Lecture Overview

- Logical and shift instructions
- Condition codes
- Set instructions
- Jump instructions
- Conditional move instructions
- How to write in x86 assembly: do while loops, while loops, for loops, switch statements

Logical and Shift Instructions

<table>
<thead>
<tr>
<th>Instructions</th>
<th>Formats</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>orr</td>
<td>D</td>
<td>D gets ¬D (complement)</td>
</tr>
<tr>
<td>and</td>
<td>S, D</td>
<td>D gets D ∧ S (bitwise logical AND)</td>
</tr>
<tr>
<td>or</td>
<td>S, D</td>
<td>D gets D ∨ S (bitwise logical OR)</td>
</tr>
<tr>
<td>xor</td>
<td>S, D</td>
<td>D gets D ⊕ S (bitwise logical XOR)</td>
</tr>
<tr>
<td>sll</td>
<td>k, D</td>
<td>D gets D logically left shifted by k bits</td>
</tr>
<tr>
<td>sar</td>
<td>k, D</td>
<td>D gets D arithmetically right shifted by k bits</td>
</tr>
<tr>
<td>shz</td>
<td>k, D</td>
<td>D gets D logically right shifted by k bits</td>
</tr>
</tbody>
</table>
Examples

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

1 movl 12(%ebp), %eax  \( y \)
2 xorl 8(%ebp), %eax  \( t_1 = x \oplus y \)
3 sarl $3, %eax  \( t_2 = t_1 >> 3 \)
4 notl %eax  \( t_3 = \neg t_2 \)
5 subl 16(%ebp), %eax  \( t_4 = t_3 - z \)

Condition Codes

A register known as EFLAGS on x86

- **CF**: carry flag. Set if the most recent operation caused a carry out of the msb. Overflow for unsigned addition.
- **ZF**: zero flag. Set if the most recent operation generated a result of the value 0.
- **SF**: sign flag. Set if the most recent operation generated a result that is negative.
- **OF**: overflow flag. Set if the most recent operation caused Z's complement overflow.

Instructions related to EFLAGS

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Synonym</th>
<th>Effect</th>
<th>Set condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>sete</td>
<td>sete</td>
<td>( D \leftarrow SF )</td>
<td>Equal / zero</td>
</tr>
<tr>
<td>setne</td>
<td>setnz</td>
<td>( D \leftarrow \neg SF )</td>
<td>Not equal / not zero</td>
</tr>
<tr>
<td>setn</td>
<td>setz</td>
<td>( D \leftarrow SF )</td>
<td>Negative</td>
</tr>
<tr>
<td>setns</td>
<td>setnz</td>
<td>( D \leftarrow \neg SF )</td>
<td>Nonnegative</td>
</tr>
<tr>
<td>setg</td>
<td>sete</td>
<td>( D \leftarrow (SF \oplus CF) )</td>
<td>Greater (signed)</td>
</tr>
<tr>
<td>setle</td>
<td>setne</td>
<td>( D \leftarrow (SF \oplus CF) )</td>
<td>Greater or equal (signed)</td>
</tr>
<tr>
<td>setl</td>
<td>setne</td>
<td>( D \leftarrow SF )</td>
<td>Less (signed)</td>
</tr>
<tr>
<td>setle</td>
<td>setne</td>
<td>( D \leftarrow (SF \oplus CF) )</td>
<td>Less or equal (signed)</td>
</tr>
<tr>
<td>seto</td>
<td>setbo</td>
<td>( D \leftarrow SF )</td>
<td>Above (unsigned)</td>
</tr>
<tr>
<td>setnp</td>
<td>setb</td>
<td>( D \leftarrow \neg SF )</td>
<td>Above or equal (unsigned)</td>
</tr>
<tr>
<td>setnz</td>
<td>setnz</td>
<td>( D \leftarrow SF )</td>
<td>Below (unsigned)</td>
</tr>
<tr>
<td>setns</td>
<td>setnz</td>
<td>( D \leftarrow \neg SF )</td>
<td>Below or equal (unsigned)</td>
</tr>
</tbody>
</table>
setl and flags for 2’s complement
(Refer 3.6.2 in CSAPP Textbook)

1. When no overflow occurs: OF is 0
   a < b if a-b <0 indicated by SF = 1
   a >=b if a-b >=0 indicated by SF =0

2. When overflow occurs: OF is 1
   a < b if a-b >0 (positive overflow) [SF = 0]
   a > b if a-b <0 (negative overflow) [SF = 1]
   (no overflow when a is equal to b)

3. So, to test for a < b, we use SF ^ OF

4. Other signed comparison tests are based on other combinations of SF ^ OF and ZF

---

**More Control Instructions**

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>jmp</td>
<td>label goto label; if EFLAGS set such that previous result was greater than 0</td>
</tr>
<tr>
<td>jne</td>
<td>label goto label if EFLAGS set such that previous result was greater than or equal to 0</td>
</tr>
<tr>
<td>jge</td>
<td>label goto label if EFLAGS set such that previous result was less than or equal to 0</td>
</tr>
<tr>
<td>jle</td>
<td>label goto label if EFLAGS set such that previous result was less than or equal to 0</td>
</tr>
</tbody>
</table>

**Jump: Relative vs Absolute**

(Relevant for Linking which we will cover in later lecture)

- Assembly Jump statements use labels but assembler and later linker translate these labels to actual instruction addresses.
- **PC Relative**: difference between address of target instruction and address right after the jump instruction. (offsets use 1, 2 or 4 bytes)
- **Absolute**: use 4 bytes to directly specify target instruction

Advantages of PC Relative:

1. Instruction can be **compactly** encoded
2. Object code can be **shifted to different positions in memory without alteration**
“if” and “if else” Stmts in Assembly

Overview of “if” and “if else” statement:

| if(condition){ | if(condition){ |
| statements; | statements1; |
| } | } 
| else{ | else{ |
| statements2; | } |

General Approach:
1. Use compare instructions to set the condition codes
2. Then use the jump instructions to execute the right set of instructions

if (x == y) {
    x++;
}

Assumptions:
- x and y are both integers
- x is already in %eax
- y is already in %edx

“if else” example

if (x<y) {
    return y-x;
} else {
    return x-y;
}

x at %ebp+8, y at %ebp+12

```
cmpl %ecx, %edx  # If set if they were equal
jne skip_incr  
incl %ecx  # x++

skip_incr:
```

```
1 movl 8(%ebp), %edx  # Get x
2 movl 12(%ebp), %eax  
3 cmpl %eax, %edx  
4 jge .L2  
5 subl %edx, %eax  
6 jmp .L3  
7 .L2:  
8 subl %eax, %edx  
9 movl %edx, %eax  
10 .L3: done: Begin completion code
```

Get y
Compare x:y
if >= go to L2
result = y-x
Goto done
result = x-y
%eax = result