Protection at the micromachine level

Charlie McDowell

University of California at San Diego
Electrical Engineering and Computer Sciences Department

ABSTRACT

One possible reason that user writable control stores on microprogrammable computers are not used heavily, is the lack of a secure environment in which to run the WCS programs. This paper describes a simple method of adding hardware protection to the micromachine level of a computer.

Introduction

Many computers today have "user writable control store" (WCS). Unfortunately the WCS features of these machines are often not used to full advantage. There are several reasons behind this lack of use: the micromachines are poorly documented, there is a complete lack of reasonable tools for developing microprograms for most of these systems, and also there is reluctance to encourage use of WCS because none of these machines provide any protection against the faulty WCS program or malicious WCS user. In this paper it is shown that there are no reasons why a reasonable amount of protection (equal to that provided by most macrolevel or ISP architectures today) cannot be implemented inexpensively at the micromachine level.

Protection at the machine level.

Most computers today, from 1 chip microcomputers to mainframes, have some type of protection mechanisms implemented in the hardware/firmware. As a minimum, this typically involves some combination of 2 or more processor modes (user/supervisor), a division of the instruction set into subsets that can only be executed when in various modes, and some form of memory protection. This can be as simple as the PDP/11 with user/supervisor mode, privileged instructions (ie. halt) that can only be executed while in supervisor mode and base and bound protection of memory, or it can have many modes and a virtual memory system as is found on the VAX11 computer.

The most recent developments in computer security involve the use of capabilities [2,4]. The construction of efficient capability based computers is a current area of research. Until more experience is gained with capability machines, a simpler method of protection (eg. virtual memory, and privileged instructions) should be provided at the micromachine level to improve the usefulness of user WCS. It would be useful for researchers in the area of capabilities to keep in mind the desirability of protection with capabilities at the micromachine level.

Why never before at the micromachine level?

Up until the last few years, there was never any reason to have protection at the micromachine level. All microprograms were written by "privileged" users, were relatively small, and were carefully(?) tested and debugged before being made available for general use. It was reasonable to assume that there were no malicious microprogramers, and also it was reasonable to assume that every effort had been made to keep the microprograms from accidentally crashing the machine.

These assumptions are no longer true. It is not unusual to find microprograms that are greater than 50K bytes in size. This increase in size, and the expansion of microprograming beyond the traditional bounds of machine instruction emulation, have increased the possibility of both malicious and faulty microprograms, particularly the later. It should be possible to safely use the writable control store on a time shared computer, with the assurance that if your user written microprogram has a bug in it, the integrity of the system will not be jeopardized.

Protection at the micromachine level

Despite the large amount of effort that has gone into providing tools for users of WCS on various machines, the problem of protection has been completely ignored. Roskos and Winner in their paper on user sharing of WCS under Unix are forced to "... leave the problem of bad microprograms for system management" [5]. Most (if not all) production model user microprogramable computers today, have no protection at the micromachine level. That is to say, a WCS routine has the ability to read or write ***any*** piece of machine state, from all of main memory (including the operating system kernel) to any internal register. In most computing environments this prevents most (if not all) users from being allowed to write and execute their own WCS programs. In order for this situation to change, some basic protection mechanisms must be provided at the micromachine level. As was suggested earlier, the very same techniques that have been used at the ISP level can be applied at the micro level.

A very simple protection method could be implemented by having a 1-bit status register that indicates privileged or non-privileged mode. Another possibility might be to put the privilege bit in the control word and let the assembler/compiler control when it is set and cleared. This bit could then be included in the microinstruction decode logic to signal a micro-trap if any privileged micro-operation is attempted without having the privileged bit set. Using just this simple mechanism, on machines with hardware memory protection, a dramatic increase in the usability of WCS would be observed with little or no performance penalty and a very small increase in the amount of hardware.

