THE PDP-11 VIRTUAL MACHINE ARCHITECTURE: A CASE STUDY*

Gerald J. Popek and Charles S. Kline
University of California, Los Angeles

At UCLA, a virtual machine system prototype has been constructed for the Digital Equipment Corporation PDP-11/45. In order to successfully implement that system, a number of hardware changes have been necessary. Some overcome basic inadequacies in the original hardware for this purpose, and others enhance the performance of the virtual machine software. Steps in the development of the modified hardware architecture, as well as relevant aspects of the software structure, are discussed. In addition, a case study of interactions between hardware and software developments is presented, together with conclusions motivated by that experience.

Key Words and Phrases: virtual machines, virtual machine monitor, PDP-11/45, computer architecture, computer security

CR Categories: 4.35, 5.24, 6.20, 6.34, 6.35

I. Virtual Machines on Midi-Computers

Virtual machine architectures are now accepted as a valuable approach for the design of large, multiprogrammed computer systems. IBM's VM/370 system is probably the most widely known commercial application of the approach. Surprisingly however, the benefits of virtual machines appear to apply equally well to the small/medium machine community. First, the usual advantages of virtual machines on large computers apply to some degree: on line development of operating systems, concurrent running of multiple operating systems, use of hardware diagnostic software during production hours without service disruption, and general support of software transferability [Goldberg 1974].

However, there are several additional operational gains in having a virtual machine system available for a "midi-computer" such as the PDP-11/45, especially one which is near the top of an upward compatible product line. This environment is one in which numbers of application specific stand alone software packages have been written for bare machines of a given line. Using a virtual machine system on the more powerful, compatible model permits the concurrent running without change of several such software packages, even together with a general multiuser system, each having been written to operate standalone.

For a university environment, such as the one in which the development has taken place, these advantages permit an interesting, cost effective application. In computer architecture and operating system courses, there is considerable pedagogical value in providing students with direct access to those architectural features of a computer which are employed in operating system functions, such as interrupt vectors and relocation registers. Unfortunately, it is rarely practical to do so, since one normally would be required to dedicate a computer, designed and priced to sustain high work loads, to an application whose actual computing needs are trivial. The virtual machine solution provides multiple, logically complete computers, each with the rich structure that is typically hidden by conventional operating systems. The cost of architecturally equivalent machines is thus cut significantly.

---

*This research was supported by the Advanced Research Projects Agency of the Department of Defense under Contract DAHC-73-C-0368.

II. The UCLA Project

An important goal of the UCLA work has been a demonstration that it is now practical to provide a general purpose, multiuser environment whose inter-process protection has reliability far greater than previously available [Popek 1974]. This goal guided the research in several significant ways.

The most important effect on the research concerns the relationship between the need for software compatibility and the necessity for new code. That is, there are enormous investments in existing software, so a practical demonstration of reliable protection must include a feasible method for the preservation of those investments. However, the usual method, selective revision of small portions of existing operating system software to correct security related errors, is generally recognized to be of limited value [Popek 1974] [Weissman 1975]. Newly emerging programming disciplines and secure systems designs, which permit the use of rigorous, formal program certification techniques, are quite promising. However, they are generally applicable only to software for which the methods have been applied at every stage in design and implementation. Therefore, to utilize them, new systems must be constructed, a prospect which is both expensive and potentially incompatible with existing software.

A virtual machine design is an attractive compromise, for it allows the use of existing operating systems and applications software at the cost of only a modest amount of new software [Popek 1974] [Weissman 1975]. See figure 1.

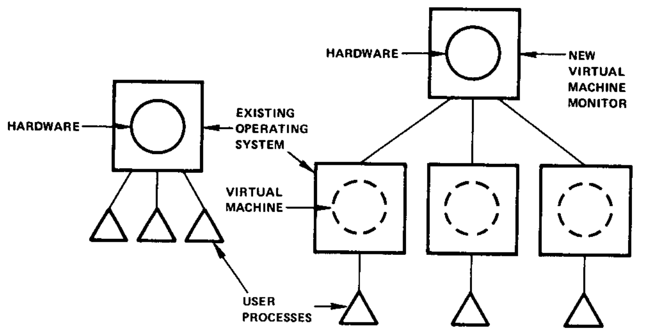

Figure 1. Virtual Machine Encapsulation of an Existing Operating System.

