1977

A multiple microprocessor simulator

Donald Eugene Chapman
Iowa State University

Recommended Citation

Chapman, Donald Eugene, "A multiple microprocessor simulator " (1977). *Retrospective Theses and Dissertations*. 5861.
https://lib.dr.iastate.edu/rtd/5861

University Microfilms International
300 North Zeeb Road
Ann Arbor, Michigan 48106 USA
St. John's Road, Tyler's Green
High Wycombe, Bucks, England HP10 8HR

77-25,975

CHAPMAN, Donald Eugene, 1941-
A MULTIPLE MICROPROCESSOR
SIMULATOR.

Iowa State University, Ph.D., 1977
Engineering, electronics and electrical

Xerox University Microfilms, Ann Arbor, Michigan 48106

A multiple microprocessor simulator

by

Donald Eugene Chapman

A Dissertation Submitted to the
Graduate Faculty in Partial Fulfillment of
The Requirements for the Degree of
DOCTOR OF PHILOSOPHY

Major: Electrical Engineering

Approved:

Signature was redacted for privacy.

In Charge of Major Work

Signature was redacted for privacy.

For ~~the Major~~ Department

Signature was redacted for privacy.

For the Graduate College

Iowa State University
Ames, Iowa
1977

TABLE OF CONTENTS

I. INTRODUCTION

In the development of digital systems technology the analysis emphasis has been shifting. In the small digital systems of several decades ago the analysis emphasis was on study of circuits of electronic components. As digital systems increased in size, it became impractical to study them component by component and gate level analysis became necessary. Subsequently, register transfer level analysis and then system level analysis were required to study the larger systems. The increasing size of digital systems created a need for computer design aid programs at all levels to carry out the complex problems of studying the relative timing of signals within the networks of logic devices.

With the advent of large scale integration hardware technology has come the trend towards replacing large blocks of discrete logic devices with programmable microprocessors. The use of microprocessors can decrease costs and increase versatility of many logic systems. However, the use of microprocessors does complicate systems design in that the response times of the logic depend upon both the hardware speed and the software programming efficiency.

At the time this work was started, in early 1975, only a few computer-aided design tools were available for use in the designing of multi-microprocessor systems. The manufacturers of some of the microprocessors had made available a few compilers, assemblers, cross-assemblers, and simulators for their particular products. As these programs were mostly proprietary, the information available about them was primarily in the form of users manuals or brief descriptions in the microprocessors' data books. The available software was intended for use in designing single processor systems or subsystems.

From its inception this work was aimed toward the problem of studying and designing systems constructed from several linked microprocessors rather than just single processors. With the declining prices and increased availability of microprocessor hardware has come a proliferation of systems constructed of several microprocessors linked by hardware and software to form complex multiple microprocessor systems. With such a rich variety of choices available, the systems designer could use a simulation program to study alternative designs.

The desirable simulator must allow simulation of systems in which a variety of linking techniques are used to couple the microprocessors. It should allow the simulation of systems utilizing processor hardware from several manufacturers in a unified system so that the designer can use the individual strengths of each of the available processors for the design of systems that have overall effectiveness. The original contributions represented by this dissertation involve generating a generalized multiple microprocessor simulation package with appropriate new constructs and then demonstrating that such a system can be implemented in a viable way.

II. REVIEW OF LITERATURE

Computer hardware and software simulation has been described by Breuer (1) as predominantly being performed at five basic levels:

1. System level
2. Register transfer level
3. Logic level
4. Gate level
5. Circuit level

As noted, the objective of this work is to help bridge the gap between system level and register transfer level simulation for multi-microprocessor systems.

System level simulation has been studied by many researchers. An example of systems modeling and an extensive bibliography is provided by MacDougall (2). MacDougall studies use of such simulation languages as GPSS and SIMSCRIPT which are oriented towards job, task and event level of simulation. Such simulations are used for the study of throughput, turnaround, and other performance parameters of an overall computing system.

An interesting systems level simulation study is provided by Katz (3) who models a dual direct coupled processor system of IBM 709X and 704X processors. Katz also simulates at the job, task and event levels of overall performance. Katz uses typical computer installation usage statistics to generate statistically representative job sets to be presented to the simulation program. To model the overhead caused by conflicting demands for device service in the dual processor system, Katz used data obtained by tracing the operation of the DCS direct coupled operating system.

Cheng (4) has done some systems level simulation in which he develops simulation models from data obtained from an event tracing of the system's performance. Cheng obtained statistics for event analysis from an interrupt driven computer system where any interrupt called a trace routine which recorded actual operating statistics on the occurrences of events within the computer system.

Simulation of computers or data processors at the register transfer level has been studied for single processor systems quite extensively. Some early work in register transfer simulation of computers was done by Chu (5) through the use of his CDL (computer design language) and its associated software. Chu's simulator accepts user defined descriptions of register data transfers and associated micro-instruction descriptions. His simulator is designed to accept data from the CDL translator, which translates the hardware descriptions into polish strings, tables and arrays. The simulator program itself is a five part program consisting of a Loader, Output Routine, Switch Routine, Simulative Routine, and a Reset Routine. The CDL language and simulator are intended to allow the user to study the functioning of a mono-processor.

The software simulation of single-instruction-stream-multiple-data-stream computer was undertaken by Shore (6). In his work he simulates the AP (associative processor) of the Advanced Avionics Digital Computer (AADC). Shore's simulator performs a combination of assembly and simulation in the sense that it accepts mnemonic instruction codes and data and then provides a detailed trace of the execution of the program by the AP. His simulator makes no distinction between relative execution speeds of different instructions but rather treats each instruction as though it requires one unit of time.

Most of the vendors of microprocessor hardware provide design support software aids for single processor systems using their particular products. The details of the structures of these design aid programs are somewhat obscured by the fact that the programs are mostly proprietary. A review of the currently available manufacturer vended software design tools is given by Watson (7).

More recently Mueller and Johnson (8,9) have been studying generalized simulators designed to allow the simulation of any microprocessor. Their SIM/GEN program provides a "skeleton routine" which is a simulation program that is complete except for the constants defining the target machine. The user provides descriptions of the architectural dimensions of the target machine and also provides vertical microprograms describing each executable instruction of the target machine. The vertical microprograms are written in Mueller and Johnson's

IDL (instruction definition language) which provides 27 primitive operators, ten basic operand designators, six relational operators for conditional statements, and the facility of defining micro instruction subroutines. Their objective was to allow for generation of a simulator for a microprocessor with one-to-five-man-days of effort as opposed to a two-to-six-man-month effort.

The register transfer level simulators described above all are intended for use in simulating mono-processor systems: systems with only one CPU. It is the intention of this work to provide the systems designer with a software design aid which will allow him to study multiple linked microprocessor systems. The desired simulator should provide adequate trace information to allow for subsequent systems level modeling such as is described by Cheng (4). The target machine descriptions are to be provided as vertical microprogramming in the PL/1 language. PL/1 was chosen for two reasons: it provides a rich variety of constructs and also is a widely know language for which compilers and translators are readily available.

III. DESIGN OF THE SIMULATOR

A. Desired Features

To allow for machine independence, portability, and ease of development the simulator is written as one large PL/1 program. The program will act upon user supplied data in a batch processing mode. The design of the program will allow for the simulation of any mix of eight bit sliced microprocessors; however, to maintain reasonable memory space requirements, the compiled version of the research simulator will be limited to the simulation of systems constructed from the Intel 8008, the Intel 8080, the Motorola 6800, and the Fairchild F8 microprocessors.

The simulator must be able to properly carry out the transformations on register, memory, flag bit, and timing parameter values to correctly provide an observable description of the results of operations internal to the microprocessors being simulated.

It would be highly desirable if the simulator could simulate true parallelism among several microprocessors in a system. There are some difficulties inherent in the proposition of simulation true parallelism. Perhaps the largest single impediment stems from the fact that the differently designed microprocessors have instruction cycles of varying length, and indeed, even a single microprocessor has instructions within its instruction set of varying lengths. To truely simulate parallelism in a multiple processor environment would require programming the simulation to reflect each internal microcommand of each instruction of each processor. Since the designs of the microprocessors are proprietary, detailed information regarding the internal microcommands was not available when this simulator was being written. The type of information that was available was the macro results of each instruction execution. Thus, only something less than simulation of true parallelism was possible. To subsitute for true parallelism an alternate mode of simulation was selected. The simulator program is designed to allow interleaved sequential processing where accurate timing information in hardware clock cycles is maintained. Through studying the clock timing values in the trace printout, the user can reconstruct the relative timing of instructions within the multiple processor environment.

It is desirable that the simulator provide some means of simulating the interrupt nature of the microprocessors. This simulator does so through the use of the INTERRUPT option which forces the execution of the stacking operations and restart characteristics of the microprocessors.

There are two predominant modes possible for linking multiple microprocessors. One mode is through use of shared memory or direct memory access. The other mode is through use of output port to input port programmed transfers, perhaps through a small set of buffering registers. This simulator makes possible the simulation of systems utilizing one or both of these linking techniques. Some ingenuity on the part of the user is required through his use of the selected options available to him. Once again the user will need to study the clock timing parameter of the trace printout to assure that buffer overflows are avoided and to insure that congruence of information availability is maintained.

The simulator program itself should be written in understandably sized blocks where possible. This simulator is written in seven major functional blocks and fifteen utility subroutines. One design criteria that was used, was to try and choose methods of coding processes that were related to the probable frequency of execution of the coding. An attempt was made to write frequently used code for fast execution and seldom used code for minimum program storage area. The successfulness of this attempt is somewhat obscured by the ways in which the compiler might interpret the subject PL/1 code.

Since potential users may be unfamiliar with the PL/1 language, the simulator should be structured so that all errors are detected by the simulator itself. Preferably, the error messages should come from the simulator program rather than from the PL/1 language facility or the operating system of the host computer. For this reason the simulator program contains code to maintain checks upon subscript ranges and similar details internal to the simulator program's structure.

B. Major Blocks of Programming

1. Initialization

The first block of the simulator program serves the other blocks of the program in that it declares and initializes the variables and arrays used throughout the program.

The arrays of timing, program counter increment, and alphanumeric mnemonics were initialized in the declaration statements in the hope that the compiler would therefore initialize the large arrays at compile and load time rather than generate run time code to do the initialization. The fate of this hope is buried somewhere in the particular way in which the compiler was written and is, of course, compiler dependent. To minimize storage requirements the variables were declared with the precision required by their possible range of values. In some cases the PL/1 compiler ignored the declared precision and specified a larger number of bits than were really required for the variables. Where possible, the variable labels were chosen to be abbreviated forms of the names descriptive of the function the variable performs within the simulator code.

2. Memory and systems definition

This block of program allows for the initial loading of programs to the simulated memories, allows for the listing of the microprocessor memories, and allows for the storage of memory values and register values to alternate protected areas in the simulator's memory area.

The memory loading subroutine accepts lists of hexidecimal values which it stores into the simulated memory locations. The choice of hexidemical notation is somewhat restrictive; however, since it is more compact than binary or octal notations, it was chosen as the uniform notation for all input and output of data for machine and memory simulation. The only exception to the use of hexidecimal is in the case of the clock timing parameter of the trace which is in decimal notation for convenience. Since many systems in actual usage utilize a mix of random access and read only memory hardware, the memory definition option allows for one continuous area of memory to be defined as read only storage. The simulator makes a continuous check to detect any attempt to write to the read only portion of memory. Since two of the simulated microprocessors utilize random access memory to store pushdown stacks, the software fence sub-option was included to allow for continuous checking for overflow of the stack beyond its assigned area of memory.

The memory listing subroutine was provided to allow the user to observe the contents of a portion of simulated memory without taking a full dump of the memory's contents. Since a full dump of memory requires many pages of printout, the user should be encouraged to use the memory listing option to observe only the segment of memory that is of interest.

The SWAP sub-block of program allows for the storage of current processor values, current memory values, and current input stream values into a protected area of the simulator program's memory area. Since the simulator program only simulates the execution of one microprocessor at a time, the user needs to be able to preserve the status of other microprocessors in the system during the time in which they are not being actively simulated. This protected area of memory is not accessible to the microprocessor being actively simulated. Subsequent swaps can trade the inactive parameters and active parameters between the accessible and inaccessible parts of the simulator's memory area.

3. I/O device and interface definition

This block of program allows for the definition of input devices and data, output devices, and first-in-first-out buffers.

The PORT option allows the user to define input and output devices. The input devices are defined by the device identifier value followed by a list of input values for that device. Output devices are defined, when necessary, by a list of device identifier values. To be consistent with other data input formats for the simulator, the device and data values must be in hexidecimal notation.

The FIFO option allows the user to define four device identifier values to act as the inputs and outputs to two first-in-first-out buffers. These buffers are maintained by the simulator program. They are intended particularly for use in the simulation of output to input programmed linking of two microprocessors within a system. They may also be used to define first-in-first-out hardware parts connected to the ports of a single microprocessor.

4. Control of simulation

Once the microprocessor's program has been loaded to memory and the input data has been predefined, the user can institute the start of the simulation process. The simulator provides two mechanisms for starting or resuming the simulated execution of instructions. These mechanisms are provided by the PROCESSOR and the INTERRPUT options.

The PROCESSOR option card is used to define which particular model of microprocessor is to be simulated. Also the user may specify sub-options for: a starting address, a breakpoint address, a reset of the clock counter, an upper limit for the clock counter, trace printout limitations, and a post execution dump of memory. Data cards may follow the PROCESSOR card to provide the values for parameters specified as sub-options.

When the simulation process has halted, the user may cause the simulation to be restarted through the use of another PROCESSOR call or through the use of the INTERRUPT option.

The INTERRUPT option restarts the simulation in the manner that the microprocessor would respond to a hardware interrupt. Stack pushes, input of interrupt vectors and all similar activity is initiated by the interrupt sub-block of simulator programming. The INTERRUPT option allows the user to simulate the interrupt nature of the microprocessors' structures. It also allows most of the control sub-options available to the PROCESSOR option.

Both of these options initiate the perpetual cycle of the remaining blocks of programming. The loop in programming includes: checking for halt conditions, fetching and decoding the new instructions, updating the timer and program counters, executing the instruction, and listing the trace of the instruction executed.

5. Check, fetch, update, and decode

This block of the simulator program first checks to see whether conditions have arisen during simulation that require termination of simulation. If termination is indicated, the simulator returns to scan the user's cards for more control options. If simulation is to continue, the next instruction's operation code is fetched from the simulated memory. Through the use of table lookups, the number of clock cycles required to execute the new instruction and the number of memory bytes of instruction are found and are added to the clock counter and the program counter to update these parameters.

The nature of the decoding process was largely dictated by the design of the block of program which simulates the execution of the instructions. Since invalid operations codes, as well as valid ones must be dealt with, the eight bit operations codes for the four different models of microprocessors require the simulation of 1024 instruction possibilities. It was felt that the simulation of each instruction execution could not be done through use of subroutine calls because the compiler would have to generate code to link each call statement. Since 1024 instruction cases needed to be simulated, the subroutine linking code would have made the simulator program unnecessarily large. The GO TO statement using the label array feature of the PL/1 language was chosen as a more compact way to perform the decoding function. This, along with the multiple label feature, allows for more compact coding and in some cases allows for sharing of code among similar simulated instructions.

6. Execution array

This block of programming is tailored to simulate the execution of the instructions for the chosen microprocessors. If an alternate choice of microprocessors is desired, then this segment of the subject code could be replaced in part to simulate other microprocessors. The structure of the execution code is that of an array of sub programs. As was described earlier, the GO TO LABEL ARRAY feature of the PL/1 language was chosen as the decoding mechanism. This creates a 1024 way branch in control to the sub-blocks of the array of the groups of instructions that simulate the execution of each of the instructions. These sub-blocks of program, through use of the fifteen common utility subroutines, carry out the operations on the register values necessary to simulate the execution of the current instruction. These sub-blocks of program also operate on the flag bits and pass information to the trace routine to generate correct post execution values in the trace listing. The manuals for the microprocessors (10,11,12, 13) were consulted in detail and PL/1 statements were written to simulate each individual instruction.

Where feasible, the multiple label facility of the PL/1 language was used to allow similar instructions to utilize in common some of the simulation program statements in order to help minimize the size of the simulator program. Perhaps the best example of this common usage occurred in the handling of operations codes that are not valid for executable instructions. Among the 1024 possible operations codes, 81 are for undefined operations. These 81 cases utilize a single sub-block of programming to generate an error message informing the user that the processor has encountered an invalid operations code. Also, because of the similarity between the Intel 8008 and the Intel 8080 microprocessors, there is an overlap of better than eighty percent in their shared usage of execution code.

Where possible, the multiple statement per card feature of PL/1 was used to help maintain integrity of the subject deck. If the statements of one sub-block of the array of sub programs are on one or a few cards, then the subject deck is less susceptible to disruption if the cards are accidently disordered. When a PL/1 program of 3679 statements was being debugged, this order independence of the deck proved to be valuable.

7. Trace

The purpose of a simulator is to give the designer a view of what is happening in the processor as it attempts to execute his programming. The worth of a simulator is most accurately evaluated from the usefulness of the trace information it provides. The trace must give the user a good view of what is happening in the registers of the processor.

The trace generation routine lists the data in all of the active registers of the processor, along with the clock, flag bits, and a mnemonic operation code for each simulated execution of an instruction. This listing gives the user a clear picture of what has happened during the simulated execution of each instruction. Since hexidecimal notation was chosen for all input and output to the simulated system, the values in the registers can be represented in fairly compact form. The compactness allows for the listing of all of the active registers on one line of trace printout. It was felt that, during use of the simulator for debugging software, the full printout of all registers would save the user the cost of additional runs of the programs. All of the internal parameters are always available for each traced instruction execution.

Since the user tends to develop working software one block at a time, is seemed desirable to give the user the ability to suppress the bulk of trace information during simulated execution of segments of program that have been tested and are known to be satisfactory. To allow the user control over the amount of trace printout it was deemed necessary to include a breakpoint facility. A multiple breakpoint capability would have been desirable but would have required a scan of a list of breakpoints upon the simulated execution of each instruction. This list scanning would have required more memory space for the simulator object code and would have also decreased the speed of simulation. Thus, a single breakpoint capability was chosen. Through the use of the STOP sub-option, the user can halt simulation at any instruction and is then given a choice of the trace detail upon restarting the simulation. The choices available are: listing of only input and output related instructions, listing of both input-output and instructions which cause non-sequential changes in the program counter, or the default of complete trace listings. These choices are allowed through the use of the I/O, BRANCH, or JUMP sub-options. Once the user has requested suppression of some of the trace printout, that request remains valid until the processing stops, perhaps at a breakpoint.

Of course, headings are printed above each column of the trace printout to identify the parameter being listed in the columns. The headings are always printed upon initiation of execution and are also printed for every two pages of trace listing. The accessibility of the headings information makes the trace data more easily understandable.

8. Fifteen utility subroutines

Where feasible, common processes were programmed as subroutines so that the overall memory space requirements for the simulator program could be kept within the space available in the host processor. A total of fifteen subroutines were written.

The FETCH subroutine accesses the simulated memory for one byte of data and checks for memory boundary errors.

The STORE subroutine checks for read-only-memory boundaries and then stores one byte of data to the simulated memory.

The DUMP subroutine is used to list the contents of portions of memory, providing a column of address values for reference.

The HEADINGS subroutine is called to generate the proper column headings during listing of a simulation trace.

The SCRATCH subroutine lists the contents of a simulated Fairchild F8 microprocessor's scratchpad memory.

The HEXIN subroutine reads hexidecimal notation values and converts them to binary integer form for internal host processor usage.

The TOHEX subroutine converts binary integers to alphanumeric hexidecimal form for output listing purposes.

The FIFO subroutine handles the input and output of data to the simulated first-in-first-out files provided by the simulator.

The ADD subroutine simulates the binary addition process and also adjusts the condition flags of the simulated microprocessors.

The LFLAGS subroutine computes the condition flags of the Intel 8008 and 8080 for the post execution conditions after a logical operation.

The TELIN subroutine extracts the consecutive data values from the input data provided by the user and supplies these values upon the execution of an input instruction.

The PUSH subroutine simulates the processes of maintaining a pushdown stack in simulated memory. It checks for boundary overflows whenever a software fence has been defined. It also checks for overflow into the low end of memory for the Intel microprocessors and warns the user if a stack push has invaded the portion of memory normally accessed by executions of RST instructions.

The POP subroutine simulates the process of popping data from a pushdown stack maintained in memory.

The DNK subroutine carries out simulation of decrementing and incrementing the Indirect Scratchpad Address Register when a Fairchild F8 microprocessor is being simulated.

The DECIN subroutine converts alphanumeric decimal input values to binary integer form for use by the simulator.

IV. USERS MANUAL FOR THE SIMULATOR

A. General Format for Input Cards

The simulator program expects to find two kinds of cards in the input deck, options cards and data cards. Options cards are alphanumeric cards that control the operations of the simulator. Data cards provide two types of information to the simulation process. One use of data cards is to provide numerical control values for the options cards immediately following the options cards. The other use of data cards is to provide numerical data for memory and input values for the microprocessor being simulated.

Options cards all start with an asterisk (*) in column one and have an option name followed by optional sub-option names.

Data cards follow their associated options cards in the deck. Data items are predominantly hexidecimal numeric values terminated by commas and several may be on one card. The last data item on a card should be terminated with a semicolon. Comments may be placed on the card following the semicolon. The last data card in a set of data cards should generally terminate with a period to indicate the end of that set of data.

B. Options for Simulation of a Single Microprocessor

There are five major options intended for use in simulating a microprocessor. They are:

MEMORY
LIST
PORT
INTERRUPT
PROCESSOR

The MEMORY option is used to allow an initial program loading of the simulated memory. There are four optional sub-options to the MEMORY option:

SIZE
ROM
FENCE
DUMP

The SIZE sub-option allows the memory size in bytes to be specified as a hexidecimal value. The default size is 256 bytes and the maximum size is 65536 bytes.

The ROM sub-option allows one contiguous section of memory to be treated as read-only-memory. The simulator will flag any attempt to write to a memory location specified to be read-only-memory. The ROM sub-option requires hexidecimal starting and ending addresses for the read only segment of memory.

The FENCE sub-option allows a part of the lower end of memory locations to be protected from stack overflows. As the simulation proceeds, all stack pushes are checked and as the fence is approached, warnings are issued. An attempt by the microprocessor to stack below the fence boundary will cause a halt in the simulation.

The DUMP sub-option causes a post execution listing of the contents of simulated memory. This sub-option should only be used when memory size is small. If memory size is large, the LIST option should be used to get a listing of only the desired portion of memory.

The MEMORY option card should be followed first by data cards providing values to the sub-option selected. These values must appear in the order in which they are listed above. Then the memory load data must be listed in the deck. The data set must contain lists of data values to be loaded into consecutive memory locations. Starting addresses for the lists may be included in the lists but they must be terminated with a semicolon. The data set must be terminated with a card starting with a period.

The LIST option allows for listing of the hexidecimal values in the bytes of simulated memory. The LIST option requires a data card to provide starting and ending addresses for the listing as hexidecimal addresses. As the list is printed, the read only bytes of memory are flagged with asterisks and the fence location is flagged with a verticle bar.

The PORT option allows for the definition of input and output ports on the simulated microprocessor and allows for the definition of the input data streams for input ports. The PORT option must have either or both of the sub-options:

IN
OUT

If both IN and OUT are specified on the PORT card, the input data cards are expected to precede the output definition cards.

The data cards for the IN sub-option can specify the input device with a hexidecimal value terminated by a colon. Subsequent hexidecimal values will be used as the input stream for that specified device. Additional devices may be specified by subsequent values terminated by colons and followed by the appropriate input stream values. The last data item in the input data set should be terminated with a period.

The OUT sub-option requires hexidecimal device identifiers terminated with colons. Several of these may be included per card. The identifier data set must be terminated with a period following the last colon. Comments can follow the period.

The PROCESSOR option specifies which model of microprocessor is to be simulated and starts the simulation. It also specifies breakpoint and time limit conditions. It must be preceded by use of the MEMORY option so that the program for the microprocessor is already in the memory. The PROCESSOR option has the following sub-options:

8008
8080
6800
F8
START
STOP
TIMELIMIT
RESET
BRANCH or JUMP
I/O
SCRATCH
DUMP

One of the 8008, 8080, 6800, or F8 sub-options must be specified so that the program will simulate the proper model of microprocessor.

The START sub-option allows a hexidecimal address value to be specified as the starting address for the microprocessor simulated execution.

The STOP sub-option allows a hexidecimal address value to be specified as a breakpoint. If the simulated microprocessor's program counter becomes equal to this value, the simulation of execution is halted prior to execution of the instruction at this specified address.

The TIMELIMIT sub-option allows a decimal value to be specified as a limiting value on the number of microprocessor clock cycles during which simulated execution will be allowed. When the clock counter becomes greater than or equal to this value, the simulation is halted. If simulation is to be restarted, the time limit may need to be respecified or the clock counter may need to be reset.

The RESET sub-option caused the clock cycle counter to be reset to zero.

The BRANCH or JUMP sub-options cause the trace listing of the simulation to include only non-sequential instructions and input or output instructions.

The I/O sub-option causes the trace listing of the simulation to include only instructions which involve input or output of data for the microprocessor being simulated.

The SCRATCH sub-option is used specifically during Fairchild F8 microprocessor simulation. It causes the contents of the F8's scratchpad memory to be listed after every 32 lines of trace printout and upon memory dump.

The DUMP sub-option should only be used when memory size is small. It causes the complete contents of memory to be listed upon termination of simulation. If memory size is large, the LIST option should be used to get a printout of the contents of only the desired segment of memory.

The INTERRUPT option permits restarting of simulation in the manner of an interrupt. It is intended to be used to restart the simulation process after some anticipated halt, such as a time limit halt. The INTERRUPT option has the following sub-options:

8008
8080
6800
F8
STOP
TIMELIMIT
RESET
BRANCH or JUMP
I/O
SCRATCH
DUMP

The 8008, 8080, 6800, and F8 sub-options allow the optional redefinition of the model of processor being simulated. The other sub-options have the same functions defined for them for use with the PROCESSOR option. Data cards should follow the INTERRUPT card to provide parameters for the sub-options and also to provide the simulated microprocessor with interrupt vectors of data as required.

C. Format of Trace and Dump Listings

The full trace listing is presented with one line of printout for each executed instruction. The first column of the printout gives the decimal value of the clock cycle counter. All of the other values in the trace printout are in hexidecimal notation. The remainder of the line is used to list post execution values for all of the programmer accessible registers, pointers, and flag bits. Following the register values is the alphanumeric mnemonic used by the manufacturer to describe the instruction executed. If input or output operations are performed by the instruction, the post identifier and data value are listed at the far right of the line. All of the columns of the trace listings are labeled with headings for every two pages of trace printout.

The memory dump listings utilize hexidecimal notation for all values. The first column of the line gives the address for the first data value on the line. Subsequent values are listed in the sequence in which they occur in memory. If the processor has been defined to be the F8 and if the SCRATCH sub-option is in force, then the dump will be followed by a listing of the simulated F8's scratchpad memory.

D. Options For the Simulation of Multiple Microprocessor Systems

The simulator can be used to simulate several processors in an interleaved sequential mode. That is to say that simulation of one microprocessor can be allowed to continue for a while and then simulation of another microprocessor can be allowed to continue for a while. By switching back and forth among processors the operation of the system as a whole can be studied. Attention must be paid to the relative timing through study of the clock cycle counter values at the times of processor interactions. To facilitate the process of switching back and forth among processors, the simulator has two options tailored to ease the process.

The SWAP option swaps all of the active registers and flag bits of the current microprocessor to an alternate protected area of the simulator's storage space. In addition, the SWAP option provides the following sub-options:

MEMORY
PORT

The MEMORY sub-option is used in systems where two processors do not share memory space. This sub-option causes the memory values to be swapped with an alternate protected storage space.

The PORT sub-option allows for the microprocessors to have separately defined input devices. This sub-option causes the input data stream definitions to be swapped with an alternate protected area of the simulator's storage space.

The FIFO option allows the user to define two first-in-first-out buffers to simulate output port to input port linking of microprocessors. The FIFO option requires four data values to define the microprocessor ports to be linked by the two first-in-first-out files. The first value is interpreted to be the output port identifier for the processor connected to the input of the file. The second value is interpreted to be the input port identifier for the processor connected to the output of the file. The next two values similarly define the input and output for the second file. Both files must be defined; however, the second file can be supplied with dummy values as parameters if it is not to be used.

V. EXAMPLE PROBLEM DESCRIPTION

Appendix A provides a simulator printout for one run that might occur in the process of using the simulator to investigate a simple dual processor system. The dual processor system to be designed is to be used to improve the reliability of data communications over a data channel. The proposed system uses an Intel 8080 at the transmitter site to encode the data for error correction, and a Motorola 6800 at the receiver site to perform error correction decoding to reconstruct the original data. For investigative purposes the data channel itself is modeled using the FIFO simulator option to simulate output port to input port linking of the processors through the channel.

The error coding technique chosen for this preliminary study involves transmitting each single byte of source data as two bytes of channel data. Although such a scheme decreases the transmission rate by a factor of two within the channel, it does allow for parity bits to be included with the data so that the data is less prone to errors of transmission on a noisy channel. The Intel 8080 first extracts the four least significant bits of the source data byte and constructs an eight bit Hamming coded byte which is output to the transmission channel. Then the most significant four bits are similarly encoded and transmitted. At the receiving end of the channel the Motorola 6800 is programmed to input each byte of channel data, correct the detectable transmission errors, extract the data bits, and then reconstruct the original single byte of data.

The actual simulation of this preliminary systems design is carried out in the following manner. First, the MEMORY option is used to load the program and subroutines for error correction and reassembly to the memory of the simulated 6800. Since the 6800 uses a pushdown stack in memory for subroutine linkage located at the word in address FF and below, the fence sub-option is called upon to continuously check for stack pushes below the work at address 7B. The DUMP sub-option is used to get a map of memory contents after the loading is completed. Since the SIZE sub-option is not invoked, the default memory size of 256 bytes is assumed by the simulator.

The next option card uses the PORT option with its OUT sub-option to define address 101 to be the memory mapped output port. The FIFO option is used to define address 100 of the 6800 to be the memory mapped input port from the simulated channel. The other addresses specified on the FIFO data card are dummy values.

After the 6800's memory has been loaded and its port configurations defined, the SWAP option is used to call for the subsequent definitions of the 8080's configuration. The MEMORY sub-option is used to simulate independent memory hardware for the two processors. Since the processors are at opposite ends of the channel, they do not share memory space. Similarly, the ports of the processors are to be independent so the PORT sub-option is specified. The simulator responds by informing the user that memory space number two is the current memory space.

The MEMORY option is used to load the 8080's memory with the program and subroutines that split the source data byte, perform the Hamming encoding operations, and transmit the two encoded bytes to the simulated channel. The PORT option is used to specify that input port number three is provided with a list of six source data bytes. The FIFO option is used to define output port number five as the connection to the simulated channel.

