Spring 1982

Study of Virtual Memory

Shridhar S. Dixit
University of Central Florida

STUDY OF VIRTUAL MEMORY

BY

SHRIDHAR S. DIXIT
B.S., V. J. Technical Institute, 1978

RESEARCH REPORT

Submitted in partial fulfillment of the requirements
for the Degree of Master of Science
in the Graduate Studies Program of the College of Engineering
University of Central Florida
Orlando, Florida

Spring Term
1982

ABSTRACT

This research report gives a general description of virtual memory systems. The mechanisms and policies and their effect on the operation and efficiency of virtual memory are explained.

A virtual memory using a real time virtual address decoder, to decode 32 bits of virtual address for the secondary memory to obtain the primary address location is discussed. The decoder is developed with the use of associative or content-addressable memories.

Replacement algorithms, used for selecting the pages of the main memory to be replaced, are described. The hardware implementation of the least recently used and least often used replacement policies using associative memories is presented.

TABLE OF CONTENTS

GLOSSARY

Address mapping: Calculating the real physical memory address of address of operands or data from the logical address.

Associative memory: This is a type of random access memory in which any stored word can be accessed directly by using all or part of that word as a key.

Auxiliary memory: This is a much larger memory and also much slower in comparison with the main memory. It is used to store the system programs and large data files not required continually by the CPU. Information in the auxiliary memory is usually indirectly addressed by the CPU via the main memory.

Block: Unit consisting of a group of words.

Cache: A high speed memory interposed between main memory and the CPU to improve the effective memory transfer rates and accordingly raise processor speeds.

CPU or central processing unit: It is defined as a general purpose instruction set processor with overall responsibility of program interpretation and execution.

Demand paging or demand swapping: Transfer of information only when it is not present in the main memory and a request is generated for such a transfer.

Dirty regions: Regions of main memory that have been modified since being loaded from the secondary.

Dynamic allocation: The ability to allocate storage and also alter (in case of multiprogramming) it during execution.

Fragmentation: Problems arising in either paged or nonpaged systems causing loss of useful storage space.

Hit ratio: The ratio of address references to the main memory to the total address reference made.

Logical address: The address used by a programmer in a program.

Main memory: It is a relatively fast memory used for program and data storage during computer operation. The locations in this memory are directly addressable by the CPU.

Operating system: Master control program exercising overall control, supervising allocation of system resources and scheduling operations and also preventing interference between different programs.

Page: Fixed size region of the memory.

Physical address: Address used by memory or those used to address actual physical locations.

Replacement algorithms: Virtual memory policies which decide which data must be expelled from the main memory and where the new data should be placed.

Segment: Variable size blocks of contiguous words in the memory.

Static allocation: Organizing the physical address space of the program so that it remains fixed during program execution.

Swapping: Exchanging information between mass storage and main memory.

CHAPTER I

INTRODUCTION

Virtual memory describes an hierarchical storage of at least two levels, which is managed by an operating system to appear to the user like a single large directly addressable main memory. A simple block diagram representing such an hierarchy is shown in Figure 1.

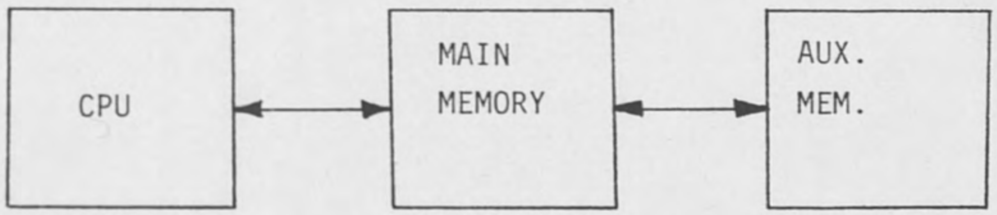

Figure 1. Memory hierarchy

Most virtual memory systems utilize a two level hierarchy shown in Figure 1, comprised of a high speed main memory of relatively smaller capacity and a higher level secondary memory which is much larger but has larger access time.

A virtual memory frees users from the storage allocation problem. It permits efficient sharing of memory among different users and achieves high access rates and low cost per bit possible with memory hierarchy. It gives the programmer an illusion that he has a very large main memory at his disposal, even though the computer actually has relatively small main memory.

To reference data in the main memory the programmer has to use certain addresses (i.e., pointers to memory locations where the data is resident). In the operation of a virtual memory two such addresses need be differentiated. They are logical or virtual address and physical or memory address. An address used by the programmer is called the virtual address and the set of such addresses is called the logical address space (N), the addresses used by memory or those used to address physical storage locations are called physical addresses. The set of such locations is called memory space (M). The translation of the logical addresses to corresponding physical addresses is required. This necessitates the use of an address translation function to perform this work. Address translation is represented at each instant of time by a mapping $f: N \rightarrow M$, such that $f(x)$ gives the physical address of the virtual address x if x has been allocated space in M, otherwise $f(x)$ is undefined. The physical interpretation of f is that a device called an address translation mechanism is interposed between the processor and memory as shown in Figure 2 (1).

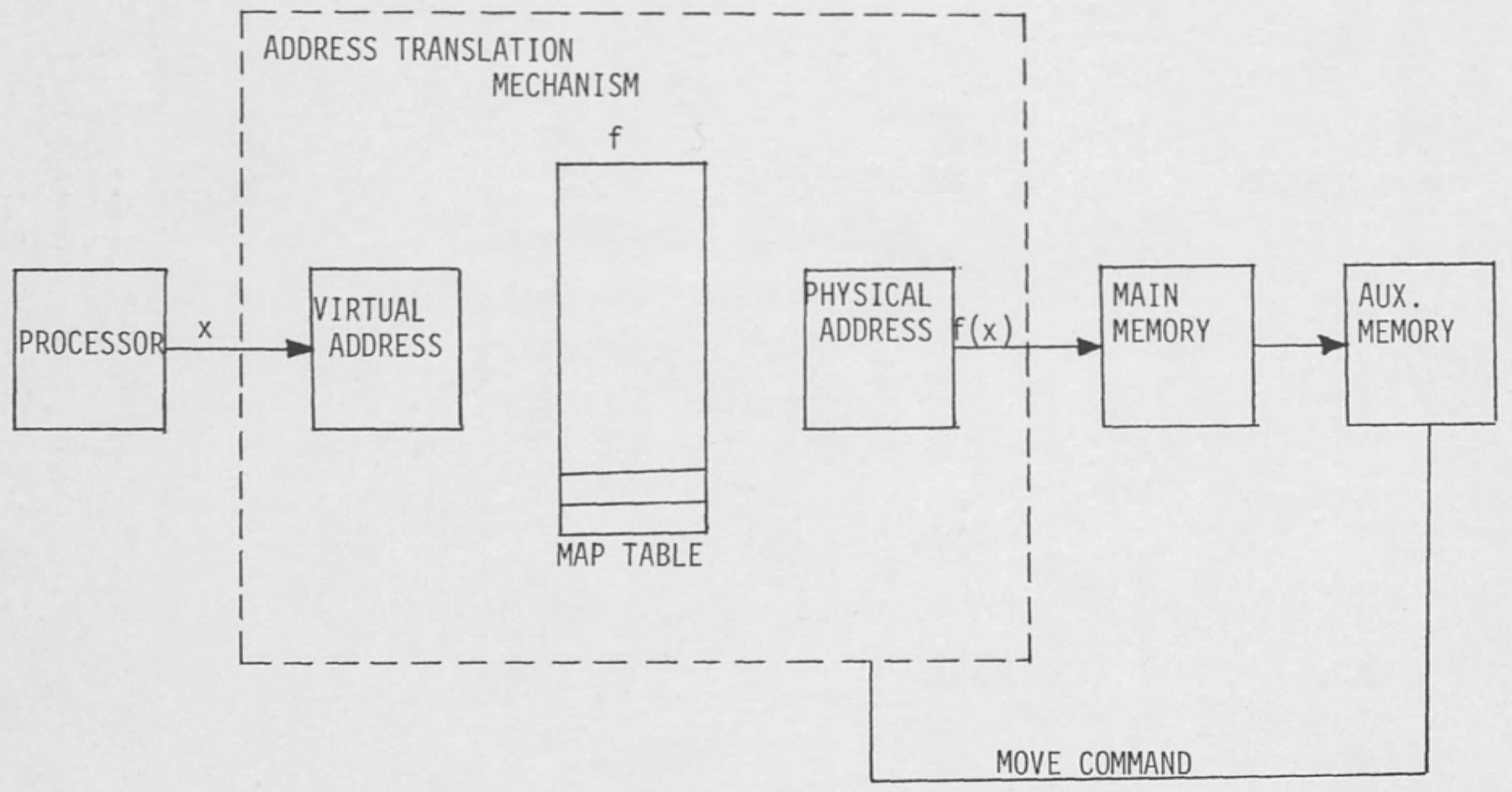

Figure 2. Implementation of address map

Whenever the processor generates a reference to an x for which f(x) is defined, the mechanism presents address f(x) to the memory and allows the reference to proceed. If f(x) is undefined the mechanism generates an fault. Now the piece of information has to be located in the auxiliary memory and transferred to the main memory and the map table f is updated to show the new changes. In getting the information, some other information in the main memory may have to be moved out to make room.

For an efficient operation of the virtual memory, it is desired that the data required by the CPU be present in the main memory or if it is resident in the auxiliary memory it should be transferred to the main memory fast so the CPU may not be idle, because the CPU can reference data only if it resides in the main memory. Since the capacity of the main memory is restricted by its size, not all information needed will be present in it.

The storage allocation policy determines, at each moment of time, how information should be allocated between the main memory and the secondary memory. Two approaches are associated with the storage allocation policy. One called the static approach assumes that the availability of memory resources and sequences of references to the information are both known. The dynamic approach assumes that availability of memory resources cannot be predicted prior to the time the program is run and sequences of references to the information can be determined only during programs execution.

The static approach was superseeded by the dynamic approach because of the unpredictable availability of memory and unpredictability of the programs and also the ability to relocate the program in any available part of the available main memory.

It is desired that only data with a high probability of being referenced next be resident in the main memory because of its limited capacity. To acheive this, replacement algorithms exist that decide which information needs to exist in the main memory and which should be replaced.

This report gives the general description of virtual memory and its characteristics. The Least-Recently-Used replacement algorithm and its hardware implementation is also discussed along with the real time virtual address decoder used in the virtual memory system implemented via use of associative memories.

CHAPTER II

VIRTUAL MEMORY CHARACTERISTICS

All processors generate a string of logical memory addresses. These addresses are always distributed among different levels of hierarchy in some fashion. But the data for a particular referenced address need always exist in the primary memory if the CPU is to utilize it. If referenced information is not found in the main memory, a process called swapping takes place, transferring information from auxiliary memory to main memory which may idle the CPU, an undesirable factor. Some amount of estimation is required so that the desired information can be transferred to the main memory before it is referenced.

Such estimations are based upon certain characteristics of programs called locality of reference. By this property, the addresses generated by any program tend to cluster or be confined to small regions in the logical address space. Figure 3 shows that even though the reference frequency to data may vary, the references tend to cluster in small groups.

