CMU-CS-78-146

79-00055

(1)

The C.mmp Multiprocessor

S. H. Fuller
Digital Equipment Corporation

S. P. Harbison
Carnegie-Mellon University

October 27, 1978

DEPARTMENT
of
COMPUTER SCIENCE

Carnegie-Mellon University

CMU-CS-78-146

The C.mmp Multiprocessor

S. H. Fuller
Digital Equipment Corporation

S. P. Harbison
Carnegie-Mellon University

October 27, 1978

Carnegie-Mellon University

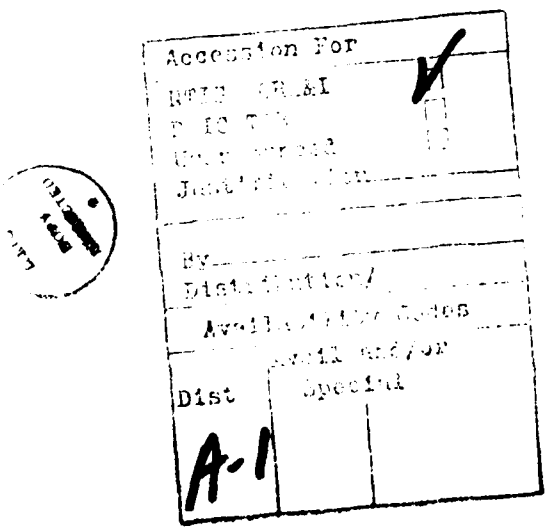
Computer Science Department

Abstract

The C.mmp multiprocessor consists of sixteen minicomputers connected to a large shared memory through a central crosspoint switch. The system was constructed beginning in 1971, and for several years has acted as a research vehicle for investigating problems in the exploitation of multicomputer structures. This paper is a description of the existing hardware system.

This research was sponsored by the Defense Advanced Research Projects Agency (DOD), ARPA Order No. 3597, monitored by the Air Force Avionics Laboratory Under Contract F33615-78-C-1551.

Table of Contents



1. Introduction

In 1971, we at CMU became interested in research issues surrounding the construction and use of multiprocessors. For that reason, we undertook the design and construction of C.mmp, a large multiprocessor constructed of minicomputer processors. C.mmp has now been in use for several years, and many of the research results we originally hoped for have been published. More are on the way. However, most of C.mmp's exposure in the computer science literature to date has been via discussions of *Hydra*, C.mmp's operating system. Hydra implements a capability-based protection scheme which establishes it as a research project in its own right. The only detailed paper on C.mmp itself is [3], which discussed the design before actual implementation had taken place. We feel it is important to describe here the actual hardware that has resulted.

C.mmp[1] is archtypical of a simple multiprocessor; it consists of a number of equal, asynchronous central processors that share a large primary memory. C.mmp differs from earlier multiprocessors such as the Burroughs D825, IBM 360/67, Honeywell 645 (Multics), etc. in two essential respects:

1. C.mmp is designed to have up to sixteen processors while other multiprocessors usually have no more than four processors.

2. C.mmp is constructed from minicomputer processors rather than the larger (32 to 48 bits/word) processors used in the other systems.

In other words, the effective use of C.mmp requires that we find and exploit a much higher degree of parallelism than has been needed by earlier multiprocessors. In the past few years, the number of existing multiprocessors has increased signifigantly to include BBN's Pluribus, Stanford's S-1, and CMU's CM* systems. However, C.mmp still remains notable for its uniform structure and support of a general-purpose operating system. Moreover, performance studies of C.mmp provide calibration for similar data from these newer machines.

2. PMS Structure

The C.mmp computer system is pictured in figure 1 and its PMS organization is shown in Figure 2. As may be seen, the principal components are sixteen modules of shared memory ($Mp_{0:15}$), a 16 x 16 switch (Smp), sixteen processors ($Pc_{0:15}$), associated input/output

---

[1] Pronounced "See-dot-em-em-pee", it is an acronym from the PMS notation of Bell and Newell[4]. PMS (Processor-Memory-Switch) notation is used to describe computer systems at the 'block diagram' level; C.mmp stands for multi-mini-processor Computer system. Other frequently used PMS names include Pc for central Processor, Mp for primary Memory, K for Kontroller, and L for Link.

devices (Kio), special relocation hardware (Dmap) associated with each processor, and an interprocessor bus (IP-bus) with special devices attached to it (K.interrupt, etc.). We will examine each of these major components in greater detail below; first we consider the overall organization more thoroughly.

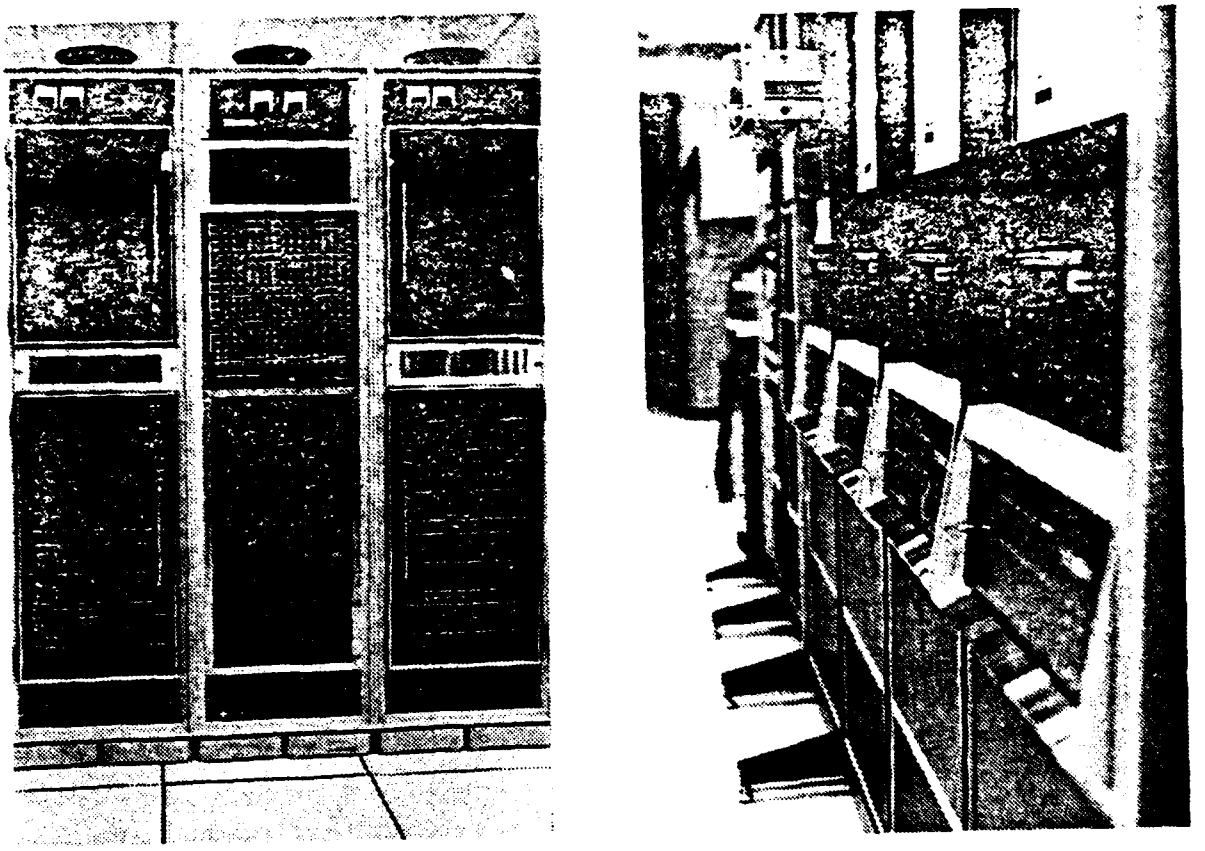


Figure 1: C.mmp: Smp and memory (l.) and four processors (r.)

The area enclosed by dashed lines in Figure 2 represents a complete minicomputer[1]: a Digital Equipment Corporation PDP-11 processor, memory (M.local), and input/output devices. These minicomputers are connected to the shared memory through the relocation hardware (Dmap) and switch (Smp). They are connected to each other and to a few common devices through the interprocessor (IP) bus.

[1]The minicomputer processors are slightly modified; we will discuss the necessary modifications later.

