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>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>II</td>
<td>CHAPTER 1</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>SIM6800 Structure</td>
<td></td>
</tr>
<tr>
<td>III</td>
<td>CHAPTER 2</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td>SIM6800 Coding</td>
<td></td>
</tr>
<tr>
<td>IV</td>
<td>CONCLUSIONS AND SUGGESTIONS FOR FUTURE WORK</td>
<td>19</td>
</tr>
<tr>
<td>V</td>
<td>APPENDIX A</td>
<td>22</td>
</tr>
<tr>
<td></td>
<td>SIM6800 User's Guide for the Motorola M6800 Microprocessor</td>
<td></td>
</tr>
<tr>
<td>VI</td>
<td>APPENDIX B</td>
<td>34</td>
</tr>
<tr>
<td></td>
<td>Sample Output from the SIM6800 Simulator Program</td>
<td></td>
</tr>
<tr>
<td>VII</td>
<td>APPENDIX C</td>
<td>40</td>
</tr>
<tr>
<td></td>
<td>Computer Listing of the SIM6800 Simulator Program</td>
<td></td>
</tr>
<tr>
<td>VIII</td>
<td>BIBLIOGRAPHY</td>
<td>53</td>
</tr>
</tbody>
</table>
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 simu­
lator, 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 lan­
guage 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 simu­
lator 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 *
170 ORG $1000
180 LDA B #8
190 LDA #P LOADS INDEX REGISTER
200 * WITH THE ADDRESS OF THE
210 * MOST SIGNIFICANT BYTE
220 * OF P
230 *
240 CLC
250 NEXT LDA A 7,X
260 ADC A 15,X
270 DAA
280 STA A 23,X
290DEX
300 DEC B
310 BNE NEXT LOOPS BACK FOR NEXT
320 * BYTE IF ADDITION NOT COMPLETED.
330 NOP
340 BRA #-1
350 * ALLOCATE A DATA AREA IN READ-WRITE
360 * MEMORY FOR THE NUMBERS TO BE ADDED
370 * (P AND Q) AND FOR THE SUM (RES)
380 ORG $0100
390 P RMB 8
400 Q RMB 8
410 RES RMB 8
420 END
430 MNY

Fig. 3.--ASSEMBLER LANGUAGE PROGRAM
Fig. 4.—ASSEMBLED PROGRAM
Fig. 5.--OBJECT CODE FILE EXAMPLE1
## Component PL/1 Variable Names

<table>
<thead>
<tr>
<th>1. Instructional address</th>
<th>BYTE1</th>
</tr>
</thead>
<tbody>
<tr>
<td>2. Operands</td>
<td>BYTE2</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>MOT Entry</th>
<th>PL/1 Variable Names</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Operator number</td>
<td>OP</td>
</tr>
<tr>
<td>2. Mnemonic code</td>
<td>MNEMONIC</td>
</tr>
<tr>
<td>3. Address mode</td>
<td>ADDR_MODE</td>
</tr>
<tr>
<td>4. Time</td>
<td>TIME</td>
</tr>
<tr>
<td>5. Bytes</td>
<td>BYTES</td>
</tr>
<tr>
<td>6. MOT number</td>
<td>MOT_NO</td>
</tr>
<tr>
<td>7. Address accumulator</td>
<td>ADDR_ACC</td>
</tr>
</tbody>
</table>

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
12

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:
<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.
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.

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.
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

\[
\text{SM } a \quad n \quad v_1 \quad v_2 \quad \ldots \quad v_n
\]

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

\[
\text{SR } \quad r \quad v
\]

The SR command sets register \( r \) to the value \( v \).
The codes for the \( r \) operand with corresponding \( v \) input bases are:

\[
\begin{array}{ll}
\text{r} & \text{v} \\
\hline
P & \text{Program counter} \\
X & \text{Index register} \\
A & \text{Accumulator A} \\
B & \text{Accumulator B} \\
C & \text{Condition codes} \\
S & \text{Stack pointer} \\
T & \text{Time} \\
\end{array}
\]

Hexadecimal

Integer

Integer

Bit-string*

Hexadecimal

Integer

* 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>
</table>
| 301          | **** ERROR 301 AAAAA  
Undefined simulator command. |
| 302          | **** ERROR 302 AAAAA  
Possible syntax error in simulator command. |
| 306          | **** ERROR 306 AAAAA  
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:

\[
\begin{align*}
1357902468097531 \\
+9258147036741852 \\
0616049504839383 (result)
\end{align*}
\]

