N73-21194

SPACE ULTRARELIABLE MODULAR COMPUTER (SUMC) INSTRUCTION SIMULATOR: FINAL REPORT

By

R. T. Curran

W. A. Hornfeck

Prepared for

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION

George C. Marshall Space Flight Center

Marshall Space Flight Center, Alabama

CONTRACT NAS8-26698

November 1972

CSC

COMPUTER SCIENCES CORPORATION

SPACE ULTRARELIABLE MODULAR COMPUTER (SUMC) INSTRUCTION SIMULATOR: FINAL REPORT

by

R. T. Curran
W. A. Hornfeck

Prepared under

Contract NAS8-26698

Prepared for

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION
GEORGE C. MARSHALL SPACE FLIGHT CENTER
MARSHALL SPACE FLIGHT CENTER, ALABAMA 35812

COMPUTER SCIENCES CORPORATION
8300 South Whitesburg Drive
Huntsville, Alabama 35802

TECHNICAL REPORT STANDARD TITLE PAGE

1. REPORT NO.	2. GOVERNMENT ACCESSION NO.	3. RECIPIENT'S CATALOG NO.
4. TITLE AND SUBTITLE Space Ultrareliable Modular Computer (SUMC) Instruction Simulator: Final Report		5. REPORT DATE Nov. 25, 1972 6. PERFORMING ORGANIZATION CODE
7. AUTHOR(S) R. T. Curran and W. A. Hornfeck		8. PERFORMING ORGANIZATION REPORT #
9. PERFORMING ORGANIZATION NAME AND ADDRESS Computer Sciences Corporation Field Services Division, Aerospace Systems Center 8300 South Whitesburg Drive Huntsville, Alabama 35802		10. WORK UNIT NO. 11. CONTRACT OR GRANT NO. NAS8-26699
12. SPONSORING AGENCY NAME AND ADDRESS National Aeronautics and Space Administration Washington, D. C. 20546		13. TYPE OF REPORT & PERIOD COVERED Contractor Report 14. SPONSORING AGENCY CODE

15. SUPPLEMENTARY NOTES

Work performed for George C. Marshall Space Flight Center Computation Laboratory

16. ABSTRACT

This report presents the design principles, description, functional operation, and recommended expansion and enhancements for the Space Ultrareliable Modular Computer (SUMC) interpretive simulator. Included as appendices are the User's Manual, Program Module Descriptions, Target Instruction Descriptions, Simulator Source Program Listing, and a sample program printout. In discussing the design and operation of the simulator, the key problems involving host computer independence and target computer architectural scope are brought into focus.

17. KEY WORDS Interpretive Simulator Host Computer Target Computer Target Program Microprogram	18. DISTRIBUTION STATEMENT Unclassified - Unlimited

19. SECURITY CLASSIF. (of this report)	20. SECURITY CLASSIF. (of this page)	21. NO. OF PAGES	22. PRICE
Unclassified	Unclassified	234	

MSFC - Form 3292 (May 1969)

FOREWORD

The work reported herein was administered in the Applications Development Branch, Computation Laboratory, Marshall Space Flight Center with Mr. Bobby C. Hodges assigned as COR. In addition to his duties as Technical Monitor, Mr. Hodges added to our insight of the development problem through careful planning and coordination.

Acknowledgement is due Mr. Thomas E. Hill, also of the Applications Development Branch, for his many useful suggestions toward the integration of the simulator into the SUMC Support Supervisory System.

Cognizant Astrionics Laboratory personnel contibuted significantly to our understanding of the SUMC design specifications.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

SECTION I. INTRODUCTION

This report presents the description, design principles, functional operation, and recommended expansion and enhancements for the Space Ultra-reliable Modular Computer (SUMC) interpretive simulator. Included in the appendices are the User's Manual, descriptions of machine instructions for the SUMC being modeled, Program Module Descriptions, Simulator Source Program listings, and a sample program printout.

Within the description of the simulation target computer, the basic architecture is discussed in terms of its effect on the simulator software organization. This section also includes a discussion of the instruction set which is executed under the initial simulator implementation as well as planned additions to this target instruction set.

The functional operation of the SUMC simulator is described according to basic operational modules which include the primary control loop, initialization, instruction parse and execute subroutines, user diagnostic aids, program termination, and interrupt simulation routines. In discussing the operation of the simulator, the key problems of host computer independence and target computer architectural scope are brought into focus.

A section is also included outlining recommended simulator expansion and enhancements. Simulation of input/output operations, expansion of the target instruction set, execution efficiency, and simulation of interrupt servicing operations are the topics discussed in this section.

SECTION II. GENERAL DESCRIPTION OF SUMC

A. SUMC Architecture

A simplified block diagram of the Space Ultrareliable Modular Computer (SUMC) is shown in Figure II-1. The Arithmetic Logic Unit (ALU), Main Memory Unit (MMU), Scratch-Pad Memory (SPM), Control Unit (CU) and Multiplexer/Register Unit (MRU) are the five basic functional units of the SUMC. The Floating-Point Unit (FPU) is a sixth basic unit; however, a particular SUMC configuration may or may not include the FPU. Modular construction of the SUMC allows the computer word length to be varied, in four-bit increments, to satisfy specific space mission requirements. For this application, the SUMC simulator models a target computer having a 32-bit word length and no floating-point arithmetic capability, i.e., the SUMC configuration does not include the FPU and only fixed-point arithmetic instructions may be processed.

Figure II-2 depicts a detailed block diagram of the SUMC with the basic functional units broken down into their major components. The Arithmetic Logic Unit (ALU) accepts inputs from the Floating-Point Multiplexer (FPM), SPM, I/O Unit, Microprogram Read-Only Memory (MROM), and Memory Register (MR). The Control Unit (CU) enables the appropriate multiplexer, depending on the instruction. The Add/Sub Units can perform an add, subtract, reverse subtract, logical AND, logical OR, logical EXCLUSIVE OR, 1's complement and 2's complement. The correct function is enabled by a signal from the Control Unit (CU), depending on the instruction.

The Multiplexer/Register Unit (MRU) accepts data from the ALU, SPM, I/O Unit, and FPU. The Product/Remainder Multiplexer (PRM) can accept data from the ALU, force zeros out, shift ALU data right one, left one, left two, right four, and left four. The Memory Address Multiplexer (MAM) gates data or zeros to the memory address register (MAR) from the ALU or MAR. The MAM can shift data right one, left one, left two, right four, and left four. The Multiply Quotient Multiplexer (MQM) gates data or zeros

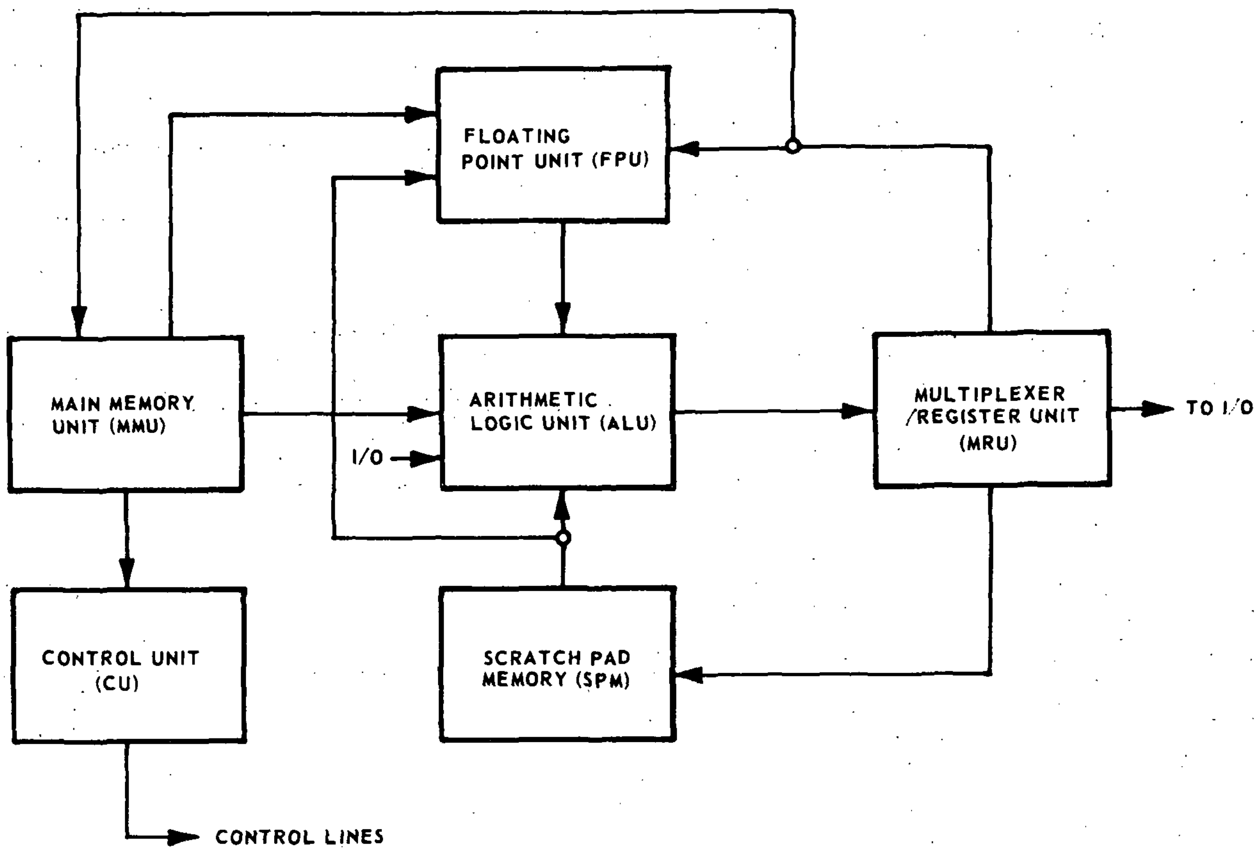


FIGURE II-1 SIMPLIFIED SUMC SYSTEM BLOCK DIAGRAM.

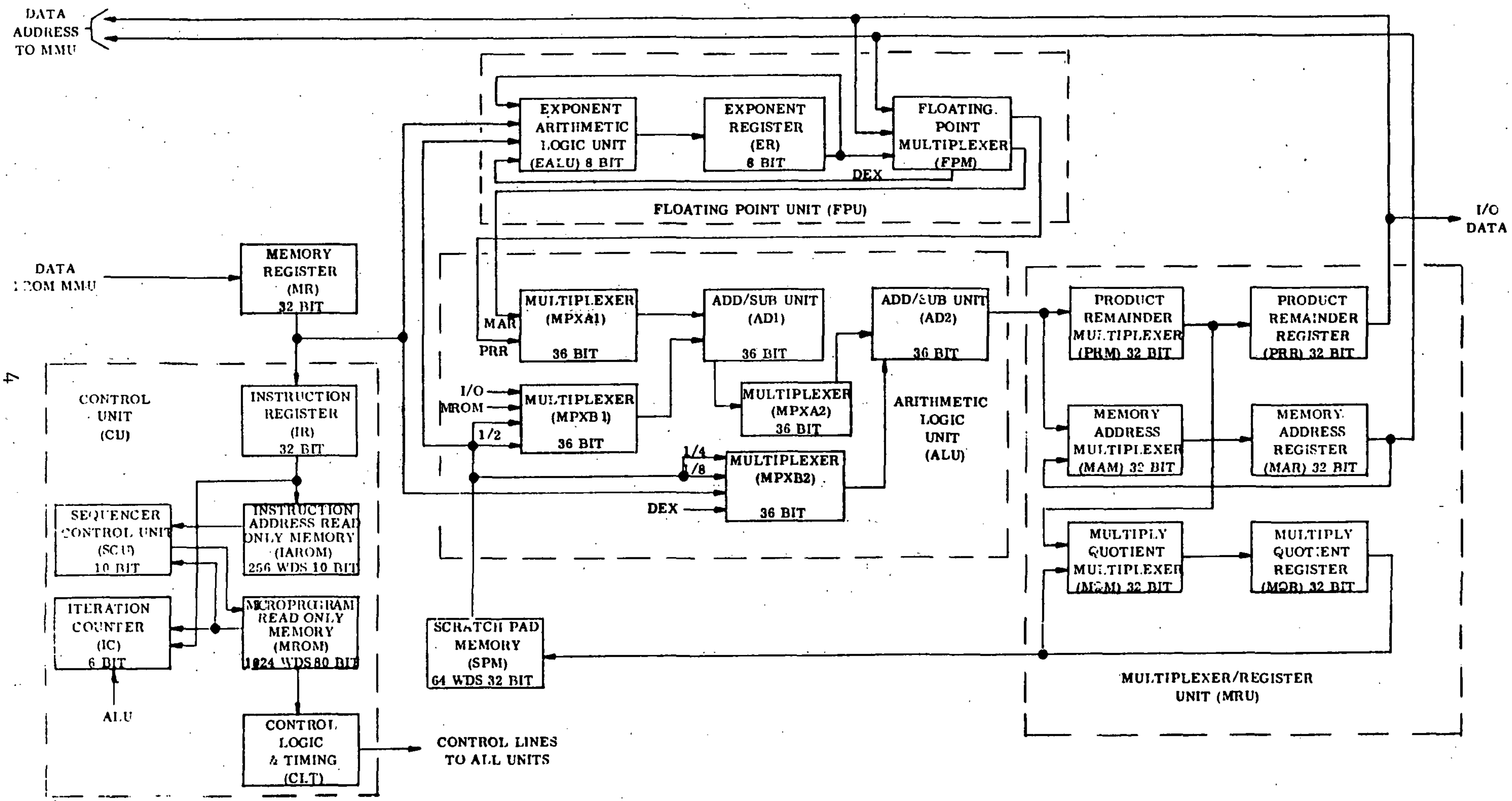


Figure II-2 Detailed SUMC System Block Diagram to the Multiply/Quotient Register (MQR) from the PRM or MQR. The data from the MQM can be shifted left one, left two, and right four. The MQR sends data to the MQM or SPM. The registers and multiplexers in the MRU are controlled by microinstruction signals from the Control Unit and allow data from the ALU to be gated through the multiplexers and clocked into various registers.

The Scratch Pad Memory (SPM) contains program addressable registers, current Program Status Word (PSW), and temporary storage area. The SUMC breadboard system, which acts as the current target computer, contains a 64-word, 32-bit scratch pad memory. The SUMC breadboard SPM contains 16 General Registers, eight Floating-Point Registers, program counter, a system mask word, a program mask word, condition code bits, protection key word and program state word. The SPM layout is shown in Figure II-3. The function of the SUMC SPM is to send data to the ALU and accept data from the MRU while controlled by microinstruction signals from the Control Unit.

The Floating-Point Unit (FPU) is a special logic section which enhances the execution of floating-point instructions. The current version of the SUMC simulator has not been designed to include floating-point capabilities and a discussion of the FPU will not be presented here.

The Control Unit (CU) decodes SUMC instructions and provides microinstruction control signals for the ALU, SPM, MMU, MRU, and FPU as required to execute the current instruction. This unit is made up of a number of distinct components and each of these is discussed briefly in the following paragraphs.

Instruction Register (IR) - This 32-bit register is used to hold the instruction which is currently being executed. The op code portion of the IR is used as an address for the IAROM.

Instruction Address Read-Only Memory (IAROM) - The IAROM is a 64-word, 12-bit read-only memory which contains the starting address of the microinstruction sequence stored in the MROM which will perform the machine instruction. The content of the IAROM, whose address is specified by the instruction op code, is gated to the Sequencer Control Unit.

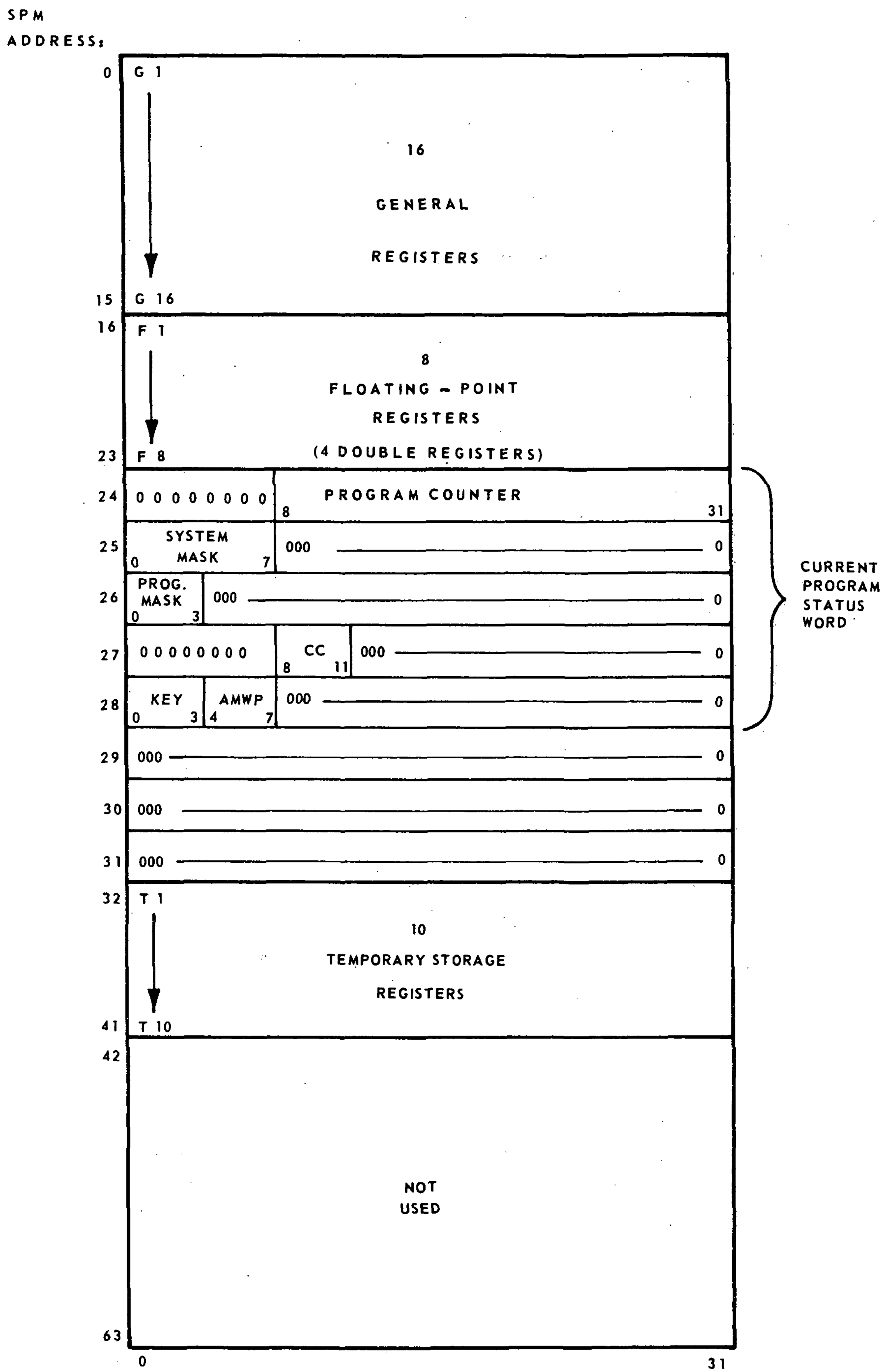


FIGURE II-3. SUMC BREADBOARD SYSTEM SCRATCH PAD MEMORY MAP.

Sequencer Control Unit (SCU) - The sequencer serves as an address register for the microprogrammed read-only memory. The ten-bit sequencer register can be loaded from the IAROM, MROM, or the ALU.

Microprogram Read-Only Memory (MROM) - The MROM is 1024-word, 72-bit read-only memory containing a prestored sequence of microinstructions required to fetch and execute the program instructions, initiate and control I/O operations, and respond to external interrupts. An instruction is executed by broadcasting a location or a sequence of locations of the MROM to the ALU, SPM, MRU, FPU, and main memory.

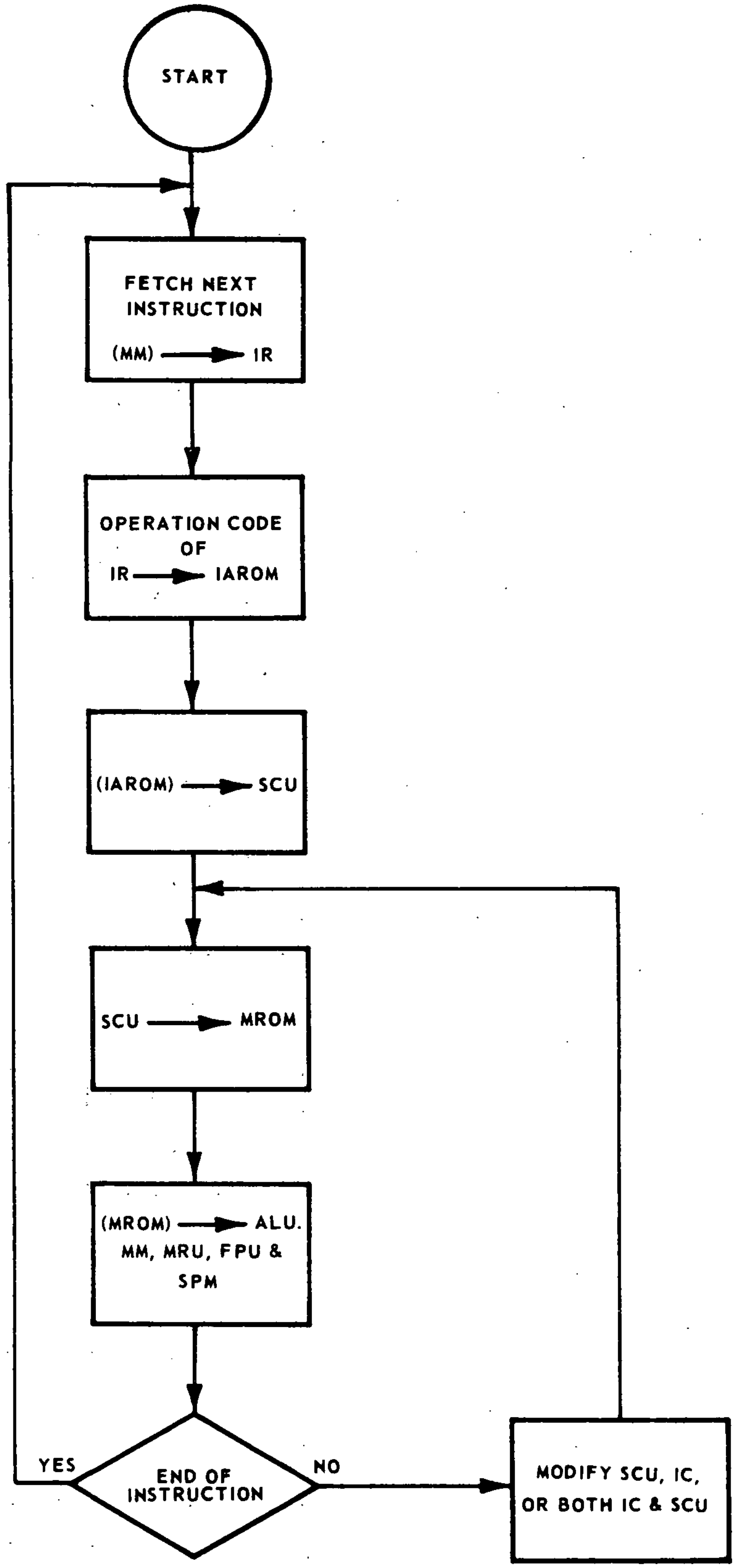
Iteration Counter (IC) - The IC is used to control the number of times a single or a sequence of microinstructions in the MROM should be repeated. The six-bit IC register can be loaded from the IR, ALU, and MROM.

Figure II-4 is a flow chart which depicts the sequence of operations performed by the Control Unit in executing an instruction. More detailed explanations of the operating of the SUMC Control Unit as well as other SUMC architectural features can be found in appropriate literature (1), (2), (3), (4), pertaining to SUMC hardware characteristics. The preceding discussion is intended to relate only the basic principles of SUMC architectural design and the source of the material has been the <u>SUMC Breadboard System Operations Guide</u> (1).

B. SUMC Instruction Set

The SUMC breadboard system is a 32-bit byte-oriented machine which performs arithmetic operations that fall into four classes: fixed point and logical arithmetic, floating-point arithmetic, character manipulation, and I/O operations. The fixed-point and logical arithmetic operations require the following data types:

- Half-word fixed-point number
- Full-word fixed-point number
- Fixed-length logical information



( ) = CONTENTS OF

FIGURE II–4. SUMC CONTROL UNIT (CU) FLOW DIAGRAM

The floating-point arithmetic operations require:

- Short floating-point number
- Long floating-point number

The character manipulation operations require:

- Packed decimal number
- Zoned decimal number
- Variable-length logical information

Figure II-5 shows the data formats for the eight different types of data mentioned above. The SUMC simulator is presently concerned with six of the eight data types in that floating-point capabilities will be added to the program at a later time.

The SUMC breadboard system uses instruction formats which may be one, two, or three half-words in length. A total of five different instruction formats are used:

RR - register-to-register operation format
RX - register-and-indexed-storage operation format
RS - register-and-storage operation format
SI - storage-and-immediate-operand operation format
SS - storage-to-storage operation format

The instruction formats are shown in Figure II-6 along with a brief description of the different fields of each instruction. In describing the execution of instructions, operands are designated as first, second, and third operands according to the manner in which they participate. The operand to which a field in an instruction format applies is denoted by the number following the code name of the field.

Table II-1 lists the instruction set which has been implemented for the SUMC breadboard system simulator. This table contains the instruction op codes, their corresponding mnemonics, and appendix page numbers referring to the instruction description. As mentioned previously, there are four types, or groups, of instructions and the present version of the simulator will interpretively execute SUMC programs for the breadboard

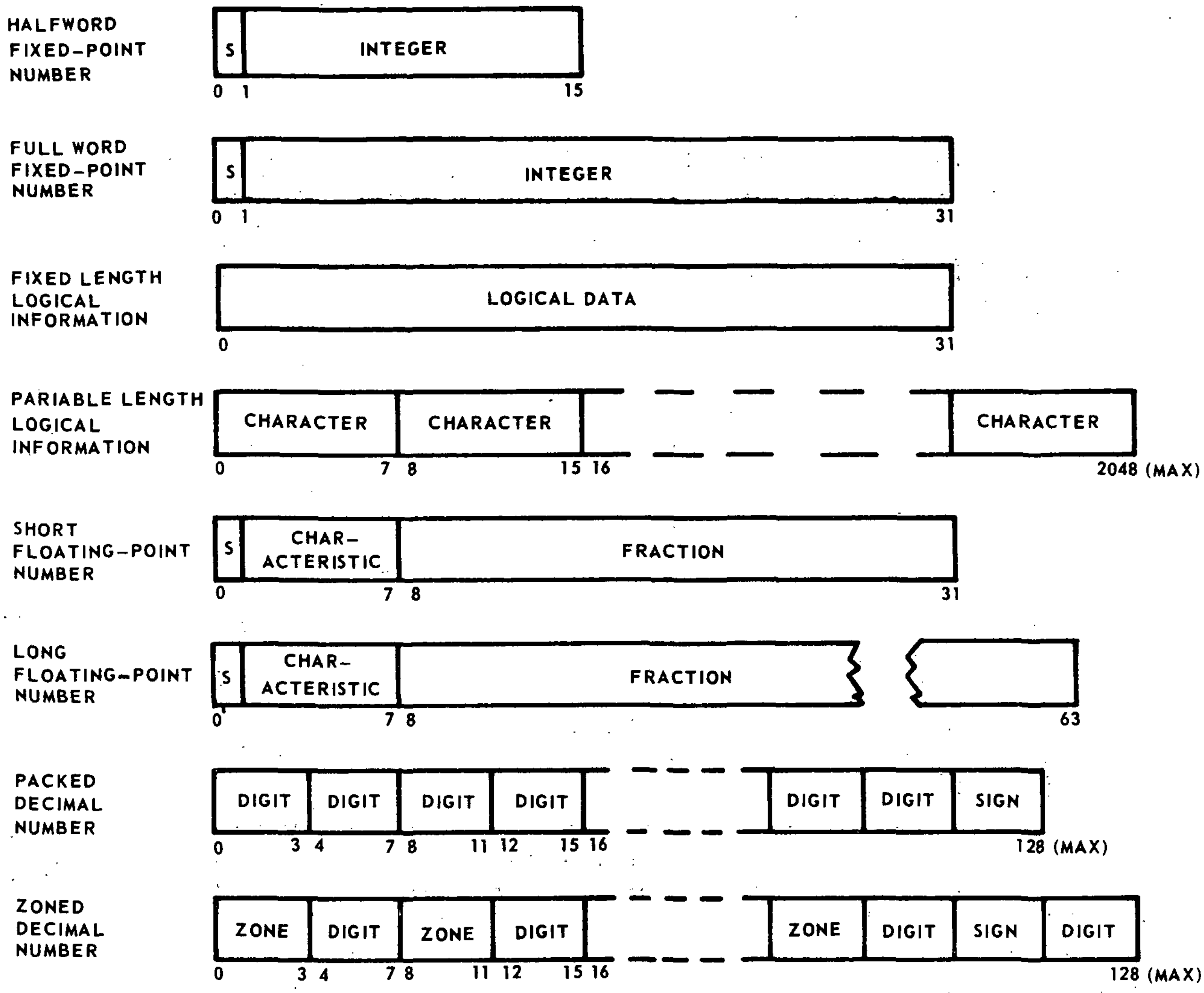
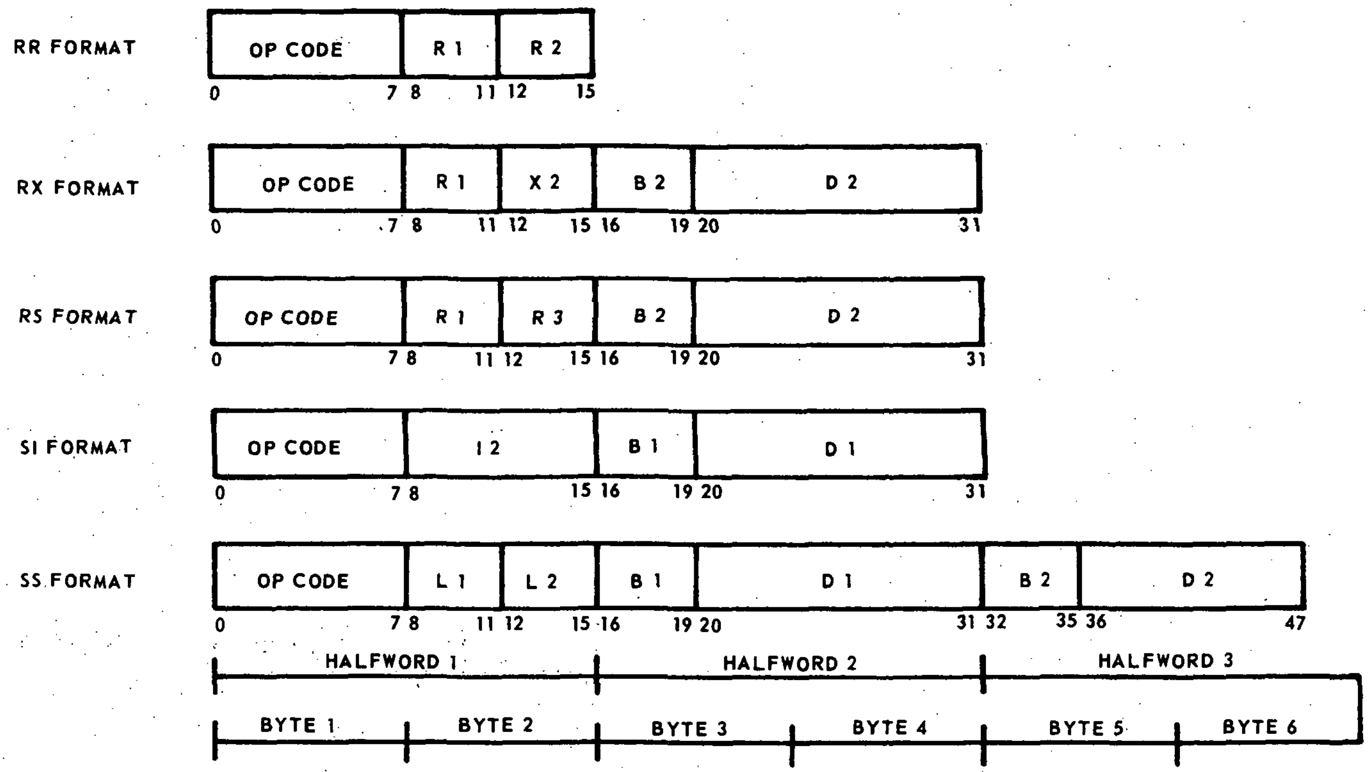


FIGURE II-5. SUMC BREADBOARD SYSTEM DATA FORMATS.



OP CODE – ISTR INSTRUCTION OPERATION CODE

R 1 – REGISTER OPERAND 1 SPM ADDRESS

R 2 – REGISTER OPERAND 2 SPM ADDRESS

R 3 – REGISTER OPERAND 3 SPM ADDRESS

X 2 – OPERAND 2 INDEX REGISTER SPM ADDRESS

B 1 – OPERAND 1 BASE REGISTER SPM ADDRESS

B 2 – OPERAND 2 BASE REGISTER SPM ADDRESS

D 1 – OPERAND 1 DISPLACEMENT

D 2 – OPERAND 2 DISPLACEMENT

I 2 – IMMEDIATE OPERAND 2

L 1 – OPERAND 1 LENGTH SPECIFICATION

L 2 – OPERAND 1 2 LENGTH SPECIFICATION

FIGURE II–6. SUMC BREADBOARD SYSTEM INSTRUCTION FORMATS.

Table II-1. Instruction Set for the SUMC Breadboard System Simulator

Group I. Standard (Fixed-Point) Instructions

Code	Mnemonic	Page	Code	Mnemonic	Page	Code	Mnemonic	Page
04	SPM	III-2	40	STH	III-6	80	SSM	III-13
05	BALR	III-2	41	LA	III-6	82	LPSW	III-13
06	BCTR	III-2	42	STC	III-7	86	BXH	III-13
07	BCR	III-2	43	IC	III-7	87	BXLE	III-13
0A	SVC	III-3	44	EX	III-7	88	SRL	III-14
			45	BAL	III-7	89	SLL	III-14
10	LPR	III-3	46	BCT	III-8	8A	SRA	III-14
11	LNR	III-3	47	BC	III-8	8B	SLA	III-14
12	LTR	III-3	48	LH	III-8	8C	SRDL	III-15
13	LCR	III-3	49	CH	III-8	8D	SLDL	III-15
14	NR	III-4	4A	AH	III-9	8E	SRDA	III-15
15	CLR	III-4	4B	SH	III-9	8F	SLDA	III-15
16	OR	III-4	4C	MH	III-9			
17	XR	III-4	4E	CVD	III-9	90	STM	III-16
18	LR	III-4	4F	CVB	III-10	91	TM	III-16
19	CR	III-5				92	MVI	III-16
1A	AR	III-5	50	ST	III-10	93	TS	III-16
1B	SR	III-5	54	N	III-10	94	NI	III-17
1C	MR	III-5	55	CL	III-10	95	CLI	III-17
1D	DR	III-5	56	O	III-10	96	OI	III-17
1E	ALR	III-6	57	X	III-11	97	XI	III-17
1F	SLR	III-6	58	L	III-11	98	LM	III-17
			59	C	III-11			
			5A	A	III-11			
			5B	S	III-11			
			5C	M	III-12			
			5D	D	III-12			
			5E	AL	III-12			
			5F	SL	III-12			

Group III. Character Manipulation Instructions

Code	Mnemonic	Page	Code	Mnemonic	Page
D1	MVN	III-19	F1	MVO	III-21
D2	MVC	III-19	F2	PACK	III-21
D3	MVZ	III-19	F3	UNPK	III-22
D4	NC	III-19			
D5	CLC	III-20			
D6	OC	III-20			
D7	XC	III-20			
DC	TR	III-20			
DD	TRT	III-21			

system which contain any of the Group I or Group III instructions of Table II-1.

Appendix II describes the execution of the Group I and Group III instructions as they are processed by the SUMC breadboard system and simulated by the SUMC interpretive simulator.

SECTION III. THE SUMC INTERPRETIVE SIMULATOR

A. Design Principles

1. Host Computer Independence. One of the primary design goals for the interpretive simulator is the capability to simulate the operation of the SUMC family of machines on a variety of host computers. To accomplish this, great care has been taken to identify all host-machine-dependent operations which must be performed during a simulation.

The need to design a host computer independent simulator has been dictated by two considerations. First, and probably more important, the resulting program would be valuable to a larger cross-section of users if the problem of transferring the simulator between host computers is not a major or costly undertaking. Secondly, the development effort may be done on whichever machine may be practical or available (in this case, an IBM 7094) and the finished simulator is then easily converted for use on other host systems (in this case, a Univac 1108).

The choice of an appropriate simulator source language was influenced strongly by the desirability of maintaining host computer independence. The standard FORTRAN IV source language was chosen for the simulator since it represents a high-level language which is common to most large-scale computer systems. Although a higher-level language would have eased programming efforts, it was decided that the machine-independence criterion was of overriding importance. It has been recognized that certain FORTRAN IV processing characteristics will vary from one system to the next; however, these have been noted and appropriate coding is used to circumvent this problem.

The requirement for program transfer among several computer installations has led to a highly modular program structure. The simulator has thus been constructed as a set of quasi-independent modules, regulated by a control module, as shown in Table III-1. This table includes all program modules which are presently included in the basic simulator package and those modules which are not completely machine independent are

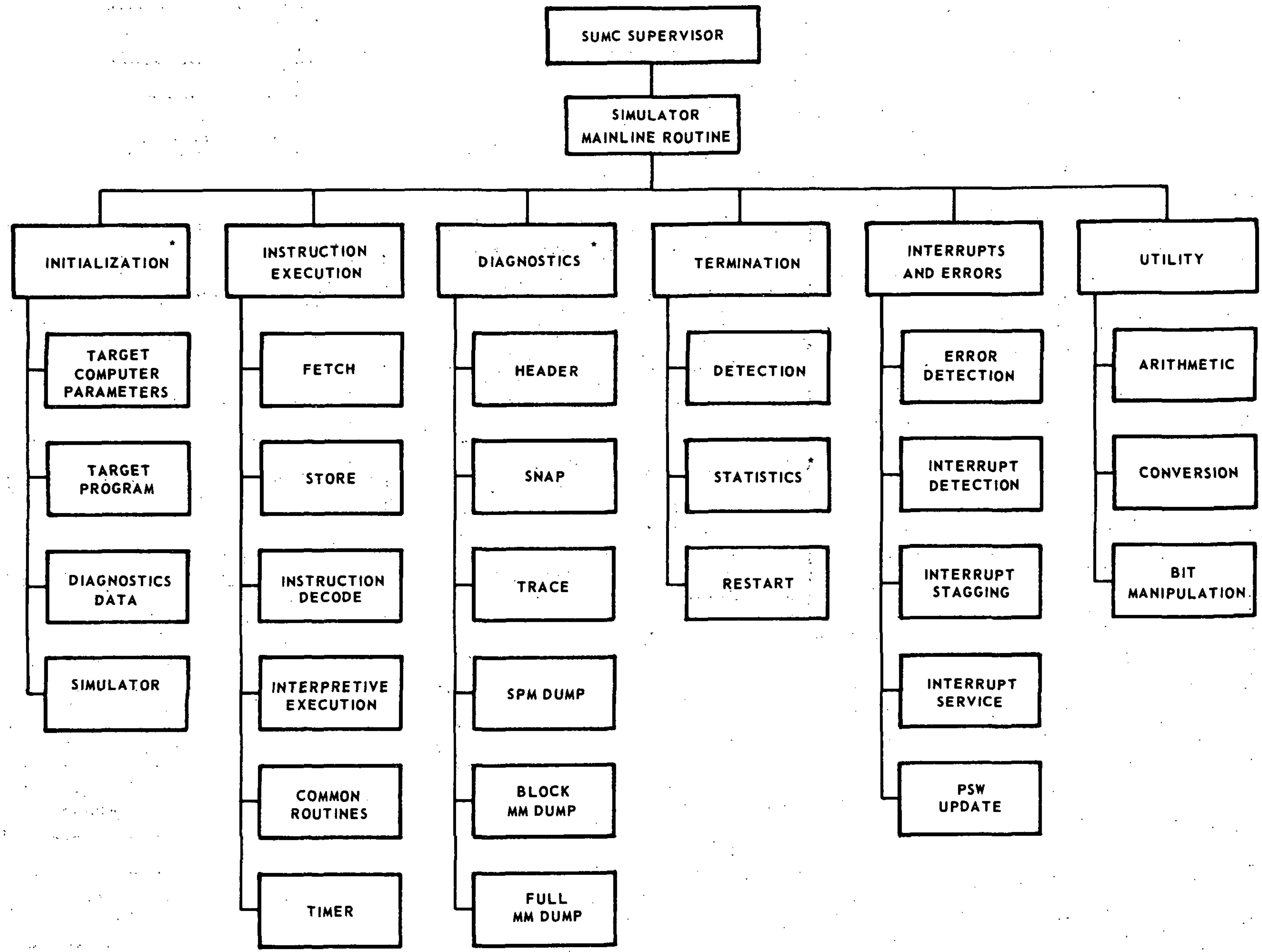


TABLE III-1. BASIC SUMC SIMULATOR MODULES.

marked with an asterisk. Table III-2 gives a brief description of the function of each program module. Transfer of the simulator among host computers would therefore require modifications or replacement of only those modules marked in the figure.

There are three primary areas of programming for the simulator in which differences in host computer characteristics had a noticeable effect. These are:

- host computer word length
- host computer arithmetic
- host computer I/O procedures

Problems encountered in each of the above areas have been resolved such that the simulation package is machine independent to the fullest possible extent.

a. Word Length. The possibility of a variation in host computer word length when transferring the simulator between host systems is a readily apparent problem. A parameter, IHOST, specifying the host computer word length has therefore been introduced as a common variable in appropriate simulator subroutines. The IHOST variable is initialized along with other standard program variables prior to start of a simulation.

b. Arithmetic. The SUMC target computer employs two's-complement arithmetic exclusively. However, a given host computer may be a two's-complement, one's-complement or sign-magnitude machine. The following possibilities are therefore reasonably likely:

- 2's-complement target computer & 2's-complement host
- 2's-complement target computer & 1's-complement host
- 2's-complement target computer & sign-magnitude host

In the first case, when both target and host computers employ 2's-complement arithmetic, simulation of target computer operations is straightforward and conversion problems are not present.

In the two latter cases, however, proper conversions must be made during simulation since arithmetic quantities are represented differently in host and target systems. The present version of the SUMC simulator

Table III-2. Simulator Module Definitions

Program Modules	Function Summary
A. INITIALIZATION	
1. Target Computer Parameters	Input values for host and target computer architectural parameters.
2. Target Program	Input target computer memory map.
3. Diagnostics Data	Input diagnostics keys and corresponding numerical data for diagnostic control.
4. Simulator Variables	Input internal simulation parameters.
B. INSTRUCTION EXECUTION	
1. Fetch	Fetch instructions and data from simulated main memory.
2. Store	Store instructions and data in simulated main memory.
3. Instruction Decode	Parse current instruction and represent contents as FORTRAN variables.
4. Interpretive Execution	Process current instruction.
5. Common Routines	Arithmetic processing common to several instruction routines.
6. Timer	Maintains records pertaining to program simulated elapsed time.
C. DIAGNOSTICS	
1. Header	Information printed to identify diagnostic output.
2. Snap	Check and execute SNAP diagnostic.
3. Trace	Check and execute TRACE diagnostic.
4. SPM Dump	Check and execute SPM DUMP diagnostic.
5. Block MM Dump	Check and execute BLOCK MM DUMP diagnostic.
6. Full MM Dump	Check and execute FULL MM DUMP diagnostic.
D. TERMINATION	
1. Detection	Detect program termination conditions.
2. Statistics	Collect and print end-of-run statistics.
3. Restart	Collect restart data.

Table III-2. Simulator Module Definitions (Continued)

Program Modules	Function Summary
E. INTERRUPTS AND ERRORS	
1. Error Detection	Detect and flag error conditions.
2. Interrupt Detection	Detect and identify interrupt conditions.
3. Interrupt Stacking	Maintain stack of pending interrupts.
4. Interrupt Service	Process pending interrupts at appropriate times according to predetermined priorities.
5. PSW Update	Maintain old and new program status words in simulated MM and SPM
F. UTILITY	
1. Arithmetic	Perform generic arithmetic operations.
2. Conversion	General conversion routines.
3. Bit Manipulation	Perform generic bit manipulation operations.

is operational on the IBM 7094 system which is a sign-magnitude machine. Two special (host machine dependent) routines are therefore present - ITWTSM, for conversion of 2's-complement data to sign-magnitude representation, and ISMTWO, for conversion of sign-magnitude to 2's-complement representation. Prior to simulation of a particular SUMC operation, appropriate target computer data must be converted to host machine form using the ITWTSM routine. Following the simulated operation, which is performed under host machine arithmetic, the results are converted to target machine form using the ISMTWO routine. It should be noted here that performing arithmetic operations using an arithmetic base other than that of the host machine would be prohibitive from an efficiency standpoint. Thus, the conversion routines are necessary.

Transfer of the simulator to a Univac 1108 operating environment would of course require appropriate conversion routines in order to perform arithmetic operations using the host computer 1's-complement representation. Substitution of the different conversion routines can be done with a minimum of difficulty.

c. I/O Operations. The most difficult problem encountered in achieving a truly machine independent simulator has been in the area of host machine Input/Output operations. That is, I/O processing characteristics are highly dependent on the particular choice of host computer. For this reason, all simulation procedures which require the utilization of host computer I/O operations have been segregated so that extensive I/O revisions are not necessary when transferring the simulator between host computers.

Host computer I/O operations are necessary during four distinct simulation phases:

- Initialization
- Diagnostics
- Termination
- Interrupt servicing

During initialization, the host computer must input the target computer memory map as well as the various target machine and host machine parameters which are needed for simulation. All READ operations which are performed by the host computer are included in the subroutine INITLZ. This routine handles the initialization operations at the start of each simulation and must input the following:

- Target computer main memory map
- Target computer SPM map
- Target computer architectural parameters
- Host computer architectural parameters
- Simulation control variables
- Diagnostics variables

This routine is written in standard FORTRAN IV for processing on the IBM 7094. Operation on another host machine could of course require modification or replacement of INITLZ.

The simulator includes a number of diagnostic features which require both READ and WRITE operations to be performed by the host computer. Since the simulation diagnostic aids are designed primarily for target computer debugging and verification in the environment of a commercially available system, the applicable routines are inherently host machine dependent to a certain extent. However, by coding these routines in the standard FORTRAN IV source language and circumventing host computer dependencies where possible, the modifications required for switchover to a different host system are not major ones.

Several host machine I/O operations are required at the termination of a simulation run. These are WRITE operations which may be any of the following:

- error messages
- statistical dumps
- target memory dumps
- diagnostics information

Each of the above output operations have been coded using distinct subroutines and standard FORTRAN IV source language. Relatively few lines of code are needed for these operations which allows subroutines that are easily transferable between host systems.

The Interrupt Service routines have not been finalized for the IBM 7094 version of the simulator. Some machine dependencies exist in the present scheme; however, it is hoped that the final version of the simulator will feature interrupt processing simulation which is completely host machine independent.

2. Target Computer Architectural Scope. The basic interpretive simulator, as presently configured, operates on an IBM 7094 host system and has the capability to model a SUMC family of target machines. This section will be devoted to a discussion of the SUMC characteristics and design parameters which may vary under the present simulation program. Flexibility has been designed into the simulator so that an even wider range of target machines may eventually be modeled through future enhancements to the basic program. These possibilities are also discussed in this section.

The simulator has been designed such that changes in the following SUMC architectural features can be accommodated presently or with little additional modification.

- SUMC word length
- main memory size
- scratch-pad memory size
- scratch-pad memory organization
- microprogram read-only-memory contents
- addition of floating-point arithmetic unit
- I/O devices
- interrupt response routines

The present version of the SUMC simulator is capable of adjusting to variations in several of the above features. Planned enhancements in the remaining areas will allow the simulation of a widely varying group of target computers. The present allowable architectural scope of the SUMC target computer, in terms of the above mentioned features, will be described here in detail.

a. SUMC Target Computer Word Length. The current version of the SUMC simulator is intended to model a target computer having a word length of 32 bits. However, the program can be readily modified to simulate a similar SUMC computer having a smaller word length. The allowable word lengths vary from a minimum of 16 bits to the present 32 bits, with the restriction that the word length be a multiple of four. This restriction is certainly reasonable since the SUMC has been designed as a four-bit modular computer. The target computer word length is a simulation parameter in the form of a COMMON variable called ITARG, which may be initialized by the user at the start of a simulation.

b. SUMC Target Computer Main Memory Size. A variation in the size of the target computer main memory is certainly a very likely possibility. Provision has been made for this occurrence in the present version of the SUMC simulator. The number of addressable locations in the SUMC main memory is specified for the simulation program via the DIMENSION statement for the array IMAINM. The size of simulated target main memory can then be changed by modification of the IMAINM array dimension statement in each appropriate subroutine or subprogram. The IMAINM array contains the current contents of simulated SUMC main memory throughout the simulation. The number of addressable locations which may be simulated for the SUMC main memory varies from a minimum of 128 locations to a maximum of 32,768 locations.

c. SUMC Target Computer SPM Size. As in the case of target computer main memory size, SUMC scratch-pad memory size is likely to vary. In the present version of the simulator, SPM registers are simulated as an array called ISPM, which contains the current contents of the target computer SPM in SUMC format. If the number of registers contained in SPM should change for a particular target computer, a simple change in the dimension statement for the ISPM array will effect a corresponding change in the simulation program. The present SUMC target machine utilizes a SPM which contains 64 registers. This represents the minimum number of SPM words which are anticipated for a particular target computer configuration. Any reasonable increase in the target computer SPM size could be handled under the present program.

d. SUMC Target Computer SPM Organization. Figure II-3, which is referred to in a previous section, describes the SUMC SPM layout which is used for the current target computer. It basically consists of:

- General registers
- Floating-point registers
- Status registers
- Temporary storage registers
- Spares

Due to the fact that the component parts of Scratch Pad Memory as well as their organization may vary from one SUMC application to the next, the SUMC simulator has been designed with considerable flexibility in this respect.

(1) General Registers - This group of registers is made up of accumulators, base registers, index registers, and general purpose registers. The present design philosophy is to allow each of the general registers to be used as an accumulator, base register, or index register. That is, a contiguous block of 16 general purpose registers. In addition, this block of registers may occupy any 16 continuous SPM locations. A particular general register is addressed by adding an offset from zero to the instruction register address, with the offset being a simulation parameter initialized by the user.

(2) Floating Point Registers - The simulator currently contains no provisions for executing floating point instructions and therefore does not allocate SPM registers for floating point operations. The problem of register storage for floating point arithmetic will be addressed when a final version of the SUMC simulator is implemented.

(3) Program Status Word (PSW) - Target computer program control and linkage is accomplished through a number of program status words, as in the IBM 360 machine. Under this scheme, the current program status word resides in scratch pad memory in the form of a group of status registers. For the simulator, all status words used in controlling the target computer are present in the form of COMMON variables. In assigning SPM locations for the status information, each simulation variable, which is actually part of the overall program status word, is made

equivalent to its desired SPM location. To change the layout of the PSW in scratch pad memory, an appropriate change in each applicable EQUIVALENCE statement is necessary.

