THE MEGABYTE MINI

Gregory L. Hopwood
Donald C. Loomis
Julian Feldman

Technical Report No. 54
January 1974

A Proposal to Attach LCS to Minicomputers

Department of Information and Computer Science
University of California, Irvine
Irvine, California
92717

January 7, 1974

SUMMARY

The major difference between minicomputers and larger computers is a state of mind. Part of that state of mind is the notion that minicomputers have small memories. Given current technology, there are no technical reasons why minicomputers cannot have large main memories. In this proposal we discuss the issues involved in configuring large memory (megabyte or larger) minicomputers.

In the first section of the proposal, we review some of the concepts associated with memories and some of the pressures for expanding primary memories. The possibility of replacing the primary memory and swapping subsystem of a multi-user timesharing mini with an extended core memory is discussed in particular detail.

The technical issues in developing the megabyte mini are presented in the second section of the proposal. Potential problems and means for solving them are discussed, and ways of organizing non-conventional memory addressing are introduced.

The technical description of a controller for the megabyte memory is presented in the third section.

The megabyte mini has three major markets. The first is for applications which require or can economically use

large memory. The second is for systems which now use modest size core memories and swapping devices. At today's LCS prices, it is possible to replace such configurations with megabyte memories at less cost with better performance and greater reliability. The third market is one in which modifications to the memory controller could significantly improve performance. This market is sometimes referred to as the "smart memory" market.

Just as the small computer with a large file system configuration is becoming popular, we believe that there is a substantial market for minicomputers with megabyte or larger memories.

I. THE LARGE MEMORY MINICOMPUTER

The problem, briefly stated, is that minicomputers normally are restricted in the amount of main memory which they can address by the small number of bits available in a memory word. The most common word size for minicomputers is sixteen bits. This allows the computer to generate, at most, 2^{16} or 65,536 (64K) different addresses. Many minicomputers use a bit in the address word to indicate indirect addressing and/or byte addressing, and thus the number of words which they can address directly is reduced by a factor of two to 32K. (Henceforth, memory sizes will be in words unless specified otherwise.)

The set of words (or data items) addressable by a program is called its logical (or virtual) address space. The set of words of main memory actually attached to the machine determines its physical address space. The size of the logical address space determines the maximum size of a program which can theoretically be run on the machine, and (in the case of multiprogramming) the size of the physical address space practically limits the program size and determines the maximum number of jobs of a given size which can reside in the main memory at one time. Even with a full complement of main memory (i.e., the logical address space

equal to the physical address space), most 16-bit minicomputers are limited to a maximum of 32K-64K of program and data storage.

In the past there was little reason to put more than 32K-64K of core memory on minicomputers. A relatively small memory was sufficient for most tasks for which minicomputers were used. Besides, since the machine could not address any more memory, an increase in memory size without a convenient hardware mechanism for increasing the addressing capability of the machine would be useless. This paper discusses such an enhancement feature.

Minicomputers today are only small in physical size compared with their giant counterparts of fifteen to twenty years ago. It is common for minicomputers to be able to perform more than one million instructions per second. This level of computational power has opened up new markets for minicomputers, markets formerly the realm of much more expensive machines.

Minis are not only being used in their traditional roles as controllers for machinery, communications processors, front-ends, or laboratory data-gatherers, but also as the central machine in time-sharing systems, graphics display systems, and general accounting systems. Many of these tasks involve multiprogramming with foreground

and background partitions, terminal communication, and real-time response rates to remote experiments. These sophisticated tasks require the services of an operating system for the scheduling of resources. If the available core storage on the machine is 32K, and 16K of that is used for the operating system, then the effective size of the machine as far as the user is concerned is 16K. The more users to be kept in core at the same time, the smaller share of memory each one can occupy.

For time-sharing applications involving multiple users on a conventional mini not only is the job size of a user severely limited, but the number of user jobs which can be held in main memory concurrently is also limited. In a single user system, where the program is too large to fit into memory at one time, a method for running the program is to break it into independent parts (overlays) and transfer them to and from main memory as they are needed via a disk or drum. In a multi-user system, the operating system can manage the main memory use and swap each job or part of a job in and out of memory as needed. Of course, this swapping requires a substantial amount of system overhead and additional equipment when compared to a system which could keep all of its jobs in core at once.

