# Cm*

# *Cm*: a Modular, Multi-Microprocessor*

R. J. Swan, S. H. Fuller, and D. P. Siewiorek
Carnegie-Mellon University
Pittsburgh, Pa. 15213

~~November 30, 1976~~ File NCC 1977

Submitted to:
The 1977 National Computer Conference

CR Categories: 6.20, 4.30, 4.32
Keywords: Multiprocessor, microprocessor, computer architecture, virtual memory

*Abstract*

This paper describes the architecture of a new large multiprocessor computer system being built at Carnegie-Mellon University. The system allows close cooperation between large numbers of inexpensive processors. All processors share access to a single virtual memory address space. There are no arbitrary limits on the number of processors, amount of memory or communication bandwidth in the system. Considerable support is provided for low level operating system primitives and inter-process communication.

This work was supported in part by the Advanced Research Projects Agency under contract F44620-73-C-0074, which is monitored by the Air Force Office of Scientific Research, and in part by the National Science Foundation Grant GJ 32758X. The LSI-11's and related equipment were supplied by Digital Equipment Corporation.

## 1. *Introduction*

Cm* is an experimental computer system designed to investigate the problems and potentials of modular, multi-microprocessors. The initial impetus for the Cm* project was provided by the continuing advances in semiconductor technology as exemplified by processors-on-a-chip and large memory arrays. In the near future processors of moderate capability, such as a PDP-11, and several thousand words of memory will be placed on a single integrated circuit chip. If large computer systems are to be built from such chips, what should be the structure of such a 'computer module'?

Initial versions of the Cm* architecture [Fuller, et al. 73] grew in part as an extension to the modular design of systems from register transfer modules, or RTMs [Bell et al, 72]. In addition there was substantial interest in the development of large multiprocessor systems such as Pluribus [Heart, et al. 73] and C.mmp[Wulf and Bell, 72]. Cm* is intended to be a testbed for exploring a number of research questions concerning multiprocessor systems, for example potential for deadlocks, structure of inter-processor control mechanisms, modularity, reliability, and techniques for decomposing algorithms into parallel cooperating processes.

The structure of Cm* is very briefly described in Section 2. Section 3 is a description of the address structure and discusses the support given for the operating system. The use of the addressing structure for inter-process communication and control operations is discussed in Section 4. A companion paper [Swan, et al. 77] discusses the various mechanisms used to implement the complex address mapping and routing structure of Cm*. Some results from the

performance modelling of Cm* are also presented. A second companion paper [Jones, et al. 77] describes the structure of the basic operating system and support software.

## 2. *The Structure of Cm**

There is a surprising diversity of ways to approach the interconnection of processors into a computing system [Anderson and Jensen, 76]. The processors could be interconnected with several serial I/O links to form a computer network; they could be interconnected in a tight synchronous fashion to build an array processor; or the processors could be organized to share primary memory. This last approach, a multiprocessor organization, was chosen for Cm* because it offers a closer degree of coupling, or communication, between the processors than would a multicomputer or network configuration. Multiprocessors also have a more general range of applicability than other multiple processor systems.

During the development of the Cm* structure a wide variety of multiprocessor switch structures were considered [Swan, et al. 76B]. The basic structure selected is represented in Figure 2.1. The essential feature which distinguishes it from other multiprocessor structures is that shared memory is not separated from the processing elements, but rather a unit of memory and a processor are closely coupled in each module and a network of buses gives a processor access to nonlocal memory. This structure allows modular expansion of the number of processors and memory modules without a rapid increase in the interconnection costs. Memory can be shared even though there is no direct physical connection between

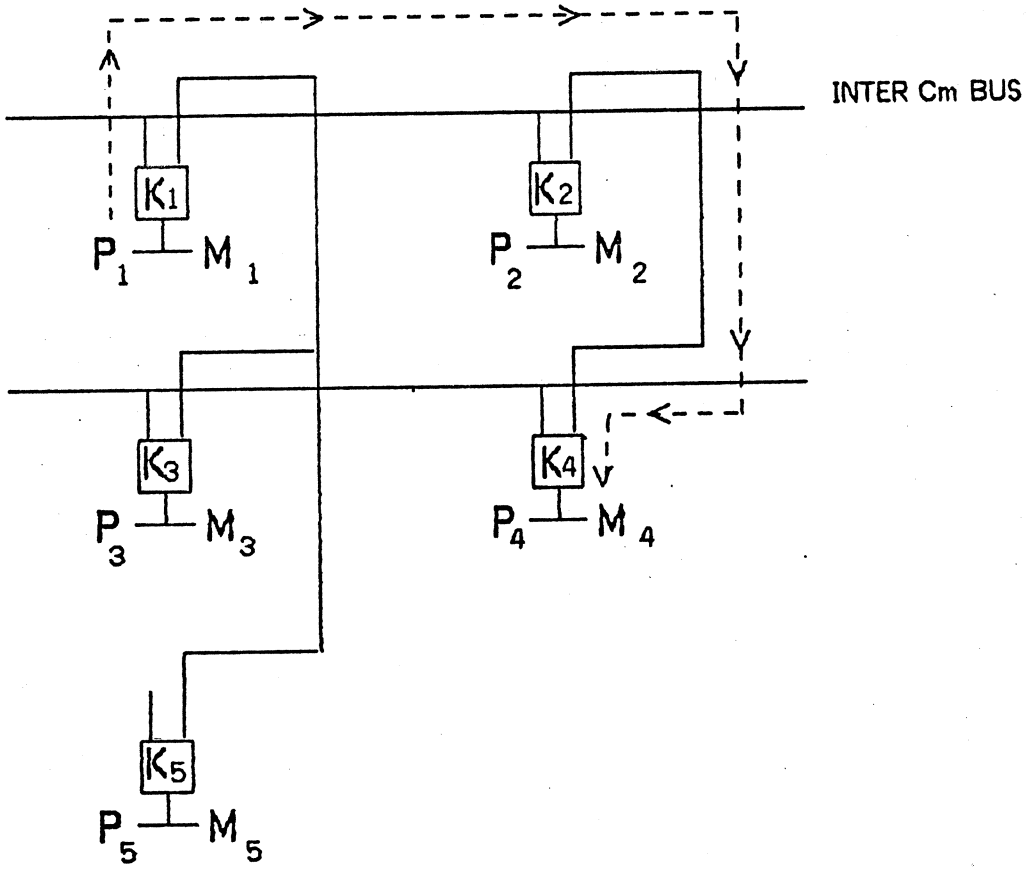


Figure 2.1 Canonical Computer Module Structure

the requesting processor and the required memory. For example, consider a request by processor, P1, to the memory, M4, in Figure 2.1. The address mapping element, K1, directs the reference from P1 onto the intermodule bus. The address is recognized by K2, which directs it onto a second inter-module bus. The reference is finally accepted by K4, which accesses the request memory location and passes back an acknowledgement or data to the requesting processor. The need for high inter-module communication rates will be minimized if a large fraction of each processor's references to primary memory 'hit' the section of memory local to the processor. (Preliminary experiments in the Fall of 1976 indicate that hit ratios of better than 90% can be expected provided that the code executed is normally held local to the processor.)

