**SYSTEMS PROGRAMMING AND COMPUTER ARCHITECTURE**  
**Assignment 2: Assembly**

Assigned on: 4th Oct 2012  
Due by: 18th Oct 2012

**NOTE**: Unless otherwise stated, the assembly code in this assignment is IA32 assembly.

## 1 Assembly basics

### 1.1 Addressing modes

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

<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x204</td>
<td>0xFF</td>
</tr>
<tr>
<td>0x208</td>
<td>0xCD</td>
</tr>
<tr>
<td>0x20C</td>
<td>0x21</td>
</tr>
<tr>
<td>0x210</td>
<td>0x11</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Register</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>%eax</td>
<td>0x2</td>
</tr>
<tr>
<td>%ecx</td>
<td>0x204</td>
</tr>
<tr>
<td>%edx</td>
<td>0x3</td>
</tr>
</tbody>
</table>

Fill in the following table showing the types (i.e., immediate, register, memory) and the values of the indicated operands:

<table>
<thead>
<tr>
<th>Operand</th>
<th>Type</th>
<th>Memory address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>%eax</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x210</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$0x210</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(%ecx)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4(%ecx)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5(%ecx, %edx)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>519(%edx, %eax)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x204(,%eax, 4)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(%ecx, %eax, 2)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### 1.2 Arithmetic operations

Use the values of the memory addresses and registers from Question 1. Handle the different instructions independently. The result of one of the instructions does not affect the others. Fill in
the following table showing the effects of the following instructions, both in terms of the register or memory location that will be updated and the resulting value:

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Destination</th>
<th>Computation and result</th>
</tr>
</thead>
<tbody>
<tr>
<td>addl %eax, (%ecx)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>subl %edx, 4(%ecx)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>imull (%ecx, %eax, 4), %eax</td>
<td></td>
<td></td>
</tr>
<tr>
<td>incl 8(%ecx)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>decl %eax</td>
<td></td>
<td></td>
</tr>
<tr>
<td>subl %edx, %ecx</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1.3 leal and movl

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

<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
<th>Address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x108</td>
<td>0xFF</td>
<td>0x100</td>
<td>0x4</td>
</tr>
<tr>
<td>0x10C</td>
<td>0xCD</td>
<td>0x110</td>
<td>0x21</td>
</tr>
</tbody>
</table>

What is the difference between the two instructions? What value ends up in %ecx? Write the formula!

`movl 8(%eax, %edx, 4), %ecx`

`leal 8(%eax, %edx, 4), %ecx`

1.4 Condition codes

Consider the instruction `addl %eax, %ebx`. As a side-effect, it sets the condition flags (OF, SF, ZF, CF) according to the result.

Assuming a 4-bit machine, convert the given decimal pairs (a, b) to their binary representation and perform the addition. Give both the arithmetical value and the interpreted value (2’s complement) of the result. List the condition flags that are set.

(-1, -1), (+4, TMin), (TMax, TMax), (TMax, -TMax),
(TMin, TMax), (TMin, TMin), (-1, TMax), (2, 3).
1.5 Conditional branches

What is the value of %eax, when the last label (respectively .L3 and .L17) is reached? First, annotate the assembly code and then, write the corresponding C-statements!

i) Assume %eax := a, %edx := d.

```assembly
...  
cmpl %eax, %edx          ...  
jle .L2                testl %ecx, %ecx  
sUBL %eax, %edx          jle .L17  
movl %edx, %eax          xorl %edx, %edx  
jmp .L3                .L18:  
.L2:  
sUBL %edx, %eax              incl %edx  
.L3:                          addl %eax, %eax  
...                          cmpl %edx, %ecx  
                          jne .L18  
                          .L17:  
                          ...
```

ii) Assume %eax := 1, %ecx := N.

```assembly
...  
...  
testl %ecx, %ecx          ...  
jle .L17                jle .L17  
xorl %edx, %edx          xorl %edx, %edx  
jmpl %edx, %ecx          jmp .L3  
jne .L18                .L18:  
.L17:  
...                          ...
```

2 More assembly

2.1 Assembly Code Fragments

Consider the following pairs of C functions and assembly code. Fill in the missing instructions in the assembly code fragments (one instruction per blank). Your answers should be correct IA32 assembly code.

a) int f1(int a, int b) {
    f1: pushl %ebp
    movl %esp, %ebp
    return a - b;
    ______________
    movl 8(%ebp), %eax
    ______________
    movl %ebp, %esp
    popl %ebp
    ret
}

b) int f2(int a) {
    f2: pushl %ebp
    movl %esp, %ebp
    movl 8(%ebp), %eax
    leal ______________
    return a*5;
    movl %ebp, %esp
    popl %ebp
    ret
}

c) int f3(int a) {
    f3: pushl %ebp
    movl %esp, %ebp
    movl 8(%ebp), %edx
    ______________
    if (a <= 0)
    movl %edx, %eax
    jle .L11
    else
    .L8: movl %ebp, %esp
    ret
    return a;
    ______________
    .L11: negl %eax
    jmp .L8
}
2.2 Switch Statement

Consider the following C function and assembly code fragments. Which of the assembly code fragments matches the C function shown?

```c
int woohoo(int a, int r)
{
    int ret = 0;
    switch(a) {
    case 11:
        ret = 4;
        break;
    case 22:
    case 55:
        ret = 7;
        break;
    case 33:
    case 44:
        ret = 11;
        break;
    default:
        ret = 1;
    }
    return ret;
}
```

