CHAPTER 8

Memory Organization

Learning objectives

- Introduction
- Types of Memory
- The Memory Hierarchy
- Main Memory
- Cache Memory
- Cache-Mapping Functions
- Replacement Algorithms
- Virtual Memory
- Paging
- Segmentation
- Numerical Examples

INTRODUCTION

In this chapter, we study the computer memory system. As we know that without a memory no information can be stored or retrieved in a computer. It is interesting to observe that as early as 1946 it was recognized by Burks, Goldstine, and Von Neumann that a computer memory has to be organized in a hierarchy. In such a hierarchy, larger and slower memories are used to supplement smaller and faster ones. This observation has since then proven essential in constructing a computer memory. We have demonstrated the use of flip-flops in storing binary information. Several flip-flops put together form a register. A register is used either to store data temporarily or to manipulate data stored in it using the logic circuitry around it.

The memory subsystem of a digital computer is functionally a set of such registers where data and programs are stored. The instructions from the programs stored in memory are retrieved by the control unit of the machine (digital computer system) and are decoded to perform the appropriate operation on the data stored either in memory or in a set of registers in the processing unit.

8.1. TYPES OF MEMORY

Memory is an essential element of a computer. Without its memory, a computer is of hardly any use. Memory plays an important role in saving and retrieving data. The performance of the computer system depends upon the size of the memory. Memory is of following types:

1. Primary Memory (Internal)
2. Secondary Memory (External)

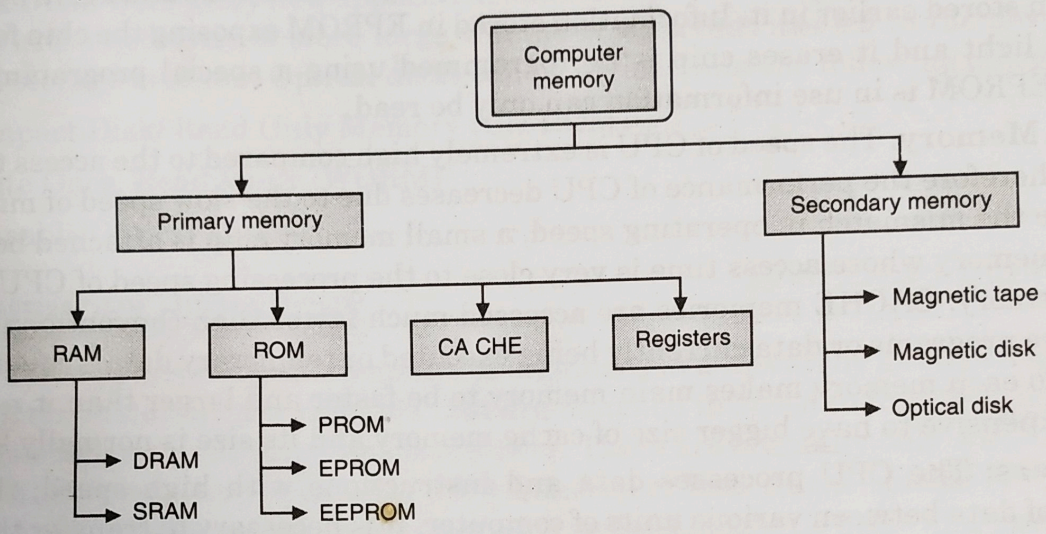

Fig. 8.1. Types of Memory

8.1.1. Primary Memory

Primary Memory is internal memory of the computer. RAM and ROM both form part of primary memory. The primary memory provides main working space to the computer. The following terms comes under primary memory of a computer are discussed below:

Random Access Memory (RAM)/Volatile: The primary storage is referred to as random access memory (RAM) because it is possible to randomly select and use any location of the memory directly store and retrieve data. It takes same time to any address of the memory as the first address. It is also called read/write memory. The storage of data and instructions inside the primary storage is temporary. It disappears from RAM as soon as the power to the computer is switched off. The memories, which lose their content on failure of power supply, are known as volatile memories .So now we can say that RAM is volatile memory.

Small Concept

Primary memory is internal memory of the computer.

Read Only Memory (ROM)/ Non-Volatile: There is another memory in computer, which is called Read Only Memory (ROM). Again it is the ICs inside the PC that form the ROM. The storage of program and data in the ROM is permanent. The ROM stores some standard processing programs supplied by the manufacturers to operate the personal computer. The ROM can only be read by the CPU but it cannot be changed. The basic input/output program is stored in the ROM that examines and initializes various equipment attached to the PC when the power switch is ON. The memories, which do not lose their content on failure of power supply, are known as non-volatile memories. ROM is non-volatile memory.

PROM: There is another type of primary memory in computer, which is called Programmable Read Only Memory (PROM). You know that it is not possible to modify or erase programs stored in ROM, but it is possible for you to store your program in PROM chip. Once the programmers' are written it cannot be changed and remain intact even if power is switched off. Therefore programs or instructions written in PROM or ROM cannot be erased or changed.

EPROM: This stands for Erasable Programmable Read Only Memory, which overcome the problem of PROM & ROM. EPROM chip can be programmed time and again by erasing the information stored earlier in it. Information stored in EPROM exposing the chip for some time ultraviolet light and it erases chip is reprogrammed using a special programming facility. When the EPROM is in use information can only be read.

Cache Memory: The speed of CPU is extremely high compared to the access time of main memory. Therefore the performance of CPU decreases due to the slow speed of main memory. To decrease the mismatch in operating speed, a small memory chip is attached between CPU and Main memory whose access time is very close to the processing speed of CPU. It is called CACHE memory. CACHE memories are accessed much faster than conventional RAM. It is used to store programs or data currently being executed or temporary data frequently used by the CPU. So each memory makes main memory to be faster and larger than it really is. It is also very expensive to have bigger size of cache memory and its size is normally kept small.

Registers: The CPU processes data and instructions with high speed; there is also movement of data between various units of computer. It is necessary to transfer the processed data with high speed. So the computer uses a number of special memory units called registers. They are not part of the main memory but they store data or information temporarily and pass it on as directed by the control unit.

8.1.2. Secondary Memory

Secondary memory is external and permanent in nature. The secondary memory is concerned with magnetic memory. Secondary memory can be stored on storage media like floppy disks, magnetic disks, magnetic tapes, This memory can also be stored optically on Optical disks - CD-ROM. The following terms comes under secondary memory of a computer are discussed below:

Magnetic Tape: Magnetic tapes are used for large computers like mainframe computers where large volume of data is stored for a longer time. In PC also you can use tapes in the form of cassettes. The cost of storing data in tapes is inexpensive. Tapes consist of magnetic materials that store data permanently. It can be 12.5 mm to 25 mm wide plastic film-type and 500 meter to 1200 meter long which is coated with magnetic material. The deck is connected to the central processor and information is fed into or read from the tape through the processor. It's similar to cassette tape recorder.

Small Concept

Secondary memory is external and permanent memory of computer.

Magnetic Disk: You might have seen the gramophone record, which is circular like a disk and coated with magnetic material. Magnetic disks used in computer are made on the same principle. It rotates with very high speed inside the computer drive. Data is stored on both the surface of the disk. Magnetic disks are most popular for direct access storage device. Each disk

consists of a number of invisible concentric circles called tracks. Information is recorded on tracks of a disk surface in the form of tiny magnetic spots. The presence of a magnetic spot represents one bit and its absence represents zero bit. The information stored in a disk can be read many times without affecting the stored data. So the reading operation is non-destructive. But if you want to write a new data, then the existing data is erased from the disk and new data is recorded. For Example-Floppy Disk.

Optical Disk: With every new application and software there is greater demand for memory capacity. It is the necessity to store large volume of data that has led to the development of optical disk storage medium. Optical disks can be divided into the following categories:

1. Compact Disk/ Read Only Memory (CD-ROM)
2. Write Once, Read Many (WORM)
3. Erasable Optical Disk

8.2. THE MEMORY HIERARCHY

Memory in a conventional digital computer is organized in a hierarchy as shown in Figure 8.1. At the top of the hierarchy are registers that are matched in speed to the CPU, but tend to be large and consume a significant amount of power. There are normally only a small number of registers in a processor, on the order of a few hundred or less. At the bottom of the hierarchy are secondary and off-line storage memories such as hard magnetic disks and magnetic tapes, in which the cost per stored bit is small in terms of money and electrical power, but the access time is very long when compared with registers. Between the registers and secondary storage are a number of other forms of memory that bridge the gap between the two.

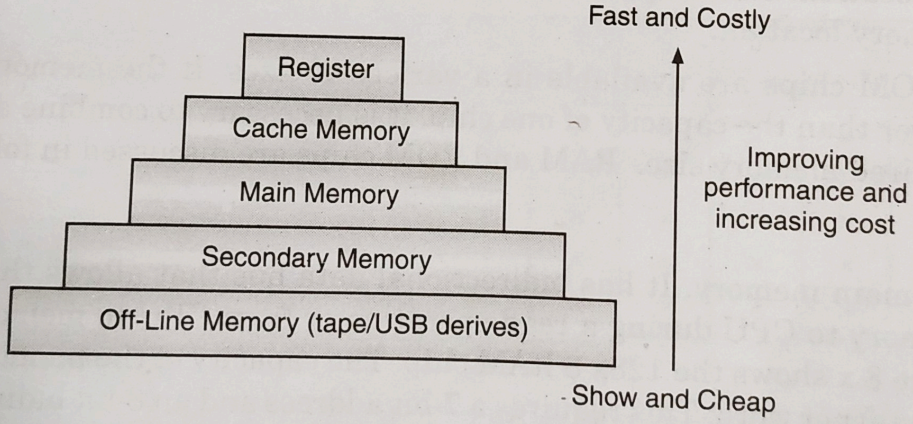

Fig. 8.2. Memory Hierarchy

Memory Type	Access Time	Cost/MB
Registers	0.5 ns	High
Cache	5-20 ns	Rs.4000
Main Memory	40-80 ns	Rs. 25
Secondary Memory	5 ms	Rs. 0.3

8.3. MAIN MEMORY

The processor communicates with the memory system over a memory interface. Computer memories are primarily built from dynamic RAM (DRAM) and static RAM (SRAM). Ideal

the computer memory system is fast, large, and cheap. In practice, a single memory only has two of these three attributes; it is either slow, small, or expensive. But computer systems can approximate the ideal by combining a fast small cheap memory and a slow large cheap memory. The fast memory stores the most commonly used data and instructions, so on average the memory system appears fast. The large memory stores the remainder of the data and instructions, so the overall capacity is large. The combination of two cheap memories is much less expensive than a single large fast memory. These principles extend to using an entire hierarchy of memories of increasing capacity and decreasing speed.