This also explains that references to physical locations do not occur in a random fashion (2). The measure of locality defined as the working set, is the smallest set of virtual addresses that may be assigned to memory locations to have the

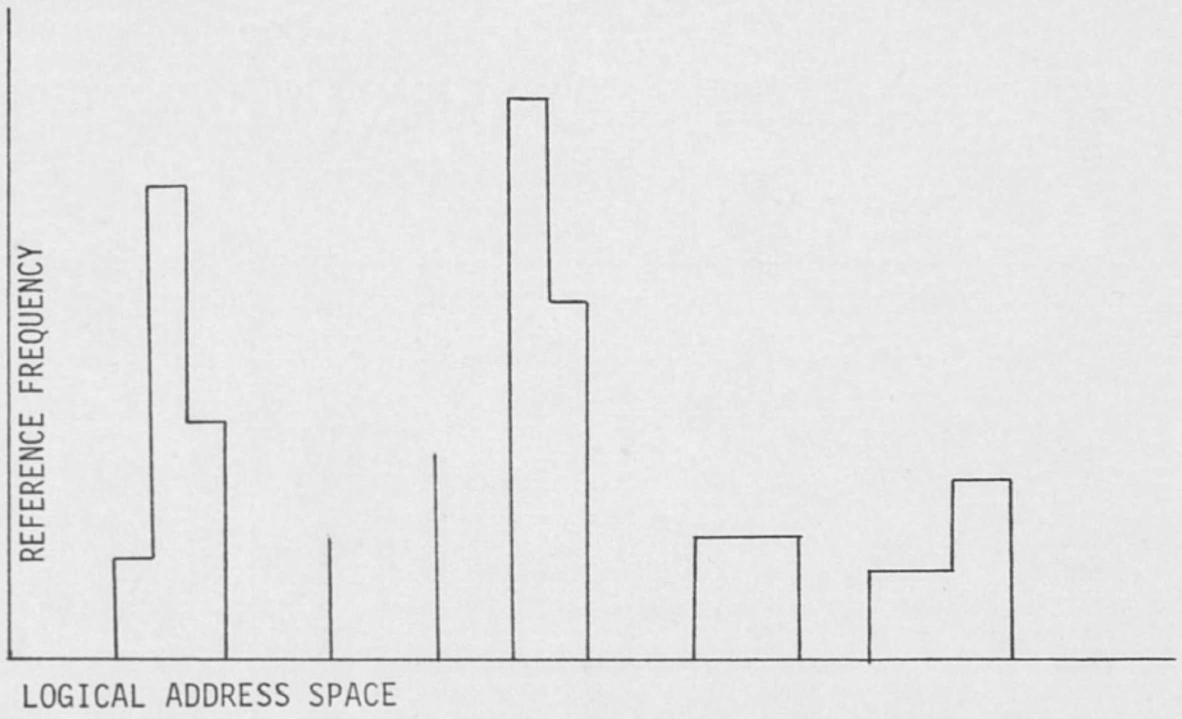

Figure 3. Locality of reference property

program operating efficiently. Locality of reference gives rise to a very important concept in implementation of virtual memory, namely, paging.

Implementation of Virtual Memory

Virtual memory would be very impractical if references made to any particular piece of information fetches only that piece. A large amount of swapping will be required if the data requested is not present in the main memory. The locality of reference gives an important clue that, if information were arranged in blocks, a reference to some address would not just fetch that data but would also fetch contiguous data with a high probability of being called soon, for each block contains a set of contiguous addresses in the address space. The locality of reference property gives rise to the concepts of paging, segmentation and the combination of these.

Segmentation is a technique which allocates main memory by segments. A segment is a variable length block of contiguous words. A word in the segment can be referenced by specifying the segment address and relative address or displacement within the segment. When a reference is made to some information and the particular segment containing it does not happen to reside in the main memory, the whole segment is swapped from the auxiliary memory. Physical addresses assigned to the segments are maintained in a segment table, which can be another segment itself. The segment table also contains a presence bit to show whether the segment is present in the main memory. It may have the block size information and also information as to which segment is to be replaced.

Paging is a technique which allocates main memory by pages. Page is a fixed length block and is assigned to the physical memory page location. Again information is referenced by specifying the logical page address and the word address within the page.

The swapping process is greatly simplified in the paging system as any page can be assigned to any memory location, unlike segmentation, where the memory needs to be scanned to fit the segment into a properly sized location. Segmentation and paging may have a problem of fragmentation. Fragmentation happens since segments have arbitrary size, resulting in a proliferation of holes or empty spaces of various sizes in the main memory giving it an appearance of being checkerboarded. Such fragmentation is called external fragmentation. No external fragmentation occurs in paging because of the uniformity of the page size and because any page can reside in any available page location. But there is every likelyhood of some pages not being filled to capacity, resulting in internal fragmentation. Another type of fragmentation which can occur in either paged or nonpaged systems is called table fragmentation. Table fragmentation occurs because certain systems tables or regions of memory must remain fixed, for instance the virtual address mapping tables would need to remain fixed for fast address translation (1).

The goal for the design of virtual memory system is to achieve performance close to the fastest memory but cost per bit should tend towards that of the auxiliary memory (Figure 1) or the device which is at the higher level in the hierarchy. Design factors such as access time of each level, storage capacity, size of information blocks to be transferred and allocation algorithms react with each other in a very complex manner and are very difficult to analyse. A very important quantity, which gives the performance of virtual memory, is the hit ratio. Defined as the probability that a logical address generated by the CPU refers to the information stored in the main memory. The hit ratio is a useful concept in determining size of pages. Hit ratio can be represented as:

$$H = \frac{\text{address references to the main memory}}{\text{total address references made}}$$

Miss ratio is defined as 1-H. A graph showing the relationship between primary storage capacity and miss ratio for a few page sizes is shown in Figure 4. The other quantities which play an important role in evaluating virtual memory systems are the cost per bit and access time.

The cost per bit depends on the cost of technologies used for the different levels and also their storage capacities. In general the cost per bit of memory is given by:

$$C = \frac{C_1 S_1 + C_2 S_2}{S_1 + S_2}$$

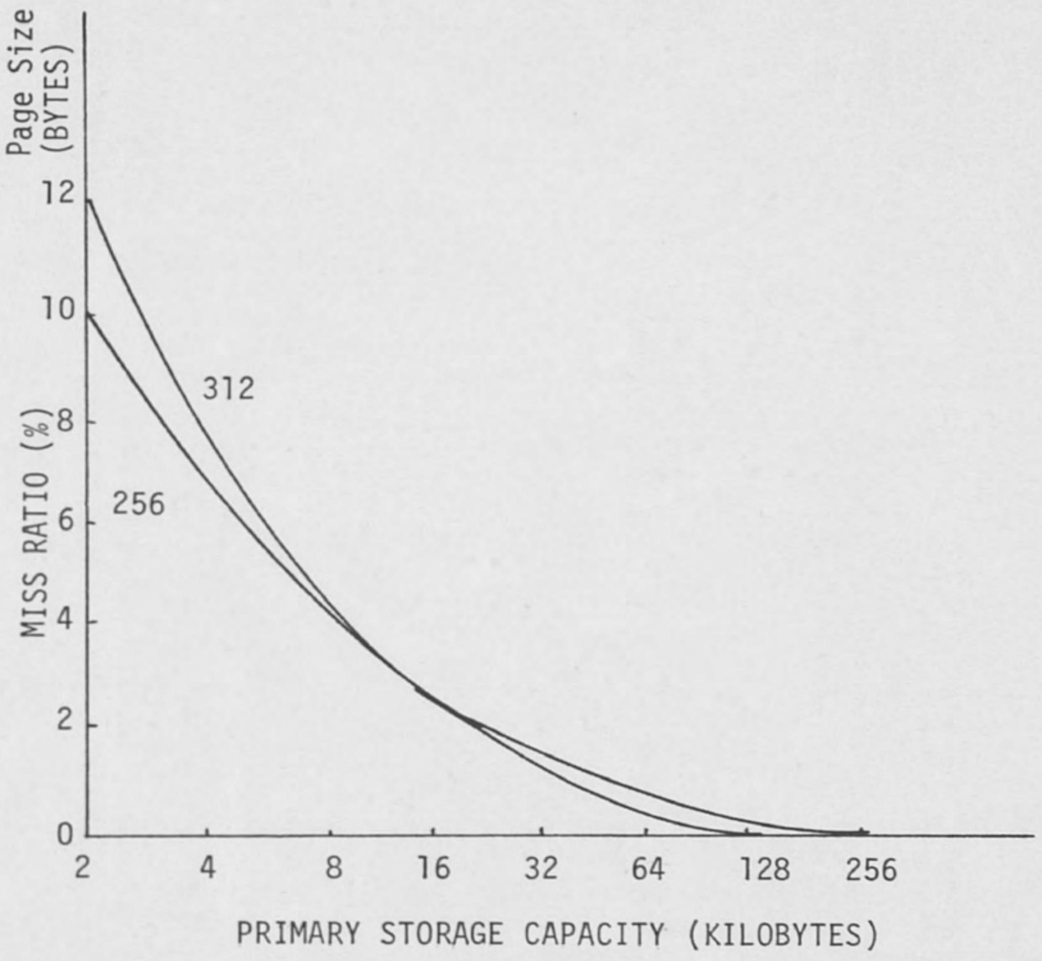

Figure 4. Relationship between primary storage capacity and Miss ratio

where C_1, C_2 and S_1, S_2 refer to the cost per bit and storage capacities of the different hierarchical levels of the virtual memory respectively (2).

Access time refers to the average time for the CPU to access a word in the memory and is given by

$$t_A = Ht_{A1} + (1-H)t_{A2}$$

where t_{A1} and t_{A2} are the access times for the different levels of the hierarchy (2). Access efficiency is defined as the ratio of access time of the main memory to average time required by the CPU to access a word in the memory. This is represented by 'e'. From the above equation 'e' can be represented as

$$e = \frac{1}{r + (1-r)H}$$

where

$$t_{A2} / t_{A1} = r$$

Just as virtual memory is implemented by different mechanisms like paging and segmentation, there exist different policies which are used to determine how information is to be allocated or moved in the memory. These policies can be classified as follows:

1. Fetch policies determine the time when the information should be loaded into the main memory.

2. Replacement polices determine which information is to be removed from the main memory.

3. Allocation policies determines the regions in the main memory where the incoming data should be assigned.

The fetch policies can be executed in two ways. One mode is called demand swapping and other is termed anticipatory swapping. In the first type, information is transferred to main memory only when it does not reside in the main memory and a request is generated for such a transfer. In the anticipatory type, as the name suggests, some amount of anticipation is utilized to transfer a block of information which may be required by the CPU. Such type of swapping is rather difficult to implement for long range estimates (i.e., estimating which information will be required over a long period of time).

The allocation process uses information stored in the main memory which is maintained by the operating system as a map. The map has an available space list specifying the regions which are available, to which new data can be transferred. The map may also keep information like an occupied space list which specifies the regions which are occupied and pages or segments which occupy it.

The allocation policies can be executed in either a static or dynamic way. The static way assumes that different blocks of information are bound to fixed regions in the main memory and cannot be changed during the program execution. The dynamic allocation policy is executed by first noting the way in which the memory is already occupied by different blocks. The dynamic allocation policy uses two methods, preemptive and nonpreemptive, for its application and is used mostly in the case of segmentation.

1. Preemptive allocation: In this strategy the incoming information can be assigned to space used by a currently running program. This necessitates shifting the currently running segment to some other region or expelling it totally from the primary memory. The segments in the memory are relocated such as to make a hole large enough to make space for a fresh segment. Expelling requires removal of segments which may be clean or dirty. Dirty blocks being ones which have been modified since it was loaded into the primary memory. Removal of dirty blocks involves rewriting them into auxiliary memory, unlike clean blocks which already have a an unchanged copy in the auxiliary memory. A technique called compaction can be used for relocation of segments existing in the main memory. In this technique all the holes present in the memory are combined into a single large hole to accomodate the incoming information. Compaction can be used every time a segment is swapped out of the memory but is rather time consuming process. Figure 5 depicts the compaction process.

