1976

A Simulator for the Motorola M6800 Microprocessor

Carolyn Elizabeth Jordan
University of Central Florida

A SIMULATOR FOR THE MOTOROLA M6800 MICROPROCESSOR

BY

CAROLYN ELIZABETH JORDAN
B.S., Florida Technological University, 1974

RESEARCH REPORT

Submitted in partial fulfillment of the requirements
for the degree of Master of Science
in the Graduate Studies Program of
the College of Engineering of
Florida Technological University

Orlando, Florida
1976

A SIMULATOR FOR THE MOTOROLA M6800 MICROPROCESSOR

by

Carolyn E. Jordan

ABSTRACT

The Motorola Company has developed a microprocessor called the M6800 Microprocessor. While the microprocessor is being configured, it is general practice to develop the software at the same time. This is where simulation of the proposed hardware operation can become very important to the success of the design effort. The simulator duplicates the microprocessor execution of machine language instructions on another computer.

The simulator discussed in this paper is denoted the SIM6800. The purpose of this paper is to describe the structure, coding, and execution of the SIM6800 simulator. A User's Guide and sample program have been included.

Christian S. Bacon, PE
Director of Research Report

TABLE OF CONTENTS

INTRODUCTION

A microprocessor is fundamentally no different from any other computer in that the five major subsystems are present, including: the ALU (arithmetic logic unit), the CPU (control function), input, output, and memory. A microprocessor is constructed by putting the ALU and the CPU into one or a small number of integrated circuit chips. The Motorola Company has developed a processor of this type called the M6800 Microprocessor.

While the microprocessor hardware is being configured, it is general practice to develop the software at the same time. This is where simulation of the proposed hardware operation can become very important to the success of a design effort. A simulator program facilitates overall checkout and error elimination from programs written for the microprocessor. The simulator duplicates the microprocessor execution of machine language instructions on another computer. Although the simulator program does not match the microprocessor's real time operating speeds, a count is kept of elapsed time cycles in simulated execution which can be used to estimate program execution times. The simulator program discussed in this paper was designed and written to operate in a time-sharing mode. Choosing the time-sharing mode over the batch mode allows the user to

have complete interactive control over the program being tested. The user gives appropriate commands to the simulator, resulting in execution of all or any part of the program being tested. This flexibility of user control would be difficult to obtain using only the microprocessor hardware.

The simulator is only part of the overall software support aids for the Motorola M6800 (Figures 1 and 2). The microprocessor users begin by writing a program in the assembler language for the computer. The assembler language program serves as input to a Cross-assembler, an assembler resident on a computer other than the computer for which it generates the machine code. The machine code from the Cross-assembler is then used as input to the simulator program. The simulator duplicates the execution of the machine language. The output from the simulator is a computer activity listing which the user can review to determine if the assembler language program has executed correctly with the intended results.

The original purpose of the simulator was to allow parallel work on the microprocessor software and hardware. The configuration of the microprocessor hardware allows only one user at a time. However, the host computer containing the simulator is usually large enough to support several terminals allowing several users of the simulator software at once. With this increased access to the

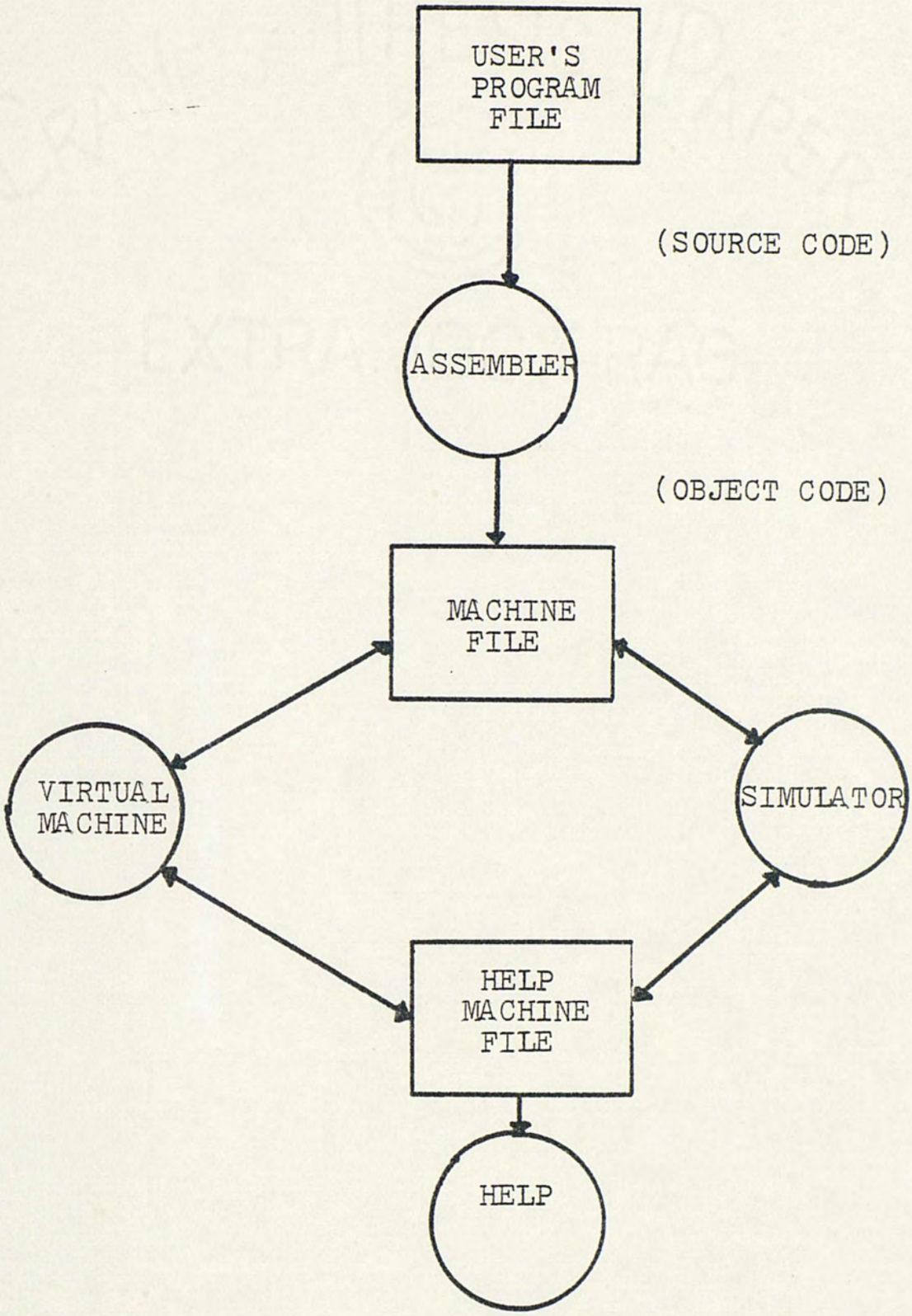

Fig. 1.--FILE INTERCONNECT SCHEME FOR SOFTWARE SUPPORT

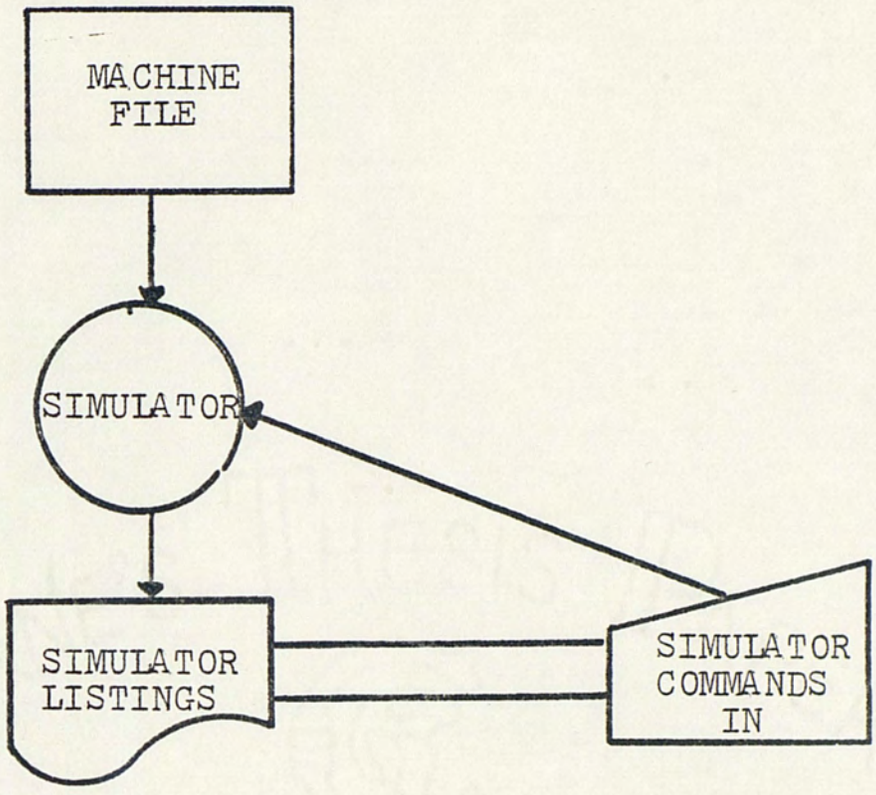

Fig. 2.--SIMULATOR, A BLOCK DIAGRAM

simulator, the use of the simulator can later be expanded to general debugging of all microprocessor software.

The simulator discussed in this paper, denoted the SIM6800, was written in IBM PL/1-F for the IBM 360-75 computer. PL/1 lends itself to this type of programming application due to the structural format of the language, the character string and bit string manipulation abilities, and the access methods to the machine's bits and addresses. The simulator program can be executed in a minimum core partition of 120K-bytes under time-sharing.

The purpose of this report is to describe the structure, coding, and execution of the SIM6800 simulator. A User's Guide and sample program have been included to allow any individual to use the SIM6800 simulator with ease.

CHAPTER 1

SIM6800 STRUCTURE

The input to the simulator is the result of several previous steps in the software support scheme. The user written assembler language program (Figure 3), the assembled program (Figure 4), and the object code file, named EXAMPLE 1 (Figure 5) have all been generated prior to the execution of the simulator.