The main memory of a computer is semiconductor memory. The main memory unit of computer is basically consists of two kinds of memory:

1. **RAM :** Random access memory; which is volatile in nature.
2. **ROM :** Read only memory; which is non-volatile.

The permanent information are kept in ROM and the user space is basically in RAM. The smallest unit of information is known as bit (binary digit), and in one memory cell we can store one bit of information. 8 bit together is termed as a byte. The maximum size of main memory that can be used in any computer is determined by the addressing scheme.

Small Concept

Main memory is part of primary memory, which includes RAM and ROM.

A computer that generates 16-bit address is capable of addressing upto 216 which is equal to 64K memory location. Similarly, for 32 bit addresses, the total capacity will be 232 which is equal to 4G memory location.

RAM and ROM chips are available in a variety of sizes. If the memory needed for the computer is larger than the capacity of one chip, it is necessary to combine a number of chips to form the required memory size. RAM and ROM chips are discussed in following sections.

RAM Chip

RAM is the main memory. It has bidirectional data bus that allows the transfer of data either from memory to CPU during a read operation or from CPU to memory during a write operation. Figure 8.x shows the 128x 8 RAM chip. The capacity of the memory is 128 words of eight bits (one byte) per word. This requires a 7-bit address and an 8-bit bidirectional data bus. The read and write inputs specify the memory operation and the two chip select (CS) control inputs are for enabling the chip only when it is selected by the processor.

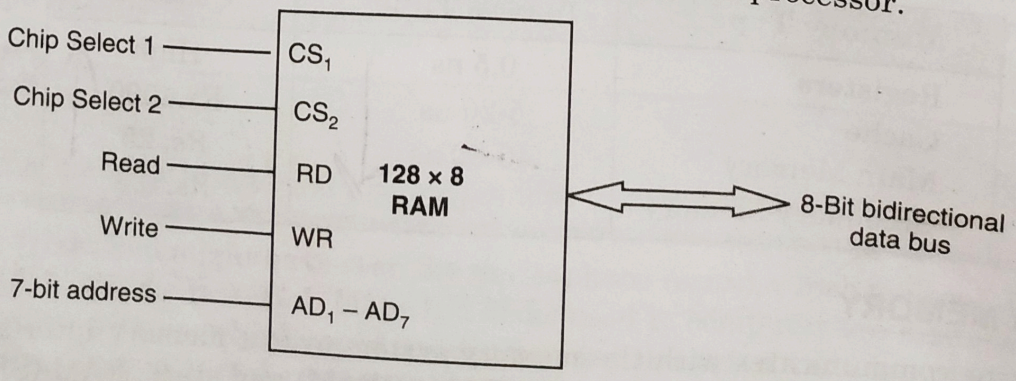

Fig. 8.3. RAM Chip

Table 8.1. Functional Table

CS_1	CS_2	RD	WR	Memory Operation	State of Data Bus
0	0	×	×	No Operation	High Impedance
0	1	×	×	No Operation	High Impedance
1	0	0	0	No Operation	High Impedance
1	0	0	1	Write Operation	Input Data to RAM
1	0	1	×	Read Operation	Output Data from RAM
1	1	×	×	No Operation	High Impedance

ROM Chip

ROM can only read, the data bus can only be in an output mode. For the same size chip, it is possible to have more bits of ROM than of RAM, because the internal binary cells in ROM occupy less space than in RAM. For this reason, the diagram specifies a 512 byte ROM, while the RAM has only 128 bytes. The nine address lines in the ROM chip specify any one of the 512 bytes stored in it.

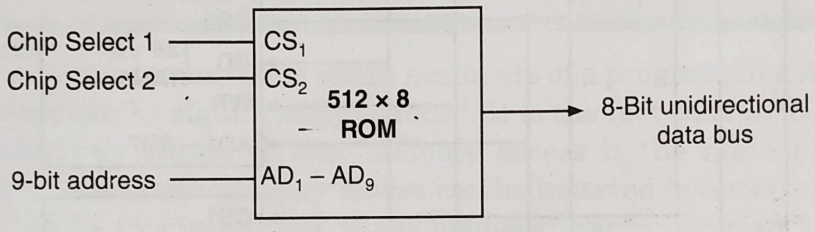

Fig. 8.4. ROM Chip

Table 8.2. Memory Address Map for Microcomputer

Device Select	Hexadecimal Address	10	9	8	7	6	5	4	3	2	1
RAM1	0000 – 007F	0	0	0	×	×	×	×	×	×	×
RAM2	0080 – 00FF	0	0	1	×	×	×	×	×	×	×
RAM3	0100 – 017F	0	1	0	×	×	×	×	×	×	×
RAM4	0180 – 01FF	0	1	1	×	×	×	×	×	×	×
ROM	0200 – 03FF	1	×	×	×	×	×	×	×	×	×

SRAM Versus DRAM :

- Both static and dynamic RAMs are volatile, that is, it will retain the information as long as power supply is applied.
- A dynamic memory cell is simpler and smaller than a static memory cell. Thus a DRAM is more dense, *i.e.*, packing density is high(more cell per unit area). DRAM is less expensive than corresponding SRAM.

Small Concept

– RAM is volatile to store temporary information.
– ROM is non-volatile to store permanent information.

- DRAM requires the supporting refresh circuitry. For larger memories, the fixed cost of the refresh circuitry is more than compensated for by the less cost of DRAM cells

- SRAM cells are generally faster than the DRAM cells. Therefore, to construct faster memory modules(like cache memory) SRAM is used.

Memory Connection to CPU

The Memory system consists of all storage devices employed in a computer system from the slow but high-capacity secondary memory to a relatively faster main memory. Fig. 8.5 shows the memory connection with CPU.

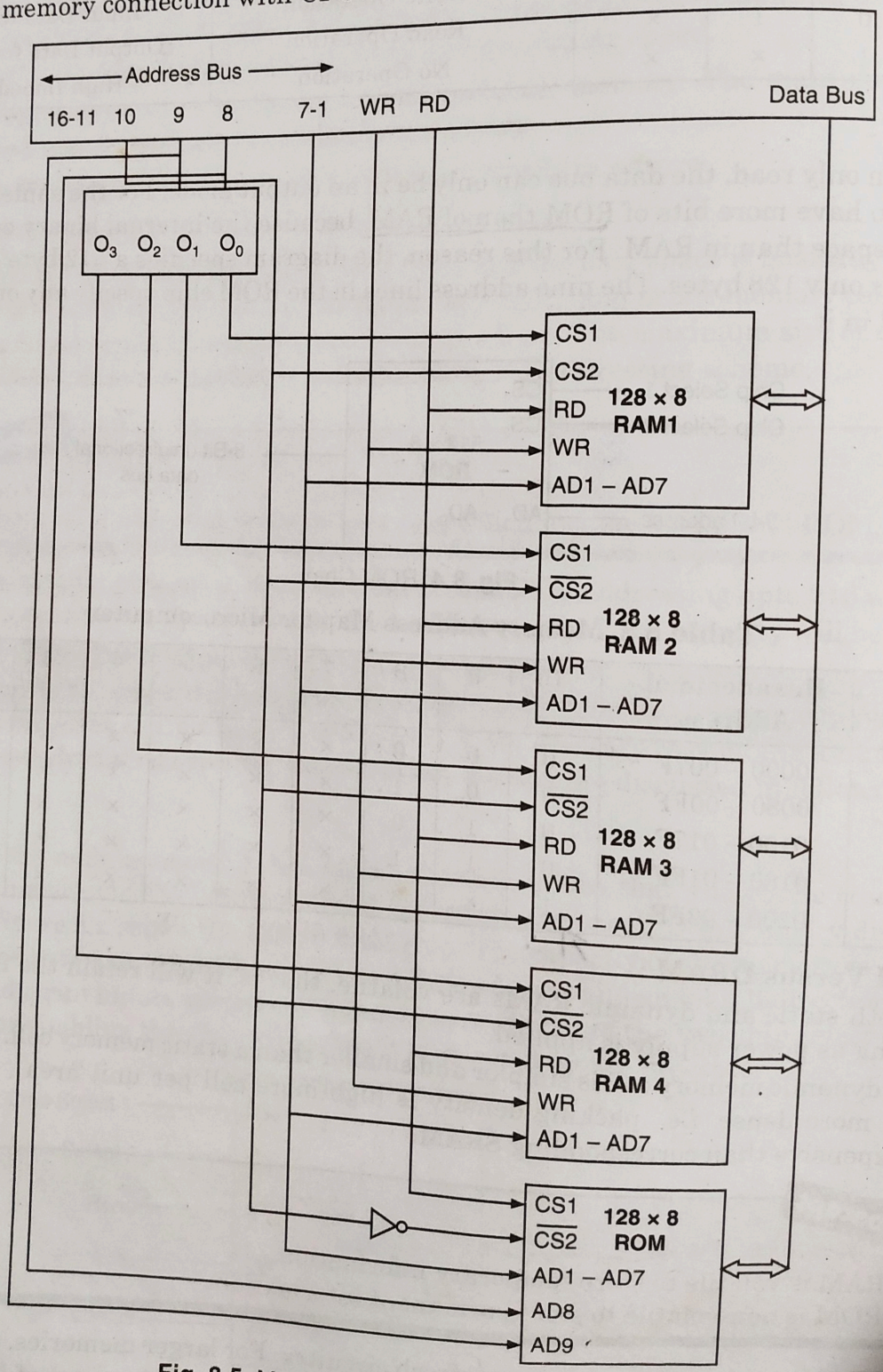

Fig. 8.5. Memory system connection to CPU

An another connection shown in figure 8.x where the configuration gives a memory capacity of 512 bytes of RAM and 512 bytes of ROM. The particular RAM chip selected is determined from lines 8 and 9 in the address bus. This is done through a 2 x 4 decoder whose outputs goes to the CS1 inputs in each RAM chip. The selection between RAM and ROM is achieved through bus line 10.

