A Simulator for the Motorola M6800 Microprocessor

1976

Carolyn Elizabeth Jordan
University of Central Florida

Find similar works at: https://stars.library.ucf.edu/rtd

University of Central Florida Libraries http://library.ucf.edu

Part of the Engineering Commons

STARS Citation


This Masters Thesis (Open Access) is brought to you for free and open access by STARS. It has been accepted for inclusion in Retrospective Theses and Dissertations by an authorized administrator of STARS. For more information, please contact lee.dotson@ucf.edu.
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.
# TABLE OF CONTENTS

I. INTRODUCTION .................................................. 1

II. CHAPTER 1 ..................................................... 6
   SIM6800 Structure

III. CHAPTER 2 .................................................... 15
   SIM6800 Coding

IV. CONCLUSIONS AND SUGGESTIONS FOR FUTURE WORK ........... 19

V. APPENDIX A ................................................... 22
   SIM6800 User's Guide for the Motorola M6800 Microprocessor

VI. APPENDIX B ................................................... 34
   Sample Output from the SIM6800 Simulator Program

VII. APPENDIX C ................................................... 40
   Computer Listing of the SIM6800 Simulator Program

VIII. BIBLIOGRAPHY ............................................... 53
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   NAME 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   MBR

Fig. 3.—ASSEMBLER LANGUAGE PROGRAM
Fig. 4.—ASSEMBLED PROGRAM
<table>
<thead>
<tr>
<th>Line</th>
<th>Code 1</th>
<th>Code 2</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>4364</td>
<td>001</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>4365</td>
<td>032</td>
<td>254</td>
<td></td>
</tr>
<tr>
<td>4366</td>
<td>189</td>
<td>4096</td>
<td></td>
</tr>
<tr>
<td>4367</td>
<td>255</td>
<td>256</td>
<td></td>
</tr>
<tr>
<td>4368</td>
<td>206</td>
<td>258</td>
<td></td>
</tr>
<tr>
<td>4369</td>
<td>142</td>
<td>319</td>
<td></td>
</tr>
<tr>
<td>4096</td>
<td>198</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>4098</td>
<td>254</td>
<td>256</td>
<td></td>
</tr>
<tr>
<td>4101</td>
<td>012</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>4102</td>
<td>166</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>4104</td>
<td>169</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>4106</td>
<td>025</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>4107</td>
<td>167</td>
<td>23</td>
<td></td>
</tr>
<tr>
<td>4109</td>
<td>009</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>4110</td>
<td>090</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>4111</td>
<td>038</td>
<td>245</td>
<td></td>
</tr>
<tr>
<td>4113</td>
<td>057</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

Fig. 5.--OBJECT CODE FILE EXAMPLE1
Component

1. Instructional address
2. Operands

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

1. Operator number
2. Mnemonic code
3. Address mode
4. Time
5. Bytes
6. MOT number
7. Address accumulator

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:
<table>
<thead>
<tr>
<th>MNEMONIC</th>
<th>MEANING</th>
</tr>
</thead>
<tbody>
<tr>
<td>DM</td>
<td>Display memory</td>
</tr>
<tr>
<td>DR</td>
<td>Display register</td>
</tr>
<tr>
<td>EX</td>
<td>Exit simulator</td>
</tr>
<tr>
<td>HR</td>
<td>Set header count</td>
</tr>
<tr>
<td>RN</td>
<td>Run</td>
</tr>
<tr>
<td>SM</td>
<td>Set memory</td>
</tr>
<tr>
<td>SR</td>
<td>Set register</td>
</tr>
</tbody>
</table>

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

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:
<table>
<thead>
<tr>
<th>Bit No.</th>
<th>Condition Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>C (Carry-borrow)</td>
</tr>
<tr>
<td>1</td>
<td>V (Overflow)</td>
</tr>
<tr>
<td>2</td>
<td>Z (Zero)</td>
</tr>
<tr>
<td>3</td>
<td>N (Negative)</td>
</tr>
<tr>
<td>4</td>
<td>I (Interrupt Mask)</td>
</tr>
<tr>
<td>5</td>
<td>H (Half-carry)</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>MACRO Command</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD</td>
<td>Delete MACRO</td>
</tr>
<tr>
<td>ML</td>
<td>List MACRO</td>
</tr>
<tr>
<td>MR</td>
<td>Read MACRO from machine file</td>
</tr>
<tr>
<td>MS</td>
<td>Save MACRO in machine file</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>MNEMONIC</th>
<th>MEANING</th>
</tr>
</thead>
<tbody>
<tr>
<td>DM</td>
<td>Display memory</td>
</tr>
<tr>
<td>DR</td>
<td>Display registers</td>
</tr>
<tr>
<td>EX</td>
<td>Exit simulator</td>
</tr>
<tr>
<td>HR</td>
<td>Set header count</td>
</tr>
<tr>
<td>RN</td>
<td>Run</td>
</tr>
<tr>
<td>SM</td>
<td>Set memory</td>
</tr>
<tr>
<td>SR</td>
<td>Set registers</td>
</tr>
</tbody>
</table>