## 2.1 Deadlock with References to Nonlocal Memory

Almost all computer systems implement accesses from processor to primary memory with *Circuit Switching*, that is, a complete path is established from a processor to the memory being referenced. Circuit switching is not feasible for a structure like Cm* where local memory is also accessible as shared memory. Figure 2.1 shows the path used for P1 to access M4 via K2. Consider a concurrent attempt by P4 to access M1 via K2. With a circuit switch implementation, a situation could arise where P1 held its local memory bus and the bus connecting K2, while P4 also holds its own memory bus plus the bus connecting K4 to K2. Neither memory reference could complete without one processor first releasing the buses it

holds. There are numerous situations where deadlock over bus allocation can occur. Resolving this deadlock requires, at the very least, a timeout and retry mechanism.

The alternative to circuit switching is *Packet Switching*. In a packet switched implementation, the address from the processor is latched at each level in the bus structure. Buses are not allocated for the full duration of a memory reference, but just for the time taken to pass a 'packet', containing an address and/or data, from one node on the bus to another. Therefore packet switching allows significantly better bus utilization and significantly reduced bus contention in Cm*-like structures. The use of packet switching eliminates the possibility of deadlock over bus allocation but introduces the possibility of deadlock over buffer allocation. [Fuller, et al. 73; Swan, et al. 76A] Buffers, or intermediate registers, are resources which can be provided very cheaply, relative to providing additional inter-Cm buses, with present technology.

## 2.2 *The Actual Structure of Cm**

Design studies indicated that very little performance loss would result from combining several individual Computer Modules into a cluster and providing a shared address mapping and routing processor, Kmap, which allowed communication with other clusters. Because the cost of the Kmap is distributed across many processors it can be endowed with considerable flexibility and power at relatively little incremental cost. Because of its commanding position in the cluster, the Kmap can ensure mutual exclusion on access to shared data structures with

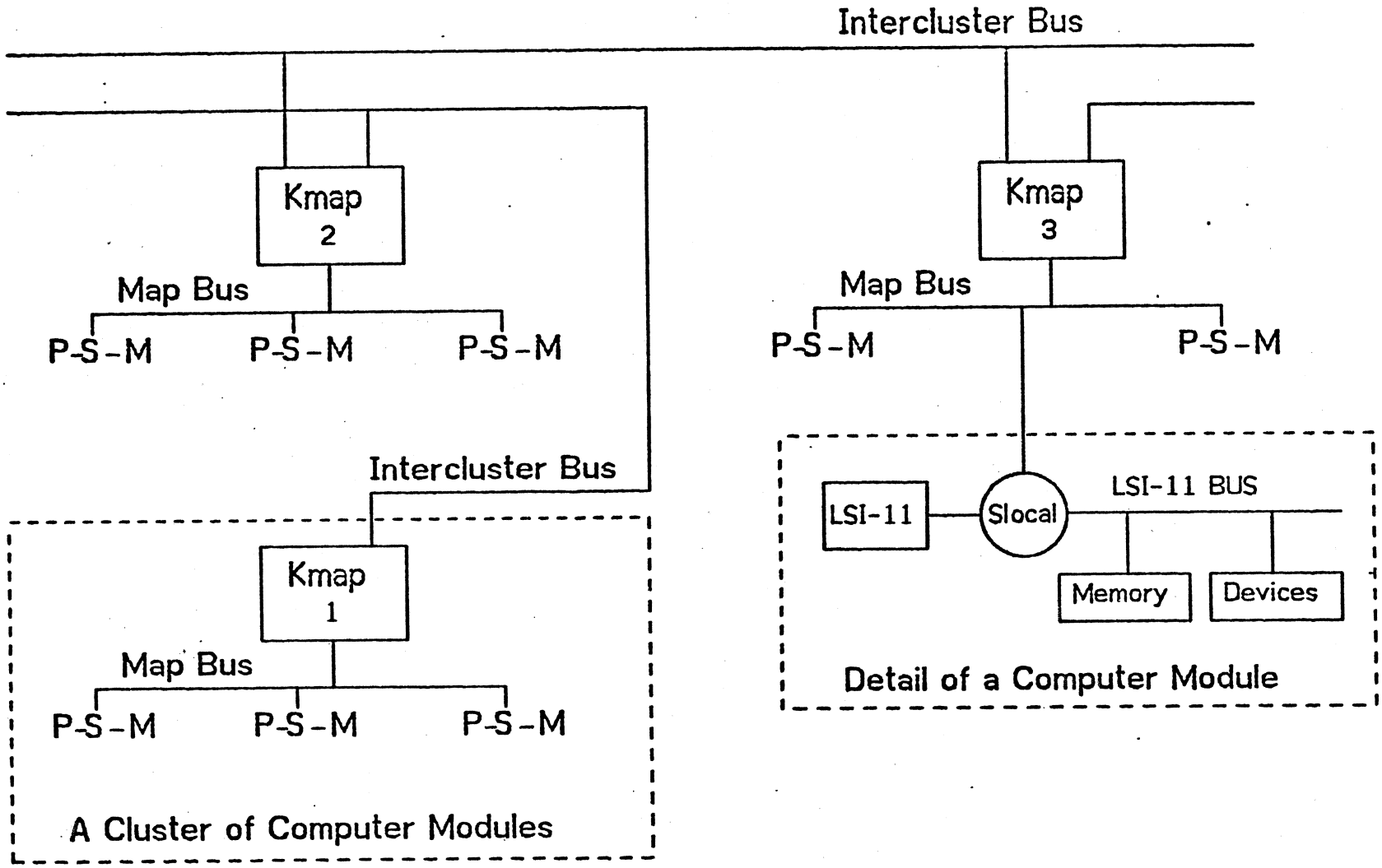


Figure 2.2 A Simple 3 Cluster Cm* System

very little overhead.

