The CERF computer system

by NEIL WILHELM, DAVID PESSEL and CHARLES MERRIAM
University of Rochester
Rochester, New York

ABSTRACT

Multiprocessor systems are becoming increasingly popular because of the increased throughput possible and the possibility of system availability despite the failure of some of the processing units. This paper describes an innovative concept in multiprocessor system design, and suggests some of the research areas which have not yet been resolved but which can be studied on this system. The system architecture is patterned after Control Data Corporation's peripheral processor "barrel" except that it possesses much more powerful functional capabilities. In addition, high speed minicomputers were used to handle all of the system I/O requirements. Research areas for which this system is particularly appropriate include computer architecture-operating systems tradeoffs and multiprocessor operating systems design and implementation. The hardware system is entirely microprogrammable, allowing for increased flexibility in evaluating various research strategies.

INTRODUCTION

Two major questions present themselves in the design of multiprocessor computer systems:

(1) How can operating systems be designed to efficiently handle processor scheduling, memory management and file I/O management on a multiprocessor system? An important aspect of this question concerns the management of the operating system itself: should it be executed by one dedicated processor or should its functions be distributed throughout the system in some fashion?

(2) What are the appropriate architecture and operating systems tradeoffs? Because of reduced hardware costs, it is becoming increasingly possible to implement many ramifications of this issue including the general ability to modify hardware implementations once they are completed without a major cost to the system.

We plan to investigate these questions, with the assistance of a unique tool: our Computer Engineering Research Facility (CERF) Computer System. This computer system, and some of our research objectives, are described on the following pages.

SYSTEM DESCRIPTION

The CERF Computer System is depicted in Figure 1. The primary elements of the system are the four Central Processing Units (CPU's). These are independent, microprogrammable processors, featuring 64 bit data paths and 72 bit microinstructions. Each processor has its own scratchpad of 64 (64-bit) registers, although microinstructions are fetched from a common (and expandable) 1024 storage. Each microinstruction features field extraction and branch capabilities, automatic stacking of subroutine return addresses to a depth of 16, indirect register references with no time penalty, and a large selection of arithmetic and logical operators. The processor clock is 10 MHz, providing 400 ns microinstruction execution times (all microinstructions take the same time).

The CPU's communicate to the external world via two buses, the Stunt Box Bus, and the Memory Bus. The Stunt Box Bus is used for the attachment of specialized hardware, such as multipliers, dividers, target machine instruction decoders, etc. The Memory Bus connects the CPU's to the main memory, which initially will be 128K bytes, arranged as 64-bit words, of solid state memory. Error detection and correction will be provided by an 8 bit Hamming code appended to each memory word.

I/O devices are divided into two categories, high speed and low speed. High speed devices, such as the disc (an 80 megabyte moving-head disc) or drum (a one megabyte fixed-head disc), are each connected to the I/O Bus via two Selector Channels. These channels are extremely fast, special purpose, programmable computers with 16-bit words.

Low speed devices are connected through an Interdata 7/16 minicomputer with 16k bytes of 1000ns primary memory. The card reader, printer and other miscellaneous peripherals are attached to the 7/16 by a
special interface called the Multiplexor Channel. This channel provides these peripherals with direct access to the 7/16’s memory on a time-multiplexed basis. The CPU console is simulated by an interface to the Multiplexor Channel. Terminals and communication equipment are connected to the 7/16 by the Terminal Controller, which is a very fast programmable computer similar to the Selector Channels. The Terminal Controller also provides the path between the 7/16 and the primary memory of the CPU.

In addition to handling the low-speed peripherals, the 7/16 assists in debugging the hardware via the simulated console, and, with its core memory, is a convenient means of bootstrapping the system.

CERF CPU description

The need to have multiprocessing capability with three or more processors poses a number of potential design problems, primarily problems with priorities, interference, and processor lock-out. In addition, having multiple processors generally means having multiple copies of the same hardware. All of these problems can be solved by a design technique similar to one used by Control Data Corporation for the peripheral processors in their 6000 and 7000 series systems. The CERF central processor hardware is divided into four disjoint subsets, corresponding to four stages of microinstruction execution. Thus a processor, at any stage of a microinstruction’s execution, needs only one of the four parts of the hardware, so that four processors can share the same hardware provided each is in a different stage of microinstruction execution. Such an arrangement, with several processors cycling through the same hardware, is often called a “barrel”.