(4) Temporary Storage Registers - Ten temporary storage registers are included in the target SUMC SPM layout. The number and location of these registers may be varied in the current version of the SUMC simulator. The temporary registers are assigned to elements of the ISPM array using FORTRAN EQUIVALENCE statements and appropriate modifications of these statements will adjust the ISPM layout accordingly.

(5) Spares - Any SPM registers which remain unused for a given scratch pad layout are included in this category and are inconsequential to simulator operations.

Table III-3 lists the critical SUMC design parameters which may vary according to the particular target computer under consideration. Minimum and maximum values permitted for each parameter are given along with incremental variations which are allowed. Table III-4 lists the values which are currently assumed for the SUMC target computer and have been implemented in the initial version of the simulator.

e. SUMC Target Computer MROM. SUMC instruction execution is controlled by signals from the microprogrammed read-only memory. Any additions, deletions, or modifications which are made to the instruction set of the target computer are implemented through a change in the applicable microcode. The SUMC simulator, in a similar manner, performs interpretive instruction execution by means of FORTRAN subroutines and, therefore, changes in the instruction set of the target computer are transformed into modifications in the appropriate subroutine. The present version of the simulator performs all actual instruction execution operations with the OPDEF subroutine and instruction set changes would, in most cases, involve only this particular subroutine.

f. SUMC Target Computer Floating Point Arithmetic. No floating point arithmetic instructions have been implemented for this version of the SUMC simulator. The addition of floating point arithmetic capability is planned as one of the early enhancements to the basic simulator.

Table III-3. Critical SUMC Architectural Parameters

Parameter	Minimum	Maximum	Increment
Word Length (bits)	16	32	4
SPM Size (words)	16*	256	-
Accumulators	1	16	1
Base Registers	1	16	1
Index Registers	1	16	1
General Registers	0	16	1
MM Size (locations)	128*	32,768	-

*The minimum is shown for illustrative purposes and is not considered to be a critical value.

Table III-4. Current SUMC Design Parameters

Parameter	Value
Word Length (bits)	32
SPM Size (words)	64
Accumulators	0
Base Registers	0
Index Registers	0
General Registers	16
MM Size (locations)	4,096

g. SUMC Target Computer Interrupts. The interrupt scheme associated with a given target computer will in general be unique to that particular machine. This dependence of the interrupt action on the target computer results in an interruption package for the SUMC simulator which will vary greatly from one application to the next. An attempt has therefore been made to isolate all interrupt operations in the simulator within a few specific program modules.

The present SUMC target computer employs an interrupt scheme which is similar to that of an IBM/360 system, with certain exceptions. An interruption consists of storing the current PSW as an old PSW and fetching a new PSW. Processing resumes in the state indicated by the new PSW. The old PSW contains the address of the instruction that would have been executed next if an interruption had not occurred and the instruction-length code of the last interpreted instruction.

The interruption action for the target computer will differ from IBM/360 operation in the following respects:

- Interrupts occur from just a single I/O channel.
- External interrupts originate only from the operator interrupt key.
- No decimal arithmetic program interruptions.
- No protection exception program interruption.

With the exception of the above differences, interrupt processing for the SUMC target computer will parallel that of the IBM/360 system.

The simulator presently is capable of detecting all target computer interrupt conditions and will notify the user of their presence. Interrupt response or service routines have not been implemented for this version of the simulator but are planned for inclusion at a later date. A detailed explanation of all interrupt conditions which are detected and the program action which is taken will be given in a later section of this report.

h. SUMC Target Computer I/O. The SUMC simulator does not provide for simulation of input/output operations performed by the target computer. Actual simulation of I/O operations will be performed by I/O device simulation routines which will be written for each individual application. Another section of this report, which covers future enhancements and expansion of the SUMC simulator, will describe in further detail a proposed I/O simulation scheme which could be added to the basic simulator at a later time.

B. Functional Operation

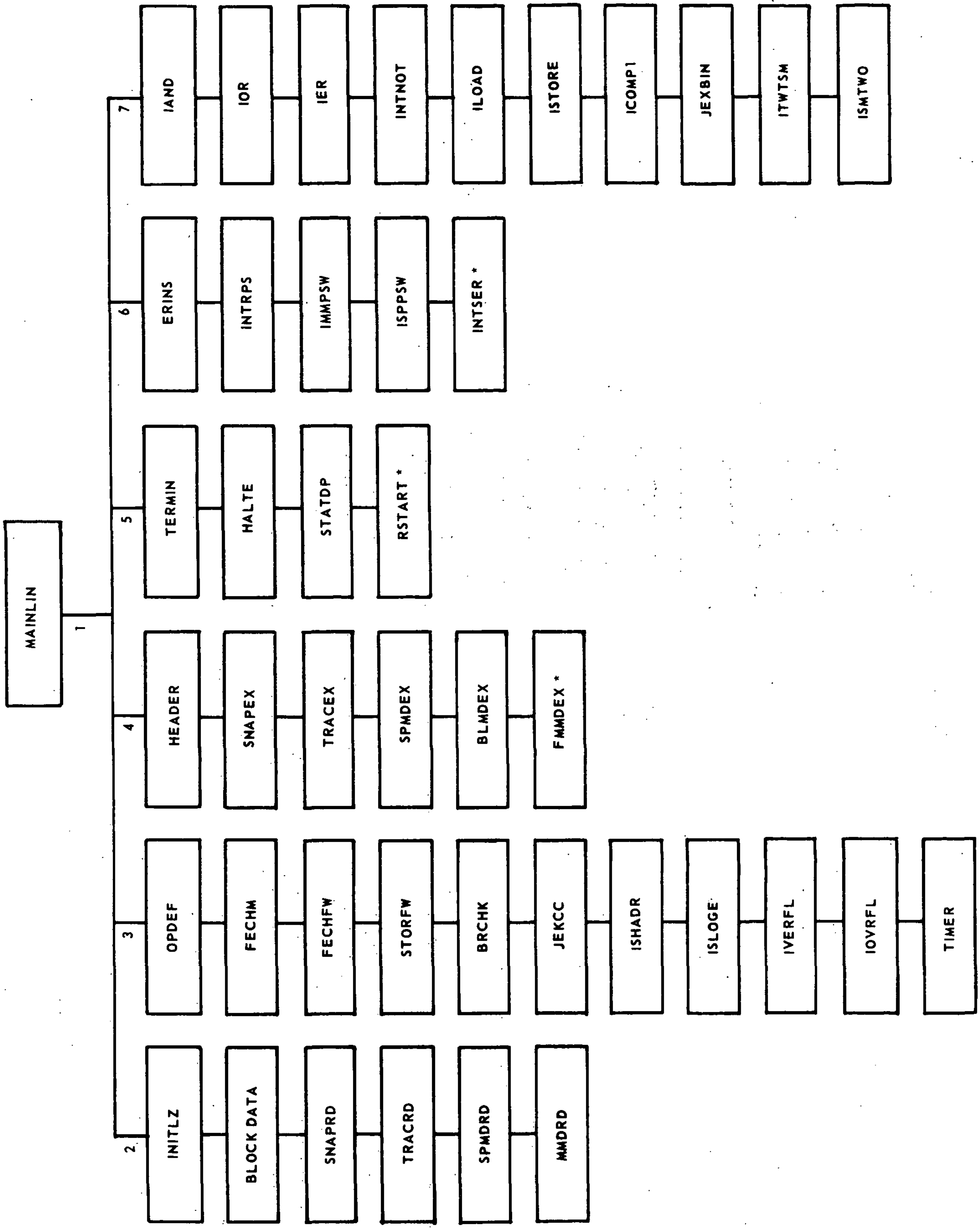
The SUMC interpretive simulator has been designed in a stand-alone, highly modularized fashion with a single supervisor module, MAINLIN, controlling all simulation sequencing. As shown in Figure III-1, the subroutines which operate under control of MAINLIN are divided into six areas:

- Initialization routines
- Instruction fetch and execute
- Diagnostics routines
- Termination routines
- Interrupt servicing
- Utility routines

Although there are no restrictions within the basic simulator package concerning CALLING and CALLED subroutines, the basic cycling of operations which are performed to interpretively execute each instruction is controlled in a macro sense by the MAINLIN program.

The simulation process performed by the complete set of program modules, under control of MAINLIN, is functionally self-contained for present operation in the IBM/7094 development environment. However, the SUMC simulator will ultimately become part of a larger set of programs devoted to support of the SUMC family of computers as shown in Figure III-2, Integration of Simulator into SUMC Support Software. This report will describe the functional operation of the simulator in its present form as an independent programming system.

1. Primary Control Loop. As mentioned in the previous section and shown in Figure III-1, the basic control for the simulator is provided


MAINLIN
1
2
INITLZ
BLOCK DATA
SNAPRD
TRACRD
SPMDRD
MMDRD
3
OPDEF
FECHM
FECHFW
STORFW
BRCHK
JEKCC
ISHADR
ISLOGE
IVERFL
IOVRFL
TIMER
4
HEADER
SNAPEX
TRACEX
SPMDEX
BLMDEX
FMMDEX *
5
TERMIN
HALTE
STATDP
RSTART *
6
ERINS
INTRPS
IMMPSW
ISPPSW
INTSER *
7
IAND
IOR
IER
INTNOT
ILOAD
ISTORE
ICOMP1
JEXBIN
ITWTSM
ISMTWO

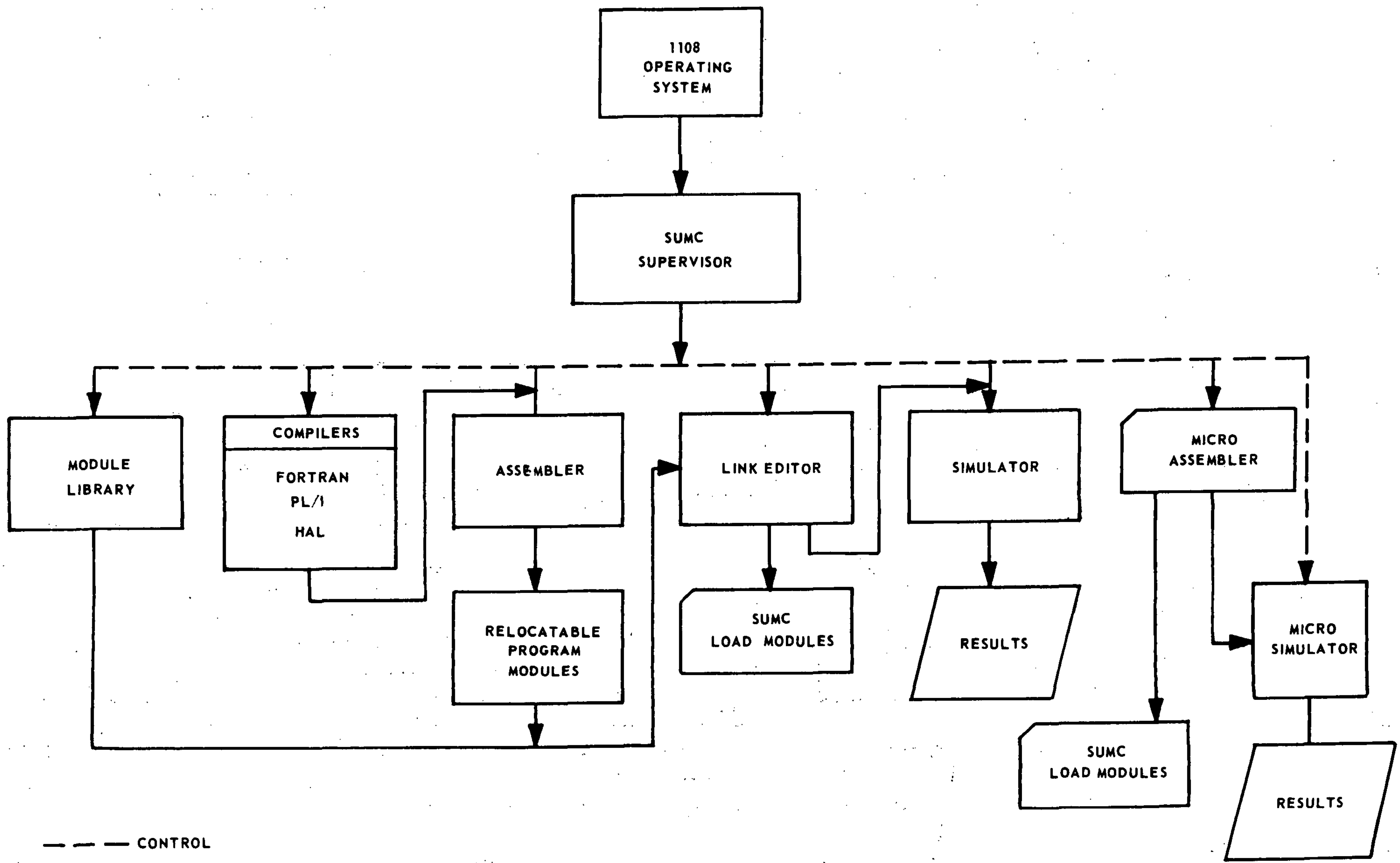


FIGURE III-2. INTEGRATION OF SIMULATOR INTO SUMC SUPPORT SOFTWARE by the MAINLIN program. The simulation process is carried out, under control of MAINLIN, in the following primary phases:

- Input of Job Description
    - host computer parameters
    - target computer parameters
    - diagnostics control variables
    - target computer memory map
- Interpretive Execution of Target Program
    - interruption action
    - fetch and decompose next target instruction
    - perform diagnostics requested
    - execute instruction and update timer
    - error termination or process next instruction
- Termination of Program
    - printout of requested results and/or error message
    - exit

A flow diagram of the MAINLIN routine is shown in Figure III-3. The initial action taken by the program is to dimension all necessary variables and define the program COMMON area. The simulation initializer routine, INITLZ, is then called in order to input the necessary simulation parameters, diagnostics keys, and target program.

If the simulator has been properly initialized, the error flag remains set at zero and the program begins the interpretive execution of target program instructions in the following steps.

a. If the interrupt counter is at zero, there are no interrupts pending from a previous instruction and control is transferred to the simulated fetch cycle. If any interrupts are pending, they are serviced according to a predetermined priority. Interrupt servicing will be handled by subroutine calls controlled by interrupt keys.

b. After all pending interrupts have been serviced, the next instruction is fetched from simulated target main memory by the FECHM

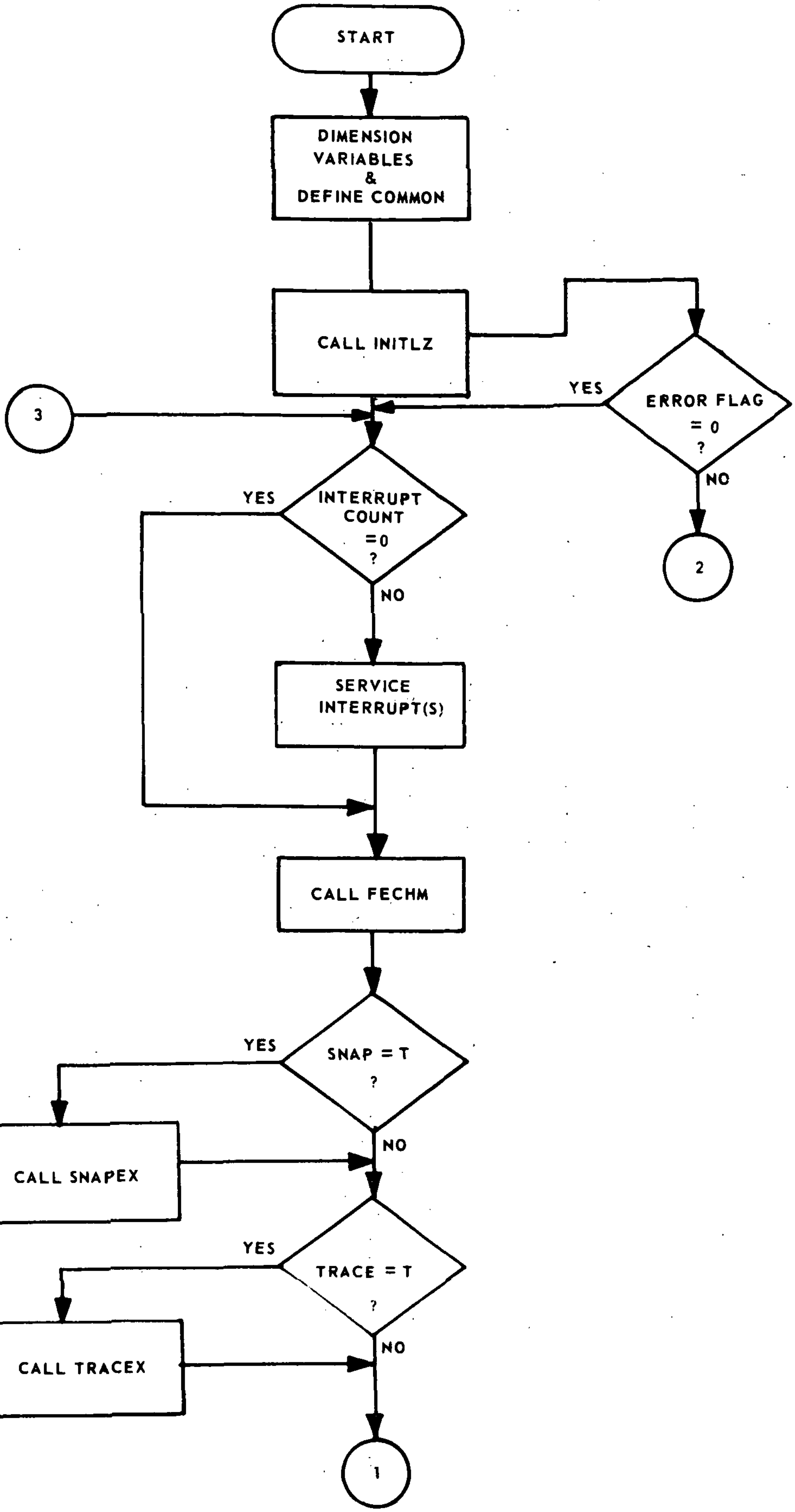


FIGURE III-3. FLOW DIAGRAM OF MAINLIN PROGRAM

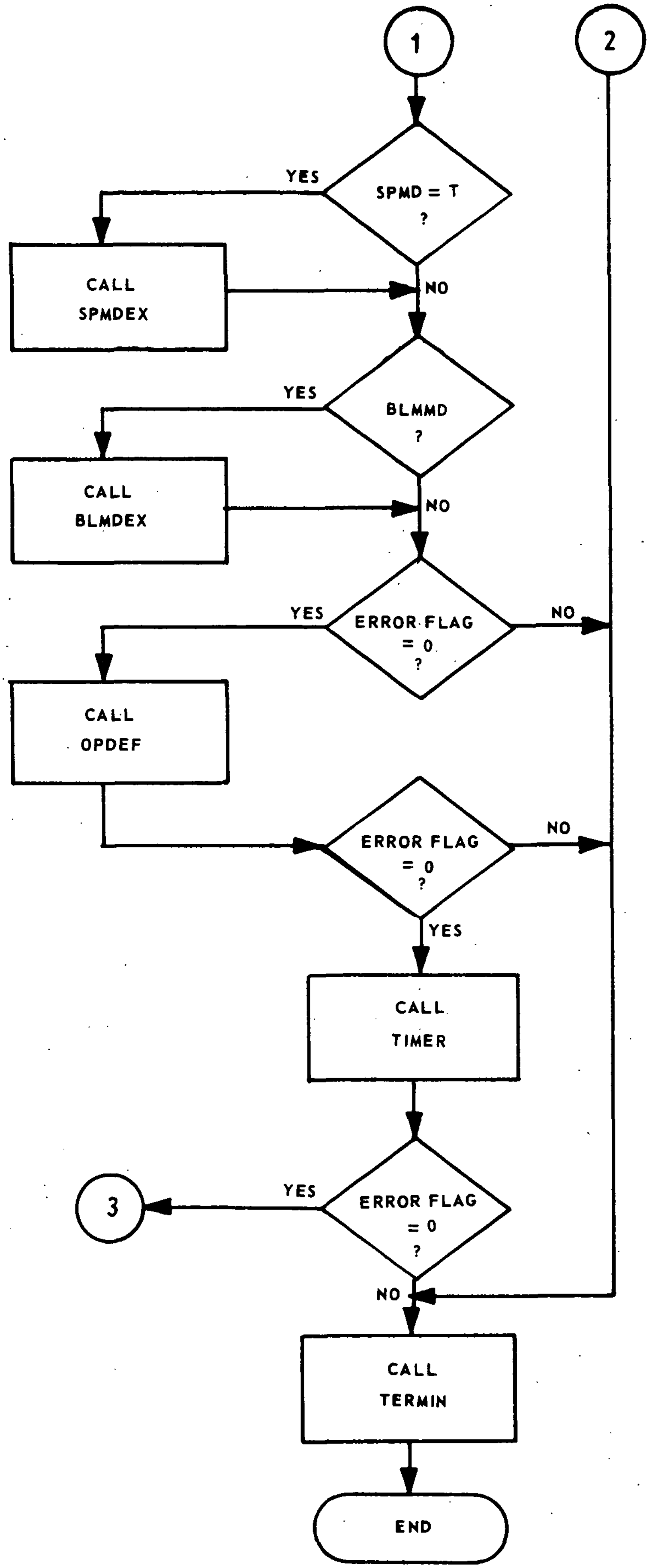


FIGURE III–3 (CON'T)

routine. This subroutine obtains the proper instruction according to a simulated program counter. FECHM also parses the current instruction in order to transform all information contained in the instruction word into the form of FORTRAN variables usable by the simulator.

c. The simulation error flag is checked again upon return from the FECHM subroutine and, if it remains zero, the program begins checking for user diagnostic requests. The four types of user diagnostics made available by the simulator are checked as follows:

(1) If any main memory "snap" diagnostics have been requested by the user, the SNAPEX subroutine is called to execute the check. This routine checks all extant snap keys to determine whether the user desires data at this point in the program. If so, the appropriate data is collected for printout.

(2) If any register traces have been requested, the TRACEX subroutine is called to execute the check. This routine checks all trace keys and collects appropriate data if a register trace is desired.

(3) If a printout of scratch pad memory contents is desired at some time during the simulation, the SPMDEX routine is called at this point to check SPM dump keys which have been supplied by the user to trigger the diagnostic. If triggered, the current contents of all SPM registers are printed before executing further.

(4) A printout of the contents of a particular block of contiguous main memory locations may be requested by the user at some point during the simulation. The BLMDEX subroutine is called to check keys used to trigger this dump. If triggered, a printout of the contents of the specified locations is executed by this routine.

d. The program is now set up to perform the interpretive execution of the current instruction. The OPDEF subroutine is called for this purpose and subsequently performs arithmetic operations and data manipulation called for by the instruction. The interpretive execution of a target instruction must maintain the contents of all SUMC registers available to the target-computer programmer. In addition, the contents of the simulated SUMC registers must duplicate, on a bit-for-bit basis, the contents of actual target computer registers under normal operation.

e. If the simulator error flag remains set at zero upon return from the OPDEF subroutine, the TIMER routine is called for the purpose of updating simulated execution time.

f. Following the return from the TIMER subroutine and another check of the error flag, the program returns to the interrupt detection loop and prepares to execute the next instruction.

At the termination of a simulation run, due to either (1) the execution of a target program HALT instruction or (2) the setting of the simulation error flag, the TERMIN subroutine is called. This routine presently handles post processing statistics printouts, but will also control collection of simulation restart data when this capability is implemented.

2. Initialization. The SUMC has been designed as a highly modular machine and is therefore capable of being configured in a number of different ways. This feature presents a unique problem in the simulation of such a machine in that quite a large number of target computer characteristics must be parameterized and input to the simulator as variables during initialization. These parameters involve both hardware and software aspects of the target computer which are subject to change from one application to the next.

The arrays which simulate SUMC scratch pad memory and main memory must be given their initial values during the initialization program. Any registers or main memory locations which are dedicated under a particular SUMC configuration must appear in appropriate EQUIVALENCE statements. These equivalence statements would require modification when changes in dedicated register assignments are made. The initialization operation for simulated SUMC main memory will of course include the input of the SUMC target program.

The simulation diagnostics which are available to the user - SNAP, TRACE, SPM DUMP, and MM DUMP - are controlled through a set of diagnostic keys which trigger an appropriate printout at a specified point in the program. The diagnostics keys, if any, are specified by the user and input to the simulator by the INITLZ routine.

Finally, those simulation variables which are required internally by the simulator are initialized by INITLZ. Figure III-4 is a flow diagram showing the distinct operations performed during initialization by the INITLZ subroutine.

The first three operations performed by INITLZ, as shown in the flow diagram, are:

- dimension variables and define COMMON
- EQUIVALENCE statements
- initialize simulation variables.

These initialization operations require no input from the user, but are handled internally by the simulator. The final four operations require external inputs to the simulator and the nature of these inputs is described in the following paragraphs.

a. READ user variables. The current version of the simulator allows the user to specify twelve of the key target computer parameters in the form of input data. Table III-5 lists the twelve variables which must be initialized by the user and Appendix I, SUMC Simulator User's Manual, specifies data formats to be followed for proper input.

b. Initialize SPM registers. Part of the SPM initialization procedure is the specification of the function of SPM registers. This is done in part by the user as inidcated by Table III-5 and variables LOCG, LOCT, and LOCF. Armed with this information, along with appropriate DIMENSION statements for each block of registers, the exact location of each general register, floating point register, and temporary storage register is known to the program. The function and location of all other registers in scratch pad memory are specified for the program through EQUIVALENCE statements.

c. READ diagnostics keys. Any of the available simulator diagnostics which the user may wish to exercise must be activated through

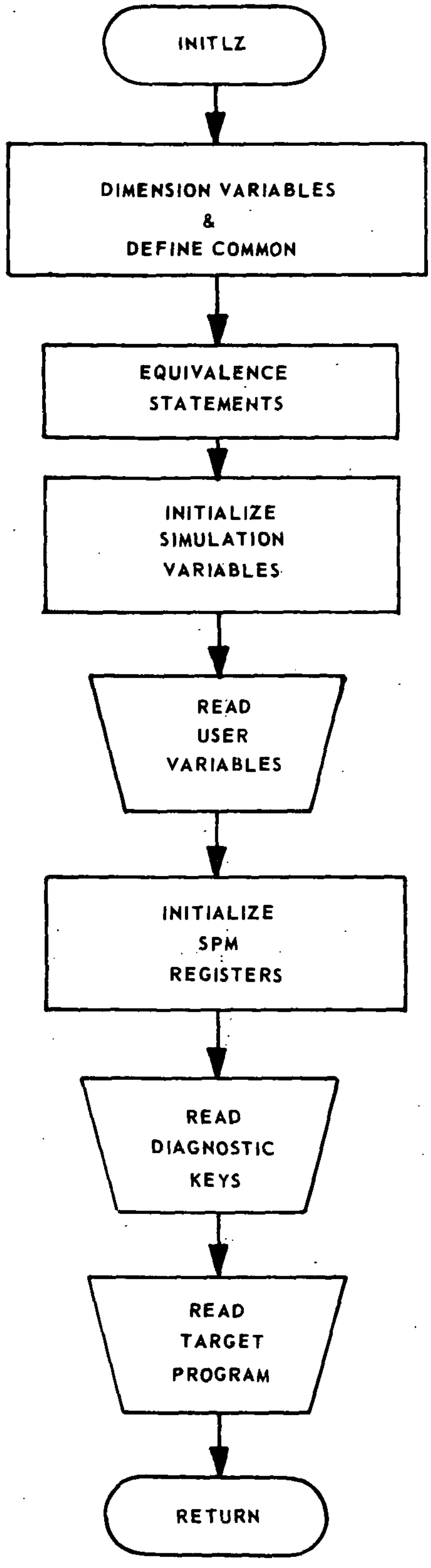


FIGURE III-4. FLOW DIAGRAM OF INITLZ ROUTINE

Table III-5. Target Computer Parameter Values Specified During Simulation

Variable	Parameter Description
IHOST	Host computer word length
ITARG	Target computer word length
MAXCOR	Number of addressable locations in target MM
LOBOUN	Target MM lower bound address
JIBOUN	Target MM upper bound address
MAXSPM	Number of registers in target SPM
LOCG	Offset to first SPM general register
LOCT	Offset to first SPM temporary register
LOCF	Offset to first SPM floating-point register
IFWA	Target program first word address
INADDR	Target program first instruction address
MAXTIM	Maximum simulated execution time

the appropriate diagnostics input data. The INITLZ routine is presently designed to read this data from cards supplied by the user. A detailed description of the simulator diagnostics features is included in a later section of this report. The proper data formats as well as the deck set-up used for diagnostics input are given in Appendix I, SUMC Simulator User's Manual.

d. READ target (SUMC) computer program. This portion of the INITLZ routine reads the target program which is to be simulated. The program is input in SUMC machine language form since the actual program input consists of the target SUMC main memory map. This memory map is stored in the IMAINM array, which will contain the simulated contents of target computer main memory throughout the simulation. The data formats and deck set-up for input of the target program are also given in Appendix I.

3. Instruction Parse and Execute. The nucleus of the SUMC interpretive simulator is made up of two subroutines which simulate the instruction fetch and execute operations. The first of these subroutines, FECHM, performs the following functions:

- Validate instruction address
- Fetch one-, two-, or three-halfword instruction from simulated SUMC main memory
- Classify current instruction and parse contents accordingly
- Validate all instruction operand addresses and fetch appropriate data.

The second subroutine, OPDEF, is called if FECHM is correctly executed and performs the following functions:

- Interpretively execute the instruction
- Validate simulated execution
- Place results in appropriate registers in target computer form
- Update timers and statistics variables.

The FECHM and OPDEF subroutines act in a supervisory capacity during the execution of each target instruction. That is, all basic operations required during the instruction execution phase are performed within the FECHM and OPDEF routines; however, frequently used or mundane arithmetic operations are processed by called subroutines or function subprograms. In addition, in the presence of error conditions or interrupts, control is transferred to an appropriate service subroutine.

A flow diagram of the basic FECHM subroutine is shown in Figure III-5. The diagram is simplified but nevertheless illustrates all basic operations and control functions executed by FECHM. An explanation will be given here of the basic steps followed in performing the simulated fetch operation.

a. The initial action taken following a CALL to the FECHM subroutine is the validation of the current instruction address which is represented by the integer variable PCNTR. Two checks are made--the first test determines whether PCNTR is larger than the maximum number of addressable locations in simulated main memory and the second test determines whether PCNTR addresses a memory location which falls within the target program area of simulated main memory. If either of the tests fail, an error flag is set to identify the anomaly and an error termination routine, ERINS, is called.

b. If PCNTR addresses a valid target program location, the instruction is fetched from IMAINM in halfword segments. Immediately after a fetch of the first halfword and the extraction of the instruction op code, a check is performed to determine (1) the validity of the op code and (2) the instruction classification. If the op code is invalid, the program is terminated by setting the appropriate error flag and calling ERINS. In the absence of any errors, the remaining one or two halfwords which make up the complete instruction are fetched from IMAINM (except in the case of an RR instruction which is comprised on only a single halfword). The number of helfwords making up a particular instruction is of course a function of the op code.

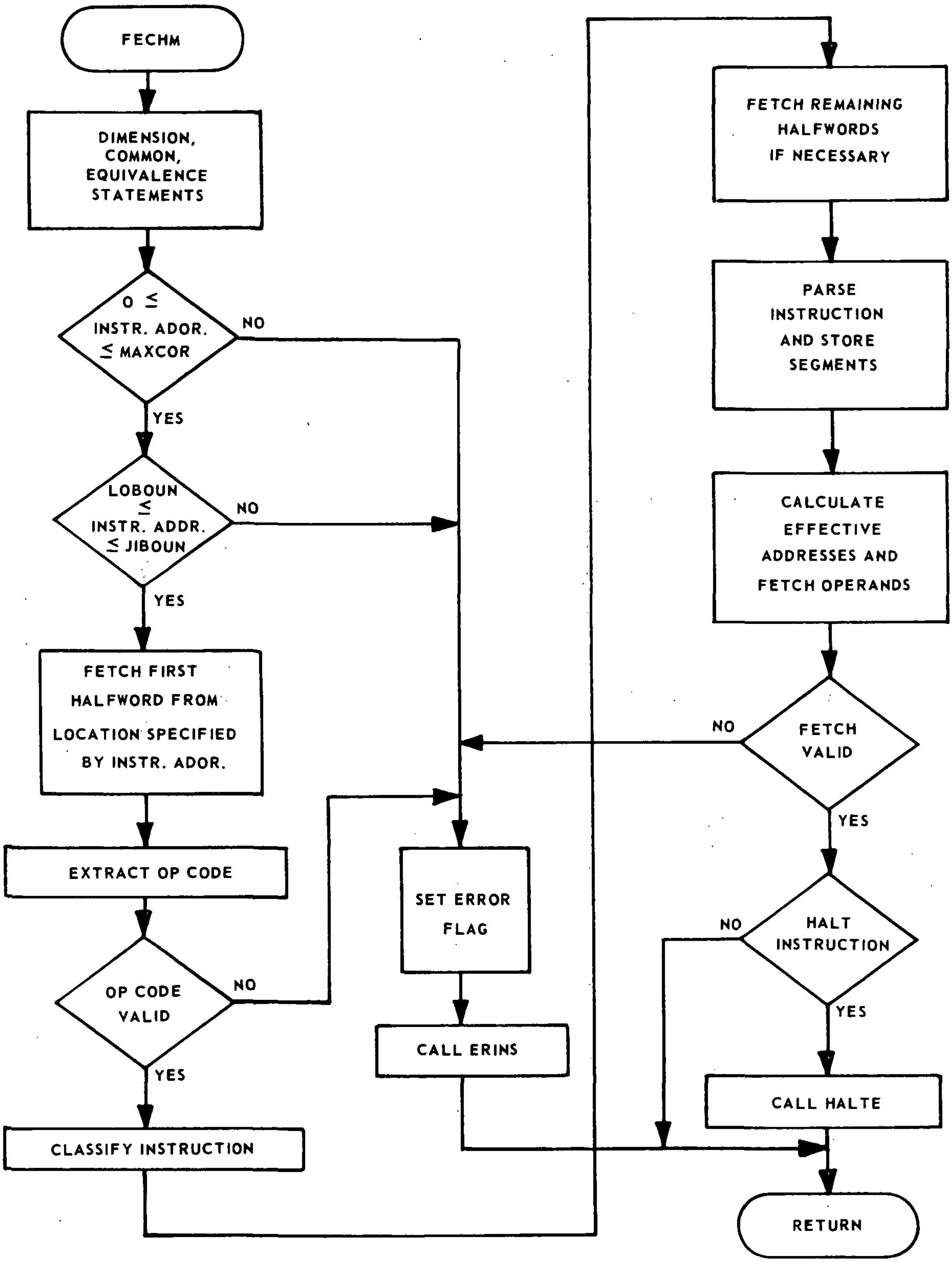


FIGURE III-5. FLOW DIAGRAM OF FECHM ROUTINE.

c. As each instruction halfword is fetched, the information contained in each segment of the halfword is parsed out and stored in a separate array, ISEGTA, which is used to store all components of the current instruction as FORTRAN variables. In addition, the effective addresses of any instruction memory operands are calculated and these operands, if any, are fetched from IMAINM and stored as FORTRAN variables.

d. As the current instruction is fetched from IMAINM and parsed by the FECHM subroutine, each instruction operand or address is validated before proceeding further. If errors or interrupt conditions are detected, appropriate flags are set to identify the source and service subroutines are called. Unless a target program HALT instruction is being processed, control is relinquished to the MAINLIN supervisor routine immediately after the fetch is completely validated.

A flow diagram of the basic OPDEF subroutine is shown in Figure III-6. This subroutine will only be called following the successful completion of an instruction fetch and simulates the execution of the current target instruction. The basic steps followed during OPDEF execution are explained in the following paragraphs.

a. After initializing the program constants which are needed by the OPDEF subroutine, a computed GO TO transfers control to that portion of the routine which will interpretively execute the fetched instruction. Since the operation to be performed by the current instruction is uniquely defined by the instruction op code, it is this parameter which is used as the transfer control variable.

b. During the execution of an instruction, both intermediate and final results are checked to determine whether an error has occurred or an interrupt condition is present. In either case, the identifying flags will be set and the appropriate error termination or interrupt service routine will be called.

c. Although the interpretive nature of the simulator allows the use of host computer hardware for efficient execution of each instruction, all instruction results must be stored in the proper simulated

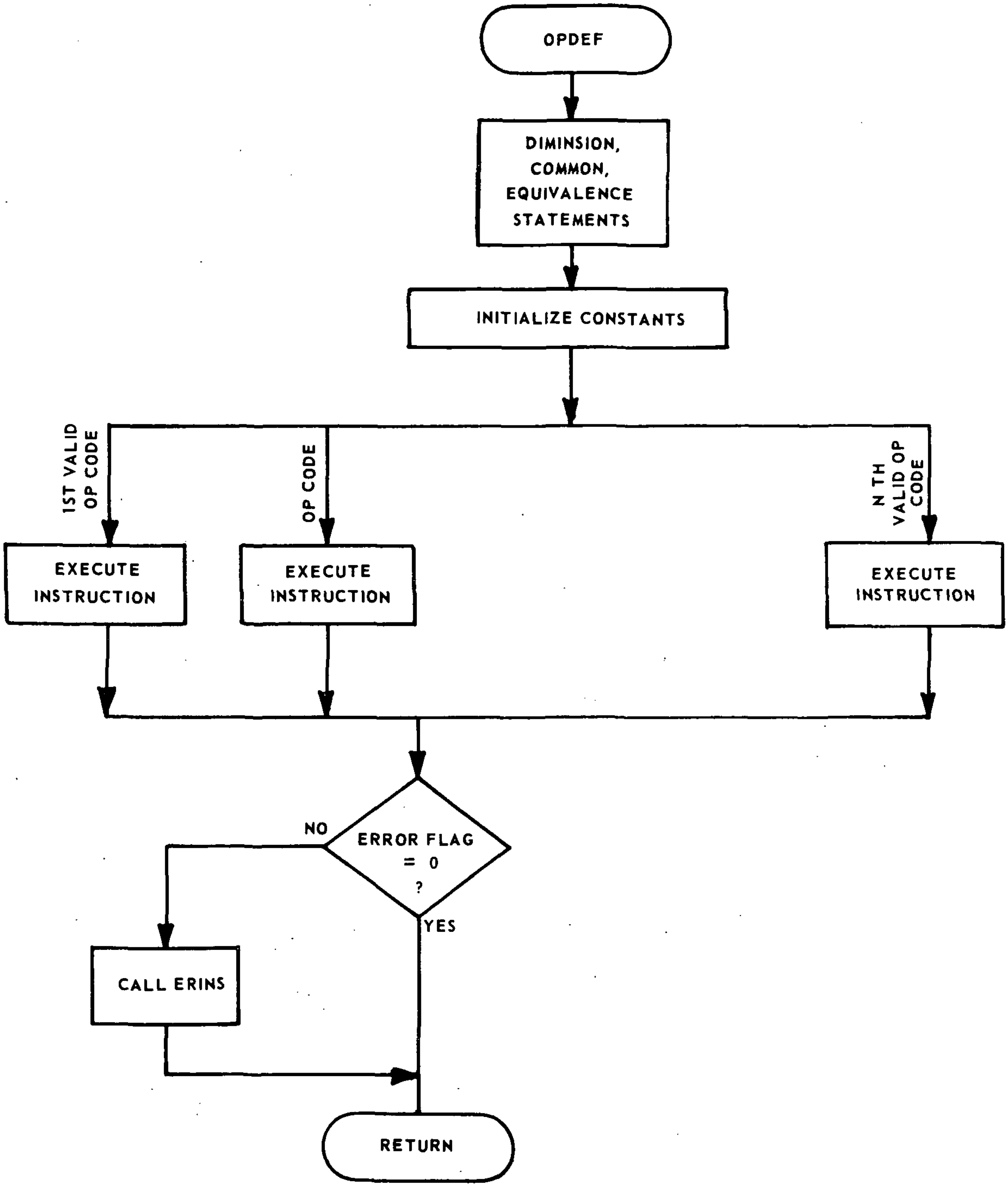


FIGURE III-6. FLOW DIAGRAM OF OPDEF

registers in exact SUMC target computer form at the conclusion of each instruction. That is, fidelity in yielding the correct result for an arbitrary instruction is the criterion rather than fiedlity in executing the precise SUMC sequence to obtain the result. Furthermore, determination of the validity of a result occurs at the level of visibility to the programmer. This means that during the execution sequence, the simulator maintains the contents of computer storage that are available to the programmer but does not necessarily maintain registers, status indicators and other computer storage not available for reference by the programmer.

d. Following the error-free execution of any instruction and a return to the MAINLIN supervisory routine, a CALL is issued to the TIMER subroutine in order to update simulation statistics. Figure III-7 shows the basic flow diagram for the TIMER routine and the following paragraphs give further details concerning statistics updating.

- The program variable IOFFST is an instruction counter and is incremented by one following the successful execution of each target program instruction.
- The complete set of all target computer instructions has been divided into ten distinct classes for statistics purposes. These classes are:

  (1) Register-Register exclusive of other categories
  (2) Arithmetic
  (3) Logical
  (4) Testing
  (5) Branch
  (6) Shift
  (7) Input/Output
  (8) Special
  (9) Privileged
  (10) Executive Call

  A count of the number of target instructions of each class which have been executed is kept current during a simulation. The TIMER routine increments the appropriate class count by one following each instruction execution. The

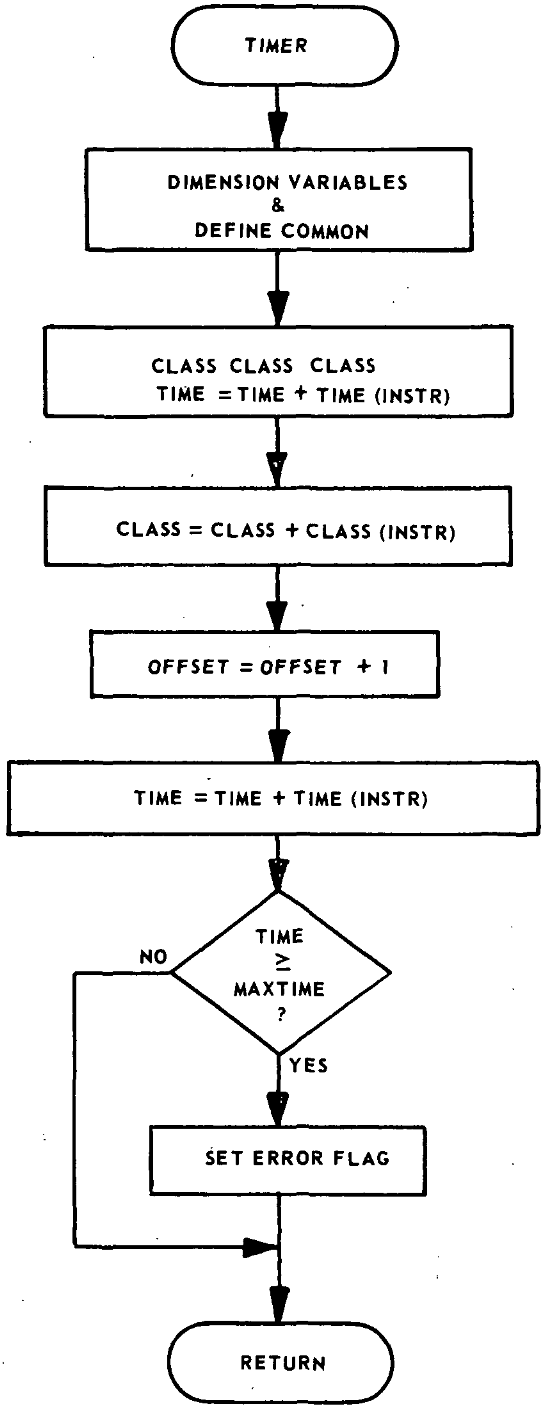


FIGURE III-7. FLOW DIAGRAM OF TIMER ROUTINE total number of instructions processed in each class is available when the simulation is terminated.

- The total simulated execution time is also computed at the conclusion of each instruction execution. This total is computed by adding the time which would be required by the target computer for executing the current instruction to the accumulated time.

- In a similar manner, the simulated execution time associated with each of the instruction classes mentioned above is computed by TIMER. When the simulation is terminated, the total execution time attributed to each of the ten classes of instructions is available.

If the current simulated execution time computed by the TIMER subroutine is less than the allowable maximum execution time, the program executes a normal return to the MAINLIN routine. In this event, barring any pending interrupts, another simulation cycle begins with an instruction fetch. If the accumulated execution time exceeds the specified maximum time, an error flag is set and the program is terminated.

4. User Diagnostic Aids. The SUMC interpretive simulator includes five basic types of diagnostic routines which the user may take advantage of:

- SNAPSHOTS of selected target main memory locations
- TRACE of contents of key target registers
- DUMP of contents of scratch pad memory
- BLOCK DUMP of selected portion of target main memory
- FULL DUMP of target main memory

To provide the necessary user control over the simulator diagnostics when operating on the IBM 7094 host computer, a group of diagnostics data cards must be included as part of the simulator input. These data cards are read by the program during the execution of the initialization routine, INITLZ, and serve to activate or deactivate each of the above diagnostic aids. When any of the diagnostic routines is activated, supporting numerical data must also be included in the input data to provide control information which triggers the diagnostic execution at the desired time.

The simulator includes separate subroutines which perform diagnostics checking and processing following each simulated instruction execution. These routines are called by the primary control loop, MAINLIN, to collect any data requested by the user at that particular point in the program just prior to the execution of the current target instruction. The following sections will give a detailed explanation of each of the five diagnostics functions which are performed by the simulator.

a. SNAP diagnostic. This diagnostic feature allows the user to obtain a printout of up to nine selected main memory locations at some predetermined point in the program. A CALL SNAPRD statement in the program initialization routine transfers simulator control to the subroutine, SNAPRD, which reads all SNAP information supplied by the user. A flow diagram of the SNAPRD subroutine is shown in Figure III-8. SNAPRD inputs data as follows:

(1) READ SNAP. A single logical variable, SNAP, is read first and, if its value is true, additional SNAP data is sought. If its value is false, the user desires no SNAP diagnostics for the program under test and SNAPRD relinquishes control back to INITLZ.

(2) READ FSNAP. This variable is given the value true if the user wishes to obtain a snapshot of specified main memory locations following each target instruction execution. Two additional data cards must be present when a full snap is specified--one card which specifies the number of MM locations to be snapped and the following lists the target memory addresses whose contents are to be printed. In addition, if a FULL SNAP has been requested, no other SNAP diagnostic may be present during the simulation. Therefore, control is transferred back to INITLZ after the FULL SNAP data has been read.

(3) READ TISNAP. If a FULL SNAP has not been requested, other SNAP diagnostics are checked, beginning with the TIME-INTERVAL SNAP.

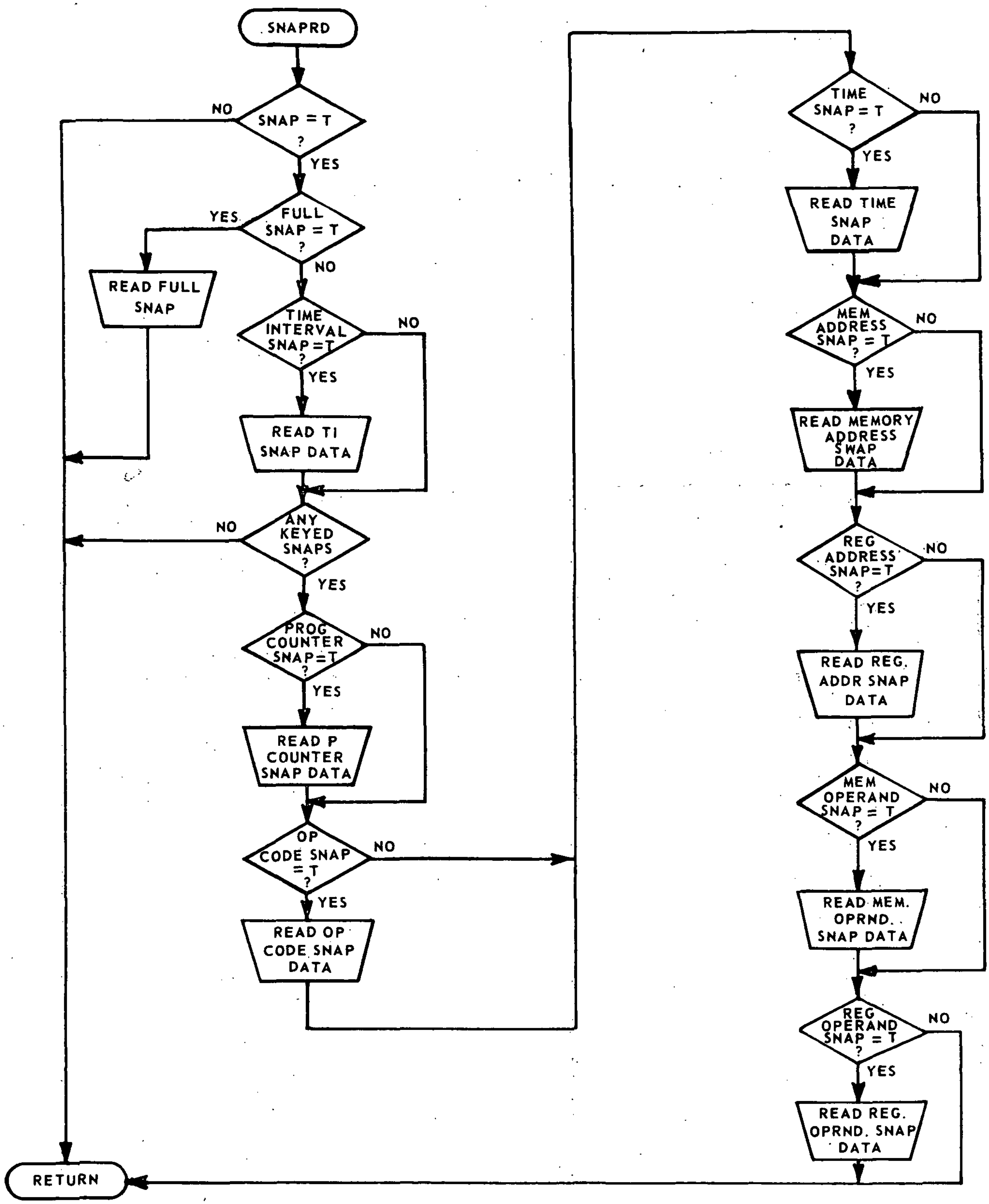


FIGURE III-8. FLOW DIAGRAM OF SNAPRD ROUTINE

For proper execution of this diagnostic, if activated, the user must specify the number of memory locations to be snapped and their addresses as well as the starting time for the SNAP execution, the time interval between each SNAP, and the time at which the final SNAP is to occur. For simplicity, only one time-interval SNAP routine may be requested for a particular simulation.

(4) READ KSNAP. A single logical variable is read at this point to determine whether any keyed SNAP diagnostics are desired during program execution. If KSNAP=.FALSE., no keyed SNAP diagnostics are wanted and control is transferred back to INITLZ since only keyed SNAP data remains to be read by SNAPRD. If KSNAP-.TRUE., additional data cards must be read by SNAPRD which indicate the SNAP keys and corresponding memory locations to be printed.

