INTRODUCTION

For some applications the native machine is not the system of choice in which to develop software, as when the target machine is unavailable (because it is still being developed, is obsolete, or is inaccessible) or inconvenient (as when there is minimal target-system support for debugging). In such cases, simulation or emulation may be preferred. Simulation has the advantage of giving the user intimate access to the target machine, usually through a rich debugging package. Typically this richness is achieved at a high development cost for the simulator and at a target-system performance degradation of four or more orders of magnitude. Emulation can offer processing speeds comparable to the target system (even faster, for slow target machines), but typically does not support a rich debugging environment.

In designing the PRIM system, we have attempted to retain the best features of both the simulation and emulation approaches while at the same time minimizing their disadvantages. PRIM provides a sharable, uniform framework for running emulations of target machines; within that framework is a rich user interface that supports interactive target-system and emulator debugging. When the user is not engaged in debugging, the target system runs at emulator speeds, but a sophisticated debugging package is available immediately when needed. PRIM was developed within a modern timesharing system so as to provide convenient access, a file system, resource management, and a large set of utilities without the cost of developing yet another operating system.

THE PRIM SYSTEM ARCHITECTURE

The emulation of a target machine under PRIM involves three different system levels: a timesharing system, which runs on a DEC PDP-10; the PRIM framework, which runs at user level on the PDP-10; and target-machine emulation tools controlled by that framework, which run on a sharable MLP-900 microprogrammable processor. (The MLP-900—Multi Lingual Processor—is the prototype of a microprogrammable processor designed by Standard Computer Corporation to follow their IC-7000.) This architecture is shown schematically in Figure 1. A more complete description of the PRIM architecture was presented by the authors at the Tenth Annual Workshop on Microprogramming. The timesharing system hardware and software provide shared access to the MLP-900. The PRIM framework supports interactive users at terminals and provides access to the file system for the emulator. And the emulator maintains a complete target-system environment. The PRIM framework can be used for both emulator development and target program debugging.

The PDP-10 is a large, general purpose computer to which new devices can be connected fairly easily, since the I/O bus is extensible and the multiported memory is external to the processor. At USC Information Sciences Institute (ISI), it runs the TENEX timesharing operating system developed at Bolt, Beranek, and Newman. TENEX is strongly oriented toward the support of interactive computing, serving both local users and remote users connected via the ARPANET. The operating system does not interact directly with the user; rather, it allocates resources, manages the file system, and supports the execution of TENEX processes, each process running in its own paged virtual memory and interacting as appropriate with its own user via a terminal of some kind. To support PRIM, TENEX was extended with software to provide access to the MLP-900 by TENEX processes; the MLP-900 was extended with hardware and software to guarantee the integrity of TENEX even against errant microcode.

The MLP-900 is a large, fast, vertical-word, microprogrammable processor with a writable control memory. The processor consists of an operating engine and a control engine. The operating engine is a 36-bit arithmetic/shift unit with 32 general registers, 16 mask registers, and a 1K internal memory. The control engine is a control unit with interrupt and branch logic, a subroutine-call stack, 128 programmable flip/flops, and 4K of writable control memory. Cycle time is 250 nanoseconds, during which either one or both engines can execute a 32-bit instruction. The MLP-900
The software at the system level consists of a small operating system resident in the MLP-900, known as the microvisor, and a new TENEX device driver to shake hands with the microvisor and govern access to the MLP-900 by TENEX processes. The MLP device driver is the sole TENEX operating system module added. It allows a TENEX process to create, run, and control a subordinate MLP process in much the same way it can a subordinate TENEX process. It schedules use of the MLP-900 among contending users and supervises the microvisor. Most of the microvisor is devoted to swapping emulator contexts (control memory and MLP registers) as the driver passes control from one user to another. The process of swapping an emulator's context into and back out of the MLP-900 requires about ten milliseconds of microvisor execution and introduces no perceptible delays beyond those ordinarily encountered in a timesharing system. The rest of the microvisor responds to emulator requests for service, manages the MLP pager, and performs other tasks required by the driver in TENEX. The microvisor runs in the privileged supervisor state that allows access to all resources; emulator microcode runs in the user state that protects all the critical resources from modification. PDP-10 memory is not directly addressed by microcode; instead, memory references are to addresses in a virtual memory identical to that of a TENEX process. These virtual addresses are translated to real addresses by the MLP pager, which is controlled (via the microvisor) by the driver in TENEX. A reference to a page not in memory results in a page-fault interrupt into the microvisor, which passes the fault on to the driver and retries the memory operation after the page is fetched by TENEX.

