Unit – 9

Memory Organization:

Memory :

- Physical device
- Used for storage
- Essential component - any digital computer
- Needed for storing Program and Data
- Most of computer require large amount of data
- Not all accumulated information needed by the **processor at same time**.
- Most economical to use low-cost storage device – for backup - **Not currently used by CPU**
- **Why need of memory hierarchy ?**

Memory Hierarchy :

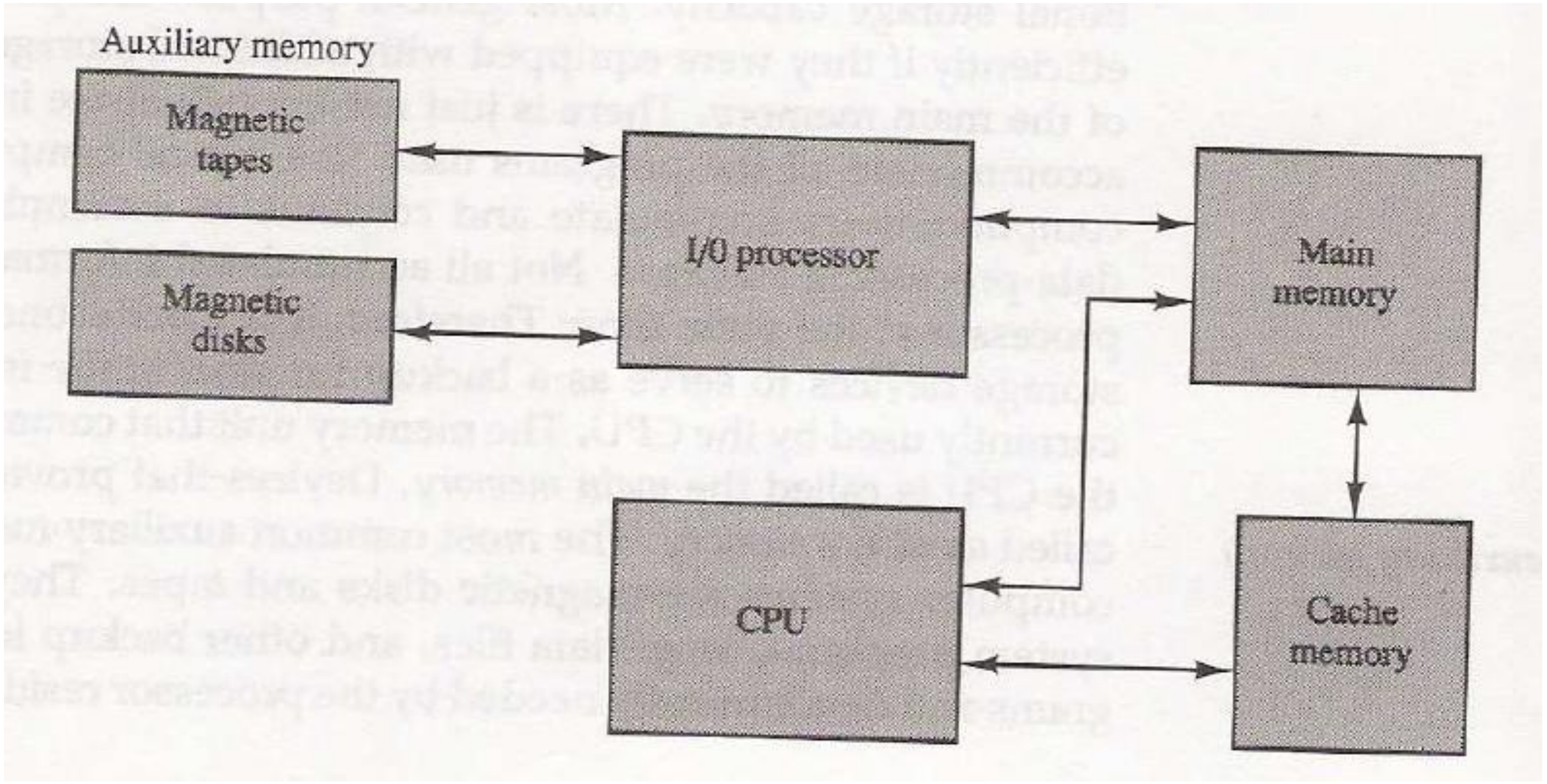


Fig - Memory Hierarchy in Computer System

Memory Hierarchy :

- The memory unit that communicates directly with the CPU is called the **Main memory**.

  Ex – RAM & ROM

- Devices that provide backup storage are called **Auxiliary Memory**.

  Ex – Magnetic disks & Tapes

- Only programs and data currently needed by the processor resides in main memory.
- Other information stored in auxiliary memory.
- Total memory capacity of computer can be visualised as being a hierarchy of components.

Memory Hierarchy :

- The slow but high capacity – Auxiliary memory
- To relative faster – Main Memory
- Even Smaller & Faster – Cache memory accessible to the high processing logic.

- **Main Memory** occupies a central position - able to direct communicate with CPU & Auxiliary memory through an **I/O Processor.**
- **CPU** need - program & Data transfer Auxiliary to Main memory
- **CPU** not need - program & Data transfer Main memory to Auxiliary.

Memory Hierarchy :

- **Cache Memory –** A special very high speed memory called cache memory.
- Increase the speed of processing.
- To compensate for the speed differential between main memory access time & processor logic.
- Small Cache between CPU and Main Memory.
- Whose access time is close to processor logic clock cycle time.
- **I/O Processor** – manage data transfer between Auxiliary to MM.
- **Cache organization** – transfer of information between MM to CPU.
- Each is involved with different level in memory hierarchy.

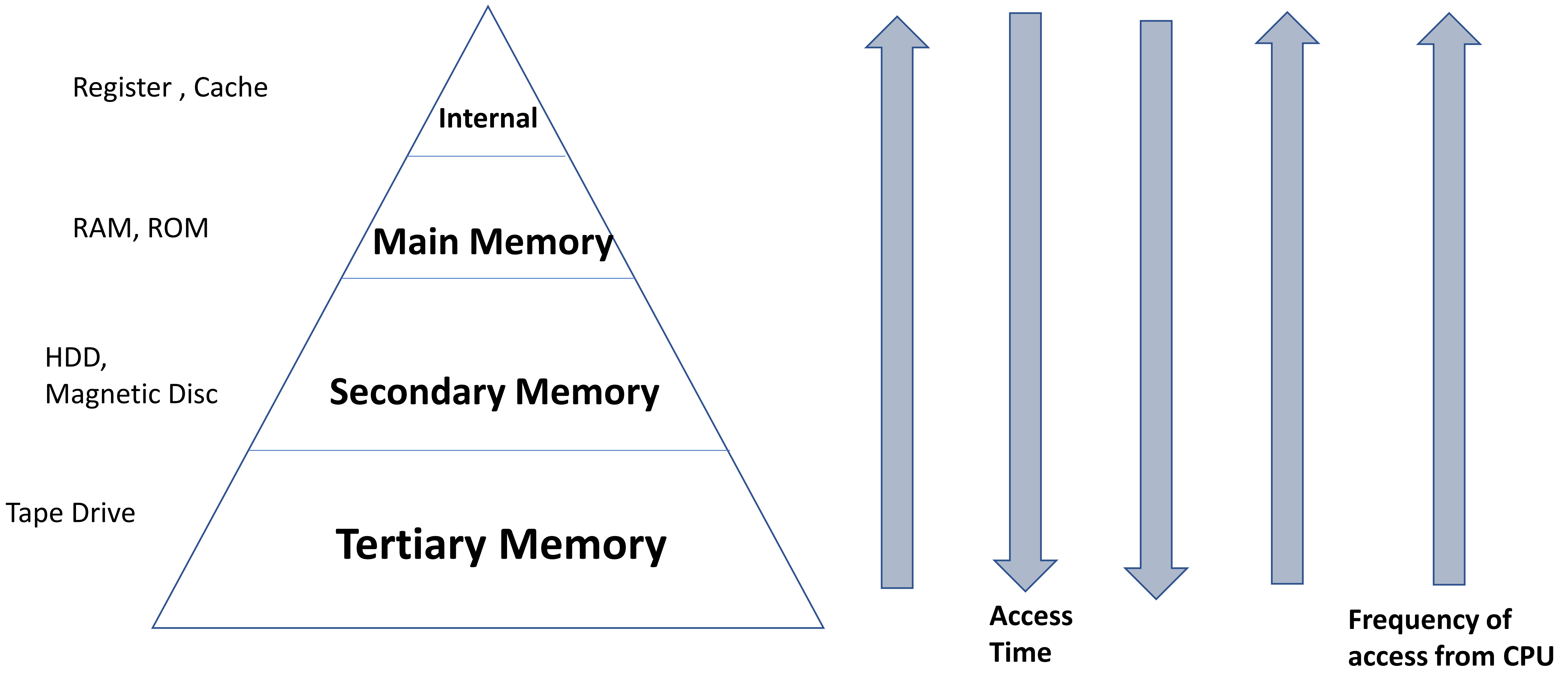
Memory Hierarchy :

- The Reason having 2 or 3 level of memory hierarchy is **economics.**

Goal of Memory Hierarchy :

1. To maximize the access speed
2. To minimize the per bit storage cost

Memory Hierarchy :



Average cost of Memory :

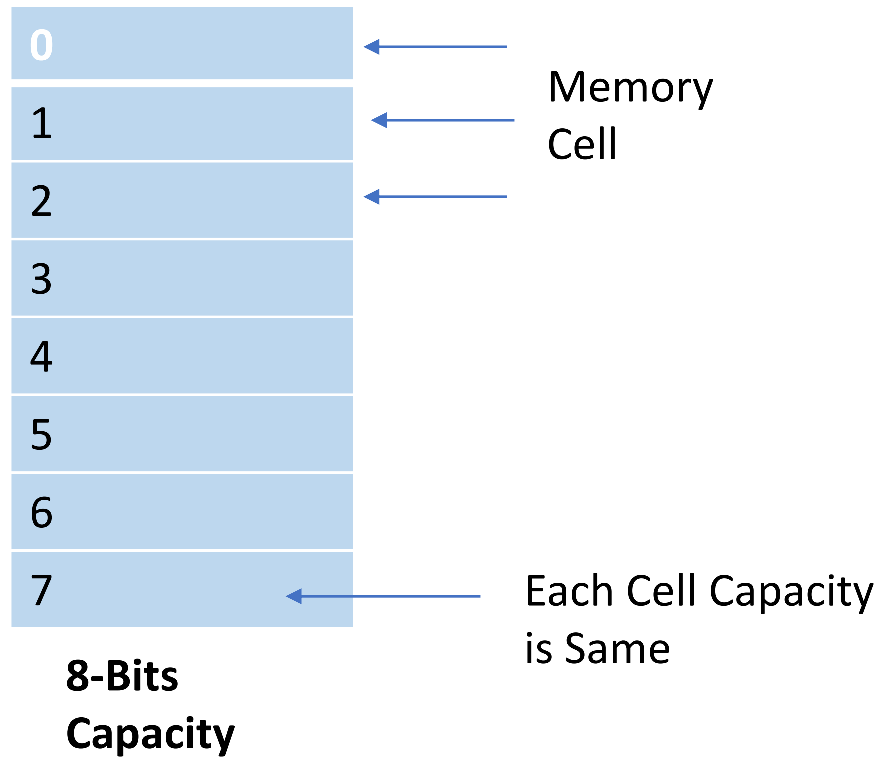
Memory	Size In Bits	Per Bit Storage Cost	Cost of a Memory
M1	S1	C1	S1*C1
M2	S2	C2	S2*C2
M3	S3	C3	S3*C3
TOTAL	**S1+S2+S3**		**S1*C1+S2*C2+S3*C3**

$$\text{Average Per bit Memory Cost} = \frac{\text{Total Cost}}{\text{Total Size}}$$

$$= \frac{\mathbf{S1*C1+S2*C2+S3*C3}}{\mathbf{S1+S2+S3}}$$

Memory Representation :

- Memory represent by = No. of Cell * 1 Cell Capacity



1) How many Cell you have?
2) Capacity of Each Cell?
3) What is the total capacity?

Total = 8*8bits = 64-bit storage capacity

Memory Representation :

- How to access each & every cell uniquely?
- CPU wants to perform some operation on this chip how to identify uniquely any cell.

Address No	**Cell No**	**8- bits**
00	0	
01	1	
10	2	
11	3	

This cell no is called address & write in binary number format.

Using this address Cell is identify uniquely.

Total = 32-bit Storage Capacity

Memory Representation :

Address No	**Cell No**	16-bits
000	0	
001	1	
010	2	
011	3	
100	4	
101	5	
110	6	
111	7	

8 cell – 8 different address location

Total Memory = No. of Cell * 1 Cell Capacity
= No of Memory Location * Bits per Location

Total = 8*16-bit Storage Capacity

Memory Representation :

No of cell	$4 = 2^2$	$8=2^3$	$16=2^4$	2^x	n
Address size	2-bits	3-bits	8-bits	X-bits	$\text{Log}_2 n$ bits

128 * 8-bits

128*1B

128B

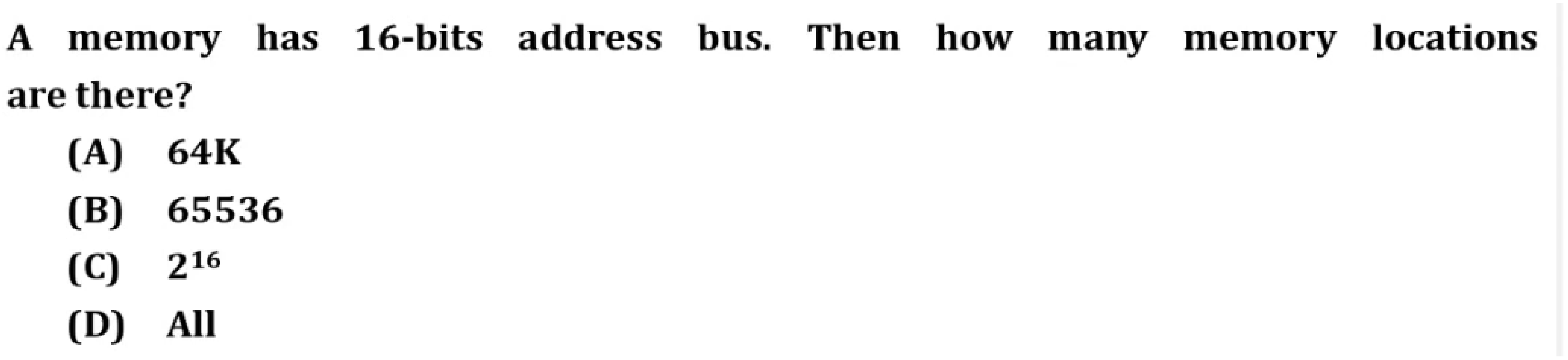
If 4GB Memory Then 4G*1B

$= 2^2 * 2^{30} * 1B$

$= 2^{32}$ B

So address line = 32 bits

Byte addressable memory means store 1 Byte at each add.


A memory has 16-bits address bus. Then how many memory locations are there?

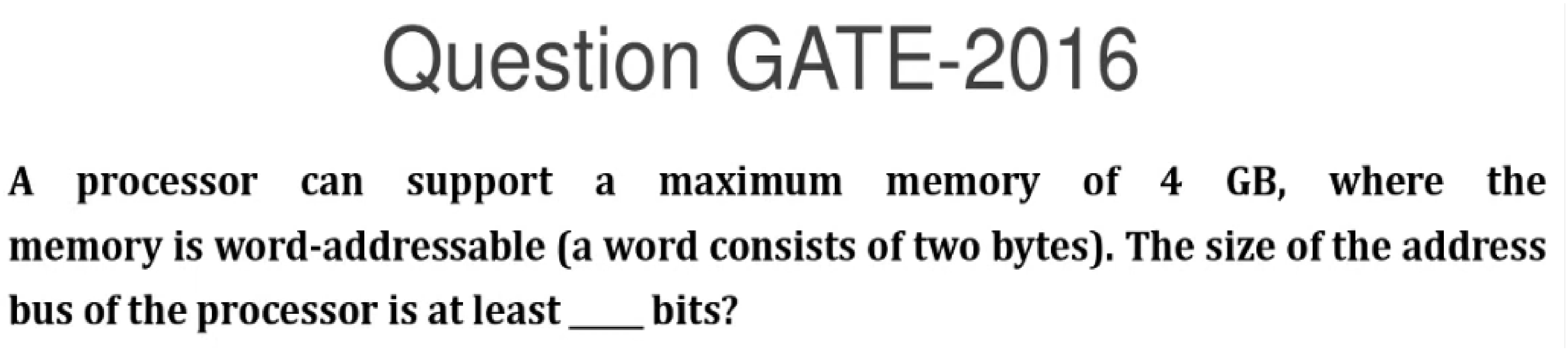
(A) 64K

(B) 65536

(C) 2^{16}

(D) All

In same question, if at each location 16-bits can be stored, then total memory capacity is how much?


Question GATE-2016

A processor can support a maximum memory of 4 GB, where the memory is word-addressable (a word consists of two bytes). The size of the address bus of the processor is at least ____ bits?

Consider a CPU which has 12-bits address bus and 16-bits data bus. Maximum size memory which can be supported by CPU is _____ Kbytes.

Main Memory :

- The memory used to store current running program (instructions) and their data.
- The main memory is the central storage unit in a computer system.
- Primary memory holds only those data and instructions on which computer is currently working.
- It has limited capacity and data is lost when power is switched off.
- It is generally made up of semiconductor device.
- These memories are not as fast as registers.
- The data and instruction required to be processed reside in main memory.

