Chapter 5: Memory Device Characteristics

5.1 Memory Hierarchy

A typical memory hierarchy starts with a small, expensive, and relatively fast unit, called the cache, followed by a larger, less expensive, and relatively slow main memory unit. Cache and main memory are built using solid-state semiconductor material (typically CMOS transistors).

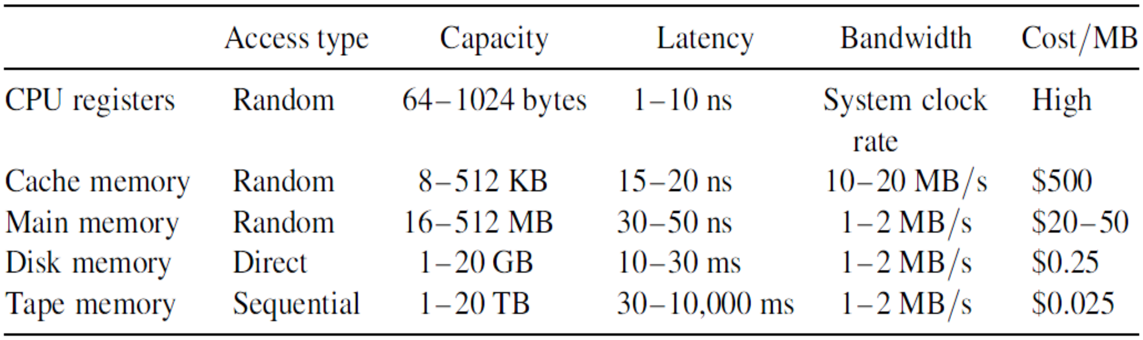
It is customary to call the fast memory level the primary memory. The solid-state memory is followed by larger, less expensive, and far slower magnetic memories that consist typically of the (hard) disk and the tape. It is customary to call the disk the secondary memory.

5.1.1 Memory Hierarchy Characteristics

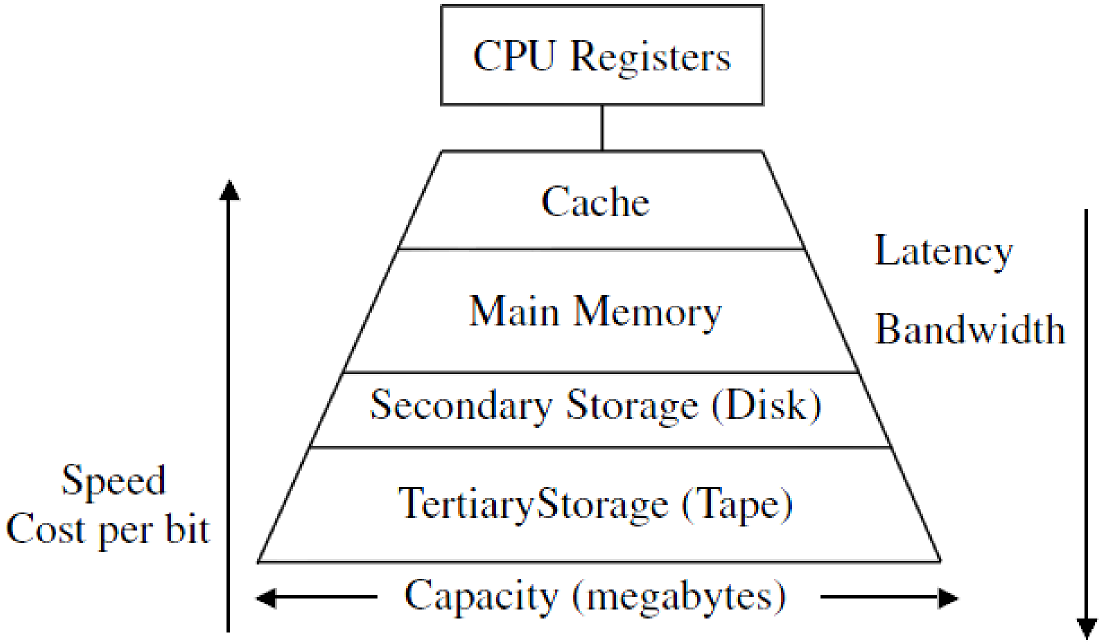
The memory hierarchy can be characterized by a number of parameters. Among these parameters are the access type, capacity, cycle time, latency, bandwidth, and cost. The term access refers to the actions that physically takes place during a read or write operation.

The capacity of a memory level is usually measured in bytes. The cycle time is defined as the time elapsed from the start of a read operation to the start of a subsequent read. The latency is defined as the time interval between the request for information and the access to the first bit of that information. The bandwidth provides a measure of the number of bits per second that can be accessed.

The term random access refers to the fact that any access to any memory location takes the same fixed amount of time regardless of the actual memory location and/or the sequence of accesses that takes place. For example, if a write operation to memory location 100 takes 15 ns and if this operation is followed by a read operation to memory location 3000, then the latter operation will also take 15 ns. This is to be compared to sequential access in which if access to location 100 takes 500 ns, and if a consecutive access to location 101 takes 505 ns, then it is expected that an access to location 300 may take 1500 ns. This is because the memory has to cycle through locations 100 to 300, with each location requiring 5 ns.



	Access type	Capacity	Latency	Bandwidth	Cost/MB
CPU registers	Random	64–1024 bytes	1–10 ns	System clock rate	High
Cache memory	Random	8–512 KB	15–20 ns	10–20 MB/s	$500
Main memory	Random	16–512 MB	30–50 ns	1–2 MB/s	$20–50
Disk memory	Direct	1–20 GB	10–30 ms	1–2 MB/s	$0.25
Tape memory	Sequential	1–20 TB	30–10,000 ms	1–2 MB/s	$0.025



5.2 Memory Organization

The memory unit that communicates directly with the CPU is called the *main memory*. Device that provide backup storage are called *auxiliary memory* (magnetic disks and tapes) Used for storing system programs, large data files and other backup information. Only programs and data currently needed by the processor reside in main memory.

A special very-high-speed memory called a *cache* is sometimes used to increase the speed of processing. The cache access time is close to processor logic clock cycle time. The cache is used for storing segments of programs currently being executed in the CPU and temporary data frequently needed in the present calculations.