The net effect of the design is a sharable emulation facility in which each emulator runs independent of all others in its own context, accessing its own virtual target memory under control of the PRIM framework that created it. The framework has potential access to all of its emulator's context and target memory and may inspect and/or modify them.

The PRIM system development started in 1972. The initial hardware and software were operational in May 1974 with the ability for multiple (local and remote) users to develop and debug emulator microcode. The complete system became available in March 1976. All software and hardware were developed at ISI. System software took about one man-year. The TENEX MLP-900 device driver represents about 2000 machine instructions; the MLP-900 microvisor occupies about 500 words of control memory.*

THE PRIM FRAMEWORK

The PRIM framework consists of TENEX processes that define and implement the PRIM user command language, create an MLP-900 emulation process and control its execution at the user's behest, and provide I/O service for that emulation process. The I/O service implements a set of primitives to perform target I/O operations on installed devices after the user has associated them with TENEX files.

An emulator is required to cooperate in the debugging process, although the demands are minimal. Essentially, an emulator is expected to stop cleanly when interrupted by the framework or on the occurrence of a small number of predefined events that it monitors and to report its reason(s)

* It is worth noting that the operating-system level of PRIM was designed, developed, and tested using a TENEX system that, except for maintenance, was in production use continuously. The hardware interfaces were checked out using a dedicated 16K-word memory module and a specially built I/O bus simulator that connected the MLP-900 to the PDP-10 via a high-speed EIA terminal line. A private MLP-900 driver controlled this "smart terminal." After the microvisor was developed and the swapping and paging algorithms validated, the memory module was shared among concurrent MLP-900 users. After validation of the hardware and software interfaces between the MLP-900 and the PDP-10, the I/O bus simulator was removed and the MLP-900 was connected to the PDP-10 I/O bus. An MLP-900 device driver was then added to TENEX and the MLP-900 connected to the full TENEX memory; since most of the driver logic had already been checked out, only a short extension of maintenance periods was required for stand-alone checkout of the new device driver. The change-over to this driver had no effect on the microvisor.
for stopping. When the emulator halts or is interrupted by user intervention, control returns to the user at command level via the framework. Specific commands are available to start (or resume) execution.

The emulator writer must supply the PRIM framework with tables that define the target system architecture and symbols and drive a target system assembler and disassembler. Except for machine and user symbols and target assembly language, the same command interactions apply to every use of an emulator and framework. The framework contains two separate command-processors, known as the exec and the debugger. The exec has a large number of commands of fixed form that are used infrequently. The debugger has a smaller number of commands, but they are of variable form and are used much more frequently. Although both command processors offer automatic command completion and help facilities, each uses a language tailored to its functions. Typically, a user interacts with the exec only briefly when starting and debugging a session; during the session he interacts primarily with a target program or the debugger. The exec commands are based on easily remembered keywords that often require several characters to establish uniqueness: the debugger commands are uniquely identified with only a single keystroke.

The PRIM debugger is a target-system-independent interactive program for debugging a PRIM emulator or a target program running on such an emulator. Basically, it permits a user to examine, search, and modify target system locations and to set and clear breakpoints. Target system assembly language and symbols are recognized and generated, expressions of arbitrary complexity are accepted, and arithmetic is performed according to the conventions of the target machine. The debugger commands are listed in Table I.

Representation commands allow the user to designate the form and notation of the debugger's representation of data (FORMAT, MODE, and the three SYMBOL commands). Target system data may be represented as text or instructions or as formatted or unformatted numeric values; numbers may be represented as symbols or as signed or unsigned integers. The user may define multi-field data formats, select user symbol tables loaded earlier via an exec command, and add new symbols to a symbol table or delete existing ones.