- **Two types of Main Memory :**

1. RAM – Random Access Memory
2. ROM – Read Only Memory

RAM : Those programs whose are execute (running) currently stored in
RAM. So why ROM is in main memory?

Characteristic : ROM – Non Volatile
RAM – Volatile → When power cut then data erased

- When you turn your computer from where can you take the instruction?
- CPU will first execute the **initial program** from ROM.
- The **bootstrap loader** is a program whose function is to start the computer software operating when power is turned on.

- CPU execute the instruction – 2 operation perform

1. Hardware check (POST – power on self test)
2. Booting Process – Bootstrap program

   OS form secondary memory to RAM

- Types of RAM – SRAM & DRAM

Static RAM	**Dynamic RAM**
Made up of : Flip-flop Bits are storage in voltage form	Made up of : Capacitor Bits are storage in form of electric energy
No Refresh	Periodic refresh is required
Faster Access speed	Slower Access speed
More Expensive	Less Expensive
Ex – Cache Memory – Level 1 & 2	Ex – Main Memory
Power consumption – Less	Power consumption – More
Physical placement – Processor Or between processor and main memory	Physical placement – Motherboard

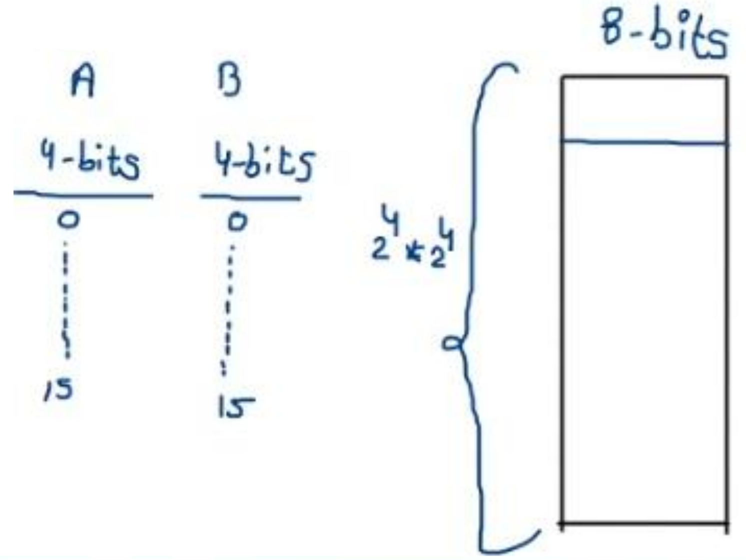
The amount of ROM needed to store the table for multiplication of two 4-bit unsigned integer is?

(A) 64 bits

(B) 128 bits

(C) 1K bits

(D) 2K bits



$2^8 \times 8$-bits

2^{11} bits

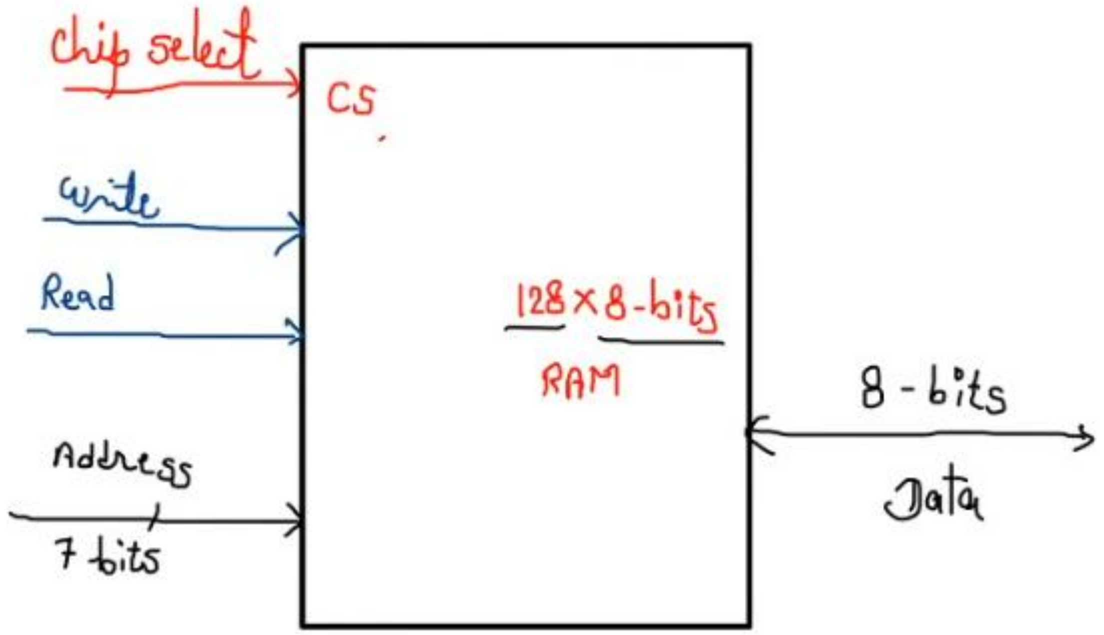
2K bits

A	B	A * B result	A * B table	A + B result	A + B table
4	4	8-bits	$2^8 \times 8$-bits	5-bits	$2^8 \times 5$-bits
n	n	2n-bits	$2^{2n} \times 2n$-bits	(n+1)-bits	$2^{2n} \times (n+1)$-bits

The amount of ROM needed to store the table for multiplication of two 8-bit unsigned integer is?

RAM Chip :

- How particular RAM chip will be access?



Address Bus : 128 = 2^7
So 7-bit address line

Data bus – 8—bits
1 line carry 1-bit only

2 Control Signal – 1. Read
2. Write

CS – Chip Select
Which enables the RAM Chip

Because minimum 2 chips required
(RAM & ROM)

CS	**RD**	**WR**	**Operation**
0	X	X	No Operation
1	0	0	No Operation
1	0	1	Write
1	1	X	Read only

X – Don’t care

Special case when Read is 1 then Write is 1 or 0 Whatever I don’t care I will perform Read only in chip.

Read & Write both are enable who can conflict the system – Write only (Read can not create any problem)

So RD & WR enable then read operation only perform.

- How many no of line (pin) required in this Chip.

No of Pins :

Address line : 7

Data line – 8

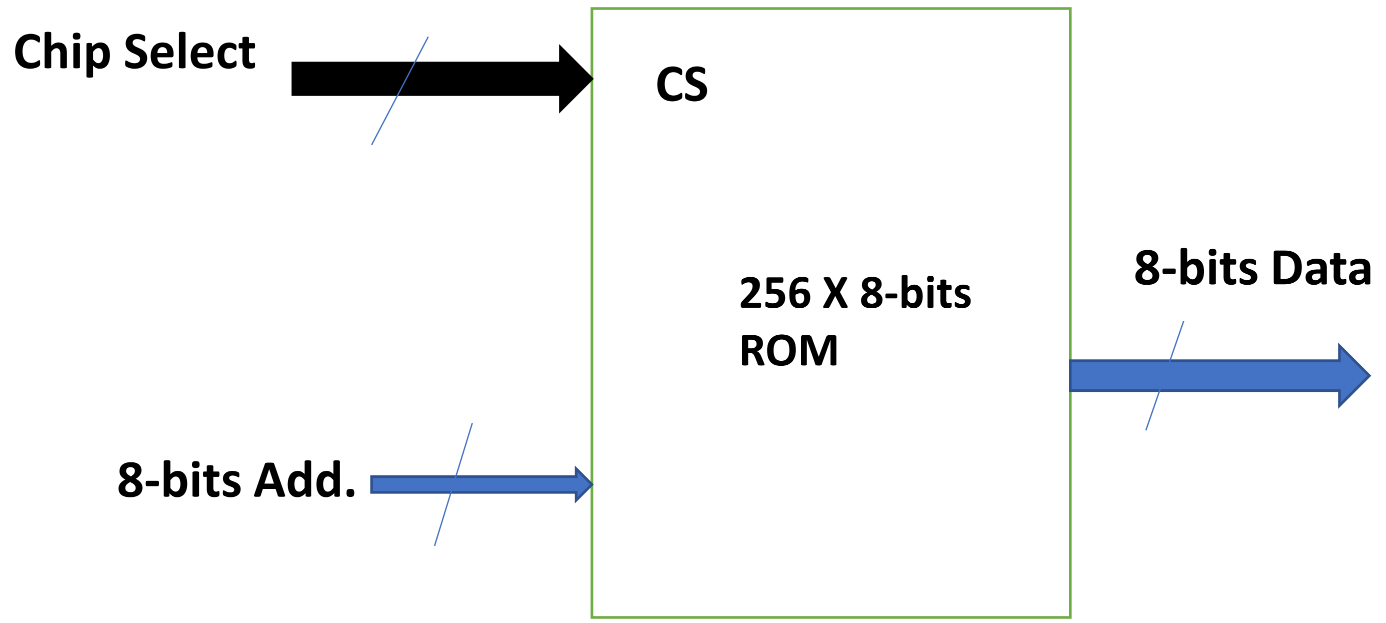
CS – 1

WR – 1

RD – 1

Total = 18 pins required to this chip design

ROM Chip :



CS	Operation
0	No operation
1	Read

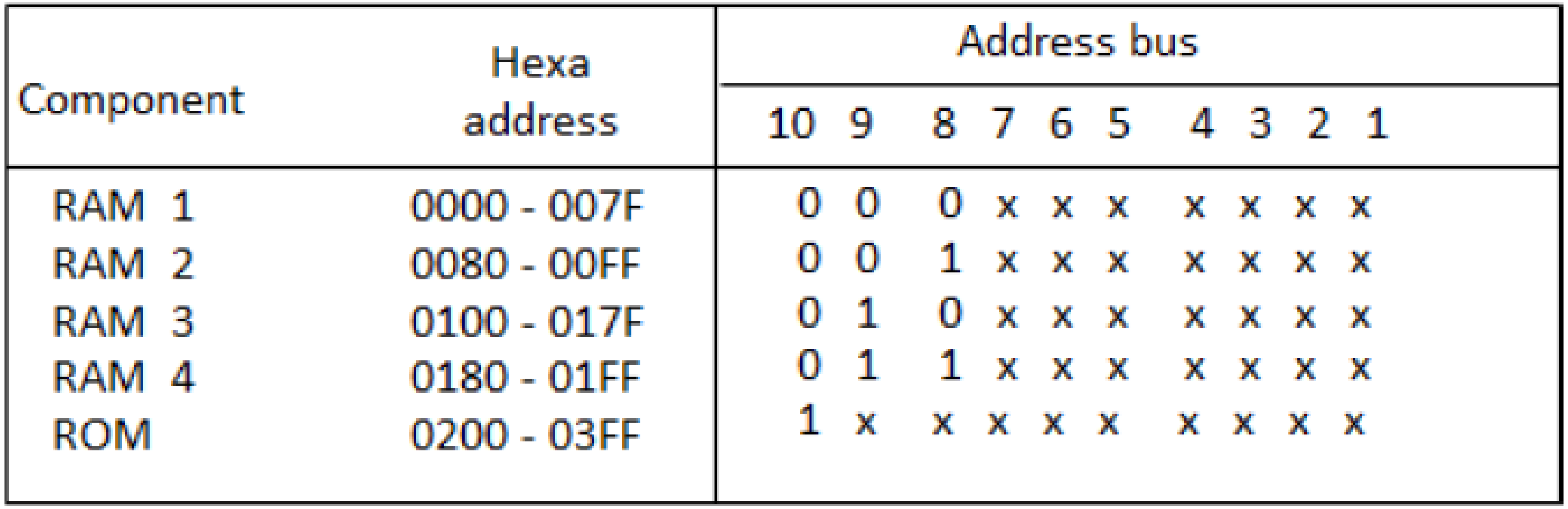
How many pins are required?

Address Line – 8
Data line -8
CS -1

Total = 17 Pins

RAM Chip :

- How particular RAM chip will be access?
- The designer of a computer system must calculate the amount of memory required for the particular application and assign it to either RAM or ROM.
- The interconnection between memory and processor is then established from knowledge of the size of memory needed and the type of RAM and ROM chips available.
- The addressing of memory can be established by means of a table that specifies the memory address assigned to each chip.
- The table, called a **memory address map**, is a pictorial representation of assigned address space for each chip in the system, shown in table.
- To demonstrate with a particular example, assume that a computer system needs 512 bytes of RAM and 512 bytes of ROM.



Component	Hexa address	Address bus									
		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

Table 9.1: Memory Address Map for Micro-procomputer

- The component column specifies whether a RAM or a ROM chip is used.
- The hexadecimal address column assigns a range of hexadecimal equivalent addresses for each chip.
- The address bus lines are listed in the third column.
- Although there are 16 lines in the address bus, the table shows only 10 lines because the other 6 are not used in this example and are assumed to be zero.

- The small x's under the address bus lines designate those lines that must be connected to the address inputs in each chip.
- The RAM chips have 128 bytes and need seven address lines. The ROM chip has 512 bytes and needs 9 address lines.
- The x's are always assigned to the low-order bus lines: lines 1 through 7 for the RAM and lines 1 through 9 for the ROM.
- It is now necessary to distinguish between four RAM chips by assigning to each a different address. For this particular example we choose bus lines 8 and 9 to represent four distinct binary combinations.
- The table clearly shows that the nine low-order bus lines constitute a memory space for RAM equal to 2^9 = 512 bytes.
- The distinction between a RAM and ROM address is done with another bus line. Here we choose line 10 for this purpose.
- When line 10 is 0, the CPU selects a RAM, and when this line is equal to 1, it selects the ROM.

A ROM chip of 1024×8 bits has four select inputs and operates from a 5-volt power supply. How many pins are needed for the IC package?

Address – 10
Data – 8
CS -4
Power – 1
Ground -1

Total = 24 Pins

Multiple Chip Support :

- Suppose in your computer system if you are using 2-chip of RAM & each RAM has 2-2 chips.
- So how much total capacity of RAM? → 4-chips
- 4-chip have – 2-2 chip → 4*2 = 8 – chips

- If you are using particular chip of 2GB only concern with main memory in your system.
- But you have total capacity of 16 GB multiple chip. So how many chips are required?
- Total Mem Capacity = No. of chips * 1 Chip Capacity

- Assume, we have two chips

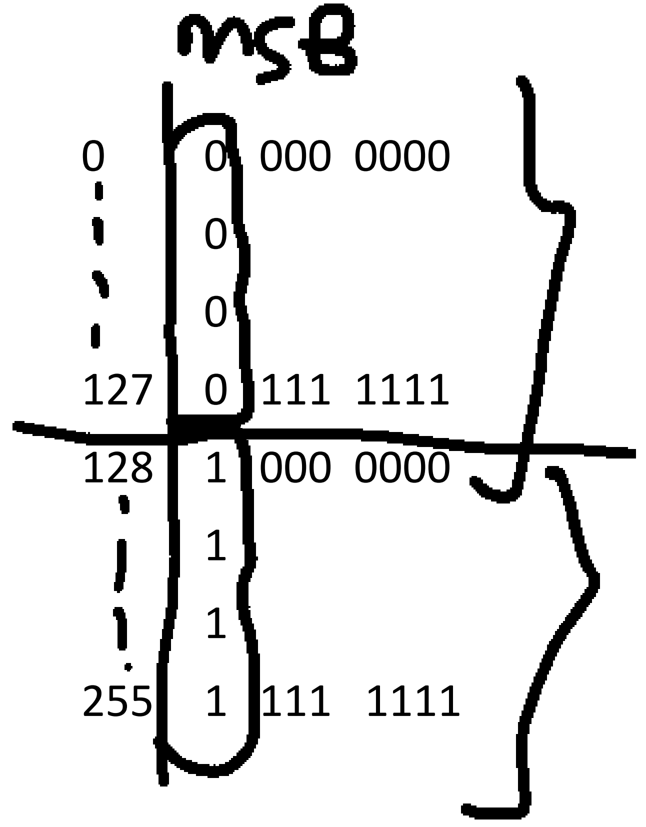
RAM 0 – 128 x 8-bits ➔ 7 –bits add

RAM 1 – 128 x 8-bits ➔ 7 –bits add

Total - 256 x 8- bits ➔ 8-bits add – CPU will generate 8-bit address

Each chip has 7-bits add. But multiple chips are there. So how address multiple chips.

CPU generate 8-bits address from 0000 0000 – 1111 1111 range.



Divided this add range into 2 equal parts.

