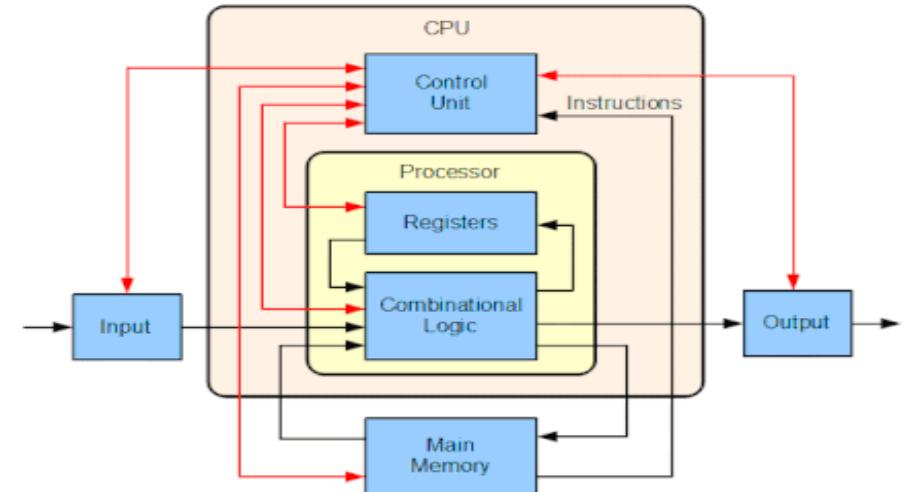
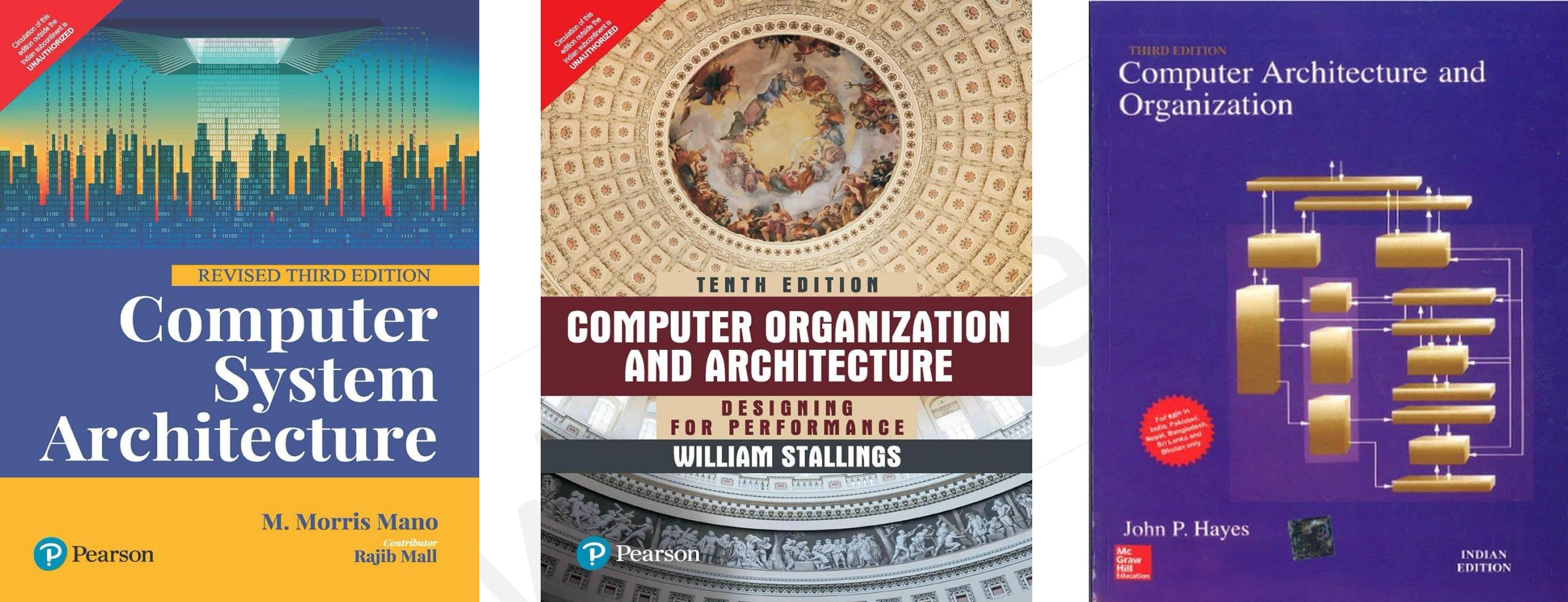


# Computer Organization and Architecture

- Core subjects for CS/IT Students
- In GATE 8-9 Marks out of 100 Marks, and 5-6 questions on an average
- In NET 20-22 Marks out of 200 marks and 10-12 questions
- Both parts are important Theory and Numerical
- Needs a little time, have to fight to scoring
- Not required in Industry





- **Syllabus:-** Machine instructions and addressing modes. ALU, data path and control unit. Instruction pipelining. Memory hierarchy: cache, main memory and secondary storage. I/O interface (interrupt and DMA mode).

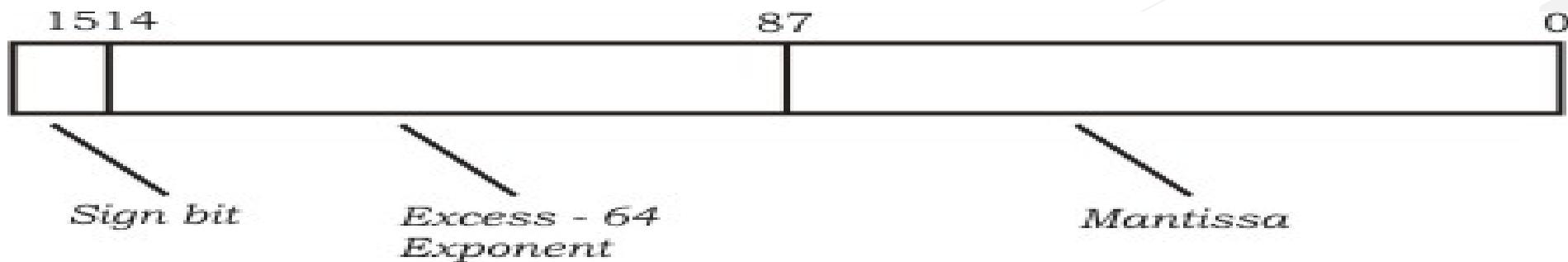
## Floating point representation

- Problem with representation we have already studied is that it do not works well if the number to be stored is either too small or too large, is it take very large amount of memory. Imagine a number  $6.023 \times 10^{23}$  will require around 70 bits to be stored. So in scientific application or statics it is a problem to store very small or very large number.
- Floating point representation a special kind of sign magnitude representation. Floating point number is stored in mantissa/exponent form i.e.  $m \times r^e$ . Mantissa is a signed magnitude fraction for most of the cases. The exponent is stored in biased form.
- The floating-point normalized number distribution is not uniform. Representable number are dense towards zero and sparses towards max value. This uneven distribution result negligible effect of rounding towards zero and dominant towards max value.

- The biased exponent is an unsigned number representing signed true exponent.
- If the biased exponent field contains K bits, the biased =  $2^{k-1}$
- True value expression is  $V = (-1)^S(.M)_2 * 2^{E-\text{Bias}}$ , note it is explicit representation
- True value expression is  $V = (-1)^S(1.M)_2 * 2^{E-\text{Bias}}$ , note it is implicit representation, it has more precision than explicit normalization.
- Biased exponent( $E$ ) = True exponent( $e$ ) + Bias
- Range of true exponent: from  $-2^{k-1}$  to  $+2^{k-1}-1$ , where k is number of bits assigned for exponent
- After adding bias  $2^{k-1}$ , new range go from 0 to  $2^k-1$

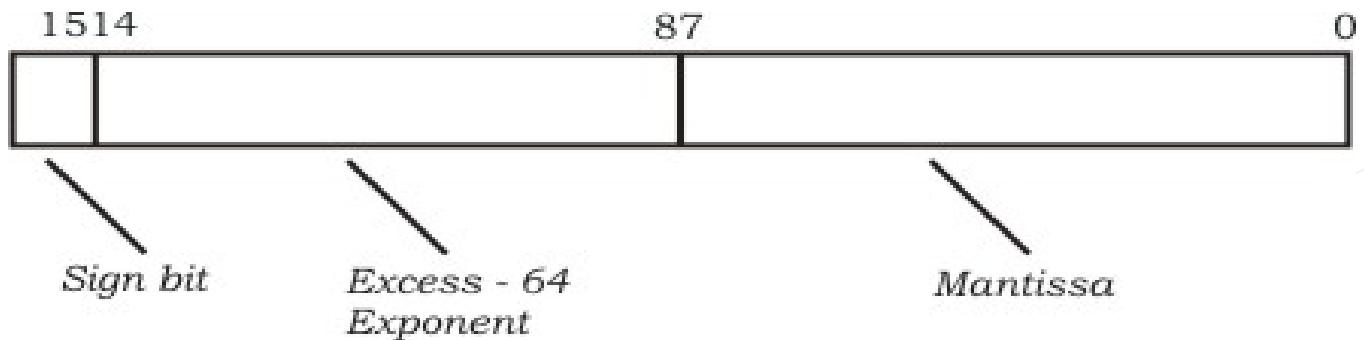
**Q** Represent -21.75 (s=1, k=7, m=8)

**Q** Consider the following floating-point format



Mantissa is a pure fraction in sign-magnitude form. The decimal number  $0.239 \times 2^{13}$  has floating point representation \_\_\_\_\_?

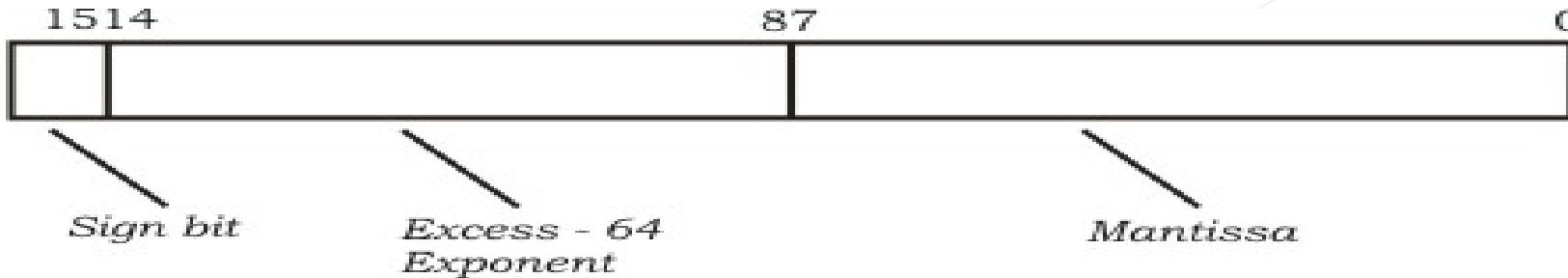
**Q Consider the following floating-point format (GATE-2005) (2 Marks)**



Mantissa is a pure fraction in sign-magnitude form. The decimal number  $0.239 \times 2^{13}$  has the following hexadecimal representation (without normalization and rounding off):

- (A) 0D24
- (B) 0D4D
- (C) 4D0D
- (D) 4D3D

**Q Consider the following floating-point format (GATE-2005) (2 Marks)**



Mantissa is a pure fraction in sign-magnitude form. The normalized representation for the above format is specified as follows. The mantissa has an implicit 1 preceding the binary (radix) point. Assume that only 0's are padded in while shifting a field. The normalized representation of the above number ( $0.239 \times 2^{13}$ ) is:

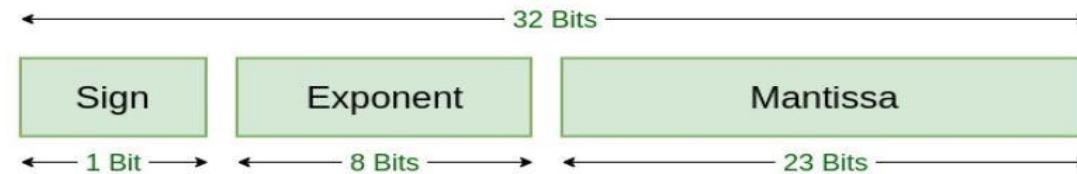
- (A) 0A 20
- (B) 11 34
- (C) 4D D0
- (D) 4A E8

# IEEE 754 floating point standard

- The **IEEE Standard for Floating-Point Arithmetic (IEEE 754)** is a technical standard for floating-point arithmetic established in 1985 by the Institute of Electrical and Electronics Engineers (IEEE).
- The standard addressed many problems found in the diverse floating-point implementations that made them difficult to use reliably and portably. Many hardware floating-point units use the IEEE 754 standard.
- It gives a provision for +0 & + $\infty$  (by reserving certain pattern for Mantissa/Exponent pattern). there are number of modes for storage from half precision (16 bits) to very lengthy notations. based of the system is 2. the floating-point number can be stored either in implicit normalized form or in fractional form. If the biased exponent field contains K bits, the biased =  $2^{k-1} - 1$ . Certain Mantissa/Exponent pattern does not denote any number (NAN i.e. not a number).

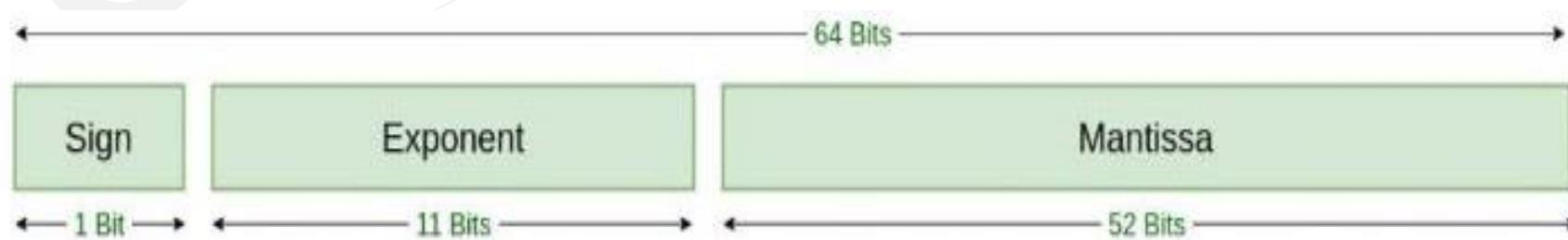
Name	Common name	Mantissa Bits	Exponent bits	Exponent bias	E min	E max
binary16	Half precision	10	5	$2^{5-1}-1 = 15$	-14	+15
binary32	Single precision	23	8	$2^{8-1}-1 = 127$	-126	+127
binary64	Double precision	52	11	$2^{11-1}-1 = 1023$	-1022	+1023
binary128	Quadruple precision	112	15	$2^{15-1}-1 = 16383$	-16382	+16383
binary256	Octuple precision	236	19	$2^{19-1}-1 = 262143$	-262142	+262143

# Single precision



Sign bit (1)	Exponent (8)	Mantissa (23)	Value
0	00.....0( $E=0$ )	00.....0( $M=0$ )	+0
1	00.....0( $E=0$ )	00.....0( $M=0$ )	-0
0	11.....1( $E=255$ )	00.....0( $M=0$ )	$+\infty$
1	11.....1( $E=255$ )	00.....0( $M=0$ )	$-\infty$
0/1	$1 \leq E \leq 254$	XX.....X ( $M \neq 0$ )	Implicit normalised number
0/1	00.....0( $E=0$ )	XX.....X ( $M \neq 0$ )	Fraction
0/1	11.....1( $E=255$ )	XX.....X ( $M \neq 0$ )	NAN (Not a Number)

## Double Precision



**Q** The format of the single-precision floating point representation of a real number as per the IEEE 754 standard is as follows:



Which one of the following choices is correct with respect to the smallest normalized positive number represented using the standard?**(GATE 2021) (1 MARKS)**

- (A) exponent =00000000 and mantissa =00000000000000000000000000000000
- (B) exponent =00000000 and mantissa =00000000000000000000000000000001
- (C) exponent =00000001 and mantissa =00000000000000000000000000000000
- (D) exponent =00000001 and mantissa =00000000000000000000000000000001

**Q** Consider three floating point numbers A , B and C stored in registers RA, RB and RC, respectively as per IEEE-754 single precision floating point format. The 32-bit content stored in these registers (in hexadecimal form) are as follows.

$$R_A = 0xC1400000 \quad R_B = 0x42100000 \quad R_C = 0x41400000$$

Which one of the following is FALSE? **(GATE 2022)(2 MARKS)**

- (a)  $A + C = 0$
- (b)  $C = A + B$
- (c)  $B = 3C$
- (d)  $(B - C) > 0$

**Q** Consider the following representation of a number in IEEE 754 single-precision floating point format with a bias of 127.

S : 1   E : 10000001     F : 11110000000000000000000000000000

Here, S,E and F denote the sign, exponent, and fraction components of the floating point representation. The decimal value corresponding to the above representation (rounded to 2 decimal places) is \_\_\_\_\_. **(GATE 2021)**

- (a) -7.75
- (b) +7.75
- (c) -5.57
- (d) 5.75

**Q** Given the following binary number in 32-bit (single precision) IEEE-754 format:

**00111110011011010000000000000000**

The decimal value closest to this floating-point number is **(GATE-2017) (2 Marks)**

**(A)** $1.45 \times 10^1$

**(B)** $1.45 \times 10^{-1}$

**(C)** $2.27 \times 10^{-1}$

**(D)** $2.27 \times 10^1$

**Q** The decimal value 0.5 in IEEE single precision floating point representation has  
**(GATE-2012) (2 Marks)**

- a) fraction bits of 000...000 and exponent value of 0
- b) fraction bits of 000...000 and exponent value of -1
- c) fraction bits of 100...000 and exponent value of 0
- d) no exact representation

**Q** The following bit pattern represents a floating-point number in IEEE 754 single precision format (GATE-2008) (1 Marks)

1 10000011 10100000000000000000000000000000

The value of the number in decimal form is

- (A) -10
- (B) -13
- (C) -26
- (D) None of these

**Q** In the IEEE floating point representation, the hexadecimal value 0x00000000 corresponds to **(GATE-2008) (2 Marks)**

a) The normalized value  $2^{-127}$

b) the normalized value  $2^{-126}$

c) the normalized value +0

d) the special value +0

**Q** How the number 85.125 will be represented in the IEEE floating point representation in single precision in hexadecimal notation ?

**Q** In the IEEE floating point representation in single precision in hexadecimal notation is 40400000, then the floating point value will be ?

**Q** If the numerical value of a 2-byte unsigned integer on a little endian computer is 255 more than that on a big endian computer, which of the following choices represent(s) the unsigned integer on a little endian computer? **(GATE 2021) (2 MARKS)**

(A) 0x66650

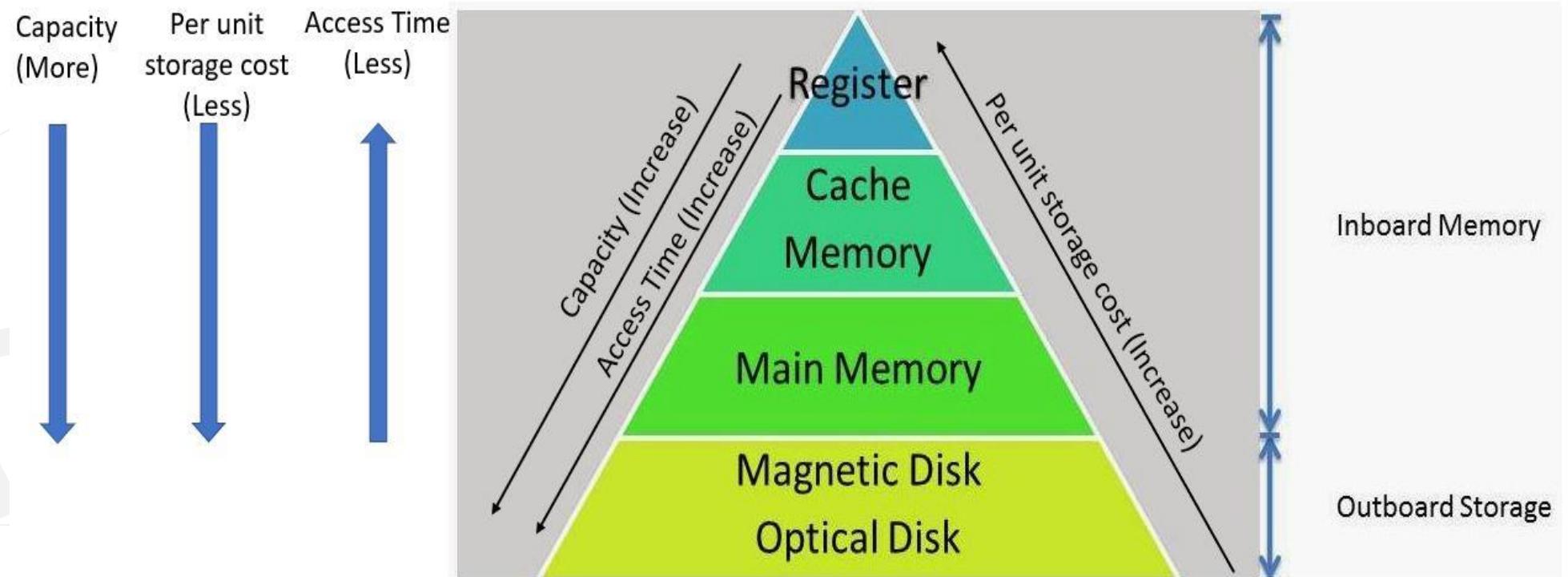
(B) X000010

(C) X42430

(D) x0100

# Memory Hierarchy

- The memory hierarchy system consists of all storage devices employed in a computer system from the slow but high-capacity auxiliary memory to a relatively faster main memory, to an even smaller and faster cache memory accessible to the high-speed processing logic.
- Let first understand what we need from a memory
  - Large capacity
  - Less per unit cost
  - Less access time(fast access)





**Cycle**



**Car**



**Airbus**



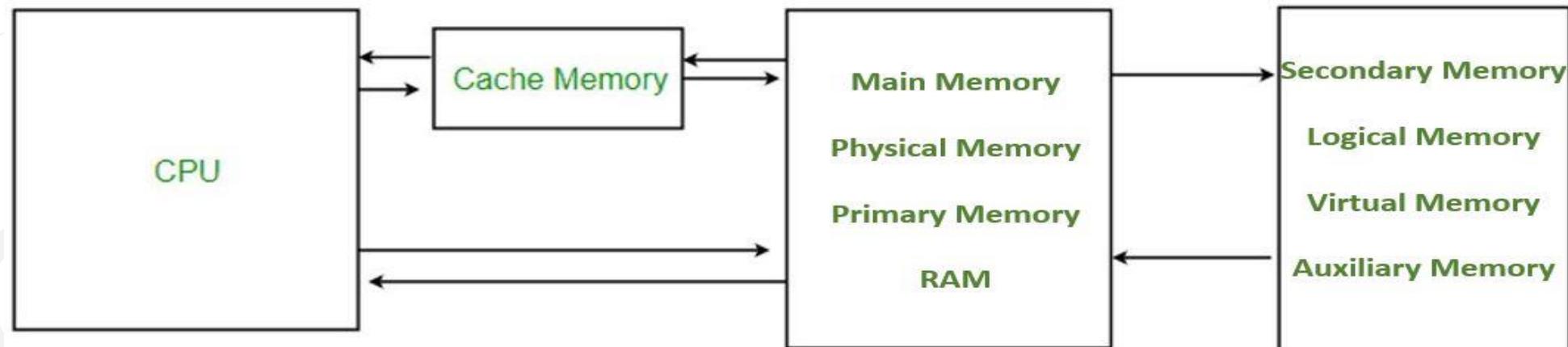
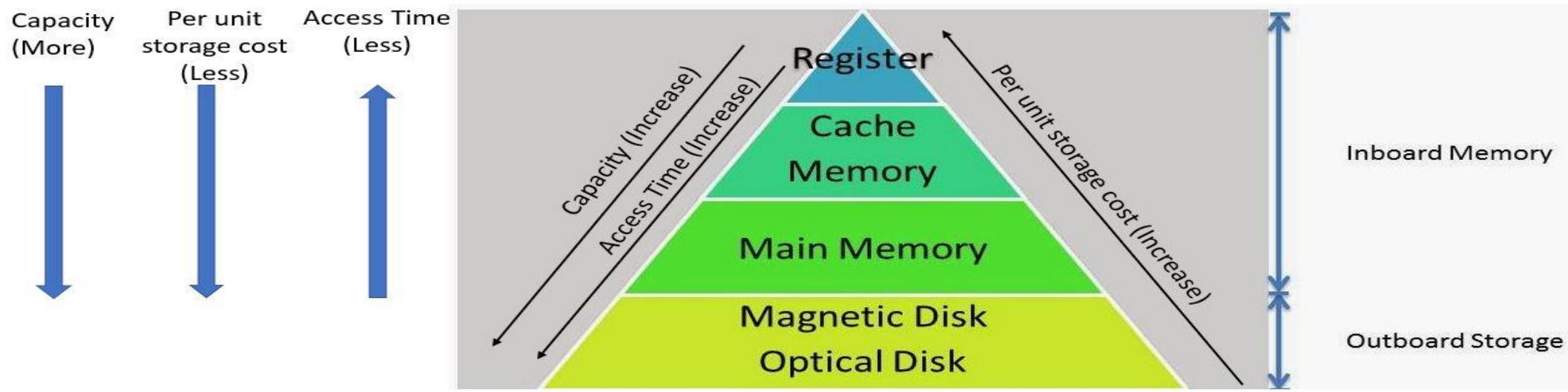
**Lathi**



**303**



**AK-47**





**Showroom**

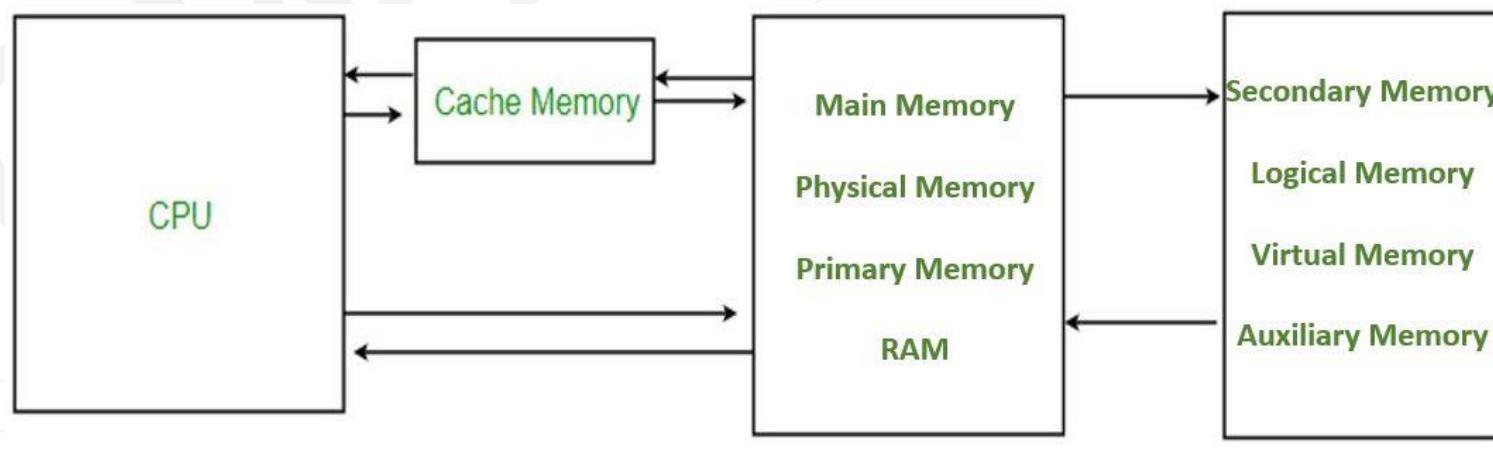


**Go down**



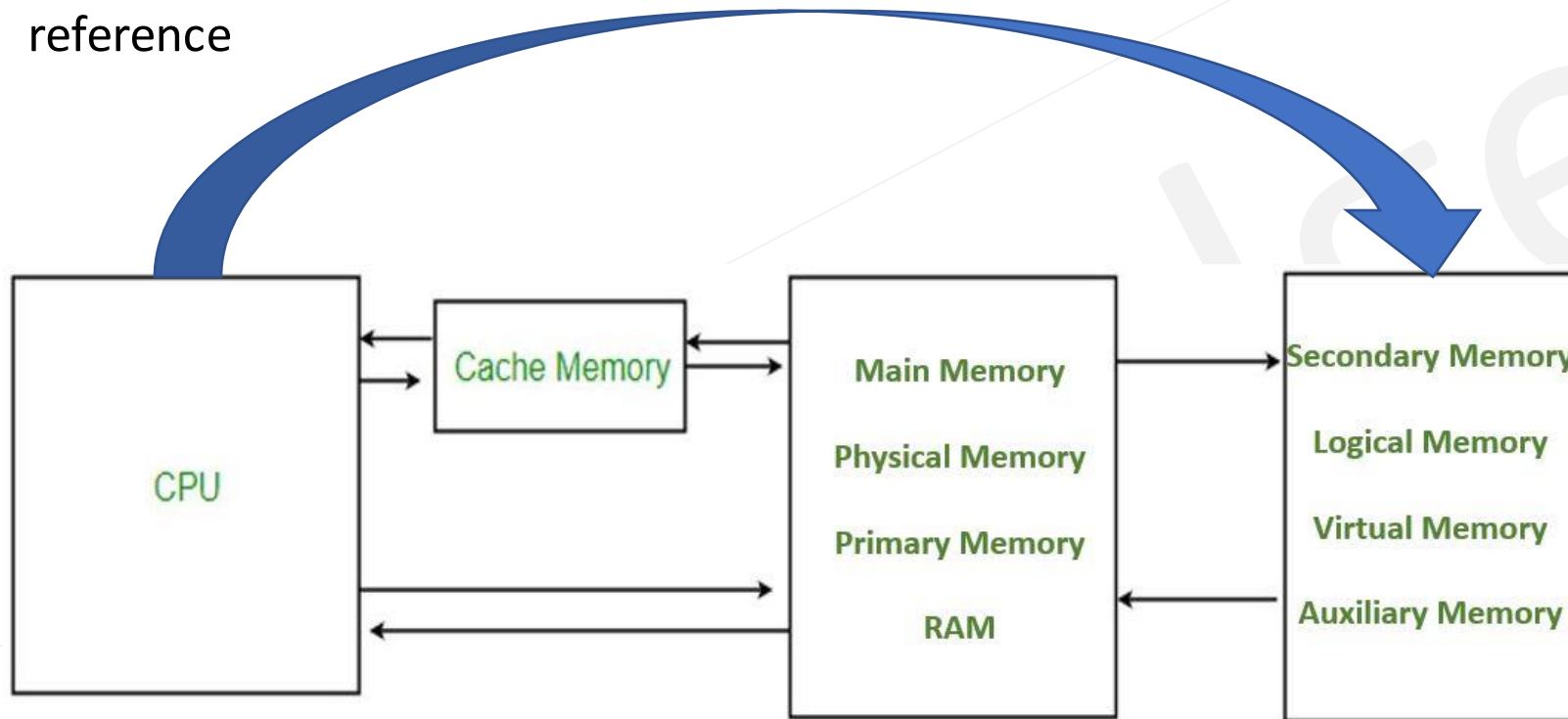
**Factory**

- If we want this hierarchy to work fine, then most of the time when CPU needs a data it must be present in Cache, if not possible main memory, worst case Secondary memory. But this is a difficult task to do as my computer has, 8 TB of Secondary Memory, 32 GB of Main Memory, but only 768KB, 4MB, 16 MB of L<sub>1</sub>, L<sub>2</sub>, L<sub>3</sub> cache respectively. If somehow we can estimate what data CPU will require in future we can prefetch in Cache and Main Memory. Locality of reference Helps us to perform this estimation.
- The main memory occupies a central position by being able to communicate directly with the CPU. When programs not residing in main memory are needed by the CPU, they are brought in from auxiliary memory. Programs not currently needed in main memory are transferred into auxiliary memory to provide space for currently used programs and data.
- A special very-high speed memory called a Cache is used to increase the speed of processing by making current programs and data available to the CPU at a rapid rate. The cache memory is employed in computer systems to compensate for the speed differential between main memory access time and processor logic.

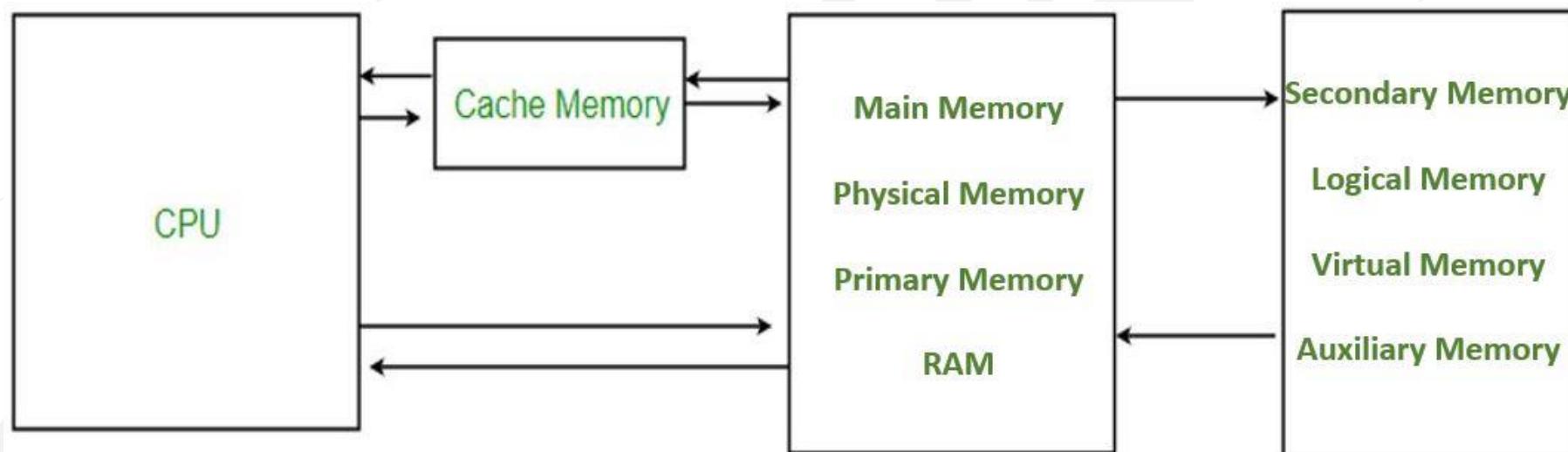


# Locality of Reference

- The references to memory at any given interval of time tend to be confined within a few localized areas in memory. This phenomenon is known as the property of locality of reference. There are two types of locality of reference



- **Spatial Locality:** Spatial locality refers to the use of data elements in the nearby locations.
- **Temporal Locality:** Temporal locality refers to the reuse of specific data, and/or resources, within a relatively small-time duration. Or, the most frequently used items will be needed soon. (LRU is used for temporal locality)



## Importance of Memory

- Memory is one of the most crucial components of a computer. Some reasons why memory plays a vital role in computer's overall performance :
  - **Storage of data**: Memory allows the processor to access the data quickly.
  - **Multitasking**: Memory enables multiple tasks simultaneously, as it allows the processor to switch between different programs and data efficiently.
  - **Running applications**: Applications and programs require a certain amount of memory to run.
  - **Operating system**: The operating system also requires memory to run smoothly.
- In summary, memory is critical for the efficient operation of a computer and plays a key role in determining its performance.

**Q** The principle of locality justifies the use of: **(Gate-1995) (1 Marks)**

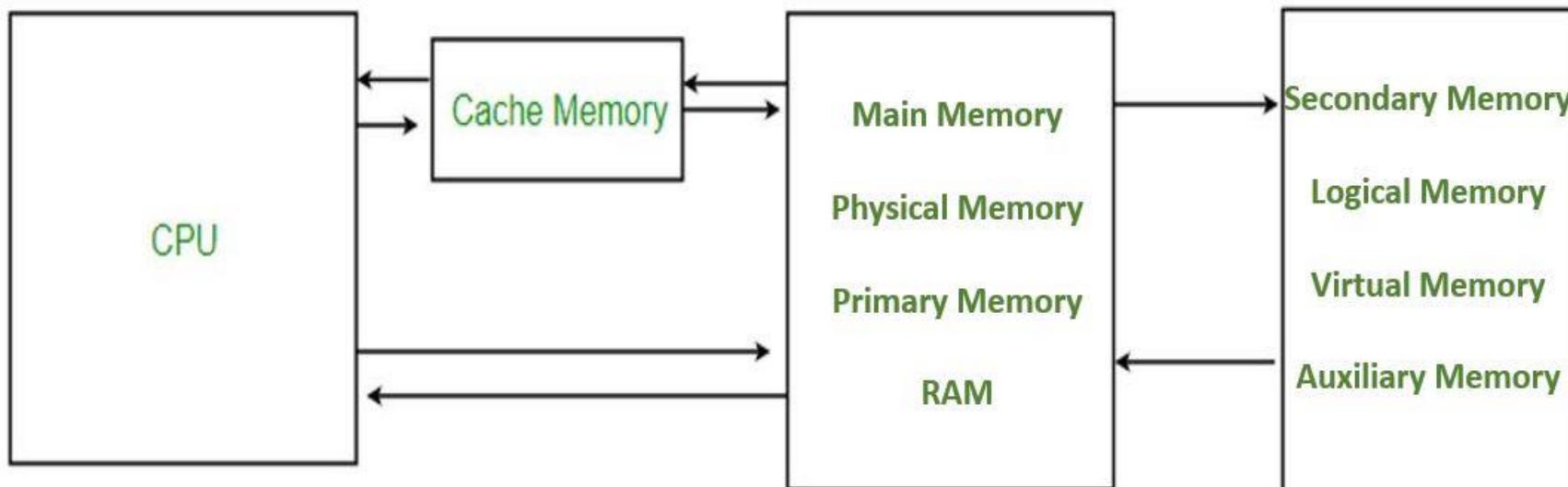
a) Interrupts

b) DMA

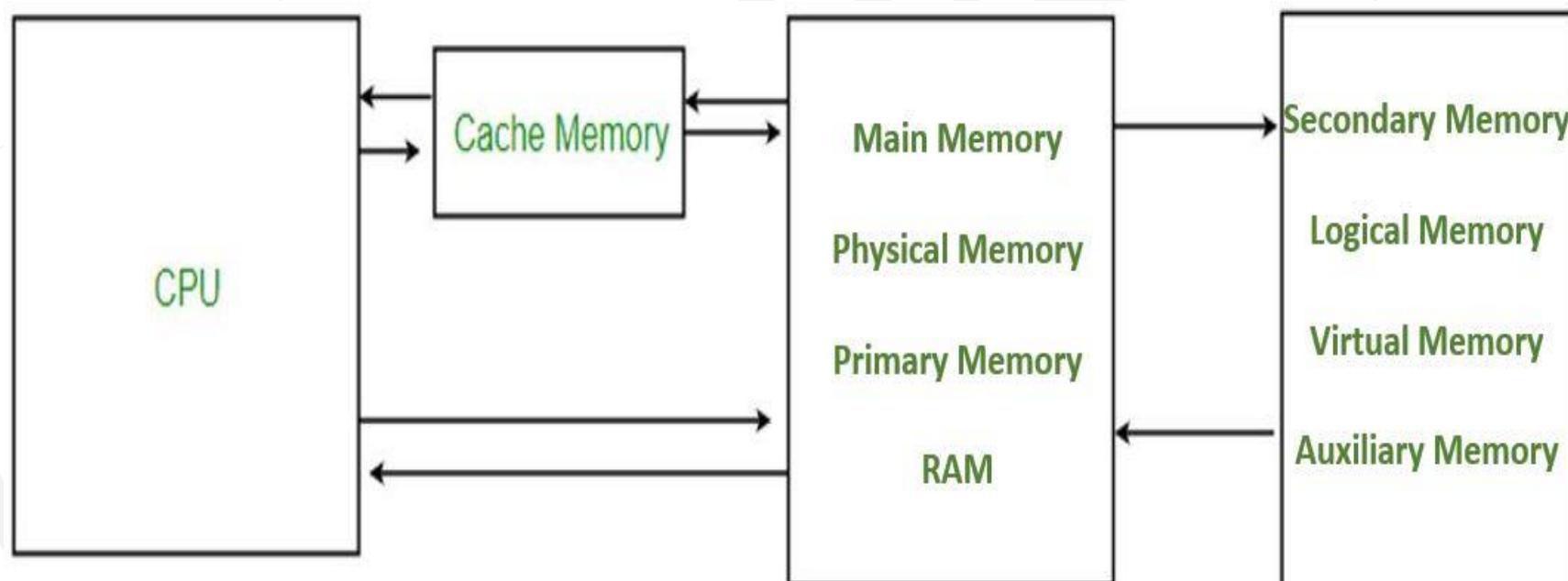
c) Polling

d) Cache Memory

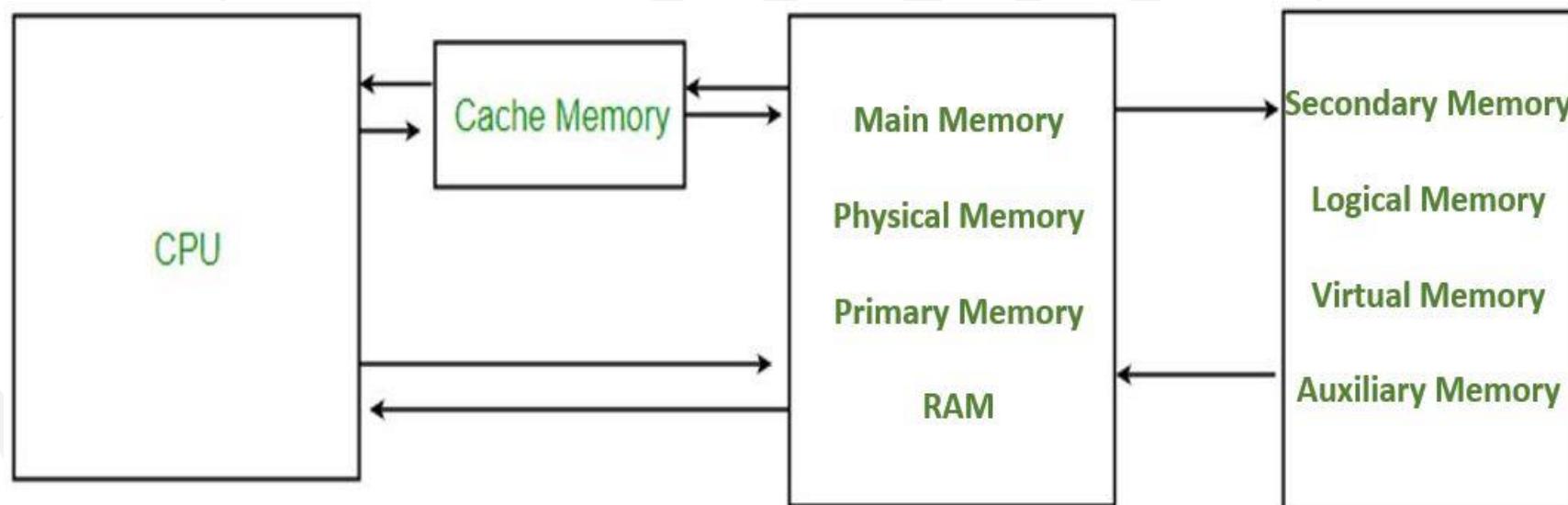
- **Cache Hit**: When a program requests data, the cache is searched first, and if the data is found in the cache, it is returned to the program. This is known as a cache hit. Cache hits are important because they significantly improve the performance of a computer system. By retrieving data from the cache, instead of the slower main memory or disk storage.
- **Hit Ratio**: The cache hit rate is the ratio of data access requests that are being satisfied by the cache. This Ratio is impacted by various factors, such as the size of the cache, the behavior of the programs using the cache etc.



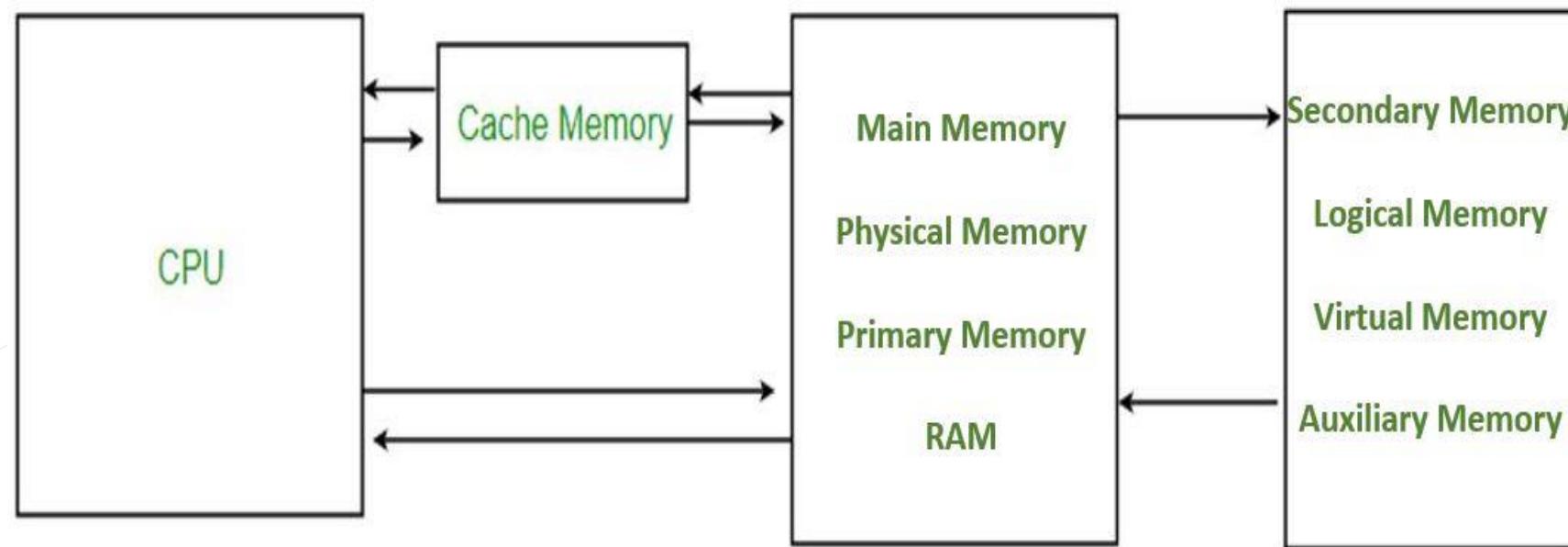
- **Hit Latency:** Hit latency refers to the time it takes for a computer system to retrieve data from the cache memory when a cache hit occurs. It measures the amount of time it takes from the moment a data access request is made to the moment the requested data is returned to the requesting program.
- There are several factors that can impact the hit latency of a cache, including the size of the cache, the architecture of the cache, the technology used to implement the cache, and the behavior of the programs using the cache.



- **Cache Miss:** A cache miss occurs when a computer system tries to retrieve data from the cache memory, but the data is not found in the cache. This means that the data must be retrieved from the main memory or disk storage.
- To reduce the number of cache misses and improve performance, several strategies can be used, such as increasing the size of the cache, optimizing the cache replacement algorithm, and using more efficient data storage strategies.

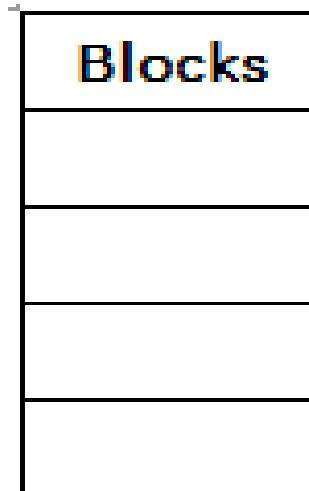


- **Miss Latency:** Cache miss latency refers to the time it takes for a computer system to retrieve data from the main memory or disk storage when a cache miss occurs. It is the time it takes from the moment a data access request is made to the moment the requested data is returned to the requesting program, in the case where the data is not found in the cache.

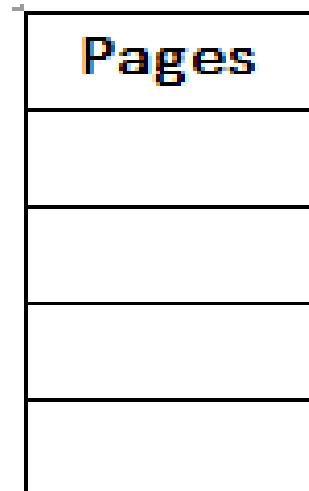




**Cache Memory**



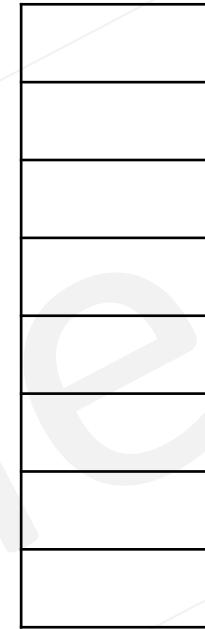
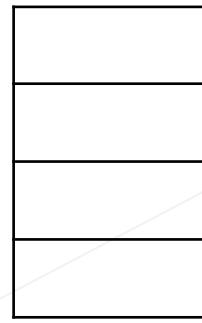
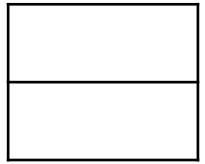
**Main Memory**



**Secondary Memory**

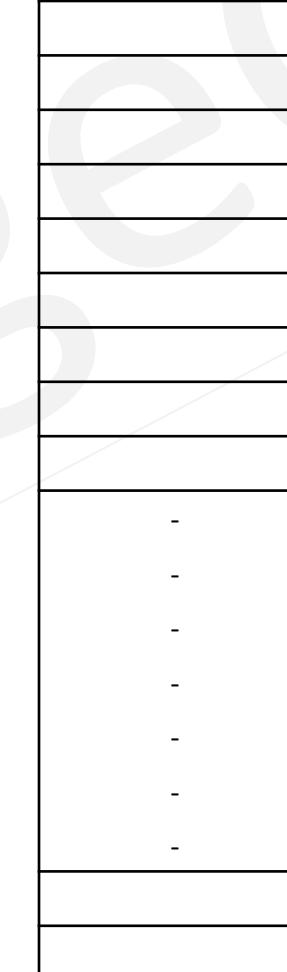
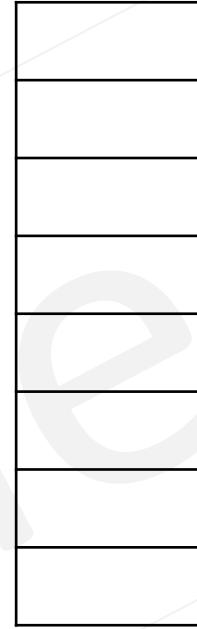
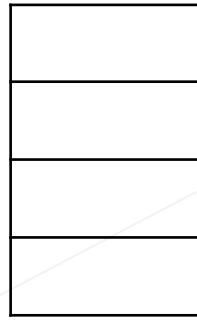
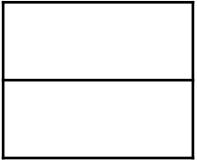
$10^3$	1 Thousand	$10^3$	1 kilo	$2^{10}$	1 kilo
$10^6$	1 Million	$10^6$	1 Mega	$2^{20}$	1 Mega
$10^9$	1 Billion	$10^9$	1 Giga	$2^{30}$	1 Giga
$10^{12}$	1 Trillion	$10^{12}$	1 Tera	$2^{40}$	1 Tera
		$10^{15}$	1 Peta	$2^{50}$	1 Peta

<b>Address Length in bits</b>	$n$
<b>No of Locations</b>	$2^n$



Memory Size = Number of Location \* Size of each Location

Address Length in bits	Upper Bound( $\log_2 n$ )
No of Locations	n



		Main Memory				
		B-0	W-0	W-1	W-2	W-3
B-0			W-0	W-1	W-2	W-3
B-1			W-1	W-2	W-3	W-4
B-2			W-2	W-3	W-4	W-5
B-3			W-3	W-4	W-5	W-6
B-4			W-4	W-5	W-6	W-7
B-5			W-5	W-6	W-7	W-8
B-6			W-6	W-7	W-8	W-9
B-7			W-7	W-8	W-9	W-10
B-8			W-8	W-9	W-10	W-11
B-9			W-9	W-10	W-11	W-12
B-10			W-10	W-11	W-12	W-13
B-11			W-11	W-12	W-13	W-14
B-12			W-12	W-13	W-14	W-15
B-13			W-13	W-14	W-15	W-16
B-14			W-14	W-15	W-16	W-17
B-15			W-15	W-16	W-17	W-18

**Q More than one word are put in one cache block to (Gate-2001) (1 Marks)**

**(A) Exploit the temporal locality of reference in a program**

**(B) Exploit the spatial locality of reference in a program**

**(C) Reduce the miss penalty**

**(D) None of the above**

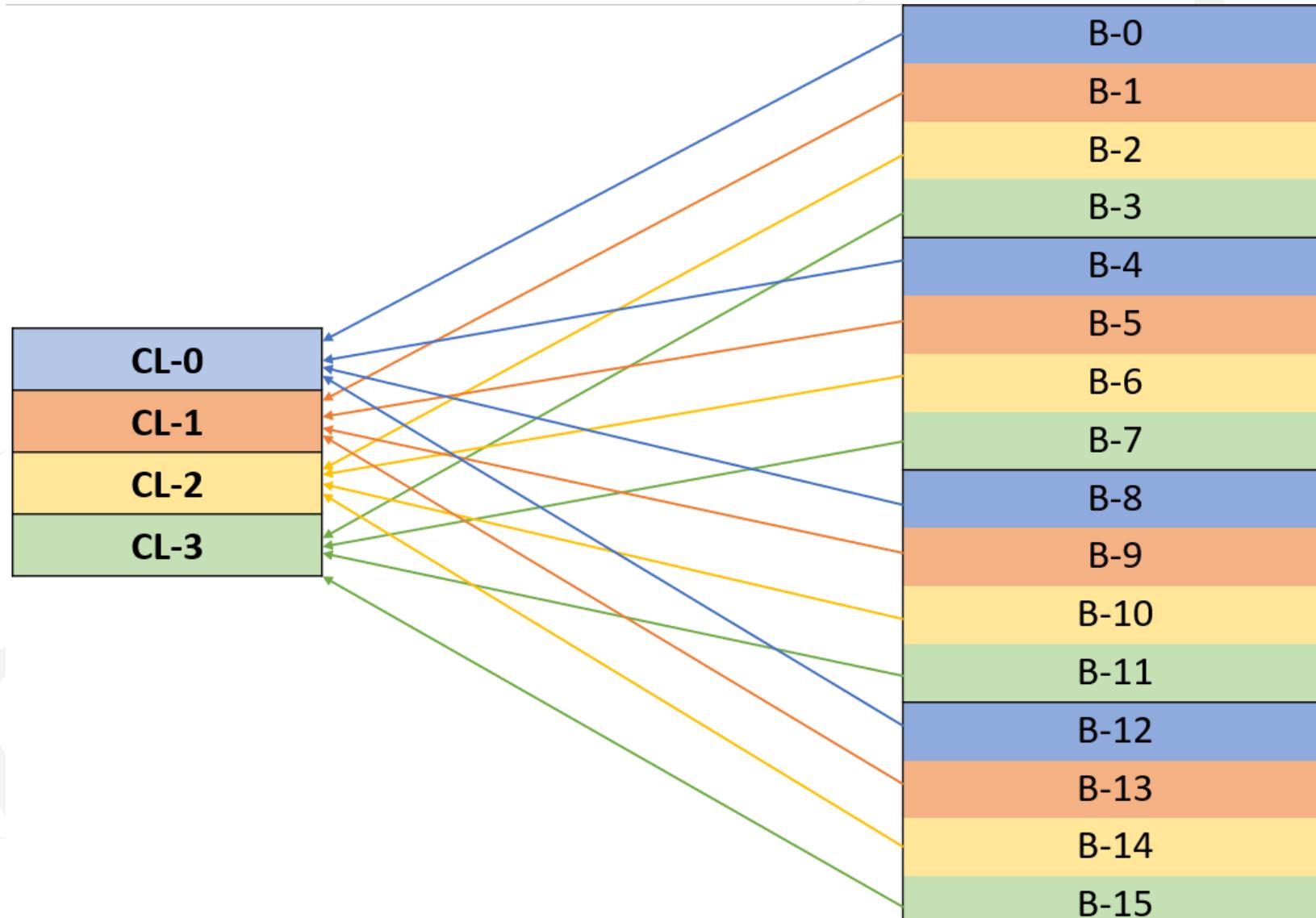
**Q** A processor can support a maximum memory of 4 GB, where the memory is word-addressable (a word consists of two bytes). The size of the address bus of the processor is at \_\_\_\_\_ least bits. **(Gate-2016) (1 Marks)**

- Cache mapping refers to the process of determining where data should be stored in the cache memory.
- The cache mapping algorithm determines which cache lines are assigned to which main memory blocks.

<b>CL-0</b>
<b>CL-1</b>
<b>CL-2</b>
<b>CL-3</b>

B-0
B-1
B-2
B-3
B-4
B-5
B-6
B-7
B-8
B-9
B-10
B-11
B-12
B-13
B-14
B-15

- There are several types of cache mapping algorithms, including:
  - **Direct mapping:** Each main memory block can be stored in only one specific cache line.

