DOCUMENT NO. Y240A302M0810 REV —

ENGINEERING TECHNICAL REPORT

SKC3120 SIMULATOR REFERENCE MANUAL

JULY 1976

SINGER
AEROSPACE & MARINE SYSTEMS

DRR NO.01911(NP)

TOTAL PAGES 102

THE SINGER COMPANY • KEARFOTT DIVISION • 1150 MCBRIDE AVENUE • LITTLE FALLS, N. J. 07424

F4200-4 2/72

SKC3120

SIMULATOR

REFERENCE MANUAL

Prepared by:

DEPARTMENT 5760

ENGINEERING PROGRAMMING AND COMPUTATION

JULY 1976

REVISION RECORD

REV	DESCRIPTION	APPROVAL AND DATE
-	RELEASE	JULY 1976

REV	-																				-
PAGE	COVER																				OTHER PAGES
REVISION SYMBOL OF REVISED PAGES																					

ABSTRACT

This manual contains user information pertaining to the SKC3120 Simulator Program as well as a complete description of all user controls and service requests. The SKC3120 Simulator utilizes the facilities of the host machine to create a functional reproduction of the SKC3120 Computer. User controls and a service facility are provided to permit effective control of the Simulator.

Proper operation of the SKC3120 Simulator requires knowledge of the SKC3120 computer and the SKC3120 (KAL31) Assembler language. Therefore, the following documents should be used in conjunction with the Simulator Reference Manual:

Y240A301M0810	SKC3120	Assembler Language Reference Manual
Y240A300M0810	SKC3120	Principles of Operation
Y240A301M0811	SKC3120	Assembler/Linkage Editor/Simulator User's Manual.

Since this Simulator was designed to be largely machine portable, it can be easily adapted to run on a variety of host machines.

This manual is Host Machine independent and describes the user control commands and service routines which may be used with any Host Machine.

Specific operating instructions are supplied for the IBM 360/370 in the Host Procedures Manual.

Y240A301M0812	SKC3120	Host Procedures for the IBM 360/370 Computers

TABLE OF CONTENTS

TABLE OF CONTENTS (CONTINUED)

1. INTRODUCTION

The Singer Company, Kearfott Division has developed a powerful Simulator program to complement the development of the SKC3120 Computer and the SKC3120 Macro Assembler. The program is executed on a host machine computer system and is intended for use by customers, or potential customers, who do not have an SKC3120 Computer System available for program checkout, or who desire to avail themselves of the extensive user control and report facilities not normally available with the SKC3120 Computer itself.

The Simulator accepts as input the SKC3120 Macro Assembler's load module, which is normally on magnetic tape. It loads this module into a simulated SKC3120 memory within the host machine's memory. The Simulator is capable of interpreting and executing SKC3120 instructions from this simulated memory; hence, it is termed an interpretive Simulator. It processes each data word and all arithmetic and logical operations with bit-by-bit accuracy.

The SKC3120 Simulator consists of three major parts: the SKC3120 hardware models (Central Processing Unit (CPU), Input/Output (I/O), Interrupt and Memory models), a service facility, and a user defined FORTRAN Control Program (FCP). The FCP may be structured to satisfy a wide variety of user requirements and may, at the highest level, take the form of an operating system, which would represent the Simulator control and forcing function for the tasks to be performed.

Simulator output files contain the configuration file,the user control input file and the on-line diagnostics file. Each of these files is formatted and printed as the normal output of the Simulator.

Trace output produced by the Simulator provides information pertaining to the instruction location, mnemonic, operand address, index register, arithmetic registers and the time status of simulated programs. The Report Program Generator (RPG) is used to format and print out this information when requested by the user.

The Simulator Trace File may contain voluminous output, and therefore the user may wish to put the raw data on magnetic tape and generate the appropriate reports on another facility (e.g. an off line mini-computer). To provide this mode of operation, the source of the RPG will be provided to facilitate its conversion to the desired computer.

The remainder of this reference manual is divided into three major parts. Sections 2 and 3 discuss Simulator capabilities and describe the environment in which the Simulator functions. Section 4 is presented as a tutorial; it is intended to instruct the user in a step-by-step manner so that he may utilize the Simulator with the default FCP. Numerous examples are provided, showing typical Simulator input control statements and the resultant output. Sections 5 through 7 contain all the information necessary to enable a user to design and implement his own FCP and to construct his own input/output model programs.

2. SIMULATOR CAPABILITIES

Effectively, the SKC3120 Simulator is a functional reproduction of the SKC3120 Computer. It possesses a Central Processing Unit (CPU), Input/Output (I/O), Interrupt hardware and memory models.

In addition, the Simulator provides control and service capabilities to the user who may thereby effectively and efficiently control the simulation process and request Simulator services that are essential to the successful implementation of user program modules.

The mechanism that is employed by the user to initiate and complete control functions and request Simulator services is termed the FORTRAN Control Program (FCP).

Essentially, the FCP is a user defined FORTRAN module that has access to the state of the SKC3120 machine (CPU, I/O, interrupt hardware and memory) and thereby provides unlimited control capabilities to the Simulator user. The FCP, in the control sense, is the functional equivalent of a pre-programmed Computer Control Unit (CCU). In addition, the FCP possesses capabilities not present in the physical CCU; e.g., the contents of all the CPU registers are accessible through the FCP but not the CCU.

Access to the Simulator is through Control Command card images, input to the Simulator via the FCP by the Host Machine FORTRAN I/O processing routines. Standard FORTRAN notation is used in the descriptions of the data cards. The structure of FORTRAN records and the form of the data fields within the records are described to the extent required by the user to recognize the requirements of the data to be processed. Further explanations are given in those cases where there is an apparent conflict. Typical would be the format description requiring 'A' (character) format, but commentary is included requiring the input to be in hexadecimal. In these cases, the character string is processed further by an internal subroutine or function sub-program which converts the input string to binary for internal representation by considering the input as a hexadecimal number of the specified length.

The logic and decision making functions that the user would normally make at the CCU can be programmed into the FCP and are limited only by the ingenuity of the Simulator user.

The Simulator provides diagnostic capabilities that are much more extensive than those provided by the actual hardware. In addition to supervision of the operational program logic and arithmetic routines in the check-out phase, the Simulator also performs a self test on the control actions and service requests to validate all user activities.

Finally, the Simulator permits the introduction of system dynamics models and provides communication links to the models, thereby providing an environment for realistic SKC3120 program validation. The user defined models may be simple ones, modeling only those factors which are considered essential, or can be as elaborate and accurate a representation of the actual system dynamics as desired.

3. SIMULATOR ENVIRONMENT

The job control language requirements vary with each computer installation. The user is referred to the appropriate host procedures manual for the JCL which is to be used at his installation.

A pictorial representation of the environment in which the Simulator operates is shown in the Simulator functional flow, Figure 3-1. This shows the resources (primary files) which are required by the macro assembler, the Linkage Editor and the Simulator, and illustrates the steps which are required to simulate an SKC3120 program.

Note that the functional flow is greatly simplified if the user does not require I/O models and can use the default FCP. When it is necessary for the user to provide his own FCP and/or I/O models, it is not necessary for him to recompile and relink these programs each time that he executes the Simulator. Once these user supplied programs have been debugged, a Simulator load module which contains them may be saved and executed in place of or in addition to the default Simulator.

Source program input card images may be supplied on any medium compatible with the host machine - usually cards, magnetic tape or disc. The same is true for the Simulator input command statements, although in practice these will almost always be supplied on cards.

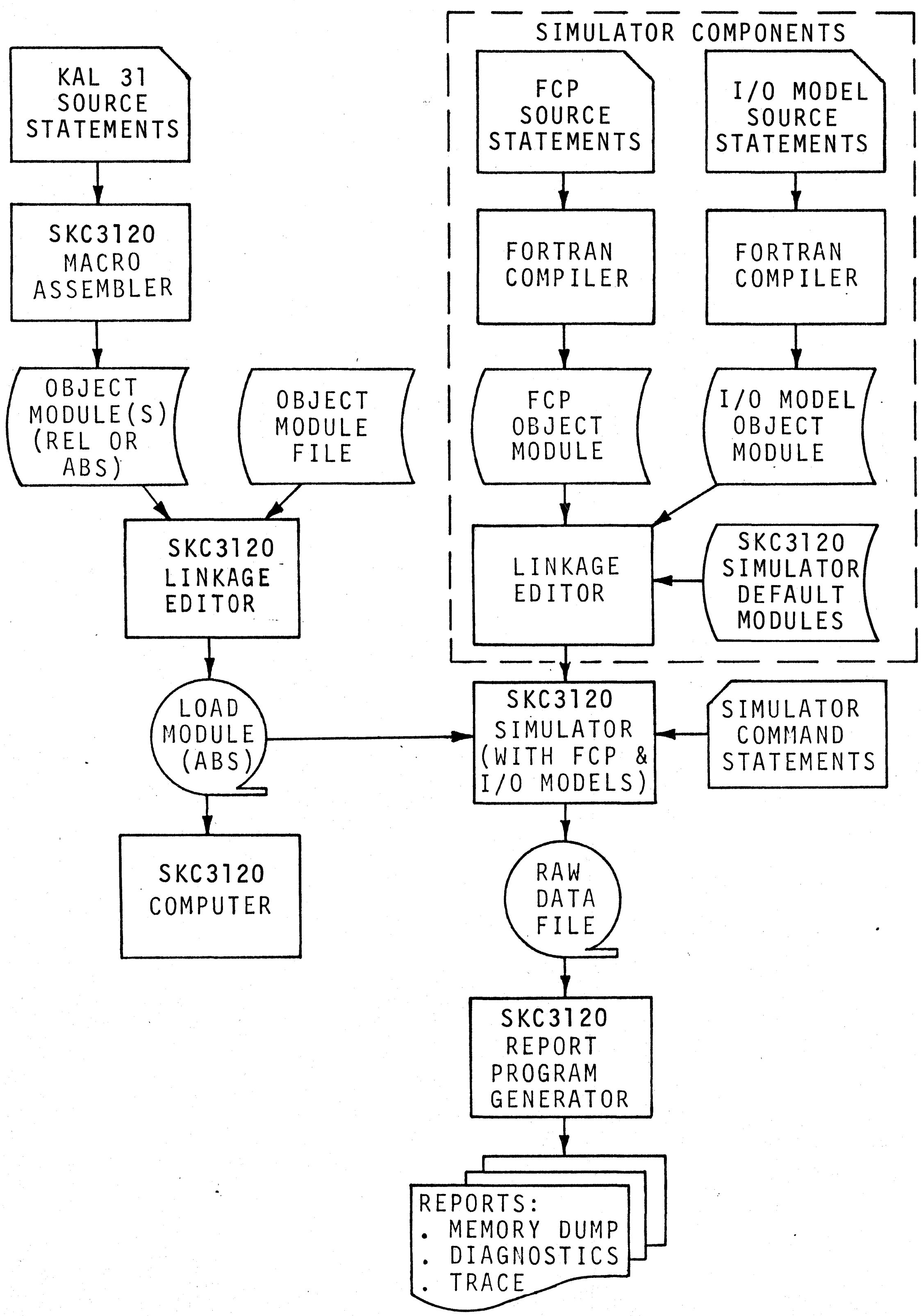

FIGURE 3-1. SKC3120 SIMULATOR FUNCTIONAL FLOW

4. DEFAULT FCP

4.1 DEFAULT FCP FUNCTIONS

The default FCP, if employed, provides a convenient mechanism by which the user is able to control the simulation process. The capabilities present with the default FCP, enable the user to simulate the SKC3120 program with a minimum effort by using simple,one-line commands which are then interpreted by the Simulator default FCP and acted upon in sequence.

These extensive controls and report facilities are provided to permit the user to simulate the SKC3120 operational program with capabilities that are in addition to those normally supplied with the SKC3120 computer.

The Simulator is, in effect, a reproduction of the SKC3120 computer and can be divided into three sections, the Central Processor, Memory and Input/Output . The controlling capability in the Simulator environment is furnished by the FORTRAN Control Program.

The Default FCP enables the user to define breakpoints, input or output data to memory, establish check points, execute the program, load memory from the output of the Linkage/Editor, define the period of the real time interrupt, selectively trace portions of the simulated program, output messages to the trace file and to terminate the simulation.

Additional control in a dynamic environment may be exercised by generating a user supplied FCP as discussed in Section 6. In general, the default FCP supports control and service directives, execution time computations, and operational program interruption.More detailed dynamic control and service request actions, I/O modeling, expansion of memory and re-definition of the memory structure, expansion of breakpoint list size, more elaborate error handlers, elapsed time and instruction count controls are possible through user definitions of the FCP and I/O model programs.

4.2 CONTROL COMMANDS

The medium by which the user may communicate the desired control and service requests to the default FCP is usually card input. The commands must be in a specified format. Table 4-I lists each of these commands with a brief functional description, and indicates the section where the command is more fully described.