For most of the reasons mentioned above, a virtual machine architecture was chosen for the UCLA work. Perhaps some of the resulting experiences that are reported here can serve a useful pilot study role for secure virtual machine retrofit efforts in more production oriented environments.

A DEC PDP-11/45 was selected as the system's hardware base for several reasons. First, it was a widely employed computer for which a great deal of software was already available. Whatever machine was used also had to serve as a terminal interface to the ARPANET, so the availability of network control software was of particular importance. Secondly, the machine's three state design and UNIBUS I/O architecture were attractive with respect to security, as pointed out later. Since these were the primary considerations, hardware selection was obviously dictated by considerations largely unrelated to virtual machines.

A number of hardware changes were required because the PDP-11/45 is not virtualizable. It was considered a constraint of the project to develop modifications which were inexpensive and would permit all existing software to be run without alteration. Those modifications, their relation to virtual machines, and the lessons learned in design and use, are the primary subject of this paper.

III. The PDP-11/45 Architecture

To understand the UCLA virtual machine system and its architectural support, a brief overview of the original machine architecture will be useful [DEC 1973], [Eckhouse 1975]. The PDP-11/45 is a multistate processor with considerable memory management hardware and an unusual I/O structure, as described below.

The CPU executes in one of three states: kernel, supervisor, or user. Different real sets of relocation registers are employed by the hardware for addressing in each mode, so that it is possible to provide separate address spaces. The relationship among these modes is similar to three hardware supported rings in the Multics sense [Graham 1968], but the separation is not as complete. While kernel mode is the most powerful, and equivalent to the typical third generation privileged mode, the other two modes are not strictly hierarchically nested. For example, the machine has one set of memory locations reserved for interrupt vectors, from whose contents hardware state information is taken when interrupts and traps occur. Since there is only one set, the result of a trap is independent of the mode in which the CPU was operating at the time of the trap. Thus it is not possible to cause user mode traps to transfer control directly to supervisor mode code while supervisor traps transfer to kernel code.

However, in other ways the supervisor address space is protected from user space. For example, special instructions exist in the CPU instruction set for inter-mode communication. It is normally not possible through the use of such instructions to refer to a mode more privileged than the one currently executing. See Appendix I for details. This hardware enforced constraint is similar to some of the checks performed during the crossing of rings in Multics [Schroeder 1973].

Another aspect of the hardware architecture relevant to the UCLA modifications is the UNIBUS I/O structure. The PDP-11 has no channels in the usual sense. Instead, each device effectively has its own simple controller, whose registers appear as memory locations in the upper 4K words of the physical address space.

To perform an I/O operation, CPU device driver software typically moves start addresses, byte counts and status information to the registers as if they were normal memory. Then, that software initiates the I/O operation by setting the start bit in a status register. This structure gives CPU software the direct responsibility for dealing with device details, including any support for device independence. It also means that multiple references to the upper 4K usually must be made in order for an I/O operation to occur. The resulting performance impact will be discussed shortly.

Memory management, the other aspect of the PDP-11/45 architecture relevant here, provides a virtual address space of 32K words (64K bytes), using the word size of 16 bits for the virtual address. There are eight relocation-bounds registers per address space.* After the effective addressing calculation is completed by the hardware, the three high order bits in the resulting address are used to select a relocation register. The contents of that register are then combined with the remainder of the effective address to obtain a physical location.

Unfortunately, this design is known to seriously limit the means by which virtual addresses can be mapped to physical memory. Each 4K portion of a virtual address space can be mapped by exactly one relocation register, since high order address bits select the register. If that register is used to specify a segment less than 4K in size, a "hole" in the virtual address space results. See figure 2.

---

*If an "instruction and data space" feature is enabled, separate register sets are used for instruction fetch and data fetch. The number of active registers, and the size of the address space, is then doubled.

Several additional disadvantages, relevant not only to virtual machine systems, result from such a design. It is not practical, for example, to use memory management to protect interdomain arguments. That is, one would like to pass arguments between processes, or between a supervisor and a process, merely by causing relocation registers belonging to the target process to be pointed at the physical memory areas containing the arguments. Unfortunately however, each such register employed consumes 4K from the virtual address space, even for an argument of small size, because of the "hole" problem described above.

