A CASE STUDY OF COMPUTER EMULATION*

T.A. MARSLAND AND J.C. DEMCO

University of Alberta, Edmonton

ABSTRACT

Through the design and construction of an emulator for DEC PDP-11 computers, the extent to which the Nanodata QM-1 can serve as a universal host is being explored. The principal results show the extent to which emulation is possible without excessive sacrifices in speed. In addition to insights into the construction of a complete emulator, the paper identifies important problems associated with concurrent emulation of different target hardware, and describes solutions to some of them.

RÉSUMÉ

Par le design et la construction d'un émulateur pour les ordinateurs PDP-11, les auteurs examinent la possibilité de se servir du Nanodata QM-1 comme hôte universel. Les résultats de l'étude indiquent jusqu'à quel point l'émulation est possible sans trop sacrificier la rapidité. En plus des détails sur la construction d'un émulateur complet, quelques uns des problèmes importants associés a l'émulation simulantée de différents équipements sont identifiés et des solutions proposées.

1 INTRODUCTION

With the diversity of computing equipment available, it is not practical to acquire all the hardware necessary to execute a wide variety of software packages. However, it may be feasible to construct *emulators* on which to run selected software. According to Rosin,[1] an emulator is "a complete set of microprograms which, when embedded in a control store, define a machine." Thus an emulator realizes a *virtual* or *target* machine, while a *host* is the one which supports the microprograms. Our primary interest is in a *universal host*,[2] one that is capable of simultaneously modelling several different target architectures.

Through the design and construction of an emulator for DEC PDP-11 computers,[3] the extent to which the Nanodata QM-1[4,5,6] can serve as a universal host is being explored. The principal results in this paper identify some necessary properties of emulation hardware and show the extent to which complete emulation is possible without excessive sacrifices in speed. By complete emulation we mean the ability to load and execute on the host the object code forms of the target machine software, and provide direct access to I/O devices.

Many other computer emulations have either been for outdated machines with long memory access times, simple instruction formats, and limited I/O capabilities,[7,8] or have not simulated I/O instructions exactly, but simply translated them into high-level requests to the host machine's operating

*Received 8 March 1977; revised 15 October 1977. Financial support for the study was obtained from the National Research Council of Canada, Grant A7902.

system.[9,10] In contrast, the study reported here considers the emulation of a contemporary computer, one with several instruction formats, addressing modes, and a main memory cycle time comparable to that of the host machine.

Our goal is to provide multiple concurrent emulation of different computer architectures,[11] typically minicomputers and microprocessors, within a single host machine. For example, software for a variety of micro-processors can be developed better on emulators which provide interrupt histories, performance monitoring, and significant I/O support. Emulation therefore is appropriate in the following situations:

- The configuration of the target machine is too small for software development. During such development the emulator could provide extra assistance in the form of better debugging aids, larger virtual machine, and access to the host's peripherals.
- The target machine does not (yet) exist.
- The application is needed too infrequently to warrant purchase of the target machine.
- An application exists which requires simultaneous use of software packages from different computers, e.g. a network processing application.

In summary, emulation is practical in any low usage application. Multiple concurrent emulation applications are not yet well developed, but a necessary prerequisite is the existence of complete emulators for contemporary machines.

This paper therefore is aimed at providing insight into the efficient emulation of one contemporary computer on another. Before describing the basic structure of the emulator, a summary of the capabilities of the target and host computers is given, with emphasis on those facilities which are difficult to implement. The paper concludes with an evaluation of the QM-1 as a host machine, especially of those features which are less than ideal.

1.1 *The target and host machines*

The PDP-11/10 target hardware is one of a well-known series of machines.[3] Sufficient for our purposes is that it may have 28K words (16 bits) of byte addressable memory, six general purpose registers, and two special ones – a stack pointer and a program counter (PC). Five different types of instruction formats exist (fig. 1), and so a special table lookup scheme may be needed to extract the various fields. Internal details of the architecture are given elsewhere.[12]

No specific I/O instructions exist in the PDP-11; instead, I/O operations are performed by reading/writing device registers located in an additional 4K words of the address space. When an I/O interrupt is recognized by the CPU, a *trap* occurs and a new Processor Status (PS) word and PC are loaded. Traps also occur for the following conditions: attempted access of nonexistent memory, stack overflow, invalid instruction detection, attempted access of a word on an odd byte boundary, and execution of a trap instruction.

In contrast, the QM-1 is a word addressable 18-bit machine, with three distinct address spaces: *main store*, *control store*, and *nanostore*. Note, however, that nanostore is 360 bits wide, divided into five fields of equal size (Appendix A). Figure 2 compares capacities of the host and target memories, and specifies

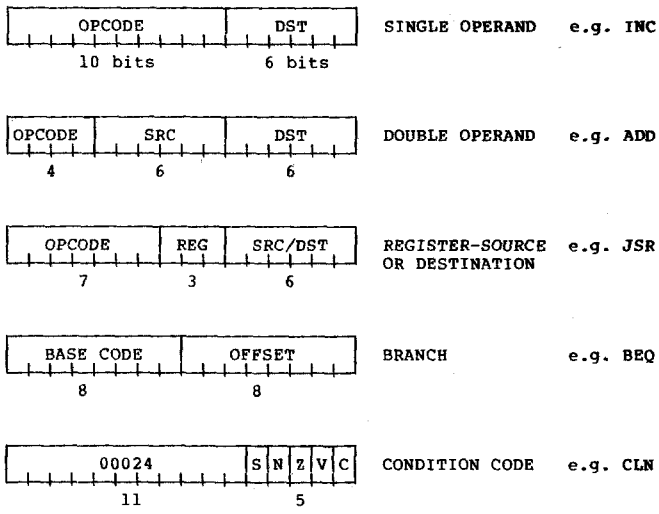

FIG. 1. Major PDP-11 instruction formats.