The control statements for the various control and service requests and the required fixed formats are described in detail in Section 4.3. This includes examples and describes the results or output produced by the command. The default FCP listing appears in Appendix A.

Where command statement formats are shown, the following method is used:

```
1...5...10...15...20...25...30...35 Format.f.
XXXXXXXX aaaa.... bbb.....          A8,1X,A8
```

The numbers represent card columns. Immediately below these numbers are the control statement commands and, if required, one or more operands. Characters which appear in upper case must be present exactly as shown. User supplied values are substituted for operands which appear in lower case. These operands are described after the format description. The format shown at the right is the FORTRAN fixed format specification that is used for reading the command statement and is furnished to give the user an insight to the Command card handling by the FORTRAN I/O processors.

Several commands require additional information which must appear on one or more successive input statements according to format specification; e.g., the $DUMP command must be followed by the dump region specification(s) and a delimiter.

In all cases,the delimiter as specified in the descriptions of the control commands is the mnemonic 'END' starting in column 1.

An example of the $DUMP command is furnished in the following statements.

```
1...5...10...15...20...25...30...35
$DUMP
     00000200 000005FF
     00001000 000011C0
END
```

TABLE 4-I DEFAULT FCP COMMANDS

Command	Functional Description	Section
$BREAK	User definition of the set of simulation break points and related action	4.3.4
$CHECK	Request to take a checkpoint	4.3.10
$CHKSM	Request to calculate memory check sum	4.3.12
$DUMP	Request to dump simulated memory region(s)	4.3.7
$EXEC	Control to initiate or resume simulation at a specified program counter or at the current program counter value	4.3.2
$INPUTFX	Request to patch using 'FIX' conversion	4.3.14
$INPUTHX	Request to patch using hex input	4.3.13
$INTRPT	User definition of interrupt flag (INTFLG)	4.3.9
$IODEF	User definition of real-time clock interrupt number, memory speed and real-time clock period	4.3.8
$NOTE	User message	4.3.5
$OUTPTFX	Request to output using 'FIX' conversion	4.3.16
$OUTPTHX	Request to output a value in hex	4.3.15
$RESTR	Request to restart simulation from last checkpoint	4.3.11
$TAPIN	Request for computer load	4.3.1
$TERM	Request to terminate simulation	4.3.6
$TRACE	User definition of trace flag (TRCFLG)	4.3.3

TABLE 4-1 DEFAULT FCP COMMANDS (continued)

Command	Functional Description	Section
$INPUTFL	Request to patch using floating input	4.3.17
$OUTPTFL	Request to output using float conversion	4.3.18
$SETAHX	Sets the A register using hex option	4.3.19
$SETAFX	Sets the A register using fix option	4.3.20
$SETAFL	Sets the A register using float option	4.3.21
$SETBHX	Sets the B register using hex option	4.3.22
$SETBFX	Sets the B register using fix option	4.3.23
$SETX	Sets the index register	4.3.24
$SETR	Sets a selected CPU register (B1,B2,IXR)	4.3.25

4.3 CONTROL COMMAND DESCRIPTIONS

4.3.1 $TAPIN Command

The $TAPIN (tape-in) command permits the user to request a computer load. This causes the Simulator to read the absolute load module into simulated memory. This is the normal way of initializing memory and corresponds to reading a perforated tape into the memory of an actual SKC3120 Computer. An alternate method for accomplishing this is via the $INPUTHX and $INPUTFX commands, which would correspond to entering a program into the SKC3120 Computer by using the switches on the CCU. Note that this alternate method may be viable for certain small programs, since it obviates the need for executing the SKC3120 Macro Assembler and Linkage-Editor programs.

The format of this command statement is shown below. There are no operands and no additional statements are required.

```
1...5...10...15...20...25...30...35   Format...
$TAPIN                                A8
```

Normally, the $TAPIN command will appear before the first $EXEC command, to assure that a program is in simulated memory before attempting to execute it. If additional $TAPIN commands appear, each will cause the next segment of the absolute load module to be read into simulated memory. If an overlay results, it is transparent to the Simulator and the user must be aware that the overlay might affect the Command statements that follow.

4.3.2 $EXEC Command

The $EXEC (execute)command causes the Simulator to begin "execution" of the program being simulated. It also allows the user to specify the address at which this execution is to begin or resume.

The format of this command is shown below. There is one optional operand. No additional statements are required.

```
1...5...10...15...20...25...30...35   Format...
$EXEC    loc.....                     A8,1X,A8
```

loc..... specifies the location at which execution is to begin or resume. If this operand is omitted, then simulation starts at the default program counter value or resumes using the current program counter value.

A location must be specified as an eight digit hexadecimal number,right justified and padded on the left with zeros

When the Simulator reads a valid $EXEC command, processing of the input commands is suspended and the Simulator begins to execute the problem program. Execution will continue until a user specified breakpoint is reached (see Section 4.3.4). The Simulator then takes the action specified by the user . The user is able to simulate a program with only the $TAPIN and $EXEC commands, but, he will not be able to exercise any control over the progress of the simulation, nor will he get much meaningful output. Operating in such a mode corresponds to loading a program into an actual SKC3120 Computer and executing it.

The commands described in the following sections provide the user with the means of getting much more useful output from the Simulator and for controlling the execution of his program.

4.3.3 $TRACE Command

The $TRACE (print state of the CPU) command permits the user to define the setting of the trace flag, TRCFLG.

The format of this command is shown below. There are no operands, but a second statement is required to indicate the state desired for the trace flag.

```
1...5...10...15...20...25...30...35   Format...
$TRACE                                A8
f                                     I1
```

f specifies the value to be assigned TRCFLG:
If f = 1, tracing is turned on.
If f = 0, tracing is turned off.

When the trace option is on (TRCFLG = 1), the Simulator will produce output information for each instruction which is executed. The RPG will use this information to produce a trace report. Table 4-II describes the items included in the Trace Report. The trace feature of the Simulator should generally be used only for relatively short programs or for small portions of larger programs, since a large volume of data can be generated by a short amount of simulated time. The trace can be turned on and off by using the $BREAK command which is described in Section 4.3.4.

A selective trace can be generated by setting a break point at the start of the trace area and another at the end of the trace area. The user can then turn trace on at the first breakpoint and turn it off at the second, so that each time the code to be traced is entered, the tracing will be activated.

TABLE 4-11 TRACE REPORT HEADINGS

OPCD	Operation code mnemonic of the instruction executed.
ILOC	Instruction location counter value (hex address).
PC	Program counter value after instruction execution.
IR	Contents of the instruction register.
SR	Status register contents after execution.
IMR	Interrupt mask register contents after execution.
A	A register contents after execution.
B	B register contents after execution.
OAR	Operand address register.
XR	Index register contents after execution.
B1	Base register 1 contents after execution.
B2	Base register 2 contents after execution.
IXR	Inactive Index register contents after execution.
CBO	Carry bit after execution.
DATUM	Contents of address specified by OAR.
MICRO SEC.	Cumulative execution time in decimal micro-seconds.

Unless otherwise indicated, all of the above items are printed in hexadecimal.

4.3.4 $BREAK Command

The $BREAK (break point) command permits the user to add or delete break points in the array BRKLST. This array contains zero or more program counter values at which the Simulator is to take special action. The default FCP provides an array, sufficient for 25 break points. The user specifies each break point and the action associated with it.

The format of this command is shown below. There are no operands on the $BREAK statement, but one or more break points and a delimiter are specified on successive statements.

```
1...5...10...15...20...25...30...35    Format...
$BREAK                                 A8
     loc..... actn                     5X,A8,1X,A4
END                                    A4
```

loc..... specifies the program counter value to be used as a break point.

A location must be specified as an eight digit hexadecimal number, right justified and padded on the left with zeros

actn specifies the action associated with the break point. The following values are permitted:

"TERM" Simulation will be terminated.

"DLTE" Delete break point from BRKLST.

"   " FCP reads additional command statements.

Some examples of the use of the $BREAK command are presented here to clarify its function. A description of each command appears to the right of the command.

```
1...5...10...15...20...25...30...35
$TAPIN                                    Read in load module
$BREAK                                    Set break points
     00005000                             Read more commands
     00005BE4 TERM                        Terminate if get to 5BE4
END
$EXEC    00005B80                         Begin execution at 5B80
$NOTE                                     User message (see 4.3.5)
PROGRAM REACHED 5000
$BREAK                                    Set break points
     00005BE4 DLTE                        Delete this addr from list
     00005C88 TERM                        Terminate if get to 5C88
END
$EXEC                                     Continue execution at 5000
```

This sequence of commands effectively breaks the program simulation into two parts. In the first part, the program is executed from 5B80 (its starting address) to 5000, at which time the Simulator stops execution at the completion of the last instruction before executing the instruction at 5000 and reads additional command statements. If the program counter had a value of 5BE4 during this time, the simulation would have been terminated. Assuming the program counter got to 5000, the $NOTE command (described in Section 4.3.5) would be processed. Then the break point at 5BE4 would be removed from the list; either we no longer care if the program counter has this value or we just want to maintain the minimum number of entries in BRKLST. Finally, a break point is set to terminate the simulation at program counter value 5C88 and execution is resumed where it was suspended.

4.3.5 $NOTE Command

The $NOTE (message) command permits the user to transmit a message to the Simulator which in turn transmits it to the output file for documentation purposes; i.e., the message will be printed on the reports generated by the RPG.

The format of this command is shown below. There are no operands. The message is supplied on a second statement.

```
1...5...10...15...20...25...30...35    Format...
$NOTE                                  A8
message...........                     20A4
```

message... specifies any string of up to 80 characters.

An example of how this statement might be used is included in the command sequence presented in the example in Section 4.3.4.

4.3.6 $TERM Command

The $TERM (terminate) command permits the user to terminate the simulation. Often, this command is used in conjunction with the $NOTE command so that the reason for termination can be specified in the output produced by the Simulator. Functionally, this command causes the same action to take place as the "TERM" operand used with the $BREAK command.

The format of this command is shown below. There are no operands nor any additional statements required.

```
1...5...10...15...20...25...30...35   Format...
$TERM                                 A8
```

Note that the following two command sequences are equivalent.

```
1...5...10...15...20...25...30...35
$TAPIN
$BREAK
     00004100
END
$EXEC    00004000
$TERM
```

```
1...5...10...15...20...25...30...35
$TAPIN
$BREAK
     00004100 TERM
END
$EXEC    00004000
```

In the first sequence, since no action is specified with the break point, the FCP will read the commands following the $EXEC statement. The $TERM statement will cause the simulation to be terminated. In the second sequence the simulation will be terminated as soon as the break point is reached because of the "TERM" action specified.

One might wonder why the first sequence would be used, since the second one accomplishes the same functions with fewer statements. To better understand this, consider the following command sequence:

```
1...5...10...15...20...25...30...35
$TAPIN
$BREAK
     00004100
END
$EXEC    00004000
$NOTE
TEST PROGRAM REACHED LOC 4100
$TERM
```

Observe that in this case, it is possible to print a message before terminating the simulation. This would not be possible when the "TERM" action of a break point is used. Another example of the use of this command is shown in Section 4.3.11.

4.3.7 $DUMP Command

The $DUMP (list memory contents) command allows the user to dump one or more regions of simulated memory.

The format of this command is presented below. There are no operands on the $DUMP statement, but one or more region specifications and a delimiter are specified on successive statements.

```
1...5...10...15...20...25...30...35   Format...
$DUMP                                 A8
     start... stop...                 5X,A8,1X,A8
END                                   A4
```

start... specifies the absolute SKC3120 starting address of the region to be dumped

stop.... specifies the absolute SKC3120 ending address of the region to be dumped.

Note. A location must be specified as an eight digit hexadecimal number,right justified and padded on the left with zeros

Any number of region specification statements can be present. The starting and ending locations of each dump region may include SKC3120 memory of different types; e.g., LSI scratchpad and core memory. When such "memory boundaries" are crossed, the Simulator service routine SDUMP takes the appropriate action. Note that the actual dump format will differ with different types of memory.

An example of a $DUMP command follows:

```
1...5...10...15...20...25...30...35
$DUMP
     00000200 0000020E
     000005FE 00000650
     00000FFE 00001100
END
```

The $DUMP command is acted upon as soon as it is encountered by the FCP. Thus a memory dump of the load module may be obtained by dumping the region of memory it occupies immediately following the $TAPIN command. This is one of the easiest ways of obtaining an SKC3120 memory dump. Of course, the $DUMP command can be processed after a specified break point has been reached to determine the contents of memory after a portion of a program has been executed. The command sequence shown below might be used to accomplish this:

```
1...5...10...15...20...25...30...35
$TAPIN
$DUMP
     00000200 000005FF
     00001000 000011C0
END
$BREAK
     000011BC
END
$EXEC    00001000
$NOTE
PROGRAM EXECUTION COMPLETE AT LOC 11BC
$DUMP
     00000200 000005FF
     00001000 000011C0
$TERM
```

Both $DUMP commands in the above sequence dump the same regions of memory, the first before the program is executed and the second after it has completed execution. A comparision of these two dumps would enable the user to determine which locations had been changed, either intentionally or inadvertently, by the program.