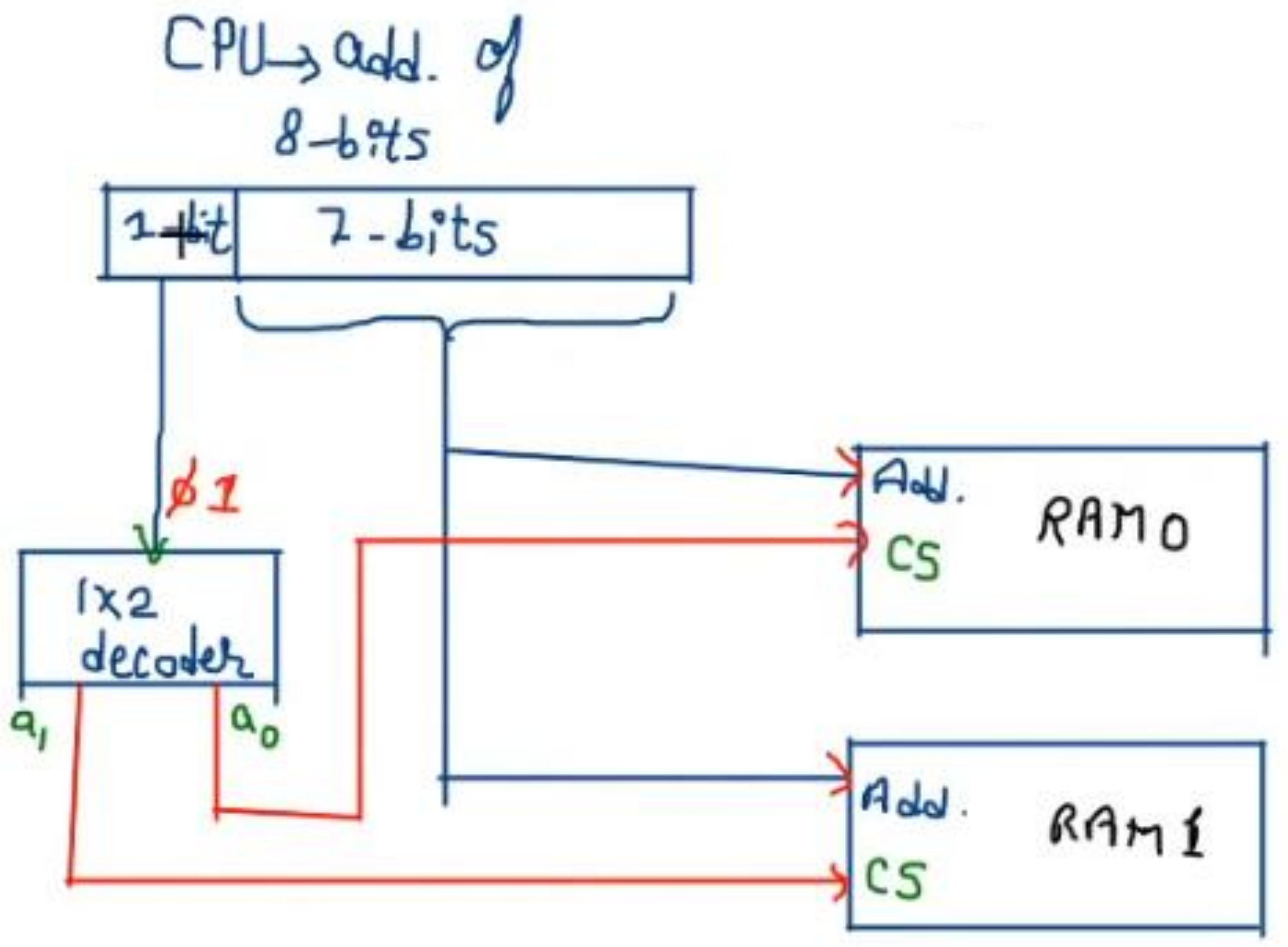
When CPU generate any starting 128 add. Then going to access RAM 0. (0 ….. 127 address)

When CPU generate second half add. then going to access RAM 1. (128 ….. 255 address)

How to identify this thing ? → Using MSB –bits

If CPU generate any address & if it MSB bit is 0 then RAM-0 will be access & if MSB bit is 1 then RAM-1 is access.

Same thing using Hardware :- (Vertical Arrangement)



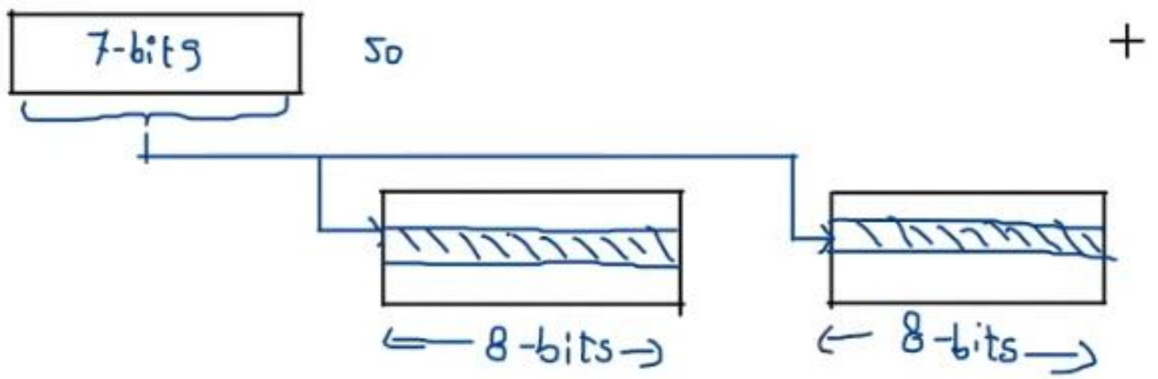
(a) How many 128×8 RAM chips are needed to provide a memory capacity of 2048 bytes?

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

(c) How many lines must be decoded for chip select? Specify the size of decoder?

How many 128×8 bits RAM chips are needed to provide a memory capacity of 128×16 bits?

(Horizontal Arrangement)



If required Address are more than chip -1 address ➔ Vertical Arrangement

If required Data are more than chip -1 data ➔ Horizontal Arrangement

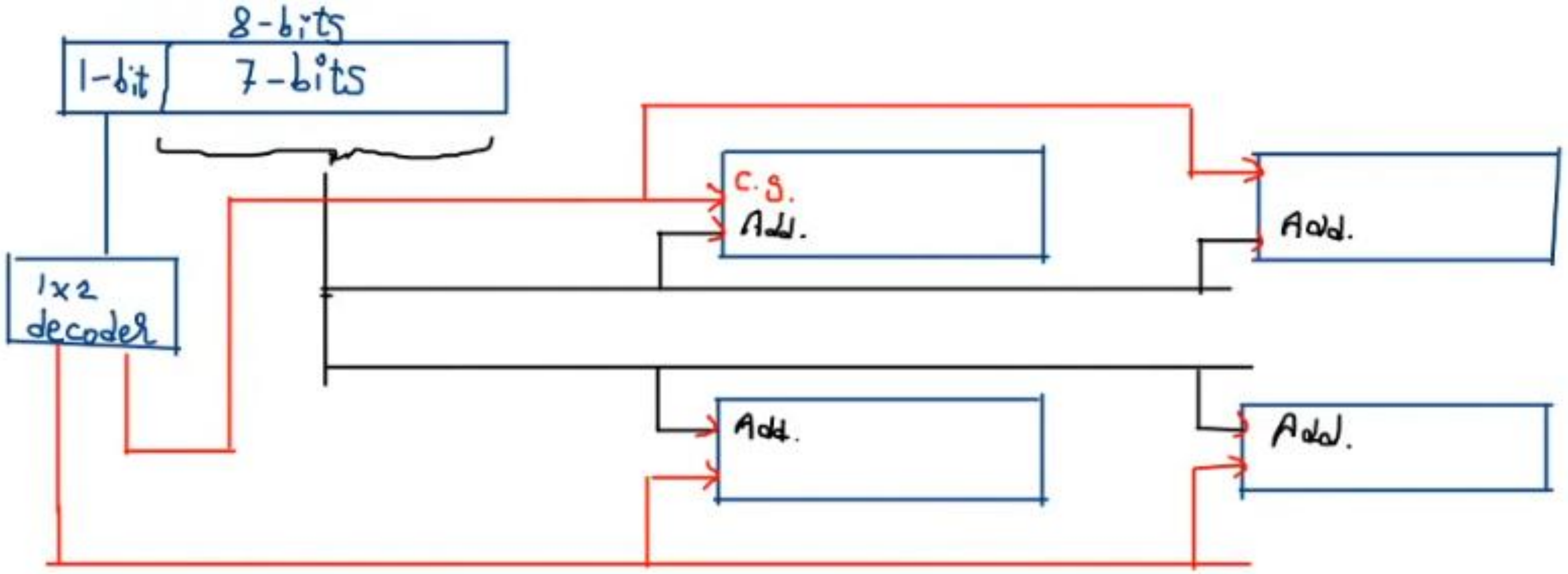
If Both are more than chip -1 ➔ Horizontal & Vertical Arrangement both apply (Hybrid)

How many 128×8 bits RAM chips are needed to provide a memory capacity of 256×16 bits?

256 / 128 = 2

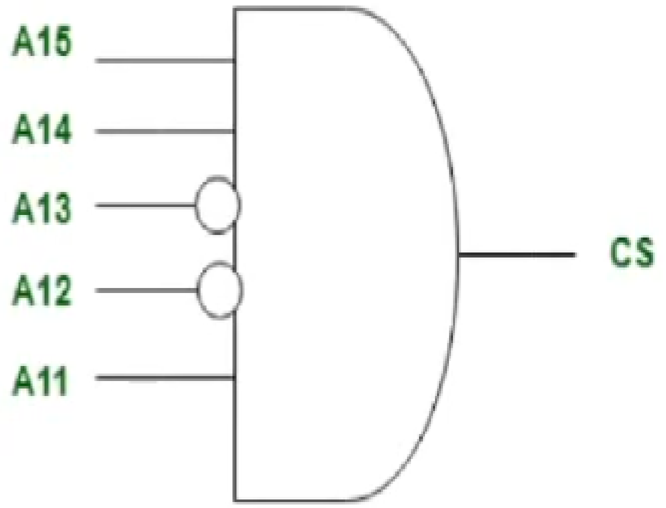
16/8 = 2

2 x 2 = 4 chip required



The chip select logic for a certain DRAM chip in a memory system design is shown below. Assume that the memory system has 16 address lines denoted by A15 to A0. What is the range of address (in hexadecimal) of the memory system that can get enabled by the chip select (CS) signal?

(A) C800 to CFFF (B) CA00 to CAFF
(C) C800 to C8FF (D) DA00 to DFFF



Question GATE-2009

+

How many 32K×1 RAM chips are needed to provide a memory capacity of 256KBytes ?

(A) 8 (B) 32
(C) 64 (D) 128

Associative Memory

- It is known as **Content Addressable Memory (CAM).**
- Cell do not have addresses.

- A memory unit accessed by content is called an associative memory or content addressable memory (CAM).
- This type of memory is accessed simultaneously and in parallel on the basis of data content rather than by specific address or location.
- The time required to find an item stored in memory can be reduced considerably if stored data can be identified for access by the content of the data itself rather than by an address.
- It consists of a memory array and logic form words with n bits per word.

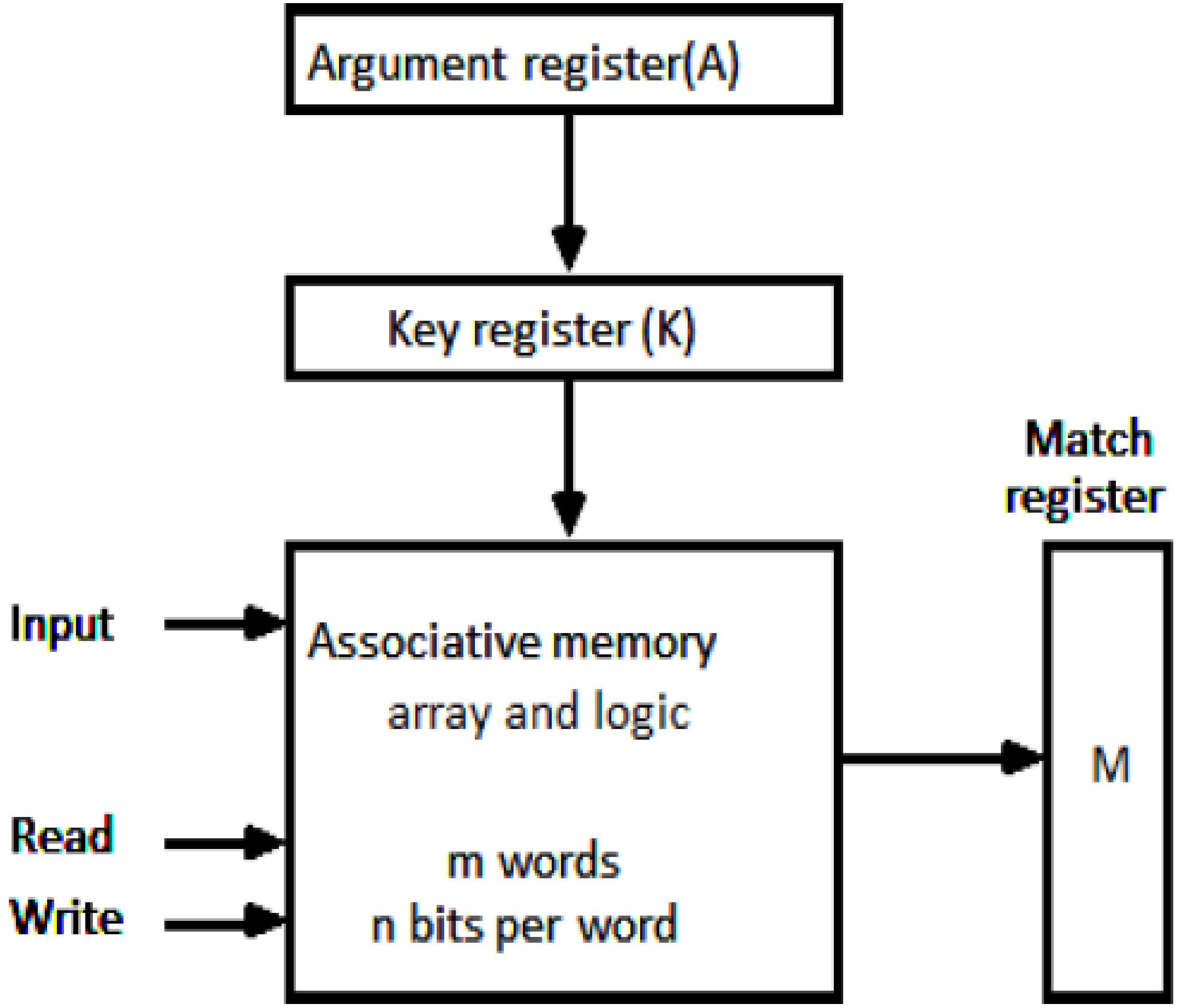


Figure 9.3: Block diagram of associative memory

- The argument register A and key register K each have n bits, one for each bit of a word.
- The match register M has m bits, one for each memory word.
- Each word in memory is compared in parallel with the content of the argument register.
- The words that match the bits of the argument register set a corresponding bit in the match register.
- After the matching process, those bits in the match register that have been set indicate the fact that their corresponding words have been matched.
- Reading is accomplished by a sequential access to memory for those words whose corresponding bits in the match register have been set.

Hardware Organization

A	101 111100	
K	111 000000	
$Word_1$	100 111100	no match
$Word_2$	101 000001	match

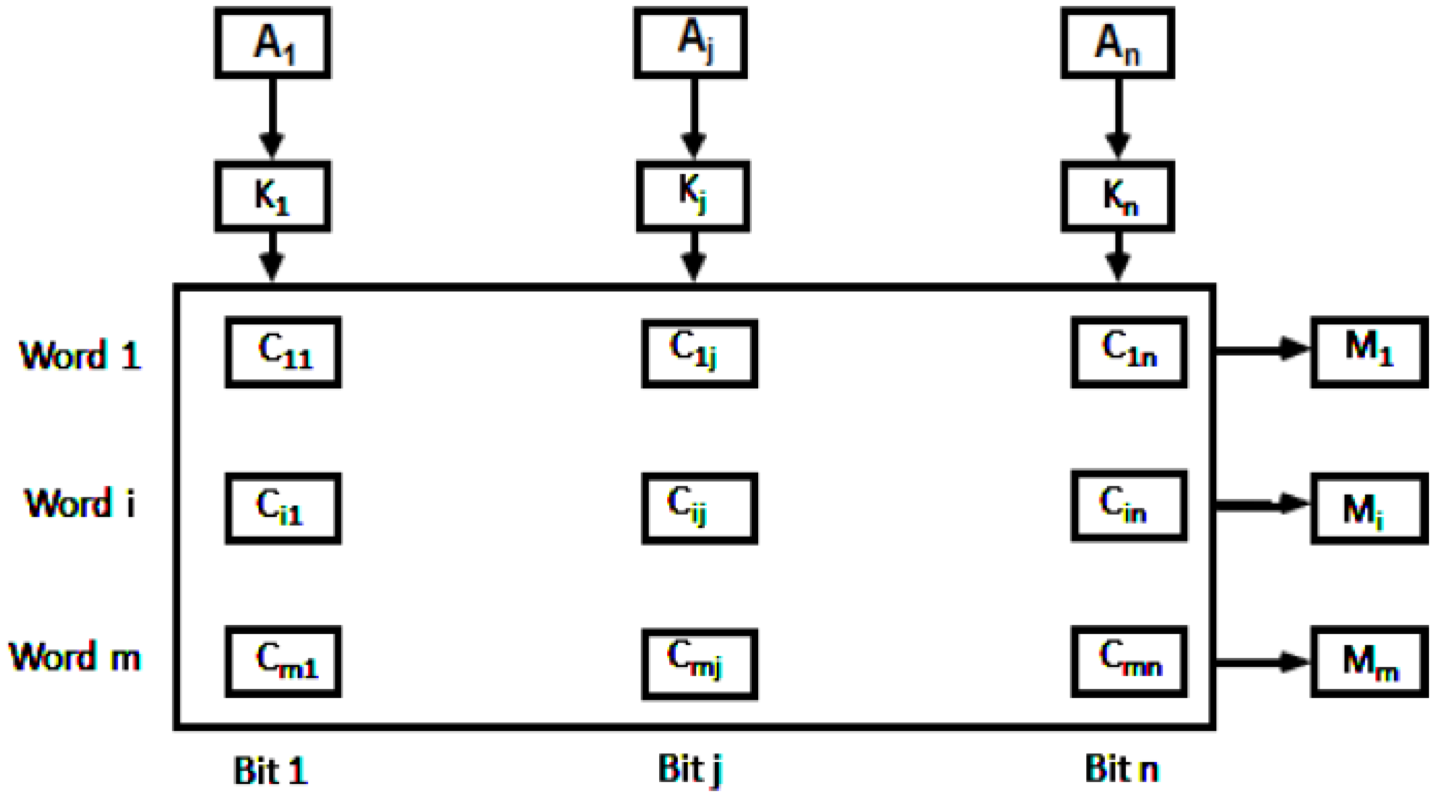


Figure 9.4: Associative memory of *m* word, n cells per word.

