MIT/LCS/TM-30

# SIM360:
# A S/360 SIMULATOR

Wm. Arthur Mc Cray

May 1972

# SIM360: A S/360 SIMULATOR

Wm. Arthur Mc Cray

MAC Technical Memorandum 30

May 1972

This research was supported by the Advanced Research Projects Agency of the Department of Defense under ARPA Order No. 2095, and was monitored by ONR under Contract No. N00014-70-A-0362-0006.

Massachusetts Institute of Technology

PROJECT MAC

Cambridge Massachusetts 02139

# SIM360: A S/360 SIMULATION

by

WM. ARTHUR Mc CRAY

Submitted to the Department of Mechanical Engineering on May 12, 1972 in partial fulfillment of the requirements for the degree of Bachelor of Science.

## ABSTRACT

Modern, large-scale computer systems typically operate under the control of an operating system or executive program, and reserve for the exclusive use of the operating system a set of privileged instructions, which the normal users may not issue. This very necessary arrangement produces a problem of equipment availability for those who wish to develop or investigate operating systems programs, because such programs cannot be run as normal user jobs under an executive program.

This thesis describes SIM360, a detailed simulator of a representative IBM S/360 computer, which was written to run student programs, programs assigned as machine problems for a course in operating systems. The simulator allows programs to issue all of the priveleged instructions of the S/360, and thus provides a readily available tool for the study of operating systems programs.

Thesis Supervisor: John J. Donovan
Title: Professor of Electrical Engineering

ACKNOWLEDGMENTS

The author wishes to express sincere appreciation to Professor John Donovan and Mr. Stuart Madnick for their suggestions, guidance, patience and encouragement, and particularly for the fact that the writing of SIM360 has been a rewarding and truly educational experience.

The students of 6.802, Advanced Operating Systems, who exhibited remarkable patience and a gentle insistence on absolute accuracy, are due special thanks for their helpful criticisms and faithful reporting of bugs.

Finally, to my incredibly patient and loving wife, typist and helpmeet, I offer my profound gratitude for her constant and irresistable encouragement and support.

TABLE OF CONTENTS

## LIST OF FIGURES

# 1. INTRODUCTION

A simulation of a system is normally undertaken to provide a manipulatable model of the system for investigation and study. In some cases the system being simulated may not exist, or may be in a development stage, and thus is unavailable for use. This would be the case with a proposed mass transit system, for example, where the capabilities and performance of the system must be carefully evaluated before committing perhaps millions of dollars for development. Another, and very frequent use of simulators in this respect, is to provide the ability to develop the hardware and software of a new computer system in parallel. A simulator of the computer system, written to operate on existing computer hardware, is used to develop and debug the software for the computer before a working prototype is completed, and in this way a large savings in total system development time can be realized.

In other cases the simulated system may exist, but for some reason be difficult or impossible to use for experimentation. One cannot, in practice, block a traffic artery in a major city to study the resulting flow of traffic, or vary the mass of the moon to study the effect on the tide. In much the same sense, a simulation of an existing computer system can provide an important tool for research, develop-

ment, and teaching. Modern, large-scale computer systems operate under the control of an operating system or executive program, and place definite restrictions on the operations which may be performed by programs run on the system. Typically, user programs may not use instructions which directly affect input/output devices, protection mechanisms, the interrupt structure, and other basic aspects of the processor state. Because the operating system provides user programs with indirect methods of performing operations with privileged facilities, most programs can be run; however, operating system programs, that is, complete programs which may issue any instruction implemented by the computer, programs which in fact may be intended to provide the indirect methods for performing privileged operations, are excluded. For this large and important class of programs, then, the computer system is unavailable[1] for testing or development. A simulator of the computer system provides a solution to this basic problem, and offers other substantial advantages as well.

---

[1] In a relative sense. Manufacturers' personnel and software support staff members at large installations may have access to a "bare-bones" system on a limited basis. Most users, even systems programmers, never have this opportunity on a large scale system, for obvious reasons of efficiency and economy.

A simulator is not the only solution to this problem; it is, however, frequently the only practical one. The obvious approach, somehow to obtain the desired computer for exclusive use, has been mentioned, and is clearly inconvenient, impractical, expensive, and not necessarily sufficiently useful when it is possible at all. Most system programmers have encountered that maddening class of program errors which exist, are perhaps regularly repeatable, but which do not occur when the CPU is stepped through the erroneous code one instruction at a time. Similar timing dependencies may exist in input-output operations of interest. Finally, the computer may not exist in the desired configuration, if some particular feature or device is desired for study.

Another method of running operating system programs involves the use of a virtual machine, such as IBM's CP-67[1] provides. The primary drawback in this approach is the requirement that a very expensive and infrequently available S/360 model 67 is required. In addition, the virtual machine does not accurately reflect the timing and behavior of the simulated computer in the area of I/O

---

[1] Control Program-67/Cambridge Monitor System User's Guide. IBM Publication.

operations and privileged instructions. This is a fairly serious drawback, since this area is the focus of interest in operating systems programs.

A simulator, in contrast, offers the advantages summarized below.

-- Readily available to users
-- Run complete programs
-- Achieve any level of accuracy desired
-- Incorporates comprehensive debugging aids
-- Allows detailed performance monitoring
-- Arbitrary configuration - size, features, and devices
-- May be optimized for solution of problem(s) of interest
-- May be readily modified - software program

A complete discussion of these points is postponed to the following section, where they are covered in depth as features of SIM360. A simulator incorporating most or all of these features is potentially useful for:

-- Software development
-- Teaching tool - student runs
-- System testing of new versions of operating system software
-- Evaluation of different system configurations

-- Evaluation of new hardware

Software development is probably the most frequently occurring reason for using a computer simulator. Most development programs for new computer systems involve the early implementation of a simulator for the reasons discussed above.

SIM360 was specifically written for use as a teaching tool in a course in advanced operating systems, and has been successfully used for two machine problems (to date) in the current academic semester. The checkout of a new version of an operating system, or some component of it, could be accomplished on a simulator without the necessity for interrupting normal operations, bringing down the current system, installing the new version, running the desired tests, bringing down the new system, reinstalling the old system, etc., etc., through many iterations of the testing procedure. With some modification, perhaps, a simulator could be used to evaluate the effects and operating characteristics of totally new hardware in the form of new devices, a more powerful system, or perhaps a completely new system (transition from a S/360 to a S/370, for example). Simulators have not been widely used in these last three areas, but because it is a uniquely complete and accurate simulation, SIM360 could be a powerful and useful tool for systems work of this type.

## 2. DESCRIPTION OF SIM360

SIM360 is a computer program written in PL/I which simulates to a high degree of accuracy the behavior of a representative member of the IBM S/360 series of computer systems. The simulator runs as a problem program under OS/360 (or other operating system which supports PL/I), implements the full complement of privileged instructions, and provides very detailed and accurate simulation of the basic I/O devices of the S/360. It is specifically designed to run student programs assigned as machine problems for a course in operating systems, but provides a general solution to the problem of computer system availability discussed above.

### 2.1 ADVANTAGES AND FEATURES

All of the advantages of a simulator listed in Section 1 are incorporated in some measure in SIM360. It is potentially readily available to any user of the computer system on which it is in use[1], and could be made available on any

---

[1] The IBM S/370-155 at MIT's Information Processing Center

S/360 or S/370 which supports the IBM S/360 operating system and can provide a 200K user partition. Further, it could be made available on any comparable large scale computer system which supports PL/I, with appropriate, but probably minor, modifications to the simulator code. In one sense, SIM360 can run complete programs; it implements all of the I/O and privileged instructions of the S/360. The complete instruction set is not implemented, but was not desired; the simulator is specifically designed to run student assignments emphasizing I/O programming, interrupt handling, and other operating system techniques. An instruction subset adequate for this purpose is provided.[1]

The level of accuracy of the simulation is as high as could be reached using available documentation. Instruction timings, for example, for those instructions which have variable length operands, are adjusted to reflect the length specified in the particular instruction being simulated. I/O operations which result in data transfers by the data channels on a cycle stealing basis are accurately reflected.[2] Interrupt timings are adjusted to account for device characteristics such as clutch points (on card

---

[1] See Appendix A.

[2] See Appendix C.

readers and card punches) and line spacing (on printers).

SIM360 incorporates powerful and comprehensive debugging aids. A program being executed by the simulator may dynamically request diagnostic or program flow information to be printed by the simulator on the basis of a number of distinct conditions:

1) Successful branch
2) Reference to a particular address as an operand
3) Reference to a particular address for instruction execution
4) Execution of a particular instruction (by class, i.e., Load or Multiply, not instance)
5) Occurrence of an interrupt
6) Occurrence of significant channel activity
7) Occurrence of a dump request

All of these conditions may be dynamically set and reset by the simulated program through the use of supplied macro instructions.[1] Because the simulator, as implemented, is not an interactive system, there are no breakpoint facilities, or other very useful capabilities usually found in interactive debugging aids. Such features could, however, be easily added to the simulator should it ever be desirable

---

[1] See Appendix A

to use it in an interactive environment. The capabilities provided for debugging also serve for performance monitoring. All aspects of system performance may be selectively examined through use of the features outlined.

SIM360 is specifically adapted for the use for which it was written. It is being used to run student programs for a relatively large class, and has many features which are desirable for this use. For example, most of the options discussed in Appendix B are provided to give the instructor a measure of control over how much machine time and output volume may be generated by student programs. To a limited extent, the options also reflect the ability to choose an arbitrary configuration for the system being simulated, but, in general, achieving a truly arbitrary configuration is a matter which requires modification of the simulator code.

## 2.2 CONFIGURATION OF THE SIMULATED SYSTEM

The simulated computer (see Figure 1) which SIM360 provides is a representative IBM S/360 with up to 32K bytes of core storage.[1] Up to six channels are available, although at present only one, the byte multiplexor channel, has attached devices. Two 2821 control units are attached to multiplexor channel 0, and each 2821 services a 2540 card

[1] This ridiculously small amount of core (for a S/360) is considerably more than adequate for student programs.

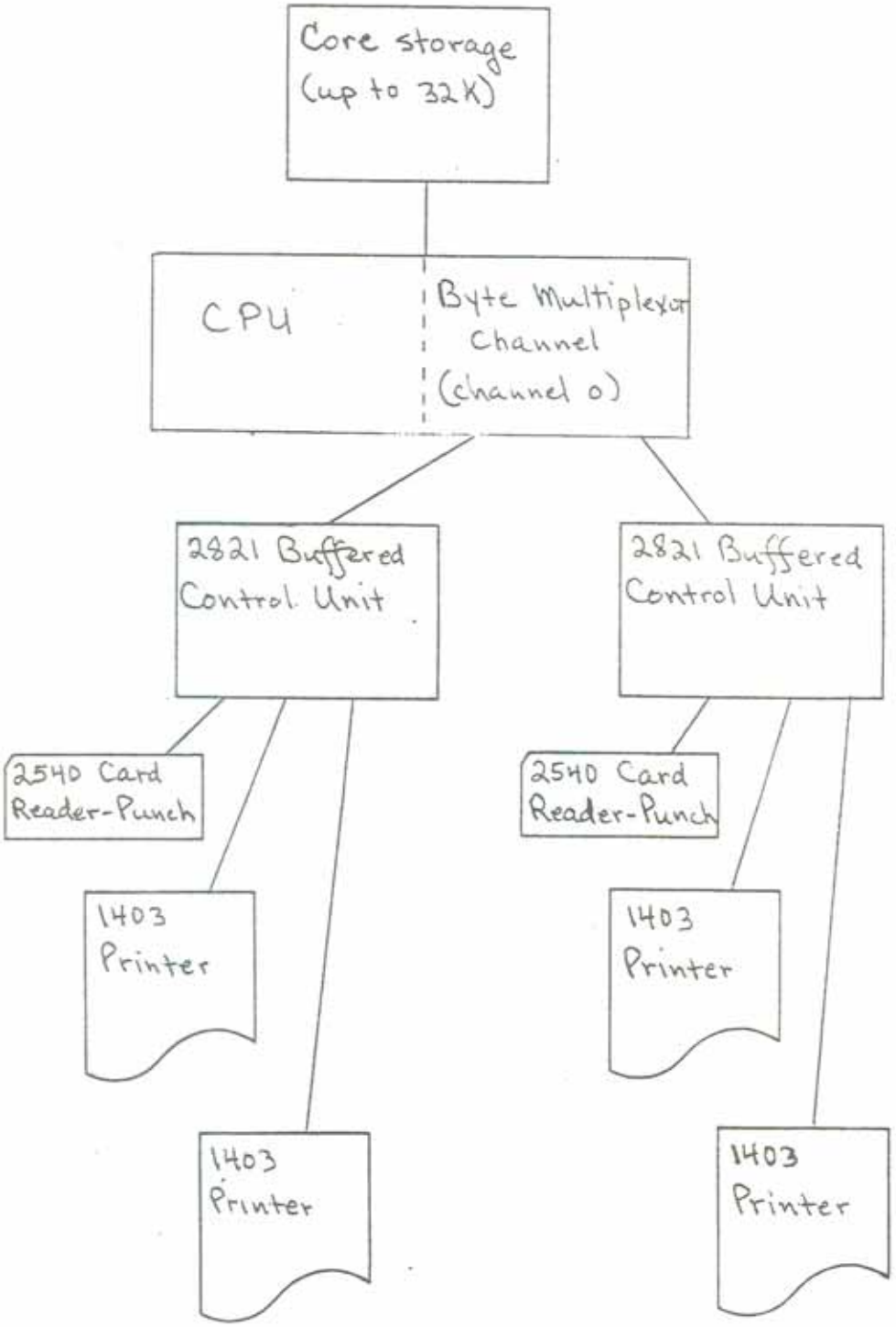


Figure 1 : Simulated Hardware Configuration

reader-punch and two 1403 printers. No special features are implemented on the CPU or any device. Certain aspects of the CPU are not simulated. The machine check interrupt and diagnostic scan-out are not available, nor is the operator's console.[1] At present, no direct access or tape devices are available, but direct access capability for 2311 and 2314 type devices are under development and will be available in the near future. Tape facilities are a possible, but not imminent, addition. This configuration provides the ability to run systems programs which deal with every phase of S/360 CPU operation except error detection, and with card and printer I/O devices.

## 2.3 STRUCTURE OF THE SIMULATOR

SIM360 is a complete system for running student programs. As an overview of the simulator structure, a brief description of how the simulator is utilized will be given; complete,detailed instructions for using the simulator are given in Appendices A and B.

Students are assigned a problem and prepare their programmed solutions in S/360 Basic Assembler Language.

---

[1] The operators console might be a useful addition to an interactive version of SIM360.

Student decks are collected and grouped into one large input deck; appropriate control cards are added for the operating system and SIM360. The entire assembled deck is submitted as one batch run. When returned, student decks are reseparated, and printed output of assembler listing, simulator output, and simulator trace and diagnostic output are assembled for each student and returned. Final results after a series of runs are submitted by the student for grading.

The simulator produces this overall result by first assembling all of the student programs using the G level assembler[1] in batch mode. Object module output of the assembler is held in a temporary file, which is the input to the second (simulation) pass over the data. When all student decks are assembled, the simulator proper is given control.

The simulator consists of four major modules.[2] The first module, a very simple loader, reads the object module output from the first (only) student deck and builds an executable program in a reserved storage area. This activity is entirely analagous to that of the S/360 Loader[3].

---

1 This is a more efficient S/360 Assembler written at the University of Waterloo.

2 See Figure 2.

3 IBM System/360 Operating System : Linkage Editor and Loader, Form GC28-6538.

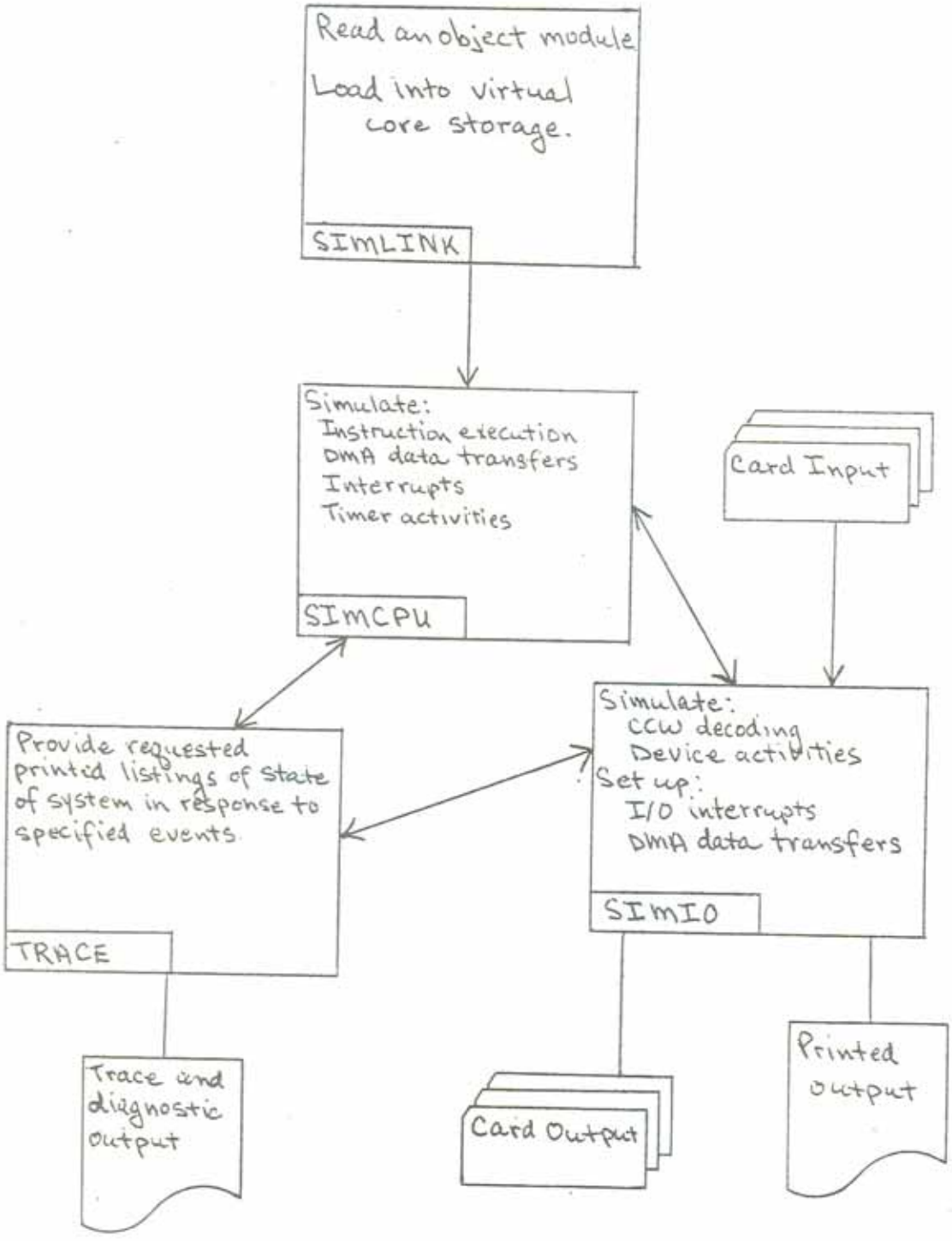


Figure 2 : Simulator Structure and Data Flow

When the program has been loaded, the CPU simulation module initiates system activity in a manner analagous to S/360 Initial Program Loading. Thereafter, under the control of the CPU simulator, the I/O simulation module and the trace module are invoked as required by the program, and the simulated execution of the student program procedes until it terminates, or until an unrecoverable error is detected. Control then returns to the module SIMLINK, which loads the next student program and reinitializes the simulation process.

## 2.4 PROGRAM OPERATION

The simulator receives control from the operating system in the module SIMLINK. First the parameters of the run are processed,[1] and then the first (or only) assembled student program is loaded. When the program is loaded, the module SIMCPU is called to simulate program execution. When SIMCPU returns, SIMLINK loads the next program and continues in this manner until all programs have been simulated.

The module SIMCPU performs some initialization, and proceeds to simulate the execution of the program by using the doubleword at simulated location zero as the initial

---

[1] See Appendix B, section B.3.1 and Appendix C, section C.2.1.

program status word. Each instruction is simulated by a small routine (typically four or five PL/I statements) which does appropriate processing to implement the instruction. After each instruction the elapsed time in the simulation is updated, and a check for an interrupt or other special condition is made. Interrupts may occur because the timer decrements from zero to minus one, or because an appropriate condition exists in the I/O subsystem. Other conditions which are handled are data transfers between core storage and I/O devices, I/O events,[1] and the special considerations which arise when the CPU is in the wait state.

If in the course of instruction simulation the simulator encounters a request that a trace condition be enabled (or disabled), the TRACE module is called at an appropriate entry point. This module checks and decodes the trace request, makes (or deletes) an appropriate entry in the list of enabled trace conditions, and, if necessary, prints any requested trace information. The other class of events which causes the TRACE module to be called is the occurrence of a condition which is currently being traced. In this case an appropriate entry point in TRACE is called to format and print the information requested by the enabled

---

[1] See Appendix C, section C.4.6.

trace condition.

