Over the non-pipelined processor (round off to two decimal places) is _____

Cycle time = $\frac{1}{2\text{GHz}}$ = 0.5 nsec.

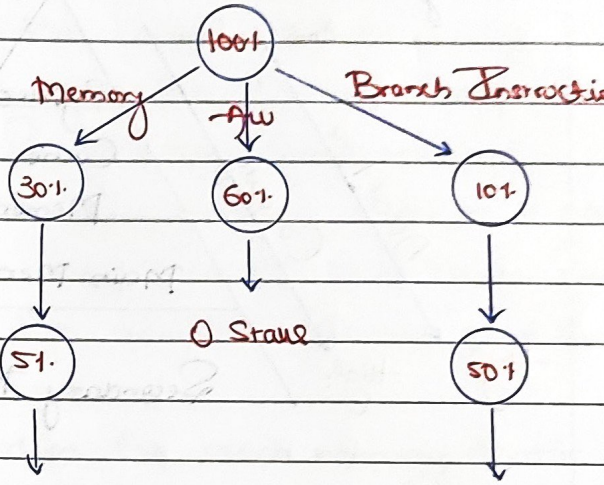
stalls/instructions =

.30 × 0.05 × 50

+ .60 × 0

+ .10 × .50 × 2

⟹ 0.85



Avg Inst. ET = (1 + 0.85) × 0.5

= 1.85 × 0.5

= 0.925 nsec

Nonpipeline: 5 clock cycle

2.5 GHz

Cycle time = $\frac{1}{2.5}$ sec = 0.4 nsec

ET nonpipe = 5 × 0.4 nsec

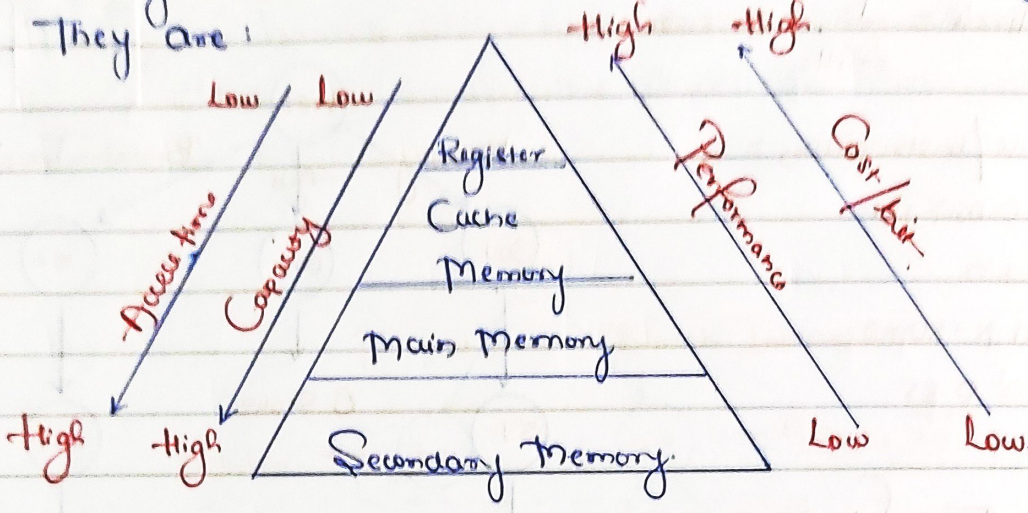
= 2 nsec

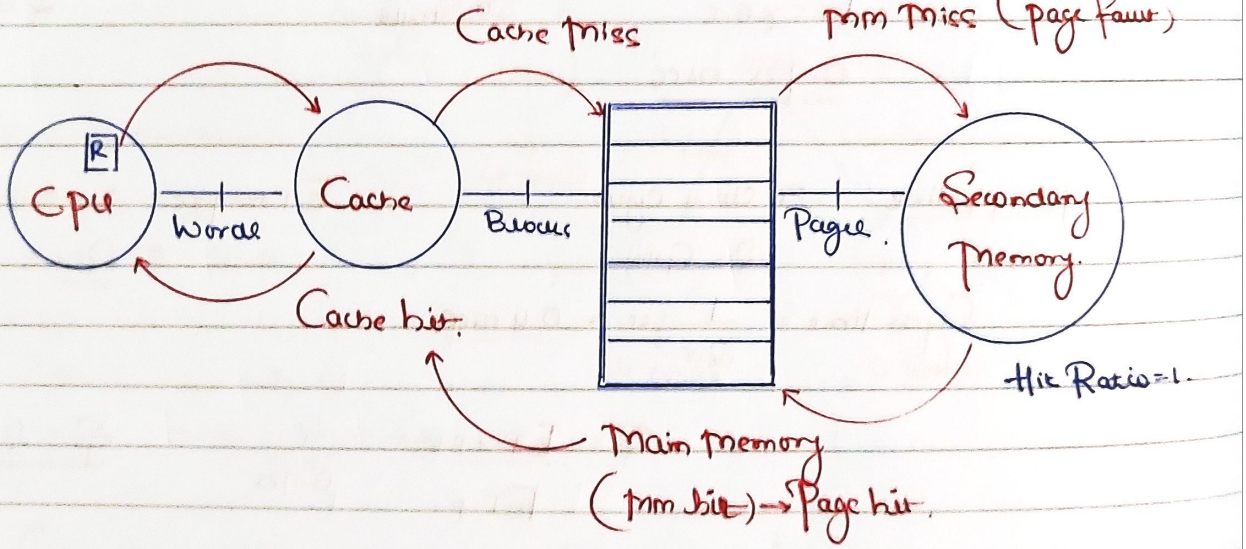
$S = \frac{ET_{NP}}{ET_P} = \frac{2}{0.925}$ ∴ S = 2.16 (Ans)

CACHE MEMORY

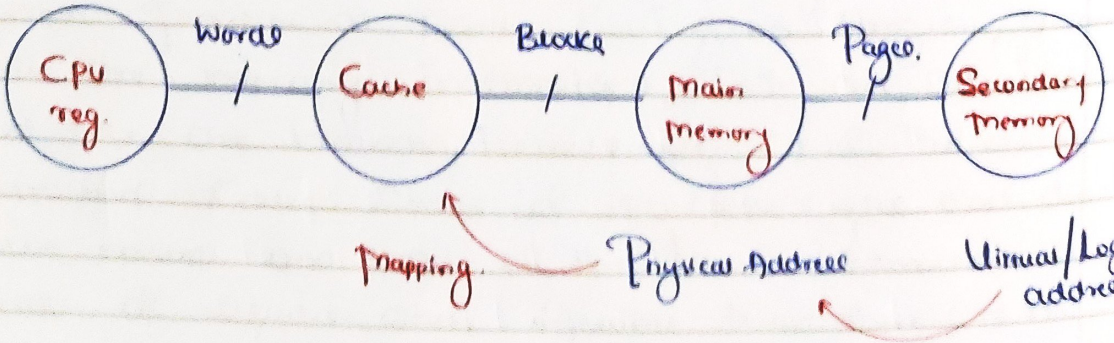
Page No.:
Date: 11/10/23

Memory Hierarchy: Hierarchy design organise the system's memory into 4 levels to minimise the accessing times.
They are:





Hit Ratio = Number of hit / Total number of access



Memory:

1. Cpu generate request initially refer to the cache.
2. If the reference (respective data) found in the cache that is called cache hit (operation is called hit) then respective data give from cache to Cpu in the form of words.
3. If the reference is not found in the cache, that is known as cache miss, the refrence is forwarded to main memory.
4. If the reference found in the main memory then it is called mm hit or page hit then respective data is given from main memory to cache in the form of blocks and cache to Cpu in the form of words.
5. If the reference is not found in the main memory that is called Mm miss or page fault then the response is forwarded to Secondary memory. So respective data is transferred from secondary memory to main memory in the form of pages, main memory to cache memory in the form of blocks, then cache memory to Cpu in the form of words.

Note: * Secondary memory is the last level of memory in which hit ratio is always "1".

* The Process of transfer the data from main memory to cache memory is called "mapping".

Average Memory Access Time [Tavg]

$$T_{avg} = [H] \times \text{Time taken by memory when there is a hit} + (1-H) \{ \text{Time taken by memory when there is a miss} \}$$

Q1. Consider a Cpu Generate 100 Request. Out of 100 requests 90 times hit and 10 times miss. When there is a hit then time taken is 20nsec and when there is a miss then time taken is 150 nsec. Calculate the Tacg?

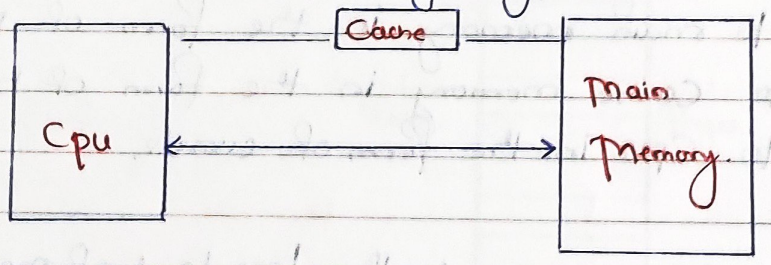
Ans.

Total Cpu Request = 100
Hit = 90 times
Miss = 10 times
Hit will take 20 nsec
Miss will take 150 nsec.
Total time = 90×20 + 10×150
= 1800 + 1500
= 3300 nsec.
Tavg = 3300/100 = 33ns (Ans)

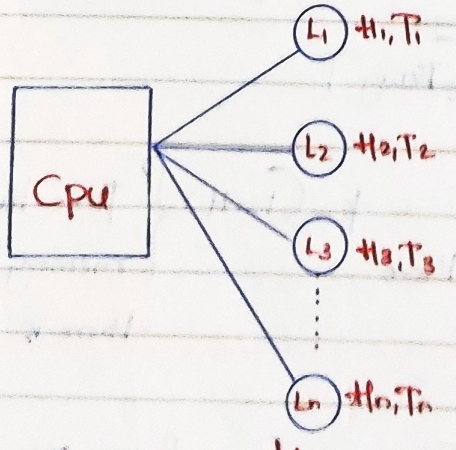
Total Cpu Request = 100
Hit = 90 times, Miss = 10 times
Hit Ratio = 90/100 = 0.9
Tavg = H × Time taken when there is hit + (1-H)(T. when there is miss)
= 0.9 (20 + 0.1 (150)
= 18+15
Tavg = 33 nsec (Ans)

Types of Memory Organisation:

1. **Simultaneous Access Memory Organisation:**



$T_{avg} = h * t_c + (1-h) t_m$ → h: hit ratio
t_c: Cache access time
t_m: main memory access time



In the Simultaneous access organisation all the level of memory is directly connected to Cpu. But forow in sequence (access in a sequence) when there is a miss in level 1 memory then reference forward to level 2 memory.

When there is also in level 2 memory then directly data is transferred from level 2 to Cpu without copying into level 1 memory.

When there is a miss in level 1 and level 2 memory and hit in level 3 memory then directly data is transferred from level 3 memory to CPU without copying into level 1 and level 2 memory.

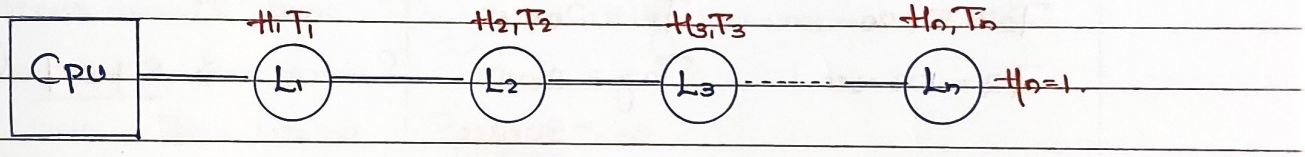
Here $h_1, h_2, h_3 \ldots$ are the hit ratio and $T_1, T_2, T_3, \ldots T_n$ are the access of respective level memory.

The time required to access 1 word from memory.

$$T_{avg} = h_1 t_1 + (1-h_1) h_2 t_2 + (1-h_1)(1-h_2) t_3 h_3 + \ldots (1-h_{n-1}) t_n h_n$$

* In last level hit ratio = 1.

② Hierarchial Access Memory Organisation



When there is a miss in level 1 and level 2 memory and hit in level 3 memory then firstly data is transferred from level 3 [L_3] memory to level 2 [L_2] memory then level 2 [L_2] memory to level 1 [L_1] memory then from level 1 [L_1] memory to CPU.

$$T_{avg} = h_1 T_1 + (1-h_1) h_2 (T_2+T_1) + (1-h_1)(1-h_2) h_3 (T_3+T_2+T_1) + \ldots (1-h_{n-1}) h_n [T_n + T_{n-1} \ldots T_2 + T_1]$$

Note: In an question if they have mentioned the keyword 'hierarchial access' or 'level of access' or hierarchial then it means using "Hierarchial Access."

Q2. Calculate the average access time with the cache access time 10ns, and main memory access time 100ns, Hit ratio 90.1.? Using hierarchial access?

Ans. $t_c = 1$ nsec $\quad h = 90.1\% = 0.9$

$t_m = 100$ nsec

$T_{avg} = h * t_c + (1-h)(t_m + t_c)$

$= 0.9 \times 1 + (1-0.9)(100+1)$

$= 0.9 + (0.1)(0.1)$

$\Rightarrow$ 11 nsec

Q3. In a 2 level memory, level 1 memory is 5 times faster than level 2. And its access time is 10ns < Average access time. If level 1 access time is 20ns, what is the hit ratio? Using Simultaneous Access org?