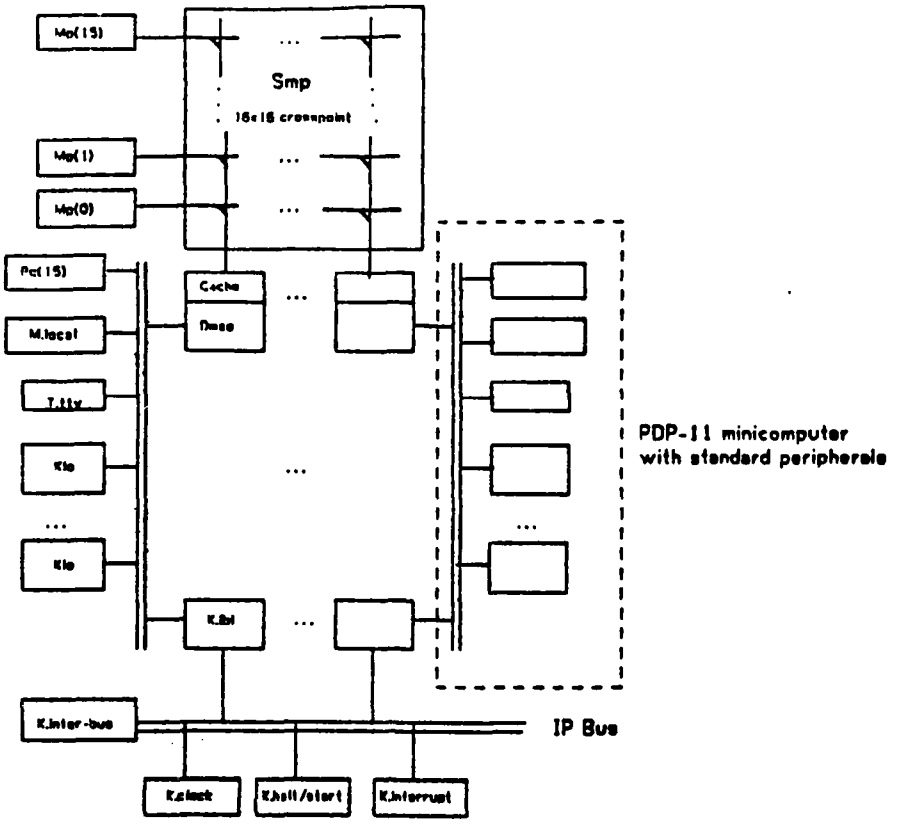


Figure 2: PMS diagram of C.mmp

All communication between components of a single PDP-11 is performed over a single bus, the UNIBUS. The processor accesses primary memory via the UNIBUS as do direct-memory-access (DMA) I/O devices. The processor also communicates control information to I/O devices on this same bus. Specifically, the control registers of I/O devices have UNIBUS addresses and appear as memory locations to the processor; the processor controls the operation of I/O devices by simply 'storing' appropriate control information into these locations -- thus there are no special I/O instructions needed in the processor [5].

Although the primary virtue of the UNIBUS is that it has allowed the cost-effective implementation of minicomputer systems, it has some other benefits in a system such as C.mmp. The DMA devices use the UNIBUS to transfer data just as the Pc does and hence the

Dmap can manage Pc and DMA requests to shared memory in a uniform manner. This is in contrast to many conventional virtual memory architectures (e.g. the IBM 360/67) which require physical (unmapped) addresses for DMA requests.

In making the design descisions described below, we were guided by a desire to make C.mmp as symmetrical as possible. All components of the system were envisioned as a pool of resources to be shared among whatever tasks were to be done. This was to be (is) true of processors, I/O, and memory singly and in combination. There was to be (is) no master-slave relation between the processors, and any user job was to be (is) able to execute on any processor at any instant. The same was to be (is) true of the operating system; virtually any portion of Hydra may execute on any processor.

To achieve the desired symmetry at the software level one must begin with symmetry at the hardware level. There are two aspects to this symmetry: access to the shared memory, and the passing of control signals between processors. The relocation hardware and switch provide the first, and the interprocessor bus provides the second.

2.1. The Processor-Memory Switch

Smp, the processor-to-memory crosspoint switch, handles single word transfers between the shared primary memory and the processors. Transfers from Kio units also access memory through Smp. The memory and switch for C.mmp can be thought of as a memory with 16 ports and consisting of 16 independent memory subunits. Up to 16 simultaneous accesses to Mp are possible if 16 Pc's are active and if they all request words in a different memory port.

Unlike most other crosspoint switches, this one is located centrally, as opposed to being distributed in the memory, e.g., as in the DECsystem10 or IBM 370/168. While this requires a larger initial configuration and implies some non-modularity, the cost of a complete system is less and we are interested in rather large configurations. A large cost component of a switch system (together with the associated mechanical and circuitry problems) is the cables. This structure requires only 16 + 16 cables, as opposed to 16 x 16 cables required for a fully connected distributed switch. A centralized switch also has less cable delays than a functionally equivalent distributed switch. BBN's Pluribus uses a distributed switch and hence provides an interesting contrast to C.mmp. See [6] for a discussion of why reliability and modularity arguments dictated the use of a distributed switch for their application. CMU's CM* multiprocessor uses a heirarchical structure and a packet-switching network to allow for unlimited expandability [7].

Another important aspect of Smp is the control provided to facilitate reconfiguration and

partitioning of the system. Each of the 256 crosspoints of Smp can be enabled or disabled either manually (from a front panel shown in Figure 3) or under program control (by setting a flip-flop addressable from a UNIBUS). For a crosspoint to be operational in the Smp, both its manual switch and its flip-flop must be 'set'. Thus, for example, a faulty Pc can be removed from the system by disabling its column of crosspoints, and C.mmp can be partitioned into two 8 x 8 configurations by disabling the 128 crosspoints that are in the first and third quadrant of the crosspoint array.

Dynamic reconfiguration is employed by the operating system to eliminate faulty hardware on the basis of continuously-gathered error histories. Manual reconfiguration may be used when a separate system is necessary for off-line testing or other maintenance.

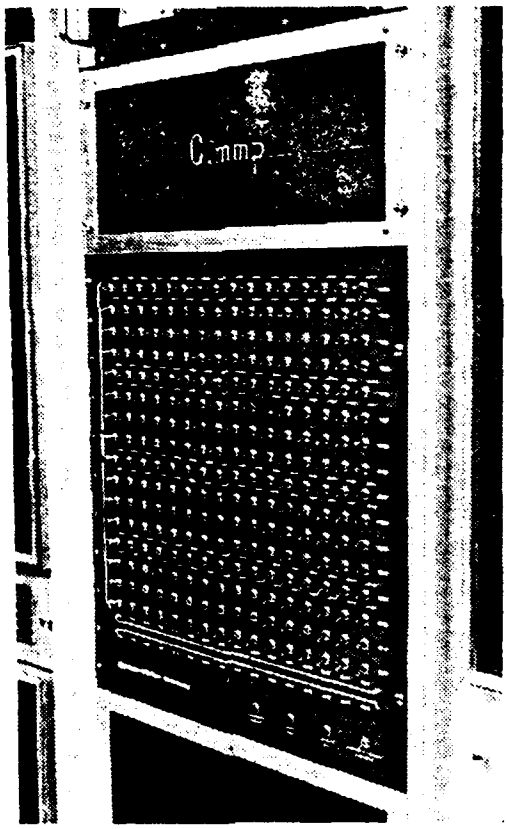


Figure 3: Crosspoint switch control panel

2.2. Memory Mapping and the Relocation Unit

Probably the greatest problem in building a large computing system from minicomputers is their small address space[2]. In C.mmp we must be able to address several million bytes of primary memory from the processors. The basic PDP-11 architecture, on the other hand, is only capable of generating 16-bit addresses and is thus limited to a 64K-byte address space.

Note that this is exactly the opposite situation faced by many operating systems which map large virtual addresses into somewhat smaller physical ones. The goal of these systems is to keep a portion of each process' virtual memory addressable, the so-called *working set* of the process. The working set consists of those pages most recently referenced by the process; experience indicates that future references will, with good probability, also lie within these pages.

In a minicomputer environment this working set concept can be turned 'inside out' to produce an address mapping scheme where the user must explicitly state which pages are in his working set and when changes in addressability are required. This notion serves as the basis for the address mapping (relocation) scheme adopted for C.mmp.

The PDP-11 processor (and thus programs executing on it) generates a 16-bit address. The UNIBUS, however, supports an 18-bit address, and the shared memory uses a 25-bit address. Somewhat arbitrarily we chose to divide these address spaces into 8K-byte units called 'pages'. Thus processor-generated addresses are divided into eight 8K pages, UNIBUS addresses are divided into 32 pages, and the shared memory is divided into 4096 pages.[1] In going from the 16-bit user address to the 18-bt UNIBUS address, the two extra bits are obtained from the Program Status register (PS) in the processor (see Figure 4). As we shall see in a moment, these bits may not be altered by a user program, and thus user programs are actually bound to operate within the eight pages described by a subset of the relocation registers.

