AN INSIGHT INTC PDP-11 EMULATION+

J. C. Demco
T. A. Marsland

Department of Computing Science
University of Alberta
Edmontcn, Alberta
Canada T6G 2H1

In order to evaluate the Nanodata QM-1 as a universal host computer, an emulator for a contemporary ccmputer, the PDP-11, was designed and constructed. It was required that the emulator be functionally equivalent to the target, without making excessive sacrifices in emulation speed. Some properties of emulation hardware necessary to achieve these gcals are identified. In addition, the paper describes a monitor designed to support different emulators concurrently on a single host machine.

1. INTRCDUCTION

Through the design and construction of an emulator for the DEC PDP-11/10 computer [1], the extent to which the Nanodata QM-1 [2, 3, 4] can serve as a universal host is being explored. The results summarized in this paper [5] identify some desirable properties of emulaticn hardware and show that complete emulation is possible without excessive sacrifices in emulation speed. One primary goal was that the emulator should execute all of the software for the target machine, rather than some specific package.

Many other computer emulations have either been for cutdated machines with long memory access times, simple instruction formats and limited I/O capabilities [6] [7], or have not simulated I/O instructions exactly, but simply translated them into high-level requests to the host machine's operating system [8] [9]. In contrast, the study reported here considers the emulation of a ccntemporary computer, one with several instruction formats, addressing modes, and a main memory cycle time comparable to that of the host machine (fig. 1).

2. THE EMULATCR

With the QM-1, two distinct ccnstruction apprcaches are possible:

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

Our emulator is essentially a direct one. Control store is used to hold tables for instructicn decoding, a condition code bit map, and microroutines (written in the MULTI [10] instruction set) to handle I/O and contrcl functions. These device drivers make the host peripherals serve the emulator as their target counterparts.

	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 STCRE		
width	18 bits	16 bits
maximum size	256K	28K
cycle time	960 nsec	980 nsec
CONTRCL STCRE		
width	18 bits	
maximum size	16K	
cycle time	160 nsec	
NANOSTORE		
width	360 bits	
maximum size	1K	
cycle time	160 nsec	

Fig. 1 Properties of User-Accessible Memory

+This study was supported by the National Pesearch Council of Canada, Grant A7902.

Naturally the microroutines themselves are driven by nanocode, but speed in processing the I/O is of lesser consequence.

For each main store instruction, the high order nine bits of a PDP-11 word are used as an index into one of the control store tables. Each entry has two fields. The first is typically the nanoaddress of a setup routine, whose purpose is to prepare source and destination values. The second usually addresses an execute routine, which carries out the desired calculation. 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

Fig. 2 is a block diagram showing the basic flow as the emulator executes PDP-11 instructions. The multi-way branch after the instruction FETCH routine reflects the execution of one of the setup routines mentioned in the previous section. The boxes labeled SINGLE OP and DOUBLE OP indicate the use of one of the execute routines. The dotted lines represent conditional invocation of CALL, whose function is to pass control to a microroutine to handle the I/O, plus the HALT, WAIT, and RESET instructions. A summary of each nanoroutine follows:

- 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 fig. 2 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.