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 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**

```plaintext
? sm 0102 8 13 57 90 24 63 93 75 31
?
? sm 010a 8 92 53 14 70 36 74 13 52
?
? sm 0112 0 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>LDS</td>
<td>1102</td>
<td>1103</td>
<td>0000</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1103</td>
<td>LDX</td>
<td>1105</td>
<td>1106</td>
<td>0102</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1109</td>
<td>STX</td>
<td>0101</td>
<td>1109</td>
<td>0102</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</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>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1100</td>
<td>LDA</td>
<td>1001</td>
<td>1002</td>
<td>0102</td>
<td>0</td>
<td>0</td>
<td>8</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1102</td>
<td>LDX</td>
<td>0101</td>
<td>1005</td>
<td>0102</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1105</td>
<td>CLC</td>
<td>1005</td>
<td>1006</td>
<td>0102</td>
<td>0</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>0</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>1106</td>
<td>LDA</td>
<td>0109</td>
<td>1003</td>
<td>0102</td>
<td>31</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1107</td>
<td>ADD</td>
<td>0111</td>
<td>100A</td>
<td>0112</td>
<td>33</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>110A</td>
<td>DAA</td>
<td>100B</td>
<td>100B</td>
<td>0102</td>
<td>33</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1108</td>
<td>STA</td>
<td>0119</td>
<td>100D</td>
<td>0102</td>
<td>33</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>110B</td>
<td>DEX</td>
<td>100E</td>
<td>100E</td>
<td>0101</td>
<td>33</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>110D</td>
<td>DEC</td>
<td>100F</td>
<td>100F</td>
<td>0101</td>
<td>33</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>110E</td>
<td>BNE</td>
<td>1010</td>
<td>1006</td>
<td>0101</td>
<td>33</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</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>1006</td>
<td>LDA A</td>
<td>0108</td>
<td>1008</td>
<td>0101</td>
<td>75</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>013D 63</td>
</tr>
<tr>
<td>1008</td>
<td>ADC A</td>
<td>0110</td>
<td>100A</td>
<td>0101</td>
<td>93</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>013D 62</td>
</tr>
<tr>
<td>100A</td>
<td>DAA A</td>
<td>100A</td>
<td>100B</td>
<td>0101</td>
<td>93</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>013D 70</td>
</tr>
<tr>
<td>100B</td>
<td>STA A</td>
<td>0118</td>
<td>100D</td>
<td>0101</td>
<td>93</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>013D 76</td>
</tr>
<tr>
<td>100D</td>
<td>DEX A</td>
<td>100D</td>
<td>100E</td>
<td>0100</td>
<td>93</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>013D 82</td>
</tr>
<tr>
<td>100E</td>
<td>DEC B</td>
<td>100E</td>
<td>100F</td>
<td>0100</td>
<td>93</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>013D 86</td>
</tr>
<tr>
<td>100F</td>
<td>BNE</td>
<td>1010</td>
<td>100F</td>
<td>0100</td>
<td>93</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>013D 86</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>1006</td>
<td>LPA A</td>
<td>0107</td>
<td>1008</td>
<td>0100</td>
<td>0</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>013D 91</td>
</tr>
<tr>
<td>1008</td>
<td>ADC A</td>
<td>010F</td>
<td>100A</td>
<td>0100</td>
<td>83</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>013D 96</td>
</tr>
<tr>
<td>100A</td>
<td>DAA A</td>
<td>100A</td>
<td>100B</td>
<td>0100</td>
<td>83</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>013D 96</td>
</tr>
<tr>
<td>100B</td>
<td>STA A</td>
<td>0117</td>
<td>100D</td>
<td>0100</td>
<td>83</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>013D 104</td>
</tr>
<tr>
<td>100D</td>
<td>DEX A</td>
<td>100D</td>
<td>100E</td>
<td>00FF</td>
<td>83</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>013D 108</td>
</tr>
<tr>
<td>100E</td>
<td>DEC B</td>
<td>100E</td>
<td>100F</td>
<td>00FF</td>
<td>83</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>013D 110</td>
</tr>
<tr>
<td>100F</td>
<td>BNE</td>
<td>1010</td>
<td>100E</td>
<td>00FF</td>
<td>83</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>013D 114</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>1006</td>
<td>LPA A</td>
<td>0107</td>
<td>1008</td>
<td>0100</td>
<td>0</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>013D 91</td>
</tr>
<tr>
<td>1008</td>
<td>ADC A</td>
<td>010F</td>
<td>100A</td>
<td>0100</td>
<td>83</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>013D 96</td>
</tr>
<tr>
<td>100A</td>
<td>DAA A</td>
<td>100A</td>
<td>100B</td>
<td>0100</td>
<td>83</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>013D 96</td>
</tr>
<tr>
<td>100B</td>
<td>STA A</td>
<td>0117</td>
<td>100D</td>
<td>0100</td>
<td>83</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>013D 104</td>
</tr>
<tr>
<td>100D</td>
<td>DEX A</td>
<td>100D</td>
<td>100E</td>
<td>00FF</td>
<td>83</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>013D 108</td>
</tr>
<tr>
<td>100E</td>
<td>DEC B</td>
<td>100E</td>
<td>100F</td>
<td>00FF</td>
<td>83</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>013D 110</td>
</tr>
<tr>
<td>100F</td>
<td>BNE</td>
<td>1010</td>
<td>100E</td>
<td>00FF</td>
<td>83</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>013D 114</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>0105</td>