- The key register provides a mask for choosing a particular field or key in the argument word.
- The entire argument is compared with each memory word if the key register contains all 1's.
- Otherwise, only those bits in the argument that have 1st in their corresponding position of the key register are compared.
- Only the three leftmost bits of A are compared with memory words because K has 1's in these position.
- Each Cell contains matching logic. So checking done parallelly.
- Check parallel – So speed of associative memory is increasing.
- Very Fast (faster than SRAM)
- Very Expensive

APPLICATION : (1) For Cache implementation

(2) For TLB (Translation Lookaside Buffer)

Locality of Reference :

- If CPU has requested one address from memory, then **particular address** or **nearby address** will be accessed soon.
- Locality : Neighbourhood – Nearer to you – Nearer places
- Reference : Which Reference – Memory Reference
- Two types of **Locality of Reference :**

  1. Temporal - According to time (Same location access at diff. time)
  2. Spatial – Near location Access

Use : To improve the performance of the system – Use of cache memory

Cache Memory :

- Use of cache reduces average memory access time.
- If the active portions of the program and data are placed in a fast small memory, the average memory access time can be reduced, thus reducing the total execution time of the program. Such a fast small memory is referred to as a **cache memory**.

• Cache is a fast small capacity memory that should hold those information which are most likely to be accessed.

• The basic operation of the cache is, when the CPU needs to access memory, the cache is examined.

• If the word is found in the cache, it is read from the fast memory. If the word addressed by the CPU is not found in the cache, the main memory is accessed to read the word.

- The performance of cache memory is frequently measured in term of a quantity called ***hit ratio***.
- When the CPU refers to memory and finds the word in cache, it is said to produce a ***hit***.
- If the word not found in cache, it is in main memory and it count as a ***miss.***

- The ratio of the number of hits divided by the total references to memory (hits plus misses)is the ***hit ratio (H)***.

$$H= \frac{\text{No. of cache hits}}{\text{Total References}} \qquad H= \frac{\text{No. of cache hits}}{\text{Total hits + Total Miss}}$$

Miss ratio = (1-H)

- In case of cache miss, a block (which contains demanded byte) is copied from main memory to cache.
- The transformation of data from main memory to cache memory is referred to as a **mapping process**.
- 3 types of mapping procedures

    1. Associative mapping
    2. Direct mapping
    3. Set - Associative mapping

Associative mapping :

- Consider the main memory can store 32K words of 12 bits each.
- The cache is capable of storing 512 of these words at any given time.
- For every word stored in cache, there is a duplicate copy in main memory.
- The CPU communicates with both memories.
- It first sends a 15-bit address to cache. If there is a hit, the CPU accepts the 12-bit data from cache, if there is miss, the CPU reads the word from main memory and the word is then transferred to cache.
- The associative memory stores both the address and content (data) of the memory word.
- This permits any location in cache to store any word from main memory.

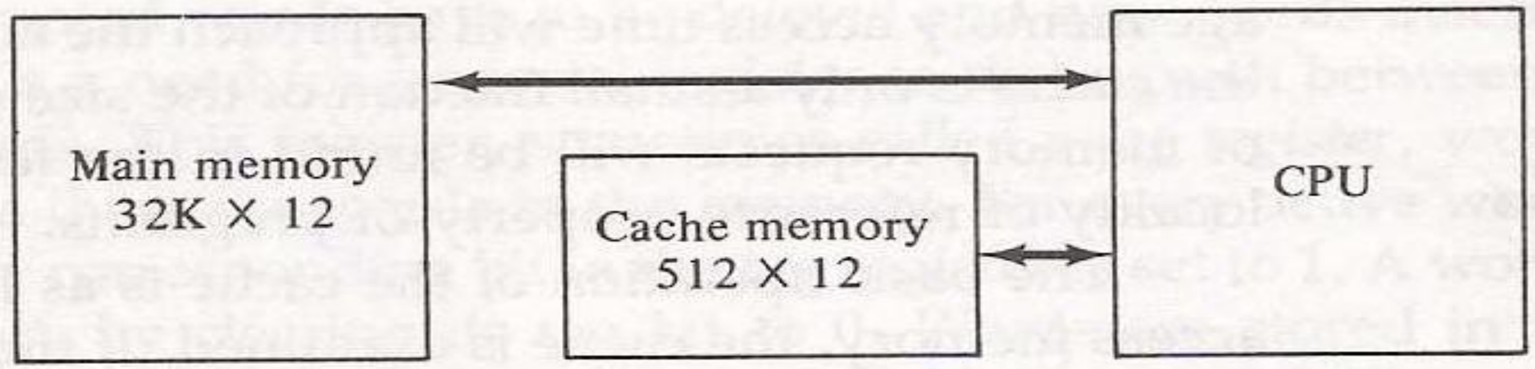
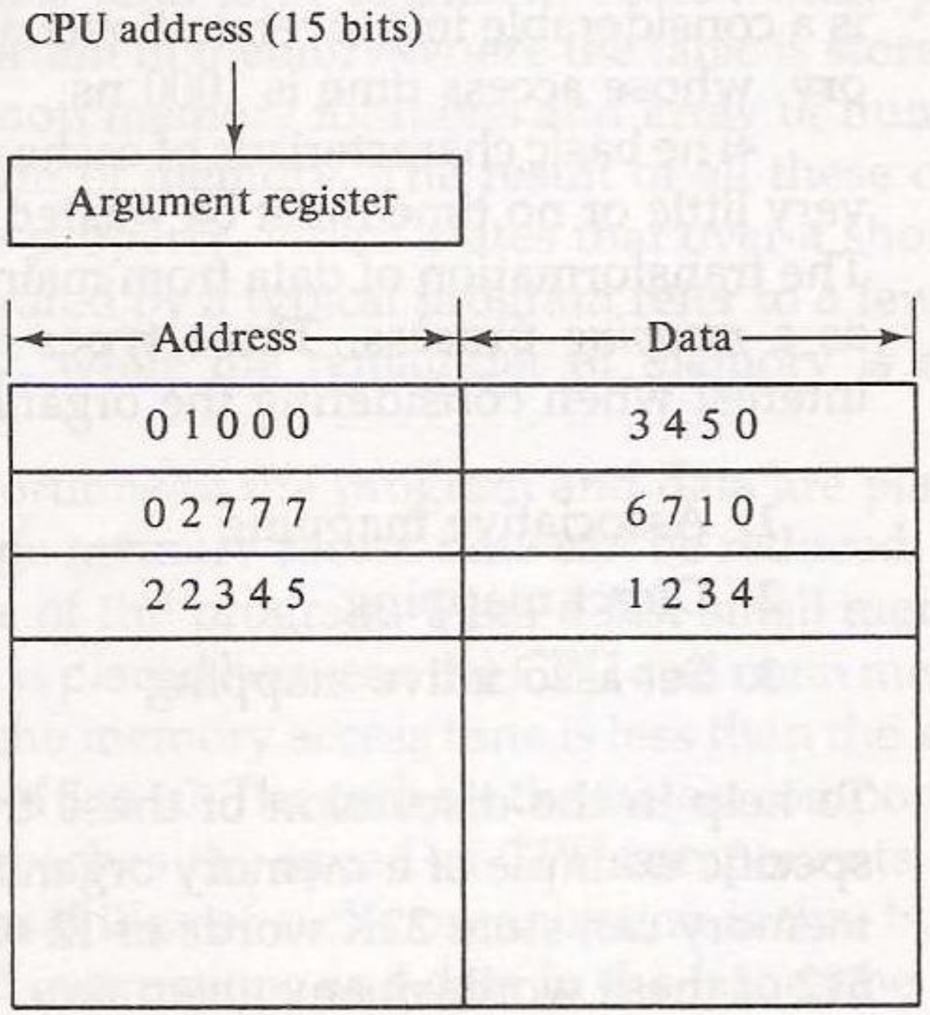


Figure 12-10 Example of cache memory.

- Shows in fig 12-11 three words presently stored in the cache.
- The address value of 15 bits is shown as a five-digit octal number and its corresponding 12-bit word is shown as a four-digit octal number.
- A CPU address of 15 bits is placed in the argument register and the associative memory is searched for a matching address.

Figure 12-11 Associative mapping cache (all numbers in octal).



- If the address is found the corresponding 12-bit data is read and sent to CPU.
- If no match occurs, the main memory is accessed for the word.
- The address data pairs then transferred to the associative cache memory.
- If the cache is full, an address data pair must be displaced to make room for a pair that is needed and not presently in the cache.
- This constitutes a first-in first-out (FIFO) replacement policy.

Associative Mapping Example :

MM Size	Cache Size	Block Size	Tag Size	Tag directory Size	Comparator
128 KB	16KB	256 B	**9 bits**	**(9 * 64) bits**	**64**
32 GB	32 KB	1 KB	**25 bits**	**(25 * 32) bits**	**32**
128 MB	512 KB	1 KB	17 BITS	**(17 * 512) bits**	**512**
16 GB		4 KB	**22 bits**	Can not get	
64 MB		64 KB	**10 bits**	Can not get	
	512 KB		7 bits		

MM = Physical Address

Block No	**Block Offset**

MM = 17 bits

Block No	**Block Offset**
9 bits = tag size	**8 bits**

Associative Mapping Example :

MM Size	Cache Size	Block Size	Tag Size	Tag directory Size	Comparator
128 KB	16KB	256 B	**9 bits**	**(9 * 64) bits**	**64**

MM

Block No	**Block Offset (B Size)**

MM = 17 bits

Block No	**Block Offset**
17-8 = 9 bits = tag size	**8 bits**

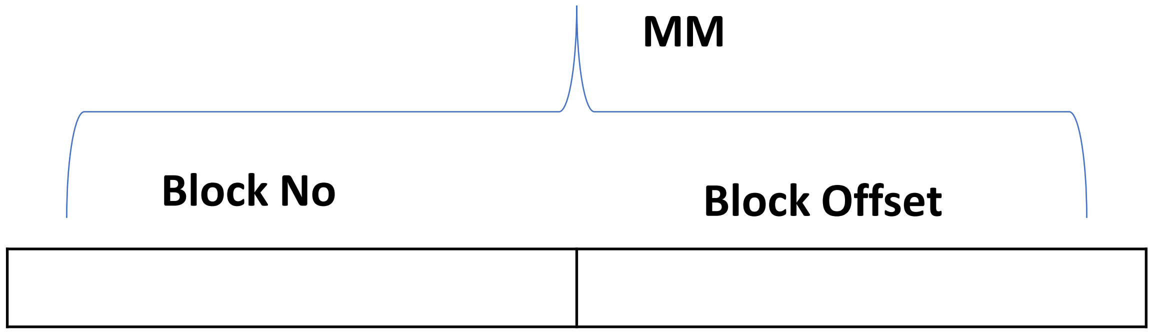
Cache size = 16 KB = 2^{14}

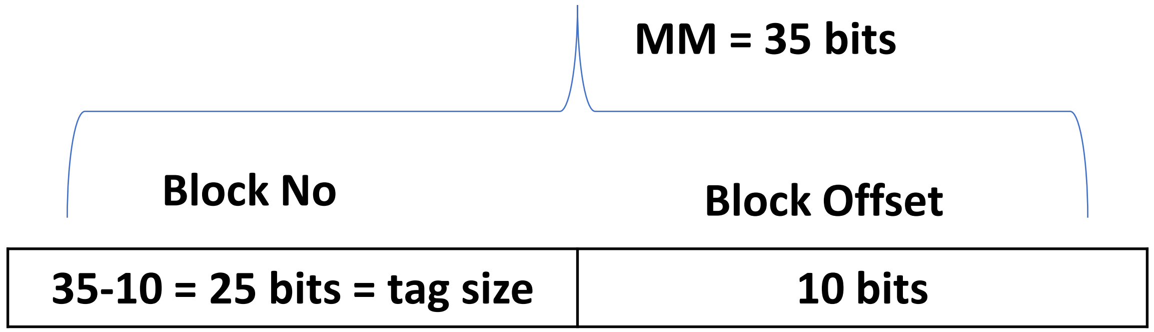
Block size = 256 B = 2^{8}

$$\frac{\textbf{Cache size}}{\textbf{Block size}} = \frac{2^{14}}{2^{8}} = 2^{6} = 64 \text{ line in cache}$$

Associative Mapping Example :

MM Size	Cache Size	Block Size	Tag Size	Tag directory Size	Comparator
32 GB	32 KB	1 KB	**25 bits**	**(25 * 32) bits**	**32**



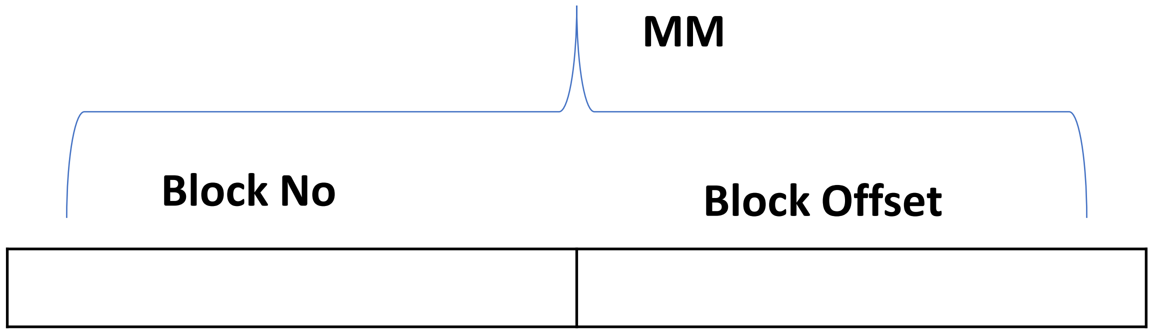


Cache size = 32 KB = 2^{15}
Block size = 1 KB = 2^{10}

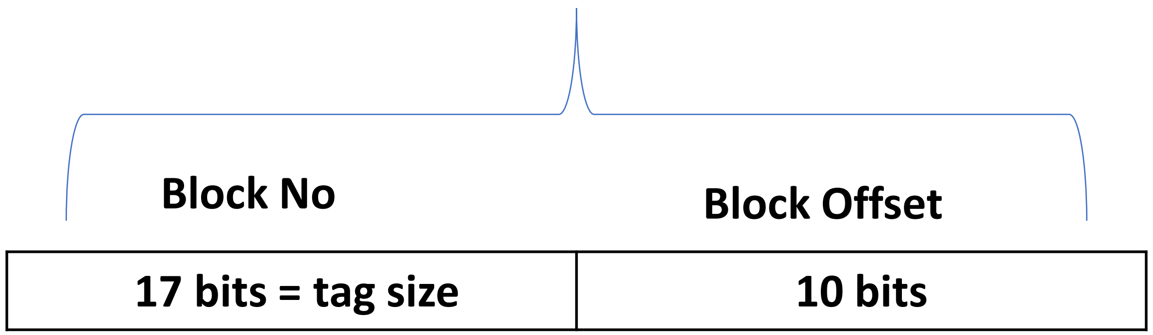
$$\frac{\textbf{Cache size} =}{\textbf{Block size} =} \frac{2^{15}}{2^{10}} = 2^{5} = 32 \text{ line in cache}$$

Associative Mapping Example :

MM Size	Cache Size	Block Size	Tag Size	Tag directory Size	Comparator
128 MB	512 KB	1 KB	17 Bits	(17 * 512) bits	512



MM = 17+10 = 27 bits = 2^{27} = 128 MB



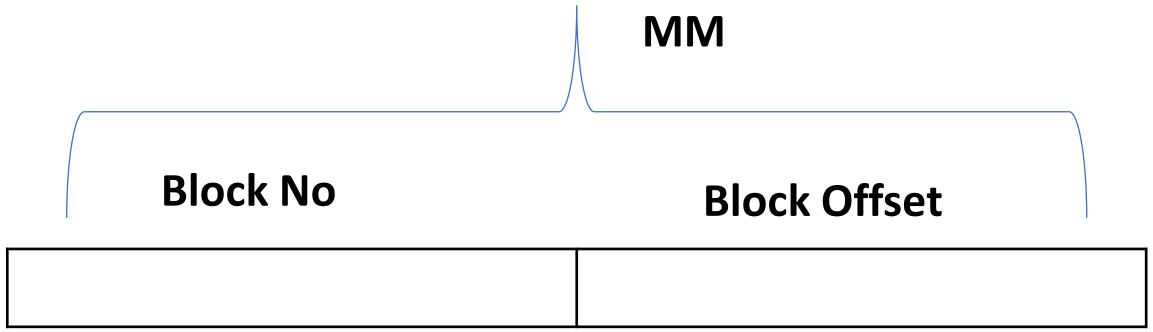
Cache size = 512 KB = 2^{19}

Block size = 1 KB = 2^{10}

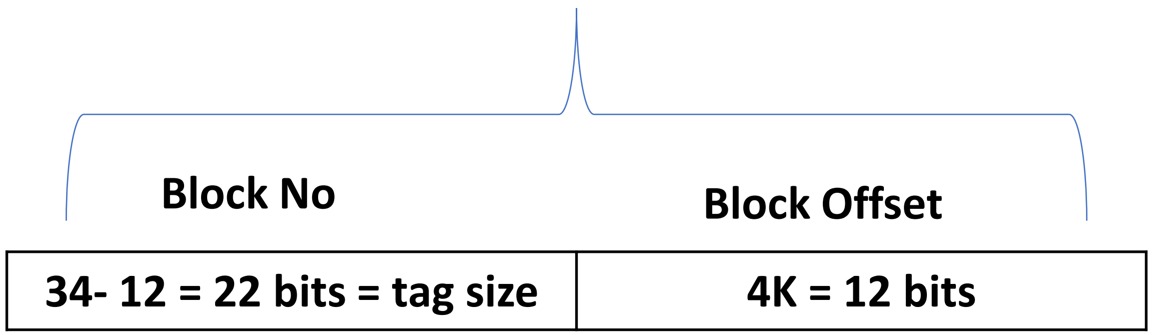
$$\frac{\textbf{Cache size}}{\textbf{Block size}} = \frac{2^{19}}{2^{10}} = 2^{9} = 512 \text{ line in cache}$$