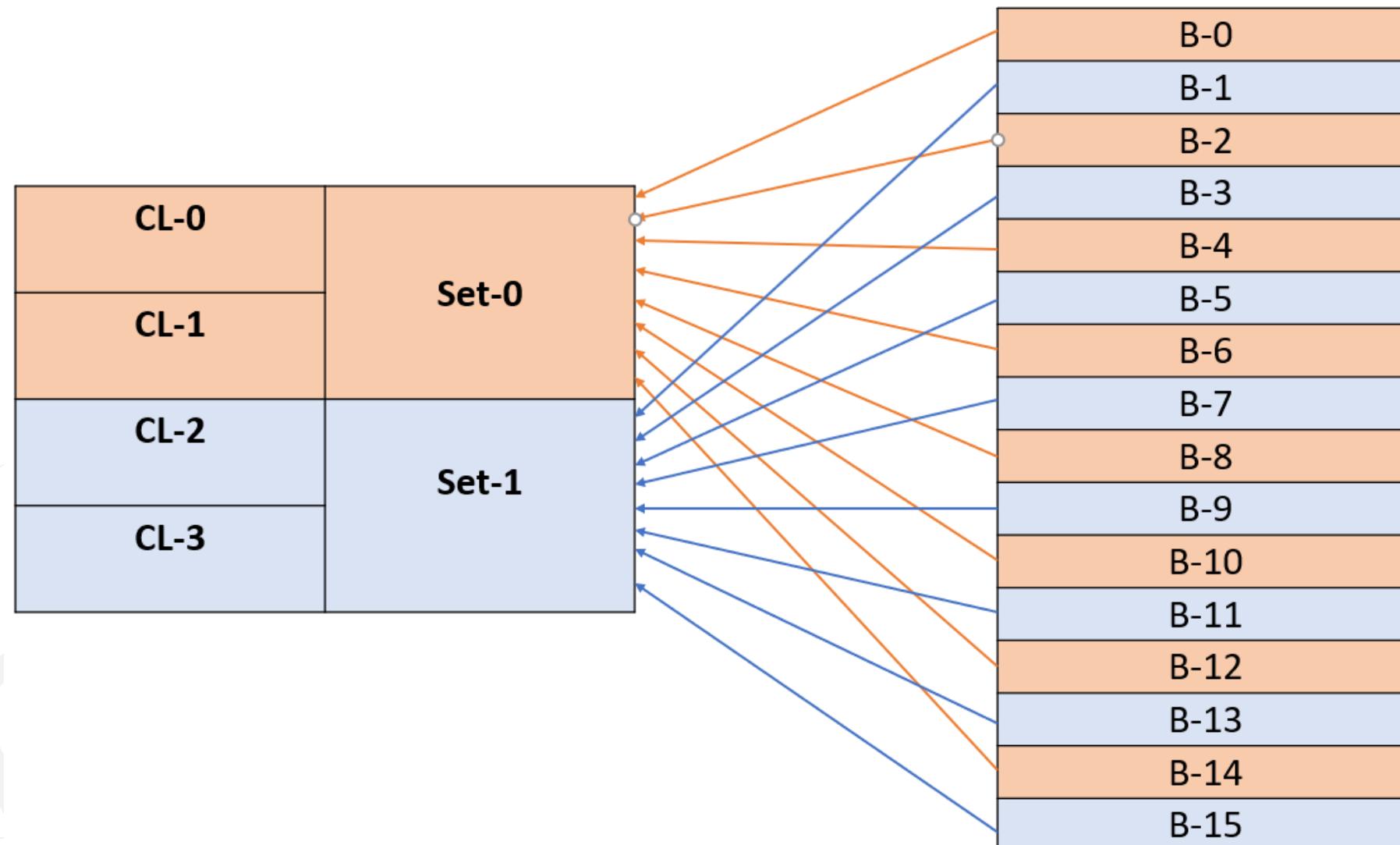


- **Associative mapping:** Any main memory block can be stored in any cache line.

<b>CL-0</b>
<b>CL-1</b>
<b>CL-2</b>
<b>CL-3</b>

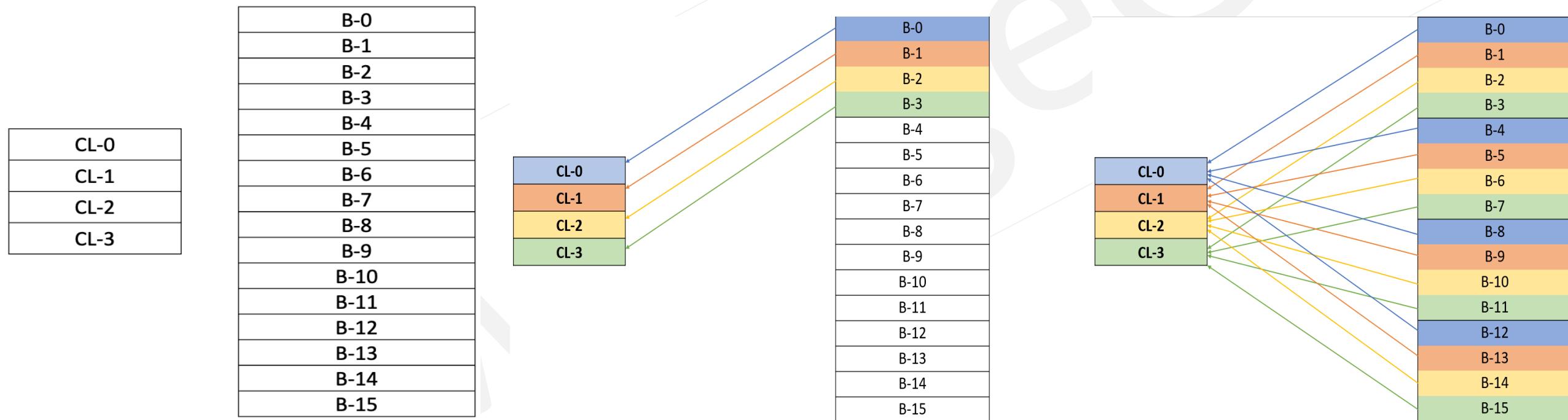
B-0
B-1
B-2
B-3
B-4
B-5
B-6
B-7
B-8
B-9
B-10
B-11
B-12
B-13
B-14
B-15

- **Set-associative mapping:** The cache is divided into sets, each of which contains several cache lines. A main memory block can be stored in any cache line within a set.

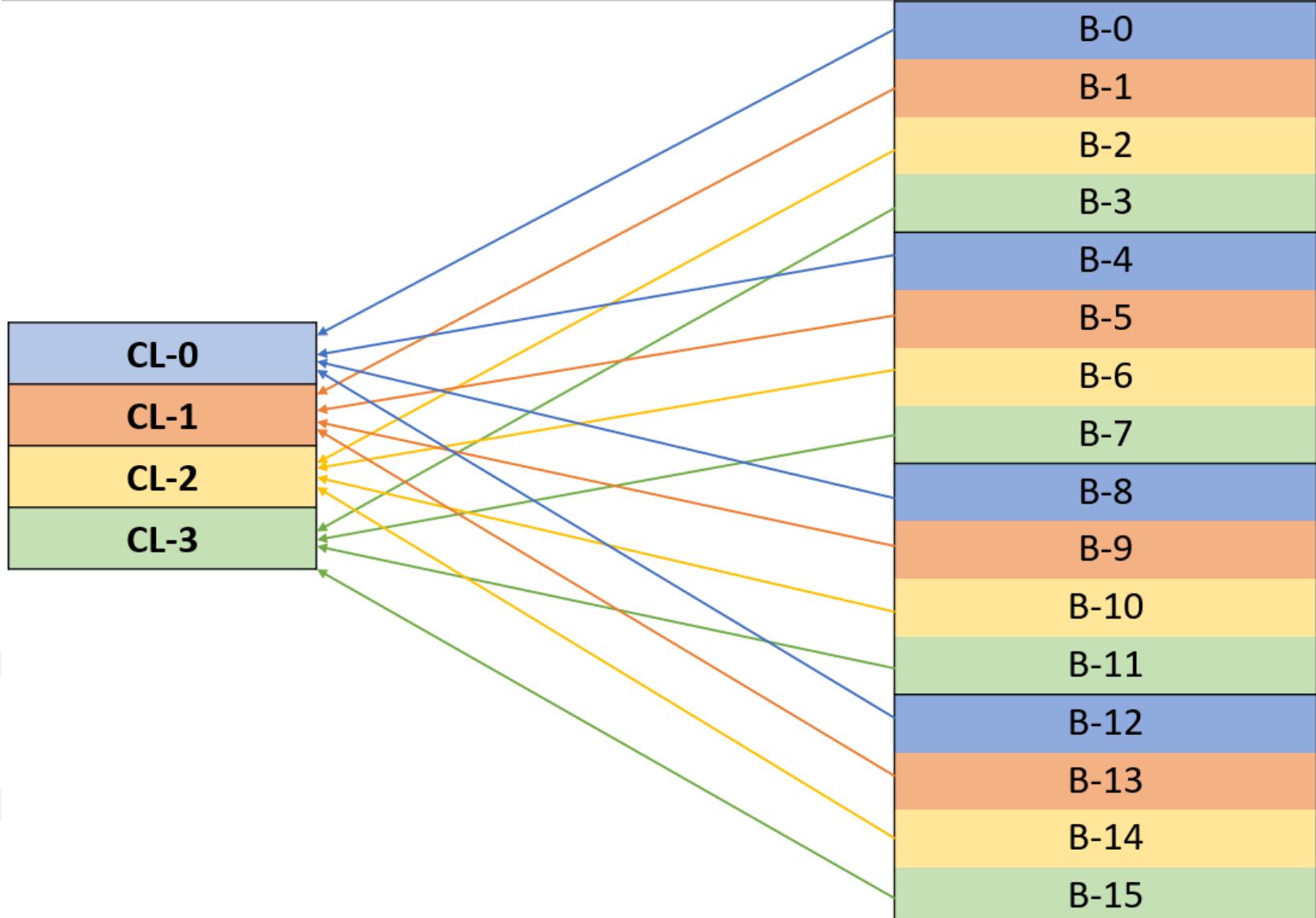


# Direct Mapping

- Direct mapping is a type of cache mapping algorithm that maps each main memory block to a specific cache line.



$\frac{\text{Block No}}{\text{No of Cache Line}} = \text{Remainder is the Address of block in Cache}$



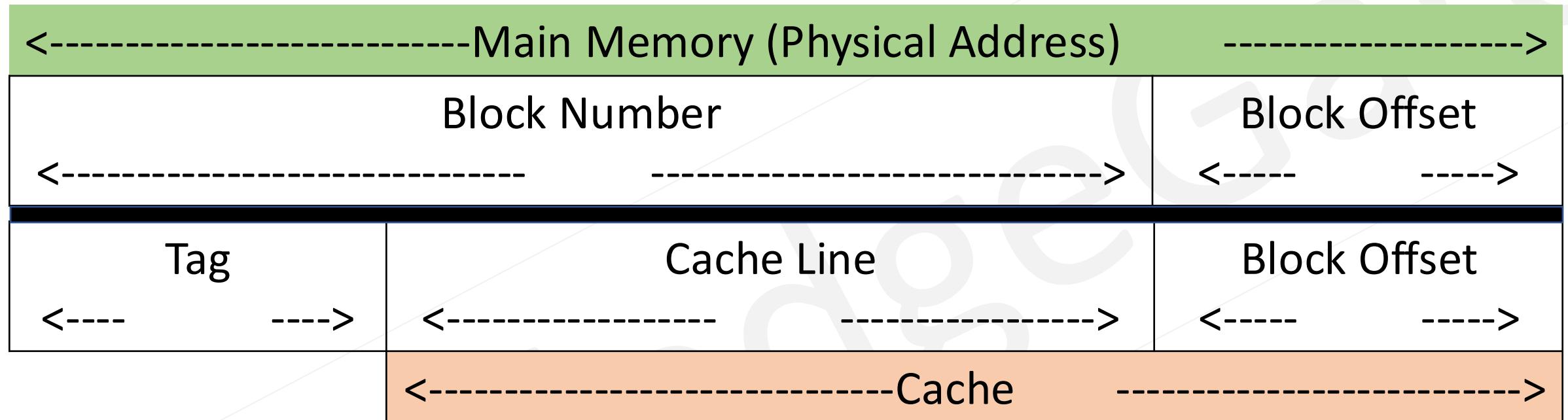
## Cache Memory

<b>CL-0</b>	B-0 / B-4 / B-8 / B-12
<b>CL-1</b>	B-1 / B-5 / B-9 / B-13
<b>CL-2</b>	B-2 / B-6 / B-10 / B-14
<b>CL-3</b>	B-3 / B-7 / B-11 / B-15

Cache Memory		
<b>CL-0</b>	TAG	W-
		W-
		W-
		W-
<b>CL-1</b>	TAG	W-
		W
		W-
		W-
<b>CL-2</b>	TAG	W-
		W-
		W-
		W-
<b>CL-3</b>	TAG	W-
		W-
		W-
		W-

## Cache

<b>CL-0</b>	B-0	W-0	B-4	W-16	B-8	W-32	B-12	W-48
		W-1		W-17		W-33		W-49
		W-2		W-18		W-34		W-50
		W-3		W-19		W-35		W-51
<b>CL-1</b>	B-1	W-4	B-5	W-20	B-9	W-36	B-13	W-52
		W-5		W-21		W-37		W-53
		W-6		W-22		W-38		W-54
		W-7		W-23		W-39		W-55
<b>CL-2</b>	B-2	W-8	B-6	W-24	B-10	W-40	B-14	W-56
		W-9		W-25		W-41		W-57
		W-10		W-26		W-42		W-58
		W-11		W-27		W-43		W-59
<b>CL-3</b>	B-3	W-12	B-7	W-28	B-11	W-44	B-15	W-60
		W-13		W-29		W-45		W-61
		W-14		W-30		W-46		W-62
		W-15		W-31		W-47		W-63



## Cache Memory

CL-0	B-0 / B-4 / B-8 / B-12
CL-1	B-1 / B-5 / B-9 / B-13
CL-2	B-2 / B-6 / B-10 / B-14
CL-3	B-3 / B-7 / B-11 / B-15

Cache Memory		
CL-0	TAG 11	W-
		W-
		W <sub>100111</sub>
		W-
		W-
CL-1	TAG 01	W-
		W
		W-
		W-
CL-2	TAG 01	W-
		W-
		W-
		W-
CL-3	TAG 10	W-
		W-
		W-
		W-

## Cache

CL-0	B-0	W-0	B-4	W-16	B-8	W-32	B-12	W-48
		W-1		W-17		W-33		
		W-2		W-18		W-34		
		W-3		W-19		W-35		
CL-1	B-1	W-4	B-5	W-20	B-9	W-36	B-13	W-52
		W-5		W-21		W-37		
		W-6		W-22		W-38		
		W-7		W-23		W-39		
CL-2	B-2	W-8	B-6	W-24	B-10	W-40	B-14	W-56
		W-9		W-25		W-41		
		W-10		W-26		W-42		
		W-11		W-27		W-43		
CL-3	B-3	W-12	B-7	W-28	B-11	W-44	B-15	W-60
		W-13		W-29		W-45		
		W-14		W-30		W-46		
		W-15		W-31		W-47		

1) W-18

0 1 0 0 1 0

2) W-51

1 1 0 0 1 1

3) W-39

1 0 0 1 1 1

4) W-22

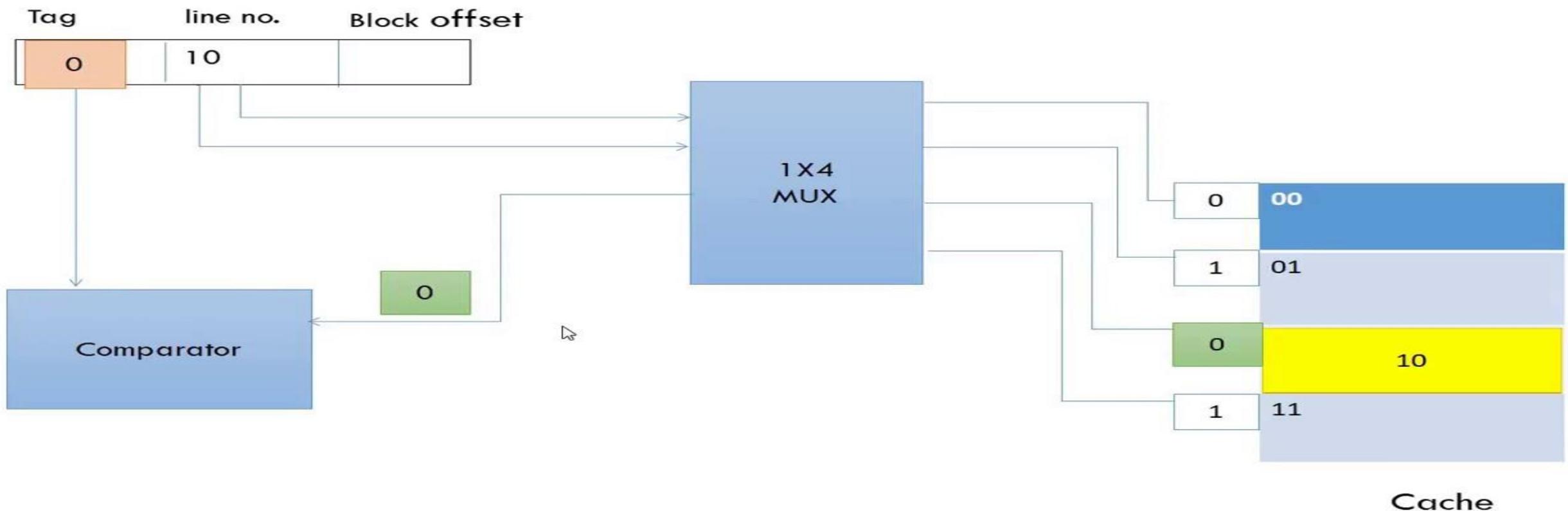
0 1 0 1 1 0

5) W-24

0 1 1 0 0 0

6) W-56

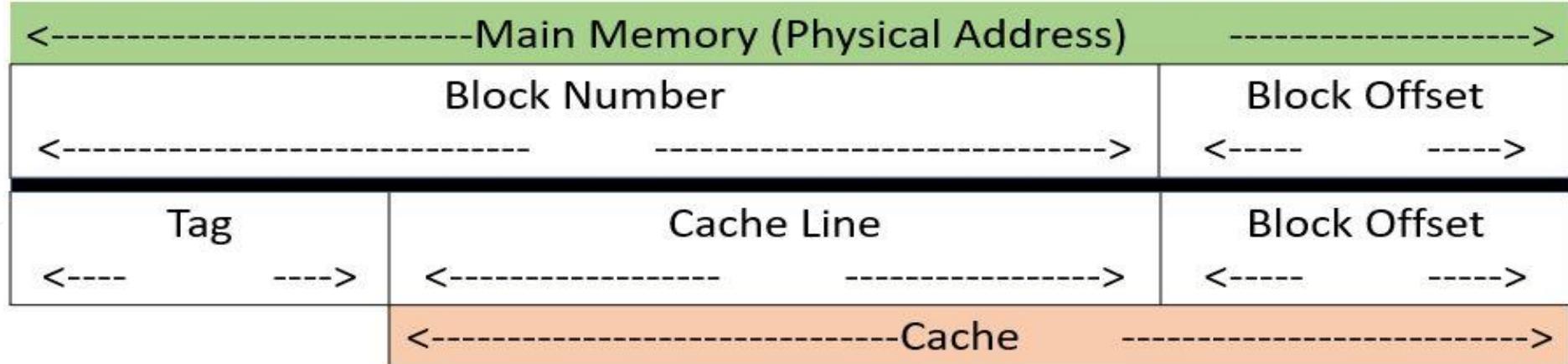
1 1 1 0 0 0



- Tag directory size = Number of tags x Tag size = Number of lines in cache x Number of bits in tag

- The main advantage of direct mapping is its simplicity. Since each main memory block maps to only one specific cache line, there is no need for complicated search algorithms to determine the location of a memory block in the cache. This makes direct mapping relatively fast and efficient.
- However, the simplicity of direct mapping also has its drawbacks. Since each cache line can store only one main memory block, it is possible for two memory blocks with different addresses to map to the same cache line. This is known as a cache conflict and can lead to cache thrashing, where the cache is constantly replacing one block with another, reducing the cache hit rate.
- Overall, direct mapping is suitable for small cache memories, where the likelihood of cache conflicts is low, and the benefits of simplicity outweigh the drawbacks.

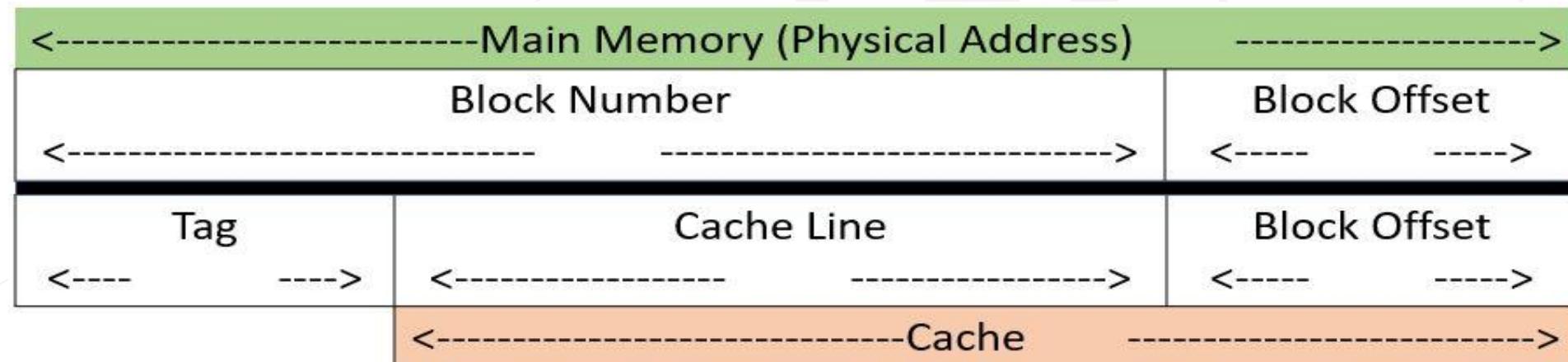
MM Size	Cache Size	Block Size	No of bits in Tag	Tag Directory Size
16 GB	32 MB	4 KB		
128 MB	256 KB	512 B		
32 GB	128 MB	1 KB		
256 MB	16 KB	1 KB		
4 GB	8 MB	2 KB	.	
512 KB	2 KB	128 B		



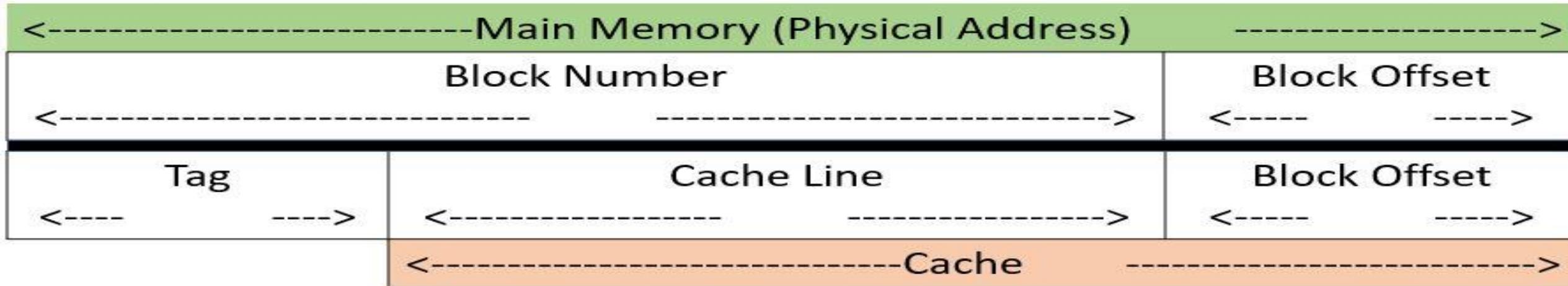
MM Size	Cache Size	Block Size	No of bits in Tag	Tag Directory Size
128 KB	16 KB	256 B	3	
32 GB	32 KB	1 KB	20	
$2^{26}$ B	512 KB	1 KB	7	
16 GB	$2^{24}$ B	4 KB	10	
64 MB	$2^{16}$ B	?	10	
$2^{26}$ B	512 KB	?	7	

**Q** Consider a computer system with a byte-addressable primary memory of size  $2^{32}$  bytes. Assume the computer system has a direct-mapped cache of size 32 KB (1 KB =  $2^{10}$  bytes), and each cache block is of size 64 bytes. The size of the tag field is \_\_\_\_\_ bits. (GATE 2021)

- (a) 17      (b) 18      (c) 15      (d) 9



**Q** Consider a machine with byte addressable memory of  $2^{32}$  bytes divided into blocks of size 32 bytes. Assume a direct mapped cache having 512 cache lines is used with this machine. The size of tag field in bits is \_\_\_\_\_ (Gate-2017) (2 Marks)



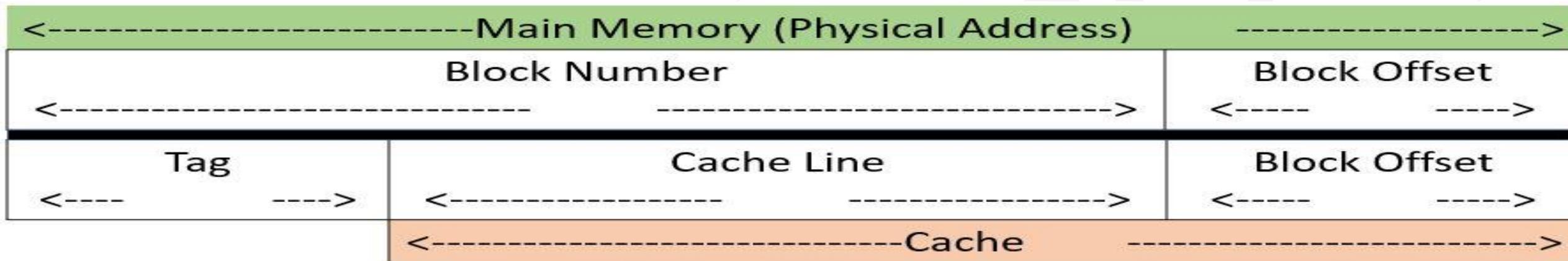
**Q** Consider a direct mapped cache of size 32 KB with block size 32 bytes. The CPU generates 32 bit addresses. The number of bits needed for cache indexing and the number of tag bits are respectively **(Gate-2005) (2 Marks)**

**(A) 10, 17**

**(B) 10, 22**

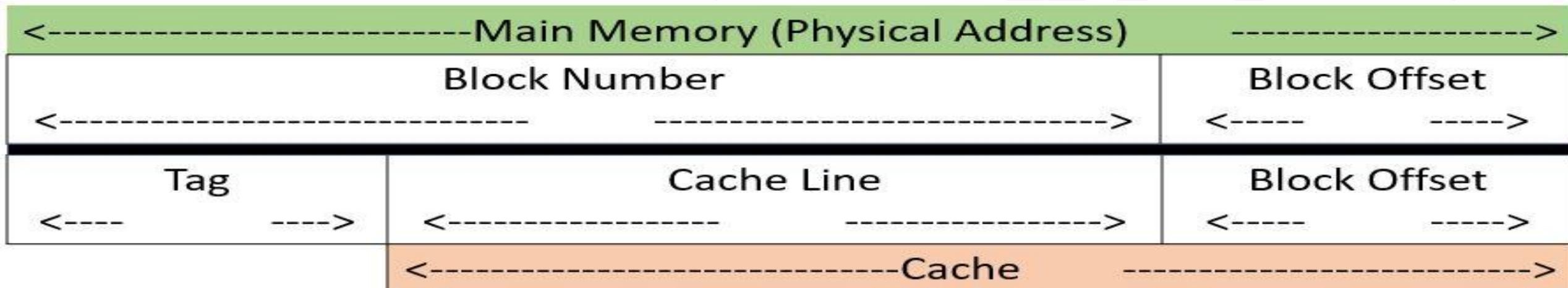
**(C) 15, 17**

**(D) 5, 17**



**Q** Consider a machine with a byte addressable main memory of  $2^{20}$  bytes, block size of 16 bytes and a direct mapped cache having  $2^{12}$  cache lines. Let the addresses of two consecutive bytes in main memory be  $(E201F)_{16}$  and  $(E2020)_{16}$ . What are the tag and cache line address (in hex) for main memory address  $(E201F)_{16}$ ? **(Gate-2015) (1 Marks)**

- (A) E, 201      (B) F, 201      (C) E, E20      (D) 2, 01F



**Q** An 8KB direct-mapped write-back cache is organized as multiple blocks, each of size 32-bytes. The processor generates 32-bit addresses. The cache controller maintains the tag information for each cache block comprising of the following.

1 Valid bit

1 Modified bit

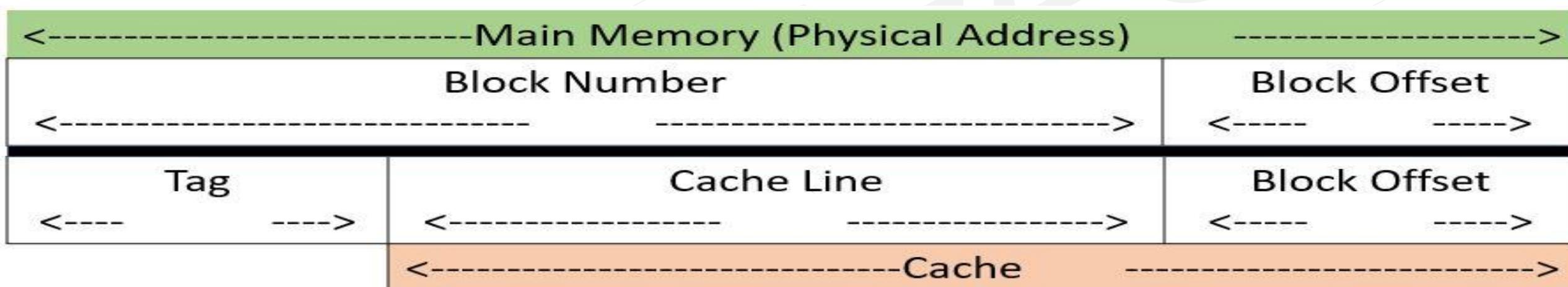
As many bits as the minimum needed to identify the memory block mapped in the cache. What is the total size of memory needed at the cache controller to store meta-data (tags) for the cache? **(Gate-2011) (2 Marks)**

**(A) 4864 bits**

**(B) 6144 bits**

**(C) 6656 bits**

**(D) 5376 bits**

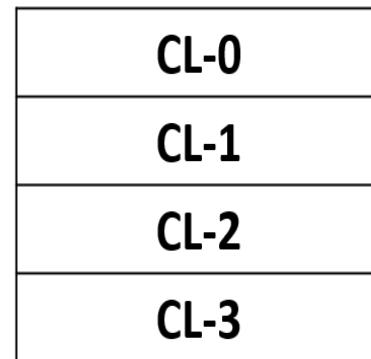


**Q** Consider a system with 2 KB direct mapped data cache with a block size of 64 bytes. The system has a physical address space of 64 KB and a word length of 16 bits. During the execution of a program, four data words P, Q, R, and S are accessed in that order 10 times (i.e., PQRSPQRS...). Hence, there are 40 accesses to data cache altogether. Assume that the data cache is initially empty and no other data words are accessed by the program. The addresses of the first bytes of P, Q, R, and S are 0xA248, 0xC28A, 0xCA8A, and 0xA262, respectively. For the execution of the above program, which of the following statements is/are TRUE with respect to the data cache? **(GATE 2022) (2 MARKS)**

- (A) Every access to S is a hit.
- (B) Once P is brought to the cache it is never evicted.
- (C) At the end of the execution only R and S reside in the cache.
- (D) Every access to R evicts Q from the cache.

## Associative Mapping

- To overcome the problem of conflict-miss in direct mapping we have Associative Mapping. A block of main memory can be mapped to any freely available cache line. This makes fully associative mapping more flexible than direct mapping. It is also known as many to many mappings.



B-0
B-1
B-2
B-3
B-4
B-5
B-6
B-7
B-8
B-9
B-10
B-11
B-12
B-13
B-14
B-15

## Cache Memory

<b>CL-0</b>	TAG = Block no	W-
		W-
		W-
		W-
<b>CL-1</b>	TAG = Block no	W-
		W
		W-
		W-
<b>CL-2</b>	TAG = Block no	W-
		W
		W-
		W-
<b>CL-3</b>	TAG = Block no	W-
		W
		W-
		W-

## Main Memory

B-0	W-0	W-1	W-2	W-3
<b>B-1</b>	W-4	W-5	W-6	W-7
<b>B-2</b>	W-8	W-9	W-10	W-11
<b>B-3</b>	W-12	W-13	W-14	W-15
<b>B-4</b>	W-16	W-17	W-18	W-19
<b>B-5</b>	W-20	W-21	W-22	W-23
<b>B-6</b>	W-24	W-25	W-26	W-27
<b>B-7</b>	W-28	W-29	W-30	W-31
<b>B-8</b>	W-32	W-33	W-34	W-35
<b>B-9</b>	W-36	W-37	W-38	W-39
<b>B-10</b>	W-40	W-41	W-42	W-43
<b>B-11</b>	W-44	W-45	W-46	W-47
<b>B-12</b>	W-48	W-49	W-50	W-51
<b>B-13</b>	W-52	W-53	W-54	W-55
<b>B-14</b>	W-56	W-57	W-58	W-59
<b>B-15</b>	W-60	W-61	W-62	W-63

## Cache Memory

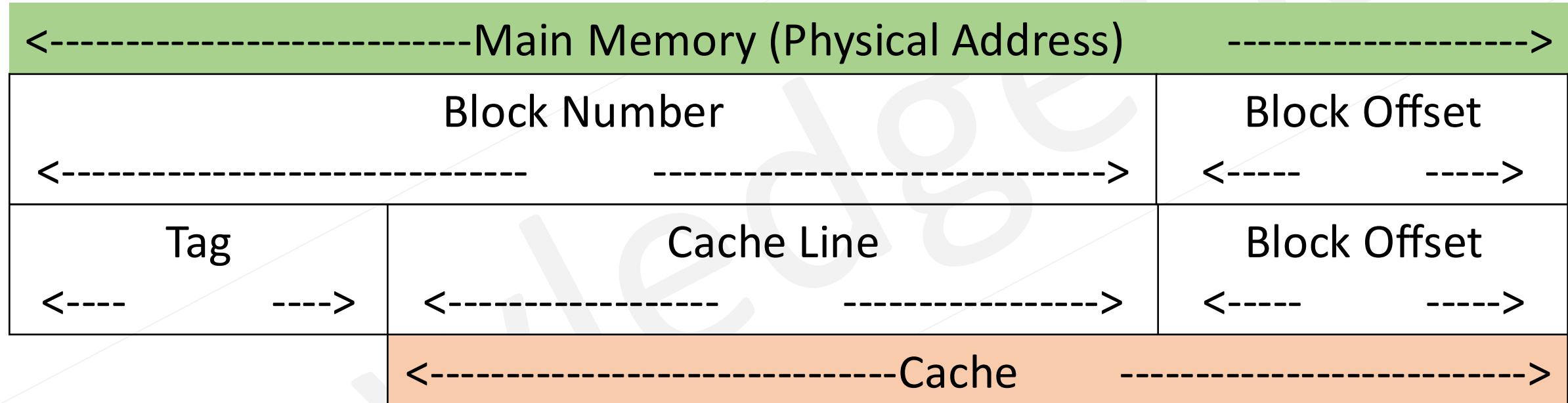
<b>CL-0</b>	B-0 / B-4 / B-8 / B-12
<b>CL-1</b>	B-1 / B-5 / B-9 / B-13
<b>CL-2</b>	B-2 / B-6 / B-10 / B-14
<b>CL-3</b>	B-3 / B-7 / B-11 / B-15

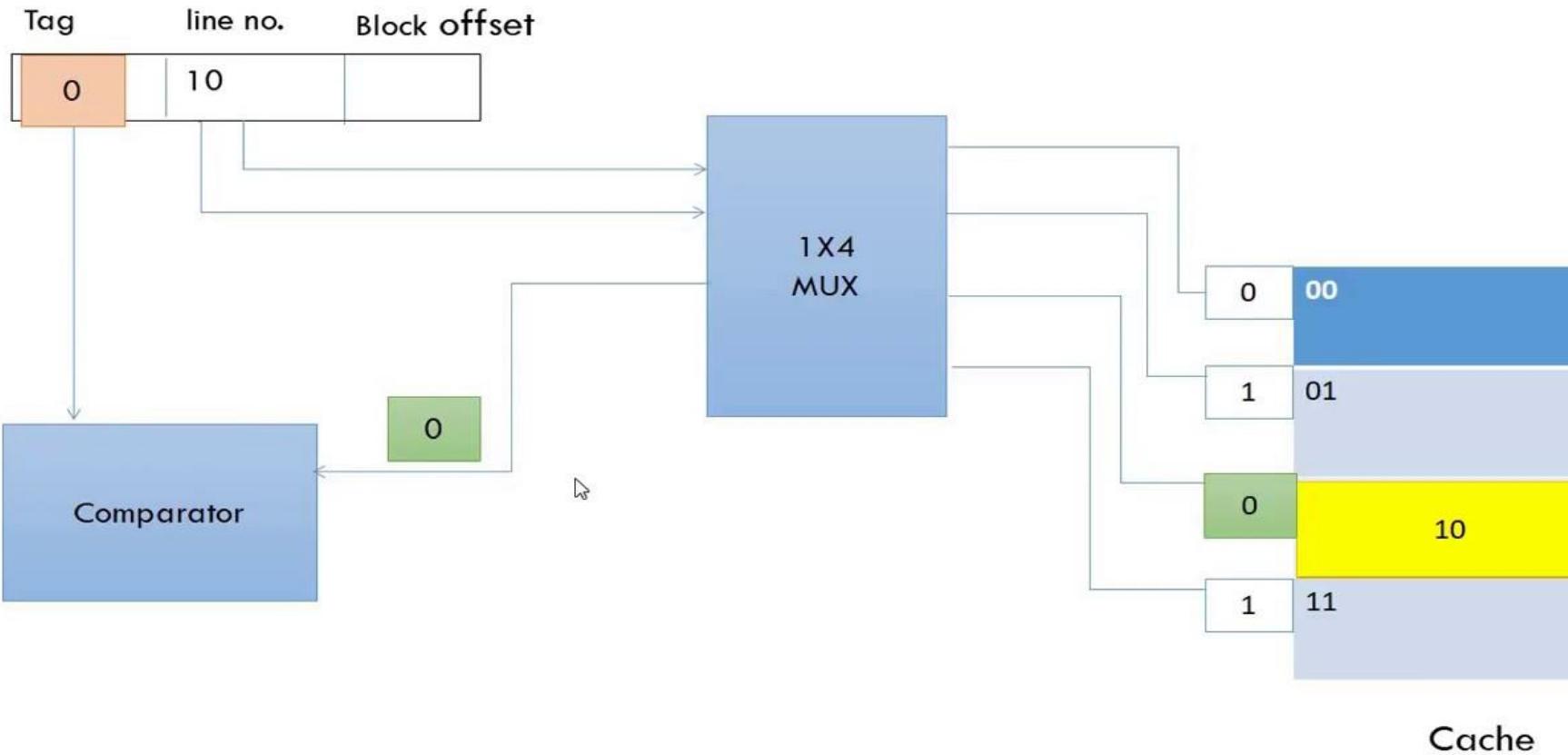
Cache Memory		
<b>CL-0</b>	TAG	W-
		W-
		W-
		W-
<b>CL-1</b>	TAG	W-
		W
		W-
		W-
<b>CL-2</b>	TAG	W-
		W-
		W-
		W-
<b>CL-3</b>	TAG	W-
		W-
		W-
		W-

## Cache

<b>CL-0</b>	B-0	W-0 W-1 W-2 W-3	B-4	W-16 W-17 W-18 W-19	B-8	W-32 W-33 W-34 W-35	B-12	W-48 W-49 W-50 W-51
<b>CL-1</b>	B-1	W-4 W-5 W-6 W-7	B-5	W-20 W-21 W-22 W-23	B-9	W-36 W-37 W-38 W-39	B-13	W-52 W-53 W-54 W-55
<b>CL-2</b>	B-2	W-8 W-9 W-10 W-11	B-6	W-24 W-25 W-26 W-27	B-10	W-40 W-41 W-42 W-43	B-14	W-56 W-57 W-58 W-59
<b>CL-3</b>	B-3	W-12 W-13 W-14 W-15	B-7	W-28 W-29 W-30 W-31	B-11	W-44 W-45 W-46 W-47	B-15	W-60 W-61 W-62 W-63

- A replacement algorithm is needed to replace a block if the cache is full. In fully associative mapping we only have two fields: Tag/Block Number field and a Block offset field. Here the number of bits in tag = number of bits to require to represent block number.

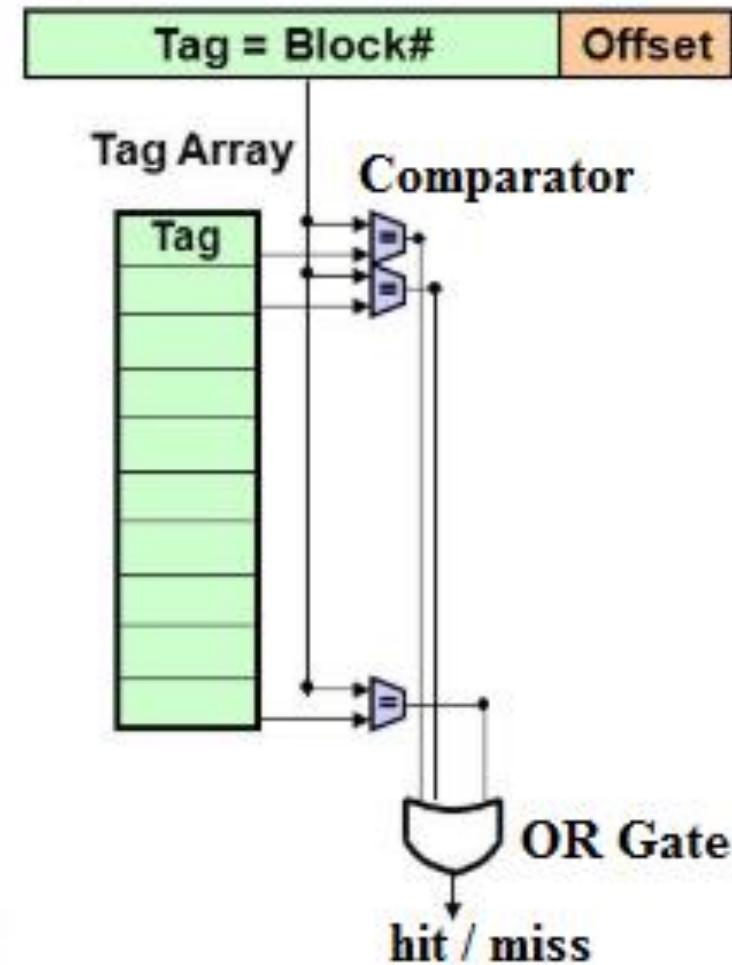




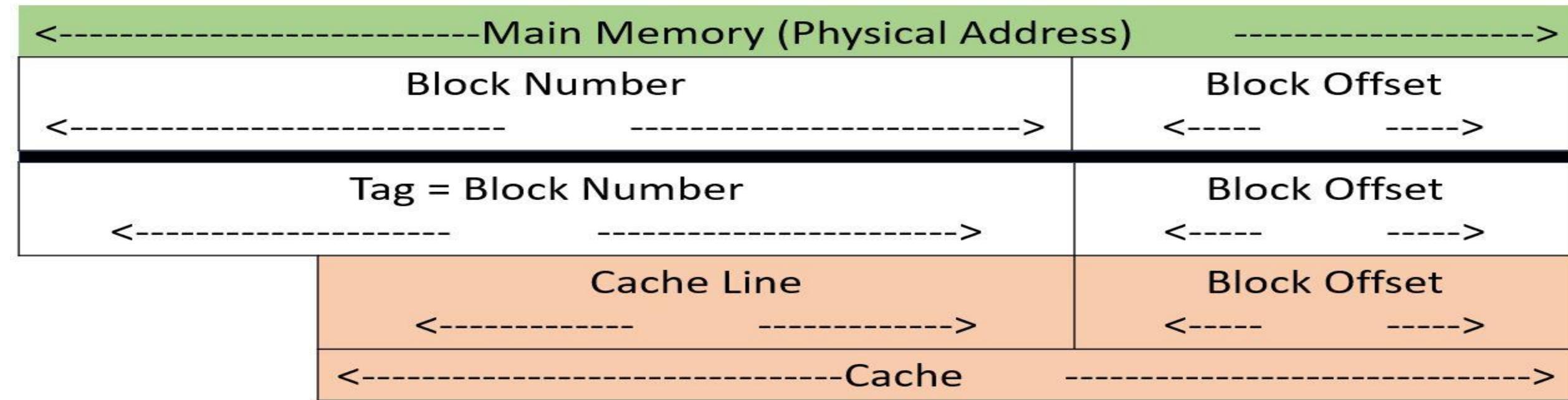
Tag directory size = Number of tags x Tag size = Number of lines in cache x Number of bits in tag

# Hardware Architecture

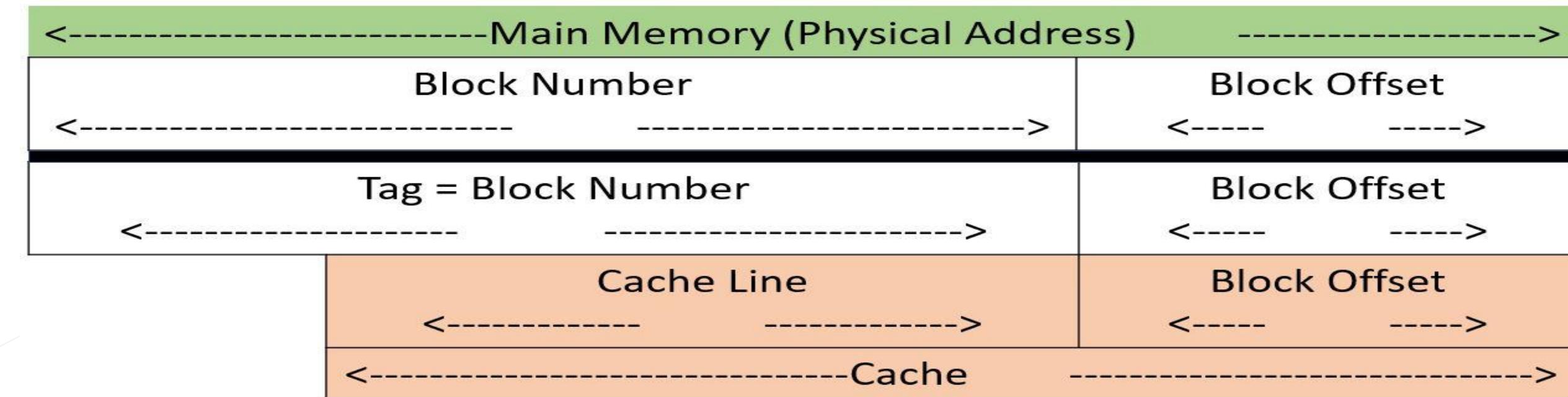
- If we have 'n' lines in cache then 'n' number of comparators are required.
- Size of comparator = Size of Tag
- If we have 'n' bit tag then we require 'n' bit comparator.



**Q** Consider a fully associative mapped cache of size 16 KB with block size 256 bytes. The size of main memory is 128 KB. Find out the: Number of bits in tag and Tag directory size?

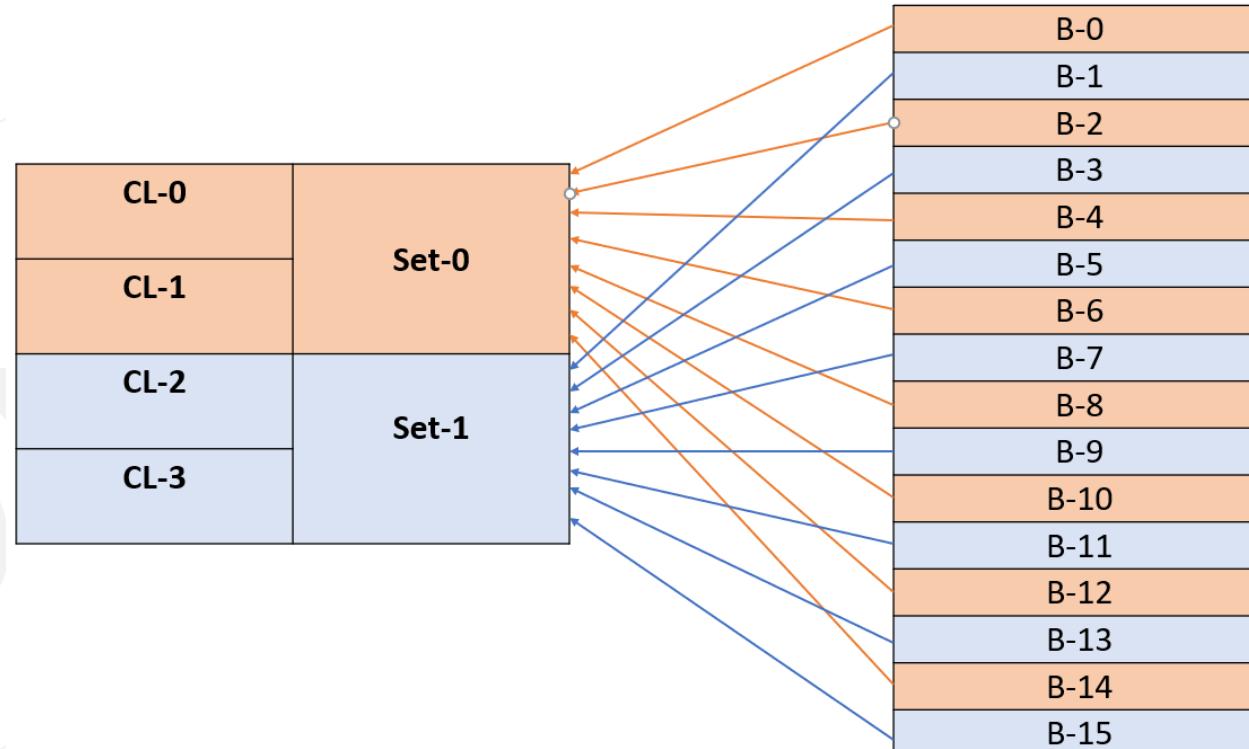


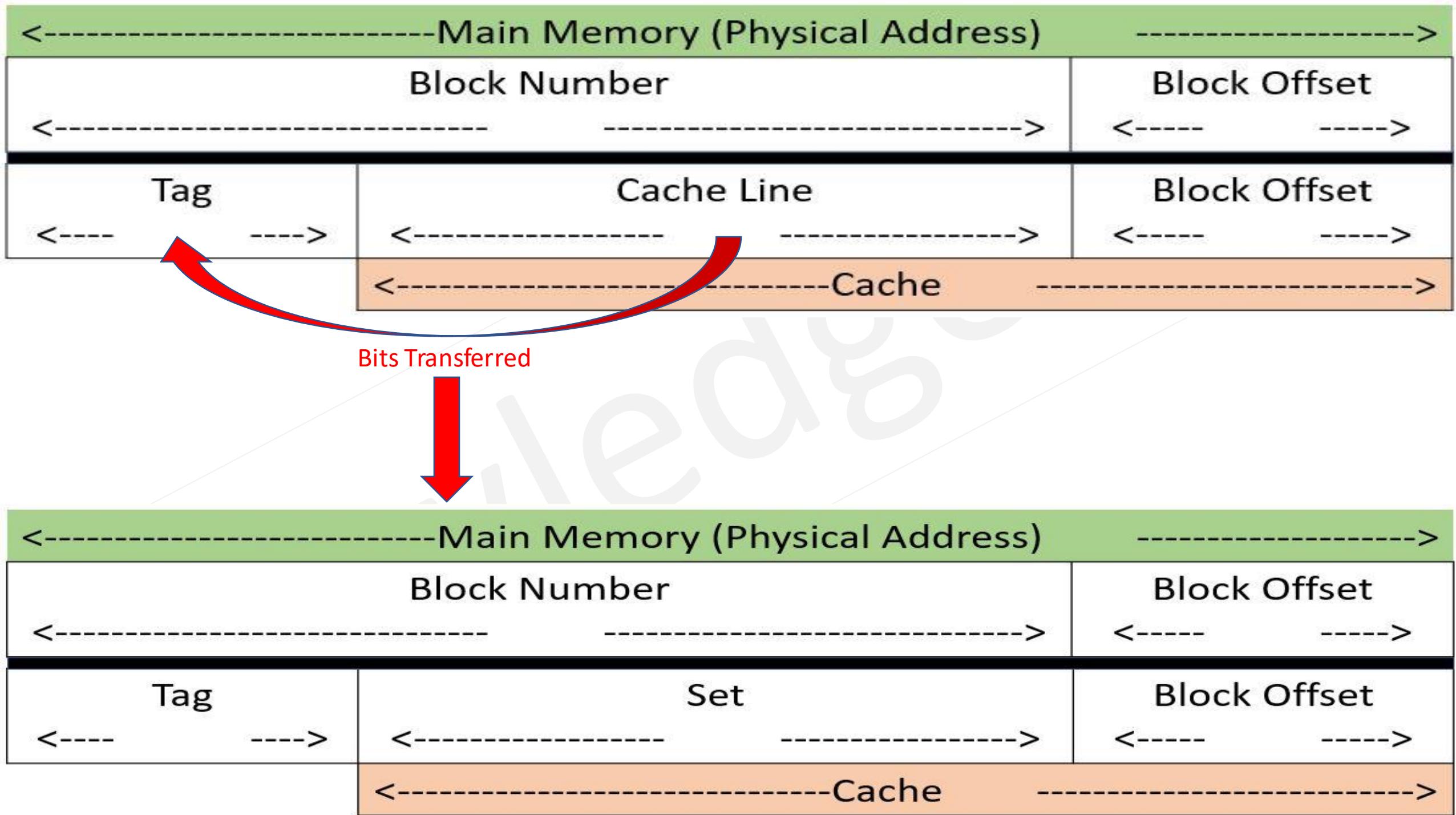
MM Size	Cache Size	Block Size	No of bits in Tag	Tag Directory Size	Comp
128 KB	16 KB	256 B			
32 GB	32 KB	1 KB			
	512 KB	1 KB	17		
16 GB		4 KB	22		
64MB			10		
	512 KB		7		



## Set Associative Mapping

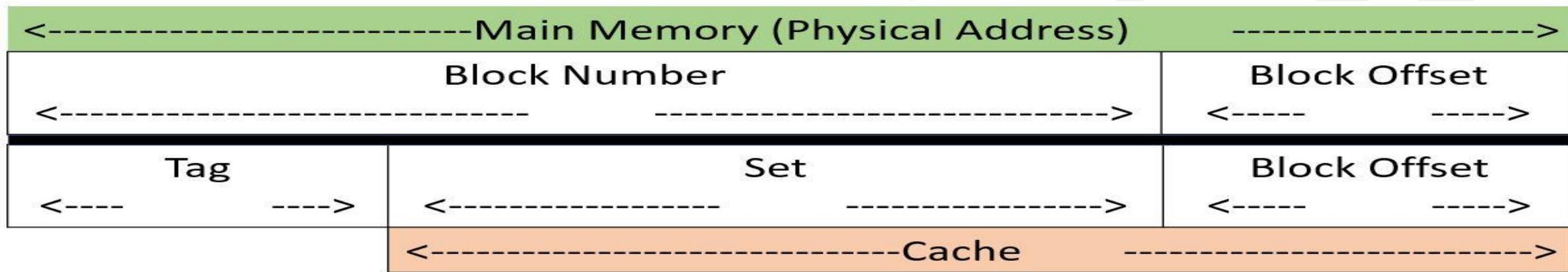
- Hardware cost is high as compared to direct mapping in associative mapping. Tag directory size is more as compared to direct mapping in associative mapping.
- In k-way set associative mapping, cache lines are grouped into sets where each set contains “k” number of lines. A particular block of main memory can map to only one particular set of the cache. However, within that set, the memory block can map to any freely available cache line.