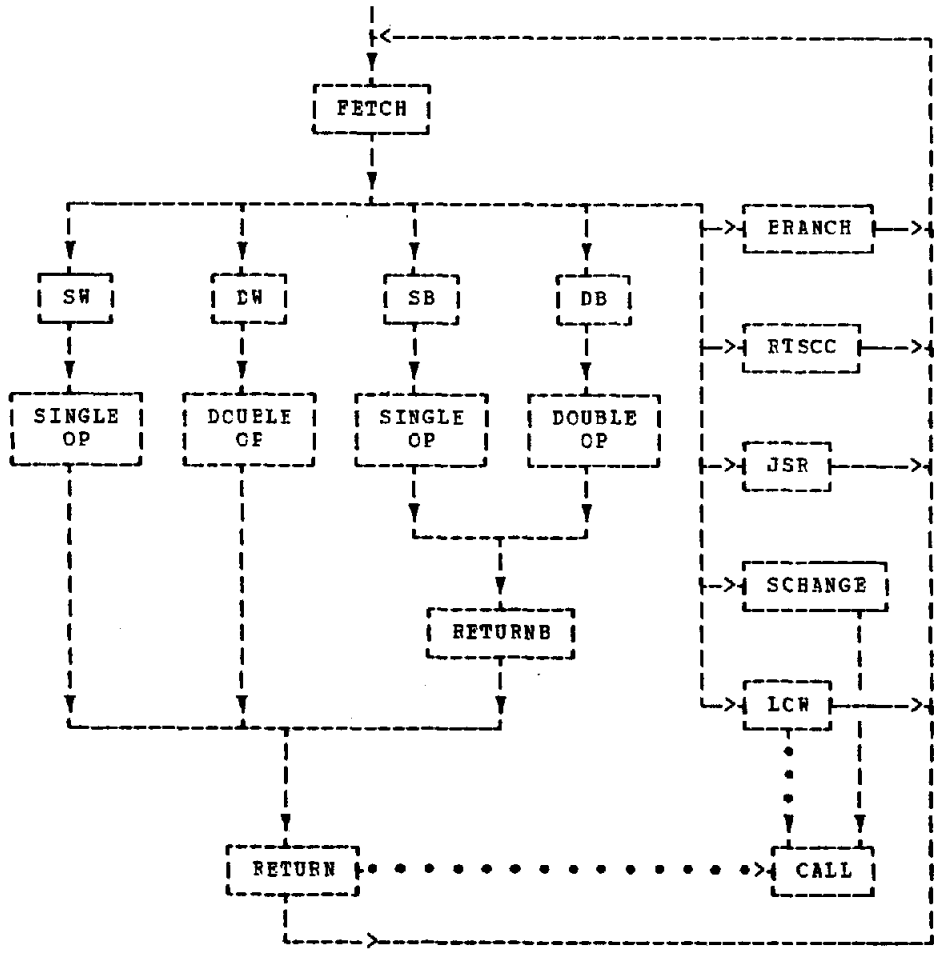

Fig. 2 Basic Flow of Control in the Emulator

• 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.

• LOW
Instructions in the range 0000XX - 0001XX are the responsibility of LOW.

• 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

The host memory requirements for each of its three address spaces are summarized in fig. 3. Since the target has a 16-bit word and the host main store word is 18 bits wide, two bits are available as tags. 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-Index (RMI) unit is valuable here for determining the tag settings, but is not essential.

2.3 Implementation Shortcomings

Although the emulator successfully executes standard instruction diagnostics, memory and disk exercisers, and also Version 8.08 of the DOS-11 operating system, it does contain some deficiencies as the following details show:

- Odd PC values are ignored; in fact, no PC checking is done at all. The extra time and space required here is probably not justified.
- Stack overflow checking is incomplete for JMP and JSR instructions. Also, the change-of-state routine will not detect stack overflow. Correcting this inaccuracy is almost impossible, given the present structure of the emulator.

