1.2.6 Cache Memory: GATE CSE 1995 | Question: 1.6

The principle of locality justifies the use of:

A. Interrupts
B. DMA
C. Polling
D. Cache Memory

1.2.7 Cache Memory: GATE CSE 1995 | Question: 2.25

https://gateoverflow.in/2638

A computer system has a 4 K word cache organized in block-set-associative manner with 4 blocks per set, 64 words per block. The number of bits in the SET and WORD fields of the main memory address format is:

A. 15, 40
B. 6, 4
C. 7, 2
D. 4, 6

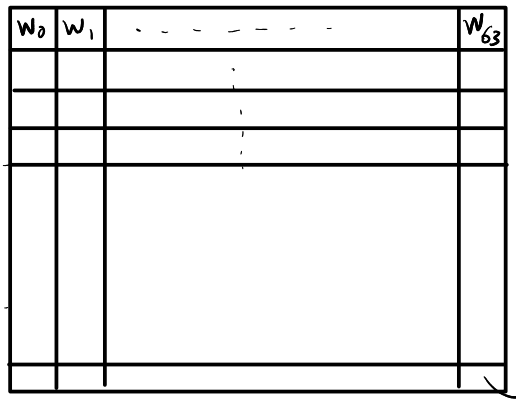
4 way = 4 lines in a set

words = 64

word bits = $\log_2$(# words) = $\log_2(64) = 6$ (enough)

4K cache size => 4×2^{10} words = 2^{12} words

Sets = $\frac{\text{\# lines}}{\text{\# lines per set}}$ = $\frac{2^6}{4}$ = 2^4



total # lines = $\frac{\text{total \# words}}{\text{\# words per line}}$ = $\frac{2^{12}}{64} = \frac{2^{12}}{2^6} = 2^6$

bits required for set = $\log_2$(# sets) = $\log_2(2^4) = 4$

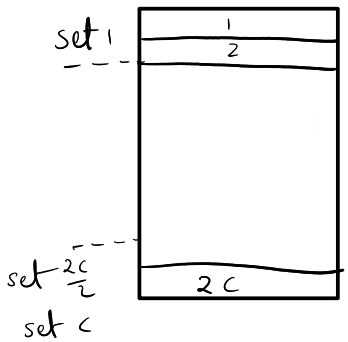
1.2.10 Cache Memory: GATE CSE 1999 | Question: 1.22

https://gateoverflow.in/1475

The main memory of a computer has 2 cm blocks while the cache has 2 c blocks. If the cache uses the set associative mapping scheme with 2 blocks per set, then block k of the main memory maps to the set:

A. $(k \mod m)$ of the cache
B. $(k \mod c)$ of the cache
C. $(k \mod 2c)$ of the cache
D. $(k \mod 2cm)$ of the cache

2c lines



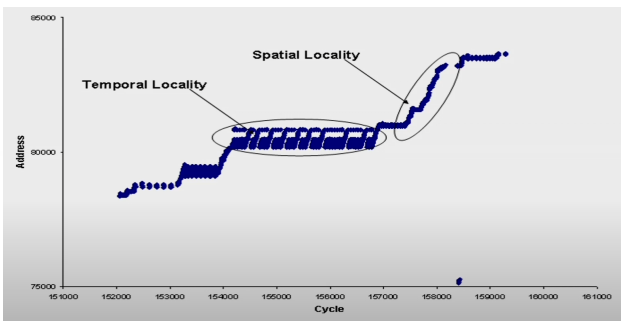
so k must go into one of c sets

i.e k % c

1.2.11 Cache Memory: GATE CSE 2001 | Question: 1.7, ISRO2008-18

More than one word are put in one cache block to:

A. exploit the temporal locality of reference in a program
B. exploit the spatial locality of reference in a program
C. reduce the miss penalty
D. none of the above



https://gateoverflow.in/750

A CPU has $32-bit$ memory address and a $256\ KB$ cache memory. The cache is organized as a $4-way$ set associative cache with cache block size of 16 bytes.

each line can hold # words $\rightarrow 2^4$ B

each set has 4 lines

by default assume that a word in memory is byte addressable

$\rightarrow 2^8 \times 2^{10}$ B $= 2^{18}$ B $\equiv$ cache size is 2^{18} words

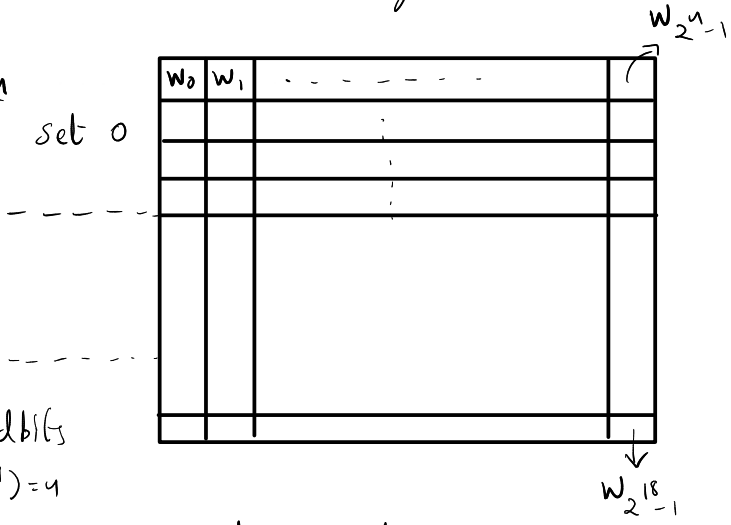
A. What is the number of sets in the cache? 2^{12}

B. What is the size (in bits) of the tag field per cache block? 16 bits

C. What is the number and size of comparators required for tag matching? 4 & 16

D. How many address bits are required to find the byte offset within a cache block? 4

E. What is the total amount of extra memory (in bytes) required for the tag bits? 2^{15}



A

$$\frac{\#\text{ words}}{\#\text{ words per line}} \Big/ \#\text{ lines per set} = \left(\frac{2^{18}}{2^4}\right)\Big/ 4 = \frac{2^{14}}{2^2} = 2^{12} = \#\text{ sets}$$

B

# tag bits	# set bits	# word bits
32 − # set bit − # word bit	$\log_2(2^{10}) = 10$	$\log_2(2^4) = 4$
$32-12-4 = 16$ → tag bits		

D. byte $\equiv$ word (in byte addressable memory)

each line in set has 16 words

bits needed to locate a word in line $= \log_2(16) = 4$

C. purpose of tag/line comparator : to identify if CPU requested memory is present in cache or not

some other example for understanding purpose

16 words $\Rightarrow \log_2(16) = 4$ bits

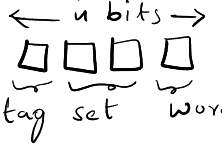
address	word
0000 0	a
0001 1	b
0010 2	c
0011 3	d
0100 4	e
0101 5	f
0110 6	g
0111 7	h
1000 8	i
1001 9	j
1010 10	k
1011 11	l
1100 12	m
1101 13	n
1110 14	o
1111 15	p

main memory

cache memory 4 sets, 2 ways, 2 words per line

2 lines per set

memory address



		word 0	word 1
00	set 0	tag 0	
		tag 1	
01	set 1	tag 0	
		tag 1	
10	set 2	tag 0	
		tag 1	
11	set 3	tag 0	
		tag 1	

(tag value = line number in set)

since tag is made up of mem address bits excluding set + word bits

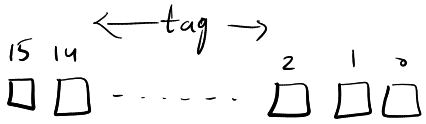
tag value will not always correspond to line number.

when we recieve a memory address from cpu we will know in which set we need to look based on set bits. inside each set we need to compare every tag, so # comparators needed = # lines per set

coming to original question, # comparators needs = 4 each with 16 bits

Note as # lines per set ↑ # comparators needed also ↑
inturn hardware complexity, Latency, cost also ↑

E.) # tag bits = 16



extra memory here is tag directory

size of tag directory = # tag bits x # total lines in cache

= 16 x # sets x # lines per set

= 16 x 2^{12} x 4

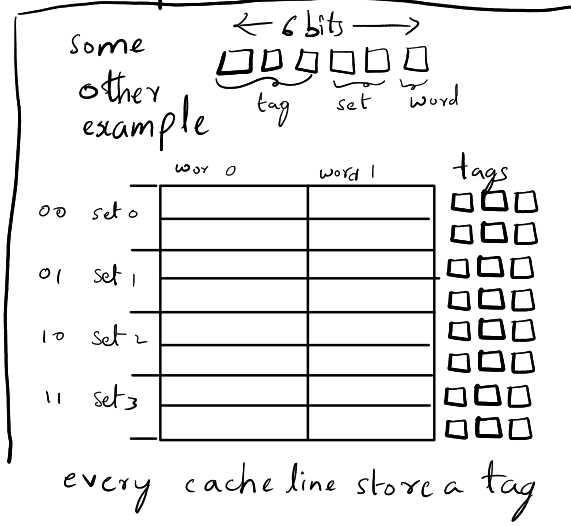
= $2^4 \times 2^{12} \times 2^2$

= 2^{18} bits

= $2^{15} \times 2^3$ bits

= $2^{15} \times 8$ bits

= 2^{15} bytes



1.2.15 Cache Memory: GATE CSE 2005 | Question: 67 https://gateoverflow.in/1390

Consider a direct mapped cache of size 32 KB with block size 32 $bytes$. The CPU generates 32 bit addresses. The number of bits needed for cache indexing and the number of tag bits are respectively,

A. 10, 17
B. 10, 22
C. 15, 17
D. 5, 17

line number

cache size 32 KB = $2^5 \times 2^{10}$ B = 2^{15} Bytes

block size = 32 bytes [memory block ≡ cache line]

block size = line capacity

assuming byte addressable memory ⇒ 1 byte = 1 word

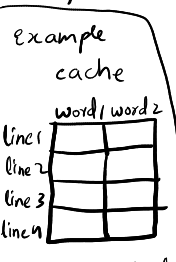
so one cache line can hold 32 words

bits required to locate word in a line = $\log_2$(# words)
= $\log_2(32) = 5$

cache lines = $\frac{\text{cache size}}{\text{line capacity}}$

= $\frac{2^{15} \text{ bytes}}{2^5 \text{ bytes}}$

= 2^{10}



1 box = 1 byte
cache size = total no. of bytes
= 8 bytes
line capacity = 2 byte
cache line = $\frac{8 \text{ bytes}}{2 \text{ bytes}}$
= 4

bits required to locate a line in cache
= # bits required for cache indexing
= $\log_2$(# cache lines) = $\log_2(2^{10}) = 10$

tag bits = total no. of memory address bits − # line bits − # word bits
= 32 − 10 − 5 = 17

Scenario Setup ⚙️

Imagine we have a very small **direct-mapped cache** that has only **4 blocks** (or lines). In a direct-mapped cache, each block from main memory can only go to one specific location in the cache. We can determine this location using the formula:

`(Memory Block Address) % (Number of Cache Blocks)`

Let's say we have the following memory blocks and their corresponding cache block destinations:

Memory Block Address	Calculation	Cache Block Destination
0	`0 % 4`	0
1	`1 % 4`	1
2	`2 % 4`	2
3	`3 % 4`	3
4	`4 % 4`	0
5	`5 % 4`	1
6	`6 % 4`	2
7	`7 % 4`	3
8	`8 % 4`	0

1. Compulsory Miss (Cold Miss) 🧊

This happens the very first time we access a block of data.

Let's access **Memory Block 2**.

Initial State: The cache is empty.

```
Cache Block 0: [      ]
Cache Block 1: [      ]
Cache Block 2: [      ] <--- We want to access data from Memory Block 2
Cache Block 3: [      ]
```

Result: It's a **miss** because the cache is empty. The data from Memory Block 2 is fetched from main memory and placed into Cache Block 2.

Final State:

```
Cache Block 0: [      ]
Cache Block 1: [      ]
Cache Block 2: [Data from Mem Block 2]  <--- Data is now cached
Cache Block 3: [      ]
```

This first miss was unavoidable because the cache had to be populated.

2. Conflict Miss 💥

This happens when the cache has space, but the specific block we need is already occupied by data that maps to the same location.

Let's continue from the previous state and now access **Memory Block 0**, and then **Memory Block 4**.

Step 1: Access Memory Block 0 (This is also a compulsory miss)

```
Cache Block 0: [      ] <--- Accessing Mem Block 0
Cache Block 1: [      ]
Cache Block 2: [Data from Mem Block 2]
Cache Block 3: [      ]
```

Result: The cache places data from Memory Block 0 into Cache Block 0.

```
Cache Block 0: [Data from Mem Block 0]
Cache Block 1: [      ]
Cache Block 2: [Data from Mem Block 2]
Cache Block 3: [      ]
```

Step 2: Now, access Memory Block 4

Remember, Memory Block 4 also maps to Cache Block 0 (`4 % 4 = 0`).

Initial State:

```
Cache Block 0: [Data from Mem Block 0] <--- Want Mem Block 4, but Block 0 is here!
Cache Block 1: [      ]
Cache Block 2: [Data from Mem Block 2]
Cache Block 3: [      ]
```

Result: This is a **conflict miss**. Even though Cache Blocks 1 and 3 are empty, Memory Block 4 *must* go into Cache Block 0. The cache is forced to **evict** the data from Memory Block 0 to make room.

Final State:

```
Cache Block 0: [Data from Mem Block 4]  <--- Old data is replaced
Cache Block 1: [      ]
Cache Block 2: [Data from Mem Block 2]
Cache Block 3: [      ]
```

3. Capacity Miss 🧺

This happens when the cache is full, and we need to fetch data that was previously evicted simply because there wasn't enough space for all the data we are actively using.

Let's fill our entire cache and then request an old item.

Step 1: Fill the cache

We access Memory Blocks 0, 1, 2, and 3.

```
Cache Block 0: [Data from Mem Block 0]
Cache Block 1: [Data from Mem Block 1]
Cache Block 2: [Data from Mem Block 2]
Cache Block 3: [Data from Mem Block 3]
```

Our cache is now completely full.

Step 2: Access a new block, forcing an eviction

Let's access **Memory Block 5**. This maps to Cache Block 1 (`5 % 4 = 1`), evicting the data from Memory Block 1.

```
Cache Block 0: [Data from Mem Block 0]
Cache Block 1: [Data from Mem Block 5] <--- Mem Block 1 was evicted
Cache Block 2: [Data from Mem Block 2]
Cache Block 3: [Data from Mem Block 3]
```

Step 3: Now, try to access Memory Block 1 again

Initial State: The cache is still full.

```
Cache Block 0: [Data from Mem Block 0]
Cache Block 1: [Data from Mem Block 5] <--- We want Mem Block 1, but it's gone!
Cache Block 2: [Data from Mem Block 2]
Cache Block 3: [Data from Mem Block 3]
```

Result: This is a **capacity miss**. Memory Block 1 was in the cache, but we had to evict it because the cache wasn't large enough to hold all the data we needed (Blocks 0, 1, 2, 3, and 5). It's a miss caused by the **limited size** of the cache. The cache will now evict the data from Memory Block 5 to load the data from Memory Block 1.

A CPU has a $32KB$ direct mapped cache with 128 byte-block size. Suppose A is two dimensional array of size 512×512 with elements that occupy 8-bytes each. Consider the following two C code segments, $P1$ and $P2$.