## Formulas

- Number of Sets = No of Lines in Cache / No of Cache line in a set( $k$ )( $k$ -way set associative)



## Main Memory

B-0	W-0	W-1	W-2	W-3
B-1	W-4	W-5	W-6	W-7
B-2	W-8	W-9	W-10	W-11
B-3	W-12	W-13	W-14	W-15
B-4	W-16	W-17	W-18	W-19
B-5	W-20	W-21	W-22	W-23
B-6	W-24	W-25	W-26	W-27
B-7	W-28	W-29	W-30	W-31
B-8	W-32	W-33	W-34	W-35
B-9	W-36	W-37	W-38	W-39
B-10	W-40	W-41	W-42	W-43
B-11	W-44	W-45	W-46	W-47
B-12	W-48	W-49	W-50	W-51
B-13	W-52	W-53	W-54	W-55
B-14	W-56	W-57	W-58	W-59
B-15	W-60	W-61	W-62	W-63

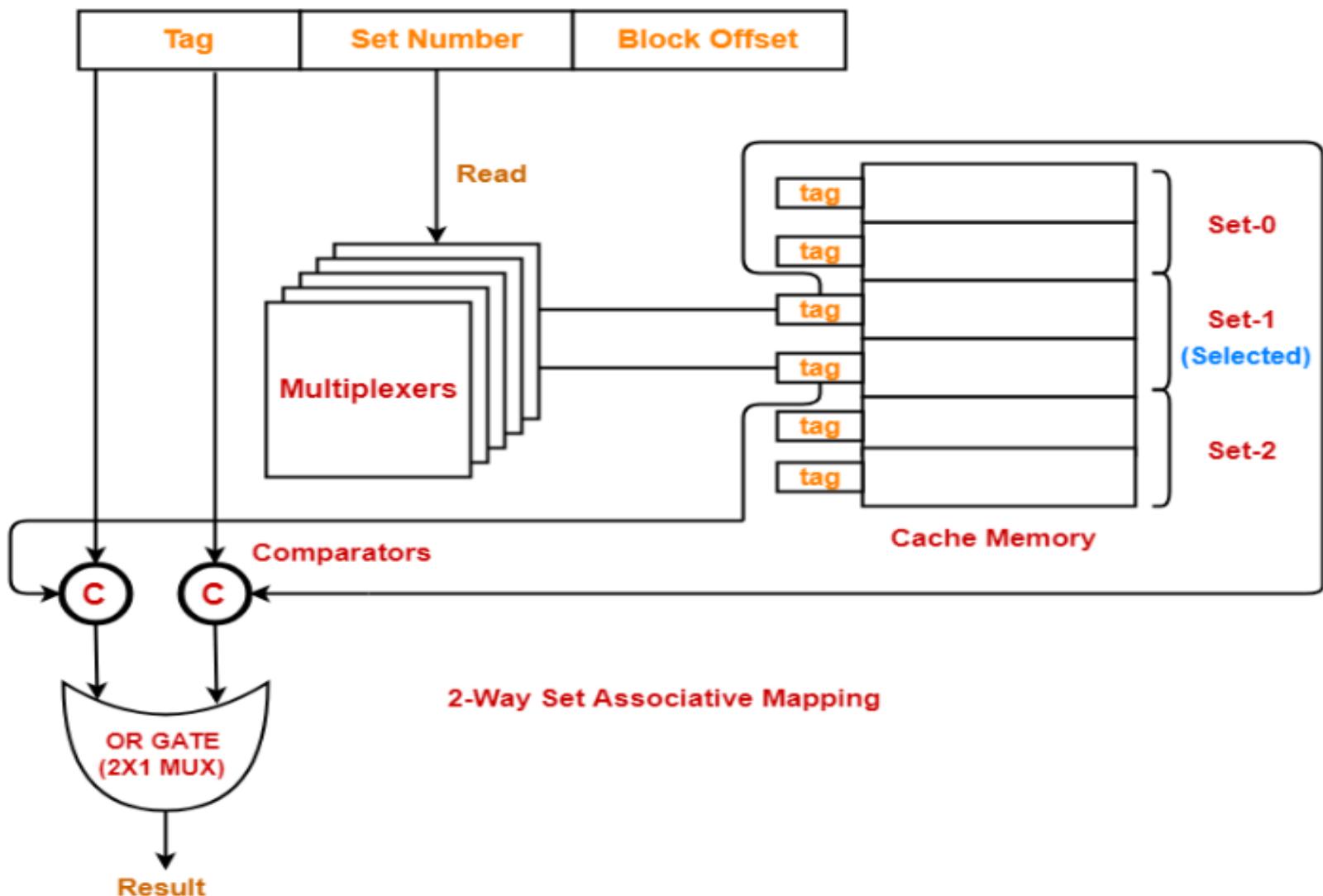
## Cache

Set Number-0	CL-0	TAG = Block no – CL	W-
			W-
			W-
			W-
Set Number-1	CL-1	TAG = Block no – CL	W-
			W
			W-
			W-
	CL-2	TAG = Block no – CL	W-
			W-
			W-
			W-
	CL-3	TAG = Block no - CL	W-
			W-
			W-
			W-

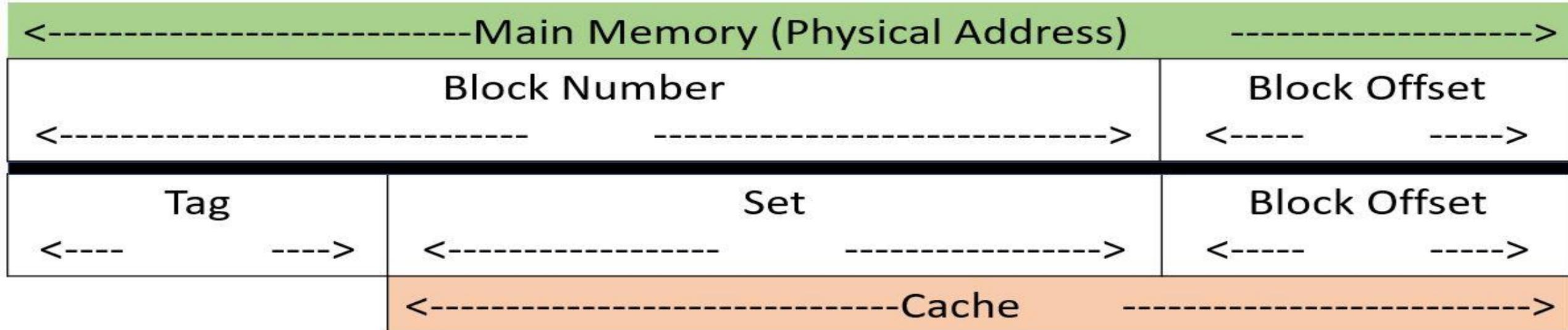
# Cache

Set Number-0	CL-0	Tag	B-0	W-0	B-2	W-8	B-4	W-16	B-6	W-24
				W-1		W-9		W-17		W-25
				W-2		W-10		W-18		W-26
				W-3		W-11		W-19		W-27
	CL-1	Tag	B-8	W-32	B-10	W-40	B-12	W-48	B-14	W-56
				W-33		W-41		W-49		W-57
				W-34		W-42		W-50		W-58
				W-35		W-43		W-51		W-59
Set Number-1	CL-2	Tag	B-1	W-4	B-3	W-12	B-5	W-20	B-7	W-28
				W-5		W-13		W-21		W-29
				W-6		W-14		W-22		W-30
				W-7		W-15		W-23		W-31
	CL-3	Tag	B-9	W-36	B-11	W-44	B-13	W-52	B-15	W-60
				W-37		W-45		W-53		W-61
				W-38		W-46		W-54		W-62
				W-39		W-47		W-55		W-63

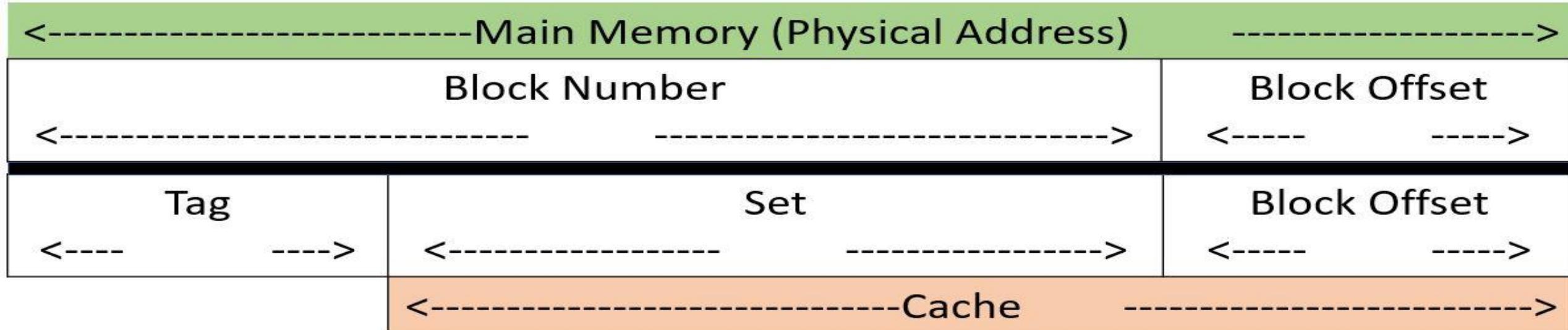
# Hardware Architecture



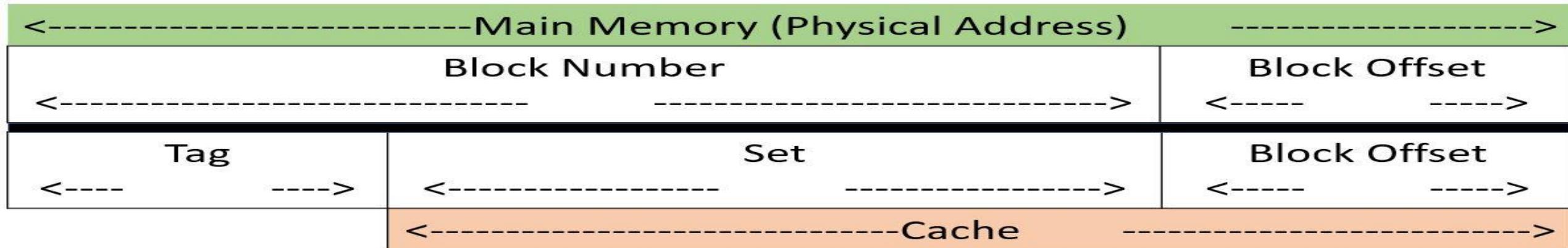
**Q** Consider the main memory size is of 128 KB, the cache size is of 16 KB, the block size is of 256 B, the set size is 2. Find Tag.



**Q Main Memory = 32 GB, Cache Size = 32 KB, Block Size = 1 KB and it is a 4-way set associative cache?**

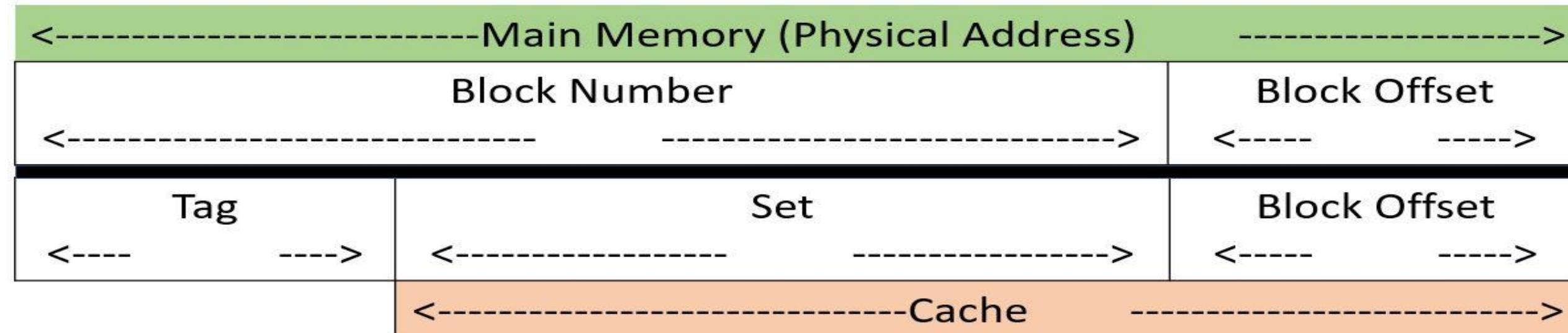


MM Size	Cache Size	Block Size	No of bits in Tag	Tag Directory Size	Set Associative
128 KB	16 KB	256 B			2-way
32 GB	32 KB	1 KB			4-way
	512 KB	1 KB	7		8-way
16 GB		4 KB	10		4-way
64MB			10		4-way
	512 KB		7		8-way



- The choice of cache mapping algorithm depends on several factors, including the size of the cache, the size of the main memory, and the type of data being stored. The most common cache mapping algorithm used in practice is set-associative mapping, as it provides a good balance between the flexibility of associative mapping and the simplicity of direct mapping. In general, the cache mapping algorithm plays an important role in determining the performance of the cache memory, as it affects the number of cache hits and misses and the speed at which data can be retrieved from the cache.

**Q** Consider a set-associative cache of size 2KB ( $1\text{KB}=2^{10}$  bytes) with cache block size of 64 bytes. Assume that the cache is byte-addressable and a 32-bit address is used for accessing the cache. If the width of the tag field is 22 bits, the associativity of the cache is \_\_\_\_\_. (GATE 2021) (1 MARKS)



**Q** A computer system with a word length of 32 bits has a 16 MB byte-addressable main memory and a 64 KB, 4-way set associative cache memory with a block size of 256 bytes. Consider the following four physical addresses represented in hexadecimal notation. (Gate-2020) (2 Marks)

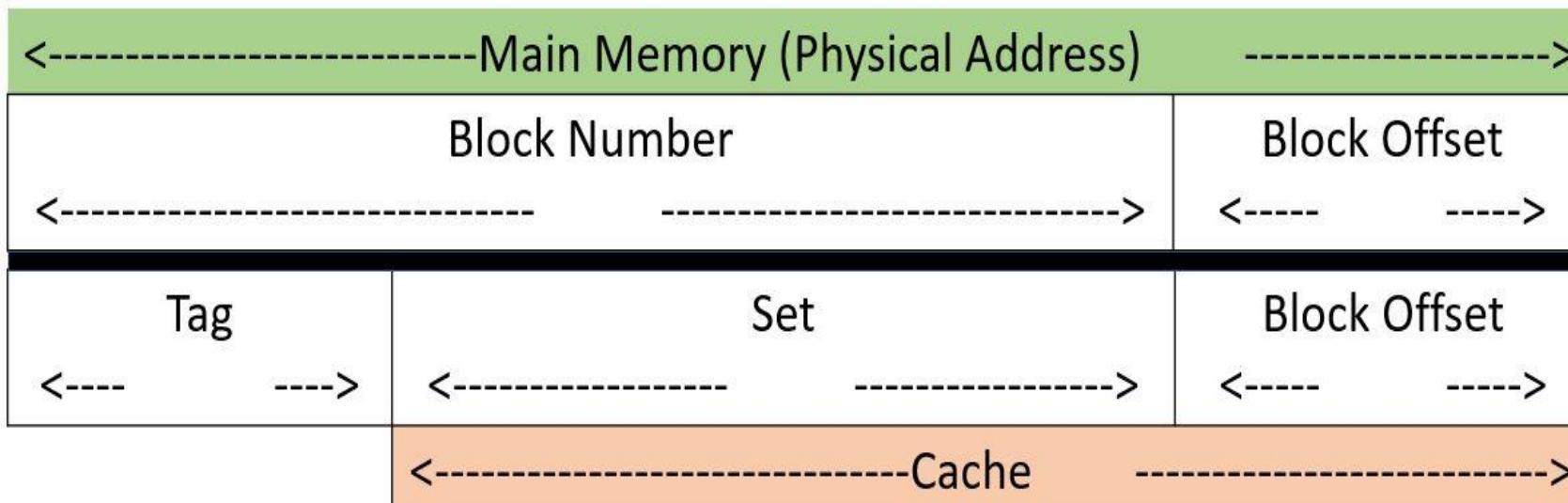
$$A_1 = 0x42C8A4,$$

$$A_2 = 0x546888,$$

$$A_3 = 0x6A289C,$$

$$A_4 = 0x5E4880$$

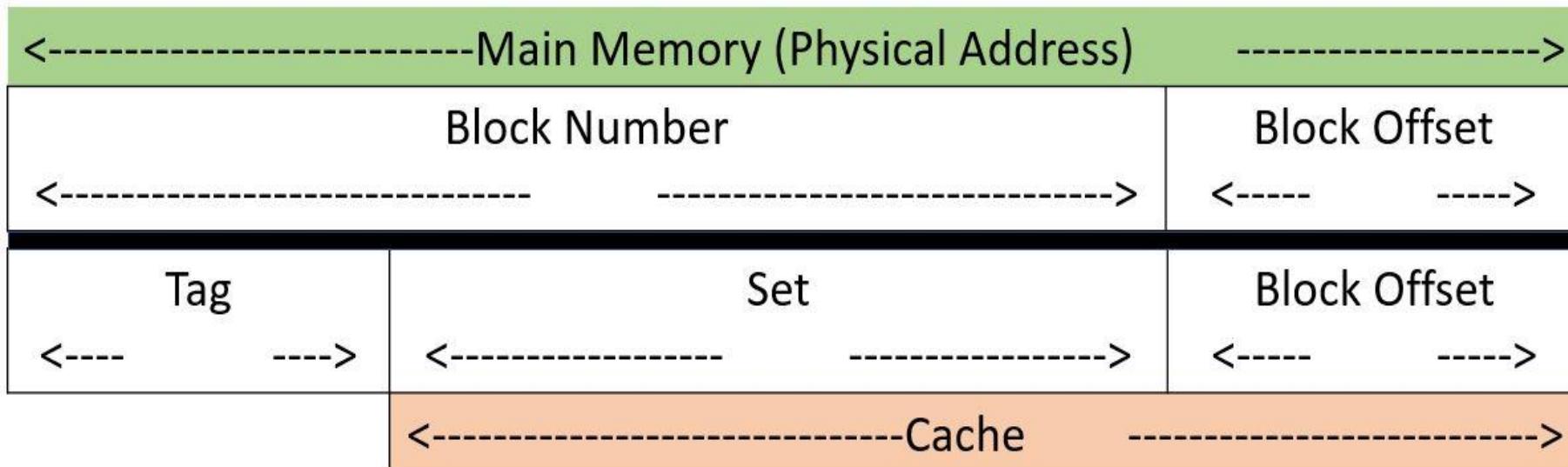
- (A)  $A_1$  and  $A_4$  are mapped to different cache sets.
- (B)  $A_2$  and  $A_3$  are mapped to the same cache set.
- (C)  $A_3$  and  $A_4$  are mapped to the same cache set.
- (D)  $A_1$  and  $A_3$  are mapped to the same cache set.



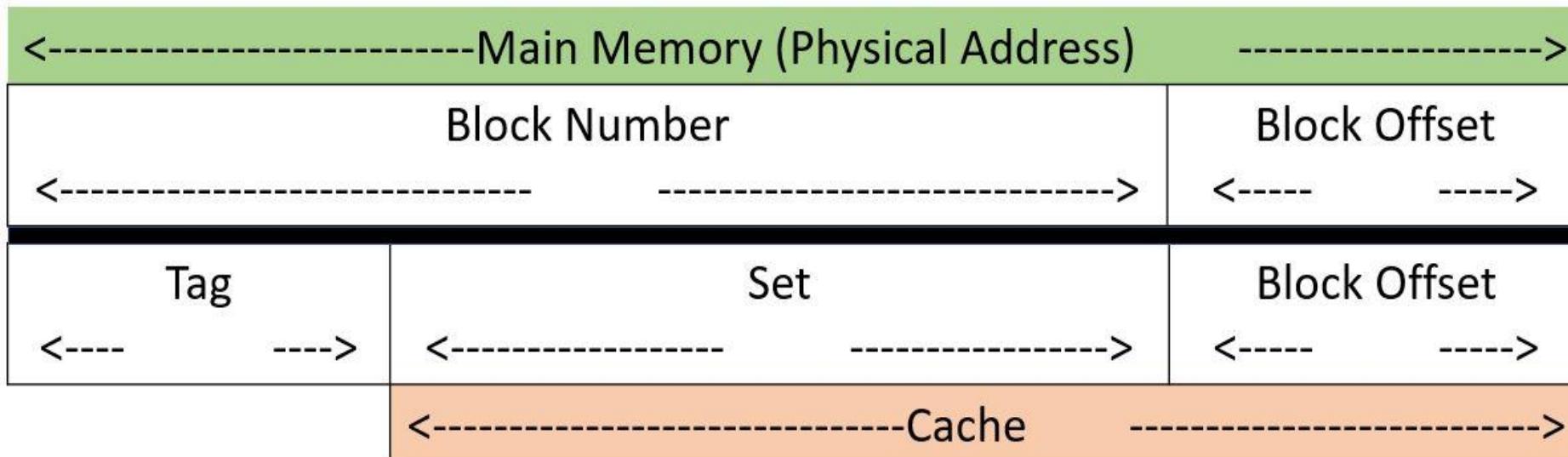
**Q** The size of the physical address space of a processor is  $2^P$  bytes. The word length is  $2^W$  bytes. The capacity of cache memory is  $2^N$  bytes. The size of each cache block is  $2^M$  words. For a K-way set-associative cache memory, the length (in number of bits) of the tag field is **(Gate-2018) (2 Marks)**

- (A)  $P - N - \log_2 K$   
(C)  $P - N - M - W - \log_2 K$

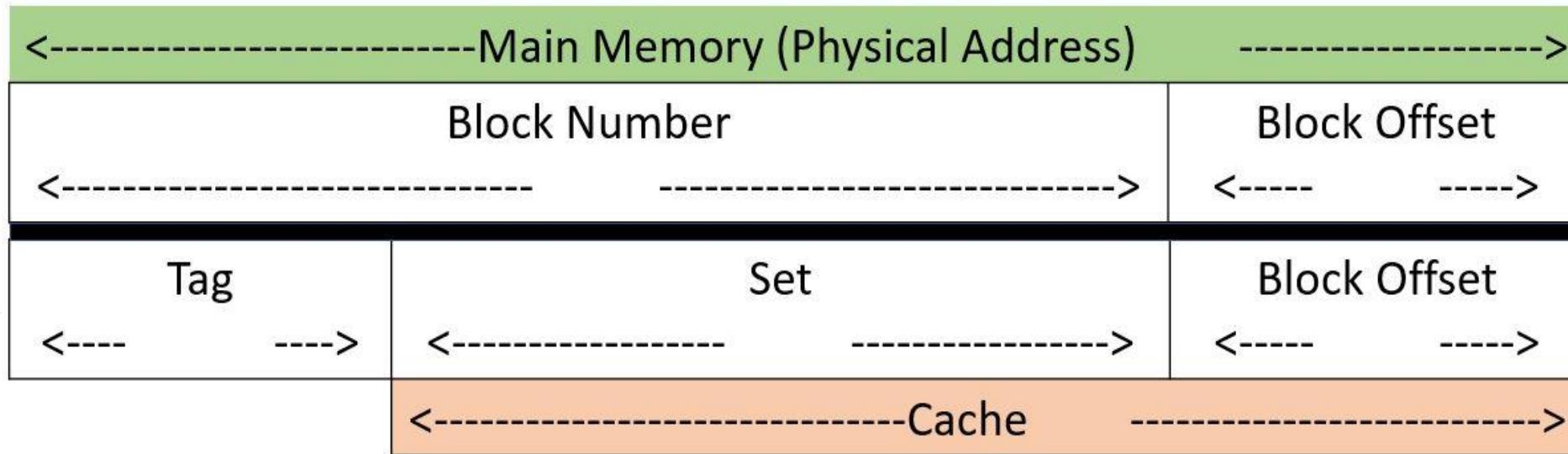
- (B)  $P - N + \log_2 K$   
(D)  $P - N - M - W + \log_2 K$



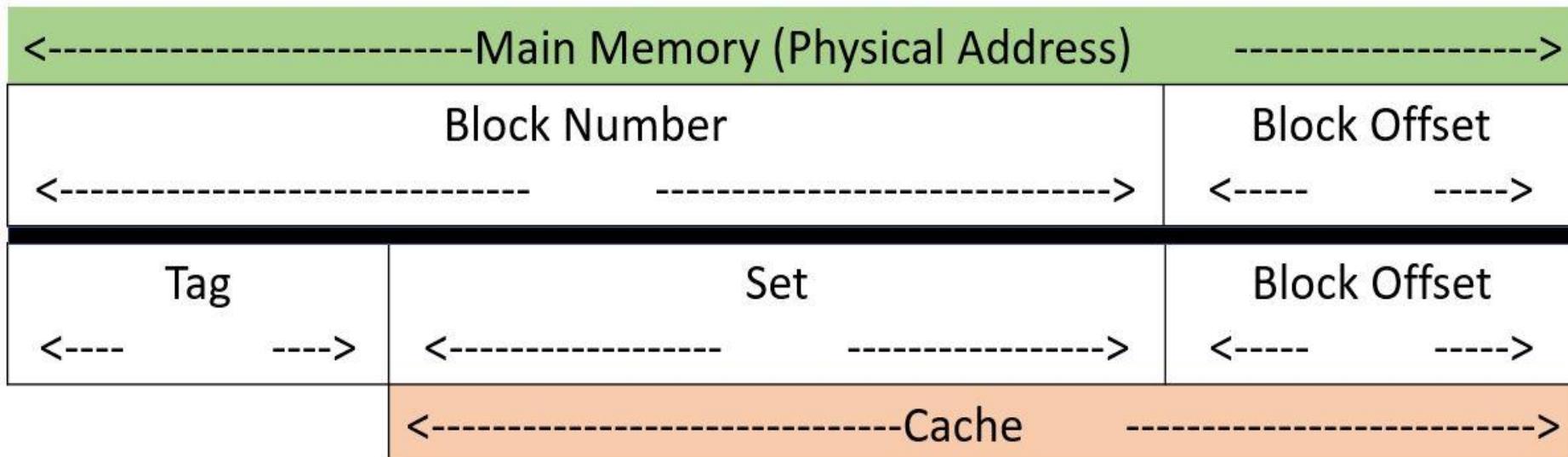
**Q** A cache memory unit with capacity of  $N$  words and block size of  $B$  words is to be designed. If it is designed as direct mapped cache, the length of the TAG field is 10 bits. If the cache unit is now designed as a 16-way set-associative cache, the length of the TAG field is \_\_\_\_\_ bits. (Gate-2017) (1 Marks)



**Q** The width of the physical address on a machine is 40 bits. The width of the tag field in a 512 KB 8-way set associative cache is \_\_\_\_\_ bits (Gate-2016) (2 Marks)



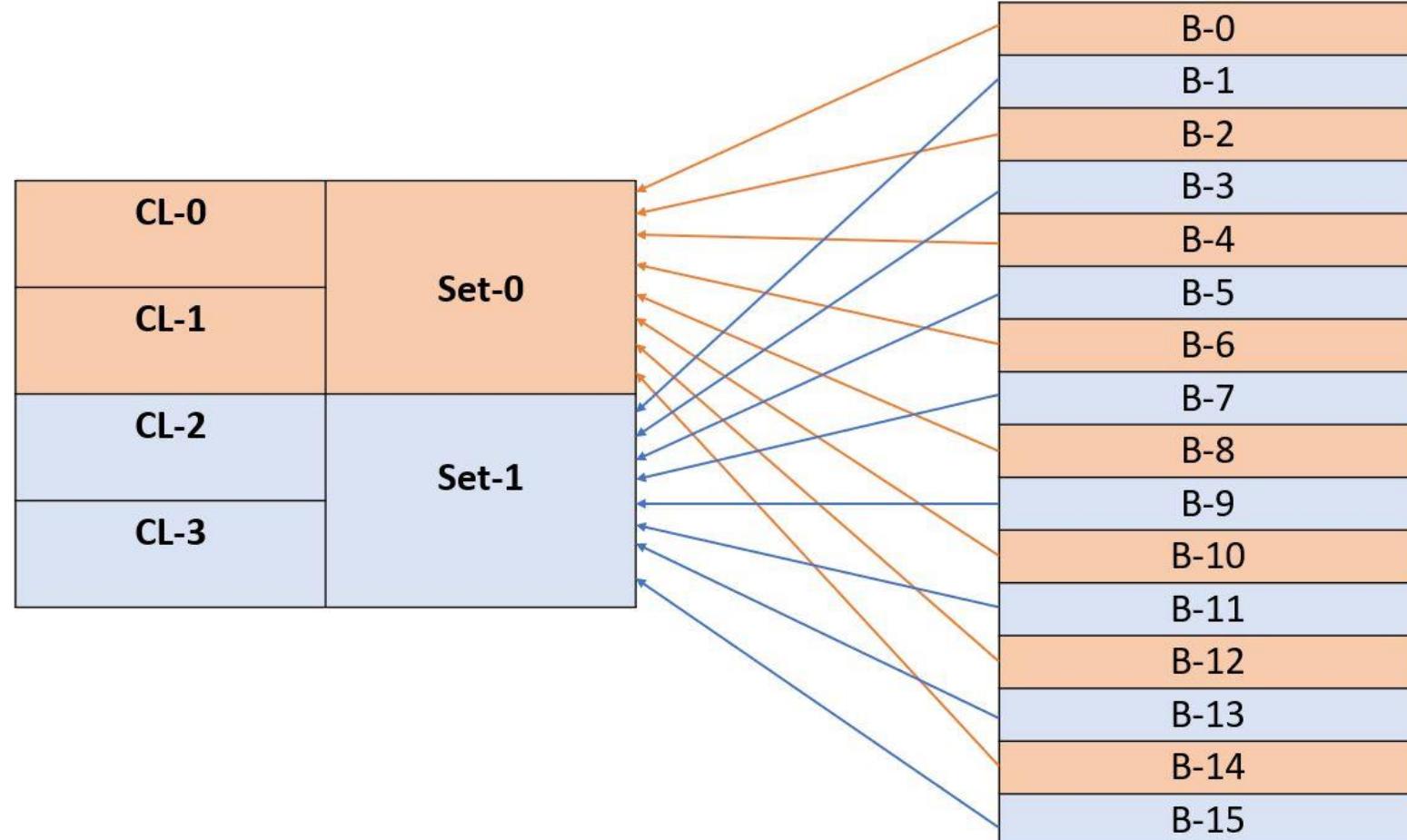
**Q** A 4-way set-associative cache memory unit with a capacity of 16 KB is built using a block size of 8 words. The word length is 32 bits. The size of the physical address space is 4 GB. The number of bits for the TAG field is \_\_\_\_\_ (Gate-2014) (1 Marks)



**Q** In a k-way set associative cache, the cache is divided into v sets, each of which consists of k lines. The lines of a set are placed in sequence one after another. The lines in set s are sequenced before the lines in set  $(s+1)$ . The main memory blocks are numbered 0 onwards. The main memory block numbered j must be mapped to any one of the cache lines from. **(Gate-2013) (1 Marks)**

- (A)**  $(j \bmod v) * k$  to  $(j \bmod v) * k + (k-1)$   
**(C)**  $(j \bmod k)$  to  $(j \bmod k) + (v-1)$

- (B)**  $(j \bmod v)$  to  $(j \bmod v) + (k-1)$   
**(D)**  $(j \bmod k) * v$  to  $(j \bmod k) * v + (v-1)$



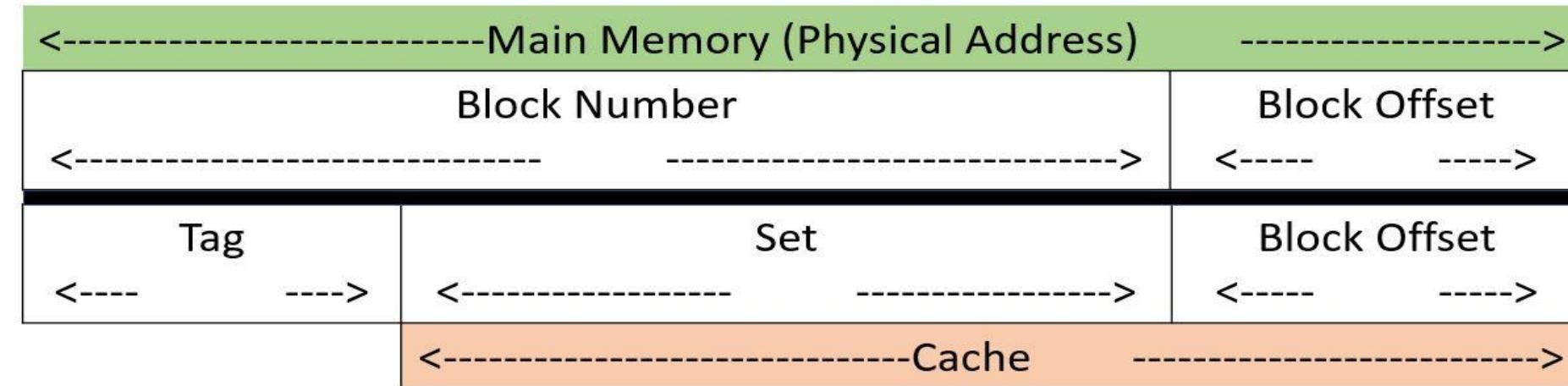
**Q** A computer has a 256 KByte, 4-way set associative, write back data cache with block size of 32 Bytes. The processor sends 32-bit addresses to the cache controller. Each cache tag directory entry contains, in addition to address tag, 2 valid bits, 1 modified bit and 1 replacement bit. The number of bits in the tag field of an address is **(Gate-2012) (2 Marks)**

(A) 11

(B) 14

(C) 16

(D) 27



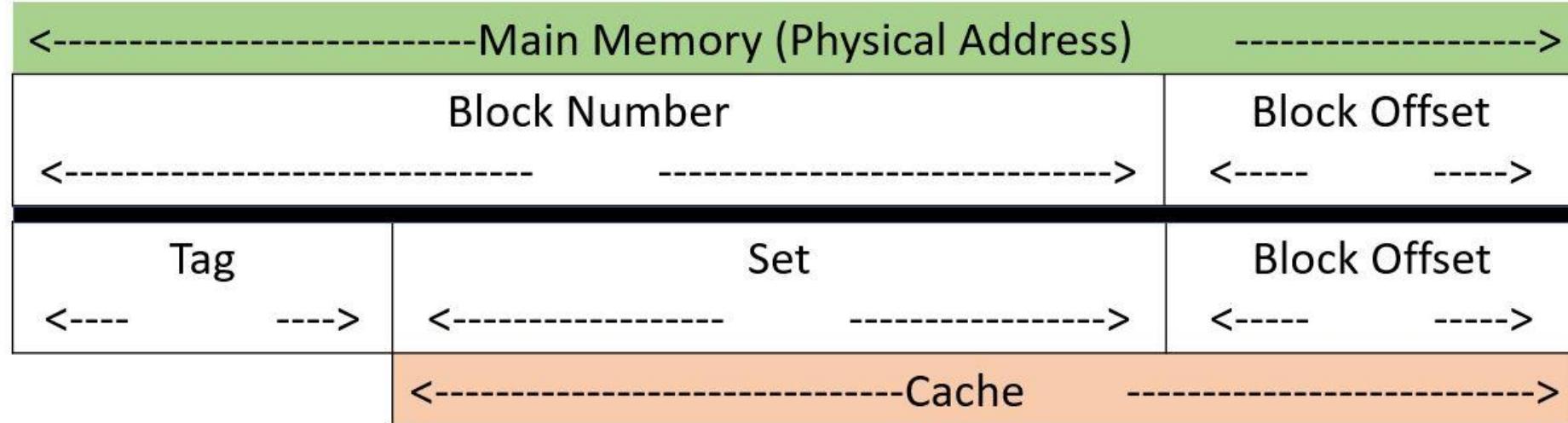
**Q The size of the cache tag directory is (Gate-2012) (2 Marks)**

a) 160 Kbits

**b) 136 Kbits**

c) 40 Kbits

d) 32 Kbits



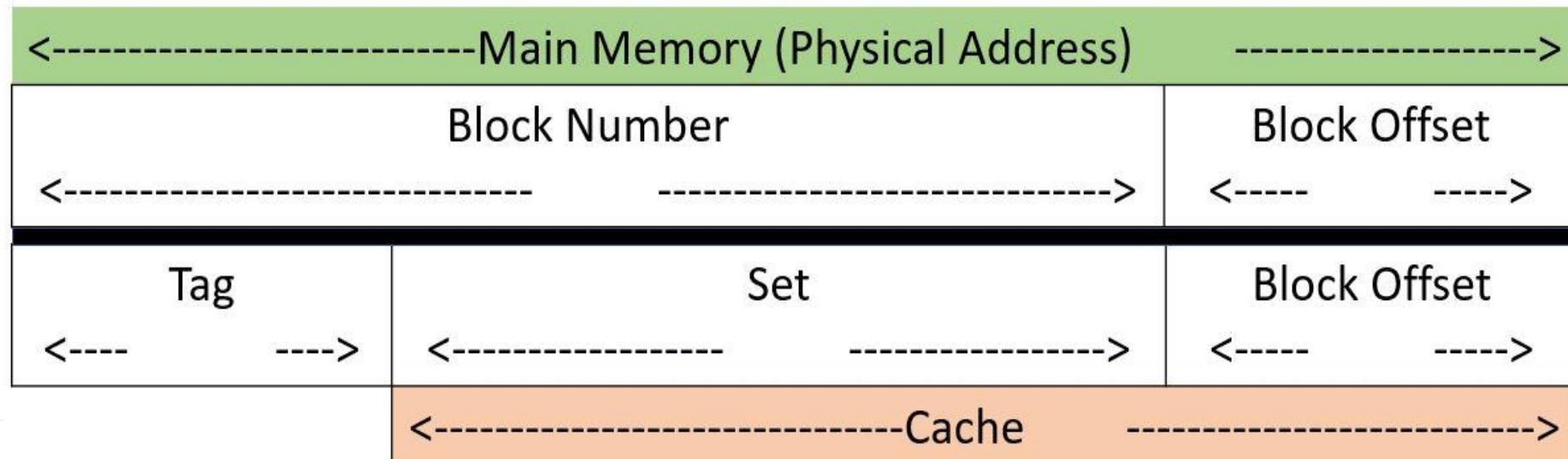
**Q** Consider a computer with a 4-way set-associative mapped cache of the following characteristics: a total of 1 MB of main memory, a word size of 1 byte, a block size of 128 words and a cache size of 8 KB. The number of bits in the TAG, SET and WORD fields, respectively are: **(Gate-2008) (2 Marks)**

**(A) 7, 6, 7**

**(B) 8, 5, 7**

**(C) 8, 6, 6**

**(D) 9, 4, 7**



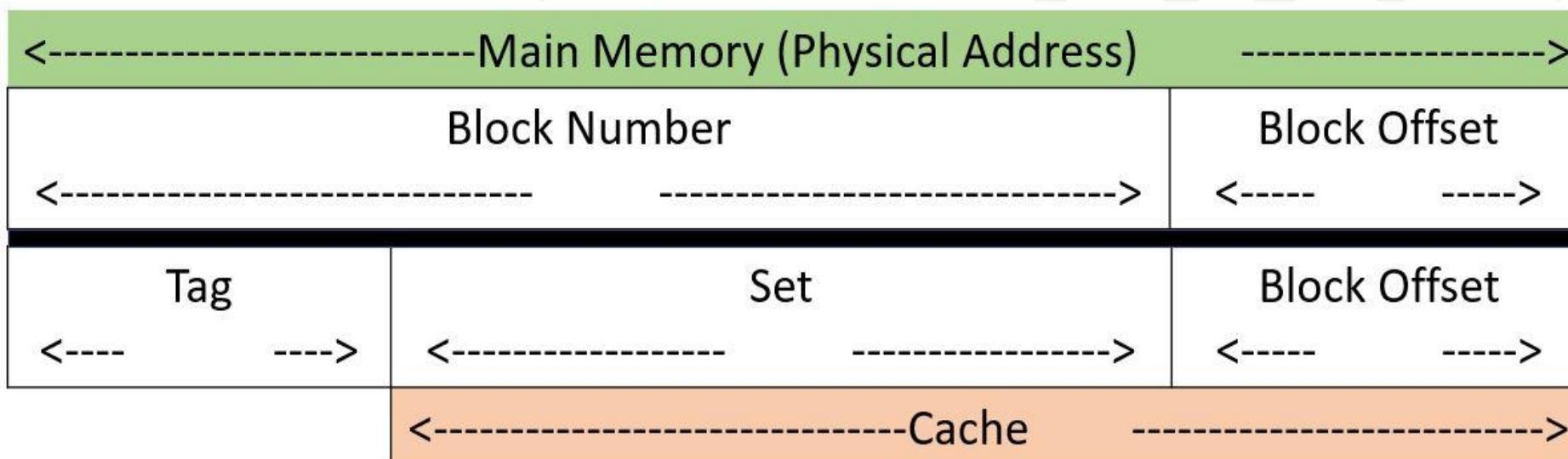
**Q** Consider a computer with a 4-way set-associative mapped cache of the following characteristics: a total of 1 MB of main memory, a word size of 1 byte, a block size of 128 words and a cache size of 8 KB. While accessing the memory location 0C795H by the CPU, the contents of the TAG field of the corresponding cache line is: **(Gate-2008)** **(2 Marks)**

a) 000011000

b) 110001111

c) 00011000

d) 110010101



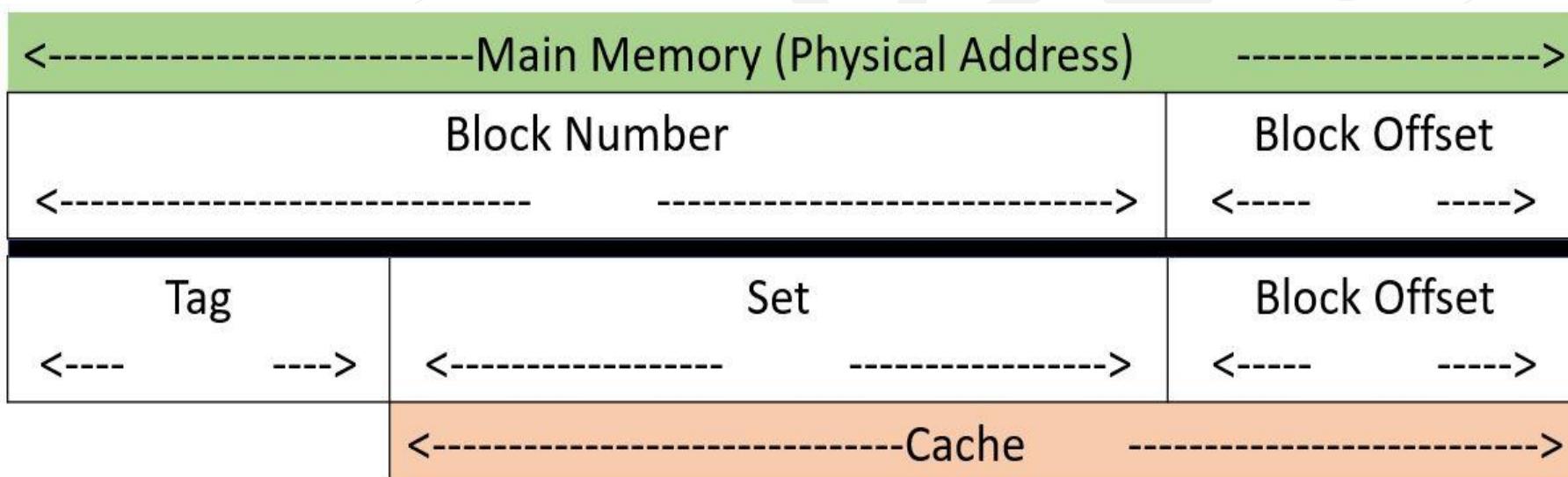
**Q** Consider a 4-way set associative cache consisting of 128 lines with a line size of 64 words. The CPU generates a 20-bit address of a word in main memory. The number of bits in the TAG, LINE and WORD fields are respectively: **(Gate-2007) (1 Marks)**

**(A) 9,6,5**

**(B) 7, 7, 6**

**(C) 7, 5, 8**

**(D) 9, 5, 6**



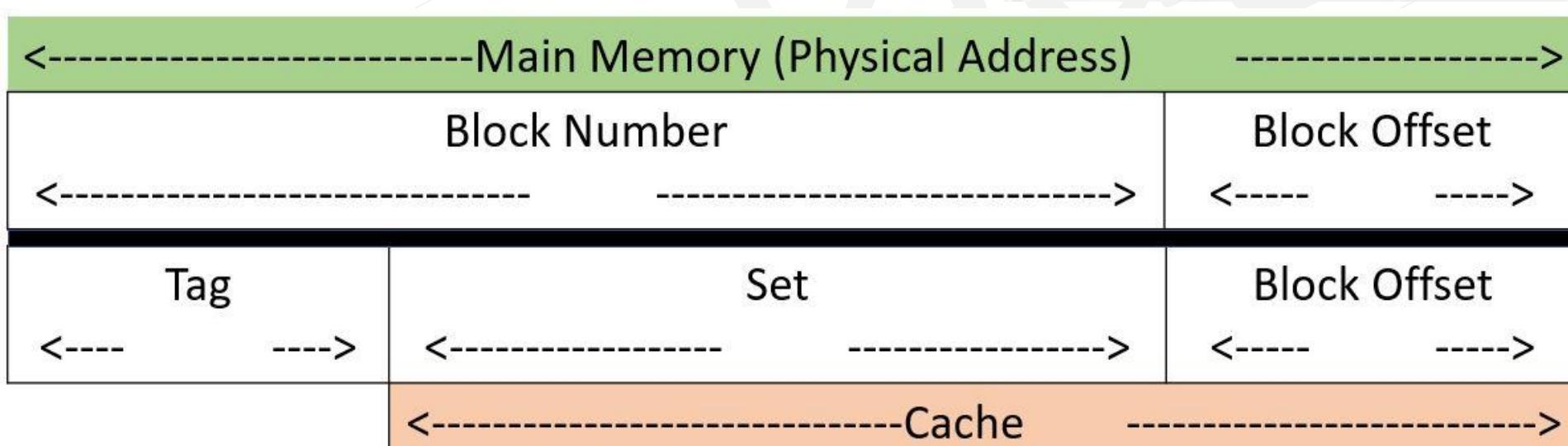
**Q** A computer system has a level-1 instruction cache (I-cache), a level-1 data cache (D-cache) and a level-2 cache (L2-cache) with the following specifications:

The length of the physical address of a word in the main memory is 30 bits. The capacity of the tag memory in the I-cache, D-cache and L2-cache is, respectively,

**(Gate-2006) (2 Marks)**

- (A) 1 K x 18-bit, 1 K x 19-bit, 4 K x 16-bit
- (B) 1 K x 16-bit, 1 K x 19-bit, 4 K x 18-bit
- (C) 1 K x 16-bit, 512 x 18-bit, 1 K x 16-bit
- (D) 1 K x 18-bit, 512 x 18-bit, 1 K x 18-bit

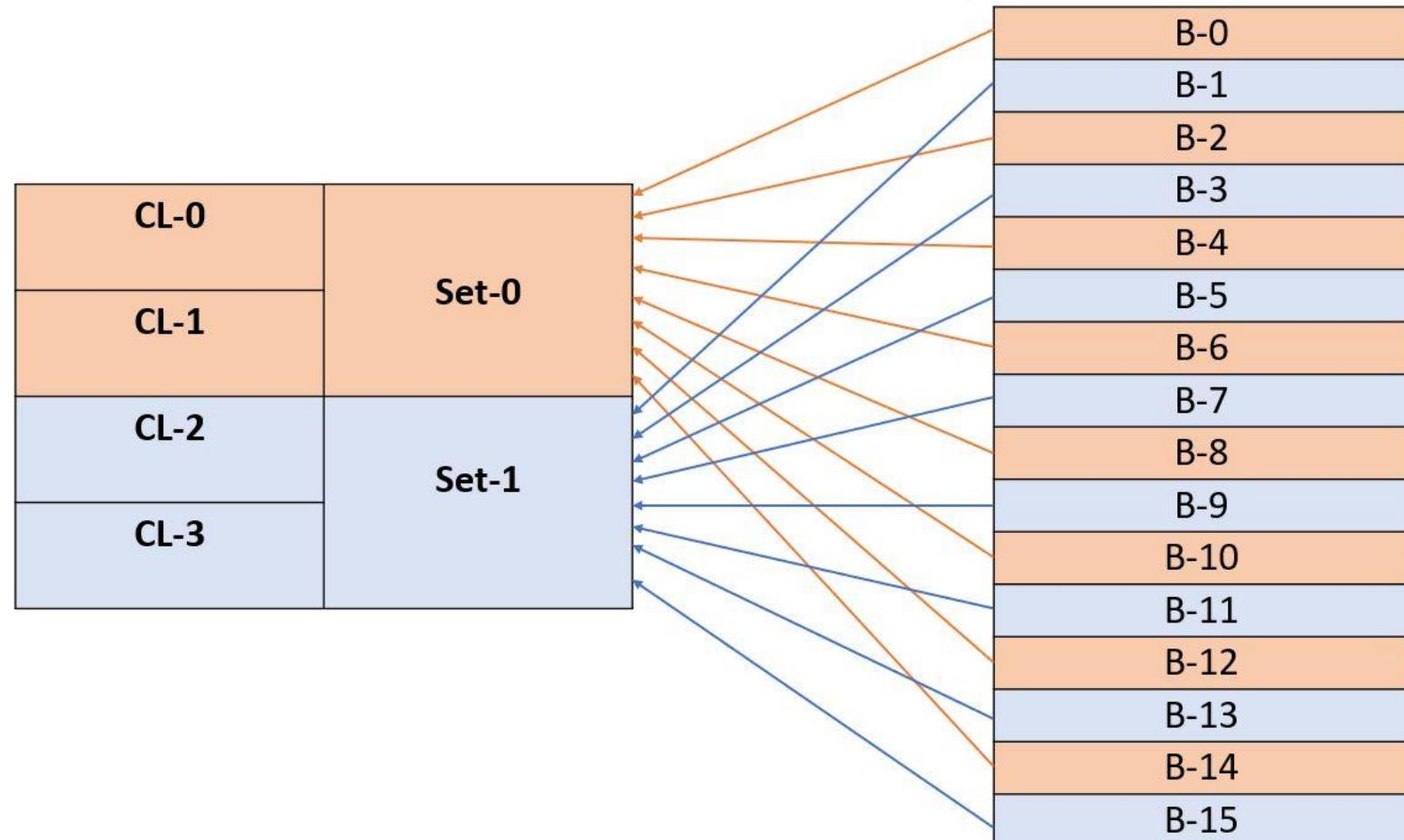
	Capacity	Mapping Method	Block size
I-cache	4K words	Direct mapping	4 Words
D-cache	4K words	2-way set associative mapping	4 Words
L2-cache	64K words	4-way set associative mapping	16 Words



**Q** The main memory of a computer has  $2^m$  blocks while the cache has  $2^c$  blocks. If the cache uses the set associative mapping scheme with  $2^s$  blocks per set, then the block  $k$  of main memory maps to the set: **(Gate-1999) (1 Marks)**

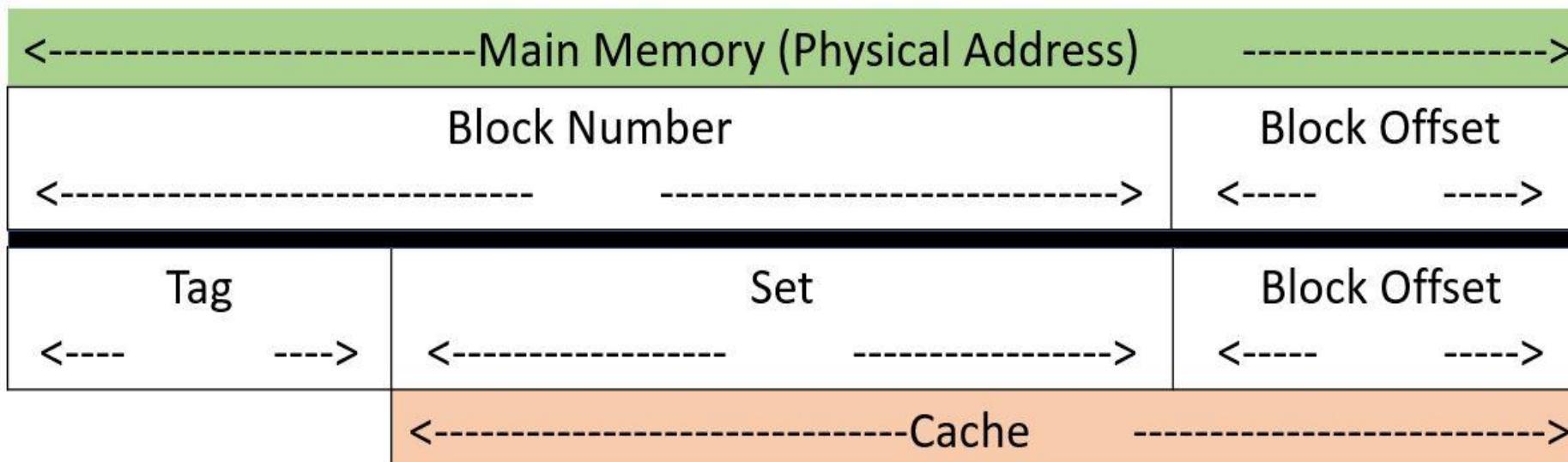
- (A)  $(k \bmod m)$  of the cache  
(C)  $(k \bmod 2^c)$  of the cache

- (B)  $(k \bmod c)$  of the cache  
(D)  $(k \bmod 2^{cm})$  of the cache



**Q** A block-set associative cache memory consists of 128 blocks divided into four block sets. The main memory consists of 16,384 blocks and each block contains 256 eight-bit words. **(Gate-1990) (2 Marks)**

- i)** How many bits are required for addressing the main memory?
- ii)** How many bits are needed to represent the TAG, SET and WORD fields?



- Q** In designing a computer's cache system, the cache block (or cache line) size is an important parameter. Which one of the following statements is correct in this context? **(Gate-2014) (1 Marks)**
- (A)** A smaller block size implies better spatial locality
  - (B)** A smaller block size implies a smaller cache tag and hence lower cache tag overhead
  - (C)** A smaller block size implies a larger cache tag and hence lower cache hit time
  - (D)** A smaller block size incurs a lower cache miss penalty

## Cache Replacement Policies

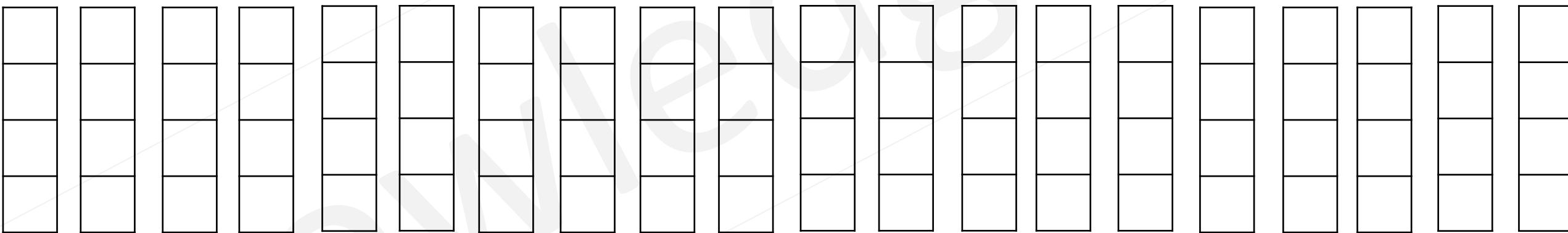
- In direct mapped cache, the position of each block is predetermined hence no replacement policy exists. In fully associative and set associative caches there exists policies. When a new block is brought into the cache and all the positions that it may occupy are full, then the controller needs to decide which of the old blocks it can overwrite.

## FIFO Policy

- The block which have entered first in the memory will be replaced first.
- This can lead to a problem known as “**Belady's Anomaly**”, it states that if we increase the number of lines in cache memory the cache miss will increase.

**Example:** Let the blocks be in the sequence: 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1 and the cache memory has 4 lines.

7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1

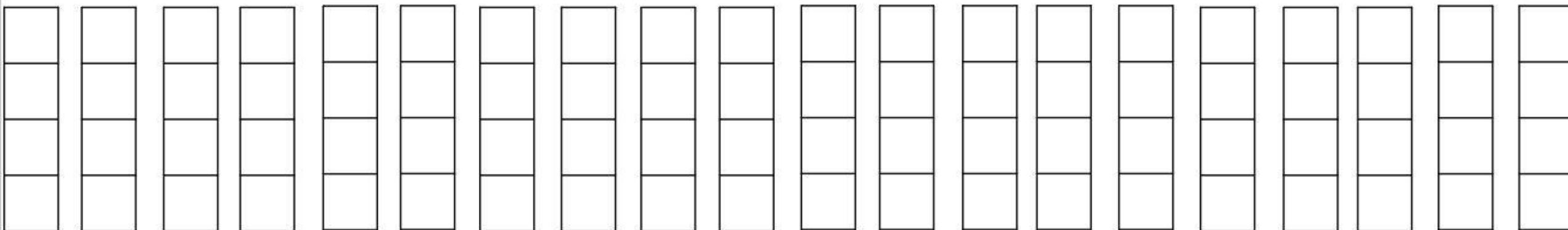


## Optimal Algorithm

- The page which will not be used for the longest period of time in future references will be replaced first.
- The optimal algorithm will provide the best performance but it is difficult to implement as it requires the future knowledge of pages which is not possible.
- It is used as a benchmark for cache replacement algorithms.

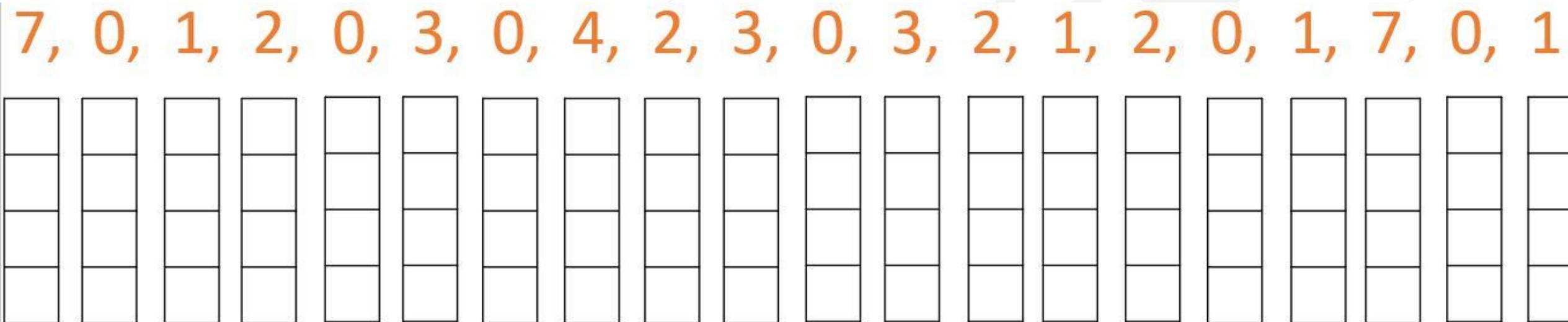
**Example:** Let the blocks be in the sequence: 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1 and the cache memory has 4 lines.

7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1



## LRU (Least Recently Used)

- The page which was not used for the longest period of time in the past will get replaced first.
- Example: Let the blocks be in the sequence: 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1 and the cache memory has 4 lines.

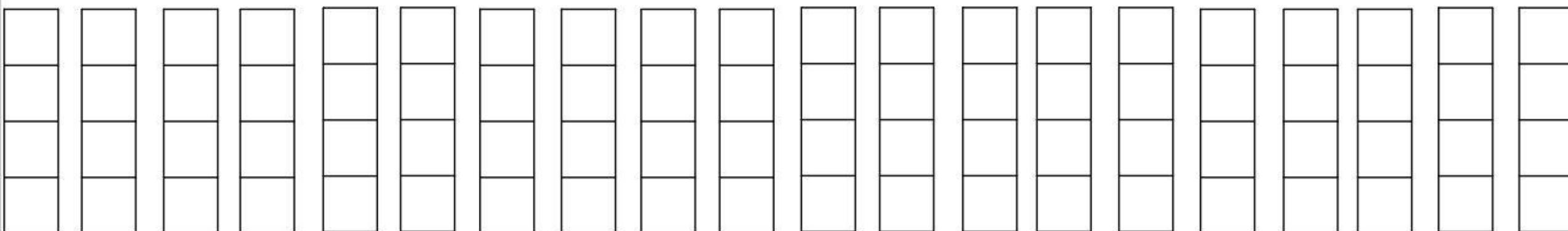


## Most Recently Used (MRU)

- The page which was used recently will be replaced first.

Example: Let the blocks be in the sequence: 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1 and the cache memory has 4 lines.

7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1



## Types of Miss

- **Compulsory Miss**
  - When CPU demands for any block for the first time then definitely a miss is going to occur as the block needs to be brought into the cache, it is known as Compulsory miss.
- **Capacity Miss**
  - Occur because blocks are being discarded from cache because cache cannot contain all blocks needed for program execution.
- **Conflict Miss**
  - In the case of set associative or direct mapped block placement strategies, conflict misses occur when several blocks are mapped to the same set or block frame; also called collision misses or interference misses.

