Full Name _____

CSCI 2400, Fall 2017
Practice Midterm Exam 3

'On my honor as a University of Colorado at Boulder student I have neither given nor received unauthorized assistance on this work.'

Instructions:

- Check that your exam has all X pages, and write your full name clearly on the front.
- Write your answers in the space provided for each problem. Feel free to use the back of each page to help you determine the answer, but make sure your answer is entered in the space provided on the front of the page. Then in the last 15 minutes of the class, you will use your laptop to upload your answers to the Moodle.
- This exam is CLOSED BOOK and no electronics are allowed, except a laptop that is only to be used in the last 15 minutes of the exam to upload your answers to the Moodle. You can use one page of personal notes and the printed midterm packet of tables. Good luck!

Problem	Possible	Score
1		
2		
3		
4		
5		
Total		

1. **[X Points]** Look at the assembly instructions below. These instructions will be pipelined to speed up the code execution. Unfortunately the code cannot be pipelined due to data dependencies. Add 'NOP' instruction in the code below at the appropriate locations to ensure that pipelining can be accomplished. (Hint: draw out a timing diagram of the different pipeline stages)

```
mov $1729, %rcx
mov $4104, %rax
mov %rcx, %rax
add $1, %rcx
add %rax, %rcx
mov $5, %rbx
mov $10, %rdx
sub %rbx, %rdx
```

2. **[X Points]** The following problem concerns optimizing a procedure for maximum performance on a machine with the following characteristics of the functional units:

Operation	Latency	Issue Time/Rate
Integer Add	1	1
Integer Multiply	3	1
Floating Point Add	2	1
Floating Point Multiply	4	2
Load or Store (Cache Hit)	1	1

Assume there is one of each functional unit, array1 and array2 have the correct types, e.g. int or floating point. Assume input1, input2, input3, out1, and out2 can be stored in registers. Finally, you can ignore inc, cmpl and jmp instructions, but wherever applicable **do not** ignore load instructions.

(a) **[5 Points]**

```
float out1, out2, out3, input1, input2;
    for (i=0; i< length; i++){
        out1 = input1 * array1[i];
        out2 = out1 + array2[i];
        out3 = out2 + input2;
    }
```

What is the CPE of this loop?

(b) **[5 Points]**

```
int input1, input2, out1;
    for (i=0; i< length; i++){
        input1 = input1 * input1;
        out1 = input2 + array1[i];
    }
```

What is the CPE of this loop?

3. **[X Points]** Consider the following code:

```
typedef int array[2][2];

void invert_diagonal(array dst, array src)
{
  int i, j;
  for (i = 0; i < 2; i++) {
    for (j = 0; j < 2; j++) {
      dst[1-i][j] = src[1-j][i];
    }
  }
}
```

Assume this code runs on a machine with the following properties: sizeof(int) == 4; the src array starts at address 0 and the dst array starts at address 16 (decimal); there is a single L1 data cache that is direct-mapped, write-through, and write-allocate, with a block size of 8 bytes; the cache has a total size of 16 data bytes and the cache is initially empty; accesses to the src and dst arrays are the only sources of read and write misses, respectively.

For each row and col, indicate whether the access to src[row][col] and dst[row][col] is a hit or a miss.

dst array

	Column 0	Column 1
Row 0		
Row 1		

src array

	Column 0	Column 1
Row 0		
Row 1		

4. **[X Points]** 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 (Valid=1 means that the valid bit is set).
- For the Data column, the bytes are numbered from left to right as 0x0 to 0x9.

In the following tables, **all numbers are given in hexadecimal. Fill your answers in hex format**

Cache			
Set index	Valid	Tag	Data
0	1	0x08	0xc2 0x4c 0xb9 0x30 0x82 0x95 0x7d 0x7e 0xe4 0x10 0x24 0x8e 0x19 0x1a 0x95 0x90
	1	0x0a	0x97 0x68 0xc3 0x13 0x9a 0x9f 0x23 0xb5 0xf4 0xa0 0x1c 0x46 0x2a 0x8e 0x19 0x5f
	1	0x11	0x8e 0xb1 0xf2 0x21 0x58 0x14 0xba 0x72 0xea 0x19 0xab 0x2f 0x0e 0x0e 0x4a 0x31
	1	0x06	0x6a 0x99 0x24 0x8e 0x19 0x5f 0x77 0xed 0xa4 0x37 0xf2 0x21 0x58 0x21 0x58 0xdd
1	1	0x17	0x40 0x4a 0x31 0x73 0xf4 0x95 0x11 0xb6 0xf4 0x77 0x97 0x68 0xc3 0x13 0x40 0x4a
	0	0x15	– – – – – – – –
	1	0x13	0x2c 0x32 0xcd 0x04 0x93 0x81 0x6a 0x8a 0x40 0x88 0x80 0x3b 0x6a 0x8a 0x40 0xeb
	1	0x1b	0xf7 0x57 0x79 0xf6 0x09 0xa2 0x7d 0xf8 0x11 0xa2 0x6a 0x99 0x24 0x8e 0x19 0x5f
2	1	0x1c	0x46 0x2a 0xc3 0x7e 0x8f 0x83 0x3a 0x1b 0x13 0x1e 0x2f 0xae 0x49 0xf5 0x27 0x1b
	1	0x08	0x3e 0x21 0x63 0xf4 0xf6 0xd7 0xbe 0x11 0x12 0x20 0x63 0xf4 0x00 0xd8 0x4f 0xca
	0	0x0d	– – – – – – – –
	1	0x1a	0x95 0x90 0xb5 0xd8 0x4f 0xca 0x51 0x92 0x76 0x80 0x73 0xf4 0x95 0x11 0xb6 0x91
3	1	0x01	0x63 0x04 0xab 0x2f 0x0e 0x42 0x41 0x3e 0xa0 0x00 0xa0 0x00 0x12 0x23 0x34 0x99
	1	0x12	0x31 0x40 0x2f 0xae 0x49 0xf5 0x27 0x1b 0xff 0x00 0x21 0x58 0x14 0xba 0x72 0xea
	1	0x0e	0xd9 0x61 0xcd 0x92 0xbc 0xd3 0x26 0x03 0x7d 0x14 0x8a 0x40 0xeb 0x40 0xeb 0x11
	1	0x12	0xc0 0xac 0x80 0x3b 0xba 0xa8 0xc8 0xe9 0xdd 0x12 0xf7 0x52 0x89 0xf6 0x19 0xa2

(a) **[10 Points]** Assume that memory address 0x729 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

(b) **[10 Points]** Assume that memory address 0x4d5 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