

# CSE351 Spring 2010 – Midterm Exam (3 May 2010)

---

Please read through the entire examination first! We designed this exam so that it can be completed in 50 minutes and, hopefully, this estimate will prove to be reasonable.

There are 3 problems for a total of 100 points. The point value of each problem is indicated in the table below. Write your answer neatly in the spaces provided. If you need more space (you shouldn't), you can write on the back of the sheet where the question is posed, but please make sure that you indicate clearly the problem to which the comments apply. Do NOT use any other paper to hand in your answers. If you have difficulty with part of a problem, move on to the next one. They are mostly independent of each other.

For problem 3, you may find it convenient to separate the page of assembly code from the rest of the exam (you do not need to turn in that page as no answers are to be written on that page) so that you can refer to it more easily for subsequent questions.

The exam is CLOSED book and CLOSED notes. Please do not ask or provide anything to anyone else in the class during the exam. Make sure to ask clarification questions early so that both you and the others may benefit as much as possible from the answers.

---

**Name:** \_\_\_\_\_

**ID#:** \_\_\_\_\_

<b>Problem</b>	<b>Max Score</b>	<b>Score</b>
1	35	
2	20	
3	45	
<b>TOTAL</b>	<b>100</b>	

## 1. Number Representation (35 points)

While on a boat, two circuit elements within the boat's 8-bit navigation computer break. Miraculously, the computer still seems to work, except that when it reads or writes any value, the two least significant bits are always 0.

For instance, this computer would erroneously compute:

$$\begin{array}{r} 0000\ 0001 \\ +\ 0000\ 0011 \\ \hline 0000\ 0000 \end{array}$$

Notice that the computer also failed to carry from the two LSBs, since it read them as 0.

a) (5 pts) What two's complement integers will this computer read correctly?

b) (10 pts) The boat uses a floating point system with 1 sign bit, 3 bits for the exponent, and 4 bits for the significand. What positive denormalized values will it read correctly? Recall that denormalized values will have an exponent of 000 and that the bias for a 3 bit exponent is  $2^{3-1}-1 = 3$ .

c) (10 pts) What is now the smallest normalized value you can represent with this broken system?

d) (10 pts) While on your boat, you discover that one of your guest lives at the south pole. Fortunately, the navigation computer represents latitude with double precision, with 1 bit for the sign, 7 bits for the exponent, and 8 bits for the significand. This means that the double precision value will occupy two bytes (and each of the two bytes will have 0s in the two LSBs – still broken). What is now the closest latitude to the south pole that the computer will represent? The south pole has a latitude of -90. Recall that with a 7 bit exponent the exponent bias will be  $2^{7-1}-1 = 63$ .

- What positive exponents can you represent correctly? \_\_\_\_\_
- What is the value of the sign bit? \_\_\_\_\_
- What is the biggest significand you can represent correctly? \_\_\_\_\_
- What is the number closest to -90 that you can represent? \_\_\_\_\_
- What will be the two byte encoding of this number?  
    Sign (1 bit) and Exponent (7 bits) \_\_\_\_\_  
    Significand (8 bits) \_\_\_\_\_

## 2. Stack Frames (20 points)

Given the following disassembly of a short function:

```
0x00001fb6 <func+0>:    push    %ebp
0x00001fb7 <func+1>:    movl   %esp,%ebp
0x00001fb9 <func+3>:    subl  $0x8,%esp
0x00001fbc <func+6>:    movl   0x8(%ebp),%eax
0x00001fbf <func+9>:    addl   %eax,%eax
0x00001fc1 <func+11>:   leave
0x00001fc2 <func+12>:   ret
```

and the following stack, %ebp, and %esp \*right after\* executing the instruction, <func+3>, at address 0x00001fb9:

```
%esp = 0x300
%ebp = 0x308
```

```
0x300: 0x00000000
0x304: 0x00000000
0x308: 0x00000328
0x30c: 0x00001fdc
0x310: 0x00000003
```

a) (8 pts) What is the value in %eax (the return value of the function) after we return and leave the function (after instruction <func+12>)?

b) (6 pts) What was the old value of the stack pointer (%esp) before we executed the first instruction of this function (at <func+0>)?

c) (6 pts) What is the old value of the base pointer (`%ebp`) before we entered this function?

### 3. Analyzing and Extending Assembly Code (45 points)

For this problem, you'll examine the assembly code for the procedure "walk", an interpreter of instructions for a simple robot. The robot keeps track of its current (x, y) position on the Cartesian plane, and its current direction:

- north (encoded as 0; facing towards higher y coordinates),
- east (encoded as 1; facing towards higher x coordinates),
- south (encoded as 2; facing towards lower y coordinates), or
- west (encoded as 3; facing towards lower x coordinates).

The "walk" function (shown on the next page) takes as arguments:

- an initial x coordinate,
- an initial y coordinate,
- an initial direction, and
- an array of instructions (encoded as unsigned integers, ending in a 0)

that tell it how to move.