**Q** Consider a 2-way set associative cache with 256 blocks and uses LRU replacement. Initially the cache is empty. Conflict misses are those misses which occur due to the contention of multiple blocks for the same cache set. Compulsory misses occur due to first time access to the block. The following sequence of accesses to memory blocks (0, 128, 256, 128, 0, 128, 256, 128, 1, 129, 257, 129, 1, 129, 257, 129) is repeated 10 times. The number of conflict misses experienced by the cache is \_\_\_\_\_. **(Gate-2017) (2 Marks)**

**Q** Consider a 4-way set associative cache (initially empty) with total 16 cache blocks. The main memory consists of 256 blocks and the request for memory blocks is in the following order: 0, 255, 1, 4, 3, 8, 133, 159, 216, 129, 63, 8, 48, 32, 73, 92, 155. Which one of the following memory blocks will NOT be in cache if LRU replacement policy is used? **(Gate-2009) (2 Marks)**

- (A) 3
- (B) 8
- (C) 129
- (D) 216

**Q** Consider a Direct Mapped Cache with 8 cache blocks (numbered 0-7). If the memory block requests are in the following order 3, 5, 2, 8, 0, 63, 9, 16, 20, 17, 25, 18, 30, 24, 2, 63, 5, 82, 17, 24. Which of the following memory blocks will not be in the cache at the end of the sequence? **(GATE-2007) (2 Marks)**

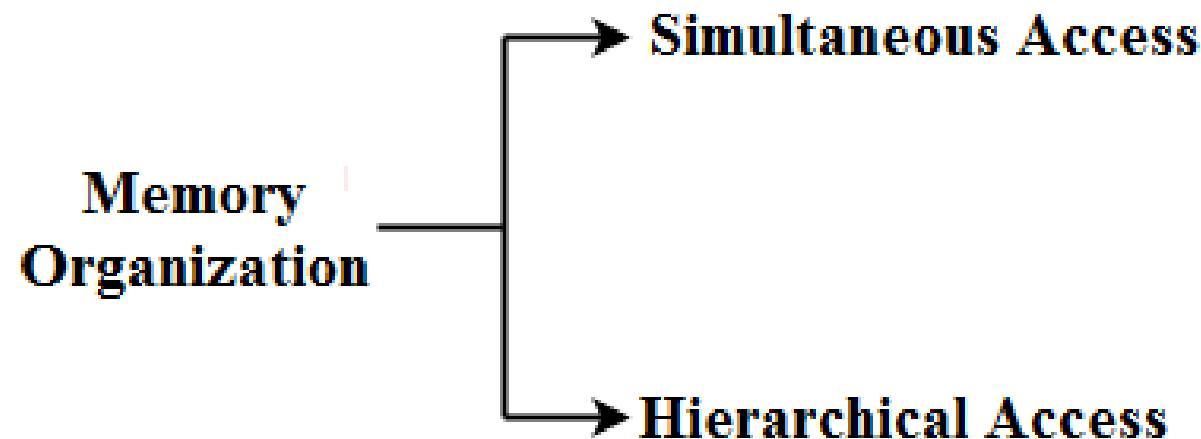


3, 5, 2, 8, 0, 63, 9, 16, 20, 17, 25, 18, 30, 24, 2, 63, 5, 82, 17, 24

**Q** Consider a small two-way set-associative cache memory, consisting of four blocks. For choosing the block to be replaced, use the least recently used (LRU) scheme. The number of cache misses for the following sequence of block addresses is 8, 12, 0, 12, 8 (**Gate-2004**) (**2 Marks**)

## Memory Organization

- Memory is organized at different levels. CPU may try to access different levels of memory in different ways. On this basis, the memory organization is broadly divided into two types

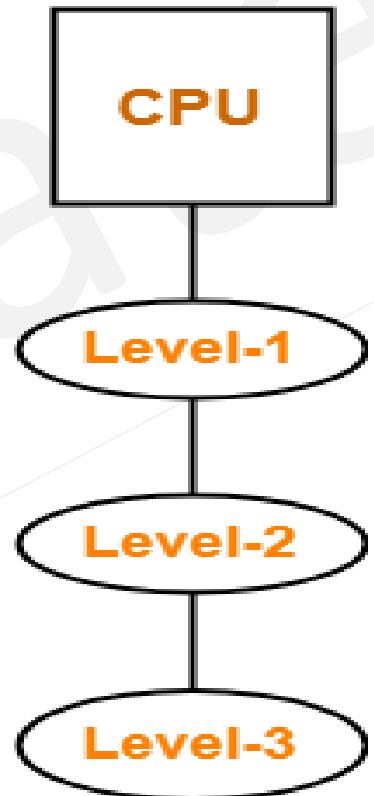


## Hierarchical Access Memory Organization

- In this memory organization, memory levels are organized as
  - Level-1 is directly connected to the CPU.
  - Level-2 is directly connected to level-1.
  - Level-3 is directly connected to level-2 and so on
- Whenever CPU requires any word,
  - It first searches for the word in level-1.
  - If the required word is not found in level-1, it searches for the word in level-2.
  - If the required word is not found in level-2, it searches for the word in level-3 and so on.



- |  |  |  |
|--|--|--|
| <ul style="list-style-type: none"> <li>• <math>T_1</math> = Access time of level <math>L_1</math></li> <li>• <math>S_1</math> = Size of level <math>L_1</math></li> <li>• <math>C_1</math> = Cost per byte of level <math>L_1</math></li> <li>• <math>H_1</math> = Hit rate of level <math>L_1</math></li> </ul> | <ul style="list-style-type: none"> <li>• <math>T_2</math> = Access time of level <math>L_2</math></li> <li>• <math>S_2</math> = Size of level <math>L_2</math></li> <li>• <math>C_2</math> = Cost per byte of level <math>L_2</math></li> <li>• <math>H_2</math> = Hit rate of level <math>L_2</math></li> </ul> | <ul style="list-style-type: none"> <li>• <math>T_3</math> = Access time of level <math>L_3</math></li> <li>• <math>S_3</math> = Size of level <math>L_3</math></li> <li>• <math>C_3</math> = Cost per byte of level <math>L_3</math></li> <li>• <math>H_3</math> = Hit rate of level <math>L_3</math></li> </ul> |
|--|--|--|



### Effective Memory Access Time

Average time required to access memory per operation =

$$H_1 * T_1 + (1 - H_1) * H_2 * (T_1 + T_2) + (1 - H_1) (1 - H_2) * H_3 * (T_1 + T_2 + T_3)$$

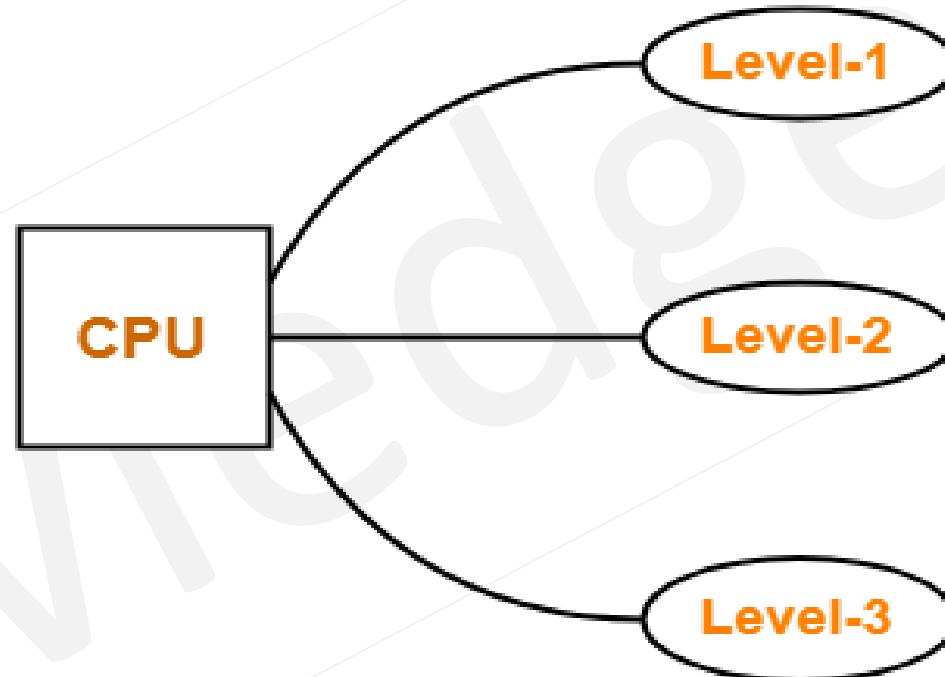
$$H_1 * T_1 + (1 - H_1) * H_2 * (T_1 + T_2) + (1 - H_1) (1 - H_2) * (T_1 + T_2 + T_3)$$

### Average Cost per Byte

- Average cost per byte of the memory = 
$$\frac{C_1 S_1 + C_2 S_2 + C_3 S_3}{S_1 + S_2 + S_3}$$

## Simultaneous Access Memory Organization

- In simultaneous access all the levels of memory are directly connected to the CPU, whenever CPU requires any word, it starts searching for it in all the levels simultaneously.



- |  |  |  |
|--|--|--|
| <ul style="list-style-type: none"> <li>• <math>T_1</math> = Access time of level L<sub>1</sub></li> <li>• <math>S_1</math> = Size of level L<sub>1</sub></li> <li>• <math>C_1</math> = Cost per byte of level L<sub>1</sub></li> <li>• <math>H_1</math> = Hit rate of level L<sub>1</sub></li> </ul> | <ul style="list-style-type: none"> <li>• <math>T_2</math> = Access time of level L<sub>2</sub></li> <li>• <math>S_2</math> = Size of level L<sub>2</sub></li> <li>• <math>C_2</math> = Cost per byte of level L<sub>2</sub></li> <li>• <math>H_2</math> = Hit rate of level L<sub>2</sub></li> </ul> | <ul style="list-style-type: none"> <li>• <math>T_3</math> = Access time of level L<sub>3</sub></li> <li>• <math>S_3</math> = Size of level L<sub>3</sub></li> <li>• <math>C_3</math> = Cost per byte of level L<sub>3</sub></li> <li>• <math>H_3</math> = Hit rate of level L<sub>3</sub></li> </ul> |
|--|--|--|

Effective Memory Access Time (EMAT) =

$$H_1 * T_1 + (1 - H_1) * H_2 * T_2 + (1 - H_1) (1 - H_2) * H_3 * T_3$$

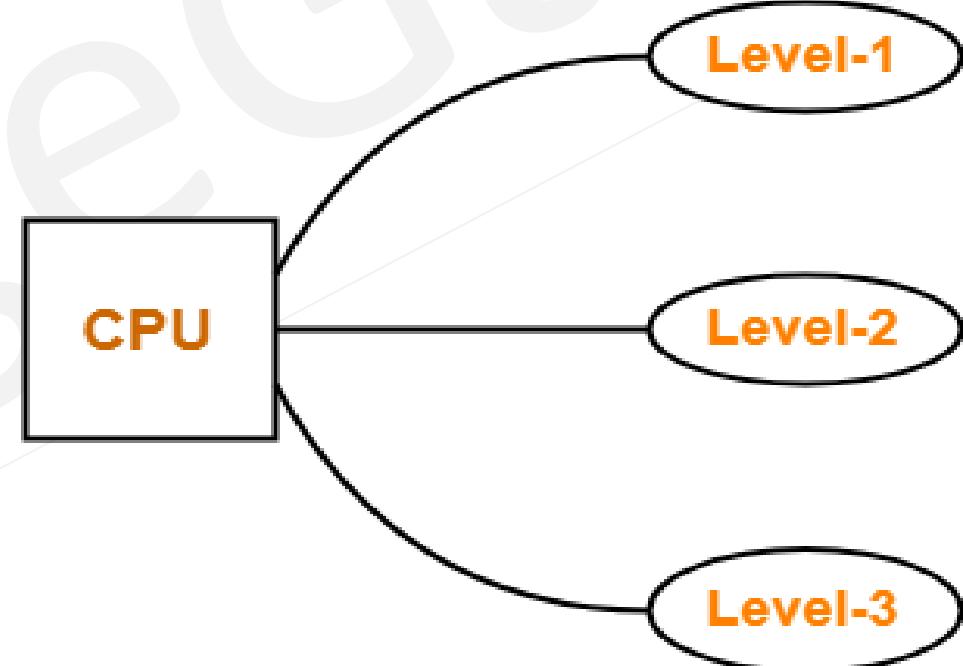
In any memory organization, the data item being searched will definitely be present in the last level (or secondary memory).

Thus, hit rate for the last level is always 1. So,

$$H_1 * T_1 + (1 - H_1) * H_2 * T_2 + (1 - H_1) (1 - H_2) * T_3$$

### Average Cost per Byte

- Average cost per byte of the memory =  $\frac{C_1 S_1 + C_2 S_2 + C_3 S_3}{S_1 + S_2 + S_3}$



**Example:** Calculate the EMAT for a machine with a cache hit rate of 80% where cache access time is 5ns and main memory access time is 100ns, both for simultaneous and hierarchical access.

**Q** Assume that for a certain processor, a read request takes 50 nanoseconds on a cache miss and 5 nanoseconds on a cache hit. Suppose while running a program, it was observed that 80% of the processor's read requests result in a cache hit. The average read access time in nanoseconds is \_\_\_\_\_ . **(GATE-2015) (2 Marks)**

## Cache Coherence Problem

- If multiple copy of same data is maintained at different level of memories then inconsistency may occur, this problem is known as cache coherence problem. Cache coherence problem can be resolved using the following techniques: Write Through, Write Back.
- **Write Through**
  - Write through is used to maintain the consistency between the cache and main memory. According to it if the cache copy is updated, at the same time main memory is also updated.
  - **Advantages**:- It provides the highest level of consistency.
  - **Disadvantages**:-It requires more number of memory access.
- **Write Back**
  - Write back is also used to maintain the consistency between the cache and main memory.
  - According to it all the changes performed on cache are reflected back to the main memory in the end.
  - **Advantage**:- Less number of memory accesses and less write operations.
  - **Disadvantage**:- Inconsistency may occur.

**Q** Let  $W_B$  and  $W_T$  be two set associative cache organizations that use LRU algorithm for cache block replacement.  $W_B$  is a write back cache and  $W_T$  is a write through cache. Which of the following statements is/are FALSE? **(GATE 2022) (1 MARKS)**

- (A) Each cache block in  $W_B$  and  $W_T$  has a dirty bit.
- (B) Every write hit in  $W_B$  leads to a data transfer from cache to main memory.
- (C) Eviction of a block from  $W_T$  will not lead to data transfer from cache to main memory.
- (D) A read miss in  $W_B$  will never lead to eviction of a dirty block from  $W_B$ .

**Q** Assume a two-level inclusive cache hierarchy,  $L_1$  and  $L_2$ , where  $L_2$  is the larger of the two. Consider the following statements.

- $S_1$ : Read misses in a write through  $L_1$  cache do not result in writebacks of dirty lines to the  $L_2$
- $S_2$ : Write allocate policy must be used in conjunction with write through caches and no-write allocate policy is used with writeback caches.

Which of the following statements is correct? **(GATE 2021) (2 MARKS)**

- (a)  $S_1$  is true and  $S_2$  is false
- (b)  $S_1$  is false and  $S_2$  is true
- (c)  $S_1$  is true and  $S_2$  is true
- (d)  $S_1$  is false and  $S_2$  is false

**Q** For inclusion to hold between two cache levels  $L_1$  and  $L_2$  in a multi-level cache hierarchy, which of the following are necessary?

## (Gate-2008) (2 Marks)

- I. L<sub>1</sub> must be a write-through cache
  - II. L<sub>2</sub> must be a write-through cache
  - III. The associativity of L<sub>2</sub> must be greater than that of L1
  - IV. The L<sub>2</sub> cache must be at least as large as the L1 cache

(A) IV only

(C) I, III and IV only

**Q** A cache memory that has a hit rate of 0.8 has an access latency 10 ns and miss penalty 100 ns. An optimization is done on the cache to reduce the miss rate. However, the optimization results in an increase of cache access latency to 15 ns, whereas the miss penalty is not affected. The minimum hit rate (rounded off to two decimal places) needed after the optimization such that it should not increase the average memory access time is \_\_\_\_\_ . **(GATE 2022) (1 MARKS)**

**Q** In a two-level cache system, the access times of  $L_1$  and  $L_2$  is 1 and 8 clock cycles, respectively. The miss penalty from the  $L_2$  cache to main memory is 18 clock cycles. The miss rate of  $L_1$  cache is twice that of  $L_2$ . The average memory access time (AMAT) of this cache system is 2 cycles. The miss rates of  $L_1$  and  $L_2$  respectively are: (Gate-2017) (2 Marks)

- (A) 0.111 and 0.056
- (C) 0.0892 and 0.1784

- (B) 0.056 and 0.111
- (D) 0.1784 and 0.0892

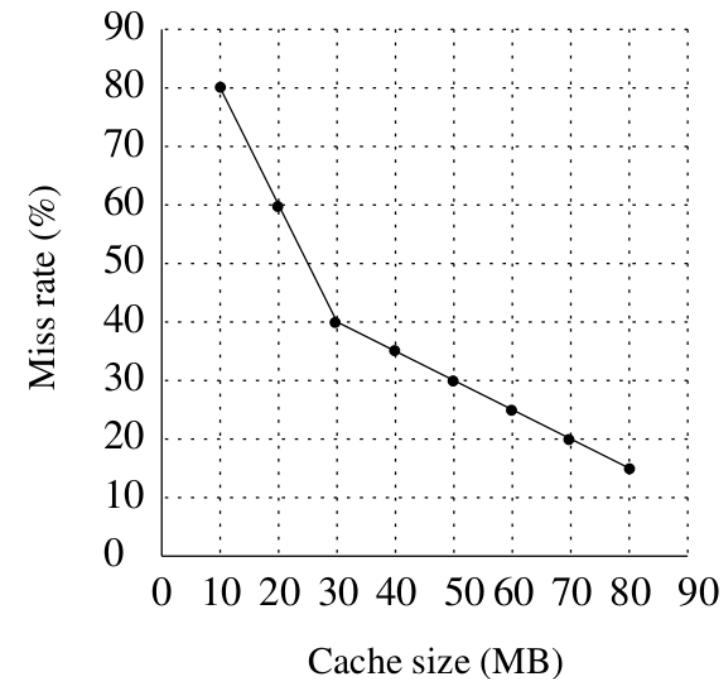
**Q** The read access times and the hit ratios for different caches in a memory hierarchy are as given below:

Cache	Read access time (in nanoseconds)	Hit ratio
I-cache	2	0.8
D-cache	2	0.9
L2-cache	8	0.9

The read access time of main memory is 90 nanoseconds. Assume that the caches use the referred-word-first read policy and the writeback policy. Assume that all the caches are direct mapped caches. Assume that the dirty bit is always 0 for all the blocks in the caches. In execution of a program, 60% of memory reads are for instruction fetch and 40% are for memory operand fetch. The average read access time in nanoseconds (up to 2 decimal places) is \_\_\_\_\_ (Gate-2017) (2 Marks)

**Q** Consider a two-level cache hierarchy  $L_1$  and  $L_2$  caches. An application incurs 1.4 memory accesses per instruction on average. For this application, the miss rate of  $L_1$  cache 0.1, the  $L_2$  cache experience on average. 7 misses per 1000 instructions. The miss rate of  $L_2$  expressed correct to two decimal places is \_\_\_\_\_. (Gate-2017) (1 Marks)

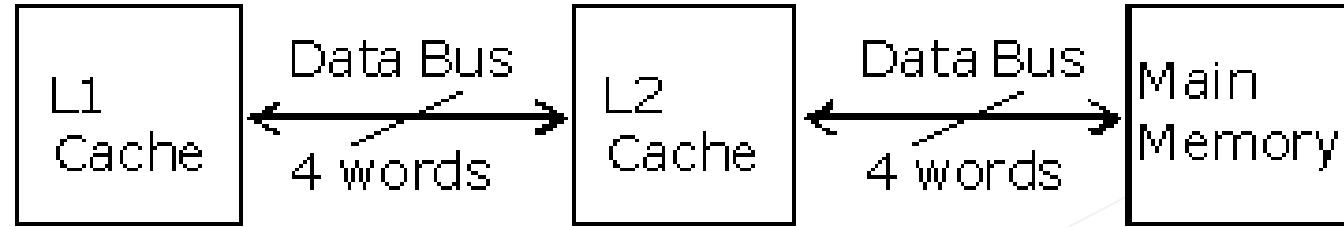
**Q** A file system uses an in-memory cache to cache disk blocks. The miss rate of the cache is shown in the figure. The latency to read a block from the cache is 1 ms and to read a block from the disk is 10 ms. Assume that the cost of checking whether a block exists in the cache is negligible. Available cache sizes are in multiples of 10 MB.



The smallest cache size required to ensure an average read latency of less than 6 ms  
is \_\_\_\_\_ MB. (Gate-2016) (2 Marks)

**Q** The memory access time is 1 nanosecond for a read operation with a hit in cache, 5 nanoseconds for a read operation with a miss in cache, 2 nanoseconds for a write operation with a hit in cache and 10 nanoseconds for a write operation with a miss in cache. Execution of a sequence of instructions involves 100 instruction fetch operations, 60 memory operand read operations and 40 memory operands write operations. The cache hit-ratio is 0.9. The average memory access time (in nanoseconds) in executing the sequence of instructions is \_\_\_\_\_. **(Gate-2014) (2 Marks)**

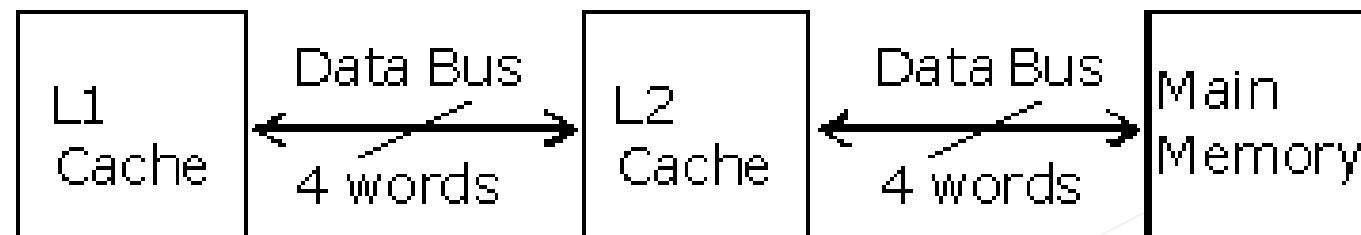
**Q** A computer system has an L<sub>1</sub> cache, an L<sub>2</sub> cache, and a main memory unit connected as shown below. The block size in L<sub>1</sub> cache is 4 words. The block size in L<sub>2</sub> cache is 16 words. The memory access times are 2 nanoseconds, 20 nanoseconds and 200 nanoseconds for L<sub>1</sub> cache, L<sub>2</sub> cache and main memory unit respectively.



When there is a miss in L<sub>1</sub> cache and a hit in L<sub>2</sub> cache, a block is transferred from L<sub>2</sub> cache to L<sub>1</sub> cache. What is the time taken for this transfer? **(Gate-2010) (2 Marks)**

- (A) 2 nanoseconds
- (B) 20 nanoseconds
- (C) 22 nanoseconds
- (D) 88 nanoseconds

**Q** A computer system has an L<sub>1</sub> cache, an L<sub>2</sub> cache, and a main memory unit connected as shown below. The block size in L<sub>1</sub> cache is 4 words. The block size in L<sub>2</sub> cache is 16 words. The memory access times are 2 nanoseconds, 20 nanoseconds and 200 nanoseconds for L<sub>1</sub> cache, L<sub>2</sub> cache and main memory unit respectively.



**Q** When there is a miss in both L<sub>1</sub> cache and L<sub>2</sub> cache, first a block is transferred from main memory to L<sub>2</sub> cache, and then a block is transferred from L<sub>2</sub> cache to L<sub>1</sub> cache. What is the total time taken for these transfers? **(Gate-2010) (2 Marks)**

- (A) 222 nanoseconds
- (B) 888 nanoseconds
- (C) 902 nanoseconds
- (D) 968 nanoseconds

**Q** Consider a system with 2 level caches. Access times of Level<sub>1</sub> cache, Level<sub>2</sub> cache and main memory are 1 ns, 10ns, and 500 ns, respectively. The hit rates of Level<sub>1</sub> and Level<sub>2</sub> caches are 0.8 and 0.9, respectively. What is the average access time of the system ignoring the search time within the cache? **(Gate-2004) (1 Marks)**

- (A)** 13.0 ns      **(B)** 12.8 ns      **(C)** 12.6 ns      **(D)** 12.4 ns

**Q** A dynamic RAM has a memory cycle time of 64 nsec. It has to be refreshed 100 times per msec and each refresh takes 100 nsec. What percentage of the memory cycle time is used for refreshing? **(Gate-2005) (1 Marks)**

**Q**A cache line is 64 bytes. The main memory has latency 32ns and bandwidth 1G.Bytes/s. The time required to fetch the entire cache line from the main memory is **(Gate-2006) (2 Marks)**

- (A) 32 ns
- (B) 64 ns
- (C) 96 ns
- (D) 128 ns

## Input / Output management

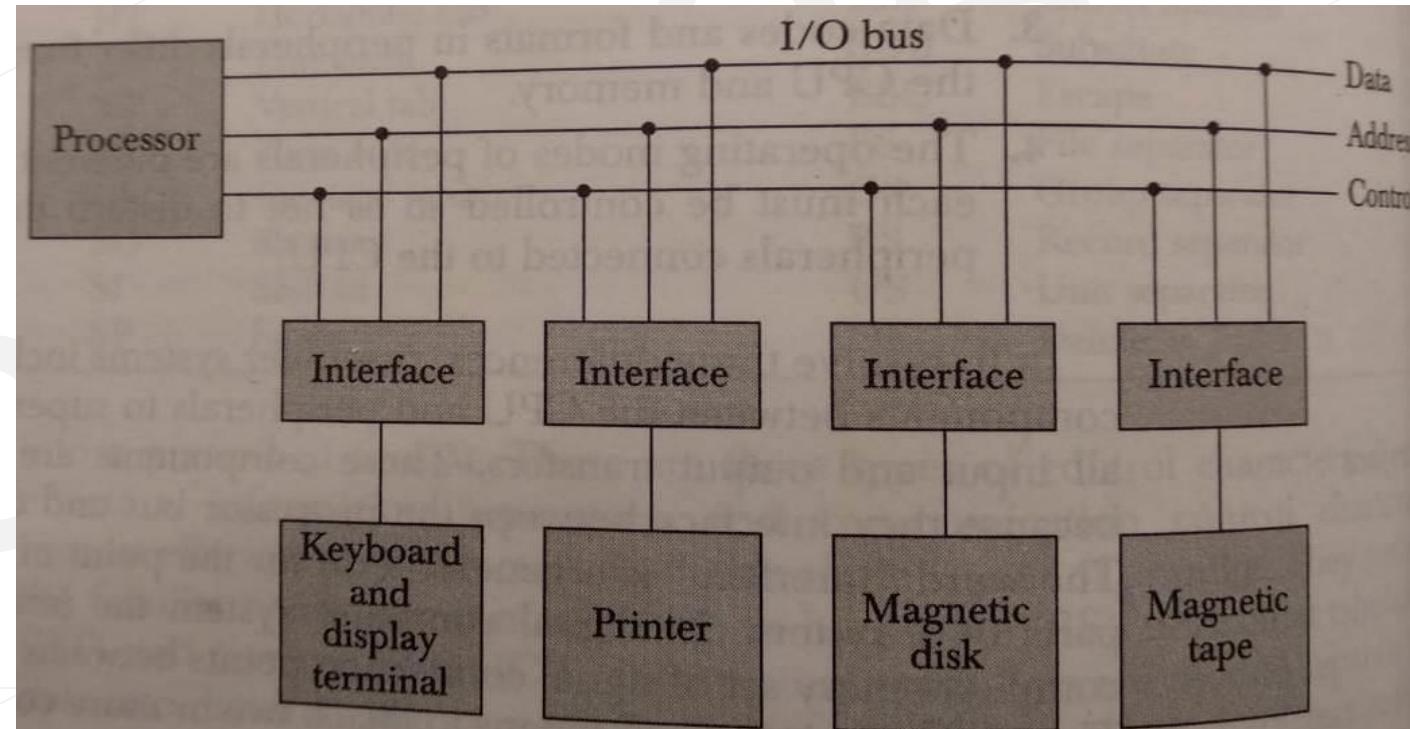
- In general, when we say a computer, we understand a CPU + Memory (cache, main memory). But a computer does not serve any purpose if it cannot receive data from the outside world or cannot transmit the data to outside world.
- I/o or peripheral devices are those independent devices which serve this purpose. So, while designing i/o for a computer we must know the number of i/o device and the capacity of each device.



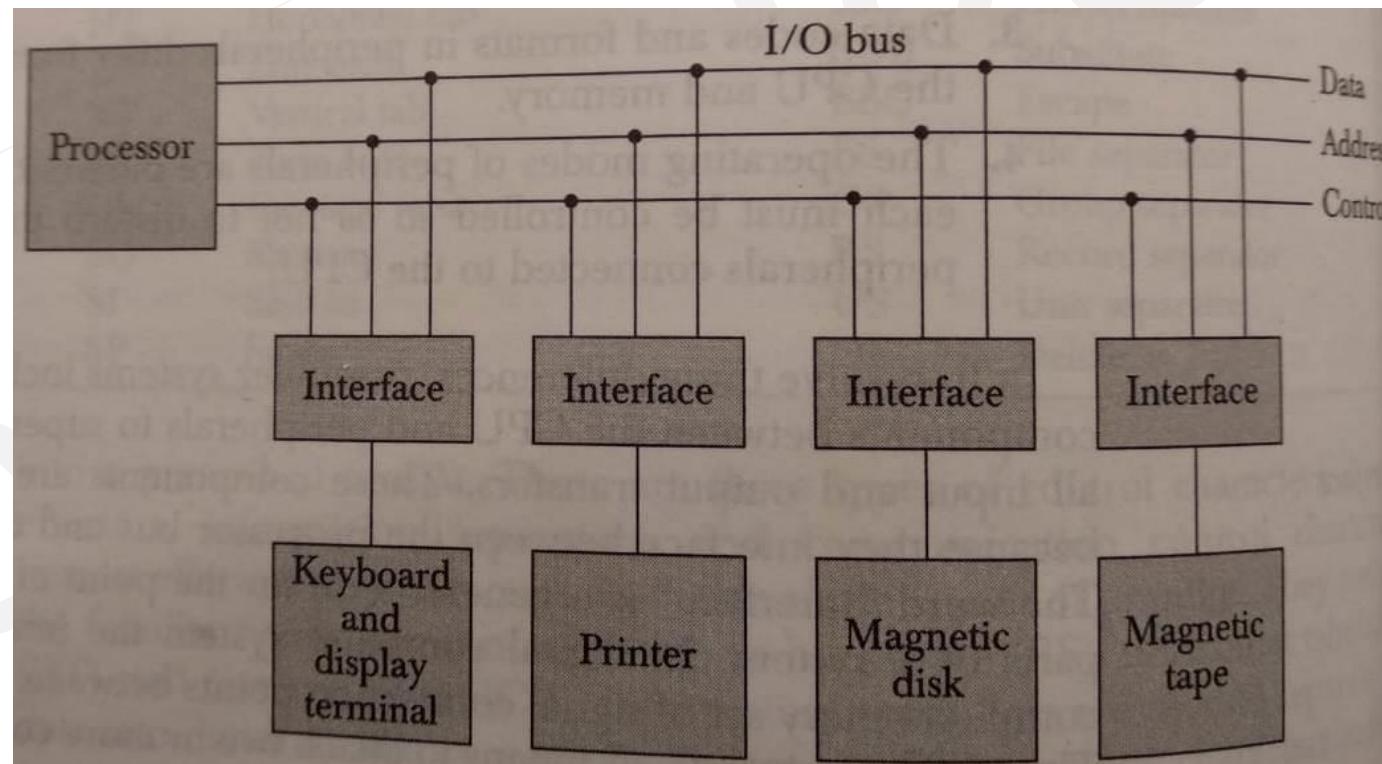
## Interface

we cannot directly connect a i/o device to computer because of the following reasons

- Speed: - The speed of CPU and i/o device will usually different.
- Format: - The data code and format of CPU and peripherals may be different. E.g. ASCII, Unicode etc.
- Physical orientation: - Different device have organizations like optical, magnetic, electrochemical and different controlling functions.
- Signal conversion: - peripherals are electromagnetic and electrochemical device and their manner of operation is different from the operations of CPU and memory, which are electronic device, signal conversion is required



- **Address bus:** - Is used to identify the correct i/o devices among the number of i/o device , so CPU put an address of a specific i/o device on the address line, all devices keep monitoring this address bus and decode it, and if it is a match then it activates control and data lines.
- **Control bus:** - After selecting a specific i/o device CPU sends a functional code on the control line. The selected device(interface) reads that functional code and execute it. E.g. i/o command, control command, status command etc.
- **Data bus:** - In the final step depending on the operation either CPU will put data on the data line and device will store it or device will put data on the data line and CPU will store it.

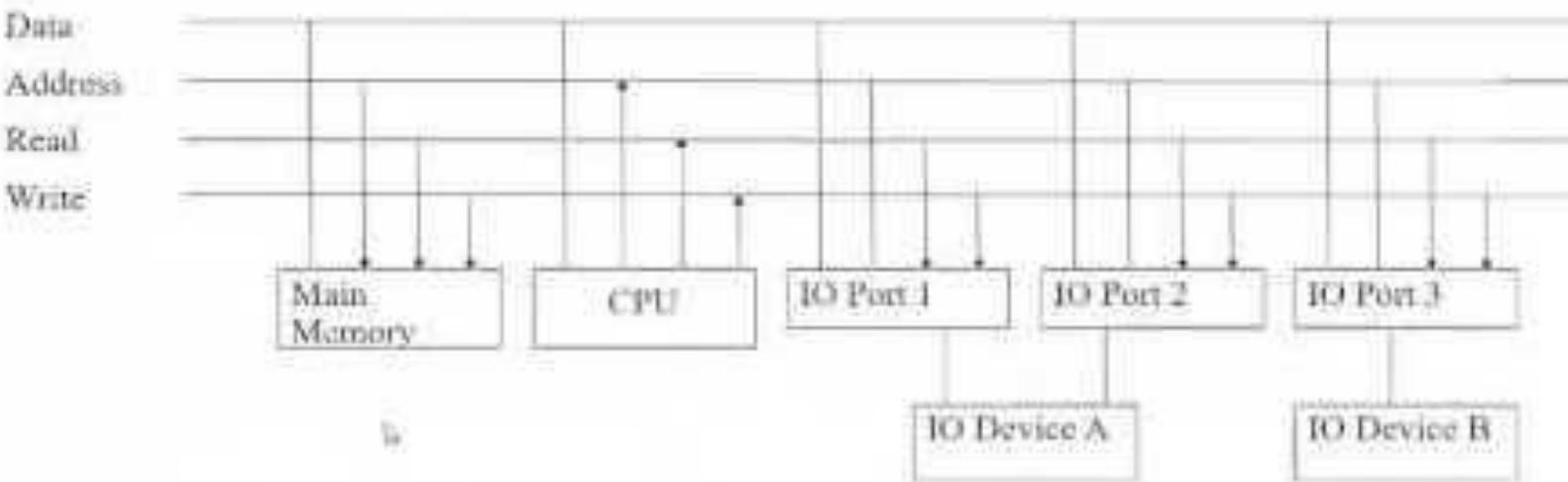


**Q** Consider a main memory system that consists of 8 memory modules attached to the system bus, which is one word wide. When a write request is made, the bus is occupied for 100 nanoseconds (ns) by the data, address, and control signals. During the same 100 ns, and for 500 ns thereafter, the addressed memory module executes one cycle accepting and storing the data. The (internal) operation of different memory modules may overlap in time, but only one request can be on the bus at any time. The maximum number of stores (of one word each) that can be initiated in 1 millisecond is \_\_\_\_\_ (Gate-2014) (2 Marks)

- (A) 1,000
- (B) 10,000
- (C) 1,00,000
- (D) 100

# How a computer (CPU) deals with Memory and I/O devices

## Memory Mapped i/o



8085

- Here there is no separate i/o instructions. The CPU can manipulate i/o data residing in the interface registers with the same instructions that are used to manipulate memory words. Here computer can use memory type instructions for i/o data. E.g. 8085
- **Advantage:** - In typical computer there are more memory reference instruction than i/o instruction, but in memory mapped i/o all instructions that refer to memory are also available for i/o.
- **Disadvantage:** - Total address get divided, some range is occupied by i/o while some memory.

- The **Commodore 64**, also known as the **C64** or the **CBM 64**, is an 8-bit home computer introduced in January 1982 by Commodore International (first shown at the Consumer Electronics Show, in Las Vegas, January 7–10, 1982).
- It has been listed in the Guinness World Records as the highest-selling single computer model of all time, with independent estimates placing the number sold between 10 and 17 million units.



# Isolated i/o

- Here common bus to transfer data between memory or i/o and CPU. The distinction between a memory and i/o transfer is made through separate read and write line. i/o read and i/o write control lines are enabled during an i/o transfer. Memory read and memory write control lines are enabled during a memory transfer. E.g 8086.
- Advantage: - Here memory is used efficiently as the same address can be used two times.
- Disadvantage: - Need different control lines one for memory and other for i/o devices.

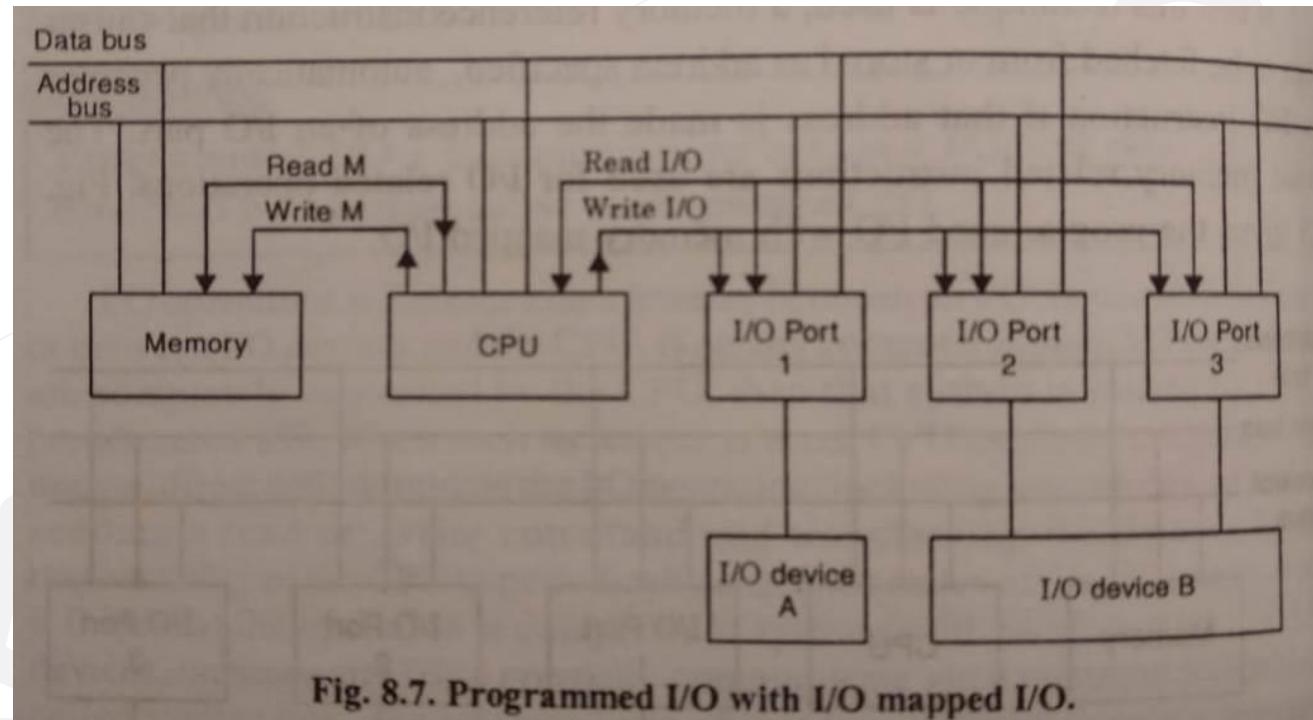
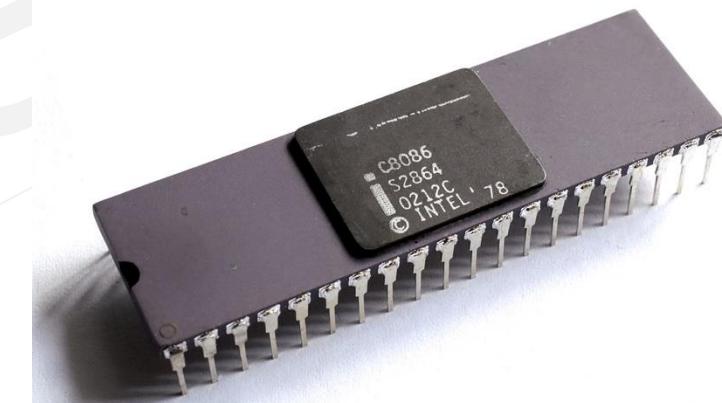
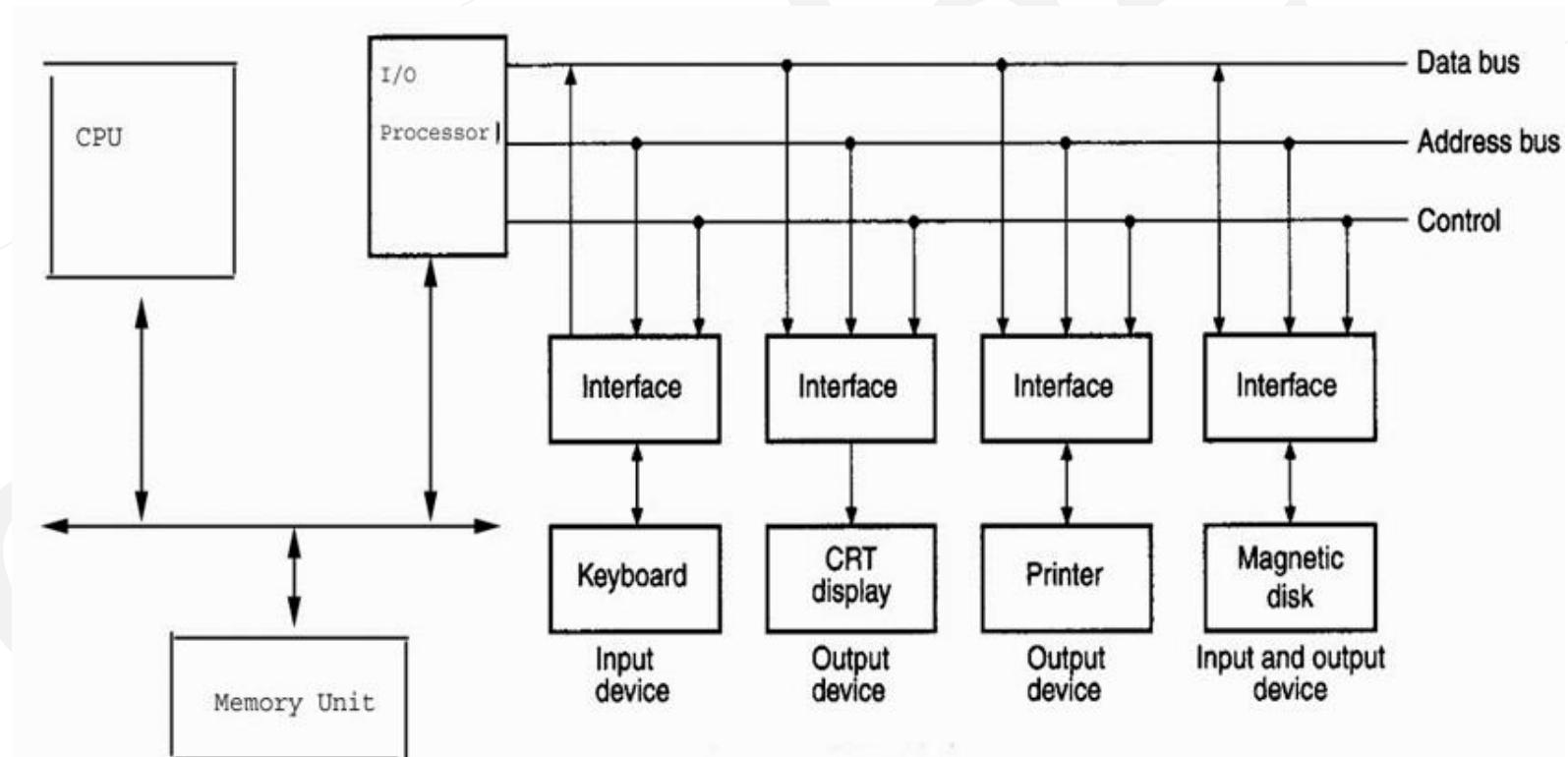


Fig. 8.7. Programmed I/O with I/O mapped I/O.



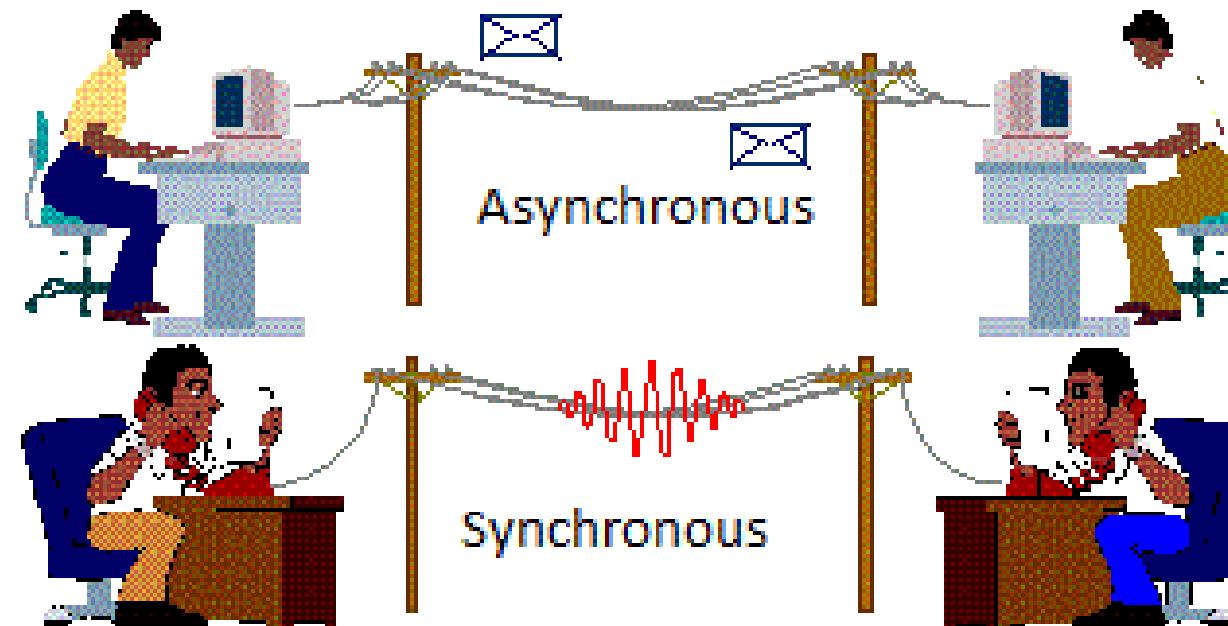
# I/O Processor

- Computer has independent sets of data, address and control buses, one for accessing memory and the other for i/o. this is done in computers that provide a separate i/o processor other than CPU.
- Memory communicate with both the CPU and iop through a memory bus.
- Iop communicates also with the input and output devices through a separate i/o bus with its own address, data and control lines. The purpose of iop is to provide an independent pathway for the transfer of information between external devices and internal memory.



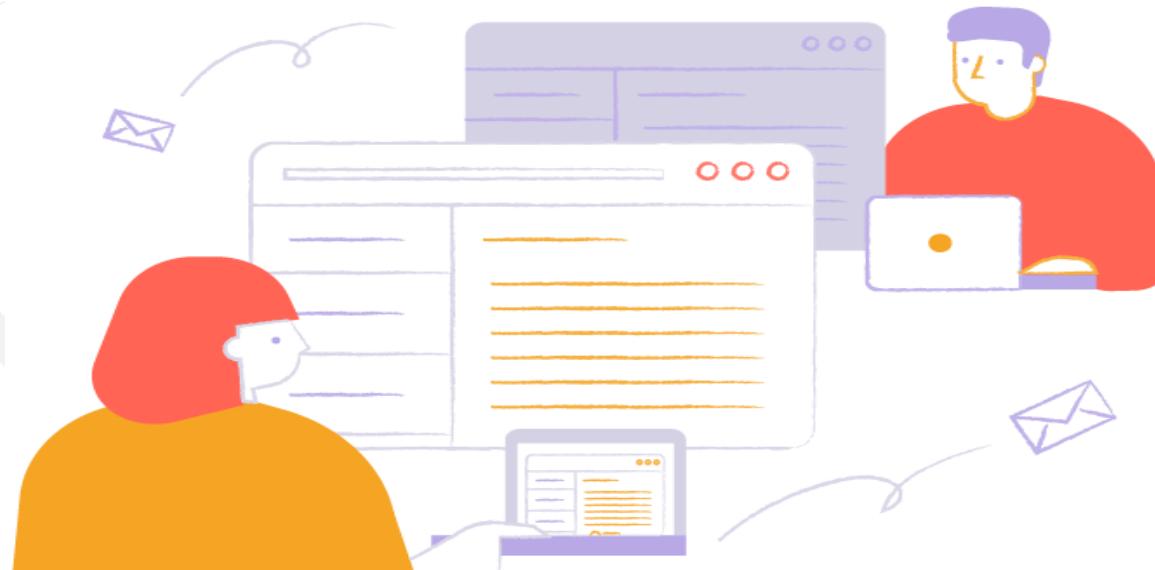
## Synchronous Vs Asynchronous data transfer

- Synchronization is achieved by a device called master generator, which generate a periodic train of clock pulse.
- The internal operations in a digital system are synchronized by means of clock pulses supplied by a common pulse generator.
- When communication happens between devices which are under a same control unit or same clock then it is called synchronous communication. e.g. communication between CPU and its registers.

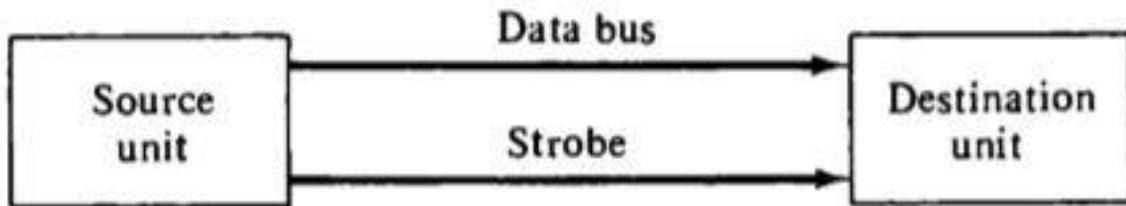


## Asynchronous communication

- When the timing units of two devices are independent that is they are under different control then it is called asynchronous communication.
- Asynchronous data transfer between two independent units required that control signals be transmitted between the communicating units to indicate the time at which data is being transmitted.
- One way to achieving this by means of strobe pulse supplied by one of the units to indicate to the other unit when the transfer has to occur.

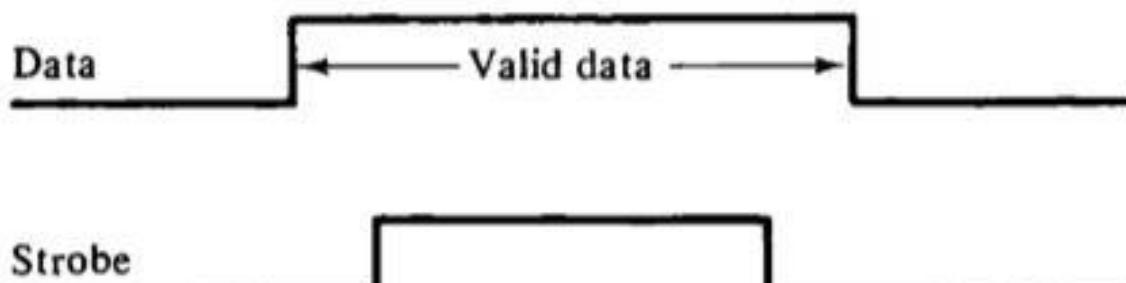


- The unit receiving the data item response with another control signal to acknowledge the receipt of the data. this type of agreement between two independent units is referred to as handshaking.
- The strobe pulse method and the handshaking method of asynchronous data transfer are not restricted to input output transfer. in fact they are used extensively on numerous occasions requiring the transfer of data between two independent units.



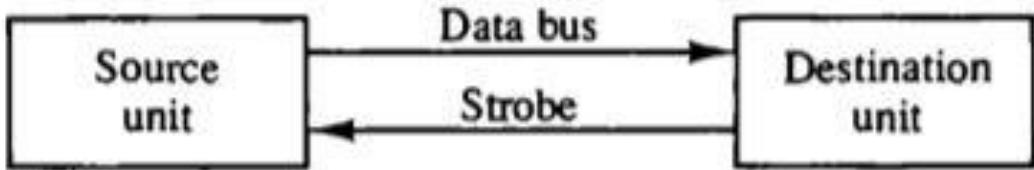
(a) Block diagram

## Source Initiated I/O



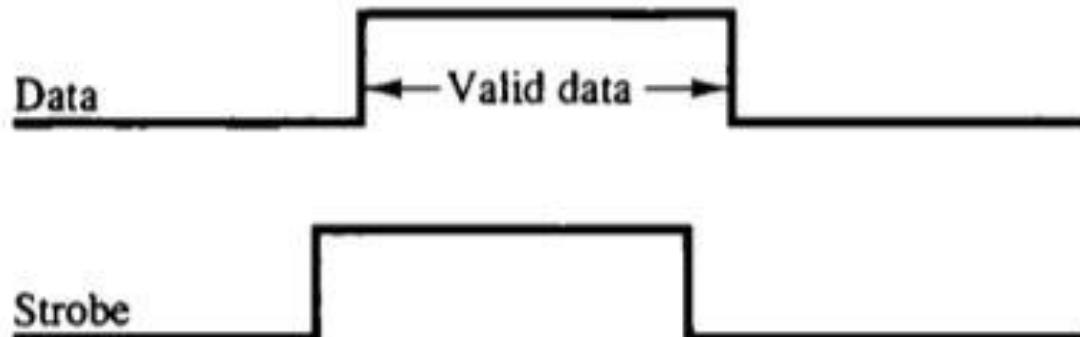
(b) Timing diagram

- The unit receiving the data item response with another control signal to acknowledge the receipt of the data. this type of agreement between two independent units is referred to as handshaking.
- The strobe pulse method and the handshaking method of asynchronous data transfer are not restricted to input output transfer. in fact they are used extensively on numerous occasions requiring the transfer of data between two independent units.



(a) Block diagram

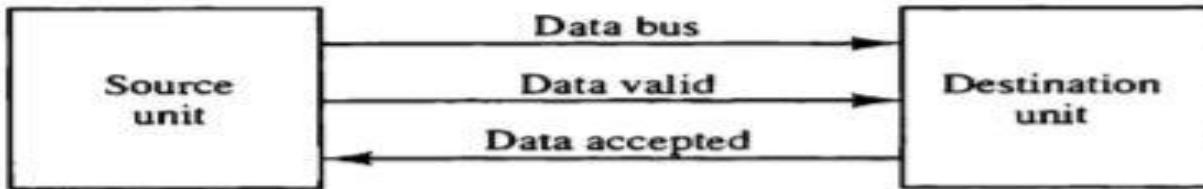
Destination Initiated I/O



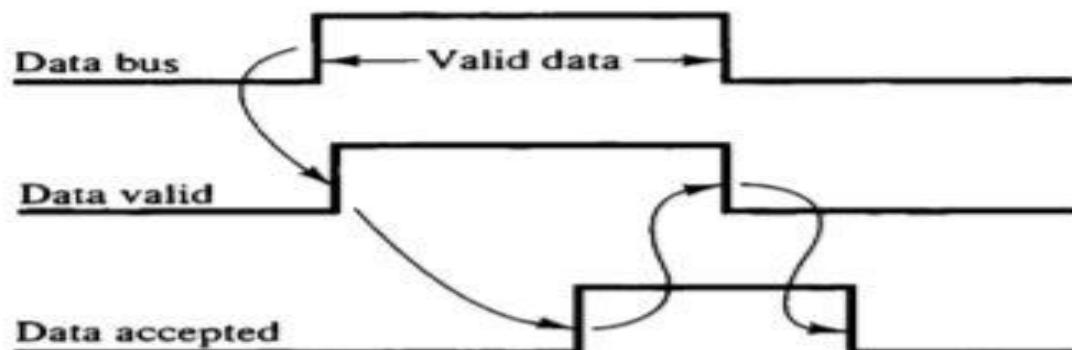
(b) Timing diagram

- the unit receiving the data item response with another control signal to acknowledge the receipt of the data. this type of agreement between two independent units is referred to as handshaking.
- The strobe pulse method and the handshaking method of asynchronous data transfer are not restricted to input output transfer. in fact they are used extensively on numerous occasions requiring the transfer of data between two independent units.