The demand for increased main memory capacity for

minicomputers has led to the recent introduction to the marketplace of 16-bit machines with a memory enhancement feature called a memory map. The hardware box essentially translates (maps) the logical addresses generated by the machine into specific areas of a large core memory by extending the length of the addresses put on the memory bus to the length necessary for addressing the physical address space of the large core store (LCS). For a byte-addressed four million byte memory this would be twenty-two bits. The bits added to the 16-bit address to form the 22-bit address are selected by the mapping hardware from a set of hardware registers called the memory map. (The map registers are loaded and changed by a supervisor program.) Once the new 22-bit address is generated by the mapping hardware, it is presented to the memory address register of the LCS for a memory cycle. Extension of the mapping concept to include memory protection is very simple and is almost always included in a map system.

Table 1 lists those minicomputer manufacturers who are currently offering memory map options on their machines.

Table 1. Minicomputer manufacturers who have announced a 16-bit machine with a memory mapping feature.

Manufacturer	Model	Max Core
DEC	PDP-11/40-45	124K
Data General	Nova 840	128K
General Auto.	SPC-16/65	128K
Prime Comp.	Prime 300	256K
Varian Data	V-70 series	256K

Table 2 lists those manufacturers of larger computers who have used the memory mapping concept in their machines. Most of these machines have a very large logical address space relative to their physical address space because they have addresses of up to twenty-four bits or more in length. Their map facilities were introduced to allow paging of main memory to secondary storage. Table 2 is included to indicate that mapping concepts are not a new hardware invention, but have been around in the large machines for about fifteen years. Primitive mapping schemes were introduced with memory bank selection registers even earlier than the machines mentioned.

Table 2. Manufacturers of large machines using a mapping feature.

Manufacturer	Model	Date Delivered
Ferranti	Atlas	1963
Burroughs	5500	1965
	6500	1969
XDS	940	1966
	Sigma 7	1968
GE	645(MULTICS)	1966
IBM	360/67	1967
	370	1972
RCA	Spectra 70/45	1968
DEC	KI-10	1972

The demand for large main memory capacity on minicomputers will increase as memory mapping hardware becomes available and the cost of core or semiconductor memory becomes lower. Also a substantial number of the minicomputer installations presently in existence would be interested in an add-on memory map and bulk core product that could be field-installed on a machine that was not originally designed to interface with such a system. Users of newer computers which have a memory map capability but who did not order the option would be potential customers for a bulk memory system which could be offered at a lower price than that which could be bought from the manufacturer. Also, minicomputer memory sizes on the order of a megabyte or more are not being offered by the mainframe manufacturers. Stores of this size are common, however, in

the 360/370 add-on market. These stores or the technology used to create them could be adapted easily to provide multi-megabyte minicomputer memory systems.

Availability of LCS memory systems for minicomputers causes certain interesting economic and performance questions to arise when one attempts to configure a mini-system for tasks which are limited by main memory size. The suggested equipment for a DEC PDP-11/45 system capable of supporting twenty-four BASIC language time-sharing users (RSTS/E) includes 56K words of core memory and a swapping subsystem. Assuming user areas of a fixed size of 8K each reside in the 56K of memory, at most seven user programs may be resident in core at any one time. Seventeen users must be on the swapping device.

If the core memory and swapping device were replaced with a megabyte of LCS with mapping capability, all twenty-four 8K users could reside in memory at once. In a megabyte memory twenty-four resident users of 8K each would occupy 24x8K=192K words and leave 512K-192K=320K words unused. With such an LCS system the machine could keep all thirty-two user programs of a total of 16K words each (the maximum program size on the PDP-11/45 system) in memory at one time with no swapping whatsoever (32x16K=512K words). An equivalent DEC system is not available. The largest main

memory configuration that may be put on the PDP-11/40-45 machines is 124K words. This would allow only seven 16K users in core at once.

The obvious advantages of an LCS system over a swapping system such as the PDP-11/40-45 system are:

1. elimination of the swapping device which is more prone to mechanical and electronic failure than the LCS;

2. elimination of swapping which reduces supervisor complexity and overhead and reduces non-essential memory accesses;

3. increase in the number of resident users and/or program size.

For systems not available with a memory map option, the availabilty of an LCS system provides a significant hardware upgrade not available before. In addition, as will be discussed later, the LCS could provide new non-linear mapping functions which are often simulated in software. Examples are the addressing of stacks, lists, and queues, special table lookup procedures, associative addressing, and hashing functions.

Other possible arguments and counterarguments which might be postulated regarding the addition of an LCS system to a minicomputer are listed below.