The above option and data cards have specified the desired dual processor system complete with programming. Thus, the PROCESSOR option is used to call upon simulation of an 8080 microprocessor acting upon the program in the current memory space number two. A trace printout is generated by the simulator showing each byte of source data being operated upon and two bytes of channel data being subsequently generated. The 8080 program was written to operate upon blocks of two source data bytes and then halt. Thus, the simulator stops tracing when the halt instruction is executed. The channel data bytes have been inserted by the simulator into first-in-first-out buffer number one where they will reside until the

6800 subsequently will extract them.

Next, the SWAP option is used to return to memory space number one, where the program for the 6800 resides. The LIST option then calls for the printout of the map of the values in memory space number one. Processing by the 6800 of the data in the simulated channel is initiated by the PROCESSOR option. The RESET sub-option is used to cause the clock cycle counter to count from zero. The START sub-option is invoked with a data value of zero to cause the 6800 to begin processing with its program counter initially pointing to the first byte in memory.

The 6800 execution is traced by the simulator showing the channel bytes being extracted from the fifo buffer and being error corrected, and the source data byte being reconstructed and then output. This processing continues until the simulator discovers that the channel data in the fifo buffer has been exhausted; whereupon, the simulator terminates the simulation and searches for option cards. Since no more option cards are found, the simulation run is completed.

The printout of the simulation run now can be analyzed to study the performance of the preliminary design of the dual processor error correcting channel system. By observing the clock cycle counter in the 8080 simulation it can be seen that the encoding process has taken 243 clock cycles to output the first channel byte and an additional 206 clock cycles to output the second channel byte. The total time between input of the first and second data bytes was 481 clock cycles. By observing the clock cycle counter in the 6800 simulation it can be seen that the error correction process for the first channel byte takes 158 clock cycles. The second channel byte is input, error corrected, and the source data is reconstructed and output in an additional 166 clock cycles. The total error correction and reconstruction has required 331 cycles of the 6800's clock. Thus, it is apparent that the 6800 is processing the data in fewer clock cycles than are required by the 8080.

At this point the simulation has made the designer aware of the mismatch in processing rates of the processors in his proposed system. He has many design options available for overcoming this mismatch. One possiblity that could be considered would be the operation of the microprocessors from independent hardware clocks with clock rates adjusted to equalize the real time processing rates. Another possibility is to add some null operations to the software programs of the 6800 to slow down its processing rate to equal that of the 8080. A third possibility would be to design an interrupt hardware and software structure for the 6800 to cause it to process channel data only upon demand from the channel. This simulation has shown which end of the system should be interrupt driven. The designer also might consider selection of a different, perhaps faster, choice of hardware or software to replace the transmission end of his system.

The designer can now choose from his alternatives, redesign his system, and use the simulator to study the implications of his new design. The simulator provided by this work allows the designer to try a wide variety of hardware and software configurations as he refines his overall systems design.

VI. CONCLUSIONS AND COMMENTS

A. Summary

In this work an original, generalized multiple microprocessor simulation package was developed with appropriate new constructs. The options available to the user allow a greater degree of systems simulation flexibility than found in previously available simulators. The simulator allows the simulation of systems in which several similar or dissimilar microprocessors are operating concurrently.

The simulation includes the following new constructs. The PROCESSOR option allows the user to define a system constructed from dissimilar microprocessors. The FIFO option allows for buffering of data passed between processors within the system. The SWAP option allows for simulation of systems in which the microprocessors have memory spaces which are disjoint from one another.

The information in Appendix A demonstrates viable effective operation of the simulator as applied to the specific design example described in section V of this paper.

B. General Performance

The simulator program has proven to be a useful design tool. It provides the multiple microprocessor systems designer with the means of exercising proposed systems structures and observing their performance.

The current version of the simulator was written as a 3679 statement PL/1 program. The object code generated by the compiler is currently on tape. This object code executes in a 512 kilobyte region of main memory in the host processor. The relatively large size of the program results in a minimum run cost of about $3.70 simply to load the object code from the tape to the main memory. Once the simulator is loaded the simulation proceeds at a cost of less than $.50 for each 100 instruction executions that are traced. A typical run cost is in the range of $8.00 to $15.00. This cost is perhaps too great to allow for use of the simulator in a tutorial environment but is reasonable in a product development environment.

Since the simulator program was written in managable blocks and since PL/1 was chosen as the source language, modification and expansion of the simulator are relatively easy. The current version of the simulator was written as an expanding program. The compiled version is capable of simulating four relatively common microprocessors. As the simulator program was written, the microprocessor microprogramming was added in a sequential manner. First, the microprogramming for the 8008 and 8080 was written and included in the source deck. Subsequently, the 6800 and the F8 modeling capabilities were added. As a measure of expandability, it is of interest to note that the addition of the F8 capability took only about 23 man hours of programming and debugging effort. This is roughly the same amount of effort reported by Mueller and Johnson (8,9) for use of their SIM/GEN simulator and, unlike their approach, no special microprogramming language has to be learned.

C. Features Not Included

During the development of the simulator, many features were considered for inclusion. Some of the proposed features were eliminated for feasibility reasons.

The trace printout refers to addresses in memory through use of hexidecimal values. The trace printout would be more readable if user defined mnemonics were substituted for these hexidecimal values. This feature could be programmed but was left out for two reasons. First, available cross assemblers generally do not pass mnemonic labels with their object code, so cross assembled software could not make advantageous use of a mnemonic label feature. Second, such a label facility would require relatively extensive tabulation of mnemonic labels and would require additional programming to link the labels to occurrences of the usage of their associated locations in the simulated memory. Since the simulator program already requires a large region of the host processor's main memory, this feature was considered to be unfeasible.

The simulator requires the use of one PROCESSOR or INTERRUPT control option for each time that simulation is resumed. This is a very simple control language structure. A wide variety of constructs could have been included. A conditional "IF" type construct could have been included. It would seem convenient to be able to condition the starting or restarting simulation upon register or memory values within the simulated system. Such a capability would require a fairly extensive syntax interpretation capability from the simulator. Since host processor memory space is already a relatively scarce commodity, this feature was not included.

The current version of the simulator allows for one protected area of storage for an inactive processor within the simulated system. This capability is exercised through use of the SWAP option. The ability to preserve the status of only one inactive processor is somewhat restrictive. A larger number of protected regions and a flexible way for accessing them would be a useful facility. However, if each protected space is to provide for storage of the total memory addressable by one microprocessor, each space would require at least 64 kilobytes of memory. Again, the memory space available within the host processor is a limiting factor.

D. Areas for Further Study

At present the simulator is oriented toward loading the simulated microprocessors' memories from data values taken from the card deck. It would be convenient if such data could be passed directly from data files generated by cross assemblers without the use of cards. A study of the formats of various cross assemblers' generated software should reveal desirable alternatives to the simulator to make it more compatible with other software design aids.

Since the available cross assemblers come from different vendors, it is probable that their output formats vary considerably. Mating the simulator to the diversity of cross assemblers should be a challenging enterprise.

The current version of the simulator uses worst case allocation for the simulated memories. As a result, the program requires a region of host processor main memory that is quite large. It seems likely that most uses of the simulator would not require a full 64 kilobyte memory for each simulated microprocessor within the scope of a simulation run. A study of usage statistics perhaps would indicate the desirability of simulating the memory array using the list processing and dynamic allocation capabilities of the PL/1 language. If, in actual usage, the microprocessors' memories are sparsely populated with data, a considerable simulation cost savings could be achieved through rewriting the FETCH and STORE subroutines to access elements in an expandable list structure.

BIBLIOGRAPHY

1. Breuer, M. A., "Recent developments in the Automated Design and Analysis of Digital Systems." Proceedings of the IEEE 60, No. 1 (1972): 12-27.

2. MacDougall, M. H., "Computer Systems Simulation: An Introduction." Computing Surveys 2, No. 3 (1970): 191-210.

3. Katz, J. H., "Simulation of a Multiprocessor Computer System." Proc. AFIPS 1966 Spring Joint Comp. Conf. 28 (1966): 127-139 (Spartan Books, Washington, D.C.).

4. Cheng, P. S., "Trace-driven System Modeling." IBM Syst. J. 8, No. 4 (1969): 280-289.

5. Chu, Y., "Introduction to Computer Organization." Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1970.

6. Shore, J. E., "Software Simulation of an Associative Processor." Naval Research Laboratory, NRL Report No. 7351, 1971.

7. Watson, I. M., "Comparison of Commercially Available Software Tools for Microprocessor Programming." Proceedings of the IEEE 64, No. 6 (1976): 910-920.

8. Mueller, R. A. and Johnson, G. R., "A Generator for Microprocessor Assemblers and Simulators." Proceedings of the IEEE 64, No. 6 (1976): 921-931.

9. Johnson, G. R. and Mueller, R. A., "Automated Generation of Cross-system Software for Microcomputers." Computer 10, No. 1 (1977): 23-31.

10. MCS-8 Users Manual. Intel Corporation, Santa Clara, California, 1974.

11. 8080 Microcomputer System Manual. Intel Corporation, Santa Clara, California, 1975.

12. M6800 Microprocessor Programming Manual. Motorola Semiconductor Products, Inc., Phoenix, Arizona, 1975.

13. F8 Microprocessor Programmer's Guide. Fairchild Semiconductor, Mountain View, California, 1975.

VIII. ACKNOWLEDGEMENTS

The author wishes to thank A. V. Pohm for his years of enthusiasm and support. The author wishes to thank W. B. Boast for his understanding and patience and support. Special thanks are due to the author's parents and to his wife-typist.

This work was partially supported by the Iowa State Affiliates Program in Solid State Electronics.

IX. APPENDIX A: EXAMPLE PROGRAM PRINTOUT

The following pages are the computer printout for the example problem described in section V of this paper. The actual printout was on standard 132 column paper and has been compressed somewhat to fit this page size. This example illustrates use of some of the major features available to the user. It is for a simple two processor system.

The first option invoked is the MEMORY option. The subsequent cards provide hexidecimal values to be loaded to the primary simulated storage area. Then the PORT option is used to define the port identifier. Then the FIFO option describes port identifiers for the first-in-first-out files, including dummy values. Next the SWAP option is employed to access the secondary simulated storage area. The MEMORY option is again invoked to load the secondary memory. The PORT option defines the second set of ports, including input data stream values. The FIFO option defines secondary access identifier values for the first-in-first-out-buffers, including dummy values. After these preliminary steps the PROCESSOR option is used to start the simulation trace. When the trace halts, the SWAP option is used to reaccess the primary storage area. The LIST option is used to obtain a map of the primary storage area. Finally, the PROCESSOR option is invoked to start the simulation trace for a second microprocessor.

```
****************************************************************************************************

PROGRAM MP8SIM MULTIPLE SIMULATOR EXECUTES AS FOLLOWS:

 * MEMORY  FENCE         DUMP
           78;
           ; MAIN PROG TO CORRECT ERRORS AND RE ASSEMBLE CODE FOR 6800
           8E,00,FF;  LDS
           B6,01,00; FIFO1
           BD,00,21;   JSR
           97,1E;
           B6,01,00;  FIFO1
           BD,00,21;  JSR
           48;
           48;
           48;
           48;
           9B,1E;
           B7,01,01; OUT PORT
           7E,00,03;
           00; TEMP1
           00; TEMP2
           00; VALUE
           00; SYNDROME
           97,1F; SUBROUTINE   STAA VALUE
           7F,00,20; CLR SYNDROME
           97,1D;
           47;
           47;
           98,1D;
           97,1D;
           47;
           47;
           47;
           47;
           98,1D;
           84,01; ANDA IMMED
           9B,20;  ADD SYND
           97,20;
           96,1F; LDA VALUE
           97,1D;    STA TEMP1
           48;
           98,1D;
           97,1D;
```

```
47;
47;
47;
47;
98,1D;
84,02; ANDA IN
9B,20; ADDA SYND
97,20;
96,1F;  LDA VALUE
97,1D;
48;
98,1D;
47;
97,1D;
48;
48;
98,1D;
84,04;
9B,20; ADDA SYND;
C6,80; LDAB IMMED
4D;
27,04; BEQ
54;  LSRB
4A;     DECA
20,F9;  BRA
D8,1F; EORB VALUE
C4,17; ANDB IM
17; TBA
C0,07; SUBB IM
2E,02; BGT
20,04; BRA
80,10; SUBA IM
8B,08; ADDA IM;
39;      RTS
. END OF 6800 MEMORY LOAD
```

```
 ** LISTING OF MEMORY            1                    FOLLOWS:
0000:  8E  00  FF  36  01  00  8D  00  21  97  1E  86  01  00  8D  00
0010:  21  48  48  48  48  9B  1E  87  01  01  7E  00  03  00  00  00
0020:  00  97  1F  7F  00  20  97  1D  47  47  98  1D  97  1D  47  47
0030:  47  47  98  1D  84  01  9B  20  97  20  96  1F  97  1D  48  98
0040:  1D  97  1D  47  47  47  47  98  1D  84  02  9B  20  97  20  96
0050:  1F  97  1D  48  98  1D  47  97  1D  48  48  98  1D  84  04  9B
0060:  20  C6  80  4D  27  04  54  4A  20  F9  D8  1F  C4  17  17  C0
0070:  07  2E  02  20  04  80  10  8B  08  39  00| 00  00  00  00  00
0080:  00  00  00  00  00  00  00  00  00  00  00  00  00  00  00  00
0090:  00  00  00  00  00  00  00  00  00  00  00  00  00  00  00  00
00A0:  00  00  00  00  00  00  00  00  00  00  00  00  00  00  00  00
00B0:  00  00  00  00  00  00  00  00  00  00  00  00  00  00  00  00
00C0:  00  00  00  00  00  00  00  00  00  00  00  00  00  00  00  00
00D0:  00  00  00  00  00  00  00  00  00  00  00  00  00  00  00  00
00E0:  00  00  00  00  00  00  00  00  00  00  00  00  00  00  00  00
00F0:  00  00  00  00  00  00  00  00  00  00  00  00  00  00  00  00

 _


*PORT   OUT
          101::
          . END OF 6800 PORT DEFINITIONS


*FIFO
          104,100,102,103;


*SWAP MEMORY PORT
**NOW IN SPACE                    2
```

```
*MEMORY   FENCE        DUMP
           52;
           31,FF,00;             LXI SP    FF
           01,02,02;             TWICE
           DB,03;
           47;
           E6,0F;
           CD,23,00;
           CD,2D,00;
           78;
           E6,F0;
           0F;
           0F;
           0F;
           0F;
           CD,23,00;
           CD,2D,00;
           0D;
           C2,06,00;
           76;
           5F;     SHIFT
           E6,07;
           57;
           78;
           E6,08;
           07;
           92;
           C9;    RET
           5F;    GEN
           E6,55;
           E4,43,00;
           78;
           E6,33;
           E4,48,00;
           78;
           E6,0F;
           E4,4D,00;
           7B;
           D3,05;
           C9;    RET
           7B;   SET C1
           F6,40;
           5F;
           C9;   RET
           7B;    SET C2
           F6,20;
```

```
          5F;
          C9;   RET
          78;   SET C4
          F6,08;
          5F;
          C9;   RET
          .END OF 8080 MEMORY LOAD

 ** LISTING OF MEMORY             2                     FOLLOWS:
0000:   31   FF   00   01   02   02   DB   03   47   E6   0F   CD   23   00   CD   2D
0010:   00   78   E6   F0   0F   0F   0F   0F   CD   23   00   CD   2D   00   0D   C2
0020:   06   00   76   5F   E6   07   57   78   E6   08   07   82   C9   5F   E6   55
0030:   E4   43   00   78   E6   33   E4   48   00   78   E6   0F   E4   4D   00   78
0040:   D3   05   C9   78   F6   40   5F   C9   78   F6   20   5F   C9   78   F6   08
0050:   5F   C9|  00   00   00   00   00   00   00   00   00   00   00   00   00   00
0060:   00   00   00   00   00   00   00   00   00   00   00   00   00   00   00   00
0070:   00   00   00   00   00   00   00   00   00   00   00   00   00   00   00   00
0080:   00   00   00   00   00   00   00   00   00   00   00   00   00   00   00   00
0090:   00   00   00   00   00   00   00   00   00   00   00   00   00   00   00   00
00A0:   00   00   00   00   00   00   00   00   00   00   00   00   00   00   00   00
00B0:   00   00   00   00   00   00   00   00   00   00   00   00   00   00   00   00
00C0:   00   00   00   00   00   00   00   00   00   00   00   00   00   00   00   00
00D0:   00   00   00   00   00   00   00   00   00   00   00   00   00   00   00   00
00E0:   00   00   00   00   00   00   00   00   00   00   00   00   00   00   00   00
00F0:   00   00   00   00   00   00   00   00   00   00   00   00   00   00   00   00

  -

*PORT    IN
         3: 0A,37,F5,26,73,F6;
         . END OF PORT DEFINITION FOR 8080


*FIFO
         5,100,100,100;


*PROCESSOR 8080         DUMP
```

```
INTEL 8080 SIMULATION                                   FLAGS ARE S Z 0 AC 0 P 1 CY

 CLOCK  PC    INSTRUCTION  A  B  C  D  E  H  L   FGS  SP

   10  0000:  31  FF  00 00 00 00 00 00 00 00  02  00FF  LXI.SP
   20  0003:  01  02  02 00 02 02 00 00 00 00  02  00FF  LXI.B
   30  0006:  DB  03  __ 0A 02 02 00 00 00 00  02  00FF  IN        IN PORT 0003=>  0A
   35  0008:  47  __  __ 0A 0A 02 00 00 00 00  02  00FF  MOV B.A
   42  0009:  E6  0F  __ 0A 0A 02 00 00 00 00  06  00FF  ANI
   59  000B:  CD  23  00 0A 0A 02 00 00 00 00  06  00FD  CALL
   64  0023:  5F  __  __ 0A 0A 02 00 0A 00 00  06  00FD  MOV F.A
   71  0024:  E6  07  __ 02 0A 02 00 0A 00 00  02  00FD  ANI
   76  0026:  57  __  __ 02 0A 02 02 0A 00 00  02  00FD  MOV D.A
   81  0027:  7B  __  __ 0A 0A 02 02 0A 00 00  02  00FD  MOV A.E
   88  0028:  E6  08  __ 08 0A 02 02 0A 00 00  02  00FD  ANI
   92  002A:  07  __  __ 10 0A 02 02 0A 00 00  02  00FD  RLC
   96  002B:  B2  __  __ 12 0A 02 02 0A 00 00  06  00FD  ORA.D
  106  002C:  C9  __  __ 12 0A 02 02 0A 00 00  06  00FF  RET
  123  000E:  CD  2D  00 12 0A 02 02 0A 00 00  06  00FD  CALL
  128  002D:  5F  __  __ 12 0A 02 02 12 00 00  06  00FD  MOV E.A
  135  002E:  E6  55  __ 10 0A 02 02 12 00 00  02  00FD  ANI
  152  0030:  E4  43  00 10 0A 02 02 12 00 00  02  00FB  CPO
  157  0043:  7B  __  __ 12 0A 02 02 12 00 00  02  00FB  MOV A.E
  164  0044:  F6  40  __ 52 0A 02 02 12 00 00  02  00FB  ORI
  169  0046:  5F  __  __ 52 0A 02 02 52 00 00  02  00FB  MOV E.A
  179  0047:  C9  __  __ 52 0A 02 02 52 00 00  02  00FD  RET
  184  0033:  7B  __  __ 52 0A 02 02 52 00 00  02  00FD  MOV A.E
  191  0034:  E6  33  __ 12 0A 02 02 52 00 00  06  00FD  ANI
  202  0036:  E4  48  00 12 0A 02 02 52 00 00  06  00FD  CPO
  207  0039:  7B  __  __ 52 0A 02 02 52 00 00  06  00FD  MOV A.E
  214  003A:  E6  0F  __ 02 0A 02 02 52 00 00  02  00FD  ANI
  231  003C:  E4  4D  00 02 0A 02 02 52 00 00  02  00FB  CPO
  236  004D:  7B  __  __ 52 0A 02 02 52 00 00  02  00FB  MCV A.E
  243  004E:  F6  08  __ 5A 0A 02 02 52 00 00  06  00FB  ORI
  248  0050:  5F  __  __ 5A 0A 02 02 5A 00 00  06  00FB  MOV E.A
  258  0051:  C9  __  __ 5A 0A 02 02 5A 00 00  06  00FD  RET
  263  003F:  7B  __  __ 5A 0A 02 02 5A 00 00  06  00FD  MOV A.E
  273  0040:  D3  05  __ 5A 0A 02 02 5A 00 00  06  00FD  OUT        OUT PORT 0005=>  5A   FIFO1
  283  0042:  C9  __  __ 5A 0A 02 02 5A 00 00  06  00FF  RET
  288  0011:  78  __  __ 0A 0A 02 02 5A 00 00  06  00FF  MOV A.B
  295  0012:  E6  F0  __ 00 0A 02 02 5A 00 00  46  00FF  ANI
  299  0014:  0F  __  __ 00 0A 02 02 5A 00 00  46  00FF  RRC
  303  0015:  0F  __  __ 00 0A 02 02 5A 00 00  46  00FF  RRC
  307  0016:  0F  __  __ 00 0A 02 02 5A 00 00  46  00FF  RRC
  311  0017:  0F  __  __ 00 0A 02 02 5A 00 00  46  00FF  RRC
```

```
328  0018:  CD  23  00 00 0A 02 02 5A 00 00  46  00FD  CALL
333  0023:  5F  __  __ 00 0A 02 02 00 00 00  46  00FD  MOV E,A
340  0024:  E6  07  __ 00 0A 02 02 00 00 00  46  00FD  ANI
345  0026:  57  __  __ 00 0A 02 00 00 00 00  46  00FD  MOV D,A
350  0027:  7B  __  __ 00 0A 02 00 00 00 00  46  00FD  MOV A,E
357  0028:  E6  08  __ 00 0A 02 00 00 00 00  46  00FD  ANI
361  002A:  07  __  __ 00 0A 02 00 00 00 00  46  00FD  RLC
365  0028:  82  __  __ 00 0A 02 00 00 00 00  46  00FD  ORA,D
375  002C:  C9  __  __ 00 0A 02 00 00 00 00  46  00FF  RET
392  0018:  CD  2D  00 00 0A 02 00 00 00 00  46  00FD  CALL
397  002D:  5F  __  __ 00 0A 02 00 00 00 00  46  00FD  MOV E,A
404  002E:  E6  55  __ 00 0A 02 00 00 00 00  46  00FD  ANI
415  0030:  E4  43  00 00 0A 02 00 00 00 00  46  00FD  CPO
420  0033:  7B  __  __ 00 0A 02 00 00 00 00  46  00FD  MOV A,E
427  0034:  E6  33  __ 00 0A 02 00 00 00 00  46  00FD  ANI
438  0036:  E4  48  00 00 0A 02 00 00 00 00  46  00FD  CPO
443  0039:  7B  __  __ 00 0A 02 00 00 00 00  46  00FD  MOV A,E
450  003A:  E6  0F  __ 00 0A 02 00 00 00 00  46  00FD  ANI
461  003C:  E4  4D  00 00 0A 02 00 00 00 00  46  00FD  CPO
466  003F:  7B  __  __ 00 0A 02 00 00 00 00  46  00FD  MOV A,E
476  0040:  D3  05  __ 00 0A 02 00 00 00 00  46  00FD  OUT        OUT PORT 0005=>  00  FIFO1
486  0042:  C9  __  __ 00 0A 02 00 00 00 00  46  00FF  RET
491  001E:  0D  __  __ 00 0A 01 00 00 00 00  12  00FF  DCR,C
501  001F:  C2  06  00 00 0A 01 00 00 00 00  12  00FF  JNZ
511  0006:  DB  03  __ 37 0A 01 00 00 00 00  12  00FF  IN        IN PORT 0003=>  37
516  0008:  47  __  __ 37 37 01 00 00 00 00  12  00FF  MOV B,A
523  0009:  E6  0F  __ 07 37 01 00 00 00 00  12  00FF  ANI
540  0008:  CD  23  00 07 37 01 00 00 00 00  12  00FD  CALL
545  0023:  5F  __  __ 07 37 01 00 07 00 00  12  00FD  MOV E,A
552  0024:  E6  07  __ 07 37 01 00 07 00 00  12  00FD  ANI
557  0026:  57  __  __ 07 37 01 07 07 00 00  12  00FD  MOV D,A
562  0027:  7B  __  __ 07 37 01 07 07 00 00  12  00FD  MOV A,E
569  0028:  E6  08  __ 00 37 01 07 07 00 00  56  00FD  ANI
573  002A:  07  __  __ 00 37 01 07 07 00 00  56  00FD  RLC
577  002B:  B2  __  __ 07 37 01 07 07 00 00  12  00FD  ORA,D
587  002C:  C9  __  __ 07 37 01 07 07 00 00  12  00FF  RET
604  000E:  CD  2D  00 07 37 01 07 07 00 00  12  00FD  CALL
609  002D:  5F  __  __ 07 37 01 07 07 00 00  12  00FD  MOV E,A
616  002E:  E6  55  __ 05 37 01 07 07 00 00  16  00FD  ANI
627  0030:  E4  43  00 05 37 01 07 07 00 00  16  00FD  CPO
632  0033:  7B  __  __ 07 37 01 07 07 00 00  16  00FD  MOV A,E
639  0034:  E6  33  __ 03 37 01 07 07 00 00  16  00FD  ANI
650  0036:  E4  48  00 03 37 01 07 07 00 00  16  00FD  CPO
655  0039:  7B  __  __ 07 37 01 07 07 00 00  16  00FD  MOV A,E
662  003A:  E6  0F  __ 07 37 01 07 07 00 00  12  00FD  ANI
679  003C:  E4  4D  00 07 37 01 07 07 00 00  12  00FB  CPO
```

```
684  004D:  7B  --  --  07 37 01 07 07 00 00  12  00F6  MOV A.E
691  004E:  F6  08  --  0F 37 01 07 07 00 00  16  00FB  ORI
696  0050:  5F  --  --  0F 37 01 07 0F 00 00  16  00FB  MOV E.A
706  0051:  C9  --  --  0F 37 01 07 0F 00 00  16  00FD  RET
711  003F:  7B  --  --  0F 37 01 07 0F 00 00  16  00FD  MOV A.E
721  0040:  D3  05  --  0F 37 01 07 0F 00 00  16  00FD  OUT          OUT PORT 0005=>  0F  FIFO1
731  0042:  C9  --  --  0F 37 01 07 0F 00 00  16  00FF  RET
736  0011:  78  --  --  37 37 01 07 0F 00 00  16  00FF  MOV A.B
743  0012:  E6  F0  --  30 37 01 07 0F 00 00  16  00FF  ANI
747  0014:  0F  --  --  18 37 01 07 0F 00 00  16  00FF  RRC
751  0015:  0F  --  --  0C 37 01 07 0F 00 00  16  00FF  RRC
755  0016:  0F  --  --  06 37 01 07 0F 00 00  16  00FF  RRC
759  0017:  0F  --  --  03 37 01 07 0F 00 00  16  00FF  RRC
776  0018:  CD  23  00  03 37 01 07 0F 00 00  16  00FD  CALL
781  0023:  5F  --  --  03 37 01 07 03 00 00  16  00FD  MOV E.A
788  0024:  E6  07  --  03 37 01 07 03 00 00  16  00FD  ANI
793  0026:  57  --  --  03 37 01 03 03 00 00  16  00FD  MOV D.A
798  0027:  7B  --  --  03 37 01 03 03 00 00  16  00FD  MOV A.E
805  0028:  E6  08  --  00 37 01 03 03 00 00  56  00FD  ANI
809  002A:  07  --  --  00 37 01 03 03 00 00  56  00FD  RLC
813  002B:  B2  --  --  03 37 01 03 03 00 00  16  00FD  ORA.D
823  002C:  C9  --  --  03 37 01 03 03 00 00  16  00FF  RET
840  001B:  CD  2D  00  03 37 01 03 03 00 00  16  00FD  CALL
845  002D:  5F  --  --  03 37 01 03 03 00 00  16  00FD  MOV E.A
852  002E:  E6  55  --  01 37 01 03 03 00 00  12  00FD  ANI
869  0030:  E4  43  00  01 37 01 03 03 00 00  12  00FB  CPO
874  0043:  7B  --  --  03 37 01 03 03 00 00  12  00FB  MOV A.E
881  0044:  F6  40  --  43 37 01 03 03 00 00  12  00FB  ORI
886  0046:  5F  --  --  43 37 01 03 43 00 00  12  00FB  MOV E.A
896  0047:  C9  --  --  43 37 01 03 43 00 00  12  00FD  RET
901  0033:  7B  --  --  43 37 01 03 43 00 00  12  00FD  MOV A.E
908  0034:  E6  33  --  03 37 01 03 43 00 00  16  00FD  ANI
919  0036:  E4  48  00  03 37 01 03 43 00 00  16  00FD  CPO
924  0039:  7B  --  --  43 37 01 03 43 00 00  16  00FD  MOV A.E
931  003A:  E6  0F  --  03 37 01 03 43 00 00  16  00FD  ANI
942  003C:  E4  4D  00  03 37 01 03 43 00 00  16  00FD  CPO
947  003F:  7B  --  --  43 37 01 03 43 00 00  16  00FD  MOV A.E
957  0040:  D3  05  --  43 37 01 03 43 00 00  16  00FD  OUT          OUT PORT 0005=>  43  FIFO1
967  0042:  C9  --  --  43 37 01 03 43 00 00  16  00FF  RET
972  001E:  0D  --  --  43 37 00 03 43 00 00  56  00FF  DCR.C
982  001F:  C2  06  00  43 37 00 03 43 00 00  56  00FF  JNZ
989  0022:  76  --  --  43 37 00 03 43 00 00  56  00FF  HALT

**END OF SIMULATION
```

```
 ** LISTING OF MEMORY          2               FOLLOWS:
0000:   31  FF  00  01  02  02  08  03  47  E6  0F  CD  23  00  CD  2D
0010:   00  78  E6  F0  0F  0F  0F  0F  CD  23  00  CD  2D  00  0D  C2
0020:   06  00  76  5F  E6  07  57  7B  E6  08  07  62  C9  5F  E6  55
0030:   E4  43  00  7B  E6  33  E4  48  00  7B  E6  0F  E4  4D  00  7B
0040:   03  05  C9  7B  F6  40  5F  C9  7B  F6  20  5F  C9  7B  F6  08
0050:   5F  C9| 00  00  00  00  00  00  00  00  00  00  00  00  00  00
0060:   00  00  00  00  00  00  00  00  00  00  00  00  00  00  00  00
0070:   00  00  00  00  00  00  00  00  00  00  00  00  00  00  00  00
0080:   00  00  00  00  00  00  00  00  00  00  00  00  00  00  00  00
0090:   00  00  00  00  00  00  00  00  00  00  00  00  00  00  00  00
00A0:   00  00  00  00  00  00  00  00  00  00  00  00  00  00  00  00
00B0:   00  00  00  00  00  00  00  00  00  00  00  00  00  00  00  00
00C0:   00  00  00  00  00  00  00  00  00  00  00  00  00  00  00  00
00D0:   00  00  00  00  00  00  00  00  00  00  00  00  00  00  00  00
00E0:   00  00  00  00  00  00  00  00  00  00  00  00  00  00  00  00
00F0:   00  00  00  00  00  00  00  00  00  00  00  33  00  1E  00  00

 -

*SWAP MEMORY PORT
**NOW IN SPACE                 1
*LIST
        00,FF;

 ** LISTING OF MEMORY          1               FOLLOWS:
0000:   8E  00  FF  86  01  00  9D  00  21  97  1E  86  01  00  8D  00
0010:   21  48  48  48  48  98  1E  87  01  01  7E  00  03  00  00  00
0020:   00  97  1F  7F  00  20  97  1D  47  47  98  1D  97  1D  47  47
0030:   47  47  98  1D  84  01  98  20  97  20  96  1F  97  1D  48  98
0040:   1D  97  1D  47  47  47  47  98  1D  84  02  98  20  97  20  96
0050:   1F  97  1D  48  98  1D  47  97  1D  48  46  98  1D  94  04  9B
0060:   20  C6  80  4D  27  04  54  4A  20  F9  D8  1F  C4  17  17  C0
0070:   07  2E  02  20  04  80  10  8B  08  39  00| 00  00  00  00  00
0080:   00  00  00  00  00  00  00  00  00  00  00  00  00  00  00  00
0090:   00  00  00  00  00  00  00  00  00  00  00  00  00  00  00  00
00A0:   00  00  00  00  00  00  00  00  00  00  00  00  00  00  00  00
00B0:   00  00  00  00  00  00  00  00  00  00  00  00  00  00  00  00
00C0:   00  00  00  00  00  00  00  00  00  00  00  00  00  00  00  00
00D0:   00  00  00  00  00  00  00  00  00  00  00  00  00  00  00  00
00E0:   00  00  00  00  00  00  00  00  00  00  00  00  00  00  00  00
00F0:   00  00  00  00  00  00  00  00  00  00  00  00  00  00  00  00