When SIMCPU encounters an I/O instruction, or when an I/O event occurs, the module SIMIO is called. Different entry points are used for different functions. The simulation of the HALT I/O and TEST I/O instructions involves little more than examining the state of the addressed channel, subchannel, and device, and setting the CSW and the condition code to appropriate values. The HALT I/O routine may also involve the scheduling and rescheduling of interrupts. The START I/O instruction, on the other hand, frequently initiates a long and very complex chain of events. In a very simple case[1] the following outline lists major activities in their order of occurrence.

- Fetch the CAW from core and validate.
- Fetch the CCW from core, decode and validate.
- Call a routine which implements the specific device involved in the operation.
- Validate the specific command to the device.
- Schedule a device end interrupt to occur after completion of mechanical activity.

---

[1] For example, the Read, Feed, and Stacker Select command to the 2540 card reader, discussed in Appendix C, section C.4.7.

- Set up data transfers between core storage and the device, to occur at appropriate intervals over an extended period of time.

- Set up conditions and parameters associated with the end of data transfer (e.g. channel end interrupt or event).

- Return to SIMCPU.

In addition to the simulation of I/O instructions, SIMIO performs I/O event processing, and initialization for and termination of I/O simulation at separate entry points. Figure 3 shows an overview of the operation of SIM360, and may help to clarify the foregoing discussion.

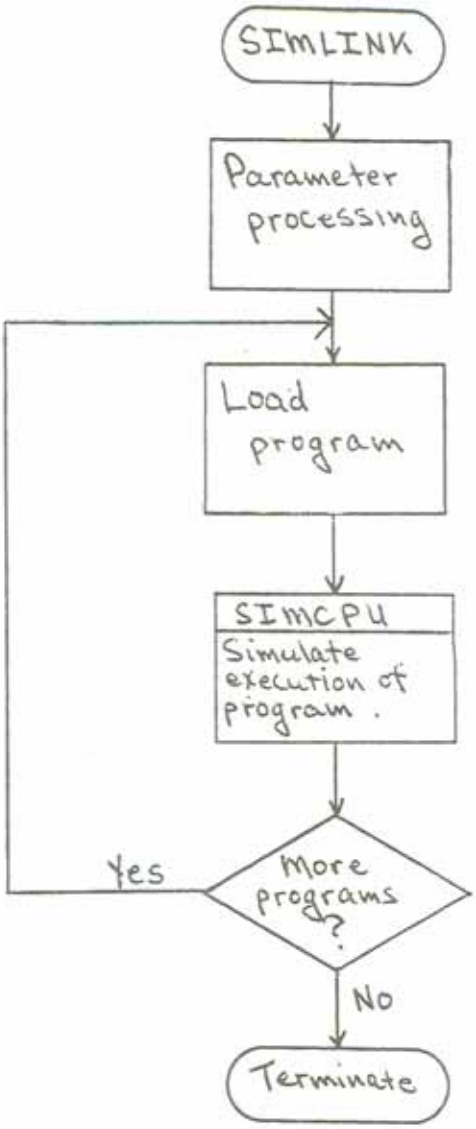


Figure 3 : Simulator Operation Overview
(continued on next page)

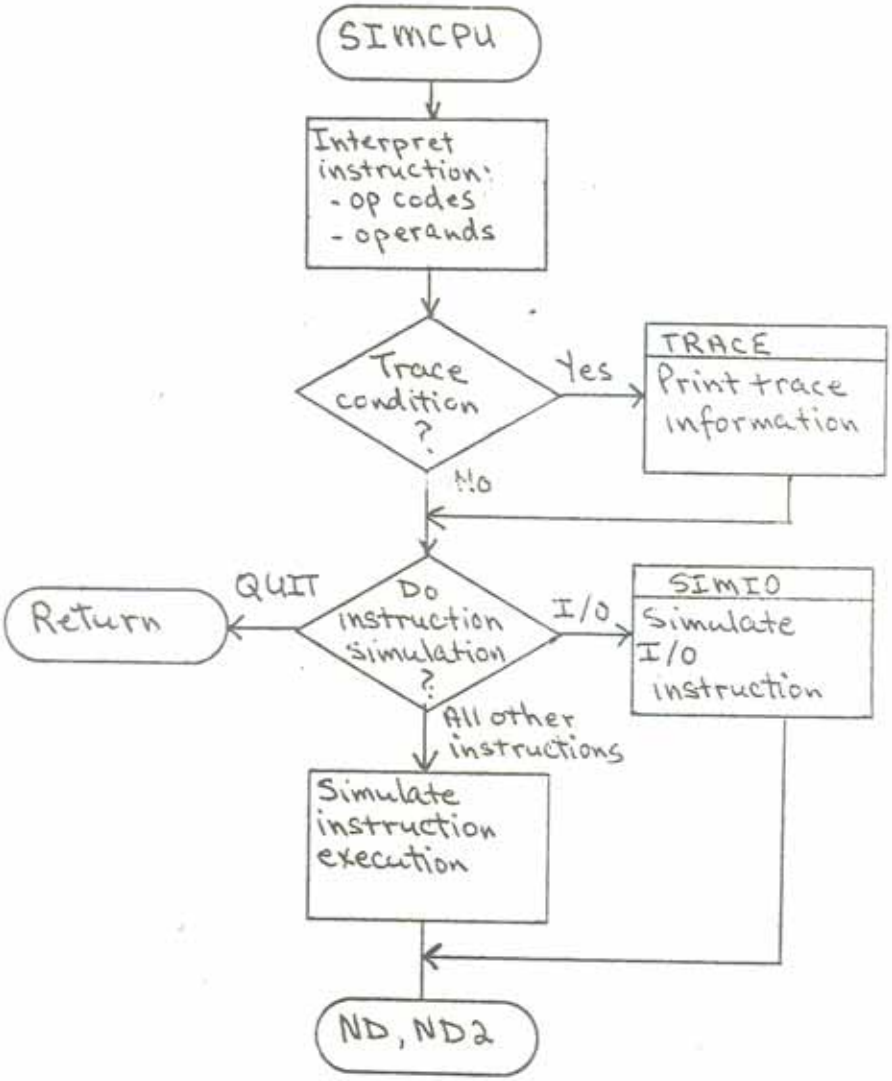


Figure 3 continued
(continued on next page)

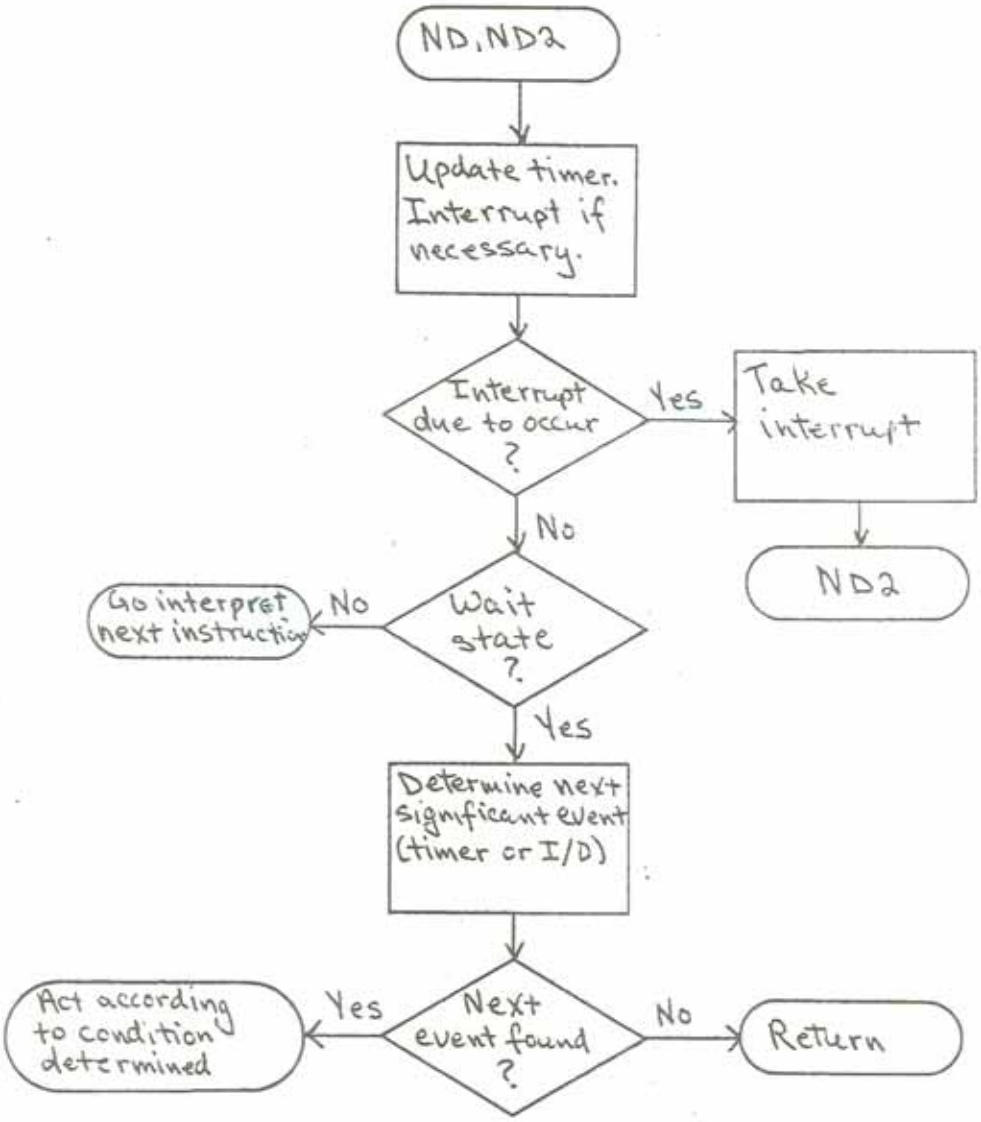


Figure 3 continued
(continued on next page)

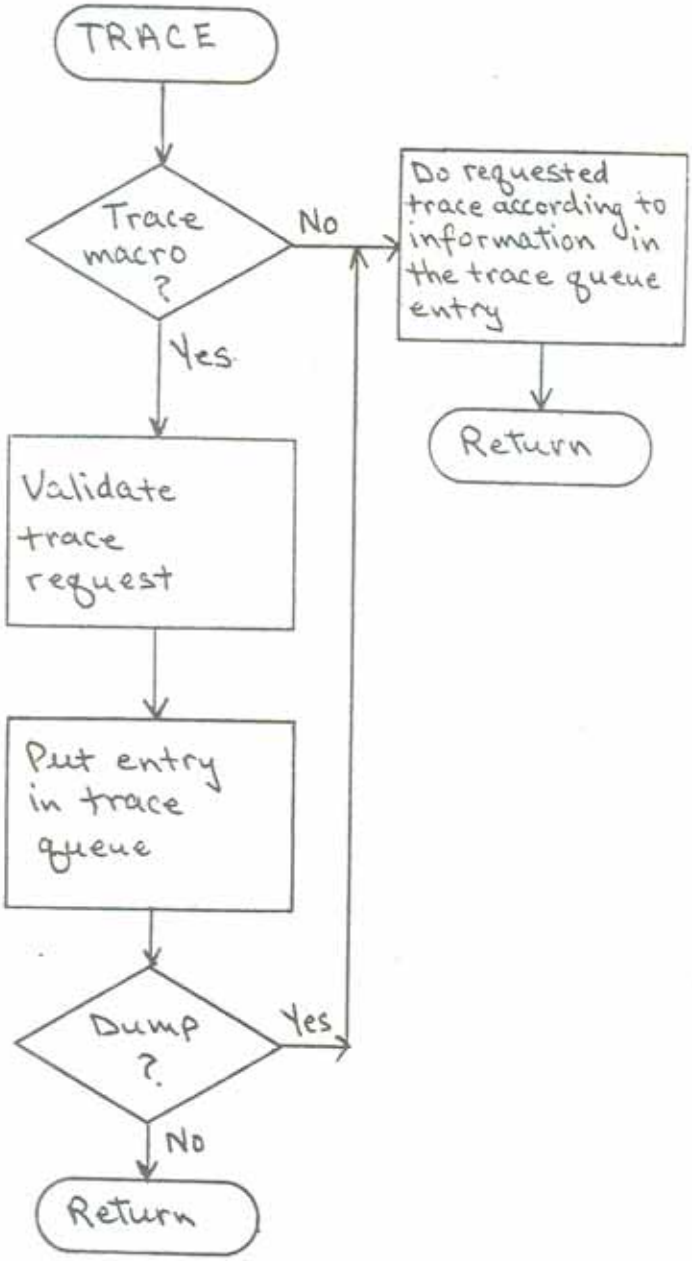


Figure 3 continued
(continued on next page)

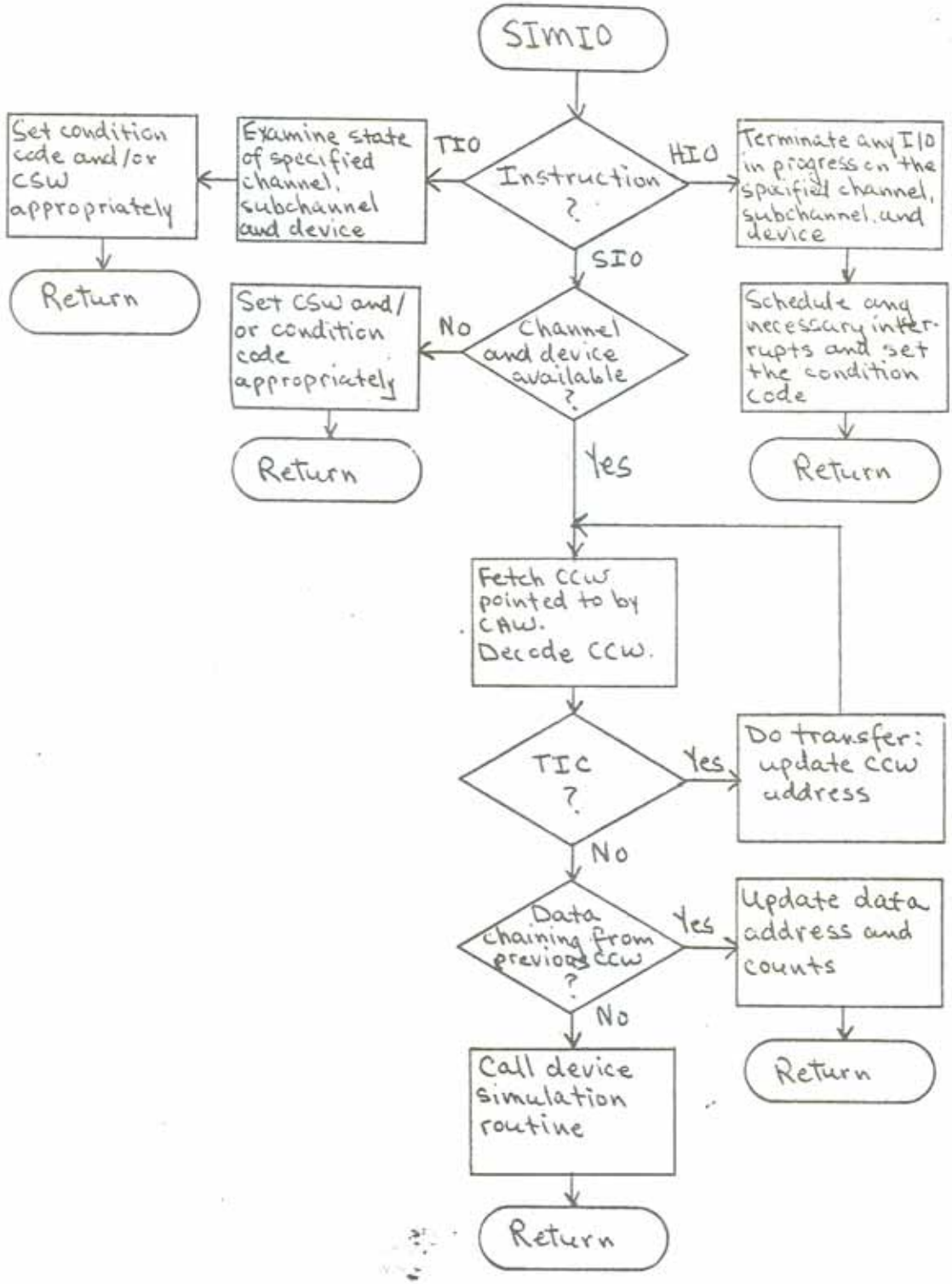


Figure 3 continued

# 3. PROGRAMMING TECHNIQUES

SIM360 is a program, and some insight into the techniques used in programming SIM360 is useful for increased understanding of the simulation and its scope. Some of the more important techniques[1] and data structures used in SIM360 are discussed in a general way in the following sections.

## 3.1 THE VIRTUAL CORE ARRAY

The contents of the core memory of the simulated computer are held in an array composed of n elements, where n is the memory size of the simulated computer. Each of the elements in the array is an eight bit logical quantity which represents one S/360 byte. The bounds of the array are so defined that the index of an element is equal to the memory address of the represented byte. Based arrays defined to contain groups of adjacent bytes are overlayed (by a pointer) on the virtual core array to allow aggregate entities (halfword, fullword, etc.) to be referenced directly. This technique is fully discussed in Appendix C.

---

[1] A representative sample only, not by any means complete.

## 3.2 THE PROGRAM STATUS WORD

The program status word, PSW, of the simulated computer is represented by a structure which contains variables corresponding to the various fields of the PSW in appropriate formats. The condition code, for example, is represented by a bit string of two bits; the program counter (instruction address) is a signed integer which can be used as an index into the virtual core array to fetch an instruction.

## 3.3 THE INTERRUPT AND EVENT QUEUE

Some interrupts, such as program interrupts, occur immediately whenever the proper circumstances arise. Other kinds of interrupts, particularly those associated with I/O, may remain pending indefinitely after they are due to occur either because they are masked off, or because some other interrupt occurs first. In addition, a device simulation routine,[1] in the course of simulating device operation, may determine that one or more interrupts should occur at some future time as a result of device operation. In such a case, an entry or entries will be placed in the interrupt and event queue, a list of pending and scheduled

---

[1] See Appendix C, section C.4.7.

interrupts or events[1] maintained in order by scheduled time of occurrence. Entries in this queue contain information which determines the channel and device involved, status information for the CSW, channel and device, and other necessary information. This queue is examined after the completion of each instruction to see if an interrupt or event is due to occur.

## 3.4 I/O SPECIFICATION BLOCKS

The I/O capabilities of the simulated computer are defined by a set of specification blocks, one for each channel, control unit, and device simulated. A channel specification block (CSB) contains information on the current state of the channel (available, interrupt pending, or working), and a pointer to the control unit specification block (CUSB) of the first attached control unit. The CUSB contains similar status information, and pointers to the next CUSB and the device specification block (DSB) of the first attached device. The DSB for a device contains all necessary information to simulate the device, for example:

-- A pointer to the device simulation routine

---

[1] Events are associated with conditions in the I/O subsystem and are fully explained in Appendix C, section C.4.6.

-- The data transfer rate of the device

-- The record size of the device (if fixed - cards=80, printers=132, etc.)

-- The device status and sense state

-- Pointers to any data in the process of being transferred to or from the device

-- Information on the CCW or chain of CCW's the device is executing

-- etc.

All of this information, and a good deal more, is used by the device simulation routines, the channel interpreter, the event processor, and other functional routines in the process of simulating I/O operations.

## 4. CONCLUSIONS

SIM360 is an unusually complete simulator of a large scale computer, complete in a manner important in the study of operating system programs. It makes available to a large number of people who have no complete access to S/360 hardware a model of that hardware which is sufficiently accurate to be useful in many areas where most simulators are of little use. It has proven useful as a teaching aid and is potentially useful as a tool for:

1) Systems program development and testing
2) Performance monitoring
3) Debugging complex programs

# APPENDIX A

# PROGRAMMING FOR THE S/360 SIMULATOR

This appendix is intended to be a self-contained and sufficient guide for students or other users of SIM360. Familiarity with the S/360 assembler language is assumed.

## A.1 INTRODUCTION

The S/360 simulator is a program written in PL/I which is designed to execute small (less than 32K) assembly language programs in such a fashion that the programmer is unaware of any difference from a physical S/360. In particular, priveleged instructions, protection mechanisms, interrupts and I/O channel programs may be used and manipulated. There are exceptions and qualifications which surround such a statement about any simulation, and several of the more important of these are discussed below. In general, however, any program which will run on the simulator will run on the S/360 and vice versa. Your primary guides in using the simulator are therefore Principles of Operation and the S/360 Assembler Language.