SIM6800 itself is a single program written in PL/1. All subprocedures are contained internally to the main program, but the simulator does require three files for it to execute. These three files are:

1. Object code file (OBJ)
2. Machine operation table (MOT)
3. Simulator commands

The name of the OBJ file is supplied by the user when dialog to SIM6800 is initiated through an interactive terminal. The procedure call for establishing communication is as follows:

```
exec sim6800(example1)
```

This statement moves the object file EXAMPLE 1 into simulated memory. Each line of the object code symbolically represents each line of the assembler program and is divided into two components:

```
100  NAM ITEM1
110  OPT MEM DIRECTS THE ASSEMBLER TO SAVE
120 *       AN OBJECT PROGRAM.
130 *
140 * ADDITION OF TWO EIGHT-BYTE
150 * BINARY-CODED-DECIMAL NUMBERS.
160 *
190  ORG $1000
200  LDA B #8
210  LDA #P  LOADS INDEX REGISTER
220 *        WITH THE ADDRESS OF THE
230 *        MOST SIGNIFICANT BYTE
240 *        OF P
250  CLC
260 NEXT LDA A 7,X
270  ADC A 15,X
280  DAA
290  STA A 23,X
300  DEX
310  DEC B
320  BNE NEXT   LOOPS BACK FOR NEXT
330 *           BYTE IF ADDITION NOT COMPLETED.
400  NOP
410  BRA *-1
500 * ALLOCATE A DATA AREA IN READ-WRITE
510 * MEMORY FOR THE NUMBERS TO BE ADDED
520 * (P AND Q) AND FOR THE SUM (RES)
530  ORG $0100
540 P RMB 8
550 Q RMB 8
560 RES RMB 8
600   END
610   MON
```

Fig. 3.--ASSEMBLER LANGUAGE PROGRAM

```
00010                          NAM    ITEM2
00020                  * REVISION 1.0
00030                          OPT    MEM
00040                          OPT    SYMBOLS

00060                  * ADDITION OF TWO-MULTIPLE-PRECISION
00070                  * BINARY-CODED-DECIMAL NUMBERS.
00080                  *
00090        0008      NB     EQU    8          SPECIFIES 8-BYTE
                       *                        OPERANDS

0110                   * BEGIN SUBROUTINE
00120 1000                     ORG    $1000
00130 1000 C6 08       BCD     LDA B  #NB
00140 1002 FE 0100             LDX    ADDR       LOADS DATA ADDR
00150 1005 0C                  CLC
00160 1006 A6 07       NEXT    LDA A  NB-1,X     START LOOP
00170 1008 A9 0F               ADC A  2*NB-1,X
00180 100A 19                  DAA
00190 100B A7 17               STA A  3*NB-1,X
00200 100D 09                  DEX
00210 100E 5A                  DEC B
00220 100F 26 F5               BNE    NEXT
00230 1011 39                  RTS                END OF BCD SUBROU
                       *                        TINE
00250                  * BEGIN MAIN PROGRAM
00260                  * TEST OF SUBROUTINE 'BCD'
00270                          ORG    $1100
00280                          LDS    #$13F      INITIALIZE STACK
                       *                        POINTER
00290                          LDX    #P         LOADS ADDRESS
                       *                        OF P
00300                          STX    ADDR
00310                          JSR    BCD        DO BCD ADDITION
00320                          NOP
00330                          BRA    *          END OF MAIN PROG

00350                  * ALLOCATE A DATA AREA IN READ-WRITE MEM
00360                          ORG    $0100
00370                  *   (1) FOR THE SUBROUTINE
00380                  ADDR    RMB    2
00390                  *   (2) FOR THE MAIN PROGRAM
00400                  P       RMB    NB
00410                  Q       RMB    NB
00420                  RES     RMB    NB
00430                          END
```

Fig. 4.--ASSEMBLED PROGRAM

```
4364 001 0
4365 032 254
4361 189 4096
4358 255 256
4355 206 258
4352 142 319
4096 198 8
4098 254 256
4101 012 0
4102 166 7
4104 169 15
4106 025 0
4107 167 23
4109 009 0
4110 090 0
4111 038 245
4113 057 0
```

Fig. 5.--OBJECT CODE FILE EXAMPLE1

Component	PL/1 Variable Names
1. Instructional address	BYTE1
2. Operands	BYTE2

The second file, MOT, is automatically moved into a storage structure called MOT. The MOT file remains the same for all programs being tested. This file contains information needed to decode and execute the object code. There are 197 entries in the MOT file and each entry contains the following categories:

MOT Entry	PL/1 Variable Names
1. Operator number	OP
2. Mnemonic code	MNEMONIC
3. Address mode	ADDR_MODE
4. Time	TIME
5. Bytes	BYTES
6. MOT number	MOT_NO
7. Address accumulator	ADDR_ACC

Once these first two files are set up, SIM6800 will print the following header:

SIMULATOR FOR THE MOTOROLA M6800 MICROPROCESSOR

WRITTEN FOR THE IBM360

VERSION 1 1976

The simulator acknowledges a request for a command by printing a "?". At this time the user creates the third file by entering a simulator command after every prompt. The simulator commands now available are:

MNEMONIC	MEANING
DM	Display memory
DR	Display register
EX	Exit simulator
HR	Set header count
RN	Run
SM	Set memory
SR	Set register

A complete explanation of these commands may be found in Appendix A.

The SIM6800 has 72 executable instructions. These instructions assemble into 1 to 3 bytes of object code. The length of the instruction depends on the particular instruction and on the type of addressing used. The length of the instruction is the value of BYTES in the MOT file.

SIM6800 has two 8-bit data registers where arithmetic calculations are performed. These are equivalent to Accumulator A and Accumulator B in the M6800 processor chip. The corresponding PL/1 variable names are ACCA and ACCB, respectively.

SIM6800 has seven types of addressing, as follows:

1. Inherent addressing
2. Relative addressing
3. Immediate addressing
4. Accumulator addressing
5. Indexed addressing

6. Extended addressing

7. Direct addressing

The addressing mode used is coded 1 through 7 and this code is entered in the MOT variable ADDR_MODE. If either Accumulator A or Accumulator B is needed in the addressing, an "A" or a "B" is placed in MOT entry ADDR_ACC.

SIM6800 uses a 16-bit register called the Index Register (IX), again, corresponding to a hardware register on the M6800 chip. This register is needed for instructions that use the indexed addressing mode. The use of IX allows the current instruction or data address to be computed during execution of the program and allows additional flexibility over the strict use of fixed addresses predetermined by the assembler.

The SIM6800 simulated memory is a two dimensional array, denoted M(5000,2). The size of memory is 10,000 8-bit memory locations. The design of M as a 2-dimensional array is based on the format of the object code for the M6800 hardware. The maximum length of a coded instruction is 3 bytes. The first byte is the memory address which will be the array M subscript. Therefore, only two dimensions of array M are needed to store the last two bytes of the instructions.

The memory location of the current instruction about to be executed is assigned to the program counter variable,

PC. That memory location is also assigned to the instruction address, IA. Once the length of the instruction has been determined, PC is incremented by that length. This new value of PC is the memory location of the next instruction to be executed.

Like most microprocessors, the Motorola M6800 is not designed to handle the conventional subroutine return-address storage scheme. Rather, the M6800 uses a hardware "push-down stack." The stack consists of any number of locations in memory providing for temporary storage and retrieval of successive bytes of information. Usually, the stacks will be one single block of successive memory locations but there could conceivably be several stacks, each consisting of a block of successive memory locations. Associated with the stack is a 16-bit stack pointer called SP. When a subroutine is called, the current contents of PC (return address) are stored on the stack at location SP. The stack pointer SP is then decremented by 2. When a return from subroutine occurs, the current contents at stack location SP (entry address) is assigned to PC. The stack pointer SP is incremented by 2.

The Condition Code Register, CC, is a 6-bit register containing information on the status of the program being run. The 6 bits in CC are:

Bit No.	Condition Code
0	C (Carry-borrow)
1	V (Overflow)
2	Z (Zero)
3	N (Negative)
4	I (Interrupt Mask)
5	H (Half-carry)

The CC can be set by any of a number of instructions. The value to which CC is set depends on the result of the last instruction executed. If, for example, the subtraction of two numbers results in a negative number, the N bit would be turned on (set to "1"). The value of the CC register can be used to control branching within a program, and is useful in implementing data comparison operations and similar program functions.

CHAPTER 2

SIM6800 CODING

The simulator program is divided into 10 subprocedures contained within a main procedure. Each subprocedure performs one step in the decoding of the object code. This modular approach aids in debugging and future modification to the simulator. Figure 6 is an overall flowchart of the SIM6800.

When the simulator is executed, the first two subprocedures CREATE1 and CREATE2 are called. Their function is to load the OBJ and MOT files already discussed. Before executing any further, the simulator prompts the user for a command. The user-supplied commands EX, SM, SR, DM, DR, and HR either set variable values or end execution of the simulator. The RN command causes all of the subprocedures to be executed.

A brief, functional description of each of these subprocedures follows.

LKUP

BYTE1 of the object code identifying the instruction should match with an entry in the MOT file if the instruction is valid. If the match is found, all entries in the MOT file structure are filled in. At this point, the simulator should have information available to it