Associative Mapping Example :

MM Size	Cache Size	Block Size	Tag Size	Tag directory Size	Comparator
16 GB	X	4 KB	**22 bits**	Can not get	X

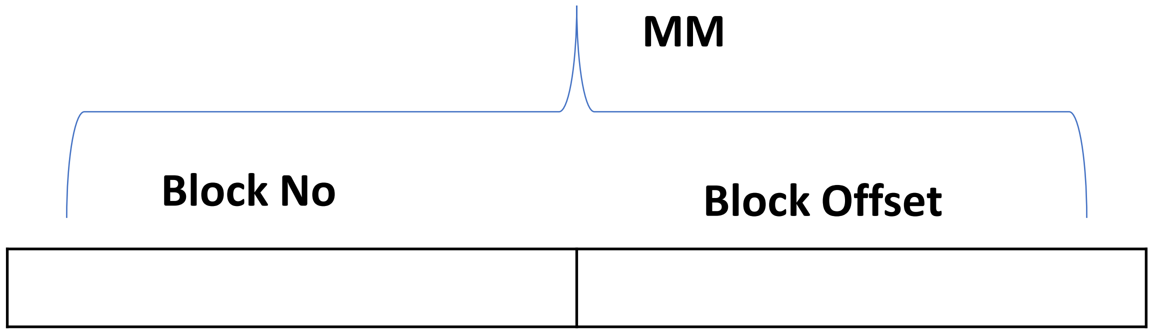


MM = 2^{34} = 34 bits = 16 GB

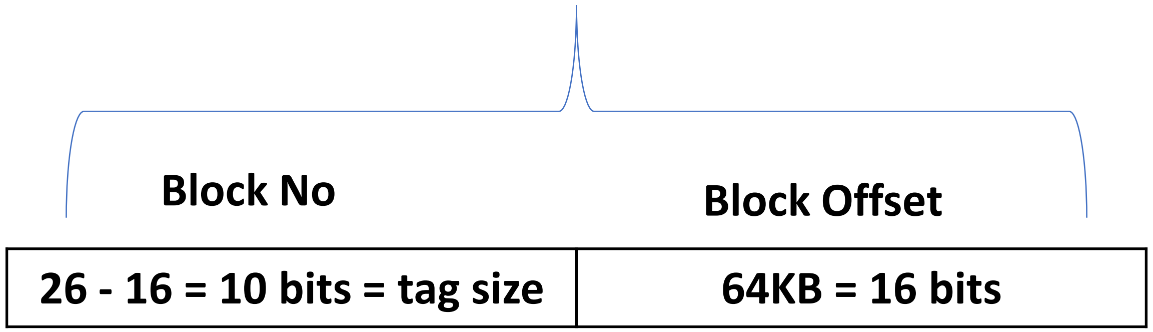


Associative Mapping Example :

MM Size	Cache Size	Block Size	Tag Size	Tag directory Size	Comparator
64 MB	x	64 KB	**10 bits**	Can not get	x

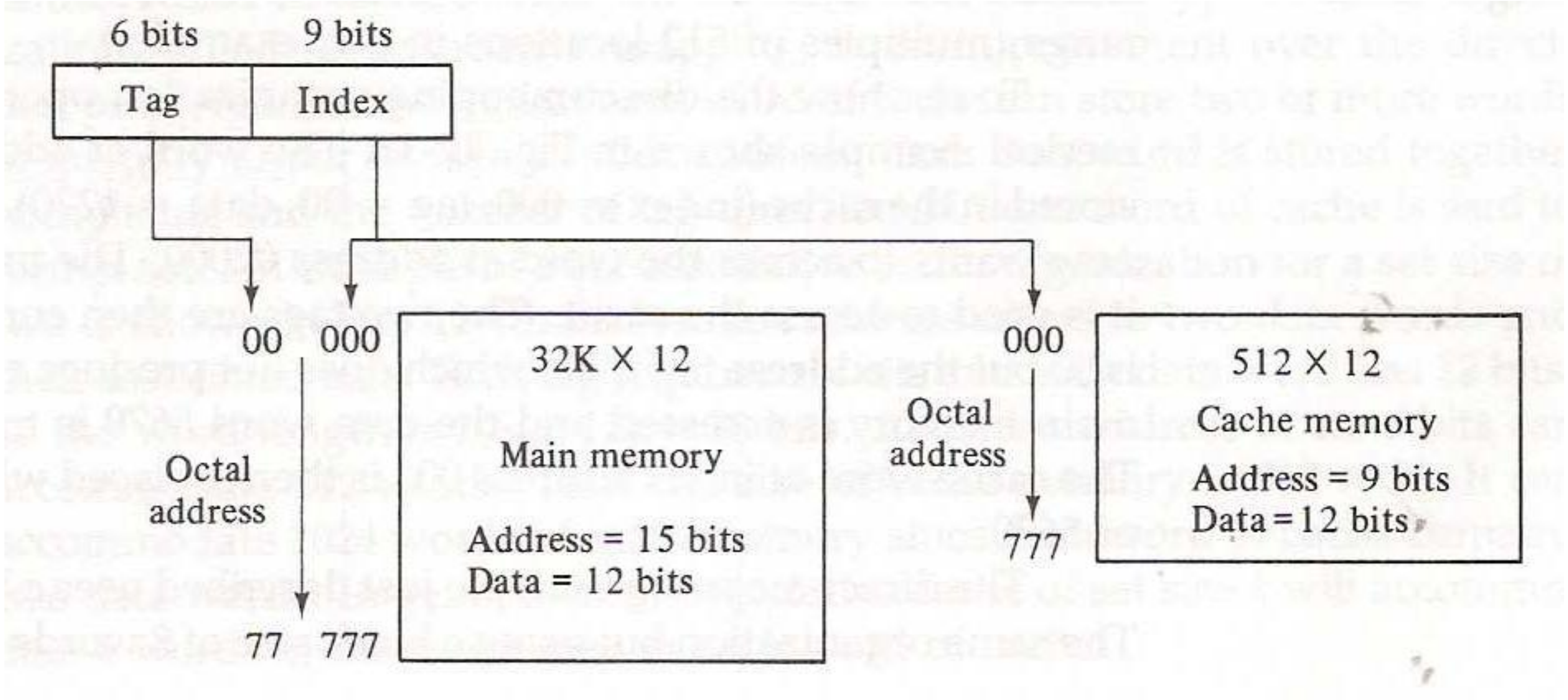


MM = 2^{26} = 26 bits = 64MB



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 from the tag field.
- The figure shows that main memory needs an address that includes both the tag and the index.



- The number of bits in the index field is equal to the number of address bits required to access the cache memory.
- The internal organization of the words in the cache memory is as shown in figure

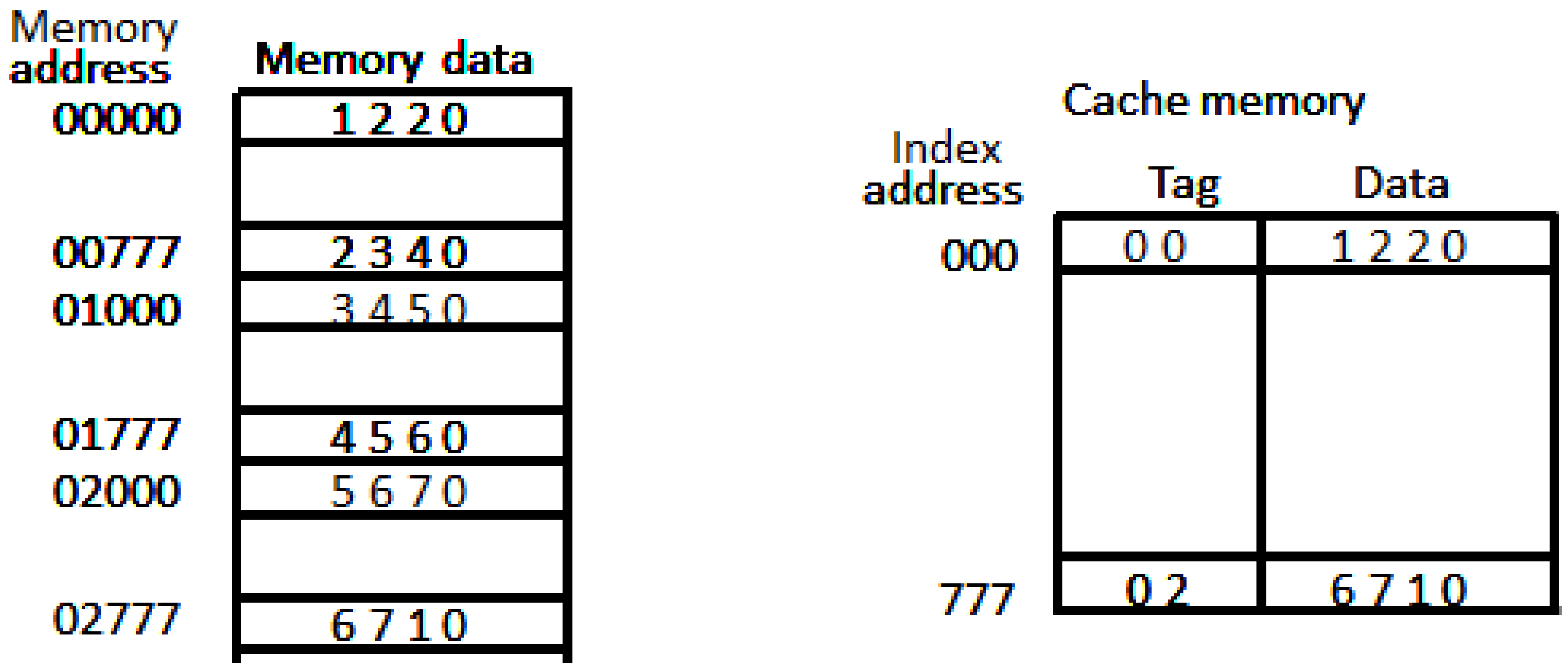


Figure Direct mapping cache organization

- Each word in cache consists of the data word and its associated tag.
- When a new word is first brought into the cache, the tag bits are stored alongside the data bits.
- When the CPU generates a memory request the index field is used for the address to access the cache.
- The tag field of the CPU address is compared with the tag in the word read from the cache.
- If the two tags match, there is a hit and the desired data word is in cache.
- If there is no match, there is a miss and the required word is read from main memory.
- It is then stored in the cache together with the new tag, replacing the previous value.
- The word at address zero is presently stored in the cache (index = 000, tag = 00, data =1220).
- Suppose that the CPU now wants to access the word at address 02000.

- The index address is 000, so it is used to access the cache. The two tags are then compared.
- The cache tag is 00 but the address tag is 02, which does not produce a match.
- Therefore, the main memory is accessed and the data word 5670 is transferred to the CPU.
- The cache word at index address 000 is then replaced with a tag of 02 and data of 5670.
- The **disadvantage** of direct mapping is that two words with the same index in their address but with different tag values cannot reside in cache memory at the same time.

- This direct-mapping example just described uses a block size of one word.
- The same organization but using a block size of 8 words shown in next figure.

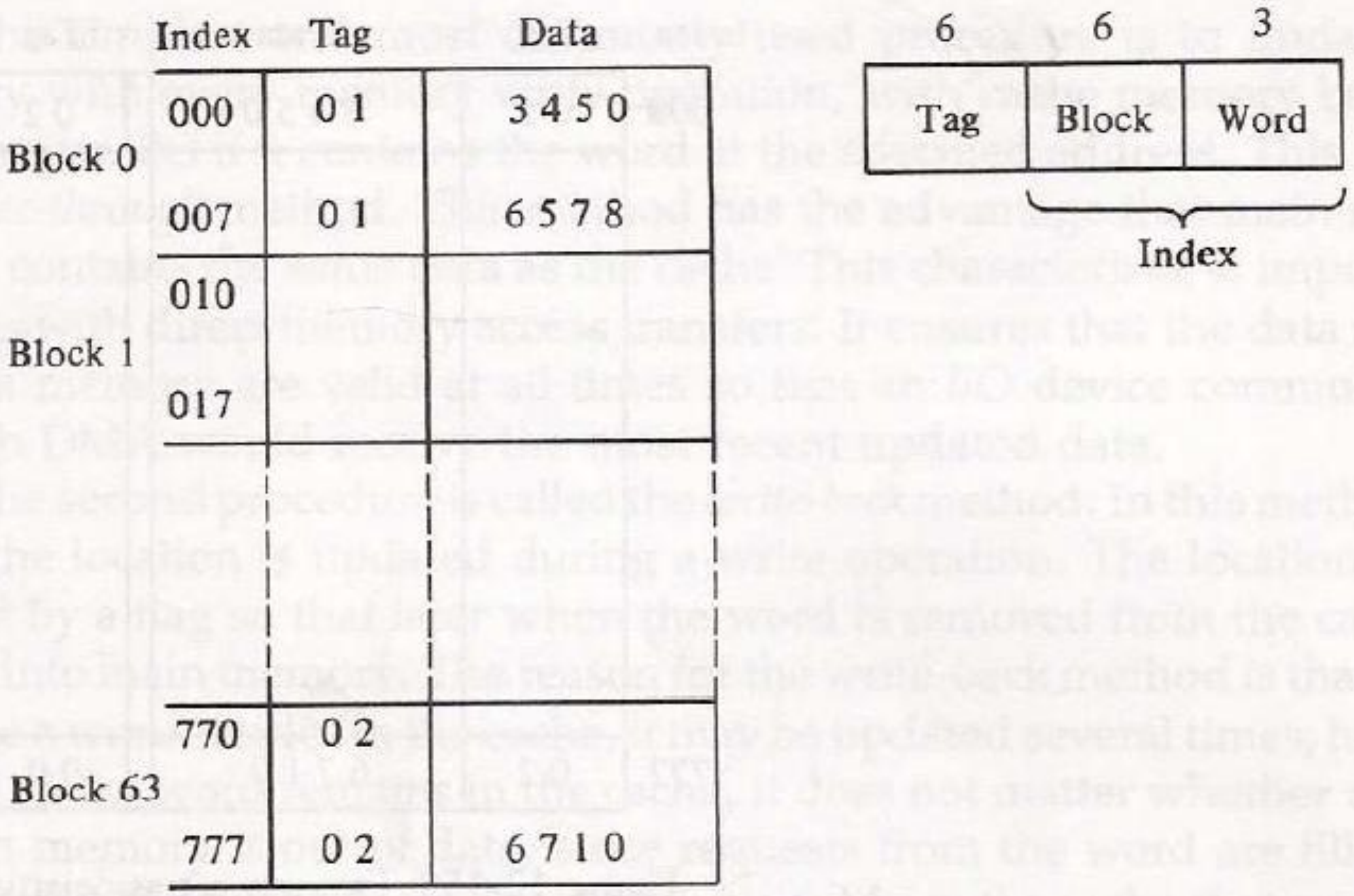


Figure 12-14 Direct mapping cache with block size of 8 words.