8.4. CACHE MEMORY

Analysis of large number of programs has shown that a number of instructions are executed repeatedly. This may be in the form of a simple loops, nested loops, or a few procedures that repeatedly call each other. It is observed that many instructions in each of a few localized areas of the program are repeatedly executed, while the remainder of the program is accessed relatively less. This phenomenon is referred to as locality of reference.

Small Concept

Cache memory is smaller than main memory to boost the speed of CPU for data access.

Now, if it can be arranged to have the active segments of a program in a fast memory, then the total execution time can be significantly reduced. It is the fact that CPU is a faster device and memory is a relatively slower device. Memory access is the main bottleneck for the performance efficiency. If a faster memory device can be inserted between main memory and CPU, the efficiency can be increased. The faster memory that is inserted between CPU and Main Memory is termed as Cache memory. To make this arrangement effective, the cache must be considerably faster than the main memory, and typically it is 5 to 10 time faster than the main memory. This approach is more economical than the use of fast memory device to implement the entire main memory. This is also a feasible due to the locality of reference that is present in most of the program, which reduces the frequent data transfer between main memory and cache memory. The inclusion of cache memory between CPU and main memory is shown in Fig. 8.6.

Operation of Cache Memory

The memory control circuitry is designed to take advantage of the property of locality of reference. Some assumptions are made while designing the memory control circuitry:

1. The CPU does not need to know explicitly about the existence of the cache.
2. The CPU simply makes Read and Write request. The nature of these two operations are same whether cache is present or not.
3. The address generated by the CPU always refer to location of main memory.
4. The memory access control circuitry determines whether or not the requested word currently exists in the cache.

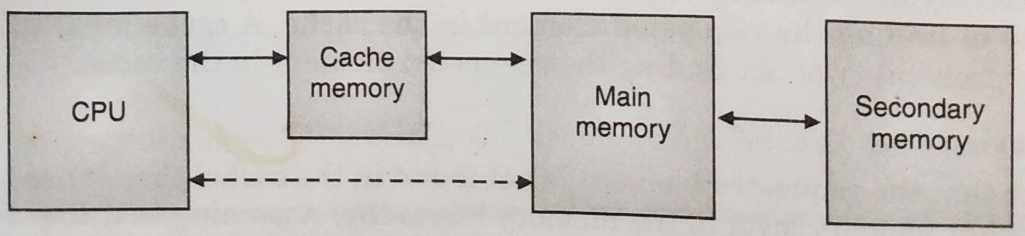

Fig. 8.6. Cache memory between CPU and the main memory

When a Read request is received from the CPU, the contents of a block of memory words containing the location specified are transferred into the cache. When any of the locations in this block is referenced by the program, its contents are read directly from the cache.

Consider the case where the addressed word is not in the cache and the operation is a read. First the block of words is brought to the cache and then the requested word is forwarded to the CPU. But it can be forwarded to the CPU as soon as it is available to the cache, instead of the whole block to be loaded in the cache. This is called load through, and there is some scope to save time while

When the cache is full and a memory word is referenced that is not in the cache, a decision must be made as to which block should be removed from the cache to create space to bring the new block to the cache that contains the referenced word. Replacement algorithms are used to make the proper selection of block that must be replaced by the new one.

When a write request is received from the CPU, there are two ways that the system can proceed. In the first case, the cache location and the main memory location are updated simultaneously. This is called the store through method or write through method.

The alternative is to update the cache location only. During replacement time, the cache block will be written back to the main memory. This method is called write back method. If there is no new write operation in the cache block, it is not required to write back the cache block in the main memory. This information can be kept with the help of an associated bit. This bit it set while there is a write operation in the cache block. During replacement, it checks this bit, if it is set, then write back the cache block in main memory otherwise not. This bit is known as dirty bit. If the bit gets dirty (set to one), writing to main memory is required.

Small Concept

CPU always generate the address of main memory location but circuit determines the location in the cache first.

The write through method is simpler, but it results in unnecessary write operations in the main memory when a given cache word is updated a number of times during its cache residency period.

During a write operation, if the address word is not in the cache, the information is written directly into the main memory. A write operation normally refers to the location of data and the property of locality of reference is not as pronounced in accessing data when write operation is involved. Therefore, it is not advantageous to bring the data block to the cache when there a write operation, and the addressed word is not present in cache.

Therefore, when the processor makes a request for a memory reference, the request is first sought in the cache. If the request corresponds to an element that is currently residing in the cache, we call that a cache hit. On the other hand, if the request corresponds to an element that is not currently in the cache, we call that a cache miss. A cache hit ratio, hc, is defined as the probability of finding the requested element in the cache. A cache miss ratio (1 - hc) is defined as the probability of not finding the requested element in the cache.

Memory Interleaving

In the case that the requested element is not found in the cache, then it has to be brought from a subsequent memory level in the memory hierarchy. Assuming that the element exists in the next memory level, that is, the main memory, then it has to be brought and placed in the

cache. In expectation that the next requested element will be residing in the neighbouring locality of the current requested element (spatial locality), then upon a cache miss what is actually brought to the main memory is a block of elements that contains the requested element. The advantage of transferring a block from the main memory to the cache will be most visible if it could be possible to transfer such a block using one main memory access time. Such a possibility could be achieved by increasing the rate at which information can be transferred between the main memory and the cache. One possible technique that is used to increase the bandwidth is memory interleaving. To achieve best results, we can assume that the block brought from the main memory to the cache, upon a cache miss, consists of elements that are stored in different memory modules, that is, whereby consecutive memory addresses are stored in successive memory modules. Figure 8.7 illustrates the simple case of a main memory consisting of eight memory modules. It is assumed in this case that the block consists of 8 bytes.

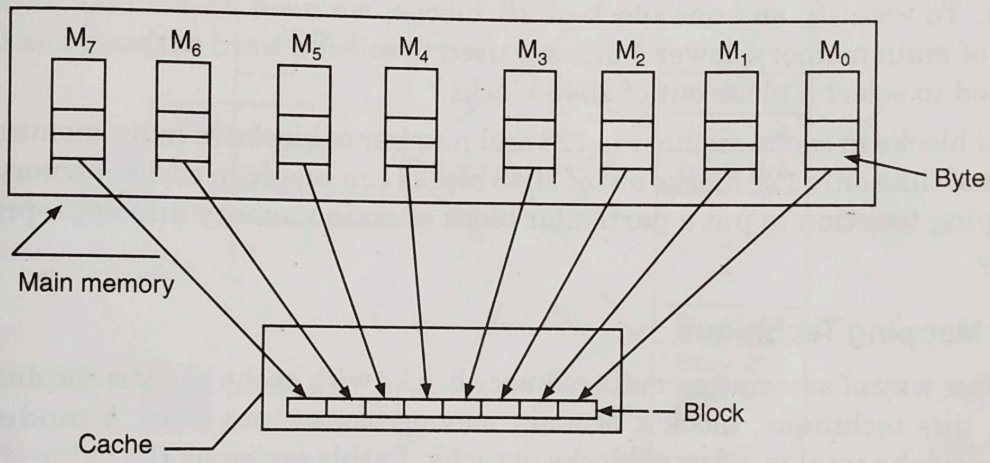

Fig. 8.7. Memory interleaving using eight modules

8.5. CACHE-MAPPING FUNCTIONS

The mapping functions are used to map a particular block of main memory to a particular block of cache. This mapping function is used to transfer the block from main memory to cache memory. Three different mapping functions are available:

- **Direct mapping:** A particular block of main memory can be brought to a particular block of cache memory. So, it is not flexible.
- **Associative mapping:** In this mapping function, any block of Main memory can potentially reside in any cache block position. This is much more flexible mapping method.
- **Block-set-associative mapping:** In this method, blocks of cache are grouped into sets, and the mapping allows a block of main memory to reside in any block of a specific set. From the flexibility point of view, it is in between to the other two methods.

Small Concept

Several types of mapping functions are used to transfer the block from main memory to cache memory.

All these three mapping methods are explained with the help of an example, which has been retrieved from the book Fundamentals of Computer Organization and Architecture by Mostafa abd-el-barr and Hesham el-rewini.

Consider a cache of 4096 (4K) words with a block size of 32 words. Therefore, the cache is organized as 128 blocks. For 4K words, required address lines are 12 bits. To select one of the block out of 128 blocks, we need 7 bits of address lines and to select one word out of 32 words, we need 5 bits of address lines. So the total 12 bits of address is divided for two groups, lower 5 bits are used to select a word within a block, and higher 7 bits of address are used to select any block of cache memory.

Let us consider a main memory system consisting 64K words. The size of address bus is 16 bits. Since the block size of cache is 32 words, so the main memory is also organized as block size of 32 words. Therefore, the total number of blocks in main memory is 2048 (2K x 32 words = 64K words). To identify any one block of 2K blocks, we need 11 address lines. Out of 16 address lines of main memory, lower 5 bits are used to select a word within a block and higher 11 bits are used to select a block out of 2048 blocks.

