Problem:

The following table gives the parameters for a number of different caches, where m is the number of physical address bits, C is the cache size (number of data bytes), B is the block size in bytes, and E is the number of lines per set. For each cache, determine the number of cache sets (S), tag bits (t), set index bits (s), and block offset bits (b).

Cache	m	C	B	E	S	t	s	b
1.	32	1024	4	4	**64**	**24**	**6**	**2**
2.	32	1024	4	256	**1**	**30**	**0**	**2**
3.	32	1024	8	1	**128**	**22**	**7**	**3**
4.	32	1024	8	128	**1**	**29**	**0**	**3**
5.	32	1024	32	1	**32**	**22**	**5**	**5**
6.	32	1024	32	4	**8**	**24**	**3**	**5**

You can find `b` by taking $\log_2$ of `B`. In other words, how many bits do you need to represent B of something? The next column to calculate is `S`, the number of sets. We know the total cache size, `C`, is equal to: `C = S * E * B`. Solving for `S`: `S = C/(E*B)`. An easy way to calculate S without using a calculator is to convert everything to a power of two. For example, For line 1: `1024/(4*4)` = $2^{10}/(2^2 * 2^2) = 2^{10}/2^4 = 2^6 = 64$. Once you have `S`, you can calculate `s` in the same way as you did column `b` (the $\log_2$ of `S`). Finally, the tag is just the leftover bits that are not used for the set bits and the block offset bits. `t = m - s - b`.

Problem:

This problem gives practice with basic cache lookups.

- The memory is byte addressable.
- Memory accesses are to 1-byte words.
- Addresses are 12 bits wide.
- The cache is 4-way set associative, with a 4 byte line size and 8 total lines.

In the following tables, all numbers are given in hexadecimal. The contents of the cache are as follows:

Set	Tag	Valid	Byte0	Byte1	Byte2	Byte3
S_0	058	0	02	55	AD	87
S_0	123	1	3E	98	47	51
S_0	12B	0	6C	77	89	14
S_0	0EF	1	B9	64	78	25
S_1	069	0	00	FF	14	43
S_1	12B	1	92	63	42	21
S_1	075	0	33	BE	AF	31
S_1	1C6	1	22	17	02	24

The box below shows the format of a physical address. Indicate (by labeling the diagram) the fields that would be used to determine the following:

B The block offset within the cache line
S The set index
T The cache tag

Blocks are 4 bytes, so we have 2 block offset bits. There are 8 lines with 4 lines per set (we are given that it is a 4-way set associative cache), so we must have 2 sets. Therefore, we have 1 set bit. Our tag bits are what is leftover, 12 - 2 - 1 = 9.

11	10	9	8	7	6	5	4	3	2	1	0
T	**T**	**T**	**T**	**T**	**T**	**T**	**T**	**T**	**S**	**B**	**B**

For the given physical address, indicate the cache entry accessed and the cache byte value returned in hex. Indicate whether a cache miss occurs.

If there is a cache miss, enter “-” for “Cache Byte returned”.

Physical address : `0x95B`

In the box below, write out the physical address (one bit per box).

11	10	9	8	7	6	5	4	3	2	1	0
1	**0**	**0**	**1**	**0**	**1**	**0**	**1**	**1**	**0**	**1**	**1**

Fill out the table below:

Parameter	**Value**
Block offset	**0x3**
Set Index	**0x0**
Cache Tag	**0x12B**
Cache Hit? (Y/N)	**N**
Cache Byte returned	-

The tag matches, but the valid bit for the line is 0.

For the given physical address, indicate the cache entry accessed and the cache byte value returned in hex. Indicate whether a cache miss occurs.

If there is a cache miss, enter “-” for “Cache Byte returned”.

Physical address : `0xE34`

In the box below, write out the physical address (one bit per box).

11	10	9	8	7	6	5	4	3	2	1	0
1	**1**	**1**	**0**	**0**	**0**	**1**	**1**	**0**	**1**	**0**	**0**

Fill out the table below:

Parameter	**Value**
Block offset	**0x0**
Set Index	**0x1**
Cache Tag	**0x1c6**
Cache Hit? (Y/N)	**Y**
Cache Byte returned	**0x22**

This question involves cache lookups. The cache below has the following properties.

- The memory is byte addressable.
- Physical addresses are 16 bits wide.
- The cache is 4-way set associative, with a 4 byte line size and 16 total lines.

In the following tables, all numbers are given in hexadecimal . The contents of the cache are as follows:

Set	Tag	Valid	Byte0	Byte1	Byte2	Byte3
S_0	321	0	02	55	AD	87
S_0	733	1	7A	F7	72	6E
S_0	750	0	C3	81	93	D5
S_0	BAC	1	B9	64	78	25
S_1	1F9	0	00	FF	14	43
S_1	ACC	1	92	63	42	21
S_1	69D	0	33	4F	F4	31
S_1	553	1	22	17	02	24
S_2	347	1	42	21	BE	AF
S_2	23E	1	57	7F	E1	69
S_2	35C	0	92	63	42	21
S_2	DBB	0	33	BF	62	B5
S_3	CBB	1	69	88	ED	54
S_3	676	0	15	92	B1	EE
S_3	62E	1	3E	98	47	51
S_3	7D3	1	6C	77	89	14

For the given address below, indicate the cache entry accessed and the cache value returned in hex. Indicate whether a cache miss occurs. **If there is a cache miss, enter "miss" for "Cache Value returned**".

Fill out the table below for the **int** at address: `0x7330`

Parameter	**Value**
Block offset	`0x0`
Set Index	`0x0`
Cache Tag	`0x733`
Cache Value Returned (or "miss")	`0x6E72F77A`

Fill out the table below for the **char** at address: `0x62EF`

Parameter	**Value**
Block offset	`0x3`
Set Index	`0x3`
Cache Tag	`0x62E`
Cache Value Returned (or "miss")	`0x51`

Fill out the table below for the **char** at address: `0x35C9`

Parameter	**Value**
Block offset	`0x1`
Set Index	`0x2`
Cache Tag	`0x35C`
Cache Value Returned (or "miss")	`0x miss`

Fill out the table below for the **short** at address: `0xACC6`

Parameter	**Value**
Block offset	`0x2`
Set Index	`0x1`
Cache Tag	`0xACC`
Cache Value Returned (Value or N/A)	`0x2142`

You are given the following code:

```
#define MAX 100000

int sum(int array[MAX][MAX]) {
    int i, j, sum;
    sum = 0;
    for (i = 0; i < MAX; i++) {
        for (j = 0; j < MAX; j++) {
            sum += array[j][i];
        }
    }
    return sum;
}
```

What kind of locality does the array access `array[j][i]` have? Fill in **one** oval.

- ○ It has good spatial locality
- ○ It has good temporal locality
- ○ **It does not have good locality**

Does the access to `sum` have good locality? Fill in **one** oval.

- ○ It has good spatial locality
- ○ **It has good temporal locality**
- ○ It does not have good locality

Rank the following memory devices in order of access time, from 1 to 4, where **1 has fastest access time and 4 has the slowest access time**.

Memory	Access Time
Cache	**2**
DRAM	**3**
Registers	**1**
Disk	**4**

Which type of cache miss occurs the first time a memory block is accessed? Fill in **one** oval.

- ○ Capacity miss
- ○ Conflict miss
- ○ **Cold (Compulsory) miss**
- ○ Coherence miss

In a 2-way set associative cache, how many lines does each set hold? Fill in **one** oval.

- ○ 1
- ○ **2**
- ○ 4
- ○ It depends on the cache size

Which cache mapping strategy requires the least hardware but may lead to the most conflict misses? Fill in **one** oval.

- ○ Fully associative
- ○ Set-associative
- ○ **Direct-mapped**
- ○ Write-back

Which cache replacement policy evicts the block that has not been accessed for the longest time? Fill in **one** oval.

- ○ FIFO (First In First Out)
- ○ Random
- ○ **LRU (Least Recently Used)**
- ○ MRU (Most Recently Used)

Which of the following cache organizations provides the greatest flexibility in placing a block? Fill in **one** oval.

- ○ Direct-mapped cache
- ○ 2-way set associative cache
- ○ 4-way set associative cache
- ○ **Fully associative cache**

What is the purpose of the ***valid bit*** in a cache block? Fill in **one** oval.

- **To indicate if the block is currently being used**
- To track the age of the data
- To signal if the block contains updated data
- To identify the set number

What does the ***tag*** in a cache block help determine? Fill in **one** oval.

- The block size in bytes
- The validity of the data
- **If a memory address maps to a cache line**
- The block's replacement policy

You are constructing a logic circuit based on a Truth Table with three inputs (A, B, C) and one output (Y). The Truth Table indicates that the output Y is High (1) only in the following two states:

Row 3: A = 0, B = 1, C = 1
Row 5: A = 1, B = 0, C = 1

Which of the following Boolean expressions represents the correct unsimplified Sum of Products (SOP) construction for this circuit?

- Y = (A & ~B & ~C) | (~A & B & ~C)
- **Y = (~A & B & C) | (A & ~ B & C)**
- Y = (~A & B & C) | (A & ~B & C)
- Y = (A & ~B & ~C) | (~A & B & ~C)