The procedure reads and decodes each instruction in turn, branching to the relevant code for each instruction and updating the robot's position and direction accordingly. The instruction set is very limited:

- HALT (encoded as 0) tells the robot to stop reading instructions and print its ending position.
- RIGHT (encoded as 1) tells the robot to stay in place and turn 90 degrees to the right. For example, if the robot reads a RIGHT instruction when it is at (2, 5), headed north, it will remain at position (2, 5) and turn to face east.
- STEP (encoded as 2) tells the robot to take one step in its current direction. For example, if the robot reads a STEP instruction when it is at (2, 4), headed north, it will move to (2, 5), still facing north.

Starting at (0, 0) headed north and given the 8 encoded instructions: 2, 1, 2, 2, 1, 2, 2, 0; the robot halts at (2, -1), headed south. Use this input/output and the assembly code for walk on the next page to answer the questions on the page after that.

```
void walk(int x, int y, unsigned dir, unsigned instructions[])
```

```
walk:                                .L16:
    pushl %ebp                        movl 16(%ebp), %eax
    movl %esp, %ebp                  addl $1, %eax
    subl $40, %esp                   andl $3, %eax
    movl $0, -12(%ebp)               movl %eax, 16(%ebp)
    jmp .L15                          addl $1, -12(%ebp)
.L25:                                jmp .L15
    movl -12(%ebp), %eax              .L17:
    movl 20(%ebp,%eax,4), %eax        movl 16(%ebp), %eax
    cmpl $1, %eax                    cmpl $0, %eax
    je .L16                           je .L20
    cmpl $2, %eax                     cmpl $1, %eax
    je .L17                           je .L21
                                        cmpl $2, %eax
                                        je .L22
                                        cmpl $3, %eax
                                        je .L23
                                        jmp .L19
.L15:                                .L20:
    movl -12(%ebp), %eax              addl $1, 12(%ebp)
    movl 20(%ebp,%eax,4), %eax        jmp .L19
    testl %eax, %eax                  .L21:
    jne .L25                          addl $1, 8(%ebp)
    [Print current position]          jmp .L19
    leave                               .L22:
    ret                                subl $1, 12(%ebp)
                                        jmp .L19
                                        .L23:
                                        subl $1, 8(%ebp)
.L19:                                .L19:
    addl $1, -12(%ebp)                addl $1, -12(%ebp)
    jmp .L15                          jmp .L15
```

```
[BBB: ]
```

### 3. Analyzing and Extending Assembly Code (continued) (45 points)

a) (8 pts) For each parameter to walk, what is the corresponding memory address used in the assembly code (in terms of `%ebp`)?

x: \_\_\_\_\_

y: \_\_\_\_\_

dir: \_\_\_\_\_

instructions: \_\_\_\_\_

b) (20 pts) Describe what each of these sections of code does, in 1-2 sentences each:

- `.L25` through `“je .L17”`

- `.L15` through `“jne .L25”`



- .L16 to .L17

- .L17 through “jmp .L15”

c) (17 pts) Implement code for a new IFNORTH instruction for the robot (encoded as 3). When the robot reads IFNORTH at index  $i$  in the instruction array:

- if the robot is facing north, it replaces its current instruction index with the integer stored at index  $i + 1$  in the instruction array and continues processing instructions starting at this new index; else
- if the robot is not facing north, it continues processing instructions starting at index  $i + 2$  in the instruction array.

For example, the instruction array IFNORTH(3), 3, STEP(2), HALT(0) should cause the robot to halt in its current position if it is facing north (the next instruction executed is the HALT at  $i = 3$ ) or, if it is not facing north, take one step forward (the next instruction executed is the STEP at  $i = 2$ ) and then halt.

We'll insert the following instructions at the location marked AAA in the assembly code:

```
    cml  $3, %eax
    je   .L30
```

Write the assembly code to implement the IFNORTH instruction (show the instructions to be inserted in the assembly code at the location marked BBB in the space provided below). Of course, they will start at the label .L30.

```
.L30:
```

## REFERENCES:

### Powers of 2:

$2^0 = 1$   
 $2^1 = 2$   
 $2^2 = 4$   
 $2^3 = 8$   
 $2^4 = 16$   
 $2^5 = 32$   
 $2^6 = 64$   
 $2^7 = 128$   
 $2^8 = 256$   
 $2^9 = 512$   
 $2^{10} = 1024$

### Assembly Code Instructions:

pushl	push a value onto the stack
leave	restore ebp from the stack
ret	pop return address from stack and jump there
movl	move 4 bytes between immediate values, registers and memory
addl	add first operand to second put result in second
subl	subtract first operand from second put result in second
andl	logical and of first operand with second and result stored in second
jmp	jump to address
je	conditional jump to address if zero flag set
jne	conditional jump to address if zero flag is not set
cmpl	subtract first operand from second and set flags
testl	logical and of first and second operands to set flags