Set Cursor Value, 1-202
Set Format Transfer Rate, 1-163
Set Mode and Clear Screen, 1-198
Set Original State, 1-228
Set Overscan and Background/Foreground Colors, 1-217
Set Transfer Rate, 1-228
Software Interrupt Routines, 1-11
State Shifts, 1-171
Symbols, 1-265, 1-284
System Configuration From Switches and Enable Video, 1-120
System Configuration—Determine From Switches, 1-107
System Data Area, 1-12, 1-13
System Initialization, 1-140
System Reset Sequence—Test for, 1-170

T

Temporary Hardware Initialization, 1-130
Terminal Emulator to Active Page, 1-218
Test for Retry, 1-147
Test for System Reset Sequence, 1-170
Time Clock (i8254 Real-Time), 1-117
Time of Day Software Interrupt Request Routine, 1-256

U

Utility Routines, 1-141

V

Vector Table—Install, 1-107, 1-124
Video Controller (Color and Monochrome), 1-9
Video Controller Test, 1-138
Video Display Data Area, 1-13
Video Interrupt Service Routine, 1-196
Video RAM, 1-15
Video RAM Buffer—Calculate Offset of Character in Text Mode, 1-223
Video RAM Byte Location, 1-80
Video ROMS—Test for and Initialize, 1-122
Video Status—Read Current, 1-220

W

Wait for Status of RS-232 Interface, 1-159
Western Digital Hard Disk, 1-15
Write Attribute and Character at Cursor, 1-214
Write Character at Cursor Position, 1-216
Write Character in Graphics Mode, 1-91
Write Clock Calendar Device, 1-242, 1-246
Write Dot, 1-79, 1-218
Write to FDU Rate Flip-Flops, 1-232