 -
```

```
*PROCESSOR 6800 RESET START   DUMP
        00:

 MOTOROLA 6800 SIMULATION                                    FLAGS ARE 1 1 H I N Z V C

   CLOCK   PC    INSTRUCTION  A  B    INDX   FGS SP
-
       3  0000:   8E   00   FF 00 00  0000   D0  00FF  LDS    IM
       7  0003:   B6   01   00 5A 00  0000   D0  00FF  LDAA   EX  IN PORT 0100=>  5A  FIF01
      16  0006:   BD   00   21 5A 00  0000   D0  00FD  JSR    EX
      20  0021:   97   1F   __ 5A 00  0000   D0  00FD  STAA    D
      26  0023:   7F   00   20 5A 00  0000   D4  00FD  CLR    EX
      30  0026:   97   1D   __ 5A 00  0000   D0  00FD  STAA    D
      32  0028:   47   __   __ 2D 00  0000   D0  00FD  ASRA
      34  0029:   47   __   __ 16 00  0000   D3  00FD  ASRA
      37  002A:   98   1D   __ 4C 00  0000   D1  00FD  EORA    D
      41  002C:   97   1D   __ 4C 00  0000   D1  00FD  STAA    D
      43  002E:   47   __   __ 26 00  0000   D0  00FD  ASRA
      45  002F:   47   __   __ 13 00  0000   D0  00FD  ASRA
      47  0030:   47   __   __ 09 00  0000   D3  00FD  ASRA
      49  0031:   47   __   __ 04 00  0000   D3  00FD  ASRA
      52  0032:   98   1D   __ 48 00  0000   D1  00FD  EORA    D
      54  0034:   84   01   __ 00 00  0000   D5  00FD  ANDA   IM
      57  0036:   9B   20   __ 00 00  0000   D4  00FD  ADDA    D
      61  0038:   97   20   __ 00 00  0000   D4  00FD  STAA    D
      64  003A:   96   1F   __ 5A 00  0000   D0  00FD  LDAA    D
      68  003C:   97   1D   __ 5A 00  0000   D0  00FD  STAA    D
      70  003E:   48   __   __ B4 00  0000   DA  00FD  ASLA
      73  003F:   98   1D   __ EE 00  0000   D8  00FD  EORA    D
      77  0041:   97   1D   __ EE 00  0000   D8  00FD  STAA    D
      79  0043:   47   __   __ F7 00  0000   DA  00FD  ASRA
      81  0044:   47   __   __ FB 00  0000   D9  00FD  ASRA
      83  0045:   47   __   __ FD 00  0000   D9  00FD  ASRA
      85  0046:   47   __   __ FE 00  0000   D9  00FD  ASRA
      88  0047:   98   1D   __ 10 00  0000   D1  00FD  EORA    D
      90  0049:   84   02   __ 00 00  0000   D5  00FD  ANDA   IM
      93  004B:   9B   20   __ 00 00  0000   D4  00FD  ADDA    D
      97  004D:   97   20   __ 00 00  0000   D4  00FD  STAA    D
     100  004F:   96   1F   __ 5A 00  0000   D0  00FD  LDAA    D
     104  0051:   97   1D   __ 5A 00  0000   D0  00FD  STAA    D
     106  0053:   48   __   __ B4 00  0000   DA  00FD  ASLA
     109  0054:   98   1D   __ EE 00  0000   D8  00FD  EORA    D
     111  0056:   47   __   __ F7 00  0000   DA  00FD  ASRA
     115  0057:   97   1D   __ F7 00  0000   D8  00FD  STAA    D
     117  0059:   48   __   __ EE 00  0000   D9  00FD  ASLA
```

```
119  005A:  48  __  __ DC 00  0000  D9  00FD  ASLA
122  005B:  98  1D  __ 28 00  0000  D1  00FD  EORA   D
124  005D:  84  04  __ 00 00  0000  D5  0CFD  ANDA  IM
127  005F:  9B  20  __ 00 00  0000  D4  00FD  ADDA   D
129  0061:  C6  80  __ 00 80  0000  D8  00FD  LDAB  IM
131  0063:  4D  __  __ 00 80  0000  D4  00FD  TSTA
135  0064:  27  04  __ 00 80  0000  D4  00FD  BEQ    R
138  006A:  D8  1F  __ 00 DA  0000  D8  00FD  EORB   D
140  006C:  C4  17  __ 00 12  0000  D0  00FD  ANDB  IM
142  006E:  17  __  __ 12 12  0000  D0  00FD  TBA
144  006F:  C0  07  __ 12 0B  0000  D0  00FD  SUBB  IM
148  0071:  2E  02  __ 12 0B  0000  D0  00FD  BGT    R
150  0075:  80  10  __ 02 0B  0000  D0  00FD  SUBA  IM
152  0077:  8B  08  __ 0A 0B  0000  D0  00FD  ADDA  IM
157  0079:  39  __  __ 0A 0B  0000  D0  00FF  RTS
161  0009:  97  1E  __ 0A 0B  0000  D0  00FF  STAA   D
165  000B:  96  01  00 00 0B  0000  D4  00FF  LDAA  EX IN PORT 0100=>  00  FIFO1
174  000E:  BD  00  21 00 0B  0000  D4  00FD  JSR   EX
178  0021:  97  1F  __ 00 0B  0000  D4  00FD  STAA   D
184  0023:  7F  00  20 00 0B  0000  D4  00FD  CLR   EX
188  0026:  97  1D  __ 00 0B  0000  D4  00FD  STAA   D
190  0028:  47  __  __ 00 0B  0000  D4  00FD  ASRA
192  0029:  47  __  __ 00 0B  0000  D4  00FD  ASRA
195  002A:  98  1D  __ 00 0B  0000  D4  00FD  EORA   D
199  002C:  97  1D  __ 00 0B  0000  D4  00FD  STAA   D
201  002E:  47  __  __ 00 0B  0000  D4  00FD  ASRA
203  002F:  47  __  __ 00 0B  0000  D4  00FD  ASRA
205  0030:  47  __  __ 00 0B  0000  D4  00FD  ASRA
207  0031:  47  __  __ 00 0B  0000  D4  00FD  ASRA
210  0032:  98  1D  __ 00 0B  0000  D4  00FD  EORA   D
212  0034:  84  01  __ 00 0B  0000  D4  00FD  ANDA  IM
215  0036:  9B  20  __ 00 0B  0000  D4  00FD  ADDA   D
219  0038:  97  20  __ 00 0B  0000  D4  00FD  STAA   D
222  003A:  96  1F  __ 00 0B  0000  D4  00FD  LDAA   D
226  003C:  97  1D  __ 00 0B  0000  D4  00FD  STAA   D
228  003E:  48  __  __ 00 0B  0000  D4  00FD  ASLA
231  003F:  98  1D  __ 00 0B  0000  D4  00FD  EORA   D
235  0041:  97  1D  __ 00 0B  0000  D4  00FD  STAA   D
237  0043:  47  __  __ 00 0B  0000  D4  00FD  ASRA
239  0044:  47  __  __ 00 0B  0000  D4  00FD  ASRA
241  0045:  47  __  __ 00 0B  0000  D4  00FD  ASRA
243  0046:  47  __  __ 00 0B  0000  D4  00FD  ASRA
246  0047:  98  1D  __ 00 0B  0000  D4  00FD  EORA   D
248  0049:  84  02  __ 00 0B  0000  D4  00FD  ANDA  IM
251  004B:  9B  20  __ 00 0B  0000  D4  00FD  ADDA   D
255  004D:  97  20  __ 00 0B  0000  D4  00FD  STAA   D
```

```
258  004F:  96  1F  __ 00 0B  0000  D4  00FD  LDAA   D
262  0051:  97  1D  __ 00 0B  0000  D4  00FD  STAA   D
264  0053:  48  __  __ 00 0B  0000  D4  00FD  ASLA
267  0054:  98  1D  __ 00 0B  0000  D4  00FD  EORA   D
269  0056:  47  __  __ 00 0B  0000  D4  00FD  ASRA
273  0057:  97  1D  __ 00 0B  0000  D4  00FD  STAA   D
275  0059:  48  __  __ 00 0B  0000  D4  00FD  ASLA
277  005A:  48  __  __ 00 0B  0000  D4  00FD  ASLA
280  005B:  98  1D  __ 00 0B  0000  D4  00FD  EORA   D
282  005D:  84  04  __ 00 0B  0000  D4  00FD  ANDA   IM
285  005F:  9B  20  __ 00 0B  0000  D4  00FD  ADDA   D
287  0061:  C6  80  __ 00 80  0000  D8  00FD  LDAB   IM
289  0063:  4D  __  __ 00 80  0000  D4  00FD  TSTA
293  0064:  27  04  __ 00 80  0000  D4  00FD  BEQ    R
296  006A:  D8  1F  __ 00 80  0000  D8  00FD  EORB   D
298  006C:  C4  17  __ 00 00  0000  D4  00FD  ANDS   IM
300  006E:  17  __  __ 00 00  0000  D4  00FD  TBA
302  006F:  C0  07  __ 00 F9  0000  D9  00FD  SUBB   IM
306  0071:  2E  02  __ 00 F9  0000  D9  00FD  BGT    R
310  0073:  20  04  __ 00 F9  0000  D9  00FD  BRA    R
315  0079:  39  __  __ 00 F9  0000  D9  00FF  RTS
317  0011:  48  __  __ 00 F9  0000  D4  00FF  ASLA
319  0012:  48  __  __ 00 F9  0000  D4  00FF  ASLA
321  0013:  48  __  __ 00 F9  0000  D4  00FF  ASLA
323  0014:  48  __  __ 00 F9  0000  D4  00FF  ASLA
326  0015:  9B  1E  __ 0A F9  0000  D0  00FF  ADDA   D
331  0017:  B7  01  01 0A F9  0000  D0  00FF  STAA   EX  OUT PORT 0101=>  0A
334  001A:  7E  00  03 0A F9  0000  D0  00FF  JMP    EX
338  0003:  36  01  00 0F F9  0000  D0  00FF  LDAA   EX IN PORT 0100=>  0F  FIF01
347  0006:  9D  00  21 0F F9  0000  D0  00FD  JSR    EX
351  0021:  97  1F  __ 0F F9  0000  D0  00FD  STAA   D
357  0023:  7F  00  20 0F F9  0000  D4  00FD  CLR    EX
361  0026:  97  1D  __ 0F F9  0000  D0  00FD  STAA   D
363  0028:  47  __  __ 07 F9  0000  D3  00FD  ASRA
365  0029:  47  __  __ 03 F9  0000  D3  00FD  ASRA
368  002A:  98  1D  __ 0C F9  0000  D1  00FD  EORA   D
372  002C:  97  1D  __ 0C F9  0000  D1  00FD  STAA   D
374  002E:  47  __  __ 06 F9  0000  D0  00FD  ASRA
376  002F:  47  __  __ 03 F9  0000  D0  00FD  ASRA
378  0030:  47  __  __ 01 F9  0000  D3  00FD  ASRA
380  0031:  47  __  __ 00 F9  0000  D7  00FD  ASRA
383  0032:  98  1D  __ 0C F9  0000  D1  00FD  EORA   D
385  0034:  84  01  __ 00 F9  0000  D5  00FD  ANDA   IM
388  0036:  9B  20  __ 00 F9  0000  D4  00FD  ADDA   D
392  0038:  97  20  __ 00 F9  0000  D4  00FD  STAA   D
395  003A:  96  1F  __ 0F F9  0000  D0  00FD  LDAA   D
```

```
MOTOROLA 6800 SIMULATICN                                     FLAGS ARE 1 1 H I N Z V C

  CLOCK  PC    INSTRUCTICN  A  B   INDX  FGS SP

    399  003C:  97  1D  __ 0F F9  0000  D0  00FD  STAA   D
    401  003E:  48  __  __ 1E F9  0000  D0  00FD  ASLA
    404  003F:  98  1D  __ 11 F9  0000  D0  00FD  EORA   D
    408  0041:  97  1D  __ 11 F9  0000  D0  00FD  STAA   D
    410  0043:  47  __  __ 08 F9  0000  D3  00FD  ASRA
    412  0044:  47  __  __ 04 F9  0000  D0  00FD  ASRA
    414  0045:  47  __  __ 02 F9  0000  D0  00FD  ASRA
    416  0046:  47  __  __ 01 F9  0000  D0  00FD  ASRA
    419  0047:  98  1D  __ 10 F9  0000  D0  00FD  EORA   D
    421  0049:  84  02  __ 00 F9  0000  D4  00FD  ANDA  IM
    424  004B:  9B  20  __ 00 F9  0000  D4  00FD  ADDA   D
    428  004D:  97  20  __ 00 F9  0000  D4  00FD  STAA   D
    431  004F:  96  1F  __ 0F F9  0000  D0  00FD  LDAA   D
    435  0051:  97  1D  __ 0F F9  0000  D0  00FD  STAA   D
    437  0053:  48  __  __ 1E F9  0000  D0  00FD  ASLA
    440  0054:  98  1D  __ 11 F9  0000  D0  00FD  EORA   D
    442  0056:  47  __  __ 08 F9  0000  D3  00FD  ASRA
    446  0057:  97  1D  __ 08 F9  0000  D1  00FD  STAA   D
    448  0059:  48  __  __ 10 F9  0000  D0  00FD  ASLA
    450  005A:  48  __  __ 20 F9  0000  D0  00FD  ASLA
    453  005B:  98  1D  __ 28 F9  0000  D0  00FD  EORA   D
    455  005D:  84  04  __ 00 F9  0000  D4  00FD  ANDA  IM
    458  005F:  9B  20  __ 00 F9  0000  D4  00FD  ADDA   D
    460  0061:  C6  80  __ 00 80  0000  D8  00FD  LDAB  IM
    462  0063:  4D  __  __ 00 80  0000  D4  00FD  TSTA
    466  0064:  27  04  __ 00 80  0000  D4  00FD  BEQ    R
    469  006A:  D8  1F  __ 00 8F  0000  D8  00FD  EORB   D
    471  006C:  C4  17  __ 00 07  0000  D0  00FD  ANDB  IM
    473  006E:  17  __  __ 07 07  0000  D0  00FD  TBA
    475  006F:  C0  07  __ 07 00  0000  D4  00FD  SUBB  IM
    479  0071:  2E  02  __ 07 00  0000  D4  00FD  BGT    R
    483  0073:  20  04  __ 07 00  0000  D4  00FD  BRA    R
    488  0079:  39  __  __ 07 00  0000  D4  00FF  RTS
    492  0009:  97  1E  __ 07 00  0000  D0  00FF  STAA   D
    496  000B:  B6  01  00 43 00  0000  D0  00FF  LDAA  EX IN PORT 0100=>  43  FIFO1
    505  000E:  BD  00  21 43 00  0000  D0  00FD  JSR   EX
    509  0021:  97  1F  __ 43 00  0000  D0  00FD  STAA   D
    515  0023:  7F  00  20 43 00  0000  D4  00FD  CLR   EX
    519  0026:  97  1D  __ 43 00  0000  D0  00FD  STAA   D
```

```
521   0028:   47   __   __  21 00   0000   D3   00FD   ASRA
523   0029:   47   __   __  10 00   0000   D3   00FD   ASRA
526   002A:   98   1D   __  53 00   0000   D1   00FD   EORA    D
530   002C:   97   1D   __  53 00   0000   D1   00FD   STAA    D
532   002E:   47   __   __  29 00   0000   D3   00FD   ASRA
534   002F:   47   __   __  14 00   0000   D3   00FD   ASRA
536   0030:   47   __   __  0A 00   0000   D0   00FD   ASRA
538   0031:   47   __   __  05 00   0000   D0   00FD   ASRA
541   0032:   98   1D   __  56 00   0000   D0   00FD   EORA    D
543   0034:   84   01   __  00 00   0000   D4   00FD   ANDA   IM
546   0036:   9B   20   __  00 00   0000   D4   00FD   ADDA    D
550   0038:   97   20   __  00 00   0000   D4   00FD   STAA    D
553   003A:   96   1F   __  43 00   0000   D0   00FD   LDAA    D
557   003C:   97   1D   __  43 00   0000   D0   00FD   STAA    D
559   003E:   48   __   __  86 00   0000   DA   00FD   ASLA
562   003F:   98   1D   __  C5 00   0000   D8   00FD   EORA    D
566   0041:   97   1D   __  C5 00   0000   D8   00FD   STAA    D
568   0043:   47   __   __  E2 00   0000   D9   00FD   ASRA
570   0044:   47   __   __  F1 00   0000   DA   00FD   ASRA
572   0045:   47   __   __  F8 00   0000   D9   00FD   ASRA
574   0046:   47   __   __  FC 00   0000   DA   00FD   ASRA
577   0047:   98   1D   __  39 00   0000   D0   00FD   EORA    D
579   0049:   84   02   __  00 00   0000   D4   00FD   ANDA   IM
582   004B:   9B   20   __  00 00   0000   D4   00FD   ADDA    D
586   004D:   97   20   __  00 00   0000   D4   00FD   STAA    D
589   004F:   96   1F   __  43 00   0000   D0   00FD   LDAA    D
593   0051:   97   1D   __  43 00   0000   D0   00FD   STAA    D
595   0053:   48   __   __  86 00   0000   DA   00FD   ASLA
598   0054:   98   1D   __  C5 00   0000   D8   00FD   EORA    D
600   0056:   47   __   __  E2 00   0000   D9   00FD   ASRA
604   0057:   97   1D   __  E2 00   0000   D9   00FD   STAA    D
606   0059:   48   __   __  C4 00   0000   D9   00FD   ASLA
608   005A:   48   __   __  88 00   0000   D9   00FD   ASLA
611   005B:   98   1D   __  6A 00   0000   D1   00FD   EORA    D
613   005D:   84   04   __  00 00   0000   D5   00FD   ANDA   IM
616   005F:   9B   20   __  00 00   0000   D4   00FD   ADDA    D
618   0061:   C6   80   __  00 80   0000   D8   00FD   LDAB   IM
620   0063:   4D   __   __  00 80   0000   D4   00FD   TSTA
624   0064:   27   04   __  00 80   0000   D4   00FD   BEQ     R
627   006A:   D8   1F   __  00 C3   0000   D8   00FD   EORB    D
629   006C:   C4   17   __  00 03   0000   D0   00FD   ANDB   IM
631   006E:   17   __   __  03 03   0000   D0   00FD   TBA
633   006F:   C0   07   __  03 FC   0000   D9   00FD   SUBB   IM
637   0071:   2E   02   __  03 FC   0000   D9   00FD   BGT     R
641   0073:   20   04   __  03 FC   0000   D9   00FD   BRA     R
```

```
      646  0079:   39   __   __ 03 FC   0000   09   00FF   RTS
      648  0011:   48   __   __ 06 FC  `0000   D0   00FF   ASLA
      650  0012:   48   __   __ 0C FC   0000   D0   00FF   ASLA
      652  0013:   48   __   __ 18 FC   0000   D0   00FF   ASLA
      654  0014:   48   __   __ 30 FC   0000   D0   00FF   ASLA
      657  0015:   9B   1E   __ 37 FC   0000   D0   00FF   ADDA    D
      662  0017:   B7   01   01 37 FC   0000   D0   00FF   STAA   EX   OUT PORT 0101=>   37
      665  001A:   7E   00   03 37 FC   0000   D0   00FF   JMP    EX
****FIFO UNDERFLO#S                  1
      669  0003:   B6   01   00 00 FC   0000   D4   00FF   LDAA   EX IN PORT 0100=>   00   FIFO1

**END OF SIMULATION




 ** LISTING OF MEMORY             1                     FOLLOWS:
0000:   8E   00   FF   B6   01   00   BD   00   21   97   1E   B6   01   00   BD   00
0010:   21   48   48   48   48   9B   1E   B7   01   01   7E   00   03   E2   07   43
0020:   00   97   1F   7F   00   20   97   1D   47   47   98   1D   97   1D   47   47
0030:   47   47   98   1D   84   01   9B   20   97   20   96   1F   97   1D   48   98
0040:   1D   97   1D   47   47   47   47   98   1D   84   02   9B   20   97   20   96
0050:   1F   97   1D   48   98   1D   47   97   1D   48   48   98   1D   84   04   9B
0060:   20   C6   80   4D   27   04   54   4A   20   F9   D8   1F   C4   17   17   C0
0070:   07   2E   02   20   04   80   10   8B   08   39   00|  00   00   00   00   00
0080:   00   00   00   00   00   00   00   00   00   00   00   00   00   00   00   00
0090:   00   00   00   00   00   00   00   00   00   00   00   00   00   00   00   00
00A0:   00   00   00   00   00   00   00   00   00   00   00   00   00   00   00   00
00B0:   00   00   00   00   00   00   00   00   00   00   00   00   00   00   00   00
00C0:   00   00   00   00   00   00   00   00   00   00   00   00   00   00   00   00
00D0:   00   00   00   00   00   00   00   00   00   00   00   00   00   00   00   00
00E0:   00   00   00   00   00   00   00   00   00   00   00   00   00   00   00   00
00F0:   00   00   00   00   00   00   00   00   00   00   00   00   00   00   00   11

 -


** NO MORE *OPTIONS CARDS FOUND                         TERMINATES
```

