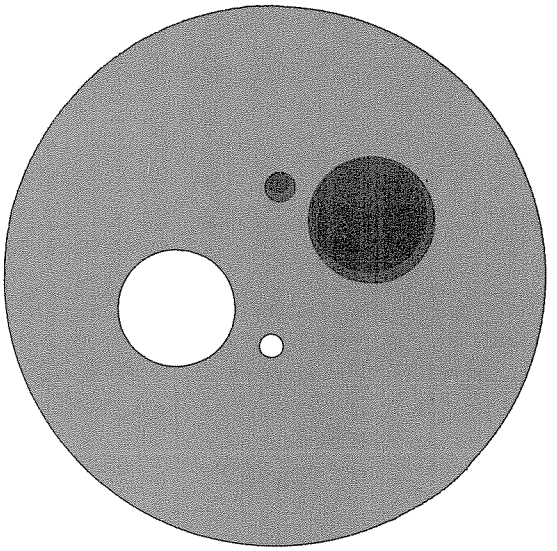
COMPUTER SCIENCES
DEPARTMENT

University of Wisconsin-
Madison



Memory Extension Techniques for Mini-Computers

by

Mary Poppendieck
and
Edouard J. Desautels

Computer Sciences Technical Report #270

March 1976

The University of Wisconsin
Computer Sciences Department
1210 West Dayton Street
Madison, Wisconsin 53706

Received: March 9, 1976

Memory Extension Techniques for Mini-Computers

by

Mary Poppendieck

and

Edouard J. Desautels

ABSTRACT

The address range of mini-computers purchased two to three years ago was typically 32K words. Many owners of these computers are realizing that this is no longer adequate for their application. Various methods of extending the memory of mini-computers are becoming available. This paper discusses and compares the various methods of addressing more memory which are becoming available for mini-computers.

There are many computer users these days who have bumped up against the end of their memories. Years ago IBM 1800, 1130, and 7094 users discovered that 32K words of addressable memory simply was not enough. Today, mini-computer users are discovering the same thing. And so are mini-computer manufacturers. Some of the newest mini's can address 128K, 256K, 512K or even 1024K (1M) bytes of main memory. The claim is that 16M byte addressing is right around the corner.

It makes sense to add more and more memory to mini-computers, because memory prices are dropping rapidly. (See figure 1.) As of January, 1976, it was possible for an end-user to obtain a 16K 16-bit word memory board for less that $1300 in unit quantity, or 0.5¢ per bit. When this is compared to the salary of a programmer trying to squeeze code into a small memory, extended memory becomes very attractive.

However, the mini-computer user contemplating memory expansion must understand the various memory expansion techniques which are available in order to evaluate their effectiveness. One computer's 1M bytes is not the same as another computer's 1M bytes. There are dramatically different ways of addressing large memories.

It is important to establish the fact that there are two kinds of limits to memory space, logical and physical. Logically, a program is limited to a certain address space by the instruction set of the machine. Any linked set of code is usually confined to a certain logical space, depending upon the address size supported by the instruction set. Most mini-computers allow a maximum of 16 bits for an address, which gives 64K addresses. However, in some computers this address is a byte address, and if the computer's word size is 16 bits, only 32K words can be addressed. If one is dealing with applications in

which floating point values are required, then the 32K of 16 bit words becomes a maximum of 16K floating point words of 32 bits. Many other mini-computers reserve one of the 16 bits as a flag, perhaps for indirect addressing, which again limits the logical address space to 32K words. If a single program is pushing itself over the boundary of the logical address space, more memory will not help unless the program is divided into segments or overlays.

One method of extending mini-computer memory is to extend the logical address space by changing the classical instruction format and allowing more than 16 bits for an address. Currently the Interdata 7/32 and 8/32 instruction sets allow a 24 bit byte address and the Scientific Engineering Laboratories SEL32 instructions allow 19 bits for an address. These computers are basically 32 bit computers at the top of the "mini" classification.

