TMS32010 SIMULATOR

USER'S GUIDE

IMPORTANT NOTICES

Texas Instruments reserves the right to make changes at any time to improve design and to supply the best possible product for the spectrum of users.

The TMS320 System is copyrighted by Texas Instruments Incorporated, and is the sole property thereof. Use of this product is defined by the license agreement SC-1 between the customer and Texas Instruments. The software may not be reproduced in any form without written permission of Texas Instruments.

Information contained in these publications is believed to be accurate and reliable. However, responsibility is neither assumed for its use nor for any infringement of patents or rights of others that may result from its use. No license is granted by implication or otherwise under any patent or patent right of Texas Instruments or others.

TABLE of CONTENTS

paragraph page

create journal file →

LIST OF FIGURES

APPENDIX

SECTION 1

1.1 General Description

This product is designed to provide simulation of the TMS32010 high-performance microcomputer for effective TMS32010 software development. Installation of this product is described fully in the TMS32010 Installation guide (dependent on the host).

The source code is written in FORTRAN 77 and is essentially split into two sections :

1. the user inter face

2. actual simulation of the chip

The actual simulation of the chip is controlled by one main subroutine with various activities such as memory reads or writes being executed by calls to the appropriate subroutines. The clock counting feature of this simulator is also implemented in this main subroutine.

1.2 Key Features

The key features of the simulator are as follows :

* Ability to simulate either of the chips two modes, the microprocessor mode or the microcomputer mode

* Ability to generate INTERRRUPTs every X number of instructions

* I/O with the 8 ports
  - allows the user to designate a file for each INPUT port as well as for each OUTPUT port

* Breakpoints
  - on Instruction Acquisition
  - Memory Reads or Writes (Data or Program)
  - Data Patterns on the D-Bus or the P-Bus
  - on Error Conditions

* Timing Analysis Relative to Clock Rate

* Trace
  - Accumulator
  - Program Counter
  - Auxiliary Registers

* Immediate execution of an Interrupt or Instruction

* Modify and Display Memory (Data or Program)
  - user can change an entire block at any time
  - user can Initialize memory before any program is loaded

* Modify and Inspect any or all Registers

* Error Messages
  - Illegal Opcodes
  - Invalid Data entry by user

* Execute user commands from a Journal File

* Save states of Simulation so one can Restart simulation

1.3 How to Begin a Debugging Session

Before beginning a debugging session, the user must first write and assemble some 320 code. If the code consists of multiple modules then the user must also link the code.The linked absolute tagged object is what should be loaded into the simulator via the load command ("L", paragraph 2.2). This is the code which will be executed during simulation.

To begin a debugging session, the user must first activate the Fortran TMS320 simulator. As the simulator begins execution it will prompt the user for commands. However, before the user is allowed to implement any of the user commands, he is asked to choose which mode (microprocessor or microcomputer) of the 320 chip he is going to simulate. The series of prompts is as follows:

```
                    SIMULATION OF THE TMS32010
                    VERSION # 0.9.2C- 7.11.21


   0 - MICROPROCESSOR MODE (ADDR 0-1535, OFF CHIP)
   1 - MICROCOMPUTER  MODE (ADDR 0-1535, ON  CHIP)

ENTER VALUE TO SELECT MODE OF OPERATION
1

YOU ARE IN THE MICROCOMPUTER MODE (ADDR 0-1535, ON CHIP)

ENTER COMMAND (D=<CR>):
```

Note that the user may select the microprocessor mode (addr 0-1535, off chip) by either entering a zero or by making no entry and merely pressing the carriage return to accept the default which is the microprocessor mode. Also note that the prompt,

```
ENTER COMMAND (D=<CR>):
```

will appear when it is time for the user to enter one of the user commands discussed in SECTION 2.

1.4 Sample Session

In the following session, the user will enter a program into the program ROM by modifying memory. This program will consist of only two instructions :

1) an "IN" instruction, from port 2 and stored in
   RAM location >0010
   (opcode = 4210)

2) an "OUT" instruction, to port 5 and from RAM
   location >0010
   (opcode = 4D10)

Hence the program when executed will read in a single number and output that same number. Note that this same program could be entered through a load command ("L") if the user created an appropriate file by writing and assembling the two commands listed above.

The series of prompts for entering and executing this program are as listed below. Note that the "BIAQ" command is used to stop execution since the 320 code has no "end" instruction.

```
                    SIMULATION OF THE TMS32010
                    VERSION # 0.9.2C- 7.11.21


   0 - MICROPROCESSOR MODE (ADDR 0-1535, OFF CHIP)
   1 - MICROCOMPUTER  MODE (ADDR 0-1535, ON  CHIP)

ENTER VALUE TO SELECT MODE OF OPERATION
0

YOU ARE IN THE MICROPROCESSOR MODE (ADDR 0-1535, ON CHIP)

ENTER COMMAND (D=<CR>):
ROM
```

```
ENTER STARTING ADDRESS (IN HEX)
0

   0 =    0
4210
   0 = 4210
+
   1 =    0
4D10
   1 = 4D10
Q

ENTER COMMAND (D=<CR>):
BIAQ

BREAK ON INSTRUCTION ACQUISITION
ENTER THE ADDRESS (IN HEX)
3

ENTER COMMAND (D=<CR>):
R

>>PC=   0    OPCODE=4210    IN                    PREVIOUS PC=    0

          ARP      AR0      AR1     TREG      PREG      ACC      CLK
INTEGER     0       16        0        0         0        0        0
HEX         0       10        0        0         0        0        0

>>STK=   0    0    0    0          DP = 0     INTF=0     OV = 0
                                   BIO= 1     INTM=0     OVM= 0

ENTER INPUT VALUE (IN HEX) OR "-" TO RETURN TO MAIN
56

* * * OUTPUT VALUE (IN HEX) IS   56

>>PC=   3    OPCODE=   0    ADD                   PREVIOUS PC=    2

          ARP      AR0      AR1     TREG      PREG      ACC      CLK
INTEGER     0        0        0        0         0        0      (4)->6
HEX         0        0        0        0         0        0      (4)->6

>>STK=   0    0    0    0          DP = 0     INTF=0     OV = 0
                                   BIO= 1     INTM=0     OVM= 0

>>> INSTRUCTION ACQUISITION BREAK POINT #  1 <<<
```

```
ENTER COMMAND (D=<CR>):
RAM                                  { here check to see that location
                                       # >10 does indeed contain the
ENTER STARTING ADDRESS (IN HEX)        value  >0056 }
10
   10 =   56
Q

ENTER COMMAND (D=<CR>):

{ here the user is returned to the operating system }
```

Note that this this example may also be used to verify correct installation of the simulator.