Number of blocks in cache memory is 128 and number of blocks in main memory is 2048, so at any instant of time only 128 blocks out of 2048 blocks can reside in cache memory. Therefore, we need mapping function to put a particular block of main memory into appropriate block of cache memory.

8.5.1. Direct Mapping Technique

The simplest way of associating main memory blocks with cache block is the direct mapping technique. In this technique, block k of main memory maps into block ***k* modulo *m*** of the cache, where *m* is the total number of blocks in cache. In this example, the value of ***m*** is 128. In direct mapping technique, one particular block of main memory can be transferred to a particular block of cache which is derived by the modulo function.

Since more than one main memory block is mapped onto a given cache block position, contention may arise for that position. This situation may occurs even when the cache is not full. Contention is resolved by allowing the new block to overwrite the currently resident block. So the replacement algorithm is trivial.

The detail operation of direct mapping technique is as follows:

The main memory address is divided into three fields. The field size depends on the memory capacity and the block size of cache. In this example, the lower 5 bits of address is used to identify a word within a block. Next 7 bits are used to select a block out of 128 blocks (which is the capacity of the cache). The remaining 4 bits are used as a TAG to identify the proper block of main memory that is mapped to cache.

Small Concept

Each mappings technique includes three fields of memory address:
(*i*) TAG, (*ii*) Block, (*iii*) Word.

When a new block is first brought into the cache, the high order 4 bits of the main memory address are stored in four TAG bits associated with its location in the cache. When the CPU generates a memory request, the 7-bit block address determines the corresponding cache blo

The TAG field of that block is compared to the TAG field of the address. If they match, the desired word specified by the low-order 5 bits of the address is in that block of the cache.

If there is no match, the required word must be accessed from the main memory, that is, the contents of that block of the cache is replaced by the new block that is specified by the new address generated by the CPU and correspondingly the TAG bit will also be changed by the high order 4 bits of the address. The whole arrangement for direct mapping technique is shown in the Fig. 8.8.

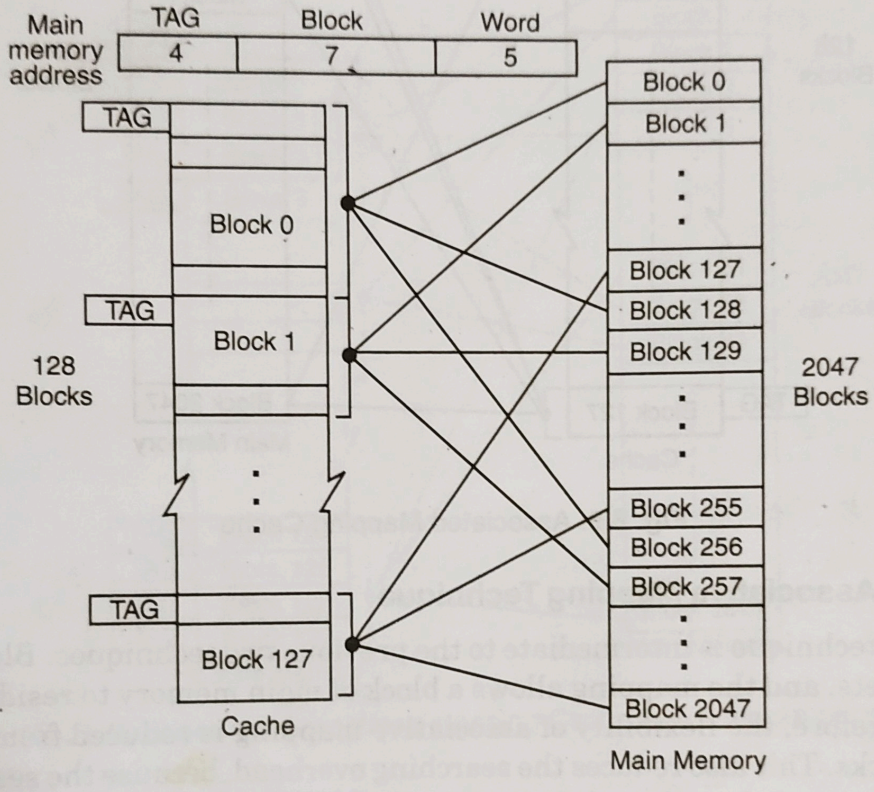

Fig. 8.8. Direct-mapping cache

8.5.2. Associated Mapping Technique

In the associative mapping technique, a main memory block can potentially reside in any cache block position. In this case, the main memory address is divided into two groups, low-order bits identifies the location of a word within a block and high-order bits identifies the block. In the example here, 11 bits are required to identify a main memory block when it is resident in the cache , high-order 11 bits are used as TAG bits and low-order 5 bits are used to identify a word within a block. The TAG bits of an address received from the CPU must be compared to the TAG bits of each block of the cache to see if the desired block is present.

In the associative mapping, any block of main memory can go to any block of cache, so it has got the complete flexibility and we have to use proper replacement policy to replace a block from cache if the currently accessed block of main memory is not present in cache. It might not be practical to use this complete flexibility of associative mapping technique due to searching overhead, because the TAG field of main memory address has to be compared with the TAG field of all the cache block. In this example, there are 128 blocks in cache and the size of TAG is 11 bits. The whole arrangement of Associative Mapping Technique is shown in the Fig. 8.9.

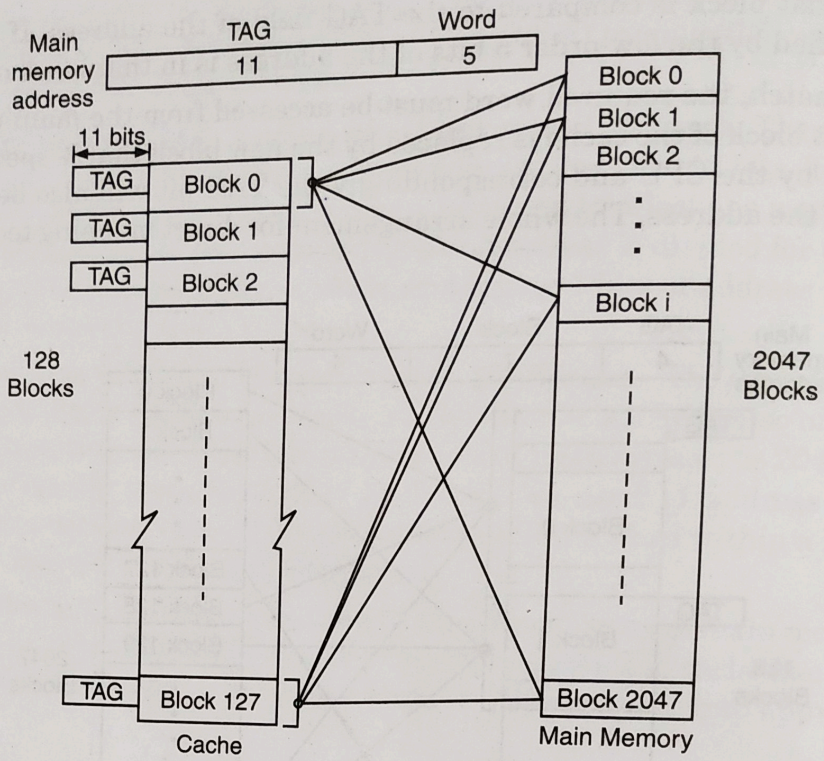

Fig. 8.9. Associated Mapping Cache

8.5.3. Block-Set-Associative Mapping Technique

This mapping technique is intermediate to the previous two techniques. Blocks of the cache are grouped into sets, and the mapping allows a block of main memory to reside in any block of a specific set. Therefore, the flexibility of associative mapping is reduced from full freedom to a set of specific blocks. This also reduces the searching overhead, because the search is restricted to number of sets, instead of number of blocks. Also the contention problem of the direct mapping is eased by having a few choices for block replacement.

Consider the same cache memory and main memory organization of the previous example. Organize the cache with 4 blocks in each set. The TAG field of associative mapping technique is divided into two groups, one is termed as SET bit and the second one is termed as TAG bit. Each set contains 4 blocks, total number of set is 32. The main memory address is grouped into three parts: low-order 5 bits are used to identifies a word within a block. Since there are total 32 sets present, next 5 bits are used to identify the set. High-order 6 bits are used as TAG bits.

The 5-bit set field of the address determines which set of the cache might contain the desired block. This is similar to direct mapping technique, in case of direct mapping, it looks for block, but in case of block-set-associative mapping, it looks for set. The TAG field of the address must then be compared with the TAGs of the four blocks of that set. If a match occurs, then the block is present in the cache; otherwise the block containing the addressed word must be brought to the cache. This block will potentially come to the corresponding set only. Since there are four blocks in the set, we have to choose appropriately which block to be replaced if all the blocks are occupied. Since the search is restricted to four block only, so the searching complexity is reduced. The whole arrangement of block-set-associative mapping technique is shown in the Fig. 8.10.

It is clear that if we increase the number of blocks per set, then the number of bits in SET field is reduced. Due to the increase of blocks per set, complexity of search is also increased. The extreme condition of 128 blocks per set requires no set bits and corresponds to the fully associative mapping technique with 11 TAG bits. The other extreme of one block per set is the direct mapping method.