Elimination of these shortcomings does not

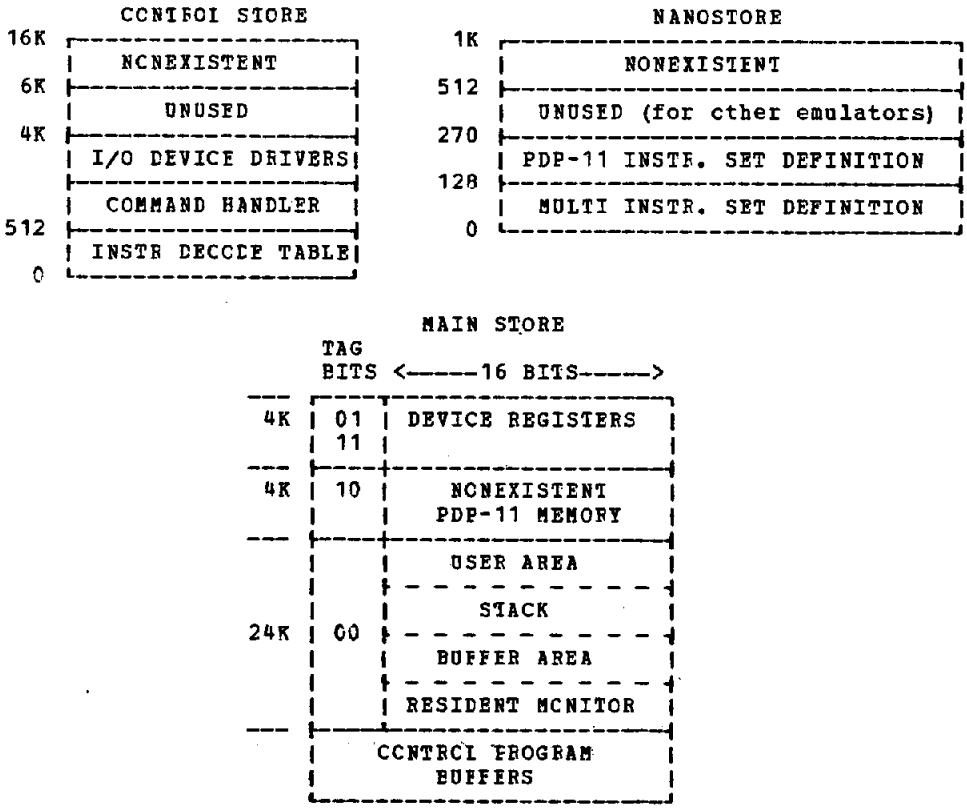

Fig. 3 Memory Allocation in the Host

warrant the excessive space overhead in the nanoprograms. In practice odd PC values should not arise, and stack overflow is detected in a real PDP-11 only after the user's buffers have been overwritten (fig. 3). In our case, the stack overflow problem stems from the fact that main store is not accessed through a common nanoprogram. The problem 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, because a common memory accessing mechanism is essential to deal with address translation.

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. No need was seen for the feature in our environment, and its implementation offered no new insight into emulation.

3. THE EMULATOR CONTROL PROGRAM

Several objectives guide the design of the Emulator Control Program (ECP). First, the control program must not impose architectural restrictions on an emulator. In fact, it should be possible to write an emulator without knowledge of ECP, and then to interface the two easily. Second, provision should be made for the concurrent support of different emulators [11]. Third, member emulators must be given low-level access to I/O devices through ECP and the emulator's device drivers. This is required in order that member emulators can be functionally equivalent to their counterparts, to the point of being able to transport existing software unchanged from the existing computer to the emulator. It was our hope that this approach to I/O handling would also 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! Finally, basic control and debug facilities must be provided.

The design and implementation of ECP was heavily influenced by a similar system called CONTROL [12], used by Nanodata Corporation to manage its Nova emulator.

3.1 Emulator - ECP Interaction

When the target 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 address is the entry point into 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 [13] which have been implemented include the simulation of an LA30 by a Tektronix 4023 CRT terminal, a PC11 paper tape reader (which is fed information via a Documation-600 card reader!), an LP11 line printer, RK05 disks, and the KW11L line clock.

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 then places the device's Unit Control Block (UCB) into a pricrity-ordered queue (fig. 4). A flag is set to signal a logical interrupt, and interrupt processing is completed. When the emulator is restarted, this flag is interrogated by the instruction fetch routine, and control is conditionally passed to the second stage. Once the logical interrupt routine gains control, a