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 hexadecimal 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 v1 v2 ... vn
```

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

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:

<table>
<thead>
<tr>
<th>r</th>
<th>v</th>
</tr>
</thead>
<tbody>
<tr>
<td>P -- Program counter</td>
<td>Hexidecimal</td>
</tr>
<tr>
<td>X -- Index register</td>
<td>Hexidecimal</td>
</tr>
<tr>
<td>A -- Accumulator A</td>
<td>Integer</td>
</tr>
<tr>
<td>B -- Accumulator B</td>
<td>Integer</td>
</tr>
<tr>
<td>C -- Condition codes</td>
<td>Bit-string*</td>
</tr>
<tr>
<td>S -- Stack pointer</td>
<td>Hexidecimal</td>
</tr>
<tr>
<td>T -- Time</td>
<td>Integer</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Error Number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>301</td>
<td>**** ERROR 301 AAAAA\</td>
</tr>
<tr>
<td></td>
<td>Undefined simulator command.</td>
</tr>
<tr>
<td>302</td>
<td>**** ERROR 302 AAAAA\</td>
</tr>
<tr>
<td></td>
<td>Possible syntax error in simulator command.</td>
</tr>
<tr>
<td>306</td>
<td>**** ERROR 306 AAAAA\</td>
</tr>
<tr>
<td></td>
<td>The operand in the simulator command caused a</td>
</tr>
<tr>
<td></td>
<td>register overflow.</td>
</tr>
<tr>
<td>313</td>
<td>**** ERROR 313 NNNN\</td>
</tr>
<tr>
<td></td>
<td>The calculated address beyond bounds of memory.</td>
</tr>
</tbody>
</table>

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:

dec sim6800(example1)

The SR command was used to initialize the program counter with the hexadecimal 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.
SIMULATOR FOR THE MOTO DLA "6800 MICROPROCESSOR
WRITTEN FOR THE IBM 360
VERSION 1 1976

? sm 0102 8 13 57 90 24 63 99 75 31
?
? sm 010a 8 92 58 14 70 36 74 18 52
?
? sm 0112 0 0 0 0 0 0 0 0
?
? sr p 1100
?
? hr 7
?
? rn 7

<table>
<thead>
<tr>
<th>IA</th>
<th>OC</th>
<th>EA</th>
<th>P</th>
<th>X</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>S</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>1100</td>
<td>LDA</td>
<td>1102</td>
<td>1103</td>
<td>0000</td>
<td>0</td>
<td>0</td>
<td>00000</td>
<td>013F</td>
<td>3</td>
</tr>
<tr>
<td>1105</td>
<td>LDX</td>
<td>1105</td>
<td>1106</td>
<td>0102</td>
<td>0</td>
<td>0</td>
<td>00000</td>
<td>013F</td>
<td>6</td>
</tr>
<tr>
<td>1107</td>
<td>STX</td>
<td>0101</td>
<td>1109</td>
<td>0102</td>
<td>0</td>
<td>0</td>
<td>00000</td>
<td>013F</td>
<td>12</td>
</tr>
<tr>
<td>1109</td>
<td>JSR</td>
<td>013E</td>
<td>1000</td>
<td>0102</td>
<td>0</td>
<td>0</td>
<td>00000</td>
<td>013D</td>
<td>21</td>
</tr>
<tr>
<td>1200</td>
<td>LDA</td>
<td>B</td>
<td>1001</td>
<td>1002</td>
<td>0102</td>
<td>0</td>
<td>00000</td>
<td>013D</td>
<td>23</td>
</tr>
<tr>
<td>1202</td>
<td>LDX</td>
<td>0101</td>
<td>1005</td>
<td>0102</td>
<td>0</td>
<td>0</td>
<td>00000</td>
<td>013D</td>
<td>26</td>
</tr>
<tr>
<td>1205</td>
<td>CLC</td>
<td>1005</td>
<td>1006</td>
<td>0102</td>
<td>0</td>
<td>0</td>
<td>00000</td>
<td>013D</td>
<td>30</td>
</tr>
</tbody>
</table>

? rn 7

<table>
<thead>
<tr>
<th>IA</th>
<th>OC</th>
<th>EA</th>
<th>P</th>
<th>X</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>S</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>1006</td>
<td>LDA</td>
<td>A</td>
<td>0109</td>
<td>1003</td>
<td>0102</td>
<td>S1</td>
<td>8</td>
<td>000000</td>
<td>013D</td>
</tr>
<tr>
<td>1007</td>
<td>ADC</td>
<td>A</td>
<td>0111</td>
<td>100A</td>
<td>0102</td>
<td>S3</td>
<td>8</td>
<td>000000</td>
<td>013D</td>
</tr>
<tr>
<td>100A</td>
<td>DAA</td>
<td></td>
<td>100A</td>
<td>100B</td>
<td>0102</td>
<td>83</td>
<td>8</td>
<td>000000</td>
<td>013D</td>
</tr>
<tr>
<td>100B</td>
<td>STA</td>
<td>A</td>
<td>0119</td>
<td>100D</td>
<td>0102</td>
<td>85</td>
<td>8</td>
<td>000000</td>
<td>013D</td>
</tr>
<tr>
<td>100D</td>
<td>DEX</td>
<td></td>
<td>100D</td>
<td>100E</td>
<td>0101</td>
<td>83</td>
<td>8</td>
<td>000000</td>
<td>013D</td>
</tr>
<tr>
<td>100E</td>
<td>DEE</td>
<td>B</td>
<td>100E</td>
<td>100F</td>
<td>0101</td>
<td>83</td>
<td>7</td>
<td>000000</td>
<td>013D</td>
</tr>
<tr>
<td>100F</td>
<td>BHE</td>
<td></td>
<td>1010</td>
<td>1006</td>
<td>0101</td>
<td>83</td>
<td>7</td>
<td>000000</td>
<td>013D</td>
</tr>
</tbody>
</table>

?
<table>
<thead>
<tr>
<th>IA</th>
<th>CC</th>
<th>EA</th>
<th>P</th>
