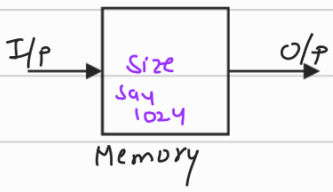
Morrish Manino

William Stallen

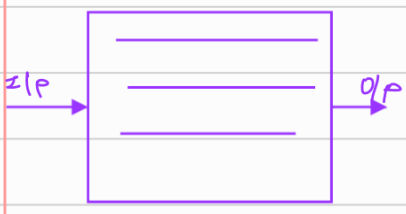
Memory -

Primary (Main functioning) — Secondary (To backup)

- The input line will take address of the input and we get output from o/p line in which we can read/write.



- 1024 bytes is avalible memory
- size of word is single location only
- If size of word if take 2 block then give it.



logical rep of memory

N bits in 2D (in 1D)

= No of location X Size of location

= $\quad \times m$ (let)

No of location $= \frac{N}{m} = 2^n \times m$

No of location

(n bits are req to store location)

I/P = n bits O/P = m bits

Ex Capicity = 1KB Size of word = 1 byte n = ? m?

N of location $= 2^{10} \times 8$ bits

$n = 10$ $m = 8$

Ex Capicity 256 bytes size of w. = 16 bits

$N = 256 \times 8$ bits $= 128 \times 16 = 2^7 \times 16$

$n = 7, \quad m = 16$

size of i/p = 7 bits size of O/p = 16 bits

- Program execution is seen from program counter present in CPU where program is stored in binary.
- This prog. counter is storing address of program/memory location.

15.00 - ① Fetch the information registor (These are the micro instruction)

- ② Decode find what operation is to be done.
- ③ ADD A B

  A → Apply address & get data.

  the binary O/p is data A

  similarly fetch B

- ④ Perform operation (by ALU)
- ⑤ add. A is applied at input and put address of o/p (final) at o/p by importing address of registor storing it.

- If the instruction are sequential then after execution of prog. the value of prog. counter increase by 1.

- A prog contain goto 100 then we have to transfer control to 100th line of the prog.

Suppose we are on line a, ∴ we need to identify relative distance b/w a to 100 now this particular value should be added in programm counter. ∴ prog. counter move (100-a) line

- Even at branch after execution of prog. ↑ PC by 1 ∴ we need to subtract it

∴ Instruction ⇒ ADD PC +(100−a−1)

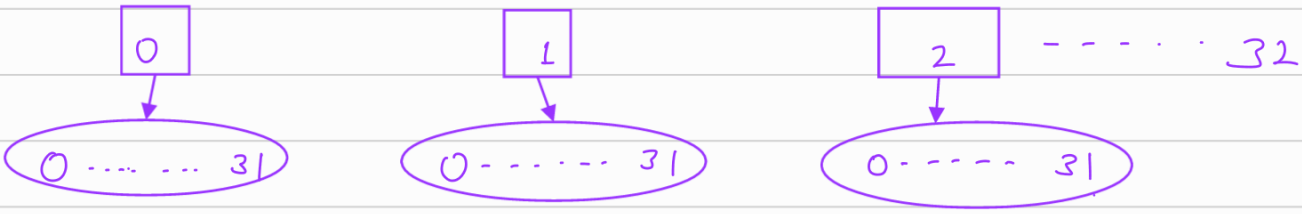
Ex In memory we have 2048 location divide in 2D

11 bits $\Rightarrow 2^{11} = 2048 = 2^{11} \times 1$

- A single word take T_1 time then for transferring 10 words it takes $10T_1$ time but if we transfer them as group of word it will take less that $10T_1$ time, now this group is 2D

- Say a group of 32 words is passed. (each 32 → forms a block)

$1024 = 32 \times 32$



1024 = 10 bits

5 bits MSB → Block NO

5 bits LSB → Word No.

- To reserve half of the memory we will use a bit of memory (from last bit of MSB)

Upper 0 → 15 (MSB 0 for upper)

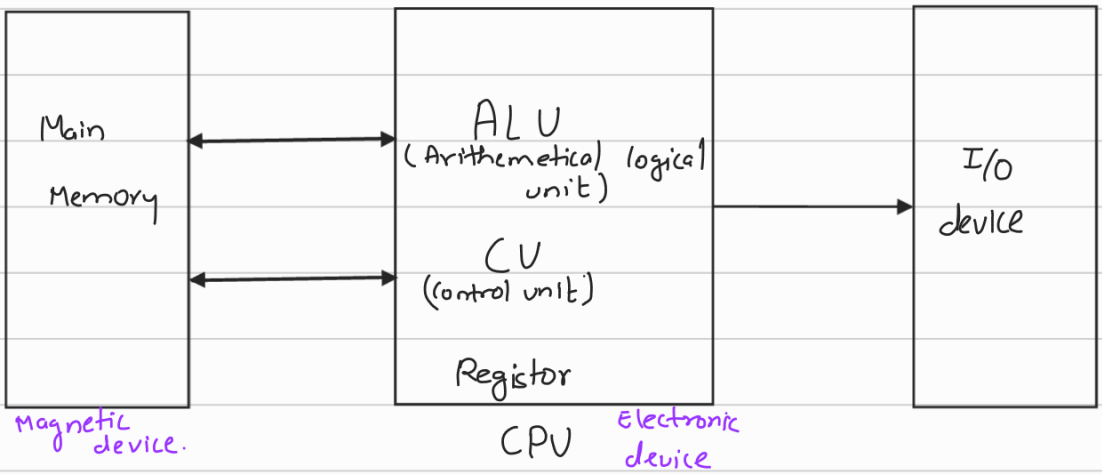
lower 16 → 32 (MSB 1 for lower)

Computer system

I/O/p process memory

Von Newman Architecture-

- The basic organization of computer was defined by john Von Newmen it consists of a CPU responsible for fetching and executing instruction. main memory directly accessable by CPU and I/P & O/P device.



- ALU perform +, *, −, % using sequential ckt & combⁿ ckt.
- Register is comb of f/f which is a 1 bit storage.
- Type of registor :- IR, PC, AR, DR, AC (Use to temp. store data)
  - IR → Instruction registor
  - PC → Prog. counter (store add. of next instruction)
  - AR → Address registor (Always 12 bit)
  - DR → Data registor
  - AC → Accumulator registor

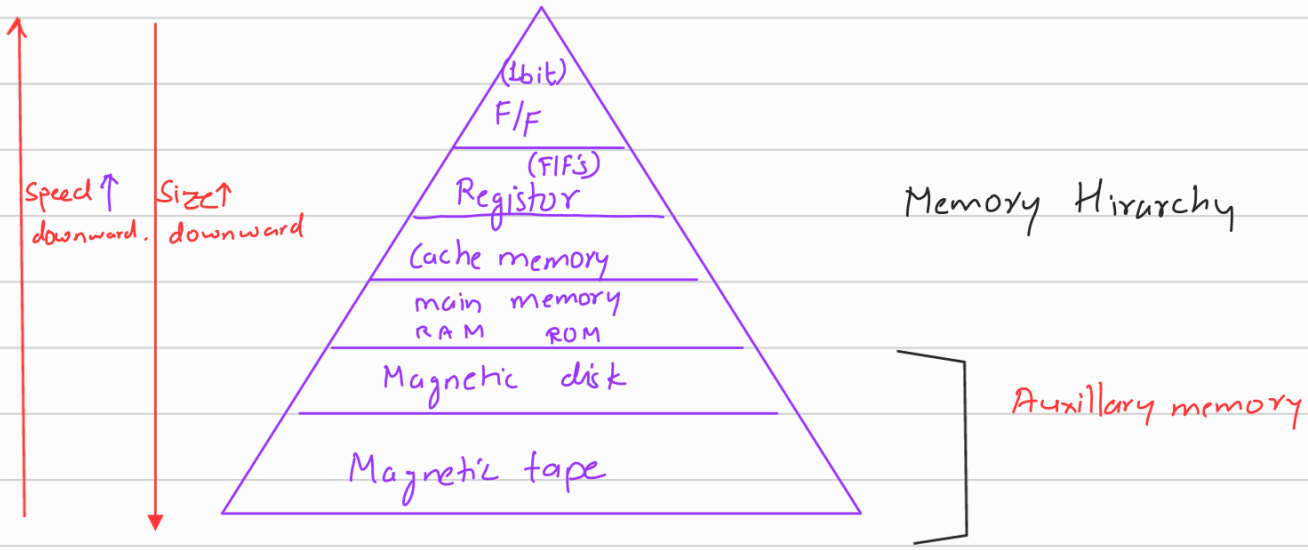
- To not waste efficiency of ALU & CU due to slow speed memory, we store temprory data (on which we have to work) in registor.
- In a 16 bit input MSB represent indirect or direct bit, next 3 bit represent operation code, and last 12 line holds address

If OP = 000
- MSB's '0' represent it is present in address in last 12 line but if it is 1 we get pointer to the data by address in last 12 line.
- ∴ This model is also called Stored data model.

If OP = 111
- Then it will be either registor or I/O instruction
- If last bit of OP is 0 it's register instruction
- If last bit of OP is 0 it's I/O instruction

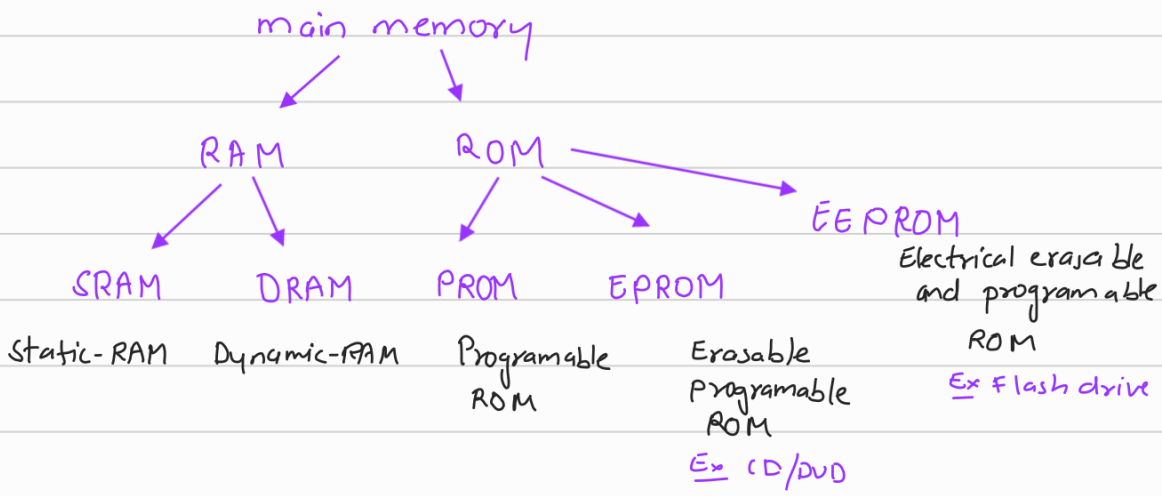
- 'n' bit microprocessor mean it process 'n' bit in a clock cycle.

Memory Orgnization-



Main memory-

- CPU directly communicate to main memory for accessing storage of data
- When the CPU require data, first it check cache then in ram then data come from hard disk.
- hard disk → ram ⇒ swapin
- ram → cache ⇒ mapping
- Their are two type of main memory



- Data transfer with help of system bus

RAM-

- It stores the operating system and current running program which are executed by CPU.

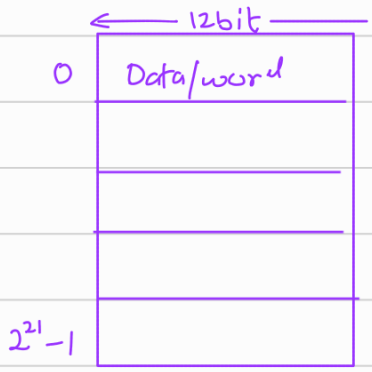
- CPU can directly access data from any random location of RAM
- Program is part of process
- OS must occupy less than 1/4 of ram rest is for process and user data

structure of RAM-

- Suppose a computer system has main memory capicity 2M x 12 bit
  ($M = 2^{20}$, $K = 2^{10}$, $G = 2^{30}$, $T = 2^{40}$, $P = 2^{50}$) $= 2^{21} \times 12$ bit

∴ data size = 12 bit i.e it store data of size 12 bit, this will store 2^{21} size data

RAM -



- Difference b/w SRAM & DRAM

SRAM	DRAM
- Made up by transistor or F/F	- It is made up by capacitor
- It is costly then DRAM	- Cheaper as compared to SRAM
- SRAM has faster access time compare to DRAM	- has more time to access compared to SRAM
- It require more power consumption then DRAM	-
- It's storage capacity is less then DRAM	-
- It is used for cache memory	- It is used as a main memory

ROM-

- It store a program named bootstrap loader it store the address of secondary storage where operating system is stored
- When computer boots up nothing to be executed by CPU
- CPU first execute bootstrap loader program and loads Operating system into the main memory

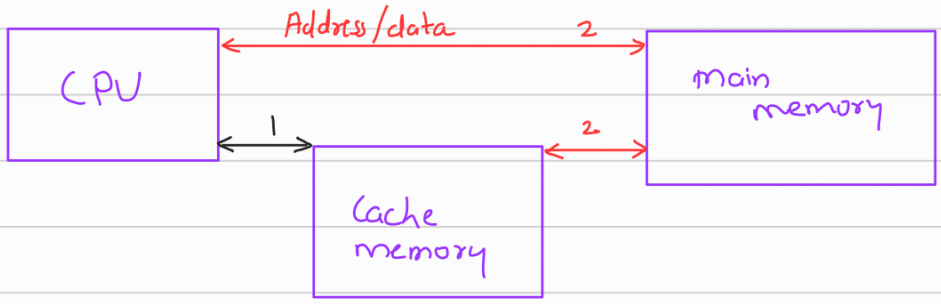
- All bios related setting are stored in rom which are necessary for computer system

RAM	ROM
- In ram both read and write operation can be performed	- In rom only read operation are performed
- The capacity of RAM is larger then rom	-
- It is faster then ROM	
- It is volatile memory ~~Data~~ Power	

Diff. b/w^ PROM, EPROM & EEPROM

Block structure-

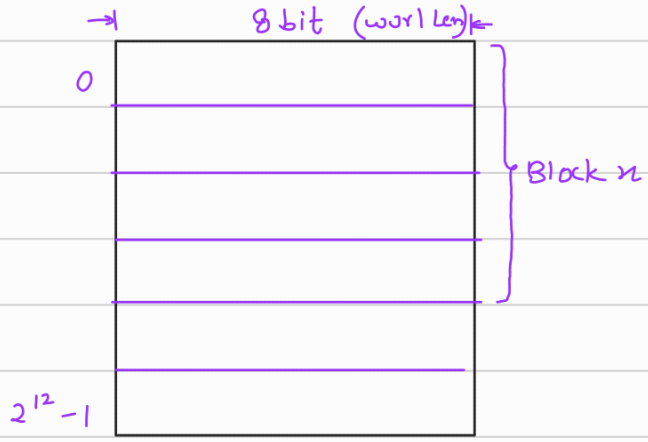
- Data is transffered to CPU by each location of RAM depending upon req.. block transfer is requiord when data is transffered b/w^ RAM & cache



Ex Memory size 4KB Block size 16 B No of block?

