CSPB 2400 - Park - Computer Systems

Started on	Thursday, 14 March 2024, 9:02 PM
State	Finished
Completed on	Thursday, 14 March 2024, 9:29 PM
Time taken	26 mins 22 secs
Marks	43.00/43.00
Grade	**10.00** out of 10.00 (**100**%)

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	13 ms	640

You may use an expression if that's useful.

13 + 0.5 * (60 / 15000) * 1

Your last answer was interpreted as follows: $13+0.5\cdot\left(\frac{60}{15000}\right)\cdot 1000+\frac{60}{15000}\cdot\left(\frac{1}{640}\right)\cdot 1000$

Correct answer, well done.
That's 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 $T_{\text{seek}}=13$, 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 $T_{\text{avg transfer}}=(60/15000)*(1/640)\text{sectors/track}*1000\text{ms/s}$.

The total access time is thus the sum or $\frac{2401}{160}$ = 15.00625.

A correct answer is $\frac{2401}{160}$, which can be typed in as follows: `2401/160`

Question **2**

Correct

Mark 4.00 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 14ms (*i.e.* the seek and rotational delay sums to 14ms) and a throughput or transfer rate of 125MBytes/s, where a megabyte is measured as 1024^2.

A solid-state drive (SSD) has an average access time of 0.035ms and a throughput of 630MB/s -- the access time serves the same role as the combined "seek" and "rotational" 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 190MB file.

How many "movies per second" can be processed by the hard disk drive? 1 / (14e-3 + 190 / 125)

Your last answer was interpreted as follows: $\frac{1}{0.014+\frac{190}{125}}$

Correct answer, well done.
Exactly!

How many "movies per second" can be processed by the SSD disk drive? 1 / (3.5e-5 + 190 / 630)

Your last answer was interpreted as follows: $\frac{1}{3.5e-5+\frac{190}{630}}$

Correct answer, well done.
Close enough!

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? 1 / (14e-3 + 512 / (125 * 1

Your last answer was interpreted as follows: $\frac{1}{0.014+\frac{512}{125\cdot 1024^2}}$

Correct answer, well done.
Close enough!

How many "IOPS" can be processed by the SSD disk drive? 1 / (3.5e-5 + 512 / (630 *

Your last answer was interpreted as follows: $\frac{1}{3.5e-5+\frac{512}{630\cdot 1024^2}}$

Correct answer, well done.
Close enough!

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 190MB file is $T_{\text{fs}} = T_{\text{seek}} + (190/\text{throughput})$. The speed in files per second is $\frac{1}{T_{\text{fs}}}$.

For the disk, this is $\frac{1}{\frac{14}{1000}+\frac{190}{125}} = \frac{500}{767} = 0.651890482399.$

For the ssd, this is $\frac{1}{\frac{\frac{7}{200}}{1000}+\frac{190}{630}} = \frac{12600000}{3800441} = 3.31540471224.$

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 $\frac{1io}{14ms * \frac{1s}{1000ms}} = \frac{500}{7} = 71.4285714286.$

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 $\frac{500}{767}$, which can be typed in as follows: `500/767`

A correct answer is $\frac{12600000}{3800441}$, which can be typed in as follows: `12600000/3800441`