(5) READ PCSNAP. This is the first of seven keyed SNAP diagnostics which must be activated or deactivated whenever the previously mentioned KSNAP variable indicates the presence of one or more keyed SNAP requests. Whenever PCSNAP=.TRUE., a program-counter-keyed SNAP is desired by the user and data is read specifying the program counter values which act as triggers along with the corresponding memory locations whose contents are to be snapped. PCSNAP=.FALSE. indicates the absence of any program-counter-keyed SNAP requests or data pertaining to such.

(6) READ OCSNAP. The OCSNAP logical variable indicates the presence or absence of an op-code-keyed SNAP request. If OCSNAP=.TRUE., an op-code-keyed SNAP diagnostic is wanted by the user and additional data is read specifying op code values which act as triggers and corresponding memory locations whose contents are to be snapped. OCSNAP=.FALSE. will of course require no additional data.

(7) READ TSNAP. This logical variable indicates the presence/absence of any time-keyed SNAP requests. If TSNAP=.TRUE., one or more time-keyed SNAP diagnostics are wanted and additional data is read specifying at which simulated execution times the SNAP data is to be collected and also the memory locations whose contents are to be snapped.

(8) READ MASNAP. The logical variable MASNAP indicates the presence/absence of any memory-address-keyed SNAP diagnostics. If MASNAP=.TRUE., memory-address-keyed SNAP diagnostics are wanted and appropriate data is read specifying the target memory locations to be snapped. MASNAP=.FALSE. requires no additional data.

(9) READ RASNAP. The presence or absence of register-address-keyed SNAP diagnostics is indicated by the RASNAP variable. If RASNAP=.TRUE., data must be included giving the target instruction register address keys which trigger the SNAP and the memory locations to be snapped. No additional data is required when RASNAP=.FALSE.

(10) READ MOSNAP. The logical variable MOSNAP indicates the presence/absence of memory-operand-keyed SNAP diagnostics. This diagnostic is identical to the memory-address-keyed SNAP with the exception that the contents of the specified memory address act to trigger the SNAP. Therefore, when MOSNAP=.TRUE., the octal contents of each memory address must also be included in the necessary data.

(11) READ ROSNAP. The presence/absence of register-operand-keyed SNAP diagnostics is indicated by the logical variable ROSNAP. This diagnostic is identical to the register-address-keyed SNAP with the exception that the contents of the specified register address must be included in the necessary data when ROSNAP=.TRUE. and act to trigger the SNAP.

When any of the SNAP diagnostics discussed above are activated, two additional data values must be specified. First, for each set of memory locations which are to have their contents printed at SNAP execution time, the number of memory locations specified to be SNAPPED must be given. Second, the number of SNAP keys of each type, i.e., program-counter-keys, op-code-keys, etc., must be specified whenever a keyed SNAP diagnostic is activated.

Appendix I contains a detailed layout of the data cards which make up the SNAP diagnostics data deck. This layout gives a brief explanation of each data card which may be present as well as indicating proper card formats, proper card sequence, and data needed to activate any combination of the available SNAP diagnostics.

The SNAPEX subroutine is part of the primary simulator control loop, MAINLIN, and is called just prior to the execution of each target instruction. This routine is responsible for checking all SNAP diagnostic keys to determine whether a SNAP response is appropriate. If the user, by supplying the proper SNAP diagnostics data cards, has requested a SNAPSHOT of certain target main memory locations at this point in the program, this data is collected for printout by the SNAPEX routine before the pending target instruction is executed.

b. TRACE diagnostic. This diagnostic feature makes it possible for the user to obtain a printout of the contents of key target computer registers at a predetermined point in the program. The user specifies the TRACE diagnostic keys, which serve to trigger the TRACE printout at the proper time, through a set of TRACE data cards that are read by the TRACRD subroutine. The TRACRD routine is called during program initialization and a flow diagram of this routine is shown in Figure III-9. TRACRD inputs data as follows:

(1) READ TRACE. The first value which is read by the TRACRD routine is the logical variable TRACE, and if its value is logical .TRUE., additional TRACE data is sought. If TRACE=.FALSE., the user desires no TRACE diagnostics for the program and TRACRD relinquishes control to INITLZ.

(2) READ FTRACE. If the logical variable FTRACE=.TRUE., the user will obtain a trace of the contents of all key target computer registers following each target instruction execution. If this FULL TRACE is activated, no other TRACE diagnostics can be specified since their presence would simply yield redundant TRACE information. Only when FTRACE=.FALSE. does the routine search for other TRACE diagnostic data.

(3) READ TITRAC. A time-interval TRACE is requested through the logical variable TITRAC. This diagnostic yields a TRACE of the key target computer registers at a specified time interval beginning and

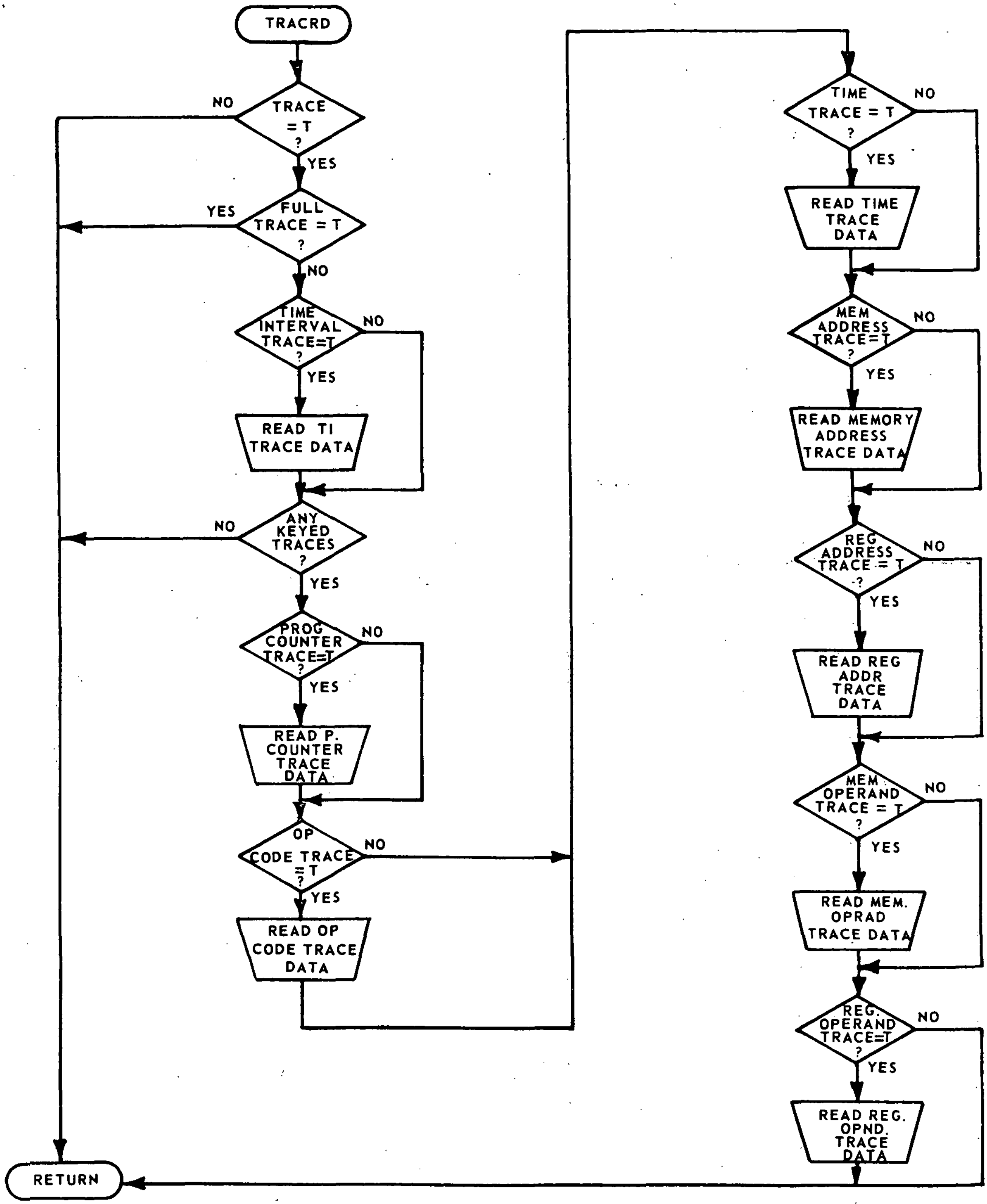


FIGURE III-9. FLOW DIAGRAM OF TRACRD ROUTINE ending at those times specified by the user. If TITRAC=.TRUE., an additional data card must be present to supply TRACE start time, stop time, and time interval.

(4) READ KTRACE. A single logical variable is read at this point to determine whether any keyed TRACE diagnostics are desired during program execution. If KTRACE=.FALSE., no keyed TRACE diagnostics are wanted and control is transferred back to INITLZ since only keyed TRACE data remains to be read by TRACRD. If KTRACE=.TRUE., additional data cards, as discussed below, are read by TRACRD in order to input all TRACE keys.

(5) READ PCTRAC. This is the first of seven keyed TRACE diagnostics which must be activated or deactivated whenever the previously mentioned KTRACE variable indicates the presence of one or more keyed TRACE requests. Whenever PCTRAC=.TRUE., a program-counter-keyed TRACE is desired by the user and data is read specifying the program counter values which are to act as triggers along with the number of consecutive instructions which are to be traced. PCSNAP=.FALSE. indicates the absence of any program-counter-keyed TRACE requests or data pertaining to such.

(6) READ OCTRAC. The OCTRAC logical variable indicates the presence or absence of an op-code-keyed TRACE request. If OCTRAC=.TRUE., an op-code-keyed TRACE diagnostic is wanted by the user and additional data is read specifying op code values which act as triggers and the corresponding number of instructions to be traced. OCTRAC=.FALSE. of course requires no additional data.

(7) READ TTRACE. This logical variable indicates the presence/absence of any time-keyed TRACE requests. If TTRACE=.TRUE., one or more time-keyed TRACE diagnostics are wanted and additional data specifies at which simulated execution times the TRACE data is to be collected and also the number of instructions for which each TRACE is to be in effect.

(8) READ MATRAC. The logical variable MATRAC indicates the presence/absence of any memory-address-keyed TRACE diagnostics. If

MATRAC=.TRUE., memory-address-keyed TRACE diagnostics are wanted and appropriate data is read specifying the target instruction memory addresses which trigger the TRACE output and the number of instructions to be traced. MATRAC=.FALSE. requires no additional data.

(9) READ RATRAC. The presence or absence of register-address-keyed TRACE diagnostics is indicated by the RATRAC variable. If RATRAC=.TRUE., data must be included giving the target instruction register address keys which trigger the TRACE and the corresponding number of instructions to be traced. No additional data is required when RATRAC=.FALSE..

(10) READ MOTRAC. The logical variable MOTRAC indicates the presence/absence of memory-operand-keyed TRACE diagnostics. This diagnostic is identical to the memory-address-keyed TRACE with the exception that the contents of the specified target instruction memory address act to trigger the TRACE. Therefore, when MOTRAC=.TRUE., each TRACE key includes a target memory address, its corresponding contents, and the number of instructions to be traced.

(11) READ ROTRAC. The presence/absence of register-operand-keyed TRACE diagnostics is indicated by the logical variable ROTRAC. This diagnostic is identical to the register-address-keyed TRACE with the exception that the contents of the specified register address must be included in the necessary data when ROTRAC=.TRUE. and act to trigger the TRACE.

Appendix I contains a detailed layout of the data cards which make up the TRACE diagnostics data deck. This layout gives a brief explanation of each data card which may be present as well as indicating proper card formats, proper card sequence, and data needed to activate any combination of the available TRACE diagnostics.

The TRACEX subroutine is called by the simulator primary control loop, MAINLIN, just prior to the execution of each target instruction. This routine is responsible for checking TRACE diagnostics keys (if any) to determine whether a register TRACE is appropriate. If the user, by supplying the proper TRACE diagnostics data cards, has requested a TRACE of the key target computer registers at this point in the program, this data is collected for subsequent printout by the TRACEX routine before the pending target instruction is processed.

c. SPM DUMP Diagnostic. This diagnostic feature enables the user to obtain a dump of the contents of SUMC scratch-pad-memory at a predetermined point in the program. The user must specify the SPM DUMP keys, which serve to trigger the dump at the proper time, through a set of SPM DUMP data cards that are read by the SPMDRD subroutine. The SPMRRD subroutine is called during program initialization and a flow diagram of this routine is shown in Figure III-10. SPMDRD inputs data as follows:

(1) READ SPMD. The first value which is read by the SPMDRD routine is the logical variable SPMD, and if its value is logical .TRUE., additional SPM DUMP data is sought. If SPMD=.FALSE., the user does not want a SPM DUMP at any point during the simulation and SPMDRD does not search for additional data but relinquishes control to INITLZ.

(2) READ PCSPMD. This is the first of seven keyed SPM DUMP diagnostics which must be activated or deactivated whenever the previously mentioned SPMD variable indicated the presence of one or more keyed SPM DUMP requests. Whenever PCSPMD=.TRUE., a program-counter-keyed SPM DUMP is desired by the user and two additional data cards must be read. The first card specifies the number of dump keys to be entered as input and the second gives the values of the program counter keys which act as triggers for SPM DUMP execution. PCSPMD=.FALSE. indicates the absence of any program-counter-keyed SPM DUMP requests or data pertaining to such.

(3) READ OCSPMD. The OCSPMD logical variable indicates the presence or absence of any op-code-keyed SPM DUMP requests. If OCSPMD=.TRUE., an op-code-keyed SPM DUMP diagnostic is wanted by the user and two additional data cards specify (a) number of op code keys to be entered an input and (b) values of the op code keys which act as SPM DUMP triggers. OCSPMD=.FALSE. of course requires no additional data.

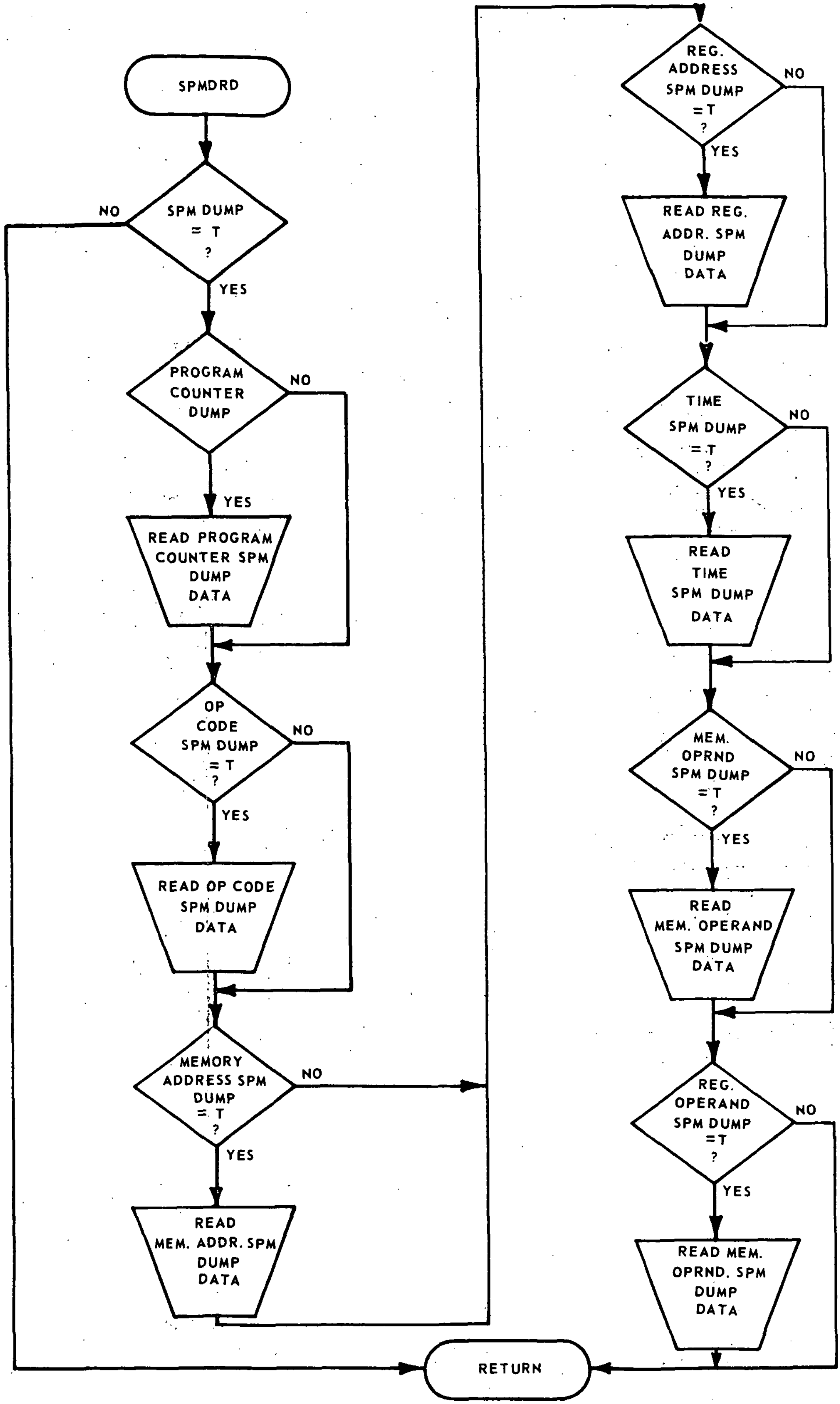


FIGURE III-10. FLOW DIAGRAM OF SPMDRD ROUTINE

(4) READ MASPMD. The logical variable MASPMD indicates the presence/absence of any memory-address-keyed SPM DUMP diagnostics. If MASPMD=.TRUE., memory-address-keyed SPM DUMP diagnostics are wanted and two additional data cards specify (a) number of memory address keys to be entered and (b) instruction memory addresses which will act as keys to trigger a SPM DUMP. MASPMD=.FALSE. requires no additional data.

(5) READ RASPMD. The presence or absence of register-address-keyed SPM DUMP diagnostics is indicated by the RASPMD variable. If RASPMD=.TRUE., one or more register-address-keyed SPM DUMP requests are present and two additional data cards specify (a) number of register address keys to be entered and (b) instruction register addresses which will act as keys to trigger a SPM DUMP. RASPMD=.FALSE. requires no additional data cards.

(6) READ TSPMD. This logical variable declares the presence/absence of time-keyed SPM DUMP diagnostic requests. If TSPMD=.TRUE., time-keyed SPM DUMP diagnostics are wanted by the user and additional data includes (a) number of time keys to be entered and (b) simulated elapsed time at which each SPM DUMP is to be executed.

(7) READ MOSPMD. The logical variable MOSPMD indicates the presence/absence of any memory-operand-keyed SPM DUMP diagnostics. If MOSPMD=.TRUE., SPM DUMP diagnostics which are keyed by instruction memory operands are wanted by the user and two additional data cards supply (a) number of memory operand keys to be entered and (b) the instruction memory addresses and their corresponding contents which will act as keys to trigger the desired SPM DUMP executions.

(8) READ ROSPMD. The logical variable ROSPMD indicates the presence/absence of register-operand-keyed SPM DUMP diagnostics. If ROSPMD=.TRUE., SPM DUMP diagnostics are wanted which are keyed by instruction register operands and two additional data cards specify (a) number of register operand keys to be entered and (b) the instruction register addresses and their corresponding contents which will act as keys to trigger SPM DUMP diagnostics.

A detailed layout of the data cards which comprise the SPM DUMP diagnostics data deck is contained in Appendix I. This layout provides a brief description of each data card which may be present as well as indicating proper card formats, proper card sequence, and data needed to activate any combination of the available SPM DUMP diagnostics.

The SPMDEX subroutine is called by the simulator primary control loop, MAINLIN, just prior to the execution of each target instruction. This routine checks all SPM DUMP diagnostics keys to determine whether a dump of SPM registers is wanted by the user at this point in the program. Whenever a SPM DUMP is appropriate, the data is collected by SPMDEX before the pending target instruction is processed.

d. Block MM Dump Diagnostic. This simulator diagnostic allows the user to obtain a dump of a selected block of contiguous target computer main memory locations at some predetermined point in the program. The user must specify the program counter values which will act to trigger the block MM DUMP at the desired point in the program. All diagnostics data which is needed to control block MM DUMP operations is read by the MMDRD subroutine. This routine is called during program initialization and the flow diagram is shown in Figure III-11.

The first value which is read by the MMDRD routine is the logical variable BLMMD, and if the logical value is .TRUE., additional block MM DUMP data is sought. If BLMMD=.FALSE., the user desires no block MM DUMP diagnostics and further supporting data should not be present.

The supporting data which is required when BLMMD=.TRUE. consists of (a) a single data card which specifies the number of program counter DUMP keys to be entered and (b) a set of data cards (one for each program counter key), each of which contains the program counter value, the block MM DUMP start address, and the number of target main memory locations to be dumped.

Appendix I contains a detailed layout of the data cards which comprise the block MM DUMP diagnostics data deck. The layout provides a brief description of each data card which may be present as well as indicating proper card formats and card sequence.

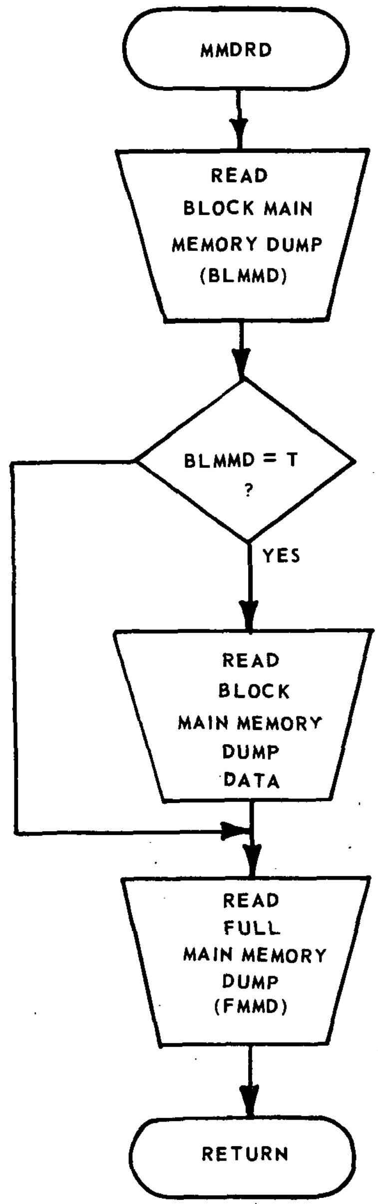


FIGURE III-11. FLOW DIAGRAM OF MMDRD ROUTINE

The execution of a block MM DUMP during simulation of a target program is accomplished by the BLMDEX subroutine. This routine is called by the simulator primary control loop, MAINLIN, just prior to the execution of each target instruction. BLMDEX compares each of the program counter DUMP keys with the current simulated target program counter, and if a match is found, the appropriate data is collected for subsequent printout.

e. Full MM DUMP Diagnostic. The user may obtain a full DUMP of the contents of all target computer main memory locations at the termination of a target program simulation. A full MM DUMP is available only at program termination and is requested by the user through a single data card which is read by the previously mentioned MMDRD routine. As shown in Figure III-11, the MMDRD subroutine reads the logical variable FMMD before returning to INITLZ. If FMMD=.TRUE., a full target MM DUMP is performed at program termination. The value of the FMMD variable is supplied by the user as the last data card in the diagnostics data deck, as shown in the data card layouts of Appendix I.

f. Diagnostics Header Information. All of the diagnostic features which have been discussed supply the user with information concerning the contents of certain target computer (SPM or MM) registers at a particular point in the simulation of a target program. In order to identify the exact point during target program execution that a particular diagnostics output was obtained, a block of header information forms the preamble to all diagnostics printouts. The information supplied by each header block includes:

- type of diagnostic in effect
- number of instructions executed
- simulated elapsed time
- current instruction address
- current instruction contents
- instruction register operands
- instruction memory operands.

5. Program Termination. When a particular SUMC program simulation is terminated, it may be due to any one of a number of different possible causes. The cause of the program termination will have a direct bearing on simulator response and the data which is provided at the conclusion of a program. Whenever possible, the termination cause is identified and a table of program statistics is provided to the user. The following paragraphs will discuss the different simulation conditions which lead to program termination and simulator response for each condition.

a. Invalid Simulation Definition. This error condition results whenever data which is supplied by the user for simulation initialization and control is invalid. The user must supply three types of definition data:

- SUMC target computer architectural parameters
- SUMC register initialization data
- diagnostics data

In the first two cases, erroneous data will not prevent the simulator from beginning target program execution; however, it is not predictable at what point during the simulation this input error will be detected. When detected, it will be identified as a program interruption and the appropriate interrupt response will be taken. In the third case, the host computer input operations will be affected and termination action will depend only on host computer characteristics.

b. Error Interrupts. Two classes of interrupt conditions are considered to be error conditions and will lead to program termination. These are

- Program interruption and
- Machine-check interruption.

A complete discussion of the SUMC interrupt scheme which includes the above two conditions as well as three additional interrupt conditions is given in the next section, Simulator Interrupt Capability.

Whenever either of the error interrupt conditions occurs, appropriate interrupt servicing routines are called by the simulator and upon their completion, the program is terminated. The information supplied to the user at termination includes an error flag identifying the cause and a table of simulation statistics. The error flag is set as follows:

Error Flag	Meaning
1	Operation exception interrupt
2	Privileged-operation exception interrupt
3	Execute exception interrupt
4	Addressing exception interrupt
5	Specification exception interrupt
6	Data exception interrupt
7	Fixed-point-overflow exception interrupt
8	Fixed-point-divide exception interrupt
9	Exponent-overflow exception interrupt
10	Exponent-underflow exception interrupt
11	Significance exception interrupt
12	Floating-point-divide exception interrupt
13	Machine-check interrupt
14	Memory boundary violation
15	Time overflow
16	Wait state

The simulation statistics include information concerning each of the ten classes of instructions previously discussed in Section III-B.3. For termination due to an error interrupt, the following information is made available:

- number of instructions of each class which were executed
- amount of time used in executing each class of instruction
- percentage of total simulated elapsed time used in executing each class of instruction
- total number of instructions executed
- total simulated elapsed time

c. Simulation Errors. In addition to detecting those error conditions which lead to a target computer interrupt response, the simulator checks two additional error conditions which also result in program termination. These are:

- target computer memory boundary violation
- simulated elapsed time overflow

A boundary violation will occur whenever instructions or data is addressed by the target program and this address exceeds the target computer core size. A time overflow occurs when the simulated elapsed time exceeds the maximum program execution time specified by the user prior to beginning the simulation. When either of these errors is detected during a simulation, the program is terminated immediately. A printout of the error flag value identifies the termination cause for the user and the normal statistics information is also printed following this type of program termination.

d. Wait State. If the SUMC simulator executes an instruction which places the target machine in the WAIT state, the simulation response will be identical to that obtained due to normal target program completion. The single exception is that an error flag is printed indicating the WAIT state.

e. Target Program Completion. A normal termination of the simulator program occurs when all target program instructions have been executed in an error-free fashion. Unless the user has requested specific diagnostics, the only information necessary at the time of a normal program completion will be the usual statistics table.

The present version of the SUMC simulator does not possess a restart capability although provisions have been made to add this feature at a later date. This feature will of course have its greatest impact on the program termination portions of the simulator. Future restart capabilities will be discussed in detail in a later section which deals with possible simulator enhancements.

6. <u>SUMC Simulator Interrupt Capability</u>. The SUMC computer which presently acts as the target machine for the current version of the simulator has an interrupt scheme modeled after that of the IBM 360 system. Under this setup, five classes of interrupt conditions are present, which are input/output, program, supervisor-call, external, and machine check interruptions.

An interruption consists of storing the current Program Status Word (PSW) as an old PSW and fetching a new PSW. Interruptions are taken only when the CPU is interruptible for the interruption source. The system mask can be used to mask I/O and external interruptions; the program mask can be used to mask three of the twelve program interruptions; and the machine-check mask can be used to mask machine-check interruptions.

The simulator checks for interruptions after one instruction interpretation is finished and before a new instruction interpretation is started. This check is performed within the primary control loop each time the simulator returns from an interpretive instruction execution. The action taken by the simulator upon the detection of an interrupt condition is as follows:

- When any type of interrupt is detected, the INTRPS subroutine is called for the purpose of maintaining the stack of pending interrupts. The current interrupt is added to the stack according to its predetermined servicing priority. The INTRPS subroutine is also responsible for maintaining the interrupt stack whenever it is modified due to a pending interrupt being "pulled" for servicing.
- The IMMPSW routine is called to convert the current PSW to an old PSW format and store this PSW in the appropriate main storage location.
- The ISPPSW routine is called to convert the new PSW in main storage into the current PSW format and store this PSW in simulated scratch pad memory.

- When an interruption is detected, the instruction which is currently being executed may or may not be completed depending on the type of interruption. Furthermore, interruptions caused by error conditions will result in a call of the ERINS subroutine which identifies the anomaly to the user and subsequently terminates the simulation program.
- If the interruption is not due to an error condition, an interrupt service routine, INTSER, is called which informs the user that the interrupt has occurred, before beginning the execution of the next instruction. The INTSER routine will eventually be expanded to perform all simulated interrupt servicing operations according to the interruption action of the simulation target computer. The present version of the SUMC simulation program simulates only interrupt detection and stack operations.

A summary of the target computer interruption conditions which must be checked by the simulator are given in Table III-6. This table lists, for each interruption source, the interruption code, system mask bits, interruption-length code, operation execution and simulation execution.

As discussed above, the present version of the interpretive simulator models the target SUMC interrupt detection and stacking operations but does not simulate the actual interrupt servicing operations. The interrupt servicing routines are highly dependent on the particular target computer being simulated and their implementation is planned for a later date, as outlined in the next section covering future simulator enhancements.

7. Simulator Utility Routines and Functions. There are a number of subroutines included in the simulation program which perform generic operations and which are used by different simulator modules. The following paragraphs give brief descriptions of these utility routines along with an explanation of their function.

Table III-6. SUMC Interruption Conditions

Source Identification	Interruption Code PSW Bits 16-31	Mask Bits	ILC Set	Operation Execution	Simulation Execution
INPUT/OUTPUT (OLD PSW 56, NEW PSW 120, PRIORITY 4)					
Channel 1	00000001 aaaaaaaa	0	x	completed	continued
PROGRAM (OLD PSW 40, NEW PSW 104, PRIORITY 2)					
Operation	00000000 00000001		1,2,3	suppressed	terminated
Privileged Oper.	00000000 00000010		1,2	suppressed	terminated
Execute	00000000 00000011		2	suppressed	terminated
Addressing	00000000 00000101		0,1,2,3	suppressed	terminated
Specification	00000000 00000110		1,2	suppressed	terminated
Data	00000000 00000111		2	terminated	terminated
Fixed-point ovfl.	00000000 00001000	36	1,2	completed	terminated
Fixed-point div.	00000000 00001001		1,2	suppressed	terminated
Exponent ovfl.	00000000 00001100		1,2	completed	terminated
Exponent unfl.	00000000 00001101	38	1,2	completed	terminated
Significance	00000000 00001110	39	1,2	completed	terminated
Floating-point div.	00000000 00001111		1,2	suppressed	terminated
SUPERVISOR-CALL (OLD PSW 32, NEW PSW 96, PRIORITY 2)					
Instruction bits	00000000 rrrrrrrr		1	completed	continued
EXTERNAL (OLD PSW 24, NEW PSW 88, PRIORITY 3)					
Interrupt key	00000000 nlnnnnnn	7	x	completed	terminated
MACHINE CHECK (OLD PSW 48, NEW PSW 112, PRIORITY 1)					
Machine malfunction	cccccccc cccccccc	13	x	terminated	terminated

NOTES: a = device address bits

r = bits of R_1, R_2 field of SVC instruction

n = other external-interruption conditions

c = target computer-dependent bits

x = unpredictable

IAND(I,J) is a function subprogram which logically AND's the contents of the two locations specified by the arguments of the function and returns this result to the calling subprogram.

IOR(I,J) is a function subprogram which logically OR's the contents of the two locations specified by the arguments of the function and returns the result to the calling subprogram.

IER(I,J) is a function subprogram which performs the logical EXCLUSIVE OR of the contents of the locations specified by the arguments of the function and returns this result to the calling subprogram.

INTNOT(K) is a function subprogram which complements the contents of the location specified by the argument and returns this result to the calling subprogram.

ITWTSM(I) is a function subprogram which converts the two's-complement value of the argument to signed-magnitude representation and returns this result to the calling subprogram.

ISMTWO(I) is a function subprogram which converts the signed-magnitude value of the argument to two's-complement representation and returns this result to the calling subprogram.

ILOAD(SOURCE,SB,NB) is a function subprogram which will move a field of data from the source word and will right-justify it as the output argument. The remaining part of the output argument word will be filled with zeros.

ISTORE(SRC1,SRC2,SB,NB) is a function subprogram which will move a right-justified field of data of NB bits in length from SRC1 and will scale to position SB. This field will then replace the same scaled field portion of word SRC2. The word then formed becomes the output argument and is returned to the calling subprogram.

JEXBIN(IBUF,IST,ILNG) is a function subprogram which converts a hexadecimal character string to a binary representation. The conversion result is the binary equivalent of the ILNG hexadecimal characters beginning with character IST of the string IBUF.

I2T1(I) and I1T2(I) are one's-complement : two's-complement conversion routines which will be implemented for later versions of the simulator. Present operation on the IBM 7094 host system, a signed-magnitude arithmetic machine, does not require the use of these routines.

ICOMP1(SOURCE,SB,NB) is a function subprogram which will complement a field of data in the source word. The NB bits beginning at bit position SB of the source word are one's-complemented in the output argument.

C. Recommended Expansion and Enhancements

1. Input/Output Simulation. The present version of the SUMC interpretive simulator does not perform simulated I/O operations. The I/O processing characteristics of the SUMC target computer will vary greatly from one target machine to the next and, for this reason, an I/O simulation program package would not be appropriate for the basic SUMC simulation program. What would be appropriate, however, is a general purpose I/O interface routine which would form part of the basic simulator. This interface routine would coordinate I/O simulation operations between the basic simulator program and various peripheral device simulation subroutines which would be needed for a particular application.

Figure III-12 contains a diagram which indicates the communication paths which would be necessary for an I/O simulation scheme as discussed above. Under this setup, the SUMC simulator would maintain status information in the COMMON scratch pad memory array and also place and retrieve data in the product-remainder register (PRR) location. The peripheral device simulation routines would handle all I/O data transfers through the PRR while under control of the SPM status registers.

I/O simulation procedures will depend heavily on the particular application; however, an I/O interface routine for the SUMC simulator will perform at least the following functions:

- Set/Reset appropriate status registers

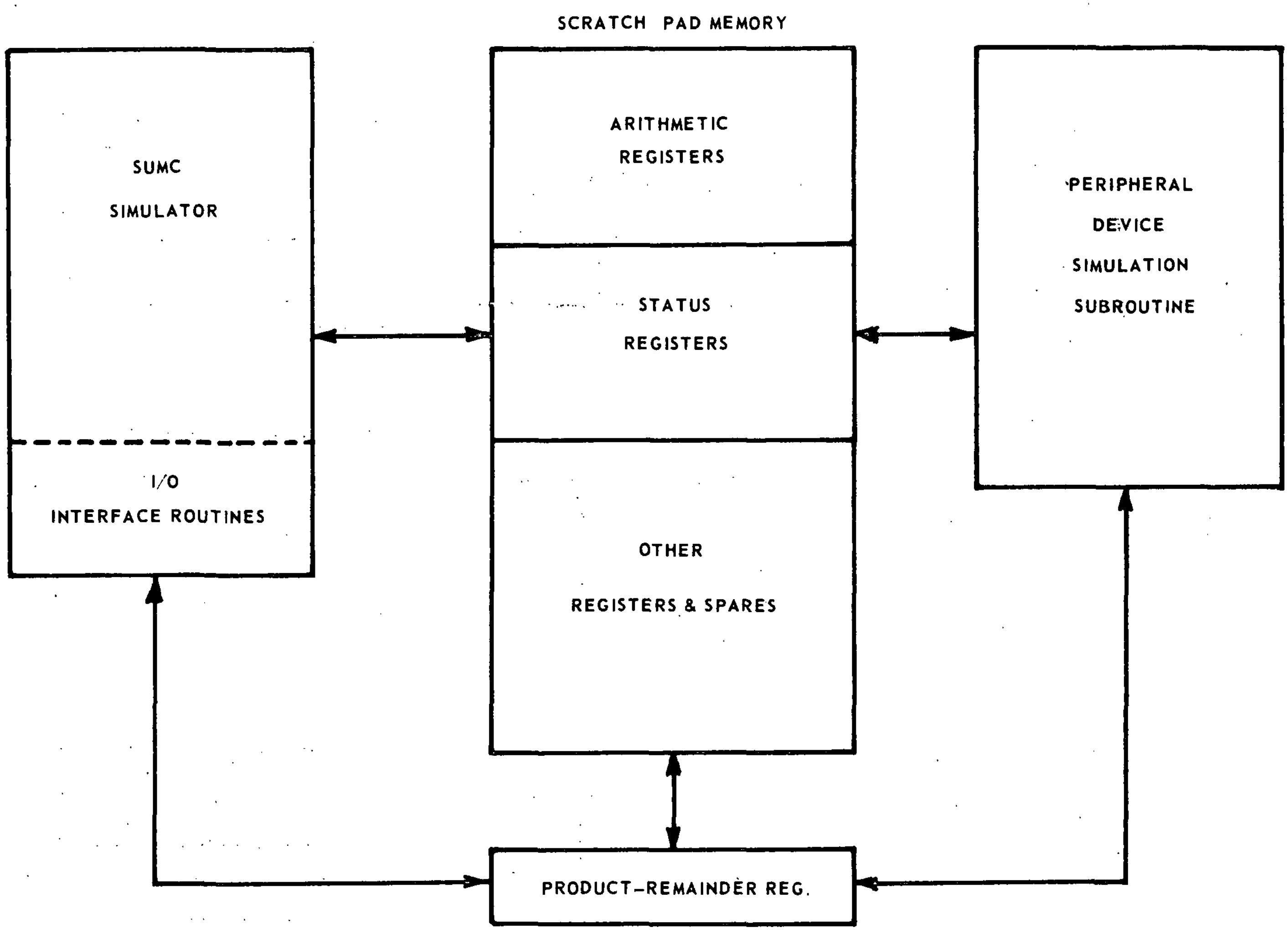


FIGURE III-12. SIMULATOR -- PERIPHERAL DEVICE SIMULATION COMMUNICATION

- Place/Retrieve data in the PRR
- Issue CALL's to specified peripheral device simulation routines.

Peripheral device simulated elapsed time will be maintained by each device simulation routine and I/O task completion must signal a RETURN to the primary control loop of the SUMC simulator. All I/O processing routines are of course triggered by I/O interruptions of the target computer CPU.

2. Simulator Execution Efficiency. The SUMC instruction simulator has been designed with generality in mind as a key simulation objective. It is meant to be a basic, general-purpose simulator in two respects:

- The simulator must have the capability to model a family of SUMC target machines, i.e., this computer family is expected to consist of machines with architectures and design parameters which may vary with the envisioned application.
- The simulator must be easily transferrable between different host computers, i.e., the SUMC simulator development is being done using an IBM 7094 host system with plans for production runs on a Univac 1108 system. (In addition, it is expected that the simulator should be easily modified to eventually operate on several other large-scale commercial host systems.)

The desired generality of simulator design cannot be realized without paying the penalty of decreased execution efficiency. This inherent design tradeoff on generality and efficiency can only be resolved through carefully defined simulation objectives and continuing studies of the basic simulator's execution efficiency. These studies are planned following the initial simulator implementation and programming changes will be made accordingly.

If the price to achieve a general-purpose simulator has been too great, there are several areas which may be investigated for enhancement of simulator execution efficiency. These are:

- Rewriting any existing program code which has not already been written in the most efficient form.
- Converting basic subroutines from the standard FORTRAN IV source language to a more efficient machine language code whenever this will enhance execution speed.
- Employing host machine hardware to the maximum possible extent.
- Sacrificing any target computer generalities which are not explicitly desired.
- Employing the capabilities of the simulator supervisory I/O routines in lieu of the present FORTRAN I/O operations.

3. Target Instruction Set. The instruction set which has been implemented for the current SUMC target computer parallels that of the IBM 360 system. The present version of the SUMC simulator executes a selected subset of this set of machine instructions. If desired, the SUMC simulator could be expanded to execute the complete instruction set used by the IBM 360 system. This would basically require the addition of floating-point, I/O, and decimal arithmetic instructions to the present simulator. Table III-7 shows a complete list of all IBM 360 instructions and those which have been included in the present version of the simulator are marked.

One of the key enhancements presently planned for the interpretive simulator is the addition of a complete set of floating-point instructions and decimal arithmetic instructions. It is of course the ultimate aim of the interpretive simulator to be capable of processing the complete instruction set of the SUMC computer which at a particular time is serving as the simulation target computer.

4. Interrupt Servicing Simulation. As discussed previously in this report, the SUMC simulator has the present capability to model the target computer interrupt detection and stacking operations. The simulation of actual interruption servicing operations has not been a part

Table III-7. Instruction Implementation Status

Code	Mnemonic	Status	Code	Mnemonic	Status	Code	Mnemonic	Status
04	SPM	X	40	STH	X	80	SSM	X
05	BALR	X	41	LA	X	82	LPSW	X
06	BCTR	X	42	STC	X	83		
07	BCR	X	43	IC	X	84	WRD	
08	SSK		44	EX		85	RDD	
09	ISK		45	BAL	X	86	BXH	X
0A	SVC	X	46	BCT	X	87	BXLE	X
10	LPR	X	47	BC	X	88	SRL	X
11	LNR	X	48	LH	X	89	SLL	X
12	LTR	X	49	CH	X	8A	SRA	X
13	LCR	X	4A	AH	X	8B	SLA	X
14	NR	X	4B	SH	X	8C	SRDL	X
15	CLR	X	4C	MH	X	8D	SLDL	X
16	OR	X	4E	CVD		8E	SRDA	X
17	XR	X	4F	CVB		8F	SLDA	X
18	LR	X	50	ST	X	90	STM	X
19	CR	X	54	N	X	91	TM	X
1A	AR	X	55	CL	X	92	MVI	X
1B	SR	X	56	O	X	93	TS	X
1C	MR	X	57	X	X	94	NI	X
1D	DR	X	58	L	X	95	CLI	X
1E	ALR	X	59	C	X	96	OI	X
1F	SLR	X	5A	A	X	97	XI	X
20	LPDR		5B	S	X	98	LM	X
21	LNDR		5C	M	X	9C	SIO	
22	LTDR		5D	D	X	9D	TIO	
23	LCDR		5E	AL	X	9E	HIO	
24	HDR		5F	SL	X	9F	TCH	
28	LDR		60	STD		D1	MVN	
29	CDR		68	LD		D2	MVC	X
2A	ADR		69	CD		D3	MVZ	
2B	SDR		6A	AD		D4	NC	
2C	MDR		6B	SD		D5	CLC	
2D	DDR		6C	MD		D6	OC	
2E	AWR		6D	DD		D7	XC	
2F	SWR		6E	AW		DC	TR	
30	LPER		6F	SW		DD	TRT	
31	LNER		70	STE		DE	ED	
32	LTER		78	LE		DF	EDMK	
33	LCER		79	CE		F1	MVO	
34	HER		7A	AE		F2	PACK	
38	LER		7B	SE		F3	UNPK	
39	CER		7C	ME		F8	ZAP	
3A	AER		7D	DE		F9	CP	
3B	SER		7E	AU		FA	AP	
3C	MER		7F	SU		FB	SP	
3D	DER					FC	MP	
3E	AUR					FD	DP	
3F	SUR							

X = Have been implemented for SUMC simulator.

of the present project. For simulator completeness, however, the simulation of the total interrupt response operations performed by the target computer is an essential part of future simulator enhancement studies.

The present SUMC target computer recognizes 16 different interruption conditions and future enhancement plans therefore include the implementation of at least this set of interrupt service routines. These interruptions are listed below.

- Program Interruptions
  - operation exception
  - privileged-operation exception
  - execute exception
  - addressing exception
  - specification exception
  - data exception
  - fixed-point overflow exception
  - fixed-point divide exception
  - exponent overflow exception
  - exponent underflow exception
  - significance exception
  - floating-point divide exception
- Supervisor-Call (SVC) Interruption
- Input/Output Interruptions
  - single I/O channel
- External Interruptions
  - interrupt key signals
- Machine-Check Interruption

SECTION IV. SUMMARY

The SUMC simulator has been designed to interpretively execute the instruction set of a SUMC target computer. Flexibility has been designed into the simulator so that a SUMC family of target computers may be simulated. This done by introducing simulation parameters which define the key architectural features of the target computer under consideration. The simulator is given added relevance to many users through a design objective requiring host machine independence for the simulator to the fullest possible extent. This goal is accomplished by isolating all host machine dependent functions performed by the simulator to a minimum number of distinct program modules.

After a brief description of the SUMC architecture and instruction set, a complete description of the SUMC interpretive simulator is given in Section III. In this section, following a discussion of simulator design principles, the different functional program modules making up the simulator are discussed separately. The simulator modules have been grouped under the following headings:

- primary control loop
- initialization
- instruction parse and execute
- diagnostics
- program termination
- program interruptions
- utility routines and functions

To supplement the simulator description given in Section III, the appendices found at the end of this report include:

- User's Manual - to provide information for efficient use of the simulator covering deck setup, required data inputs, and data card formats;
- Module Descriptions - to provide brief descriptions of all functional modules which comprise the complete simulator;

- SUMC Instructions - to provide brief descriptions of the Breadboard System instructions which have been implemented in the interpretive simulator.
- Simulator Source Program Listing - to provide a complete record of the SUMC simulator as it presently exists.
- Sample Output Listing - to provide an example of the type of simulation output obtained when simulating a typical SUMC target program.

Recommended expansion and enhancements for the simulator are also pointed out in Section III. The following categories are covered:

- Input/Output
- Execution efficiency
- Target instruction set
- Interrupt servicing

V. BIBLIOGRAPHY

1. Space Ultrareliable Modular Computer (SUMC) Breadboard System Operations Guide, Tech. Rpt., NASA-MSFC, Computers Div., Astrionics Lab.

2. MSFC Advanced Aerospace Computer, E. Eastin et al, Contractor Rpt. SP-232-0384 prepared for MSFC by Sperry Rand Corp. under NASA Contract NAS8-20055, Huntsville, Ala., July 6, 1970.

3. Microelectronics Research for Shuttle and Space Station, NASA TM X-64504, Research Achievements Review, Volume III, Rpt. No. 11, George C. Marshall Space Flight Center, 1970.

4. Space Ultrareliable Modular Computer (SUMC) Central Processing Unit, Tech. Rpt., NASA-MSFC, Computers Div., Astrionics Lab.

5. Techniques in the Generation of Support Software for the SUMC Functional Requirements, Tech. Rpt., MDAC Contract No. NAS8-27202, August 19, 1971.

6. Techniques in the Generation of Support Software for the SUMC - Preliminary Design Specification, Tech. Rpt., MDAC Contract No. NAS8-27202, August 19, 1971.

7. Program Description - Generalized Aerospace Computer Simulator, Tech. Rpt. No. CS-6921-R0128, Logicon, August 1969.

8. Shuttle Computation System, E. Eastin, Contractor Rpt. SP-233-0252 prepared for MSFC by Sperry Rand Corp. under NASA Contract No. NAS8-20055, Huntsville, Ala., June 8, 1970.

9. Proposed Instruction Set for SUMC System, E. Thompson et al, Contractor Rpt. SP-232-0405-1 prepared for MSFC by Sperry Rand Corp. under NASA Contract NAS8-20055, Huntsville, Ala., September 4, 1970.

10. LICOS - An Interpretive Simulation of the Librascope Computer Model-1 and Model-3, M. McLennan, Contract Rpt. prepared under NASA Contract NAS3-3232, December 20, 1963.

11. Simulation of the SEL-810A Computer on Maniac II (SELMA), N. Metropolis et al, Report prepared by Los Alamos Scientific Laboratory of the University of California, Los Alamos, New Mexico, November 1966.

12. A Burroughs 220 Emulator for the IBM 360/25, T. A. Schoen and M. R. Belsole, Jr., IEEE Transactions on Computers, July 1971.

13. Microprogramming: Principles and Practices, S. S. Husson, Prentice-Hall, Englewood Cliffs, N.J., 1970.

14. IBM Systems Reference Library, IBM 7090/7094 Support Package for IBM System/360, C28-6501-2, IBM Programming & Publications, New York, N.Y., 1964.

15. FORTRAN IV Computing and Applications, R. L. Nolan, Addison-Wesley Publishing Co., Inc., 1971.

16. Programming the IBM 7090, J. A. Saxon, Prentice-Hall, Inc., Englewood Cliffs, N.J., 1963.

17. Computer Software, I. Flores, Prentice-Hall, Inc., Englewood Cliffs, N.J., 1965.

18. Investigation and Simulation of a Self-Repairing Digital Computer, I. Terris, University Microfilms, Inc., Ann Arbor, Mich., 1965.

19. IBM Systems Journal, Volume Eight, Number Four, 1969, pp. 251-350.

20. The Use of Simulation in the Design of a Real-Time, Multiprocessing Computer System, C. E. Price, Fourth Annual Simulation Symposium Record of Proceedings, Gordon and Breach Science Publishers, New York, N.Y., 1971.

21. The PMS and ISP Descriptive Systems for Computer Structures, C. G. Bell and A. Newell, Proc. AFIPS Spring Joint Computer Conf., 1970, pp. 351-374.

22. Evaluation of Aerospace Computer Architecture, M. D. Anderson and V. J. Marek, Contractor Rpt. prepared under NASA Contract NAS12-589.

23. Modular Design of Computers Through the Use of Multi-Level Simulation, K. A. Duke et al, Rpt. RC2872 (#13504), IBM Thomas J. Watson Research Center, Yorktown Heights, N.Y., May 8, 1970.

24. Evaluation of Hardware-Firmware-Software Trade-offs with Mathematical Modeling, H. Barsamian and A. De Cegama, Proc. AFIPS Spring Joint Computer Conf., 1971, pp. 151-162.

25. Extended FORTRAN IV Reference Manual for XDS Sigma 5/7 Computers, Xerox Data Systems, El Segundo, Calif., April 1970.

26. IBM System/360 and System/370 FORTRAN IV Language, GC28-6515-8, IBM Corp., Programming Publications, New York, N.Y., January 1971.

27. Univac 1108 Multi-Processor System FORTRAN V, UP-4060, Rev. 1, Sperry Rand Corp., 1969.

28. IBM 7090/7094 IBSYS Operating System: Version 13 FORTRAN IV Language, GC28-6390-4, IBM Corp., Programming Publications, New York, N.Y., November 1968.

APPENDIX I. USER MANUAL

The intention of this section is to provide a brief but exact summary of the operational characteristics of the SUMC interpretive simulator program. Proper deck setup is illustrated for execution on the IBM 7094 host computer and detailed descriptions of all required or optional data cards is also given.

A. Deck Setup

Figure AI-1 shows the basic components which make up the complete simulator source deck. The simulator is presently a self-contained set of program modules designed to operate in the batch programming mode. The following component decks are therefore required for execution of a SUMC program using the interpretive simulator.

