Preserving Computing's Past: Restoration and Simulation

Maxwell M. Burnet
Robert M. Supnik

Restoration and simulation are two techniques for preserving computing systems of historical interest. In computer restoration, historical systems are returned to working condition through repair of broken electrical and mechanical subsystems, if necessary substituting current parts for the original ones. In computer simulation, historical systems are re-created as software programs on current computer systems. In each case, the operating environment of the original system is presented to a modern user for inspection or analysis. This differs with computer conservation, which preserves historical systems in their current state, usually one of disrepair. The authors argue that an understanding of computing's past is vital to understanding its future, and thus that restoration, rather than just conservation, of historic systems is an important activity for computer technologists.

The Computing Past

The continuous improvements in computing technology cause the rapid obsolescence of computer systems, architectures, media, and devices. Since old computing systems are rarely perceived to have any value, the danger of losing portions of the computing record is significant. When a computing architecture becomes extinct, its software, data, and written and oral records often disappear with it.

Older computer systems embody major investments in software, the value of which may persist long after the systems have lost their technical relevancy. For example, the PDP-11 computer has not been a leading-edge architecture since the introduction of 32-bit systems in the late 1970s and has not received a new hardware implementation since 1984. Nonetheless, PDP-11 systems continue to be used worldwide, particularly in real-time and control applications. The unavailability of suitable replacements of worn-out original parts is a serious issue for PDP-11 systems still in use.

Another area of potential loss is data. In recent years, archival storage media have undergone rapid technologic evolution, and the industry standards of computing's first 30 years, such as 0.5-inch magnetic tape, are now antiques. Salvaging data from original media is an industry-wide problem and has generated a small cottage industry of specialists in data recovery. This problem will only proliferate, as transitions in media types accelerate. Ten years from now, the large-diameter optical disks used for today's archives will look as quaint as DECtape and magnetic tape storage systems do to current computer users.

Finally, the disappearance of older equipment typically entails loss of information: not only design sketches, blueprints, and documentation but also the folklore about these systems. The absence of systematic archiving, as well as the absence of a perceived value of the archived data, causes continual information decay about design and operational details.

This paper describes two techniques for preserving computing systems of historical interest. The first section of the paper discusses the restoration of old computers to working order. It also includes a description of the Australian Museum collection and the process of restoring a particular PDP-11 minicomputer. The second section discusses the simulation of old computers on modern systems. It describes a simulation framework called SIM, which has been used to implement simulators for the PDP-8, PDP-11, PDP-4/7/9/15, and Nova minicomputers.

Restoring Old Computers

Since the computer became a mass-produced item in the late 1960s, its typical life cycle has consisted of initial installation, rental or depreciation for about five years, retention and use for a few more years (just in case), and then retirement and a trip to the refuse dump. There is only a brief window of opportunity to collect old computers at the end of their working life. Once that window is closed, the computers are gone forever.

The Australian Museum Collection

In Sydney, Australia, this window of opportunity first became apparent in 1971, when the early PDP systems reached the ends of their life cycles. Digital's Australian subsidiary began collecting systems by a creative program of trade-ins for new equipment.[1] It was especially urgent to obtain examples of the 12-bit, 18-bit, and 36-bit PDP series, as they were relatively few in number. Table 1 lists the percentage of available units that have been collected. The status of each is given as

- Static—can never be made to work for various reasons
- Restorable—could be made to work with enough care, patience, time, and effort
- Working—running its operating system the last time it was turned on

Once a representative sample of the early PDP systems had been collected, the urgency abated. Hundreds of PDP-11 and VAX systems were then brought to Australia; the window of opportunity for collecting them is still open.

The collection has grown significantly during the last 25 years. At the present time, we have in Sydney a comprehensive collection of most early Digital machines, including hardware, manuals, software, and spares (see Table 2). The collection is catalogued in a 6,000-line database that resides, appropriately, on a MicroVAX I computer, running the first version of the MicroVMS operating system. Figure 1 shows an example from the collection, a PDP-8/E computer system with peripheral equipment.

The goals of the collection are varied and are summarized in Table 3. Apart from the academic challenge of keeping all old data media running, there is the responsibility to ensure that they can be kept alive and available. The extensive variety of media types offered by Digital alone in only 30 years is summarized in Table 4. The evolving status of the collection has been reported at several Australian DECUS Symposia.[2,3] The restoration of the Australian collection will probably ensure a retirement job for the curator for the next 30 years!

General Issues in Restoration

Restoration is a painstaking and time-consuming process. The goal of restoration is to return a system to a state where it will reliably run a major operating system and offer as many media conversion facilities of the vintage as possible. Fortunately, computers do not deteriorate greatly in storage, provided the storage area is dry. (One item that does decay dramatically is the black foam used to line side panels and to separate

Table 1
Early Digital CPUs in Australia

Model Name	Number Brought to Australia	Number in Museum Collection	Condition
PDP-5	1	1	Restorable
PDP-6	1	1	Some items
PDP-7	1	1	Static
PDP-8	28	3	Working
PDP-8/S	20	2	Static
LINC-8	2	2	Restorable
PDP-9	7	1	Restorable
PDP-10	8	1	Some items
PDP-12	2	2	Restorable
PDP-8/I	24	2	Restorable
PDP-8/L	21	2	Restorable
PDP-15	10	1	Static
PDP-8/E	90	4	Working

Table 2
The Digital Australian Collection (chronological order)