Ans. $L_1 = T_1$ $\quad$ $L_2 = T_2$.

$$5 = \frac{T_2}{T_1}$$

$$\boxed{T_2 = 5 * T_1}$$

$T_2 = 5 \times 20$

$T_2 = 100$ nsec

$$5 = \frac{P_{L1}}{P_{L2}} = \frac{1/T_1}{1/T_2}$$

$T_1 = T_{avg} - 10$

$T_{avg} = T_1 + 10$

$T_1 = 20$ nsec

$T_{avg} = 20 + 10$

$= 30$ nsec

$T_{avg} = h * t_c + (1-h) * t_m$

$30 = h \times 20 + (1-h)\,100$

$= 20h + 100 - 100h$

$80h = 70$

$h = \frac{70}{80} = 0.875$

$\Rightarrow$ 87.5%. (Ans)

Q4 Assume that for a certain Processor, a read request takes 50 nsec on a Cache miss and 5 nsec on a Cache hit. Suppose while running a program, it was observed that 80% of the processor read request result in Cache hit. The average read access time in nsec is _____.

Ans Hit Ratio $= 80 = 0.8$ $\quad$ When there is hit, time = 5

Miss ratio $= (1-0.8) = 0.2$ $\quad$ When there is miss, time = 50

$T_{avg} = 0.8 \times 5 + 0.2 \times 50$

$= 4 + 10$

$= 14$ nsec (Ans)

Hierarchial Access:

$T_{avg} = h * t_c + (1-h)[t_m + t_c]$

$\Rightarrow h t_c + t_m - h t_m + t_c - h t_c$

$\Rightarrow t_c + t_m - h t_m$

$$\boxed{T_{avg} = t_c + (1-h)\, t_m.}$$

eg. Hierarchial access bit ratio 80:1, tc = 20ns, tm = 100 nsec

$T_{avg} = h * t_c + (1-h)[t_m + t_c]$

$= 0.8 \times 20 + (1-0.8)(100+20)$

$= 16 + 0.2(120)$

$T_{avg.} = 40$ nsec

$T_{avg} = t_c + (1-h) t_m$

$= 20 + (1-0.8)100$

$= 20 + (0.2)100$

$T_{avg.} = 40$ nsec.

Qs. Consider a system with 2 levels. Level 1 Access time is 20ns less 2 Access time = 150ns T_{avg} = 30 using simultaneous access.

(i) What is the bit ratio?

Ans $T_1 = 20ns$, $T_2 = 150$ ns $T_{avg} = 30ns$

$T_{avg} = ht_1 + (1-h) t_2$

$30 = h*20 + (1-h)150$

$30 = 20h + 150 - 150h$

$120 = 130h$

$h = \frac{120}{130} = 0.923 \Rightarrow 92.3\%$

(ii) If the bit ratio is made to 100:1 then what is the access time of L_1 and L_2 memory?

Ans $T_1 = 20ns$, $T_2 = 150$ ns

L_1 and L_2 access remain same, $h = 1$

If bit ratio = 100:1

$T_{avg} = ht_1 + (1-h) t_2$

$= 1 \times 20 + (1-1)150$

$T_{avg} = 20$ nsec.

} Only T_{avg} changes

Q6. In the above question if T_{avg} is increased by 10% then what is the % change in the bit ratio?

Ans. $T_1 = 20$, $T_2 = 150$ nsec, $T_{avg} = 30$ nsec + 10% = 33 nsec

$T_{avg} = h_1 * t_1 + (1-h_1)\, t_2$

$33 = h * 20 + (1-h)\, 150$

$33 = 20h + 150 - 150h$

$130h = 117$

$h = \frac{117}{130} \Rightarrow h = 0.9$

$= 90\%$

Previous hit ratio = 92.3%

$\Rightarrow$ Hit ratio decreased by 2.3%

Note: * 1 Word Access Time = T_{avg}

$$\#\text{words/sec} = \frac{1}{T_{avg}}$$ Data Transfer Rate.

* Hierarchial Access Memory org: 3 level

h_1: hit ratio in level 1

$(1-h_1)$: miss ratio in level 1 (m_1)

$(1-h_2)$: miss ratio in level 2 (m_2)

$$T_{avg} = h_1 t_1 + (1-h_1) h_2 (t_2+t_1) + (1-h_1)(1-h_2) h_3 (t_3+t_2+t_1)$$

$$T_{avg} = h_1 t_1 + m_1 h_2 (t_2+t_1) + m_1 m_2 h_3 (t_3+t_2+t_1)$$

m_1: Miss ratio in level 1.

Q7. Assume that for a certain processor, a read request takes 50 ns on a cache miss and 5 ns on a cache hit. Suppose while running a program, it was observed that 80% of the processor's read request result in a cache hit. The average read access time in nanoseconds is _____.

(i) What is the data transfer rate (performance) of this memory system (in words/sec)?

Ans. Data Transfer rate $= \frac{1}{T_{avg}}$ words/sec

$= \frac{1}{14 \times 10^{-9}}$ words/sec

$= \frac{1}{14} \times 10^9$ words/sec

$\frac{1000}{14} \times 10^6$

$= 71.42 \times 10^6$

$= 71.42$ million words/sec

$= 72$ million words/sec (Ans)

(ii) What is the bandwidth required of this memory System if word size is 8 bit?

Ans Bandwidth: Word size = 8 bits

72 million words/sec = 72×10^6 words/sec

$= 72 \times 10^6 \times 8$ bits per sec

$= 72 \times 8$ m bit/sec

$= 576$ m bits/sec or 72 mBytes/sec (Ans)

Locality of Reference

Accessing the higher level of memory data from the level 1 memory (Cache memory) is called locality of refrence (Faster Memory)

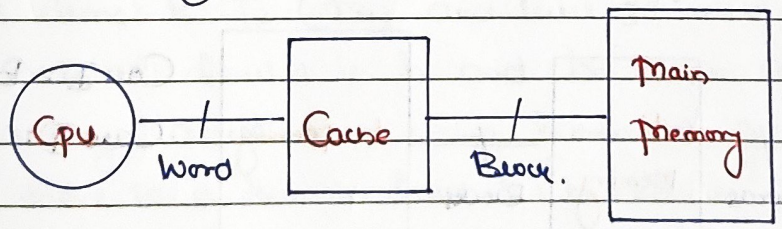
1. Temperal LOR
2. Spatial LOR

(1.) Temporal LOR: This means the same word in the same block is referenced by the CPU in near future (frequntly).

Or

Same data which access again and again then that type of data Stored in Temporal LOR.

(2.) Spatial LOR means adjacent word is in the same block is referenced by the CPU in a Sequence.



Assume:

1 Block Size = 32 Byte / 32 words

(32 Byte / 32 word)

If miss in Cache then 1 complete block is transfered from main memory to Cache, the respective word (which is requested by the CPU) given from Cache memory to CPU.

(Next reference there is a Cache hit)

Q.1 Here 1 block Size = 32 word/32 Byte & CPU require only 1 word?

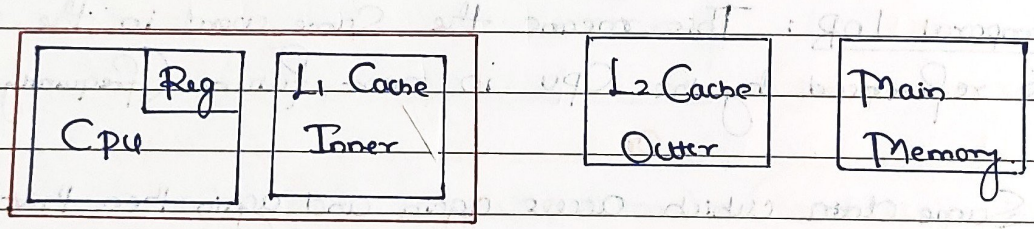
Ans In this process, 1 complete block of 32 words/32 Byte transfered

from main memory to cache memory then requested 1 word is transferred from cache memory to CPU.

Note: So in the next instructions time when CPU request some word or adjacent word then that request available in cache (cache hit)

Types of Cache

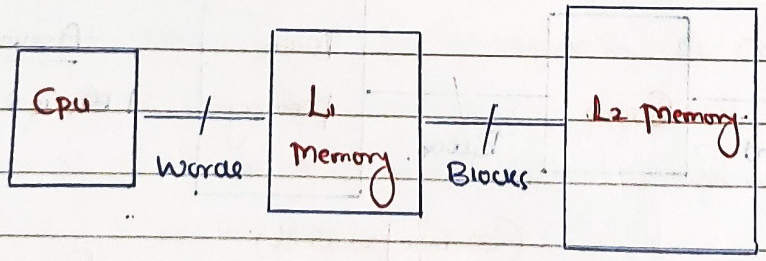
1. Unified Cache: Instruction and data both are placed in same cache.
2. Split Cache: This cache logically divided into two parts
   (i) Instruction Cache (I-cache)
   (ii) Data Cache (D-cache)
3. Multilevel Cache:



Size $L_1 < L_2$

Speed $L_1 > L_2$

2 Level of Memory (If locality of reference included)



Case I: Block Size = 1 word

Case II: Block size = n words

CPU always access the data from the fastest memory (level 1 / cache memory). If there is a miss in level 1 (cache) memory then one complete block is transferred from level 2 (slow) memory to level 1 (cache) memory and addressed word (which is requested) that respective word given from cache memory to CPU

T_B: Block transfer time from L_2 memory to L_1 memory.

→ Case 1: Block size = 1 word

$$T_B = T_2$$

→ Case 2: Block size = n words

$$T_B = n * T_2$$

If 2 level memory:

$h_2 = 1$

$T_{avg} = h t_1 + (1-h)(t_2 + t_1)$

or

$T_{avg} = t_1 + (1-h)\, t_2$

If locality of reference included ($T_B = n * T_2$)

$T_{avg} = h t_1 + (1-h)[T_B + T_1]$

or

$T_{avg} = t_1 + (1-h) T_B$.

If 3 level memory:

$h_3 = 1$.

$T_{avg} = h_1 t_1 + (1-h_1) h_2 (t_2 + t_1) + (1-h_1)(1-h_2)(t_3 + t_2 + t_1)$

If locality of reference included:

$T_{avg} = h_1 t_1 + (1-h_1) h_2 (T_{B_1} + T_1) + (1-h_1)(1-h_2)[T_{B_2} + T_{B_1} + T_1]$

Q1. In a 3 level memory, level 1 memory access time is T_1, level 2 memory access time is T_2 (T_{B_1}) and level 3 memory access time is T_3 (T_{B_2}). Hit ratio of level 1 is h_1 and hit ratio of level 2 is h_2. What is the average access time using hierarchical access?

(i) If there is a hit in level 1 ($h_1 = 100\%$)?

Ans.

$h_1 = 100 \Rightarrow 1$

$(1-h_1) = 0$ $\quad 0 * x = 0$

$T_{avg} = T_1$.

$h_1 = 100 \Rightarrow h_1 = 1$

$(1-h_1) = 0 \Rightarrow 0 * x = 0$

$T_{avg} = T_1$

(ii) If there is a miss in level 1 and hit in level 2 ($h_2 = 100\%$)

Ans.

$h_1 = 0$ & $h_2 = 100 \Rightarrow h_2 = 1$

$1 - h_2 = 0$

$T_{avg} = T_2 + T_1$

$h_1 = 0$, $h_2 = 1$ $\quad (1-h_2) = 0$

$T_{avg} = T_{B_1} + T_1$

(iii) If there is a miss in level 1 and level 2 and hit in level 3

Ans. $h_1=0, h_2=0, h_3=1$ | $h_1=0, h_2=0, h_3=1$

$T_{avg} = T_3 + T_2 + T_1$ | $T_{avg} = T_{B_2} + T_{B_1} + T_1$

Q2 In a 2 level memory. level 1 memory access time is 30ns and level 2 memory access time is 250ns/word. Hit ratio of level 1 is 90%. If there is a miss in level 1 then 4 word block must be transferred (moved) from level 2 into level 1 and then addressed words is given to the CPU. What is the average access time?

Ans. $h_1 = 90\% = 0.9$ $t_1 = 30$ nsec

$t_2 = 250$ nsec/word

$T_{B_1} = 4\text{ word} \times 250\text{ ns/word} = 1000$ nsec.

$T_{avg} = h_1 t_1 + (1-h_1)(T_{B_1} + T_1)$

$= 0.9[30] + (1-0.9)[1000+30]$

$= 27 + 103 \Rightarrow 130$ nsec (Ans)

Q3. 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 to offset the optimization such that it should not increase the average memory access time is _____.

