Computer architecture
Handout assignment
Due date : 2014.10.15.

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

Address	Value
0x100	0xFF
0x104	0xAB
0x108	0x13
0x10c	0x11

Address	Value
%eax	0x100
%ecx	0x1
%edx	0x3

Fill in the following table showing the values for the indicated operands.

Operand	Value
%eax	
0x104	
$0x108	
(%eax)	
4(%eax)	
9(%eax, %edx)	
260(%ecx, %edx)	
0xFC(,%ecx,4)	
(%eax, %edx, 4)	

2. You are given the following information. A function with prototype

```
int decode2(int x, int y, int z);
```

is compiled into assembly code. The body of the code is as follows:

```
1 movl 16(%ebp),%eax
2 movl 12(%ebp),%edx
3 subl %eax,%edx
4 movl %edx,%eax
5 imull 8(%ebp),%edx
6 sall $31,%eax
7 sarl $31,%eax
8 xorl %edx,%eax
```

Parameters x, y, and z are stored at memory locations with offsets 8, 12, and 16 relative to the address in register %ebp. The code stores the return value in register %eax.

Write C code for `decode2` that will have an effect equivalent to our assembly code. You can test your solution by compiling your code with the -S switch. Your compiler may not generate identical code, but it should be functionally equivalent.

3. Suppose register %eax holds value x and %ecx holds value y. Fill in the table below with formulas indicating the value that will be stored in register %edx for each of the following assembly code instructions.

Expression	Results
leal 6(%eax), %edx	
leal (%eax, %ecx), %edx	
leal (%eax, %ecx, 4), %edx	
leal 7(%eax, %eax, 8), %edx	
leal 0xA(,$ecx,4), %edx	
leal 9(%eax,%ecx,2), %edx	

4. Suppose we want to generate assembly code for the following C function:

```
int shift_left2_rightn(int x, int n)
{
        x <<= 2;
        x >>= n;
        return x;
}
```

The following is a portion of the assembly code that performs the actual shifts and leaves the final value in register %eax. Two key instructions have been omitted. Parameters x and n are stored at memory locations with offsets 8 and 12, respectively, relative to the address in register %ebp.

```
1 movl 12(%ebp),%ecx        Get x
2 movl 8(%ebp),%eax         Get n
3 _____________             x <<= 2
4 _____________             x >>= n
```

Fill in the missing instructions, following the annotations on the right. The right shift should be performed arithmetically.

5. The following assembly code:

Initially x, y, and n are offsets 8, 12, and 16 from %ebp

```
1       movl 8(%ebp),%ebx
2       movl 16(%ebp),%edx
3       xorl %eax,%eax
4       decl %edx
5       js .L4
6       movl %ebx,%ecx
7       imull 12(%ebp),%ecx
8       .p2align 4,,7       Inserted to optimize cache performance
9   .L6:
10      addl %ecx,%eax
11      subl %ebx,%edx
12      jns .L6
13  .L4:
```

was generated by compiling C code that had the following overall form

```
1 int loop(int x, int y, int n)
2 {
3       int result = 0;
4       int i;
5       for (i = _____; i _____ ; i = _____) {
6               result += _____ ;
7       }
8       return result;
9 }
```

Your task is to fill in the missing parts of the C code to get a program equivalent to the generated assembly code. Recall that the result of the function is returned in register %eax. To solve this problem, you may need to do a little bit of guessing about register usage and then see whether that guess makes sense.

A. Which registers hold program values result and i?
B. What is the initial value of i?
C. What is the test condition on i?
D. How does i get updated?
E. The C expression describing how to increment result in the loop body does not change value from one iteration of the loop to the next. The compiler detected this and moved its computation to before the loop. What is the expression?
F. Fill in all the missing parts of the C code.

6. Starting with C code of the form

```
int branch(int x, int y) {
        int val;
        if ( _____________ ) {
                val = _____________ ;
        } else if ( _____________ ) {
                val = _____________ ;
        } else {
                val = _____________ ;
        }
        return val;
}
```

GCC generates the following assembly code:

x at %ebp+0x8, y at %ebp+0xc

```
    movl 8(%ebp), %edx
    movl 12(%ebp), %ecx
    cmpl $-43981, %edx
    jl .L7
    movl %edx, %eax
    subl %ecx, %eax
    addl %edx, %ecx
    cmpl $3, %edx
    cmovl %ecx, %eax
    popl %ebp
    ret
.L7:
    movl %ecx, %eax
    sarl $2, %eax
    imull %edx, %eax
    popl %ebp
    ret
```

Fill in the missing expressions in the C code.

