START J094A

Δ M094C 42 TO 32
Δ M0955 14 TO 12

0840-0FFA

EXIT: ESCAPE

6800 Trace and Disassemble Program

This program puts you on the trail of runaway routines.

Richard Carickhoff
812 Pulaski Dr.
Lansdale PA 19446

Did you ever write a program that didn't work and then spend hours, or even days, debugging it? Did you ever wonder how the program got to that particular location? . . . why that compare instruction wasn't working as you thought it should? . . . why that multiply routine didn't work?

Well, I've been down that road many times myself, so I decided to write a program that would allow me to trace a program instruction by instruction while, at the same time, see exactly what was taking place before and after the execution of each instruction.

The 6800 Trace and Disassemble program does just that. The program enables the user to perform the following functions:

- Program trace function
- Go to user's program function
- Program disassemble function
- Memory examine and change function
- Register examine and change function

The detailed explanations, along with operating procedures for each of these functions, are described in the following paragraphs.

At the start of each function it is assumed that the last data character printed by the terminal is a colon (:), which is the program's prompt character. All values entered must be in hexadecimal format.

Program Trace Function

The program trace function will trace the user's program one instruction at a time, while outputting to the terminal the location, mnemonic, operand, contents of all MPU registers (CC, B, A, X, SP) and the next return address in the stack. The trace function will do this for each instruction prior to its execution.

The trace funtion can be performed by typing one of the following two responses:

: T nnnn
or
: T nnnn, mmmm

The first response must be terminated with a carriage return. The character T specifies a trace function. The four hexadecimal digits following T specify the starting address of the first instruction to be traced. This response instructs the program to trace only one instruction (see Example 1).

At this point the trace function waits for the operator to enter a character. If the character is any character other than the Escape (1B hex), the instruction displayed will be executed and the next instruction will be output to the terminal along with the contents of all the MPU registers (see Example 2).

The contents of the following MPU registers are printed along with each instruction:

cc—Condition code register
b—B register
a—A register
x—X register
sp—Stack pointer

```
: T 0103                    cc  b   a   x    sp    rtn
0103  JMP  0225             8F  19  FF  2242 A049  7B05
```

Example 1.

```
: T 0103                    cc  b   a   x    sp    rtn
0103  JMP  0225             8F  19  FF  2242 A049  7B05
0225  LDS  22               CF  19  FF  2242 A049  7B05
```

Example 2.

```
: G 0103,022F
022F   LDX   #0080   C1   00   00   07A1   0000
```

Example 3.

If the program does not reach the breakpoint address and the operator wishes to return to the trace and disassemble program, he must perform a system reset and return through the system monitor. However, the software interrupt still exists at the breakpoint address.

To remove the interrupt and replace it with the original instruction, the Go to User's Program function can be executed where the starting address is set to the breakpoint address. The program will immediately return, displaying the original instruction at the terminal. The operator can then terminate the trace function by depressing the Escape key.

```
: D 0225
0225   9E   22   LDS   22
```

Example 4.

```
: D 0225
0225   9E   22     LDS   22
0227   BD   062C   JSR   062C
```

Example 5.

Program Disassemble Function

This function allows the operator to disassemble any 6800 program including the Trace and Disassemble program itself. The disassemble function can disassemble one instruction at a time or a sequence of instructions, while outputting to the terminal the location, object code, mnemonic and operand for each instruction.

The disassemble function can be performed by typing one of the following two responses:

: D nnnn

or

: D nnnn, mmmm

The first response must be terminated with a carriage return. The character D specifies a disassemble function. The four hexadecimal digits following D specify the starting address of the instruction to be disassembled (see Example 4).

At this point the disassemble function waits for the operator to enter a character. If the character is any character other than an Escape, the next instruction in sequence will be disassembled (see Example 5). In doing so, the operator can step through a disassembly of a program one instruction at a time.

The second response is used to disassemble a list of instructions. The first four hexadecimal digits specify the first instruction to be disassembled.

```
:D 0225,0280
0225 9E 22      LDS   22
0227 BD 062C    JSR   062C
022A FE 01FE    LDX   01FE
022D DF 2A      STX   2A
022F CE 0080    LDX   #0080
0232 DF C2      STX   C2
0234 CE 0030    LDX   #0030
0237 DF C0      STX   C0
0239 9F 26      STS   26
023B 8D B8      BSR   01F5
023D 8D 07      BSR   0246
023F 20 FA      BRA   023B
0241 8C 1066    CPX   #1066
0244 20 F3      BRA   0239
0246 CE 0117    LDX   #0117
0249 DF BC      STX   BC
024B 81 30      CMP A #30
024D 24 56      BCC   02A5
024F 81 08      CMP A #08
0251 25 91      BCS   01E4
0253 48         ASL A
0254 97 BD      STA A BD
0256 DE BC      LDX   BC
0258 EE 17      LDX   17,X
025A 6E 00      JMP   00,X
025C BD 062C    JSR   062C
025F 86 21      LDA A #21
0261 97 C1      STA A C1
0263 BD 0109    JSR   0109
0266 86 80      LDA A #80
0268 97 C3      STA A C3
026A D6 2B      LDA B 2B
026C 96 2A      LDA A 2A
026E F0 01FF    SUB B 01FF
0271 B2 01FE    SBC A 01FE
0274 BD 0542    JSR   0542
0277 96 C0      LDA A C0
0279 27 0F      BEQ   028A
027B CE 0293    LDX   #0293
027F DF 2A      STX   2A
0280 BD 05AD    JSR   05AD
```

Program B. Disassemble function.

The second four hexadecimal digits following the comma specify the last instruction to be disassembled.

Once the last digit is entered, the program will immediately list each instruction in sequence until the last address is reached. The last address specified must be on an instruction boundary. Otherwise, the disassembly will continue past the last address. The Escape key can be used to terminate any list sequence.

When the last address is reached, the disassembly will stop. The operator can continue the disassembly one instruction at a time by depressing any key other than Escape. Otherwise, the Escape key will terminate the disassembly and return control back to the control

rtn—First return address at the top of the stack

The contents of the program counter is the location of the instruction to be executed.

With the use of the trace function, the operator can step through his program one instruction at a time. The contents of all the MPU registers are always visible before and after the execution of each instruction. Also, the instruction is always printed before it is executed so the operator can decide whether to terminate the trace at that point (depressing Escape key) or to continue.

The second response to the prompt character is used to trace a program until the breakpoint address is reached. The first four hexadecimal digits define the starting address of the first instruction of the trace sequence. The second four hexadecimal digits following the comma define the breakpoint address. Once the last digit is entered, the program will immediately start tracing the program starting at the start address.

The output format is the same as the single trace function except that the program will continue outputting each instruction until the breakpoint address is reached. At that point the trace function operates in the same manner as the single trace function. That is, depressing the Escape key terminates the trace and depressing any other key executes the last instruction printed and outputs the next instruction. The Escape key is also used to terminate a trace sequence prior to reaching the breakpoint address.