## A.2 IMPLEMENTED INSTRUCTIONS

The simulator does not handle the full complement of

S/360 instructions. A subset designed to be adequate for systems programming use is implemented:

| | MNEMONIC | FORMAT | HEXADECIMAL OP-CODE | NAME |
|---|---|---|---|---|
| I. | LOAD INSTRUCTIONS | | | |
| 1. | L | RX | 58 | Load |
| 2. | LR | RR | 18 | Load |
| 3. | LM | RS | 98 | Load Multiple |
| 4. | LH | RX | 48 | Load Halfword |
| 5. | LTR | RR | 12 | Load and Test |
| II. | STORE INSTRUCTIONS | | | |
| 1. | ST | RX | 50 | Store |
| 2. | STM | RS | 90 | Store Multiple |
| 3. | STH | RX | 40 | Store Halfword |
| 4. | STC | RX | 42 | Store Character |
| III. | ADD INSTRUCTIONS | | | |
| 1. | A | RX | 5A | Add |
| 2. | AR | RR | 1A | Add |
| 3. | AH | RX | 4A | Add Halfword |
| IV. | SUBTRACT INSTRUCTIONS | | | |
| 1. | S | RX | 5B | Subtract |
| 2. | SR | RR | 1B | Subtract |
| 3. | SH | RX | 4B | Subtract Halfword |

V. MULTIPLY INSTRUCTIONS

| | | | | |
|---|---|---|---|---|
| 1. | M | RX | 5C | Multiply |
| 2. | MR | RR | 1C | Multiply |
| 3. | MH | RX | 4C | Multiply Halfword |

VI. DIVIDE INSTRUCTIONS

| | | | | |
|---|---|---|---|---|
| 1. | D | RX | 5D | Divide |
| 2. | DR | RR | 1D | Divide |

VII. COMPARE INSTRUCTIONS

| | | | | |
|---|---|---|---|---|
| 1. | C | RX | 59 | Compare |
| 2. | CR | RR | 19 | Compare |
| 3. | CH | RX | 49 | Compare Halfword |

VIII. COMPARE LOGICAL INSTRUCTIONS

| | | | | |
|---|---|---|---|---|
| 1. | CL | RX | 55 | Compare Logical |
| 2. | CLR | RR | 15 | Compare Logical |
| 3. | CLC | RS | D5 | Compare Logical |
| 4. | CLI | SI | 95 | Compare Logical |

IX. MOVE INSTRUCTIONS

| | | | | |
|---|---|---|---|---|
| 1. | MVC | SS | D2 | Move |
| 2. | MVI | SI | 92 | Move |

X. AND INSTRUCTIONS

| | | | | |
|---|---|---|---|---|
| 1. | N | RX | 54 | And |
| 2. | NR | RR | 14 | And |
| 3. | NC | SS | D4 | And |
| 4. | NI | SI | 94 | And |

XI. OR INSTRUCTIONS

| | | | | |
|---|---|---|---|---|
| 1. | O | RX | 56 | Or |
| 2. | OR | RR | 16 | Or |
| 3. | OC | SS | D6 | Or |
| 4. | OI | SI | 96 | Or |

XII. XOR (EXCLUSIVE OR) INSTRUCTIONS

| | | | | |
|---|---|---|---|---|
| 1. | X | RX | 57 | Exclusive Or |
| 2. | XR | RR | 17 | Exclusive Or |
| 3. | XC | SS | D7 | Exclusive Or |
| 4. | XI | SI | 97 | Exclusive Or |

XIII. SHIFT INSTRUCTIONS (LOGICAL)

| | | | | |
|---|---|---|---|---|
| 1. | SLDL | RS | 8D | Shift Left Double |
| 2. | SLL | RS | 89 | Shift Left Single |
| 3. | SRDL | RS | 8C | Shift Right Double |
| 4. | SRL | RS | 88 | Shift Right Single |

XIV. BRANCH INSTRUCTIONS

| | | | | |
|---|---|---|---|---|
| 1. | BAL | RX | 45 | Branch and Link |
| 2. | BALR | RR | 05 | Branch and Link |
| 3. | BC | RX | 47 | Branch on Condition |
| 4. | BCR | RR | 07 | Branch on Condition |
| 5. | BCT | RX | 46 | Branch on Count |
| 6. | BCTR | RR | 06 | Branch on Count |
| 7. | EX | RX | 44 | Execute |

XV. GENERAL INSTRUCTIONS

| | | | | |
|---|---|---|---|---|
| 1. | LA | RX | 41 | Load Address |
| 2. | IC | RX | 43 | Insert Character |

XVI. I/O INSTRUCTIONS

| | | | | |
|---|---|---|---|---|
| 1. | SIO | SI | 9C | Start I/O |
| 2. | HIO | SI | 9E | Halt I/O |
| 3. | TIO | SI | 9D | Test I/O |
| 4. | TCH | SI | 9F | Test Channel |

XVII. SYSTEM CONTROL INSTRUCTIONS

| | | | | |
|---|---|---|---|---|
| 1. | LPSW | SI | 82 | Load PSW |
| 2. | SVC | RR | 0A | Supervisor Call |
| 3. | SPM | RR | 04 | Set Program Mask |
| 4. | SSM | SI | 80 | Set System Mask |
| 5. | ISK | RR | 09 | Insert Storage Key |
| 6. | SSK | RR | 08 | Set Storage Key |

Use of a valid S/360 instruction which is not implemented by the simulator results in a program interrupt for an operation exception. In addition to the machine instructions listed above, there is a set of simulator extensions to the S/360 instruction set which currently includes:

1. TRACE and TRACEOFF - discussed in section A-5
2. QUIT - the simulator termination commands.

These are implemented as macro-instructions and should be used as such, as they are subject to change.

## A.3 PREPARING A PROGRAM

A program must consist of a single control section with no external references, and must be assembled starting at relative location zero. The simulator initiates execution of a program in a manner similar to the hardware IPL function, and the programmer must provide at location 0 an initial PSW. For example:

```
EXAMPLE1    CSECT
IPLPSW      DC      A(0,START)
UNUSED      DC      7XL8'0002000000000000'
CSWETC      DC      6F'0'
NOINTS      DC      5XL8'0002000000000000'
START       SR      12,12           SET UP BASE
            USING   EXAMPLE1,12
            L       4,BEGINADR
            SR      5,5
            LA      6,256
INITLOOP    ST      45,0(4)
            LA      5,1(5)
            LA      4,4(4)
            BCT     6,INITLOOP
            QUIT
```

```
BEGINADR    DC      A(BLOCK)
BLOCK       DS      256F
            END
```

The DC labeled IPLPSW defines a doubleword at location zero which will be used by the simulator as the initial PSW. In this particular example:

1. All maskable interrupts are disabled.
2. The storage protection key is zero, providing unlimited access to all storage.
3. The CPU is in the running state and the supervisor state.
4. The initial condition code is zero.
5. The first instruction to be executed is at location START.

Other details illustrated by this example are:

1. The programmer must somehow initialize the permanently assigned core addresses (24 - 127) to the initial values he desires. The method used here is recommended.
2. The programmer must provide the assembler with a base register and initialize the register. Another example will show an alternative method.

3. The simulator should be terminated by the use of the QUIT macro-instruction.

## A.4 INPUT/OUTPUT ENVIRONMENT

The current version of the simulator implements only a byte multiplexor channel with two attached 2821 control units. Each 2821 has attached one 2540 card reader-punch and two 1403 printers. The assigned device addresses are:

| | | |
|---|---|---|
| First 2821: | 00C | Reader |
| | 00D | Punch |
| | 00E | First Printer |
| | 00F | Second Printer |
| Second 2821: | 012 | Reader |
| | 013 | Punch |
| | 010 | First Printer |
| | 011 | Second Printer |

Detailed information on the programming required to support these devices is contained in Principles of Operation and in IBM 2821 Control Unit, Component Description (A24-3312-7).

Special considerations involved in programming for these devices on the simulator are:

1. No special features are supported.

2. Stacker select commands to the reader punch are not simulated. Stacker select information in 2540 commands must be valid, but is ignored by the simulator.
3. Carriage skip commands to the printer are all interpreted as a skip to channel 1 (head of form). Carriage skip information in 1403 commands must be valid, but regardless of the channel specified, the paper is positioned at head of form.

## A.5 DEBUGGING AIDS AND MONITORING FEATURES

### A.5.1 FACILITIES

The simulator has extensive and powerful trace facilities to aid in debugging programs. Proper use of these facilities will greatly reduce the number of runs required to solve a given programming problem. The trace facilities are dynamically controlled at execution time by the use of the TRACE and TRACEOFF simulator control instruction. The following trace features are provided:

1. Branch tracing: whenever a successful branch instruction is executed, the standard trace information will be printed. Options may be specified.

2. Address tracing: whenever a given location is referenced as an instruction operand, the standard trace information will be printed. Options may be specified.

3. Execution tracing: whenever the given location is referenced for execution, the standard trace information will be printed. Options may be specified.

4. Instruction tracing: whenever a given instruction (LR, M, SSM, etc.) is executed, the standard trace information will be printed. Options may be specified.

5. Interrupt tracing: whenever the specified type of interrupt occurs, an abbreviated version of the standard trace information will be printed. Options may be specified.

6. Channel tracing: whenever the specified channel performs significant operations, an explanatory message is printed. Examples are:

a) A new CCW is fetched in a chain of command chained CCW's. This information, and the address and text of the fetched CCW are printed.

b) The channel receives status from an attached device. This information and the status byte are printed.

Options may not be specified.

7. Snapshot: whenever the trace command itself is encountered, an abbreviated version of the standard trace information will be printed. Options may be specified.

In addition to the above, the simulator can print the standard trace information for each instruction executed. This facility is not dynamically controlled, and must be set by the instructor.

A.5.2 STANDARD TRACE INFORMATION

The standard trace information mentioned above contains the following information:

1. The current hexadecimal value of the location counter (LOC).

2. The type of the trace request which caused this message. For example:

   PGM INTERRUPT
   ADDRESS 00F6
   INSTRUCTION SSM
   SNAPSHOT

3. The instruction count at the time of the trace message, i.e., the number of instructions which have been executed (COUNT).
4. The elapsed virtual (simulated) time since the start of execution.
5. The contents of the current PSW (hexadecimal).
6. The IBM mnemonic op-code of the instruction associated with the trace message (OP).
7. A hexadecimal dump of the instruction associated with the trace message (INSTRUCTION).
8. The hexadecimal absolute addresses of address operands 1 and 2, if present (ADR1, ADR2).
9. The first four bytes of operands 1 and 2, if present (OPERAND1, OPERAND2).

The abbreviated trace information printed in an interrupt trace message contains only items 1 - 5. The information printed in a trace message associated with instruction execution(Branch, Address, Execution, and Instruction trace

types) reflects the state of the CPU at a theoretical point in time after instruction fetch and address generation, and before any data has been changed by the execution of the instruction. The information printed in a trace message associated with an interrupt is that existing after the old PSW has been stored and before the new PSW has been fetched. Snapshot information is associated with a point in time after completion of the execution of the instruction preceding the trace request and before fetching the instruction following the trace request.

### A.5.3 OPTIONS

The three types of options which may be specified in a trace command are status, registers, and core dump. Status information is that contained in the permanently assigned low core area from location $24_{10}$ - $127_{10}$. This includes the old and new PSW's for the five interrupt classes, the channel status word, the channel address word, and the timer. Any status options requested are formatted appropriately and identified. The registers option is obviously the 16 general purpose registers which are dumped in hexadecimal and decimal and identified. Core dumps are in hexadecimal and character format. Further discussion of the options is included in the syntax description.

## A.5.4 TRACE SYNTAX

The syntax of the trace command is

$$\left[\text{LABEL}\right] \quad \text{TRACE} \quad \left\{ \begin{array}{l} \text{BRANCH} \\ \text{ADDRS,adr} \\ \text{EXEC,adr} \\ \text{INSTR,opcode} \\ \text{INSTPT,type} \\ \text{DUMP} \\ \text{CHANL,chadr} \end{array} \right\} \quad \left[\text{,options}\right]$$

where

adr = a label or decimal address

opcode = hexadecimal opcode of instruction

$$\text{type} = \left\{ \begin{array}{l} \text{PGM} \\ \text{I/O} \\ \text{EXT} \\ \text{SVC} \\ \text{MCK} \end{array} \right\}$$

chadr = an integer 0 - 6

$$\text{options} = \left[ \begin{array}{l} \text{STATUS=} \left\{ \begin{array}{l} \text{status spec} \\ \text{(status spec } \left[\text{,status spec}\right] \text{ )} \end{array} \right\} \\ \text{REGS=} \left\{ \begin{array}{l} \text{integer 0 - 15} \\ \text{(integer 0 - 15 } \left[\text{,integer 0 - 15}\right] \text{ )} \end{array} \right\} \\ \text{CORE=(core spec } \left[\text{,core spec}\right] \text{ )} \end{array} \right]$$

status spec = [ OLDPGM | NEWPGM | OLDI/O | NEWI/O | OLDEXT | NEWEXT | OLDSVC | NEWSVC | OLDMCK | NEWMCK | TIMER | CSW | CAW ]

core spec = adr, wordcount

wordcount = decimal integer 1 - 128

Note the following points:

1. Each 'core spec' is a pair:

   address,wordcount

2. In each trace command allowing options a maximum of

   16 registers

   13 status "fields"

   8 core dump specifications

   may be specified.

3. The example uses snapshots heavily. This is acceptable for very simple programs but for complex problems the more useful information comes from interrupt and branch tracing, and well chosen instruction tracing.

The sample program which follows should help to clarify the information given above. The circled numbers on the trace listing refer to the notes which follow the example.

```
        LSTNAME,A.E.     SECTION--     .     INSTRUCTOR--     .                                    PAGE    1

  LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                                        10 MAY 72
000000                              2 LSTNAME  CSECT
000000 000000000000000                3          DC    A(0,BEGIN)                 INITIAL PSW.

000008 000200000000000                5          DC    10X'0002000000000000'      UNUSED LOCATIONS.

000050 000000000000000                7          DC    A(0,TSTRT)                 EXTERNAL NEW PSW FOR TIMER.
000058 000200000000000                8          DC    X'0002000000000000'        UNUSED PSW'S IN THIS PROGRAM.
000060 000000000000198                9          DC    A(0,PGMINT)                PROGRAM INTERRUPT PSW.
000070 000200000000000               10          DC    X'0002000000000000'        UNUSED PSW.
000078 000000000001C8                11          DC    A(0,IOINT)                 IO INTERRUPT HANDLER.



000000                               13          USING LSTNAME,0                  PROGRAM IS ABSOLUTE.
                                     14          PRINT NOGEN

                                     16 *NOT EXPECTING PROGRAM INTERRUPTS AT THIS TIME, SO IF ONE OCCURS,
                                     17 *DUMP ALL OF USED CORE AND THE REGISTERS TO SEE WHAT HAPPENED.
                                     18 BEGIN    TRACE INTRPT,PGM,STATUS=OLDPGM,REGS=(0,1,2,3,4,5,6,7,8,9,10), X
                                                       CORE=(0,128)


                                     30 *WE WILL FIRST SET UP A TIMER INTERRUPT.  THE WAIT WILL BE AN
                                     31 *ARBRITRARY 3.3 MS, BECAUSE THAT IS ONE TIMER UNIT. WE WILL TURN ON
                                     32 *THE EXTERNAL INTERRUPT TRACE TO WATCH THE INTERRUPT OCCUR.
                                     33          TRACE INTRPT,EXT                 TO SEE TIMER INTERRUPT.
000098 1B66                          42          SR    6,6                        MAKE A NOTHING--3.3 MILLISEC.
00009A 5060 0050          00050      43          ST    6,80                       STORE IN THE TIMER WORD.
00009E 4150 0001          00001      44          LA    5,1                        FOR FUTURE USE.


                                     46 *TO VERIFY THAT WE'VE GOTTEN THIS FAR, DO A SNAPSHOT.
                                     47          TRACE DUMP
                                     56 *NOW GO INTO THE WAIT STATE UNTIL THE TIMER INTERRUPT OCCURS.
                                     57 *NOTE THAT THE EXTERNAL INTERRUPT BIT IS SET ON IN OUR WAIT PSW.
0000AC 82C0 0080          00080      58          LPSW  WAIT                       WAIT FOR A TIMER INTERRUPT.


0000B0                               60          DS    0D
0000B0 010200000000000               61 WAIT     DC    X'0102000000000000'



                                     63 *WHEN THE TIMER INTERRUPT OCCURS, WE COME HERE (SEE THE EXTERNAL NEW
                                     64 *PSW).  NOW WE WILL DO A SHORT LOOP WHICH PRINTS A LINE,DOUBLE SPACING,
                                     65 *AND WATCH AN INSTRUCTION, A LOCATION, AND I/O INTERRUPTS.
                                     66 TSTRT    TRACE ADDRS,SWITCH,REGS=(15,4,5),CORE=(MSG2,1,80,1)
                                     79          TRACE INSTR,48                   SHOW THE LH INSTRUCTION AT CHEND
                                     88          TRACE INTRPT,I/O                 WATCH THE I/O INTERRUPTS.
0000CE 95EF 015B          00158      97 GOON     CLI   SWITCH,X'EF'             TEST FOR LOOP DONE.
```

```
 LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT
000002 4780 0168            00168    98          BE    TCONT                    GO DO PGM INTERRUPTS.
000006 9C00 000E       0000E         99          TIO   X'00E'                   PRINTER FREE?
00000A 4770 011C            0011C   100          BNZ   ABORT                    IF NOT, ERROR, SO QUIT.
00000E D203 3049 01C8 00048 001C8   101          MVC   72(4),=A(PRINTCCW)       SET UP CAW.
000014 9C00 000E       0000E        102          SIO   X'00E'                   AND START THE I/O OPERATION.
000018 9200 0100       00100        103          LPSW  WAITIO                   WAIT TILL I/O OP DONE.
0001CC                              104          DS    0D
0001CC 80020000000C00C0             105 WAITIO   DC    X'8002000000000000'      NOTE CHANNEL 0 BIT ON.



                                    107 *I/O INTERRUPT HANDLER.
                                    108 *THIS VERY STUPID ROUTINE LOOKS FOR A CHANNEL END OR DEVICE END,
                                    109 *AND TAKES APPROPRIATE ACTION IF THEY ARE FOUND.  ANYTHING ELSE IS
                                    110 *CONSIDERED AN ERROR.  NOTE THAT INTERRUPTS ARE ASSUMED TO COME FROM
                                    111 *THE PROPER DEVICE.
0001C8 4850 0044            00044   112 ICINT    LH    5,68                     GET CSW STATUS BITS.
0001CC 4950 01CC            001CC   113          CH    5,=X'0800'               TEST FOR CHANNEL END.
0001D0 4780 0134            00134   114          BE    CHEND                    GO DO CHANNEL END PROCESSING.
0001D4 4950 01CE            001CE   115          CH    5,=X'0400'               TEST FOR DEVICE END.
0001D8 4780 00DE            00CDE   116          BE    GOON                     IF DEVICE END, CONTINUE LOOP.
                                    117 *NOT CHANNEL END OR DEVICE END. IN THIS VERY SIMPLE-MINDED EXAMPLE,
                                    118 *WE WILL ASSUME THAT THIS IS AN ERROR AND ABORT THE PROGRAM.
                                    119 ABORT    TRACE DUMP,STATUS=(OLDI/O,CSW),CCRE=(0,128)
                                    130          QUIT                           ABORT PROGRAM.



                                    135 *GOT A CHANNEL END FROM THE OPERATION IN PROGRESS.
0001D4 5840 0158            00158   136 CHEND    L     4,MSG2                   DECREMENT OUR COUNTER--
0001D8 4150 0C01            00001   137          LA    5,1                      WE CAN DO THIS BECAUSE CHANNEL
0001DC 1845                         138          SR    4,5                      END SIGNIFIES THE END OF DATA
0001DE 5040 0158            00158   139          ST    4,MSG2                   TRANSFER.
                                    140 *WE MUST, HOWEVER, WAIT TILL DEVICE END BEFORE WE CAN INITIATE
                                    141 *ANOTHER OPERATION TO THE SAME DEVICE.
0001E2 9200 0100       00100        142          LPSW  WAITIO
0001E6 0000
0001E8 110001C80C0000CC             143 PRINTCCW CCW   X'11',MSG,X'80',12       TRANSFERS TEXT 'LOOP COUNT='
0001F0 03C001E20CC000C4             144          CCW   X'03',MSG2,X'20',4       NOTE SLI BIT, NOT DOING 132 CH.
0001F8 F0F3F0F3                     145 MSG2     DC    C'0003'
0001FC D3D6D6D740C3D6E4             146 MSG      DC    C'LOOP COUNT= '
000208                              147 SWITCH   EQU   MSG2+3



                                    149 *TURN OFF THE TRACES AND SHOW THAT NOTHING HAPPENS NOW.
                                    150 TCONT    TRACEOFF  ADDF  SWITCH
000210 5840 [illegible]     00158   156          L     4,MSG2
                                    157          CLI   SWITCH,X'00'
```

