SIMH Users' Guide, V4.0
14-Feb-2022

COPYRIGHT NOTICE

The following copyright notice applies to the SIMH source, binary, and documentation:

> Original code published in 1993-2017, written by Robert M Supnik
> Copyright (c) 1993-2017, Robert M Supnik
>
> Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:
>
> The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.
>
> THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL ROBERT M SUPNIK BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
>
> Except as contained in this notice, the name of Robert M Supnik shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Software without prior written authorization from Robert M Supnik.

Introduction

This memorandum documents the SIMH simulators. These simulators are freeware; refer to the license terms above for conditions of use. Support is not available. The best way to fix problems or add features is to read and modify the sources yourself. Alternately, you can send Internet mail to simh AT trailing-edge DOT com, but a response is not guaranteed.

The simulators use a common command interface. This memorandum describes the features of the command interface. The details of each simulator are documented in separate, machine-specific memoranda.

1 Running A Simulator

To start the simulator, simply type its name. (On VMS the simulators must then be defined as foreign commands in order to be started by name.) The simulator recognizes three command line switches: -q, -v, and -e. If -q is specified, certain informational messages are suppressed. The -v and -e switches pertain only to command files and are described in Section 3.13.

The simulator interprets the arguments on the command line, if any, as the file name and arguments for a `DO` command:

```
% pdp10 {switches} {<startup file> {arg,arg,...}}
```

If no file is specified on the command line, the simulator looks for a startup file consisting of the simulator name (including its path components) plus the extension .ini. If a startup file is specified, either on the command line or implicitly via the .ini capability, it should contain a series of non-interactive simulator command, one per line. These commands can be used to set up standard parameters, for example, disk sizes.

After initializing its internal structures and processing the startup file (if any), the simulator types out its name and version and then prompts for input with:

```
sim>
```

2 Compiling A Simulator

The simulators have been tested on VAX VMS, Alpha VMS, IA64 VMS, Alpha UNIX, NetBSD, FreeBSD, OpenBSD, Linux, Solaris, Windows 9x/NT/2000/XP/Win7/Win8, MacOS X, and OS/2. Porting to other environments will require changes to the operating system dependent code in the SIMH libraries (sim_fio.c, sim_timer.c, sim_console.c, sim_ether.c, sim_sock.c, sim_disk.c, sim_serial.c).

The simulator sources are provided in a zip archive and are organized hierarchically. Source files for the simulator framework components are in the top level directory; source files for each simulator are in individual subdirectories. Note that the include files in the top level directory are referenced from the subdirectories, without path identifiers. Your build tool needs to search the top level directory for include files not present in the simulator-specific directory. File manifests for each simulator are given in that simulator's documentation.

The simulators recognize or require a few compile-time #defines:

- The 18b simulators require that the model name be defined as part of the compilation command line (i.e., PDP4 for the PDP-4, PDP7 for the PDP-7, PDP9 for the PDP-9, PDP15 for the PDP-15).

- The PDP-10 and IBM 7094 simulators use 64b integer variables, requiring that USE_INT64 be defined as part of the compilation command line. Since 64b integer declarations vary, sim_defs.h has conditional declarations for Windows (_int64) and Digital UNIX (long). The default is GNU C (long long). If your compiler uses a different convention, you will have to modify sim_defs.h.

- The PDP-10, PDP-11, and VAX simulators share common peripherals. To distinguish the target system, one of three variables must be defined on the command line: VM_PDP10 for the PDP-10; VM_PDP11 for the PDP-11; or VM_VAX for the VAX.

- The PDP-11, and VAX simulators optionally support Ethernet. At present, Ethernet support has been tested only on Windows, Linux, OSX, NetBSD, OpenBSD, FreeBSD, Solaris, Alpha and IA64 VMS, but it should work in any host environment that supports the Pcap library (see the Ethernet readme file).

- The PDP-11 and VAX simulators support disks and sequential tape files greater than 2GB when the host OS is capable of manipulating files greater than 2GB.

- The HP2100 Fast FORTRAN Processor (FFP) and 1000-F CPU options require 64b integer support. Define HAVE_INT64 (not USE_INT64) as part of the compilation command line if your host compiler supports 64b integers. On systems without 64b support, the 1000 F-Series CPU will be unavailable, and FFP extended-precision instructions (e.g., XADD) will be disabled; the remainder of the FFP instructions will work normally. There may be some compilation warnings.

2.1 Compiling Under UNIX/Linux/Mac OS-X

The sources originate on a Windows system and have CR-LF line endings. For use on other systems, the sources may need to be converted to LF line endings. This can be done with the UNZIP utility (unzip -a).

The supplied makefile will compile the simulators for UNIX and Unix like systems. The VAX and PDP-11 can be compiled with or without Ethernet support. The makefile will automatically build these simulators with Ethernet support if the necessary network components are available on the system which is doing the building. The recommended libpcap components are those packaged and provided by the host operating system vendor. Specific details about building with network support is documented in the 0readme_ethernet.txt in the top level directory of the simh source.

To compile with or without Ethernet support:

```
gmake {target|ALL|clean}
```

Notes for hand compilation:

- The default UNIX terminal handling model is the POSIX TERMIOS interface, which is supported by Linux, Mac OS/X, and Alpha UNIX. If your UNIX only supports the BSD terminal interface, BSDTTY must be defined as part of the compilation command line.

- The PDP-8, PDP-11, 18b PDP, PDP-10, and Nova simulators use the math library. If your UNIX does not link the math library automatically, you must add -lm to the compilation command line.

Examples:

- PDP-11 under TERMIOS UNIX:

```
% cc -DVM_PDP11 pdp11_*.c scp.c sim_*.c -lm -o pdp11
```

- PDP-9 under TERMIOS UNIX:

```
% cc -DPDP9 pdp18b_*.c scp.c sim_*.c -lm -o pdp9
```

- PDP-10 under BSD terminal UNIX:

```
% cc -DVM_PDP10 -DUSE_INT64 -DBSDTTY pdp10_*.c scp.c sim_*.c -lm -o pdp10
```

2.2 Compiling Under Windows

2.2.1 Compiling with Ethernet Support

The Windows-specific Ethernet code uses the npcap or the WinPCAP 4.x package. These packages provides the libpcap functionality package that is freely available for Unix systems. Building simulators with built-in Ethernet support can be done if the required npcap/WinPcap components are available on your build system at compile time. These components are available by downloading the file:
https://github.com/simh/windows-build/archive/windows-build.zip
This zip file contains a file called README.md which explains how to locate the unpacked zip file contents to build simulators with Ethernet support. If your build environment has git, the build activity will automatically download the windows-build components the first time you build a simulator.

In order for a simulator (built with Ethernet support) to provide working Ethernet functionality at run time, npcap WinPCAP must be installed on the system. In order to install the npcap or WinPcap package the following should be performed:

- Download npcap from https://npcap.com/#download or WinPcap http://www.winpcap.org.
- Install the package as directed.

2.2.2 Compiling Under Visual C++

Visual C++ requires projects to be defined for each executable which is being built. You can define your own project for each simulator you are interested in, or you can use the predefined project definitions for this simulator release.

2.2.2.1 Using the predefined project definitions

The current simh source includes a directory “Visual Studio Projects” which contains the visual studio projects for all supported simulators. This directory contains a file named 0ReadMe_Projects.txt which explains how to locate simh source files with respect to the related required build components provide in the windows_build.zip file mentioned above in “Compiling with Ethernet Support”.

These projects produce executables in the BIN\NT\Win32-Debug or BIN\NT\Win32-Release directories. All intermediate build files are kept in BIN\NT\Project\{simulator-name} directories.

2.2.2.2 Defining your own Visual C++ project definitions

Each simulator must be organized as a separate Visual C++ project. Starting from an empty console application,

- Add all the files from the simulator file manifest to the project.
- Open the Properties (Visual Studio Express 2008) box.
- Under C/C++, Category: General, add any required preprocessor definitions (for example, USE_INT64).
- Under C/C++, Category: Preprocessor, add the top level simulation directory to the Additional Include Directories. For the VAX and PDP-10, you must also add the PDP-11 directory.
- Under Link, add wsock32.lib and winmm.lib at the end of the list of Object/Module Libraries.
- If you are building the PDP-11 and VAX with Ethernet support, you must also add the WinPCAP libraries (packet.lib, wpcap.lib) to the list of Object/Module libraries.

If you are using Visual C++ .NET, you must turn off /Wp64 (warn about potential 64b incompatibilities) and disable Unicode processing. You will also have to turn off warning 4996 ("deprecated" string functions), or lower the warning level to /W1. Otherwise, the compilations will generate a lot of spurious conversion warnings.

Alternatively, you can 'start' from a preexisting Visual Studio project which may have similar components to the project you are trying to create. Starting from such a project is done by copying the existing .vcproj file to a project file with a new name. The MOST IMPORTANT step of this process is to assure that the copy of the project file has a unique GUID. Generate a new GUID by invoking the guidgen.exe program from the Visual Studio directories (C:\Program Files (x86)\Microsoft Visual Studio 9.0\Common7\Tools\guidgen.exe). Click on the Copy button to get the newly generated GUID into the clipboard. Open the target project file in a text editor (notepad) and replace the ProjectGUID value with the clipboard contents. Replace the Name value (PDP1 for example) with the name of your new simulator. Carefully do a global replace of all instances of the prior Name value (PDP1 in this case) with the name of your new simulator. Save the project file. The next step is to add the project file to the existing simh solution. Open the simh.sln with visual studio and in the Solution Explorer pane, right click on the Solution Simh and select Add->Existing Project. Browse and select your new project file. All other changes to the project file (the particular source and include files which make up your simulator) can be added or adjusted here.

2.2.3 Compiling Under MinGW

MinGW (Minimalist GNU for Windows) is a free C compiler available from http://www.mingw.org. Msys is a minimal set of Unix utilities to support MinGW, also available from http://www.mingw.org. The distribution includes a batch file (build_mingw.bat) that will build all the simulators from source. By default, the PDP-11 and VAX family simulators are built with Ethernet support if the necessary Winpcap components are available at build time. The compiled executables will be produced in the BIN directory which will be created if needed.

2.3 Compiling Under OpenVMS

Compiling on OpenVMS requires DEC C. The simulators that require 64b (PDP-10, VAX and others) will not compile on OpenVMS/VAX. The SIMH distribution includes an MMS command file descrip.mms that will build all the simulators from source. An example of hand compilation:

- PDP-8 under VMS:

```
$ cc scp.c,sim_*.c,[.pdp8]pdp8*.c
$ link/exec=pdp8 scp.obj,sim_*.obj,[.pdp8]pdp8*.obj
```

3 Simulator Conventions

A simulator consists of a series of devices, the first of which is always the CPU. A device consists of named registers and one or more numbered units. Registers correspond to device state, units to device address spaces. Thus, the CPU device might have registers like PC, ION, etc., and a unit corresponding to main memory; a disk device might have registers like BUSY, DONE, etc., and units corresponding to individual disk drives. Except for main memory, device address spaces are simulated as unstructured binary disk files in the host file system. The `SHOW CONFIG` command displays the simulator configuration.

A simulator keeps time in terms of arbitrary units, usually one time unit per instruction executed. Simulated events (such as completion of I/O) are scheduled at some number of time units in the future. The simulator executes synchronously, invoking event processors when simulated events are scheduled to occur. Even asynchronous events, like keyboard input, are handled by polling at synchronous intervals. The `SHOW QUEUE` command displays the simulator event queue.

