Full Name:________________________________

EECS 213 Fall 2010
Final Exam

1. (10 points):

Given a 32-bit address machine with a 2048 byte cache, fill in for each block size (B) and lines per set (E) as specified below the number of cache sets (S), tag bits (t), set index bits (s), and block offset bits (b). Give answers in decimal.

B	E	S	t	s	b
8	4				
8	256				
16	1				
16	128				
32	1				
32	4				

2. (10 points):

Describe the sequence of events in this program and its output. Assume the necessary includes are done and all functions return normally.

```
pid_t pid, cid;

void handler1(int sig) {
  printf("zip");
  fflush(stdout); /* Flushes the printed string to stdout */
  kill(cid, SIGUSR1);
}

void handler2(int sig) {
  printf("zap");
  exit(0);
}

main() {
  signal(SIGUSR1, handler1);
  pid = getpid();
  if ((cid = fork()) == 0) {
    signal(SIGUSR1, handler2);
    kill(pid, SIGUSR1);
    while(1) {};
  }
  else {
    int status;
    if (wait(&status) > 0) {
      printf("zoom");
    }
  }
}
```

3. (12 points):

Consider an allocator that uses an implicit free list, where the layout of each memory block is

- 32 bit header, with size and use bits
- the payload
- 32 bit footer, with size and use bits

The size of each memory block is a multiple of eight bytes, including the header and footer. The low-order bit 0 of the size is set to 1 if the block is allocated, 0 if free.

Implement the code as indicated by the comments. Make use of previous results and the function `size()` when possible.

C note: pointer arithmetic is valid only on pointers to actual types, not `void *`.

```
void *p = malloc(20);

void *hp = ____________________________;  /* hp points to the header of the block containing p */

void *fp = ____________________________;  /* fp points to the footer of the block containing p */

int used = ____________________________;  /* used = 1 if the block is  allocated, 0 if free */

void *pp = ____________________________;  /* pp points to the header of prior  block in memory *

/* return the  size in the header pointed to by hp */

int size(void *hp) { return ____________________________;    }
```

4. (10 points)

Assume the heap structure of the previous problem. In the diagrams below, each rectangle is a 32-bit word. On the left, block 2 is free, the rest are in use. Fill in the header values in hex. On the right, show the heap after block 3 is freed, and coalescing is done.

0x____	0x____
1	1
0x____	0x____
0x____	0x____
2	2
0x____	0x____
0x____	0x____
3	3
0x____	0x____
0x____	0x____
4	4
0x____	0x____

5. (10 points)

Given:

- Memory accesses are to **1-byte words** (not 4-byte words).
- Physical addresses are 13 bits wide.
- The cache below:

2-way Set Associative Cache													
Index	Tag	Valid	Byte 0	Byte 1	Byte 2	Byte 3	Tag	Valid	Byte 0	Byte 1	Byte 2	Byte 3
0	09	1	86	30	3F	10	00	0	99	04	03	48
1	45	1	60	4F	E0	23	38	1	00	BC	0B	37
2	EB	0	2F	81	FD	09	0B	0	8F	E2	05	BD
3	06	0	3D	94	9B	F7	32	1	12	08	7B	AD
4	C7	1	06	78	07	C5	05	1	40	67	C2	3B
5	71	1	0B	DE	18	4B	6E	0	B0	39	D3	F7
6	91	1	A0	B7	26	2D	F0	0	0C	71	40	10
7	46	0	B1	0A	32	0F	DE	1	12	C0	88	37

Clearly label the address diagram below with the fields for the cache offset (**CO**),cache index (**CI**) and cache tag (**CT**).

12	11	10	9	8	7	6	5	4	3	2	1	0

Fill in the bits for this address. : 0x**0E34**

12	11	10	9	8	7	6	5	4	3	2	1	0

For this address, fill in the table below. Use hex where indicated. Indicate whether a cache miss occurs. If so, enter "-" for "Byte returned".

Parameter	Value
Cache Offset	0x_______
Cache Index	0x_______
Cache Tag	0x_______
Cache Hit? (Y/N)	
Byte returned	0x_______

Problem 6. (10 points):

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 640x480 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:

```
struct pixel {
   char r;
   char g;
   char b;
   char a;
};

struct pixel buffer[480][640];

register int i, j;
register char *cptr;
register int *iptr;
```

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

A. What percentage of the writes in the following code will miss in the cache?

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

Miss rate for writes to buffer: _______ %

B. What percentage of the writes in the following code will miss in the cache?

```
      char *cptr;
      cptr = (char *) buffer;
      for (; cptr < (((char *) buffer) + 640 * 480 * 4); cptr++)
      *cptr = 0;
```

Miss rate for writes to buffer: _______ %

C. What percentage of the writes in the following code will miss in the cache?

```
      int *iptr;
      iptr = (int *) buffer;
      for (; iptr < (buffer + 640 * 480); iptr++)
      *iptr = 0;
```

Miss rate for writes to buffer: _______ %

D. Which code (A, B, or C) should be the fastest? _______