2. Non-preemptive allocation: In this strategy a new segment of information can only be placed in a region which is unoccupied and large engough to accomodate it. This strategy can attain a high hit ratio with a good allocation algorithm. But in the wrost case this could lead to an excessive amount of swapping. This phenomenon is called thrashing. This strategy does not pose much problems since fixed size pages can be accomodated in any available page

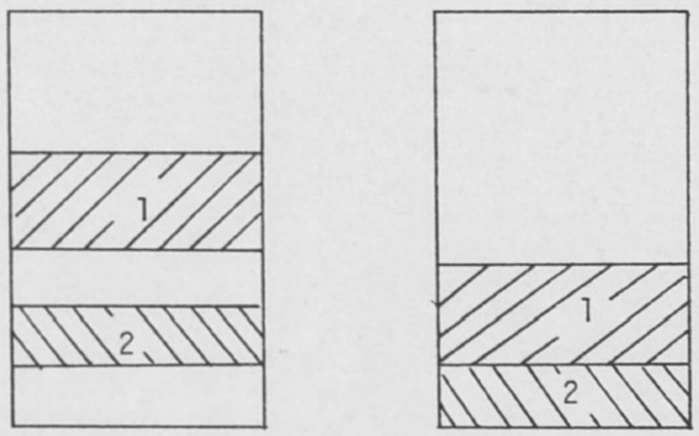

Figure 5. Compaction

location. But segments, being of varying size, need some policy to fit each in the best available space. Two policies exist called the best fit and the first fit, in case of nonassociative tables.

First Fit: In this, an incoming segment of information is placed in the first available memory space as obtained from the map.

Best Fit: This method scans the entire empty space map and places the incoming segment into an hole such that the minimum amount of space is wasted.

The first algorithm exhibits better efficiency because of the time saved in scanning the entire map. A diagram illustrating these algorithms is shown in Figure 6.

Figure 6 shows the first fit algorithm allocates incoming segment B_4 to space between B_1 and B_2 leaving a small hole but the best fit algorithm allocates B_4 to space between B_2 and B_3 thus minimizing any hole formation.

An useful parameter which optimizes the efficiency of a virtual memory system is the page size. The factors governing the page size are fragmentation and swapping operations. If the page size is large the information required next has a high probability of being present on a page in the main memory. This would thus reduce the swapping rate. The probability of internal fragmentation increases with increase in the page size as the last page will remain unfilled. Further, the page table increases if the page size is reduced; thus a compromise between a smaller page size and a large page size must be made to obtain an optimum page

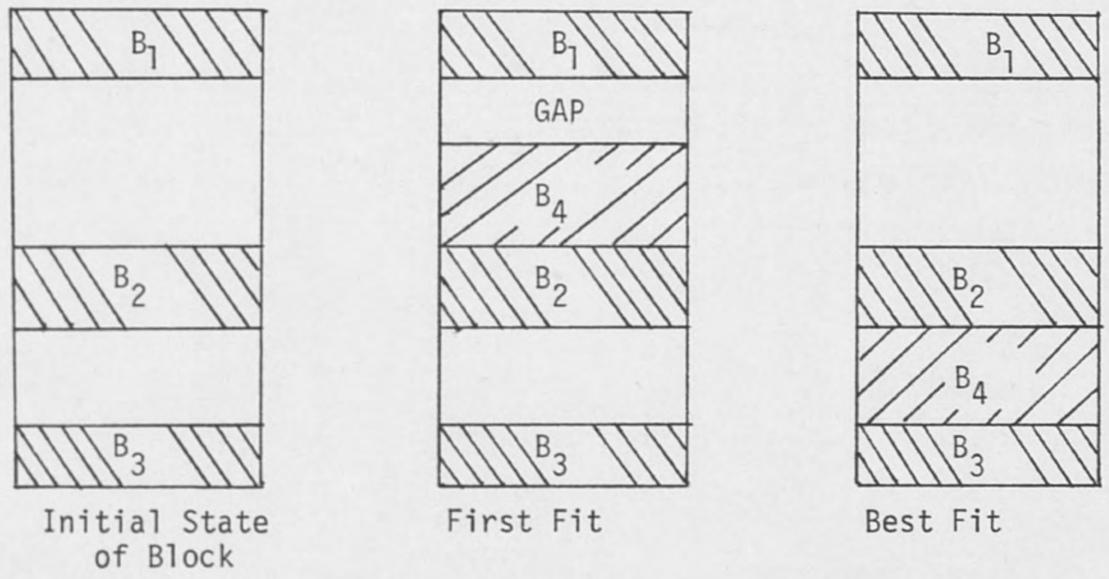

Figure 6. Blocks depicting the first fit and best fit algorithm

size. The optimum page size can be obtained as shown.

Let S_s be the average length of an active program part in words (Figure 3) and S_p the page size. An estimate of the optimum page size, S^{opt}, is one which maximizes the space utilization, R, defined as the ratio of the memory space occupied by the active parts of user program, S_s, to total memory space available, $S+S_s$. Space utilization is given as $R=S_s\ /(S + S_s)$where S is the memory space overhead associated with each segment). If $S_s>>S_p$ then

$$S = S_p/2 + S_sS_p$$

where S_s/S_p is the size of the page table associated with each segment and $S_p/2$ is the average number of words assigned to the last page of the segment. (2). Then the estimate of the optimum page size is one which minimizes S or maximizes the space utilization R. Thus by differentiating S or R with respect to S_p and equating it to zero we see that the value of S^{opt} obtained is the same in both cases and given by

$$S^{opt} = \sqrt{2S_s}$$

One can show that the allocation and replacement algorithms for segmented memories are more complex than the algorithms for paged memories. The problems of fragmentation exists in both paged and nonpaged systems. The fragmentation present in a paged system is internal and can be controlled by selecting a proper page size. In the nonpaged system external fragmentation exists and can be controlled by proper placement policies or by compaction. Thus the paged system divides the memory uniformly and also eases the transfer of pages from secondary memory to primary memory. This comparison between paged and nonpaged or segmented memories exposes the superiority of paging.

CHAPTER III

MAPPING FUNCTIONS

In a virtual memory system, the secondary memory contains a larger number of pages than the primary memory. To compress these secondary pages to primary memory a mapping function is used. Any page location in the primary memory storage must hold many different virtual pages, at different times. The actual number of virtual pages that a logical page location can accomodate varies with the mapping function. The minimum number is the total page capacity of secondary divided by the total page capacity of the primary memory and the maximum number is the total page capacity of the secondary storage itself. These two form the extreme cases representating the direct and associative mapping respectively. Between these two extreme cases there can exist any number of possibilities, which gives rise to set associative mapping.

The three different mapping functions are achieved by initially considering the following (3). The primary memory is divided into a fixed number of sets called frames and each frame consists of a fixed number of pages. All the virtual pages must be accomodated in these primary pages. Consider the secondary memory divided into groups of pages, each group being

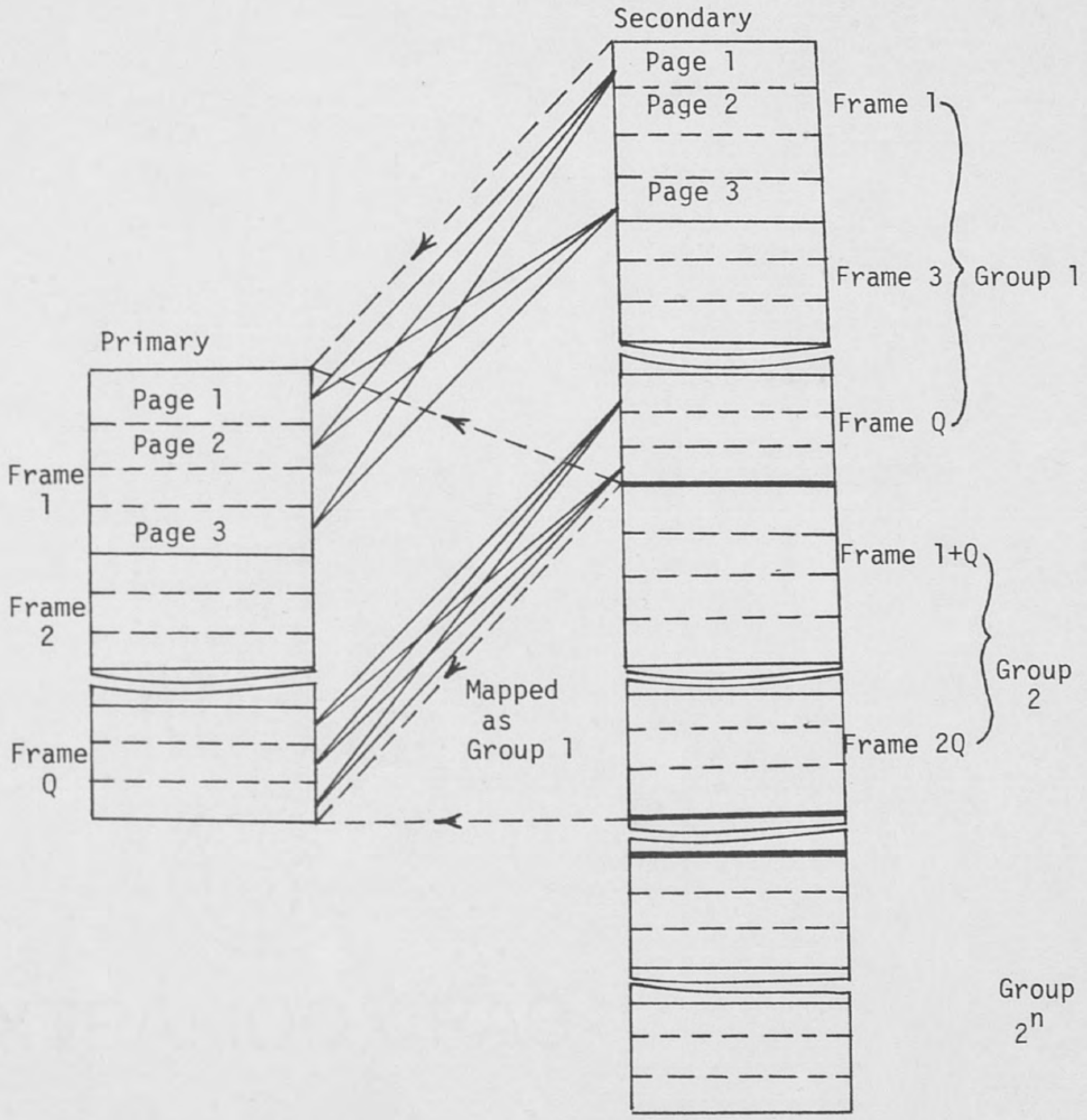

Figure 7. General mapping

equal to the length of the primary memory. Each group contains the frames and pages corresponding to the primary memory. Each frame in the primary memory must be shared by certain fixed frames in the secondary storage; for example, Figure 7 indicates frame 1 of the primary memory must be shared by frames 1, 1+Q, 1+2Q, etc., from the secondary memory. Pages within a frame of the secondary are associatively mapped into any of the pages within the frame of primary. All the virtual pages are thus assigned to one and only one logical frame and can be located in any one of the pages within that frame. Frames from different groups can be intermixed within the primary memory; hence not all frames from one given group need be simultaneously present in the primary memory.

