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 [5 points]
3. UNIX Commands [5 points]
4. C Programming [15 points]
5. C Tracing [9 points]
6. ISA Concepts [6 points]
7. Asm Tracing [5 points]
8. Asm Programming [20 points]

This is the solution set to the practice exam. The solutions appear in blue boxes.
Question 1: Numbers [10 pts]

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

   **Answer:**
   First we need to convert 67 to binary.
   
   \[ 67 = 64 + 2 + 1 = 0100\,0011. \]
   
   Next we need to flip the bits (1011 1100) and add 1, to get the answer: 1011 1101.

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

   **Answer:**
   Easiest is to convert to binary: 0011\,1111.
   
   Then convert to decimal: \[ 32 + 16 + 8 + 4 + 2 + 1 = 63, \] which is the answer.

3. Convert the 16-bit signed 2’s complement binary number 1001 0001 1111 1010 to hex:

   **Answer:**
   Each group of 4 bits converts to 1 hex digit:
   
   \[ 1001 = 9 \]
   \[ 0001 = 1 \]
   \[ 1111 = F \]
   \[ 1010 = A \]
   
   So the answer is 0x91FA
4. Convert -35.75 to its hexadecimal representation in IEEE floating point format:

**Answer:**
First, we need to write down the binary representation of 35.75:
100011.11
Next, we need to put normalize it:
1.0001111 * 2^5
Now we can get the 3 pieces of the number we need:
S: 1 (negative)
E: 132 (127 + 5) = 1000 0100
M: 000 1111 0000 0000 0000 0000
Now we just put all the bits together:
1100 0010 0000 1111 0000 0000 0000 0000
and convert each group of 4 to a hex digit:
Answer: 0xC20F0000

5. Convert the hexadecimal IEEE format floating point number 0x40200000 to decimal:

**Answer:**
First, convert the hex to binary:
0100 0000 0010 0000 0000 0000 0000 0000 Then pull out each of the 3 pieces: S: 0 (positive)
E: 1000 0000 = 128. Taking 128 - 127 = 1
M: 010 0000 0000 0000 0000 0000
So we have 1.01 * 2^1 = 10.1
Then convert that to decimal to get the answer: 2.5
Question 2: Binary Math [5 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 you work.

```
  1 1
  1 1 1 1 0 0 1
  1 1 0 0 0 0 1 0
_______________
  1 0 1 0 1 1 1 1
```

**Sum:**

**Answer:**
1010 1011

**Signed Overflow:**

**Answer:**
No (CI=CO=1 for the last bit)

**Unsigned Overflow:**

**Answer:**
Yes (CO = 1)

2. What is 0110 1101 ^ 1100 1001?

**Answer:**
1010 0100
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.

```
1 1 1 1 1 1 1 1
0 1 1 0 1 1 1 1
0 1 1 0 1 0 1 0
```

Sum:

<table>
<thead>
<tr>
<th>Answer:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1101 1001</td>
</tr>
</tbody>
</table>

Signed Overflow:

<table>
<thead>
<tr>
<th>Answer:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes (CI=1, but CO=0 for the last bit)</td>
</tr>
</tbody>
</table>

Unsigned Overflow:

<table>
<thead>
<tr>
<th>Answer:</th>
</tr>
</thead>
<tbody>
<tr>
<td>No (CO = 0)</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Answer:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1111 1000</td>
</tr>
</tbody>
</table>

5. What is 1100 0011 >> 0000 0011 when the operands are treated as unsigned integers?

<table>
<thead>
<tr>
<th>Answer:</th>
</tr>
</thead>
<tbody>
<tr>
<td>0001 1000</td>
</tr>
</tbody>
</table>
### Question 3: UNIX Commands [5 pts]

