Emulation and computing history

The Computer History Simulation Project and SIMH

M. Tim Jones (mtj@mtjones.com)
Senior Architect
Emulex

22 March 2011

The simplest computing devices we use today have more processing capability than the most capable computing systems of yesterday. For example, the VAX 11/780 delivered around 0.5 MIPS in the early 1980s. Compare that to an IBM zEnterprise™ 196 (z196) mainframe of today, which can support well over 52 KMIPS. However, we can learn a lot from early computing history. If you've ever wanted to boot an IBM 1130, PDP-11, or MITS Altair, then the Computer History Simulation Project is just what you've been looking for.

In late 1978, as a birthday gift, I received my first computer. It was a TRS-80 model 1 with 4KB of memory and cassette tape mass storage, which I later upgraded to an Exatron stringy floppy. Within a few weeks, my BASIC programming skills had evolved to the point that I had exceeded the available memory for my yet-to-be-completed program: a sad day. Little did I know 30 years later, as an embedded firmware engineer, I would still spend much of my time trying to squeeze more code and data into a smaller address space.

Computer history is fascinating, as are some of the early computers that were developed. Many of the early machines were rudimentary calculators, such as Konrad Zuse's Z1, which he invented in 1931, and going back even farther, Herman Hollerith's mechanical sorting machine that was used in the 1890 census; six years later his is one of the companies that merged and became IBM. Zuse also introduced the first algorithmic programming language called *Plankalkül* for his Z4 computer. The Z4 was electromechanical (relay-based), supported 64 words of memory, and ran at a whopping 40Hz (at 4KW of consumed power). Professor John Atanasoff developed the first digital computer, which he began in 1937 and completed in 1941, that used binary for computation at Iowa State College. The ideas of the Atanasoff-Berry Computer (ABC) were used in the first general-purpose electronic computer, ENIAC. Programming these systems would be foreign to most of us, who grew up with Pascal, `c`, or LISP. For example, programming the first ENIAC required physical rewiring to change its programming. Wikipedia provides fascinating information on computer evolution (see Resources).

The Computer History Simulation Project

Like my old TRS-80, which sits lifeless in a dusty box, computing history can easily slip away. Luckily, there are folks like Bob Supnik who have invested time and energy into rescuing some of computing's most important history. Supnik created the Computer History Simulation Project as a way to restore significant computing systems through simulation. Supnik's project, which has contributions from people around the world, has resulted in a multi-system framework called *SIMH* that simulates a large number of computing systems. Some of the systems include the Data General Nova, DEC's PDPs and VAX, IBM 1401 and 7090/7094, Interdata systems, and even the MITS Altair systems (for both 8080 and Z80). See Resources for other simulated systems.

The SIMH architecture

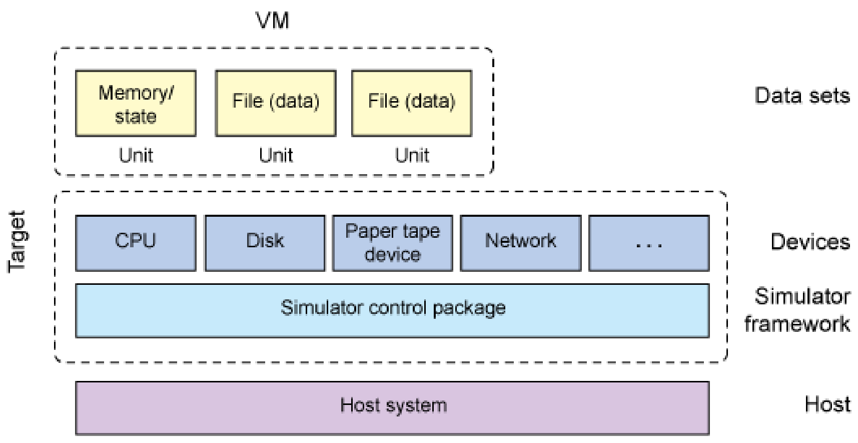
Let's first examine the general architecture of SIMH, and then explore some sample usages of SIMH to boot older, simulated hardware.

More on SIMH architecture

The gory details of SIMH can be found in a document called "Writing a Simulator for the SIMH System." This document presents the major functions of SIMH and details of the various machines that are simulated (see Resources).