<td>1008</td>
<td>00FF</td>
<td>68</td>
<td>5</td>
<td>000000</td>
<td>013D</td>
</tr>
<tr>
<td>1003</td>
<td>ADC</td>
<td>A</td>
<td>010E</td>
<td>100A</td>
<td>00FF</td>
<td>104</td>
<td>5</td>
<td>000000</td>
<td>013D</td>
</tr>
<tr>
<td>100A</td>
<td>DAA</td>
<td>A</td>
<td>100A</td>
<td>100B</td>
<td>00FF</td>
<td>4</td>
<td>5</td>
<td>000000</td>
<td>013D</td>
</tr>
<tr>
<td>100B</td>
<td>STA</td>
<td>A</td>
<td>0116</td>
<td>100D</td>
<td>00FF</td>
<td>4</td>
<td>5</td>
<td>000000</td>
<td>013D</td>
</tr>
<tr>
<td>100D</td>
<td>DEX</td>
<td>100D</td>
<td>100E</td>
<td>00FE</td>
<td>4</td>
<td>5</td>
<td>000000</td>
<td>013D</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>5</td>
<td>000000</td>
<td>013D</td>
</tr>
<tr>
<td>100F</td>
<td>BNE</td>
<td>1010</td>
<td>1006</td>
<td>00FE</td>
<td>4</td>
<td>5</td>
<td>000000</td>
<td>013D</td>
<td></td>
</tr>
<tr>
<td>?</td>
<td>dm</td>
<td>0102</td>
<td>8</td>
<td>13</td>
<td>57</td>
<td>30</td>
<td>24</td>
<td>63</td>
<td>9</td>
</tr>
<tr>
<td>?</td>
<td>dm</td>
<td>010a</td>
<td>8</td>
<td>92</td>
<td>58</td>
<td>14</td>
<td>70</td>
<td>36</td>
<td>74</td>
</tr>
<tr>
<td>?</td>
<td>dm</td>
<td>0112</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>83</td>
</tr>
<tr>
<td>?</td>
<td>rn</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1A</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>1006</td>
<td>LDA</td>
<td>A</td>
<td>0105</td>
<td>1008</td>
<td>00FE</td>
<td>24</td>
<td>4</td>
<td>000000</td>
<td>013D</td>
</tr>
<tr>
<td>1003</td>
<td>ADC</td>
<td>A</td>
<td>010D</td>
<td>100A</td>
<td>00FE</td>
<td>95</td>
<td>4</td>
<td>000000</td>
<td>013D</td>
</tr>
<tr>
<td>100A</td>
<td>DAA</td>
<td>A</td>
<td>100A</td>
<td>100B</td>
<td>00FE</td>
<td>95</td>
<td>4</td>
<td>000000</td>
<td>013D</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>4</td>
<td>000000</td>
<td>013D</td>
</tr>
<tr>
<td>100D</td>
<td>DEX</td>
<td>100D</td>
<td>100E</td>
<td>00FD</td>
<td>95</td>
<td>4</td>
<td>000000</td>
<td>013D</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>3</td>
<td>000000</td>
<td>013D</td>
</tr>
<tr>
<td>100F</td>
<td>BNE</td>
<td>1010</td>
<td>1006</td>
<td>00FD</td>
<td>95</td>
<td>3</td>
<td>000000</td>
<td>013D</td>
<td></td>
</tr>
<tr>
<td>?</td>
<td>dm</td>
<td>0102</td>
<td>8</td>
<td>13</td>
<td>57</td>
<td>30</td>
<td>24</td>
<td>63</td>
<td>9</td>
</tr>
<tr>
<td>?</td>
<td>dm</td>
<td>010a</td>
<td>8</td>
<td>92</td>
<td>58</td>
<td>14</td>
<td>70</td>
<td>36</td>
<td>74</td>
</tr>
<tr>
<td>?</td>
<td>dm</td>
<td>0112</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>83</td>
</tr>
<tr>
<td>?</td>
<td>rn</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></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 A</td>
<td>0103</td>
<td>1002</td>
<td>00FC</td>
<td>57</td>
<td>2</td>
<td>000000C</td>
<td>013D</td>
<td>203</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>0000000V0</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>0000000C</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>0000000C</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>0000000C</td>
<td>013D</td>
<td>220</td>
</tr>
<tr>
<td>100E</td>
<td>DEC</td>
<td>100E</td>
<td>100F</td>
<td>00FB</td>
<td>16</td>
<td>1</td>
<td>0000000C</td>
<td>013D</td>
<td>222</td>
</tr>
<tr>
<td>100F</td>
<td>BNE</td>
<td>1010</td>
<td>1005</td>
<td>00FB</td>
<td>16</td>
<td>1</td>
<td>0000000C</td>
<td>013D</td>
<td>226</td>
</tr>
<tr>
<td>13</td>
<td>57</td>
<td>90</td>
<td>24</td>
<td>68</td>
<td>9</td>
<td>75</td>
<td>31</td>
<td></td>
<td></td>
</tr>
<tr>
<td>92</td>
<td>58</td>
<td>14</td>
<td>70</td>
<td>36</td>
<td>74</td>
<td>13</td>
<td>52</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>16</td>
<td>4</td>
<td>95</td>
<td>4</td>
<td>83</td>
<td>93</td>
<td>83</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<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>100S</td>
<td>LDA A</td>
<td>0102</td>
<td>1008</td>
<td>00FB</td>
<td>13</td>
<td>1</td>
<td>0000000C</td>
<td>013D</td>
<td>231</td>
</tr>
<tr>
<td>1003</td>
<td>ADC A</td>
<td>010A</td>
<td>100A</td>
<td>00FB</td>
<td>106</td>
<td>1</td>
<td>0000000C</td>
<td>013D</td>
<td>236</td>
</tr>
<tr>
<td>100A</td>
<td>DAA</td>
<td>100A</td>
<td>100B</td>
<td>00FB</td>
<td>6</td>
<td>1</td>
<td>0000000C</td>
<td>013D</td>
<td>238</td>
</tr>
<tr>
<td>100B</td>
<td>STA A</td>
<td>0112</td>
<td>100D</td>
<td>00FB</td>
<td>6</td>
<td>1</td>
<td>0000000C</td>
<td>013D</td>
<td>244</td>
</tr>
<tr>
<td>100D</td>
<td>DEX</td>
<td>100D</td>
<td>100E</td>
<td>00FA</td>
<td>6</td>
<td>1</td>
<td>0000000C</td>
<td>013D</td>
<td>248</td>
</tr>
<tr>
<td>100E</td>
<td>DEC</td>
<td>100E</td>
<td>100F</td>
<td>00FA</td>
<td>6</td>
<td>0</td>
<td>0000000Z0C</td>
<td>013D</td>
<td>250</td>
</tr>
<tr>
<td>100F</td>
<td>BNE</td>
<td>1010</td>
<td>1011</td>
<td>00FA</td>
<td>6</td>
<td>0</td>
<td>0000000Z0C</td>
<td>013D</td>
<td>254</td>
</tr>
<tr>
<td>13</td>
<td>57</td>
<td>90</td>
<td>24</td>
<td>68</td>
<td>9</td>
<td>75</td>
<td>31</td>
<td></td>
<td></td>
</tr>
<tr>
<td>92</td>
<td>58</td>
<td>14</td>
<td>70</td>
<td>36</td>
<td>74</td>
<td>13</td>
<td>52</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>15</td>
<td>4</td>
<td>95</td>
<td>4</td>
<td>83</td>
<td>93</td>
<td>83</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```python
READY
```
APPENDIX C