$\frac{4KB}{16B} = 2^8 = 256$ block. size of word len = 8bit

∴ Structure



4KB words

$2^8 \times$ (8 bits) words

2^{12} words

- <u>Designing of ram and rom memory</u>-
- Suppose a ram and rom chip is given with its size and we have to design a raw memory using it and no of chips is calculated by size of raw memory divided by size of ram or rom chip.

Q. A computer system has to design ram of size 2048 Bytes using 256X4 bit, ram chips find the number of ram chips and design a ram?

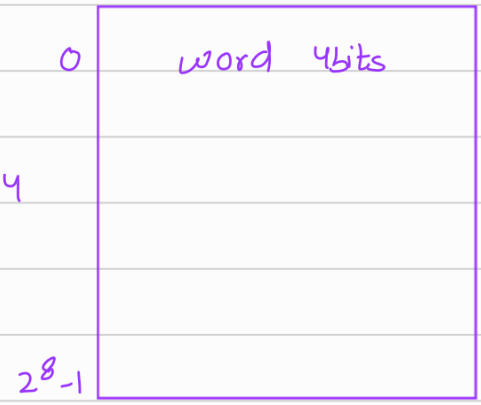
size of ram chip = 256 X 4 bit

$= 2^8 \times 4$ bit

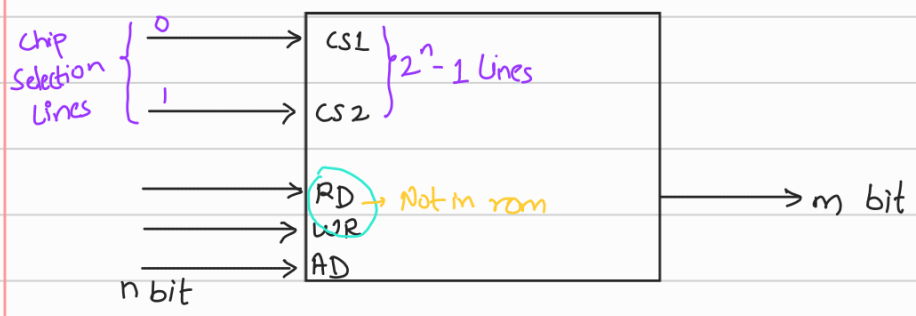
8 = address word size = 4

No of ram chips required = $\frac{\text{size of raw mem}}{\text{size of ram chip}}$

$= \frac{2048\,B}{256 \times 4b} = \frac{2048 \times 8\,b}{256 \times 4\,b} = 16$



Structure



size = $2^n \times m$ bits

Nothing here take bits

Q How many 128X8 ram chips are needed to provide memory capicity of 2048 B show the design structure.

No of chips = $\frac{2048 \times 8 \text{ bits}}{128 \times 8} = 16$

a x b bits

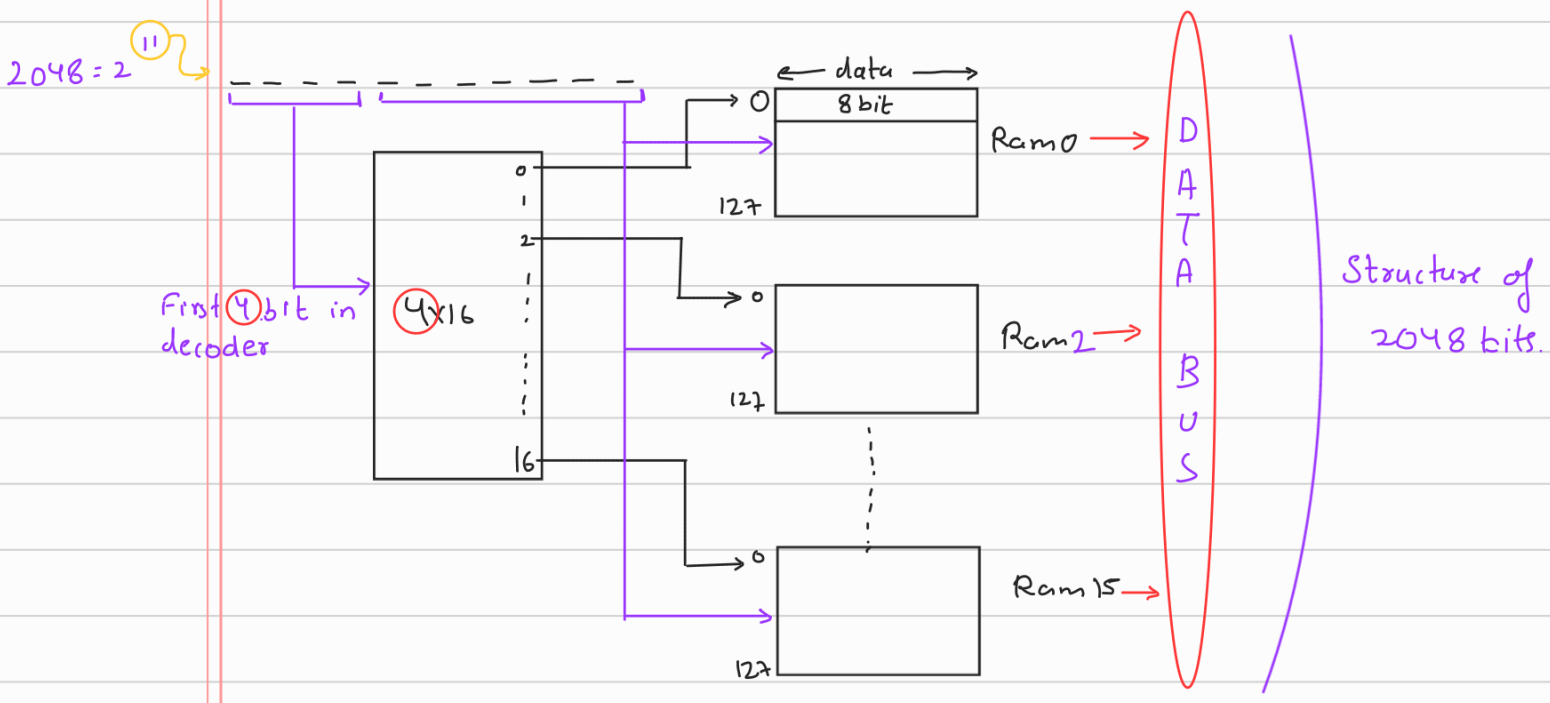
a bytes

Size of address bit = 11 bits (of ram req) $(2^{11} \times 8 \text{ bits})$

Size of given ram = 7 bits $(128 \times 8b)$

Size of decoder = $n \times 2^n$ or $\log_2 N \times N$ $(N = 2^n)$

i.e here 4×16 (How many time given chip fit in req chip)



Note. Output of RAM moves to data bus

Q. How many 128 X 1 ram chips are needed to provide memory capacity of 128 bytes show the design structure?

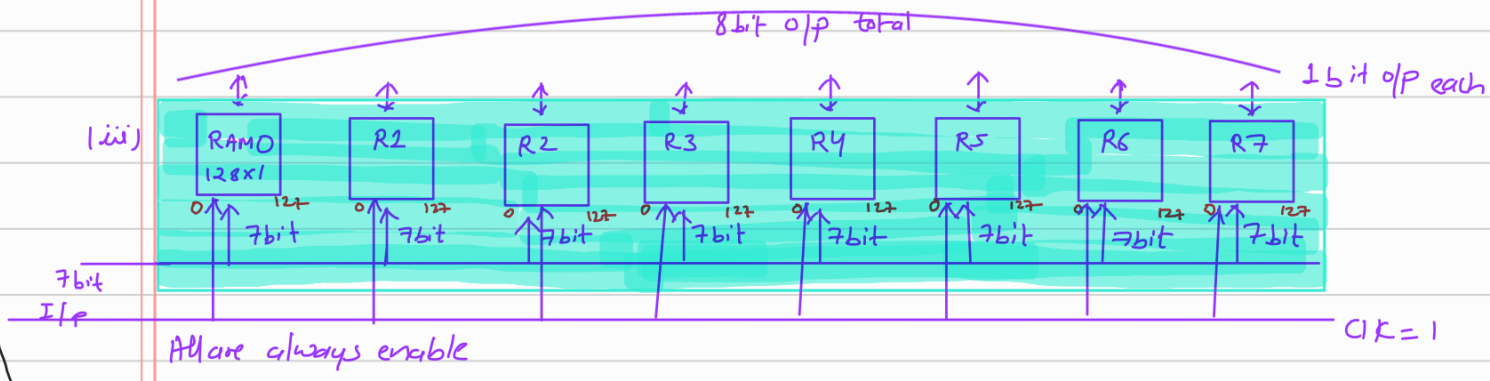
Q. Req capacity 16K X 8 , Given size 128 X 1 design structure?

(i) No of ram chip = $\frac{128 \times 8}{128 \times 1} = 8$

(ii) Address bit for req ram ⇒ $(128 \times 8 = 2^7 \times 8)$ i.e 7

(iii) Size of decoder = $\frac{128}{128} = 1$ ie no decoder is req. as all fit

- If only no of words change then the chip arranged vertically
- If only size of words change then the chip arranged horizontally.

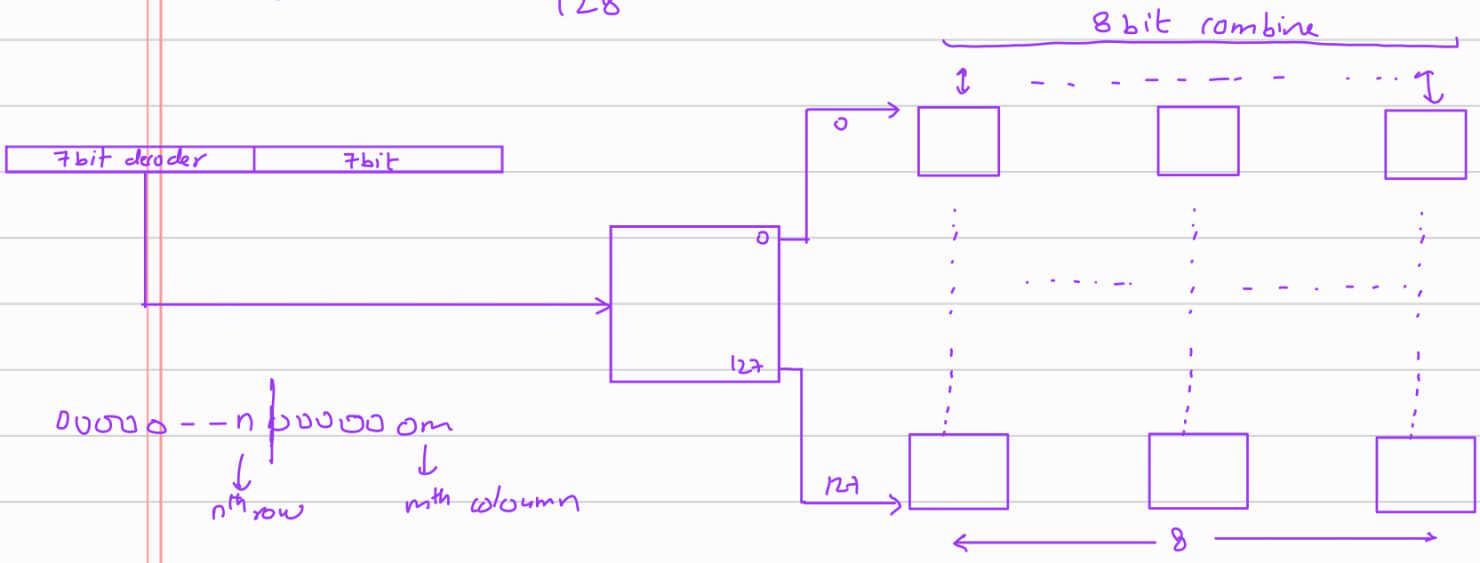


16 K x 8 , 128 x 1

(i) No of ram chip $= \frac{16 \times 1024 \times 8}{128} = 2^{10} = 1024$ Chips

(ii) Address bit $= 16K \times 8 = 2^{14} \times 8$ i.e 14 Address bit

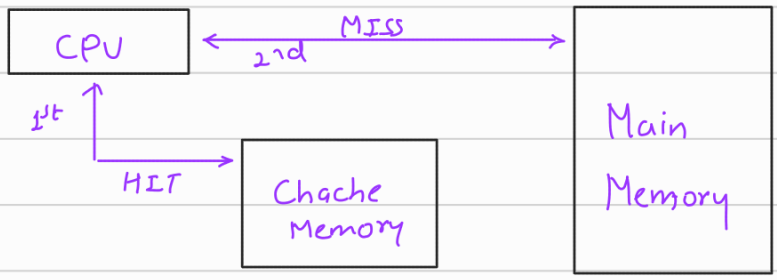
(iii) Size of decoder $= \frac{16 \times 1024}{128} = 2^7 = 128$ i.e 7×2^7



Q. A comp. system has ram chips of 256x8bits & rom chips of 1024x8 bit the comp. needed 2KB ram and 4KB rom design it

Cache Memory -

- It is the fastest search memory it stores fraction of data of main memory
- The data which are frequently used by CPU are stored in the cache memory.