SIMH is a framework into which system emulators are built. When you explore the hierarchy of the SIMH source tree, you'll find a set of general simulator files, and then a set of subdirectories that contains machine simulations (or classes of machine simulations). As shown in Figure 1, a simulator is made up of the simulator control package (containing such things as the simulator console, terminal and file I/O libraries, and timers and sockets) and a set of devices. The spirit of the devices is a set of registers to maintain state about the device (while it's active) and units representing data sets (commonly files that are used to represent the device). Not all devices must be present, for example, networking may not be suitable.

Figure 1. Basic architecture of SIMH



When a virtual machine starts on the simulator, the simulator control package first identifies the particulars of the environment (array of devices, binary loader routine, instruction set simulator, and so on). As you'll see shortly, after the simulator is started, you can interact with it to boot a virtual machine (or place this interaction in a file to simplify the steps). The simulator in essence is a debugger, so you can also set breakpoints and interact with the devices, CPU, and memory. Considering the amount of functionality provided in SIMH, the code is easily readable and not as large as you may expect.

Setting up SIMH

Let's explore how SIMH can return you to the early days of computing. This section first explains how to install and build SIMH, then discusses software kits.

Installing SIMH

The following session illustrates how to download and build SIMH. It demonstrates version 38.1 of SIMH, although you should check the website to see whether the package has evolved. As noted in Listing 1, I needed to create the ./BIN subdirectory before the package could be made.

Listing 1. Installing and building SIMH

```
$ mkdir simh ; cd simh

$ wget http://simh.trailing-edge.com/sources/simhv38-1.zip

$ unzip simhv38-1.zip
$ mkdir BIN
$ make -f makefile
```

After you complete the steps in Listing 1, you'll have a set of binaries in the ./BIN subdirectory representing the individual simulators (such as `ibm1130` and `vax780`).

Software kits

In the preceding step, you built the SIMH simulator. However, for it to be useful, you need software (operating systems and applications) for these simulators. Because these systems operated with paper tape and magnetic tapes, the software represented by those media has been repackaged into files for ease of use. As part of these demonstrations, you'll explore the necessary software kits and learn where to download them.

Navigating the simulator

When you start a simulator (named for the machine that it's simulating, such as `altairz80`), you'll see a `sim>` prompt. This prompt tells you that you're in the base simulator, which you can then configure to bring up the simulated target. This article explores some of the many commands that can be executed. You can use `help` to see the extensive list. It's also important to note that while emulating one of the support machines, you can use Ctrl-E to return you to the simulator. From there, you can set breakpoints, examine registers, look at the simulator event queue, and more.

Using SIMH

Now that you have a basic understanding of SIMH, let's explore its use with a variety of computer systems. You'll look at LISP on an IBM 1130 system, UNIX® on the Interdata 32-bit system, CPM on the MITS Altair machine, and Focal on a PDP-15.

IBM 1130

The IBM 1130 system was a popular computing system that focused on lower-cost markets. It relied on punched cards and paper tape but also used inexpensive disk storage (1MB total). The disk stored the operating system and data.

The 1130 system was introduced in 1965, when the primary programming language was FORTRAN (whose compiler, written entirely in assembly language, ran with only 4000 words of memory). It used a 15-bit word-based address space, limiting the machine to 64KB of core memory.

One of the interesting aspects of the IBM 1130 system was its support for alternative languages. In addition to FORTRAN, the 1130 could be programmed using APL and RPG. Guy Steele, who had access to an 1130 at Boston's Latin (high) school, wrote a LISP interpreter that we can still use today. Let's begin by getting the LISP interpreter from the site that maintains the 1130, ibm1130.org (see Listing 2). Note that I assume that you're in the ./simh subdirectory, where you installed SIMH.

Listing 2. Installing the LISP interpreter software kit

```
$ mkdir kits/ibm1130 ; cd kits/ibm1130
$ wget http://media.ibm1130.org/lisp.zip
$ unzip lisp.zip
```

This emulation hides many of the details of working with SIMH but is an interesting historical peek at one use of the 1130. This demonstration is a batch operation in which you specify a job to the simulator, whose output is then emitted to a list file. Your job is a very simple use of the LISP interpreter (which you can see in the output). Listing 3 provides the batch session.