X. APPENDIX B: LISTING OF PL/1 SOURCE DECK FOR THE SIMULATOR

The contents of this Appendix are the card images of the source deck for the current version of the simulator program. The actual PL/1 text, of course, is the most accurate description of the simulator available. The source statements are, with a few exceptions, hierarchically indented to somewhat improve readability. The multiple statement per card feature of PL/1 was extensively used to help maintain integrity of the deck. The program is structured to give preference to forward directed GO TO statements.

```
MP8SIM: PROCEDURE OPTIONS(MAIN);
/* WRITTEN 1975 AND 1976 BY DONALD E. CHAPMAN  */
     DCL FETCH ENTRY(BINARY FIXED(8), BINARY FIXED(31));
     DCL STORE ENTRY(BINARY FIXED(8) , BINARY FIXED(31));
     DCL DUMP ENTRY (BINARY FIXED(31),BINARY FIXED(31));
     DCL HEXIN ENTRY(BINARY FIXED(31),CHAR(1)         ,BINARY FIXED);
     DCL DECIN ENTRY(BINARY FIXED(31),CHAR(1),BINARY FIXED);
     DCL TOHEX ENTRY (BINARY FIXED(31),CHAR(4));
     DCL FIFO ENTRY(BINARY FIXED,BIT(1),BINARY FIXED(8));
     DCL ADD ENTRY(BINARY FIXED(8),BINARY FIXED(8),BINARY FIXED(8),
       BIT(1),BIT(1),BIT(1),BIT(1),BIT(1),BIT(1),BIT(1));
     DCL LFLAGS ENTRY(BIT(1),BIT(1),BIT(1),BIT(1),BINARY FIXED(8));
     DCL TELIN ENTRY(BINARY FIXED(8),BINARY FIXED);
     DCL PUSH ENTRY(BINARY FIXED,BINARY FIXED,BINARY FIXED(8),
       BINARY FIXED(31));
     DCL POP ENTRY(BINARY FIXED,BINARY FIXED,BINARY FIXED(8),
       BINARY FIXED(31));
     DCL DNK ENTRY (BINARY FIXED,BINARY FIXED);
     DCL EX(4,0:255) LABEL;
     DCL MEM(4,16384) BINARY FIXED(8) EXTERNAL INITIAL(
       (16384)0,(16384)0,(16384)0,(16384)0);
     DCL MEM2(4,16384) BINARY FIXED INITIAL(
       (16384)0,(16384)0,(16384)0,(16384)0);
     DCL TIMINC(4,0:255) BINARY FIXED INITIAL(
       (2)8,10,6,8,10,16,6,(6)((3)10,6,16,10,16,6),(4)6,16,10,18,6,        TI2
       (2)((3)(18,16),22,16),(6)((3)(18,12),22,12),                        TI3
       (15)((7)10,16),(7)14,8,                                             TI4
       (4)(4,10,7,(3)5,7,4),(2)(4,10,16,(3)5,7,4),4,10,13,5,(3)10,         TI5
       (2)4,10,13,(3)5,7,4,   (6)((6)5,7,5),(8)7,(6)5,7,5,(8)((6)4,7,4)    TI6
       ,5,(3)10,(2)11,7,11,5,(3)10,11,17,7,11,(2)(5,(3)10,(2)11,7,11),     TI7
       5,(2)10,18,(2)11,7,11,(2)5,10,4,(2)11,7,11,5,(2)10,4,(2)11,         TI8
       7,11,(2)5,10,4,(2)11,7,11,                                          TI9
       (8)2,(2)4,(22)2,(24)4,(3)5,10,(3)9,12,                             TI10
       (32)2,(14)7,4,7,(14)6,3,6,                                         TI11
       (12)2,3,8,(9)3,4,(4)3,(3)4,(8)5,6,(4)5,6,8,6,7,(7)4,5,(4)4,5,9,    TI12
```

```
  5,6,  (12)2,(11)3,4,(6)3,4,(8)5,6,(6)5,6,7,(7)4,5,(6)4,5,6  ,          TI13
  (8)4,(2)16,4,(7)16,(4)4,(2)10,4,4,(4)8,4,4,(6)10,                      TI14
  (2)16,26,22,24,4,8,(19)6,(64)4,                                        TI15
  (8)12,(7)10,8,(16)12,(2)8,(14)16,(2)8,(14)16,                          TI16
  (16)4,(16)8,(32)4)    ;                                                TI17
DCL DELPC(4,0:255) BINARY FIXED INITIAL(
  (8)((4)1,2,1,2,1),(32)(3,1),(128)1,                                     PC2
  1,3,(4)1,2,(7)1,2,(2)1,3,(4)1,2,1,3,(5)1,2,(2)1,(2)3,(3)1,              PC3
  2,(3)1,3,(3)1,2,(2)1,(2)3,(3)1,2,(3)1,3,(3)1,2,(131)1,(3)3,             PC4
  1,2,(3)1,(4)3,2,1,(2)((2)1,3,2,3,1,2,1),(4)((2)1,3,1,3,1,2,1),          PC5
  (32)1,(16)2,(16)1,(32)1,(16)2,(16)3,(12)2,3,2,3,(33)2,(16)3,            PC6
  (12)2,(4)3,(32)2,(16)3 ,                                                PC7
  (32)1,(8)2,(3)3,(21)1,(64)1,(8)2,(7)1,(17)2,(32)1,(64)1) ;              PC8
DCL DELTIM(4) BINARY FIXED INITIAL(4,6,0,2);
DCL FENCE BINARY FIXED(31) EXTERNAL INITIAL(-1);
DCL FENCE2 BINARY FIXED(31) INITIAL(-1);
DCL TPOIN BINARY FIXED(31);
DCL (DC,DC1,PC1) BINARY FIXED(31) INITIAL(0);
DCL ITIME BINARY FIXED(31) INITIAL(0);
DCL TLIM BINARY FIXED(31);
DCL LINE BINARY FIXED(31) INITIAL(0) EXTERNAL;
DCL(ROMST,ROMEND) BINARY FIXED(31) INITIAL(-1) EXTERNAL;
DCL MEMSIZ2 BINARY FIXED(31) INITIAL(255);
DCL(ROMST2,ROMEND2) BINARY FIXED(31)      INITIAL(-1)   ;
DCL MEMSIZ BINARY FIXED(31) EXTERNAL INITIAL(255);
DCL (LOADP,VALUE) BINARY FIXED(31);
DCL IPORT BINARY FIXED(31) EXTERNAL;
DCL INVAL BINARY FIXED (31) EXTERNAL;
DCL MAXI BINARY FIXED(31);
DCL PORTID(32) BINARY FIXED(31) INITIAL((32)-1) EXTERNAL;
DCL PSTART(32) BINARY FIXED(31) INITIAL((32)-1) EXTERNAL;
DCL PEND(0:32) BINARY FIXED(31) INITIAL((33)-1) EXTERNAL;
DCL PORTLT(32) BINARY FIXED(31) EXTERNAL INITIAL((32)-1);
DCL PORTID2(32) BINARY FIXED(31) INITIAL((32)-1);
DCL PSTART2(32) BINARY FIXED(31);
DCL PEND2(32) BINARY FIXED(31) INITIAL((32)-1);
DCL PORTLT2(32) BINARY FIXED(31) INITIAL((32)-1);
DCL STK08(0:7) BINARY FIXED(31) INITIAL((8)0);
DCL ILPOIN BINARY FIXED(31);
DCL (K,M) BINARY FIXED(31);
DCL MSIZ BINARY FIXED(31) ;
DCL (FI1,FO1,FI2,FO2) BINARY FIXED(31) EXTERNAL INITIAL(-3);
DCL (FI12,FO12,FI22,FO22) BINARY FIXED(31) INITIAL(-3);
DCL (INDX,ADX) BINARY FIXED(31) INITIAL(0);
DCL (PC,PCN,SP) BINARY FIXED(31) INITIAL(0);
DCL PC2 BINARY FIXED(31) INITIAL(0);
DCL PCN2 BINARY FIXED(31) INITIAL(0);
DCL STOP BINARY FIXED(31) INITIAL(-1);
DCL (PORT,OUT) BINARY FIXED(31) EXTERNAL;
DCL BGD BINARY FIXED(31);
DCL(SP2,INDX2,DC2,DC12,PC12,ITIME2)BINARY FIXED(31) INITIAL(0);
DCL STK082(0:7)BINARY FIXED(31) INITIAL((8)0);
DCL (ISAP2,SP082) BINARY FIXED INITIAL(0);
DCL SWCT BINARY FIXED INITIAL(1) EXTERNAL;
DCL TEMPB BINARY FIXED;
DCL MACH2 BINARY FIXED INITIAL(0);
DCL INS BINARY FIXED    ; /* INSTR. REG.*/
DCL PT BINARY FIXED;
DCL ISAR BINARY FIXED INITIAL(0);
DCL MACH BINARY FIXED EXTERNAL INITIAL(0);
DCL ILISP BINARY FIXED;
DCL SP08 BINARY FIXED INITIAL(0);
DCL COL BINARY FIXED;
DCL FIP(2) BINARY FIXED EXTERNAL INITIAL((2)0);
DCL FOP(2) BINARY FIXED EXTERNAL INITIAL((2)0);
DCL FINU BINARY FIXED EXTERNAL;
DCL IDATA(4096 ) BINARY FIXED(8) INITIAL((4096 )0)EXTERNAL;
DCL IDATA2(4096) BINARY FIXED(8) INITIAL((4096)0);
DCL FIFS(2,2048) EXTERNAL BINARY FIXED(8) INITIAL((4096)0);
DCL SCF8(0:63) BINARY FIXED(8) EXTERNAL INITIAL((64)0);
DCL SCFB2(0:63) BINARY FIXED(8) INITIAL((64)0);
DCL TEMPA BINARY FIXED(8);
DCL SD BINARY FIXED(8);
DCL(A,B,C,D,E,H,L) BINARY FIXED(8) INITIAL(0);
DCL(A2,B2,C2,D2,E2,H2,L2) BINARY FIXED(8) INITIAL(0);
DCL CARD CHAR(80) EXTERNAL;
DCL MNE(4,0:255) CHAR(8) INITIAL(
   'HLT     ','HLT     ','RLC     ','RNC     ','ADI     ',
   'RST 0   ','MVI A   ','RET     ','INR B   ','DCR B   ',
   'RRC     ','RNZ     ','ACI     ','RST 1   ','MVI B   ',
   'RET     ','INR C   ','DCR C   ','RAL     ','RP      ',
   'SUI     ','RST 2   ','MVI C   ','RET     ','INR D   ',
   'DCR D   ','RAR     ','RPO     ','SBI     ','RST 3   ',
   'MVI D   ','RET     ','INR E   ','DCR E   ','***ERROR',
   'RC      ','ANI     ','RST 4   ','MVI E   ','RET     ',
   'INR H   ','DCR H   ','***ERROR','RZ      ','XRI     ',
   'RST 5   ','MVI H   ','RET     ','INR L   ','DCR L   ',
   '***ERROR','RM      ','ORI     ','RST 6   ','MVI L   ',
   'RET     ','***ERROR','***ERROR','***ERROR','RPE     ',
   'CPI     ','RST 7   ','MVI M   ','RET     ','JNC     ',
   'IN 0    ','CNC     ','IN 1    ','JMP     ','IN 2    ',
   'CALL    ','IN 3    ','JNZ     ','IN 4    ','CNC     ',
   'IN 5    ','JMP     ','IN 6    ','CALL    ','IN 7    ',
   'JP      ','OUT 8   ','CP      ','OUT 9   ','JMP     ',
   'OUT A   ','CALL    ','OUT B   ','JPO     ','OUT C   ',
   'CPO     ','OUT D   ','JMP     ','OUT E   ','CALL    ',
   'OUT F   ','JC      ','OUT 10  ','CC      ','OUT 11  ',
   'JMP     ','OUT 12  ','CALL    ','OUT 13  ','JZ      ',
   'OUT 14  ','CZ      ','OUT 15  ','JMP     ','OUT 16  ',
   'CALL    ','OUT 17  ','JM      ','OUT 18  ','CM      ',
   'OUT 19  ','JMP     ','OUT 1A  ','CALL    ','OUT 1B  ',
   'JPE     ','OUT 1C  ','CPE     ','OUT 1D  ','JMP     ',
   'OUT 1E  ','CALL    ','OUT 1F  ','ADD A   ','ADD B   ',
   'ADD C   ','ADD D   ','ADD E   ','ADD H   ','ADD L   ',
   'ADD M   ','ADC A   ','ADC B   ','ADC C   ','ADC D   ',
'ADC E   ','ADC H   ','ADC L   ','ADC M   ','SUB A   ',
'SUB B   ','SUB C   ','SUB D   ','SUB E   ','SUB H   ',
'SUB L   ','SUB M   ','SBB A   ','SBB B   ','SBB C   ',
'SBB D   ','SBB E   ','SBB H   ','SBB L   ','SBB M   ',
'ANA A   ','ANA B   ','ANA C   ','ANA D   ','ANA E   ',
'ANA H   ','ANA L   ','ANA M   ','XRA A   ','XRA B   ',
'XRA C   ','XRA D   ','XRA E   ','XRA H   ','XRA L   ',
'XRA M   ','ORA A   ','ORA B   ','ORA C   ','ORA D   ',
'ORA E   ','ORA H   ','ORA L   ','ORA M   ','CMP A   ',
'CMP B   ','CMP C   ','CMP D   ','CMP E   ','CMP H   ',
'CMP L   ','CMP M   ','NOP     ','MOV A,B ','MOV A,C ',
'MOV A,D ','MOV A,E ','MOV A,H ','MOV A,L ','MOV A,M ',
'MOV B,A ','MOV B,B ','MOV B,C ','MOV B,D ','MOV B,E ',
'MOV B,H ','MOV B,L ','MOV B,M ','MOV C,A ','MOV C,B ',
'MOV C,C ','MOV C,D ','MOV C,E ','MOV C,H ','MOV C,L ',
'MOV C,M ','MOV D,A ','MOV D,B ','MOV D,C ','MOV D,D ',
'MOV D,E ','MOV D,H ','MOV D,L ','MOV D,M ','MOV E,A ',
'MOV E,B ','MOV E,C ','MOV E,D ','MOV E,E ','MOV E,H ',
'MOV E,L ','MOV E,M ','MOV H,A ','MOV H,B ','MOV H,C ',
'MOV H,D ','MOV H,E ','MOV H,H ','MOV H,L ','MOV H,M ',
'MOV L,A ','MOV L,B ','MOV L,C ','MOV L,D ','MOV L,E ',
'MOV L,H ','MOV L,L ','MOV L,M ','MOV M,A ','MOV M,B ',
'MOV M,C ','MOV,M,D ','MOV M,E ','MOV M,H ','MOV M,L ',
'HLT     ',
'NOP     ','LXI,B   ','STAX,B  ','INX,B   ','INR,B   ',                 81
'DCR,B   ','MVI,B   ','RLC     ','***ERROR','DAD,B   ',                 82
'LDAX,B  ','DCX,B   ','INR,C   ','DCR,C   ','MVI,C   ',                 83
'RRC     ','***ERROR','LXI,D   ','STAX,D  ','INX,D   ',                 84
'INR,D   ','DCR,D   ','MVI,D   ','RAL     ','***ERROR',                 85
'DAD,D   ','LDAX,D  ','DCX,D   ','INR,E   ','DCR,E   ',                 86
'MVI,E   ','RAR     ','***ERROR','LXI,H   ','SHLD    ',                 87
'INX,H   ','INR,H   ','DCR,H   ','MVI,H   ','DAA     ',                 88
'***ERROR','DAD,H   ','LHLD    ','DCX,H   ','INR,L   ',                 89
'DCR,L   ','MVI,L   ','CMA     ','***ERROR','LXI,SP  ',                 90
'STA     ','INX,SP  ','INR,M   ','DCR,M   ','MVI,M   ',                 91
```

```
'STC     ','***ERROR','DAD,SP  ','LDA     ','DCX,SP  ',                    92
'INR,A   ','DCR,A   ','MVI,A   ','CMC     ','MOV B,B ',                    93
'MOV B,C ','MOV B,D ','MOV B,E ','MOV B,H ','MOV B,L ',
'MOV B,M ','MOV B,A ','MOV C,B ','MOV C,C ','MOV C,D ',
'MOV C,E ','MOV C,H ','MOV C,L ','MOV C,M ','MOV C,A ',
'MOV D,B ','MOV D,C ','MOV D,D ','MOV D,E ','MOV D,H ',
'MOV D,L ','MOV D,M ','MOV D,A ','MOV E,B ','MOV E,C ',
'MOV E,D ','MOV E,E ','MOV E,H ','MOV E,L ','MOV E,M ',
'MOV E,A ','MOV H,B ','MOV H,C ','MOV H,D ','MOV H,E ',
'MOV H,H ','MOV H,L ','MOV H,M ','MOV H,A ','MOV L,B ',
'MOV L,C ','MOV L,D ','MOV L,E ','MOV L,H ','MOV L,L ',
'MOV L,M ','MOV L,A ','MOV M,B ','MOV M,C ','MOV M,D ',
'MOV M,E ','MOV M,H ','MOV M,L ','HALT    ','MOV M,A ',
'MOV A,B ','MOV A,C ','MOV A,D ','MOV A,E ','MOV A,H ',
'MOV A,L ','MOV A,M ','MOV A,A ','ADD,B   ','ADD,C   ',
'ADD,D   ','ADD,E   ','ADD,H   ','ADD,L   ','ADD,M   ',
'ADD,A   ','ADC,B   ','ADC,C   ','ADC,D   ','ADC,E   ',
'ADC,H   ','ADC,L   ','ADC,M   ','ADC,A   ','SUB,B   ',
'SUB,C   ','SUB,D   ','SUB,E   ','SUB,H   ','SUB,L   ',
'SUB,M   ','SUB,A   ','SBB,B   ','SBB,C   ','SBB,D   ',
'SBB,E   ','SBB,H   ','SBB,L   ','SBB,M   ','SBB,A   ',
'ANA,B   ','ANA,C   ','ANA,D   ','ANA,E   ','ANA,H   ',
'ANA,L   ','ANA,M   ','ANA,A   ','XRA,B   ','XRA,C   ',
'XRA,D   ','XRA,E   ','XRA,H   ','XRA,L   ','XRA,M   ',
'XRA,A   ','ORA,B   ','ORA,C   ','ORA,D   ','ORA,E   ',
'ORA,H   ','ORA,L   ','ORA,M   ','ORA,A   ','CMP,B   ',
'CMP,C   ','CMP,D   ','CMP,E   ','CMP,H   ','CMP,L   ',
'CMP,M   ','CMP,A   ','RNZ     ','POP,B   ','JNZ     ',
'JMP     ','CNZ     ','PUSH,B  ','ADI     ','RST,0   ',
'RZ      ','RET     ','JZ      ','***ERROR','CZ      ',
'CALL    ','ACI     ','RST,1   ','RNC     ','POP,D   ',
'JNC     ','OUT     ','CNC     ','PUSH,D  ','SUI     ',
'RST,2   ','RC      ','***ERROR','JC      ','IN      ',
'CC      ','***ERROR','SBI     ','RST,3   ','RPO     ',
'POP,H   ','JPO     ','XTHL    ','CPO     ','PUSH,H  ',
'ANI      ','RST,4    ','RPE      ','PCHL     ','JPE      ',
'XCHG     ','CPE      ','***ERROR','XRI      ','RST,5    ',
'RP       ','POP,PSW  ','JP       ','DI       ','CP       ',
'PUSH,PSW','ORI      ','RST,6    ','RM       ','SPHL     ',
'JM       ','EI       ','CM       ','***ERROR','CPI      ',
'RST,7    ',
'***ERROR','***ERROR','NOP      ','***ERROR','***ERROR',
'***ERROR','TAP      ','TPA      ','INX      ','DEX      ',
'CLV      ','SEV      ','CLC      ','SEC      ','CLI      ',
'SEI      ','SBA      ','CBA      ','***ERROR','***ERROR',
'***ERROR','***ERROR','TAB      ','TBA      ','***ERROR',
'DAA      ','***ERROR','ABA      ','***ERROR','***ERROR',
'***ERROR','***ERROR','BRA     R','***ERROR','BHI     R',
'BLS     R','BCC     R','BCS     R','BNE     R','BEQ     R',
'BVC     R','BVS     R','BPL     R','BMI     R','BGE     R',
'BLT     R','BGT     R','BLE     R','TSX      ','INS      ',
'PULA     ','PULB     ','DES      ','TXS      ','PSHA     ',
'PSHB     ','***ERROR','RTS      ','***ERROR','RTI      ',
'***ERROR','***ERROR','WAI      ','SWI      ','NEGA     ',
'***ERROR','***ERROR','COMA     ','LSRA     ','***ERROR',
'RORA     ','ASRA     ','ASLA     ','ROLA     ','DECA     ',
'***ERROR','INCA     ','TSTA     ','***ERROR','CLRA     ',
'NEGB     ','***ERROR','***ERROR','COMB     ','LSRB     ',
'***ERROR','RORB     ','ASRB     ','ASLB     ','ROLB     ',
'DECB     ','***ERROR','INCB     ','TSTB     ','***ERROR',
'CLRB     ','NEG    IX','***ERROR','***ERROR','COM    IX',
'LSR    IX','***ERROR','ROR    IX','ASR    IX','ASL    IX',
'ROL    IX','DEC    IX','***ERROR','INC    IX','TST    IX',
'JMP    IX','CLR    IX','NEG    EX','***ERROR','***ERROR',
'COM    EX','LSR    EX','***ERROR','ROR    EX','ASR    EX',
'ASL    EX','ROL    EX','DEC    EX','***ERROR','INC    EX',
'TST    EX','JMP    EX','CLR    EX','SUBA   IM','CMPA   IM',
'SBCA   IM','***ERROR','ANDA   IM','BITA   IM','LDAA   IM',
'***ERROR','EORA   IM','ADCA   IM','ORAA   IM','ADDA   IM',
'CPX    IM','BSR     R','LDS    IM','***ERROR','SUBA    D',
'CMPA   D','SBCA   D','***ERROR','ANDA   D','BITA   D',
'LDAA   D','STAA   D','EORA   D','ADCA   D','ORAA   D',
'ADDA   D','CPX    D','***ERROR','LDS    D','STS    D',
'SUBA  IX','CMPA  IX','SBCA  IX','***ERROR','ANDA  IX',
'BITA  IX','LDAA  IX','STAA  IX','EORA  IX','ADCA  IX',
'ORAA  IX','ADDA  IX','CPX   IX','JSR   IX','LDS   IX',
'STS   IX','SUBA  EX','CMPA  EX','SBCA  EX','***ERROR',
'ANDA  EX','BITA  EX','LDAA  EX','STAA  EX','EORA  EX',
'ADCA  EX','ORAA  EX','ADDA  EX','CPX   EX','JSR   EX',
'LDS   EX','STS   EX','SUBB  IM','CMPB  IM','SBCB  IM',
'***ERROR','ANDB  IM','BITB  IM','LDAB  IM','***ERROR',
'EORB  IM','ADCB  IM','ORAB  IM','ADDB  IM','***ERROR',
'***ERROR','LDX   IM','***ERROR','SUBB   D','CMPB   D',
'SBCB   D','***ERROR','ANDB   D','BITB   D','LDAB   D',
'STAB   D','EORB   D','ADCB   D','ORAB   D','ADDB   D',
'***ERROR','***ERROR','LDX    D','STX    D','SUBB  IX',
'CMPB  IX','SBCB  IX','***ERROR','ANDB  IX','BITB  IX',
'LDAB  IX','STAB  IX','EORB  IX','ADCB  IX','ORAB  IX',
'ADDB  IX','***ERROR','***ERROR','LDX   IX','STX   IX',
'SUBB  EX','CMPB  EX','SBCB  EX','***ERROR','ANDB  EX',
'BITB  EX','LDAB  EX','STAB  EX','EORB  EX','ADCB  EX',
'ORAB  EX','ADDB  EX','***ERROR','***ERROR','LDX   EX',
'STX   EX',
'LR A,KU ','LR A,KL ','LR A,QU ','LR A,QL ','LR KU,A ',
'LR KL,A ','LR QU,A ','LR QL,A ','LR K,P  ','LR P,K  ',
'LR A,IS ','LR IS,A ','PK      ','LR P,Q  ','LR Q,DC ',
'LR DC,Q ','LR DC,H ','LR K,P  ','SR 1    ','SL 1    ',
'SR 4    ','SL 4    ','LM      ','ST      ','COM     ',
'LNK     ','DI      ','EI      ','POP     ','LR W,J  ',
'LR J,W  ','INC     ','LI      ','NI      ','OI      ',
'XI      ','AI      ','CI      ','IN      ','OUT     ',
'PI      ','JMP     ','DCI     ','NOP     ','XDC     ',
'***ERROR','***ERROR','***ERROR','DS R0   ','DS R1   ',
'DS R2   ','DS R3   ','DS R4   ','DS R5   ','DS R6   ',
'DS R7   ','DS R8   ','DS R9   ','DS RA   ','DS RB   ',
'DS ISAR ','DS ISAR+','DS ISAR-','NOP      ','LR A,R0 ',
'LR A,R1 ','LR A,R2 ','LR A,R3 ','LR A,R4 ','LR A,R5 ',
'LR A,R6 ','LR A,R7 ','LR A,R8 ','LR A,R9 ','LR A,RA ',
'LR A,RB ','LRA,ISAR','LRAISAR+','LRAISAR-','NOP      ',
'LR R0,A ','LR R1,A ','LR R2,A ','LR R3,A ','LR R4,A ',
'LR R5,A ','LR R6,A ','LR R7,A ','LR R8,A ','LR R9,A ',
'LR RA,A ','LR RB,A ','LRISAR,A','LRISAR+A','LRISAR-A',
'NOP     ','LISU 0  ','LISU 1  ','LISU 2  ','LISU 3  ',
'LISU 4  ','LISU 5  ','LISU 6  ','LISU 7  ','LISL 0  ',
'LISL 1  ','LISL 2  ','LISL 3  ','LISL 4  ','LISL 5  ',
'LISL 6  ','LISL 7  ','CLR     ','LIS 01  ','LIS 02  ',
'LIS 03  ','LIS 04  ','LIS 05  ','LIS 06  ','LIS 07  ',
'LIS 08  ','LIS 09  ','LIS 0A  ','LIS 0B  ','LIS 0C  ',
'LIS 0D  ','LIS 0E  ','LIS 0F  ','***ERROR','BP      ',
'BC      ','BT      ','BZ      ','BT      ','BT      ',
'BT      ','AM      ','AMD     ','NM      ','OM      ',
'XM      ','CM      ','ADC     ','BP7     ','BR      ',
'BM      ','BNC     ','BF      ','BNZ     ','BF      ',
'BF      ','BF      ','BNO     ','BF      ','BF      ',
'BF      ','BF      ','BF      ','BF      ','BF      ',
'INS   P0','INS   P1 ','INS   P2 ','INS   P3 ','INS   P4 ',
'INS  P5 ','INS   P6 ','INS   P7 ','INS   P8 ','INS   P9 ',
'INS  PA ','INS   PB ','INS   PC ','INS   PD ','INS   PE ',
'INS  PF ','OUTS   P0','OUTS   P1','OUTS   P2','OUTS   P3',
'OUTS   P4','OUTS   P5','OUTS   P6','OUTS   P7','OUTS   P8',
'OUTS   P9','OUTS   PA','OUTS   PB','OUTS   PC','OUTS   PD',
'OUTS   PE','OUTS   PF','AS R0    ','AS R1    ','AS R2    ',
'AS R3   ','AS R4    ','AS R5    ','AS R6    ','AS R7    ',
'AS R8   ','AS R9    ','AS RA    ','AS RB    ','AS ISAR ',
'AS ISAR+','AS ISAR-','NOP      ','ASD R0   ','ASD R1   ',
'ASD R2  ','ASD R3   ','ASD R4   ','ASD R5   ','ASD R6   ',
'ASD R7  ','ASD R8   ','ASD R9   ','ASD RA   ','ASD RB   ',
'ASD ISAR','ASDISAR+','ASDISAR-','NOP      ','XS R0    ',
'XS R1   ','XS R2    ','XS R3    ','XS R4    ','XS R5    ',
'XS R6   ','XS R7    ','XS R8    ','XS R9    ','XS RA    ',
          'XS RB    ','XS ISAR ','XS ISAR+','XS ISAR-','NOP      ',
          'NS R0    ','NS R1    ','NS R2    ','NS R3    ','NS R4    ',
          'NS R5    ','NS R6    ','NS R7    ','NS R8    ','NS R9    ',
          'NS RA    ','NS RB    ','NS ISAR ','NS ISAR+','NS ISAR-',
          'NOP      ')    ;
        DCL(PORTCH,OUTCH) CHAR(4);
        DCL CDUMY CHAR(4);
        DCL CSD CHARACTER(3) DEF CDUMY POSITION(2);
        DCL CDFT CHAR (2)  DEF CDUMY POSITION(1);
        DCL CDBK CHAR(2) INITIAL('  ');
        DCL TERMIN CHAR(1);
        DCL PARTAB BIT(256) EXTERNAL INITIAL('1001011001101001011010011001      P1
011001101001100101101001011001101001011010011001011010010110011010011 00      P2
