Full Name: ________________________________

CSC 374 Computer Systems II Midterm Exam

Winter Quarter

February 6th, 2018

Instructions

- Make sure that your exam is not missing any sheets, and then write your full name on the front.
- Write your answers in the space provided below the problem. Clearly indicate your final answer.
- The exam has a maximum score of 64 points.
- The problems are of varying difficulty. The point value of each problem is indicated. Pile up the easy points quickly and then come back to the harder problems.
- You are allowed 1 sheet of notes and a calculator.
- This exam is closed book.
- Good Luck!

Problem (Points)	Score
1 (6):	
2 (7):	
3 (8):	
4 (10):	
5 (5):	
6 (8):	
7 (8):	
8 (4):	
9 (8):	
Total (64):	

Problem 1 (6 points):

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	1				
2.	32	1024	4	4				
3.	32	1024	8	128				
4.	32	1024	16	16				
5.	32	1024	32	1				
6.	32	1024	32	8				

Problem 2 (7 points)

The following problem concerns basic cache lookups.

- The memory is byte addressable.
- Memory accesses are to 1-byte words (not 4-byte words).
- Physical addresses are 12 bits wide.
- The cache is 2-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:

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	A7	1	86	30	3F	10	F2	1	99	04	03	48
1	5A	1	60	4F	E0	23	B4	0	00	BC	0B	37
2	E1	0	2F	81	FD	09	90	1	8F	E2	05	BD
3	36	1	3D	94	9B	F7	0A	0	12	08	7B	AD

Part 1

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:

CO	The block offset within the cache line
CI	The cache index
CT	The cache tag

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

Part 2

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: `0xF21`

A. Physical address (one bit per box)

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

B. Physical memory reference

Parameter	Value
Byte offset	0x
Cache Index	0x
Cache Tag	0x
Cache Hit? (Y/N)	
Cache Byte returned	0x

Part 3

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: `0x5AA`

A. Physical address (one bit per box)

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

B. Physical memory reference

Parameter	Value
Byte offset	0x
Cache Index	0x
Cache Tag	0x
Cache Hit? (Y/N)	
Cache Byte returned	0x

Problem 3 (8 points)

A bitmap image is composed of pixels. Each pixel in the image is represented as four values: three for the primary colors (red, green and blue - RGB) and one for the transparency information defined as an alpha channel.

In this problem, you will compare the performance of direct mapped and 4-way associative caches for a square bitmap image initialization. Both caches have a size of 128 bytes. The direct mapped cache has 16-byte blocks while the 4-way associative cache has 8-byte blocks.

You are given the following definitions

```
typedef struct{
     unsigned char r;
     unsigned char g;
     unsigned char b;
     unsigned char a;
} pixel_t;

pixel_t pixel[16][16];
int i, j;
```

Also assume that

- `sizeof(unsigned char) = 1`
- `pixel` begins at memory address 0
- Both caches are initially empty
- The array is stored in row-major order
- Variables `i` and `j` are stored in registers and any access to these variables does not cause a cache miss

A. What fraction of the writes in the following code will result in a miss in the direct mapped cache?

```
for (i = 0; i < 16; i ++){
     for (j = 0; j < 16; j ++){
          pixel[i][j].r = 0;
          pixel[i][j].g = 0;
          pixel[i][j].b = 0;
          pixel[i][j].a = 0;
     }
}
```

Miss rate for writes to pixel:__________%

B. Using code in part A, what fraction of the writes will result in a miss in the 4-way associative cache?

Miss rate for writes to pixel: __________ %

C. What fraction of the writes in the following code will result in a miss in the direct mapped cache?

```
for (i = 0; i < 16; i ++){
     for (j = 0; j < 16; j ++){
          pixel[j][i].r = 0;
          pixel[j][i].g = 0;
          pixel[j][i].b = 0;
          pixel[j][i].a = 0;
     }
}
```

Miss rate for writes to pixel:__________%

Part D

Using code in part C, what fraction of the writes will result in a miss in the 4-way associative cache?

Miss rate for writes to pixel:__________%

Problem 4 (10 points)

After a stressful quarter, you suddenly realize that you haven't bought a single Christmas present. Fortunately, you see that an online retailer has designer scarves on sale. You don't have much time to decide which scarf will make the best present for which friend, so you decide to automate the decision process. For that, you use a database containing an entry for each of your friends. It is implemented as an `8x8` matrix using a data structure, `person`. You add to this data structure a field for each scarf that you consider:

```
struct person{
     char name[16];
     int age;
     int male;
     short ravenclaw;
     short hufflepuff;
     short slytherin;
     short gryffindor;
}

struct person db[8][8];
int i, j;
```

Part 1

After thinking for a while you come up with the following smart routine that finds the ideal present for everyone.

```
void generate_presents(){

     for (j=0; j<8; j++){
          for (i=0; i<8; i++) {
               db[i][j].ravenclaw = 0;
               db[i][j].hufflepuff = 0;
               db[i][j].slytherin = 0;
               db[i][j].gryffindor = 0;
          }
     }
     for (j=0; j<8; j++){
          for (i=0; i<8; i++) {
               if(db[i][j].age < 30){
                    if(db[i][j].male)
                         db[i][j].ravenclaw = 1;
                    else
                         db[i][j].hufflepuff = 1;
               }
               else{
                    if(db[i][j].male)
                         db[i][j].slytherin = 1;
                    else
                         db[i][j].gryffindor = 1;
               }
          }
     }
}
```

Of course, runtime is important in this time-critical application, so you decide to analyze the cache performance of your routine.