The physical limits of the memory space are determined by the number of wires carrying an address to the memory, as well as the granule size which is selected by the address. For example, in a PDP 11/45, eighteen wires carry the address to memory, allowing 256K addresses. But since each address is a byte address and there are two bytes per word, only 128K words may be physically addressed, with the top 4K reserved for the peripheral page. The logical address space of a PDP 11 is 32K words, with 4K of that dedicated to the peripheral page.

If memory extension beyond the logical address space is implemented, then the computer can no longer simply use the logical address to select a physical address; some translation must be applied to the logical address to convert it to a physical address. This translation consists of combining the logical address, or some portion of that address, to a base address which is obtained in some specific way. The method of selecting the base address and combining it with the logical address can be used to classify physical memory extension

techniques. There are three basic methods of implementing the translation of a logical address into a physical address - memory mapping bank switching, and the use of base registers.

Memory mapping involves dividing the logical address into two parts - a descriptor, which is a set of the most significant bits, and a displacement, which is the remaining set of bits of the address. A function is performed on the descriptor, and the result of this function is a physical address which may be called a page address. The displacement is added to the page address to give a physical address. Generally, pages in memory do no overlap, and the number of bits in the displacement determine the granularity, or page size, of the memory. Typical page sizes vary between 256 and 1024 words. In most cases the significant part of the page address is effectively concatenated with the displacement. (see figure 2a.) In some implementations (e.g. PDP 11/45 and 11/70) the page size is smaller than the displacement field, so that the displacement field must be subdivided into a second descriptor and a second displacement. The second descriptor is added to the beginning page address to obtain the true page address, which is added to (or concatenated with) the second displacement to obtain the physical address. This technique gives the PDP 11/45 and 11/70 a page size between 32 words and 4K words. But a limited number of mapping registers makes it necessary to use 4K pages in order to gain sufficient address space. (see figure 2b.)

Bank switching involves concatenating a bank indicator, which is separate from the logical address, with the entire logical address to obtain a physical address. In some machines, two separate bank indicators are used for instruction and data memory references. In any case, since the entire logical memory address determines the displacement within a page, the memory granularity is quite large. (see figure 3a).

Using base registers involves adding the logical address to an address found in a base register. (Fig. 3b) Again, there may be two registers, one for instructions and one for data references. With base registers the granularity of memory is not usually a consideration, since the base register may generally contain any address. However, some base register schemes tend to limit the logical address space by limiting the displacement which is added to the base register. In this case, the logical address space is expanded by changing the base registers. The IBM 360's and 370's use base register to determine all addresses, but there do not seems to be any mini-computers which use base registers to determine the addresses for every memory reference. However, many mini-computers add a signed displacement field to the program counter to determine some operand addresses. By using the program counter as a base register in this manner, programs are not restricted to particular segments of physical memory, although they are still somewhat confined by the size of the displacement. The Data General Nova has an 8 bit displacement; the Interdata 7/32 allows 15 bits.

Bank switching had its roots in the early 1960's, on a few IBM 7094's and DEC's PDP5. About 1962, MIT added an extra 32K to the 7094II used for their Compatible Time Sharing System (CTSS). A single bit register was used to select which 32K bank of memory was in use. In addition to expanding the memory, this system prevented programs in one half of memory from interfering with programs in the other half. About the same time, the PDP 5, predecessor of the PDP 8, was able to address 32K of memory with only 12 bits for the address. This was done with a set of two three-bit registers which could be read and written with special instructions. These registers selected one of eight banks of 4096 words. One register selected the instruction bank and one selected the operand (data) bank. This technique is used by the PDP-8 today.

Computer Automation Naked Mini and Megabyter LSI-2 Computers can address 512K and 1M bytes respectively, using a bank switching option. The logical address space is only 32K. General Automation's SPC-16 and the Microdata 1600 also have a bank switching option.

There are many variations of bank switching available. The SPC-16 divides its 32K address space into eight 4K banks. The lower four banks are always the same, but the upper four banks may be switched to any of eight additional 4K banks which are selected by an I/O command. This provides 64K of physical memory, with 32K selected at any one time. In one system at the University of Wisconsin, a Honeywell DDP-124 has a flip flop hanging on the end of an IO channel acting as a bank switch. The lower 8K of each bank is physically the same, and in this 8K resides a monitor which communicates with the 24K left in either bank and does all the bank switching. With this system, a total of 16K of address space is lost to the monitor.