P1:

```
for (i=0; i<512; i++)
{
    for (j=0; j<512; j++)
    {
        x +=A[i] [j];
    }
}
```

row major

cache size = $2^5 \times 2^{10}$ bytes = 2^{15} bytes

line capacity = 128 bytes = 2^7 bytes

lines = $\frac{2^{15}}{2^7} = 2^8$

P2:

```
for (i=0; i<512; i++)
{
    for (j=0; j<512; j++)
    {
        x +=A[j] [i];
    }
}
```

column major

$P1$ and $P2$ are executed independently with the same initial state, namely, the array A is not in the cache and i, j, x are in registers. Let the number of cache misses experienced by $P1$ be M_1 and that for $P2$ be M_2.

The value of M_1 is:

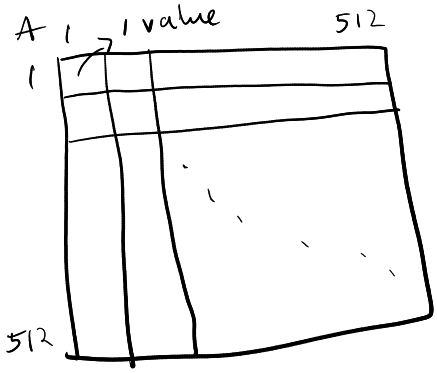
A. 0
B. 2048
C. 16384
D. 262144

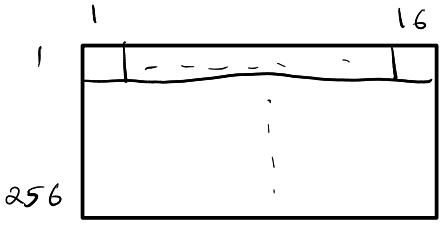
one cache line is 128 bytes

one value in array cell occupies 8 bytes

no. of values one cache line can hold = $\frac{128}{8} = 16$

main memory





512 rows, 512 columns

i	j	A[i][j]	hit/miss
0	0	A[0][0]	miss
0	1	A[0][1]	hit
⋮			⋮
0	15	A[0][15]	hit
0	16	A[0][16]	miss

move block of 16 values A[0][0] to A[0][15] into cache line 1

move block of 16 values A[0][16] to A[0][31] into cache line 2

So in every row out of 512 values for every 16 values there is a miss

$\therefore$ # misses = $M_1 = 512 \times \frac{512}{16}$

$= 512 \times 32 = 16384$

for same setting above

The value of the ratio $\frac{M_1}{M_2}$:

A. 0 → $M_1 \neq 0$

B. $\frac{1}{16}$

C. $\frac{1}{8}$

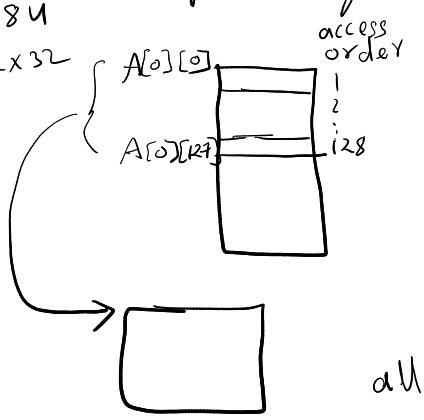
D. 16 → $M_1 < M_2$

so $\frac{M_1}{M_2} < 1$

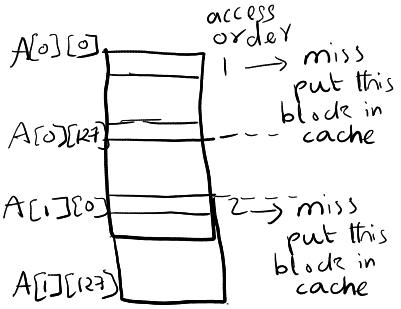
we $M_1 = 16384$

$= 512 \times 32$

previously



now



all values are miss

$\frac{M_1}{M_2} = \frac{512 \times 32}{512 \times 512} = \frac{2^5}{2^9} = \frac{1}{2^4} = \frac{1}{16}$

$M_2 = \#\text{misses} = 512 \times 512$

1.2.20 Cache Memory: GATE CSE 2007 | Question: 10 top

https://gateoverflow.in/1208

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

A. 9,6,5
B. 7,7,6
C. 7,5,8
D. 9,5,6

given,

4-way ≡ each set has 4 lines } $\#\text{sets} = \frac{\#\text{lines}}{\#\text{lines per set}}$

total no. of lines = 128

$= \frac{128}{4} = 32$

so # sets = 32

then # bits required to locate set = $\log_2(\#\text{sets}) = \log_2(32) = 5$

words in a line = 64

then # bits required to locate word = $\log_2(\#\text{words}) = \log_2(64) = 6$

bits required to locate tag = total no. of bits − # set bits − # word bits

= 20 − 5 − 6

= 20 − 11 = 9

1.2.21 Cache Memory: GATE CSE 2007 | Question: 80 top

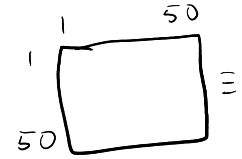
https://gateoverflow.in/1273

Consider a machine with a byte addressable main memory of 2^{16} bytes. Assume that a direct mapped data cache consisting of 32 lines of 64 bytes each is used in the system. A 50×50 two-dimensional array of bytes is stored in the main memory starting from memory location $1100H$. Assume that the data cache is initially empty. The complete array is accessed twice. Assume that the contents of the data cache do not change in between the two accesses.

How many data misses will occur in total?

A. 48
B. 50
C. 56
D. 59

Assuming access order to be row-major order, why if it was column-major then # misses = # elements in array but options are significantly less than that



≡

50 × 50 = 2500 bytes

block of memory

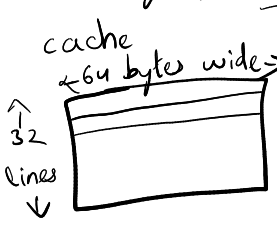
b1 1 — 64 64 bytes
b2 65 — 128 64 bytes
⋮
b39 2433 — 2496 64 bytes
b40 2497 — 2500 4 bytes

} 39

64 × 39 + 4

= 2496 + 4

= 2500



one block of memory with 64 bytes goes into one line in cache i.e 64 bytes wide

memory block b_1 — cache line l_1 → l_1 | b_1

b_2 — l_2 → l_1 | b_1 ; l_2 | b_2

1st access

b_{32} — l_{32} → l_1 | b_1 ; l_2 | b_2 ; ⋮ ; l_{32} | b_{32}

b_{33} — l_1 $33 \% 32 = 1$ → l_1 | b_{33} ; l_2 | b_2 ; ⋮ ; l_{32} | b_{32}

so far 40 misses

b_{40} — l_8 $40 \% 32 = 8$ → l_1 | b_{33} ; l_2 | b_{34} ; l_8 | b_{40} ; l_{32} | b_{32}

2nd access

8 misses

b_1 — l_1 → but l_1 already has b_{33}

so b_{33} gets evicted

b_1 miss

→ l_1 | b_1 ; l_2 | b_{34} ; l_8 | b_{40} ; l_{32} | b_{32}

⋮

b_8 — l_8 → but l_8 already has b_{40}

so b_{40} gets evicted

b_8 miss

→ l_1 | b_1 ; l_2 | b_2 ; l_8 | b_8 ; l_{32} | b_{32}

from b_9 to b_{32} all are hits

since previous data was not changed

8 misses

b_{33} — l_1 → but l_1 already has b_1

so b_1 gets evicted

b_{33} miss

⋮

b_{40} — l_8 → but l_8 already has b_8

so b_8 gets evicted

b_{40} miss

in 1st access 40 miss + 2nd access 16 misses

total no. of data misses = 56

total 128 lines 4 line sets = 4 way associative

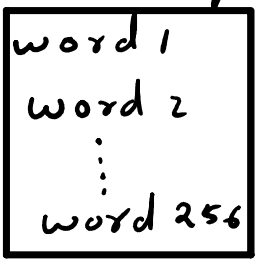
A block-set associative cache memory consists of 128 blocks divided into four block sets. The main memory consists of 16, 384 blocks and each block contains 256 eight bit words.

1. How many bits are required for addressing the main memory? 22
2. How many bits are needed to represent the TAG, SET and WORD fields?

9 5 8

1 main memory blocks goes into 1 cache line

main memory



block 16384

1 word = 8 bits = 1 byte } => memory is byte addressable

#bits required to locate a word in mem block/cache line = $\log_2$(#words)
= $\log_2(256) = 8$

main memory size = total no. of words that it can hold
= no. of blocks x no. of words per block
= 16384 x 256 words
= $2^{14} \times 2^8$ bytes
= 2^{22} bytes

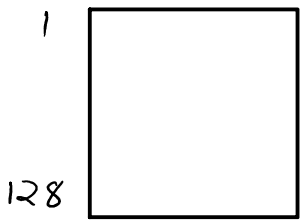
bits required to locate a word or byte in memory
= # bits required to address main memory
= $\log_2$(main memory size) = $\log_2(2^{22}) = 22$

from given,

total lines in cache = 128

lines per set = 4

=> #sets = $\frac{\text{total lines}}{\text{line per set}} = \frac{128}{4} = 32$



bits required to locate a set = $\log_2$(#sets) = $\log_2(32) = 5$

tag bits = total address bits − # set bits − # word bit = 22 − 5 − 8 = 9

Consider a machine with a 2-way set associative data cache of size 64 $Kbytes$ and block size 16 $bytes$. The cache is managed using 32 bit virtual addresses and the page size is 4 $Kbytes$. A program to be run on this machine begins as follows:

```
double ARR[1024][1024];
int i, j;
/*Initialize array ARR to 0.0 */
for(i = 0; i < 1024; i++)
    for(j = 0; j < 1024; j++)
        ARR[i][j] = 0.0;
```

The size of double is 8 bytes. Array ARR is located in memory starting at the beginning of virtual page $0xFF000$ and stored in row major order. The cache is initially empty and no pre-fetching is done. The only data memory references made by the program are those to array ARR.

The total size of the tags in the cache directory is:

A. 32 $Kbits$
B. 34 $Kbits$
C. 64 $Kbits$
D. 68 $Kbits$

to know tag directory size we need to know
tag bits

to know # tag bits need to know
set bits & # word bits

$$\text{total \# lines in cache} = \frac{\text{cache size}}{\text{line size}} = \frac{\text{total \# bytes cache can hold}}{\text{\# bytes per line}}$$

$$= \frac{64\ KB}{16\ B} = \frac{2^6 \times 2^{10}\ B}{2^4\ B} = 2^{12}$$

$$\text{\# sets} = \frac{\text{total \# lines in cache}}{\text{\# lines per set}} = \frac{2^{12}}{2} = 2^{11}$$

2 since 2 way set associative

word bits = no. of bits required to locate word or byte in one memory block or one cache line
$= \log_2(\text{line size}) = \log_2(16) = 4$

tag bits = no. of bits in memory address − # set bits − # word bits

$= 32 - 11 - 4 = 17$

tag directory size $= 2^{\#\text{tag bits}} = 2^{17}$

all possible tags $= 2^{17}$

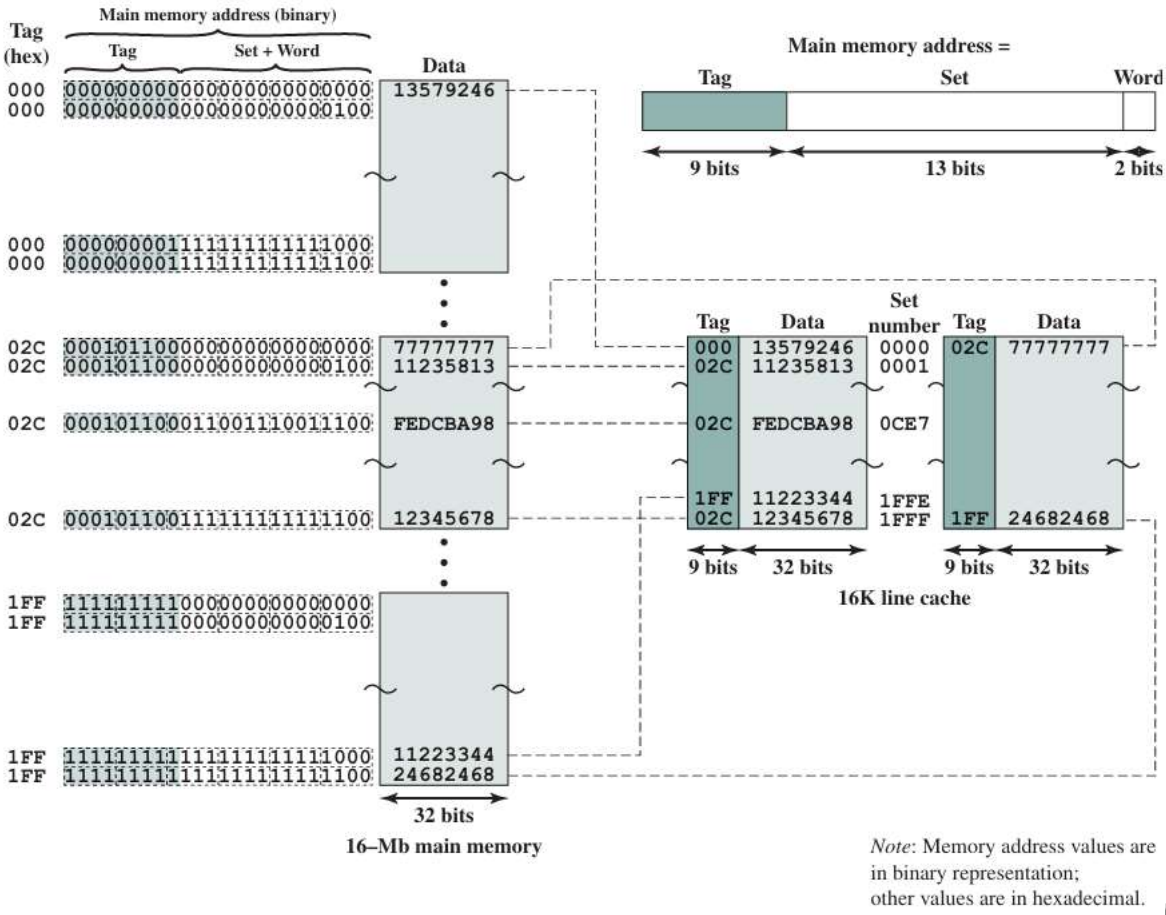


Figure 4.15 Two-Way Set-Associative Mapping Example

total no. of tags = total no. of lines in cache i.e for every line there is a tag

so tag directory size =

(# tags or # cache lines) × # tag bits

$= 2^{12} \times 17$ bits

$= 4096 \times 17$ bits $= 69632$ bits

4096*17

69632

convert bits to Kilo bits

1 Kilo bits = 1024 bits (exact)

? = 69632 bits

$? = \frac{69632}{1024} = \frac{2^{12} \times 17}{2^{10}} = 2^2 \times 17 = 68$

∴ tag directory size = 68 Kilo bits

Consider a machine with a 2-way set associative data cache of size 64 $Kbytes$ and block size 16 $bytes$. The cache is managed using 32 bit virtual addresses and the page size is 4 $Kbytes$. A program to be run on this machine begins as follows:

```
double ARR[1024][1024];
int i, j;
/*Initialize array ARR to 0.0 */
for(i = 0; i < 1024; i++)
    for(j = 0; j < 1024; j++)
        ARR[i][j] = 0.0;
```

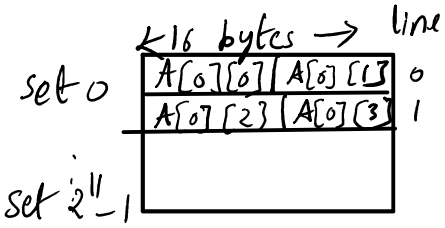
Good explanation including Virtual page address
This conversation has lot more info do check it
https://gemini.google.com/app/91c25fb7ed2008a3

The size of double is 8 bytes. Array ARR is located in memory starting at the beginning of virtual page $0xFF000$ and stored in row major order. The cache is initially empty and no pre-fetching is done. The only data memory references made by the program are those to array ARR.