COMPUTER LISTING OF THE SIM6800 SIMULATOR PROGRAM
MAIN: PROC OPTIONS(MAIN);  
  DCL I, J, K, L, M, N, O, P, Q FIXED 8 IN;
  DCL A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q FIXED 8 IN;
  DCL LSL, RS, BC, DE, HI, LM, MN, NO, PL FIXED 8 IN;
  DCL STRA, STRB CHAR(16);
MAIN: PROC OPTIONS(MAIN):

LBL2(17)=E17: LBL2(34)=E34: LBL2(52)=E52: LBL2(70)=E70:

SPECS:
PC=PC MOT(MM), BYTES:
TL=T MOT(MM), TIME:
OC=MOT(MM), MNEMONIC:
CALL ADDRESS:

IF ADDR>5000 THEN 00:
EQR=1:
PUT START:
PUT EDIT(!***ERROR 313***CONVERT(UNSPEC(ADDR)),
(COL(14),A(14),COL(16),A(4),SKIP+,A)):
RETURN:
GOTO TO LBL2(MOT(MM), MOT_NO):

E1: /* ABA */
X=ACCA:
ACCA=ACCA+ACCB:
XY=ACCA:
CALL CODE(X,Y;XY,+++++A,H+N,Z,V+C;ACCS):
RETURN:

E2: /* ADD */
IF MOT(MM), ADDR MODE > THEN ADDR=ADDR+2:
IF ACC_CODE THEN 001 X=ACCA+BINC:
Y=ACCA:
ACCA=ACCA+ADDR+BINC:
XY=ACCA:
END:
ELSE 001 X=ACCA:
Y=ADDR:
ACCA=ADDR+BINC:
XY=ACCA:
END:
SACC_CODE=ACC_CODE:
CALL CODE(X,Y;XY,+++++A,H+N,Z,V+C;ACCS):
RETURN:

E3: /* ADD */
IF MOT(MM), ADDR MODE > THEN ADDR=ADDR+2:
IF ACC_CODE THEN 001 X=ACCA:
Y=ADDR:
ACCA=ADDR+BINC:
XY=ACCA:
END:
ELSE 001 X=ACCA:
Y=ADDR:
ACCA=ADDR+BINC:
XY=ACCA:
END:
CALL CODE(X,Y;XY,+++++A,H+N,Z,V+C;ACCS):
RETURN:

E4: /* END */
IF ACC_CODE THEN 001 ACCA=DEC(BIT(ACCA) & BIT(ADDR+2)):
X=ACCA:
END:
ELSE 001 X=ACCA:
Y=ADDR:
ACCA=ADDR+BINC:
XY=ACCA:
END:
CALL CODE(X,Y;XY,+++++A,H+N,Z,V+C;ACCS):
RETURN:

E5: /* ASL */
IF ADDR=0 THEN 001 IF ACC_CODE THEN 0FL=ACCA:
ELSE 0EL=ACCB:
END:
ELSE REL=(ADDR+2):
REL=REL:
C=SUBSTR(RS,8,1):
SUBSTR(RS,8,7)=SUBSTR(RS,9,7):
SUBSTR(RS,15,1)=0'B1:
REL=REL:
XY=REL:
CALL CODE(X,Y;XY,+++++A,H+N,Z,V+C;ACCS):
V=(N&C):
IF ADDR=0 THEN 001 IF ACC_CODE THEN ACCA=REL:
ELSE ACCB=REL:

42
MAIN: PROC OPTIONS(MAIN);
MAIN: PROC OPTIONS(MAIN)

E21: /* RSC */
PC=PC+2
SP=SP-1
RETURN

E22: /* SVC */
IF V THEN PC=PC+2+ADDR1
RETURN

E23: /* SVS */
IF V THEN PC=PC+2+ADDR1
RETURN

E24: /* CRA */
REL=ACCA-ADDR1
X=ACCA1 Y=ADDR1
CALL CODE(X,Y,XY,1,4,4,4,5,5,H,N,Z,V,C,ACX1)
RETURN

E25: /* CLC */
G=1
RETURN

E26: /* CLR */
IF ADDR=0 THEN DO1
IF ACC_CODE THEN DO1
ACC=01
XY=ACCA1 END1
ELSE DO1
ACC=001
XY=ACCB1 END1
ELSE DO1
M=(ADDR+2)+1
XY=M(ADDR+2)1 END1
CALL CODE(X,Y,XY,1,2,3,2,2,2,2,H,N,Z,V,C,ACX1)
RETURN

E27: /* CLD */
IF ADDR=0 THEN DO1
IF ACC_CODE THEN DO1
ACC=011
XY=ACCA1 END1
ELSE DO1
ACC=001
XY=ACCB1 END1
END1
ELSE DO1
M=(ADDR+2)+1
XY=M(ADDR+2)1 END1
CALL CODE(X,Y,XY,1,2,3,2,2,2,2,H,N,Z,V,C,ACX1)
RETURN

E28: /* CLV */
RETURN

E29: /* CMP */
IF ACC_CODE THEN REL=ACCA1
ELSE REL=ACCB1
X=REL Y=ADDR1
REL=REL-ADDR1
XY=REL CALL CODE(X,Y,XY,1,4,4,4,5,H,N,Z,V,C,ACX1)
RETURN

E30: /* CMX */
IF ADDR=0 THEN DO1
IF ACC_CODE THEN DO1
ACC=DEC(-BIT(ACCA))1
XY=ACCA1 END1
ELSE DO1
ACC=DEC(-BIT(ACCB))1
XY=ACCB1 END1
END1
ELSE DO1
M=(ADDR+2)+DEC(-BIT(M(ADDR+2)))1 END1
CALL CODE(X,Y,XY,1,4,4,4,1,H,N,Z,V,C,ACX1)
RETURN

E31: /* CPX */
X=1
Y=ADDR1
REL=X ADDR1
XY=REL CALL CODE(X,Y,XY,1,4,4,4,1,H,N,Z,V,C,ACX1)
RETURN

E32: /* DAA */
CALL CODE(X,Y,XY,1,4,4,4,2,7,H,N,Z,V,C,ACX1)
IF ACC_CODE THEN ACC=ACX1
ELSE ACC=ACX1
RETURN

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

E34: /* DEX */
RETURN

E35: /* DOR */
IF ACC_CODE THEN DO1
ACC=DEC(RIT(ACCA))1(BIT(M(ADDR+2)))1
XY=ACCA1 END1
ELSE DO1
ACC=DEC(RIT(ACCA))1(BIT(M(ADDR+2)))1
XY=ACCB1 END1

E36: /* DRE */
ACCA=DEC(RIT(ACCA))1(BIT(M(ADDR+2)))1
XY=ACCA1 END1
ELSE DO1
ACCB=DEC(RIT(ACCA))1(BIT(M(ADDR+2)))1
XY=ACCB1 END1

E37: /* DRS */
ACCA=DEC(RIT(ACCA))1(BIT(M(ADDR+2)))1
XY=ACCA1 END1
ELSE DO1
ACCB=DEC(RIT(ACCA))1(BIT(M(ADDR+2)))1
XY=ACCB1 END1
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 DO1 X=M(ADDR+2)1 M(ADDR+2)=M(ADDR+2)+1
380  IF ACC_CODE THEN DO1 X=ACC1 ACC1=ACC1+1
381  ELSE DO1 X=ACC1 ACC1=ACC1+1
382  END1
383  CALL CODE(X,Y,XY*1,4+4*2,1+M+N+Z+V+C,ACCX)1
384  IF X=127 THEN V=1191 ELSE V=1391

