This is a full length practice midterm exam. If you want to take it at exam pace, give yourself 75 minutes to take the entire test. Just like the real exam, each question has a point value. There are 75 points in the exam, so that you can pace yourself to average 1 point per minute (some parts will be faster, some slower).

Questions:

1. Numbers [10 points]
2. Binary Math [10 points]
3. UNIX Commands [10 points]
4. C Programming [15 points]
5. C Tracing [15 points]
6. ISA Concepts [10 points]
7. Asm Programming [15 points]
8. Asm Tracing [15 points]
Question 1: Numbers [10 pts]

1. Convert the decimal number -67 to 8-bit, signed 2’s complement binary:

2. Convert the 8-bit signed 2’s complement hex number 0x3F to decimal:

3. Convert the 16-bit signed 2’s complement binary number 1001 0001 1111 1010 to hex:
4. Convert -35.75 to its hexadecimal representation in IEEE floating point format:

5. Convert the hexadecimal IEEE format floating point number 0x40200000 to decimal:
Question 2: Binary Math [10 pts]

1. Add the 8-bit 2’s complement numbers 1110 1001 + 1100 0010. State the result of the addition (as a binary value), as well as whether overflow occurred if the number were treated as signed, and whether overflow occurred if the numbers were treated as unsigned. Show your work.

\[
\begin{array}{cccccccc}
1 & 1 & 1 & 0 & 1 & 0 & 0 & 1 \\
1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\
\hline
1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\
\end{array}
\]

Signed Overflow:

Unsigned Overflow:

2. What is 0110 1101 \(^{1100 1001}\)?
3. Add the 8-bit 2’s complement numbers 0110 1111 + 0110 1010. State the result of the addition (as a binary value), as well as whether overflow occurred if the number were treated as signed, and whether overflow occurred if the numbers were treated as unsigned. Show you work.

\[
\begin{array}{cccccccc}
0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\
0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 \\
\end{array}
\]

Sum:

Signed Overflow:

Unsigned Overflow:

4. What is 1100 0011 >> 0000 0011 when the operands are treated as signed integers?

5. What is 1100 0011 >> 0000 0011 when the operands are treated as unsigned integers?
**Question 3: UNIX Commands [10 pts]**

Match each of the following tasks with the UNIX program most suited to that task:

1. Compile a C program
   - A allfiles
   - B cp
   - C edit
   - D emacs
   - E gcc
   - F gdb
   - G less
   - H ls
   - I mkdir
   - J nethack
   - K newfldr
   - L rm
   - M scp
   - N ssh
   - O svn
   - P textwzrd

2. Copy a file to a different computer

3. Edit a text file

4. View the contents of a file (not to edit it)

5. Show what files are in the current directory
Question 4: C Programming [15 pts]
Given the following linked list node definition:

```c
struct ll_node {
    int data;
    struct ll_node * next;
};
typedef struct ll_node node;
```

Fill in the function below which finds the largest element in the list. If the list is empty, this function should return 0 (although, a non-empty list may contain only negative numbers, in which case, your code should function correctly).

```c
int findMax(node * head) {
```
Question 5: C Tracing [15 pts]

What is the output when the following C program is run?

```c
#include <stdio.h>

void f (int x, int * p) {
    *p = x;
    x = 33;
}

int main(void) {
    int a = 4;
    int b = 2;
    int * p = &a;
    *p = 55;
    printf("1: a = %d, b=%d\n", a, b);
    f(a,&b);
    printf("2: a = %d, b=%d\n", a, b);
    int ** q = & p;
    *q = & b;
    *p = 42;
    printf("3: a = %d, b=%d\n", a, b);
    return 0;
}
```
Question 6: ISA Concepts [10 pts]

Briefly answer each of the following:

1. ISA stands for “Instruction Set Architecture,” but what does it mean?

2. Describe the defining characteristics of an accumulator-style ISA:

3. What is an advantage of a 3-operand ISA over a 2-operand ISA? (Hint: you might give an example assembly sequence)
**Question 7: Asm Programming [15 pts]**
Write MIPS assembly for the following C function.

```c
int sumArray(int * ptr, int count) {
    int total = 0;
    while (count > 0) {
        total += *ptr;
        ptr ++;
        count--;
    }
    return count;
}
```

```
.globl sumArray
.ent sumArray
.text
sumArray:
```

```
.end sumArray
```
Question 8: Asm Tracing [15 pts]

Trace the following MIPS assembly (note the code is split into two columns to fit the page, but the top of the second column is sequentially after the bottom of the first):

```
start:  # 1000
    lw $t0, 0($sp)
    li $t1, 3
    mul $t0, $t0, $t1

a:  # 1010
    bne $a0, $a1, c

b:  # 1014
    sw $a0, 0($sp)

c:  # 1018
    li $a0, 42
    li $a1, 5
    sw $ra, 4($sp)
    jal xyz

d:  # 1028
    lw $ra, 4($sp)

e:  # 102C
    jr $ra

f:  # 1030
    li $v0, 10
    syscall   # exit

xyz:  # 1038
    add $v0, $a0, $a1
    addi $v0, $v0, 2
    jr $ra
```

The initial state (locationregisters/memory) is given in the table below. As you execute the above assembly, whenever you encounter a label (i.e., immediately before you execute the instruction right after the label), write down the label’s name, and the current register and memory values. You may not encounter all labels. The instruction addresses for each label are given in a comment on the same line as each label, if you find you need them. All values shown are in hex and you should also write hex values.

<table>
<thead>
<tr>
<th>Label</th>
<th>$v0</th>
<th>$a0</th>
<th>$a1</th>
<th>$t0</th>
<th>$t1</th>
<th>$sp</th>
<th>$ra</th>
<th>Memory</th>
<th>FFE0</th>
<th>FFE4</th>
</tr>
</thead>
<tbody>
<tr>
<td>start</td>
<td>0</td>
<td>1142</td>
<td>7FE5</td>
<td>FFFF</td>
<td>9999</td>
<td>FFE0</td>
<td>1234</td>
<td>3</td>
<td>42</td>
<td></td>
</tr>
</tbody>
</table>