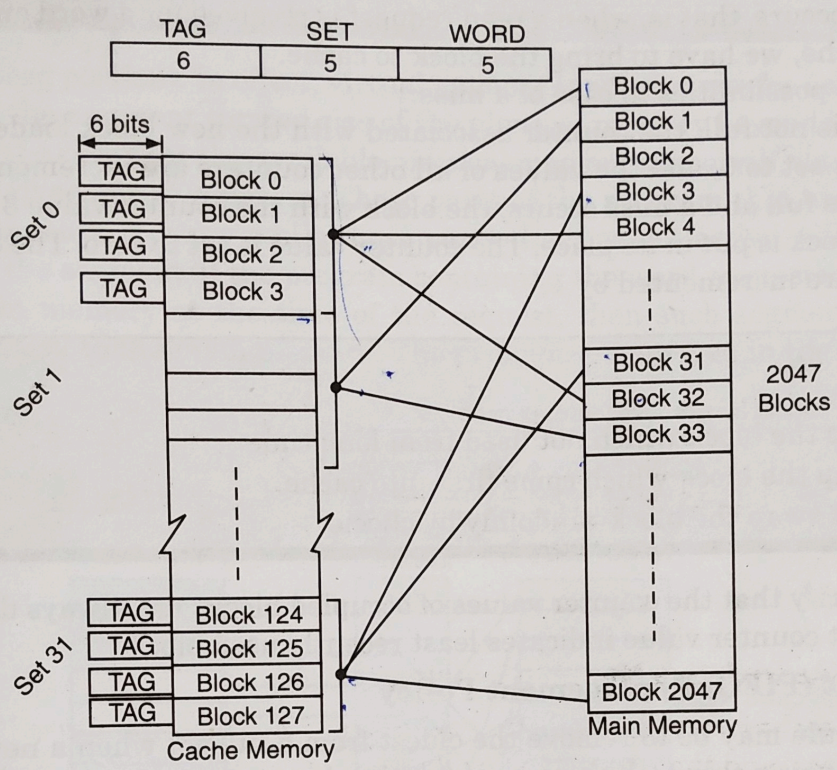

Fig. 8.10. Block-set Associated mapping Cache with 4 blocks per set

8.6. REPLACEMENT ALGORITHMS

When a new block must be brought into the cache and all the positions that it may occupy are full, a decision must be made as to which of the old blocks is to be overwritten. In general, a policy is required to keep the block in cache when they are likely to be referenced in near future. However, it is not easy to determine directly which of the block in the cache are about to be referenced. The property of locality of reference gives some clue to design good replacement policy.

Least Recently Used (LRU) Replacement policy: Since program usually stay in localized areas for reasonable periods of time, it can be assumed that there is a high probability that blocks which have been referenced recently will also be referenced in the near future. Therefore, when a block is to be overwritten, it is a good decision to overwrite the one that has gone for longest time without being referenced. This is defined as the least recently used (LRU) block. Keeping track of LRU block must be done as computation proceeds.

Small Concept

Replacement policies are used when cache is required to swap the block due to space unavailability.

Consider a specific example of a four-block set. It is required to track the LRU block of this four-block set. A 2-bit counter may be used for each block.

When a hit occurs, that is, when a read request is received for a word that is in the cache, the counter of the block that is referenced is set to 0. All counters which values originally lower than the referenced one are incremented by 1 and all other counters remain unchanged.

When a miss occurs, that is, when a read request is received for a word and the word is not present in the cache, we have to bring the block to cache.

There are two possibilities in case of a miss:

1. If the set is not full, the counter associated with the new block loaded from the main memory is set to 0, and the values of all other counters are incremented by 1.
2. If the set is full and a miss occurs, the block with the counter value 3 is removed , and the new block is put in its place. The counter value is set to zero. The other three block counters are incremented by 1.

Small Concept

- LRU swap the block which not used from long time.
- FIFO swap the block which come first into cache.
- RANDOM swap the block randomly by chocie.

It is easy to verify that the counter values of occupied blocks are always distinct. Also it is trivial that highest counter value indicates least recently used block.

First In First Out (FIFO) Replacement Policy

A reasonable rule may be to remove the oldest from a full set when a new block must be brought in. While using this technique, no updation is required when a hit occurs. When a miss occurs and the set is not full, the new block is put into an empty block and the counter values of the occupied block will be increment by one. When a miss occurs and the set is full, the block with highest counter value is replaced by new block and counter is set to 0, counter value of all other blocks of that set is incremented by 1. The overhead of the policy is less, since no updation is required during hit.

Random Replacement Policy

The simplest algorithm is to choose the block to be overwritten at random. Interestingly enough, this simple algorithm has been found to be very effective in practice. According to this replacement policy, a page is selected randomly for replacement. This is the simplest replacement mechanism. It can be implemented using a pseudo-random number generator that generates numbers that correspond to all possible page frames. At the time of replacement the random number generated will indicate the page frame that must be replaced. Although simple, this technique may not result in efficient use of the main memory, that is, a low hit ratio h. Random replacement has been used in the Intel i860 family of RISC processor.

8.7. VIRTUAL MEMORY

The hard disk provides an illusion of more capacity than actually exists in the main memory. It is thus called virtual memory. Like books in the basement, data in virtual memory takes a long time to access. Main memory, also called physical memory, holds a subset of the virtual memory. Hence, the main memory can be viewed as a cache for the most commonly used data from the hard disk.

The concept of virtual memory is similar to the cache memory principle. A virtual memory is a concept to optimize the use of the main memory (the higher speed portion) with the hard

Small Concept

Virtual memory is not a memory, it is a concept.

disk (the lower speed portion). In effect, virtual memory is a technique for using the secondary storage to extend the apparent limited size of the physical memory beyond its actual physical size. It is usually the case that the available physical memory space will not be enough to host all the parts of a given active program. Those parts of the program that are currently active are brought to the main memory while those parts that are not active will be stored on the magnetic disk. If the segment of the program containing the word requested by the processor is not in the main memory at the time of the request, then such segment will have to be brought from the disk to the main memory. The principles employed in the virtual memory

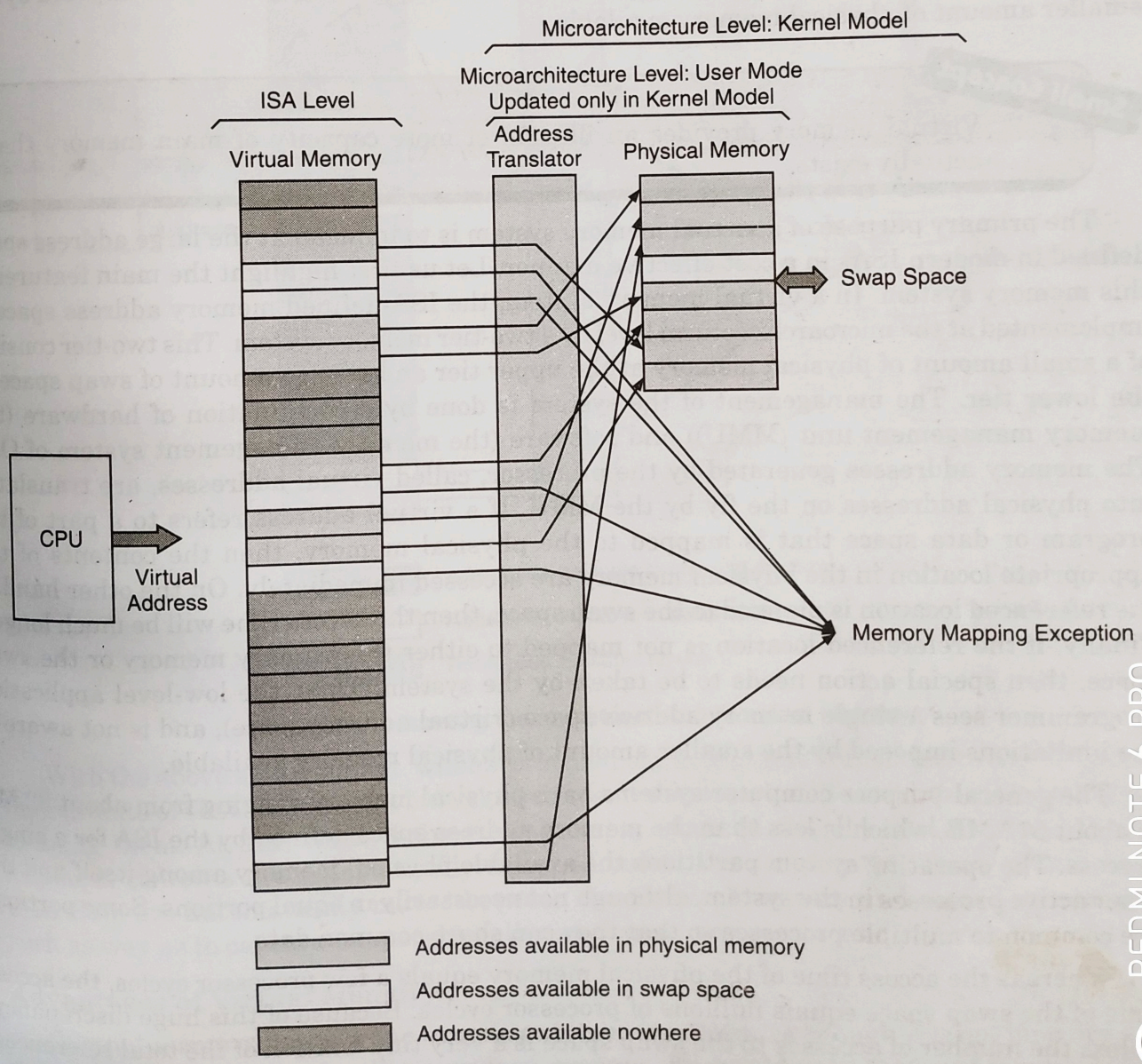

Fig. 8.11. Illustrating the concept of virtual memory

design are the same as those employed in the cache memory. The most relevant principle is that of keeping active segments in the high-speed main memory and moving inactive segments back to the hard disk.

