THE I/O PORT ARCHITECTURE FOR COMPUTER MODULES

S. H. Fuller and R. C. Chen

Departments of Computer Science
and Electrical Engineering
Carnegie-Mellon University
Pittsburgh, Pennsylvania

March, 1973

This work was supported by the National Science Foundation under Grant GJ-32758X.

ABSTRACT

In an earlier paper [Bell, et al., 1973] a new set of digital modules, called Computer Modules, or simply CM's, is described. In that paper it was stated that the correct design of the CM I/O structure is critical to their success, and moreover, the I/O structure of a CM is what distinguishes it from contemporary minicomputers. This report describes a preliminary design of the I/O structure for computer modules.

The functions of the I/O ports are discussed here, and it is observed that they must allow sharing of memory and passing of control among interconnected CM's. CM's use a variation of the segmented address space scheme to implement the functions of the I/O port. A complete ISP description of the I/O port is included, as well as a number of examples of inter-CM communications that illustrate the various ways the I/O ports can be used.

1. INTRODUCTION

This report describes a preliminary design of the input/output ports of Computer Modules. Computer Modules are building blocks that will be used to construct a wide variety of digital systems, spanning a wide range of cost and performance. Our basic assumptions and the reasoning leading to the choice of Computer Modules as building blocks are described elsewhere [Bell et al., 1973] and will not be repeated here. For completeness, however, we will briefly note the assumptions that are necessary for understanding this report.

Each Computer Module (CM) includes a processor of about the power of a minicomputer processor, local memory ranging from 1K to 64K words, with 8 or 16 bits per word, and a few input/output ports through which it communicates with other CM's. A few to perhaps several thousand CM's are used in any digital system, high performance being achieved by using multiple CM's in parallel. For CM systems to obtain high performance CM's must be able to communicate efficiently in a closely coupled manner. For this reason, the input/output port design is a critical aspect of the design of CM's.

This report is a design of the input/output port at an ISP level. The report does not address such important questions as the arbitration of requests to use a communication bus.

In the next section, Section 2, the functions of the I/O port are examined in detail. Section 3 follows with a description of the architecture of the I/O ports. Section 3 assumes the Pc of a CM is similar to a PDP-11 [Bell et al., 1970]. This was done for two reasons: choosing a reasonable well known processor relieves us of the task of describing the processor; and at least the first few CM's will in fact use PDP-11 processors. Section 4 gives a brief example of a PDP-11 CM used as a Pio* with block transfer abilities.

2. FUNCTION OF I/O PORTS

At the most fundamental level, the I/O ports of the CM exist to facilitate inter-processor communication; the primary purpose of the I/O ports is not to boost the processing power of a CM. If CM's are being applied to a problem that can benefit from a powerful Pio (I/O processor), then one or more CM's should be used to build the Pio; the Pio should not be a component of a CM.

Current multi-processor and multiple computer systems provide for two types of inter-processor communication. There exists schemes for:

1. sharing data;
2. passing control among processors (processes).

2.1. Data Sharing

There are many ways for processors to share data; they can share some secondary storage via separate controllers, they can have channel to channel links, or they can share part, or all, of Mp (e.g., the archetypical multi-processor structure). Clearly, the most elegant solution here is to give all the processors in the system access to all the words in the address space. (Postpone for a moment the issue of protection.)

C.mmp, a multi-processor with a complete crossbar switch [Bell and Wulf, 1972] allows all the Pc's to share the same address space. The

*We use the PMS notation of Bell and Newell [1971] in this report to describe the hardware organization of CM's.

question must be answered: why should C.mmp not be the prototype of a <u>large module</u> (LM) set of Pc's, Mp's, and Smp's (memory-processor crossbar switches)? There are at least two reasons:

1. In C.mmp several hundred nanoseconds are lost in an Mp access by a Pc. This is because of Pc conflicts for Mp modules as well as switching delays.

2. A central Smp (crossbar switch) reduces the potential reliability of the system. Smp can be distributed among the Mp's (or Pc's) but then Problem 1 is further aggravated.

