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           PDATA1      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     PDATA1  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 50                 CMP B   #$50    P?
0051 0EF2 26 BC                 BNE     START   IF NEITHER, START OVER
0052 0EF4 FE A002               LDX     #$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 OPCODE 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 E8     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
S1130E80F19F19F19F37F19F19F19F37F3CBF
S1130E90F0AF37F37F3CF37F37F37F3C41
S1130EA0D0A3F2004508D71203F00000000000017
S1130EB0CE0EA0BDE07EBDE1ACB70EA5BDE0CCBDBD
S1130EC0E047FFA002BDE0CCBDE047FFA004BDE0C9
S1130ED0CCBDE047FFA014B6A015F6A014B0A00343
S1130EF0F2A002B7A017F7A016F60EA5C14427B6C4
S1130EF0A15026BCFEA002A600444444444B70F97
S1130F0005CE0E80EE00AD0020DFFEA002A6008118
S1130F108C27298182752801EFEA002A600FEA074
S1130F2014A70008FFA014FEA002BCA004260 37EA0
S1130F300EB008FFA002398DE08DDE398DDBA600EE
S1130F40817F23058DD38DD139E601FBA017B9A08C
S10A0F50168DCB178DC83983
S9
*
```