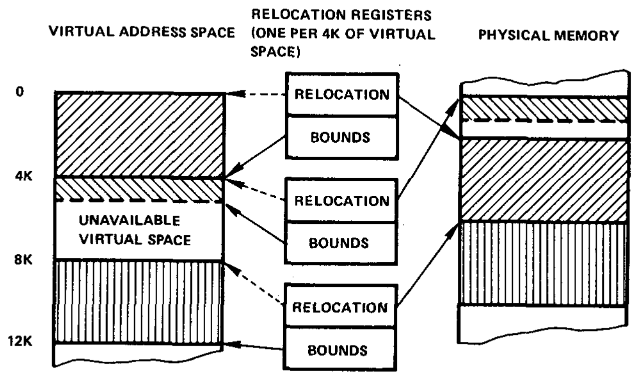

Figure 2. The PDP-11/45 Virtual Address Space Hole Problem.

More serious from the virtual machine viewpoint, it is not practical, for similar reasons, to use memory management facilities to give a program selective access to particular I/O registers. It is not even possible to isolate I/O registers using relocation hardware because the minimum segment size is too large. This limitation potentially imposes a performance penalty of unacceptable proportions, as illustrated later.

IV. The UCLA CPU Modification

A computer must exhibit certain hardware characteristics in order to permit construction of a virtual machine system. Efforts have been made both to develop new architectures specifically designed to support virtual machines and to retrofit existing machines. New architectures are discussed in [Goldberg 1974], while hardware and software methods which have been employed for existing third generation architectures are described in [Buzen 1973]. In these latter cases, the virtual machine monitor typically is run in privileged mode, and all other software in user mode. For this approach to be successful, all "sensitive" instructions must trap when execution is attempted in user mode, so that they can be simulated. Intuitively, a sensitive instruction is one whose execution references or changes the real, rather than virtual, state of the central or peripheral processors. Items such as I/O masks, interrupt vectors, or physical address values are examples of relevant aspects of the hardware's real state [Popek 1974a]. Secondly, the trap must occur in a fashion which allows virtual machine monitor software, run in privileged mode, to simulate the effect of the sensitive instruction.

In the PDP-11/45 processor, there are ten sensitive instructions which do not trap properly. Those instructions, and the difficulties they cause, are discussed in Appendix I. UCLA, together with DEC, designed a set of field installable modifications to the PDP-11/45 processor to make it virtualizable. This modification contains several features. First, the ten sensitive but non-privileged instructions are modified to trap when execution is attempted in other than kernel mode. Next, mode dependent trapping is provided, so that traps occurring while executing in user mode can be sent by hardware to supervisor mode, and traps occurring in supervisor mode transfer control to kernel mode. This feature is provided by adding a second set of trap vectors, located in the kernel address space as the first. The second set is used by the hardware when user mode traps occur.

This facility is especially convenient for the structure of the UCLA software [Popek 1974b], [Fiorani 1975], [Walton 1975]. The system's security kernel is run in hardware kernel mode and the remaining virtual machine monitor code runs in supervisor mode. All other code, including operating systems and their user processes, runs in user mode. The three virtual modes are simulated there.

Lastly, a user mode stack limit register was also added to aid virtualization. The standard hardware already contains one such register for kernel mode. There, a trap occurs if the kernel stack grows beyond the limit indicated by the register. The additional register is valuable because: a) all virtual machine software, including virtual kernel mode code, must run in real user mode; and b) the simulation of its existence in real user mode without hardware support would impose unacceptable performance penalties.

Each of the modifications just described are enabled under program control by loading several additional registers located in the standard device area, at the top of the physical address space. Therefore, selective use of the facilities can be made. Importantly, software prepared for standard PDP-11/45s can be run directly by not enabling the changes.

V. The UCLA Performance Modification

The second modification being made by UCLA addresses performance issues [Vahey 1975]. It was not designed until an operating prototype of the virtual machine monitor was available for measurement, since without hard data it is difficult to predict reliably the precise source of performance overhead. Therefore this hardware is not yet available.