The full structure of Cm* is shown in Figure 2.2. Individual Computer Modules, or Cms, consist of a DEC LSI-11 processor, an Slocal and standard LSI-11 bus memory and devices. The processor is program compatible with PDP-11s; thus a large body of software is immediately available. The prime function of the Slocal, or local switch, is to direct references from the processor selectively either to local memory or to the Map Bus, and to accept references from the Map Bus to the local memory.

Up to 14 Computer Modules and one Kmap form a cluster. The Kmap, or mapping processor, consists of three major components. The Kbus arbitrates and controls the Map bus. The Pmap is a horizontally microcoded 150 ns cycle time processor. The basic configuration has 1 K x 80 bits of writable control store and 5K x 16 bits of bipolar RAM for holding mapping tables etc. The third level of the Cm* structure is provided by the intercluster buses which allow communication between clusters. The Linc provides the interface to two intercluster buses.

There are no arbitrary limits to the size of a Cm* system. Memories, processors and Kmaps can be incrementally added to suit needs. Any processor can access any memory location in the system. The routing of a processor's reference to a target memory is transparent to the program, thus the system can be reconfigured dynamically in response to hardware failures.

## 3. *Architecture of the Cm* Address Translation Mechanisms*

Many of the more conventional aspects of the architecture of the Cm* system are consequences of using LSI-11's for the central processing elements. The organization and encoding of the instructions, interrupt and trap sequencing, and the 64K byte processor address space of a Cm* system are all a result of the PDP-11 architecture as implemented on the LSI-11. By selection of the LSI-11, however, we do not want to imply that the PDP-11 architecture is ideally suited to multiprocessor systems. The ideal solution would have been for us to have designed our own processors. However, practical considerations of time, money, and existing support software lead us in early 1975 to recognize that by chosing the LSI-11 we could concentrate on those aspects of the Cm* architecture unique to multiprocessor systems. This section, and the following section on control structures, will discuss the Cm* architecture as we extended it beyond the standard PDP-11 architecture.

The addressing structure is one of the the most important aspects of any computer architecture, it is even more significant when cooperation between multiple processors is to be achieved by sharing an address space. Denning [1970] lists four objectives for a memory mapping scheme:

(a) Program modularity: the ability to independently change and recompile program modules.
(b) Variable size data structures.
(c) Protection
(d) Data and program sharing: allowing independent programs to access the same physical memory addresses with different program names.

For Cm*, where we are using processors with only a 64K byte address space, we must add the following requirement:

(e) Expansion of a processor's address space.

Cm* has a $2^{32}$ byte segmented virtual address space. Segments are of variable size up to a maximum of 4K bytes. There is a capability-based protection scheme enforced by the Kmap. The addressing structure provides considerable support for operating system primitives such as context switching and interprocess message transmission.

## 3.1 The Path from Processor to Memory

The Slocal (see Figures 2.2 and 3.1) provides the first level of memory mapping. A reference to local memory is simply relocated, on 4K byte page boundaries, by the relocation table in the Slocal. As discussed above, it is assumed that most memory references will be made by processors to their local memory. Relocation of local memory references can be implemented with no performance overhead because the synchronous processor has sufficiently wide timing margins at the points where address relocation is performed. For segments which are not in a processor's local memory the relocation table has a status bit which causes the address to be latched, the processor forced off the LSI-11 bus, and a Service Request to be signalled to the Kmap. All transactions on the Map bus are controlled by the Map bus controller, or Kbus, which is a component of the Kmap. The address generated by the processor is transferred via the Map bus to the Pmap, the microprogrammed processor within the Kmap. If the reference is for memory within the cluster then the Pmap generates a physical address and sends it to the appropriate Slocal. If