4.3.8 $IODEF Command

The $IODEF (real time clock definition) command allows the user to define the real time clock, level of interrupt and the memory speed to be used in execution time computations, and the real time clock period.

The format of this command is shown below. There are no operands, but a second statement is required for specifying the necessary data.

```
1...5...10...15...20...25...30...35   Format...
$IODEF                                A8
ci mems irtclk                        I2,1X,I4,1X,I10
```

ci — specifies the real time clock interrupt number (1) where ci is interpreted as 2**ci i.e. if ci=0 then interrupt number is 2**0 or 1.

mems — specifies the memory speed. The scaling of the least significant digit is 10**-3 MHZ. If this value is zero, no time computations are performed.

irtclk — specifies the real time clock period in nano-seconds.

If this statement is used, it must appear once before the first $EXEC statement. If this statement does not appear anywhere in the command sequence, before the first $EXEC card, then default values will be used.

The default values assigned are:

mems	1228
irtclk	0

Note that on the SKC3120 Computer, the real time interrupt number is usually 1, and thus ci=0 should be specified when the $IODEF command is used. It is recommended that the user specify a memory speed of zero to inhibit time computations unless simulated execution time is required; this allows the Simulator to operate more efficiently.

Use of this command is illustrated in the following examples:

```
1...5...10...15...20...25...30...35
$IODEF
 1 1750   00005000
```

```
1...5...10...15...20...25...30...35
$IODEF
 1    0   00006125
```

In both examples, the real time interrupt number is set to 1. In the first example, a CPU clock period of 1.75 micro-seconds (1750 nano-seconds) is specified, with a 5 ms real time clock period. In the second example, the memory speed is specified as 0 so that no execution time computations will be performed, and the real time clock period is set at 6.125 milli-seconds. Since no time computations are performed, the occurrence of the real time interrupt is inhibited.

4.3.9 $INTRPT Command

The $INTRPT (interrupt) command permits the user to set the interrupt flag, INTFLG.

The format of this statement is shown below. There are no operands, but a second statement is required to specify the interrupt flag value.

```
1...5...10...15...20...25...30...35   Format...
$INTRPT                               A8
intf                                  A4
```

intf — specifies the setting of the interrupt flag, INTFLG.
If intf = 1, interrupt 1 is present.
If intf = 2, interrupt 2 is present.
If intf = 3, both interrupts 1 and 2 are present.

An example of how this command might be used to cause interrupt 2 to occur when the program reaches location 4C00 is shown in the example which follows:

```
1...5...10...15...20...25...30...35
$TAPIN
$BREAK
     00004C00
END
$EXEC    00004800
$INTRPT
   2
$EXEC
```

When the break point at 4C00 is reached, the Simulator will read and process the $INTRPT command, which specifies that interrupt 2 is to be set, and will then continue execution where it was suspended.

Notes on the generation of interrupts.

Note 1: Interrupt 1 is usually the interrupt number assigned to the real time clock. Such an interrupt will occur each time that the real time clock period, specified on an $IODEF command, has elapsed, provided that interrupts have not been disabled by the program being simulated. Interrupt 1 is the interrupt which is usually scheduled to occur at regular time intervals. Interrupt 2, which is normally the interrupt generated by the interrupt status register hardware, can be generated by use of the $IODEF command for an application where regular occurrence of interrupt 2 would be meaningful.

Note 2: The user should note the difference in specification of interrupts in this command and the $IODEF command.

$IODEF refers to the interrupt as a power of two.

$INTRPT refers to the interrupt directly as an integer.

Note 3: The interrupts generated by the commands $INTRPT and $IODEF make use of simulated hardware scratch pad locations 3,4,5 and 6 as described in the SKC3120 Principles of Operation.

4.3.10 $CHECK Command

The $CHECK (check point) command permits the user to invoke the checkpoint service. This will cause the current state of the program being simulated to be saved on secondary storage, so that it may later be recovered by using the $RESTR command.

The format of this command is shown below. There are no arguments.

```
1...5...10...15...20...25...30...35   Format...
$CHECK                                A8
```

An example of how this command is used in conjunction with the $RESTR command is shown in Section 4.3.11.

4.3.11 $RESTR Command

The $RESTR (restart) command permits the user to invoke the restart service. This will restart the Simulator from the previous checkpoint.

The format of this command is shown below. There are no arguments.

```
1...5...10...15...20...25...30...35    Format...
$RESTR                                 A8
```

An example of how the checkpoint and restart commands are used is given below:

```
1...5...10...15...20...25...30...35
$TAPIN
$CHECK
$BREAK
     000011F0
END
$EXEC    00001000
$RESTR
$INPUTHX
     000010E4 0000A001
END
$EXEC    00001000
$RESTR
$INPUTHX
     000010E4 0000A002
END
$EXEC    00001000
$TERM
```

The $CHECK command saves the state of the computer before program execution has begun. This is used later by the two $RESTR commands to restore everything to the initial condition. The problem program is executed three times, each time with a different instruction at location 10E4. After the third execution, the simulation is terminated.

4.3.12 $CHKSM Command

The $CHKSM (check sum) command directs the Simulator to read the load module produced by the loader and compute a checksum, i.e., a 2's complement of the sum of the contents of all read-only-memory cells. In addition, the $CHKSM control specifies the location of a ROM cell, which will ultimately contain the computed checksum, and directs the Simulator to generate a new load module file containing the checksum information.

The format of this command is shown below. There is one argument.

```
1...5...10...15...20...25...30...35  Format
$CHKSM   loc.....                     A8,1X,A8
```

loc..... a string of 8 hexadecimal characters specifying the ROM location that will receive the computed checksum

NOTE: The string must be left padded with zeros to fill the 8 character field.

4.3.13 $INPUTHX Command

The $INPUTHX (input hex) command allows the user to patch simulated memory with hex data. The format of this command is shown below. There are no operands, but one or more data specification statements and a delimiter are required.

```
1..f5...10...15...20...25...30...35   Format...
$INPUTHX                              A8
     loc..... data....                5X,A8,1X,A8
END                                   A4
```

loc..... specifies the absolute SKC3120 address into which the data is to be placed.

data.... specifies the hex data which is to be entered into the location specified.

Both loc and data are eight character hexadecimal number strings right justified and padded on the left with zeros. An example is shown below to illustrate the use of this command:

```
1...5...10...15...20...25...30...35
$INPUTHX
     00000200 0000FFFF
     000012CC 0000A001
     000012E0 00008FFF
END
```

The above command would cause the following actions to take place:

FFFF would be inserted into location 200 (constant area), this might be a constant or perhaps a flag or a mask.

Similarily, A001 is placed in location 12CC and 8FFF is place in location 12E0.

The $INPUTHX command may appear anywhere in the command sequence. The Simulator acts on this command as soon as it is encountered. Often, this command can be used for initializing data before the program execution has begun, or for changing data or instructions for a subsequent execution during the same simulation run (see the example in Section 4.3.11).

The function of this command can be likened to patching a program in actual SKC3120 memory via the CCU. Sometimes, a re-assembly can be avoided by simply patching those locations which have to be modified.

4.3.14 $INPUTFX Command

The $INPUTFX (input fixed point) command allows the user to patch simulated memory with conversion from host machine floating point to target machine (SKC3120) fixed point with scaling. Ordinarily, this command would be used for patching data; the $INPUTHX command would be used for patching instructions.

The format of this command is shown below. There are no operands, but one or more data specification statements and a delimiter are required.

```
1...5...10...15...20...25...30...35   Format...
$INPUTFX                              A8
     loc..... data.......... scale.   5X,A8,1X,E14.7,1X,E14.7
END                                   A4
```

loc..... specifies the absolute SKC3120 address into which the data is to be placed. This address is specified as an eight character hexadecimal number and must be padded on the left with zeros.

data..... specifies the value of the data which is to be put into the location specified. The value is expressed as a decimal floating point number.

scale..... specifies the scaling of the least significant bit (LSB) of the resulting fixed point quantity. It is expressed as a decimal floating point number.

A few examples are shown below to illustrate the use of this command:

```
1...5...10...15...20...25...30...35
$INPUTFX
     000000E4     512.            1.
     000000E5     512.            4.
     000000E6   12345.67          17.5
END
```

Note that the data and scale values may be supplied in either "E" or "F" format; the Simulator will automatically convert values given in "F" format to "E" format. In fact, the above command would appear in the command output listing as:

```
1...5...10...15...20...25...30...35
$INPUTFX
     000000E4  0.5120000E+03  0.1000000E+01
     000000E5  0.5120000E+03  0.4000000E+01
     000000E6  0.1234567E+05  0.1750000E+02
END
```

Note that in the last set of values, some conversion error will have occurred.

The above command would cause the following actions to take place; after processing the command, the contents of the locations specified would be:

Location	Contents
000000E4	00200
000000E5	00080
000000E6	002C1

The $INPUTFX command may appear anywhere in the command sequence. The Simulator acts on this command as soon as it is encountered. Often, this command may be used for initializing data before beginning program execution, or for modifying this data for a subsequent execution during the same simulation run.

The function of this command, as the $INPUTHX command, can be likened to patching a program in actual SKC3120 memory via the CCU. Of course, this command also performs a conversion function, which cannot be done on the CCU.

4.3.15 $OUTPTHX Command

The $OUTPTHX (output hex) command allows the user to output a value from simulated memory directly in hex format.

The format of this command is shown below. There are no operands, but one or more address specification statements and a delimiter are required.

```
1...5...10...15...20...25...30...35   Format...
$OUTPTHX                              A8
     loc.....                         5X,A8
END                                   A4
```

loc..... specifies the absolute SKC3120 address from which the data is to be written. The address is specified as an eight character hexadecimal number and must be padded on the left with zeros.

An example will illustrate how this command may be used. Note that the action is similar to that caused by the $DUMP statement.

```
1...5...10...15...20...25...30...35
$OUTPTHX
     000000E0
     000041C8
END
```

This command will cause the contents of locations "E0" and "41C8" to be printed in hex. It may be likened to examining the contents of a memory word via the CCU.

4.3.16 $OUTPTFX Command

The $OUTPTFX (output fixed point) command allows the user to output a value from simulated memory and convert it to host machine floating point format.

The format of this command is shown below. No operands are required, but one or more address specification statements and a delimiter are needed.

```
1...5...10...15...20...25...30...35    Format...
$OUTPTFX                                A8
     loc..... scale.........            5X,A8,1X,E14.7
END                                     A4
```

loc..... specifies the absolute SKC3120 address from which the data is to be written. The address is specified as an eight character hexadecimal number and must be padded on the left with zeros.

scale..... specifies the scaling of the least significant bit (LSB) of the quantity in SKC3120 simulated memory. It is expressed as a decimal floating point number.

A few examples are shown below to illustrate how this command might be used:

```
1...5...10...15...20...25...30...35
$OUTPTFX
     000000E4      1.
     000000E5      4.
     000000E6     17.5
END
```

As with the $INPUTFX command, the scale value may be specified in either "E" or "F" format; the Simulator will automatically convert values specified in "F" format to "E" format. The above command would appear in the output command listing as:

```
1...5...10...15...20...25...30...35
$OUTPTFX
     000000E4  0.1000000E+01
     000000E5  0.4000000E+01
     000000E6  0.1750000E+02
END
```

Note that some minor conversion error may occur, depending on the value input.

The table below shows the hex contents of the locations specified and the corresponding real values which are output by the Simulator:

Location	Hex value	Real value	Scale
000000E4	00200	0.5120000E+03	0.1000000E+01
000000E5	00080	0.5120000E+03	0.4000000E+01
000000E6	002C1	0.1234567E+05	0.1750000E+02

The $OUTPTFX command may appear anywhere in the command sequence. The Simulator acts on this command as soon as it is encountered.

4.3.17 $INPUTFL Command

The $INPUTFL (input floating point) command allows the user to patch simulated memory with conversion from host machine floating point to target machine (SKC3120) floating point.

The format of this command is shown below. There are no operands, but one or more data specification statements and a delimiter are required.

```
1...5...10...15...20...25...30...35..    Format...
$INPUTFL                                 A8
     loc.....  ..................data    5X,A8,1X,D22.0
END                                      A4
```

loc..... specifies the absolute SKC3120 address of the first word of the double word, into which the datum is to be placed. This address is specified as an eight character hexadecimal number and must be padded on the left with zeros.

data..... specifies the double precision value of the datum. The value is expressed as a decimal floating point number.