The combination of limited memory management facilities and unadorned UNIBUS style I/O were found to cause considerable difficulty. In a more classical architecture, one or two CPU instructions are executed per I/O operation. On the PDP-11, in addition to normal I/O support code, it is not uncommon for there to be ten or more actual instruction references to the upper 4K for each I/O operation. In order to protect the I/O registers, memory management must be used, to trap any attempt to reference the upper 4K. The reference must then be simulated. In addition, a number of the I/O registers are sensitive, and references to them would require simulation even if the memory management hardware were more selective.* The number of upper 4K references is crucial because each such sensitive instruction must be simulated at a considerable performance cost.

Therefore the overhead for virtualizing UNIBUS-like I/O is much greater than for channel architectures. This overhead is unfortunate since, from both a virtualization and a security viewpoint, potentially only the last reference, which starts the device, need be intercepted. At that point, all the other registers for the given device could be checked, and their contents altered if necessary.** This alteration is analogous to channel program translation in VM/370, except that here the "channel program" is hard wired and it is only the parameters which are being translated. Virtualization would be considerably more efficient if it were possible to selectively give the software running in the virtual machine direct access to most of the I/O registers, completely eliminating any overhead for a majority of upper 4K references.

----

*A virtual machine system built on a IBM 360/40 had a similar problem with low order physical addresses, which contain some analogous information [Bellini 1973].

**This description is a simplification. See the discussion in the following paragraphs.

The need for some form of hardware support for I/O mapping is illustrated elsewhere in the hardware virtualizer presented by Goldberg [Goldberg 1973] and by the use of relocation bounds registers on certain Honeywell channels. The UCLA performance hardware is in some respects a hybrid hardware/software implementation of memory mapping for UNIBUS I/O devices.

The unit actually provides selective access control to the upper 4K. However, since I/O registers must appear to be located in the virtual machine's address space, software must be able to read as well as modify all such registers at any time. The contents of those registers are thus relevant to the virtual machine environment. Unfortunately however, the contents of virtual registers should typically be different from their physical counterparts. For example, a start address in virtual memory is in general different from the actual physical start address, since the relevant segment may be located at an arbitrarily different physical address. Therefore the physical address that must be loaded in the real (but logically program addressable) register to obtain correct I/O operation is incorrect from the viewpoint of virtual machine software. It is for this reason that many I/O registers are sensitive locations, for they reflect the real state of the hardware. Virtual machine instructions which reference those locations will either cause an incorrect result for the virtual machine or affect real I/O, violating protection.

Therefore it is necessary to provide virtual registers which the virtual machine software can reference and modify directly. These virtual registers are actually supported by hardware in the UCLA performance modification. The user program reads and modifies their contents. Only when an I/O operation is to be started does the virtual machine monitor software intervene to translate the register parameters, perform security checks, and load the real I/O registers with the appropriate values to start the real I/O operation. The virtual register contents are left unchanged. Such a hardware support facility, if implementable in a practical way, would substantially contribute to virtual machine performance.

The performance assist hardware incorporates the design described above, and is illustrated in figure 3.

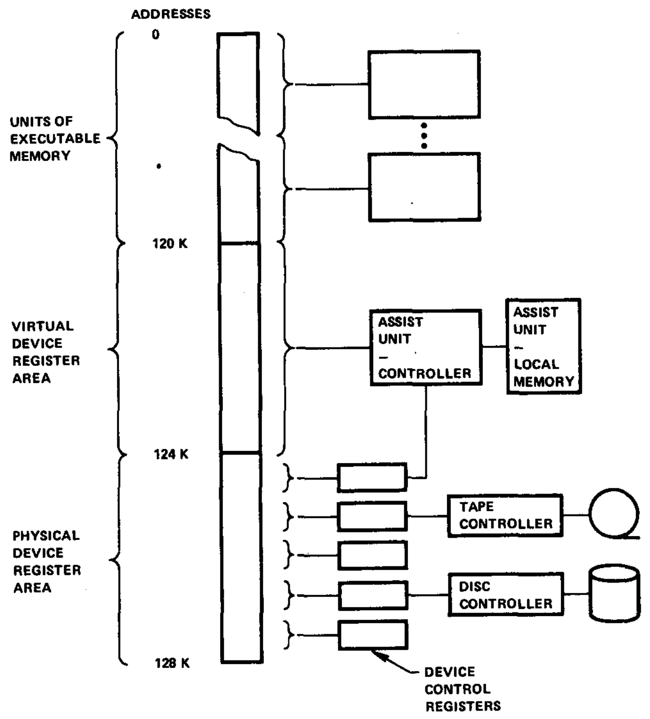
ADDRESSES
0
UNITS OF EXECUTABLE MEMORY
120 K
VIRTUAL DEVICE REGISTER AREA
124 K
PHYSICAL DEVICE REGISTER AREA
128 K
ASSIST UNIT - CONTROLLER
ASSIST UNIT - LOCAL MEMORY
TAPE CONTROLLER
DISC CONTROLLER
DEVICE CONTROL REGISTERS