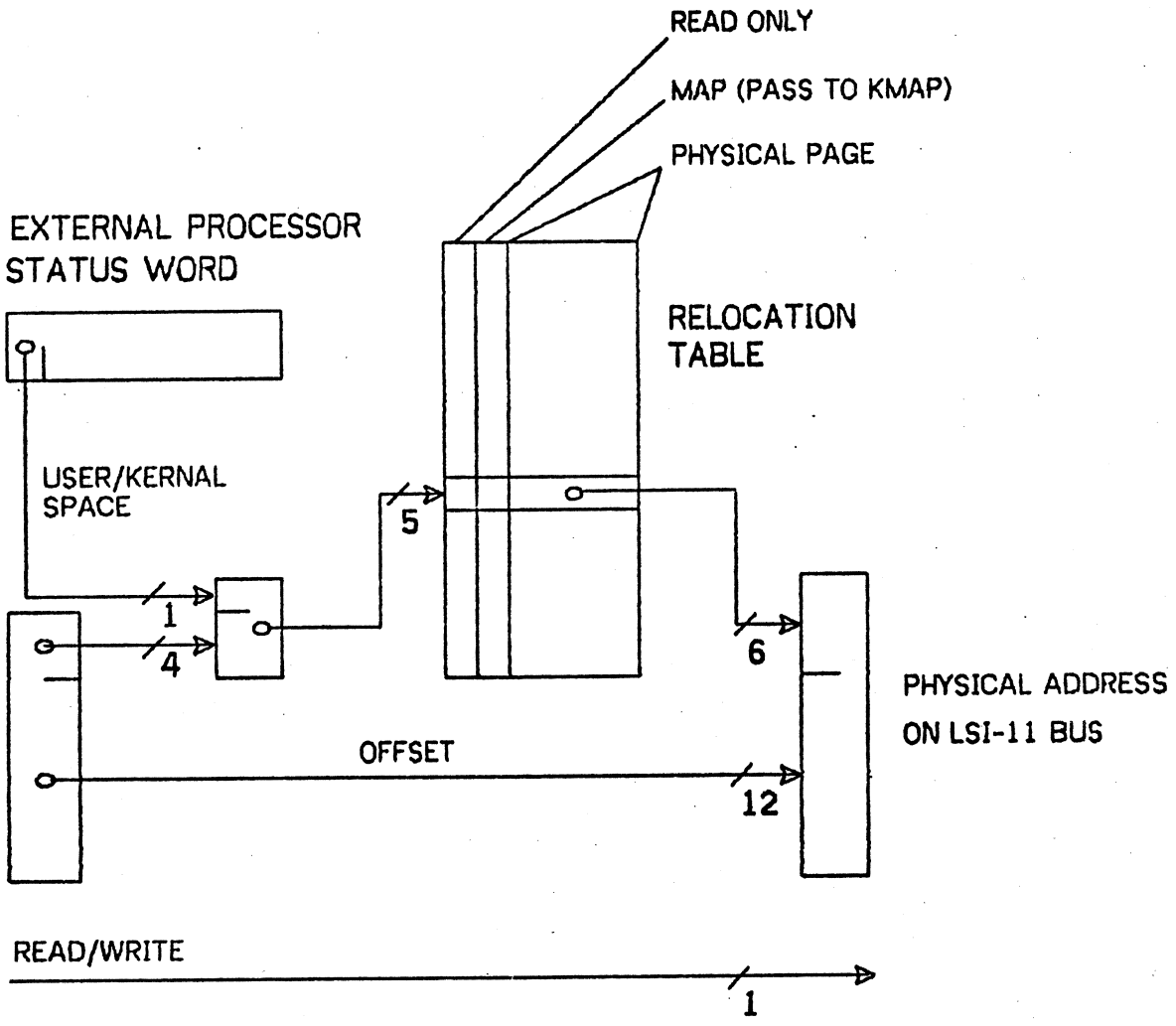


Figure 3.1. Addressing Mechanism for a Local Memory Reference

it is a write operation, data is passed directly from the source Slocal to the destination Slocal; the data does not have to be routed through the Kmap. The selected destination Slocal performs the requested memory reference and the processor in the destination Computer Module is not involved. When the reference is complete the Kbus transfers the data read from the destination Slocal directly back to the requesting processor via the Map bus and its Slocal.

If the processor references a segment in another cluster then the Pmap will transmit a request to the desired cluster via the Linc and the Intercluster buses. (See Fig. 2.2.) If the destination cluster is not directly connected to the source cluster, that is, if it does not share a common intercluster bus, then the message will be automatically routed via intermediate clusters. When the message reaches the destination cluster, the memory reference is performed similar to a request from a processor within the cluster. An acknowledgement, or Return, message (containing data in the case of a read) is always sent back to the source cluster and subsequently to the requesting processor.

## 3.2 The Addressing Environment of a Process

The virtual address space of Cm* is subdivided into up to $2^{16}$ *Segments.* Each segment is defined by a *Segment Descriptor.* The standard type of segment is similar to segments in other computer systems; it is simply a vector of memory locations. The segment descriptor specifies the physical base address of the segment and the length of the segment. Segments

are variable in size from 2 bytes to 4 K bytes. However, other segment types may be more than simple linear vectors of memory; references to segments may invoke special operations. Segments may have the properties of stacks, queues or other data structures. Some segments may not have any memory associated with them, and a reference to the segment would invoke a control operation. For each segment type, up to eight distinct operations can be defined. For normal segments the operations are Read and Write. Conceptually, segments are never addressed directly; they are always referenced indirectly via a *Capability*. A capability is a two-word item containing the name of a segment and a *Rights* field. Each bit in the rights field indicates whether the corresponding operation is permitted on the segment.

To provide efficient support for context swapping, message-sending etc., it is necessary for the Kmap microcode to understand some of the structure of an executable software module (variously called a process, activity, address space etc. ). Each executable software module is represented by an *Environment*, Figure 3.2. An environment is a three-level structure composed of segments. The first level in the structure is a *Primary Capability List*, CL[0]. The first entry in CL[0] is a Capability for a *State Vector*, which holds the process state while it is not executing on a processor. Entries CL[0](1) to CL[0](7) in the Primary Capability list may contain Capabilities for *Secondary Capability Lists* referred to as CL[1] through CL[7] respectively. The remaining entries in the Primary Capability List and all the entries in the Secondary Capability Lists contain Capabilities for segments which can be made directly addressable by the process when it executes. These may be code, data or