LSTNAME,A.E. SECTION-- . INSTRUCTOR-- . PAGE 3

10 MAY 72

```
LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT
                                 168 *DONE WITH OUR PRINT LOOP.  NOW WE WILL CREATE SOME FIXED POINT
                                 169 *OVERFLOW CONDITIONS AND WATCH THE INTERRUPTS.  FIRST CHANGE THE
                                 170 *PROGRAM INTERRUPT TRACE TO GIVE US SLIGHTLY LESS DUMP VOLUME.
                                 171          TRACE INTRPT,PGM,STATUS=OLDPGM,REGS=(4,5)
001B8 5870 01C0          001C0   180          L     7,FOFLON                  PROGRAM MASK TO ALLOW FIXED
001BC 0470                       181          SPM   7                         POINT INTERRUPTS.

001BE 5840 01C4          001C4   183          L     4,MAXPOS                  GET THE MAXIMUM POSITIVE NUMBER.
001C2 4150 0002          00002   184          LA    5,2                       AND ADD 2 TO IT TO CAUSE AN
001C6 1A54                       185          AR    5,4                       OVERFLOW
                                 186 *TRACE LISTING SHOULD SHOW A PROGRAM INTERRUPT.
                                 187 *NOW TURN OFF FIXED POINT EXCEPTION INTERRUPTS AND TRY AGAIN.
001C8 1B77                       188          SR    7,7
001CA 0470                       189          SPM   7
001CC 5840 01C4          001C4   190          L     4,MAXPOS                  SAME PROCEDURE.
001D0 4150 0002          00002   191          LA    5,2
001D4 1A45                       192          AR    4,5
                                 193          TRACE DUMP,REGS=(4,5)
                                 202 *ENOUGH   TERMINATE.
                                 203          QUIT

001D8 8200 0028      00028       208 PGMINT   LPSW  40                        THIS IGNORES PROGRAM INTERRUPTS.

001DC
001DC 08000000                   211 FOFLON   DC    X'08000000'
001E0 7FFFFFFF                   212 MAXPOS   DC    X'7FFFFFFF'
                                 213          END
001E4 00000148                   214                =A(PRINTCCW)
001E8 0800                       215                =X'0800'
001EA 0400                       216                =X'0400'
```

CROSS-REFERENCE



| SYMBOL | LEN | VALUE | DEFN | REFERENCES | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| ABORT | 1 | 00C11C | 122 | 100 | | | | | |
| BEGIN | 1 | 00C080 | 21 | 3 | | | | | |
| CHEND | 4 | 00C134 | 136 | 114 | | | | | |
| EOFLDN | 4 | 00C1CC | 211 | 180 | | | | | |
| GOON | 4 | 0000DE | 97 | 116 | | | | | |
| ICINT | 4 | 00C10P | 112 | 11 | | | | | |
| LSTNAME | 1 | 00C00C | 2 | 13 | | | | | |
| MAXFDS | 4 | 00C1C4 | 212 | 183 | 190 | | | | |
| MSG | 12 | 00C15C | 146 | 143 | | | | | |
| MSG2 | 4 | 00C158 | 145 | 73 | 136 | 139 | 144 | 147 | 156 |
| PGMINT | 4 | 00019A | 208 | 9 | | | | | |
| PRINTCCW | 8 | 00C140 | 143 | 101 | 214 | | | | |
| SWITCH | 4 | 00015P | 147 | 70 | 97 | 154 | 157 | | |
| TCCNT | 1 | 00C158 | 153 | 98 | | | | | |
| TSTRT | 1 | 00C08P | 69 | 7 | | | | | |
| WAIT | 8 | 00C080 | 61 | 58 | | | | | |
| WAITIO | 8 | 00C10C | 105 | 103 | 142 | | | | |

RELOCATION DICTIONARY



| PCS.ID | REL.ID | FLAGS | ADDRESS |
|---|---|---|---|
| 01 | 01 | 0C | 000004 |
| C1 | 01 | CC | 00005C |
| C1 | 01 | CC | 00CC6C |
| 01 | 01 | 0C | 00007C |
| C1 | 01 | C4 | 0000BA |
| C1 | 01 | C4 | 000CC0 |
| 01 | 01 | 08 | 000149 |
| 01 | 01 | C8 | 000151 |
| 01 | 01 | 04 | 00016A |
| 01 | 01 | 0C | 0001C8 |

DIAGNOSTICS



STMT ERROR CODE MESSAGE

ASMG046I AT LEAST ONE RELOCATABLE Y-TYPE CONSTANT IN ASSEMBLY.

NO STATEMENTS FLAGGED IN THIS ASSEMBLY
4 WAS HIGHEST SEVERITY CODE

LSTNAME TRACE LISTING. 0 CONDITIONS ENABLED.

| LOC | TRACE TYPE | COUNT | TIME | | OP | INSTRUCTION | ADR1 | ADR2 | OPERAND1 | OP |
|---|---|---|---|---|---|---|---|---|---|---|
| | (1) | | (2) | | | | | | | |
| 00AC | SNAPSHOT | 3 | 1 | PSW: 00 0 0 0000 *01'B *00'B 0 0000AC | | | | | | |
| 0000 | EXT INTERRUPT (3) | 4 | 3,333 | PSW: 01 0 2 0080 *10'B *00'B 0 000000 | | | | | | |
| 00E2 | ADDRESS: 0158 | 4 | 3,333 | PSW: 00 0 0 0000 *10'B *00'B 0 0000E2 | CLI | 95EF 0158 | | 0158 | | F3D3D6D6 |
| | REGISTERS: | | | | | | | | | |
| | R 4: FFFFFFFF, | | -1 R 5: 00000001, | 1 R15: FFFFFFFF, | | -1 | | | | |
| | CORE DUMP: | | | | | | | | | |
| | 0158, 344: F0F0F0F3 | | | | 0003 | | | | | |
| | 0050, 8C: FFFFFF00 | | | | ???? | | | | | |
| 00C0 | I/O INTERRUPT | 11 | 3,428 | PSW: 80 0 2 000E *10'B *00'B 0 000000 | | | | | | |
| 010C | LH INSTRUCTION | 11 | 3,428 | PSW: 00 0 0 0000 *10'B *00'B 0 00010C | LH | 4850 0044 | | 0044 | | 08000074 |
| 0136 | ADDRESS: 015E (5) | 14 | 3,431 | PSW: 00 0 0 0000 *1C'B *00'B 0 000138 | L | 5840 0158 | | 0158 | | F0F0F0F3 |
| | REGISTERS: | | | | | | | | | |
| | R 4: FFFFFFFF, | | -1 R 5: 00000800, | 2048 R15: FFFFFFFF, | | -1 | | | | |
| | CORE DUMP: | | | | | | | | | |
| | 0158, 344: F0F0F0F3 | | | | 0003 | | | | | |
| | 0050, 80: FFFFFF00 | | | | ???? | | | | | |
| 0142 | ADDRESS: 0158 | 17 | 3,433 | PSW: 00 0 0 0000 *10'B *01'B 0 000142 | ST | 5040 0158 | | 0158 | | F0F0F0F3 |
| | REGISTERS: | | | | | | | | | |
| | R 4: F0F0F0F2, | | -252645134 R 5: 00000001, | 1 R15: FFFFFFFF, | | -1 | | | | |
| | CORE DUMP: | | | | | | | | | |
| | 0158, 344: F0F0F0F3 | | | | 0003 | | | | | |
| | 0050, 80: FFFFFF00 | | | | ???? | | | | | |
| 00C0 | I/O INTERRUPT | 19 | 99,680 | PSW: 80 0 2 000E *10'B *00'B 0 000000 | | | | | | |
| 010C | LH INSTRUCTION | 19 | 99,680 | PSW: 00 0 0 0000 *10'B *00'B 0 00010C | LH | 4850 0044 | | 0044 | | 04000000 |
| 00E2 | ADDRESS: 0158 | 24 | 99,684 | PSW: 00 0 0 0000 *10'B *00'B 0 0000E2 | CLI | 95EF 0158 | | 0158 | | F2D3D6D6 |
| | REGISTERS: | | | | | | | | | |
| | R 4: F0F0F0F2, | | -252645134 R 5: 00000400, | 1024 R15: FFFFFFFF, | | -1 | | | | |
| | CORE DUMP: | | | | | | | | | |
| | 0158, 344: F0F0F0F2 | | | | 0002 | | | | | |
| | 0050, [illegible] | | | | ???? | | | | | |
| 00C0 | I/O INTERRUPT | 31 | 99,779 | PSW: 80 0 2 000E *10'B *00'B 0 000000 | | | | | | |

LSTNAME TRACE LISTING. 5 CONDITIONS ENABLED.

```
LCC  TRACE TYPE          COUNT   TIME                                                    OP   INSTRUCTION   ADR1 ADR2  OPERAND1 OP
---- ----------------    -----   ----------                                              ---- ------------  ---- ----  -------- --

0138 ADDRESS: 0158          34      99,782  PSW: 00 0 0 0000  *10*B *00*B 0 000138      L    5840 0158     0158       F0F0F0F2
       REGISTERS:
         R 4: F0F0F0F2, -252645134 R 5: 00000800,       2048 R15: FFFFFFFF,                    -1
       CORE DUMP:
         0158,   344: F0F0F0F2                                                  0002
         0050,    80: FFFFE300                                                  ??T?

0142 ADDRESS: 0158          37      99,784  PSW: 00 0 0 0000  *10*B *01*B 0 000142      ST   5040 0158     0158       F0F0F0F2
       REGISTERS:
         R 4: F0F0F0F1, -252645135 R 5: 00000001,          1 R15: FFFFFFFF,                    -1
       CORE DUMP:
         0158,   344: F0F0F0F2                                                  0002
         0050,    80: FFFFE300                                                  ??T?

0CC0 I/O INTERRUPT          39     198,181  PSW: 80 0 2 000E  *10*B *00*B 0 000000

01CC LH   INSTRUCTION       39     198,181  PSW: 00 0 0 0000  *10*B *00*B 0 00010C      LH   4850 0044     0044       04000000

00E2 ADDRESS: 015B          44     198,186  PSW: 00 0 0 0000  *10*B *00*B 0 0000E2      CLI  95EF 015B     015B       F1D3D6D6
       REGISTERS:
         R 4: F0F0F0F1, -252645135 R 5: 00000400,       1024 R15: FFFFFFFF,                    -1
       CORE DUMP:
         0158,   344: F0F0F0F1                                                  0001
         0050,    80: FFFFC500                                                  ??E?

0CC0 I/O INTERRUPT          51     198,281  PSW: 80 0 2 000E  *10*B *00*B 0 000000

010C LH   INSTRUCTION       51     198,281  PSW: 00 0 0 0000  *10*B *00*B 0 00010C      LH   4850 0044     0044       08000074

0138 ADDRESS: 0158          54     198,284  PSW: 00 0 0 0000  *10*B *00*B 0 000138      L    5840 0158     0158       F0F0F0F1
       REGISTERS:
         R 4: F0F0F0F1, -252645135 R 5: 00000800,       2048 R15: FFFFFFFF,                    -1
       CORE DUMP:
         0158,   344: F0F0F0F1                                                  0001
         005C,    80: FFFFC500                                                  ??E?

0142 ADDRESS: 0158          57     198,285  PSW: 00 0 0 0000  *10*B *01*B 0 000142      ST   5040 0158     0158       F0F0F0F1
       REGISTERS:
         R 4: F0F0F0F0, -252645136 R 5: 00000001,          1 R15: FFFFFFFF,                    -1
       CORE DUMP:
         0158,   344: F0F0F0F1                                                  0001
         0050,    80: FFFFC500                                                  ??E?

0CC0 I/O INTERRUPT          59     300,985  PSW: 80 0 2 000E  *10*B *00*B 0 000000
```

LISTING. TRACE LISTING. 5 CONDITIONS ENABLED.



| LOC | TRACE TYPE | COUNT | TIME | | OP | INSTRUCTION | ADR1 | ADR2 | OPERAND1 | OPE |
|---|---|---|---|---|---|---|---|---|---|---|
| | REGISTERS: | | | | | | | | | |
| | R 4: FCF0F0F0, -252645136 R 5: 00000400, | | | 1C24 R15: FFFFFFFF, | | -1 | | | | |
| | CORE DUMP: | | | | | | | | | |
| | 0158, 344: F0F0F0F0 | | | | | | | | | |
| | 0050, 80: FFFFA600 | | | | 0000 | | | | | |
| | | | | | ???? | | | | | |
| 00CC | I/O INTERRUPT | 71 | 301,085 | PSW: 80 0 2 000E *10*B *00*B 0 000000 | | | | | | |
| 010C | LH INSTRUCTION | 71 | 301,085 | PSW: 00 0 0 0000 *10*B *00*B 0 00010C | LH | 4850 0044 | 0044 | | 08000074 | |
| 0138 | ADDRESS: 0158 | 74 | 301,088 | PSW: 00 0 0 0000 *10*B *00*B 0 000138 | L | 5840 0158 | 0158 | | F0F0F0F0 | |
| | REGISTERS: | | | | | | | | | |
| | R 4: FCF0FCF0, -252645136 R 5: 00000800, | | | 2048 R15: FFFFFFFF, | | -1 | | | | |
| | CORE DUMP: | | | | | | | | | |
| | 0158, 344: F0F0FCF0 | | | | | | | | | |
| | 0050, 80: FFFFA600 | | | | 0000 | | | | | |
| | | | | | ???? | | | | | |
| 0142 | ADDRESS: 0158 | 77 | 301,089 | PSW: 00 0 0 0000 *10*B *01*B 0 000142 | ST | 5040 0158 | 0158 | | F0F0F0F0 | |
| | REGISTERS: | | | | | | | | | |
| | R 4: FCF0F0EF, -252645137 R 5: 00000001, | | | 1 R15: FFFFFFFF, | | -1 | | | | |
| | CORE DUMP: | | | | | | | | | |
| | 0156, 344: F0F0F0F0 | | | | | | | | | |
| | 0050, 80: FFFFA600 | | | | 0000 | | | | | |
| | | | | | ???? | | | | | |
| 00CC | I/O INTERRUPT | 79 | 396,693 | PSW: 80 0 2 000E *10*B *00*B 0 000000 | | | | | | |
| 010C | LH INSTRUCTION | 79 | 396,693 | PSW: 00 0 0 0000 *10*B *00*B 0 00010C | LH | 4850 0044 | 0044 | | 04000000 | |
| 00E2 | ADDRESS: 0158 | 84 | 396,697 | PSW: 00 0 0 0000 *10*B *00*B 0 0000E2 | CLI | 95EF 0158 | 0158 | | EFD3D6D6 | |
| | REGISTERS: | | | | | | | | | |
| | R 4: FCF0FCEF, -252645137 R 5: 00000400, | | | 1024 R15: FFFFFFFF, | | -1 | | | | |
| | CORE DUMP: | | | | | | | | | |
| | 0158, 344: F0F0F0EF | | | | | | | | | |
| | 0050, 8C: FFFF85C0 | | | | 0007 | | | | | |
| | | | | | ???? | | | | | |
| 017E | SNAPSHOT | 83 | 396,700 | PSW: 00 0 0 0000 *01*B *10*B 0 00017E | | | | | | |
| 0198 | PGM INTERRUPT | 92 | 396,702 | PSW: 00 0 0 0008 *01*B *11*B 8 000198 | | | | | | |
| | STATUS: | | | | | | | | | |
| | CLD PRG: CC 0 C 0008 *01*B *11*B 8 000198 | | | | | | | | | |
| | REGISTERS: | | | | | | | | | |
| | R 4: 7FFFFFFF, 2147483647 R 5: 80000001, -2147483647 | | | | | | | | | |
| 0180 | SNAPSHOT | 95 | 396,708 | PSW: 00 0 0 0000 *01*B *11*B 0 000180 | | | | | | |
| | REGISTERS: | | | | | | | | | |
| | R 4: FCF0F0F1, -252645135 R 5: 00000001, 1 | | | | | | | | | |

NORMAL SIMULATOR TERMINATION FOR PROGRAM LSTNAME

SIMULATED REAL TIME: 396,705.687

SIMULATED CPU TIME: 120.625 .

```
LSTNAME                    OUTPUT TO PRINTER OOE STARTS AT HEAD OF FORM ON NEXT PAGE
```

```
LOOP COUNT= 0003
LOOP COUNT= 0002
LOOP COUNT= 0001
LOOP COUNT= 0000
```

Notes on the example:

1) The location shown in this column is usually the location of the instruction following the instruction associated with the trace.

2) The time is the elapsed time, in microseconds, since the start of the simulation.

3) See the LPSW instruction at statement 58, and its operand at statement 61.

4) This group defines one iteration of the print loop.

5) Note that although this instruction does not address the traced location directly, the location is contained in the operand, and the trace occurs.

A.5.5 TURNING OFF TRACING

Any requested trace facility may be removed when no longer needed by means of the TRACEOFF command. The syntax of this command is identical to that of the trace command, except that no options are included. In addition, all traces of a given type may be turned off by replacing the explicit specification adr, opcode, type or chadr with the word ALL. For example:

```
TRACE     INSTR,45        TRACE BAL
TRACE     INSTR,82        TRACE LPSW
TRACE     INSTR,50        TRACE ST
TRACE     INSTR,40        TRACE STH
TRACE     INSTR,42        TRACE STC

      - - code - -

TRACEOFF  INSTR,ALL       TURN OFF ALL INSTR
```

A.6 HINTS

1. Do not place any cards containing // or /* in columns 1 and 2 in your deck.
2. Use your last name (maximum of 8 characters) as the label on the CSECT card which must be the first card of your program. This makes it easy to identify your assembler listing and output. Use a TITLE card with your name also for further ease

in identifying listings.

3. Your deck should have one each CSECT card (above, first card) and END card (last card).
4. Do not use the EXTRN or ENTRY statements or Q or V address constants.
5. The first 128 locations (16 doublewords) must be properly initialized.
6. The instructor can set the maximum number of instructions you can execute and the amount of virtual time which you have to run. Be efficient in your code and use the wait state properly.

# APPENDIX B

# INSTRUCTORS MANUAL

An overview of the steps in running the simulator was given in section 2.3. The detailed procedures are only slightly more involved. The following sections discuss in detail all the necessary considerations.

## B.1 STUDENT DECKS

Each student deck must be one (only) S/360 assembler language control section. The simulator's loader cannot link control sections, properly process external symbols, or relocate to a base address other than zero. To enable all of the simulator output to be easily collated, the name (label) on the CSECT card is used by the simulator as an identifying tag on all ouptput. A student deck should therefore look like

```
name     CSECT                    FIRST CARD

         assembly language statements

         END                      LAST CARD
```

where name is an appropriate identifier (student's last name or assigned I.D. number, for example). It should be

noted that improper use of the ICTL assembler control instruction can cause the assembler to terminate processing, and thus abort many assemblies in a batch run. It has been a successful policy to simply make no mention whatsoever of this, since this feature of the assembler is very rarely used. Should the problem arise, students can be instructed never to use this statement.

B.2 ASSEMBLER INSTRUCTIONS

All student decks for a given run should be grouped into one large deck, checking each student deck for the presence of an END card. In front of the student decks appropriate Job Control Language control cards must be provided. The following cards are the appropriate ones for M.I.T.'s Information Processing Center:

```
//              JOB,                    PROVIDED BY IPC
// 'SUBMITTER'S NAME',REGION=200K,CLASS=B,MSGLEVEL=(1,1)
/*MITID USER=(M1234,5678)
/*SRI LOW
/*MAIN TIME=5,LINES=6
//STEPNAME  EXEC  ASM,LEVEL=G,PARM.C='LOAD,NODECK,BATCH'
//C.SYSLIB    DD  DSNAME=USERFILE.M4568.10113.MACLIB,
// DISP=SHR
//C.SYSIN  DD  *
          student decks
/*
- more control cards to follow, discussed below -
```

The first four cards are the job and job parameter cards. The number of cards and information required here will vary widely from installation to installation. The cards shown are included for completeness. The next three cards shown are included for completeness. The next three cards are those required when a catalogued procedure such as the one provided by IBM is available; refer to the Assembler(F) Programmer's Guide (C28-3756) for further information. The important point here is that the temporary data set named &&TEMP must be created, contain the object module output of the assembler, and be passed to the next job step. Note that a private macro library, containing the TRACE, TRACEOFF, and QUIT macros, must be used.

## B.3 SIMULATOR INSTRUCTIONS

The complete JCL necessary to run the simulator is:

```
//STEPSIM EXEC PGM=SIM360,
// PARM='MAXTIME=10000,MAXCOUNT=4000,CARDS=2,PRINT=3,MAXPGE=11'
//STEPLIB  DD  DSNAME=USERFILE.M4568.10113.LDLIB,DISP=SHR
//SIMLIN   DD  DSNAME=&&TEMP,DISP=(OLD,DELETE)
//SYSPRINT  DD  SYSOUT=A
//STRACE    DD  SYSOUT=A
//SIMPRNT   DD  SYSOUT=A
//SIMPRN2   DD  DUMMY
//SIMPRN3   DD  SYSOUT=A
//SIMPRN4   DD  DUMMY
//SIMPNCH   DD  DUMMY
//SIMPNC2   DD  DUMMY
```

```
//SIMPNC3  DD  DUMMY
//SIMPNC4  DD  DUMMY
//SIMIN  DD  *
- data cards for simulated reader 00C -
//SIMIN2  DD  *
- data cards for reader 012 -
//SIMIN3  DD  DUMMY
//SIMIN4  DD  DUMMY
```

The use of the various cards is explained below, after the discussion of the parameters which may be included in the PARM= field.

B.3.1 SIMULATOR OPTIONS

The PARM= parameter on the EXEC card which invokes the simulator may contain any combination of the following options. Defaults assumed by the simulator are underlined; any error in the parameter field produces a terse diagnostic, and the simulator will not run. It does examine the entire parameter field for validity, however.