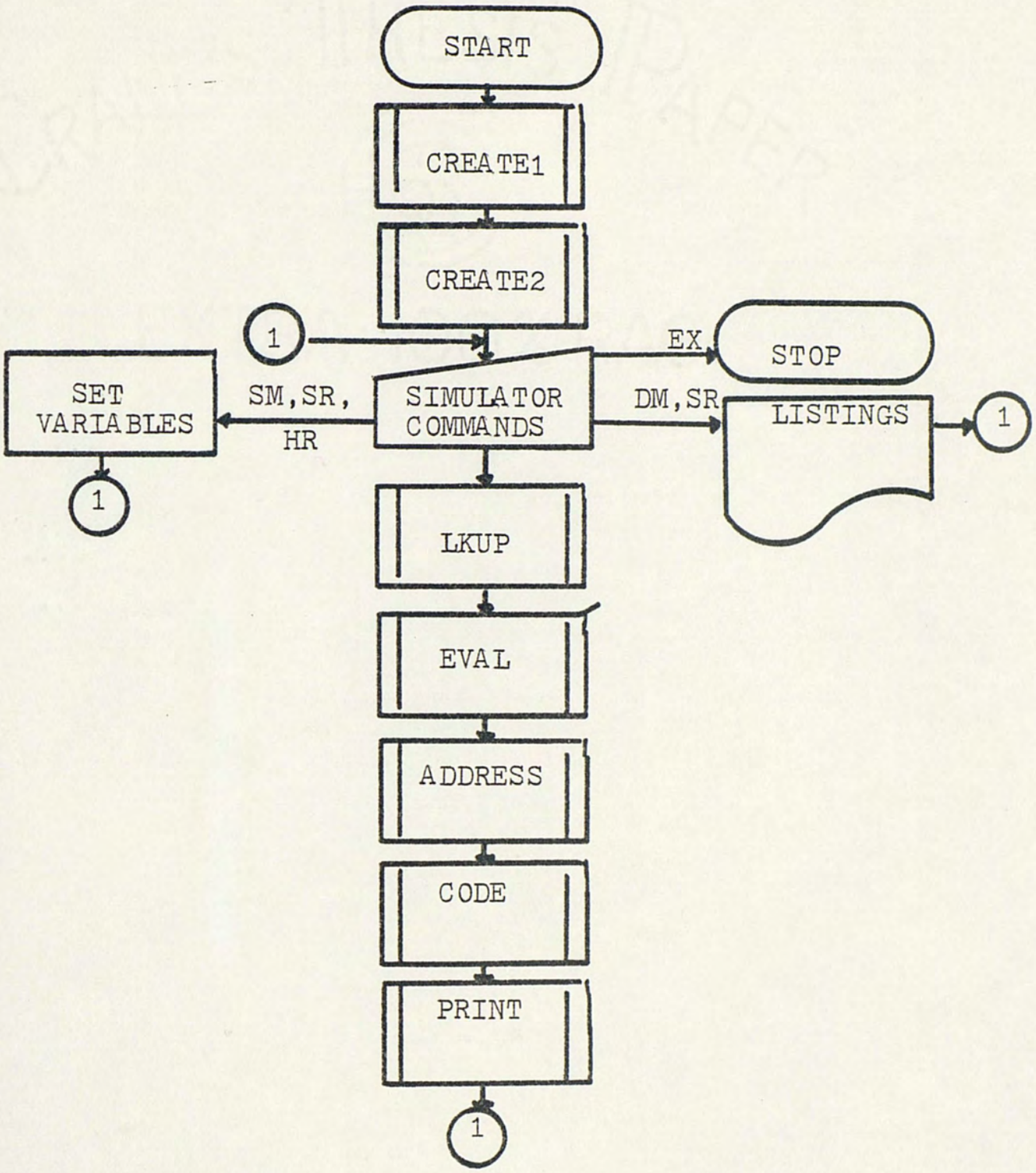

Fig. 6.--SIM6800 FLOWCHART

concerning the instruction abbreviation (MNEMONIC), the type of address used (ADDR_MODE), and number of execution cycles needed (TIME), length of the instruction (BYTES), the instruction number from 1 to 72 (MOT_NO), and whether or not an accumulator is needed in the addressing (ADDR_MODE). If a match is not found, Error #301 will be printed on the terminal to flag the problem (see Appendix A for a list of error codes and corrective actions).

ADDRESS

The variable ADDR_MODE in the MOT file determines which of the seven types of addressing is required. The address of the instruction operands is calculated in this subprocedure to be used in another procedure. If the calculated address exceeds the bounds of simulated memory, Error #313 will be printed (see Appendix A).

EVAL

The operation number OP from the OBJ file determines what type of operation the instruction is to perform (e.g. ADD). Using the address generated by ADDRESS, the operands needed in the operation are obtained and the instruction can then be completely evaluated.

CODE

The results of the operation identified in EVAL are now tested. The CC bits are set to a specific value based on the particular instruction. The Boolean formulae used to test the results are the same as those used by Motorola

in the hardware operation.

PRINT

After each instruction is evaluated, the values of the simulated machine registers are printed along with the register abbreviation as a header.

CONCLUSIONS AND SUGGESTIONS FOR FUTURE WORK

The SIM6800 shows how a simulator can be used to verify a microprocessor program without the use of the microprocessor hardware. The PL/1 simulator program was written with "ease of use" in mind and was written for a wide range of users, from the less experienced to the experienced. The SIM6800 also shows execution of the programs in various amounts of detail as required by the user. Two additional concepts, MACRO commands and HELP files are suggested as future work to the SIM6800. The following is an overall description of these concepts.

MACRO COMMANDS

A user defined simulator command(s) is called a MACRO. MACRO commands allow the user to create a string of commands or redefine existing commands. Suppose the user repeatedly enters:

```
RN  7
DM  0102  8
DM  010A  8
DM  0112  8
```

To avoid repetition, the user could define a MACRO command named XY by typing

```
? XY(RN 7.DM 0102 8.DM 010A 8.DM 0112 8)
```

A MACRO command is executed in the same way as an existing command.

?XY

Once the MACRO concept has been implemented, it can be expanded to include MACROs with variable parameters and MACROs with multiple variable parameters.

Commands would then be needed to give the user control over MACRO commands. These would be:

MACRO Command	Meaning
MD	Delete MACRO
ML	List MACRO
MR	Read MACRO from machine file
MS	Save MACRO in machine file

HELP FILES

Once the entire M6800 software and hardware have been assembled, a HELP system should be created. The HELP system provides up-to-date information on improvements and new developments of the M6800 microprocessor system. The users enter a request for assistance and a search is made through the HELP file for the desired information. The HELP file is a set of messages, each divided into three sections:

1. the message number
2. the message description
3. the message text

The amount of detail to be printed with the HELP message is specified by format control commands. With the added

feature of format control, the user could print all messages added within the last month, all simulator messages, all HELP messages, or any combination of messages desired.

The inclusion of MACROs and HELP files will round out user control and interface with SIM6800.

APPENDIX A

SIM6800 USER'S GUIDE

FOR THE MOTOROLA M6800 MICROPROCESSOR

EXECUTING THE SIM6800 SIMULATOR

The SIM6800 simulator is designed to be used in a time-sharing mode. Since every computer installation has its own form of time-sharing, procedures may vary. The sample problem included in this appendix and Appendix B was executed on an IBM2741 terminal to an IBM360/75 computer operating under OS/360. To execute the SIM6800 simulator, the user logs on the computer and types in:

```
exec sim6800(example1)
```

(EXAMPLE1 is the file containing the object code of the program being simulated.)

SIMULATOR COMMANDS

The following is a list of the currently available M6800 simulator commands:

MNEMONIC	MEANING
DM	Display memory
DR	Display registers
EX	Exit simulator
HR	Set header count
RN	Run
SM	Set memory
SR	Set registers

The exact format and explanation of each of these commands are found on the following pages. There are several basic directives which must be observed when executing SIM6800.

1. Insert at least one blank between operands in the command.
2. Hexidecimal values must contain four positions. Leading zeros must be added if needed.
3. Real number values must not contain a decimal point.
4. All operands listed in the command format must be included.

5. All memory locations (addresses) must be hexi-
decimal and other operands must be integers
(unless otherwise stated).

DM--DISPLAY MEMORY

Format

DM a n

Display n memory positions starting at memory address a.

The DM command can be used to display n memory locations containing data values or n memory locations containing the object code of the program being simulated.

DR--DISPLAY REGISTERS

Format

DR

The DR command causes the value of the registers to be printed.

The codes for the registers to be displayed are:

IA -- Instruction address

OC -- Operation code

EA -- Effective address

P -- Program counter

X -- Index register

A -- Accumulator A

B -- Accumulator B

C -- Condition codes

S -- Stack pointer

T -- Time

Note: This simulator command has no operands.

EX--EXIT FROM SIMULATOR

Format

EX

The EX command causes all simulation to stop. The only file saved is the machine program file.

After entering the EX command, the user will return to an idle state under TSO.

Note: This simulator has no operands.

HR--SET HEADER COUNT

Format

HR n

The HR command controls the number of times the header line will be printed. The header will print every n lines of output.

RN--RUN n INSTRUCTIONS

Format

RN n

The RN command causes the simulator to execute the next n instructions of the machine program file.

Simulation (execution) continues until:

1. n instructions have been executed
2. an error occurs.

SM--SET MEMORY (CHANGE MEMORY)

Format

SM a n v_1 v_2 . . . . v_n

Starting at memory address a, set n memory positions with the values v_1, v_2, . . . v_n.

The SM command can be used to enter data before execution or can be used to modify the actual machine program file.

SR--SET REGISTERS

Format

SR r v

The SR command sets register r to the value v.

The codes for the r operand with corresponding v input bases are:

r	v
P -- Program counter	Hexidecimal
X -- Index register	Hexidecimal
A -- Accumulator A	Integer
B -- Accumulator B	Integer
C -- Condition codes	Bit-string*
S -- Stack pointer	Hexidecimal
T -- Time	Integer

* To turn all condition codes "on" enter '111111'B.
To turn all condition codes "off" enter '000000'B.

ERROR MESSAGES

Error Number	Description
301	**** ERROR 301 AAAA Undefined simulator command.
302	**** ERROR 302 AAAA Possible syntax error in simulator command.
306	**** ERROR 306 AAAA The operand in the simulator command caused a register overflow.
313	**** ERROR 313 NNNN The calculated address beyond bounds of memory.

AAAA -- The simulator address which caused the error.

NNNN -- A numeric field.

APPENDIX B

SAMPLE OUTPUT FROM THE SIM6800 SIMULATOR PROGRAM

The M6800 program being tested is one designed to add the following two 8-byte binary coded decimal numbers:

```
  1357902468097531
+ 9258147036741852
  0616049504839383  (result)
```

The object code exists in a file named "EXAMPLE1". The simulator was started by the statement:

```
exec sim6800(example1)
```

The SR command was used to initialize the program counter with the hexidecimal value 1100. This is the address in memory of the first instruction.

The SM command was used to enter the data values to be added.