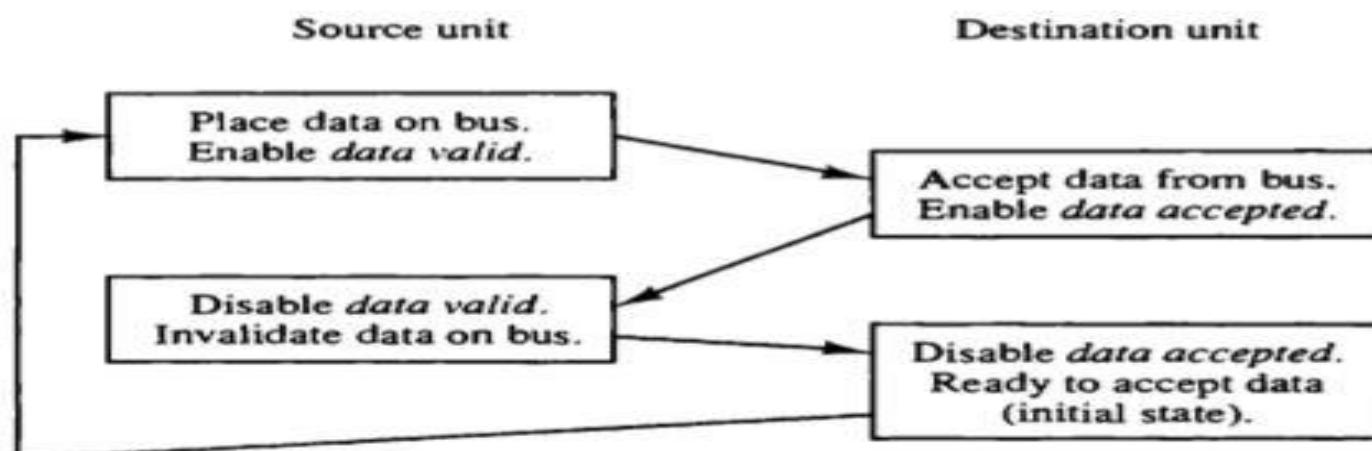
# Source Initiated Transfer



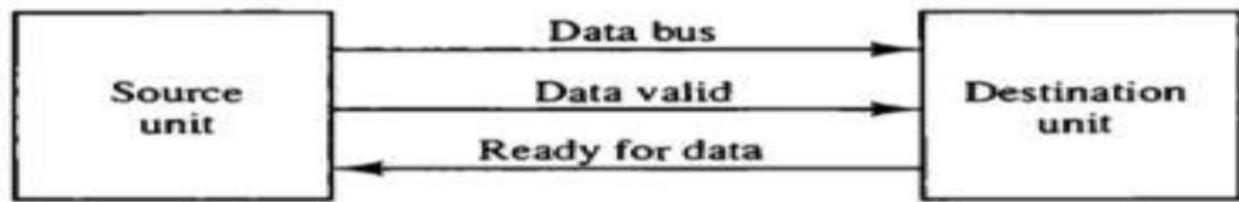
(a) Block diagram



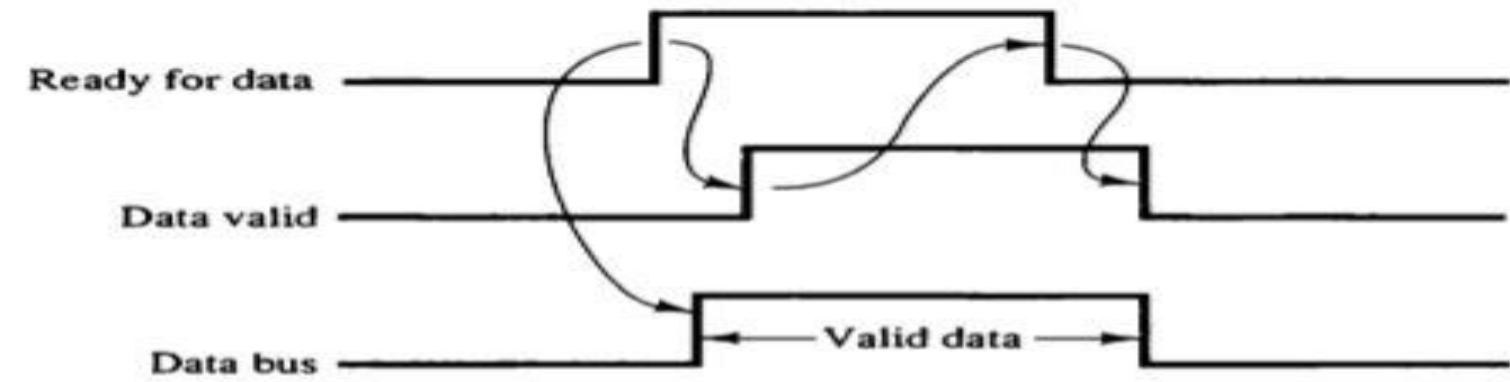
(b) Timing diagram



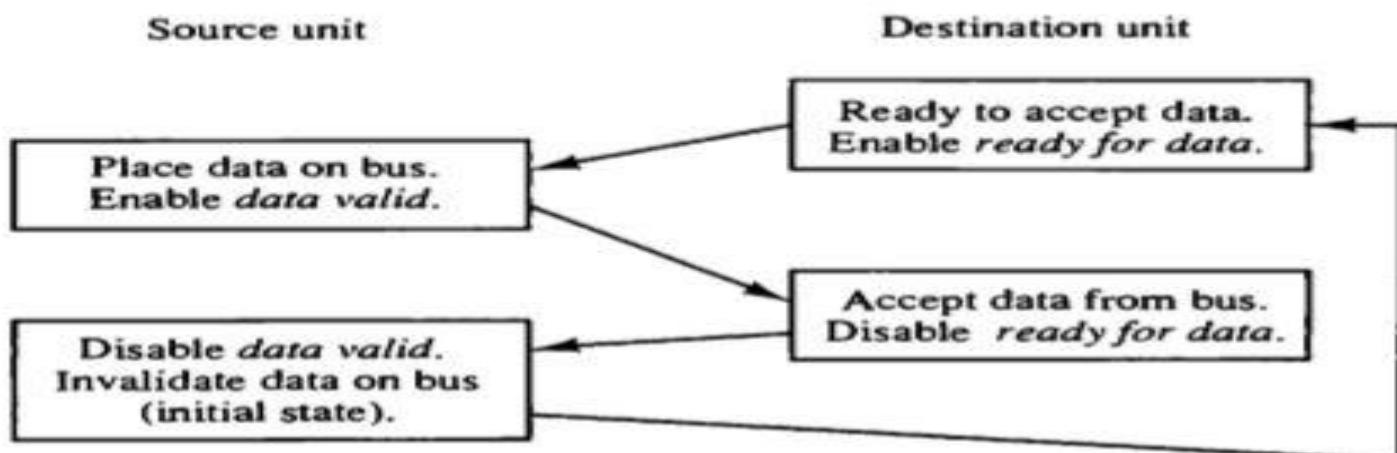
# Destination initiated transfer



(a) Block diagram



(b) Timing diagram



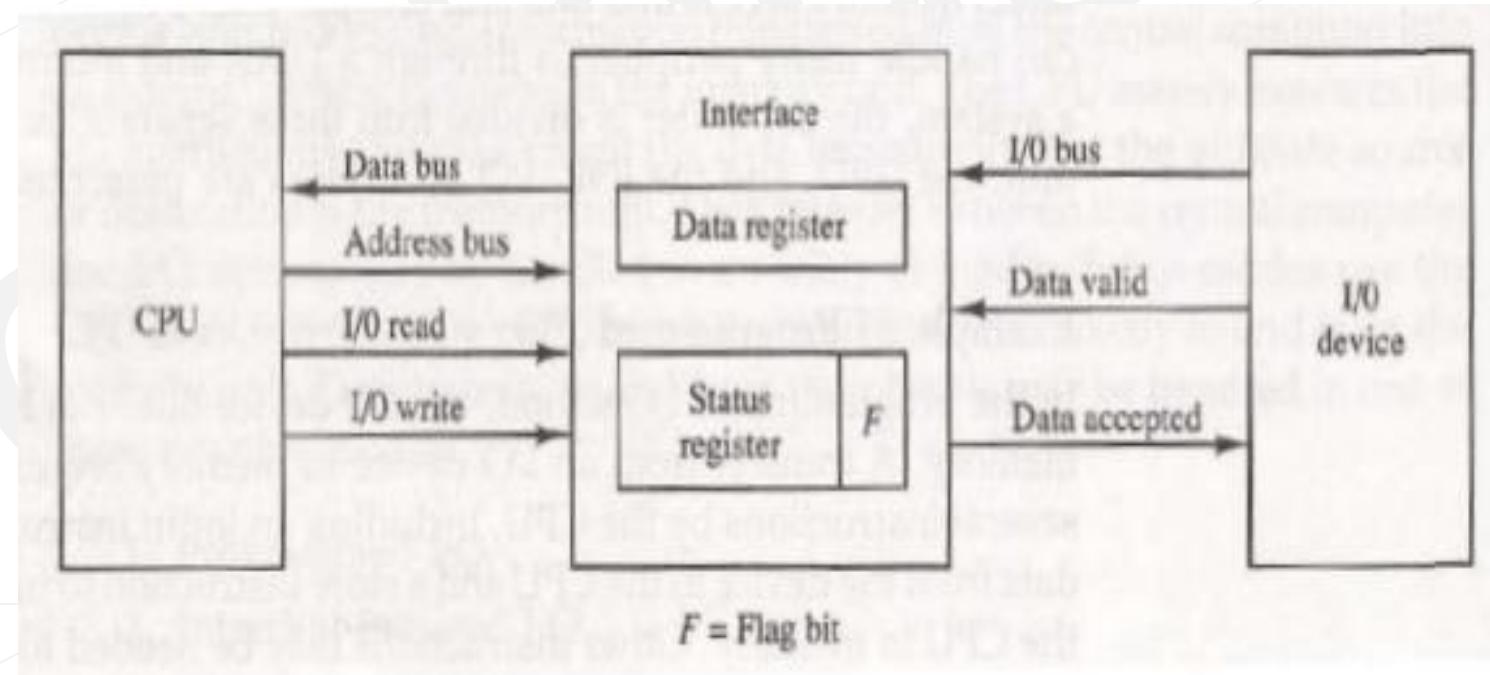
## Modes of Data Transfer

Here we deal with the problem that how data communication will take place CPU and i/o device. There are popularly three methods of data transfer: -

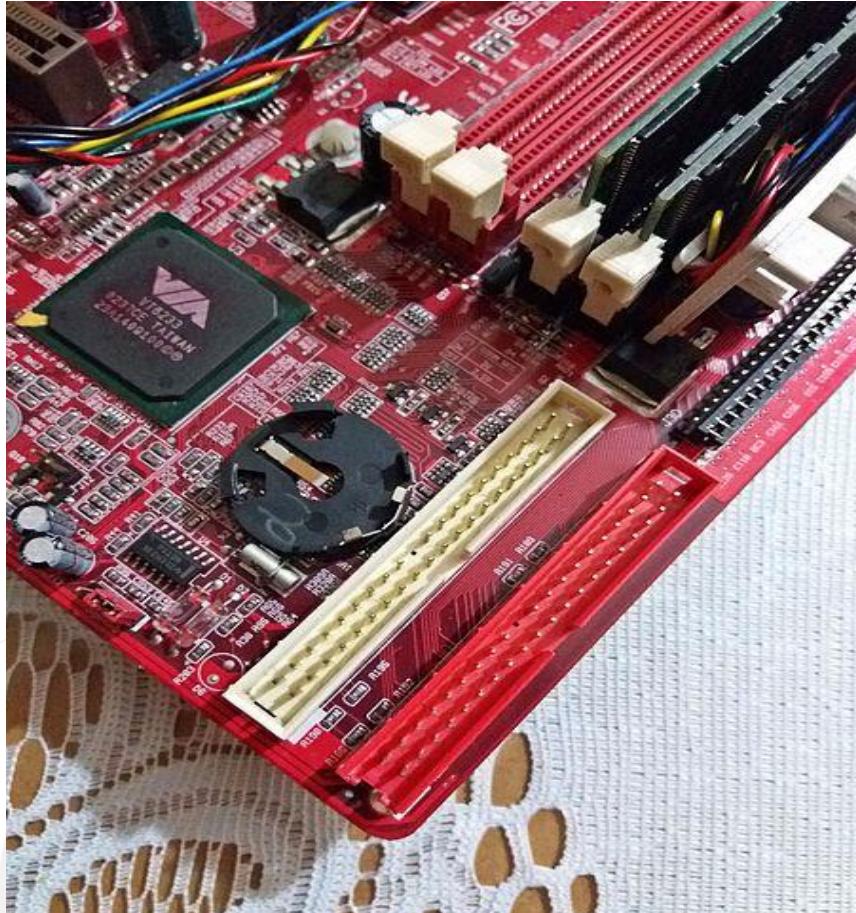
- **Programmed i/o**
- **Interrupt initiated i/o**
- **Direct Memory Transfer**

## Programmed I/O

- In this i/o device cannot access the memory directly. To complete data transfer number of instructions will be executed out of which input instruction are those which transfer data from device to CPU and store instructions from CPU to memory.
- **Step 1:** - i/o device will put data on the data bus and will enable valid data signal.
- **Step 2:** - Interface is continuously sensing for data valid signal and when it receives the signal it will copy data from the data bus into it's data register and set its flag bit to 1 and enable data accepted line.
- **Step 3:** - CPU is continuously monitoring the status register in the programmed mode and as soon as it sees flag bit as 1, it immediately copies data from data register on the data bus and clear flag bit to zero.
- **Step 4:** - Now interface will disable data accepted line to tell i/o device, I am ready for new transitions.
- **Conclusion:** - CPU works in programmed mode or in busy wait mode so no of clock cycles are wasted. It will be difficult to handle multiple i/o device at the same time. It is not appropriate with the high-speed i/o devices.



- The best known example of a PC device that uses programmed I/O is the **ATA interface**
- **Parallel ATA (PATA)**, originally **AT Attachment**, is an interface standard for the connection of storage devices such as hard disk drives, floppy disk drives, and optical disc drives in computers.



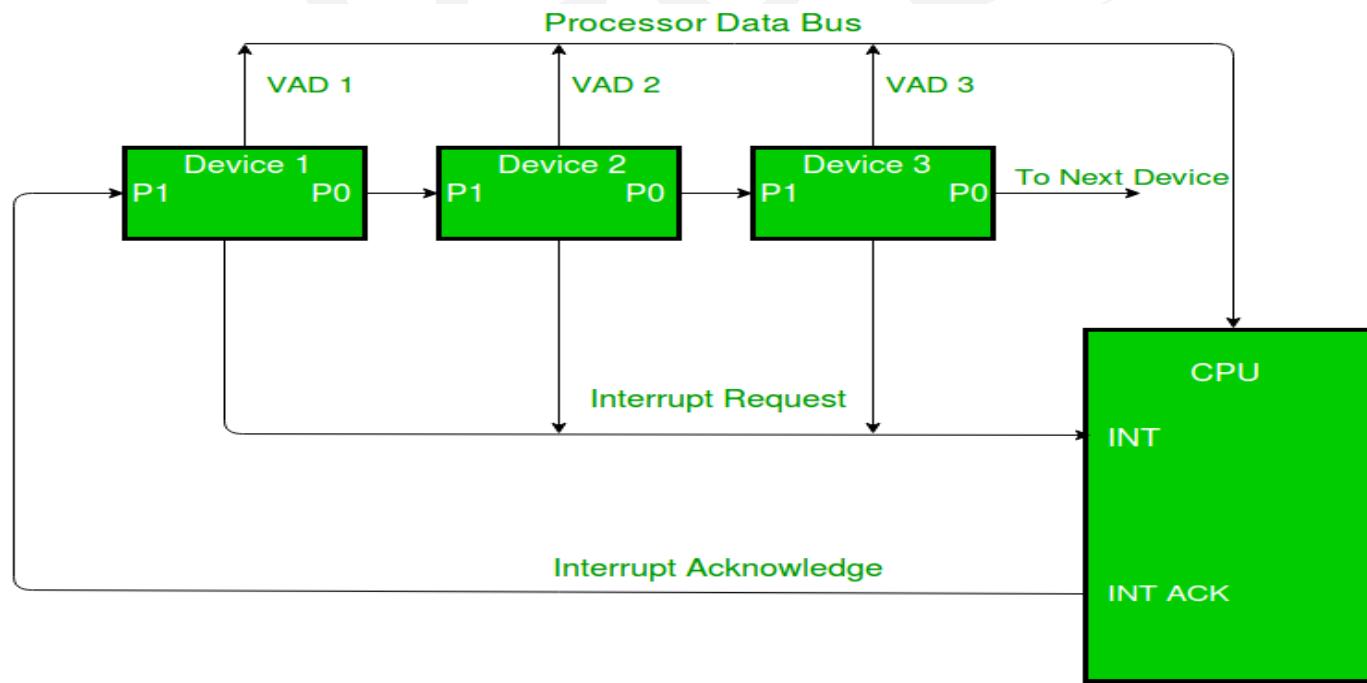
## Interrupt initiated i/o

- In this method I/o device interrupt CPU when it is ready for data transfer. CPU keep executing instructions and after executing one instruction and before starting another instructions CPU wait and see if there is interrupt or not and if there is interrupt then takes a decision whether CPU should entertain this interrupt or continue with the execution.
- **Note:** - instruction is always absolute in nature and there is nothing like partial execution of instruction.
- The method of handling interrupts of different i/o devices are different, therefor every device has a program or routine called ISR (interrupt service routine) which tell CPU how the interrupt should be managed (and it saves CPU time).
- Interrupt can be of two types: -
  - **Non-vectorized interrupt:** - Here there is a mutual understanding between CPU and device that where this routine is stored in memory (high priority device)
  - **Vectored interrupt:** - Some interrupts may be vectored where the interrupting device will also tell the address that where this routine is stored in the memory.
- It is possible that different i/o devices interrupt at the same time. Now CPU must have a priority decision that which interrupt should be service first and which should be service later.

Intel 82C59A



- **H/w solution:** - It can be serial or parallel, serial solution is known as Daisy Channing is a h/w solution which is used to decide priority among different i/o devices (VAD- vector address device).
- Out of all possible devices 1 or more device may send an interrupt with a common line. When CPU completes 1 instruction and check interrupt in line and found an interrupt, then CPU will enable interrupt acknowledgment line to 1.
- The i/o device placed first in the arrangement will always get acknowledgement first. If it wants to perform i/o then it will put 0 on priority outline and will put the address of its interrupt service routine (ISR) on the vector address line. If device do not want to perform i/o then will set priority output as 1 and will give chance to the second device, and the processor continue.
- **Advantage:** - very simple, easy to use, easy to understand, relatively fast.
- **Disadvantage:** - here priority fixed and even in case of requirement we cannot change it.



**Q** The following are some events that occur after a device controller issues an interrupt while process L is under execution. **(GATE-2018) (2 Marks)**

- (P)** The processor pushes the process status of L onto the control stack.
- (Q)** The processor finishes the execution of the current instruction.
- (R)** The processor executes the interrupt service routine.
- (S)** The processor pops the process status of L from the control stack.
- (T)** The processor loads the new PC value based on the interrupt.

**(A)** QPTRS

**(B)** PTRSQ

**(C)** TRPQS

**(D)** QTPRS

**Q A CPU generally handles an interrupt by executing an interrupt service routine:  
(GATE-2009) (1 Marks)**

- a) As soon as an interrupt is raised.
- b) By checking the interrupt register at the end of fetch cycle.
- c) By checking the interrupt register after finishing the execution of the current instruction.
- d) By checking the interrupt register at fixed time intervals.

**Q** For the daisy chain scheme of connecting I/O devices, which of the following statements is true? **(GATE-1996) (1 Marks)**

- a) It gives non-uniform priority to various devices
- b) It gives uniform priority to all devices
- c) It is only useful for connecting slow devices to a processor device
- d) It requires a separate interrupt pin on the processor for each device

**Q** A device with data transfer rate 10 KB/sec is connected to a CPU. Data is transferred byte-wise. Let the interrupt overhead be 4 microsec. The byte transfer time between the device interface register and CPU or memory is negligible. What is the minimum performance gain of operating the device under interrupt mode over operating it under program-controlled mode? **(GATE-2005)**

**(2 Marks)**

(A) 15

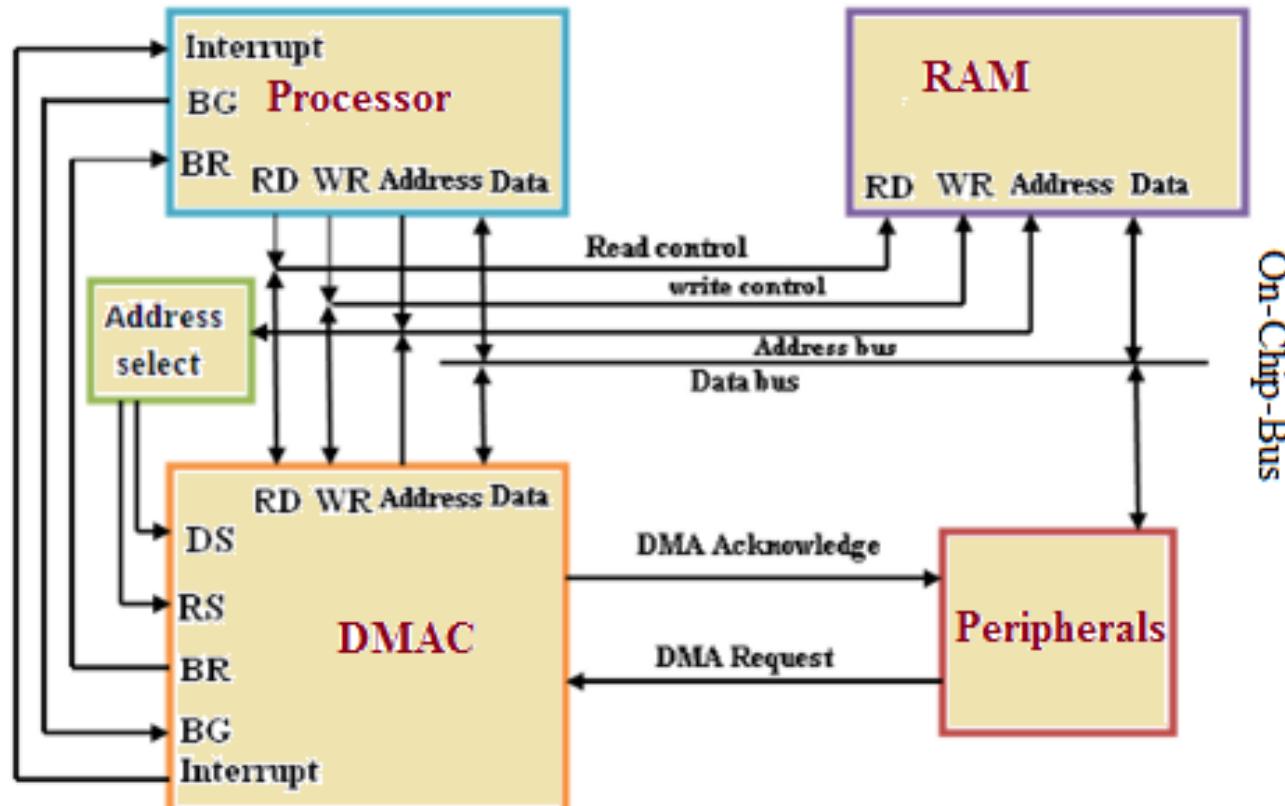
(B) 25

(C) 35

(D) 45

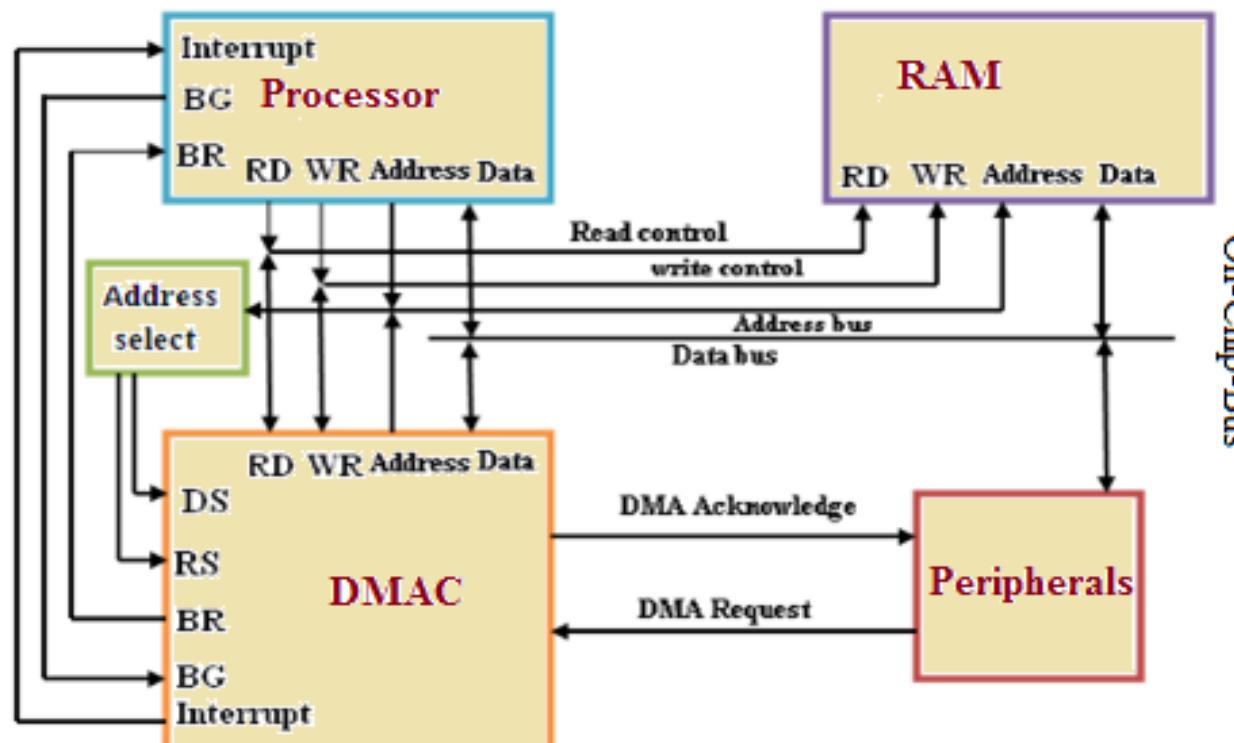
## Direct Memory Access (DMA)

- When we want to perform i/o operations then the actual source or destination is either i/o device or memory, but CPU is placed in between just to manage and control the transfer.
- **DMA** is the idea where we use a new device is call DMA controller using which CPU allow DMA controller to take control of system buses and perform direct data transfer either from device to memory or from memory to device.



## Sequence of DMA transfer: -

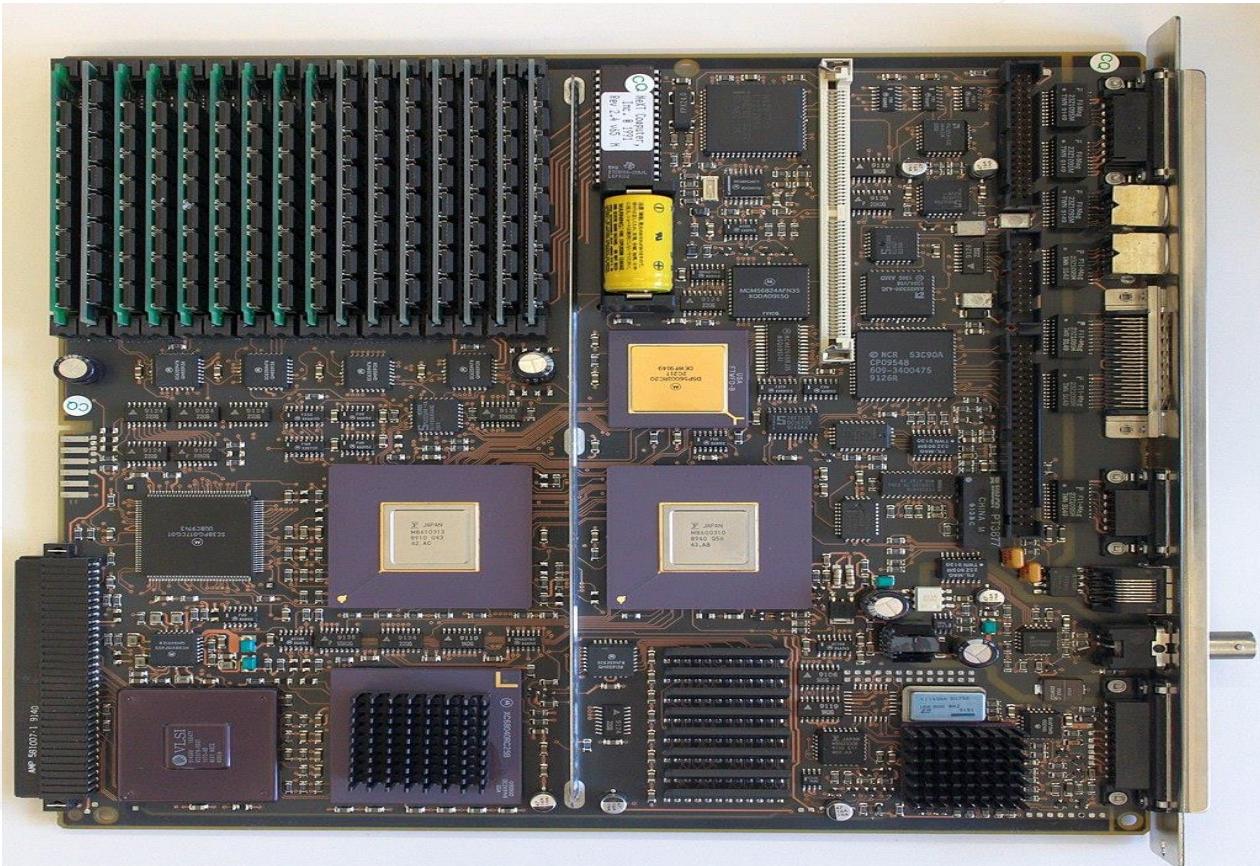
- **Step 1:** - I/O device will send a DMA request to DMA controller to perform an i/o operation.
- **Step 2:** - DMA controller will set interrupt and bus request line to 1.
- **Step 3:** - CPU using address bus will select device and registers and then will initiate i/o address register(location), counter (no of words)
- **Step 4:** - CPU will put on the bus grant line to tell DMA controller, now you are the master of system buses.
- **Step 5:** - now DMA controller will put 1 in DMA acknowledgement and using read/write control lines and address line will perform i/o directly between memory and device.



## Mode of transfer: -

- **Burst mode:** - when entire i/o transfer is completed and then control comes back to CPU then it is called burst mode transfer. i.e. with high speed device like magnetic disk.
- **Cycle stealing mode:** - When CPU executes an instruction then normally there could be following phases.
  - **IF** – Instruction Fetch
  - **ID** – Instruction Decode
  - **OF** – Operand fetch
  - **IX** – Instruction execute
  - **WB** – write back or store result
- Normally in ID and IE phases CPU don't require system buses and if only in that time control is given to DMA controller then it is called cycle stealing.

- Many hardware systems use DMA, including disk drive controllers, graphics cards, network cards and sound cards.
- In the original IBM PC (and the follow-up PC/XT), there was only one Intel 8237 DMA controller capable of providing four DMA channels (numbered 0–3).



- Motherboard of a NeXTcube computer (1990).
- The two large integrated circuits below the middle of the image are the DMA controller (l.) and - unusual - an extra dedicated DMA controller (r.) for the magneto-optical disc used instead of a hard disk drive in the first series of this computer model.

**Q** The size of the data count register of a DMA controller is 16 bits. The processor needs to transfer a file of 29,154 kilobytes from disk to main memory. The memory is byte addressable. The minimum number of times the DMA controller needs to get the control of the system bus from the processor to transfer the file from the disk to main memory is \_\_\_\_\_ (GATE-2016) (2 Marks)

**Q** A hard disk with a transfer rate of 10 Mbytes/ second is constantly transferring data to memory using DMA. The processor runs at 600 MHz, and takes 300 and 900 clock cycles to initiate and complete DMA transfer respectively. If the size of the transfer is 20 Kbytes, what is the percentage of processor time consumed for the transfer operation? **(GATE-2004) (2 Marks)**

**Q** On a non-pipelined sequential processor, a program segment, which is a part of the interrupt service routine, is given to transfer 500 bytes from an I/O device to memory.

Initialize the address register

Initialize the count to 500

LOOP: Load a byte from device

Store in memory at address given by address register

Increment the address register

Decrement the count

If count != 0 go to LOOP

Assume that each statement in this program is equivalent to machine instruction which takes one clock cycle to execute if it is a non-load/store instruction. The load-store instructions take two clock cycles to execute. The designer of the system also has an alternate approach of using DMA controller to implement the same transfer. The DMA controller requires 20 clock cycles for initialization and other overheads. Each DMA transfer cycle takes two clock cycles to transfer one byte of data from the device to the memory. What is the approximate speedup when the DMA controller-based design is used in place of the interrupt driven program-based input-output?

**(GATE-2011) (2 Marks)**

(A) 3.4

(B) 4.4

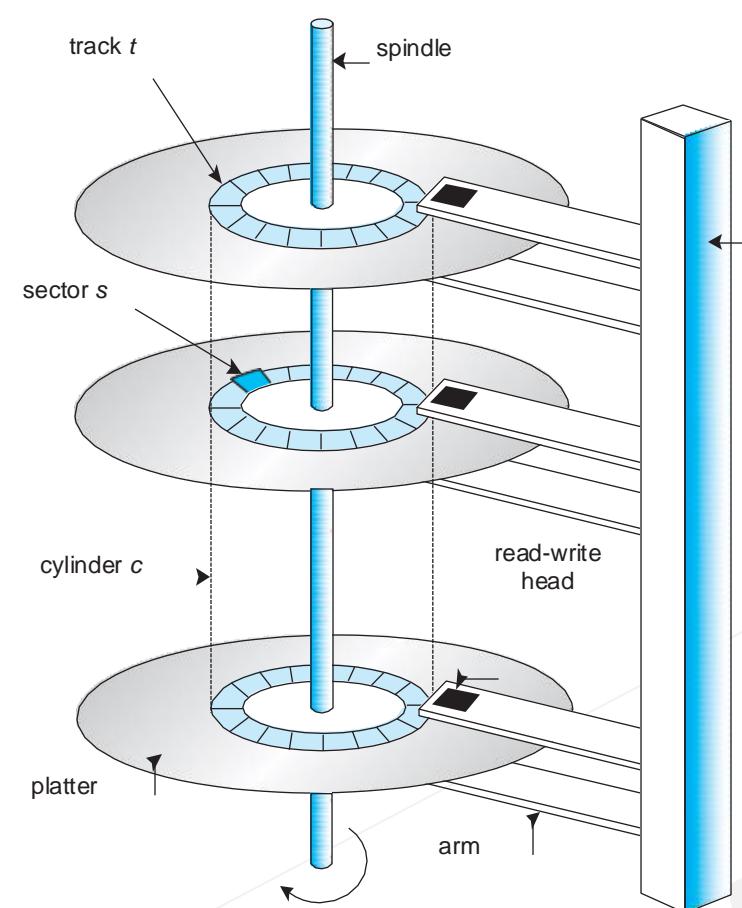
(C) 5.1

(D) 6.7

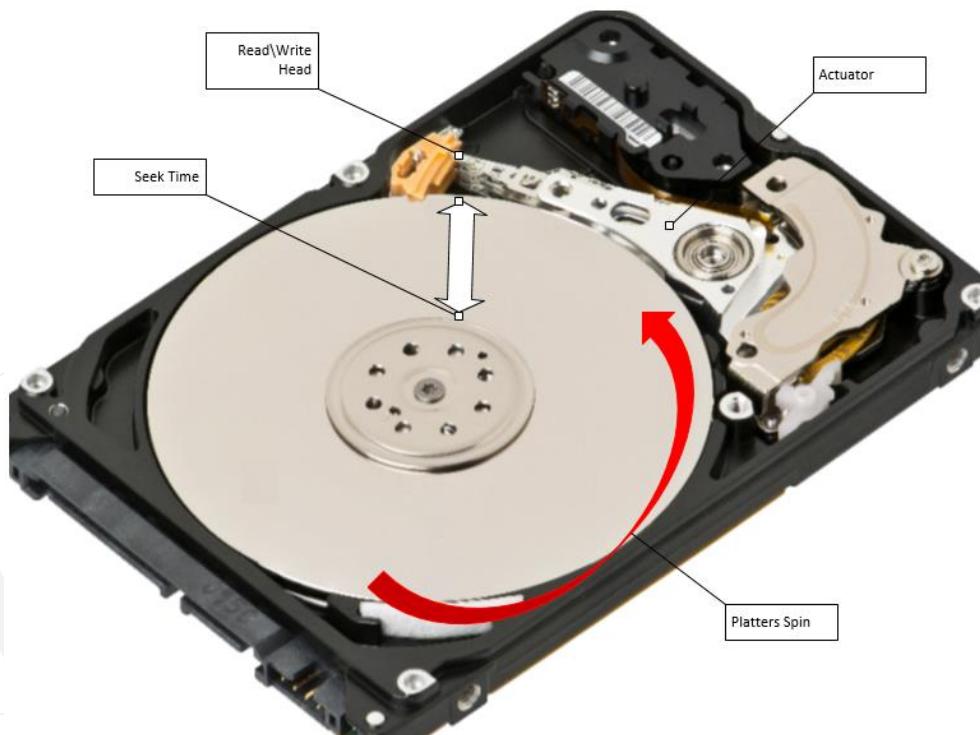
**Q** Consider a computer system with DMA support. The DMA module is transferring one 8-bit character in one CPU cycle from a device to memory through cycle stealing at regular intervals. Consider a 2 MHz processor. If 0.5% processor cycles are used for DMA, the data transfer rate of the device is \_\_\_\_\_ bits per second.(GATE 2021) (1 MARKS)

**Q Which one of the following facilitates transfer of bulk data from hard disk to main memory with the highest throughput?(GATE 2022) (1 MARKS)**

- (A) DMA based I/O transfer
- (B) Interrupt driven I/O transfer
- (C) Polling based I/O transfer
- (D) Programmed I/O transfer

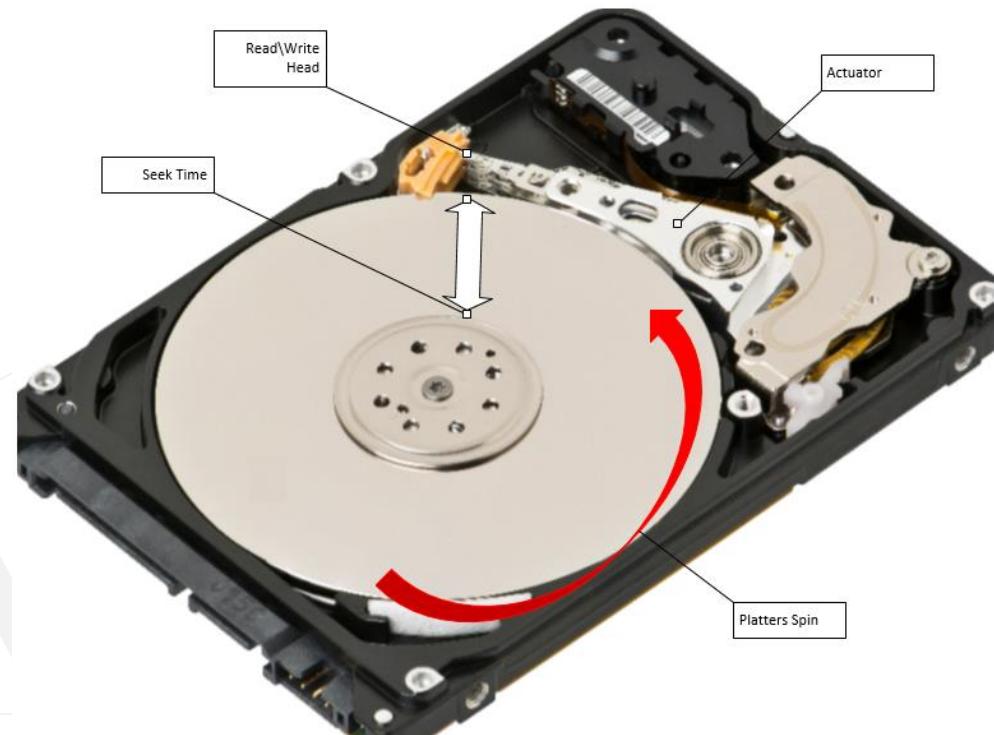


- **Seek Time:** - It is a time taken by Read/Write header to reach the correct track. (Always given in question)
- **Rotational Latency:** - It is the time taken by read/Write header during the wait for the correct sector. In general, it's a random value, so far average analysis, we consider the time taken by disk to complete half rotation.
- **Transfer Time:** - it is the time taken by read/write header either to read or write on a disk. In general, we assume that in 1 complete rotation, header can read/write the either track.



$$\text{Total Transfer Time} = \text{Seek Time} + \text{Rotational Latency} + \text{Transfer Time}$$

- Total time will be = (File Size/Track Size) \*time taken to complete one revolution.



**Q** Consider a disk where there are 512 tracks, each track is capable of holding 128 sector and each sector holds 256 bytes, find the capacity of the track and disk and number of bits required to reach correct track, sector and disk.

**Q** Consider a disk pack with 16 surfaces, 128 tracks per surface and 256 sectors per track. 512 bytes of data are stored in a bit serial manner in a sector. The capacity of the disk pack and the number of bits required to specify a particular sector in the disk are respectively **(GATE-2006) (2 Marks)**

(A) 256 Mbyte, 19 bits

(B) 256 Mbyte, 28 bits

(C) 512 Mbyte, 20 bits

(D) 64 Gbyte, 28 bit

**Q** consider a disk where each sector contains 512 bytes and there are 400 sectors per track and 1000 tracks on the disk. If disk is rotating at speed of 1500 RPM, find the total time required to transfer file of size 1 MB. Suppose seek time is 4ms?

**Q** Consider a system with 8 sector per track and 512 bytes per sector. Assume that disk rotates at 3000 rpm and average seek time is 15ms standard. Find total time required to transfer a file which requires 8 sectors to be stored.

- a)** Assume contiguous allocation
- b)** Assume Non-contiguous allocation

**Q** Consider a typical disk that rotates at 15000 rotations per minute (RPM) and has a transfer rate of  $50 \times 10^6$  bytes/sec. If the average seek time of the disk is twice the average rotational delay and the controller's transfer time is 10 times the disk transfer time, the average time (in milliseconds) to read or write a 512 byte sector of the disk is \_\_\_\_\_ (GATE-2015) (2 Marks)

**Q** Consider a disk pack with a seek time of 4 milliseconds and rotational speed of 10000 rotations per minute (RPM). It has 600 sectors per track and each sector can store 512 bytes of data. Consider a file stored in the disk. The file contains 2000 sectors. Assume that every sector access necessitates a seek, and the average rotational latency for accessing each sector is half of the time for one complete rotation. The total time (in milliseconds) needed to read the entire file is \_\_\_\_\_. (GATE-2015) (1 Marks)

**Q** An application loads 100 libraries at start-up. Loading each library requires exactly one disk access. The seek time of the disk to a random location is given as 10 milli second. Rotational speed of disk is 6000 rpm. If all 100 libraries are loaded from random locations on the disk, how long does it take to load all libraries? (The time to transfer data from the disk block once the head has been positioned at the start of the block may be neglected) (GATE-2011) (2 Marks)

**Q** For a magnetic disk with concentric circular tracks, the seek latency is not linearly proportional to the seek distance due to **(GATE-2008) (2 Marks)**

- (A) non-uniform distribution of requests
- (B) arm starting and stopping inertia
- (C) higher capacity of tracks on the periphery of the platter
- (D) use of unfair arm scheduling policies

**Q** If the disk is rotating at 360 rpm, determine the effective data transfer rate which is defined as the number of bytes transferred per second between disk and memory. (track size = 512bytes)  
**(GATE-1995) (2 Marks)**

**Q** A certain moving arm disk storage, with one head, has the following specifications:

Number of tracks/recording surface = 200

Disk rotation speed = 2400 rpm

Track storage capacity = 62,500 bits

The average latency of this device is P ms and the data transfer rate is Q bits/sec. Write the values of P and Q. **(GATE-1993) (2 Marks)**

**Q**A hard disk system has the following parameters: **(GATE-2007) (2 Marks)**

Number of tracks = 500

Number of sectors / track = 100

Number of bytes / sector = 500

Time taken by the head to move from one track to adjacent track = 1 ms, Rotation speed = 600 rpm. What is the average time taken for transferring 250 bytes from the disk?

**(A)** 300.5 ms

**(B)** 255.5 ms

**(C)** 255.0 ms

**(D)** 300.0 ms

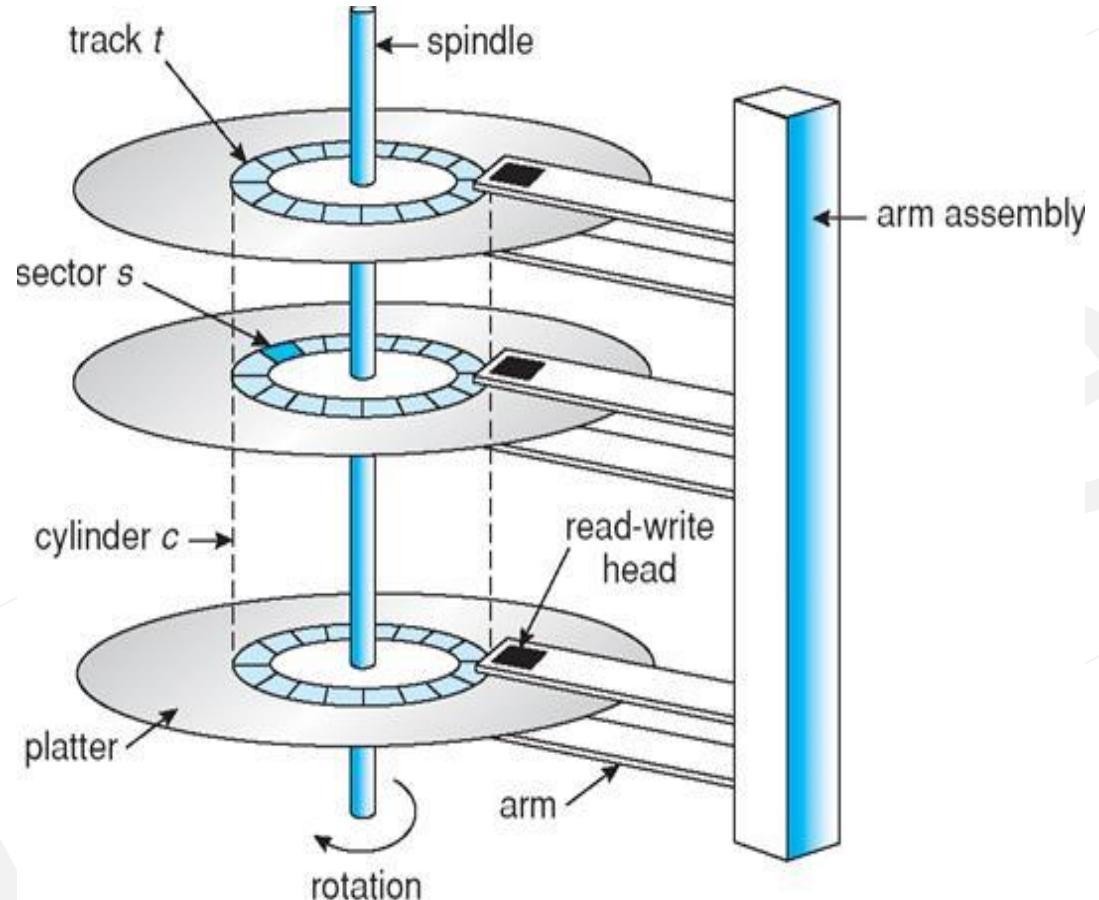
Q A hard disk has 63 sectors per track, 10 platters each with 2 recording surfaces and 1000 cylinders. The address of a sector is given as a triple  $(c, h, s)$ , where  $c$  is the cylinder number,  $h$  is the surface number and  $s$  is the sector number. Thus, the 0th sector is addressed as  $(0, 0, 0)$ , the 1st sector as  $(0, 0, 1)$ , and so on. The address  $<400,16,29>$  corresponds to sector number: (GATE-2009) (2 Marks)

(A) 505035

(B) 505036

(C) 505037

(D) 505038



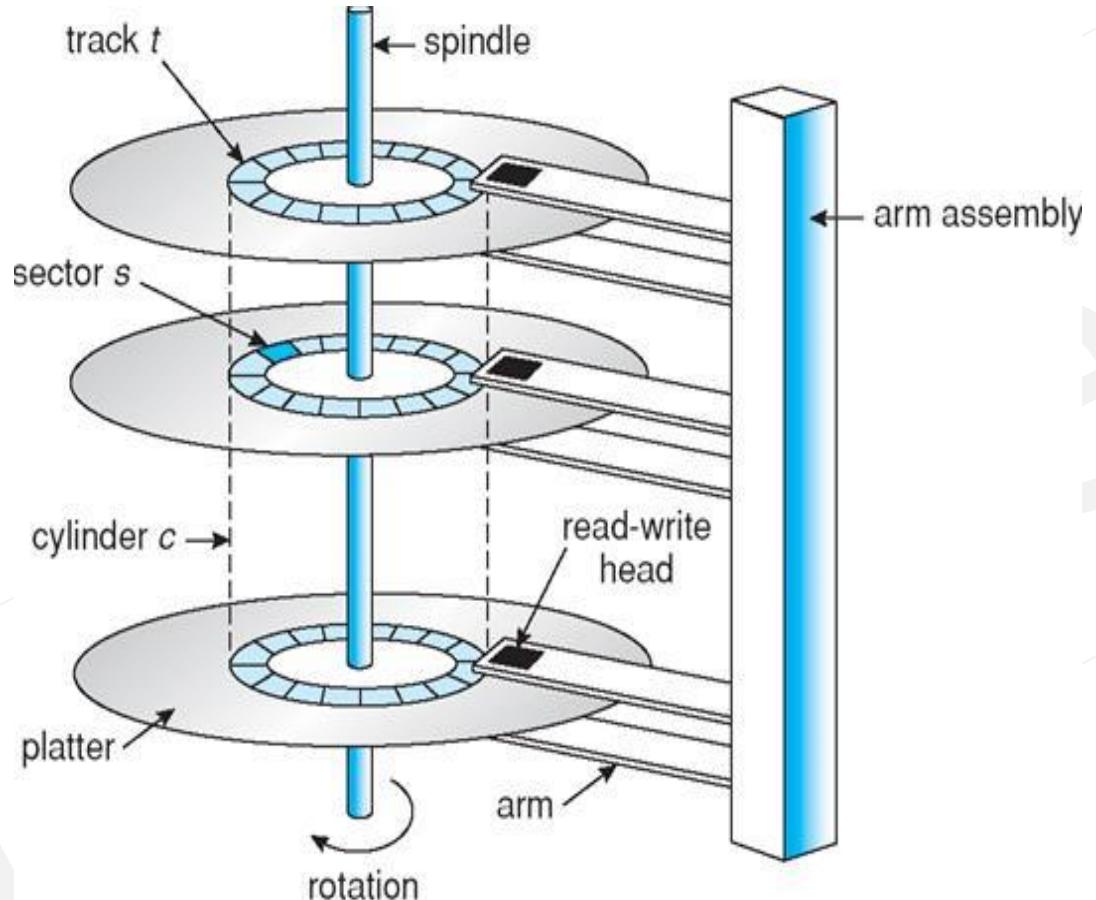
Q A hard disk has 63 sectors per track, 10 platters each with 2 recording surfaces and 1000 cylinders. The address of a sector is given as a triple  $(c, h, s)$ , where  $c$  is the cylinder number,  $h$  is the surface number and  $s$  is the sector number. Thus, the 0th sector is addressed as  $(0, 0, 0)$ , the 1st sector as  $(0, 0, 1)$ , and so on. The address  $<400,16,29>$  corresponds to sector number: (GATE-2009) (2 Marks)

(A) 505035

(B) 505036

(C) 505037

(D) 505038



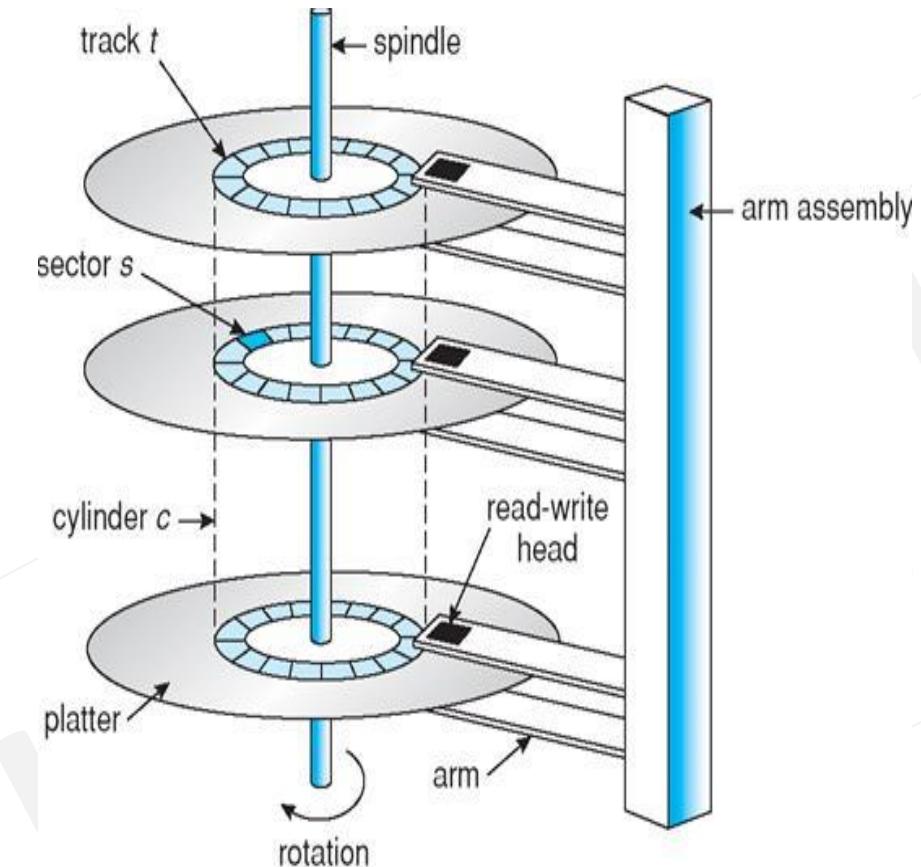
**Q** Consider a hard disk with 16 recording surfaces (0-15) having 16384 cylinders (0-16383) and each cylinder contains 64 sectors (0-63). Data storage capacity in each sector is 512 bytes. Data are organized cylinder-wise and the addressing format is <cylinder no., surface no., sector no.>. A file of size 42797 KB is stored in the disk and the starting disk location of the file is <1200, 9, 40>. What is the cylinder number of the last sector of the file, if it is stored in a contiguous manner? **(GATE-2013) (1 Marks )**

**(A)** 1281

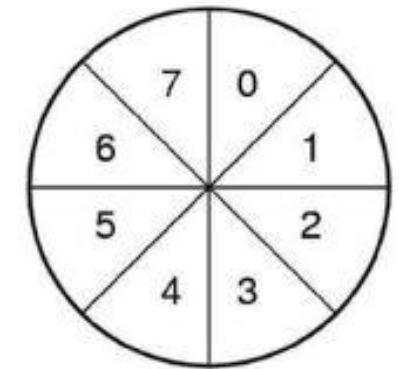
**(B)** 1282

**(C)** 1283

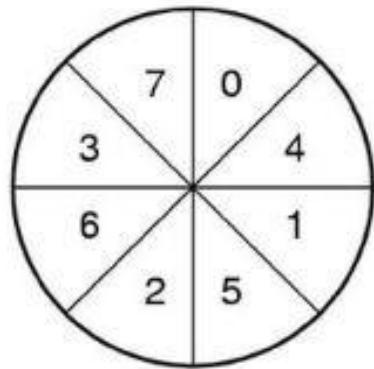
**(D)** 1284



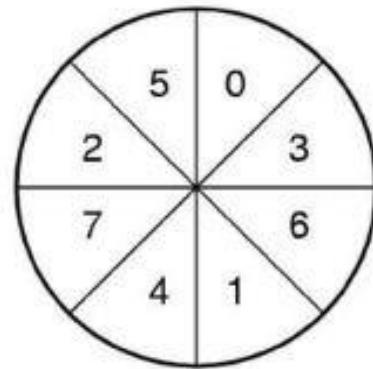
- **Single interleaving:** - in 2 rotation we read 1 track
- **Double interleaving:** - in 2.75 rotation we read 1 track



(a)



(b)

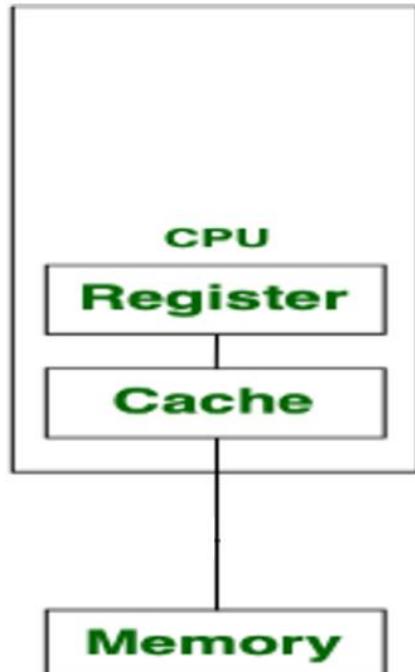


(c)

- No interleaving
- Single interleaving
- Double interleaving

# Uniprocessing

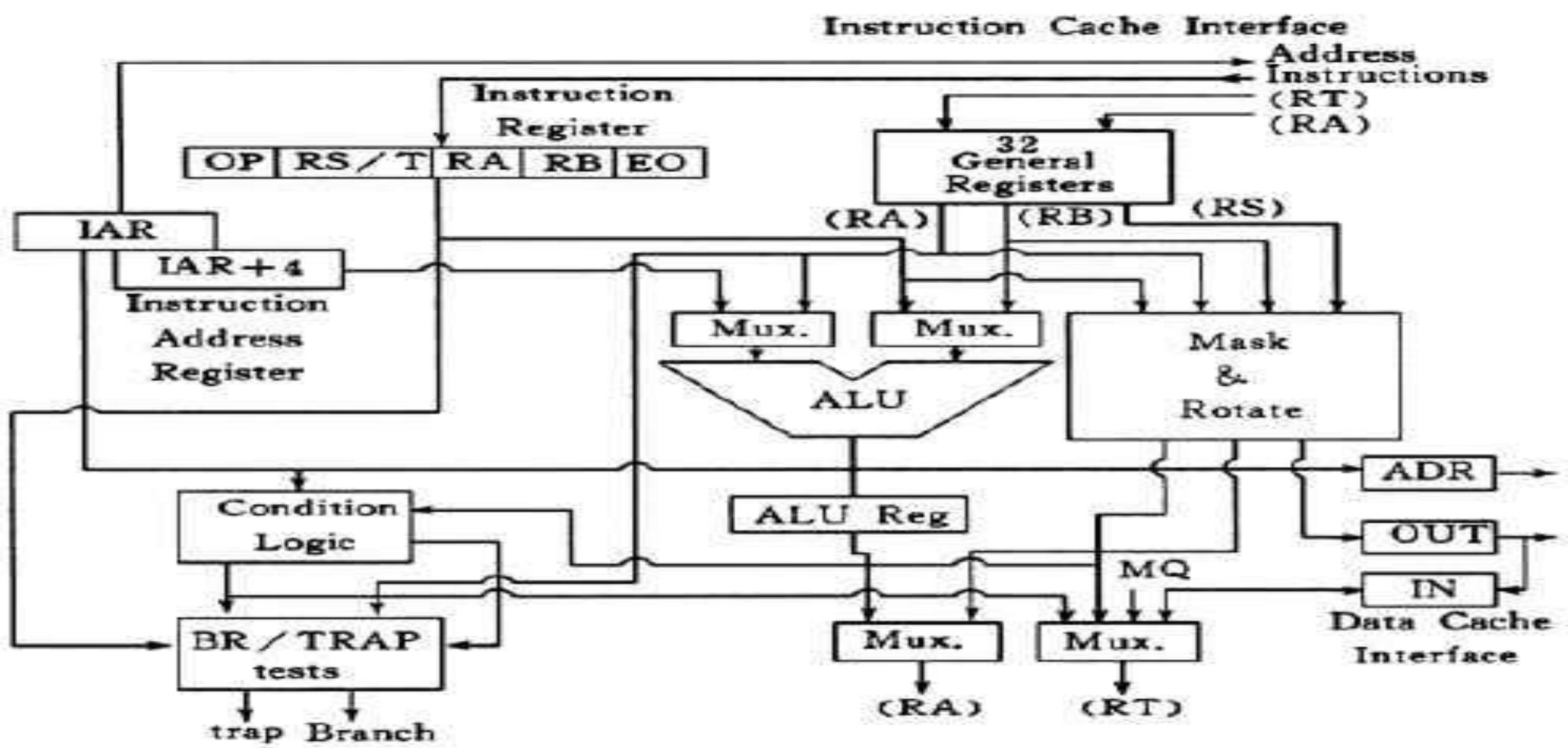
- When we actually execute an instruction, it is executed into number of phases, like
  - Instruction Fetch, Instruction Decode, Instruction executes, Instruction Store
- If the system has only one processor then at most one instruction can be executed at a time.
- Where when one phase is completed then only we start with next phase.