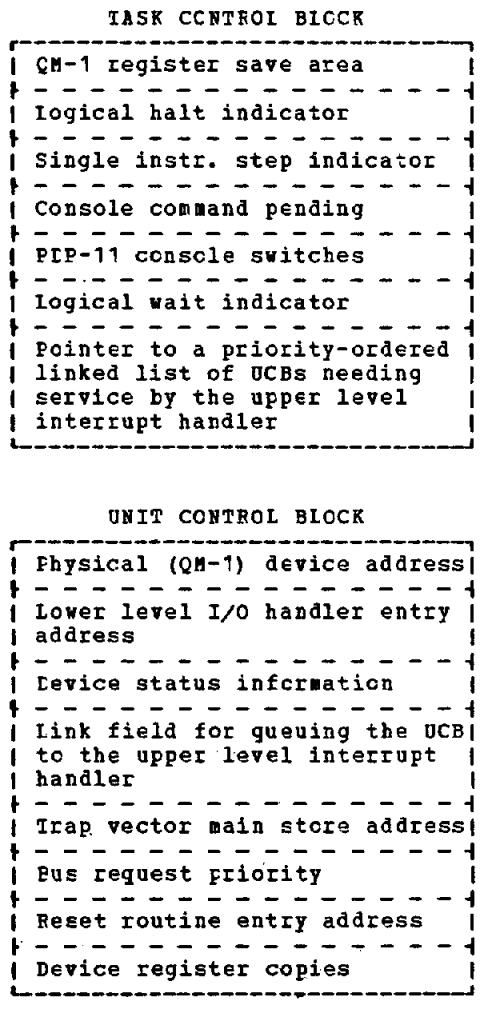

Fig. 4 ECP Control Block Structure (single emulator environment)

check is made first of the "command pending" word in the Task Control Block (TCB), and if necessary the command handler is invoked. The queue of UCB's is then inspected. 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, loss of an interrupt from a device owned by an emulator which is not currently running does not occur

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. Similarly, more than one emulator may use the same console device.

One 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.

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 [14]; the statements following 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 16 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 [15] to specify the existence or purpose of each word of the virtual machine memory (fig. 3).

The PDP-11 emulator is fairly fast, executing instructions at better than one-half the speed of a Model 10, approaching the physical limits imposed by the main store. 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 optional BMI 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 instruction execution. Parallelism, however, is not sufficiently great for the PDP-11 emulation to check stack overflow concurrently with effective address calculation.

The QM-1 does not have an elementary N-way branch capability, and only one level of subroutine nesting is readily available at the nanoprogram level.

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 which is awkward to perform. 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.

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 processed, 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 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 favorably with simulation, 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 S/360, plus a number of lesser known machines.

In the area of multiple emulation, the common device drivers and the bi-level interrupt structure of the ECP provide uniform access to shared peripherals, without the need to inhibit interrupts for long periods of time. Work is continuing on that topic, and also on the design of universal file systems to simplify concurrent emulator support.

ACKNOWLEDGMENT

The repeated discussions with Steven 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] Digital Equipment Corporation, PDP11 Processor Handbook, Maynard, Mass., 1975.

[2] Nanodata Corporation, QM-1 Hardware Level User's Manual, 2nd ed., Williamsville, New York, 1974.

[3] R. Rosin, G. Frieder, and R. Eckhouse, An Environment for Research in Microprogramming and Emulation, Communications of the ACM, vol. 15, no. 8, August, 1973, 748-760.

[4] A.K. Agrawala and T.G. Rauscher, Foundations of Microprogramming, Academic Press, New York, 1976.

[5] T. A. Marsland and J. C. Demco, A Contemporary Computer Emulation, Technical Report 76-1, Department of Computing Science, University of Alberta, Edmonton, Alberta, February, 1976.

[6] I. Schoen and M. Belsole, A Burroughs 220 Emulator for the IBM 360/25, IEEE Transactions on Computers, vol. C-20, no. 7, July, 1971, 795-798.

[7] R. Benjamin, The Spectra 70/45 Emulator for the RCA 301, Communications of the ACM, vol. 8, no. 12, December, 1965, 748-752.

[8] G. Allred, System/370 integrated emulation under OS and DOS, AFIPS Conference Proceedings, vol. 38, 1971, 163-168.

[9] S. Tucker, Emulation of large Systems, Communications of the ACM, vol. 8, no. 12, December, 1965, 753-761.