Ans $h = 0.8$, $t_c = 10$ nsec, miss penalty $(t_m) = 100$ nsec

$T_{avg} = ht_c + (1-h)(t_m + t_c)$

$= 0.8 \times 10 + (1-0.8)(100+10)$

$= 8 + 0.2(110) \Rightarrow 30$ nsec

New Optimization : $t_{c\,new} = 15$ nsec

$T_{avg(new)} = h * t_c + (1-h)(t_m + t_c)$

$30 = h \times 15 + (1-h)(100+15)$

$30 = 15h + 115 - 115h$

85 = 100% ∴ 1% = $\frac{85}{100}$ = 0.85 (Ans)

Q4. A direct mapped cache memory of 1MB has a block size of 256 Bytes. The cache has an access time of 3ns and a hit rate of 94%. During a cache miss, it takes 20ns to bring the first word of a block from the main memory, while each subsequent word takes 5ns. The word size is 64 bit. The average memory access time in ns is _____

Ans. Block Size = 256 Byte

1 word Size = 64 bit = 8 Byte

No. of words = $\frac{256 \text{ Byte}}{8 \text{ Byte}}$ = 32 words

1 word takes 20nsec

Remaining 31 words takes 5ns.

$T_{avg} = .94 \times 3 + (1-0.94)(3+20+31(5))$

$= 2.82 + (0.06)(3+20+155)$

$= 13.5$ nsec (Ans)

Basic Elements of Cache Design:

1. Memory Organisation
2. Mapping technique → i. Direct mapping
3. Replacement algorithm → ii Set associative mapping
4. Updating technique → iii Fully associative mapping.
5. Multi level Cache

* In the cache design, data is transfered from main memory to cache memory in the form of blocks.
* So both memory (main memory and cache memory) must be organised based on block size.
* Main memory and cache memory are divided into parts (equal size parts) based on block size called Mm Block and Cm Block (number of lines) respectively.

$\#\text{Mm Blocks} = \frac{\text{Mm Size}}{\text{Block Size}}$

$\#\text{Cm Blocks} = \frac{\text{Cm Size}}{\text{Block Size}}$

Cache Memory:

Before organisation:

CM = 8Byte

000	1 Byte
001	1B
010	1B
011	1B
100	1B
101	1B
110	1B
111	1B.

Cache index = 3bit

*Consider a 8 Byte Cache with 2Byte Block size then CM organisation is as follows:

After Organisation

00	0	2Byte
	1	
01	0	2Byte
	1	
10	0	2Byte
	1	
11	0	2Byte
	1	

line offset | word offset

$$\#\text{CM Lines} = \frac{\text{CM Size}}{\text{Block Size}}$$

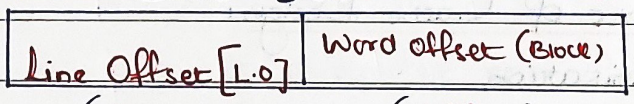
$$= \frac{8B}{2B} = 4$$

CM Size = 4 x 2Byte

= 8 Byte (After Organisation)

Before and after the organisation Cache memory Capacity is same but internal structure is different.

Address



(2 bit) → #lines/Blocks in the cache memory.

$\lceil \log_2 4 \rceil = 2$ bit $\quad \lceil \log_2 \#\text{lines} \rceil$

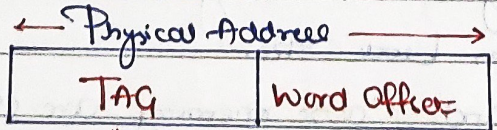
(1 bit) → word address in a block

$\lceil \log_2 \text{Block Size} \rceil$

$\lceil \log_2 2 \rceil = 1$ bit

Physical Address is interpreted as

MM Size > CM Size



No. of main memory blocks → $\lceil \log_2 \#\text{MM Blocks} \rceil$

Word address in a block → $\lceil \log_2 \text{Block Size} \rceil$

Q1. Consider a Cache memory which is indexed using 16 bit address, organised into 32 byte Block size and Physical address is 32 bits.

i. # Cm Block
ii. # mm Block
iii. Cm Address format
iv. mm Address format.

Ans. Cache Index = 16 bit → Cm Size = 2^{16} Byte = 64 KByte

Block Size = 32 Byte

Block/word offset = $\lceil \log_2 32 \rceil$ = 5 bits

Physical address = 32 bit

mm Size = 2^{32} Byte = $2^2 \cdot 2^{30}$ Byte = 4 GByte.

(i) # Cm Blocks = $\frac{\text{Cm Size}}{\text{Block Size}} = \frac{64\,KB}{32B}$

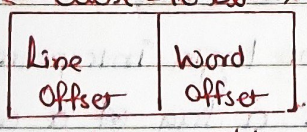
$= \frac{2^{16}B}{2^5B} = 2^{11} = 2K$ L.O = 11 bit

(ii) # mm Block = $\frac{\text{mm Size}}{\text{Block Size}} = \frac{4GB}{32B} = \frac{2^{32}B}{2^5B}$

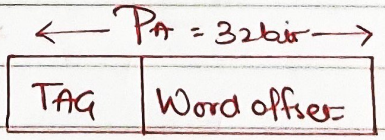
$= 2^{27}$

= 128 M Blocks

(iii)



(iv)



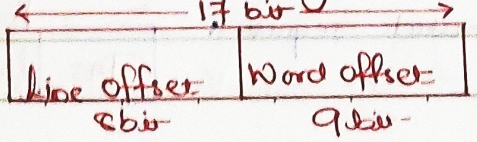
Q2. Consider a System mm = 256 MB, Cache = 128 KB, and Block Size = 512 Byte then

1. # lines.
2. L.O and W.O Format
3. # mm Blocks
4. TAG and W.O Format.

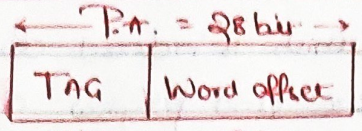
Ans. mm Size = 256 MB = 2^{28} Byte

Cache Size = 128 KB = 2^{17} Byte

Block Size = 512 B = 2^9 Byte. W.O = 9 bit.



Lines = 2^8 = 256 Lines



$\#$ min. Block $= 2^{19} = 512$ K Blocks

Mapping: The process of transfering the data from main memory to Cache memory is called mapping. There are 3 types of mapping:

1. Direct mapping
2. Set Associative mapping
3. Fully associative mapping.

Mapping function: * Because there are fewer Cache lines than main memory blocks, an algorithm is needed for mapping main memory blocks into Cache lines.

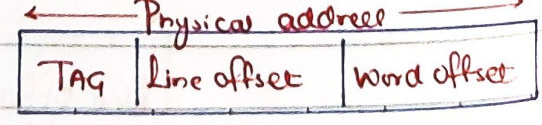
* Three techniques can be used:

(1) Direct: * The simplest technique
* Maps each block of main memory into only one possible Cache lines.

(2) Associative: * Permits each main memory block to be loaded into any line of the Cache.
* The Cache control logic interprets a memory address simply as a Tag & a word field.
* To determine whether a block is in the Cache, the Cache control logic must simultaneously examine every line's Tag for a match.

(3) Set Associative: * A compromise that exhibits the strengths of both the direct and associative approaches while reducing their disadvantages.

Direct Mapping: In this direct Cache technique interprets the CPU generated request:



Line = Cm Size / Block Size

Word offset = $\log_2$(Block Size)

Line offset = $\log_2$(Line)

→ TAG = Physical Address − (Line offset + Word Offset)

→ TAG memory Size = # Lines × Tag bits (depend on the mapping technique)

Q3. Consider a direct mapping if the size of the Cache memory is 512 KB and main memory 512 MB and Cache line size (Block) is 64 KB the Calculate the bit required for

(i) PA (iii) L.O (v) # lines

(ii) TAG (iv) W.O (vi) Tag memory Size

Ans Cache size = 512 KB ⇒ 2^{19} Byte

MM Size = 512 MB ⇒ 2^{29} Byte PA = 29 bit.

Block Size = 64 KB = 2^{16} Byte W.O = 16 bit

← PA = 29 bit →

TAG	L.O	W.O
10 bit	3 bit.	16 bit.

lines = Cm Size / Block Size = 512 KB / 64 KB = 2^{19} Bit / 2^{16} bit

= 8 lines

Tag memory Size = # lines * Tag bit Line Offset = 3 bit

= 8 × 10 ⇒ 80 bits

Q4 Consider a direct mapping, Cache Size = 64 Byte, Line Size = 8 Byte, MM = 256 Byte then # bits for P.A, TAG, L.O, W.O, Tag memory Size?

Ans ← PA = 8 bit →

TAG	L.O	W.O
2 bit	3 bit	3 bit

Line = Cm Size / Block Size = 64 B / 8 B = 8 Lines

L.O = 3 bit

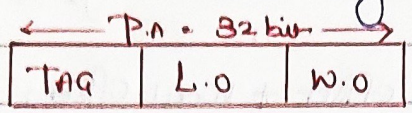
Tag Memory Size = # Lines × Tag bits

= 8 × 2 ⇒ 16 bits

Q5. Consider a machine with a byte addressable main memory of 2^{32} bytes, divided into blocks of size 32 bytes. Assume that a direct

mapped cache having 512 cache lines is used with this machine. The size of the tag field in bits is 18

Ans



TAG = 32 − (9+5)

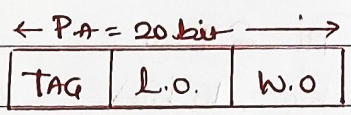
⇒ 18 bits (Ans)

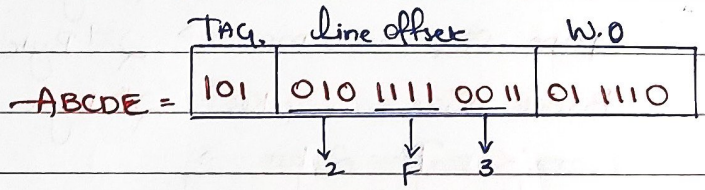
Q6. Consider a direct mapping cache Size = 128KB, Line Size = 64B. Main memory is 1MB then what is the line number of physical address $(ABCDE)_{16}$?

Ans MM = 1MB = 2^{20}B ⇒ P.A = 20 bit

Line size = 64B = 2^6B ⇒ W.O = 6 bit

$$\#\text{lines} = \frac{\text{Cm Size}}{\text{Block Size}} = \frac{128\text{KB}}{64\text{B}} = \frac{2^{17}\text{B}}{2^6\text{B}} = 2^{11}\text{ lines}$$





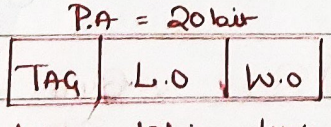
TAG = 20 − (11+6)

= 3 bit

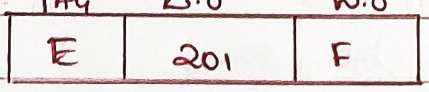
= $(2F3)_{16}$ (Ans)

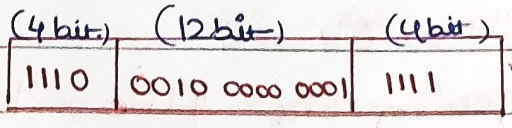
Q7. Consider a machine with a byte addressable main memory of 2^{32} bytes, block size of 16 bytes and a direct mapped cache having 2^{12} cache lines. Let the addresses of two consecutive bytes in main memory be $(E201F)_{16}$ and $(E2020)_{16}$. What are the tag and cache line address (in hex) for main memory address $(E201F)_{16}$?

Ans



$(E201F)_{16}$



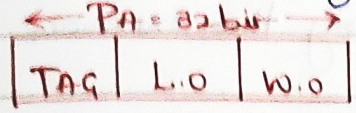


⇒ E, 201 (Ans)

Q8. Consider a machine with a byte addressable main memory of 2^{32} bytes divided into blocks of size 32 bytes. Assume that a direct mapped cache having 512 cache lines is used with this machine

The size of the tag field in bits is _____

Ans



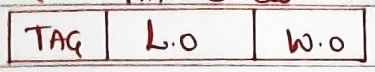
TAG = 32 - (9+5)

⇒ 18 bit (Ans)

Q9. 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 32KB (1KB = 2^{10} bytes) and each cache block of size 64 bytes.

The size of tag field is 17 bits.

Ans



$\#Lines = \frac{Cm\ Size}{Block\ Size} = \frac{32KB}{64B} = \frac{2^{15}B}{2^{6}B}$

$= 2^9$ Lines

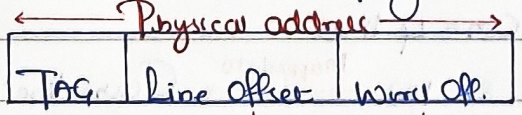
L.O = 9 bit

TAG = 32 - (9+6)

= 17 bit (Ans)

Direct Mapping (Continued)

In direct mapped cache physical address is interpreted as



No. of mm blocks are possible in one cache line. But not at same time

Number of lines

$\lceil \log_2 \#Cm\ Lines \rceil$

No of address blocks in a block

$\lceil \log_2 Block\ Size \rceil$

$$TAG = \frac{Mm\ Size}{Cm\ Size}$$

eg: TAG = 2 bit $\Rightarrow 2^2 = 4$ mm Block are mapped to one cache line

In direct mapping an mapping function is used to transfer (copying) the data from main memory to cache memory.

Mapping function: $K \bmod N = i$

K: Mm Block no./Mm request

i: Cache line number (0 to n-1)

N: Number of Cm Lines

$K \bmod N = i$ means 'K' main memory block are transferred into K mod N Cache memory line number.

Assume we have 4 Cache lines in Cm and mm block no 11 (B_{11}),

(i) Then mm block no 11 (B_{11}) are mapped into $11 \bmod 4 \rightarrow$ Cm line no '3'.