Caution: The trace function traces a program with the use of the software interrupt (SWI). Always terminate any trace sequence using the Escape key. Using the system reset may leave a software interrupt in the user's program.

This method of tracing a program is normally used to determine how a program arrived at a particular location. If a CRT is being used for a terminal, the last 15 instructions executed will still appear on the screen (assuming the CRT has a minimum of 16 lines). The rate at which the program executes is controlled by the output rate of the terminal being used.

Program A shows an example of the trace function. The program selected is Tom Pittman's 6800 Tiny BASIC. I chose this program because it is well known and is an interesting program to trace. It also demonstrates the visibility of a program using the trace function.

The starting address was set at 0103 hex, which is Tiny BASIC's warm start address. The breakpoint address was set at an address that would not be reached. This allowed me to terminate the program at any point during the trace.

In Program A there are several instructions that are disassembled with asterisks (***) for the mnemonic and ROM for the operand. This alerts the operator that the trace function came upon a ROM address that could not be loaded with the software interrupt. The trace function in this case places the software interrupt at the return address. The trace function assumes that routines in ROM are functional and always return via the RTS (return subroutine) instruction.

The ROM address shown in Program A is the MIKBUG output routine (E1D1). Examining the contents of the A register prior to executing the output routine shows the character being output. Also, the output is reflected in the trace printout as indicated by the line feed following the first output by Tiny BASIC.

```
:T 0103,0FFF
0103  JMP    0225     C1 19 0D 2242 A07D 022A
0225  LDS    22       C1 19 0D 2242 A07D 022A
0227  JSR    062C     C9 19 0D 2242 A07F 0000
062C  LDA A  #0D      C9 19 0D 2242 A07D 022A
062E  BSR    0649     C1 19 0D 2242 A07D 022A
0649  CLR    00BF     C1 19 0D 2242 A07B 0630
064C  JMP    0598     C4 19 0D 2242 A07B 0630
0598  INC    00BF     C4 19 0D 2242 A07B 0630
059B  BMI    05A7     C0 19 0D 2242 A07B 0630
059D  STX    BA       C0 19 0D 2242 A07B 0630
059F  PSH B           C0 19 0D 2242 A07B 0630
05A0  JSR    0109     C0 19 0D 2242 A07A 1906
0109  JMP    E1D1     C0 19 0D 2242 A078 05A3
E1D1  ***    ROM

05A3  PUL B           C0 19 0D 2242 A07A 1906
05A4  LDX    BA       C0 19 0D 2242 A07B 0630
05A6  RTS             C0 19 0D 2242 A07B 0630
0630  LDA B  0111     C0 19 0D 2242 A07D 022A
0633  ASL B           C0 03 0D 2242 A07D 022A
0634  BEQ    063E     C0 06 0D 2242 A07D 022A
0636  PSH B           C0 06 0D 2242 A07D 022A
0637  BSR    0642     C0 06 0D 2242 A07C 0602
0642  CLR A           C0 06 0D 2242 A07A 0639
0643  TST    0111     C4 06 00 2242 A07A 0639
0646  BPL    0649     C0 06 00 2242 A07A 0639
0649  CLR    00BF     C0 06 00 2242 A07A 0639
064C  JMP    0598     C4 06 00 2242 A07A 0639
0598  INC    00BF     C4 06 00 2242 A07A 0639
059B  BMI    05A7     C0 06 00 2242 A07A 0639
059D  STX    BA       C0 06 00 2242 A07A 0639
059F  PSH B           C0 06 00 2242 A07A 0639
05A0  JSR    0109     C0 06 00 2242 A079 0606
0109  JMP    E1D1     C0 06 00 2242 A077 05A3
E1D1  ***    ROM
05A3  PUL B           C1 06 00 2242 A079 0606
05A4  LDX    BA       C1 06 00 2242 A07A 0639
05A6  RTS             C1 06 00 2242 A07A 0639
0639  PUL B           C1 06 00 2242 A07C 0602
063A  DEC B           C1 06 00 2242 A07D 022A
063B  DEC B           C1 05 00 2242 A07D 022A
063C  BNE    0636     C1 04 00 2242 A07D 022A
0636  PSH B           C1 04 00 2242 A07D 022A
0637  BSR    0642     C1 04 00 2242 A07C 0402
0642  CLR A           C1 04 00 2242 A07A 0639
0643  TST    0111     C4 04 00 2242 A07A 0639
0646  BPL    0649     C0 04 00 2242 A07A 0639
0649  CLR    00BF     C0 04 00 2242 A07A 0639
064C  JMP    0598     C4 04 00 2242 A07A 0639
0598  INC    00BF     C4 04 00 2242 A07A 0639
059B  BMI    05A7     C0 04 00 2242 A07A 0639
059D  STX    BA       C0 04 00 2242 A07A 0639
059F  PSH B           C0 04 00 2242 A07A 0639
05A0  JSR    0109     C0 04 00 2242 A079 0406
0109  JMP    E1D1     C0 04 00 2242 A077 05A3
E1D1  ***    ROM
```

Program A.

Trace Function Restrictions

There are only two restrictions on the trace function. The first is that it will not trace a program that uses a software interrupt, since the software interrupt interferes with the trace function's software interrupt. The second restriction is that the trace function cannot be used to trace itself.

Go to User's Program Function

This function allows the operator to execute his program. The operator may specify a breakpoint address in order to return to the trace program. This function can be performed by typing one of the following two responses:

: G nnnn

or

: G nnnn, mmmm

The first response must be terminated with a carriage return. The character G specifies a Go function. The four hexadecimal digits following G specify the starting address of the program to be executed (e.g., : G 0103).

The only way to return to the Trace and Disassemble program with this response is through the system monitor.

The second response is used to execute a user's program until the breakpoint address is reached. The first four hexadecimal digits define the starting address of the program to be executed. The second four hexadecimal digits following the comma define the breakpoint address. Once the last digit is entered, the MPU will start executing the user's program. Once the breakpoint address is reached, the control of the program is returned to the trace function (see Example 3).

The program can be traced from this point one instruction at a time by simply depressing any key other than the Escape key. The trace will operate in the same manner as if a trace function was being performed.

```
MPU Register   : R
cc             A077 C1        (space)
B              A078 19   FE
A              A079 0D   A0
XH             A07A 22        (space)
XL             A07B 42        (space)
PCH            A07C 01        (space)
PCL            A07D 03        (space)
RTNH           A07E 02        (space)
RTNL           A07F 2A        (space)
               A080 FF        (escape)
               : T 0103
               0103   JMP   0225   C1   FE   A0   2242   A07D   022A
```

Example 7.

amine and change the contents of the MPU registers prior to executing the trace or Go to User's Program function. The trace and Go to User's Program functions use the return from interrupt (RTI) instruction to return to the user's program. The RTI instruction updates all the MPU registers with the values stored away in the stack.

The register examine and change function is initiated by entering the character R after the colon. The location of the first MPU register and its contents will be printed. The examining and changing of the data is done in the same manner as the M function (see Example 7).