You assume that

- Your machine has a 512-byte direct-mapped data cache with 128 byte blocks.
- `db` begins at memory address 0
- The cache is initially empty.
- The only memory accesses are to the entries of the array `db`. Variables `i` and `j` are stored in registers.
- `sizeof(person) = 32`

Answer the following questions:

A. What is the total number of read and write accesses? ______.

B. What is the total number of read and write accesses that miss in the cache? ______

C. So the fraction of all accesses that miss in the cache is: ______.

Part 2

Then you consider the following alternative implementation of the same algorithm:

```
void generate_presents(){
        for (i=0; i<8; i++){
                for (j=0; j<8; j++) {
                        if(db[i][j].age < 30) {
                                if(db[i][j].male) {
                                        db[i][j].ravenclaw=0;
                                        db[i][j].hufflepuff=1;
                                        db[i][j].slytherin=0;
                                        db[i][j].gryffindor=0;
                                }
                                else {
                                        db[i][j].ravenclaw=1;
                                        db[i][j].hufflepuff=0;
                                        db[i][j].slytherin=0;
                                        db[i][j].gryffindor=0;
                                }
                        }
                        else {
                                if(db[i][j].male) {
                                        db[i][j].ravenclaw=0;
                                        db[i][j].hufflepuff=0;
                                        db[i][j].slytherin=0;
                                        db[i][j].gryffindor=1;
                                }
                                else{
                                        db[i][j].ravenclaw=0;
                                        db[i][j].hufflepuff=0;
                                        db[i][j].slytherin=1;
                                        db[i][j].gryffindor=0;
                                }
                        }
                }
        }
}
```

Making the same assumptions as in Part 1, answer the following questions.

A. What is the total number of read and write accesses? ______

B. What is the total number of read and write accesses that miss in the cache? ______

C. So the fraction of all accesses that miss in the cache is: _______

Problem 5 (5 points)

A. Briefly describe the purpose of Linking as part of the compilation and loading of programs.

B. Describe the difference between static and dynamic linking.

C. List at least 2 advantages of dynamic linking over static linking.

Problem 6 (8 points)

Consider the C program below. (For space reasons, we are not checking error return codes, so assume that all functions return normally.)

```
main() {

     if (fork() == 0) {
          if (fork() == 0) {
               printf("a");
          }
          else {
               pid_t pid;
               int status;
               if ((pid = wait(&status)) > 0) {
                    printf("b");
               }
          }
     }
     else {
          if (fork() == 0) {
               printf("c");
               exit(0);
          }
          printf("d");
     }

     printf("e");
     return 0;
}
```

For each of the 5 outputs listed below, circle only the valid outputs of this program. Assume that all processes run to normal completion.

A. `deabecd` Y N

B. `abcdeee` Y N

C. `daeecbe` Y N

D. `deaebec` Y N

E. `adeebce` Y N

Problem 7 (8 points)

This problem tests your understanding of exceptional control flow in C programs. For parts 1-4, indicate how many "`hello`" output lines the program would print. *Caution*: Don' t overlook the `printf` function in `main`.

Part 1

```
void doit() {
     fork();
     fork();
     printf("hello\n");
     return;
}

int main() {
     doit();
     printf("hello\n");
     exit(0);
}
```

Answer: ______ output lines.

Part 2

```
void doit() {
     if (fork() == 0) {
          fork();
          printf("hello\n");
          exit(0);
     }
     return;
}

int main() {
     doit();
     printf("hello\n");
     exit(0);
}
```

Answer: ______ output lines.

Part 3

```
void doit() {
     if (fork() == 0) {
          fork();
          printf("hello\n");
          return;
     }
     return;
}

int main() {
     doit();
     printf("hello\n");
     exit(0);
}
```

Answer: ______ output lines.

For part 4, indicate the value of the counter variable that the program would print.

Part 4

```
int counter = 1;

int main() {
     if (fork() == 0) {
          counter++;
          exit(0);
     }
     else {
          wait(NULL);
          counter--;
          printf("counter = %d\n", counter);
     }
     exit(0);
}
```

Answer: `counter =` ______.

Problem 8 (4 points)

Consider the following C program. (For space reasons, we are not checking error return codes. You can assume that all functions return normally.)

```
int val = 10;

void handler(sig){
     val -= 5;
     return;
}

int main(){

     int pid;

     signal(SIGCHLD, handler);
     if ((pid = fork()) == 0) {
          val -= 3;
          exit(0);
     }

     waitpid(pid, NULL, 0);

     val++;
     printf("val = %d\n", val);
     exit(0);
}
```

What is the output of this program? `val =` __________

Problem 9 (8 points)

This problem tests your understanding of file descriptor tables and what happens when you `fork` a child process. Suppose the disk file `sample.txt` consists of the six ASCII characters, `abcdef`. In each part below, read the program and list the possible outputs.

Part 1 Consider the following program:

```
int main() {
	int fd;
	char c;

	fd = open("sample.txt", O_RDONLY, 0);
	read(fd, &c, 1);
	printf("%c\n", c);

	if(fork() == 0) {
		read(fd, &c, 1);
		printf("%c\n", c);
		exit(0);
	}

	read(fd, &c, 1);
	close(fd);
	printf("%c\n", c);

	return(0);
}
```

List the possible outputs of the program in part 1. Briefly explain.

Part 2 Now consider another program:

```
int main() {
     int fd;
     char c;

     fd = open("sample.txt", O_RDONLY, 0);
     read(fd, &c, 1);
     printf("%c\n", c);

     if(fork() == 0) {
          fd = open("sample.txt", O_RDONLY, 0);

          read(fd, &c, 1);
          printf("%c\n", c);

          read(fd, &c, 1);
          printf("%c\n", c);

          close(fd);
          exit(0);
     }

     wait(NULL);

     read(fd, &c, 1);

     printf("%c\n", c);
     close(fd);

     return(0);
}
```

List the possible outputs of the program in part 2. Briefly explain.