1. Mainframe manufacturer reluctance to maintain system--many minicomputer owners maintain their own systems; maintenance strategies similar to those used in the 360/370 add-on memory market can be used in the mini LCS market.

2. Compatibility with map and protection features (if any) of manufacturer are necessary so manufacturer's software works--such compatibility is not needed when using an LCS system on a machine which was not designed for a mapped memory; user-owned software can be modified to use the new features of the LCS memory system.

3. Increase in cost over standard system--the cost of an LCS system must be considered in relation to the equipment which it replaces and to any additional capability it may provide. On the 24-user PDP-11 system the DEC memory, memory map, and the swapping discs and controller would cost about $60,000 using DEC's new (January 1974) 16K memory board prices. The entire system configuration would cost between $150,000 and $180,000 including the recommended peripherals and twenty-four terminal devices. A megabyte of DEC memory (if you could put it on their PDP-11 machine) would cost about $200,000. This means that a megabyte LCS

system selling for between $50,000 and $100,000 would be an attractive economic alternative as well as quadrupling the available physical address space of the machine.

4. Decreased memory speeds of the LCS causes programs to run slower--LCS memories may be constructed of slower components than manufacturer supplied smaller memories in order to reduce the cost of the unit. However, with the addition of a semiconductor cache memory (high speed buffer between the LCS memory and the central processor) this speed differential can be eliminated and the LCS system could possibly run faster than the standard core available from the mainframe manufacturer. The lack of swapping decreases the number of unproductive memory cycles needed to run programs, and could increase the effective job throughput.

II. DESIGN CONSIDERATIONS

The use of a large amount of memory on a minicomputer requires the solution of a number of problems and evaluation of a number of tradeoffs. The importance of these problems is not that they are obstacles to use of large amounts of memory on minicomputers but that they are questions for which the best solutions are not obvious. Connecting a large amount of memory to a minicomputer will provide an opportunity for evaluation of the relative merits of alternative solutions to the tradeoffs. Each of the following sections summarizes a potential obstacle and some alternative solutions.

LINEAR ADDRESSING

Providing a linear address space which allows programs to address any of the memory locations using sequential addresses for instruction execution and data references is important to allow the use of existing computer programs. While memory could be treated as an external storage device similar to drum storage, the full effectiveness and efficiency of fast, random access storage can best be used by providing a means of addressing it for execution and operand reference in the same way as the conventional

minicomputer memory.

Because of the importance of this requirement, this proposal concentrates on evaluating alternatives for providing this linear addressing mechanism. The use of a paging strategy has been selected as the most promising. The fundamentals of a paging mechanism have been described previously. Details on the operation of the proposed implementation are given in the memory interface specification at the end of this report. For comparison with other mechanisms which have been rejected as less suitable, the operation of paging is summarized here.

The problem to be overcome in all of the solutions is to provide a means of addressing all of a large amount of memory using the address bits of the minicomputer. Minicomputers normally have only as many address bits as the word size--typically sixteen. To address four million bytes of memory requires twenty-two bits. Hence a useful means must be provided to convert the 16-bit address to a 22-bit address.

Paging

Paging has been used in larger computers (e.g., Atlas, SDS 940, IBM 360/67, 370) to provide a mapping between a virtual (logical) address space and real memory address space. As is indicated in Table 1 minicomputer

manufacturers are now beginning to market small machines with such a mapping capability. In a typical paged system the high order four bits of the 16-bit address may be used to address a very fast memory which would give the 10 high order bits of a 22-bit address. The low order twelve bits of the 16-bit address would be used directly as the low order twelve bits of the 22-bit memory address. Based on the values loaded in the fast memory (map), the high order four bits of the 16-bit address would specify a 4K byte (word) "page" in the memory. The low order twelve bits would directly specify the desired byte (word). Because there are four bits to select a word in the map, a program in the minicomputer can directly address any of sixteen pages which have entries in the map. By changing the entries in the map the program can gain access to any of the locations in the memory.

Extension Registers

Some computers (e.g., PDP-1 and SDS 930) have used extension registers to develop additional address bits. "Bank switching" on computers such as the PDP-8 are extension register implementations. When the high order bits of the short address are zeros, the short address is extended on the left with zeros for the required longer memory address. Thus the program can directly address

locations at the bottom of memory in the normal manner. To gain access to other locations the extension registers are used.