A few examples are shown below to illustrate the use of this command:

```
1...5...10...15...20...25...30...35..
$INPUTFL
     000000E4                 51.2E+01
     000000E6                     512.
     000000E8               1234567D-2

END
```

Note that the data values may be supplied in "D", "E", or "F" format; the simulator will automatically convert values given in "D" or "F" format to "E" format. However, the decimal number must be be right-justified in the 22 character field. The above command would appear in the command output listing as:

```
1...5...10...15...20...25...30...35..
$INPUTFL
     000000E4  0.512000000000000D 03
     000000E6  0.512000000000000D 03
     000000E8  0.123456700000000D 05

END
```

The above command would cause the following actions to take place; after processing the command, the contents of the locations specified would be:

ADDR	HEX	HEX	REAL
000000E4	6686	6666	0.5120000E+02
000000E6	008A	4000	0.5120000E+03
000000E8	1C95	4B5A	0.1234567E+07

The $INPUTFL command may appear anywhere in the command sequence. The simulator acts on this command as soon as it is encountered. Often, this command may be used for initializing data before beginning program execution, or for modifying this data for a subsequent execution during the same simulation run.

4.3.18 $OUTPTFL Command

The $OUTPTFL (output floating point) command allows the user to output an SKC3120 double precision value from simulated memory and convert it to host format.

The format of this command is shown below. No operands are required, but one or more address specification statements and a delimiter are needed.

```
1...5...10...15.f.20...25...30...35   Format...
$OUTPTFL                              A8
     loc.....                         5X,A8
END                                   A4
```

loc..... specifies the absolute SKC3120 address of the first word of the double precision floating point datum. This address is specified as an eight character hexadecimal number and must be padded on the left with zeros.

A few examples are shown below to illustrate how this command might be used:

```
1...5...10...15...20...25...30...35
$OUTPTFL
     000000E4
     000000E6
     000000E8
END
```

The table below shows the hex contents of the locations specified and the corresponding floating point values which are output by the simulator:

ADDR	HEX	HEX	REAL
000000E4	6686	6666	0.5120000E+02
000000E6	008A	4000	0.5120000E+03
000000E8	1C95	4B5A	0.1234567E+07

The $OUTPTFL command may appear anywhere in the command sequence. The simulator acts on this command as soon as it is encountered.

4.3.19 $SETAHX Command

The $SETAHX (set A register with hex input) command allows the user to set the A register to an input value.

The format of this command is shown below. There are no operands, but one data specification statement is required.

```
1...5...10...15..f20...25...30...35   Format...
$SETAHX                               A8
     data....                         5X,A8
```

data.... specifies the value to insert in the A register. The data is specified as an eight character hexadecimal number and must be padded on the left with zeros.

An example will illustrate how this command may be used. Note The A register will be set to 41C8.

```
1...5...10...15...20...25...30...35
$SETAHX
     000041C8
```

4.3.20 $SETAFX Command

The $SETAFX (set A register fixed point input) command allows the user to set the A register to an input value.

The format of this command is shown below. There are no operands, but one data specification statement is required.

```
1...5...10...15...20...25...30...35   Format...
$SETAFX                               A8
     data.... scale.........          5X,E14.7,1X,E14.7
END                                   A4
```

data..... specifies the value of the data which is to be put into the A register. The value is expressed as a decimal floating point number.

scale..... specifies the scaling of the least significant bit (LSB) of the resulting fixed point quantity. It is expressed as a decimal floating point number.

A few examples are shown below to illustrate how this command might be used:

```
1...5...10...15...20...25...30...35
$SETAFX
     45.          .0054932
```

As with the $INPUTFX command, the scale value may be specified in either "E" or "F" format; the simulator will automatically convert values specified in "F" format to "E" format. The above command would appear in the output command listing as:

```
1...5...10...15...20...25...30...35
$SETA
     HEX            REAL                SCALE
    00002000   0.4500000E+02       0.5493201E+020E+01
```

The $SETAFX command may appear anywhere in the command sequence. The simulator acts on this command as soon as it is encountered.

4.3.21 $SETAFL Command

The $SETAFL (set AB register to floating point value) command allows the user to set the AB registers to an input value.

The format of this command is shown below. There are no operands, but one data specification statement is required.

```
1...5...10...15...20...25...30...35   Format...
$SETAFL                               A8
    data....                          5X,D22.0
```

data.... specifies the double precision floating point data. Note that the data values may be supplied in "D", "E", or "F" format; the simulator will automatically convert values given in "D" or "F" format to "E" format. However, the decimal number must be right-justified in the 22 character field.

A few examples are shown below to illustrate how this command might be used:

```
1...5...10...15...20...25...30...35
$SETAFL
                     32767
$SETAFL
                  32767D03
```

The table below shows the hex contents of the register and the corresponding floating point values which are input to the simulator:

```
$SETA
      HEX (A)     Hex (B)          Real
    00007FFF    0000008F      0.3276700E+05
$SETA
      HEX (A)     Hex (B)          Real
    00007CFF    00000699      0.3276699E+08
```

The $SETAFL command may appear anywhere in the command sequence. The simulator acts on this command as soon as it is encountered.

4.3.22 $SETBHX Command

The $SETBHX (set B register from hex input) command is identical to the $SETAHX command except that the B register receives the result of the conversion. Refer to Section 4.3.19, $SETAHX command.

4.3.23 $SETBFX Command

The $SETBFX (set B register from a fixed point input) command is identical to the $SETAFX command except that the B register receives the result of the conversion. Refer to Section 4.3.20, $SETAFX command.

4.3.24 $SETX Command

The $SETX (set XR register from a hex input) command is identical to the $SETAHX command except that the XR register receives the result of the conversion. Refer to Section 4.3.19, $SETAHX command.

4.3.25 $SETR Command

The $SETR (set base or inactive index register from hex input) command allows the user to select a register and to set the specified CPU register (i.e. Base register 1, 2, or the inactive index register). The value is input in Hex.

The format of this command is shown below. There are no operands, but one or more data specification statements and a delimiter are required.

```
1...5...10...15...20...25...30...35   Format...
$SETR                                 A8
     rg data....                      5X,I2,1X,A8
```

rg — specifies the register into which the data is to be placed. Register specification is shown below.

Reg (rg)	Interpretation
3	Base register one selection
4	Base register two selection
5	Inactive index register selection

data.... — specifies the hex data which is to be entered into the location specified. The data is specified as an eight character hexadecimal number and must be padded on the left with zeros.

An example is shown below to illustrate the use of this command:

```
1...5...10...15...20...25...30...35
$SETR
     03 0000789A
```

The Simulator will output the following.

```
$SETR
         REG                    HEX
         B1                     0000789A
```

The $SETR command may appear anywhere in the command sequence. The simulator acts on this command as soon as it is encountered.

5. FCP ENVIRONMENT SPECIFICATIONS

The Simulator provides to the FORTRAN Control Program (FCP) an interface, which is termed Labeled Common Control Blocks. Essentially, the control blocks are employed in the transmission of the state of the machine to the FCP, which may thereby dynamically control the simulation process and request various Simulator services. The control and service actions performed by the FCP are operative in response to any normal or abnormal condition as diagnosed by the Simulator.

The following control blocks are discussed in the sections which follow:

Symbol	Description	Section
CPU	CPU state	5.1
IO	Input/output model linkage	5.2
MEMORY	Simulated memory definition	5.3
IODEF	Memory speed, real-time clock interrupt number and period	5.4
BLIST	Break point list	5.5

5.1 CPU CONTROL BLOCK

The CPU control block provides to the user the state of the CPU and permits the user to examine and change its state, and to initiate and complete various control functions. The information content of the block is presented in Table 5-I.

Examination and/or modification of the CPU registers may be performed directly by the FCP or indirectly by service requests. The services - SSETA, SSETB, SSETX, SMEMRY, that may be invoked for A, B, PC, OAR or XR register access are described in Section 6.5, FCP Service Requests.

The error indicator, ERRFLG, is set by the Simulator to indicate that the Simulator has diagnosed a user error. The user error may have resulted from faulty program logic or from invalid user control and service request actions. Table 5-II presents the complete set of diagnostics provided by the Simulator. This table is divided into two parts, the first for program logic or arithmetic errors and the second for service request errors.

The interrupt flag, INTFLG, may be employed by the user to initiate interrupts. The Simulator, based upon the state of the interrupt hardware, will queue, initiate if possible, and unqueue the interrupts. The state of the interrupt hardware is determined by the SR, IMR, and active and pending interrupt lists.

The SR, IMR and Interrupt trap and save locations are accessible to the user. The INTFLG values are presented in Table 5-I.

The trace flag, TRCFLG, may be employed by the user to turn tracing on or off as a function of any abnormal or normal condition. The TRCFLG settings are also included in Table 5-I.

The termination flag, TRMFLG, may be employed by the user to terminate the simulation as a function of any condition. Again, its values are presented in Table 5-I.

TABLE 5-I CPU CONTROL BLOCK

Word	Symbol	Description	Init Value
1	A	A register	0
2	B	B register	0
3	PC	Pointer to next instruction	4096
4	PCOLD	Pointer to last instruction	4096
5	C	C register	0
6	D	D register	0
7	CBO	Carry bit	0
8	OAR	Pointer to datum	0
9	IR	Instruction Register	0
10	SR	Status Register	0
11	IMR	Interrupt Mask Register	0
12	XR	Index register	0
13	PSU(3)	3 spare	0
16	B1	base register one	0
17	B2	base register two	0
18	IXR	inactive index register	0
19	PSU2(57)	57 spare	0
76	SWR	Datum	0
77	TIME(2)	Elapsed time (sec and nsec)	0
79	ERRFLG	Error indicator	0
		(see Table 5-II)	
80	INTFLG	Interrupt flag	0
		0 = no interrupts	
		1 = level 1 interrupt	
		2 = level 2 interrupt	
		3 = level 1 & 2 interrupts	
81	TRCFLG	Trace flag	0
		0 = trace off	
		1 = trace on	
82	TRMFLG	Termination flag	0
		0 = no termination	
		1 = termination	

TABLE 5-II SIMULATOR DIAGNOSTICS

ERRFLG	Interpretation	Service	Action
0	No error		Normal
4	Fixed point overflow		A = result
8	Floating point overflow		A = result
12	Floating point underflow		A = result
20	Fixed point divide check		NOP
24	Floating point normalization		A, B = 0
28	Storage protect		No store
32	Addressability (PC)		NOP
	Addressability (OAR)		NOP
44	Illegal instruction		NOP
48	Invalid device code (I/O)		No I/O oper
52	Device not connected (I/O)		No I/O oper
200	Invalid array size	SDUMP	FCP return
		SINPUT	FCP return
		SOUTPT	FCP return
		STABLE	FCP return
204	Undefined symbol	SDUMP	Next region
		SINPUT	FCP return
		SOUTPT	FCP return
		STABLE	Next symbol
208	Invalid address	SDUMP	Next region
		SINPUT	FCP return
		SOUTPT	FCP return
		STABLE	Next addr.
212	Invalid conversion (FIX)	SINPUT	No convert
	(-2**16 LT x LT 2**16-1)	SSETA	No convert
		SSETB	No convert
	Invalid conversion (FLT)	SINPUT	No convert
	.1469367D-38 LE R(host)	SSETA	No convert
	LE .3402823D39		
216	Invalid CPU register	SSETR	FCP return
	(B1, B2, IXR) selection		

5.2 IO CONTROL BLOCK

The Simulator permits the user to introduce models of external devices for the purpose of dynamic simulation. The Simulator provides the necessary linkage between itself and the models, and transmits the information content of the I/O instruction via the IO control block. The information content of the IO control block is presented in Table 5-III, and represents the state of the I/O section of the Simulator.

The type of instruction that is being executed is determined by the INOUT flag setting. DMA operations may be accomplished by the services SINPUT and SOUTPT, which are described in Sections 6.5.3 and 6.5.4, respectively.

TABLE 5-III IO CONTROL BLOCK

Word	Symbol	Description	Init Value
1	LFLAG	Not used	0
2	COMMND	Not used	0
3	ACK	Acknowledge bit	0
4	TMEDLY	Time delay	0
5	INOUT	Input or output operation 1 = input 0 = output	0
6	LOC	Not used	0
7	DATA	Not used	0
8	DC	Device code	0

5.3 MEMORY CONTROL BLOCK

The MEMORY control block may be employed by the user to override the default memory size of 16K words. The size of the memory model is specified by the dimension of the SIMMEM array. The width of the memory is one full-word of the host machine, which is sufficiently large to accommodate the prescribed word lengths of the simulated SKC3120 Computer. Table 5-IV presents the information content of the MEMORY control block.

TABLE 5-IV MEMORY CONTROL BLOCK

Word	Symbol	Description	Initial Value
N*	SIMMEM(N)	Simulated memory	HLT's **

* N ranges from 1 to 16384

** The initialization pattern is embedded in the Simulator Configuration File and is normally set to the 'HLT' instruction code.