Listing 3. Using the LISP interpreter with the IBM 1130 simulator

```
$ ../../BIN/ibm1130 job lisptest

IBM 1130 simulator V3.8-1
PRT: creating new file
Loaded DMS V2M12 cold start card

Wait, IAR: 0000002A (4c80 BSC  I  ,0028   )
sim> quit
Goodbye
$ more lisptest.lst

PAGE   1

// JOB    1234

LOG DRIVE   CART SPEC   CART AVAIL  PHY DRIVE
  0000        1234        1234        0000

V2 M12   ACTUAL 32K  CONFIG 32K
^L
PAGE   1

// JOB

LOG DRIVE   CART SPEC   CART AVAIL  PHY DRIVE
  0000        1234        1234        0000
```

```
V2 M12   ACTUAL 32K  CONFIG 32K

// XEQ LISP

***** 1130 LISP 1.6 ***** BOSTON LATIN SCHOOL ***** LITHP ITH LITHTENING...


(SETQQ A (X Y Z))

(X Y Z)


(CAR A)

X


(CDR A)

(Y Z)


(PLUS 1 2 3)

6


(QUIT)

***** 1130 LISP 1.6 ***** END OF RUN ***** THO LONG, COME AGAIN THOON
$
```

In Listing 3, you start the IBM 1130 simulator and specify a job to perform. Under the covers, the code configures the simulator for the disk (where it loads the LISP interpreter), the card reader (where it reads the job), and the printer (where it emits the output). Your job appears in Listing 4 (which is lisptest.job in the software kit). You can find the output of the job in the same subdirectory as the job name with the `.lst` suffix.

Listing 4. The file lisptest.job

```
// JOB
// XEQ LISP
(SETQQ A (X Y Z))
(CAR A)
(CDR A)
(PLUS 1 2 3)
(QUIT)
```

IBM1130.org developed and maintains the IBM 1130 software kit. At this site, you can also find an APL deck (in addition to FORTRAN and RPG).

Interdata 32 bit with UNIX V6

Interdata developed a set of 16- and 32-bit minicomputers beginning in 1966 (in 1973 becoming part of Perkin-Elmer). The Interdata-7, introduced in 1974, was one of the first 32-bit computers. The architecture for the Interdata was loosely based on the IBM System/360 mainframe architecture. This example looks a bit deeper at how the machine is brought up from the perspective of the simulator (configuring options through the simulator).

In Listing 5, you download the UNIX version 6 image and extract it into a subdirectory under kits. When you extract the download, you start the Interdata simulator and begin configuration. You first enable the console terminal (TTP) and associate the programmable asynchronous line controller (PAS) and magnetic tape controller (MT) to avoid device number conflicts.

Your boot device is the cartridge disk controller (DP), which you associate with the external file from the software kit (iu6_dp0.dsk). With the boot disk defined, you can now boot this disk, which results in starting the UNIX V6 image. The Interdata was one of the first ports of UNIX to a non-PDP system.

Listing 5. Simulating the Interdata 32b system with UNIX

```
$ mkdir kits/id_unix_v6 ; cd kits/id_unix_v6
$ wget http://simh.trailing-edge.com/kits/iu6swre.zip
$ unzip iu6swre.zip
$ ../../BIN/id32

Interdata 32b simulator V3.8-1
sim> set ttp ena
sim> set pas dev=12
sim> set mt dev=85
sim> att -e dp0 iu6_dp0.dsk
sim> boot dp0
?unix
Memory = 182.50 K

login: root
You have mail.
# ls -la
total 361
drwxr-xr-x  9 root      272 Jun  4 00:16 .
drwxr-xr-x  9 root      272 Jun  4 00:16 ..
drwxr-xr-x  2 root     1104 Nov 14  1978 bin
drwxr-xr-x  2 root      784 Jun  4 01:07 dev
drwxr-xr-x  2 root      528 Nov 14  1978 etc
drwxr-xr-x  2 root      240 Jun  3 21:44 lib
-rw-r--r--  1 root      552 Jun  3 20:48 mdl
drwxr-x---  2 root       32 Aug  2  1978 mnt
drwxrwxrwx  2 root      144 Jun  4 15:38 tmp
-rw-r--r--  1 root      424 Jun  4 00:27 tpboot
-rw-r--r--  1 root      568 Jun  3 20:48 tuboot
-rw-r--r--  1 root      728 Jun  3 20:48 uboot
-rw-r-----  1 root    52272 Jun  3 23:56 unix
-rwxrwxrwx  1 root    59236 Jun  4 01:21 unix.oxon
-rwxrwxrwx  1 rm      60852 Jun  4 01:44 unix.sydney
drwxr-xr-x 14 root      240 Jun  4 15:43 usr
#
```