- If not found in chache then opt data from main memory
- CPU searching data in cache memory that process is called HIT process if it doesn't find data in cache this process is miss and data will be search on main memory

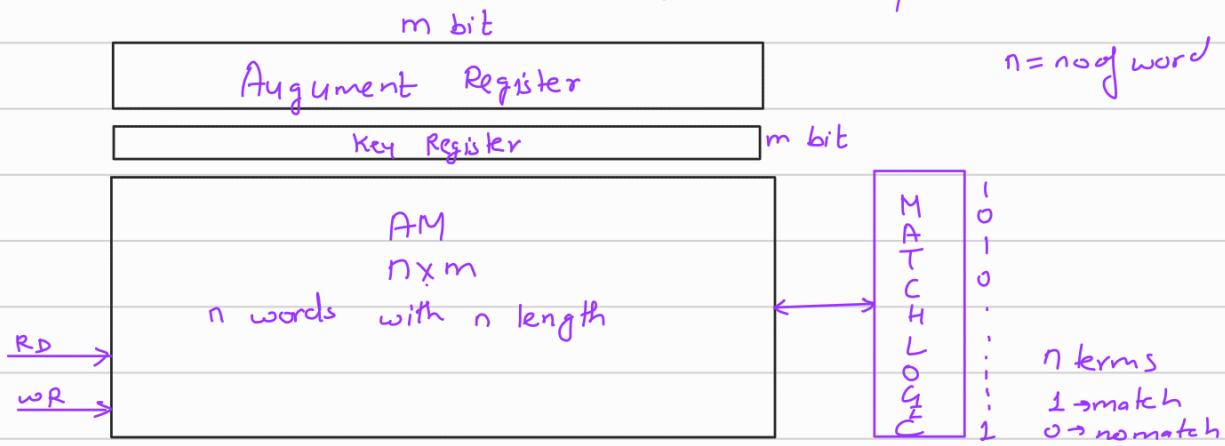
$$\text{Hit ratio} = \frac{\text{Hit}}{\text{Hit} + \text{Miss}}$$

How can cache be design ??

- Associative Cache memory /Full associative
- Direct mapped cache memory
- Set associative cache memory

Associative memory -

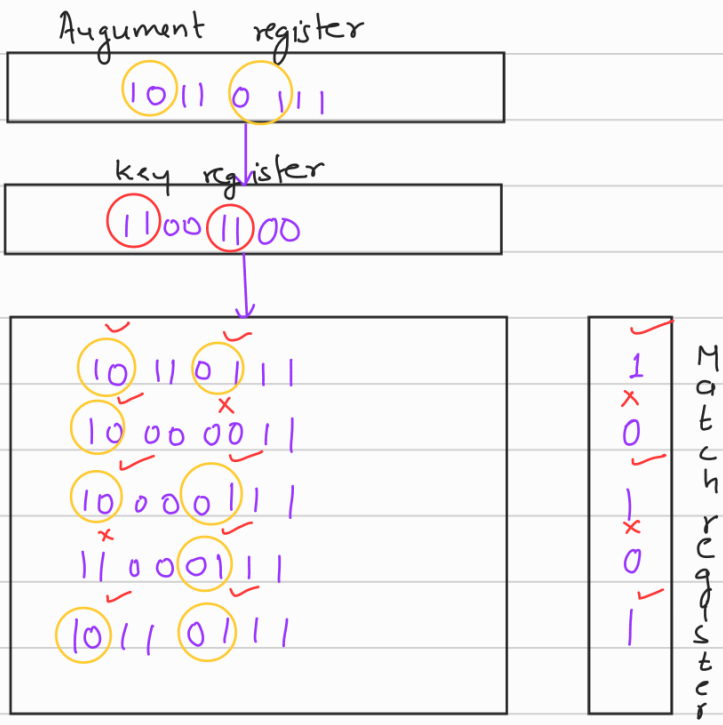
- No physical existance
- It is used for parallel search of data from memory



- for 1011, if 1000 is used as key register it give 1 at 1001, 1011 i.e it ignore after 1 ---.

no of 1 = no of bit matched

I/P	1011		1011
Key	1000		–
Search	1001	1	0
	1011	1	0
	1100	1	0
	1011	1	1

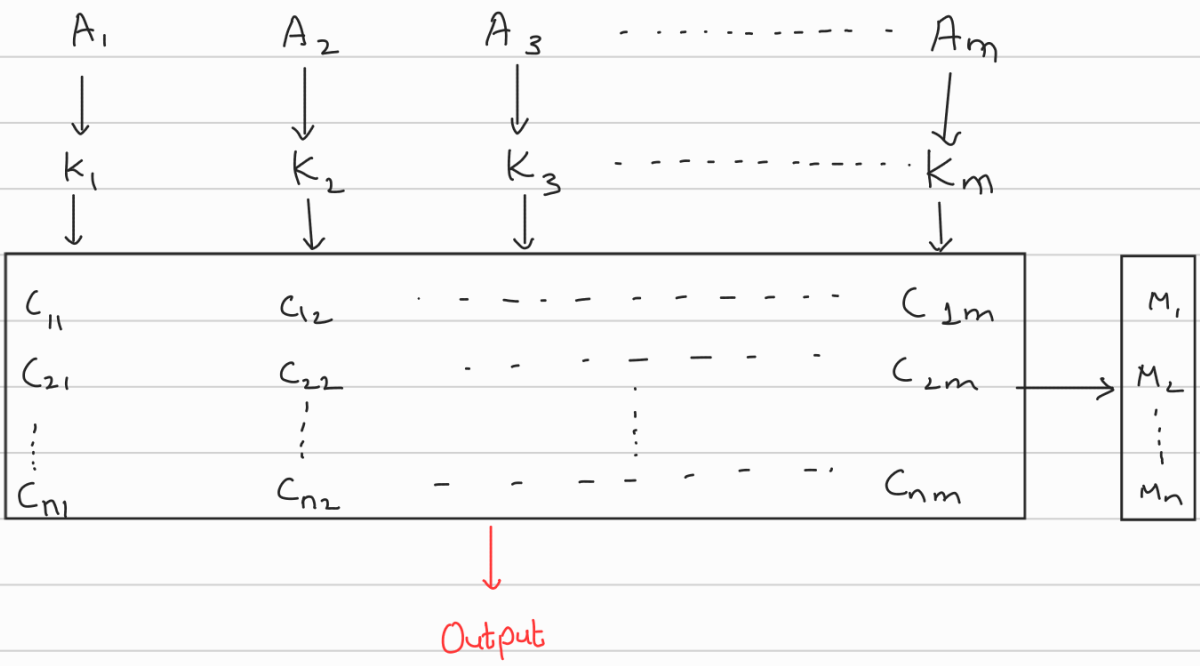


without key register

$$\begin{cases} x_j = A_i \odot C_{ij} \\ M_i = x_1 . x_2 . x_3 \ldots x_m \end{cases}$$

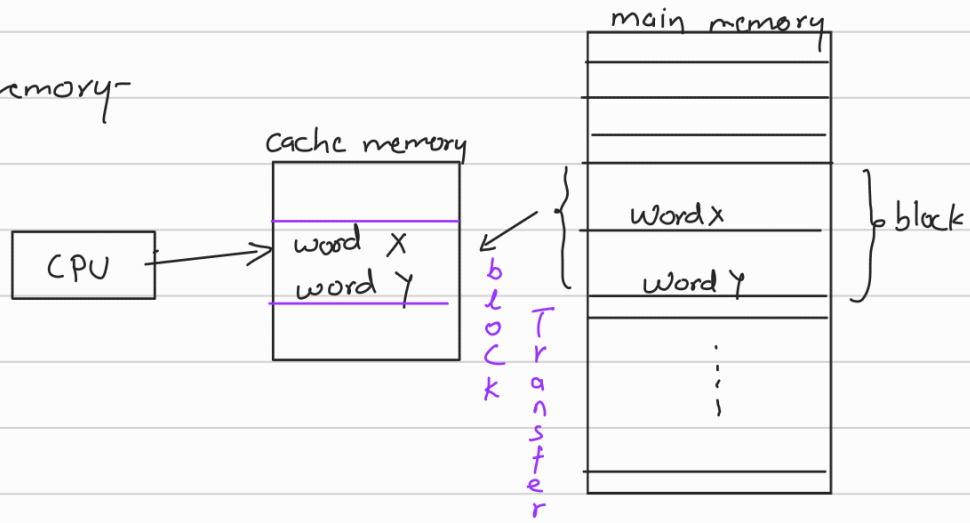
with key register

$$x_j + K_j = \begin{cases} 1 & K_j = 0 \\ x_j & K_j = 1 \end{cases}$$



- When miss is their and data is taken from main memory instead of cache then it's transferred in form of block
- The cache memory works on the principle of locality of reference their are two type of principle of locality of reference

Associate memory-



- Two type of principle locality-

① Temporal locality - When same word of cache memory will be reffered again and again in future is called temporal locality

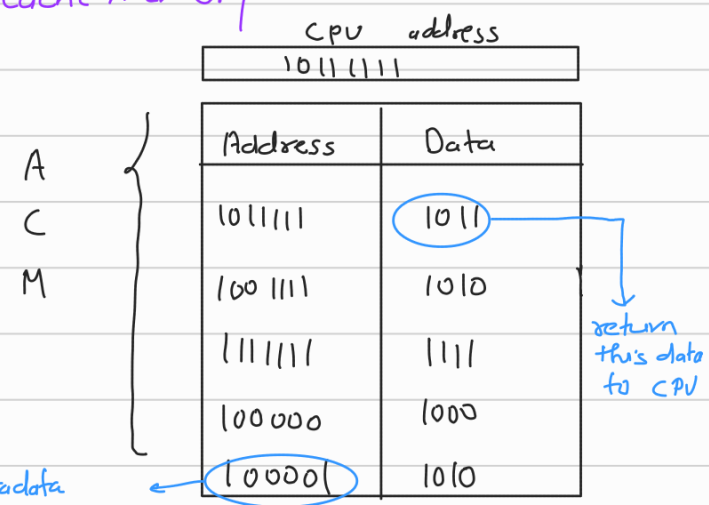
② Spatial locality - When the word near to last reffered word will be reffered in future is called spatial locality