<th>X</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>S</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>0100</td>
<td>LPA A</td>
<td>0108</td>
<td>1008</td>
<td>0101</td>
<td>75</td>
<td>7</td>
<td>000000</td>
<td>013D</td>
<td>63</td>
</tr>
<tr>
<td>0108</td>
<td>ADC A</td>
<td>0110</td>
<td>100A</td>
<td>0101</td>
<td>93</td>
<td>7</td>
<td>000000</td>
<td>013D</td>
<td>63</td>
</tr>
<tr>
<td>010A</td>
<td>DAA</td>
<td>100A</td>
<td>100B</td>
<td>0101</td>
<td>93</td>
<td>7</td>
<td>000000</td>
<td>013D</td>
<td>70</td>
</tr>
<tr>
<td>010B</td>
<td>STA A</td>
<td>0118</td>
<td>100D</td>
<td>0101</td>
<td>93</td>
<td>7</td>
<td>000000</td>
<td>013D</td>
<td>76</td>
</tr>
<tr>
<td>010D</td>
<td>DEX</td>
<td>100D</td>
<td>100E</td>
<td>0101</td>
<td>93</td>
<td>7</td>
<td>000000</td>
<td>013D</td>
<td>83</td>
</tr>
<tr>
<td>010E</td>
<td>DEC B</td>
<td>100E</td>
<td>100F</td>
<td>0109</td>
<td>93</td>
<td>6</td>
<td>000000</td>
<td>013D</td>
<td>82</td>
</tr>
<tr>
<td>010F</td>
<td>BNE</td>
<td>1010</td>
<td>1006</td>
<td>0100</td>
<td>93</td>
<td>6</td>
<td>000000</td>
<td>013D</td>
<td>66</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>IA</th>
<th>CC</th>
<th>EA</th>
<th>P</th>
<th>X</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>S</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>0112</td>
<td>LPA A</td>
<td>0118</td>
<td>1008</td>
<td>0101</td>
<td>75</td>
<td>7</td>
<td>000000</td>
<td>013D</td>
<td>63</td>
</tr>
<tr>
<td>0110</td>
<td>ADC A</td>
<td>0110</td>
<td>100A</td>
<td>0101</td>
<td>93</td>
<td>7</td>
<td>000000</td>
<td>013D</td>
<td>63</td>
</tr>
<tr>
<td>011A</td>
<td>DAA</td>
<td>100A</td>
<td>100B</td>
<td>0101</td>
<td>93</td>
<td>7</td>
<td>000000</td>
<td>013D</td>
<td>70</td>
</tr>
<tr>
<td>011B</td>
<td>STA A</td>
<td>0118</td>
<td>100D</td>
<td>0101</td>
<td>93</td>
<td>7</td>
<td>000000</td>
<td>013D</td>
<td>76</td>
</tr>
<tr>
<td>011D</td>
<td>DEX</td>
<td>100D</td>
<td>100E</td>
<td>0101</td>
<td>93</td>
<td>7</td>
<td>000000</td>
<td>013D</td>
<td>83</td>
</tr>
<tr>
<td>011E</td>
<td>DEC B</td>
<td>100E</td>
<td>100F</td>
<td>0109</td>
<td>93</td>
<td>6</td>
<td>000000</td>
<td>013D</td>
<td>82</td>
</tr>
<tr>
<td>011F</td>
<td>BNE</td>
<td>1010</td>
<td>1006</td>
<td>0100</td>
<td>93</td>
<td>6</td>
<td>000000</td>
<td>013D</td>
<td>66</td>
</tr>
<tr>
<td>IA</td>
<td>OC</td>
<td>EA</td>
<td>P</td>
<td>X</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>S</td>
<td>T</td>
</tr>
<tr>
<td>----</td>
<td>----</td>
<td>----</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>1006</td>
<td>LDA</td>
<td>A</td>
<td>0106</td>
<td>1008</td>
<td>00FE</td>
<td>00</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1008</td>
<td>ADC</td>
<td>A</td>
<td>010E</td>
<td>100A</td>
<td>00FE</td>
<td>104</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>100A</td>
<td>DAA</td>
<td>100A</td>
<td>100B</td>
<td>00FE</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>100B</td>
<td>STA</td>
<td>A</td>
<td>0118</td>
<td>100D</td>
<td>00FE</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>100D</td>
<td>DEX</td>
<td>100D</td>
<td>100E</td>
<td>00FE</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>100E</td>
<td>DEC</td>
<td>B</td>
<td>100E</td>
<td>100F</td>
<td>00FE</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>100F</td>
<td>BNE</td>
<td>1010</td>
<td>1006</td>
<td>00FE</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

? dm 0102 3
   13 57 90 24 63 9 75 31
? dm 010a 3
   92 58 14 70 36 74 13 52
? dm 0112 3
   0 0 0 0 4 83 93 83

? rn 7

<table>
<thead>
<tr>
<th>IA</th>
<th>OC</th>
<th>EA</th>
<th>P</th>
<th>X</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>S</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>1006</td>
<td>LDA</td>
<td>A</td>
<td>0105</td>
<td>1008</td>
<td>00FE</td>
<td>24</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1008</td>
<td>ADC</td>
<td>A</td>
<td>010D</td>
<td>100A</td>
<td>00FE</td>
<td>95</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>100A</td>
<td>DAA</td>
<td>100A</td>
<td>100B</td>
<td>00FE</td>
<td>95</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>100B</td>
<td>STA</td>
<td>A</td>
<td>0115</td>
<td>100D</td>
<td>00FE</td>
<td>95</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>100D</td>
<td>DEX</td>