	Host QM-1	Target PDP-11
REGISTERS		
general purpose	32 18-bit 12 6-bit	8 16-bit
special purpose	32 18-bit 20 6-bit	device regs (variable #)
MAIN STORE		
width	18 bits	16 bits
maximum size	256K	28K
cycle time	960 nsec	980 nsec
CONTROL STORE		
width	18 bits	
maximum size	16K	
cycle time	160 nsec	
NANOSTORE		
width	360 bits	
maximum size	1K	
cycle time	160 nsec	

FIG. 2. Properties of user-accessible memory.

the cycle time of each type. Since most of the potential target hardwares have an address space which is larger than our control store, all target machine images are kept in main store.

Several QM-1 hardware features allow nanocode to interpret as instructions information in control store. The opcode of each control store *microinstruction* serves as the address of the nanoprogram which carries out the instruction. Parameters may also be passed to the nanocode from the operand fields of the microinstruction. In this way, a complete microinstruction set can be defined dynamically depending on the contents of nanostore. In turn, microprograms can be written, for example, to interpret main store data as the instruction

set of an existing or hypothetical computer. Of course, nanoprograms can also be written to interpret a main store instruction set directly, so both control store and main store can serve simply as data areas to an executing nanoprogram.

The QM-1 has two banks of thirty-two 18-bit registers, referred to as *local store* and *external store*. The former are for general use, while the latter serve primarily to interface the CPU with I/O channels and to hold interrupt addresses. The machine has both an *ALU* and *shifter* which may be used to operate on 18-bit, 36-bit, and 16-bit quantities. These and the special ALU s which exist for index and mask operations are described fully elsewhere.[4] Major data paths within the QM-1 are 18 bits wide (fig. 3). Included in the basic architecture is a set of thirty-two 6-bit registers called *F-store*, most of which are used to control bus connections. For example, if the value 2 is placed into the F-register FAOD (*A*rithmetic *O*utput *D*ata), then the output of the ALU is logically connected to local store register 2. Several other F-registers (*G*-store) are available for general use, while the remainder have specific machine control and status functions.

Even when a bus connection exists, no data are transferred until requested by a nanoprogram. The transfer of data does not affect the bus connection, providing a measure of *residual control*[13] since these connections are altered only by the explicit loading of special F-registers. Several data transfers may be established and performed at once, as shown in the sample nanoprogram of Appendix A, figure A2.

2 The Emulator

With the QM-1, the emulator builder may exercise the hardware at two levels below main store programming: nanoprograms, for highly parallel operations such as instruction field decode, and microprograms for sequential resource usage such as I/O device control. Thus two distinct approaches are possible:*

- Design a special microinstruction set, and implement it in nanocode. The emulator may now be built rapidly as a collection of microprograms.
- Implement the emulator's instruction set completely in nanocode. This is referred to as *direct* emulation, and should provide faster execution.

Our emulator lies between these two extremes. The PDP-11 instruction set is implemented in nanocode, while control store is used to hold tables for instruction decoding, a condition code bit map, and microroutines (written in the MULTI[14] instruction set) to handle I/O and control functions. These device drivers make the host peripherals serve the emulator as their target counterparts. Naturally the microinstructions are themselves defined by nanocode, but speed in processing the I/O is of lesser consequence.

Each main store instruction is decoded by using the high order nine bits of the PDP-11 word as the index into a control store table. From there, a hierar-

*Alternatively one could model the PDP-11 microroutines themselves. This approach might be quick to implement, but had unacceptably slow execution times for our purposes.

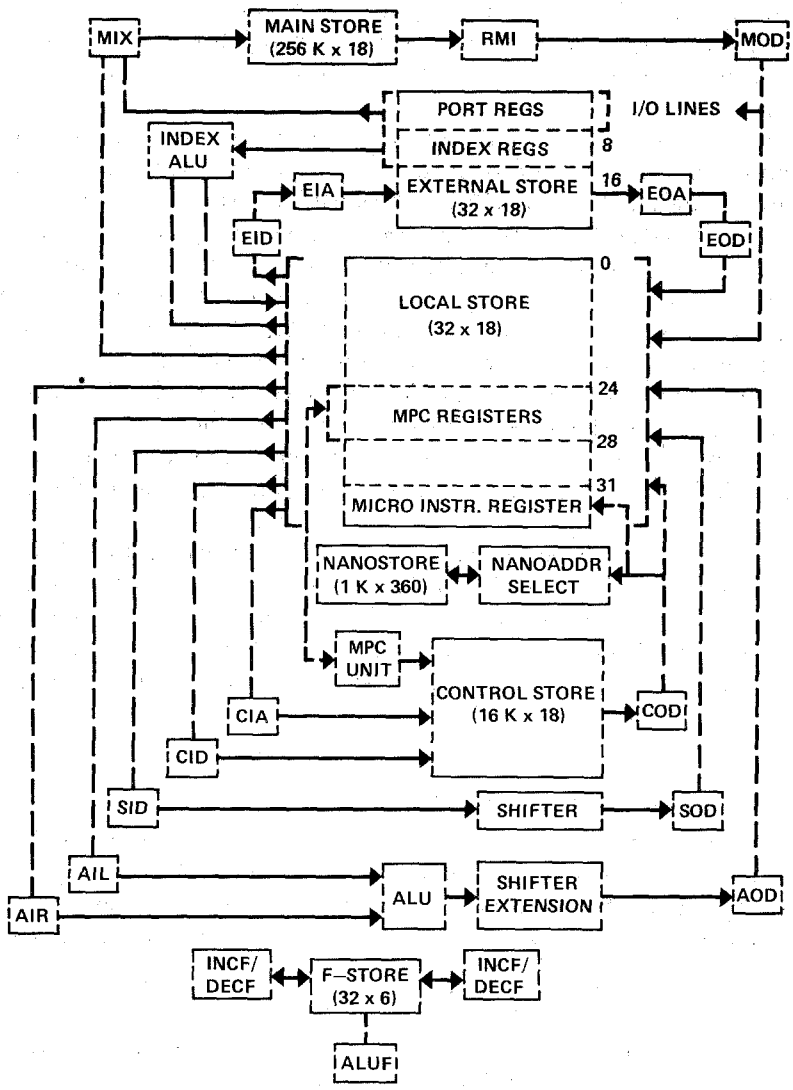

FIG. 3. Host bus control structure.

