| | |
|---|---|
| **Started on** | Saturday, 7 March 2020, 9:48 PM |
| **State** | Finished |
| **Completed on** | Monday, 9 March 2020, 4:37 PM |
| **Time taken** | 1 day 17 hours |
| **Marks** | 22.24/43.00 |
| **Grade** | **5.17** out of 10.00 (**52**%) |

Question **1**

Correct

Mark 2.00 out of 2.00

Estimate the time (in ms) to access a sector on the following disk

| Rotational Rate | Average Seek Time | Average Sectors per track |
|---|---|---|
| 15000 | 10 ms | 405 |

You may use an expression if that's useful.

12

Your last answer was interpreted as follows: 12

Correct answer, well done.
That's close enough to the precise value.

The total access time is is $T_{\text{access}} = T_{\text{avg. seek}} + T_{\text{avg rotation}} + T_{\text{avg transfer}}$.

The seek time is $T_{\text{seek}} = 10$, as specified in the problem.

The rotation delay is
$T_{\text{avg rotation}} = 1/2 \times T_{\text{max rotation}} = 1/2 * (60/15000) * 1000\text{ms/s}.$

The transfer delay is
$T_{\text{avg transfer}} = (60/15000) * (1/15000)\text{sectors/track} * 1000\text{ms/s}.$

The total access time is thus the sum or $\frac{4864}{405}$ = $12.0098765432$.

A correct answer is $\frac{4864}{405}$, which can be typed in as follows: `4864/405`

Question **2**

Not answered

Marked out of 4.00

At a high level, you can model the time to access a hard disk drive as the "access time" and the "transfer time" and a solid state disk can be modeled in a similar way.

Assume that a specific hard disk drive has an average access time of $9$ms (*i.e.* the seek and rotational delay sums to $9$ms) and a throughput or transfer rate of $190$MBytes/s, where a megabyte is measured as $1024^2$.

A solid-state drive (SSD) has an average access time of $0.035$ms and a throughput of $610$MB/s -- the access time serves the same role as the combined "seek" and "rotationaal" delay of a disk and represents the time to talk to the SSD and the overhead time for the non-volatile memory to be accessed.

**Big Reads**

Some applications, such as playing back a movie, read large files -- they do a single "seek" or access to the beginning of the file and then read a large collection of blocks. Assume that a movie playing application does a single "access" and then reads a $125$MB file.

How many "movies per second" can be processed by the hard disk drive?

How many "movies per second" can be processed by the SSD disk drive?

Each answer should be accurate to within 5% "movies per second" and you can use algebraic expressions if you like.

**Random I/O**

Many other applications are limited by "IOPS", or the "random I/O operations per second". An example of this would be a database that needs to seek to a part of the disk and read a small amount of data of 512 bytes repeatedly.

How many "IOPS" can be processed by the hard disk drive?

How many "IOPS" can be processed by the SSD disk drive?

Each answer should be accurate to within one "IOPS" and you can use algebraic expressions if you like.

The time to seek to and read a $125$MB file is $T_{\text{fs}} = T_{\text{seek}} + (125/\text{throughput})$. The speed in files per second is $\frac{1}{T_{\text{fs}}}$.

For the disk, this is $\frac{1}{\frac{9}{1000}+\frac{125}{190}} = \frac{19000}{12671} = 1.4994870176.$

For the ssd, this is $\frac{1}{\frac{7}{\frac{200}{1000}+\frac{125}{610}}} = \frac{12200000}{2500427} = 4.87916663834.$

There's a relatively small (3-10x) difference in the number of movies-per-second that can be served, related to the difference in throughput.

The number of IOPS is $1/T_{\text{seek}}$.

For the disk this is $\frac{1io}{9ms*\frac{1s}{1000ms}} = \frac{1000}{9} = 111.111111111.$

For the ssd this is $\frac{1io}{0.035ms*\frac{1s}{1000ms}} = \frac{200000}{7} = 28571.4285714.$

There's a relatively large (~500x) difference in the number of movies-per-second that can be served, related to the difference in latency.

---

A correct answer is $\frac{19000}{12671}$, which can be typed in as follows: `19000/12671`

A correct answer is $\frac{12200000}{2500427}$, which can be typed in as follows: `12200000/2500427`

A correct answer is $\frac{1000}{9}$, which can be typed in as follows: `1000/9`

A correct answer is $\frac{200000}{7}$, which can be typed in as follows: `200000/7`

Question **3**

Correct

Mark 1.00 out of 1.00

Assume you're using a computer write a 2-way set associate 32KB cache where writes that miss in the cache are directly written to the next cache hierarchy (i.e. a write-around policy). Assume that the constant **N** is very large.

Fill in the loops in a way that minimizes cache misses.

```
double A[N][N];
double B[N][N];

void foo()
{
    int i, j;
    [                    ✔ {
    for(i = 0; i < N; i ++) {  ✔ {
      for(j = 0; j < N; j++) {  ];
      }
    }
}
```

Your answer is correct.

The correct answer is:

Assume you're using a computer write a 2-way set associate 32KB cache where writes that miss in the cache are directly written to the next cache hierarchy (i.e. a write-around policy). Assume that the constant **N** is very large.

Fill in the loops in a way that minimizes cache misses.

```
double A[N][N];
double B[N][N];

void foo()
{
    int i, j;
    [for(i = 0; i < N; i ++) {] {
      [for(j = 0; j < N; j++) {] {
           A[i][j] = B[i][j];
      }
    }
}
```

Question **4**

Correct

Mark 1.00 out of 1.00

Given the following definition of structs

```
#define N 1000
typedef struct {
   int vel[3];
   int acc[3];
} point;
points p[N];
```

and the three following routines (which all do the same thing):

## (a) clear1

```
void clear1(point *p, int n) {
   int i, j;
   for (i = 0; i < N; i++) {
      for (j = 0; j < 3; j++)
         p[i].vel[j] = 0;
      for (j = 0; j < 3; j++)
         p[i].acc[j] = 0;
    }
}
```

## (b) clear2

```
void clear2(point *p, int n) {
   int i, j;
   for (i = 0; i < N; i++) {
      for (j = 0; j < 3; j++) {
         p[i].vel[j] = 0;
         p[i].acc[j] = 0;
    }
}
```

## (c) clear3

```
void clear1(point *p, int n) {
   int i, j;
   for (j = 0; j < 3; j++) {
      for (i = 0; i < N; i++)
         p[i].vel[j] = 0;
      for (i = 0; i < 3; i++)
         p[i].acc[j] = 0;
    }
}
```

| | |
|---|---|
| Best spatial locality | clear1 ✔ |
| Worst spatial locality | clear3 ✔ |
| Not best and not worst locality | clear2 ✔ |

Your answer is correct.

The correct answer is: Best spatial locality → clear1, Worst spatial locality → clear3, Not best and not worst locality → clear2

Question **5**

Partially correct

Mark 3.00 out of 4.00

Determine the number of cache sets (S), tag bits (t), set index bits (s), and block offset bits (b) for a 1024-byte cache using 32-bit memory addresses, 4-byte cache blocks and a single (direct-mapped) set.

This cache has S= 0 ✖ sets, t= 22 ✔ tag bits, s= 8 ✔ set index bits and b= 2 ✔ block offset bits.

Question **6**

Partially correct

Mark 3.64 out of 4.00

Assume the following:

- . The memory is byte addressable.
- . Memory accesses are to 1-byte words (not to 4-byte words).
- . Addresses are 10 bits wide.
- . The cache is 2-way associative cache (E=2), with a 8-byte block size (B=8) and 4 sets (S=4).

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

CO - The cache block offset

CI - The cache set index

CT - The cache tag

| 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|
| CT | CT | CT | CT | CT | CI | CI | CO | CO | CO |

A cache with this configuration could store a total of 1024 ✖ bytes of memory (ignoring the tags and valid bits).

Question **7**

Partially correct

Mark 3.60 out of 4.00

Assume the following:

- . The memory is byte addressable.
- . Memory accesses are to 1-byte words (not to 4-byte words).
- . Addresses are 9 bits wide.
- . The cache is 2-way associative cache (E=2), with a 4-byte block size (B=4) and 2 sets (S=2).

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

CO - The cache block offset

CI - The cache set index

CT - The cache tag

| 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| CT | CT | CT | CT | CT | CT | CI | CO | CO |

A cache with this configuration could store a total of 512 ✖ bytes of memory (ignoring the tags and valid bits).

Question **8**

Correct

Mark 4.00 out of 4.00

The heart of the recent hit game *SimAquarium* is a tight loop that calculates the average position of 256 algae. You are evaluating its cache performance on a machine with a 1024-byte direct-mapped data cache with 16-byte blocks (B = 16).

You are given the following definitions:

```
struct algae_position {
   int x;
   int y;
};

struct algae_position grid[16][16];
int total_x = 0, total_y = 0;
int i, j;
```

When the following code is executed:

```
for (i = 0; i < 16; i++) {
  for (j = 0; j < 16; j++) {
    total_x += grid[i][j].x;
  }
}
for (i = 0; i < 16; i++) {
  for (j = 0; j < 16; j++) {
    total_y += grid[i][j].y;
   }
}
```

there are 512 ✔ total reads or loads and 256 ✔ reads or loads that miss in the cache,
resulting in a cache miss rate of 50 ✔ %.

Question **9**

Correct

Mark 4.00 out of 4.00

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 64 KB direct-mapped cache with 4-byte lines.

The C structures you are using are as follows:

```
struct pixel {
  char r;
  char g;
  char b;
  char a;
};
struct pixel buffer[480][640];
int i,j;
char *cptr;
int *iptr;
```

And assuming the following:

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

Determine the cache performance of the following code:

```
for (i = 0; i < 480; i++) {
   for (j = 0; j < 640; j++) {
     buffer[i][j].r = 0;
     buffer[i][j].g = 0;
     buffer[i][j].b = 0;
     buffer[i][j].a = 0;
  }
}
```

The miss rate is 25 ✔ % (accurate to +/-0.5%)

Each pixel structure is 4 bytes, so each 4-byte cache line holds exactly one structure. For each structure, there is a miss, followed by three hits, for a miss rate of 25%.

Question **10**

Not answered

Marked out of 5.00

Assume the following:

- . 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 4-way associative cache (E=4), with a 16-byte block size (B=16) and 16 sets (S=16).
- The cache contents are as shown below

| Set # | Way #0 | Way #1 | Way #2 | Way #3 |
|---|---|---|---|---|
| 0: | V=1;Tag=0x4; Data = 0xbb 0x1e 0xcf 0xef 0x7b 0x4c 0x03 0x42 0x8a 0xf0 0xdb 0xa8 0x7f 0xd1 0x48 0x3e | V=0;Tag=0xc; Data = -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- | V=1;Tag=0xd; Data = 0xf6 0x6e 0x0d 0x8f 0x9e 0x25 0x5b 0xe7 0x1f 0xf5 0x83 0xf5 0x75 0xfb 0x51 0xdb | V=1;Tag=0x3; Data = 0x9b 0x3e 0xd6 0x28 0x72 0x12 0x6b 0x88 0xab 0xb6 0x45 0xc8 0x75 0x13 0xa1 0x27 |
| 1: | V=1;Tag=0x7; Data = 0xd0 0x52 0x4e 0x20 0x43 0x4a 0x12 0x1f 0x5b 0x9a 0x91 0x68 0x24 0xe9 0x04 0xa0 | V=1;Tag=0xd; Data = 0x46 0x51 0x4e 0xe5 0x24 0x4e 0xfe 0x48 0x4d 0xbf 0x83 0x5a 0xe4 0xa8 0x54 0xcd | V=1;Tag=0x0; Data = 0x96 0xe0 0x6b 0x40 0xf9 0x1f 0xae 0x82 0x50 0x6a 0x5b 0xb2 0x50 0xc9 0x4f 0x49 | V=1;Tag=0x1; Data = 0x8a 0x99 0xe8 0x59 0x55 0x6a 0x97 0x65 0x61 0xd9 0x62 0x16 0x9a 0xb9 0xd6 0x93 |
| 2: | V=1;Tag=0x0; Data = 0x6a 0x2e 0xdd 0xb1 0x49 0x2e 0xee 0x4b 0xb7 0xb4 0x5b 0xd1 0xf1 0x0e 0xe6 0x02 | V=1;Tag=0x1; Data = 0x16 0x0c 0xa0 0xaa 0x78 0x36 0x86 0xbc 0x73 0xbe 0x50 0x52 0x38 0x7f 0x48 0x57 | V=1;Tag=0x7; Data = 0xea 0x70 0x3d 0xf2 0x01 0xc9 0x8d 0x86 0x13 0x92 0x6d 0xaa 0x34 0x85 0x11 0xee | V=1;Tag=0x6; Data = 0x1b 0x80 0x01 0xce 0x76 0xc6 0xd3 0xae 0xf3 0x96 0x20 0xed 0x80 0x7f 0xf3 0xc8 |
| 3: | V=1;Tag=0x6; Data = 0x64 0x7c 0x52 0x9d 0xf7 0x70 0x5c 0x28 0xe5 0x96 0xc3 0x06 0x73 0x52 0x7f 0x8c | V=0;Tag=0xc; Data = -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- | V=0;Tag=0xd; Data = -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- | V=1;Tag=0x9; Data = 0xfd 0xd5 0x47 0xc7 0xaf 0xc7 0x49 0x56 0xb0 0xd1 0x69 0x29 0x39 0x61 0x99 0xef |
| 4: | V=1;Tag=0xe; Data = 0xfa 0x07 0x1b 0xa4 0x89 0x1b 0xc4 0x52 0x3f 0x4d 0xaf 0x21 0xaf 0x57 0x7d 0x70 | V=1;Tag=0x0; Data = 0x50 0x73 0x11 0x5f 0xd4 0x26 0x73 0xae 0x88 0xf5 0xc4 0xae 0xde 0x4c 0x72 0x3e | V=1;Tag=0x3; Data = 0xc2 0x1c 0xd9 0xeb 0x11 0xd8 0xa7 0x54 0x2e 0xfc 0xf1 0x36 0x42 0x38 0xd2 0x14 | V=1;Tag=0xd; Data = 0x3e 0x13 0x2a 0x1a 0x71 0xc4 0x4f 0x4f 0x3b 0x09 0xae 0x55 0xe3 0xd1 0x44 0x5d |
| 5: | V=1;Tag=0x0; Data = 0xe8 0xce 0x90 0x3d 0x3f 0x50 0x34 0xc7 0xb6 0x99 0xbe 0xf6 0x23 0x6a 0x38 0x06 | V=1;Tag=0xd; Data = 0x5b 0x90 0x18 0xf1 0x60 0xf5 0x89 0x4a 0x86 0xef 0x00 0x39 0xaf 0xd0 0xbf 0x5d | V=1;Tag=0x6; Data = 0xa2 0x9f 0xf2 0x42 0xf2 0xd6 0x56 0x52 0x9d 0x66 0x8d 0x19 0x33 0x76 0xf3 0xee | V=1;Tag=0x2; Data = 0xcf 0x86 0x3c 0x17 0x7e 0x70 0x75 0x4b 0x6c 0x3d 0x6f 0x48 0x3e 0x9a 0xab 0x9c |
| 6: | V=0;Tag=0x4; Data = -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- | V=1;Tag=0x9; Data = 0x27 0x50 0x3c 0x59 0x8f 0x6c 0xec 0x2a 0x79 0x3c 0x6e 0xc4 0x02 0x5a 0x56 0xb4 | V=1;Tag=0x8; Data = 0x89 0xed 0x45 0x5e 0x45 0xdd 0x5f 0xce 0x83 0xb7 0xf3 0x92 0x8f 0xb4 0xc1 0x81 | V=1;Tag=0xc; Data = 0xfb 0x49 0x8b 0xdc 0xc8 0xd8 0x1f 0xfd 0x3f 0xfa 0x51 0x53 0x4b 0x10 0xbb 0x6b |
| 7: | V=1;Tag=0xc; Data = 0xd5 0xc1 0x36 0x15 0x2e 0x09 0xee 0x21 0xe0 0x1e 0x4a 0x15 0x1f 0xa2 0x76 0x80 | V=1;Tag=0x2; Data = 0xf5 0x9f 0xcf 0x31 0x98 0x78 0x07 0x8e 0xdd 0x7a 0x80 0x04 0xa6 0xf4 0x0a 0x6a | V=1;Tag=0x4; Data = 0xc0 0x87 0xc0 0xd0 0x7d 0xa1 0x43 0xa3 0xcc 0xa7 0x80 0xfc 0xe0 0x3c 0xa3 0xdf | V=1;Tag=0x9; Data = 0xbd 0x8f 0x64 0x70 0x93 0x94 0x23 0x76 0x94 0x67 0xaa 0xd1 0xe0 0x33 0xc2 0x84 |
| 8: | V=1;Tag=0x6; Data = 0x32 0xb1 0x56 0x51 0xeb 0xd0 0x52 0x68 0x50 0xc7 0xc2 0x86 0x59 0xef 0xd6 0x1c | V=1;Tag=0x3; Data = 0xde 0x58 0x57 0xe4 0x84 0xa0 0x9e 0xb5 0x8d 0x59 0x91 0x76 0x52 0x09 0xc3 0x5f | V=1;Tag=0xe; Data = 0xe3 0x93 0x88 0x8b 0xee 0x3c 0xdc 0x0d 0xa7 0xe8 0x30 0x4d 0xa0 0x1f 0xf7 0x26 | V=0;Tag=0x4; Data = -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- |
| 9: | V=1;Tag=0x3; Data = 0xda 0xd5 0x99 0x40 0x91 0x66 0x75 0xd9 0x25 0x49 0xe6 0x14 0xef 0x16 0xca 0x9f | V=1;Tag=0x8; Data = 0x31 0x7f 0x00 0xc0 0x40 0x9b 0xdd 0xe8 0x78 0x1e 0x3e 0x06 0xd3 0x6f 0xf4 0xf0 | V=1;Tag=0x9; Data = 0x6f 0xc8 0xc7 0x9e 0xc5 0x6f 0x73 0x25 0x90 0x18 0xfd 0xf9 0x5d 0x34 0xda 0x3c | V=1;Tag=0x2; Data = 0x33 0x31 0x94 0xfe 0xdd 0xad 0xaa 0x60 0x42 0xfd 0x10 0xb4 0x28 0x92 0x4f 0x32 |
| 10: | V=1;Tag=0x2; Data = 0x3b 0xd6 0xfb 0x58 0x7a 0xbe 0x4e 0xbb 0xdc 0x1a 0xd5 0x5a 0x45 0x50 0x1d 0xaf | V=1;Tag=0xd; Data = 0x30 0xbd 0x6f 0x9b 0x2f 0xd1 0xf8 0xda 0x45 0x45 0x74 0x76 0xf1 0x3a 0x1c 0xae | V=1;Tag=0x9; Data = 0xad 0x9c 0xe5 0xd2 0xe5 0xf3 0x38 0x6b 0x33 0x51 0xed 0x67 0x42 0x3e 0x55 0x8e | V=1;Tag=0x7; Data = 0xa0 0xe0 0xdd 0xbc 0xc7 0xe5 0x66 0x94 0xc5 0x8b 0x00 0x03 0x9f 0xc3 0x02 0xa7 |

| | | | | |
|---|---|---|---|---|
| 11: | V=1;Tag=0xe; Data = 0x83 0xc6 0x31 0xc8 0x3b 0x14 0x58 0x89 0x54 0x8d 0xd3 0x5d 0xc2 0x3d 0x9b 0x11 | V=1;Tag=0xa; Data = 0x0f 0xf4 0x8e 0xd0 0x21 0xf0 0x1e 0x3b 0x96 0x7f 0x4c 0x63 0xb0 0xc4 0x15 0xc6 | V=1;Tag=0x6; Data = 0x0a 0xb9 0x22 0x3f 0xae 0x51 0xd2 0x99 0xa5 0x58 0x7f 0x96 0x7c 0xea 0x1a 0xdb | V=1;Tag=0x7; Data = 0x80 0x21 0x92 0x8d 0xbc 0x47 0xb4 0xa6 0x9f 0x14 0xae 0x87 0x90 0x6c 0x21 0xab |
| 12: | V=1;Tag=0x6; Data = 0x74 0x17 0xbf 0xe1 0xaf 0x94 0x9d 0x3a 0x55 0x43 0xa1 0x3d 0xe6 0x43 0x35 0xdb | V=1;Tag=0x2; Data = 0xf2 0x53 0xe6 0x10 0xbc 0x41 0x11 0x05 0xea 0x66 0xf6 0x88 0x72 0x95 0xf7 0x1f | V=0;Tag=0xc; Data = -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- | V=1;Tag=0xb; Data = 0xf5 0x14 0xc2 0x73 0x12 0x6a 0x52 0x89 0x7a 0x3a 0x78 0x0e 0x99 0xd4 0x61 0x0c |
| 13: | V=1;Tag=0xd; Data = 0xd6 0xdf 0x1d 0x2e 0x78 0x39 0x28 0x44 0x71 0x7b 0x8a 0x4d 0x5e 0xf1 0x30 0xab | V=1;Tag=0xe; Data = 0xd5 0x69 0x2f 0x29 0x1c 0x05 0xec 0xe7 0xb6 0xe0 0xc5 0x76 0xb3 0x67 0xa3 0xd4 | V=1;Tag=0x5; Data = 0x7b 0x60 0xfc 0xba 0xbc 0x12 0xd3 0x85 0x8b 0xde 0x78 0x17 0x21 0x4c 0x54 0xc8 | V=1;Tag=0x9; Data = 0xb0 0xda 0x80 0x41 0x27 0x45 0x16 0x7e 0xf0 0x07 0x81 0x93 0x37 0x16 0x56 0x74 |
| 14: | V=1;Tag=0xd; Data = 0xaa 0x69 0x3c 0xeb 0xe1 0x25 0x35 0x6a 0xd6 0xfd 0xc1 0xab 0x78 0xd5 0x22 0x90 | V=1;Tag=0x9; Data = 0xc1 0x0a 0x5f 0x1c 0x56 0x88 0x9d 0x19 0xe0 0xe0 0x22 0x10 0x9a 0x3e 0xd7 0x4f | V=1;Tag=0x3; Data = 0xf3 0xee 0xf6 0x99 0xd8 0xd2 0x48 0xa5 0x22 0xb5 0x61 0x97 0x1b 0x98 0xf1 0x93 | V=1;Tag=0x5; Data = 0x2c 0xd7 0x4a 0xe9 0xd2 0xe3 0xaa 0x87 0xb7 0x0b 0xb7 0x2c 0xd8 0xd2 0x87 0xc2 |
| 15: | V=1;Tag=0x1; Data = 0xce 0x08 0xc6 0xca 0x11 0x4e 0xbd 0x25 0x3d 0x1c 0x43 0x7c 0x15 0x33 0xa9 0x41 | V=1;Tag=0x9; Data = 0xdc 0xce 0x6e 0xa7 0x0d 0xbd 0xfb 0x9c 0xf4 0x3f 0x9f 0xc2 0xc8 0xb6 0x05 0xa5 | V=1;Tag=0x8; Data = 0x88 0x1d 0xe8 0x4c 0x47 0x1e 0x2e 0x9c 0x32 0xa2 0xf4 0x0f 0x1c 0xd5 0xa9 0x4f | V=1;Tag=0x7; Data = 0x61 0x23 0x9e 0x6d 0xc1 0xe2 0xb3 0xec 0x2d 0x90 0x02 0x68 0xaa 0x47 0x5f 0xdc |

Assume that memory address **0xad9** has been referenced by a load instruction. 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 the "Cache Byte Returned". For values that need a hexidecimal value, do not enter leading zeros even if leading zeros are shown in the value above.

Cache block Offset (CO) 0x ✖

Cache set index (CI) 0x ✖

Cache tag (CT) 0x ✖

Cache hit (Y/N)? ✖

Cache byte returned 0x ✖

Question **11**

Not answered

Marked out of 5.00

Assume the following:

- . 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 2-way associative cache (E=2), with a 16-byte block size (B=16) and 16 sets (S=16).
- The cache contents are as shown below

| Set # | Way #0 | Way #1 |
|---|---|---|
| 0: | V=1;Tag=0x0; Data = 0xa9 0x21 0xca 0x59 0x2f 0x0a 0xf0 0xa9 0x1f 0x68 0x60 0x07 0x38 0x56 0x35 0x57 | V=1;Tag=0xe; Data = 0x94 0x49 0x4e 0xcf 0xd7 0xe0 0x5b 0x70 0x05 0xc6 0x5b 0xf6 0xcb 0x5c 0x19 0xcc |
| 1: | V=1;Tag=0xe; Data = 0x5d 0xed 0x2f 0x9e 0x20 0x25 0xf8 0x41 0x93 0x0b 0x13 0x1d 0x74 0x4f 0x21 0x60 | V=1;Tag=0xb; Data = 0x8f 0x86 0x20 0x48 0x23 0xb3 0x1b 0x18 0xe4 0x95 0xe7 0xc5 0xc0 0xa3 0xb5 0x67 |
| 2: | V=1;Tag=0x5; Data = 0x6d 0x5a 0xe7 0xcf 0xe8 0x24 0x6e 0x7b 0xe7 0x7e 0x91 0x57 0xbe 0xc0 0xc7 0xd5 | V=1;Tag=0x7; Data = 0xf3 0x06 0x35 0x29 0x06 0x5c 0x2c 0x4c 0x55 0x5d 0x16 0x91 0xb0 0x1b 0x32 0xb5 |
| 3: | V=1;Tag=0x4; Data = 0x5e 0x6c 0x1a 0x82 0x9c 0x2f 0x53 0xd4 0xf1 0x20 0x4f 0x1f 0xb8 0x56 0x0e 0x42 | V=1;Tag=0xa; Data = 0x9f 0x92 0xe6 0xd9 0x95 0x32 0x14 0x54 0x6e 0x8f 0x65 0x1a 0xdf 0x6f 0xad 0x75 |
| 4: | V=1;Tag=0x9; Data = 0xf2 0x52 0xb6 0xe5 0x45 0x1b 0x03 0x73 0xdb 0xb7 0x09 0x7d 0xf5 0x6d 0xd4 0xab | V=1;Tag=0x6; Data = 0xdf 0xdd 0x18 0x94 0xbb 0x0d 0x7c 0xae 0x13 0x0b 0x08 0x66 0x1f 0x42 0x8d 0xbc |
| 5: | V=1;Tag=0xa; Data = 0xa6 0x68 0xb5 0x5b 0xc6 0xa4 0xbb 0x1a 0xc8 0xca 0xaf 0xe4 0x34 0x69 0x12 0x87 | V=0;Tag=0xe; Data = -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- |
| 6: | V=0;Tag=0x0; Data = -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- | V=1;Tag=0x4; Data = 0x29 0xbd 0x61 0x67 0x98 0xa5 0xfe 0x72 0xad 0x0f 0xca 0x8c 0xa9 0x1d 0xd2 0x9d |
| 7: | V=1;Tag=0x4; Data = 0x44 0x8a 0x45 0xb7 0x8f 0x71 0x49 0x57 0xcb 0xb5 0x61 0xc8 0x91 0x87 0x78 0xfa | V=1;Tag=0xd; Data = 0x39 0x1b 0x03 0x71 0xd4 0x53 0x30 0x8b 0x92 0xf2 0xeb 0x96 0x51 0xb8 0xbc 0x3f |
| 8: | V=1;Tag=0xb; Data = 0x71 0x3e 0x1e 0xf7 0x91 0x05 0x60 0xd7 0x65 0x5a 0x72 0x0f 0x3e 0x12 0x4d 0x52 | V=1;Tag=0x4; Data = 0x1e 0x8a 0xc0 0xd0 0x5e 0x2e 0xa2 0xd1 0x14 0x42 0xbf 0x96 0x2e 0x76 0xd6 0x31 |
| 9: | V=1;Tag=0x1; Data = 0xb0 0x10 0xf7 0x74 0x86 0xf6 0xdb 0xf4 0xde 0x29 0x89 0x26 0x2b 0xa7 0xf8 0x06 | V=1;Tag=0x6; Data = 0xd2 0x28 0x62 0x0f 0x1c 0x42 0xb9 0x74 0x5e 0xae 0xb3 0x6c 0x20 0x75 0x11 0x24 |
| 10: | V=1;Tag=0xb; Data = 0xd5 0xf2 0xe2 0x25 0x28 0x00 0xb4 0xc5 0x61 0x57 0x05 0xa4 0xf4 0xb7 0x2a 0x06 | V=1;Tag=0x3; Data = 0x9f 0xc2 0xa9 0xe5 0x1a 0x68 0x61 0xe3 0x51 0x71 0x91 0x9a 0xc0 0x63 0x38 0x2e |

| | | |
|---|---|---|
| 11: | V=1;Tag=0xd; Data = 0x09 0x2d 0x5c 0x7d 0x92 0x57 0x76 0x79 0x59 0xf4 0xef 0x87 0x66 0xbc 0x92 0xc5 | V=1;Tag=0x7; Data = 0x4e 0x6c 0xa1 0x7e 0x5e 0x3a 0x13 0xcc 0x8d 0x19 0x52 0x0d 0xa7 0x47 0x67 0xf3 |
| 12: | V=1;Tag=0x4; Data = 0xdb 0x42 0x31 0xd6 0x16 0x08 0x67 0x84 0x44 0x7b 0x9b 0xed 0x7b 0x43 0x35 0x0f | V=1;Tag=0x0; Data = 0xa0 0x6c 0x09 0x3e 0x37 0x8c 0xa5 0x9f 0xdc 0xc0 0xda 0xcb 0xad 0xc0 0x34 0xaa |
| 13: | V=1;Tag=0x4; Data = 0x6b 0xcc 0x0e 0x82 0xba 0x66 0xf1 0xe1 0xfd 0xd7 0x68 0xc6 0xf9 0xd0 0xe7 0x08 | V=1;Tag=0x7; Data = 0x7e 0x6a 0x21 0x05 0xe6 0x69 0x7a 0x99 0x52 0x7d 0x4c 0xe9 0xaf 0x6b 0x8b 0xd7 |
| 14: | V=1;Tag=0x2; Data = 0x49 0xc2 0x78 0x78 0x3d 0x27 0x2d 0x3b 0xfd 0x77 0xe2 0x08 0x73 0x69 0x0f 0xe0 | V=1;Tag=0x7; Data = 0xbc 0x28 0xad 0x73 0xd9 0xd3 0x13 0x0a 0x3a 0xe4 0xd0 0x51 0xbf 0xb9 0xe1 0x42 |
| 15: | V=1;Tag=0x4; Data = 0xc5 0x35 0x16 0x7e 0xd4 0xa0 0x2f 0x16 0xe9 0x8b 0x92 0x3a 0x56 0xd5 0x39 0x5c | V=1;Tag=0x8; Data = 0xc2 0xcc 0x20 0xce 0xa3 0x74 0xb0 0x55 0xea 0xda 0x22 0xbd 0x94 0xf0 0x85 0xb5 |

Assume that memory address **0x43a** has been referenced by a load instruction. 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 the "Cache Byte Returned". For values that need a hexidecimal value, do not enter leading zeros even if leading zeros are shown in the value above.