4 Commands

Simulator commands consist of a command verb, optional switches, and optional arguments. Switches take the form:

```
-<letter>{<letter>...}
```

Multiple switches may be specified separately or together: -abcd and -a -b -c -d are treated identically. Verbs, switches, and other input (except for file names) are case insensitive.

Any command beginning with semicolon (;) is considered a comment and ignored.

4.1 Loading and Saving Programs

The `LOAD` command (abbreviation `LO`) loads a file in binary loader format:

```
load <filename> {implementation options}
```

The types of formats supported are simulator specific. Options (such as load within range) are also simulator specific. The load command may not be meaningful or supported at all in some simulators.

The `DUMP` command (abbreviation `DU`) dumps memory in binary loader format:

```
dump <filename> {implementation options}
```

The types of formats supported are simulator specific. Options (such as dump within range) are also simulator specific. The dump command may not be meaningful or supported at all in some simulators.

4.2 Saving and Restoring State

The SAVE command (abbreviation SA) save the complete state of the simulator to a file. This includes the contents of main memory and all registers, and the I/O connections of devices:

```
save <filename>
```

The RESTORE command (abbreviation REST, alternately GET) restores a previously saved simulator state:

```
restore {-d} {-f} {-q} <filename>
```

Notes:

1) SAVE file format compresses zeroes to minimize file size.
2) The simulator can't restore active incoming telnet sessions to multiplexer devices, but the listening ports will be restored across a save/restore.
3) –d switch avoids detaching and reattaching devices during a restore
4) –f switch overrides the date timestamp check for attached files during a restore
5) –q suppresses warning messages about save file version information

4.3 Resetting Devices

The RESET command (abbreviation RE) resets a device or the entire simulator to a predefined condition. If switch -p is specified, the device is reset to its power-up state:

```
RESET                       reset all devices
RESET -p                    powerup all devices
RESET ALL                   reset all devices
RESET <device>              reset specified device
```

Typically, RESET stops any in-progress I/O operation, clears any interrupt request, and returns the device to a quiescent state. It does not clear main memory or affect I/O connections.

4.4 Connecting and Disconnecting Devices

Except for main memory and network devices, units are simulated as unstructured binary disk files in the host file system. Before using a simulated unit, the user must specify the file to be accessed by that unit. The ATTACH (abbreviation AT) command associates a unit and a file:

```
ATTACH <unit> <filename>
```

If the -n switch is specified when an attach is executed, a new file is created, and an appropriate message is printed.

If the file does not exist, and the -e switch was not specified, a new file is created, and an appropriate message is printed. If the -e switch was specified, a new file is not created, and an error message is printed. If the –n switch was specified, a new file is created or the existing file is truncated to zero length.

If the -r switch is specified, or the file is write protected, ATTACH tries to open the file read only. If the file does not exist, or the unit does not support read only operation, an error occurs. Input-only devices, such as paper-tape readers, and devices with write lock switches, such as disks and tapes, support read only operation; other devices do not. If a file is attached read only, its contents can be examined but not modified.

If the –a switch is specified, is a sequential output only device (like a line printer, paper tape punch, etc.), the file being attached will be opened in append mode thus adding to any existing file data beyond what may have already been there.

For simulated magnetic tapes, the ATTACH command can specify the format of the attached tape image file:

```
ATTACH -f <tape_unit> <format> <filename>
```

The currently supported tape image file formats are:

```
SIMH                        SIMH simulator format
E11                         E11 simulator format
TPC                         TPC format
P7B                         Pierce simulator 7-track format
```

The tape format can also be set with the SET command prior to ATTACH:

```
SET <tape_unit> FORMAT=<format>
ATT <tape_unit> <filename>
```

The format of an attached file can be displayed with the SHOW command:

```
SHOW <tape_unit> FORMAT
```

For Telnet-based terminal emulators, the ATTACH command associates the master unit with a TCP/IP port:

```
ATTACH <unit> <port>
```

The port is a decimal number between 1 and 65535 that is not used by standard TCP/IP protocols.

For Ethernet emulators, the ATTACH command associates the simulated Ethernet with a physical Ethernet device:

```
ATTACH <unit> <physical device name>
```

The DETACH (abbreviation DET) command breaks the association between a unit and a file, port, or network device:

```
DETACH ALL                  detach all units
DETACH <unit>               detach specified unit
```

The EXIT command performs an automatic DETACH ALL.

4.5 Examining and Changing State

There are four commands to examine and change state:

- EXAMINE (abbreviated E) examines state
- DEPOSIT (abbreviated D) changes state
- IEXAMINE (interactive examine, abbreviated IE) examines state and allows the user to interactively change it
- IDEPOSIT (interactive deposit, abbreviated ID) allows the user to interactively change state

All four commands take the form

```
command {modifiers} <object list>
```

Deposit must also include a deposit value at the end of the command.

There are four kinds of modifiers: switches, device/unit name, search specifier, and for EXAMINE, output file. Switches have been described previously. A device/unit name identifies the device and unit whose address space is to be examined or modified. If no device is specified, the CPU (main memory)is selected; if a device but no unit is specified, unit 0 of the device is selected.

The search specifier provides criteria for testing addresses or registers to see if they should be processed. A specifier consists of a logical operator, a relational operator, or both, optionally separated by spaces.

```
{<logical op> <value>} <relational op> <value>
```

where the logical operator is & (and), | (or), or ^ (exclusive or), and the relational operator is = or == (equal), ! or != (not equal), >= (greater than or equal), > (greater than), <= (less than or equal), or < (less than). If a logical operator is specified without a relational operator, it is ignored. If a relational operator is specified without a logical operator, no logical operation is performed. All comparisons are unsigned.

The output file modifier redirects command output to a file instead of the console. An output file modifier consists of @ followed by a valid file name.

Modifiers may be specified in any order. If multiple modifiers of the same type are specified, later modifiers override earlier modifiers. Note that if the device/unit name comes after the search specifier, the search values will interpreted in the radix of the CPU, rather than of the device/unit.

The "object list" consists of one or more of the following, separated by commas:

```
register                    the specified register
register[sub1-sub2]         the specified register array locations,
                            starting at location sub1 up to and
                            including location sub2
register[sub1/length]       the specified register array locations,
                            starting at location sub1 up to but
                            not including sub1+length
register[ALL]               all locations in the specified register
                            array
register1-register2         all the registers starting at register1
                            up to and including register2
address                     the specified location
address1-address2           all locations starting at address1 up to
                            and including address2
address/length              all location starting at address up to
                            but not including address+length
STATE                       all registers in the device
ALL                         all locations in the unit
$                           use the most recently referenced value as
                            an address to reference (indirect)
.                           use the most recent referenced address
                            again.
```

Switches can be used to control the format of display information:

```
-o or -8                    display as octal
-d or -10                   display as decimal
-h or -16                   display as hexadecimal
-2                          display as binary
```

Simulators typically provide these additional switches for address locations:

```
-a                              display as ASCII
-c                              display as character string
-m                              display as instruction mnemonic
```

...and accept symbolic input (see documentation with each simulator).

Examples:

```
ex 1000-1100                    examine 1000 to 1100
ex .                            examine most recent address again
de PC 1040                      set PC to 1040
ie 40-50                        interactively examine 40:50
ie >1000 40-50                  interactively examine the subset
                                of locations 40:50 that are >1000
ex rx0 50060                    examine 50060, RX unit 0
ex rx sbuf[3-6]                 examine SBUF[3] to SBUF[6] in RX
ex r4,$                         examine R4 and where it points
de all 0                        set main memory to 0
de &77>0 0                      set all addresses whose low order
                                bits are non-zero to 0
ex -m @memdump.txt 0-7777       dump memory to file
```

Note: to terminate an interactive command, simply type a bad value (eg, XYZ) when input is requested.

4.6 Evaluating Instructions

The `EVAL` command evaluates a symbolic instruction and returns the equivalent numeric value. This is useful for obtaining numeric arguments for a search command:

```
EVAL <expression>
```

Examples:

On the VAX simulator:

```
sim> eval addl2 r2,r3
       0:      005352C0
       sim> eval addl2 #ff,6(r0)
       0:      00FF8FC0
       4:      06A00000
       sim> eval 'AB
       0:      00004241
```

On the PDP-8:

```
sim> eval tad 60
       0:      1060
       sim> eval tad 300
       tad 300
       Can't be parsed as an instruction or data
```

'tad 300' fails, because with an implicit PC of 0, location 300 can't be reached with direct addressing.

4.7 Running a Simulated Program

The RUN command (abbreviated RU) resets all devices, deposits its argument (if given) in the PC, and starts execution. If no argument is given, execution starts at the current PC.

The GO command does *not* reset devices, deposits its argument (if given) in the PC, and starts execution. If no argument is given, execution starts at the current PC.

The CONTINUE command (abbreviated CO) does *not* reset devices and resumes execution at the current PC.

The STEP command (abbreviated S) resumes execution at the current PC for the number of instructions given by its argument. If no argument is supplied, one instruction is executed.

The BOOT command (abbreviated B) resets all devices and bootstraps the device and unit given by its argument. If no unit is supplied, unit 0 is bootstrapped. The specified unit must be attached.

As it initializes all of the I/O devices, the RUN command is almost never the proper command to use after a program has been started. The GO or CONTINUE commands are generally equivalent in hardware to running the CPU and is the usual way of resuming execution after a programmed halt. If an I/O reset is required before resuming execution, the RESET and GO commands are recommended instead of RUN. If RUN is entered a second time without an explicit RESET preceding it, a warning is printed on the simulation console:

```
Resetting all devices...  This may not have been your intention.
The GO and CONTINUE commands do not reset devices.
```

...before execution is resumed. The warning may be suppressed by adding the -Q switch to the RUN command.

4.7.1 Controlling the Simulation Rate

By default, the simulator runs as fast as possible (although at lower than normal priority) and will consume all available processing resources on the host system. This will raise power consumption (and the operating temperature) of many PC's and drain the battery of a laptop.

The SET THROTTLE command allows the user to reduce the effective execution rate to a specified number of instructions per second, or to a specified percentage of total host computing time:

```
SET THROTTLE xM                  set execution rate to x mips
SET THROTTLE xK                  set execution rate to x kips
SET THROTTLE x%                  limit simulator to x% of host time
SET THROTTLE insts/delay         execute 'insts' instructions and
                                 then sleep for 'delay' milliseconds
```

Throttling is only available on host systems that implement a precision real-time delay function.

xM, xK and x% modes require the simulator to execute sufficient instructions to actually calibrate the desired execution rate relative to wall clock time. Very short running programs may complete before calibration completes and therefore before the simulated execution rate can match the desired rate.

The SET NOTHROTTLE command turns off throttling. The SHOW THROTTLE command shows the current settings for throttling.

Some simulators implement a different form of resource management called idling. Idling suspends simulated execution whenever the program running on the simulator is doing nothing, and runs the simulator at full speed when there is work to do. Throttling and idling are mutually exclusive.

4.8 Stopping the Simulator

Programs run until the simulator detects an error or stop condition, or until the user forces a stop condition.