Figure 2 shows the data flow paths of the four processors. Each processor has its own bank of 64 registers of 64 bits each; these are the only elements which are not shared among the processors. The functional elements, which operate on data, consist of the field extractor, which can extract any field of contiguous bits from a word, the field depositor, which can deposit an arbitrary length field into a word, and the arithmetic-logic unit (ALU) which can perform the usual arithmetic and logic operations plus a number of specialized ones. Note that the B-bus, which provides one of the two operands required by the ALU and field depositor, is fed by the field extractor, so that one of the operands could in fact come from any field of a word in a register.

An important issue which must be considered is whether or not the operations shown can be done with reasonable dispatch, so that the processors are fairly fast. Calculations, using manufacturer’s worst-case propagation delay specifications, show that a staging time of less than 100 nanoseconds, i.e., a clock rate of 10 MHz, is feasible. This yields a microinstruction time per processor of 400 ns which, from the rule-of-thumb that 10-30 microinstructions are required for each target machine instruction, implies a target machine instruction execution time of 4 to 12 micro-
seconds on each processor. The overall execution rate would be $10^6$ microinstructions per second, and .3 to 1 million target machine instructions per second.

It is essential that the microinstruction set be adaptable and powerful. To understand what this implies in terms of microinstructions, one must first realize what consumes most of the effort of emulators: decoding the instructions of the target machine. To reduce this overhead the set of microinstructions must include ones for extracting fields of bits from words and for making conditional branches and subroutine calls. All of these objectives are met with the CERF CPU microinstructions.

The design of CERF is presently complete, and it is anticipated that its construction will be complete by June, 1976. Its design has been made in full cognizance of and with careful consideration for the research areas described here. CERF will provide a unique research vehicle, and has been configured in such a way so as to provide a general purpose timesharing system to the University community. This user community is necessary to provide a load on the system during the testing of research problems.

We now describe some aspects of the hardware implementation.

HARDWARE IMPLEMENTATION

The CPU's

The Schottky TTL logic family was selected for the implementation of the CPU's. Emitter-coupled logic (of the ECL 10K family) was considered, but it was rejected because of power consumption, wiring (particularly the need for pulldowns), and interfacing problems (requires level translators to mate with other logic families), because of a shortage of MSI and LSI function as compared with TTL, and because of its high cost. An ECL or ECL-TTL hybrid system could easily run twice as fast.

Recent advances in bipolar memory technology not only reduced circuit complexity over preliminary estimates, but also reduced system cost. Processor register banks are implemented with $64 \times 9$ RAM's (Fairchild 93419) which provide a worst-case access time of 50ns (actual measurements with a 1GHz sampling system show an address-to-output access time of about 30ns). Thus the entire set of four banks of $64 \times 64$ registers requires only 32 integrated circuits. The microstore is constructed from 1024 $\times 1$ bipolar RAM's (Fairchild 93415) with a worst-case access time of 70ns (measurements show a typical access time of about 37 ns), requiring only 72 integrated circuits for a 1024 word storage.

The primary limiting factor in CPU speed is the delay in the ALU-test-branch decision path. Because branch options on every microinstruction are desirable, and because branching should carry no time penalty (since it is done so frequently in microprograms), this path is crucial to the machine's performance. The only solution is to calculate both of the possible successors to each microinstruction, and select the correct one at the last possible instant.

Main memory

Dynamic MOS RAM's were the choice for the main memory, because of their low cost, availability, and reliability. Memory system reliability is enhanced by using an 8 bit Hamming code, which provides single error correction and double error detection, on each 64 bit datum. Thus memory words are 72 bits in length.

Four-way interleaving is used to increase memory bandwidth which, for an approximate cycle time of 500 ns, is 8 megawords or 64 megabytes per second. The useable bandwidth is somewhat less than this. New memory accesses can be initiated while others are in progress, thereby reducing idle time.

Selector channels

The selector channels are implemented with 4-bit slice processor elements (AMD 2901), being essentially very fast 16 bit minicomputers. Each has typically 1024 words of bipolar memory (the same as the microstore) for programs and data. Special "vector" instructions are used to handle the extremely high data rates of the disc drives. Special functions, such as error correcting coding, are implemented in hardware.