As shown in Listing 5, after the system boots, you can interact with it in a way similar to modern-day UNIX systems.

PDP-15 with FOCAL

DEC's last 18-bit system was the PDP-15, introduced in 1969. The PDP-15 was implemented with TTL-integrated circuits, unlike earlier PDPs, which were built from discrete transistors. The PDP-15 was compatible with the earlier PDP-9 and included various advanced features, including memory protection and floating points.

Although the PDP-15 supported a variety of operating systems, one interesting use of this machine is with the formula calculator (FOCAL) language. FOCAL was originally written for the PDP-8 by Richard Merrill and could run in a system with only 3000 words of memory (12 bits), with 1000 words for the user program. FOCAL required no operating system and was a complete environment in itself. The FOCAL system is provided as a paper-tape image used in the binary loader format (see Listing 6). After starting the PDP-15 simulator, you load the focal15 image and issue a run to start. You're greeted with the FOCAL15 prompt, where you enter a short FOCAL program and start it with the `GO` command.

Listing 6. Demonstrating FOCAL on a PDP-15

```
$ ../../BIN/pdp15

PDP-15 simulator V3.8-1
sim> load focal15.bin
sim> run

FOCAL15 V6B
*01.10 ASK "WHAT YEAR WERE YOU BORN?", BORN
*01.20 ASK "WHAT YEAR IS IT?", YEAR
*01.30 SET AGE=YEAR-BORN
*01.40 TYPE "YOU ARE ", AGE-1, " OR ", AGE, " YEARS OLD.", !
*GO
WHAT YEAR WERE YOU BORN?:1964
WHAT YEAR IS IT?:2010
YOU ARE    45.0000 OR    46.0000 YEARS OLD.
*
<Ctrl-E>
Simulation stopped, PC: 000221 (SPA)
sim> show dev
PDP-15 simulator configuration

CPU, idle disabled
CLK, 60Hz, devno=00
FPP
PTR, devno=01
PTP, devno=02
TTI, devno=03
TTO, devno=04
LP9, disabled
LPT, devno=65-66
RF, devno=70-72
RP, devno=63-64, 8 units
DT, devno=75-76, 8 units
MT, devno=73, 8 units
TTIX, lines=1, devno=40-47
TTOX
sim>
```

Also shown here is an interaction with the simulator. Press Control-E to exit the simulation and return to the simulator framework console. At this point, you can request that the devices be enumerated, which shows the various devices simulated for this PDP-15 (floating-point processor, paper tape reader, paper tape punch, and so on).

This example demonstrates another aspect of SIMH, where your image requires no operating system and is interactive (differing from the earlier batch run on the IBM 1130 system).

MITS Altair with CP/M

The last demonstration of SIMH simulates the world's first minicomputer, the MITS Altair. The Altair, designed in 1975, was based on the Intel® 8080 processor, with only 256 bytes of memory. Despite its minimal capabilities, this model sold thousands in the first month. Its bus, which became the S-100, was extensible enough that numerous vendors developed cards for the 18-slot system (such as serial cards and disk controllers), making it a useful computer.

Although there were computer systems designed before the Altair (some of which were also based on the Intel 8008 processor), the Altair was powerful enough to be useful (such as for running BASIC, one of the key languages being taught at the time). It is widely credited as the spark that began the personal computer revolution.