4.8.1 Simulator Detected Stop Conditions

These simulator-detected conditions stop simulation:

- HALT instruction. If a HALT instruction is decoded, simulation stops.
- Breakpoint. The simulator may support breakpoints (see below).
- I/O error. If an I/O error occurs during simulation of an I/O operation, and the device stop-on-I/O-error flag is set, simulation usually stops.
- Processor condition. Certain processor conditions can stop simulation; these are described with the individual simulators.

4.8.2 User Specified Stop Conditions

Typing the interrupt character stops simulation. The interrupt character is defined by the `WRU` (where are you) console option and is initially set to 005 (^E).

4.8.3 Breakpoints

A simulator may offer breakpoint capability. A simulator may define breakpoints of different types, identified by letter (for example, E for execution, R for read, W for write, etc.). At the moment, most simulators support only E (execution) breakpoints.

Associated with a breakpoint are a count and, optionally, one or more actions. Each time the breakpoint is taken, the associated count is decremented. If the count is less than or equal to 0, the breakpoint occurs; otherwise, it is deferred. When the breakpoint occurs, the optional actions are automatically executed.

A breakpoint is set by the `BREAK` command:

```
BREAK {-types} {<addr range>{[count]},{addr range...}}{;action;action...}
```

If no type is specified, the simulator-specific default breakpoint type (usually E for execution) is used. If no address range is specified, the current PC is used. As with `EXAMINE` and `DEPOSIT`, an address range may be a single address, a range of addresses low-high, or a relative range of address/length. Examples:

```
BREAK                       set E break at current PC
BREAK -e 200                set E break at 200
BREAK 2000/2[2]             set E breaks at 2000,2001 with count = 2
BREAK 100;EX AC;D MQ 0      set E break at 100 with actions EX AC and
                             D MQ 0
BREAK 100;                  delete action on break at 100
```

Currently set breakpoints can be displayed with the `SHOW BREAK` command:

```
SHOW {-types} BREAK {ALL|<addr range>{,<addr range>...}}
```

Locations with breakpoints of the specified type are displayed.

Finally, breakpoints can be cleared by the `NOBREAK` command.

If an action command must contain a semicolon, that action command should be enclosed in quotes so that the entire action command can be distinguished from the action separator.

4.8.4 Execution Time Limits

A simulator user may want to limit the maximum execution time that a simulator may run for. This might be appropriate to limit a runaway diagnostic which didn't achieve explicit success or failure within some user specified time. The RUNLIMIT command provides ways to limit execution.

```
RUNLIMIT n {CYCLES|MICROSECONDS|SECONDS|MINUTES|HOURS}
NORUNLIMIT
```

Equivalently:

```
SET RUNLIMIT n {CYCLES|MICROSECONDS|SECONDS|MINUTES|HOURS}
SET NORUNLIMIT
```

The run limit state can be examined with:

```
SHOW RUNLIMIT
```

If the units of the run limit are not specified, the default units are cycles. Once an execution run limit has been reached, any subsequent GO, RUN, CONTINUE, STEP or BOOT commands will cause the simulator to exit. A previously defined RUNLIMIT can be cleared with the NORUNLIMIT or the establishment of a new limit.

4.9 Halt on Output Data

In addition to breakpoints (halting the simulator on specific addresses), the simulator can halt when a specific string has been output to the console. The EXPECT command provides a way to define a rule which will stop execution and take actions when specific output has been generated by the simulated system.

```
EXPECT {dev:line} {[count]} {HALTAFTER=n,}"<string>" {actioncommand {; actioncommand} …}
NOEXPECT {dev:line}
SHOW EXPECT {dev:line}
```

<dev:line> specifies a particular device a line in that simulated device. If it is not specified, the simulated system's console device is the default device.

If a [count] is specified, the rule will match after the match string has matched count times.

The string argument must be delimited by quote characters. Quotes may be either single or double but the opening and closing quote characters must match. Data in the string may contain escaped character strings.
When expect rules are defined, they are evaluated against recently produced output as each character is output to the device. Since this evaluation processing is done on each output character, rule matching is not specifically line

oriented. If line oriented matching is desired, then rules should be defined which contain the simulated system's line ending character sequence (i.e. "\r\n").
Once data has matched any expect rule, that data is no longer eligible to match other expect rules which may already be defined. Data which is output prior to the definition of an expect rule is not eligible to be matched against.

The NOEXPECT command removes a previously defined EXPECT command for the console or a specific multiplexer line.

The SHOW EXPECT command displays all of the pending EXPECT state for the console or a specific multiplexer line.

If an action command must contain a semicolon, that action command should be enclosed in quotes so that the entire action command can be distinguished from the action separator.

4.9.1 Switches

Switches can be used to influence the behavior of EXPECT rules

4.9.1.1 –p

Expect rules default to be one shot activities. That is, a rule is automatically removed when a match occurs unless the rule is designated as a persistent rule by using a –p switch when the rule is defined.

4.9.1.2 –c

If an expect rule is defined with the –c switch, it will cause all pending expect rules on the current device to be cleared when the rule matches data in the device output stream.

4.9.1.3 –r

If an expect rule is defined with the –r switch, the string is interpreted as a regular expression applied to the output data stream. This regular expression may contain parentheses delimited sub-groups.

The syntax of the regular expressions available are those supported by the Perl Compatible Regular Expression package (aka PCRE). As the name implies, the syntax is generally the same as Perl regular expressions. See http://perldoc.perl.org/perlre.html for more details.

If the PCRE package isn't available in the environment when a simulator is built, the local system's regular expression package (if available) is used and the regular expression syntax is limited to what may be provided there.

4.9.1.4 –i

If a regular expression expect rule is defined with the –i switch, character matching for that expression will be case independent. The –i switch is only valid for regular expression expect rules (-r).

4.9.1.5 Escaping String Data

The following character escapes are explicitly supported when expect rules are defined NOT using regular expression match patterns:

\r	Expect the ASCII Carriage Return character (Decimal value 13)
\n	Expect the ASCII Linefeed character (Decimal value 10)
\f	Expect the ASCII Formfeed character (Decimal value 12)
\t	Expect the ASCII Horizontal Tab character (Decimal value 9)
\v	Expect the ASCII Vertical Tab character (Decimal value 11)
\b	Expect the ASCII Backspace character (Decimal value 8)
\\	Expect the ASCII Backslash character (Decimal value 92)
\'	Expect the ASCII Single Quote character (Decimal value 39)
\"	Expect the ASCII Double Quote character (Decimal value 34)
\?	Expect the ASCII Question Mark character (Decimal value 63)
\e	Expect the ASCII Escape character (Decimal value 27)

as well as octal character values of the form:

\n{n{n}} where each n is an octal digit (0-7)

and hext character values of the form:

\xh{h} where each h is a hex digit (0-9A-Fa-f)

4.9.2 Determining which output matched

When an expect rule matches data in the output stream, the rule which matched is recorded in the environment variable _EXPECT_MATCH_PATTERN. If the expect rule was a regular expression rule, then the environment variable _EXPECT_MATCH_GROUP_0 is set to the whole string which matched and if the match pattern had any parentheses delimited sub-groups, the environment variables _EXPECT_MATCH_GROUP_1 thru _EXPECT_MATCH_GROUP_n are set to the values within the string which matched the respective sub-groups.

4.10 Injecting Input Data

The SEND command provides a way to insert input data in to the console device of a simulated system as if it were entered by a user:

```
SEND {AFTER=n,} {DELAY=n,}"<string>"

NOSEND

SHOW SEND
```

The string argument must be delimited by quote characters. Quotes may be either single or double but the opening and closing quote characters must match. Data in the string may contain escaped character strings.

The SEND command can also insert input into any serial device on a simulated system as if it was entered by a user.

SEND <dev>:line {AFTER=n,}{DELAY=n,}"<string>"

NOSEND <dev>:line

SHOW SEND <dev>:line

The NOSEND command removes any undelivered input data which may be pending on the CONSOLE or a specific multiplexer line.

The SHOW SEND command displays any pending SEND activity for the CONSOLE or a specific multiplexoer line.

4.10.1 Delay

Specifies a positive integer representing a minimal instruction delay between characters being sent. The value specified in a delay argument persists across SEND commands to the same device (console or serial device). The delay parameter can be set by itself with:

SEND {<dev>:line} {DELAY=n}

The default value of the delay parameter is 1000.

4.10.2 After

Specifies a positive integer representing a minimul number of instructions which must execute before the first character in the string is sent. The value specified as the after parameter persists across SEND commands to the same device (console or serial device). The after parameter value can be set by itself with:

SEND {<dev>:line} {AFTER=n}

If the after parameter isn't explicitly set, it defaults to the value of the delay parameter.

4.10.3 Escaping String Data

The following character escapes are explicitly supported:

\r	Sends the ASCII Carriage Return character (Decimal value 13)
\n	Sends the ASCII Linefeed character (Decimal value 10)
\f	Sends the ASCII Formfeed character (Decimal value 12)
\t	Sends the ASCII Horizontal Tab character (Decimal value 9)
\v	Sends the ASCII Vertical Tab character (Decimal value 11)
\b	Sends the ASCII Backspace character (Decimal value 8)
\\	Sends the ASCII Backslash character (Decimal value 92)
\'	Sends the ASCII Single Quote character (Decimal value 39)
\"	Sends the ASCII Double Quote character (Decimal value 34)
\?	Sends the ASCII Question Mark character (Decimal value 63)
\e	Sends the ASCII Escape character (Decimal value 27)

as well as octal character values of the form:

\n{n{n}} where each n is an octal digit (0-7)

and hext character values of the form:

\xh{h} where each h is a hex digit (0-9A-Fa-f)

4.11 Setting Device Parameters

The SET command (abbreviated SE) changes the status of one or more device parameters:

```
SET <device> <parameter>{=<value},{<parameter>{=<value>},...}
```

or one or more unit parameters:

```
SET <unit> <parameter>{=<value>},{<parameter>{=<value>},...}
```

Most parameters are simulator and device specific. Disk drives, for example, can usually be set WRITEENABLED or write LOCKED; if a device supports multiple drive types, the SET command can be used to specify the drive type.

All devices recognize the following parameters:

```
OCT                          sets the data radix = 8
DEC                          sets the data radix = 10
HEX                          sets the data radix = 16
```

4.12 Displaying Parameters and Status

The SHOW command (abbreviated SH) displays the status of one or more device parameters:

```
SHOW {<modifiers} <device> <parameter>{=<value>},
      {<parameter>{=<value>},...}
```

or one or more unit parameters:

```
SHOW {<modifiers} <unit> <parameter>{=<value>},
      {<parameter>{=<value>},...}
```

There are two kinds of modifiers: switches and output file. Switches have been described previously. The output file modifier redirects command output to a file instead of the console. An output file modifier consists of @ followed by a valid file name.

All devices implement parameters RADIX (the display radix), MODIFIERS (list of valid modifiers), and NAMES (logical name). Other device and unit parameters are implementation-specific.