In virtual memory scheme, the ISA defined memory address space is implemented at the microarchitectural level by a two-tier memory system, consisting of a small amount of physical memory at the upper tier and a large amount of swap space at the lower tier, managed by the operating system. The memory addresses generated by the processor, called virtual addresses, are translated into physical addresses on the fly by a Memory Management Unit (MMU). If a virtual address refers to a part of the program or data space that is mapped to the physical memory, then the contents of the appropriate location in the physical memory are accessed immediately. On the other hand, if the referenced location is mapped to the swap space, then an exception is generated to turn control over to the operating system, which gets the requested word after a much longer time. Finally, if the referenced location is not mapped to either the physical memory or the swap space, then special action needs to be taken by the operating system. The low-level application programmer, is not aware of the limitations imposed by the smaller amount of physical memory available.

Small Concept

Virtual memory provides an illusion of more capacity of main memory than actually exists.

The primary purpose of a virtual memory system is to implement the large address space defined in modern ISAs in a cost-effective manner. Let us first highlight the main features of this memory system. In a virtual memory system, the ISA-defined memory address space is implemented at the microarchitectural level by a two-tier memory system. This two-tier consists of a small amount of physical memory at the upper tier and a large amount of swap space at the lower tier. The management of the system is done by a combination of hardware (the memory management unit (MMU)) and software (the memory management system of OS). The memory addresses generated by the processor, called virtual addresses, are translated into physical addresses on the fly by the MMU. If a virtual address refers to a part of the program or data space that is mapped to the physical memory, then the contents of the appropriate location in the physical memory are accessed immediately. On the other hand, if the referenced location is mapped to the swap space, then the access time will be much longer. Finally, if the referenced location is not mapped to either the primary memory or the swap space, then special action needs to be taken by the system. Thus, the low-level application programmer sees a single memory address space (virtual address space), and is not aware of the limitations imposed by the smaller amount of physical memory available.

The general-purpose computer systems have physical memory ranging from about 32 MB to about 512 MB, which is less than the memory address space defined by the ISA for a single process. The operating system partitions the available physical memory among itself and the other active processes in the system, although not necessarily in equal portions. Some portions are common to multiple processes so that they can share common data.

Whereas the access time of the physical memory equals a few processor cycles, the access time of the swap space equals millions of processor cycles. Because of this huge discrepancy, unless the number of accesses to the swap space is a very tiny fraction of the total references, the performance of the system will be really bad. In order to obtain good performance, the

mapping (from virtual addresses to physical addresses) is dynamically adapted in such a manner that the locations that were frequently accessed in the recent past (and are therefore expected to be accessed again in the near future) are mapped to the physical memory. The infrequently accessed locations are mapped to the swap space.

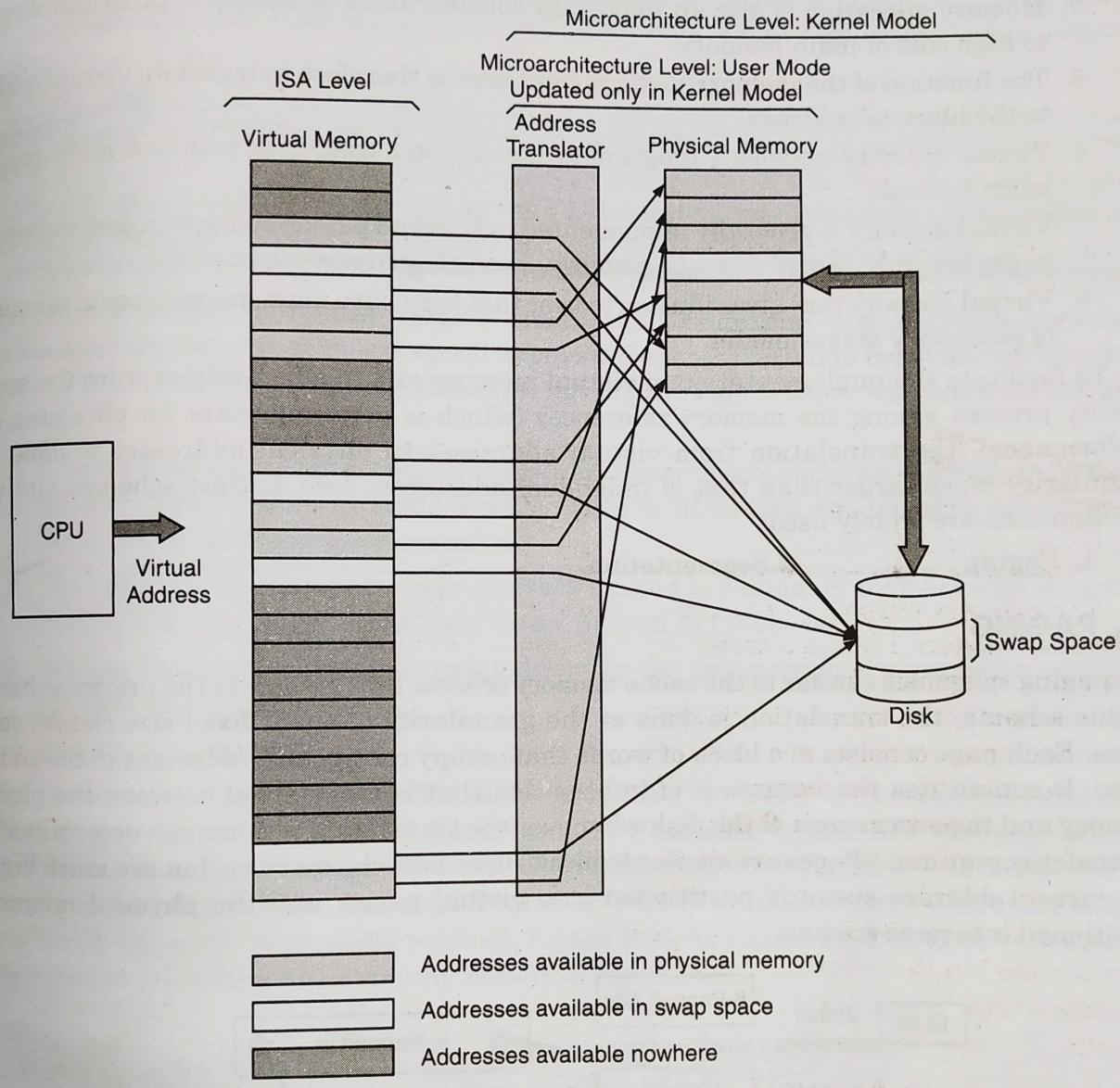

Fig. 8.12. Illustrating the implementation of virtual memory

With the above arrangement, whenever a memory access gets translated to the swap space, the contents of that location are brought into a suitable location in the physical memory prior to using them. At that time, if there is no empty space in the physical memory, then the displaced contents are automatically swapped out to the swap space. Thus, programs and their data are automatically moved between the physical memory and secondary storage in such as way as to capture temporal locality among memory accesses.

Advantages of Virtual Memory

Virtual memory permits the user to construct program as though a large memory space were available, equal to totality auxiliary memory. Each address that is referenced by CPU

goes through an address mapping from so called virtual address to physical address main memory.

There are following advantages with virtual memory:

1. Virtual memory helps in improving the processor utilization.
2. Memory allocation is also an important consideration in computer programming due to high cost of main memory.
3. The function of the memory management unit is therefore to translate virtual address to the physical address.
4. Virtual memory enables a program to execute on a computer with less main memory when it needs.
5. Virtual memory is generally implemented by demand paging concept In demand paging, pages are only loaded to main memory when they are required
6. Virtual memory that gives illusion to user that they have main memory equal to capacity of secondary stages media.

To facilitate the implementation of virtual memory and to effectively capture the spatial locality present among the memory references (which is very important for obtaining good performance), the translation from virtual addresses to physical addresses is done at a granularity much larger than that of individual addresses. Two distinct schemes and their combinations are widely used:

1. Paging 2.Segmentation.

8.8. PAGING

The paging scheme is similar to the cache memory scheme that we saw in the previous chapter. In this scheme, the translation is done at the granularity of equal fixed-size chunks called pages. Each page consists of a block of words that occupy contiguous addresses in the address space. It constitutes the basic unit of information that is transferred between the physical memory and the swap space in the disk whenever the translation mechanism determines that a transfer is required. (Pages are similar to blocks used in cache memory, but are much bigger.) The virtual address space is partitioned into virtual pages, and the physical memory is partitioned into page frames.

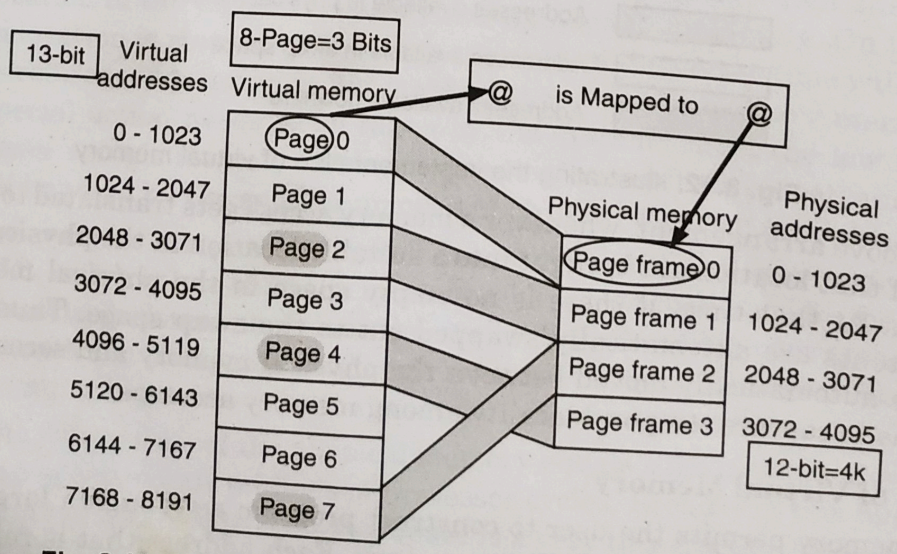

Fig. 8.13. A mapping between a virtual and a physical memory.

