* Modes of Transfer :-

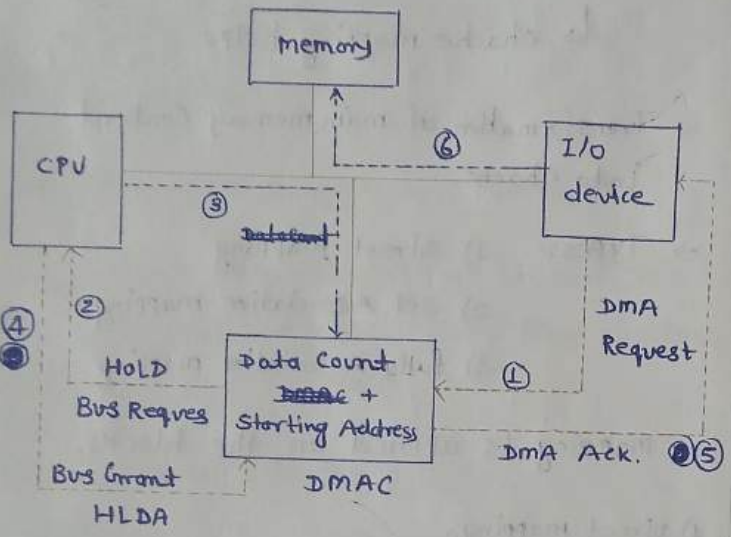
→ B/w Io and CPU

. DMA (Direct memory Access) :-

→ Enables data transfer b/w I/o and memory Without CPU Intervention.

→ Need a Hardware — DMAC

— (DMA Controller)



•Starting Address : Memory Address starting from Where the data transfer Should be transfered.

• Data Count : No. of Bytes or Words to be transferred.

If memory is byte Addressable –

Data Count = No. of Bytes

If memory is Word Addressable –

Data Count = No. of Words

• During DMA Transfer CPU Can perform only those M-operation which do not require system buses.

⇒ if Data Count register has 4 bit means max data Count = $(1111)_2$

$= \underline{15}$

DMAC is a Special purpose processor Which generates address and Control signal for data transfer between memory and I/o.

• Modes of DMA Transfer :-

↳ for how much duration & when the Control of buses should be given to DMAC.

1) Burst mode → old

2) cycle stealing mode → New

3) Interleaving mode

1) Burst mode :-

→ A burst of data (1KB – 2KB generally) is transffered in one time before CPU takes back the Control of the buses.

2) cycle stealing mode :-

→ Slow I/o device takes Some time to prepare a byte or word. during this time CPU keeps the Control of the buses. Whenever the word or byte is ready then CPU give the Control of buses to DMAC for one memory cycle. During this cycle the prepared byte or Word is transffered with memory.

3) Inter Leaving mode :-

→ Whenever CPU performs Internal operation and does not require buses then DMAC will be given the Control of the buses.

→ Hence CPU will Never be blocked due to DMA

- Time required to prepare the data in I/o = t_x
- Time required to transfer the data to memory = t_y

% of time CPU is blocked due to DMA = $\frac{t_y}{t_x + t_y} \times 100$

↳ for burst mode

% of time CPU is blocked due to DMA = $\frac{t_y}{t_x} \times 100$

↳ for Cycle Stealing mode

→ t_x is dependent on device's Internal Speed.

→ t_y ↗ given directly in question
↘ dependent on memory

* Locality of Reference :-

- If CPU has requested one Address for memory Access, then that particular Address or nearby Addresses will be accessed soon.

- Types -

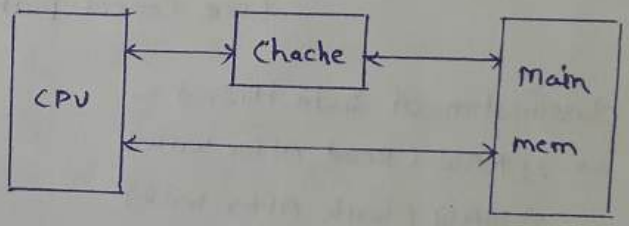
(1) Temporal (based on Time)
↳ If same address is reffered soon.

(2) Spatial (based on Space)
↳ Nearby Address are reffered soon.

* Chache memory :-

→ Based on locality of reference Concept, current demanded localities are kept into a smaller & faster memory known as chache.

→ use of chache reduces CPU memory Access time.



- chache Hit - CPU's demanded Content is present in Chache.
- chache miss - CPU's demanded Content is Not present in chache.