Control commands allow the user to execute debugger commands conditionally or indirectly (IF and USE), to continue execution of a target system, either at the system's program counter or at a supplied address (GO), to single-step a target program (SINGLE STEP), to set and clear breakpoints in the target system and enter or edit debugger command "programs" to be executed at break time (BREAK, DEBREAK, and PROGRAM EDIT), and to return control to the PRIM exec (RETURN); privileged commands permit an emulator writer to single-step or set or clear a break in the MLP-900 (MLP STEP and MLP BREAK). Conditional and indirect execution are particularly important when a command is to be executed at a later time in an indeterminate context, such as in a command file or a break-time program. Breakpoints can be set on references (read, write, or execute) to target machine locations or on events that have been predefined by the emulator writer. The most powerful features of breakpointing is the ability for a user to associate with each breakpoint a sequence of debugger commands (the break-time program) to be executed when the break condition is detected and control is passed to the debugger. Coupled with conditional and indirect execution of debugger commands and the ability to continue target system execution, break-time programs offer the user a very powerful facility to trace execution, monitor events, and perform other complex exploratory functions automatically and repeatedly.

Display commands allow the user to examine the contents of designated target locations (TYPE, PRIOR, NEXT, SAME, and EQUALS), to search extended areas within the target system according to content (LOCATE), to examine arbitrary expressions (EVALUATE), and to review a record of recent target-system control transfers (JUMP HISTORy); a privileged command permits an emulator writer to display MLP-900 control memory symbolically (MLP TYPE). Any displayed location may be modified.

The miscellaneous commands offer convenience features that do not fit in the other categories. They allow the user to change the contents of target system locations without having to display them first (SET and CLEAR) and to annotate transcripts or break-time output (COMMENT); a privileged command permits an emulator writer to compile code changes directly into MLP-900 control memory (MLP CHANGE).

The PRIM exec comprises a diverse collection of commands that are not directly involved in the intimate control of debugging. The exec commands are listed in Table II. One major group of commands is concerned with the establishment of a target machine configuration (INSTALL, SET, and UNINSTALL) and with the mounting of TENEX files on the configured I/O devices (CANCEL, MOUNT, REASIGN,REWIND, and UNMOUNT). Another group of commands involves the loading of various classes of state information from files (LOAD, TABLES, RESTORE, and SYMBOLS) and with the saving of all or part of the target context on a file for subsequent restoration (SAVE). There are commands that transfer control from the exec to some other state (QUIT, DEBUG, and GO), and commands that alter the state of the PRIM framework (NO, CHANGE, COMMANDS, and ENABLE). PRIM will, on request, maintain a transcript of all or a part of a PRIM session (TRANSCRIPT and CLOSE); such transcripts have been

<table>
<thead>
<tr>
<th>Representation</th>
<th>Control</th>
<th>Display</th>
<th>Miscellaneous</th>
</tr>
</thead>
<tbody>
<tr>
<td>FORMAT</td>
<td>MLP BREAK*</td>
<td>MLP TYPE*</td>
<td>MLP CHANGE*</td>
</tr>
<tr>
<td>KILL SYMBOL</td>
<td>MLP STEP*</td>
<td>EQUALS</td>
<td>CLEAR</td>
</tr>
<tr>
<td>MODE</td>
<td>BREAK</td>
<td>EVALUATE</td>
<td>COMMENT</td>
</tr>
<tr>
<td>NEW SYMBOL</td>
<td>DEBREAK</td>
<td>JUMP HISTORy</td>
<td>SET</td>
</tr>
<tr>
<td>OPEN SYMBOL</td>
<td>GO</td>
<td>LOCATE</td>
<td></td>
</tr>
<tr>
<td></td>
<td>IF</td>
<td>NEXT</td>
<td></td>
</tr>
<tr>
<td>PROGRAM EDIT</td>
<td>PRIOR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RETURN</td>
<td>SAME</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SINGLE STEP</td>
<td>TYPE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>USE</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