Cache size – 512

1 block size = 8 word

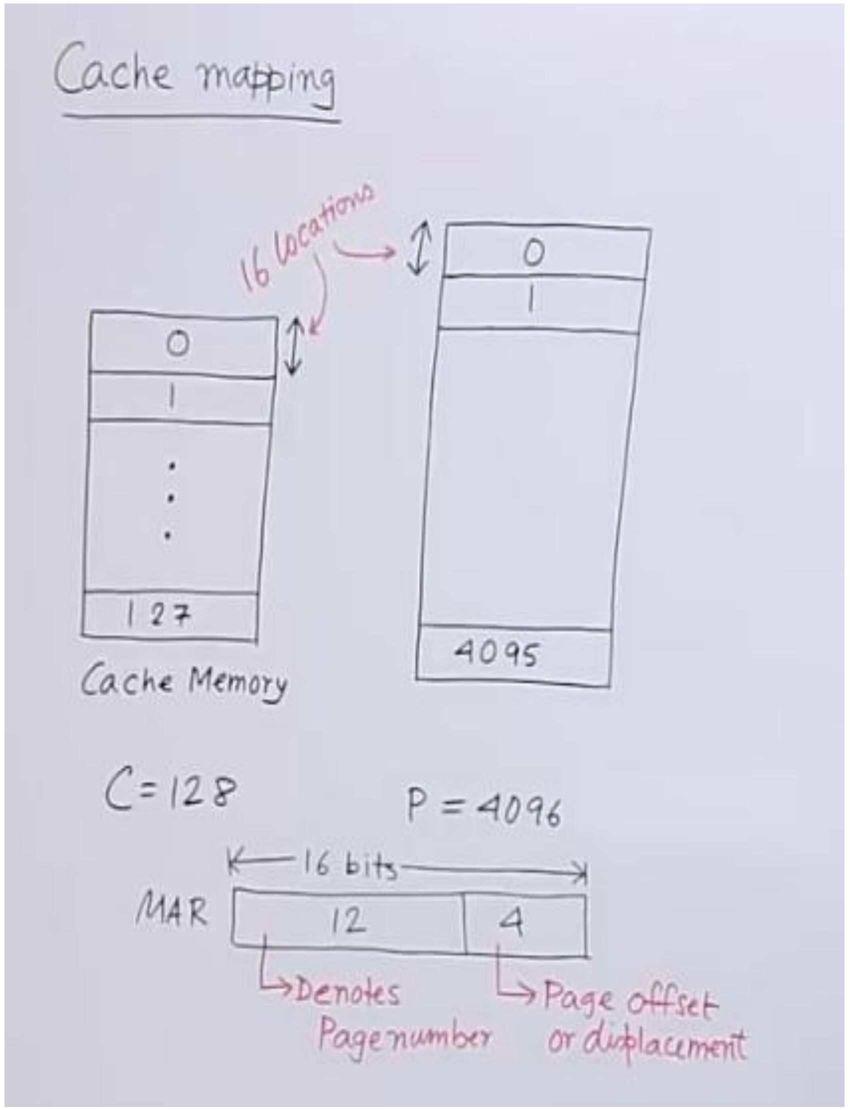
No. of block = 512 / 8
= 64

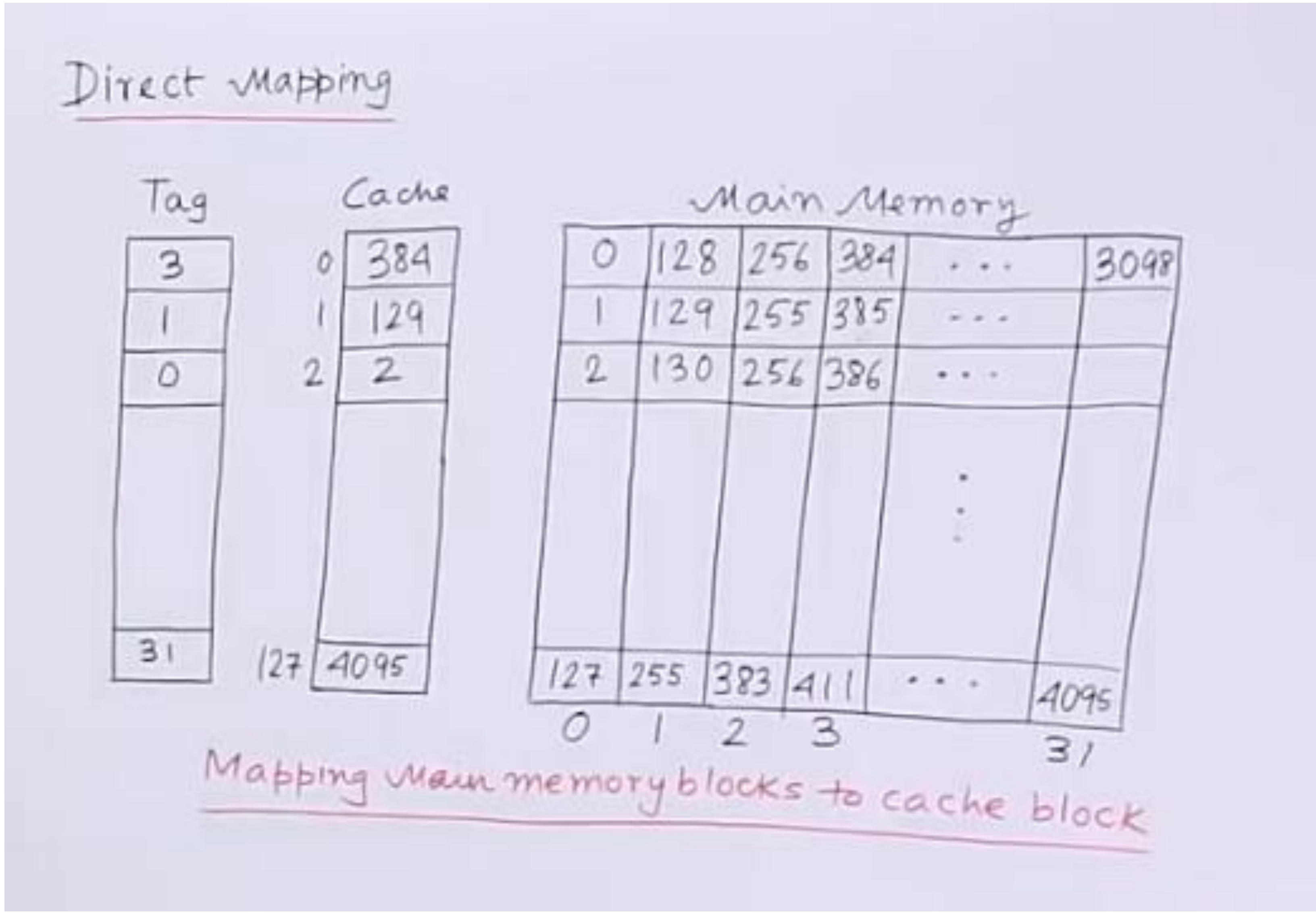
- **Index field** is divided into 2 parts **(1) Block Field** = 6-bit field
  (2) Word Field = 3-bit field

- The tag field stored within cache is common to all 8 words of the same block.

- Every time a miss occurs, an entire block of 8 words must be transferred from main memory to cache memory.

- Although this take extra time, the hit ratio will most likely improve with a larger block size because of the sequential nature of computer programs.





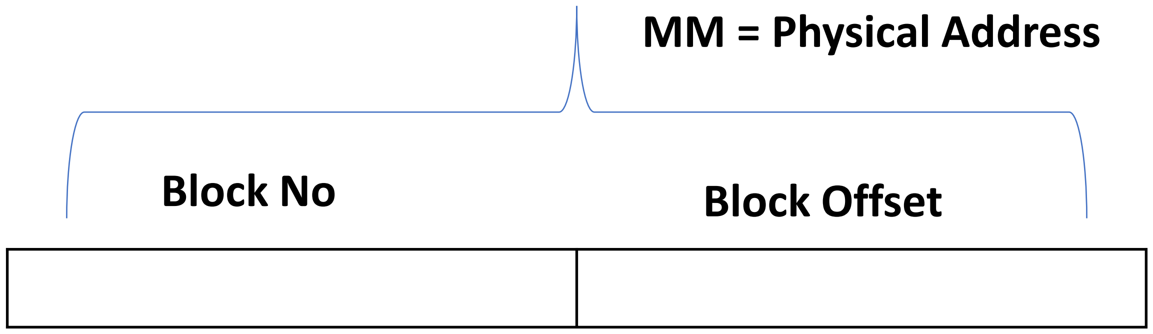
$$\frac{4096}{128} = \frac{2^{12}}{2^{7}} = 2^{5} = 32 \ (0 \text{ to } 31)$$

CM	**7**	**4**

Discuss direct cache mapping technique.
Complete missing parameter in below table using direct cache mapping technique, assume memory is byte addressable.

Direct cache Mapping Example :

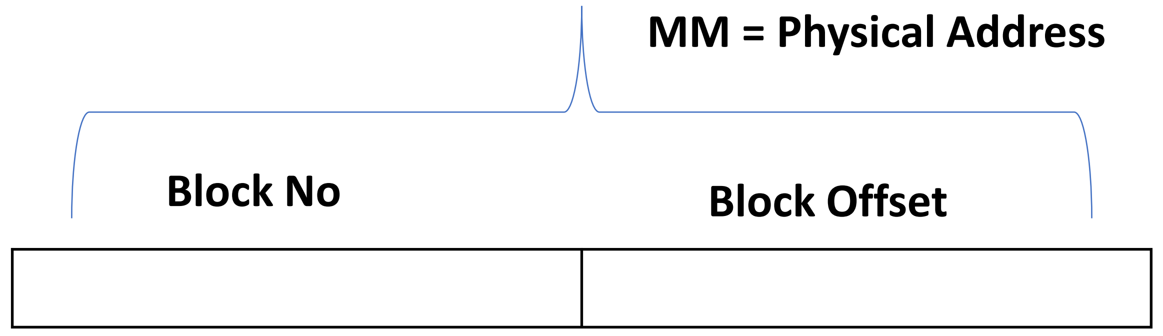
MM Size	Cache Size	Block Size	Tag Size	Tag directory Size	Comparator
128 KB	16KB	256 B	-----	-----	
32 GB	32 KB	1 KB	-----	-----	
-----	512 KB	1 KB	17 BITS	-----	
16 GB	-----	4 KB	-----	-----	
64 MB	-----	64 KB	-----	-----	
-----	512 KB	-----	7 bits	-----	



Discuss direct cache mapping technique.
Complete missing parameter in below table using direct cache mapping technique, assume memory is byte addressable.

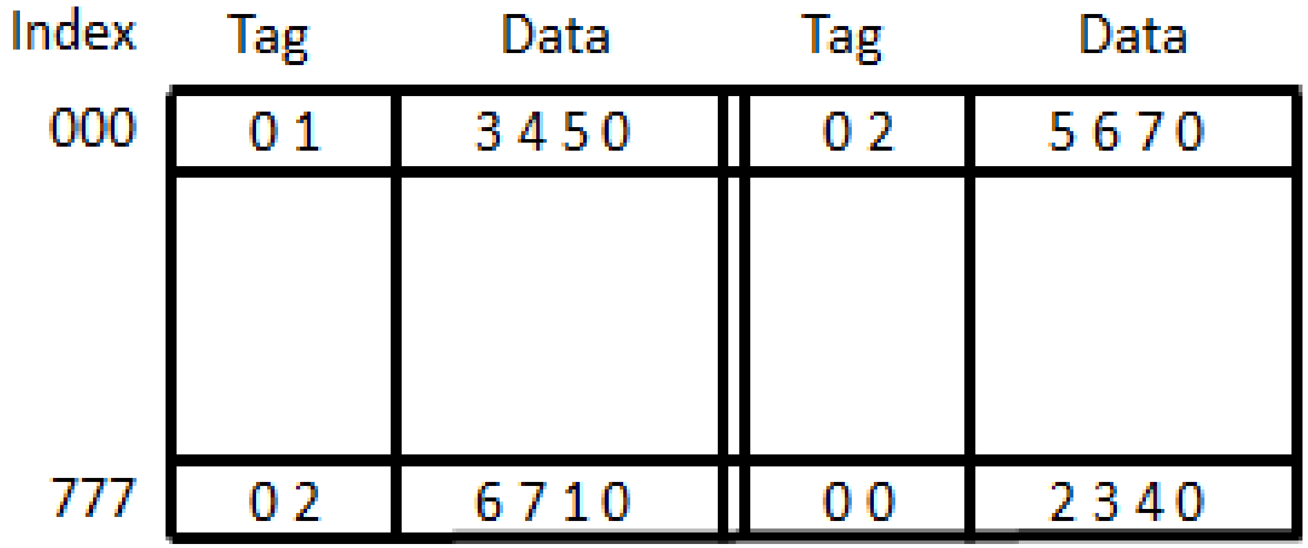
Direct cache Mapping Example :

MM Size	Cache Size	Block Size	Tag Size	Tag directory Size	Comparator
128 KB	16KB	256 B	**9 bits**	**(9 * 64) bits**	**No**
32 GB	32 KB	1 KB	**25 bits**	**(25 * 32) bits**	
128 MB	512 KB	1 KB	17 BITS	**(17 * 512) bits**	
16 GB		4 KB	**22 bits**	Can not get	
64 MB		64 KB	**10 bits**	Can not get	
	512 KB		7 bits		



Set - Associative mapping :

- A third type of cache organization, called set associative mapping 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 to form a set.
- An example one set-associative cache organization for a set size of two is shown in figure



Index	Tag	Data	Tag	Data
000	0 1	3 4 5 0	0 2	5 6 7 0
777	0 2	6 7 1 0	0 0	2 3 4 0

Figure : Two-way set-associative mapping cache

- Each index address refers to two data words and their associated terms.
- Each tag required 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 × 36. It can accommodate 1024 words or main memory since each word of cache contains two data words.
- In generation a set-associative cache of set size k will accommodate K word of main memory in each word of cache.
- The octal numbers listed in figure are with reference to the main memory contents.
- The words stored at addresses 01000 and 02000 of main memory are stored in cache memory at index address 000.
- Similarly, the words at addresses 02777 and 00777 are stored in cache at index address 777.

- When the CPU generates a memory request, the index value of the address is used to access the cache.

- The tag field of the CPU address is then compared with both tags in the cache to determine if a match occurs.

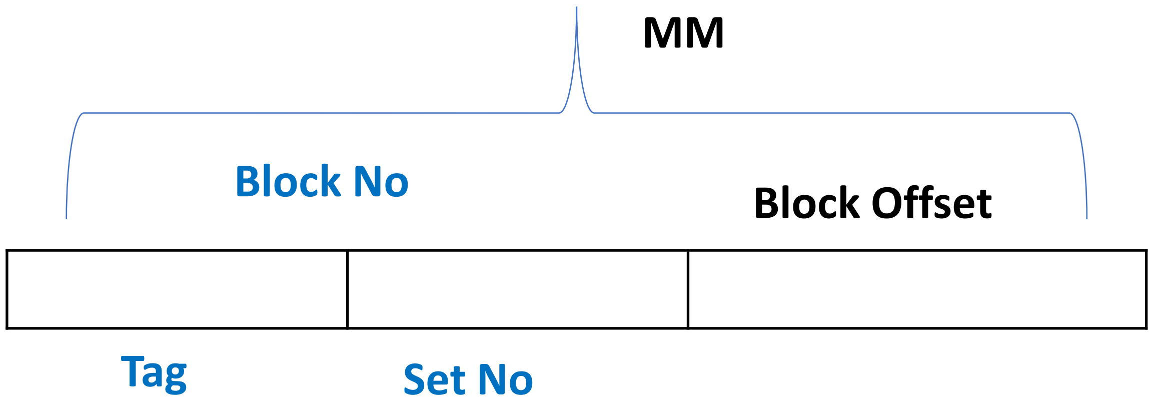
- The comparison logic is done by an associative search of the tags in the set similar to an associative memory search: thus the name "set-associative”.

- When a miss occurs in a set-associative cache and the set is full, it is necessary to replace one of the tag-data items with a new value.

- The most common replacement algorithms used are: random replacement, first-in first out (FIFO), and least recently used (LRU).

Set – Associative Mapping Example:

MM Size	Cache Size	Block Size	Tag Size	Tag directory Size	Set – Associative way	Comparator
128KB	16KB	256 KB	**4**	**(4* 64) bits**	2-way	**2**



4 –way set associative mem means no of line in each set is 4. (4 line in each set)

Set – Associative Mapping Example:

MM Size	Cache Size	Block Size	Tag Size	Tag directory Size	Set – Associative way	Comparator
128KB	16KB	256 B	**4**	**(4* 64) bits**	2-way	**2 * 4 bits**

MM = PA = 17-bits

Block No **Block Offset**

17-5-8 = **4**	**5**	256 B = **8** bits
Tag	**Set No**	

1 set contain 2-line

Set no

Sets = $2^6 / 2 = 2^5 = 32$

Lines= $\dfrac{\textbf{Cache size} = 2^{14}}{\textbf{Block size} = 2^{8}} = 2^6 = 64$

Set – Associative Mapping Example:

MM Size	Cache Size	Block Size	Tag Size	Tag directory Size	Set – Associative way	Comparator
32 GB	32KB	1 KB	**22**	**(22* 32) bits**	4-way	**4* 22 bits**

MM = PA = 35-bits

Block No		**Block Offset**
35-3-10 = **22**	**3**	1 KB = **10** bits
Tag	**Set No**	

1 set contain 4-line

Set no

Sets = 2^5 / 4 = $2^{\mathbf{3}}$ = 8

$$\text{Lines} = \frac{\textbf{Cache size} = 2^{15}}{\textbf{Block size} = 2^{10}} = 2^5 = 32$$

Set – Associative Mapping Example:

MM Size	Cache Size	Block Size	Tag Size	Tag directory Size	Set – Associative way	Comparator
8 MB	512 KB	1 KB	7	**(7* 512) bits**	8-way	**8* 7 bits**

MM = PA = 7+6+10 = 23-bits = 8 MB

Block No | **Block Offset**

7	**6**	1 KB = **10** bits
Tag	**Set No**	

1 set contain 8-line

Set no

Sets = 2^{9} / 8 = 2^{6} = 64

Lines= $\frac{\textbf{Cache size} = 2^{19}}{\textbf{Block size} = 2^{10}} = 2^{9} = 512$

Set – Associative Mapping Example:

MM Size	Cache Size	Block Size	Tag Size	Tag directory Size	Set – Associative way	Comparator
16 GB	**64 MB**	4 KB	10	**(10* 4 KB) bits**	4-way	**4* 10 bits**