Figure 3. The UCLA Performance Assist Unit.

The actual hardware is merely another device on the UNIBUS, with control registers residing in the upper 4K of the PDP-11 address space, as for all devices. Memory local to the assist unit maintains virtual register contents. The unit is also designed to occupy the UNIBUS addresses 120-124K just below actual device registers, as if it were main memory. Any instruction which references a real address within the range 120-124K will receive a response from the assist unit. Therefore, a relocation register for a virtual address space can point at the unit's physical address window and a virtual machine's instruction which attempts to reference I/O registers will reference the UCLA unit instead. In this way the unit's memory can simulate the contents of I/O registers. Through the use of the facilities just described, virtual machine programs can read and write, with no overhead, those virtual upper 4K registers which are to contain information such as byte count and start addresses.

There are certain references to device registers for which intervention by VMM software is still required. A good example is a reference to a register containing a device start bit. Therefore, the performance assist unit contains a sensitivity mask bit for each virtual location. After a reference to a given location is complete, if the corresponding mask bit is set, the unit interrupts the CPU in the same fashion as any other device.

When that interrupt is received, VMM software needs to know the I/O registers referenced by the preceding instruction, but the previous program counter is not available for software instruction decoding. (While the new program counter is stacked by the PDP-11 hardware interrupt mechanism, the previous program counter is not uniquely determined, because various instructions are composed of differing numbers of words). Therefore the assist unit also maintains the virtual addresses of the last eight upper 4K references (the maximum number of references possible by a single instruction).

The program status word, (PS or PSW) an I/O register on the PDP-11, presents an additional difficulty, for it contains rapidly changing condition codes. These codes are changed by many instructions, and the resulting PSW can be read by software, just as any other device register. The codes are therefore made directly available to the unit so that reads of the PSW, an operation frequently performed by software such as DEC's standard disc operating system DOS, will operate correctly.

While the I/O area of the physical address space is 4K words in size, any typical configuration has only a few locations in use, the rest acting like non-existent memory when referenced. The performance unit therefore implements a maximum of 128 virtual memory locations, but on a per process basis. That is, several 128 word blocks are provided, with the switching from block to block (process to process) requiring the execution of a single CPU instruction to modify one of the unit's upper 4K control registers. A separate sensitivity mask bit, as well as individual register access controls, are provided for each word in each process block.

As described, the unit would still be insufficient, because of several PDP-11 device hardware peculiarities. In particular, many device registers contain bits which are read only or write only. In order that this aspect of upper 4K references be properly handled, the unit also contains a read mask block and a write mask block, each 128 words long. The appropriate word in the appropriate block is masked by the new hardware with the referenced data word in the process block when that word is referenced by a CPU instruction.

The logical functions just described are rather simple and potentially easily implemented in hardware. A microprocessor and firmware implementation was chosen instead for flexibility. Customized support for unusual devices or specific performance bottlenecks is therefore possible. As an example of the value of that processor, there are a few PDP-11 devices for which a reference to one I/O register causes a change to the contents of a related register. Such specialized behavior can be supported by programming the microprocessor appropriately.