The relocation hardware is attached to the UNIBUS and responds to 30 of the 32 possible UNIBUS page address ranges. These 30 pages are mapped from the UNIBUS address into primary memory by means of 30 hardware registers; the mapping is quite straightforward. The five most significant bits of the UNIBUS address are used as an index into this set of

---

[1] The memory mapping mechanism for C.mmp that is being described is not related to any of the memory mapping schemes available on other PDP-11's (ie. the PDP-11/40, 11/45, and 11/70) [8]. A few of the outstanding differences between memory management on C.mmp and other PDP-11's include: the relocation registers in C.mmp map from the (virtual) UNIBUS address space into the (real) shared memory address space while other PDP-11's map from the (virtual) processor address space to the (real) unibus address space; C.mmp maps addresses generated by the I/O units, while other PDP-11's do not map addresses from I/O units; and C.mmp has fixed size pages while other PDP-11's allow pages to be variable size.

registers. A 12-bit field from the selected register is left-concatenated to the remaining 13 bits of UNIBUS address, thus forming a 25 bit address in primary memory (see Figure 4). Should the five most significant bits specify page 30 or 31 (i.e. non-existent mapping registers) the mapping hardware simply ignores the bus cycle. These two pages are reserved for the processor's private local memory and access to its peripherals' registers.

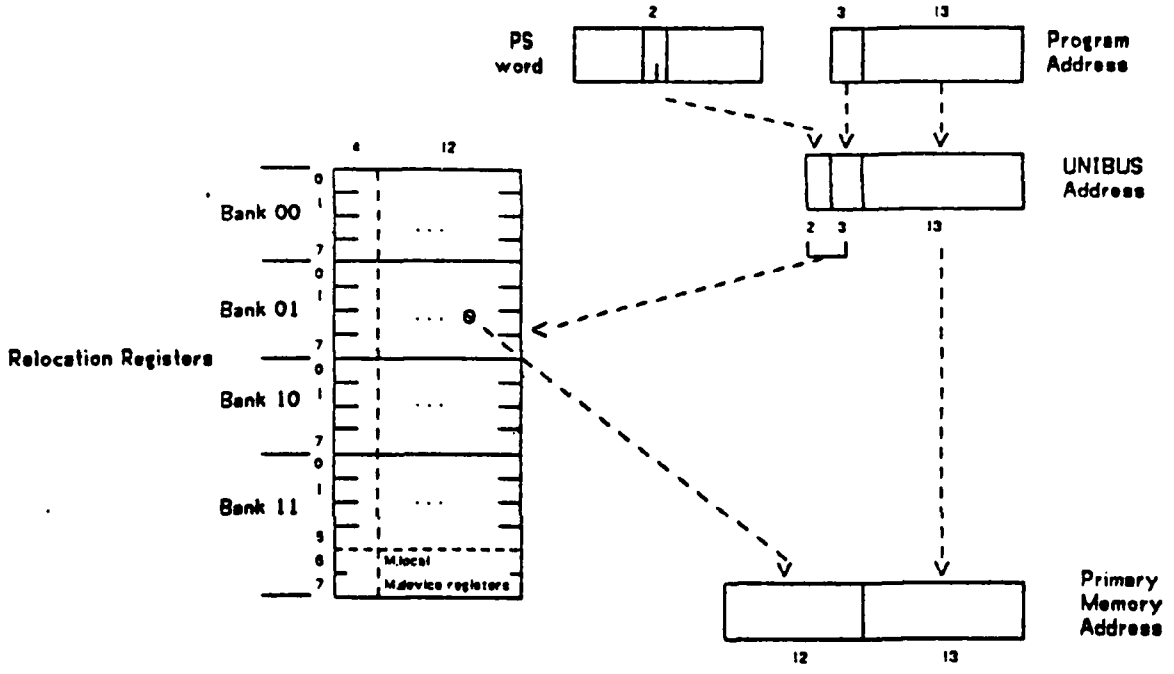


Figure 4: Address translation in C.mmp

Each relocation register also contains a field of control and status bits, as shown in Figure 5. The *non-existent memory bit* can be set by the operating system to prevent access to shared memory through the register. This permits the system to place a small user job in the machine without allocating a full 64K-byte address space. The *write-protect bit*, when set, permits read cycles to proceed through the register but blocks write cycles. This feature can be used to guarantee the integrity of code pages. The *written-into bit* (or *'dirty' bit*) in a register is set to '1' by any write cycle throught that register. This mechanism is used by the operating system to avoid writing out pages which have not been altered. The *cacheable bit* is used in conjunction with those processors that have a cache (see section 2.3) to indicate which pages may be buffered.

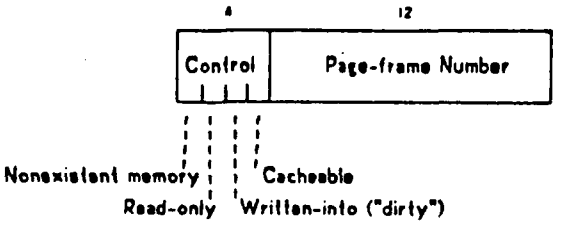


Figure 5: Format of the relocation registers

2.3. Caches

C.mmp has a provision in its design for 1000-word, per-processor caches on all the PDP-11's. As of November 1978 only one cache was installed, and it is not used by the operating system. Therefore, we must describe the intended operation of the cache.

Caches present a special difficulty for multiprocessor systems because data shared between processors may be modified in one processor's cache without the modification being reflected to other processors. Our caches implement write-through to shared memory, but the contents of caches on other processors are not affected. C.mmp's solution to this problem is to have the operating system designate (via the cacheable bit in the relocation registers) which pages are safe to cache. Fortunately, studies on the PDP-11 indicate that about 70% of all memory references are to code pages, which can be forced by Hydra to be unmodifiable and hence cacheable. Stack pages (see section 2.4.2) are guaranteed private to a processor, and hence are also cacheable.

The caches designed for C.mmp are not fast; their importance lies in their ability to eliminate switch contention by catching a significant fraction of the memory fetches. This is especially important because Hydra encourages the sharing of code pages among cooperating processes, thus inviting significant contention. (See [1] for a discussion of the effect of this contention and how we dealt with it in the absence of caches.)

2.4. Processor Extensions

An important goal in almost every operating system must be the protection of itself and of

other users should one of the user programs fail.[1] Attaing this goal involves many considerations; the execution of a 'halt' instruction by the user is an obvious example of an action which must be prohibited. A less obvious example is the alteration of the stack pointer such that an interrupt would cause the overwriting of operating system code or tables.

2.4.1. Address Spaces

A central aspect of the design of the processor modifications is the partitioning of the 18-bit UNIBUS address into four *address spaces*. The natural partitioning is to associate each space with a particular configuration of the two bits in the PDP-11 PS register which form the high-order two bits of the address. Programs not executing in '11'-space cannot alter the PS (thus changing their space) since the PS register itself is addressable only in '11'-space, as are all device registers and interrupt vectors. We therefore designated '11'-space to be the *kernel space*, in which only operating system code would execute. '00'-space became *user space*, where all user code executes, and the remaining two spaces were reserved for DMA I/O traffic and special applications.

The PDP-11 provides a reasonable method for transfering control between spaces. Executing one of several 'trap' instructions, or the occurance of any hardware interrupt, causes the current (PS,PC) pair to be stacked and a new (PS,PC) pair to be fetched from a fixed address in kernel space. Typically the new (PS,PC) forces control to pass to operating system routines in kernel space. The execution of a 'return-from-interrupt' instruction (RTI or RTT) reverses this process, fetching the old (PS,PC) from the stack.

2.4.2. Handling the Stack

The PDP-11 has several addressing modes which facilitate managing a stack, and programming convention dictates the use of a standard stack area for interrupt processing, subroutine calls, and parameter passing. This stack area is by convention pointed to by a particular PDP-11 register called the *stack pointer (SP)*.

The stack introduces some problems in switching address spaces, since the stacking of the old (PS,PC) at interrupt time occurs in the old (e.g. user) space while the unstacking by RTI or RTT occurs in the new (kernel) space. Several solutions to this problem are available, the most obvious being the addition of mechanisms to retrieve data from the 'previous' space. (Not a trivial task, since we must provide for multiple nested interrupts.) For C.mmp, however,

---

[1] A failure in a user program can result from improper data, incorrect program or malicious intent. Regardless of the cause of failure, the operating system needs hardware support to protect itself as well as other user programs.

we decided to force all address spaces to use the same stack[1]. We do this by first establishing the convention that the low-order 8K bytes of each address space are to be used for the stack. Processor modifications force the SP to point to an even address in this page (except when executing in kernel space), and the relocation registers are modified so that the stack page register in each space holds the same value. Having the operating system and the user share the same stack makes changing address spaces easy and allows users to pass arguments to the OS simply and efficiently. However, it does present some protection problems.