A proposal to increase the performance of C.mmp involves attaching local cache memories to each Pc. Note C.mmp would then have Pc-Mp pairs: the first step toward CM's.

2.2. <u>Control Passing</u>

Two structures are in common use in computer systems to implement control passing: message queueing and interrupt structures. Roughly, interrupt structures are important in real-time control computers, and mini-computers in general, because of demanding response requirements, while message queueing is used in multi-programming and time sharing operating systems because of its flexibility. If CM's are to be used as primitive components in digital system design, they must be adaptable to a wide range of performance requirements; hence some form of interrupt mechanism is essential. Message queueing can be implemented in software when needed.

2.3 Other I/O Port Functions

Although CM I/O ports must implement some form of shared memory and inter-process control signaling, are these functions sufficient for the efficient operation of CM's? We don't know; experimentation with CM's should investigate this question.

A function that is present in the I/O structure of most high-performance computers is the ability to transfer blocks of information in or out of Mp via an I/O port, independent of the Pc. Consideration should be given to enabling the I/O ports to supervise block transfers, in parallel with the operation of the Pc.

This concludes the brief functional description of the CM's I/O port. Since the I/O ports are not implemented as stored-program processors, they will be called I/O controllers, i.e., Kio's, rather than I/O processors.

3. THE ARCHITECTURE OF A Kio

3.1. Implementation of Data Sharing

The most obvious scheme to allow Pc's to share data is to give them all the same global linear address space. This naive scheme is lacking on a number of counts: the linear address space must be very large ($\sim 2^{30}$ words) in order to handle contemporary problems; a linear address space for a CM network does not match the underlying multiple CM structure; and a protection scheme is not easily embedded in a linear address space.

An addressing scheme that more closely matches the structure of CM systems is a segmented address space [cf. Denning, 1970 and Randell and Kuehner, 1968]. Specifically, a CM's address space will contain a set of segments;

segments will be called local segments or external segments, depending on their function. Local segments map words in the virtual address space into the physical memory space of the Mp local to the CM. External segments map words in the virtual address space out of the CM onto an inter-CM bus, and ultimately into the physical memory space of another CM. One of the two major functions of the Kio's is to implement this inter-module address mapping function. There exists a bewildering variety of ways to implement a segmented address space (e.g., B5000, IBM 360/67 (and now the 370's), GE 645, PDP-11/45,...). The following design is driven by:

1. a desire for simplicity in the Kio;
2. delays in the Kio must be kept to an absolute minimum;
3. realization the Pc of initial CM's is the PDP-11.

The Broadcast Address Space

There must exist some addressing, or tagging, scheme on the inter-CM bus so CM's can exchange messages over a shared bus. Initially, we gave unique names to each Kio on a bus and CM's sent messages to other CM's, using these names to route the calls. This scheme has the disadvantage that the instantaneous processor-process binding for all the CM's in the system has to be known by each CM initiating messages.

The second scheme developed was to bind names to the messages, rather than the CM's. This is more reasonable and was the approach used in much of the early design. However, the concept of a broadcast address space has resulted in a startling simplification and unification of the concepts of CM memory sharing, even though it is really our second scheme in a clever disguise.

Let each bus in a CM system have a virtual address space, called a broadcast address space. To access a word not in the CM's local segment, the Pc requests a word from an external segment, and the appropriate Kio translates the address from the CM's address space into the broadcast address space of the bus. The other CM's on the bus are monitoring the address lines, and when an address is within the range of a bus segment, the Kio maps the bus address into the address space of the remote CM.

The broadcast address space of a bus is analogous to the electromagnetic broadcast spectrum. Like the broadcast spectrum, the broadcast address space will become a limiting, critical resource when many CM's (stations) concurrently require segments (broadcast bands) for transmission of messages. Figure 3.1 shows how an address is mapped from CM A into CM B via the broadcast address space of bus 1. If an address is mapped into an external segment of CM B, then another level of mapping is initiated, as shown in Figure 3.2. The ability to have multi-level mapping allows a CM, e.g., CM B in Figure 3.2, to act as a switch. This circuit switching ability is crucial if networks of CM's need to interact in a closely-coupled manner. If only relatively loose coupling is needed, as, for example, in the ARPA network, then CM's can implement message switching by mapping addresses into local segments and then have the local Pc move the message from the local segment to an external segment; see Figure 3.3. The fact that CM's can function as circuit or message switches substantially enlarges the range of applications they can efficiently accommodate.