Basic Memory Map

The 6800 Trace and Disassemble program resides in less than 2K of memory. The hex listing accompanies the article. The program uses some of the MIKBUG I/O routines. Table 1 lists I/O routines used by the program.

There are some parameters that may have to be changed depending on your particular machine. The stack pointer, for example, is initially loaded to $A042. If this value is changed, it should be set to at least ten locations down from the top of the stack.

The software interrupt vector is normally stored at location $FFFA. In my home-brew system the software interrupt vector points to a ROM subroutine that uses location $A014 as a programmable software interrupt vector. The Trace and Disassemble program initializes location $A014 to the return address of the trace function. This address ($A014) in the program will have to be changed to $FFFA (if programmable) or to whatever the programmable location is in your particular machine.

The Back Space and Escape Codes can be modified. They are presently set to 08 hex and 1B hex, respectively.

Break Test Routine

The break test is used by the program during a trace or disassemble program function. After each line of output the program jumps to the break test routine. The break test checks for a key being depressed. If one is not, the program returns normally. If a key is depressed, the character is input and tested for the Escape Code. If the character is not the Escape Code, the program exits from the routine normally. If the character is the Escape Code, the program returns to the control monitor.

Any changes to the break test must be made within the first three instructions. The remaining four are used by other routines within the program. There are some spare locations at the end of the program starting at $0FF4 for modifications to the break test (see Example 8).

Summary

The 6800 Trace and Disassemble program is an effective debugging tool. It requires no hardware changes, as long as your system has a programmable SWI vector. I've used it many times and so have other 6800 users. It allows you to trace your program instruction by instruction. You can make changes to your program, disassemble your patches and then trace them. You can make a listing of your program and even the trace of your program.

If you would like to get a copy of the listing of the program for relocation purposes or whatever, just send $5 with your name and address to:

Richard Carickhoff
812 Pulaski Drive
Lansdale PA 19446

If you have any problems with the program just send a self-addressed, stamped envelope to me and I'll try to answer any questions that you may have.■

```
                    JMP  $E1AC     OUTPUT 2 HEX CHARS AND SPACE
BASIC MEMORY MAP
0900-0911           I/O ROUTINES
0912-0949           TEMPORARY STORAGE
094A                START OF PROGRAM
094A-0D18           EXECUTABLE PROGRAM
0B8A-0BA2           PROMPT, INVALID CODE AND CRLF MESSAGES
0D19-0FF3           MNEMONIC AND CODE TABLES
MIKBUG I/O ROUTINES
0900        JMP  $E0CC     OUTPUT SPACE
0903        JMP  $E0CA     OUTPUT 2 HEX CHARS AND SPACE
0906        JMP  $E0C8     OUTPUT 4 HEX CHARS AND SPACE
0909        JMP  $E07E     OUTPUT MESSAGE
090C        JMP  $E1AC     INPUT A CHAR
090F        JMP  $E1D1     OUTPUT A CHAR
PARAMETERS
094B-094C   $A042          MIDDLE OF STACK
0954-0955   $A014          SWI VECTOR (NORMALLY $FFFA)
0A37        $08            BACKSPACE CODE
0AB7        $1B            ESCAPE CODE
```

Table 1. Memory map of I/O routines and parameters.

T nnnn (CR)	Trace instruction at location nnnn.
T nnnn, mmmm	Trace program starting at location nnnn with breakpoint address set at mmmm.
G nnnn (CR)	Go to user's program starting at location nnnn.
G nnnn, mmmm	Go to user's program starting at location nnnn with breakpoint address set at mmmm.
D nnnn (CR)	Disassemble instruction at location nnnn.
D nnnn, mmmm	Disassemble instruction at location nnnn and ending at location mmmm.
M nnnn	Examine memory location nnnn.
R	Examine MPU registers starting with condition code.
(ESC)	Escape from present function and return to control monitor.

Table 2. Summary of control functions.

```
0AAE   BREAK   LDAA   $8009     PIA STATUS-KEY DEPRESSED?
0AB1           BPL    EXIT       NO
0AB3           LDAA   $8008     YES, INPUT CHAR
0AB6   CHECK   CMPA   #1B       ESCAPE CODE?
0AB8           BNE    EXIT       NO
0ABA           JMP    CONTROL    YES, RETURN TO CONTROL MONITOR
0ABD   EXIT    RTS               RETURN NORMAL
```