MM = PA = 34 -bits

Block No **Block Offset**

10	34-10-12=**12**	4 KB = **12** bits

Tag **Set No**

Cache size = No of set * Line per set * Size of line

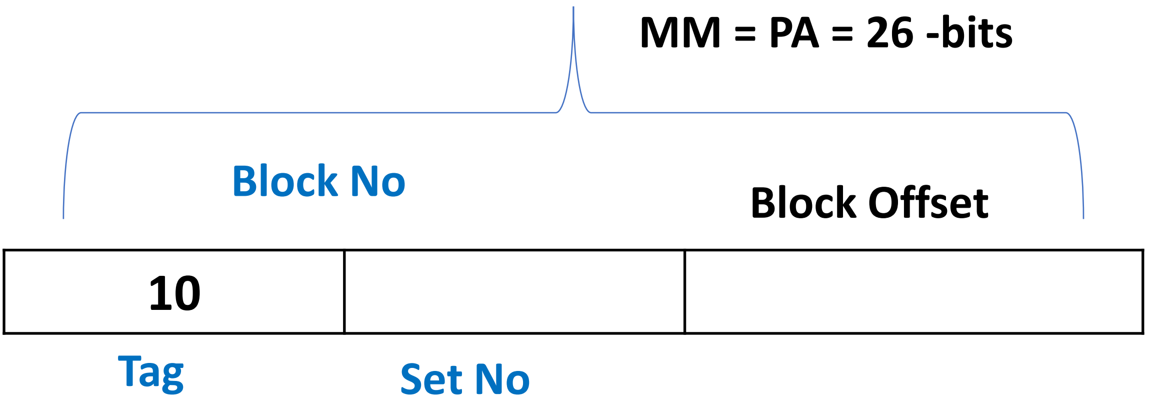
$= 2^{12} \ * \ 4 \ * \ 2^{12}$ (4KB)

$= 2^{26} = 2^{6}$ MB = **64 MB**

Set – Associative Mapping Example:

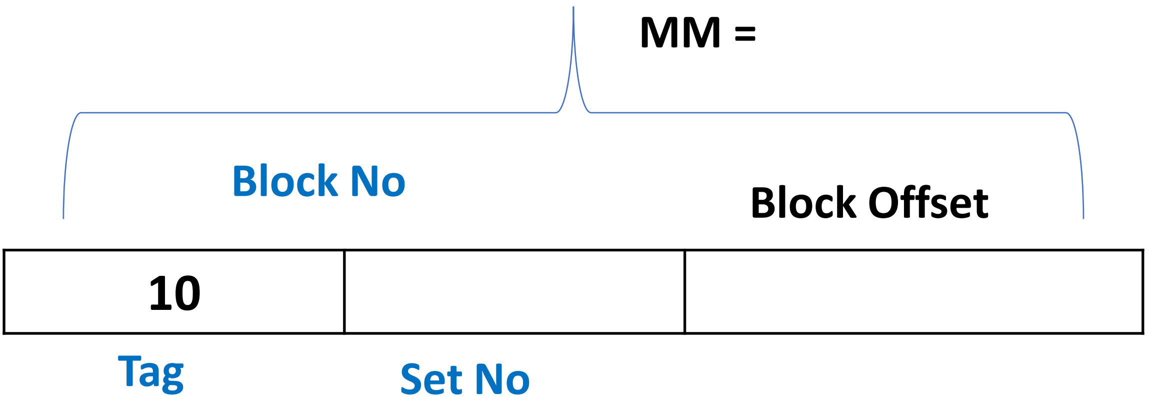
MM Size	Cache Size	Block Size	Tag Size	Tag directory Size	Set – Associative way	Comparator
64 MB	**x**	x	10	**x**	4-way	**4* 10 bits**

MM = PA = 26 -bits



Set – Associative Mapping Example:

MM Size	Cache Size	Block Size	Tag Size	Tag directory Size	Set – Associative way	Comparator
x	512 KB	x	7	**x**	8 –way	**8 * 7 bits**



Writing into Cache :

- 2 Types → **(1) Write Through** (2) Write-Back (Copy-Back)

- The simplest and most commonly used procedure is to update main memory with every memory write operation.
- The cache memory being updated in parallel if it contains the word at the specified address. This is called the *write-through* method.
- This method has the advantage that main memory always contains the same data as the cache.
- This characteristic is important in systems with direct memory access(**DMA**) transfers.
- It ensures that the data residing in main memory are valid at all times so that an I/O device communicating through DMA would receive the most recent updated data.

Writing into Cache :

- 2 Types → (1) Write Through **(2) Write-Back (Copy-Back)**

- 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 for the write-back method is that during the time a word resides in the cache, it may be updated several times.
- However, as long as the word remains in the cache, it does not matter whether the copy in main memory is out of date, since requests from the word are filled from the cache.
- It is only when the word is displaced from the cache that an accurate copy need be rewritten into main memory.

Cache Initialization : (Problem of Initialization)

- Cache is initialized when power on or main memory is loaded from Auxiliary memory.
- After initialized the cache is considered to be empty, but it contains some **non-valid data**.
- To include with each word in cache a **valid bit** to indicate whether or not the contains valid data.
- The cache is initialized by **clearing all the valid bits to 0**.
- The **cache word is set to 1** the first time **word is loaded from main memory** & stays set unless the cache has to initialized again.
- The **valid bit set 1 means valid data**.
- The **valid bit set 0 means invalid data** – so replaces word.

Virtual Memory :

- Virtual memory is a concept used in some large computer systems that permit the user to construct programs as though a large memory space were available, equal to the totality of auxiliary memory.

- Virtual memory is used to give programmers the illusion that they have a very large memory at their disposal, even though the computer actually has a relatively small main memory.

- A virtual memory system provides a mechanism for translating program-generated addresses into correct main memory locations.

- ***Address space***

An address used by a programmer will be called a **virtual address**, and the set of such addresses is known as **address space**.

- ***Memory space***

An address in main memory is called a **location or physical address**. The set of such locations is called the **memory space**.

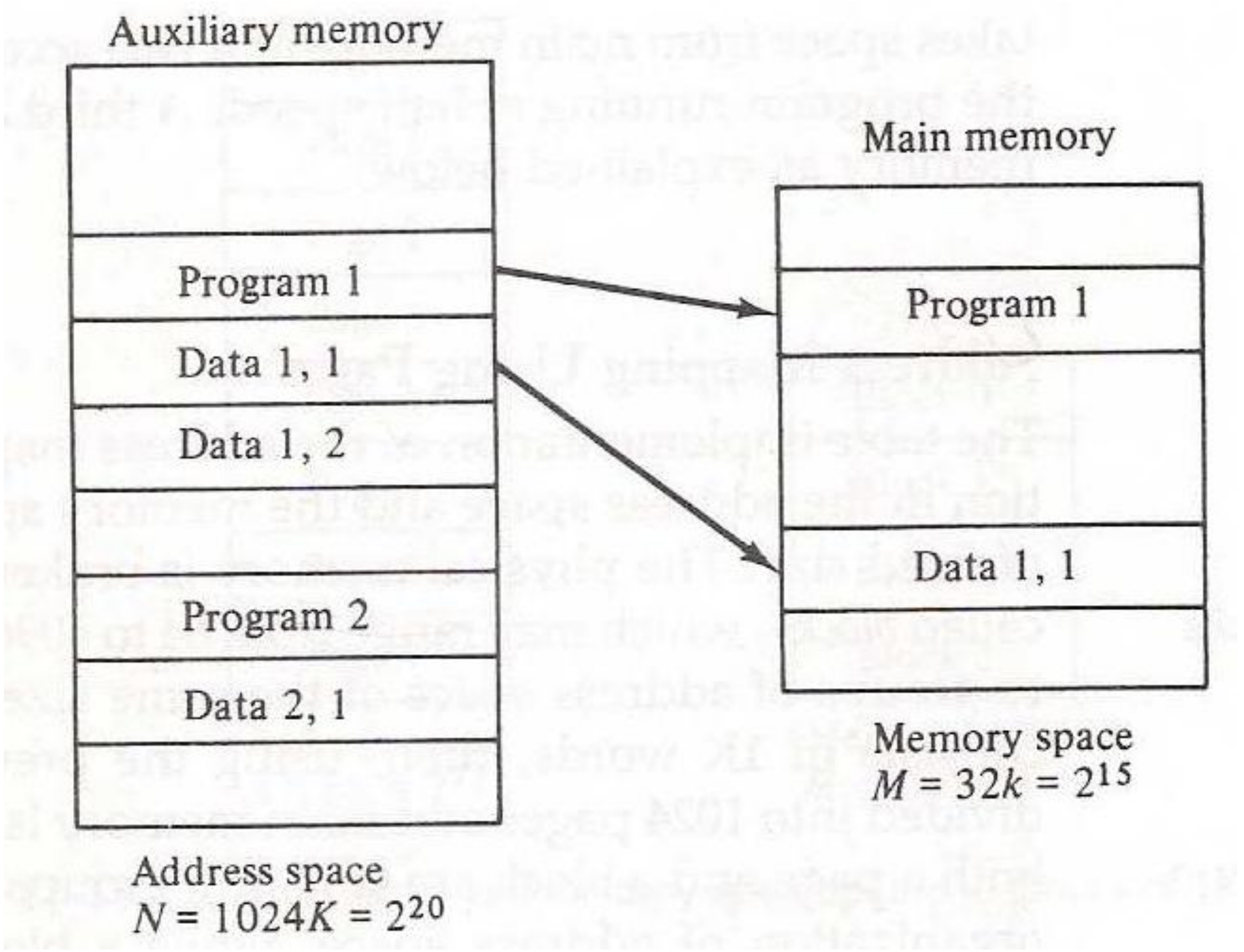


Figure : Relation between address and memory space in a virtual memory system

- As an illustration, consider a computer with a main-memory capacity of 32K words (K =1024). Fifteen bits are needed to specify a physical address in memory since **32K = 2^{15}.**
- Suppose that the computer has available auxiliary memory for storing **2^{20} = 1024K** words.
- Thus auxiliary memory has a capacity for storing information equivalent to the capacity of 32 main memories.
- Denoting the address space by N and the memory space by M, we then have for this example **N = 1024K** and **M = 32K**.
- In a multiprogramming computer system, programs and data are transferred to and from auxiliary memory and main memory based on demands imposed by the CPU.
- Suppose that program 1 is currently being executed in the CPU. Program 1 and a portion of its associated data are moved from auxiliary memory into main memory as shown in figure.

- Portions of programs and data need not be in contiguous locations in memory since information is being moved in and out, and empty spaces may be available in scattered locations in memory.
- In our example, the address field of an instruction code will consist of 20 bits but physical memory addresses must be specified with only 15 bits.
- Thus CPU will reference instructions and data with a 20-bit address, but the information at this address must be taken from physical memory because access to auxiliary storage for individual words will be too long.

Virtual Memory Mapping Technique :

- Mapping of address from virtual memory (auxiliary memory) to physical memory (main memory) is called ***virtual memory mapping***.

- 3 Techniques :
  1. Mapping with pages
  2. Mapping with segmentation
  3. Mapping with segmented paging

Address Mapping with pages :

- An address used by programmer will be called a ***virtual address*** and the set of such addresses the ***address space***.
- An address in main memory is called as a ***location or physical address***. The set of such locations is called the ***memory space***.
- The table implementation of the address mapping is simplified if the information in the **address space and the memory space** are each divided into groups of **fixed size**.

- **Virtual & Physical address** space will be divided into **equal size blocks**.

- In **Virtual address space** these blocks are called **Pages** &

  In **Physical address** space these blocks are called **Frames**.

- Size (Page) = Size (Frame)

- **Virtual address** divided into 2 parts :

  Page : (1) Page No (P)

  (2) Offset (d)

- **Physical address** divided into 2 parts :

  Frame : (1) Frame No (F)

  (2) Offset (d)

  - A mapping is needed from pages to frames.
  - Consider a computer with an **address space of 8K** and a **memory space of 4K**.

- If we split each into groups of 1K words we obtain **eight pages** and **four blocks** as shown in figure

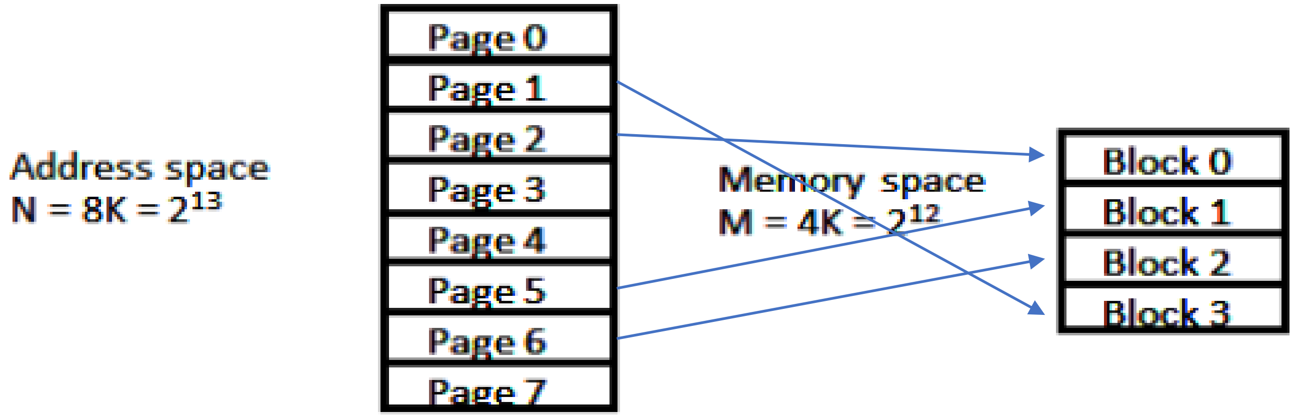


Figure 9.10 Address and Memory space split into group of 1K words

- size(Page) = size (Frame) = 1K
- **8 Pages & 4 Frames**

- Virtual address :

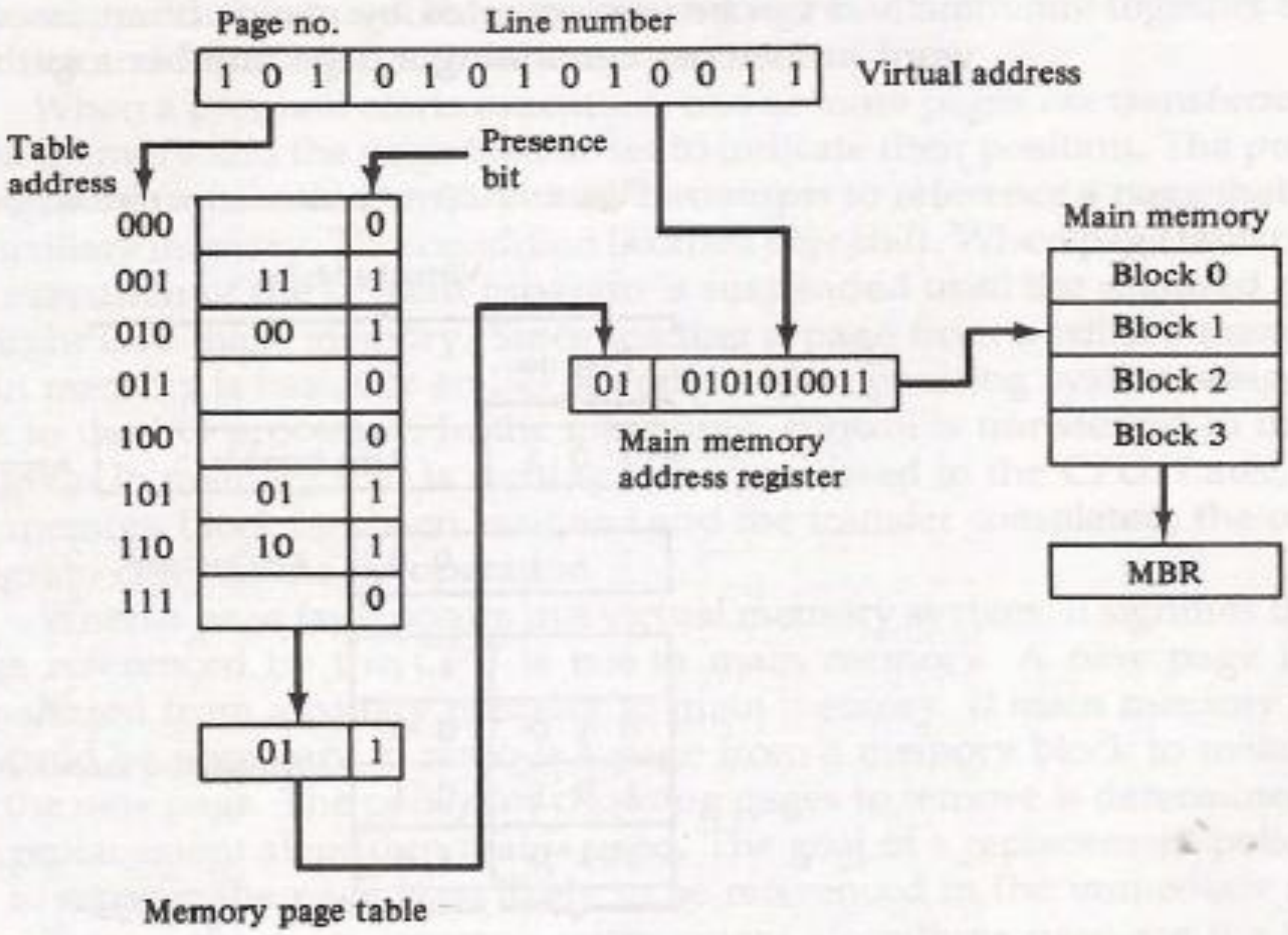
Page (3)	**Offset (10)**

- Physical address :

Frame (2)	**Offset (10)**

- Need a table which keeps track of which page of virtual memory is loaded in which frame of physical memory. This table is called ***Page table.***
- The organization of the memory mapping table in a paged system is shown in figure
- CPU generate **13-bits virtual address** & divided in two parts – **page no & Line no**.