See Host Procedures Manual for selection of the appropriate Simulator Configuration File. The format of the Simulator Configuration File is described in the Assembler Linkage/Editor Simulator Users Manual.

5.4 IODEF CONTROL BLOCK

The IODEF control block may be employed by the user to initialize the real time clock period and to specify the CPU clock period for instruction execution time computations. IRTCLK may be set to any integer value, where the least significant digit has a value of one nano-second. This serves as the basis by which the Simulator will initiate real-time interrupts. MEMSPD may be set to specify the CPU clock period, where the least significant digit scale factor is 10**-3 MHZ. The default value of 0, if not overridden, indicates that a no-timing option is selected. The information content of this control block is presented in Table 5-V. The IODEF control block must be initialized before the first instruction is executed or the default values will be used throughout the run.

TABLE 5-V IODEF CONTROL BLOCK

Word	Symbol	Description	Initial Value
1	IRTCLK	Real-time clock period	0
2	MEMSPD	CPU clock period	1228

5.5 BLIST CONTROL BLOCK

The BLIST control block consists of the array BRKLST with a default length of 25 words. This control block may be employed by the user to define a set of simulation break points, SKC3120 absolute addresses at which control will be returned to the FCP. The information content of the BLIST control block is shown in Table 5-VI.

This list may not exceed the 25 words allocated if the default FCP is used. The default FCP maintains a count of the break points in use and passes the count to the Simulator as Argument 1 of subroutine 'CONTRL' (see section 6.1).

If the user generated FCP is used, Argument 1 of 'CONTRL' must be updated each time the break point list changes. The user may also change the number of locations allocated in the BLIST control block,as indicated in TABLE 5-VI.

The Simulator can be made to respond to symbolic references passed to BLIST control block from a KAL31 Assembler languge driver, assembled with the program being simulated. In some instances,there would be an advantage in using this method over the absolute addressing required by the BLIST control block when it is used directly.

TABLE 5-VI BLIST CONTROL BLOCK

Word	Symbol	Description	Initial Value
1	BRKLST(1)	PC break point	0
:			:
:			:
25	BRKLST(25)		0
:			:
:			:
N	BRKLST(N)		0

6 FCP LANGUAGE SPECIFICATIONS

The FORTRAN Control Program (FCP) must consist of a set of FORTRAN (or FORTRAN compatible) statements. The language and associated rules must coincide with the level of FORTRAN chosen by the user. The Simulator requires that the FCP be written as a subroutine and must include linkage and calling sequences, type declarations and allocation of common blocks as described in the following paragraphs.

6.1 FCP SUBROUTINE LINKAGE AND CALLING SEQUENCES

The name of the FCP must be "CONTRL". The format of the FORTRAN subroutine statement is:

```
1...5...10...15...20...25...30...35...40...45...50...55...60
      SUBROUTINE CONTRL (ARG1,ARG2,ARG3,ARG4)
```

ARG1 is a fixed point quantity which is output from the FCP and specifies the number of user defined active break points. The BRKLST array in the BLIST control block must be at least as large as this value. If ARG 1 = 0, then the user desires to single step through the simulation process; i.e., control will be returned to the FCP after each instruction execution.

ARG2 is a fixed point quantity which is output from the FCP and specifies the elapsed time at which control should be returned to the FCP. The least significant bit (LSB) has a value of one micro-second. If ARG 2 = 0, the function is ignored. i.e. Elapsed time does not cause the Simulator to return control to the FCP.

ARG3 is a fixed point quantity which is output from the FCP and specifies the count of instructions after which control is returned to the FCP. If ARG 3 = 0, the function is ignored. i.e. Instruction count does not cause the Simulator to return control to the FCP.

ARG4 is a fixed point quantity which is input to the FCP and is set by the Simulator to indicate the condition that caused control to be transferred to the FCP.

The calling sequence, with the exception of ARG4, is user defined, as part of the FCP, and specifies the size of the break point list and the instruction count or elapsed time after which control will be returned to the FCP.

All arguments have an initial value of 0 except ARG4, which has an initial value of 16. This allows the user to define or initialize the Simulator control blocks before starting to simulate the problem program , by comparing the value of the arguments and then initializing if equal to zero or processing if not equal to zero. If the FCP is written to read an input file during initialization, external control can be exercised over arguments 1,2 and 3 which are then passed to the Simulator from the FCP control program.

If the Simulator diagnoses a user operational program logic or arithmetic error, control will be returned unconditionally to the FCP. This permits the user to define error handlers for all of the settings of ERRFLG, as presented in Table 5-II.

The condition indicator, ARG4, which is set by the Simulator, indicates the reason for transfer of control from the Simulator to the FCP. The ARG4 settings are presented in Table 6-I.

TABLE 6-I ARG4 SETTINGS

ARG4 Value	Interpretation
2**0 = 1	Error condition (defined by ERRFLG)
2**1 = 2	Real-time interrupt
2**2 = 4	Elapsed time interrupt
2**3 = 8	Instruction count interrupt
2**4 = 16	Break point interrupt

It is possible that several bits of ARG4 are set to indicate multiple conditions; e.g., a value of 5 (2**2 + 2**0) would indicate both an error condition and the presence of an elapsed time interrupt.

6.2 TYPE DECLARATION

All information transmitted to/from the FCP via the control blocks described in Section 3 must be integer full words. Hence, an IMPLICIT statement or explicit INTEGER statements for each variable must be present. The IMPLICIT statement format is:

```
 1...5...10...15...20...25...30...35...40...45...50...55...60
       IMPLICIT INTEGER (A-Z)
```

See the listing of the Default FCP program in Appendix A.

6.3 CONTROL BLOCKS

Areas must be defined for Labeled Common IO,CPU,BLIST,IODEF and MEMORY as shown in the listing of the Default FCP program in Appendix A.

A typical definition statement follows:

```
1...5...10...15...20...25...30...35...40...45...50...55...60
      COMMON/IO/LFLAG,COMMND,ACK,TMEDLY,INOUT,LOC,DATA,DC
```

6.4 CONTROL FUNCTIONS

The control functions that may be dynamically performed by the FCP are numerous and varied. A partial list is presented in Table 6-II. All functions are operative in response to the normal and abnormal conditions specified in Section 5.

TABLE 6-II FCP CONTROL FUNCTIONS

Control Function	Associated Mechanism
Multiple break points	/BLIST/ BRKLST
Single step	CONTRL calling sequence
Multiple starting points	/CPU/ PC
Masking and unmasking	/CPU/ SR,IMR
Interruption	/CPU/ INTFLG
Tracing (on/off)	/CPU/ TRCFLG
Termination	/CPU/ TRMFLG
Error handlers	/CPU/ ERRFLG
Memory definition and size	/MEMORY/ SIMMEM
Initiate DMA operations	/MEMORY/SINPUT,SOUTPUT *
Initiate I/O via A register	/IO/ INOUT,SDVICE *

*SINPUT,SOUTPUT and SDVICE may be used to simulate the transfer of data to and from the KAL31 Assembly language program being simulated.

6.5 FCP SERVICE REQUESTS

The Simulator services that are provided are listed alphabetically in Table 6-III. This table also indicates in which section each service request is described. The linkage conventions that must be observed are presented in succeeding sections. All service requests may be issued in response to any normal or abnormal conditions as diagnosed by the Simulator.

6.5.1 STRACE REQUEST

The STRACE request may be issued by the FCP. The service supplied by the Simulator is to record the state of the CPU, including elapsed time, at the point in the Simulation where the call was made. The format of the FORTRAN statement is shown below; there are no arguments.

```
1...5...10...15...20...25...30...35...40...45...50...55...60
      CALL STRACE
```

Tracing may also be controlled by setting TRCFLG, as described in Section 5.1.

When the trace option is on (TRCFLG = 1), the Simulator will produce output information for each instruction which is executed. The RPG will use this information to produce a trace report. Table 4-II describes the items included in the Trace Report. The trace feature of the Simulator should generally be used only for relatively short programs or for small portions of larger programs, since a large volume of data can be generated by a short amount of simulated time. The trace can be turned on and off at successive break points.

A selective trace can be generated by setting a break point at the start of the trace area and another at the end of the trace area. The user can then turn trace on at the first breakpoint and turn it off at the second, so that each time the code to be traced is entered, the tracing will be activated.

TABLE 6-III SIMULATOR SERVICES

Service	Function	Section
SCHECK	Records state of SKC3120 machine	6.5.11
SDUMP	Transmits data from specified memory regions	6.5.2
SDVICE	Defines the data to appear on the I/O interface when the appropriate input instruction is executed	6.5.5
SINPUT	Transmits data to selected memory locations under format control	6.5.3
SIO	Logically connects an I/O device code to the name of a subroutine to be invoked when data is transmitted on that channel	6.5.6
SMEMRY	Determines a symbol's value	6.5.7
SNOTE	Transmits a user defined message	6.5.14
SOUTPT	Transmits data from selected memory locations under format control	6.5.4
SRESTR	Restarts simulation from last checkpoint	6.5.12
SSETA	Sets the A register under format control	6.5.8
SSETB	Sets the B register under format control	6.5.9
SSETX	Sets the index register	6.5.10
SSETR	Sets a selected CPU register (B1,B2,IXR)	6.5.16
STABLE	Determines value of an array of symbols	6.5.15
STAPIN	Performs a computer load	6.5.13
STRACE	Records state of the CPU	6.5.1

6.5.2 SDUMP REQUEST

When the SDUMP request is issued by the user, the service provided by the Simulator is to record the contents of the regions of memory specified by the user. The starting and stopping locations (dump regions) may be specified either symbolically or absolutely. The Simulator diagnostic action and setting of the ERRFLG indicator are specified in Table 5-II. The format of the FORTRAN statement is shown below:

```
1...5...10...15...20...25...30...35...40...45...50...55...60
      CALL SDUMP (ARG1,ARG2,ARG3,ARG4)
```

ARG1 is a fixed point quantity and specifies the ARG3 and ARG4 array sizes.

ARG2 is a double-word character string (8 characters) and specifies the deckname if the symbolic referencing option is selected. If ARG2 = "NODECK ", absolute addressing is selected.

ARG3 is a fixed point or character string array which specifies the dump starting locations.

ARG4 is a fixed point or character string array which specifies the dump ending locations.

If absolute addressing is selected, then ARG3 and ARG4 must be single subscripted arrays of dimension (ARG1) and must contain fixed point absolute SKC3120 addresses. If symbolic referencing is selected, then ARG3 and ARG4 must be double subscripted arrays of dimension (4,ARG1), and all symbols must be left-justified and padded with blanks on the right.

The starting and ending locations of each dump region may include SKC3120 memory of different types; e.g., LSI scratchpad and core memory. When such "memory boundaries" are crossed, the Simulator service routine SDUMP takes the appropriate action. Note that the actual dump format will differ with different types of memory.

6.5.3 SINPUT REQUEST

The SINPUT request may be issued by the FCP and the service supplied by the Simulator is to input (under format control) into memory and record the results of the service operation. The input starting location may be defined symbolically or absolutely. The type of conversions may be none or host machine floating to SKC3120 fixed point. The Simulator diagnostic action and setting of the ERRFLG indicator are specified in Table 5-II. The format of the FORTRAN statement is:

```
1...5...10...15...20...25...30...35...40...45..f50...55...60
      CALL SINPUT (ARG1,ARG2,ARG3,ARG4,ARG5,ARG6)
```

ARG1 is a full-word character string and specifies the type of conversion desired. The following values are permitted:

- "HEX " No conversion; ARG5 must be an array of fixed point quantities.
- "FIX " ARG5 and ARG6 must be arrays of host machine floating point quantities; the ARG6 array represents the scale factors of the LSB's of the resulting fixed point quantities.
- "FLT " ARG5 must be a double precision floating point quantity. When "FLT " option is specified, the conversion is from host machine floating point to SKC3120 floating point.

ARG2 is a double-word character string (8 characters) and gives the deckname if symbolic addressing is being used. If ARG2 = "NODECK ", absolute addressing is used.

ARG3 is a fixed point word or character string and specifies the input memory starting location.

ARG4 is a fixed point quantity and specifies the ARG5 and ARG6 array sizes.

ARG5 is a floating or fixed point array of values to input.

ARG6 is a floating point array of scale factors and is required only for conversion from host machine floating point to SKC3120 fixed point.

If absolute addressing is selected, then ARG3 must be a fixed point scalar and contain an absolute SKC3120 address. If symbolic referencing is selected, then ARG3 must be a single subscripted array of dimension (4). All symbols must be left-justified and padded with blanks on the right.

