Questions on ALL types of Cache

GATE PYQs on cache memory

- Questions based on Tag bits, Index Bits etc
- Questions based on loops and arrays ← today
- Questions based on effective time to access cache where multiple levels (L1, L2,..) caches are given
- 1 or 2 Questions based on Write policies – Write through, Write back
- 1 or 2 Questions based on virtual or physical indexing of cahces

Locality Example #1

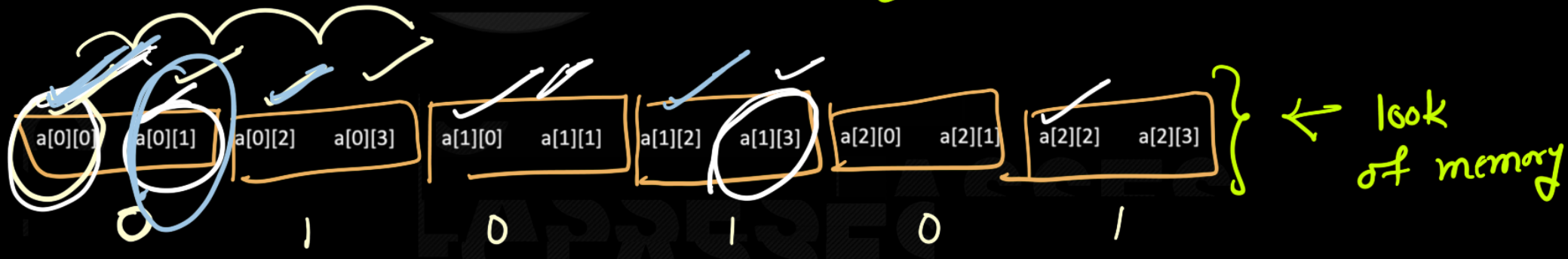
```
int sum_array_rows(int a[M][N])
{
    int i, j, sum = 0;

    for (i = 0; i < M; i++)
        for (j = 0; j < N; j++)
            sum += a[i][j];
    return sum;
}
```

spatial

a[1][2] a[1][3]

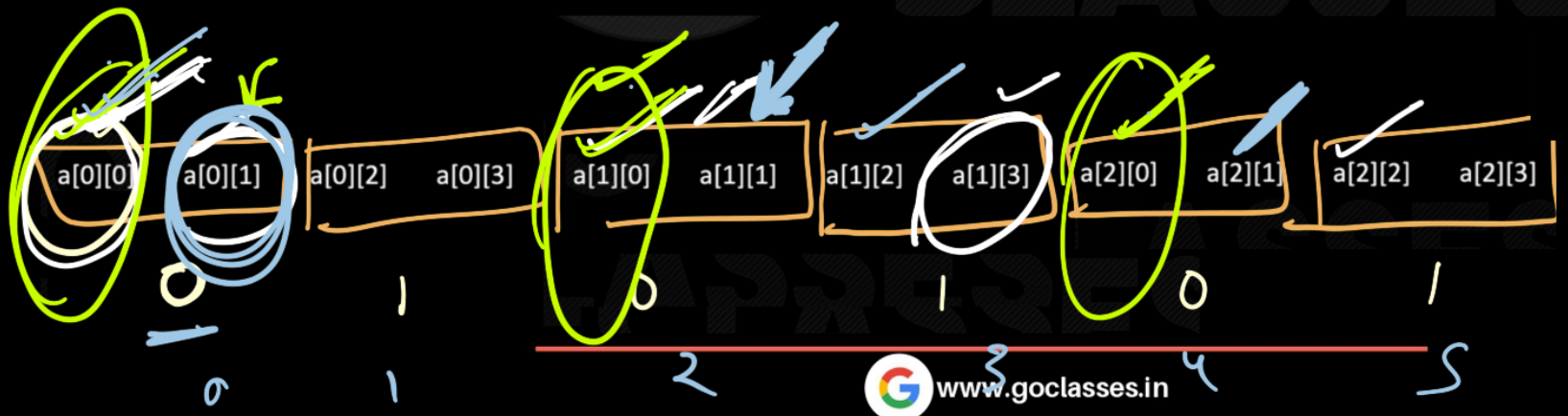
a[0][0] a[0][1]
a[1][2] a[1][3]
cache

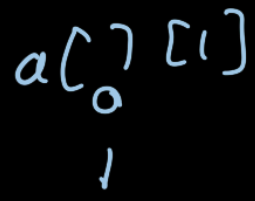
a[2][2] a[2][3]

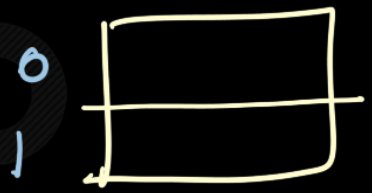


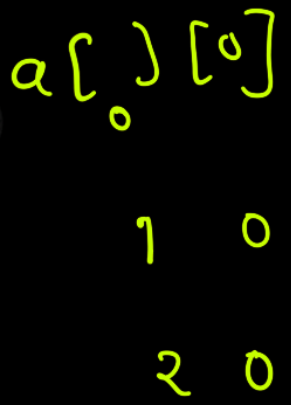
Locality Example #2

```
int sum_array_rows(int a[M][N])
{
    int i, j, sum = 0;

    for (i = 0; i < M; i++)
        for (j = 0; j < N; j++)
            sum += a[i][j];
    return sum;
}
```









Question:

$\frac{\#\ miss}{total\ access}$

What is the miss rate for a cache of size 512 bytes that is direct mapped and has 16-byte cache blocks.

```
int x[128];
int i;
int sum = 0;

for (i = 0; i < 128; i++) {
     sum += x[i];
}
```

CS = 2^9 Bytes = 2^5 lines

BS = 2^4 B = 2^2 elements

Assume sizeof(int) = 4.

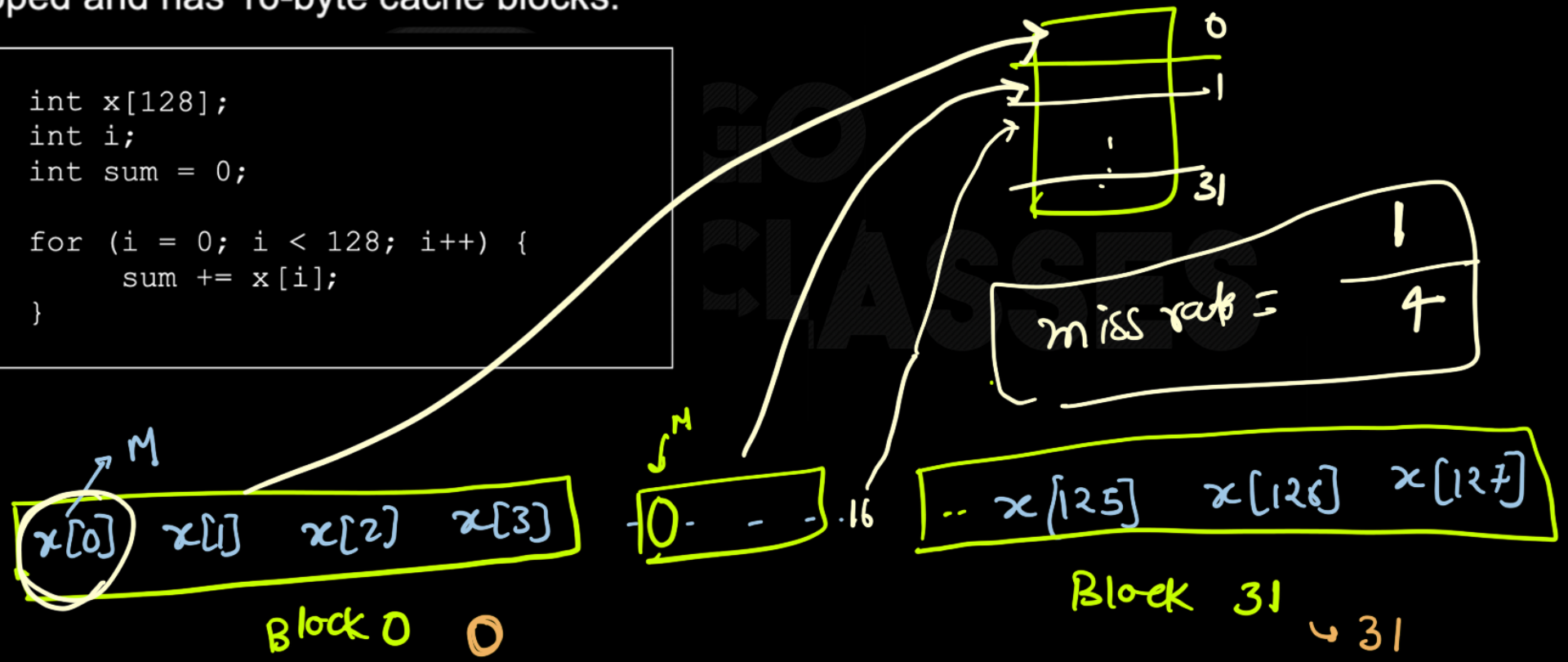
Array size = 2^7 elements = $2^7 \times 2^2 = 2^9$ B

Question:

$\frac{\#\ miss}{total\ access}$

What is the miss rate for a cache of size 512 bytes that is direct mapped and has 16-byte cache blocks.

```
int x[128];
int i;
int sum = 0;

for (i = 0; i < 128; i++) {
     sum += x[i];
}
```

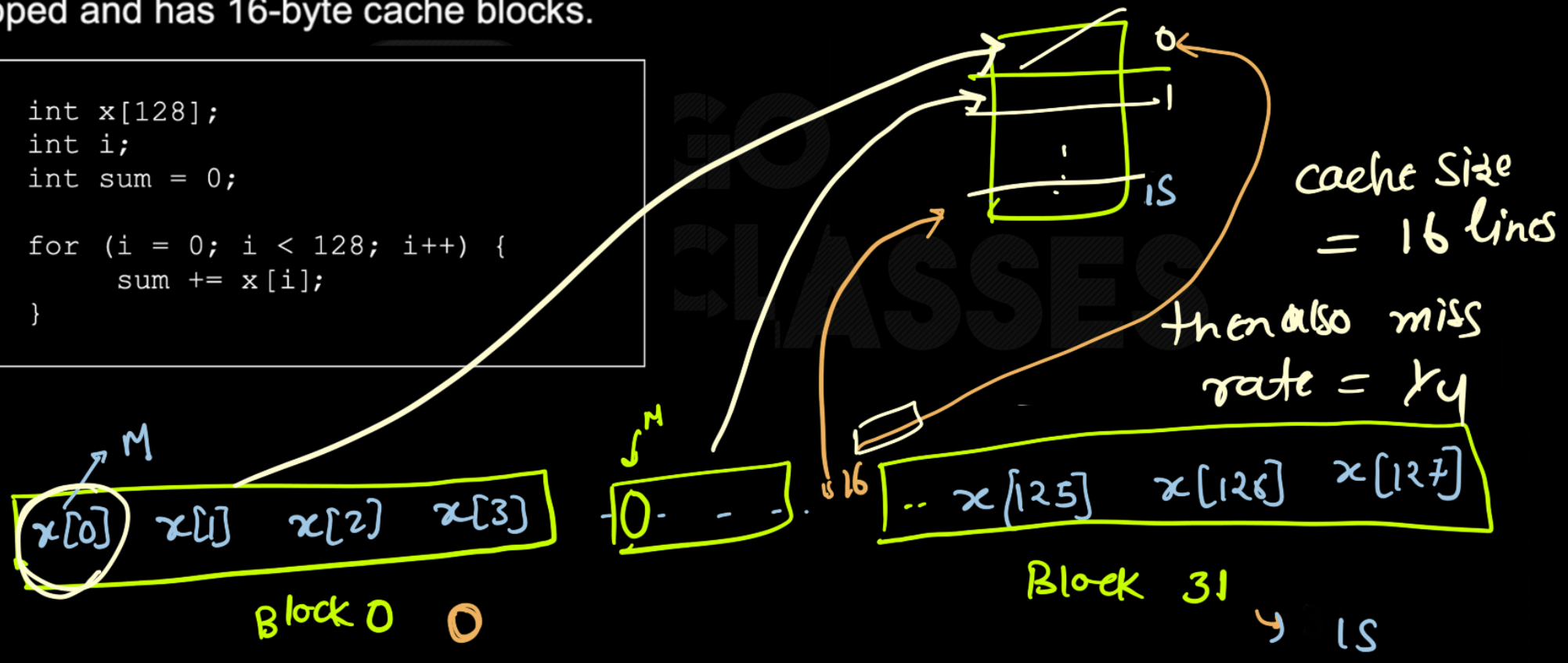


Question:

$\frac{\#\ miss}{total\ access}$

What is the miss rate for a cache of size 512 bytes that is direct mapped and has 16-byte cache blocks.

```
int x[128];
int i;
int sum = 0;

for (i = 0; i < 128; i++) {
     sum += x[i];
}
```

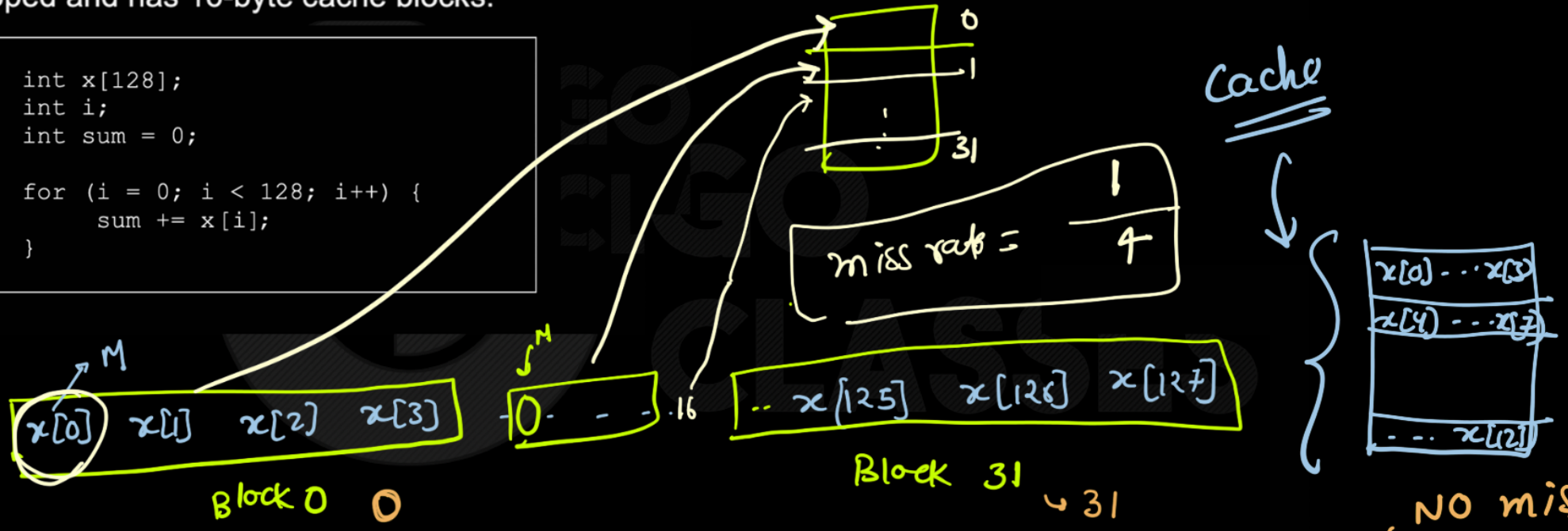


Question:

$\frac{\#\ miss}{total\ access}$

What is the miss rate for a cache of size 512 bytes that is direct mapped and has 16-byte cache blocks.

```
int x[128];
int i;
int sum = 0;

for (i = 0; i < 128; i++) {
     sum += x[i];
}
```



NO miss

↓

cache has complete array

miss rate on 2nd iteration of same loop = 0

Question:

In this problem, you will perform cache analysis for three code sequences. Assume a very small direct mapped 16 byte data cache with two cache lines. We assume `int` requires 4 bytes. Drawing the cache helps.

initially empty

For given code sequence, we assume a cold cache and that the array X is cache aligned (that is, $X[0]$ is loaded into the the beginning of the first cache line. All other variables are held in registers.

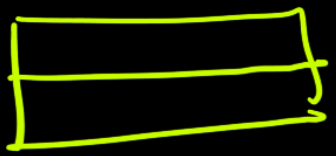
1 line = 8 bytes

Recall that miss rate is defined as $\frac{\#\text{misses}}{\#\text{accesses}}$.

CS = 16 bytes = 2 lines

Code :

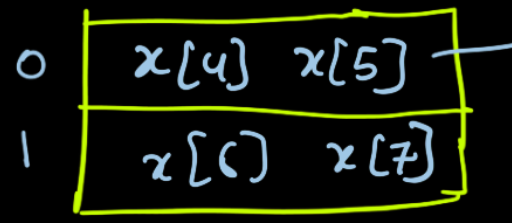
```
int X[8], t = 0;
for(int j = 0; j < 2; j++)
  for(int i = 0; i < 8; i++)
    t += X[i];
```



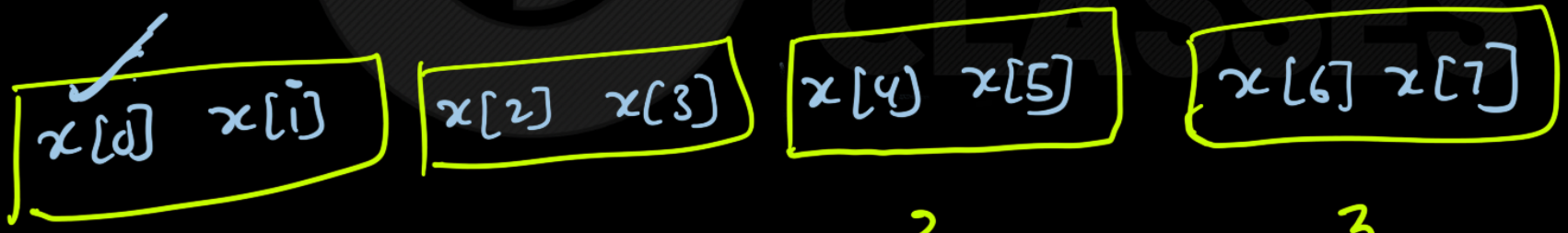
What is the number of cache misses ?

https://www.cs.cmu.edu/afs/cs/academic/class/15213-f09/www/oldexams/exam2-s09-v1.pdf

```
int X[8], t = 0;
for(int j = 0; j < 2; j++)
  for(int i = 0; i < 8; i++)
    t += X[i];
```



end of 1st iteration



Block No: 0 1 2 3

0 1

miss ratio = 1/2

Answer: 50%

https://www.cs.cmu.edu/afs/cs/academic/class/15213-f09/www/oldexams/exam2-s09-v1-sol.txt

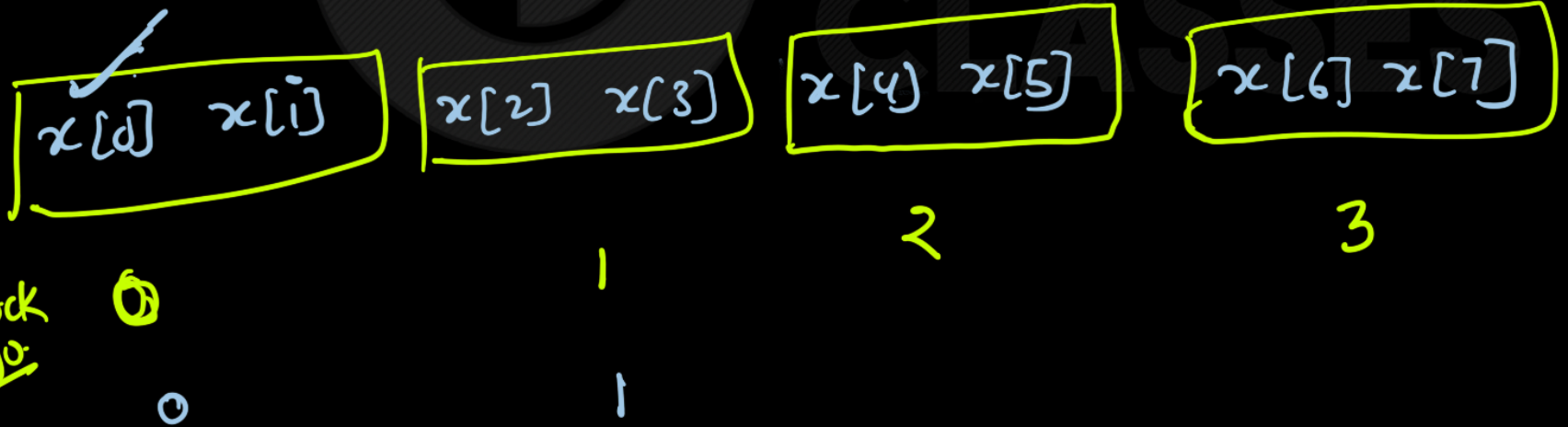
GO CLASSES

Variation

```
int X[8], t = 0;
for(int j = 0; j < 2; j++)
  for(int i = 0; i < 8; i++)
    t += X[i];
```

# of cache lines = 1

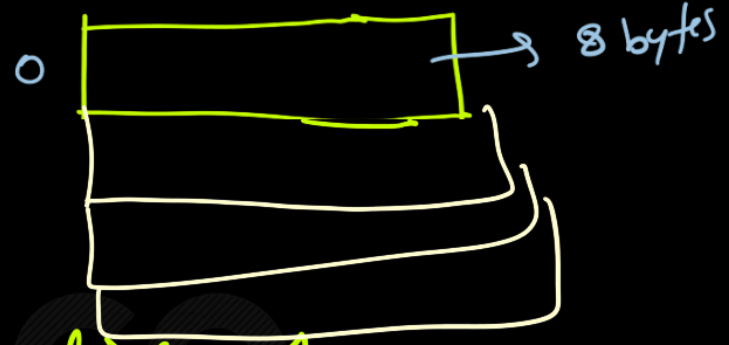
miss rate = $\frac{1}{2}$



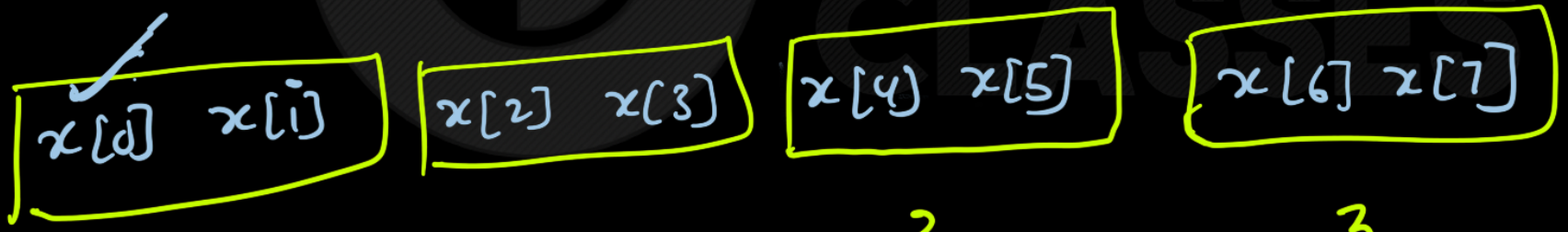
GO CLASSES

Variation -2

```
int X[8], t = 0;
for(int j = 0; j < 2; j++)
  for(int i = 0; i < 8; i++)
    t += X[i];
```



cache lines = 4



Block No.

0 1

miss ratio = $\frac{4}{16} = \frac{1}{4} = 25\%$

Question:

The following question deals with a different matrix, declared as `int arr[5][5]`. Again assume that `i`, `j`, and `sum` are all stored in registers.

Consider the following piece of code:

```
int i, j;
int sum = 0;

  for(i=0; i<5; i++){
    for(j=0; j<5; j++){
      sum += arr[i][j];
    }
  }
```

https://www.cs.cmu.edu/afs/cs/academic/class/15213-f09/www/oldexams/final-f06.pdf

For each of the following caches, specify the total number of **cache misses** for the above code. **Important:** Assume that the matrix is aligned so that `arr[0][0]` is the first element in a cache block.

i. Direct-mapped, 16 byte block-size, 4 blocks

Number of cache misses _____

ii. 2-way set associative, 8 byte block-size, 2 sets

Number of cache misses _____

```
int i, j;
int sum = 0;

  for(i=0; i<5; i++){
    for(j=0; j<5; j++){
      sum += arr[i][j];
    }
  }
```

For each of the following caches, specify the total number of **cache misses** for the above code. **Important:** Assume that the matrix is aligned so that `arr[0][0]` is the first element in a cache block.

i. Direct-mapped, 16 byte block-size, 4 blocks

Number of cache misses _____

```
int i, j;
int sum = 0;

for(i=0; i<5; i++){
  for(j=0; j<5; j++){
    sum += arr[i][j];
  }
}
```

1 Block = 4 elements

a[0][0] a[0][1] a[0][2] a[0][3] a[0][4] a[1][0] ---

6 + 1 = 7 misses

a[4][4] . . . .

New array

For each of the following caches, specify the total number of **cache misses** for the above code. **Important:** Assume that the matrix is aligned so that `arr[0][0]` is the first element in a cache block.