If the number of pages per frame is small or the number of frames is large, then locating a page becomes easier because of the fewer logical places a page can reside in. Similarly, if the number of frames is small and the number of pages per frame is large, the number of places in which a virtual page can reside is larger, thus making it difficult to locate a page. The choice of mapping function is also dependent on the contention problem. It is required that the virtual pages from frames 1, 1+Q etc., in the secondary reside in frame 1 of the primary memory. Suppose a frame has 10 pages, and if it required that 6 pages from frame 1 and 5 pages from frame 1+Q of the secondary be present in frame 1 of the primary at one time, then the capacity of the primary frame will be exceeded and there will be two virtual pages competing for a single page in the primary frame giving rise to contention problem. This contention problem can be reduced if the data is organized properly in the secondary memory, i.e., if certain pages from the secondary are required to be present in the primary at one time, then these pages should be allocated in the secondary frames, such that these frames will not be required to share the same primary frame. But it is difficult to estimate how the data will be used, hence organizing data could present problems. Three types of mapping functions and their impact on the contention problem are discussed.

Fully Associative Mapping

In fully associative mapping, illustrated in Figure 8, the main consideration is to maximize the number of pages in a frame. Maximizing the number of pages in a frame results in a frame containing pages equal to the total primary capacity, and this would reduce the number of frames to one; thus, any virtual page can be mapped logically into any primary page. The contention problem for this type of mapping is minimum because two virtual pages can contend for a single primary page only when the total number of pages required simultaneously exceeds the primary memory capacity and this, in general, is unlikely; thus, the hit ratio provided is high. Since an associative type of compare is required in the address translation, this mapping is difficult to implement (3).

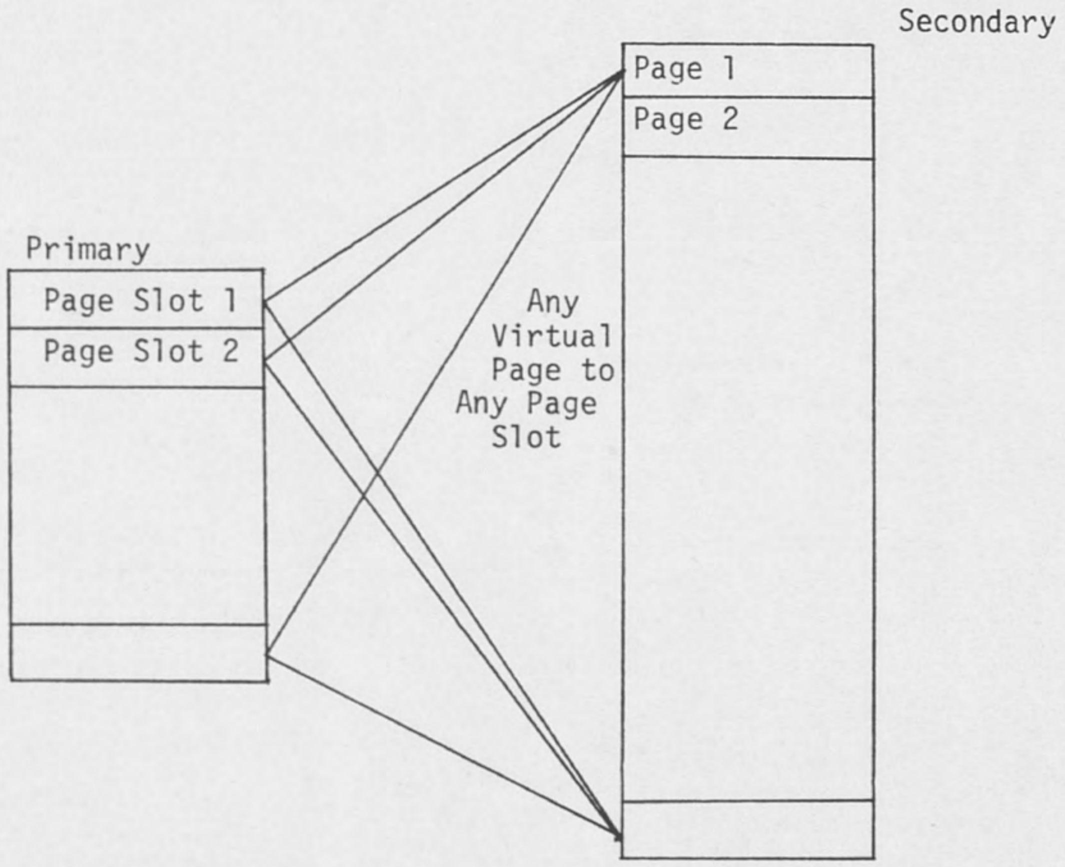

Figure 8. Fully associative mapping

Direct Mapping

The main aim in this type of mapping is to minimize the number of pages in a frame. This results in a frame having only one page and thus the number of frames equal the total primary memory capacity. Figure 9 illustrates that any page in the secondary memory can reside only in a specific page in the primary, i.e., page 1 in the primary must be shared by virtual pages 1, 1+Q, 1+2Q etc. In this case, if more than one virtual page, say 1, 1+Q etc., is required to be present in the primary page 1 at one time, then serious contention problems could occur as a result of more than one virtual page competing for a single page in the primary. This type of a mapping can result in a low hit ratio.

Set Associative Mapping

Set associative mapping is a compromise between the complex address translation problem in the fully associative memory and the serious contention problem faced in the direct mapping. The number of pages in a frame, in this case, is fixed to certain value say 2^n. A virtual page from certain fixed frames in the secondary can reside in any of the 2^n logical pages of the primary memory frame. For example, if the number of pages per frame is 2, then a virtual page can reside in either of the two logical pages in the primary memory, as shown in Figure 10. Contention problems caused because of two or more virtual pages competing for a single page in the primary can be reduced by selecting a higher number of pages per frame as this would allow more primary pages per frame in which the virtual pages can fit.

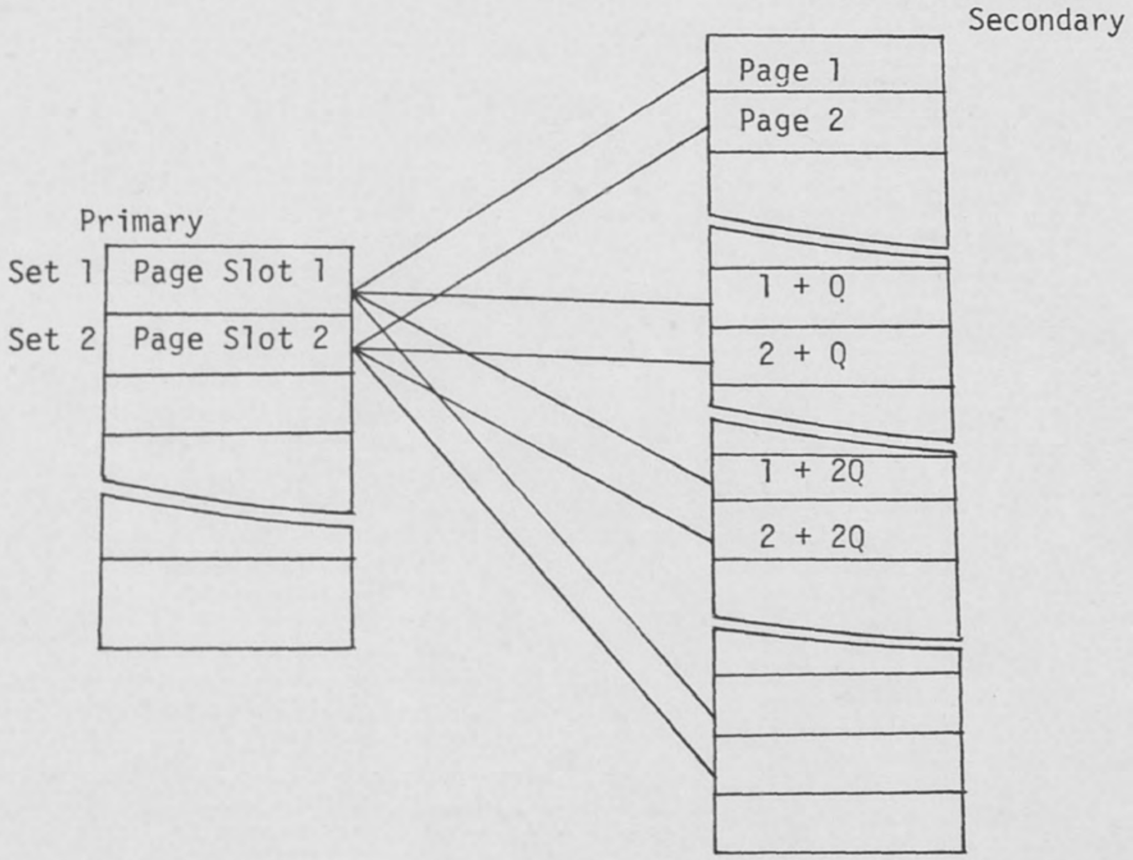

Figure 9. Direct mapping

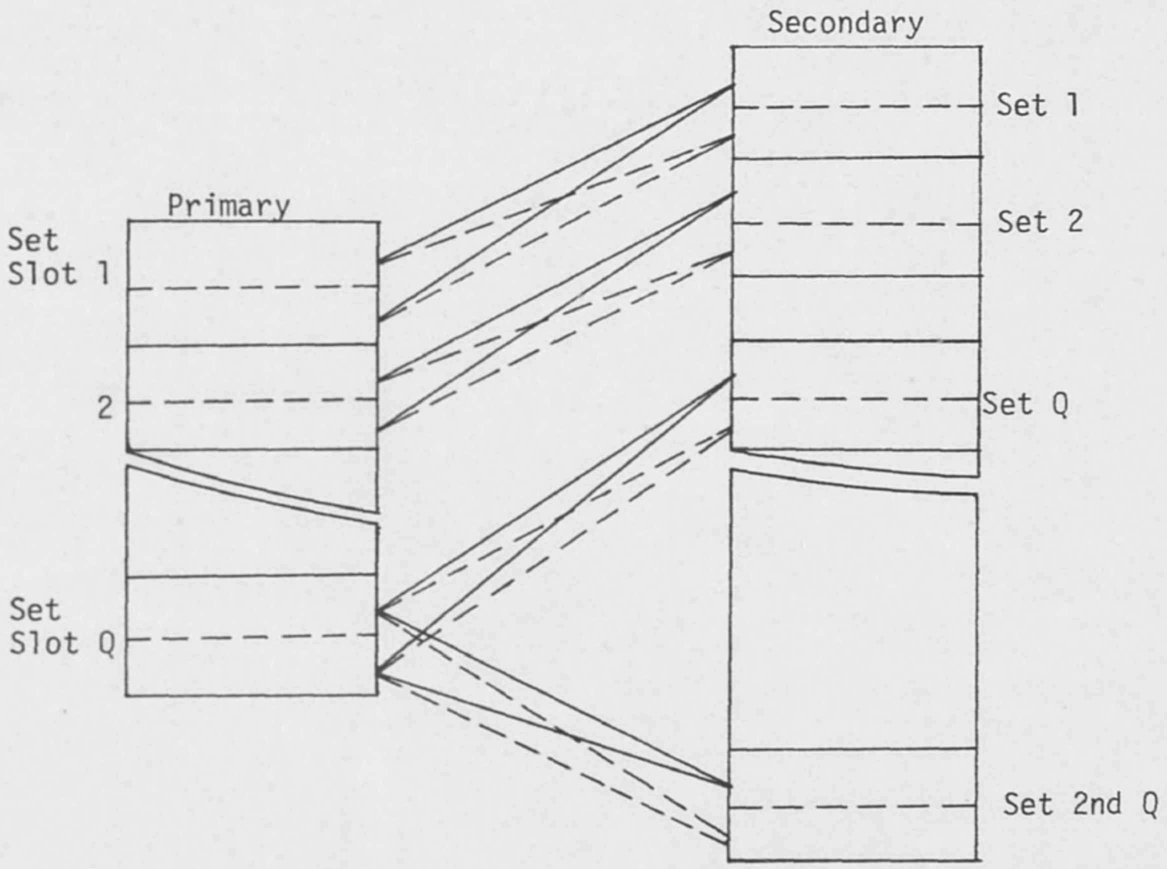

Figure 10. Set associate mapping

CHAPTER IV

ASSOCIATIVE AND CACHE MEMORIES

Associative Memory

Associative memory is a type of random access memory in which any stored word can be accessed directly by using all or a part of that word as a key. Thus, all the words that match the key are specifically identified as available and can be addressed in the subsequent memory cycles. Such memories are also called content addressable memories.

