ECE/CS 250 Midterm Exam #1, Spring 2017

Name: _____

Duke students are bound by an academic integrity standard:

1. I will not lie, cheat, or steal in my academic endeavors, nor will I accept the actions of those who do.

2. I will conduct myself responsibly and honorably in all my activities as a Duke student.

Please sign your name below to acknowledge that you follow this standard:

PLEASE CAREFULLY READ THE QUESTIONS.

MAKE SURE YOU ARE ANSWERING THE QUESTIONS AS WRITTEN.

NOTE: This question says "6-bit 2s complement" but should've said "7-bit 2s complement" for parts (a), (b), and (c).

1) [2 points] (a) Represent the base-10 number 17 in 6-bit 2s complement representation. Put your final answer in the box. (No need to show work.)

0010001

(b) [3 points] Represent the base-10 number -55 in 6-bit 2s complement representation. Show your work and then put your final answer in the box.

1001001

(c) [5 points] Add these two numbers together using 2s complement arithmetic. Do NOT invert the negative number and then add two positive numbers together. You must just directly add the two numbers. Show your work and put the final result in the box.

1011010

2) (a) [2 points] Write 187_{10} in hexadecimal (base 16). Show your work and put your final result in the box.

0xBB

(b) [8 points] The IEEE 754 floating point standard specifies that 32-bit floating point numbers have one sign bit, an 8-bit exponent (with a bias of 127), and a 23-bit significand (with an implicit "1"). What floating point number is represented by the following 32 bits? Show your work and put your final result in the box.

0 01000001 01100000000000000000000

msb = 0 --> positive

exponent = 65 unsigned --> exponent is 65 - 127 = -62

mantissa is 1.011000 0

This ended up being kind of ugly because of the exponent. If you wrote 1.011 x 2^-62, you got credit.

3) [5 points] (a) You bought software that runs on your laptop that has an Intel processor. Why does that software not run on a smartphone with an ARM processor?

They have different ISAs. A computer can only execute its ISA. (E.g., all AMD processors execute the x86 ISA.)

(b) [5 points] Which of the following issues are part of the instruction set architecture? Circle the issues that are part of the ISA.

- **The number of registers** YES
- **The number of bits in each register** YES
- **A detailed specification of the hardware that performs addition** NO
- **The memory addressing modes** YES
- **The number of memory addresses** YES

(c) [5 points] Are the following statements true or false? Circle one for each.

All ISAs are pretty much like MIPS. **TRUE** **FALSE** F

A MIPS jump (j) instruction has a register operand. **TRUE** **FALSE** F

In C, all large variables should be malloc'ed. **TRUE** **FALSE** F

I can copy a string by copying the pointer to its 1st char. **TRUE** **FALSE** F

A carry-out in 2s complement addition always indicates an overflow. **TRUE** **FALSE** F

4) (a) [5 points] Write <u>one line</u> of C code to dynamically allocate space for 16 pointers to characters on the heap.

```
char** dan_ptr = (char**)malloc(16*sizeof(char*));
```

(b) [10 points] At the end of the following snippet code, what are the values of A, B, C, D, and E? Put your results in the table to the right. If any line that writes to A, B, C, D, or E causes a seg fault, write "seg fault for that value" and continue with the program (even though a seg fault would normally end a program). Assume a 32-bit machine with 32-bit ints.

```
int numbers[100]; // assume numbers[0] is at address 1000
for (i=0; i<50; i++){
        numbers[i] = i;
}
for (i=50; i<100; i++){
        numbers[i] = (int) & numbers[i];
}
int A = *(numbers+2);
int B = numbers[52];
int* x = (numbers + 10); // assume x is at address 2000
int** y = &x;   // assume y is at address 2004
int* C = *y;    // assume C is at address 2008
int D = *C;   // assume D is at address 2012
int* E = numbers + 100;   // assume E is at address 2016
```

A	2
B	1208
C	1040
D	10
E	1400

5) [20] Convert the following C code for the function f() into MIPS code. Use appropriate MIPS conventions for procedure calls, including the passing of arguments and return values, as well as the saving/restoring of registers. Assume that there are 2 argument registers ($a0-$a1), 2 return value registers ($v0-$v1), 2 general-purpose callee saved registers ($s0-$s1), and 2 general-purpose caller-saved registers ($t0-$t1). Assume $ra is callee-saved. The C code is obviously somewhat silly and unoptimized, but YOU MAY NOT OPTIMIZE IT -- you must translate it as is.

```
1: int f (int num){
2:     // set up stack frame
3:     int x = 0; // x must be in $t0
4:     int y = 1; // y must be in $s0
5:     if (num == 0) {
6:         y = y + x;
7:     }
8:   x = bar(x,y);
9:   y = x + y;
10:   return (y + 2);
11:   // clean up stack frame
12: }

 int bar (int arg) {
     // don't worry about bar(),
     // but bar could call function
 }
```

C lines	Lines of Assembly

1-2	subi $sp, $sp, 12
	sw $ra, 0($sp)
	sw $s0, 4($sp)

3-4	li $t0, 0 li $s0, 1
5-7	bneqz $a0, next add $s0, $s0, $t0 next:

8	sw $t0, 8($sp)
	move $a0, $t0
	move $a1, $s0
	jal bar
	lw $t0, 8($sp)

9	add $s0, $v0, $s0
10	addi $v0, $s0, 2

11-12	lw $s0, 4($sp)
	lw $ra, 0($sp)
	addi $sp, $sp, 12
	jr $ra