- In above ex if 'x' is finded multiple time then location of 'Y' become spatial

Associative mapping-

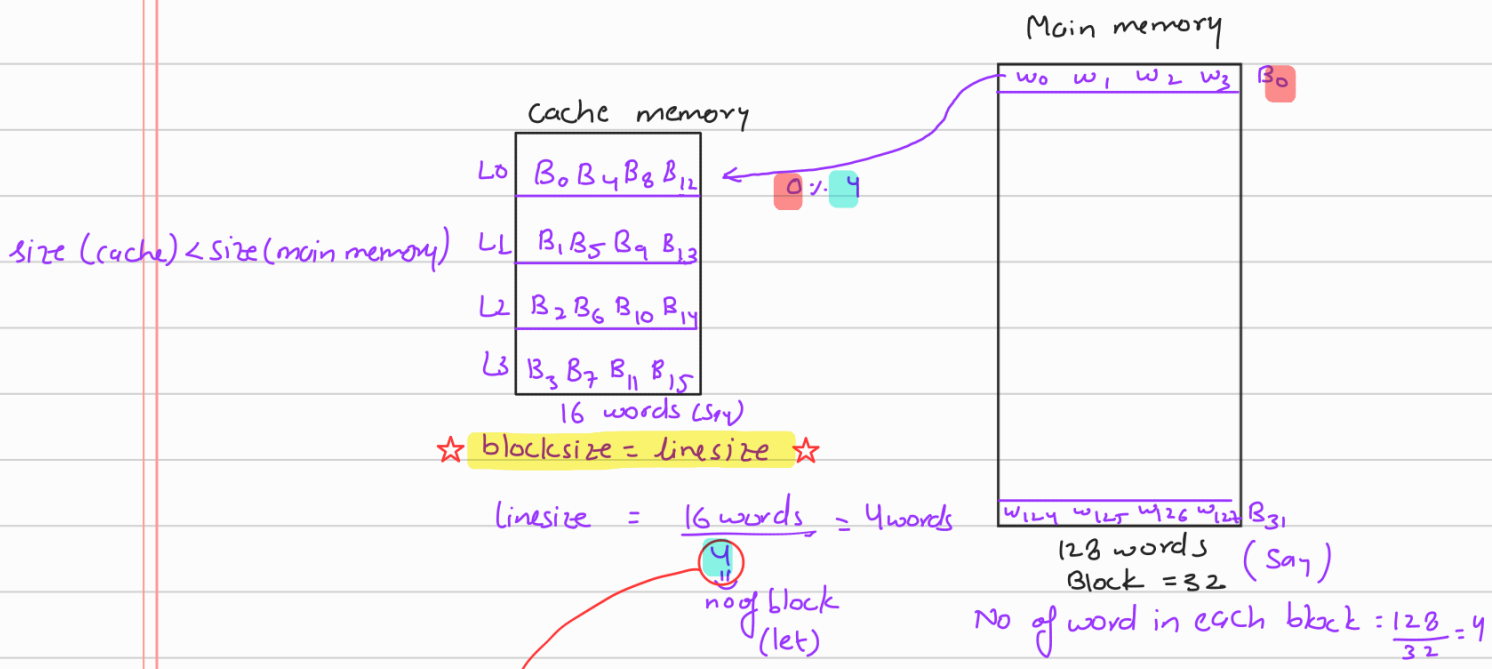
- It works on the principle of associative memory where CPU addresses and memory data is stored, CPU generate the address of memory based on the request, it searches the address in associative cache memory if matched then gets data from cache memory

Cache → Address + Data



- Fast parallel searching of data in cache - Advantage

Drawback - Hardware cost is high because it store data as well as address approximately 50% of cache memory is filled with the addresses



line no. in cache = K mod n where k = block number, n = no of line

where is B_0? $0 \bmod 4 = 0$

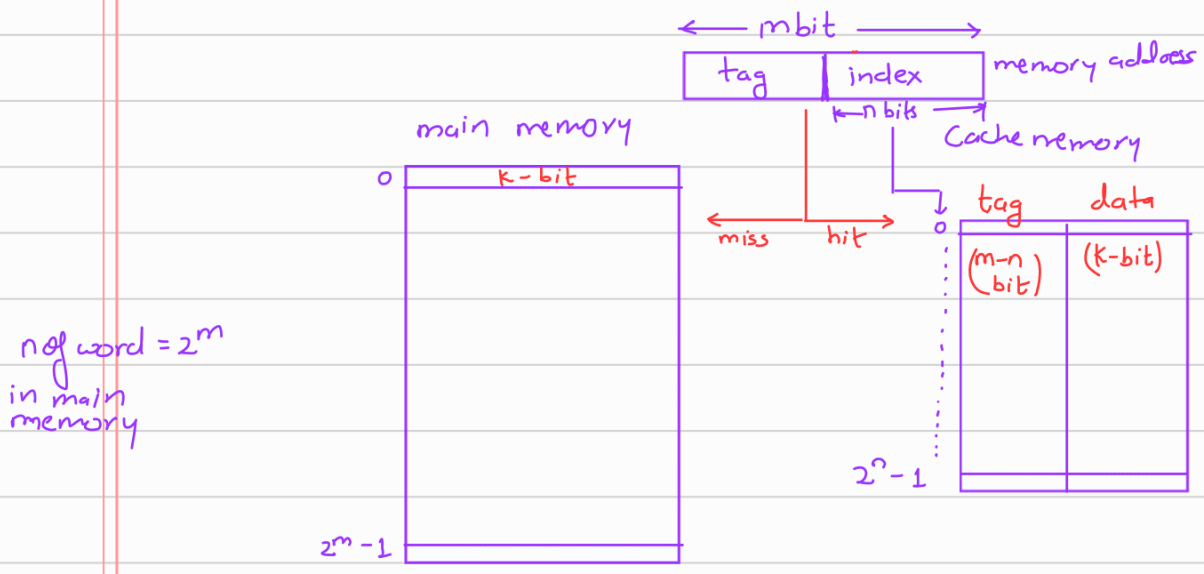
B_{15}?? $15 \bmod 4 = 1$ (on L1 in cache)

B_{16}?? $16 \bmod 4 = 0$ but cache is full Ab kya kare?

- when cache become full then it will replace other to get it place using its page replacement algorithm

Direct mapping -

- In direct mapping the memory address divided into two parts tag and index



let memory size = $2^{20} \times 8$ bit $m = 20$

cache size = $2^{10} \times 8$ index or line = 10 $\therefore$ tag = 20−10 = 10

- size of memory space = 2^m
- size of cache = 2^n
- size of word in cache = $m - n + k$ i.e tag + data
- size of meta cache bit = $(m-n) \cdot 2^n$ → no of line
- no of meta cache = $m - n$
- size of overall cache = $2^n (m - n + k)$

no of bit in memory ≠ size of word in memory

Q. Suppose a computer system has main memory of size 2M X 10 bits and cache memory of size 1K words if cache is organized by direct mapping find

✓ (i) No of bit to represent tag and index field in main memory

✓ (ii) Size of word of a cache

✓ (iii) size of meta cache or cache tag directory where tag has some additional bits one bit for valid / invalid and two bit for security purpose

(iv) Size of overall cache by including additional bits

size of main memory = $2 \times 2^{20} \times 10$ bits = $2^{21} \times 10$ bits

size of cache -- 2^{10} bits

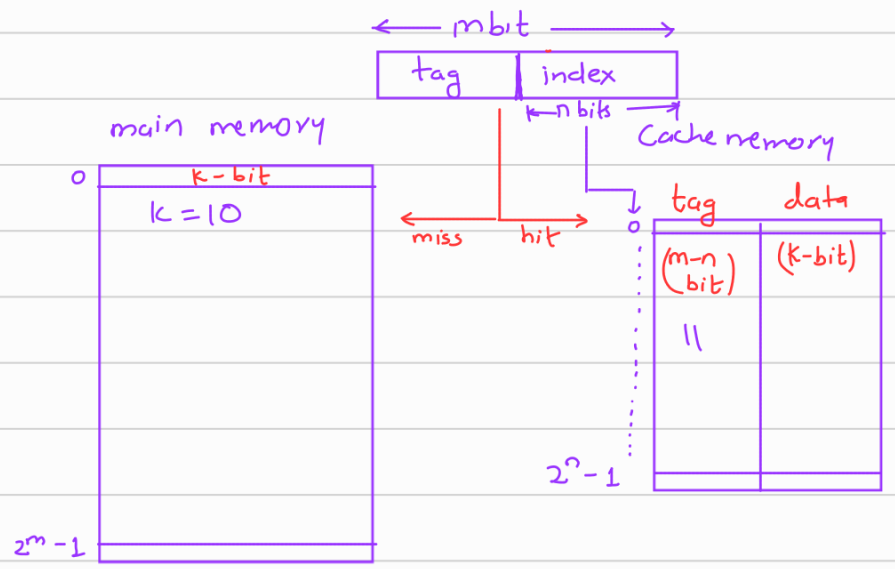
(i) $m = 21$ index = 10 tag = 21−10 = 11 $n = 10$

(iii) size of meta cache = $(21-10+3) \cdot 2^{10} = 14 \cdot 2^{10}$ bits

$k = 10$ (word size of main)

(ii) size of word in cache = only data will be reflected = $11+10=21$

(iv) size of cache = $(21+3) \cdot 2^{10} = 24 \cdot 2^{10}$



$m = 21$

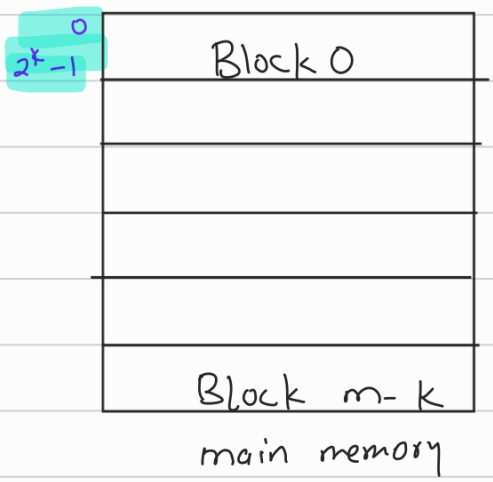
$n = 10$

$m-n+k = 21$

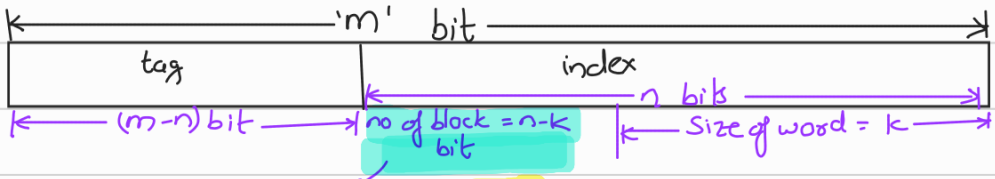
Organization of cache & memory in blocks-

- Suppose the size of main memory is $2^m \times K$ bits and size of cache is 2^n words now divide cache into block each block with 2^k word
- Size of block of cache must be equal to size of block of main memory
-

No of block in cache $= \frac{\text{cachesize}}{\text{blocksize}} = 2^{n-k}$



memory address



No of block $= 2^{n-k}$ (either 1 or 0)

Size of meta cache

$= 2^{n-k} * (m-n)$

no of block * tag size

Size of overall cache

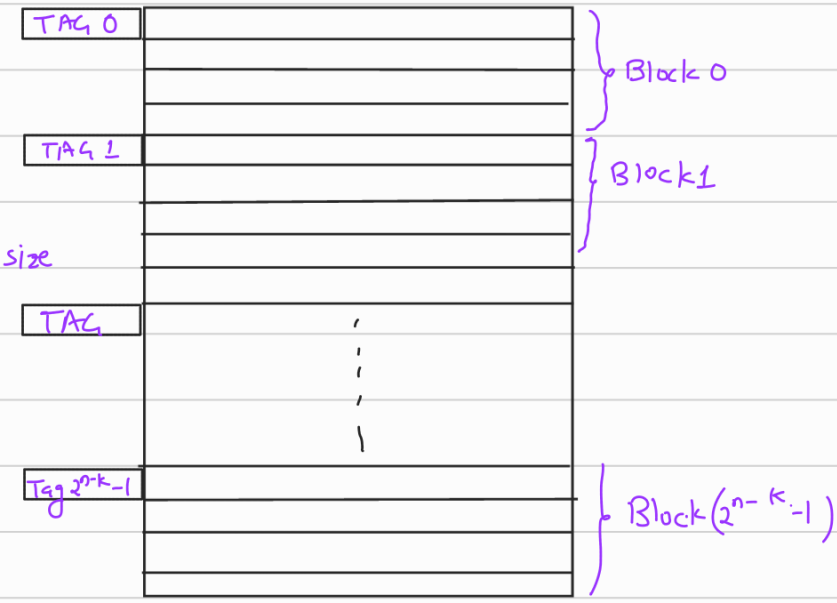
$= 2^{n-k} * (m-n+2^k.b)$ + block size

$= 2^{n-k} * (m-n) + 2^{n-k}.2^k.b$

$= 2^{n-k}(m-n) + 2^n.b$

block size = no of word * word size

$= 2^k * b$



Cache memory

Advantages -

- In block organization of cache instead of storing tag per location of cache we store tag per block of cache it save size of meta cache or cache tag directory
- It will increase principle of locality of reference

Q. let a computer system has 16 MB of main memory and 4 kB of cache memory cache memory is further divided into block each with 16 bytes long if cache is organised by direct mapping find

(i) No. of bit tag, No of bit in block, no of bits in byte/word field within the memory address

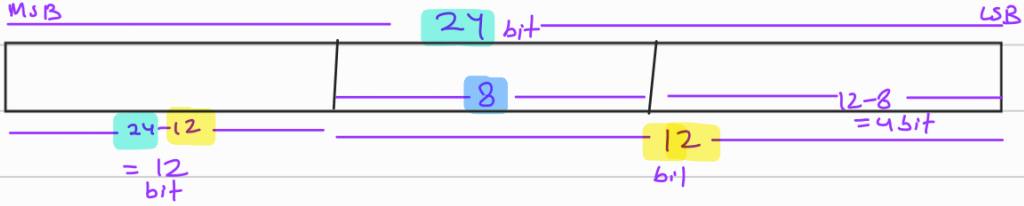
(ii) size of meta cache directory by including 1 valid, 1 invalid and 2 security bit per tag.

(iii) size of overall cache by including one valid, one invalid and 2 security bits per tag.