Example 8.

```
0900  7E E0 CC 7E E0 CA 7E E0 C8 7E E0 7E 7E E1 AC 7E
0910  E1 D1 A6 42 00 00 00 00 0A 7E 00 08 00 0A 00 00
0920  09 00 30 30 30 39 04 30 38 20 20 20 20 20 20 20
0930  20 49 4E 58 20 20 20 20 20 20 20 20 20 20 04
0940  00 00 00 00 00 00 00 00 00 00 8E A0 42 BF 09 12
0950  CE 0A E0 FF A6 14 BE 09 12 7F 09 21 CE 0B 8A BD
0960  09 09 BD 09 0C 16 BD 09 00 C1 44 27 18 C1 47 27
0970  7B C1 54 27 27 C1 52 27 06 C1 4D 27 05 20 D7 7E
0980  0A 11 7E 0A 17 BD 09 CE BD 0B A3 CE 09 22 BD 09
0990  09 CE 0B 9C BD 09 09 BD 0A 97 20 EC BD 09 CE 30
09A0  B6 09 17 A7 05 B6 09 18 A7 06 BD 0B A3 CE 09 22
09B0  86 04 B7 09 26 BD 09 09 CE 09 2F BD 09 09 BD 0A
09C0  BE CE 0B 9C BD 09 09 BD 0A 97 BD 0B 11 3B BD 0A
09D0  65 FF 09 17 BD 09 0C 81 0D 27 0A BD 0A 59 FF 09
09E0  1F 7C 09 21 39 CE 0B 9C BD 09 09 39 BD 0A 65 FF
09F0  09 15 30 B6 09 15 A7 05 B6 09 16 A7 06 7F 09 1E
0A00  BD 09 0C 81 0D 27 09 BD 0A 59 FF 09 17 BD 0B 11
0A10  3B 30 FF 09 15 20 03 BD 0A 65 CE 0B 9C BD 09 09
0A20  CE 09 15 BD 09 06 FE 09 15 BD 09 03 FF 09 15 BD
0A30  09 0C 81 20 27 E4 81 08 26 0A FE 09 15 09 09 FF
0A40  09 15 20 D6 BD 0A 81 BD 0A 75 09 A7 00 A1 00 27
0A50  C9 B6 3F BD 09 0F 7E 09 56 8D 0A CE 0B 9C BD 09
0A60  09 FE 09 15 39 8D 0C B7 09 15 8D 07 B7 09 16 FE
0A70  09 15 39 8D 09 48 48 48 48 16 8D 02 1B 39 BD 09
0A80  0C 80 30 2B 0F 81 09 2F 0A 81 11 2B 07 81 16 2E
0A90  03 80 07 39 7E 09 56 7D 09 21 27 0D FE 09 17 BC
0AA0  09 1F 26 0A 7F 09 21 20 05 BD 09 0C 20 08 B6 80
0AB0  09 2A 0A B6 80 08 81 1B 26 03 7E 09 56 39 FE 09
0AC0  12 08 BD 09 03 BD 09 03 BD 09 03 BD 09 06 08 FF
0AD0  09 15 CE 09 15 BD 09 06 FE 09 15 08 BD 09 06 39
0AE0  BF 09 12 30 6D 06 26 02 6A 05 6A 06 8D 09 30 EE
0AF0  05 FF 09 17 7E 09 AA B6 09 1E 84 3C 81 08 26 08
0B00  FE 09 19 B6 09 14 A7 00 FE 09 17 B6 09 1B A7 00
0B10  39 B6 09 1E 84 1C 26 03 8D 32 39 81 04 26 08 FE
0B20  09 19 FF 09 17 20 F1 81 08 26 04 8D 53 20 E9 81
0B30  0C 26 11 4F F6 09 19 30 EB 06 A9 05 B7 09 17 F7
0B40  09 18 20 D4 30 EE 09 FF 09 17 20 CC FE 09 17 A6
0B50  00 B7 09 1B 86 3F A7 00 A1 00 27 23 CE 09 17 BD
0B60  09 06 CE 0B 92 BD 09 09 BD 0A 97 B6 09 1E 85 20
0B70  27 05 FE 09 1C 20 03 30 EE 0B FF 09 17 20 CD 39
0B80  FE 09 19 A6 00 B7 09 14 20 CA 0D 0A 00 00 00 00
0B90  3A 04 20 2A 2A 2A 20 20 20 52 4F 4D 0D 0A 00 00
0BA0  00 00 04 CE 09 22 C6 1D 86 20 A7 00 08 5A 26 FA
0BB0  86 04 A7 00 FE 09 17 A6 00 B7 09 1B 48 CE 0D F4
0BC0  FF 09 1C 24 03 7C 09 1C F6 09 1D 1B B7 09 1D 24
0BD0  03 7C 09 1C CE 09 22 B6 09 17 BD 0C FF B6 09 18
0BE0  BD 0C FF FE 09 1C A6 00 B7 09 1C E6 01 F7 09 1E
0BF0  C4 03 FE 09 17 A6 01 B7 09 19 A6 02 B7 09 1A 37
0C00  CE 09 27 B6 09 1B BD 0C FF 08 5A 27 0F B6 09 19
0C10  BD 0C FF 5A 27 06 B6 09 1A BD 0C FF 33 FE 09 17
0C20  08 5A 26 FC FF 09 17 B6 09 1C 16 CE 0D 19 FF 09
0C30  1C 48 24 04 7C 09 1C 0C 1B 24 03 7C 09 1C 0C BB
0C40  09 1D 24 03 7C 09 1C B7 09 1D FE 09 1C A6 00 B7
0C50  09 31 A6 01 B7 09 32 A6 02 B7 09 33 C6 20 F7 09
0C60  34 B6 09 1E 85 C0 27 08 2A 04 C6 41 20 02 C6 42
0C70  F7 09 35 CE 09 37 B6 09 31 81 2A 26 08 B6 09 1B
0C80  BD 0C FF 20 40 B6 09 1E 85 02 27 39 B6 09 1B 81
0C90  8D 27 12 84 F0 B7 09 1B 81 80 27 04 81 C0 26 05
0CA0  86 23 A7 00 08 B6 09 1B 81 8D 27 38 84 F0 81 20
0CB0  27 32 B6 09 19 BD 0C FF B6 09 1E 85 01 27 06 B6
0CC0  09 1A BD 0C FF B6 09 1B 81 60 27 08 81 A0 27 04
0CD0  81 E0 26 09 86 2C A7 00 08 86 58 A7 00 FE 09 17
0CE0  FF 09 1C 39 4F F6 09 19 2A 01 4A 0C FB 09 16 F7
0CF0  09 1A B9 09 17 B7 09 19 8D 05 17 8D 02 20 DE 36
0D00  8D 06 32 08 8D 06 08 39 44 44 44 44 84 0F 8B 30
0D10  81 39 23 02 8B 07 A7 00 39 2A 2A 2A 4E 4F 50 54
0D20  41 50 54 50 41 49 4E 58 44 45 58 43 4C 56 53 45
0D30  56 43 4C 43 53 45 43 43 4C 49 53 45 49 53 42 41
0D40  43 42 41 54 41 42 54 42 41 44 41 41 41 42 41 42
0D50  52 41 42 48 49 42 4C 53 42 43 43 42 43 53 42 4E
0D60  45 42 45 51 42 56 43 42 56 53 42 50 4C 42 4D 49
0D70  42 47 45 42 4C 54 42 47 54 42 4C 45 54 53 58 49
0D80  4E 53 50 55 4C 44 45 53 54 58 53 50 53 48 52 54
0D90  53 52 54 49 57 41 49 53 57 49 4E 45 47 43 4F 4D
0DA0  4C 53 52 52 4F 52 41 53 52 41 53 4C 52 4F 4C 44
0DB0  45 43 49 4E 43 54 53 54 43 4C 52 4A 4D 50 53 55
0DC0  42 43 4D 50 53 42 43 41 4E 44 42 49 54 4C 44 41
0DD0  45 4F 52 41 44 43 4F 52 41 41 44 44 43 50 58 42
0DE0  53 52 4C 44 53 53 54 41 53 54 53 4A 53 52 4C 44
0DF0  58 53 54 58 00 01 01 01 00 01 00 01 00 01 00 01
0E00  02 01 03 01 04 01 05 01 06 01 07 01 08 01 09 01
0E10  0A 01 0B 01 0C 01 0D 01 0D 01 00 01 00 01 00 01
0E20  0E 01 0F 01 00 01 10 01 00 01 11 01 00 01 00 01
0E30  00 01 00 01 12 06 00 01 13 0A 14 0A 15 0A 16 0A
0E40  17 0A 18 0A 19 0A 1A 0A 1B 0A 1C 0A 1D 0A 1E 0A
0E50  1F 0A 20 0A 21 01 22 01 23 81 23 41 24 01 25 01
0E60  26 81 26 41 00 01 27 11 00 01 28 11 00 01 00 01
0E70  29 01 2A 01 2B 81 00 01 00 01 2C 81 2D 81 00 01
0E80  2E 81 2F 81 30 81 31 81 32 81 00 01 33 81 34 81
0E90  00 01 35 81 2B 41 00 01 00 01 2C 41 2D 41 00 01
0EA0  2E 41 2F 41 30 41 31 41 32 41 00 01 33 41 34 41
0EB0  00 01 35 41 2B 02 00 01 00 01 2C 02 2D 02 00 01
0EC0  2E 02 2F 02 30 02 31 02 32 02 00 01 33 02 34 02
0ED0  36 0E 35 02 2B 03 00 01 00 01 2C 03 2D 03 00 01
0EE0  2E 03 2F 03 30 03 31 03 32 03 00 01 33 03 34 03
0EF0  36 07 35 03 37 82 38 82 39 82 00 01 3A 82 3B 82
0F00  3C 82 00 01 3D 82 3E 82 3F 82 40 82 41 03 42 26
0F10  43 03 00 01 37 82 38 82 39 82 00 01 3A 82 3B 82
0F20  3C 82 44 82 3D 82 3E 82 3F 82 40 82 41 02 00 01
0F30  43 02 45 02 37 82 38 82 39 82 00 01 3A 82 3B 82
0F40  3C 82 44 82 3D 82 3E 82 3F 82 40 82 41 02 46 2E
0F50  43 02 45 02 37 83 38 83 39 83 00 01 3A 83 3B 83
0F60  3C 83 44 83 3D 83 3E 83 3F 83 40 83 41 03 46 27
0F70  43 03 45 03 37 42 38 42 39 42 00 01 3A 42 3B 42
0F80  3C 42 00 01 3D 42 3E 42 3F 42 40 42 00 01 00 01
0F90  47 03 00 01 37 42 38 42 39 42 00 01 3A 42 3B 42
0FA0  3C 42 44 42 3D 42 3E 42 3F 42 40 42 00 01 00 01
0FB0  47 02 48 02 37 42 38 42 39 42 00 01 3A 42 3B 42
0FC0  3C 42 44 42 3D 42 3E 42 3F 42 40 42 00 01 00 01
0FD0  47 02 48 02 37 43 38 43 39 43 00 01 3A 43 3B 43
0FE0  3C 43 44 43 3D 43 3E 43 3F 43 40 43 00 01 00 01
0FF0  47 03 48 03 00 00 00 00 00 00 00 00 00 00 00 00
>
```