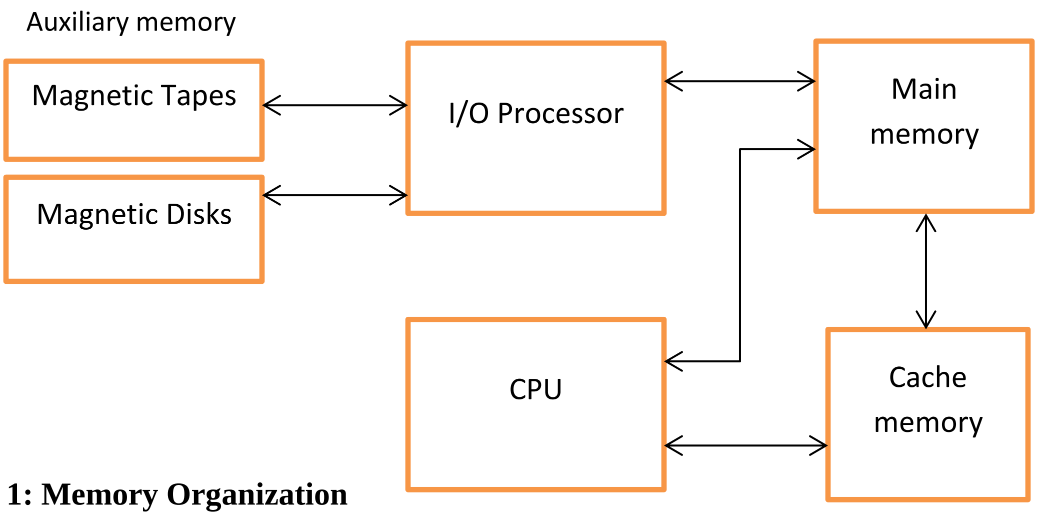


Figure 1: Memory Organization

5.2.1 Main Memory

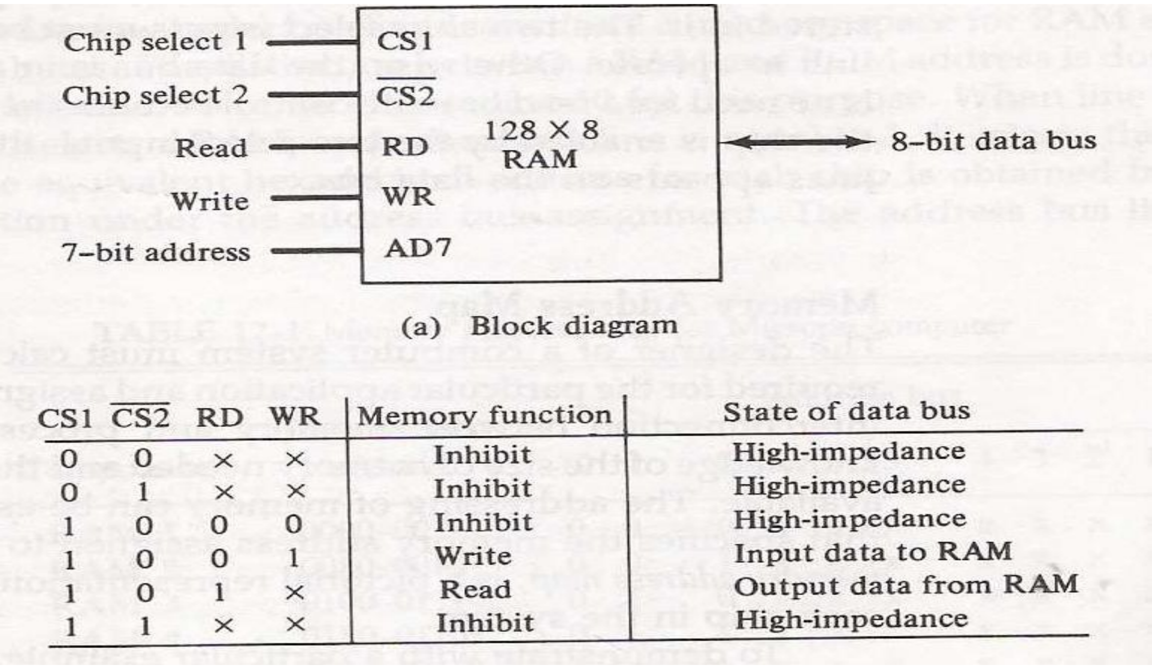
Main memory used to store programs and data during the computer operation. Technology based on semiconductor integrated circuits.

1- RAM (Random Access Memory)

The static RAM consists of internal Flip-Flops. Easier to use, shorter read/ write cycle. Dynamic RAM stores the binary information in the form of electric charges that are applied to capacitors. The capacitors must be periodically recharged. Large storage capacity, reduced power consumption.

2- ROM (Read Only Memory)

Used to storing programs that is permanently resident in the computer. ROM store initial program called *bootstrap loader*



(a) Block diagram

CS1	$\overline{\text{CS2}}$	RD	WR	Memory function	State of data bus
0	0	×	×	Inhibit	High-impedance
0	1	×	×	Inhibit	High-impedance
1	0	0	0	Inhibit	High-impedance
1	0	0	1	Write	Input data to RAM
1	0	1	×	Read	Output data from RAM
1	1	×	×	Inhibit	High-impedance

Figure 2: Typical RAM chip

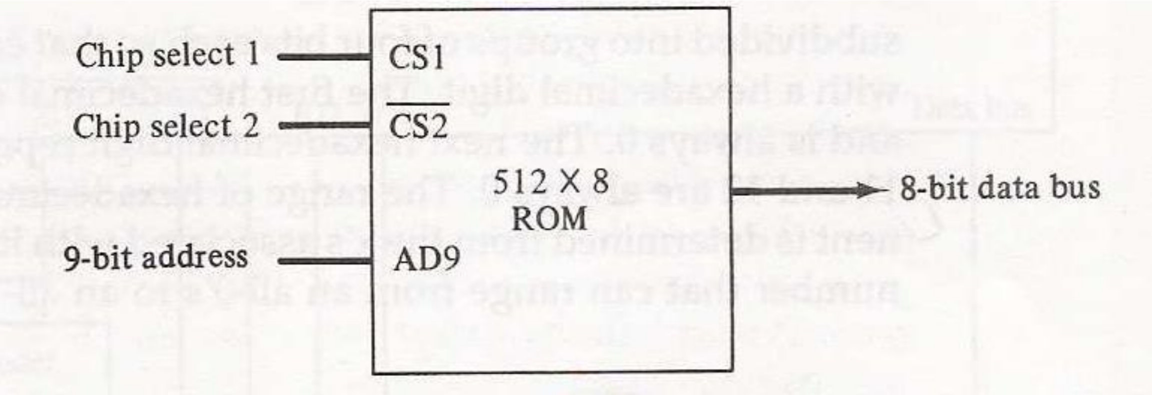
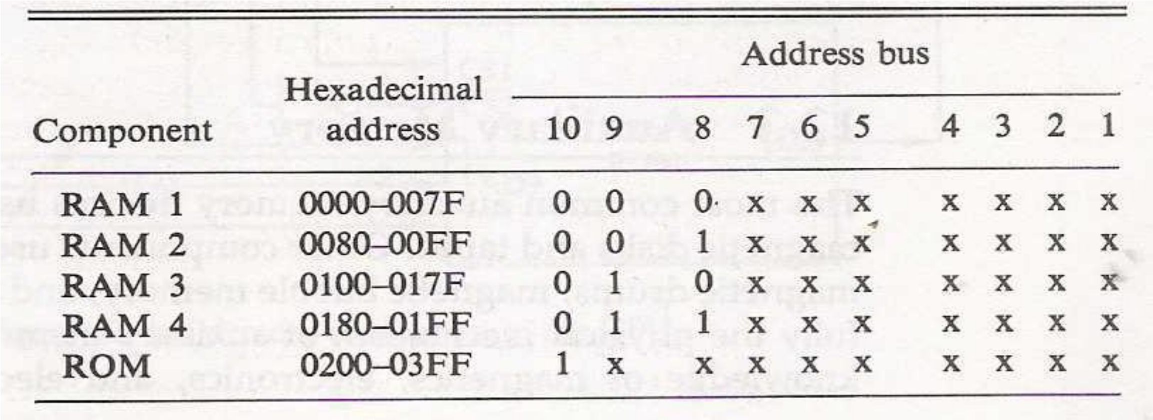


Figure 3: Typical ROM chip

Example:

Assume that a computer system needs 512 bytes of RAM and 512 bytes of ROM. The RAM and ROM chips available are RAM(128 x 8) ROM(512 x 8)



	Hexadecimal	Address bus									
Component	address	10	9	8	7	6	5	4	3	2	1
RAM 1	0000–007F	0	0	0	x	x	x	x	x	x	x
RAM 2	0080–00FF	0	0	1	x	x	x	x	x	x	x
RAM 3	0100–017F	0	1	0	x	x	x	x	x	x	x
RAM 4	0180–01FF	0	1	1	x	x	x	x	x	x	x
ROM	0200–03FF	1	x	x	x	x	x	x	x	x	x