Instruction  
Fetch  
Decode  
Execute  
Write  
Clock

		1			2			
1	2	3	4	5	6	7	8	

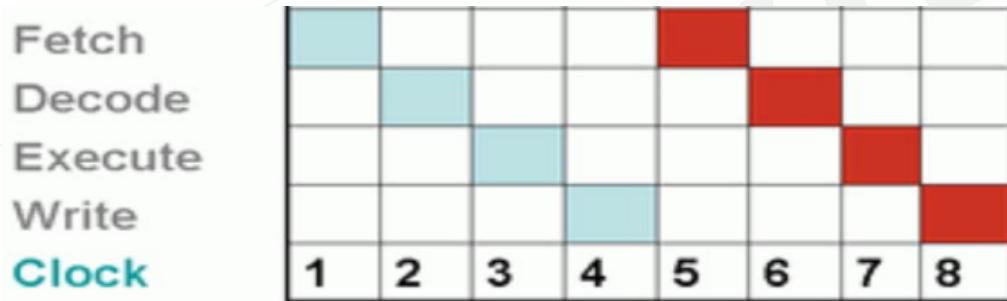
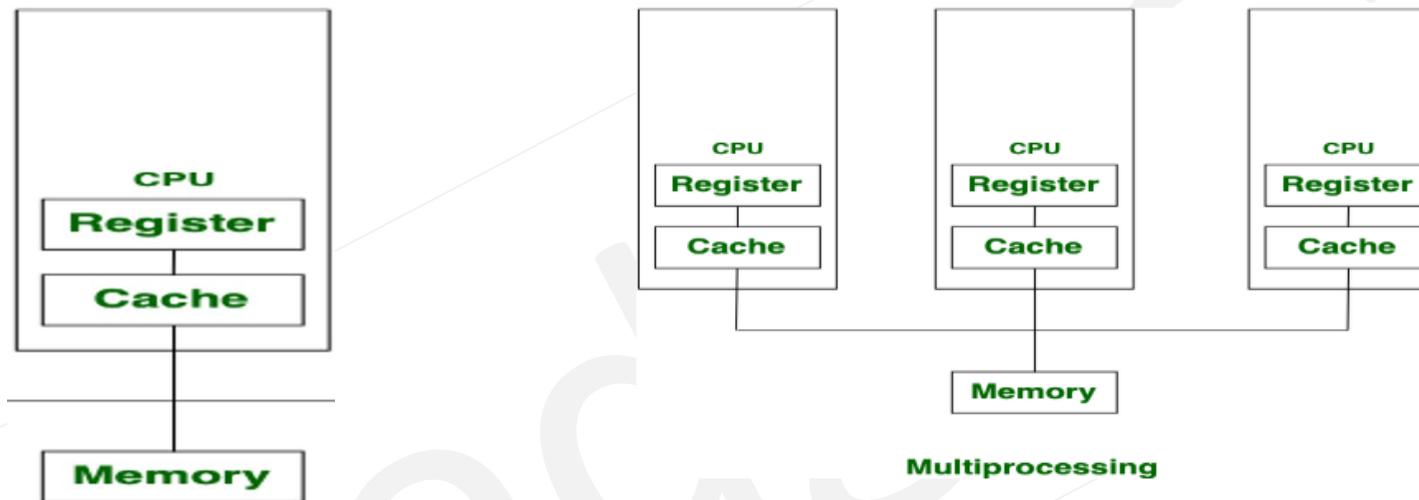
Non-Pipelined



**Fig. 3-2 IBM 801 Architecture**

# Uniprocessing vs Multiprocessing

- If we really want to execute multiple instruction together or concurrently then we must have multiple processors.

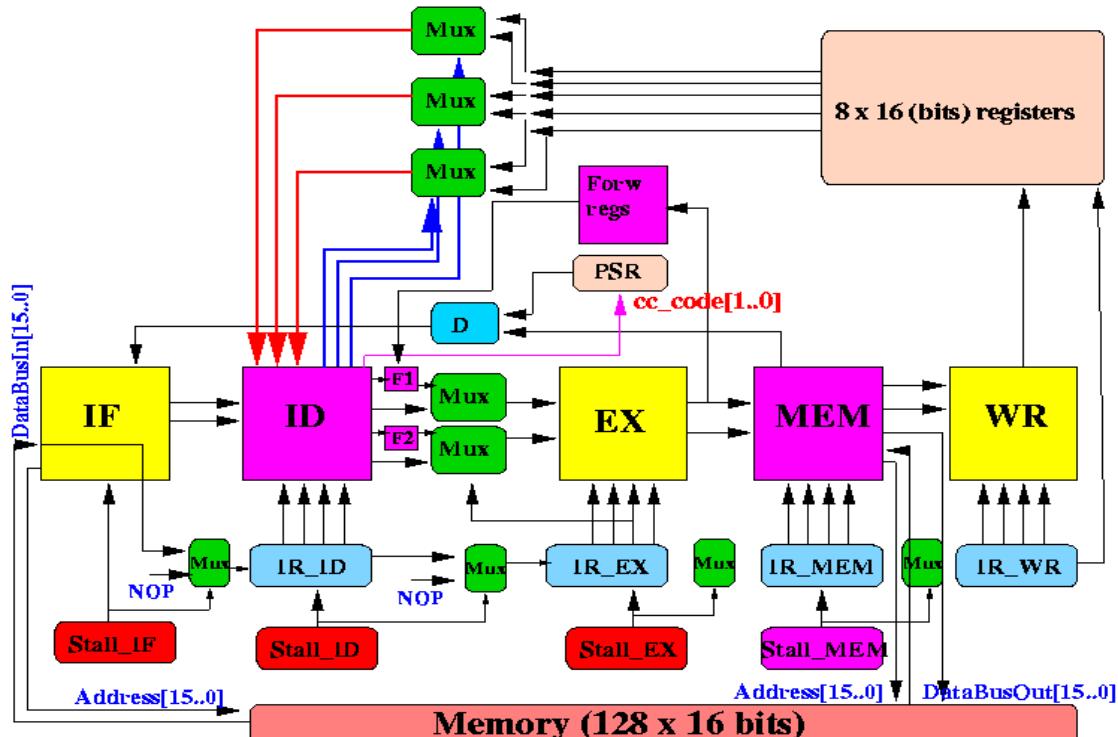


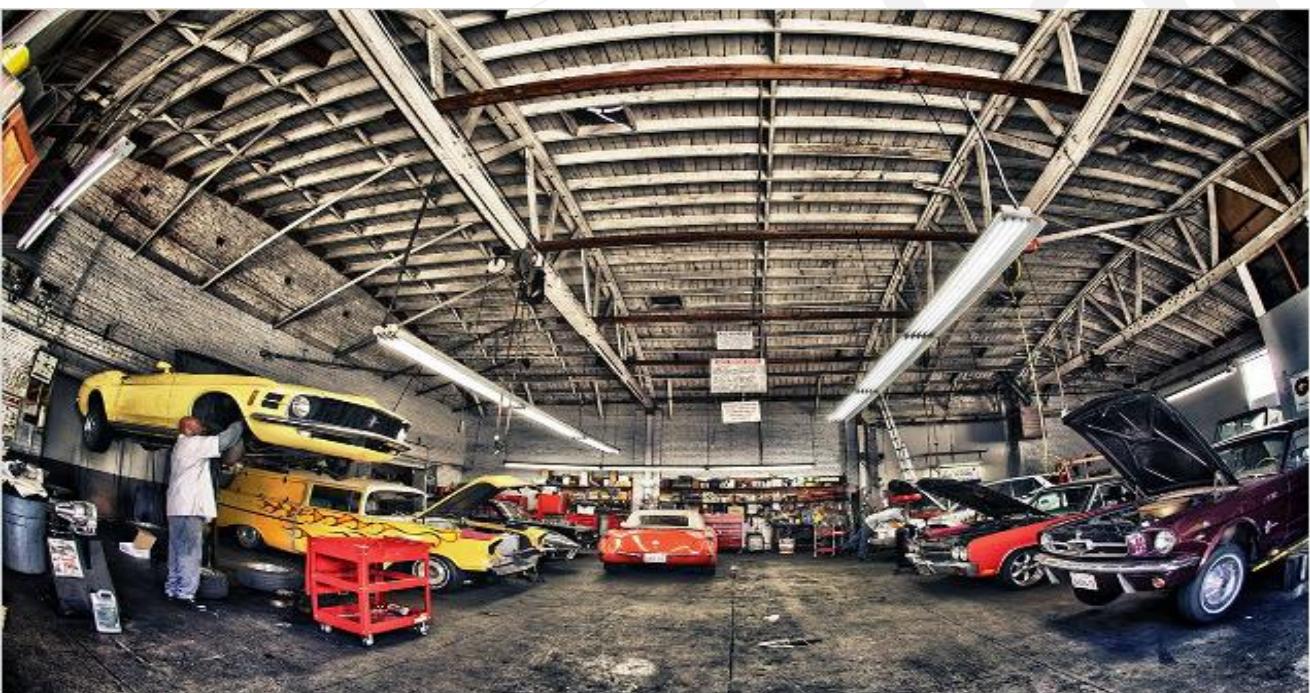
**Processor<sub>1</sub>**

**Processor<sub>2</sub>**

# Pipelining

- Pipelining is a phenomena or method using which, we will able to run more than one instruction at the same time, on a single processor. The idea if make special processor (pipelined processor), where the circuit of every phase is different and buffers are placed between stages. Then we can start executing next instruction before completing all the phases of the current executing one. This idea is called pipelining.





Fetch								
Decode								
Execute								
Write								
Clock	1	2	3	4	5	6	7	8

## Uniprocessing

Fetch								
Decode								
Execute								
Write								
Fetch								
Decode								
Execute								
Write								
Clock	1	2	3	4				

**Processor<sub>1</sub>**

**Processor<sub>2</sub>**

## Multiprocessing

Instruction	1	2						
Fetch								
Decode								
Execute								
Write								
Clock	1	2	3	4	5			

## Pipelining

**Q** Consider a system where clock is triggering at a speed of 1MHz (1 clock = 1  $\mu$ s). In a pipelined processor there are 4 stages and each stage take only 1 clock, if a program has 10 instruction then it will take what time?

- On a non-pipelined processor

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
IF																
ID																
EX																
WB																

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40												
IF	I <sub>1</sub>					I <sub>2</sub>							I <sub>3</sub>								I <sub>4</sub>					I <sub>5</sub>					I <sub>6</sub>					I <sub>7</sub>					I <sub>8</sub>				I <sub>9</sub>					I <sub>10</sub>		
ID		I <sub>1</sub>					I <sub>2</sub>				I <sub>3</sub>				I <sub>4</sub>				I <sub>5</sub>				I <sub>6</sub>				I <sub>7</sub>				I <sub>8</sub>				I <sub>9</sub>					I <sub>10</sub>												
EX			I <sub>1</sub>				I <sub>2</sub>				I <sub>3</sub>				I <sub>4</sub>				I <sub>5</sub>				I <sub>6</sub>				I <sub>7</sub>				I <sub>8</sub>				I <sub>9</sub>				I <sub>10</sub>													
WB				I <sub>1</sub>				I <sub>2</sub>				I <sub>3</sub>				I <sub>4</sub>				I <sub>5</sub>				I <sub>6</sub>				I <sub>7</sub>				I <sub>8</sub>				I <sub>9</sub>				I <sub>10</sub>												

- If all instructions are identical (time taken for specific phase is same for all instruction)
- Time without pipeline ( $T_{wp}$ ) =
  - (sum of clocks for each phase of one instruction) \*(no of instruction)\*time of one clock
- If each phase requires same clock usually one (as we set the frequency in such a way)  
= (no of phases \* no of instruction) \* time of one clock

**Q** Consider a system where clock is triggering at a speed of 1MHz (1 clock = 1  $\mu$ s). In a pipelined processor there are 4 stages and each stage take only 1 clock, if a program has 10 instruction then it will take what time?

- On a pipelined processor

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
IF																
ID																
EX																
WB																

	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>	<b>6</b>	<b>7</b>	<b>8</b>	<b>9</b>	<b>10</b>	<b>11</b>	<b>12</b>	<b>13</b>
<b>IF</b>	I <sub>1</sub>	I <sub>2</sub>	I <sub>3</sub>	I <sub>4</sub>	I <sub>5</sub>	I <sub>6</sub>	I <sub>7</sub>	I <sub>8</sub>	I <sub>9</sub>	I <sub>10</sub>			
<b>ID</b>		I <sub>1</sub>	I <sub>2</sub>	I <sub>3</sub>	I <sub>4</sub>	I <sub>5</sub>	I <sub>6</sub>	I <sub>7</sub>	I <sub>8</sub>	I <sub>9</sub>	I <sub>10</sub>		
<b>EX</b>			I <sub>1</sub>	I <sub>2</sub>	I <sub>3</sub>	I <sub>4</sub>	I <sub>5</sub>	I <sub>6</sub>	I <sub>7</sub>	I <sub>8</sub>	I <sub>9</sub>	I <sub>10</sub>	
<b>WB</b>				I <sub>1</sub>	I <sub>2</sub>	I <sub>3</sub>	I <sub>4</sub>	I <sub>5</sub>	I <sub>6</sub>	I <sub>7</sub>	I <sub>8</sub>	I <sub>9</sub>	I <sub>10</sub>

- If all instructions are identical (time taken for specific phase is same for all instruction).
- If each phase requires same clock usually one (as we set the frequency in such a way).
- Time with pipeline ( $T_p$ ) = ((no of phase) + (no of instruction - 1)) \*time of one clock

- Speed up = (Time without pipeline ( $T_{wp}$ )) / (Time with pipeline ( $T_p$ )) =
- Max Speed up = no of stages = (In this case 4)
- Efficiency = (speed up/max speed up) \* 100 =

N = 10, 100, 1000

**Q** Consider a system where we have 'm' stages and program contains 'n' instruction such that  $m << n$ , then find speed up?

**Q Consider 5 instruction with following clock requirement?**

	F	D	E	WB
I <sub>1</sub>	1	2	1	1
I <sub>2</sub>	1	2	2	1
I <sub>3</sub>	2	1	3	2
I <sub>4</sub>	1	3	2	1
I <sub>5</sub>	1	2	1	2

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
I <sub>1</sub>																		
I <sub>2</sub>																		
I <sub>3</sub>																		
I <sub>4</sub>																		
I <sub>5</sub>																		

Q Consider a 4 stage pipeline processor. The number of cycles needed by the four instructions  $I_1, I_2, I_3, I_4$  in stages  $S_1, S_2, S_3, S_4$  is shown below:  
**(Gate-2009) (2 Marks)**

For ( $I = 1; I \leq 2; I++$ )

{

$I_1$

$I_2$

$I_3$

$I_4$

	F	D	E	WB
$I_1$	2	1	1	1
$I_2$	1	3	2	2
$I_3$	2	1	1	3
$I_4$	1	2	2	2

}

a) 16

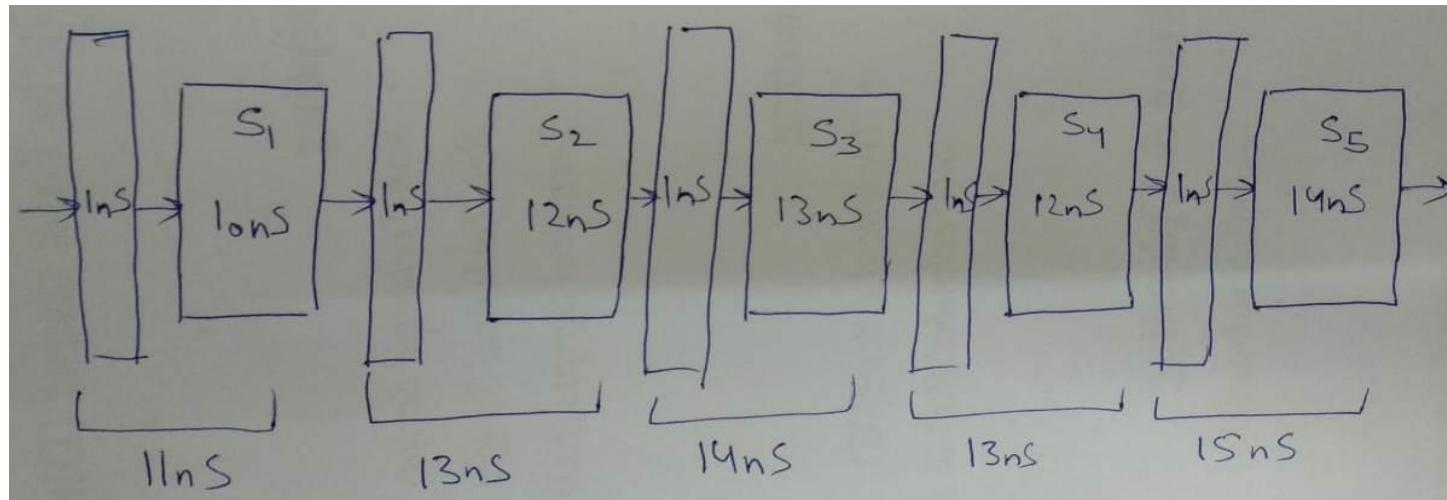
b) 23

c) 28

d) 30

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
$I_1$																											
$I_2$																											
$I_3$																											
$I_4$																											
$I_1$																											
$I_2$																											
$I_3$																											
$I_4$																											

**Q** After considering this diagram what must be the frequency of the processor to ensure that work of every stage will complete in 1 clock stage wise?



	$S_1$	$S_2$	$S_3$	$S_4$	$S_5$
1ns					
2ns					
3ns					
5ns					
15ns					

- We understand that different stages in a pipeline may have different delays, it also depends on the type of instruction that how much time a particular stage will take for a specific instruction.
- If we increase the time of a clock to the time taken by the slowest stage of the pipeline, then each instruction takes one clock then with pipe-line processor in long run, we achieve
  - CPI(Clock Per Instruction) = 1

**Q** Consider the following processors (ns stands for nanoseconds). Assume that the pipeline registers have zero latency. **(Gate-2014) (2 Marks)**

**P<sub>1</sub>:** Four-stage pipeline with stage latencies 1 ns, 2 ns, 2 ns, 1 ns.

**P<sub>2</sub>:** Four-stage pipeline with stage latencies 1 ns, 1.5 ns, 1.5 ns, 1.5 ns.

**P<sub>3</sub>:** Five-stage pipeline with stage latencies 0.5 ns, 1 ns, 1 ns, 0.6 ns, 1 ns.

**P<sub>4</sub>:** Five-stage pipeline with stage latencies 0.5 ns, 0.5 ns, 1 ns, 1 ns, 1.1 ns.

Which processor has the highest peak clock frequency?

- (A) P<sub>1</sub>
- (B) P<sub>2</sub>
- (C) P<sub>3</sub>
- (D) P<sub>4</sub>

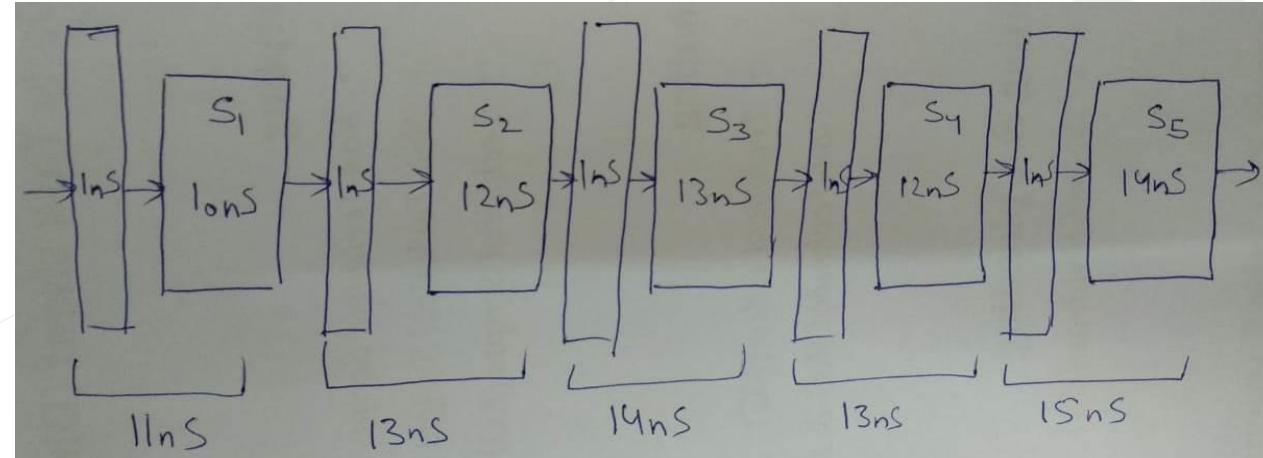
**Q** Consider the time  $T_1$  is taken for a single instruction on a pipelined processor and time  $T_2$  is taken for a single instruction on a non-pipeline processor?

a)  $T_1 = T_2$

b)  $T_1 \leq T_2$

c)  $T_1 \geq T_2$

d) NONE



**Q** A 4-stage pipeline has the stage delays as 150, 120, 160 and 140 nanoseconds respectively. Registers that are used between the stages have a delay of 5 nanoseconds each. Assuming constant clocking rate, the total time taken to process 1000 data items on this pipeline will be  
**(Gate-2004) (2 Marks)**

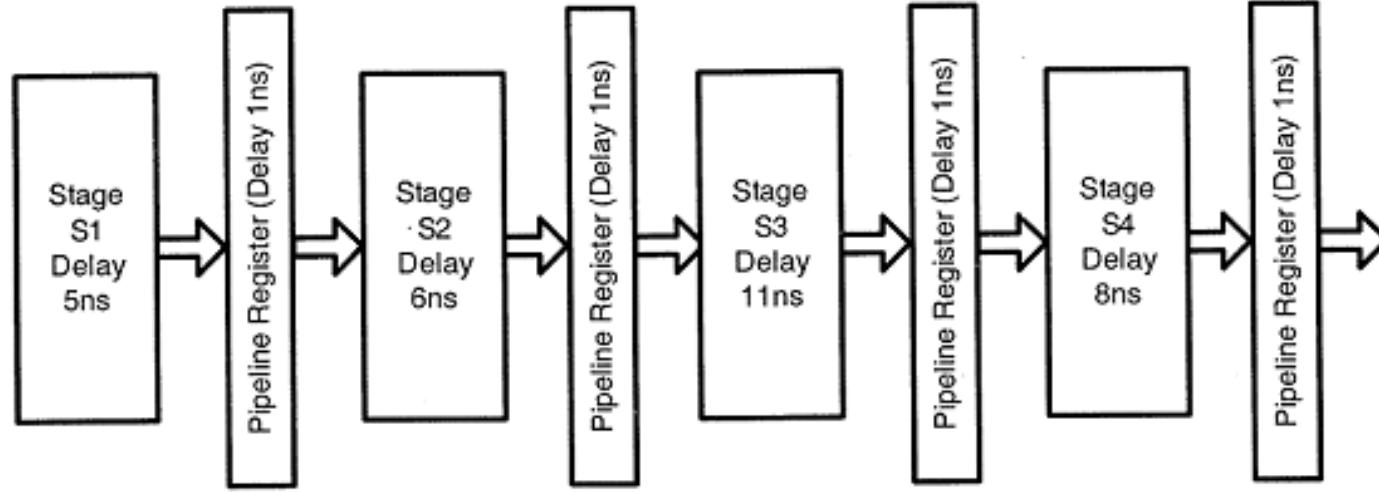
**(A)** 120.4 microseconds

**(B)** 160.5 microseconds

**(C)** 165.5 microseconds

**(D)** 590.0 microseconds

**Q** Consider an instruction pipeline with four stages ( $S_1, S_2, S_3$  and  $S_4$ ) each with combinational circuit only. The pipeline registers are required between each stage and at the end of the last stage. Delays for the stages and for the pipeline registers are as given in the figure:



What is the approximate speed up of the pipeline in steady state under ideal conditions when compared to the corresponding non-pipeline implementation? **(Gate-2011) (2 Marks)**

- (A) 4.0
- (B) 2.5
- (C) 1.1
- (D) 3.0

**Q** A non-pipelined single cycle processor operating at 100 MHz is converted into a synchronous pipelined processor with five stages requiring 2.5 nsec, 1.5 nsec, 2 nsec, 1.5 nsec and 2.5 nsec, respectively. The delay of the latches is 0.5 nsec. The speedup of the pipeline processor for a large number of instructions is **(Gate-2008) (2 Marks)**

**(A) 4.5**

**(B) 4.0**

**(C) 3.33**

**(D) 3.0**

**Q** We have two designs  $D_1$  and  $D_2$  for a synchronous pipeline processor.  $D_1$  has 5 pipeline stages with execution times of 3 nsec, 2 nsec, 4 nsec, 2 nsec and 3 nsec while the design  $D_2$  has 8 pipeline stages each with 2 nsec execution time. How much time can be saved using design  $D_2$  over design  $D_1$  for executing 100 instructions? **(Gate-2005) (2 Marks)**

(A) 214 nsec

(B) 202 nsec

(C) 86 nsec

(D) 200 nsec

**Q** Instruction execution in a processor is divided into 5 stages. Instruction Fetch (IF), Instruction Decode (ID), Operand Fetch (OF), Execute (EX), and Write Back (WB). These stages take 5, 4, 20, 10 and 3 nanoseconds (ns) respectively.

A pipelined implementation of the processor requires buffering between each pair of consecutive stages with a delay of 2 ns.

Two pipelined implementations of the processor are contemplated:

(i) a naïve pipeline implementation (NP) with 5 stages and

(ii) an efficient pipeline (EP) where the OF stage is divided into stages  $OF_1$  and  $OF_2$  with execution times of 12 ns and 8 ns respectively.

The speedup (correct to two decimal places) achieved by EP over NP in executing 20 independent instructions with no hazards is \_\_\_\_\_.

**(Gate-2017) (2-marks)**

**Q** The stage delays in a 4-stage pipeline are 800, 500, 400 and 300 picoseconds. The first stage (with delay 800 picoseconds) is replaced with a functionally equivalent design involving two stages with respective delays 600 and 350 picoseconds. The throughput increase of the pipeline is \_\_\_\_\_ percent. **(Gate-2016) (2 Marks)**

**Q** Consider a non-pipelined processor with a clock rate of 2.5 gigahertz and average cycles per instruction of four. The same processor is upgraded to a pipelined processor with five stages; but due to the internal pipeline delay, the clock speed is reduced to 2 gigahertz. Assume that there are no stalls in the pipeline. The speed up achieved in this pipelined processor is \_\_\_\_\_.  
**(Gate-2015)( 2-marks)**

**Q** Consider a 3 GHz (gigahertz) processor with a three-stage pipeline and stage latencies  $\tau_1$ ,  $\tau_2$  and  $\tau_3$  such that  $\tau_1 = (3 \tau_2) / 4 = 2\tau_3$ . If the longest pipeline stage is split into two pipeline stages of equal latency, the new frequency is \_\_\_\_\_ GHz, ignoring delays in the pipeline registers. **(Gate-2016) (2 Marks)**

**Q** Suppose the functions F and G can be computed in 5 and 3 nanoseconds by functional units  $U_F$  and  $U_G$ , respectively. Given two instances of  $U_F$  and two instances of  $U_G$ , it is required to implement the computation  $F(G(X_i))$  for  $1 \leq i \leq 10$ . ignoring all other delays, the minimum time required to complete this computation is \_\_\_\_\_ nanoseconds **(Gate-2016) (2 marks)**

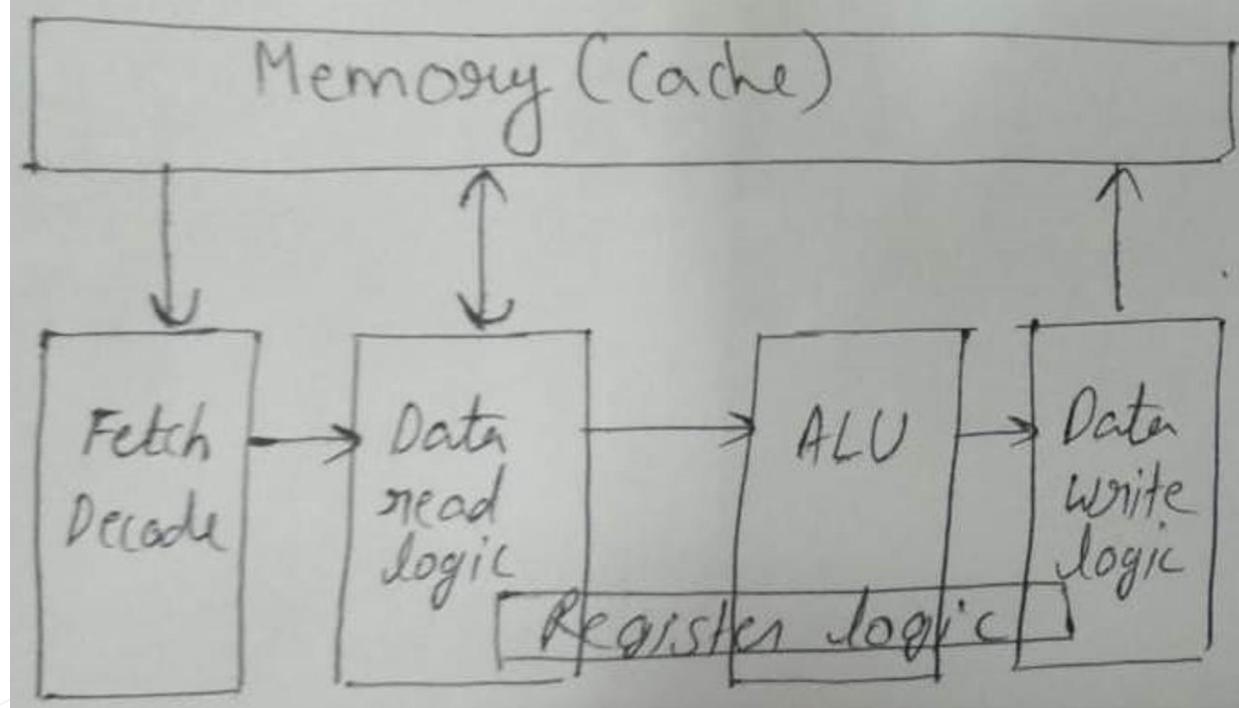
**Q** A five-stage pipeline has stage delays of 150,120,150,160 and 140 nanoseconds. The registers that are used between the pipeline stages have a delay of 5 nanoseconds each. The total time to execute 100 independent instructions on this pipeline, assuming there are no pipeline stalls, is \_\_\_\_\_ nanoseconds. **(GATE 2021) (2 MARKS)**

Consider two processors  $P_1$  and  $P_2$  executing the same instruction set.

Assume that under identical conditions, for the same input, a program running on  $P_2$  takes 25% less time but incurs 20% more CPI (clock cycles per instruction) as compared to the program running on  $P_1$ . If the clock frequency of  $P_1$  is 1GHz, then the clock frequency of  $P_2$  (in GHz) is \_\_\_\_\_.

- Sometimes we cannot execute instructions with full efficiency in a pipeline because of certain dependency or hazards.
  - **Structural hazards**
  - **Control hazards**
  - **Data hazards**

- Even by having 'm' stages in a pipeline processor and assuming that the combinational circuit of every stage is different many times there will be dependence because of external h/w a part from CPU like system bus, memory etc. because of this we have to use stall and the efficiency or speed up decreases.

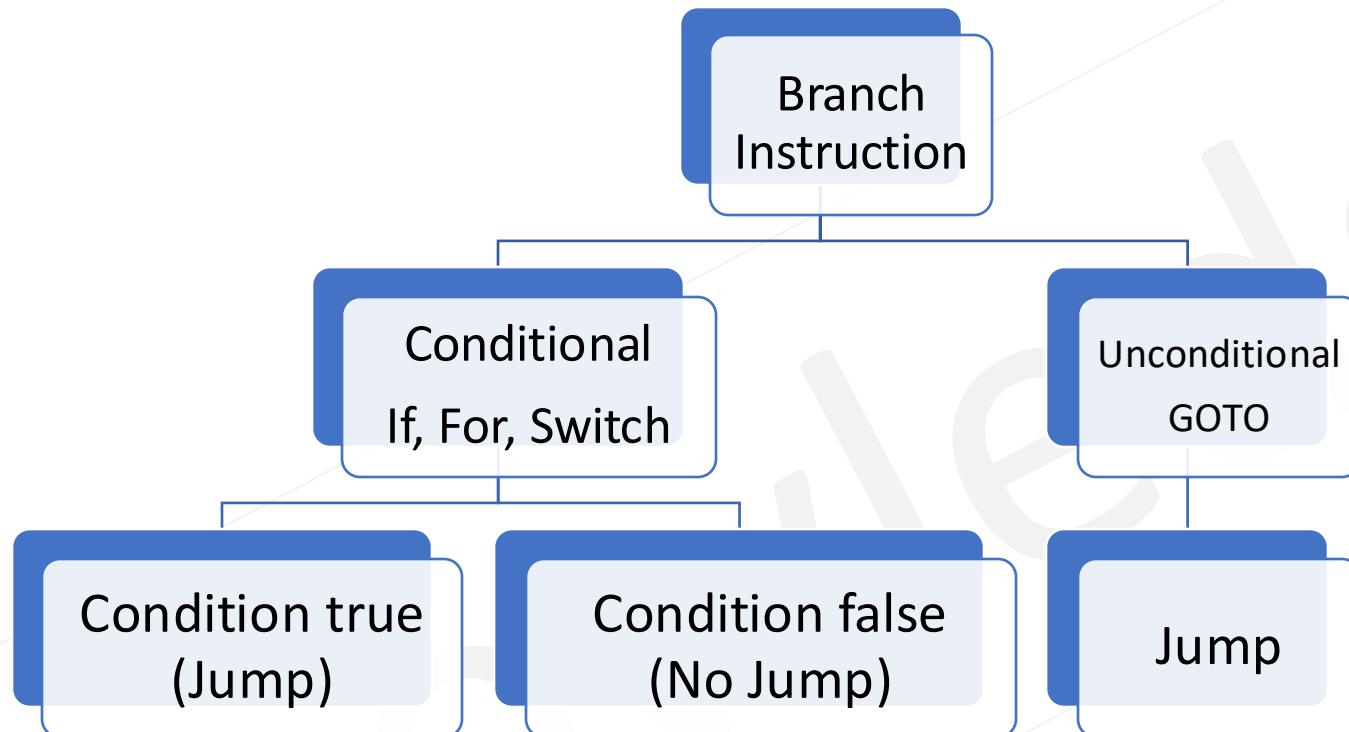


	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>	<b>6</b>	<b>7</b>	<b>8</b>	<b>9</b>	<b>10</b>	<b>11</b>	<b>12</b>
<b>I<sub>1</sub></b>												
<b>I<sub>2</sub></b>												
<b>I<sub>3</sub></b>												
<b>I<sub>4</sub></b>												



- A structural hazard cannot be removed using efficient program. Therefore, one solution could be resource duplication but the cost of implementation will be very high.

- If some instruction  $I_1$  is branch instruction and after executing the entire instruction we understand that there is jump and next instruction to be executed is  $I_{10}$ . This time we have already partially executed  $I_1, I_2, I_3$  which is the problem.



	1	2	3	4	5	6	7	8	9	10	11	12
I <sub>1</sub>	IF	ID	EX	WB								
I <sub>2</sub>		IF	ID	EX	WB							
I <sub>3</sub>			IF	ID	EX	WB						
I <sub>4</sub>				IF	ID	EX						
I <sub>5</sub>					IF	ID						
I <sub>6</sub>												
I <sub>7</sub>							IF	ID	EX	WB		
I <sub>8</sub>								IF	ID	EX	WB	

I<sub>3</sub> -----> I<sub>7</sub>

**Q** Consider a system where 20% instruction are branch instruction and because of them we suffer following stall. Find the CPI or Clock Per Instruction because of these control dependencies.

**1)** Branch Instruction = 1 Stall

	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>	<b>6</b>	<b>7</b>	<b>8</b>
<b>I<sub>1</sub></b>	<b>IF</b>	<b>ID</b>	<b>EX</b>	<b>WB</b>				
<b>I<sub>2</sub></b>			<b>IF</b>	<b>ID</b>	<b>EX</b>	<b>WB</b>		
<b>I<sub>3</sub></b>					<b>IF</b>	<b>ID</b>	<b>EX</b>	<b>WB</b>
<b>I<sub>4</sub></b>								

**Q** Consider a system where 20% instruction are branch instruction and because of them we suffer following stall. Find the CPI or Clock Per Instruction because of these control dependencies.

**2) Branch Instruction = 2 Stall**

	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>	<b>6</b>	<b>7</b>	<b>8</b>
<b>I<sub>1</sub></b>	<b>IF</b>	<b>ID</b>	<b>EX</b>	<b>WB</b>				
<b>I<sub>2</sub></b>				<b>IF</b>	<b>ID</b>	<b>EX</b>	<b>WB</b>	
<b>I<sub>3</sub></b>								
<b>I<sub>4</sub></b>								

**Q** Consider a system where 20% instruction are branch instruction and because of them we suffer following stall. Find the CPI or Clock Per Instruction because of these control dependencies.

**3) Branch Instruction = 3 Stall**

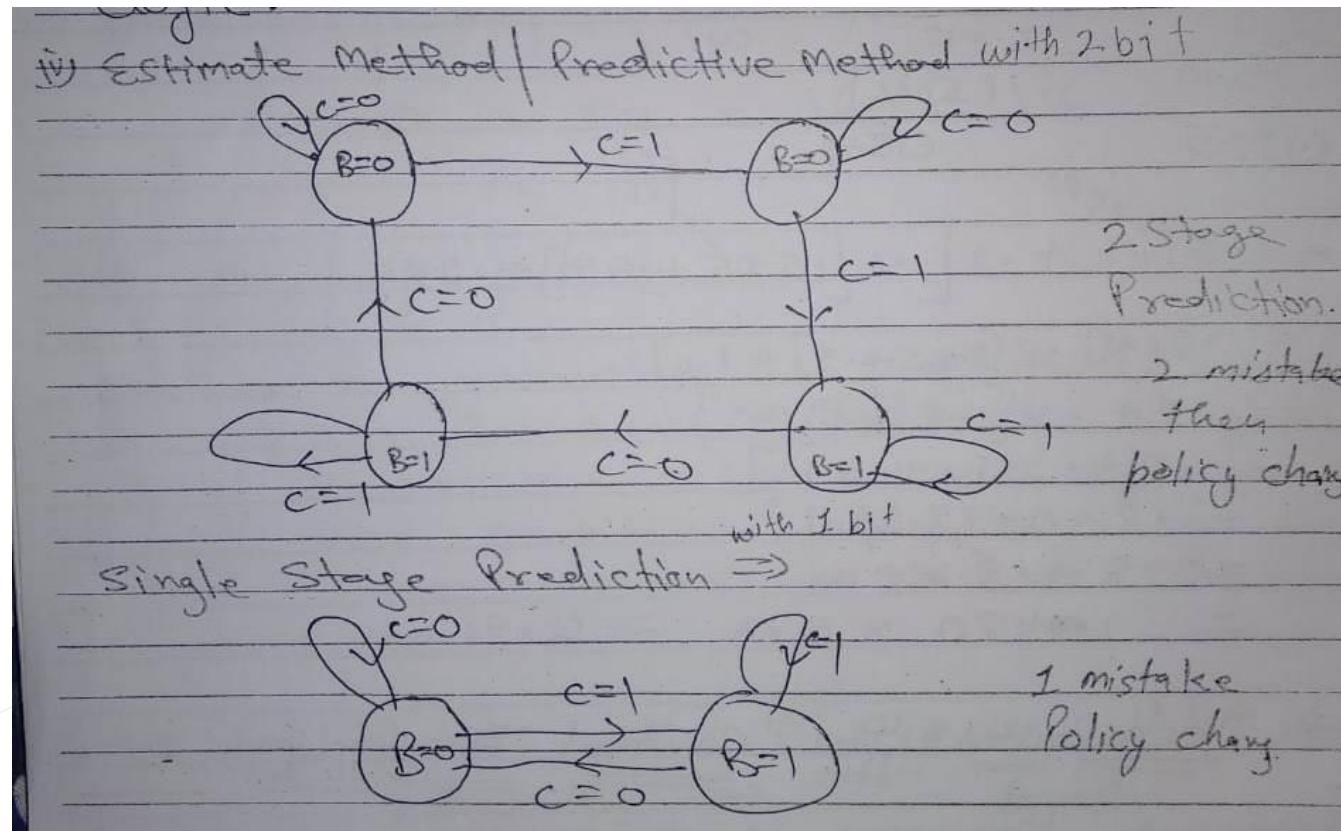
	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>	<b>6</b>	<b>7</b>	<b>8</b>
<b>I<sub>1</sub></b>	<b>IF</b>	<b>ID</b>	<b>EX</b>	<b>WB</b>				
<b>I<sub>2</sub></b>					<b>IF</b>	<b>ID</b>	<b>EX</b>	<b>WB</b>
<b>I<sub>3</sub></b>								
<b>I<sub>4</sub></b>								

**Q** Consider a 6-stage instruction pipeline, where all stages are perfectly balanced. Assume that there is no cycle-time overhead of pipelining. When an application is executing on this 6-stage pipeline, the speedup achieved with respect to non-pipelined execution, if 25% of the instructions incur 2 pipeline stall cycles is **(Gate-2014) (2marks)**

## Solution of control hazards

- **Flushing/Stalling or NOP (no-operation):** -It is the worst-case scenario either we remove the computed data or we don't work at all. Which means compiler don't execute any instruction here.
- **Code rearrangement or delayed load:** - Here with smart compilers we can first execute some instruction which are independent from the current logic.

- **Estimation method/ predictive method:** - here we use either 2 stage prediction, then policy change or 1 stage prediction then policy change.



**Q** An instruction pipeline has five stages, namely, instruction fetch (IF), instruction decode and register fetch (ID/RF), instruction execution (EX), memory access (MEM), and register writeback (WB) with stage latencies 1 ns, 2.2 ns, 2 ns, 1 ns, and 0.75 ns, respectively (ns stands for nanoseconds).

To gain in terms of frequency, the designers have decided to split the ID/RF stage into three stages (ID, RF1, RF2) each of latency 2.2/3 ns.

Also, the EX stage is split into two stages (EX1, EX2) each of latency 1 ns. The new design has a total of eight pipeline stages. A program has 20% branch instructions which execute in the EX stage and produce the next instruction pointer at the end of the EX stage in the old design and at the end of the EX2 stage in the new design. The IF stage stalls after fetching a branch instruction until the next instruction pointer is computed. All instructions other than the branch instruction have an average CPI of one in both the designs. The execution times of this program on the old and the new design are P and Q nanoseconds, respectively. The value of P/Q is \_\_\_\_\_. (**Gate-2014, 2marks**) ?

**Q** Consider an instruction pipeline with five stages without any branch prediction: Fetch Instruction (FI), Decode Instruction (DI), Fetch Operand (FO), Execute Instruction (EI) and Write Operand (WO). The stage delays for FI, DI, FO, EI and WO are 5 ns, 7 ns, 10 ns, 8 ns and 6 ns, respectively. There are intermediate storage buffers after each stage and the delay of each buffer is 1 ns. A program consisting of 12 instructions  $I_1, I_2, I_3, \dots, I_{12}$  is executed in this pipelined processor. Instruction  $I_4$  is the only branch instruction and its branch target is  $I_9$ . If the branch is taken during the execution of this program, the time (in ns) needed to complete the program is (Gate-2013) (2 Marks)

- (A) 132  
(B) 165  
(C) 176  
(D) 328

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
$I_1$																				
$I_2$																				
$I_3$																				
$I_4$																				
$I_5$																				
$I_6$																				
$I_7$																				
$I_8$																				
$I_9$																				
$I_{10}$																				
$I_{11}$																				
$I_{12}$																				

**Q** A CPU has a five-stage pipeline and runs at 1 GHz frequency. Instruction fetch happens in the first stage of the pipeline. A conditional branch instruction computes the target address and evaluates the condition in the third stage of the pipeline. The processor stops fetching new instructions following a conditional branch until the branch outcome is known. A program executes  $10^9$  instructions out of which 20% are conditional branches. If each instruction takes one cycle to complete on average, the total execution time of the program is: **(Gate-2006) (2 Marks)**

**(A) 1.0 second**

**(B) 1.2 seconds**

**(C) 1.4 seconds**

**(D) 1.6 seconds**

# Data Hazards

- Data hazards occur when instructions that exhibit data dependence, modify data in different stages of a pipeline. Hazard cause delays in the pipeline. E.g.

Instruction	Meaning of instruction
I <sub>0</sub> : MUL R <sub>2</sub> , R <sub>0</sub> , R <sub>1</sub>	R <sub>2</sub> ← R <sub>0</sub> * R <sub>1</sub>
I <sub>1</sub> : DIV R <sub>5</sub> , R <sub>3</sub> , R <sub>4</sub>	R <sub>5</sub> ← R <sub>3</sub> / R <sub>4</sub>
I <sub>2</sub> : ADD R <sub>2</sub> , R <sub>5</sub> , R <sub>2</sub>	R <sub>2</sub> ← R <sub>5</sub> + R <sub>2</sub>
I <sub>3</sub> : SUB R <sub>5</sub> , R <sub>2</sub> , R <sub>6</sub>	R <sub>5</sub> ← R <sub>2</sub> - R <sub>6</sub>



knowledge

- There are mainly three types of data hazards: This condition is called **Bernstein condition**. RAW (Read after Write) [Flow/True data dependency].
- RAW hazard occurs when instruction J tries to read data before instruction I write it.

$$I: R_2 \leftarrow R_1 + R_3$$

$$J: R_4 \leftarrow R_2 + R_3$$

Here, J is trying to read  $R_2$  before I have written it.

- WAR (Write after Read) [Anti-Data dependency]
- WAR hazard occurs when instruction J tries to write data before instruction I read it.

I:  $R_2 \leftarrow R_1 + R_3$

J:  $R_3 \leftarrow R_4 + R_5$

Here, J is trying to write  $R_3$  before I have read it.

- WAW (Write after Write) [Output data dependency]
- WAW hazard occurs when instruction J tries to write output before instruction I write it.

I:  $R_2 \leftarrow R_1 + R_3$

J:  $R_2 \leftarrow R_4 + R_5$

Here J is trying to write  $R_2$  before I.

***WAR and WAW hazards occur during the out-of-order execution of the instructions.***

**We say that instruction  $S_2$  depends in instruction  $S_1$ , when:** This condition is called **Bernstein condition.**

Three cases exist:

Flow (data) dependence:  $O(S_1) \cap I(S_2)$ ,  $S_1 \rightarrow S_2$  and  $S_1$  writes after something read by  $S_2$

Anti-dependence:  
overwrites it  $I(S_1) \cap O(S_2)$ ,  $S_1 \rightarrow S_2$  and  $S_1$  reads something before  $S_2$

Output dependence:  $O(S_1) \cap O(S_2)$ ,  $S_1 \rightarrow S_2$  and both write the same memory location.

$$[I(S_1) \cap O(S_2)] \cup [O(S_1) \cap I(S_2)] \cup [O(S_1) \cap O(S_2)] \neq \emptyset$$

## Solution of Data dependency

- We can use code movement or code relocation and can execute the dependent instruction after some time.
- Here we can use operator forwarding using which we can directly access the result after execution instead of waiting that it gets store in memory.

**Q** A 5-stage pipelined processor has Instruction Fetch (IF), Instruction Decode (ID), Operand Fetch (OF), Perform Operation (PO)and Write Operand (WO)stages. The IF, ID, OF and WO stages take 1 clock cycle each for any instruction. The PO stage takes 1 clock cycle for ADD and SUB instructions,3 clock cycles for MUL instruction, and 6 clock cycles for DIV instruction respectively. Operand forwarding is used in the pipeline. What is the number of clock cycles needed to execute the following sequence of instructions? **(Gate-2010) (2 Marks)**

Instruction                      Meaning of instruction

$I_0: MUL R_2, R_0, R_1$	$R_2 \rightarrow R_0 * R_1$
$I_1: DIV R_5, R_3, R_4$	$R_5 \rightarrow R_3/R_4$
$I_2: ADD R_2, R_5, R_2$	$R_2 \rightarrow R_5 + R_2$
$I_3: SUB R_5, R_2, R_6$	$R_5 \rightarrow R_2 - R_6$

**(A) 13**

**(B) 15**

**(C) 17**

**(D) 19**

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20



**Q** The instruction pipeline of a RISC processor has the following stages: Instruction Fetch (IF), Instruction Decode (ID), Operand Fetch (OF), Perform Operation (PO) and Writeback (WB). The IF, ID, OF and WB stages take 1 clock cycle each for every instruction. Consider a sequence of 100 instructions. In the PO stage, 40 instructions take 3 clock cycles each, 35 instructions take 2 clock cycles each, and the remaining 25 instructions take 1 clock cycle each. Assume that there are no data hazards and no control hazards. The number of clock cycles required for completion of execution of the sequence of instruction is \_\_\_\_\_. (GATE-2018) (2Marks)

**Q Consider the sequence of machine instructions given below: (Gate-2015) (2 Marks)**

MUL R<sub>5</sub>, R<sub>0</sub>, R<sub>1</sub>  
DIV R<sub>6</sub>, R<sub>2</sub>, R<sub>3</sub>  
ADD R<sub>7</sub>, R<sub>5</sub>, R<sub>6</sub>  
SUB R<sub>8</sub>, R<sub>7</sub>, R<sub>4</sub>

In the above sequence, R<sub>0</sub> to R<sub>8</sub> are general purpose registers. In the instructions shown, the first register stores the result of the operation performed on the second and the third registers. This sequence of instructions is to be executed in a pipelined instruction processor with the following 4 stages:

- (1) Instruction Fetch and Decode (IF),
- (2) Operand Fetch (OF),
- (3) Perform Operation (PO) and
- (4) Write back the Result (WB).

The IF, OF and WB stages take 1 clock cycle each for any instruction. The PO stage takes 1 clock cycle for ADD or SUB instruction, 3 clock cycles for MUL instruction and 5 clock cycles for DIV instruction.

The pipelined processor uses operand forwarding from the PO stage to the OF stage. The number of clock cycles taken for the execution of the above sequence of instructions is \_\_\_\_\_

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

**Q Consider a pipelined processor with the following four stages: (Gate-2007) (2 Marks)**

**IF:** Instruction Fetch

**ID:** Instruction Decode and Operand Fetch

**EX:** Execute

**WB:** Write Back

The IF, ID and WB stages take one clock cycle each to complete the operation. The number of clock cycles for the EX stage depends on the instruction. The ADD and SUB instructions need 1 clock cycle and the MUL instruction needs 3 clock cycles in the EX stage. Operand forwarding is used in the pipelined processor. What is the number of clock cycles taken to complete the following sequence of instructions?

ADD R<sub>2</sub>, R<sub>1</sub>, R<sub>0</sub>      R<sub>2</sub> <- R<sub>0</sub> + R<sub>1</sub>

MUL R<sub>4</sub>, R<sub>3</sub>, R<sub>2</sub>      R<sub>4</sub> <- R<sub>3</sub> \* R<sub>2</sub>

SUB R<sub>6</sub>, R<sub>5</sub>, R<sub>4</sub>      R<sub>6</sub> <- R<sub>5</sub> - R<sub>4</sub>

**(A) 7**

**(B) 8**

**(C) 10**

**(D) 14**

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

**Q** Consider a pipelined processor with 5 stages, Instruction Fetch(IF), Instruction Decode(ID), Execute (EX), Memory Access (MEM), and Write Back (WB). Each stage of the pipeline, except the EX stage, takes one cycle. Assume that the ID stage merely decodes the instruction and the register read is performed in the EX stage. The EX stage takes one cycle for ADD instruction and the register read is performed in the EX stage. The EX stage takes one cycle for ADD instruction and two cycles for MUL instruction. Ignore pipeline register latencies. Consider the following sequence of 8 instructions:

**ADD, MUL, ADD, MUL, ADD, MUL, ADD, MUL**

Assume that every MUL instruction is data-dependent on the ADD instruction just before it and every ADD instruction (except the first ADD) is data-dependent on the MUL instruction just before it. The speedup defined as follows.

**Speedup = (Execution time without operand forwarding) / (Execution time with operand forwarding)**

The Speedup achieved in executing the given instruction sequence on the pipelined processor (rounded to 2 decimal places) is \_\_\_\_\_.  
**(GATE 2021) (2 MARKS)**

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30

**Q** A processor  $X_1$  operating at 2 GHz has a standard 5-stage RISC instruction pipeline having a base CPI (cycles per instruction) of one without any pipeline hazards. For a given program P that has 30% branch instructions, control hazards incur 2 cycles stall for every branch. A new version of the processor  $X_2$  operating at same clock frequency has an additional branch predictor unit (BPU) that completely eliminates stalls for correctly predicted branches. There is neither any savings nor any additional stalls for wrong predictions. There are no structural hazards and data hazards for  $X_1$  and  $X_2$ . If the BPU has a prediction accuracy of 80%, the speed up (rounded off to two decimal places) obtained by  $X_2$  over  $X_1$  in executing P is \_\_\_\_\_. **(GATE 2022) (2 MARKS)**

# Instruction format

- Instructions in a computer system can be classified according to the number of **operands** or **references** made within the instruction. The main types are:
  - **3-Address Instruction:** Involves three operands, typically used for operations where both source and destination operands are specified separately.
  - **2-Address Instruction:** Involves two operands, commonly used for instructions that operate on one source and one destination operand.
  - **1-Address Instruction:** Involves a single operand, often used when one operand is implied (like an accumulator).
  - **0-Address Instruction:** Uses no explicit operands; often used in **stack-based architectures** where operands are implicitly taken from the stack.

## 3 Address Instruction

- **Structure:** A three-address instruction contains **one opcode** and **three address fields**. One field is for the **destination** and the other two are for **source operands**.
- **Advantages:** These instructions tend to produce shorter, more understandable programs because they can handle complex operations in a single step.
- **Considerations:**
  - While the instructions contain more information, the overall speed of the program may not significantly increase. Each **micro-operation**, such as loading data into registers or placing an address on the address bus, still takes one cycle to execute.
  - **More bits** are required to encode these instructions in binary due to the three addresses. This increases the complexity of the processing and decoding circuits, requiring more advanced hardware.
- $X = (A + B) \times (C + D)$

Mode/Opcode	Destination	Source <sub>1</sub>	Source <sub>2</sub>
-------------	-------------	---------------------	---------------------

ADD R<sub>1</sub>, A, B

R<sub>1</sub> <- M[A] + M[B]

ADD R<sub>2</sub>, C, D

R<sub>2</sub> <- M[C] + M[D]

MUL X, R<sub>1</sub>, R<sub>2</sub>

M[X] <- R<sub>1</sub> × R<sub>2</sub>

## 2 Address Instruction

Mode/Opcode	Destination/Source <sub>1</sub>	Source <sub>2</sub>
-------------	---------------------------------	---------------------

- **Structure:** Two addresses are specified in the instruction. One address serves as both **Source1** and the **destination**, while the second address is for **Source2**.
- **Advantages:**
  - **Smaller Instruction Size:** By removing the need for a separate destination address, the instruction length is reduced compared to a three-address instruction.
  - **Efficiency:** Fewer registers are required, and the overall instruction format is more compact.
- **Considerations:**
  - **Optimization Difficulty:** Code optimization is more challenging due to the dual-use of one address for both a source and the result. This might complicate operations like pipelining or parallel execution.

$X = (A + B) \times (C + D)$	MOV	$R_1, A$	$R_1 \leftarrow M[A]$
	ADD	$R_1, B$	$R_1 \leftarrow R_1 + M[B]$
	MOV	$R_2, C$	$R_2 \leftarrow C$
	ADD	$R_2, D$	$R_2 \leftarrow R_2 + D$
	MUL	$R_1, R_2$	$R_1 \leftarrow R_1 * R_2$
	MOV	$X, R_1$	$M[X] \leftarrow R_1$

# 1 Address Instruction

Mode/Opcode	Destination/Source
-------------	--------------------

- **Structure:** This instruction type utilizes an **implied accumulator (AC) register** for data manipulation.
- **Operation:**
  - One operand is stored in the **accumulator**, while the other operand is retrieved from a **register or memory location**.
  - The CPU inherently knows that one operand is in the accumulator, so there is no need to explicitly specify the accumulator in the instruction.
- **Advantages:**
  - **Simplicity:** The accumulator's implied use reduces the complexity of the instruction format since fewer fields are required to specify operands.
  - **Efficiency:** By eliminating the need to specify the accumulator, instruction size is reduced, making the operations faster.

$X = (A + B) \times (C + D)$	LOAD	A	$AC \leftarrow M[A]$
	ADD	B	$AC \leftarrow AC + M[B]$
	STORE	T	$M[T] \leftarrow AC$
	LOAD	C	$AC \leftarrow M[C]$
	ADD	D	$AC \leftarrow AC + M[D]$
	MUL	T	$AC \leftarrow AC * M[T]$
	STORE	X	$M[X] \leftarrow AC$

## 0 Address Instruction

- **Structure:** Early computers without complex circuitry like ALUs often relied on registers organized as stacks. These computers typically do not require explicit address fields for certain instructions, such as ADD and MUL.
- **Operation:** The result of an operation is implied to be at the **top of the stack**, so no explicit location for the result is needed. **Stack-based instructions** like PUSH and POP require an address field to specify the operand that interacts with the stack, but arithmetic and logic instructions do not.
- **Advantages: Simplification:** Eliminating the need for address fields in many operations simplifies the instruction set, making it more efficient for stack-based processing.
- **Considerations:** While the simplicity of stack operations is beneficial for early computers or specific tasks, the lack of address flexibility may limit the general-purpose functionality compared to more advanced systems.

Mode/Opcode	PUSH	A	TOP $\leftarrow$ A
	PUSH	B	TOP $\leftarrow$ B
	ADD		TOP $\leftarrow$ A+B
	PUSH	C	TOP $\leftarrow$ C
	PUSH	D	TOP $\leftarrow$ D
	ADD		TOP $\leftarrow$ C+D
	MUL		TOP $\leftarrow$ (C+D)*(A+B)
	POP	X	M[X] $\leftarrow$ TOP

- **Expression Evaluation:** In a stack-based computer, arithmetic expressions must be converted into **reverse polish notation** (RPN) for evaluation. This postfix notation simplifies the processing of operations by eliminating the need for parentheses and making the order of operations explicit.
- **Naming:** These systems are called **zero-address computers** because instructions do not require an address field. The operands are implied to be on the stack.
- **Instruction Format:** Only the **mode or opcode** is needed to perform an operation, as the stack handles the operand locations implicitly.
- **Pros and Cons:**
  - **Advantages:** The instruction format is **simple** and **compact**, reducing the instruction size and simplifying the code.
  - **Challenges:** While the simplicity of the format is beneficial, **design complexity** increases, and the **processing speed** tends to be slower compared to more advanced architectures.