<td>100D</td>
<td>100E</td>
<td>00FD</td>
<td>95</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>100E</td>
<td>DEC</td>
<td>B</td>
<td>100E</td>
<td>100F</td>
<td>00FD</td>
<td>95</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>100F</td>
<td>BNE</td>
<td>1010</td>
<td>1006</td>
<td>00FD</td>
<td>95</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

? dm 0102 3
   13 57 90 24 63 9 75 31
? dm 010a 3
   92 58 14 70 36 74 13 52
? dm 0112 3
   0 0 0 0 4 83 93 83

? rn 7

<table>
<thead>
<tr>
<th>IA</th>
<th>OC</th>
<th>EA</th>
<th>P</th>
<th>X</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>S</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>1006</td>
<td>LDA</td>
<td>A</td>
<td>0104</td>
<td>1008</td>
<td>00FD</td>
<td>92</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1008</td>
<td>ADC</td>
<td>A</td>
<td>010C</td>
<td>100A</td>
<td>00FD</td>
<td>104</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>100A</td>
<td>DAA</td>
<td>100A</td>
<td>100B</td>
<td>00FD</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>100B</td>
<td>STA</td>
<td>A</td>
<td>0114</td>
<td>100D</td>
<td>00FD</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>100D</td>
<td>DEX</td>
<td>100D</td>
<td>100E</td>
<td>00FC</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>100E</td>
<td>DEC</td>
<td>B</td>
<td>100E</td>
<td>100F</td>
<td>00FC</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>100F</td>
<td>BNE</td>
<td>1010</td>
<td>1006</td>
<td>00FC</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>IA</td>
<td>CC</td>
<td>EA</td>
<td>P</td>
<td>X</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>S</td>
<td>T</td>
</tr>
<tr>
<td>----</td>
<td>----</td>
<td>----</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>1006</td>
<td>LDA A</td>
<td>0103</td>
<td>1008</td>
<td>00FC</td>
<td>57</td>
<td>2</td>
<td>00000C</td>
<td>013D</td>
<td>023</td>
</tr>
<tr>
<td>1008</td>
<td>ADC A</td>
<td>0108</td>
<td>100A</td>
<td>00FC</td>
<td>116</td>
<td>2</td>
<td>00000C</td>
<td>013D</td>
<td>208</td>
</tr>
<tr>
<td>100A</td>
<td>DAA</td>
<td>100A</td>
<td>100B</td>
<td>00FC</td>
<td>16</td>
<td>2</td>
<td>00000C</td>
<td>013D</td>
<td>210</td>
</tr>
<tr>
<td>100B</td>
<td>STA A</td>
<td>0113</td>
<td>100D</td>
<td>00FC</td>
<td>16</td>
<td>2</td>
<td>00000C</td>
<td>013D</td>
<td>215</td>
</tr>
<tr>
<td>100D</td>
<td>DEX</td>
<td>100D</td>
<td>100E</td>
<td>00FB</td>
<td>16</td>
<td>2</td>
<td>00000C</td>
<td>013D</td>
<td>220</td>
</tr>
<tr>
<td>100E</td>
<td>DEC B</td>
<td>100E</td>
<td>100F</td>
<td>00FB</td>
<td>16</td>
<td>2</td>
<td>00000C</td>
<td>013D</td>
<td>222</td>
</tr>
<tr>
<td>100F</td>
<td>BNE</td>
<td>1010</td>
<td>1006</td>
<td>00FB</td>
<td>16</td>
<td>2</td>
<td>00000C</td>
<td>013D</td>
<td>226</td>
</tr>
</tbody>
</table>

?  
| dm 0102 3 | 13 | 57 | 90 | 24 | 68 | 9 | 75 | 31 |
?  
| dm 010a 3 | 92 | 58 | 14 | 70 | 36 | 74 | 13 | 52 |
?  
| dm 0112 3 | 0 | 0 | 4 | 95 | 4 | 83 | 93 | 83 |
?  
| ex |  

READY
APPENDIX C

COMPUTER LISTING OF THE SIM6800
SIMULATOR PROGRAM
MAIN: PROC OPTIONS(MAIN);

OCL 1 MORT (200),
OP FIXED BIN,
Mnemonic CHAR (3),
ADDRESS MODE FIXED BIN,
TIME FIXED BIN,
BYTES FIXED BIN,
NOT NO FIXED BIN,
ADDRESS CHAR(1) INIT(" ");
OCL M(1:5000,2) FIXED BIN,
OCL S(1:120) FIXED BIN,
OCL BIT(1:5) CC CHAR (6) INIT(10' R);
OCL T(1:20) FIXED BIN,
OCL C(2:6) FIXED BIN,
OCL X(4:Y:Z) FIXED BIN,
OCL H FIXED BIN INIT (5);
OCL M(1) FIXED BIN INIT (0);
OCL X(2:3) FIXED BIN INIT (0: 9);
OCL X(3:4) FIXED BIN INIT (0:9);
MAIN: PROC OPTIONS(MAIN):

LAL2(11)=EI: LAL2(29)=EF: LAL2(47)=EF: LAL2(65)=EE:
LAL2(12)=EI: LAL2(20)=EF: LAL2(48)=EF: LAL2(66)=EE:
LAL2(13)=EI: LAL2(21)=EI: LAL2(49)=EF: LAL2(67)=EF:
LAL2(14)=EI: LAL2(22)=EI: LAL2(50)=EF: LAL2(68)=EF:
LAL2(15)=EI: LAL2(23)=EI: LAL2(51)=EF: LAL2(69)=EF:
LAL2(16)=EI: LAL2(24)=EI: LAL2(52)=EF: LAL2(70)=EF:
LAL2(17)=EI: LAL2(25)=EI: LAL2(53)=EF: LAL2(71)=EF:
LAL2(18)=EI: LAL2(26)=EI: LAL2(54)=EF: LAL2(72)=EF:

RTF2EM(PC+2):

SPCSCI;
PC=PC+X(MM), BYTES;
T1=TIME(MM), TIME1;
QC=MOT(MM), M,N,MONIC;
CALL ADDRESS1;

IF ADDR>500 THEN 001:
E1:
PUT 8:
RETURN:

E2:

E3:

E4:

E5:

/* ADD */
X=A(0)
Y=ACC;
ACC=ACC+ACC:
RETURN;

/* ADC */
IF MOT(MM), ADDR MODE > THEN ADDR=ADDR+AD2:
IF ACC_CODE THEN 001:
X=ACC-BIN(C): Y=ADDR:
ACC=ADDR+BIN(C):
XY=(ACC+B):
ELSE 001:
X=ACC:
Y=ADDR:
ACC=ACC+ADDR+BIN(C):
XY=(ACC+B):
RETURN;

/* ADD */
IF MOT(MM), ADDR MODE > THEN ADDR=ADDR+AD2:
IF ACC_CODE THEN 001:
X=ACC:
Y=ADDR:
ACC=ACC+ADDR+BIN(C):
XY=(ACC+B):
ELSE 001:
X=ACC:
Y=ADDR:
ACC=ACC+ADDR+BIN(C):
XY=(ACC+B):
RETURN;

/* AND */
IF ACC_CODE THEN 001:
ACC=DEC(BIT(ACC) & BIT(ADDR+2)):
XY=(ACC+B):
ELSE 001:
XY=(ACC+B):
RETURN;

/* ASL */
IF ADDR>0 THEN 001:
IF ACC_CODE THEN REL=ACC:
ELSE REL=ACC:
END:
ELSE ADDR=(ADDR+2):
REL=REL:
C=SUBSTR(RS,8,1):
SUBSTR(RS,7,1):SUBSTR(RS,9,7):
SUBSTR(RS,15,1)=0B1:
REL=REL:
XY=(REL):
CALL CODE(X+Y+X+1+++1+L%M+N%Z%V%C%CCX):
V=(REL):
(-N%):}

/* ADD */
IF ADDR>0 THEN 001:
IF ACC_CODE THEN ACC=REL:
ELSE ACC=REL:
MAIN: PROC OPTIONS(MAIN):

215 216
217
END
ELSE M(ADDR+2)=REL1
RETURN;

221
E6: /* ASR */
IF ADDR=0 THEN DO1 IF ACC_CODE THEN REL=ACC1
ELSE REL=ACC61
END:
ELSE REL=M(ADDR+2):
BS=BIT(HELI):
C=SUBSTR(25,15+1):
SUBSTR(25,8+7)=SUBSTR(25,9+1):
SUBSTR(25,8+1)=I
SYNT:
REL=DEC(25):
XY=REL CALL CODE(X+Y+XY+1+4+4+4+1+1+H+N+Z+V+C+ACCX):
V=(N&AC) | (~N&AC):
IF ADDR=0 THEN DO1 IF ACC_CODE THEN ACC=REL:
ELSE ACC=REL1
END:
ELSE M(ADDR+2)=REL1
RETURN1

243
E7: /* ACC */
IF ADDR THEN PC=PC+2+ADDR1
RETURN1

246
E8: /* AC1 */
IF ADDR THEN PC=PC+2+ADDR1
RETURN1

249
E9: /* AEO */
IF 2 THEN PC=PC+2+ADDR1
RETURN1

252
E10: /* ARE */
IF (N&V) THEN PC=PC+2+ADDR1
RETURN1

255
E11: /* ART */
IF (Z) & (N&V) THEN PC=PC+2+ADDR1
RETURN1

258
E12: /* AMI */
IF (Z) & (~Z) THEN PC=PC+2+ADDR1
RETURN1

261
E13: /* ART */
IF ACC_CODE THEN CALL CODE(X+Y+XY+1+4+4+4+1+H+N+Z+V+C+ACCX):
RETURN1

264
E14: /* ALE */
IF (Z) & (~N-V) THEN PC=PC+2+ADDR1
RETURN1

267
E15: /* ALS */
IF (Z) THEN PC=PC+2+ADDR1
RETURN1

270
E16: /* ALT */
IF (Z) & (~N-V) THEN PC=PC+2+ADDR1
RETURN1

273
E17: /* AMI */
IF (Z) THEN PC=PC+2+ADDR1
RETURN1

276
E18: /* AM1 */
IF (Z) THEN PC=PC+2+ADDR1
RETURN1

279
E19: /* APL */
IF (N) THEN PC=PC+2+ADDR1
RETURN1

282
E20: /* ARP */
PC=PC+2+ADDR1
RETURN1
E21: /* B3 */
   PC=PC+1
   SP=SP-1
   RETURN

E22: /* SVC */
   IF V THEN PC=PC+ADD1
   RETURN

E23: /* SYS */
   IF V THEN PC=PC+ADD1
   RETURN

E24: /* CR1 */
   REL=ACC1-ACC81
   X=ACC81 Y=ACC91
   CALL CODE(X,Y,XY1,4,4,4,5,5,H,N,Z,V,C,ACC91)
   RETURN

E25: /* CLC */
   G=0
   RETURN

E26: /* CLD */
   IF ADD1=0 THEN DO1 IF ACC_CODE THEN DO1 ACCA=01
   XY=ACCA END1
   ELSE DO1 ACCB=01
   XY=ACCB END1
   END1
   ELSE DO1 M(ADDR2)=1
   XY=M(ADDR2) END1
   CALL CODE(X,Y,XY1,2,3,2,2,5,H,N,Z,V,C,ACC91)
   RETURN

E27: /* CLV */
   RETURN

E28: /* CLV */
   RETURN

E29: /* CMP */
   IF ACC_CODE THEN REL=ACCA
   ELSE REL=ACCB
   X=REL Y=ADD1
   RETURN

E30: /* CMP */
   IF ADD1=0 THEN DO1 IF ACC_CODE THEN DO1 ACCA=DEC(-BIT(ACCA))
   XY=ACCA END1
   ELSE DO1 ACCB=DEC(-BIT(ACCB))
   XY=ACCB END1
   END1
   ELSE DO1 M(ADDR2)=DEC(-BIT(M(ADDR2))) END1
   CALL CODE(X,Y,XY1,4,4,4,4,4,5,H,N,Z,V,C,ACC91)
   RETURN

E31: /* CPX */
   X=ADD1
   Y=ADD1
   CALL CODE(X,Y,XY1,1,2,2,3,3,4,5,5,H,N,Z,V,C,ACC91)
   RETURN

E32: /* DAA */
   CALL CODE(X,Y,XY1,2,2,3,3,7,7,H,N,Z,V,C,ACC91)
   IF ACC_CODE THEN ACCA=ACCA
   ELSE ACCA=ACC91
   RETURN

E33: /* DES */
   SP=SP-1
   RETURN

E34: /* DEX */
   RETURN

E35: /* DEX */
   RETURN
MAIN: PROC OPTIONS(MAIN)

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

378 E37: /* INC */
379 IF ADDR -= 0 THEN DO: 1 M ADDR 2 2 1 = M ADDR 2 2 1
380 IF ACC_CODE THEN DO: 1 ACCA ACCA = ACCA = 11
381 ELSE DO: 1 X = ACCA END;
382 ELSE DO: 1 X = ACCA ACCB = ACCB = 11
383 END;
384 CALL CODE(X+Y+XY+1,4+4+1,1+M+N+Z+V+C+ACCX)1
385 /* IF X = 127 THEN V = 1'B1 ELSE V = 0'B1 */
386 RETURN

387 E38: /* INS */
388 SP = SP - 1
389 RETURN

390 E39: /* INX */
391 SP = SP - 1
392 RETURN

393 E40: /* JUMP */
394 PC = ADDR:
395 RETURN

396 E33: /* DEC */
397 IF ADDR = 0 THEN DO: 1 M ADDR 2 2 1 = M ADDR 2 2 1
398 IF ACC_CODE THEN DO: 1 X = ACCA END;
399 ELSE DO: 1 X = ACCA ACCB = ACCB = 11
400 END;
401 CALL CODE(X+Y,XY+1,4+4+1,1+M+N+Z+V+C+ACCX)1
402 /* IF XIN(X) > 80 THEN V = 1'B1 ELSE V = 0'B1 */
403 RETURN

404 E35: /* NEXT */
405 IF ADDR = 0 THEN DO: 1 M ADDR 2 2 1 = M ADDR 2 2 1
406 IF ACC_CODE THEN DO: 1 X = ACCA END;
407 ELSE DO: 1 X = ACCA ACCB = ACCB = 11
408 END;
409 CALL CODE(X+Y,XY+1,4+4+1,1+M+N+Z+V+C+ACCX)1
410 /* IF XIN(X) > 80 THEN V = 1'B1 ELSE V = 0'B1 */
411 RETURN

412 E41: /* JSR */
413 EA = SP - 1
414 SP = SP - 1
415 PC = ADDR:
416 RETURN

417 E42: /* LDS */
418 IF NOT(MM) ADDR MODE = 4 THEN ADDR = M ADDR 2 2 1
419 IF ACC_CODE THEN DO: 1 ACCA = ACCA END;
420 ELSE DO: 1 ACCA ACCB = ACCB = 11
421 END;
422 CALL CODE(X+Y,XY+1,4+4+2+1+M+N+Z+V+C+ACCX)1
423 RETURN

424 E43: /* LDS */
425 IF NOT(MM) ADDR MODE = 4 THEN ADDR = M ADDR 2 2 1
426 SP = SP - 1
427 IF ACC_CODE THEN DO: 1 X = ACCA END;
428 ELSE DO: 1 X = ACCA ACCB = ACCB = 11
429 END;
430 CALL CODE(X+Y,XY+1,4+4+2+1+M+N+Z+V+C+ACCX)1
431 RETURN

432 /* IF NOT(MM) ADDR MODE = 4 THEN ADDR = M ADDR 2 2 1
433 IF NOT(MM) ADDR MODE = 4 THEN ADDR = M ADDR 2 2 1
434 CALL CODE(X+Y,XY+1,4+4+2+1+M+N+Z+V+C+ACCX)1
435 RETURN

436 E44: /* LDX */
437 IF NOT(MM) ADDR MODE = 4 THEN ADDR = M ADDR 2 2 1
438 END:
439 IF NOT ADDR = M ADDR 2 2 1
440 CALL CODE(X+Y,XY+1,4+4+2+1+M+N+Z+V+C+ACCX)1
441 RETURN

442 /* IF NOT(MM) ADDR MODE = 4 THEN ADDR = M ADDR 2 2 1
443 IF NOT(MM) ADDR MODE = 4 THEN ADDR = M ADDR 2 2 1
444 CALL CODE(X+Y,XY+1,4+4+2+1+M+N+Z+V+C+ACCX)1
445 RETURN

446 /* IF NOT(MM) ADDR MODE = 4 THEN ADDR = M ADDR 2 2 1
447 IF NOT ADDR = M ADDR 2 2 1
448 CALL CODE(X+Y,XY+1,4+4+2+1+M+N+Z+V+C+ACCX)1
449 RETURN

450 /* IF NOT(MM) ADDR MODE = 4 THEN ADDR = M ADDR 2 2 1
451 IF NOT ADDR = M ADDR 2 2 1
452 CALL CODE(X+Y,XY+1,4+4+2+1+M+N+Z+V+C+ACCX)1
453 RETURN

454 /* IF NOT(MM) ADDR MODE = 4 THEN ADDR = M ADDR 2 2 1
455 IF NOT ADDR = M ADDR 2 2 1
456 CALL CODE(X+Y,XY+1,4+4+2+1+M+N+Z+V+C+ACCX)1
457 RETURN

458 /* IF NOT(MM) ADDR MODE = 4 THEN ADDR = M ADDR 2 2 1
459 IF NOT ADDR = M ADDR 2 2 1
460 CALL CODE(X+Y,XY+1,4+4+2+1+M+N+Z+V+C+ACCX)1
461 RETURN

462 /* IF NOT(MM) ADDR MODE = 4 THEN ADDR = M ADDR 2 2 1
463 IF NOT ADDR = M ADDR 2 2 1
464 CALL CODE(X+Y,XY+1,4+4+2+1+M+N+Z+V+C+ACCX)1
465 RETURN

466 /* IF NOT(MM) ADDR MODE = 4 THEN ADDR = M ADDR 2 2 1
467 IF NOT ADDR = M ADDR 2 2 1
468 CALL CODE(X+Y,XY+1,4+4+2+1+M+N+Z+V+C+ACCX)1
469 RETURN
MAIN: PROC OPTIONS(MAIN)

E45: /* LSP */
IF ADDR=0 THEN DO1 IF ACC_CODE THEN REL=ACCA1
   ELSE REL=ACCB1
END1
ELSE REL=M(ADDR+2)1
   ARITH REL11
CS=SUBSTR(8S+15,1)1
SUBSTR(8S+9,7)=SUBSTR(8S+9,7)1
SUBSTR(8S+1,9)=M REL11
REL=DEC(BS)1
X=REL1 CALL CODE(X+Y+X+12+1+1+H+N+Z+V+C+ACCX)1
V(NAC)1 (NKC)1
IF ADDR=0 THEN DO1 IF ACC_CODE THEN ACCA=REL1
   ELSE ACCB=REL1
END1
ELSE M(ADDR+2)=REL1
RETURN1

E46: /* NEQ */
IF ADDR=0 THEN DO1 X=ACCA1 ACCA=X=ACCA1
   ELSE DO1 X=ACCB1 ACCB=X=ACCB1
END1
ELSE DO1 X=M(ADDR+2)1 M(ADDR+2)=M(ADDR+2)1
   X=M(ADDR+2)1 END1
CALL CODE(X+Y+X+12+1+1+H+N+Z+V+C+ACCX)1
IF X=0 THEN V=1181 ELSE V=1081
IF X=0 THEN C=1181 ELSE C=1081
RETURN1

E47: /* NDR */
RETURN1

E48: /* ADR */
IF ACC_CODE THEN DO1 M(ADDR)=ACC1 BIT(ACCA)1 BIT(M(ADDR+2)11)
   ELSE DO1 M(ADDR)=ACC1 BIT(ACCB)1 BIT(M(ADDR+2)11)
CALL CODE(X+Y+X+12+1+1+H+N+Z+V+C+ACCX)1
RETURN1

E49: /* PSH */
IF ACC_CODE THEN M(SP+2)=ACC1
   ELSE M(SP+2)=ACCB1
SP=SP+11
RETURN1

E50: /* PUL */
SP=SP-11
IF ACC_CODE THEN ACCA=M(SP+2)1
   ELSE ACCB=M(SP+3)1
RETURN1

E51: /* ROL */
IF ADDR=0 THEN DO1 IF ACC_CODE THEN REL=ACCA1
   ELSE REL=ACCB1
END1
ELSE REL=M(ADDR+2)1
   ARITH REL11
CS=SUBSTR(8S+9,1)1
SUBSTR(8S+9,7)=SUBSTR(8S+9,7)1
SUBSTR(8S+1,9)=M REL11
REL=DEC(BS)1
X=REL1 CALL CODE(X+Y+X+12+1+1+H+N+Z+V+C+ACCX)1
V(NAC)1 (NKC)1
IF ADDR=0 THEN DO1 IF ACC_CODE THEN ACCA=REL1
   ELSE ACCB=REL1
END1
ELSE M(ADDR+2)=REL1
RETURN1

E521 /* ROR */
IF ADDR=0 THEN DO1 IF ACC_CODE THEN REL=ACCA1
   ELSE REL=ACCB1
END1
ELSE M(ADDR+2)=REL1
MAIN: PROC OPTIONS (MAIN):

AS=AEIT (REL);  
SUBSTR (BS, 7) = SUBSTR (BS, 8) + 1;  
SUBSTR (BS, 8) = CI;  
REL = ACC (BS);  
XY = REL; CALL CODE (X, Y, X, Y, 1, 4, 4, 3, 1, H, N, Z, V, C, ACCX);  
V = (ACC) + 1;  
IF ADDR = 0 THEN DO: IF ACC_CODE THEN ACCA = REL; ELSE ACCB = REL;  
ENDI

ELSE M (ADDR + 2) = REL;  
RETURN;

D4:

ES4: /* STS */  
SP = SP + 2;  
PE = (SP + 2);  
E = &SPI;  
RETURN;

ES5: /* SBA */  
ACCA = ACCA + ACCB;  
RETURN;

ES6: /* SEC */  
IF MOT (MM), ADDR MODE > 4 THEN ADDR = M (ADDR + 2);  
IF ACC_CODE THEN ACCA = ACCA - ADDR - CI; ELSE ACCB = ACCB - ADDR - CI;  
RETURN;

ES7: /* SEC */  
RETURN;

ES9: /* SEV */  
V = M (13);  
RETURN;

E0: /* STA */  
IF ACC_CODE THEN DO: M (ADDR + 2) = ACCA;  
XY = ACCA;  
ENDI  
E01: /* STA */  
IF ACC_CODE THEN ACCA = ACCA + 1;  
XY = ACCA;  
ELSE  
E02: /* STA */  
RETURN;

E1: /* STA */  
M (ADDR + 2) = &SPI;  
XY = SP; CALL CODE (X, Y, X, Y, 1, 4, 4, 2, 1, H, N, Z, V, C, ACCX);  