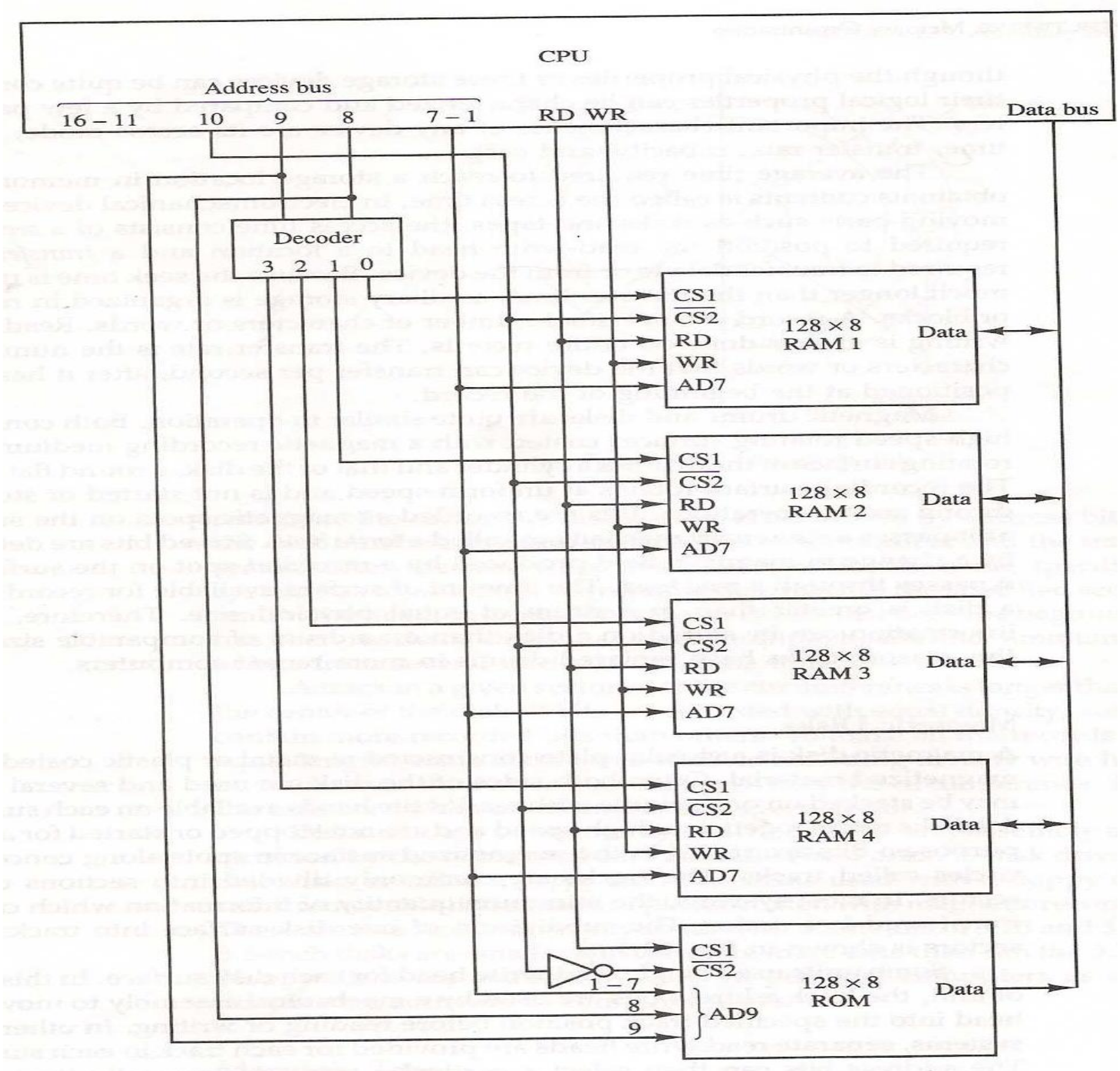


Figure 4: Example solution

5.2.2 Auxiliary Memory

The most devices are magnetic disks and tapes. The important characteristics of any device are its *access mode*, *access time*, *transfer rate*, *capacity and cost.*

The average time required to reach a storage location in memory and obtain its contents is called the access time. Seek time required to position the read/write head to a location and a transfer time required to transfer data to or from the device. Auxiliary storage is organized in records or blocks. The transfer rate is the number of characters or words that the device can transfer per second.

1- Magnetic Disks

A magnetic disk is a circular plate constructed of metal or plastic coated with magnetized material. Often both sides of the disk are used and several disks may be stacked on one spindle with read/write heads available on each surface. All disks rotate together at high speed and are not stopped or started for access purposes. Bits are stored in the magnetized surface in spots along concentric circles called tracks. The tracks are commonly divided into sections called sectors.

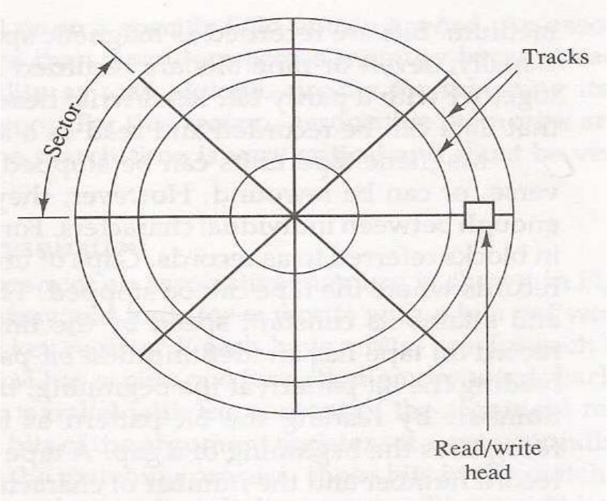


Figure 5: Magnetic Disk

2- Magnetic Tape

The tape itself is a strip of plastic coated with a magnetic recording medium. Bits are recorded as magnetic spots on the tape along several tracks. Read/Write heads are mounted one in each track so that data can be recorded and read as a sequence of characters. Magnetic tape units can be stopped started to move forward or in reverse. Gaps of unrecorded tape are inserted between records where the tape can be stopped.

3- Associative Memory

Many search algorithms have been developed to minimize the number of accesses while searching for an item in a random or sequential access memory. A memory unit accessed by content is called an associative memory or content addressable memory (CAM)”no address is given”.

5.2.3 Cache Memory

The idea behind using a cache as the first level of the memory hierarchy is to keep the information expected to be used more frequently by the CPU in the cache (a small high-speed memory that is near the CPU).

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.

The effectiveness of a memory hierarchy depends on the principle of moving information into the fast memory infrequently and accessing it many times before replacing it with new information. This principle is possible due to a phenomenon called locality of reference; that is, within a given period of time, programs tend to reference a relatively confined area of memory repeatedly. There exist two forms of locality: spatial and temporal locality. Spatial locality refers to the phenomenon that when a given address has been referenced, it is most likely that addresses near it will be referenced within a short period of time, for example, consecutive instructions in a straight line program. Temporal locality, on the other hand, refers to the phenomenon that once a particular memory item has been referenced, it is most likely that it will be referenced next, for example, an instruction in a program loop.

A hierarchy consisting only of two levels, that is, the cache and the main memory. We assume that the main memory access time is tm and the cache access time is tc. We will measure the impact of locality in terms of the average access time, defined as the average time required to access an element (a word) requested by the processor in such a two-level hierarchy.

Impact of Temporal Locality

In this case, we assume that instructions in program loops, which are executed many times, for example, n times, once loaded into the cache, are used more than once before they are replaced by new instructions. The average access time, tav, is given by

$$t_{\mathrm{av}} = \frac{nt_{\mathrm{c}} + t_{\mathrm{m}}}{n} = t_{\mathrm{c}} + \frac{t_{\mathrm{m}}}{n}$$

Impact of Spatial Locality

In this case, it is assumed that the size of the block transferred from the main memory to the cache, upon a cache miss, is m elements. We also assume that due to spatial locality, all m elements were requested, one at a time, by the processor.

Based on these assumptions, the average access time, tav, is given by

$$t_{\mathrm{av}} = \frac{mt_{\mathrm{c}} + t_{\mathrm{m}}}{m} = t_{\mathrm{c}} + \frac{t_{\mathrm{m}}}{m}$$

Impact of Combined Temporal and Spatial Locality

In this case, we assume that the element requested by the processor created a cache miss leading to the transfer of a block, consisting of m elements, to the cache (that take tm). Now, due to spatial locality, all m elements constituting a block were requested, one at a time, by the processor (requiring mtc). Following that, the originally requested element was accessed (n 2 1) times (temporal locality), that is, a total of n times access to that element. Based on these assumptions, the average access time, tav, is given by

$$t_{\mathrm{av}} = \frac{\left(\frac{mt_{\mathrm{c}} + t_{\mathrm{m}}}{m}\right) + (n-1)t_{\mathrm{c}}}{n} = \frac{t_{\mathrm{c}} + \left(\frac{t_{\mathrm{m}}}{m}\right) + (n-1)t_{\mathrm{c}}}{n} = \frac{t_{\mathrm{m}}}{nm} + t_{\mathrm{c}}$$

A further simplifying assumption to the above expression is to assume that tm = mtc. In this case the above expression will simplify to

$$t_{\mathrm{av}} = \frac{mt_{\mathrm{c}}}{nm} + t_{\mathrm{c}} = t_{\mathrm{c}} + \frac{t_{\mathrm{c}}}{n} = \frac{n+1}{n} t_{\mathrm{c}}$$

Mapping process

The transformations of data from maim memory to cache memory known as mapping.

