Unit 10 Memory Unit – Part II

Structure:

10.1 Introduction

Memory unit is used for storage, and retrieval of data and instructions. A typical computer system is equipped with a hierarchy of memory subsystems, some internal to the system and some external. Internal memory systems are accessible by the CPU directly and external memory systems are accessible by the CPU by an I/O module.

Objectives:

By the end of Unit 10, you should be able to:

1. With neat diagram explain the cache memory. Also explain its interaction with the processor.
2. Explain the cache read operation

10.2 Cache Memory

For all instruction cycles, the CPU accesses the memory at least once to fetch the instruction and sometimes again accesses memory to fetch the operands. The rate at which the CPU can execute instructions is limited by the memory cycle time. This limitation is due to the mismatch between the memory cycle time and processor cycle time. Ideally, the main memory should be built with the same technology as that of CPU registers, giving memory cycle times comparable to processor cycle times but this is a too expensive strategy.

The solution is to exploit the principle of locality by providing a small, fast memory between the CPU and the main memory. This memory is known as cache memory. Thus the intention of using cache memory is to give memory speed approaching the speed of fastest memories available, and at the same time provide a large memory size at the price of less expensive types of semiconductor memory.

Principles of Cache Memory

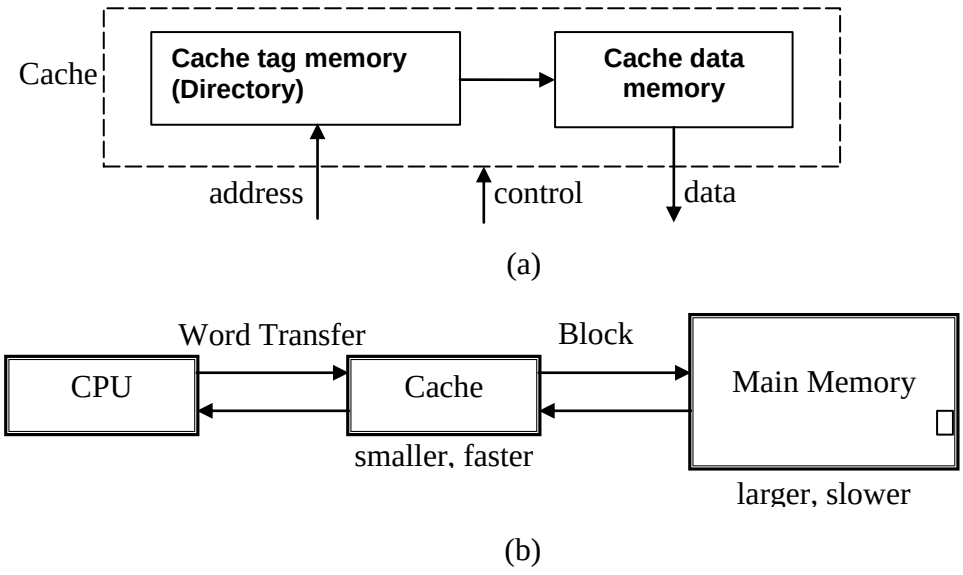

Figure 10.1 (a & b): Cache and main memory

The concept is illustrated in figure 10.1. The cache memory contains the copy of portions of main memory. When CPU attempts to read a word from main memory, a check is made to find if the word is in cache memory. If so cache and then the word is delivered to the CPU. The purpose of reading a block from main memory is that it is likely that future references will be to other words in the block.

At any time, a subset of main memory resides in the lines of the cache. Each line of the cache contains a *tag*, which identifies the location in memory of the block it contains. Word is delivered to the CPU. If not then a block of main memory consisting of fixed number of words is read into. This tag is usually a portion of the main memory address. The collections of tags which are currently assigned to the cache are stored in a special memory, called the cache *tag memory* or directory as shown in fig 10.1(a). The time

required to access the cache memory is less than the time required to access main memory.

During a Read or Write cycle the CPU initiates the memory access by placing an address on the memory bus. This address is compared with the tag address of the cache. If the tag address matches with address, *cache hit* is said to occur. Otherwise a *cache miss* is said to occur. When a cache hit occurs, the Read or Write operation is performed in cache, main memory is not involved. In the case of a cache miss, the required data is brought from main memory into cache. When the cache is full, the cache control hardware must decide which block is to be removed to create space for the new block that contains the referenced word. The collection of rules for making this decision constitutes the *replacement algorithm*. The complete cache organization is as shown in figure 10.1(c). The CPU does not know explicitly about the existence of the cache.

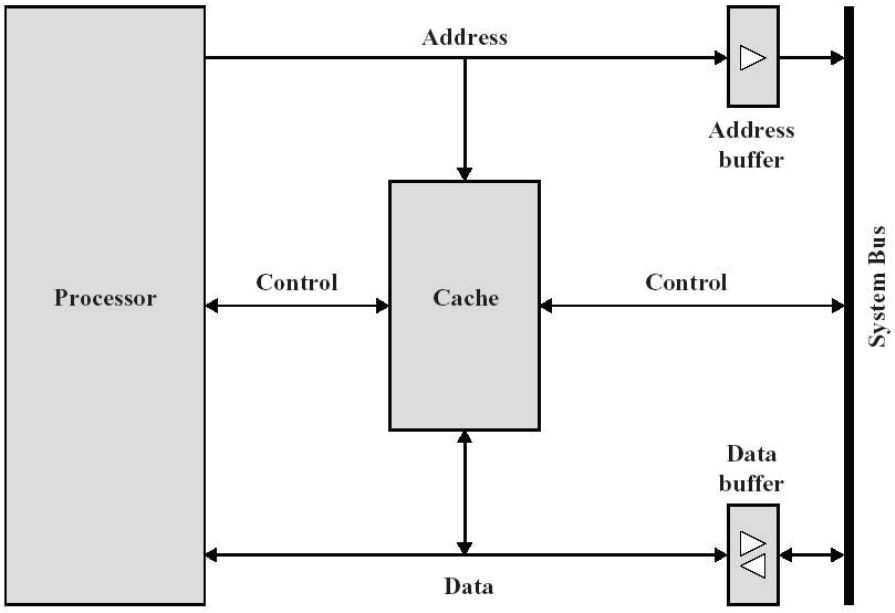

Figure 10.1(c): Cache organization showing the interconnection with the processor

When it makes Read and Write requests using the addresses that refer to locations in the main memory, the memory access control circuitry determines whether the requested word currently exists in the cache. The transfers between the main memory and CPU are fast because of small block size and fast RAM access methods.

Structure of cache and main memory

The structure of the cache and main memory is as shown in figure 10.2. Main memory consists of 2^n addressable words, each word having a unique n bit address. This memory is considered to consist of a number of fixed length blocks of size = K words each. That is $M = 2^n/K$ blocks. Cache consists of C lines of K words each. The number of lines (C) of cache is less than the number of main memory blocks (M).

C<<M

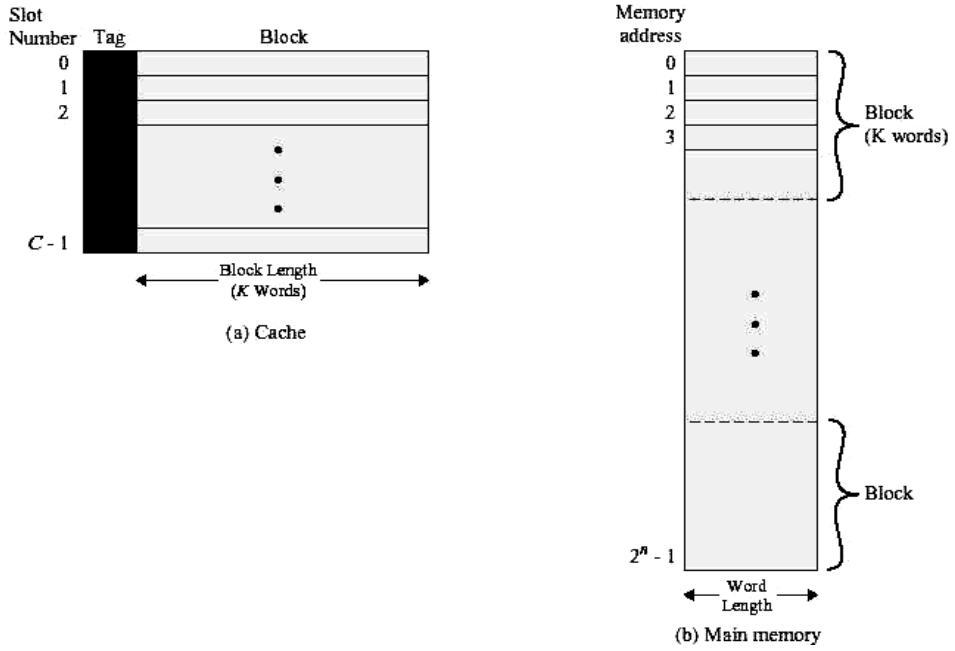

Figure 10.2: Cache and Main memory

Cache Read operation

Now let us study how cache read operation is executed. The flow chart illustrating the steps of cache read operation is given in figure 10.3. The processor generates the address 'RA' of a word to be read. If the word is contained in the cache, it is delivered to the processor. Otherwise, the block containing that word is loaded into the cache and simultaneously the word is delivered to the processor from main memory. These last two operations occur in parallel.

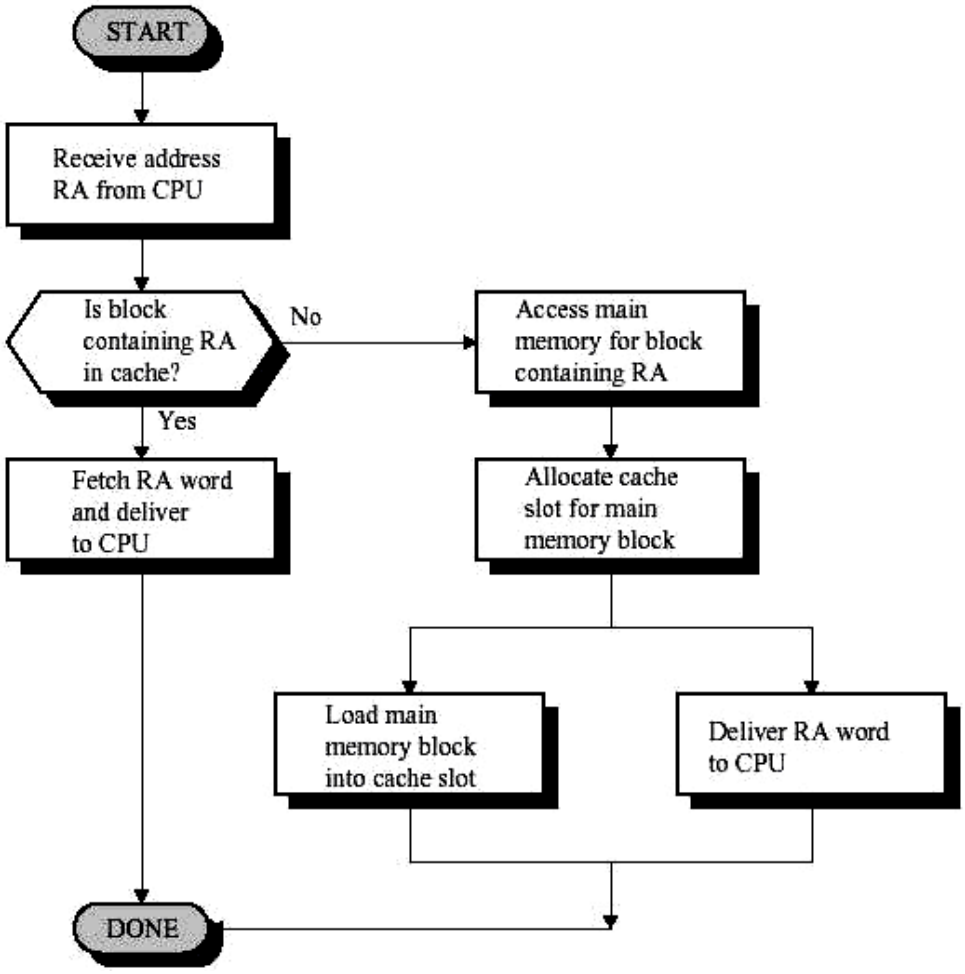

Figure 10.3: Cache Read Operation

In case of interleaved memory, contiguous block transfers are very efficient. Transferring data in blocks between the main memory and the cache enables an interleaved memory to operate at its maximum possible speed. A cache can be introduced in two general ways. They are *look-aside design* and *look-through design.*

A) Look-aside design

Figure 10.4 shows a *look-aside* design. In this case both CPU and main memory are directly connected to the system bus. When a cache miss occurs, it results in data transfer between cache and main memory through the system bus, making it unavailable for other I/O operations. This drawback can be overcome in look-through design.

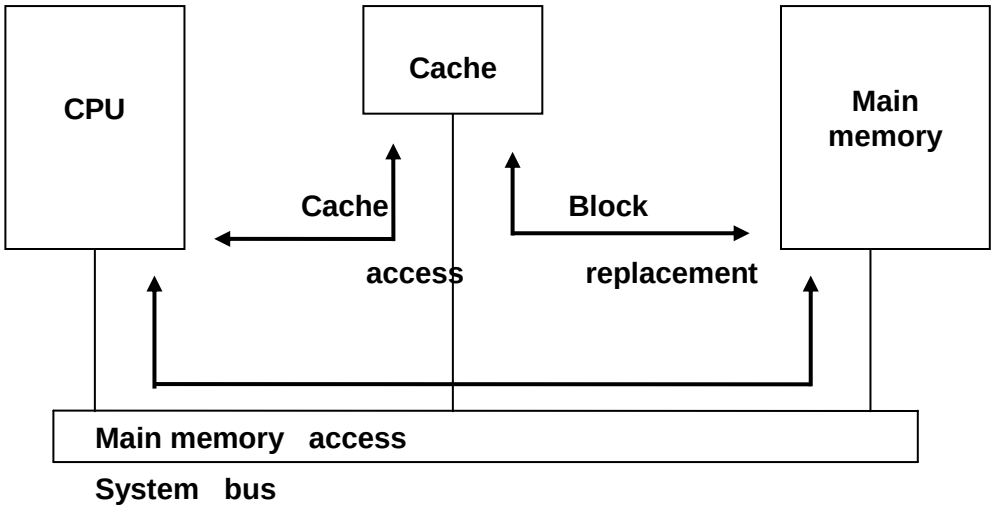

Figure 10.4: Organization of look-aside design

B) Look-through design

It is a faster, complex and costly organization when compared to the look-aside organization. Block diagram of look through design is given in figure 10.5. The communication between CPU and cache is through a separate bus, which is isolated from the main system bus. Hence system bus is

available for other units, such as I/O controllers to communicate with main memory.

Thus the memory accesses which do not involve system bus can be performed concurrently. In a look-through cache the local bus linking main memory and cache is wider than the system bus. Therefore data transfer between cache and main memory is faster. For example if system data bus is 32 bits wide and cache block size is 128 bits, a 128 bit data bus might be provided to link between cache and main memory.

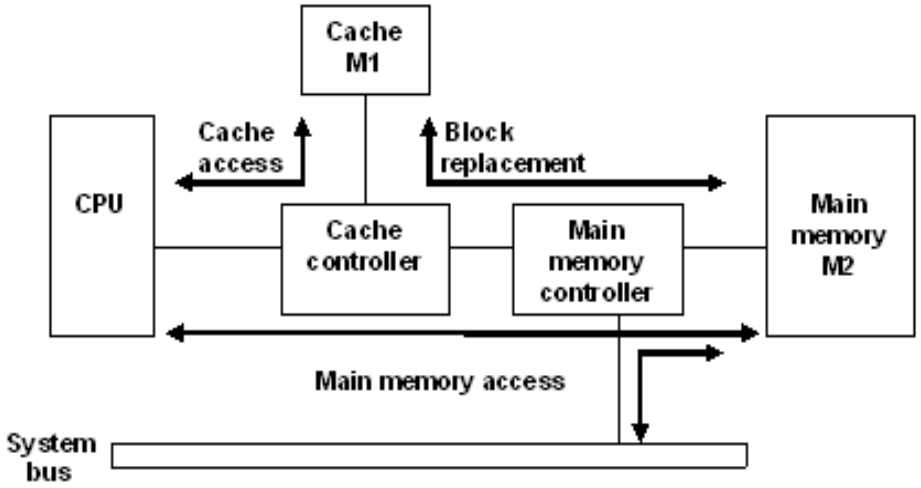

Figure 10.5: Organization of look-through cache

The main drawback of this design is that it takes more time for main memory to respond to the CPU when a cache miss occurs.

Performance using Cache Memory

The data is transferred from the main memory to the cache in blocks. Typical block size is 4 to 64 words. When the cache is full, one of the existing blocks will be evicted using standard replacement policies such as *First-in-First-out* or *Least Recently Used* (LRU). This block transfer is carried in anticipation that the block size is very likely to be referenced again and

again in the near future. The performance of a system that employs cache can be analyzed as follows:

Let t_c, h and t_m represent the cache access time, hit ratio and the main memory access time respectively. Then the average access time can be determined by the equation,

$\bar{t} = h.t_c + (1-h)(t_c + t_m)$ The hit ratio always lies in the closed interval 0 and 1, and it specifies the relative number of successful references to the cache. The above equation can be delivered using the fact that when there is a cache hit, the main memory will not be accessed, and in the case of cache miss, both main memory and cache will be accessed.

Suppose the ratio of main memory access time to the cache access time is γ, then an expression for the efficiency of a system that uses a cache can be delivered as follows.

$$\text{Efficiency} = \eta = t_c / \bar{t}$$

$$= t_c / [ht_c + (1-h)(t_c + t_m)]$$

$$= 1/[h + (1-h)(1+ t_m/t_c)]$$

$$= 1/[h + (1-h)(1+ \gamma])$$

$$= 1/(h + 1 - h + \gamma h)$$

$$= 1/[1 + \gamma(1-h)]$$

Thus η is maximum when h=1; i.e. efficiency of the system that uses cache is maximum when all the references are confined to the cache.

Elements of Cache Design

- *Cache Size:* Small cache => performance problems, large cache => too expensive.
- *Mapping Function:*

a) Direct Mapping:

Each line of cache can store specific blocks of main memory. The line number is given as

i = j modulo m,

where i = cache line number, j = main memory block number, and m = number of lines in the cache.

b) Associative Mapping:

Permits of loading each main memory block to any line of the cache.

c) Set associative Mapping:

It is a compromise that exhibits the strengths of both the direct and associative approaches while reducing their disadvantages.

We will be discussing the mapping function in detail in the following section:

- *Replacement Algorithm (for lines)*
  When a new block is brought into cache, one of the existing blocks must be replaced. The most common algorithms are:
- Least recently used (LRU)
- First in first out (FIFO)
- Least frequently used (LFU)
- Random

We will be discussing the Replacement algorithms in detail in the following section:

- *Write Policy:*
  Before a block that is resident in the cache can be replaced, it is necessary to consider whether it has been altered in the cache but not in the main memory.
- *Write through:*
  All write operations are both directly done to main memory and to cache. Main memory always contains valid content.
- *Write back:*
  Writes are only done to cache. There is an UPDATE bit set when there is a write. Before a cache block (line) is replaced, if UPDATE bit is set, its content is written to main memory. Problem is that portions of main memory are invalid for a certain period of time. If other devices access those locations, they will get wrong content. Therefore access to main memory by I/O modules can only be allowed through the cache. This makes complex circuitry and potential bottleneck.
- *Write once:*
  Problem: If there are other cache's in the system (e.g. multi-processor system sharing the same main memory), even in "write through", other cache's may contain invalid content.
- *Line Size:*
  Greater line size => more hit (+) but also more line replacements (-). Too big line size => less chance of hit for some parts of the block. Researchers suggest that 2 to 8 words seems reasonably close to optimum.

- *Number of caches:*
  Cache, internal to processor is called Level-1 (L1) cache, external cache is called Level-2 (L2) cache.

Mapping functions

The correspondence between the main memory and CPU are specified by a *mapping function*. There are three standard mapping functions namely

1. Direct mapping
2. Associative mapping
3. Block set associative mapping

In order to discuss these methods consider a cache consisting of 128 blocks of 16 words each. Assume that main memory is addressable by a 16 bit address. For mapping purpose main memory is viewed as composed of 4K blocks.

1. Direct mapping technique

This is the simplest mapping technique. In this case, block K of the main memory maps onto block K *modulo 128* of the cache. Since more than one main memory block is mapped onto a given cache block position, contention may arise for that position even when the cache is not full. This is overcome by allowing the new block to overwrite the currently resident block.

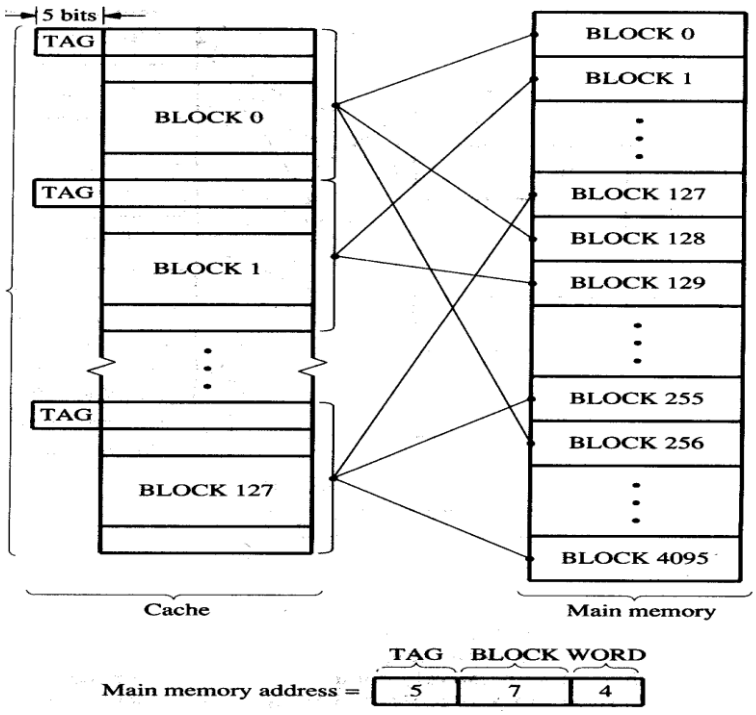

Figure 10.6: Direct mapping cache

A main memory address can be divided into three fields, TAG, BLOCK and WORD as shown in figure 10.6. The TAG bit is required to identify a main memory block when it is resident in the cache. When a new block enters the cache the 7-bit cache block field determines the cache position in which this block must be stored. The tag field of that block is compared with tag field of the address. If they match, then the desired word is present in that block of cache. If there is no match, then the block containing the required word must be first read from the main memory and then loaded into the cache.

2. Associative mapping technique

This is a much more flexible mapping technique. Here any main memory block can be loaded to any cache block position. Associative mapping is illustrated as shown in figure 10.7. In this case 12 tag bits are required to identify a main memory block when it is resident in the cache.

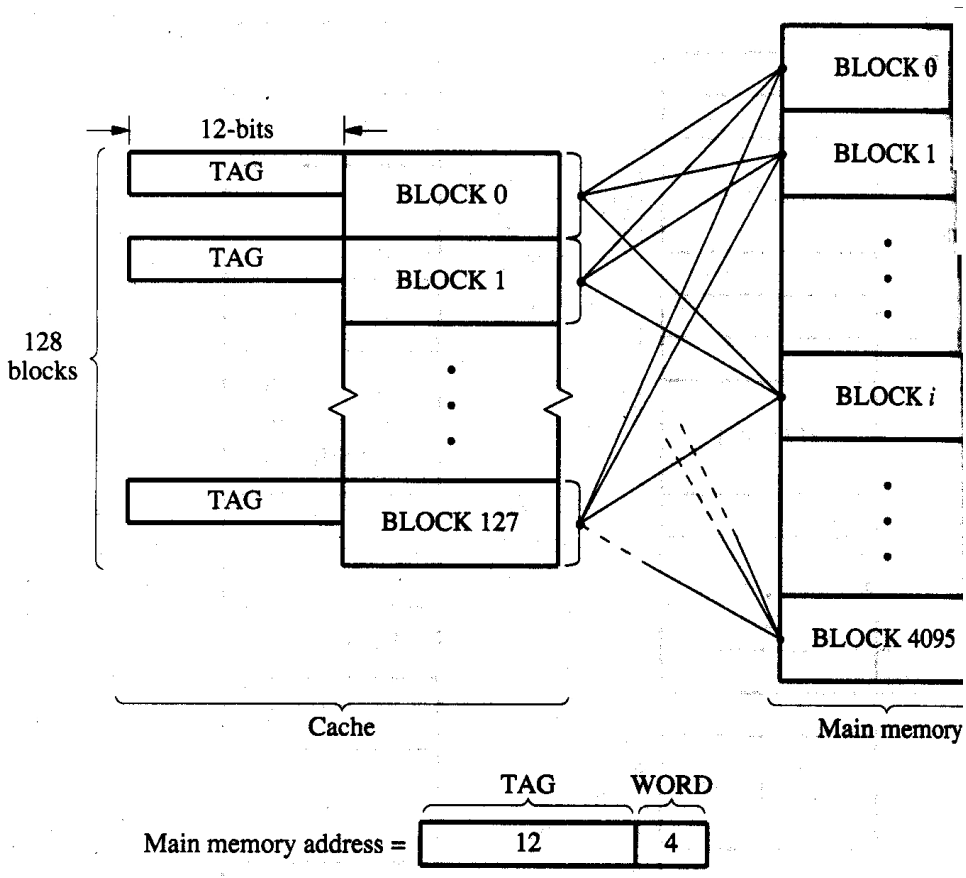

Figure 10.7: Associative mapping cache

The tag bits of an address received from the CPU are compared with the tag bits of each cache block to see if the desired block is present in the cache. Here we need to search all 128 tag patterns to determine whether a given block is in the cache. This type of search is called *associative search*. Therefore the cost of implementation is higher. Because of complete freedom in positioning, a wide range of replacement can be used.

3. Block set associative mapping

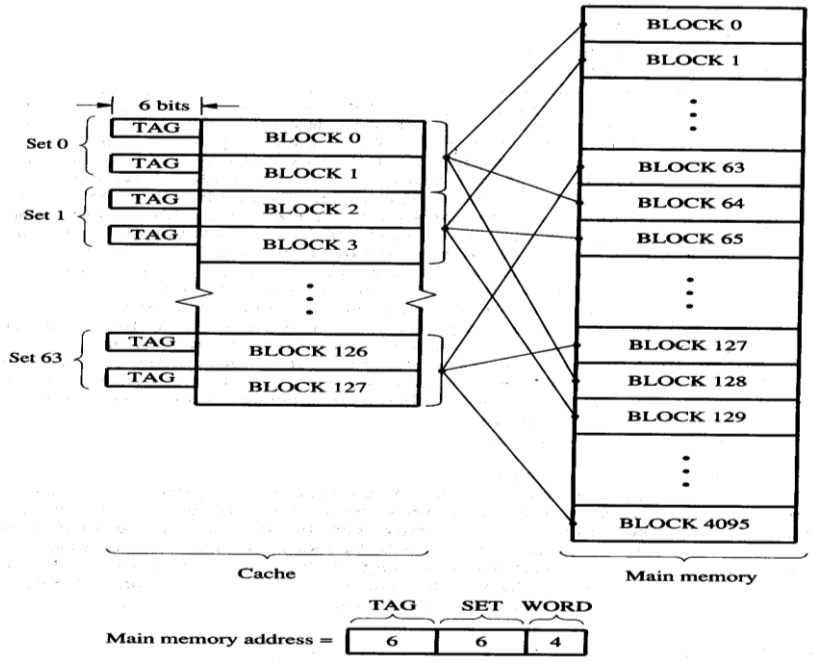

Figure 10.8: Block set associative mapping cache with two blocks per set

This is a combination of two techniques discussed above. In this case blocks of the cache are grouped into *sets* and the mapping allows a block of main memory to reside in any block of a particular set. Set associative mapping is illustrated as shown in figure 10.8.

Consider an example, a cache with two blocks per set. The 6 bit set field of the address determines which set of the cache might contain the addressed block. The tag field of the address must be associatively compared to the tags of the two blocks of the set to check if the desired block is present.

Advantages

a) The contention problem of the direct method is overcome by having few choices for block replacement.
b) The hardware cost is reduced by decreasing the size of the associative search.

Note

- The number of blocks per set can be selected according to the requirements of a particular computer system. Four blocks per set can be accommodated by a 5 bit set field, eight blocks per set by a 4-bit set field and so on. The extreme conditions are one block per set (direct mapping method) and 128 blocks per set (fully associative technique).
- Each block must be provided with a valid bit where it indicates whether the block contains valid data. When the main memory is loaded with new programs and data from mass storage devices, valid bit of a particular cache block is set to 1. It stays at 1 unless a main memory is updated by a source that bypasses the cache. In this case, a check is made to determine whether the block is currently in the cache. If it is, its valid bit is set to 0.

Replacement algorithms

For any set associative mapping a replacement algorithm is needed. The most common algorithms are discussed here. When a new block is to be brought into the cache and all the positions that it may occupy are full, then the cache controller must decide which of the old blocks to overwrite. Because the programs usually stay in localized areas for a reasonable period of time, there is high probability that the blocks that have been referenced recently will be referenced again soon. Therefore when a block is to be overwritten, the block that has not referenced for the longest time is overwritten. This block is called ***least-recently-used block*** **(LRU)**, and the technique is called the LRU *replacement algorithm*. In order to use the LRU algorithm, the cache controller must track the LRU block as computation proceeds.

There are several replacement algorithms that require less overhead than LRU method. One method is to remove the oldest block from a full set when

a new block must be brought in. This method is referred to as ***FIFO***. In this technique no updating is needed when hit occurs. However, because the algorithm does not consider the recent patterns of access to blocks in the cache, it is not effective as LRU approach in choosing the best block to remove. There is another method called ***least frequently used (LFU)*** that replaces that block in the set which has experienced the fewer references. It is implemented by associating a counter with each slot. Yet another simplest algorithm called ***random***, is to choose a block to be overwritten in random.

10.3 External Memory

Magnetic Disk

A disk is a circular platter constructed of metal or of plastic coated with a magnetic material. Data are recorded on and later retrieved from the disk via a conducting coil named the **head**. During a read or write operation, the head is stationary while the platter rotates beneath it. Writing is achieved by producing a magnetic field which records a magnetic pattern on the magnetic surface.

Data Organization and Formatting

Figure 10.9 depicts the data layout of disk. The head is capable of reading or writing from a portion of the platter rotating beneath it. This gives rise to organization of data on the platter in a concentric set of rings called Tracks. Each track is the same width as the head. Adjacent tracks are separated by gaps that minimize errors due to misalignment of head. Data is transferred to and from the disk in blocks. And the block is smaller than the capacity of a track. Data is stored in block regions which is an angular part of a track and is referred to as a sector. Typically 10-100 sectors are there per track. These may be either of fixed or variable length.

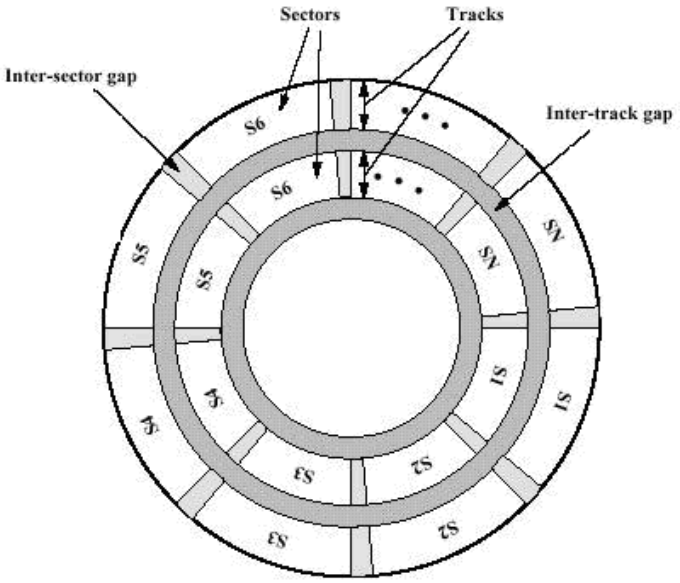

Figure 10.9: Disk Data Layout

Physical Characteristics

Head motion	:	Fixed-head disk (one / track) or Movable-head disk (one/surface).
Disk portability	:	No removable disk vs. Movable disk.
Sides	:	Double-sided vs. single-sided.
Platters	:	Single platter vs. Multiple platter disks.
Head mechanism	:	Contact (floppy), Fixed gap, Aerodynamic gap (Winchester [= hard disk]).

Disk Performance Parameters

1. Seek time: Time required to move the disk arm (head) to the required track. $Ts = m.n + s$

   Where Ts = estimated seek time, n = number of tracks traversed, m = constant that depends on the disk drive, s = startup time.

2. Rotational delay: time required to rotate the disk to get wanted sector beneath the head.
3. Transfer time: T = b / (r N)

   Where T = transfer time, b = number of bytes to be transferred, N = number of bytes on a track, r = rotation speed, in revolutions per second.
4. Access time: Ta = total average access time.

   Ta = Ts + (1 / 2 r) + (b / r N) where Ts = average seek time.

RAID

1. RAID is a set of physical disk drives viewed by the operating system as a single logical drive.
2. Data are distributed across the physical drives of an array.
3. Redundant disk capacity is used to store parity information, which guarantees data recoverability in case of a disk failure.

Optical Memory & Magnetic Tape are other two external memories.

10.4 Virtual memory

Virtual (or logical) memory is a concept that, when implemented by a computer and its operating system, allows programmers to use a very large range of memory or storage addresses for stored data. The computing system maps the programmer's virtual addresses to real hardware storage addresses. Usually, the programmer is freed from having to be concerned about the availability of data storage.

In addition to managing the mapping of virtual storage addresses to real storage addresses, a computer implementing virtual memory or storage also manages storage swapping between active storage (RAM) and hard disk or other high volume storage devices. Data is read in units called "pages" of sizes ranging from a thousand bytes (actually 1,024 decimal bytes) up to

several megabytes in size. This reduces the amount of physical storage access that is required and speeds up overall system performance.

The memory control circuitry translates the address specified by the program into an address that can be used to access the physical memory. This address is called *logical address* or *virtual address*. A set of virtual addresses constitute the *virtual address space*.

The mechanism that translates virtual addresses into physical address is usually implemented by a combination of hardware and software components. If a virtual address refers to a part of the program or data space that is currently in the physical memory, then the contents of the appropriate physical location in the main memory are accessed. Otherwise its contents must be brought into a suitable location in the main memory. The mapping function is implemented by a special memory control unit called as *memory management unit*. Mapping function can be changed during the program execution according to the system requirement.

The simplest method of translation assumes that all programs and data are composed of fixed length units called *pages*. Each page consists of a block of words that occupy contiguous locations in the main memory or in the secondary storage. Page normally ranges from 1K to 8K bytes in length. They form the basic unit of information that is transmitted between main memory and secondary storage devices.

Virtual address translation method

A virtual address translation method based on the concept of fixed length pages is shown in Fig. 10.10. Each virtual address generated by the processor is interpreted as a page number followed by a word number. Information about the disk or the main memory is kept in a *page table* in the main memory.