Which of the following array elements have the same cache index as $ARR[0][0]$?

A. $ARR[0][4]$
B. $ARR[4][0]$
C. $ARR[0][5]$
D. $ARR[5][0]$

↳ set number



single value in A occupies 8 bytes, so one cache line can hold 2 values

From previous question breakdown of which portions of address are used for what:

Tag (17 bits)	Index (11 bits)	Offset (4 bits)
Bits 31-15	Bits 14-4	Bits 3-0

The problem tells us:

Array `ARR` is located at the beginning of **virtual page** `0xFF000`.

- This `0xFF000` is the **Virtual Page Number (VPN)**. Let's check if it's 20 bits. In hexadecimal, `F` is 4 bits (`1111`). So, `FF000` is 5 hex digits, which is $5 \times 4 = 20$ bits. Perfect!
- "At the beginning" means the **offset is 0**. We need to represent this with 12 bits, which is simply `0000 0000 0000` in binary, or `0x000` in hex.

Now, we just combine the two parts to form the full 32-bit starting address:

- **VPN (20 bits):** `0xFF000`
- **Offset (12 bits):** `0x000`

Combine them: `0xFF000 000` = `0xFF000000`

What's the size of one "page"?
The problem states the **page size is 4 KBytes**.
$4\ \text{KB} = 4 \times 1024 = 4096$ bytes.
So, each virtual "page" is a 4096-byte chunk of memory.

How big is the Page Offset?
To find any single byte within a 4096-byte page, we need a certain number of address bits. This is the **page offset**.
We calculate this with a logarithm: $\log_2(4096) = 12$.
This means the **lower 12 bits** of any 32-bit virtual address are used as the offset to pinpoint a byte *inside* its page.

How big is the Virtual Page Number (VPN)?
If the lower 12 bits are the offset, the rest of the bits must identify the page itself.
$32 \text{ total bits} - 12 \text{ offset bits} = 20 \text{ bits}$.
So, the **upper 20 bits** of a virtual address represent the **Virtual Page Number (VPN)**.

2. Finding the Cache Index for `ARR[0][0]`

The array `ARR` starts at the beginning of virtual page `0xFF000`. With a page size of 4 KBytes (2^{12} bytes), the starting address is `0xFF000` followed by 12 zero bits.

- **Address of `ARR[0][0]`:** `0xFF000000`

Let's look at the lower 15 bits of this address in binary:
`...000 0000 0000 0000`

- **Index (bits 14-4):** `00000000000` (which is 0 in decimal).
- **Offset (bits 3-0):** `0000`

"cache is managed using 32 bit virtual addresses and the page size is 4 KB". This indicates that word or byte in cache is located using virtual address only, memory management unit does address translation when needed and we don't have to worry about it. Remember in
Book [Page Number | Line Number] is like a
Main Memory [Physical Frame Number | Word Offset], i.e like
Virtual Memory [Virtual Page Number| Page Offset] after translation. Virtual address = VPN + page offset

We need to find which option also has an index of 0 then it falls in same set. The address of an element ARR[i][j] is calculated as: **Address = Base Address + (i * 1024 + j) * sizeof(double)** where the Base Address is 0xFF000000 and sizeof(double) is 8 bytes

A. `ARR[0][4]` Mul 1024 Since arr is 1024 x 1024 we need to skip 1024 elements whenever we move from one row to another

- Byte Offset = $(0 \times 1024 + 4) \times 8 = 32$.
- Address = `0xFF000000` + 32 = `0xFF000020`.
- Binary (lower bits): `...0010 0000`.
- Index bits (bits 14-4): `...0000010`. This is **2**. ❌

B. `ARR[4][0]`

- Byte Offset = $(4 \times 1024 + 0) \times 8 = 4 \times 1024 \times 8 = 32768$.
- Note that $32768 = 2^{15}$.
- Address = `0xFF000000` + 32768 = `0xFF008000`.
- Binary (lower bits): `...1000 0000 0000 0000`.
- Index bits (bits 14-4): `...00000000000`. This is **0**. ✅

Mul sizeof(double) i.e 8 since in memory one value occupies 8 bytes so if one value starts at address 0 it spans till 7 next value starts at 8 and ends at 15, so to locate next value we multiply by 8

C. `ARR[0][5]`

- Byte Offset = $(0 \times 1024 + 5) \times 8 = 40$.
- Address = `0xFF000000` + 40 = `0xFF000028`.
- Binary (lower bits): `...0010 1000`.
- Index bits (bits 14-4): `...0000010`. This is **2**. ❌

D. `ARR[5][0]`

- Byte Offset = $(5 \times 1024 + 0) \times 8 = 40 \times 1024 = 40960$
- Address = `0xFF000000` + 40960 = `0xFF00A000`.
- Binary (lower bits): `...1010 0000 0000 0000`.
- Index bits (bits 14-4): `...01000000000`. This is **512**. ❌

1.2.26 Cache Memory: GATE CSE 2008 | Question: 73

https://gateoverflow.in/43491

Consider a machine with a 2-way set associative data cache of size 64 $Kbytes$ and block size 16 $bytes$. The cache is managed using 32 bit virtual addresses and the page size is 4 $Kbytes$. A program to be run on this machine begins as follows:

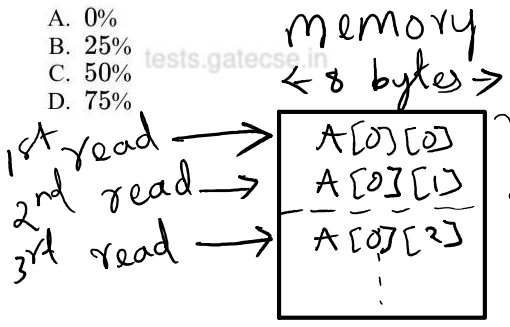
```
double ARR[1024][1024];
int i, j;
/*Initialize array ARR to 0.0 */
for(i = 0; i < 1024; i++)
    for(j = 0; j < 1024; j++)
        ARR[i][j] = 0.0;
```

The size of double is 8 bytes. Array ARR is located in memory starting at the beginning of virtual page $0xFF000$ and stored in row major order. The cache is initially empty and no pre-fetching is done. The only data memory references made by the program are those to array ARR.

The cache hit ratio for this initialization loop is:

A. 0%
B. 25%
C. 50%
D. 75%

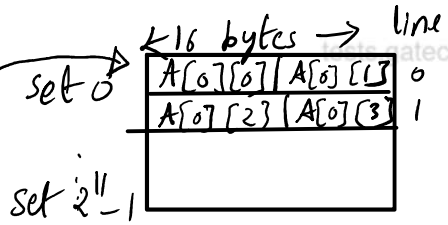
memory block → cache line/block



1st is a miss so → this memory block is written to appropriate cache line

2nd read is Hit

3rd read is miss



so out of every 2 reads 1 is a hit & 1 is a miss since one cache line can hold only 2 values at a time

so hit ratio = $\frac{1}{2}$ = 50%

Hypothetical scenario:
if cache could hold u values in a line
then hit ratio = $\frac{u-1}{u}$ & miss ratio = $\frac{1}{u}$

1.2.27 Cache Memory: GATE CSE 2009 | Question: 29

https://gateoverflow.in/1315

Consider a 4-way set associative cache (initially empty) with total 16 cache blocks. The main memory consists of 256 blocks and the request for memory blocks are in the following order:

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

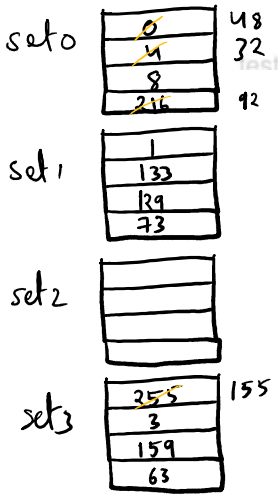
Which one of the following memory block will NOT be in cache if LRU replacement policy is used?

A. 3
B. 8
C. 129
D. ~~216~~

cache line

u-way ≡ u lines per set #sets = $\frac{\#\text{cache lines}}{\text{lines per set}} = \frac{16}{u} = 4$

t=1: 0 % 4 = 0, t=2: 255 % 4 = 3, t=3: 1 % 4 = 1
t=4: 4 % 4 = 0, t=5: 3 % 4 = 3, t=6: 8 % 4 = 0
t=7: 133 % 4 = 1, t=8: 159 % 4 = 3, t=9: 216 % 4 = 0
t=10: 129 % 4 = 1, t=11: 63 % 4 = 3, t=12: 8 % 4 = 0 (hit)
t=13: 48 % 4 = 0, t=14: 32 % 4 = 0, t=15: 73 % 4 = 1
t=16: 92 % 4 = 0, t=17: 155 % 4 = 3



though 8 entered cache before 216 it was accessed recently thus 8 won't get evicted

need to put 92, t=16 in set 0

48 @ t=13
32 @ t=14
8 @ t=12
216 @ t=9 oldest so replace

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

- 1 valid bit
- 1 modified bit
- As many bits as the minimum needed to identify the memory block mapped in the cache.

gatecse.in goclasses.in tests.gate

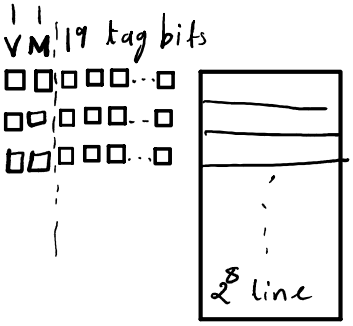
What is the total size of memory needed at the cache controller to store meta-data (tags) for the cache?

A. 4864 bits
B. 6144 bits
C. 6656 bits
D. 5376 bits

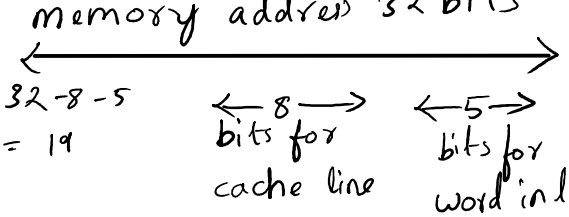
given cache size = 8 KB = $2^3 \times 2^{10}$ B = 2^{13} B

cache line can hold 32 B = 2^5 B

lines = $\frac{\text{cache size}}{\text{size of line}} = \frac{2^{13}}{2^5} = 2^8$



each cache line has a tag of 19 bits



then only tag part from address requested can be compared vs tags stored for already cached data in tag directory (part of cache memory system) and whether data is present or not can be determined.

tag directory size
= no. of bits for tag × no. of cache lines
= (1 + 1 + 19) × 2^8
= 21 × 256
= 5376

Note: if write-through policy was used then no need of modified/dirty bit

Feature	Write-Through	Write-Back
Write Operation	Data is written to both cache and main memory simultaneously.	Data is written only to the cache.
Performance	Slower write operations due to main memory latency.	Faster write operations.
Data Consistency	High. Main memory is always up-to-date.	Lower. Main memory can be temporarily out of sync.
Complexity	Simpler to implement.	More complex, requires a dirty bit.
Data Loss Risk	Low. Data is immediately saved to main memory.	Higher. Data can be lost in case of power failure.
Bandwidth Usage	High. Every write goes to main memory.	Low. Reduces writes to main memory.

1.2.31 Cache Memory: GATE CSE 2012 | Question: 54

https://gateoverflow.in/2192

A computer has a 256-KByte, 4-way set associative, write back data cache with block size of 32-Bytes. The processor sends 32-bit addresses to the cache controller. Each cache tag directory entry contains, in addition to address tag, 2 valid bits, 1 modified bit and 1 replacement bit.] useless info

The number of bits in the tag field of an address is

A. 11
B. 14
C. 16
D. 27

cache size = 256 kB = $2^8 \times 2^{10}$ B = 2^{18} B

cache line capacity = 32 byte = 2^5 B or 2^5 words

4-way set = 4 lines per set

word bits = $\log_2(2^5) = 5$

cache lines = $\frac{\text{cache size}}{\text{line capacity}} = \frac{\text{total no. of words cache can hold}}{\text{\# words per line}} = \frac{2^{18}\text{ B}}{2^5\text{ B}} = 2^{13}$

sets = $\frac{\text{\# cache lines}}{\text{\# lines per set}} = \frac{2^{13}}{4} = \frac{2^{13}}{2^2} = 2^{11}$

tag bits = total address bits − # set bits − # word bits

= 32 − 11 − 5 = 16

1.2.32 Cache Memory: GATE CSE 2012 | Question: 55

https://gateoverflow.in/43311

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

The size of the cache tag directory is:

A. 160 Kbits
B. 136 Kbits
C. 40 Kbits
D. 32 Kbits

no. of tag bits (→ actual tag bits + additional bits like these) × no. of cache lines

$= (2+1+1+16) \times 2^{11}$ bits $= 20 \times 2^{11} = 20 \times 2 \times 2^{10}$ bits $= 40$ kilobits

Mapping Technique	Number of Tags	Tag Size	Number of Tag Comparisons per Access	Reason for Comparisons
Direct-Mapped	Number of Cache Lines	Smallest	1	The memory address's index points to a **single, specific cache line**. The system only needs to check the tag of that one line to determine a hit or miss.
N-Way Set-Associative	Number of Cache Lines	Medium	N (ways)	The address's index points to a **specific set**. The desired block could be in any of the 'N' lines within that set, so all 'N' tags in the set must be checked simultaneously.
Fully Associative	Number of Cache Lines	Largest	All lines in the cache	There is **no index**, so a memory block could be in any line in the entire cache. To find it, the system must compare the address tag against the tags of every single line in the cache at once.

1.2.33 Cache Memory: GATE CSE 2013 | Question: 20

https://gateoverflow.in/1442

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

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

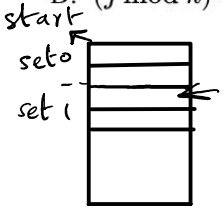
mem block j must go into one of v sets

so its $j \bmod v = j \,\%\, v$ → set number

$(j \,\%\, v) \times k$ → tells where the starting line in each set is

$(j \,\%\, v) \times k + (k-1)$ → last line in set

$(j \,\%\, v) \times k + 0, (j \,\%\, v \times k) + 1, \ldots (j \,\%\, v) \times k + (k-1)$ i.e k lines



An access sequence of cache block addresses is of length N and contains n unique block addresses. The number of unique block addresses between two consecutive accesses to the same block address is bounded above by k. What is the miss ratio if the access sequence is passed through a cache of associativity $A \geq k$ exercising least-recently-used replacement policy?

A. $\left(\frac{n}{N}\right)$
B. $\left(\frac{1}{N}\right)$
C. $\left(\frac{1}{A}\right)$
D. $\left(\frac{k}{n}\right)$

lets take an example https://gemini.google.com/app/ec86fc67acc14d88

unique blocks = $\{B_0, B_1, B_2, B_3, B_4\}$ $n = 5$

access sequence $B_0, B_1, B_2, B_3, B_2, B_4, B_3$ $N = 7$ (B_2 to B_3: $1 < K$; B_2 to B_4: $2 < K$)

$K = 3$ i.e $B_i, \ldots, B_i$ ($< K$)

associativity $A = 4$, no info on set associative or fully associative. we can assume two cases:

1) set associative with all Blocks mapping to same set, since no mechanism was given to calculate into which set a Block goes into.