- Performance of Chache is given by Hit Ratio.

$$H = \frac{\text{No. of Hits}}{\text{Total mem. References}}$$

Miss Ratio = 1 - H

* Working of chache :-

→ first CPU search the data in chache memory, if data found, then Processes it.

→ If data Not found in chache then search in Primary memory and loads it into the chache.

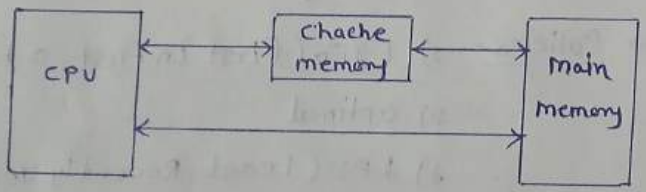
→ frequently Accessed data are always found.

* Average memory Access time -

T_{avg} = H * Mem Access time for Each Hit + (1-H) * mem Access time for Each misc

* Types of chache Access :-

1) Simultaneously Access (Parallel) :-



$$T_{avg} = H * t_{chache} + (1-H)\, t_{mem}$$

2) Hierarichal Access (serial) :- (Level wise)

$$T_{avg} = H * t_{chache} + (1-H)(t_{cm} + t_{mm})$$

$$= H * t_{cm} + t_{cm} - H t_{cm} + (1-H) t_{mm}$$

$$T_{avg} = t_{cm} + (1-H)\, t_{mm}$$

• Block Access time in main mem = Block size × t_{mm}

⇒ T_{avg} When block Transfer included-

1) Simultaneously -

$$T_{avg} = H * t_{cm} + (1-H)(t_{block} + t_{cm})$$

2) Hierarichal -

$$T_{avg} = H * t_{cm} + (1-H)(t_{cm} + t_{block} + t_{cm})$$

* Chache - Write :- (Write Propogation)

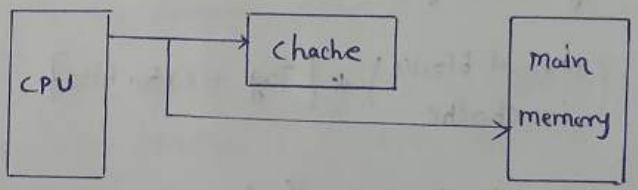
→ If CPU update the Content in chache then the original content in ~~mem~~ main memory also should be updated.

→ Two methods-

1) Write Through

2) write back

1) Write Through :-



→ The write operation is performed in chache and main memory simultaneously

→ ⊕ : 1) Data Consistency

→ ⊖ : 1) Time Consuming
↳ for a write opr^n main memory is accesed irrespective of hit / miss

$$(T_{avg})_{Read} = H * t_{cm} + (1-H)\, t_{mm}$$

or

$$H * t_{cm} + (1-H)(t_{cm} + t_{mm})$$

$(T_{avg})_{write} = \max[t_{cm} + t_{mm}]$

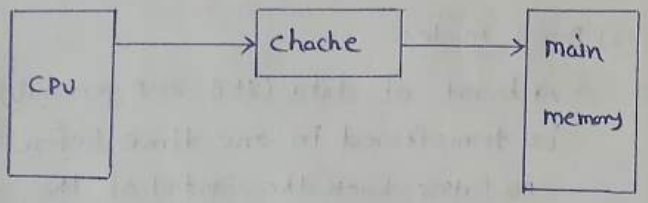
$= t_{mm}$

$T_{avg} = \left(\begin{matrix}\text{fraction}\\ \text{of}\\ \text{Read}\end{matrix}\right) * T_{Read} + \left(\begin{matrix}\text{fraction}\\ \text{of}\\ \text{write}\end{matrix}\right) * (T_{avg})_{write}$

Effective Hit ~~Ratio~~ Rate -

fraction of Read * H for read

(2) Write Back :-



→ CPU updates only chache Content Whenever a block is replaced from chache then it is written back to main memory.

⊕ → 1) ~~Time~~ Time Saving

⊖ → 1) Data Inconsistency

$(T_{avg})_{Read\ or\ write} =$

$H * t_{cm} + (1-H)(t_{mm} + t_{write-back})$

OR

$H * t_{cm} + (1-H)(t_{cm} + t_{mm} + t_{write-back})$

- write - Allocate :-

for a write miss, the missed block should be copied to chache from mem.

- write - No - Allocate :-

for a write miss, the missed block is not copied to chache.

→ for better performance -

write Through - uses No write Allocate

write Back - uses write Allocate

* chache mapping :-

→ using main memory Address how chache memory is searched.

↳ chache mapping helps

→ Transformation of main memory Content into chache.

→ Types :- 1) Direct mapping
2) set Associative mapping
3) fully Associative mapping

→ mapping is applied on the blocks.