A programmable *stack underflow register* is used by the operating system to prevent users from accessing data belonging to their callers or to the operating system. A fixed *stack limit* further restricts the stack, defining an area in the lower portion of the stack page which can be used for the communication of global information between the operating system and the user.

Finally, PDP-11 programmers will remember that some hardware trap vectors ((PS,PC) pairs) are located in low addresses which on C.mmp would be in the stack page. We relocate these UNIBUS addresses by OR'ing #740000 with them, placing them in kernel space in the local memory associated with each processor.

2.4.3. Instruction Set Modifications

The HALT, WAIT, and RESET instructions were made illegal in any but the kernel space. Likewise, RTI and RTT were made illegal since they obtain the new (PS,PC) from an area addressable by the user. The trap instructions (TRAP, EMT, IOT) are legal from any space since they obtain their new (PS,PC) from a protected area and the operating system can verify the environment at the time of the trap.

2.4.4. Extensions for Error Detection

A principle advantage of multiprocessors is their (potential) ability to withstand various types of hardware errors by isolating and eliminating faulty components. C.mmp in particular is insensitive to both the number of processors and the number of memory pages actually present, so reconfiguration techniques are studied carefully. The PDP-11 minicomputer did not have adequate error-detecting capabilities, so we augmented it with several mechanisms of our own.

---

[1] Experience with the Image Understanding System at CMU indicates that separate user and operating system stacks may be better in general. However, C.mmp was initially constructed from PDP-11/20 processors, on which the necessary modifications would have been more difficult than on our (and IUS') current PDP-11/40E models. Retrofitting the change now is impractical.

The most significant improvement was the implementation of parity bits in shared memory. The relocation hardware computes parity bits for each byte written to memory and for every address sent to the memory. To catch common failure modes of 'all ones' and 'all zeros,' we use even parity on one byte of each data word and odd parity on the other byte. Address parity is checked by the memory controller on the 'far' side of the switch, and data parity is checked on each 'read' cycle by the relocation hardware at the processor.[1]

The PDP-11's variable-length instructions and its rich set of addressing modes makes locating the exact source of an error (e.g. a parity error) difficult. For this reason, we implemented two *tracking registers*. The *bus address tracking register* is latched upon the occurance of a switch-detected error (e.g. a data or address parity error), and thus accurately specifies the UNIBUS address causing the error. The *PC tracking register* latches the address of the current instruction under the same circumstances, thus providing the information needed to retry an instruction.

Maintenance functions are also implemented, including the ability to simulate address parity errors and the ability to write incorrect parity into shared memory. There are also facilities to address the cache memory as a normal RAM by making it respond to a specified shared memory page.

2.5. The Interprocessor Bus

Interprocessor communication is an important consideration in controlling a multiprocessor. Furthermore, to qualify as a symmetric multiprocessor it is necessary for each processor to control these functions on every other processor. This might, however, require as many as 120 cables among the sixteen processors (for the number of functions considered, this would lead to more than 1000 wires). In order to simplify the situation we have designed an interprocessor bus, a controller for it, and interfaces to it. These are shown in the PMS diagram of Figure 2.

The interface allows a processor to invoke a certain function on any subset of the processors, including itself, by simply 'ORing' a mask into the interface register associated with that function. The interface currently contains six such registers, one each for HALT, START and CONTINUE and three for different levels of priority interruption. Each of these function registers is 16 bits wide. Setting the *i*th bit of the register associated with one of the functions will invoke that function on the *i*th processor. Thus for example, moving a mask

---

[1] Actually, the switch has data paths wide enough for error-correcting codes on each data word, but such a mechanism has not been implemented.

of all 1's into the HALT register will stop the entire machine.

In addition to these functions, the IP bus provides other facilities to the processors, such as a (per-processor) programmable interval timer. Each timer consists of a time count register and a control register. The operating system can store a value into the count register which will be decremented every 16 microseconds as long as the run bit is set in its control register. Additionally, the control register can cause an interrupt to be generated when the count register reaches zero. Because the interrupt might not be serviced right away, the count register keeps decrementing so that precise timings can be obtained. Furthermore, should the count be decremented to zero again before the interrupt is serviced, a status bit in the control register is set to indicate counter wrap-around.

Finally the interface provides access to the 56-bit, one-microsecond-resolution, time-of-day clock. The interprocessor bus controller, which will be described below, continuously broadcasts this clock value on part of the bus. When a processor wishes to know the time, it reads the first of the four registers in the interface. This causes the interface to load all four registers from the interprocessor bus. The processor can then read the other three registers without fear of the value changing. The interface alters the clock value by concatenating it with the processor number. This is important to provide uniqueness since Hydra uses the clock value as a unique name generator.

Corresponding to Smp's switch panel, the IP-bus interface on each processor is equipped with an operator panel (Figure 6) which permits the IP-bus to be paritioned in the same way as the memory crosspoint. The IP-bus panels also contain status lights indicating the presence of the various error conditions implemented in the relocation hardware.

3. The Present C.mmp Configuration

The PMS diagram of C.mmp in Figure 2 is a conceptual diagram intended to help introduce the control features of C.mmp; it clearly is not intended to describe an actual C.mmp configuration at any point in time. However, a number of interesting issues arise in configuring C.mmp and Figure 7 is a PMS diagram of the current C.mmp configuration.

3.1. Processors •

There are eleven PDP-11/40E processors currently on C.mmp. The 11/40E differs from the standard 11/40 in having a 1000-word x 80-bit writeable control store, allowing us to tailor the instruction set to special applications and to the operating system. We have used this facility to implement various block-transfer instructions, floating-point instructions, and some special instructions to speed up a simulator for the CM* multiprocessor.