MAXTIME=n — n must be a positive decimal integer which represents the maximum amount of simulated real time in milliseconds which will be allowed to elapse for one program. Default is 1000, or one second.

| | |
|---|---|
| MAXCOUNT=n | n must be a positive decimal integer which represents the maximum number of instructions which the simulator will execute for one program. Default is 500. |
| MAXPGE=n | n must be a positive decimal integer which represents the maximum number of pages of trace output (printer data set STRACE) which will be allowed for <u>each</u> program run. Default is 5. |
| CARDS= 0,<u>1</u>,2,3,4 | The number of input streams to the simulator, corresponding to the DD statements labeled SIMIN (corresponding to 1), SIMIN2 (corresponding to 2), SIMIN3 (corresponding to 3), SIMIN4 (corresponding to 4).[1] |
| PUNCH= <u>0</u>,1,2,3,4 | The number of punch output streams to be used. Corresponding DD statements are SIMPNCH - SIMPNC4.[1] |

[1] Only two 2540 card reader-punches are available in the current version of the simulator; the card input (output) streams 3 and 4 are available for expansion.

PRINT= 0,1,2,3,4 — The number of print output streams to be used. Corresponding DD statements are SIMPRNT - SIMPRN4.

PNCHDEST= PRNT,PNCH — If it is desired to print rather than punch the punch output streams, use'PRNT'. If they are to be punched, use'PNCH'. The DD cards SIMPNCH - SIMPNC4 must be correspondingly adjusted.

PGMINT= YES,NO — If 'NO', then a program interrupt which occurs after a program interrupt and before an LPSW instruction is executed will cause the simulator to print a diagnostic and terminate the program. 'YES' causes the simulator to ignore this condition.

TRACE= ALL,NONE — If 'ALL', then every instruction causes the standard trace message to be printed. Otherwise, only trace conditions enables dynamically by the program are printed.

The following options are for maintenance and debugging use only. Note that they can cause many thousands of lines to go to SYSPRINT.

TDUMP= 0,1,2 — '1' causes the trace queue to be dumped every time a new trace condition is enabled. '2' causes a trace queue dump as for '1', and in addition on every occasion when a trace message is printed. '0' inhibits all trace queue dumps.

IQDUMP= 0,1,2,3 — '1' causes the interrupt and event queue to be dumped each time an I/O interrupt occurs. '2' causes the channel specification block and the I/O specification block to be dumped after the initiation of each device operation. '3' causes both of the above. '0' is for no dumps.

PDUMP= YES,NO — The simulator's link-loader module will print relevant information on programs loaded and initiation of simulation if "YES".

B.3.2 SIMULATOR JCL

```
//STEPLIB  DD  DSNAME=          etc.
```

This defines the 'library' which will be searched first to find SIM360 when the system starts to execute the simulator. Refer to IBM System/360 Operating System: Job Control Language Reference, Form C28-6704.

```
//SIMLIN  DD  DSNAME=&&TEMP,DISP=(OLD,DELETE)
```

This defines the assembler output from the previous step as the program input to the simulator.

```
//SYSPRINT  DD  SYSOUT=A
```

In the event of a serious error detected by the simulator or the operating system, diagnostic information will be printed on the SYSPRINT data set. This control card is also used by the maintenance and debugging options of the simulator (TDUMP=, etc.).

```
//STRACE  DD  SYSOUT=A
```

All trace information generated by the simulator goes to this data set. Note that the MAXPGE= option may be used to prevent an erroneous program from generating hundreds of pages of trace output.

```
//SIMPRNT  DD  SYSOUT=A
```

Output to printer 00E goes to the data set defined by this control card. This card and all following may be punched as follows if the simulated I/O device is not to be used:

```
//SIMPRN2  DD  DUMMY
```

Remember that correspondence is required between the PRINT= option of the simulator and the SIMPRNx control cards

(PRINT=1 implies that only SIMPRNT and printer 00E will be used; PRINT=2 implies that SIMPRN2 and printer 00F will be used in addition, etc.).

```
//SIMPRN3  DD  SYSOUT=A
```

This particular simulator run (of an assigned student machine problem) was using printers 00E and 010, but not printer 00F or 011.

```
//SIMPRN4  DD  DUMMY
```

Not used in this case.

```
//SIMPNCH  DD  DUMMY
//SIMPNC2  DD  DUMMY
//SIMPNC3  DD  DUMMY
//SIMPNC4  DD  DUMMY
```

No card punches were used in this example. If a punch were to be used, the corresponding JCL card would normally be

```
//SIMPNCx  DD  SYSOUT=B
```

```
//SIMIN  DD  *
```

This JCL card must be followed by the data cards which are to be read by the simulated card reader 00C. Note that because of System/360 Operating System conventions, none of

these data cards may contain // or /* in columns 1 and 2.

```
//SIMIN2  DD  *
```

To be followed by cards for simulated reader 012.

```
//SIMIN3  DD  DUMMY
//SIMIN4  DD  DUMMY
```

The simulator in its current implementation has no device which uses these data sets. It is recommended, however, that they be included, since the simulator may attempt to open the data set if an error occurs in punching the CARDS= option.

# APPENDIX C

# GUIDE TO MAINTENANCE, MODIFICATION AND REPROGRAMMING

Because it is written in PL/I, maintenance or reprogramming of the simulator should be fairly straightforward. The following discussion will therefore trace the overall logic and functional behavior of the code, and avoid detailed description except where necessary.

## C.1 OVERVIEW

The simulator is composed of four major modules:

1. SIMLINK - Reads and loads into virtual core array the object module (assembler output). Also does initial parameter processing.
2. SIMCPU - Does simulation of CPU functions, instruction execution, timer, interrupts. Also does simulation of DMA data transfers.
3. TRACE - Processes dynamic trace command interpretation and does the processing and formatting associated with trace output.
4. SIMIO - [illegible]s all processing related to I/O

instructions, CCW's, and the internal performance of I/O subsystems.

In addition to these four major components, there is a very small (42 BAL instructions) assembly language subroutine which does simulation of fullword multiplication and division. This is necessary because these two S/360 instructions (M and D) require 64 bits of precision and PL/I does not have this capability.

## C.2 MODULE SIMLINK

### C.2.1 PARAMETER PROCESSING

This module contains the initial entry point to the simulator. First the parameters passed to the simulator from the PARM= field on the EXEC card are processed. Processing is very straightforward and is outlined in Figure C-1. Refer to the Appendix B for further information on parameter keywords and their effect.

### C.2.2 PROGRAM LOADING

When parameter processing is completed, the error switch is tested and if an error has occurred the program terminates. Otherwise, some initialization is performed (at the label RESTRT; see Figure C-2), and the output data set of the assembler is implicitly opened and the first card read. Of the five valid record types produced by the assembler,[1] only

[1] IBM System/360 Operating System: Assembler(F) Programmer's Guide, Form C26-3756

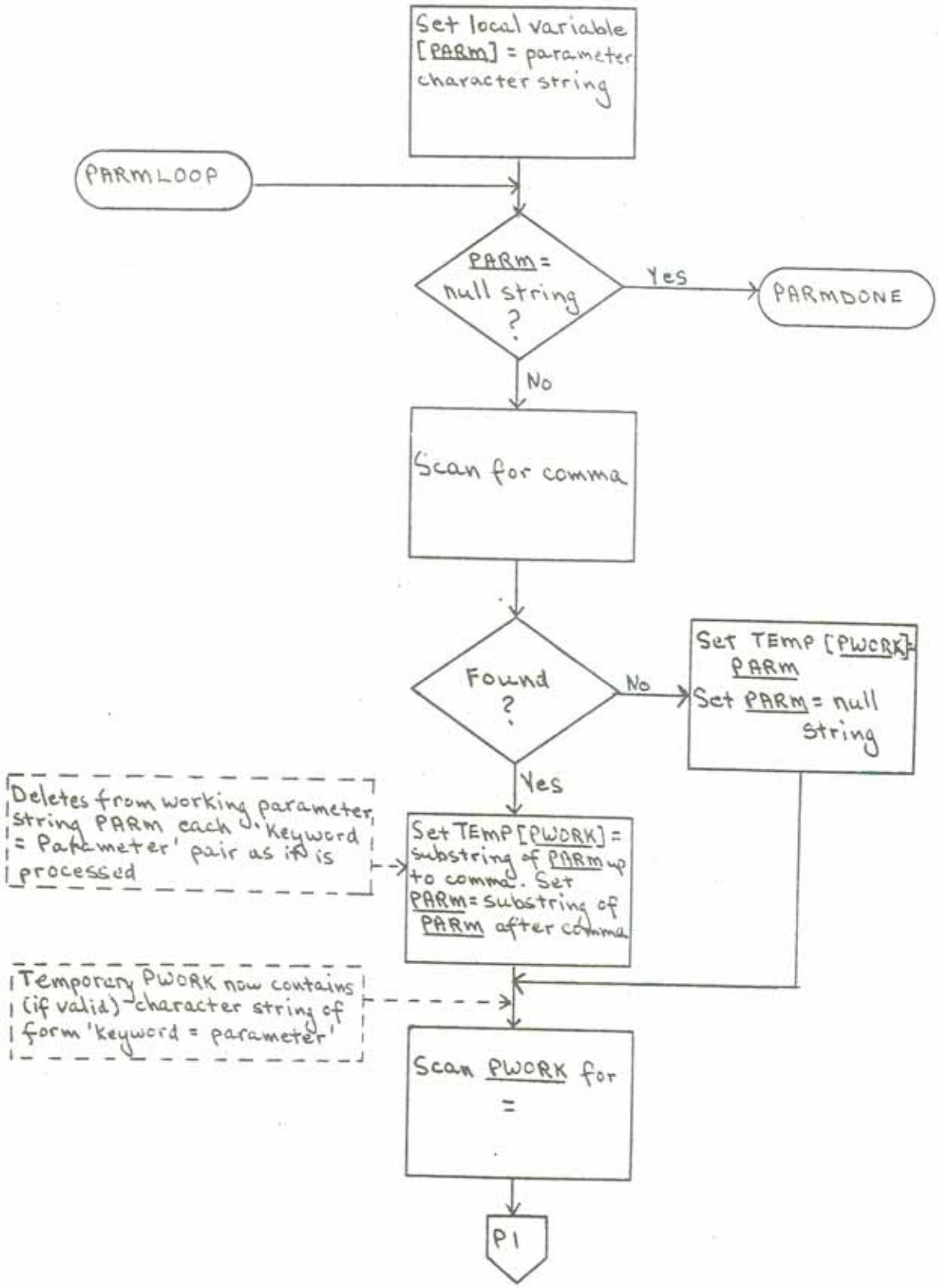


Figure C-1 : Parameter Processing
(continued on next page)

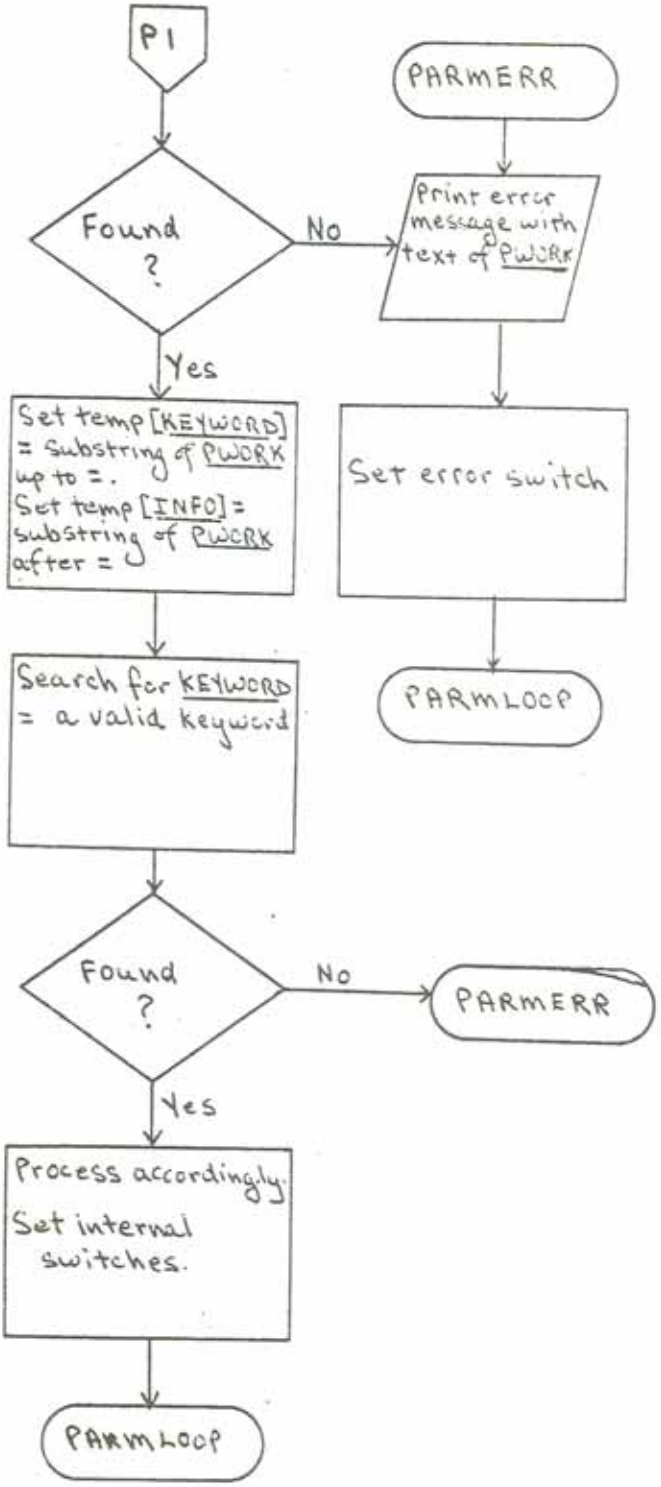


Figure C-1, continued
(continued on next page)

| Valid Keywords | Internal Switches or Variables Affected |
|---|---|
| MAXTIME | MAXT |
| MAXCOUNT | MAXI |
| TRACE | TSWITCH |
| PGMINT | PGM_SW |
| CARDS | NOINSTR |
| PUNCH | NOPNSTR |
| PRINT | NOPRSTR |
| MAXPGE | MXPGCNT |
| PNCHDEST | PPRNTSW |
| TDUMP | TQDMPSW |
| IQDUMP | CDDMPSW, IQDMPSW |
| PDUMP | PDUMPSW |

Figure C-1 continued

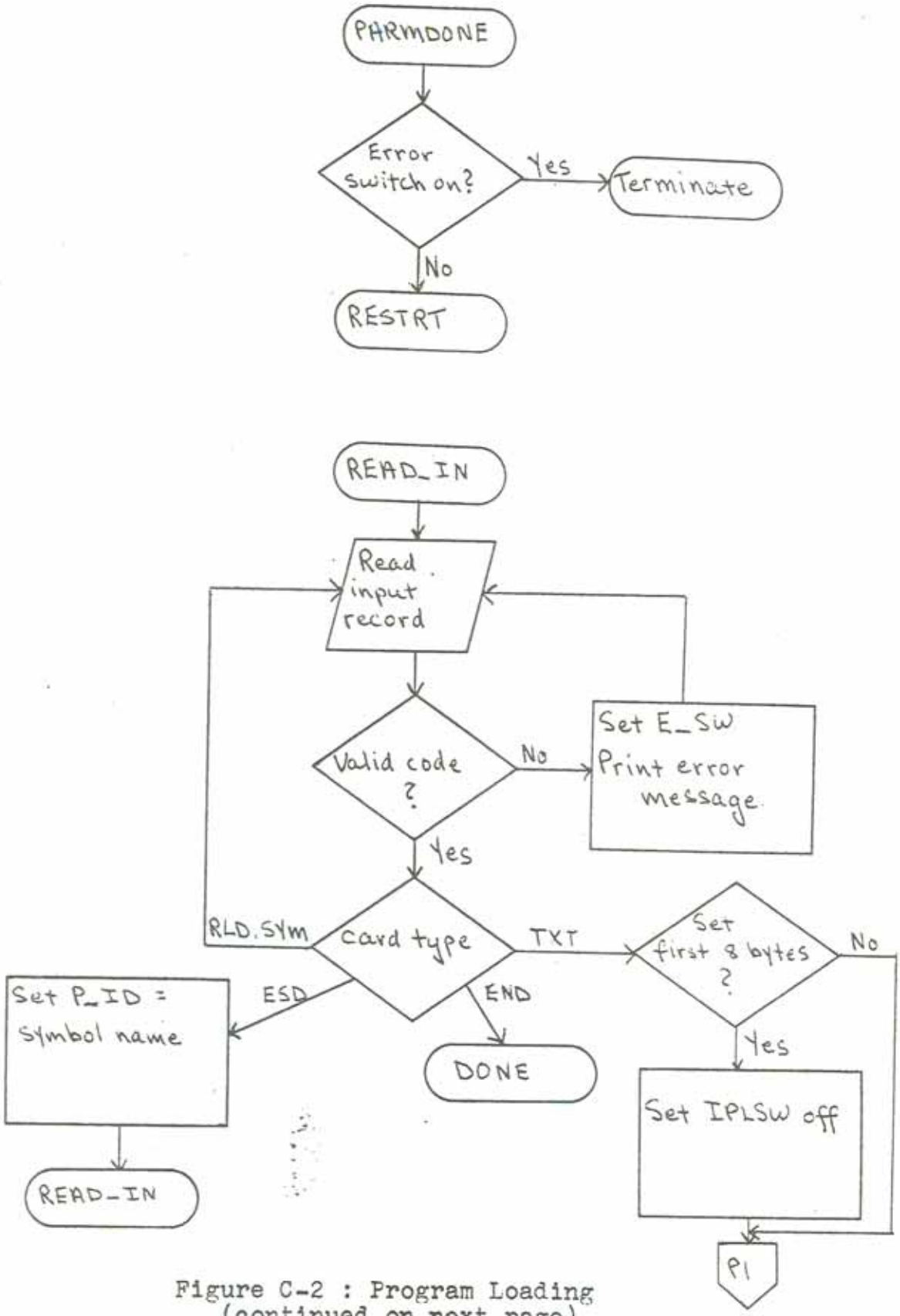


Figure C-2 : Program Loading
(continued on next page)

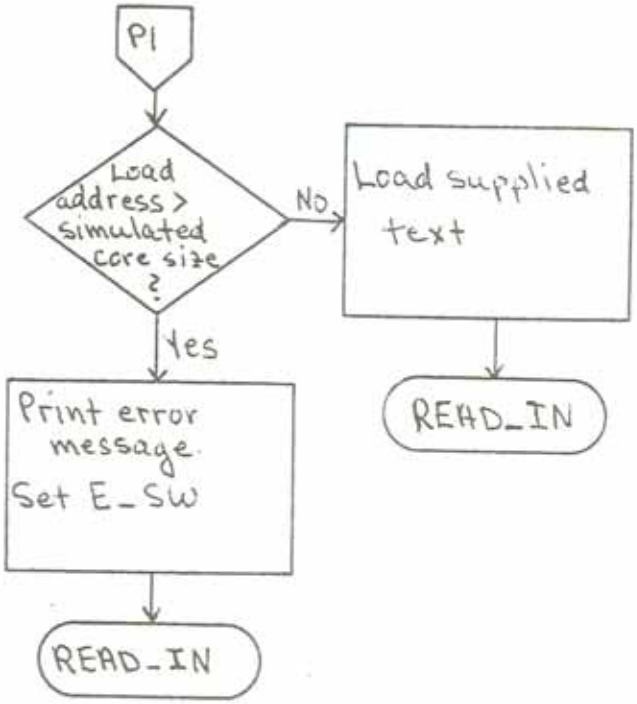


Figure C-2 continued
(continued on next page)

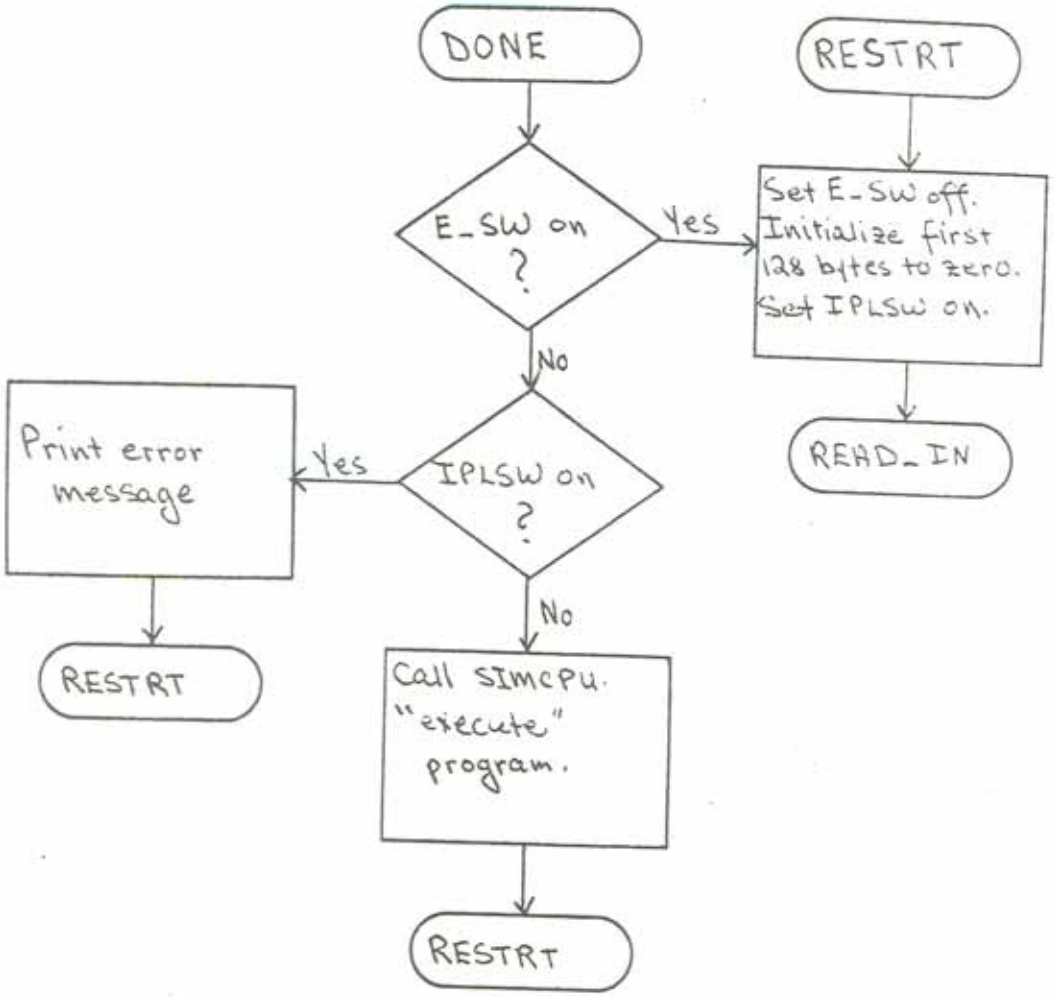


Figure C-2 continued

three are processed by the assembler; RLD and SYM records are ignored (without causing an error condition). The ESD record is used to establish the identifying name of the program being simulated. If the conventions for program preparation outlined in Appendices A and B are followed, this will be the name (label) on the CSECT card of the program being simulated. The END record signals the end of the program and causes the actual simulation process to be initiated. TXT records supply the text of the program and are loaded into the virtual core array PROG. This is a one dimensional array of aligned eight bit elements which is used to represent the core storage of the simulated computer. Figure C-2 shows the logical flow of the loading process. Note that if an invalid card type is detected or the student program does not initialize the first eight bytes of core storage (used as the initial PSW) the program will not be "executed". Also note that when the simulation of one program is finished (return from call to SIMCPU), SIMLINK reinitializes and continues to load following programs, terminating only when an end of file condition on the input occurs.