Hex listing of Trace and Disassemble program.

monitor.

Program B shows the disassembly of Tiny BASIC starting at address 0225 hex and finishing at 0280 hex. All values are in hexadecimal. Branch operands are the actual branch address. Direct addressing instructions are shown with two digit operands. If a location does not contain a valid op code, the disassembler will assume it is data and output asterisks (***) for the mnemonic.

Memory Examine and Change Function

This function can be used by the operator for inputting a program or making changes to an existing program. This function can be performed by typing in the following response:

: M nnnn

The character M specifies a memory change function. The four hexadecimal digits following M specify the address to be examined or changed. Once the last digit is entered, the program will respond with the address and its contents:

: M 0103
0103 7E

The operator must now decide whether to change memory, space to the next location, back space to the previous location or return to the control monitor.

If the contents of memory are to be changed, just enter the new value. The program will automatically output the next address and its contents. If the contents of memory cannot be changed, the program will output a (?) and return to the control monitor.

If the operator wishes to space to the next location, he'll just depress the space bar. The program will output the next location and its contents. For back spacing to the previous location, just depress the back space key (08 hex). The program will output the previous location and its contents. The back space function is useful for back spacing when an incorrect value is entered.

The memory change function can be terminated by depressing the Escape key or entering an invalid hex character (see Example 6).

Register Examine and Change Function

This function is used to ex-

```
: M 0103
0103   7E      (space)
0104   02      (back space)
0103   7E  BD
0104   02      (back space)
0103   BD  7E
0104   02      (space)
0105   25      (back space)
0104   02      (back space)
0103   7E      (escape)
```

Example 6.

To use the program, do the following:

1. Load the program.
2. Use the MIKBUG memory change function to set the program counter, addresses A048 and A049, to 0EB0, the start address of the relocator program.
3. From the MIKBUG monitor, type "G" to begin execution. The program will prompt with a carriage return, line feed, "?", and a space.
4. Type a "P" for program segment or a "D" for data segment to be transferred. The computer will respond with a space.
5. Enter (in hexadecimal) the start address of the program or data block to be transferred, the old end address, and the new start address. The computer will put spaces between each. Transfer is completed when the prompt is re-displayed. Additional segments may be transferred by returning to step 4.

Table 3. System Requirements

NAME:	BLKXFER
FUNCTION:	Move program or data block from one memory location to another, retaining executability of program in new location.
RESULTS:	New program is produced in different memory location.
HARDWARE CONFIGURATION:	SWTPC 6800 microprocessor, CT-1024 TV terminal, AC-30 cassette interface.
MEMORY REQUIRED:	Program resides in $0EB0 to $0E56. Scratchpad memory required from $A002 to $A017. Memory also required for program in initial and final storage locations.
SOFTWARE SUPPORT:	MIKBUG
ASSEMBLER:	Motorola Co-Resident Assembler

257

M6800 Program Relocator

By Dr. Gordon W. Wolfe

One of the major advantages of a 6800-based microcomputer is the great amount of software available, either in microcomputer magazines or from the manufacturer or support companies which sell software. Basic interpreters, text editors, assemblers and disassemblers as well as several game and utility programs have all been published and are on the market as well.

One minor problem with software not written by the user is that the program may not reside in a convenient segment of RAM memory. For example, a printer handler may occupy the same memory location as the executive portion of a disassembler (obtained from a different source) requiring a handler for a printer. In order to use the handler to print the results of the disassembly, it will be necessary to have the handler in a usable location.

In the case of a short program like a handler, it would be very easy to re-write the program for a more appropriate place in memory. For longer programs, the program may be re-assembled for a new memory location. This latter option assumes that the user has an assembler program with sufficient memory to accommodate it, and also that the user has a copy of the source program on paper tape or cassette which may be loaded and edited.

Persons having a minimum system, such as the SWTPC 6800 with only 4K of RAM or the MITS 680 with a minimum 1K memory, have insufficient memory to accommodate an assembler/editor. A cassette interface or paper tape read/punch is virtually a necessity for a microcomputer user, but many people may not have one. In addition, some programs are available only in machine language form.

The following program will transfer a block of data or a machine-language program from one location to another and allow it to remain executable in the new location. For example, if the program to be moved is in location 1200-1400 and a particular 3-byte instruction, say at HEX address 1380, is STX $1250 (FF 1250), the instruction will reside at HEX address 0580 after transfer to location 0400-0600, and will be FF 0450 in machine language.

This program was written for the SWTPC 6800. The system used has 12K of memory, a CT-1024 terminal, and AC-30 cassette interface. The SWTPC 6800 has scratchpad memory at HEX addresses A000 to A07F, control interface at HEX locations 8004 to 8007, and MIKBUG ROM monitor occupying the upper 8K of memory above location E000.

The program described here makes use of the scratchpad memory and uses four routines in MIKBUG. There is a provision in the program that 3-byte instructions containing addresses of the interfaces, the scratchpad, or MIKBUG will be transferred with these addresses unchanged so that addresses or routines in these areas may be referenced by the transferred program.