1) Direct mapping :-

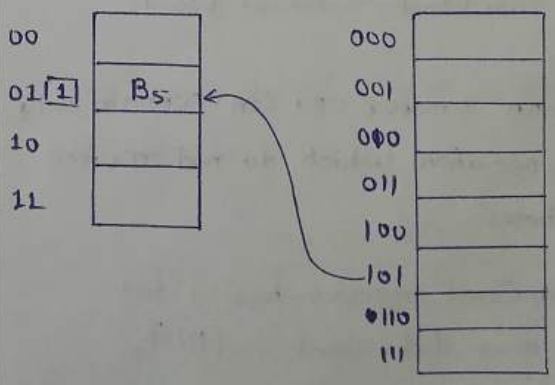
$\text{No. of blocks/ Lines} = \dfrac{\text{Chache size}}{\text{Line or Block size}}$

- Physical Address is divided into 3 parts -

Tag	Number of chache Line	Byte offset

- chache Line Number = M/m block No.

$\left(\begin{matrix}\text{M/m block}\\ \text{No.}\end{matrix}\right) \% \left(\begin{matrix}\text{Number of}\\ \text{blocks in}\\ \text{chache}\end{matrix}\right)$



for M/m = 101

$(101)_2 = 5$, $5\%4 = 1$. $\dfrac{1}{\text{Tag}}\ \dfrac{01}{\text{Line No.}}$

Tag	cm block No.	Byte offset

No. of bits in Byte offset = $\log_2$ (block size)

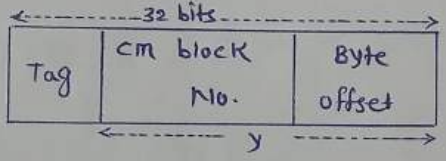
No. of bits in cm block No. = $\log_2$ (No. of blocks in chache)

* No. of tag Info Stored in chache Controller
= No. of block in chache

* Total Tag size stored =
(No. of blocks in chache) * (Tag-size)
→ size of meta data
or
size of Tag directory

Q Consider a direct mapped chache of size 128 KB. The CPU generates of 32 bit. The Number of tag bits in main memory Address are ?

Ans -



$y = \log_2$ (cm size)
$= \log_2$ (128 KB)
= 17 bits

Tag bits = 15 bits

* Index bits = No. of bits in chache mem block No.

• Valid - Invalid bit
↳ There are one Extra bit attach In tag directory that verify that block fetches from main memory or Not?.

Valid bit → 0 → Invalid bit
Valid bit → 1 → Valid bit

• Write - back chache performance
↳ It Can be * Improved if we Can keep track of Each chache block that which is modified & which one is not.
→ modified block only will require write back.
→ We use modifred / dirty bit.

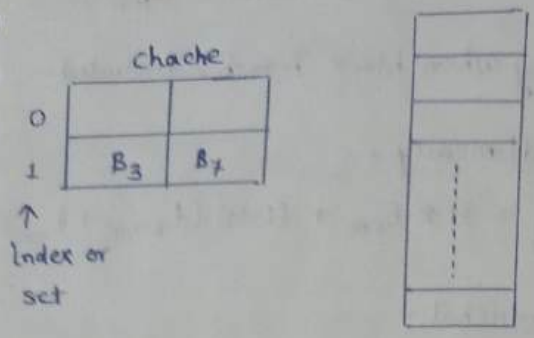
dirty bit → 0 - only read
dirty bit → 1 - modifred

• Tag directory size =
(No. of blocks in chache) * (Tag bits + Extra bits)

• Tavg for write - back chache -
$(T_{avg})_{Read} = (T_{avg})_{write}$
$= H * t_{cm} + (1-H)[t_{block} + x * t_{block}]$
or
$H * t_{cm} + (1-H)[t_{cm} + t_{block} + x * t_{block}]$

x: fraction of dirty block, to be replaced from main memory

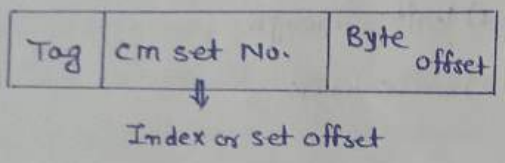
2) Set - Associative mapping :-



* No. of Indexes or No. of sets = $\frac{\text{No. of blocks in chache}}{\text{Associativity}}$

* CM set No. = $\left(\begin{matrix}\text{Mem block} \\ \text{No.}\end{matrix}\right)$ % $\left(\begin{matrix}\text{No. of sets} \\ \text{in chache}\end{matrix}\right)$