Cache block Offset (CO) 0x ✖

Cache set index (CI) 0x ✖

Cache tag (CT) 0x ✖

Cache hit (Y/N)? ✖

Cache byte returned 0x ✖

Question **12**

Not answered

Marked out of 5.00

Assume the following:

- . 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 2-way associative cache (E=2), with a 8-byte block size (B=8) and 16 sets (S=16).
- The cache contents are as shown below

| Set # | Way #0 | Way #1 |
|---|---|---|
| 0: | V=1;Tag=0x05; Data = 0x46 0x86 0x7c 0x9d 0x39 0x8e 0x88 0xa5 | V=1;Tag=0x07; Data = 0x2f 0x24 0x1d 0x60 0x85 0x9e 0xdc 0x4a |
| 1: | V=1;Tag=0x18; Data = 0xc9 0x8a 0x00 0x89 0xd2 0x0a 0x1d 0xc5 | V=1;Tag=0x1d; Data = 0x87 0x7b 0xdb 0x3d 0x6d 0x62 0x30 0x8f |
| 2: | V=1;Tag=0x04; Data = 0x5c 0x91 0x55 0x72 0x06 0x17 0x61 0xdb | V=1;Tag=0x0f; Data = 0x94 0x91 0x93 0xe3 0xe9 0x50 0x21 0xdb |
| 3: | V=0;Tag=0x19; Data = -- -- -- -- -- -- -- -- | V=1;Tag=0x0d; Data = 0x73 0xdc 0x12 0xb5 0xab 0xe0 0xae 0x7a |
| 4: | V=1;Tag=0x0e; Data = 0xf1 0x30 0x07 0xbd 0xcc 0xc7 0xe1 0x47 | V=1;Tag=0x00; Data = 0x58 0x42 0xf0 0xb2 0x06 0x01 0xb3 0xbf |
| 5: | V=1;Tag=0x01; Data = 0xcf 0x2d 0xa2 0x5f 0xc4 0xf0 0x62 0xe3 | V=1;Tag=0x02; Data = 0x4d 0x8f 0x35 0x5c 0x36 0x89 0xa0 0x2d |
| 6: | V=1;Tag=0x08; Data = 0x19 0xac 0x63 0x09 0xf7 0xfa 0xfc 0xc4 | V=1;Tag=0x1a; Data = 0x7c 0xba 0xd1 0x8e 0x3b 0xd8 0xe3 0x13 |
| 7: | V=1;Tag=0x1b; Data = 0x7a 0x75 0x0e 0xea 0xd4 0xe9 0xbf 0xeb | V=1;Tag=0x11; Data = 0xf0 0x18 0x6e 0x37 0x92 0xae 0x94 0x20 |
| 8: | V=0;Tag=0x0d; Data = -- -- -- -- -- -- -- -- | V=1;Tag=0x1d; Data = 0x97 0x2e 0x01 0x56 0x43 0xba 0x08 0x22 |
| 9: | V=1;Tag=0x12; Data = 0x5b 0x86 0x88 0x85 0x06 0x43 0x1d 0xe4 | V=1;Tag=0x09; Data = 0xd1 0x8d 0xf1 0x94 0x15 0xc3 0xb8 0xea |
| 10: | V=1;Tag=0x17; Data = 0x45 0x37 0x31 0xa8 0xf2 0x3d 0xd6 0x96 | V=1;Tag=0x15; Data = 0x16 0xce 0x33 0x2b 0x7f 0x27 0x83 0x80 |
| 11: | V=0;Tag=0x04; Data = -- -- -- -- -- -- -- -- | V=1;Tag=0x0d; Data = 0xcb 0xe2 0xcc 0x7a 0x2e 0x54 0x2f 0x97 |
| 12: | V=0;Tag=0x11; Data = -- -- -- -- -- -- -- -- | V=1;Tag=0x0d; Data = 0xf3 0x8f 0xd8 0x25 0x5c 0x8a 0x42 0x98 |
| 13: | V=1;Tag=0x1c; Data = 0xc4 0xa5 0xe1 0x15 0x8f 0x60 0xfc 0x13 | V=1;Tag=0x02; Data = 0x9a 0x27 0x5b 0x39 0xd1 0x10 0x0a 0x1a |
| 14: | V=1;Tag=0x01; Data = 0xc9 0x93 0x5e 0xc0 0xe0 0x93 0x0c 0x4d | V=1;Tag=0x16; Data = 0x41 0x99 0x54 0xc8 0x04 0x81 0x0f 0xf1 |
| 15: | V=1;Tag=0x03; Data = 0xaf 0xdc 0x20 0x57 0xa1 0x93 0x67 0xfe | V=1;Tag=0x1a; Data = 0x90 0x4f 0x66 0x95 0xe4 0xcb 0xd9 0x2a |

Assume that memory address **0x216** has been referenced by a load instruction. 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 the "Cache Byte Returned". For values that need a hexidecimal value, do not enter leading zeros even if leading zeros are shown in the value above.

Cache block Offset (CO) 0x ✖

Cache set index (CI) 0x ✖

Cache tag (CT) 0x ✖

Cache hit (Y/N)? ✖

Cache byte returned 0x ✖