Memory = 16 MB $= 2^{24}$ bytes

Cache size = 4 kB $= 2^{12}$ bytes

block size = 16 byte $= 2^4$ No of block $= \frac{4\,kB}{16\,B} = 2^8$



(i) 12, 8, 4

(ii) $2^8 * (12 \text{ bit} + (1+1+2)) = 2^8 \times 16 \text{ bit}$

(iii) $2^8 * ((12+4) \text{ bit} + 16 \text{ byte}) = 2^8 (18 \text{ byte})$

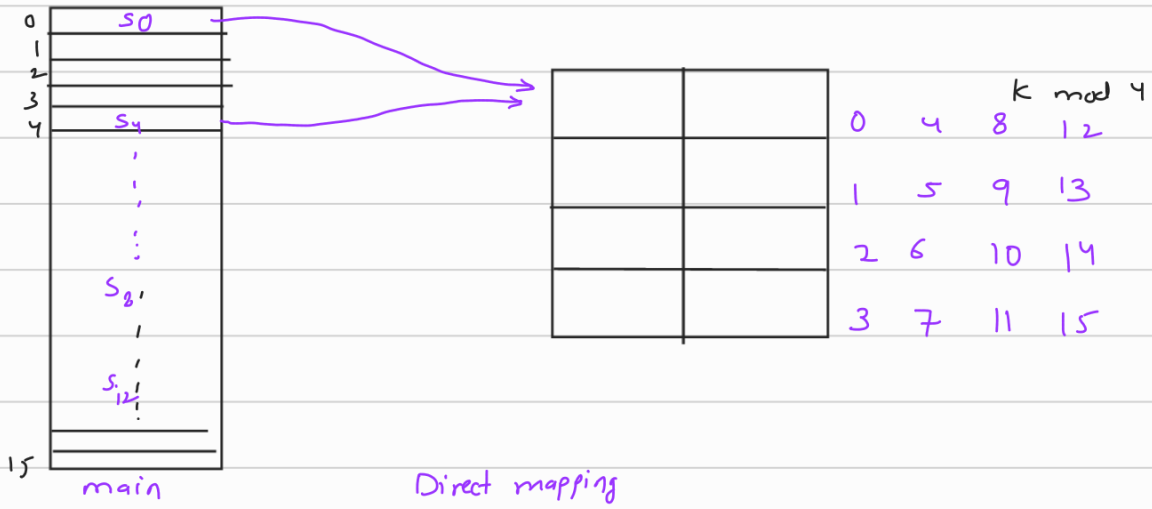
Q. Size of main memory = 4 GB, cache size = 1 MB, Blocksize = 4 KB word size = 1 B you have to calculate no of bits in tag, no of bits in line, and no of bit in block offset / word, and size of tag directory?

Q Main memory size = 16 GB Block size 16 KB
No of tag bit = 10

bit of line/block offset ? overall block size?

Set associative cache memory-

- It is the comb. of associative cache memory with direct mapping suppose cache has 4 blocks and memory has 16 blocks how many memory blocks will be mapped to a single location of a cache

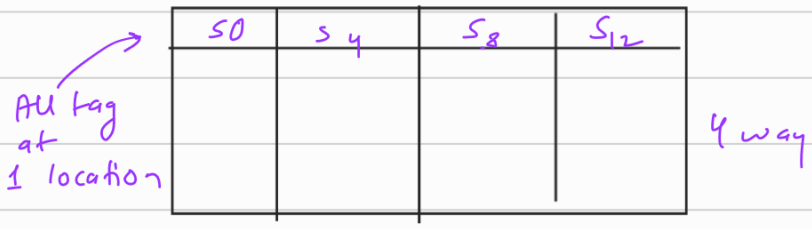


Direct mapping

A/C S1 & S2 ↑ Miss ratio

∴ we use set associative

Set associative -



- Initially cache is empty there will be cache miss for S0 and for S4 again their will be cache miss although cache block 1, 2, 3 is empty in the direct mapping every time S0 and S4 play alternatively will be cache miss (let No of block in cache should be 4 here)

Solution of the problem is organized cache in such a way that multiple block can be presented at some index of cache, it can be done by set associative mapping

let cache size with 16 block

$2^0 = 1$ way set associative.
Direct mapping.

$2^1 = 2$ way set associative

$2^2 = 4$ way set associative

- All the block present at same set index will be excess parallaly by the concept of associative memory suppose main has $2^m \times K$ bit & cache size is 2^n words
Cache is organised as 2^s way set associative

2^n words $\longrightarrow$ 1 way set associative

2^s - way $\longrightarrow$ 2^{n-s}

$s \in [0, 1, 2, \ldots]$
0 = 1 way 1 = 2 way
2 = 4 way

4 way

	w0		w1		w2		w3		$2^s - 1$
0	T	D	T	D	T	D	T	D	
$2^{n-s} - 1$									

T = Tag D = data

★★ Assume 1 block = 1 word

Address (m bit)

'm' bit	
Tag	no (bits to represent index)
$m-n+s$	$n-s$

- Word size of the cache = Tag + data = $m-n+s+k$ (in bit)
- Size of meta cache = $(m-n+s).2^n$ $(2^n = 2^{n-s} * 2^s)$
- overall cache size = $2^n((m-n+s)+k)$

Q. A computer system has cache memory of size 4 K words and main memory of size 1 $\boxed{\text{MB}}$ the cache is organized as 4 way set associative cache, find the following
(i) No of bits to represent set and tag in memory address
(ii) Size of each word of a cache including one valid and one invalid bit
(iii) Size of meta cache including one valid and one invalid bit
(iv) Size of overall cache including one valid and one invalid bit

↳ wordsize

size of cache = 4k words = 2^{12}

Size of main memory = 1MB = $2^{20} \times 8$ words

(i)

20	
Tag	Index / set

$4 = 2^2$

Index = $n-s$ = $12-2$ = 10 ∴ Tag = $20-10$

(ii) Size of word = Tag + data + valid/invalid = 10 + 8 + 2

(iii) Size of meta cache = 2^{12} (tag + valid/invalid) = $2^{12} \cdot 12$

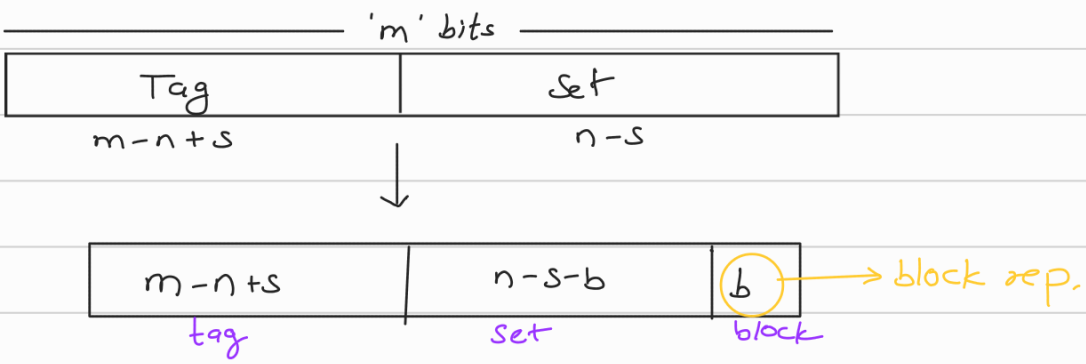
(iv) Size of overall cache = 2^{12} (12 + 8) = $2^{12} \cdot 20$

Blocks in cache & main memory :-

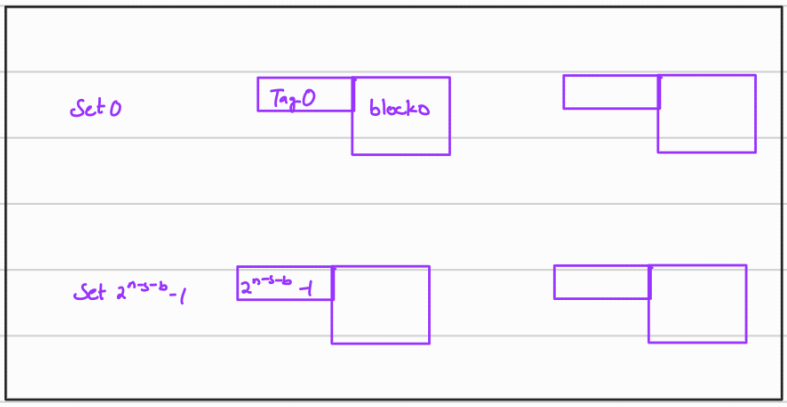
- Let main memory is of $2^m \cdot K$ bit and cache memory of size 2^n words cache is divided into block each of size 2^b words, cache is organised as 2^s way set associative.

2^s way set associative = 2^n words

1-way = 2^{n-s} words



representation →



Size of Meta cache = $2^{n-s-b} \cdot 2^s \cdot (tag) = 2^{n-b} \cdot (tag)$

Size of overall cache = $2^{n-s-b} \cdot 2^s \cdot (tag + \underbrace{blocksize}_{\text{not bit its blocksize}})$

Q A comp. sys. has a cache & main memory of size 128 KB and 512 MB respetively the cache memory is further divided into blocks each of size 16 bytes if cache memory is organized as 4 way set associative cache find

(i) No. of bit in tag, set and block filled in memory address

(ii) Find the size of meta cache by including 1 valid & 1 invalid bit

(iii) Size of overall cache including 1 valid & 1 invalid bit

(iv) Find the additional memory needed to organise cache.

Main memory = 512 MB = $2^9 \times 2^{20} \times 8$ bits
= $2^{29} \times 8$ bits

Cache memory = 128 KB = $2^7 \times 10^{10} \times 8$ bits
= $2^{17} \times 8$ bits

4 way

1) Tag = $29 - 17 + 2 = 14$
Set = 2
Block = 16 bytes = $2^4 \times 8$ bits.

2) Size of meta cache = $2^{17-2-4} \cdot 2^2 (14+2) = 2^{13} \cdot 16$ bit

3) Size of overall cache = $2^{13} (14+1+1 + 2^4 \cdot 8 \text{ bits})$
= $2^{13} (18 \text{ bytes})$
= $2^{13} \cdot 18$ bytes

4) Size of extra $2^{13} \cancel{(1+1)} = 2^{14}$ overall - given
= $(2^{13} \cdot 18 - 2^{17} \times 8)$ bytes
= 2^{14} bytes

<u>Block replacement in cache.</u>

- When a block is not avalible in cache CPU searches it in memory and if avalible in memory then it will be loaded in cache there are 3 algorithm to replace victim block

1. FIFO
2. LRU (least recently used)
3. Optimal
4. LFU (least frequently used)

- For direct mapping there will be same number of page fault (Miss)

Associative cache memory-

Q. The block will be allocated in associative cache memory consider a main memory with 16 blocks (0-15) and cache memory with 4 blocks (0-3) initially cache is empty the following blocks of memory are accessed by CPU in order 1, 3, 4, 3, 5, 12, 12, 1, 4, 5, 10, 5, 10, 7, 5, 10, 1, 5, 7, 10.

FIFO:-

1	3	4	5	12, 12	1, 4, 5	10, 5, 10	7	5, 10, 1, 5, 7, 10
1	1	1	1	12	12	12	12	5
	3	3	3	3	1	1	1	1
		4	4	4	4	10	10	10
			5	5	5	5	7	7

- No of cache miss = 9 hit ratio = 11/20
- No of hit = 11 miss ratio = 9/20

LRU:-

1	3	4	5 (hit 3)	12	12 (Hit)	1	4	10 (hit 5)	5 (Hit)	10 (Hit)	7	5 (Hit)
1	1	1	1	12	12	12	12	10			10	
	3	3	3	3	3	3	4	4			4	
		4	4	4	4	1	1	1			7	
			5	5 (see farthest not use)	5	5	5	5			5	

10 (Hit)	5	5 (Hit)	7 (Hit)	10 (Hit)
	10			
	1			
	7			
	5			

- no of cache hit = 10 hit hit ratio = 1/2
- no of miss = 10 miss miss ratio = 1/2

Optimal -

1, 4, 5 → 5, 10, 7, 1 →

1	3	4	3	5	12	12	1	4	5	10	5	10	7	5	10	1	5	7	10
		H	H			H	H	H	H		H	H		H	H	H	H	H	H
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	3	3	3	3	12	12	12	12	12	12	12	12	7	7	7	7	7	7	7
		4	4	4	4	4	4	4	4	10	10	10	10	10	10	10	10	10	10
				5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5

can change 12, 4 then replace with FIFO

- no of hit = 13 hit rate = 13/20
- no of miss = 7 miss rate = 7/20

Direct mapped cache - (Hash %4)
↳ size of cach

1	3	4	3	5	12	12	1	4	5	10	5	10	7	5	10	1	5	7	10
		4	4	4	12	12	12	4	4	4	4	4	4	4	4	4	4	4	4
1	1	1	1	5	5	5	1	1	5	5	5	5	5	5	5	1	5	5	5
										10	10	10	10	10	10	10	10	10	10
	3	3	3	3	3	3	3	3	3	3	3	3	7	7	7	7	7	7	7

- no of miss = 12 miss rate = 12/20
- no of hit = 8 hit rate = 8/20

Set associative

- let cache is organized as 2-way set associative cache, cache index = memory block address, remainder 1-way set associative cache size, 1 way set ass. cache = 2 blocks, 2-way set associative cache = 4 blocks, 4 way set ass cache = 16 blocks

2 way

FIFO (%2)

1	3	4	3	5	12	12	1	4	5	10	5	10	7	5	10	1	5	7	10
		4	4	4	4	4	4	4	4	10	10	10	10	10	10	10	10	10	10
					12	12	12	12	12	12	12	12	12	12	12	12	12	12	12
1	3	1	3	5	5	5	5	1	1	1	1	1	7	7	5	5	5	7	7

3 comes first from begining

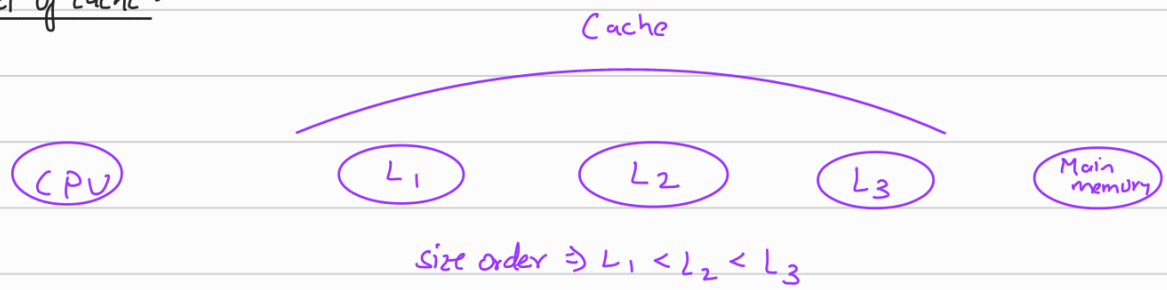
LRU = 10

Optimal 9

Hit = 9 hr = 9/20

Miss = 11 mr = 11/20

Level of cache -



- Most frequent used in L_1
- Cache is more faster than main memory

Writing in cache -

- Two methods
  - Write through (Simultaneously change data in cache & main memory)
  - Writing back (First change in cache the transfer back to main memory)
    - till all the operation b/w^ write in cache & transfer to main memory this bit is locked.