SHOW is also used to display global simulation state:

```
SHOW CONFIGURATION           shows the simulator configuration and
                             the status of all devices and units
SHOW DEVICES                 shows the simulator configuration
SHOW FEATURES                shows the simulator configuration with
                             descriptions
```

```
SHOW MODIFIERS                shows all available modifiers
SHOW NAMES                    show all logical names
SHOW QUEUE                    shows the simulator event queue
SHOW TIME                     shows the elapsed time since the last RUN
SHOW VERSION                  show the simulator version and options
SHOW <device>                 shows the status of the named device
SHOW <unit>                   shows the status of the named unit
SHOW THROTTLE                 shows the current throttling mode
SHOW SHOW                     shows the show options for all devices
SHOW ETHERNET                 shows the status/availability of host
                              Ethernet devices
SHOW SERIAL                   shows the status of host serial ports
SHOW MULTIPLEXER              shows the status of all multiplexer devices
                              which have been attached
SHOW DEFAULT                  shows the current working directory
SHOW DEBUG                    shows the debug state of the simulator
SHOW LOG                      shows the logging state of the simulator
```

The SHOW DEVICES, SHOW CONFIGURATION, SHOW FEATURES will normally display all devices which exist in the simulator. These commands can optionally only display the enabled devices when the –E switch is on the command line (i.e. SHOW –E DEVICES).

SHOW QUEUE and SHOW TIME display time in simulator-specific units; typically, one time unit represents one instruction execution.

4.13 Altering the Simulated Configuration

In most simulators, the SET <device> DISABLED command removes the specified device from the configuration. A DISABLED device is invisible to running programs. The device can still be RESET, but it cannot be ATTACHed, DETACHed, or BOOTed. SET <device> ENABLED restores a disabled device to a configuration.

Most multi-unit devices allow units to be enabled or disabled:

```
SET <unit> ENABLED
SET <unit> DISABLED
```

When a unit is disabled, it will not be displayed by SHOW DEVICE.

The standard device names can be supplemented with logical names. Logical names must be unique within a simulator (that is, they cannot be the same as an existing device name). To assign a logical name to a device:

```
ASSIGN <device> <log-name>    assign log-name to device
```

To remove a logical name:

```
DEASSIGN <device>             remove logical name
```

To show the current logical name assignment:

```
SHOW <device> NAMES           show logical name, if any
```

To show all logical names:

```
SHOW NAMES
```

4.14 Console Options

Console options are controlled by the `SET CONSOLE` command.

The console terminal normally runs in the controlling window. Optionally, the console terminal can be connected to a Telnet port. This allows systems to emulate a VT100 using the built-in terminal emulation of the Telnet client.

```
SET CONSOLE TELNET=<port>     connect console terminal to Telnet session
                              on port
SET CONSOLE NOTELNET          disable console Telnet
```

Connections to the specified port, by default, will be unrestricted. Connections from particular IPv4 or IPv6 addresses can be restricted or allowed based on rules you can add to the "port" specifier on the attach command. You can add as many rules as you need to the attach command specified with ";ACCEPT=rule-detail" or ";REJECT=rule-detail" where rule-detail can be an IP address, hostname or network block in CIDR form. Rules are interpreted in order and if, while processing the list, the end is reached the connection will be rejected.

Normally a console terminal configured to listen to on a telnet port requires that a telnet connection be active for the simulator to run. A telnet console can have its contents written to a buffer and which will allow the simulator to run without an active telnet connection. When a telnet connection is established, the buffer contents is presented to the telnet session and execution continues as if the connection had been there all along. Console buffering can be enabled by the following command:

```
SET CONSOLE TELNET=BUFFERED{=bufsiz}
                              enable console buffering and optionally
                              set the buffer size to 'bufsiz'.  The
                              default buffer size is 32768.
SET CONSOLE TELNET=NOBUFFER   disable console buffering
```

Output to the console telnet session can be logged simultaneously to a file:

```
SET CONSOLE TELNET=LOG=<filename>
                              log console port output to file
SET CONSOLE TELNET=NOLOG
                              disable logging
```

The console provides a limited key remapping capability:

```
SET CONSOLE WRU=<value>       interpret ASCII code value as WRU
SET CONSOLE BRK=<value>       interpret ASCII code value as BREAK
                              (0 disables)
SET CONSOLE DEL=<value>       interpret ASCII code value as DELETE
SET CONSOLE PCHAR=<value>     bit mask of printable characters in range
                              [31,0]
```

A simulator console can be connected to a serial port on the host system.

```
SET CONSOLE SERIAL=ser0       connect console to serial port ser0
```

```
      SET CONSOLE SERIAL=COM1       connect console to serial port COM1
      SET CONSOLE SERIAL=/dev/ttyS0 connect console to serial port /dev/ttyS0
```

```
The available Serial ports on the host system can be displayed with the command:
```

```
      SHOW SERIAL                   display available serial ports on host
```

Serial port speed, character size, parity and stop bits can be indicated on the by appending the speed, character size, parity and stop bits to the serial port name:

```
      SET CONSOLE SERIAL=ser0;2400-8N1
```

```
This will connect at 2400 with 8 bit characters, no parity and 1 stop bit.  The
default serial speed, character size, parity and stop bits is 9600-8N1.
```

Values are hexadecimal on hex CPU's, octal on all others.

The `SHOW CONSOLE` command displays the current state of console options:

```
      SHOW CONSOLE                  show all console options
      SHOW CONSOLE TELNET           show console Telnet state
      SHOW CONSOLE WRU              show value assigned to WRU
      SHOW CONSOLE BRK              show value assigned to BREAK
      SHOW CONSOLE DEL              show value assigned to DELETE
      SHOW CONSOLE PCHAR            show value assigned to PCHAR
```

Both `SET CONSOLE` and `SHOW CONSOLE` accept multiple parameters, separated by commas, e.g.,

```
      SET CONSOLE WRU=5,DEL=177     set code values for WRU and DEL
```

4.15 Remote Console

During simulator execution, it may occasionally be useful to enter commands to query or adjust some internal details of the simulator's configuration or operation. These activities could be achieved directly through the initiating session which started the simulator. However, a simulator may be running in the context of a background process and not actually have an interactive interface to the initiating process, or it may be inconvenient to access the initiating process. To support circumstances like this, a Remote Console facility can be enabled in the simulator. The remote console capability is configured with the following commands:

```
      SET REMOTE TELNET=<port>      enable remote console to connects via Telnet
                                    to port
      SET REMOTE BUFFERSIZE=bufsize specify remote console command output buffer
                                    size
      SET REMOTE NOTELNET           disable remote console Telnet
      SET REMOTE CONNECTIONS=n      specify the number of concurrent remote
                                    console sessions available
      SET REMOTE TIMEOUT=secs       specify the remote console idle command
                                    timeout
```

Connections to the specified port, by default, will be unrestricted. Connections from particular IPv4 or IPv6 addresses can be restricted or allowed based on rules you can add to the "port" specifier on the attach command. You can add as many rules as you need to the attach command specified with ";ACCEPT=rule-detail" or ";REJECT=rule-detail" where rule-detail can be an IP address, hostname or network block in CIDR form. Rules are interpreted in order and if, while processing the list, the end is reached the connection will be rejected.

The remote console configuration details can be viewed with the command:

```
SHOW REMOTE                    display remote console configuration
```

The remote console facility has two modes of command input and execution:

Single Command Mode: In single command mode you enter one command at a time and aren't concerned about what the simulated system is doing while you enter that command. The command is executed once you've hit return.

Multiple Command Mode: In multiple command mode you initiate your activities by entering the WRU character (usually ^E). This will suspend the current simulator execution. You then enter commands as needed and when you are done you enter a CONTINUE command. While entering Multiple Command commands, if you fail to enter a complete command before the timeout (specified by "SET REMOTE TIMEOUT=seconds"), a CONTINUE command is automatically processed and simulation proceeds.

A subset of normal simh commands are available for use in remote console sessions.

The Single Command Mode commands are: ATTACH, DETACH, PWD, SHOW, DIR, LS, ECHO, HELP

The Multiple Command Mode commands are: EXAMINE, IEXAMINE, DEPOSIT, EVALUATE, ATTACH, DETACH, ASSIGN, DEASSIGN, STEP, CONTINUE, PWD, SAVE, SET, SHOW, DIR, LS, ECHO, HELP

A remote console session will close when a EOF character is entered (i.e. ^D or ^Z).

4.15.1 Master Mode

Remote Console Master mode allows full control of simulator operation to performed over a remote TCP session. This mode is potentially useful for applications like a simulated CPU front panel interface or remote debugging with a GDB “Serial Stub”.
Master mode is entered from a simulator command file with the following command:

```
SET REMOTE MASTER       enter master mode
```

```
Operation in Remote Console Master mode requires that the simulator’s console
port be serviced via a telnet connect with a “SET CONSOLE TELNET=port” command.
```

```
Both a telnet connection to the console port AND to the Remote Console port is
required be a simulator will enter Master Mode.

A Master Mode session can return control to the initiating simulator command
with:

      SET REMOTE NOMASTER           leave master mode
```

The Remote Console Master Mode commands are: EXAMINE, IEXAMINE, DEPOSIT, EVALUATE, ATTACH, DETACH, ASSIGN, DEASSIGN, STEP, CONTINUE, PWD, SAVE, CD, SET, SHOW, DIR, LS, ECHO, HELP, RUN, GO, BOOT, BREAK, NOBREAK, EXIT

4.16 Executing Command Files

The simulator can execute command files with the `DO` command:

```
      DO <filename> {arguments...}  execute commands in file
```

The `DO` command allows command files to contain substitutable arguments. The string %n, where n is between 1 and 9, is replaced with argument n from the `DO` command line. The string %0 is replaced with <filename>. The sequences \% and \\ are replaced with the literal characters % and \, respectively. Arguments with spaces can be enclosed in matching single or double quotation marks.

If the switch -v is specified, the commands in the file are echoed before they are executed.

If the switch -e is specified, command processing (including nested command invocations) will be aborted if a command error is encountered. (Simulation stop never abort processing; use ASSERT to catch unexpected stops.) Without the switch, all errors except ASSERT failures will be ignored, and command processing will continue.

DO commands may be nested up to ten invocations deep.

Several commands are particularly useful within command files. While they may be executed interactively, they have only limited functionality when so used.

4.16.1 Default Command File executed on Simulator Startup

When a simulator starts execution, the following sequence of simh command files are executed if they are found:
1_ If a file named **simh.ini** is located in your HOME directory, it is executed.
2. If the simh.ini file in your HOME directory isn't found, a file named **simh.ini** in your current working directory is executed if it exists.
3. If the simulator is invoked with any arguments, then the arguments are presumed to be a command file and possible arguments to that command file which is executed.
4. If the simulator is invoked without any arguments, then a command file with the same name as the simulator binary with **.ini** appended that is located in the current working directory is executed.

Note, that up to 2 separate command files may be executed on simulator startup. The simh.ini file allows the user to define local user preferences that align with their personal goals for simulator execution across all simulators that may be used on their system.

4.16.2 Change Command Execution Flow

Commands in a command file execute in sequence until either an error trap occurs (when a command completes with an error status), or when an explicit request is made to start command execution elsewhere with the GOTO command:

```
GOTO <label>
```

Labels are lines in a command file which the first non whitespace character is a ":". The target of a GOTO is the first matching label in the current do command file which is encountered. Since labels don't do anything else besides being the targets of goto's, they could also be used to provide comments in do command files.