After the performance assist unit is installed, the modified PDP-11/45 should not exhibit a virtualization overhead greatly different from other computers. That is, the amount of CPU overhead per sensitive operation and the types of such operations should be comparable with other architectures. However, the question remains: how often do those sensitive operations occur in the mix of instructions typically executed on the PDP-11, compared with other virtual machine systems, such as VM/370 for the IBM System/370. It appears that there frequently are more sensitive instructions on the PDP-11, in part because it is frequently performing all of its own I/O handling, including the support of full duplex, character oriented terminals, which generate many interrupts. Larger machines often have their own front end processors and other features that minimize sensitive instructions and communications interrupts. In one of the PDP-11 systems measured, the UNIX operating system [Thompson 1974], as high as one percent of the instructions remain sensitive even after assuming that the hardware described here was operating. For such an instruction mix, one can expect software to run slower in a virtual machine than stand alone. However, the actual impact will not be known until measurements are made of the functioning system, since many PDP-11 systems, including DOS and the ARPA network terminal system ANTS, appear to spend much of their time executing sensitive instructions in idle loops, even when heavily loaded. The increased execution time of instructions in these loops is of little consequence.

VI. Conclusions

The following points attempt to capture some of the lessons and insights gained during the still ongoing development of the virtual machine system for the PDP-11/45. Examples, together with related experiences, are discussed below. Perhaps others considering embarking on similar efforts can make use of some of our occasionally painful insights.

1. The construction of a virtual machine monitor is in many respects an excellent way to become intimately familiar with the hardware architecture and details of the

host machine, since it is necessary to mirror all of them on a per process basis. An unfortunate number of peculiarities in the PDP-11 were uncovered, and are listed in Appendix II.

2. Virtualization of the PDP-11/45 is practical, although with more effort than might be expected. In fact, the logical correctness of the virtual machine environment produced in the UCLA implementation is accurate enough that it has even been possible to mirror a number of timing considerations. As a result, nearly all hardware diagnostic software runs without change. Such software usually contains considerable timing dependencies. The packages also provide excellent means to debug the virtual machine implementation, and have been extensively used for this purpose.

3. UNIBUS style I/O architectures are generally inefficient with respect to virtualization since so many I/O registers are sensitive. However, compared to typical channel architectures, they have rather attractive features for reliable security. Channel checking software is often complex and incorrect [Weissman 1975a]. By contrast, each device on the PDP-11 effectively presents a simple, hardwired subroutine for which it is merely necessary to check parameters.

4. Architectural changes contain pitfalls for the unwary. Desires to slide hardware changes "under" an existing architecture arise in a number of other areas. When protection and security are important, for example, capability and domain architectures are often proposed. Proponents are advised however that, despite considerable early effort to forsee difficulties, not all problems were uncovered by the UCLA project until large portions of the detailed design were nearly complete. A few of the hardware peculiarities mentioned in the appendix were not noticed, and the magnitude of certain of the sources of performance overhead were inaccurately estimated. Fortunately these were not catastrophic.

As part of a case study of the unknown troubles which can await one who attempts to adapt an architecture for a task not included in its initial design, the experiences are instructive. Complexities appear in the most surprising of ways. It is little comfort that many of the difficulties encountered, upon closer examination, are architecturally representative and could easily cause trouble for applications other than virtualization.

5. It has been argued that one of the most promising application areas for program verification, at least with respect to cost effectiveness, is in code a) that is frequently executed for many users, and b) whose failure has significant consequences: in other words, segments of operating systems. However, verification methods first require an axiomatic representation of the environment in which the programs of interest run. Operating system code has hardware details, rather than high level programming language constraints alone, as part of its relevant environment. Typical hardware complexities complicate the verification task considerably, precisely at one of the points where it could be so useful. Until hardware architectures are simplified, this impediment is likely too limit the utility of operating system verification.

6. The current software crisis has led to a number of guides for programming discipline, under the label of structured programming. While still in many respects imprecise, the valuable notions of careful layering and clean interfaces appear frequently. Hardware can be considered the zero-th level in a software system. A clean, simply structured interface is valuable there, too. Poor interfaces cause errors and reliability difficulties in any level of a system.

Unfortunately, many hardware architectures do not meet these goals. Economic constraints of course may make it difficult to do so. In fact, questions of cost are also sometimes raised regarding the construction of structured software. However, in both cases it appears that one of the real impacts of the economic pressure may be to postpone and displace costs, rather than reducing them. Poorly structured software exhibits considerably increased life cycle cost. Complex hardware interfaces increase the construction costs of operating systems that utilize them. In sum, arguments applied in favor of clean software also apply to clean hardware.