Figure 3.4 illustrates another way the broadcast address space can be used. In Figure 3.4(a) several CM's are set up to map the same bus segment

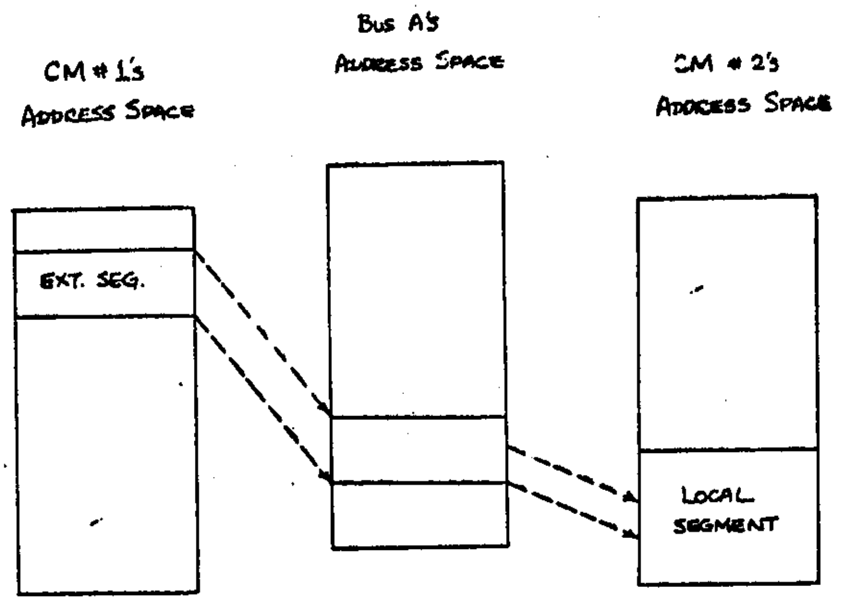


FIGURE 3.1
ADDRESS MAPPING ON A CM BUS

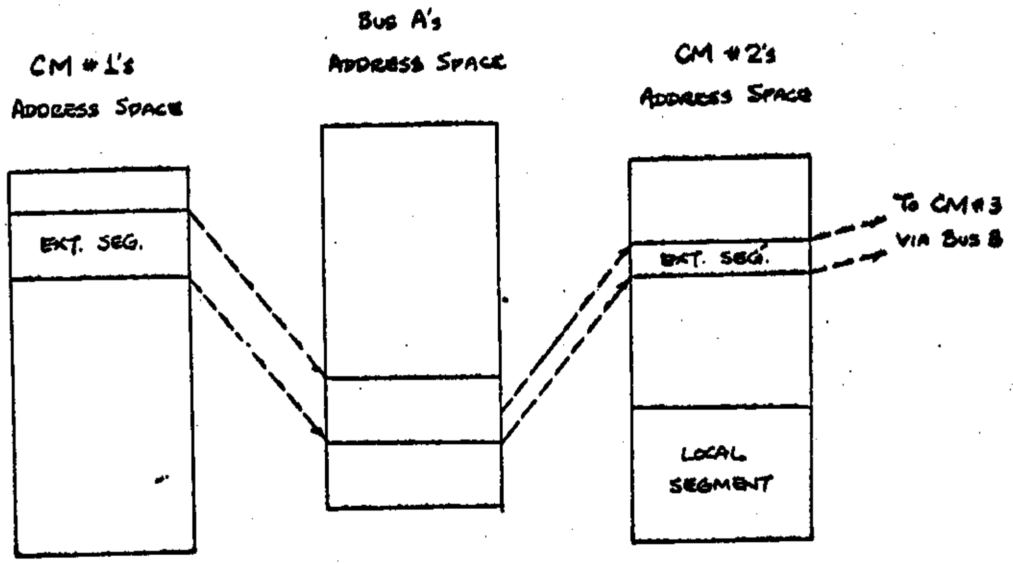


FIGURE 3.2
CIRCUIT SWITCHING

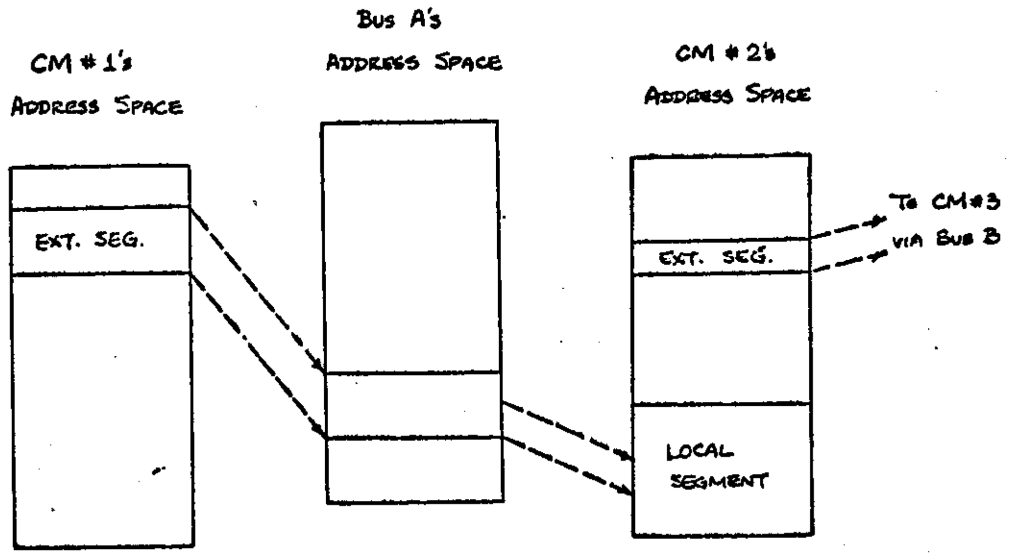