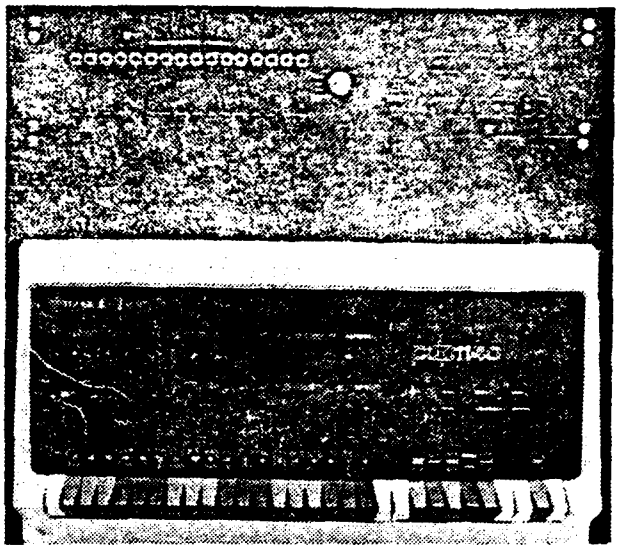


Figure 6: PDP-11/40 with IP-bus interface panel

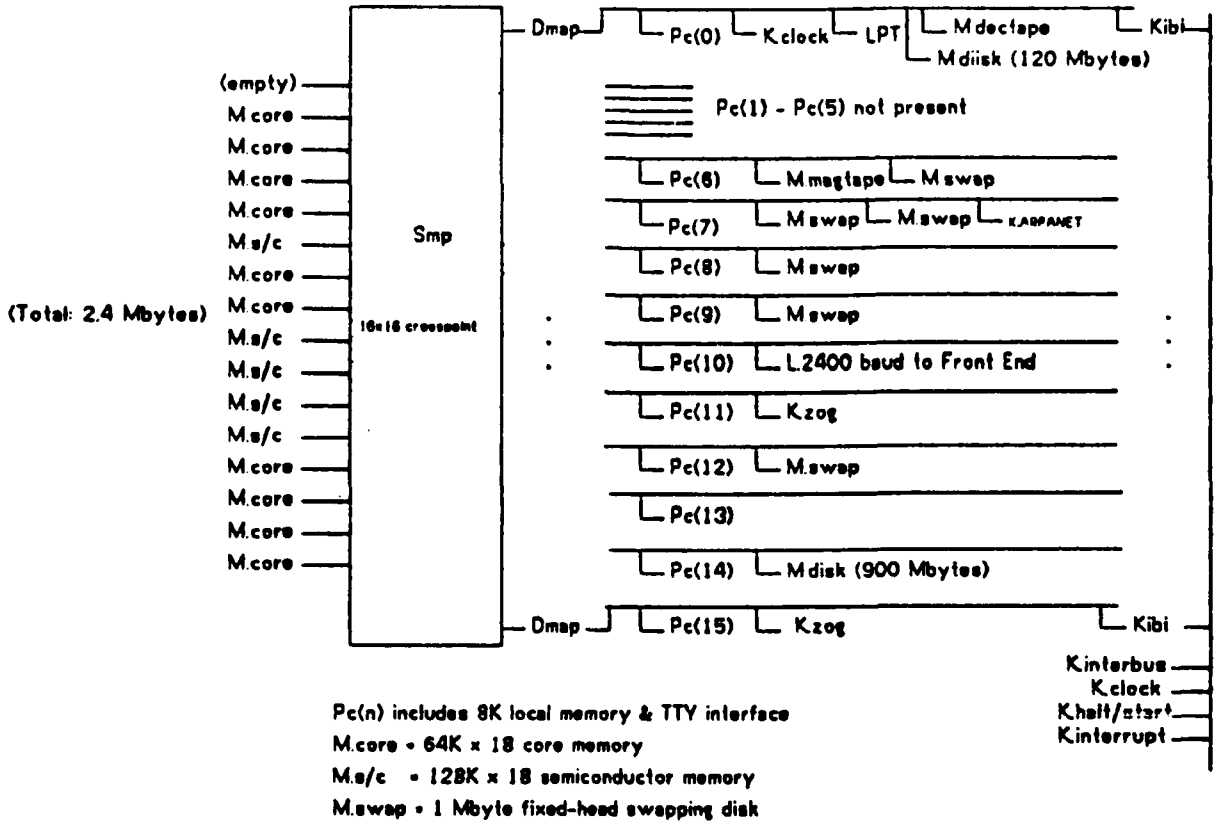


Figure 7: The Current Configuration of C.mmp

C.mmp's PDP-11/40E processors also incorporate a small ROM which facilitates the interprocessor START function described in section 2.5. When invoked, the IP-bus interface loads the address of ROM into the processor and commences execution. The program in the ROM loads the relocation registers and searches shared memory until it finds a page containing a particular key value in its first word. Execution is then passed to the program contained in that page. This allows us complete freedom in configuring C.mmp's shared memory, since no specific page frame needs to be present. Another program contained in the ROM provides a system bootstrapping mechanism to read in the system from a DECtape.

3.2. Memory

C.mmp currently has approximately 2.5 million bytes of primary memory distributed over the sixteen memory ports. Eleven ports use ferrite core memories while five ports use newer MOS memory.

3.3. Switch and IP Bus

Smp and the IP bus exist in their full 16-processor, 16-memory configuration.

3.4. Peripheral Devices

C.mmp has a normal complement of I/O equipment including disks, magnetic tape, and a line printer.

Pages are swapped in and out of memory using six fixed-head disks as a buffer between primary and secondary storage. The disks are on separate processors, allowing simultaneous transfers to each device. The backing-to-primary store ratio is about 3:1.

Permanent secondary storage consists of two moving-head disk systems providing a total of about 700 million bytes of storage. One controller has two 20-megabyte drives and two 40-megabyte drives, and the other controller has three 200-megabyte drives.

3.5. Links to Other Facilities

Hydra has not been developed in a vacuum, nor is C.mmp intended to be used primarily as a stand-alone computer system. It receives considerable support from other systems. Figure 8 shows how C.mmp is connected to the ARPANET and the other computer systems at the Computer Science Department. The connection to the ARPANET allows remote access of C.mmp from other points on the network and also provides a reasonably high speed (50K baud) link to all three of the DECsystem10's in the Department. Since much of the software support for C.mmp is developed on the PDP-10's, these links are an important facility.

3.6. Additional Hardware

A certain amount of additional hardware is present for special applications. As indicated in Figure 7, two advanced terminals for the ZOG man-machine-communication project [9] are on C.mmp. In the past C.mmp has also been the host of an audio-spectrum analyser for speech understanding research, a UNIBUS cycle counter, and the Hardware Monitor [10].

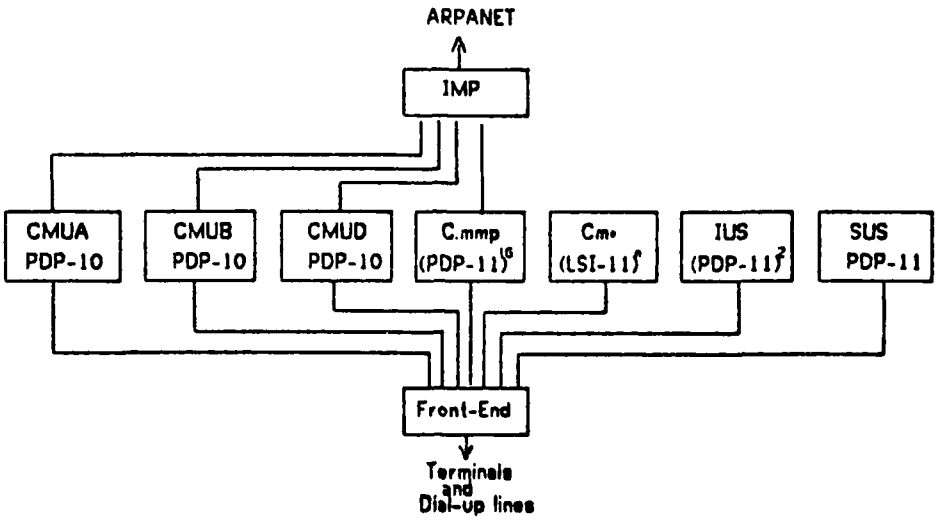


Figure 8: Interconnection of C.mmp to other computer systems

The Hardware Monitor is a special purpose device that resides on two UNIBUS's; one is the host that is being monitored, while the second is the controller, or supervisor. In addition to monitoring the UNIBUS of the host, the monitor has several high impedance probes which can be attached to any of a number of interesting signals either in the processor, an I/O device, or the switch.

4. Comments on the Implementation of C.mmp

Too often descriptions of new computer systems fail to point out those details in the construction of the systems that materially affect its final structure. We now look at some of the most important implementation features of C.mmp.

As a general comment we note that, with the exception of a few off-the-shelf components purchased later, C.mmp was built entirely with 1970-1972 technology.

4.1. The Processor-Memory Switch

Smp, the processor-memory switch discussed in section 2.1, is the largest component of C.mmp. Its construction was simplified by building it with only four basic modules: *switching*

module, *processor interface module*, *memory control module*, and *processor priority resolution module*. Each of these modules is simple enough to be implemented on a single printed circuit board.

The main processor-memory data paths in the switch are 72 bits wide and are implemented with the switch modules in a bit-slice fashion. Figure 9 shows a single bit-slice of the switch. 16-to-1 multiplexors (SN74150's) implement the 256 crosspoints. Sixteen of the multiplexors are used to implement the paths from the Pc's to the Mp ports and the other sixteen multiplexors are used to implement the return paths from Mp to the Pc's. Note the remarkable symmetry between the multiplexors forming the forward and return points. In fact, a switch module consists of sixteen multiplexors, and two modules are used to implement the bit slice shown in Figure 10. Control of the multiplexors comes from the processor priority resolution modules. The 144 switch modules needed to construct the data paths in the switch form the bulk of the logic in the crosspoint switch.

The processor interface module contains the steering logic to partially decode the address lines and route the he memory request to the designated memory module. This module also sets the selection lines for the switch from memory to the processor, thus determining which memory the processor will read. Finally, this module buffers data read from memory; this allows the switch to overlap the end of a read cycle with the start of the next cycle for another processor. The processor interface card is shown in Figure 11.