Year	Item	Description	Status
1958	138	A/D converter	Static
1960	ASR-33	Teletype reader/punch, 110 baud	Working
1962	KSR-35	Heavy-duty Teletype	Working
1963	PDP-6	Modules of first Digital computer in Australia	Parts
1963	PDP-5	First minicomputer in Australia	Working
1967	PDP-7	Third Digital computer in Australia	Static
1965	PDP-8	Classic, table-top model	Working
1965	PDP-8	Cabinet model	Restorable
1965	PDP-8	Typesetting system	Static
1965	PDP-8	Cabinet model, first in New Zealand	Restorable
1965	COPE-45	Remote batch (OEM PDP-8)	Restorable
1966	PDP-9	18-bit computer	Static
1966	KA10	Console of PDP-10 mainframe	Static
1966	Linc-8	Early medical computer	Working
1967	PDP-8/S	Serial, under $10,000, CPU	Static
1967	PDP-8/S	Serial computer	Static
1967	DF32	Digital's first disk, 1/16 Mb	Static
1967	PDP-9/L	Last transistor logic, 18-bit	Static
1968	PDP-8/I	Digital's first IC minicomputer	Working
1968	PDP-8/L	OEM version of PDP-8/I	Static
1969	PDP-12	Laboratory computer	Working
1969	PDP-12	Laboratory computer	Static
1969	PDP-15	Last of 18-bit family	Static
1969	KI10	Console of DECsystem-10	Static
1970	PDP-8/E	Pinnacle of PDP-8 development	Working
1970	PDP-8/E	Full LAB 8 configuration	Working
1970	PDP-11/20	The first PDP-11	Working
1970	CR11	Card reader, 285 cpm	Working
1971	PDP-8/F	Small PDP-8/E	Working
1971	VT05	Digital's first video terminal	Working
1971	LA30P	Digital's first hard-copy terminal	Working
1971	PDP-11/45	Last PDP-11	Static
1972	GT40	Graphics workstation	Broken
1972	PDP-11/10	Small PDP-11	Static
1973	PDP-11E10	First packaged system	Working
1973	PDP-11/35	Mid-range PDP-11	Static
1973	PDP-8/A	Last non-chip PDP-8	Working
1974	PDP-11/40	Mid-range, end-user PDP-11	Restorable
1975	VT50	Video terminal	Working
1975	LA36	DECwriter II printer	Working
1975	DS310	Desk-based commercial system	Working
1975	PDP-11/70	Largest PDP-11	Restorable
1976	PDP-11/34	Mid-range PDP-11	Working
1977	PRS01	Portable paper tape reader	Working
1977	LS120	DECwriter printer	Working
1977	WS78	Word processor, 8-inch floppy disks	Working
1978	LA120	DECwriter III printer, 180 cps	Working
1978	VAX-11/780	Original unit of 1 VAX-11/780	Restorable

continued on next page

Table 2 (continued)

Year	Item	Description	Status
1979	VT100	Famous video terminal	Working
1980	MINC	LSI-11 lab unit with RT-11	Working
1980	VAX-11/750	Mid-range VAX system	Restorable
1980	PDT-150	Table-top LSI-11 with RX01 drives	Working
1981	GIGI	Low-cost terminal for schools	Working
1982	VT125	Video terminal with graphics	Working
1982	WS278	DECmate I word processor	Restorable
1982	VAX-11/730	Low-performance VAX system	Working
1982	LA12	Portable hard-copy terminal	Static
1982	LQP03	Letter-quality printer	Working
1982	DECmate II	Word processor on mobile stand	Working
1982	DECmate II	Word processor	Working
1982	Rainbow	Personal computer	Working
1982	PRO350	Professional PC	Working
1983	VT241	Graphics color terminal	Working
1983	MicroVAX I	Smallest VAX .3 VUP	Working
1983	VAX-11/725	Lowest cabinet VAX .3 VUP	Working
1984	LN03	Laser printer	Working
1985	MicroVAX II	Famous MicroVAX II	Working
1986	VAXmate	286-based PC with RX33 drive	Working
1986	DECmate III	Small word processor	Working
1987	MicroVAX III	3-VUP MicroVAX II system	Working
1987	VAX 8250	Dual VAX CPU, BI-based	Restorable
1989	VAX 9000	Chip set	Static
1990	DS3100	Mips UNIX workstation	Restorable

ribbon cables. After 20 years, it turns into a sticky, gooey mess. It should be removed as soon as possible; otherwise, it falls into the modules and backplane. Replacing it with a modern equivalent can be done but is not essential.)

The first step in restoration is to collect hardware, software, and documentation.

- Collect the hardware, if possible two or ideally three items of each example. This provides a system to work on and a spare, as well as the ability to make comparisons between units.
- Collect diagnostic and operating software on original bootstrap media. Sources are very useful, particularly for diagnostics.
- Collect hardware manuals and schematics.

There is a network of enthusiasts around the world who can help at this stage.

Once the "ingredients" have been collected, the steps needed to restore a 1960s or 1970s vintage machine are as follows:

- Inspect the hardware for physical safety, particularly the heavy drawers and slide mechanisms.
- Physically assemble the hardware, checking module allocations, cabling, etc.
- Carefully inspect the power system, high-voltage sources can kill. Although most of the power wiring material appears to stand the test of time, the early machines often had rather thin coverings on terminals. Safety-first is a principal criterion in restoration, since someday nontechnical people may open the back door.
- Assemble a minimal system of CPU, memory, and console switch register for initial tests.
- Power up the computer, checking supply voltages, fans, and front console for signs of life.
- Use simple routines at the switch register to check for elementary operation.
- Fit a serial line unit so that a VT or a Teletype console can be used.
- Get the keyboard echoing to the screen or printer with simple routines.
- If they are available, run the internal tests of the read-only memory (ROM).