(ii) If mm block = 7 (B_7) then they are placed into $7 \bmod 4 \rightarrow$ Cm line no '3'.

i.e here block no B_{11} and Block no 7 are mapped into cache line number 3. But any one block present in cache line no 3.

eg: MM = 50 Blocks, Cm = 10 lines

$$TAG = \frac{MM}{Cm} = \frac{50}{10} \rightarrow 5:1$$

→ 5 mm Blocks are fighting / mapped to one Cache ~~eger~~ line. But out of 5 only any one mm block present in cache at a time

eg: MM = 16 Block, Cm = 4 lines

Block No (0, 4, 8, 12) —Mapped to→ Cache line '0'

Block No (1, 5, 9, 13) → Cache line '1'

Block No (2, 6, 10, 14) → Cache line '2'

Block No (3, 7, 11, 15) → Cache line '3'

Q10 Why before mapping we organise bits the memory?

eg: mm Size = 16 B, Cache Memory = 8 Byte, Block Size = 2 Byte.

Ans Before mapping:

Cache Mem: ← Cache = 3bit →

L0	W0
2bit	1bit

Main Memory: ← PA = 4bit →

TAG	W0
3bit	1bit

Here TAG bit tells the number of main memory blocks.

After Mapping: Cache controller interpreted the physical address as:

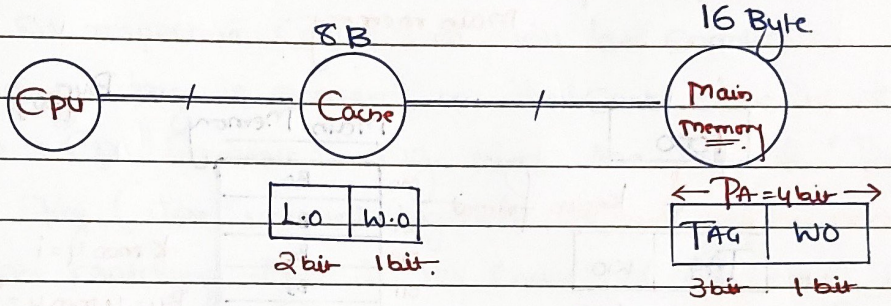
PA = 4 bit

TAG	L.O	W.O
1 bit	2 bit	1 bit

$TAG = \frac{mm\ Size}{Cm\ Size} = \frac{16B}{8B} = 2$

→ Here TAG bit is 1 → 2^1 → 2 mm Blocks are mapped for one cache line but only one mm block can be present in any one cache line {2:1}

→ This TAG bit is used to check (Identify) where there is Cache hit or Cache miss.



Direct Mapping:

PA = 4 bit

TAG	L.O	W.O
1 bit	2 bit	1 bit

Mapping function:

$K \bmod 4 = i$.

L.O = 2 bit

lines = $2^2 = 4$

Consider the following program:

I_1: MOV R0 [0000]

I_2: MOV R1 [1000]

I_3: ADD R0 R1

CPU Generated Request Refer to Cache

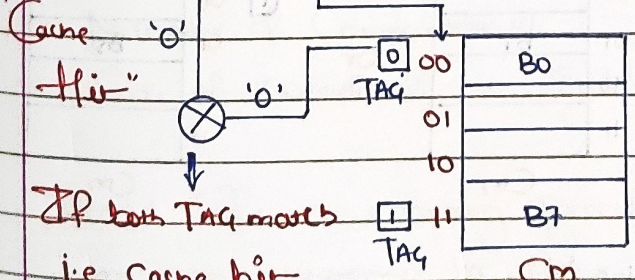
I_1: 0000

PA = 4 bit

TAG	LO	WO
1 bit	2 bit	1 bit

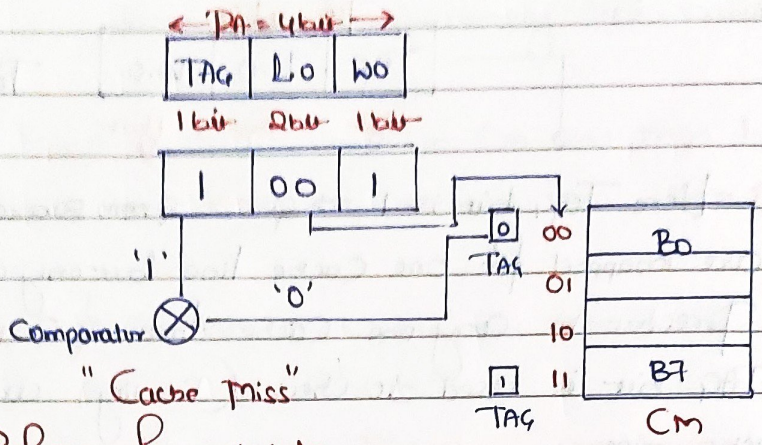
Direct mapped cache (known format)

TAG	LO	WO
0	00	0

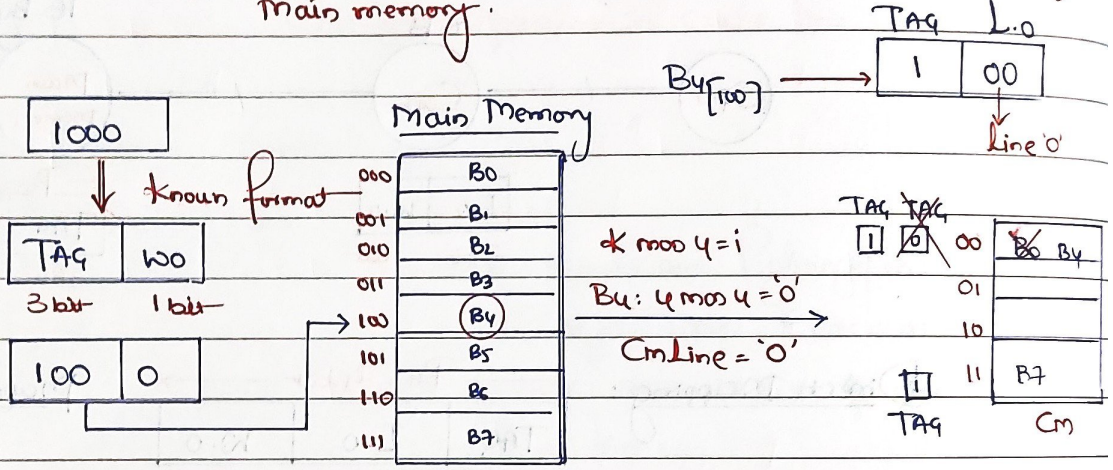


If both TAG matches i.e Cache hit then data is given from Cache to CPU directly.

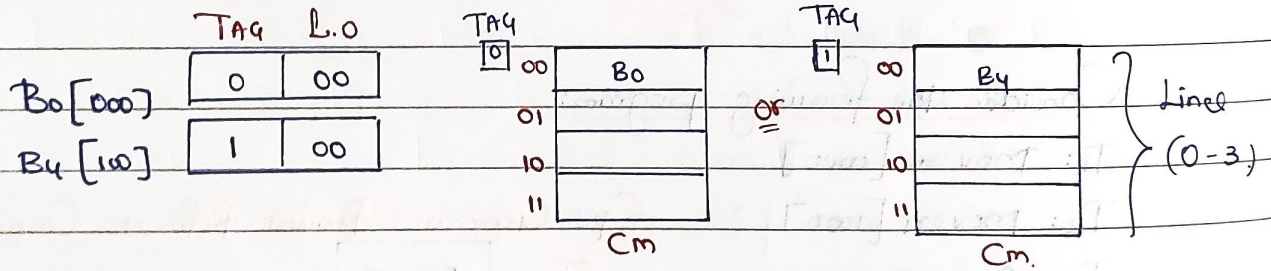
I_2 : 1000



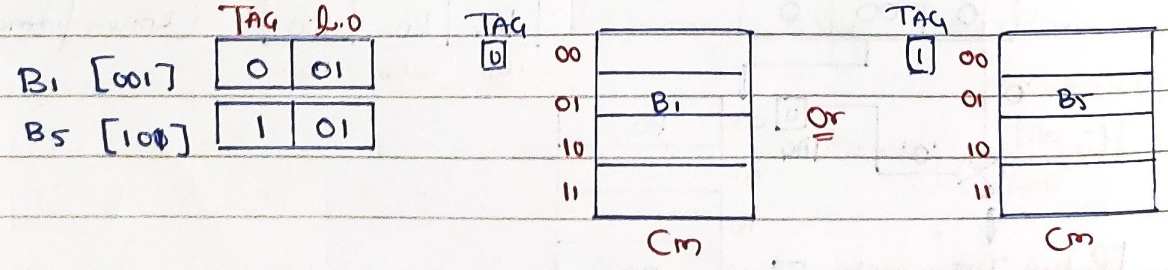
Reference forwarded to main memory.



(i) B0 and B4 mapped to Cm line '0'



(ii) B1 and B5 mapped to Cm line '1'.



(iii) B3 and B7 mapped to Cm Line '3'

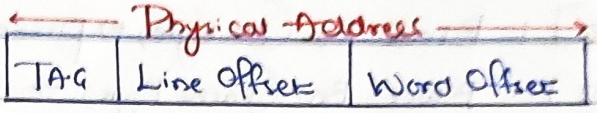
(iv) B2 and B6 mapped to Cm Line '2'

Page No.
Date 14/10/23
YOUVA

Tag Memory Size = #Lines × Tag bits { Depends upon the mapping technique }

Complete Working:

1. CPU Generated request firstly refer to the cache memory.
2. (Directly Mapped Cache) Cache controller interpreted the physical address as:



3. Line offset field is directly connected to the address logic of cache memory so respective cache line will be enabled.
4. The existing tag is enabled in the cache line is compared with the CPU Generated tag with the help of Tag comparator.
5. If both Tag (CPU Generated tag and existed tag on enabled cache line Tag') are matching then it is called Cache hit (i.e data is present in the cache). So based on word offset (0th Byte of block or 1st byte of block) respective data, respective word is transferred from cache to CPU.
6. If both Tag are not matching then that operation is called cache miss, then reference is forwarded to main memory.
7. According to mm format, respective mm block is enabled and transfer from MM to CM with the help of direct mapping function [K mod N] into particular cache line the cache to CPU.
8. Here Cache Controller maintain Some Extra bits for each Cache line and that stored in Tag memory / memory.

Tag memory = No. of CM lines × Tag bits

→ Number of Comparator = 1.

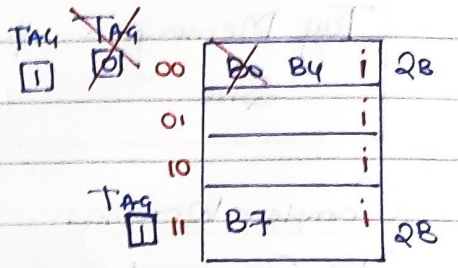
→ Size of comparator = Tag bits.

Note: * In a mapping process complete 1 mm block is transferred from main memory to Cache memory.

* But respective word / byte of that particular block which is demanded by the CPU is given by from Cache to CPU.

Consider the following Program:

I_1 : Mov [0000] 0th Byte of B_0
I_2 : Mov [0001] 1st Byte of B_0
I_3 : Mov [1000] 0th Byte of B_4
I_4 : Mov [1001] 1st Byte of B_4.



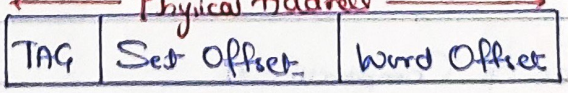
I_1 : [0000] : Block B_0 (0th Byte) ⇒ present in cache : Cache hit.
I_2 : [0001] : Block B_0 (Cache Byte) ⇒ present in cache. Due to locality of reference
I_3 : [1000] : B_4 ⇒ Cache miss so bring B_4 from main memory to Cache Memory
↳ then 0th Byte of given to CPU (mm to Cache [B_4] then cache to CPU 0th Byte of B_4)
I_4 : [1001] : By B_4 (1st byte of B_4) : Cache hit. Due to locality of reference.

Disadvantages of Direct mapping:

* In direct mapping each cache line is able to hold only 1 main memory block at a time so number of conflict miss increase.
* If CPU is refering the multiple blocks which are mapped into the same cache line number (eg: B_0, B_4, B_0, B_4 ⇒ Cm line No '0') then the number of conflict miss will be very high Even other cache line are empty. {Thrashing}.

Set Associative Mapping:

Set associative cache controller, interpreter the CPU generated request as follows:



→ Word Offset = $\log_2$ Block Size.
→ No. of Sets = $\frac{\#\text{ Lines}}{\text{N- way.}}$
→ Set offset = $\log_2$ Sets
→ TAG = physical address − (S.O + W.O)

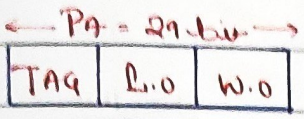
$\#\text{ Lines} = \frac{\text{Cm Size}}{\text{Block Size}}$

Q1. Consider a Direct Mapped Cache if the size of the cache memory is 512KB and main memory 512MB and Cache line is 64KB then, Calculate the number of bit required for:

(i) PA (iii) TAG (iv) L.O (iv) W.O,

(ii) # Lines

Ans.