2.) fully associative, a memory Block can go into any line.

access	Hit or miss	cache [] initially empty
B_0	miss	$[B_0]$
B_1	miss	$[B_1, B_0]$
B_2	miss	$[B_2, B_1, B_0]$
B_3	miss	$[B_3, B_2, B_1, B_0]$
B_2	Hit	$[B_2, B_3, B_1, B_0]$
B_4	miss	$[B_4, B_2, B_3, B_1]$ B_0 gets evicted
B_3	Hit	$[B_3, B_4, B_2, B_1]$

misses = # unique block = n i.e all are compulsory misses out of N = 7 accesses = miss ratio = $\frac{n}{N}$

1.2.35 Cache Memory: GATE CSE 2014 Set 2 | Question: 43 top

https://gateoverflow.in/2009

In designing a computer's cache system, the cache block (or cache line) size is an important parameter. Which one of the following statements is correct in this context?

A. A smaller block size implies better spatial locality
B. A smaller block size implies a smaller cache tag and hence lower cache tag overhead
C. A smaller block size implies a larger cache tag and hence lower cache hit time
D. A smaller block size incurs a lower cache miss penalty

option A, B and C are not correct and D is correct.
A: for higher spacial locality, if we access a word x from cache then as many words as possible around x must be in same cache line.

B: smaller block size implies more cache lines/sets each of which has a tag, then more tag comparisons are needed so high cache tag overhead.

D: miss penalty is extra time needed to fetch and write group of words from main memory, if cache block size is small then less words needs to be fetched and written, this lower miss penalty.

C:

Claim 1: "A smaller block size implies a larger cache tag"

This is not true under standard assumptions. A physical memory address is typically divided into three parts to find data in the cache:

```
[ Tag | Index | Block Offset ]
```

Let's see how changing the block size affects these parts, assuming the total data capacity of the cache is fixed:

1. **Block Offset:** This part identifies a specific word within a block. The number of offset bits is determined by the block size (Offset bits = $\log_2$(Block Size)). If you make the **block size smaller**, you need **fewer offset bits**.
2. **Index:** This part identifies which cache line (or set) the data belongs to. The number of lines is `Cache Size / Block Size`. If you make the **block size smaller**, the number of lines in the cache **increases**, which means you need **more index bits**.
3. **Tag:** This part is what's left over. The total number of bits in a physical address is constant.

As it turns out, the decrease in offset bits is exactly canceled out by the increase in index bits. This means the **size of the tag remains the same**.

Example:

- Assume a 32-bit address and a 64 KB direct-mapped cache.
- **Case 1: 64-byte blocks**
  - Offset = $\log_2(64)$ = 6 bits
  - Index = $\log_2$(64 KB / 64 B) = $\log_2(1024)$ = 10 bits
  - Tag = 32 - 10 - 6 = **16 bits**
- **Case 2: 32-byte blocks (smaller)**
  - Offset = $\log_2(32)$ = 5 bits (decreased by 1)
  - Index = $\log_2$(64 KB / 32 B) = $\log_2(2048)$ = 11 bits (increased by 1)
  - Tag = 32 - 11 - 5 = **16 bits** (no change)

Since the first part of the statement is false, the entire proposition is incorrect.

Claim 2: "...and hence lower cache hit time"
This part of the statement is also flawed. Cache hit time is the time taken to retrieve data that is already in the cache. It depends on factors like the time to compare tags and the time to select the data from the block.

- A smaller block size means less data to choose from within a cache line, which could slightly speed up the data selection process (the final step of a hit).
- However, a smaller block size also means more cache lines, which requires a larger and potentially slower decoder to select the correct line based on the index.

The overall effect on hit time is complex and not guaranteed to be lower. More importantly, the reasoning provided ("hence") is based on a false premise (a larger tag). A larger tag would require a larger comparator, which would likely *increase*, not decrease, the hit time.
From <https://www.perplexity.ai/search/i-know-why-option-a-and-b-are-h6NTjb79SquZaBheWHbMrA>

1.2.36 Cache Memory: GATE CSE 2014 Set 2 | Question: 44

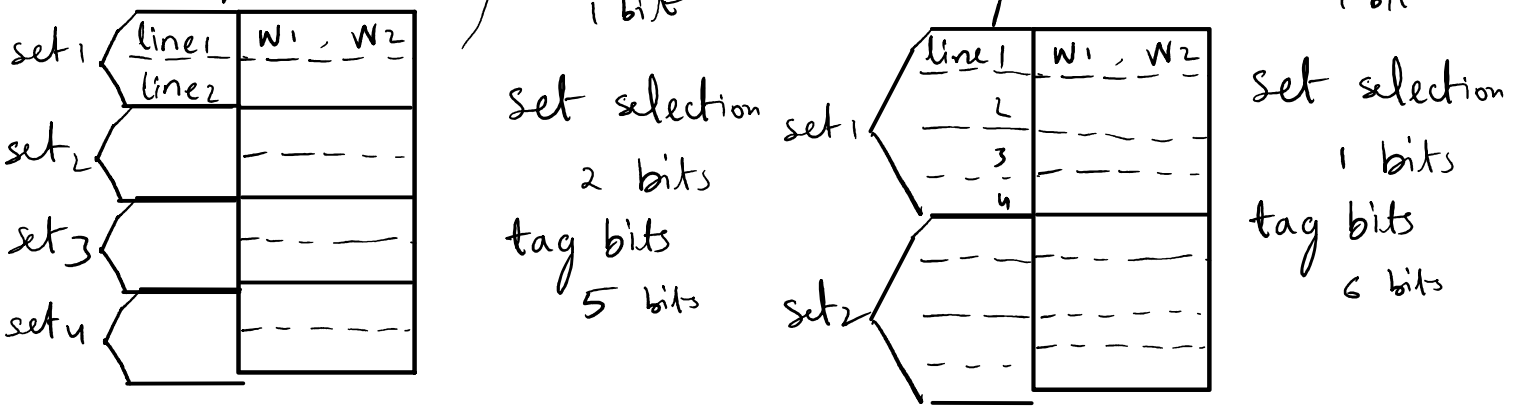
https://gateoverflow.in/2010

If the associativity of a processor cache is doubled while keeping the capacity and block size unchanged, which one of the following is guaranteed to be NOT affected?

goclasses.in tests.gate

A. Width of tag comparator
B. Width of set index decoder
C. Width of way selection multiplexer
D. Width of processor to main memory data bus

address 8 bits



way/line selection multiplexer
previously needed to route 2 ways now has to route 4 ways

1.2.37 Cache Memory: GATE CSE 2014 Set 2 | Question: 9

https://gateoverflow.in/1963

A 4-way set-associative cache memory unit with a capacity of 16 KB is built using a block size of 8 words. The word length is 32 bits. The size of the physical address space is 4 GB. The number of bits for the TAG field is ____

cache size = 16 KB = $2^4 \times 2^{10}$ B

line/block size = 8 words = 8 x 32 bits = 8 x 4 x 8 bits
= 32 bytes
= 32 B
↳ # word offset bits = $\log_2(32) = 5$

cache lines = $\frac{\text{cache size}}{\text{line size}} = \frac{2^{14}\text{ B}}{2^5\text{ B}} = 2^9$

sets = $\frac{\#\text{cache lines}}{\#\text{lines per set}} = \frac{2^9}{4} = 2^7$
↳ # set index bits $= \log_2(2^7) = 7$

physical address space = 4 GB
i.e main memory size = $2^2 \times 2^{30}$ B = 2^{32} B

bits needed to locate word in main memory
= $\log_2$(main memory size) = $\log_2(2^{32}) = 32$

tag bits = 32 − # set bits − # word bits = 32 − 7 − 5 = 20

1.2.40 Cache Memory: GATE CSE 2015 Set 3 | Question: 14 top

main memory address bits = 20

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

Hex	4 bit binary
0	0000
1	
2	
⋮	⋮
A: 10	1010
⋮	⋮
F: 15	1111

Tag, line

~~A.~~ E, 201

~~B.~~ F, 201

~~C.~~ E, E20

~~D.~~ 2, 01F

Tag is left most / most significant bits

option B has right most / least significant bits

D has some middle bits as tag

Tag bits & line bits are clearly seperate with no overlap but in option C has hex value E in both tag & line parts

Systematic ways:

bits in physical memory address = $\log_2$(size of memory)
= $\log_2(2^{20}) = 20$ bits

block size = 16 B ⇒ # word offset bits = $\log_2$(block size)
= $\log_2(16) = 4$

cache lines = 2^{12} ⇒ # line bits = $\log_2$(# lines)
= $\log_2(2^{12}) = 12$

$(E201F)_{16}$ = E 2 0 1 F = 1110 0010 0000 0001 1111

1110: remaining for tag; 0010 0000 0001: 12 bits for line; 1111: 4 bits for word

∴ Tag = $(E)_{16}$ & line = $(201)_{16}$

1.2.41 Cache Memory: GATE CSE 2016 Set 2 | Question: 32 top

https://gateoverflow.in/39622

The width of the physical address on a machine is 40 bits. The width of the tag field in a 512 KB 8-way set associative cache is 24 bits.

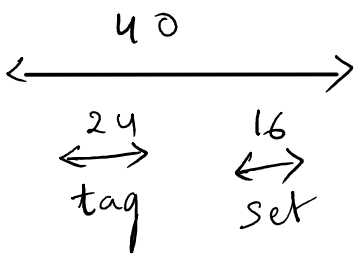
since no info given about block/line size we will assume 1 block/line can hold 1 word/byte cause that is minimum

cache size = $2^9 \times 2^{10}$ B = 2^{19} B

1 byte – 1 word, 8 way ≡ 1 set has 8 lines/block

sets = $\frac{2^{19} B}{8 B} = \frac{2^{19} B}{2^3 B} = 2^{16}$

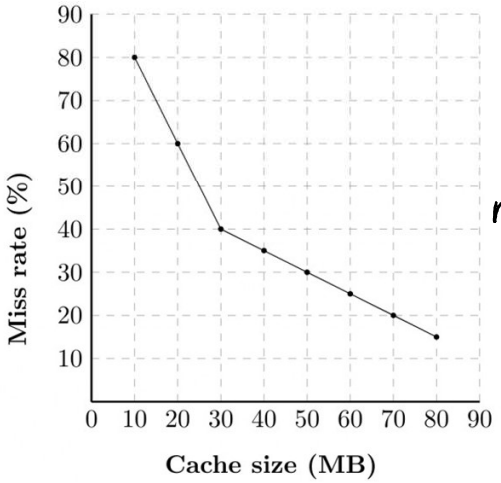
* since there is only 1 word per line no need of word offset bits, just locating set is enough
* if tag matches we will retrieve word from line corresponding to matched tag.



1.2.42 Cache Memory: GATE CSE 2016 Set 2 | Question: 50 top

https://gateoverflow.in/39592

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



avg latency = hit ratio x cache latency + miss ratio x memory latency

always strive for lowest miss rate

miss rate	cache	avg latency
20%	70MB	(0.8) 1 ms + (0.2) 10 ms = (0.8 + 2) ms = 2.8
30%	50MB	(0.7) 1 + (0.3) 10 = (0.7 + 3) ms = 3.7
40%	30MB	(0.6) 1 + (0.4) 10 = (0.6 + 4) ms = 4.6
50%	not available	
60%	20MB	(0.4) 1 + (0.6) 10 = (0.4 + 6) ms = 6.4

The smallest cache size required to ensure an average read latency of less than 6 ms is 30 MB.

1.2.44 Cache Memory: GATE CSE 2017 Set 1 | Question: 54 top

https://gateoverflow.in/118748

A cache memory unit with capacity of N words and block size of B words is to be designed. If it is designed as a direct mapped cache, the length of the TAG field is 10 bits. If the cache unit is now designed as a 16 – *way* set-associative cache, the length of the TAG field is _____ bits.

#word bits = $\log_2(B)$ #tag bits = 10

line bits = $\log_2$(#lines) = $\log_2(\frac{N}{B})$

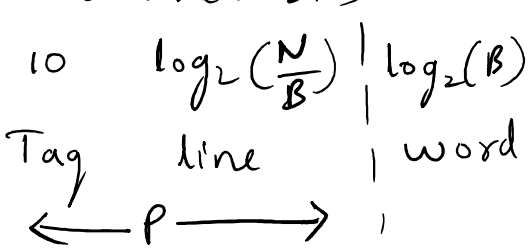
#sets = $\frac{\#\text{lines}}{\#\text{lines per set}} = \frac{(N/B)}{16}$

set bits = $\log_2(\frac{N}{16B})$

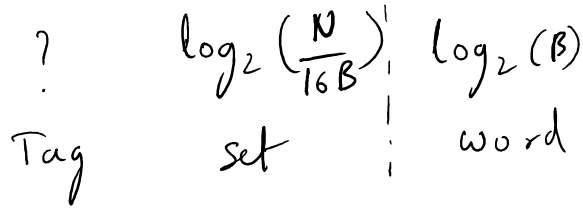
block size / line size doesn't change

So # word bits also don't change

old



new



$P = 10 + \log_2(\frac{N}{B}) = ? + \log_2(\frac{N}{16B})$ let $\frac{N}{B} = X$

$10 + \log_2(X) = ? + \log_2(\frac{X}{16})$ we know $\log \frac{a}{b} = \log a - \log b$

$10 + \log_2(X) = ? + \log_2(X) - \log_2(16)$

$10 = ? - \log_2(2^4)$

$? = 10 + 4 = 14$ i.e no. of new Tag bits

1.2.47 Cache Memory: GATE CSE 2017 Set 2 | Question: 53 https://gateoverflow.in/118613

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

word bits = $\log_2(32) = 5$ # line bits = $\log_2(512) = 9$

tag bits = $32 - 9 - 5 = 18$

1.2.48 Cache Memory: GATE CSE 2018 | Question: 34 https://gateoverflow.in/204108

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

A. $P - N - \log_2 K$
B. $P - N + \log_2 K$
C. $P - N - M - W - \log_2 K$
D. $P - N - M - W + \log_2 K$

→ main memory address bits = P

cache size = 2^N Bytes

block/line size: 2^M words = $2^M \times 2^W$ Bytes = 2^{M+W} Bytes

bits to locate word in line = $\log_2(2^{M+W}) = M+W$

lines = $\frac{\text{cache size}}{\text{line size}} = \frac{2^N B}{2^{M+W} B}$ # set bits = $\log_2(\frac{2^{N-M-W}}{K}) = N - M - W - \log_2 K$

#Sets = $\frac{\#\text{lines}}{\#\text{lines per set}} = \frac{2^{N-M-W}}{K} = 2^{N-M-W}$ # tag bits = $P - (N - M - W - \log_2 K) - (M + W)$

$= P - N + M + W + \log_2 K - M - W$

1.2.49 Cache Memory: GATE CSE 2019 | Question: 1 top

A certain processor uses a fully associative cache of size 16 kB, The cache block size is 16 bytes. Assume that the main memory is byte addressable and uses a 32-bit address. How many bits are required for the *Tag* and the *Index* fields respectively in the addresses generated by the processor?