When the high order bits of the short address are not zero, they specify an extension register from which the high order bits of the 22-bit address are to be taken. The extension register technique is equivalent to a page map except the map entry used when the high order bits of the short address are all zeros cannot be changed and always specifies the first page in memory. The extension registers are equivalent to the other entries in a map. Since it is slightly more flexible and can be implemented just as easily, the full page map is a more desirable alternative.

Relocation Registers

Another alternative for developing 22-bit addresses from 16-bit addresses is to provide a relocation register which will be loaded with a value to be added to every 16-bit address to give the required 22-bit address. By placing the appropriate value in the relocation register the 16-bit address could be adjusted to reference any portion of the memory. When only one relocation register is used, all of the addresses will be relocated by the same amount. It is awkward to access data areas in memory not addressable using this one relocation register value. The value of the

relocation register must be changed for each access to a different area.

The use of several relocation registers can provide the same ability to translate 16-bit addresses into arbitrary areas of the 22-bit memory as the paging scheme. The high order bits of the 16-bit address can be used to select a relocation register and the low order bits used as the displacement to be added to the value of the relocation register. Relocation registers provide all the features of a page map when the number of relocation registers is at least as large as the number of page map entries.

Systems utilizing relocation registers usually have only a small number of them (e.g., the DEC PDP-10 KA-10 processor has only two relocation registers). With a small number of relocation registers it is possible but still relatively inconvenient for a program to access a data base which is located in a number of parts of the memory since the contents of the relocation registers must be changed frequently as data items in different areas of memory are referenced.

When all data used together is located in no more areas than there are relocation registers, frequent reloading of the relocation registers can be avoided. This complicates the allocation of real memory. When new items are to be

placed in memory not only must the required amount of free memory be found, but it must be contiguous and have the required proximity to the other data areas with which it is used. Frequently it will be necessary to do a large amount of compaction of data in the memory to create the available space meeting all the conditions.

If a large number of relocation registers are provided, the capabilities and implementation are very similar to paging. Comparing a relocation register system with a page map system where the number of relocation registers is equal to the number of entries in the page map shows the relation. Each relocation register (typically implemented as a word of a fast memory array) corresponds to an entry in the page map (a word in the map memory). In both systems the word is selected on the basis of the high order bits of an address. The difference is in the way these values are used to generate the new memory address. The low order bits of the address would be <u>added</u> to a selected relocation register. The page map system would use the low order bits directly and they would be <u>concatenated</u> with the high order bits from the map entry to form the full address. These methods are equivalent if the low order bits of the value in the relocation register are always zero. The difference in the hardware implementation is the requirement for an adder and

the need for a larger relocation register memory in order to hold the low order bits.

The relocation register scheme can be described as a page mapping scheme in which pages need not be located in real memory on address boundaries which are multiples of the page size. It is questionable whether this added flexibility warrants the slight additional cost of the hardware. To simplify allocation and use of memory it is most convienent to allocate the memory in blocks of a fixed size (page) anyway.

Loading the Page Map

A mechanism must be provided for the program to place values into the page map table. There are four ways this can be accomplished.

1. The instruction set could include the map entries as specialized central processor registers. For a page map table which is integrated with the central processor, this is viable. However, the address translation is inherently unrelated to the central processor activity and can just as well be implemented separately from the central processor. For a map facility which is supplied by another vendor, independence from the central processor is important for two reasons. It simplifies the interface and it

makes the map more easily adaptable to different computer systems.

2. A second possibility is to provide a means by which the map would automatically be loaded with values from the computer memory, the values having been placed previously in fixed locations by the program. While the hardware to control this automatic loading is relatively complex, the benefits over the program directly loading the map seem small. Either alternative 3 or 4 for directly loading the map is acceptable. The choice will depend on the particular minicomputer.

3. The memory map can be interfaced to the I/O bus to allow loading in the same way I/O is performed. While the main computer facilities can be the same as I/O, the all electronic nature of the map will allow higher speed operation than normal I/O. The program will never have to wait for the device to become ready, complete operations, etc.

4. Another possibility is to have certain locations in the memory address space correspond to the map. Thus when the program stores data into these locations (which would normally be regular memory), the values will go into the fast memory for the map. For

computers which have a single bus for both I/O devices and memory, alternatives 3 and 4 are equivalent. (All I/O devices are addressed in the same way as memory.) Where there are separate I/O and memory buses, loading the map via the memory bus instead of the I/O bus avoids the need for interfacing the memory map hardware to the I/O bus as well as the memory bus.

