**ICS 51: Introductory Computer Organization** **[Winter 2015]**

**Lab#1 (lab1.zip)**

**(Due: Sunday Jan. 25 - 11:55pm)**

The primary goal of this lab is to teach you how to use the low-level x86 assembly instructions to implement *conditional statements*. Since this is your first assembly lab, let's briefly overview some of the basic x86 instructions and registers before diving into our main discussion.

**x86 registers:**

Because the processor accesses registers more quickly than it accesses memory, you can make your programs run faster by keeping the most-frequently used data in registers.

For now, just know that the EAX, EDX, ECX, EBX, EBP, EDI, and ESI registers are 32-bit general-purpose registers in x86 architecture, used for temporary data storage and memory access. In this lab we use registers only for *temporary data storage*. We will learn about memory access and also other available registers in future labs.

**Common x86 instructions:**

Here are some of the most useful x86 instructions that you need to use in this lab and future ICS 51 labs. Pay attention to the order of source (src) and destination (dest) operands for each instruction.

| Instruction | Effect | Examples |
|---|---|---|
| **Copying Data** | | |
| mov dest, src | Copy src to dest | mov eax, 10<br>mov ebx, eax |
| **Arithmetic** | | |
| add dest, src | dest = dest + src | add eax, ecx |
| sub dest, src | dest = dest - src | sub ecx, 4 |
| mul reg | edx:eax = eax * reg | mul ecx |
| div reg | edx = edx:eax % reg<br>eax = edx:eax / reg | div ebx |
| **Bitwise Logical Operations** | | |

| | | |
|---|---|---|
| and dest, src | dest = src & dest | add eax, ecx |
| or dest, src | dest = src \| dest | or ecx, 80h |
| xor dest, src | dest = src ^ dest | xor eax, ebx |
| shr dest, count | dest = dest >> count | shr ecx, 2 |
| shl dest, count | dest = dest << count | shl eax, 4 |
| **Conditionals and Unconditional Jumps** | | |
| cmp arg1,arg2 | Compare arg1 to arg2; must immediately precede any of the conditional jump instructions | cmp eax, 0 |
| je label | Jump to label if arg1 == arg2 | je endloop |
| jne label | Jump to label if arg1 != arg2 | jne loopstart |
| jg label | Jump to label if arg1 > arg2 | jg exit |
| jge label | Jump to label if arg1 >= arg2 | jge format_disk |
| jl label | Jump to label if arg1 < arg2 | jl error |
| jle label | Jump to label if arg1 <= arg2 | jle finish |
| jmp label | Unconditional relative jump | jmp exit |

Note that, there are certain restrictions on operands that an instruction can take. As a rule of thumb, there cannot be two memory operands for an instructions. For example, while register-to-register moves are possible, direct memory-to-memory moves are not. In cases where memory transfers are desired, the source memory contents must first be loaded into a register, then can be stored to the destination memory address.

*Move Instruction Syntax*

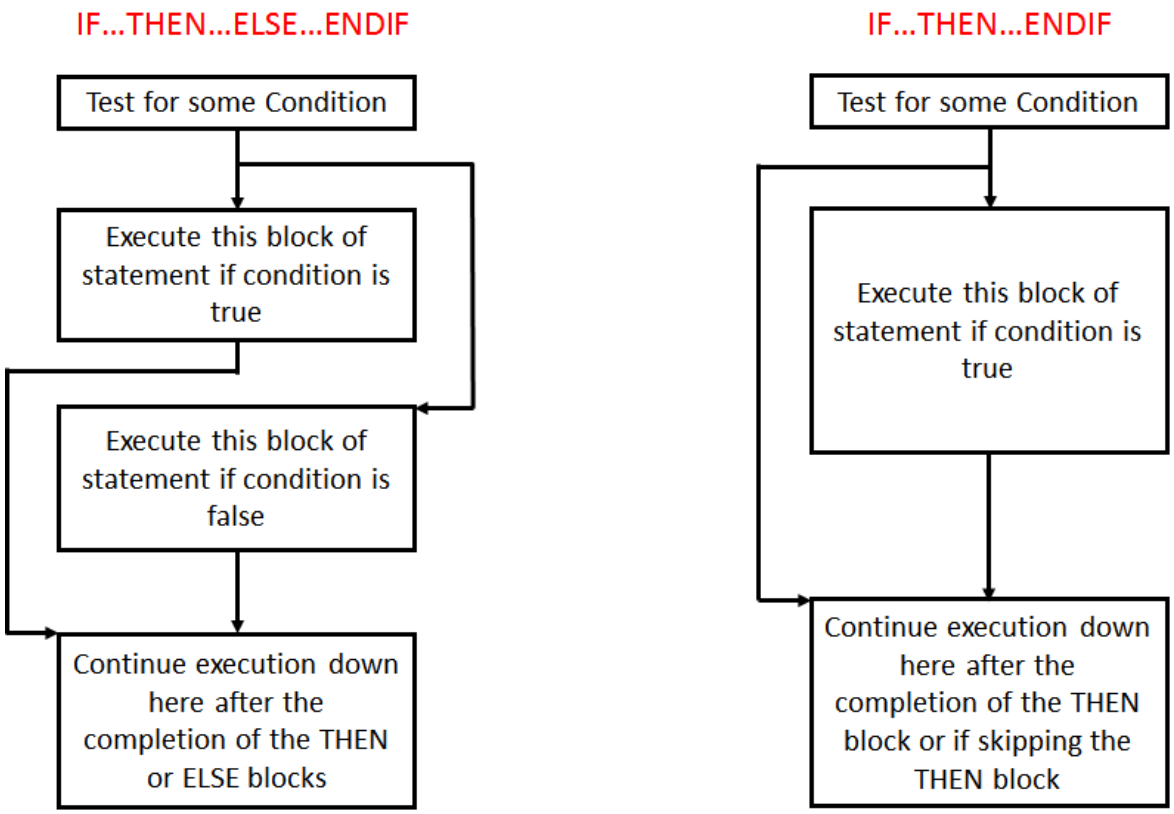
```
mov <reg>,<reg>
mov <reg>,<mem>
mov <mem>,<reg>
mov <reg>,<const>
mov <mem>,<const>
```