chical lookup proceeds for those instructions whose decoding requires more than nine bits. Each entry in this table has two fields:

- The first contains the address of a nanoroutine. In the case of single- and double-operand instructions, a *setup* routine is called, whose purpose is to prepare source and destination values. In the case of most other instructions, it is the address of the routine to carry out the entire instruction.

- The second field has different uses depending on the main store instruction. For single- and double-operand instructions, it is the address of an *execute* routine, to carry out the desired calculation. For a conditional branch, the field contains the address of a routine which changes the program counter appropriately. If the instruction is TRAP or EMT, the second field is a pointer to the main store location of the new PS and PC. In some cases, the second field is unused.

If an I/O device register is accessed during instruction execution, control is passed to a microroutine to initiate the I/O, before the execution of the next instruction.

2.1 *Instruction flow and routine descriptions*

As PDP-11 instructions are decoded, different sequences of nanoroutines are executed. After processing by the *FETCH* routine the basic flow follows that shown in figure 4. Instructions with source or destination operands are first

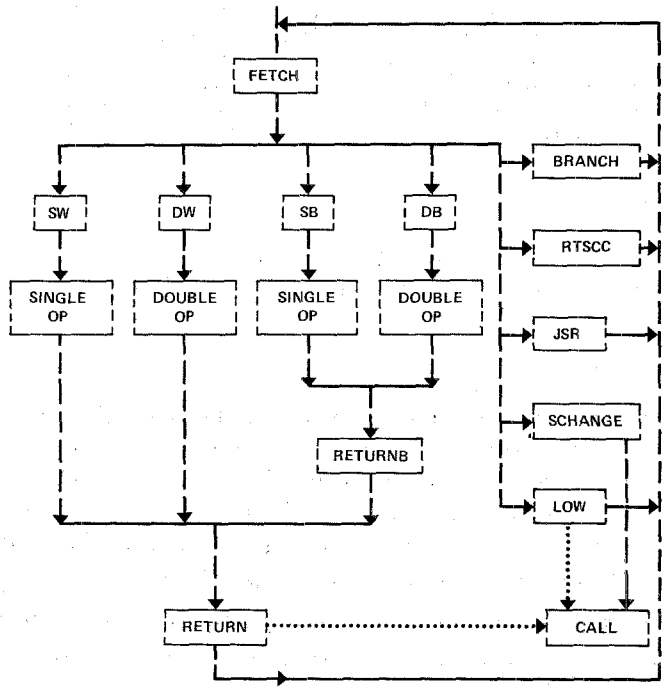

FIG. 4. Basic flow of control in the emulator.

processed by one of the previously mentioned setup routines, before being passed on to an execute routine such as *SINGLE OP* or *DOUBLE OP*. The remaining branching and condition code manipulating instructions are executed immediately. In the diagram the dotted lines represent conditional incovations of *CALL*, a routine which gives control to a microprogram to handle all I/O and the HALT, WAIT, and RESET instructions. A description of these nanoroutines has appeared in an earlier report,[15] but may be summarized:

• *FETCH*
Fetches the next instruction from main store and begins decoding it via a setup routine.

• *MODE*
A subroutine which calculates the effective address of a PDP-11 instruction operand, and returns its value. *MODE* is not shown in figure 4 because it is called from many places. Error checking is performed, with control conditionally passed to *SCHANGE*.

• *SCHANGE*
The state-change routine is used to perform TRAP, EMT, BPT, and IOT instructions; it also handles I/O traps and error traps.

• *CALL*
The invocation of microsubroutines to handle I/O, HALT, RESET, WAIT, and logical interrupts is made by this routine.

• *RTSCC*
Handles the group of instructions in the range 0002XX–0003XX, a mixture of condition code operations, subroutine return, and the SWAB instruction.

• *LOW*
Instructions in the range 0000XX–0001XX are the responsibility of *LOW*. In the case of HALT, WAIT, and RESET, control is passed to *CALL*. In the case of BPT and IOT, *SCHANGE* is used. In the case of RTI and RTT, the second part of *SCHANGE* is used to load a new PC and PS. In the case of JMP, *MODE* is called to calculate the effective address indicated by the DST field (bits 0–5) of the instruction. The PC is set to this address, and the next instruction is fetched. In all other cases, an illegal instruction trap occurs.

• *RETURNB*
Re-constructs a word after a byte operation.

• *RETURN*
Places the result in the location specified by the DST field. If a device word was either read or written during the instruction execution, control passes to *CALL*; otherwise, control passes to *FETCH*.

2.2 *Memory mapping and utilization*

Figure 5 summarizes memory requirements for the host's three address spaces. The nanostore and control store allocations are self-explanatory; however, the use of main store may need some clarification. Since the target has a 16-bit word and the host main store word is 18 bits wide, two bits are available as tags.[16] These tags are used to differentiate the target machine's device registers from the remaining existent and non-existent memory.

- Bit 17 is 1 if the word does not exist: referencing this location will cause a trap via *SCHANGE*.
- Bit 16 is 1 if the word is a device register, in which case *RETURN* will pass control to *CALL* to perform the I/O function.