Figure 11 shows the structure of a word organized associative memory (2). The main blocks of the structure are the mask register, the select circuit and the storage array where the words are stored. The masked register is used to specify the desired key. The select circuit enables the data fields to be accessed. Each unit of stored information is a fixed length word. The operation of the memory is as follows. The hit word is in the input register. The mask register identifies which fields of the hit word are to be compared to all corresponding fields in the storage cell array. The masked hit-word is called the key. Those words which match the key emit a match signal which enters the select circuit. The select circuit enables the data fields to be accessed. If several entries have the same key, then the select

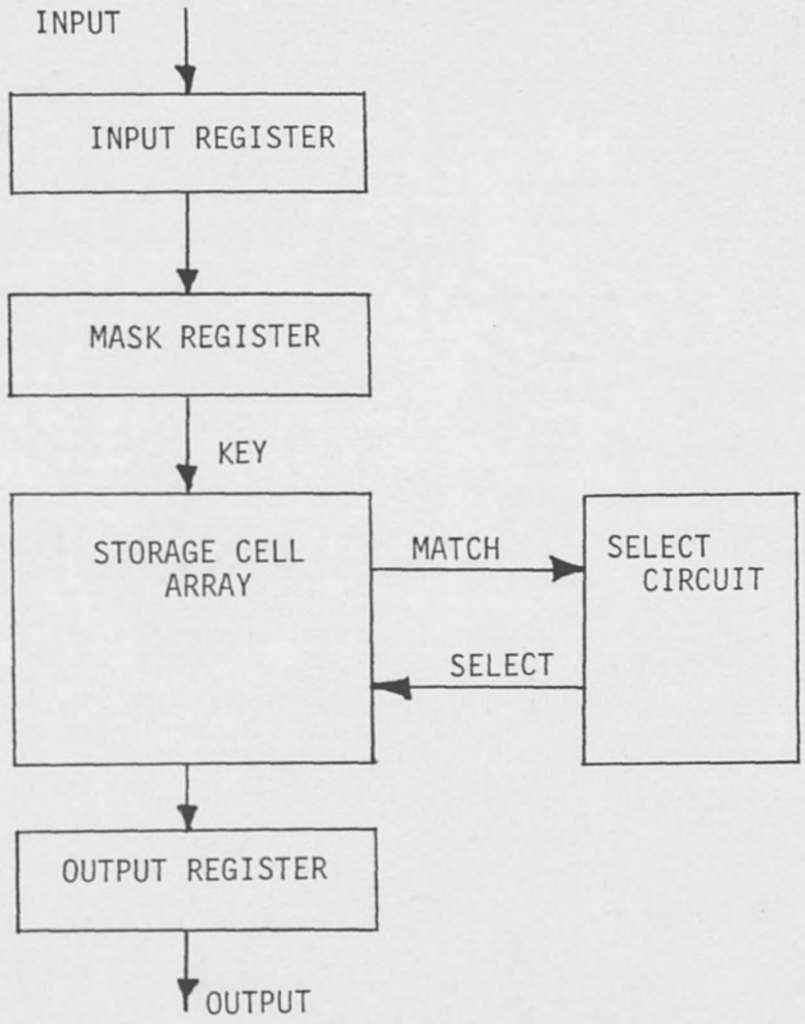

Figure 11. Structure of word-organized associative memory

circuit decides which data field is to be read out, maybe in some predetermined order. Each word must have its own match circuit since each word's key must be compared with the input key.

Cache Memory

A cache is a small fast memory placed between the CPU and the main memory to bridge the speed-gap. It acts like a buffer for the main memory and forms a two level hierarchy with it; this time the cache acting like a primary memory and the main memory being the secondary. Figure 12 illustrates a three-level hierarchy with the cache included.

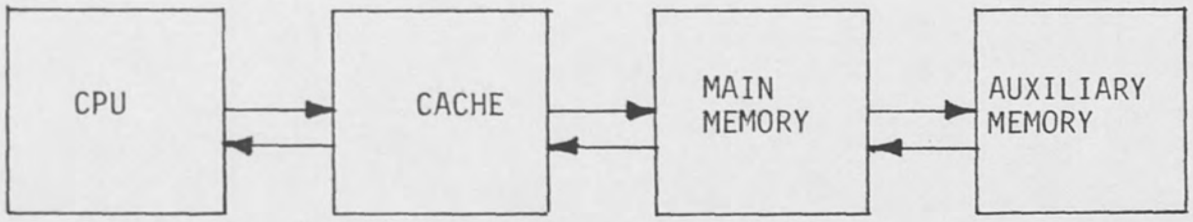

Figure 12. Three level hierarchy

The cache should satisfy most of the memory references so that its hit ratio should approach 1. The cache-main memory configuration is similar to the main-secondary hierarchy. The access time of cache-main memory is very much less than main-secondary memory, the page size is also much smaller in case of the cache. The cache-memory address translation is normally implemented by hardware. Another important difference is that the processor has direct access to both cache and main memory. The architecture of Figure 13 illustrates that when the information is not found in the cache, the CPU can directly access the main memory. The cache hit ratio can thus be increased.

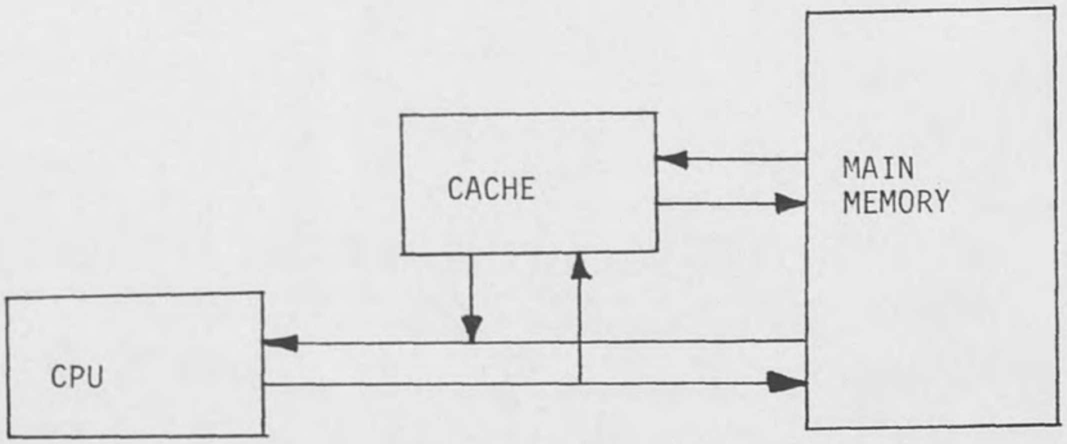

Figure 13. Cache used in Hierarchical memory

CHAPTER V

REAL TIME VIRTUAL MEMORY

In this section an effort is made to design a nearly real time system that can decode 32 bits of virtual address from the secondary memory to obtain the primary memory address locations. We define real time as a virtual memory address translation time of 300 nsec but which may range between 100 nsec to 900 nsec. The primary memory size is 16 megabyte and is organized in 16K pages, each consisting of 1024 bytes. The 10 least significant bits of the 32 bit virtual address give the page address. A block diagram for the system is shown in Figure 14.

The solution technique for this problem is as follows: We use an associative memory which can potentially solve the problem. A modified organization of this is shown in Figure 15.

X_i is a 22 bit word of associative memory which identifies a page of 1024 words of virtual memory (i.e., on the disk) which has been transferred to the primary memory page address Y. Y_i is a loaded page in the primary memory. This page of 1024 bytes corresponds to secondary location X. Each word of memory Y contains a primary memory page's address.