A. 24 bits and 0 bits
B. 28 bits and 4 bits
C. 24 bits and 4 bits
D. 28 bits and 0 bits

cache size = 16 kB = $2^4 \times 2^{10}$ B = 2^{14} B

block/line size = 16 B = 2^4 B ⇒ #word/index bits = $\log_2(2^4)$ = 4

#tag bits = 32 − 4 = 28

in fully associative cache no need of set bits or line bits

tag bits are enough to locate line/block in cache.

1.2.52 Cache Memory: GATE CSE 2020 | Question: 30 top

https://gateoverflow.in/333201

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

- $A1 = 0x42C8A4$,
- $A2 = 0x546888$,
- $A3 = 0x6A289C$,
- $A4 = 0x5E4880$

main memory 16 MB = $2^4 \times 2^{20}$ B = 2^{24} B

main mem address bits = $\log_2$(mem size) = $\log_2(2^{24})$ = 24

total cache size = 64 KB = $2^6 \times 2^{10}$ B = 2^{16} B

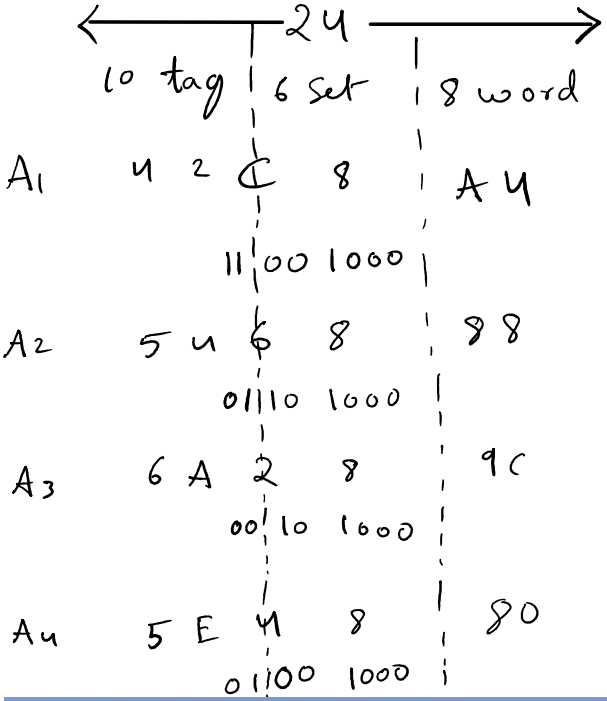
block/line size = 256 bytes = 2^8 B → 1 line has 2^8 bytes or words so 8 bits will locate word in line

$$\#\text{lines} = \frac{\text{cache size}}{\text{line size}} = \frac{2^{16}}{2^8} = 2^8$$

$$\#\text{sets} = \frac{\#\text{lines}}{\#\text{lines per set}} = \frac{2^8}{4} = \frac{2^8}{2^2} = 2^6$$

Which one of the following is TRUE?

A. $A1$ and $A4$ are mapped to different cache sets.
B. $A2$ and $A3$ are mapped to the same cache set.
C. $A3$ and $A4$ are mapped to the same cache set.
D. $A1$ and $A3$ are mapped to the same cache set.



A2 & A3 same set

A1 & A4 same set

0000	0	1000	8
0001	1	1001	9
0010	2	1010	A
0011	3	1011	B
0100	4	1100	C
0101	5	1101	D
0110	6	1110	E
0111	7	1111	F

1.2.53 Cache Memory: GATE CSE 2021 Set 1 | Question: 22 top

https://gateoverflow.in/357429

Consider a computer system with a byte-addressable primary memory of size 2^{32} bytes. Assume the computer system has a direct-mapped cache of size 32 KB (1 KB = 2^{10} bytes), and each cache block is of size 64 bytes.

The size of the tag field is _____ bits.

$$\text{Tag bits} = \text{PAS}_{bits} - \log_2(\text{Cache Size}) + \log_2(K) \text{ (where } K \text{ is associativity)}$$

$$= 32 - 15 + 0 = 17 \text{ bits}$$

1.2.54 Cache Memory: GATE CSE 2021 Set 2 | Question: 19

https://gateoverflow.in/357521

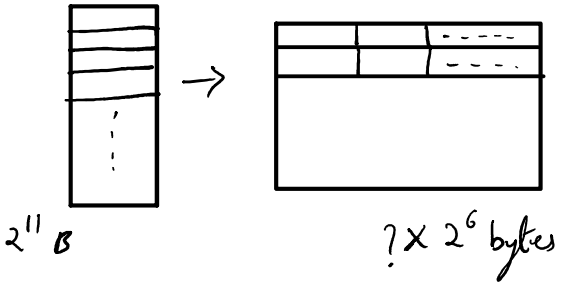
Consider a set-associative cache of size 2KB ($1KB = 2^{10}$ bytes) with cache block size of 64 bytes. Assume that the cache is byte-addressable and a 32 -bit address is used for accessing the cache. If the width of the tag field is 22 bits, the associativity of the cache is 2

2^6 #word bits = 6

bits for tag T = 22 & word W = 6

$32 = T + s + w$

$32 = 22 + s + 6$



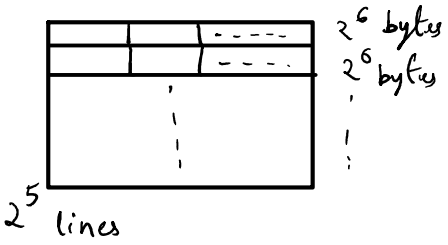
#set bits $s = 4 \rightarrow$ #sets $= 2^4$

cache size = 2KB = $2^1 \times 2^{10}$ B = 2^{11} B

$? \times 2^6$ bytes = 2^{11} bytes

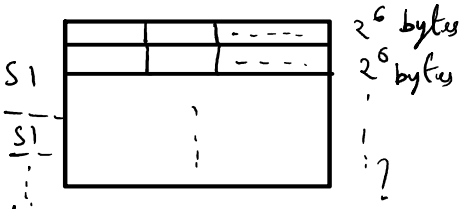
no.of lines x no.of words/bytes per line = total no.of words or byte a cache can store

$? = \frac{2^{11}}{2^6} = 2^5$ lines



there are 2^4 sets

how many lines per set?



no.of sets x no.of lines per set = no.of total lines

$2^4 \times ? = 2^5 \Rightarrow ? = 2$ i.e 2 way associative

1.2.57 Cache Memory: GATE IT 2004 | Question: 48

https://gateoverflow.in/3691

Consider a fully associative cache with 8 cache blocks (numbered $0 - 7$) and the following sequence of memory block requests:

4, 3, 25, 8, 19, 6, 25, 8, 16, 35, 45, 22, 8, 3, 16, 25, 7

If LRU replacement policy is used, which cache block will have memory block 7?

A. 4
~~B. 5~~
C. 6
D. 7

	4 miss write	3 miss	25 miss	8 miss	19 miss	6 miss	25 hit	8 hit	16 miss
0:	4 t_1	4 t_1	4 t_1	4 t_1	4 t_1	4 t_1	4 t_1	4 t_1	4 t_1
1:		3 t_2	3 t_2	3 t_2	3 t_2	3 t_2	3 t_2	3 t_2	3 t_2
2:			25 t_3	25 t_3	25 t_3	25 t_3	25 t_7	25 t_7	25 t_7
3:				8 t_4	8 t_4	8 t_4	8 t_4	8 t_8	8 t_8
4:					19 t_5	19 t_5	19 t_5	19 t_5	19 t_5
5:						6 t_6	6 t_6	6 t_6	6 t_6
6:									16 t_9
7:									

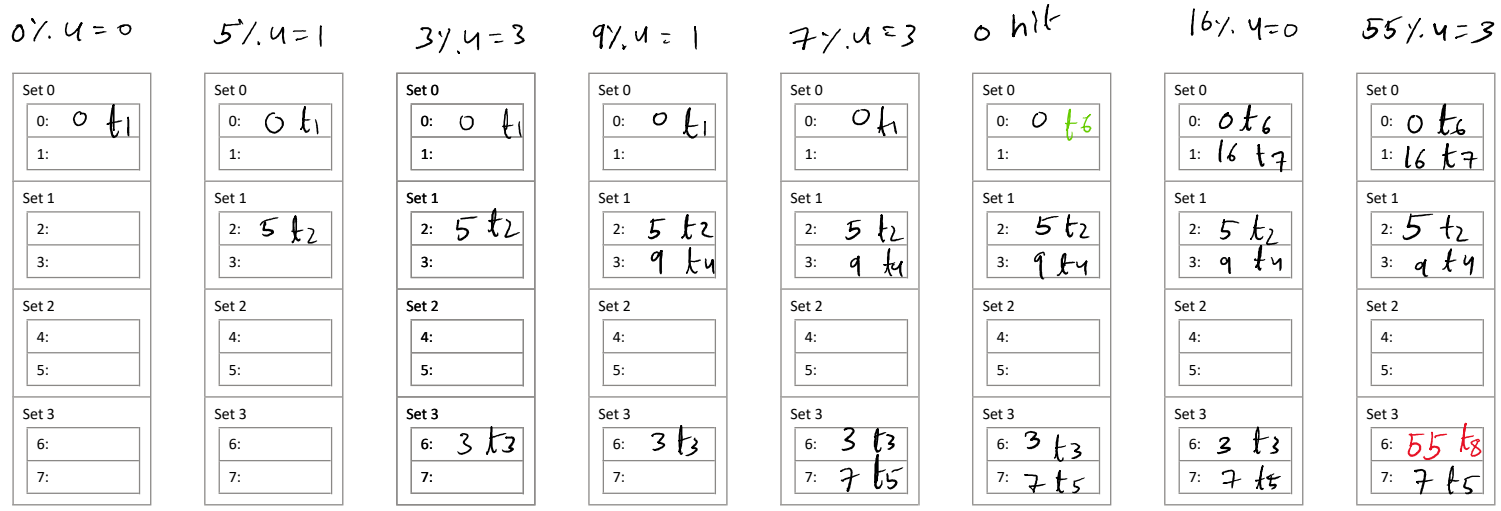
	35 miss	45 miss 4 X	22 miss 3X	8 hit	3 miss 19X	16 hit	25 hit	7 miss 6X
0:	4 t_1	45 t_{11}	45 t_{11}	45 t_{11}	45 t_{11}	45 t_{11}	45 t_{11}	45 t_{11}
1:	3 t_2	3 t_2	22 t_{12}	22 t_{12}	22 t_{12}	22 t_{12}	22 t_{12}	22 t_{12}
2:	25 t_7	25 t_7	25 t_7	25 t_7	25 t_7	25 t_7	25 t_{16}	25 t_7
3:	8 t_8	8 t_8	8 t_8	8 t_{13}	8 t_{13}	8 t_{13}	8 t_{13}	8 t_{13}
4:	19 t_5	19 t_5	19 t_5	19 t_5	3 t_{14}	3 t_{14}	3 t_{14}	3 t_{14}
5:	6 t_6	6 t_6	6 t_6	6 t_6	6 t_6	6 t_6	6 t_6	7 t_{17}
6:	16 t_9	16 t_9	16 t_9	16 t_9	16 t_9	16 t_{15}	16 t_9	16 t_9
7:	35 t_{10}	35 t_{10}	35 t_{10}	35 t_{10}	35 t_{10}	35 t_{10}	35 t_{10}	35 t_{10}

1.2.58 Cache Memory: GATE IT 2005 | Question: 61 top

Consider a 2-way set associative cache memory with 4 sets and total 8 cache blocks $(0-7)$ and a main memory with 128 blocks $(0-127)$. What memory blocks will be present in the cache after the following sequence of memory block references if **LRU** policy is used for cache block replacement. Assuming that initially the cache did not have any memory block from the current job?

0 5 3 9 7 0 16 55

A. 0 3 5 7 16 55
B. 0 3 5 7 9 16 55
C. 0 5 7 9 16 55
D. 3 5 7 9 16 55



1.2.61 Cache Memory: GATE IT 2007 | Question: 37 top https://gateoverflow.in/3470

Consider a Direct Mapped Cache with 8 cache blocks (numbered $0-7$). If the memory block requests are in the following order

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

Which of the following memory blocks will not be in the cache at the end of the sequence ?

A. 3
B. 18
C. 20
D. 30

tests.gatecse.in goclasses.in tests.gate

time →

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

line: 3 5 2 0 0 7 1 0 4 1 1 2 6 0 2 7 5 2 1 0

%8

1.2.62 Cache Memory: GATE IT 2008 | Question: 80 top https://gateoverflow.in/3403

Consider a computer with a 4-ways set-associative mapped cache of the following characteristics: a total of 1 MB of main memory, a word size of 1 byte, a block size of 128 words and a cache size of 8 KB. $= 2^3 \times 2^{10}\ B = 2^{13}$

↳ $2^7 \Rightarrow W = 7$ bits

The number of bits in the TAG, SET and WORD fields, respectively are:

A. 7, 6, 7
B. 8, 5, 7
C. 8, 6, 6
D. 9, 4, 7

$$\#\text{Sets} = \frac{\#\text{lines}}{\#\text{lines per set}} = \frac{\left(\frac{\text{cache size}}{\text{block/line size}}\right)}{\#\text{lines per set}} = \frac{\frac{2^{13}\ B}{2^7\ B}}{4} = \frac{2^6}{2^2} = 2^4$$

$$\#\text{set bits} = \log_2(\#\text{sets}) = \log_2(2^4) = 4$$

Consider a computer with a 4-ways set-associative mapped cache of the following characteristics: a total of 1 MB of main memory, a word size of 1 *byte*, a block size of 128 words and a cache size of 8 KB. 20 bit mem addr $\Leftarrow 1 \times 2^{20} B$

↳ $2^7 \Rightarrow$ # word bits = 7

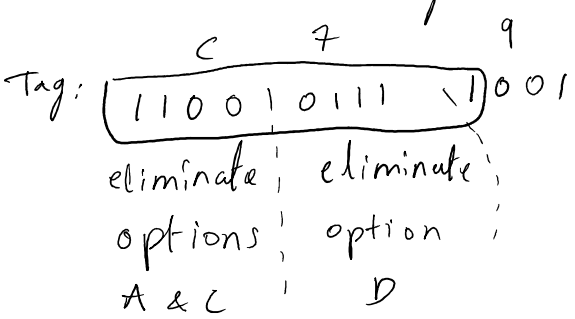
While accessing the memory location $0C795H$ by the **CPU**, the contents of the TAG field of the corresponding cache line is:

~~A.~~ 000011000
~~B.~~ 110001111
~~C.~~ 00011000
~~D.~~ 110010101

$$\#Sets = \frac{\#lines}{\#lines\ per\ set} = \frac{\left(\frac{cache\ size}{block/line\ size}\right)}{\#lines\ per\ set} = \frac{\frac{2^{13}B}{2^7 B}}{4} = \frac{2^6}{2^2} = 2^4$$

$$\#\ set\ bits = \log_2(\#sets) = \log_2(2^4) = 4$$

$$\#\ tag\ bits = 20 - 4 - 7 = 9$$



0000	0	1000	8
0001	1	1001	9
0010	2	1010	A
0011	3	1011	B
0100	4	1100	C
0101	5	1101	D
0110	6	1110	E
0111	7	1111	F

---

access time basics: lets say we made 10 memory access cache - 1 ns, main mem - 1 μs

time taken for access i = ta_i $\{^{H}_{M}\}$ → hit or miss

time taken for access 1 if Hit = ta_1^H, time taken for access 2 if miss = ta_2^M

total time for all memory access — 7 hits, 3 Miss

$$= ta_1^H + ta_2^M + ta_3^H + ta_4^H + ta_5^M + ta_6^H + ta_7^H + ta_8^H + ta_9^H + ta_{10}^M$$

$$= 1ns + (1ns + 1\mu s + 1ns) + 1ns + 1ns + (1ns + 1\mu s + 1ns) + 1ns + 1ns + 1ns + (1ns + 1\mu s + 1ns)$$

$$= 7 \times 1ns + 3 \times (1ns + 1\mu s + 1ns)$$

- 7 ← no. of hits
- 1ns ← cache access time on hit
- 3 ← no. of miss
- $(1ns + 1\mu s + 1ns)$ ← 1st access cache and (1ns) got to know its a miss; Miss penalty { then read memory (1μs) and update cache (1ns)

update cache happens implicitly so its time is largely ignored

average time taken for all memory access : total No. of access N = 10

$$\frac{7 \times 1ns + 3 \times (1ns + 1\mu s + 1ns)}{10} = \frac{7 \times 1ns}{10} + \frac{3 \times (1ns + 1\mu s + 1ns)}{10}$$