- Whenever cpu searches a block in cache and it is avalible if CPU is using the block for read purpose then no changes will occours in memory & cache, but if CPU is using the block for writing purpose then content of the block will be changed and that change will be done on cache memory and changes in main memory depends on two methods

Write through -

- When there is a write request in cache at the same time memory will also be updated hence memory will have updated data every time

Write back -

- When there is a write request in cache than only cache block will be updated and corresponding memory block / page will be set as dirty so that no other process can refer it, when the block is going to remove from cache at that time it will be updated in memory

- Their are two type of request can be possible read & write, there are also two types of hit rates one for read & other for write

h_r = hit rate for read request
h_w = hit rate for write request
T_{cm} = Time to access cache memory
T_{mm} = Time to access main memory

Write through

(i) Read cycle

$$T_{avg}(R) = h_r \cdot T_{cm} + (1-h_r) \cdot (T_{mm} + T_{cm})$$

(ii) Write cycle

$$T_{avg}(w) = \max(T_{cm}, T_{mm})$$
$$= T_{mm}$$

(iii) By considering both read and write request

$$T_{avg} = (R\%)\, T_{avg(R)} + (W\%)\, T_{avg(w)} \qquad W\% = 1 - \%R$$

(iv) Average hit rate

$$= h_w * (\%w) + h_r * (\%R)$$

Write back-

- let P_d is a page dirty probability

set in main mem.
↑
P_d is time for which mm is blocked

(i) Read cycle

$$T_{avg}(R) = h_r * (T_{cm}) + (1-h_r) * (T_{cm} + (1+P_d)\, T_{mm})$$

(ii) Write cycle

$$T_{avg}(W) = h_w * T_{cm} + (1-h_w) * [T_{cm} + (1+P_d) * T_{mm} + T_{cm}]$$

Miss + Main + write

(iii) By considering both read and write request

$$T_{avg} = (R\%)\, T_{avg(R)} + (W\%)\, T_{avg(w)}$$

(iv) Average hit rate

$= h_w * (\%W) + h_r * (\%R)$

Q. A computer system has a cache memory and a main memory with access time 20 ns and 100 ns respectively it is observed that 60% of the request are read request the hit rate for read & write request are 0.9 and 0.8 respectively, the page dirty probability is 0.6 find the

(i) avg access time for read cycle

(ii) avg access time for write cycle

(iii) avg access time by considering both read & write cycle

(iv) avg hitrate

Considering both writing operation.

$T_{cm} = 20ns$ $T_{mm} = 100ns$ $h_r = 0.9$ $h_w = 0.8$ $Pd = 0.6$ $R\% = 60$

Write through

(i) $T_{avg(R)} = 0.9 * 20 + 0.1 * (120) = 30ns$

(ii) $T_{avg(w)} = max(20, 100) = 100ns$

(iii) $T_{avg} = 0.6 \times 30 + 0.4 \times 100 = 58 ns$

(iv) $h_{Avg} = 0.8 \times 0.4 + 0.9 \times 0.6 = \frac{32}{100} + \frac{54}{100} = \frac{86}{100} = 0.86$

Write back

(i) $T_{avg(R)} = 0.9 * (20) + (1-0.9) * (20 + (1+0.6) * 100)$

$= 18 + 0.1 (20 + 160)$

$= 36 ns$

(ii) $T_{avg(w)} = 0.8 * (20) + (0.2) * (20 + (1.6) * 100 + 20)$

$= 16 + 0.2 * (20 + 160 + 20)$

$= 16 + 0.2 * (200)$

$= 16 + 40$

$= 56$

(iii) $T_{avg} = 0.6 * 36 + 0.4 * 56$
$= 44$

(iv) 0.86

Q. Access time of the main memory & the cache memory in a computer sys. are 500 ns & 50 ns respectively it is estimated that 80% of the main memory request are for read the rest for write, the hit ratio for the read access only is 0.9 & the write through policy is used determine the avg. time of the main memory in nanosecond.

Q. Assume that for a certain processor a read request take 50ns on a cache miss and 5 ns on a cache hit suppose while running a prog. it was observed that 80% of the processor's read request results in a cache hit find avg. read access time in nano second?

$T_{cm} = 5$ ns $\qquad T_{mm} = 50ns - 5ns = 45ns$

$T_{avg} = 0.8(5) + 0.2(50)$
$= 14$ ns

<u>Register transfer language-</u>

- The symbolic notation used to describe the microoperation transfers among register is called register transfer language the term register transfer implies the availibility of hardware logic circuit that perform a micro operation & transfer the result of the operation to the same or another register

<u>Register-</u>

- register is 'n'- bit storage device, it is represent by capital letter

32 or 64
p(max)

16 bit/32bit

ALU

DR →	16 bit	Data register
AC →	16 bit	Accumulator registor (Processor registor)
IR →	16 bit	Instruction registor
☆☆☆PC →	12 bit	Programm counter
AR →	12 bit	Address register (hold address)
MAR →	16 bit	Memory address register
MDR →	16 bit	Memory data register
INPR →	8 bit	
OUTR →	8 bit	

All above are in ALU but PC is most imp. as it hold address to next program or task or instruction

Representation 'k' bit register (k=8) (4bit = Nibble)

PC (let)

7	6	5	4	3	2	1	0
(H)				(L)			

If we want to represent lower or higher bit only PC(L) or PC(H)

Control function- A statement that specify a register transfer implies that circuit are avalible from the output of the source register to the input of the destination register and that the destination registor has a parallel load capability normally we transfer the data under a predetermine control condition this can be represent by if - then condition.

Ex if (P=1) then ($R_1 \longleftarrow R_2$)

P: $R_1 \longleftarrow R_2$ (if P=0 then $R_1 \not\longleftarrow R_2$ else transferred)

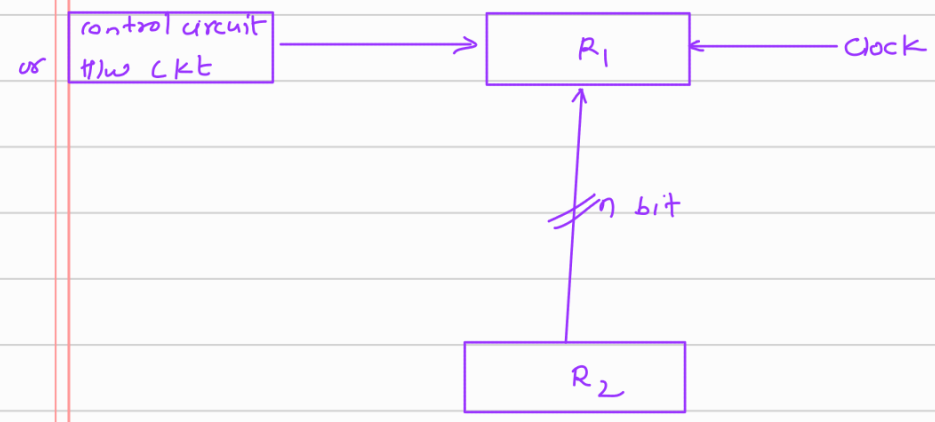
any hardware logic circuit

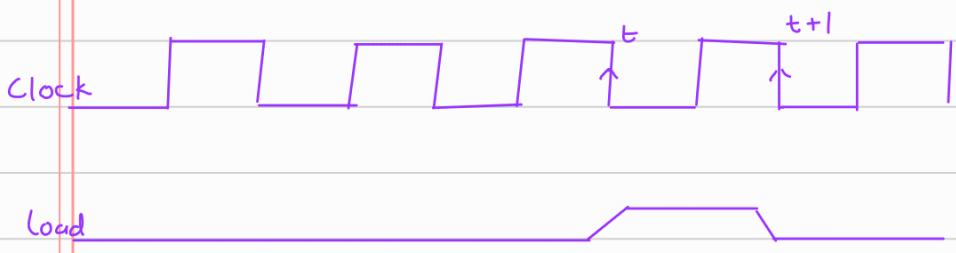
Note → Multiple operation are seprated with comma.

Ex: $R_1 \longleftarrow R_2$, $R_3 \longleftarrow R_2$

$\xleftarrow{R_1} R_2$

Design ckt ??

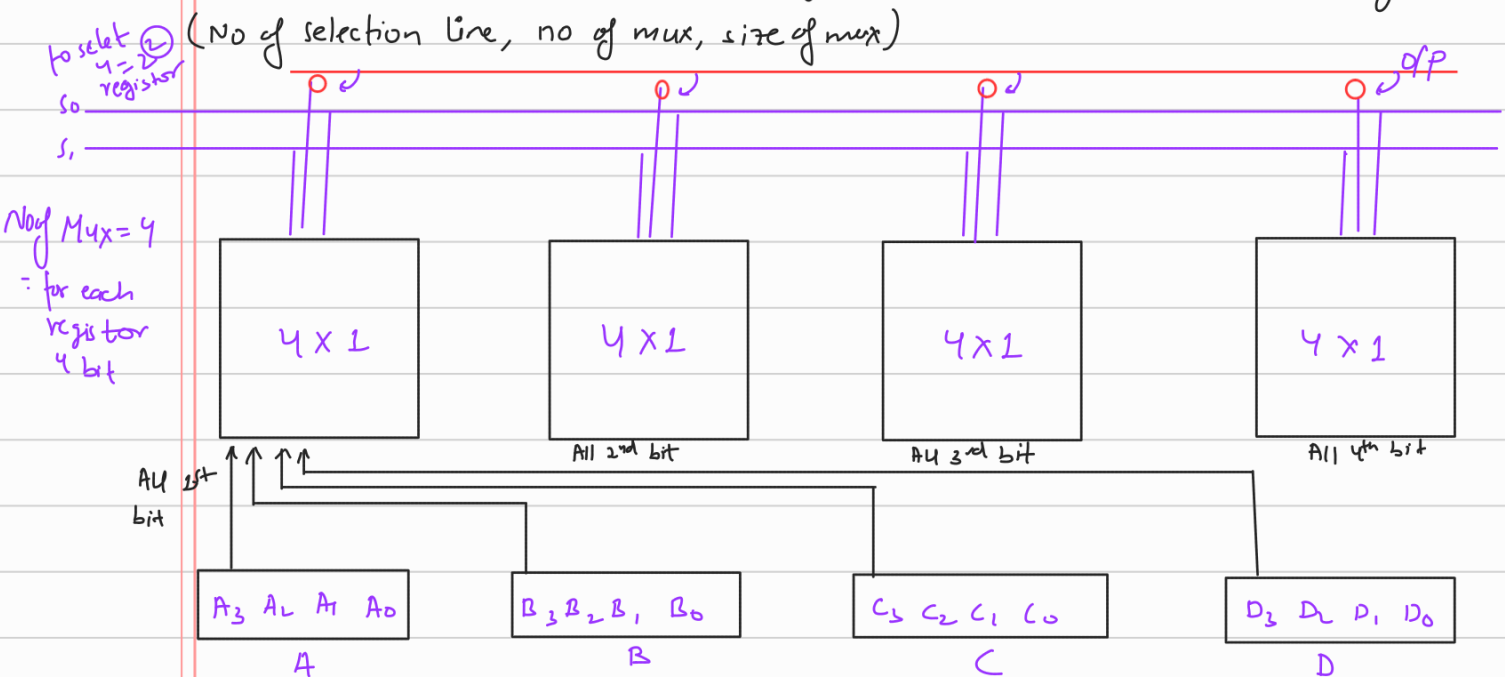




System bus / Common bus -

- A digital computer has many registers and path to transfer information from one register to another, the numbers of wires will be excessive if seprate lines are used b/w^n each registors and all others registors in the system, more efficient scheme for transferring information b/w^n registers in a multiple register configuration is a common bus system, the system bus consists of address lines, data lines and control lines

Q. Draw a system bus with 4 registor and each register having 4 bit (No of selection line, no of mux, size of mux)



No of Selection line = $\log_2$(No of register)

Size of Mux = No of register X 1

No. of Mux = No. of bit in each register.