RETURN;

E2: /* STA */  
CALL CODE (X, Y, X, Y, 1, 4, 4, 2, 1, H, N, Z, V, C, ACCX);  
RETURN;

E3: /* SBR */  
IF MOT (MM), ADDR MODE > 4 THEN ADDR = M (ADDR + 2);  
IF ACC_CODE THEN ACCA = ACCA - ADDR;  
ELSE  
RETURN;

E4: /* SBR */  
ACC = ACCA;  
RETURN;

E5: /* TAA */  
ACCA = ACCA;  
XY = ACCA; CALL CODE (X, Y, X, Y, 1, 4, 4, 2, 1, H, N, Z, V, C, ACCX);  
RETURN;

E6: /* TAP */  
AS=AIT (ACC);  
CC = SUBSTR (BS);  
W = SUBSTR (CC, 1);  
= SUBSTR (CC, 2);  
= SUBSTR (CC, 3);  
RETURN;

E7: /* TAA */  
ACCA = ACCB;  
XY = ACCA; CALL CODE (X, Y, X, Y, 1, 4, 4, 2, 1, H, N, Z, V, C, ACCX);  
RETURN.
MAIN: PROC OPTIONS(MAIN)

649 E48: /* TP4 */
CCM [I | I | N | I | Z | V | C]
650 SUBTR(S+Y+2)+11141
651 SUBTR(S+1)+CCI
652 ACC=DEC(S+5)
653 RETURN

654 E59: /* TST */
655 IF ADDR=0 THEN DO1: IF ACC_CODE THEN XY=ACCA
656 ELSE XY=ACCB
657 END
658 ELSE
659 XY=ADDR+2)
660 CALL CODE(X,Y,X+1,X+2+Z+N+Z+V+C+ACC)
661 RETURN

663 E70: /* TSX */
664 X=SP-1
665 RETURN

667 E71: /* TIX */
668 SP=1X-1
669 RETURN

667 E99: RETURN:

ADDRESS: PROC:
670 DCL ONE FIXED ATN: XX BIT(15)
671 DCL LAL1 | 1 LABEL
672 LAL1 | 1 LAL2
673 LAL1 | 1 LAL4
674 LAL1 | 1 LAL6
675 LAL1 | 1 LAL8
676 LAL1 | 1 LAL10
677 LAL1 | 1 LAL11
678 LAL1 | 1 LAL2
679 IF NOT(MM) AND ACC=1 THEN ACC_CODE=1
680 ELSE IF (NOT(MM) AND ACC=2) THEN ACC_CODE=0+1;
681 GO TO LBL1 | NOT(MM) AND ACC=2
682 A1T: /* INHERENT */
683 RETURN

686 A2: /* RELATIVE */
687 IF BYTE2 <= 240 THEN BYTE2 = (255-BYTE2+1) + (-11)
688 ADDR=BYTE2
689 RETURN

691 A3: /* IMMEDIATE */
692 IF NOT(MM) AND NOT NO = 31 | NOT(MM) AND NOT NO = 4 |
693 IF NOT(MM) AND NOT NO = 44 THEN EA=SPC+1
694 ELSE EA=BYTE2
695 RETURN

696 A4: /* ACCUMULATOR */
697 EA=SPC;
698 ADDR=1
699 RETURN

699 A5: /* INDIRECT */
700 EA=BYTE2+1X;
701 ADDR=BYTE2
702 RETURN

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

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

708 END ADDRESS

48
710  
711  GO MM=1 BY 1 TO 1971
712  IF MM+1=NOT(MM).OP THEN GO TO LK11
713  END
714  LK11  END LKUPI

715  CODE: PROC((XY,YYYY,XX+YY+XX+XY+CC) 1)
716  DCL (H+2,V+2) BIT(11);  DCL (T+4,T+5) FIXED BIN
717  DCL (X+Y,XY,A..CCX) LIT(15) FIXED BIN
718  DCL (XY+YY,YYYY+,A..CCX) LIT(15) FIXED BIN
719  DCL (L+6).LIT(15) AT(5) 1
720  DCL (L+6).LIT(15) AT(5) 1
721  DCL (L+6).LIT(15) AT(5) 1
722  DCL (L+6).LIT(15) AT(5) 1
723  DCL (L+6).LIT(15) AT(5) 1
724  DCL (L+6).LIT(15) AT(5) 1
725  DCL (L+6).LIT(15) AT(5) 1
726  DCL (L+6).LIT(15) AT(5) 1
727  DCL (L+6).LIT(15) AT(5) 1
728  DCL (L+6).LIT(15) AT(5) 1
729  DCL (L+6).LIT(15) AT(5) 1
730  DCL (L+6).LIT(15) AT(5) 1
731  DCL (L+6).LIT(15) AT(5) 1
732  DCL (L+6).LIT(15) AT(5) 1
733  DCL (L+6).LIT(15) AT(5) 1
734  DCL (L+6).LIT(15) AT(5) 1
735  DCL (L+6).LIT(15) AT(5) 1
736  DCL (L+6).LIT(15) AT(5) 1
737  DCL (L+6).LIT(15) AT(5) 1
738  DCL (L+6).LIT(15) AT(5) 1
739  DCL (L+6).LIT(15) AT(5) 1
740  DCL (L+6).LIT(15) AT(5) 1
741  DCL (L+6).LIT(15) AT(5) 1
742  DCL (L+6).LIT(15) AT(5) 1
743  DCL (L+6).LIT(15) AT(5) 1
744  DCL (L+6).LIT(15) AT(5) 1
745  DCL (L+6).LIT(15) AT(5) 1
746  DCL (L+6).LIT(15) AT(5) 1
747  DCL (L+6).LIT(15) AT(5) 1
748  DCL (L+6).LIT(15) AT(5) 1
749  DCL (L+6).LIT(15) AT(5) 1
750  DCL (L+6).LIT(15) AT(5) 1
751  DCL (L+6).LIT(15) AT(5) 1
752  DCL (L+6).LIT(15) AT(5) 1
753  DCL (L+6).LIT(15) AT(5) 1
754  DCL (L+6).LIT(15) AT(5) 1
755  DCL (L+6).LIT(15) AT(5) 1
756  DCL (L+6).LIT(15) AT(5) 1
757  DCL (L+6).LIT(15) AT(5) 1
758  DCL (L+6).LIT(15) AT(5) 1
759  DCL (L+6).LIT(15) AT(5) 1
760  DCL (L+6).LIT(15) AT(5) 1
761  DCL (L+6).LIT(15) AT(5) 1
762  DCL (L+6).LIT(15) AT(5) 1
763  DCL (L+6).LIT(15) AT(5) 1
764  DCL (L+6).LIT(15) AT(5) 1
765  DCL (L+6).LIT(15) AT(5) 1
766  DCL (L+6).LIT(15) AT(5) 1
767  DCL (L+6).LIT(15) AT(5) 1
768  DCL (L+6).LIT(15) AT(5) 1
769  DCL (L+6).LIT(15) AT(5) 1
770  DCL (L+6).LIT(15) AT(5) 1
771  DCL (L+6).LIT(15) AT(5) 1
772  DCL (L+6).LIT(15) AT(5) 1
773  DCL (L+6).LIT(15) AT(5) 1
774  DCL (L+6).LIT(15) AT(5) 1
775  DCL (L+6).LIT(15) AT(5) 1
776  DCL (L+6).LIT(15) AT(5) 1
777  DCL (L+6).LIT(15) AT(5) 1
778  DCL (L+6).LIT(15) AT(5) 1
779  DCL (L+6).LIT(15) AT(5) 1
780  DCL (L+6).LIT(15) AT(5) 1
781  DCL (L+6).LIT(15) AT(5) 1
782  DCL (L+6).LIT(15) AT(5) 1
783  DCL (L+6).LIT(15) AT(5) 1
784  DCL (L+6).LIT(15) AT(5) 1
785  DCL (L+6).LIT(15) AT(5) 1