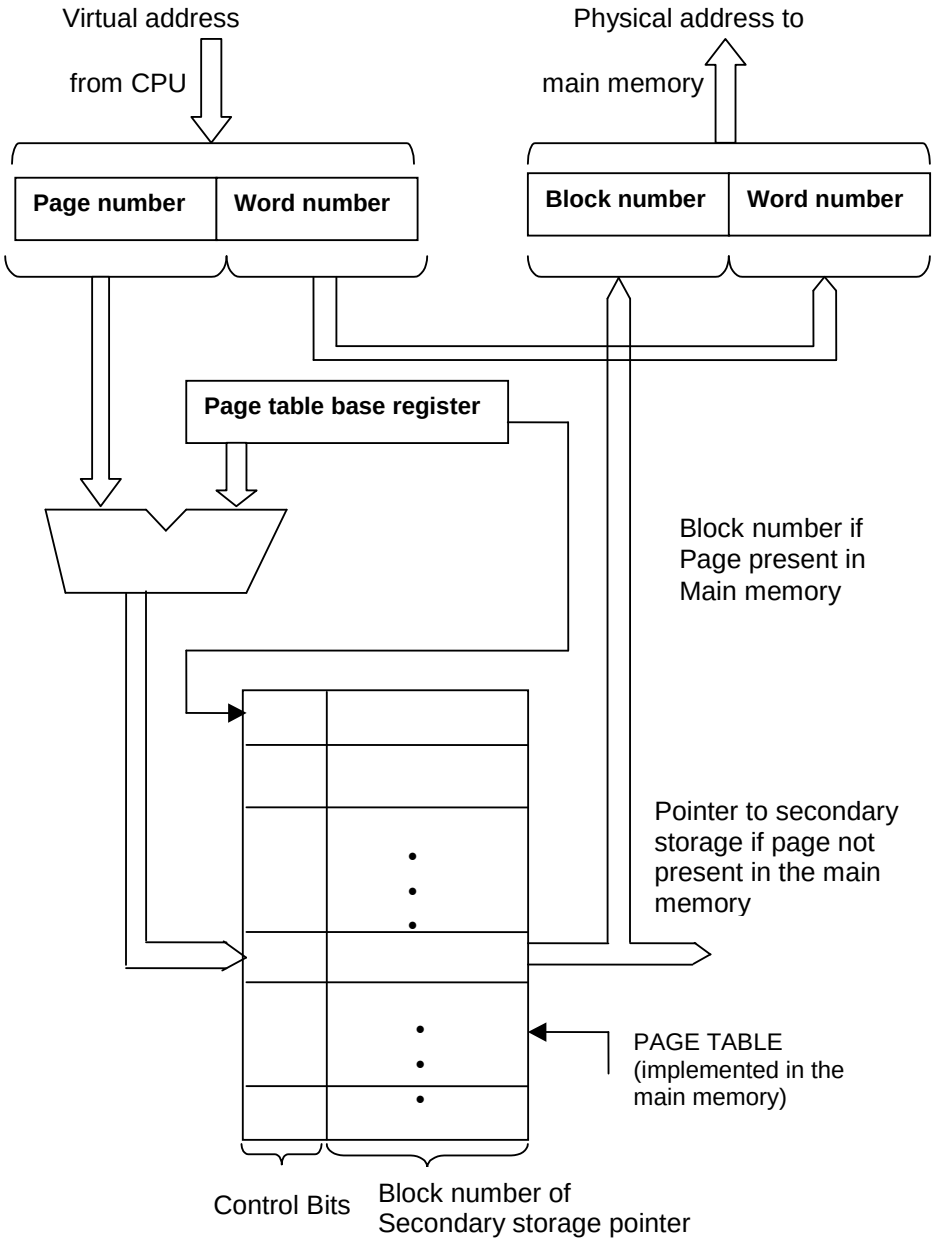

Fig. 10.10: Virtual memory address translation

The starting address of this table is kept in a *page table base register.* By adding the page number to the contents of this register, the address of corresponding entry in the page table is obtained. The content of this location gives the starting address of the page if that page currently resides in the main memory. Otherwise they indicate the location where the page is to be found in the secondary storage. In this case the entry in the table usually points to an area in the main memory where the secondary storage address of the page is held. Each entry also includes some control bits to describe the status of the page while it is in the main memory. One control bit indicates whether the page has been modified when it was in the main memory.

If the page table is stored in the main memory unit, then the two main memory accesses must be made for every main memory access requested by the program. This may result in a degradation of speed by a factor of two. However a specialized cache memory is used in most of the systems to speed up the translation process by storing recently used virtual to physical address translation.

Virtual memory increases the effective size of the main memory. Only the active space of the virtual address space is mapped onto locations in the physical main memory, whereas the remaining virtual addresses are mapped onto the bulk storage devices used. During a memory cycle the addressing spacing mechanism (hardware or software) determines whether the addressed information is in the physical main memory unit. If it is, the proper information is accessed and the execution proceeds. If it is not, a contiguous block of words containing the desired information are transferred from the bulk storage to main memory, displacing some block that is currently inactive.

10.5 Memory Management in Operating Systems

In virtual memory concept we assumed that only one large program is being executed. If program and data does not fit into the physical main memory, then a secondary storage should hold the overflow. The operating system automatically swaps the programs between the main memory and secondary storage.

Management routines are the parts of the operating system of the computer. Virtual address space is divided into two parts *system space* and *user space*. Operating system routines reside in *system space* and user application programs reside in the *user space*.

In a multiuser environment each user will have a separate user space with a separate page table. The memory management unit uses a page table base register to determine the address of table to be used in the translation process. Hence by changing the contents of this register operating system can switch from one space to another. The physical main memory is thus shared by the active pages of the system space and several user spaces. However, only the pages that belong to one of these spaces are accessible at any given time. In a multiuser environment, no program should be allowed to modify either data or instructions of other programs in the main memory. Hence protection should be given.

Such protection can be provided in several ways. Let us first consider the most basic form of protection. Recall that in the simplest case the processor has two states namely, the *supervisor state* and *user state.* The processor is placed in the supervisor state while operating system routines are being executed and in the user state to execute user programs. In the user state some machine instructions cannot be executed. These *privileged instructions* which include operations such as modifying the page table base register cannot be executed in the user state. Hence a user program is

prevented from accessing the page tables of other user spaces or of the system space.

It is sometimes desirable for one application program to have access to certain pages belonging to another program. The operating system can arrange this by causing these pages to appear in both spaces. The shared pages will therefore have entries in two different page tables. The control bits in each table entry can be set differently to control the Read/Write access privileges granted to each program. The types of *partitioning* methods are Fixed-size and unequal-size partitions and dynamic partitioning. *Paging* is more efficient than partitioning. Programs are divided to "logical" chunks known as "pages", assigned to available chunks of memory known as "frames" or "page frames".

10.6 Summary

An intermediate memory called cache memory can be introduced between main memory and CPU. We have introduced the reader to the principle structure of cache memory along with the mapping functions and replacement algorithms. Finally, we have touched upon a few external memory elements, and another method to increase the effective size of the memory is to introduce virtual memory

Self Assessment Questions

1. _____ memory contains the copy of portions of main memory.
2. In _____ design both CPU and main memory are directly connected to the system bus.
3. Efficiency of the system that uses cache is _____, when all the references are confined to the cache.
4. Data are recorded on and later retrieved from the disk via a conducting coil named _____.
5. User application programs reside in the _____.

10.7 Terminal Questions

1. Explain the replacement algorithms.
2. Discuss the physical characteristics of DISK.
3. Write a note about RAID.

10.8 Answers

Self Assessment Questions

1. cache
2. look-aside design
3. maximum
4. head
5. user space

Terminal Questions

1. Refer Section 10.2
2. Refer Section 10.3
3. Refer Section 10.3