$\frac{7}{10}$ ← hit ratio; $\frac{3}{10}$ ← miss ratio

In the three-level memory hierarchy shown in the following table, p_i denotes the probability that an access request will refer to M_i.

tests.gatecse.in goclasses.in tests.gatecse.in

Hierarchy Level (M_i)	Access Time (t_i)	Probability of Access (p_i)	Page Transfer Time (T_i)
M_1	10^{-6}	0.99000	0.001 sec
M_2	10^{-5}	0.00998	0.1 sec
M_3	10^{-4}	0.00002	---

tests.gatecse.in goclasses.in tests.gatecse.in

If a miss occurs at level M_i, a page transfer occurs from M_{i+1} to M_i and the average time required for such a page swap is T_i. Calculate the average time t_A required for a processor to read one word from this memory system.

$$t_a = \underbrace{p_1}_{\text{Chance the word is in M1}} \times \underbrace{(t_1)}_{\text{Time to read from M1}}$$

$$+ \underbrace{p_2}_{\text{Chance the word is in M2}} \times \underbrace{\left(\overbrace{t_1}^{\text{1. Check M1 (it's a miss)}} + \overbrace{t_2}^{\text{2. Read from M2 (it's a hit)}} + \overbrace{T_1}^{\text{3. Update M1 with data from M2}} \right)}_{\text{Total time cost when the word is found in M2}}$$

$$\underbrace{p_3}_{\text{Chance the word is in M3}} \times \underbrace{\left(\overbrace{t_1}^{\text{1. Check M1 (miss)}} + \overbrace{t_2}^{\text{2. Check M2 (miss)}} + \overbrace{t_3}^{\text{3. Read from M3 (hit)}} + \overbrace{T_2}^{\text{4. Update M2 with data from M3}} + \overbrace{T_1}^{\text{5. Update M1 with data from M2}} \right)}_{\text{Total time cost when the word is found in M3}}$$

Step 1: Identify the Three Possible Scenarios

Scenario 1: Hit at M_1

- Probability: $p_1 = 0.99000$
- Time required: $t_1 = 10^{-6}$ sec

Scenario 2: Miss at M_1, Hit at M_2

- Probability: $p_2 = 0.00998$
- Time required: $t_1 + T_1 + t_2$
- Calculation: $10^{-6} + 0.001 + 10^{-5} = 0.001011$ sec

Scenario 3: Miss at M_1 and M_2, Hit at M_3

- Probability: $p_3 = 0.00002$
- Time required: $t_1 + T_1 + t_2 + T_2 + t_3$
- Calculation: $10^{-6} + 0.001 + 10^{-5} + 0.1 + 10^{-4} = 0.101111$ sec

Step 2: Apply the Average Access Time Formula

The **average access time** t_a is calculated using:

$t_a = p_1 \times t_1 + p_2 \times (t_1 + T_1 + t_2) + p_3 \times (t_1 + T_1 + t_2 + T_2 + t_3)$

Step 3: Calculate Each Term

Term 1: $0.99000 \times 10^{-6} = 9.9 \times 10^{-7}$ sec

Term 2: $0.00998 \times 0.001011 = 1.009 \times 10^{-5}$ sec

Term 3: $0.00002 \times 0.101111 = 2.022 \times 10^{-6}$ sec

Step 4: Sum All Terms

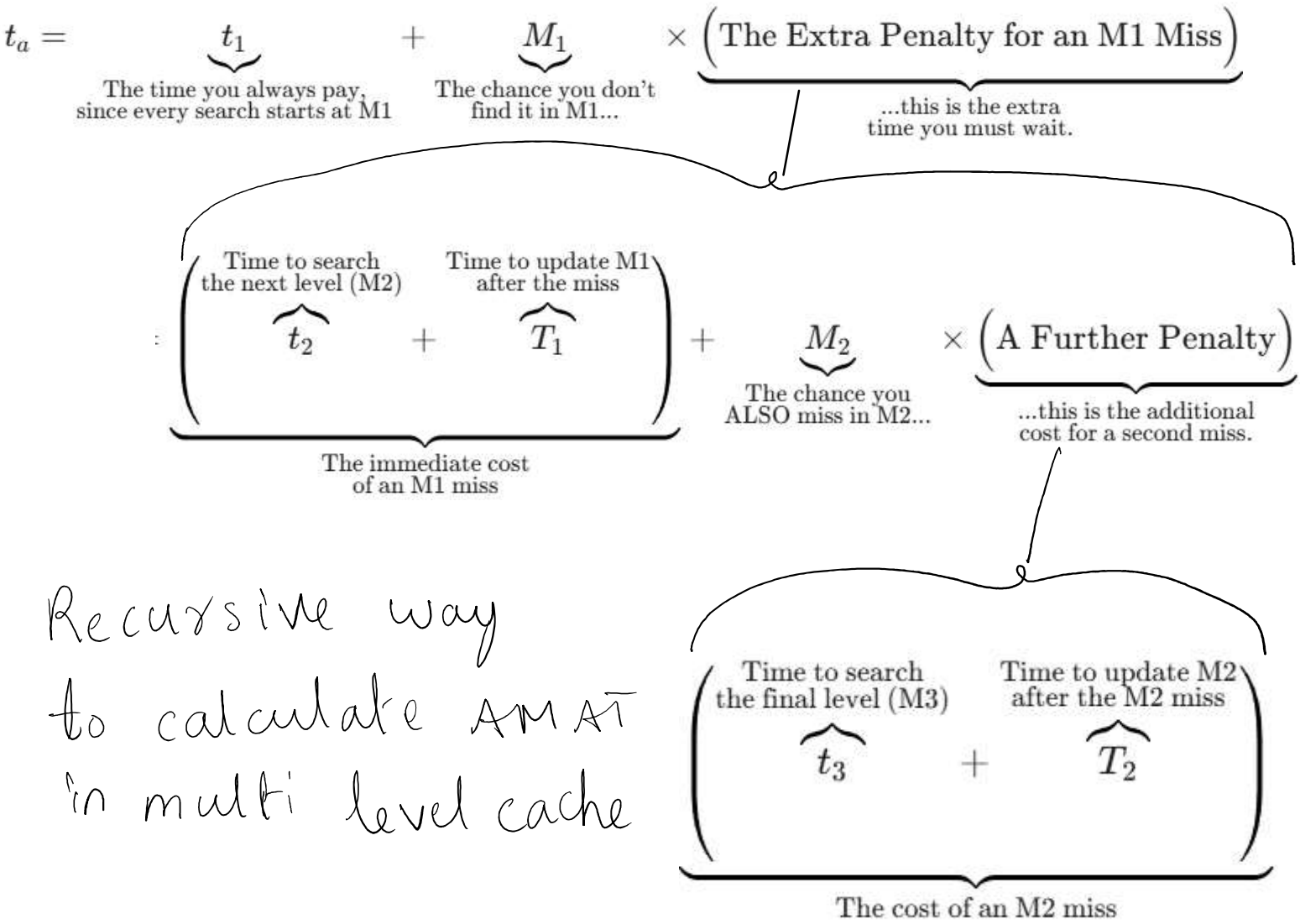
$t_a = 9.9 \times 10^{-7} + 1.009 \times 10^{-5} + 2.022 \times 10^{-6}$

Converting to the same power of 10:

$t_a = 0.99 \times 10^{-6} + 10.09 \times 10^{-6} + 2.022 \times 10^{-6}$

$t_a = (0.99 + 10.09 + 2.022) \times 10^{-6} = \mathbf{13.102 \times 10^{-6}}$ **sec**

13.1 microseconds



Recursive way to calculate AMAT in multi level cache

1.2.8 Cache Memory: GATE CSE 1996 | Question: 26 top

https://gateoverflow.in/2778

A computer system has a three-level memory hierarchy, with access time and hit ratios as shown below:

Level 1 (Cache memory)Access time $= 50nsec/byte$

Size	Hit ratio
$8M$ bytes	0.80
$16M$ bytes	0.90
$64M$ bytes	0.95

Level 2 (Main memory)Access time $= 200nsec/byte$

Size	Hit ratio
$4M$ bytes	0.98
$16M$ bytes	0.99
$64M$ bytes	0.995

Level 3Access time $= 5\mu$sec/byte

Size	Hit ratio
$260M$ bytes	1.0

A. What should be the minimum sizes of level 1 and 2 memories to achieve an average access time of less than $100nsec$?

B. What is the average access time achieved using the chosen sizes of level 1 and level 2 memories?

For a set-associative Cache organization, the parameters are as follows:

t_c	Cache Access Time
t_m	Main memory access time
l	Number of sets
b	Block size
$k \times b$	Set size

Calculate the hit ratio for a loop executed 100 times where the size of the loop is $n \times b$, and $n = k \times m$ is a non-zero integer and $1 \leq m \leq l$.

Give the value of the hit ratio for $l = 1$.

In the relation $n = k \times m$

- k = lines (blocks) **per set**
- n = total **blocks touched by the loop**
- l = total **sets in the cache**
- m is chosen so that $1 \leq m \leq l$

Hence m **counts how many cache sets the loop's data occupy.**

Why this is the right interpretation:

- One set can hold exactly k blocks.
- If the loop needs n blocks and $n = k \times m$, it needs m whole sets' worth of space.
- The bound $m \leq l$ ensures the loop's footprint never references more sets than the cache actually has.
- When $l = 1$, m must be 1, meaning the entire loop fits in that single set—this is the special case you computed earlier.

So, practically, m = **number of distinct sets accessed by the loop.**

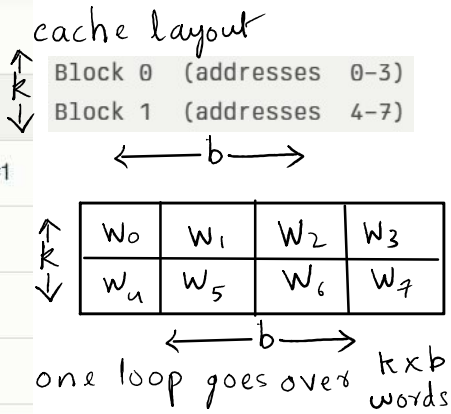
Illustrative walk-through (concrete numbers)

To keep everything small and visible, pick the following values that respect the given constraints $l = 1$ (one set) and $n = k$:

Parameter	Chosen value	Reason
Block size b	4 words	Small enough to list every access
Lines per set k	2	Gives one 2-line set (the whole cache)
Number of sets l	1	Matches the case we solved earlier
"Loop size" n	$k = 2$ blocks	So the loop touches exactly 2 blocks = 8 words
# iterations	100	Required by the problem

First pass inside the loop (iteration 1)

Access	Hit / miss	Action in cache	Comments
Word 0	**Miss**	Load *Block 0* into line 0.	Compulsory miss #1
Word 1	Hit	—	Still in Block 0
Word 2	Hit	—	
Word 3	Hit	—	
Word 4	**Miss**	Load *Block 1* into line 1.	Compulsory miss #2
Word 5	Hit	—	Still in Block 1
Word 6	Hit	—	
Word 7	Hit	—	



one loop goes over k×b words

every loop accesses same words in same order

total # access = # loops × K × b = 100×k×b

only in 1st iteration there misses, since we access iterating over same words and no eviction happened thus from next loop onwards all are Hits only

Hit ratio = $\frac{\#\text{times accessed word was found in cache } (N_H)}{\text{total no. of accesses } (N_A) \rightarrow 100 \times k \times b} = \frac{N_A - N_M}{N_A} = \frac{800-2}{800}$

so we got $1 - \frac{2}{800}$ i.e $1 - \frac{\#\text{misses}}{\#\text{access}} = 1 - \frac{N_M}{100 \times k \times b}$ here N_M was equal to k since all compulsory miss $\Rightarrow 1 - \frac{1}{100b}$ $100 \times 2 \times 4 = 800$

$= \frac{800}{800} - \frac{2}{800}$

$= 1 - \frac{1}{400}$

$2^5 \times 2^{10}$ B

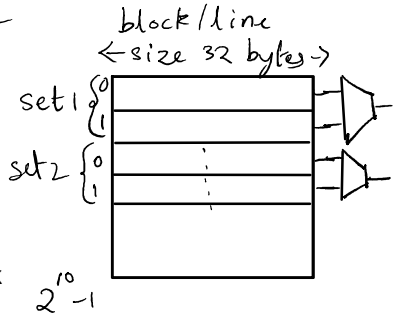
Consider two cache organizations. First one is 32 KB 2-way set associative with 32 byte block size, the second is of same size but direct mapped. The size of an address is 32 bits in both cases . A 2-to-1 multiplexer has latency of 0.6 ns while a k-bit comparator has latency of $\frac{k}{10}$ ns. The hit latency of the set associative organization is h_1 while that of direct mapped is h_2.