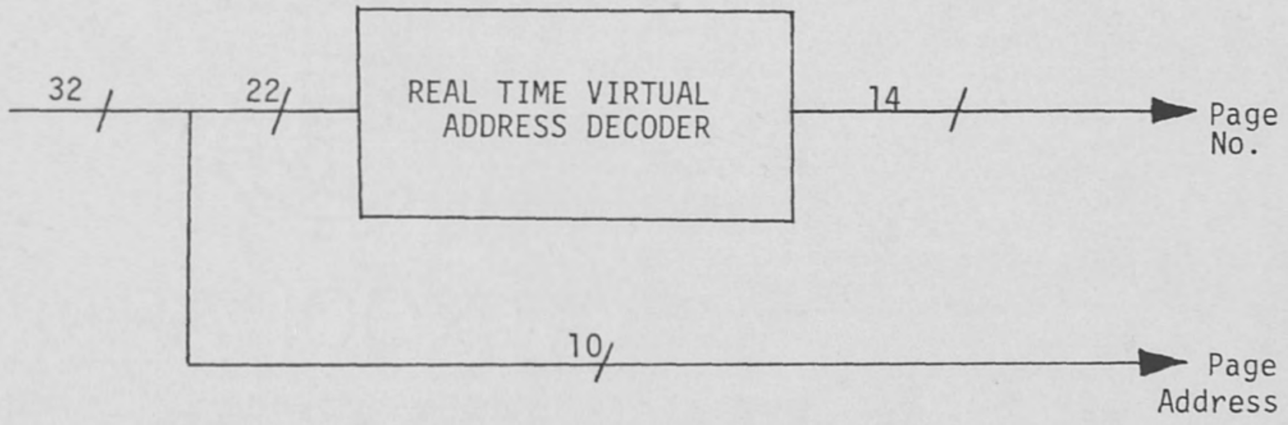

Figure 14. The real time virtual address decoder

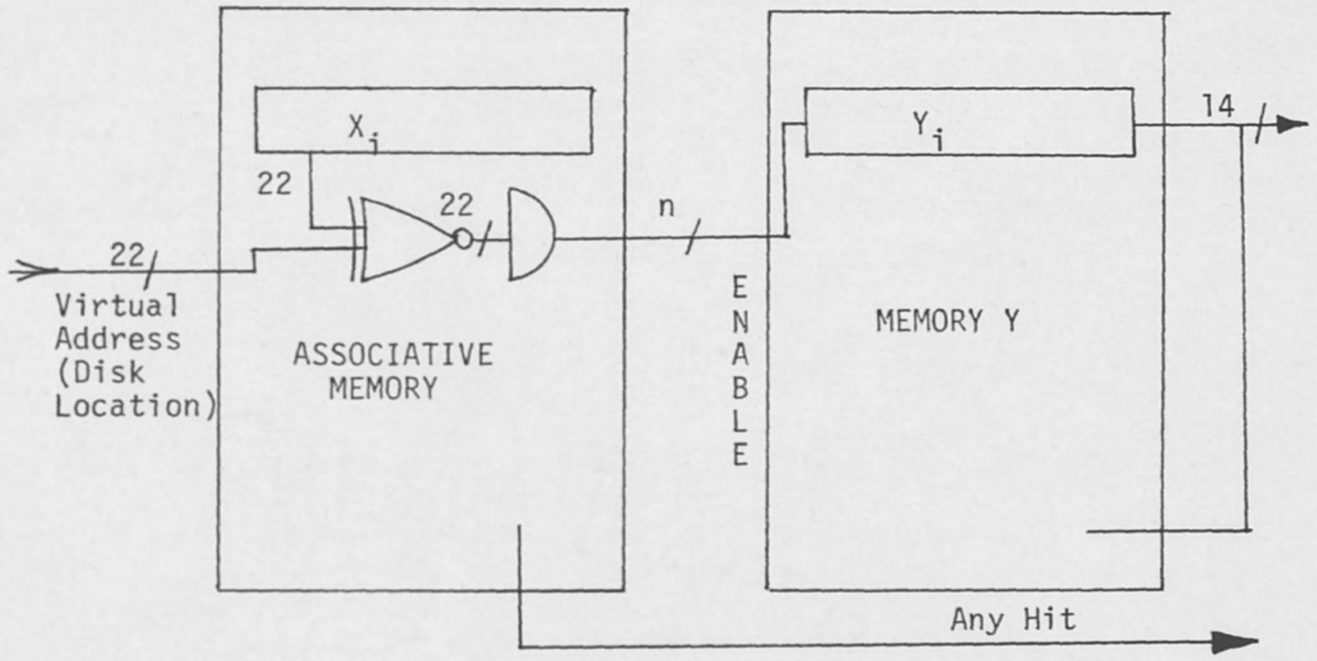

Figure 15. Structure of virtual address decoder

We need to put the associative memory X and memory Y on one IC. If the present memory is 2K * 8 to 8K * 8 and if the exclusive OR gates require equal or less area compared to a memory cell, then we assume that one IC can hold about 8k bits of memory plus the exclusive OR gates to establish the associative memory. The size of the associative memory X and memory Y would be about

$$\frac{8 \times 1024}{36} = 228$$

where 36 = 22 + 14 = memory bits per map location.
So we assume 256 locations of virtual address can be obtained on one IC (i.e., the value of n in Figure 15 is equal to 256).

Figure 16 gives the structure of a virtual address decoder IC. In Figure 17, the virtual address is compared with the addresses in the associative memory simultaneously and if a match occurs, a hit has occurred and the primary memory register gives the address of the data in the primary. Buffering is required to store the virtual address so that it can be simultaneously compared with the addresses in the associative memory.

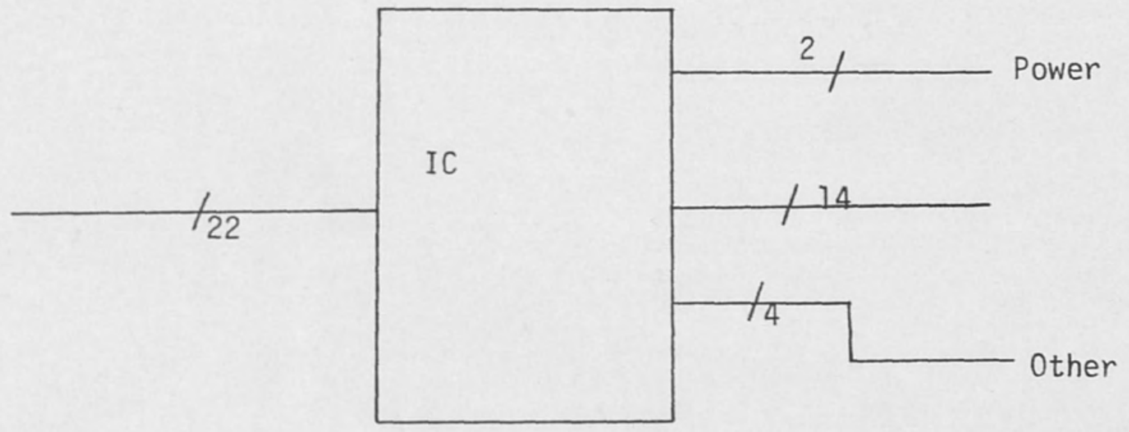

Figure 16. Virtual address decoder IC

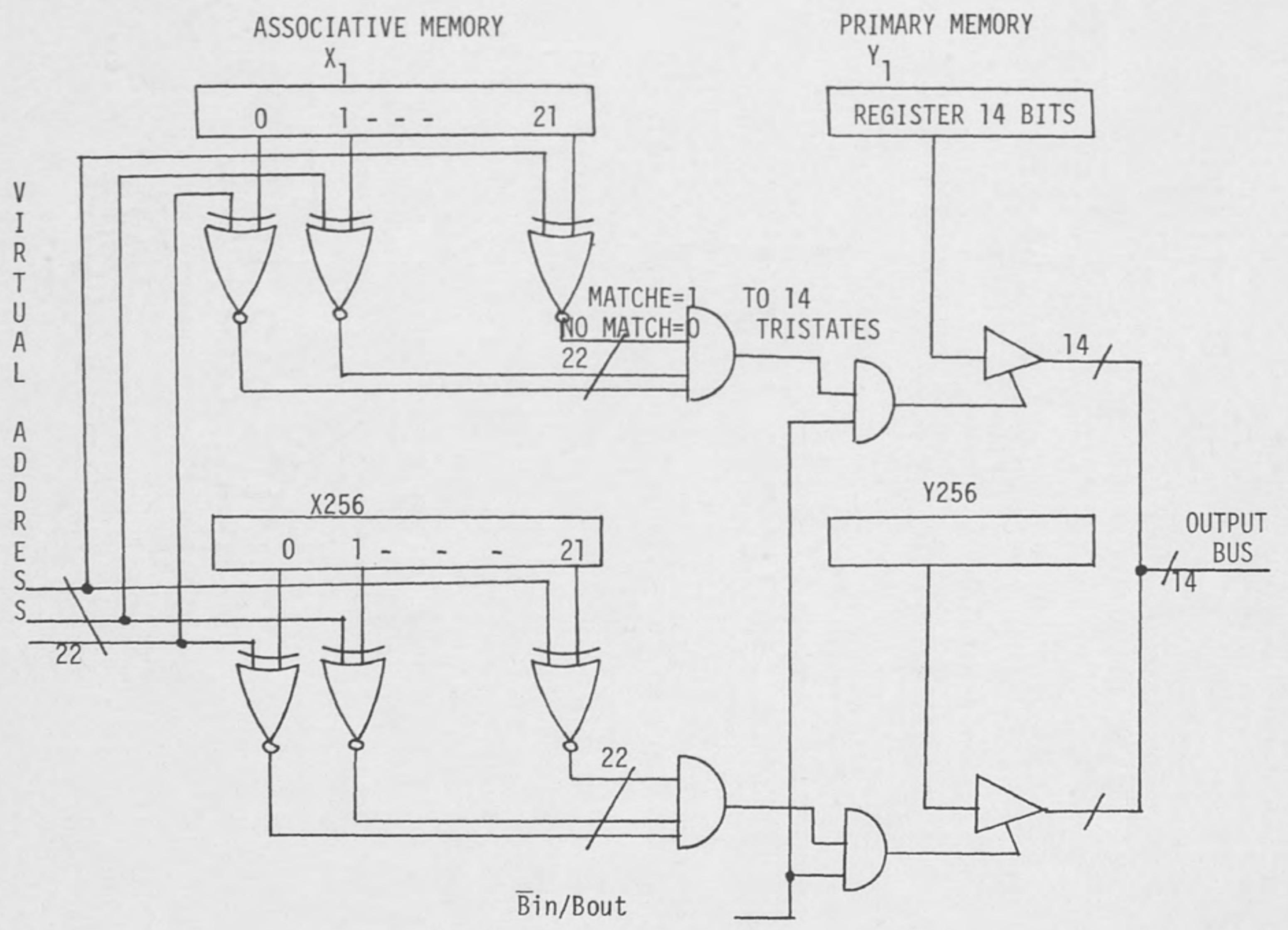

Figure 17. Associative memory function of virtual address decoder

Figure 18 illustrates a possible pin designation for the virtual address decoder. One can define the pin functions as follows:

$\bar{R}/W = 0$	Read
$\bar{R}/W = 1$	Write
$\bar{B}in/Bout = 0$	Output bus B disabled
$\bar{B}in/Bout = 1$	Enable B bus

The functions of pins Mode/ Hit and $V/\bar{L}$ can be compounded and put in a table form.

	Mode/Hit = 0	Mode/Hit = 1
$V/\bar{L} = 1$	No Hit	Virtual Address Hit
$V/\bar{L} = 0$	Write B to Y Register	Write B to X Register

The bus B is bidirectional because it is used to either read out data or it can also be used to write into the IC. The Mode/Hit bus is also bidirectional because if $V/\bar{L} = 1$ the bus reads out from the IC and if $V/\bar{L} = 0$, Mode/Hit is an input line. The method to read and write from or to the IC is given below.

First the method to read from the Virtual Address Decoder is given:

1. Set $\bar{R}/W=0$ and read the virtual address on A.

2. Decode A for an hit. If $\bar{B}in/Bout = 0$, then the output bus B is disabled.

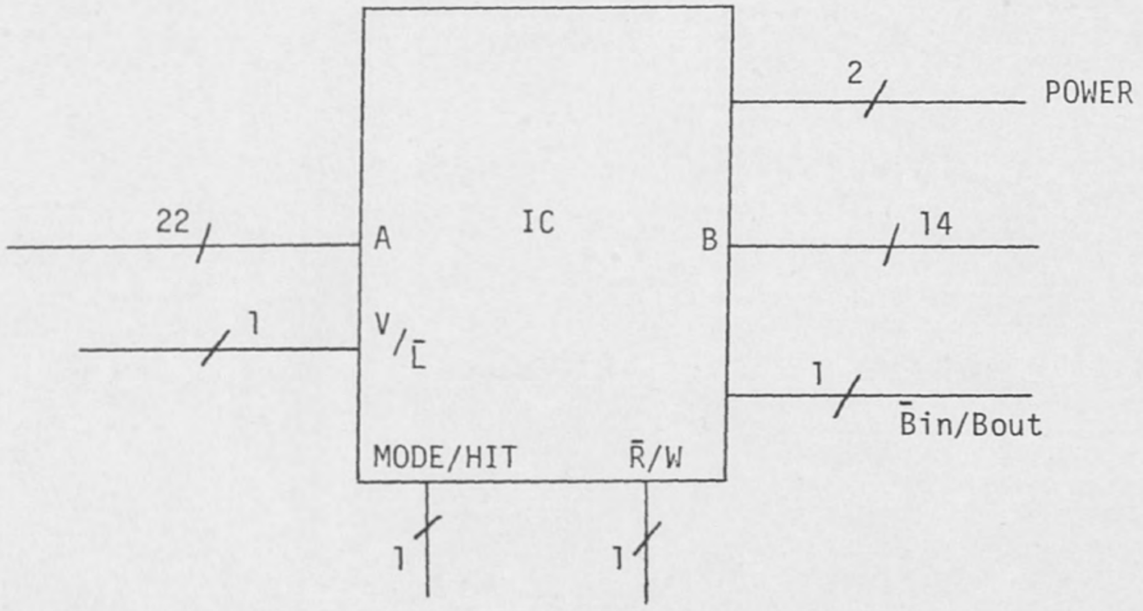

Figure 18. Pin designation for the I.C.