USING THE PROGRAM

The relocator program may be used on machine-language programs resident in RAM at HEX addresses 7FFF or lower (to avoid conflicts with scratchpad, interfaces, or the monitor) or on HEX data resident anywhere in the machine. It will not transfer both in one operation. If a program contains character or HEX data within the body of the program, each block of program or data must be relocated in a separate move. For example, the program may be used to relocate itself but will require two operations, one for the data from HEX locations 0E80 to 0EAF, and a separate one for the program proper located between 0EB0 and 0F56.

To use the program, do the following:

1. Load the program.
2. Use the MIKBUG memory change function to set the program counter, addresses A048 and A049, to 0EB0, the start address of the relocator program.
3. From the MIKBUG monitor, type "G" to begin execution. The program will prompt with a carriage return, line feed, "?", and a space.
4. Type a "P" for program segment or a "D" for data segment to be transferred. The computer will respond with a space.
5. Enter (in hexadecimal) the start address of the program or data block to be transferred, the old end address, and the new start address. The computer will put spaces between each. Transfer is completed when the prompt is re-displayed. Additional segments may be transferred by returning to step 4.

HOW IT WORKS

Hexadecimal data is simply transferred byte-by-byte from the old location to the new.

Program transfers make use of an interesting fact about the 6800 instruction set; the most significant 4 bits of the opcode define the total number of bytes in the instruction. (See Table 1.) If the opcode is represented by a two-character hexadecimal number, say 20 for Branch Always, which is a two-byte instruction, all opcodes beginning with the HEX number two will be two-byte instructions. All opcodes beginning with seven, such as 7E (Jump Extended), are 3-byte instructions where the second and third byte are an address (the Jump address). One-byte instructions are inherent operations and are merely transferred to the new location.

Table 1.

First 4 Bits of OPCODE (HEX)	No. Bytes	Address Mode
0	1	Inherent
1	1	Inherent
2	2	Relative
3	1	Inherent
4	1	Inherent
5	1	Inherent
6	2	Indexed
7	3	Extended
8	2 or 3	Immediate or Relative
9	2	Direct
A	2	Indexed
B	3	Extended
C	2	Immediate
D	2	Direct
E	2	Indexed
F	3	Extended

Two-byte instructions are also transferred byte-by-byte to the new location. The first byte is the opcode, and the second is a relative or direct address which will remain unchanged in the transfer or immediate data, also unchanged.

In the case of 3-byte instructions, the opcode is transferred first. Then the address is tested to see if it is 7FFF or less. If the second two bytes are less than 7FFF, a new address is calculated by adding the difference between the new start address and the old start address to the second two bytes of the instruction. If the second two bytes are 8000 or greater, no new address is calculated. The second two bytes are then transferred.

Note that opcodes beginning with HEX 8, such as 8A (OR

Note that opcodes beginning with HEX 8, such as 8A (ORA A), may be either 2- or 3-byte instructions. The program will test for this and execute the appropriate transfer routine.

Table 2.

Address	Name	Called from ADDR	Purpose
E07E	PDATA1	0EB3	Outputs character string pointed to by index register, terminated by HEX 04.
E1AC	INEEE	0EB6	Inputs ASCII character to A accumulator
E047	BADDR	0EBF 0EC8 0ED1	Inputs 4 HEX characters and stores them in index register
E0CC	OUTS	0EBC 0EC5 0ECE	outputs a space
A002-A003		0EC2 0EDD,0EE0 0EF4 0F0A 0F19 0F27 0F33	Storage location of old start address
A004-A005		0ECB 0F2A	Storage location of old end address
A014-A015		0ED4 0ED&,0EDA 0F1E 0F24	Storage location of new start address
A016-A017		0EE3,0EE6 0F4B,0F4E	Storage location of transfer vector

ADAPTING THE PROGRAM RELOCATOR

Table 2 shows the addresses in scratchpad and MIKBUG and the location of their calls, as well as the purpose of the memory location or subroutine.

If a particular machine does not use MIKBUG, routines must be supplied to take the place of MIKBUG's routines PDATA1, INEEE, BADDR, and OUTS. The calls to these routines should be changed to reflect the location of the routines. If an assembler is available, this is accomplished most easily by changing the EQU statements at the beginning of the programs and re-assembling.

Table 3. System Requirements

NAME:	BLKXFER
FUNCTION:	Move program or data block from one memory location to another, retaining executability of program in new location.
RESULTS:	New program is produced in different memory location.
HARDWARE CONFIGURATION:	SWTPC 6800 microprocessor, CT-1024 TV terminal, AC-30 cassette interface.
MEMORY REQUIRED:	Program resides in $0E80 to $0E56. Scratchpad memory required from $A002 to $A017. Memory also required for program in initial and final storage locations.
SOFTWARE SUPPORT:	MIKBUG
ASSEMBLER:	Motorola Co-Resident Assembler

MIKBUG's scratchpad RAM in location A000 to A07F is also used for temporary storage in BLKXFER. If a machine does not have RAM at this location, temporary storage must be placed in RAM. There is sufficient storage for this purpose in the reserve memory bytes area between 0EA6 and 0EAF.

Lastly, BLKXFER will not recompute addresses higher than 7FFF. This upper limit may be changed to any 256-byte block by changing the data for the compare immediate instruction at HEX address 0F40.□