* Commands for emulator developers only.
TABLE II.—PRIM Exec Commands

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Save/Restore</th>
<th>Miscellaneous</th>
</tr>
</thead>
<tbody>
<tr>
<td>CANCEL</td>
<td>LOAD*</td>
<td>NO*</td>
</tr>
<tr>
<td>FILESTATUS</td>
<td>TABLES*</td>
<td>CHANGE</td>
</tr>
<tr>
<td>INSTALL</td>
<td>RESTORE</td>
<td>CLOSE</td>
</tr>
<tr>
<td>MOUNT</td>
<td>SAVE</td>
<td>COMMANDS</td>
</tr>
<tr>
<td>PERIPHERALS</td>
<td>SYMBOLS</td>
<td>DEBUGGER</td>
</tr>
<tr>
<td>REASSIGN</td>
<td></td>
<td>ENABLE</td>
</tr>
<tr>
<td>REWIND</td>
<td></td>
<td>GO</td>
</tr>
<tr>
<td>SET</td>
<td></td>
<td>NEWS</td>
</tr>
<tr>
<td>SHOW</td>
<td></td>
<td>QUIT</td>
</tr>
<tr>
<td>UNINSTALL</td>
<td></td>
<td>TIME</td>
</tr>
<tr>
<td>UNMOUNT</td>
<td></td>
<td>TRANSCRIPT</td>
</tr>
</tbody>
</table>

* Commands for emulator developers only.

found invaluable by users with CRT terminals—and therefore no written record of their work. The remaining commands (FILESTATUS, PERIPHERALS, SHOW, NEWS, and TIME) are information-displaying commands.

The PRIM framework was written in BLISS.4,5 It occupies about 35K words of PDP-10 memory, including fixed work space, and took somewhat over three man-years to produce.

EMULATOR MODEL

A prototypical PRIM emulator has been developed based on the constraints of the MLP-900 environment, the objectives of the PRIM system, the requirements of the PRIM framework, and the specific interface conventions that framework defines.6 The environment consists of execute-only microcode residing in control memory, the MLP-900 registers, and a 256K 36-bit (virtual) main memory; the registers and virtual memory together comprise the context into which are mapped the registers and memory of the target machine plus various other regions devoted to required PRIM functions. The mapping is arranged at the convenience of the emulator writer, with accompanying tables describing this mapping to the PRIM framework. The emulator can modify its context in the course of emulation, stop (thereby returning control to the framework), and request I/O services from the framework.

The most fundamental PRIM requirement is for a bit-compatible emulation of the target machine with accurate timing. Beyond that, an emulator must have a control structure that allows it to stop after any cycle and subsequently resume emulation in a manner totally transparent to the target machine. While a single target instruction constitutes the typical cycle, other events, such as interrupts or I/O data transfers, must also be treated as emulator cycles. Changes to the context made by the user during an emulator stop must appropriately affect the target machine on resumption of emulation. In particular, this requires that the emulator have no hidden copies of target state information when it stops.

Target timing in PRIM is virtual. The emulator is required to increment an internal, high-resolution (typically, 50 nanosecond) clock to reflect the passage of target-machine time; there is no fixed relationship among target time, MLP-900 time, PRIM framework time, and real time. Emulated cycles that consume target time (e.g., instruction execution) advance the clock; emulated cycles that nominally occur in parallel with the former (e.g., I/O controller activity) are scheduled for service relative to that internal clock. The result is a small, event-driven, discrete simulation system with target instruction execution being treated as a background task. Figure 2 is a top-level outline of a typical PRIM emulation tool.

Target peripheral devices are included in an emulation. Device control logic, timing, and interaction with the target machine are contained entirely within the emulator. An I/O transfer is performed by the I/O server in the PRIM framework in response to a call by the emulator, which must then monitor the transfer for completion. The transfer is carried on in parallel with emulator execution, and the emulator may have several I/O operations outstanding simultaneously. The freedom given the user to define target system configurations and the protocols for use of the I/O server primitives, together constitute a generalized framework for device emulation.