3. If Hit = 1, then the IC decodes the virtual address to the physical location.

4. Set Bin/Bout equal to 1 to allow the decoded physical address to appear on the B bus.

To write into the Virtual Address Decoder the following steps are used.

1. Set $V/\bar{L} = 0$ thus Mode/Hit is an input line and we can write to the Virtual Address Decoder.

2. Set Bin/Bout = 0 which would disable the B bus output drivers.

3. Set $\bar{R}/W = 0$ thus we are ready to read in information to be put on the B bus.

4. Now set the Y register address on the B bus and allow set up time. The address on B bus is now read into the IC.

5. Set Mode/Hit = 0 which chooses Y register to be addressed by the B bus.

6. Set Y register data on the A bus.

7. Set $\bar{R}/W=1$ which writes A bus data in the Y register.

8. Set $\bar{R}/W=0$ so that the address and/or data can change without effecting data stored in the device.

9. Now set Mode/Hit=1 which chooses the X register to be addressed by the B bus.

10. We now have to write data in the X register at the address specified on the B bus. The B bus address should not change between these writes since (X,Y) form an associative pair.

11. Set $\bar{R}/W=0$ and remove data input to B bus.

12. $\bar{B}in/Bout=1$ enable the B bus.

13. Set $V/\bar{L}=1$ which means that we are in the virtual mode.

CHAPTER VI

REPLACEMENT ALGORITHMS

A virtual memory system is a combination of hardware and software techniques. The memory management software system handles all software operations for efficient utilization of memory space. It must decide (1) which page in the primary ought to be removed to make room for new one, (2) when the new page is to be transferred from the secondary memory to the primary memory, and (3) where the page is to be placed in the primary memory. The criteria for choosing a page of information to be replaced is called the replacement algorithm or policy. The main objective behind selecting a replacement algorithm is to maximize the hit ratio and hence to ensure that a page which is to be referenced is present in the primary memory.

The hit ratio can be maximized if the time interval between successive page faults can be increased. One strategy, called the optimal replacement strategy, or also known as OPT, achieves this by estimating at a time, say t_1, the time t_2 when a particular page will be referenced again. A page to be thus replaced is one for which t_2-t_1 is maximum. This means that the page targeted for replacement has less chance of being referenced soon compared to other pages. This policy is implemented in two runs. First is a simulation run to determine the sequences of the addresses generated by a program and noting the times when the different blocks are referenced. The second run executes this policy by replacing appropriate blocks. This is not a very practical strategy because of the time spent on simulation runs, and also the block sequences may be long, making the simulation expensive (2).

In another policy, called the FIFO or first-in-first-out (2), the block selected to be replaced is one which has been loaded the first. Each block has associated with it a loading sequence number in the occupied space list which is updated each time a block is transferred to or from the main memory. The block to be replaced can be easily determined by inspection of these numbers. This policy is easy to implement but has some defects. A block with high probability of being referenced may be displaced because it happens to be the oldest block.

Least-Often-Used and Least-Recently-Used (2) are two other replacement algorithms which select a block to be replaced that has been referenced the minimum number of times. These superseded the FIFO algorithm in a way that a block with high probability of being referenced will not be replaced. These algorithms are rather difficult to implement compared to the FIFO because special hardware is required to maintain a record of the use of the pages so that the ones referenced least can be selected to be replaced (2).

In the least often used policy the pages are arranged in a descending order of use count, such that the table moves the pages used most to the top and the ones used least to the bottom. This policy uses counters to maintain the age of the pages. The size of a counter must be large compared to the length of a page. Figure 19 depicts the implementation of this policy. Whenever a page is referenced, the counter associated with it is incremented, and this count is placed above one with a lower count. This shifting of counts, such as to place them in a descending order of magnitude, is achieved by using a temporary register (TEMP).

The least recently used policy is implemented by means of constantly updating the table (i.e., a page which has been referenced most recently bubbles to the top of the associative memory). Figure 19 illustrates the implementation of this policy. Whenever a page is referenced, it first goes to a temporary register where it is stored until the remaining pages are pushed down; then the page in the temporary register is inserted at the top of the table to indicate that it is the most recently used page.
Then the virtual pages at the bottom of the least used tables are the ones which can be replaced. The operation of the LRU hardware can be explained in the following steps.

1. The primary memory page address enters A.

2. One of the registers contains a matching page address so a hit is activated at register i.

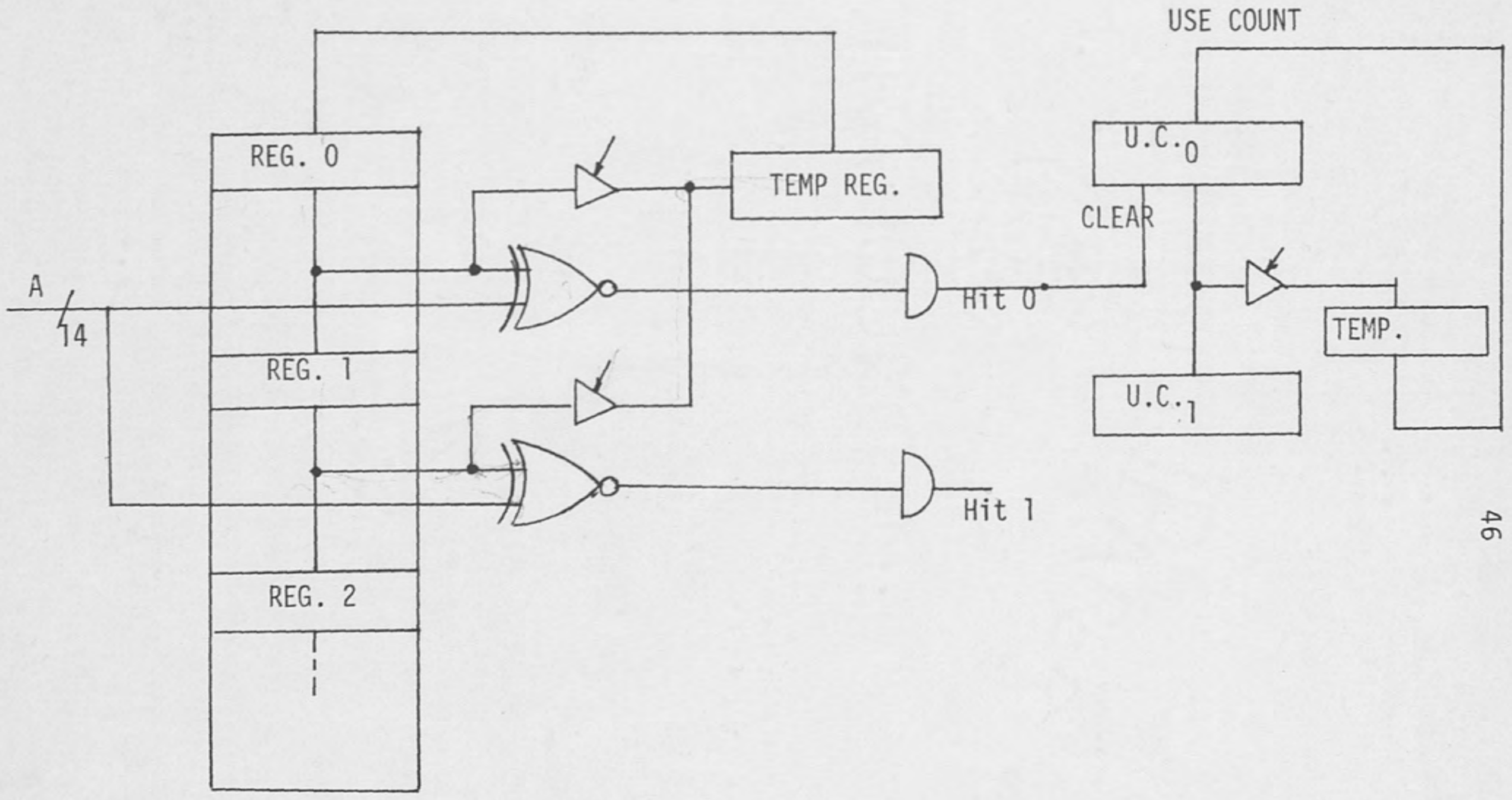

Figure 19. Hardware implementation of LRU and least often used algorithm

3. The contents of register i are loaded in the Temp register.

4. Then the register set from 0 through i are shifted as follows:

$(Reg_i \leftarrow Reg_{i-1} \leftarrow Reg_{i-2} \leftarrow \ldots \leftarrow Reg_0 \leftarrow \text{Temp Reg})$

(i.e., perform the shift on the left first and end with the shift on the right from Temp register to Reg_0).

5. Registers i+1 to n remain unchanged.

In this way most recently used address move to the top of the associative memory and least recently used primary addresses move down.

CHAPTER VII

VIRTUAL MEMORY FOR MICROPROCESSORS

This chapter is a reference to the work of Judith A. Anderson and G. J. Lipovski, (4). The availability of efficient and inexpensive microprocessors has changed the computer outlook drastically. The cost of input/output devices are relatively high compared to the microprocessors, so I/O devices must be shared among different microprocessors. The cost of required memory also outruns the cost of the microprocessor. Thus the size of local memory should be reduced but should not be too small because the cost of software then increases. A small virtual memory for each microcomputer provides a good outlet for these problems.

A virtual memory physical page block diagram is shown in Figure 20 and some implications behind its development are considered. A page of virtual memory with associated logic is shown in Figure 20.

A few things must be considered before deciding what functions of a virtual memory should be divided between hardware and software. One is that the system should be extensible i.e., addition of new pages should involve minimum change in hardware and software. Execution time or storage overhead should be minimum and operating system requirements should be minimum. Much of the system should be independent of the processor type.