Figure 12-19 Memory table in a paged system.



- The memory-page table consists of eight words, one for each page.
- The address in the page table denotes the page number and the content of the word give the block number where that page is stored in main memory.
- The table shows that pages 1, 2, 5, and 6 are now available in main memory in blocks 3, 0, 1, and 2, respectively.

- A **presence bit** in each location indicates whether the page has been transferred from auxiliary memory into main memory.
- A **0 in the presence bit** indicates that this page is not available in main memory.

- The CPU references a word in memory with a virtual address of 13 bits.
- The three high-order bits of the virtual address specify a page number and also an address for the memory-page table.

- The content of the word in the memory page table at the page number address is read out into the memory table buffer register.
- If the presence bit is a 1, the block number thus read is transferred to the two high-order bits of the main memory address register.

- The line number from the virtual address is transferred into the 10 low-order bits of the memory address register.
- A read signal to main memory transfers the content of the word to the main memory buffer register ready to be used by the CPU.

- If the presence bit in the word read from the page table is 0, it signifies that the content of the word referenced by the virtual address does not reside in main memory.

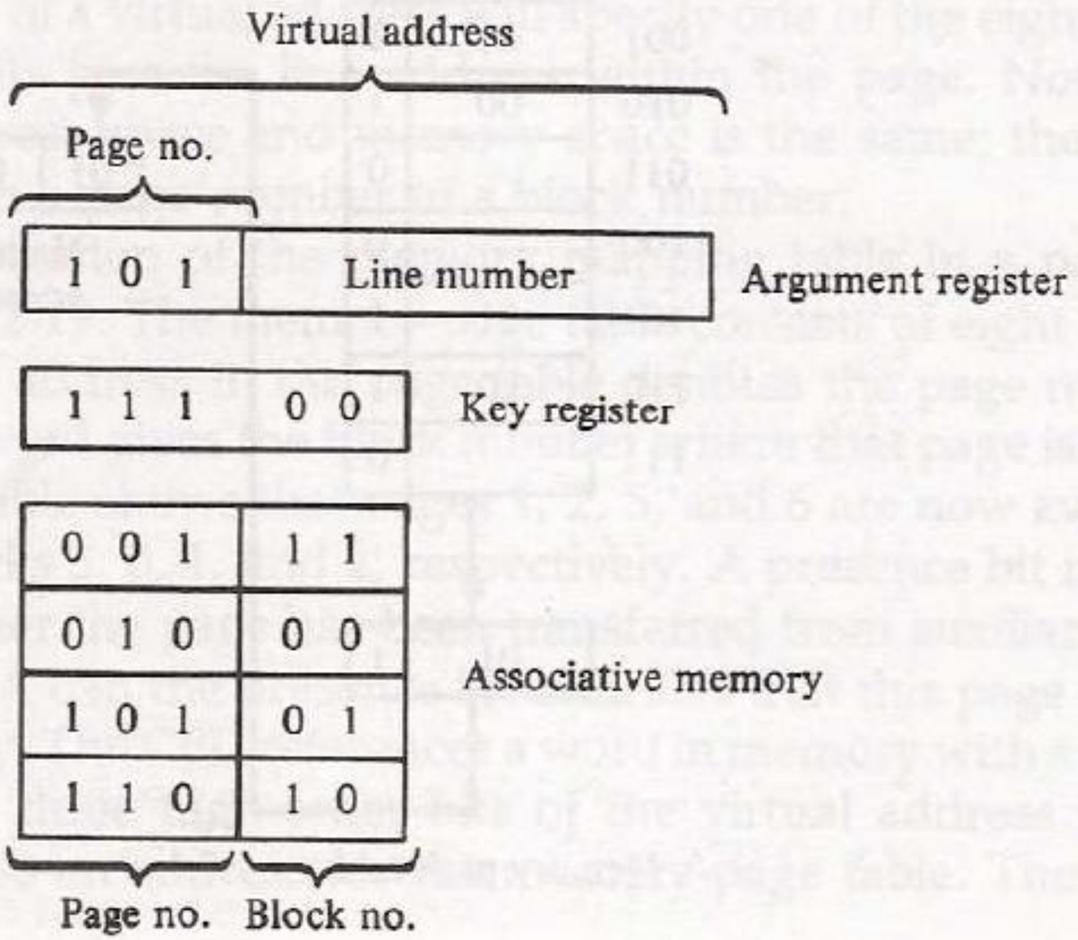
Associative Memory Page Table :

- Random access memory page table is inefficient with respect to storage utilization.
- In previous example **8-word of memory** needed, one for each page. But at least **4 words will always be marked empty** because main memory cann’t accommodate **more than 4 blocks**.

- Add space – 1024K & memory space- 32K words
- Pages=1024 & Block = 32
- At least 1024-32 = 992 locations will be empty & not in use.
- More efficient way to organize to page table would be to construct it with number of words equal to no of blocks in main memory.

- In this way, size of memory is reduced and each location is fully utilized.

Figure 12-20 An associative memory page table.



- We replace random access memory page table with an associative memory of 4 words.
- Each entry have 2 field : (1) 3 bits –Page No

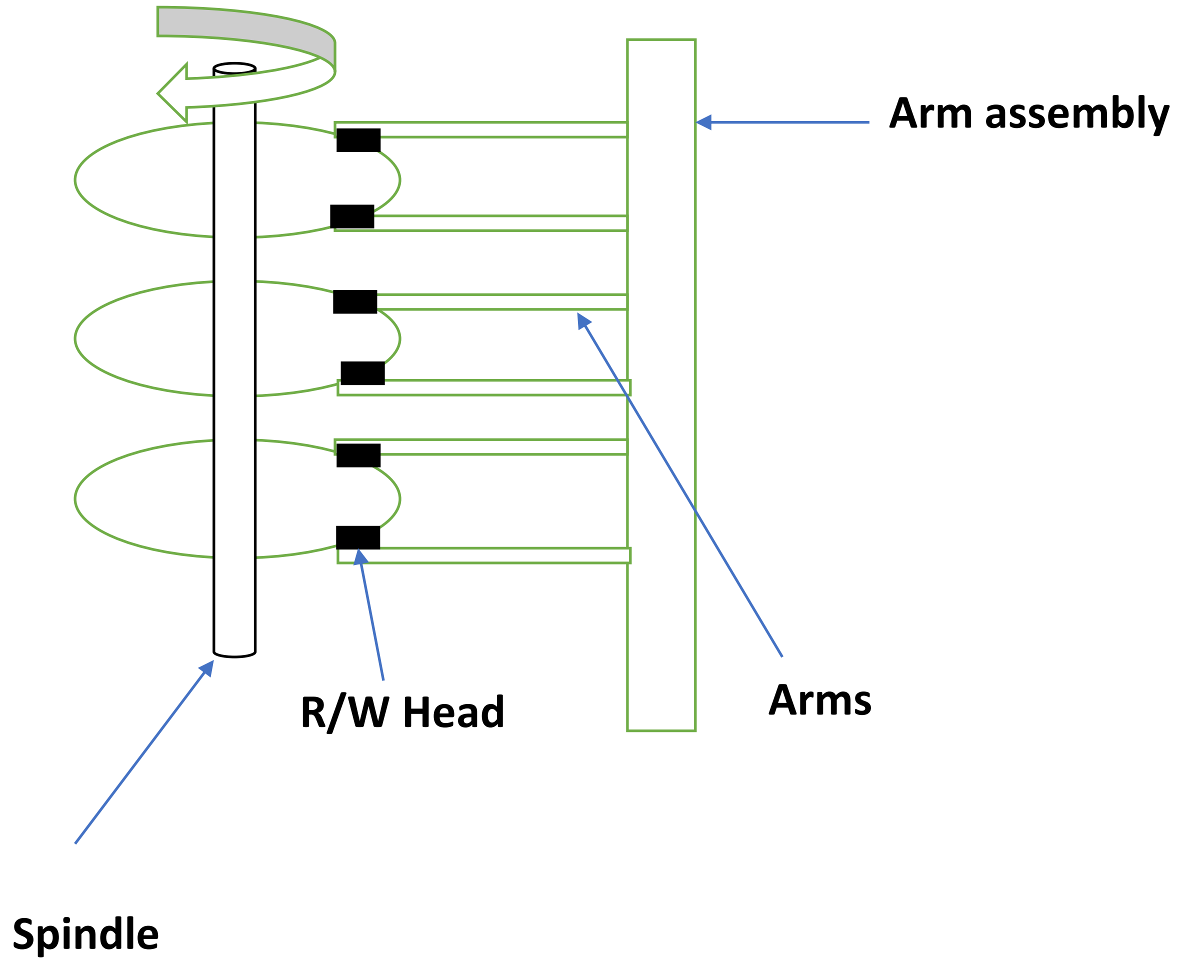
  (2) 2 bits – Block no
- Virtual address placed in Argument register.
- Page no bits in argument reg. are compared with all the page no in page field of associative memory.
- If page number is found, the 5-bit word is read out from memory.
- If no match occur, a call to the OS is generated to bring the required page from auxiliary memory.
- Page Replacement Algorithm : FIFO & LRU (Least recently used)

Auxiliary Memory :

- Most common auxiliary memory devices – Magnetic Disks & Tapes.
- The average time required to reach a storage location in memory and obtain its content is called the ***access time***.

- Compact Disk – only one surface for storage
- Magnetic Disk – Both surface store data

- Number of disk are there in Magnetic Disks and it is called as ***platter***.
- 3 platter – 6 side storage available
- In magnetic surface store data is called recording.



Read or Write on the surface then required one small pointer is called **Read/Write head**.

Each Surface have particular R/W Head.

Attach R/W head to one hardware is called as **arms**.

The arms are attached with column wise structure is called **Arms assembly**.

Disk rotation happens in only single direction.

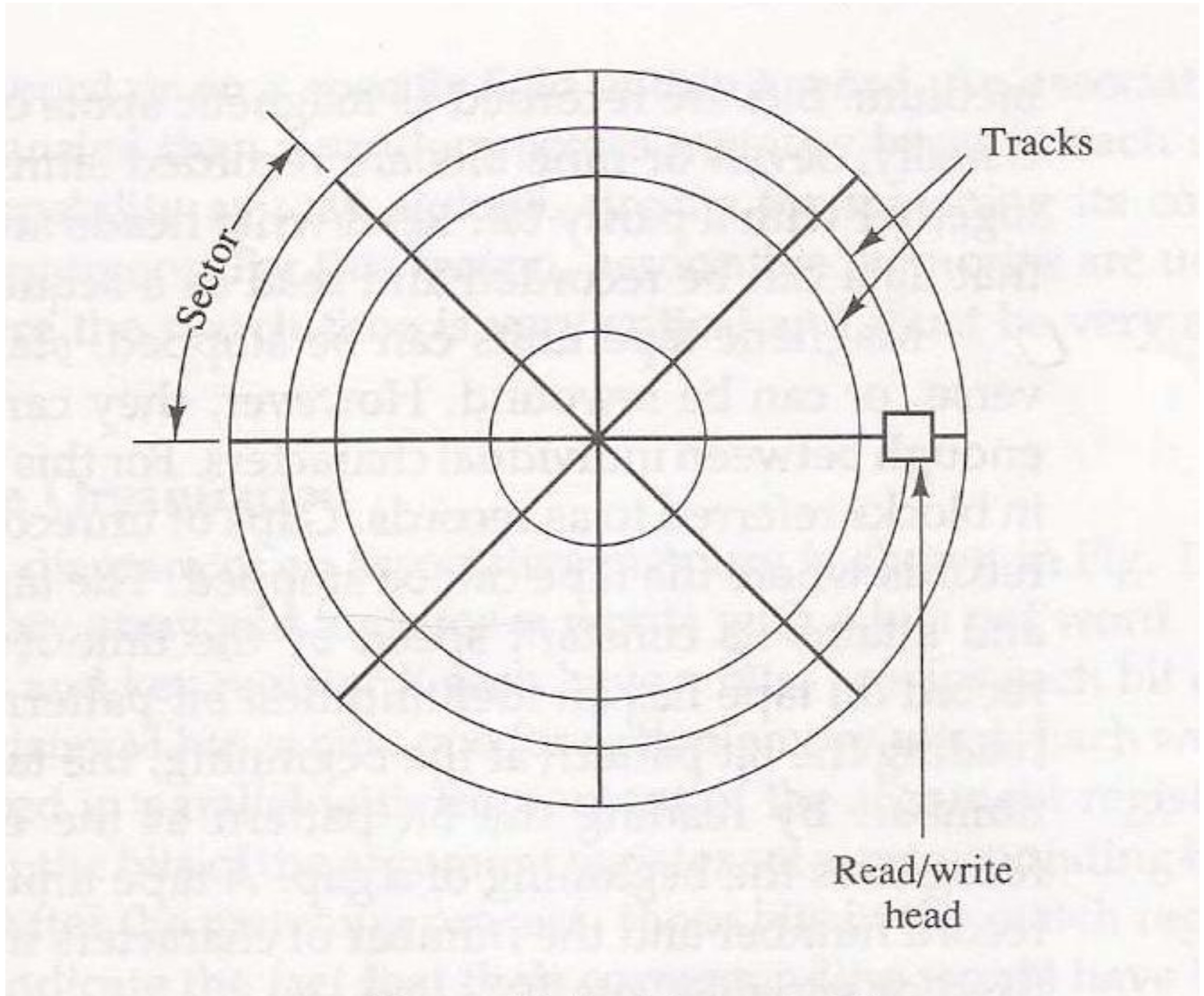


Figure 12-5 Magnetic disk.

Divide such surface in concentrical **track** .

Concentric track because there is only one centre.

Tracks also divided further into **sector.**

Collection of consecutive sector of track is called **cluster.** (only in one track)

- Disk access time = seek time + Rotational latency (delay) + 1 sector transfer time + additional delay

Sector is smallest unit of disk, which can be read or write at onces.

Sector is the addressable unit of the disc.

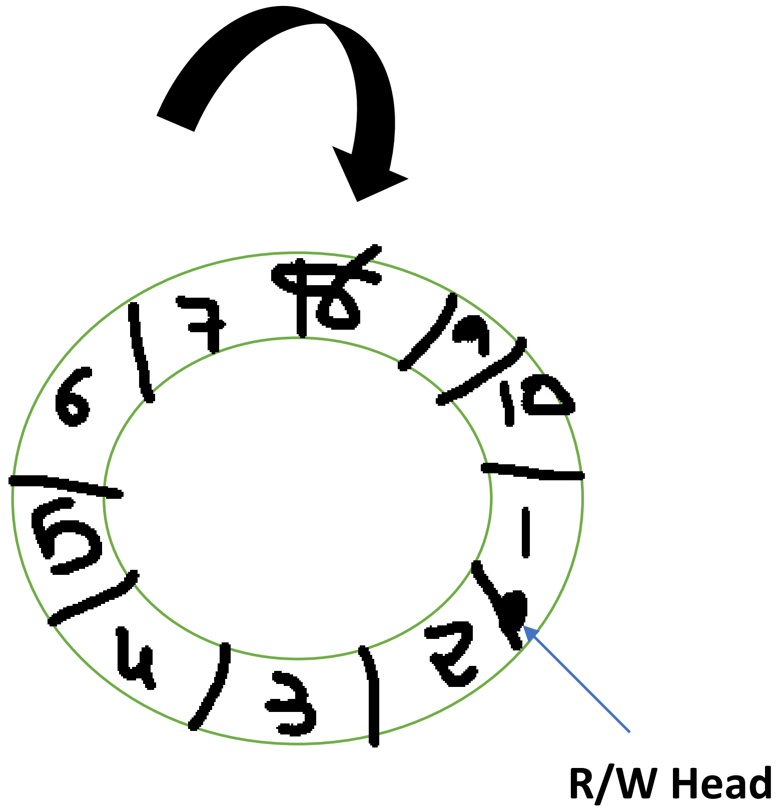
Disk access time is nothing but 1 sector access time.

All arms are move together. (1 R/W Head is outer most track then all R/W head will be on outer most track)

- First R/W head move to particular track.
- Move particular arm forward or backward some time is required. So that R/W head can reach to particular track & this time is called as ***seek time***.
- Then spindle come to a picture.
- The spindle rotate the disk & particular sector comes under R/W head, takes some time is called ***rotational latency (delay***).
- Then read or write happens. So time take is called ***transfer time***.
- Also required some additional delay if given then put otherwise zero.

Disk access time = seek time + Rotational latency (delay) + 1 sector transfer time + additional delay

- **Seek time :** Time required to position the arm over the track.
- **Rotational delay** : Time required to rotate desired sector under R/W head.
- **Transfer Time :** Time required to read or write 1 sector.
- **Additional delay :** Disk controller – hardware delay



R/W head always at end of some sector & Starting of some sector.

In fig now R/W head is at starting sector of 2 & end of sector 1.

- If read sector 2 then no any extra time required.

Note – Disk rotate in single direction.

If read sector 1 then directly not possible – rotate whole disc.

Assume, 1 disk rotation time = 10 msec

1 sector rotation time = 1disk rotation time / no of sector per track
= 10/10 = 1 msec

If current head position = start of 3rd sector
Target sector = 7th sector

→ How many time required to rotate disk for reach target sector.

Rotational latency = 4 sector rotation time (7-3)
= 4 * 1 msec
= 4 msec