$PA = 512MB = 2^{29}B \rightarrow PA = 29$ bit

$Line\ Size = 64KB = 2^{16}B \rightarrow WO = 16$ bit

$\#Lines = \frac{Cm\ Size}{Block\ Size} = \frac{512KB}{64KB} = \frac{2^{19}B}{2^{16}B}$

Tag memory Size = # Lines × Tag bit

= 8 × 10

= 80 bit

Lines = 8

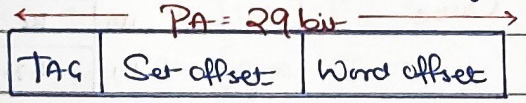
L.O = 3 bit.

TAG = PA − (L.O + W.O)

= 29 − (3 + 16)

= 10 bit

Above Question with 2-way Set associative mapping:



$\#Lines = \frac{Cm\ Size}{Block\ Size} = \frac{512KB}{64KB} = \frac{2^{19}}{2^{16}}$

$= 2^3 = 8$ lines

Tag memory Size = # Lines × Tag bits

= 8 × 11

= 88 bits

$\#SET = \frac{\#Lines}{N\text{-way}} = \frac{8}{2} = 4$

Set offset = 2 bit

Or

Tag memory Size = # Set × Block per set × Tag bits

= 4 × 2 × 11 bits

= 88 bits

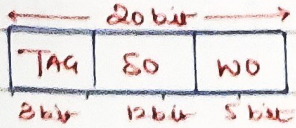
TAG = PA − (S.O + W.O)

= 29 − (2 + 16)

= 11 bit

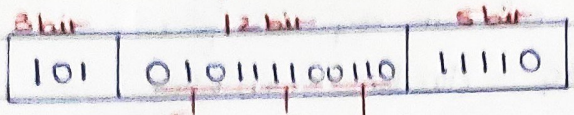
Q2. Consider a 2-way Set associative Cache Size = 256 ke, Line size = 32B, MM = 1MB, then what is the Set number of physical address $(ABCDE)_{16}$?

Ans.



$\#Lines = \frac{256KB}{32B} = \frac{2^{18}B}{2^5B} = 2^{13}$ lines

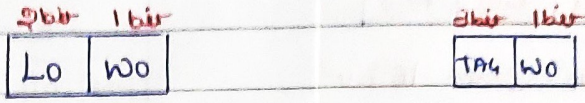
Set = $\frac{\#\text{Lines}}{\text{N-way}} = \frac{2^{14}}{2^2} = 2^{12}$. ∴ So = 12 bit

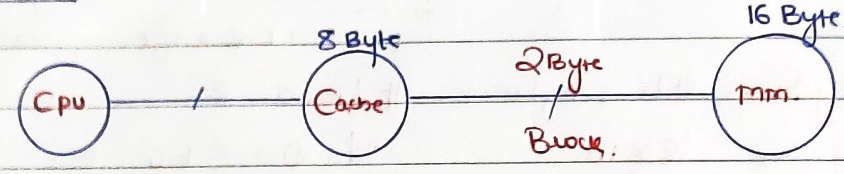


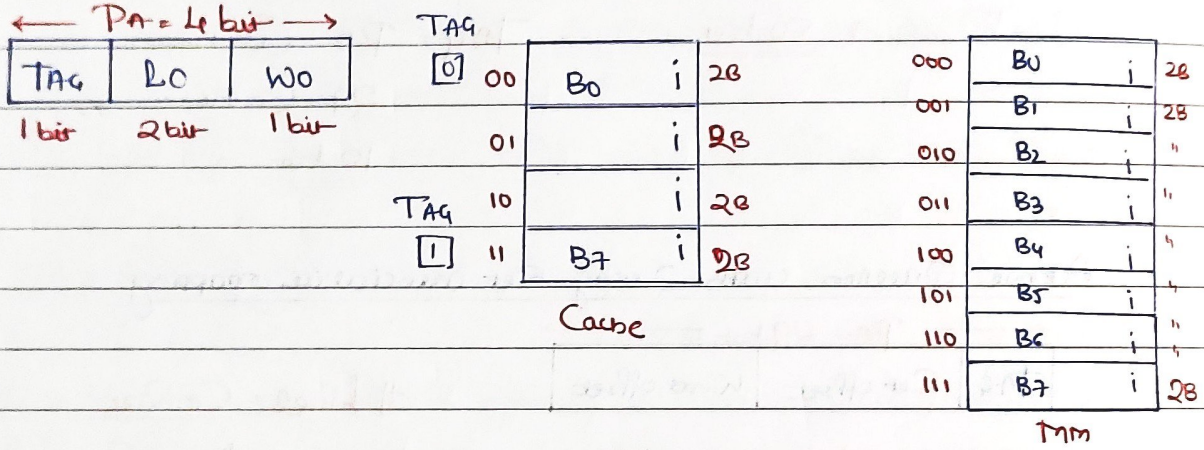
⟹ (5 3 6)16

Direct Mapping

Example.

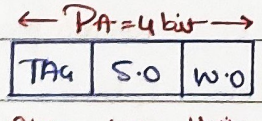






Q3 MM = 16 B, CM = 8B, Block Size = 2B, 2 way set associative mapping then Calculate bits for PA, TAG, S.O, W.O and TAG memory Size?

Ans



$\#\text{Line} = \frac{\text{CM Size}}{\text{Block Size}} = \frac{8B}{2B} = 4 \text{ Lines}$

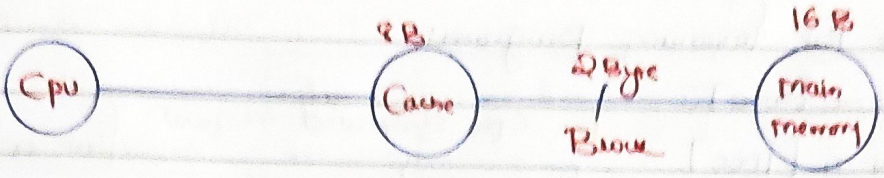
$\#\text{Set} = \frac{\#\text{Lines}}{\text{N-way}} = \frac{4}{2} = 2 \text{ set}$.

Tag memory = 4 × 2 Set offset = 1 bit.

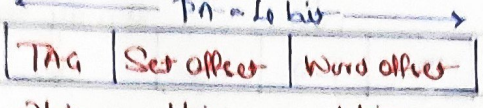
Size = 8 bit. (Ans)

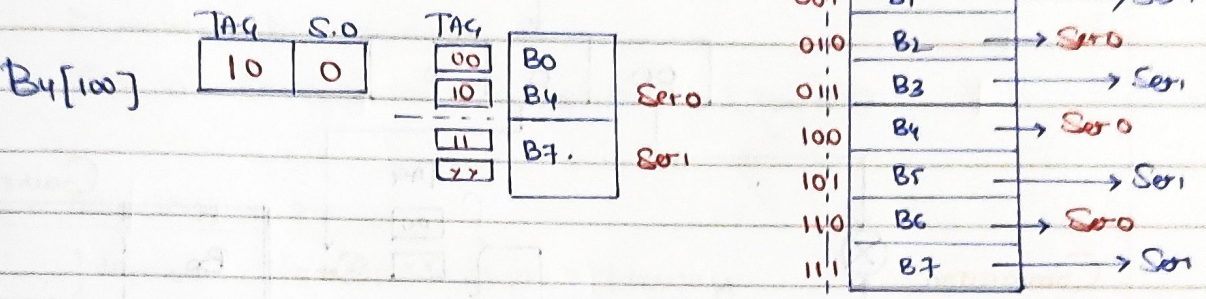
---

* Here Tag = 2 bit ie $2^2 \Rightarrow$ 4 mm blocks are fighting (mapped) to any one Cache set.



2 way Set associative mapping.





Advantage of Associative Mapping

I1 : Mov R0 [0000] 0th Byte of Block B0 I1 : Hit (Cache hit)
I2 : Mov R1 [1000] 0th " " B4 I2 : Miss (Cache miss)
I3 : Mov R2 [0001] 1st " " B0 I3 : Hit
I4 : Mov R3 [1001] 1st " " B4 I4 : Hit
I5 : Mov R4 [0000] 0th " " B0 I5 : Hit
I6 : Mov R5 [1000] 0th " " B4 I6 : Hit

6 Access : Total : 1 Miss
5 Hit

Note:

* 2 way Set associative : In one cache memory Set we can store a mm block in 2 ways [2mm Blocks].
* N-way set associative : In one Cm Set, we can store mm Block by N ways.

Set associative mapping function

Cache set = mm Request mod #set in cache address

or

K mod S = i

K : mm Block number i : cache set number
S : # cache set

Page No.:
Date: 16/10/24

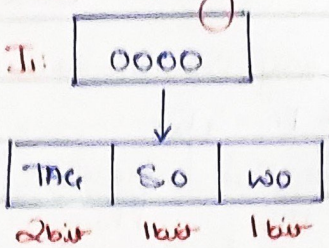
Consider the following program:

I_1 : mov R0 [1000]

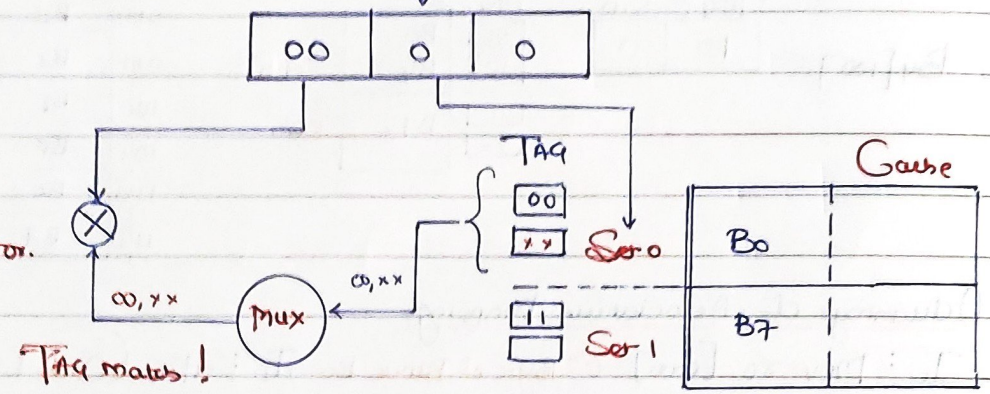
I_2 : mov R1 [1000]

I_3 : Add R0, R1

I_1: CPU generated request first refer to cache.

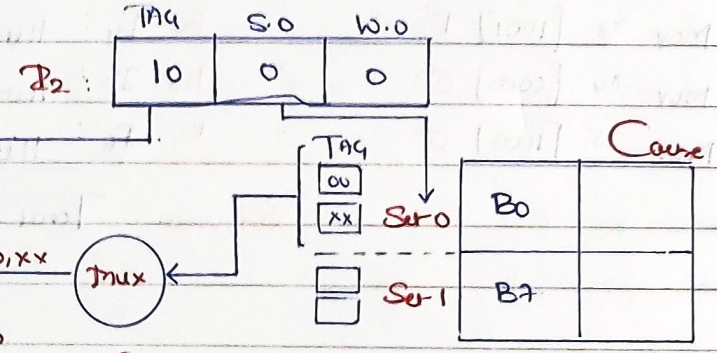


2 way set associative cache known format.

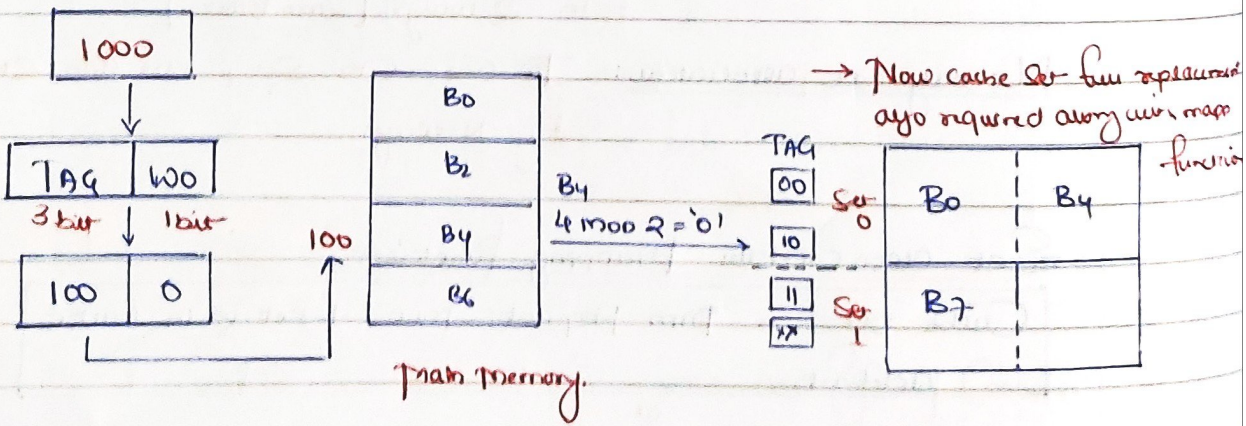


Tag matches!

Respective data cache bit given from cache to CPU.



No tag matches so cache miss, so reference forwarded to main memory.



→ In direct mapping:

Hit Latency = Latency of tag comparator.

→ In set associative mapping:

Hit Latency = Hit latency of tag comparator + latency of multiplexer.