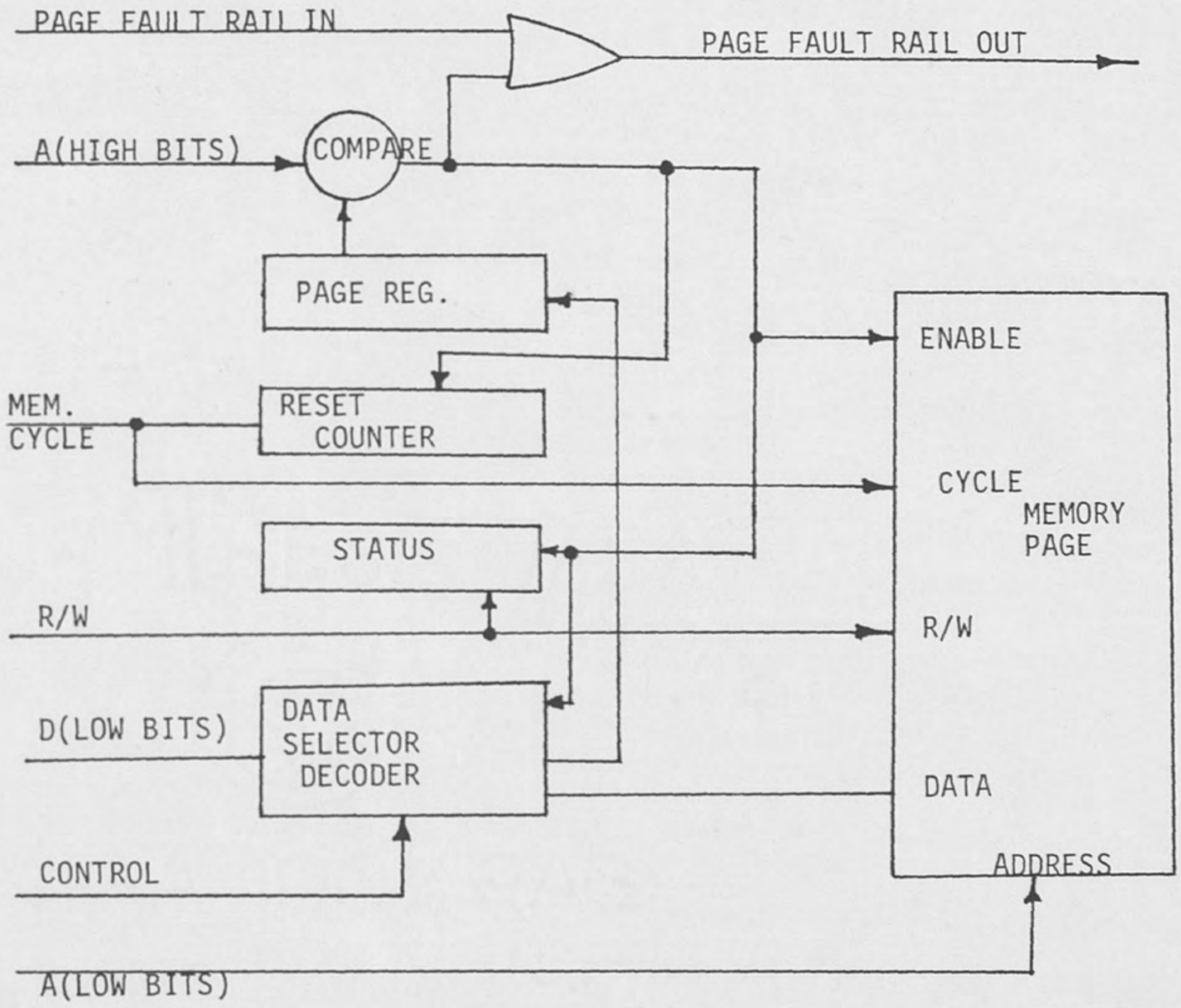

Figure 20. A virtual memory physical page

The address translation for the primary memory and the page fault detection should be done by hardware because software involves large execution time as well as storage overhead. It is better for each page of the memory to have associated with it the hardware required for the above functions, because even though the cost may be a little more, the system can be extensible. This strategy would allow the address translation logic to be incorporated directly on a memory chip containing this page. This gives some idea of the construction of the virtual memory in Figure 20. There is a page register which gives the virtual page number. This is compared with higher order bits sent by the microcomputer. If a match occurs the memory is enabled and the word, selected by the lower order bits of address, can be read or written. If no match occurs, a page fault takes place which can be handled by the interrupt facility. If a page fault occurs some page must be expelled to make place for a new incoming page. The page to be replaced can either be dirty or clean depending on whether the page has been modified or not since being loaded from the secondary to the primary. If a page is dirty then it needs to have a copy in the secondary. Hardware can be used to distinguish a clean or a dirty page because only one bit is required to do this comparison. The bit can be set if the page is dirty. If software is used to distinguish a clean or dirty page, additional subroutine is required which may involve a relatively long execution time.

The Least-Recently-Used algorithm can be implemented either by using the associative queue method or by the use of counters. Associative queue involved pushing the referenced page to the top. This forces the least recently used page to the bottom, which can thus be displaced. The use of counters involve associating a counter with every page. When a page is accessed, the counter associated with it is reset and counters of all other pages are incremented. When a page is required the contents of the counters associated with every page is read and the maximum found. The maximum value of the counter corresponds to the least recently used page. Overflow of counters should be avoided, but this could present some problems. If the overflow is avoided by holding the counter which has reached it maximum count constant, then choosing the least recently used page would be difficult as there could be more than one counters which may have reached their maximum value. Thus even though a page with maximum count will be displaced, this page may not be one which was least recently used. But the counter method provides more flexibility than the associative queue method and is preferred.

The instruction which was being executed when the page fault occured should be re-executed, because either the memory data being requested was not available or the instruction did not reside in the primary memory. Thus the address of the instruction and the page number of the missing data should be saved. The virtual page number being requested during page fault can be determined by loading only one register which is enabled by the page fault. Reading the register would require only one command. When a page fault occurs, a page fault interrupt must be enabled because the program may continue with invalid results.

The use of virtual memory in microcomputers can be obstructed because of the amount of space taken up by the operating system in the real memory. A solution for this could be that the operating system that services the interrupts and errors for input/output devices can be resident in the controller while the actual service routines for the keyboard and display in the terminal need not be resident but could be paged in when required. These are a few of the things to be taken into account for a virtual memory.

The virtual memory described in this chapter uses a method wherein every page of the main memory has associated with it a page comparator and counter. The address translation and page fault detection hardware is cellular, i.e., each page of the memory has an associated cell for address translation and page fault detection. This can help to incorporate the entire logic on the memory chip containing that page and thus also make the system extensible. But the number of comparisons required in this system are very many. The system described in Chapter V (Figure 17) uses an associative memory to carry on the address translation function. The comparisons carried on are associative and the same number of separate comparisons are required for each page as a system based on Figure 20. The Least-Recently-Used replacement algorithm used is implemented in Chapter V by the use of an associative queue method as compared to the counter method adopted here.

CHAPTER VIII

CONCLUSION

A virtual memory system is a very efficient and economical system for storing information. Virtual memory offers advantages of low cost of bit storage and high access rates which are possible because of the hierarchical use of different memory technologies. The concept of address space removes much of the restrictions posed by a nonvirtual memory system in which the user is limited in addressing to the actual size of the main memory. Also, the automatic storage allocation which is achieved by the operating system relieves the user of any storage allocation problems. Virtual memory is very efficient and useful where time-shared systems are used. But even with these advantages certain problems exist which are of interest.

Since the virtual memory gives an illusion to the programmer as being one single large directly addressable main memory, the programmer may tend not to restrict the size of the program as he would have in a non virtual memory system. This illusion may generate unnecessary overhead in the automatic storage allocation mechanism and reduce efficiency of multiprogramming operation. Another problem which exists is of fragmentation. The type of fragmentation prevalent in paged and nonpaged systems are different, but both basically cause the loss of useful storage space. Many time-sharing systems tend to use demand paging policy which tends to increase the cost of operation (1). It is desired that the hit ratio should be high for efficient operation; if this happens to be low, an excessive amount of swapping takes place leading to the phenomenon called thrashing.

Virtual memory is implemented by different mechanisms and policies which are used to determine how the information is to be allocated or moved in the memory. Paging and segmentation are two mechanisms. Paging divides the memory into fixed size regions and offers a uniform treatment of the memory, but it suffers from internal fragmentation and thrashing problems under multiprogramming. Segmentation uses variable length blocks or segments and these are susceptible to external fragmentation. Paging is preferred to segmentation because of the ease it offers in memory management and transfer of pages.

In a virtual memory operation, it is required that the information be present in the main memory when it is to be referenced, otherwise the information needs to be transferred from the auxiliary memory which will be very time consuming. The locality of reference property, according to which data normally clusters in groups, is very useful and gives rise to the important concept of implementation of virtual memory, namely paging.

Memory, being a limited resource, must be shared among different programs. Dynamic allocation allocates and also alters (i.e., in case of multiprogramming) the storage space during program execution. The allocation techniques discussed were the preemptive and nonpreemptive. The preemptive technique applies for both paged and segmented systems. In this technique, the incoming information could be assigned space by expelling or rearranging (i.e., case of segments) the existing pages or segments. In the nonpreemptive method, used only in the case of segmentation, the segment could be placed in a region if the region was unoccupied and large enough to accomodate the segment. The best and the first fit were the two ways of achieving the nonpreemptive technique.

In the preemptive allocation strategy the pages or segments could be expelled or preempted by using certain replacement policies. The three different policies discussed were FIFO, least recently used, and least often used. FIFO even though easy to implement, was not practical, as it expelled blocks independent of use and only on the order of access. The other two policies work by expelling pages which have not been referenced often. Least often expelled by using counters and least recently used technique by which the addresses referenced most recently were always found at the top of the list leaving the lower addresses to be preempted. Either the least recently used or least often used policy is preferred to FIFO.

Use of cache and associative memories have great influence on the system performance. A cache is a small fast memory used mainly to bridge the CPU and main memory speed gap. The cost of this element restricts its size, but because of the fast access time and also because of the concept of locality of reference property, the cache hit ratio can be made high. Associative memories offer an advantage of time saving by comparing pages for matches in one memory cycle. The real time virtual address decoder discussed uses an associative memory for performing the address translation function.

REFERENCES

1. Denning, Peter J. "Virtual Memory." Computing Surveys 2 (September 1970): 155-187.

2. Hayes, John P. Computer Architecture and Organization. New York: McGraw-Hill, 1978.

3. Matick, Richard E. Computer Storage Systems and Technology. New York: John Wiley & Sons, 1977.

4. Anderson, Judith A., and Lipovski, G. J. "A Virtual Memory For Microprocessors." IEEE 2nd Symposium On Computer Architecture, pp. 80-90. Long Beach, California: IEEE Computer Society Publications Office, 1975.

BIBLIOGRAPHY

Anderson, Judith A., and Lipovski, G. J. "A Virtual Memory For Microprocessors." IEEE 2nd Symposium On Computer Architecture, pp. 80-90. Long Beach, California: IEEE Computer Society Publications Office, 1975.

Crofut, W. A., and Sottile, M. R. "Design Tec-niques of a Delay Line Content Addressed Memory." IEEE Transactions On Electric Computers. EC-15(4) (August 1966): 529-534.

Denning, Peter J. "Virtual Memory." Computing Surveys 2 (September 1970): 155-187.

Digby, David W. "A Search Memory For Many-To-Many Comparison." IEEE Transaction On Computers. C-22(8) (August 1973): 768-772.

Hayes, John P. Computer Architecture and Organization. New York: McGraw-Hill, 1978.

Hill, F. J., and Peterson, G. R. Digital Systems: Hardware Organization And Design. New York: John Wiley & Sons, 1978.

Johnson, Colin R. "Microsystems Exploit Mainframe Methods." Electronics (August 1981): 119-127.

Knuth, D. E. The Art Of Computer Programming, Vol. 2. Reading, Massachussetts: Addison-Wesley, 1968.

Lipovski, Jack G. "On Virtual Memory And Micronetworks." IEEE 4th Annual Symposium On Computer Architecture, pp. 125-134. Long Beach, California: IEEE Computer Society Publications Office, 1977.

Matick, Richard E. Computer Storage Systems and Technology. New York: John Wiley & Sons, 1977.

Miller, Richard E. Computer Storage Systems And Technology. New York: John Wiley & Sons, 1977.

Strecker, William D. "Cache Memories For PDP-11 Family Computers." IEEE 3rd Annual Symposium On Computer Architecture, pp. 155-158. Long Beach, California: IEEE Computer Society Publications Office, 1976.

Vogt, Peter D. "Virtual Memory Extension For An Existing Minicomputer." Computer Design. (July 1981): 151-156.