## C.3 MODULE SIMCPU

### C.3.1 STARTUP AND INITIALIZATION

On entry to this module various local variables are initialized, and two calls are made to initialization entry

points in the modules SIMIO and TRACE (SIMIO and SIMTRAS, respectively). The first operand address is forced to zero and the LPSW instruction is given control to load the initial PSW.

C.3.2 INSTRUCTION SIMULATION

Instruction simulation, in itself, is quite straightforward. The interpretation and decoding of the instructions (Figure C-3) is not quite so simple, and the actions taken after the completion of the simulation of each instruction are quite complex. Figure C-3 shows the outline of the algorithm for instruction interpretation; reference should be made, if necessary, to IBM System/360 Principles of Operation. Given the information in Figure C-3 and the diagram showing the accessing scheme for the virtual core array (Figure C-4), understanding the code which simulates the various instructions is easy (most instructions involve only four or five lines of PL/I code).

As shown in Figure C-4, the virtual core array may be accessed in six ways:

1) As a byte - 8 bit logical value
2) As a halfword - 16 bit logical value
3) As a halfword signed integer in the range -32768 to 32767
4) As a fullword - 32 bit logical value

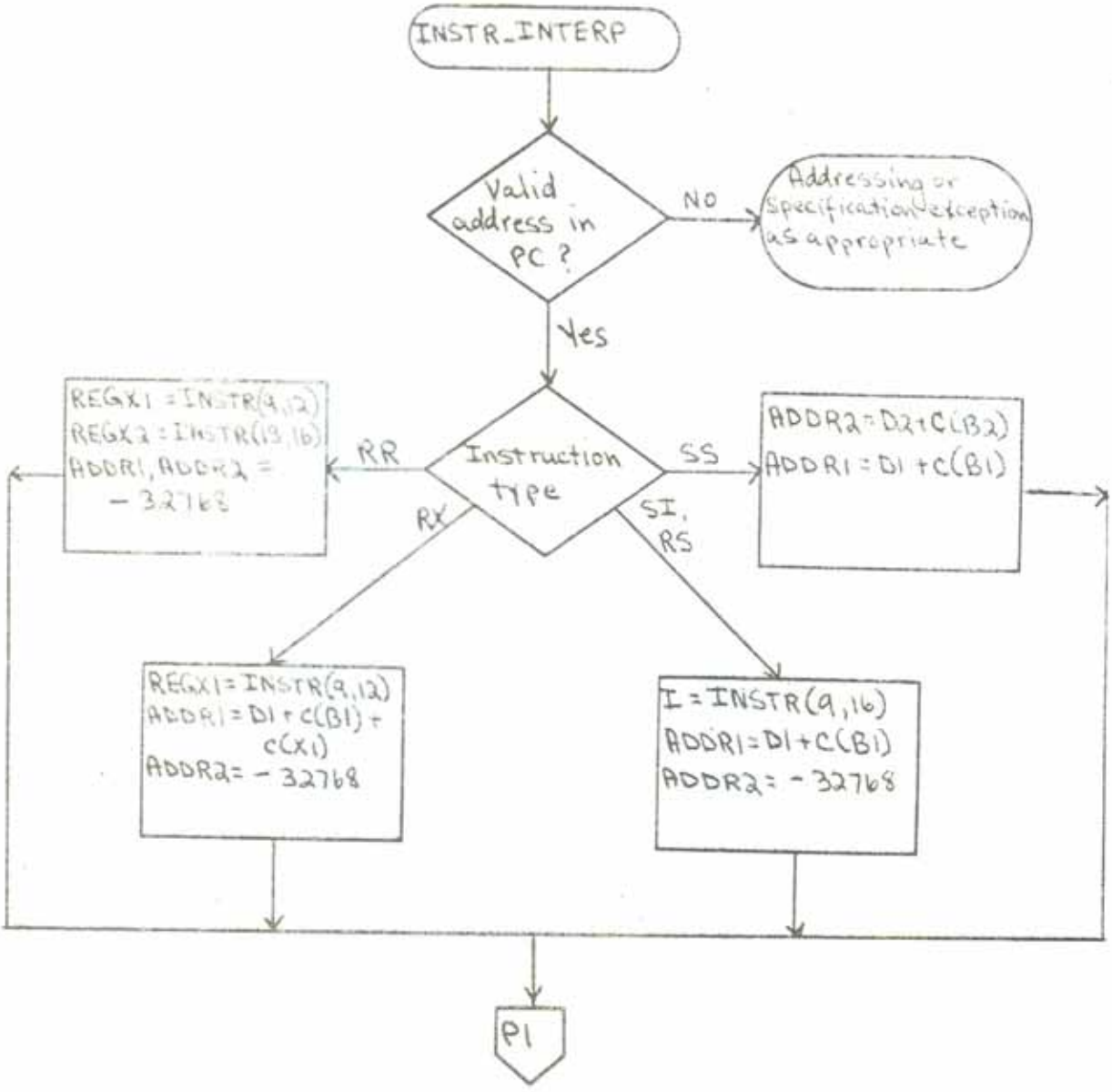


Figure C-3 : Instruction Interpretation

(continued on next page)

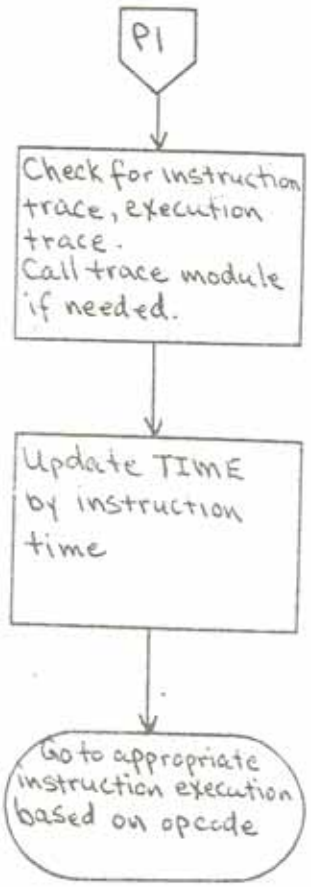


Figure C-3 continued
(continued on next page)

| Variable | Represents |
|---|---|
| ADDR1 | Address of operand 1 (if present- RX, SI, RS format) |
| ADDR2 | Address of operand 2 (if present- SS format) |
| I | Value of operand 2 (if an immediate operand) or value of count field (if SS format) |
| REGX1 | R1 specification for general purpose register operand (if RR or RX format) |
| REGX2 | R2 specification for general purpose register operand (RR format) |

Figure C-3 continued

A similar scheme is used for the general purpose registers.

Figure C-4 : Accessing the Virtual Core Array

5) As a fullword signed integer in the range $-2^{31}$ to $2^{31}-1$

6) As a doubleword - 64 bit logical value

The general purpose registers are also represented by an array (extent 16--0:15) and are accessed in the same way as 32 bit logical, 31 bits with sign integer, and 64 bit logical values. The frequently referenced procedures AL_PROT and PROT check operand locations for boundary alignment (AL_PROT only), address tracing requests, and memory protection violation (see Figure C-5).

C.3.3 POST-INSTRUCTION PROCESSING

After the execution of each instruction, and before interpreting the next instruction, the simulator must check for a variety of conditions, and perform the necessary processing associated with the conditions found.

C.3.3.1 TIMER UPDATING AND INTERRUPT SCHEDULING

The label ND (very rarely ND2) is the point where post instruction processing begins. The instruction count is updated (and checked against the allowed maximum), and the simulated real time (R_TIME) is updated by the execution time of the instruction just completed (nominally; value of temp TIME). Then the timer counter (T_TIME) is updated and a check is made to see if 3333 microseconds or more (virtual time) have elapsed since the timer was last decremented. If this condition exists, then the simulated timer (fullword at location 80) is decremented by an appropriate amount, and, if

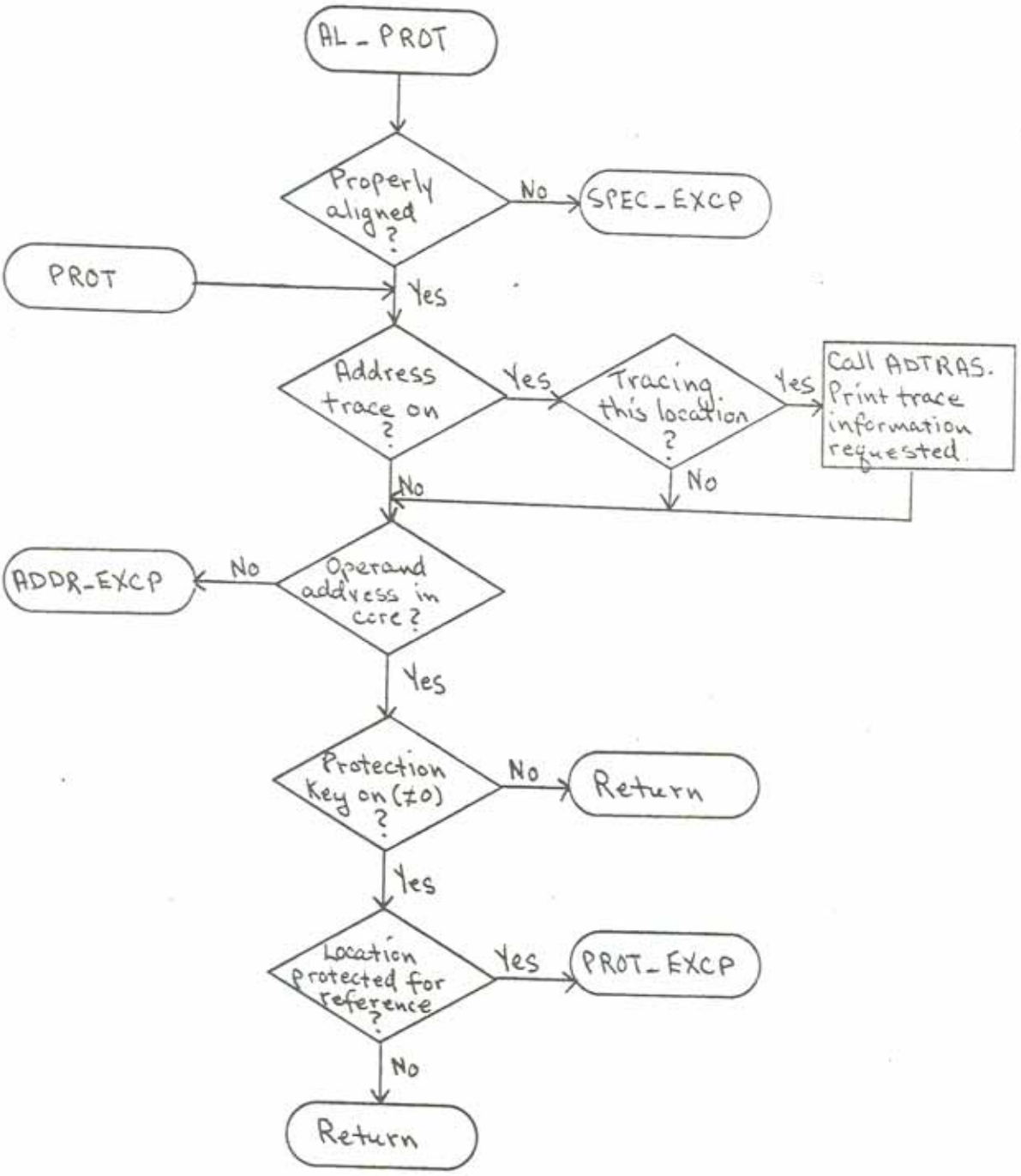


Figure C-5 : Alignment and Protection Checking

the timer has gone from a positive to a negative value as a result, appropriate interrupt processing is done. If the system mask allows external interrupts (bit 7 = 1) then the interrupt is taken immediately; if external interrupts are masked off, the interrupt is scheduled to occur as soon as the external interrupt mask bit is set to allow the interrupt.

C.3.3.2 SEARCH FOR INTERRUPT

When timer processing is completed, the interrupt and event queue is searched to see if any pending (or previously masked off) interrupt or event is due to occur. An event occurs when its scheduled time (in the queue entry) is less than or equal to the elapsed virtual time in the simulation. However, an event due to occur in time may not take place, because, for example, it is a timer interrupt and the external interrupt mask bit is zero. An event which is not an interrupt might be the transfer of a byte to or from core storage by a channel in the process of data transfer, or the occurrence of a device end for an operation initiated by a channel command word with the command chaining bit on. Figure C-6 shows the outline of this process. Note that the interrupt and event queue is maintained in sorted order by scheduled time of occurrence, and that masked interrupts are simply left at the head of the queue, and thus will be

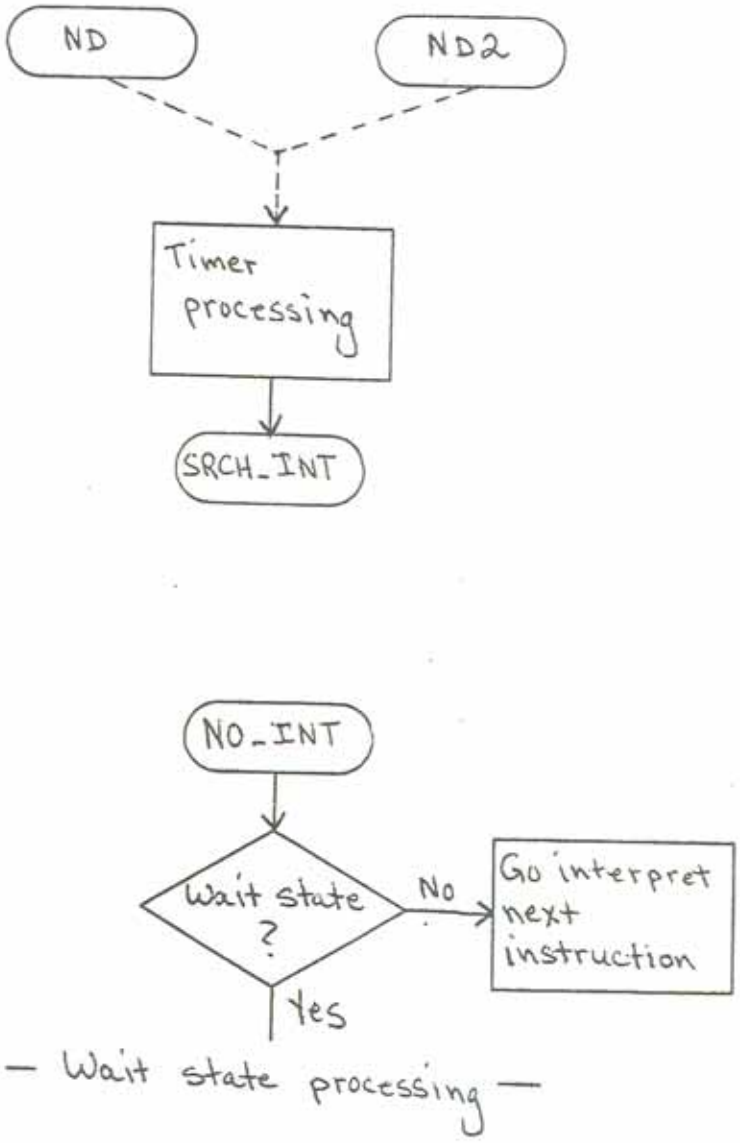


Figure C-6 : Interrupt and Event Processing
(continued on next page)

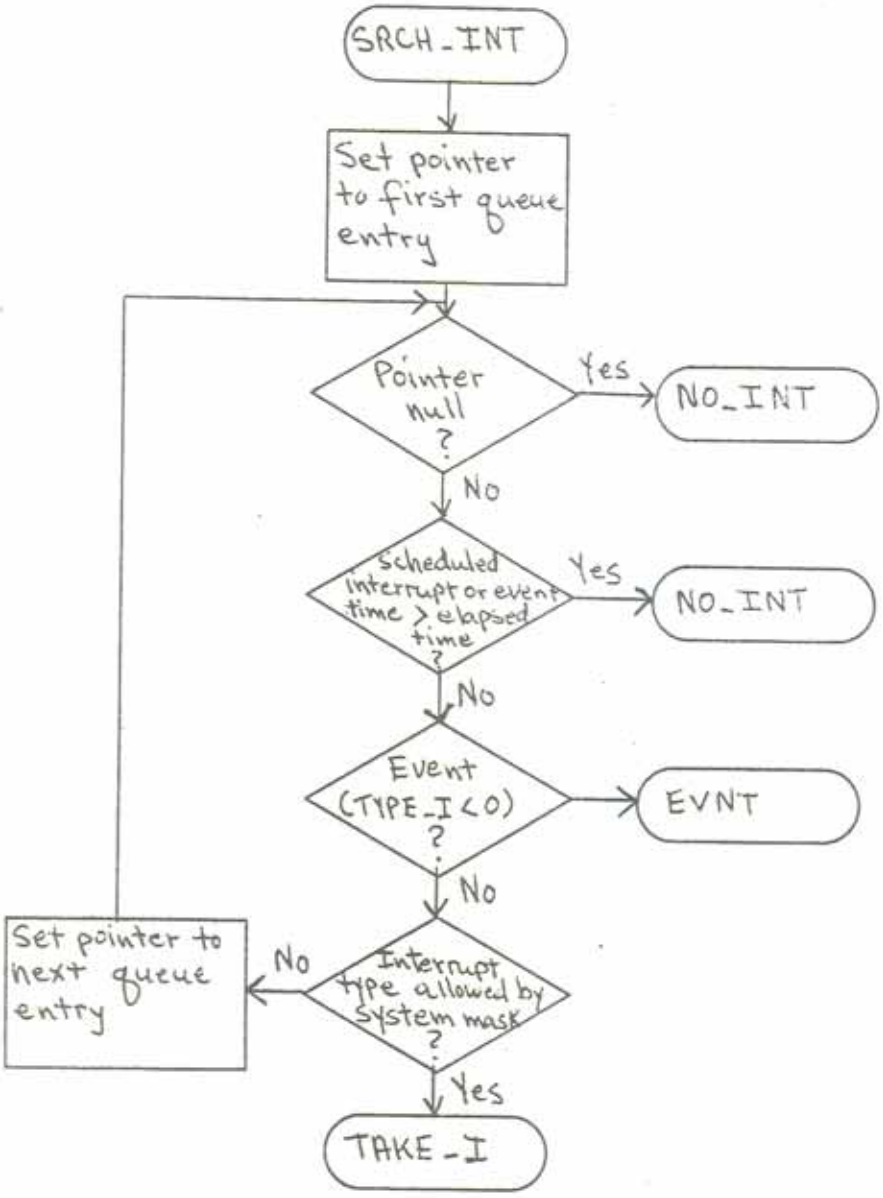


Figure C-6 continued
(continued on next page)