**Q** Consider a processor with 64 registers and an instruction set of size twelve. Each instruction has five distinct fields, namely, opcode, two source register identifiers, one destination register identifier, and a twelve-bit immediate value. Each instruction must be stored in memory in a byte-aligned fashion. If a program has 100 instructions, the amount of memory (in bytes) consumed by the program text is \_\_\_\_\_ (Gate-2016) (2 Marks)

**Q** A CPU has 24-bit instructions. A program starts at address 300 (in decimal). Which one of the following is a legal program counter (all values in decimal)?  
**(Gate-2006) (1 Marks)**

- (A)** 400
- (B)** 500
- (C)** 600
- (D)** 700

Q A processor has 16 integer registers ( $R_0, R_1, \dots, R_{15}$ ) and 64 floating point registers ( $F_0, F_1, \dots, F_{63}$ ). It uses a 2-byte instruction format. There are four categories of instructions: Type-1, Type-2, Type-3, and Type 4. Type-1 category consists of four instructions, each with 3 integers register operands (3Rs). Type-2 category consists of eight instructions, each with 2 floating point register operands (2Fs). Type-3 category consists of fourteen instructions, each with one integer register operand and one floating point register operand (1R+1F). Type-4 category consists of N instructions, each with a floating-point register operand (1F). The maximum value of N is \_\_\_\_\_. (Gate-2018) (2 Marks)

**Q** Consider a hypothetical machine where each instruction of 16 bit. If address uses 4 bits then.

- i) How many 1-address instruction could be supported?
- ii) How many 2-address instruction could be supported?
- iii) How many 3-address instruction could be supported?

**Q** Consider a system where memory contains 512 words if we want 2048, 1-address instructions and 4, 2-address instructions find the length of instruction?

**Q** Consider a hypothetical machine which support both 1-address and 2-address instruction, if each instruction of 16 bit and memory contain only 128 words. If it known that there are two, 2-address instruction in the system how many one address instruction can be supported?

# Addressing Mode

- **Operand Reference:** Operand addressing specifies the various methods for referencing the location of an operand.
- **Effective Address:** This refers to the **final address** where the operand is located in memory.
- **Address Calculation:**
  - **Non-Computable Addressing:** Direct reference without the need for arithmetic calculations.
  - **Computable Addressing:** Involves the use of **arithmetic operations** to compute the final address of the operand.

- **Criteria for Selecting Addressing Modes:**
  - **Speed**: Addressing modes should allow for fast execution of instructions.
  - **Instruction Length**: The length of the instruction must be minimized for efficiency.
  - **Pointer Support**: The addressing mode should facilitate the use of pointers.
  - **Looping and Indexing Support**: Addressing modes should support looping constructs and indexing of data structures.
  - **Program Relocation**: Addressing should allow for the relocation of programs in memory.

# Immediate mode addressing

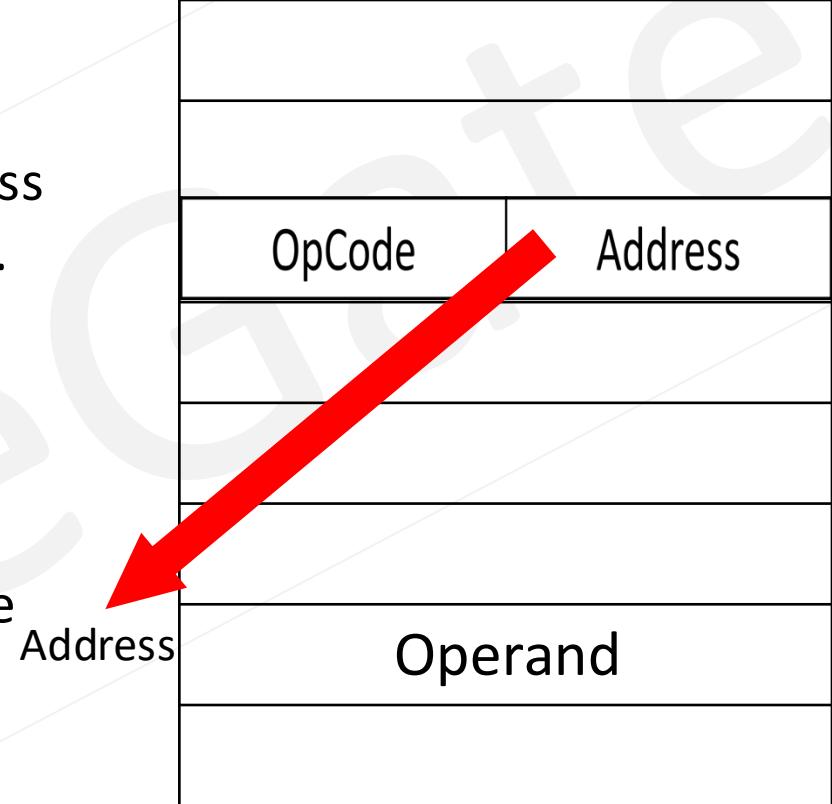
- **Concept:** The operand is embedded directly within the instruction.  
For example, ADD 3 means "Add 3 to the accumulator."
- **Advantages:**
  - Ideal for constants, where values are predetermined.
  - Extremely fast as no memory reference is required.
- **Disadvantages:**
  - Cannot be used for variables with unknown values at the time of programming.
  - Limited to smaller constants, as larger values might not fit within the instruction.
- **Application:**
  - Commonly used when data needs to be directly loaded into a register or memory.

OpCode	Operand

**Q** A processor has 40 distinct instructions and 24 general purpose registers. A 32-bit instruction word has an opcode, two register operands and an immediate operand. The number of bits available for the immediate operand field is \_\_\_\_\_. **(Gate-2016) (1 Marks)**

## Direct mode addressing (absolute address mode)

- Direct addressing mode involves the instruction containing the address of the memory location where the data is present (effective address). Only one memory reference operation is required to access the data.
- **Advantages:**
  - Ideal for variables whose values are unknown during program writing, as it simplifies data access.
  - No restrictions on the range of data values; it allows access to the system's largest data values.
  - Useful for accessing global variables whose addresses are known at compile time.
- **Disadvantages:**
  - Slower compared to immediate mode due to memory reference operations.
  - The number of variables that can be used is limited.
  - May not be efficient for large calculations due to performance limitations.

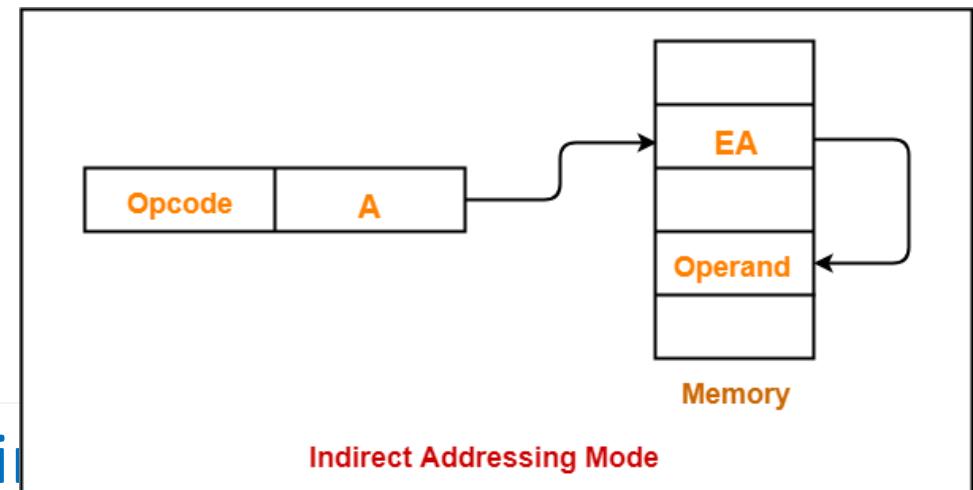
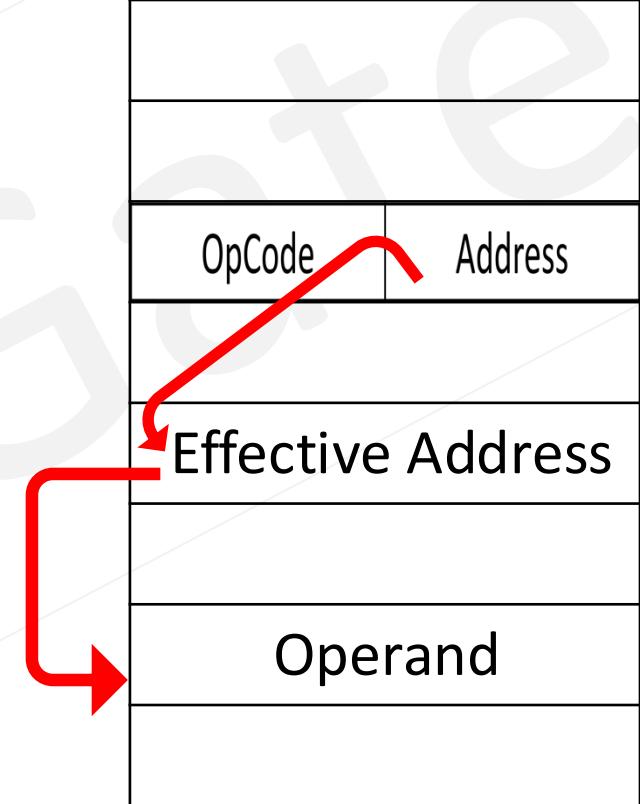


**Q In the absolute addressing mode (Gate-2002) (2 Marks)**

- (A) the operand is inside the instruction
- (B) the address of the operand is inside the instruction
- (C) the register containing address of the operand is specified inside the instruction
- (D) the location of the operand is implicit

## Indirect mode addressing

- In this addressing mode, the instruction stores the address where the effective address (the address of the variable) is stored. Two references are required:
  - The first reference retrieves the effective address.
  - The second reference accesses the actual data.
- **Advantages:**
  - No limitations on the number or size of variables.
  - Facilitates pointer implementation with relatively more security.
- **Disadvantages:**
  - It is relatively slow because memory must be accessed multiple times.

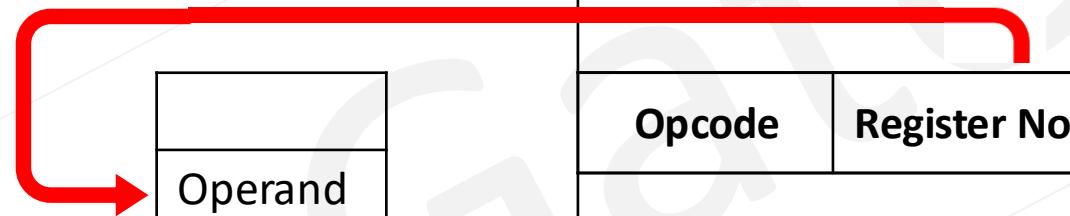


# Implied mode addressing

- In this mode, operands are not explicitly stated; they are implicitly defined by the instruction itself. This is commonly used when operations involve registers like an accumulator, where the operand location is predetermined by the CPU.
  - **Implied Mode Instructions:** Any instruction that directly references registers such as the accumulator (e.g., increment or complement operations) uses implied mode addressing.
  - **Zero Address Instructions:** In stack-organized computers, zero address instructions also fall under implied mode addressing, often called stack addressing mode.
- **Example Instructions:**
  - Increment accumulator (INC A)
  - Complement accumulator (CPL A)

## Register mode addressing

- In this mode, variables are stored directly in the CPU's registers instead of memory. The instruction will specify which register contains the data by providing the register number.
- **Advantages:**
  - **Speed:** Register access time is faster than cache or memory access, making this addressing mode extremely fast.
  - **Efficient Use of Bits:** Since the number of registers is limited, fewer bits are required to specify the register in the instruction.
- **Disadvantages:**
  - **Limited Capacity:** Due to the limited number of registers, only a few variables can be handled using this method, making it less suitable for programs with a large number of variables.



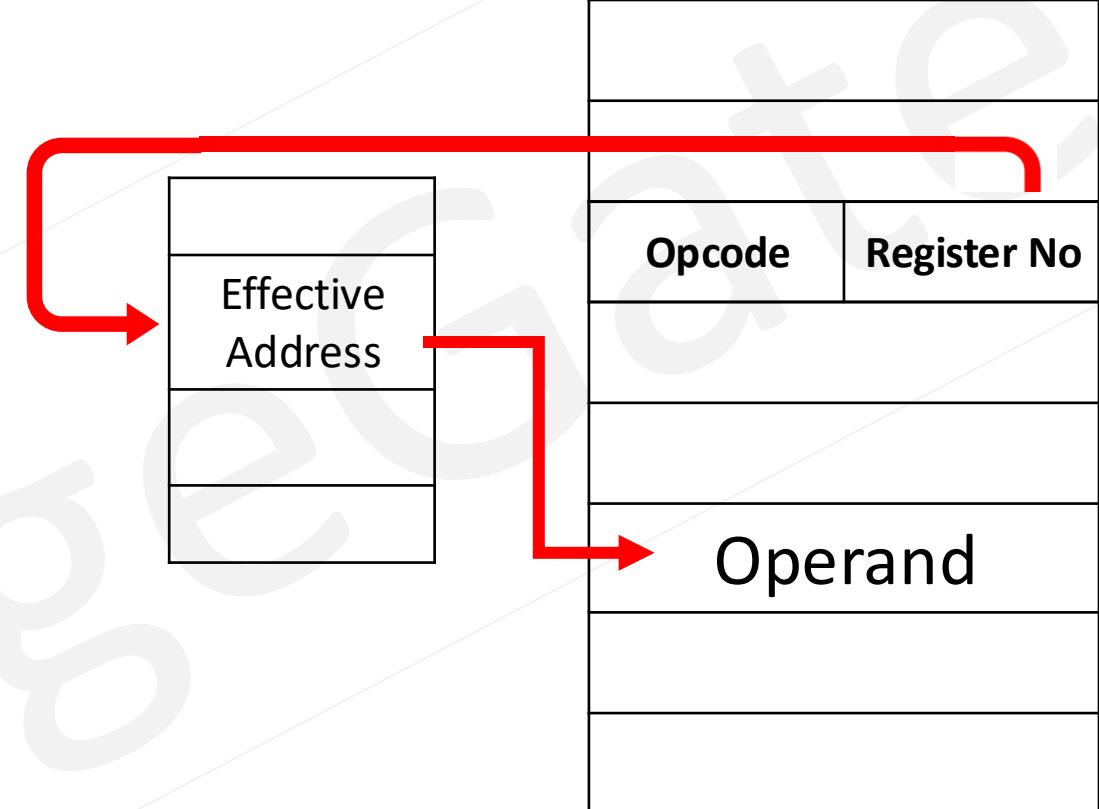
Opcode	Register No

The diagram illustrates the structure of a register mode instruction. It features a stack-like structure with three horizontal slots. The top slot is labeled "Operand". A red arrow points from the "Register No" column of the table above to the "Operand" slot, indicating that the register number is used to identify the source or destination register. The other two slots in the stack are empty.

**Q** A machine has a 32-bit architecture, with 1-word long instructions. It has 64 registers, each of which is 32 bits long. It needs to support 45 instructions, which have an immediate operand in addition to two register operands. Assuming that the immediate operand is an unsigned integer, the maximum value of the immediate operand is \_\_\_\_\_. (Gate-2014) (1 Marks)

# Register indirect mode addressing

- In register indirect mode, the instruction specifies a register containing the memory address (effective address) of the variable. The CPU uses the content of the register to determine where the actual data is stored.
  - **Key Points:**
    - **Efficient for Repeated Access:** This mode is highly efficient when the same address is needed multiple times, as the register holds the memory address.
    - **Flexible Addressing:** By changing the content of the register, it can point to different memory locations without modifying the instruction, making it more flexible compared to the register mode.
    - **Useful for Pointer Arithmetic:** This mode enhances flexibility in pointer arithmetic as the register holds the reference to the memory.
    - **Auto-Increment/Decrement:** This mode can be enhanced by using auto-increment and auto-decrement commands, especially useful when dealing with continuous data structures like arrays or matrices, as it allows sequential access to elements.

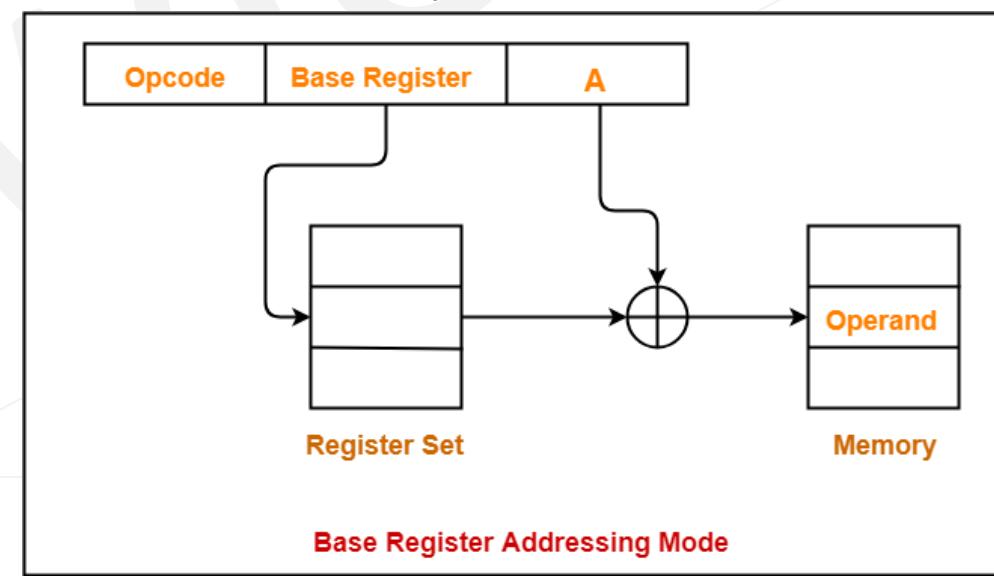


**Q Which of the following is/are true of the auto-increment addressing mode? (Gate-2008) (2 Marks)**

- i) It is useful in creating self-relocating code
  - ii) If it is included in an Instruction Set Architecture, then an additional ALU is required for effective address calculation
  - iii) The amount of increment depends on the size of the data item accessed
- a) I only
  - b) II only
  - c) III only
  - d) II and III only

## Base register (off set) mode

- In a multiprogramming environment, the memory location of processes can change dynamically. Direct addressing could cause issues when a process moves within memory.
- To address this problem, a **base register** is used to store the starting address of the program. Instead of providing direct memory addresses in instructions, an **offset** is given.
- **Effective Address = Base Address + Offset**
- This way, if a process is relocated in memory, only the base register needs to be updated, while the instruction continues to function as intended. The final memory address is calculated by adding the base register value to the offset provided in the instruction.
- **Advantages:**
  - Facilitates process relocation without modifying the program code.
  - Allows flexible process placement in memory.



## Based Indexed Addressing

- The **effective address** is calculated by adding the contents of a **base register** to the **address part of the instruction**.
- This method is particularly useful for accessing arrays and structures in memory, where an index can be added to a base memory location to reference different elements efficiently.
- **Summary:**
- Effective Address = Base Register Content + Address in Instruction
- Used in array and structure access.

**Q** Consider a hypothetical processor with an instruction of type LW R<sub>1</sub>, 20(R<sub>2</sub>), which during execution reads a 32-bit word from memory and stores it in a 32-bit register R<sub>1</sub>. The effective address of the memory location is obtained by the addition of a constant 20 and the contents of register R<sub>2</sub>. Which of the following best reflects the addressing mode implemented by this instruction for operand in memory? **(Gate-2011) (1 Marks)**

**(A)** Immediate Addressing

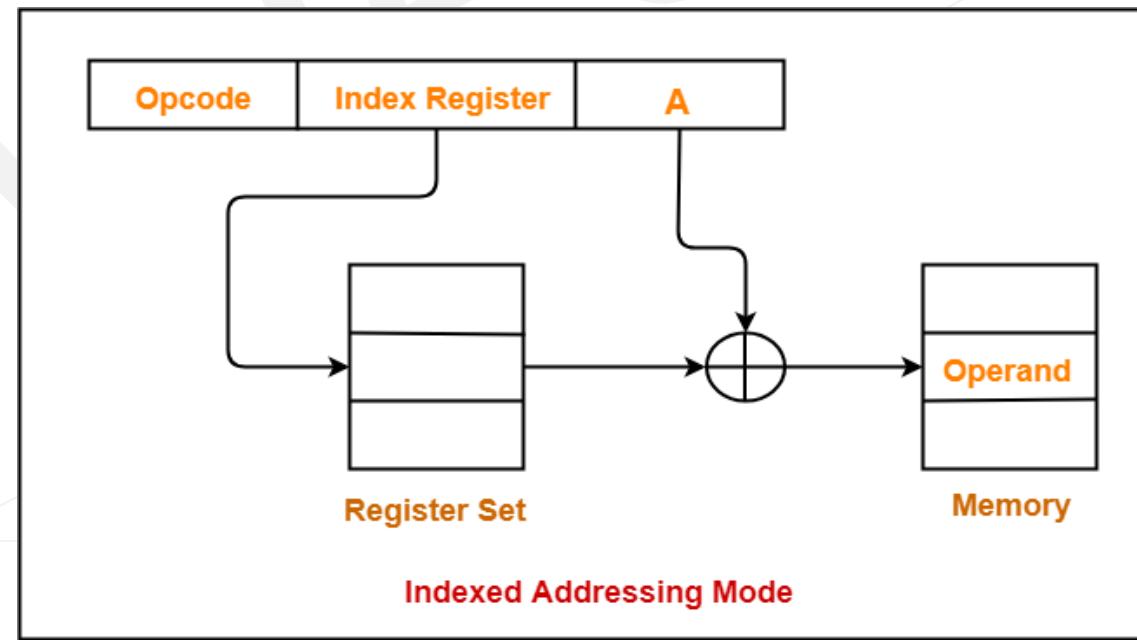
**(B)** Register Addressing

**(C)** Register Indirect Scaled Addressing

**(D)** Base Indexed Addressing

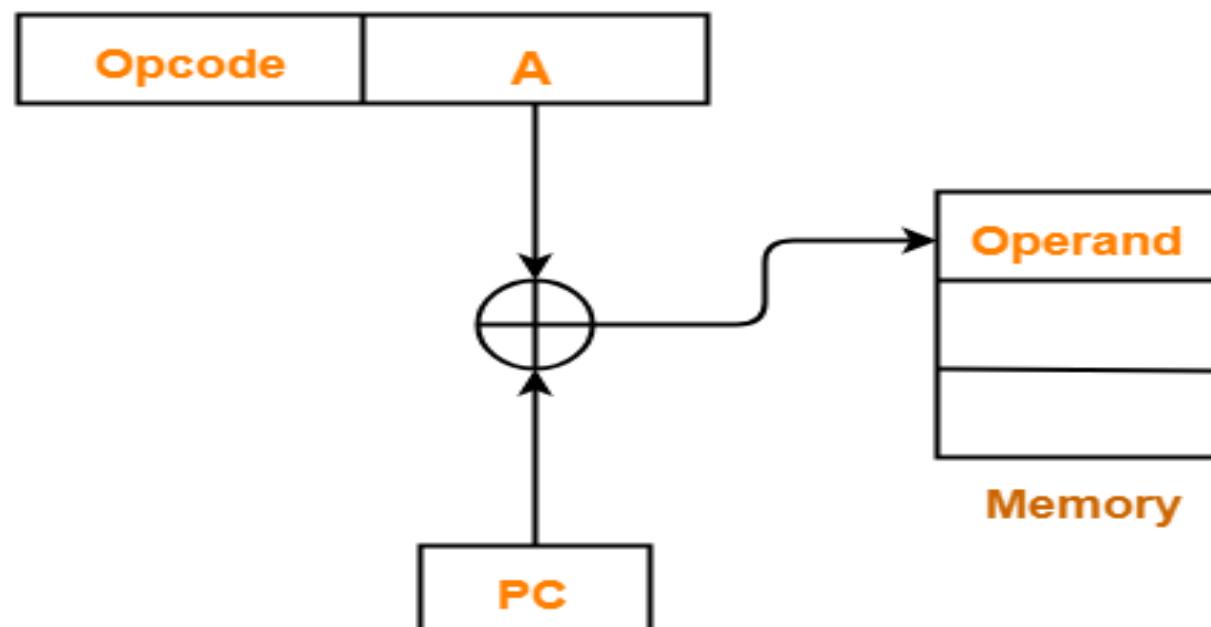
# Index addressing mode

- Index addressing mode is used when the CPU has multiple registers, one of which serves as an index register. This mode is particularly useful when accessing large arrays.
- The key idea is to store the base address of the array in the instruction, while the index (which points to the specific element of the array) is stored in the index register. By modifying the value of the index register, the same instruction can be used to access different elements within the array.
- **Base address:** Provided in the instruction.
- **Index:** Stored in a register, and can be changed to access different array elements.
- This approach is efficient for iterating through arrays, as the base address remains constant while only the index needs to be updated.



## Relative Addressing Mode

- The effective address of the operand is calculated by adding the content of the program counter (PC) to the address specified in the instruction. This is useful in situations where the operand's address is defined relative to the current instruction, such as in branching or looping operations.
- This mode simplifies code relocation and enhances program flexibility by enabling instructions to operate relative to their current location in memory.



**Q** Which of the following addressing modes are suitable for program relocation at run time? (Gate-2004) (1 Marks)

- (i) Absolute addressing
  - (iii) Relative addressing
  - (ii) Based addressing
  - (iv) Indirect addressing
- (A)** (i) and (iv)
- (B)** (i) and (ii)
- (C)** (ii) and (iii)
- (D)** (i), (ii) and (iv)

**Q** For computers based on three-address instruction formats, each address field can be used to specify which of the following: **(Gate-2015) (1 Marks)**

**S<sub>1</sub>**: A memory operand

**S<sub>2</sub>**: A processor register

**S<sub>3</sub>**: An implied accumulator register

**(A)** Either S<sub>1</sub> or S<sub>2</sub>

**(B)** Either S<sub>2</sub> or S<sub>3</sub>

**(C)** Only S<sub>2</sub> and S<sub>3</sub>

**(D)** All of S<sub>1</sub>, S<sub>2</sub> and S<sub>3</sub>

**Q Which is the most appropriate match for the items in the first column with the items in the second column: (Gate-2001) (2 Marks)**

X. Indirect Addressing	I. Array implementation
Y. Indexed Addressing	II. Writing relocatable code
Z. Base Register Addressing	III. Passing array as parameter

- a) (X, III), (Y, I), (Z, II)
- b) (X, II), (Y, III), (Z, I)
- c) (X, III), (Y, II), (Z, I)
- d) (X, I), (Y, III), (Z, II)

**Q The most appropriate matching for the following pairs (Gate-2000) (1 Marks)**

X: Indirect addressing	1 : Loops
Y: Immediate addressing	2 : Pointers
Z: Auto decrement addressing	3: Constants

- (A) X-3, Y-2, Z-1
- (B) X-1, Y-3, Z-2
- (C) X-2, Y-3, Z-1
- (D) X-3, Y-1, Z-2

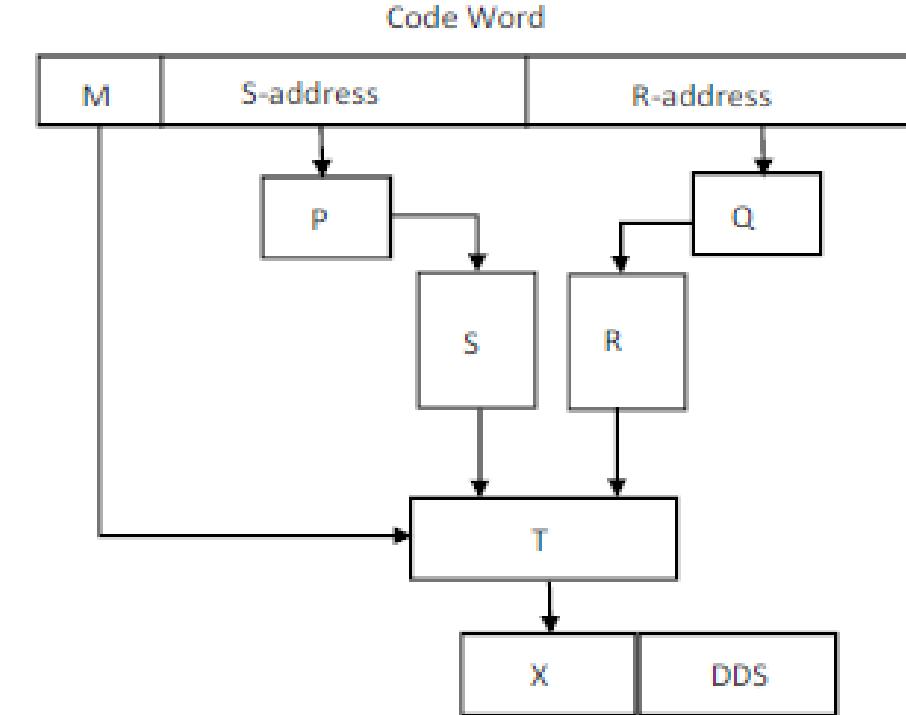
**Q** Consider the following instruction sequence where registers R<sub>1</sub>, R<sub>2</sub> and R<sub>3</sub> are general purpose and MEMORY[X] denotes the content at the memory location X. Assume that the content of the memory location 5000 is 10, and the content of the register R3 is 3000. The content of each of the memory locations from 3000 to 3010 is 50. The instruction sequence starts from the memory location 1000. All the numbers are in decimal format. Assume that the memory is byte addressable. After the execution of the program, the content of memory location 3010 is \_\_\_\_\_ . (GATE 2021) (2 MARKS)

Instruction	Semantics	Instruction Size (bytes)
MOV R1, (5000)	R1 $\leftarrow$ MEMORY[5000]	4
MOV R2, (R3)	R2 $\leftarrow$ MEMORY[R3]	4
ADD R2, R1	R2 $\leftarrow$ R1+R2	2
MOV (R3), R2	MEMORY[R3] $\leftarrow$ R2	4
INC R3	R3 $\leftarrow$ R3+1	2
DEC R1	R1 $\leftarrow$ R1-1	2
BNZ 1004	Branch if not zero to the given absolute address	2
HALT	Stop	1

Q. Consider a digital display system (DDS) shown in the figure that displays the contents of register X. A 16-bit code word is used to load a word in X, either from S or from R. S is a 1024-word memory segment and R is a 32-word register file. Based on the value of mode bit M, T selects an input word to load in X. P and Q interface with the corresponding bits in the code word to choose the addressed word.

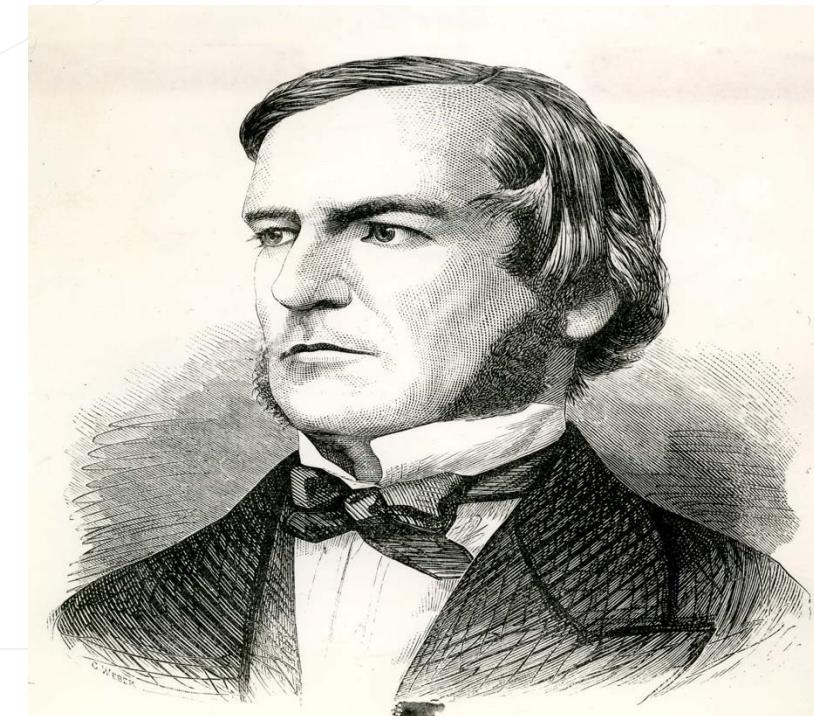
Which one of the following represents the functionality of P, Q, and T? (GATE 2022) (2 MARKS)

- (A) P is 10:1 multiplexer; Q is 5:1 multiplexer; T is 2:1 multiplexer
- (B) P is 10:210 decoder; Q is 5:25 decoder; T is 2:1 encoder
- (C) P is 10:210 decoder; Q is 5:25 decoder; T is 2:1 multiplexer
- (D) P is 1:10 de-multiplexer; Q is 1:5 de-multiplexer; T is 2:1 multiplexer



# Boolean algebra

- **Boolean Algebra:** Introduced by George Boole in the 19th century, specifically in his books "The Mathematical Analysis of Logic" (1847) and "An Investigation of the Laws of Thought" (1854).
- **Binary System:** In digital systems, signals usually have two discrete values, representing binary (0 and 1).
- **Core Operations:** Boolean algebra operates with variables representing truth values (true/false or 1/0), unlike traditional algebra which deals with numerical values.
  - **Conjunction (AND):** Represented as  $\wedge$
  - **Disjunction (OR):** Represented as  $\vee$
  - **Negation (NOT):** Represented as  $\neg$



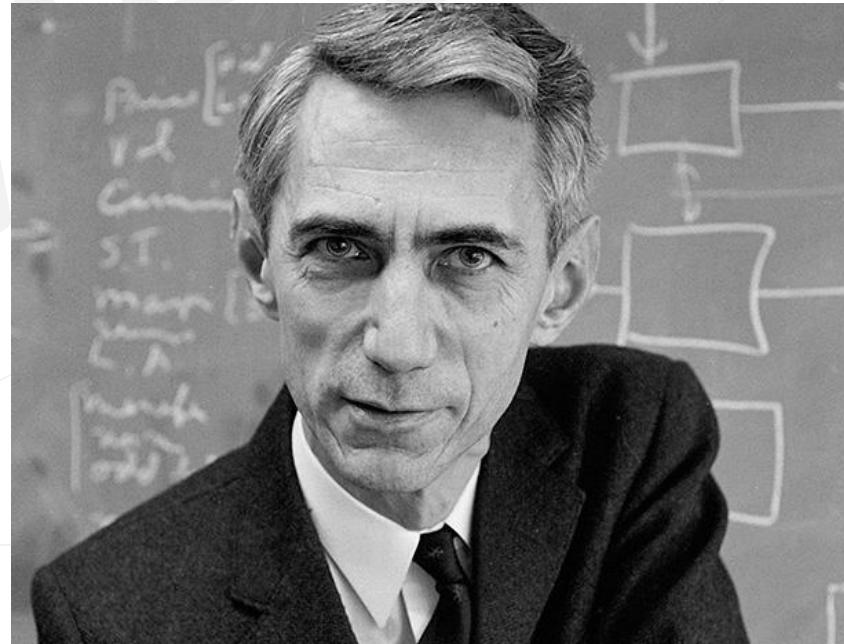
## Turing Machine

- **Church-Turing Thesis (1936)**: States that any algorithmic procedure that can be performed by a human or a computer can also be performed by a Turing machine.
- **Universal Acceptance**: The Turing machine is widely recognized by computer scientists as the ideal theoretical model of computation.



# Digital System

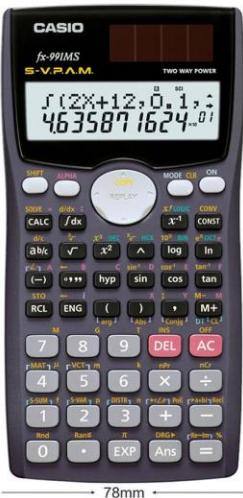
- Claude Shannon (1930s): Applied **Boolean algebra** to switching circuits, introducing **switching algebra** for circuit design using **logic gates**.
- Shannon's 1938 Work: Shannon, father of information theory, built upon Boole's work to create a systematic framework for digital hardware design.
- Key Points:
  - Shannon used **Boolean algebra** for switching algebra and circuit design.
  - Boolean constants: **0** and **1**. Variables represent signals like input/output.
  - Variables (a, b, c...) follow **Boolean laws** essential to **combinational logic**.



- Historically, there have been two types of computers:
  - **Fixed Program Computers** (Dedicated Devices / Embedded Systems):
    - These have a specific function and cannot be reprogrammed. Examples include calculators and washing machines.
  - **Stored Program Computers** (General Purpose Computers / von Neumann Architecture):
    - These can be programmed to perform various tasks, with applications stored within the system, which is why they are named as such.

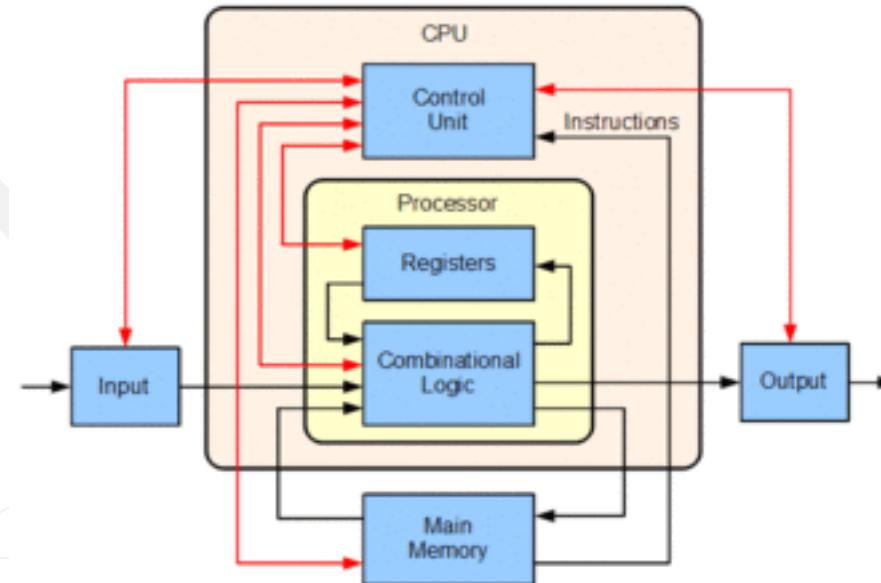
# Fixed Program Computers / Dedicated device / Embedded system

- **Embedded Systems:** These are designed to perform specific tasks, with their functionality permanently programmed into a chipset. Examples include washing machines and microwaves.
- **Characteristics:** Embedded systems contain small microprocessors that are hardwired to perform only specific tasks and cannot be reprogrammed for general-purpose computing.



# Stored Program Computers / General Purpose Computer / Von Neumann Architecture

- **Modern Computers:** They are based on the **stored-program concept** introduced by **John von Neumann**. This architecture allows computers to perform any task that a **Turing machine** can theoretically perform.
- **Stored-Program Concept:** A programmer writes a program, stores it in memory, and the computer executes it. Programs can be changed to alter the computer's functionality, making them adaptable to various tasks.
- **General-Purpose Computers:** These are programmable, allowing users to accomplish different tasks by simply changing the program stored in memory, reflecting the flexibility of the stored-program architecture.



A CPU generally consists of three key components:

- **Control Unit (CU):**

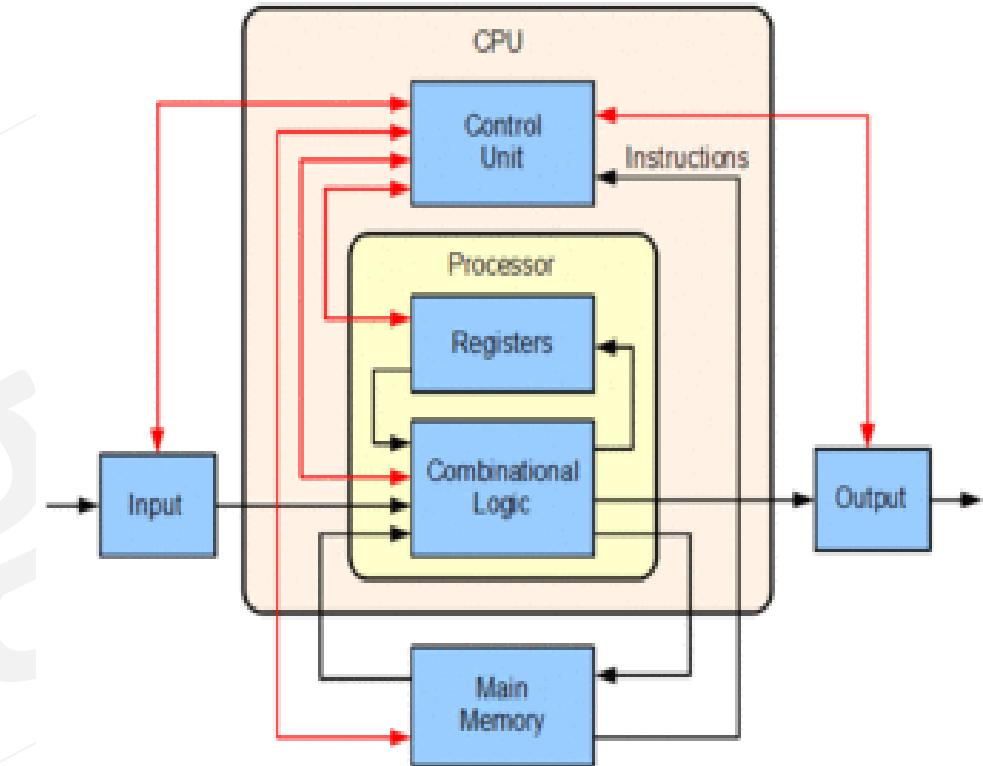
- Acts as the **master generator** of control signals, directing the operations of all other components within the CPU.
- Manages **input/output flow, instruction fetching**, and controls how data moves across the system, ensuring that instructions are executed in the correct order.

- **Registers:**

- Essential for the efficient functioning of the CPU, some key registers include:
  - **Program Counter (PC):** Holds the address of the next instruction to be executed.
  - **Instruction Register (IR):** Stores the address of the current instruction being executed.
  - **Base Register:** Contains the base address of the program being processed.

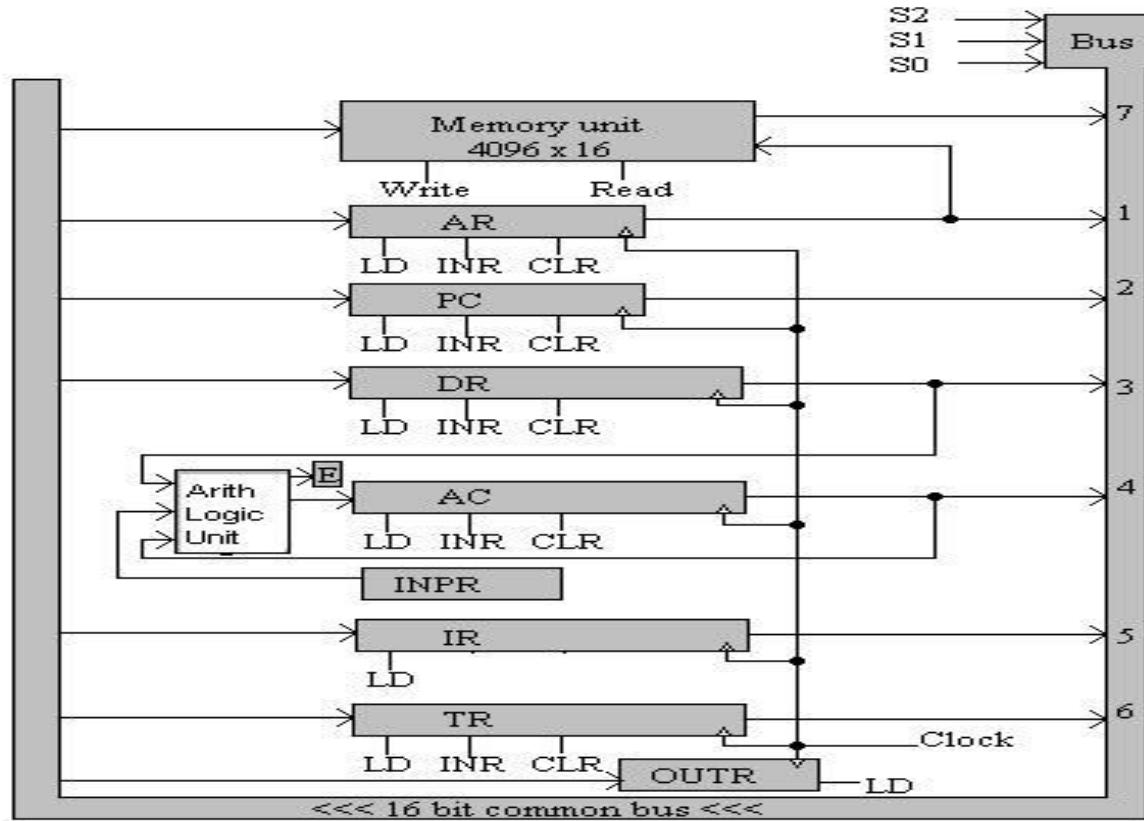
- **Arithmetic Logic Unit (ALU):**

- A **combinational circuit** capable of performing various operations:
  - **Arithmetic Operations:** Such as addition and subtraction.
  - **Bit Shifting Operations:** Manipulates bits within data.
  - **Logical Operations:** Includes comparisons and other boolean logic functions.



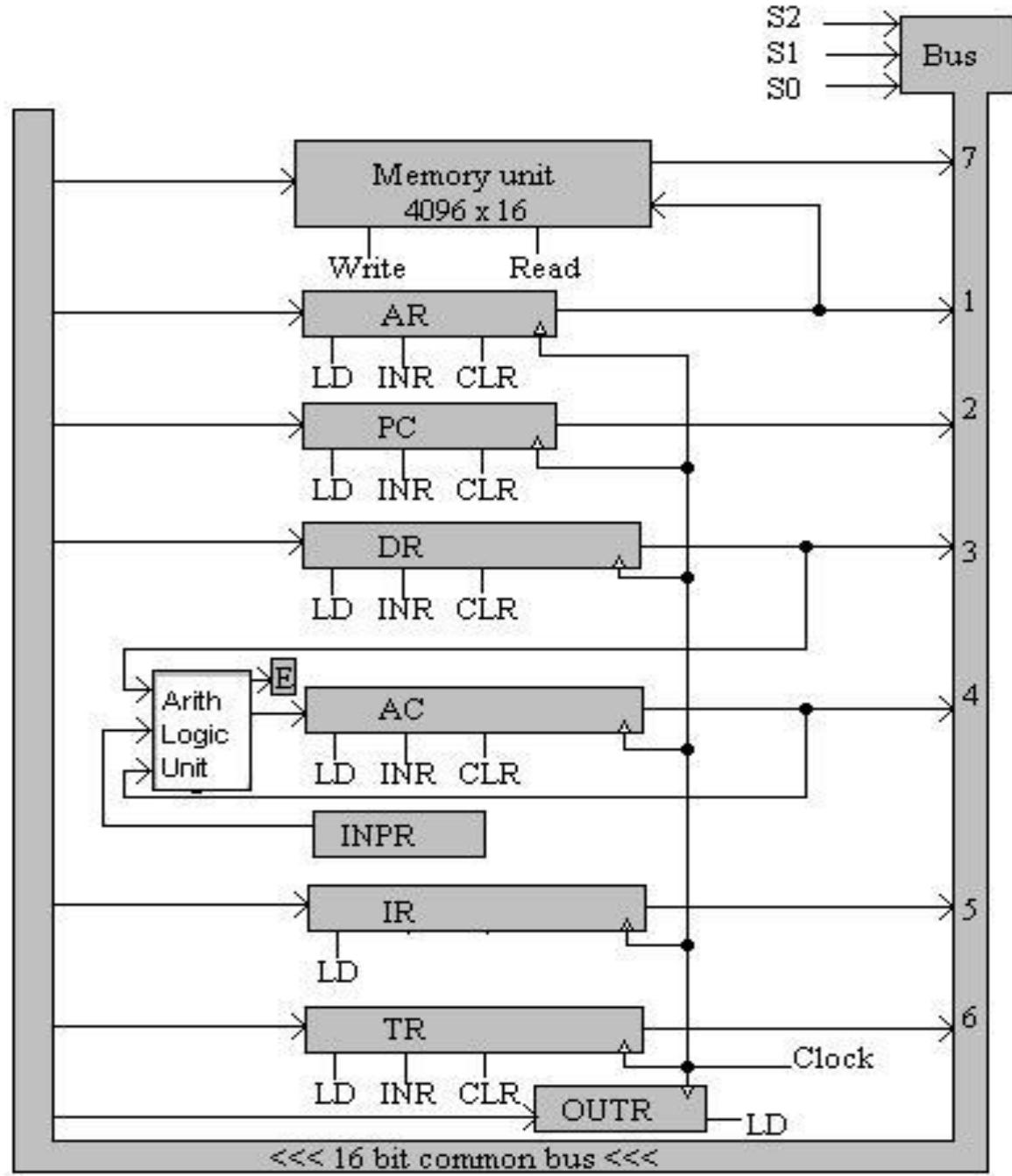
- **General Process of Operations in a CPU:**

- **Memory to Register**: Data is fetched from memory and moved into a register.
- **Register to ALU**: The register sends the data to the ALU (Arithmetic Logic Unit) for processing.
- **ALU Performs Operation**: The ALU carries out the required arithmetic or logical operation.
- **Result to Register**: The result of the operation is placed back into a register.
- **Register to Memory**: Finally, the result in the register is written back to memory if needed.

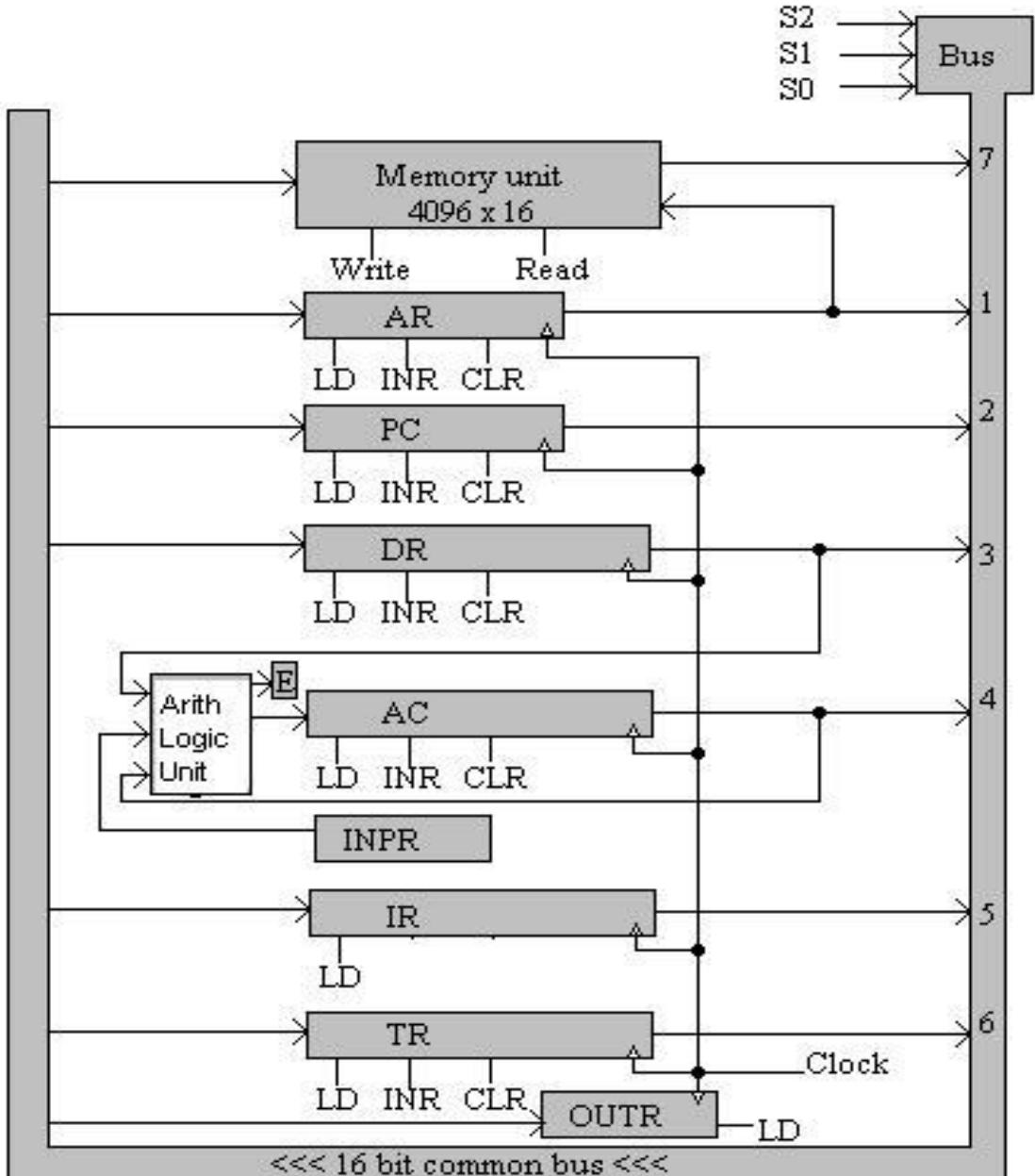


**Registers** - Registers refer to high-speed storage areas in the CPU. The data processed by the CPU are fetched from the registers. There are different types of registers used in architecture.

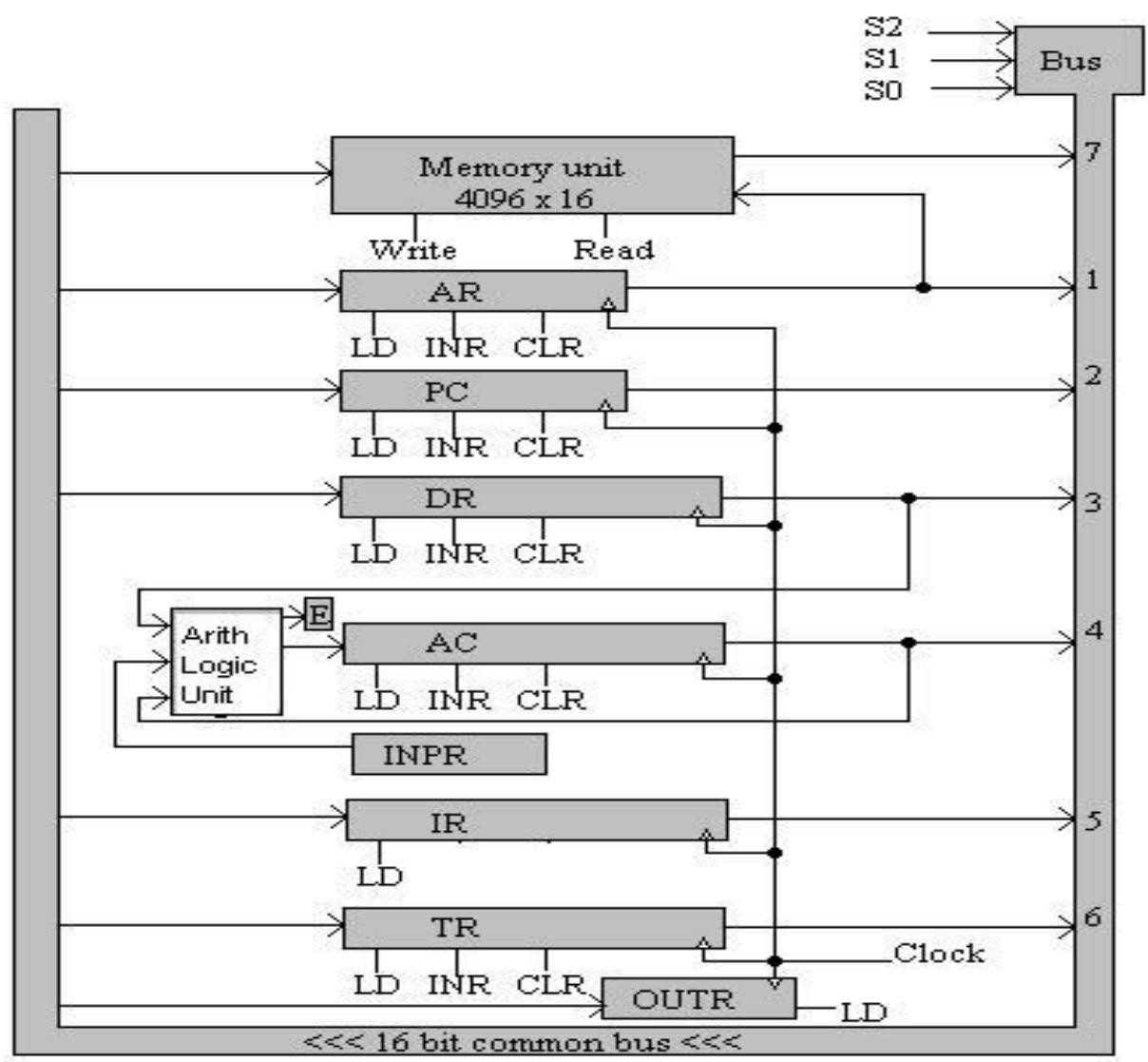
Register Number symbol	of bits	Register name	Function
<i>DR</i>	16	Data register	Holds memory operand
<i>AR</i>	12	Address register	Holds address for memory
<i>AC</i>	16	Accumulator	Processor register
<i>IR</i>	16	Instruction register	Holds instruction code
<i>PC</i>	12	Program counter	Holds address of instruction
<i>TR</i>	16	Temporary register	Holds temporary data
<i>INPR</i>	8	Input register	Holds input character
<i>OUTR</i>	8	Output register	Holds output character



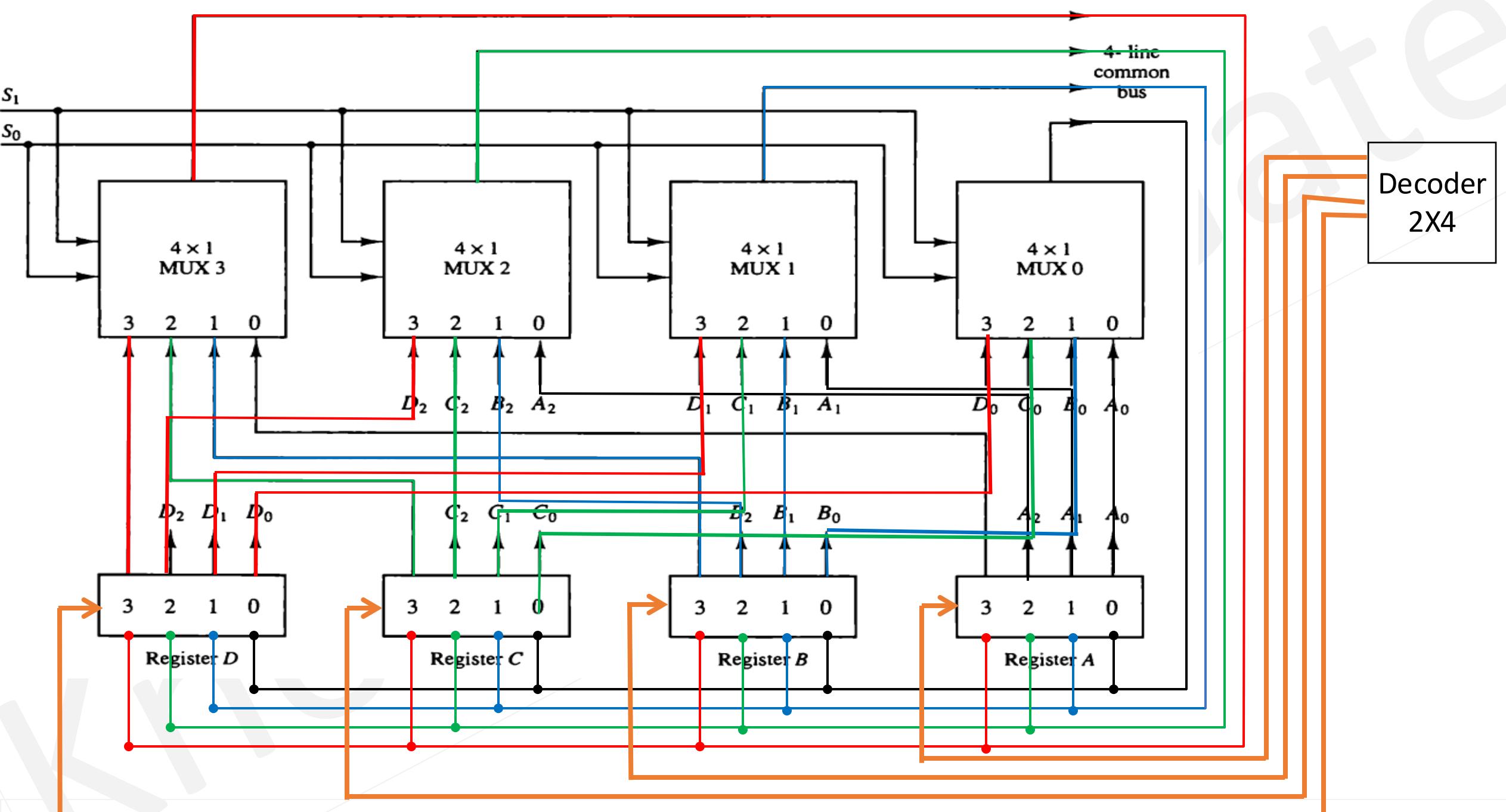
- The **data register (DR)** is a register which is used to store any data that is to be transferred to memory or which are fetched from memory.
- The **accumulator (AC)** register is a general-purpose processing register. Will hold the intermediate arithmetic and logic results of the operation performed in the ALU, as ALU is not directly connected to the memory.
- **Instruction register-** is used to store instruction which we fetched from memory, so that we analyses the instruction using a decoder and can understand what instruction want to perform.
- The **temporary register (TR)** is used for holding temporary data during the processing.