Paging is a form of automatic overlaying that is managed by the operating system. The address space is partitioned into equal sized blocks, called pages. Pages are normally an integral power of two in size such as 210 = 1024 bytes. Paging makes the physical memory appear larger than it truly is by mapping the physical memory address space to some portion of the virtual memory address space, which is normally stored on a disk. An illustration of a virtual memory mapping scheme is shown in Fig. 8.13. Eight virtual pages are mapped to four physical page frames.

Small Concept

Paging is managed by operating system automatically to implement the concept of virtual memory.

An implementation of virtual memory must handle references that are made outside of the portion of virtual space that is mapped to physical space. The following sequence of events is typical when a referenced virtual location is not in physical memory, which is referred to as a page fault:

1. A page frame is identified to be overwritten. The contents of the page frame are written to secondary memory if changes were made to it, so that the changes are recorded before the page frame is overwritten.
2. The virtual page that we want to access is located in secondary memory and is written into physical memory, in the page frame located in (1) above.
3. The page table (see below) is updated to map the new section of virtual memory onto the physical memory.
4. Execution continues.

For the virtual memory shown in Fig. 8.13, there are 213 = 8192 virtual locations and so an executing program must generate 13-bit addresses, which are interpreted as a 3-bit page number and a 10-bit offset within the page. Given the 3-bit page number, we need to find out where the page is: it is either in one of the four page frames, or it is in secondary memory. In order to keep track of which pages are in physical memory, a page table is maintained, as illustrated in Fig. 8.14, which corresponds to the mapping shown in Fig. 8.13.

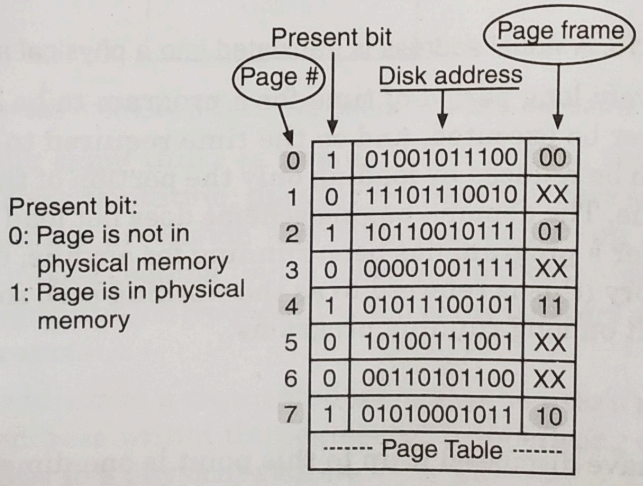

Fig. 8.14. A page table for a virtual memory

The page table has as many entries as there are virtual pages. The present bit indicates whether or not the corresponding page is in physical memory. The disk address field is a pointer to the location that the corresponding page can be found on a disk unit. The operating system normally manages the disk accesses, and so the page table only needs to maintain the disk addresses that the operating system assigns to blocks when the system starts up. The disk addresses normally do not change during the course of computation. The page frame field indicates which physical page frame holds a virtual page, if the page is in physical memory. For pages that are not in physical memory, the page frame fields are invalid, and so they are marked with "xx" in Fig. 8.14.

In order to translate a virtual address to a physical address, we take two page frame bits from the page table and append them to the left of the 10-bit offset, which produces the physical address for the referenced word. Consider the situation shown in Fig. 8.15, in which a reference is made to virtual address 1001101000101. The three leftmost bits of the virtual address (100) identify the page. The bit pattern that appears in the page frame field (11) is appended to the left of the 10-bit offset (1101000101), and the resulting address (111101000101) indicates which physical memory address holds the referenced word.

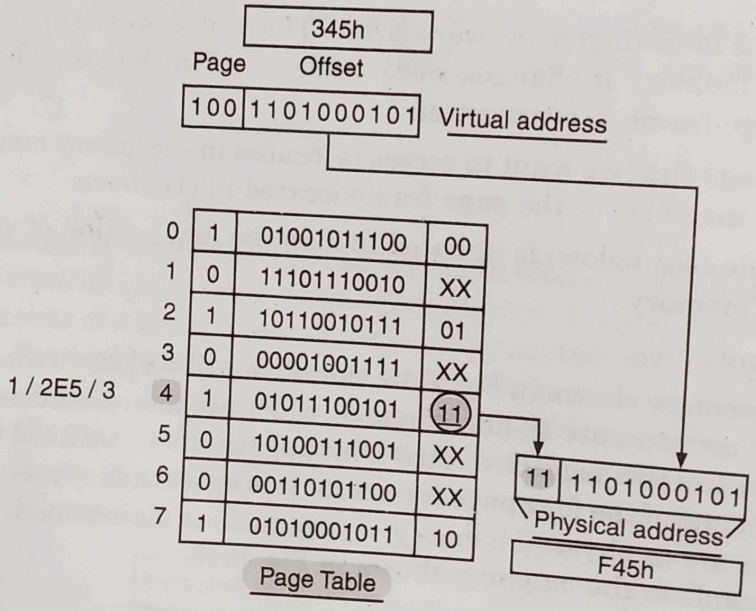

Fig. 8.15. A virtual address is translated into a physical address

It may take a relatively long period of time for a program to be loaded into memory. The entire program may never be executed, and so the time required to load the program from a disk into the memory can be reduced by loading only the portion of the program that is needed for a given interval of time. The demand paging scheme does not load a page into memory until there is a page fault. After a program has been running for a while, only the pages being used will be in physical memory (this is referred to as the working set), so demand paging does not have a significant impact on long running programs.

8.9. SEGMENTATION

Virtual memory as we have discussed it up to this point is one-dimensional in the sense that addresses grow either up or down. Segmentation divides the address space into segments

which may be of arbitrary size. Each segment is its own one-dimensional address space. This allows tables, stacks, and other data structures to be maintained as logical entities that grow without bumping into each other. Segmentation allows for protection, so that a segment may be specified as "read only" to prevent changes, or "execute only" to prevent unauthorized copying. This also protects users from trying to write data into instruction areas.

Small Concept

Paging divides address space into fixed size pages while segmentation divides into the segments of arbitrary size.

When segmentation is used with virtual memory, the size of each segment's address space can be very large, and so the physical memory devoted to each segment is not committed until needed.

Figure 8.16. illustrates a segmented memory. The executable code for a word processing program is loaded into Segment #0. This segment is marked as "execute only" and is thus protected from writing. Segment #1 is used for the data space for user #0, and is marked as "read/write" for user #0, so that no other user can have access to this area. Segment #2 is used for the data space for user #1, and is marked as "read/write" for user #1. The same word processor can be used by both user #0 and user #1, in which case the code in segment #0 is shared, but each user has a separate data segment.

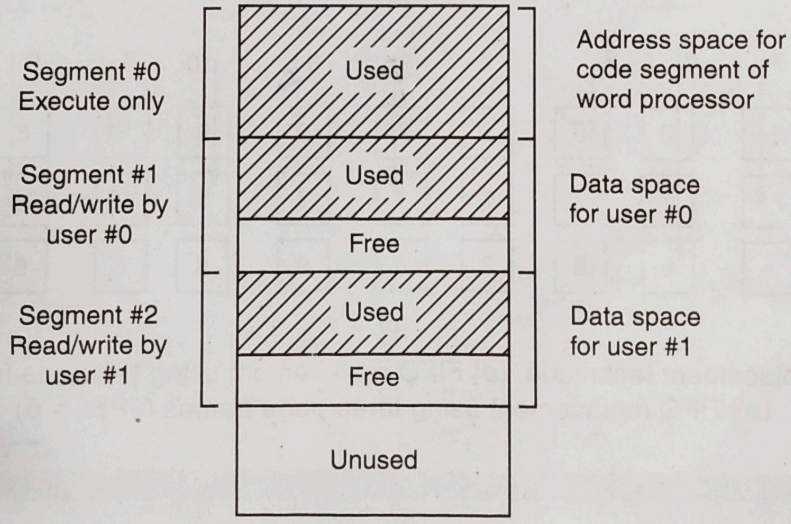

Fig. 8.16. A segmented memory allows two users to share the same word processor

Segmentation is not the same thing as paging. With paging, the user does not see the automatic overlaying. With segmentation, the user is aware of where segment boundaries are. The operating system manages protection and mapping, and so an ordinary user does not normally need to deal with bookkeeping, but a more sophisticated user such as a computer programmer may see the segmentation frequently when array pointers are pushed past segment boundaries in errant programs.

In order to specify an address in a segmented memory, the user's program must specify a segment number and an address within the segment. The operating system then translates the user's segmented address to a physical address.

8.10. NUMERICAL EXAMPLES

EXAMPLE 8.1.

Consider the following reference string of pages made by a processor:

6, 7, 8, 9, 7, 8, 9, 10, 8, 9, 10.

Using FIFO page replacement policy show the page faults for following two cases:

(*a*) the number of page frames allocated in the main memory is TWO and

(*b*) the number of page frames allocated are THREE.

SOLUTION: Figure 8.17 illustrates a trace of the reference string for the two cases. As can be seen from the figure, when the number of page frames is TWO, there were 11 page faults (these are shown in bold in the figure). When the number of page frames is increased to THREE, the number of page faults was reduced to five. Since five pages are referenced, this is the optimum condition. The FIFO policy results in the best (minimum) page faults when the reference string is in strict order of increasing page number references.