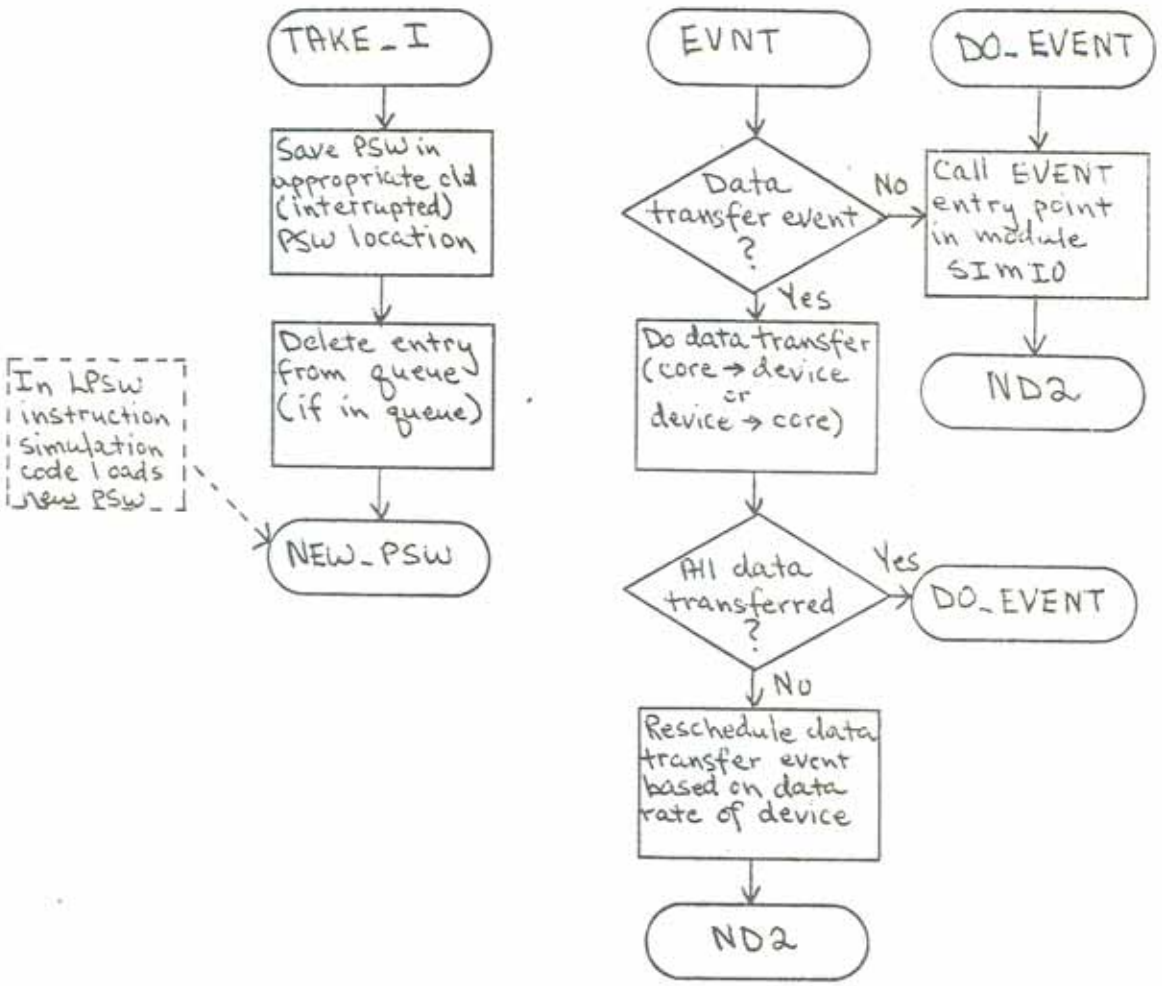


Figure C-6 continued

examined every time the queue is searched. As shown in the figure, if an interrupt or event occurs, the post instruction processing section of the simulator is re-entered at the timer processing point (effectively the start of the section). This is because time is required for an interrupt or event to take place, and thus the elapsed time must be again updated, and a possible timer interrupt checked for.

C.3.3.3 WAIT STATE PROCESSING

As shown in Figure C-6, if no interrupt or event takes place, then the wait/run state bit of the CPU is checked. If the CPU is in the run state, the instruction interpretation code is invoked and the simulator continues. If the processor is in the wait state, then a somewhat clumsy and hard to follow section of code attempts to find the next point in time when an event will occur and possibly cause processing to continue. Candidate events are a timer interrupt or some type of I/O interrupt or event. If the simulator cannot determine that there exists such an event, then the simulation is terminated and an error message is printed.

C.3.3.4 PROGRAM INTERRUPTS

Program interrupts fall outside of the structure outlined above. Most program interrupts cannot be masked, and those which can be masked do not remain pending until

enabled; they are completely ignored. Therefore a special section of code handles detected program exceptions. This routine sets the appropriate interrupt code in the program old PSW (simulated core location 40), and after appropriate processing goes to the label TAKE_I in Figure C-6. The appropriate processing may include completing an arithmetic operation in which overflow was detected or perhaps detecting that the program exception which occurred was masked off, and ignoring it altogether.

## C.4 MODULE SIMIO

This module has six separate entry points to perform different functions related to I/O. The entry points are:

| | |
|---|---|
| SIMIO | Initialize the control blocks and data sets associated with I/O device simulation. |
| SIMIOT | Called to clean up I/O simulation on termination of program being simulated. Closes data sets, flushes buffers, etc. |
| HALTIO | Performs processing associated with the HIO instruction. |
| STARTIO | Implements the SIO instruction. Initiates appropriate device activity as specified by the CCW and the state of the I/O subsystem. |
| TESTIO | Simulates the TIO instruction by examining the state of the simulated I/O subsystem |

and the specified device, and appropriately setting the condition code and the status portion of the channel status word.

EVENT Performs the processing associated with the occurrence of an I/O event.

Before attempting to understand the functioning of the I/O simulation module, it is extremely important to understand in detail the operation of the S/360 I/O subsystems. Because the S/360 can accomodate an extremely wide range of I/O devices and because the I/O capabilities of the 360 are very "powerful", I/O operations are quite complex and difficult to understand, and the occurrence of subtleties and exceptions is quite frequent. Therefore, the maintenance programmer who is not very familiar with S/360 I/O is encouraged to carefully study the I/O section of Principles of Operations[1] in conjunction with this guide and the program listing of SIMIO.

## C.4.1 I/O INITIALIZATION

As mentioned in section C.3.1, one of the first steps in initialization for simulation is to call the

---

[1] IBM System/360 Operating System: Principles of Operation, Form A22-6821

initialization entry point SIMIO in the I/O simulation module. Initialization is quite straightforward. All channels and devices are put in the available state, printer and card punch data sets are opened and identifying headers are written, and a few entries in the device specification blocks are initialized to put the system in a clean, ready for operation state.

C.4.2 I/O TERMINATION

The I/O termination function simply checks to see if any data is contained in the device specification blocks which has been output by the program being simulated, but has not yet been written to the appropriate print or punch data set. If there is any such data, it is punched or printed.

C.4.3 HALT I/O INSTRUCTION

The entry point HALTIO, in simulating the HIO instruction, first checks whether or not the addressed channel is operating in burst mode. If the channel is so operating then the device with which the channel is communicating is determined, the data transfer operation is terminated, and appropriate interrupts are scheduled. If the channel is available, then the addressed device is found and its state examined. If the addressed device is working, then any data transfer in progress (there may be data transfer in progress on the multiplexor channel without being in burst

mode) is terminated, and all interrupts which would normally occur due to device operation are scheduled to occur (with appropriate changes to reflect the HIO). The condition code is set, and the simulation of the HIO is completed. Figure C-7 shows the operation of this routine.

C.4.4 TEST I/O INSTRUCTION

The entry point TESTIO first checks for the channel working state (burst operation), and, if found, sets the condition code appropriately ($=10_2$) and returns. Otherwise, the addressed device is found and examined. If the device is available, the condition code is set ($00_2$) and a return to caller is executed. If the device is in the interrupt pending state, then the CSW information associated with the interrupt is stored, the interrupt is cleared, and the condition code is set to $01_2$ (CSW stored). If the device is working, the busy bit is set in the stored CSW, and the condition code is set to $01_2$. See Figure C-8 for further information.

C.4.5 START I/O

Upon entry to the START IO routine the channel and device are checked for availability. If one or the other is not available, action very similar to that of TESTIO for the corresponding situation is taken. If both the channel and device are available, the channel interpretation code

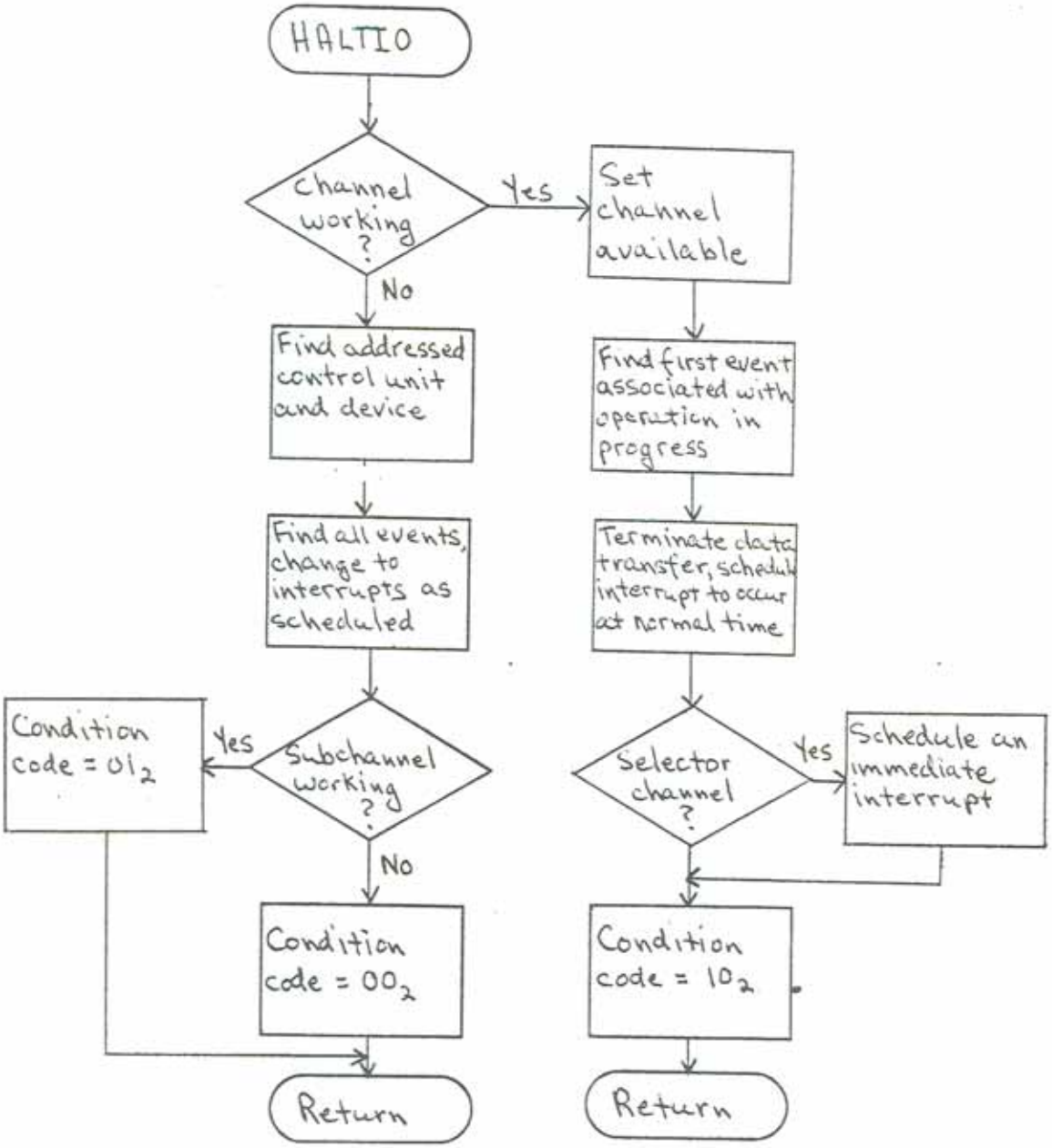


Figure C-7. Simulation of HALT I/O

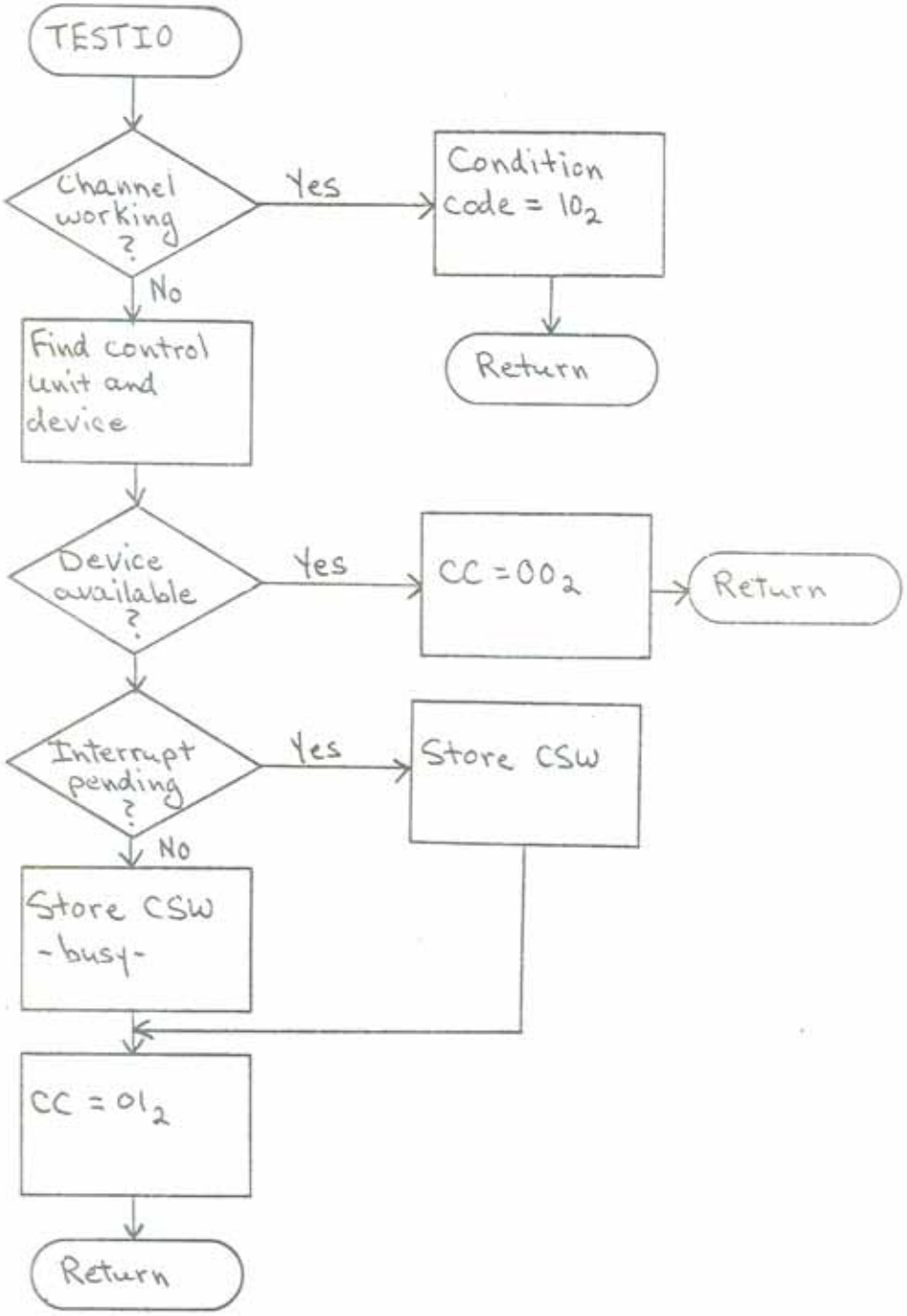


Figure C-8 : Simulation of TEST I/O

is entered.

Channel interpretation starts by checking the channel address word (location 72) for validity, and, if valid, setting the protection key and the CCW address. The channel command word location is checked against the key for fetch protection, and if no protection error is found, the CCW is fetched and the CCW address is updated. The CCW is first checked to see if it is a TIC (transfer in channel). If it is, then some validity checks are performed on the command and its occurrence (i.e., a TIC cannot start a command chain). If invalid, appropriate action is taken, and if valid the CCW address is set to the address given in the TIC. The channel interpretation code is reentered at the point where the next CCW is fetched (see Figure C-9).

If the channel command word is not a TIC, then it is checked for validity. If valid, the PCI (program controlled interrupt) flag is examined, and if set, an interrupt is scheduled. Then the various fields of the CCW are extracted and the chain data flag from the previously executed CCW is examined. If chain data is on, the parameters of the data transfer in progress are updated with the data from the new CCW, and the data transfer is continued (note that this particular action cannot result from a start I/O--no previous CCW--but only from an event; see section C.4.6).

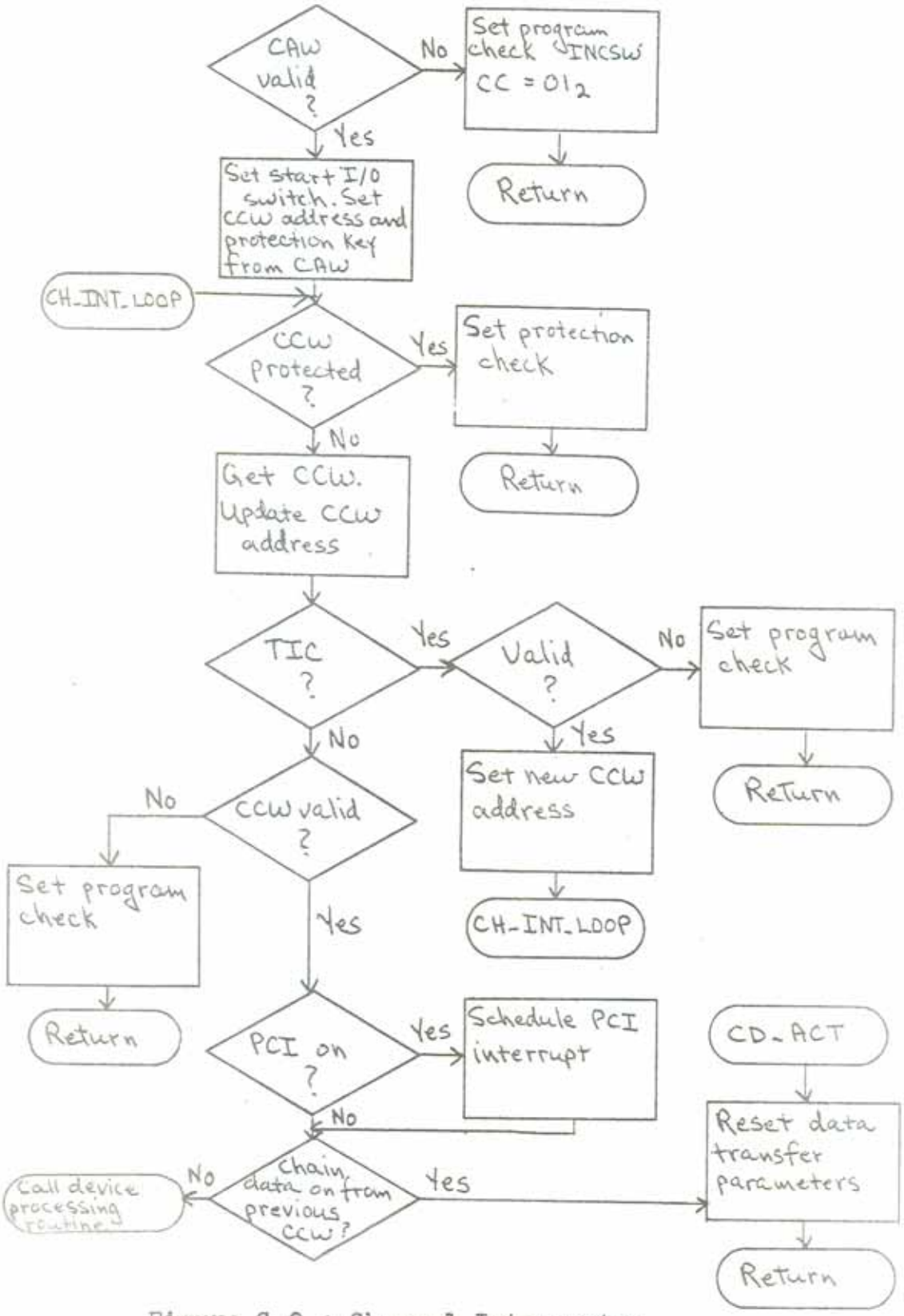


Figure C-9 : Channel Interpreter

If there is no data chaining from the previous CCW, then a processing routine for the specific device is called to initiate the operation specified by the CCW.