The value of h_1 is:

A. 2.4 ns
B. 2.3 ns
C. 1.8 ns
D. 1.7 ns

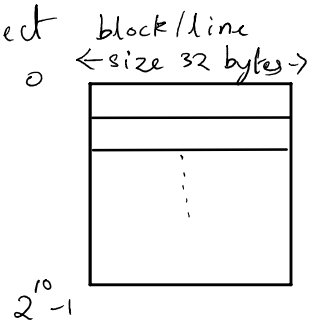
the set bits from memory address decides a set to look into, then its corresponding mux also gets enabled

Set



total size 32×2^{10} B

Direct



total size 32×2^{10} B

$\#sets = \frac{\#lines}{lines\ per\ set} = \frac{2^{10}}{2} = 2^9$

$S = \log_2(2^9) = 9$

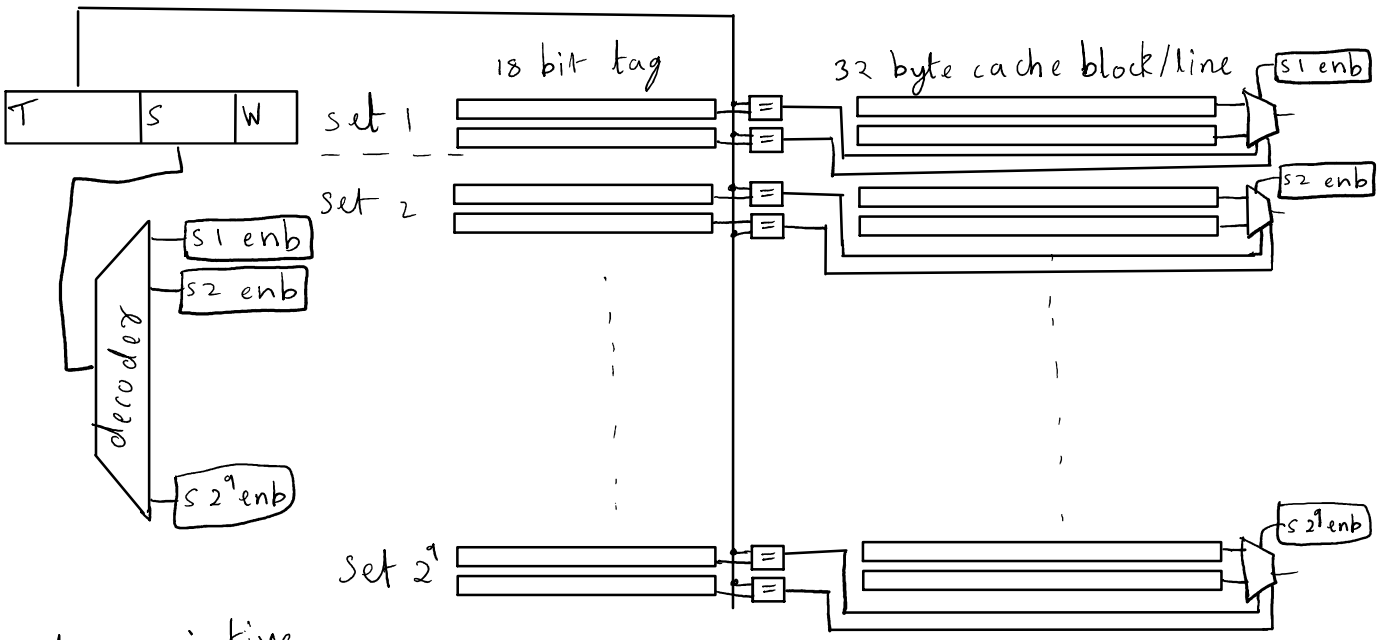
$W_s = \log_2(32) = 5$

$T_s = 32 - 9 - 5 = 18$

$W_d = 5$

$I = \log_2(\#lines) = \log_2(2^{10}) = 10$

$T_d = 32 - 10 - 5 = 17$



set associative comparator latency = $\frac{18}{10}$ ns = 1.8 ns

within a set combined result from all comparators acts as select bits for that set's Mux

another mux will select wanted word from matched block

way selection mux latency = 0.6 ns

hit latency = 1.8 + 0.6 = 2.4 ns

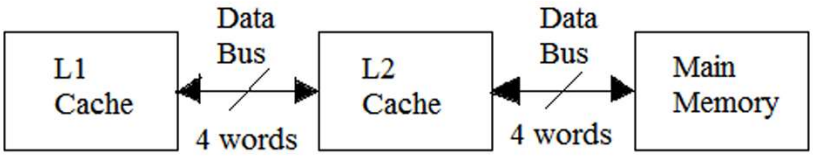
The value of h_2 is: in direct map there is no way selection mux

A. 2.4 *ns*
B. 2.3 *ns*
C. 1.8 *ns*
D. 1.7 *ns*

direct map comparator latency

$= \frac{17}{10} = 1.7$ ns

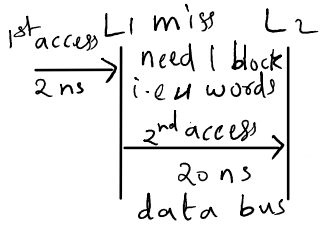
A computer system has an $L1$ cache, an $L2$ cache, and a main memory unit connected as shown below. The block size in $L1$ cache is 4 words. The block size in $L2$ cache is 16 words. The memory access times are 2 nanoseconds, 20 nanoseconds and 200 nanoseconds for $L1$ cache, $L2$ cache and the main memory unit respectively.



tests.gatecse.in

When there is a miss in $L1$ cache and a hit in $L2$ cache, a block is transferred from $L2$ cache to $L1$ cache. What is the time taken for this transfer?

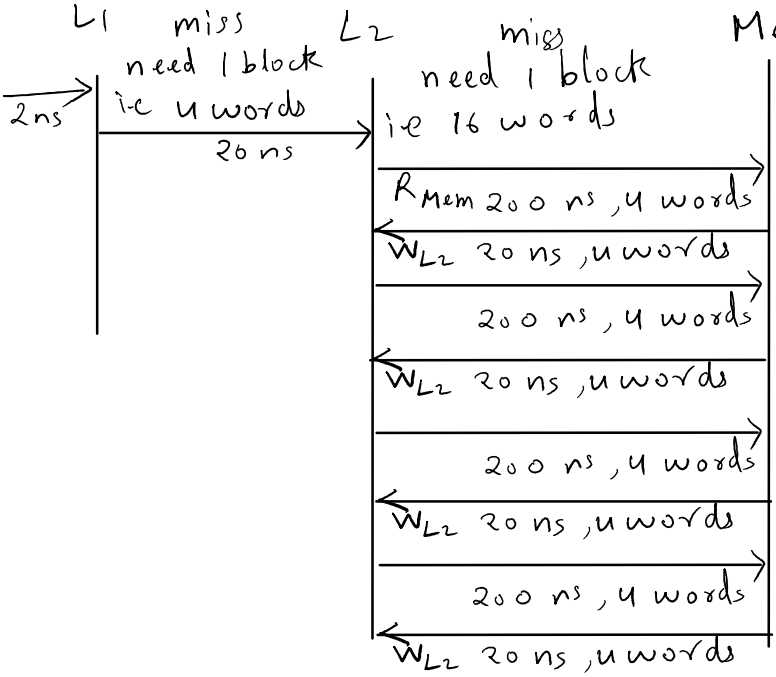
A. 2 nanoseconds
B. 20 nanoseconds
C. 22 nanoseconds
D. 88 nanoseconds



time taken = 2ns + 1 x 20ns

When there is a miss in both $L1$ cache and $L2$ cache, first a block is transferred from main memory to $L2$ cache, and then a block is transferred from $L2$ cache to $L1$ cache. What is the total time taken for these transfers?

A. 222 nanoseconds
B. 888 nanoseconds
C. 902 nanoseconds
D. 968 nanoseconds



data bus can transfer only 4 words at a time so need to read memory 4 times & write to L2 4 times

$4 \times (200 + 20)$

$= 4 \times 220$

$= 880$

total access time = CPU to L1 + L1 to L2 + L2 to Mem

$= 2 + 20 + 880$

$= 902$

1.2.38 Cache Memory: GATE CSE 2014 Set 3 | Question: 44 top

https://gateoverflow.in/2078

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

miss ratio is 0.1

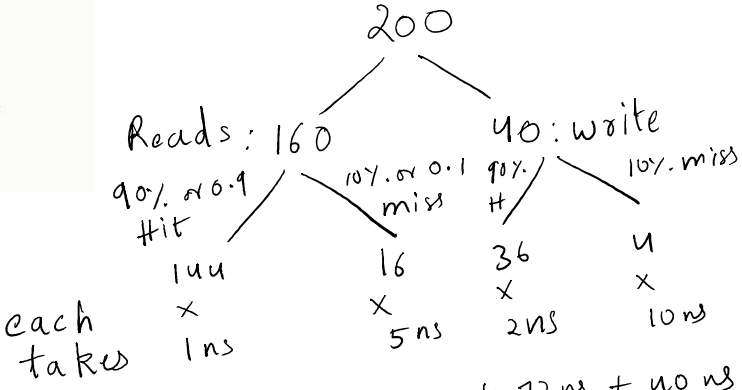
given

$R_H = 1$ ns $R_M = 5$ ns

$W_H = 2$ ns $W_M = 10$ ns

Hit ratio = 0.9

total number of access = 200



total access time = 144 ns + 80 ns + 72 ns + 40 ns

= 336 ns

$$\text{average access time} = \frac{\text{total access time}}{\text{total number of accesses}}$$

$$= \frac{336 \text{ ns}}{200} = 1.68 \text{ ns}$$

1.2.39 Cache Memory: GATE CSE 2015 Set 2 | Question: 24 top

https://gateoverflow.in/8119

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

$0.8 \times 5 \text{ ns} + 0.2 \times 50 \text{ ns} = 4 \text{ ns} + 10 \text{ ns} = 14 \text{ ns}$

1.2.43 Cache Memory: GATE CSE 2017 Set 1 | Question: 25 top

https://gateoverflow.in/118305

Consider a two-level cache hierarchy with $L1$ and $L2$ caches. An application incurs 1.4 memory accesses per instruction on average. For this application, the miss rate of $L1$ cache is 0.1; the $L2$ cache experiences, on average, 7 misses per 1000 instructions. The miss rate of $L2$ expressed correct to two decimal places is _____.

1 instruction — 1.4 memory access

1000 " — 1400 " "

∴ L2 miss rate = $\frac{7}{140}$

1400 memory access → L1 → 90% hit

→ 10% miss i.e 1400 × 0.1 = 140 miss → L2 → 7 miss out of 140

miss rate is in terms of memory access so convert # instruction into #memref

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

L_1 2M L_2 M

A. 0.111 and 0.056
B. 0.056 and 0.111
C. 0.0892 and 0.1784
D. 0.1784 and 0.0892

tests.gatecse.in goclasses.in tests.gatecse.ir

clock cycles is a measure of time

L_1 1 cc

L_2 8 cc $L_2 \rightarrow$ mem 18 cc

let ⇒ miss rate of L_2 be M

then miss rate of L_1 will be 2M

$$2\,cc = \underbrace{1\,cc}_{L_1\ access\ time} + \underbrace{2M}_{L_1\ miss\ rate} \times \left(\underbrace{8\,cc}_{L_2\ access\ time} + \underbrace{M}_{L_2\ miss\ rate} \times \underbrace{18\,cc}_{L_2 \rightarrow Mem\ miss\ penalty} \right)$$

in options B

if M = 0.111 then

2M = 0.222 not 0.056

$$2\,cc = 1\,cc + 2M \times 8\,cc + 2 \times M^2 \times 18\,cc$$

$$2\,cc = 1\,cc + 16M\,cc + 36M^2\,cc$$

$$0 = (36M^2 + 16M - 1)\,cc$$

1111 option C so eliminate both.

option A substitute M = 0.056

```
36*(sqr(0.056))+16*0.056-1
                0.008896000
```

↳ very close to 0

option D

```
36*(sqr(0.0892))+16*0.0892-1
                0.7136390
```

↳ not close to 0

∴ M = 0.056

2M = 0.111

option A ✓

Two ways to calculate AMAT

①

Hit ratio × cache access time + miss ratio × (cache access time + main memory time)

⇒ cache access time × (Hit ratio + miss ratio) + miss ratio × memory time i.e extra time upon miss

(Hit ratio + miss ratio = 1)

⇒ cache access time + miss ratio × Miss penalty ②

used this form since miss penalty info was given directly in question

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

since Hit ratio is given lets use Weighted form of AMAT instead of penalty AMAT

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

tests.gatecse.in gocl sses.in tests.gatecse.ir

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

↳ data

↳ updates only cache & not main memory

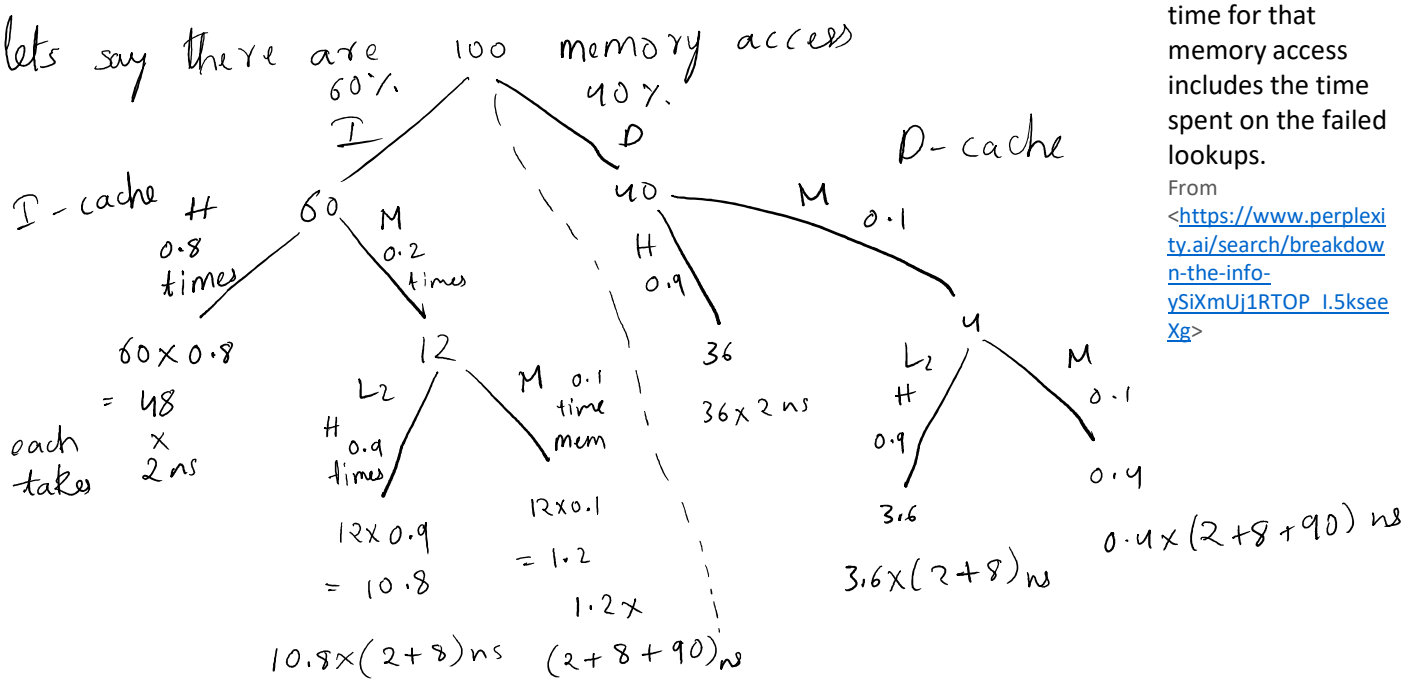
avg read access time
= Hit ratio x read access time +
Miss ratio x (read access time +
larger memory access time)

or

= read access time + Miss ratio x larger memory

referred word first policy is a technique where,
without "referred word first" policy, on a miss data word from memory to cache and then to processor, two sequential transfers
with "referred word first" policy, on a miss
1 data word from memory to processor
2 data word from memory to cache
1 and 2 happens simultaneously
From <https://www.perplexity.ai/search/in-caches-what-is-referred-wor-7afutkMrSYyaAxe0s0gv8Q>

When a cache miss occurs, the total time for that memory access includes the time spent on the failed lookups.
From <https://www.perplexity.ai/search/breakdown-the-info-ySiXmUj1RTOP_I.5kseeXg>