Acknowledgements

Grateful acknowledgement is made to the following people: Evelyn Walton participated in many insightful discussions; Michael Vahey developed the detailed design of the performance assist hardware; Piero Fiorani and Mark Kampe performed systems measurements; the referees strengthened the paper and improved its tone; and R. P. Goldberg provided valuable criticism.

Bibliography

Bellino, J. and C. Hans, "Virtual Machine or Virtual Operating System", Proceedings of ACM SIGARCH-SIGOPS Workshop on Virtual Computer Systems, March 26-27, 1973, Cambridge, Mass. pp 20-29.

Bisbey, R. and G. J. Popek, "Encapsulation: An Approach to Operating System Security", Proceedings of the 1974 National Conference of the ACM, San Diego, October 1974.

Buzen, J. P. and U. O. Gagliardi, "The Evolution of Virtual Machine Architecture", National Computer Conference Proceedings, AFIPS Press, Vol. 42, June 4-8, 1973, New York, New York, pp 291-299.

Digital Equipment Corporation, "PDP-11/45 Processor Handbook", Digital Equipment Corporation, Maynard, Massachusetts, 1973.

Eckhouse, R., Minicomputer Systems: Organization and Programming (PDP-11), Prentice Hall, Englewood Cliffs, New Jersey 1975, 380 pp.

Fiorani, P., "The UCLA Virtual Machine Monitor", Masters Thesis, UCLA Computer Science Dept., 1975, 460 pp.

Goldberg, R. P., "Architecture of Virtual Machines", National Computer Conference Proceedings, AFIPS Press, Vol. 42, June 4-8, 1973, New York, New York, pp 309-318.

Goldberg, R. P., "A Survey of Virtual Machine Research", I.E.E.E. Computer, Vol. 7, No. 6, June 1974, pp 34-45.

Graham, R. S., "Protection in an Information Processing Utility", Communications of the ACM, May 1968, pp. 365-369.

Popek, G. J., "Protection Structures", IEEE Computer, Vol. 7, No. 6, June 1974, pp 22-33.

Popek, G. J. and C. S. Kline, "Verifiable Secure Operating System Software", Proceedings of the National Computer Conference, AFIPS Press, 1974, pp. 135-142.

Popek, G. J. and R. P. Goldberg, "Formal Requirements for Virtualizable Third Generation Architectures", Communications of the ACM, July 1974, pp. 412-421.

Ritchie, D. M. and K. Thompson, "The UNIX Time-Sharing System", Communications of the ACM, Vol. 17, No. 7, July 1974, pp 365-375.

Schroeder, M. D. and J. H. Saltzer, "A Hardware Architecture for Implementing Protection Rings", Communications of the ACM, March 1972, pp. 157-170.

Vahey, M., "A Virtualization Efficiency Device", Masters Thesis, UCLA Computer Science Dept., 1975, 132 pp.

Walton, E. J., "The UCLA Security Kernel", Masters Thesis, UCLA Computer Science Dept., 1975, 259 pp.

Weissman, C., "Secure Computer Operation with Virtual Machine Partitioning", National Computer Conference proceedings, AFIPS Press, Vol. 44, May 19-22, 1975, Anaheim, Calif., pp 929-934.

Appendix I
Sensitive Instructions on the PDP-11/45

The sensitive but unprivileged instructions on the PDP-11/45 are:

RTI, RTT, SPL, RESET, WAIT,
HALT, MTPI, MTPD, MFPI, MFPD.

The first two, return from interrupt and return from trap, are sensitive because they load the hardware PSW. SPL (set priority level) sets processor interrupt priority, also contained in the PSW. RESET clears all real interrupts, as well as setting device registers in a standard state. WAIT and HALT affect allocation of the CPU. The remaining four are the "M-series" instructions: move to (from) the previous instruction (data) address space in which the CPU was executing, prior to the current one. Their effective virtual address is first calculated using the address information (index registers, indirection, etc.) relevant to the current address space. That address is then used in the virtual address space of the previous mode in which the CPU was running. The contents of the address in that previous mode's virtual space is returned to, or written from, the stack of the current address space. However, it is not possible for user or supervisor mode software to store a previous mode value more privileged than the current mode without direct access to the PSW register, where that information is kept.