6.5.4 SOUTPT REQUEST

The SOUTPT request may be issued by the FCP and the service supplied by the Simulator is to output (under format control) from memory and record the results of the service operation. The output starting location may be defined symbolically or absolutely. The type of conversion may be none or SKC3120 fixed point to host machine floating point. The Simulator diagnostic action and setting of the ERRFLG indicator are specified in Table 5-II. The format of the FORTRAN statement is:

```
1...5...10...15...20...25...30...35...40...45...50...55...60
      CALL SOUTPT (ARG1,ARG2,ARG3,ARG4,ARG5,ARG6)
```

ARG1 is a full-word character string and specifies the type of conversion desired. The following values are permitted:

- "HEX " No conversion; ARG5 must be an array of fixed point quantities.
- "FIX " ARG5 and ARG6 must be arrays of host machine floating point quantities; the ARG6 array represents the scale factors of the LSB's of the fixed point quantities to be output.
- "FLT " ARG5 must be a double precision floating point quantity. When "FLT " option is specified, the conversion is from host machine floating point to SKC3120 floating point.

ARG2 is a double-word character string (8 characters) and specifies the deckname if symbolic addressing is to be used; if ARG2 = "NODECK ", absolute addressing is to be used.

ARG3 is a fixed point or character string array specifying the input memory starting location.

ARG4 is a fixed point quantity and specifies the ARG5 and ARG6 array sizes.

ARG5 is a fixed or floating point array which will receive the quantities from memory.

ARG6 is a floating point array of scale factors and is required only for conversion from SKC3120 fixed point to host machine floating point.

If absolute addressing is selected, then ARG3 must be a fixed point scalar and contain an absolute SKC3120 address. If symbolic referencing is selected, then ARG3 must be a single subscripted array of dimension (4). All symbols must be left-justified and padded with blanks on the right.

6.5.5 SDVICE REQUEST

The SDVICE request may be issued by the FCP and the service supplied by the Simulator is to transfer one data word to the A register. This service permits the user to execute closed-loop simulations without providing sophisticated models of external devices. The Simulator diagnostic action and setting of the ERRFLG indicator are specified in Table 5-II. The format of the FORTRAN statement is:

```
1...5...10...15...20...25...30...35...40...45...50...55...60
      CALL SDVICE (ARG1,ARG2,ARG3)
```

ARG1 is a fixed point number and specifies any legal device code number.

ARG2 is a fixed point number and specifies any legal device code number.

ARG3 is a fixed point quantity and represents the datum to be transmitted to the A register.

ARG1 and ARG2 specify a range of device codes such that if during simulation of an input/output instruction, a device code is generated which is within the range, then the ARG3 datum is transmitted to the A register.

6.5.6 SIO REQUEST

The SIO request may be issued by the FCP and the service supplied by the Simulator is to logically connect an I/O device code to the name of a FORTRAN subroutine to be invoked when data is to be transmitted or received on that channel. This service permits the user to define and connect models of external devices and thereby permits inputting and/or outputting to/from the A register (under format control) or memory (under format control). Refer to Section 5 for a description of the IO control block and to Section 7 for the I/O model specification. The Simulator diagnostic action and the setting of the ERRFLG indicator are specified in Table 5-II. The format of the FORTRAN statement is:

```
1...5...10...15...20...25...30...35...40...45...50...55...60
      CALL SIO (ARG1,ARG2,ARG3)
```

ARG1 is a fixed point quantity and specifies the starting number of a range of device codes.

ARG2 is a fixed point quantity and specifies the ending number of a range of device codes.

ARG3 is a fixed point quantity and specifies the number of the FORTRAN subroutine to be invoked when a device code in the range given by ARG1 and ARG2 is specified in an input/output instruction.

The number used in ARG3 is the integer portion of the subroutines MOD0 through MOD80 which are supplied as dummy subroutines and are over-ridden by the user supplied MOD0 through MOD80 as defined by SIO.

6.5.7 SMEMRY REQUEST

The SMEMRY request may be issued by the FCP and the service supplied by the Simulator is to evaluate the input symbol and return the SKC3120 absolute address. The SMEMRY request is a statement function and thus differs from other subroutine services. The SMEMRY service permits the user to perform symbolic arithmetic and logic operations using the PC and the OAR, elements of the CPU control block. The Simulator diagnostic action and setting of the ERRFLG indicator are presented in Table 5-II. The format of the FORTRAN statement is:

```
1...5...10...15...20...25...30...35...40...45...50...55...60
      ASSIGN = SMEMRY (ARG1,ARG2)
```

ARG1 is a double-word character string specifying the deckname.

ARG2 is a character string array and specifies the symbol to be evaluated.

The ARG2 array must be a single subscripted array of dimension (4). Both symbols must be left-justified and padded with blanks on the right.

6.5.8 SSETA REQUEST

The SSETA request may be issued by the FCP and the service supplied by the Simulator is to set the A register as a function of the user specified conversion type and record the results of the operation. The type of conversion permited is none or host machine floating point to SKC3120 fixed point. The Simulator diagnostic action and the setting of the ERRFLG indicator are specified in Table 5-II. The format of the FORTRAN statement is:

```
1...5...10...15...20...25...30...35...40...45...50...55...60
      CALL SSETA (ARG1,ARG2,ARG3)
```

ARG1 is a full-word character string and specifies the type of conversion desired; the following values are permitted:

- "HEX " No conversion.
- "FIX " ARG2 and ARG3 must be host machine floating point scalars, where ARG3 represents the scale factor of the LSB of the resulting fixed point quantity.
- "FLT " ARG2 must be a double precision floating point quantity. When "FLT " option is specified, the conversion is from host machine floating point to SKC3120 floating point.

ARG2 is the host machine floating or fixed point quantity to be input.

ARG3 is a fixed point quantity and represents the LSB value of the conversion from host machine floating point to SKC3120 fixed point.

6.5.9 SSETB REQUEST

The SSETB request is identical to the SSETA request except that the B register receives the result of the conversion. Refer to Section 6.5.8, SSETA request.

6.5.10 SSETX REQUEST

The SSETX request may be issued by the FCP and the service supplied by the Simulator is to set the index register and record the results of the service operation. The Simulator diagnostic action and the setting of the ERRFLG indicator are specified in Table 5-II. The format of the FORTRAN statement is shown below:

```
1...5...10...15...20...25...30...35...40...45...50...55...60
      CALL SSETX (ARG1)
```

ARG1 — is a fixed point scalar quantity which is to be loaded into the index register, XR.

6.5.11 SCHECK REQUEST

The SCHECK request may be issued by the FCP and the service supplied by the Simulator is to record on secondary storage the state of the simulated SKC3120 machine for subsequent restart. The format of the FORTRAN statement is shown below. There are no arguments.

```
1...5...10...15...20...25...30...35...40...45.f.50...55...60
      CALL SCHECK
```

6.5.12 SRESTR REQUEST

The SRESTR request may be issued by the FCP and the service supplied by the Simulator is to re-initialize the SKC3120 machine to the state it had at the time of the previous checkpoint. The Simulator diagnostic action and the setting of the ERRFLG indicator are specified in Table 5-II. The format of the FORTRAN statement is shown below. There are no arguments.

```
1...5...10...15...20...25...30...35...40...45...50...55...60
      CALL SRESTR
```

6.5.13 STAPIN REQUEST

The STAPIN request may be issued by the FCP and the service supplied by the Simulator is to perform a computer memory load; i.e., load the memory with the operational program residing on secondary storage. The Simulator diagnostic action and the setting of the ERRFLG indicated are specified in Table 5-II. The format of the FORTRAN statement is shown below. There are no arguments.

```
1...5...10...15...20...25...30...35...40...45...50...55...60
      CALL STAPIN
```

6.5.14 SNOTE REQUEST

The SNOTE request may be issued by the FCP and the service supplied by the Simulator is to record a user specified message. The format of the FORTRAN statement is:

```
1...5...10...15...20...25...30...35...40...45...50...55...60
      CALL SNOTE (ARG1,ARG2)
```

ARG1 is a fixed point quantity and specifies the number of characters in ARG2.

ARG2 is a character string and specifies the message to be recorded.

6.5.15 STABLE REQUEST

The STABLE request may be issued by the FCP and the service supplied by the Simulator is to evaluate the input array of symbols and return the corresponding SKC3120 absolute addresses. The Simulator diagnostic action and setting of the ERRFLG statement are presented in Table 5-II. The format of the FORTRAN statement is:

```
1...5...10...15...20...25...30...35...40...45...50...55...60
      CALL STABLE (ARG1,ARG2,ARG3,ARG4)
```

ARG1 is a fixed point quantity and specifies the ARG3 and ARG4 array sizes.

ARG2 is a double-word character string (8 characters) and specifies the deckname.

ARG3 is a character string array and specifies the symbols to be evaluated.

ARG4 is a fixed point array of SKC3120 absolute addresses.

ARG3 must be a double subscripted array of dimension (4,ARG1). All symbols must be left-justified and padded with blanks on the right.

6.5.16 SSETR REQUEST

The SSETR request may be issued by the FCP and the service supplied is to set the user specified CPU register (i.e. Base register 1, 2, or the inactive index register) and record the results of the service operation. The simulator diagnostic action and the setting of the ERRFLG indicator are specified in table 5-II.

```
1...5...10...15...20...25...30...35...40...45...50...55...60
      CALL SSETR (ARG1,ARG2)
```

ARG1 is a fixed point scalar quantity which specifies the selected CPU register. permissable ARG1 values are presented in Table 6-IV.

ARG2 is a fixed point scalar quantity which is to be loaded into the selected CPU register.

TABLE 6-IV ARG1 SETTINGS

ARG1 Value	Interpretation
3	Base register one selection
4	Base register two selection
5	Inactive index register selection

This page intentionally left blank

7. I/O MODEL LANGUAGE SPECIFICATION

7.1 MODEL LINKAGE

The model names of the user defined external device models are presented in Table 7-I. The mechanism that must be exercised by the user to inform the Simulator that dynamic simulation is desired is the SIO service request. Issuance of the SIO request results in the Simulator action of logically connecting the specified SIO device codes to the name of the FORTRAN model.

7.2 MODEL CONTROL FUNCTIONS

All control functions that are permitted in the FCP are permissable in the user models. Refer to Section 6.4, FCP Control Functions.

7.3 MODEL SERVICE REQUESTS

All services provided in the FCP are available in the user models. Refer to Section 6.5, FCP Service Requests.

TABLE 7-I MODEL NAMES

Subroutine Model Names	Function
MOD1	I/O model
MOD2	I/O model
:	
:	
MOD63	I/O model
MOD64	I/O model invoked by level 1 interrupt
MOD65	I/O model invoked by level 2 interrupt
MOD66	I/O model
:	
:	
MOD80	I/O model

A maximum of 80 I/O model subroutines are permitted,but Models MOD64 and MOD65 are called unconditionally in the event that a level 1 or level 2 interrupt occurs, respectively. MOD64 and MOD65 are automatically connected and thus need not be specified in an SIO request, however, if any additional processing is required because of the interrupt occurrence, that processing may be included within a MOD64 or MOD65 I/O model supplied by the user.