4.16.3 Subroutine Calls

Control can be transferred to a labeled subroutine with:

```
CALL <label>
EXIT

:label
ECHO subroutine called
RETURN
```

4.16.4 Pausing Command Execution

A simulator command file may wait for a specific period of time with:

```
SLEEP NUMBER[SUFFIX]...
```

Pause for NUMBER seconds. SUFFIX may be 's' for seconds (the default), 'm' for minutes, 'h' for hours or 'd' for days. NUMBER may be an arbitrary floating point number. Given two or more arguments, pause for the amount of time specified by the sum of their values. Note: A SLEEP command is interruptible with SIGINT (^C).

4.16.5 Displaying Arbitrary Text

The `ECHO` and ECHOF commands are useful ways of annotating command files.

4.16.5.1 ECHO command

ECHO prints out its argument on the console (and log) followed by a newline.

```
ECHO <string>                 output string to console
```

If there is no argument, ECHO prints a blank line on the console. This may be used to provide spacing in the console display or log.

4.16.5.2 ECHOF command

ECHOF prints out its argument on the console (and log) followed by a newline.

```
ECHOF {-n} "string"|<string>              output to console
ECHOF {-n} dev:line "string"|<string>     output to specified line
```

If there is no argument, ECHOF prints a blank line. The –n switch suppresses the output of a newline character. If the string to be output is surrounded by quotes, the string within the quotes is interpreted as described in 4.10.3 prior to being output (without the surrounding quotes).

A command alias can be used to replace the ECHO command with the ECHOF command as described in 4.16.9

4.16.6 Testing Simulator State

The `ASSERT` command tests a simulator state condition expression and halts command file execution if the condition is false:

```
ASSERT <condition-expression>
```

If the result of the condition is false, an "Assertion failed" message is printed, and any running command file is aborted. Otherwise, the command has no effect.

The `IF` command tests a simulator state condition expression and executes one or more commands if the condition is true:

```
IF <condition-expression> <action>{; <action>…}
{ELSE <action>{; <action>…}}
```

If the result of the condition is true, the action command(s) are executed. Otherwise, the command has no effect. An optional ELSE command immediately following an IF command will have its command arguments executed if the IF condition wasn't satisfied.

If an action command must contain a semicolon, that action command should be enclosed in quotes so that the entire action command can be distinguished from the action separator.

4.16.6.1 Condition Expressions

4.16.6.1.1 Internal Simulator Variables Expressions

```
{NOT} {<dev>} <reg>{<logical-op><value>}<conditional-op><value>
```

If <dev> is not specified, CPU is assumed. <reg> is a register (scalar or subscripted) belonging to the indicated device. The <conditional-op> and optional <logical-op> are the same as those used for "search specifiers" by the `EXAMINE` and `DEPOSIT` commands (see above). The <value>s are expressed in the radix specified for <reg>, not in the radix for the device.

If the <logical-op> and <value> are specified, the target register value is first altered as indicated. The result is then compared to the <value> via the <conditional-op>. If the NOT unary operator precedes the expression, the results value is inverted.

4.16.6.1.2 C Style Expressions

Comparisons can optionally be done with complete C style computational expressions which leverage the C operations in the below table and can optionally reference any combination of values that are constants or contained in environment variables or simulator registers. C style expression evaluation is initiated by enclosing the expression in parenthesis.

(	Open Parenthesis
)	Close Parenthesis
-	Subtraction
+	Addition
*	Multiplication
/	Division
%	Modulus
&&	Logical AND
||	Logical OR
&	Bitwise AND
|	Bitwise Inclusive OR
^	Bitwise Exclusive OR
>>	Bitwise Right Shift
<<	Bitwise Left Shift
==	Equality
!=	Inequality
<=	Less than or Equal
<	Less than
>=	Greater than or Equal
>	Greater than
!	Logical Negation
~	Bitwise Compliment

4.16.6.1.3 String Comparison Expressions

String Values can be compared with:

```
{-i} {NOT} "<string1>"|EnvVarName1 <compare-op> "<string2>"|EnvVarName2
```

The –i switch, if present, causes comparisons to be case insensitive. The -w switch, if present, causes comparisons to allow arbitrary runs of whitespace to be equivalent to a single space. The -i and -w switches may be combined. <string1> and <string2> are quoted string values which may have environment variables substituted as desired. Either string may be an environment variable name whose expanded value will be used in place of the explicitly quoted string. <compare-op> may be one of:

== - equal
EQU - equal
!= - not equal
NEQ - not equal
< - less than
LSS - less than
<= - less than or equal
LEQ - less than or equal
> - greater than
GTR - greater than
>= - greater than or equal
GEQ - greater than or equal

Comparisons are generic. This means that if both string1 and string2 are comprised of all numeric digits, then the strings are converted to numbers and a numeric comparison is performed. For example: “+1” EQU “1” will be true.

4.16.6.1.4 File Existence Test

File existence can be determined with:

{NOT} EXIST "<filespec>"

{NOT} EXIST <filespec>

Specifies a true (false {NOT}) condition if the file exists.

4.16.6.1.5 File Comparison Test

Files can have their contents compared with:

-F{W} {NOT} "<filespec1>" == "<filespec2>"

Specifies a true (false {NOT}) condition if the indicated files have the same contents. If the -W switch is present, allows arbitrary runs of whitespace to be considered a single space during file content comparison.

4.16.6.2 Example Test of Simulator State

A command file might be used to bootstrap an operating system that halts after the initial load from disk. The `ASSERT` command is then used to confirm that the load completed successfully by examining the CPU's "A" register for the expected value:

```
; OS bootstrap command file
;
ATTACH DS0 os.disk
BOOT DS
; A register contains error code; 0 = good boot
ASSERT A=0
ATTACH MT0 sys.tape
ATTACH MT1 user.tape
```

```
RUN
```

In the example, if the A register is not 0, the "ASSERT A=0" command will be echoed, the command file will be aborted with an "Assertion failed" message. Otherwise, the command file will continue to bring up the operating system.

Alternative, the IF command could be used to solve the same problem:

```
; OS bootstrap command file
;
ATTACH DS0 os.disk
BOOT DS
; A register contains error code; 0 = good boot
IF NOT A=0 ECHO Assertion Failed; EXIT AFAIL
ATTACH MT0 sys.tape
ATTACH MT1 user.tape
RUN
```

Alternative, the IF command could have an arbitrarily complex C expression syntax to solve the same problem:

```
; OS bootstrap command file
;
ATTACH DS0 os.disk
BOOT DS
; A register contains error code; 0 = good boot
IF NOT (((A*256)>>7)==0) ECHO Assertion Failed; EXIT AFAIL
ATTACH MT0 sys.tape
ATTACH MT1 user.tape
RUN
```

4.16.7 Trapping on command completion conditions

Error traps can be taken when any command returns a non success status. Actions to be performed for particular status returns are specified with the ON command.

4.16.7.1 Enabling Error Traps

Error trapping is enabled with:

```
set on                  enable error traps
```

Error trapping is initially disabled.

4.16.7.2 Disabling Error Traps

Error trapping is disabled with:

```
set noon                disable error traps
```

4.16.7.3 ON Command

To set the action(s) to take when a specific error status is returned by a command in the currently running do command file:

```
on <statusvalue> commandtoprocess{; additionalcommandtoprocess}
```

To clear the action(s) taken take when a specific error status is returned:

```
on <statusvalue>
```

To set the default action(s) to take when any otherwise unspecified error status is returned by a command in the currently running do command file:

```
on error commandtoprocess{; additionalcommandtoprocess}
```

To clear the default action(s) taken when any otherwise unspecified error status is returned:

```
on error
```

4.16.7.3.1 Parameters

Error traps can be taken for any command which returns a status other than `SCPE_STEP`, `SCPE_OK`, and `SCPE_EXIT`.

`ON` Traps can specify any of these status values:

```
NXM, UNATT, IOERR, CSUM, FMT, NOATT, OPENERR, MEM, ARG,
STEP, UNK, RO, INCOMP, STOP, TTIERR, TTOERR, EOF, REL,
NOPARAM, ALATT, TIMER, SIGERR, TTYERR, SUB, NOFNC, UDIS,
NORO, INVSW, MISVAL, 2FARG, 2MARG, NXDEV, NXUN, NXREG,
NXPAR, NEST, IERR, MTRLNT, LOST, TTMO, STALL, AFAIL,
NOTATT, AMBREG
```

These values can be indicated by name or by their internal numeric value (not recommended).

`CONTROL-C Trapping`

A special ON trap is available to describe action(s) to be taken when CONTROL_C (aka SIGINT) occurs during the execution of simh commands and/or command procedures.

```
on CONTROL_C <action>    perform action(s) after CTRL+C
on CONTROL_C             restore default CTRL+C action
```

The default ON CONTROL_C handler will exit nested DO command procedures and return to the sim> prompt.

Note 1: When a simulator is executing instructions entering CTRL+Cwill cause the CNTL+C character to be delivered to the simulator as input. The simulator instruction execution can be stopped by entering the WRU character (usually CTRL+E). Once instruction execution has stopped, CTRL+C can be entered and potentially acted on by the ON CONTROL_C trap handler.
Note 2: The ON CONTROL_C trapping is not affected by the SET ON and SET NOON commands.

4.16.8 Command arguments

```
Token "%0" expands to the command file name.
Token %n (n being a single digit) expands to the n'th argument
Token %* expands to the whole set of arguments (%1 ... %9)

The input sequence "%%" represents a literal "%".  All other character
combinations are rendered literally.

Omitted parameters result in null-string substitutions.

Tokens preceeded and followed by % characters are expanded as environment
variables, and if an environment variable isn't found then it can be one of  the
available Build-In variables.
```

4.16.8.1 DO command argument manipulation

```
The SHIFT command will shift the %1 thru %9 arguments to the left one position.
```

4.16.8.2 Built-In Variables

```
          %DATE%              yyyy-mm-dd
          %TIME%              hh:mm:ss
          %DATETIME%          yyyy-mm-ddThh:mm:ss
          %LDATE%             mm/dd/yy (Locale Formatted)
          %LTIME%             hh:mm:ss am/pm (Locale Formatted)
          %CTIME%             Www Mmm dd hh:mm:ss yyyy (Locale Formatted)
          %DATE_YYYY%         yyyy        (0000-9999)
          %DATE_YY%           yy          (00-99)
          %DATE_MM%           mm          (01-12)
          %DATE_MMM%          mmm         (JAN-DEC)
          %DATE_MONTH%        month       (January-December)
          %DATE_DD%           dd          (01-31)
          %DATE_WW%           ww          (01-53)     ISO 8601 week number
          %DATE_WYYYY%        yyyy        (0000-9999) ISO 8601 week year number
          %DATE_D%            d           (1-7)       ISO 8601 day of week
          %DATE_JJJ%          jjj         (001-366) day of year
          %DATE_19XX_YY%      yy          A year prior to 2000 with the same
                                          calendar days as the current year
          %DATE_19XX_YYYY%    yyyy        A year prior to 2000 with the same
                                          calendar days as the current year
          %TIME_HH%           hh          (00-23)
          %TIME_MM%           mm          (00-59)
          %TIME_SS%           ss          (00-59)
          %TIME_MSEC%         msec        (000-999)
          %STATUS%            Status value from the last command executed
          %TSTATUS%           The text form of the last status value
          %SIM_VERIFY%        The Verify/Verbose mode of the current Do command
file
          %SIM_VERBOSE%       The Verify/Verbose mode of the current Do command
file
          %SIM_QUIET%         The Quiet mode of the current Do command file
```

```
          %SIM_MESSAGE%       The message display status of the current Do
command file
          %SIM_NAME%          The name of the computer that is being simulated.
          %SIM_BIN_NAME%      The name of the simulator program binary
          %SIM_BIN_PATH%      The full path of the simulator program binary
```

4.16.8.3 Environment Variables

Environment variables can be explicitly defined and subsequently available for command substitution:

```
SET ENV variablename=value
```

4.16.8.3.1 Gathering User Input Into an Environment Variable

Input from a user can be obtained by:

SET ENV -p "Prompt String" name=default

The -p switch indicates that the user should be prompted with the indicated prompt string and the input provided will be saved in the environment variable 'name'. If no input is provided, the value specified as 'default' will be used.