385  RETURN

386  E38: /* INS */
387  IF ADDR = 0 THEN DO1 X=INS1 INS1=INS1+1

388  RETURN

389  E39: /* INX */
390  IF ADDR = 0 THEN DO1 X=INX1 INX1=INX1+1

391  RETURN

392  E40: /* JUMP */
393  IF ADDR = 0 THEN DO1 X=JUMP1 JUMP1=JUMP1+1

394  RETURN

395  E41: /* DEC */
396  IF ADDR = 0 THEN DO1 X=DEC1 DEC1=DEC1-1

397  RETURN

398  E42: /* DEX */
399  IF ADDR = 0 THEN DO1 X=DEX1 DEX1=DEX1-1

400  RETURN

401  E43: /* JSR */
402  IF ADDR = 0 THEN DO1 X=JSR1 JSR1=JSR1+1

403  RETURN
MAIN: PROC OPTIONS(MAIN)!

E45: /* LSP */

IF ADDR=0 THEN DO:
  IF ACC_CODE THEN REL=ACCA:
  ELSE REL=ACCB:
ENDIF:
ELSE REL=M(ADDR+2):
RESULT:
CALL STR(85*15+1):
SUBSTR(85*9+7):
SUBSTR(85*9+6):
REL=DEC(85):
RETURN:
END:
ELSE M(ADDR+2)=REL:
RETURN:

E46: /* NEG */

IF ADDR=0 THEN DO:
  IF ACC_CODE THEN DO:
    X=ACCA:
    Y=ACCB:
  ELSE DO:
    X=ACCB:
    Y=ACCA:
  ENDIF:
ENDIF:
ELSE M(ADDR+2)=REL:
RETURN:

E47: /* NOP */

RETURN:

E48: /* OR */

IF ACC_CODE THEN DO:
  ACC.CODE
  BIT(M(ADDR+2)):
ELSE DO:
  BIT(M(ADDR+2)):
ENDIF:
CALL CODE(X+Y+1+4+2+1)+H+N+Z+V+C+ACC):
RETURN:

E49: /* PSH */

IF ACC_CODE THEN M(SP,2)=ACCA:
ELSE M(SP,2)=ACCB:
RETURN:

E50: /* POP */

SP=SP+1:
IF ACC_CODE THEN ACC=SP:
ELSE ACC=SP:
RETURN:

E51: /* ROL */

IF ADDR=0 THEN DO:
  IF ACC_CODE THEN REL=ACCA:
  ELSE REL=ACCB:
ENDIF:
ELSE REL=M(ADDR+2):
RESULT:
CALL STR(85*9+7):
SUBSTR(85*9+6):
REL=DEC(85):
RETURN:
END:
ELSE M(ADDR+2)=REL:
RETURN:

E52: /* ROR */

IF ADDR=0 THEN DO:
  IF ACC_CODE THEN REL=ACCA:
  ELSE REL=ACCB:
ENDIF:
ELSE REL=M(ADDR+2):
RETURN:
MAIN: PROC OPTIONS MAIN;
  AS=RTC(REL);
  SUBSTR(AS\&7)=SUBSTR(AS\&7);
  SUBSTR(AS\&1)=C;
  REL=CC(AS);
  XY=REL;
  CALL CODE(X+Y+X+Y+1+4+2+1+H+N+Z+V+C+ACCX);
  V=\$M(ACCA);
  IF ADDR=0 THEN DO 1 IF ACC=CODE THEN ACC=REL;
  ELSE ACC=REL;
  END;
  RETURN;
ES4: /* STS */
X=SP=SP+2;
PC=MT(SP+2);
RETURN;
ES5: /* SBA */
ACCA=ACCA-ACC;
RETURN;
ES6: /* SEC */
IF ACC CODE THEN ACCA=ACCA-ADDR-C;
ELSE ACC=ACC-ADDR-C;
RETURN;
ES7: /* SEC */
RETURN;
ES8: /* SEV */
V=MT(8);
RETURN;
ES9: /* STA */
IF ACC CODE THEN W(ADDR+2)=ACCA;
XY=(ACCA);
ELSE
END;
CALL CODE(X+Y+X+Y+1+4+2+1+H+N+Z+V+C+ACCX);
RETURN;
ES10: /* STS */
X=SP+1;
RETURN;
ES11: /* STS */
CALL CODE(X+Y+X+Y+1+4+2+1+H+N+Z+V+C+ACCX);
RETURN;
ES12: /* STX */
X=MT(ADDR+2);
CALL CODE(X+Y+X+Y+1+4+2+1+H+N+Z+V+C+ACCX);
RETURN;
ES13: /* STS */
IF ACC CODE THEN ACCA=ACCA-ADDR-C;
ELSE
END;
CALL CODE(X+Y+X+Y+1+4+2+1+H+N+Z+V+C+ACCX);
RETURN;
ES14: /* STI */
RETURN;
ES15: /* STS */
RETURN;
ES16: /* STX */
RETURN;
ES17: /* STS */
RETURN;
ES18: /* STX */
RETURN;
ES19: /* STX */
RETURN;
ES20: /* STX */
RETURN;
ES21: /* STX */
RETURN;
ES22: /* STX */
RETURN;
ES23: /* STX */
RETURN;
ES24: /* STX */
RETURN;
ES25: /* STX */
RETURN;
ES26: /* STX */
RETURN;
ES27: /* STX */
RETURN;
ES28: /* STX */
RETURN;
ES29: /* STX */
RETURN;
ES30: /* STX */
RETURN;
ES31: /* STX */
RETURN;
ES32: /* TAG */
ACCA=ACCA;
RETURN;
ES33: /* TAG */
RETURN;
ES34: /* TAG */
RETURN;
ES35: /* TAG */
RETURN;
ES36: /* TAG */
RETURN;
ES37: /* TAG */
RETURN;
ES38: /* TAG */
RETURN;
ES39: /* TAG */
RETURN;
ES40: /* TAG */
RETURN;
ES41: /* TAG */
RETURN;
ES42: /* TAG */
RETURN;
ES43: /* TAG */
RETURN;
ES44: /* TAG */
RETURN;
ES45: /* TAG */
RETURN;
ES46: /* TAG */
RETURN;
ES47: /* TAG */
RETURN;
MAIN: PROC OPTIONS(MAIN)