A correct answer is $\frac{500}{7}$, which can be typed in as follows: `500/7`

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++) {       ✓  {
           A[i][j] = B[i][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;
    }
}
```

Not best and not worst locality	clear2	✓
Worst spatial locality	clear3	✓
Best spatial locality	clear1	✓

Your answer is correct.

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

Question **5**

Correct

Mark 4.00 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= 256 ✓ sets, t= 22 ✓ tag bits, s= 8 ✓ set index bits and b= 2 ✓ block offset bits.

Question **6**

Correct

Mark 4.00 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 64 ✓ bytes of memory (ignoring the tags and valid bits).

Question **7**

Correct

Mark 4.00 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 16 ✓ 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**

Correct

Mark 5.00 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 11 bits wide.
- . The cache is 4-way associative cache (E=4), with a 4-byte block size (B=4) 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=0x1d; Data = 0x9a 0xe5 0xeb 0xfb	V=1;Tag=0x0d; Data = 0x88 0xde 0xde 0x77	V=1;Tag=0x18; Data = 0x4c 0xe8 0x8d 0xda	V=1;Tag=0x15; Data = 0xe8 0x85 0xd6 0xba
1:	V=1;Tag=0x1d; Data = 0xf5 0x47 0x8b 0x63	V=1;Tag=0x16; Data = 0xdd 0x82 0x5f 0x4e	V=1;Tag=0x00; Data = 0x62 0x3e 0xb3 0x2e	V=1;Tag=0x11; Data = 0x52 0x7d 0xdd 0x39
2:	V=1;Tag=0x0f; Data = 0xc2 0x73 0x45 0xf2	V=1;Tag=0x1c; Data = 0x15 0xb8 0xb6 0xb4	V=1;Tag=0x13; Data = 0x7c 0x85 0x2c 0x41	V=1;Tag=0x06; Data = 0xc6 0xe9 0x14 0xe9
3:	V=1;Tag=0x0c; Data = 0x4d 0xd0 0x44 0x2b	V=1;Tag=0x14; Data = 0xc8 0x56 0x8a 0x11	V=1;Tag=0x0a; Data = 0xdf 0x4c 0x72 0x0d	V=1;Tag=0x18; Data = 0x92 0xee 0x4d 0x2c
4:	V=1;Tag=0x13; Data = 0x04 0xa3 0xbf 0xde	V=1;Tag=0x1c; Data = 0xb9 0x02 0x7b 0x4b	V=1;Tag=0x06; Data = 0x3b 0x9f 0x9e 0x64	V=1;Tag=0x03; Data = 0xb4 0xe7 0x1d 0xb9
5:	V=1;Tag=0x01; Data = 0x7e 0x94 0x6a 0x7c	V=1;Tag=0x08; Data = 0xec 0x1d 0x44 0x8e	V=0;Tag=0x04; Data = -- -- -- --	V=1;Tag=0x11; Data = 0xa8 0x5b 0x10 0x90
6:	V=1;Tag=0x10; Data = 0xe7 0xf5 0x31 0xf8	V=1;Tag=0x03; Data = 0xc1 0x03 0xc0 0xbd	V=1;Tag=0x1e; Data = 0x52 0xc3 0xe1 0x7c	V=1;Tag=0x0d; Data = 0x84 0xd9 0x22 0x97
7:	V=1;Tag=0x12; Data = 0xdc 0xbd 0x12 0xa6	V=1;Tag=0x02; Data = 0x76 0xad 0x87 0x37	V=1;Tag=0x0c; Data = 0xa0 0xdf 0x0a 0x91	V=0;Tag=0x1b; Data = -- -- -- --
8:	V=1;Tag=0x10; Data = 0x4a 0xe0 0x2c 0xeb	V=1;Tag=0x12; Data = 0x34 0x7c 0x75 0x24	V=1;Tag=0x1e; Data = 0x12 0xc8 0xd5 0x66	V=1;Tag=0x08; Data = 0xf2 0xc7 0xa6 0x5b
9:	V=1;Tag=0x19; Data = 0xc1 0xee 0x5b 0x63	V=1;Tag=0x09; Data = 0x6c 0xf1 0x87 0x52	V=1;Tag=0x0a; Data = 0xc0 0xe5 0x64 0x7f	V=0;Tag=0x18; Data = -- -- -- --
10:	V=1;Tag=0x05; Data = 0x90 0x1d 0x6d 0x12	V=0;Tag=0x1b; Data = -- -- -- --	V=1;Tag=0x19; Data = 0x49 0xf0 0xa4 0xdc	V=1;Tag=0x07; Data = 0xef 0x62 0x68 0x1f
11:	V=1;Tag=0x0f; Data = 0xcf 0x56 0x00 0xa8	V=1;Tag=0x1a; Data = 0x52 0x73 0x24 0x96	V=1;Tag=0x19; Data = 0x9a 0xf1 0xd7 0x67	V=1;Tag=0x1d; Data = 0xcc 0x6f 0x10 0xb8
12:	V=1;Tag=0x08; Data = 0xe5 0x03 0xfc 0x3b	V=0;Tag=0x1d; Data = -- -- -- --	V=0;Tag=0x09; Data = -- -- -- --	V=1;Tag=0x0c; Data = 0x4e 0x97 0x67 0x50
13:	V=1;Tag=0x05; Data = 0xb5 0xc4 0x27 0x23	V=1;Tag=0x1c; Data = 0x3a 0x78 0xfb 0xd9	V=1;Tag=0x0c; Data = 0x62 0x41 0x46 0x22	V=0;Tag=0x17; Data = -- -- -- --
14:	V=1;Tag=0x0a; Data = 0xe2 0x0b 0x3c 0x43	V=1;Tag=0x02; Data = 0xda 0x65 0xb4 0xb1	V=1;Tag=0x15; Data = 0xe4 0xb4 0x5f 0x91	V=1;Tag=0x05; Data = 0x9c 0x28 0xe8 0xeb
15:	V=1;Tag=0x1c; Data = 0xd4 0x73 0x37 0x4c	V=1;Tag=0x15; Data = 0x25 0x57 0x3f 0x83	V=1;Tag=0x0b; Data = 0x37 0xb7 0x1e 0xd7	V=1;Tag=0x16; Data = 0x53 0x87 0xe7 0x72

Assume that memory address **0x421** 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 1	✓
Cache set index (CI)	0x 8	✓
Cache tag (CT)	0x 10	✓

Cache hit (Y/N)?	yes ✓
Cache byte returned	0x e0 ✓

Question **11**

Correct

Mark 5.00 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 11 bits wide.
- . The cache is 4-way associative cache (E=4), with a 16-byte block size (B=16) and 4 sets (S=4).
- The cache contents are as shown below

Set #	Way #0	Way #1	Way #2	Way #3
0:	V=1;Tag=0x04; Data = 0x9e 0x38 0x11 0xca 0x6f 0xdf 0xfd 0xa1 0xeb 0xfd 0xd2 0xa7 0x31 0x5d 0xd8 0xaf	V=1;Tag=0x07; Data = 0x77 0xdf 0x59 0x13 0xf6 0x20 0x45 0xc5 0x9f 0x71 0x39 0x3d 0x54 0x97 0x8b 0xb6	V=1;Tag=0x1e; Data = 0xc5 0xdd 0x10 0xfd 0x5d 0xdb 0xa9 0x97 0xd7 0xa3 0x67 0xd5 0x52 0x38 0xd0 0x00	V=1;Tag=0x02; Data = 0x4c 0xa2 0x18 0x8c 0x72 0x89 0xbe 0x04 0x66 0x87 0x89 0x3c 0x10 0xb0 0x74 0x61
1:	V=1;Tag=0x10; Data = 0x54 0xec 0x79 0x2c 0x5d 0xd4 0x33 0x9b 0xa0 0xeb 0x09 0xd9 0x24 0xeb 0x3e 0xf6	V=0;Tag=0x0b; Data = -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --	V=1;Tag=0x1a; Data = 0x0b 0x5d 0xb2 0x66 0x4d 0xde 0xca 0x7e 0x33 0x6d 0xd3 0x37 0xb5 0x21 0xf5 0x03	V=1;Tag=0x11; Data = 0x74 0x7e 0xf0 0x7a 0x3d 0x9c 0x4a 0xdc 0x0d 0xfe 0xbd 0x3a 0xdd 0x00 0x6c 0x95
2:	V=1;Tag=0x1b; Data = 0x09 0x19 0x8d 0xa4 0xc8 0xd2 0x1f 0xdc 0x1e 0x4a 0xee 0x30 0x89 0x37 0x87 0x8f	V=1;Tag=0x0d; Data = 0x68 0x19 0x6a 0x02 0x33 0x43 0x44 0xf8 0x3d 0xc7 0xdb 0xb2 0x6d 0xaa 0xfa 0xdd	V=1;Tag=0x0e; Data = 0x0d 0xba 0x2a 0xb1 0x00 0x99 0x31 0xe3 0xeb 0xf4 0xd8 0xda 0x4c 0xb4 0x97 0xb1	V=1;Tag=0x13; Data = 0x46 0x95 0x65 0xc1 0x5e 0x69 0xf7 0x10 0xba 0x1b 0x05 0x35 0x55 0x1b 0xb3 0x79
3:	V=1;Tag=0x0a; Data = 0x4a 0x1b 0x7a 0x45 0x42 0xd9 0xe0 0xdf 0xef 0xbf 0xea 0xd9 0x74 0x90 0x04 0x27	V=1;Tag=0x04; Data = 0x25 0x6f 0x26 0xca 0x98 0xfe 0x7a 0x78 0xaa 0x3d 0xcf 0xe2 0x0e 0x00 0x99 0x3e	V=0;Tag=0x05; Data = -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --	V=1;Tag=0x16; Data = 0x7d 0x33 0xd6 0x70 0x63 0xfd 0xcc 0x20 0x77 0xce 0xf0 0x72 0x19 0xeb 0x7b 0x08

Assume that memory address **0x2db** 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 b ✓
Cache set index (CI)	0x 1 ✓
Cache tag (CT)	0x b ✓
Cache hit (Y/N)?	no ✓
Cache byte returned	0x - ✓

Question **12**

Correct

Mark 5.00 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 10 bits wide.
- . The cache is direct mapped cache (E = 1), with a 8-byte block size (B=8) and 8 sets (S=8).
- The cache contents are as shown below

Set #	**Way #0**
0:	V=1;Tag=0x0; Data = 0x4d 0x3e 0x3f 0xc6 0x40 0x86 0x5c 0x66
1:	V=1;Tag=0xb; Data = 0xd8 0xaf 0xf5 0x76 0x0b 0x7d 0xe0 0x6e
2:	V=1;Tag=0xd; Data = 0xc8 0xb7 0x73 0x57 0x79 0x40 0x3c 0x6a
3:	V=1;Tag=0x6; Data = 0x58 0x85 0x47 0x8f 0x38 0x01 0x34 0xc5
4:	V=1;Tag=0x8; Data = 0xdf 0x81 0x51 0x39 0xf4 0x36 0x4f 0x87
5:	V=1;Tag=0x1; Data = 0xe0 0xc5 0x66 0xf0 0x08 0x3b 0x2b 0xa6
6:	V=1;Tag=0x8; Data = 0x5e 0xab 0x8c 0x6b 0x99 0xbd 0xe6 0x41
7:	V=1;Tag=0x4; Data = 0x0f 0x8c 0xed 0x51 0xe8 0x37 0x80 0x11

Assume that memory address **0x40** 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 0	✓
Cache set index (CI)	0x 0	✓
Cache tag (CT)	0x 1	✓
Cache hit (Y/N)?	no	✓
Cache byte returned	0x -	✓