[10] Nanodata Corporation, MULTI Multiprogramming Support System, Williamsville, New York, August, 1973.

[11] J. C. Demco, Principles of Multiple Concurrent Computer Emulation, M.Sc. Thesis, Department of Computing Science, University of Alberta, Edmonton, Alberta, August, 1975.

[12] Nanodata Corporation, CONTROL - Micro-System Control Program, Williamsville, New York, August, 1973.

[13] Digital Equipment Corporation, PDP11 Peripherals Handbook, Maynard, Mass., 1975.

[14] S. Fuller, V. Lesser, C. Bell, and C. Kaman, Microprogramming and its Relationship to Emulation and Technology, Seventh Workshop on Microprogramming, (Preprints), September, 1974, 151-158.

[15] E. Feustel, On the Advantages of Tagged Architecture, IEEE Transactions on Computers, vol. C-22, no. 7, July, 1973, 644-656.

APPENDIX I

INSTRUCTION TIMING

The instruction execution times of the PDP-11 emulator and of the PDP-11/10 [1, 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 SRC Time[1]	PDP-11 DST Time[2]	Emulator SRC, DST Time
0	0.0	0.0	0.16
1	0.9	2.4	3.92
2	0.9	2.4	4.40[3]
3	2.4	3.4	4.96
4	0.9	2.4	4.72[4]
5	2.4	3.4	4.96
6	2.4	3.4	4.40
7	3.4	4.7	5.28

[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 non-modifying instruction (CMPB, BITB, TSTB).
2. add 2.4 usec for a modifying instruction.
[3] - If register is 6 or 7, subtract 0.08 usec. If increment is 1, add 0.08 usec.
[4] - If register is 6 or 7, subtract 0.08 usec. If decrement is 1, subtract 0.08 usec.

BASIC TIME

Single Operand
Instr Time = Basic + DST

Instr.	PDP-11 Basic Time	Emulator Basic Time[1]
CLR	3.4	4.88
COM	3.4	4.88
INC	3.4	4.96
DEC	3.4	4.96
NEG	3.4	5.12
ASR	3.4	5.92[2]
ASL	3.4	6.00[2]
ROR	3.4	6.00[2]
ROL	3.4	6.00[2]
ADC	3.4	5.12
SBC	3.4	5.28
TST	2.2	4.88
SWAB	4.3	6.16

[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.

Double Operand
Instr Time = Basic + SRC + DST

Instr.	PDP-11 Basic Time	Emulator Basic Time[1]
ADD	3.7	5.12
SUB	3.7	5.60
BIC	3.7	5.12
BIS	3.7	5.12
CMP	2.5	5.36
BIT	2.5	5.20
MOV	3.7[2]	5.12[3]

[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.

Branch Instructions

Instr.	PDP-11[1] branch	Emulator branch	Emulator no branch
BGE	2.5	3.20[2]	2.72[2]
BLT	2.5	3.20[2]	2.72[2]
BGT	2.5	3.68[2]	3.20[2]
BLE	2.5	3.60[2]	3.12[2]
BR	2.5	2.64	----
others	2.5	3.12	2.64

[1] - Subtract 0.6 usec if no branch.
[2] - Depending on N and V settings, add 0.0 to 0.48 usec.

Jump Instructions
Instr Time = Basic + DST

Instr.	PDP-11 Basic Time	Emulator Basic Time
JMP	1.0	3.52
JSR	3.8	3.76

Control, Trap, and Miscellaneous Instructions

Instr.	PDP-11 Instr Time	Emulator Instr Time
RTS	3.8	4.96
RTI	4.4	6.56[1]
CLR CC	2.5	4.56
SET CC	2.5	4.56
HALT	1.8	3.92[1]
WAIT	1.8	4.32[1]
RESET	100 msec	4.88[1]
EMT	8.2	8.32[1]
TRAP	8.2	8.32[1]
BPT	8.2	9.36[1]
IOT	8.2	9.36[1]

[1] - Then invoke microroutine.