On machines like the VAX11/780 or PRIME 300, where the memory protection mechanism is built into the hardware (ie. not totally implemented by microcode), memory protection can be realized simply, by carefully selecting the operations that are privileged. In base and bounds systems, it would be necessary to have modification of the base and bounds registers be a privileged operation. In a virtual memory system any registers used to do the virtual to physical mapping should be privileged and any physical memory reference micro-operations should be privileged.

In order to provide some memory protection on micromachines that have no memory protection mechanisms in the hardware, it would be necessary to make all memory reference operations privileged. Depending on the memory protection scheme, this could cause a substantial reduction in execution speed for non-privileged microprograms. However, without some protection, in many

cases these same non-privileged WCS programs would not ever be allowed in the system. This type of system could still be very useful. In some cases even the non-privileged microprogram might be faster than the machine level routine. It would also allow for the testing of WCS routines in non-privileged mode, and then later installing the routines as privileged microprograms. The addition of even a simple base and bounds memory check in the hardware could eliminate most of the time penalty for memory protection at the micro level on these machines.

In addition to main memory protection, there are two types of micromachine registers that should not be accessable to a non-privileged microprogram. The first type consists of registers reserved for exclusive use by the resident or kernel microprogram for storing global information that it needs to have remain from one machine instruction to the next (eg. status information not implemented in the hardware, but needed for machine instruction emulation). The second type consists of registers that are accessable only to privileged machine instructions. This would include such things as memory management registers, interrupt masks etc. To implement this protection at the micromachine level, a subset of the general registers of the micromachine should be marked as privileged. The number of registers to mark, would depend on the intended ISP that was going to be emulated by the resident microprogram. Granted this would reduce the number of registers available to the WCS user, but it would be a small price to pay for the freedom to use the WCS in a general timeshared environment.

An example of a microarchitecture with protection

In this section I am going to outline a set of modifications that could be made to the PRIME 300 [1] in order to provide the same degree of protection at the micromachine level as is currently available at the ISP level.

The PRIME 300 is a typical production micromachine. A block diagram of the PRIME 300 is shown in figure 1. For the purposes of this paper it is only important to notice that there is a 32 word local register file and the ability to force microtraps. These microtraps take the form of unexpected jumps to subroutines with the trapped microaddress pushed on the control store address stack. One other thing that is not shown in the block diagram is that the PRIME 300 supports virtual memory.

The microinstruction word of the PRIME 300 is 64 bits long. Only three fields are of interest here. The R field controls which word of the register file is selected by the ALU and shifter, the M field controls what type of memory action to perform, and the T field controls the masking of traps. The choices for the M field include virtual or absolute, read, write 16 bits, or write 8 bits using either the low or high byte of MDR. Microprogram traps include, memory errors, page fault, and write protection errors.

To provide some protection at the microprogram level, as previously discussed, it would be necessary to add a 1-bit privilege flag P, some control logic for P, and some additional trap conditions. For the purposes of this paper assume that of the PRIME 300's 32 registers, 16 of them (numbers 16 to 31) are to be marked as privileged.

P should be cleared automatically on all traps, and a microoperation provided to restore it to its previous value when returning. The current value of P should be pushed onto the microstack along with the microaddress on microtraps.