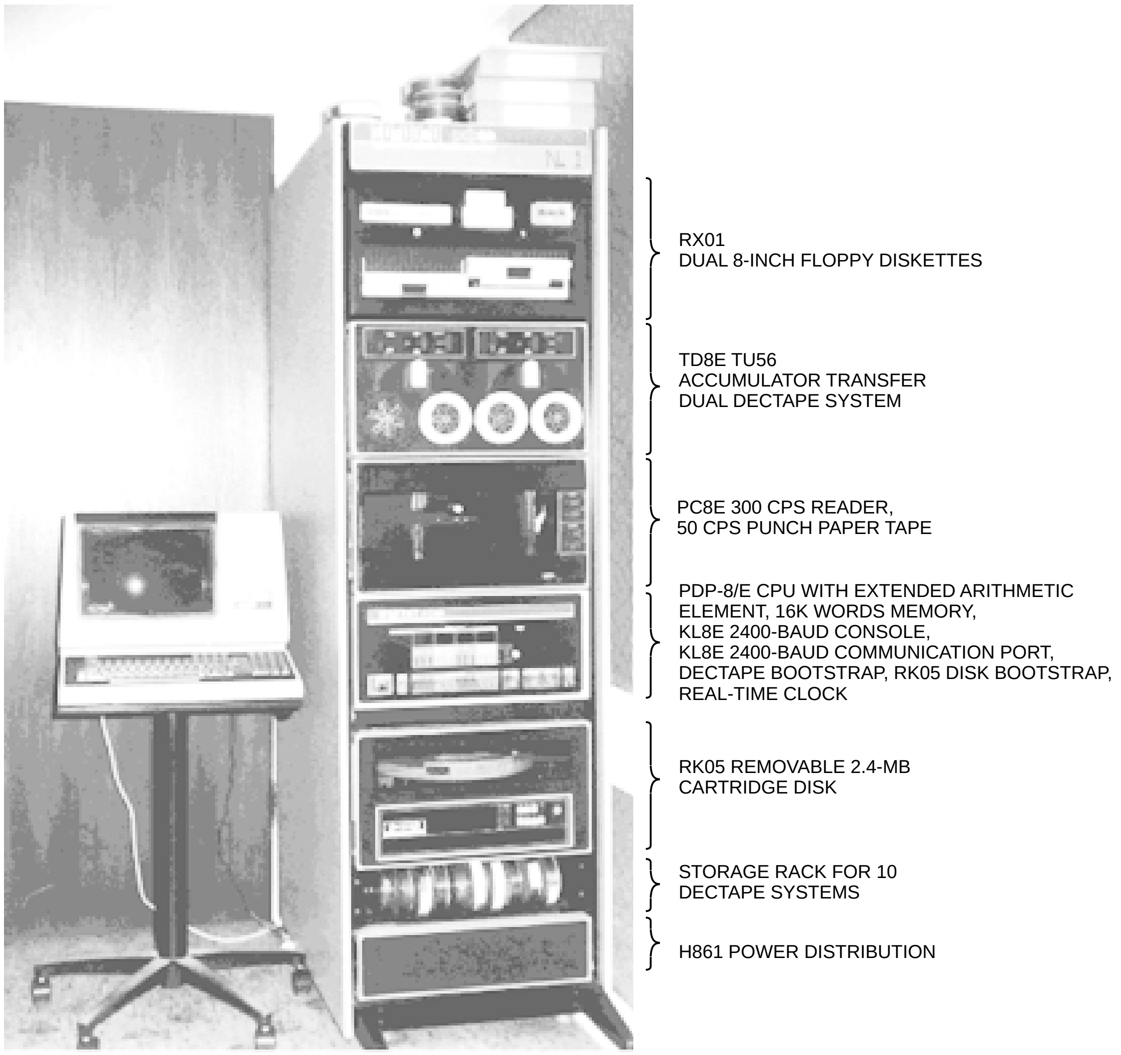

Figure 1
PDP-8/E Computer System

Conventional wisdom would now advise that all the diagnostic routines be run. However, diagnostics were (philosophically) always used to find bugs in a previously good machine; they are too complex when huge chunks of the machine might still be missing. The most practical next step is to get mass storage on-line. Depending on the manufacturer, the target device may be a floppy disk drive, a cartridge hard disk drive, or some form of magnetic tape. With a working mass storage device and a bootstrap routine, it becomes possible to boot a simple operating system (like OS/8 or RT-11 for Digital's systems). This quickly shows whether the machine is working or not.

If a mass storage device is not available, the next best thing is paper tape. This can be either the system's rack-mounted reader and punch or the paper tape reader on an ASR33 or ASR35 console. The reliability is questionable, however, and the procedure is tedious. Many diagnostics were on paper tape, but usually the quickest test is to load a complete paper system (such as FOCAL for Digital's systems). If the diagnostics run, the system is probably functional.

Once the CPU, console, and memory are verified, additional peripherals can be added, one at a time. It pays to take the time and effort to research bus addresses, interrupt vectors, power supply loading, and module placement, and to keep a log book with configuration diagrams and results. In general, if the configuration rules are followed, the items will work. There are few electronic failures, even in 20- or 30-year-old modules. When a problem arises, it is usually address vector strapping, physical damage, or missing cables. Corrosion of board contacts can be a problem; they should be cleaned with a clean cloth or cardboard

Table 3
Goals of the Australian Digital Museum

To preserve one of each model of Digital's computers
To keep each major Digital operating system working
To have a working unit of each Digital terminal, console, and PC
To provide conversion and archival facilities for old media
To preserve significant Digital literature and manuals
To preserve a VAX-11/780 computer as the original unit of 1 VUP
To disseminate instructive and educational material
To educate and amuse our staff, our customers, and the public
To support the DECUS NOP (nostalgic obsolete product) Special Interest Group
To preserve spares, tools, test gear, and documentation to keep the collection working
To preserve and protect these treasures for future generations

(for example, a business card), not with a pencil eraser, which leaves residues. Silicon components appear to be very stable and a tribute to the conservative design principles of early computer engineers.

The main components that seem to age are power supply capacitors, fans, and lights. The filter capacitors across the high-voltage sources can short, and reference electrolytic capacitors in power supply regulators can dry out. Although the large capacitors in power supply RC filters have proven to be reliable, some restorers replace them as a matter of course for safety reasons. Small rotary fans may seize if they have logged many hours. Incandescent panel lamps are always failing and can be replaced by modern light-emitting diodes (LEDs) if required. The irony is that the panel lamps are needed only during initial checkout; once the operating system is running, they are rarely used.

Once restored, are old units reliable? Experience proves that they are. A classic PDP-8 system restored in 1988 still turns on happily (untouched) eight years later. A fully configured PDP-8/E system is still working four years after restoration.

Restoring a Minicomputer: A Case Study

An ongoing project is the restoration of a large, UNIBUS-based PDP-11 system with many UNIBUS peripherals attached to it. The project was started using the original PDP-11/20 CPU. Since many PDP-11 peripherals were designed long after the PDP-11/20 CPU, it could not cope with single-board direct memory access (DMA) devices, metal-oxide semiconductor (MOS) memory, and other later inventions. The project refocused on the mid-range PDP-11/34, which in retrospect has proved wise. The PDP-11/34 supports MOS memory, has an LED and push-button console, and represents a mature implementation of the PDP-11 instruction set. It has an optional cache, battery backup, floating-point operation, and the extended instruction set (EIS).

Table 4
Digital Data Media from 1960 to 1996

Paper tape
80-column punched and mark sense cards
7-track, half-inch magnetic tape
9-track, half-inch magnetic tape
DECtape and LINCtape systems
Audiocassette
DECtape II cartridge (TU58)
CompacTape (TK50, etc.)
Quarter-inch cartridge tape
Digital audio tape
8-inch floppy disk
5.25-inch floppy disk
3.5-inch floppy disk
RK05 removable disk
RK06, RK07 removable disk
RL01, RL02 removable disk
RP01…RP06 removable disk
RM03, RM05 removable disk
RC25 removable disk

The current configuration occupies three large cabinets in what used to be the dining room of Max Burnet's house. The virtues of the UNIBUS are many; in particular, it allows modular connection of I/O devices and other components. However, I/O devices of the era often weigh 100 pounds and are mounted in 10-inch drawers; their sheer physical size and weight are disincentives to reconfiguration.

The project currently uses the RT-11 operating system because of its simplicity and extensive device drivers. Eventually, it may be possible to run the RSX-11M and the RSTS/E systems, but there is little to gain from a media conversion point of view, because RT-11 includes utilities for dealing with foreign file formats.

The main difficulties encountered have been associated with the power supply: the DC low signal threads its way through every peripheral. The absence of UNIBUS grant continuity cards can create havoc. Since this PDP-11 system is very large, it is straining the design rules concerning floating vectors, current loading, and bus loads.

The CPU and memory are relatively easy to check out. Due to the versatility of the UNIBUS, however, checking out the I/O system is very laborious. Starting with programmed I/O tests works best, followed by interrupt tests, and finally DMA or nonprocessor reference (NPR) tests. Experience shows that tests need to be rerun whenever a new peripheral is added.