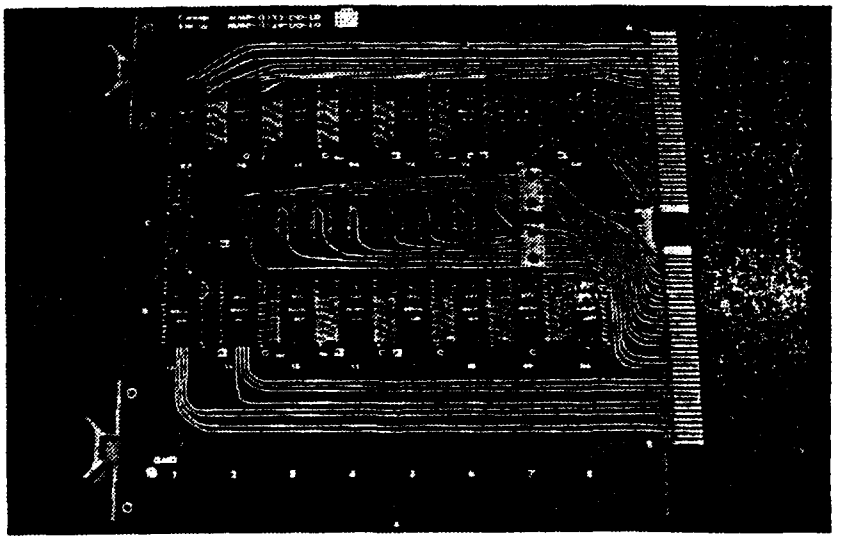


Figure 10: SW16 switch module

The memory control modules (Figure 11) are very straightforward. This module checks the

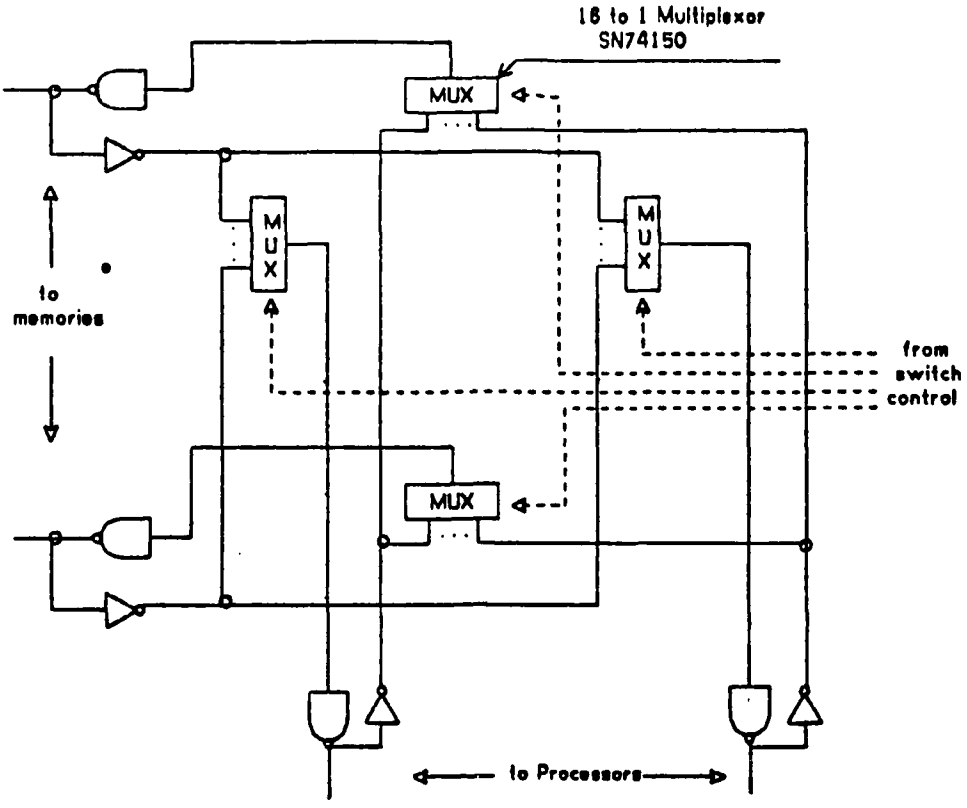


Figure 9: Bit-slice of crosspoint switch data paths

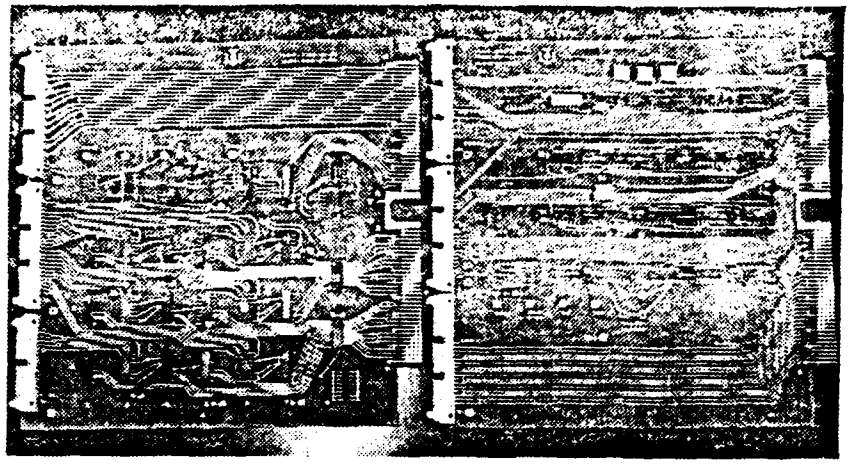


Figure 11: Processor interface (l.) and memory control (r.)

address parity which was generated in the relocation hardware. If an error occurs, it is reported back to the processor. This module also allows for easy configuration of memory by informing the processor if it should try to access a section of memory which is not present. Finally, this module communicates with the processor priority resolution module in order to generate the timing and control pulses for the actual memory modules.

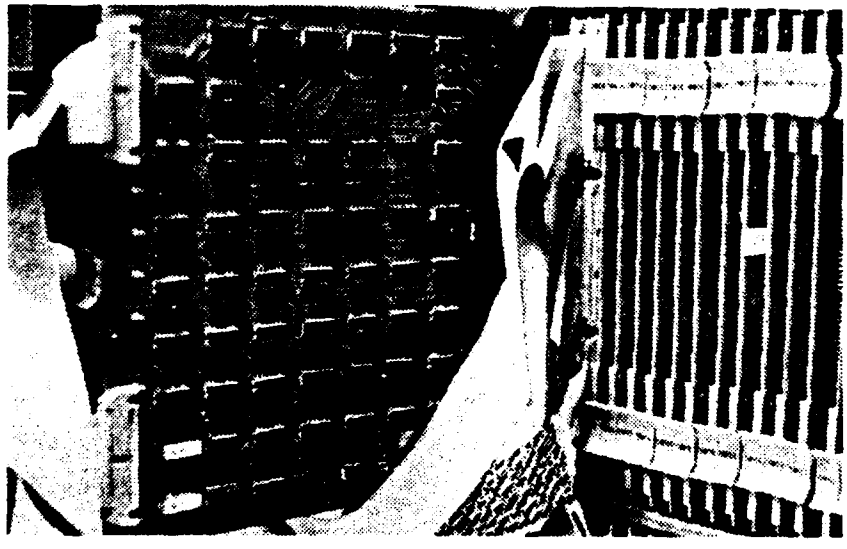


Figure 13: Priority resolution module in Smp

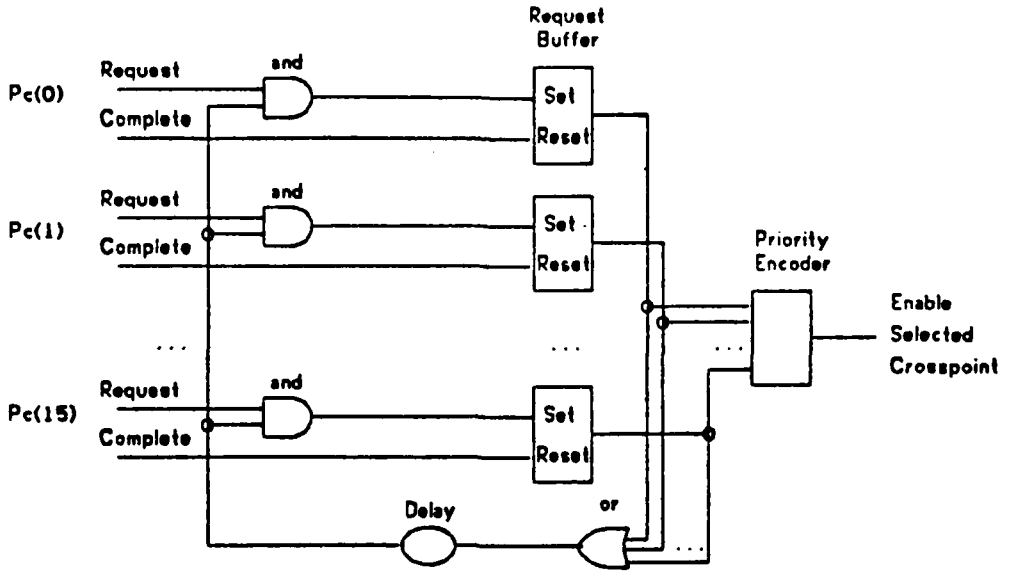


Figure 12: Simplified processor arbitration logic

The processor priority resolution module (Figure 13) is the most sophisticated component in the switch design. This module maintains a request buffer whose operation is illustrated in Figure 12. The function of the processor priority resolution module is to arbitrate between Pc's that are simultaneously requesting access to the same Mp port, and to queue those requests that must wait for other requests to complete. The arbitration logic shown in Figure 12 works in the following manner. When Pc(i) requests access to a particular Mp port (as indicated by the value of the four most significant address bits) it attempts to set bit i of the request buffer. However, the AND gate in front of the SET input to the buffer prevents Pc(i) from setting latch(i) until the request buffer contains all zeros. When the request buffer is empty all 16 AND gates feeding the SET inputs of the request buffer are enabled via the OR gate and DELAY shown at the bottom of the diagram. Now those Pc's with outstanding requests will set their corresponding latches in the request buffer. As long as a single Pc is making a request, and sets its corresponding latch, the column of AND gates will be disabled since the request buffer is no longer empty. Now the outputs of the 16 latches of the request buffer are fed into a priority encoder that indicates on four output lines the lowest numbered latch that is set. It is this priority encoder, therefore, that ultimately does the arbitration. After a Pc has been selected and it has read or written a word into Mp, the Pc