NON-LINEAR ADDRESSING

The use of addressing mechanisms where the data in memory is not randomly addressable through a linear address space has been considered very little. The organization of most computers is based on the von Neumann model with consecutive integers used to select the bytes or words of memory. Notable exceptions have been Burroughs machines such as the 5500 and 6500 series where many of the program's operands are kept by the hardware in a stack structure so the program need not be concerned with their address in memory.

Many changes in the way in which computer hardware provides non-linear addressing to data in main memory involve the computer instruction set and central processor applications. However, there are a number of functions which can be independent of the central processor and thus

can be used with currently available processors. These can be implemented by providing a memory system which interfaces to the central computer in the standard manner but operates in modes other than just storing and fetching at locations specified by a linear address space.

Historically linear addressing came first, and thus became traditional. While providing both linear and non-linear addressing modes entails somewhat more hardware than simple linear addressing, the difference is not that great. Some very useful non-linear addressing facilities can be easily included with linear addressing by using much of the circuitry in common with the linear addressing.

Even though the memory may be organized to be addressed linearly, a relativly small amount of hardware interfacing the central processor to memory can implement non-linear address modes in the same manner that a page map translates linear address from a virtual address space to a real address space. These facilities are potential products a memory manufacturer could provide as part of its memory products. The following are some candidates for investigation and possible development.

Stacks

One or more locations in the computer's address space can be regarded as the top location of a stack which

operates in the following manner. Whenever the computer stores data at this address, the data is instead placed in memory at the location specified by another word called the top of stack pointer. The top of stack pointer is then automatically adjusted to point to the next word in memory. Another store will place the new data in the word adjoining the first and again adjust the pointer. When the program fetches a word from the top of stack location, the last word stored in the stack is returned and the pointer adjusted to the next-to-last word stored. Hence, if another fetch follows, the next-to-last word stored will be returned. Since the last data item stored is returned first and the first data stored is returned last, this function is known as either a last-in-first-out (LIFO) or first-in-last-out (FILO) stack. Additional means of assuring a program does not overfill the stack and use more than the amount of memory available or remove items when the stack is empty enhance the facility. The area in the memory where the data in the stack is kept may also be addressable through the computer's linear address space. If it is not, all accesses must be on the LIFO basis.

Although the hardware implementation can be as simple as a counter and some gating, stacks can facilitate programming in a number of ways. They are useful in many

situations where data is read in the opposite order from which it was created. Examples include the storing of return addresses and program context for nested subroutine calls. The information about an activity can be stacked while the computer services higher priority interrupt activity. Assemblers, compilers, and other processors utilize stacks to hold data while they scan ahead to see what comes next in the input text. A simple hardware stack will free the programmer from coding software to perform the function. The hardware implementation will cause the program to execute faster. In cases where software stacks are not feasible, hardware implementation will allow programs to have additional features such as recursion. The usefulness of hardware stack mechanisms has been known for years and several computers, including some minis, have such hardware incorporated into their mainframe. However, the implementation of the stack hardware, especially on minicomputers, is often not complete particularly with regard to stack limit checking.

Queues

Another function which can be implemented in the same way as the stack is the queue. A double ended queue (deque) is like a stack with separate pointers to the top and bottom of the data storage area. Hence, data items can be added or

removed from either end. More commonly, only a first-in-first-out (FIFO) facility is needed so it is only necessary to add items at one end and remove them at the other. As with the stack, checks are useful to detect attempts to remove items when the queue is empty or to store more items than can be held in the available memory. Important applications for queues include maintenance of the queue of tasks to dispatch in the operating system scheduler and the buffering of data to or from I/O devices.

Linked Lists

For many computer programs it is convenient, either because of the inherent structure of the problem's data or as a programming convenience to make extensive use of address pointers or links. The major advantage of the links is that they are easy to manipulate and change. Using multiple pointers to a single copy of a large data item eliminates having a copy of the data item everywhere it is relevant. A reference to a data item can be added or deleted by changing a pointer without moving the whole data item. The use of pointers can be extended to include lists of pointers to pointers (to other pointers for many levels). Such structures are known as lists.

Lists of one form or another are used in a wide range of circumstances from the linking of control blocks in an

operating system to many sorting algorithms. For problems where list structures are necessary, languages such as LISP, SLIP and L[6] have been developed.