The system currently runs the RT-11 version 5.04 operating system on a configuration comprising

- RT-11/34 CPU with real-time clock and bootstraps
- 256 kilobits of MOS memory
- RX01 and RX02 floppy disks
- Dual RL02 disks
- TU56 dual DECtape storage system
- TU58 DECtape II storage system
- Serial line units for console and serial printer
- CM11 mark sense and CR11 punched card reader
- TU60 cassette
- PC11 paper tape reader and punch

Although the following peripherals are available, they await installation time and effort:

- LPS-40 analog-to-digital (A/D) converter
- TU10 magnetic tape
- TSV03 magnetic tape
- Cache and commercial instruction set
- Battery backup kit

The eventual goal is to keep "the last great (UNIBUS) PDP-11" running with almost every UNIBUS peripheral ever made.[4] Time will tell.

Simulating Old Computers

A simulator is a computer program operating on one computer system (known as the host system) which mimics the behavior of another computer system (known as the target system). The simulator's data is the state of the target computer system—registers, memory, timed events, and so on. The simulator operates on presented state and transforms it, usually by sequential evaluation, in the same manner as would the target computer system.

Simulators typically consist of an execution engine, which performs the state transformations; a simple timed-event mechanism, which supports deferred and asynchronous events such as I/O completions; and a control panel, which provides user access to simulated state. The execution engine is responsible for decoding instructions in simulated memory and performing the specified alterations of simulated machine state. The execution engine keeps track of simulated time in arbitrary units, which may be precise representations of the execution time of the target system, or simple representations of advancing time, such as the number of instructions executed. The event mechanism provides a way to schedule events, such as I/O completion, for later evaluation. It can also implement other time-based mechanisms such as keyboard polling. Finally, the control panel provides access to simulated state as well as basic control commands such as start and stop. It may also provide more elaborate facilities to support performance instrumentation or debugging.

Historically, simulators have been used for many purposes, including the following:

- Design of new systems. The simulator mimics the behavior of a future chip or computer system and is used to understand and debug the behavior of the proposed design. For example, prior to fabrication, all modern microprocessors are extensively simulated, first as abstract performance models and then at increasing levels of detail. [5–9]
- Debugging for embedded systems. If the simulator contains facilities for program debugging, it becomes a useful tool for debugging programs that run in highly constrained environments such as embedded systems. Simulators can capture more state and provide a wider range of facilities than in situ debuggers. For example, simulators can implement program counter (PC) change queues, data access breakpoints, or precise traps on errors.
- Replicable event tracing. Most simulators are fully deterministic. Asynchronous events are scheduled based on simple, nonrandom algorithms, such as fixed time-out or calculated seek time. As a result, simulators allow for straightforward replication or playback of complicated sequences, removing the randomness factor that often plagues the debugging of asynchronous software on real systems.
- Preservation of past software. Simulators can provide migration assistance in the transition from older to newer architectures. Many transitional computer systems have provided simulators for older architectures, typically at the microcode level, to assist customers and developers in preserving their investments in the previous architecture. Examples include the early IBM System/360 series, which had models that simulated the 1401, 1410, 7070, and 7090 families, and the early Digital VAX systems, which included a PDP-11 compatibility mode.[10,11]

Simulation Levels

Simulators can be written at various levels of detail and thus various levels of fidelity to the target system. Three common levels of simulation are register transfer level (RTL), instruction, and software specific.

An RTL simulator attempts to mimic the major hardware blocks of the target system and to implement its actual logic equations. The goal is absolute fidelity, the test of which is that no piece of software running on the simulator should behave differently than it would on the target hardware. In practice, such perfect mimicry is difficult to achieve, as it requires a painstaking re-creation of timing detail (for example, the actual acceleration curve of a DECtape storage system) and access to implementation documentation that has often vanished. Nonetheless, some simulators have achieved results very close to this goal: MIMIC, a DECsystem-10 simulator written at Applied Data Research, was able to run CPU- and device-specific diagnostics. (As testimony to the vulnerability of computing's past, all machine-readable copies of the MIMIC sources appear to have been lost.)

An instruction simulator steps back from the RTL level and tries to simulate at the functional or the behavioral level. System elements are treated as functions that transform state according to the abstract definitions of the system architecture, rather than as logic blocks that transform state based on implementation equations. Instruction simulators sacrifice absolute fidelity to the idiosyncrasies of a particular implementation and focus on the intentions of the architecture specification. As a result, instruction simulators can usually run systems software and applications but can rarely fool diagnostics.

Finally, a software-specific simulation further abstracts the functions of the target system to only those needed by a particular piece of target system software. For example, the OS/8 operating system on the PDP-8 computer does not use program interrupts; a simulator aimed at running only the OS/8 operating system would not need to implement interrupts or even queued events. A recent PDP-11 simulator designed to run the 2.9 BSD UNIX operating system abstracted parts of the PDP-11 system's interrupt model and could not run other PDP-11 operating systems.[12]

Simulating Minicomputers: A Case Study

SIM is a portable instruction-level minicomputer simulator implemented in C. Its objectives are to facilitate the study and use of historic computer architectures by making simulated implementations and historic software available to anyone who has a 32-bit computer. It supports the following target architectures

- PDP-8
- PDP-11
- Nova
- 18-bit PDP series (PDP-4, PDP-7, PDP-9, PDP-15)

and has been successfully ported to the VAX VMS, the Alpha OpenVMS, the Digital UNIX, and the Linux architectures. Ports to the Windows NT and the Windows 95 architectures and to an IBM 1401 simulator are under way.

General Design Considerations The design of an instruction-level simulator is not technically complicated; indeed, simulating a PDP-8 system is a common problem in undergraduate computer science courses. SIM follows the processor-memory-switch (PMS) structure proposed by Bell and Newell and implemented in MIMIC and countless other simulators since.[10,13] The simulated system is a collection of devices, one of which has special properties (the CPU). Each device has state (registers) and one or more units. Each unit has state and fixed- or variable-sized storage. In the CPU device, the storage is main memory. In an I/O device, the storage is the device media. The CPU is distinguished from other devices by having the master routine for instruction execution. This routine is responsible for the sequential evaluation of instructions and for the state transformations that represent simulated execution. The CPU also provides a few systemwide routines, such as symbolic disassembly and input and a binary loader.

The devices interface to a control panel that provides access to simulated state and control over execution. The available commands in SIM are listed in Table 5.

The control panel also includes routines that are needed by most simulators, such as event queue maintenance and character-by-character terminal I/O. Different simulators need not use the same time base, but all the SIM-based implementations to date use the number of instructions executed as the time base.