C.4.6 I/O EVENTS

There are three different types of I/O events. Two are data transfer events, and their occurrence is marked by the transfer of one or more bytes from virtual core storage to a simulated device, or vice versa. The remaining event type is associated with the occurrence of a channel end or device end condition which arises in the process of an input or output operation. Since the termination of data transfer sometimes (on a multiplexor channel, for example) causes a channel end condition, a data transfer event is acted upon exactly as a normal (third type discussed above) event when the last byte of data specified by the operation has been transferred (see Figure C-6). All three event types are kept in the interrupt and event queue in sorted order by time of occurrence. They are placed in the queue by the device processing routines and contain information that reflects the characteristics of the device and the operation being performed. Figure C-10 shows the PL/I declaration of an entry in the interrupt or event queue, with comments explaining the items.

```
DECLARE
  1 INT_Q  BASED(P_CI),
     2 PREV_I  POINTER,              /*NULL IF FIRST ENTRY IN
                                     QUEUE*/
     2 NEXT_I  POINTER,              /*NULL IF LAST ENTRY*/
     2 TIME_I  DEC  FLOAT,           /*SCHEDULED TIME OF
                                     OCCURRENCE*/
     2 P_DEV_DATA  POINTER,          /*LOCATES DATA AT THE DEVICE*/
     2 E_CH  POINTER,                /*IDENTIFIES ASSOCIATED
                                     CHANNEL*/
     2 E_DEV  POINTER,               /*IDENTIFIES ASSOCIATED
                                     DEVICE*/
     2 TIME_INTRVL  DEC  FLOAT,      /*FOR DATA TRANSFER EVENTS-
                                     TIME BETWEEN BYTE TRANSFERS*/
     2 TYPE_I  FIXED  BIN(15),       /*NEGATIVE FOR EVENTS*/
     2 CODE_I  BIT(16)  ALIGNED,     /*DEVICE ADR. FOR PSW*/
     2 CSW_I  BIT(64)  ALIGNED,      /*CSW ASSOCIATED W/
                                     INTERRUPT OR EVENT*/
     2 CORE_INDEX  FIXED  BIN(15),   /*CORE LOCATION FOR NEXT
                                     BYTE TRANSFER*/
     2 DEV_INDEX  FIXED  BIN(15),    /*IDENTIFIES NEXT BYTE
                                     TRANSFER AT THE DEVICE*/
     2 DATA_COUNT  FIXED  BIN(15),   /*NO. OF BYTES TO BE TRANS
                                     FERRED*/
```

Figure C-10 : Interrupt and Event Queue Entries
(continued on next page)

```
2 INCREM  FIXED  BIN(15),     /*NEGATIVE FOR READ
                              BACKWARD*/
2 CH_STAT  CHAR(1),           /*A,I, OR W. STATUS AFTER
                              OCCURRENCE*/
2 DEV_STAT  CHAR(1),          /*LIKEWISE FOR DEVICE*/
2 MASK_I  BIT(8)  ALIGNED,    /*.AND. W/ SYSTEM MASK TO
                              SEE IF INTERRUPT ENABLED*/
2 IO_PROT  BIT(4)  ALIGNED;   /*PROTECTION KEY ASSOCIATED
                              WITH OPERATION*/
```

Figure C-10 continued

The event processing routine (EVENT in SIMIO) handles only normal events (data transfers are done in SIMCPU; see section C.3.3.2). Upon entry to the routine the channel and device involved in the operation associated with the event are determined (using E_CH and E_DEV, Figure C-10) and the status bits of the CSW associated with the event are examined for unusual status (usually an error). If there is unusual status then any chaining in effect is cancelled, and an interrupt is scheduled to notify the program of the unusual condition. In the absence of unusual status, the status bits of the CSW are tested for device end. Upon device end, and data if chaining is present, the channel interpretation loop is entered (CH_INT_LOOP, Figure C-9). If command chaining is on from the previous queue, then the event is deleted from the queue, and the channel interpretation loop is entered. If there is no chaining, then the event is changed to an interrupt (to occur immediately, if enabled) and a return is made. If the event is not a device end, then if data chaining is on, the channel interpretation loop is entered. If command chaining is on, the event is deleted from the queue and otherwise ignored; in the absence of chaining, an interrupt is scheduled as above. See Figure C-11 for further detail.

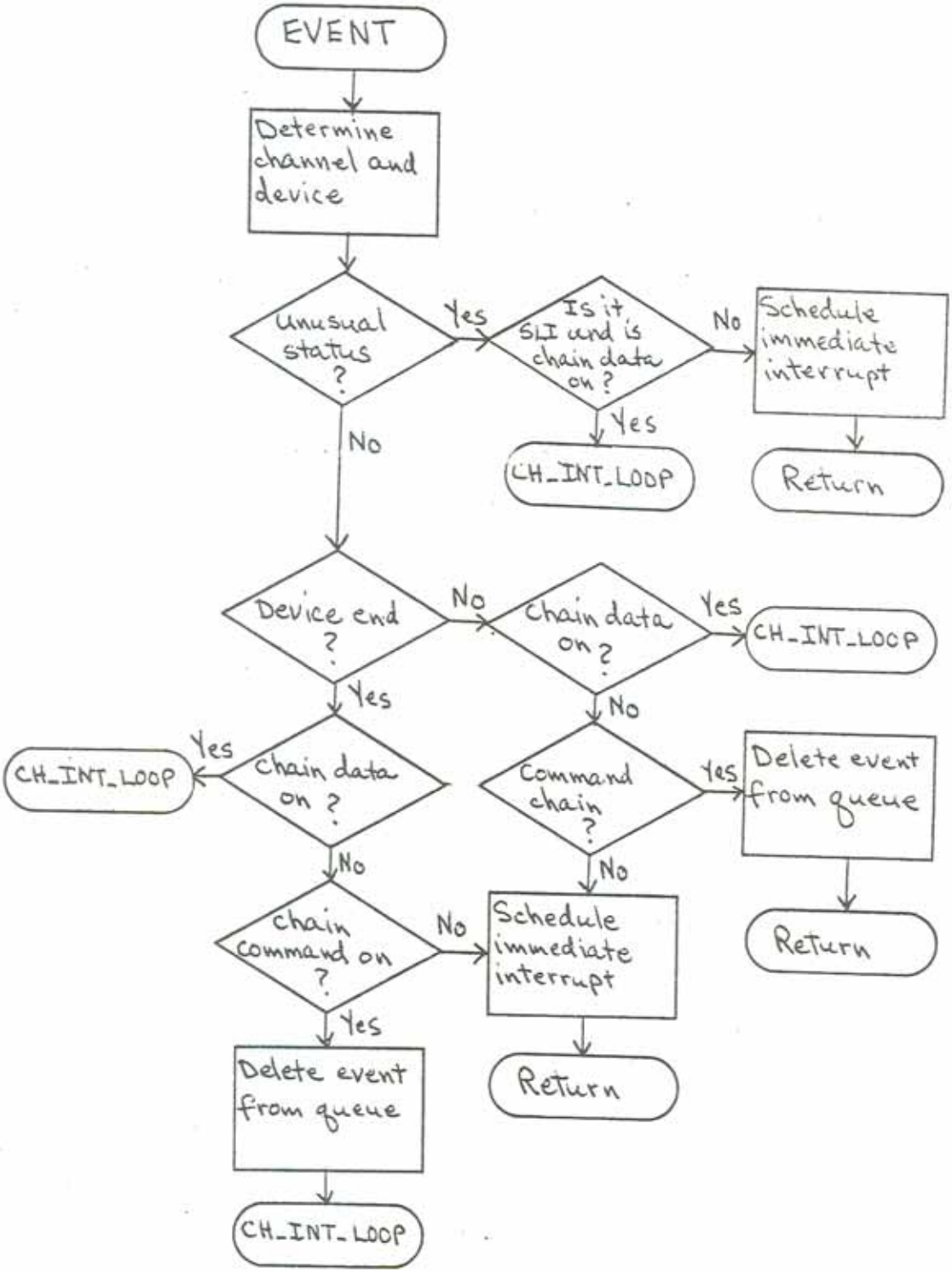


Figure C-11 : Event Processing

C.4.7 DEVICE SIMULATION ROUTINES

The details of the simulation of an I/O operation to a given device are handled by a set of routines, one for each class of devices (see section C.4.5 and Figure C-9). Each individual device is defined by a device specification block (DSB) which contains all necessary information about the device and its current state; one piece of this information identifies the particular device routine which is used in simulating the device. The device routines decode the command byte from the CCW and initiate appropriate action. Entries are placed in the interrupt and event queue as necessary. Any necessary I/O operations are performed, as in the case of a simulated card reader where an input data set of the simulator supplies the "cards" for the simulated reader. All relevant command information is checked for validity and proper sequence (there are invalid command sequences on many devices), and appropriate error action is taken if an anomaly is detected. Since these routines vary widely in form with the device simulated, an example of such a routine is shown in some detail in Figure C-12, but no attempt will be made to explain in detail the functioning of each such routine. The appropriate reference manual for a device will provide detailed information on its performance, and a complete understanding of the behavior of the device

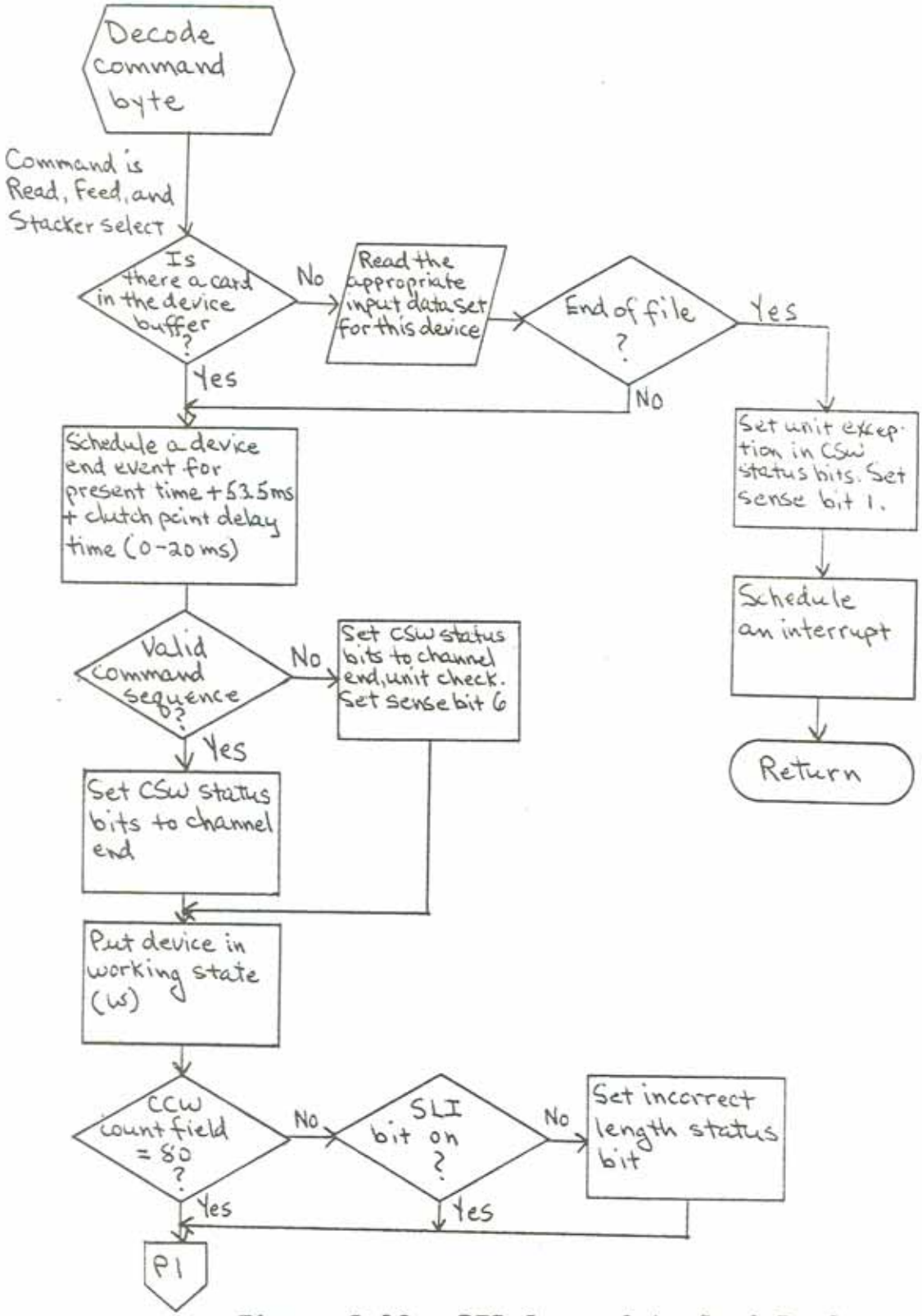


Figure C-12 : RFS Command to Card Reader
(continued on next page)

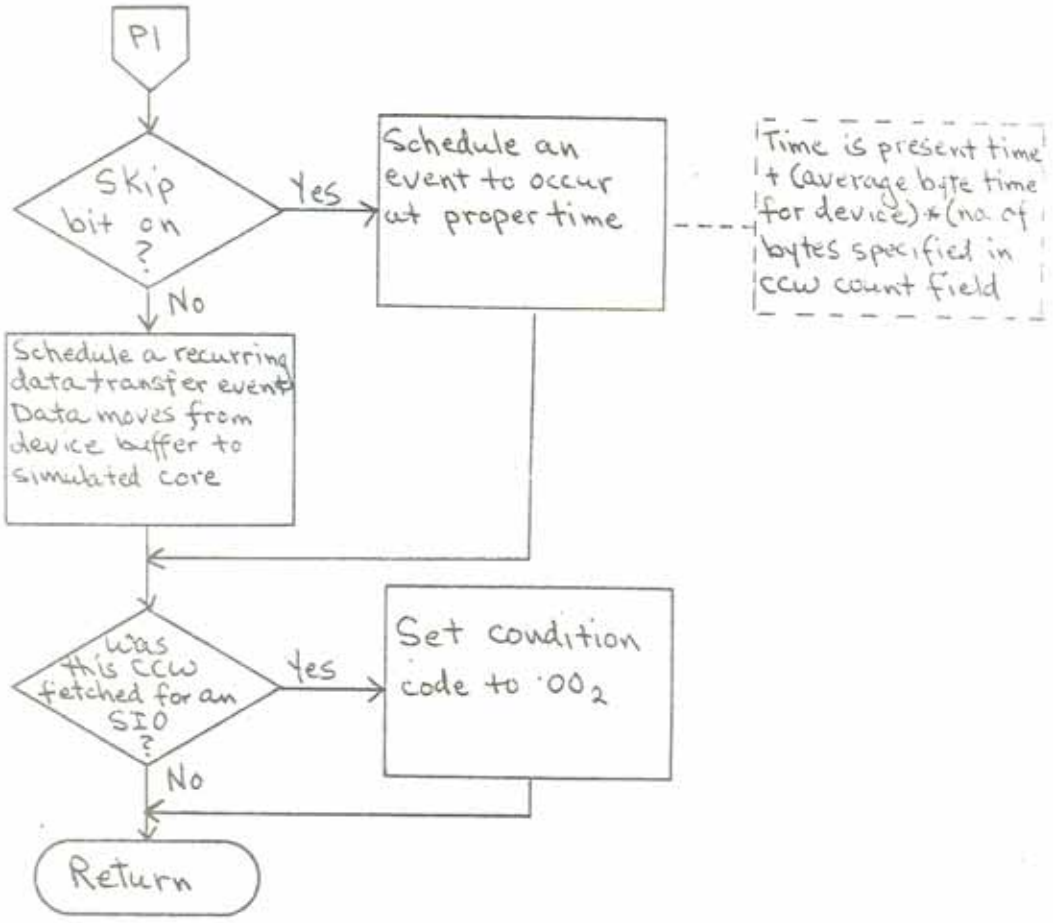


Figure C-12 continued

will tend to lead to an understanding of the device simulation routine.

## C.5 TRACE MODULE

There are 9 entry points to this module, two to process dynamic trace requests by the simulated program, and 6 to do the formatting and printing associated with a trace message. The entry points are:

TRACE - enables a trace condition in accordance with information supplied in the trace request

NO_TRAS - turns off any existing trace conditions of the type specified

BTRACE - called to do printing associated with a branch trace

ITRACE - called to do an instruction trace

NTRACE - called to do an interrupt trace

ADTRAS - called to do an address trace

ETRACE - called to do an execution trace

CTRACE - called to do a channel trace

SIMTRAS - initializes for simulation

The processing done by the TRACE module is not particularly interesting or difficult to understand. With a few exceptions, it consists of getting such and such a field to print in character position n, and thus is painfully detailed but conceptually unchallenging. Discussion will therefore be brief.

C.5.1 DYNAMIC TRACE REQUESTS

A trace request extracts information compiled into the program code by the Trace macro instruction, checks it for validity, and makes an entry in the trace queue, a list of enabled trace conditions. Figure C-13 shows the data format in the program code, and Figure C-14 gives the PL/I structure declaration of an entry in the trace queue. The transformation from one to the other is almost one for one, and quite obvious. One item of interest is that if invalid data is found in a trace request, it is assumed that the program being simulated has erroneously modified instruction locations. In this case, an attempt is made to find the end of trace request flag, and if it can be found, the invalid trace request is ignored; otherwise, an operation exception (program interrupt for invalid op-code) is taken. There is a separate routine to process each type of trace request, but they are very small (about five PL/I statements) and are necessary only because a different

```
         DS    0H          ALIGN ON HALFWORD BOUNDARY
         DC    X'02'       TRACE OP-CODE
         DC    X'0n'       TRACE TYPE:
*                              0:  BRANCH
*                              1:  INSTRUCTION
*                              2:  ADDRESS
*                              3:  INTERRUPT
*                              4:  EXECUTION
*                              5:  CHANNEL
*                              6:  UNUSED
*                              7:  UNUSED
*                              8:  DUMP
         DC    XL2'id'     ADDRESS,OPCODE,INT.TYPE,ETC.
         DC    BL2'status bit switches'    BIT SWITCHES FOR
*                                          STATUS DUMPS.
         DC    BL2'register bit switches'
*FOLLOWING PAIRS ARE CORE DUMP SPECIFICATIONS
         DC    Y(address)  FIRST ADDRESS TO BE DUMPED
         DC    AL1(n,s)    n = NUMBER OF WORDS DUMPED
*                          s = indirect switch
*THERE MAY BE UP TO EIGHT PAIRS, TERMINATED BY THE
*FOLLOWING SENTINEL.
         DC    XL2'8000'   TERMINATOR
```

Figure C-13 : Trace Macro Data

```
DECLARE
     1  TRACE_LIST  BASED(P_C),
        2  PREV  POINTER,
        2  NEXT  POINTER,
        2  TYPE  FIXED BIN(15),
        2  ID  BIT(16)  ALIGNED,
        2  STATUS  BIT(16)  ALIGNED,
        2  REGS  BIT(16)  ALIGNED,
        2  DUMP_OPTIONS(8),
           3  ADDRESS  FIXED  BIN(15),
           3  DCOUNT  FIXED  BIN(15);
```

Figure C-14 : Trace Queue Entries

internal indicator for each trace type is used to indicate that a trace condition is enabled.

A Traceoff command is processed at entry point NO_TRAS, and simply removes from the trace list the particular instance of the trace type specified, or, if ALL of the given type were specified, then every instance.

C.5.2 TRACE OUTPUT ROUTINES

The six entry points associated with trace output all do very much the same thing. The trace list is searched for the entry associated with the trace condition. Note that because the Traceoff command only removes the entry from the trace list, the internal indicator which flags a trace condition may still be set. In this case, when the list is searched, no corresponding entry will be found, and the output routine will then reset the internal indicator and return. In the more normal case, where an entry is found in the trace list, then a call is made to an internal procedure (TDUMP) which formats and prints the trace output as specified by the information in the trace queue entry.

It should be noted that the snapshot (DUMP) type is something of an exception. Because the dynamic trace request is, in effect, the trace condition in this case, a slightly different sequence of events results. However,

examination of the code will show that no difficulties are involved. Using existing code and procedures, a DUMP trace request:

- sets up an entry in the trace list in the normal way
- calls TDUMP in the normal way to print the information requested
- enters the NC_TRAS routine in an appropriate place to delete from the trace list the entry created in the first step above
- returns to caller (SIMCPU).

# BIBLIOGRAPHY

CMS Program Logic Manual, Form GY28-0591.

Control Program-67/Cambridge Monitor System User's Guide - IBM Publication.

CP-67 Program Logic Manual, Form GY20-0590.

IBM System/360 Component Descriptions - 2841 and Associated DASD, Form GA26-5988.

IBM System/360 Operating System: Assembler(F) Programmer's Guide, Form GC26-3756.

IBM System/360 Operating System: Assembler Language, Form GC28-6514.

IBM System/360 Operating System: Job Control Language Reference, Form GC28-6704.

IBM System/360 Operating System: Job Control Language User's Guide, Form GC28-6703.

IBM System/360 Operating System: Linkage Editor and Loader, Form GC28-6538.

IBM System/360 Operating System: Linkage Editor(F) Program Logic Manual, Form GY28-6667.

IBM System/360 Operating System: Loader Program Logic Manual, Form GY28-6714.

IBM System/360 Operating System: PL/I Language Reference Manual, Form GC28-8201.

IBM System/360 Operating System: PL/I(F) Programmer's Guide, Form GC28-6594.

IBM System/360: Principles of Operation, Form GA22-6821.

IBM System/370 Model 155 Functional Characteristics, Form GA22-6942.

IBM System/370: Principles of Operation, Form GA22-7000.

IBM 2821 Control Unit: Component Description, Form A24-3312.