User commands to inspect and modify the target machine environment are implemented by the PRIM framework via tables associated with each emulator, requiring no direct intervention by the emulator itself. But user commands involving the breakpoint facility do require the cooperation of the emulator. Each breakpoint a user sets is marked by a flag in a portion of the shared context outside the address space of the target machine; these flags are called "meta-bits." The emulator must monitor the target machine con-

```
initialize emulator;
FOREVER DO
BEGIN
  IF reason to stop
    THEN BEGIN
      execute interrupt cycle
    END;
ELSE
  IF interrupt pending
    THEN execute interrupt cycle
  ELSE execute event routine
END;
```

Figure 2—PRIM emulator outline
tinually for the occurrence of any breakpoints flagged by these meta bits and, when one occurs, log it and prepare to stop at the end of the current emulation cycle. Interrupting execution at the end of the cycle is consistent with the requirement for stopping cleanly and also avoids a problem that would arise if execution were interrupted in mid-cycle on the occurrence of the break condition—that of responding to possible changes made in the context during the break while not reporting the same breakpoint repeatedly.

PRIM defines two classes of breakpoints, known as reference breaks and event breaks. A reference break occurs on a read (data fetch), write, or execute reference to specified target locations. The meta bits for reference breaks are the four high-order bits of each 36-bit context word. As every target location can have its own meta bits, there is no practical limit on the number of simultaneous breakpoints the user may set. An event break occurs whenever a specified target-machine event arises. Each emulator supports its own set of event breaks, but at a minimum includes instruction execution (target machine single-step), jumps or branches, memory stores, interrupts, and I/O transfers. One meta bit is used for each type of event break an emulator monitors. The event breaks are particularly useful for close monitoring of target machine behavior.

To date, there are three supported PRIM emulations: the AN/UYK-20 (a modern 16-bit computer), the Univac U1050 (an ancient byte-addressed decimal machine with a 30-bit instruction word), and the Intel 8080 chip. Both the AN/UYK-20 and the U1050 emulations include a comprehensive set of peripherals. These two emulators each require about 3K words of control memory; the 8080 emulator, which supports only a simple terminal-like device, requires about 1K words of control memory. For the AN/UYK-20, which is a well-documented system of moderate complexity, the emulation of the CPU and the peripherals each required about six man-months. Although the U1050 is of comparable complexity, development of the emulator took somewhat longer because of the unavailability of accurate and up-to-date documentation. The 8080 emulation took only one man-month. All of these emulators require on the order of five microseconds to emulate one target memory cycle. For the AN/UYK-20, the ratio of emulated-to-real time is about 101; for the U1050 and the 8080, the ratio is about 1/1. For all emulations, as the I/O traffic increases, the performance ratio increases.

DISCUSSION

Since it became operational, PRIM has been used extensively by the System Design Laboratory (SDL) at the Naval Ocean Systems Center in San Diego, by the Logistics group at Gunter Air Force Station, and by a "smart terminal" project at ISI. SDL is using PRIM to support the design and development of naval systems employing the AN/UYK-20 minicomputer. The Logistics group at Gunter has used PRIM to examine the consequences of replacing existing peripherals with higher-performance ones on a Univac U1050 system. In the process, they discovered an unexpected design limitation in a communications interrupt routine and also uncovered several system bugs for which patches were developed on the spot. An Intel 8080 emulation was used at ISI to debug terminal software that would otherwise have been impossible to observe. The MLP-900 has been used an average 20 hours per month (representing an average of about 40 hours of connect time) by as many as six concurrent users. During this time an average of ten hours of TENEX time per month was used by the PRIM framework. The maximum level of use to date has been 90 hours of MLP-900 time in one month. Ordinarily, a PRIM user is unaware that he is sharing the MLP-900 with other users.

By cleanly and sharply separating the debugging and target-machine emulation tasks, PRIM has been able to avoid most of the disadvantages of simulation and emulation while at the same time combining their advantages. In achieving this sharp separation of function, PRIM established a uniform and systematic structure for the development of emulators. This structure not only minimizes the involvement of the emulator in the debugging process, but also greatly simplifies the task of emulator development as it utilizes a standard package of I/O service routines and provides a convenient control structure suitable for a large family of target-machine emulations. As most of PRIM consists of sharable system-level and user-level code that is common to this potentially large family of target system emulations, a more extensive development effort (with its consequently more sophisticated design) could be justified than would have been appropriate for a single-machine emulation or simulation.

REFERENCES