ASSEMBLY LISTING

```
0001                            NAM     BLKXFER
0002 E07E           PDATAI      EQU     *
0003 E1AC           INEEE       EQU     *
0004 E047           BADDR       EQU     *
0005 E0CC           OUTS        EQU     *

0006                            ORG     $0E80

0007 0E80 0F19      TABLE       FDB     $0F19   TABLE OF OPCODE FIRST HEX
0008 0E82 0F19                  FDB     $0F19   CHARACTER: SUBROUTINE
0009 0E84 0F37                  FDB     $0F37   ADDR FOR DATA TYPE
0010 0E86 0F19                  FDB     $0F19
0011 0E88 0F19                  FDB     $0F19
0012 0E8A 0F19                  FDB     $0F19
0013 0E8C 0F37                  FDB     $0F37
0014 0E8E 0F3C                  FDB     $0F3C
0015 0E90 0F0A                  FDB     $0F0A
0016 0E92 0F37                  FDB     $0F37
0017 0E94 0F37                  FDB     $0F37
0018 0E96 0F3C                  FDB     $0F3C
0019 0E98 0F37                  FDB     $0F37
0020 0E9A 0F37                  FDB     $0F37
0021 0E9C 0F37                  FDB     $0F37
0022 0E9E 0F3C                  FDB     $0F3C
0023 0EA0 0D        CRLF        FCB     $0D     OUTPUT PROMPT
0024 0EA1 0A                    FCB     $0A
0025 0EA2 3F20                  FCC     /? /
0026 0EA4 04                    FCB     $04
0027 0EA5 0008      TYPE        RMB     B       WORK SPACE

0028 0E80 CE 0EA0   START       LDX     #CRLF   PROGRAM START
0029 0E83 BD E07E               JSR     PDATAI  OUTPUT PROMPT
0030 0E86 BD E1AC               JSR     INEEE   INPUT DATA TYPE
0031 0E89 B7 0EA5               STA A   TYPE
0032 0E8C BD E0CC               JSR     OUTS
0033 0E8F BD E047               JSR     BADDR   INPUT OLD START ADDR
0034 0EC2 FF A002   AØ22        STX     $A002
0035 0EC5 BD E0CC               JSR     OUTS
0036 0EC8 BD E047               JSR     BADDR   INPUT OLD END ADDR
0037 0ECB FF A004   AØ24        STX     $A004
0038 0ECE BD E0CC               JSR     OUTS
0039 0ED1 BD E047               JSR     BADDR   INPUT NEW START ADDR
0040 0ED4 FF A014   AØ34        STX     $A014
0041 0ED7 B6 A015   AØ35        LDA A   $A015   COMPUTE TRANSFER VECTOR
0042 0EDA F6 A014   AØ34        LDA B   $A014
0043 0EDD B0 A003   AØ23        SUB A   $A003
0044 0EE0 F2 A002   AØ22        SBC B   $A002
0045 0EE3 B7 A017   AØ37        STA A   $A017
0046 0EE6 F7 A016   AØ36        STA B   $A016
0047 0EE9 F6 0EA5   LOOP        LDA B   TYPE    PROGRAM OR DATA?
0048 0EEC C1 44                 CMP B   #$44    D?
0049 0EEE 27 B6                 BEQ     TYPED   GO TO DATA ROUTINE
0050 0EF0 C1 58                 CMP B   #$58    B?
0051 0EF2 26 BC                 BNE     START   IF NEITHER, START OVER
0052 0EF4 FE A002   AØ22        LDX     #$A002
0053 0EF7 A6 00                 LDA A   0,X     GET MOST SIGNIFICANT 4 BITS
0054 0EF9 44                    LSR A           OF OPCODE
0055 0EFA 44                    LSR A
0056 0EFB 44                    LSR A
0057 0EFC 44                    LSR A
0058 0EFD 48                    ASL A           MULTIPLY BY 2
0059 0EFE B7 0F05               STA A   $0F05   LOOK UP ADDR FOR OPCO   TYPE
0060 0F01 CE 0E80               LDX     #TABLE
0061 0F04 EE 00                 LDX     0,X
0062 0F06 AD 00                 BSR     0,X     GO TO OPCODE TYPE SUBROUTINE
0063 0F08 20 DF     AØ22        BRA     LOOP    NEXT INSTRUCTION
0064 0F0A FE A002   TYPE8       LDX     $A002   OPCODE=8X, SEE IF 2 OR 3 BYTE
0065 0F0D A6 00                 LDA A   0,X     INSTRUCTION
0066 0F0F 81 8C                 CMP A   #$8C
0067 0F11 27 29                 BEQ     TYPE3
0068 0F13 81 8E                 CMP A   #$8E
0069 0F15 27 25                 BEQ     TYPE 3
0070 0F17 20 1E     AØ22        BRA     TYPE2   2 BYTES
0071 0F19 FE A002   TYPE1       LDX     $A002   TRANSFER 1-BYTE INSTRUCTION
0072 0F1C A6 00     AØ34        LDA A   0,X     OR DATA WIHTOUT CHANGE
0073 0F1E FE A014   TYPEA       LDX     $A014
0074 0F21 A7 00                 STA A   0,X
0075 0F23 08                    INX
0076 0F24 FF A014   AØ34        STX     $A014
0077 0F27 FE A002   INCRX       LDX     $A002
0078 0F2A BC A004   AØ22        CPX     $A004   END OF DATA?
0079 0F2D 26 03     AØ24        BNE     END1    NO
0080 0F2F 7E 0E80               JMP     START   YES
0081 0F32 08        END1        INX
0082 0F33 FF A002   AØ22        STX     $A002
0083 0F36 39                    RTS
0084 0F37 8D E0     TYPE2       BSR     TYPE1   TRANSFER 2-BYTE INSTRUCTION
0085 0F39 8D DE                 BSR     TYPE1   WITH NO CHANGE
0086 0F3B 39                    RTS
0087 0F3C 8D DB     TYPE3       BSR     TYPE1   3 BYTES: TRANSFER OPCODE
0088 0F3E A6 00                 LDA A   0,X
0089 0F40 81 7F     -3F ?       CMP A   #$7F    IN RAM?
0090 0F42 23 05                 BLS     MORE    YES
0091 0F44 8D D3                 BSR     TYPE1   NO-TRANSFER ADDR UNCHANGED
0092 0F46 8D D1                 BSR     TYPE1
0093 0F48 39                    RTS
0094 0F49 E6 01     MORE        LDA B   1,X     COMPUTE NEW ADDR
0095 0F4B FB A017   AØ37        ADD B   $A017
0096 0F4E B9 A016               ADC A   $A016
0097 0F51 8D CB     AØ36        BSR     TYPEA   TRANSFER IT
0098 0F53 17                    TBA
0099 0F54 8D C8                 BSR     TYPEA
0100 0F56 39                    RTS
0101                            END
```

OBJECT CODE MIKBUG HEX DUMP

```
P
S1130E800F190F190F370F190F190F190F370F3CBF
S1130E900F0A0F370F370F3C0F370F370F370F3C41
S1130EA00D0A3F2004508D71203F0000000000017
S1130EB0CE0EA05DE07EBDE1ACB70EA5BDE0CCBDBD
S1130EC0E047FFA002BDE0CCBDE047FFA004BDE0C9
S1130ED0CCBDE047FFA014B6A015F6A014B0A00343
S1130EF0F2A002B7A017F7A016F60EA5C14427B6C4
S1130EF0A15026BCFEA002A6004444444448B70F97
S1130F0005CE0E80EE00AD0020DFFEA002A600811B
S1130F108C2729818E2725201EFEA002A600FEA074
S1130F2014A70008FFA014FEA002BCA00426037EA0
S1130F300EB008FFA002398DE08DDE398DDBA600EE
S1130F40817F23058DD38DD139E601FBA017B9A08C
S10A0F50168DCB178DC83983
S9
*
```