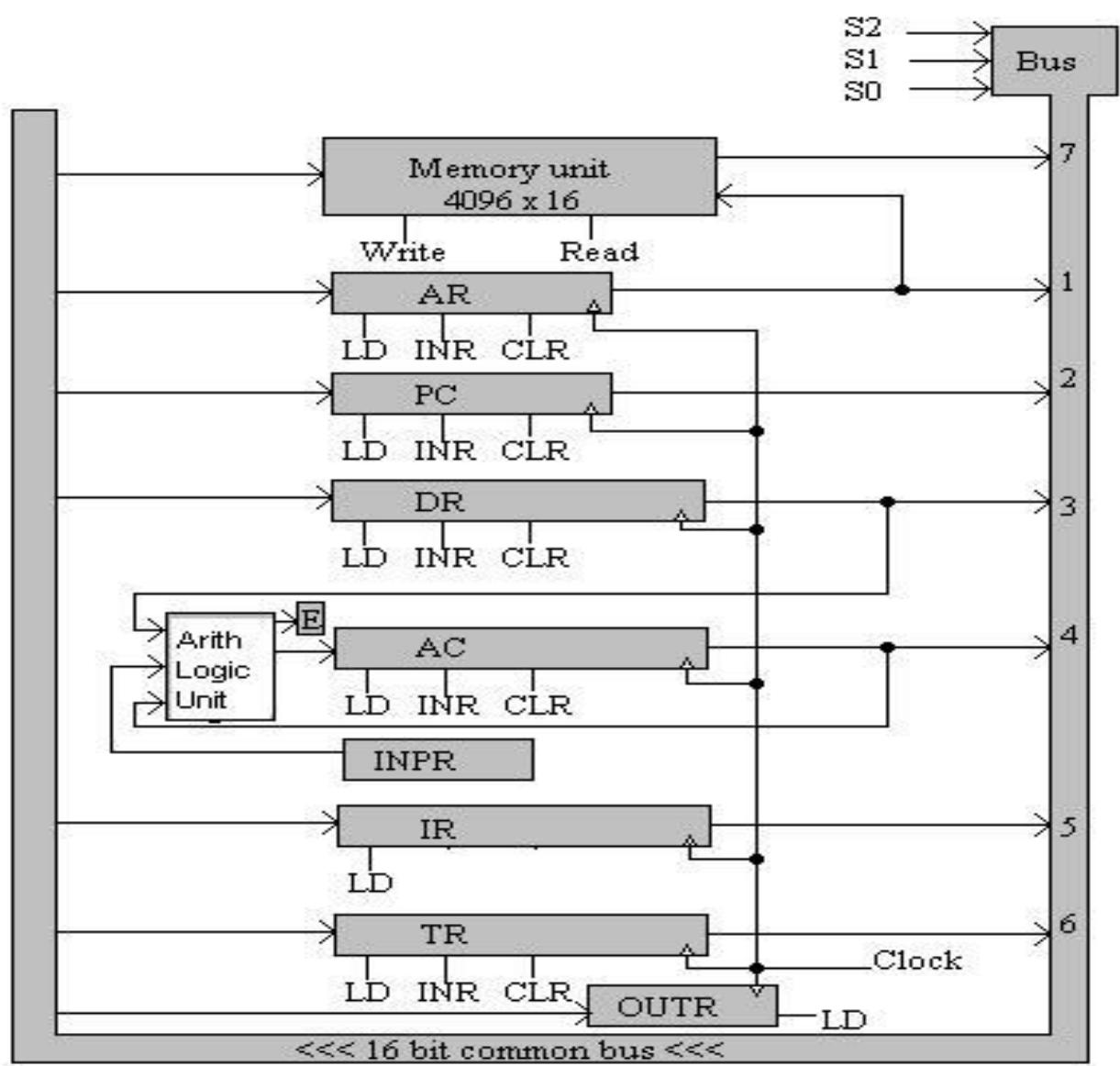


- The memory **address register (AR)** has 12 bits since this is the width of a memory address (always used least significant bits of bus). It stores the memory locations of instructions or data that need to be fetched from memory or stored in memory.
- The **program counter (PC)** also has 12 bits and it holds the address of the next instruction to be read from memory after the current instruction is executed. The PC goes through a counting sequence and causes the computer to read sequential instructions previously stored in memory.
- Two registers are used for input and output. The **input register (INPR)** receives an 8-bit character from an input device, and pass it to ALU then accumulator and then to memory.
- The output **register (OUTR)** holds an 8-bit character for an output device, screen, printer etc.

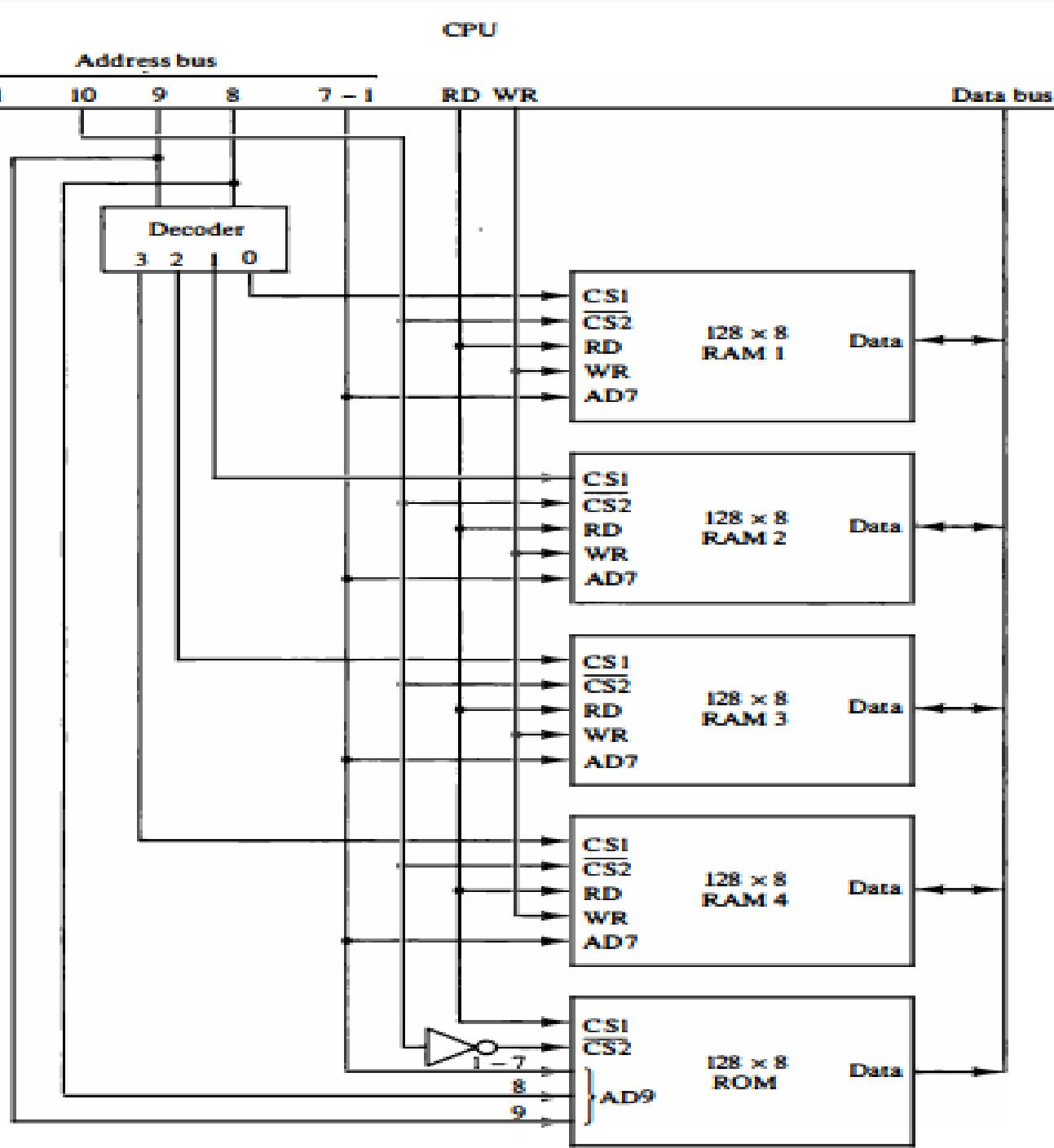


- The outputs of seven registers and memory are connected to the common bus.
- The specific output that is selected for the bus lines at any given time is determined from the binary value of the selection variables  $S_2$ ,  $S_1$ , and  $S_0$ .
- The number along each output shows the decimal equivalent of the required binary selection.
- Example: the number along the output of DR is 3. The 16-bit outputs of DR are placed on the bus lines when  $S_2S_1S_0 = 011$ .



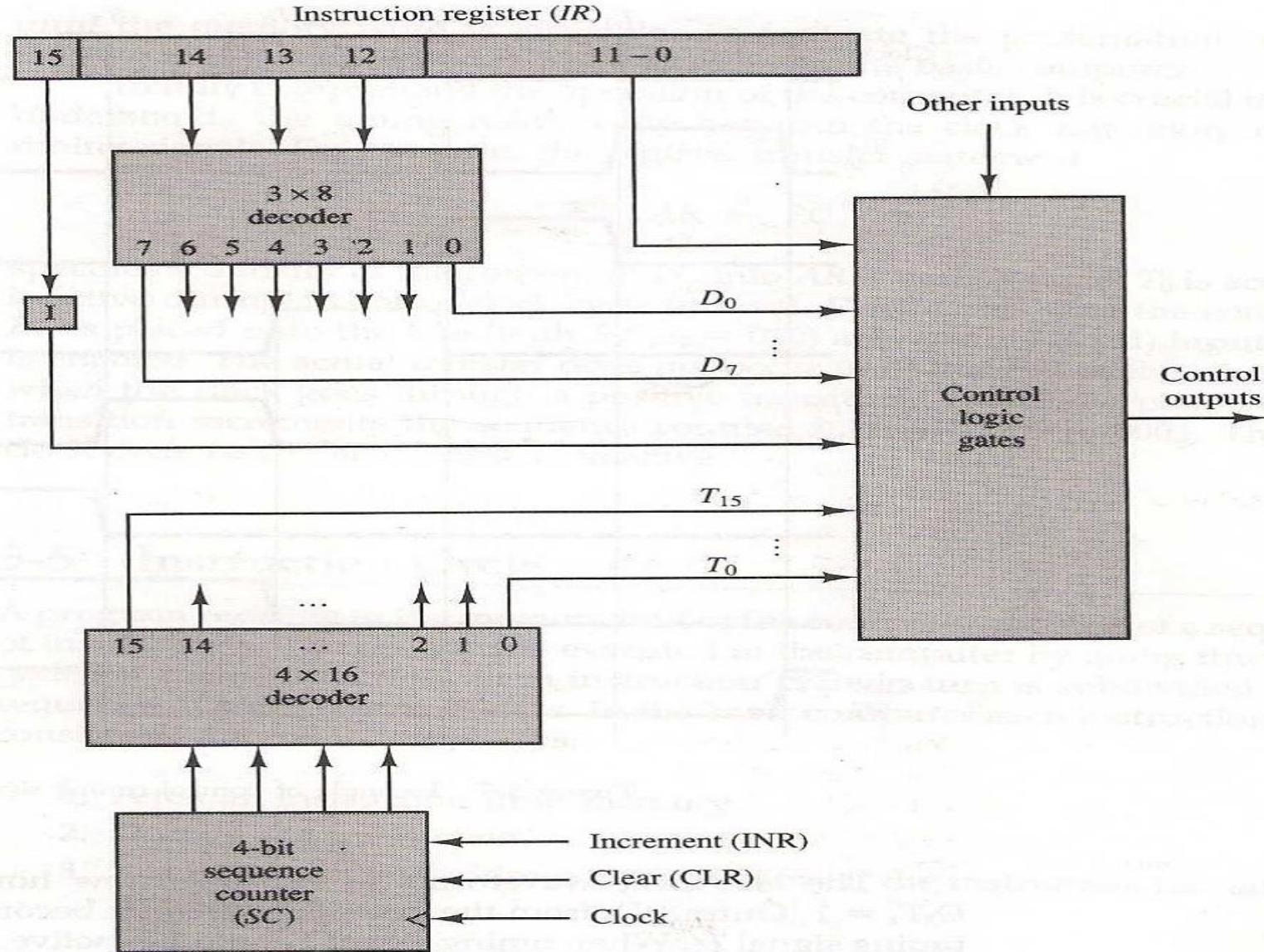


- The particular register whose LD (load) input is enabled receives the data from the bus during the next clock pulse transition. The memory receives the contents of the bus when its write input is activated. The memory places its 16-bit output onto the bus when the read input is activated and  $S_2S_1S_0 = 111$ .
- The input register INPR and the output register OUTR have 8 bits each and communicate with the eight least significant bits in the bus. INPR is connected to provide information to the bus but OUTR can only receive information from the bus. There is no transfer from OUTR to any of the other registers.
- Some additional points to notice
  - The 16 lines of the common bus receive information from six registers and the memory unit.
  - The bus lines are connected to the inputs of six registers and the memory.
  - Five registers have three control inputs: LD (load), INR (increment), and CLR (clear)



## Timing Circuit

- In order to perform a instruction we need to perform a number of micro-operators, and these micro-operations must be timed
  - $AR \leftarrow PC$
  - $IR \leftarrow M[AR]$ ,  $PC \leftarrow PC + 1$
- We should distinguish one clock pulse from another during the execution of instruction.
- Sequence counter followed by decoder is used to generate time signals



**Instruction Cycle** - A program residing in the memory unit of the computer consists of a sequence of instructions. The program is executed in the computer by going through a cycle for each instruction. Each instruction cycle in turn is subdivided into a sequence of sub cycles or phases. from fetching of instruction to the completion of execution of instruction whatever happens is called instruction cycle. The reason we called this cycle because it will happen for every instruction.

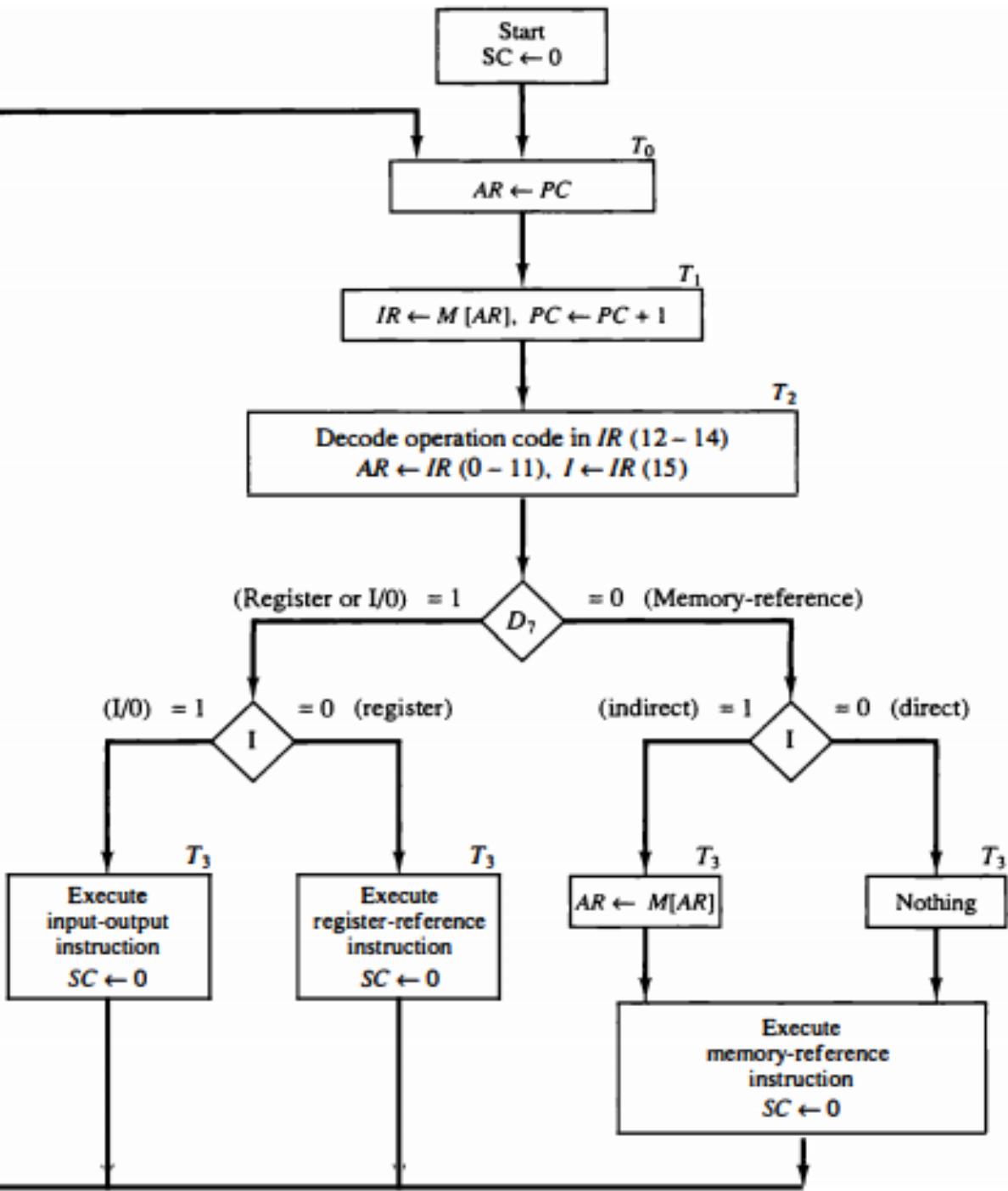
- Each instruction cycle consists of the following phases:
  - Fetch an instruction from memory.
  - Decode the instruction.
  - Read the effective address from memory if the instruction has an indirect address.
  - Execute the instruction.
  - Fetch and Decode

- **Program Counter (PC)**: Initially, the **PC** is loaded with the address of the first instruction in the program.
- **Sequence Counter (SC)**: The **SC** is cleared to 0, generating the timing signal **T0**. After each clock pulse, **SC** increments by one, producing a sequence of timing signals **T0, T1, T2**, and so on.
- **Fetch Phase**:
  - At **T0**, the address in **PC** is transferred to the **Address Register (AR)**, since **AR** is the only register connected to the memory's address inputs.
  - At **T1**, the instruction is read from memory and placed into the **Instruction Register (IR)**. Simultaneously, the **PC** increments by one, preparing for the next instruction.
- **Decode Phase**:
  - At **T2**, the operation code in **IR** is decoded. The indirect bit is transferred to flip-flop **I**, and the address part of the instruction is transferred to **AR**.
- **Sequential Timing**: After each clock pulse, the **SC** increments, moving the sequence through **T0, T1, and T2**, which coordinate the phases of fetching, decoding, and preparing for the next instruction.

**T<sub>0</sub>:**  $AR \leftarrow PC$

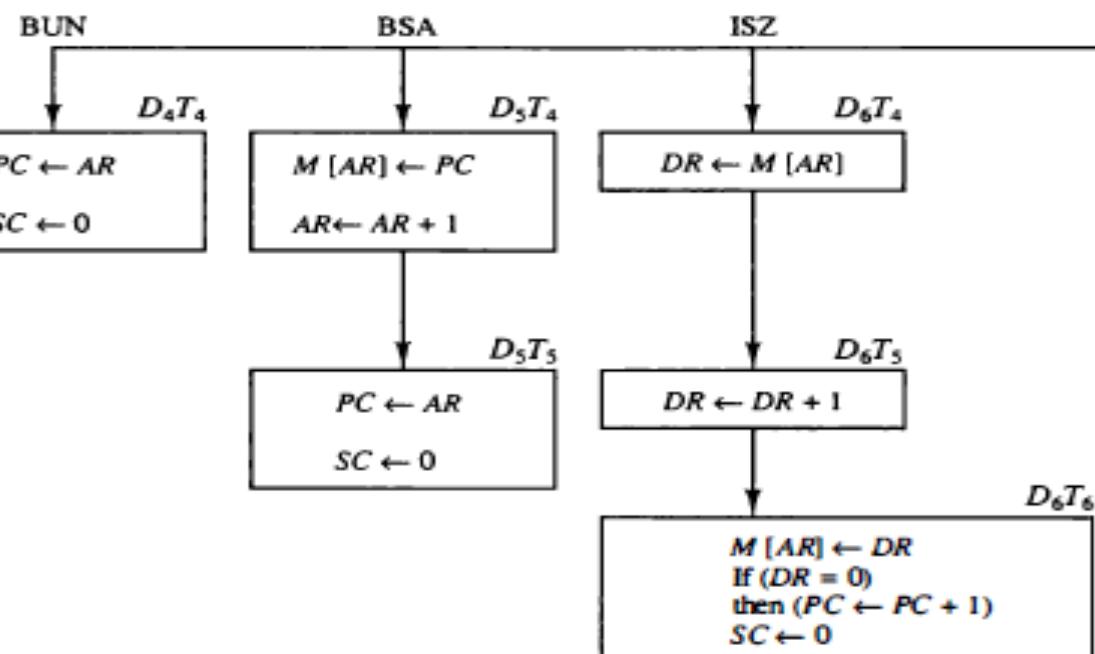
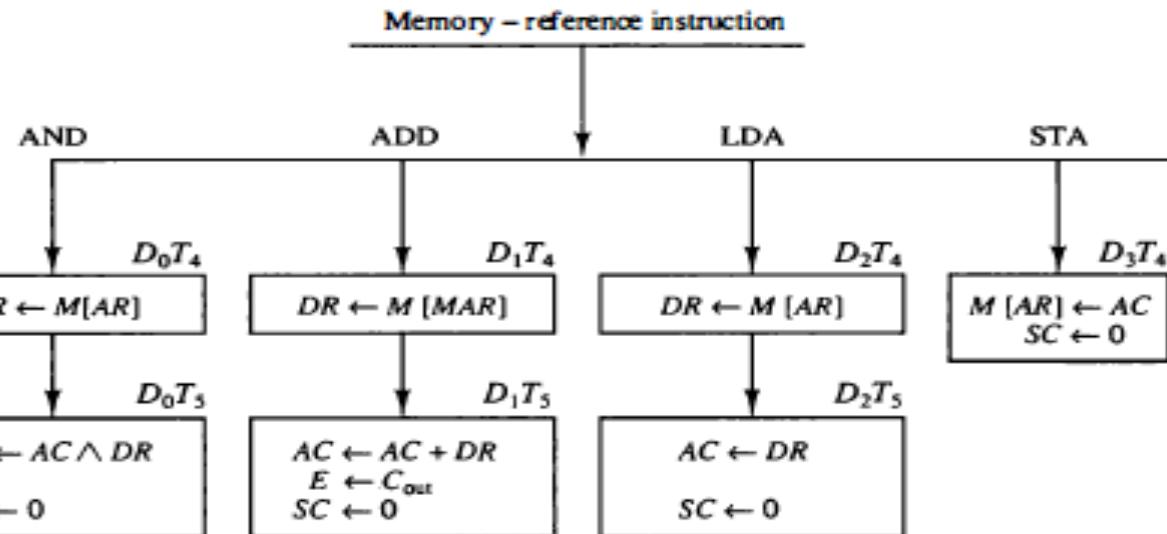
**T<sub>1</sub>:**  $IR \leftarrow M[AR]$ ,  $PC \leftarrow PC + 1$

**T<sub>2</sub>:**  $D_0, \dots, D_7 \leftarrow \text{Decode } IR(12-14)$ ,  $AR \leftarrow IR(0-11)$ ,  $I \leftarrow IR(15)$



- Timing Signal T3:** After decoding, the active timing signal is **T3**, where the control unit determines the type of instruction retrieved from memory.
- Instruction Type Check:**
  - Decoder Output (D7):** If the decoder output **D7 = 1**, the instruction is either a **register-reference** or **input-output** instruction.
  - If **D7 = 0**, the instruction is a **memory-reference** instruction, with the operation code ranging between binary values **000** to **110**.
- Indirect Address Check:**
  - If **D7 = 0** and the first bit (stored in flip-flop **I**) equals **1**, it indicates a **memory-reference instruction with an indirect address**.
  - In this case, the effective address must be read from memory, represented by the microoperation:
    - AR ← M[AR]:** Transfer the content at the memory address held in **AR** to **AR** itself.
- Direct Addressing:**
  - If **I = 0**, the instruction is a **direct memory-reference**, and no additional memory read is necessary since the effective address is already in **AR**.
- Sequence Control:**
  - After determining the instruction type, the sequence counter (**SC**) is either incremented or cleared to **0** with each positive clock transition, preparing the CPU for the next operation.

# Memory-Reference Instructions



Symbol	Operation decoder	Symbolic description
AND	$D_0$	$AC \leftarrow AC \wedge M[AR]$
ADD	$D_1$	$AC \leftarrow AC + M[AR], E \leftarrow C_{out}$
LDA	$D_2$	$AC \leftarrow M[AR]$
STA	$D_3$	$M[AR] \leftarrow AC$
BUN	$D_4$	$PC \leftarrow AR$
BSA	$D_5$	$M[AR] \leftarrow PC, PC \leftarrow AR + 1$
ISZ	$D_6$	$M[AR] \leftarrow M[AR] + 1,$ If $M[AR] + 1 = 0$ then $PC \leftarrow PC + 1$

- AND to AC - This is an instruction that performs the AND logic operation on pairs of bits in AC and the memory word specified by the effective address. The result of the operation is transferred to AC. The microoperations that execute this instruction are:

$D_0T_4: DR \leftarrow M[AR]$

$D_0T_5: AC \leftarrow AC \wedge DR, SC \leftarrow 0$

- ADD to AC - This instruction adds the content of the memory word specified by the effective address to the value of AC. The sum is transferred into AC and the output carry Cout is transferred to the E (extended accumulator) flip-flop. The microoperations needed to execute this instruction are:

$D_1T_4: DR \leftarrow M[AR]$

$D_1T_5: AC \leftarrow AC + DR, E \leftarrow C_{out}, SC \leftarrow 0$

- LDA: Load to AC - This instruction transfers the memory word specified by the effective address to AC. The microoperations needed to execute this instruction are

$D_2T_4: DR \leftarrow M[AR]$

$D_2T_5: AC \leftarrow DR, SC \leftarrow 0$

- STA: Store AC - This instruction stores the content of AC into the memory word specified by the effective address. Since the output of AC is applied to the bus and the data input of memory is connected to the bus, we can execute this instruction with one microoperation:  
 $D_3T_4: M[AR] \leftarrow AC, SC \leftarrow 0$
- BUN: Branch Unconditionally - This instruction transfers the program to the instruction specified by the effective address. The BUN instruction allows the programmer to specify an instruction out of sequence and we say that the program branches (or jumps) unconditionally. The instruction is executed with one microoperation:

$D_4T_4: PC \leftarrow AR, SC \leftarrow 0$

- BSA: Branch and Save Return Address - This instruction is useful for branching to a portion of the program called a subroutine or procedure

$M[AR] \leftarrow PC, PC \leftarrow AR + 1$

- ISZ: Increment and Skip if Zero - This instruction increments the word specified by the effective address, and if the incremented value is equal to 0, PC is incremented by 1.

$D_6T_4: DR \leftarrow M[AR]$

$D_6T_5: DR \leftarrow DR + 1$

$D_6T_6: M[AR] \leftarrow DR, \text{ if } (DR = 0) \text{ then } (PC \leftarrow PC + 1), SC \leftarrow 0$

## Register-Reference Instructions

- Register-reference instructions are recognized by the control when  $D_7 = 1$  and  $I = 0$ .
  - These instructions use bits 0 through 11 of the instruction code to specify one of 12 instructions. These 12 bits are available in  $IR(0-11)$ .
  - These instructions are executed with the clock transition associated with timing variable  $T_3$ .
  - Each control function needs the Boolean relation  $D_7 I' T_3$ .
- 
- 

$D_7 I' T_3 = r$  (common to all register-reference instructions)

$IR(i) = B_i$  [bit in  $IR(0-11)$  that specifies the operation]

	$r: SC \leftarrow 0$	Clear $SC$
CLA	$rB_{11}: AC \leftarrow 0$	Clear $AC$
CLE	$rB_{10}: E \leftarrow 0$	Clear $E$
CMA	$rB_9: AC \leftarrow \overline{AC}$	Complement $AC$
CME	$rB_8: E \leftarrow \overline{E}$	Complement $E$
CIR	$rB_7: AC \leftarrow \text{shr } AC, AC(15) \leftarrow E, E \leftarrow AC(0)$	Circulate right
CIL	$rB_6: AC \leftarrow \text{shl } AC, AC(0) \leftarrow E, E \leftarrow AC(15)$	Circulate left
INC	$rB_5: AC \leftarrow AC + 1$	Increment $AC$
SPA	$rB_4: \text{If } (AC(15) = 0) \text{ then } (PC \leftarrow PC + 1)$	Skip if positive
SNA	$rB_3: \text{If } (AC(15) = 1) \text{ then } (PC \leftarrow PC + 1)$	Skip if negative
SZA	$rB_2: \text{If } (AC = 0) \text{ then } PC \leftarrow PC + 1$	Skip if $AC$ zero
SZE	$rB_1: \text{If } (E = 0) \text{ then } (PC \leftarrow PC + 1)$	Skip if $E$ zero
HLT	$rB_0: S \leftarrow 0$ ( $S$ is a start-stop flip-flop)	Halt computer

## Input-Output Configuration

- **Serial Communication:** The terminal handles data in a serial format, with each piece of data consisting of eight bits representing an alphanumeric character.
- **Input Register (INPR):** Serial data from the keyboard is stored in the **INPR** register.
- **Output Register (OUTR):** Serial data intended for the printer is stored in the **OUTR** register.
- **Communication:** These registers communicate serially with the external devices (keyboard and printer) and interact with the **Accumulator (AC)** in parallel within the system.

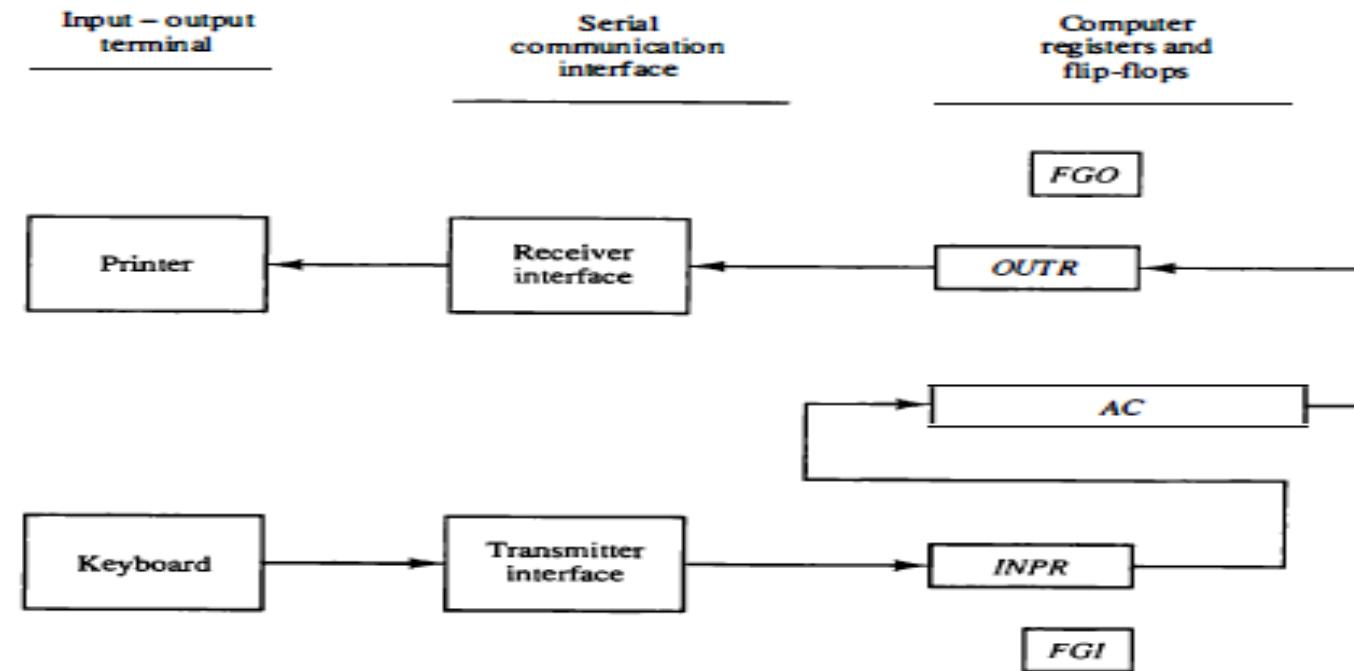
$$D_7IT_3 = p$$

$$IR(i) = B_i, i = 6, \dots, 11$$

INP	p: $SC \leftarrow 0$ pB <sub>11</sub> : $AC(0-7) \leftarrow INPR, FGI \leftarrow 0$	Clear SC Input char. to AC
OUT	pB <sub>10</sub> : $OUTR \leftarrow AC(0-7), FGO \leftarrow 0$	Output char. from AC
SKI	pB <sub>9</sub> : if( $FGI = 1$ ) then ( $PC \leftarrow PC + 1$ )	Skip on input flag
SKO	pB <sub>8</sub> : if( $FGO = 1$ ) then ( $PC \leftarrow PC + 1$ )	Skip on output flag
ION	pB <sub>7</sub> : $IEN \leftarrow 1$	Interrupt enable on
IOF	pB <sub>6</sub> : $IEN \leftarrow 0$	Interrupt enable off

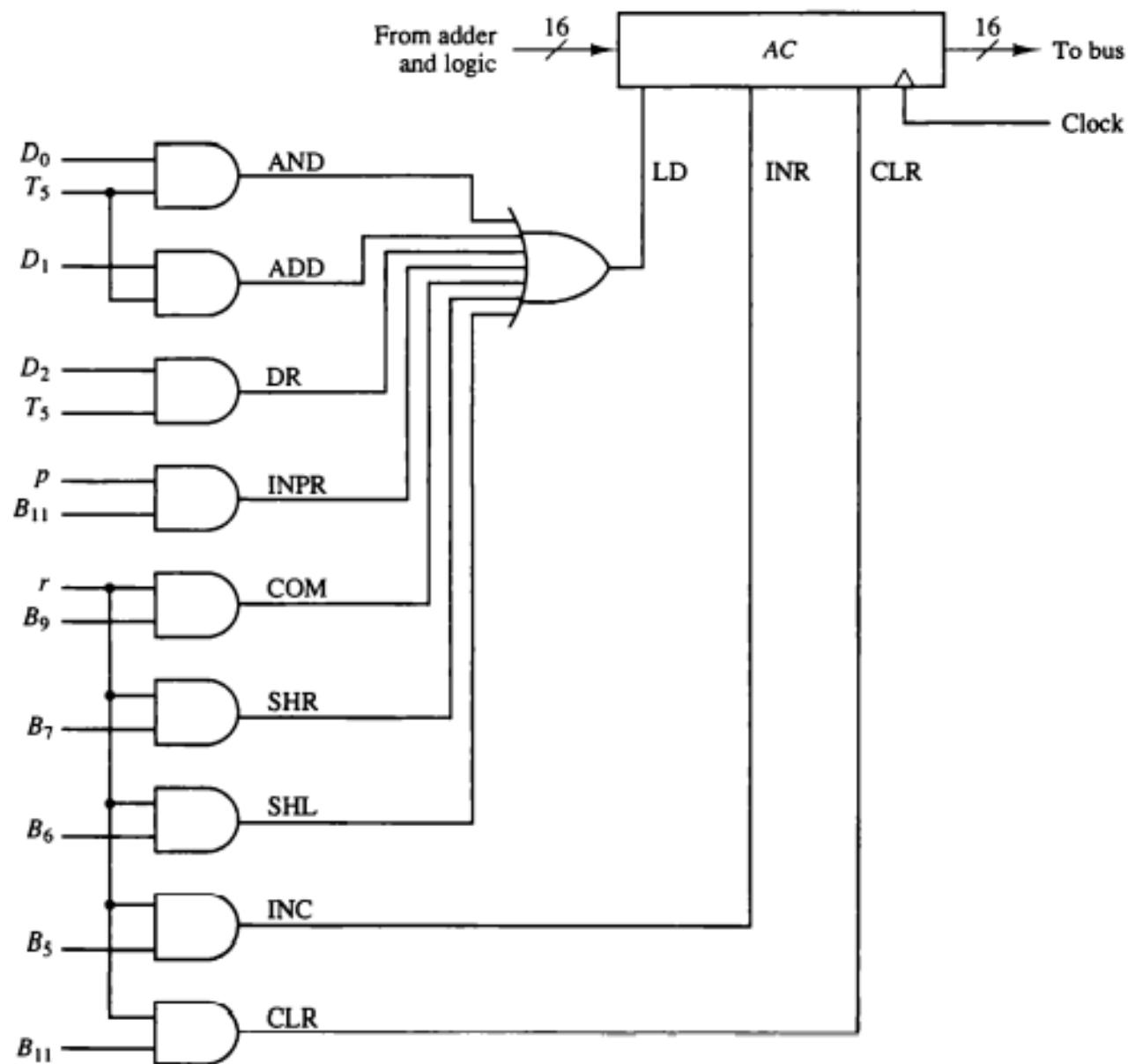
## Input-Output Configuration

- **Serial Communication:** The terminal sends and receives serial data, with each piece of information containing eight bits of an alphanumeric code.
- **INPR and OUTR: INPR (Input Register):** Serial data from the keyboard is shifted into this register. **OUTR (Output Register):** Stores serial data intended for the printer.
- **Communication Interface:** These registers communicate serially with external devices (keyboard and printer) and in parallel with the **Accumulator (AC)**.
- **Transmitter and Receiver Interfaces:** The **transmitter interface** receives serial data from the keyboard and passes it to **INPR**. The **receiver interface** takes data from **OUTR** and sends it serially to the printer.
- **Control Flag:** The **Input Flag (FGI)** is a 1-bit control flip-flop. The flag is set to **1** when new data is available in the input device and cleared to **0** when the data is accepted by the computer. The flag helps synchronize the timing difference between the input device and the computer, ensuring smooth data transfer.



### Hexadecimal code

Symbol	<i>I</i> = 0	<i>I</i> = 1	Description
AND	0xxx	8xxx	AND memory word to <i>AC</i>
ADD	1xxx	9xxx	Add memory word to <i>AC</i>
LDA	2xxx	Axxx	Load memory word to <i>AC</i>
STA	3xxx	Bxxx	Store content of <i>AC</i> in memory
BUN	4xxx	Cxxx	Branch unconditionally
BSA	5xxx	Dxxx	Branch and save return address
ISZ	6xxx	Exxx	Increment and skip if zero
CLA	7800		Clear <i>AC</i>
CLE	7400		Clear <i>E</i>
CMA	7200		Complement <i>AC</i>
CME	7100		Complement <i>E</i>
CIR	7080		Circulate right <i>AC</i> and <i>E</i>
CIL	7040		Circulate left <i>AC</i> and <i>E</i>
INC	7020		Increment <i>AC</i>
SPA	7010		Skip next instruction if <i>AC</i> positive
SNA	7008		Skip next instruction if <i>AC</i> negative
SZA	7004		Skip next instruction if <i>AC</i> zero
SZE	7002		Skip next instruction if <i>E</i> is 0
HLT	7001		Halt computer
INP	F800		Input character to <i>AC</i>
OUT	F400		Output character from <i>AC</i>
SKI	F200		Skip on input flag
SKO	F100		Skip on output flag
ION	F080		Interrupt on
IOF	F040		Interrupt off



**Q** Consider a hypothetical control unit implemented by hardware it uses three, 8-bits register A, B, C. It supports the two instruction  $I_1$  and  $I_2$ , the following table gives control signals required for each micro-operation for both instructions?

Micro-operation	Control Signal	
	$I_1$	$I_2$
$T_1$	$A_{in}, B_{out}$	$A_{in}, A_{out}$
$T_2$	$B_{in}, C_{out}$	$C_{in}, A_{out}$
$T_3$	$B_{in}, B_{out}$	$C_{in}, C_{out}$
$T_4$	$A_{in}, A_{out}$	$A_{in}, B_{out}$

## Designing of Control Unit

- **Control Signal Generation:** Control signals in a computer system can be generated in two ways:
  - **Hardware Control Unit Design:** Control signals are generated using digital hardware, where each signal is expressed as a **Sum of Products (SOP)** expression and realized through logic gates.
  - **Micro-Program Control Unit Design:** Control signals are generated using a combination of **memory** and **hardware**. This approach stores control instructions in memory and executes them via hardware.

- **Hardwired Control Unit Summary:**

- **Operation Determination**: The sequence of operations in a **hardwired control unit** is determined by the wiring of logic elements, hence the name "hardwired."
- **Logic Circuits**: Control signals are generated by **fixed logic circuits** that correspond directly to **Boolean expressions**.
- **Speed**: This is the **fastest control unit design**, making it ideal for **real-time applications**. For example, **RISC (Reduced Instruction Set Computer) architecture** employs hardwired control units. The speed of this control method is superior to micro-programmed control units.
- **Limitation**: The hardwired approach lacks flexibility. Any modification requires a complete redesign and rewiring of the hardware components, making it unsuitable for environments that require frequent changes, debugging, or flexibility.
- **Solution**: The inflexibility of the hardwired control unit is resolved by using a **micro-programmed control unit**, which allows easier modification and debugging.

## Micro-Programmed Control Unit

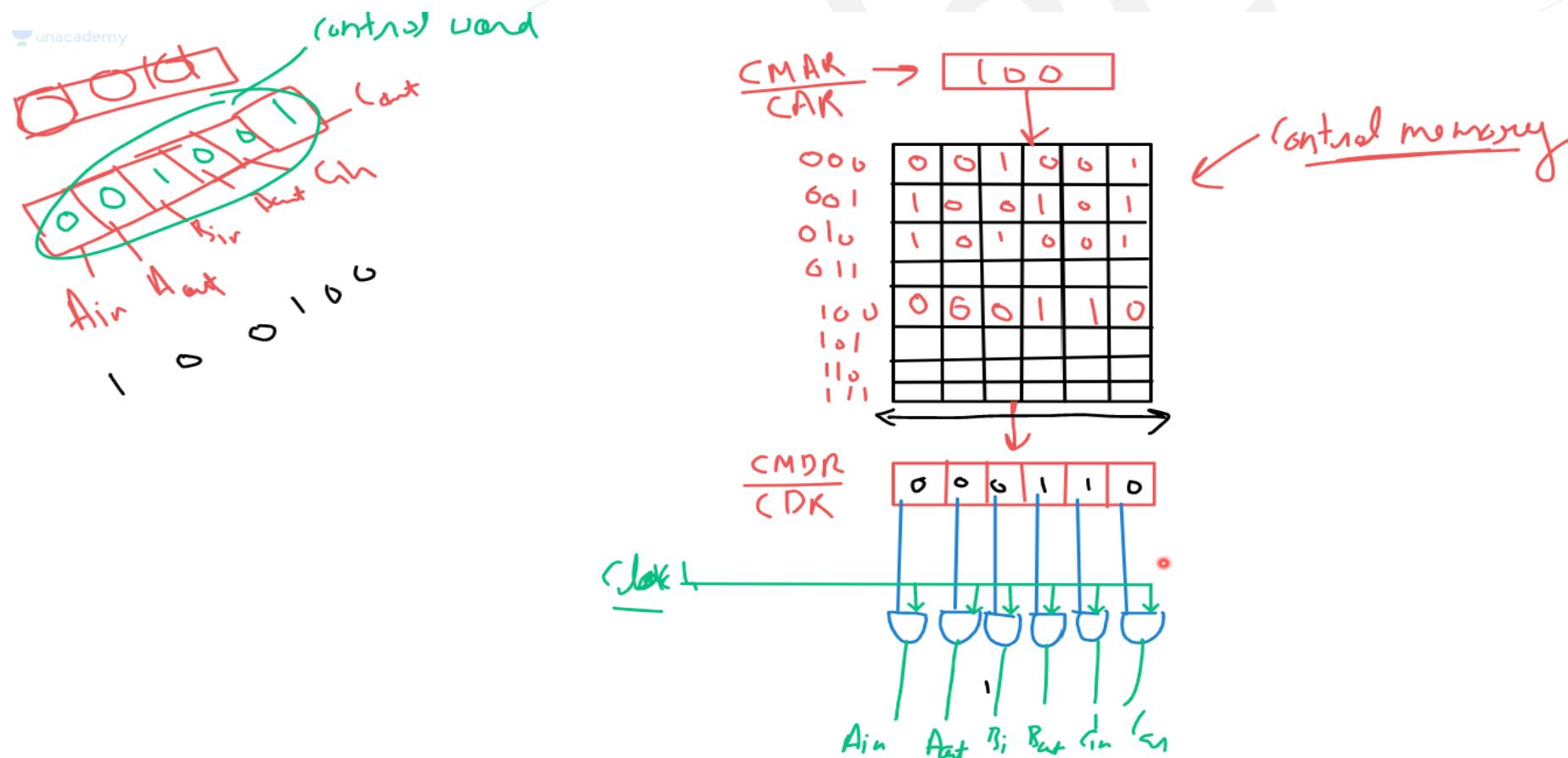
- **Introduction:** The concept of microprogramming was introduced by **Maurice Wilkes** in 1951 as an intermediate level to control the execution of program instructions.
- **Microprograms:** Organized as a sequence of **microinstructions** stored in special **control memory**. These microinstructions guide the operation of the computer.
- **Advantages:** The **main advantage** of the microprogrammed control unit is the **simplicity** of its structure. Microinstructions (control signals) can be easily modified by updating the control memory, without changing the hardware.



- **Operation:** The **binary patterns** of control signals are stored in control memory. When a control word is accessed from memory, hardware generates the required control signals.
- **Flexibility:** This design is more flexible than hardwired control units, as changes are made by updating the control word rather than altering hardware. It is widely used in **CISC (Complex Instruction Set Computing)** systems.
- **Types of Micro-Programming:**
  - **Horizontal Micro-Programming:** Control signals are stored with more detail, leading to wider control words.
  - **Vertical Micro-Programming:** Control signals are stored more compactly, leading to shorter control words.

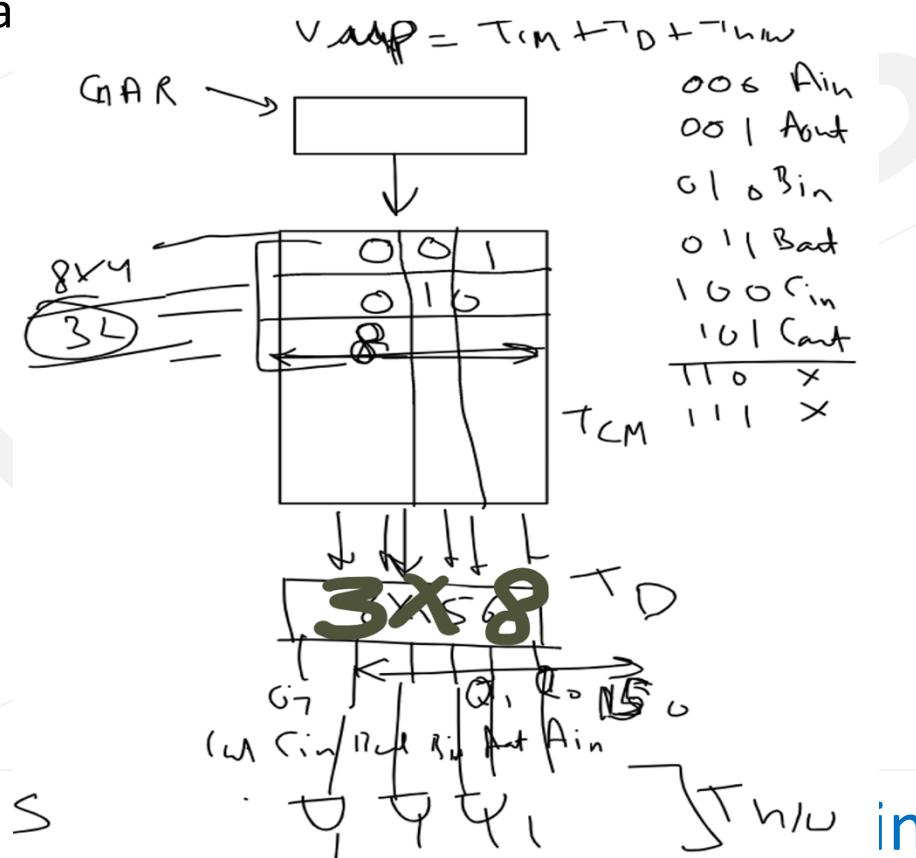
## Horizontal micro-Program

- **Parallelism:** Horizontal micro-programming provides a **higher degree of parallelism**, making it suitable for **multi-processor systems**.
- **Control Word:** It requires more bits for the control word, as each control signal is represented by a single bit (1 bit per control signal).



# Vertical Micro-program

- **Control Word Size:** Vertical micro-programming reduces the size of control words by encoding the control signal patterns before storing them in control memory. This method compresses the control signals, making it more efficient in terms of memory usage.
- **Flexibility:** Vertical micro-programming offers **more flexibility** than horizontal micro-programming because of its ability to encode signals.
- **Parallelism:** The degree of parallelism is limited to **1**, due to the use of a **decoder**, meaning operations are generally executed sequentially



# Conclusion

- In many applications, a **combination of Horizontal and Vertical Micro-programmed Control Units** is preferred. This hybrid approach leverages the strengths of both methods—horizontal micro-programming for parallelism and vertical micro-programming for efficiency and flexibility—resulting in an optimal control unit design for various computational tasks.

# Control word sequencing

- To implement a micro-program, **control words** are read sequentially from control memory. The sequencing of control words is managed using an **address instruction**, which ensures that the correct control words are executed in the right order during the micro-program's operation.

Flag	Control Signal	Address
------	----------------	---------

Q Let Micro-programmed control unit has to be support 256 instruction, each of which on average takes 16 micro-operation. The system has to be support 16 flag conditions and generate 48 control signals. if horizontal micro-programming is used

- 1) how many bits are required for each control word
- 2) what is the control memory size in byte.

Q A microprogram control unit having the behaviours such that the control signals are divided into 10 mutually exclusive groups. the no of signals for each group is given below

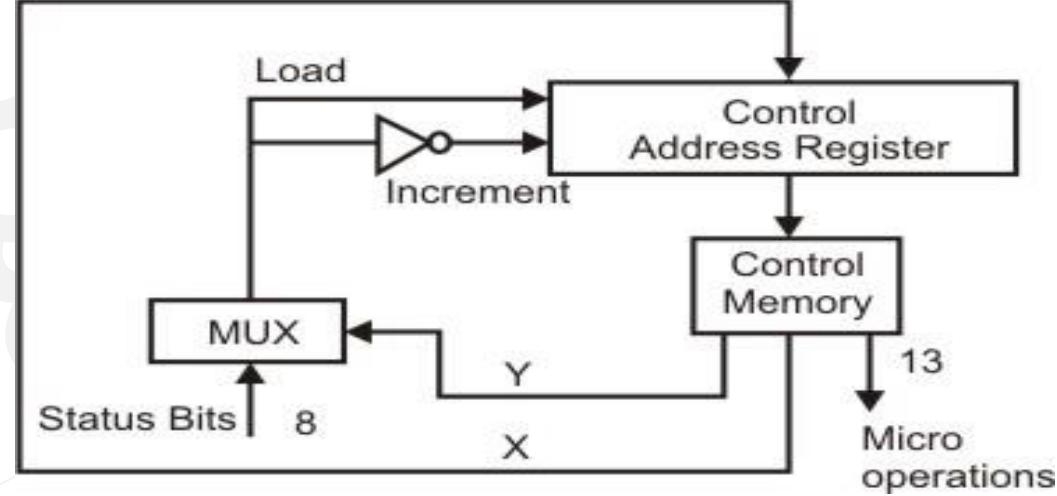
Group	G1	G2	G3	G4	G5	G6	G7	G8	G9	G10
Signal	3	12	7	5	11	11	14	3	5	3

Q Consider a CPU where all the instructions require 7 clock cycles to complete execution. There are 140 instructions in the instruction set. It is found that 125 control signals are needed to be generated by the control unit. While designing the horizontal microprogrammed control unit, single address field format is used for branch control logic. What is the minimum size of the control word and control address register? (Gate-2008) (2 Marks)

- (A) 125, 7      (B) 125, 10      (C) 135, 7      (D) 135, 10

**Q** The microinstructions stored in the control memory of a processor have a width of 26 bits. Each microinstruction is divided into three fields: a micro-operation field of 13 bits, a next address field (X), and a MUX select field (Y). There are 8 status bits in the inputs of the MUX. How many bits are there in the X and Y fields, and what is the size of the control memory in number of words? **(GATE-2004) (2 Marks)**

- (A) 10, 3, 1024      (B) 8, 5, 256      (C) 5, 8, 2048      (D) 10, 3, 512



# Reduced Instruction Set Computer (RISC)

## Basic Characteristics:

- **Fewer Instructions:** RISC processors use a small set of simple, general-purpose instructions compared to the complex instruction sets of CISC processors.
- **Fewer Addressing Modes:** Memory access is limited to load and store operations, making memory interaction more efficient.
- **Registers Focus:** All operations are performed within CPU registers, minimizing memory access.
- **Fixed-Length Instructions:** Instructions are of a fixed length, making them easier to decode and often allowing for single-cycle execution.
- **Hardwired Control:** RISC processors typically use hardwired control rather than microprogramming, which increases execution speed.

- **Load/Store Architecture:** The architecture limits memory access to specific instructions (load and store), ensuring that most operations are performed directly within the CPU's registers.
- **Historical Context:**
  - **Development:** The RISC concept gained traction in the 1980s through projects at **Stanford University** (MIPS architecture) and **University of California, Berkeley** (RISC architecture).
  - **Commercialization:** Stanford's MIPS and Berkeley's RISC were successfully commercialized as **MIPS** and **SPARC**, respectively.
  - **IBM's Contribution:** IBM's efforts during this period led to the development of **IBM POWER**, **PowerPC**, and **Power ISA** architectures.

- **Modern Usage:**
  - **Variety of Designs:** RISC architectures include **ARC, ARM, Alpha, MIPS, SPARC**, and others.
  - **Widespread Adoption:** RISC processors are widely used in modern devices, from **smartphones** and **tablets** (using ARM architecture) to **supercomputers** like **Summit**, which is one of the fastest supercomputers in the world as of 2020.



## Complex Instruction Set Computer (CISC)

- **Definition:** A **Complex Instruction Set Computer (CISC)** is a type of computer architecture where single instructions can execute multiple low-level operations (e.g., memory loads, arithmetic operations, memory stores) or can handle multi-step operations within a single instruction.
- **Characteristics:**
  - **Large Instruction Set:** Typically includes between **100 to 250 instructions**.
  - **Specialized Instructions:** Some instructions are highly specialized but used infrequently.
  - **Variety of Addressing Modes:** Typically, CISC systems offer **5 to 20 different addressing modes**.
  - **Variable-Length Instructions:** Instructions have different lengths and formats.
  - **Memory Operand Manipulation:** Instructions can manipulate data directly in memory, unlike RISC, where load/store operations are separate.

- **Historical Context:**
  - The term **CISC** was coined in contrast to **RISC (Reduced Instruction Set Computer)** and has since been used as an umbrella term for architectures that do not fit RISC principles. This includes both **mainframes** and **microcontrollers**.
  - Early examples of CISC architectures include **System/360**, **PDP-11**, **VAX**, and **Data General Nova**.
- **Modern Context:**
  - Many modern microprocessors and microcontrollers, such as the **Motorola 68000-family**, **Intel 8080**, **x86-family**, and **Zilog Z80**, are categorized as CISC due to their complex instruction sets.
  - Some architectures, such as **Microchip Technology PIC**, have been ambiguously classified as both RISC and CISC due to certain shared features.

<b>Complex instruction set format (CISC)</b>	<b>Reduced instruction set format (RISC)</b>
Large number of instructions (around 1000) Application point of view they are useful but for system designer it is a headache	Relatively Few numbers of instruction, only those instructions which are strictly required, and in case of implementation of a complex function a set of simple instruction will perform together
Some instruction is there which perform specific task and are used infrequently for e.g. instructions for testing	Few instructions will be there which will be performing more frequent work
Large number of addressing mode, leading to lengthen instruction, but compilation and translation will be faster	Number of instruction mode will be less, hardly 3 to 4
Variable length instruction format	Fixed length easy to decode instruction format
Instruction that manipulate operand in memory	Do not perform operation directly in memory, first load them in register and then perform, so all operations will be done with in the registers of CPU
Powerful but costly	Relatively less powerful but cheap
Microprogrammed control unit, relatively slow	Hardwired control unit, so fast