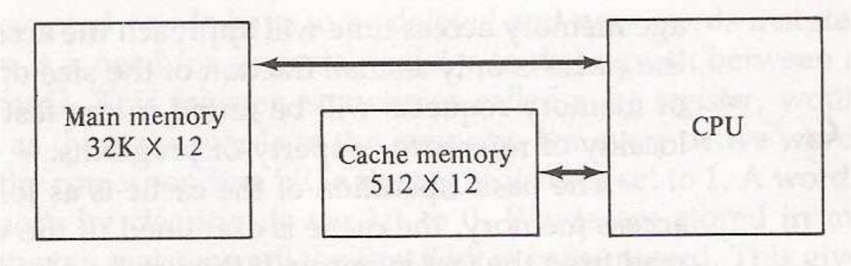


Figure 6: Two level memory hierarchy

1- Associative Mapping

The associative memory stored both the address and content (data) of the memory word. This permits any location in cache to store any word from main memory.

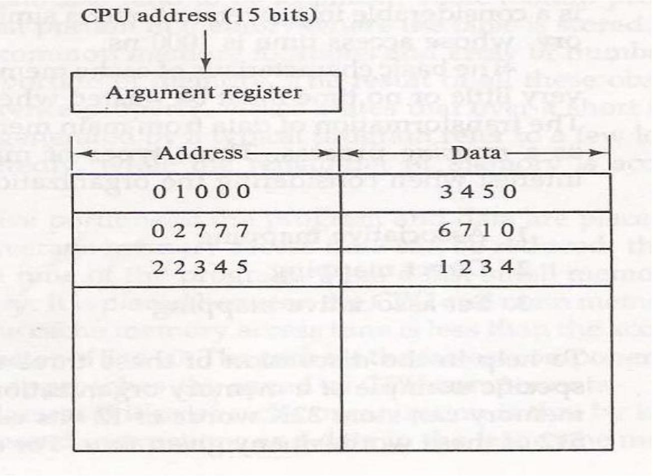


Figure 7: Associative Mapping

2- Direct Mapping

The CPU address of 15 bits is divided into two fields. The nine least significant bits constitute the *index* field and the remaining six bits form the *tag* field.

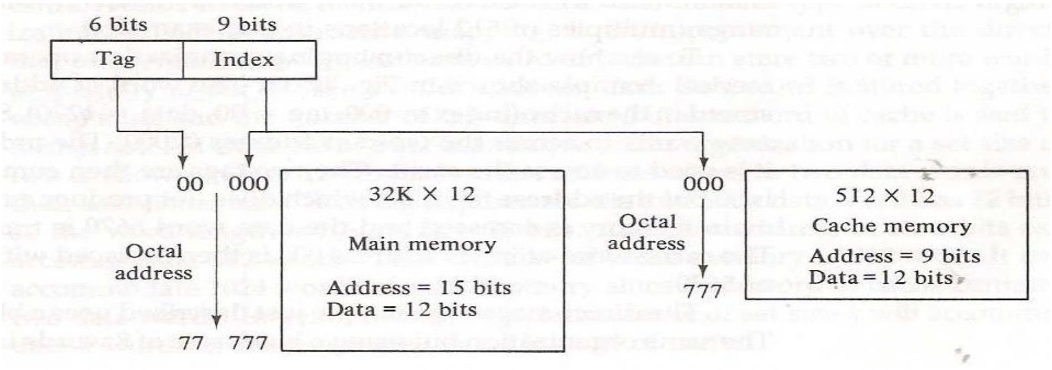


Figure 8: Direct Mapping

The disadvantage of direct mapping is that the hit ratio drop considerable if two or more words whose addresses have the same index but different tags are accessed repeatedly.

3- Set Associative Mapping

It was mentioned previously that the disadvantage of direct mapping is that two words with the same index in their address but different tag values cannot reside in cache memory at the same time.

A third type of cache organization, called set-associative mapping, is an improvement over the direct mapping organization in that each word of cache can store two or more words of memory under the same index address.

Each data word is stored together with its tag and the number of tag-data items in one word of cache is said form a set. Each tag requires six bits and each data word has 12 bits, so the word length is 2(6 + 12) = 36 bits. An index address of nine bits can accommodate 512 words. Thus the size of cache memory is 512 x 36. it can accommodate 1024 words of main memory since each word of cache contains two data words.

Replacement Techniques

A number of replacement techniques can be used. These include a randomly selected block (random selection), the block that has been in the cache the longest (first-in-first- out, FIFO), and the block that has been used the least while residing in the cache (least recently used, LRU).

The LRU algorithm requires the use of a cache controller circuit that keeps track of references to all blocks while residing in the cache. This can be achieved through a number of possible implementations. Among these implementations is the use of counters. In this case each cache block is assigned a counter. Upon a cache hit, the counter of the corresponding block is set to 0, all other counters having a smaller value than the original value in the counter of the hit block are incremented by 1, and all counters having a larger value are kept unchanged. Upon a cache miss, the block whose counter is showing the maximum value is chosen for replacement, the counter is set to 0, and all other counters are incremented by 1.

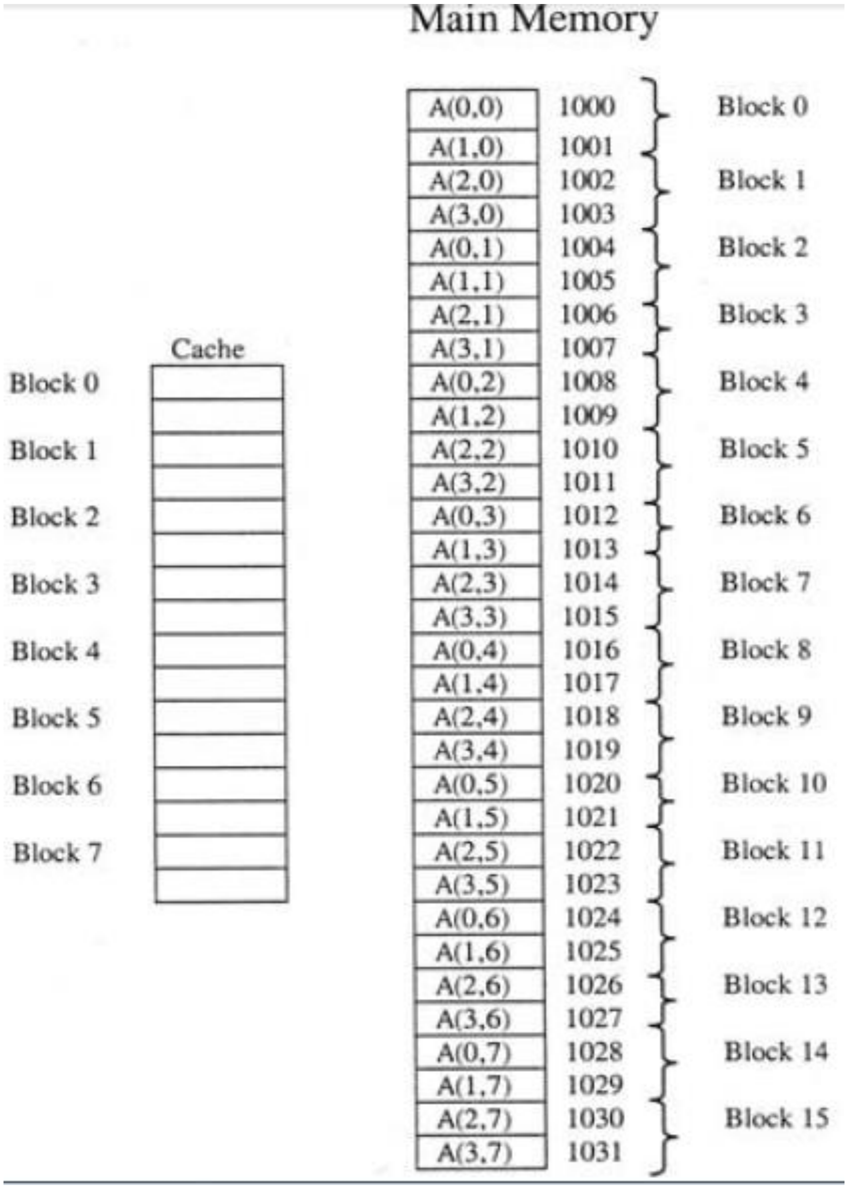
Example

Consider the case of a 4x8 two-dimensional array of numbers, A. Assume that each number in the array occupies one word and that the array elements are stored column-major order in the main memory from location 1000 to location 1031. The cache consists of eight blocks each consisting of just two words. Assume also that whenever needed, LRU replacement policy is used. We would like to examine the changes in the cache if each of the above three mapping techniques is used as the following sequence of requests for the array elements are made by the processor:

$$a_{0,0},\ a_{0,1},\ a_{0,2},\ a_{0,3},\ a_{0,4},\ a_{0,5},\ a_{0,6},\ a_{0,7}$$