4000 → 4007

I 4030

ASSEMBLY LISTING

```
0001                            NAM     BLKXFER
0002 E07E           PDATAI      EQU     *
0003 E1AC           INEEE       EQU     *
0004 E047           BADDR       EQU     *
0005 E0CC           OUTS        EQU     *

0006                            ORG     $0E80

0007 0E80 0F19      TABLE       FDB     $0F19   TABLE OF OPCODE FIRST HEX
0008 0E82 0F19                  FDB     $0F19   CHARACTER: SUBROUTINE
0009 0E84 0F37                  FDB     $0F37   ADDR FOR DATA TYPE
0010 0E86 0F19                  FDB     $0F19
0011 0E88 0F19                  FDB     $0F19
0012 0E8A 0F19                  FDB     $0F19
0013 0E8C 0F37                  FDB     $0F37
0014 0E8E 0F3C                  FDB     $0F3C
0015 0E90 0F0A                  FDB     $0F0A
0016 0E92 0F37                  FDB     $0F37
0017 0E94 0F37                  FDB     $0F37
0018 0E96 0F3C                  FDB     $0F3C
0019 0E98 0F37                  FDB     $0F37
0020 0E9A 0F37                  FDB     $0F37
0021 0E9C 0F37                  FDB     $0F37
0022 0E9E 0F3C                  FDB     $0F3C
0023 0EA0 0D        CRLF        FCB     $0D     OUTPUT PROMPT
0024 0EA1 0A                    FCB     $0A
0025 0EA2 3F20                  FCC     /? /
0026 0EA4 04                    FCB     $04
0027 0EA5 000B      TYPE        RMB      B      WORK SPACE

0028 0EB0 CE 0EA0   START       LDX     #CRLF   PROGRAM START
0029 0EB3 BD E07E               JSR     PDATAI  OUTPUT PROMPT
0030 0EB6 BD E1AC               JSR     INEEE   INPUT DATA TYPE
0031 0EB9 B7 0EA5               STA A   TYPE
0032 0EBC BD E0CC               JSR     OUTS
0033 0EBF BD E047               JSR     BADDR   INPUT OLD START ADDR
0034 0EC2 FF A002               STX     $A002
0035 0EC5 BD E0CC               JSR     OUTS
0036 0EC8 BD E047               JSR     BADDR   INPUT OLD END ADDR
0037 0ECB FF A004               STX     $A004
0038 0ECE BD E0CC               JSR     OUTS
0039 0ED1 BD E047               JSR     BADDR   INPUT NEW START ADDR
0040 0ED4 FF A014               STX     $A014
0041 0ED7 B6 A015               LDA A   $A015   COMPUTE TRANSFER VECTOR
0042 0EDA F6 A014               LDA B   $A014
0043 0EDD B0 A003               SUB A   $A003
0044 0EE0 F2 A002               SBC B   $A002
0045 0EE3 B7 A017               STA A   $A017
0046 0EE6 F7 A016               STA B   $A016
0047 0EE9 F6 0EA5   LOOP        LDA B   TYPE    PROGRAM OR DATA?
0048 0EEC C1 44                 CMP B   #$44    D?
0049 0EEE 27 B6                 BEQ     TYPED   GO TO DATA ROUTINE
0050 0EF0 C1 58                 CMP B   #$58    B?
0051 0EF2 26 BC                 BNE     START   IF NEITHER, START OVER
0052 0EF4 FE A002               LDX     #$A002
0053 0EF7 A6 00                 LDA A   0,X     GET MOST SIGNIFICANT 4 BITS
0054 0EF9 44                    LSR A           OF OPCODE
0055 0EFA 44                    LSR A
0056 0EFB 44                    LSR A
0057 0EFC 44                    LSR A
0058 0EFD 48                    ASL A           MULTIPLY BY 2
0059 0EFE B7 0F05               STA A   $0F05   LOOK UP ADDR FOR OPCO TYPE
0060 0F01 CE 0E80               LDX     #TABLE
0061 0F04 EE 00                 LDX     0,X
0062 0F06 AD 00                 BSR     0,X     GO TO OPCODE TYPE SUBROUTINE
0063 0F08 20 DF                 BRA     LOOP    NEXT INSTRUCTION
0064 0F0A FE A002   TYPE8       LDX     $A002   OPCODE=8X, SEE IF 2 OR 3 BYTE
0065 0F0D A6 00                 LDA A   0,X     INSTRUCTION
0066 0F0F 81 8C                 CMP A   #$8C
0067 0F11 27 29                 BEQ     TYPE3
0068 0F13 81 8E                 CMP A   #$8E
0069 0F15 27 25                 BEQ     TYPE 3
0070 0F17 20 1E                 BRA     TYPE2   2 BYTES
0071 0F19 FE A002   TYPE1       LDX     $A002   TRANSFER 1-BYTE INSTRUCTION
0072 0F1C A6 00                 LDA A   0,X     OR DATA WIHTOUT CHANGE
0073 0F1E FE A014   TYPEA       LDX     $A014
0074 0F21 A7 00                 STA A   0,X
0075 0F23 08                    INX
0076 0F24 FF A014               STX     $A014
0077 0F27 FE A002   INCRX       LDX     $A002
0078 0F2A BC A004               CPX     $A004   END OF DATA?
0079 0F2D 26 03                 BNE     END1    NO
0080 0F2F 7E 0EB0               JMP     START   YES
0081 0F32 08        END1        INX
0082 0F33 FF A002               STX     $A002
0083 0F36 39                    RTS
0084 0F37 8D E0     TYPE2       BSR     TYPE1   TRANSFER 2-BYTE INSTRUCTION
0085 0F39 8D DE                 BSR     TYPE1   WITH NO CHANGE
0086 0F3B 39                    RTS
0087 0F3C 8D DB     TYPE3       BSR     TYPE1   3 BYTES: TRANSFER OPCODE
0088 0F3E A6 00                 LDA A   0,X
0089 0F40 81 7F                 CMP A   #$7F    IN RAM?
0090 0F42 23 05                 BLS     MORE    YES
0091 0F44 8D D3                 BSR     TYPE1   NO-TRANSFER ADDR UNCHANGED
0092 0F46 8D D1                 BSR     TYPE1
0093 0F48 39                    RTS
0094 0F49 E6 01     MORE        LDA B   1,X     COMPUTE NEW ADDR
0095 0F4B FB A017               ADD B   $A017
0096 0F4E B9 A016               ADC A   $A016
0097 0F51 8D CB                 BSR     TYPEA   TRANSFER IT
0098 0F53 17                    TBA
0099 0F54 8D C8                 BSR     TYPEA
0100 0F56 39                    RTS
0101                            END
```

OBJECT CODE MIKBUG HEX DUMP

```
P
S1130E800F190F190F370F190F190F190F370F3CBF
S1130E900F0A0F370F370F3C0F370F370F370F3C41
S1130EA00D0A3F2004508D71203F00000000000017
S1130EB0CE0EA05DE07EBDE1ACB70EA5BDE0CCBDBD
S1130EC0E047FFA002BDE0CCBDE047FFA004BDE0C9
S1130ED0CCBDE047FFA014B6A015F6A014B0A00343
S1130EF0F2A002B7A017F7A016F60EA5C14427B6C4
S1130EF0A15026BCFEA002A6004444444448B70F97
S1130F0005CE0E80EE00AD0020DFFEA002A600811B
S1130F108C2729818E2725201EFEA002A600FEA074
S1130F2014A70008FFA014FEA002BCA00426037EA0
S1130F300EB008FFA002398DE08DDE398DDBA600EE
S1130F40817F23058DD38DD139E601FBA017B9A08C
S10A0F50168DCB178DC83983
S9
*
```


φE80
0F60