The implementation of non-linear addressing facilities which are designed to operate with pointers and lists can greatly facilitate calculations with data structures so organized. Providing these facilities for a minicomputer has several advantages. It reduces the number of computer instructions needed to access an element of a list. This makes coding a program easier. It reduces the size of the resulting programs and consequent memory requirements. Also, since fewer instructions need be executed to perform a given function, the programs will run faster. The most important implication for use on minicomputers is that the amount of data a program may address can be extended. Placing responsibility for much of the handling of the address pointers as part of the computer-memory interface makes it possible to use fields larger than the normal 16-bit addresses most minicomputers can process. The computer-memory interface can handle pointers (e.g., 22-bits) which can address all of real memory. To manipulate a large pointer the central processor must use double precision arithmetic. Providing automatic facilities reduces the number of places this is necessary and makes the

use of large pointers practical.

Associative Look-up

Another non-linear addressing technique to increase the usefulness of memory is look-up or associative addressing facilities. A typical use of this capability would be for the central processor to provide to the memory some data instead of a memory address. The memory would then compare this search key with each of a number of data keys. The data returned to the processor would then be a data item which the processor had previously associated with the key. Both the data keys and data items can be kept in conventional memory by giving the computer-memory interface the ability to reinitiate memory cycles until data matching the search key is read. More elaborate schemes could allow the data keys to be kept in faster memory or in an associative memory designed to compare all data keys with the search key in parallel during one cycle.

Associative or look-up facilities are useful in applications such as sort programs, compilers, assemblers, and information retrieval systems.

Hashing

Storage organization techniques known as hashing or scatter storage can be used to speed finding data items when searching for an item with a particular key. The principle

of hashing is to take each data key and perform an operation such as modulo division which gives a result having a relatively small number of bits. The particular operation is arbitrary but very important. Operations which are even simpler than modulo division can be used. The important properties of this operation are that for a given key value it always gives the same result and that the results are fairly uniformly distributed over the possible results for typical keys. Use of the hash function on the data keys allows each to be classified into one of a number of categories. Keeping all items in the same category either together in memory or on the same linked list facilitates a search since the search key can be hashed and then only the data items of the relevant category searched.

Hashing could be facilitated by the computer-memory interconnection as a repeatable, partial scrambling of the address bits. In this mode the program could use a data item as an address which would be translated into the real memory address for the beginning of the data items or list of data items for the category. Hashing could be implmented either separately or in conjunction with an associative look-up facility. It is also useful in sort programs, compilers, assemblers, and information retrieval systems.

PROTECTION

With increased memory size the probable increase in program complexity makes memory protection an extremely desirable if not mandatory feature. It is necessary to protect the operating system areas from damage by a user program, inadvertent destruction of executable code mistaken as a data area, and in multiprogramming systems the destruction of one program's data or code by another program.

The primary protection problem for minicomputers is the lack (in most minicomputers) of dual-state (supervisor state/problem state) processors to allow a supervisor program to control the protection mechanism. To solve this problem we have developed a scheme for implementing a mapped, problem state mode and an unmapped, supervisor mode in conjunction with the page map interface between the central processor and the memory. This mechanism will allow a supervisor program to control the areas accessible by a problem program and intercept any attempts to access protected areas.

While the use of keys associated with blocks of physical memory is sufficient to specify read and write protections for blocks of memory, in some cases other techniques can be useful. When there are more programs than

available keys, the use of keys associated with physical blocks of memory becomes cumbersome since the keys in memory must frequently be changed when the processor is switched from execution of one program to another.

A page map itself is a powerful protection mechanism since the contents of the page map specify what areas of memory a program can access. Those areas of memory which are not addressable through the page map cannot be read or written by the program. The addition of a protection bit for each entry of the page map allows selective specification of read or write access for each of the pages which is accessible through the page map.

A final protection technique is the use of limit registers. A limit register can be used to specify a maximum displacement allowed from a relocation register. Limit registers can also be used to limit the bounds of list addresses, stacks and queues, or look-up areas.

I/O AND DIRECT MEMORY ACCESS

The use of I/O devices designed to fetch or store data directly from memory is affected since the I/O controllers have been designed to access only a small amount of memory. In larger computers designed for large memory address spaces an unmapped address in the real memory address space is

usually provided to the I/O controller. This will not work with a large amount of memory on minicomputers since the I/O controllers were not designed to handle a large address. One solution is to only allow I/O into a relatively small fixed area of memory (e.g., the number of locations at the bottom of memory corresponding to the original computer size). This data must then be moved by the central processor to or from the other areas of memory where it is used.