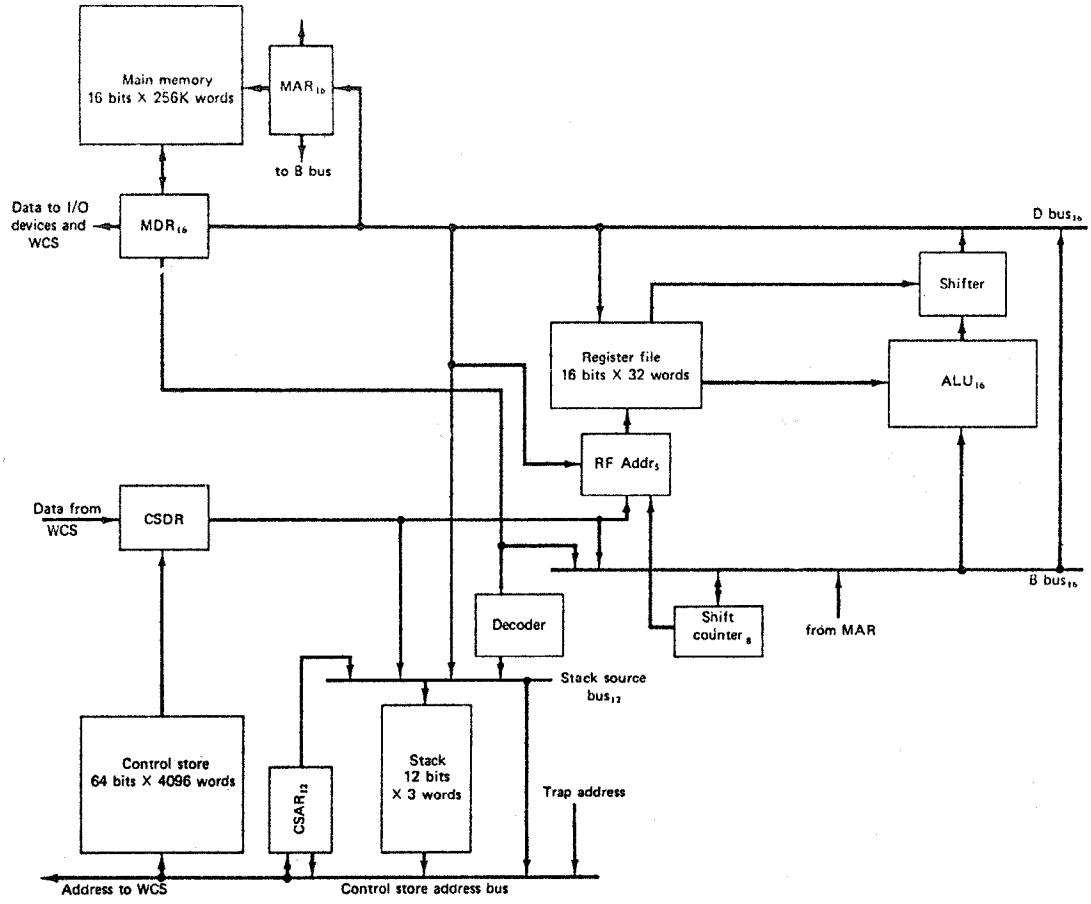

Figure 1
Architecture of the PRIME 300 [1]

A microtrap should occur if any of the following occur while P=1:

1. R field addresses registers 16-31,
2. M field specifies absolute addressing,
3. T field specifies masking of any traps,
4. an attempt is made to clear P.

Conclusion

I have presented here an inexpensive method for building protection into the microarchitecture of a machine, with little or no execution time penalty. This would allow users to write and execute WCS microprograms with the same level of protection that is currently offered by most machines at the normal assembly programming level. An example architecture was presented and the necessary changes were described.

Bibliography

1. Agrawala, A. K. and Rauscher, T. G., *Foundations of Microprogramming*, Academic Press, Inc., New York, 1976.

2. Denning, P. J., "Fault Tolerant Operating Systems," *Computing Surveys*, volume 8 number 4 (December 1976), pages 359-389.

3. Lampson, B. W., "Protection," *Proc. Fifth Princeton Symposium on Information Sciences and Systems*, Princeton University, (March 1971), pages 437-443, reprinted in *Operating Systems Review*, volume 8 number 1 (January 1974), pages 18-24.

4. Linden, T. A., "Operating System Structures to Support Security and Reliable Software," *Computing Surveys*, volume 8 number 4 (December 1976), pages 409-445.

5. Roskos, J. E. and Winner, R. I., "Toward User Sharing of the Microprogramming Level Under UNIX on the Perkin-Elmer 3220," *14th Annual Microprogramming Workshop*, (1981), pages 67-73.