4.16.8.3.2 Arithmetic Expressions

Arithmetic Computations into a Variable

SET ENVIRONMENT -A name=expression

Expression can contain any of these C language operators:

(	Open Parenthesis
)	Close Parenthesis
-	Subtraction
+	Addition
*	Multiplication
/	Division
%	Modulus
&&	Logical AND
||	Logical OR
&	Bitwise AND
|	Bitwise Inclusive OR
^	Bitwise Exclusive OR

>>	Bitwise Right Shift
<<	Bitwise Left Shift
==	Equality
!=	Inequality
<=	Less than or Equal
<	Less than
>=	Greater than or Equal
>	Greater than
!	Logical Negation
~	Bitwise Compliment

Operator precedence is consistent with C language precedence. Expression can contain arbitrary combinations of constant values, simulator registers and environment variables. For example:

```
sim> SET ENV -A A=7+2
sim> ECHO A=%A%
A=9
sim> SET ENV -A A=A-1
sim> ECHO A=%A%
A=8
```

4.16.9 Command Aliases

Commands can be aliases with environment variables. For example:

```
sim> set env echo=echof
sim> echo "Hello there"
Hello there
```

4.17 Executing System Commands

The simulator can execute operating system commands with the ! (spawn) command:

```
! <host operating system command>
```

If no operating system command is provided, the simulator attempts to launch the host operating system's command shell.

4.18 Getting Help

The HELP command prints out information about a command or about all commands:

```
HELP                          print all HELP messages
HELP <command>                print HELP for command
HELP <device>                 print HELP for device
HELP <device> REGISTERS       print HELP for device register variables
HELP <device> ATTACH          print HELP for device specific attach
```

```
HELP <device> SET             print HELP for device specific SET commands
HELP <device> SHOW            print HELP for device specific SHOW commands
```

4.19 Recording Simulator activities

The interactions performed with the simulator (at the "sim>" prompt) and the output that those interactions produce can be recorded to a log file/

```
SET LOG <filename>            direct log output to file
SET LOG STDERR                direct log output to stderr
SET LOG DEBUG                 direct log output to debug file
SET NOLOG                     disable output logging
```

```
Output produced by the simulated console device will also be written to the
configured log file if the console is not provided via a telnet session (i.e.
SET CONSOLE TELNET=port).  The output produced by a console telnet session can
also be written to the simulator log file with:
```

```
SET CONSOLE TELNET=LOG=LOG    direct log output to log file
```

4.19.1 Switches

By default, log output is written at the end of the specified log file. A new log file can be created if the –N switch is used on the command line.

4.20 Controlling Debugging

Some simulated devices may provide debug printouts to help in diagnosing complicated problems. Debug output may be sent to a variety of places, or disabled entirely:

```
SET DEBUG STDOUT              direct debug output to stdout
SET DEBUG STDERR              direct debug output to stderr
SET DEBUG LOG                 direct debug output to log file
SET DEBUG <filename>          direct debug output to file
SET NODEBUG                   disable debug output
```

4.20.1 Switches

Debug message output contains a timestamp which indicates the number of simulated instructions which have been executed prior to the debug event.

Debug message output can be enhanced to contain additional, potentially useful information.

4.20.1.1 –f

The -F switch suppresses the internal logic which coalesces successive identical debug output lines into one followed by an indicator of how many times the same line was output. This mode is most appropriate when output is being displayed in real time to STDOUT or STDERR.

4.20.1.2 –t

The -T switch causes debug output to contain a time of day displayed as hh:mm:ss.msec.

4.20.1.3 -a

The -A switch causes debug output to contain a time of day displayed as seconds.msec.

4.20.1.4 -r

The -R switch causes the time of day displayed due to the -T or –A switches to be relative to the start time of debugging. If neither -T or -A is explicitly specified, -T is implied.

4.20.1.5 –p

The -P switch adds the output of the PC (Program Counter) to each debug message.

4.20.1.6 -n

The -N switch causes a new/empty file to be written to. The default is to append to an existing debug log file.

4.20.1.7 -d

The -D switch causes data blob output to also display the data as RADIX-50 characters.

4.20.1.8 -e

The -E switch causes data blob output to also display the data as EBCDIC characters.

4.20.2 Device Debug options

If debug output is enabled, individual devices can be controlled with the `SET` command. If a device has only a single debug flag:

```
SET <device> DEBUG            enable device debug output
SET <device> NODEBUG          disable device debug output
```

If the device has individual, named debug flags:

```
SET <device> DEBUG            enable all debug flags
SET <device> DEBUG=n1;n2;...  enable debug flags n1, n2, ...
SET <device> NODEBUG=n1;n2;... disable debug flags n1, n2, ...
SET <device> NODEBUG          disable all debug flags
```

If debug output is directed to stdout, it will be intermixed with normal simulator output.

4.20.3 Displaying Debug settings

The current debug settings for output destination, options and device specific debug settings can be displayed with:

```
SHOW DEBUG                    display the current debug settings
```

4.21 Exiting the Simulator

`EXIT` (synonyms `QUIT` and `BYE`) returns control to the operating system. An optional numeric exit status may be provided on the `EXIT` command line that an operating system script may act on.

```
EXIT {status}                  return to the operating system
```

4.22 Manipulating Files within the Simulator

Tools to manipulate file containers and to transfer files/data into or out of a simulated environment are provided.

In general, these are tools natively found on the host operating system. They are explicitly support directly from SCP to allow for platform neutral scripts that either test or build running environments for simh users.

4.22.1 Manipulating File Archives

The **tar** command is provided to pack unpack archives as needed.

```
tar -czf xyz.tar.gz *.dsk      archive disk image files
tar -xf xyz.tar                extract files from an archive
```

4.22.2 Transferring data from the web

The **curl** command is provided to access data across the web.

```
curl -L https://github.com/simh/simh/archive/master.zip --output
                               master.zip
```

Appendix 1: File Representations

All file representations are little-endian. On big-endian hosts, the simulator automatically performs any required byte swapping.

A.1 Hard Disks

Hard disks are represented as unstructured binary files of 16b data items for the 12b and 16b simulators, of 32b data items for the 18b, 24b, and 32b simulators, and 64b for the 36b simulators. Device simulations which use the sim_disk library can also have hard disks which are Virtual Hard Disks (as described by the Microsoft Open Specification) and for Windows and Linux hosts can be raw host disks and/or CDROM devices. Disk containers may have 512 bytes of metadata beyond the emulated capacity of the drive. If present, the metadata describes details about the actual drive being emulated and potential additional parameters which may be useful in simulation.

A.2 Floppy Disks

Floppy disks are represented as unstructured binary files of 8b data items. They are nearly identical to the floppy disk images for Doug Jones' PDP-8 simulator but lack the initial 256 byte header. A utility for converting between the two formats is easily written.

A.3 Magnetic Tapes

SIMH format magnetic tapes are represented as unstructured binary files of 8b data items. Each record starts with a 32b record header, in little endian format. If the record header is not a special header, it is followed by n 8b bytes of data, followed by a repeat of the 32b record header. A 1 in the high order bit of the record header indicates an error in the record. If the byte count is odd, the record is padded to even length; the pad byte is undefined.

Special record headers occur only once and have no data. The currently defined special headers are:

```
0x00000000        file mark
0xFFFFFFFF        end of medium
0xFFFFFFFE        erase gap
```

Magnetic tapes are endian independent and consistent across simulator families. A magnetic tape produced by the Nova simulator will appear to have its 16b words byte swapped if read by the PDP-11 simulator.

SIMH can read and write E11-format magnetic tape images. E11 format differs from SIMH format only for odd-length records; the data portion of E11 records is not padded with an extra byte.

SIMH can read TPC-format magnetic tape images. TPC format uses a 16b record header, with 0x0000 denoting file mark. The record header is not repeated at the end of the record. Odd-length records are padded with an extra byte. Some TPC formatted tapes have an end of medium indicated as a record length of 0xffff with no data following the record length.

SIMH can read Pierce-format seven-track magnetic tape images. Pierce format uses only 6 data bits, and one parity bit, in each byte. The high order bit indicates start of record. End of file is indicated by a record of one (occasionally two) bytes consisting of code 017 (octal).

SIMH can read and write AWS format tape images. AWS format uses a 3 16b (little endian) word record header which are a next record size, previous record size and a flag word. The flag word uses value 0x40 to flag tape marks, and 0xA0 for data records.

SIMH can directly read from TAR files. TAR files are read directly and presented through the tape interface as tape records of a fixed size. The default TAR record size is 10240. A specific record size can be specified by using the –B switch on the ATTACH command. Since the specified input file is merely presented as fixed sized records to the simulated system reading from a tape, other data could also be presented via this same mechanism. For example, if a system was able to read 80 byte card images from a tape drive, a binary input file could be attached with a –B 80 and it would be read 80 bytes at a time.

SIMH can present the content of local text and binary files on a pseudo tape device. This is achieved by using one of ANSI-VMS, ANSI-RT11, ANSI-RSTS or ANSI-RSX11 tape formats. The attach command for any ANSI format tape takes a list of file names (which may contain wildcards) and makes them available to a simulated system via the attached tape device presented as an ANSI labeled tape with each file having its attributes presented to the simulated OS. Text files can contain LF or CRLF line endings and they will be visible to an operating system which can read an ANSI tape as normal data in the expected form. Binary files are presented as fixed sized 512 byte records.

A.4 Line Printers

Line printer output is represented by an ASCII file of lines separated by the newline character. Overprinting is represented by a line ending in return rather than newline.

A.5 DECtapes

DECtapes are structured as fixed length blocks. PDP-1/4/7/9/15 DECtapes use 578 blocks of 256 32b words. Each 32b word contains 18b (6 lines) of data. PDP-11 DECtapes use 578 blocks of 256 16b words. Each 16b word contains 6 lines of data, with 2b omitted. This is compatible with native PDP-11 DECtape dump facilities, and with John Wilson's PUTR Program. PDP-8 DECtapes use 1474 blocks of 129 16b words. Each 16b word contains 12b (4 lines) of data. PDP-8 OS/8 does not use the 129th word of each block, and OS/8 DECtape dumps contain only 128 words per block. A utility, DTOS8CVT.C, is provided to convert OS/8 DECtape dumps to simulator format.

A known issue in DECtape format is that when a block is recorded in one direction and read in the other, the bits in a word are scrambled (to the complement obverse). The PDP-11 deals with this problem by performing an automatic complement obverse on reverse writes and reads. The other systems leave this problem to software.

The simulator represents this difference as follows. On the PDP-11, all data is represented in normal form. Data reads and writes are not direction sensitive; read all and write all are direction sensitive. Real DECtapes that are read forward will generate images with the correct representation of the data.