In the terminology of [Popek 1974a], all of these instructions, with the exception of WAIT, are behavior sensitive, since their behavior changes depending on the real hardware mode of the CPU. In addition, the first six are control sensitive because they affect resource control.

Appendix II
PDP-11/45 Difficulties Encountered

Samples of the causes of difficulties encountered while developing the PDP-11/45 virtual machine architecture follow.

a) The machine has two instructions, RTI (return from interrupt) and RTT (return from trap), which load the new PSW and program counter. They are typically used to return from interrupt handling and privileged operations. Separately, a hardware trace facility is provided through the use of a bit in the PSW. When that bit is set by loading the PSW via an RTT, one instruction more is executed (the traced instruction), and an interrupt occurs, which presumably will be used to return control to a debugging package. However, if RTI loads a PSW with the trace bit set, the trap occurs immediately. The following problem occurs. Suppose a user program has just set the trace bit, via either RTI or RTT. Suppose an I/O interrupt then immediately occurs. It is serviced by an operating system interrupt routine, and now that interrupt routine wishes to exit. Which instruction should be used, RTI or RTT?

There is no correct choice. If the user executed an RTI, so should the interrupt routine, so that the trap will occur before the next instruction. If the user executed an RTT, the interrupt routine should do likewise, so that the instruction which is to be traced can execute. Since the interrupt hardware saved the new rather than old program counter, it is not even possible for interrupt software to determine the previously executed op code in order to decide what to do. (Recall that instructions occupy a variable number of words and so it is not possible in general to find preceding op codes.) Hence it is not possible for traces to be performed in a deterministic way.

It therefore would also appear to be impossible to virtualize this aspect of the hardware. Fortunately, success is available at some cost in complexity, since the user-executed RTI and RTT are privileged instructions on the modified hardware. It causes each to trap in a fashion that allows the virtual machine monitor to know which case was present.

b) The SPL instruction, which sets the interrupt priority level of the CPU, always inhibits interrupts for one instruction cycle. The 32K user address space is cyclic, so that if a running program fills its virtual memory with SPLs, often easily done through I/O, there will be no way to regain control of the machine without operator intervention. This is a general resource allocation problem, since without care, it will be difficult to guarantee service to more than one process concurrently. It occurred in several second generation computers.

c) Loading and checking the contents of I/O address registers to guarantee that the intended operation affect only the desired memory segment can be surprisingly involved. Physical addresses are 18 bits, but the ADD instruction, used in address bounds calculations, employs only 16 bits, potentially leading to a protection error when the target segment crosses the "17 bit boundary". Further, portions of addresses must be shifted in order to properly load the registers, but the machine has only an arithmetic shift operation, which propagates the sign bit. Additional masking operations are therefore required. For these reasons, the simple logical task of loading an address register can involve a number of errors. Verification that the associated code always operates properly is more difficult than it would have appeared.

d) The HALT instruction traps in a fashion indistinguishable from bus errors and other failure conditions. That is, the normal action of a user program, which should be interpreted, is indistinguishable from an abnormal hardware failure condition, which should be handled by special system action. This behavior alone would make virtualization impossible on the standard hardware. (The problem has been corrected on the PDP-11/70.)

e) The hardware does not make the effective address used by interrupted or trapped instructions available to programs. It is therefore necessary to perform software simulation of PDP-11 addressing in order to determine, for example, which I/O register a process was attempting to reference.

f) The lack of coordination among device registers of the various PDP-11 devices has made device independence much more complex than necessary. Byte counts, start addresses, status bits, and start bits typically all appear in different places within the device register set. When the address in a register is incremented by the device, sometimes carry from the sixteenth to seventeenth bit is implemented, sometimes not. Complex, device specific software is thus required for each device. These details and the problems they cause could have been largely legislated away.

g) No hardware protection is provided to constrain the referencing of main memory by peripheral devices. Therefore an error in setting up I/O register contents potentially allows the device to clobber any memory cell. All protection therefore must be provided by CPU enforced translation and checking of I/O commands.