A more pleasing alternative is to have I/O done through a page map. Since it is important to be able to have I/O as independent as possible for the I/O devices and processor there should be two maps. However, much of the control circuitry can be shared. Thus some entries will be for use by the central processor and others by the I/O devices. Essentially the only additional cost for the I/O mapping is the additional fast memory elements to hold the additional map entries.

PROGRAMMING SUPPORT

A primary criterion in this analysis of the use of large memory on minicomputers is to permit existing programs to run unaltered. Additional advantages can be achieved by tailoring software to the potential of the larger memory

capacity. Although user programs can run unchanged, alterations to algorithms to make use of larger memory and non-linear addressing facilities could improve their performance. The other area of improvement is the system support software. Additional operating system type functions are potentially very useful. For minicomputers having operating systems these functions will enhance their performance. Where there is no operating system they can constitute a simple program support kernel. The additional memory will make it feasible to run more than one program at a time in many cases. To support this, context switching software to keep track of the progress of multiple programs is required. This involves memory allocation, map setup for addressing, protection setup, and handling of protection violations. While the impetus for using a large memory on a minicomputer is to retain most of the data required in main memory, the paging box hardware permits easy implementation of demand paging. In some circumstances this may be useful.

MECHANICAL AND ELECTRICAL

There are a number of mechanical and electrical details worth noting at this point. Good solutions to the problems discussed previously necessitate the use of a moderate amount of electronic circuitry. Thus to connect an existing

minicomputer with a traditionally organized memory an interface unit between them is needed. It seems reasonable that a single basic interface design could be adapted for use with a variety of minicomputers. Forethought to this problem could allow the special requirements of each computer to be met with only a specialized cable and appropriate jumpers in the interface unit.

The mechanical considerations include the interface packaging and its location in the same cabinet with the memory, minicomputer, or separately. Power cables, and signal cables to both the memory and minicomputer must be provided.

Electrical considerations include both the logic levels, cable terminations, and signal protocols. Adaptability to a variety of minicomputers requires considerable flexibility in adapating to different synchronous or asynchronous timings of the computer memory bus.

III. INTERFACE DESCRIPTION

The previous sections of this report have discussed in general the capabilities and the problems of utilizing a large amount of memory on a minicomputer. The following is a proposed implementation to demonstrate its usefulness and provide a vehicle for further exploration. All of the ideas discussed previously are not included in this design. The hardware is intended to be relatively easy to alter to allow experimentation with the various ideas.

The interface will allow the Lockheed Electronics SUE minicomputer to utilize Data Products 6361 large core store (LCS). The SUE computer is a relatively new design of moderate speed which is typical of other 16-bit single bus architecture machines (PDP-11, etc.). Although the interface will be designed specifically to operate with the SUE, with slight modification and new cables it will operate with other machines such as the PDP-11, or even with machines having separate memory and I/O buses (e.g., Varian 620 or Data General Nova).

The 6361 LCS is 1.8 microsecond core memory which can be accessed in either 32 or 64 bit wide modes. Originally designed for use with IBM 360's as bulk core storage, it includes a parity bit for each 8-bit byte and a 8-bit

protection key for each 2 kilobyte block.

Basic Operation

The basic operation of the memory interface is shown in Figure 1.