On the other systems, forward write creates data in normal form, while reverse write creates data in complement obverse form. Forward read (and read all) performs no transformations, while reverse read (and read all) changes data to the complement obverse. Real DECtapes that are read forward will generate data in normal form for blocks written forward, and complement obverse data for blocks written in reverse, corresponding to the simulator format.

Appendix 2: Debug Status

The debug status of each simulated CPU and device is as follows:

```
legend:           y = runs operating system or sample program
                  d = runs diagnostics
                  h = runs hand-generated test cases
                  n = untested
                  - = not applicable

      System    PDP-8 PDP-11 Nova PDP-1 18b PDP
device
CPU               y     y     y     y     y
FPU               y     y     -     -     y
EIS/CIS           -     y     -     -     -
console           y     y     y     y     y
paper tape        y     y     y     y     y
card reader       -     y     -     -     -
line printer      y     y     y     h     y
clock             y     y     y     -     y
extra terminal    y     y     y     -     y
hard disk         y     y     y     -     y
fixed disk        y     y     h     -     y
floppy disk       y     y     y     -     -
drum              -     -     -     h     h
DECtape           y     y     -     h     y
mag tape          y     y     y     -     y

      system    1401  2100  PDP-10 H316 MicroVAX 3900
device
CPU               y     y     y     h     y
FPU               -     y     y     -     y
EIS/CIS           -     y     y     -     -
console           y     y     y     h     y
paper tape        -     y     n     h     -
card reader       y     -     -     -     y
line printer      y     y     y     h     y
clock             -     y     y     h     y
extra terminal    -     y     y     -     y
hard disk         h     y     y     h     y
fixed disk        -     y     -     h     -
floppy disk       -     -     -     -     y
drum              -     y     -     -     -
DECtape           -     -     -     -     -
mag tape          y     y     y     h     y

      system    GRI   1620   i16   i32  SDS940
device
CPU               h     y     d     y     d
FPU               -     y     d     y     -
CIS               -     -     -     -     -
console           h     y     d     y     h
paper tape        h     y     d     y     h
card reader       -     y     -     -     -
line printer      -     y     d     y     h
```

clock	h	-	d	y	n
extra terminal	-	-	h	y	h
hard disk	-	h	d	y	h
fixed disk	-	-	-	-	h
floppy disk	-	-	d	d	-
drum	-	-	-	-	h
DECtape	-	-	-	-	-
mag tape	-	-	d	y	h

system	LGP-30	3000	780
device			
CPU	h	y	y
FPU	-	y	y
CIS	-	y	y
console	h	y	y
paper tape	h	-	-
card reader	-	-	y
line printer	-	y	y
clock	-	y	y
extra terminal	-	y	y
hard disk	-	y	y
fixed disk	-	-	-
floppy disk	-	-	y
drum	-	-	-
DECtape	-	-	-
mag tape	-	y	y

Revision History (covering Rev 2.0 to Rev 3.5)

Starting with Rev 2.7, detailed revision histories can be found in file sim_rev.h.

Rev 3.5, Sep, 05
- Overhauled sources for readability
- Added VAX-11/780

Rev 3.4, May, 05
- Revised memory interaction model

Rev 3.3, Nov, 04
- Added PDP-11/VAX DHQ11 support
- Added PDP-11/VAX TM02/TM03 support
- Added PDP-11 model-specific emulation support
- Added full VAX support
- Replaced SET ONLINE/OFFLINE with SET ENABLED/DISABLED

Rev 3.2, Apr, 04
- Added LGP-30/LGP-21 simulator
- Added global SHOW modifier capability
- Added global SET DEBUG modifier
- Added global SHOW DEBUG,RADIX,MODIFIERS,NAME modifiers
- Added VAX extended physical memory support (Mark Pizzolato)
- Added VAX RXV21 support
- Revised terminal multiplexer library to support variable number of lines per multiplexer
- Added PDP-15 LT19 support (1-16 terminals)

Rev 3.1, Dec, 03
- Added Alpha/VMS, FreeBSD, Mac OS/X Ethernet library support
- Added Eclipse floating point and interval timer support (from Charles Owen)
- Added PDP-1 parallel drum support
- Added PDP-8 TSC8-75 and TD8E support
- Added H316/516 DMA/DMC, magtape, fixed head disk support
- Added PDP-8, PDP-15, 32b Interdata instruction history support

Rev 3.0, May, 03
- Added logical name support
- Added instruction history support
- Added multiple tape format support
- Added 64b address support
- Added PDP-4 EAE support
- Added PDP-15 FP15 and XVM support

Rev 2.10, Nov, 02
- Added Telnet console capability, removed VT emulation
- Added DO with substitutable arguments (from Brian Knittel)
- Added .ini initialization file (from Hans Pufal)
- Added quiet mode (from Brian Knittel)
- Added ! command (from Mark Pizzolato)
- Added Telnet BREAK support (from Mark Pizzolato)
- Added device enable/disable support
- Added optional simulator hooks for input, output, commands

- Added breakpoint actions
- Added magnetic tape simulation library
- Added PDP-11 KW11P programmable clock
- Added PDP-11 RK611/RK06/RK07 disk
- Added PDP-11/VAX TMSCP tape
- Added PDP-11/VAX DELQA Ethernet support (from David Hittner)
- Added PDP-11/PDP-10 RX211/RX02 floppy disk
- Added PDP-11/VAX autoconfiguration support
- Added PDP-10/PDP-11/VAX variable vector support
- Added PDP-1 DECtape
- Added PDP-1, PDP-4 Type 24 serial drum support
- Added PDP-8 RX28 support
- Added PDP-9 RB09 fixed head disk, LP09 line printer
- Added HP2100 12845A line printer
- Added HP2100 13183 magtape support
- Added HP2100 boot ROM support
- Added HP2100 interprocessor link support
- Added IBM 1620
- Added SDS 940
- Added Interdata 16b and 32b systems
- Added 16b DECtape file format support
- Added support for statically buffered devices
- Added magnetic tape end of medium support
- Added 50/60Hz support to line frequency clocks
- Added 7B/8B support to terminals and multiplexers
- Added BREAK support to terminals and multiplexers

Rev 2.9, Jan, 02

- Added circular register arrays
- Replaced ENABLE/DISABLE with SET ENABLED/DISABLED
- Replaced LOG/NOLOG with SET LOG/NOLOG
- Generalized the timer calibration package
- Added additional routines to the multiplexer library
- Added SET DISCONNECT, SHOW STATISTICS commands to multiplexers
- Re-implemented PDP-8 TTX as a unified multiplexer
- Implemented a PC queue in most simulators
- Added VAX simulator
- Added GRI-909 simulator
- Added Peter Schorn's MITS 8080/Z80 simulator
- Added Brian Knittel's IBM 1130 simulator
- Added HP2100 DQ, DR, MS, MUX devices
- Added SET VT/NOVT commands

Rev 2.8, Dec, 01

- Added DO command
- Added general breakpoint facility
- Added extended SET/SHOW capability
- Replaced ADD/REMOVE with SET ONLINE/OFFLINE
- Added global register name recognition
- Added unit-based register arrays
- Added Charles Owen's System 3 simulator
- Added PDP-11 I/O bus map
- Added PDP-11/VAX RQDX3
- Added PDP-8 RL8A
- Revised 18b PDP interrupt structure
- Revised directory and documentation structure

- Added support for MINGW environment

Rev 2.7, Sep, 01
- Added DZ11 (from Thord Nilson and Art Krewat) to PDP-11, PDP-10
- Added additional terminals to PDP-8
- Added TSS/8 packed character format to PDP-8
- Added sim_sock and sim_tmxr libraries
- Added sim_qcount and simulator exit detach all facilities
- Added Macintosh sim_sock support (from Peter Schorn)
- Added simulator revision level, SHOW version
- Changed int64/uint64 to t_int64/t_uint64 for Windows
- Fixed bug in PDP-11 interrupt acknowledge
- Fixed bugs in PDP-11 TS NXM check, boot code, error status; added extended characteristics and status
- Fixed bug in PDP-11 TC stop, stop all functions
- Fixed receive interrupt while disconnected bug in DZ11
- Fixed multi-unit operation bugs, interrupt bugs in PDP-11
- RP, PDP-10 RP, PDP-10 TU
- Fixed carrier detect bug in PDP-11, PDP-10 DZ
- Fixed bug in PDP-8 reset routine
- Fixed conditional in PDP-18b CPU
- Fixed SC = 0 bug in PDP-18b EAE
- Fixed bug in PDP-7 LPT
- Upgraded Nova second terminal to use sim_tmxr
- Upgraded PDP-18b second terminal to use sim_tmxr
- Upgraded PDP-11 LTC to full KW11-L
- Removed hack multiple console support

Rev 2.6b, Aug, 01
- Added H316/516 simulator
- Added Macintosh support from Louis Chrétien, Peter Schorn, and Ben Supnik
- Added bad block table option to PDP-11 RL, RP
- Removed register in declarations
- Fixed bugs found by Peter Schorn
    - endian error in PDP-10, PDP-11 RP
    - space reverse error in PDP-11 TS
    - symbolic input in 1401
- Fixed bug in PDP-1 RIM loader found by Derek Peschel
- Fixed bug in Nova fixed head disk

Rev 2.6a, Jun, 01
- Added PDP-9, PDP-15 API option
- Added PDP-9, PDP-15 second terminal
- Added PDP-10 option for TOPS-20 V4.1 bug fix
- Added PDP-10 FE CTRL-C option for Windows
- Added console logging
- Added multiple console support
- Added comment recognition
- Increased size of string buffers for long path names
- Fixed bug in big-endian I/O found by Dave Conroy
- Fixed DECtape reset in PDP-8, PDP-11, PDP-9/15
- Fixed RIM loader PC handling in PDP-9/15
- Fixed indirect pointers in PDP-10 paging
- Fixed SSC handling in PDP-10 TM02/TU45
- Fixed JMS to non-existent memory in PDP-8
- Fixed error handling on command file

Rev 2.6, May, 01

- Added ENABLE/DISABLE devices
- Added SHOW DEVICES
- Added examination/modification of register arrays
- Added PDP-10 simulator
- Added clock autocalibration to SCP, Nova, PDP-8, PDP-11, PDP-18b
- Added PDP-8, PDP-11, PDP-9/15 DECtape
- Added PDP-8 DF32
- Added 4k Disk Monitor boot to PDP-8 RF08 and DF32
- Added PDP-4/7 funny format loader support
- Added extension handling to the PDP-8 and -9/15 loaders
- Added PDP-11 TS11/TSV05
- Added integer interval timer to SCP
- Added filename argument to LOAD/DUMP
- Revised magnetic tape and DECtape bootstraps to rewind before first instruction
- Fixed 3 cycle data break sequence in PDP-8 RF
- Fixed 3 cycle data break sequence in 18b PDP LP, MT, RF
- Fixed CS1.TRE write, CS2.MXF,UPE write, and CS2.UAI in PDP-11 RP
- Fixed 4M memory size definition in PDP-11
- Fixed attach bug in RESTORE
- Fixed detach bug for buffered devices
- Updated copyright notices, fixed comments