APPENDIX A

Default FCP Program Listing

APPENDIX A

```
C       DEFAULT FORTRAN CONTROL PROGRAM (FCP)
C
        SUBROUTINE CONTRL(BSIZE,ETIME,COUNT,CONDFG)
C
C       FORTRAM CONTROL PROGRAM - CONTRL
C       ENVIRONMENT AND LANGUAGE SPECIFICATIONS
C
C       BSIZE  - SIZE OF ARRAY BRKLST
C       BRKLST - BREAK POINTS
C       ETIME  - ELASPSED TIME, SCALE LSB=1 US
C       COUNT  - INSTRUCTION COUNT
C       CONDFG - CONDITION FLAG
C       CONDFG = 1  INDICATES ERROR CONDITION
C              = 2  INDICATES REAL TIME CLOCK INTERRUPT
C              = 3  INDICATES ELAPSED TIME INTERRUPT
C              = 4  INDICATES INSTRUCTION COUNT INTERRUPT
C              = 5  INDICATES BREAK POINT INTERRUPT
C
C
C       BLOCK ALLOCATION
C
        DIMENSION SERV(25),ACTION(25),NODCK(2),KARD(20)
        DIMENSION MASK(5),ETIME(2)
        DIMENSION ADDR(25),ISDATA(25),ASCALE(25)
        DIMENSION RSDATA(1),RESULT(1)
        EQUIVALENCE (ISDATA(1),RSDATA(1)), (ISDATA(1),RESULT(1))
C
        COMMON/MEMORY/SIMMEM(16384)
C
C       COMMON BLOCK     NAME=MEMORY
C
C        NAME        DEFINTION
C       SIMMEM - SIMULATED MEMORY (DEFAULT 16K FULL WORDS)
C
C
C
        COMMON/IODEF/IRTCLK,MEMSPD
C
```

```
C       COMMON BLOCK  NAME=IODEF
C
C        NAME        DEFINITION
C       IRTCLK - REAL TIME CLOCK PERIOD, SCALE LSB=1 US  (DEFAULT 0)
C       MEMSPD - SPEED OF MEMORY IN NANO-SECONDS (DEFAULT 1228)
C
        COMMON/CPU/A,B,PC,PCOLD,C,D,CBO,OAR,IR,SR,IMR,XR,PSEU1(3),
       * B1,B2,IXR,PSEU2(57),SWR,TIME(2),ERRFLG,INTFLG,TRCFLG,TRMFLG
C
C       COMMON BLOCK  NAME=CPU
C
C        NAME        DEFINITION
C       A       - A-REGISTER
C       B       - B-REGISTER
C       PC      - UPDATED PROGRAM COUNTER
C       PCOLD   - CURRENT PROGRAM COUNTER
C       C       - C-REGISTER
C       D       - D-REGISTER
C       OAR     - OPERATION ADDRESS REGISTER
C       IR      - INSTRUCTION REGISTER
C       SR      - STATUS REGISTER
C       IMR     - INTERRUPT MASK REGISTER
C       XR      - INDEX REGISTERS (64)
C       SWR     - SWITCH REGISTER
C       TIME    - ELAPSED TIME IN MICRO-SECONDS
C       ERRFLG - ERROR INDICATOR
C       INTFLG - INTERRUPT FLAGS
C       TRCFLG - TRACE FLAG
C       TRMFLG - TERMINATION FLAG
C
        COMMON/IO/LFLAG,COMMND,ACK,TMEDLY,INOUT,LOC,DATA,DC
C
C       COMMON BLOCK  NAME=IO
C
C       NAME         DEFINTION
C       LFLAG   - LONG OR SHORT INSTRUCTION
C       LFLAG   = 0  A REGISTER OPERATION
C               = 1  MEMORY OPERATION
C       COMMND - COMMAND BIT
C       ACK     - ACKNOWLEDGE BIT
C       TMEDLY - TIME DELAY
C       INOUT   - INDICATES INPUT OR OUTPUT OPERATION
C       INOUT   = 0  INPUT OPERATION
C               = 1  OUTPUT OPERATION
C       LOC     - MEMORY LOCATION (HALF WORD ADDRESS
C       DATA    - DATA TRANSMITTED IN OR OUT
C       DC      - DEVICE CODE
C
        COMMON/BLIST/BRKLST(25)
C
```

```
C     COMMON BLOCK  NAME=BLIST
C
C      NAME     DEFINITION
C     BRKLST - LIST OF BREAK POINTS
C
C
C     EXPLICIT TYPE DECLARATIONS
C
      INTEGER A,ACTION,ADDR,ACK,ACTCMD
      INTEGER B, BRKLST, BSIZE, BLNK , B1, B2
      INTEGER C,CBO, CORSTP, CORSTR,COUNT,CONDFG,COMMND
      INTEGER D, DMPSTR, DMPSTP, DCK, DUMPSR,DUMPSP,DATA,DC
      INTEGER HED,HEXDAT
      INTEGER ERRFLG,ETIME,ENDCD
      INTEGER OAR,ONOFF
      INTEGER PCOLD, PC, PCLOC, PSEU1, PSEU2
      INTEGER RAMSTR, RAMSTP, ROMSTR, ROMSTP
      INTEGER SIMMEM, SR, SRMSK, SWR, SYMBOL
      INTEGER START, STOP, SHORTL
      INTEGER TIME, TRCFLG, TRMFLG, TERM, TMEDLY, TMP1(8), TMP2(8)
      INTEGER XR
      DOUBLE PRECISION SERV,REQST,FLT,RDDATA(25)
C
C     DATA DEFINITIONS
C
      DATA SERV/8H$IODEF  ,8H$BREAK  ,8H$INTRPT ,8H$TAPIN  ,
     *          8H$NOTE   ,8H$DUMP   ,8H$INPUTHX,8H$INPUTFX,
     *          8H$INPUTFL,8H$OUTPTHX,8H$OUTPTFX,8H$OUTPTFL,
     *          8H$TRACE  ,8H$CHECK  ,8H$RESTR  ,8H$TERM   ,
     *          8H$EXEC   ,8H$CHKSM  ,8H$SETAHX ,8H$SETAFX ,
     *          8H$SETAFL ,8H$SETBHX ,8H$SETBFX ,8H$SETX   ,
     *          8H$SETR   /
      DATA ENDCD/4HEND /,BLNK/4H    /,TERM/4HTERM/
      DATA MASK/1,2,4,8,16/
      DATA NODCK/4HNODE,4HCK  /
      DATA MDFLAG/0/
      ICNT = 0
  610 ICNT = ICNT + 1
      IF(ICNT.GT.5) RETURN
      IF( IAND( MASK(ICNT), CONDFG ) .EQ. 0 ) GOTO 610
  600 GO TO (1000, 2000, 3120, 4000, 5000),ICNT
 1000 CONTINUE
C
C     ERROR CONDITION PROCESSING
C
      IF(ERRFLG.EQ.32)GOTO 9001
      IF(ERRFLG.EQ.44)GOTO 9002
      IF(ERRFLG.GE.76.AND.ERRFLG.LE.111)GOTO 9003
      GO TO 610
```

```
 9001 CONTINUE
C
C     ADDRESSABILITY ERROR
C
      TRMFLG=1
      CALL SNOTE(33,33HTERMINATION DUE TO ADDRESSABILITY)
      RETURN
 9002 CONTINUE
C
C     ILLEGAL INSTRUCTION
C
      TRMFLG=1
      CALL SNOTE(38,38HTERMINATION DUE TO ILLEGAL INSTRUCTION)
      RETURN
 9003 CONTINUE
C
C     RTA ERROR-CHANNEL ACTIVE BUT NOT CURRENT, OR INACTIVE OR PENDING
C
      TRMFLG=1
      CALL SNOTE(36,36HTERMINATION DUE TO RTA-ADDRESS ERROR)
      RETURN
 2000 CONTINUE
C
C     REAL TIME INTERRUPT CONDITION
C     INTFLG SET TO INITIATE REAL TIME INTERRUPT
      INTFLG=2**INTRTN
C
      GO TO 610
 3120 CONTINUE
C
C     ELAPSED TIME INTERRUPT
C
      GO TO 610
 4000 CONTINUE
C
C     INSTRUCTION COUNT INTERRUPT
C
      GO TO 610
 5000 CONTINUE
C
C     BREAK POINT INTERRUPT
C
      IF(BSIZE.EQ.0)GOTO 1
C
C     COMMAND PROCESSOR
C
```

```
C
C     DETERMINE USER DEFINED ACTION
C
      DO 400 KK=1,BSIZE
      IF(PC.EQ.BRKLST(KK))GOTO 402
  400 CONTINUE
      RETURN
  402 CONTINUE
C
C     TERMINATION ?
C        YES, THEN GOTO TERMINATION SEQUENCE
      IF(ACTION(KK).EQ.TERM)GOTO 7500
C        NO, THEN PARSE ADDITIONAL COMMANDS
C
    1 READ(4,200) REQST,TMP1
      LOC = IHEXCN(8,TMP1(1))
      DO 410 J=1,25
      IF(REQST.EQ.SERV(J))GO TO 420
  410 CONTINUE
C
C     COMMAND NOT RECOGNIZED, WRITE DIAGNOSTIC
C
      WRITE(6,300) REQST,TMP1
      GO TO 1
 420  WRITE(6,210) REQST,TMP1
      GOTO(6000,6100,6200,6300,6400,6500,6600,6700,6800,6900,
     * 7000,7100,7200,7300,7400,7500,7600,7700,7800,7900,8000,
     * 8100,8200,8300,8400), J
C
 6000 CONTINUE
C     $IODEF - DEFINE REALTIME CLOCK INTERRUPT NUMBER
C            - DEFINE MEMORY SPEED
C            - DEFINE REAL TIME CLOCK INTERVAL PERIOD
C
      READ(4,102)INTRTN,MEMSPD,IRTCLK
      WRITE(6,112) INTRTN,MEMSPD,IRTCLK
      GOTO 1
C
 6100 CONTINUE
C     $BREAK - BREAK POINT LIST DEFINITION
C
      READ(4,104) HED,TMP1,ACTCMD
      WRITE(6,114) HED,TMP1,ACTCMD
      PCLOC = IHEXCN(8,TMP1(1))
      IF(HED.EQ.ENDCD) GO TO 1
      CALL PCSTOP(PCLOC,ACTCMD,BSIZE,ACTION(1))
      GOTO 6100
C
```

```
 6200 CONTINUE
C     $INTRRPT - INTERRUPT DEFINITION
C
      READ(4,108) (TMP1(IDX),IDX=1,4)
      WRITE(6,118) (TMP1(IDX),IDX=1,4)
      INTNO=IHEXCN(4,TMP1(1))
      INTFLG= INTNO
      GO TO 1
C
 6300 CONTINUE
C     $TAPIN - TAPE LOAD COMMAND
C
      CALL STAPIN
      GO TO 1
C
 6400 CONTINUE
C     $NOTE -SNOTE COMMAND
C
      READ(4,101) (KARD(I),I=1,20)
      WRITE(6,111) KARD
      CALL SNOTE(80,KARD(1))
      GO TO 1
C
 6500 CONTINUE
C     $DUMP - DUMP REGION DEFINITION
C
      READ(4,100) HED,TMP1,TMP2
      WRITE(6,110) HED,TMP1,TMP2
      START = IHEXCN(8,TMP1(1))
      STOP  = IHEXCN(8,TMP2(1))
      IF(HED.EQ.ENDCD)GOTO 1
      CALL SDUMP(1,NODCK(1),START,STOP)
      GOTO 6500
C
 6600 CONTINUE
C     $INPUT - INPUT DEFINITION, HEX OPTION
C
 6601 NUMBER = 0
 6602 READ(4,100) HED,TMP1,TMP2
      WRITE(6,110) HED,TMP1,TMP2
      IF(HED.EQ.ENDCD) GO TO 6603
      NUMBER = NUMBER + 1
      ADDR(NUMBER)   = IHEXCN(8,TMP1(1))
      ISDATA(NUMBER) = IHEXCN(8,TMP2(1))
      IF( NUMBER .LT. 25 ) GOTO 6602
```

```
 6603 CONTINUE
      IF( NUMBER .EQ. 0 ) GOTO 1
      KOUNT = 1
      IDX = 1
      IF( NUMBER .EQ. 1 ) GOTO 6615
      DO 6610 J = 2, NUMBER
      KOUNT = KOUNT + 1
      IF( ADDR(J-1) + 1 .EQ. ADDR(J) ) GOTO 6610
      CALL SINPUT(4HHEX ,NODCK(1),ADDR(IDX),KOUNT-1,ISDATA(IDX),DMY)
      KOUNT = 1
      IDX = J
 6610 CONTINUE
 6615 CALL SINPUT(4HHEX ,NODCK(1),ADDR(IDX),KOUNT,ISDATA(IDX),DMY)
      IF( HED .NE. ENDCD ) GOTO 6601
      GOTO 1
C
 6700 CONTINUE
C     $INPUT - INPUT DEFINITION, FIX OPTION
C
 6701 NUMBER = 0
 6702 READ(4,106) HED,TMP1,WORD,SCALE
      IF(HED.EQ.ENDCD) GOTO 6703
      NUMBER = NUMBER + 1
      ADDR(NUMBER)  = IHEXCN(8,TMP1(1))
      RSDATA(NUMBER) = WORD
      ASCALE(NUMBER) = SCALE
      WRITE(6,116) HED,TMP1,WORD,SCALE
      IF (NUMBER .LT. 25 ) GOTO 6702
 6703 CONTINUE
      IF( NUMBER .EQ. 0 ) GOTO 1001
      KOUNT = 1
      IDX = 1
      IF( NUMBER .EQ. 1 ) GOTO 6715
      DO 6710 J = 2, NUMBER
      KOUNT = KOUNT + 1
      IF( ADDR(J-1) + 1 .EQ. ADDR(J) ) GOTO 6710
      CALL SINPUT(4HFIX ,NODCK(1),ADDR(IDX),KOUNT-1,
     * RSDATA(IDX),ASCALE(IDX))
      KOUNT = 1
      IDX = J
 6710 CONTINUE
 6715 CALL SINPUT(4HFIX ,NODCK(1),ADDR(IDX),KOUNT,
     * RSDATA(IDX),ASCALE(IDX))
      IF( HED .NE. ENDCD ) GOTO 6701
      GOTO 1001
C
```

```
 6800 CONTINUE
C      $INPUT - INPUT DEFINITION, FLT OPTION
C
 6801 NUMBER = 0
 6802 READ(4,107) HED,TMP1,FLT
      IF(HED.EQ.ENDCD) GOTO 6803
      NUMBER = NUMBER + 1
      ADDR(NUMBER)  = IHEXCN(8,TMP1(1))
      RDDATA(NUMBER) = FLT
      WRITE(6,117) HED,TMP1,FLT
      IF( NUMBER .LT. 25 ) GOTO 6802
 6803 CONTINUE
      IF( NUMBER .EQ. 0 ) GOTO 1001
      KOUNT = 1
      IDX = 1
      IF( NUMBER .EQ. 1 ) GOTO 6815
      DO 6810 J = 2, NUMBER
      KOUNT = KOUNT + 1
      IF( ADDR(J-1) + 2 .EQ. ADDR(J) ) GOTO 6810
      CALL SINPUT(4HFLT ,NODCK(1),ADDR(IDX),KOUNT-1,RDDATA(IDX),DMY)
      KOUNT = 1
      IDX = J
 6810 CONTINUE
 6815 CALL SINPUT(4HFLT ,NODCK(1),ADDR(IDX),KOUNT,RDDATA(IDX),DMY)
      IF( HED .NE. ENDCD ) GOTO 6801
      GOTO 1001
C
 6900 CONTINUE
C      $OUTPT - OUTPUT DEFINITION, HEX OPTION
C
 6901 NUMBER = 0
 6902 READ(4,100) HED,TMP1
      WRITE(6,1101) HED,TMP1
      IF(HED.EQ.ENDCD)GOTO 6903
      NUMBER = NUMBER + 1
      ADDR(NUMBER)  = IHEXCN(8,TMP1(1))
      IF( NUMBER .LT. 25 ) GOTO 6902
 6903 CONTINUE
      IF( NUMBER .EQ. 0 ) GOTO 1
      KOUNT = 1
      IDX = 1
      IF( NUMBER .EQ. 1 ) GOTO 6915
      DO 6910 J = 2, NUMBER
      KOUNT = KOUNT + 1
      IF( ADDR(J-1) + 1 .EQ. ADDR(J) ) GOTO 6910
```

```
      CALL SOUTPT(4HHEX ,NODCK(1),ADDR(IDX),KOUNT-1,ISDATA(IDX),DMY)
      KOUNT = 1
      IDX = J
 6910 CONTINUE
 6915 CALL SOUTPT(4HHEX ,NODCK(1),ADDR(IDX),KOUNT,ISDATA(IDX),DMY)
      IF( HED .NE. ENDCD ) GOTO 6901
      GOTO 1
C
 7000 CONTINUE
C     $OUTPT - OUTPUT DEFINITION, FIX OPTION
C
 7001 NUMBER = 0
 7002 READ(4,106) HED,TMP1,SCALE
      IF(HED.EQ.ENDCD) GOTO 7003
      NUMBER = NUMBER + 1
      ADDR(NUMBER)  = IHEXCN(8,TMP1(1))
      ASCALE(NUMBER) = SCALE
      WRITE(6,1161) HED,TMP1,SCALE
      IF( NUMBER .LT. 25 ) GOTO 7002
 7003 CONTINUE
      IF( NUMBER .EQ. 0 ) GOTO 1001
      KOUNT = 1
      IDX = 1
      IF( NUMBER .EQ. 1 ) GOTO 7015
      DO 7010 J = 2, NUMBER
      KOUNT = KOUNT + 1
      IF( ADDR(J-1) + 1 .EQ. ADDR(J) ) GOTO 7010
      CALL SOUTPT(4HFIX ,NODCK(1),ADDR(IDX),KOUNT-1,
     *  RSDATA(IDX),ASCALE(IDX))
      KOUNT = 1
      IDX = J
 7010 CONTINUE
 7015 CALL SOUTPT(4HFIX ,NODCK(1),ADDR(IDX),KOUNT,
     *  RSDATA(IDX),ASCALE(IDX))
      IF( HED .NE. ENDCD ) GOTO 7001
      GOTO 1001
C
 7100 CONTINUE
C     $OUTPT -OUTPUT DEFINITION, FLT OPTION
C
 7101 NUMBER = 0
 7102 READ(4,100) HED,TMP1
      IF(HED.EQ.ENDCD)GOTO 7103
      NUMBER = NUMBER + 1
      ADDR(NUMBER)  = IHEXCN(8,TMP1(1))
      WRITE(6,1101) HED,TMP1
      IF( NUMBER .LT. 25 ) GOTO 7102
```

```
 7103 CONTINUE
      IF( NUMBER .EQ. 0 ) GOTO 1001
      KOUNT = 1
      IDX = 1
      IF( NUMBER .EQ. 1 ) GOTO 7115
      DO 7110 J = 2, NUMBER
      KOUNT = KOUNT + 1
      IF( ADDR(J-1) + 2 .EQ. ADDR(J) ) GOTO 7110
      CALL SOUTPT(4HFLT ,NODCK(1),ADDR(IDX),KOUNT-1,RDDATA(IDX),DMY)
      KOUNT = 1
      IDX = J
 7110 CONTINUE
 7115 CALL SOUTPT(4HFLT ,NODCK(1),ADDR(IDX),KOUNT,RDDATA(IDX),DMY)
      IF( HED .NE. ENDCD ) GOTO 7101
      GOTO 1001
C
 7200 CONTINUE
C     $TRACE - TRACE COMMAND
C
      READ(4,103) ONOFF
      WRITE(6,113) ONOFF
      TRCFLG = ONOFF
      GO TO 1
C
 7300 CONTINUE
C     $CHECK -CHECK POINT COMMAND
C
      CALL SCHECK
      GOTO 1
C
 7400 CONTINUE
C     $RESTR - RESTART COMMAND
C
      CALL SRESTR
      GOTO 1
C
 7500 CONTINUE
C     $TERM - TERMINATION COMMAND
C
      TRMFLG=1
      RETURN
C
 7600 CONTINUE
C     $EXEC - EXECUTE SIMULATOR, START AT PC
C
      IF(LOC.NE.0)PC=LOC
      RETURN
```

```
C
7700   CONTINUE
C      $CHKSM - CHECKSUM COMMAND
C
       CALL SCHKSM(LOC)
       GO TO 1
C
 7800  CONTINUE
C      $SETAHX - SET A REGISTER, HEX OPTION
C
       READ(4,100) HED,TMP1
       WRITE(6,110) HED,TMP1
       DATA = IHEXCN(8,TMP1(1))
       CALL SSETA(4HHEX ,DATA,DMY)
       GOTO 1
C
 7900  CONTINUE
C      $SETAFX - SET A REGISTER, FIX OPTION
C
       READ(4,120) HED,WORD,SCALE
       WRITE(6,121) HED,WORD,SCALE
       CALL SSETA(4HFIX ,WORD,SCALE)
       GOTO 1
C
 8000  CONTINUE
C      $SETAFL - SET A REGISTER, FLT OPTION
C
       READ(4,122) HED,FLT
       WRITE(6,123) HED,FLT
       CALL SSETA(4HFLT ,FLT,DMY)
       GOTO 1
C
 8100  CONTINUE
C      $SETB - SET B REGISTER, HEX OPTION
C
       READ(4,100) HED,TMP1
       WRITE(6,110) HED,TMP1
       DATA = IHEXCN(8,TMP1(1))
       CALL SSETB(4HHEX ,DATA,DMY)
       GOTO 1
C
 8200  CONTINUE
C      $SETB - SET B REGISTER, FIX OPTION
C
       READ(4,120) HED,WORD,SCALE
       WRITE(6,121) HED,WORD,SCALE
       CALL SSETB(4HFIX ,WORD,SCALE)
       GOTO 1
```

```
C
 8300 CONTINUE
C      $SETX - SET XR REGISTER
C
       READ(4,100) HED, TMP1
       WRITE(6,110) HED, TMP1
       DATA = IHEXCN(8,TMP1(1))
       CALL SSETX(DATA)
       GOTO 1
C
 8400 CONTINUE
C      $SETR - SET CPU REGISTERS(B1, B2, IXR)
C
       READ(4,124) HED,NUMBER,TMP1
       WRITE(6,125) HED,NUMBER,TMP1
       DATA = IHEXCN(8,TMP1(1))
       CALL SSETR(NUMBER,DATA)
       GOTO 1
 1001 WRITE(6,1102)  HED
       GOTO 1
  100 FORMAT (A4,2(1X,8A1))
  101 FORMAT(20A4)
  102 FORMAT (I2,1X,I4,1X,I10)
  103 FORMAT(I1)
  104 FORMAT (A4,1X,8A1,1X,A4)
  105 FORMAT (8A1)
  106 FORMAT (A4,1X,8A1,2(1X,E14.7))
  107 FORMAT (A4,1X,8A1,1X,D22.0)
  108 FORMAT (4A1)
  109 FORMAT (2(I2,1X))
  110 FORMAT(1X,A4,2(1X,8A1))
  111 FORMAT(1X,20A4)
  112 FORMAT (1X,I2,1X,I4,1X,I10)
  113 FORMAT(1X,I1)
  114 FORMAT(1X,A4,1X,8A1,1X,A4)
  116 FORMAT(1X,A4,1X,8A1,2(1X,E14.7))
  117 FORMAT(1X,A4,1X,8A1,1X,D22.15)
  118 FORMAT(1X,4A1)
  119 FORMAT (1X,2(I2,1X))
  120 FORMAT (A4,2(1X,E14.7))
  121 FORMAT (1X,A4,2(1X,E14.7))
  122 FORMAT (A4,1X,D22.0)
  123 FORMAT (1X,A4,D22.15)
  124 FORMAT (A4,1X,I2,1X,8A1)
  125 FORMAT (1X,A4,1X,I2,1X,8A1)
  200 FORMAT (A8,1X,8A1)
  210 FORMAT(1X,A8,1X,8A1,87X,18HCOMMAND RECOGNIZED)
  300 FORMAT(1X,A8,1X,8A1,87X,17HUNDEFINED REQUEST)
```

```
1101 FORMAT(1X,A4,1X,8A1)
1102 FORMAT(1X,A4)
1161 FORMAT(1X,A4,1X,8A1,1X,E14.7)
9999 STOP
     END
     SUBROUTINE PCSTOP(PCLOC,ACT,BSIZE,ACTN)
     DIMENSION ACTN(1),TYPE(3)
     COMMON/BLIST/BRKLST(1)
     INTEGER PCLOC,ACT,BSIZE,BRKLST,ACTN,TYPE
     DATA TYPE/4H    ,4HTERM,4HDLTE/
     IF(BSIZE.EQ.0)GOTO 15
     DO 10 I=1,BSIZE
     IF(PCLOC.EQ.BRKLST(I))GO TO 20
  10 CONTINUE
  15 BSIZE=BSIZE+1
     IDX = BSIZE
     IF(BSIZE.GT.25)RETURN
     BRKLST(IDX) = PCLOC
     GO TO 30
  20 IDX = I
  30 DO 40 I=1,3
     IF(ACT.EQ.TYPE(I))GO TO 50
  40 CONTINUE
     ACTN(IDX) = TYPE(1)
     RETURN
  50 GO TO (1,2,3),I
   1 ACTN(IDX) = TYPE(1)
     RETURN
   2 ACTN(IDX) = TYPE(2)
     RETURN
   3 BRKLST(IDX) = BRKLST(BSIZE)
     ACTN(IDX) = ACTN(BSIZE)
     BRKLST(BSIZE)=0
     BSIZE=BSIZE-1
     RETURN
     END
```