786  DCL (L+6).LIT(15) AT(5) 1
787  DCL (L+6).LIT(15) AT(5) 1
788  DCL (L+6).LIT(15) AT(5) 1
789  DCL (L+6).LIT(15) AT(5) 1
790  DCL (L+6).LIT(15) AT(5) 1
791  DCL (L+6).LIT(15) AT(5) 1
792  DCL (L+6).LIT(15) AT(5) 1
793  DCL (L+6).LIT(15) AT(5) 1
794  DCL (L+6).LIT(15) AT(5) 1
795  DCL (L+6).LIT(15) AT(5) 1
796  DCL (L+6).LIT(15) AT(5) 1
797  DCL (L+6).LIT(15) AT(5) 1

719  CC1: GO TO LBL3(C1);  CC2: GO TO LBL4(C2);
720  C3: GO TO LBL5(C3);  CC4: GO TO LBL6(C4);
721  CC5: GO TO LBL7(C5);  C6: GO TO LBL8(C6);
722  C7: GO TO LBL9(C7);  C8: GO TO LBL10(C8);
723  C9: GO TO LBL11(C9);  CC10: GO TO LBL12(C10);
724  CC11: GO TO LBL13(C11);  CC12: GO TO LBL14(C12);
725  CC13: GO TO LBL15(C13);  CC14: GO TO LBL16(C14);
726  CC15: GO TO LBL17(C15);  CC16: GO TO LBL18(C16);
727  CC17: GO TO LBL19(C17);  CC18: GO TO LBL20(C18);
728  CC19: GO TO LBL21(C19);  CC20: GO TO LBL22(C20);
729  CC21: GO TO LBL23(C21);  CC22: GO TO LBL24(C22);
730  CC23: GO TO LBL25(C23);  CC24: GO TO LBL26(C24);
731  CC25: GO TO LBL27(C25);  CC26: GO TO LBL28(C26);
732  CC27: GO TO LBL29(C27);  CC28: GO TO LBL30(C28);
MAIN: PROC OPTIONS(MAIN);  

VP2:  
GO TO CCS;  

CC5:  
GO TO CALT(C4);  

CC9:  
GO TO CCS;  

CCS:  
GO TO CCS;  

CP1:  
C(E(SUBSTR(X+1) & SUBSTR(Y,X+1)));  

CP2:  
C(SUBSTR(X+1) & SUBSTR(Y,X+1));  

CP:  
C((SUBSTR(X+1) & SUBSTR(Y,X+1)));  

CC6:  
END CODE;  

CONVERT: PROC(A) RETURNS(CHAR(16)VAR);  

CON: PROC(CHARREP) RETURNS(FIXED BIN(15));  

CREATE: PROC(DEST) GO TO INEND;  

CREATE?: PROC(DEST) GO TO INEND;}
MAIN: PROC OPTIONS(MAIN);

890 PUT EDIT('SIMULATOR FOR THE MOTOROLA M6800 MICROPROCESSOR', 'WRITTEN FOR THE IBM 360', 'VERSION 1', 1976);

891 (COL(5) = A SKIP, COL(15) = A SKIP, COL(15) = A);

892 PUT SKIP;

893 MINT: CALL CREATE1;

894 CALL CREATE2;

895 MSTR1:

896 PUT SKIP;

897 PUT EDIT('?', COL(1) = A(1));

898 PUT SKIP;

899 READ FILE(SYSIN) INTO BUFFER;

900 VERIFY BUFFER, 1;

901 BUFFER = SUBSTR BUFFER, 10;

902 GET STRING BUFFER, EDIT(Y0) (A(2));

903 INDEX BUFFER, 1;

904 BUFFER = SUBSTR BUFFER, 10;

905 INDEX COMMAND Y0;

906 IF ZO = 0 THEN DO1;

907 PUT SKIP;

908 PUT EDIT('** ERROR 301 **', 'UNDEFNEO SIMULATOR COMMAND');

909 (COL(1) = A(15), COL(17) = A(2), SKIP, COL(1) = A);

910 PUT SKIP;

911 GO TO MSTR1;

912 END;

913 ZO = (Z0 + 1) / 21;

914 LAL3(1) = M1;

915 LAL3(2) = M2;

916 LAL3(3) = M3;

917 LAL3(4) = M4;

918 LAL3(5) = M5;

919 LAL3(6) = M6;

920 LAL3(7) = M7;

921 GO TO LAL3(201);

922 M1:

923 # SET MEMORY # /

924 ZO = VERIFY BUFFER, 1;

925 BUFFER = SUBSTR BUFFER, 10;

926 GET STRING BUFFER, EDIT (CHARP) (A(4));

927 INDEX BUFFER, 1;

928 BUFFER = SUBSTR BUFFER, 10;

929 GET STRING BUFFER, LIST(Z2);

930 INDEX COMMAND Y1;

931 CON (CHARP);

932 GET STRING BUFFER, LIST(Z2);

933 IF Z0 = 0 THEN Z0 = Z2 / 21;

934 M3:

935 GET STRING BUFFER, LIST(Z2);

936 IF Z0 = 0 THEN Z0 = Z2 / 21;

937 IF Z0 = 0 THEN M3;

938 ELSE M3;

939 Z0 = (Z0 + 1) / 21;

940 IF Z0 = 0 THEN M3;

941 M3_1:

942 PUT EDIT('M3(D3) DO Z3=Z1 TO Z2) ;(20) (12 F(5), SKIP)');

943 GO TO M3_1;

944 M3_2:

945 /* SET REGISTERS */

946 VERIFY BUFFER, 1;

947 BUFFER = SUBSTR BUFFER, 10;

948 GET STRING BUFFER, EDIT(Y1) (A(1));

949 INDEX BUFFER, 1;

950 BUFFER = SUBSTR BUFFER, 10;

951 VERIFY BUFFER, 1;

952 BUFFER = SUBSTR BUFFER, 10;

953 INDEX REGISTER Y1;

954 IF ZO = 0 THEN GO TO M3_21;

955 M3_21:

956 GET STRING BUFFER, EDIT (CHARP) (A(4));

957 IF Z0 = 0 THEN PC = CON CHARP;

958 IF Z0 = 0 THEN SP = CON CHARP;

959 IF PC > 5000 THEN PC = 5000;

960 IF SP > 5000 THEN SP = 5000;

961 PUT SKIP;
MAIN: PROC OPTIONS(MAIN)

946 PUT EDIT('**** ERROR 306 'Y1'),
*THE OPERAND IN THE SIMULATOR COMMAND '*
' CAUSED A REGISTER OVERFLOW')
(COL(1), (15), COL(17), (11), SKIP, COL(1), (8), SKIP, COL(1), (9))
947 PUT SKIP
948 ENO1
949 ENO1
950 END

M3_2: GET STRING BUFF) LIST(Z1)
951 IF 70=5 THEN ACCA=Z11
952 IF 70=7 THEN ACCB=Z11
953 GO TO M3_1

M3_3: GET STRING(BUFFER) EDIT(CC)(R16)!
954 #=SUBSTR(CC+1,1)!
955 v=SUBSTR(CC,5,1)!
956 z=SUBSTR(CC,4,1)!
957 GO TO M3_1

M4: /* RUN */
958 GET STRING(BUFFER) LIST(Z1)
959 ON Z2= 1 TO Z11
960 CALL SIM
961 IF ERR THEN GO TO M3_1
962 GO TO M3_1

M5: /* SET HEADER COUNT */
963 GET STRING(BUFFER) LIST(Z1)
964 SHC=1
965 GO TO M3_1

M6: /* DISPLAY REGISTERS */
966 CALL PRINT
967 GO TO M3_1

M7: /* EXIT */
END MAIN
BIBLIOGRAPHY