RESEARCH GOALS

Most conventional operating systems are almost exclusively oriented towards the concept of single processor systems. Although these operating systems may allow multiprogramming, the ramifications of multiprocessing are generally avoided. Multiprocessing systems have been examined in limited detail elsewhere. This study will consider in depth various aspects of multiprocessor operating systems. In particular, we will study whether one operating system can effectively and efficiently coordinate the activities of more than one processor in a multiprocessor system. In a multiprocessor system each processor may have an independent address space mapped onto one physical memory, or independent memories may be available to each processor. In either case, the operating system is now faced with the task of coordinating various memory maps. Similarly, each processor in a multiprocessor system may execute programs from independent job streams or there may be only one job stream being scheduled onto all of the processors. In both cases, the operating system must handle scheduling of a far more complex form than on a uniprocessor system. Finally, a separate file and I/O system may exist for each processor. This results in yet another set of operations more complex than on conventional sys-
tems. Various problems may arise as a direct result of this type of system. Among these are: (1) primary memory contention: since more than one simultaneously active processor and task can address the same memory module, techniques must be explored to reduce or avoid memory contention at that module, and (2) file protection: adequate file protection is complicated because of the existence of simultaneously active processors and tasks which may access the same files. File structures which insure data integrity must be developed and implemented.

A second area to be considered concerns architecture operating systems trade-offs. An operating system is very dependent upon the architecture (i.e., the instruction set, registers, etc.) of the processor on which it is run. This relationship is so close that not only the effectiveness and efficiency of the operating system but the very structure of the operating system is determined by the processor architecture. For example, the “cactus stack” design of the B-6700 leads naturally to the tree-like hierarchy of processes used by the operating system. Of course one can always impose an operating system design on an unsuitable processor, at the risk of losing efficiency.

We can delineate three basic areas in which the operating system-processor interaction is crucial. These are processor scheduling, memory management and file and I/O management. For each of these areas, trade-offs can be made between performing important functions as “hardware macro’s” (e.g. as special processor instructions or using dedicated processors) or as “software macro’s” (i.e., aggregates of primitive hardware instructions). We can increase system speed by using more “hardware macro’s” in exchange for greater processor cost and complexity and reduced system flexibility. On the other hand, we can reduce system cost and increase flexibility by using “software macro’s”, but at a reduction in efficiency.

Existing systems are examples of a priori decisions regarding the distribution of functions between hardware and software. For the IBM S/360, essentially all operating system functions are implemented without hardware assistance. On the other hand, the Berkeley Computer Corp. BCC-500⁶ (now at the University of Hawaii) features specialized processors performing such functions as scheduling and memory management.

We propose to investigate, theoretically and empirically, several specific areas of operating system architecture trade-offs, using the CERF multiprocessor computer system as a tool. One of these areas is the design of operating systems for multiprocessors, emphasizing the synchronization primitives and the architectural features necessary to support them. Hardware provisions for mutual exclusion are especially important in a multiprocessor system, since the simple expedient of turning off the interrupts and simulating mutual exclusion primitives in system software works only for a uniprocessor system. Since P and V operations may imply changes in processor scheduling, the relationship between the hardware and software is very close at this point.

Another important consideration is the distribution of the operating system functions to various system processors. Should important functions such as processor scheduling be shared among the central processors (as in the original plan of the CMU Hydra system), should one of the central processors be dedicated to these operations, with the other processors as slaves, or should these functions be distributed to more specialized “peripheral processors” (as done in the BCC-500 and the CDC 6000 and 7000 series)? The CERF system is ideally suited for this type of investigation. One could write special microcode for one of the central processors, making it the master. On the other hand, the drum controller can be reprogrammed to handle the memory management, or the Interdata 7/16 could be programmed to perform processor scheduling.

Our current plan, now under way, is to develop an initial target architecture which is “extensible” so that we may add the features we need with relative ease, and is also well-structured, so that ALGOL-like languages can be fully-supported and compiled efficiently. Because the basic machine architecture will remain relatively fixed, we can expect to get meaningful performance comparisons as we explore the various possibilities.

CONCLUSION

We have described an innovative approach to multiprocessor system architecture, yielding greatly increased hardware flexibility and applicability at only a slight increase in cost. This system provides a unique vehicle for research in multiprocessor architecture and operating systems. The research objectives have been briefly described, and the results of these studies will be presented in future publications.

ACKNOWLEDGMENT

The authors gratefully acknowledge the support of the College of Engineering and Applied Science in funding this project.

REFERENCES