- Instruction format → Types
  - Memory reference
  - Register reference
  - I/O reference

Type

- opcode
- address / operand

Type

- Zero address
- One address
- Two address
- Three address

- Their are 3 instruction code format using opcode each format has 16 bits the opcode part of the instruction contains 3 bits & the meaning of remaining 13 bits depends on the operation code encounter

① Memory reference -

- In Memory ref. inst. uses 12 bits to specify an address and 1 bit to specify the addressing mode

1 bit	3 bit	12 bit
I	opcode	Address

I = 0 ⇒ direct add. mode — data is stored at address

I = 1 ⇒ Indirect add. mode — pointer to data is present at address

- opcode ≠ 111 then it is memory ref. instruction
- opcode = 111 ⇒ register or I/O reference

Symbol	Hexadecimal code (16 bit) I = 0 (each bit = 4 bit) 0000	I = 1 1000	Description (Accumulator) & (Add. register)
000 (AND)	0XXX	8XXX	AC ← AC ∧ M[AR]
001 (ADD)	1XXX	9XXX	AC ← AC + M[AR]
010 (LDA)	2XXX		AC ← M[AR]
011 (STA)	3XXX		M[AR] ← AC
100 (BUN)	4XXX		PC ← AR (Add. of other file is used)
101 (BSA)	5XXX		M[AR] ← PC (Saving prev. open file) PC ← AR + 1
110 (ISZ)	6XXX		M[AR] ← M[AR] + 1 if M[AR] + 1 = 0 then PC ← PC + 1

Load to accumulator

Store to accumulator

Branch unconditionally

Branch & save return Addresses

Increament skip if zero

- Interept ⇒ by Hardware or by software, cpu may switch (Powercut, stack underflow
- MAC address = 48 bit
- If opcode = 111 & I = 0 ⇒ register reference
- If opcode = 111 & I = 1 ⇒ I/O Instruction

Register reference instruction.

0	111	Register operation

X XXX XXXX XXXX XXXX

Symbol	Hexadecimal code (showing register ref. inst.)	Description
CLA	7800	Clear AC (In remaining 12 first bit is 1) AC ← 0
CLE	7400	Clear E E ← 0
CMA	7200	Complement AC AC ← $\overline{AC}$
CME	7100	Complement E E ← $\overline{E}$
CIR	7080	Circular right E ← AC(0) AC ← shr AC AC(15) ← E (LSB)

CIL	7040	Circular left	$E \leftarrow AC(15)$ $AC \leftarrow Shl\ AC$ $AC(0) \leftarrow E$
INC	7020		$AC \leftarrow AC+1$
SPA	7010		Skip nxt inst. if AC ⊕ve If $AC[15]==0$ $PC \leftarrow PC+1$
SNA	7008		Skip nxt inst. if AC is ⊖ve If $AC[15]==1$ $PC \leftarrow PC+1$
SZA	7004	complete AC is '0'	Skip nxt inst. if AC is 0 if $(AC==0)$ then $PC \leftarrow PC+1$
SZE	7002		Skip nxt inst. if E is 0 if $(E==0)$ the $PC \leftarrow PC+1$
HLT	7001		Halt/Stop

Input/Output refrence instruction

15	14 12	11 0
1	1 1 1	I/O - operation

Symbol	Hexadecimal code	Description
INP (8 bit)	F800	I/P character to AC $AC[0-7] \leftarrow INPR$
OUT (8 bit)	F400	O/P character from AC $OUTR \leftarrow AC[0-7]$
SKI	F200	Skip on i/p flag if $(FGI==1)$ then $PC = PC+1$
SKO	F100	Skip an o/p flag if $(FGO==1)$ $PC = PC+1$
ION	F080	Interept enable on $IEN \leftarrow 1$
IOF	F040	Interept enable off $IEN \leftarrow 0$

Flag show status of ALU, Sign(s), zero(z), carray(c), etc

8259 ⇒ Interept chip
8257 ⇒ DMA chip
8253 ⇒ Clock chip
8251 ⇒ USART

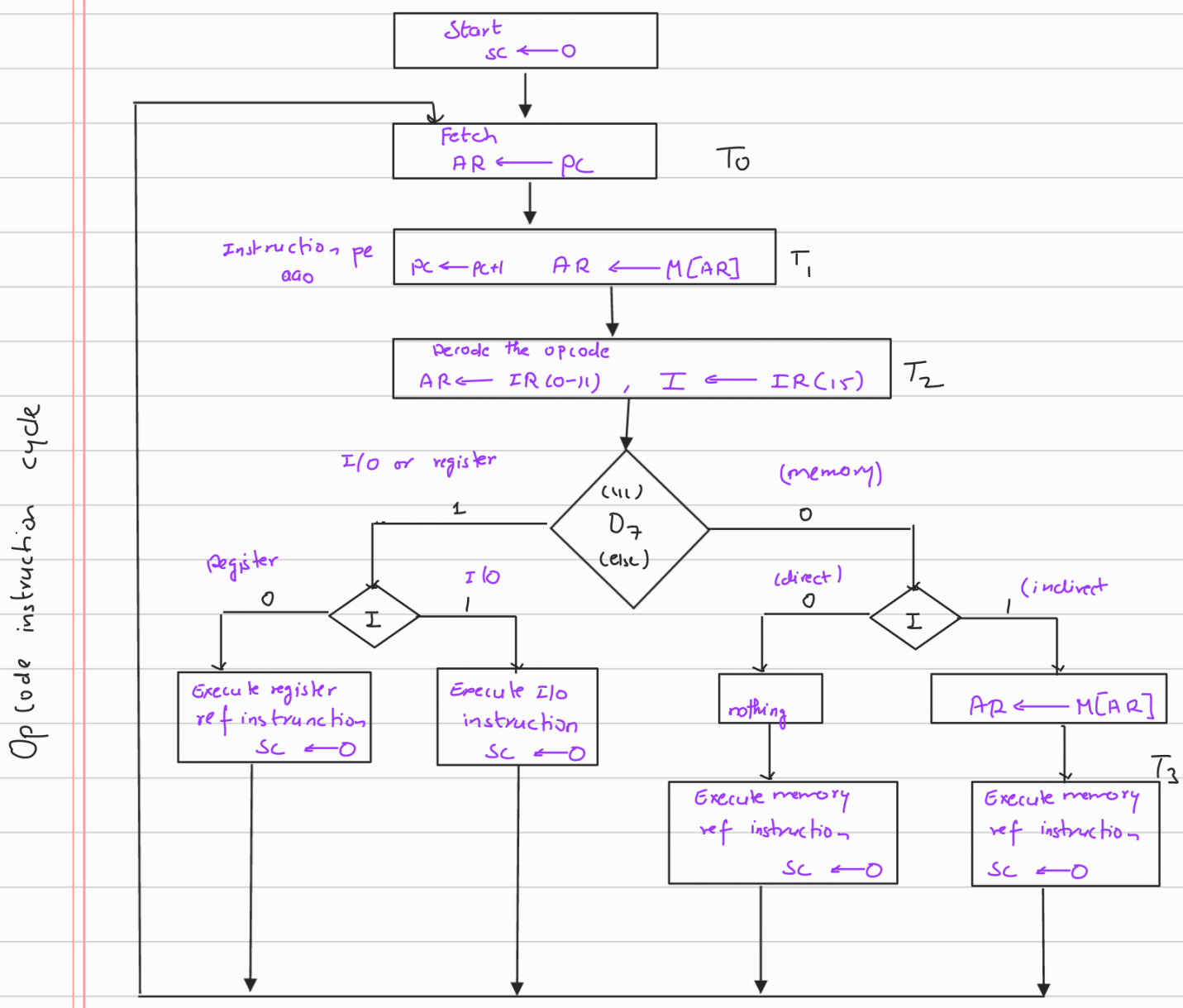
} chips helps microprocessor

Instruction cycle-

- A prog. stored in the memory unit of computer consist of a sequence of instruction each instruction cycle is subdivided into a sequence of subcycles or phases
- In the basic computer each instruction cycle consists of following phases

1. Fetch instrunction from memory
2. Decode the instruction
3. Read the effective address from memory, if the instruction has an indirect address
4. Execute the instruction

Based on opcode -



- (12 bit) PC : Prog. cntr
- (12 bit) AR : Address rgstr
- I : MSB
- SC : Sequence cntr
- (16 bit) IR : Instruction rgstr.
- D_7 bit 12 - 14 from LSB

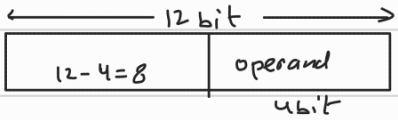
- Based on address / operand -

Zero address instruction / Stack address instruction -

- In '0' address instruction no operand is in instruction
- Instruction has only opcode
- If 12 bit instruction format then all 12 bit represents opcode only then no. of '0' address instruction equal to 2^{12}

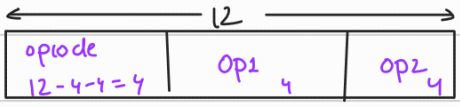
One address instruction -

- In '1' address instruction one operand (address or) is present in the instruction
- If instruction format has 12 bit & 4 bit operand then op-code can be 8 bit & no. of '1' address instruction is 2^8



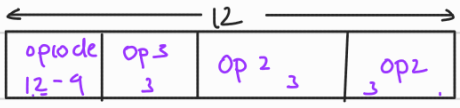
Two address instruction -

- There are two operand in the instruction, if 12 bit instruction format & 4 bit operand the no. of bit in opcode is 4
- No of 2 add. inst. = 2^4



Three address instruction -

- There are 3 operand present in the instruction
- If 12 bit instruction format & 3 bit operand then opcode = 3 bit & no. of 3 address instruction = 2^3



Ex 3add: C = A + B ADD $R_3 \leftarrow R_1, R_2$ In 1 inst. for 3 add.

Ex 2add: C = A + B ADD R_1 R_2 In 2 inst & value of A is lost

Ex 0add: Add 7

8		15
5	⇒	5
6		6
7		7

Computer performance - (Book : Some series CSO)

- While designing computer we have multiple choices we can compare the parts on cost & performance
- Their is trade off b/w^n cost & performance
- Cost ↑ ⇒ performance ↑
- We use a quantitative matric (time or throughput) to measure performance

- Performance ∝ throughput ∝ 1/time
- Diff categories of time on the basis of which performance is measure

Time	Throughput
- Execution time (Time req (only) by CPU to execute a prog. (in sec/prog))	- No of prog executed per unit time (in Prog/s)
- Response time (After initiating cmd the prog is fached by CPU this prog has CP bound inst. & I/O bound inst., so when user initiate cmd first CPU cmd are made in b/w^n if I/O bound instruction come CPU cmd wait) Fetch + execution + waiting time	

- latency Amt. of time for executing 1 inst., inst. latency I/O bound job > CPU bound job as CPU is much faster, if asked net latency we give avg. using probability

  types:
  - CPU latency
  - Memory latency
  - Instruction latency

$P_1 \quad t_1$ (I/O) $\qquad P_2 \quad t_2$ (CPU) latency

$P_1 + P_2 = 1$

$$\text{latency} = \frac{P_1 t_1 + P_2 t_2}{P_1 + P_2}$$

Q. Mach. A runs a prog in 20 s & B runs a prog in 25 s, compare performance of machine A & B ?

- $\frac{\text{Performance of A}}{\text{Performance of B}} = \frac{T(B)}{T(A)} = \frac{25}{20} = \frac{5}{4}$

Q.

Plane	speed (mph)	Range (miles)	Passanger
Airbus-A	610	4150	470
Airbus-B	1350	4000	132

Compare performance of Airbus-A & B ?

- On basis of time or throughput but nothing is stated as time throughput

Prog/sec ≈ miles/sec

$\because \frac{P(A)}{P(B)} = \frac{610}{1350}$

- Considering performace (higher capacity ⇒ higher performance)

∴ take throughput as passanges miles/hour

$\frac{P(A)}{P(B)} = \frac{470 \times 610}{1350 \times 132}$

- ∵ No time for refeulling is given we can't consider range

- For switching b/w^n prog 1 & prog 2 it is known as context switching time, also while switching we need to keep various data this ↓ performance

- For performance CPU utilization dominate
  CPU utilization >>>> context switching time

<u>Diff. cases in which response time, throughput ↑ or ↓-</u>

- ↑ the cpu speed
  - Response time ↓ (As it include execution time)
  - Throughput ↑

- Not changing speed of CPU by ↑ Multiprogramming
  - throughput ↑ (CPU utilization ↑)
  - Response time ↑ (Waiting time ↑ due to context switching)

- No. of prog ↑ & No of comp also ↑ such that comp = prog.
  - throughput ↑ (More comp., more production ⇒ more throughput)
  - Response time will be same.

- In a single comp. we are ↑ the no. of processor along with multiple job
  - throughput ↑ (due to ↑ no of prog.)
  - Response time almost same (some effect is due to context switching)

If Count(CPU) = Count(Prog) no context switching.