The Rotate, Mask and Index (RMI) unit is valuable here for determining tag settings, but is not essential. In addition, a base-bound option maps main store addresses, enabling concurrent residence of more than one emulator.

2.3 *Complete emulation*

The PDP-11 emulator successfully executes standard instruction diagnostics; memory, tape, and disk exercisers; and also the DOS-11 and MINI-UNIX operating systems. No changes whatever were made to these programs; indeed, for many the source code was not available.

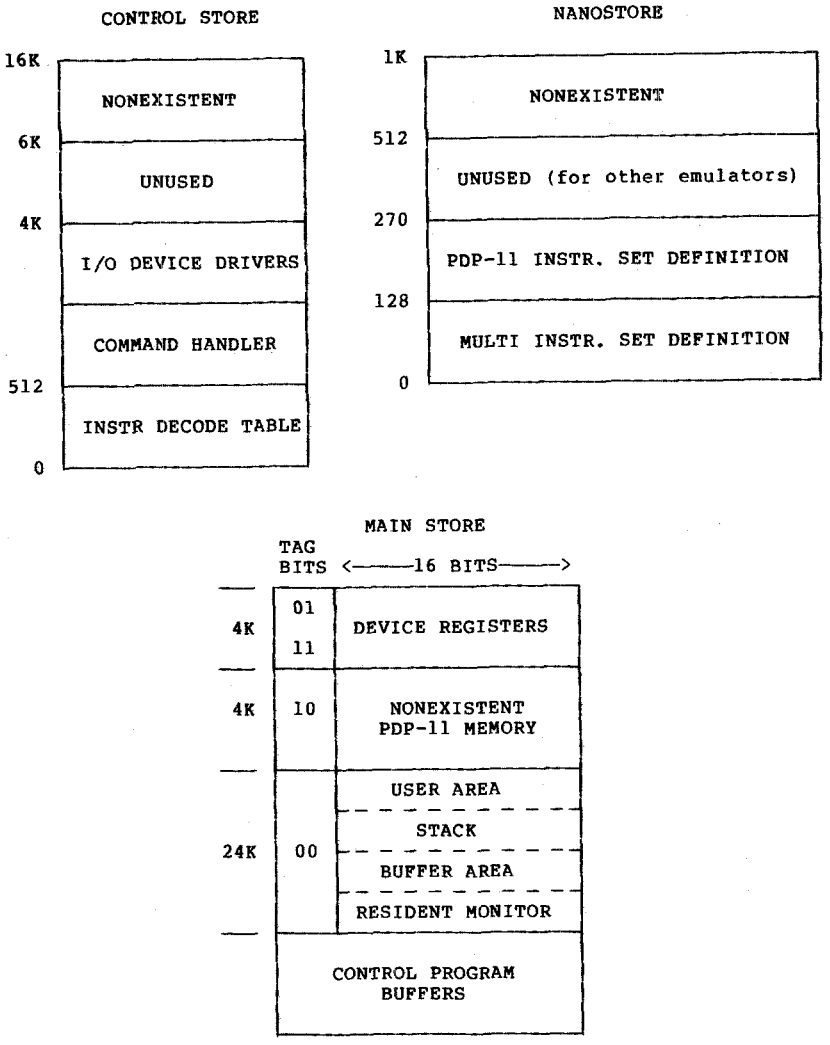

FIG. 5. Memory allocation in the host.

The MINI-UNIX operating system is a small-processor version of the UNIX operating system[17] developed at Bell Laboratories. It ran successfully on the emulator the day it arrived, with no problems whatever. Since then the emulator has been upgraded with the addition of the extended shift, exclusive-or, multiply, and divide instructions. Because the operating system can now execute these instructions directly instead of fielding illegal instruction traps and simulating the instructions in software, a marked increase in performance has been achieved. For example, the execution time of a sample batch stream benchmark (including a compilation, line printer I/O, and some repetitive numerical calculations) decreased from over sixteen minutes to under six minutes.

The above results demonstrate that the emulator is functionally complete; however, it does contain some deficiencies as the following details show:

- Odd PC values are ignored; in fact, no PC checking is done at all.
- Stack overflow checking is incomplete for JMP and JSR instructions. Also, the change-of-state routine will not detect stack overflow.

Elimination of these shortcomings does not warrant the excessive space overhead in the nanoprograms. In practice these details do not affect the correct execution of programs since odd PC values do not arise, and stack overflow is detected in a real PDP-11 only after the user's buffers have been overwritten (fig. 5). The stack overflow problem stems from the fact that main store is not accessed through a common nanoprogram. The difficulty is a result of the limited subroutine nesting capabilities within nanostore and means that this emulator cannot be easily extended to handle the PDP-11/45 memory segmentation unit.

Finally, the trace trap debugging feature was not implemented. It can be added to the microcode without modification to the nanoprogram portion of the emulator. Since the QM-1 and the control program already provide comparable debugging facilities, no requirement exists for this feature, and its implementation offered no new insight into emulation.

3 The Emulator Control Program

Although the instruction set of the target machine can be modelled in the host without great difficulty, carrying out I/O operations and responding to interrupts is not easily done by an emulator control program (ECP). Ideally the control program should not impose any extra architectural restrictions on the emulator, and it should be possible to write one without detailed knowledge of the other. Our additional requirements for concurrent support of several emulators further complicate the issue, since member emulators must be given *direct* low-level access to I/O devices through the ECP and the device drivers. Only by this means can the emulators be functionally equivalent to their target counterparts, to the extent that existing software can be transported to the emulator without change.

With the above ideals in mind, the following features were implemented in our control program:

- The system includes task control blocks (TCB s) for emulator management, dynamically modifiable device ownership, and a task switch capability.
- Low-level I/O support is provided, including a bi-level interrupt structure, device drivers, and unit control blocks (UCB s) to standardize device management.
- Emulator support facilities include storage display and modification, front panel simulation, and debug features such as register trace and single step.