Harris Slash 5 computer can directly address 32K 24 bit words with most instructions, but they support 64K of memory. In this case the top bit of the address comes from the location counter. A special jump instruction has a 16 bit address field for access to the other bank, or else indirect addressing can be used.

The HP2100 instructions can directly address only locations in the current 2K memory bank. The upper bits of the address are obtained from the location counter. Indirect addressing expands the address space to 32K.

The top bits of the address can also be obtained from the status register. For instance if there is both a system and a user mode which is indicated in the status register, the system/user mode bit can be used to determine the top bit of the address. In the modified PDP 11's used by the Carnegie-Mellon multi-mini processor, a two bit bank indicator is contained in the program status word. This gives the advantage of saving the bank indicator automatically whenever the status word is saved.

Three distinct disadvantages of bank switching should be mentioned. First of all, communication between banks may be very difficult, if not impossible. This may be an advantage, since memory protection increases as cross-talk becomes more difficult. However, if a supervisor is to run programs in all banks, some method of communication is usually needed. Usually communication between banks must be done via registers or disk. In the SPC 16 and DDP-124 systems described above, communication is simplified because the lower quarter or half of memory remains unchanged.

The second serious drawback of bank switching is the granule size. The banks are typically large and the logical space of any program must usually be confined to one bank. This is a rather rigid restriction which would generally result in a lot of wasted memory space. Moreover, memory protection within each bank may not be available. However, in some systems, particularly dedicated ones, there are often only two users of approximately equal sizes, and two banks. Or it may be feasible to separate the supervisor and the user into separate banks, especially if the supervisor is large or can be assigned enough tasks to make it into a second user.

The third serious drawback of bank switching is that banks must be switched. First of all, what is going to happen to the location counter when the banks are switched? This is not a problem when the program counter is also the bank indicator, but is definitely a problem otherwise. If there isn't some sort of memory common to both banks in which the bank switching can occur, then there must be some way of changing the location counter at the same time as switching the banks. Secondly, how is the switching mechanism governed? Switching from one bank to another usually requires positive action on the part of the user or supervisor. Protecting the program status word from the user (if it contains the bank indicator) or not allowing the user access to the bank register is probably necessary. Some method of saving the current bank in case of an interrupt and restoring it after the interrupt is needed. And of course, some means of deciding when to switch from one bank to another must be available.

Like bank switching, memory mapping was first used in the early sixties. The Atlas computer developed at Manchester University used a memory mapping technique which was eventually adapted for use with associative memories and became the prototype of several memory mapping systems. Memory was divided into 32 "pages" of 512 words, and there were 32 Page Address Registers, one for each physical page. Each register contained the logical page address of the program which resided in the physical page corresponding to that register. When a memory reference was made, the registers were searched, using the logical page number as a key. If a match was found then the physical page number of the memory reference was simply the register number of the match. If no match was found, the missing logical page was brought in off a drum. (See figure 4a)

A sequential search of several registers for every memory reference can be slow, but content addressable or associative memories remove this drawback. A key and a value are stored in one slot of a small associative memory (on the order of eight registers are usually used). The key is the description from the logical address and the value is the physical page. When a descriptor is sent to the associative memory, all keys are simultaneously searched. If a match is found, the value - a physical page number - is the result of the compare. If no match is found, the associative registers are updated from a table in memory. (See figure 4b) This type of map was first used in the IBM 360/67 (1966) and in Scientific Control Corporation 4700 mini-computer (1968). Today it is found in the Prime Computer Models 300 and 400.

Another type of memory map was used on the Berkeley Timesharing system SDS 930, about 1965. In this system there were 8 six-bit registers which could be selected by the top 3 bits of a logical address, used as an index. The contents of the selected register contained a six bit physical memory page number. (See figure 5a) A memory map of this type is available from Fabri-tek for the PDP-11/05, 11/10, 11/15, and 11/20. Note that the top three bits of each address will always

select a register - there is no possibility of a no-match. Therefore an entire program must be in memory before execution can begin, unless there is some way of telling the system whether or not each register contains a valid address.