A repetition of the RN and DM commands were used to execute the instructions and to display the results.

```
exec sim6800(example1)
```



```
     SIMULATOR FOR THE MOTO DLA M6800 MICROPROCESSOR
               WRITTEN FOR THE IBM 360
               VERSION 1              1976

?
sm 0102 8 13 57 90 24 68 99 75 31

?
sm 010a 8 92 58 14 70 36 74 18 52

?
sm 0112 8 0 0 0 0 0 0 0 0

?
sr p 1100

?
hr 7

?
rn 7

IA     OC        EA     P      X        A      B      C      S      T
1100   LDS       1102 1103    0000       0      0 00N000 013F      3
1103   LDX       1105 1106    0102       0      0 00N000 013F      6
1106   STX       0101 1109    0102       0      0 00N000 013F     12
1109   JSR       013E 1000    0102       0      0 00N000 013D     21
1000   LDA   B   1001 1002    0102       0      8 000000 013D     23
1002   LDX       0101 1005    0102       0      8 00N000 013D     28
1005   CLC       1005 1006    0102       0      8 00N000 013D     30

?
rn 7

IA     OC        EA     P      X        A      B      C      S      T
1006   LDA   A   0109 1008    0102      31      8 000000 013D     35
1008   ADC   A   0111 100A    0102      83      8 00N0V0 013D     40
100A   DAA       100A 100B    0102      83      8 00N000 013D     42
100B   STA   A   0119 100D    0102      83      8 00N000 013D     48
100D   DEX       100D 100E    0101      83      8 00N000 013D     52
100E   DEC   B   100E 100F    0101      83      7 000000 013D     54
100F   BNE       1010 1006    0101      83      7 000000 013D     58

?
```

```
dm 0102 8                                  37
   13   57   90   24   68    9   75   31
?
dm 010a 8
   92   58   14   70   36   74   18   52
?
dm 0112 8
    0    0    0    0    0    0    0   83
?
rn 7

IA     OC       EA     P      X       A      B      C      S      T
1006   LDA  A   0108 1008   0101     75      7 000000 013D     63
1008   ADC  A   0110 100A   0101     93      7 00N0V0 013D     68
100A   DAA      100A 100B   0101     93      7 00N000 013D     70
100B   STA  A   0118 100D   0101     93      7 00N000 013D     76
100D   DEX      100D 100E   0100     93      7 00N000 013D     80
100E   DEC  B   100E 100F   0100     93      6 000000 013D     82
100F   BNE      1010 1006   0100     93      6 000000 013D     86

?
dm 0102 8
   13   57   90   24   68    9   75   31
?
dm 010a 8
   92   58   14   70   36   74   18   52
?
dm 0112 8
    0    0    0    0    0    0   93   83
?
rn 7

IA     OC       EA     P      X       A      B      C      S      T
1006   LDA  A   0107 1008   0100      9      6 000000 013D     91
1008   ADC  A   010F 100A   0100     83      6 000000 013D     96
100A   DAA      100A 100B   0100     83      6 00N000 013D     98
100B   STA  A   0117 100D   0100     83      6 00N000 013D    104
100D   DEX      100D 100E   00FF     83      6 00N000 013D    108
100E   DEC  B   100F 100F   00FF     83      5 000000 013D    110
100F   BNE      1010 1006   00FF     83      5 000000 013D    114

?
dm 0102 8
   13   57   90   24   68    9   75   31
?
dm 010a 8
   92   58   14   70   36   74   18   52
?
dm 0112 8
    0    0    0    0    0   83   93   83
?
```

```
rn 7

IA     OC        EA     P       X        A      B      C       S       T
1006   LDA  A    0106  1008    00FF      68      5  000000  013D     119
1008   ADC  A    010E  100A    00FF     104      5  00N0V0  013D     124
100A   DAA       100A  100B    00FF       4      5  00000C  013D     126
100B   STA  A    0116  100D    00FF       4      5  00000C  013D     132
100D   DEX       100D  100E    00FE       4      5  00000C  013D     136
100E   DEC  B    100E  100F    00FE       4      4  00000C  013D     130
100F   BNE       1010  1006    00FE       4      4  00000C  013D     142

?
dm 0102 8
   13   57   90   24   68    9   75   31
?
dm 010a 8
   92   58   14   70   36   74   18   52
?
dm 0112 8
    0    0    0    0    4   83   93   85
?
rn 7

IA     OC        EA     P       X        A      B      C       S       T
1006   LDA  A    0105  1008    00FE      24      4  00000C  013D     147
1008   ADC  A    010D  100A    00FE      95      4  00N0V0  013D     152
100A   DAA       100A  100B    00FE      95      4  00N0CC  013D     154
100B   STA  A    0115  100D    00FE      95      4  00N000  013D     160
100D   DEX       100D  100E    00FD      95      4  00N000  013D     164
100E   DEC  B    100E  100F    00FD      95      3  000000  013D     160
100F   BNE       1010  1006    00FD      95      3  000000  013D     170

?
dm 0102 8
   13   57   90   24   68    9   75   31
?
dm 010a 8
   92   58   14   70   36   74   18   52
?
dm 0112 8
    0    0    0   95    4   83   93   83
?
rn 7

IA     OC        EA     P       X        A      B      C       S       T
1006   LDA  A    0104  1008    00FD      90      3  00N000  013D     175
1008   ADC  A    010C  100A    00FD     104      3  00N000  013D     180
100A   DAA       100A  100B    00FD       4      3  00000C  013D     182
100B   STA  A    0114  100D    00FD       4      3  00000C  013D     188
100D   DEX       100D  100E    00FC       4      3  00000C  013D     192
100E   DEC  B    100E  100F    00FC       4      2  00000C  013D     194
100F   BNE       1010  1006    00FC       4      2  00000C  013D     198

?
```

```
dm 0102 8
   13   57   90   24   68    9   75   31
?
dm 010a 8
   92   58   14   70   36   74   18   52
?
dm 0112 8
    0    0    4   95    4   83   93   83
?
rn 7

IA    OC       EA    P      X       A     B     C       S       T
1006  LDA  A   0103  1008   00FC     57     2  000000C  013D    203
1008  ADC  A   010B  100A   00FC    116     2  H0N0V0   013D    208
100A  DAA      100A  100B   00FC     16     2  H0000C   013D    210
100B  STA  A   0113  100D   00FC     16     2  H0000C   013D    216
100D  DEX      100D  100E   00FB     16     2  H0000C   013D    220
100E  DEC  B   100E  100F   00F3     16     1  H0000C   013D    222
100F  BNE      1010  1006   00FB     16     1  H0000C   013D    226

?
dm 0102 8
   13   57   90   24   68    9   75   31
?
dm 010a 8
   92   58   14   70   36   74   18   52
?
dm 0112 8
    0   16    4   95    4   83   93   83
?
rn 7

IA    OC       EA    P      X       A     B     C       S       T
1006  LDA  A   0102  1008   00FB     13     1  H0000C   013D    231
1008  ADC  A   010A  100A   00FB    106     1  00N000   013D    236
100A  DAA      100A  100B   00FB      6     1  000000C  013D    238
100B  STA  A   0112  100D   00FB      6     1  000000C  013D    244
100D  DEX      100D  100E   00FA      6     1  000000C  013D    248
100E  DEC  B   100E  100F   00FA      6     0  000Z0C   013D    250
100F  BNE      1010  1011   00FA      6     0  000Z0C   013D    254

?
dm 0102 8
   13   57   90   24   68    9   75   31
?
dm 010a 8
   92   58   14   70   36   74   18   52
?
dm 0112 8
    6   16    4   95    4   83   93   83
?
ex

READY
```