Note that the control panel provides for starting simulation, but termination is determined entirely by the simulated CPU. By convention, the CPU returns control to the control panel under the following conditions:

1. If a HALT instruction is executed
2. If a fatal exception is detected
3. If a fatal I/O error is detected
4. If a special character is typed at the controlling terminal

Likewise, the control panel does not implement any debugging facilities beyond state examination and modification and instruction stepping. To facilitate debugging with operating systems, CPUs provide a simple instruction breakpoint capability and a one-level PC trace facility.

Implementation The implementation of a particular simulator begins with collecting reference manuals, maintenance manuals, design documents, folklore, and prior simulator implementations for the target system. This is nontrivial. In the early days of computing, companies did not systematically collect and archive design documentation. In addition, collected material is subject to information decay, as noted

Table 5
Commands Available in SIM

Command	Definition
attach <unit> <file>	Associate file with unit's media.
detach <unit> | ALL	Disassociate unit's (all units) media from any file.
reset <device> | ALL	Reset device (all devices).
load <file>	Load binary program from file.
boot <unit>	Reset all devices and bootstrap from unit.
run {<new PC>}	Reset all devices and resume execution at the current PC {or new PC}.
go {<new PC>}	Resume execution at the current PC {or new PC}.
cont	Resume execution at the current PC.
step {<number>}	Execute one instruction {or number instructions}.
examine <list>	Display contents of list of memory locations or registers.
iexamine <list>	Display contents of list of memory locations or registers and allow interactive modification.
deposit <list> <value>	Store value in list of memory locations or registers.
ideposit <list>	Interactively modify list of memory locations or registers.
save <file>	Save simulator state in file.
restore <file>	Restore simulator state from file.
show queue	Display the simulator's event queue.
show configuration	Display the simulator's configuration.
show time	Display the simulated time counter.
show <device>	Show device's configuration options.
set <device> <option>	Set a device configuration option.
help	Display a terse help message.
exit | quit | bye	Leave the simulator.

earlier. Lastly, the material is likely to be contradictory, embodying differing revisions or versions of the architecture, as well as errors that have crept in during the documentation process.

For Digital's 12-bit and 16-bit minicomputers, the typical hierarchy of documentation was the following:

- Processor Handbook. Providing an all-inclusive summary of the instruction set architecture, peripherals, bus interface, and software, these paperback-size books are the most common form of system documentation but also the least accurate.
- Subsystem Reference Manual. As the programmer's reference manual for a particular subsystem, such as the CPU or the disk drive, these manuals describe the registers and functions accurately but omit maintenance-level features and other fine points.
- Subsystem Maintenance Manual. As the maintenance engineer's manual for a particular subsystem, these manuals describe the registers and functions at the hardware implementation level, often including substantial abstracts from the print set. Because of the level of detail, the maintenance manuals have proven to be the most useful references for simulator implementation.
- Design documents. For systems that do not have very large-scale integration (VLSI), the only extant design documents are the logic prints and the binary microcode ROM listings. The prints are essential for RTL simulation: they provide the only documentation of implementation quirks. For VLSI systems, there are chip-level design specifications as well as human-readable microprogram listings.
- Folklore. During the useful lifetime of a system, its users exchange information and create an informal record, both written and verbal, of shared experiences (folklore) regarding the fine points of operations, hardware/software interfaces, system "personality," and other factors. Folklore is subject to rapid information decay, particularly once the target system becomes obsolete.
- Prior implementations. Prior simulator implementations can provide useful information, but it must be used cautiously. Unless the prior implementation is an RTL model, it embodies simplifications and abstractions that are not explicitly documented. The MIMIC sources (which are fragmentary and available only on paper) proved trustworthy, but others did not: for example, the 1970s PDP-11 simulator in the DECUS archives is highly misleading about interrupts, condition codes, and other details.

An important consideration is that much of the documentation, all the folklore, and most working systems are in the hands of individual collectors. The Internet plays a vital role in locating material held by enthusiasts, through news conferences such as alt.folklore.computers, alt.sys.pdp8, alt.sys.pdp11, and comp.emulators.misc, and more recently, through World Wide Web sites devoted to historic systems.[14–16] The sources for each simulator in SIM are listed in Table 6.

The last step in implementation is collecting software to run on the simulator. Software collection immediately raises the problem of media translation. Software for historic systems resides on paper tapes, DECtape storage systems, 200/556/800 bits-per-inch magnetic tapes, disk cartridges, 8-inch floppy disks, and so on. Few if any modern systems have these peripherals; and few if any historic systems have modern network interconnects. Thus, media translation usually entails linking a working version of the target system to a modern system by means of a serial line. KERMIT or some other simple protocol allows for a byte-by-byte network copy from the original media to a file on a modern system.

Once the software has been located and moved to a file, the next issue is sources. Without sources, diagnostics and other test programs are useless; detected errors cannot be traced back to causes without manual decode of the binary program. The absence of sources was a principal reason for including symbolic disassembly and input in SIM.

The final issue in software is licensing. Even though the target systems are obsolete and often no longer manufactured, the operating system software may be protected by copyrights and licenses. Most PDP-8 software is in the public domain; however, the PDP-11 and Nova operating systems are still licensed, as are all versions of UNIX. Corporate licensing policies rarely accommodate hobbyists; this limits operating system distribution to legitimate (that is, business) users. Table 7 lists the software found for each simulator in SIM.

Debug The debug path for a simulator depends on the available software. Ideally, the simulator would be debugged with the same software tests used to debug the target hardware, but this software is rarely archived. Diagnostics can provide low-level checking, but diagnostics typically check for broken parts in a correct implementation, rather than an incorrect implementation. Even when diagnostics do check architecture rather than implementation (as in the basic instruction diagnostics on the PDP-11 system), the absence of sources limits their utility. Consequently, the simulators were debugged mostly with simple hand tests and then with the operating systems.