If protection bits of some sort are added to the registers the requirement that an entire program must be in memory before execution is removed from the index-type mapping scheme. Protection bits are usually used to restrict users from reading, writing into, or executing certain pages of physical memory. However, since reference to a completely protected page will cause a trap of some sort, the protection bits can be used to indicate that a page is not in memory and must be fetched from mass storage. Protection bits can be used during a program's execution to determine whether or not a page has been changed and thus whether or not the page must be saved before it is overlaid.

In contrast to associative mapping registers, registers which are addressed by the top bits of a logical address used as an index are relatively inexpensive. With an associative map, the various maps are usually stored in main memory and when a memory reference creates a "no-find" condition, the associative registers are updated from main-memory. With an index-type map, the map or maps are stored in the mapping hardware registers. Several sets of registers may be available. The HP21MX has two maps - a system and a user map. The PDP 11/45 and 11/70 have three maps - kernel, supervisor, and user. The Modcomp IV has four maps - one for system, two for large users, and one which may be subdivided among four small users. The Data General Eclipse has three user maps and the supervisor is not mapped. The Varian 75 provides sixteen maps in 1024 registers. The Harris Slash 4 map provides 1024 registers for any number of users, since the beginning of the active map is selected by a 10 bit base register.

More hardware maps allow faster switching between users. Some computers, like the PDP 11/45, 11/70 and the Modcomp IV, divide the maps up into an instruction and a data map. If the data and instructions can be completely separated, the address space available to any one program could conceivably be doubled, since the same addresses could be used for instructions and data, and then the map could separate the two. (See figure 6)

However, the limitations on logical address space in any mapped system is the most serious drawback of any mapping scheme. No matter how large the physical addresses can become, the logical address space remains limited by the computer's instruction set. The only way around a limited logical address space is program segmentation and overlays; more physical memory, if it is mapped, won't solve the problem of a program which has outgrown its logical address space.

A very important drawback of memory mapping is I/O handling. In general, I/O does not occur through the memory map. Even if most I/O could go through the map, DMA (direct memory access) channels from a disk, say, must access large sections of data quickly, with no time for a "no match" condition to occur. Moreover, if one user asks for I/O, control may very well switch to another user, so that the user map would not correspond to the I/O map. In most systems, therefore, I/O is handled using physical memory addresses. If block I/O is going to physical addresses, but the I/O buffer is not in consecutive locations in physical memory (as can easily happen with mapping) it is probably necessary for a supervisor program to do all I/O through its own buffers and copy the buffers from or to the user buffer. Whereas this might be reasonable in a timesharing system, it can be a severe time restriction in a dedicated system, especially one with the large data buffers that are common in real time monitor systems.

Some computers have some of the I/O going through a separate map. The Eclipse has one data channel map, the HP21MX has two such maps. The PDP 11/70 has a Unibus map. However, to be useful for large data buffers, these maps must apply to DMA access. The high speed I/O channel on the PDP 11/70 is not mapped, thus high speed transfers of data across block boundaries cannot be easily handled.

The question must also be asked - is the supervisor mapped? If so, just how is I/O to be handled? Often the supervisor is not mapped, to increase speed (no time lost to mapping) and to ensure that block I/O has consecutive memory locations. If the supervisor is mapped but I/O is handled with physical addresses, then the question is block I/O across page boundaries must be dealt with in some way. Probably the map in the supervisor would have to correspond to consecutive physical addresses in this case.

A third problem with memory mapping is a slight decrease in speed. The PDP 11/45 map adds 90 ns to each memory reference; the Varian 75 map adds 30 to 40 ns. Manufacturers claim that the HP21MX and PDP 11/70 maps do not add to the memory reference speed. This may be true for today's memories, but probably not for the higher speed memories of tomorrow. However, there is more to be considered. Besides the actual hardware construction of each address, the mapping unit must be set up with each user's map. The maps must be switched as new users are enabled. Associative registers require an extra memory reference with each "no match" condition. Unmapped I/O which must cross page boundaries can dramatically decrease "high speed" I/O time. And finally, if a segment of a program must be fetched from mass storage, a large swapping time will be added.