1. Host Control Cards. This is a standard deck of control cards used by the host computer for processing a particular simulation job. The makeup of this set of cards is entirely dependent on the particular computer installation which is used as the host system.

2. Simulator Modules. This set of program modules comprises the basic SUMC interpretive simulator. The simulator has been modularized in this fashion for ease of implementation, convenience of program changes, and also to isolate host computer programming dependencies. A brief description of each program module included in the basic simulator is given in Appendix II.

3. $DATA Card. This card is required under the IBM 7094 host system in order to signal the presence of user-supplied input data cards.

4. Diagnostics Data. This set of data cards provides the diagnostics keys and accompanying information needed to activate and execute any SNAP, TRACE, SPM DUMP, or MM DUMP diagnostics wanted by the user. A detailed explanation of the contents of this data deck is given later in this section.

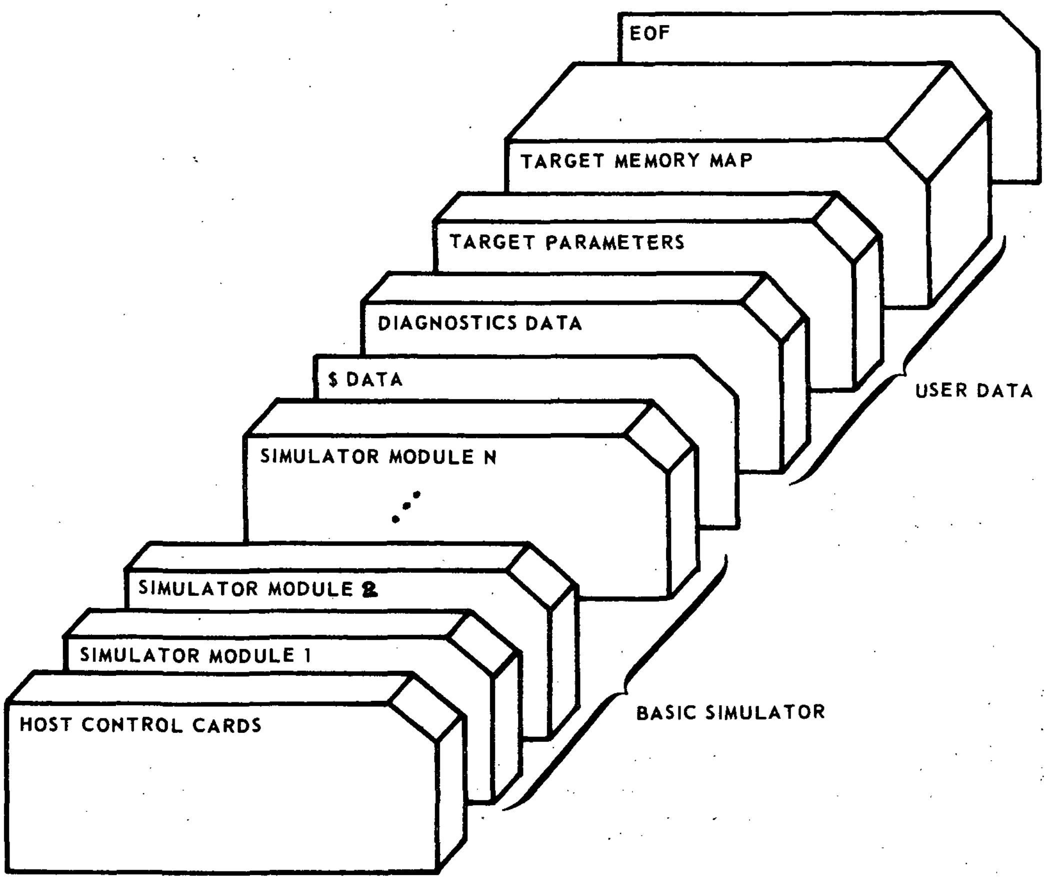


FIGURE A1-1. SUMC INTERPRETIVE SIMULATOR SOURCE DECK SETUP.

5. Target Parameters. The architectural features of the SUMC target computer which may be varied for a particular application are supplied with values appearing in this data deck. A detailed explanation of the data cards needed to assign values to the different parameters is given later in this section.

6. Target Memory Map. The target program which is to be interpretively executed by the SUMC simulator is described through data contained in the target memory map deck. Each card included in this deck contains a SUMC main memory address and the hexadecimal value which is to be loaded into the location. A detailed layout of this card deck will be given later in this section.

B. Diagnostics Data Deck Setup

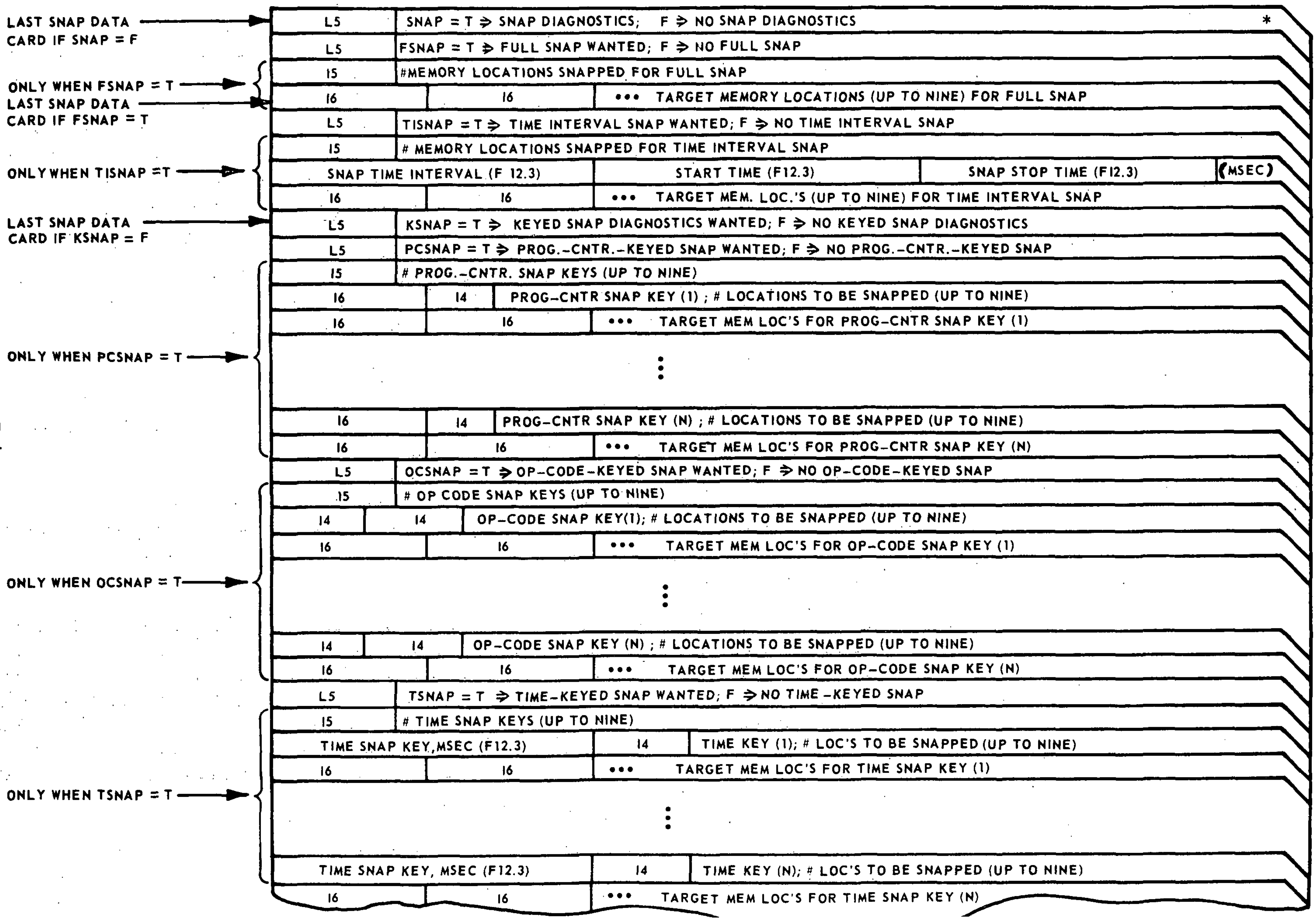
The following five pages present a detailed layout of the data cards which may be included in the user's diagnostics deck. The exact sequence required for input of this data as well as the necessary formats are given. The diagnostics data is divided into four classifications:

- SNAP diagnostics data (Figure A1-2)
- TRACE diagnostics data (Figure A1-3)
- SPM DUMP diagnostics data (Figure A1-4)
- MM DUMP diagnostic data (Figure A1-4)

The five data cards which must always be included as part of this data deck are marked with an asterisk.

C. Target Computer Definition Data

The target computer architectural parameters which may be specified by the user are given specific values in this set of data cards. The present version of the SUMC simulator allows user specification of eleven simulation variables and the eight data cards which input these variables are described in Figure A1-5. All eight data cards must be present for proper program execution.



LAST SNAP DATA CARD IF SNAP = F
ONLY WHEN FSNAP = T
LAST SNAP DATA CARD IF FSNAP = T
ONLY WHEN TISNAP = T
LAST SNAP DATA CARD IF KSNAP = F
ONLY WHEN PCSNAP = T
ONLY WHEN OCSNAP = T
ONLY WHEN TSNAP = T
L5 | SNAP = T ⇛ SNAP DIAGNOSTICS; F ⇛ NO SNAP DIAGNOSTICS | *
L5 | FSNAP = T ⇛ FULL SNAP WANTED; F ⇛ NO FULL SNAP
I5 | #MEMORY LOCATIONS SNAPPED FOR FULL SNAP
I6 | I6 | ••• TARGET MEMORY LOCATIONS (UP TO NINE) FOR FULL SNAP
L5 | TISNAP = T ⇛ TIME INTERVAL SNAP WANTED; F ⇛ NO TIME INTERVAL SNAP
I5 | # MEMORY LOCATIONS SNAPPED FOR TIME INTERVAL SNAP
SNAP TIME INTERVAL (F 12.3) | START TIME (F12.3) | SNAP STOP TIME (F12.3) | (MSEC)
I6 | I6 | ••• TARGET MEM. LOC.'S (UP TO NINE) FOR TIME INTERVAL SNAP
L5 | KSNAP = T ⇛ KEYED SNAP DIAGNOSTICS WANTED; F ⇛ NO KEYED SNAP DIAGNOSTICS
L5 | PCSNAP = T ⇛ PROG.-CNTR.-KEYED SNAP WANTED; F ⇛ NO PROG.-CNTR.-KEYED SNAP
I5 | # PROG.-CNTR. SNAP KEYS (UP TO NINE)
I6 | I4 | PROG-CNTR SNAP KEY (1) ; # LOCATIONS TO BE SNAPPED (UP TO NINE)
I6 | I6 | ••• TARGET MEM LOC'S FOR PROG-CNTR SNAP KEY (1)
I6 | I4 | PROG-CNTR SNAP KEY (N) ; # LOCATIONS TO BE SNAPPED (UP TO NINE)
I6 | I6 | ••• TARGET MEM LOC'S FOR PROG-CNTR SNAP KEY (N)
L5 | OCSNAP = T ⇛ OP-CODE-KEYED SNAP WANTED; F ⇛ NO OP-CODE-KEYED SNAP
I5 | # OP CODE SNAP KEYS (UP TO NINE)
I4 | I4 | OP-CODE SNAP KEY(1); # LOCATIONS TO BE SNAPPED (UP TO NINE)
I6 | I6 | ••• TARGET MEM LOC'S FOR OP-CODE SNAP KEY (1)
I4 | I4 | OP-CODE SNAP KEY (N) ; # LOCATIONS TO BE SNAPPED (UP TO NINE)
I6 | I6 | ••• TARGET MEM LOC'S FOR OP-CODE SNAP KEY (N)
L5 | TSNAP = T ⇛ TIME-KEYED SNAP WANTED; F ⇛ NO TIME-KEYED SNAP
I5 | # TIME SNAP KEYS (UP TO NINE)
TIME SNAP KEY, MSEC (F12.3) | I4 | TIME KEY (1); # LOC'S TO BE SNAPPED (UP TO NINE)
I6 | I6 | ••• TARGET MEM LOC'S FOR TIME SNAP KEY (1)
TIME SNAP KEY, MSEC (F12.3) | I4 | TIME KEY (N); # LOC'S TO BE SNAPPED (UP TO NINE)
I6 | I6 | ••• TARGET MEM LOC'S FOR TIME SNAP KEY (N)

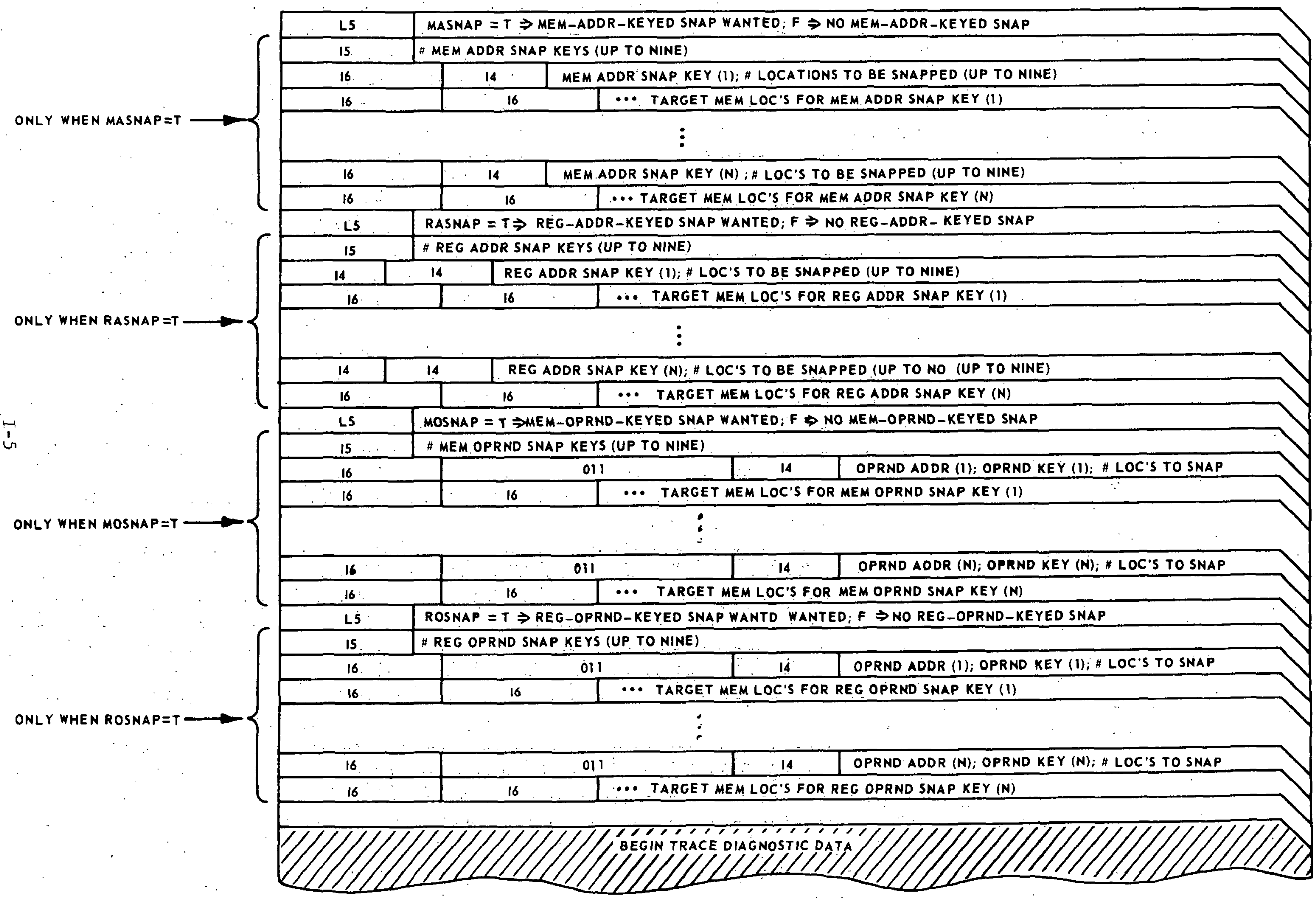
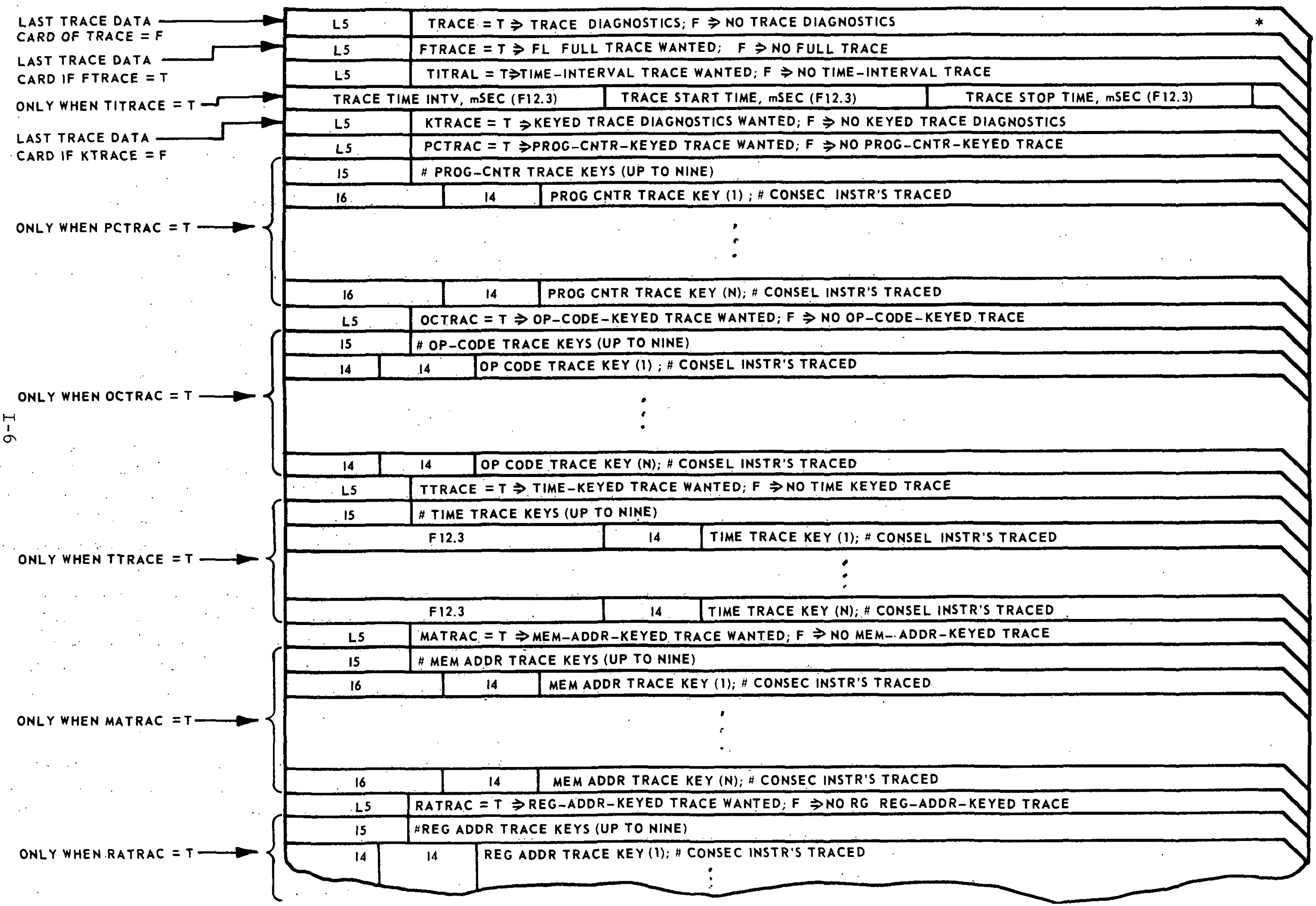


FIGURE A1-2. (CON'T)


LAST TRACE DATA CARD OF TRACE = F
LAST TRACE DATA CARD IF FTRACE = T
ONLY WHEN TITRACE = T
LAST TRACE DATA CARD IF KTRACE = F
ONLY WHEN PCTRAC = T
ONLY WHEN OCTRAC = T
ONLY WHEN TTRACE = T
ONLY WHEN MATRAC = T
ONLY WHEN RATRAC = T
L5 | TRACE = T ⇛ TRACE DIAGNOSTICS; F ⇛ NO TRACE DIAGNOSTICS *
L5 | FTRACE = T ⇛ FL FULL TRACE WANTED; F ⇛ NO FULL TRACE
L5 | TITRAL = T⇛TIME-INTERVAL TRACE WANTED; F ⇛ NO TIME-INTERVAL TRACE
TRACE TIME INTV, mSEC (F12.3) | TRACE START TIME, mSEC (F12.3) | TRACE STOP TIME, mSEC (F12.3)
L5 | KTRACE = T ⇛KEYED TRACE DIAGNOSTICS WANTED; F ⇛ NO KEYED TRACE DIAGNOSTICS
L5 | PCTRAC = T ⇛PROG-CNTR-KEYED TRACE WANTED; F ⇛NO PROG-CNTR-KEYED TRACE
I5 | # PROG-CNTR TRACE KEYS (UP TO NINE)
I6 | I4 | PROG CNTR TRACE KEY (1) ; # CONSEC INSTR'S TRACED
I6 | I4 | PROG CNTR TRACE KEY (N); # CONSEL INSTR'S TRACED
L5 | OCTRAC = T ⇛ OP-CODE-KEYED TRACE WANTED; F ⇛ NO OP-CODE-KEYED TRACE
I5 | # OP-CODE TRACE KEYS (UP TO NINE)
I4 | I4 | OP CODE TRACE KEY (1) ; # CONSEL INSTR'S TRACED
I4 | I4 | OP CODE TRACE KEY (N); # CONSEL INSTR'S TRACED
L5 | TTRACE = T ⇛ TIME-KEYED TRACE WANTED; F ⇛NO TIME KEYED TRACE
I5 | # TIME TRACE KEYS (UP TO NINE)
F12.3 | I4 | TIME TRACE KEY (1); # CONSEL INSTR'S TRACED
F12.3 | I4 | TIME TRACE KEY (N); # CONSEL INSTR'S TRACED
L5 | MATRAC = T ⇛MEM-ADDR-KEYED TRACE WANTED; F ⇛NO MEM--ADDR-KEYED TRACE
I5 | # MEM ADDR TRACE KEYS (UP TO NINE)
I6 | I4 | MEM ADDR TRACE KEY (1); # CONSEC INSTR'S TRACED
I6 | I4 | MEM ADDR TRACE KEY (N); # CONSEC INSTR'S TRACED
L5 | RATRAC = T ⇛REG-ADDR-KEYED TRACE WANTED; F ⇛NO RG REG-ADDR-KEYED TRACE
I5 | #REG ADDR TRACE KEYS (UP TO NINE)
I4 | I4 | REG ADDR TRACE KEY (1); # CONSEC INSTR'S TRACED

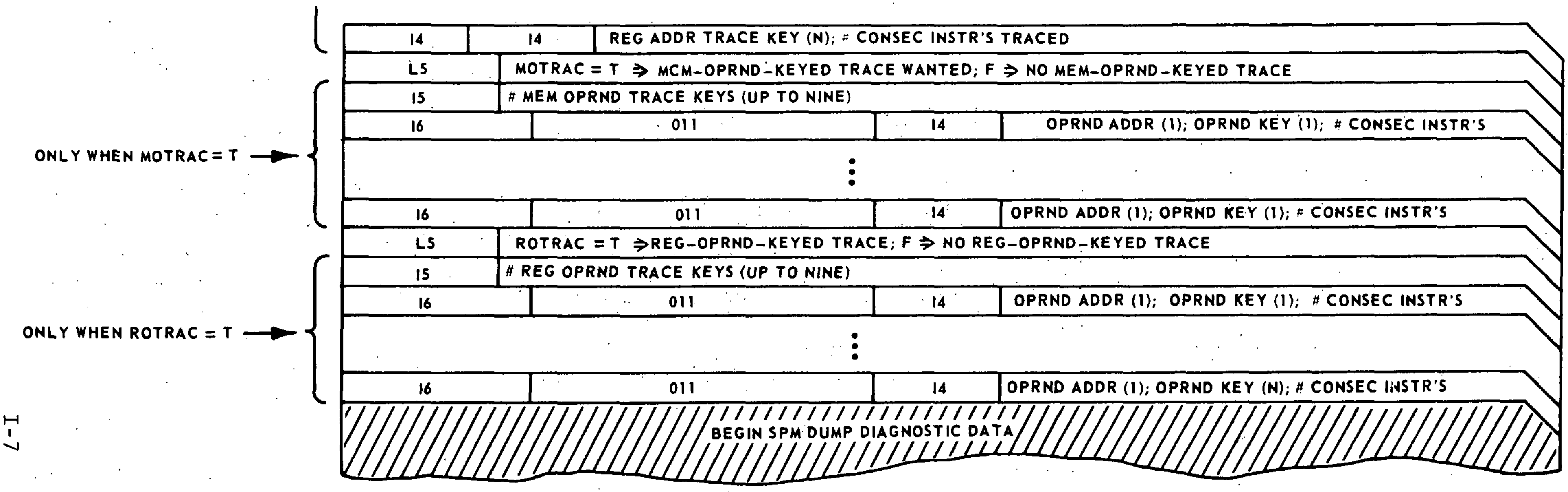


FIGURE A1-3. (CON'T)

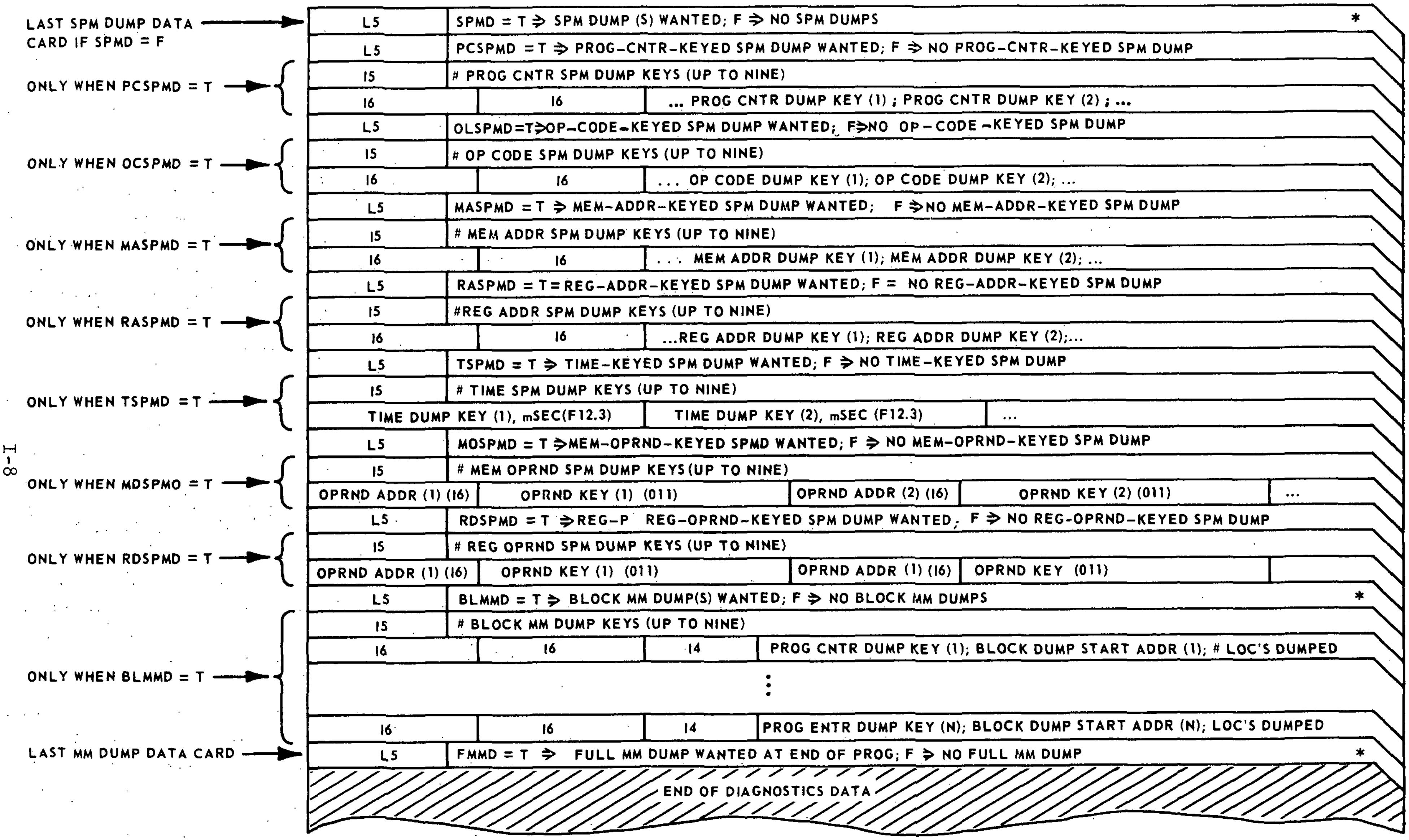


FIGURE A1-4. LAYOUT OF SPM DUMP AND MM DUMP DATA DECKS.

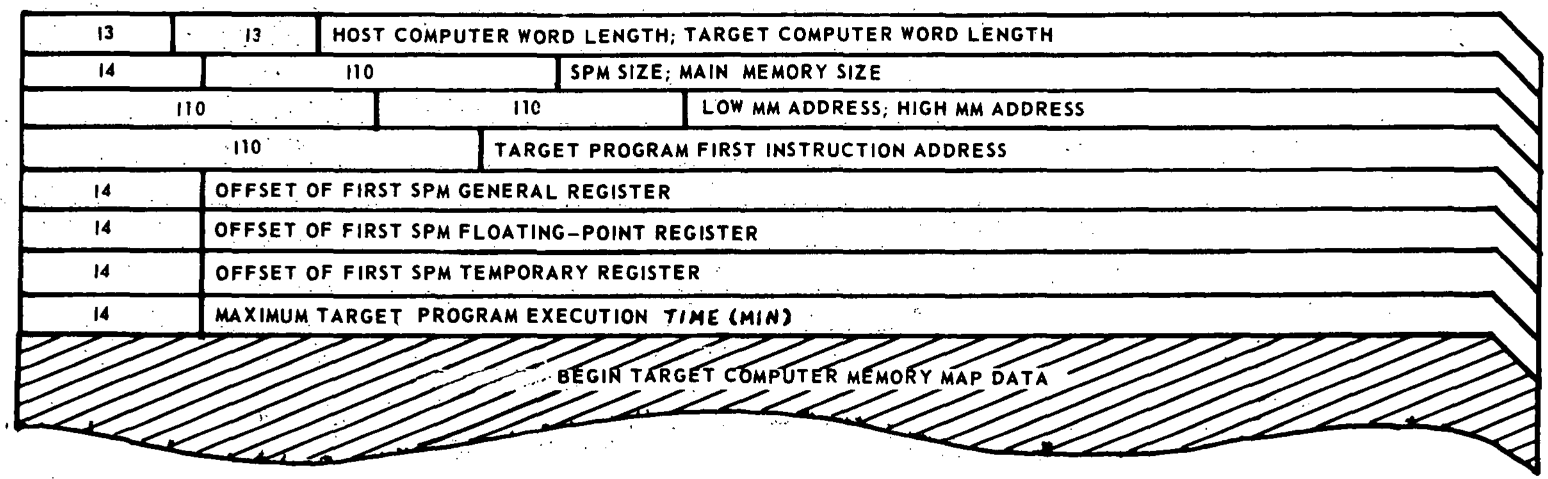


FIGURE A1-5. TARGET COMPUTER DEFINITION DATA

D. Target Computer Memory Map

The target program is defined for the simulator by specifying the memory map for the SUMC target computer. This is presently done by means of the memory map card deck which supplies memory data to the simulator on a single location per card basis. Each card contained in the memory map deck includes:

- SUMC MM address in hexadecimal
- contents of the specified MM address in hexadecimal
- number of halfwords being specified (1 or 2)

Figure A1-6 illustrates the card layouts to be used for the memory map deck. Note that the first data card must specify an offset which can be used to relocate the target memory map with respect to location zero of simulated SUMC main memory. If the offset is zero, the MM address specified in the data represent absolute addresses.

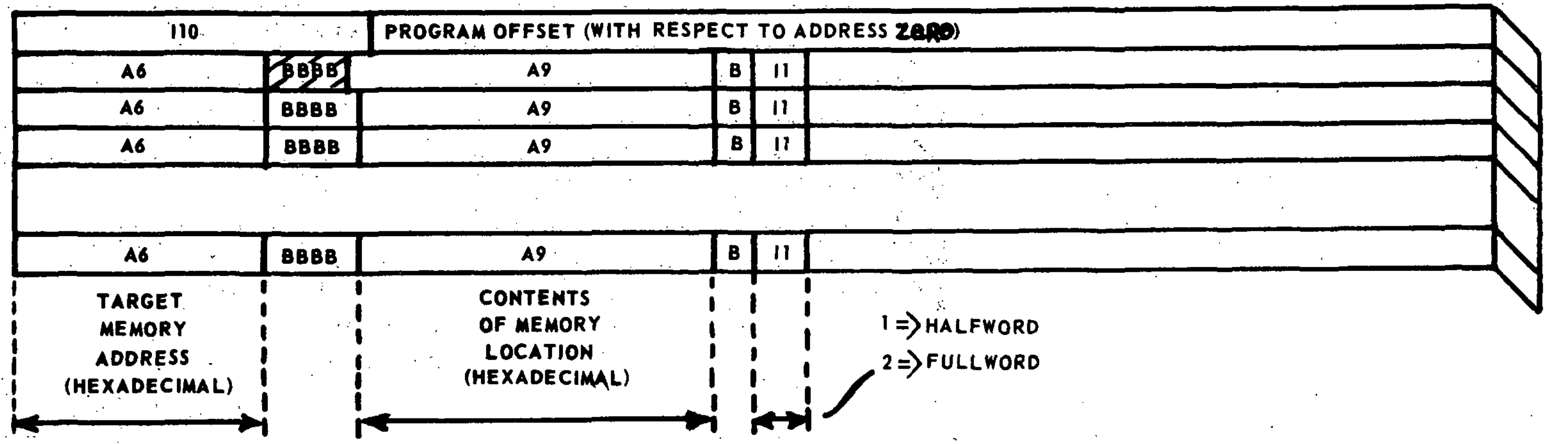


FIGURE A1-6. TARGET COMPUTER MEMORY MAP DATA

APPENDIX II. MODULE DESCRIPTIONS

This appendix provides a brief description of all program modules making up the complete SUMC interpretive simulator program. The material is organized so that each program module is described on a single page, and the information provided with each module description includes:

- procedure or module identifier
- purpose of the module
- programming approach
- external procedures referenced by the module
- external data referenced by the module

FUNCTIONAL PROCEDURE DESCRIPTION

Procedure Identifier: COR 001

Purpose: Mainline program for interpretive simulator.

Approach: Program is a series of subroutine calls and error checks. Snap, Dump, Trace requests are checked and executed after initialization is completed. Interrupt detection logic follows. After interrupts are serviced the computer instruction is interpretively executed. This procedure is repeated until an end of program test is successful.

External Procedure Referenced: Subroutines: COR 008 (INITLZ), TRACSR (TRACEX), SNAPSR (SNAPEX), SPMDSR (SPMDEX), BMMDSR (BLMDEX), COR 003 (FECHM), COR 005 (OPDEF), COR 014 (TIMER), COR 015 (INTRPS), COR 013 (TERMIN)

External Data Referenced: Block data subprogram, input parameters.

FUNCTIONAL PROCEDURE DESCRIPTION

Procedure Identifier: COR 002

Purpose: Block data subprogram provides initialization of parameters.

Approach:

External Procedure Referenced:

External Data Referenced:

FUNCTIONAL PROCEDURE DESCRIPTION

Procedure Identifier: COR 003 (FECHM)

Purpose: Extract current Op Code from instruction. Select Op Code dependent parameters for parse, execution time, and statistics dump. Parse instruction, perform error checking on results of parse.

Approach: Using the Op Code extracted from the instruction as a pointer, tabular values of the number of halfwords, segments/halfword, and bits/segment are accessed. These data are utilized to extract each instruction segment and store it in the segment table for use by other subprograms.

External Procedure Referenced: BRCHK, FLD, ERINS, HALTE

External Data Referenced: COMMON

FUNCTIONAL PROCEDURE DESCRIPTION

Procedure Identifier: COR 004 (STATDP)

Purpose: Publish run data summary.

Approach: This subroutine accumulates and updates simulation statistics throughout program execution and prints final results at program termination. Statistics are kept concerning number of instructions processed, type of instructions processed, and times for each.

External Procedure Referenced:

External Data Referenced:

FUNCTIONAL PROCEDURE DESCRIPTION

Procedure Identifier: COR 005 (OPDEF)

Purpose: To take Op Code and parse data from COR 003 and interpretively execute the Op Code by means of a sequence of simulation instructions.

Approach: In the execution sequence, error checks are routinely done simulating the error checking facilities of the SUMC. A computed GO TO sends program control to the proper instruction simulation routine according to the value of the current instruction Op Code.

External Procedure Referenced: BRCHK, INTRPS, IOR, HALTE, FECHFW, IPPSW, FLD, JEKCC, INTNOT, IER, STORCH, ERINS, IAND, OVERFL

External Data Referenced: COMMON

FUNCTIONAL PROCEDURE DESCRIPTION

Procedure Identifier: COR 006 (BRCHK)

Purpose: To perform a validity check upon the main storage address (MSA) furnished in COMMON variable IDATAS.

Approach: CALL BRCHK(J) where J is the integer variable set to 1 if the MSA is valid and 2 if the MSA is invalid. The main storage address must be stored in IDATAS prior to the subroutine call.

External Procedure Referenced: INTRPS

External Data Referenced: COMMON

FUNCTIONAL PROCEDURE DESCRIPTION

Procedure Identifier: COR 007 (STORFW)

Purpose: To store in main storage the full word, halfword, or character in the address provided in the call sequence.

Approach: CALL STORFW (Address, & default SNO)
CALL STORHW (Address, & default SNO)
CALL STORCH (Address, & default SNO)

The full word, halfword or character must be previously entered in IDATAS (right justified).

External Procedure Referenced: FLD, IAND

External Data Referenced: COMMON

FUNCTIONAL PROCEDURE DESCRIPTION

Procedure Identifier: COR 008 (INITLZ)

Purpose: Perform the required parameter initialization for the simulator. Input the required data.

Approach: For simulation data which must be supplied by the user, FORTRAN READ statements are employed. BLOCK DATA statements provide values for internal simulation variables. Certain parameters are defined through EQUIVALENCE statements.

External Procedure Referenced:

External Data Referenced:

FUNCTIONAL PROCEDURE DESCRIPTION

Procedure Identifier: COR 009 (ERINS)

Purpose: Output error messages for hardware failures. Cause interruption to occur in a manner similar to SUMC hardware exceptions.

Approach: IERFLG must be set to the code that indicates the error mode.

External Procedure Referenced: INTRPS

External Data Referenced:

FUNCTIONAL PROCEDURE DESCRIPTION

Procedure Identifier: COR 010 (HALTE)

Purpose: To output the HALT message indicating program termination due to failure.

Approach:

External Procedure Referenced:

External Data Referenced:

FUNCTIONAL PROCEDURE DESCRIPTION

Procedure Identifier: COR 011 (FECHFW)

Purpose: To fetch a datum from main storage.

Approach: CALL FECHFW (Address, & default SNO) full word
CALL FECHHW (Address, & default SNO) halfword
CALL FECHAR (Address, & default SNO) character

Datum is returned in IDATAS (right justified). Full word, halfword, character. Validity check is done on the MSA.

External Procedure Referenced: BRCHK, FLD, IAND

External Data Referenced: COMMON

FUNCTIONAL PROCEDURE DESCRIPTION

Procedure Identifier: COR 012 (JEKCC)

Purpose: To test the status of the condition codes.

Approach: FUNCTION JEKCC (MASK = ICODE). If any bit in the mask matches the condition code, JEKCC = 2. If no bits match, JEKCC = 1.

External Procedure Referenced: IAND

External Data Referenced:

FUNCTIONAL PROCEDURE DESCRIPTION

Procedure Identifier: COR 013 (TERMIN)

Purpose: To output message indicating successful conclusion of simulation and to institute restart subprogram if required.

Approach: A printout is initiated which identifies the termination cause and the STATDP routine is CALLED in order to print appropriate end-of-run statistics.

External Procedure Referenced: STATDP, RSTART

External Data Referenced: COMMON

FUNCTIONAL PROCEDURE DESCRIPTION

Procedure Identifier: COR 014 (TIMER)

Purpose: To maintain simulated elapsed execution time. If expected elapsed time is exceeded, simulation is terminated.

Approach: Elapsed time is incremented using parameters fetched using op code as a printer. Operations counter is incremented. Elapsed time is compared with Time Limit. If time limit is exceeded, simulation is terminated.

External Procedure Referenced: TERMIN

External Data Referenced: COMMON

FUNCTIONAL PROCEDURE DESCRIPTION

Procedure Identifier: COR 015 (INTRPS)

Purpose: To simulate SUMC interrupt capability.

Approach: Tables of current pending interrupts are maintained and searched upon request. An active enabled interrupt causes the appropriate service subprogram to be called. The service subprograms are not part of this contract.

CALL PUSH (priority level, interruption code, channel status, default SNO)
CALL PULL (priority level, J, interruption code, channel status word)
J = 1 No active interrupt at this level.
J = 2 Active interrupt.

External Procedure Referenced: IMMPSW, TERMIN, HALTE, ISPPSW, IAND, ERINS

External Data Referenced: COMMON

FUNCTIONAL PROCEDURE DESCRIPTION

Procedure Identifier: COR 016 (RSTART)

Purpose: To provide facility for the restart procedures that may be added at a later date.

Approach:

External Procedure Referenced:

External Data Referenced:

FUNCTIONAL PROCEDURE DESCRIPTION

<u>Procedure Identifier:</u> COR 017 IAND (I,J)
COR 018 IOR (I,J)
COR 019 IER (I,J)

<u>Purpose:</u> To provide the logical operations of And, Or, and Exclusive OR on computer words.

<u>Approach:</u> The functions are implemented in a completely host-machine-independent manner.

<u>External Procedure Referenced:</u> AND, OR

<u>External Data Referenced:</u> NONE

FUNCTIONAL PROCEDURE DESCRIPTION

<u>Procedure Identifier</u>: COR 020 (FUNCTION IMMPSW(K))

<u>Purpose</u>: Converts current PSW's to old PSW formats and stores in appropriate main storage locations.

<u>Approach</u>: K is the address in main storage of the 1st PSW of the group. The main storage address is checked for validity.

<u>External Procedure Referenced</u>: BRCHK, FLD, INTRPS

<u>External Data Referenced</u>: COMMON

FUNCTIONAL PROCEDURE DESCRIPTION

Procedure Identifier: COR 021 (FUNCTION ISPPSW(K))

Purpose: To convert new PSW's in main storage into current PSW format and store into simulated scratch pad memory.

Approach: K is address in main storage of 1st PSW required. Main Storage address is checked for validity.

External Procedure Referenced: INTRPS, BRCHK, FLD

External Data Referenced:

FUNCTIONAL PROCEDURE DESCRIPTION

Procedure Identifier: COR 022 (FUNCTION INTNOT(K))

Purpose: To provide 1's complement of variable K

Approach: The implementation stresses host-machine-independence.

External Procedure Referenced: COMPL

External Data Referenced:

FUNCTIONAL PROCEDURE DESCRIPTION

Procedure Identifier: COR 023 (INTSER)

Purpose: To provide facility for the addition of interrupt service routines at a later date.

Approach:

External Procedure Referenced:

External Data Referenced:

FUNCTIONAL PROCEDURE DESCRIPTION

Procedure Identifier: COR 024 (ITWTSM(I))
COR 025 (ISMTWO(I))

Purpose: To provide function subprogram which perform conversion operations form 2's-complement representation to sign-magnitude (ITWTSM) and sign-magnitude to 2's-complement (ISMTWO).

Approach: The routines are intended for use in simulating a 2's-complement arithmetic target computer on a host system which employs sign-magnitude arithmetic.

External Procedure Referenced: IAND, IOR, INTNOT, ILOAD

External Data Referenced:

FUNCTIONAL PROCEDURE DESCRIPTION

Procedure Identifier: COR 026 (ISTORE (ISOR,IDEST,IS,IN))

Purpose: To provide a host-independent routine for placing a right-justified field of data from the source word in the specified field of the destination word.

Approach: ISOR and IDEST specify the data source word and destination word, respectively. The field of data is stored in the destination word starting at bit position IS and is IN bits in length. Bit numbering is right to left with the rightmost bit designated as bit number one.

External Procedure Referenced: FLD

External Data Referenced:

FUNCTIONAL PROCEDURE DESCRIPTION

Procedure Identifier: COR 027 (JEXBIN (IBUF, IST, ILNG))

Purpose: To convert character strings in hexadecimal format to an internal IBM 7094 binary format.

Approach: IBUF is the character string location, IST is a pointer to the first character in the string IBUF, and ILNG specifies the number of hexadecimal characters to be converted. A maximum of 80 characters may be converted by the function.

External Procedures Referenced: ILOAD

External Data Referenced:

FUNCTIONAL PROCEDURE DESCRIPTION

Procedure Identifier: COR 028 (ILOAD (SOURCE, SB, NB))

Purpose: To provide a host-machine-independent routine for loading a field of data as a right-justified field in the output argument.

Approach: The data source word is specified by SOURCE. The field of data to be loaded as the right-justified output argument begins at bit position SB of the source word and is NB bits in length. Bit numbering is right to left with the rightmost bit designated as bit number one.

External Procedure Referenced: FLD

External Data Referenced:

FUNCTIONAL PROCEDURE DESCRIPTION

Procedure Identifier: COR 029 (ISHADR(K))

Purpose: To extract the shift count from the current target instruction.

Approach: ISHADR is an integer function subprogram which is useful in executing several different target computer instructions.

External Procedure Referenced: ILOAD

External Data Referenced:

FUNCTIONAL PROCEDURE DESCRIPTION

Procedure Identifier: COR 030 (IVERFL(L,LOF))

Purpose: To compute overflow for conventional arithmetic operations based on signs of inputs and results.

Approach: Rules for generation of overflow are applied.

External Procedure Referenced: IAND

External Data Referenced: L is result of arithmetic operation; LOF is overflow parameter returned. LOF = 1 $\Rightarrow$ O/F, LOF = 2 $\Rightarrow$ No O/F.

FUNCTIONAL PROCEDURE DESCRIPTION

Procedure Identifier: COR 031 (ISLOGE(N))

Purpose: To collect COMMON logic associated with SS logical instructions.

Approach: N/A

External Procedure Referenced: FECHAR, IAND, IER, IOR, STORCH

External Data Referenced: COMMON, N is call parameter defining logical operation,

N = 1, logical AND
N = 2, logical OR
N = 3, exclusive OR

FUNCTIONAL PROCEDURE DESCRIPTION

Procedure Identifier: COR 032 (IOVRFL(N,J))

Purpose: To compute overflow based on logical arithmetic operations.

Approach: Special rules for logical overflow are applied to inputs and results of operations.

External Procedures Referenced: IAND

External Data Referenced: COMMON, N, J
N is result of operation, J is parameter which defines O/F. J = 1 $\Rightarrow$ overflow; J = 2 $\Rightarrow$ no O/F.

FUNCTIONAL PROCEDURE DESCRIPTION

Procedure Identifier: COR 033 (ICOMP1 (WRDIN,SB,NB))

Purpose: To provide a host-machine-independent routine for complementing a specified field of bits in the source word.

Approach: WRDIN is the source word and the function subprogram complements NB bits of this word beginning at bit position SB. Bit numbering is right to left with the rightmost bit designated as bit number one.

External Procedure Referenced: ILOAD, ISTORE

External Data Referenced:

FUNCTIONAL PROCEDURE DESCRIPTION

Procedure Identifier: TYPESP

Purpose: Data subprogram for classification of all valid target computer instructions according to diagnostics keys which may be checked.

Approach: Each valid target instruction is assigned to a particular classification which groups instructions in accordance with the appropriateness of their contents in checking diagnostics keys.

External Procedure Referenced:

External Data Referenced: /DVAR/

FUNCTIONAL PROCEDURE DESCRIPTION

Procedure Identifier: HEADSR (HEADER)

Purpose: Printout of target machine status information which serves as a header for diagnostics information which has been requested. This header will be common to all types of diagnostics.

Approach: When the mainline routine has recognized a diagnostics request, the header subroutine is called just prior to the calling of the appropriate subroutine to execute the diagnostic printout. This routine provides the user with information concerning the diagnostic requested, program time, program offset and current instruction contents.

External Procedure Referenced:

External Data Referenced: /DVAR/, /UNCON/, /UARRAY/

FUNCTIONAL PROCEDURE DESCRIPTION

Procedure Identifier: SNAPRS (SNAPRD)

Purpose: Routine to read external data which are used as keys to trigger main memory SNAP diagnostic at desired time during simulation. Each SNAP key triggers a printout of a unique set of main memory locations.

Approach: Each SNAP key specifies (1) type of key, (2) value of key, and (3) memory locations to be snapped.

External Procedure Referenced:

External Data Referenced: /FS/, /KS/

FUNCTIONAL PROCEDURE DESCRIPTION

Procedure Identifier: TRACRS (TRACRD)

Purpose: Routine to read external data which are used as keys to trigger the register TRACE diagnostic at desired time during simulation. Each trace key triggers a printout of the same key registers from SPM.

Approach: Each TRACE key specifies (1) type of key, and (2) value of key.

External Procedure Referenced:

External Data Referenced: /TR/, /KT/

FUNCTIONAL PROCEDURE DESCRIPTION

Procedure Identifier: SPMDRS (SPMDRD)

Purpose: Routine to read external data which are used as keys to trigger a dump of the SPM contents at the desired time during the simulation. Each TRACE key triggers a printout of the contents of all registers in simulated SPM.

Approach: The SPM dump key specifies (1) type of key, and (2) value of key.

External Procedure Referenced:

External Data Referenced: /SP/, /MM/

FUNCTIONAL PROCEDURE DESCRIPTION

Procedure Identifier: MMDRSR (MMDRD)

Purpose: Routine to read external data which are used as keys to trigger a block main memory dump diagnostic at the desired time during the simulation. Each main memory block dump key triggers a printout of a unique block of contiguous main memory locations.

Approach: Each main memory block dump key specifies the value of the simulated program counter which is to be used to trigger the block dump.

External Procedure Referenced:

External Data Referenced:

FUNCTIONAL PROCEDURE DESCRIPTION

Procedure Identifier: SNAPSR (SNAPEX)

Purpose: Routine which checks SNAP keys prior to execution of each instruction and collects the desired SNAP diagnostics data if triggered.

Approach: The present version of the subroutine issues a printout of SNAP data when triggered. Subsequent versions will have provisions for collecting and storing SNAP data for printout at some later time.

External Procedure Referenced: HEADSR (HEADER)

External Data Referenced: /FS/, /KS/, /DVAR/, /UNCON/, /UARRAY/

FUNCTIONAL PROCEDURE DESCRIPTION

Procedure Identifier: TRACSR (TRACEX)

Purpose: Routine which checks TRACE keys prior to execution of each instruction and collects the desired TRACE diagnostics data if triggered.