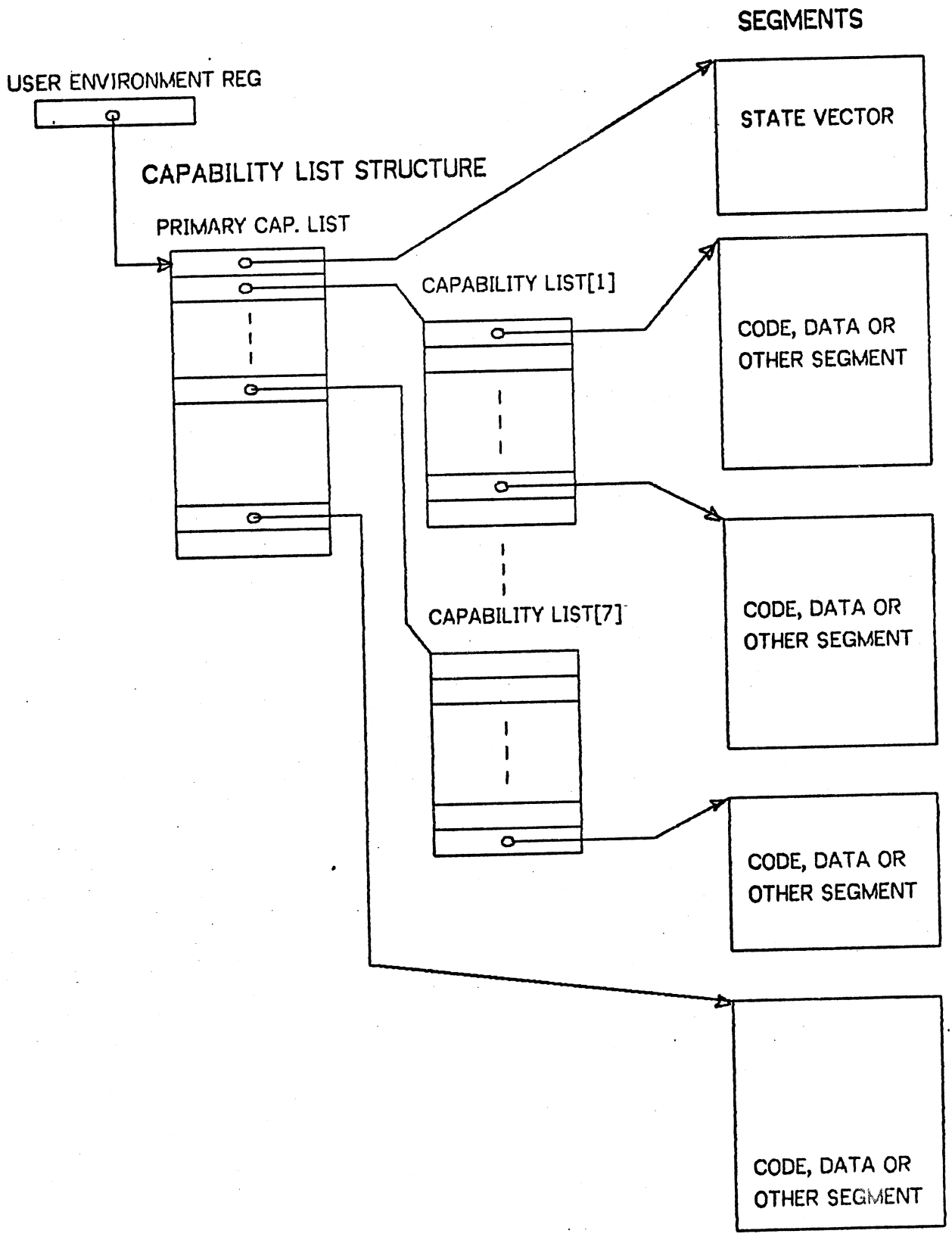


Figure 3.2 The Environment of a User Software Module

any other type of segment. The provision of up to eight Capability Lists facilitates the sharing of segments and sets of segments by cooperating processes. A software module can only access those segments for which it has capabilities and perform only those operations permitted by the capabilities.

### *3.3 Virtual Address Generation*

The processors in Cm*, LSI-11s, can directly generate only a 16 bit address. This 64 K byte address space is divided into 16 pages of 4 K bytes each. Each page provides a window into the system wide $2^{28}$ byte virtual address space, (see Figure 3.3) and can be independently bound to a different segment in the virtual address space. The top page in the processor's address space, page 15, is reserved for direct program interaction with the Kmap. This mechanism is analogous to the I/O page convention in standard PDP-11s. In page 15 there are 15 pseudo registers, called *Window Registers*. These define the binding between page frames in the processor's immediate address space and segments in the virtual address space. This binding is done indirectly via capabilities. Each window register holds an index for a capability in the currently executing software module's capability list structure. A Capability List index consists of a three bit field to select one of the up to eight Capability Lists, plus an offset within the C-List.

To overlay the processor's address space, ie. to change the mapping from page frame to segment in the virtual address space, a program simply writes a new capability index into the

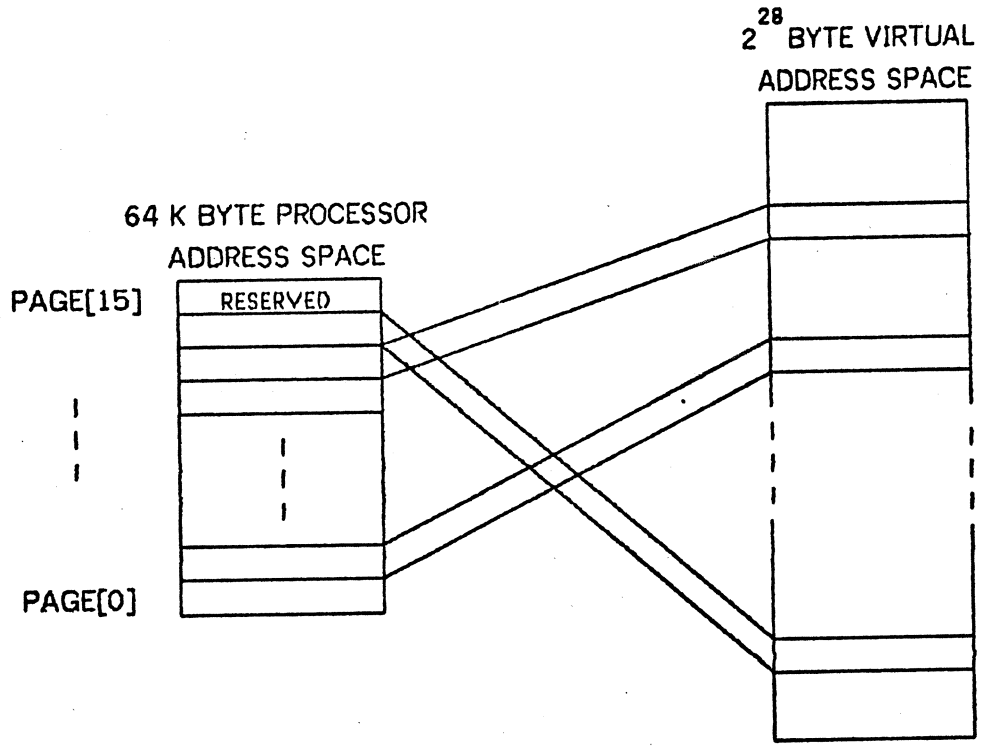


Figure 3.3 Windows from the Processor*s Immediate Address Space to the Virtual Address Space

appropriate window register. This overlay operation is completely protected; the program can only reference segments for which it has a Capability. The act of writing the Capability index into the window register activates the Kmap. The Kmap retrieves the selected Capability from main memory and places it in its "Capability cache". The Kmap adjusts its internal tables so that subsequent references to the page frame will map to the segment specified by the Capability. If the segment is local to the processor then the Kmap may also change the relocation register in the Slocal so that references to the segment can be performed at full speed without the intervention of the Kmap. The Slocal, for cost and performance reasons, does not have the hardware necessary for bounds checking on variable sized segments. Thus only fixed size 4 K byte segments can be accessed without Kmap assistance.

The Cm* mechanism for address space overlaying should be contrasted with mechanisms in other computer systems. When executing a large program on a processor with a small immediate address space, the time taken to overlay the address space can have a crucial effect on performance. Measurements made of the execution of the operating system HYDRA [Wulf et al, 76] on the C.mmp multiprocessor showed that relocation registers were being changed approximately every 12 instructions. (This does not, however, imply that user programs perform overlay operations this frequently.) Within the operating system this overlay operation is a single PDP-11 MOVE instruction because no protection is involved. However for user programs running under HYDRA, an overlay operation requires invocation of the operating system with several hundred instructions of software overhead. Subsequent optimization, and partial microcoding, have greatly reduced this overhead.

Figure 3.4 shows the conceptual translation from a 16 bit processor-generated address to a virtual address. The four high order address bits from the processor select one of 15 Window registers. The Window register holds an index for a Capability in the executing software modules Capability List structure. The 16 bit segment name from the selected Capability is concatenated with the 12 low order bits from the processor to form a 28 bit virtual address. Figure 3.4 also shows the read/write indicator from the processor being concatenated with two bits in the address expansion registers to form a three bit opcode. The corresponding bit in the Capability rights field is selected and tested. If the operation is not permitted then an error trap is forced.