asserts its 'access complete' line that clears the Pc's latch in the request buffer. The priority encoder now selects the lowest numbered request of the remaining requests. Hence, Pc's are serviced in priority order from 0 to 15 and each Pc is guaranteed to wait no more than 15 memory cycles before gaining access to memory. The scheduling discipline induced by the priority resolution modules can be thought of as a quasi- round-robin discipline.[1]

4.2. Processor Modifications

The modifications to the processors can be considered to be in two classes: additions and actual alterations. On both the PDP-11/40 and the 11/20 only a very small percentage of the work is in altering existing logic. For instance, the detection and trapping of reserved instructions in the user space requires only the addition of two IC's and the replacement of two others on the instruction decode module of the processor.

The addition of the other features requires that about 30 processor-generated signals be acquired from the backplane of the processor. Additions to each processor are all contained on one new PDP-11 system board (Figure 14). (A standard PDP-11/40 is implemented on five such boards.)

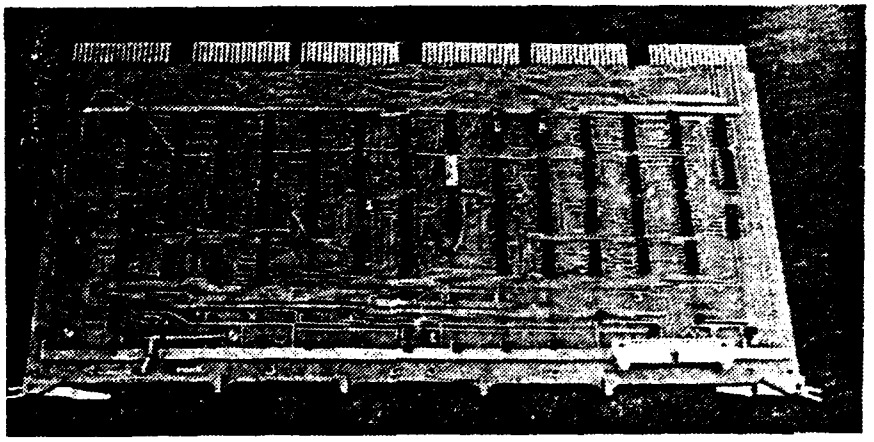


Figure 14: PDP-11/40 modification board

[1]This discussion of the priority resolution module is a simplification of the actual operation. In reality, there is also a high priority input to each latch in the request buffer that circumvents the column of AND gates. This high priority feature is intended for very high performance I/O devices which may not be able to tolerate a high level of memory interference.

4.3. Relocation and Error Detection

The two boards which implement the memory relocation logic (Figure 15) are also the site of much of the error-detection circuitry. How the errors are reflected to the user is an interesting implementation detail.

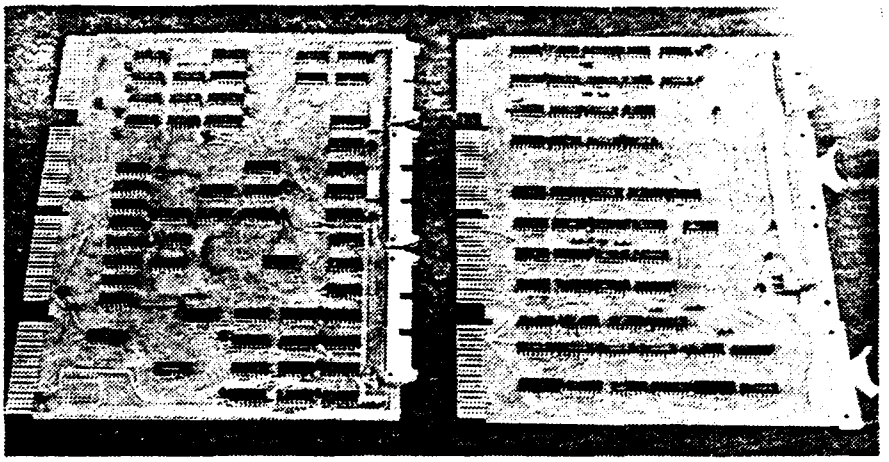


Figure 15: Relocation modules for PDP-11/40

Upon detection of a switch-related error (parity errors, writing a read-only page, etc.) the logic causes the processor to take a normal NXM (non-existent memory) trap by blocking the acknowledgment signal (SSYN in PDP-11 terminology) from memory. That a *trap* is taken is important. because traps can take effect before the completion of an instruction. Status bits set in a control register allow the software to determine the actual cause of the error and can cause later errors to be ignored until the processor's state is recorded.

Other exceptional conditions, including stack underflow, violation of the SP conventions, and attempting to execute an illegal instruction, cause normal interrupts.

Even with our error-detection extensions, the PDP-11's complicated addressing modes (which often have side effects) makes it almost impossible to 'back up' and retry an instruction which failed. This is unfortunate, since the majority of detected hardware errors appear to be transient.

4.4. Interprocessor Bus

The interprocessor bus controller (Figure 16) performs three functions. It implements and

broadcasts the time-of-day clock value discussed above. It generates and broadcasts the timing pulses which are used by the interval timer in the interfaces. Finally it generates and broadcasts the timing and control signals necessary to time-multiplex the various interprocessor functions on the IP bus. By using a time-sliced function bus we have reduced a potential 1500+ wire requirement to 16 cables of 20 wires each; however, we give up knowing which processor invoked a function. Neither of these has a major impact on the Hydra design.

Figure 17 shows the master clock module, which is equipped with a switch panel to allow manual alteration of the time base.

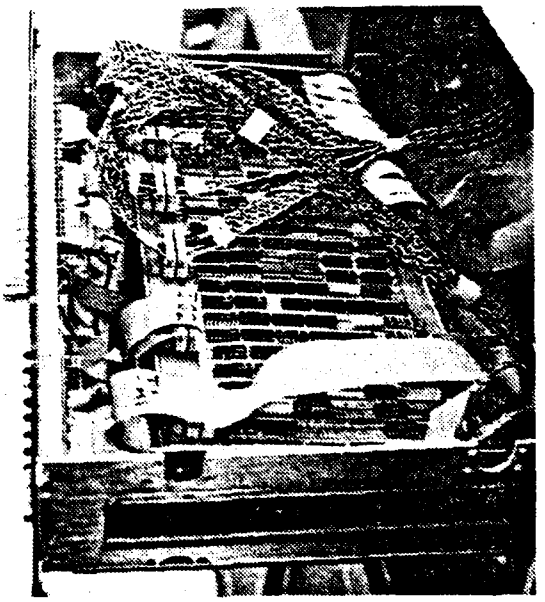


Figure 16: Interprocessor bus interface

4.5. Peripherals

The paging disks mentioned in section 3.4 are perhaps worthy of special note. C.mmp's page size is exactly equal to the capacity of one track on the disk. By modifying the controller slightly, this coincidence can be exploited in such a way that there is no latency on disk transfers which are one page long. Rotational latency is avoided by having the controller start the transfer at the beginning of the next physical block (16 words) and transferring 8K bytes without track switching. This zero-latency scheme provides better

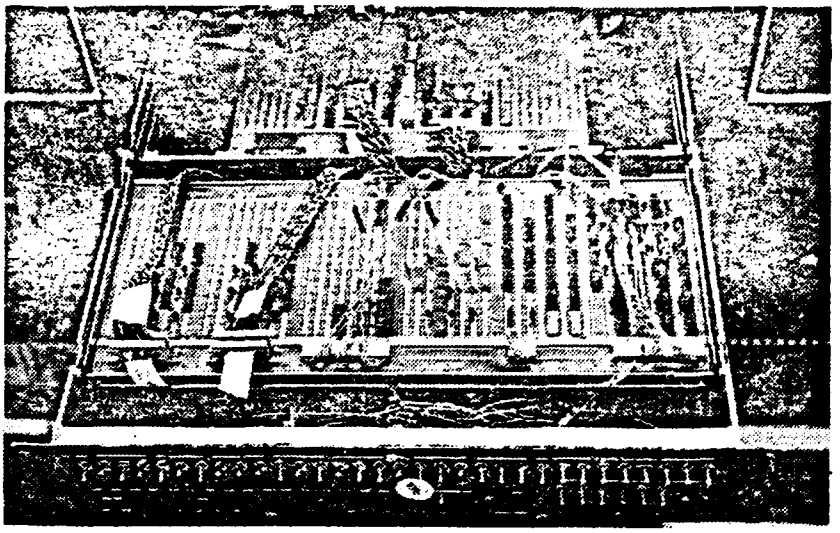


Figure 17: C.mmp master clock

service than SLTF or any of the other scheduling disciplines that have been developed to optimize the performance of paging disks with latency.