Fragment 1

```assembly
woohoo: pushl %ebp
    movl %esp, %ebp
    movl 8(%ebp), %edx
    movl $0, %ecx
    cmpl $11, %edx
    jne .L2
    movl $4, %ecx
    jmp .L3
.L2:
    cmpl $22, %edx
    jne .L3
    movl $7, %ecx
.L3:
    cmpl $55, %edx
    jne .L5
    movl $7, %ecx
.L5:
    cmpl $33, %edx
    sete %al
    cmpl $44, %edx
    sete %dl
    orl %edx, %eax
    testb $1, %al
    je .L6
    movl $11, %ecx
    jmp .L9
.L6:
    movl %ecx, %eax
    popl %ebp
    ret
```

Fragment 2

```assembly
woohoo: pushl %ebp
    movl $1, %eax
    movl %esp, %ebp
    movl 8(%ebp), %edx
    decl %edx
    cmpl $4, %edx
    ja .L2
    jmp *.L9(,%edx,4)
.section .rodata
    .align 4
.L9:
    .long .L3
    .long .L5
    .long .L7
    .long .L7
    .long .L5
.text
.L3:  movl $4, %eax
    jmp .L2
.L5:  movl $7, %eax
    jmp .L2
.L7:  movl $11, %eax
.L2:  popl %ebp
    ret
```

Fragment 3

```assembly
woohoo: pushl %ebp
    movl %esp, %ebp
    movl 8(%ebp), %eax
    subl $11,%eax
    je .L6
    subl $11,%eax
    je .L7
    subl $11,%eax
    je .L8
    subl $11,%eax
    je .L7
    jmp .L9
.L6:  movl $4,%eax
    jmp .L4
.L7:  movl $7,%eax
    jmp .L4
.L8:  movl $11,%eax
    jmp .L4
.L9:  movl $11,%eax
.L4:  popl %ebp
    ret
```
2.3 For Loop

This problem tests your understanding of how for loops in C relate to IA32 machine code. Consider the following IA32 assembly code for a procedure dog().

Based on the assembly code, fill in the blanks in its corresponding C source code. (Note: you may only use symbolic variables x, y, i, and result from the source code in your expressions below. Do not use register names.)

```c
int dog(int x, int y)
{
    int i, result;
    result = _____;
    for (i = _____; __________; _____)
    {
        result = _________________;
    }
    return result;
}
```

2.4 Reading Condition Codes with C

In this exercise you will obtain and print the processor’s condition codes for different assembly instructions using a C program. To facilitate your task we have prepared a program skeleton that already does most of the work (ccodes.c). You can download the skeleton (ccodes.c) from the course web page.

On Intel processors the condition codes are stored in the 32-bit wide EFLAGS register. The program skeleton first executes an assembly instruction and stores the resulting contents of the EFLAGS register to a variable. Your task is to complete the function getccodes() (no other part of the program needs to be modified). This function extracts the four condition codes of interest (sign flag, carry flag, zero flag, overflow flag) from the EFLAGS register and stores their values to a C struct of type struct ccodes.

The layout of the EFLAGS register is described in the Intel Architecture Software Developer’s Manual (Volume 1, Section 3.4.3); the link to the Intel manuals is indicated on the course web page.

When the program is complete, compile and run it to test if it functions properly. You can also add new test cases to the main() function.

ADVICE: All required tools to compile the programs should be installed on the lab machines. If you want to build it on your own machine, make sure to install gcc and gcc-multilib (e.g. on Ubuntu: `sudo apt-get install build-essential gcc-multilib`).

Compile ccodes.c with option `-m32` set to generate code for 32-bit environment. For example, the command may look like this: `gcc -m32 -o ccodes ccodes.c`
2.5 Assembly to C

Express the operations of the following assembly language sequence as a C program.

foo:
  pushl %ebp
  movl %esp, %ebp
  movl 8(%ebp), %edx
  movl 8(%ebp), %eax
  movl (%eax), %eax
  incl %eax
  movl %eax, (%edx)
  movl $0, %eax
  movl %ebp, %esp
  popl %ebp
  ret

Give an example how the function foo can be called (provide type declarations for all parameters).

2.6 Parameters in Assembly

What parameters are expected by function huh and what does it do?

huh:
  pushl %ebp
  movl %esp, %ebp
  movl 12(%ebp), %eax
  leal 0(%eax,4), %edx
  movl 8(%ebp), %eax
  movl $999, (%eax,%edx)
  movl $1, %eax
  movl %ebp, %esp
  popl %ebp
  ret

2.7 Array Basics

For each of the arrays declared below provide

a) the size of one element in bytes,

b) the total size of the array in bytes,

c) the byte address of element \(i\) if the array starts at address \(x<arrayidentifier>\),

d) two different C expressions for accessing element \(i\) of the array.

e) a C expression that dereferences an actual char, short, or int, at index 2 (index (2,0), index (2,0,0) respectively) of the array (i.e. char value = A[2] ).

```c
char A[5];
char *B[3];
```
char **C[8];
short D[2];
short *E[9];
int F[4];
int *G[7];

Hand In Instructions

Except for Question 2.4, this is a paper exercise. If you want your solution to be revised please hand it in during your exercise class on the due date. Upload your ccodes.c (Question 2.4) to a subfolder assignment2 of your SVN folder. Refer to Assignment 1 for instructions on using SVN.