The design and implementation of ECP was heavily influenced by a system used by the Nanodata Corporation to manage its Nova emulator.

3.1 *Emulator – ECP interaction*

A few technical difficulties are still being resolved with the multiple emulation

capability of the ECP, although some successful experiments have been performed.* Control of a single emulator is now done without problem. For example, each time the PDP-11 accesses a device register, control is passed to the ECP. A table of address pairs is searched to match the device register address; the second entry points to the device handler. Also passed to the I/O routine is a read/write flag. This process could occur twice in one main store instruction, if both source and destination are device registers. The device handler responds to the request by accessing the I/O devices directly. At present PDP-11 devices[18] which have been mapped onto comparable QM-1 hardware include: an LA30 terminal, a PC11 paper tape reader, an LP11 line printer, RK05 disks, TM11 magnetic tape drives, a CR11 card reader, and a KW11L line clock. Since no paper tape reader actually exists on our machine, the PC11 and CR11 share a real card reader!

ECP s interrupt handling mechanism has two stages. Interrupts are caught first by the low level handler (with interrupts masked for only a short period), which places the device's UCB into its present owner's priority-ordered queue (fig. 6). A logical interrupt is signalled by setting a particular G-register, and the physical interrupt is dismissed. When the emulator which owns the interrupting device is restarted, this register is interrogated by the instruction fetch routine, and control is conditionally passed to the second stage. Once the logical interrupt routine gains control, a check is made first of the "command pending" word in the TCB, and if necessary the command handler is invoked. The interrupt is then handled in a manner appropriate to the type of emulator. In the case of the PDP-11, the queue of UCB s is inspected, and if the CPU priority is lower than the bus request priority of the first UCB on the queue, a change of state in the emulator is forced via *SCHANGE*. Otherwise, the emulator is restarted at the point of interruption.

This bi-level structure has two important features:

- No restrictions are placed on the way interrupts are handled by a member emulator's device drivers. This is especially important if virtual machines with different interrupt structures are to be supported concurrently.
- The handling of an interrupt by a device driver is completely independent of any emulator's software interrupt handler. In particular, an interrupt from a device owned by an emulator which is not currently active is not lost.

In a system with a small number of terminal devices, it may be desirable to have an emulator console double as the system console. This is easily done by providing a simple mechanism called console redirection to "point" keyboard interrupts at the appropriate handler, either the emulator's or the system's. When the primary console is "owned" by the emulator, receipt of a special control prefix passes the ownership to the ECP. Commands may then be executed, even while the emulator is active. A command is provided to return console ownership to the emulator. By this means, more than one emulator may also use the same console device.

*During preparations for the Canadian Computer Chess Workshop (Sigart Newsletter, Nov. 1976) held in June 1976, a dual emulator environment was implemented on the QM-1 allowing two chess programs written for the Nova minicomputer to play each other. The environment included a manual task switch facility, and the sharing of the console terminal, disk, and clock.

TASK CONTROL BLOCK

QM-1 register save area
Logical halt indicator
Single instr. step indicator
Console command pending
PDP-11 console switches
Logical wait indicator
Pointer to a priority-ordered linked list of UCBs needing service by the upper level interrupt handler

UNIT CONTROL BLOCK

Physical (QM-1) device address
Lower level i/o handler entry address
Device status information
Link field for queuing the UCB to the upper level interrupt handler
Trap vector main store address
Bus request priority
Reset routine entry address
Device register copies

FIG. 6. ECP control block structure (single emulator environment).

Typical of the instructions which the control program has to handle is RESET, which invokes routines to re-initialize the devices that the emulator currently owns. Other obvious problems are for example emulating the HALT instruction, which should not stop the QM-1 (especially in a multi-emulator environment). Similarly the WAIT instruction cannot be dealt with by simply having an interruptible loop in nanocode, since the next interrupt need not necessarily come from a device which the emulator owns. Rather, WAIT sets the logical wait indicator in the TCB and decrements the PC so that the instruction is re-executed until a device owned by the emulator causes an interrupt.

Debug facilities in ECP include front panel simulation, single step, storage display and modification (register, main store, and control store), register tracing, and profiling. The profiling feature uses a 32K word buffer of QM-1 main store as an array of counters. When profiling is enabled, each line clock interrupt causes the PC to be sampled and the appropriate word in the buffer to be incremented. This allows maximum resolution with a minimum of interference (less than 0.05% overhead). With the aid of a specially microcoded device to give the emulator access to the profile buffer, and slight modifications to the UNIX "prof" program, a very accurate histogram showing system time utilization can be produced and analyzed. Preliminary measurements show that the major bottlenecks are the register save and restore routines, and the memory clearing section of the process swap routine: these code segments will be the first candidates for implementation directly in microcode or nanocode.

The Emulator Control Program has proved useful in debugging and controlling a single target machine, and in providing flexible low-level I/O which is transparent to the main store programmer. These facilities will remain as useful when multi-emulation capabilities are completed.

4 Universal Host Evaluation

A number of architectural characteristics of the PDP-11 and of the QM-1 affect the emulation of the former by the latter. An investigation into those components of architecture which are appropriate for general purpose emulation has been made[13]; the following statements may also be extended to emulation in general.

Ideally a host machine should have significantly more registers than the target. Of course, emulation of machines with a great many more registers, or registers wider than 18 bits, can be handled by maintaining them in control store. In our case the PDP-11 registers were easily accommodated in local store.

Fortunately, host main store is 2 bits wider than target memory, so data transfer is simplified; these extra bits are used as tags to specify the existence or purpose of each word of the virtual machine memory (fig. 5).