* M/m Address -



* Tag directory size -

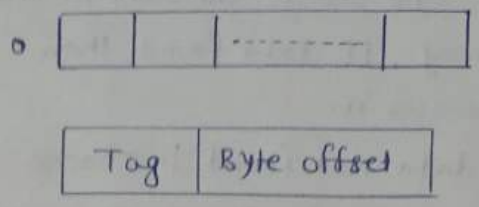
$\left(\begin{matrix}\text{No. of blocks} \\ \text{in chache}\end{matrix}\right)$ * [Tag + Extra bits]

* No. of bits in set offset =

$\log_2$ (No. of sets in chache)

3) fully - Associative mapping :-

Set No. → 0



Q If Cache Contains 2^{16} blocks and main memory Contains 2^{24} blocks then size of chache tag directory size for fully set Associative chache is?

Ans m/m blocks = 2^{24}

= 24 bits

tag bits = 24 bits

Tag directory size = 2^{16} * 24 bits

* Block Replacement Policies :-

→ Direct mapping does not require any replacement policy.

→ Policies - 1) FIFo (First in First out)
2) Optimal
3) LRU (Least Recently Used)

1) FIFo :- The block which Come first in chache should be replaced.

→ Let 4 way set Associative with total 16 chache blocks.

reqvess - (0, 255, 1, 4, 3, 8, 133, 159, 216, 129, 63, 8, 48, 32, 73, 92, 155)

H

S_0	0 48	4 32	8 92	216
S_1	1	133	129	73
S_2				
S_3	255 155	3	159	63

(2) optimal Replacement :-

→ Replace a block which is not going to be used in future for very very long time.

Block Reqves – 2, 3, 4, 7, 6, 3, 4, 7, 5, 4, 7, 8
H H H H H

0	2 6
1	3 5
2	4 8
3	7

(3) LRU (Least Recently used) :-

→ Replace a block which has not been referred since long time.

→ 4way set Associative with 16 chache blocks

↳ Request : (0, 255, 1, 4, 3, 8, 133, 159, 216, 129, 63, 8, 48, 32, 73, 92, 155)

0, 255, 1, 4, 3, 8, 133, 159, 216, 129, 63, 8, 48, 32, 73, 92, 155
H

00	0 48	~~4~~ 32	8	216 92
01	1	133	129	73
10				
11	255 155	3	159	63

* chache miss Penalty :-

→ Time Required to bring a missed block from main memory to ~~ch~~ Cache.

→ Let

cycle Required to send Address to memory = 1 cycle

Cycle require to access 1 main memory cell = 5 cycle

Cycle Require to transfer 1 cell data to chache = 1 cycle

- if cache block size = 4B
  main memory block size = 1B

⇒ miss penalty = 1 + (4 * 5) + (4 * 1)
= 25 cycles

→ Types –

1) Cold or Compulsory miss
2) Capacity miss
3) Conflict miss

(1) Cold or Compulsory miss

↳ First time access of a block will always cause a miss.

↳ To Reduce cold miss –
Increase Block size

(2) Capacity miss :-

→ If it's not a Cold miss and miss occurs because chache is full.

→ To reduce capacity misses :-
" Increase the chache size "

(3) Conflict miss :-

↳ It's not a cold and Capacity miss but due to the # set is full and miss occurs.

↳ To reduce Conflict misses -

→ No Conflict miss in fully Associative mapping.

• multilevel chache :-

1) Simultaneous Access -

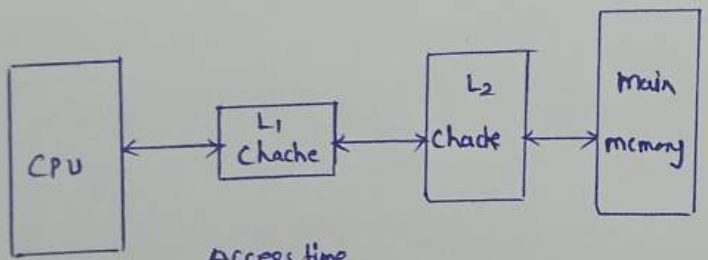
$$T_{avg} = H_1 * t_1 + (1-H_1) H_2 * t_2 + (1-H_1)(1-H_2) * t_{mm}$$

2) Hierarichal Access -

$$T_{avg} = H_1 * t_1 + (1-H_1) H_2 * (t_1 + t_2) + (1-H_1)(1-H_2) * (t_1 + t_2 + t_{mm})$$

or

$$= t_1 + (1-H_1)[t_2 + (1-H_2) t_{mm}]$$



Access time t_1 t_2

Hit ration H_1 H_2