7. A function *iffun* has the following overall structure and the GCC C compiler has generated the following assembly code for it:

```
int iffun(int x, int y, int z)
{
     int val = 0;

     if ( _____________ ) {

          val = _____________ ;

     } else if( _____________ ) {

          val = _____________ ;

     } else if( _____________ ) {

          val = _____________ ;

     } else {

          val = _____________ ;

     }

     return val;
}
```

```
x at %ebp+8, y at %ebp+12, z at %ebp+16

fun:
     pushl %ebp
     movl %esp, %ebp
     movl 8(%ebp), %edx
     movl 12(%ebp), %ecx
     movl 16(%ebp), %ebx
     testl %edx, %edx
     leal 12(%edx,%edx), %eax
     jle .L8
.L3:
     movl %ebp, %esp
     popl %ebp
     ret
.L8:
     leal 1(%ebx), %esi
     cmpl %ecx, %esi
     leal -16(%edx,%ecx), %eax
     jl .L3
     cmpl $-4, %ebx
     jge .L9
     leal (%ecx,%edx), %edx
     leal (%edx,%ebx), %eax
     movl %ebp, %esp
     popl %ebp
     ret
.L9:
     movl %edx, %eax
     sarl $31, %edx
     idivl %ecx
     jmp .L3
```

Use the assembly code version to fill in the missing parts of the C code.

8. The CPU maintains a set of single-bit condition code registers describing attributes of the most recent arithmetic of logical operation. These registers can then be tested to perform conditional branches. Fill in the omitted entries in the following table.

CF	Carry Flag	The most recent operation generated a carry out of the most significant bit. Used to detect overflow for unsigned operations.
ZF	Zero Flag	
SF	Sign Flag	
OF	Overflow Flag	

9. A function *forfun* has the following overall structure and the GCC C compiler has generated the following assembly code for it:

```
int forfun(unsigned x) {
        int val = 0;
        int i;
        for ( ____ ; ____ ; ____ ) {
                if( ____ ) {
                ____;
                }
        }
        return val;
}
```

```
x at %ebp+8

forfun:
    pushl %ebp
    xorl %eax, %eax
    movl %esp, %ebp
    movl $1, %edx
    pushl %ebx
    movl 8(%ebp), %ebx
    addl $1, %ebx
    cmpl $1, %ebx
    ja .L7
    jmp .L2
.L5:
    leal (%eax,%edx), %ecx
    testb $1, %dl
    cmove %ecx, %eax
.L7:
    addl $1, %edx
    cmpl %edx, %ebx
    ja .L5
.L2:
    popl %ebx
    popl %ebp
    ret
```

Reverse engineer the operation of this code and then do the following:
A. Use the assembly code version to fill in the missing parts of the C code.
B. Describe what this function computes.

10. For the C function *rfun* with the following general structure GCC generates the assembly code as shown below:

```
int rfun(unsigned x, unsigned y)
{
        if ( ____ ) return ____ ;
        if ( ____ ) return ____ ;
        return ____ ;
}
```

```
rfun:
    pushl %ebp
    movl %esp, %ebp
    subl $40, %esp
    movl %ebx, -12(%ebp)
    movl %esi, -8(%ebp)
    movl %edi, -4(%ebp)
    movl 8(%ebp), %esi
    movl 12(%ebp), %ebx
    movl $1, %eax
    cmpl %esi, %ebx
    je .L2
    movl %esi, %eax
    cmpl $1, %ebx
    je .L2
    subl $1, %esi
    leal -1(%ebx), %eax
    movl %eax, 4(%esp)
    movl %esi, (%esp)
    call rfun
    movl %eax, %edi
    movl %ebx, 4(%esp)
    movl %esi, (%esp)
    call rfun
    addl %edi, %eax
.L2:
    movl -12(%ebp), %ebx
    movl -8(%ebp), %esi
    movl -4(%ebp), %edi
    movl %ebp, %esp
    popl %ebp
    ret
```

A. Fill in the missing expressions in the C code shown above.
B. (Assume x≥ y) Describe what function this code computes.