Memory mapping usually provides a supervisor and user mode, as well as block level memory protection and traps on certain instructions attempted while in user mode (such as halts and I/O instructions). It allows for multiple users of various sizes. However, it requires a layer of software and a layer of hardware to be added to the existing system. Both the software and the hardware must be of a rather

sophisticated design and they therefore carry a relatively high initial cost. Moreover, this software and hardware may degrade the running time of existing programs to levels which are unacceptable, especially in a real time environment.

In a dedicated system with only a few users, or a single user and a supervisor, bank switching may be the more economical and simpler technique for extending memory. Figure 7 gives an overall picture of the tradeoffs involved in using bank switching vs. memory mapping for such a system.

One possibility, being implemented at the University of Wisconsin on a Scientific Control Corporation 4700 with a 32K logical address space, is a hybrid using some features of memory mapping in a bank switching environment. In this implementation, real time considerations ruled out the use of a fully mapped system, although mapping hardware was available. It was estimated that the associative map would add about 10% to execution time. Furthermore, the transfer of large data buffers (usually 2K) through DMA units was necessary, and these could not be copied by the supervisor without an intolerable degradation in speed. Thus it would be necessary to map all user pages consecutively, if mapping were to be used. Instead, the system is divided into two 32K banks, with one bank dedicated to the supervisor and one to the user. A large and independent data handling routine became part of the supervisor, giving two approximately equal size programs. The mode bit in the status register (system or user) determines which bank is in use. Bank switching is accomplished by a system call instruction (switch to one of 64 system routines) and system return (return to caller) instruction normally used by the mapping hardware. These instructions simultaneously switch the system/user bit in the status register and save or restore all registers, including the status register. Communication between supervisor and user can take place because the supervisor can place itself in an "indirect user" mode. In this mode, the last memory reference of an indirect address sequence is done in user mode, while all other memory references are system mode. This allows the supervisor

to read or write in the user bank under special circumstances. Thus in this implementation, a few features of a memory map system are being used to overcome drawbacks of bank switching, resulting in a very efficient bank switching system.

REFERENCES

(1) Bell, C. G.; and Newell, A. Computing Structures: Readings and Examples, McGraw-Hill, New York, 1971.

(2) Freeman, P. Software Systems Principles - A Survey, Science Research Associates, Chicago, 1975.

(3) Hobbs, L. C.; and McLaughlin, R. A. "Minicomputer Survey," Datamation 20, 7 (July, 1974), pp. 50-61.

(4) Hollingworth, D. Minicomputers: A Review of Current Technology, Systems, and Applications, Rand Corporation, Santa Monica, California, 1973.

(5) Kilburn, T.; Edwards, D. B. G.; Lanigan, M. J.; Sumner, F. H. "One Level Storage System," IRE Trans., EC-11, vol. 2 (April, 1962), pp. 223-235.

(6) Lampson, B. W.; Lichtenberger, W. W.; Pirtle, M. W. "A User Machine in a Time-sharing System," Proc. IEEE, 54, vol. 12 (December, 1966), pp. 1766-1774.

(7) Michaels, John "The Mega-Mini Succeeds the Model T", Datamation 20, 2 (February, 1974), pp. 71-74.

(8) Reagan, Fonnie H., Editor Datapro Reports on Minicomputers, Datapro Research Corporation, Delran, N. J., 1975.

(9) Watson, R. W. Timesharing System Design Concepts, McGraw-Hill, New York, 1970.