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

<table>
<thead>
<tr>
<th>Task</th>
<th>Answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Compile a C program</td>
<td><strong>E. gcc</strong></td>
</tr>
<tr>
<td>2. Copy a file to a different computer</td>
<td><strong>M. scp</strong></td>
</tr>
<tr>
<td>3. Edit a text file</td>
<td><strong>D. emacs</strong></td>
</tr>
<tr>
<td>4. View the contents of a file (not to edit it)</td>
<td><strong>G. less</strong></td>
</tr>
<tr>
<td>5. Show what files are in the current directory</td>
<td><strong>H. ls</strong></td>
</tr>
</tbody>
</table>
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) {
    if (head == NULL) {
        return 0;
    }
    if (head->next == NULL) {
        return head->data;
    }
    int temp = findMax(head->next);
    if (head->data > temp) {
        return head->data;
    }
    return temp;
}
```
Question 5: C Tracing [9 pts]
What is the output when the following C program is run?

```
#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;
}
```

Answer:
1: a = 55, b=2
2: a = 55, b=55
3: a = 55, b=42
Question 6: ISA Concepts [6 pts]

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

**Answer:**
The ISA is the contract between the software and the hardware. It specifies how instructions are encoded, and what functional behavior they have.

2. What is an advantage of a 3-operand ISA over a 2-operand ISA? (Hint: you might give an example assembly sequence)

**Answer:**
3-operand ISAs allow the compiler more flexibility in where the result is written: it does not have to implicitly be one of the source registers. This is especially useful when the source registers’s values are needed for future computations. Consider the following 3-operand assembly sequence:

- `add r1 = r2 + r3`
- `sub r4 = r2 - r3`

A 2-operand ISA would have required a register copy to preserve the value of either r2 or r3 for the second instruction, as one of them would need to be the destination of the first instruction.
Question 7: Asm Tracing [5 pts]
Trace (execute by hand) the following fragment of assembly. The starting register and memory state is given to you in the table below. Write the final register and memory state in the blank row of the table.

\[
\begin{align*}
\text{lw } & \text{ $t0, 0($sp$)} \\
\text{li } & \text{ $t1, 3} \\
\text{add } & \text{ $t0, $t0, $t1} \\
\text{sw } & \text{ $t0, 4($sp$)} \\
\text{lw } & \text{ $v0, -8($fp$)}
\end{align*}
\]

<table>
<thead>
<tr>
<th></th>
<th>Registers</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>t0</td>
<td>t1</td>
</tr>
<tr>
<td>Initial</td>
<td>0x00</td>
<td>0x00</td>
</tr>
<tr>
<td>Final</td>
<td>0x14</td>
<td>0x3</td>
</tr>
</tbody>
</table>
Question 8: Asm Programming [20 pts]
Write MIPS assembly for the following C function.

```c
int countMatching(int * ptr, int n) {
    int count = 0;
    while (n > 0) {
        int x = testFn(*ptr);
        if (x != 0) {
            count++;
        }
        ptr++;
        n--;
    }
    return count;
}
```

Answer on next page
.globl countMatching
.ent countMatching
.text

countMatching:

addiu $sp, $sp, -32 # make space for stack frame
sw $fp, 0($sp) # save regs to stack
sw $ra, 4($sp)
sw $s0, 8($sp)
sw $s1, 12($sp)
sw $s2, 16($sp)
addiu $fp, $sp, 28 # setup new fp
move $s0, $a0 # move ptr into s0
move $s1, $a1 # move n into s1
li $s2, 0 # count in s2
.L_cm_while:
    blez $s1, .L_cm_end_while # while (n > 0) {
    lw $a0, 0($s0) # put *ptr in $a0
    jal testFn # call testFn
    # x returned in v0
    beqz $v0, .L_cm_endif # if (x != 0) {
    addi $s2, $s2, 1 # count++
    .L_cm_endif: # } of if
    addiu $s0, $s0, 4 # ptr++ (add 4 b/c int *)
    addi $s1, $s1, -1 # n--
    b .L_cm_while # } of while loop
.L_cm_endwhile:
move $v0, $s2 # put return result in v0
lw $s2, 16($sp) # restore regs from stack
lw $s1, 12($sp)
lw $s0, 8($sp)
lw $ra, 4($sp)
lw $fp, 0($sp)
addiu $sp, $sp, 32 # put sp back (release space)
jr $ra # return to caller

.end countMatching