Solution to CS 3843 Midterm Exam Two Fall 2013

Name (Last)_______________, (First)_________________

You may use a calculator and one sheet of notes on this exam, but no other materials and no computer.

Show all the major steps in your work to receive partial credits.

Problem 1 (27 points)

Assume the following values are stored at the indicated memory addresses and registers

Address	**Value**	**Register**	**Value**
0x200	0x3A	%eax	0x200
0x204	0x45	%ecx	0x08
0x208	0x3D	%edx	0x03
0x20C	0x11		
0x210	0x2F		
0x214	0x09		

a) (12 points - 1.5 points each) Fill the following table :

Operand	**Value**	**Operand**	**Value**
12(%eax)	0x11	4(%eax, %edx, 4)	0x2F
4(%eax, %ecx)	0x11	0x1F0(, %edx, 8)	0x3D
-3(%eax, %edx)	0x3A	0x200(%ecx, %edx, 4)	0x09
leal -0x10(%eax, %ecx, 8), %edx	0x230	leal 0xFC(%ecx), %edx	0x104

b) (15 points - 1.5 point each) Fill in the following table

Instruction	**Destination**	**Value**
xorl %edx, %ecx	%ecx	0x0B
subl %ecx, %edx	%edx	0xFB or 0xFFFFFFFB
addl %edx, 4(%eax, %ecx, 2)	0x214	0x0C
imul $4, (%eax, %edx, 4)	0x20C	0x44
incl 0x8(%eax)	0x208	0x3E

Problem 2 (35 points)

Fill in the following table. Assume that *x* and *y* are of a new type `short` integer which is 12 bits. Enter the value of (*y*-*x*) in decimal. This is the value that would be stored in *z* if `short z = y - x;`

The range of 12-bit signed number: -2048 ~ 2047
The range of 12-bit unsigned number: 0 ~ 2^{12}-1=4095

Sol: 2’s complement representation:

-7=N*=$2^{12}-N=2^{12}-7=4089$

0x7fe =$2^{11}-2=2046$

-0x7fe=$2^{12}-0x$7fe=4096-2046=2050

Consider the instruction: `cmpw %eax, %ecx`

(2 points each) Fill in the value of the flags if `%eax` contains *x* and `%ecx` contains *y*.

x	*y*	*z* = *y* - *x*	ZF	SF	OF	CF
15	7	7-15=-8(signed)	0	1	0	1
-7	15	15-(-7)=22 (signed) 15-4089=-4074 (unsigned)	0	0	0	1
-7	-7	(-7)-(-7)=0	1	0	0	0
7	0x7fe	2046-7=2039	0	0	0	0
7	-0x7fe	-2046-7=-2053(signed) 2050-7=2043(unsigned)	0	0	1	0
0x7fe	7	7-2046=-2039	0	1	0	1
0x7fe	-7	-7-2046=-2053(signed) 4089-2046=2043 (unsigned)	0	0	1	0

Problem 3 (18 points)

Please check whether the following instruction is TRUE or FALSE, and if FALSE, and what’s wrong with each line?

1) movl %edx, 0xFD(%eax) TURE (X) FALSE ( )

   If FALSE, explain why?

2) movl (%ecx), 0xC(%esp) TURE ( ) FALSE (X)

   If FALSE, explain why?

Ans: Cannot have both source and destination be memory address.

3) movb $0xFF, (%ah) TURE ( ) FALSE (X)

If FALSE, explain why?

Ans: Cannot use %al as address register

4) movw %ecx, (%edx) TURE ( ) FALSE (X)

If FALSE, explain why?

Ans: Mismatch between instruction suffix and register ID.

5) movb %cl, %dx TURE ( ) FALSE (X)

If FALSE, explain why?

Ans: Destination operand incorrect size.

6) movw %ax, $0xCD TURE ( ) FALSE (X)

If FALSE, explain why?

Ans: Cannot have immediate as destination

Problem 4 (20 points)

A function *fun* has the following overall structure:

```
int fun(unsigned x) {
    int val = 0;
    int i;
    while (x) {
          val ^= x;
          x >>= 1;
     }
  return   val & 0x1;
}
```

The GCC C compiler generates the following assembly code:

```
    x at %ebp+8                              //  comments here
```

```
        movl     8(%ebp),   %edx                        //  %edx = x
        movl     $0,   %eax                             //    %eax = val = 0
        testl    %edx, %edx                             //    test x = 0 or not
        je       .L7                                    //     jump to .L7 if x = 0
    .L10
        xorl     %edx, %eax                             //    val ^= x
        shrl     %edx           shift right by 1        // x >> = 1
        jne      .L10                                   //  if x != 0,  jump to .L10
    .L7:
        andl     $1, %eax                               //   return   val & 0x1
```

Reverse engineer the operation of this code and then do the following:

A. (8 points) Comment each assembly instruction.

Sol: See above.

B. (6 points) Use the assembly-code version to fill in the missing parts of the C code.

Sol: See above.

B. (6 points) Describe in English what this function computes.

Sol: This code computes the parity of argument *x*. That is, it returns 1 if there is an odd number of ones in *x* and 0 if there is an even number.