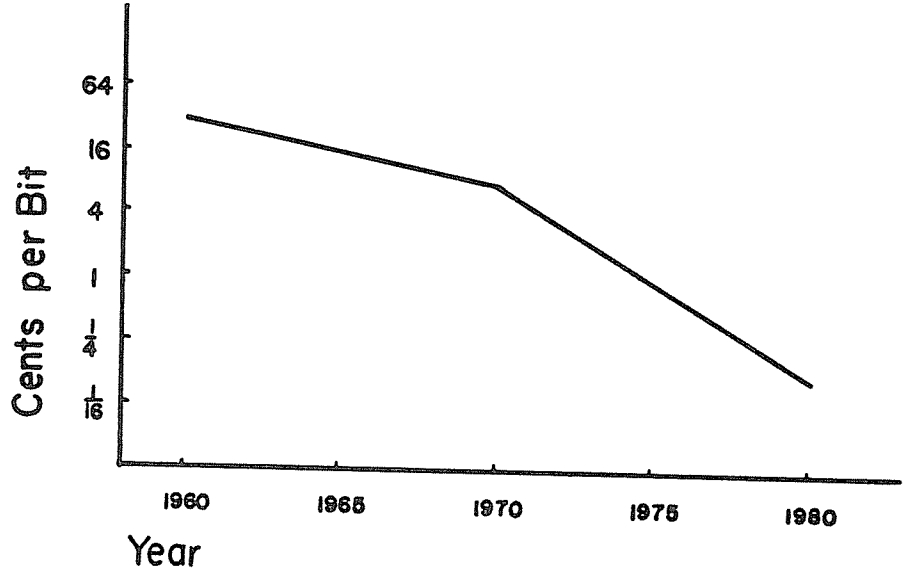
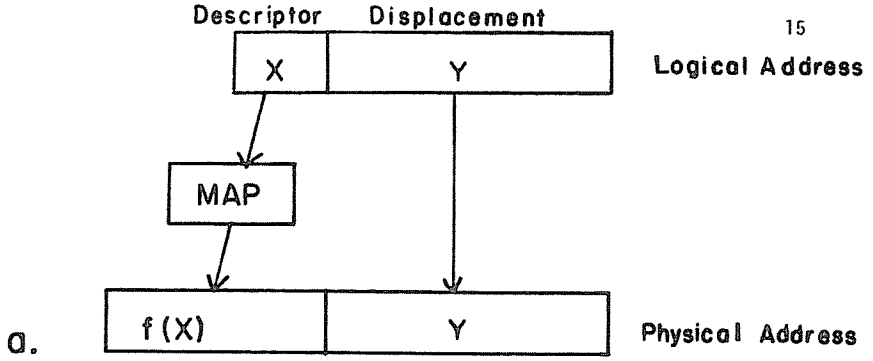
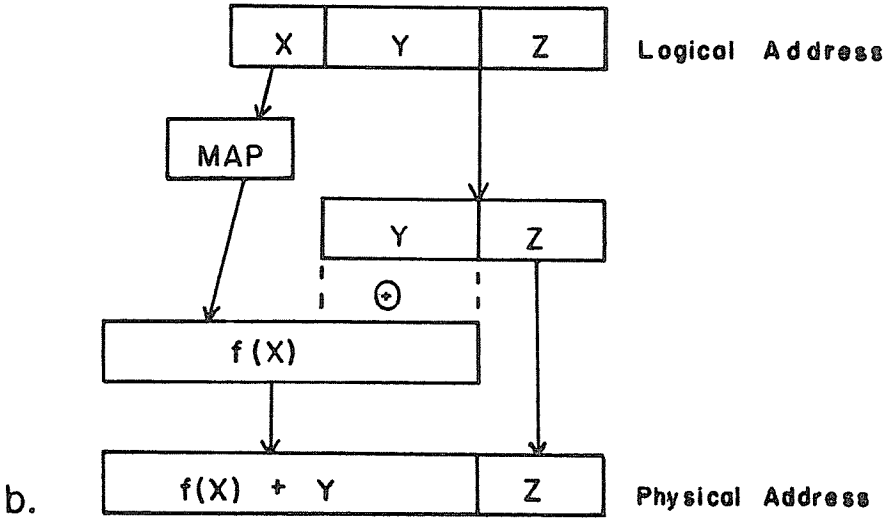


FIGURE 1 - Cost of Memory

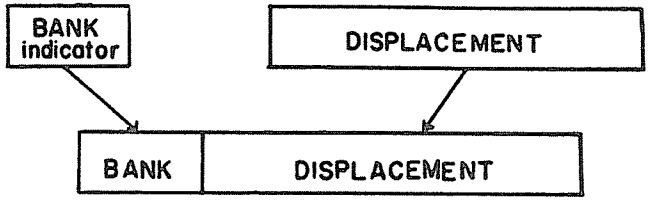


a.