Direct Mapping:

Number of comparator = 1

Size of comparator = # bits in Tag (Tag bits)

Set associative mapping:

Number of Comparator = N-way

Size of Comparator = # bits in Tag (Tag bits)

Important points about Set-Associative:

* # Line = L & N way Set associative # Sets [S] = $\frac{L}{N}$
* If N = 1 : Direct Mapping
* If N = L ; Associative mapping ; (S=1) i.e only 1 set
* Set associative mapping follows direct mapping as well as Set associative mapping. Using mapping function as:

K mod S = i

Within the Set MM Block Can be placed anywhere.

Note: In the Set associative mapping replacement algorithm is required along with the mapping function when cache Set is full.

Tag bits in N way set = Tag bits in direct mapping + $\log_2 N$ associative

eg: Consider a direct mapping if the Size of Cache memory is 512KB and main memory 512MB and Cache line size is 2KB then Calculate the number of bit required for TAG?

Ans.

PA = 29 bit

TAG	L.O	W.O
10 bit	8 bit	11 bit

$\#Lines = \frac{Cm\ Size}{Line} = \frac{512KB}{2KB} = 256$

L.O = 8 bit.

eg: Consider a 2-way Set associative mapping, if the Size of the Cache memory is 512KB and main memory 512MB and Cache line Size is 2KB then Calculate the number of bit required for TAG?

Ans.

PA = 29 bit

TAG	S.O	W.O
11 bit	7 bit	11 bit

$\#Lines = 2^8$

$\#Set = \frac{2^8}{2^1} = 2^7$

S.O = 7 bit.

29 - (7+11)

TAG = 11 bit

2 Way Set Associative

$Tag\ bit = Tag\ bit\ in\ direct\ map + \lceil \log_2 N\text{-way} \rceil$

$= 10 + \lceil \log_2 2 \rceil$

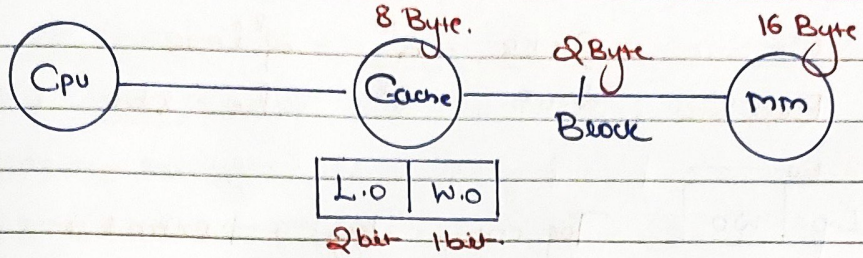
$= 11\ bits$

Associative Mapping:

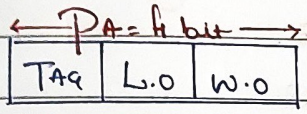
* In this no mapping function is used to transfer the data from MM to Cm.
* Any block of main memory can be placed anywhere in cache memory.
* No conflict miss.
* This Cache is designed without address called as Content addressable memory.
* In associative mapping more tag bits are required along with more memory Size.
* In associative mapping more hardware required, expensive (N tag comparator, here N is number of lines, for each comparator is required.)
* In associative Cache design, Counter Sequence is used to map the data, means any main memory block any Cache memory line in a Sequence
* In associative mapping / physical address is interpreted as

TAG	WO

eg:



Direct mapping



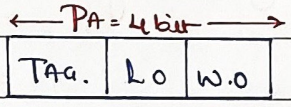
K mod N = i

K mod 4 = i

Tag memory = #line x Tag bit

⇒ 4 x 1 = 4 bits

Set-Associative mapping



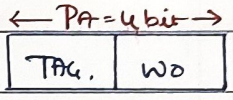
K mod 2 = i

K mod 2 = i

Tag memory = 4 x 2

= 8 bit

Associative / Fully associative

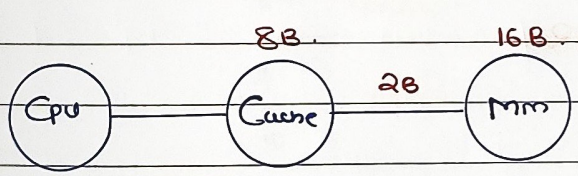


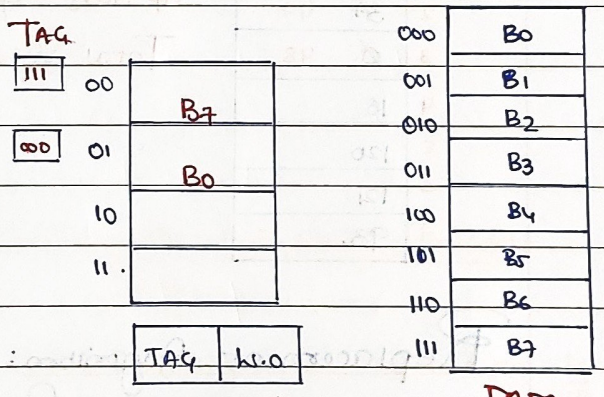
No mapping function

Tag memory = 4 x 3

= 12 bits

In associative mapping no mapping function is used to transfer the data from MM to CM





MM Block	Associative Mapping.	CM Line
B7 [111]	No mapping function	Any Line
B0 [000]	No mapping function	Any line

eg. Direct mapped Cache

Cache Size = 64 KB

Block Size = 64 word

Word length = 32 bits

MM = 4 Byte = 2^{32} Byte

Block Size = 64 Word

= 64 x 4 Byte

= 256 Byte

W.O = 8 Bit

#Lines = Cm Size / Block Size = 64 KB / 256 B = $2^{16} / 2^{8}$ = 2^{8} lines

L.O = 8 bits

PA = 32 bits

TAG	L.O	W.O
16 bits	8 bits	8 bits

Tag entry = Tag bits + extra bits

= 16 + 1 + 1

= 18 bits

= 32 - (8 + 8)

= 16 bits

Tag memory Size = # Lines × Tag bits

= $2^{8} \times 18$

= 256×18

= 4608 bits

Q1. Consider fully associative Cache Consist 8 Block and mm Contain 128 Block and request made by the CPU: 119, 84, 37, 0, 16, 0, 84, 120, 121, 93, 37, 0, 43, 39, 47, 48.
Calculate the # of compulsory and capacity miss?

Ans.

0	119 43
1	84 39
2	37 47
3	0 48
4	16
5	120
6	121
7	93.

Hit : 0, 84, 37, 0

Hits = 4

Total miss = 12.

Replacement Algorithm:

When the cache is full, the replacement algorithm are required to replace the exist cache block with new block.

In the Cm design 3 type of replacement algorithm is used.

1. Random Algorithm
2. FIFO Algorithm
3. LRU replacement.

In the random algorithm, any cache block can be replaced based on the random selection.

Types of Miss:

In the Cm design 3 types of misses are present:

1. Compulsory miss – (Cold Start miss / first reference miss)
. This miss will occur when the very first reference to the cache itself is a miss.
2. Capacity miss – This will occur only when the capacity is full.
3. Conflict miss – (Collision miss / reference miss)
This miss will occur when too many blocks are placed into same cache line or same cache set.

Q2 Consider 4 block cache memory (initially empty) with the following mm blocks referenced: 7, 8, 10, 15, 7, 8, 10, 7, 8, 10 Identify the hit ratio using:

Ans.

7
8
10
15
} Compulsory miss

mm request = 10
Cache line = 4.

7 → conflict
8 → conflict
15 → compulsory miss
7
8
10
} Conflict.

Direct mapped Cache:

Hit ratio = 3/10 = 0.3

2 Way Set Associative with LRU

Hit = 4

Mm request = 10 Hit Ratio = $\frac{4}{10}$ = 0.4

Q3. Consider a small two-way set associative cache memory, consisting of 4 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 address is 8, 12, 0, 12, 8.

Ans. # Cm Lines = 4 2 way Set associative

Set = 4/2 = 2 K mod 2 = i

8 mod 2 = 0 → M

12 mod 2 = 0 → M

0 / 2 = 0 → M (Cache set is full, apply LRU replacement algo.)

12 / 2 = 0 → Hit

8 / 2 = 0 → miss (apply LRU)

Set 0	8 ✗ 8	
	12	
Set 1		

∴ Cache misses = 4

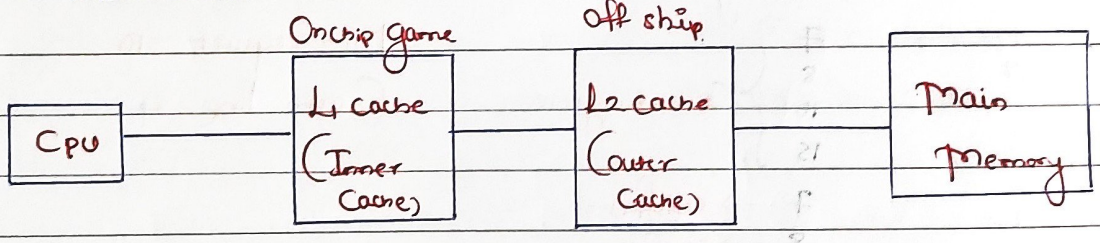
How to reduce all the miss:

1. Compulsory miss: We can reduce the compulsory miss by increasing the block size (large block size).
2. Capacity miss: Capacity miss occur when cache is full. If we increase the cache size then capacity miss can be reduced.
3. Confict Miss: Confict misc can be reduced by increasing the associativity.

Multilevel Cache:

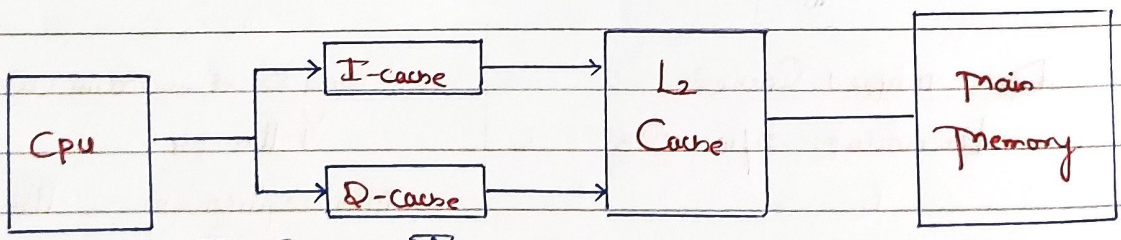
2 Level Cache:
- → L1 cache
- → L2 cache.

L1 Cache → I cache
L1 Cache → D cache



Size L2 > L1

Speed L1 > L2.



I-Cache: Instruction Cycle

D-Cache: Data Cache.

Working Principal of multilevel Cache:

1. Principal of Inclusion: Data present in L1 Cache must be present in L2 Cache.
2. Principal of Exclusion: Data present in L1 Cache must be different from L2 cache.

Why Multilevel Cache is used:

level 1 Cache access = 10nsec time

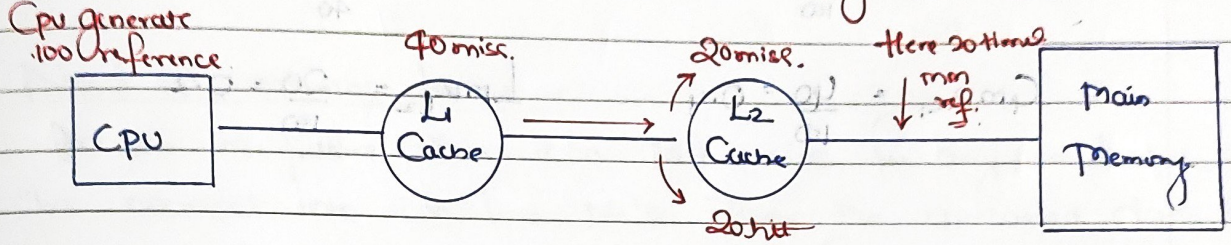
Main memory access time = 300ns.

L_1 Cache access time = 10 nsec

L_2 cache access time = 15 nsec

mm access time = 300 ns

So Tag will reduce



If only L_1 Cache then if there is miss in level 1 Cache then directly refer to main memory

(Here higher time ~~use~~ main memory access so Tag increase).

But in 2 level Cache, if miss in level 1 Cache then that miss forwarded to level 2 cache (so some hit in level 2) then after level 2 there is a miss then we go to main memory.

So Tag will be reduced (because L_1 and L_2 Cache access time very low).