Approach: The present version of this subroutine issues a printout of the contents of key registers when triggered. Subsequent versions will have provisions for collecting and storing TRACE data for printout at some later time.

External Procedure Referenced: HEADSR (HEADER)

External Data Referenced: /TR/, /KT/, /DVAR/, /UNCON/, /UARRAY/

FUNCTIONAL PROCEDURE DESCRIPTION

Procedure Identifier: SPMDSR (SPMDEX)

Purpose: Routine which checks SPM dump keys prior to execution of each instruction and collects the desired SPM contents if triggered.

Approach: The present version of this subroutine issues a printout of SPM contents when triggered. Subsequent versions will have provisions for storing current SPM contents for printout at some later time.

External Procedure Referenced: HEADSR (HEADER)

External Data Referenced: /SP/, /MM/, /DVAR/, /UNCON/, /UARRAY/

FUNCTIONAL PROCEDURE DESCRIPTION

Procedure Identifier: BMMDSR (BLMDEX)

Purpose: Routine which checks the block main memory dump keys prior to execution of each instruction and collects the desired target main memory location contents if triggered.

Approach: A block dump of a specified set of target main memory locations can be triggered only by specified values of the simulated program counter. The present version of the simulator issues a printout of the specified target main memory contents when triggered. Subsequent versions will store the current contents of the target main memory locations for printout at some later time.

External Procedure Referenced: HEADSR (HEADER)

External Data Referenced: /SP/, /MM/, /DVAR/, /UNCON/, UARRAY/

FUNCTIONAL PROCEDURE DESCRIPTION

Procedure Identifier: FMMDSR (FMMDEX)

Purpose: Routine which checks the main memory dump key prior to termination of the simulation run, and executes a full target main memory dump if requested.

Approach: A full target main memory dump is available only at the termination of a simulation program. The dump may be executed following an error termination or following a normal program halt.

External Procedure Referenced:

External Data Referenced: /SP/, /MM/, /DVAR/, /UNCON/, /UARRAY/

APPENDIX III. SUMC INSTRUCTIONS

This appendix gives a brief description of each of the SUMC Breadboard System instructions which are presently implemented on the SUMC interpretive simulator.

GROUP I INSTRUCTIONS

Op Code	Mnemonic	Type	Instruction
04	SPM	RR	Set Program Mask

Description: Bits 2-7 of the general register specified by the R_1 field replace the condition code and the program mask bits of the current PSW. Bits 0, 1, and 8-31 of the register specified by the R_1 field are ignored. The contents of the register specified by the R_1 field remain unchanged.

Op Code	Mnemonic	Type	Instruction
05	BALR	RR	Branch and Link

Description: The right-most 32 bits of the PSW, including the updated instruction address, are stored as link information in the general register specified by R_1. Subsequently, the instruction address is replaced by the branch address. The branch address is determined before the link information is stored.

Op Code	Mnemonic	Type	Instruction
06	BCTR	RR	Branch on count

Description: The content of the general register specified by R_1 is algebraically reduced by one. When the result is zero, normal instruction sequencing proceeds with the updated instruction address. When the result is not zero, the instruction address is replaced by the branch address. The branch address is determined prior to the counting operation.

Op Code	Mnemonic	Type	Instruction
07	BCR	RR	Branch on condition

Description: The updated instruction address is replaced by the branch address if the state of the condition code is as specified by the contents of the R_1 field; otherwise, normal instruction sequencing proceeds with the updated instruction address.

Op Code	Mnemonic	Type	Instruction
0A	SVC	RR	Supervisor call

Description: The instruction causes a supervisor-call interruption with the R_1, R_2 field of the instruction providing the interruption code.

Op Code	Mnemonic	Type	Instruction
10	LPR	RR	Load positive

Description: The absolute value of the second operand is placed in the first operand location. The operation includes complementation of negative numbers; positive numbers remain unchanged.

Op Code	Mnemonic	Type	Instruction
11	LNR	RR	Load negative

Description: The 2's-complement of the absolute value of the second operand is placed in the first operand location. The operation complements positive numbers; negative numbers and zero remain unchanged.

Op Code	Mnemonic	Type	Instruction
12	LTR	RR	Load and test

Description: The second operand is placed in the first operand location, and the sign and magnitude of the second operand determine the condition code. The second operand is unchanged.

Op Code	Mnemonic	Type	Instruction
13	LCR	RR	Load complement

Description: The 2's-complement of the second operand is placed in the first operand location.

Op Code	Mnemonic	Type	Instruction
14	NR	RR	AND

Description: The logical product (AND) of the bits of the first and second operands is placed in the first operand location. Operands are treated as unstructured logical quantities, and the connective AND is applied bit by bit.

Op Code	Mnemonic	Type	Instruction
15	CLR	RR	Compare logical

Description: The first operand is compared with the second operand, and the result is indicated in the condition code.

Op Code	Mnemonic	Type	Instruction
16	OR	RR	OR

Description: The logical sum (OR) of the bits of the first and second operands is placed in the first operand location. Operands are treated as unstructured logical quantities, and the connective inclusive OR is applied bit by bit.

Op Code	Mnemonic	Type	Instruction
17	XR	RR	Exclusive OR

Description: The modulo-two sum (exclusive OR) of the bits of the first and second operands is placed in the first operand location. Operands are treated as unstructured logical quantities, and the connective exclusive OR is applied bit by bit.

Op Code	Mnemonic	Type	Instruction
18	LR	RR	Load

Description: The second operand is placed in the first operand location. The second operand is not changed.

Op Code	Mnemonic	Type	Instruction
19	CR	RR	Compare

Description: The first operand is compared with the second operand, and the result determines the setting of the condition code. Comparison is algebraic, treating both comparands as 32-bit signed integers.

Op Code	Mnemonic	Type	Instruction
1A	AR	RR	Add

Description: The second operand is added to the first operand, and the sum is placed in the first operand location.

Op Code	Mnemonic	Type	Instruction
1B	SR	RR	Subtract

Description: The second operand is subtracted from the first operand, and the difference is placed in the first operand location.

Op Code	Mnemonic	Type	Instruction
1C	MR	RR	Multiply

Description: The product of the multiplier (second operand) and the multiplicand (first operand) replaces the multiplicand. Multiplier and multiplicand are 32-bit signed integers and the product is a 64-bit signed integer occupying the even/odd register pair specified by the R_1 field of the instruction.

Op Code	Mnemonic	Type	Instruction
1D	DR	RR	Divide

Description: The dividend (first operand) is divided by the divisor (second operand) and replaced by the remainder and the quotient. The divisor is a 32-bit signed integer. The divident is a 64-bit signed integer occupying the even/odd register pair specified by the R_1 field of the instruction. A 32-bit signed remainder and a 32-bit signed quotient replace the dividend in the even-numbered and off-numbered registers, respectively.

Op Code	Mnemonic	Type	Instruction
1E	ALR	RR	Add logical

Description: The second operand is added to the first operand, and the sum is placed in the first operand location. A carry out in the sign position is recorded in the condition code. Logical addition adds all 32 bits of both operands without further change to the resulting sign bit.

Op Code	Mnemonic	Type	Instruction
1F	SLR	RR	Subtract logical

Description: The second operand is subtracted from the first operand, and the difference is placed in the first operand location. A carry out in the sign position is recorded in the condition code. All 32 bits of both operands participate, without further change to the resulting sign bit.

Op Code	Mnemonic	Type	Instruction
40	STH	RX	Store halfword

Description: The first operand is stored at the halfword second operand location. The 16 high-order bits of the first operand are ignored.

Op Code	Mnemonic	Type	Instruction
41	LA	RX	Load address

Description: The address of the second operand is inserted in the low-order 24 bits of the general register specified by R_1. The remaining bits of the general register are made zero.

Op Code	Mnemonic	Type	Instruction
42	STC	RX	Store character

Description: Bit positions 24-31 of the register designated as the first operand are placed in the second operand address.

Op Code	Mnemonic	Type	Instruction
43	IC	RX	Insert character

Description: The 8-bit character at the second operand address is inserted into bit positions 24-31 of the register specified as the first operand location.

Op Code	Mnemonic	Type	Instruction
44	EX	RX	Execute

Description: The single instruction at the branch address is modified by the content of the general register specified by R_1, and the resulting subject instruction is executed. Bits 8-15 of the instruction designated by the branch address are OR'ed with bits 24-31 of the register specified by R_1, except when register 0 is specified, which indicates no modification takes place.

Op Code	Mnemonic	Type	Instruction
45	BAL	RX	Branch and link

Description: The right-most 32 bits of the PSW, including the updated instruction address, are stored as link information in the general register specified by R_1. Subsequently, the instruction address is replaced by the branch address. The branch address is determined before the link information is stored.

Op Code	Mnemonic	Type	Instruction
46	BCT	RX	Branch on count

Description: The content of the general register specified by R_1 is algebraically reduced by one. When the result is zero, normal instruction sequencing proceeds with the updated instruction address. When the result is not zero, the instruction address is replaced by the branch address. The branch address is determined prior to the counting operation.

Op Code	Mnemonic	Type	Instruction
47	BC	RX	Branch on condition

Description: The updated instruction address is replaced by the branch address if the state of the condition code is as specified by the contents of the R_1 field; otherwise, normal instruction sequencing proceeds with the updated instruction address.

Op Code	Mnemonic	Type	Instruction
48	LH	RX	Load Halfword

Description: The halfword second operand is placed in the first operand location. The second operand sign bit value is propagated through the 16 high-order bit positions before insertion.

Op Code	Mnemonic	Type	Instruction
49	CH	RX	Compare Halfword

Description: The first operand is compared with the halfword second operand, and the result determines the setting of the condition code. The comparison is algebraic.

Op Code	Mnemonic	Type	Instruction
4A	AH	RX	Add halfword

Description: The halfword second operand is added to the first operand and the sum is placed in the first operand location. The halfword second operand is expanded to a full word before addition.

Op Code	Mnemonic	Type	Instruction
4B	SH	RX	Subtract halfword

Description: The halfword second operand is subtracted from the first operand, and the difference is placed in the first operand location. The halfword second operand is expanded to a full word before subtraction.

Op Code	Mnemonic	Type	Instruction
4C	MH	RX	Multiply halfword

Description: The product of the halfword multiplier (second operand) and the muliplicand (first operand) replaces the multiplicand. The halfword multiplier is expanded to a full word before multiplication and the low-order part of the product replaces the multiplicand (first operand).

Op Code	Mnemonic	Type	Instruction
4E	CVD	RX	Convert to decimal

Description: The radix of the first operand is changed from binary to decimal, and the result is stored in the second operand location. The number is treated as a right-aligned signed integer both before and after conversion. The result has the packed decimal format, occupies a double-word in storage, and must be located on an integral boundary.

Op Code	Mnemonic	Type	Instruction
4F	CVB	RX	Convert to binary

Description: The radix of the second operand is changed from decimal to binary, and the result is placed in the first operand location. The second operand has the packed decimal data format and occupies a double-word storage field, which must be located on an integral boundary.

Op Code	Mnemonic	Type	Instruction
50	ST	RX	Store

Description: The first operand is placed in the second operand location.

Op Code	Mnemonic	Type	Instruction
54	N	RX	AND

Description: The logical product (AND) of the first and second operands is placed in the first operand location. Operands are treated as unstructured logical quantities, and the connective AND is applied bit by bit.

Op Code	Mnemonic	Type	Instruction
55	CL	RX	Compare logical

Description: The first operand is compared with the second operand, and the result is indicated in the condition code.

Op Code	Mnemonic	Type	Instruction
56	O	RX	OR

Description: The logical sum (OR) of the bits of the first and second operands is placed in the first operand location. Operands are treated as unstructured logical quantities, and the connective inclusive OR is applied bit by bit.

Op Code	Mnemonic	Type	Instruction
57	X	RX	Exclusive OR

Description: The modulo-two sum (exclusive OR) of the first and second operands is placed in the first operand location. Operands are treated as unstructured logical quantities, and the connective exclusive OR is applied bit by bit.

Op Code	Mnemonic	Type	Instruction
58	L	RX	Load

Description: The second operand is placed in the first operand location, with the second operand left unchanged.

Op Code	Mnemonic	Type	Instruction
59	C	RX	Compare

Description: The first operand is compared with the second operand, and the result determines the setting of the condition code. The 32-bit signed integer operands are compared algebraically.

Op Code	Mnemonic	Type	Instruction
5A	A	RX	Add

Description: The second operand is added to the first operand, and the sum is placed in the first operand location.

Op Code	Mnemonic	Type	Instruction
5B	S	RX	Subtract

Description: The second operand is subtracted from the first operand, and the difference is placed in the first operand location.

Op Code	Mnemonic	Type	Instruction
5C	M	RX	Multiply

Description: The product of the multiplier (second operand) and the multiplicand (first operand) replaces the multiplicand. Multiplier and multiplicand are 32-bit signed integers and the product is a 64-bit signed integer occupying the even/odd register pair specified by the R_1 field of the instruction.

Op Code	Mnemonic	Type	Instruction
5D	D	RX	Divide

Description: The dividend (first operand) is divided by the divisor (second operand) and replaced by the remainder and the quotient. The divisor is a 32-bit signed integer. The divident is a 64-bit signed integer occupying the even/odd register pair specified by the R_1 field of the instruction. A 32-bit signed integer remainder and a 32-bit signed integer quotient replace the divident in the even-numbered and odd-numbered registers, respectively.

Op Code	Mnemonic	Type	Instruction
5E	AL	RX	Add logical

Description: The second operand is added to the first operand, and the sum is placed in the first operand location. A carry out in the sign position is recorded in the condition code. Logical addition adds all 32 bits of both operands without further change to the resulting sign bit.

Op Code	Mnemonic	Type	Instruction
5F	SL	RX	Subtract logical

Description: The second operand is subtracted from the first operand, and the difference is placed in the first operand location. A carry out in the sign position is recorded in the condition code. All 32 bits of both operands participate, without further change to the resulting sign bit.

Op Code	Mnemonic	Type	Instruction
80	SSM	SI	Set system mask

Description: The byte at the location designated by the operand address replaces the system mask bits of the current PSW.

Op Code	Mnemonic	Type	Instruction
82	LPSW	SI	Load PSW

Description: The double word at the location designated by the operand address replaces the PSW. The operand address must be a double word address. The double word which is loaded becomes the PSW for the next sequence of instructions.

Op Code	Mnemonic	Type	Instruction
86	BXH	RS	Branch on index high

Description: An increment is added to the first operand, and the sum is compared algebraically with a comparand. Subsequently, the sum is placed in the first operand location, regardless of whether the branch is taken. When the sum is high, the instruction address is replaced by the branch address. When the sum is low or equal, instruction sequencing proceeds with the updated instruction address. The first operand and the increment are in the registers specified by R_1 and R_3. The comparand register address is odd and is either one larger than R_3 or equal to R_3. The branch address is determined prior to the addition and comparison.

Op Code	Mnemonic	Type	Instruction
87	BXLE	RS	Branch on index low or equal

Description: An increment is added to the first operand, and the sum is compared algebraically with a comparand. Subsequently, the sum is placed in the first operand location, regardless of whether the branch is taken. When the sum is low or equal, the instruction address is replaced by the branch address. When the sum is high, normal instruction sequencing proceeds with the updated instruction address. The first operand and the increment are in the registers specified by R_1 and R_3. The comparand register address is odd and is either one larger than R_3 or equal to R_3. The branch address is computed prior to the addition and comparison.

Op Code	Mnemonic	Type	Instruction
88	SRL	RS	Shift right single

Description: The first operand is shifted right the number of bits specified by the low-order six bits of the second operand address field. Zero's are shifted into vacated register positions.

Op Code	Mnemonic	Type	Instruction
89	SLL	RS	Shift left single

Description: The first operand is shifted left the number of bits specified by the low-order six bits of the second operand address field. Zero's are shifted into vacated register positions.

Op Code	Mnemonic	Type	Instruction
8A	SRA	RS	Shift right single

Description: The integer part of the first operand is shifted right the number of bits specified by the low-order six bits of the second operand address field. Bits equal to the sign are supplied to vacated register positions.

Op Code	Mnemonic	Type	Instruction
8B	SLA	RS	Shift left single

Description: The integer part of the first operand is shifted left the number of bits specified by the low-order six bits of the second operand address field. Zero's are shifted into vacated register positions.

Op Code	Mnemonic	Type	Instruction
8C	SRDL	RS	Shift right double

Description: The double word first operand is shifted right the number of bits specified by the low-order six bits of the second operand address field. The R_1 field of the instruction must contain an even register address specifying an even/odd register pair. Zero's are supplied to vacated register positions.

Op Code	Mnemonic	Type	Instruction
8D	SLDL	RS	Shift left double

Description: The double word first operand is shifted right the number of bits specified by the low-order six bits of the second operand address field. The R_1 field of the instruction must contain an even register address specifying an even/odd register pair. Zero's are supplied to vacated register positions.

Op Code	Mnemonic	Type	Instruction
8E	SRDA	RS	Shift right double

Description: The double-length integer part of the first operand is shifted right the number of places specified by the low-order bits of the second operand address field. The R_1 field of the instruction must contain an even register address specifying an even/odd register pair. Bits equal to the sign bit are supplied to vacated register positions.

Op Code	Mnemonic	Type	Instruction
8F	SLDA	RS	Shift left double

Description: The double-length integer part of the first operand is shifted left the number of places specified by the low-order six bits of the second operand address field. The R_1 field of the instruction must contain an even register address specifying an even/odd register pair. Zero's are supplied to vacated register positions.

Op Code	Mnemonic	Type	Instruction
90	STM	RS	Store multiple

Description: The set of general registers starting with the register specified by R_1 and ending with the register specified by R_3 is stored at the locations designated by the second operand address. The general registers are stored in the ascending order of their addresses, starting with the register specified by R_1 and continuing through the register specified by R_3, with register 0 following register 15.

Op Code	Mnemonic	Type	Instruction
91	TM	SI	Test under mask

Description: The byte of immediate data, I_2, is used as an 8-bit mask to set the condition code, The bits of the mask are made to correspond one for one with the bits of the character in storage specified by the first operand address. A mask bit of one indicates that the storage bit is to be tested; when zero, the storage bit is ignored. When all storage bits thus selected are zero, the condition code is made zero. The code is also made zero when the mask is all-zero. When the selected bits are all-one, the code is made 3; otherwise, the code is made 1.

Op Code	Mnemonic	Type	Instruction
92	MVI	SI	Move

Description: The 8-bit byte immediate operand is placed in the first operand location.

Op Code	Mnemonic	Type	Instruction
93	TS	SI	Test and set

Description: The leftmost bit of the byte located at the first operand address is used to set the condition code, and the entire address byte is set to all ones.

Op Code	Mnemonic	Type	Instruction
94	NI	SI	AND

Description: The logical product (AND) of the bits of the first operand and the immediate operand is placed in the first operand location. Operands are treated as unstructured logical quantities, and the connective AND is applied bit by bit.

Op Code	Mnemonic	Type	Instruction
95	CLI	SI	Compare logical

Description: The first operand is compared with the immediate operand, and the result is indicated in the condition code.

Op Code	Mnemonic	Type	Instruction
96	OI	SI	OR

Description: The logical sum (OR) of the bits of the first and immediate operands is placed in the first operand location. Operands are treated as unstructured logical quantities, and the connective inclusive OR is applied bit by bit.

Op Code	Mnemonic	Type	Instruction
97	XI	SI	Exclusive OR

Description: The modulo-two sum (exclusive OR) of the bits of the first and immediate operands is placed in the first operand location. Operands are treated as unstructured logical quantities and the connective exclusive OR is applied bit by bit.

Op Code	Mnemonic	Type	Instruction
98	LM	RS	Load Multiple

Description: The set of general registers starting with the register specified by R_1 and ending with the register specified by R_3 is loaded from the locations designated by the second operand address. The general registers are loaded in the ascending order of their addresses, starting with the register specified by R_1 and continuing through the register specified by R_3, with register 0 following register 15.

GROUP III INSTRUCTIONS

Op Code	Mnemonic	Type	Instruction
D1	MVN	SS	Move numerics

Description: The low-order four bits of each byte in the second operand field, the numerics, are placed in the low-order bit positions of the corresponding bytes in the first operand fields. Movement is left to right through each field one byte at a time, and the fields may overlap in any desired way. The high-order four bits of each byte, the zones, remain unchanged.

Op Code	Mnemonic	Type	Instruction
D2	MVC	SS	Move

Description: The second storage operand is placed in the first storage operand location. Movement is left to right through each field a byte at a time and the fields may overlap in any desired way.

Op Code	Mnemonic	Type	Instruction
D3	MVZ	SS	Move zones

Description: The high-order four bits of each byte in the second operand field, the zones, are placed in the high-order four bit positions of the corresponding bytes in the first operand field. Movement is left to right through each field one byte at a time, and the fields may overlap in any desired way. The low-order four bits of each byte, the numerics, remain unchanged.

Op Code	Mnemonic	Type	Instruction
D4	NC	SS	AND

Description: The logical product (AND) of the bits of the first and second storage operands is placed in the first operand location. Operands are treated as unstructured logical quantities, and the connective AND is applied bit by bit.

Op Code	Mnemonic	Type	Instruction
D5	CLC	SS	Compare logical

Description: The first storage operand is compared with the second storage operand, and the result is indicated in the condition code. Comparison is binary, and all codes are valid.

Op Code	Mnemonic	Type	Instruction
D6	OC	SS	OR

Description: The logical sum (OR) of the bits of the first and second storage operands is placed in the first operand location. Operands are treated as unstructured logical quantities, and the connective inclusive OR is applied bit by bit.

Op Code	Mnemonic	Type	Instruction
D7	XC	SS	Exclusive OR

Description: The modulo-two sum (exclusive OR) of the bits of the first and second storage operands is placed in the first operand location. Operands are treated as unstructured logical quantities, and the connective exclusive OR is applied bit by bit.

Op Code	Mnemonic	Type	Instruction
DC	TR	SS	Translate

Description: The eight-bit bytes of the first operand are used as arguments to reference the list designated by the second operand address. Each eight-bit function byte selected from the list replaces the corresponding argument in the first operand. The bytes of the first operand are selected one by one for translation, proceeding left to right. All data is valid and the operation proceeds until the first operand field is exhausted.

Op Code	Mnemonic	Type	Instruction
DD	TRT	SS	Translate and test

Description: The eight-bit bytes of the first operand are used as arguments to reference the list designated by the second operand address. Each eight-bit function byte thus selected from the list is used to determine the continuation of the operation. When the function byte is a zero, the operation proceeds by fetching and translating the next argument byte. When the function byte is non-zero, the operation is completed by inserting the related argument address in general register 1, and inserting the function byte in general register 2. Fetching of the function byte from the list proceeds as in TRANSLATE. When the first operand field is exhausted before a non-zero function byte is encountered, the condition code is set to 0. The condition code is set to 1 when one or more argument bytes have not been translated. The condition code is set to 2 if the last function byte is non-zero.

Op Code	Mnemonic	Type	Instruction
F1	MVO	SS	Move with offset

Description: The second operand is placed to the left of and adjacent to the low-order four bits of the first operand. The fields are processed right to left. If necessary, the second operand is extended with high-order zero's. If the first operand field is too short to contain all bytes of the second operand, the remaining information is ignored.

Op Code	Mnemonic	Type	Instruction
F2	PACK	SS	Pack

Description: The format of the second operand is changed from zoned to packed, and the result is placed in the first operand location. The fields are processed right to left. If necessary, the second operation is extended with high-order zero's. If the first operand field is too short to contain all significant digits of the second operand field, the remaining high-order bits are ignored. Overlapping fields may occur.

Op Code	Mnemonic	Type	Instruction
F3	UNPK	SS	Unpack

Description: The format of the second operand is changed from packed to zoned, and the result is placed in the first operand location. The fields are processed right to left. The second operand is extended with high-order zero digits before unpacking, if necessary. If the first operand field is too short to contain all significant digits of the second operand field, the remaining high-order digits are ignored. Overlapping fields may occur.