FIGURE 3.3
MESSAGE SWITCHING

FIGURE 3.4
SHARED SEGMENTS IN BROADCAST ADDRESS SPACE

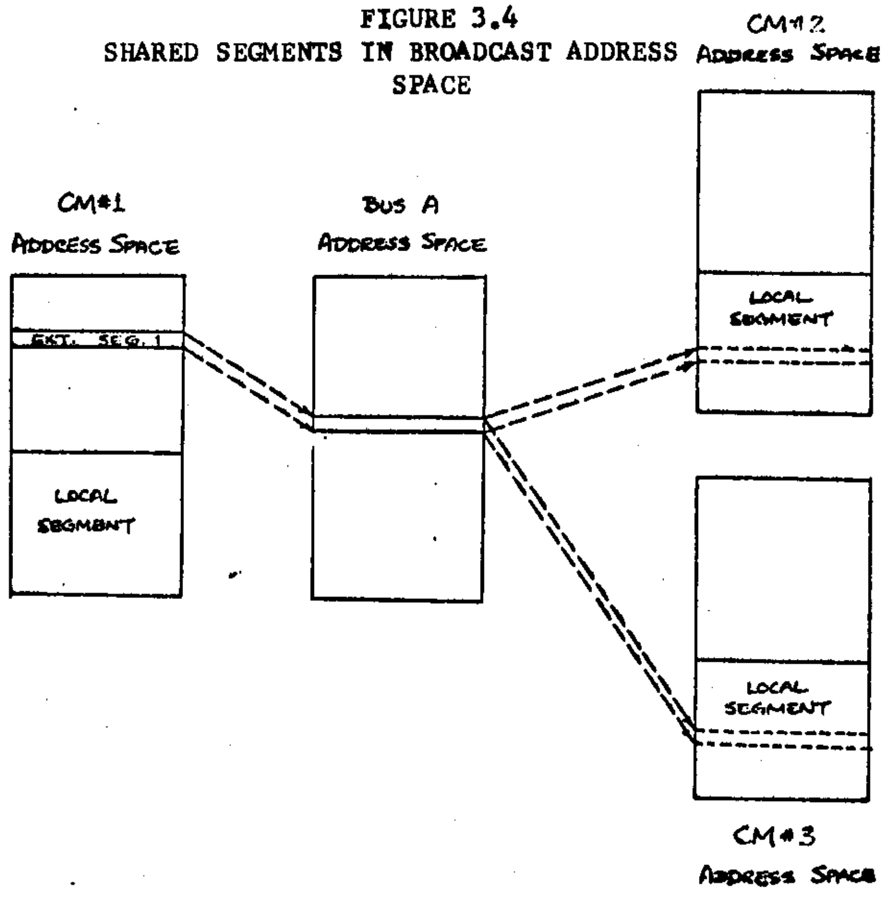


FIGURE 3.4(a)
MANY-TO-ONE MAPPING

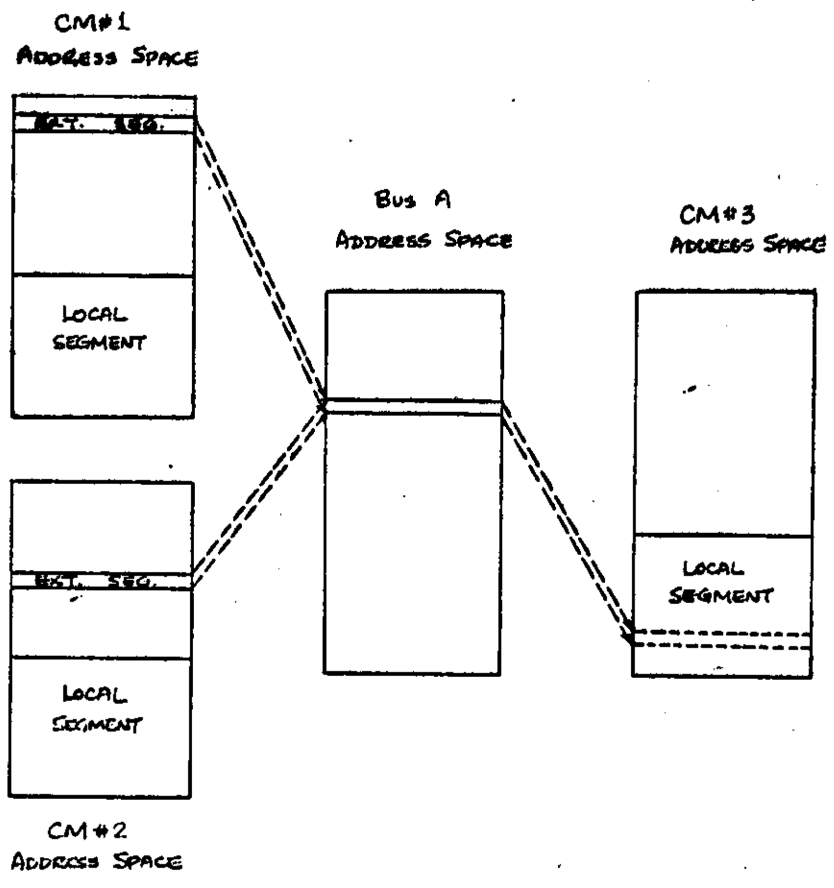


FIGURE 3.4(b)
ONE-TO-MANY MAPPING (BROADCASTING)

into their local segments. This arrangement gives CM systems a broadcast or one-to-many mapping ability; for example, CM A in Figure 3.4(a), by writing into external segment 1 simultaneously sends information to the other CM's in the figure. On the other hand, Figure 3.4(b) shows how a CM system can implement a many-to-one mapping. This arrangement is needed whenever several concurrent processes share a common data structure.