Rev 2.5a, Dec, 00

- Added CMD flop to HP paper tape and line printer
- Added status input for HP paper tape punch and TTY
- Added Charles Owen's 1401 mag tape boot routine
- Added Bruce Ray's Nova plotter and second terminal modules
- Added Charles Owen's Eclipse CPU support
- Added PDP-9/PDP-15 RIM/BIN loader support
- Added PDP-9/PDP-15 extend/bank initial state registers
- Added PDP-9/PDP-15 half/full duplex support
- Moved software documentation to a separate file
- Fixed SCP handling of devices without units
- Fixed FLG, FBF initialization in many HP peripherals
- Fixed 1401 bugs found by Charles Owen
    - 4, 7 char NOPs are legal
    - 1 char B is chained BCE
    - MCE moves whole character, not digit, after first
- Fixed Nova bugs found by Bruce Ray
    - traps implemented on Nova 3 as well as Nova 4
    - DIV and DIVS 0/0 set carry
    - RETN sets SP from FP at outset
    - IORST does not clear carry
    - Nova 4 implements two undocumented instructions
- Fixed bugs in 18b PDP's
    - XCT indirect address calculation
    - missing index instructions in PDP-15
    - bank mode handling in PDP-15

Rev 2.5, Nov, 00

- Removed Digital and Compaq from copyrights, as authorized by Compaq Sr VP Bill Strecker
- Revised save/restore format for 64b simulators
- Added examine to file

- Added unsigned integer data types to sim_defs
- Added Nova 3 and Nova 4 instructions to Nova CPU
- Added HP2100
- Fixed indirect loop through autoinc/dec in Nova CPU
- Fixed MDV enabled test in Nova CPU

Rev 2.4, Jan, 99
- Placed all sources under X11-like open source license
- Added DUMP command, revised sim_load interface
- Added SHOW MODIFIERS command
- Revised magnetic tape format to include record error flag
- Fixed 64b problems in SCP
- Fixed big endian problem in PDP-11 bad block routine
- Fixed interrupt on error bug in PDP-11 RP/RM disks
- Fixed ROL/ROR inversion in PDP-11 symbolic routines

Rev 2.3d, Sep, 98
- Added BeOS support
- Added radix commands and switches
- Added PDP-11 CIS support
- Added RT11 V5.3 to distribution kits
- Fixed "shift 32" bugs in SCP, PDP-11 floating point
- Fixed bug in PDP-11 paper tape reader
- Fixed bug in ^D handling

Rev 2.3c, May, 98
- Fixed bug in PDP-11 DIV overflow check
- Fixed bugs in PDP-11 magnetic tape bootstrap
- Fixed bug in PDP-11 magnetic tape unit select
- Replaced UNIX V7 disk images

Rev 2.3b, May, 98
- Added switch recognition to all simulator commands
- Added RIM loader to PDP-8 paper tape reader and loader
- Added second block bootstrap to PDP-11 magnetic tape
- Fixed bug in PDP-8 RF bootstrap
- Fixed bug in PDP-11 symbolic display
- Fixed bugs in PDP-11 floating point (LDEXP, STEXP, MODf, STCfi, overflow handling)

Rev 2.3a, Nov, 97
- Added search capability
- Added bad block table command to PDP-11 disks
- Added bootstrap to PDP-11 magnetic tape
- Added additional Nova moving head disks
- Added RT-11 sample software
- Fixed bugs in PDP-11 RM/RP disks
- Fixed bugs in Nova moving head disks
- Fixed endian dependence in 18b PDP RIM loader

Rev 2.3, Mar, 97
- Added PDP-11 RP
- Added PDP-1
- Changed UNIX terminal I/O to TERMIOS
- Changed magnetic tape format to double ended
- Changed PDP-8 current page mnemonic from T to C
- Added endian independent I/O routines

- Added precise integer data types
- Fixed bug in sim_poll_kbd
- Fixed bug in PDP-8 binary loader
- Fixed bugs in TM11 magnetic tape
- Fixed bug in RX11 bootstrap
- Fixed bug in 18b PDP ADD
- Fixed bug in 18b PDP paper tape reader
- Fixed bug in PDP-4 console
- Fixed bug in PDP-4, 7 line printer

Rev 2.2d, Dec, 96
- Added ADD/REMOVE commands
- Added unit enable/disable support to device simulators
- Added features for IBM 1401 project
- Added switch recognition for symbolic input
- Fixed bug in variable length IEXAMINE
- Fixed LCD bug in RX8E
- Initial changes for Win32
- Added IBM 1401

Rev 2.2b, Apr, 96
- Added PDP-11 dynamic memory size support

Rev 2.2a, Feb, 96
- New endian independent magnetic tape format

Rev 2.2 Jan, 96
- Added register buffers for save/restore
- Added 18b PDP's
- Guaranteed TTI, CLK times are non-zero
- Fixed breakpoint/RUN interaction bug
- Fixed magnetic tape backspace to EOF bug
- Fixed ISZ/DCA inversion in PDP-8 symbol table
- Fixed sixbit conversion in PDP-8 examine/deposit
- Fixed origin increment bug in PDP-11 binary loader
- Fixed GCC longjmp optimization bug in PDP-11 CPU
- Fixed unit number calculation bug in SCP and in Nova, PDP-11, 18b PDP moving head disks

Rev 2.1 Dec, 95
- Fixed PTR bug (setting done on EOF) in PDP-8, Nova
- Fixed RX bug (setting error on INIT if drive 1 is not attached) in PDP-8, PDP-11
- Fixed RF treatment of photocell flag in PDP-8
- Fixed autosize bug (always chose smallest disk if new file) in PDP-11, Nova
- Fixed not attached bug (reported as not attachable) in most mass storage devices
- Fixed Nova boot ROMs
- Fixed bug in RESTORE (didn't requeue if delay = 0)
- Fixed bug in RESTORE (clobbered device position)
- Declared static constant arrays as static const
- Added PDP-8, Nova magnetic tape simulators
- Added Dasher mode to Nova terminal simulator
- Added LINUX support

Rev 2.0 May, 95
- Added symbolic assembly/disassembly

Acknowledgements

SIMH would not have been possible without help from around the world. I would like to acknowledge the help of the following people, all of whom donated their time and talent to this "computer archaeology" project:

Bill Ackerman	PDP-1 consulting
Alan Bawden	ITS consulting
Winfried Bergmann	Linux port testing
Phil Budne	Solaris port testing
Max Burnet	PDP information, documentation, and software
J. David Bryan	HP Simulators
Robert Alan Byer	VMS socket support and testing
James Carpenter	LINUX port testing
Chip Charlot	PDP-11 RT-11, RSTS/E, RSX-11M legal permissions
Louis Chrétien	Macintosh porting
Dave Conroy	HP 21xx documentation, PDP-10, PDP-18b debugging
L Peter Deutsch	PDP-1 LISP software
Ethan Dicks	PDP-11 2.9 BSD debugging
John Dundas	PDP-11 CPU debugging, programmable clock simulator
Jonathan Engdahl	PDP-11 device debugging
Carl Friend	Nova and Interdata documentation, and RDOS software
Megan Gentry	PDP-11 integer debugging
Dave Gesswein	PDP-8 and PDP-9.15 documentation, PDP-8 DECtape, disk, and paper-tape images, PDP-9/15 DECtape images
Dick Greeley	PDP-8 OS/8 and PDP-10 TOPS-10/20 legal permissions
Gordon Greene	PDP-1 LISP machine readable source
Lynne Grettum	PDP-11 RT-11, RSTS/E, RSX-11M legal permissions
Franc Grootjen	PDP-11 2.11 BSD debugging
Doug Gwyn	Portability debugging
Kevin Handy	TS11/TSV05 documentation, make file
Ken Harrenstein	KLH PDP-10 simulator
Bill Haygood	PDP-8 information, simulator, and software
Wolfgang Helbig	DZ11 implementation
Mark Hittinger	PDP-10 debugging
Dave Hittner	SCP debugging, DEQNA emulator and Ethernet library
Sellam Ismail	GRI-909 documentation
Jay Jaeger	IBM 1401 consulting
Doug Jones	PDP-8 information, simulator, and software
Brian Knittel	IBM 1130 simulator, SCP extensions for GUI support
Al Kossow	HP 21xx, Varian 620, TI 990, Interdata, DEC documentation and software
Arthur Krewat	DZ11 changes for the PDP-10
Mirian Crzig Lennox	ITS and DZ11 debugging
Don Lewine	Nova documentation and legal permissions
Tim Litt	PDP-10 hardware documentation and schematics, tape images, and software sources
Tim Markson	DZ11 debugging
Bill McDermith	HP 2100 debugging, 12565A simulator
Scott McGregor	PDP-11 UNIX legal permissions
Jeff Moffatt	HP 2100 information, documentation, and software
Alec Muffett	Solaris port testing
Terry Newton	HP 21MX debugging
Thord Nilson	DZ11 implementation
Charles Owen	Nova moving head disk debugging, Altair simulator, Eclipse simulator,

	IBM System 3 simulator, IBM 1401 diagnostics, debugging, and magtape boot
Sergio Pedraja	MINGW environment debugging
Derek Peschel	PDP-10 debugging
Paul Pierce	IBM 1401 diagnostics, media recovery
Mark Pizzolato	SCP, Ethernet, and VAX simulator improvements
Hans Pufal	PDP-10 debugging, PDP-15 bootstrap, DOS-15 recovery, DOS-15 documentation, PDP-9 restoration
Bruce Ray	Software, documentation, bug fixes, and new devices for the Nova, OS/2 porting
Craig St Clair	DEC documentation
Richard Schedler	Public repository maintenance
Peter Schorn	Macintosh porting
Stephen Schultz	PDP-11 2.11 BSD debugging
Olaf Seibert	NetBSD port testing
Brian & Barry Silverman	PDP-1 simulator and software
Tim Shoppa	Nova documentation, RDOS software, PDP-10 and PDP-11 software archive, hosting for SIMH site
Van Snyder	IBM 1401 zero footprint bootstraps
Michael Somos	PDP-1 debugging
Hans-Michael Stahl	OS/2 port testing, TERMIOS implementation
Tim Stark	TS10 PDP-10 simulator
Larry Stewart	Initial suggestion for the project
Bill Strecker	Permission to revert copyrights
Chris Suddick	PDP-11 floating point debugging
Ben Supnik	Macintosh timing routine
Bob Supnik	SIMH simulators
Ben Thomas	VMS character-by-character I/O routines
Warren Toomey	PDP-11 UNIX software
Deb Toivonen	DEC documentation
Mike Umbricht	DEC documentation, H316 documentation and schematics
Leendert Van Doorn	PDP-11 UNIX V6 debugging, TERMIOS implementation
Fred Van Kempen	Ethernet code, RK611 emulator, PDP-11 debugging, VAX/Ultrix debugging
Holger Veit	OS/2 socket support
David Waks	PDP-8 ESI-X and PDP-7 SIM8 software
Tom West	Nova documentation
Adrian Wise	H316 simulator, documentation, and software
John Wilson	PDP-11 simulator and software
Joe Young	RP debugging on Ultrix-11 and BSD
Jordi Guillaumes i Pons	Testing and CR11/CD11 fixes

In addition, the following companies have graciously licensed their software at no cost for hobbyist use:

Data General Corporation
Digital Equipment Corporation
Compaq Computer Corporation
Mentec Corporation
The Santa Cruz Operation
Caldera Corporation
Hewlett-Packard Corporation