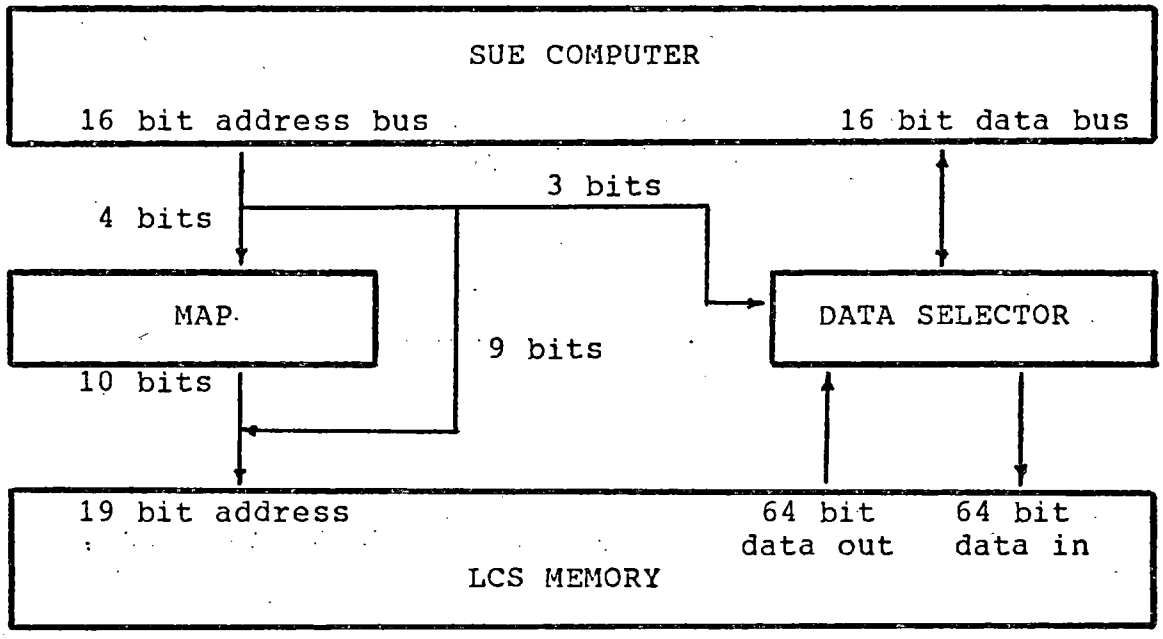

Figure 1. LCS memory mapping interface.

The address paths are shown at the left and the data paths at the right. For each memory operation the SUE processor provides a 16-bit address to the interface unit. The high order four bits are used to select one of the sixteen

entries in the page map. The 10-bit entry selected is used as the high order bits of the address given to the LCS. The low order nine bits of the 19-bit address of a doubleword given to the LCS are taken directly from the SUE's 16-bit address. The final part of the addressing logic is the use of the low order three bits of the SUE's address by the data selector. On a read operation the data selector uses these bits to select the proper bits from the LCS bus out lines for either a byte or 16-bit word fetch. These values are placed on the SUE bidirectional data bus. On a write operation the data selector places the contents of the SUE data bus on the correct LCS data in lines and asserts the proper mark lines to write into the appropriate LCS memory bytes.

Mapped/Unmapped Modes

The memory interface contains a flip-flop which indicates operation in the mapped-problem state mode or in the unmapped-supervisor mode. In the mapped mode the interface operates as described in the basic operation section. In the unmapped mode the page map is bypassed and addresses will refer to the first sixteen blocks of real memory except for the special address described in the following. One special address enables the supervisor to address the mode flip-flop and thereby set the system into

mapped mode. To allow the processor a chance to set the proper mapped address in the program counter this action will not take effect for three memory cycles. Note that a user (problem) program cannot directly affect the map mode since the special map location is not in its address space.

Loading the Map

Special addresses are recognized by the interface unit when operating in the unmapped-supervisor mode. Data stored into these locations will be placed in the various entries of the map.

Protection Violations

Each entry in the map includes, in addition to the 10-bit real page address, a bit which specifies whether or not a program may write on that page. If a program in the mapped mode attempts to write in a write-protected location, access a logical address less than 128 (decimal), or access the second page of real memory, the interface recognizes a protection violation. When a protection violation occurs the interface will initiate an interrupt and switch to unmapped mode operation. Thus read only access is granted to a page of memory by setting the appropriate map bit. "No access" through a map entry is set by pointing that entry to the second real memory page. Since real memory page 2 will most likely hold supervisor code it will not need to be

referenced in mapped mode. Since interrupt vectors are kept in low memory, the special handling of references to the first 128 locations provides a convenient means of causing a switch to the unmapped mode when an interrupt occurs. This is necessary since in a multiprogramming system the interrupt may signal completion of an I/O operation started by a different program than the one executing at the time of the interrupt. This special convention also allows programs to call the supervisor and enter unmapped mode by referencing one of the low locations.

Clock and Watchdog Timer

To enable timeslicing in a time-sharing environment a clock interrupts execution in the mapped mode fifty times a second. While many minicomputers already have such a clock, this is implemented in the memory interface to serve as a watchdog timer. If, after the memory interface unit signals a timer interupt to the central processor, the interrupt does not occur within a reasonably short period of time, then the user program has left the interrupts disabled or possibly even issued a halt instruction. If set to do so, the memory interface unit will provide automatic system recovery by resetting and restarting the computer.