Variable Segment Size

A number of schemes have been used to specify a segment in the address space:

1. Fixed size segments, i.e., pages. This naturally suggests pages be mapped into page frames, rather than an arbitrary place in memory.

2. Arbitrary segment size, e.g., B5000. The extent of a segment is specified by a base, limit register pair. This is completely general, but problems with memory fragmentation arise, and addition rather than equality testing must be used to map an address into the segment.

3. Paged, variable size segments. Classic implementation of virtual memory systems, e.g., S360/67 and GE 645 [cf. Dennis, 1965; Arden et al., 1966].

The more complex of the above schemes are not appropriate for CM's because the address mapping must be an efficient process if CM's are to be able to cooperate in a closely-coupled manner. On the other hand, it is important that the external segments be variable size. There are three reasons for this:

1. CM address space. At present, assuming PDP-11 CM's, a CM has a variable number of external segments and to fix external segment size would put an upper limit on the number of external segments per CM.

2. Protection. In order to efficiently access data structures, external segments must be as large as the largest data structures, but to protect adjacent data structures from unwarranted destruction, external segments need to be as small as the smallest data structures.

3. Broadcast address space. Some applications will require a large number of CM's per bus, and unless external segments are of variable size the internal fragmentation of bus segments will result in a critical shortage of words in the broadcast address space.

ISP Description of a Kio

The next page gives the ISP of a Kio. See Bell and Newell [1970] for details of the ISP notation and Bell et al. [1970] for the ISP of the PDP-11 processor, the processor we will be using in the initial CM's.

Figure 3.1 should help clarify the address translation processes from CM to bus and back again. By giving the bus a broadcast address space, the symmetry between CM to bus and bus to CM translation is sufficiently complete

ISP Description of a Kio

Kio State

CM␣Segment␣Register/CSR[0:15]<15:0>	
Bus␣Segment␣Register/BSR[0:15]<15:0>	
Control␣Register/CR[0:15]<15:0>	
Logarithm␣of␣Segment␣Size/LSS<3:0>	:= CR<15:12>
Write␣protect/WP	:= CR<11>
Reference␣bit/RB	:= CR<10>
Change␣bit/CB	:= CR<9>
Interrupt␣enable/IE	:= CR<8>
Interrupt␣pending/IP	:= CR<7>
I/O␣bit/IOB	:= CR<6>
NXM	:= CR<5>
Spare bits	:= CR<4:0> Currently unassigned bits.
Mask Register/MR<15:0>	:= a pseudo register that is the unary encoding of LSS field in CR.

Address Translation Process

```
y<15:0> := (IOB = 0) → (                                map from CM to bus.
              (z ∧ MR  =  CSR ∧ MR) →                   effective address is
                                                        within external segment.
                  (BSR ∧ MR) ∨ (z ∧ (¬ MR))             concatenation process
                          )

z<15:0> := (IOB = 1) → (                                map from bus to CM.
              (y ∧ MR  =  BSR ∧ MR) →                   broadcast address is
                                                        within bus segment.
                  (CSR ∧ MR) ∨ (y ∧ (¬ MR))             concatenation process.
```

Where: z := effective address of operand in CM address space
and   y := effective address of operand in bus broadcast address space.

to allow CSR/BSR pairs to translate addresses in either direction, the current direction being specified by the I/O bit in the CR.

In the ISP, a Kio is shown to have 16 CSR/BSR/CR sets. This is not meant to imply that all, or even most, Kio's will have this many register sets. Two to four register sets are presently seen as a practical number to have on a single Kio. In fact, if only one CSR/BSR/CR set is included in each Kio, no power is lost because nowhere is it necessary to make the restriction a CM can only have one Kio per bus. However, to conserve bus drivers and receivers it is appropriate to group several register sets on a single Kio.