APPENDIX C

COMPUTER LISTING OF THE SIM6800 SIMULATOR PROGRAM

```
MAIN: PROC OPTIONS(MAIN);

 1              MAIN: PROC OPTIONS(MAIN);
 2                    DCL 1 MOT(200),
                         2 OP FIXED BIN,
                         2 MNEMONIC CHAR(3),
                         2 ADDR_MODE FIXED BIN,
                         2 TIME FIXED BIN,
                         2 BYTES FIXED BIN,
                         2 MOT_NO FIXED BIN,
                         2 ADDR_ACC CHAR(1) INIT(' ');
 3                    DCL M(1:5000,2) FIXED BIN;
 4                    DCL (C,V,Z,N,I,H,ERR) BIT(1) INIT('0'B);
 5                    DCL BS BIT(15),CC BIT(6);
 6                    DCL (C1,C2,C3,C4,C5) FIXED BIN;
 7                    DCL (X,Y,XY) FIXED BIN;
 8                    DCL HC FIXED BIN INIT(5);
 9                    DCL $HC FIXED BIN INIT(0);
10                    DCL (BYTE2,IA,ADDR) FIXED BIN INIT(0);
11                    DCL (ACC_CODE,$ACC_CODE) BIT(1) INIT('0'B);
12                    DCL CHAREP CHAR(4);
13                    DCL HEXSTR CHAR(26);
14                    DCL HEXREP CHAR(16) VAR;
15                    DCL CONVERT RETURNS(CHAR(16)VAR);
16                    DCL CON RETURNS(FIXED BIN(15));
17                    DCL CODE ENTRY(FIXED BIN,FIXED BIN,FIXED BIN,
                      FIXED BIN,FIXED BIN,FIXED BIN,FIXED BIN,FIXED BIN,
                      BIT(1),BIT(1),BIT(1),BIT(1),BIT(1),FIXED BIN);
18                    DCL (T,EA,PC,IX,ACCA,ACCB,SP,ACCX) FIXED BIN INIT(0);
19                    DCL CC_CHAR CHAR(6);
20                    DCL OC_CHAR(3) INIT('000');
21                    DCL BUFFER CHAR(120);
22                    DCL LBL3(10) LABEL;
23                    DCL (Z0,Z1,Z2,Z3)  FIXED BIN;
24                    DCL Y0 CHAR(2);
25                    DCL (Y1,Y2,Y3) CHAR(1);
26                    DCL REGISTER CHAR(7) INIT('PXSCABT');
27                    DCL COMMAND CHAR(14) INIT('SMOMSRRNHROREX');
28                    DCL DSO FILE INPUT;
29                    DCL DSM FILE INPUT;
30                    OPEN FILE(SYSPRINT) LINESIZE(132);


31             PRINT: PROC;
32                    IF $HC=1 THEN DO;
34                    PUT SKIP;
35                    PUT EDIT('IA','OC','EA','P','X','A','B','C','S','T')
                      (COL(1),A(2),COL(7),A(2),COL(15),A(2),COL(21),A(1),
                       COL(27),A(1),COL(35),A(1),COL(41),A(1),COL(47),A(1),
                         COL(53),A(1),COL(60),A(1));
36                    PUT SKIP;
37                    END;
38                    CC_CHAR='000000';
39                    IF H THEN SUBSTR(CC_CHAR,1,1)='H';
41                    IF I THEN SUBSTR(CC_CHAR,2,1)='I';
43                    IF N THEN SUBSTR(CC_CHAR,3,1)='N';
45                    IF Z THEN SUBSTR(CC_CHAR,4,1)='Z';
47                    IF V THEN SUBSTR(CC_CHAR,5,1)='V';
49                    IF C THEN SUBSTR(CC_CHAR,6,1)='C';
51                    PUT EDIT(CONVERT(UNSPEC(IA)),OC,MOT(MM).ADDR_ACC,
                      CONVERT(UNSPEC(EA)),
                      CONVERT(UNSPEC(PC)),CONVERT(UNSPEC(IX)),ACCA,ACCB,CC_CHAR,
                      CONVERT(UNSPEC(SP)),T)
                      (COL(1),A(4),COL(7),A(4),COL(12),A(1),COL(15),A(4),
                      COL(20),A(4),COL(27),A(4),COL(33),F(5),COL(39),F(5),
                      COL(45),A(6),COL(52),A(4),COL(58),F(5));
52                    PUT SKIP;
53                    END PRINT;


54             EVAL: PROC;
55                    DCL LBL2(72) LABEL;
56                    LBL2(1)=E1;   LBL2(19)=E19; LBL2(37)=E37; LBL2(55)=E55;
60                    LBL2(2)=E2;   LBL2(20)=E20; LBL2(38)=E38; LBL2(56)=E56;
64                    LBL2(3)=E3;   LBL2(21)=E21; LBL2(39)=E39; LBL2(57)=E57;
68                    LBL2(4)=E4;   LBL2(22)=E22; LBL2(40)=E40; LBL2(58)=E99;
72                    LBL2(5)=E5;   LBL2(23)=E23; LBL2(41)=E41; LBL2(59)=E59;
76                    LBL2(6)=E6;   LBL2(24)=E24; LBL2(42)=E42; LBL2(60)=E60;
80                    LBL2(7)=E7;   LBL2(25)=E25; LBL2(43)=E43; LBL2(61)=E61;
84                    LBL2(8)=E8;   LBL2(26)=E99; LBL2(44)=E44; LBL2(62)=E62;
88                    LBL2(9)=E9;   LBL2(27)=E27; LBL2(45)=E45; LBL2(63)=E63;
92                    LBL2(10)=E10; LBL2(28)=E28; LBL2(46)=E46; LBL2(64)=E99;
```

```
MAIN: PROC OPTIONS(MAIN);

 96                  LBL2(11)=E11; LBL2(29)=E29; LBL2(47)=E47; LBL2(65)=E65;
100                  LBL2(12)=E12; LBL2(30)=E30; LBL2(48)=E48; LBL2(66)=E66;
104                  LBL2(13)=E13; LBL2(31)=E31; LBL2(49)=E49; LBL2(67)=E67;
108                  LBL2(14)=E14; LBL2(32)=E32; LBL2(50)=E50; LBL2(68)=E68;
112                  LBL2(15)=E15; LBL2(33)=E33; LBL2(51)=E51; LBL2(69)=E69;
116                  LBL2(16)=E16; LBL2(34)=E34; LBL2(52)=E52; LBL2(70)=E70;
120                  LBL2(17)=E17; LBL2(35)=E35; LBL2(53)=E99; LBL2(71)=E71;
124                  LBL2(18)=E18; LBL2(36)=E36; LBL2(54)=E54; LBL2(72)=E99;
128                  BYTE2=M(PC,2);
129                  $PC=PC;
130                  PC=PC+MOT(MM).BYTES;
131                  T=T+MOT(MM).TIME;
132                  OC=MOT(MM).MNEMONIC;
133                  CALL ADDRESS;
134                  IF ADDR>5000 THEN DO;
136                  ERR='1'B;
137                  PUT SKIP;
138                  PUT EDIT('**** ERROR 313 ',CONVERT(UNSPEC(ADDR)),
                     'THE CALCULATED ADDRESS BEYOND BOUNDS OF MEMORY')
                     (COL(1),A(14),COL(16),A(4),SKIP,A);
139                  PUT SKIP;
140                  RETURN;
141                  END;
142                  GO TO LBL2(MOT(MM).MOT_NO);

143             E1:  /* ABA */
144                  X=ACCA;  Y=ACCB;
145                  ACCA=ACCA+ACCB;
146                  XY=ACCA;
147                  CALL CODE(X,Y,XY,4,4,4,4,6,H,N,Z,V,C,ACCX);
148                  RETURN;

149             E2:  /* ADC */
150                  IF MOT(MM).ADDR_MODE >4 THEN ADDR=M(ADDR,2);
151                  IF ACC_CODE THEN DO;     X=ACCA+BIN(C);
154                                           Y=(ADDR);
155                                           ACCA=ACCA+ADDR+BIN(C);
156                                           XY=(ACCA);
157                  END;
158                  ELSE DO; X=(ACCB);  Y=(ADDR);
161                           ACCB=ACCB+ADDR+BIN(C);
162                           XY=(ACCB);
163                  END;
164                  $ACC_CODE=ACC_CODE;
165                  CALL CODE(X,Y,XY,4,4,4,4,6,H,N,Z,V,C,ACCX);
166                  RETURN;

167             E3:  /* ADD */
168                  IF MOT(MM).ADDR_MODE >4 THEN ADDR=M(ADDR,2);
169                  IF ACC_CODE THEN DO;  X=ACCA;  Y=ADDR;
173                                        ACCA=ACCA+ADDR;  XY=ACCA;
175                                        END;
176                              ELSE DO; X=ACCB;  Y=ADDR;
179                                       ACCB=ACCB+ADDR;  XY=ACCB;
181                                   END;
182                  CALL CODE(X,Y,XY,4,4,4,4,6,H,N,Z,V,C,ACCX);
183                  RETURN;

184             E4:  /* AND */
185                  IF ACC_CODE THEN DO;    ACCA=DEC( BIT(ACCA) & BIT(M(ADDR,2)));
187                                          XY=ACCA;  END;
189                              ELSE DO;    ACCB=DEC( BIT(ACCB) & BIT(M(ADDR,2)));
191                                          XY=ACCB;  END;
193                  CALL CODE(X,Y,XY,1,4,4,2,1,H,N,Z,V,C,ACCX);
194                  RETURN;

195             E5:  /* ASL */
196                  IF ADDR=0 THEN DO; IF ACC_CODE THEN REL=ACCA;
199                                                 ELSE REL=ACCB;
200                  END;
201                  ELSE REL=M(ADDR,2);
202                  BS=BIT(REL);
203                  C=SUBSTR(BS,8,1);
204                  SUBSTR(BS,8,7)=SUBSTR(BS,9,7);
205                   SUBSTR(BS,15,1)='0'B;
206                  REL=DEC(BS);
207                  XY=REL;  CALL CODE(X,Y,XY,1,4,4,1,1,H,N,Z,V,C,ACCX);
209                  V=(N&¬C) | (¬N&C);
210                  IF ADDR=0 THEN DO; IF ACC_CODE THEN ACCA=REL;
214                                                 ELSE ACCB=REL;
```

```
MAIN: PROC OPTIONS(MAIN);

215                     END;
216                     ELSE M(ADDR,2)=REL;
217                     RETURN;

218               E6:   /* ASR */
219                      IF ADDR=0 THEN DO; IF ACC_CODE THEN REL=ACCA;
222                                                     ELSE REL=ACCB;
223                     END;
224                     ELSE REL=M(ADDR,2);
225                     BS=BIT(REL);
226                     I=SUBSTR(BS,8,1);  C=SUBSTR(BS,15,1);
228                      SUBSTR(BS,8,7)=SUBSTR(BS,9,7);
229                     SUBSTR(BS,8,1)=I;
230                     I='0'B;
231                     REL=DEC(BS);
232                     XY=REL;  CALL CODE(X,Y,XY,1,4,4,1,1,H,N,Z,V,C,ACCX);
234                     V=(N&¬C) | (¬N&C);
235                     IF ADDR=0 THEN DO;  IF ACC_CODE THEN ACCA=REL;
239                                                     ELSE ACCB=REL;
240                     END;
241                     ELSE M(ADDR,2)=REL;
242                     RETURN;

243               E7:   /* BCC */
244                     IF ¬C THEN PC=PC+2+ADDR;
245                     RETURN;

246               E8:   /* BCS */
247                     IF C THEN PC=PC+2+ADDR;
248                     RETURN;

249               E9:   /* BEQ */
250                     IF Z THEN PC=PC+2+ADDR;
251                     RETURN;

252               E10:  /* BGE */