### *3.4 Virtual to Physical Address Mapping*

The mapping from virtual to physical address depends on the location of the segment in the network and, of course, on the type of the segment. We begin with the case of a simple read/write segment residing within the same cluster as the processor referencing the segment. This mapping is shown in Figure 3.5. The segment name is used to access the corresponding segment descriptor. The descriptor provides a limit value which is checked against the 12 bit offset in the virtual address. If the reference is out of the bounds of the segment then an error trap occurs. The offset is added to the physical base address from the descriptor. The resulting 18 bit value is a physical address within the 256 K byte address space of the computer module also specified in the descriptor.

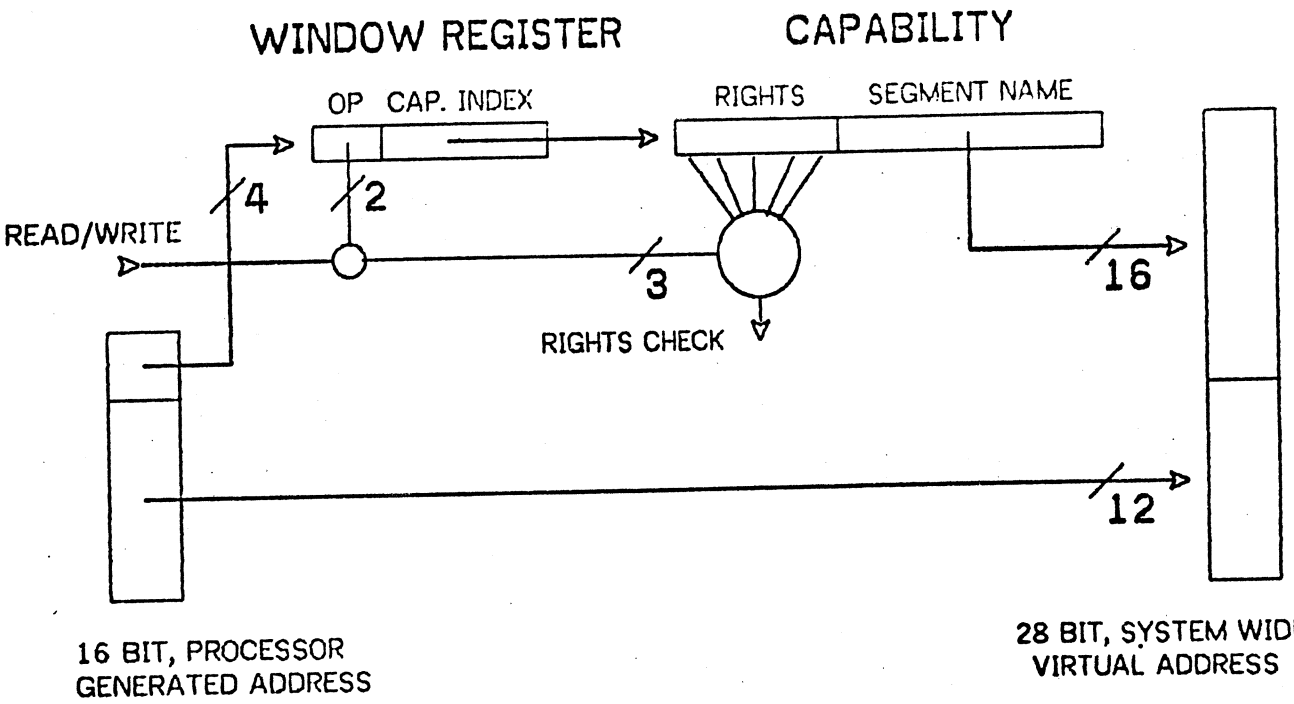


Figure 3.4 Conceptual Virtual Address Generation and Rights Checking.