Operating systems are both exacting and imprecise tests of implementation correctness. Unless an operating system takes a deliberately restrictive view of hardware (for example, OS/8 does not use the PDP-8 interrupt system, and RT-11 does not use any

Table 6
Sources for Simulators in SIM

Architecture	Documents	Location
PDP-8	Minicomputer Handbook	Private collection
	Reference manuals	Digital archive
	Maintenance manuals	Digital Australia collection
	Print sets	Digital Australia collection
	Prior implementations	Public archive[17]
		Public archive[18]
		MIMIC, private collection
PDP-11	Minicomputer Handbook	Private collection
	Reference manuals	Digital archive
	Maintenance manuals	Digital Australia collection
	Chip specifications	Private collection
	Microcode listings	Private collection
	Prior implementations	Public archive[19]
		MIMIC, private collection
Nova	System Reference Manual	Private collection
	Reference manuals	Data General archive
	Maintenance manuals	Private collection
	Prior implementations	MIMIC, private collection
18-bit PDP	Reference manuals	Digital archive
	Maintenance manuals	Digital archive
	Print sets	Digital archive

Table 7
Software for Simulators in SIM

Architecture	Software	Location
PDP-8	Basic instruction tests 1 and 2	Digital Australia collection
	Memory management test	Digital Australia collection
	FOCAL69	Digital Australia collection
	OS/8 system disk	Public archive[18]
PDP-11	RT-11	Transcribed from real system
	RSX-11M	Transcribed from real system
	RSTS/E	Transcribed from real system
	UNIX V5, V6, V7, 2.9 BSD	PDP UNIX Preservation Society (PUPS) archive[20]
	2.11 BSD	Private collection
Nova	RDOS	Private collection
18-bit PDP	No software to date	

optional PDP-11 instructions), the operating system will be sensitive to every error in implementation. For example, Digital's second-generation PDP-11 systems—the PDP-11/05, 11/40, and 11/45—were debugged with DOS-11 and RSTS after diagnostics failed to detect certain subtle implementation errors. Unfortunately, in an operating system, the distance in time and space between the error and the symptom may be enormous, and the traceable path may be lengthy and complicated. Artifacts in the software can also complicate debug: the OS/8 disk image on the Internet contains a copy of BASIC that is broken.

Results SIM implements four minicomputer architectures: PDP-8, PDP-11, Nova, and 18-bit PDP. Each simulator includes a particular CPU; basic peripherals such as terminal, paper tape, clock, and printer; and a selection of mass storage peripherals (see Table 8).

The PDP-8 simulator has run the FOCAL69 and the OS/8 operating systems. The PDP-11 simulator has run the following operating systems: RT-11 V4 and V5; RSX-11M V4; RSTS/E V8; UNIX V5, V6, and V7; and BSD V2.9 and V2.11. The Nova simulator has run the RDOS V7.5 operating system. No system software for the 18-bit PDP systems has been found. The simulators were exercised on an AlphaStation 3000/600 workstation (approximately 120 SPECint92); the performance is given in Table 9.

Figures 2, 3, and 4 show screen shots from the various simulators running their principal operating systems.

In Defense of Computing's History

As professional engineers who have been lucky enough to witness the computer revolution, the authors believe that the industry has a duty to keep early machines alive. There are practical reasons, such as preservation of software and data; beyond that, there is an obligation to future generations. In 100 years, the systems from computing's early history will appear to be absolute dinosaurs of the past. Yet their educational and sociological value will be considerable. A computer is a machine with a soul, and it must be kept alive with its operating environment to show its abilities and the contemporary state of the art.

Acknowledgments

Max Burnet: I would like to thank Digital Equipment Corporation Australia Pty Ltd for tolerating my eccentricity and for supporting the Australian Digital museum collection. Also the DECUS Australia NOP (nostalgic obsolete product) SIG members for help, encouragement, knowledge, good humor, and camaraderie on the last Wednesday of the month. My thanks to my coauthor Bob Supnik for his continued inspiration; it is great to see a V.P. who can cut code with the best of them. My thanks also to the contributors to the Digital Notes files, a great source of folklore. Therein lies a treasure trove of solutions from people who are helping each other solve the same problems.

Bob Supnik: The design, implementation, and debug of SIM was made possible by the generous help of many people. Craig St. Clair and Deb Toivonen of the Digital archives located rare manuals and documents on Digital's 12-bit, 16-bit, and 18-bit systems. Tom West and Don Lewine of Data General Corporation provided documentation and support on the Nova. Carl Friend's private collection of Data General hardware and software was a crucial source of documentation and software for the Nova and the RDOS operating system. Doug Jones, Bill Haygood, and John Wilson allowed me to use the sources to their simulators and freely answered arcane questions about

Table 8
Architectures Implemented by SIM

	PDP-8	PDP-11	Nova
CPU	PDP-8/E	J-11, Q-bus	Nova 820
Options	KE8E EAE, KM8E memory extension	Integral FP11	Multiply/divide
Memory	4–32K words	16 KB–4 MB	4–32K words
Terminal	KL8E	DL11	KSR-33, Dasher
Paper tape	PC8E	PC11	Yes
Clock	DK8E	KW11L	Yes
Printer	LE8E	LP11	Yes
Storage	RX8E/RX01 RK8E/RK05 RF08/RS08	RX11/RX01 RK11/RK05 RLV11/RL01,2	4019 4046/4047, 4048, 4057, 4234
Magnetic tape	TM8E/TU10	TM11/TU10	6026

	PDP-4	PDP-7	PDP-9	PDP-15
CPU	PDP-4	PDP-7	PDP-9	PDP-15/30
Options		T177 EAE, T148 memory extension	KE09A EAE, KX09A memory protection KP09A power	KE15 EAE, KM15 memory protection KP15 power
Memory	4–8K words	4–32K words	4–32K words	4–128K words
Terminal	KSR-28	KSR-33	KSR-33	KSR-35
Paper tape	Integral T75 punch	T444 reader T75 punch	PC09A reader- punch	PC15 reader- punch
Clock	Yes	Yes	Yes	Yes
Printer	T62	T647	T647E	LP15
Storage		T24 drum	RF09/RS09	RF15/RS09 RP15/RP02
Magnetic tape			TC59/TU10	TC59/TU10

the hardware. In addition, Bill provided a working OS/8 system disk, and John copied several PDP-11 operating system disks off a working PDP-11/34. Megan Gentry was an important source of PDP-11 folklore, debugged some of the subtlest problems, created the Makefile, and provided the first and most frequently used distribution site. Ben Thomas provided the character-by-character I/O routines for VMS. Chris Suddick helped debug the PDP-11 floating-point code. Warren Toomey and the enthusiasts at PUPS (the PDP UNIX Preservation Society) in Australia allowed me access to their archive of early UNIX releases. Leendert Van Doorn debugged the PDP-11 simulator with UNIX V6, and Franc Grootjen with 2.11 BSD. Larry Stewart provided the initial impetus to the project, and Ken Harrenstein made an important contribution to preservation by implementing a DECsystem-10 simulator. Last, but not least, Max Burnet generously provided documentation and software from the Digital Australia collection, answered questions based on his 30 years of experience with Digital's systems, and made connections with and introductions to the worldwide community of historic machine hobbyists and enthusiasts.

References and Notes

1. As managing director of Digital's Australian subsidiary from 1975 to 1982, Max Burnet created and operated the PDP trade-in program.