Note in the above ISP we have solved the segment size problem. Only segment sizes that are powers of two are allowed: 1,2,4,8,...,64K. By restricting our segment sizes to powers of two we avoid the addition implicit in more conventional base/limit schemes. We also ease the memory fragmentation problem in the broadcast address space since we can now use the buddy system. This novel memory allocation scheme was independently discovered by H. Markowitz (1963) and K. Knowlton (1965) and is analyzed by Knuth [1968]. The DEC Unibus window [DEC, 1972] also limits its segment (window) size to be a power of two. However, the Unibus window sets a lower limit on segment size, 512 bytes, and this appears to be a mistake. Many data structures that are shared between processors are smaller than 512 bytes, yet the cost to provide smaller segment sizes is negligible.

Most of the fields in the control register are self-evident. The fields dealing with inter-CM interrupts will be discussed in the next section. The NXM bit is used to enable/disable the CM-bus translation for the associated CSR/BSR pair.

3.2 Implementation of Intermodule Interrupts

It is useful to consider inter-CM interrupts to be a generalization of subroutine calls. Specifically, we need a jump to subroutine (JSR) instruction that passes control to the called procedure, as is now done, and we need JSR's that initiate new loci of control, without suspending the locus of control in the calling program. In other words, we need to implement, in the context of CM's, what has been variously called fork, begin parallel, split, etc. This new JSR (let's call it ISR, for initiate subroutine) also seems to have utility within a CM as a clean way to implement multi-programming and multi-processing.

While the format of the ISR instruction looks similar to the JSR instruction in the calling CM, to the target CM the ISR looks like an interrupt. If the interrupt enable (IE) is on in the Kio of the target CM, then the address of the ISR instruction is used as the interrupt vector for the target CM. The data lines of the inter-CM bus are used to pass the current value of the program counter to the target CM. If the interrupt enable is off, then the interrupt pending bit is set in the CR and the interrupt vector is stored in a buffer register. This buffer register is not directly accessible by any CM in the system, but it is effectively cleared by clearing the interrupt pending bit of the CR.

If a Pc executes a JSR, rather than an ISR, into an external segment of its address space, we will simply have a situation where the Pc of one CM is fetching its instructions from the Mp of another CM.

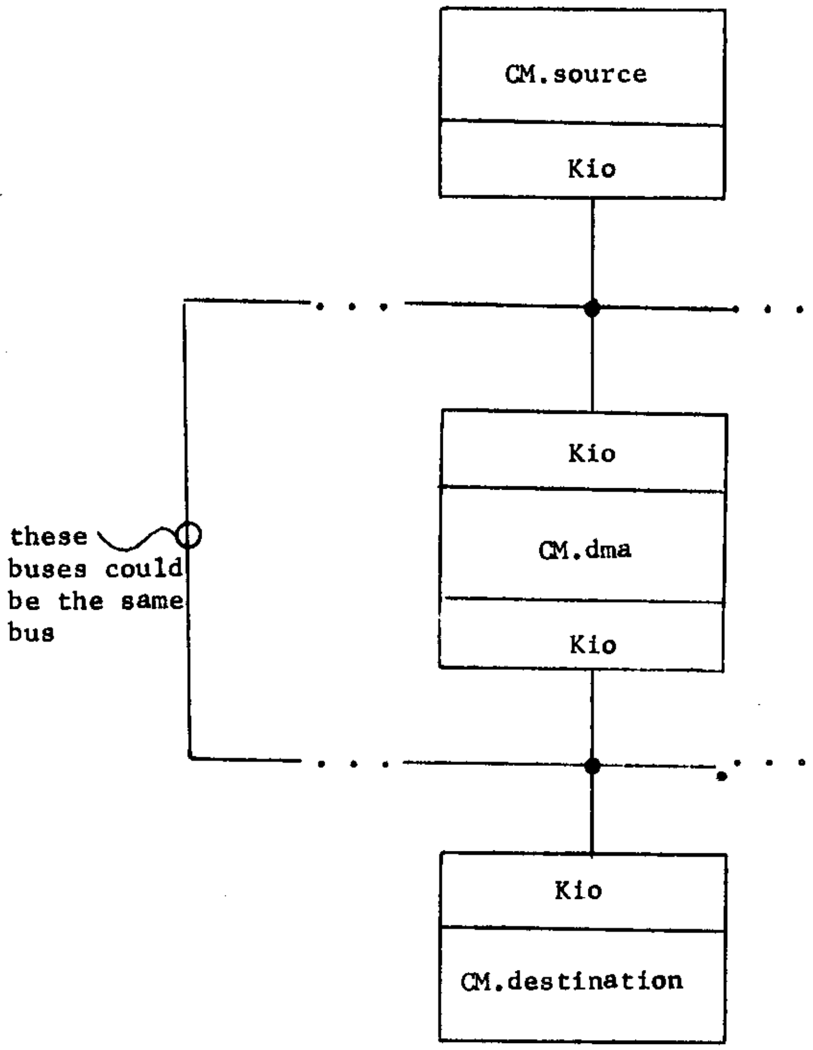
It would be consistent with the above implementation of interprocessor interrupts to generalize the Wait instruction of the PDP-11 to implement the join, or merge, function needed in parallel processor systems. However, either the increment-and-skip-if-zero (ISZ) instruction common to most mini-computers or the "read-pause" feature of C.mmp [Wulf and Bell, 1972] is sufficient.