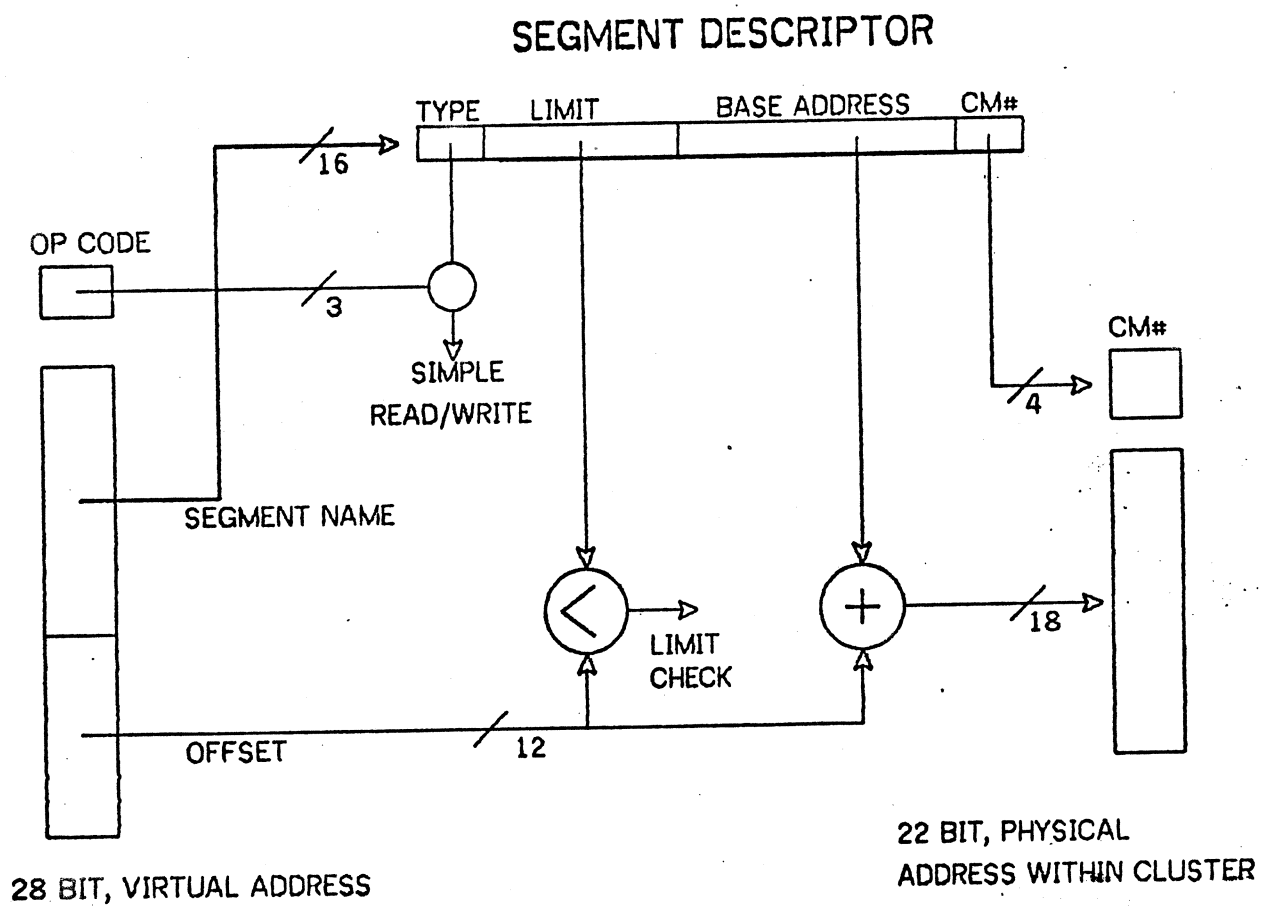


Figure 3.5 Virtual to Physical Address Mapping for a Variable Sized Segment

If the virtual address references a segment outside the source cluster then the segment name is used to access an *Indirect Descriptor Reference* rather than the descriptor itself. The indirect reference simply indicates in which cluster the segment resides. The Kmap then passes the virtual address to that cluster via the inter-cluster buses. An alternative approach would be to have duplicate copies of the segment descriptors in every cluster. Thus the virtual-to-physical mapping could be done at the source cluster, with possibly some savings in overhead. However, any attempt to change the virtual-to-physical binding of a segment (e.g. moving it to a different memory module or onto backing store) would require an effectively simultaneous change to all copies of the segment descriptor. In a large network this operation would be slow and cumbersome, if not impossible. A further advantage to ensuring that only a single descriptor exists for each segment is that a *Lock Bit* can be provided in the descriptor. The lock bit can be used to ensure mutual exclusion for special segment operations.

### *3.5 The Kernel Address Space*

Each processor can execute in either of two address spaces. One is the *User Address Space* which was described above. The second is the *Kernel Address Space*, which is similar to a user address space with the addition of some mechanisms reserved for the operating system. The currently executing address space is selected by a bit in the Processor Status Word of the LSI-11. A *Kernel Environment* is similar to a User Environment; however

segments at the third level of the Capability List structure (Figure 3.2) can be User Primary Capability Lists. That is, a Kernel Capability list structure can have user environments as substructures.

There are several additional pseudo registers provided in page 15 of the kernel address space. One of these, the *User Environment* register, holds an index for a Capability in the kernel environment which points to a user environment. This register specifies the current user environment for this processor. If the kernel writes a new index into the register the addressing state of the old user process is saved by the Kmap in the state vector part of the old user environment. The addressing state of the new user is then loaded from the specified new user environment. The addressing state is the value of the window and other system registers in page 15 of the executing program. Ideally, this operation, which performs a context swap by saving one addressing state and loading another, would also save the internal processor registers. Unfortunately there is no way for the Kmap to access the internal registers of an LSI-11. Thus internal registers must be saved and restored under program control.

### *4 The Use of the Addressing Structure for Control Operations*

The philosophy in Cm* is to implement all special control operations, such as interprocessor interrupts, by references to the physical address space. This not only avoids a proliferation of special control signals, but also allows the power of the system's address

mapping and protection mechanisms to be applied to control operations.

The Slocal provides a three priority level interrupt scheme. An interrupt is invoked by writing into the appropriate physical address on the LSI-11 bus of the target processor. Thus an interrupt can be requested by a process anywhere in the network, provided the process has a Capability for a segment which maps to the correct physical address. Another example is the abort operation. If the appropriate bit is written, a NXM (Non Existent Memory) trap by the local processor is forced. This mechanism will be used when an error occurs during a remote reference by the processor.

The following examples show how references to special typed segments, or special operations on standard segments, are used to invoke microcoded operations in the Kmap.

### *4.1 Primitive Lock Operations*

For processors in the PDP-11 family, most write operations are part of a read-modify-write sequence. In standard PDP-11s (including LSI-11's) this sequence is implemented as an indivisible, single bus operation. This improves performance by reducing bus overhead and allowing optimization of references to memory with destructive read operations (e.g. core and dynamic MOS memory). In C.mmp the indivisibility of these operations is maintained through the switch to shared memory. This allows the implementation of Locks and Semaphores because a memory location can be both tested and set without fear of an intervening access by some other processor. Indivisible

read-modify-write operations to nonlocal memory will not be implemented in Cm* because of increased bus and memory contention and hardware complexity. We will provide an equivalent function by making use of the Kmap's ability to lock a segment descriptor while it makes a series of references to the segment. To implement a basic lock mechanism two special segment operations are defined:

Inspect the word addressed. If greater than zero, then decrement. Return the original value.

Increment the word addressed. Return the original value.

*4.2 An Inter-Process Message System*

Message systems can provide particularly clean mechanisms for communication between processes [Brinch-Hansen, 73. Jefferson , 77]. In the past, a drawback to message systems has been the substantial operating system overhead in transferring a message from one process to another in a fully protected way. The architecture of Cm* provides an opportunity to build a fully protected message system which can be used with very low overhead.

A message port, or mail box, will be a special segment type. Messages will either be entire segments, passed by transferring capabilities, or will be single data words encoded as data capabilities. Two representative operations on Mailbox segments are:

Send( Message, ReplyMBox, MailBox)
This transfers capabilities for a message and a reply mail box from the caller's Capability List to the Mail box. If the mailbox is full then the caller is suspended.

Receive(MailBox)
If the mailbox contains a message then a Capability for the message and a Reply Mailbox will be transferred into the caller's Capability List. Otherwise the caller is suspended.

Provided that the above operations are successful, they are performed completely in Kmap microcode, and messages may be passed with probably less than 100 microseconds delay. If the operation cannot be completed because the Mailbox is full or empty, then the operating system is invoked to suspend the requesting process. The Kmap can also request the operating system to wake up a suspended process when the operation is complete.

## *5 Development Aids*

The development of hardware and software for a new computer system is a major undertaking. We have attempted to ease this burden by using a variety of aids. All the major hardware components were drafted using an interactive drawing package (a version of the Stanford Drawing Package). To facilitate the development of software, prior to the availability of hardware, a functional simulation of Cm* was programmed, which executes on C.mmp. Development of the Kmap hardware and microcode has been greatly benefited by the use of the "hooks" mechanism in the Kmap. This connection to the Kmap allows a program executing on an LSI-11 almost complete access to the internal state of the Kmap.