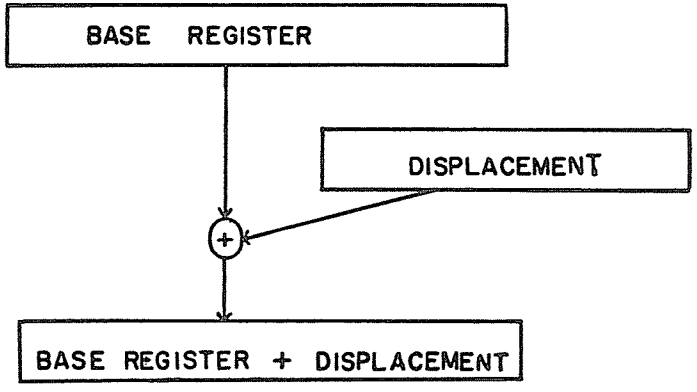


b.

FIGURE 2 - Two Mapping Implementations

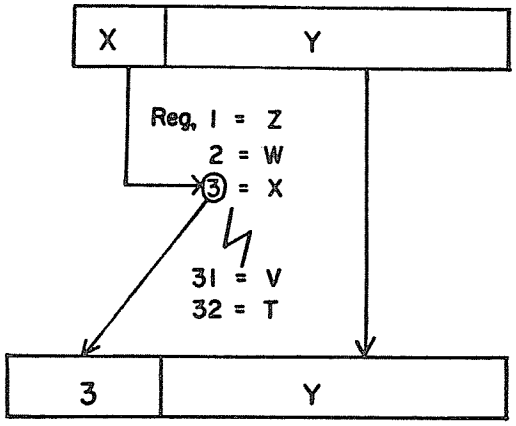


a. BANK SWITCHING

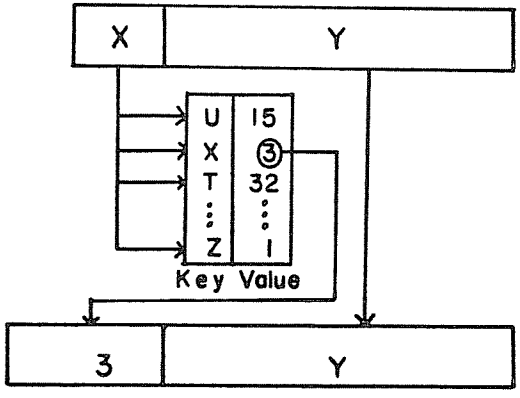


b. BASE REGISTER

FIGURE 3

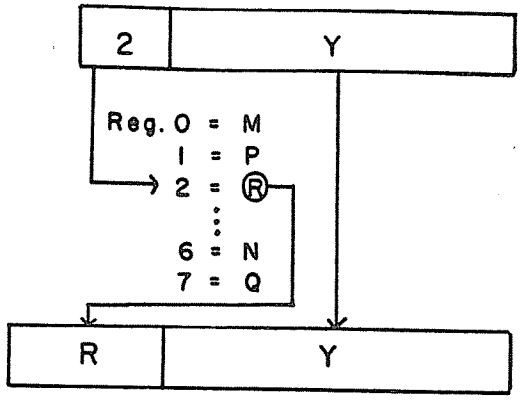


a. ATLAS MAP

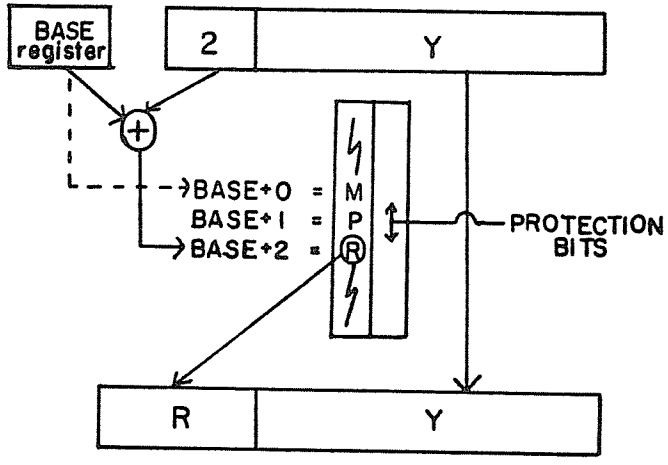


b. ASSOCIATIVE MAP

FIGURE 4



a. SDS 930 MAP



b. GENERALIZED INDEXING MAP

FIGURE 5

Manufacturer	Model	Basic Unit	Directly Addressable Address Space	Indirectly Addressable Address Space	Maximum Physical Memory	Type of Address Translation	Page Size	Number of CPU maps /System	Number of CPU maps /User	Number of I/O Maps
Computer Automation	Naked Mini	8 or 16 bits	32Kw	32Kw	256Kw	Bank Switch		--	--	--
Computer Automation	Megabyter	8 or 16 bits	32Kw	32Kw	512Kw	Bank Switch		--	--	--
Data General	Nova 3/12	8 or 16 bits	1Kw	32Kw	128Kw	Index Map	1Kw	0	2	2
Data General	Eclipse	8, 16, or 32 bits	1Kw	32Kw	128Kw	Index Map	1Kw	0	2	1
Digital Computer Controls	D-616	16 bits	1Kw	32Kw	1Mw	Index Map		--	--	--
Digital Equipment	PDP-8	12 bits	256w	4Kw	32Kw	Bank Switch	4Kw	--	--	--
Digital Equipment	PDP-11/45	8 or 16 bits	32Kw	32Kw	124Kw	Index Map	32w - 4Kw	2*	2*	0
Digital Equipment	PDP-11/70	8 or 16 bits	32Kw	32Kw	2Mw	Index Map	32w - 4Kw	2*	2*	1†
General Automation	Solution 16/440	8 or 16 bits	64Kw	64Kw	1Mw	Index Map		1	1	2