10110011010010110100110010110011010011001011010010110011010011001011001      P3
10100101101001100101101001011001101001011010011001011001101001100101101      P4
001011001101001'B);                                                          P5
        DCL TST BIT(8);
        DCL TIMESW BIT(1) INITIAL('0'B);
        DCL (CY,Z,S,P,AC,V) BIT(1) INITIAL('0'B);
        DCL I682 BIT(1) INITIAL('1'B);
        DCL(ICB2,CY2,Z2,S2,P2,AC2,V2) BIT(1) INITIAL('0'B);
        DCL ICB BIT(1) INITIAL('0'B);
        DCL I68 BIT(1) INITIAL('1'B);
        DCL INTE BIT(1) INITIAL('1'B);
        DCL INTE2 BIT(1) INITIAL('1'B);
        DCL (CYT,ACT) BIT(1);
        DCL FENSW BIT(1) EXTERNAL INITIAL('0'B);
        DCL FENSW2 BIT(1) INITIAL('0'B);
        DCL IOSW BIT(1) INITIAL('0'B);
        DCL PTSW BIT(1);
        DCL IOOVSW BIT(1) EXTERNAL INITIAL('0'B);
        DCL MEMSW BIT(1) EXTERNAL;
        DCL STSW BIT(1);
        DCL DPSW BIT(1) INITIAL('0'B);
        DCL(ROMSW,SZSW) BIT(1) INITIAL('0'B) EXTERNAL;
```

```
  DCL FIFOSW BIT(1) INITIAL('0'B) EXTERNAL;
  DCL FIFOS2 BIT(1) INITIAL('0'B);
  DCL DELTSW BIT(1) INITIAL('0'B);
  DCL SCRSW BIT(1) EXTERNAL INITIAL('0'B);
  DCL BRSW BIT(1) INITIAL('0'B);
  DCL INSW BIT(1) EXTERNAL INITIAL('0'B);
  DCL OUTSW BIT(1) EXTERNAL INITIAL('0'B);
  DCL INHIB BIT(1) INITIAL('0'B) EXTERNAL;
  DCL MESW BIT(1) INITIAL('0'B);
  DCL NULSW BIT(1) EXTERNAL;
  DCL FISW BIT(1) EXTERNAL INITIAL('0'B);
  DCL MLSW2 BIT(1) INITIAL('0'B);
  DCL MLSW BIT(1) INITIAL('0'B);
  DCL PINSW BIT(1) INITIAL('0'B);
  DCL POUTSW BIT(1) INITIAL('0'B);
  DCL STOPSW BIT(1) INITIAL('0'B);
  DCL DB BIT(1); /* DUMMY*/
  DCL (DB2,DB3) BIT(1);
ON ENDFILE BEGIN; PUT LIST('** NO MORE *OPTIONS CARDS FOUND'
  ,' TERMINATES')SKIP;     STOP; END;
PUT LIST(' PROGRAM MPBSIM MULTIPLE SIMULATOR EXECUTES AS FOLLOWS:')
  SKIP;
PNSSC:       MESW='0'B;
NSSC: GET EDIT(CARD)(A(80));
IF(SUBSTR(CARD,1,1)='*') THEN PUT LIST('   ')PAGE;
IF(SUBSTR(CARD,1,1)¬='*') THEN DO;
  IF(¬MESW) THEN PUT LIST('** CARDS PRECEEDING  *OPTIONS CARD'
    ,'  IGNORED')SKIP; PUT EDIT('         ',CARD)(A(9),A(80))SKIP;
  MESW='1'B; GOTO NSSC;
  END;
PUT LIST(CARD)SKIP;
IF(INDEX(CARD,'SWAP')¬=0) THEN DO;
IF(INDEX(CARD,'MEM')¬=0) THEN MSWS='1'B;ELSE MSWS='0'B;
IF(INDEX(CARD,'PORT')¬=0) THEN DO;
  MAXI=0; DO I = 1 TO 32;
```

```
    K=PORTLT(I); PORTLT(I)=PORTLT2(I); PORTLT2(I)=K;
    K=PSTART(I); PSTART(I)=PSTART2(I); PSTART2(I)=K;
    K=PORTID(I); PORTID(I)=PORTID2(I); PORTID2(I)=K;
    K=PEND(I); PEND(I)=PEND2(I); PEND2(I)=K;
    IF(PEND(I)>MAXI) THEN MAXI=PEND(I);
    IF(PEND2(I)>MAXI) THEN MAXI=PEND2(I);
  END;
  DO I=1 TO MAXI;
    TEMPA=IDATA(I); IDATA(I)=IDATA2(I); IDATA2(I)=TEMPA;
  END;
END; /*PORT OPTION OF A SWAP OPTION */
DB=FIFOSW; FIFOSW=FIFOS2; FIFOS2=DB;
K=PCN; PCN=PCN2; PCN2=K;
  K=PC; PC=PC2; PC2=K;
  K=SP; SP=SP2; SP2=K;
  TEMPB=SP08; SP08=SP082; SP082=TEMPB;
TEMPB=MACH; MACH=MACH2; MACH2=TEMPB;
  K=INDX; INDX=INDX2; INDX2=K;
  TEMPA=A; A=A2; A2=TEMPA;
  TEMPA=B; B=B2; B2=TEMPA;
  TEMPA=C; C=C2; C2=TEMPA;
  TEMPA=D; D=D2; D2=TEMPA;
  TEMPA=E; E=E2; E2=TEMPA;
  TEMPA=H; H=H2; H2=TEMPA;
  TEMPA=L; L=L2; L2=TEMPA;
  ACT=I68; I68=I682;I682=ACT;
  ACT=ICB; ICB=ICB2;ICB2=ACT;
  ACT=INTE; INTE=INTE2; INTE2=ACT;
  ACT=CY; CY=CY2; CY2=ACT;
  ACT=AC; AC=AC2; AC2=ACT;
  ACT=Z; Z=Z2;Z2=ACT;
  ACT=S; S=S2;S2=ACT;
  ACT=P; P=P2;P2=ACT;
  ACT=V; V=V2;V2=ACT;
  TEMPB=ISAR; ISAR=ISAR2; ISAR2=TEMPB;
  K=DC; DC=DC2; DC2=K;
  K=DC1; DC1=DC12; DC12=K;
  K=ITIME; ITIME=ITIME2; ITIME2=K;
  DO I=0 TO 63; TEMPA=SCF8(I); SCF8(I)=SCF82(I); SCF82(I)=TEMPA;END;
  DO I=0 TO 7;
    K=STK08(I);
    STK08(I)=STK082(I);  STK082(I)=K; END;
  K=FI1; FI1=FI12; FI12=K;
  K=FO1; FO1=FO12; FO12=K;
  K=FI2; FI2=FI22; FI22=K;
  K=FO2; FO2=FO22; FO22=K;
DB=MLSW; MLSW=MLSW2; MLSW2=DB;
IF(MSWS) THEN DO;
DB=FENSW; FENSW=FENSW2; FENSW2=DB;
K=FENCE; FENCE=FENCE2; FENCE2=K;
  K=MEMSIZ; MEMSIZ=MEMSIZ2; MEMSIZ2=K;
  K=ROMST; ROMST=ROMST2; ROMST2=K;
  K=ROMEND; ROMEND=ROMEND2; ROMEND2=K;
    MSIZ=MEMSIZ; IF(MEMSIZ2>MEMSIZ) THEN MSIZ=MEMSIZ2; MSIZ=MSIZ+1;
  DO I= 1 TO 4; DO J=1 TO 16384;
  IF(16384*(I-1)+J>MSIZ) THEN GOTO ESCAPE;
    TEMPA=MEM(I,J); MEM(I,J)=MEM2(I,J); MEM2(I,J)=TEMPA    ;
END; END;
ESCAPE: END;
SWCT=SWCT+1; IF(SWCT>2)THEN SWCT=1;
PUT LIST('**NOW IN SPACE ',SWCT)SKIP;
GOTO PNSSC;
END;
IF(INDEX(CARD,'LIST')¬=0) THEN DO;
IF(INDEX(CARD,'SCR')¬=0) THEN SCRSW='1'B; ELSE SCRSW='0'B;
  COL=80; CALL HEXIN(K,TERMIN     ,COL); IF(TERMIN=';')THEN COL=80;
  CALL HEXIN(M,TERMIN     ,COL);      COL=80;
  CALL DUMP(K,M); GOTO PNSSC; END;
IF(INDEX(CARD,'MEM')¬=0) THEN DO; MEMSW='1'B; /*MEMLOAD OPTION*/
IF(INDEX(CARD,'DUMP')¬=0) THEN DPSW='1'B; ELSE DPSW='0'B;
```

```
   IF(INDEX(CARD,'SIZE')¬=0) THEN SZSW='1'B;  ELSE SZSW='0'B;
   IF(INDEX(CARD,'ROM')¬=0) THEN ROMSW='1'B;  ELSE ROMSW='0'B;
   IF(INDEX(CARD,'FENCE')¬=0) THEN FENSW='1'B; ELSE FENSW='0'B;
   COL=80;
   IF(SZSW) THEN DO;
     CALL HEXIN(MEMSIZ,TERMIN     ,COL); MEMSIZ=MEMSIZ-1; END;
   IF(ROMSW) THEN DO;
     CALL HEXIN(ROMST,TERMIN     ,COL);
     CALL HEXIN(ROMEND,TERMIN     ,COL); END;
   IF(FENSW) THEN CALL HEXIN(FENCE,TERMIN     ,COL);
   COL=80;
   LOADP=0;
MLOAD: CALL HEXIN(VALUE,TERMIN     ,COL);
   IF(TERMIN='.') THEN GOTO DPQ;
         IF(TERMIN=':') THEN DO;
              LOADP=VALUE;
              GOTO MLOAD;
              END;
         IF(LOADP>MEMSIZ) THEN DO;
              PUT LIST ('**** TRYS TO LOAD BEYOND END OF MEMORY')
              SKIP; GOTO PNSSC; END;
         TEMPA=VALUE; CALL STORE(TEMPA,LOADP);
         LOADP=LOADP+1;
         IF (TERMIN=',') THEN GOTO MLOAD;
         IF(TERMIN=';') THEN DO; COL=80; GOTO MLOAD; END;
   DPQ:MEMSW='0'B;
   K=0;
        IF(DPSW) THEN CALL DUMP(K,MEMSIZ);
   MLSW='1'B;
   GOTO PNSSC;
 END;         /*END OF MLOAD OPTION*/
   IF(INDEX(CARD,'PORT')¬=0) THEN DO;
     IF(INDEX(CARD,'IN')¬=0) THEN PINSW='1'B; ELSE PINSW='0'B;
     IF(INDEX(CARD,'OUT')¬=0) THEN POUTSW='1'B; ELSE POUTSW='0'B;
   IF(¬PINSW&¬POUTSW) THEN DO;
```

```
     PUT LIST('**** MUST DEFINE WHETHER IN OR OUT PORTS',
       'ON *PORT CARD') SKIP; STOP; END;
   IF(PINSW)THEN DO;
      DO I=1 TO 32; PORTID=-1; END; /* CALL OF *PORT REDEFINES ALL*/
   ILISP=0; ILPOIN=1;
   COL=80;
ILOAD: CALL HEXIN(VALUE,TERMIN     ,COL);
   IF(TERMIN='.') THEN GOTO LOUT;
   IF(TERMIN=':') THEN DO;
     ILISP=ILISP+1;
     IF(ILISP>32) THEN DO; PUT LIST('**** TOO MANY PORTS')SKIP;
     GOTO PNSSC; END;
     PORTID(ILISP)=VALUE;
     PSTART(ILISP)=ILPOIN;
     PEND(ILISP-1)=ILPOIN-1;
     GOTO ILOAD;
     END;
   IF(TERMIN=','|TERMIN=';') THEN DO;
     IF(ILPOIN>4096 )THEN BUFO: DO;
       PUT LIST('**** INPUT OVERFLOWS INTERNAL BUFFER') SKIP;
     GOTO PNSSC; END;
     IF(TERMIN=';') THEN COL=80;
     IDATA(ILPOIN)=VALUE;
     ILPOIN=ILPOIN+1;
     GOTO ILOAD;
     END;
   LOUT: PEND(ILISP)=ILPOIN-1;
   END;
   IF(POUTSW) THEN DO;
       DO I=1 TO 32; PORTLT(I)=-1; END; /*REDEFINES OUTPORTS*/
   COL=80;
     I=0;
     AGNS: CALL HEXIN(K,TERMIN     ,COL);
   IF(TERMIN='.') THEN DO;
       COL=80; GOTO STARTS; END;
```

```
  IF(TERMIN=',') THEN GOTO AGNS;
  IF(TERMIN=';') THEN DO;
      COL=80; GOTO AGNS; END;
    I=I+1;
    IF(I>32&TERMIN=':') THEN DO;
      PUT LIST('** TOO MANY OUTPUT PORTS DEFINED   ONLY 32 RECOGNIZED'
      )SKIP; GOTO STARTS; END    ;
    IF(TERMIN=':') THEN DO;PORTLT(I)=K; GOTO AGNS; END;
  STARTS: K=0;
  END;
  GOTO PNSSC;
END;  /*END OF PORT DEFINITION OPTION*/
  IF(INDEX(CARD,'FIFO')¬=0) THEN DO;
    IF(INDEX(CARD,'CLEAR1')¬=0) THEN DO; FIP(1)=0;FOP(1)=0; END;
    IF(INDEX(CARD,'CLEAR2')¬=0) THEN DO; FIP(2)=0;FOP(2)=0; END;
    COL=80;
    CALL HEXIN(FI1,TERMIN     ,COL);
    CALL HEXIN(FO1,TERMIN     ,COL);
    CALL HEXIN(FI2,TERMIN     ,COL);
    CALL HEXIN(FO2,TERMIN     ,COL);
    COL=80;
    FIFOSW='1'B;
    GOTO PNSSC;
  END;
  IF(INDEX(CARD,'INTER')¬=0) THEN DO;
    IF(INDEX(CARD,'BRA')¬=0|INDEX(CARD,'JUM')¬=0)THEN BRSW='1'B; ELSE
        BRSW='0'B;
    IF(INDEX(CARD,'I/O')¬=0) THEN IOSW='1'B; ELSE IOSW='0'B;
    IF(INDEX(CARD,'SCR')¬=0) THEN SCRSW='1'B; ELSE SCRSW='0'B;
    IF(INDEX(CARD,'8008')¬=0) THEN MACH=1;
    IF(INDEX(CARD,'8080')¬=0) THEN MACH=2;
    IF(INDEX(CARD,'6800')¬=0) THEN MACH=3;
    IF(INDEX(CARD,'F8')¬=0) THEN MACH=4;
    IF(INDEX(CARD,'DUMP')¬=0) THEN DPSW='1'B;ELSE DPSW='0'B;
    IF(INDEX(CARD,'STOP')¬=0) THEN STOPSW='1'B; ELSE STOPSW='0'B;
IF(INDEX(CARD,'TIME')¬=0) THEN TIMESW='1'B; ELSE TIMESW='0'B;
IF(INDEX(CARD,'RESET')¬=0) THEN ITIME=0;
COL=80;
IF(STOPSW) THEN CALL HEXIN(STOP,TERMIN,COL);
IF(TIMESW) THEN CALL DECIN(TLIM,TERMIN,COL);
IF(MACH=0) THEN DO; PUT LIST('**** NO PROCESSOR DEFINED');
  STOP; END;
IF(MACH=1) THEN DO; COL=80;
CALL HEXIN(BGD,TERMIN,COL); INS=BGD;
  CALL HEADINGS; LINE=4; INHIB='0'B; GOTO EX(1,INS);
END;
IF(MACH=2) THEN DO;
  IF(INTE='0'B) THEN DO; PUT LIST(' **8080 INTERUPT NOT ENABLED');
  GOTO PNSSC; END;
CALL HEXIN(BGD,TERMIN,COL); INS=BGD; CALL HEADINGS; LINE=4;
INHIB='0'B; GOTO EX(2,INS);
END;
IF(MACH=3) THEN DO;
  IF(I68='0'B) THEN DO; PUT LIST(' **6800 INTERUPT NOT ENABLED');
  GOTO PNSSC; END;
IF(INS¬=62) THEN DO;
C=MOD(PCN,256); CALL PUSH(0,1,C,SP);
C=(PCN-C)/256; CALL PUSH(0,1,C,SP);
C=MOD(INDX,256); CALL PUSH(0,1,C,SP);
C=(INDX-C)/256; CALL PUSH(0,1,C,SP);
CALL PUSH(0,1,A,SP); CALL PUSH(0,1,B,SP);
C='11'B||AC||I68||S||Z||V||CY; CALL PUSH(0,1,C,SP);
ITIME=ITIME+8;
END;
ITIME=ITIME+4; CALL FETCH(SD,65528);
PCN=SD*256; CALL FETCH(SD,65529); PCN=PCN+SD;
CALL HEADINGS; LINE=4; INHIB='0'B;
GOTO NEWINS;
END;
IF(MACH=4) THEN DO;
```

```
  IF(ICB='0'B) THEN DO; PUT LIST(' **F8 INTERUPT NOT ENABLED');
   GOTO PNSSC; END;
  IF(INS=12|INS>26&INS<30|INS>38&INS<41|INS>175&INS<192) THEN DO;
    PUT LIST(' ** LAST F8 INSTRUCTION EXECUTED WAS PRIVILEGED');
    END;
  ITIME=ITIME+16; PC1=PCN; COL=80; CALL HEXIN(PCN,TERMIN,COL);
  CALL HEADINGS; LINE=4; INHIB='0'B; GOTO NEWINS;
  END;
END;
IF(INDEX(CARD,'PROC')=0)THEN GOTO PNSSC;
IF(¬MLSW) THEN DO;
  PUT LIST('**** MEMORY LOAD MUST PRECEED PROCESSOR CALL') SKIP;
  STOP; END;
IF(INDEX(CARD,'BRA')¬=0|INDEX(CARD,'JUM')¬=0)THEN BRSW='1'B; ELSE
    BRSW='0'B;
IF(INDEX(CARD,'I/O')¬=0) THEN IOSW='1'B; ELSE IOSW='0'B;
IF(INDEX(CARD,'SCR')¬=0) THEN SCRSW='1'B; ELSE SCRSW='0'B;
IF(INDEX(CARD,'8008')¬=0) THEN DO; MACH=1; INHIB='0'B; END;
IF(INDEX(CARD,'8080')¬=0) THEN DO; MACH=2; INHIB='0'B; END;
IF(INDEX(CARD,'6800')¬=0) THEN DO; MACH=3; INHIB='0'B; END;
IF(INDEX(CARD,'F8')¬=0) THEN DO; MACH=4; INHIB='0'B; END;
IF(INDEX(CARD,'DUMP')¬=0) THEN DPSW='1'B;  ELSE DPSW='0'B;
IF(INDEX(CARD,'START')¬=0) THEN STSW='1'B; ELSE STSW='0'B;
IF(INDEX(CARD,'STOP')¬=0) THEN STOPSW='1'B; ELSE STOPSW='0'B;
IF(INDEX(CARD,'TIME')¬=0) THEN TIMESW='1'B; ELSE TIMESW='0'B;
IF(INDEX(CARD,'RESET')¬=0) THEN ITIME=0;
COL=80;
IF(MACH=0) THEN DO;
  PUT LIST('**** NO VALID PROCESSOR REQUEST')SKIP; STOP; END;
IF(STSW) THEN CALL HEXIN(PCN,TERMIN    ,COL); STSW='0'B;
IF(STOPSW)THEN CALL HEXIN(STOP,TERMIN    ,COL);
IF(TIMESW) THEN CALL DECIN(TLIM,TERMIN,COL);
CALL HEADINGS;
LINE=0;
  COL=80;
```

```
NEWINS: IF(INHIB) THEN DO;
     PUT LIST('**END OF SIMULATION') SKIP(2);
   K=0;
     SCRSW='1'B;
     IF(DPSW) THEN CALL DUMP(K,MEMSIZ); PUT LIST(' _') SKIP(3);
     GOTO PNSSC; END;
   IF(STOPSW) THEN IF(PCN=STOP) THEN DO;
     CALL TOHEX(PCN,CDUMY);
     PUT LIST('**STOPS BEFORE EXECUTION AT PC=',CDUMY,
       'AS REQUESTED')SKIP;
     INHIB='1'B; GOTO NEWINS; END  ;
   IF(TIMESW) THEN IF(ITIME>=TLIM) THEN DO;
     PUT LIST(' **STOPS AT CLOCK=',ITIME,'  THAT IS >= TO TIME LIMIT='
       ,TLIM,' AS REQUESTED') SKIP; INHIB='1'B; GOTO NEWINS; END;
   PC=PCN;
   CALL FETCH(TEMPA,PC);
   INS=TEMPA;
   PCN=PC+DELPC(MACH,INS);
   IF(MACH=3) THEN DO;
   /* RELATIVE  */
     IF(INS>31&INS<48|INS=141) THEN DO;
       CALL FETCH(TEMPA,PC+1);
       IF(TEMPA>127) THEN TEMPA=TEMPA-256;
       ADX=PC+2+TEMPA; END;
   /* DIRECT  */
     IF(INS>143&INS<160|INS>207&INS<224) THEN DO;
       CALL FETCH(TEMPA,PC+1);
       ADX=TEMPA; END;
   /* INDEXED  */
     IF(INS>95&INS<112|INS>159&INS<176|INS>223&INS<240) THEN DO;
       CALL FETCH(TEMPA,PC+1);
       ADX=INDX+TEMPA; END;
   /* EXTENDED */
     IF(INS>111&INS<128|INS>175&INS<192|INS>239) THEN DO;
       CALL FETCH(TEMPA,PC+1);
       ADX=TEMPA*256;
       CALL FETCH(TEMPA,PC+2);
       ADX=ADX+TEMPA;  END;
   /* IMMEDIATE  */
     IF(INS>127&INS<144&INS¬=141|INS>191&INS<208) THEN DO;
       ADX=PC+1;  END;
   END;
   GOTO EX(MACH,INS);
/* INSTRUCTION ERROR FOR ALL MACHS   */
   EX(1,34): EX(1,42): EX(1,50): EX(1,56): EX(1,57):EX(1,58):
   EX(2,8): EX(2,16): EX(2,24): EX(2,32): EX(2,40): EX(2,48): EX(2,56):
   EX(2,203): EX(2,217): EX(2,221): EX(2,237): EX(2,253):
   EX(3,0): EX(3,1): EX(3,3): EX(3,4): EX(3,5): EX(3,18): EX(3,19):
   EX(3,20): EX(3,21): EX(3,24): EX(3,26): EX(3,28): EX(3,29):
   EX(3,30): EX(3,31): EX(3,33): EX(3,56): EX(3,58): EX(3,60):
   EX(3,61): EX(3,65): EX(3,66): EX(3,69): EX(3,75): EX(3,78):
   EX(3,81): EX(3,82): EX(3,85): EX(3,91): EX(3,94): EX(3,97):
   EX(3,98): EX(3,101): EX(3,107): EX(3,113): EX(3,114): EX(3,117):
   EX(3,123): EX(3,131): EX(3,135): EX(3,143): EX(3,147): EX(3,157):
   EX(3,163): EX(3,179): EX(3,195): EX(3,199): EX(3,204): EX(3,205):
   EX(3,207): EX(3,211): EX(3,220): EX(3,221): EX(3,227): EX(3,236):
   EX(3,237): EX(3,243): EX(3,252): EX(3,253):
             EX(4,45): EX(4,46): EX(4,47): EX(4,128):
     K=INS; CALL TOHEX(K,CDUMY); CDFT=CDBK;
     PUT LIST('****INSTRUCTION ERROR', CDUMY) SKIP;INHIB='1'B;
     GOTO TIMER;
/* RET GROUP FOR 80            */
   EX(2,192): IF(¬Z) THEN GOTO PRET80; GOTO TIMER;
   EX(2,200): IF( Z) THEN GOTO PRET80; GOTO TIMER;
   EX(2,208): IF(¬CY)THEN GOTO PRET80; GOTO TIMER;
   EX(2,216): IF(CY) THEN GOTO PRET80; GOTO TIMER;
   EX(2,224): IF(¬P) THEN GOTO PRET80; GOTO TIMER;
   EX(2,232): IF( P) THEN GOTO PRET80; GOTO TIMER;
   EX(2,240): IF(¬S) THEN GOTO PRET80; GOTO TIMER;
   EX(2,248): IF( S) THEN GOTO PRET80; GOTO TIMER;
   PRET80: DELTSW='1'B;
   EX(2,201): CALL POP(0,1,TEMPA,SP); PCN=TEMPA;
     CALL POP(0,1,TEMPA,SP); PCN=PCN+256*TEMPA;GOTO TIMER;
/* PUSH POP GROUPS FOR 80         */
   EX(2,197):CALL PUSH(1,0,B,SP);CALL PUSH(1,0,C,SP); GOTO TIMER;
   EX(2,193): CALL POP(0,1,C,SP);CALL POP(0,1,B,SP);GOTO TIMER;
   EX(2,213): CALL PUSH(1,0,D,SP); CALL PUSH(1,0,E,SP); GOTO TIMER;
   EX(2,209): CALL POP(0,1,E,SP); CALL POP(0,1,D,SP); GOTO TIMER;
   EX(2,229): CALL PUSH(1,0,H,SP); CALL PUSH(1,0,L,SP); GOTO TIMER;
   EX(2,225): CALL POP(0,1,L,SP); CALL POP(0,1,H,SP); GOTO TIMER;
   EX(2,245): CALL PUSH(1,0,A,SP);
     TEMPA=S||Z||'0'B||AC||'0'B||P||'1'B||CY;
     CALL PUSH(1,0,TEMPA,SP); GOTO TIMER;
   EX(2,241): CALL POP(0,1,TEMPA,SP); TST=TEMPA; S=SUBSTR(TST,1,1);
     Z=SUBSTR(TST,2,1); AC=SUBSTR(TST,4,1); P=SUBSTR(TST,6,1);
     CY=SUBSTR(TST,8,1); CALL POP(0,1,A,SP); GOTO TIMER;
/* CALL GROUP FOR 80          */
   EX(2,196): IF(¬Z) THEN GOTO PCAL80; GOTO TIMER;
   EX(2,204): IF( Z) THEN GOTO PCAL80; GOTO TIMER;
   EX(2,212): IF(¬CY)THEN GOTO PCAL80; GOTO TIMER;
   EX(2,220): IF(CY) THEN GOTO PCAL80; GOTO TIMER;
   EX(2,228): IF(¬P) THEN GOTO PCAL80; GOTO TIMER;
   EX(2,236): IF( P) THEN GOTO PCAL80; GOTO TIMER;
   EX(2,244): IF(¬S) THEN GOTO PCAL80; GOTO TIMER;
   EX(2,252): IF( S) THEN GOTO PCAL80; GOTO TIMER;
   PCAL80: DELTSW='1'B;
   EX(2,205): TEMPA=(PCN-MOD(PCN,256))/256; CALL PUSH(1,0,TEMPA,SP);
     TEMPA=MOD(PCN,256);CALL PUSH(1,0,TEMPA,SP);CALL FETCH(TEMPA,PC+1)
     ;PCN=TEMPA;CALL FETCH(TEMPA,PC+2); PCN=PCN+256*TEMPA;GOTO TIMER;
/* RST GROUP FOR 80      */
   EX(2,199):  TEMPA=(PC-MOD(PC,256))/256; CALL PUSH(1,0,TEMPA,SP);
     TEMPA=MOD(PC,256); CALL PUSH(1,0,TEMPA,SP);PCN= 0; GOTO TIMER;
   EX(2,207):  TEMPA=(PC-MOD(PC,256))/256; CALL PUSH(1,0,TEMPA,SP);
     TEMPA=MOD(PC,256); CALL PUSH(1,0,TEMPA,SP);PCN= 8; GOTO TIMER;
   EX(2,215):  TEMPA=(PC-MOD(PC,256))/256; CALL PUSH(1,0,TEMPA,SP);
```

```
     TEMPA=MOD(PC,256); CALL PUSH(1,0,TEMPA,SP);PCN=16; GOTO TIMER;
   EX(2,223):  TEMPA=(PC-MOD(PC,256))/256; CALL PUSH(1,0,TEMPA,SP);
     TEMPA=MOD(PC,256); CALL PUSH(1,0,TEMPA,SP);PCN=24; GOTO TIMER;
   EX(2,231):  TEMPA=(PC-MOD(PC,256))/256; CALL PUSH(1,0,TEMPA,SP);
     TEMPA=MOD(PC,256); CALL PUSH(1,0,TEMPA,SP);PCN=32; GOTO TIMER;
   EX(2,239):  TEMPA=(PC-MOD(PC,256))/256; CALL PUSH(1,0,TEMPA,SP);
     TEMPA=MOD(PC,256); CALL PUSH(1,0,TEMPA,SP);PCN=40; GOTO TIMER;
   EX(2,247):  TEMPA=(PC-MOD(PC,256))/256; CALL PUSH(1,0,TEMPA,SP);
     TEMPA=MOD(PC,256); CALL PUSH(1,0,TEMPA,SP);PCN=48; GOTO TIMER;
   EX(2,255):  TEMPA=(PC-MOD(PC,256))/256; CALL PUSH(1,0,TEMPA,SP);
     TEMPA=MOD(PC,256); CALL PUSH(1,0,TEMPA,SP);PCN=56; GOTO TIMER;
/* 80 PCHL, XCHG, SPHL, LDA, STA, NOP      */
   EX(2,233): PCN=256*H+L; GOTO TIMER;
   EX(2,235): TEMPA=H; H=D;D=TEMPA;TEMPA=L;L=E;E=TEMPA;GOTO TIMER;
   EX(2,249): SP=256*H+L; GOTO TIMER;
   EX(2,58): CALL FETCH(TEMPA,PC+1); TPOIN=TEMPA;
     CALL FETCH(TEMPA,PC+2); TPOIN=TPOIN+256*TEMPA;
     CALL FETCH(A,TPOIN); GOTO TIMER;
   EX(2,50): CALL FETCH(TEMPA,PC+1); TPOIN=TEMPA;
     CALL FETCH(TEMPA,PC+2); TPOIN=TPOIN+256*TEMPA;
     CALL STORE(A,TPOIN); GOTO TIMER;
   EX(2,0): GOTO TIMER;
/* HALT GROUP FOR 08 AND 80      */
   EX(1,0): EX(1,1): EX(1,255): EX(2,118): INHIB='1'B; GOTO TIMER;
/* RST GROUP FOR 08                  */
   EX(1,5): TEMPA=0; GOTO RST08;
   EX(1,13): TEMPA=1; GOTO RST08;
   EX(1,21):TEMPA=2; GOTO RST08;
   EX(1,29): TEMPA=3; GOTO RST08;
   EX(1,37): TEMPA=4; GOTO RST08;
   EX(1,45): TEMPA=5; GOTO RST08;
   EX(1,53): TEMPA=6; GOTO RST08;
   EX(1,61): TEMPA=7; GOTO RST08;
   RST08: STK08(SP08)=PCN; SP08=MOD(SP08+1,8); PCN=TEMPA*8; GOTO TIMER;
/* CALL GROUP FOR 08            */
```

```
   EX(1,66): IF(¬CY) THEN GOTO CALL08; ELSE GOTO TIMER;
   EX(1,74): IF(¬Z) THEN GOTO CALL08; ELSE GOTO TIMER;
   EX(1,82): IF(¬S) THEN GOTO CALL08; ELSE GOTO TIMER;
   EX(1,90): IF(¬P) THEN GOTO CALL08; ELSE GOTO TIMER;
   EX(1,98): IF(CY) THEN GOTO CALL08; ELSE GOTO TIMER;
   EX(1,106): IF(Z) THEN GOTO CALL08; ELSE GOTO TIMER;
   EX(1,114): IF(S) THEN GOTO CALL08; ELSE GOTO TIMER;
   EX(1,122): IF (P) THEN GOTO CALL08; ELSE GOTO TIMER;
   CALL08: EX(1,70): EX(1,78): EX(1,86): EX(1,94): EX(1,102):
   EX(1,110): EX(1,118): EX(1,126): STK08(SP08)=PCN; DELTSW='1'B;
     SP08=MOD(SP08+1,8); CALL FETCH(TEMPA,PC+1); PCN=TEMPA;
     CALL FETCH(TEMPA,PC+2); PCN=256*TEMPA+PCN; GOTO TIMER;
/* RETURN GROUP FOR 08       */
   EX(1,3): IF(¬CY) THEN GOTO RET08; ELSE GOTO TIMER;
   EX(1,11): IF(¬Z) THEN GOTO RET08; ELSE GOTO TIMER;
   EX(1,19): IF(¬S) THEN GOTO RET08; ELSE GOTO TIMER;
   EX(1,27): IF(¬P) THEN GOTO RET08; ELSE GOTO TIMER;
   EX(1,35): IF(CY) THEN GOTO RET08; ELSE GOTO TIMER;
   EX(1,43): IF (Z) THEN GOTO RET08; ELSE GOTO TIMER;
   EX(1,51): IF(S) THEN GOTO RET08; ELSE GOTO TIMER;
   EX(1,59): IF (P) THEN GOTO RET08; ELSE GOTO TIMER;
   RET08: EX(1,7): EX(1,15): EX(1,23): EX(1,31): EX(1,39):
   EX(1,47): EX(1,55): EX(1,63):  SP08=MOD(SP08+7,8);
    PCN=STK08(SP08); DELTSW='1'B; GOTO TIMER;