This page intentionally left blank

APPENDIX B

Operation of the Simulator in Alternate Configurations

APPENDIX B

Operation of the Simulator in Alternate Configurations

B 1 CONFIGURATION SELECTION

This manual may also be used to simulate other available SKC3120 series computers. The Simulator requires a 'Target Machine Configuration File' for initialization. This file is transparent to the user but must be selected for the target machine as described in the Host Procedures Manual. The format of the 'Target Machine Configuration File' is described in the Assembler/Linkage/Editor/Simulator Users Manual.

B 2 MEMORY CONTROL BLOCK

The MEMORY control block may be employed by the user to override the default memory size of 16K words. The size of the memory model is specified by the dimension of the SIMMEM array. The width of the memory is one full-word of the host machine, which is sufficiently large to accommodate 15-, 16- or 19-bit word lengths of the simulated SKC3120 Computer. Table 5-IV presents the information content of the MEMORY control block.

B 3 SIMULATOR DIAGNOSTICS

Simulator diagnostics are identical to those described in Table 5-II except invalid conversion diagnostics will be generated if the value exceeds the limits -2**16 LE x LE (2**16)-1 for 16 bit data words and -2**19 LE x LE (2**19)-1 for 19 bit words.

COMMENTS AND EVALUATIONS

Your evaluation of this document is welcomed by the Singer Company.

Any errors, suggested corrections or general comments may be made and continued on the reverse side. Please include page number and reference paragraph and forward to:

The Singer Company
Aerospace and Marine Systems
Kearfott Division
150 Totowa Road
Wayne, New Jersey 07470
Attention: Department 5760

Name --

Company Affiliation --

Address --

Comments: