**Question 1:**

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

| Cache | *m* | *C* | *B* | *E* | *S* | *t* | *s* | *b* |
|---|---|---|---|---|---|---|---|---|
| 1. | 32 | 1,024 | 4 | 4 | | | | |
| 2. | 32 | 1,024 | 4 | 256 | | | | |
| 3. | 32 | 1,024 | 8 | 1 | | | | |
| 4. | 32 | 1,024 | 8 | 128 | | | | |
| 5. | 32 | 1,024 | 32 | 1 | | | | |
| 6. | 32 | 1,024 | 32 | 4 | | | | |

C = B * E * S => S = C/BE
t = m - s - b
s = lg(S)
b = lg(B)

| S | t | s | b |
|---|---|---|---|
| 64 | 24 | 6 | 2 |
| 1 | 30 | 0 | 2 |
| 128 | 22 | 7 | 3 |
| 1 | 29 | 0 | 3 |
| 32 | 22 | 5 | 5 |
| 8 | 24 | 3 | 5 |

**Question 2:**

Suppose we have a system with the following properties:

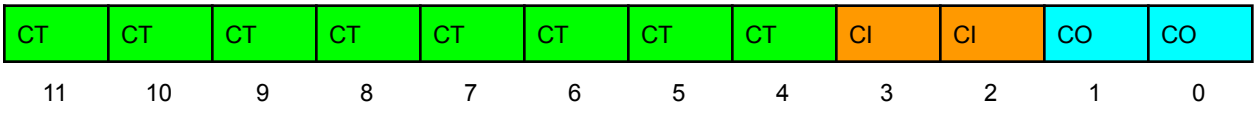
- The memory is byte addressable.
- Memory accesses are to 1-byte words (not to 4-byte words).
- Addresses are 12 bits wide.
- The cache is two-way set associative (E = 2), with a 4-byte block size (B = 4) and four sets (S = 4).

The contents of the cache are as follows, with all addresses, tags, and values given in hexadecimal notation:

| Set index | Tag | Valid | Byte 0 | Byte 1 | Byte 2 | Byte 3 |
|---|---|---|---|---|---|---|
| 0 | 00 | 1 | 40 | 41 | 42 | 43 |
| | 83 | 1 | FE | 97 | CC | D0 |
| 1 | 00 | 1 | 44 | 45 | 46 | 47 |
| | 83 | 0 | — | — | — | — |
| 2 | 00 | 1 | 48 | 49 | 4A | 4B |
| | 40 | 0 | — | — | — | — |
| 3 | FF | 1 | 9A | C0 | 03 | FF |
| | 00 | 0 | — | — | — | — |

**A.** The following diagram shows the format of an address (1 bit per box). Indicate (by labeling the diagram) the fields that would be used to determine the following:

- CO. The cache block offset (# of words in a block = 4, bits = lg(4) = 2)
- CI. The cache set index (# of sets in cache = 4, bits = lg(4) = 2)
- CT. The cache tag (remaining, 12 - 2 + 2 = 8 bits)




**B.** For each of the following memory accesses, indicate if it will be a cache hit or miss when carried out in sequence as listed. Also give the value of a read if it can be inferred from the information in the cache.

| Operation | Address | Hit? | Read Value (or unknown) |
|---|---|---|---|
| Read | 0x834 | miss | unknown |
| Write | 0x836 | hit | unknown |
| Read | 0xFFD | hit | C0 |

**Question 3:**

You are writing a new 3D game that you hope will earn you fame and fortune. You are currently working on a function to blank the screen buffer before drawing the next frame. The screen you are working with is a 640 × 480 array of pixels. The machine you are working on has a 32 KB direct-mapped cache with 8-byte lines. The C structures you are using are as follows:

```
1   struct pixel {
2       char r;
3       char g;
4       char b;
5       char a;
6   };
7
8   struct pixel buffer[480][640];
9   int i, j;
10  char *cptr;
11  int *iptr;
```

Assume the following:

- sizeof(char) = 1 and sizeof(int) = 4.
- buffer begins at memory address 0.
- The cache is initially empty. .
- The only memory accesses are to the entries of the array buffer. Variables i, j, cptr, and iptr are stored in registers.

What percentage of writes in the following code will hit in the cache? Give a brief explanation of how you reached your answer. Values without explanation will not receive credit.

```
1       for (j = 639; j >= 0; j--) {
2           for (i = 479; i >= 0; i--){
3               buffer[i][j].r = 0;
4               buffer[i][j].g = 0;
5               buffer[i][j].b = 0;
6               buffer[i][j].a = 0;
7           }
8       }
```

75% of writes will hit the cache. After the first write which 4 bytes are read from, the next 3 are going to be in the cache. Leading to my answer of 75% because ¾ are in the cache.