/* 80 INX DCX GROUP                */
   EX(2, 3): CALL ADD(C,C,0,'1'B,CYT,ACT,ACT,ACT,ACT,ACT);
     CALL ADD(B,B,0,CYT,CYT,ACT,ACT,ACT,ACT,ACT); GOTO TIMER;
   EX(2,19): CALL ADD(E,E,0,'1'B,CYT,ACT,ACT,ACT,ACT,ACT);
     CALL ADD(D,D,0,CYT,CYT,ACT,ACT,ACT,ACT,ACT); GOTO TIMER;
   EX(2,35): CALL ADD(L,L,0,'1'B,CYT,ACT,ACT,ACT,ACT,ACT);
     CALL ADD(H,H,0,CYT,CYT,ACT,ACT,ACT,ACT,ACT); GOTO TIMER;
   EX(2,51): SP=SP+1;IF(SP>65535)THEN SP=0; GOTO TIMER;
   EX(2,11): CALL ADD(C,C,255,'0'B,CYT,ACT,ACT,ACT,ACT,ACT);
     CALL ADD(B,B,255,CYT,CYT,ACT,ACT,ACT,ACT,ACT); GOTO TIMER;
   EX(2,27): CALL ADD(E,E,255,'0'B,CYT,ACT,ACT,ACT,ACT,ACT);
```

```
     CALL ADD(D,D,255,CYT,CYT,ACT,ACT,ACT,ACT,ACT); GOTO TIMER;
   EX(2,43): CALL ADD(L,L,255,'0'B,CYT,ACT,ACT,ACT,ACT,ACT);
     CALL ADD(H,H,255,CYT,CYT,ACT,ACT,ACT,ACT,ACT); GOTO TIMER;
   EX(2,59): SP=SP-1; IF (SP<0) THEN SP=65535; GOTO TIMER;
/* 80 DAD GROUP                   */
   EX(2, 9): CALL ADD(L,L,C,'0'B,CY,ACT,ACT,ACT,ACT,ACT);
     CALL ADD(H,H,B,CY,CY,ACT,ACT,ACT,ACT,ACT); GOTO TIMER;
   EX(2,25): CALL ADD(L,L,E,'0'B,CY,ACT,ACT,ACT,ACT,ACT);
     CALL ADD(H,H,D,CY,CY,ACT,ACT,ACT,ACT,ACT); GOTO TIMER;
   EX(2,41): CALL ADD(L,L,L,'0'B,CY,ACT,ACT,ACT,ACT,ACT);
     CALL ADD(H,H,H,CY,CY,ACT,ACT,ACT,ACT,ACT); GOTO TIMER;
   EX(2,57): TEMPA=MOD(SP,256);
     CALL ADD(L,L,TEMPA,'0'B,CY,ACT,ACT,ACT,ACT,ACT);
     TEMPA=(SP-TEMPA)/256;
     CALL ADD(H,H,TEMPA,CY,CY,ACT,ACT,ACT,ACT,ACT); GOTO TIMER;
/* LHLD   SHLD   FOR 80          */
   EX(2,42): CALL FETCH(TEMPA,PC+1); TPOIN=TEMPA;
     CALL FETCH(TEMPA,PC+2); TPOIN=TPOIN+256*TEMPA;
     CALL FETCH(L,TPOIN);CALL FETCH(H,TPOIN+1); GOTO TIMER;
   EX(2,34): CALL FETCH(TEMPA,PC+1); TPOIN=TEMPA;
     CALL FETCH(TEMPA,PC+2); TPOIN=TPOIN+256*TEMPA;
     CALL STORE(L,TPOIN); CALL STORE(H,TPOIN+1); GOTO TIMER;
/* 80  CMA,STC,CMC,EI,DI,XTHL            */
   EX(2,47): A=255-A; GOTO TIMER;
   EX(2,55): CY='1'B; GOTO TIMER;
   EX(2,63): CY=¬CY; GOTO TIMER;
   EX(2,251): INTE='1'B; GOTO TIMER;
   EX(2,243): INTE='0'B; GOTO TIMER;
   EX(2,227): TEMPA=L; CALL FETCH(L,SP); CALL STORE(TEMPA,SP);
     TEMPA=H; CALL FETCH(H,SP+1); CALL STORE(TEMPA,SP+1); GOTO TIMER;
/* INTEL INPUT GROUP                       */
   EX(1,65):  CALL TELIN(A,0); GOTO TIMER;
   EX(1,67):  CALL TELIN(A,1); GOTO TIMER;
   EX(1,69):  CALL TELIN(A,2); GOTO TIMER;
   EX(1,71):  CALL TELIN(A,3); GOTO TIMER;
   EX(1,73):  CALL TELIN(A,4); GOTO TIMER;
   EX(1,75):  CALL TELIN(A,5); GOTO TIMER;
   EX(1,77):  CALL TELIN(A,6); GOTO TIMER;
   EX(1,79):  CALL TELIN(A,7); GOTO TIMER;
   EX(2,219): CALL FETCH(TEMPA,PC+1);
     TEMPB=TEMPA; CALL TELIN(A,TEMPB); GOTO TIMER;
/* OUT GROUP FOR 08 AND 80      */
   EX(1,81): PORT=8;OUT=A;OUTSW='1'B; GOTO TIMER;
   EX(1,83): PORT= 9; OUT=A;OUTSW='1'B; GOTO TIMER;
   EX(1, 85): PORT=10;OUT=A;OUTSW='1'B; GOTO TIMER;
   EX(1, 87): PORT=11;OUT=A;OUTSW='1'B; GOTO TIMER;
   EX(1, 89): PORT=12;OUT=A;OUTSW='1'B; GOTO TIMER;
   EX(1, 91): PORT=13;OUT=A;OUTSW='1'B; GOTO TIMER;
   EX(1, 93): PORT=14;OUT=A;OUTSW='1'B; GOTO TIMER;
   EX(1, 95): PORT=15;OUT=A;OUTSW='1'B; GOTO TIMER;
   EX(1, 97): PORT=16;OUT=A;OUTSW='1'B; GOTO TIMER;
   EX(1, 99): PORT=17;OUT=A;OUTSW='1'B; GOTO TIMER;
   EX(1,101): PORT=18;OUT=A;OUTSW='1'B; GOTO TIMER;
   EX(1,103): PORT=19;OUT=A;OUTSW='1'B; GOTO TIMER;
   EX(1,105): PORT=20;OUT=A;OUTSW='1'B; GOTO TIMER;
   EX(1,107): PORT=21;OUT=A;OUTSW='1'B; GOTO TIMER;
   EX(1,109): PORT=22;OUT=A;OUTSW='1'B; GOTO TIMER;
   EX(1,111): PORT=23;OUT=A;OUTSW='1'B; GOTO TIMER;
   EX(1,113): PORT=24;OUT=A;OUTSW='1'B; GOTO TIMER;
   EX(1,115): PORT=25;OUT=A;OUTSW='1'B; GOTO TIMER;
   EX(1,117): PORT=26;OUT=A;OUTSW='1'B; GOTO TIMER;
   EX(1,119): PORT=27;OUT=A;OUTSW='1'B; GOTO TIMER;
   EX(1,121): PORT=28;OUT=A;OUTSW='1'B; GOTO TIMER;
   EX(1,123): PORT=29;OUT=A;OUTSW='1'B; GOTO TIMER;
   EX(1,125): PORT=30;OUT=A;OUTSW='1'B; GOTO TIMER;
   EX(1,127): PORT=31;OUT=A;OUTSW='1'B; GOTO TIMER;
   EX(2,211): CALL FETCH(TEMPA,PC+1); PORT=TEMPA;OUTSW='1'B;
     OUT=A; GOTO TIMER;
/* 08 JUMP GROUP    */
   EX(1,96): IF(CY) THEN GOTO JMP8; ELSE GOTO TIMER;
   EX(1,64): IF(¬CY) THEN GOTO JMP8; ELSE GOTO TIMER;
   EX(1,104): IF(Z) THEN GOTO JMP8; ELSE GOTO TIMER;
   EX(1,72):IF(¬Z) THEN GOTO JMP8; ELSE GOTO TIMER;
   EX(1,80):IF(¬S) THEN GOTO JMP8; ELSE GOTO TIMER;
   EX(1,112): IF(S) THEN GOTO JMP8; ELSE GOTO TIMER;
   EX(1,120): IF(P) THEN GOTO JMP8; ELSE GOTO TIMER;
   EX(1,88): IF(¬P) THEN GOTO JMP8; ELSE GOTO TIMER;
   JMP8: EX(1,68): EX(1,76): EX(1,84): EX(1,92):
   EX(1,100): EX(1,108): EX(1,116): EX(1,124):
     CALL FETCH(TEMPA,PC+1);
     PCN=TEMPA;
     CALL FETCH(TEMPA,PC+2);
     PCN=PCN+256*MOD(TEMPA,64);
     /* DELTSW SHOULD MAKE TIMER USE INCREMENTED MACH TIME  */
     DELTSW='1'B; GOTO TIMER;
/* JUMP GROUP FOR 80       */
   EX(2,194): IF(¬Z) THEN GOTO JMP80; ELSE GOTO TIMER;
   EX(2,202): IF(Z) THEN GOTO JMP80; ELSE GOTO TIMER;
   EX(2,210): IF(¬CY) THEN GOTO JMP80; ELSE GOTO TIMER;
   EX(2,218): IF(CY) THEN GOTO JMP80; ELSE GOTO TIMER;
   EX(2,226): IF(¬P) THEN GOTO JMP80; ELSE GOTO TIMER;
   EX(2,234): IF(P) THEN GOTO JMP80; ELSE GOTO TIMER;
   EX(2,242): IF(¬S) THEN GOTO JMP80; ELSE GOTO TIMER;
   EX(2,250): IF(S) THEN GOTO JMP80; ELSE GOTO TIMER;
   JMP80: EX(2,195):  CALL FETCH(TEMPA,PC+1);
     PCN=TEMPA;
     CALL FETCH(TEMPA,PC+2);
     PCN=PCN+256*TEMPA;  GOTO TIMER;
/* ROTATE GROUP FOR 08 & 80      */
   EX(1,2): EX(2,7): IF(A>127) THEN CY='1'B; ELSE CY='0'B;
     A=MOD(2*A,256);
     IF(CY) THEN A=A+1;
     GOTO TIMER;
   EX(1,10): EX(2,15): IF(MOD(A,2)¬=0) THEN CY='1'B; ELSE CY='0'B;
     A=(A-MOD(A,2))/2;
     IF(CY) THEN A=A+128;
     GOTO TIMER;
   EX(1,18): EX(2,23): IF(CY) THEN A=2*A+1; ELSE A=2*A;
     IF(A>255) THEN CY='1'B; ELSE CY='0'B;
     A=MOD(A,256);
     GOTO TIMER;
   EX(1,26): EX(2,31): IF(CY) THEN A=A+256;
     IF(MOD(A,2)¬=0) THEN CY='1'B; ELSE CY='0'B;
     A=(A-MOD(A,2))/2;
     GOTO TIMER;
/* 80 LDAX STAX AND LXI'S          */
   EX(2,2): TPOIN=256*B+C; GOTO STAX;
   EX(2,18): TPOIN=256*D+E; GOTO STAX;
   STAX: CALL STORE(A,TPOIN); GOTO TIMER;
   EX(2,10): TPOIN=256*B+C; GOTO LDAX;
   EX(2,26): TPOIN=256*D+E; GOTO LDAX;
   LDAX: CALL FETCH(A,TPOIN); GOTO TIMER;
   EX(2,1):CALL FETCH(B,PC+2); CALL FETCH(C,PC+1); GOTO TIMER;
   EX(2,17): CALL FETCH(D,PC+2); CALL FETCH(E,PC+1); GOTO TIMER;
   EX(2,33): CALL FETCH(H,PC+2); CALL FETCH(L,PC+1); GOTO TIMER;
   EX(2,49): CALL FETCH(TEMPA,PC+1); SP=TEMPA;
     CALL FETCH(TEMPA,PC+2); SP=SP+256*TEMPA; GOTO TIMER;
/* INR DCR GROUP       */
   EX(1,8): EX(2,4): CALL ADD(B,B,0,'1'B,DB,Z,S,P,AC,V);
     GOTO TIMER;
   EX(1,16): EX(2,12): CALL ADD(C,C,0,'1'B,DB,Z,S,P,AC,V);
     GOTO TIMER;
   EX(1,24): EX(2,20): CALL ADD(D,D,0,'1'B,DB,Z,S,P,AC,V);
     GOTO TIMER;
   EX(1,32): EX(2,28): CALL ADD(E,E,0,'1'B,DB,Z,S,P,AC,V);
     GOTO TIMER;
   EX(1,40): EX(2,36): CALL ADD(H,H,0,'1'B,DB,Z,S,P,AC,V);
     GOTO TIMER;
   EX(1,48): EX(2,44): CALL ADD(L,L,0,'1'B,DB,Z,S,P,AC,V);
     GOTO TIMER;
   EX(2,52): CALL FETCH(TEMPA,256*H+L);
     CALL ADD(TEMPA,TEMPA,0,'1'B,DB,Z,S,P,AC,V);
     CALL STORE(TEMPA,256*H+L); GOTO TIMER;
   EX(2,60): CALL ADD(A,A,0,'1'B,DB,Z,S,P,AC,V); GOTO TIMER;
   EX(1,9): EX(2,5): CALL ADD(B,B,255,'0'B,DB,Z,S,P,AC,V);
     GOTO TIMER;
   EX(1,17): EX(2,13): CALL ADD(C,C,255,'0'B,DB,Z,S,P,AC,V);
     GOTO TIMER;
   EX(1,25): EX(2,21): CALL ADD(D,D,255,'0'B,DB,Z,S,P,AC,V);
     GOTO TIMER;
   EX(1,33): EX(2,29): CALL ADD(E,E,255,'0'B,DB,Z,S,P,AC,V);
     GOTO TIMER;
   EX(1,41): EX(2,37): CALL ADD(H,H,255,'0'B,DB,Z,S,P,AC,V);
     GOTO TIMER;
   EX(1,49): EX(2,45): CALL ADD(L,L,255,'0'B,DB,Z,S,P,AC,V);
     GOTO TIMER;
   EX(2,53): CALL FETCH(TEMPA,256*H+L);
     CALL ADD(TEMPA,TEMPA,255,'0'B,DB,Z,S,P,AC,V);
     CALL STORE(TEMPA,256*H+L); GOTO TIMER;
   EX(2,61): CALL ADD(A,A,255,'0'B,DB,Z,S,P,AC,V); GOTO TIMER;
/* IMEDIATE OP GROUP     */
   EX(1,4): EX(2,198): CALL FETCH(TEMPA,PC+1); GOTO ADDG;
   EX(1,12): EX(2,206): CALL FETCH(TEMPA,PC+1); GOTO ADCG;
   EX(1,20): EX(2,214): CALL FETCH(TEMPA,PC+1); GOTO SUBG;
   EX(1,28): EX(2,222): CALL FETCH(TEMPA,PC+1); GOTO SBBG;
   EX(1,36): EX(2,230): CALL FETCH(TEMPA,PC+1); GOTO ANAG;
   EX(1,44): EX(2,238): CALL FETCH(TEMPA,PC+1); GOTO XRAG;
   EX(1,52): EX(2,246): CALL FETCH(TEMPA,PC+1); GOTO ORAG;
   EX(1,60): EX(2,254): CALL FETCH(TEMPA,PC+1); GOTO CMPG;
/* ADD GROUP  */
   EX(1,128): EX(2,135):  TEMPA=A; GOTO ADDG;
   EX(1,129): EX(2,128): TEMPA=B; GOTO ADDG;
   EX(1,130): EX(2,129): TEMPA=C; GOTO ADDG;
   EX(1,131): EX(2,130): TEMPA=D; GOTO ADDG;
   EX(1,132): EX(2,131): TEMPA=E; GOTO ADDG;
EX(1,133): EX(2,132): TEMPA=H; GOTO ADDG;
EX(1,134): EX(2,133): TEMPA=L; GOTO ADDG;
EX(1,135): EX(2,134): CALL FETCH(TEMPA,256*H+L); GOTO ADDG;
ADDG: CALL ADD(A,A,TEMPA,'0'B,CY,Z,S,P,AC,V); GOTO TIMER;
EX(1,136): EX(2,143): TEMPA=A; GOTO ADCG;
EX(1,137): EX(2,136): TEMPA=B; GOTO ADCG;
EX(1,138): EX(2,137): TEMPA=C; GOTO ADCG;
EX(1,139): EX(2,138): TEMPA=D; GOTO ADCG;
EX(1,140): EX(2,139): TEMPA=E; GOTO ADCG;
EX(1,141): EX(2,140): TEMPA=H; GOTO ADCG;
EX(1,142): EX(2,141): TEMPA=L; GOTO ADCG;
EX(1,143): EX(2,142): CALL FETCH(TEMPA,256*H+L); GOTO ADCG;
ADCG: CALL ADD(A,A,TEMPA,CY,CY,Z,S,P,AC,V); GOTO TIMER;
EX(1,144): EX(2,151): TEMPA=A; GOTO SUBG;
EX(1,145): EX(2,144): TEMPA=B; GOTO SUBG;
EX(1,146): EX(2,145): TEMPA=C; GOTO SUBG;
EX(1,147): EX(2,146): TEMPA=D; GOTO SUBG;
EX(1,148): EX(2,147): TEMPA=E; GOTO SUBG;
EX(1,149): EX(2,148): TEMPA=H; GOTO SUBG;
EX(1,150): EX(2,149): TEMPA=L; GOTO SUBG;
EX(1,151): EX(2,150): CALL FETCH(TEMPA,256*H+L); GOTO SUBG;
SUBG: CALL ADD(A,A,(255-TEMPA),'1'B,CY,Z,S,P,AC,V);
  IF(MACH=2) THEN CY=¬CY; GOTO TIMER;
EX(1,152): EX(2,159): TEMPA=A; GOTO SBBG;
EX(1,153): EX(2,152): TEMPA=B; GOTO SBBG;
EX(1,154): EX(2,153): TEMPA=C; GOTO SBBG;
EX(1,155): EX(2,154): TEMPA=D; GOTO SBBG;
EX(1,156): EX(2,155): TEMPA=E; GOTO SBBG;
EX(1,157): EX(2,156): TEMPA=H; GOTO SBBG;
EX(1,158): EX(2,157): TEMPA=L; GOTO SBBG;
EX(1,159): EX(2,158): CALL FETCH(TEMPA,256*H+L); GOTO SBBG;
SBBG: IF(MACH=2) THEN CY=¬CY; CALL ADD(A,A,(255-TEMPA),CY,CY,Z,
  S,P,AC,V); IF(MACH=2) THEN CY=¬CY; GOTO TIMER;
EX(1,160): EX(2,167): TEMPA=A; GOTO ANAG;
EX(1,161): EX(2,160): TEMPA=B; GOTO ANAG;
EX(1,162): EX(2,161):  TEMPA=C; GOTO ANAG;
EX(1,163): EX(2,162): TEMPA=D; GOTO ANAG;
EX(1,164): EX(2,163): TEMPA=E; GOTO ANAG;
EX(1,165): EX(2,164): TEMPA=H; GOTO ANAG;
EX(1,166): EX(2,165): TEMPA=L; GOTO ANAG;
EX(1,167): EX(2,166): CALL FETCH(TEMPA,256*H+L); GOTO ANAG;
ANAG: A=BOOL(A,TEMPA,'0001'B); GOTO LOGG;
EX(1,168): EX(2,175): TEMPA=A; GOTO XRAG;
EX(1,169): EX(2,168): TEMPA=B; GOTO XRAG;
EX(1,170): EX(2,169): TEMPA=C; GOTO XRAG;
EX(1,171): EX(2,170): TEMPA=D; GOTO XRAG;
EX(1,172): EX(2,171): TEMPA=E; GOTO XRAG;
EX(1,173): EX(2,172): TEMPA=H; GOTO XRAG;
EX(1,174): EX(2,173): TEMPA=L; GOTO XRAG;
EX(1,175): EX(2,174): CALL FETCH(TEMPA,256*H+L); GOTO XRAG;
XRAG: A=BOOL(A,TEMPA,'0110'B); GOTO LOGG;
EX(1,176): EX(2,183): TEMPA=A; GOTO ORAG;
EX(1,177): EX(2,176): TEMPA=B; GOTO ORAG;
EX(1,178): EX(2,177): TEMPA=C; GOTO ORAG;
EX(1,179): EX(2,178): TEMPA=D; GOTO ORAG;
EX(1,180): EX(2,179): TEMPA=E; GOTO ORAG;
EX(1,181): EX(2,180): TEMPA=H; GOTO ORAG;
EX(1,182): EX(2,181): TEMPA=L; GOTO ORAG;
EX(1,183): EX(2,182): CALL FETCH(TEMPA,256*H+L); GOTO ORAG;
ORAG: A=BOOL(A,TEMPA,'0111'B); GOTO LOGG;
LOGG: CALL LFLAGS(CY,Z,S,P,A); GOTO TIMER;
EX(1,184): EX(2,191): TEMPA=A; GOTO CMPG;
EX(1,185): EX(2,184): TEMPA=B; GOTO CMPG;
EX(1,186): EX(2,185): TEMPA=C; GOTO CMPG;
EX(1,187): EX(2,186): TEMPA=D; GOTO CMPG;
EX(1,188): EX(2,187): TEMPA=E; GOTO CMPG;
EX(1,189): EX(2,188): TEMPA=H; GOTO CMPG;
EX(1,190): EX(2,189): TEMPA=L; GOTO CMPG;
EX(1,191): EX(2,190): CALL FETCH(TEMPA,256*H+L); GOTO CMPG;
CMPG: CALL ADD(TEMPA,A,(255-TEMPA),'1'B,CY,Z,S,P,AC,V);
```

```
    IF(MACH=2) THEN CY=¬CY; GOTO TIMER;
/* MVI GROUP        */
   EX(1,6): EX(2,62): CALL FETCH(A,PC+1); GOTO TIMER;
   EX(1,14): EX(2,6): CALL FETCH(B,PC+1); GOTO TIMER;
   EX(1,22): EX(2,14): CALL FETCH(C,PC+1); GOTO TIMER;
   EX(1,30): EX(2,22): CALL FETCH(D,PC+1); GOTO TIMER;
   EX(1,38): EX(2,30): CALL FETCH(E,PC+1); GOTO TIMER;
   EX(1,46): EX(2,38): CALL FETCH(H,PC+1); GOTO TIMER;
   EX(1,54): EX(2,46): CALL FETCH(L,PC+1); GOTO TIMER;
   EX(1,62): EX(2,54): CALL FETCH(TEMPA,PC+1);
    CALL STORE(TEMPA,256*H+L); GOTO TIMER;
/* MOVE GROUP FOR 08 AND 80       */
   EX(1,192):EX(2,127): A=A; GOTO TIMER;
   EX(1,193):EX(2,120): A=B; GOTO TIMER;
   EX(1,194):EX(2,121): A=C; GOTO TIMER;
   EX(1,195): EX(2,122): A=D; GOTO TIMER;
   EX(1,196): EX(2,123): A=E; GOTO TIMER;
   EX(1,197): EX(2,124): A=H; GOTO TIMER;
   EX(1,198): EX(2,125): A=L; GOTO TIMER;
   EX(1,199): EX(2,126): CALL FETCH(A,256*H+L); GOTO TIMER;
   EX(1,200): EX(2,71): B=A; GOTO TIMER;
   EX(1,201): EX(2,64): B=B; GOTO TIMER;
   EX(1,202): EX(2,65): B=C; GOTO TIMER;
   EX(1,203): EX(2,66): B=D; GOTO TIMER;
   EX(1,204): EX(2,67): B=E; GOTO TIMER;
   EX(1,205): EX(2,68): B=H; GOTO TIMER;
   EX(1,206): EX(2,69): B=L; GOTO TIMER;
   EX(1,207): EX(2,70): CALL FETCH(B,256*H+L); GOTO TIMER;
   EX(1,208): EX(2,79): C=A; GOTO TIMER;
   EX(1,209): EX(2,72): C=B; GOTO TIMER;
   EX(1,210): EX(2,73): C=C; GOTO TIMER;
   EX(1,211): EX(2,74): C=D; GOTO TIMER;
   EX(1,212): EX(2,75): C=E; GOTO TIMER;
   EX(1,213): EX(2,76): C=H; GOTO TIMER;
   EX(1,214): EX(2,77): C=L; GOTO TIMER;
EX(1,215): EX(2,78): CALL FETCH(C,256*H+L); GCTO TIMER;
EX(1,216): EX(2,87): D=A; GOTO TIMER;
EX(1,217): EX(2,80): D=B; GOTO TIMER;
EX(1,218): EX(2,81): D=C; GOTO TIMER;
EX(1,219): EX(2,82): D=D; GOTO TIMER;
EX(1,220): EX(2,83): D=E; GOTO TIMER;
EX(1,221): EX(2,84): D=H; GOTO TIMER;
EX(1,222): EX(2,85): D=L; GOTO TIMER;
EX(1,223): EX(2,86): CALL FETCH(D,256*H+L); GOTO TIMER;
EX(1,224): EX(2,95): E=A; GOTO TIMER;
EX(1,225): EX(2,88): E=B; GOTO TIMER;
EX(1,226): EX(2,89): E=C; GOTO TIMER;
EX(1,227): EX(2,90): E=D; GOTO TIMER;
EX(1,228): EX(2,91): E=E; GOTO TIMER;
EX(1,229): EX(2,92): E=H; GOTO TIMER;
EX(1,230): EX(2,93): E=L; GOTO TIMER;
EX(1,231): EX(2,94): CALL FETCH(E,256*H+L); GCTO TIMER;
EX(1,232): EX(2,103): H=A; GOTO TIMER;
EX(1,233): EX(2,96): H=B; GOTO TIMER;
EX(1,234): EX(2,97): H=C; GOTO TIMER;
EX(1,235): EX(2,98): H=D; GOTO TIMER;
EX(1,236): EX(2,99): H=E; GOTO TIMER;
EX(1,237): EX(2,100): H=H; GOTO TIMER;
EX(1,238): EX(2,101): H=L; GCTO TIMER;
EX(1,239): EX(2,102): CALL FETCH(H,256*H+L); GOTO TIMER;
EX(1,240): EX(2,111): L=A; GOTO TIMER;
EX(1,241): EX(2,104): L=B; GOTO TIMER;
EX(1,242): EX(2,105): L=C; GOTO TIMER;
EX(1,243): EX(2,106): L=D; GOTO TIMER;
EX(1,244): EX(2,107): L=E; GOTO TIMER;
EX(1,245): EX(2,108): L=H; GOTO TIMER;
EX(1,246): EX(2,109): L=L; GOTO TIMER;
EX(1,247): EX(2,110): CALL FETCH(L,256*H+L); GOTO TIMER;
EX(1,248): EX(2,119): CALL STORE(A,256*H+L); GOTO TIMER;
EX(1,249): EX(2,112): CALL STORE(B,256*H+L); GOTO TIMER;
```

```
   EX(1,250): EX(2,113): CALL STORE(C,256*H+L); GOTO TIMER;
   EX(1,251): EX(2,114): CALL STORE(D,256*H+L); GOTO TIMER;
   EX(1,252): EX(2,115): CALL STORE(E,256*H+L); GOTO TIMER;
   EX(1,253): EX(2,116): CALL STORE(H,256*H+L); GOTO TIMER;
   EX(1,254): EX(2,117): CALL STORE(L,256*H+L); GOTO TIMER;
/* DAA FOR 80 AND         IS OK FOR 68    */
   EX(2,39): EX(3,25): CYT=CY; ACT='0'B;
     IF(MOD(A,16)>9|AC) THEN DO;
       CALL ADD(A,A,6,'0'B,CY,Z,S,P,AC,V); ACT=AC; END;
     CYT=CY|CYT;
     IF(A>=145|CYT) THEN CALL ADD(A,A,96,'0'B,CY,Z,S,P,AC,V);
     CY=CY|CYT; AC=ACT; GOTO TIMER;
/* BRANCH GROUP FOR 68   */
   EX(3,32): PCN=ADX; GOTO TIMER;          /* BRA */
   EX(3,34): IF(¬(CY|Z)) THEN PCN=ADX; GOTO TIMER;         /* BHI */
   EX(3,35): IF(CY|Z) THEN PCN=ADX; GOTO TIMER;     /* BLS*/
   EX(3,36): IF(¬CY) THEN PCN=ADX; GOTO TIMER;    /*BCC*/
   EX(3,37): IF(CY) THEN PCN=ADX; GOTO TIMER;     /*BCS*/
   EX(3,38): IF(¬Z) THEN PCN=ADX; GOTO TIMER;     /*BNE*/
   EX(3,39): IF(Z) THEN PCN=ADX; GOTO TIMER;       /*BEQ*/
   EX(3,40): IF(¬V) THEN PCN=ADX; GOTO TIMER;    /*BVC*/
   EX(3,41): IF(V) THEN PCN=ADX; GOTO TIMER;     /*BVS*/
   EX(3,42): IF(¬S) THEN PCN=ADX; GOTO TIMER;     /*BPL*/
   EX(3,43): IF(S) THEN PCN=ADX; GOTO TIMER;     /*BMI*/
   EX(3,44): IF(S&V|¬S&¬V) THEN PCN=ADX; GOTO TIMER;       /*BGE*/
   EX(3,45): IF(¬S&V|S&¬V) THEN PCN=ADX; GOTO TIMER;     /*BLT*/
   EX(3,46): IF(¬(Z|¬S&V|S&¬V)) THEN PCN=ADX; GOTO TIMER; /*BGT*/
   EX(3,47): IF(Z|¬S&V|S&¬V) THEN PCN=ADX; GOTO TIMER;    /*BLE*/
/* 68 ADDA ADDB ABA ADCA ADCB LDAA LDAB STAA STAB   */
   EX(3,139): EX(3,155): EX(3,171): EX(3,187): CALL FETCH(TEMPA,ADX);
     CALL ADD(A,A,TEMPA,'0'B,CY,Z,S,P,AC,V); GOTO TIMER;
   EX(3,203): EX(3,219): EX(3,235): EX(3,251): CALL FETCH(TEMPA,ADX);
     CALL ADD(B,B,TEMPA,'0'B,CY,Z,S,P,AC,V); GOTO TIMER;
   EX(3,27): CALL ADD(A,A,B,'0'B,CY,Z,S,P,AC,V); GOTO TIMER;
   EX(3,137): EX(3,153): EX(3,169): EX(3,185): CALL FETCH(TEMPA,ADX);
   CALL ADD(A,A,TEMPA,CY,CY,Z,S,P,AC,V); GOTO TIMER;
  EX(3,201): EX(3,217): EX(3,233): EX(3,249): CALL FETCH(TEMPA,ADX);
   CALL ADD(B,B,TEMPA,CY,CY,Z,S,P,AC,V); GOTO TIMER;
  EX(3,134): EX(3,150): EX(3,166): EX(3,182): CALL FETCH(A,ADX);
   CALL LFLAGS(V,Z,S,P,A); GOTO TIMER;
  EX(3,198): EX(3,214): EX(3,230): EX(3,246): CALL FETCH(B,ADX);
   CALL LFLAGS(V,Z,S,P,B);      GOTO TIMER;
  EX(3,151): EX(3,167): EX(3,183): CALL LFLAGS(V,Z,S,P,A);
   CALL STORE(A,ADX); GOTO TIMER;
  EX(3,215): EX(3,231): EX(3,247): CALL LFLAGS(V,Z,S,P,B);
   CALL STORE(B,ADX); GOTO TIMER;
/* 68 TAB TBA PSHA PSHB PULA PULB     */
  EX(3,22): CALL LFLAGS(V,Z,S,P,A); B=A ; GOTO TIMER;
  EX(3,23): CALL LFLAGS(V,Z,S,P,B); A=B ; GOTO TIMER;
  EX(3,54):  CALL PUSH(0,1,A,SP); GOTO TIMER;
  EX(3,55):  CALL PUSH(0,1,B,SP); GOTO TIMER;
  EX(3,50): CALL POP(1,0,A,SP); GOTO TIMER;
  EX(3,51): CALL POP(1,0,B,SP); GOTO TIMER;
/* 68 ANDA ANDB ORAA ORAB EORA EORB BITA BITB   */
  EX(3,132): EX(3,148): EX(3,164): EX(3,180): CALL FETCH(TEMPA,ADX);
   A=BOOL(A,TEMPA,'0001'); CALL LFLAGS(V,Z,S,P,A); GOTO TIMER;
  EX(3,196): EX(3,212): EX(3,228): EX(3,244): CALL FETCH(TEMPA,ADX);
   B=BOOL(B,TEMPA,'0001'); CALL LFLAGS(V,Z,S,P,B); GOTO TIMER;
  EX(3,138): EX(3,154): EX(3,170): EX(3,186): CALL FETCH(TEMPA,ADX);
   A=BOOL(A,TEMPA,'0111'); CALL LFLAGS(V,Z,S,P,A); GOTO TIMER;
  EX(3,202): EX(3,218): EX(3,234): EX(3,250): CALL FETCH(TEMPA,ADX);
   B=BOOL(B,TEMPA,'0111'); CALL LFLAGS(V,Z,S,P,B); GOTO TIMER;
  EX(3,136): EX(3,152): EX(3,168): EX(3,184): CALL FETCH(TEMPA,ADX);
   A=BOOL(A,TEMPA,'0110'); CALL LFLAGS(V,Z,S,P,A); GOTO TIMER;
  EX(3,200): EX(3,216): EX(3,232): EX(3,248): CALL FETCH(TEMPA,ADX);
   B=BOOL(B,TEMPA,'0110'); CALL LFLAGS(V,Z,S,P,B); GOTO TIMER;
  EX(3,133): EX(3,149): EX(3,165): EX(3,181): CALL FETCH(TEMPA,ADX);
   TEMPA=BOOL(A,TEMPA,'0001');CALL LFLAGS(V,Z,S,P,TEMPA);GOTO TIMER;
  EX(3,197): EX(3,213): EX(3,229): EX(3,245): CALL FETCH(TEMPA,ADX);
   TEMPA=BOOL(B,TEMPA,'0001');CALL LFLAGS(V,Z,S,P,TEMPA); GOTO TIMER;
```

```
/* 69  SUBA SBCA CMPA SUBB SBCB CMPB  */