5. Technology and Costs

C.mmp is a mixture of off-the-shelf and custom-built hardware. Figure 18 gives an approximate breakdown of the equipment in terms of complexity and cost.

The portions of C.mmp built at CMU use a mixture of TTL and Shottky TTL technology. ECL was not used because at the time of C.mmp's construction ECL did not offer the range of MSI components available in TTL. Likewise the large amount of ferrite core memory on C.mmp is due to the state of MOS memory technology *ca* 1972.

The cost figures given in Figure 18 are only estimates. The cost for the PDP-11/40 and for memory was the purchase price of the equipment when we bought it. The other hardware was built at CMU, and the figures given are our rough estimates of the replication cost, excluding design and setup costs, and excluding any profit margin.

6. Performance

We have now had several years to evaluate the performance of C.mmp. To date, Oleinick's work [1] is the most comprehensive study of a single multiprocess application on C.mmp. We

Part	No. Boards	No. IC's	Unit Cost	No. in System
11/40	5	332	$12,000	11 (16 max.)
ustore	2	200	$1,300	1/Pc
Pc mod's	1	57	$600	1/Pc
Dmap	3	120	$1,500	1/Pc
Smp			$50,000 (16 x 16 configuration)	
SW16	1	24		70
PI	1	26		1/Pc
MC	1	20		1/Mp
PRI	1	54		1/Mp
IP-bus				
Ctrl	2	200	$3,000	1
I'face	1	200	$3,000	1/Pc
Mp (core)	Unit=8Kx18		$1,300	80
Mp (MOS)	Unit=128Kx18		$12,000	5

Approximate total replication cost (excluding peripherals): $600,000

Figure 18: C.mmp Technology and Costs

believe that the existing studies confirm our early beliefs that C.mmp is a powerful and cost-effective computing resource.

An in-depth performance study is not appropriate here, but we have tried to gather together in Figure 19 some low-level measurements of the C.mmp hardware. The measurements of Pc and Mp speed are from [1] and are averages over all Pc's and Mp's of the same type.

Note that the timings for Pc and Mp indicate that contention will degrade performance whenever more than two processors are trying to access the same memory port.

PDP-11/40 execution speed	0.68×10^6 memory references/second[1]
Mp (core)	1.49×10^6 memory references/second[2]
Mp (MOS)	1.71×10^6 memory references/second[3]
200-Mbyte disk	2.5 usec/word transfer rate 28 ms average seek 8 ms average latency
20,40-Mbyte disk	7.5 usec/word transfer rate 29 ms average seek 12.5 ms average latency
Paging disks	4.1 usec/word transfer rate 17 ms page read time 34 ms page write time (w/verify)

Figure 19: C.mmp hardware performance

7. Acknowledgements

The development of C.mmp and Hydra represents the combined effort of many people. The original design group included Bill Wulf, Sam Fuller, Ellis Cohen, Bill Corwin, David Jefferson, Roy Levin, Chuck Pierson, and Fred Pollack. The project later included Tom Lane, Rick Gumpertz, Sam Harbison, Guy Almes, Joe Newcomer, Mary Thompson, Hank Mashburn, Dave Babcock, Navindra Jain, and George Robertson. The hardware was designed and built by Bill Broadley and the CMU Computer Science Engineering Laboratory.

We would like to thank Bill Wulf and Joe Newcomer for their criticisms of this paper, and Roy Levin, the original editor. We would also like to thank Hank Mashburn, the C.mmp/Hydra

[1] Measured speed of a single 11/40 executing out of a single memory port with no other processors contending. On an 11/40, one instruction requires about 2.5 memory references on the average, so .68 million ref/sec translates to about .27 MIPS (million instructions per second) for each processor, or about 3 MIPS for the 11-Pc configuration and 4.3 MIPS for the full 16-Pc configuration without caches.

[2] Measured rate of memory references to a memory port when all processors are contending on that port.

[3] See note for Mp (core).

Project Manager, and David Babcock, C.mmp's chief engineer, for their help in preparing the paper and providing the tender loving care which does wonders for system MTBF.

The photographs are by the author (Harbison).

Authors' address: Samuel H. Fuller, Digital Equipment Corporation, Tewksbury, MA 01876; Samuel P. Harbison, Computer Science Deparment, Carnegie-Mellon University, Pittsburgh, PA 15213.

8. Further Reading

[2] is a retrospective look at the successes and failures of the C.mmp/Hydra project to date. The best introduction to Hydra is found in the original Hydra paper [11] and in the three papers presented at the *Fifth Symposium on Operating System Principles* [12]. Hardware and software performance are analyzed in [1]. Potential users should consult [13, 14, 15]. Additional references can be found in the complete bibliography in [2].

9. References

[1] Oleinick, P., "Implementation and Evaluation of a Parallel Algorithm on C.mmp," Technical Report, Computer Science Department, Carnegie-Mellon University, 1978.

[2] Wulf, W. A., and Harbison, S. P., "Reflections in a Pool of Processors," Technical Report, Computer Science Department, Carnegie-Mellon University, June 1978.

[3] Wulf, W. A. and Bell, C. G., "C.mmp -- A Multi-mini-processor," *Proceedings of the Fall Joint Computer Conference*, 1972, pp. 765-777.

[4] Bell, C. G. and Newell, A., *Computer Structures: Readings and Examples*, McGraw-Hill, New York, 1971.

[5] Digital Equipment Corporation, *PDP-11/05/10/35/40 Processor Handbook*, Maynard, Massachusetts, 1973.

[6] Heart, F. et. al., "A New Minicomputer/Multiprocessor for the ARPA Network," *Proceedings AFIPS National Computer Conference*, 42, 1973.

[7] Swan, R. J., Fuller, S. H., and Siewiorek, D. P., "CM*: A Modular, Multi-processor," *Proceedings of the National Computer Conference*, Dallas, TX, June 1977.

[8] Digital Equipment Corporation, *LSI-11 Microcomputer*, Maynard, Massachusetts, 1975.

[9] Robertson, G., Newell, A., and Ramakrishna, K., "ZOG: A Man-Machine Communication Philosophy," Technical Report, Department of Computer Science, Carnegie-Mellon University, August 1977.

[10] Fuller, S., Swan, R., and Wulf, W., "The Instrumentation of C.mmp: a Mult-mini-processor," IEEE Compcon, 1973.

[11] Wulf, W. A. et. al., "Hydra: The Kernel of a Multiprocessor Operating System," *CACM 17,6* (June, 1974) pp. 337-345.

[12] Wulf, W. A. et. al., "Overview of the Hydra Operating System," *and* Levin, R. et. al., "Policy/Mechanism Separation in Hydra," *and* Cohen, E. et. al., "Protection in the Hydra Operating System," *Proceedings of the Fifth Symposium on Operating System Principles*, Austin, Texas, November 1975, pp. 122-160.

[13] Newcomer, J. et. al., "Hydra: Basic Kernel Reference Manual," Technical Report, Department of Computer Science, Carnegie-Mellon University, 1976.

[14] Reiner, A., and Newcomer, J., ed., "The Hydra User's Manual," Technical Report, Department of Computer Science, Carnegie-Mellon University, August 1977.

[15] Gumpertz, R. H., Karlton, P. L., and Lamb, D. A., "The Hydra Users' Library," Technical Report, Department of Computer Science, Carnegie-Mellon University, August 1978.