APPENDIX IV. SUMC SIMULATOR SOURCE PROGRAM

This appendix contains the complete source listing for the basic SUMC interpretive simulator. The source language is FORTRAN IV and the program has been developed for operation on an IBM 7094 host computer.

```
$JCB            C114 CURRAN        ,300790,00,12,I4MCP
$JCB            C114 CURRAN        ,300790,00,12,I4MCP
$EXECUTE        IBJCB
$IBJCB          NCMAP
$IBFTC CCR001   DECK
C     MAILIN
      COMMON /TES/ LEVELS(4),IOUTPU(5),IPOINT(5)
      LOGICAL CSLAE,IRESTA
      COMMON/UNCON/IFWA,IHOST,ITARG,IOPCW,IBIT,IPARS,IDATAS,IHW,MAXCOR,
     1 LCCG,MPCS,TIMBC,NRCODE,NSTAT,LBPS1,LBPS2,NULL,IONE,IT,
     1CSLAE,ILC,INTCDE,ICOUNT,EXTIME,TIMLIM,
     1IMM,IMN,IERFLG,IRESTA,
     1LCBOUN,JIBOUN,MONE,IFETC
     1,MAXNEG,MAXBIT
       COMMON /UARRAY/ ISEGTA(20),TIME(10),CLASS(10),INSDAT(256,5),
     1INSPAR(5,20),IPCTR(5),IPPCTR(5),ISGCD(5,20),IADR(3),ISPM(64),
     1IHALFW(6),ISTACK(5,6),IMAINM(4096)
      COMMON /CVAR/ICPCC(255),ITCNTR,IOFFST,IOPRND(3),ISTAT
      DIMENSION FLPTRG(8)
      INTEGER  EXOPSW(4),SVCOPS(4),PROPSW(4),MAOPSW(4),IOOPSW(4),
     1CSWCRD(4),CAWCRD(4),EXNPSW(4),SVCNPS(4),PRNPSW(4),MANPSW(4),
     1 EXOLCC,SVOLCC,PROLCC,EXNLOC,SVNLOC,PRNLOC,
     1ICNPSW(4),CURPSW(4),GENREG(16),       TSTORG(10)
      EQUIVALENCE (ISYSMK,ISPM(26)),(IPRMSK,ISPM(27)),(IAWP,ISPM(29)),
     1(ICC,ISPM(28))
      EQUIVALENCE(IMAINM(1),INIPSW),(IMAINM(3),INCCW1),(IMAINM( 5),
     1 INCCW2),(IMAINM( 7),EXOPSW),(IMAINM( 9),SVCOPS),(IMAINM(11),
     1 PROPSW),(IMAINM(13),MAOPSW),(IMAINM(15),IOOPSW),(IMAINM(17),
     1 CSWCRD),(IMAINM(19),CAWORD),(IMAINM(23),EXNPSW),(IMAINM(25),
     1 SVCNPS),(IMAINM( 27),PRNPSW),(IMAINM( 29),MANPSW),(IMAINM( 31),
     1 ICNPSW),(ISPM(25),INACCR),(ISPM(25),CURPSW),(ISPM,GENREG),
     1 (ISPM(17),FLPTRG),(ISPM(33),TSTORG),(IMAINM(1),INPRCT),
     1 (IMAINM( 3),EXOLOC),(IMAINM(10),SVOLOC),(IMAINM(12),PROLOC),
     1 (IMAINM(14),MAOLOC),(IMAINM(15),IOOLOC),(IMAINM(24),EXNLOC),
     1 (IMAINM(26),SVNLOC),(IMAINM( 28),PRNLOC),(IMAINM( 30),MANLOC),
     1 (IMAINM( 32),ICNLOC)
      LOGICAL SNAP
      LOGICAL FSNAP,TISNAP,KSNAP,
     1         PCSNAP,CCSNAP,TSNAP,LTSNAP(9),MASNAP,RASNAP,MOSNAP,ROSNAP
      INTEGER SLOC(9),TISLOC(9),NTSKL(9),TSLOC(9,9)
      REAL TSK(9)
      INTEGER PCSK(9),NPCSKL(9),PCSLOC(9,9),
     1         CCSK(9),NCCSKL(9),CCSLOC(9,9),
     1         MASK(9),NMASKL(9),MASLOC(9,9),
     1         RASK(9),NRASKL(9),RASLOC(9,9),
     1         MOSKA(9),MOSK(9),NMOSKL(9),MOSLOC(9,9),
     1         ROSKA(9),ROSK(9),NROSKL(9),ROSLOC(9,9)
      LOGICAL TRACE
      LOGICAL FTRACE,TITRAC,KTRACE,
     1         PCTRAC,CCTRAC,TTRACE,LTTRAC(9),MATRAC,RATRAC,MOTRAC,ROTRAC
      INTEGER PCTK(9),NIPCTK(9),CCTK(9),NICCTK(9),NITTK(9),
     1         MATK(9),NIMATK(9),RATK(9),NIRATK(9),
     1         MOTKA(9),MOTK(9),NIMOTK(9),ROTKA(9),ROTK(9),NIROTK(9)
      REAL TTK(9)
      LOGICAL SPMD
      LOGICAL PCSPMD,CCSPMD,TSPMD,LTCK(9),MASPMD,RASPMD,MOSPMD,ROSPMD
      INTEGER PCCK(9),CCCK(9),MACK(9),RACK(9),
     1         MOCKA(9),MOCK(9),ROCKA(9),ROCK(9)
      REAL TCK(9)
```

```
      INTEGER BPCDK(9),BDSTAD(9),NBDL(9)
      LOGICAL FMMC
      EXTERNAL ITWTSM,ISMTWC
      COMMON /FS/SNAP,FSNAP,NFSL,SLOC,
     1                 TISNAP,STI,SSTRT,SSTPT,NTISNA,TISLOC
     1       /KS/KSNAP,PCSNAP,NPCSK,PCSK,NPCSKL,PCSLOC,
     1                 OCSNAP,NOCSK,OCSK,NOCSKL,OCSLOC,
     1                 MASNAP,NMASK,MASK,NMASKL,MASLOC,
     1                 RASNAP,NRASK,RASK,NRASKL,RASLOC,
     1                 TSNAP,NTSK,TSK,NTSKL,TSLOC,LTSNAP,
     1                 MOSNAP,NMOSK,MOSKA,MOSK,NMOSKL,MOSLOC,
     1                 ROSNAP,NROSK,ROSKA,ROSK,NROSKL,ROSLOC
      COMMON /TR/TRACE,FTRACE,TITRAC,TRTI,TSTRT,TSTPT,KTRACE
     1       /KT/PCTRAC,NPCTK,PCTK,NIPCTK,OCTRAC,NOCTK,OCTK,NIOCTK,
     1            MATRAC,NMATK,MATK,NIMATK,RATRAC,NRATK,RATK,NIRATK,
     1            TTRACE,NTTK,TTK,NITTK,LTTRAC,
     1            MCTRAC,NMOTK,MOTKA,MOTK,NIMOTK,
     1            RCTRAC,NROTK,ROTKA,ROTK,NIRCTK
      COMMON /SP/SPMC,PCSPMC,NPCDK,PCDK,OCSPMD,NOCDK,OCDK
     1            ,MASPMC,NMADK,MADK,RASPMD,NRADK,RADK,TSPMD,NTDK,TDK,LTDK
     1            ,MOSPMD,NMODK,MODKA,MODK,ROSPMD,NRODK,RODKA,RODK
     1       /MM/FMMC,BLMMC,NBLDK,BPCDK,BDSTAD,NBDL
C         INITIALIZATION
      CALL INITLZ
      IF(IERFLG.NE.0) GO TO 71
   10 CONTINUE
C     INTERRUPT DETECTION LOOP
      IF(ICOUNT.EQ.0) GO TO 52
      DO  51 I= 1,4
      IPR=LEVELS(I)
      CALL PULL (IPR,J,IPRSTA,ICSWO)
   51 CONTINUE
      IT=1
   52 CALL FECHM
      IF (SNAP) CALL SNAPEX
      IF (TRACE) CALL TRACEX
      IF (SPMC) CALL SPMCEX
      IF (BLMMC) CALL BLMCEX
      IF(IERFLG.NE.0) GO TO 71
   50  CALL CPCEP
      IF(IERFLG.NE.0) GO TO 71
   73 CALL TIMER
      IF(IERFLG.EQ.0) GOTO 10
   71 CALL TERMIN
   72 CALL EXIT
      RETURN
      END
$*
$IBFTC CCR002  DECK
      BLOCK DATA
      LOGICAL CSLAE,IRESTA
      EQUIVALENCE (ISPM(25),INACDR)
       LOGICAL IFETC
      COMMON /TES/ LEVELS(4),IOUTPU(5),IPOINT(5)
      COMMON/UNCON/IFWA,IHOST,ITARG,IOPCW,IBIT,IPARS,IDATAS,IHW,MAXCOR,
     1 LOCG,MPOS,TIMBC,NRCODE,NSTAT,LBPS1,LBPS2,NULL,IONE,IT,
     1CSLAE,ILC,INTCDE,ICOUNT,EXTIME,TIMLIM,
     1IMM,IMN,IERFLG,IRESTA,
     1LOWCUN,JIMCUN,MONE,IFETC
     1,MAXNEG,MAXBIT
       COMMON /UARRAY/ ISEGTA(20),TIME(10),CLASS(10),INSDAT(256,5),
     1INSDPU(5,20),IECTR(5),IPRCTR(5),ISGCO(5,20),IADR(3),ISPM(64),
```

```
      DATA LEVELS/ 1,4,3,2/
      DATA IHCST,ITARG,IOPCW,IBIT/36,32,8,1/
      DATA  LBPS1,LBPS2/ 30,28/
      DATA ( (INSDAT(I,J), I=1,255), J=1,2 )
     1          / 0,0,0,9,5,5,5,9,9,10,0,0,0,0,0,1,1,4,1,3,4,3,3,1,4,
     12,2,2,2,2,2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
     10,0,0,0,0,0,1,1,1,1,10,5,5,5,1,3,2,2,2,0,8,8,1,0,0,0,3,4,3,3,1,4,
     12,2,2,2,2,2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
     10,0,0,0,0,9,0,9,9,9,9,4,4,6,6,6,6,6,6,6,6,1,4,1,4,7,7,7,7,1,0,0,0,
     17,7,7,7,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
     10,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,10,10,10,10,4,3,3,0,0,0,0,
     10,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
     10,0,0,0,
     10,0,1,1,1,1,1,1,1,1,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
     11,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
     12,2,2,2,2,2,2,2,2,2,2,2,0,2,2,2,2,0,0,2,2,2,2,2,2,2,2,2,2,2,
     12,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,
     14,4,4,4,4,4,4,3,3,3,3,3,3,3,3,3,3,3,4,4,4,4,4,4,4,3,1,1,4,4,4,4,
     11,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
     10,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,5,5,5,5,5,5,5,0,0,0,0,0,0,0,0,
     10,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
     10/
      DATAISGCD/1,1,1,1,1,1,2,2,2,14,14,3,8,3,5,15,0,5,5,11,5,0,
     111,11,0,11,0,0,0,0,5,0,0,0,0,12,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
     10,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
     1 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0/
      DATA  INSPAR/ 1,2,2,2,3,3,3,3,2,3,0,2,2,2,2,0,0,0,0,2,8,8,8,8,8,
     1 4,4,4,8,4,4,4,4,4,4,4,0,4,4,12,4,0,12,12,0,12,0,0,0,0,4,0,0,0,0,12,
     1 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
     1 0,0,0,0,0,0,0,0,0,0,0,0,0/
      END
$IBFTC CCR003  DECK
      SUBROUTINE FECHM
      EQUIVALENCE (ISPM(25),INADDR)      ,(ILC,NRPWDS)
      EQUIVALENCE (ISYSMK,ISPM(26)),(IPRMSK,ISPM(27)),(IAWP,ISPM(29)),
     1(ICC,ISPM(28))
      COMMON /TES/ LEVELS(4),IOUTPU(5),IPOINT(5)
      COMMON/UNCON/IFWA,IHCST,ITARG,IOPCW,IBIT,IPARS,IDATAS,IHW,MAXCOR,
     1 LCCG,MPCS,TIMBC,NRCODE,NSTAT,LBPS1,LBPS2,NULL,IONE,IT,
     1CSLAE,ILC,INTCDE,ICOUNT,EXTIME,TIMLIM,
     1IMM,IMN,IERFLG,IRESTA,
     1LCBCUN,JIBCUN,MONE,IFETC
     1,MAXNEG,MAXBIT
       COMMON /UARRAY/ ISEGTA(20),TIME(10),CLASS(10),INSDAT(256,5),
     1INSPAR(5,20),IPCTR(5),IPPCTR(5),ISGCD(5,20),IADR(3),ISPM(64),
     1IHALFW(6),ISTACK(5,6),IMAINM(4096)
      COMMON /DVAR/IOPCD(255),ITCNTR,IOFFST,IOPRND(3),ISTAT
      LOGICAL CSLAE,IRESTA
      IA=IHCST-IHW
      IDATAS=INADDR
      CALL BRCHK(J)
      GO TO (2018,3000),J
 2018  IF(INADDR.GE.LCBCUN) GO TO 2020
 2019 IERFLG=14
 3000 CALL ERINS
      RETURN
 2020 IF(INADDR.GT.JIBCUN) GO TO 2019
C     POF  = 0   OR BOUNDARY EXCURSION TERMINATES
 2006 JJ=MOD(INADDR,2)
      JJ=JJ+1
      GO TO (2006,2019),JJ
```

```
      IPCTR(IT)=INACCR/4+1
      JJ=INACCR/2
      =MOD(JJ,2)
      JJ=0
       I=I+1
C     30 EVEN HW        2001 ODD HW
      GO TO (30,2001),I
30    CSLAE=.FALSE.
      ICPSTA=ITARG
2003   I= IPCTR(IT)
C     GET  OP CODE TO SEGMENT TABLE
      ISEGTA(1)=0
      KK=IMAINM(I)
      K=ILOAD(KK,ICPSTA,ICPCW)
      ISEGTA(1)=K
      IF(K.EQ.0.OR.K.GT.255) GO TO 8
      GO TO (8,8,8,9,9,9,9,8,8,9,8,8,8,8,8,9,9,9,9,9,9,9,9,9,9,9,
     19,9,9,9,9,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,
     18,8,8,8,8,8,9,9,9,9,9,9,9,9,9,9,9,9,9,8,9,9,9,9,9,9,9,9,9,9,9,9,
     19,9,9,9,9,9,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,
     18,8,8,8,8,9,8,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,8,
     18,8,9,9,9,9,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,
     18,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,9,9,9,9,9,9,9,8,8,8,
     18,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,
     18,8,8,8,8      ) , K
8     IERFLG=ONE
      WRITE(  )K    ,IERFLG
      GO TO 3
C     GET P     CLASS
C     IPARS=   AT(K,2)
C     GET S    ICS CLASS
      ISTAT=   DAT(K,  )
2009   NRHWDS=INSPAR(IPARS,1)
      N=5
C     M IS STARTING BIT POSITION IN  SOURCE WORD
      DO 2010 IL=1,NRHWDS
      M=ICPSTA
      J=INSPAR(IPARS,IL+1)
C     J IS NR OF SEG IN A HW
      DO 2007  L=1,J
      JJ=JJ+1
      MM=IHCST-INSPAR(IPARS,N)
C     MM IS START BIT IN  DESTINATION
      ISGCTR= ISGCC (IPARS,JJ)
      KK=IMAINM(I)
      KA=INSPAR(IPARS,N)
      ISEGTA(ISGCTR)=ILOAD(KK,M,KA)
      M=M-KA
2007  N=N+1
      IHALFW(IL)=ILOAD(KK,ICPSTA,IHW)
2     FORMAT (I10)
      INACCR=INACCR+2
C     TOGGLE CSLAE
       CSLAE=  .NOT.CSLAE
      IF(CSLAE) GO TO 2002
C      T= E
      IPPCTR(IT) = IPCTR(IT)
      IPCTR(IT) - IPCTR(IT) +1
      I=I+1
      ICPSTA=ITARG
2010  CONTINUE
C     CALCULAT  EFFECTIVE ADDRESS
```

```
C     CASE
          GO TO 2026
 2021 K=0
          GO TO 2025
 2024 L=0
 2026  GO TO (2011,2012,2013,2014,2015,2016), IPARS
C     PARSE CLASS 1    RR
 2011 IMM= LOCG + ISEGTA(2)
      IMN= LOCG + ISEGTA(3)
      GO TO  10
C     PARSE CLASS 2    RX
 2012 IADR(1)=K+L+ISEGTA(11)
      IMM=LOCG+ISEGTA(2)
      GO TO 32
C     PARSE CLASS 3    RS
 2013 IADR(1)=L+ISEGTA(11)
      IMM=LOCG+ISEGTA(2)
      IMN=LOCG+ISEGTA(3)
      GO TO 32
C     PARSE CLASS 4    SI
 2014 IADR(1)=L+ISEGTA(11)
      GO TO 32
C     PARSE CLASS 5    SS
 2015 IADR(1)=L+ISEGTA(11)
      IF(ISEGTA(6).GT.0) GO TO 2028
 2027 M=0
       GO TO 2029
 2028  IMM= LOCG+ISEGTA(6)
      M=ISPM(IMM)
 2029 IADR(2)=M+ISEGTA(12)
      K=IADR(2)
      CALL FECHFW(K,$31)
      ICPRND(2)=IDATAS
 32   K=IADR(1)
      CALL FECHFW(K,$31)
      ICPRND(1)=IDATAS
 10   CONTINUE
      RETURN
 31   IERFLG=14
      GO TO 3000
C      PARSE CLASS  6  NOT IMPLEMENTED
 2016  CALL HALTE
      RETURN
 2002  ICPSTA=IHW
C     EVEN
      GO TO 2010
 2001  OSLAE= .TRUE.
      ICPSTA=IHW
C     THIS HW  CCC
      GO TO 2003
      END
$IBFTC CCR004  DECK
      SUBROUTINE STATCP
      INTEGER  EXOPSW(4),SVCOPS(4),PROPSW(4),MAOPSW(4),IOOPSW(4),
     1CSWORD(4),CAWORD(4),EXNPSW(4),SVCNPS(4),PRNPSW(4),MANPSW(4),
     1 EXOLOC,SVOLOC,PROLOC,EXNLOC,SVNLOC,PRNLOC,
     1IONPSW(4),CURPSW(4),GENREG(16),          TSTORG(10)
      EQUIVALENCE (ISYSMK,ISPM(26)),(IPRMSK,ISPM(27)),(IAWP,ISPM(29)),
     1(ICC,ISPM(28))
      EQUIVALENCE(IMAINM(1),INIPSW),(IMAINM(3),INCCW1),(IMAINM( 5),
     1 INCCW2),(IMAINM( 7),EXOPSW),(IMAINM( 9),SVCOPS),(IMAINM(11),
     1 PROPSW),(IMAINM(13),MAOPSW),(IMAINM(15),IOOPSW),(IMAINM(17),
```

```
     1 SVCNPS),(IMAINM( 27),PRNPSW),(IMAINM( 29),MANPSW),(IMAINM( 31),
     1 ICNPSW),(ISPM(25),INADCR),(ISPM(25),CURPSW),(ISPM,GENREG),
     1 (ISPM(17),FLPTRG),(ISPM(33),TSTORG),(IMAINM(1),INPRCT),
     1 (IMAINM( 8),EXCLOC),(IMAINM(10),SVOLOC),(IMAINM(12),PROLOC),
     1 (IMAINM(14),MACLOC),(IMAINM(15),IOOLOC),(IMAINM(24),EXNLOC),
     1 (IMAINM(26),SVNLOC),(IMAINM( 28),PRNLOC),(IMAINM( 30),MANLOC),
     1 (IMAINM( 32),ICNLOC)
      DIMENSION  PCNT(10)
      LOGICAL CSLAE,IRESTA
      COMMON /TES/ LEVELS(4),IOUTPU(5),IPOINT(5)
      COMMON/UNCON/IFWA,IFCST,ITARG,IOPCW,IBIT,IPARS,IDATAS,IHW,MAXCOR,
     1 LOCG,MPCS,TIMEO,NRCODE,NSTAT,LBPS1,LBPS2,NULL,IONE,IT,
     1CSLAE,ILC,INTCDE,ICOUNT,EXTIME,TIMLIM,
     1IMM,IMN,IERFLG,IRESTA,
     1LOBCUN,JIHCUN,MONE,IFETC
     1,MAXNEG,MAXBIT
       COMMON /UARRAY/ ISECTA(20),TIME(10),CLASS(10),INSDAT(256,5),
     1INSPAR(5,20),IPCTR(5),IPPCTR(5),ISGCD(5,20),IADR(3),ISPM(64),
     1IHALFW(6),ISTACK(5,6),IMAINM(4096)
      COMMON /DVAR/IOPCO(255),ITCNTR,IOFFST,IOPRND(3),ISTAT
      SUM1= 0.
      SUM2= 0.
      DO 10 I=1,NSTAT
   10 SUM1=SUM1+CLASS(I)
      DO 20 I=1,NSTAT
      PCNT(I)= CLASS(I)/SUM1* 100.
   20 SUM2= SUM2+PCNT(I)
C     OUTPUT HEADERS
      WRITE (6,3)
      WRITE(6,1) (I,CLASS(I),TIME(I),PCNT(I),I=1,NSTAT)
      WRITE (6,2) SUM1,TIMEC,SUM2
      RETURN
    1 FORMAT(I11,1X,F14.1,7X,F14.1,7X,F14.1)
    2 FORMAT(12X,F14.1,7X,F14.1,7X,F14.1)
    3 FORMAT(35H      INSTRUCTION CLASS DISTRIBUTION///68H        CLASS
     1       COUNT                 TIME                 PERCENT)
      END
$*
$IBFTC CORO05  DECK
      SUBROUTINE OPDEF
      INTEGER  EXOPSW(4),SVCOPS(4),PROPSW(4),MAOPSW(4),IOOPSW(4),
     1CSWORD(4),CAWCRD(4),EXNPSW(4),SVCNPS(4),PRNPSW(4),MANPSW(4),
     1 EXCLOC,SVOLOC,PROLOC,EXNLOC,SVNLOC,PRNLOC,
     1ICNPSW(4),CURPSW(4),GENREG(16),          TSTORG(10)
      EQUIVALENCE  (ISYSMK,ISPM(26)),(IPRMSK,ISPM(27)),(IAWP,ISPM(29)),
     1(ICC,ISPM(28))
      EQUIVALENCE(IMAINM(1),INIPSW),(IMAINM(3),INCCW1),(IMAINM( 5),
     1 INCCW2),(IMAINM( 7),EXOPSW),(IMAINM( 9),SVCOPS),(IMAINM(11),
     1 PROPSW),(IMAINM(13),MAOPSW),(IMAINM(15),IOOPSW),(IMAINM(17),
     1 CSWORD),(IMAINM(19),CAWORD),(IMAINM(23),EXNPSW),(IMAINM(25),
     1 SVCNPS),(IMAINM( 27),PRNPSW),(IMAINM( 29),MANPSW),(IMAINM( 31),
     1 ICNPSW),(ISPM(25),INADCR),(ISPM(25),CURPSW),(ISPM,GENREG),
     1 (ISPM(17),FLPTRG),(ISPM(33),TSTORG),(IMAINM(1),INPRCT),
     1 (IMAINM( 8),EXCLOC),(IMAINM(10),SVOLOC),(IMAINM(12),PROLOC),
     1 (IMAINM(14),MACLOC),(IMAINM(15),IOOLOC),(IMAINM(24),EXNLOC),
     1 (IMAINM(26),SVNLOC),(IMAINM( 28),PRNLOC),(IMAINM( 30),MANLOC),
     1 (IMAINM( 32),ICNLOC)
      LOGICAL CSLAE,IRESTA
      COMMON /TES/ LEVELS(4),IOUTPU(5),IPOINT(5)
      COMMON/UNCON/IFWA,IFCST,ITARG,IOPCW,IBIT,IPARS,IDATAS,IHW,MAXCOR,
     1 LOCG,MPCS,TIMEO,NRCODE,NSTAT,LBPS1,LBPS2,NULL,IONE,IT,
```

```
      DATA IBIT15/32678/
C     FIND CP CODE  INDEX AND  CALL  SUBROUTINE
      KADR=2**25-1
C    KADR IS HW DEP
C     INSDAT IS VECTOR W  INSTRUCTION PARAMETERS
      IF(ICPCD.EQ.0) GC TC 3002
 1000    GC TC (3002,3002,3002,4,5,6,7,8,9,10,3002,3002,3002,3002,3002,
     116,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,3002,3002,3002,
     13002,3002,3002,3002,3002,3002,3002,3002,3002,3002,3002,3002,3002,
     13002,3002,3002,3002,3002,3002,3002,3002,3002,3002,3002,3002,3002,
     13002,3002,3002,64,65,66,67,68,69,70,71,72,73,74,75,76,3002,78,79,
     180,3002,3002,3002,84,85,86,87,88,89,90,91,92,93,94,95,3002,3002,
     13002,3002,3002,3002,3002,3002,3002,3002,3002,3002,3002,3002,3002,
     13002,3002,3002,3002,3002,3002,3002,3002,3002,3002,3002,3002,3002,
     13002,3002,3002,3002,128,3002,130,131,132,133,134,135,136,137,138,
     1139,140,141,142,143,144,145,146,147,148,149,150,151,152,3002,
     13002,3002,156,157,158,159,3002,3002,3002,3002,3002,3002,3002,3002,
     13002,3002,3002,3002,3002,3002,3002,3002,3002,3002,3002,3002,3002,
     13002,3002,3002,3002,3002,3002,3002,3002,3002,3002,3002,3002,3002,
     13002,3002,3002,3002,3002,3002,3002,3002,3002,3002,3002,3002,3002,
     13002,3002,209,210,211,212,213,214,215,3002,3002,3002,3002,3002,
     13002,3002,3002,3002,3002,3002,3002,3002,3002,3002,3002,3002,3002,
     13002,3002,3002,3002,3002,3002,3002,3002,3002,3002,3002,3002,3002,
     13002,3002,3002,3002,3002,3002,3002,3002,3002,3002 ), IOPCD
C
C
C
C
C     INSTRUCTION EXECUTION
C     OP CODES  01,02,03  NOT USED
C      04  IS  SPM    BITS 2-7  OF  R1  GO TO  PSW  CC,PROG MASK
 4    ICC=ILOAD(ISPM(IMM),LBPS1,2)
      IPRMSK=ILOAD(ISPM(IMM),LBPS2,4)
      GO TC 300
C     05 IS BALR
 5    K=INADDR
      INADDR=IAND(KADR,ISPM(IMN))
      ISPM(IMM)=K
      ISPM(IMM)=ISTORE(IPRMSK,ISPM(IMM),LBPS2,4)
      ISPM(IMM)=ISTORE(ICC,ISPM(IMM),LBPS1,2)
      IF(ISEGTA(3).EC.0) GO TO 300
 3000 IDATAS=INADDR
      CALL BRCHK(J)
      GO TC 301
C     06   BCTR
 6    ISPM(IMM)=ISPM(IMM)-1
      IF(ISEGTA(3).EC.0) GO TO 300
      IF(ISPM(IMM).EC.0) GO TO 300
 3001 IPPCTR(IT) = IPCTR(IT)
      INADDR=IAND(KADR,ISPM(IMN) )
      IDATAS=INADDR
      CALL BRCHK(J)
      GO TC 301
C
C     07    BCR
 7    IF(ISEGTA(2).EC.0.OR.ISEGTA(3).EQ.0) GO TO 300
 3003 IF(ISEGTA(2)-15)2999,3005,3002
 3005 INADDR=IAND(ISPM(IMN),KADR)
      IDATAS=INADDR
      CALL BRCHK(J)
      GO  TC 301
```

```
      J=JEACC(MASK)
      GC TO (300,3005),J
3002  IERFLG=IONE
2998  CALL ERINS
      RETURN
8     GC  TC 3002
C     8  NCT USED, 9 NCT
9     GC TC 3002
C     10    SVC  0A
10    K=INADDR-1
      CALL FECHAR(K,$301)
      K=IDATAS
      CALL PUSH(2,K,0,$301)
2     FORMAT(5I15)
      GO TC 300
C
C     16  LPR   10    OB -OF  NOT USED
16    N=ISPM(IMN)
      IF(N.EQ.0) GCTO3007
      IF(IAND(MAXNEG,N).EQ.C) GO TO 3C08
      IF(N.LE.MAXNEG) GO TO 3C19
      ISPM(IMM)=INTNCT(N)+1
      ICC=2
      GC TC 300
3007  ICC=0
C     SET  CC 8   0
3C09  ISPM(IMM)= ISPM(IMN)
      GO TC 300
3008  ICC=2
C     SET CC  2   CRT 0
      GO TC 3009
C
C     17   LNR   11
17    IF(ISPM(IMN).EC.0) GO TO 3013
      IF(IANC(MAXNEG,ISPM(IMN)).EQ.0) GO TO 3014
3012  ICC=1
C     SET CC 4  LTZ
3015  ISPM(IMM)=ISPM(IMN)
      GC  TC 300
3013  ICC=0
C     SET CC 8   Z
      GC TC 3015
3014  ISPM(IMM)=INTNCT(ISPM(IMN))+1
      ICC=1
      GC  TC 300
C
C
C     18    LTR   12
18     ISPM(IMM)=ISPM(IMN)
      IF(ISPM(IMN).EC.0) GO TC 3C17
      IF(IANC(MAXNEG,ISPM(IMN)).EQ.0) GO TO 3018
3016  ICC=1
      GC TC 300
3017  ICC=0
      GC TC 300
3018  ICC=2
      GC TC 300
C
C     19  LCR   13
19    ISPM(IMM)=INTNCT(ISPM(IMN))+1
      IF(ISPM(IMM).EC.0) GO TO 3026
      IF(IAND(MAXNEG,ISPM(IMM)).EQ.0) GO TO 3023
```

```
C        SET CC 4  LTZ
         GO TO 300
 3023    ICC=2
C        SET CC 2   GRTZ
         GO TO 300
C
C
C         20   NR  14
 20      L=ISPM(IMM)
         M=ISPM(IMN)
         ISPM(IMM)=IAND(L,M)
 3027    IF(ISPM(IMM).EQ.0) GO TO 3026
 3025    ICC=1
         GO TO 300
 3026    ICC=0
C        SET CC 8   Z
         GO TO 300
C
C
C         21  CLR  15
 21      IF(ISPM(IMM).EQ.ISPM(IMN)) GO TO 2901
         IF(ISPM(IMM).LT.ISPM(IMN)) GO TO 2900
         GO TO 2902
 2901    CONTINUE
         ICC=0
         GO TO 300
 2900    ICC=1
         GO TO 300
 2902    ICC=2
         GO TO 300
C        22    OR    16
C
 22      L=ISPM(IMM)
         M=ISPM(IMN)
 3028    ISPM(IMM)= IOR(L,M)
         GO TO 3027
C
C
C        23   XR  17
 23      L=ISPM (IMM)
         M=ISPM (IMN)
 3048    ISPM(IMM) = IER(L,M)
         GO TO 3027
C
C
C
C        24   LR  18
 24      ISPM(IMM)=ISPM(IMN)
         GO  TO 300
C
C
C
C        25  CR   19
 25      N=ISPM(IMN)
         M=ISPM(IMM)
 3047    N=ITWTSM(N)
         M=ITWTSM(M)
         IF(M.GT.N) GO TO 3031
         IF(M.EQ.N) GO TO 3030
 3029    ICC=1
C        SET  CC 4  1ST CP  LC
         GO TO 300
```

```
C      SET CC 8  EQ
       GO TO 300
 3031  CC=2
C      SET CC 2   1ST OP HI
       GO TO 300
C
C      26  AR   1A
 26    K=ISPM(IMM)
       ICATAS=ISPM(IMN)
       L=ITWTSM(K)+ITWTSM(ICATAS)
       CALL IVERFL(L,J)
       ISPM(IMM)=ISMTWC(L)
       GO  TO (3032,3033), J
 3032  ICC=3
C      J=1  C/F
       CALL PUSH(2,8,0,$300)
       GO  TO 300
 3033  IF(ISPM(IMM).EQ.0) GO TO 3035
       IF(IAND(MAXNEG,ISPM(IMM)).EQ.0) GO TO 3036
 3034  ICC=1
       GO  TO  300
 3035  ICC=0
       GO  TO  300
 3036  ICC=2
       GO  TO  300
C
C      27   SR        1B
 27    K=ISPM(IMM)
       M=ISPM(IMN)
       L=ITWTSM(K)-ITWTSM(M)
       ICATAS=INTNOT(M)+1
       CALL IVERFL(L,J)
       ISPM(IMM)=ISMTWC(L)
       GO TO (3037,3033),J
 3037  ICC=3
       CALL PUSH(2,8,0,$300)
       GO TO 300
C
C
C
C      28  MR        1C
 28    J= MOD(ISEGTA(2),2 )
       J=J+1
       GO TO (3058,3042),J
  3042 CALL PUSH(2,6,0,$3002)
       GO TO 300
C***** INITIALIZE CONSTANTS *****
  3058 ISK=0
       IHTD=IHOST-ITARG
       IML=IMM+1
C***** READ MULTIPLICAND O CONVERT IF NECESSARY *****
       ICAND=ISPM(IML)
       IF (ISEGTA(1).EQ.76) ICAND=ISPM(IMM)
       IF (ICAND.LT.MAXNEG) GO TO 510
       ISK=ISK+1
       ICAND=ICCMP1(ICAND,ITARG,ITARG)
       ICAND=ICAND+1
C***** READ MULTIPLIER O CONVERT IF NECESSARY *****
   510 IF (ISEGTA(1).EQ.28) IIER=ISPM(IMN)
       IF (ISEGTA(1).EQ.92) IIER=IOPRND(1)
       IF (ISEGTA(1).EQ.76) IIER=ICATAS
```

```
      IIER=IIER+1
C***** INITIALIZE PARAMETERS FOR ITERATIVE MULTIPLICATION *****
  511 IPRR=0
      IMAR=0
      IF ((ICANC.EQ.0).OR.(IIER.EQ.0)) GO TO 500
      INB=4
      IKEY=0
      IC=ITARG/4
      IF (ISEGTA(1).EQ.76) IC=ITARG/8
C***** ITERATIVE 4-BIT MULTIPLICATION *****
      DO 501 I=1,IC
      IKEY=ILOAD(IIER,INB,4)
      IPRR=IPRR+IKEY*ICANC
      IMAR=ISTORE(IPRR,IMAR,INB,4)
      IPRR=IPRR/16
      INB=INB+4
  501 CONTINUE
C***** CONVERT PRODUCT IF NECESSARY *****
      IF (ISEGTA(1).EQ.76) GO TO 513
      IF (ISK.NE.1) GO TO 500
      IPRR=ICOMP1(IPRR,ITARG,ITARG)
      IF (IMAR.NE.0) GO TO 503
      IPRR=IPRR+1
      GO TO 500
  503 IMAR=ICOMP1(IMAR,ITARG,ITARG)
      IMAR=IMAR+1
C***** STORE PRODUCT IN SPM O RETURN TO MAINLINE *****
  500 ISPM(IMM)=IPRR
      IF (ISEGTA(1).EQ.76) GO TO 300
      ISPM(IML)=IMAR
      GO TO 300
  513 IF ((ISK.NE.1).OR.(IMAR.EQ.0)) GO TO 514
      IMAR=ICOMP1(IMAR,IHW,IHW)
      IMAR=IMAR+1
  514 ISPM(IMM)=IMAR
      GO TO 300
C
C     29      CR    10
   29 J=MOD(ISEGTA(2),2)
      J=J+1
      GO TO (4001,4002),J
 4002 CALL PUSH(2,6,0,$3002)
      GO TO 300
C***** INITIALIZE CONSTANTS O READ DIVIDEND *****
 4001 IML=IMM+1
      ISSK=0
      IDSK=0
      IHTO=IHOST-ITARG
      IQUO=0
      IPR=0
      IDEND=ISPM(IMM)
      IDEND1=ISPM(IML)
C***** READ DIVISOR O CONVERT IF NECESSARY *****
      IF (ISEGTA(1).EQ.29) ISOR=ISPM(IMN)
      IF (ISEGTA(1).EQ.93) ISOR=IOPRND(1)
      IF (ISOR.LT.MAXNEG) GO TO 1501
      ISSK=1
      ISOR=ICOMP1(ISOR,ITARG,ITARG)
      ISOR=ISOR+1
C***** CONVERT DIVIDEND IF NECESSARY *****
 1501 IF (IDEND.LT.MAXNEG) GO TO 502
      IDSK=1
```

```
      IDENC1=ICCMP1(ICENC1,ITARG,ITARG)
      ICENC1=ICENC1+1
 1507 ICEND=ICCMP1(ICENC,ITARG,ITARG)
      IF (ICENC1.EQ.0) ICEND=ICEND+1
  502 ISK=ICSK+ISSK
C***** CHECK FCR FIXEC-POINT CIVICE EXCEPTION *****
      IF (ICENC.EC.MAXNEG) GO TO 1510
      IPR=ICENC-ISCR
      IF (IPR.GE.0) GC TC 1510
      ICENC=2*ICENC
      IIPR=ILCAC(ICENC1,ITARG,1)
      IF (IIPR.EQ.1) ICENC=ICENC+1
      IPR=ICENC-ISCR
      IF (IPR.GE.0) GC TO 1510
C***** PERFCRM ITERATIVE CIVISION *****
      K=ITARG-1
      IPR=ICENC
      DC 505 I=1,K
      IPR=2*IPR
      KK=ITARG-I
      IIPR=ILCAC(ICENC1,KK,1)
      IF (IIPR.EC.1) IPR=IPR+1
      IF (IPR-ISCR) 505,506,506
  506 IPR=IPR-ISCR
      IQUO=ICOMP1(IQUC,KK,1)
  505 CONTINUE
C***** CONVERT PRODUCT ANC QUOTIENT IF NECESSARY 0 STORE IN SPM *****
      IF ((ISK.NE.1).CR.(IQUO.EQ.0)) GO TO 507
      IQUO=ICCMP1(IQUC,ITARG,ITARG)
      IQUC=IQUO+1
  507 ISPM(IML)=IQUC
      IF ((ICSK.EQ.0).OR.(IPR.EQ.0)) GO TO 508
      IPR=ICOMP1(IPR,ITARG,ITARG)
      IPR=IPR+1
  508 ISPM(IMM)=IPR
      GO TO 300
C***** CALL INTERRUPT FOR FIXEC-POINT DIVIDE EXCEPTION *****
 1510 CALL PUSH(2,9,0,$300)
      GO TO 300
C
C     30     ALR    IE
30    L=ISPM(IMM)+ISPM(IMN)
      ICATAS=ISPM(IMN)
      GO TO 3112
C
C
C
C     31      SLR    IF
31    K=ISPM(IMN)
      L=ISPM(IMM)+INTNCT(K)+1
      ICATAS=INTNCT(K)+1
3112  M=IOVRFL(L,J)
      ISPM(IMM)=IANC(MAXBIT,L)
      GO TO (3107,3108),J
3107  IF(ISPM(IMM).EC.0) GO TC 3111
3109  ICC=3
      GO  TC 300
3108  IF(ISPM(IMM).NE.0) GO TC 3110
      ICC=0
      GO TO 300
3111  ICC=2
```

```
C     65    LA     41
65    ISPM(IMM)=0
      ISPM(IMM)=ILCAD(IACR(1),24,24)
      GO TC 300
C     66    STC    42
66    K=IACR(1)
      IDATAS=ISPM(IMM)
      CALL STCRCH(K,$301)
      GO TC 300
C     67    IC     43
67    K=IACR(1)
      CALL FECHAR(K,$301)
      K=ISTCRE(ICATAS,ISPM(IMM),8,8)
      GO TC 300
68    GO TO 3002
C     69    BAL    45
69    ISPM(IMM)=INACCR
      ISPM(IMM)=ISTORE(IPRMSK,ISPM(IMM),LBPS2,4)
      ISPM(IMM)=ISTORE(ICC,ISPM(IMM),LBPS1,2)
3203  INACCR=IAND(IACR(1),KACR)
      IDATAS=INACCR
      CALL BRCHK(J)
      GO TO (301,301),J
C
C
C     70    [illegible]    46
C     M=ISP[illegible]
      ISPM([illegible]SMTWC(ITWTSM(M)-1)
      IF(IS[illegible]).EQ.0) GO TO 300
3200  INACCR=[illegible](IACR(1),KACR)
      IDATAS=INACCR
      CALL BRCHK(J)
      GO TC (301,301),J
C
C
C     71    BC     47
71    M=ISEGTA(2)
      IF(M.GT.15.OR.M.LT.0) GO TO 3002
      IF(M.EQ.15) GO TO 3200
      IF(M.EQ.0) GC TC 300
C     MASK VALUE 15
3202  MASK=ISEGTA(2)
      J=JEKCC(MASK)
      GO TC ( 300 ,3200 ) ,J
C     J=2  MATCH  J=1  NO
C
C
C       48   72   LH
72    K=IACR(1)
      CALL FECHHW(K,$301)
      ISPM(IMM)=IDATAS
      IF(IAND(IBIT15,ICATAS).GT.0) GO TO 3205
      GO TO 300
3205  M=MAXBIT
      K=ISTCRE(M,ISPM(IMM),ITARG,IHW)
      GC TC 300
C     73    CH     49
73    K=IACR(1)
      CALL FECHHW(K,$301)
      L=MAXBIT
      IF(IAND(IBIT15,ICATAS).GT.0) GO TO 3207
      M=ICATAS
```

```
      M=ISPM(IMM)
      K=ISTORE(L,M,ITARG,IHW)
      GO TO 3047
 3207 K=ISTORE(L,IDATAS,ITARG,IHW)
      GO TO 3206
 3208 M=ISPM(IMM)
      GO TO 3047
C     74     AH      4A
 74   K=IADR(1)
      CALL FECHHW(K,$301)
      K=0
      IF(IAND(IDATAS,IBIT15).EQ.0) GO TO 3209
      K=MAXBIT
 3209 K=ISTORE(ILOAD(IDATAS,IHW,IHW),K,IHW,IHW)
      M=ISPM(IMM)
      L=ITWTSM(M)+ITWTSM(K)
 3212 CALL IVERFW(L,J)
      ISPM(IMM)=ISMTWO(L)
      GO TO (3037,3033),J
C     75     SH      4B
 75   K=IADR(1)
      CALL FECHHW (K,$301)
      K=0
      IF(IAND(IDATAS,IBIT15).EQ.0) GO TO 3213
      K=MAXBIT
 3213 K=ISTORE(ILOAD(IDATAS,IHW,IHW),K,IHW,IHW)
      M=ISPM(IMM)
      L=ITWTSM(M)+INTNOT(K)+1
      GO TO 3212
C
C     76     MH      4C
 76   GO TO 3058
C
C     80     ST      50
 78   GO TO 3002
 79   GO TO 3002
 80   IDATAS=ISPM(IMM)
      K=IADR(1)
      CALL STOREW(K,$301)
      GO TO 300
C
C     84     N       54
 84   ISPM(IMM)=IAND(ISPM(IMM),IOPRND(1))
 3310 IF(ISPM(IMM).EQ.0) GO TO 3301
 3300 ICC=1
      GO TO  300
 3301 ICC=0
      GO TO 300
C
C
C     85     CL      55
 85   IF(ISPM(IMM).EQ.IOPRND(1)) GO TO 2901
      IF(ISPM(IMM).LT.IOPRND(1)) GO TO 2900
      GO TO 2902
C
C
C     86     O       56
 86   ISPM(IMM)=IOR(ISPM(IMM),IOPRND(1))
      GO TO 3310
C
C
```

```
C
C
C      89     C     59
 89    K=ICPRNC(1)
       M=ISPM(IMM)
       GC TC 3047
C
C
C
C      90     A     5A
 90    IDATAS=ICPRNC(1)
       K=ISPM(IMM)
       L=ITWTSM(K)+ITWTSM(ICATAS)
       CALL IVERFL(L,J)
       ISPM(IMM)=ISMTWC(L)
 3222  GC TC (3037,3033),J
C
C
C      91     S     5B
 91    ICATAS=ICPRNC(1)
       K=ISPM(IMM)
       L=ITWTSM(K)-ITWTSM(ICATAS)
       IDATAS=INTNCT(ICATAS)+1
       CALL IVERFL(L,J)
       ISPM(IMM)=ISMTWO(L)
       GO TO 3222
C
C
C
C      92     M     5C
 92    GO TO 28
C
C
C      93     C     5C
 93    GO TO 29
C
C
C
C      94     ALR     5E
 94    L=ISPM(IMM)+ICPRNC(1)
       IDATAS=ICPRNC(1)
       GC TC 3112
C      95     SLR     5F
 95    K=IDPRNC(1)
       L=ISPM(IMM)+INTNOT(K)+1
       ICATAS=INTNOT(K)+1
       GC TO 3112
C      128  SSM     80
 128   K=IACR(1)
       IF(IAND(IAWP,IBIT).GT.0) GO TO 3263
       CALL FECHAR(K,$301)
       ISYSMK=ICATAS
       GC TC 300
C      130     LPSW  82
 130   K=IACR(1)
       IF(IAND(K,7).NE.0) GO TC 3264
C      CHECK PROBLEM STATE
       IF(IAND(IAWP,IBIT).GT.0) GO TO 3263
       J=ISPPSW(K)
C      CHECK I/C CHANNEL  TBC
C      CHECK WAIT STATE
       IF(IAND(IAWP,2).EQ.0) GO TC 300
```

```
      GC TC 300
1001  FORMAT(16H      WAIT STATE)
3264  IERFLG=5
      GC TO 2998
3263  IERFLG=2
      GC TO 2998
131   GO TC 3002
C
C     132   WRC    84
132   GC TC 3002
C     NCT  CEFINED
C
C     133  RCD     85
133   GO TC 3002
C     NCT  CEFINED
C     134   BXH    86
134   M=ISPM(IMM)
      N=ISPM(IMN)
      IRSUM=ITWTSM(M)+ITWTSM(N)
      IML=IMN
      IMLT=(ISEGTA(3)/2)*2
      IF(IMLT.EC.ISEGTA(3)) IML=IML+1
      IRC=ITWTSM(ISPM(IML))
      IF(IMM.EC.IML) IRSUM=IRSUM-ITWTSM(N)
      IF(IRSUM.GT.IRC) INACCR=IACR(1)
      ISPM(IMM)=ISMTWC(IRSUM)
      IF(IMM.EC.IML) ISPM(IMM)=ISMTWO(ITWTSM(M)+ITWTSM(N))
      GO TC 301
C     135   BXLE   87
135   M=ISPM(IMM)
      N=ISPM(IMN)
      IRSUM=ITWTSM(M)+ITWTSM(N)
      IML=IMN
      IMLT=(ISEGTA(3)/2)*2
      IF(IMLT.EC.ISEGTA(3)) IML=IML+1
      IRC=ITWTSM(ISPM(IML))
      IF(IMM.EC.IML) IRSUM=IRSUM-ITWTSM(N)
      IF(IRSUM.LE.IRC) INACCR=IACR(1)
      ISPM(IMM)=ISMTWC(IRSUM)
      IF(IMM.EC.IML) ISPM(IMM)=ISMTWO(ITWTSM(M)+ITWTSM(N))
      GO TC 301
C     136   SRL    88
136   I=ISHACR(K)
      IF(I.GT.31) GO TC 3231
      K=ITARG-I
      M=NULL
      M=ILCAC(ISPM(IMM),ITARG,K)
      ISPM(IMM)=M
      GO TC 300
C
C     137   SLL    89
137   I=ISHACR(K)
      IF(I.GT.31) GC TC 3231
      L=NULL
      M=ITARG-I
      ISPM(IMM)=ISTCRE(ISPM(IMM),L,ITARG,M)
      GO  TC  300
3231  ISPM(IMM)=NULL
      GO TC  300
C
C     138   SRA    8A
138   I=ISHACR(K)
```

```
      IF(ISPM(IMM).EQ.0) GO TO 3017
      ICC=2
      GO TO 300
3229  IF(IAND(ISPM(IMM),MAXNEG).GT.0) GO TO 3230
      ISPM(IMM)=NULL
      GO TO 3017
3230  ISPM(IMM)=MAXBIT
      GO TO 3016
C     139     SLA       88
139   I=ISHADR(K)
      IF(I.GE.31) GO TO 3232
      L=0
      M=ITARG-I-1
      K=ITARG-1
      J=0
      J=ISTORE(ISPM(IMM),L,K,M)
      N=NULL
      M=0
      IF(IAND(ISPM(IMM),MAXNEG).GT.0) GO TO 3228
3224  M=ISTORE(ILOAD(ISPM(IMM),K,I),M,K,I)
3234  ISPM(IMM)=J
3261  IF(IER(M,N).EQ.0) GO TO 3225
C     O/F
      ICC=3
      CALL PUSH(2,8,0,$300)
      GO TO 300
3228  M=MAXBIT
      N=M
      J=IOR(J,MAXNEG)
      GO TO 3224
3225  IF(ISPM(IMM).EQ.0) GO TO 3017
C     NO O/F
      IF(IAND(ISPM(IMM),MAXNEG).GT.0) GO TO 3016
      ICC=2
      GO TO 300
3232  M=ISPM(IMM)
      IF(IAND(MAXNEG,ISPM(IMM)).GT.0) GO TO 3233
      ISPM(IMM)=NULL
      N=NULL
      GO TO 3261
3233  ISPM(IMM)=MAXNEG
      N=MAXBIT
      GO TO 3261
C
C
C     SRDL      140
140   I=ISHADR(K)
      MM=NULL
      IF(I.GT.31) GO TO 3060
      M=ITARG-I
      MM=ILOAD(ISPM(IMM+1),ITARG,M)
      ISPM(IMM+1)=ISTORE(MM,ISPM(IMM+1),I,ITARG)
      MM=0
      ISPM(IMM)=ILOAD(ISPM(IMM),ITARG,M)
      GO TO 300
3060  LL=64-I
      ISPM(IMM+1)=ISTORE(ISPM(IMM),MM,LL,LL)
      ISPM(IMM)=NULL
      GO TO 300
C     SLDL      141
141   I=ISHADR(K)
      MM=NULL
```

```
      M=ITARG-I
      ISPM(IMM)=ISTORE(ISPM(IMM),MM,ITARG,M)
      ISPM(IMM)=ILOAD(ISPM(IMM+1),ITARG,I)
      MM=NULL
      ISPM(IMM+1)=ISTORE(ISPM(IMM+1),MM,ITARG,M)
      GO TO 300
3061  LL=64-I
      ISPM(IMM)=ISTORE(ISPM(IMM+1),MM,ITARG,LL)
      ISPM(IMM+1)=NULL
      GO TO 300
C     SRDA    142
142   I=ISHADR(K)
      IF(IAND(ISPM(IMM),MAXNEG).GT.NULL) GO TO 3062
      MN=NULL
3063  IF(I.EQ.NULL) GO TO 3064
      MM=MN
      K=ITARG-I-1
      M=ITARG-1
      IA=ITARG-I
      IF(I.GT.31) GO TO 3065
      MM=ILOAD(ISPM(IMM+1),ITARG,IA)
      ISPM(IMM+1)=ISTORE(MM,ISPM(IMM+1),ITARG,I)
      MM=MN
      MM=ILOAD(ISPM(IMM),M,K)
      ISPM(IMM)=MM
      IF(IAND(ISPM(IMM),MAXNEG).GT.NULL) GO TO3062
3064  IF(ISPM(IMM).EQ.NULL.AND.ISPM(IMM+1).EQ.NULL) ICC=0
      IF(IAND(ISPM(IMM),MAXNEG).EQ.NULL) ICC=2
      GO TO 300
3062  MN=MAXBIT
      ICC=1
      GO TO 3063
3065  LL=63-I
      ISPM(IMM+1)=MN
      ISPM(IMM+1)=ILOAD(ISPM(IMM),M,LL)
      ISPM(IMM)=MN
      GO TO 3064
C     SLDA    143
143   I=ISHADR(K)
      IF(IAND(ISPM(IMM),MAXNEG).GT.NULL) GO TO 3066
      MN=NULL
3070  IF(I.EQ.NULL) GO TO 3067
      MM=MN
      TSTORG(1)=MN
      TSTORG(2)=MN
      IF(I.GT.31) GO TO 3068
      K=ITARG-I-1
      M=ITARG-1
      IA=ITARG-I
      TSTORG(1)=ILOAD(ISPM(IMM),M,I)
      ISPM(IMM)=ISTORE(ISPM(IMM),MM,M,K)
      ISPM(IMM)=ILOAD(ISPM(IMM+1),ITARG,I)
      MM=MN
      ISPM(IMM+1)=ISTORE(ISPM(IMM+1),MM,ITARG,IA)
      IF(IAND(ISPM(IMM),MAXNEG).GT.NULL) GO TO 3069
3071  IF(IER(TSTORG(1),MN).EQ.NULL) GO TO 3067
C     C/F
      ICC=3
      CALL PUSH(2,8,0,$300)
      GO TO 300
3069  MN=MAXNEG-1
```

```
      ISPM(IMM)=ISTCRE(ISPM(IMM+1),MM,M,LL)
      ISPM(IMM+1)=NULL
      GO TO 3071
 3066 MN=MAXBIT
      ICC=1
      GO TO 3070
C     144     STM     90
  144 K=IADR(1)
      IF(IMM.GT.IMN) GO TO 3075
      DO 3059 J=IMM,IMN
      IDATAS=ISPM(J)
      CALL STORFW(K,$301)
 3059 K=K+4
      GO TO 300
 3075 DO 3073 J=IMM,16
      IDATAS=ISPM(J)
      CALL STORFW(K,$301)
 3073 K=K+4
      L=IMM-1
      DO 3074 J=1,L
      IDATAS=ISPM(J)
      CALL STORFW(K,$301)
 3074 K=K+4
      GO TO 300
C     145     TM    91
  145 K=IADR(1)
      CALL FECHAR(K,$301)
      IF(ISEGTA(14))3002,3049,3050
 3050 L=ISEGTA(14)
      M=IER(L,IDATAS)
      IF (M) 3002,3051,3052
 3049 ICC=0
      GO TO 300
 3051 ICC=3
      GO TO 300
 3052 ICC=1
      GO TO 300
C     146     MVI     92
  146 K=IADR(1)
      IDATAS=ISEGTA(14)
      CALL STORCH(K,$301)
      GO TO 300
C     147      TS     93
  147 K=IADR(1)
      CALL FECHAR(K,$300)
      ICC=IAND(IDATAS,128)
      IDATAS=255
      CALL STORCH(K,$300)
      GO TO 300
C     148     NI    94
  148 L=ISEGTA(14)
      K=IADR(1)
      CALL FECHAR(K,$301)
      IDATAS=IAND(L,IDATAS)
      CALL STORCH(K,$301)
      IF(IDATAS)3025,3026,3025
C     149     CLI     95
  149 L=ISEGTA(14)
      K=IADR(1)
      CALL FECHAR(K,$301)
      M=IDATAS
      GO TO 3047
```

```
150   L=ISEGTA(14)
      K=IADR(1)
      CALL FECHAR(K,$301)
      IDATAS=ICR(L,IDATAS)
      CALL STCRCH(K,$301)
      IF(IDATAS)3025,3026,3025
C     151   XI   97
151   L=ISEGTA(14)
      K=IADR(1)
      CALL FECHAR(K,$301)
      IDATAS= IER(L,IDATAS)
      CALL STCRCH(K,$301)
      IF(IDATAS)3025,3026,3025
C     152   LM   98
152   K=IADR(1)
      IF(IMM.GT.IMN) GC TC 3053
3054  DC 3055 J=IMM,IMN
      CALL FECHFW(K,$301)
      ISPM(J)=IDATAS
3055  K=K+4
      GC TC 300
3053  CO 3056 J=IMM,16
      CALL FECHFW(K,$301)
      ISPM(J)=IDATAS
3056  K=K+4
      L=IMM-1
      CO 3057 J=1,L
      CALL FECHFW(K,$301)
      ISPM(J)=IDATAS
3057  K=K+4
      GC TC 300
C
C     156   SIC        9C
156   GO TC 3002
157   GO TO 3002
158   GO TO 3002
159   GO TC 3002
C       SIC  NCT CEFINEC
C       HIC NCT CEFINEC
C     158   HIC       9E
209   GO TC 3002
C     210   MVC     C2
210   K=IADR(1)
      L=IADR(2)
      M=ISEGTA(14)*16+ISEGTA(15)
      IF(M.LT.0.CR.M.GT.255) GC TO 3002
      DC 3400 I=1,M
      CALL FECHAR(L,$301)
      CALL STCRCH (K,$301)
      K=K+1
3400  L=L+1
      GC TC 300
211   GC TC 3002
212   N=1
      N=ISLCGE(N)
      GC TC 300
213   GC TC 3002
214   N=2
      N=ISLCGE(N)
      GC TC 300
215   N=3
      N=ISLCGE(N)
```

```
      EQUIVALENCE (ISPM(25),INADDR)
      LOGICAL CSLAE,IRESTA
      COMMON /TES/ LEVELS(4),IOUTPU(5),IPOINT(5)
      COMMON/UNCON/IFWA,IHOST,ITARG,IOPCW,IBIT,IPARS,IDATAS,IHW,MAXCOR,
     1 LOCG,MPOS,TIMBO,NRCODE,NSTAT,LBPS1,LBPS2,NULL,IONE,IT,
     1OSLAE,ILC,INTCDE,ICOUNT,EXTIME,TIMLIM,
     1IMM,IMN,IERFLG,IRESTA,
     1LOBOUN,JIBOUN,MONE,IFETC
     1,MAXNEG,MAXBIT
       COMMON /UARRAY/ ISEGTA(20),TIME(10),CLASS(10),INSDAT(256,5),
     1INSPAR(5,20),IPCTR(5),IPPCTR(5),ISGCD(5,20),IADR(3),ISPM(64),
     1IHALFW(6),ISTACK(5,6),IMAINM(4096)
      COMMON /CVAR/ICPCO(255),ITCNTR,IOFFST,IOPRND(3),ISTAT
      IF(IDATAS.GT.MAXCOR.OR.IDATAS.LT.0) GO TO 20
   10 J=1
      RETURN
   20 CALL PUSH(2,5,0,$301)
  301 J=2
      IERFLG=14
      RETURN
      END
$IBFTC CCR007  DECK
      SUBROUTINE STORFW(ICADR,*)
      COMMON /TES/ LEVELS(4),IOUTPU(5),IPOINT(5)
      COMMON/UNCON/IFWA,IHOST,ITARG,IOPCW,IBIT,IPARS,IDATAS,IHW,MAXCOR,
     1 LOCG,MPOS,TIMBO,NRCODE,NSTAT,LBPS1,LBPS2,NULL,IONE,IT,
     1OSLAE,ILC,INTCDE,ICOUNT,EXTIME,TIMLIM,
     1IMM,IMN,IERFLG,IRESTA,
     1LOBOUN,JIBOUN,MONE,IFETC
     1,MAXNEG,MAXBIT
       COMMON /UARRAY/ ISEGTA(20),TIME(10),CLASS(10),INSDAT(256,5),
     1INSPAR(5,20),IPCTR(5),IPPCTR(5),ISGCD(5,20),IADR(3),ISPM(64),
     1IHALFW(6),ISTACK(5,6),IMAINM(4096)
      COMMON /CVAR/ICPCO(255),ITCNTR,IOFFST,IOPRND(3),ISTAT
      LOGICAL IFETC,OSLAE,GITMO
      GITMO=.FALSE.
      K=IDADR/2
      I=MOD(K,2)
      K=ICADR/4+1
      I=I+1
      GO TO (1000,1001),I
C     1000 E, 1001 O
 1000 IMAINM(K)=ILCAD(ICATAS,ITARG,ITARG)
 1040 IF(GITMO)  GO TO 1041
      RETURN
 1041 GITMO=.FALSE.
      K=K+1
      GO TO 1020
 1001 GITMO=.TRUE.
      I=ITARG
      L=IHW
 1002 II=IAND(IMAINM(K),MAXBIT)
      IMAINM(K)=ISTORE(ILOAD(ICATAS,I,IHW),II,L,IHW)
      GO TO 1040
      ENTRY STORHW(ICADR,*)
      GITMO=.FALSE.
      K=ICADR/2
      I=MOD(K,2)
      I=I+1
      K=IDACR/4+1
      GO TO (1020,1003),I
C     1020 E, 1002 O
```

```
      L=ITARG
      GC TO 1002
      NTRY STCRCH(ICACR,*)
      =ICACR/4+1
      N=ITARG-IANC(ICACR,3)*8
      IDATAS=ILCAC(IMAINM(K),N,8)
      RETURN
1003  I=IHW
      L=IHW
      GC TO 1002
      END
$IBFTC CCR008  DECK
      SUBRCUTINE INITLZ
      LCGICAL TESTC1
      INTEGER IBUF(80)
       CCMMCN/BUF1/ IBUF
      CATA M1/C716060606060/
C     7094  CEP
      INTEGER  EXCPSW(4),SVCOPS(4),PROPSW(4),MACPSW(4),IOOPSW(4),
     1CSWORC(4),CAWORC(4),EXNPSW(4),SVCNPS(4),PRNPSW(4),MANPSW(4),
     1 EXCLCC,SVCLCC,PRCLCC,EXNLCC,SVNLOC,PRNLOC,
     1IONPSW(4),CURPSW(4),GENREG(16),         TSTORG(10)
      DIMENSICN FLPTRG(8)
      EQUIVALENCE(IMAINM(1),INIPSW),(IMAINM(3),INCCW1),(IMAINM( 5),
     1 INCCW2),(IMAINM( 7),EXCPSW),(IMAINM( 9),SVCOPS),(IMAINM(11),
     1 PROPSW),(IMAINM(13),MACPSW),(IMAINM(15),IOOPSW),(IMAINM(17),
     1 CSWORC),(IMAINM(19),CAWORC),(IMAINM(23),EXNPSW),(IMAINM(25),
     1 SVCNPS),(IMAINM( 27),PRNPSW),(IMAINM( 29),MANPSW),(IMAINM( 31),
     1 ICNPSW),(ISPM(25),INACCR),(ISPM(25),CURPSW),(ISPM,GENREG),
     1 (ISPM(17),FLPTRG),(ISPM(33),TSTORG),(IMAINM(1),INPRCT),
     1 (IMAINM( 8),EXCLOC),(IMAINM(10),SVOLOC),(IMAINM(12),PROLOC),
     1 (IMAINM(14),MACLOC),(IMAINM(15),IOOLOC),(IMAINM(24),EXNLOC),
     1 (IMAINM(26),SVNLCC),(IMAINM( 28),PRNLOC),(IMAINM( 30),MANLOC),
     1 (IMAINM( 32),ICNLCC)
      ECUIVALENCE (ISPM(25),INADDR)
      LOGICAL CSLAE,IRESTA
      COMMCN /TES/ LEVELS(4),IOUTPU(5),IPOINT(5)
      COMMCN/UNCON/IFWA,IHOST,ITARG,IOPCW,IBIT,IPARS,IDATAS,IHW,MAXCOR,
     1 LCCG,MPCS,TIMBC,NRCODE,NSTAT,LBPS1,LBPS2,NULL,IONE,IT,
     1 CSLAE,ILC,INTCDE,ICOUNT,EXTIME,TIMLIM,
     1IMM,IMN,IERFLG,IRESTA,
     1LCBOUN,JIBCUN,MCNE,IFETC
     1,MAXNEG,MAXBIT
       COMMON /UARRAY/ ISEGTA(20),TIME(10),CLASS(10),INSDAT(256,5),
     1INSPAR(5,20),IPCTR(5),IPPCTR(5),ISGCD(5,20),IADR(3),ISPM(64),
     1IHALFW(6),ISTACK(5,6),IMAINM(4096)
      CCMMCN /CVAR/ICPCO(255),ITCNTR,IOFFST,IOPRND(3),ISTAT
      LOGICAL TRACE
      INTEGER CHALL8
      CATA CHALL8/4294967295/
      CCMMCN/TR/ TRACE,FTRACE,TITRAC,TRTI,TSTRT,TSTPT,KTRACE
      ICOUNT=0
C     GENERAL REGISTER CFFSET
      LCCG=1
      IERFLG=0
      MAXCCR=32768
      IRESTA=.FALSE.
1001  FORMAT(4I10,F16.3)
      EXTIME=0.
      TIMBC=0.
      MPCS=IHCST-1
```

```
      CALL MMCRC
      IOFFST=0
      READ 1001 ,  LOBOUN,JIBOUN,IFWA,INADDR,TIMLIM
      NRIN=0
      TEST 01= .FALSE.
600   READ(10,610) (IBUF(I),I=1,72)
      NRIN=NRIN+1
      IDATAS=0
       DO  601  I= 1,6
      IF(IBUF(I).EQ.M1) TEST01=.TRUE.
601    CONTINUE
      IF( TEST01)  GO TO  602
      K=JEXBIN(IBUF,1,6)+IFWA
      I= JEXBIN(IBUF,20,1)
      IDATAS=JEXBIN(IBUF,10,9)
      GO TO ( 603,604 ), I
603   CALL STORHW(K,$605)
      GO TO 600
604   CALL STORFW(K,$605)
      GO TO 600
602   CONTINUE
      WRITE(6,1001) LOBOUN,JIBOUN,IFWA,INADDR,TIMLIM
      K=IFWA/4+1
      L=K+NRIN
      WRITE(6,1003) (I,IMAINM(I),I=K,L)
1003   FORMAT(I15,O20)
      RETURN
605   IERFLG=4
      RETURN
610   FORMAT(80A1)
      END
$IBFTC CCR009  DECK
      SUBROUTINE  ERINS
      EQUIVALENCE (ISPM(25),INADDR)
      LOGICAL CSLAE,IRESTA
       LOGICAL IFETC
      COMMON /TES/ LEVELS(4),IOUTPU(5),IPOINT(5)
      COMMON/UNCON/IFWA,IHOST,ITARG,IOPCW,IBIT,IPARS,IDATAS,IHW,MAXCOR,
     1 LCCG,MPCS,TIMBC,NRCODE,NSTAT,LBPS1,LBPS2,NULL,IONE,IT,
     1OSLAE,ILC,INTCDE,ICOUNT,EXTIME,TIMLIM,
     1IMM,IMN,IERFLG,IRESTA,
     1LOBOUN,JIBOUN,MONE,IFETC
     1,MAXNEG,MAXBIT
       COMMON /UARRAY/ ISEGTA(20),TIME(10),CLASS(10),INSDAT(256,5),
     1INSPAR(5,20),IPCTR(5),IPPCTR(5),ISGCD(5,20),IADR(3),ISPM(64),
     1IHALFW(6),ISTACK(5,6),IMAINM(4096)
      COMMON /CVAR/ICPCO(255),ITCNTR,IOFFST,IOPRND(3),ISTAT
      DIMENSION  INTCOD(13)
C     IERFLG  1  OPERATION
C             1  PRIVILEGED
C              3  EXECUTE
C              4  ADDRESS
C            5  SPECIFICATION
C              6  DATA
C            FIX PT O/F
C           FIX PT DIV
C            9  EXPON O/F
C           10  EXPN U/F
C           11 SIGNIFICANCE
C           12  FL PT CHK
C          13  MACHINE CHK
```

```
C       15   TIME C/F
C            16   WAIT STATE
      WRITE(6,1)INACCR,IADR(1),IOPRND(1),IERFLG
    1 FORMAT(21H      BAD INSTRUCTION,3017,I10)
      GO TO (10,10,10,10,10,10,10,10,10,10,10,10,10,100,100,100,100,
     1100,100,100), IERFLG
   10 I=INTCOD(IERFLG)
      CALL PUSH (2,I,0,$300)
  300 RETURN
  100 CONTINUE
      RETURN
      DATA INTCOD/1,2,3,5,6,7,8,9,12,13,14,15,0/
      END
$IBFTC CCR010  DECK
      SUBROUTINE HALTE
      EQUIVALENCE (ISPM(25),INADDR)
      LOGICAL CSLAE,IRESTA
      COMMON /TES/ LEVELS(4),IOUTPU(5),IPOINT(5)
      COMMON/UNCON/IFWA,IHOST,ITARG,IOPCW.IBIT,IPARS,IDATAS,IHW,MAXCOR,
     1 LOCG,MPCS,TIMBO,NRCODE,NSTAT,LBPS1,LBPS2,NULL,IONE,IT,
     1CSLAE,ILC,INTCDE,ICOUNT,EXTIME,TIMLIM,
     1IMM,IMN,IERFLG,IRESTA,
     1LOBCUN,JIBCUN,MONE,IFETC
     1,MAXNEG,MAXBIT
       COMMON /UARRAY/ ISEGTA(20),TIME(10),CLASS(10),INSDAT(256,5),
     1INSPAR(5,20),IPCTR(5),IPPCTR(5),ISGCD(5,20),IADR(3),ISPM(64),
     1IHALFW(6),ISTACK(5,6),IMAINM(4096)
      COMMON/OVAR/ ICPCO(255),ITCNTR,IOFFST,IOPRND(3)
      K=IPCTR(IT)
      WRITE(6,1) IMAINM(K),K,INACCR
      RETURN
    1 FORMAT(1H0,10H      HALT,3I15)
      END
$IBFTC CCR011  DECK
      SUBROUTINE FECHEW(ICADR,*)
      EQUIVALENCE (ISPM(25),INADDR)
      COMMON /TES/ LEVELS(4),IOUTPU(5),IPOINT(5)
      COMMON/UNCON/IFWA,IHOST,ITARG,IOPCW,IBIT-IPARS,IDATAS,IHW,MAXCOR,
     1 LOCG,MPCS,TIMBO,NRCODE,NSTAT,LBPS1,LBPS2,NULL,IONE,IT,
     1CSLAE,ILC,INTCDE,ICOUNT,EXTIME,TIMLIM,
     1IMM,IMN,IERFLG,IRESTA,
     1LOBCUN,JIBCUN,MONE,IFETC
     1,MAXNEG,MAXBIT
      COMMON /UARRAY/ISEGTA(20),TIME(10),CLASS(10),INSDAT(256,5),
     1INSPAR(5,20),IPCTR(5),IPPCTR(5),ISGCD(5,20),IADR(3),ISPM(64),
     1IHALFW(6),ISTACK(5,6),IMAINM(4096)
      COMMON /OVAR/ICPCO(255),ITCNTR,IOFFST,IOPRND(3),ISTAT
      LOGICAL GITMO
      IF(ISEGTA(1).GE.136.AND.ISEGTA(1).LE.143) RETURN
      MS24BR=2**25-1
C    EQU KADR
      IDATAS=IAND(ICADR,MS24BR)
      CALL BRCHK(J)
C     HOST DEPENDENT
      GO TO (1003,1004),J
 1003 GITMO=.FALSE.
      K=ICADR/2
      I=MOD(K,2)
      ICADR=ICADR/4+1
      I= I+1
      IF(I.GT.2.OR.I.LT.1) GO TO 1005
```

```
 1041   ITMO = .FALSE.
        MM=ITARG
        I=IHW
        ICADR=ICADR+1
        GO TO 1021
 1001   GITMO= .TRUE.
        MM=IHW
        I=ITARG
 1021   IDATAS=ISTORE(ILOAD(IMAINM(ICADR),MM,IHW),IDATAS,I,IHW)
        GO TO 1040
        ENTRY FECHHW(ICADR,*)
        GITMO = .FALSE.
        K=ICADR/2
        L=MOD(K,2)
        ICADR=ICADR/4+1
        L=L+1
        GO TO (1020,1022), L
C       1020 ,E   ,1002 C
 1020   I=ITARG
 1023   IDATAS=ILOAD(IMAINM(ICADR),I,IHW)
        RETURN
 1022   I=IHW
        GO TO [illegible]23
        ENTRY [illegible]AR(ICADR,*)
        K=ICA[illegible]
        N=ITA[illegible]NC(ICADR,3))*8
        IDATA[illegible]D(IM[illegible]NM(K),N,8)
        RETURN
 1004   J=2
        RETURN
        END
$IBFTC CCR012  DECK
        FUNCTION JEKCC(MASK)
C      TARGET DEP
C   JEKCC  2, MATCH     JEKCC  1  NO MATCH
        COMMON/UNCON/ IFWA,IHCST,ITARG,IOPCW,IBIT,IPARS,IDATAS,IHW,MAXCOR,
       1LOCG,MPOS,TIMBG,NRCODE,NSTAT,LBPS1,LBPS2,NULL,IONE,IT,
       1OSLAE,ILC,INTCDE,ICOUNT,EXTIME,TIMLIM,
       1IMM,IMN,TERFLG,IRESTA,
       1LCBCUN,JIBCUN,MONE,IFETC
       1,MAXNEG,MAXBIT
        COMMON/UARRAY/ ISEGTA(20),TIME(10),CLASS(10),INSDAT(256,5),
       1INSPAR(5,20),IPCTR(5),IPPCTR(5),ISGCD(5,20),IADR(3),ISPM(64),
       1IHALFW(6),ISTACK(5,6),IMAINM(4096)
        EQUIVALENCE (ICC,ISPM(28))
         ICODE=ICC
        ICODE=ICODE+1
        GO TO (100,101,102,103),ICODE
  100   ILL=8
        GO TO 106
  101   ILL=4
        GO TO 106
  102   ILL=2
        GO TO 106
  103   ILL=1
  106   JEKCC=IAND(MASK,ILL)
        IF(JEKCC.NE.0) GO TO 310
  300    JEKCC=1
        RETURN
  310    JEKCC=2
        RETURN
        END
```

```
      SUBROUTINE TERMIN
      LOGICAL CSLAE,IRESTA
      COMMON /TES/ LEVELS(4),IOUTPU(5),IPOINT(5)
      COMMON/UNCON/IFWA,IHOST,ITARG,IOPCW,IBIT,IPARS,IDATAS,IHW,MAXCOR,
     1 LOCG,MPCS,TIMBC,NRCODE,NSTAT,LBPS1,LBPS2,NULL,IONE,IT,
     1OSLAE,ILC,INTCDE,ICOUNT,EXTIME,TIMLIM,
     1IMM,IMN,IERFLG,IRESTA,
     1LOBOUN,JIBOUN,MONE,IFETC
     1,MAXNEG,MAXBIT
       COMMON /UARRAY/ ISEGTA(20),TIME(10),CLASS(10),INSDAT(256,5),
     1INSPAR(5,20),IPCTR(5),IPPCTR(5),ISGCD(5,20),IADR(3),ISPM(64),
     1IHALFW(6),ISTACK(5,6),IMAINM(4096)
      COMMON /CVAR/ICPCO(255),ITCNTR,IOFFST,IOPRND(3),ISTAT
      IF(IRESTA) GO TO 10
   20 WRITE(6,1) IERFLG,ISPM(25)
    1 FORMAT(10H       DONE,I10,O20)
      CALL STATDP
      RETURN
   10 CALL RSTART
      GO TO 20
      END
$IBFTC CCR014  DECK
      SUBROUTINE TIMER
      COMMON /TES/ LEVELS(4),IOUTPU(5),IPOINT(5)
      COMMON/UNCON/IFWA,IHOST,ITARG,IOPCW,IBIT,IPARS,IDATAS,IHW,MAXCOR,
     1 LOCG,MPCS,TIMBO,NRCODE,NSTAT,LBPS1,LBPS2,NULL,IONE,IT,
     1 OSLAE,ILC,     INTCDE,ICOUNT,
     1EXTIME,TIMLIM,
     1IMM,IMN,IERFLG,IRESTA,
     1LOBOUN,JIBOUN,MONE,IFETC
     1,MAXNEG,MAXBIT
       COMMON /UARRAY/ ISEGTA(20),TIME(10),CLASS(10),INSDAT(256,5),
     1INSPAR(5,20),IPCTR(5),IPPCTR(5),ISGCD(5,20),IADR(3),ISPM(64),
     1IHALFW(6),ISTACK(5,6),IMAINM(4096)
      COMMON /CVAR/IOPCO(255),ITCNTR,IOFFST,IOPRND(3),ISTAT
      TIME(ISTAT)=TIME(ISTAT)+10.
      CLASS(ISTAT)=CLASS(ISTAT)+1.
      IOFFST=IOFFST+1
      EXTIME=EXTIME+5.
      TIMBO=EXTIME
      OPERNR=OPERNR+1.
      IF(EXTIME.GT.TIMLIM) GO TO 10
   20 RETURN
   10 IERFLG=15
      RETURN
      END
$IBFTC CCR015  DECK
      SUBROUTINE INTRPS
C     TARGET DEP
      INTEGER C16,C10,C15
      DATA IPMASK,I4,O10,C15,O16/15,4,8,13,14/
      COMMON /TES/ LEVELS(4),IOUTPU(5),IPOINT(5)
C     I/O L4,EXTERNAL L3,PROGRAM L2,MACHINE L1
      INTEGER  EXOPSW(4),SVCOPS(4),PROPSW(4),MAOPSW(4),IOOPSW(4),
     1CSWORD(4),CAWORD(4),EXNPSW(4),SVCNPS(4),PRNPSW(4),MANPSW(4),
     1 EXOLOC,SVOLOC,PROLOC,EXNLOC,SVNLOC,PRNLOC,
     1IONPSW(4),CURPSW(4),GENREG(16),        TSTORG(10)
      DIMENSION FLPTRG(8)
      EQUIVALENCE(IMAINM(1),INIPSW),(IMAINM(3),INCCW1),(IMAINM( 5),
     1 INCCW2),(IMAINM( 7),EXOPSW),(IMAINM( 9),SVCOPS),(IMAINM(11),
     1 PROPSW),(IMAINM(13),MACPSW),(IMAINM(15),ICOPSW),(IMAINM(17),
```

```
      ICSLAE,ILC,INTCDE,ICCUNT,EXTIME,TIMLIM,
      1IMM,IMN,IERFLG,IRESTA,
      1LCHCUN,JIBCUN,MCNE,IFETC
      1,MAXNEG,MAXBIT
       COMMCN /UARRAY/ ISEGTA(20),TIME(10),CLASS(10),INSDAT(256,5),
      1INSPAR(5,20),IPCTR(5),IPPCTR(5),ISGCC(5,20),IADR(3),ISPM(64),
      1IHALFW(6),ISTACK(5,6),IMAINM(4096)
       COMMON /CVAR/IOPCO(255),ITCNTR,IOFFST,IOPRND(3),ISTAT
       EQUIVALENCE (ISYSMK,ISPM(26)),(IPRMSK,ISPM(27)),(IAWP,ISPM(29)),
      1(ICC,ISPM(28))
       ENTRY PUSH(IPL,INTCC,ICSW,*)
       INPUSH=ICUTPU(IPL)
27     IF(INPUSH.EC.IPOINT(IPL)) GO TO 29
23     GO TO (25,25,25,25,26),INPUSH
26     INPUSH=0
25     INPUSH=INPUSH+1
28     IF(ISTACK(INPUSH,IPL).NE.INTCO) GO TO 27
24     RETURN
29     IPOI=IPOINT(IPL)
       GO TO (40,40,40,40,61),IPOI
61     IPOINT(IPL)=0
40     IPOINT(IPL)=IPOINT(IPL)+1
       INPUSH=IPOINT(IPL)
       ISTACK(INPUSH,IPL)=INTCO
       ICCUNT=ICCUNT+1
       GO TO(30,30,10,20),IPL
20     ISTACK(INPUSH,IPL+1)=ICSW
10     RETURN
30     RETURN 1
       ENTRY PULL (IPR,J,IPRSTA,ICSWO )
       IF(ICUTPU(IPR).EQ.IPOINT(IPR)) GO TO 160
       IF(ICUTPU(IPR).EC.5) IOUTPU(IPR)=0
       IOUTPU(IPR)=IOUTPU(IPR)+1
       IOUT=IOUTPU(IPR)
       ICCUNT=ICCUNT-1
170    GO TO(120,130,140,150),IPR
C      120 MACHINE CHECK BIT 1 ISPM(29)
C      MACHINE DEPENDENT
120    IF(IAND(IAWP,I4).NE.0) GO TO 122
160    J=1
       RETURN
C      ANY OF THE MASKED CAUSES
C       130 PRORAM EXCEPTION GET INTERRTION CDE
C      SVC  MACHINE DEPENDENT
130    IF(ISEGTA(1).EC.10) GC TO 63
131    LIK=ISTACK(ICUT,IPR)
C      FIX PT C/F
       I=IAND(LIK,IPRMSK)
       IF(IER(LIK,8).EC.0.ANC.I.EQ.0) GO TO 160
C      EXPONENT O/F
       IF(IER(LIK,2).EC.0.ANC.I.EC.0) GO TO 160
       IF(IER(LIK,1).EC.0.ANC.I.EC.0) GO TO 160
C      SIGNIFICANCE
122     IPRSTA=ISTACK(ICUT,IPR)
       GO TO (100,100,100,110),IPR
110    ICSWO=ISTACK(ICUT,IPR+1)