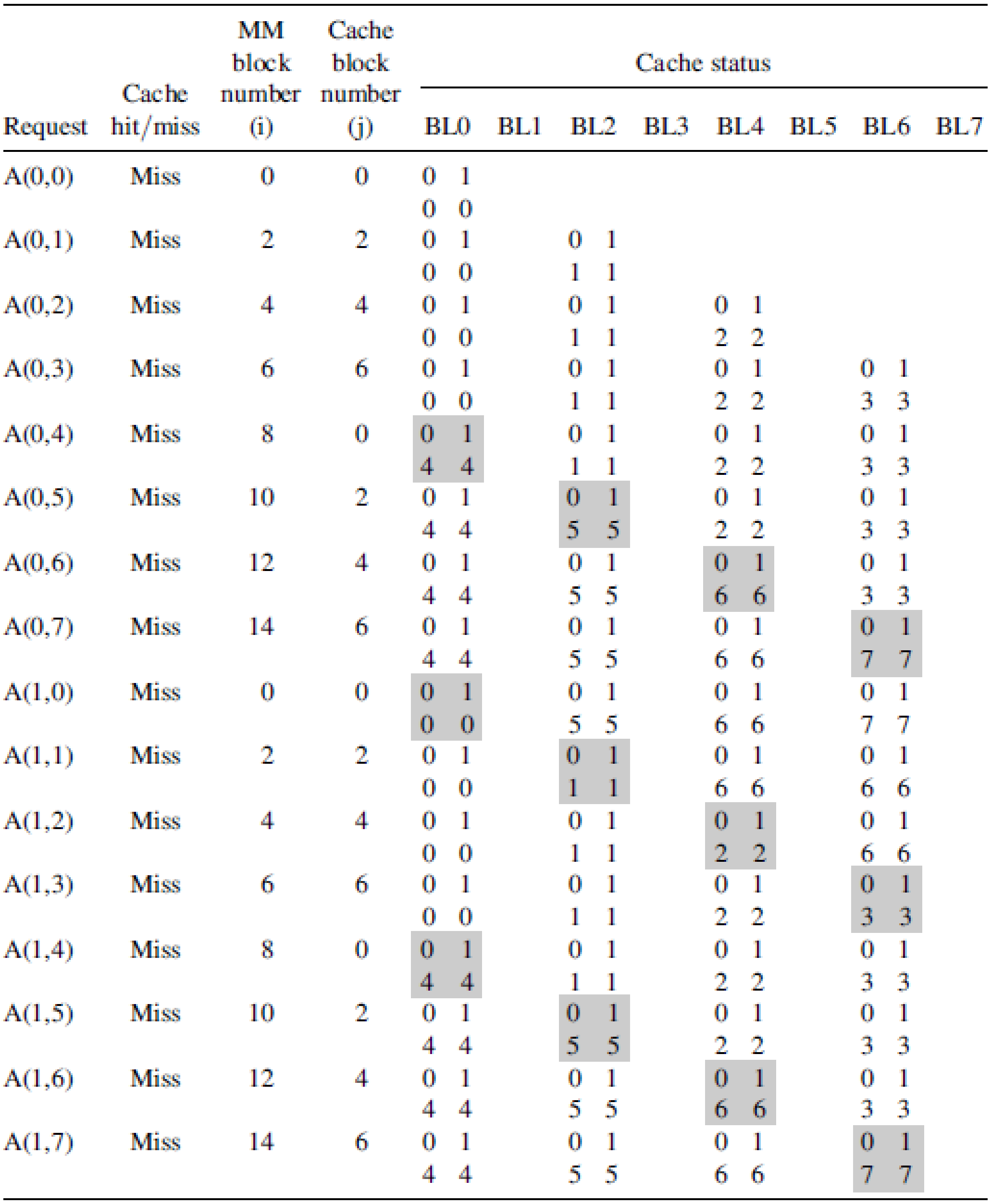
$$a_{1,0},\ a_{1,1},\ a_{1,2},\ a_{1,3},\ a_{1,4},\ a_{1,5},\ a_{1,6},\ a_{1,7}$$

Solution



Direct Mapping shows that there were 16 cache misses (not a single cache hit) and that the number of replacements made is 12 (these are shown tinted). It also shows that out of the available eight cache blocks, only four (0, 2, 4, and 6) are used, while the remaining four are inactive all the time. This represents a 50% cache utilization.

TABLE 6.3 Direct Mapping



Request	Cache hit/miss	MM block number (i)	Cache block number (j)	Cache status							
				BL0	BL1	BL2	BL3	BL4	BL5	BL6	BL7
A(0,0)	Miss	0	0	0 1 0 0							
A(0,1)	Miss	2	2	0 1 0 0		0 1 1 1					
A(0,2)	Miss	4	4	0 1 0 0		0 1 1 1		0 1 2 2			
A(0,3)	Miss	6	6	0 1 0 0		0 1 1 1		0 1 2 2		0 1 3 3	
A(0,4)	Miss	8	0	0 1 4 4		0 1 1 1		0 1 2 2		0 1 3 3	
A(0,5)	Miss	10	2	0 1 4 4		0 1 5 5		0 1 2 2		0 1 3 3	
A(0,6)	Miss	12	4	0 1 4 4		0 1 5 5		0 1 6 6		0 1 3 3	
A(0,7)	Miss	14	6	0 1 4 4		0 1 5 5		0 1 6 6		0 1 7 7	
A(1,0)	Miss	0	0	0 1 0 0		0 1 5 5		0 1 6 6		0 1 7 7	
A(1,1)	Miss	2	2	0 1 0 0		0 1 1 1		0 1 6 6		0 1 6 6	
A(1,2)	Miss	4	4	0 1 0 0		0 1 1 1		0 1 2 2		0 1 6 6	
A(1,3)	Miss	6	6	0 1 0 0		0 1 1 1		0 1 2 2		0 1 3 3	
A(1,4)	Miss	8	0	0 1 4 4		0 1 1 1		0 1 2 2		0 1 3 3	
A(1,5)	Miss	10	2	0 1 4 4		0 1 5 5		0 1 2 2		0 1 3 3	
A(1,6)	Miss	12	4	0 1 4 4		0 1 5 5		0 1 6 6		0 1 3 3	
A(1,7)	Miss	14	6	0 1 4 4		0 1 5 5		0 1 6 6		0 1 7 7	

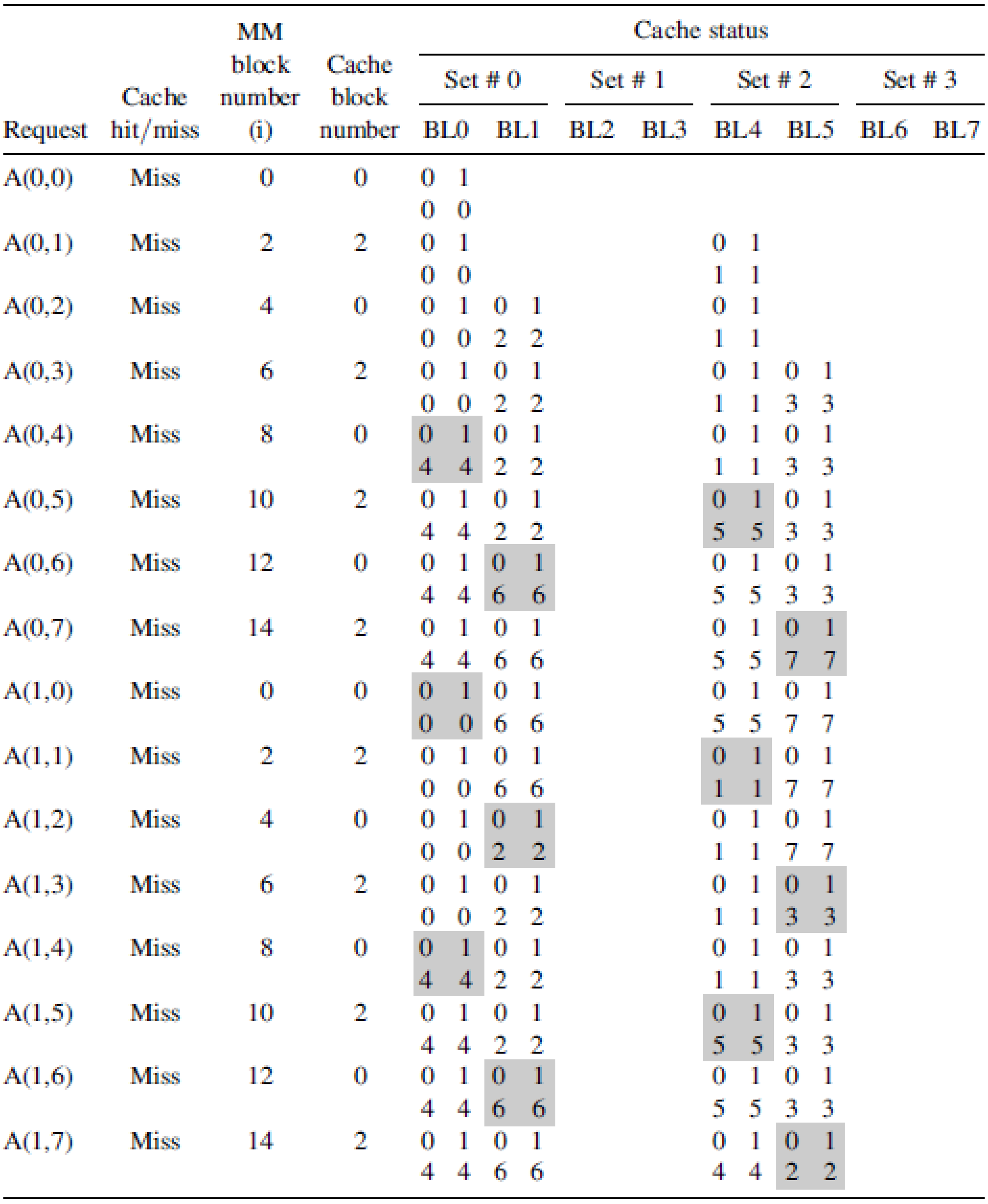
Fully Associative Mapping shows that there were eight cache hits (50% of the total number of requests) and that there were no replacements made. It also shows 100% cache utilization.

Set-Associative Mapping (With Two Blocks per Set) shows that there were 16 cache misses (not a single cache hit) and that the number of replacements made is 12 (these are shown tinted). It also shows that of the available four cache sets, only two sets are used, while the remaining two are inactive all the time. This represents 50% cache utilization.

TABLE 6.4 Fully Associative Mapping

		MM block number	Cache block	Cache status															
Request	Cache hit/miss	(i)	number	BL0		BL1		BL2		BL3		BL4		BL5		BL6		BL7	
A(0,0)	Miss	0	0	0	1														
				0	0														
A(0,1)	Miss	2	1	0	1	0	1												
				0	0	1	1												
A(0,2)	Miss	4	2	0	1	0	1	0	1										
				0	0	1	1	2	2										
A(0,3)	Miss	6	3	0	1	0	1	0	1	0	1								
				0	0	1	1	2	2	3	3								
A(0,4)	Miss	8	4	0	1	0	1	0	1	0	1	0	1						
				0	0	1	1	2	2	3	3	4	4						
A(0,5)	Miss	10	5	0	1	0	1	0	1	0	1	0	1	0	1				
				0	0	1	1	2	2	3	3	4	4	5	5				
A(0,6)	Miss	12	6	0	1	0	1	0	1	0	1	0	1	0	1	0	1		
				0	0	1	1	2	2	3	3	4	4	5	5	6	6		
A(0,7)	Miss	14	7	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
				0	0	1	1	2	2	3	3	4	4	5	5	6	6	7	7
A(1,0)	Hit	0	0	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
				0	0	1	1	2	2	3	3	4	4	5	5	6	6	7	7
A(1,1)	Hit	2	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
				0	0	1	1	2	2	3	3	4	4	5	5	6	6	7	7
A(1,2)	Hit	4	2	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
				0	0	1	1	2	2	3	3	4	4	5	5	6	6	7	7
A(1,3)	Hit	6	3	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
				0	0	1	1	2	2	3	3	4	4	5	5	6	6	7	7
A(1,4)	Hit	8	4	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
				0	0	1	1	2	2	3	3	4	4	5	5	6	6	7	7
A(1,5)	Hit	10	5	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
				0	0	1	1	2	2	3	3	4	4	5	5	6	6	7	7
A(1,6)	Hit	12	6	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
				0	0	1	1	2	2	3	3	4	4	5	5	6	6	7	7
A(1,7)	Hit	14	7	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
				0	0	1	1	2	2	3	3	4	4	5	5	6	6	7	7

TABLE 6.5 Set-Associative Mapping



		MM		Cache status							
		block	Cache	Set # 0		Set # 1		Set # 2		Set # 3	
	Cache	number	block								
Request	hit/miss	(i)	number	BL0	BL1	BL2	BL3	BL4	BL5	BL6	BL7
A(0,0)	Miss	0	0	0 1 0 0							
A(0,1)	Miss	2	2	0 1 0 0				0 1 1 1			
A(0,2)	Miss	4	0	0 1 0 0	0 1 2 2			0 1 1 1			
A(0,3)	Miss	6	2	0 1 0 0	0 1 2 2			0 1 1 1	0 1 3 3		
A(0,4)	Miss	8	0	0 1 4 4	0 1 2 2			0 1 1 1	0 1 3 3		
A(0,5)	Miss	10	2	0 1 4 4	0 1 2 2			0 1 5 5	0 1 3 3		
A(0,6)	Miss	12	0	0 1 4 4	0 1 6 6			0 1 5 5	0 1 3 3		
A(0,7)	Miss	14	2	0 1 4 4	0 1 6 6			0 1 5 5	0 1 7 7		
A(1,0)	Miss	0	0	0 1 0 0	0 1 6 6			0 1 5 5	0 1 7 7		
A(1,1)	Miss	2	2	0 1 0 0	0 1 6 6			0 1 1 1	0 1 7 7		
A(1,2)	Miss	4	0	0 1 0 0	0 1 2 2			0 1 1 1	0 1 7 7		
A(1,3)	Miss	6	2	0 1 0 0	0 1 2 2			0 1 1 1	0 1 3 3		
A(1,4)	Miss	8	0	0 1 4 4	0 1 2 2			0 1 1 1	0 1 3 3		
A(1,5)	Miss	10	2	0 1 4 4	0 1 2 2			0 1 5 5	0 1 3 3		
A(1,6)	Miss	12	0	0 1 4 4	0 1 6 6			0 1 5 5	0 1 3 3		
A(1,7)	Miss	14	2	0 1 4 4	0 1 6 6			0 1 4 4	0 1 2 2		

Writing into Cache

If the operation is a write, there are two ways that the system can precede. The simplest and most commonly used procedure is to update main memory with every memory write operation, with cache memory being updated in parallel if it contains the word at the specified address. This is called the *write-through* method. The advantage that the main memory always contains the same data as the cache.

The second procedure is called the write-back method. In this method only the cache location is updated during a write operation. The location is then marked by a flag so that later when the word is removed from the cache it is copied into main memory. The reason to reduce the number of update time for the same main memory location.

5.2.4 Virtual Memory

Address Space and Memory Space: An address used by a programmer will be called a virtual address, and the set of such addresses the address space. An address in main memory is called a location or physical address. The set of such locations is called the memory space.