Similarly in arithmetic and logical instructions, at most one operand can be a memory operand.

For more information on x86 refer to the discussion slides and other online resources.

**Implementing conditional statements in x86 assembly:**

The most common conditional statements are the IF..THEN or IF..THEN..ELSE statements. These two statements take the form shown in the figure below:




The IF..ENDIF statement is just a special case of the IF..ELSE..ENDIF statement (with an empty ELSE block). Therefore, we will only consider the more general IF..ELSE..ENDIF form. The basic implementation of an IF..THEN..ELSE statement in x86 assembly language looks something like this:

```
      {Sequence of statements to test some condition}

      Jcc ElseCode

      {Sequence of statements corresponding to the THEN block}

      jmp EndOfIF

ElseCode:

      {Sequence of statements corresponding to the ELSE block}

EndOfIF:
```

In the code above, Jcc represents some conditional jump instruction. A list of most common ones was given above in the table of x86 instructions.

For example, to convert the C/C++ statement:

```
     if( a == b )
          c = 0;
     else
          b = b + 1;
```

to assembly language, you could use the following x86 code:

```
          mov eax, a
          cmp eax, b
          jne ElseCode
               mov c, 0
          jmp EndOfIf
ElseCode:
               inc b
 EndOfIF:
```

Should the condition expression become more complex, the associated assembly language code complexity increases as well. To convert a statement like:

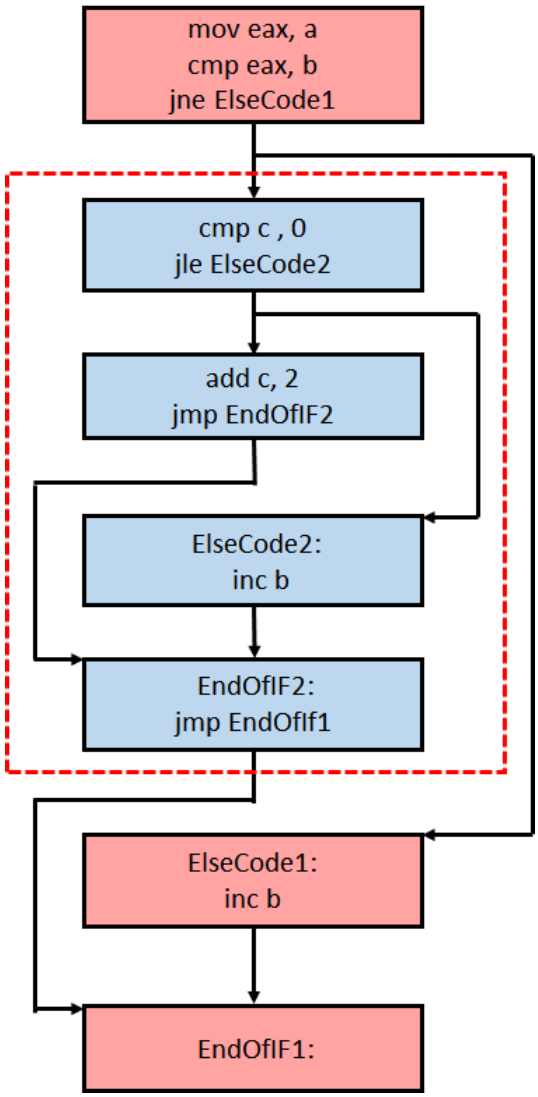
```
     if( a > b && c < 0 )
          b = a - b;
     else
          b = b + 1;
```

to assembly, break the IF statement into a sequence of two different IF statements as follows:

```
     if( a = b )
          if ( c > 0 )
               c = c + 2;
          else
               b = b + 1;
     else
          b = b + 1;
```

and consequently, the corresponding x86 assembly code would be:

```
          mov eax, a
          cmp eax, b
          jne ElseCode1
                    cmp c , 0
                    jle ElseCode2
                         add c, 2
                    jmp EndOfIF2
           ElseCode2:
                         inc b
           EndOfIF2:
           jmp EndOfIf1
ElseCode1:
                  inc b
 EndOfIF1:
```




**Your task:**

Here is the specification of the function you need to write code for in x86 assembly: You are given three integer numbers. You need to add the smallest number to the largest number and multiply that sum by two and return the result. The template code (lab1.c) and the tester (lab1-testing.c) are given to you. Download them from here. You need to make a new project and add these two files to it. However, you are only allowed to make modifications in the marked block of lab1.c. **For more details, read the comment section of lab1.c**.

**VERY IMPORTANT:**

**Submit ONLY the completed lab1.c file**
**and**
**do NOT zip it!**