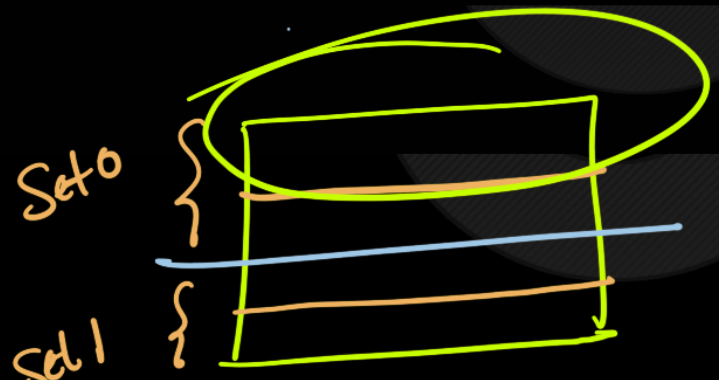
i. Direct-mapped, 16 byte block-size, 4 blocks

Number of cache misses _____

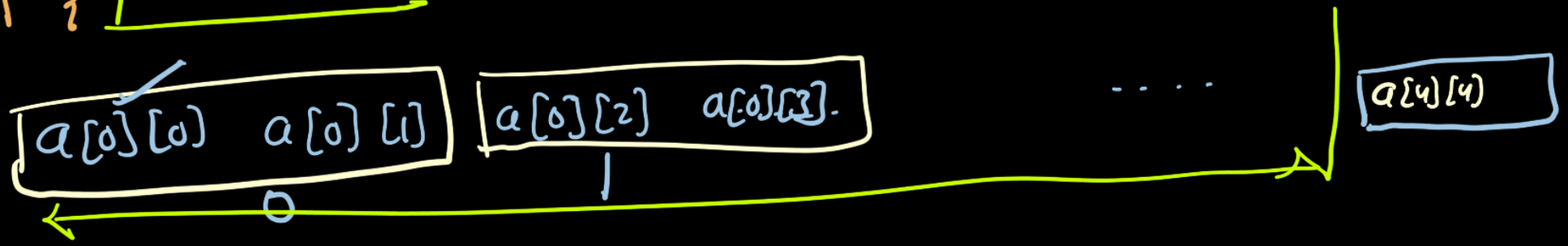
ii. 2-way set associative, 8 byte block-size, 2 sets

Number of cache misses _____

```
int i, j;
int sum = 0;

  for(i=0; i<5; i++){
    for(j=0; j<5; j++){
      sum += arr[i][j];
    }
  }
```



12 + 1 = 13 misses



Answer:

(a) (i) 7 misses, (ii) 13 misses

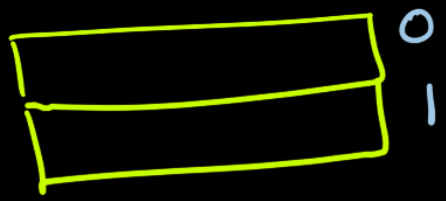
Question b:

```
for(k=0; k<ITERATIONS; k++){
  for(i=0; i<5; i++){
    for(j=0; j<5; j++){
      sum += arr[i][j];
    }
  }
}
```

(2 points) If ITERATIONS is 2 (Total accesses: 50).

i. Direct-mapped, 64 byte block-size, 2 line

Number of cache misses _____



1 + 1 = 2

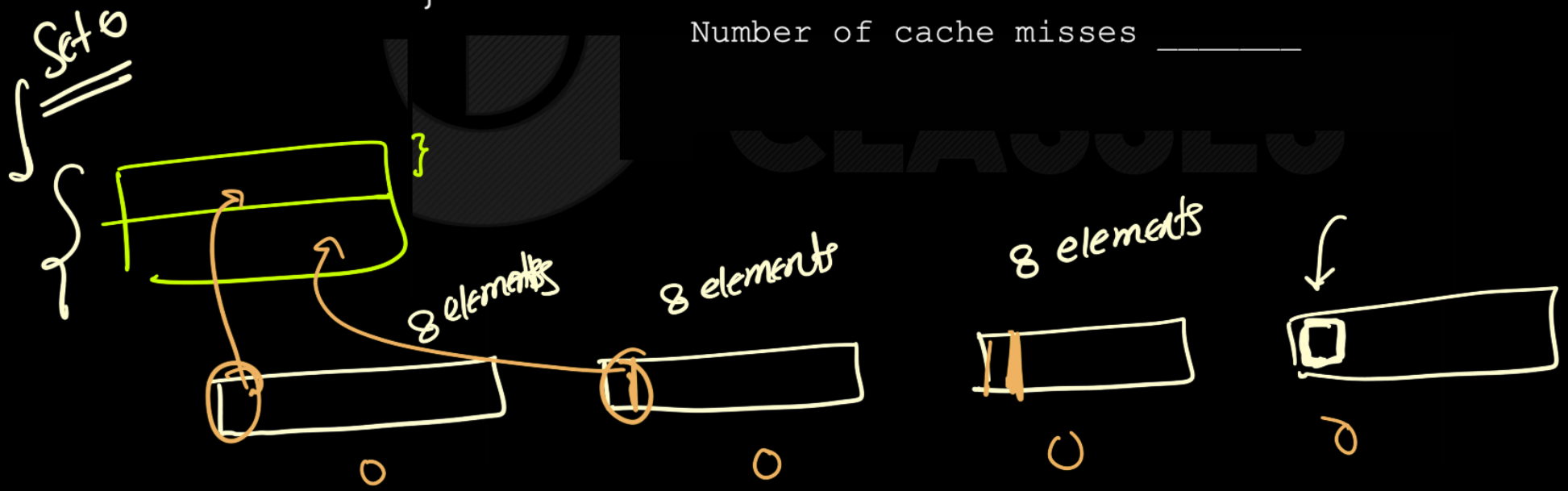
Question b:

```
for(k=0; k<ITERATIONS; k++){
   for(i=0; i<5; i++){
      for(j=0; j<5; j++){
         sum += arr[i][j];
      }
   }
}
```

4 miss (1st iteration) + 4 miss (2nd it.)

ii. 2-way set associative, 32 byte block-size, 1 set

Number of cache misses _____



Answer:

(b) (i) 2 misses, (ii) 8 misses

Question:

This problem evaluates the cache performances for different loop orderings. You are asked to consider the following two loops, written in C, which calculate the sum of the entries in a 128 by 64 matrix of 32-bit integers:

Loop A	*Loop B*

`sum = 0;`
`for (i = 0; i < 128; i++)`
`  for (j = 0; j < 64; j++)`
`  sum += A[i][j];`
	`sum = 0;`
	`for (j = 0; j < 64; j++)`
	`  for (i = 0; i < 128; i++)`
	`  sum += A[i][j];`

$\rightarrow 2^{12}B$ $\rightarrow 2^{5}B$

Consider a 4KB direct-mapped data cache with 32-byte cache lines.

$CS = 2^{12}\ B$

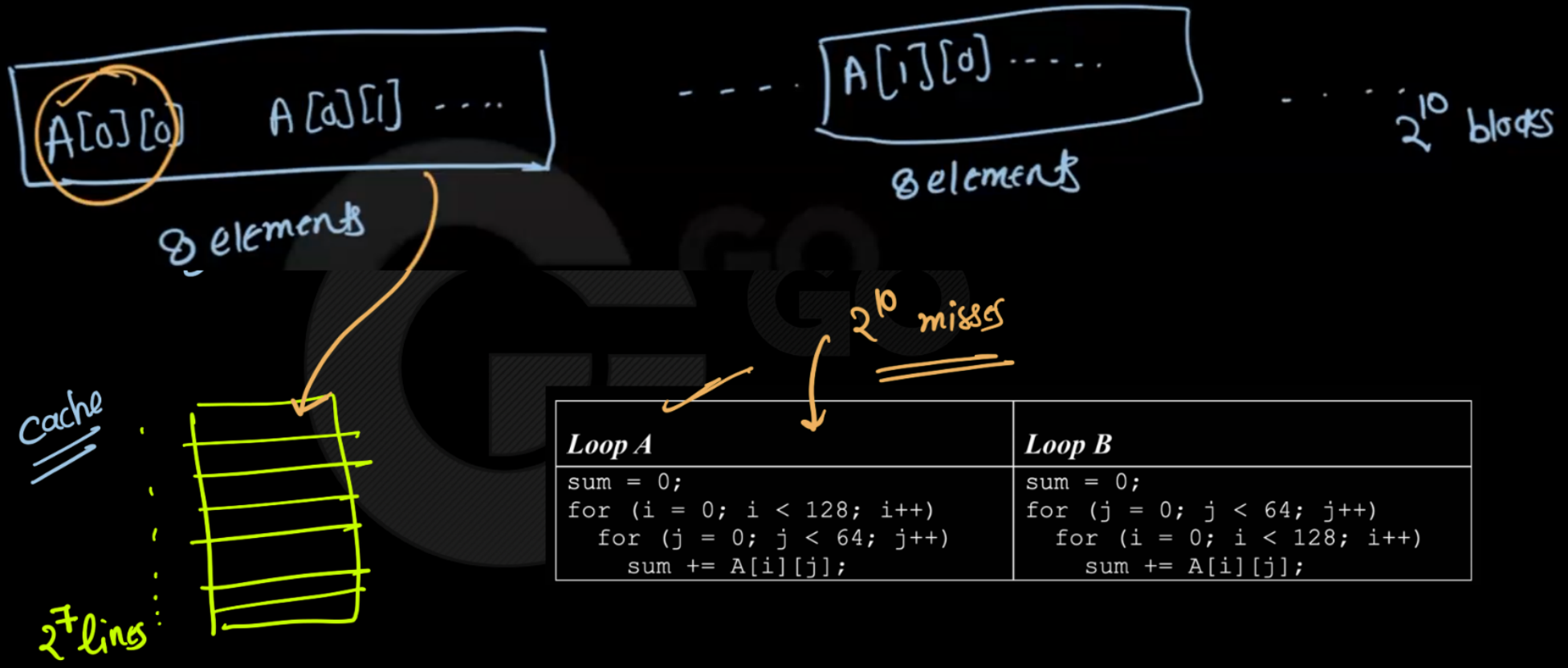
$LS = 2^{5}\ B$

$= 2^{3}$ elem.

The number of cache misses for Loop A:_____

The number of cache misses for Loop B:_____

http://csg.csail.mit.edu/6.823S15/StudyMaterials/pset2.pdf



Loop A	*Loop B*

`sum = 0;`
`for (i = 0; i < 128; i++)`
`  for (j = 0; j < 64; j++)`
`  sum += A[i][j];`
	`sum = 0;`
	`for (j = 0; j < 64; j++)`
	`  for (i = 0; i < 128; i++)`
	`  sum += A[i][j];`

Loop B

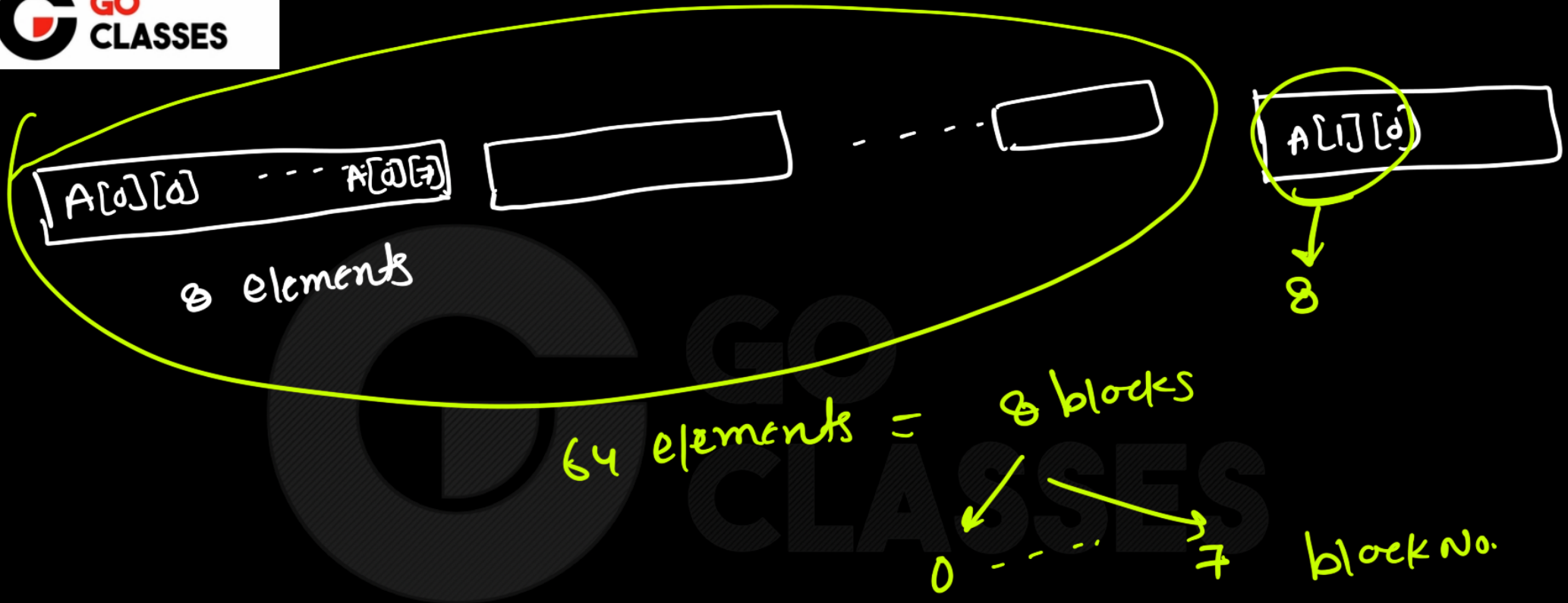
```
sum = 0;
for (j = 0; j < 64; j++)
  for (i = 0; i < 128; i++)
    sum += A[i][j];
```

A[0] [0]

A[1] [0]

A[2] [0]

A[3] [0]



Access pattern of block Nos = 0, 8, 16, 24, 32, ...

GO CLASSES

Access pattern of block nos: 0, 8, 16, 24, 32, 40, 48, 56, 64, ... 120, 128, 136, ...

120

Line nos ⇒ 0 8 16

Block No

$120/8 = 15$

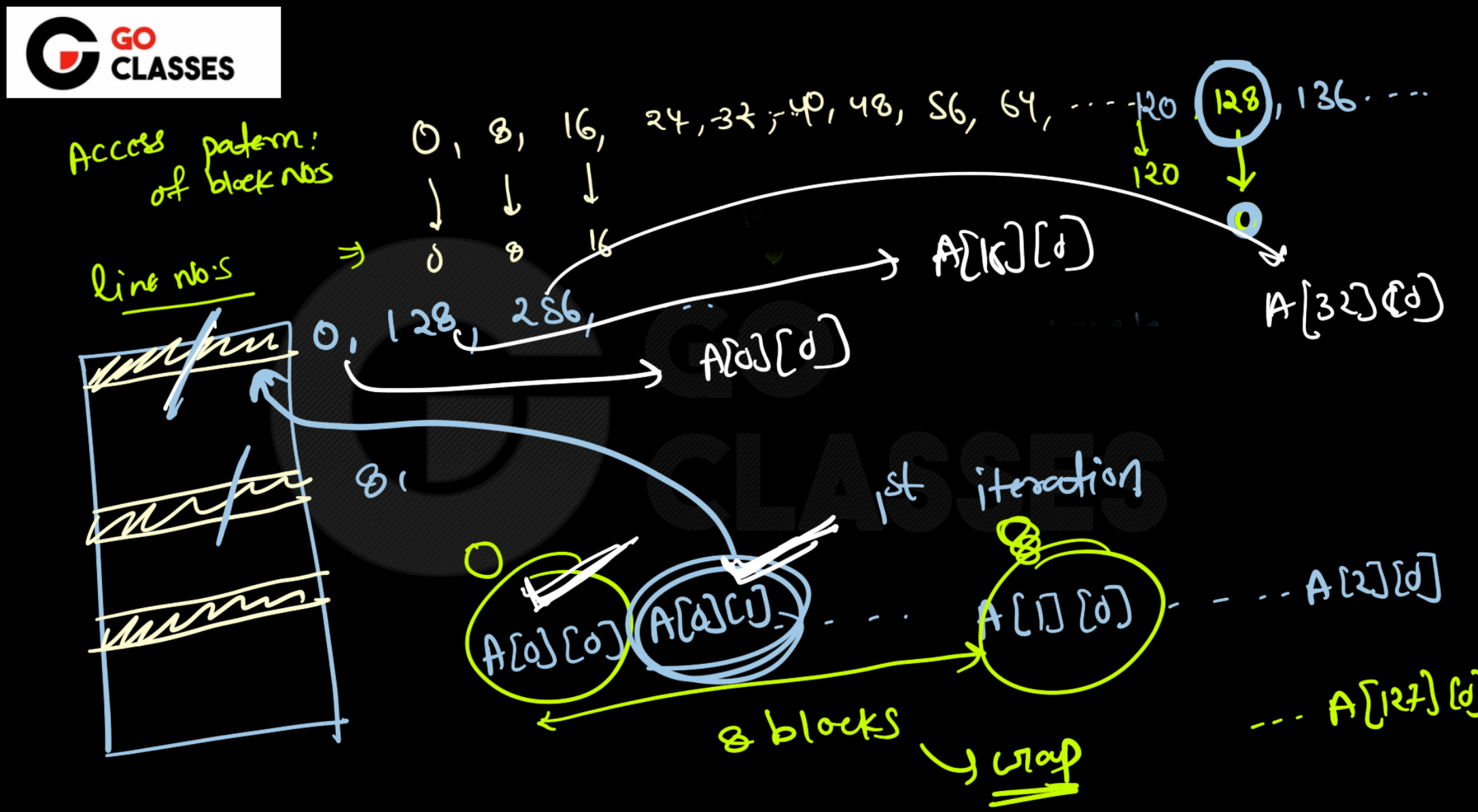
Not going to access this

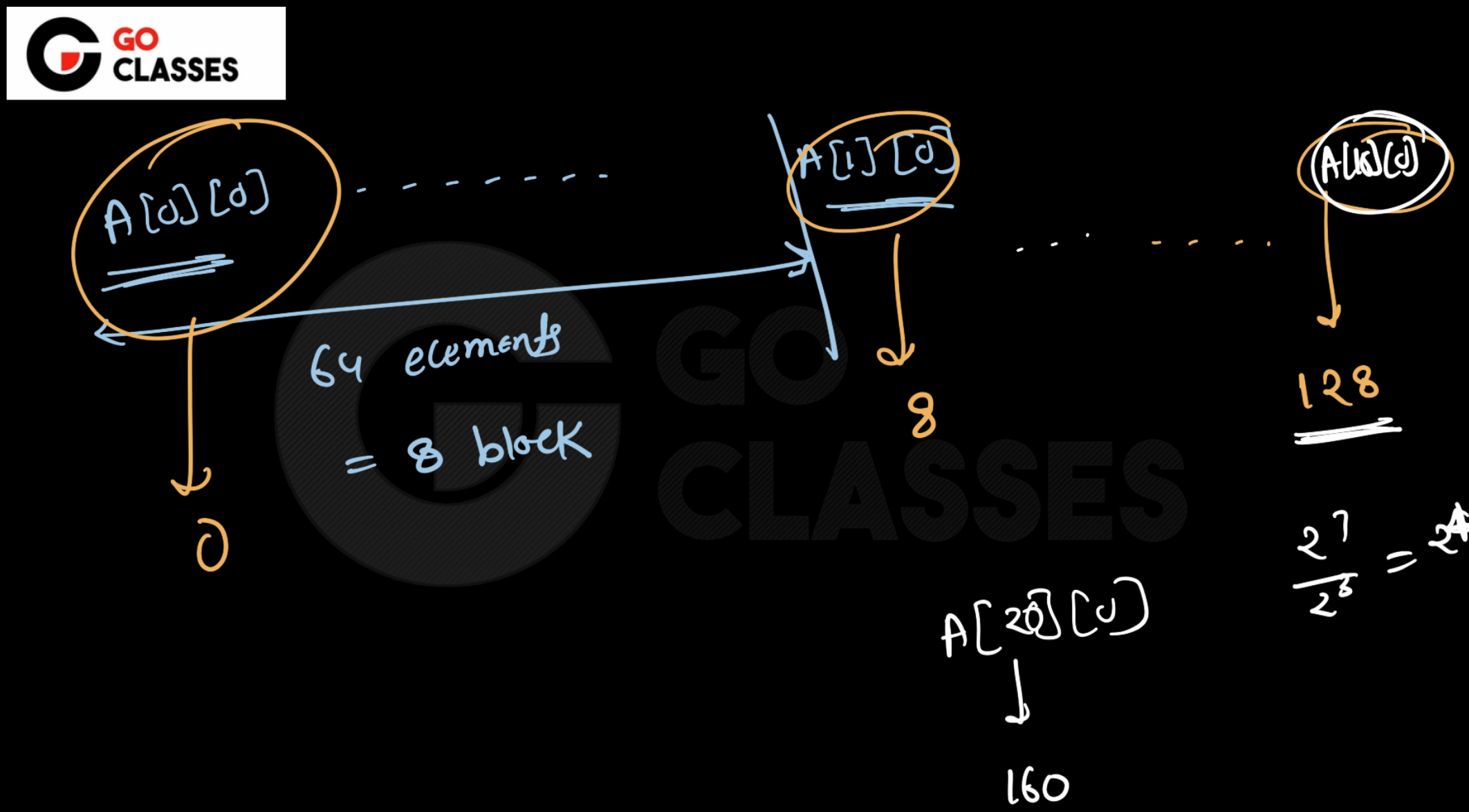
$132/8$

$136/8 = 17$

$= 1024$ Bn

2^{10} blocks

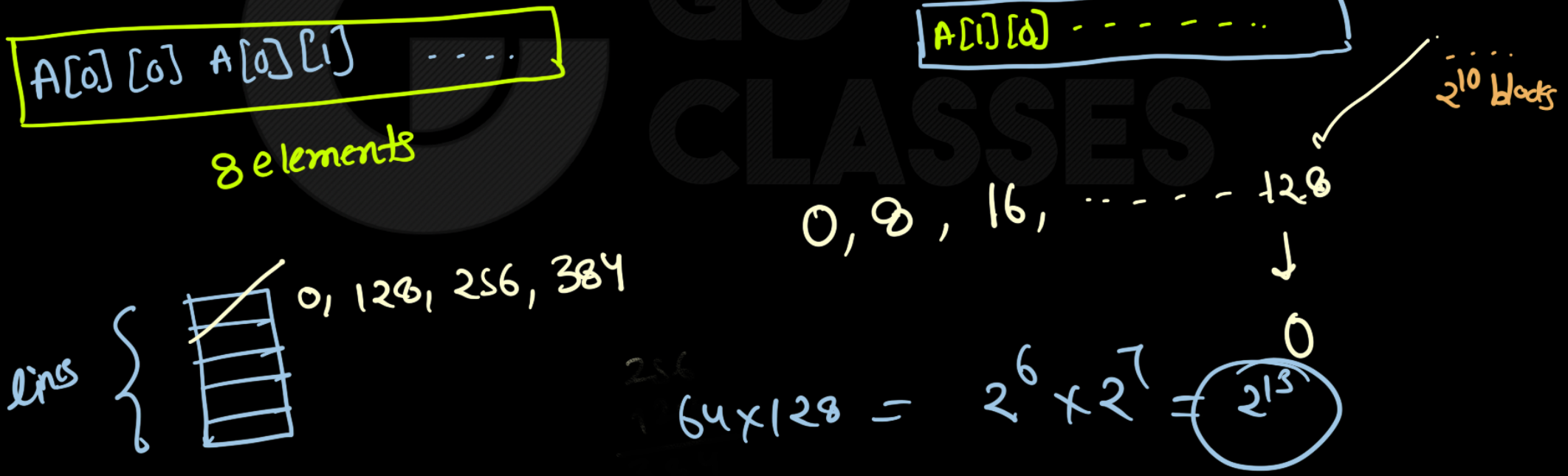

GO CLASSES
Access pattern of block nos
0, 8, 16, 24, 32, 40, 48, 56, 64, ... 120, 128, 136, ...
120
0
A[16][0]
A[32][0]
Line nos
0, 128, 256, ...
A[0][0]
8
1st iteration
A[0][0]
A[0][1]
A[1][0]
A[2][0]
8 blocks
wrap
A[127][0]


A[0][0]
A[1][0]
A[k][d]
64 elements
= 8 block
0
8
128
$\frac{2^7}{2^3} = 2^4$
A[20][0]
160

GO CLASSES

Loop B

```
sum = 0;
for (j = 0; j < 64; j++)
  for (i = 0; i < 128; i++)
    sum += A[i][j];
```



sum += A[i][j];

GO CLASSES

A[0][0] A[0][1] - A[0][7]

8 elements

A[1][0] - - - - - - -

2^{10} blocks

0, 8, 16, - - - - 128

↓

0

Access pattern of MM block in

1st iteration : 0, 8, 16, 24, - - -

2nd iteration : 0, 8, - - - - - - -

8th item 0, 8, 16, - - -

9th item 1, 9, 17, - - -

Answer:

Each element of the matrix can only be mapped to a particular cache location because the cache here is a Direct-mapped data cache. *Matrix A* has 64 columns and 128 rows. Since each row of matrix has 64 32-bit integers and each cache line can hold 8 words, each row of the matrix fits exactly into eight (64÷8) cache lines as the following:

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

Loop A accesses memory sequentially (each iteration of *Loop A* sums a row in *matrix* A), an access to a word that maps to the first word in a cache line will miss but the next seven accesses will hit. Therefore, *Loop A* will only have compulsory misses (128×64÷8 or 1024 misses).

The consecutive accesses in *Loop B* will use every eighth cache line (each iteration of *Loop B* sums a column in *matrix* A). Fitting one column of matrix A, we would need 128×8 or 1024 cache lines. However, our 4KB data cache with 32B cache line only has 128 cache lines. When Loop B accesses a column, all the data that the previous iteration might have brought in would have already been evicted. Thus, every access will cause a cache miss (64×128 or 8192 misses).

The number of cache misses for Loop A: 1024

The number of cache misses for Loop B: 8192

2^{13} elements = misses

http://csg.csail.mit.edu/6.823S15/StudyMaterials/pset2_sol.pdf

Question:

8. Caches - High-level Analysis

You are given the following code to analyze:

```
int x[2][128];
int i;
int sum = 0;
for (i = 0; i < 128; i++) {
    sum += x[0][i] * x[1][i];
}
```

H.W.

Assume we execute this under the following conditions:

- `sizeof(int) = 4`.
- Array x starts at **memory address 0x0** and is stored in **row-major** order.
- In each case below, the **cache is initially empty**.
- The only memory accesses are to the entries of the array x. All other variables are stored in registers.
- Associative caches use **Least Recently Used (LRU)** replacement policy.

Given these assumptions, estimate the **miss rates** for the following cases:

https://pages.cs.wisc.edu/~gerald/cs354/Spring18/exams/final/CS354FinalExamKey.pdf

Cache Type	Total cache size (C)	Cache block size (B)	Miss rate (%)
Direct-mapped	512 bytes	16 bytes	
Direct-mapped	1024 bytes	16 bytes	
2-way set associative	512 bytes	16 bytes	
2-way set associative	1024 bytes	16 bytes	
2-way set associative	512 bytes	32 bytes	
2-way set associative	256 bytes	64 bytes	

H.W.

```
int x[2][128];
int i;
int sum = 0;
for (i = 0; i < 128; i++) {
    sum += x[0][i] * x[1][i];
}
```

x[0][i]
$2^7 \times 2^2 = 2^9$

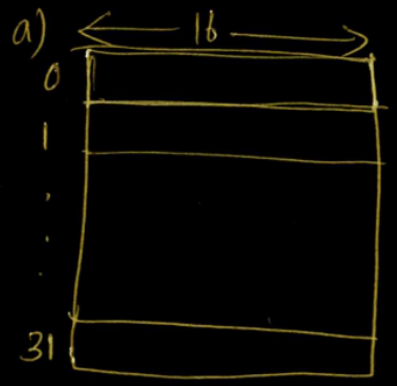
size of $x_{0,0} - x_{0,127}$
= 512 bytes

$\Rightarrow x_{00} - x_{03}$ and $x_{10} - x_{13}$ map to the same set in direct mapped cache of 512 bytes.

8. Caches - High-level Analysis

(a) You are given the following code to analyze:

```
int x[2][128];
int i;
int sum = 0;
for (i = 0; i < 128; i++) {
    sum += x[0][i] * x[1][i];
}
```



Assume we execute this under the following conditions:

- `sizeof(int) = 4`.
- Array x starts at **memory address 0x0** and is stored in **row-major** order.
- In each case below, the **cache is initially empty**.
- The only memory accesses are to the entries of the array x. All other variables are stored in registers.
- Associative caches use **Least Recently Used (LRU)** replacement policy.

b) 64 sets ⇒ whole array x can fit in cache ∴ only cold miss

Given these assumptions, estimate the **miss rates** for the following cases:

	Cache Type	Total cache size (C)	Cache block size (B)	Miss rate (%)
S = 32	a) Direct-mapped	512 bytes 2^9	16 bytes 2^4	100%
S = 64	b) Direct-mapped	1024 bytes	16 bytes	25%
S = 16	c) 2-way set associative	512 bytes	16 bytes	25%
S = 32	d) 2-way set associative	1024 bytes	16 bytes	25%
	e) 2-way set associative	512 bytes	32 bytes	12.5%
	f) 2-way set associative	256 bytes	64 bytes	6.125%

6

45

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];
    }
}
```

P2:

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

$P1$ 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

56

Best answer

Code being **C** implies array layout is row-major.

http://en.wikipedia.org/wiki/Row-major_order

When $A[0][0]$ is fetched, 128 consecutive bytes are moved to cache. So, for the next $\frac{128}{8} - 1 = 15$ memory references there won't be a cache miss.

For the next iteration of i loop also the same thing happens as there is no temporal locality in the code. So, number of cache misses for $P1$ is

$$= \frac{512}{16} \times 512$$

$$= 32 \times 512$$

$$= 2^{14} = 16384$$

Correct Answer: C

answered Nov 7, 2014 • edited May 16, 2019 by **Naveen Kumar 3**

edit flag hide comment Follow

Pip Box Delete with Reason Wrong Useful

share this

Arjun

GATE CSE 2006 | Question: 81

$P1$ 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 $M1$ and that for $P2$ be $M2$.

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

A. 0
B. $\frac{1}{16}$
C. $\frac{1}{8}$
D. 16

https://gateoverflow.in/43517/gate-cse-2006-question-81

49

Best answer

$$\text{Number of Cache Lines} = \frac{2^{15}B}{128B} = 256$$

$$\text{In 1 Cache Line} = \frac{128B}{8B} = 16\ elements$$

$$P_1 = \frac{\text{total elements in array}}{\text{elements in a cache line}}$$

$$= \frac{512 \times 512}{16} = 2^{14} = 16384.$$

$$P_2 = 512 \times 512 = 2^{18}$$

$$\frac{P_1}{P_2} = \frac{16384}{512 \times 512}$$

$$= 2^{14-18} = 2^{-4} = \frac{1}{16}$$

It is so, because for P_1 for every line there is a miss, and once a miss is processed we get 16 elements in memory.
So, another miss happens after 16 elements.
For P_2 for every element there is a miss because storage is row major order(by default) and we are accessing column wise.

Hence, answer is option B.

answered Apr 29, 2016 • edited Dec 25, 2017 by **pavan singh**

edit flag hide comment Follow

Pip Box Delete with Reason Wrong Useful

amarVashishth

GATE CSE 2007 | Question: 80

85

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

https://gateoverflow.in/1273/gate-cse-2007-question-80

0
1
2
3
4
5
6
7
8
9
10
11
12
⋮
30
31

279

Bits used to represent the address = $\log_2 2^{16} = 16$

Each cache line size $= 64$ bytes; means offset requires 6-bits

Total number of lines in cache $= 32$; means line # requires 5-bits

So, tag bits $= 16 - 6 - 5 = 5$

Best answer

We have a 2D-array each of its element is of size $= 1$ Byte;
Total size of this array $= 50 \times 50 \times 1$ Byte $= 2500$ Bytes

So, total number of lines it will require to get contain in cache

$$= \frac{2500B}{64B} = 39.0625 \approx 40$$

Starting address of array $= 1100H = 00010\ 00100\ 000000$
The group of bits in middle represents Cache Line number $\implies$ array starts from cache line number 4,
We require 40 cache lines to hold all array elements, but we have only 32 cache lines

Lets group/partition our 2500 array elements in those 40 array lines, we call this first array line as A_0 which will have 64 of its elements.

This line(group of 64 elements) of array will be mapped to cache line number 4 as found by analysis of starting address of array above.

This all means that among those 40 array lines some array lines will be mapped to same cache line, coz there are just 32 cache lines but 40 of array lines.

This is how mapping is:

0	A_{28}	
1	A_{29}	
2	A_{30}	
3	A_{31}	
4	A_0	A_{32}
5	A_1	A_{33}
6	A_2	A_{34}
7	A_3	A_{35}
8	A_4	A_{36}
9	A_5	A_{37}
10	A_6	A_{38}
11	A_7	A_{39}
12	A_8	
$\vdots$		
30	A_{26}	
31	A_{27}	

So, if we access complete array twice we get $= 32 + 8 + 8 + 8 = 56$ miss because only 8 lines from cache line number 4 to 11 are miss operation, rest are **Hits(not counted) or Compulsory misses(first 32).**

Hence, answer is option **(C).**

GATE CSE 2007 | Question: 81

39

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 x 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.

Which of the following lines of the data cache will be replaced by new blocks in accessing the array for the second time?

A. line 4 to line 11
B. line 4 to line 12
C. line 0 to line 7
D. line 0 to line 8

https://gateoverflow.in/43511/gate-cse-2007-question-81

59

Best answer

Cache Organization:

Staring Address $= 1100H = 16^3 + 16^2 + 0 + 0 = 4352B$ is the starting address.

We need to find Starting block $= \dfrac{4352\ B}{64\ B} = 68^{th}$ block in main memory from where array start storing elements.

$50 \times 50\ B = \text{array size} = 50 \times \dfrac{50\ B}{64\ B} = 39.0625$ blocks needed $\approxeq 40\ blocks$

$68,69,70....107$ block we need $= 40$ blocks

Starting block is $68 \pmod{32} = 4^{th}$ cache block and after that in sequence they will be accessed.
As shown in below table, line number 4 to 11 has been replaced by array in second time

Cache Block Number	First Cycle	Second cycle
0	96	
1	97	
2	98	
3	99	
4	68 // 100	68
5	69 // 101	69
6	70//102	70
7	71//103	71
8	72//104	72
9	73//105	73
10	74/106	74
11	75//107	75
12	76	
13	77	
14	78	
15	79	
16	80	
17	81	
18	82	
19	83	
20	84	
21	85	
22	86	
23	87	
24	88	
25	89	
26	90	
27	91	
28	92	
29	93	
30	94	
31	95	

GATE CSE 2008 | Question: 73

30

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%

https://gateoverflow.in/43491/gate-cse-2008-question-73

33

Block size $= 16\text{B}$ and one element $= 8\text{B}$.
So, in one block 2 element will be stored.

For 1024×1024 element num of block required $= \dfrac{1024 \times 1024}{2} = 2^{19}$ blocks required.

Best answer

In one block the first element will be a miss and second one is hit(since we are transferring two unit at a time)

$$\Rightarrow \text{hit ratio} = \frac{\text{Total hit}}{\text{Total reference}}$$

$$= \frac{2^{19}}{2^{20}}$$

$$= \frac{1}{2} = 0.5$$

$$= 0.5 \times 100 = 50\%$$

Correct Answer: C

answered May 4, 2016 • edited Jun 20, 2021 by **Lakshman Bhaiya**

edit flag hide comment Follow

Pip Box Delete with Reason Wrong Useful

asu

GATE CSE 2002 | Question: 10

32

In a C program, an array is declared as $\text{float } A[2048]$. Each array element is 4 Bytes in size, and the starting address of the array is $0x00000000$. This program is run on a computer that has a direct mapped data cache of size 8 Kbytes, with block (line) size of 16 Bytes.

A. Which elements of the array conflict with element $A[0]$ in the data cache? Justify your answer briefly.

B. If the program accesses the elements of this array one by one in reverse order i.e., starting with the last element and ending with the first element, how many data cache misses would occur? Justify your answer briefly. Assume that the data cache is initially empty and that no other data or instruction accesses are to be considered.

https://gateoverflow.in/863/gate-cse-2002-question-10

52

Best answer

A. Data cache size $= 8\ KB$.

Block line size $= 16\ B$.

Since each array element occupies $4\ B$, four consecutive array elements occupy a block line (elements are aligned as starting address is 0)

Number of cache blocks $= \dfrac{8\ KB}{16\ B} = 512.$

Number of cache blocks needed for the array $= \dfrac{2048}{4} = 512.$

So, all the array elements has its own cache block and there is no collision.

We can also explain this with respect to array address.
Starting address is $0\text{x}00000000 = 0_b0000\ldots0(32\ 0\text{'s}).$
Ending address is
$0\text{x}00001\text{FFF} = 0_b0000\ldots01111111111111(4 \times 2048 = 8192\ \text{locations}),\ 0 - 8191$
.

Here, the last 4 bits are used as OFFSET bits and the next 9 bits are used as SET bits. So, since the ending address is not extending beyond these 9 bits, all cache accesses are to diff sets.

B. If the last element is accessed first, its cache block is fetched. (which should contain the previous 3 elements of the array also since each cache block hold 4 elements of array and 2048 is and exact multiple of 4). Thus, for every 4 accesses, we will have a cache miss $\Rightarrow$ for 2048 accesses we will have 512 cache misses. (This would be same even if we access array in forward order).

GATE CSE 1998 | Question: 18

32

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 $l = 1$.

gate1998 co-and-architecture cache-memory descriptive

https://gateoverflow.in/1732/gate-cse-1998-question-18

42

Size of the loop $= n \times b = k \times m \times b$

Size of a set $= k \times b$ ($k - way$ associative)

Here, size of the loop is smaller than size of cache as $m \leq l$. So we are guaranteed that the entire loop is in cache without any replacement. (Here we assumed that the memory size of $n \times b$ used by the loop is contiguous, or else we could have a scenario where the entire loop size gets mapped to the same set causing cache replacement).

For the first iteration:

No. of accesses $= n \times b$
No. of misses $= n$ as each new block access is a miss and loop body has n blocks each of size b for a total size of $n \times b$.

For, the remaining 99 iterations:

No. of accesses $= n \times b$
No. of misses $= 0$

So, total no. of accesses $= 100nb$

Total no. of hits $=$ Total no. of accesses $-$ Total no. of misses

$= 100nb - n$

So, hit ratio $= \frac{100nb-n}{100nb} = 1 - \frac{1}{100b}$

The hit ratio is independent of l, so for $l = 1$ also we have hit ratio $= 1 - \frac{1}{100b}$

GATE CSE 2008 | Question: 72

56

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.

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

A. $\mathrm{ARR}[0][4]$
B. $\mathrm{ARR}[4][0]$
C. $\mathrm{ARR}[0][5]$
D. $\mathrm{ARR}[5][0]$

59

Best answer

Number of sets = cache size/ size of a set
$= 64$ KB$/(16$ B $\times 2)$ (two blocks per set)
$= 2$ K $= 2^{11}$
So, we need 11 bits for set indexing.

Number of WORD bits required $= 4$ as a cache block consists of 16 bytes and we need 4 bits to address each of them.

So, number of tag bits $= 32 - 11 - 4 = 17$

Total size of the tag $= 17\times$ Number of cache blocks
$= 17 \times 2^{11} \times 2$ (since each set has 2 blocks)

$= 68$ KB

We use the top 17 bits for tag and the next 11 bits for indexing and next 4 for offset. So, for two addresses to have the same cache index, their 11 address bits after the 4 offset bits from right must be same.

$ARR[0][0]$ is located at virtual address 0x FF000 000. (FF000 is page address and 000 is page offset). So, index bits are 00000000000

Address of $ARR[0][4] =$ 0xFF000 $+ 4\times$ sizeof (double)
$=$ 0xFF000 000 $+ 4 \times 8 =$ 0xFF000 020$(32 = 20$ in hex) (index bits differ)

Address of $ARR[4][0]$ = 0xFF000 $+ 4 \times 1024\times$ sizeof (double) [since we use row major storage]
$=$ 0xFF000 000 $+ 4096 \times 8 =$ 0xFF000 000 $+$ 0x8000 $=$ 0xFF008 000 (index bits matches that of ARR $[0][0]$ as both read $000\ 0000\ 0000$)

Address of $ARR[0][5] = 0xFF000 + 5\times$ sizeof (double)
$= 0xFF000\ \ 000 + 5 \times 8 = 0xFF000\ \ 028 (40 = 28$ in hex) (index bits differ)

Address of $ARR[5][0] = 0xFF000 + 5 \times 1024\times$ sizeof (double) [since we use row major storage]
$= 0xFF000\ \ 000 + 5120 \times 8 = 0xFF000\ \ 000 + 0xA000 = 0xFF00A\ \ 000$ (index bits differ)

So, only $ARR[4][0]$ and $ARR[0][0]$ have the same cache index.

The inner loop is iterating from 0 to 1023, so consecutive memory locations are accessed in sequence. Since cache block size is only 16 bytes and our element being double is of size 8 bytes, during a memory access only the next element gets filled in the cache. i.e.; every alternative memory access is a cache miss giving a hit ratio of 50%. (If loops i and j are reversed, all accesses will be misses and hit ratio will become 0).

Correct Answer: B

answered Aug 16, 2016 • edited Jun 20, 2021 by **Lakshman Patel RJIT**

edit flag hide comment Follow

Pip Box Delete with Reason Wrong Useful

Arjun