Example: N=1024K and M =32K.

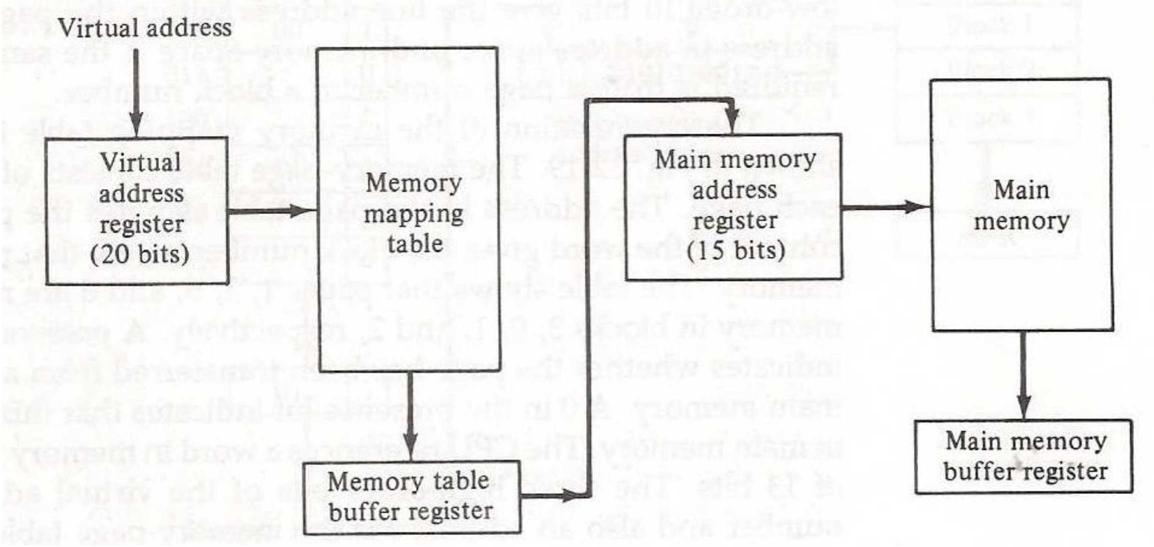

Virtual address
Virtual address register (20 bits)
Memory mapping table
Memory table buffer register
Main memory address register (15 bits)
Main memory
Main memory buffer register

Figure 9: Virtual Memory

Address Mapping Using Pages

The physical memory is broken down into groups of equal size called *blocks*, which may range from 64 to 4096 words each. The term page refers to groups of address space of the same size. Example: if a page or block consists of 1K words, then, address space is divided into 1024 pages, and main memory divided into 32 blocks.

In a computer with 2^p words per page, p bits are used to specify a line address and the remaining high-order bits of the virtual address specify the page number. Example: N = 8K and M = 4K

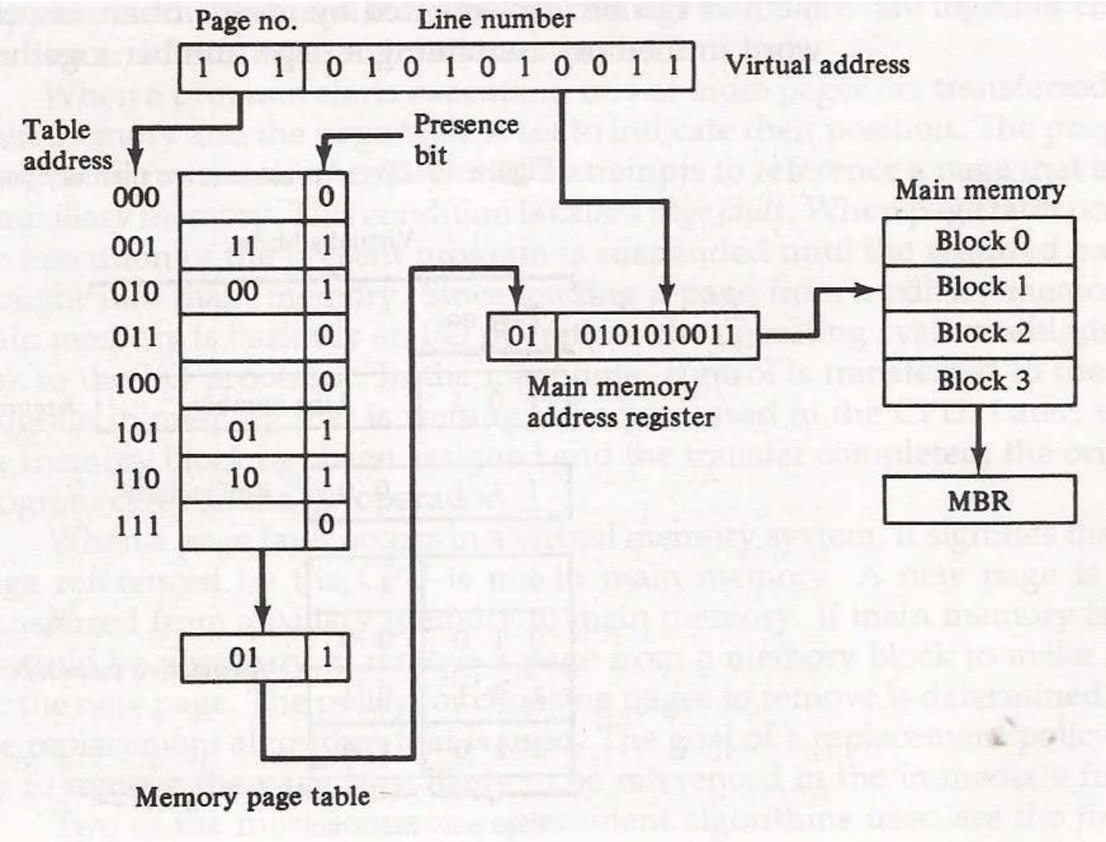


Figure 10: Address mapping using pages

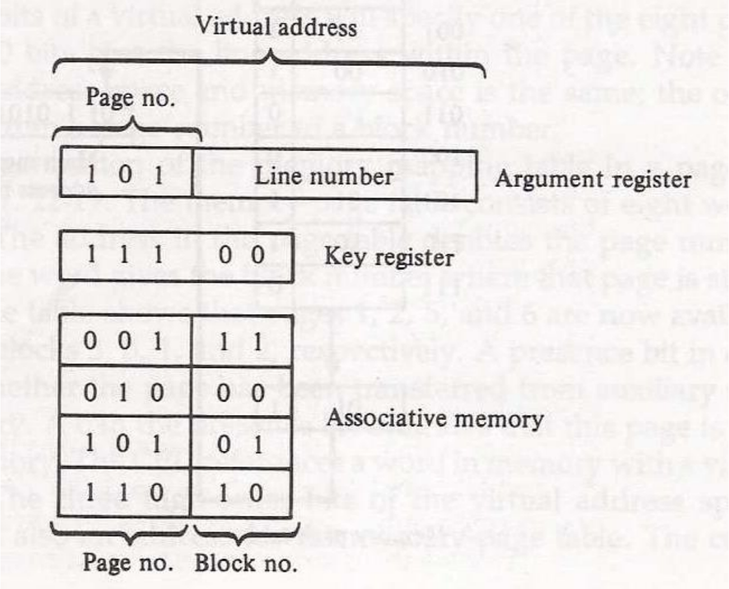
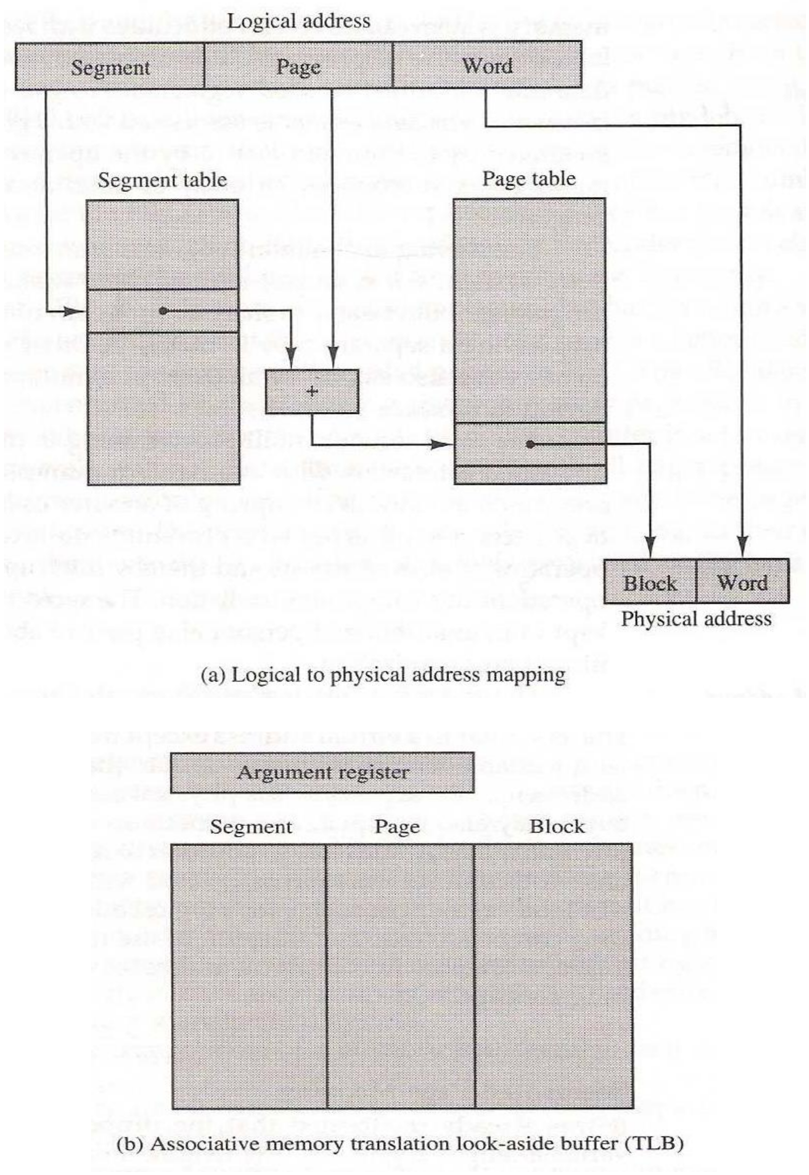