Harris	Series 200	24 bits	32Kw	192Kw	256Kw	Index Map	1Kw	Variable		0
Hewlett-Packard	21MX	8, 16, or 32 bits	2kw	32Kw	1Mw	Index Map	1Kw	1	1	2
Inderdata	8/32	8, 16, or 32 bits	1Mb	--	1Mb	not needed	--	--	--	--
Modcomp Computer Systems	Modcomp IV	16 or 32 bits	64Kw	64Kw	256Kw	Index Map	256w	1	3-8	0
Prime Computer	Prime 400	16 bits	64Kw	64Kw	8Mb	Asso. Map				
Systems Engineering Laboratories	SEL 32	32 bits	1Mb	1Mb	1Mb	not needed	--	--	--	--
Varian	V75	16 bits	32Kw	32Kw	256Kw	Index Map	512w	1	15	0

FIGURE 6 - Memory Expansion in Representative Mini-Computers
Information from Datapro Reports on Minicomputers
and Vendor-Supplied Literature.

Footnotes:

*One instruction and one data map

†for Unibus I/O only

K = 1024
M = 1024K
w = word
b = byte

Memory Mapping				Bank Switching			
Software	IO	Hardware		Software	IO	Hardware	
2	3	2		3	3	3	Low Implementation Cost
2	1	2-3		4	4	4	High Speed
3	2	3		3-4	4	4	Minimum Memory Used for Implementation
3-4	2	3-4		1	3	1	Minimum Memory lost to Fragmentation
4	4	3-4		1	1	1	Large Number of Users in Memory
3	2	4		2	4	1	Transparency to User
3	2	3		1-3	3	1-3	Communication between Supervisor & User

4 = Very Good
3 =
2 =
1 = Very Poor

Figure 7 - Bank switching vs. memory mapping for a dedicated mini-computer. Each row should be weighted for the importance to the application. Grids with variable scores are highly dependent on the implementation.