   EX(3,128): EX(3,144): EX(3,160): EX(3,176): CALL FETCH(TEMPA,ADX);
     CALL ADD(A,A,(255-TEMPA),'1'B,CY,Z,S,P,DB,V); CY=¬CY; GOTO TIMER;
   EX(3,130): EX(3,146): EX(3,162): EX(3,178): CALL FETCH(TEMPA,ADX);
     CY=¬CY; CALL ADD(A,A,(255-TEMPA),CY  ,CY,Z,S,P,DB,V); CY=¬CY;
     GOTO TIMER;
   EX(3,129): EX(3,145): EX(3,161): EX(3,I77): CALL FETCH(TEMPA,ADX);
     CALL ADD(TEMPA,A,(255-TEMPA),'1'B,CY,Z,S,P,DB,V); CY=¬CY;
     GOTO TIMER;
   EX(3,192): EX(3,208): EX(3,224): EX(3,240): CALL FETCH(TEMPA,ADX);
     CALL ADD(B,B,(255-TEMPA),'1'B,CY,Z,S,P,DB,V); CY=¬CY; GOTO TIMER;
   EX(3,194): EX(3,210): EX(3,226): EX(3,242): CALL FETCH(TEMPA,ADX);
     CY=¬CY; CALL ADD(B,B,(255-TEMPA), CY,CY,Z,S,P,DB,V); CY=¬CY;
     GOTO TIMER;
   EX(3,193): EX(3,209): EX(3,225): EX(3,241): CALL FETCH(TEMPA,ADX);
     CALL ADD(TEMPA,B,(255-TEMPA),'1'B,CY,Z,S,P,DB,V); CY=¬CY;
     GOTO TIMER;
/* 68 CPX DEX INX LDX STX           */
   EX(3,140): EX(3,156): EX(3,172): EX(3,188): C=MOD(INDX,256);
     D=(INDX-C)/256; CALL FETCH(TEMPA,ADX+1);
     CALL ADD(E,C,(255-TEMPA),'1'B,DB2,DB3,S,P,P,V);
     CALL FETCH(TEMPA,ADX);
     CALL ADD(E,C,(255-TEMPA),DB2 ,DB2,Z,S,P,P,V);
     Z=Z&DB3; GOTO TIMER;
   EX(3,9): INDX=INDX-1; IF(INDX<0) THEN INDX=65535;
     IF(INDX=0) THEN Z='1'B; ELSE Z='0'B; GOTO TIMER;
   EX(3,8): INDX=INDX+1; IF(INDX>65535)THEN INDX=0;
     IF(INDX=0) THEN Z='1'B; ELSE Z='0'B; GOTO TIMER;
   EX(3,206): EX(3,222): EX(3,238): EX(3,254): CALL FETCH(TEMPA,ADX);
     INDX=TEMPA*256; CALL FETCH(TEMPA,ADX+1); INDX=INDX+TEMPA;
     V='0'B; IF(INDX=0) THEN Z='1'B; ELSE Z='0'B;
     IF(INDX>32767) THEN S='1'B; ELSE S='0'B;
     GOTO TIMER;
   EX(3,223): EX(3,239): EX(3,255): C=MOD(INDX,256);
     D=(INDX-C)/256; CALL STORE(D,ADX); CALL STORE(C,ADX+1);
```

```
      V='0'B; IF(INDX=0) THEN Z='1'B; ELSE Z='0'B;
      IF(INDX>32767) THEN S='1'B; ELSE S='0'B;
      GOTO TIMER;
/* 68  NOP LDS STS INS DES TXS TSX     */
    EX(3,2): GOTO TIMER;
    EX(3,142): EX(3,158): EX(3,174): EX(3,190): CALL FETCH(TEMPA,ADX);
      SP=TEMPA*256; CALL FETCH(TEMPA,ADX+1); SP=SP+TEMPA;
      V='0'B; IF(SP=0) THEN Z='1'B; ELSE Z='0'B;
      IF(SP>32767) THEN S='1'B; ELSE S='0'B; GOTO TIMER;
    EX(3,159): EX(3,175): EX(3,191): C=MOD(SP,256); D=(SP-C)/256;
      CALL STORE(D,ADX); CALL STORE(C,ADX+1);
      V='0'B; IF(SP=0) THEN Z='1'B; ELSE Z='0'B;
      IF(SP>32767) THEN S='1'B; ELSE S='0'B; GOTO TIMER;
    EX(3,49): SP=SP+1; IF(SP>65535) THEN SP=0; GOTO TIMER;
    EX(3,52): SP=SP-1; IF(SP<0) THEN SP=65535; GOTO TIMER;
    EX(3,53): SP=INDX-1; IF(SP<0) THEN SP=65535; GOTO TIMER;
    EX(3,48): INDX=SP+1; IF(INDX>65535) THEN INDX=0; GOTO TIMER;
/* 68 (BSR,JSR)  RTS JMP TAP TPA     */
    EX(3,141): EX(3,173): EX(3,189): C=MOD(PCN,256);
      D=(PCN-C)/256; CALL PUSH(0,1,C,SP); CALL PUSH(0,1,D,SP);
      PCN=ADX; GOTO TIMER;
    EX(3,57): CALL POP(1,0,D,SP); CALL POP(1,0,C,SP);
      PCN=256*D+C;  GOTO TIMER;
    EX(3,110): EX(3,126): PCN=ADX; GOTO TIMER;
    EX(3,6): TST=A; AC=SUBSTR(TST,3,1); I68=SUBSTR(TST,4,1);
      S=SUBSTR(TST,5,1); Z=SUBSTR(TST,6,1); V=SUBSTR(TST,7,1);
      CY=SUBSTR(TST,8,1); GOTO TIMER;
    EX(3,7): A='11'B||AC||I68||S||Z||V||CY; GOTO TIMER;
/* 68 CLV SEV CLC SEC CLI SEI  SBA CBA    */
    EX(3,10): V='0'B; GOTO TIMER;
    EX(3,11): V='1'B; GOTO TIMER;
    EX(3,12): CY='0'B; GOTO TIMER;
    EX(3,13): CY='1'B; GOTO TIMER;
    EX(3,14): I68='1'B; GOTO TIMER;
    EX(3,15): I68='0'B; GOTO TIMER;
```

```
   EX(3,16): CALL ADD(A,A,(255-B),'1'B,CY,Z,S,P,DB,V); CY=¬CY;
     GOTO TIMER;
   EX(3,17): CALL ADD(C,A,(255-B),'1'B,CY,Z,S,P,DB,V); CY=¬CY;
     GOTO TIMER;
/* 68 NEGA NEGB NEG  COMA COMB COM   */
   EX(3,64): A=256-A; IF(A>127) THEN S='1'B; ELSE S='0'B;
     IF(A=0) THEN Z='1'B; ELSE Z='0'B;
     IF(A=128) THEN V='1'B; ELSE V='0'B; CY=Z;
     GOTO TIMER;
   EX(3,80): B=256-B; IF(B>127) THEN S='1'B; ELSE S='0'B;
     IF(B=0) THEN Z='1'B; ELSE Z='0'B;
     IF(B=128) THEN V='1'B; ELSE V='0'B; CY=Z;
     GOTO TIMER;
   EX(3,96): EX(3,112): CALL FETCH(C,ADX);
             C=256-C; IF(C>127) THEN S='1'B; ELSE S='0'B;
     IF(C=0) THEN Z='1'B; ELSE Z='0'B;
     IF(C=128) THEN V='1'B; ELSE V='0'B; CY=Z;
     CALL STORE(C,ADX);       GOTO TIMER;
   EX(3,67): A=255-A; IF(A=0) THEN Z='1'B;ELSE Z='0'B;
     IF(A>127) THEN S='1'B; ELSE S='0'B; V='0'B; CY='1'B;
     GOTO TIMER;
   EX(3,83): B=255-B; IF(B=0) THEN Z='1'B;ELSE Z='0'B;
     IF(B>127) THEN S='1'B; ELSE S='0'B; V='0'B; CY='1'B;
     GOTO TIMER;
   EX(3,99): EX(3,115): CALL FETCH(C,ADX);
             C=255-C; IF(C=0) THEN Z='1'B;ELSE Z='0'B;
     IF(C>127) THEN S='1'B; ELSE S='0'B; V='0'B; CY='1'B;
     CALL STORE(C,ADX); GOTO TIMER;
/* 68 DECA DECB DEC INCA INCB INC TSTA TSTB TST CLRA CLRB CLF */
   EX(3,74): CALL ADD(A,A,255,'0'B,DB,Z,S,P,DB,V); GOTO TIMER;
   EX(3,90): CALL ADD(B,B,255,'0'B,DB,Z,S,P,DB,V); GOTO TIMER;
   EX(3,106): EX(3,122): CALL FETCH(C,ADX);
             CALL ADD(C,C,255,'0'B,DB,Z,S,P,DB,V);
     CALL STORE(C,ADX); GOTO TIMER;
   EX(3, 76): CALL ADD(A,A,1,'0'B,DB,Z,S,P,DB,V); GOTO TIMER;
   EX(3, 92): CALL ADD(B,B,1,'0'B,DB,Z,S,P,DB,V); GOTO TIMER;
   EX(3,108): EX(3,124): CALL FETCH(C,ADX);
              CALL ADD(C,C,1,'0'B,DB,Z,S,P,DB,V);
     CALL STORE(C,ADX); GOTO TIMER;
   EX(3,77): IF(A=0) THEN Z='1'B; ELSE Z='0'B;
     IF(A>127) THEN S='1'B; ELSE S='0'B; CY='0'B; V='0'B; GOTO TIMER;
   EX(3,93): IF(B=0) THEN Z='1'B; ELSE Z='0'B;
     IF(B>127) THEN S='1'B; ELSE S='0'B; CY='0'B; V='0'B; GOTO TIMER;
   EX(3,109): EX(3,125): CALL FETCH(C,ADX);
             IF(C=0) THEN Z='1'B; ELSE Z='0'B;
     IF(C>127) THEN S='1'B; ELSE S='0'B; CY='0'B; V='0'B; GOTO TIMER;
   EX(3,79): A=0; S='0'B;Z='1'B;V='0'B;CY='0'B; GOTO TIMER;
   EX(3,95): B=0; S='0'B;Z='1'B;V='0'B;CY='0'B; GOTO TIMER;
   EX(3,111): EX(3,127): C=0; CALL STORE(C,ADX);
                  S='0'B;Z='1'B;V='0'B;CY='0'B; GOTO TIMER;
/* 68 LSRA LSRB LSR  ASRA ASRB ASR  RORA RORB ROR  */
   EX(3,68): D=MOD(A,2); IF(D¬=0) THEN CY='1'B; ELSE CY='0'B;
     A=(A-D)/2;
     D=A; GOTO FL68;
   EX(3,84): D=MOD(B,2); IF(D¬=0) THEN CY='1'B; ELSE CY='0'B;
     B=(B-D)/2;
     D=B; GOTO FL68;
   EX(3,100): EX(3,116): CALL FETCH(C,ADX);
             D=MOD(C,2); IF(D¬=0) THEN CY='1'B; ELSE CY='0'B;
     C=(C-D)/2;
     D=C; GOTO FLST68;
   EX(3,71): D=MOD(A,2); IF(D¬=0)THEN CY='1'B; ELSE CY='0'B;
     A=(A-D)/2; IF(S) THEN A=A+128;
     D=A; GOTO FL68;
   EX(3,87): D=MOD(B,2); IF(D¬=0)THEN CY='1'B; ELSE CY='0'B;
     B=(B-D)/2; IF(S) THEN B=B+128;
     D=B; GOTO FL68;
   EX(3,103): EX(3,119): CALL FETCH(C,ADX);
             D=MOD(C,2); IF(D¬=0)THEN CY='1'B; ELSE CY='0'B;
     C=(C-D)/2; IF(S) THEN C=C+128;
     D=C; GOTO FLST68;
   EX(3,70): DB=CY; D=MOD(A,2); IF(D¬=0) THEN CY='1'B; ELSE CY='0'B;
     A=(A-D)/2; IF(DB)THEN A=A+128;
     D=A; GOTO FL68;
   EX(3,86): DB=CY; D=MOD(B,2); IF(D¬=0) THEN CY='1'B; ELSE CY='0'B;
     B=(B-D)/2; IF(DB)THEN B=B+128;
     D=B; GOTO FL68;
   EX(3,102): EX(3,118): CALL FETCH(C,ADX);
             DB=CY; D=MOD(C,2); IF(D¬=0) THEN CY='1'B; ELSE CY='0'B;
     C=(C-D)/2; IF(DB)THEN C=C+128;
     D=C; GOTO FLST68;
/* 68 ASLA ASLB ASL  ROLA ROLB ROL      */
   EX(3,72): IF(A>127) THEN CY='1'B; ELSE CY='0'B; A=MOD(2*A,256);
     D=A; GOTO FL68;
   EX(3,88): IF(B>127) THEN CY='1'B; ELSE CY='0'B; B=MOD(2*B,256);
     D=B; GOTO FL68;
   EX(3,104): EX(3,120): CALL FETCH(C,ADX);
             IF(C>127) THEN CY='1'B; ELSE CY='0'B; C=MOD(2*C,256);
     D=C; GOTO FLST68;
   EX(3,73): DB=CY; IF(A>127) THEN CY='1'B; ELSE CY='0'B;
     A=MOD(2*A,256); IF(DB) THEN A=A+1;
     D=A; GOTO FL68;
   EX(3,89): DB=CY; IF(B>127) THEN CY='1'B; ELSE CY='0'B;
     B=MOD(2*B,256); IF(DB) THEN B=B+1;
     D=B; GOTO FL68;
   EX(3,105): EX(3,121): CALL FETCH(C,ADX);
             DB=CY; IF(C>127) THEN CY='1'B; ELSE CY='0'B;
     C=MOD(2*C,256); IF(DB) THEN C=C+1;
     D=C; GOTO FLST68;
   FLST68: CALL STORE(C,ADX);
   FL68: IF(D>127) THEN S='1'B; ELSE S='0'B;
   IF(D=0) THEN Z='1'B; ELSE Z='0'B;
     IF(¬S&CY|S&¬CY) THEN V='1'B; ELSE V='0'B;
     GOTO TIMER;
/* 68 SWI RTI WAI     */
```

```
   EX(3,63): C=MOD(PCN,256); CALL PUSH(0,1,C,SP);
       C=(PCN-C)/256; CALL PUSH(0,1,C,SP);
       C=MOD(INDX,256); CALL PUSH(0,1,C,SP);
       C=(INDX-C)/256; CALL PUSH(0,1,C,SP);
       CALL PUSH(0,1,A,SP);
       CALL PUSH(0,1,B,SP);
       C='11'B||AC||I68||S||Z||V||CY; CALL PUSH(0,1,C,SP);
       CALL FETCH(C,65530); PCN=256*C;CALL FETCH(C,65531); PCN=PCN+C;
       GOTO TIMER;
   EX(3,59): CALL POP(1,0,C,SP); TST=C;
       AC=SUBSTR(TST,3,1); I68=SUBSTR(TST,4,1);
       S=SUBSTR(TST,5,1); Z=SUBSTR(TST,6,1);
       V=SUBSTR(TST,7,1); CY=SUBSTR(TST,8,1);
       CALL POP(1,0,B,SP); CALL POP(1,0,A,SP);
       CALL POP(1,0,C,SP); INDX=256*C; CALL POP(1,0,C,SP); INDX=INDX+C;
       CALL POP(1,0,C,SP); PCN=C*256; CALL POP(1,0,C,SP); PCN=PCN+C;
       GOTO TIMER;
   EX(3,62): C=MOD(PCN,256); CALL PUSH(0,1,C,SP);
       C=(PCN-C)/256; CALL PUSH(0,1,C,SP);
       C=MOD(INDX,256); CALL PUSH(0,1,C,SP);
       C=(INDX-C)/256; CALL PUSH(0,1,C,SP);
       CALL PUSH(0,1,A,SP);
       CALL PUSH(0,1,B,SP);
       C='11'B||AC||I68||S||Z||V||CY; CALL PUSH(0,1,C,SP);
       INHIB='1'B; GOTO TIMER;
/* F8 NOPS:    */
   EX(4,43): EX(4,63): EX(4,79): EX(4,95): EX(4,207):
   EX(4,223): EX(4,239): EX(4,255):
   GOTO TIMER;
/* F8 00 THROUGH 32    */
   EX(4,0): A=SCF8(12); GOTO TIMER;
   EX(4,1): A=SCF8(13); GOTO TIMER;
   EX(4,2): A=SCF8(14); GOTO TIMER;
   EX(4,3): A=SCF8(15); GOTO TIMER;
   EX(4,4): SCF8(12)=A; GOTO TIMER;
EX(4,5): SCF8(13)=A; GOTO TIMER;
EX(4,6): SCF8(14)=A; GOTO TIMER;
EX(4,7): SCF8(15)=A; GOTO TIMER;
EX(4,8): SCF8(13)= MOD(PC1,256);SCF8(12)=(PC1-SCF8(13))/256;
  GOTO TIMER;
EX(4,9):PCN=256*SCF8(14)+SCF8(15);  GOTO TIMER;
EX(4,10): A=MOD(ISAR,64); GOTO TIMER;
EX(4,12):PC1=PCN;PCN=SCF8(12)*256+SCF8(13); GOTO TIMER;
EX(4,13): DC=SCF8(14)*256+SCF8(15); GOTO TIMER;
EX(4,14): SCF8(15)=MOD(DC,256); SCF8(14)=(DC-SCF8(15))/256;
  GOTO TIMER;
EX(4,15): DC=SCF8(14)*256+SCF8(15); GOTO TIMER;
EX(4,16): DC=SCF8(10)*256+SCF8(11); GOTO TIMER;
EX(4,17): SCF8(11)=MOD(DC,256); SCF8(10)=(DC-SCF8(11))/256;
  GOTO TIMER;
EX(4,18): A=(A-MOD(A,2))/2; V='0'B;IF(A=0) THEN Z='1'B;ELSE Z='0'B;
  CY='0'B; S='1'B; GOTO TIMER;
EX(4,20):A=(A-MOD(A,16))/16; V='0'B; S='1'B; CY='0'B;
  IF(A=0) THEN Z='1'B;ELSE Z='0'B; GOTO TIMER;
EX(4,19): A=MOD(A*2,256); IF(A>127) THEN S='0'B;ELSE S='1'B;
  IF(A=0) THEN Z='1'B; ELSE Z='0'B;V='0'B; CY='0'B; GOTO TIMER;
EX(4,21): A=MOD(A*16,256); IF(A>127) THEN S='0'B; ELSES='1'B;
  IF(A=0) THEN Z='1'B; ELSE Z='0'B; V='0'B; CY='0'B; GOTO TIMER;
EX(4,22): CALL FETCH(A,DC); DC=MOD(DC+1,65536); GOTO TIMER;
EX(4,23): CALL STORE(A,DC); DC=MOD(DC+1,65536); GOTO TIMER;
EX(4,24): A=255-A; IF(A=0) THEN Z='1'B; ELSE Z='0'B;
  IF(A>127) THEN S='0'B; ELSE S='1'B; CY='0'B;V='0'B; GOTO TIMER;
EX(4,25): CALL ADD(A,A,0,CY,CY,Z,S,P,AC,V); S=¬S; GOTO TIMER;
EX(4,26): ICB='0'B; GOTO TIMER;
EX(4,27): ICB='1'B; GOTO TIMER;
EX(4,28):PCN=PC1 ; GOTO TIMER;
EX(4,29): TST=SCF8(9); ICB=SUBSTR(TST,4,1); V=SUBSTR(TST,5,1);
  Z=SUBSTR(TST,6,1); CY=SUBSTR(TST,7,1); S=SUBSTR(TST,8,1);
  GOTO TIMER;
EX(4,30): SCF8(9)='000'B||ICB||V||Z||CY||S; GOTO TIMER;
```

```
   EX(4,31): CALL ADD(A,A,1,'0'B,CY,Z,S,P,AC,V); S=¬S; GOTO TIMER;
   EX(4,32): CALL FETCH(A,PC+1); GOTO TIMER;
/* F8 BRANCH GROUPS  */
   EX(4,129): ACT=S; GOTO BRF8;
   EX(4,130): ACT=CY; GOTO BRF8;
   EX(4,131): ACT=S|CY; GOTO BRF8;
   EX(4,132): ACT=Z; GOTO BRF8;
   EX(4,133): ACT=S|Z; GOTO BRF8;
   EX(4,134): ACT=CY|Z; GOTO BRF8;
   EX(4,135): ACT=S|CY|Z; GOTO BRF8;
   EX(4,143): IF(MOD(ISAR,8)¬=7) THEN ACT='1'B; ELSE ACT='0'B;
     GOTO BRF8;
   EX(4,144): ACT='1'B; GOTO BRF8;
   EX(4,145): ACT=¬S; GOTO BRF8;
   EX(4,146): ACT=¬CY; GOTO BRF8;
   EX(4,147): ACT=¬CY&¬S; GOTO BRF8;
   EX(4,148): ACT=¬Z; GOTO BRF8;
   EX(4,149): ACT=¬Z&¬S; GOTO BRF8;
   EX(4,150): ACT=¬Z&¬CY; GOTO BRF8;
   EX(4,151): ACT=¬Z&¬CY&¬S; GOTO BRF8;
   EX(4,152): ACT=¬V; GOTO BRF8;
   EX(4,153): ACT=¬V&¬S; GOTO BRF8;
   EX(4,154): ACT=¬V&¬CY; GOTO BRF8;
   EX(4,155): ACT=¬V&¬CY&¬S; GOTO BRF8;
   EX(4,156): ACT=¬V&¬Z; GOTO BRF8;
   EX(4,157): ACT=¬V&¬Z&¬S; GOTO BRF8;
   EX(4,158): ACT=¬V&¬Z&¬CY; GOTO BRF8;
   EX(4,159): ACT=¬V&¬Z&¬CY&¬S; GOTO BRF8;
   BRF8: IF(ACT) THEN DO;
     CALL FETCH(C,PC+1); IF(C>127)THEN C=C-256;PCN=PC+1+C;
     DELTSW='1'B; END;
     GOTO TIMER;
/* F8 IN AND OUTS  */
   EX(4,38):  CALL FETCH(C,PC+1);    PT=C; GOTO INF;
   EX(4,160): PT=0; ITIME=ITIME-8; GOTO INF;
EX(4,161): PT=1; ITIME=ITIME-8; GOTO INF;
EX(4,162): PT=2; GOTO INF;
EX(4,163): PT=3; GOTO INF;
EX(4,164): PT=4; GOTO INF;
EX(4,165): PT=5; GOTO INF;
EX(4,156): PT=6; GOTO INF;
EX(4,167): PT=7; GOTO INF;
EX(4,168): PT=8; GOTO INF;
EX(4,169): PT=9; GOTO INF;
EX(4,170): PT=10;GOTO INF;
EX(4,171): PT=11;GOTO INF;
EX(4,172): PT=12;GOTO INF;
EX(4,173): PT=13;GOTO INF;
EX(4,174): PT=14;GOTO INF;
EX(4,175): PT=15;GOTO INF;
INF: CALL TELIN(A,PT); V='0'B;CY='0'B;IF(A>127) THEN S='0'B;
  ELSE S='1'B;IF(A=0) THEN Z='1'B; ELSE Z='0'B; GOTO TIMER;
EX(4,39): CALL FETCH(C,PC+1); PORT=C; GOTO OUTF;
EX(4,176): PORT=0 ; ITIME=ITIME-8; GOTO OUTF;
EX(4,177): PORT=1 ; ITIME=ITIME-8; GOTO OUTF;
EX(4,178): PORT=2 ; GOTO OUTF;
EX(4,179): PORT=3 ; GOTO OUTF;
EX(4,190): POPT=4 ; GOTO OUTF;
EX(4,181): PORT=5 ; GOTO OUTF;
EX(4,132): PORT=6 ; GOTO OUTF;
EX(4,183): PORT=7 ; GOTO OUTF;
EX(4,134): PORT=8 ; GOTO OUTF;
EX(4,135): PORT=9 ; GOTO OUTF;
EX(4,186): PORT=10; GOTO OUTF;
EX(4,137): PORT=11; GOTO OUTF;
EX(4,198): PORT=12; GOTO OUTF;
EX(4,199): PORT=13; GOTO OUTF;
EX(4,190): PORT=14; GOTO OUTF;
EX(4,191): PORT=15; GOTO OUTF;
OUTF: OUT=A; OUTSW='1'B; GOTO TIMER;
```

```
/* LISA FOR F8   ALSO LSIAR  */
   EX(4,112): A=0 ; GOTO TIMER;
   EX(4,113): A=1 ; GOTO TIMER;
   EX(4,114): A=2 ; GOTO TIMER;
   EX(4,115): A=3 ; GOTO TIMER;
   EX(4,116): A=4 ; GOTO TIMER;
   EX(4,117): A=5 ; GOTO TIMER;
   EX(4,118): A=6 ; GOTO TIMER;
   EX(4,119): A=7 ; GOTO TIMER;
   EX(4,120): A=8 ; GOTO TIMER;
   EX(4,121): A=9 ; GOTO TIMER;
   EX(4,122): A=10; GOTO TIMER;
   EX(4,123): A=11; GOTO TIMER;
   EX(4,124): A=12; GOTO TIMER;
   EX(4,125): A=13; GOTO TIMER;
   EX(4,126): A=14; GOTO TIMER;
   EX(4,127): A=15; GOTO TIMER;
   EX(4,11):ISAR=MOD(A,64); GOTO TIMER;
/* F8 DEC ADD AND BIN ADD GROUPS   */
   EX(4,137): CALL FETCH(C,DC);DC=MOD(DC+1,65536); GOTO PDAD;
   EX(4,208): PT=0 ; GOTO PDAD;
   EX(4,209): PT=1 ; GOTO PDAD;
   EX(4,210): PT=2 ; GOTO PDAD;
   EX(4,211): PT=3 ; GOTO PDAD;
   EX(4,212): PT=4 ; GOTO PDAD;
   EX(4,213): PT=5 ; GOTO PDAD;
   EX(4,214): PT=6 ; GOTO PDAD;
   EX(4,215): PT=7 ; GOTO PDAD;
   EX(4,216): PT=8 ; GOTO PDAD;
   EX(4,217): PT=9 ; GOTO PDAD;
   EX(4,218): PT=10; GOTO PDAD;
   EX(4,219): PT=11; GOTO PDAD;
   EX(4,220): PT=ISAR; GOTO PDAD;
   EX(4,221): PT=ISAR; CALL DNK(1,ISAR); GOTO PDAD;
   EX(4,222): PT=ISAR; CALL DNK(-1,ISAR); GOTO PDAD;
   PDAD: C=SCF8(PT);
   FDAD: CALL ADD(A,A,C,'0'B,CY,Z,S,P,AC,V);
     IF(¬AC) THEN CALL ADD(A,A,250,'0'B,CYT,Z,S,P,AC,V);
     IF(¬CY) THEN CALL ADD(A,A,160,'0'B,CYT,Z,S,P,AC,V);
     S=¬S;GOTO TIMER;
   EX(4,36): CALL FETCH(C,PC+1);GOTO BAD;
   EX(4,136): CALL FETCH(C,DC); DC=MOD(DC+1,65536); GOTO BAD;
   EX(4,192): PT=0 ; GOTO PBAD;
   EX(4,193): PT=1 ; GOTO PBAD;
   EX(4,194): PT=2 ; GOTO PBAD;
   EX(4,195): PT=3 ; GOTO PBAD;
   EX(4,196): PT=4 ; GOTO PBAD;
   EX(4,197): PT=5 ; GOTO PBAD;
   EX(4,198): PT=6 ; GOTO PBAD;
   EX(4,199): PT=7 ; GOTO PBAD;
   EX(4,200): PT=8 ; GOTO PBAD;
   EX(4,201): PT=9 ; GOTO PBAD;
   EX(4,202): PT=10; GOTO PBAD;
   EX(4,203): PT=11; GOTO PBAD;
   EX(4,204): PT=ISAR; GOTO PBAD;
   EX(4,205): PT=ISAR; CALL DNK( 1,ISAR); GOTO PBAD;
   EX(4,206): PT=ISAR; CALL DNK(-1,ISAR); GOTO PBAD;
   PBAD: C=SCF8(PT);
   BAD: CALL ADD(A,A,C,'0'B,CY,Z,S,P,AC,V); S=¬S;
     GOTO TIMER;
/* LR GROUPS FOR F8   */
   EX(4,64): PT=0 ; GOTO LAC;
   EX(4,65): PT=1 ; GOTO LAC;
   EX(4,66): PT=2 ; GOTO LAC;
   EX(4,67): PT=3 ; GOTO LAC;
   EX(4,68): PT=4 ; GOTO LAC;
   EX(4,69): PT=5 ; GOTO LAC;
   EX(4,70): PT=6 ; GOTO LAC;
   EX(4,71): PT=7 ; GOTO LAC;
   EX(4,72): PT=8 ; GOTO LAC;
   EX(4,73): PT=9 ; GOTO LAC;
   EX(4,74): PT=10; GOTO LAC;
   EX(4,75): PT=11; GOTO LAC;
   EX(4,76): PT=ISAR; GOTO LAC;
   EX(4,77): PT=ISAR; CALL DNK( 1,ISAR); GOTO LAC;
   EX(4,78): PT=ISAR; CALL DNK(-1,ISAR); GOTO LAC;
   LAC: A=SCF8(PT); GOTO TIMER;
   EX(4,80): PT=0 ; GOTO LOR;
   EX(4,81): PT=1 ; GOTO LOR;
   EX(4,82): PT=2 ; GOTO LOR;
   EX(4,83): PT=3 ; GOTO LOR;
   EX(4,84): PT=4 ; GOTO LOR;
   EX(4,85): PT=5 ; GOTO LOR;
   EX(4,86): PT=6 ; GOTO LOR;
   EX(4,87): PT=7 ; GOTO LOR;
   EX(4,88): PT=8 ; GOTO LOR;
   EX(4,89): PT=9 ; GOTO LOR;
   EX(4,90): PT=10; GOTO LOR;
   EX(4,91): PT=11; GOTO LOR;
   EX(4,92): PT=ISAR; GOTO LOR;
   EX(4,93): PT=ISAR; CALL DNK( 1,ISAR); GOTO LOR;
   EX(4,94): PT=ISAR; CALL DNK(-1,ISAR); GOTO LOR;
   LOR:  SCF8(PT)=A; GOTO TIMER;
/* LISAR GROUPS FOR F8    */
   EX(4,96 ): PT=0; GOTO LISU;
   EX(4,97 ): PT=1; GOTO LISU;
   EX(4,98 ): PT=2; GOTO LISU;
   EX(4,99 ): PT=3; GOTO LISU;
   EX(4,100): PT=4; GOTO LISU;
   EX(4,101): PT=5; GOTO LISU;
   EX(4,102): PT=6; GOTO LISU;
   EX(4,103): PT=7; GOTO LISU;
   LISU: ISAR=MOD(ISAR,8)+8*PT; GOTO TIMER;
   EX(4,104): PT=0; GOTO LISL;
   EX(4,105): PT=1; GOTO LISL;
   EX(4,106): PT=2; GOTO LISL;
   EX(4,107): PT=3; GOTO LISL;
   EX(4,108): PT=4; GOTO LISL;
   EX(4,109): PT=5; GOTO LISL;
   EX(4,110): PT=6; GOTO LISL;
   EX(4,111): PT=7; GOTO LISL;
   LISL: ISAR=ISAR-MOD(ISAR,8)+PT; GOTO TIMER;
/* F8  DS GROUP     */
   EX(4,48): PT=0 ; GOTO DSF;
   EX(4,49): PT=1 ; GOTO DSF;
   EX(4,50): PT=2 ; GOTO DSF;
   EX(4,51): PT=3 ; GOTO DSF;
   EX(4,52): PT=4 ; GOTO DSF;
   EX(4,53): PT=5 ; GOTO DSF;
   EX(4,54): PT=6 ; GOTO DSF;
   EX(4,55): PT=7 ; GOTO DSF;
   EX(4,56): PT=8 ; GOTO DSF;
   EX(4,57): PT=9 ; GOTO DSF;
   EX(4,58): PT=10; GOTO DSF;
   EX(4,59): PT=11; GOTO DSF;
   EX(4,60): PT=ISAR; GOTO DSF;
   EX(4,61): PT=ISAR; CALL DNK( 1,ISAR); GOTO DSF;
   EX(4,62): PT=ISAR; CALL DNK(-1,ISAR); GOTO DSF;
   DSF: C=SCF8(PT); CALL ADD(C,C,255,'0'B,CY,Z,S,P,AC,V); S=¬S;
     SCF8(PT)=C; GOTO TIMER;
/* F8 NI NM NS  XI XM XS GROUP   */
   EX(4,33): CALL FETCH(C,PC+1); GOTO NGP;
   EX(4,138): CALL FETCH(C,DC); DC=MOD(DC+1,65536); GOTO NGP;
   EX(4,240): PT=0 ; GOTO PNGP;
   EX(4,241): PT=1 ; GOTO PNGP;
   EX(4,242): PT=2 ; GOTO PNGP;
   EX(4,243): PT=3 ; GOTO PNGP;
   EX(4,244): PT=4 ; GOTO PNGP;
   EX(4,245): PT=5 ; GOTO PNGP;
   EX(4,246): PT=6 ; GOTO PNGP;
   EX(4,247): PT=7 ; GOTO PNGP;
   EX(4,248): PT=8 ; GOTO PNGP;
   EX(4,249): PT=9 ; GOTO PNGP;
   EX(4,250): PT=10; GOTO PNGP;
   EX(4,251): PT=11; GOTO PNGP;
   EX(4,252): PT=ISAR; GOTO PNGP:
   EX(4,253): PT=ISAR; CALL DNK( 1,ISAR); GOTO PNGP;
   EX(4,254): PT=ISAR; CALL DNK(-1,ISAR); GOTO PNGP;
   PNGP: C=SCF8(PT);
   NGP: A=BOOL(A,C,'0001'); V='0'B; CY='0'B;
     IF(A=0) THEN Z='1'B; ELSE Z='0'B;
   IF(A>127) THEN S='0'B; ELSE S='1'B; GOTO TIMER;
   EX(4,35): CALL FETCH(C,PC+1); GOTO XGP;