Figure 11: Associative Memory Page Table

Memory Management

Memory management system is a collection of hardware and software procedures for managing the various programs residing in memory. It is more convenient to divide programs and data into logical parts called *segments.* A segment is a set of logically related instructions or data elements associated with a given name. Segments may be generated by the programmer or by the operating system.

The address generated by a segmented program is called a *logical address*. This is similar to a virtual address except that logical address space is associated with variable-length segments rather that fixed length pages.

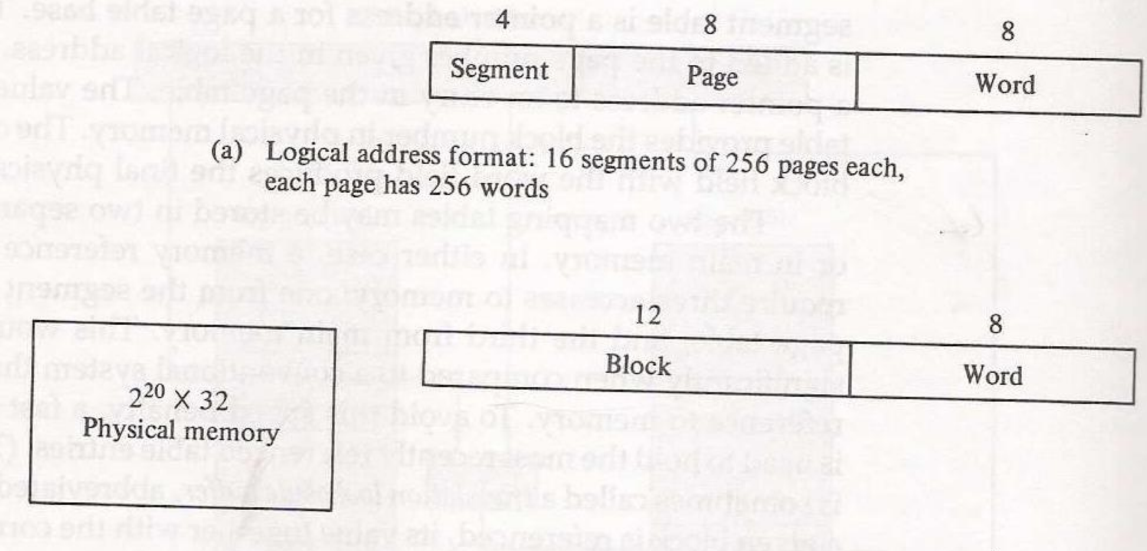


(a) Logical to physical address mapping

(b) Associative memory translation look-aside buffer (TLB)

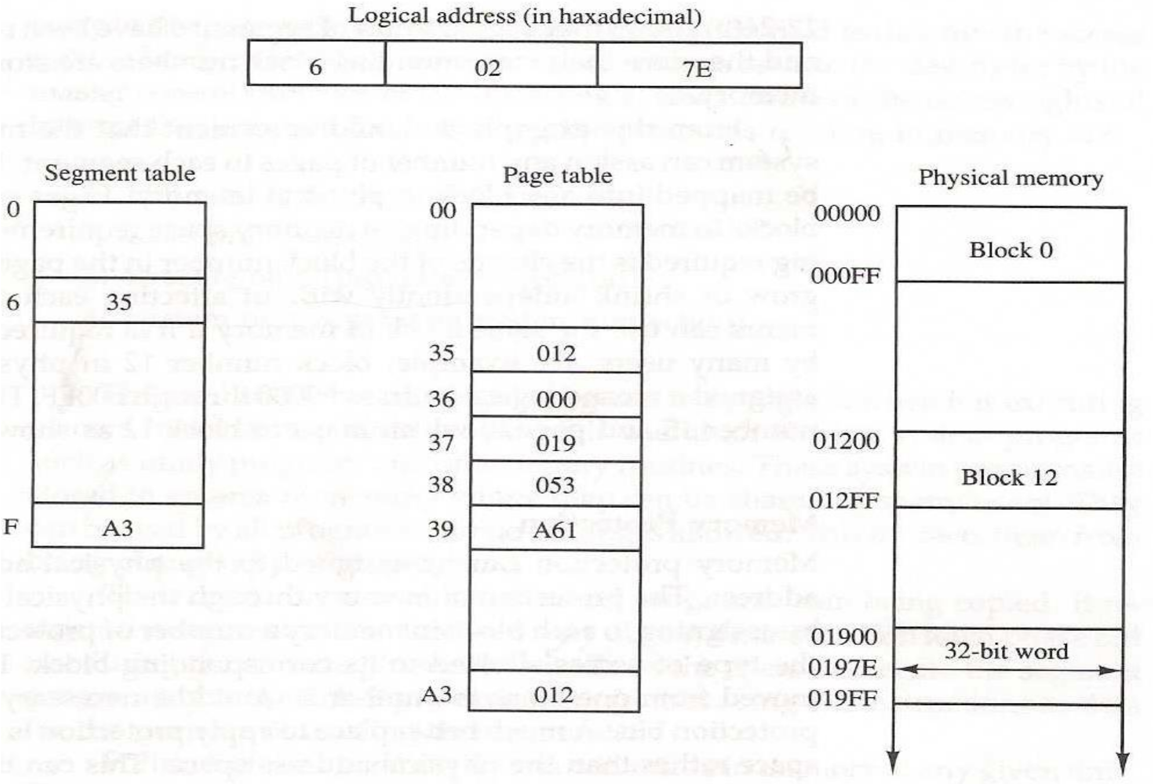
Figure 12: Segmented-page Mapping

Example



(a) Logical address format: 16 segments of 256 pages each, each page has 256 words

(b) Physical address format: 4096 blocks of 256 words each, each word has 32 bits


Logical address (in haxadecimal)
6
02
7E
Segment table
0
6
35
F
A3
Page table
00
35
012
36
000
37
019
38
053
39
A61
A3
012
Physical memory
00000
Block 0
000FF
01200
Block 12
012FF
01900
0197E
32-bit word
019FF