<u>CPU time of the prog →</u>

- We compare diff. comp. on bases of CPU time (in sec/prog)
- CPU time is time req by CPU to execute a prog.
- For calculating CPU time we need to has instruction count (i.e no of inst. per prog)
- Diff. computer may perform same operation in diff. way
- Along with inst. count & execution time we need some other factor
- We need to divide inst. in cycle

  '0' - address → 'a' cycle
  '3' - address → 'b' cycle

  } using these kind of division we can calculate

- Two prog: P1: 4 '+' P2: 1 '*' say + req 1 cycle & * req 4 cycle ⇒ the execution time of P1 = P2
- So we can say we need no. of cycle / prog, using this we can evaluate we just need time to complete cycle (sec/cycle)

$$\therefore \text{CPU time} = \left[\frac{\text{no. of cycle}}{\text{no. of prog.}}\right] \times \left[\frac{\text{no. of sec}}{\text{no. of cycle}}\right]$$

- CPI = Cycle per inst.
- In HLL instruction cnt may change due to lang. use
- No. of cycle may change with computer

$$\text{CPU time} = IC \times CPI \times \text{duration of cycle}$$

$$= \frac{\text{Instruction}}{\text{Prog.}} \times \frac{\text{cycle}}{\text{Inst.}} \times \frac{\text{secs}}{\text{cycle}}$$

- $\text{Clock rate} = \frac{1}{\text{Cycle time}} = \frac{\text{cycle}}{\text{sec}}$
- $\text{Throughput} = \frac{1}{\text{Time.}}$

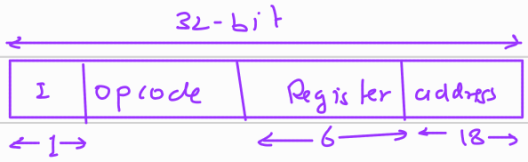
Q A comp. uses a memory unit with 256K words of 32 bit each, a binary instruction code is stored in 1 word of memory the instruction has 4 parts an indirect bit, an operation code, a register code part to specify one of 64 registers and an address part?

a) How many bits are their in the operation code, the register code part and the address part

b) Draw the instruction word format & indicate the no of bits in each part

c) How many bits are there in the data & address input of the memory

if n scal it to nearest 2^n

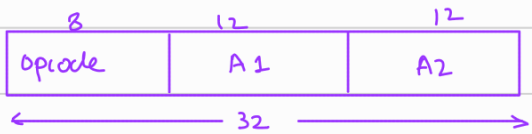


- 64 registers = 2^6 ∴ 6 bit for register
- 1 bit for indirect bit
- Comp memory address 256K = 2^{18} ⇒ 18 address bit
- Opcode = 32 - 18 - 6 - 1 = 7
- Address line bit = 18

  data bit = 32

Q. A comp has 32 bit instruction format and 12 bit address their are 250 2-address instruction how many one address instruction can be formulated

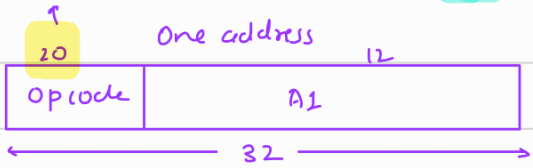
Two address



Opcode = 32 - 2 * 12 = 8

8 are used in 2 address ∴ left = 12

One address



opcode = 32 - 1 * 12 = 20

Max 2-add inst = 2^8 - 250 = 6

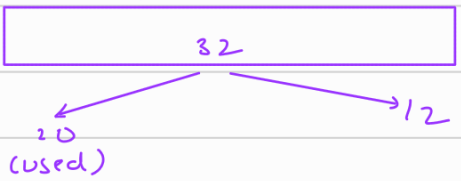
if Max-2 add = 0 the no. of 1-add ≠ 0

No of 1-add inst = 6 * 2^{12}

- i^{th} address is calculated as above i.e it depends of $(i+1)^{th}$ add inst.

Now for 100 1-add inst. find no. of zero add inst

zero add.

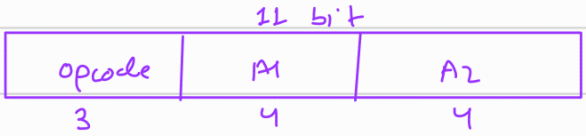


$$(6 * 2^{12} - 100) * 2^{12}$$

Q. In an 11 bit comp inst. format the size of address field is 4 bits the comp uses 5 two-add. inst. and 32 1 add-inst. find no. of zero add. inst. it can support?

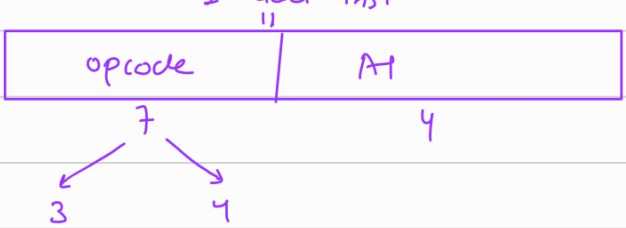
2 - add. inst

11 bit



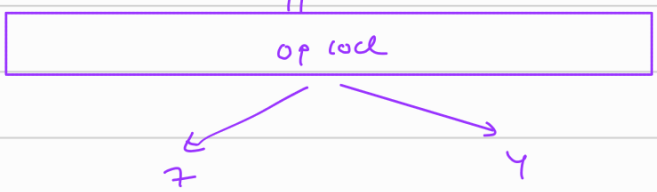
$2^3 - 5 = 3$

1 - add inst



$3 * 2^4 = 3 * 16 = 48$

0 - add inst



$Ans = (48 - 32) * 2^4 = 256$

Q A comp sys has 12 bit inst. format with 4 bit for operand field if their are 120 one-address inst. then find the possible no of 2-add. inst.

$120 = P.\ Unused \times 2^{operand}$

$120 = P.\ Unsued \times 2^4$

$\left\lceil \frac{120}{16} \right\rceil = P.\ unused$

$8 = P.\ unused$

$2^{operand} - used = P.\ unused$

$2^4 - used = 8$

$used = 8$

Addressing mode-

- The add. mode specify a rule for interpreting or modifying the address field of the instruction before the operand is actually referenced, comp. uses add. mode for the purpose of reduce the no. of bits in the addressing field of the instruction.

Opcode	Operand

or
Address to operand
or
Register where operand is present
or
I/O

Types of addressing mode-

- Following notation are used

A = Content of an address field in the instruction

R = Content of an address field of the inst. that refers to a register

EA (Effective Address) = The content of address field in the inst. that reference the address of operent

(x) = Content of memory location X or register X.

- **Implied mode-**
- In this mode the operand are specified implicitly in the definition of the instr.
-

opcode

instruction

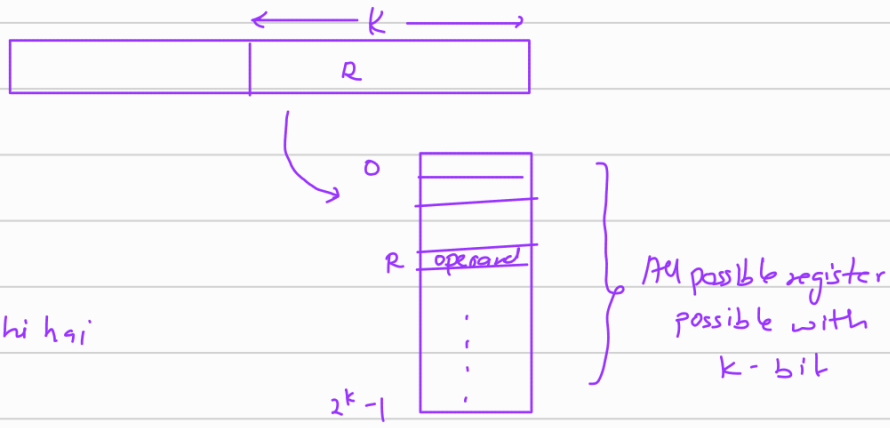
- All the register instruction uses accumulator are implied mode instruction, here no need to specify operand.
- Complement accumulator is an implied mode instruction bcz the operand in the accumulator register is implicitly defined in the instruction.
- All register reference instruction that uses an accumulator are implied mode instr.

- **Immediate mode-**
- Operand are specified in the instruction itself, in this mode of inst. has an operand field rather than address field.
-

opcode	Operand field.

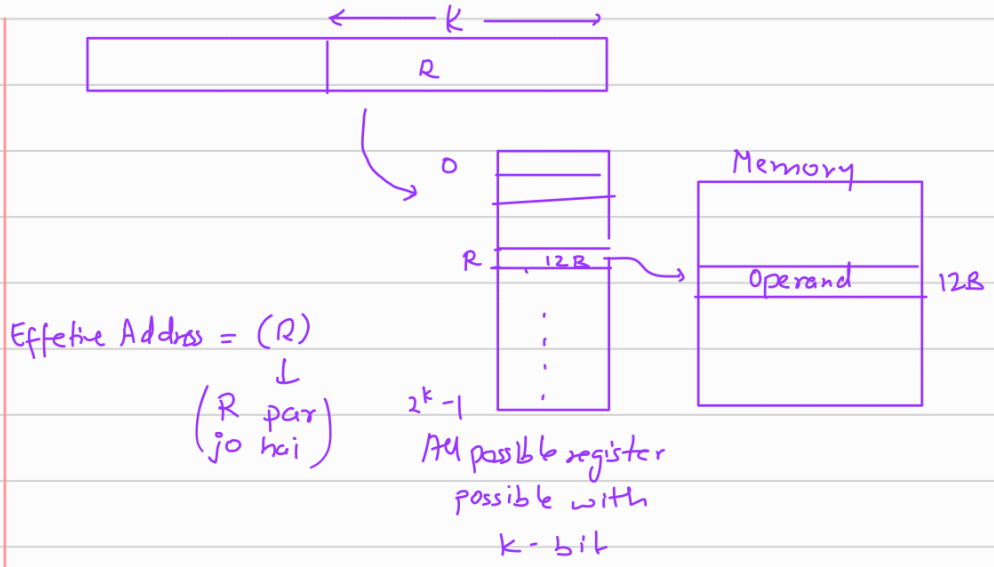
Ex MOV R1 #12B

- **Register mode-**
- In this mode the operands are in registers that store in the CPU, when the address field specify a processor register, the instruction is said to be in the register mode
- particular register is selected from the register field of the instruction, K-bit field specify any one of 2^k registers



Effective Address = R
↳ 'R' hi hai

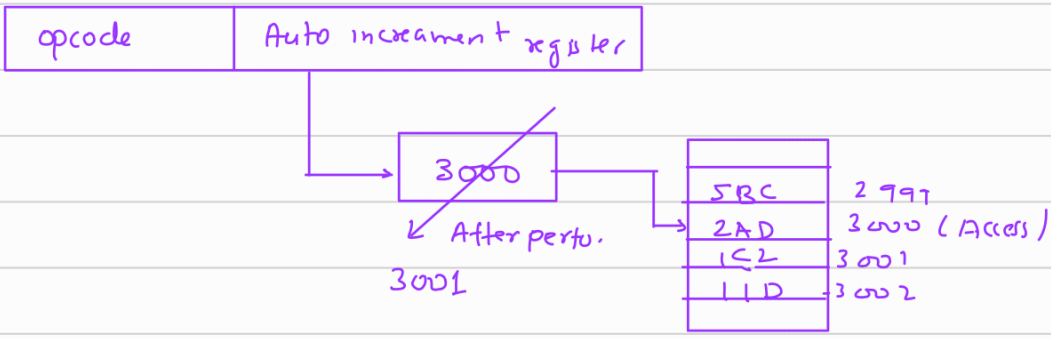
- **Register indirect mode-**
- In this mode the instruction specify register in the cpu whose content give the addresses of operand in memory



Effective Address = (R)
↓
(R par jo hai)

Auto increament addressing mode-

- It is similar to register indirect mode except that register is increament after its value is used to access memory, when the addresses stored in register refers to a table of data in memory

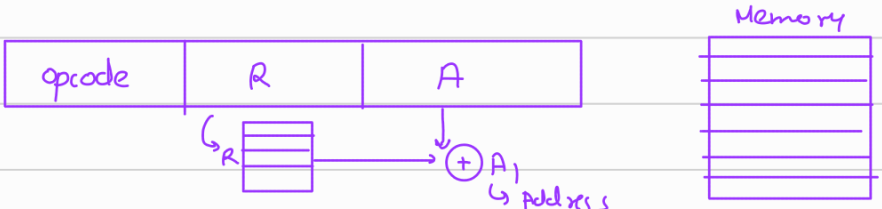


- Reverse happens in auto decrement addressing mode.

Effective address-

- The address field of an instruction is used by the control unit in the cpu to obtain the operand from memory sometime the value given in the address field is the address of the operand but sometime it is just an address from which address of the operand is calculated.

Eff. address or displacement address is the address of operand in a computa-tional type instruction.



$$E_A = A + (R)$$

E_A ↓ Eff. Address

Relative addressing mode -

- In this mode the content of the PC (prog. cntr) is added to the address part of the instruction in order to obtain the effective address

$$EA = X[PC] + A$$

$X[PC]$ ↓ value at PC

Index addressing mode -

- $E_A = X[IND] + A$

$X[IND]$ ↓ value at index register

Base addressing mode -

- $E_A = X[Base] + A$

Q. Find Effective address and content of AC register for different addressing mode

(i) Immediate mode

(ii) Register

(iii) Register indirect

(iv) direct

(v) Indirect

(vi) Auto ↑

(vii) Auto ↓

(viii) Relative

(ix) Index

R_1: 400

XR: 100

AC: ?

200	LDA	mode
201	Address = 500	
202	Next instruction	
399	450	
400	700	
500	800	
600	900	
702	325	
800	300	

(i) EA = 201 X[AC] = 500

(ii) EA = - X[AC] = 400

(iii) EA = 400 X[AC] = 700

(iv) E_A = Address = 500 X [AC] = 800

(v) E_A = 800 X [AC] = 300

(vi) E_A = 400 X [AC] = 700 } if not continous run else 400 700

(vii) E_A = 400 X [AC] = 700 399 450

(viii) E_A = add + X = 600 X [AC] = 900

(ix) E_A = add + NI = 500+202 X [AC] = 300