Multi Level Cache

* To reduce the miss penalty multi level Caches are used in the System design.
* The number of cycles ~~reduced~~ required to transfer the data from higher levels to L_1 due to miss operations is called as miss penalty

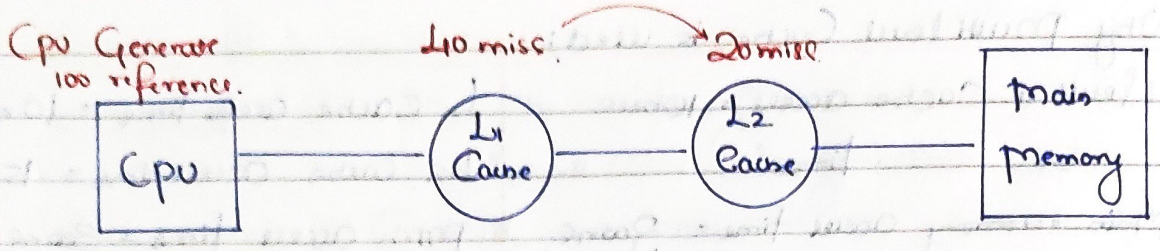
Miss Rate

Local Miss Rate (LMR)

$$LMR = \frac{\#\text{misses in cache}}{\#\text{access to that cache}}$$

Global Miss Rate (GMR)

$$GMR = \frac{\#\text{misses in the Cache}}{\text{Total \# CPU generated reference.}}$$



$LMR_{L1} = \frac{40}{100} = 0.4$ $\quad LMR_{L2} = \frac{20}{40} = 0.5$

$GMR_{L1} = \frac{40}{100} = 0.4$ $\quad LMR_{L2} = \frac{20}{100} = 0.2$

Access time in Multilevel Cache:

Level 1:

$$T_{avg} = h t_1 + (1-h)(t_m + t_1)$$

or

$$T_{avg} = t_1 + (1-h)(t_m)$$

Level 2:

$$T_{avg} = h_1 t_1 + (1-h_1) h_2 (t_2 + t_1) + (1-h_1)(1-h_2)(t_m + t_2 + t_1)$$

or

$$T_{avg} = t_1 + (1-h_1)\{t_2 + (1-h_2) t_m\}$$

t_1 : hit time in level 1
$(1-h_1)$: miss rate of level 1.
t_2 : hit time of level 2.
$(1-h_2)$: miss rate of level 2
t_m : access time

Updating Technique:

* CPU perform read or write operations on cache memory.
* When the data (mm block) is available in cache then it is known as read hit and write hit.
* If the data is not available in the cache then it is called read miss and write miss then we have to perform Read allocate and write allocate.

Read Allocate: When there is a read miss in the cache then transferring that data from main memory to cache memory is

Called read allocate.

Write Allocate: When there is a write miss in the cache then transferring that data (mm block) from main memory to cache memory is called write allocate.

Need of updating technique:

Assume in future there is a cache miss for any mm block, which are mapped into cm line No '0'. Then the updated data (of block B0) is replaced by new block. (All the updation get lost).

That means whatever work done by the CPU and load into the cache get lost. Because, that block is replaced by the new main memory block.

Now in future if we bring Block B0 then we get the old value.

Updating Techniques:

1. **Write Through:** In the write through both cache memory and main memory update simultaneously so no problem of co-herence (No inconsistency)

$$t_w\left[\begin{matrix}\text{word updation}\\ \text{time}\end{matrix}\right] = \max\left\{\begin{matrix}\text{word updation}\\ \text{time in cache,}\end{matrix}\ \begin{matrix}\text{word updation}\\ \text{time in MM}\end{matrix}\right\}$$

$$T_{avg\,read} = h_r t_c + (1-h_r)(t_m + t_c)$$

h_r → Read hit; t_c → Read data; $(1-h_r)$ → Read miss; t_m → Read allocate; t_c → Read data from cm.

$$T_{avg\,write} = h_w t_w + (1-h_r)(t_m + t_w)$$

h_w → write hit; t_w → word updation; $(1-h_r)$ → write miss; t_m → write allocate; t_w → word updation.

$$T_{avg\,WT} = \%\text{ of read operation} \times T_{avg\,read} + \%\text{ of write operation} \times T_{avg\,write}.$$

$$\eta_{WT} = \frac{1}{T_{avg\,WT}}$$

Disadvantage in WT: If there is a frequent updation in cache, then WT give worst performance because every updation we have to update mm and Cm block.

Simultaenous Access:

$$T_{avg\,read} = h_r t_c + (1-h_r) t_m \qquad T_{avg\,write} = t_w$$

(2.) Write Block Update Technique: In the WB updating technique main memory update later, only cache memory update. (So in WB Cache memory update and mm update later so coherence is present).

But this coherence does not create any problem

* Because in WB, each cache line maintain some extra bit (like valid/invalid, dirty bit (modified bit)) to maintain the status of Cm Block.
* If the Cm Block is updated then modified bit (dirty bit) is set to 1. If block is not updated then clear bit is set (dirty bit = 0).
* Before replacing the Cm Block we check the status of the block.
* If a block is a updated block (modified bit set) then firstly this updated data write block into main memory (higher memory) then replace the Cm block with any new (request) block.
* If block is not updated block then directly replace with new main memory (requested block).

$$T_{avg_{read}} = hr \cdot tc + (1-hr)\{\%\ data\ modified\ (tx + tm + tc) + \%\ clean\ bit\ (tm + tc)\}$$

$$T_{avg_{write}} = hw \cdot tc + (1-hw)\{\%\ data\ modified\ (ty + tm + tc) + \%\ clean\ bit\ (tm + tc)\}$$

$$T_{avg_{WB}} = Fr \times T_{avg_{read}} + Fw \times T_{avg_{write}}$$

$$\eta_{WB} = \frac{1}{T_{avg_{WB}}}$$

Fr = % of read operations

Fw = Frequency (%) of write operations

Q1. Let WB and WT be two set associative cache organisations that use LRU algorithm for cache block replacement. WB is a write back cache and WT is a write through cache. Which of the following statements is/are true/false?

Ans. A. Each cache block in WB and WT has a dirty bit.

B. Every write hit in WB leads to data transfer from cache to main memory.

C. Eviction of a block from WT will not lead to data transfer from cache to main memory.

D. A read miss in WB will never lead to eviction of a dirty block from WB.

While Cache Accessing:

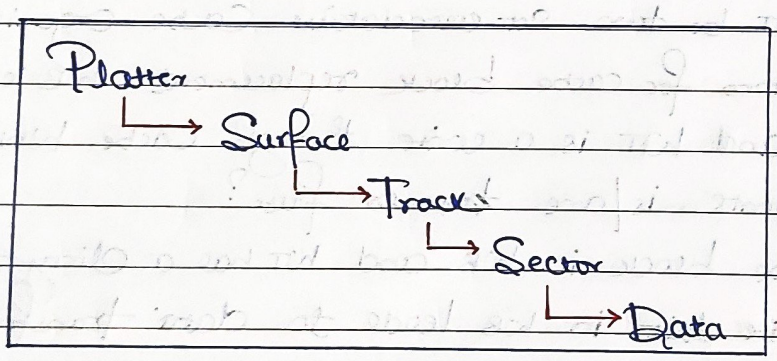
1. Physical Index Cache : PM address is used in address mapping.
2. Virtual Index Cache : VM address is used in address mapping.

(VIVT)

↳ Virtual Index Virtual Tag.

SECONDARY MEMORY & IO INTERFACE

* Magnetic disks provide the bulk of secondary storage for modern computer systems. Conceptually disks are relatively simple.
* Each disk platter has a flat circular shape, like a CD. Common platter diameters range from 1.85 to 3.5 inches.
* The two surfaces of a platter are covered with magnetic heads are attached to disk arm that moves all the heads as a unit.
* The surface of a platter logically divided into circular tracks, which are subdivided into sectors.
* The set of tracks that are at one arm position makes up a cylinder. There maybe thousands of concentric cylinders in a disk drive, and each track may contain hundreds of sectors. The storage capacity of common disk drives is measured in disk drives.



Moving-Head Disk Mechanism

* Each platter has 2 surface
* Read/write head with each surface to read/write data.
* These read/write head all connected with the arm.
* All the these arms are connected to arm assembly and rotate at the same time.

eg: If there are 20 surfaces then:

Read/write head : 20
Arm : 20
Arm Assembly : 1

Note: All read/write head (Arms) rotating/moving at the same time but only 1 R/W head performing read/write operations.

Q1. Consider a disk which has 16 platter, each platter has two surface. Every surface has 1K Track is further divided into 512 Sector and every Sector can store the 8KB data, then calculate.

(i) What is the capacity of the disk?

Ans 16 platter, each platter has 2 surface

Total number of Surface = 32 [16×2] Surface

Tracks per Surface = 1K, total = 32K tracks

Sectors/track = 512

Total # sectors = 32K × 512 Sector

Each Sector Capacity = 8KB

Disk Capacity = 32K × 512 × 8KB

= 128 GB

(ii) How many bits are required to identify any particular sector of disk?

Ans # bits to identify Particular Sector = 16 × 2 × 1K × 512

$= 2^4 \times 2^1 \times 2^{10} \times 2^9$

$= 2^{24}$

= 24 bit

Q2. 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

Ans Disk capacity = 16 × 128 × 256 × 512 Byte

$= 2^4 \times 2^7 \times 2^8 \times 2^9$ Byte

$= 2^{28}$ Byte

= 256 MB

bits required to identify particular Sector = 16 × 128 × 256

$= 2^4 \times 2^7 \times 2^8$

$= 2^{19}$

= 19 bit

13/10/23

Disk I/O Operation:

1. **Seek time**: The amount of time taken to move the read/write header from its current position to the desired track is called as Seek time.
2. **Rotating Latency**: The amount of time taken to rotate the track when the read/write header comes exact to exact position.
   Rotational Latency is considered as $\frac{1}{2}$ Rotation Time
3. **Transfer time**: The amount of time taken to transfer the required data
4. **Transfer rate**: The number of bytes found for unique line is called as transfer rate of disk.

* The read/write header can never be outside the track it will be pointing to any particular track of the surface.
* Read/write header can move in forward and backward direction and disk in one direction (either clockwise or anticlockwise)

eg: 600 Rpm. (600 rotation per minutes)

$$1 \text{ Rotation} = \frac{60}{600} = \frac{1}{10} \text{ sec} \times \frac{1000}{1000} = 100 \times 10^{-3} \text{ sec}$$

$$= 100 \text{ msec}$$

1 Rotation time = $\frac{1}{10}$ sec or 100 msec.

$$\text{Avg rotational latency} = \frac{1}{2} \times 100$$

$$= 50 \text{ msec}$$

eg: Assume 1 Rotation time 1/6 sec, 1 Track Capacity = 4KB then Calculate data transfer time for x Byte and data transfer rate?

(i) **Data Transfer time**:

4KB ⟶ 1/6 sec

1 Byte ⟶ 1/6×4K sec

x Byte ⟶ x Byte / 6×4KB sec

(ii) Data Transfer Rate: 4KB. → 1/6 sec

1/6 sec → 4KB

1 sec → 4KB × 6 sec

= 24 KBps

Disk Access Time = S.T + R.L + D.T.T + Overhead (if any). (D.A.T)

Q1. Consider a disk system, which has an average seek time of 30ms and rotational rate of the disk is 360 Rpm. Each track of the disk has 512 sector, each of the size 512 Byte, then calculate.

(i) Average Seek time

(ii) Average rotational latency.

(iii) Data transfer time for 4 sector (Continous)?

(iv) Data transfer rate?

Ans 360 Rpm

1 Rotation = $\frac{60}{360} = \frac{1}{6}$ sec.

Avg. R.L = $\frac{1}{2} \times \frac{1}{6} = \frac{1}{12}$ sec

= 0.083 sec

Avg Seek time = 30 msec

1 Track Capacity = # sector/track × each sector Capacity

= 512 × 512 Byte = $2^9 \times 2^9 B = 2^{18} B$

= 256 KB

Data Transfer Amount = 4 sector → 4 × 512B

= $2^2 \times 2^9 B$

= 2KB

Data Transfer Time:

256 KB ——— 1/6 sec

2 KB ——— $\frac{2 KB}{6 \times 256 KB}$ sec

2KB = $\frac{2KB}{6 \times 256KB}$ sec ⇒ $\frac{1}{768}$ sec.

Data Transfer Rate:

256 KB → 1/6 sec

1/6 sec → 256 KB

In 1 Sec → 6 × 256 KBps

= 1536 KBps

Fixed Sector Capacity: Each Sector (innermost and outermost sector) has fixed storage. (Storage density is variable)

Variable Sector Capacity: Each sector has variable Storage. Innermost and outermost Sector Capacity is variable. (Storage density is fixed).

* In Todays time we generally use "fixed Sector Capacity".

Need of Cylinder:

Assume that we have a very large file size (in GB) and 1 track size is 32 MB.

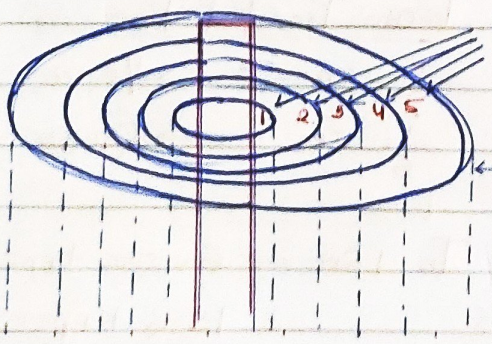
If we store a file on a disk then we can store 32 MB on one track then we have to go next track of the same surface so on storing data. So we have to suffer unnecessary very high Seek latency (time).

So we store the data cylinder wise as we know each surface r/w head point to same address at a time.

So first we store { track '0' of surface '0', track '0' of surface '1', ..., track '0' of surface 2000 } 1 Seek time (latency)

With the help of cylinder we can store 2000 Track data in one Seek time.

* Same track number in all the surface will form a cylinder



Number of platter = 2
Number of Surface = 4
Number of tracks per Surface = 5
Number of cylinder = 5 (0-4)
Number of Sector per track = 8
Number of Sector per Cylinder = 4 × 8 = 32


8 Sectors per track

In this example each track contain 8 Sector. This is for 1st cylinder (Cylinder-0).
First we traversed (cross) all sector of track 0 (1st track) of surface 0
Then we traverse (cross) " " " " 0 of Surface 1
" " " " " " 0 " " 2
" " " " " " " " 3

I Cylinder : Cylinder Number 0

Step 1 : Surface number 0 (I st surface) track no '0' (1st track) all 8 Sector are traversed (crossed).