   EX(4,140): CALL FETCH(C,DC); DC=MOD(DC+1,65536); GOTO XGP;
   EX(4,224): PT=0 ; GOTO PXGP;
   EX(4,225): PT=1 ; GOTO PXGP;
   EX(4,226): PT=2 ; GOTO PXGP;
   EX(4,227): PT=3 ; GOTO PXGP;
   EX(4,228): PT=4 ; GOTO PXGP;
   EX(4,229): PT=5 ; GOTO PXGP;
   EX(4,230): PT=6 ; GOTO PXGP;
   EX(4,231): PT=7 ; GOTO PXGP;
   EX(4,232): PT=8 ; GOTO PXGP;
   EX(4,233): PT=9 ; GOTO PXGP;
   EX(4,234): PT=10; GOTO PXGP;
   EX(4,235): PT=11; GOTO PXGP;
   EX(4,236): PT=ISAR; GOTO PXGP;
   EX(4,237): PT=ISAR; CALL DNK( 1,ISAR); GOTO PXGP;
   EX(4,238): PT=ISAR; CALL DNK(-1,ISAR); GOTO PXGP;
   PXGP: C=SCF8(PT);
   XGP: A=BOOL(A,C,'0110'); V='0'B; CY='0'B;
     IF(A=0) THEN Z='1'B; ELSE Z='0'B;
   IF(A>127) THEN S='0'B; ELSE S='1'B; GOTO TIMER;
/* F8  PI JMP DCI XDC ADC */
   EX(4,40): PC1=PCN;    GOTO TIMER;
   EX(4,41): CALL FETCH(C,PC+1); PCN=256*C;
     CALL FETCH(C,PC+2); PCN=PCN+C; GOTO TIMER;
   EX(4,42): CALL FETCH(C,PC+1); DC=256*C; CALL FETCH(C,PC+2);
     DC=DC+C; GOTO TIMER;
   EX(4,44): K=DC1; DC1=DC; DC=K; GOTO TIMER;
   EX(4,142): IF(A>127) THEN C=A-256; ELSE C=A; DC=DC+C;
     IF(DC<0) THEN DC=65536+DC; GOTO TIMER;
/* FB OR GROUP  CMP GROUP   */
   EX(4,34): CALL FETCH(C,PC+1); GOTO ORGF;
   EX(4,139): CALL FETCH(C,DC); DC=MOD(DC+1,65536); GOTO ORGF;
   ORGF: A=BOOL(A,C,'0111'); V='0'B; CY='0'B;
     IF(A=0) THEN Z='1'B; ELSE Z='0'B;
     IF(A>127) THEN S='0'B; ELSE S='1'B; GOTO TIMER;
   EX(4,37): CALL FETCH(C,PC+1); GOTO CMPF;
   EX(4,141): CALL FETCH(C,DC); DC=MOD(DC+1,65536); GOTO CMPF;
   CMPF: CALL ADD(C,C,(255-A),'1'B,CY,Z,S,P,AC,V);S=¬S; GOTO TIMER;
TIMER: ITIME=ITIME+TIMINC(MACH,INS);
   IF(DELTSW) THEN DO;
     ITIME=ITIME+DELTIM(MACH); DELTSW='0'B; END;
   GOTO TRACE;
TRACE: IF(INHIB) THEN GOTO TR2;
       IF(BRSW) THEN IF(¬INSW&¬OUTSW) THEN IF(PCN=PC+DELPC(MACH,INS))
     THEN GOTO NEWINS;
       IF(IOSW&¬INSW&¬OUTSW) THEN GOTO NEWINS;
   TR2: LINE=LINE+1;
     PUT EDIT(ITIME)(F(8))SKIP;
   CALL TOHEX(PC,CDUMY); PUT EDIT('  ',CDUMY,':')(A(2),A(4),A(1));
   DO I=1 TO 3;
   IF(I<=DELPC(MACH,INS))THEN DO;
     M=PC+I-1; CALL FETCH(TEMPA,M);K=TEMPA;CALL TOHEX(K,CDUMY);
     CDFT=CDBK; PUT EDIT(CDUMY)(A(4));END;
     ELSE PUT EDIT('  __')(A(4));
   END;
   IF(MACH=1|MACH=2)THEN DO;
     K=A; CALL TOHEX(K,CDUMY);CDFT=CDBK;PUT EDIT(CSD)(A(3));
```

```
  K=B; CALL TOHEX(K,CDUMY);CDFT=CDBK;PUT EDIT(CSD)(A(3));
  K=C; CALL TOHEX(K,CDUMY);CDFT=CDBK;PUT EDIT(CSD)(A(3));
  K=D; CALL TOHEX(K,CDUMY);CDFT=CDBK;PUT EDIT(CSD)(A(3));
  K=E; CALL TOHEX(K,CDUMY);CDFT=CDBK;PUT EDIT(CSD)(A(3));
  K=H; CALL TOHEX(K,CDUMY);CDFT=CDBK;PUT EDIT(CSD)(A(3));
  K=L; CALL TOHEX(K,CDUMY);CDFT=CDBK;PUT EDIT(CSD)(A(3));
END;
I=(MACH=1) THEN DO;
  TEMPA='0000'B||S||Z||AC||CY;
  M=TEMPA; CALL TOHEX(M,CDUMY);CDFT=CDBK;
  PUT EDIT(CDUMY)(A(4));PUT EDIT('   ',SP08)(A(3),F(1));END;
I=(MACH=2) THEN DO;
  TEMPA=S||Z||'0'B||AC||'0'B||P||'1'B||CY; M=TEMPA;
  CALL TOHEX(M,CDUMY);CDFT=CDBK;
PUT EDIT(CDUMY)(A(4));  CALL TOHEX(SP,CDUMY);
  PUT EDIT('  ',CDUMY)(A(2),A(4)); END                  ;
I=(MACH=3) THEN DO;
  M=A; CALL TOHEX(M,CDUMY);CDFT=CDBK; PUT EDIT(CSD)(A(3));
  M=B; CALL TOHEX(M,CDUMY);CDFT=CDBK; PUT EDIT(CSD)(A(3));
  CALL TOHEX(INDX,CDUMY); PUT EDIT('  ',CDUMY)(A(2),A(4));
  TEMPA='11'B||AC||I68||S||Z||V||CY;  M=TEMPA; CALL TOHEX(M,CDUMY);
  CDFT=CDBK; PUT EDIT(CDUMY)(A(4));
  CALL TOHEX(SP,CDUMY); PUT EDIT('  ',CDUMY)(A(2),A(4));
END;
I=(MACH=4) THEN DO;
  K=A; CALL TOHEX(K,CDUMY); CDFT=CDBK;
  PUT EDIT(CDUMY)(X(2),A(4));
  CALL TOHEX(PC1,CDUMY); PUT EDIT(CDUMY)(X(2),A(4));
  CALL TOHEX(DC,CDUMY); PUT EDIT(CDUMY)(X(2),A(4));
  CALL TOHEX(DC1,CDUMY); PUT EDIT(CDUMY)(X(2),A(4));
  C='000'B||ICB||V||Z||CY||S; K=C; CALL TOHEX(K,CDUMY);
CDFT=CDBK; PUT EDIT(CDUMY)(X(2),A(4));
  K=ISAR; CALL TOHEX(K,CDUMY); CDFT=CDBK;
  PUT EDIT(CDUMY)(X(2),A(4));
END;
```

```
   IF(FIFOSW) THEN IF(OUTSW) THEN IF(PORT=FI1|PORT=FI2) THEN DO;
     IF(PORT=FI1) THEN I=1; ELSE I=2; TEMPA=OUT;
     CALL FIFO(I,'1'B,TEMPA); END;
   PUT EDIT(MNE(MACH,INS))(X(2),A(8),X(2));
   IF(INSW) THEN DO;       INSW='0'B;
     CALL TOHEX(IPORT,PORTCH); CALL TOHEX(INVAL,OUTCH);
     OUTCH='  '||SUBSTR(OUTCH,3,2); PUT EDIT(' IN PORT ',PORTCH,
     '=>',OUTCH) (A(9),A(4),A(2),A(4));END;
   IF(OUTSW) THEN DO; CALL TOHEX(PORT,PORTCH); OUTSW='0'B;
     CALL TOHEX(OUT,OUTCH); OUTCH='  '||SUBSTR(OUTCH,3,2);
     PUT EDIT('  OUT PORT ',PORTCH,'=>',OUTCH)
       (A(11),A(4),A(2),A(4)); END;
   IF(FISW) THEN PUT EDIT('FIFO',FINU)(X(2),A(4),F(1));
   FISW='0'B;
   IF(MACH=4) THEN IF(MOD(LINE,32)=0) THEN DO;
       CALL SCRATCH; IF(SCRSW) THEN LINE=LINE+8; END;
   IF(LINE>112) THEN DO; CALL HEADINGS; LINE=0; END;
   GOTO NEWINS;
FETCH: PROCEDURE(VALUE,ADRES);
     DCL(FO1,FO2) BINARY FIXED(31) EXTERNAL;
     DCL(MI,MA) BINARY FIXED(31);
     DCL ADRES BINARY FIXED(31);
     DCL MEMSIZ BINARY FIXED(31) EXTERNAL;
     DCL IPORT BINARY FIXED(31) EXTERNAL;
     DCL INVAL BINARY FIXED (31) EXTERNAL;
     DCL PORTID(32) BINARY FIXED(31) EXTERNAL;
     DCL PSTART(32) BINARY FIXED(31) EXTERNAL;
     DCL PEND(0:32) BINARY FIXED(31) EXTERNAL;
     DCL MEM(4,16384) BINARY FIXED(8) EXTERNAL;
     DCL MACH BINARY FIXED EXTERNAL;
     DCL I BINARY FIXED;
     DCL VALUE BINARY FIXED(8);
     DCL CHD CHAR(4);
     DCL INHIB  BIT(1) EXTERNAL;
     DCL FIFOSW BIT(1) EXTERNAL;
```

```
      DCL IOOVSW BIT(1) EXTERNAL;
      DCL INSW BIT(1) EXTERNAL;
      DCL IDATA(4096 ) BINARY FIXED(8) EXTERNAL;
    IF(MACH=1) THEN IF(ADRES>16383) THEN DO;
      PUT LIST('**** ADDRESS BEYOND THE SIZE CAPABILITY OF 8008')SKIP;
      PUT LIST('     MAXIMUM IS HEX 3FFF')SKIP;
      PUT LIST(' _')SKIP;
      INHIB='1'B;
      VALUE= 0; RETURN;
      END;
    IF(FIFOSW&¬IOOVSW) THEN
      IF(MACH=3) THEN
        IF(ADRES=F01|ADRES=F02) THEN DO;
      IF(ADRES=F01) THEN I=1; ELSE I=2; CALL FIFO(I,'0'B,VALUE) ;
      INSW='1'B; INVAL=VALUE; IPORT=ADRES;
      RETURN; END;
    IF(¬IOOVSW&MACH=3) THEN DO I = 1 TO 32;
      IF(PORTID(I)<0)THEN GOTO MEMV;
      IF(ADRES=PORTID(I)) THEN DO;
        IF(PSTART(I)>PEND(I))THEN DO;
          CALL TOHEX(ADRES,CHD);
           PUT LIST('**** PORT ',CHD,' LIST EXHAUSTED') SKIP;
          INHIB='1'B; RETURN;END;
      INSW='1'B; IPORT=ADRES; K=PSTART(I); VALUE=IDATA(K);
      INVAL=VALUE; PSTART(I)=PSTART(I)+1; RETURN; END;
      END;
    MEMV: IF(ADRES>MEMSIZ|ADRES<0) THEN DO; PUT LIST
      ('****ADDRESS OUTSIDE OF MEMORY SPACE ',ADRES)SKIP;
    PUT LIST('_')SKIP;
      VALUE=0; INHIB='1'B; RETURN;
        END;
    MA=ADRES+1;MI=MA/16384+1;MA=MOD(MA,16384);
    VALUE=MEM(MI,MA);
END FETCH;
STORE: PROCEDURE(VALUE,ADRES);
```

```
  DCL FIFOSW BIT(1) EXTERNAL;
  DCL (FI1,FI2) BINARY FIXED(31) EXTERNAL;
  DCL LINE BINARY FIXED(31) EXTERNAL;
  DCL ADRES BINARY FIXED(31);
  DCL MEMSIZ BINARY FIXED(31) EXTERNAL;
  DCL (ROMST,ROMEND) BINARY FIXED(31) EXTERNAL;
  DCL PORTLT(32) BINARY FIXED(31) EXTERNAL;
  DCL (PORT,OUT) BINARY FIXED(31) EXTERNAL;
  DCL(MI,MA) BINARY FIXED(31);
  DCL MACH BINARY FIXED EXTERNAL;
  DCL I BINARY FIXED;
  DCL VALUE BINARY FIXED(8);
  DCL MEM(4,16384) BINARY FIXED(8) EXTERNAL;
  DCL INHIB  BIT(1) EXTERNAL;
  DCL OUTSW BIT(1) EXTERNAL;
  DCL MEMSW BIT(1) EXTERNAL;
  DCL ROMSW BIT(1) EXTERNAL;
IF(FIFOSW) THEN IF(¬MEMSW) THEN IF(MACH=3)
  THEN IF(ADRES=FI1|ADRES=FI2) THEN DO;
  OUTSW='1'B; OUT=VALUE; PORT=ADRES; RETURN; END;
IF(¬MEMSW) THEN IF(MACH=1) THEN IF(ADRES>16383) THEN DO;
  PUT LIST('**** ADDRESS BEYOND THE SIZE CAPABILITY OF 8008')SKIP;
  PUT LIST('    MAXIMUM IS HEX 3FFF')SKIP;
  PUT LIST(' _')SKIP;
  INHIB='1'B;
  RETURN;
  END;
IF(ROMSW&¬MEMSW) THEN IF(ADRES>=ROMST&ADRES<=ROMEND) THEN DO;
  PUT LIST('**WRITE TO LOCATION DESIGNATED AS ROM IGNORED')SKIP;
  LINE=LINE+1;
  IF(LINE>112) THEN DO; CALL HEADINGS; LINE=0; END;
    RETURN; END;
IF(MACH=3&¬MEMSW) THEN DO I = 1 TO 32; IF (PORTLT(I)=-1)THEN
    GOTO MFD;
  IF(PORTLT(I)=ADRES) THEN DO; OUTSW='1'B; OUT=VALUE;PORT=ADRES;
```

```
       RETURN; END;
     END;
   MED: OUTSW='0'B;
   IF(ADRES>MEMSIZ|ADRES<0)THEN DO; PUT LIST
     ('****ADDRESS OUTSIDE OF MEMORYS SPACE ',ADRES)SKIP;
       INHIB='1'B; RETURN;
       END;
   MA=ADRES+1;MI=MA/16384+1;MA=MOD(MA,16384);
   VALUE=MOD(VALUE,256);
   MEM(MI,MA)=VALUE;
   RETURN;
END STORE;
DUMP: PROCEDURE(START,END);
     DCL MEMSIZ BINARY FIXED(31) EXTERNAL;
     DCL FENCE BINARY FIXED(31) EXTERNAL;
     DCL(START,END) BINARY FIXED(31);
     DCL MACH BINARY FIXED EXTERNAL;
     DCL TEMP BINARY FIXED(31);
     DCL SWCT  BINARY   FIXED EXTERNAL;
     DCL(ROMST,ROMEND) BINARY FIXED(31) EXTERNAL;
     DCL ROMSW BIT(1) EXTERNAL;
     DCL IOOVSW BIT(1) EXTERNAL;
     DCL FENSW BIT(1) EXTERNAL;
     DCL HEXVAL CHAR(4);
     DCL HEXC1 CHAR(1) DEF HEXVAL POSITION(1);
     DCL HEXFT CHAR(2) DEF HEXVAL POSITION(1);
     DCL DUPP BINARY FIXED(31);
     DCL TEMPE BINARY FIXED(8);
   DUPP=START;
   IF(END>MEMSIZ) THEN END=MEMSIZ;
   IF(MACH=4) THEN CALL SCRATCH;
   IOOVSW='1'B;
   PUT LIST(' ** LISTING OF MEMORY',SWCT, ' FOLLOWS:')PAGE;
   NEWLINE: CALL TOHEX(DUPP,HEXVAL);PUT EDIT(HEXVAL,':')(A(4),A(1))
     SKIP;
   NEWVAL: CALL FETCH(TEMPE,DUPP); TEMP=TEMPE; CALL TOHEX(TEMP,HEXVAL);
     HEXFT='  '; IF(ROMSW)THEN IF(DUPP>=ROMST&DUPP<=ROMEND) THEN
     HEXFT=' *';
   IF(FENSW) THEN IF(DUPP=FENCE) THEN HEXC1='|';
   PUT EDIT(HEXVAL)(A(4)); DUPP=DUPP+1;
     IF(DUPP>END) THEN DO;
   IOOVSW='0'B;
       PUT LIST(' _')SKIP(2); RETURN; END;
   IF(MOD(DUPP,16)=0) THEN GOTO NEWLINE;
   GOTO NEWVAL;
END DUMP;
HEADINGS: PROCEDURE;
     DCL MACH BINARY FIXED EXTERNAL;
   IF(MACH=1) THEN PUT LIST(' INTEL 8008 SIMULATION  '
     ,'     FLAGS ARE 0 0 0 0 S Z P CY ')PAGE;
   IF(MACH=2) THEN PUT LIST(' INTEL 8080 SIMULATION  ',
     '     FLAGS ARE S Z 0 AC 0 P 1 CY  ')PAGE;
   IF (MACH=1|MACH=2 ) THEN
     PUT EDIT
('    CLOCK   PC    INSTRUCTION   A   B   C   D   E   H   L    FGS   SP')(A(57))
     SKIP(2);
   IF(MACH=3) THEN DO;    PUT LIST(' MOTOROLA 6800 SIMULATION',
     '    FLAGS ARE 1 1 H I N Z V C')PAGE ;
     PUT EDIT
('    CLOCK   PC    INSTRUCTION   A   B    INDX   FGS  SP')(A(47))SKIP(2);
   END;
   IF(MACH=4) THEN DO;
     PUT LIST(' FAIRCHILD F8 SIMULATION          ',
       ' FLAGS ARE 0 0 0 ICB OV Z CRY +') PAGE;
     PUT EDIT
('    CLOCK   PC    INSTRUCTION      A    PC1    DC     DC1      FGS    ISAR')
     (A(65))SKIP(2);
   END;
   PUT LIST('_')SKIP;
END HEADINGS;
```

```
SCRATCH: PROCEDURE;
     DCL TEMP BINARY FIXED(31);
     DCL MACH BINARY FIXED EXTERNAL;
     DCL DUPP BINARY FIXED(31);
     DCL SCRSW BIT(1) EXTERNAL ;
     DCL I BINARY FIXED;
     DCL HEXVAL CHAR(4);
     DCL HEXFT CHAR(2) DEF HEXVAL  POSITION(1);
     DCL SCF8(0:63) BINARY FIXED(8) EXTERNAL ;
   IF(¬SCRSW) THEN RETURN;
   IF(MACH=4) THEN DO;
     PUT LIST(' DUMP OF F8 SCRATCH FOLLOWS:') SKIP(2);
     DUPP=0;
   NEWROW: I=0;
     CALL TOHEX(DUPP,HEXVAL); HEXFT='  ';
     PUT EDIT(HEXVAL,':')(X(1),A(4),A(1))SKIP;
   NEWONE: TEMP=SCF8(DUPP); CALL TOHEX(TEMP,HEXVAL); HEXFT='  ';
     PUT EDIT( HEXVAL,',')(A(4),A(1)); DUPP=DUPP+1;
     IF(DUPP>63) THEN GOTO LAST;
     I=I+1; IF(I>15) THEN GOTO NEWROW;
     GOTO NEWONE;
   LAST: PUT LIST(' _')SKIP(2);
   END;
END SCRATCH;
HEXIN: PROCEDURE (VALUE,TERMIN      ,COL);
  /* INPUTS HEXIDECIMAL CODED VALUES AS BINARY VALUES */
  /* FOR FIRST CALL COL SHOULD BE >=80       */
     DCL CARD CHAR(80) EXTERNAL;
     DCL HEXLIS CHAR(20) INITIAL('0123456789ABCDEF,;:.') STATIC;
     DCL TERMIN CHAR(1);
     DCL VALUE BINARY FIXED(31);
     DCL (COL,TEMP) BINARY FIXED;
     DCL INHIB BIT(1) EXTERNAL;
     DCL NULSW BIT(1) EXTERNAL;
   ON ENDFILE BEGIN;
```

```
      PUT LIST('****TERMINATES LOOKING FOR MORE HEX INPUT')
                SKIP;STOP; END;
   VALUE=0;
  SCAN: COL=COL+1; IF(COL>80) THEN    DO; COL=1;     NULSW='0'B;
                                        GET EDIT (CARD)(A(80));
                               PUT EDIT (CARD)(X( 9),A(80))SKIP;
                                        END;
   TERMIN=SUBSTR(CARD,COL,1);
   IF(TERMIN=' ') THEN GOTO SCAN;
   TEMP=INDEX(HEXLIS,TERMIN);
   IF(TEMP=0) THEN DO; PUT LIST(' **** INVALID HEX  CHARACTER',TERMIN)
     SKIP; STOP; END;
   IF(TEMP=18&¬NULSW) THEN DO COL=80; GOTO SCAN; END; ELSE NULSW='1'B;
   IF(TEMP>16) THEN RETURN;
   VALUE=VALUE*16+TEMP-1;
   IF(VALUE>65536) THEN DO;
     PUT LIST('**** HEX VALUE > 10000 TOO BIG   CAUSES TERMINATION')
     SKIP; STOP; END;
   GOTO SCAN;
END HEXIN;
TOHEX:PROCEDURE (VALUE,HEX);
     DCL(VALUE,VALU1,   REM)BINARY FIXED(31);
     DCL I BINARY FIXED;
     DCL HIXLIS CHAR(16) INITIAL('0123456789ABCDEF') STATIC;
     DCL HEX CHAR(4);
     DCL HEXVAL CHAR(4);
     DCL HXCHA(4) CHAR(1) DEF HEXVAL;
   VALU1=VALUE;
   IF(VALU1<0) THEN DO; HEXVAL='####'; RETURN; END;
   DO I=1 TO 4;
     REM=MOD(VALU1,16); VALU1=(VALU1-REM)/16; REM=REM+1;
     HXCHA(5-I)=SUBSTR(HIXLIS,REM,1);
   END;
   HEX=HEXVAL; RETURN;
END TOHEX;
```

```
FIFO: PROCEDURE ( NUMBER,TOFROM,VALUE);
     DCL INHIB BIT(1) EXTERNAL;
     DCL FIFS(2,2048) EXTERNAL BINARY FIXED(8) ;
     DCL FINU BINARY FIXED EXTERNAL;
     DCL FISW BIT(1) EXTERNAL;
     DCL FIP(2) BINARY FIXED EXTERNAL;
     DCL FOP(2) BINARY FIXED EXTERNAL;
     DCL NUMBER BINARY FIXED;
     DCL TOFROM BIT(1);
     DCL VALUE BINARY FIXED(8);
     DCL TEMP BINARY FIXED;
     FISW='1'B; FINU=NUMBER;
   IF(TOFROM) THEN DO;FIP(NUMBER)=FIP(NUMBER)+1; TEMP=FIP(NUMBER);
      IF(TEMP>2048)THEN DO; PUT LIST('****FIFO OVERFLOWS',NUMBER)SKIP;
   INHIB='1'B;
       RETURN; END;
     FIFS(NUMBER,TEMP)=VALUE; RETURN;
     END;
   ELSE DO; FOP(NUMBER)=FOP(NUMBER)+1; TEMP=FOP(NUMBER);
     IF(TEMP>FIP(NUMBER)) THEN DO; VALUE=0;
   INHIB='1'B;
       PUT LIST('****FIFO UNDERFLOWS',NUMBER)SKIP; RETURN;END;
     IF(TEMP>2048) THEN DO;
   INHIB='1'B;
       PUT LIST('****FIFO STILL OVERFLOWED BUFFER',NUMBER)SKIP;
       VALUE=0; RETURN;END;
     VALUE=FIFS(NUMBER,TEMP); RETURN; END;
END FIFO;
ADD: PROCEDURE (AD,AS1,AS2,CI,CO,Z,S,P,AC,V);
   /*  TO USE WITH 08 OR 80                     */
   /* FOR ADD  CALL WITH A,A,SOURCE,CI=0,CO=C   */
   /* FOR ADC  CALL WITH A,A,SOURCE,CI=C,CO=C   */
   /* FOR SUB CALL WITH A,A,(255-SOURCE),CI=1,CO=C */
   /* FOR SBB CALL WITH A,A,(255-SOURCE),CI=C,CO=C */
   /* FOR CMP CALL WITH DUMMY,A,(255-SOURCE),CI=1,CO=C */
```

```
    DCL (CI,CO,Z,S,P,AC,V) BIT(1);
    DCL (AD,AS1,AS2) BINARY FIXED(8);
    DCL PARTAB BIT(256) EXTERNAL;
    DCL CD BINARY FIXED;
  CD=0; IF(CI)  THEN CD=1;
  IF(MOD(AS2,16)+MOD(AS1,16)+CD>15) THEN AC='1'B;
    ELSE AC='0'B;
  CD=AS1+AS2+CD; /*SUM WITH CARRY*/
  IF (CD>255) THEN CO='1'B;
    ELSE CO='0'B;
  CD=MOD(CD,256);
  CD=MOD(CD,256);
  IF(CD>127&AS1<128&AS2<128|CD<128&AS1>127&AS2>127)THEN V='1'B;
    ELSE V='0'B;
  AD=CD;
  IF(AD=0) THEN Z='1'B; ELSE Z='0'B;
  IF(AD>127) THEN S='1'B; ELSE S='0'B;
  CD=CD+1;
  P=SUBSTR(PARTAB,CD,1);
END ADD;
LFLAGS: PROCEDURE(R,Z,S,P,VALUE);
  /* FOR USE WITH 08 OR 80 TO SET FLAGS AFTER LCGICOP     */
    DCL VALUE BINARY FIXED(8);
    DCL CD BINARY FIXED;
    DCL (R,Z,S,P) BIT(1);
    DCL PARTAB BIT(256) EXTERNAL;
  R='0'B;
  IF(VALUE=0) THEN Z='1'B; ELSE Z='0'B;
  IF(VALUE>127) THEN S='1'B; ELSE S='0'B;
  CD=VALUE+1;
  P=SUBSTR(PARTAB,CD,1);
END LFLAGS;
TELIN: PROCEDURE(VALUE,PORT);
    DCL IDATA(4096 ) BINARY FIXED(8) EXTERNAL;
    DCL PORTIO(32 ) BINARY FIXED(31) EXTERNAL;
```

```
     DCL PSTART(32 ) BINARY FIXED(31) EXTERNAL;
     DCL PEND(0:32 ) BINARY FIXED(31) EXTERNAL;
     DCL INVAL BINARY FIXED (31) EXTERNAL;
     DCL IPORT BINARY FIXED(31) EXTERNAL;
     DCL(FO1,FO2) BINARY FIXED(31) EXTERNAL;
     DCL I BINARY FIXED;
     DCL PORT BINARY FIXED;
     DCL J BINARY FIXED;
     DCL VALUE BINARY FIXED(8);
     DCL PORCH CHAR (4);
     DCL INSW BIT(1) EXTERNAL;
     DCL FIFOSW BIT(1) EXTERNAL;
     DCL INHIB BIT(1) EXTERNAL;
   IF(FIFOSW)THEN IF(PORT=FO1|PORT=FO2) THEN DO;
     IF(PORT=FO1) THEN I=1; ELSE I=2; CALL FIFO(I,'0'B,VALUE);
     IPORT=PORT; INVAL=VALUE; INSW='I'B;
     RETURN; END;
   INSW='1'B; IPORT=PORT;
     CALL TOHEX(IPORT,PORCH);
   DO I=I TO 32 ;
   IF(PORTID(I)=-1) THEN GOTO NOPOR;
   IF(PORTID(I)=PORT) THEN DO;
     IF(PSTART(I)>PEND(I)) THEN DO;
       PUT LIST('**** INPUT PORT',PORCH,'  LIST EXHAUSTED')SKIP;
       INHIB='1'B; PETURN;END;
     J=PSTART(I);VALUE=IDATA(J);INVAL=VALUE;PSTAPT(I)=PSTART(I)+1;
     RETURN;END;
   NOPOR: PUT LIST('****ACCESS TO NON-DEFINED INPUT PORT LIST',PORCH)
     SKIP; INHIB='1'B; RETURN;
END TELIN;
PUSH:PPOCEDURE(PRE,POST,VALUE,STKP);
     DCL LINE BINARY FIXED(31) EXTERNAL;
     DCL STKP BINARY FIXED(31);
     DCL FENCE BINARY FIXED(31) EXTERNAL       ;
     DCL (PPE,POST) BINARY FIXED;
     DCL VALUE BINARY FIXED(8);
     DCL FENSW BIT(1) EXTERNAL ;
   STKP=STKP-PRE ; IF(STKP<0)THEN STKP=65535;
   IF(MACH=2) THEN
     IF(STKP>=0&STKP<=63) THEN DO;
     PUT LIST('**STACK INTO RST AREA')SKIP; LINE=LINE+1;
       IF(LINE>112) THEN DO; CALL HEADINGS; LINE=0; END; END;
   IF(FENSW) THEN DO;
   IF(STKP<FENCE+8&STKP>=FENCE)THEN DO;
     PUT LIST('**STACK PUSH INTO FENCE WARNING ZONE')SKIP;
     LINE=LINE+1; IF(LINE>112) THEN DO; CALL HEADINGS; LINE=0; END;END;
     IF(STKP<FENCE) THEN DO; PUT LIST
       ('****STACK PUSH TO BELOW FENCE') SKIP; INHIB='1'B; END;
   END;
   CALL STORE(VALUE,STKP);
   STKP=STKP-POST; IF(STKP<0) THEN STKP=65535;
END PUSH;
POP: PROCEDURE(PRE,POST,VALUE,STKP);
     DCL LINE BINARY FIXED(31) EXTERNAL;
     DCL (PRE,POST) BINARY FIXED;
     DCL VALUE BINARY FIXED(8);
     DCL STKP BINARY FIXED(31);
   STKP=STKP+PRE; IF(STKP>65535) THEN STKP=0;
   IF(MACH=2) THEN
     IF(STKP>=0&STKP<=63) THEN DO;
     PUT LIST('**STACK FROM RST AREA')SKIP; LINE=LINE+1;
       IF(LINE>112) THEN DO; CALL HEADINGS; LINE=0; END;END;
   CALL FETCH(VALUE,STKP);
   STKP=STKP+POST; IF(STKP>65535) THEN STKP=0;
END POP;
DNK: PROCEDURE(DEL,ISAR);
     DCL(DEL,ISAR) BINARY FIXED;
     DCL TEMP BINARY FIXED  ;
   TEMP=MOD(ISAR,8);
   ISAR=ISAR-TEMP;
   TEMP=TEMP+DEL; IF(TEMP<0)THEN TEMP=7; TEMP=MOD(TEMP,8);
   ISAR=ISAR+TEMP;
   RETURN;
END DNK;
DECIN: PROCEDURE(VALUE,TERMIN,COL);
  /*INPUTS DECIMAL CODED VALUES TO BINARY INTEGERS*/
     DCL CARD CHAR(80) EXTERNAL;
     DCL DECLIS CHAR(14) INITIAL('0123456789,;:.') STATIC;
     DCL TERMIN CHAR(1);
     DCL VALUE BINARY FIXED(31);
     DCL (COL,TEMP) BINARY FIXED;
     DCL INHIB BIT(1) EXTERNAL;
     DCL NULSW BIT(1) EXTERNAL;
   ON ENDFILE BEGIN;
      PUT LIST(' ****TERMINATES LOOKING FOR MORE DECIMAL INPUT')
         SKIP;STOP; END;
   VALUE=0;
  SCAND: COL=COL+1; IF(COL>80) THEN DO; COL=I; NULSW='0'B;
            GET EDIT(CARD)(A(80));
            PUT EDIT(CARD)(X(9),A(80)) SKIP; END;
   TERMIN=SUBSTR(CARD,COL,1);
   IF(TERMIN=' ') THEN GOTO SCAND;
   TEMP=INDEX(DECLIS,TERMIN);
   IF(TEMP=0) THEN DO; PUT LIST(' ****INVALID DECIMAL CHARACTER'
       ,TERMIN) SKIP; STOP; END;
   IF(TEMP=12&¬NULSW) THEN DO; COL=80; GOTO SCAND; END;
         ELSE NULSW='1'B;
   IF(TEMP>10) THEN RETURN;
   VALUE=VALUE*10+TEMP-1;
   GOTO SCAND;
END DECIN;
END MP8SIM;
```