Within SIMH, you can run a standard Intel 8080-based Altair or derivative Altair systems using the Zilog Z80 or Intel 8086 processor. The simulated Altair also provides some additional capabilities, such as banked memory. The software kit for the Altair is one of the simplest ways to get Control Program for Microcomputers (CP/M) running. In the example shown in Listing 7, you start the Altair simulator (in this case, the Z80 processor variant), and then use the cpm2 script file to boot CP/M. This script uses two disks (an operating system disk and an application disk) to create the CP/M environment. With the CP/M environment created, you check out the files that are available, and then run Microsoft basic to interpret the Eliza program.

Listing 7. Demonstrating CP/M on the MITS Altair

```
$ mkdir cpm ; cd cpm
$ wget http://simh.trailing-edge.com/kits/psaltair.zip
$ unzip psaltair.zip
$ ../../../BIN/altairz80

Altair 8800 (Z80) simulator V3.8-1
sim> do cpm2

62K CP/M Version 2.2 (SIMH ALTAIR 8800, BIOS V1.17, 28-Apr-02)
A>dir
A: PIP      COM : LS       COM : XSUB     COM : STAT     COM
A: GO       COM : RSETSIMH MAC : SYSCOPY  COM : SHOWSEC  COM
A: DIF      COM : R        COM : W        COM : L80      COM
A: M80      COM : WM       HLP : WM       COM : CBIOSX   MAC
A: FORMAT   COM : SYSCPM2  SUB : DDTZ     COM : DSKBOOT  MAC
A: TSTART   COM : ED       COM : DDT      COM : EX8080   MAC
A: LOAD     COM : ASM      COM : LU       COM : MBASIC   COM
A: ELIZA    BAS : DUMP     COM : CREF80   COM : EXZ80    MAC
A: UNERA    COM : BOOT     COM : OTHELLO  COM : WORM     COM
A: LADDER   DAT : LADDER   COM : ZSID     COM : ZTRAN4   COM
A: SURVEY   MAC : CPMBOOT  COM : TSHOW    MAC : TSTART   MAC
A: TSTOP    MAC : UNERA    MAC : MOVER    MAC : EX8080   SUB
A: EXZ80    SUB : CCP      MAC : DSKBOOT  COM : USQ      COM
A: MC       SUB : MCC      SUB : BDOS     MAC : RSETSIMH COM
A: TSHOW    COM : TSTOP    COM : UNCR     COM : SURVEY   COM
A: EX8080   COM : EXZ80    COM : COPY     COM : SID      COM
A: BOOT     MAC : BOOTGEN  COM : LIB80    COM : DO       COM
A>
A>mbasic eliza.bas
BASIC-80 Rev. 5.21
[CP/M Version]
Copyright 1977-1981 (C) by Microsoft
Created: 28-Jul-81
32824 Bytes free
```

```
               **************************
                         ELIZA
                   CREATIVE COMPUTING
                 MORRISTOWN, NEW JERSEY

                  ADAPTED FOR IBM PC BY
                   PATRICIA DANIELSON AND PAUL HASHFIELD
                    BE SURE THAT THE CAPS LOCK IS ON

               PLEASE DON'T USE COMMAS OR PERIODS IN YOUR INPUTS

               *************************



HI! I'M ELIZA. WHAT'S YOUR PROBLEM?
? I'M IN LOVE WITH RETROCOMPUTING
DID YOU COME TO ME BECAUSE YOU ARE IN LOVE WITH RETROCOMPUTING
?
```

As you can see from the listing, the disk contains a large number of useful utilities, such as a CP/M assembler (ASM.COM), a line editor (ED.COM), a pair of debuggers (SID.COM for the Intel 8080 processor and ZID.COM for the Z80 processor), and even a screen editor (WM.COM).

Other emulation projects

Although SIMH is a great simulator for older computing systems, it's one in a growing family of simulators and emulators. Examples of other historically interesting emulators include Hercules, which emulates IBM mainframe computers (such as the System/370, IBM System/390®, and IBM zSeries®) on commodity hardware.

Some emulators focus not on reviving historical hardware but instead on bringing video games to life for which hardware may no longer exist. One of the most interesting is the Multi-Arcade Machine Emulator (MAME). This emulator provides system emulation for a large number of vintage video game hardware (including arcade machines) and therefore provides an emulation of older processors and the hardware environments that were built around them (data buses, storage devices, audio and video hardware, and so on). Today, the MAME project provides emulation for more than 4500 unique games. MAME is also the core of the Multi-Emulator-Super-System (MESS), which emulates almost 500 unique consoles, computers, and calculators.