Step 2 : Surface number 1 (II Surface) track no 0 (1st track) all 8 Sector are traversed (crossed).

Step 3 : Surface number 2 (III Surface) track no 0 (1st track) all 8 Sector are traversed (crossed).

Step 4 : Surface number 3 (4th Surface) track no 0 (1st track) all 8 Sectors are traversed (crossed).

Now move to next Cylinder and so on.

I Cylinder : Cylinder Number 0

<C, h, S>

0th Surface {
<0, 0, 0> 0th Sector
<0, 0, 1> 1st sector
⋮
<0, 0, 7> 7th Sector.

1st Surface {
<0, 1, 0> 8th Sector
And so on upto 32nd Sector.

Sector/track : 8
Surface : 4
track per Surface = 5
Cylinder = 5 (0-4)
Surface = 4 (0-3)

* All 32 sector Stored in 4 Surface of cylinder no 0.
* Now we move to the next cylinder 2nd cylinder (cylinder number 1). And so on to all cylinders.
* In this manner we Store total 160 ~~Store~~ Sector (which Stored in (32 sector per cylinder x 5 cylinder).

eg: What is the sector no. of <4, 3, 7>

(i) To Cross : 4 cylinder (0,1,2,3) = 4x4x8 = 128

(ii) To Cross 3 Surface = 3x8 = 24

(iii) To Cross 7th Surface = 7

159 (Ans)

Q1. 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.

=# Surface = 10x2 = 20 Surface

=# Sector/track = 63

Sector per cylinder = 20x63

= 1260 Sector/cylinder.

(i) The address <400, 16, 29> Corresponds to Sector number:

<400, 16, 19>

(i) To Cross 400 cylinder = 400x20x63 = 504,00

(ii) To Cross 16 Surface = 16x63 = 1008

(iii) To Cross 29 Sector = 29

505037

Alternate method

Sector Number = S + ST*h + ST*TC*c

ST : # Sector Per track
TC : # Track per Cylinder
SC : # Sector Per Cylinder
↳ ST * TC

Sector Number = 29 + 63×16 + 63×20×400
= 505037 (Ans)

(ii) The address of 1039th Sector is :

<0 + 15×63 + 31> = 976
<0 + 16×63 + 30> = 1038
<0 + 16×63 + 31> = 1039
<0 + 17×63 + 31> = 1102

∴ <0, 16, 31> (Ans)

$C = \left\lfloor \frac{\text{Given Sector No}}{\text{Total \# Sector/cylinder}} \right\rfloor = \left\lfloor \frac{1039}{1260} \right\rfloor = 0$ (1st cylinder)

$h = (1039 \bmod 1260)/63 = \left\lfloor \frac{1039}{63} \right\rfloor$

= 16 Surface

How much Sector remaining

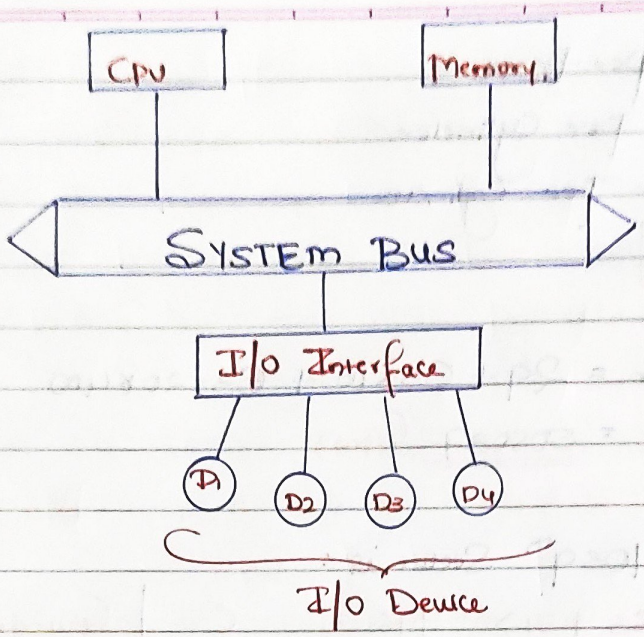
= 1039 − 16×63
= 1039 − 1008
= 31

Note: <C, h, s>

C (Cylinder Number) $= \left\lfloor \frac{\text{Given Sector}}{\text{\# Sector/cylinder}} \right\rfloor$

h (Surface Number) $= \left\lfloor \frac{(\text{Given Sector} \% \text{ \# Sector per cylinder})}{\text{Total Sector per track}} \right\rfloor$

Sector Number $y - x * \text{\#Sector}$



Input - Output Interface

Input-output interface provides a method for transferring information between internal storage and external I/O devices. Peripherals connected to a computer needs special communication links for interfacing them with the central processing unit. The purpose of the communication link is to resolve the differences that exist between the central computer and each peripheral. The major differences are:

1. Peripherals are electromechanical and electromagnetic devices and their manner of operation is different from the operation of the CPU and memory, which are electronic devices. Therefore, a conversion of signal value maybe required.
2. The data transfer rate of peripherals is usually slower than the transfer rate of the CPU and consequently, a synchronization mechanism maybe needed.
3. Data codes and formats in peripheral differ from the word format in the CPU and memory.
4. The operating modes of peripherals are different from each other and each must be controlled so as not to disturb the operations of other peripherals connected to the CPU

I/O Organisation

1. I/O devices are electro-magnetic components and CPU is a electronic component. So, there is a difference that exist in the term of operating modes, data transfer rate and word formats.
2. To synchronise the I/O speed with a CPU, high speed interface chip is used named as I/O interface or I/O module.
3. I/O interface chip is responsible for I/O operations so, the computer designer design I/O devices are connected to system bus via I/O interface chip.

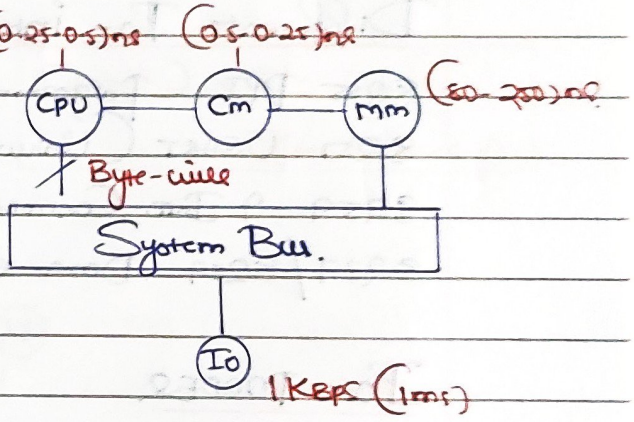
System without IO-interface (Programmed-IO).

1 KB - 1 Sec

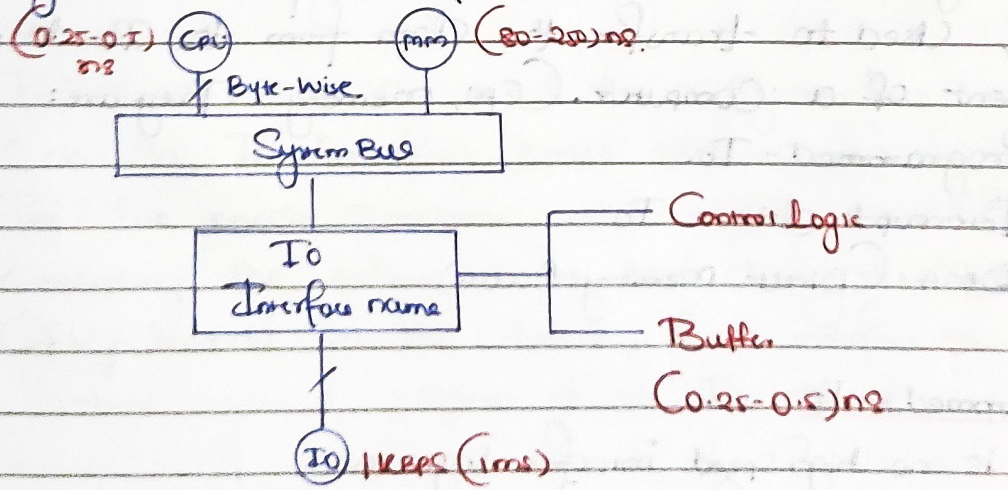
1 B - ?

$ETIO = \frac{1\,B}{1\,KB}\ sec = 10^{-3}\ sec$

$= 1\ ms.$



System with IO-Interface (INT-Driven IO).



Access Sequence / Working Process

1. CPU initializes the I/O interface chip along with a I/O command and then CPU will go and perform other useful task.
2. IO-interface control logic interprets the IO-Commands and

accordingly Io port will be enabled for the operation.

③ Based on the speed of a Io device and amount of data to be transfer consume the time to prepare the data, then data is transferred from Io device to a interface buffer.

④ When the data is available in the buffer Io interface generate the interrupt signals and send to the CPU and waiting for ack signal.

⑤ After recieving the ack signal, buffer content will be transferred to CPU. In this process, CPU will be accessing the Io- data from interface buffer therefore speed gap is synchronized & time is saved.

Different Io- interface chip used in the computer designing:
8255 PPI (Programmable Peripheral Interface)
8251 USART (Universal Sync. asynchronous reciever transmitter).
8259 A Int. Controller
8237/8257 DMA etc.

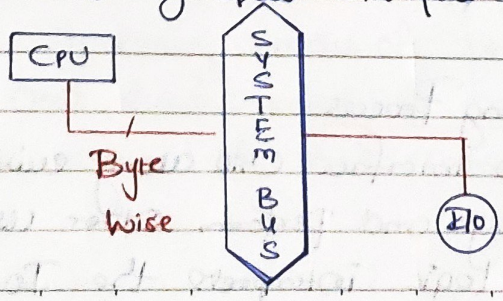
Io- MODES

Three types of Io- transfer modes are present in the computer system, used to transfer the data from the Io to other component of a computer. (CPU, memory). They are:

1. Programmed - Io.
2. Interrupt driven Io
3. DMA (Direct Memory Access).

① Programmed - Io

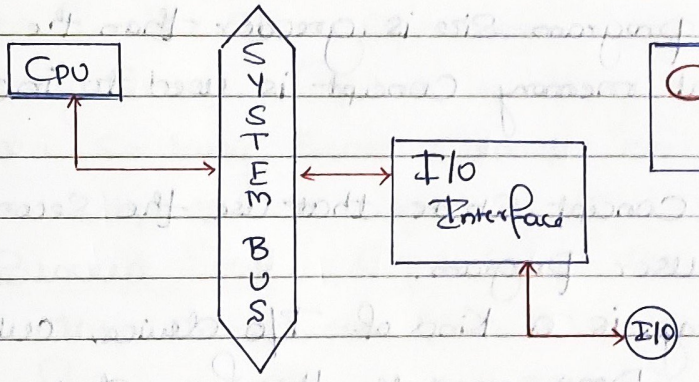
There is no high speed interface logic used



CPU Time (Data transfer Time) = Speed of I/O Device

- In this mode, Io- devices are directly connected to CPU without Io- interface chip.
- In this mode, CPU takes the responsibility to complete the Io operation, so CPU will be blocked (waiting) until the Io- Operation is completed.
- In this mode CPU utilization is inefficient.
  CPU time depends on the speed of a Io- device and the size of a data unit to be transferred
- This mode is suitable in the system centric application where the Io time is important than CPU time.

(2) Interrupt Driven Io



CPU Time = I/O Interface Time Latency

- In this mode, Io- operations are controlled based on the interrupt signals.
- Io- devices are connected to a system bus via Io-interface chip so, Io- interface takes the responsibility of a Io open.
- In this mode processor utilization is more efficient so CPU executing the other useful task during the Io open.
- CPU time depends on the latency of a interface chip rather than speed of an Io device.

Drawback: More complexity in interrupt implementation.
Solution: Use priority sense highest priority interrupt.
Who will assign:
1. Daisy Chain Method (Static Approach): Fixed: Starvation occur
2. Polling Method (Dynamic Approach): Changing: No Starvation.

Page No.
Date 27/10/23

① Programmed I/O
② Interrupt Driven I/O

* In these two we cannot access the memory directly (we have to access memory with the involvement of CPU.
* These are used for small amount of data transfer.

③ Direct Memory Access:

* In this mode, bulk amount of data will be transferred from the IO to main memory without involvement of a CPU. Then Processor and DMA Controller use the System bus in Master Slave mode.
* When the user program size is greater than the main memory size than virtual memory concept is used to increase the address space.
* Virtual memory concept state that use the secondary memory to store the user program.
* Secondary memory is a kind of I/O device, which is a interfaced to a DMA module, therefore during the program execution data will be transferred from I/O to main memory via DMA without involvement of a CPU.

Access Sequence (DMA)

1. CPU initialized the DMA module along with a I/O command later busy with other useful task.
2. I/O Command Contains the information about Port address, memory address, Control Signal and Count value.
3. DMA module Control logic interprets the command and enables the respective port for the Operation.
4. Based on the Speed of a device consumed the time to Prepare the data later enables the "DMA-REQ" signal.
5. After recieving this signal, DMA module enables the hold Signal to CPU to gain the control of a System bus and