2. M. Burnet, "An Update on the Museum Treasures," *DECUS Australia Symposium Proceedings,* August 1993.

3. M. Burnet, "The '94 Update on the Museum Treasures," *DECUS Australia Symposium Proceedings,* August 1994.

4. M. Burnet, "The Last Great PDP-11," *DECUS Australia Symposium Proceedings,* August 1995.

Table 9
Simulator Performance

Simulator	Simulated Instructions per Second	Real Instructions per Second	Ratio
PDP-8	1,800,000	400,000	4.5:1
PDP-11	440,000	500,000	.88:1
Nova	1,700,000	750,000	2.26:1

```
ucoder> pdp8

PDP-8 simulator V2.2b
sim> att rk0 os8.dsk
sim> boot rk0

.DA 08-APR-96

.DIR

 08-Apr-96

COPYIT.SV   2 09-Mar-93   PASS2 .SV  20 11-Oct-92   FORT3 .LD   3 06-Jul-93
DIRECT.SV   7 11-Oct-92   PASS20.SV   5 11-Oct-92   CLOSE .SV   2 10-Jul-93
CCLX  .SV  24 25-Feb-93   PASS3 .SV   8 11-Oct-92   FORT4 .FT   1 11-Jul-93
PIP   .SV  11 11-Oct-92   RALF  .SV  19 11-Oct-92   FORT4 .LD   2 04-Aug-93
FOTP  .SV   8 11-Oct-92   RESORC.SV  10 11-Oct-92   FORT6 .LD   2 09-Aug-93
ABSLDR.SV   5 11-Oct-92   RUNOFF.SV  24 11-Oct-92   FORT5 .FT   1 09-Aug-93
BASIC .SV  11 11-Oct-92   SABR  .SV  24 11-Oct-92   FORT5 .LD   2 09-Aug-93
BATCH .SV  10 11-Oct-92   SCROLL.SV  17 11-Oct-92   FORT6 .FT   1 09-Aug-93
BCOMP .SV  26 11-Oct-92   SET   .SV  20 11-Oct-92   METSC .SV  10 11-Aug-93
BITMAP.SV   5 11-Oct-92   SRCCOM.SV   5 11-Oct-92   METSC2.SV  10 11-Aug-93
BLOAD .SV  10 11-Oct-92   TECO  .SV  32 11-Oct-92   EMAT  .SV   9 11-Aug-93
BOOT  .SV   5 11-Oct-92   VERSN3.SV  10 11-Oct-92   EMDCT .SV  14 11-Aug-93
BRTS  .SV  24 11-Oct-92   BUILD .SV  33 11-Oct-92   EMTST .SV  10 11-Aug-93
CHEKMO.SV  15 11-Oct-92   BASIC .OV  16 11-Oct-92   SINST1.SV  14 11-Aug-93
COMPAF.SV   5 11-Oct-92   BUILD6.SV  33 11-Oct-92   ADDER .SV  13 11-Aug-93
CREF  .SV  13 11-Oct-92   BUILT .SV  33 12-Oct-92   FORT7 .FT   1 30-Aug-93
EDIT  .SV  10 11-Oct-92   HELP  .HE   1 18-Oct-92   CLEAR .LS   2 13-Jan-94
EDITS .SV   6 11-Oct-92   HELP  .HL  72 18-Oct-92   CLEAR .CF   2 13-Jan-94
EPIC  .SV  14 11-Oct-92   HELP  .OC   4 18-Oct-92   CLEAR .SV   2 13-Jan-94
F4    .SV  20 11-Oct-92   FORT7 .LD   2 07-Sep-93   CLEAR .PA   1 13-Jan-94
FRTS  .SV  26 11-Oct-92   JMPTST.SV   3 18-Oct-92   CLEAR .BN   2 13-Jan-94
FUTIL .SV  26 11-Oct-92   JMPJMS.SV   3 18-Oct-92   DEMO  .    28 21-Mar-95
HELP  .SV   5 11-Oct-92   RK8ENS.BN   1 30-Oct-92   DOS   .PA   4 25-Jan-94
LIBRA .SV  11 11-Oct-92   INST1 .SV  14 01-Dec-92   DOS   .BN   1 25-Jan-94
LIBSET.SV   5 11-Oct-92   INST2 .SV  11 01-Dec-92   DOS   .LS  10 25-Jan-94
LOAD  .SV  16 11-Oct-92   FORT  .FT   1 17-Jun-93   SHELL .PA   1 25-Jan-94
LOADER.SV  12 11-Oct-92   FORT  .LD   2 09-Jul-93   SHELL .BN   1 25-Jan-94
MATST .SV   9 11-Aug-93   FORT2 .LD   2 09-Jul-93   SHELL .LS   2 25-Jan-94
MDTST .SV  14 11-Aug-93   FORT2 .FT   1 22-Jun-93   BASIC .WS   1 10-Mar-94
OCOMP .SV   8 11-Oct-92   DOS   .SV   2 25-Jan-94   FOO   .PA   1 31-Mar-94
OPTF4 .SV  13 11-Oct-92   SHELL .SV   2 25-Jan-94   FOO   .BN   1 31-Mar-94
PAL8  .SV  19 11-Oct-92   FORT3 .FT   1 26-Jun-93

  95 Files In  980 Blocks - 2212 Free Blocks

.
Simulation stopped, PC: 01207 (KSF)
sim>
```

Figure 2
PDP-8 Simulator Running OS/8

```
ucoder> nova

NOVA simulator V2.2b
sim> att dp0 rdos.dsk
sim> set tti dasher
sim> boot dp0

Filename?

NOVA RDOS Rev 7.50
Date (m/d/y) ? 4 8 96
Time (h:m:s) ? 16 26 0

R
list/e sys-.-
SYS5.LB         17216  D      05/24/77 13:18  05/31/85  [001017]    0
SYS.SV          56320  SD     12/14/95 16:21  12/14/95  [005057]    0
SYS.LB          20240  D      04/30/85 14:49  05/31/85  [000746]    0
SYS.OL          30720  C      12/14/95 16:21  12/14/95  [005272]    0
SYSGEN.SV       23040  SD     05/02/85 22:20  05/31/85  [001401]    0
R
disk
LEFT: 2158   USED: 2706   MAX. CONTIGUOUS: 2054
R

Simulation stopped, PC: 41740 (LDA 1,4,3)
sim>
```

Figure 3
Nova Simulator Running RDOS

5. A. Ahi, G. Burroughs, A. Gore, S. LaMar, C.-Y. Lin, and A. Wiemann, "Design Verification of the HP 9000 Series 700 PA-RISC Workstations," *Hewlett-Packard Journal,* vol. 43, no. 4 (1992).

6. W. Anderson, "Logical Verification of the NVAX CPU Chip Design," *Digital Technical Journal,* vol. 4, no. 3 (1992): 38–46.

7. R. Calcagni and W. Sherwood, "VAX 6000 Model 400 CPU Chip Set Functional Design Verification," *Digital Technical Journal,* vol. 2, no. 2 (1990): 64–72.