The PDP-11 emulator is fairly fast, executing instructions at better than one-half the speed of a Model 10. A more accurate measure of the relative speed can be obtained from a statistical analysis of the instruction times in Appendix B. The simpler instructions, and instructions using the simpler addressing modes, are relatively slower because of the rather long (2.5 microsecond) fetch and decode routine. On the other hand, use of the RMI unit to extract the 3-bit subfields in the operands reduces the instruction decode times by about 0.4 microseconds per operand.

4.1 *Emulation problems*

The large number of buses and the presence of residual control in the host enhance its capabilities for parallelism. This is especially important in instruction fetch and decode, which is usually the most complicated part of instruc-

tion execution. Parallelism, however, is not sufficiently great for the PDP-11 emulation to check stack overflow concurrently with effective address calculation.

Although the QM-1 has a 2-way branching facility at the nanoprogram level, a general N-way branch cannot readily be generated nor are arbitrary levels of subroutine call provided. In practice this is not a serious problem, but suggests the need for a hardware modification to provide, for instance, a 16-word stack to facilitate a chain of subroutine calls.

To emulate the PDP-11 efficiently, operations on various data widths are required. For example, the PDP-11 has byte operations which cannot be handled directly by the QM-1's shifter and ALU. There is no difficulty with arithmetic operations on bytes, since these are stored in the upper part of the word, but shifts and rotates require proper insertion of the carry bit – an operation that is awkward to perform. Compare for example the relative times for the ROR and INC instructions in Appendix B.* Although PDP-11 s execute these instructions in the same time, in our emulator the ROR is much slower for byte operations because of the complexity of bit insertion into the middle of a QM-1 word. Clearly a desirable feature for a universal host machine is a truly variable-width arithmetic and shifting capability, including correct generation of conditions such as carry out and overflow. However, an extremely complicated (and almost unusable) structure might result. For example, the PDP-11 includes the carry bit in its shifts; the IBM 360 does not. To provide a parallel variable-width shifter with both capabilities is an unenviable task for any designer.

Condition codes generated by the host's hardware must undergo a non-trivial-mapping to convert them to virtual machine condition codes. A powerful single-bit capability is required here, which the host does not have (table lookup is employed in the emulator). A similar problem exists whenever a signed conditional branch is needed, in order to take overflow into account.

The difference in unit of memory addressability between host and target forces a good deal of time- and resource-consuming housekeeping on the emulator. A PDP-11 byte address must be shifted right by one bit to produce the corresponding QM-1 address. In the case of byte operations, the low-order bit of the PDP-11 address is used as a byte selector, and the byte which is not affected by the operation must be saved before the operation is carried out and restored after its completion. An efficient variable-width memory access capability would be an asset to a universal host machine.

4.2 *Observations*

The QM-1 was more than capable of hosting the complete emulation of a fairly complex machine, the PDP-11. Since the host could do many difficult things easily, we were perhaps overcritical whenever some features were awkward to implement. Nevertheless, the final emulation speed was within a factor of two of the target machine, comparing favourably with simulation,

*A version of the Appendix appears in the Micro 9 proceedings under "An insight into PDP-11 emulation" by the authors.

where a reduction factor of thirty is more realistic. In addition, the QM-1 supports a variety of other emulators: the Nova 1200, IBM 7094 and S360, plus a number of lesser known machines.

Designing and implementing the PDP-11 emulator has required approximately nine man-months of work. Three months were taken up with the instruction set interpreter, and six were needed for the writing of the ECP and a full complement of device drivers.

In the area of multiple emulation, the common device drivers and the bi-level interrupt structure of the ECP provide uniform, low-level access to shared peripherals without the need to inhibit interrupts for long periods of time. It was our hope that this approach to I/O handling would spur thought in such areas as dynamic device ownership, fundamental differences between computer emulators and high-level language emulators, and even the question of whether or not a computer should know how to perform low-level I/O! Work is continuing on these topics, and also on the design of universal file systems to simplify concurrent emulator support.

Acknowledgments

The repeated discussions with S. Sutphen, regarding the hardware features of the PDP-11 and QM-1 computers, are much appreciated and helped reduce the inaccuracies in this study.

References

(1) R. Rosin, "Contemporary concepts of microprogramming and emulation," Computing Surveys, vol. 1, Dec. 1969, 197–212.
(2) M.J. Flynn and R.F. Rosin, "Microprogramming: an introduction and a viewpoint," IEEE Trans. on Computers, vol. C-20, July 1971, 727–731.
(3) Digital Equipment Corporation, *PDP11 processor handbook*, Maynard, Mass., 1975.
(4) Nanodata Corporation, *QM-1 hardware level user's manual*, Williamsville, N.Y., Second Ed., Mar. 1976.
(5) A.K. Agrawala and T.G. Rauscher, *Foundations of microprogramming*, New York: Academic Press, 1976.
(6) R. Rosin, G. Frieder, and R. Eckhouse, "An environment for research in microprogramming and emulation," Comm. ACM, vol. 15, Aug. 1973, 748–760.
(7) T. Schoen and M. Belsole, "A Burroughs 220 emulator for the IBM 360/25," IEEE Trans. on Computers, vol. C-20, July 1971, 795–798.
(8) R. Benjamin, "The Spectra 70/45 Emulator for the RCA 301," Comm. ACM, vol. 8, Dec. 1965, 748–752.
(9) G. Allred, "System/370 integrated emulation under OS and DOS," AFIPS Conf. Proc, vol. 38, 1971, 163–168.
(10) S. Tucker, "Emulation of large systems," Comm. ACM, vol. 8, Dec. 1965, 753–761.
(11) J.C. Demco, *Principles of multiple concurrent computer emulation*, M SC thesis, Dept. of Computing Science, University of Alberta, Edmonton, Aug. 1975.
(12) J.F. O'Loughlin, "Microprogramming a fixed architecture machine," *Microprogramming*, Infotech State of the Art Report 23, 1976, 205–224.
(13) S.H. Fuller, V.R. Lesser, C.G. Bell, and C.H. Karman, "The effects of emerging technology and emulation requirements on microprogramming," IEEE Trans. on Computers, vol. C-25, Oct. 1976, 1000–1009.
(14) Nanodata Corporation, "MULTI micromachine description," Williamsville, New York, Mar. 1976.
(15) J.C. Demco and T.A. Marsland, *A complete PDP-11 emulation*, Tech. Rep. 75–13, Dept. of Computing Science, University of Alberta, Edmonton, Aug. 1975.