100    J=2
       GO TO 63
C      EXTERNAL
140    LIK=  IAND(ISYSMK,ICNE)
       LIK=LIK+1
```

```
150    CALL HALTC
C      MACHINE CHECK ELIMINATE SVC,PROG INT
53     GO TO(54,55,56,57),IPR
54     IF(ICUTPU(2)-IPOINT(2))58,59,61
58     IJ=ICUTPU(2)
       GO TO (64,64,64,64,62),IJ
C      TERMINATE
62     ICUTPU(2)=2
64     ICUTPU(2)=ICUTPU(2)+1
       ICOUNT=ICOUNT-1
       GO TO 54
63     K=9
       IMAINM(K)=ISTACK(IOUT,IPR)
       J=IMMPSW(32)
       J=ISPPSW(96)
       GO TO 1000
59     K=13
       IMAINM(K)=IPRSTA
       J=IMMPSW(48)
       J=ISPPSW(112)
       IT=1
       IERFLG=3
       CALL INSER1
C      MACHINE CHECK
       CALL ERINS
       CALL TERMIN
55     K=11
       IMAINM(K)=IPRSTA
       J=IMMPSW(40)
       J=ISPPSW(104)
1000   IF(IPARS.EQ.1) ILC=1
       IF(IPARS.GE.2.AND.IPARS.LE.4) ILC=2
       IF(IPARS.EQ.5) ILC=3
       IMAINM(K+1)=ISTORE(ILC,IMAINM(K+1),32,2)
       IT=2
       CALL INSER2
       RETURN
56     K=7
       IMAINM(K)=IPRSTA
       J=IMMPSW(24)
       J=ISPPSW(88)
       IT=3
       CALL INSER3
       RETURN
57     K=15
       IMAINM(K)=IPRSTA
       J=IMMPSW(56)
       J=ISPPSW(120)
       IT=4
       CALL INSER4
66     RETURN
       END
$IBFTC CCR016  DECK
       SUBROUTINE RSTART
       EQUIVALENCE (ISYSMK,ISPM(26)),(IPRMSK,ISPM(27)),(IAWP,ISPM(29)),
      1(ICC,ISPM(28))
       COMMON /TES/ LEVELS(4),ICUTPU(5),IPOINT(5)
       COMMON/UNCON/IFWA,IHOST,ITARG,IOPCW,IBIT,IPARS,IDATAS,IHW,MAXCOR,
      1 LOCG,APOS,TIMBO,NRCODE,NSTAT,LBPS1,LBPS2,NULL,IONE,IT,
      1CSLAE,ILC,INTCDE,ICOUNT,EXTIME,TIMLIM,
      1IMM,IMN,IERFLG,IRESTA,
```

```
$IBFTC CCR017  DECK
      FUNCTION  IAND(I,J)
C     HOST DEP
      K=I
       CALL BIN(K,0,36,0,J,1)
      IAND=K
      RETURN
      END
$IBFTC CCR018  DECK
C     HOST DEP
      FUNCTION IOR (I,J)
      K=I
      CALL BIN (K,4,32,4,J,2)
      IOR=K
      RETURN
      END
$IBFTC CCR019  DECK
      FUNCTION IER(I,J)
       L=INTNCT(IAND(I,J))
       IER=IAND(L,IOR(I,J))
      RETURN
      END
$IBFTC CCR020  DECK
      FUNCTION IMMPSW(K)
C     TARGET DEP
      COMMON /TES/ LEVELS(4),IOUTPU(5),IPOINT(5)
      COMMON/UNCON/IFWA,IHOST,ITARG,IOPCW,IBIT,IPARS,IDATAS,IHW,MAXCOR,
     1 LOCG,MPOS,TIMBO,NRCODE,NSTAT,LBPS1,LBPS2,NULL,IONE,IT,
     1CSLAE,ILC,INTCDE,ICOUNT,EXTIME,TIMLIM,
     1IMM,IMN,IERFLG,IRESTA,
     1LOBOUN,JIBOUN,MONE,IFETC
     1,MAXNEG,MAXBIT
      COMMON /UARRAY/ ISEGTA(20),TIME(10),CLASS(10),INSDAT(256,5) ,
     1INSPAR(5,20),IPCTR(5),IPPCTR(5),ISGCD(5,20),IADR(3),ISPM(64),
     1IHALFW(6),ISTACK(5,6),IMAINM(4096)
      COMMON /CVAR/ICPCO(255),ITCNTR,IOFFST,IOPRND(3),ISTAT
      EQUIVALENCE (ISYSMK,ISPM(26)),(IPRMSK,ISPM(27)),(IAWP,ISPM(29)),
     1 (ICC,ISPM(28)),(INACCR,ISPM(25))
      IDATAS=K
      CALL BRCHK(J)
      GO TO (10,300),J
10    CONTINUE
      I=K/4+1
      L=I+1
      IMAINM(L)=ISTORE(INACCR,IMAINM(L),24,24)
      IMAINM(I)=ISTORE(ISYSMK,IMAINM(I),32,8)
      IMAINM(L)=ISTORE(ICC,IMAINM(L),30,2)
      IMAINM(L)=ISTORE(IPRMSK,IMAINM(L),28,4)
      IMAINM(I)=ISTORE(IAWP,IMAINM(I),20,4)
      IMMPSW=1
300   RETURN
      END
$IBFTC CCR021  DECK
      FUNCTION ISPPSW(K)
C     TARGET DEP
      COMMON /TES/ LEVELS(4),IOUTPU(5),IPOINT(5)
      COMMON/UNCON/IFWA,IHOST,ITARG,IOPCW,IBIT,IPARS,IDATAS,IHW,MAXCOR,
     1 LOCG,MPOS,TIMBO,NRCODE,NSTAT,LBPS1,LBPS2,NULL,IONE,IT,
     1CSLAE,ILC,INTCDE,ICOUNT,EXTIME,TIMLIM,
     1IMM,IMN,IERFLG,IRESTA,
     1LOBOUN,JIBOUN,MONE,IFETC
```

```
      COMMON /UARRAY/ ISEGTA(20),TIME(10),CLASS(10),INSDAI(256,5) ,
     1ISSPAR(5,20),IPCTR(5),IPPCTR(5),ISGCD(5,20),IADR(3),ISPM(64),
     1HALFW(6),ISTACK(5,6),IMAINM(4096)
      COMMON /DVAR/IOPCO(255),ITCNTR,IOFFST,IOPRND(3),ISTAT
      EQUIVALENCE (ISYSMK,ISPM(26)),(IPRMSK,ISPM(27)),(IAWP,ISPM(29)),
     1(ICC,ISPM(28)),(INADDR,ISPM(25))
      IDATAS=K
      CALL BRCHK(J)
      GO TO (10,300), J
   10 CONTINUE
      I=K/4+1
      L=I+1
      INADDR=ILOAD(IMAINM(L),24,24)
      ISYSMK=ILOAD(IMAINM(I),32,8)
      IPRMSK=ILOAD(IMAINM(L),28,4)
      ICC=ILOAD(IMAINM(L),30,2)
      IAWP=ILOAD(IMAINM(I),20,4)
      ISPPSW=2
  300 RETURN
      END
$IBFTC CCR022  DECK
      FUNCTION INTNOT(K)
C    HOST DEP
      COMMON/UNCON/IFWA,IHOST,ITARG,IOPCW,IBIT,IPARS,I DATAS,IHW,MAXCOR,
     1 LOCG,MPGS,TIMBC,NRCODE,NSTAT,LBPS1,LBPS2,NULL,IONE,IT,
     1OSLAE,ILC,INTCDE,ICOUNT,EXTIME,TIMLIM,
     1IMM,IMN,IERFLG,IRESTA,
     1LOBOUN,JIBOUN,MONE,IFETC
     1,MAXNEG,MAXBIT
      L=K
      INTNOT=IAND(MAXBIT,ICCMP1(L,ITARG,ITARG))
      RETURN
      END
$IBFTC COR023  DECK
      SUBROUTINE INTSER
    1 FORMAT(12H       INSER1)
    2 FORMAT(12H       INSER2)
    3 FORMAT(12H       INSER3)
    4 FORMAT(12H       INSER4)
      ENTRY INSER1
      WRITE(6,1)
      RETURN
      ENTRY INSER2
      WRITE(6,2)
      RETURN
      ENTRY INSER3
      WRITE(6,3)
      RETURN
      ENTRY INSER4
      WRITE(6,4)
      RETURN
      END
C*********************************************************************
$IBFTC COR024  DECK
      INTEGER FUNCTION ITWTSM(I)
C     TWO COMP TO SIGN MAG
      DATA M1/1/
      COMMON/UNCON/IFWA,IHOST,ITARG,IOPCW,IBIT,IPARS,I DATAS,IHW,MAXCOR,
     1 LOCG,MPGS,TIMBC,NRCODE,NSTAT,LBPS1,LBPS2,NULL,IONE,IT,
     1OSLAE,ILC,INTCDE,ICOUNT,EXTIME,TIMLIM,
     1IMM,IMN,IERFLG,IRESTA,
```

```
      ISMTWO=MAXNEG
      RETURN
      END
$IBFTC CCR025  CECK
      INTEGER FUNCTICN ISMTWO(I)
C     SIGN MAG TO  TWC COMP
      IF(I.GE.0) GC TO 20
      COMMCN/UNCCN/IFWA,IHOST,ITARG,IOPCW,IBIT,IPARS,I DATAS,IHW,MAXCOR,
     1 LCCG,MPCS,TIMBC,NRCODE,NSTAT,LBPS1,LBPS2,NULL,IONE,IT,
     1OSLAE,ILC,INTCDE,ICCUNT,EXTIME,TIMLIM,
     1IMM,IMN,IERFLG,IRESTA,
     1LOBCUN,JIBCUN,MCNE,IFETC
     1,MAXNEG,MAXBIT
      L=INTNOT(I)+1
      ISMTWO=ICR(L,MAXNEG)
   10 RETURN
   20 IF(I.EQ.0) GC TO 30
      ISMTWC=I
      RETURN
   30 ISMTWC=0
      RETURN
      END
$IBFTC CCRO2   DECK
      FUNCTION ISTCRE(ISOR,IDEST,IS,IN)
      COMMON/UNCON/ IFWA,IHOST,ITARG,IOPCW,IBIT,IPARS,IDATAS,IHW,MAXCOR,
     1LOCG,M    TIMBC,NRCODE,NSTAT,LBPS1,LBPS2,NULL,IONE,IT,
     1OSLAE,I   INTCDE,ICCUNT,EXTIME,TIMLIM,
     1IMM,IM    FLG,IRESTA,
     1LOBCU     UN,MCNE,IFETC
     1,MAXN     BIT
      ITMP=
      CALL      MP,I   ST-IS,IN,IHOST-IN,ISOR)
      ISTORE=IT
      RETURN
      END
$IBFTC CCR027  CECK
      FUNCTION  JEXBIN(IBUF,IST,ILNG)
C     CONVERTS  CHAR STRING IN  HEX - INTERNAL 7094 TO  BINARY
C     IBUF IS  STRING LOC
C     IST  IS  PTR TO 1ST CHAR IN IBUF
C     ILNG IS  NR OF HEX CHARS
      INTEGER IBUF(80)
       COMMON/BUF1/ IBUF
      DATA I60B8/C60606060606060/
      IHOST=36
      K=0
      IBUMP=0
      IAUMP=0
      L=IST+ILNG
      DO 501 I=1,ILNG
      ILUMP=0
      M=L-I
      ICUMP=IBUF(M)
      IF(ICUMP.EQ.I60B8) GO TO 501
      K=K+1
      ILUMP=ILCAD(ICUMP,IHOST,6)
      IF(ILUMP.GT.9) ILUMP=ILUMP-7
      IAUMP=IAUMP+ILUMP*16**(K-1)
  501 CONTINUE
      JEXBIN=IAUMP
      RETURN
      END
```

```
      FUNCTION ILOAD(SOURCE,SB,NB)
      INTEGER SOURCE,SB,NB
      INTEGER ITEMP,QNBWD
      QNBWD = 36
      ITEMP = 0
      CALL FLD(ITEMP,QNBWD - NB,NB,QNBWD - SB,SOURCE)
      ILOAD = ITEMP
      RETURN
      END
$IBFTC CCR029  DECK
C     FN  TO EXTRACT SHIFT COUNT EITHER  DIRECT OR INDIRECT
      INTEGER FUNCTION ISHADR (K)
      COMMON /UARRAY/ ISEGTA(20),TIME(10),CLASS(10),INSDAT(256,5),
     1 INSPAR(5,20),IPCTR(5),IPPCTR(5),ISGCD(5,20),IADR(3),ISPM(64),
     1IHALFW(6),ISTACK(5,6),IMAINM(4096)
      ISHADR=ILOAD(IADR(1),6,6)
      K=0
      RETURN
      END
$IBFTC CCR030  DECK
      SUBROUTINE IVERFL(L,LOF)
C      HOST DEPENDENT
C     L IS SM RESULT   ICATAS IS OPERAND
      COMMON/UNCON/IFWA,IHOST,ITARG,IOPCW,IBIT,IPARS,I DATAS,IHW,MAXCOR,
     1 LOCG,MPCS,TIMBC,NRCODE,NSTAT,LBPS1,LBPS2,NULL,IONE,IT,
     1OSLAB,ILC,INTCDE,ICOUNT,EXTIME,TIMLIM,
     1IMM,IMN,IERFLG,IRESTA,
     1LOBOUN,JIBOUN,MONE,IFETC
     1,MAXNEG,MAXBIT
      COMMON /UARRAY/ ISEGTA(20),TIME(10),CLASS(10),INSDAT(256,5),
     1 INSPAR(5,20),IPCTR(5),IPPCTR(5),ISGCD(5,20),IADR(3),ISPM(64),
     1IHALFW(6),ISTACK(5,6),IMAINM(4096)
      EQUIVALENCE(LL,MAXBIT)
      N=IAND(ICATAS,MAXNEG)
      IF(M.GT.0.AND.N.EQ.0.OR.M.EQ.0.AND.N.GT.0) GOTO 20
C     ELSE LIKE SIGNS
      IF(M.GT.0) GO TO 30
C     POSITIVE CASE
      IF(IAND(MAXNEG,L).EQ.NULL) GO TO 20
C     1 IS O/F
40    LOF=1
      RETURN
C     NEGATIVE CASE
30    IF(IAND(L,MAXNEG).EQ.0) GO TO 40
C     2 IS NO/F
20    LOF=2
      RETURN
C     O/F
      END
$IBFTC COR031  DECK
      FUNCTION ISLODE(N)
      COMMON/OVAR/IOPCD(255),ITCNTR,IOFFST,IOPRND(3),ISTAT
      COMMON /UARRAY/ ISEGTA(20),TIME(10),CLASS(10),INSDAT(256,5),
     1 INSPAR(5,20),IPCTR(5),IPPCTR(5),ISGCD(5,20),IADR(3),ISPM(64),
     1IHALFW(6),ISTACK(5,6),IMAINM(4096)
      COMMON/UNCON/ IFWA,IHOST,ITARG,IOPCW,IBIT,IPARS,IDATAS,IHW,MAXCOR,
     1LOCG,MPCS,TIMBC,NRCODE,NSTAT,LBPS1,LBPS2,NULL,IONE,IT,
     1OSLAB,ILC,INTCDE,ICOUNT,EXTIME,TIMLIM,
     1IMM,IMN,IERFLG,IRESTA,
     1LOBOUN,JIBOUN,MONE,IFETC
     1,MAXNEG,MAXBIT
```

```
      GO TO (3406,3407,3408),N
 3406 IDATAS=IAND(IDATAS,M)
      GO TO 3409
 3407 IDATAS=IER(IDATAS,M)
      GO TO 3409
 3408 IDATAS=IOR(IDATAS,M)
 3409 IF(IDATAS.GT.NULL) GO TO 3403
      ICC=0
 3404 CALL STORCH(K,$300)
      K=K+4
 3402 M=M+4
      GO TO 300
 3403 ICC=1
      GO TO 3404
  300 RETURN
      END
$IBFTC COR032  DECK
      FUNCTION IOVRFL(N,J)
C     J=1,O/F,  =2 N   O/F
      COMMON/CVAR/ICPCC(255),ITCNTR,IOFFST,IOPRND(3),ISTAT
      COMMON /UARRAY/ ISEGTA(20),TIME(10),CLASS(10),INSDAT(256,5),
     1 INSPAR(5,20),IPCTR(5),IPPCTR(5),ISGCD(5,20),IADR(3),ISPM(64),
     1IHALFW(6),ISTACK(5,6),IMAINM(4095)
      COMMON/UNCON/ IFWA,IHCST,ITARG,ICPCW,IBIT,IPARS,IDATAS,IHW,MAXCOR,
     1 LOCG,MPCS,TIMPC,NRCODE,NSTAT,LBPS1,LBPS2,NULL,IONE,IT,
     1CSLAE,ILC,INTCCE,ICCUNT,EXTIME,TIMLIM,
     1IMM,IMN,IERFLG,IRESTA,
     1LOBOUN,JIBOUN,MONE,IFETC
     1,MAXNEG,MAXBIT
      IOVRFL=N
C     BOTH +
      IF(IAND(MAXNEG,ISPM(IMM)).EQ.0.AND.IAND(MAXNEG,IDATAS).EQ.0)
     1  GO TO 10
C     BOTH -
      IF(IAND(MAXNEG,ISPM(IMM)).NE.0.AND.IAND(MAXNEG,IDATAS).NE.0)
     1  GO TO 20
C     -,+  RESULT +
      IF(IAND(MAXNEG,N).EQ.0) GO TO 20
   10 J=2
      RETURN
   20 J=1
      RETURN
      END
$IBFTC COR033  DECK
      FUNCTION ICOMP1(WRDIN,SB,NB)
      INTEGER WRDIN,SB,NB,TWORD,QMAXHV
      QMAXHV=34359738367
      TWORD=ILOAD(WRDIN,SB,NB)
      TWORD=QMAXHV-TWORD
      ICOMP1=ISTORE(TWORD,WRDIN,SB,NB)
      RETURN
      END
$IBFTC TYPESP  DECK
      BLOCK DATA
C
      COMMON /CVAR/ICPCC(255),ITCNTR,IOFFST,IOPRND(3),ISTAT
C
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C   THE ARRAY IOPCC ASSIGNS AN INSTRUCTION TYPE NUMBER TO
C   EACH VALID SUPC INSTRUCTION. THE INSTRUCTIONS ARE THUS CLASSIFIED
C   ACCORDING TO THE TYPE AND NUMBER OF KEYS WHICH THEY CONTAIN.
C   INSTRUCTION TYPE 0 DESIGNATES AN INVALID INSTRUCTION OR ONE
```

```
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C
      DATA ICPCO/    0, 0, 0, 1, 2, 2, 1, 2, 2, 0, 0, 0, 0, 0,
     1              2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
     1              2, 2, 2, 2, 2, 0, 0, 0, 2, 2, 2, 2, 2, 2, 2, 2,
     1              2, 2, 2, 2, 2, 0, 0, 0, 2, 2, 2, 2, 2, 2, 2, 2,
     1              4, 4, 4, 4, 2, 4, 4, 5, 4, 4, 4, 4, 4, 0, 4, 4,
     1              4, 0, 0, 0, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4,
     1              4, 0, 0, 0, 0, 0, 0, 0, 4, 4, 4, 4, 4, 4, 4, 4,
     1              4, 0, 0, 0, 0, 0, 0, 0, 4, 4, 4, 4, 4, 4, 4, 4,
     1              5, 0, 5, 5, 5, 5, 3, 3, 1, 1, 1, 1, 1, 1, 1, 1,
     1              0, 5, 5, 5, 5, 5, 5, 5, 0, 0, 0, 0, 5, 5, 5, 5,
     1              0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
     1              0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
     1              0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
     1              0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
     1              0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
     1              0, 6, 6, 6, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0/
C
      END
$IBFTC HEADSR  DECK
      SUBROUTINE HEADER
C
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C   DEFINE COMMON
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C
      COMMON /DVAR/ICPCO(255),ITCNTR,IOFFST,IOPRND(3),ISTAT
      COMMON/UNCON/IFWA,IHOST,ITARG,IOPCW,IBIT,IPARS,IDATAS,IHW,MAXCOR,
     1 LOCG,MPCS,TIMBO,NRCODE,NSTAT,LBPS1,LBPS2,NULL,IONE,IT,
     1CSLAE,ILC,INTCDE,ICOUNT,EXTIME,TIMLIM,
     1IMM,INN,IERFLG,IRESTA,
     1LOBCUN,JIBCUN,MONE,IFETC
     1,MAXNEG,MAXBIT
      COMMON /UARRAY/ ISEGTA(20),TIME(10),CLASS(10),INSDAT(256,5),
     1INSPAR(5,20),IPCTR(5),IPPCTR(5),ISGCO(5,20),IADR(3),ISPM(64),
     1IHALFW(6),ISTACK(5,6),IMAINM(4096)
      EQUIVALENCE (INADDR,ISPM(25))
C
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C   PRINT CURRENT VALUES OF PROGRAM OFFSET, ELAPSED TIME,
C   PCOUNTER, AND INSTRUCTION CONTENTS.
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C
      WRITE(6,501) IOFFST
  501 FORMAT(1H0,22H PROGRAM IS AT OFFSET ,I6,35H FROM FIRST EXECUTABLE
     1INSTRUCTION.)
      WRITE(6,502) TIMBO
  502 FORMAT(1H0,27H SIMULATED ELAPSED TIME IS ,F12.3,6H MSEC.)
      WRITE(6,503) INADDR
  503 FORMAT(1H0,46H THE CURRENT INSTRUCTION ADDRESS(PCOUNTER) IS ,I6)
      K=ISEGTA(1)
      IJ=ICPCO(K)
      IF((IJ.EQ.2).OR.(K.EQ.4).OR.(K.EQ.7)) GO TO 551
      IF (IJ.NE.6) GO TO 552
      WRITE(6,504) IHALFW(1),IHALFW(2),IHALFW(3)
  504 FORMAT(1H0,48H THE CURRENT INSTRUCTION(IN OCTAL HALFWORDS) IS ,O6,
     11X,O6,1X,O6)
      GO TO 553
  552 WRITE(6,505) IHALFW(1),IHALFW(2)
  505 FORMAT(1H0,48H THE CURRENT INSTRUCTION(IN OCTAL HALFWORDS) IS ,O6,
     11X,O6)
```

```
C * *   * * * * * * * * * * * * * * * * * * * * * * * * * *   * * * * *
C
  553 IF (IJ.EQ.0) GO TO 554
      IF ((IJ.EQ.5).OR.(IJ.EQ.6)) GO TO 555
      IK=1
      K=IMM-1
      WRITE(6,507) IK,K,ISPM(IMM)
  507 FORMAT(1H0,18H REGISTER OPERAND ,I1,17H AT SPM ADDRESS  ,I3,11H IS
     1(OCTAL) ,O11)
      IF (IJ.EQ.1) GO TO 554
      IF (IJ.EQ.4) GO TO 555
      IK=2
      K=IMN-1
      WRITE(6,507) IK,K,ISPM(IMN)
      IF (IJ.EQ.2) GO TO 554
  555 IK=1
      WRITE(6,509) IK,IACR(1),IOPRND(1)
  509 FORMAT(1H0,16H MEMORY OPERAND ,I1,16H AT MM ADDRESS  ,I6,11H IS(OC
     1TAL) ,O11)
      IF (IJ.NE.6) GO TO 554
      IK=2
      WRITE(6,509) IK,IACR(2),IOPRND(2)
C
  554 RETURN
      END
$IBFTC SNAPRS  DECK
      SUBROUTINE SNAPRC
C
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C   DIMENSION SNAP VARIABLES AND DEFINE COMMON
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C
      LOGICAL SNAP
      LOGICAL FSNAP,TISNAP,KSNAP,
     1         PCSNAP,OCSNAP,TSNAP,LTSNAP(9),MASNAP,RASNAP,MOSNAP,ROSNAP
      INTEGER SLOC(9),TISLOC(9),NTSKL(9),TSLOC(9,9)
      REAL TSK(9)
      INTEGER PCSK(9),NPCSKL(9),PCSLOC(9,9),
     1         OCSK(9),NOCSKL(9),OCSLOC(9,9),
     1         MASK(9),NMASKL(9),MASLOC(9,9),
     1         RASK(9),NRASKL(9),RASLOC(9,9),
     1         MOSKA(9),MOSK(9),NMOSKL(9),MOSLOC(9,9),
     1         ROSKA(9),ROSK(9),NROSKL(9),ROSLOC(9,9)
C
      COMMON /FS/SNAP,FSNAP,NFSL,SLOC,
     1                  TISNAP,STI,SSTRT,SSTPT,NTISNA,TISLOC
     1        /KS/KSNAP,PCSNAP,NPCSK,PCSK,NPCSKL,PCSLOC,
     1                  OCSNAP,NOCSK,OCSK,NOCSKL,OCSLOC,
     1                  MASNAP,NMASK,MASK,NMASKL,MASLOC,
     1                  RASNAP,NRASK,RASK,NRASKL,RASLOC,
     1                  TSNAP,NTSK,TSK,NTSKL,TSLOC,LTSNAP,
     1                  MOSNAP,NMOSK,MOSKA,MOSK,NMOSKL,MOSLOC,
     1                  ROSNAP,NROSK,ROSKA,ROSK,NROSKL,ROSLOC
C
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C     THE REQUIRED SNAP/NO-SNAP VARIABLE IS READ HERE.
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C
      READ 1001,SNAP
 1001 FORMAT(L5)
      IF(.NOT.SNAP) GO TO 2301
C
```

```
C      THE FULLSNAP/NCFULLSNAP VARIABLE IS READ HERE AND,IF TRUE, IS THE
C      ONLY SNAP DIAGNOSTIC ALLOWED.
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C
       READ 1001,FSNAP
       IF(.NOT.FSNAP) GO TO 2002
       READ 1002,NFSL
 1002  FORMAT(I5)
 1003  FORMAT(9I5)
       READ 1003,(SLCC(I),I=1,NFSL)
       GO TO 2001
C
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C      THE PRESENCE/ABSENCE OF AN INTERVAL SNAP IS READ HERE AND
C      APPROPRIATE DATA IS READ IF DIAGNOSTIC IS DESIRED.
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C
 2002  READ 1001,TISNAP
       IF(.NOT.TISNAP) GO TO 2003
       READ 1002,NTISNA
       READ 1004,STI,SSTRT,SSTPT
 1004  FORMAT(3F12.3)
       READ 1003,(TISLCC(I),I=1,NTISNA)
C
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C      THE KEYEDSNAP/NCKEYEDSNAP VARIABLE IS READ HERE. IF KEYED SNAPS
C      ARE PRESENT, THE KEYS ARE READ ALONG WITH APPROPRIATE DIAGNOSTIC
C      DATA.
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C
 2003  READ 1001,KSNAP
       IF(.NOT.KSNAP) GO TO 2001
C
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C      CHECK FOR PRESENCE/ABSENCE OF P-COUNTER-KEYED SNAP.
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C
       READ 1001,PCSNAP
       IF(.NOT.PCSNAP)GO TO 2004
       READ 1002,NPCSK
       DO 001 I=1,NPCSK
       READ 1005,PCSK(I),NPCSKL(I)
 1005  FORMAT(I6,I4)
       JJ=NPCSKL(I)
  001  READ 1003,(PCSLCC(I,J),J=1,JJ)
C
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C      CHECK FOR PRESENCE/ABSENCE OF OP-CODE-KEYED SNAP.
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C
 2004  READ 1001,CCSNAP
       IF(.NOT.CCSNAP) GO TO 2005
       READ 1002,NCCSK
       DO 002 I=1,NCCSK
       READ 1006,CCSK(I),NCCSKL(I)
 1006  FORMAT(I4,I4)
       JJ=NCCSKL(I)
  002  READ 1003,(CCSLCC(I,J),J=1,JJ)
C
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C      CHECK FOR PRESENCE/ABSENCE OF TIME-KEYED SNAP.
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
```

```
 1020 FORMAT(F12.3,I4)
      JJ=NTSKL(I)
  003 READ 1003,(TSLOC(I,J),J=1,JJ)
C
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C     CHECK FOR PRESENCE/ABSENCE OF MEMORY ADDRESS-KEYED SNAP.
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C
 2006 READ 1001,MASNAP
      IF(.NOT.MASNAP) GO TO 2007
      READ 1002,NMASK
      DO 004 I=1,NMASK
      READ 1005,MASK(I),NMASKL(I)
      JJ=NMASKL(I)
  004 READ 1003,(MASLOC(I,J),J=1,JJ)
C
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C     CHECK FOR PRESENCE/ABSENCE OF REGISTER ADDRESS-KEYED SNAP.
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C
 2007 READ 1001,RASNAP
      IF(.NOT.RASNAP) GO TO 2008
      READ 1002,NRASK
      DO 005 I=1,NRASK
      READ 1005,RASK(I),NRASKL(I)
      JJ=NRASKL(I)
  005 READ 1003,(RASLOC(I,J),J=1,JJ)
C
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C     CHECK FOR PRESENCE/ABSENCE OF MEMORY OPERAND-KEYED SNAP.
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C
 2008 READ 1001,MOSNAP
      IF(.NOT.MOSNAP) GO TO 2009
      READ 1002,NMOSK
      DO 006 I=1,NMOSK
      READ 1007,MOSKA(I),MOSK(I),NMOSKL(I)
 1007 FORMAT(I6,O11,I4)
      JJ=NMOSKL(I)
  006 READ 1003,(MOSLOC(I,J),J=1,JJ)
C
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C     CHECK FOR PRESENCE/ABSENCE OF REGISTER OPERAND-KEYED SNAP.
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C
 2009 READ 1001,ROSNAP
      IF(.NOT.ROSNAP) GO TO 2001
      READ 1002,NROSK
      DO 007 I=1,NROSK
      READ 1007,ROSKA(I),ROSK(I),NROSKL(I)
      JJ=NROSKL(I)
  007 READ 1003,(ROSLOC(I,J),J=1,JJ)
C
 2001 RETURN
      END
$IBFTC SNAPSR  DECK
      SUBROUTINE SNAPEX
C
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C   DIMENSION SNAP VARIABLES AND DEFINE COMMON
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
```

```
      LOGICAL FSNAP,TISNAP,KSNAP,
     1        PCSNAP,OCSNAP,TSNAP,LTSNAP(9),MASNAP,RASNAP,MOSNAP,ROSNAP
      INTEGER SLOC(9),TISLOC(9),NTSKL(9),TSLOC(9,9)
      REAL TSK(9)
      INTEGER PCSK(9),NPCSKL(9),PCSLOC(9,9),
     1        OCSK(9),NOCSKL(9),OCSLOC(9,9),
     1        MASK(9),NMASKL(9),MASLOC(9,9),
     1        RASK(9),NRASKL(9),RASLOC(9,9),
     1        MOSKA(9),MOSK(9),NMOSKL(9),MOSLOC(9,9),
     1        ROSKA(9),ROSK(9),NROSKL(9),ROSLOC(9,9)
C
      COMMON /FS/SNAP,FSNAP,NFSL,SLOC,
     1                 TISNAP,STI,SSTRT,SSTPT,NTISNA,TISLOC
     1         /KS/KSNAP,PCSNAP,NPCSK,PCSK,NPCSKL,PCSLOC,
     1                 OCSNAP,NOCSK,OCSK,NOCSKL,OCSLOC,
     1                 MASNAP,NMASK,MASK,NMASKL,MASLOC,
     1                 RASNAP,NRASK,RASK,NRASKL,RASLOC,
     1                 TSNAP,NTSK,TSK,NTSKL,TSLOC,LTSNAP,
     1                 MOSNAP,NMOSK,MOSKA,MOSK,NMOSKL,MOSLOC,
     1                 ROSNAP,NROSK,ROSKA,ROSK,NROSKL,ROSLOC
      COMMON/CVAR/ ICPCO(255),ITCNTR,IOFFST,IOPRND(3)
      COMMON/UNCON/IFWA,IHOST,ITARG,IOPCW,IBIT,IPARS,IDATAS,IHW,MAXCOR,
     1 LOCG,NPCS,TIMBC,NRCODE,NSTAT,LBPS1,LBPS2,NULL,IONE,IT,
     1OSLAE,ILC,INTCDE,ICOUNT,EXTIME,TIMLIM,
     1IMM,IMN,IERFLG,IRESTA,
     1LOBOUN,JIBOUN,MONE,IFETC
     1,MAXNEG,MAXBIT
       COMMON /UARRAY/ ISEGTA(20),TIME(10),CLASS(10),INSDAT(256,5),
     1INSPAR(5,20),IPCTR(5),IPPCTR(5),ISGCD(5,20),IADR(3),ISPM(64),
     1IHALFW(6),ISTACK(5,6),IMAINM(4096)
      EQUIVALENCE (INACDR,ISPM(25))
C
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C   IF A FULL SNAP DIAGNOSTIC IS IN EFFECT, THE DESIRED MM LOCATIONS
C   ARE SNAPPED AND ANY OTHER SNAP DIAGNOSTICS ARE NOT ALLOWED.
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C
      IF(.NOT.FSNAP) GO TO 100
      WRITE(6,101)
  101 FORMAT(1H1,50H ******************DIAGNOSTICS*******************)
      WRITE(6,102)
  102 FORMAT(1H0,45H A FULL SNAP DIAGNOSTIC ROUTINE IS IN EFFECT.)
      CALL HEADER
      WRITE(6,104)
  104 FORMAT(1H0,5X,11HMM LOCATION,7X,14HOCTAL CONTENTS)
      DO 103 I=1,NFSL
      II=SLOC(I)
      III=IMAINM(II)
  105 FORMAT(1H0,7X,I6,12X,O11)
  103 WRITE(6,105) II,III
      GO TO 409
C
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C   THE TIME INTERVAL SNAP KEY IS CHECKED AND APPROPRIATE VARIABLES
C   ARE UPDATED IF A SNAP IS PERFORMED.
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C
  100 IF(.NOT.TISNAP) GO TO 106
      IF(SSTRT-TIMBC) 107,107,106
  107 WRITE(6,101)
      WRITE(6,125)
  125 FORMAT(1H0,14H SNAP REQUEST.)
```

```
  124 FORMAT(1H),65H THE SNAPPED LOCATIONS AND THEIR (OCTAL) CONTENTS AR
     1E AS FOLLOWS.)
      WRITE(6,104)
      DO 111 I=1,NTISNA
      II=TISLOC(I)
      III=IMAINM(II)
  111 WRITE(6,105) II,III
      GO TO 100
C
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C   THE PRESENCE/ABSENCE OF ANY KEYED SNAPS IS CHECKED
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C
  106 IF(.NOT.KSNAP) GO TO 409
C
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C   INSTRUCTION ADDRESS(PCOUNTER) SNAP KEYS ARE CHECKED
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C
      IF(.NOT.PCSNAP) GO TO 401
      DO 301 I=1,NPCSK
      IF(PCSK(I).NE.INADDR) GO TO 301
      WRITE(6,101)
      WRITE(6,125)
      WRITE(6,112) PCSK(I)
  112 FORMAT(1H0,44H TRIGGERED BY INSTRUCTION ADDRESS(PCOUNTER) ,I6)
      CALL HEADER
      WRITE(6,124)
      WRITE(6,104)
      K=NPCSKL(I)
      DO 302 J=1,K
      II=PCSLOC(I,J)
      III=IMAINM(II)
  302 WRITE(6,105) II,III
  301 CONTINUE
C
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C   OP CODE SNAP KEYS ARE CHECKED
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C
  401 IF(.NOT.OCSNAP) GO TO 402
      DO 303 I=1,NOCSK
      IF(OCSK(I).NE.ISEGTA(1)) GO TO 303
      WRITE(6,101)
      WRITE(6,125)
      WRITE(6,113) OCSK(I)
  113 FORMAT(1H0,32H TRIGGERED BY (DECIMAL) OP CODE ,I4)
      CALL HEADER
      WRITE(6,124)
      WRITE(6,104)
      K=NOCSKL(I)
      DO 304 J=1,K
      II=OCSLOC(I,J)
      III=IMAINM(II)
  304 WRITE(6,105) II,III
  303 CONTINUE
C
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C   TIME SNAP KEYS ARE CHECKED
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C
```

```
      DO 404 I=1,NTSK
      IF(.NOT.LTSNAP(I)) GO TO 404
      IF(TIMBC-TSK(I)) 404,405,405
  405 LTSNAP(I)=.FALSE.
      WRITE(5,101)
      WRITE(6,125)
      WRITE(6,114) TSK(I)
  114 FORMAT(1H0,35H TRIGGERED BY PROGRAM ELAPSED TIME ,F12.3,6H MSEC.)
      CALL HEADER
      WRITE(6,124)
      WRITE(6,104)
      K=NTSKL(I)
      DO 406 J=1,K
      II=TSLOC(I,J)
      III=IMAINY(II)
  406 WRITE(6,105) II,III
  404 CONTINUE
C
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C   THE PRESENCE/ABSENCE OF REGISTER OR MEMORY SNAP KEYS IS CHECKED
C   AND BRANCH TO APPROPRIATE LOGIC IS TAKEN.
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C
  403 IF((.NOT.RASNAP).AND.(.NOT.MASNAP).AND.(.NOT.ROSNAP).AND.(.NOT.MOS
     1NAP)) GO TO 409
      K=ISEGTA(1)
      IJ=IOPCD(K)
      IF(IJ.LT.1) GO TO 409
C
      GO TO (407,407,407,407,408,408),IJ
C
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C   REGISTER OPERAND 1 ADDRESS KEY IS CHECKED
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C
  407 IF(.NOT.RASNAP) GO TO 410
      DO 305 I=1,NRASK
      IF(RASK(I).NE.ISEGTA(2)) GO TO 305
      WRITE(5,101)
      WRITE(6,125)
      WRITE(6,115) RASK(I)
  115 FORMAT(1H0,31H TRIGGERED BY REGISTER ADDRESS ,I4)
      CALL HEADER
      WRITE(6,124)
      WRITE(6,104)
      K=NRASKL(I)
      DO 411 J=1,K
      II=RASLOC(I,J)
      III=IMAINY(II)
  411 WRITE(6,105) II,III
  305 CONTINUE
C
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C   REGISTER OPERAND 1 KEY IS CHECKED
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C
  410 IF(.NOT.ROSNAP) GO TO 490
      DO 306 I=1,NROSK
      IF(ROSKA(I).NE.ISEGTA(2)) GO TO 306
      IF(ROSK(I).NE.ISPY(IMY)) GO TO 306
      WRITE(5,101)
```

```
      DO 412 J=1,K
      II=RCSLCC(I,J)
      III=IMAINM(III)
  412 WRITE(6,105) II,III
  306 CONTINUE
C
  490 GC TO (409,413,413,408,408,408),IJ
C
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C   REGISTER CPERAND 2 ACCRESS KEY IS CHECKED
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C
  413 IF(.NCT.RASNAP) GC TO 414
      DC 307 I=1,NRASK
      IF(RASK(I).NE.ISEGTA(3)) GO TO 3C7
      WRITE(6,101)
      WRITE(6,125)
      WRITE(6,115) RASK(I)
      CALL HEACER
      WRITE(6,124)
      WRITE(6,104)
      K=NRASKL(I)
      DO 415 J=1,K
      II=RASLCC(I,J)
      III=IMAINM(II)
  415 WRITE(6,105) II,III
  307 CONTINUE
C
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C   REGISTER OPERANC 2 KEY IS CHECKED
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C
  414 IF(.NCT.ROSNAP) GC TO 491
      DO 308 I=1,NRCSK
      IF(ROSKA(I).NE.ISEGTA(3)) GO TO 308
      IF(RCSK(I).NE.ISPM(IMN)) GO TO 3C8
      WRITE(6,101)
      WRITE(6,125)
      WRITE(6,116) RCSK(I),ROSKA(I)
      CALL HEACER
      WRITE(6,124)
      WRITE(6,104)
      K=NRCSKL(I)
      DC 416 J=1,K
      II=RCSLOC(I,J)
      III=IMAINM(II)
  416 WRITE(6,105) II,III
  308 CONTINUE
C
  491 GC TC (409,409,408,408,408,408),IJ
C
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C   MEMCRY CPERANC IIII ACCRESS KEY IS CHECKED
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C
  408 IIII=1
  493 IF(.NCT.MASNAP) GO TO 492
      DC 309 I=1,NMASK
      IF(MASK(I).NE.IACR(IIII)) GO TO 3C9
      WRITE(6,101)
      WRITE(6,125)
      WRITE(6,117) MASK(I)
```

```
      CALL HEADER
      WRITE(6,124)
      WRITE(6,104)
      K=NMASKL(I)
      DO 417 J=1,K
      II=MASLOC(I,J)
      III=IMAINM(II)
  417 WRITE(6,105) II,III
  309 CONTINUE
C
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C   MEMORY OPERAND IIII KEY IS CHECKED
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C
  492 IF(.NOT.MOSNAP) GO TO 494
      DO 310 I=1,NMOSK
      IF(MOSKA(I).NE.IACR(IIII)) GO TO 310
      IF(MOSK(I).NE.ICPRNC(IIII)) GO TO 310
      WRITE(6,101)
      WRITE(6,125)
      WRITE(6,118) MOSK(I),MOSKA(I)
  118 FORMAT(1H0,29H TRIGGERED BY MEMORY OPERAND ,O11,13H AT LOCATION ,I
     16)
      CALL HEADER
      WRITE(6,124)
      WRITE(6,104)
      K=NMOSKL(I)
      DO 418 J=1,K
      II=MOSLOC(I,J)
      III=IMAINM(II)
  418 WRITE(6,105) II,III
  310 CONTINUE
C
  494 IIII=IIII+1
      IF ((IIII.EQ.2).AND.(IJ.EQ.6)) GO TO 493
C
  409 RETURN
      END
$IBFTC TRACRS  DECK
      SUBROUTINE TRACRC
C
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C   DIMENSION TRACE VARIABLES AND DEFINE COMMON
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C
      LOGICAL TRACE
      LOGICAL FTRACE,TITRAC,KTRACE,
     1        PCTRAC,OCTRAC,TTRACE,LTTRAC(9),MATRAC,RATRAC,MOTRAC,ROTRAC
      INTEGER PCTK(9),NIPCTK(9),OCTK(9),NIOCTK(9),NITTK(9),
     1        MATK(9),NIMATK(9),RATK(9),NIRATK(9),
     1        MOTKA(9),MOTK(9),NIMOTK(9),ROTKA(9),ROTK(9),NIROTK(9)
      REAL TTK(9)
C
      COMMON /TR/TRACE,FTRACE,TITRAC,TRTI,TSTRT,TSTPT,KTRACE
     1       /KT/PCTRAC,NPCTK,PCTK,NIPCTK,OCTRAC,NOCTK,OCTK,NIOCTK,
     1          MATRAC,NMATK,MATK,NIMATK,RATRAC,NRATK,RATK,NIRATK,
     1          TTRACE,NTTK,TTK,NITTK,LTTRAC,
     1          MOTRAC,NMOTK,MOTKA,MOTK,NIMOTK,
     1          ROTRAC,NROTK,ROTKA,ROTK,NIROTK
C
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
```

```
C     THE FULLTRACE/NOFULLTRACE VARIABLE IS READ HERE AND, IF TRUE, IS
C     THE ONLY TRACE DIAGNOSTIC ALLOWED.
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C
      READ 1001,FTRACE
      IF(FTRACE) GO TO 2010
C
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C     THE PRESENCE/ABSENCE OF AN INTERVAL TRACE IS READ HERE AND
C     APPROPRIATE DATA IS READ IF DIAGNOSTIC IS ALLOWED.
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C
      READ 1001,TITRAC
      IF(.NOT.TITRAC) GO TO 2011
      READ 1008,TRTI,TSTRT,TSTPT
 1008 FORMAT(3F12.3)
C
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C     THE KEYEDTRACE/NOKEYED TRACE VARIABLE IS READ HERE. IF KEYED
C     TRACES ARE PRESENT, THE KEYS ARE READ ALONG WITH APPROPRIATE
C     DIAGNOSTIC DATA.
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C
 [illegible]011 READ [illegible]TRACE
      IF(.N[illegible]RACE) GO TO 2010
C
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C     CHECK [illegible]PRESE[illegible]E/ABSENCE OF P-COUNTER-KEYED T[illegible].
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C
      READ 1[illegible]CTRA[illegible]
      IF(.NOT.[illegible]CTRAC) GO TO 2012
      READ 1002,NPCTK
 1002 FORMAT(I5)
      DO 404 I=1,NPCTK
  404 READ 1009,PCTK(I),NIPCTK(I)
 1009 FORMAT(I6,I4)
C
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C     CHECK FOR PRESENCE/ABSENCE OF OP-CODE-KEYED TRACE.
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C
 2012 READ 1001,OCTRAC
      IF(.NOT.OCTRAC) GO TO 2013
      READ 1002,NOCTK
      DO 405 I=1,NOCTK
  405 READ 1010,OCTK(I),NIOCTK(I)
 1010 FORMAT(I4,I4)
C
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C     CHECK FOR PRESENCE/ABSENCE OF TIME-KEYED TRACE.
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C
 2013 READ 1001,TTRACE
      IF(.NOT.TTRACE) GO TO 2014
      READ 1002,NTTK
      DO 400 I=1,NTTK
  400 LTTRAC(I)=.TRUE.
      DO 406 I=1,NTTK
  406 READ 1011,TTK(I),NITTK(I)
 1011 FORMAT(F12.3,I4)
C
```

```
C     CHECK FOR PRESENCE/ABSENCE OF MEMORY ADDRESS-KEYED TRACE.
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C
 2014 READ 1001,MATRAC
      IF(.NOT.MATRAC) GO TO 2015
      READ 1002,NMATK
      DO 407 I=1,NMATK
  407 READ 1009,MATK(I),NIMATK(I)
C
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C     CHECK FOR PRESENCE/ABSENCE OF REGISTER ADDRESS-KEYED TRACE.
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C
 2015 READ 1001,RATRAC
      IF(.NOT.RATRAC) GO TO 2016
      READ 1002,NRATK
      DO 408 I=1,NRATK
  408 READ 1010,RATK(I),NIRATK(I)
C
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C     CHECK FOR PRESENCE/ABSENCE OF MEMORY OPERAND-KEYED TRACE.
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C
 2016 READ 1001,MOTRAC
      IF(.NOT.MOTRAC) GO TO 2017
      READ 1002,NMOTK
      DO 409 I=1,NMOTK
  409 READ 1012,MOTKA(I),MOTK(I),NIMOTK(I)
 1012 FORMAT(I6,O11,I4)
C
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C     CHECK FOR PRESENCE/ABSENCE OF REGISTER OPERAND-KEYED TRACE.
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C
 2017 READ 1001,ROTRAC
      IF(.NOT.ROTRAC) GO TO 2010
      READ 1002,NROTK
      DO 410 I=1,NROTK
  410 READ 1012,ROTKA(I),ROTK(I),NIROTK(I)
C
 2010 RETURN
      END
$IBFTC TRACSR  DECK
      SUBROUTINE TRACEX
C
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C   DIMENSION TRACE VARIABLES AND DEFINE COMMON
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C
      LOGICAL FTRACE,TITRAC,KTRACE,
     1        PCTRAC,OCTRAC,TTRACE,LTTRAC(9),MATRAC,RATRAC,MOTRAC,ROTRAC
      INTEGER PCTK(9),NIPCTK(9),OCTK(9),NIOCTK(9),NITTK(9),
     1        MATK(9),NIMATK(9),RATK(9),NIRATK(9),
     1        MOTKA(9),MOTK(9),NIMOTK(9),ROTKA(9),ROTK(9),NIROTK(9)
      REAL TTK(9)
C
      COMMON /TR/TRACE,FTRACE,TITRAC,TRTI,TSTRT,TSTPT,KTRACE
     1       /KT/PCTRAC,NPCTK,PCTK,NIPCTK,OCTRAC,NOCTK,OCTK,NIOCTK,
     1           MATRAC,NMATK,MATK,NIMATK,RATRAC,NRATK,RATK,NIRATK,
     1           TTRACE,NTTK,TTK,NITTK,LTTRAC,
     1           MOTRAC,NMOTK,MOTKA,MOTK,NIMOTK,
     1           ROTRAC,NROTK,ROTKA,ROTK,NIROTK
```

```
      COMMON /UARRAY/ ISEGTA(20),TIME(10),CLASS(10),INSDAT(36,5),
     1INSPAR(5,20),IPCTR(5),IFPCTR(5),ISGCD(5,20),IADR(3),ISPM(64),
     1IHALFW(6),ISTACK(5,6),IMAINM(4096)
      EQUIVALENCE (INACCR,ISPM(25))
C
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C   IF A FULL TRACE DIAGNOSTIC IS IN EFFECT, THE REGISTER TRACE IS
C   PERFORMED AND ANY OTHER TRACE DIAGNOSTICS ARE NOT ALLOWED.
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C
      IF (.NOT.FTRACE) GO TO 600
      WRITE(6,701)
  701 FORMAT(1H1,50H ***************** DIAGNOSTICS ****************** )
      WRITE(6,702)
  702 FORMAT(1H0,38H A FULL TRACE DIAGNOSTIC IS IN EFFECT.)
      CALL HEADER
      GO TO 697
C
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C   THE TIME-INTERVAL-TRACE KEYS ARE CHECKED AND APPROPRIATE
C   VARIABLES ARE UPDATED IF A TRACE IS PERFORMED.
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C
  600 IF (.NOT.TITRAC) GO TO 605
      IF (TSTRT-TIMBO) 602,602,605
  602 TSTRT=TSTRT+TRTI
      IF (ITCNTR.EQ.0) ITCNTR=ITCNTR+1
      IF (TSTPT-TSTRT) 604,630,630
  604 TITRAC=.FALSE.
  630 GO TO 600
C
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C   THE PRESENCE/ABSENCE OF ANY KEYED TRACES IS CHECKED
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C
  605 IF (.NOT.KTRACE) GO TO 698
C
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C   INSTRUCTION ADDRESS(PCOUNTER) TRACE KEYS ARE CHECKED
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C
      IF (.NOT.PCTRAC) GO TO 606
      DO 607 I=1,NPCTK
      IF((PCTK(I).EQ.INACCR).AND.(NIPCTK(I).GT.ITCNTR)) ITCNTR=NIPCTK(I)
  607 CONTINUE
C
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C   OP CODE TRACE KEYS ARE CHECKED
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C
  606 IF (.NOT.OCTRAC) GO TO 608
      DO 609 I=1,NOCTK
      IF((OCTK(I).EQ.ISEGTA(1)).AND.(NIOCTK(I).GT.ITCNTR)) ITCNTR=NIOCTK
     1(I)
  609 CONTINUE
C
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C   TIME TRACE KEYS ARE CHECKED
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C
  608 IF (.NOT.TTRACE) GO TO 610
      DO 611 I=1,NTTK
```

```
      IF (ITYPE-ITK(I))611,612,612
  612 LTTRAC(I)=.FALSE.
      F (NITTK(I).GT.ITCNTR) ITCNTR=NITTK(I)
  61  CNTINUE
C
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C   THE PRESENCE/ABSENCE CF REGISTER OR MEMORY TRACE KEYS IS
C   CHECKED ANC BRANCH TO APPROPRIATE LOGIC IS TAKEN.
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C
  610 IF((.NOT.RATRAC).ANC.(.NOT.MATRAC).AND.(.NOT.MOTRAC).AND.(.NCT.RCT
     1RAC)) GC TC 698
      K=ISEGTA(1)
      IJ=ICPCC(K)
      IF(IJ.LT.1) GC TC 698
C
      GC TC (613,613,613,613,614,614),IJ
C
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C   REGISTER OPERANC 1 ACCRESS KEY IS CHECKED
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C
  613 IF(.NCT.RATRAC) GC TO 615
      DC 616 I=1,NRATK
      IF((RATK(I).EC.ISEGTA(2)).ANC.(NIRATK(I).GT.ITCNTR)) ITCNTR=NIRATK
     1(I)
  616 CCNTINUE
C
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
    REGISTE[illegible]ERANC 1 KEY IS CHECKED
  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

  615 IF(.N[illegible]RAC) GC TC 617
      DO 618 [illegible],NROTK
      IF((RCT[illegible](I).EC.ISEGTA(2)).AND.(ROTK(I).EQ.ISPM(IMM)).AND.(NIROTK(
     1I).GT.ITCNTR)) ITCNTR=NIROTK(I)
  618 CCNTINUE
C
  617 GC TC (698,519,619,614,614,614),IJ
C
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C   REGISTER CPERANC 2 ACCRESS KEY IS CHECKED
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C
  619 IF (.NOT.RATRAC) GO TC 620
      CC 621 I=1,NRATK
      IF((RATK(I).EC.ISEGTA(3)).AND.(NIRATK(I).GT.ITCNTR)) ITCNTR=NIRATK
     1(I)
  621 CCNTINUE
C
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C   REGISTER OPERANC 2 KEY IS CHECKED
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C
  620 IF (.NCT.RCTRAC) GO TC 622
      CC 623 I=1,NRCTK
      IF((RCTKA(I).EC.ISEGTA(3)).ANC.(ROTK(I).EQ.ISPM(IMN)).AND.(NIROTK(
     1I).GT.ITCNTR)) ITCNTR=NIROTK(I)
  623 CCNTINUE
C
  622 CO TC(698,698,614,614,614,614),IJ
```

```
      IF((MTKA(I).EQ.IACR(IIII)).AND.(NIMATK(I).GT.ITCNTR)) ITCNTR=NIMAT
     1K(I)
  627 CONTINUE
C
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C   MEMORY OPERAND IIII KEY IS CHECKED
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C
  626 IF (.NOT.MOTRAC) GO TO 628
      DO 629 I=1,NMOTK
      IF((MOTKA(I).EQ.IACR(IIII)).AND.(MOTK(I).EQ.IOPRND(IIII)).AND.(NIM
     1OTK(I).GT.ITCNTR)) ITCNTR=NIMOTK(I)
  629 CONTINUE
C
  628 IIII=IIII+1
      IF ((IIII.EQ.2).AND.(IJ.EQ.6)) GO TO 625
C
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C   ITCNTR IS A COUNTER INDICATING THE CURRENT NUMBER OF SUCCESSIVE
C   INSTRUCTIONS TO BE TRACED.
C   IF ITCNTR=0, NO TRACE HAS BEEN REQUESTED.
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C
  698 IF (ITCNTR.LT.1) GO TO 699
      ITCNTR=ITCNTR-1
      WRITE(6,701)
      WRITE(6,704)
  704 FORMAT(1H0,15H TRACE REQUEST.)
      CALL HEADER
  697 WRITE(6,703)
  703 FORMAT(1H0,66H THE CURRENT (OCTAL) CONTENTS OF THE SPM REGISTERS A
     1RE AS FOLLOWS.)
      WRITE(6,758)
  758 FORMAT(1H0,12HSPM LOCATION,1X,14HOCTAL CONTENTS,3X,12HSPM LOCATION
     1,1X,14HOCTAL CONTENTS,3X,12HSPM LOCATION,1X,14HOCTAL CONTENTS,3X,1
     12HSPM LOCATION,1X,14HOCTAL CONTENTS)
      DO 759 I=1,29,4
      II=I-1
      III=I+1
      IIII=I+2
      IIIII=I+3
  759 WRITE(6,760)II,ISPM(I),I,ISPM(III),III,ISPM(IIII),IIII,ISPM(IIIII)
 760    FORMAT(1H0,3 (I3,O15,12X),I3,O15)
C
  699 RETURN
      END
$IBFTC SPMDRS  DECK
      SUBROUTINE SPMDRD
C
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C   DIMENSION SPM DUMP VARIABLES AND DEFINE COMMON
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C
      LOGICAL SPMD
      LOGICAL PCSPMD,CCSPMD,TSPMD,LTDK(9),MASPMD,RASPMD,MOSPMD,ROSPMD
      INTEGER PCDK(9),CCDK(9),MADK(9),RADK(9),
     1        MODKA(9),MODK(9),RODKA(9),RODK(9)
      REAL TDK(9)
C
      COMMON /SP/SPMD,PCSPMD,NPCDK,PCDK,OCSPMD,NOCDK,OCDK
     1           ,MASPMD,NMADK,MADK,RASPMD,NRADK,RADK,TSPMD,NTDK,TDK,LTDK
     1           ,MOSPMD,NMODK,MODKA,MODK,ROSPMD,NRODK,RODKA,RODK
```

```
C
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C      REQUIRED SPM-DUMP/NO-SPM-DUMP VARIABLE IS READ HERE.
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C
      READ 1001,SPMD
 1001 FORMAT(L5)
      IF(.NOT.SPMD) GO TO 2018
C
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C   CHECK FOR PRESENCE/ABSENCE OF PCOUNTER-KEYED SPM DUMP.
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C
      READ 1001,PCSPMD
      IF(.NOT.PCSPMD) GO TO 2019
      READ 1002,NPCCK
 1002 FORMAT(I5)
      READ 1013,(PCCK(I),I=1,NPCCK)
 1013 FORMAT(9I6)
C
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C     CHECK FOR PRESENCE/ABSENCE OF OP-CODE-KEYED SPM DUMP.
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C
 2019 READ 1001,OCSPMD
      IF(.NOT.OCSPMD) GO TO 2020
      READ 1002,NOCCK
      READ 1013,(OCCK(I),I=1,NOCCK)
C
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C     CHECK FOR PRESENCE/ABSENCE OF MEMORY-ADDRESS-KEYED SPM DUMP.
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C
 2020 READ 1001,MASPMD
      IF(.NOT.MASPMD) GO TO 2021
      READ 1002,NMACK
      READ 1013,(MACK(I),I=1,NMACK)
C
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C     CHECK FOR PRESENCE/ABSENCE OF REGISTER-ADDRESS-KEYED SPM DUMP.
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C
 2021 READ 1001,RASPMD
      IF(.NOT.RASPMD) GO TO 2022
      READ 1002,NRACK
      READ 1013,(RACK(I),I=1,NRACK)
C
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C     CHECK FOR PRESENCE/ABSENCE OF TIME-KEYED SPM DUMP.
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C
 2022 READ 1001,TSPMD
      IF(.NOT.TSPMD) GO TO 2023
      READ 1002,NTCK
      DO 400 I=1,NTCK
  400 LTCK(I)=.TRUE.
      READ 1014,(TCK(I),I=1,NTCK)
 1014 FORMAT(9F12.3)
C
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C     CHECK FOR PRESENCE/ABSENCE OF MEMORY-OPERAND-KEYED SPM DUMP.
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
```

```
C     CHECK FOR PRESENCE/ABSENCE OF REGISTER-OPERAND-KEYED SPM DUMP.
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C
 2024 READ 1001,RCSPMC
      IF(.NOT.RCSPMC) GO TO 2018
      READ 1002,NRODK
      READ 1015,(RODKA(I),RODK(I),I=1,NRODK)
C
 2018 RETURN
      END
$IBFTC SPMDSR  DECK
      SUBROUTINE SPMDEX
C
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C   DIMENSION SPM DUMP VARIABLES AND DEFINE COMMON
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C
      LOGICAL PCSPMC,OCSPMD,TSPMC,LTDK(9),MASPMD,RASPMD,MOSPMD,ROSPMD
      INTEGER PCDK(9),OCDK(9),MADK(9),RADK(9),
     1MODKA(9),MODK(9),RODKA(9),RODK(9)
      REAL TDK(9)
C
      COMMON /SP/SPMC,PCSPMC,NPCDK,PCDK,OCSPMD,NOCDK,OCDK
     1           ,MASPMC,NMADK,MADK,RASPMD,NRADK,RADK,TSPMD,NTDK,TDK,LTDK
     1           ,MCSPMC,NMODK,MODKA,MODK,ROSPMD,NRODK,RODKA,RODK
     1       /MM/FMMC,BLMMC,NBLDK,BPCDK,BDSTAD,NBDL
      COMMON/DVAR/ IOPCC(255),ITCNTR,IOFFST,IOPRND(3)
      COMMON/UNCON/IFWA,IHOST,ITARG,IOPCW,IBIT,IPARS,IDATAS,IHW,MAXCOR,
     1 LOCG,MPCS,TIMBC,NRCODE,NSTAT,LBPS1,LBPS2,NULL,IONE,IT,
     1CSLAS,ILC,INTCDE,ICOUNT,EXTIME,TIMLIM,
     1IMM,IMN,IERFLG,IRESTA,
     1LCBCUN,JIBCUN,MCNE,IFETC
     1,MAXNEG,MAXBIT
       COMMON /UARRAY/ ISEGTA(20),TIME(10),CLASS(10),INSDAT(256,5),
     1INSPAR(5,20),IPCTR(5),IPPCTR(5),ISGCD(5,20),IADR(3),ISPM(64),
     1IHALFW(6),ISTACK(5,6),IMAINM(4096)
      EQUIVALENCE (INACCR,ISPM(25))
C
      II=0
C
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C   INSTRUCTION ADDRESS(PCOUNTER) SPM DUMP KEYS ARE CHECKED
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C
      IF (.NOT.PCSPMC) GO TO 701
      DO 702 I=1,NPCDK
      IF (PCDK(I).EQ.INACCR) II=1
  702 CONTINUE
      IF (II.GT.0) GO TO 798
C
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C   OP CODE SPM DUMP KEYS ARE CHECKED
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C
  701 IF (.NOT.OCSPMC) GO TO 703
      DO 704 I=1,NOCDK
      IF (ISEGTA(I).EQ.OCDK(I)) II=1
  704 CONTINUE
      IF (II.GT.0) GO TO 798
C
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
```

```
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C
  70( IF (.NOT.TSPMD) GO TO 705
      DO 706 I=1,NTCK
      IF (.NOT.LTCK(I)) GO TO 706
      IF (TIMPC-TCK(I)) 706,707,707
  707 LTCK(I)=.FALSE.
      II=1
  706 CONTINUE
      IF (II.GT.0) GO TO 798
C
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C   THE PRESENCE/ABSENCE OF REGISTER OR MEMORY SPM DUMP KEYS IS
C   CHECKED AND BRANCH TO APPROPRIATE LOGIC IS TAKEN.
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C
  705 IF ((.NOT.MASPMD).AND.(.NOT.RASPMD).AND.(.NOT.MOSPMD).AND.(.NOT.RO
     1SPMD)) GO TO 799
      K=ISEGTA(1)
      IJ=ICPCC(K)
      IF (IJ.LT.1) GO TO 799
C
      GO TO (708,708,708,708,709,709),IJ
C
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C   REGISTER OPERAND 1 ADDRESS KEY IS CHECKED
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C
  708 IF (.NOT.RASPMD) GO TO 710
      DO 711 I=1,NRACK
      IF (RACK(I).EQ.ISEGTA(2)) II=1
  711 CONTINUE
      IF (II.GT.0) GO TO 798
C
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C   REGISTER OPERAND 1 KEY IS CHECKED
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C
  710 IF (.NOT.ROSPMD) GO TO 712
      DO 713 I=1,NROCK
      IF ((RODKA(I).EQ.ISEGTA(2)).AND.(RODK(I).EQ.ISPM(IMM))) II=1
  713 CONTINUE
      IF (II.GT.0) GO TO 798
C
  712 GO TO (799,714,714,709,709,709),IJ
C
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C   REGISTER OPERAND 2 ADDRESS KEY IS CHECKED
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C
  714 IF (.NOT.RASPMD) GO TO 715
      DO 716 I=1,NRACK
      IF (RADK(I).EQ.ISEGTA(3)) II=1
  716 CONTINUE
      IF (II.GT.0) GO TO 798
C
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C   REGISTER OPERAND 2 KEY IS CHECKED
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C
  715 IF (.NOT.ROSPMD) GO TO 717
      DO 718 I=1,NROCK
```

```
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C
  709 IIII=1
  723 IF (.NOT.MASPMC) GO TC 719
      DC 720 I=1,NMACK
      IF (MACK(I).EC.IACR(IIII)) II=1
  720 CCNTINUE
      IF (II.GT.0) GC TC 798
C
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C   MEMORY CPERAND IIII KEY IS CHECKED
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C
  719 IF (.NOT.MOSPMC) GO TC 722
      DC 721 I=1,NMCCK
      IF ((MOCKA(I).EC.IACR(IIII)).AND.(MOCK(I).EQ.IOPRND(IIII))) II=1
  721 CONTINUE
      IF (II.GT.0) GC TO 798
C
  722 IIII=IIII+1
      IF (IIII.GT.2) GC TC 799
      GC TC 723
C
  798 WRITE(6,750)
  750 FORMAT(1H1,51H ****************** DIAGNOSTICS ******************.)
      WRITE(6,751)
  751 FORMAT(1H0,18H SPM DUMP REQUEST.)
      CALL HEADER
      WRITE(6,758)
  758 FORMAT(1H0,12HSPM LOCATION,1X,14HOCTAL CONTENTS,3X,12HSPM LOCATION
     1,1X,14HCCTAL CCNTENTS,3X,12HSPM LOCATION,1X,14HOCTAL CONTENTS,3X,1
     12HSPM LCCATICN,1X,14HCCTAL CONTENTS)
      DC 759 I=1,61,4
      II=I-1
      III=I+1
      IIII=I+2
      IIIII=I+3
  759 WRITE(6,760)II,ISPM(I),I,ISPM(III),III,ISPM(IIII),IIII,ISPM(IIIII)
  760 FORMAT(1H0,8X,I3,4X,O11,12X,I3,4X,O11,12X,I3,4X,O11,12X,I3,4X,O11)
C
  799 RETURN
      END
$IBFTC MMDRSR  DECK
      SUBROUTINE MMCRC
C
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C   DIMENSION MM DUMP VARIABLES AND DEFINE COMMON
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C
      LOGICAL BLMMC
      LOGICAL FMMC
      INTEGER BPCCK(9),BCSTAC(9),NBCL(9)
C
      COMMON /SP/SPMC,PCSPMC,NPCCK,PCDK,OCSPMD,NCCOK,OCOK
     1          ,MASPMC,NMACK,MACK,RASPMC,NRACK,RACK,TSPMD,NTDK,TDK,LTDK
     1          ,MCSPMC,NMCCK,MOCKA,MOCK,ROSPMC,NRODK,RODKA,PODK
     1       /MM/FMMC,BLMMC,NBLCK,BPCCK,BOSTAD,NBDL
C
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C     THE BLOCK-MAIN-MEMORY DUMP/NO-BLOCK-MAIN-MEMORY DUMP VARIABLE
C     MUST BE IN THE DATA STREAM AND IS READ HERE. IF A BLOCK MM DUMP
```

```
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C
      READ 1001,BLMMC
 1001 FORMAT(L3)
      IF(.NOT.BLMMC) GO TO 2025
      READ 1002,NBLCK
 1002 FORMAT(I5)
      DO 1000 I=1,NBLCK
 1000 READ 1016,BPCCK(I),BCSTAC(I),NBCL(I)
 1016 FORMAT(I6,I6,I4)
C
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C     THE FULL-MAIN-MEMORY DUMP/NO-FULL-MAIN-MEMORY DUMP MUST BE IN THE
C     DATA STREAM AND IS READ HERE. A FULL MM DUMP IS AVAILABLE ONLY AT
C     THE TERMINATION OF THE PROGRAM.
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C
 2025 READ 1001,FMMC
C
      RETURN
      END
$IBFTC BMMCSR  DECK
      SUBROUTINE BLMDEX
C
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C   DIMENSION MM BLOCK DUMP VARIABLES AND DEFINE COMMON
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C
      INTEGER BPCCK(9),BCSTAC(9),NBDL(9)
C
      COMMON /SP/SPMC,PCSPMC,NPCCK,PCDK,OCSPMD,NCCDK,OCDK
     1          ,MASPMC,NMACK,MACK,RASPMD,NRADK,RADK,TSPMD,NTDK,TDK,LTDK
     1          ,MCSPMC,NMCCK,MCCKA,MOCK,ROSPMC,NRODK,RODKA,RODK
     1        /MM/FMMC,BLMMC,NBLCK,BPCCK,BCSTAC,NBDL
      COMMON/CVAR/ ICPCO(255),ITCNTR,ICFFST,IOPRNO(3)
      COMMON/UNCON/IFWA,IHOST,ITARG,IOPCW,IBIT,IPARS,IDATAS,IHW,MAXCOR,
     1 LCCG,MPCS,TIMEC,NRCODE,NSTAT,LBPS1,LBPS2,NULL,IONE,IT,
     1CSLAE,ILC,INTCDE,ICOUNT,EXTIME,TIMLIM,
     1IMM,IMN,IERFLG,IRESTA,
     1LCBCUN,JIBCUN,MONF,IFETC
     1,MAXNEG,MAXBIT
       COMMON /UARRAY/ ISEGTA(20),TIME(10),CLASS(10),INSDAT(256,5),
     1INSPAR(5,20),IPCTR(5),IPPCTR(5),ISGCD(5,20),IADR(3),ISPM(64),
     1IHALFW(6),ISTACK(5,6),IMAINM(4096)
       EQUIVALENCE (INADCR,ISPM(25))
C
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C   INSTRUCTION ADDRESS(PCOUNTER) MM BLOCK DUMP KEYS ARE CHECKED
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C
      DO 761 I=1,NBLCK
      IF (INADCR.NE.BPCCK(I)) GO TO 761
      WRITE(6,762)
  762 FORMAT(1H1,51H ***************** DIAGNOSTICS ***************** )
      WRITE(6,763)
  763 FORMAT(1H0,23H MM BLOCK DUMP REQUEST.)
      CALL HEADER
  768 WRITE(6,770)
  770 FORMAT(1H0,10X,12H MM LOCATION,7X,13H OCTAL CONTENTS)
      K=NBDL(I)
      DO 771 III=1,K
```