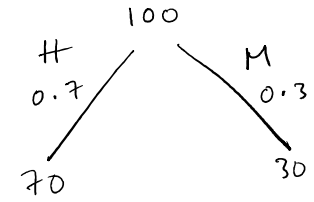
$$= \frac{48 \times 2ns + 10.8 \times 10.ns + 1.2 \times 100\,ns + 36 \times 2ns + 3.6 \times 10\,ns + 0.4 \times 100\,ns}{100}$$

$$= \frac{96 + 108 + 120 + 72 + 36 + 40}{100} \quad (180 \mid 300)$$

$$= \frac{300 + 172}{100} = \frac{472}{100} = 4.72\ ns$$

recap of AMAT

1 level

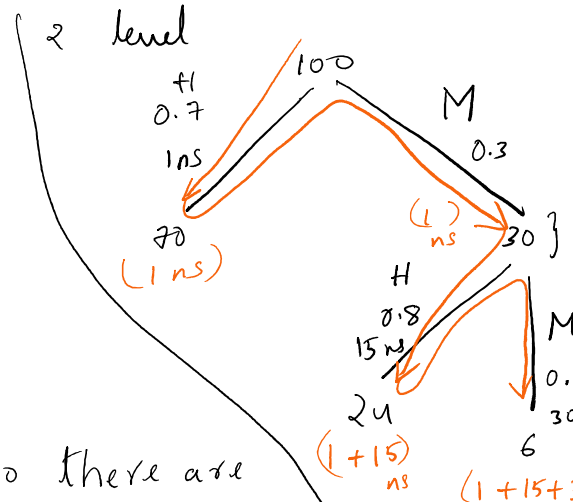


$$\frac{70 \times 1\,ns + 30 \times (1\,ns + 10\,ns)}{100}$$

access cache 1ns

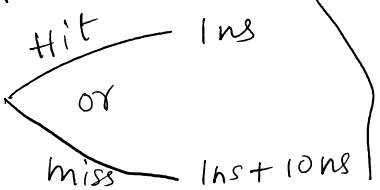
if Hit return to cpu else if miss go to next memory level i.e 10 ns

2 level



this is also a cache so Hit or miss can happen & no garuntee of finding data here & last one is main memory garuteed to find data.

so there are 2 paths



1.2.50 Cache Memory: GATE CSE 2019 | Question: 45

https://gateoverflow.in/302803

A certain processor deploys a single-level cache. The cache block size is 8 words and the word size is 4 bytes. The memory system uses a 60-MHz clock. To service a cache miss, the memory controller first takes 1 cycle to accept the starting address of the block, it then takes 3 cycles to fetch all the eight words of the block, and finally transmits the words of the requested block at the rate of 1 word per cycle. The maximum bandwidth for the memory system when the program running on the processor issues a series of read operations is _____$\times 10^6$ bytes/sec

60 MHz 60×10^6 Hz $\Rightarrow$ 60×10^6 clock cycles per 1 second

block 8 words, 1 word – 4 bytes

1 cycle to accept starting address

3 cycles to fetch a block i.e 8 words

& each word takes 1 cycles to transmit

no. of cycles to get & send a block

1 cycle + 3 cycle + 8 words each 1 cycle = 12 cycles

so in 12 cycle 1 block i.e 8 words ie 8x4 bytes 32 bytes are fetched & transmitted.

12 cycle – 32 bytes

60×10^6 cycles – ? bytes

$$\frac{60 \times 10^6}{12} = \frac{?}{32} \Rightarrow ? = 160 \times 10^6 \text{ bytes}$$

in 1 second 60×10^6 cycles happens

so in 1 second 160×10^6 bytes gets transferred

1.2.51 Cache Memory: GATE CSE 2020 | Question: 21 top

https://gateoverflow.in/333210

A direct mapped cache memory of 1 MB has a block size of 256 bytes. The cache has an access time of 3 ns and a hit rate of 94%. During a cache miss, it takes 20 ns to bring the first word of a block from the main memory, while each subsequent word takes 5 ns. The word size is 64 bits. The average memory access time in ns (round off to 1 decimal place) is_____.

word – 8 bytes

block - 256 bytes

block - $\frac{256}{8}$ words

line - block – 32 words

given, in a block

first word takes 20 ns

& remaining each word takes 5 ns

so in a block of 32 words first word takes 20 ns

remaining 31 takes 5 ns each

i.e $1 \times 20\,ns + 31 \times 5\,ns = 20 + 155 = 175\,ns$

to bring 1 block from memory it takes 175 ns

AMAT = $3\,ns + 0.06 \times 175\,ns = 3ns + 10.5\,ns = 13.5\,ns$

1.2.55 Cache Memory: GATE CSE 2021 Set 2 | Question: 27 top

https://gateoverflow.in/357513

Assume a two-level inclusive cache hierarchy, $L1$ and $L2$, where $L2$ is the larger of the two. Consider the following statements.

- S_1: Read misses in a write through $L1$ cache do not result in writebacks of dirty lines to the $L2$
- S_2: Write allocate policy *must* be used in conjunction with write through caches and no-write allocate policy is used with writeback caches.

Which of the following statements is correct?

A. S_1 is true and S_2 is false
B. S_1 is false and S_2 is true
C. S_1 is true and S_2 is true
D. S_1 is false and S_2 is false

A dirty line in cache is a block of cached data that has been modified by the CPU but not yet written back to main memory, marked by a dirty bit to ensure it's synchronized later for data consistency.

From <https://www.perplexity.ai/search/what-is-a-dirty-line-in-cache-EiwwztX6RbCAKXxC.n0otQ>

1. Cache Write Policies

These govern how data writes are handled between the cache and main memory [1] [2].

- **Write-Through**: Data is written to both cache and main memory at the same time for immediate consistency [1] [2] [3].
- **Write-Back** (also called Write-Behind): Data is written only to the cache initially, with main memory updated later (e.g., on eviction) for better performance [1] [2] [3].
- **Write-Around**: Writes bypass the cache and go directly to main memory, reducing cache pollution from rarely read data [1] [2].
- **Write-Allocate** (or Fetch on Write): On a write miss, the block is loaded into the cache before writing [2] [4].
- **No-Write-Allocate**: On a write miss, data is written directly to memory without loading it into the cache [2] [4].

Combinations:

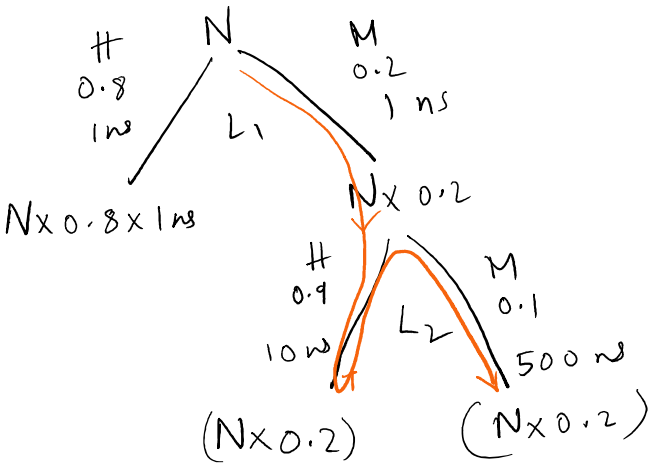
- Write-Through with Write-Allocate: Writes update both cache and memory and cache block is loaded on write miss.
- Write-Back with Write-Allocate: Writes update cache and defer memory update; cache block loaded on write miss.
- Write-Through with No-Write-Allocate: Writes update both cache and memory but block not loaded on write miss.
- Write-Back with No-Write-Allocate: Writes update cache, memory updated on eviction; write miss writes directly to memory without loading cache block.

Revisit

tests.gatecse.in gateclasses.in tests.gate

Consider a system with 2 level cache. Access times of Level 1 cache, Level 2 cache and main memory are 1 ns, 10 ns, and 500 ns respectively. The hit rates of Level 1 and Level 2 caches are 0.8 and 0.9, respectively. What is the average access time of the system ignoring the search time within the cache?

A. 13.0
B. 12.8
C. 12.6
D. 12.4



$$AMAT = \frac{N \times 0.8 \times 1\,ns + N \times 0.2 \times 0.9 (1+10)\,ns + N \times 0.2 \times 0.1 (1+10+500)\,ns}{N}$$

$$= 0.8\,ns + 0.18 (1+10)\,ns + 0.02 (1+10+500)\,ns$$

given to ignore search times

$$= 0.8\,ns + 1.8\,ns + 10\,ns$$

$$= 12.6\,ns$$

possibly due to Hardware components like wires, etc

A cache line is 64 bytes. The main memory has latency 32 ns and bandwidth 1 $GBytes/s$. The time required to fetch the entire cache line from the main memory is:

A. 32 ns
B. 64 ns
C. 96 ns
D. 128 ns

so it takes 64 ns to transfer 64 bytes + adding delay / latency

64 ns + 32 ns = 96 ns

1×10^9 bytes per second

10^9 byte per 10^9 ns

$\Rightarrow$ 1 byte – 1 ns

we need to transfer 64 byte

64 bytes – 64 ns

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

	Capacity	Mapping Method	Block Size
I-Cache	$4K$ words	Direct mapping	4 words
D-Cache	$4K$ words	2 -way set associative mapping	4 words
$L2$-Cache	$64K$ words	4-way set associative mapping	16 words

set — # lines — Word bits 2^b — T+I or T+S

- — 1K — 2 — 28
1K/2 — 1K — 2 — 28
4K/4 = 1K — 4K — 4 — 26

↳ every line will have a tag, after knowing # lines eliminate option C & D

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

I D L2

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

lets see at L2

T+S = 26 bits , #sets = 1K ie 1000 $= 2^{10}$ or 1024 to be exact

T+10 = 26

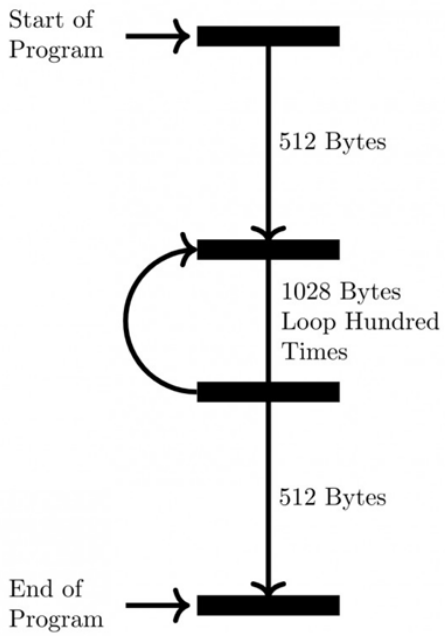
#set bits = S = 10

=) T = 16

option A

A certain computer system was designed with cache memory of size 1 Kbytes and main memory size of 256 Kbytes. The cache implementation was fully associative cache with 4 bytes per block. The CPU memory data path was 16 bits and the memory was 2−way interleaved. Each memory read request presents two 16−bit words. A program with the model shown below was run to evaluate the cache design.

tests.gatecse.in goclasses.in tests.gatecse.ir



Answer the following questions:

i. What is the hit ratio?
ii. Suggest a change in the program size of model to improve the hit ratio significantly.

in cache #blocks = #lines

A block-set associative cache memory consists of 128 blocks divided into four block sets. The main memory consists of 16, 384 blocks and each block contains 256 eight bit words.

1. How many bits are required for addressing the main memory? 22
2. How many bits are needed to represent the TAG, SET and WORD fields? 9 5 8

= 256 bytes

main memory
no. of blocks x no. of line per blocks
16384*256
4194304 → Total no of lines

logXbase2(4194304)
22
↓
no of bits required to address main memory

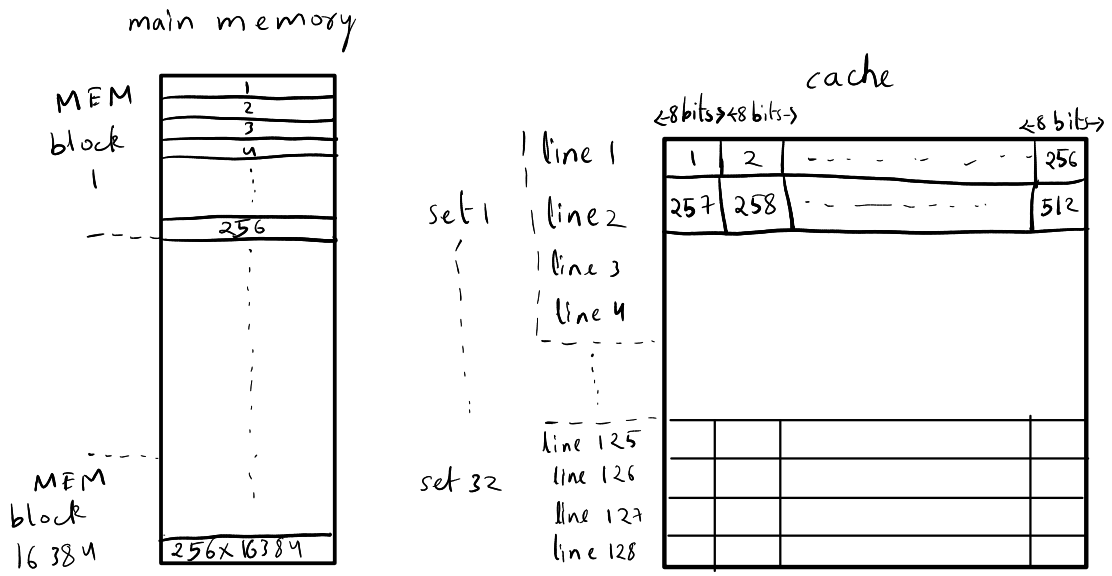
← 22 bits →
TAG SET WORD

no. of words = 256 ⇒ no. of bits required for word = $\log_2(256)$ = 8

four block sets = 4 line sets = 4 way
i.e in each set there are 4 lines

← 22 bits →
TAG SET WORD
← 14 → ← 8 →
← 9 → ← 5 →

set 0
set 1
128
? × 4 = 128
? = $\frac{128}{4}$ = 32
sets = 32 ⇒ # set bits = $\log_2(32)$ = 5



1.2.4 Cache Memory: GATE CSE 1992 | Question: 5-a

https://gateoverflow.in/584

The access times of the main memory and the Cache memory, in a computer system, are 500 n sec and 50 nsec, 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 and a write-through policy (where both main and cache memories are updated simultaneously) is used. Determine the average time of the main memory (in ns).

asking Average memory access time

access means both Read & Write

= time taken for certain no. of Read requests out of all requests
+ time taken for remaining no. of Write requests out of all requests

AMAT = (Percentage of Reads × Average Read Time) + (Percentage of Writes × Average Write Time)

cache read can Hit or Miss

80% read requests; hit ratio: in case of hit only read from cache; or miss ratio: in case of miss have to read from memory & update the same in cache; 20% write request; general access time to memory here its write

$= 0.8 \times [(0.9) \times 50 \text{ ns} + (0.1) \times (500 \text{ ns} + 50 \text{ ns})] + 0.2 \times 500 \text{ ns}$

$= 0.8 \times [(45 + 55) \text{ ns}] + 100 \text{ ns}$

$= (0.8 \times 100) \text{ ns} + 100 \text{ ns}$

$= 80 + 100 \text{ ns} = 180 \text{ ns}$

1.2.14 Cache Memory: GATE CSE 2004 | Question: 65

https://gateoverflow.in/1059

2-way = 2 lines per set

4 blocks = 4 lines

Consider a small two-way set-associative cache memory, consisting of four blocks. For choosing the block to be replaced, use the least recently used (LRU) scheme. The number of cache misses for the following sequence of block addresses is:

8, 12, 0, 12, 8.

A. 2
B. 3
C. 4
D. 5

t=1 find address 8 — 8 % 4 = 0 — set 0 → 8 — 8 not in set 0 ∴ miss & write — set 1

t=2 find 12 — 12 % 4 = 0 — miss & write — set 0: 8, → 12 — fill next empty space — set 1

t=3 find 0 — 0 % 4 = 0 — miss & write — replace 8, long ago used — set 0: → 0, 12 — set 1

t=4 find 12 — 12 % 4 = 0 — Hit — 12 is in set 0 — set 0: 0, 12 — set 1

t=5 find 8 — 8 % 4 = 0 — miss & write — replace 12 long ago used — set 0: 0, → 8 — set 1