(16) E. Feustel, "On the advantages of tagged architecture," IEEE Trans. on Computers, vol. C-22, July 1973, 644–656.
(17) D.M. Ritchie and K. Thompson, "The UNIX time-sharing system," Comm. ACM, vol. 17, July 1974, 365–375.
(18) Digital Equipment Corporation, *PDP11 peripherals handbook*, Maynard, Mass., 1975.

Appendix A: Nanoprogramming

Some examples of direct control programming for the QM-1 appear in the technical works.[4,5] For completeness, an illustrated example from our PDP-11 instruction fetch/decode nanoprogram is presented, along with a condensed general description of nanoprograms. Fach nanoword is divided into five 72-bit fields: a *K-vector*, and four *T-vectors*. The K-vector has a number of subfields which are used to hold such things as local constants, ALU and shifter controls, interrupt masks, and a field (KN) for the address of an executable nanoword. The T-vectors (named T1, T2, T3, and T4) have subfields corresponding to fundamental actions (called *nanoprimitives*) which may occur. When a new nanoword becomes active, the K-vector and T1 of the word gain control of the machine. All of the primitives selected in T1 are executed, and one *T-period* later control passes to the K-vector and T2. The cycle T1, T2, T3, T4, T1, T2, ... continues until one of the primitives causes another word to be read from nanostore and made active. The length of a T-period is 80 ns, but it must be stretched to 160 ns if two interacting primitives occur in the same T-vector.

The address used to locate the next nanoword receiving control can be taken from three sources:

- the 10-bit KN field of the active K-vector; one of 30 interrupt vectors in external store; or
- the nanoprogram counter (NPC).

In turn, the NPC can be loaded from:

- the KN field;
- the incremented NPC; or
- a word on the data bus from control store.

Conditional branching can be performed, but since neither the KN field nor the NPC may be inspected, only one level of subroutine call is possible. This inconvenience can be overcome by doing procedure oriented work at the microprogram level.

Some operations take more than one T-period to complete. For example, neither the shifter nor the ALU are pipeline devices, so input lines must not be altered until outputs are valid, usually two T-periods. Consider for example the one word nanoprogram in figure A1, which is part of the PDP-11 emulator's instruction fetch and decode routine. As indicated by the comments, the

SPARE	KN	KA	KB	KS	KX	KT	KALC	KSHC	KSHA	MASKS
	301	31	21	20	04	10	00	05	01	
2	10	6	6	6	6	6	6	6	6	12 bits

The above K-vector contains the local constants needed in the following four T-steps. The masks specify that interrupts are enabled for this sample.

```
S...                                First T-step is to be stretched (S)
                                    to 160ns

      GATE NS(NOT X)                Using KX as a mask on the special
                                    conditions, loop in T1 until main
                                    store is idle

      LOAD NPC(KN)                  Load address of next nanoword to be
                                    executed

      KB->FMIX                      Connect reg 21 to MIX bus

      KB->FSOD                      Connect SOD bus to reg 21

      G(G.PC-20.), G->FSID          Connect reg 7 (the PC) to SID bus

.S..  GATE SH                       Convert PDP-11 address to QM-1 address
                                    via right logical shift of length 1

      G(G KS), G->FMOD              Ready to send main store word to reg 20

      KT->FEOA, G->FSID             Bus connections for future use

      READ NS                       Read next nanoword, taking address from
                                    E-store if interrupt pending, or from
                                    the NPC if not

..X.                                Third T-step is to be 80ns long (X)

      KA->FEOD, KA->FSOD            More bus connections for future use
      KA->FCIA

      READ MS                       Start full cycle read from main store

      GATE NS                       Pass control to nanoword chosen by
                                    previous READ NS

...X                                T-step unused in this nanoword.
```

FIG. A1. A sample nanoprogram.

machine stays in T1 until main store is idle. The byte address of the next PDP-11 instruction is then taken from the PC (register 7) and converted into a QM-1 word address via the shifter. Finally, a full-cycle read of main store is initiated, and control passes to the next nanoword. Meanwhile, bus connections are being set up (fig. A2) to aid in the decoding process once the instruction has been read. Each time a new nanoword is loaded from nanostore, the K-vector is reset to its assembly time value. However, some of the fields can be altered during nanoword execution and used as local variables.

