CSE410 Autumn 2013 – Midterm Exam (Nov. 8, 2013)

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

There are 5 problems worth a total of 100 points. 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 independent of each other.

The last pages of the test contain reference information. Feel free to separate those pages from the exam if it is convenient.

The exam is CLOSED book and CLOSED notes, no calculators, electronic devices, telepathy, or other communications. If you have questions, please raise your hand and someone will come to you.

Name: _____ UW ID # _____

Problem	Max Score	Score
1	16	
2	20	
3	14	
4	25	
5	25	
TOTAL	**100**	

1. Bits (16 points) The following two questions are similar to the questions in Lab 1 and the same ground rules apply:

- Assume all values are 32-bit 2's complement integers.
- You may only use the operators !, ~, &, ^, |, +, <<, and >> and integer constants from 0 through 255 (0xFF). No other operators, control constructs, data structures, etc., and you may not call other functions.
- You may only use function arguments and any additional `int` local variables you declare. No global variables or other external data, no data structures like arrays.

For this exam problem, you may assume that no arithmetic overflow will occur if you add two ints.

(a) (6 points)

```
/* Return the 2's complement of x. (i.e., compute */
/* -x without using "-".)                         */
int negate(int x) {




     return ____________________________;

}
```

(b) (10 points)

```
/* Return 1 if x < y, else return 0. */
int isLess(int x, int y) {




     return ____________________________;

}
```

2. Number representation – Integers (20 points) Suppose we are working on a machine with 6-bit, 2-s complement integers.

(a) What are the largest possible and smallest possible 2's complement numbers that can be expressed in 6 bits? i.e., $TMin_6$ and $TMax_6$?

(b) What is the decimal value of 110100_2 if we interpret it as a 6-bit 2's complement signed number?

(c) What is the decimal value of 110100_2 if we interpret it as a 6-bit ***unsigned*** number?

(d) Add 110100 and 010011 as 2's complement integers and convert the result to decimal.

```
  110100
+ 010011
--------
```

3. Number representation – Floating Point (14 points) A floating-point number has the form $(-1)^S$ x M x 2^E. M is the mantissa and 2^E is the exponent. Also recall that M ranges from 1.0 to (almost) 2.0.

If we convert the decimal number 9.625 to floating point, what is the binary value of M and the decimal value of E? You do not need to encode this in IEEE 754 representation with a biased exponent. Just give the values of M and E. (Hint: see the tables on the last few pages for powers of 2.)

What is the binary value of frac (the fractional bits of the mantissa that are actually stored in the IEEE floating point representation)?

4. x86 and C code. (25 points)

We have a small x86 assembly language function that we need to disassemble.

```
mystery:
     pushl   %ebp
     movl    %esp,%ebp
     movl    16(%ebp),%eax
     cmpl    12(%ebp),%eax
     jge     .L3
     testl   %eax,%eax
     js      .L3             # js means jump if sign bit set
     movl    8(%ebp),%edx
     movl    (%edx,%eax,4), %eax
     leave
     ret
.L3:
     movl    $0, %eax
     leave
     ret
```

Your job is to write an equivalent C function on the next page. The name of the function is `mystery` (the label at the start of the assembly language code).

The original C version of the function did not use any explicit pointer operators (no `*` or `&`) so for full credit your solution should not use them either (although we'll award generous partial credit if you can only convert the program to C with a few pointer operations). Think about other possible data structures that might be involved.

You can detach this page if it is convenient – it does not need to be turned in.

Hint: Remember that the result of an integer-valued function is returned in register eax.

Hint: In the x86 calling convention for a function `mystery(a,b,c)`, the leftmost argument `a` is stored in the stack at `8(%ebp)`, `b` is at `12(%ebp)`, `c` is at `16(%ebp)`, and so on.

Hint: remember there is reference information on the last two pages of the exam.

It would be worth taking a minute to figure out where variables are stored in memory or registers.

4. x86 Code and C (cont.)

Write your C version of the assembly language function from the previous page here. Your answer should only need a few lines of code.

5. x86-64 Programming. (25 points)

Write an x86-64 assembly language function that is equivalent to the following C code

```
int loop(int x, int y) {
  while (x < y)
    x = x + y;
  return x;
}
```

Remember to use the 64-bit register and function call conventions (summarized on the last pages). The solution does not require too many assembly language instructions – maybe a dozen or so at the most. Please include brief comments in your code to help us understand how you are using the registers and what assembly code corresponds to the different parts of the C code.

REFERENCES

Powers of 2:

$2^0 = 1$	
$2^1 = 2$	$2^{-1} = .5$
$2^2 = 4$	$2^{-2} = .25$
$2^3 = 8$	$2^{-3} = .125$
$2^4 = 16$	$2^{-4} = .0625$
$2^5 = 32$	$2^{-5} = .03125$
$2^6 = 64$	$2^{-6} = .015625$
$2^7 = 128$	$2^{-7} = .0078125$
$2^8 = 256$	$2^{-8} = .00390625$
$2^9 = 512$	$2^{-9} = .001953125$
$2^{10} = 1024$	$2^{-10} = .0009765625$

Assembly Code Instructions:

`push`	push a value onto the stack and decrement the stack pointer
`pop`	pop a value from the stack and increment the stack pointer
`call`	jump to a procedure after first pushing a return address onto the stack
`ret`	pop return address from stack and jump there
`mov`	move a value between registers and memory
`lea`	compute effective address and store in a register
`add`	add 1st operand to 2nd with result stored in 2nd
`sub`	subtract 1st operand from 2nd with result stored in 2nd
`and`	bit-wise AND of two operands with result stored in 2nd
`or`	bit-wise OR of two operands with result stored in 2nd
`sar`	shift data in the 2nd operand to the right (arithmetic shift) by the number of bits specified in the 1st operand
`jmp`	jump to address
`jne`	conditional jump to address if zero flag is not set
`js`	conditional jump to address if sign flag is set (i.e., negative)
`cmp`	subtract 1st operand from 2nd operand and set flags
`test`	bit-wise AND 1st operand from 2nd operand and set flags

Register map for x86-64:

Note: all registers are caller-saved except those explicitly marked as callee-saved, namely, `rbx`, `rbp`, `r12`, `r13`, `r14`, and `r15`. `rsp` is a special register.