4. ASYNCHRONOUS BLOCK TRANSFERS

In many contemporary computer systems the ability of I/O processors to transfer large blocks of information, independent of the Pc, is important for the efficient operation of the system. This report proposes that the Kio's of CM's not have this direct memory access (DMA) ability. If a DMA channel or Pio is needed, it can be built from a CM. Figure 4.1 shows a CM functioning as a DMA channel.

ACKNOWLEDGMENTS

This report discusses work that is part of a larger project that includes Gordon Bell, John Grason, Daniel Siewiorek, Douglas Clark, Satish Rege, and Richard Swan. All have made suggestions that have improved the design presented here.



Now if we have the simple PDP-11 code sequence in the CM.dma:

```
        ⋮
        MOV  (Sr)+,(Dr)+
        CMP  Sr, Limit␣Register
        BNE  *-2
        ⋮
```

The Sr and Dr point to words in external segments of the CM.dma's address space.

FIGURE 4.1. Using a CM as a DMA Channel.

REFERENCES

1. Arden, B. W., Galler, B. A., O'Brien, T. C., and Westervelt, F. H. (1966), "Program and addressing structure in a time-sharing environment." J. ACM 13, 1 (Jan., 1966), 1-16.

2. Bell, C. G. and Newell, A. (1971), Computer Structures: Readings and Examples. McGraw-Hill Book Company, New York, N. Y. (1971).

3. Bell, C. G., Cady, R., McFarland, F., Belagi, E., O'Laughlin, J., Noonan, R., and Wulf, W. (1970), "A new architecture for minicomputers -- the DEC PDP-11." Proc. AFIPS SJCC 1970, 657-675.

4. Bell, C. G., Chen, R. C., Fuller, S. H., Grason, J., Rege, S., and Siewiorek, D. (1973), "The architecture and application of computer modules: a set of components for digital design." IEEE CompCon '73 (March 1973), 177-180.

5. DEC (1972) DA11-F Unibus Window Maintenance Manual. DEC-11-HDAFA-A-D. Digital Equipment Corporation, Maynard, Mass.

6. Dennis, J. B. (1965), "Segmentation and the design of multiprogrammed computer systems." J. ACM 12, 4 (October 1965), 589-602.

7. Denning, P. J. (1970), "Virtual Memory." Computing Surveys 2, 3 (September 1970), 153-191.

8. Knuth, D. E. (1968), The Art of Computer Programming, Volume 1/Fundamental Algorithms. Addision-Wesley Company, Reading, Mass. (1968).

9. Randell, B. and Kuehner, C. J. (1968), "Dynamic storage allocation systems." Comm. ACM 11, 5 (May 1968), 297-305.

10. Wulf, W. A. and Bell, C. G. (1972), "C.mmp -- a multi-mini-processor." Proc. AFIPS FJCC 1972, 765-777.