In order to expedite hardware debugging and software development, a host program development system was constructed. The host is connected to each Cm in the system by a

Serial Line Unit (SLU) to allow down line memory loading and dumping from the associated Cm. In addition, the SLU makes console control functions for each LSI-11 available to the host computer [van Zoren, 75]. The Host in turn is connected to a PDP-10 timesharing system.

## *6 Concluding Remarks and Project Status*

Cm* is projected to be constructed in three stages. The first stage is a ten-processor, three Kmap system. The subsequent stages will include 30-processors and later 100-processors. Detailed hardware design began in late July, 1975. As of late summer, 1976, a three-processor, one-Kmap system was operational. It is expected that the first stage Cm* configuration will be operational in the second quarter of 1977. The initial operating system is described in [Jones, et al. 77] and is being developed both on the Cm* simulator which runs on C.mmp and on the real hardware with the support of the Host Development system.

The essential features of the Cm* architecture have been presented. Both the coupling of a processor directly with each unit of shared memory and the three level bus structure which makes all memory accessible by every processor are primary features of the Cm* structure. Much of the sophistication in the architecture is associated with the address translation mechanisms. A description has been given of how the small processor address space of the PDP-11 is mapped into the larger global virtual address space of the Cm* system and how the global virtual address space is mapped onto the distributed physical

address space of the Cm* system. A number of important aspects of the Cm* project are outside the scope of this paper and interested readers are referred to other papers for a more complete discussion [Jones, et al. 77, Swan, et al. 76A, 76B, 77, Ingle and Siewiorek, 76A, Ingle and Siewiorek, 76B, Siewiorek, et al. 76], Reliability and performance models have been developed concurrently with the hardware design of the system and have been used to guide several important decisions concerning the structure of the Cm* implementation.

Acknowledgements

During the years of its initial development, many individuals have contributed to this project. Gordon Bell, Bob Chen, Doug Clark and Don Thomas contributed ideas to earlier versions of this architecture. Anita Jones and Victor Lessor have contributed to the present architecture. Miles Barel, Paulo Corrulupi, Levy Raskin and Paul Rubinfeld have all contributed to bringing the hardware to an early fruition. Kwok-Woon Lai and John Ousterhout are largely responsible for the successful development of the Kmap. Andy Bechtolsheim designed the Linc. Lloyd Dickman, Rich Olsen, Steve Teicher and Mike Titelbaum at Digital Equipment Corporation have provided information, ideas, and support critical to the success of the project.

*References*

[Anderson and Jensen, 76] Anderson, G. A. and E. D. Jensen., "Computer Interconnection Structures: Taxonomy, Characteristics and Examples", *Computing Surveys* 7, 4, December 1975, 197-213.

[Bell et al. 1972] Bell, C. G., J. L. Eggert, J. Grason, and P. Williams, "The Description and the Use of Register Transfer Modules (RTMs)," *IEEE Transactions on Computers*, Vol. C-21, No. 5, May 1972, 495-500.

[Bell et al. 1973] Bell, C. G., R. C. Chen, S. H. Fuller, J. Grason, S. Rege, and D. P. Siewiorek, "The Architecture and Applications of Computer Modules: A Set of Components for Digital Design," *IEEE Computer Society International Conference*, CompCon 73, March 1973, 177-180.

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

[Brinch-Hansen 1973] Brinch-Hansen, Per, *Operating System Principles*, Chapter 8, "A Case Study: RC-4000," Prentice Hall, 1973.

[Denning 1970] Denning, P. J., "Virtual Memory," *Computing Surveys*, Vol. 2, No. 3,

September 1970, 153-190.

[Fuller et al. 1973] Fuller, S. H., D. P. Siewiorek, and R. J. Swan, "Computer Modules: An Architecture for Large Digital Modules," *Proceedings of the First Annual Symposium on Computer Architecture*, University of Florida, Gainesville. Also in ACM SIGARCH, *Computer Architecture News*, Vol. 2, No. 4, December 1973, 231-236.

[Heart et al. 1973] Heart, F. E., S. M. Ornstein, W. R. Crowther, and W. B. Barker, "A New Minicomputer/Multiprocessor for the ARPA Network," *IFIPS Conference Proceedings*, Vol. 42, NCC 1973, 529-537.

[Ingle and Siewiorek, 1976] Ingle, Ashok and D. P. Siewiorek, "Reliability Modeling of Multiprocessor Structures," *Proceedings IEEE CompCon '76*, September 1976.

[Ingle and Siewiorek, 1976B] Ingle, Ashok and D. P. Siewiorek, "Reliability Models for Multiprocessor Systems with and without Periodic Maintenance", Computer Science Technical Report, Carnegie-Mellon University, September 1976.

[Jefferson, 1977] Jefferson, David, "The Hydra Message System," to be published.

[Jones, et al. 77] Jones, A. K., R. J. Chansler, I. Durham, P. Feiler and K. Schwans. "Software Management of Cm*, a Distributed Multiprocessor, Submitted to the 1977 National Computer Conference.

[Siewiorek, et al. 76] Siewiorek, D. P., W. C. Brantley Jr., and G. W. Lieve, "Modeling Multiprocessor Implementations of Passive Sonar Signal Processing", Final Report, Carnegie-Mellon University. Pittsburgh, Pa. 15213, October 1976.

[Swan et al. 1976A] Swan, R. J., L. Raskin, and A. Bechtolsheim, "Deadlock Issues in the Design of the Linc," Internal Memo, March 1976.

[Swan et al., 1976B] Swan, R. J., S. H. Fuller and D. P. Siewiorek, "The Structure and Architecture of Cm*: A Modular, Multi-Microprocessor". *Computer Science Research Review 1975-76*, Carnegie-Mellon University, Department of Computer Science, Pittsburgh, Pa., December 1976, pp 25-47.

[Swan, et al. 1977] Swan, R. J., A. Bechtolsheim, K. Lai and J. Ousterhout. "The Implementation of the Cm* Multi-Microprocessor", submited to the 1977 National Computer Conference.

[Van Zoren, 75] Van Zoren, H. "Cm* Host User's Manual", Department of Computer Science, Carnegie-Mellon University, December 1975.

[Wulf and Bell 1972] Wulf, W. A. and C. G. Bell, "C.mmp - A Multi-Mini- Processor," *AFIPS Conference Proceedings*, Vol. 41, part II, FJCC 1972, 765-777.