649 E48: /* TP6 */
650 CALL MAIN
651 SUBST(85+0)=CCI
652 ACC=DEC(85)+1
653 RETURN

654 E49: /* TST */
655 IF ADDR=0 THEN DOJ
656 IF ACC_CODE THEN XY=ACCA1
657 ELSE XY=ACCB1
658 END
659 ELSE XY=ADDR*2+1
660 CALL CODE(X,Y,X+1,F+2+Z+H+N+Z+V+C+ACCX)
661 RETURN

662 E70: /* TIX */
663 RETURN

664 E71: /* TIX */
665 SP=11
666 RETURN

667 E99: /* END EVAL1 */

ADDRESS: PROC

670 DCL ONE FIXED ATN: XX BIT(15)
671 DCL LAL1(17): LABEL1
672 LAL1(17): NAME1
673 LAL1(1): NAME1
674 LAL1(1): NAME1
675 LAL1(1): NAME1
676 LAL1(1): NAME1
677 LAL1(1): NAME1
678 LAL1(1): NAME1
679 LAL1(1): NAME1
680 IF(MOT(MM).ADDR.ARCC=(1A)) THEN ACC_CODE=11111
681 ELSE IF(MOT(MM).ADDR.ARCC=111) THEN ACC_CODE=0111;
682 GO TO LBL1(MOT(MM).ADDR.MODE)
683 RETURN

684 A1: /* INHERENT */

685 RETURNS

686 A2: /* RELATIVE */

687 IF BYTE2>=240 THEN BYTE2=(255-BYTE2)+1 (-1)
688 ADDR=BYTE2+1
689 RETURN

690 A3: /* IMMEDIATE */

691 IF MOT(MM).MOT-NO=31 |
692 MOT(MM).MOT-NO=43 |
693 THEN EA=SPC+11 |
694 ADDR=BYTE2+1
695 RETURN

696 A4: /* ACCUMULATOR */

697 EA=SPC+1
698 ADDR=11
699 RETURN

700 A5: /* INDEXED */

701 ADDR=BYTE2+1X
702 ADDR=BYTE2+1X
703 ADDR=ADDR+11
704 RETURN

705 A6: /* EXTENDED */

706 ADDR=ADDR+11
707 ADDR=ADDR+11
708 ADDR=ADDR+11
709 ADDR=ADDR+11
710 END ADDRESS
MAIN: PROC OPTIONS(MAIN):

VP2: GO TO CCS;

CC2: GO TO CALL(C4);

CC3: GO TO CCS;

CC4: GO TO CCS;

CR2: GO TO CCR;

CC6: RETURN;

CC1: GO TO CCS;

CR1: GO TO CCS;

CC5: GO TO CCR;

END CODE;

CONVERT: PROC(A1) RETURNS(CHAR(16)VAR);

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

CREATE1: PROC;

CREATE2: PROC;

STAT: PUT SKIP;
MAIN: PROC OPTIONS(MAIN); 

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

(COL(5) A SKIP COL(15) A SKIP COL(15) A1); 

PUT SKIP; 

MINT: CALL CREATE1; 
CALL CREATE2; 

MSTART: 

PUT SKIP; 

PUT EDIT('(?)(COL(1) A 1)'); 

PUT SKIP; 

READ FILE(SYSPIN) INTO BUFFER1; 

VERIFY BUFFER1; 

BUFFER=SUBSTR BUFFER1(20); 

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

INDEX BUFFER1; 

BUFFER=SUBSTR BUFFER1(20); 

INDEX COMMAND Y0; 

IF Z0=0 THEN DO1; 

PUT SKIP; 

PUT EDIT('### ERROR 301###'); 

IF 'UNDEFINED SIMULATOR COMMAND' 
(COL(1) A COL(17) A COL(2) SKIP COL(1) A1) 
PUT SKIP; 

GO TO MSTART; 

END; 

Z0=(Z0+1)/21; 

LAL3(1)=M1; 

LAL3(2)=M2; 

LAL3(3)=M3; 

LAL3(4)=M4; 

LAL3(5)=M5; 

LAL3(6)=M6; 

LAL3(7)=M7; 

GO TO LAL3(Z0); 

M1: 

GO TO LAL3(Z1); 

Z0=VERIFY BUFFER1; 

BUFFER=SUBSTR BUFFER1(20); 

GET STRING BUFFER EDIT(CHAREP)(A(4)); 

INDEX BUFFER1; 

BUFFER=SUBSTR BUFFER1(20); 

GET STRING BUFFER LIST(Z2); 

CON(CHAREP); 

BUFFER=SUBSTR BUFFER1(20); 

GET STRING BUFFER LIST(Z2); 

CON(CHAREP); 

Z1=INDEX REGISTER Y1; 

IF Z1>4 THEN GO TO M3; 

IF Z0=0 THEN Z0=Z2/12; 

GET STRING BUFFER LIST(Z2); 

CON CHAREP; 

IF PC>5000 THEN PC=CON CHAREP; 

IF IX>5000 THEN IX=CON CHAREP; 

PUT EDIT('M3(Z2) DO Z3=Z1 TO Z2)((Z0)(12 F(5),SKIP)); 

GO TO MSTART; 

M2: 

/* DISPLAY MEMORY */ 

VERIFY BUFFER1; 

BUFFER=SUBSTR BUFFER1(20); 

GET STRING BUFFER EDIT(CHAREP)(A(4)); 

INDEX BUFFER1; 

BUFFER=SUBSTR BUFFER1(20); 

GET STRING BUFFER LIST(Z2); 

INDEX REGISTER Y1; 

IF Z0=0 THEN Z0=Z2/12; 

GET STRING BUFFER LIST(Z2); 

IF X=CON CHAREP; 

IF PC>5000 THEN PC=CON CHAREP; 

IF IX>5000 THEN IX=CON CHAREP; 

PUT EDIT('M4(Z3) DO Z3=Z1 TO Z2)((Z0)(12 F(5),SKIP)); 

GO TO MSTART; 

M3: 

/* SET REGISTER */ 

VERIFY BUFFER1; 

BUFFER=SUBSTR BUFFER1(20); 

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

INDEX BUFFER1; 

BUFFER=SUBSTR BUFFER1(20); 

VERIFY BUFFER1; 

BUFFER=SUBSTR BUFFER1(20); 

INDEX REGISTER Y1; 

IF Z0=0 THEN Z0=M3-21; 

IF Z0=0 THEN Z0=M3-31; 

M3_1: 

GET STRING BUFFER EDIT(CHAREP)(A(4)); 

IF Z0=0 THEN PC=CON CHAREP; 

IF Z0=0 THEN IX=CON CHAREP; 

IF Z0=0 THEN IX=CON CHAREP; 

IF PC>5000 THEN PC=5000; 

IF IX>5000 THEN IX=5000; 

PUT SKIP;
**MAIN: PROC OPTIONS(MAIN)**

966  PUT EDIT(**** ERROR 306 'Y');
967  THE OPERAND IN THE SIMULATOR COMMAND '
968  CAUSED A REGISTER OVERFLOW')
969  (COL(1), COL(15), COL(17), 1, 1, SKIP, COL(1), 1, SKIP, COL(1), 1)
970  END
971  GO TO MSTRT;
972  M3_2: GET STRING(BUFFER) LIST(Z1);
973  IF 70=5 THEN ACCA=211
974  IF 70=7 THEN ACCB=211
975  GO TO MSTRT;
976  M3_3: GET STRING(BUFFER) EDIT(GC)(A16);  
977  =SUBSTR(GC,1,11) =SUBSTR(GC,2,11) =SUBSTR(GC,3,11)  
978  =SUBSTR(GC,4,11) =SUBSTR(GC,5,11) =SUBSTR(GC,6,11)
979  GO TO MSTRT;
980  M4: /* RUN */
981  GET STRING(BUFFER) LIST(Z1);
982  ON Z2=1 TO Z11
983  CALL SIM;
984  IF ERR THEN GO TO MSTRT;
985  GO TO MSTRT;
986  M5: /* SET HEADER COUNT */
987  GET STRING(BUFFER) LIST(Z1);
988  HC=Z1;
989  GO TO MSTRT;
990  M6: /* DISPLAY REGISTERS */
991  CALL PRT;
992  GO TO MSTRT;
993  M7: /* EXIT */
994  END MAIN;
995
996
997
998
999
1000