8. A. Hutchings, "The Evolution of the Custom CAD Suite Used on the MicroVAX II System," *Digital Technical Journal,* vol. 1, no. 2 (1986): 48–55.

9. M. Kantrowitz and L. Noack, "Functional Verification of a Multiple-issue, Pipelined, Superscalar Alpha Processor—the Alpha 21164 CPU Chip," *Digital Technical Journal,* vol. 7, no. 1 (1995): 136–144.

10. D. Siewiorek, C. Bell, and A. Newell, *Computer Structures: Principles and Examples,* "The IBM System/360, System/370, 3030, and 4300: A Series of Planned Machines That Span a Wide Performance Range," and "PMS Notation" (New York: McGraw-Hill, 1982).

11. R. Brunner, ed., *VAX Architecture Reference Manual,* chapter 9, "Compatibility Mode" (Bedford, Mass.: Digital Press, 1991).

12. This simulator has since been withdrawn from the network.

13. R. Rustin, ed., *Debugging Techniques in Large Systems,* R. Supnik, "Debugging Under Simulation" (Englewood Cliffs, N. J.: Prentice-Hall, 1971).

14. For information on and pictures of Data General minicomputers, see C. Friend's web page at http://www.ultranet.com/~engelbrt/carl/museum/index.html.

15. For information on and pictures of many historic computers, see J. Jaeger's web page at http://www.msn.fullfeed.com/~cube/collect.htm.

16. For information on and pictures of many historic computers, see P. Pierce's web page at http://www.teleport.com/~prp/collect/index.html.

17. For documentation and relevant links, see D. Jones's web page at www.cs.uiowa.edu/~jones/pdp8/. For his simulator, cross assembler, and core images, see ftp://ftp.cs.uiowa.edu/pub/jones/pdp8.

18. For information on his simulator and OS/8 disk image, see W. Haygood's web page at ftp://sunsite.unc.edu/pub/academic/computer-science/history/pdp-8/emulators/haygood.

19. For more information on J. Wilson's simulator (executable only), see his web page at ftp://ftp.update.uu.se/pub/ibmpc/emulators.

20. For more information on the PDP-11 UNIX archive, see the PUPS home page at http://minnie.cs.adfa.oz.au/PUPS/index.html.

```
ucoder> pdp11

PDP-11 simulator V2.2b
sim> att rk0 rtrk.dsk
sim> boot rk0

RT-11SJ (S) V05.04

.

.da 8-apr-96

.dir
 08-Apr-96
NL    .SYS     2  18-Sep-89      RT11FB.SYS    94  18-Sep-89
RT11SJ.SYS    80  18-Sep-89      SPOOL .REL    11  14-Apr-87
PTESTX.MAC    23  27-Jan-94      GVI   .SAV     5  18-Apr-90
BINCOM.SAV    24  27-Sep-88      DUP   .SAV    49  27-Sep-88
DIR   .SAV    19  27-Sep-88      IND   .SAV    58  27-Sep-88
LIBR  .SAV    24  27-Sep-88      MACRO .SAV    61  27-Sep-88
LINK  .SAV    49  27-Sep-88      RESORC.SAV    25  27-Sep-88
FORMAT.SAV    24  27-Sep-88      ODT   .SAV     8  05-Oct-89
PBCOPY.SAV     2  16-Feb-89      SYSLIB.OBJ    55P 05-Oct-89
ODT   .OBJ     8  05-Oct-89      SYSMAC.SML    61  16-Mar-89
SIPP  .SAV    21  27-Sep-88      DATE  .SAV     3  02-Feb-89
IOP   .SAV    11  24-Apr-89      SWAP  .SYS    27  27-Sep-88
TT    .SYS     2  18-Sep-89      DL    .SYS     4  18-Sep-89
DM    .SYS     5  18-Sep-89      DP    .SYS     3  18-Sep-89
DX    .SYS     4  18-Sep-89      RK    .SYS     3  18-Sep-89
LS    .SYS     5  05-Oct-89      MT    .SYS     9  18-Sep-89
LP    .SYS     2  18-Sep-89      SP    .SYS     6  18-Sep-89
PIP   .SAV    30  27-Sep-88      HANDLE.SAV     7  16-Feb-89
LD    .SYS     8  26-Dec-90      MAC   .SAV    61  27-Sep-88
LC    .SYS     2  01-Jan-80      UCL   .SAV    13  22-Dec-89
UCL   .CCL     4  07-Oct-90      STARTS.COM     1  19-Jan-94
MTPIP .SAV    28  27-Feb-87      MTROL .SAV    17  27-Feb-87
MLIB  .SYS   300  20-Dec-90      HELP  .SAV   132  20-Dec-90
XPC   .SAV    16  25-Jun-91      DESS  .SAV    18  09-Mar-88
PTESTX.OBJ     8
 49 Files, 1432 Blocks
 3330 Free blocks

.sho dev

Device    Status          CSR     Vector(s)
------    ------          ---     ---------
  NL      Installed      000000   000
  TT      Installed      000000   000
  DL      Installed      174400   160
  DM      Not installed  177440   210
  DP      Not installed  176710   254
  DX      Installed      177170   264
  RK      Resident       177400   220
  LS     -Not installed  176500   470 474 300 304
  MT      Installed      172520   224
  LP      Installed      177514   200
  SP      Installed      000000   110
  LD      Installed      000000   000
  LC      Installed      177514   200

.
Simulation stopped, PC: 146506 (ASR R5)
sim>
```

Figure 4
PDP-11 Simulator Running RT-11

Biographies

Maxwell M. Burnet
Max Burnet has been with Digital in Australia for 29 years. During that time, he has sold, serviced, or marketed all the machines in the collection. He managed the Digital Australia subsidiary for seven years. He was a salesman in Boston during 1971 and managed to replace an IBM 1620 at Tufts University with a PDP-10. He is currently the oldest surviving "techie" in the Sydney office and makes many corporate presentations in Australia. He manages the Australian DECUS Society, the Subsidiary's local content and export obligations with the Australian Government, and the local Product Assurance Group. He has collected a museum of early Digital machines and is known around Sydney as "Museum Max." He received a B.Sc. (honours) from Melbourne University.

Robert M. Supnik
Bob Supnik has been with Digital in the United States for 19 years. He joined the Mass Storage Group and then moved into Semiconductor Engineering, where he successively managed the last PDP-11 implementation (the J-11), Advanced Development, the first single-chip VAX implementation (the MicroVAX chip), and the VAX Microprocessor Group. He also wrote or contributed to the microcode of every single-chip VAX microprocessor. In 1988, he started the Alpha program, which he managed through launch of the first products in 1992. He then became technical director, first of Engineering and then of the Computer Systems Division. In 1996, he became vice president of Research and Advanced Development. He has B.A. degrees in mathematics and in history from MIT, and an M.A. in history from Brandeis University.