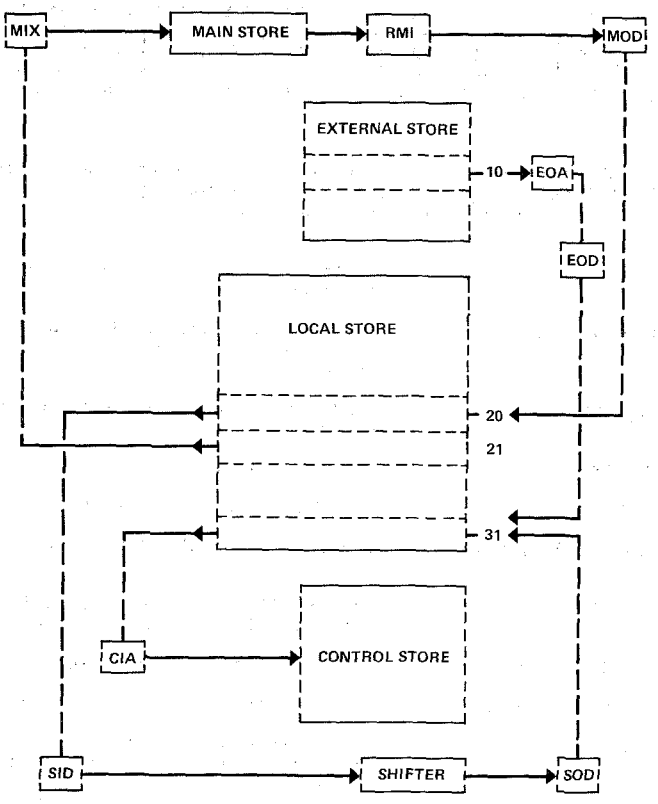

FIG. A2. Bus connections made by sample nanoprogram.

APPENDIX B: INSTRUCTION TIMING

The instruction execution times of the PDP-11 emulator and of the PDP-11/10 (3, Appendix B) are compared in the tables below. All timing information is in microseconds, unless otherwise noted.

SOURCE AND DESTINATION ADDRESS TIMES

Mode	PDP-11/10 SRC Time[1]	PDP-11/10 DST Time[2]	Emulator SRC, DST Time[3]
0	0.0	0.0	0.16
1	0.9	2.4	3.60
2	0.9	2.4	4.08[4]
3	2.4	3.4	4.64
4	0.9	2.4	4.40[5]
5	2.4	3.4	4.64
6	2.4	3.4	4.08
7	3.4	4.7	4.96

[1]For SRC Time, add 1.3 usec for Odd Byte addressing.

[2]For DST Time, and Odd Byte addressing: (1) add 1.3 usec for a nonmodifying instruction (CMPB, BITB, TSTB).
(2) add 2.4 usec for a modifying instruction.

[3]If stack overflow check is disabled, subtract 0.32 usec.

[4]If register is 6 or 7, subtract 0.08 usec. If increment is 1, add 0.08 usec.

[5]If register is 6 or 7, subtract 0.08 usec. If decrement is 1 subtract 0.08 usec.

DOUBLE OPERAND INSTRUCTIONS

Instruction	PDP-11/10 Basic Time	Emulator Basic Time[1]
ADD	3.7	6.08
SUB	3.7	6.56
BIC	3.7	6.08
BIS	3.7	6.08
CMP	2.5	6.32
BIT	2.5	6.16
MOV	3.7[2]	6.08[3]
XOR	—[4]	5.52

[1]If Byte instruction, add 1.44 usec.

[2]3.1 usec if Word instruction and Mode 0.

[3]If Byte instruction and DST Mode is 0, add 0.24 usec.

[4]Not available on the PDP-11/10.

Note: Instr Time = Basic Time + SRC Time + DST Time.

SINGLE OPERAND INSTRUCTIONS

Instruction	PDP-11/10 Basic Time	Emulator Basic Time[1]
CLR	3.4	5.44
COM	3.4	5.44
INC	3.4	5.44
DEC	3.4	5.52
NEG	3.4	5.68
ASR	3.4	6.48[2]
ASL	3.4	6.56[2]
ROR	3.4	6.56[2]
ROL	3.4	6.56[2]
ADC	3.4	5.68
SBC	3.4	5.84
TST	2.2	5.44
SWAB	4.3	6.72
SXT	—[3]	5.68

[1]If Byte instruction, add 0.80 usec for odd address, 0.72 usec for even address.
[2]If Byte instruction, add 0.80 usec.
[3]Not available on the PDP-11/10.
Note: Instr Time = Basic Time + DST Time.

JUMP INSTRUCTIONS

Instruction	PDP-11/10 Basic Time	Emulator Basic Time
JMP	1.0	3.60
JSR	3.8	3.84

Note: Instr Time = Basic Time + DST Time.

BRANCH INSTRUCTIONS

Instruction	PDP-11/10[1] (branch)	Emulator (branch)	Emulator (no branch)
BGE	2.5	3.28[2]	2.48[2]
BLT	2.5	3.28[2]	2.48[2]
BGT	2.5	3.76[2]	2.16[2]
BLE	2.5	3.44[2]	2.16[2]
BR	2.5	3.20	—
SOB	—[3]	3.36	2.96
All others	2.5	3.36	2.08

[1]Subtract 0.6 usec if no branch.
[2]Depending on N and V settings, add 0.0 to 0.64 usec.
[3]Not available on the PDP-11/10.

EXTENDED INSTRUCTIONS

Instruction	PDP-11/40 Basic Time[1]	Emulator Basic Time
MUL	8.88	14.72
DIV	11.30	15.88
ASH	2.58[2]	6.64[4]
ASHC	3.26[3]	6.72

[1]Not available on the PDP-11/10. Basic times given are for the PDP-11/40; 11/40 SRC times are not given.

[2]Add 0.3 usec per shift, plus 0.2 usec for left shift.

[3]Subtract 0.48 usec if no shift. Add 0.3 usec per shift.

[4]Add 0.24 usec for left shift.

Note: Instr Time = Basic Time + SRC Time.

CONTROL, TRAP, AND MISCELLANEOUS INSTRUCTIONS

Instruction	PDP-11/10 Instr. Time	Emulator Instr. Time
RTS	3.8	5.04
RTI	4.4	6.64[1]
CLR N, Z, V, C	2.5	4.00
SET N, Z, V, C	2.5	4.00
HALT	1.8	4.00[1]
WAIT	1.8	4.40[1]
RESET	100 msec	4.96[1]
EMT, TRAP	8.2	8.40[1]
BPT, IOT	8.2	9.44[1]

[1]Then invoke microroutine.