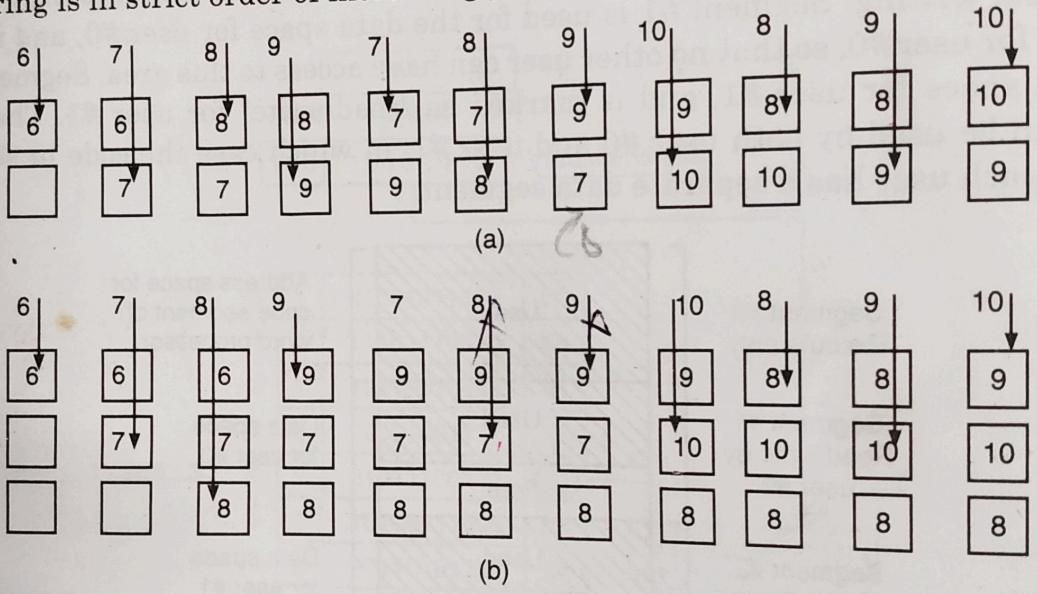

Fig. 8.17. FIFO replacement technique. (*a*) FIFO replacement using two page frames (#PFs = 11), (*b*) FIFO replacement using three page frames (#PFs = 5)

EXAMPLE 8.2.

Consider the following reference string of pages made by a processor:

4, 7, 5, 7, 6, 7, 10, 4, 8, 5, 8, 6, 8, 11, 4, 9, 5, 9, 6, 9, 12, 4, 7, 5, 7.

Assume that the number of page frames allocated in the main memory is FOUR. Compute the number of page faults generated using LRU page replacement policy.

SOLUTION: The trace of the main memory contents is shown in Fig. 8.18. Number of page faults = 17. In presenting the LRU, we have a particular implementation, called stack-based LRU. In this implementation, the most recently accessed page is now represented by the top page rectangle. The rectangles do not represent specific page frames as they did in the FIFO diagram. Thus, each reference generating a page fault is now on the top row. It should be noted that as more pages are allotted to the program the page references in each row do not change. Only the number of page faults changes. This will make the set of pages in memory for

n page frames be a subset of the set of pages for n + 1 page frames. In fact, the diagram could be considered a STACK data structure with the depth of the stack representing the number of page frames. If a page is not on the stack (i.e., is found at a depth greater than the number of page frames), then a page fault occurs.

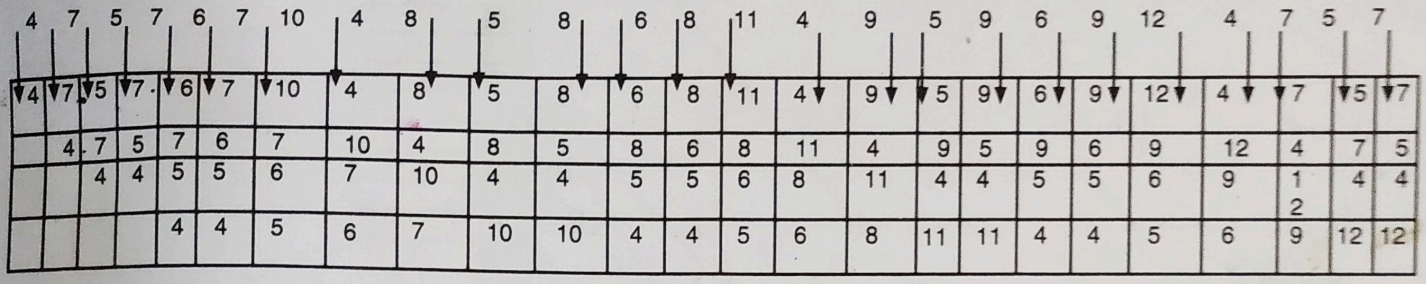

Fig. 8.18. LRU replacement technique

EXAMPLE 8.3.

Consider the case of a two-dimensional 8x8 array A. The array is stored in row-major order. For THREE page frames, compute how many page faults are generated by the following array-initialization loop. Assume that an LRU replacement algorithm is used and that all frames are initially empty. Assume that the page size is 16.

```
for I = 0 to 7 do
    for J = 0 to 7 do
        A[I, J] = 0;
    End for
End for
```

SOLUTION: The arrangement of the array elements in the secondary storage is shown in Fig. 8.19. The sequence of requests for the array elements in the first TWO external loop executions is as follows:

When I = 0 then J = 0, 1, 2, 3, 4, 5, 6, 7

a[0,0], a[0,1], a[0,2], a[0,3], a[0,4], a[0,5], a[0,6], a[0,7]

The number of page faults (PFs) = 1

When I = 1 then J = 0, 1, 2, 3, 4, 5, 6, 7

a[1,0], a[1,1], a[1,2], a[1,3], a[1,4], a[1,5], a[1,6], a[1,7]

The number of page faults (PFs) = 1

From the above analysis, it is clear that there will be one page fault (PF) in every external loop execution. This makes the total number of page faults (PFs) be 8. It should be noted that if the array was stored column-major, then every internal loop execution would generate eight page faults, thus causing the total number of page faults (PFs) to become 64.

- The property of hard disk provides an illusion of more capacity than actually exists in the main memory is called virtual memory.
- Two distinct schemes and their combinations are widely used for virtual memory:
  (*i*) paging (*ii*) segmentation.

SOLVED PROBLEMS

8.1. (*i*) How many 128 × 8 RAM chips are needed to provide a memory capacity of 2048 bytes?

(*ii*) How many lines of the address bus must be used to access 2048 bytes of memory? How many of there lines will be common to all chips?

(*iii*) How many lines must be decoded for chip select? Specify the size of the decoders.

Ans. (*i*) Memory capacity = 2048 bytes

2^{11} = 2048 bytes

2^7 = 128

2^4 = 16 chips RAM

(*ii*) Memory capacity = 2048 bytes

address bus = 11(2^{11} = 2048)

128 = 2^7

= 7 lines to address each chip and 4 lines to decoder for selecting 16 chips. Thus, 7 lines out of 11 will be common to all chips.

(*iii*) Remaining 4 lines must be decoded for chip select the size of decoder is 4 × 16.

8.2. How many ROM chips are required to produce a memory capacity of 4000 bytes? How many address lines are required to access the 4000 bytes? How many of these addresses will be common to all these chips?

Ans. 4000 bytes = 4K bytes = $2^2 \times 2^{10}$ bytes = 2^{12} bytes.

ROM size = 256 byte = 2^8 byte

No. of ROM chip req. = $\frac{2^{12}}{2^8} = 2^4 = 16$

No. of add. Lines required to access 4 K bytes = 12.

256 No. of addresses will be common to all these chips.

8.3. For the following memory units (specified by the number of words the number of bits per word), determine the number of address lines, input/output lines and the number of bytes that can be stored in the specified memory

(i) 64K × 8 **(ii) 16M × 32**

(iii) 4G × 64 **(iv) 2K × 16**

Ans. (i) 64K × 8

i/p, o/p lines = 8

Address lines = 16

Mem = 64K

(ii) 16M x 32

i/p, o/p lines = 32

Add = 24

Mem = 64M (16M × 4)

(iii) 4G × 64

i/p, o/p lines = 64

Add = 32

Mem = 32GB (4G x 8)

(iv) 2K x 16

i/p, o/p = 16

Add = 11

Mem = 4K

8.4. Discuss the different ways in which ROM can be programmed.

Ans. The required path in a ROM may be programmed in three different ways. The first, mask programming, is done by the semiconductor company during the last fabrication process of the unit. The procedure for fabricating a ROM requires that the customer fill out the truth table that he or she wishes the ROM to satisfy. The truth table may be submitted in a special form provided by the manufacturer makes the corresponding mask for the paths to produce the 1's and 0's according to the customer's truth table.

This procedure is costly because the vendor charges the customer a special fee for custom masking the particular ROM. For this reason, mask programming is economical only if a large quantity of the same ROM configuration is to be ordered.

For small quantities it is more economical to use a second type of ROM called a programmable read-only memory or PROM. When ordered, PROM units contain all the fuses intact, giving all 1's in the bits of the stored words. The fuses in the PROM are blown by application of current pulses through the output terminals for each address. A blown fuse defines a binary 0 state, and an intact fuse gives a binary 1 state.

This allows users to program PROMs in their own laboratories to achieve the desired relationship between input addresses and words. Special instruments called PROM programmers are available commercially to facilitate this procedure. In any case, all procedures for programming ROMs are hardware procedures even through the word "programming" is used.

The hardware procedure for programming ROMs or PROMs is irreversible, and once programmed, the fixed pattern is permanent and cannot be altered. Once a bit pattern has been established, the unit must be discarded if the bit pattern is to be changed. A third type of ROM available is called erasable PROM or EPROM can be restructured to the initial value even through its fuses have been blown previously. When the EPROM is placed under a special ultraviolet light for a given period of time, the shortwave radiation discharges the internal gates that serve as fuses. After erasure, the EPROM returns to its initial state and can be reprogrammed to a new set of words.

Certain PROMs can be erased with electrical signals instead of ultraviolet light. These PROMs are called electrically *erasable* PROM or EEPROM.