3427 RETURN
3428 END

APPENDIX V. SAMPLE PROGRAM OUTPUT

This appendix contains a sample program printout which illustrates the various types of diagnostics and statistics which the SUMC simulator makes available to the user. The target program used in obtaining the printout consisted of diagnostic routines which are currently being used for checkout of the SUMC Breadboard System instruction set.

The sample program includes the following types of diagnostic printouts:

- TRACE - A time-keyed TRACE diagnostic was requested such that a printout of the contents of key target computer registers is obtained at simulated elapsed time 50320.000 msec and the TRACE would remain in effect for ten consecutive instructions.
- SPM DUMP - An op-code-keyed scratch pad memory dump was requested such that any halfword operations encountered during the simulation would trigger a dump of the contents of SPM prior to the instruction execution. The current SUMC instruction set includes five instructions which specify halfword operations; those are LH, CH, AH, SH and MH and their op codes are (hexadecimal) 48, 49, 4A, 4B and 4C, respectively.
- SNAP - A memory-address-keyed SNAP diagnostic was requested such that any target instructions addressing MM location 32 or MM location 96 would trigger a printout of the contents of MM locations 96 and 100 or MM locations 32 and 36, respectively. In this case, SNAP locations were chosen so that both the old and new program status words could be checked when a Supervisor Call (SVC) instruction is executed.

- BLOCK MM DUMP - A block dump of the contents of MM locations 24 through 137 was requested whenever the current value of the program counter was equal to 6956. This particular memory dump would allow the user to check current values of program status words residing in main memory.

The standard statistics table is shown at the conclusion of the program printout and the following information is supplied concerning the target program which has been simulated:

- Number of instructions of various classes which were executed.
- Time used in executing each class of instruction.
- Percentage of total time used in executing each class of instruction.
- Total number of target instructions executed.
- Total simulated elapsed time.

```
$DATA
T
F
F
T
F
F
F
T
2
32    2
96    100
96    2
32     36
F
F
F
T
F
F
T
F
F
T
1
50320.000   10
F
F
F
F
T
F
T
5
72      73      74      75      76
F
F
F
F
F
T
1
6956      24 114
F

36 32
64       32758
         0        32767
         0
0
16
32
25

       800
000000            0008 1
```

INPUT DATA FOR SAMPLE PROGRAM

DIAGNOSTICS DATA

TARGET PARAMETERS

TARGET MEMORY MAP

2

EOF

****************** DIAGNOSTICS ******************

SPM DUMP REQUEST.

PROGRAM IS AT OFFSET 9970 FROM FIRST EXECUTABLE INSTRUCTION.

SIMULATED ELAPSED TIME IS 49850.000 MSEC.

THE CURRENT INSTRUCTION ADDRESS(PCOUNTER) IS 3404

THE CURRENT INSTRUCTION(IN OCTAL HALFWORDS) IS 045000 011202

REGISTER OPERAND 1 AT SPM ADDRESS 0 IS(OCTAL) 00000000000

MEMORY OPERAND 1 AT MM ADDRESS 646 IS(OCTAL) 37777600401

SPM LOCATION	OCTAL CONTENTS	SPM LOCATION	OCTAL CONTENTS	SPM LOCATION	OCTAL CONTENTS	SPM LOCATION	OCTAL CONTENTS
0	00000000000	1	00000000004	2	20100200400	3	20100200400
4	20100200400	5	20100200400	6	00100200401	7	37677577377
8	20100200400	9	20100200400	10	20000000001	11	20100200400
12	20000000001	13	00000000002	14	00000010004	15	00000006504
16	00000000000	17	00000000000	18	00000000000	19	00000000000
20	00000000000	21	00000000000	22	00000000000	23	00000000000
24	00000006514	25	00000000200	26	00000000000	27	00000000002
28	00000000000	29	00000000000	30	00000000000	31	00000000000
32	00000000000	33	00000000000	34	00000000000	35	00000000000
36	00000000000	37	00000000000	38	00000000000	39	00000000000
40	00000000000	41	00000000000	42	00000000000	43	00000000000
44	00000000000	45	00000000000	46	00000000000	47	00000000000
48	00000000000	49	00000000000	50	00000000000	51	00000000000
52	00000000000	53	00000000000	54	00000000000	55	00000000000
56	00000000000	57	00000000000	58	00000000000	59	00000000000
60	00000000000	61	00000000000	62	00000000000	63	00000000000

****************** DIAGNOSTICS ******************

SPM DUMP REQUEST.

PROGRAM IS AT OFFSET 9974 FROM FIRST EXECUTABLE INSTRUCTION.

SIMULATED ELAPSED TIME IS 49870.000 MSEC.

THE CURRENT INSTRUCTION ADDRESS(PCOUNTER) IS 3428

THE CURRENT INSTRUCTION(IN OCTAL HALFWORDS) IS 045000 011362

REGISTER OPERAND 1 AT SPM ADDRESS 0 IS(OCTAL) 37777777777

MEMORY OPERAND 1 AT MM ADDRESS 758 IS(OCTAL) 00000200000

SPM LOCATION	OCTAL CONTENTS	SPM LOCATION	OCTAL CONTENTS	SPM LOCATION	OCTAL CONTENTS	SPM LOCATION	OCTAL CONTENTS
0	37777777777	1	00000000004	2	20100200400	3	20100200400
4	20100200400	5	20100200400	6	00100200401	7	37677577377
8	20100200400	9	20100200400	10	20000000001	11	20100200400
12	20000000001	13	00000000502	14	00000010004	15	00000006504
16	00000000000	17	00000000000	18	00000000000	19	00000000000
20	00000000000	21	00000000000	22	00000000000	23	00000000000
24	00000006544	25	00000000230	26	00000000000	27	00000000000
28	00000000000	29	00000000000	30	00000000000	31	00000000000
32	00000000000	33	00000000000	34	00000000000	35	00000000000
36	00000000000	37	00000000000	38	00000000000	39	00000000000
40	00000000000	41	00000000000	42	00000000000	43	00000000000
44	00000000000	45	00000000000	46	00000000000	47	00000000000
48	00000000000	49	00000000000	50	00000000000	51	00000000000
52	00000000000	53	00000000000	54	00000000000	55	00000000000
56	00000000000	57	00000000000	58	00000000000	59	00000000000
60	00000000000	61	00000000000	62	00000000000	63	00000000000

```
****************** DIAGNOSTICS ******************

TRACE REQUEST.

PROGRAM IS AT OFFSET  10064 FROM FIRST EXECUTABLE INSTRUCTION.

SIMULATED ELAPSED TIME IS    50320.000 MSEC.

THE CURRENT INSTRUCTION ADDRESS(PCOUNTER) IS   3718

THE CURRENT INSTRUCTION(IN OCTAL HALFWORDS) IS 054060 011240

REGISTER OPERAND 1 AT SPM ADDRESS    3 IS(OCTAL) 20100200400

MEMORY OPERAND 1 AT MM ADDRESS    676 IS(OCTAL) 12525252525

THE CURRENT (OCTAL) CONTENTS OF THE SPM REGISTERS ARE AS FOLLOWS.
```

SPM LOCATION	OCTAL CONTENTS	SPM LOCATION	OCTAL CONTENTS	SPM LOCATION	OCTAL CONTENTS	SPM LOCATION	OCTAL CONTENTS
0	000000052525	1	000000000004	2	037777777702	3	020100200400
4	020100200400	5	020100200400	6	000100200401	7	037677577377
8	037777777776	9	020100200400	10	020000000001	11	020100200400
12	020000000001	13	000000000002	14	000000010004	15	000000007202
16	500000000000	17	500000000000	18	500000000003	19	500000000000
20	500000000000	21	500000000000	22	500000000000	23	500000000000
24	000000007206	25	000000000200	26	000000000000	27	000000000002
28	000000000000	29	500000000000	30	500000000000	31	500000000000

****************** DIAGNOSTICS ******************

TRACE REQUEST.

PROGRAM IS AT OFFSET 10065 FROM FIRST EXECUTABLE INSTRUCTION.

SIMULATED ELAPSED TIME IS 50325.000 MSEC.

THE CURRENT INSTRUCTION ADDRESS(PCOUNTER) IS 3722

THE CURRENT INSTRUCTION(IN OCTAL HALFWORDS) IS 056040 011240

REGISTER OPERAND 1 AT SPM ADDRESS 2 IS(OCTAL) 37777777702

MEMORY OPERAND 1 AT MM ADDRESS 676 IS(OCTAL) 12525252525

THE CURRENT (OCTAL) CONTENTS OF THE SPM REGISTERS ARE AS FOLLOWS.

SPM LOCATION	OCTAL CONTENTS	SPM LOCATION	OCTAL CONTENTS	SPM LOCATION	OCTAL CONTENTS	SPM LOCATION	OCTAL CONTENTS
0	000000052525	1	000000000004	2	037777777702	3	012525252525
4	020100200400	5	020100200400	6	000100200401	7	037677577377
8	037777777776	9	020100200400	10	020000000001	11	020100200400
12	020000000001	13	000000000002	14	000000010004	15	000000007202
16	500000000000	17	500000000000	18	500000000000	19	500000000000
20	500000000000	21	500000000000	22	500000000000	23	500000000000
24	000000007212	25	000000000200	26	000000000000	27	000000000002
28	000000000000	29	500000000000	30	500000000000	31	500000000000

****************** DIAGNOSTICS ******************

TRACE REQUEST.

PROGRAM IS AT OFFSET 1066 FROM FIRST EXECUTABLE INSTRUCTION.

SIMULATED ELAPSED TIME IS 50330.000 MSEC.

THE CURRENT INSTRUCTION ADDRESS(PCOUNTER) IS 3726

THE CURRENT INSTRUCTION(IN OCTAL HALFWORDS) IS 054440 011310

REGISTER OPERAND 1 AT SPM ADDRESS 2 IS(OCTAL) 03434343434

MEMORY OPERAND 1 AT MM ADDRESS 716 IS(OCTAL) 03434343434

THE CURRENT (OCTAL) CONTENTS OF THE SPM REGISTERS ARE AS FOLLOWS.

SPM LOCATION	OCTAL CONTENTS	SPM LOCATION	OCTAL CONTENTS	SPM LOCATION	OCTAL CONTENTS	SPM LOCATION	OCTAL CONTENTS
0	000000052525	1	000000000004	2	003434343434	3	007070707071
4	020100200400	5	020100200400	6	000100200401	7	037677577377
8	037777777776	9	020100200400	10	020000000001	11	020100200400
12	020000000001	13	000000000002	14	000000010004	15	000000007202
16	500000000000	17	500000000000	18	500000000000	19	500000000000
20	500000000000	21	500000000000	22	500000000000	23	500000000000
24	000000007216	25	000000000200	26	000000000000	27	000000000002
28	000000000000	29	500000000000	30	500000000000	31	500000000000

****************** DIAGNOSTICS ******************

TRACE REQUEST.

PROGRAM IS AT OFFSET 10067 FROM FIRST EXECUTABLE INSTRUCTION.

SIMULATED ELAPSED TIME IS 50335.000 MSEC.

THE CURRENT INSTRUCTION ADDRESS(PCOUNTER) IS 3730

THE CURRENT INSTRUCTION(IN OCTAL HALFWORDS) IS 043600 017222

MEMORY OPERAND 1 AT MM ADDRESS 3734 IS(OCTAL) 13114011314

THE CURRENT (OCTAL) CONTENTS OF THE SPM REGISTERS ARE AS FOLLOWS.

SPM LOCATION	OCTAL CONTENTS	SPM LOCATION	OCTAL CONTENTS	SPM LOCATION	OCTAL CONTENTS	SPM LOCATION	OCTAL CONTENTS
0	000000052525	1	000000000004	2	003434343434	3	007070707071
4	020100200400	5	020100200400	6	000100200401	7	037677577377
8	037777777776	9	020100200400	10	020000000001	11	020100200400
12	020000000001	13	000000000002	14	000000010004	15	000000007202
16	500000000000	17	500000000000	18	500000000000	19	500000000000
20	500000000000	21	500000000000	22	500000000000	23	500000000000
24	000000007222	25	000000000200	26	000000000000	27	000000000000
28	000000000000	29	500000000000	30	500000000000	31	500000000000

****************** DIAGNOSTICS ******************

TRACE REQUEST.

PROGRAM IS AT OFFSET 10068 FROM FIRST EXECUTABLE INSTRUCTION.

SIMULATED ELAPSED TIME IS 50340.000 MSEC.

THE CURRENT INSTRUCTION ADDRESS(PCOUNTER) IS 3738

THE CURRENT INSTRUCTION(IN OCTAL HALFWORDS) IS 054460 011314

REGISTER OPERAND 1 AT SPM ADDRESS 3 IS(OCTAL) 070707070711

MEMORY OPERAND 1 AT MM ADDRESS 720 IS(OCTAL) 07070707071

THE CURRENT (OCTAL) CONTENTS OF THE SPM REGISTERS ARE AS FOLLOWS.

SPM LOCATION	OCTAL CONTENTS	SPM LOCATION	OCTAL CONTENTS	SPM LOCATION	OCTAL CONTENTS	SPM LOCATION	OCTAL CONTENTS
0	000000052525	1	000000000004	2	003434343434	3	007070707071
4	020100200400	5	020100200400	6	000100200401	7	037677577377
8	037777777776	9	020100200400	10	020000000001	11	020100200400
12	020000000001	13	000000000002	14	000000010004	15	000000007202
16	500000000000	17	500000000000	18	500000000000	19	500000000000
20	500000000000	21	500000000000	22	500000000000	23	500000000000
24	000000007232	25	000000000200	26	000000000000	27	000000000000
28	000000000000	29	500000000000	30	500000000000	31	500000000000

****************** DIAGNOSTICS ******************

TRACE REQUEST.

PROGRAM IS AT OFFSET 10069 FROM FIRST EXECUTABLE INSTRUCTION.

SIMULATED ELAPSED TIME IS   50345.000 MSEC.

THE CURRENT INSTRUCTION ADDRESS(PCOUNTER) IS   3742

THE CURRENT INSTRUCTION(IN OCTAL HALFWORDS) IS 043600 017236

MEMORY OPERAND 1 AT MM ADDRESS   3746 IS(OCTAL) 13044011254

THE CURRENT (OCTAL) CONTENTS OF THE SPM REGISTERS ARE AS FOLLOWS.

SPM LOCATION	OCTAL CONTENTS	SPM LOCATION	OCTAL CONTENTS	SPM LOCATION	OCTAL CONTENTS	SPM LOCATION	OCTAL CONTENTS
0	000000052525	1	000000000004	2	003434343434	3	007070707071
4	020100200400	5	020100200400	6	000100200401	7	037677577377
8	037777777776	9	020100200400	10	020000000001	11	020100200400
12	020000000001	13	000000000002	14	000000010004	15	000000007202
16	500000000000	17	500000000000	18	500000000000	19	500000000000
20	500000000000	21	500000000000	22	500000000000	23	500000000000
24	000000007236	25	000000000200	26	000000000000	27	000000000000
28	000000000000	29	500000000000	30	500000000000	31	500000000000

```
****************** DIAGNOSTICS ******************

TRACE REQUEST.

PROGRAM IS AT OFFSET  10070 FROM FIRST EXECUTABLE INSTRUCTION.

SIMULATED ELAPSED TIME IS    50350.000 MSEC.

THE CURRENT INSTRUCTION ADDRESS(PCOUNTER) IS   3750

THE CURRENT INSTRUCTION(IN OCTAL HALFWORDS) IS 054220 011254

REGISTER OPERAND 1 AT SPM ADDRESS    9 IS(OCTAL) 20100200400

MEMORY OPERAND 1 AT MM ADDRESS    688 IS(OCTAL) 05252525252

THE CURRENT (OCTAL) CONTENTS OF THE SPM REGISTERS ARE AS FOLLOWS.
```

SPM LOCATION	OCTAL CONTENTS	SPM LOCATION	OCTAL CONTENTS	SPM LOCATION	OCTAL CONTENTS	SPM LOCATION	OCTAL CONTENTS
0	000000052525	1	000000000004	2	003434343434	3	007070707071
4	020100200400	5	020100200400	6	000100200401	7	037677577377
8	037777777776	9	020100200400	10	020000000001	11	020100200400
12	020000000001	13	000000000002	14	000000010004	15	000000007202
16	500000000000	17	500000000000	18	500000000000	19	500000000000
20	500000000000	21	500000000000	22	500000000000	23	500000000000
24	000000007246	25	000000000200	26	000000000000	27	000000000000
28	000000000000	29	500000000000	30	500000000000	31	500000000000

```
****************** DIAGNOSTICS ******************
TRACE REQUEST.
PROGRAM IS AT OFFSET  10071 FROM FIRST EXECUTABLE INSTRUCTION.
SIMULATED ELAPSED TIME IS    50355.000 MSEC.
THE CURRENT INSTRUCTION ADDRESS(PCOUNTER) IS   3754
THE CURRENT INSTRUCTION(IN OCTAL HALFWORDS) IS 056200 011344
REGISTER OPERAND 1 AT SPM ADDRESS     8 IS(OCTAL) 37777777776
MEMORY OPERAND 1 AT MM ADDRESS     744 IS(OCTAL) 25252525252
THE CURRENT (OCTAL) CONTENTS OF THE SPM REGISTERS ARE AS FOLLOWS.
```

SPM LOCATION	OCTAL CONTENTS	SPM LOCATION	OCTAL CONTENTS	SPM LOCATION	OCTAL CONTENTS	SPM LOCATION	OCTAL CONTENTS
0	000000052525	1	000000000004	2	003434343434	3	007070707071
4	020100200400	5	020100200400	6	000100200401	7	037677577377
8	037777777776	9	005252525252	10	020000000001	11	020100200400
12	020005000001	13	000000000002	14	000000010004	15	000000007202
16	500000000000	17	500000000000	18	500000000000	19	500000000000
20	500000000000	21	500000000000	22	500000000000	23	500000000000
24	000000007252	25	000000000200	26	000000000000	27	000000000000
28	000000000000	29	500000000000	30	500000000000	31	500000000000

****************** DIAGNOSTICS ******************

TRACE REQUEST.

PROGRAM IS AT OFFSET 10072 FROM FIRST EXECUTABLE INSTRUCTION.

SIMULATED ELAPSED TIME IS 50360.000 MSEC.

THE CURRENT INSTRUCTION ADDRESS(PCOUNTER) IS 3758

THE CURRENT INSTRUCTION(IN OCTAL HALFWORDS) IS 054600 011330

REGISTER OPERAND 1 AT SPM ADDRESS 8 IS(OCTAL) 36161616161

MEMORY OPERAND 1 AT MM ADDRESS 732 IS(OCTAL) 36161616161

THE CURRENT (OCTAL) CONTENTS OF THE SPM REGISTERS ARE AS FOLLOWS.

SPM LOCATION	OCTAL CONTENTS	SPM LOCATION	OCTAL CONTENTS	SPM LOCATION	OCTAL CONTENTS	SPM LOCATION	OCTAL CONTENTS
0	000000052525	1	000000000004	2	003434343434	3	007070707071
4	020100200400	5	020100200400	6	000100200401	7	037677577377
8	036161616161	9	034343434344	10	020000000001	11	020100200400
12	020000000001	13	000000000002	14	000000010004	15	000000007202
16	500000000000	17	500000000000	18	500000000000	19	500000000000
20	500000000000	21	500000000000	22	500000000000	23	500000000000
24	000000007256	25	000000000200	26	000000000000	27	000000000000
28	000000000000	29	500000000000	30	500000000000	31	500000000000

```
****************** DIAGNOSTICS ******************
TRACE REQUEST.
PROGRAM IS AT OFFSET  10073 FROM FIRST EXECUTABLE INSTRUCTION.
SIMULATED ELAPSED TIME IS    50365.000 MSEC.
THE CURRENT INSTRUCTION ADDRESS(PCOUNTER) IS   3762
THE CURRENT INSTRUCTION(IN OCTAL HALFWORDS) IS 043600 017262
MEMORY OPERAND 1 AT MM ADDRESS    3766 IS(OCTAL) 13144011304
THE CURRENT (OCTAL) CONTENTS OF THE SPM REGISTERS ARE AS FOLLOWS.
SPM LOCATION OCTAL CONTENTS   SPM LOCATION OCTAL CONTENTS   SPM LOCATION OCTAL CONTENTS   SPM LOCATION OCTAL CONTENTS
  0  000000052525          1  000000000004          2  003434343434          3  007070707071
  4  020100200400          5  020100200400          6  000100200401          7  037677577377
  8  036161616161          9  034343434344         10  020000000001         11  020100200400
 12  020000000001         13  000000000002         14  000000010004         15  000000007202
 16  500000000000         17  500000000000         18  500000000000         19  500000000000
 20  500000000000         21  500000000000         22  500000000000         23  500000000000
 24  000000007262         25  000000000200         26  000000000000         27  000000000000
 28  000000000000         29  500000000000         30  500000000000         31  500000000000
       INSER2
```

******************* DIAGNOSTICS *******************

SPM DUMP REQUEST.

PROGRAM IS AT OFFSET 10251 FROM FIRST EXECUTABLE INSTRUCTION.

SIMULATED ELAPSED TIME IS 51255.000 MSEC.

THE CURRENT INSTRUCTION ADDRESS(PCOUNTER) IS 5694

THE CURRENT INSTRUCTION(IN OCTAL HALFWORDS) IS 045400 011150

REGISTER OPERAND 1 AT SPM ADDRESS 0 IS(OCTAL) 00000000001

MEMORY OPERAND 1 AT MM ADDRESS 620 IS(OCTAL) 37777722645

SPM LOCATION	OCTAL CONTENTS	SPM LOCATION	OCTAL CONTENTS	SPM LOCATION	OCTAL CONTENTS	SPM LOCATION	OCTAL CONTENTS
0	00000000001	1	00000000004	2	00000000000	3	20000000000
4	00000000000	5	00000000000	6	00000000000	7	37677577377
8	36161616161	9	34343434344	10	20000000001	11	20100200400
12	37777777777	13	00000000003	14	00000010004	15	02000013066
16	00000000000	17	00000000000	18	00000000000	19	00000000000
20	00000000000	21	00000000000	22	00000000000	23	00000000000
24	00000013076	25	00000000200	26	00000000000	27	00000000002
28	00000000000	29	00000000000	30	00000000000	31	00000000000
32	00000000000	33	00000000000	34	00000000000	35	00000000000
36	00000000000	37	00000000000	38	00000000000	39	00000000000
40	00000000000	41	00000000000	42	00000000000	43	00000000000
44	00000000000	45	00000000000	46	00000000000	47	00000000000
48	00000000000	49	00000000000	50	00000000000	51	00000000000
52	00000000000	53	00000000000	54	00000000000	55	00000000000
56	00000000000	57	00000000000	58	00000000000	59	00000000000
60	00000000000	61	00000000000	62	00000000000	63	00000000000

```
****************** DIAGNOSTICS ******************
SPM DUMP REQUEST.
PROGRAM IS AT OFFSET  10256 FROM FIRST EXECUTABLE INSTRUCTION.
SIMULATED ELAPSED TIME IS    51280.000 MSEC.
THE CURRENT INSTRUCTION ADDRESS(PCOUNTER) IS   5722
THE CURRENT INSTRUCTION(IN OCTAL HALFWORDS) IS 045440 011156
REGISTER OPERAND 1 AT SPM ADDRESS    2 IS(OCTAL) 37777722645
MEMORY OPERAND 1 AT MM ADDRESS     626 IS(OCTAL) 24551255132
```

SPM LOCATION	OCTAL CONTENTS	SPM LOCATION	OCTAL CONTENTS	SPM LOCATION	OCTAL CONTENTS	SPM LOCATION	OCTAL CONTENTS
0	00000000002	1	00000000004	2	37777722645	3	20000000000
4	00000000000	5	00000000000	6	00000000000	7	37677577377
8	36161616161	9	34343434344	10	20000000001	11	20100200400
12	37777777777	13	00000000003	14	00000010004	15	02000013066
16	00000000000	17	00000000000	18	00000000000	19	00000000000
20	00000000000	21	00000000000	22	00000000000	23	00000000000
24	00000013132	25	00000000200	26	00000000000	27	00000000000
28	00000000000	29	50000000000	30	00000000000	31	00000000000
32	00000000000	33	00000000000	34	00000000000	35	00000000000
36	00000000000	37	00000000000	38	00000000000	39	00000000000
40	00000000000	41	00000000000	42	00000000000	43	00000000000
44	00000000000	45	00000000000	46	00000000000	47	00000000000
48	00000000000	49	00000000000	50	00000000000	51	00000000000
52	00000000000	53	00000000000	54	00000000000	55	00000000000
56	00000000000	57	00000000000	58	00000000000	59	00000000000
60	00000000000	61	00000000000	62	00000000000	63	00000000000

INSER2

```
*******************DIAGNOSTICS******************
SNAP REQUEST.
TRIGGERED BY MEMORY ADDRESS      96
PROGRAM IS AT OFFSET 228936 FROM FIRST EXECUTABLE INSTRUCTION.
SIMULATED ELAPSED TIME IS  1144930.000 MSEC.
THE CURRENT INSTRUCTION ADDRESS(PCOUNTER) IS   6852
THE CURRENT INSTRUCTION(IN OCTAL HALFWORDS) IS 050000 010134
REGISTER OPERAND 1 AT SPM ADDRESS     0 IS(OCTAL) 00000000000
MEMORY OPERAND 1 AT MM ADDRESS      96 IS(OCTAL) 20000000000
THE SNAPPED LOCATIONS AND THEIR (OCTAL) CONTENTS ARE AS FOLLOWS.
     MM LOCATION        OCTAL CONTENTS
          32               20000005514
          36               00000000000
     INSER2
```

```
*******************DIAGNOSTICS*******************

SNAP REQUEST.

TRIGGERED BY MEMORY ADDRESS      32

PROGRAM IS AT OFFSET 228990 FROM FIRST EXECUTABLE INSTRUCTION.

SIMULATED ELAPSED TIME IS  1144950.000 MSEC.

THE CURRENT INSTRUCTION ADDRESS(PCOUNTER) IS   6904

THE CURRENT INSTRUCTION(IN OCTAL HALFWORDS) IS 054000 010034

REGISTER OPERAND 1 AT SPM ADDRESS     0 IS(OCTAL) 20000015364

MEMORY OPERAND 1 AT MM ADDRESS      32 IS(OCTAL) 00000000125

THE SNAPPED LOCATIONS AND THEIR (OCTAL) CONTENTS ARE AS FOLLOWS.

    MM LOCATION       OCTAL CONTENTS

         96             00000000000

        100             00000000000
```

```
********************DIAGNOSTICS********************

SNAP REQUEST.

TRIGGERED BY MEMORY ADDRESS     96

PROGRAM IS AT OFFSET 228999 FROM FIRST EXECUTABLE INSTRUCTION.

SIMULATED ELAPSED TIME IS  1144995.000 MSEC.

THE CURRENT INSTRUCTION ADDRESS(PCOUNTER) IS   6946

THE CURRENT INSTRUCTION(IN OCTAL HALFWORDS) IS 050000 010134

REGISTER OPERAND 1 AT SPM ADDRESS    0 IS(OCTAL) 20000000000

MEMORY OPERAND 1 AT MM ADDRESS      96 IS(OCTAL) 00000000000

THE SNAPPED LOCATIONS AND THEIR (OCTAL) CONTENTS ARE AS FOLLOWS.

    MM LOCATION        OCTAL CONTENTS

         32               20000005514

         36               00000000000
```

****************** DIAGNOSTICS ******************

MM BLOCK DUMP REQUEST.

PROGRAM IS AT OFFSET 229002 FROM FIRST EXECUTABLE INSTRUCTION.

SIMULATED ELAPSED TIME IS 1145010.000 MSEC.

THE CURRENT INSTRUCTION ADDRESS(PCOUNTER) IS 6956

THE CURRENT INSTRUCTION(IN OCTAL HALFWORDS) IS 005252 010140

MM LOCATION	OCTAL CONTENTS
24	20000004574
25	20000000000
26	20000015460
27	20000000000
28	20000015204
29	20000000000
30	20000004574
31	20000000000
32	20000005514
33	00000000000
34	00000000000
35	00000000000
36	00000000000
37	00000000000
38	00000000000
39	00000000000
40	00000000000
41	00000000000
42	00000000000
43	00000000000
44	00000000000
45	00000000000

50	00000000001
51	00000000000
52	00000000000
53	00000000000
54	00000000000
55	00000000000
56	00000000000
57	00000000000
58	00000000000
59	00000000000
60	00000000000
61	00000000000
62	00000000000
63	00000000000
64	00000000000
65	00000000000
66	00000000000
67	00000000000
68	00000000000
69	00000000000
70	00000000000
71	00000000000
72	00000000000
73	00000000000
74	00000000000
75	00000000000
76	00000000000
77	00000000000
78	00000000000
79	00000000000

81	00000000000
82	00000000000
83	00000000000
84	00000000000
85	00000000000
86	00000000000
87	00000000000
88	00000000000
89	00000000000
90	00000000000
91	00000000000
92	00000000000
93	00000000000
94	00000000000
95	00000000000
96	00000000000
97	00000000000
98	00000000000
99	00000000000
100	00000000000
101	00000000000
102	00000000004
103	25000400000
104	25000001450
105	25000000000
106	04000001514
107	00000000000
108	00000000004
109	00000000010
110	00000000014

	116	00000000044
	117	00000000050
	118	00000000054
	119	00000000060
	120	00000000064
	121	00000000070
	122	00000000074
	123	00000000000
	124	00000000004
	125	00000000010
	126	00000000014
	127	00000000020
	128	00000000024
	129	00000000030
	130	00000000034
	131	00000000040
	132	00000000044
	133	00000000050
	134	00000000054
	135	00000000060
	136	00000000064
	137	00000000070
INSER2		

```
********************DIAGNOSTICS********************

SNAP REQUEST.

TRIGGERED BY MEMORY ADDRESS      32

PROGRAM IS AT OFFSET 229003 FROM FIRST EXECUTABLE INSTRUCTION.

SIMULATED ELAPSED TIME IS  1145015.000 MSEC.

THE CURRENT INSTRUCTION ADDRESS(PCOUNTER) IS    6964

THE CURRENT INSTRUCTION(IN OCTAL HALFWORDS) IS 054000 010034

REGISTER OPERAND 1 AT SPM ADDRESS     0 IS(OCTAL) 21000015460

MEMORY OPERAND 1 AT MM ADDRESS      32 IS(OCTAL) 37700000252

THE SNAPPED LOCATIONS AND THEIR (OCTAL) CONTENTS ARE AS FOLLOWS.

   MM LOCATION        OCTAL CONTENTS

        96              00000010000

       100              00000000000
   DONE       15        00000001754
  INSTRUCTION CLASS DISTRIBUTION

   CLASS          COUNT              TIME         PERCENT
     1           2969.0           29890.0            1.3
     2          46091.0          460910.0           20.1
     3             17.0             170.0            0.0
     4           1240.0           12400.0            0.5
     5         178658.0         1786580.0           78.0
     6             32.0             320.0            0.0
     7             -0.0              -0.0           -0.0
     8             -0.0              -0.0           -0.0
     9             12.0             120.0            0.0
    10              2.0              20.0            0.0
               229041.0         1145205.0          100.0
```

NASA—MSFC

CSC

COMPUTER SCIENCES CORPORATION

Major Offices and Facilities Throughout the World