SECTION 2

This section is provided to demonstrate and discuss each of the user commands. It should be noted that the user would be wise to comply with the testing procedure for verifying correct installation before attempting to execute the simulator. This testing procedure is found in paragraph 1.3.

2.1 DM - Display main Menu

The "DM" command will display the main menu shown in figure 2.1. Entering a carriage return by itself will also cause the main menu to be diplayed. It should be noted that all of the user commands are executed from the main menu level, eventhough they may not be found in the main menu but merely referenced. (This is true of the breakpoint and modify/inspect memory commands.) The user is prompted for a command at the main menu level by the appearance of the following :

```
ENTER COMMAND (D=<CR>):
```

He may then enter any of the simulator commands. The command will be executed when the user presses the carriage return after entering the desired command. Note that the user will be prompted for another command as soon as his first command has finished executing.

The "DM" command is executed through this series of prompts:

```
ENTER COMMAND (D=<CR>):
DM
{ here figure 2.1 will be displayed }
ENTER COMMAND (D=<CR>):
```

@DUA0:[UTILITY.TMS320]XASM

FIGURE 2.1 Main Menu of User Commands

```
AVAILABLE COMMANDS ARE:
   BH       = BREAKPOINT HELP
   DM,<CR>  = DISPLAY MAIN MENU
   DT       = DISPLAY THE TRACE BUFFER
   STR      = SAVE THE TRACE BUFFER
   EX       = EXECUTE COMMANDS FROM A GIVEN FILE
   JF       = SELECT JOURNAL FILE
   L        = LOAD NEW OBJECT FILE
   LF       = LIST OF THE FILES ASSIGNED TO PORTS
   MH       = MODIFY AND INSPECT MEMORY HELP
   NB       = NUMBER OF INSTR TILL BREAK
   TIC      = NUMBER OF CLOCK TICS TILL INTERRUPT
   ZTIC     = DISABLES THE TIC COMMAND
   NU       = NUMBER OF INSTR TILL SCREEN UPDATE
   Q        = QUIT SIMULATION
   RH       = MODIFY AND INSPECT REGISTERS/FLAGS HELP
   RS       = RESET SIMULATOR
   R,C      = RUN OR CONTINUE SIMULATION
   SI       = SELECT INPUT PORT FILE
   RSI      = RESET SELECTED INUT PORT FILE
   SO       = SELECT OUTPUT PORT FILE
   ST       = STATUS OF REGISTERS
   SS       = SINGLE STEP
   TR       = TOGGLE TRACE MODE (ON OR OFF)
   Z        = ZERO CLOCK COUNTER
```

2.2 L - Load new object file

This option allows the user to load a different object file for simulation. To load a new object file first enter the "L" command, and the following series of prompts will occur:

```
ENTER COMMAND (D=<CR>):
L                          { LOAD  will also work }
ENTER A NEW OBJECT FILE
NAME.OBJ
* * * * LOADING PROGRAM "NAME     " * * * *

ENTER COMMAND (D=<CR>):
```

This example loads into program rom the new object file NAME.OBJ. From this time on the simulation will be done using the program that was put into the file NAME.OBJ.

2.3 SI - Select Input port file

This option allows the user to associate an input port to a file. If there is already a file associated with the specified port then the new one file will overide the old file. The associated files can be listed using the "LF" command. Whenever an "IN" instruction is executed the file associated with the port will be used to read the data from. If no file has been assigned then the simulation will stop and prompt the user for the input. Once the user has supplied input the simulation resumes. The folowing is an example of the "SI" command:

```
ENTER COMMAND (D=<CR>):
SI

ENTER THE INPUT PORT (0,...,7)
3
ENTER FILE NAME FOR INPUT
NAME.INP

ENTER COMMAND (D=<CR>):
```

This series of communications will cause the file NAME.INP to be associated with INPUT port number 3. Hence any data to be input through port number 3 will be read from the file NAME.INP.

2.3 SO - Select Output port file

This option is very similar to the "SI" command. It allows the user to associate an OUTPUT port to a file. The only notable difference is that if the user wants to execute an "OUT" instruction in his loaded program, he does not have to create an output file before beginning the simulator session. The "SI" command will create the file for the user once he indictes what it is to be called. If there was another file associated with the port then the new one will overide the old one. Hence, execution of an "OUT" instruction will result in a write to the file associated with the appropriate port. The output will default to the screen if no file is associated with the appropriate port.The following is an example of the "SO" command:

```
ENTER COMMAND (D=<CR>):
SO

ENTER THE OUTPUT PORT (0,...,7)
5
ENTER FILE NAME FOR OUTPUT
NAME.OUT

ENTER COMMAND (D=<CR>):
```

This example will result in the file NAME.OUT being associated with output port number 3. Hence when an "OUT" instruction that is directed to port number 3 is executed then there will be a write to the file NAME.OUT.

2.5 LF - List of the Files assigned to ports

This option allows the user to list the input and output ports and the files associated with them. The following is an example of the "LF" command:

```
ENTER COMMAND (D=<CR>):
LF

INPUT PORT #            FILE NAME
____________            _________
     0                  NONE
     1                  NONE
     2                  NONE
     3                  NAME.INP
     4                  NONE
     5                  NONE
     6                  NONE
     7                  NONE

OUTPUT PORT #           FILE NAME
_____________           _________
     0                  NONE
     1                  NONE
     2                  NONE
     3                  NONE
     4                  NONE
     5                  NAME.OUT
     6                  NONE
     7                  NONE

ENTER COMMAND (D=<CR>):
```

This example lists all the ports and the files associated with them. The only two ports that have files associated with them are input port #3 and output port #4.

2.6 BH = Breakpoint Help

The "BH" command is used to display the available breakpoint commands. The limit to the number of breakpoints that the user may have assigned at any one time is 20. Using the "BH" command results in the following series of prompts:

```
ENTER COMMAND (D=<CR>):
BH

{ figure 2.2 is displayed here }

ENTER COMMAND (D=<CR>):
```

The user may now select the desired breakpoint command. It should be noted that the user need not display the breakpoint help menu in order to use a breakpoint command if he remembers the command syntax.

FIGURE 2.2 Breakpoint Help Menu

```
BREAKPOINT COMMANDS ARE:
   BDP      = BREAKPOINT ON DATA PATTERN WHEN R/W FROM/TO DATA RAM
   BPP      = BREAKPOINT ON DATA PATTERN WHEN READ FROM PROGRAM ROM
   BDR      = BREAKPOINT ON DATA RAM READ
   BDRW     = BREAKPOINT ON DATA RAM READ AND WRITE
   BDW      = BREAKPOINT ON DATA RAM WRITE
   BER      = BREAKPOINT ON AN ERROR CONDITION
   BIAQ     = BREAKPOINT ON INSTRUCTION ACQUISITION
   BPR      = BREAKPOINT ON PROGRAM ROM READ
   DB       = DISPLAY ALL BREAKPOINTS
   RB       = REMOVE A BREAKPOINT
```

2.6.1 BDP - Breakpoint on Data Pattern when r/w from/to data ram

This option allows the simulator to halt execution when a certain bit pattern is read(written) from(to) the data RAM. The user will be expected to enter a pattern of ones, zeroes, and Xs. The Xs correspond to a don't care state. Using the "BDP" command results in this series of prompts:

```
ENTER COMMAND (D=<CR>):
BDP

BREAK ON DATA RAM R/W
ENTER BIT PATTERN OF 16 BITS (0,1,X)
```

This example will cause simulation to halt when the value >F0F? (hex) is read(written) from(to) Data RAM . Note that execution will halt after execution of the present instruction has finished.

2.6.2 BPP - Breakpoint on data Pattern when read from Program rom

This option allows the simulator to halt execution when a certain bit pattern is read from the Program ROM. Once again the user is expected to enter a pattern of ones, zeroes, and Xs. The Xs correspond to a don't care state. By entering the command "BPP" the user will receive the following sreies of prompts:

```
ENTER COMMAND (D=<CR>):
BPP

BREAK ON PROGRAM ROM READ
ENTER BIT PATTERN OF 16 BITS (0,1,X)
FIRST BIT IS MSB
0111101111XXXXXX

THE 16 BITS ENTERED ARE:
15 14 13 12 11 10  9  8  7  6  5  4  3  2  1  0
 0  1  1  1  1  0  1  1  1  X  X  X  X  X  X  X

ENTER COMMAND (D=<CR>):
```

This example will cause simulation to halt when a "LST" indirect instruction is read from Program ROM. Execution will halt after execution of the present instruction has finished.

2.6.3 BDR - Breakpoint on Data ram Read

This option allows the simulator to halt execution when a certain address within a range(user specified) of addresses is accessed for a read operation from data memory. Entering the "BDR" command will produce the following prompts:

```
ENTER COMMAND (D=<CR>):
BDR

BREAK ON DATA RAM READ
ENTER THE BEGINNING ADDRESS (IN HEX)
5
ENTER THE ENDING ADDRESS (IN HEX)
F

ENTER COMMAND (D=<CR>):
```

This example will cause the simulation to halt when a read operation is performed on Data Memory within the range of >5 to >f. Execution will halt after execution of the present instruction completes.

2.6.4 BDRW - Breakpoint on Data ram Read and Write

This option allows the simulator to halt execution when a certain address within a range of addresses is accessed for a read or write operation. Entering a "BDRW" command will result in the following series of prompts:

```
ENTER COMMAND (D=<CR>):
BDRW

BREAK ON DATA RAM READ AND WRITE
ENTER THE BEGINNING ADDRESS (IN HEX)
5
ENTER THE ENDING ADDRESS (IN HEX)
F

ENTER COMMAND (D=<CR>):
```

This example will cause the simulation to halt when a read or write operation is performed on Data RAM within the range >5 to >F. Execution will halt after execution of the present instruction has finished.

2.6.5 BDW - Breakpoint on Data ram Write

This option allows the simulator to halt execution when a certain address within a range(user specified) of addresses is accessed for a write operation from data memory. Entering the "BDW" command will produce the following prompts:

```
ENTER COMMAND (D=<CR>):
BDW

BREAK ON DATA RAM WRITE
ENTER THE BEGINNING ADDRESS (IN HEX)
5
ENTER THE ENDING ADDRESS (IN HEX)
F

ENTER COMMAND (D=<CR>):
```

This example will cause the simulation to halt when a write operation is performed on Data Memory within the range of >5 to >f. Execution will halt after execution of the present instruction completes.

2.6.6 BER - Breakpoint on an ERror condition

This option allows the simulator to halt execution when a certain error condition is met. It first lists the error conditions and whether or not they will cause a break in the simulation. Then the user is asked to enter the number of the error condition that he wants changed. If the error condition is on it will be turned off and vice-versa. Then the user will be asked to enter another number. entering nothing (just pressing the carriage return) will terminate the "BER" command. It should be noted that all conditions are initially off with exception of condition numbers 5 and 8.Entering the "ER" command will result in the following prompts:

```
ENTER COMMAND (D=<CR>):
BER

          BREAK ON ERROR CONDITIONS
          ----- -- ----- ----------

1) STACK OVERFLOW  = OFF     2) STACK UNDERFLOW = OFF
2) AR OVERFLOW     = OFF     4) AR UNDERFLOW    = OFF
5) MPY 8000 X 8000 = ON      6) ACC OVERFLOW    = OFF
7) PROGRAM MEMORY ADDRESS >1535      = OFF
8) ATTEMPTED TBL WRITE INTO CHIP ROM = ON

ENTER CONDITION # TO BE TOGGLED
8
ENTER CONDITION # TO BE TOGGLED
4
ENTER CONDITION # TO BE TOGGLED
                                  { user presses carriage return }


ENTER COMMAND (D=<CR>):
```

In this example, auxiliary register underflow or a multiply of >8000 by >8000 will cause the simulator to halt execution. However, if an auxiliary register overflow occurs the simulator will continue executing since the break on AR OVERFLOW is still shut off.

2.6.7 BIAQ - Breakpoint on Instruction Acquisition

This option allows the simulator to halt execution when an instruction is fetched from a given location in Program memory.

```
ENTER COMMAND (D=<CR>):
BIAQ

BREAK ON INSTRUCTION ACQUISITION
ENTER THE ADDRESS (IN HEX)
10

ENTER COMMAND (D=<CR>):
```

This example will cause the simulator to halt when an instruction is fetched from the program memory location >10. Execution will halt before execution of the present instruction (the one from the BIAQ address).

2.6.8 BPR - Breakpoint on Program rom Read

This option allows the simulator to halt execution when a certain address within a range of addresses is accessed as a read operation from program memory. The series of prompts is as follows:

```
ENTER COMMAND (D=<CR>):
BPR

BREAK ON PROGRAM ROM READ
ENTER THE BEGINNING ADDRESS (IN HEX)
1
ENTER THE ENDING ADDRESS (IN HEX)
F

ENTER COMMAND (D=<CR>):
```

This example will cause the simulator to halt when a read is performed on Program Memory within the range >1 to >F. Execution will halt after execution of the present instruction is completed.

2.6.9 BPW - Breakpoint on Program rom Write

This option allows the simulator to halt execution when a certain address within a range of addresses is accessed as a write operation from program memory. The series of prompts is as follows:

```
ENTER COMMAND (D=<CR>):
BPW

BREAK ON PROGRAM ROM WRITE
ENTER THE BEGINNING ADDRESS (IN HEX)
1
ENTER THE ENDING ADDRESS (IN HEX)
F

ENTER COMMAND (D=<CR>):
```

This example will cause the simulator to halt when a write is performed on Program Memory within the range >1 to >F. Execution will halt after execution of the present instruction is completed.

2.6.10 DB - Display all Breakpoints

This option allows the user a list of all the Breakpoints presently assigned. The following is an example of how the "DB" command will work:

```
ENTER COMMAND (D=<CR>):
DB

REF#    SET BY      ADDRESS                 VALUE
____    ______      _______                 _____
  1     BDP                           >1111000011111XXXX
  2     BIAQ    >  10
  3     BDR     >   5 - >   F

ENTER COMMAND (D=<CR>):
```

Note that this listig of breakpoints would indicate that there were only three breakpoints currently set.

2.6.11 RB - Remove a Breakpoint

This option allows the user to remove a breakpoint by using the breakpoints reference number (see 2.6.10). The reference number is displayed under the REF# heading when the "DB" command is issued. The following is an example of how to remove a breakpoint.

```
ENTER COMMAND (D=<CR>):
RB

ENTER A BREAKPOINT REFERENCE NUMBER
2

BREAKPOINT DELETED
BIAQ        >  10

ENTER COMMAND (D=<CR>):
```

This example will delete the breakpoint that was created with a "BIAQ" command and which had the address >10 assigned to it. Note that the breakpoints which had a reference number bigger than three before three was deleted will now have a reference number one number lower than what they were.

2.7 MH - modify and inspect Memory Help

The "MH" command is used to display the available modify/inspect memory commands. Using the "MH" command produces the following series of prompts:

```
ENTER COMMAND (D=<CR>):
MH

{ figure 2.3 is displayed here }

ENTER COMMAND (D=<CR>):
```

Note that any of the modify/inspect commands may be used without displaying the Memory Help menu.

FIGURE 2.3 Memory Help Menu

```
MODIFY AND INSPECT MEMORY COMMANDS ARE:
   RAM  = MODIFY/INSPECT INDIVIDUAL DATA RAM LOCATIONS
   ROM  = MODIFY/INSPECT INDIVIDUAL PROGRAM ROM LOCATIONS
   RAMH = DISPLAY DATA RAM IN HEX
   RAMI = DISPLAY DATA RAM IN INTEGER
   ROMH = DISPLAY PROGRAM ROM IN HEX
   ROMI = DISPLAY PROGRAM ROM IN INTEGER
```

2.7.1 RAM - modify/inspect individual data RAM

This option allows the user to scan Data RAM memory and modify the contents of any location in Data RAM. Initially the user is asked for a starting address (where does he want to start the scan). Once he chooses a starting address, the address along with the contents of that location will be displayed.

The user then has several options. He may enter any one of the following commands any repeated number of times (until he decides to quit scanning the memory by entering the "Q" command):

"+" , displays the memory address (+1) and its contents
"-" , displays the memory address (-1) and its contents
"#" , modifies the contents of the memory address currently being displayed so that it then contains the number "#" where "#" is just a hex number
"@#", displays the memory address "#" (where number is just any hex number) and its contents regardless of what address was previously displayed

The following series of prompts will occur when using the "RAM" command:

```
ENTER COMMAND (D=<CR>):
RAM

ENTER STARTING ADDRESS (IN HEX)
1

   1 =   0
+
   2 =   0
-
   1 =   0
10
   1 =  10
@130
* * * STARTING ADDRESS CHANGED * * *
 130 =   0
Q

ENTER COMMAND (D=<CR>):
```

This example is self explanatory.

2.7.2 ROM - modify/inspect individual program ROM

This option allows the user to scan Program ROM memory and modify the contents of any location in Program ROM. Initially the user is asked for a starting address (where does he want to start the scan). Once he chooses a starting address, the address along with the contents of that location will be displayed.

The user then has several options. He may enter any one of the following commands any repeated number of times (until he decides to quit scanning the memory by entering the "Q" command):

"+" , displays the memory address (+1) and its contents
"-" , displays the memory address (-1) and its contents
"#" , modifies the contents of the memory address currently being displayed so that it then contains the number "#" where "#" is just a hex number
"@#", displays the memory address "#" (where number is just any hex number) and its contents regardless of what address was previously displayed

The following series of prompts will occur when using the "ROM" command:

```
ENTER COMMAND (D=<CR>):
ROM

ENTER STARTING ADDRESS (IN HEX)
44

  44 =   0
+
  45 =   0
-
  44 =   0
10
  44 =  10
@111
* * * STARTING ADDRESS CHANGED * * *
 111 =   0
Q

ENTER COMMAND (D=<CR>):
```

This example is self explanatory.

2.7.3 RAMH - display data RAM in Hex

This option allows the user to display the Data RAM memory in block style with the contents of each address being displayed as hex numbers. There is no need for a starting or an ending address since the whole RAM will be displayed. The "RAMH" resuts in the following series of prompts:

```
ENTER COMMAND (D=<CR>):
RAMH
```

	0	1	2	3	4	5	6	7	8
0	10	0	0	0	0	0	0	0	0
10	20	0	0	0	0	0	0	0	0
20	0	0	0	0	0	0	0	0	0
30	0	0	0	0	0	0	0	0	0
40	0	0	0	0	0	0	0	0	0
50	0	0	0	0	0	0	0	0	0
60	0	0	0	0	0	0	0	0	0
70	0	0	0	0	0	0	0	0	0
80	0	0	0	0	0	0	0	0	0
90	0	0	0	0	0	0	0	0	0
100	0	0	0	0	0	0	0	0	0
110	0	0	0	0	0	0	0	0	0
120	0	0	0	0	0	0	0	0	0
130	0	0	0	0	0	0	0	0	0
140	0	0	0	0	0	0	0	0	0

```
ENTER COMMAND (D=<CR>):
```

Note that the >10 in address location 0 is a hex number as is the >20 in location 10. Note also that the addresses are given as decimal integers, they are not in hex.

2.7.4 RAMI - display data RAM in Integer

This option allows the user to display the Data RAM memory in block style with the contents of each address being displayed as integer numbers. There is no need for a starting or an ending address since the whole RAM will be displayed. The "RAMI" resuts in the following series of prompts:

```
ENTER COMMAND (D=<CR>):
RAMI                    { RAMD will also work }
```

	0	1	2	3	4	5	6	7	8
0	16	0	0	0	0	0	0	0	0
10	32	0	0	0	0	0	0	0	0
20	0	0	0	0	0	0	0	0	0
30	0	0	0	0	0	0	0	0	0
40	0	0	0	0	0	0	0	0	0
50	0	0	0	0	0	0	0	0	0
60	0	0	0	0	0	0	0	0	0
70	0	0	0	0	0	0	0	0	0
80	0	0	0	0	0	0	0	0	0
90	0	0	0	0	0	0	0	0	0
100	0	0	0	0	0	0	0	0	0
110	0	0	0	0	0	0	0	0	0
120	0	0	0	0	0	0	0	0	0
130	0	0	0	0	0	0	0	0	0
140	0	0	0	0	0	0	0	0	0

```
ENTER COMMAND (D=<CR>):
```

Note that the 16 in address location 0 is an integer number as is the 32 in location 10. Note also that the addresses are given as decimal integers, they are not in hex.

2.7.5 ROMH - display program ROM in Hex

This option allows the user to display an entire block of Program ROM memory (160 addresses and their contents). Here, the user is asked to enter a starting address. A block of 160 consecutive words of Program ROM memory is then displayed. The series of prompts is as follows:

```
ENTER COMMAND (D=<CR>):
ROMH

ENTER STARTING ADDRESS (IN HEX)
0

>>PC
   0     10     0     0     0     0     0     0     0
   8     20     0     0     0     0     0     0     0
  10      0     0     0     0     0     0     0     0
  18      0     0     0     0     0     0     0     0
  20      0     0     0     0     0     0     0     0
  28      0     0     0     0     0     0     0     0
  30      0     0     0     0     0     0     0     0
  38      0     0     0     0     0     0     0     0
  40      0     0     0     0     0     0     0     0
  48      0     0     0     0     0     0     0     0
  50      0     0     0     0     0     0     0     0
  58      0     0     0     0     0     0     0     0
  60      0     0     0     0     0     0     0     0
  68      0     0     0     0     0     0     0     0
  70      0     0     0     0     0     0     0     0
  78      0     0     0     0     0     0     0     0
  80      0     0     0     0     0     0     0     0
  88      0     0     0     0     0     0     0     0
  90      0     0     0     0     0     0     0     0

ENTER COMMAND (D=<CR>):
```

Note that the >10 in location 0 is a hex number as is the >20 in location 8. Note also that the addresses are displayed in hex.

2.7.6 ROMI - display program ROM in Integer

This option allows the user to display an entire block of Program ROM memory (160 addresses and their contents). Here, the user is asked to enter a starting address. A block of 160 consecutive words of Program ROM memory is then displayed. The series of prompts is as follows:

```
ENTER COMMAND (D=<CR>):
ROMI                    { ROMD  will also work }

ENTER STARTING ADDRESS (IN HEX)
0

>>PC
   0     16      0      0      0      0      0      0      0
   8     32      0      0      0      0      0      0      0
  10      0      0      0      0      0      0      0      0
  18      0      0      0      0      0      0      0      0
  20      0      0      0      0      0      0      0      0
  28      0      0      0      0      0      0      0      0
  30      0      0      0      0      0      0      0      0
  38      0      0      0      0      0      0      0      0
  40      0      0      0      0      0      0      0      0
  48      0      0      0      0      0      0      0      0
  50      0      0      0      0      0      0      0      0
  58      0      0      0      0      0      0      0      0
  60      0      0      0      0      0      0      0      0
  68      0      0      0      0      0      0      0      0
  70      0      0      0      0      0      0      0      0
  78      0      0      0      0      0      0      0      0
  80      0      0      0      0      0      0      0      0
  88      0      0      0      0      0      0      0      0
  90      0      0      0      0      0      0      0      0

ENTER COMMAND (D=<CR>):
```

Note that the 16 in location 0 is an integer number as is the 32 in location 8. Note also that the addresses are displayed in hex.

2.8 RH - modify/inspect Registers/flags Help

This option allows the user to display the available modify and inspect register commands. Using the "RH" command results in the following series of prompts:

```
ENTER COMMAND (D=<CR>):
RH

{ figure 2.4 is displayed here }

ENTER COMMAND (D=<CR>):
```

Note that the user need not display the help menu in order to use one of the commands.

* NOTE: In all modify/inspect registers/flags commands, pressing the carriage return after the presnt value has been displayed will result in the present value remaining unchanged,ie. the result is that the user only inspects the register/flag and does not change its value.

FIGURE 2.4 Registers/flags Help Menu

```
MODIFY REGISTERS/FLAGS COMMANDS ARE:
   ACC      = MODIFY/INSPECT ACCUMULATOR
   AR       = MODIFY/INSPECT AUXILIARY REGISTERS
   ARP      = MODIFY/INSPECT AUXILIARY REGISTER POINTER
   BIO      = MODIFY/INSPECT I/O BRANCH CONTROL
   CC       = MODIFY/INSPECT CLOCK COUNTER
   DP       = MODIFY/INSPECT DATA MEMORY PAGE POINTER
   INTF     = MODIFY/INSPECT INTERRUPT FLAG REGISTER
   INTM     = MODIFY/INSPECT INTERRUPT FLAG MODE REGISTER
   OV       = MODIFY/INSPECT OVERFLOW FLAG REGISTER
   OVM      = MODIFY/INSPECT OVERFLOW MODE REGISTER
   P        = MODIFY/INSPECT P REGISTER
   PC       = MODIFY/INSPECT PROGRAM COUNTER
   SK       = MODIFY/INSPECT STACK
   T        = MODIFY/INSPECT T REGISTER
```

2.8.1 ACC - modify/inspect the ACCumulator

This option will permit the user to inspect and change the accumulator. The present accumulator value is displayed and then the user can enter a new value or leave it the same (see * in 2.8). Entering the "ACC" command will produce the following prompts:

```
ENTER COMMAND (D=<CR>):
ACC
PRESENT ACCUMULATOR VALUE
>       0
ENTER NEW VALUE (IN HEX)
10

ENTER COMMAND (D=<CR>):
```

This example is self explanatory.

2.8.2 AR - modify/inspect ARxiliary registers

This option allows the user to inspect or change any of the auxiliary registers. He is first prompted for the auxiliary register number and he is then allowed to inspect(see * in 2.8) and/or change that register. The following is an example of how this will look:

```
ENTER COMMAND (D=<CR>):
AR

ENTER THE AUXILIARY REGISTER NUMBER(0 OR 1) OR ENTER "-" TO TERMINATE
1
AR1 = >0010
ENTER NEW VALUE (IN HEX)
7
ENTER THE AUXILIARY REGISTER NUMBER(0 OR 1) OR ENTER "-" TO TERMINATE
-

ENTER COMMAND (D=<CR>):
```

This example is self explanatory.

2.8.3 ARP - modify/inspect Auxiliary Register Pointer

This option allows the user to modify and/or inspect (see * in 2.8) the auxiliary register pointer. The prompts appear as follows:

```
ENTER COMMAND (D=<CR>):
ARP

PRESENT VALUE OF THE AUXILIARY POINTER
1
ENTER NEW VALUE (0,1)
0

ENTER COMMAND (D=<CR>):
```

This example is self explanatory.

2.8.4 BIO - modify/inspect the I/O Branch control

This option allows the user to modify and/or inspect (see * in 2.8) the I/O Branch control pin. The prompts appear as follows:

```
ENTER COMMAND (D=<CR>):
BIO

PRESENT VALUE OF THE I/O BRANCH CONTROL
1
ENTER NEW VALUE (0,1)
0

ENTER COMMAND (D=<CR>):
```

This example is self explanatory.

2.8.5 CC - modify/inspect the Clock Counter

This option allows the user to modify and/or inspect (see * in 2.8) the clock counter. The prompts appear as follows:

```
ENTER COMMAND (D=<CR>):
CC

PRESENT VALUE FOR CLOCK COUNTER
>  50
ENTER A NEW VALUE FOR THE CLOCK COUNTER (IN HEX)
10

ENTER COMMAND (D=<CR>):
```

This example is self explanatory.

2.8.6 DP - modify/inspect Data memory Page pointer

This option allows the user to modify and/or inspect (see * in 2.8) the data memory page pointer. The prompts appear as follows:

```
ENTER COMMAND (D=<CR>):
DP

PRESENT VALUE OF THE DATA MEMORY PAGE POINTER
1
ENTER NEW VALUE (0,1)
0

ENTER COMMAND (D=<CR>):
```

This example is self explanatory.

2.8.7 INTF - modify/inspect INTerrupt Flag register

This option allows the user to modify and/or inspect (see * in 2.8) the interrupt flag register. The prompts appear as follows:

```
ENTER COMMAND (D=<CR>):
INTF

PRESENT VALUE OF THE INTERRUPT FLAG REGISTER
1
ENTER NEW VALUE (0,1)
0

ENTER COMMAND (D=<CR>):
```

In this example the interrupt flag value is changed to 1. Note that the next time the simulator is started running an interrupt will occur if the interrupt mode register is equal to zero.

2.8.8 INTM - modify/inspect INTerrupt Mode register

This option allows the user to modify and/or inspect (see * in 2.8) the interrupt mode register. The prompts appear as follows:

```
ENTER COMMAND (D=<CR>):
INTM

PRESENT VALUE OF THE INTERRUPT MODE REGISTER
0
ENTER NEW VALUE (0,1)
1

ENTER COMMAND (D=<CR>):
```

In this example the interrupt mode value is changed to one. Note that whether the interrupt flag register is one or zero no interrupt will occur when simulation is resumed.

2.8.9 OV - modify/inspect OVerflow flag

This option allows the user to modify and/or inspect (see * in 2.8) the overflow flag register. The prompts appear as follows:

```
ENTER COMMAND (D=<CR>):
OV

PRESENT VALUE OF THE OVERFLOW FLAG REGISTER
1
ENTER NEW VALUE (0,1)
0

ENTER COMMAND (D=<CR>):
```

In this example the overflow flag is changed to a zero which means that no overflow ha occurred. (a 1 means that an overflow has occurred).

2.8.10 OVM - modify/inspect OVerflow Mode register

This option allows the user to modify and/or inspect (see * in 2.8) the overflow mode register. The prompts appear as follows:

```
ENTER COMMAND (D=<CR>):
OVM

PRESENT VALUE OF THE OVERFLOW MODE REGISTER
1
ENTER NEW VALUE (0,1)
0

ENTER COMMAND (D=<CR>):
```

In this example the overflow mode is changed from a one to a zero. This will cause the accumulator to not saturate on an overflow.

2.8.11 P - modify/inspect P register

This option allows the user to modify and/or inspect (see * in 2.8) the P register. The prompts appear as follows:

```
ENTER COMMAND (D=<CR>):
P
PRESENT VALUE OF THE P REGISTER
>       0
ENTER NEW VALUE (IN HEX)
890F

ENTER COMMAND (D=<CR>):
```

This example is self explanatory.

2.8.12 PC - modify/inspect Program Counter

This option allows the user to modify and/or inspect (see * in 2.8) the program counter. Since the program counter is only twelve bits the user should enter at most three hex digits. The prompts appear as follows:

```
ENTER COMMAND (D=<CR>):
PC
PRESENT VALUE FOR PROGRAM COUNTER
> 23
ENTER NEW VALUE (IN HEX)
0

ENTER COMMAND (D=<CR>):
```

This example will change the program counter from >023 to >000 which will cause the simulator to start executing at >000 when simulation is started.

2.8.13 SK - modify/inspect StacK

This option allows the user to inspect and change all four levels of the stack. The stack will be displayed in order from the top of the stack to the bottom of the stack. As each level is shown the user will be able to change that level. If he decides not to change the value he may just enter a carriage return and then procede on to the next level. If the user wants to terminate before he reaches the bottom of the stack all he needs to do is enter a "-" and the "SK" command will return to the Modify Registers menu. The following is an example of how the "SK" command will work:

NO. Returns to "ENTER COMMAND" Prompt.

```
ENTER COMMAND (D=<CR>):
SK

TOP OF STACK - 0
>548
ENTER NEW VALUE (IN HEX) OR "-" TO TERMINATE


TOP OF STACK - 1
>52F
ENTER NEW VALUE (IN HEX) OR "-" TO TERMINATE
000

TOP OF STACK - 2
>548
ENTER NEW VALUE (IN HEX) OR "-" TO TERMINATE
-

ENTER COMMAND (D=,<CR>):
```

This example did not change the top of the stack but zeroed the second level of the stack. At the third level a "-" was entered which terminated the process of modifying the stack.

2.8.14 T - modify/inspect T register

This option allows the user to modify and/or inspect (see * in 2.8) the T register. The prompts appear as follows:

```
ENTER COMMAND (D=<CR>):
T
PRESENT VALUE OF THE P REGISTER
>     F89
ENTER NEW VALUE (IN HEX)
0

ENTER COMMAND (D=<CR>):
```

This example is self explanatory.

2.9 ST - STatus of registers

This option allows the user to simultaneously display the contents of the program counter, the previous value of the program counter, the opcode (plus its mnemonic name) of the current instruction, and all four stack locations in hex.It will also display the value of the following:

auxiliary registers	auxiliary register pointer
T register	P register
accumulator	clock counter
data memory page pointer	I/O branch control
interrupt flag register	interrupt mode register
overflow flag register	overflow mode register

Using the "ST" command results in the following series of prompts:

```
ENTER COMMAND (D=<CR>):
ST

>>PC=   0    OPCODE=   0    ADD                    PREVIOUS PC=    0

          ARP      AR0      AR1     TREG      PREG      ACC      CLK
INTEGER     1       16        0        0         0        0        0
HEX         1       10        0        0         0        0        0

>>STK=   2    1    4    0          DP = 1     INTF=0     OV = 0
                                   BIO= 1     INTM=1     OVM= 0
```

Note that the stack is displayed with the left-most element being the top of the stack and the right-most element being the bottom of the stack.

2.10 NB - Number of instructions till Break

This option allows the user to have the simulation halt execution after a specified number of instructions have been executed. If the number is set to zero then this command is void. A carriage return for the new value will leave the present value unchanged. The default for this command is zero. Using the "NB" command results in the following series of prompts:

```
ENTER COMMAND (D=<CR>):
NB
ENTER NUMBER OF INSTRUCTIONS TILL BREAK
5

ENTER COMMAND (D=<CR>):
```

Note that the new number is entered as a decimal integer.

2.11 NU - Number of instructions till screen Update

This option allows the user, when running the simulation, to have a certian number of instructions execute before the next line of output to the screen. If the value is one then the user will see output every instruction. This is mainly used to keep the number of lines of output to the screen to a minimun when the program is just about error free. If the value is set to zero then there is no output to the screen when the program is running and the only way to see what is going on is to break the simulation by breakpoint or some other means. The default value for this command is zero. The following is an example of the "NU" command.

```
ENTER COMMAND (D=<CR>):
NU

ENTER NUMBER OF INSTRUCTIONS TILL SCREEN UPDATE
12

ENTER COMMAND (D=<CR>):
```

This example changed the number of instructions till screen update to 12. Note that the new number should be entered as a decimal integer.

2.12 NI - Number of instructions till Interrupt

The "NI" command allows the user to have periodic interrupts while running the simulation. To use this command the user must first type an "NI" after command prompt. This will allow the user to enter the number of instructions that will execute before the next interrupt will occur. Once the interrupt occurrs, the simulator will put PC + 1 on top of the stack, put a two into the PC, Disable interrupts (INTM = 1), and clear the interrupt flag (INT = 0). This interrupt will occur every X instructions where X is the number entered after the "NI" command. The default value for this is 0 where a 0 will mean that no interrupts will occur. The following is an example of how the "NI" command will work:

```
ENTER COMMAND (D=<CR>):
NI

ENTER NUMBER OF INSTRUCTIONS TILL INTERRUPT
10

ENTER COMMAND (D=<CR>):
```

This example will cause a simulated interrupt to occur every 10 instructions. Note that the new number should be entered as a decimal integer.

2.13 TR - toggle TRace (on or off)

This option toggles the trace mode on or off. The trace is a circular buffer that traces the auxiliary registers, accumulator, and the program counter. It is 256 samples long. So the user, with the trace on, can look at the last 256 states of the simulation. The user can display the trace buffer using the "DT" command. The default for the trace is off. The following is an example of the "TR" command:

```
ENTER COMMAND (D=<CR>):
TR

TRACE MODE IS ON

ENTER COMMAND (D=<CR>):
```

In this example, the the trace mode is toggled from off to on.

2.14 DT - Display the Trace buffer

This option allows the user to display the trace buffer (described in 2.13). Using the "DT" command results in the following series of prompts:

```
ENTER COMMAND (D=<CR>):
DT

PC=   1     ACC=       2     AR0=  FF      AR1=   0
PC=   2     ACC=       4     AR0=  CD      AR1=   0
PC=   3     ACC=       6     AR0=  A8      AR1=   0
PC=   6     ACC=       8     AR0=  10      AR1=   0
PC=   7     ACC=      10     AR0=  10      AR1=  C6

ENTER COMMAND (D=<CR>):
```

Note that the register values are displayed in hex. Also a warning will appear at the beginning of the display if the trace was longer than 256 states.

2.15 EX - EXecute commands from a given file

This option allows for execution of commands from a file that was created using the "JF" command. To execute commands from a given file first enter the "EX" command. Then the following series of prompts will occur:

```
ENTER COMMAND (D=<CR>):
EX

ENTER FILE NAME
JOURNAL.TXT

{ here, the commands found in the file JOURNAL.TXT are displayed as
  they are executed }

ENTER COMMAND (D=<CR>):
```

2.16 JF - select Journal File

This option allows the user to have the information that is being entered saved in a file, so when the simulation is started again the user can just give the name of the file and the program will execute the commands that were saved in that file. If there is already a file then the user may leave it the same by pressing the carriage return when prompted for a new file. If he wishes to change the file he must enter a new file name. One of the two following series of prompts will occur when using the "JF" command:

1)
```
ENTER COMMAND (D=<CR>):
JF

A JOURNAL FILE HAS NOT BEEN CREATED
ENTER FILE NAME
JOURNAL.TXT

ENTER COMMAND (D=<CR>):
```

2)
```
ENTER COMMAND (D=<CR>):
JF

JOURNAL FILE = JOURNAL

ENTER NEW FILE NAME
NEWJOURNAL.TXT


ENTER COMMAND (D=<CR>):
```

Note that when the "JF" command is invoked, the rest of the simulator session is recorded in the "journal" file. (ie., until simulation is halted with a "Q" command)

2.17 RS - Reset Simulator

This option causes a reset to occur. The reset consists of loading the program counter with zero, clearing the overflow flag register (OV = 0), setting the interrupt mode register to one to disable interrupts (INTM = 1), and clearing any pending interrupts (INTM = 0). This does not start the simulator running. This command followed by the run command ("R") will perform the same set of actions as a hardware reset. Using the "RS" command produces the following series of prompts:

```
ENTER COMMAND (D=<CR>):
RS

ENTER COMMAND (D=<CR>):
```

2.18 SS - Single Step execution

This option allows the user to halt simulation after the execution of each instruction of the loaded program. Once the "SS" command has been invoked, a carriage return causes the execution of another instruction. Single step simulation is halted by entering a "-". It should be noted that the "ST" command (show status of registers) is automatically implemented after an instruction of the loaded progam is executed so that the new status of the various registers is displayed after the execution of every single instruction. The prompts appear as follows:

```
ENTER COMMAND (D=<CR>):
SS

{ the "ST" command is implemented here }

ENTER <CR> TO CONTINUE
      "-" TO TERMINATE
-

ENTER COMMAND (D=<CR>):
```

This example causes the simulator to halt after executing one instruction. The "-" terminates the single step session.

2.19 R,C - Start or Continue the simulation

This option allows the user to begin simulation. This command may be invoked by entering either "R" or "C". The simulation may be stopped by encountering a breakpoint, reaching the given limit on the number of instructions that may be executed ("NB" command), or by striking a designated key while the simulator is in a run mode.

NOTE :

> The designated key for VAX host is a <CTRL>C . However, if an MSDOS based host is used then any key except for a <CTRL>C will cause simulation to be halted when it is entered from the board.

It should be noted that when the user enters the designated key the "ST" command is executed and that then he is returned to the command entry level of the simulator. The following is an example of the "R" or "C" command:

```
ENTER COMMAND (D=<CR>):
R                        { RUN  &  C  will also work }

{ here the "ST" command is automatically implemented }

ENTER COMMAND (D=<CR>):
```

Note that the "R" or "C" commands are like the "SS" command in that when simulation is halted, the "ST" command is automatically implemented. Hence, the value of the variuos registers is displayed.

2.20 Z - Zero clock counter

This option zeroes the clock counter. The clock counter counts the number of clock cycles that have occurred since the simulation was started or the clock counter was zeroed. The following is an example of the "Z" command:

```
ENTER COMMAND (D=<CR>):
Z

CLOCK COUNTER HAS BEEN ZEROED

ENTER COMMAND (D=<CR>):
```

This example is self explanatory.

2.21 Q - Quit simulation

This option terminates the simulation session and returns the user to the operating system. The command "Q" is entered as follows:

```
ENTER COMMAND (D=<CR>):
Q                          { QUIT  will also work }

{ here the user is returned to the operating system }
```

This example is self explanatory.

2.22 TIC - number of clock TICs till next interrupt

This option allows the user generate interrupts every X number of clock tics. The user is first asked how often he wants an interrupt to occur, then he asked how many times he would like the interrupt to be generated. Hence, the user may specify that that there is to be an interrupt generated every 1000 clock tics until five interrupts have been genereated. The series of prompts for this command are as follows :

```
ENTER COMMAND (D=<CR>):
TIC

ENTER THE NUMBER OF CLOCK TICS TILL INTERRUPT
1000
ENTER THE NUMBER OF TIMES TO REPEAT THE INTERRUPT CYCLE
5

ENTER COMMAND (D=<CR>):
```

This example is self explanatory.

2.23 ZTIC - disable the TIC command

This option allows the user to stop generating the interrupts specified by the TIC command. The prompts are as follows :

```
ENTER COMMAND (D=<CR>):
ZTIC

THE TIC COMMAND HAS BEEN DISABLED

ENTER COMMAND (D=<CR>):
```

This example is self explanatory.

2.24 RSI - Reset Selected Input port file

This option allows the user to reset an input port file so that data is taken from the top of the file. (ie. pointer is repositioned to the top of the file) The prompts are as follows :

```
ENTER COMMAND (D=<CR>):
RSI

ENTER THE INPUT PORT (0,...,7)
2
INPUT PORT FILE  # 2 HAS BEEN RESET

ENTER COMMAND (D=<CR>):
```

The file associated with port #2 (by the "SI" command) has been reset. It should be noted that it is not neccessary to use this command when the end of a file is reached since the simulator does exhibit an auto wrap feature. Once the end of a file has been reached. Any attempt to read from the file again will result in an automatic reset (as described above).

2.25 STR - Save the TRace buffer

This option allows the user to save the contents of the trace buffer in a file for retrieval after the simulator session is halted. Entering the "STR" command will result in one of the two following series of prompts:

1.) ENTER COMMAND (D=<CR>):
STR

ENTER TRACE BUFFER FILE NAME - NONE EXISTS
TRACE.DAT

2.) ENTER COMMAND ((D=<CR>):
STR

TRACE BUFFER FILE ALREADY EXISTS
ENTER NEW NAME
TRACE1.DAT

ENTER COMMAND (D=<CR>):

It should be noted that the second prompt occurs only if the "STR" command has been previously used during the same session. Also, failure to indicate a file name in 1.) will cause the default file FOR088.DAT to be used. Failure to indicate a new file in 2.) will result in the old file being used to save the current trace buffer. The old file in this case will be appended to, not over written.

APPENDIX A

The following is a list of the various run time stop codes that may occur during execution of a loaded program. Exactly one of these "stopcodes" is displayed each time execution of a program is suspended.

Note that illegal trap codes should never be seen by the user. These traps indicate the existence of states which will not occur in a properly functioning simulator.

STOPCODES:

2600	Illegal Trap
2695	Break on Data Read
2780	Illegal Trap
2795	Break on Output Write
3505	Illegal Trap
3665	Break on Table Read
4055	Break on Table Write
4065	Break on Table Write
4190	Illegal Trap
7601	Illegal Opcode
8405	Break on Instruction Acquisition
8662	Illegal Indirect Addressing Structure (bits 1,2 and 6 are not zero)
8670	Illegal Indirect Addressing Structure (bits 4 and 5 are both on)
8680	Break on Data Memory Read (during development of indirect addressing)
8683	Illegal Trap
9011	Branch to Self
9020	Break on Instruction Acquisition
9105	Illegal Trap
9950	Accumulator Was Used First Clock Cycle After an "SUBC"
10000	"Steps" Expired
10100	Addressed Beyond End of 1536 Word Program ROM
10144	Addressed Beyond End of 144 Word Data RAM
10400	Error Breakpoint (over/underflow, etc.)
10496	Addressed Beyond End of 4096 Word Program ROM
11000+N	Instruction Acquisition Breakpoint #N
12000+N	Program ROM Breakpoint #N
13000+N	Data Ram Breakpoint #N

APPENDIX B

It should be noted that the <control>C trap, which may be used to exit the run mode of the simulator without halting the session entirely, is dependent on the VMS 3.2 operating system. Hence, transportation of the simulator to another operating system will require editing of the following modules: CHIPSM, GRSAST, and GRSCTLC.