253                     IF (N&V) | (¬N&¬V) THEN PC=PC+2+ADDR;
254                     RETURN;

255               E11:  /* BGT */
256                     IF (¬Z) & (N&V) | (¬N&¬V) THEN PC=PC+2+ADDR;
257                     RETURN;

258               E12:  /* BHI */
259                     IF (¬C) & (¬Z) THEN PC=PC+2+ADDR;
260                     RETURN;

261               E13:  /* BIT */
262                     IF ACC_CODE THEN XY=BIN( BIT(ACCA) & BIT(M(ADDR,2)) );
263                                 ELSE XY=BIN( BIT(ACCB) & BIT(M(ADDR,2)) );
264                     CALL CODE(X,Y,XY,1,4,4,2,1,H,N,Z,V,C,ACCX);
265                     RETURN;

266               E14:  /* BLE */
267                     IF (Z) | (N&¬V) | (¬N&V) THEN PC=PC+2+ADDR;
268                     RETURN;

269               E15:  /* BLS */
270                     IF (C) | (Z) THEN PC=PC+2+ADDR;
271                     RETURN;

272               E16:  /* BLT */
273                     IF (N&¬V) | (¬N&V) THEN PC=PC+2+ADDR;
274                     RETURN;

275               E17:  /* BMI */
276                     IF N THEN PC=PC+2+ADDR;
277                     RETURN;

278               E18:  /* BNE */
279                     IF ¬Z THEN PC=$PC+2+ADDR;
280                     RETURN;

281               E19:  /* BPL */
282                     IF ¬N THEN PC=PC+2+ADDR;
283                     RETURN;

284               E20:  /* BRA */
                        PC=PC+2+ADDR;
285                     RETURN;
```

```
MAIN: PROC OPTIONS(MAIN);

286             E21: /* BSR */
                     PC=PC+2;
287                  SP=SP-2;
288                  PC=PC+ADDR;
289                  RETURN;

290             E22: /* BVC */
291                  IF ¬V THEN PC=PC+2+ADDR;
292                  RETURN;

293             E23: /* BVS */
294                  IF V THEN PC=PC+2+ADDR;
295                  RETURN;

296             E24: /* CBA */
                     REL=ACCA-ACCB;
297                  X=ACCA;  Y=ACCB;  XY=REL;
300                  CALL CODE(X,Y,XY,1,4,4,5,5,H,N,Z,V,C,ACCX);
301                  RETURN;

302             E25: /* CLC */
                     C='0'B;
303                  RETURN;

304             E27: /* CLR */
305                  IF ADDR=0 THEN DO;  IF ACC_CODE THEN DO; ACCA=0;
309                                      XY=ACCA;  END;
311                                         ELSE DO; ACCB=0;
313                                      XY=ACCB;  END;
315                  END;
316                  ELSE DO; M(ADDR,2)=0;   XY=M(ADDR,2);  END;
320                  CALL CODE(X,Y,XY,1,2,3,2,2,H,N,Z,V,C,ACCX);
321                  RETURN;

322             E28: /* CLV */
                     V='0';
323                  RETURN;

324             E29: /* CMP */
325                  IF ACC_CODE THEN REL=ACCA;
326                              ELSE REL=ACCB;
327                  X=REL;  Y=ADDR;
329                  REL=REL-ADDR;
330                  XY=REL;  CALL CODE(X,Y,XY,1,4,4,5,5,H,N,Z,V,C,ACCX);
332                  IF REL<0  THEN N='1'B;
334                  IF REL=0  THEN Z='1'B;
336                  RETURN;

337             E30: /* COM */
338                  IF ADDR=0 THEN DO; IF ACC_CODE THEN DO; ACCA=DEC(¬BIT(ACCA));
342                                                          XY=ACCA;  END;
344                                     ELSE DO; ACCB=DEC(¬BIT(ACCB));
346                                              XY=ACCB;  END;
348                  END;
349                  ELSE DO; M(ADDR,2)=DEC(¬BIT(M(ADDR,2)));  END;
352                  CALL CODE(X,Y,XY,1,4,4,2,3,H,N,Z,V,C,ACCX);
353                  RETURN;

354             E31: /* CPX */
355                  X=IX;  Y=ADDR;
356                  REL=IX-ADDR;
357                  XY=REL;  CALL CODE(X,Y,XY,1,4,4,4,1,H,N,Z,V,C,ACCX);
359                  RETURN;

360             E32:  /* DAA */
                      CALL CODE(X,Y,XY,1,4,4,2,7,H,N,Z,V,C,ACCX);
361                  IF ¬ACC_CODE THEN ACCA=ACCX;
363                               ELSE ACCB=ACCX;
364                  RETURN;

365             E34: /* DES */
                     SP=SP-1;
366                  RETURN;

367             E36: /* EOR */
368                  IF ACC_CODE THEN DO;
369               ACCA=DEC((BIT(ACCA)&¬BIT(M(ADDR,2))) | (¬BIT(ACCA)&BIT(M(ADDR,2))));
370                  XY=ACCA;  END;
372                  ELSE DO;
373               ACCB=DEC((BIT(ACCB)&¬BIT(M(ADDR,2))) | (¬BIT(ACCB)&BIT(M(ADDR,2))));
374                  XY=ACCB;  END;
```

```
MAIN: PROC OPTIONS(MAIN);

376                 CALL CODE(X,Y,XY,1,4,4,2,1,H,N,Z,V,C,ACCX);
377                 RETURN;

378           E37:  /* INC */
379                 IF ADDR ¬=0 THEN DO; X=M(ADDR,2); M(ADDR,2)=M(ADDR,2)+1;
382                                      XY=M(ADDR,2);  END;
384                 ELSE DO;  IF ACC_CODE THEN DO; X=ACCA; ACCA=ACCA-1;
389                                                XY=ACCA;  END;
391                                       ELSE DO; X=ACCB; ACCB=ACCB+1;
394                                                XY=ACCB;  END;
396                 END;
397                 CALL CODE(X,Y,XY,1,4,4,1,1,H,N,Z,V,C,ACCX);
398                 IF X=127 THEN V='1'B;  ELSE V='0'B;
401                 RETURN;

402           E38:  /* INS */
                    SP=SP+1;
403                 RETURN;

404           E39:  /* INX */
                    IX=IX+1;
405                 XY=IX;  CALL CODE(X,Y,XY,1,1,4,1,1,H,N,Z,V,C,ACCX);
407                 RETURN;

408           E40:  /* JUMP */
                    PC=ADDR;
409                 RETURN;

410           E33:  /* DEC */
411                 IF ADDR ¬= 0 THEN DO;
412                 X=BIT(M(ADDR,2));
413                     M(ADDR,2)=M(ADDR,2)-1;
414                 XY=BIT(M(ADDR,2));
415                 GO TO E333;
416                 END;
417                 IF ACC_CODE THEN DO;  X=(ACCA);
420                                       ACCA=ACCA-1;
421                                       XY=(ACCA);
422                 END;
423                 ELSE DO;  X=(ACCB);  ACCB=ACCB-1;
426                           XY=(ACCB);
427                 END;
428           E333: CALL CODE(X,Y,XY,1,4,4,1,1,H,N,Z,V,C,ACCX);
429                  IF BIN(X)>80 THEN V='1'B;  ELSE V='0'B;
432                 RETURN;

433           E35:  /* DEX */
                    IX=IX-1;
434                 XY=(IX);
435                 CALL CODE(X,Y,XY,1,1,4,1,1,H,N,Z,V,C,ACCX);
436                 RETURN;

437           E41:  /* JSR */
                    EA=SP-1;
438                 M(SP,2)=PC;
439                 SP=SP-2;
440                 PC=ADDR;
441                 RETURN;

442           E42:  /* LDA */
443                 IF MOT(MM).ADDR_MODE >4 THEN ADDR=M(ADDR,2);
444                 IF ACC_CODE THEN DO; ACCA=ADDR;  XY=(ACCA); END;
449                           ELSE DO; ACCB=ADDR;  XY=(ACCB); END;
453                 CALL CODE(X,Y,XY,1,4,4,2,1,H,N,Z,V,C,ACCX);
454                 RETURN;

455           E43:  /* LDS */
456                 IF MOT(MM).ADDR_MODE >4 THEN ADDR=M(ADDR,2);
457                 SP=ADDR;
458                 XY=(SP);
459                 CALL CODE(X,Y,XY,1,4,4,2,1,H,N,Z,V,C,ACCX);
460                 RETURN;

461           E44:  /* LDX */
462                 IF MOT(MM).ADDR_MODE >4 THEN DO;
463                                              ADDR=M(ADDR,2);
464                 END;
465                 IX=ADDR;
466                 XY=(IX);
467                 CALL CODE(X,Y,XY,1,4,4,2,1,H,N,Z,V,C,ACCX);
468                 RETURN;
```

```
MAIN: PROC OPTIONS(MAIN);

469             E45: /* LSR */
470                  IF ADDR=0 THEN DO; IF ACC_CODE THEN REL=ACCA;
473                                     ELSE REL=ACCB;
474                  END;
475                  ELSE REL=M(ADDR,2);
476                  BS=BIT(REL);
477                  C=SUBSTR(BS,15,1);
478                   SUBSTR(BS,8,7)=SUBSTR(BS,9,7);
479                  SUBSTR(BS,8,1)='0'B;
480                  REL=DEC(BS);
481                  XY=REL;  CALL CODE(X,Y,XY,1,2,4,1,1,H,N,Z,V,C,ACCX);
483                  V=(N&¬C) | (¬N&C);
484                  IF ADDR=0 THEN DO; IF ACC_CODE THEN ACCA=REL;
488                                     ELSE ACCB=REL;
489                  END;
490                  ELSE M(ADDR,2)=REL;
491                  RETURN;

492             E46: /* NEG */
493                  IF ADDR=0 THEN DO;
494                     IF ACC_CODE THEN DO; X=ACCA; ACCA=0-ACCA;
498                                          XY=ACCA;  END;
500                                 ELSE DO; X=ACCB; ACCB=0-ACCB;
503                                          XY=ACCB;  END;
505                  END;
506                  ELSE DO; X=M(ADDR,2); M(ADDR,2)=0-M(ADDR,2);
509                           XY=M(ADDR,2);  END;
511                  CALL CODE(X,Y,XY,1,4,4,1,1,H,N,Z,V,C,ACCX);
512                  IF X=80 THEN V='1'B;  ELSE V='0'B;
515                  IF X=0 THEN C='1'B;  ELSE C='0'B;
518                  RETURN;

519             E47: /* NOP */
                     RETURN;

520             E48: /* ORA */
521                  IF ACC_CODE THEN DO; ACCA=DEC( BIT(ACCA) | BIT(M(ADDR,2)) );
523                                       XY=ACCA;  END;
525                              ELSE DO; ACCB=DEC( BIT(ACCB) | BIT(M(ADDR,2)) );
527                                       XY=ACCB;  END;
529                   CALL CODE(X,Y,XY,1,4,4,2,1,H,N,Z,V,C,ACCX);
530                  RETURN;

531             E49: /* PSH */
532                  IF ACC_CODE THEN M(SP,2)=ACCB;
533                             ELSE M(SP,2)=ACCB;
534                  SP=SP-1;
535                  RETURN;

536             E50: /* PUL */
                     SP=SP+1;
537                  IF ACC_CODE THEN ACCA=M(SP,2);
539                             ELSE ACCB=M(SP,2);
540                  RETURN;

541             E51: /* ROL */
542                  IF ADDR=0 THEN DO; IF ACC_CODE THEN REL=ACCA;
545                                     ELSE REL=ACCB;
546                  END;
547                  ELSE REL=M(ADDR,2);
548                  BS=BIT(REL);
549                  I=C;
550                  C=SUBSTR(BS,8,1);
551                  SUBSTR(BS,8,7)=SUBSTR(BS,9,7);
552                  SUBSTR(BS,15,1)=I;
553                  I='0'B;
554                  REL=DEC(BS);
555                  XY=REL;  CALL CODE(X,Y,XY,1,4,4,1,1,H,N,Z,V,C,ACCX);
557                  V=(N&¬C) | (¬N&C);
558                  IF ADDR=0 THEN DO; IF ACC_CODE THEN ACCA=REL;
562                                     ELSE ACCB=REL;
563                  END;
564                  ELSE M(ADDR,2)=REL;
565                  RETURN;

566             E52: /* ROR */
567                  IF ADDR=0 THEN DO; IF ACC_CODE THEN REL=ACCA;
570                                     ELSE REL=ACCB;
571                  END;
572                  ELSE REL=M(ADDR,2);
```

```
MAIN: PROC OPTIONS(MAIN);

573                     BS=BIT(REL);
574                     I=SUBSTR(BS,15,1);
575                      SUBSTR(BS,8,7)=SUBSTR(BS,8,7);
576                      SUBSTR(BS,8,1)=C;
577                     C=I;
578                     I='0'B;
579                     REL=DEC(BS);
580                     XY=REL;  CALL CODE(X,Y,XY,1,4,4,1,1,H,N,Z,V,C,ACCX);
582                     V=(N&¬C) | (¬N&C);
583                     IF ADDR=0 THEN DO; IF ACC_CODE THEN ACCA=REL;
587                                                    ELSE ACCB=REL;
588                     END;
589                     ELSE M(ADDR,2)=REL;
590                     RETURN;

591             E54:  /* RTS */
                      SP=SP+2;
592                   PC=M(SP,2);
593                   EA=SP;
594                   RETURN;

595             E55:  /* SBA */
                      ACCA=ACCA-ACCB;
596                   RETURN;

597             E56:  /* SBC */
598                   IF MOT(MM).ADDR_MODE >4 THEN ADDR=M(ADDR,2);
599                   IF ACC_CODE THEN ACCA=ACCA-ADDR-C;
601                              ELSE ACCB=ACCB-ADDR-C;
602                   RETURN;

603             E57:  /* SEC */
                      C='1'B;
604                   RETURN;

605             E59:  /* SEV */
                      V='1'B;
606                   RETURN;

607             E60:  /* STA */
608                   IF ACC_CODE THEN DO; M(ADDR,2)=ACCA;
610                                        XY=(ACCA);
611                                 END;
612                                 ELSE DO; M(ADDR,2)=ACCB;
614                                        XY=(ACCB);
615                                 END;
616                   CALL CODE(X,Y,XY,1,4,4,2,1,H,N,Z,V,C,ACCX);
617                   RETURN;

618             E61:  /* STS */
                      M(ADDR,2)=SP;
619                   XY=SP;  CALL CODE(X,Y,XY,1,4,4,2,1,H,N,Z,V,C,ACCX);
621                   RETURN;

622             E62:  /* STX */
                      M(ADDR,2)=IX;
623                   XY=IX;  CALL CODE(X,Y,XY,1,4,4,2,1,H,N,Z,V,C,ACCX);
625                   RETURN;

626             E63:  /* SUB */
627                   IF MOT(MM).ADDR_MODE > 4 THEN ADDR=M(ADDR,2);
628                   IF ACC_CODE THEN ACCA=ACCA-ADDR;
630                              ELSE ACCB=ACCB-ADDR;
631                   RETURN;

632             E65:  /* TAB */
                      ACCB=ACCA;
633                   XY=ACCB;  CALL CODE(X,Y,XY,1,4,4,2,1,H,N,Z,V,C,ACCX);
635                   RETURN;

636             E66:  /* TAP */
                      BS=BIT(ACCA);
637                   CC=SUBSTR(BS,10,6);
638                   N=SUBSTR(CC,1,1); I=SUBSTR(CC,2,1); N=SUBSTR(CC,3,1);
641                   Z=SUBSTR(CC,4,1); V=SUBSTR(CC,5,1); C=SUBSTR(CC,6,1);
644                   RETURN;

645             E67:  /* TBA */
                      ACCA=ACCB;
646                   XY=ACCA;  CALL CODE(X,Y,XY,1,4,4,2,1,H,N,Z,V,C,ACCX);
648                   RETURN;
```

```
MAIN: PROC OPTIONS(MAIN);

649               E68: /* TPA */
                       CC=H || I || N || Z || V || C;
650                    SUBSTR(BS,8,2)='11'B;
651                    SUBSTR(BS,10,6)=CC;
652                    ACCA=DEC(BS);
653                    RETURN;

654               E69: /* TST */
655                    IF ADDR=0 THEN DO;  IF ACC_CODE THEN XY=ACCA;
658                                                    ELSE XY=ACCB;
659                    END;
660                    ELSE XY=M(ADDR,2);
661                    CALL CODE(X,Y,XY,1,4,4,2,2,H,N,Z,V,C,ACCX);
662                    RETURN;

663               E70: /* TSX */
                       IX=SP+1;
664                    RETURN;

665               E71: /* TXS */
                       SP=IX-1;
666                    RETURN;

667               E99: RETURN;
668                    END EVAL;


669               ADDRESS: PROC;
670                    DCL ONE FIXED BIN,  XX BIT(15);
671                    DCL LBL1(7) LABEL;
672                    LBL1(1)=A1;
673                    LBL1(2)=A2;
674                    LBL1(3)=A3;
675                    LBL1(4)=A4;
676                    LBL1(5)=A5;
677                    LBL1(6)=A6;
678                    LBL1(7)=A7;
679                    IF(MOT(MM).ADDR_ACC='A') THEN ACC_CODE='1'B;
681                    ELSE IF (MOT(MM).ADDR_ACC='B') THEN ACC_CODE='0'B;
683                    GO TO LBL1(MOT(MM).ADDR_MODE);

684               A1:  /* INHERENT */
                       EA=$PC;
685                    RETURN;

686               A2:  /* RELATIVE */
                       EA=$PC+1;
687                    IF BYTE2>=240 THEN BYTE2=(255-BYTE2+1) * (-1);
689                    ADDR=BYTE2;
690                    RETURN;

691               A3:  /* IMMEDIATE */
                       IF MOT(MM).MOT_NO = 31 |
                          MOT(MM).MOT_NO = 43 |
692                       MOT(MM).MOT_NO = 44 THEN EA=$PC+2;
693                    ELSE EA=$PC+1;
694                    ADDR=BYTE2;
695                    RETURN;

696               A4:  /* ACCUMULATOR */
                       EA=$PC;
697                    ADDR=0;
698                    RETURN;

699               A5:  /* INDEXED */
                       EA=BYTE2+IX;
700                    ADDR=BYTE2+IX;
701                    RETURN;

702               A6:  /* EXTENDED */
                       ADDR=BYTE2;
703                    EA=ADDR+1;
704                    RETURN;

705               A7:  /* DIRECT */
                       ADDR=BYTE2;
706                    EA=ADDR+1;
707                    RETURN;

708                    END ADDRESS;
```

```
MAIN: PROC OPTIONS(MAIN);

LKUP: PROC;
      DO MM=1 BY 1 TO 197;
      IF M(PC,1)=MOT(MM).OP THEN GO TO LK1;
      END;
LK1:
      END LKUP;

CODE: PROC(XX,YY,XYXY,C1,C2,C3,C4,C5,H,N,Z,V,C,ACCX);
      DCL (H,N,Z,V,C) BIT(1);
      DCL (C1,C2,C3,C4,C5) FIXED BIN;
      DCL (X,Y,XY) BIT(15) INIT( (15) '0'B);
      DCL (XX,YY,XYXY,ACCX,LO,UP) FIXED BIN;
      DCL (BITUP,BITLO) BIT(15);
      DCL (LBL3(4),LBL4(4),LBL5(4),LBL6(5),LBL7(5)) LABEL;

      LBL3(1)=CC2; LBL4(1)=CC3; LBL5(1)=CC4; LBL6(1)=CC5; LBL7(1)=CC6;
      LBL3(2)=HCL; LBL4(2)=NCL; LBL5(2)=ZCL; LBL6(2)=VCL; LBL7(2)=CCL;
      LBL3(3)=HS;  LBL4(3)=NS;  LBL5(3)=ZS;  LBL6(3)=VS;  LBL7(3)=CS;
      LBL3(4)=HP;  LBL4(4)=NP;  LBL5(4)=ZP;  LBL6(4)=VP;  LBL7(4)=CP;
                                             LBL6(5)=VP2; LBL7(5)=CP2;

      UP=XX/10;  BITUP=BIT(UP);
      LO=XX-FLOOR(XX/10)*10;  BITLO=BIT(LO);
      SUBSTR(X,8,8)=SUBSTR(BITUP,12,4) || SUBSTR(BITLO,12,4);
      UP=YY/10;  BITUP=BIT(UP);
      LO=YY-FLOOR(YY/10)*10;  BITLO=BIT(LO);
      SUBSTR(Y,8,8)=SUBSTR(BITUP,12,4) || SUBSTR(BITLO,12,4);
      IF C5<=5 THEN DO;
      UP=XYXY/10;  BITUP=BIT(UP);
      LO=XYXY-FLOOR(XYXY/10)*10;  BITLO=BIT(LO);
      SUBSTR(XY,8,8)=SUBSTR(BITUP,12,4) || SUBSTR(BITLO,12,4);
      END;
      ELSE DO;
      UP=BIN(SUBSTR(X,8,4)) + BIN(SUBSTR(Y,8,4));
      LO=BIN(SUBSTR(X,12,4)) + BIN(SUBSTR(Y,12,4));
      IF LO>=16 THEN DO; LO=LO-16;  UP=UP+1;  END;
      BITUP=BIT(UP);  BITLO=BIT(LO);
 SUBSTR(XY,8,8)=SUBSTR(BITUP,12,4) || SUBSTR(BITLO,12,4);
   END;
      IF C5=6 THEN C5=4;
     IF C5<7 THEN GO TO CC1;
      LO=BIN(SUBSTR(XY,12,4));
      IF LO>9 | H='1'B THEN LO=LO+6;
      UP=BIN(SUBSTR(XY,8,4));
      IF LO>=16 THEN DO; LO=LO-16;  UP=UP+1;  END;
      IF UP>9 | C='1'B THEN DO; UP=UP+6; C='1'B; END;
                       ELSE C='0'B;
      BITUP=BIT(UP);  BITLO=BIT(LO);
      XY='0000000'B || SUBSTR(BITUP,12,4) || SUBSTR(BITLO,12,4);
      ACCX=BIN(SUBSTR(BITUP,12,4))*10 + BIN(SUBSTR(BITLO,12,4));
      C5=1;

CC1:  GO TO LBL3(C1);
HCL:  H='0'B;  GO TO CC2;
HS:   H='1'B;  GO TO CC2;
HP:   H=( SUBSTR(X,12,1) & SUBSTR(Y,12,1)) |
        ( SUBSTR(Y,12,1) &¬SUBSTR(XY,12,1)) |
        (¬SUBSTR(XY,12,1) & SUBSTR(X,12,1));
 CC2:   GO TO LBL4(C2);
NCL:  N='0'B;  GO TO CC3;
NS:   N='1'B;  GO TO CC3;
NP:   N=SUBSTR(XY,8,1);
CC3:  GO TO LBL5(C3);
ZCL:  Z='0'B;  GO TO CC4;
ZS:   Z='1'B;  GO TO CC4;
ZP:   Z=¬SUBSTR(XY,8,1) & ¬SUBSTR(XY,9,1) &
        ¬SUBSTR(XY,10,1) & ¬SUBSTR(XY,11,1) &
        ¬SUBSTR(XY,12,1) & ¬SUBSTR(XY,13,1) &
        ¬SUBSTR(XY,14,1) & ¬SUBSTR(XY,15,1);
CC4:  GO TO LBL6(C4);
VCL:  V='0'B;  GO TO CC5;
VS:   V='1'B;  GO TO CC5;
VP:   V=( SUBSTR(X,8,1) & SUBSTR(Y,8,1) &¬SUBSTR(XY,8,1) ) |
        (¬SUBSTR(X,8,1) &¬SUBSTR(Y,8,1) & SUBSTR(XY,8,1) );
```

```
MAIN: PROC OPTIONS(MAIN);

822                     GO TO CC5;
823                VP2: V=( SUBSTR(X,8,1) & ¬SUBSTR(Y,8,1) & ¬SUBSTR(XY,8,1) ) |
                          (¬SUBSTR(X,8,1) &  SUBSTR(Y,8,1) &  SUBSTR(XY,8,1));
824                CC5:  GO TO LAL7(C5);
825                CCL:  C='0'B;  GO TO CC6;
827                CS:   C='1'B;  GO TO CC6;
829                CP:   C=( SUBSTR(X,8,1) & SUBSTR(Y,8,1) ) |
                           ( SUBSTR(Y,8,1) &¬SUBSTR(XY,8,1) ) |
                           (¬SUBSTR(XY,8,1) & SUBSTR(X,8,1) );
830                     GO TO CC6;
831                CP2: C=(¬SUBSTR(X,8,1) &  SUBSTR(Y,8,1) ) |
                           ( SUBSTR(Y,8,1) &  SUBSTR(XY,8,1) ) |
                           ( SUBSTR(XY,8,1) & ¬SUBSTR(X,8,1) ) ;
832                CC6:  RETURN;
833                     END CODE;

834                CONVERT: PROC(A) RETURNS(CHAR(16)VAR);
835                     DCL A BIT(*),HEXREP CHAR(16)VAR INIT(''),
                        HEX_DIGITS CHAR(16) INIT('0123456789ABCDEF') STATIC,
                        TABLE(0:15) CHAR(1) DEF HEX_DIGITS,DIGIT FIXED BIN(4);
836                LP: DO K=1 BY 1 TO LENGTH(A)/4;
837                       DIGIT=SUBSTR(A,4*K-3,4);
838                       HEXREP=HEXREP||TABLE(DIGIT);
839                       END LP;
840                     RETURN(HEXREP);
841                     END CONVERT;

842                CON: PROC(CHAREP) RETURNS(FIXED BIN(15));
843                     DCL CHAREP CHAR(4);
844                     DCL BITS FIXED BIN;
845                     DCL DIGIT CHAR(15) INIT('123456789ABCDEF');
846                     BITS=0;
847                     DO II=1 BY 1 TO 4;
848                     K=INDEX (DIGIT,SUBSTR(CHAREP,II,1));
849                     BITS=BITS+(K)*(16**(4-II));
850                     END;
851                     RETURN(BITS);
852                     END CON;

853                SIM: PROC;
854                     IA=PC;
855                     CALL LKUP;
856                     CALL EVAL;
857                     IF ERR THEN RETURN;
859                     ADDR=0;
860                     $HC=$HC+1;
861                     IF $HC>HC THEN $HC=1;
863                     CALL PRINT;
864                     RETURN;
865                     END SIM;

866                CREATE1: PROC;
867                     DO MM=1 BY 1 TO 197;
868                     GET FILE(DSM) EDIT(MOT(MM))
                        (COL(1),F(3),COL(5),A(3),COL(9),F(1),COL(11),F(1),COL(13),F(1),
                        COL(15),F(2),COL(18),A(1));
869                     END;
870                     RETURN;
871                     END CREATE1;

872                CREATE2: PROC;
873                     ON ENDFILE(DS0) GO TO INEND;
875                     DO WHILE('1'B);
876                     GET FILE(DS0) LIST(NN,N1,N2);
877                     M(NN,1)=N1;
878                     M(NN,2)=N2 ;
879                     END;
880                INEND: RETURN;
881                     END CREATE2;

882                     ON ERROR BEGIN;
884                     PUT SKIP;
885                     PUT EDIT('**** ERROR 302 ',Y0,
                        'POSSIBLE SYNTAX ERROR IN SIMULATOR COMMAND')
                        (COL(1),A(15),COL(17),A(2),SKIP,COL(1),A);
886                     PUT SKIP;
887                     GO TO STRT;
888                     END;
889                STRT:  PUT SKIP;
```

```
MAIN: PROC OPTIONS(MAIN);

890                 PUT EDIT('SIMULATOR FOR THE MOTOROLA M6800 MICROPROCESSOR',
                          'WRITTEN FOR THE IBM 360',
                          'VERSION 1          1976')
                    (COL(5),A,SKIP,COL(15),A,SKIP,COL(15),A);
891                 PUT SKIP;

892               MINT: CALL CREATE1;
893                     CALL CREATE2;

894               MSTRT:
                        PUT SKIP;
895                     PUT EDIT('?')(COL(1),A(1));
896                     PUT SKIP;
897                     READ FILE(SYSIN) INTO(BUFFER);
898                     Z0=VERIFY(BUFFER,' ');
899                     BUFFER=SUBSTR(BUFFER,Z0);
900                     GET STRING(BUFFER) EDIT(Y0)(A(2));
901                     Z0=INDEX(BUFFER,' ');
902                     BUFFER=SUBSTR(BUFFER,Z0);
903                     Z0=INDEX(COMMAND,Y0);
904                     IF Z0=0 THEN DO;
906                     PUT SKIP;
907                     PUT EDIT('**** ERROR 301 ',Y0,
                        'UNDEFINED SIMULATOR COMMAND')
                        (COL(1),A(15),COL(17),A(2),SKIP,COL(1),A);
908                     PUT SKIP;
909                     GO TO MSTRT;
910                     END;
911                     Z0=(Z0+1)/2;
912                     LBL3(1)=M1;
913                     LBL3(2)=M2;
914                     LBL3(3)=M3;
915                     LBL3(4)=M4;
916                     LBL3(5)=M5;
917                     LBL3(6)=M6;
918                     LBL3(7)=M7;
919                     GO TO LBL3(Z0);
920               M1:   /* SET MEMORY */
                        Z0=VERIFY(BUFFER,' ');
921                     BUFFER=SUBSTR(BUFFER,Z0);
922                     GET STRING(BUFFER) EDIT(CHAREP)(A(4));
923                     Z0=INDEX(BUFFER,' ');
924                     BUFFER=SUBSTR(BUFFER,Z0);
925                     GET STRING(BUFFER) LIST(        Z2);
926                     Z1=CON(CHAREP);
927                     Z0=Z1+Z2-1;
928                     GET STRING(BUFFER) LIST(        Z2,(M(Z3,2) DO Z3=Z1 TO Z0));
929                     GO TO MSTRT;

930               M2:   /* DISPLAY MEMORY */
                        Z0=VERIFY(BUFFER,' ');
931                     BUFFER=SUBSTR(BUFFER,Z0);
932                     GET STRING(BUFFER) EDIT(CHAREP)(A(4));
933                     Z0=INDEX(BUFFER,' ');
934                     BUFFER=SUBSTR(BUFFER,Z0);
935                     GET STRING(BUFFER) LIST(Z2);
936                     Z1=CON(CHAREP);
937                     Z2=Z2+Z1-1;
938                     Z0=MOD(Z2,12);
939                     IF Z0=0 THEN Z0=Z2/12;
941                             ELSE Z0=Z2/12+1;
942                     PUT EDIT((M(Z3,2) DO Z3=Z1 TO Z2))((Z0)(12 F(5),SKIP));
943                     GO TO MSTRT;

944               M3:   /* SET REGISTERS */
                        Z0=VERIFY(BUFFER,' ');
945                     BUFFER=SUBSTR(BUFFER,Z0);
946                     GET STRING(BUFFER) EDIT(Y1)(A(1));
947                     Z0=INDEX(BUFFER,' ');
948                     BUFFER=SUBSTR(BUFFER,Z0);
949                     Z0=VERIFY(BUFFER,' ');
950                     BUFFER=SUBSTR(BUFFER,Z0);
951                     Z0=INDEX(REGISTER,Y1);
952                     IF Z0>4 THEN GO TO M3_2;
954                     IF Z0=4 THEN GO TO M3_3;
956               M3_1: GET STRING(BUFFER) EDIT(CHAREP)(A(4));
957                     IF Z0=1 THEN PC=CON(CHAREP);
959                     IF Z0=2 THEN IX=CON(CHAREP);
961                     IF Z0=3 THEN SP=CON(CHAREP);
963                     IF PC>5000 | SP>5000 | IX>5000 THEN DO;
965                     PUT SKIP;
```

```
MAIN: PROC OPTIONS(MAIN);

 966                    PUT EDIT('**** ERROR 306 ',Y1,
                        'THE OPERAND IN THE SIMULATOR COMMAND ',
                        'CAUSED A REGISTER OVERFLOW')
                        (COL(1),A(15),COL(17),A(1),SKIP,COL(1),A,SKIP,COL(1),A);
 967                    PUT SKIP;
 968                    GO TO MSTRT;
 969                    END;
 970                    GO TO MSTRT;
 971              M3_2: GET STRING(BUFFER) LIST(Z1);
 972                    IF Z0=5 THEN ACCA=Z1;
 974                    IF Z0=6 THEN ACCB=Z1;
 976                    IF Z0=7 THEN T=Z1;
 978                    GO TO MSTRT;

 979              M3_3: GET STRING(BUFFER) EDIT(CC)(A(6));
 980                    H=SUBSTR(CC,1,1); I=SUBSTR(CC,2,1); N=SUBSTR(CC,3,1);
 983                    Z=SUBSTR(CC,4,1); V=SUBSTR(CC,5,1); C=SUBSTR(CC,6,1);
 986                    GO TO MSTRT;

 987              M4:   /* RUN */
                        GET STRING(BUFFER) LIST(Z1);
 988                    DO Z2=1 BY 1 TO Z1;
 989                    CALL SIM;
 990                    IF ERR THEN GO TO MSTRT;
 992                    END;
 993                    GO TO MSTRT;

 994              M5:    /* SET HEADER COUNT */
                        GET STRING(BUFFER) LIST(Z1);
 995                    HC=Z1;
 996                    SHC=0;
 997                    GO TO MSTRT;

 998              M6:    /* DISPLAY REGISTERS */
                        CALL PRINT;
 999                    GO TO MSTRT;

1000              M7:   /* EXIT */
                        END MAIN;
```

BIBLIOGRAPHY

Cohen, Leo J. Operating System Analysis and Design. Rochelle Park, N.J.: Hayden Book Company, Inc., 1970.

M6800 Microprocessor Programming Manual. Motorola Semiconductor Products, Inc., 1975.

Ogdin, Jerry L. "Microprocessors: The Inevitable Technology." Modern Data 8 (January 1975): 42-47.

O'Malley, John. The Digital Computer. New York: Holt, Rinehart, and Winston, Inc., 1972.

Struble, George. Assembler Language Programming: The IBM System/360. Reading, Massachusetts: Addison-Wesley Publishing Company, 1971.