You can also find simulators for specialized hardware, such as the Apollo Guidance Computer (AGC) used in the Apollo lunar missions. A similar effort simulates the launch vehicle digital computer (LVDC) that managed the firing of the rocket engines during ascent into orbit. Although the LVDC was a computer designed from transistors, the AGC was the first computer designed with integrated circuits. Both had custom instruction sets, with programs designed in their raw machine code.

See Resources for more information on these and other emulators.

Going further

There's something really fascinating about retro-computing. Everything we have today is derived from older computing systems, many of which no longer have functioning hardware we can use. Thankfully, the SIMH project brings this hardware (and operating systems and applications) back to life so that they can be enjoyed by a new generation.

Resources

- Konrad Zuse's "Z" machines are well documented at the Technical University of Berlin. You can read more about Zuse's Z3, Z4, and the first language, called Plankalkül.
- This Wikipedia article on the evolution of the computer provides a fascinating history of the silent race to build the computer and the various schemes that were defined (from Konrad Zuse's early Z machines to the secret British Colossus computers used to break German codes and beyond).
- As of 2010, the inventor of the first digital computer to automate computation has been a battle ground. Although most of us think of ENIAC and the University of Pennsylvania, there was prior work that has had little mention. In the late 1930s, John Atanasoff and Clifford Berry began building the first computer, now called *ABC.*. A new book has been published that looks at this early history from Jane Smiley, who was covered recently in this article from *Wired*: Pulitzer Prize-Winning Novelist Tells the Tale of the World's First Computer. In addition to covering some of the history, Smiley uncovers some of the political, psychological, and corporate drama behind this invention.
- In 1971, Intel released the first commercially available microprocessor in a chip called the 4004. The 4004 processor was a 4-bit CPU that could execute 92,000 instructions per second. Its successor was the 4040 processor (released in 1974), which had an expanded instruction set, program memory, register set, and stack.
- The Computer History Simulation Project implements simulators for a large number of computing systems. You can see the list of more than 30 significant systems at the project's website.
- In the document Writing a Simulator for the SIMH System, you can learn more about the simulator architecture as well as some of the details of the various simulated machines. You can also read through this presentation from Bob Supnik titled SIMH: Forward into the Past for additional information, including how historical systems are understood for simulation.
- Peter Schorn maintains the Altair emulator for SIMH and a website that has a tremendous amount of information and source for the Altair. At his site, you can find a large list of operating systems for the Altair, alternative programming languages, and other applications.
- Hercules is an open source emulator for a variety of IBM mainframes (including the System/370, ESA/390, and z/Architecture). Hercules can be used on a variety of host operating systems, including Linux® and FreeBSD.
- This article discussed other emulators that focus on specialized electronics, such as the AGC. The Virtual AGC (and related emulators) preserves another interesting aspect of computing systems. You can learn more about the AGC and its history in Journey to the Moon: The History of the Apollo Guidance Computer.
- Game emulators are an interesting area of development. Although the emulators may be freely available, the ROMs that provide the games are still owned by the license holders. Make sure to check out the relevant licenses before downloading or using downloaded ROMs. In many cases, these older games can be purchased to run on newer hardware. The Multi-Arcade Machine Emulator is one of the most interesting projects, in addition to the Multi-Emulator Super-System (which uses MAME at its core). You can also find a list of video game console emulators and a larger list of emulators at Wikipedia.

- The SIMH User's Guide presents a large amount of information about the SIMH implementation as well as the commands that can be used in the simulator framework. You can also find more information on other emulator projects (including computing system restoration) at the Computer History and Simulation Links.
- Check out developerWorks blogs and get involved in the developerWorks community.

About the author

M. Tim Jones

M. Tim Jones is an embedded firmware architect and the author of *Artificial Intelligence: A Systems Approach, GNU/Linux Application Programming* (now in its second edition), *AI Application Programming* (in its second edition), and *BSD Sockets Programming from a Multilanguage Perspective.* His engineering background ranges from the development of kernels for geosynchronous spacecraft to embedded systems architecture and networking protocols development. Tim is a software architect and author in Longmont, Colorado.