CMPT 295
Machine-Level Programming
Lecture 12 – Program Control – Conditional Statement
Last Lecture

- **Arithmetic & logical operations**
  - **Arithmetic instructions:** \( \text{add}^*, \text{sub}^*, \text{imul}^* \)
    - \( \text{inc}^*, \text{dec}^*, \text{neg}^*, \text{not}^* \)
    - **Examples:** \text{addq}, \text{subq}, \text{imulq}, \text{incq}, \text{decq}, \text{negq}, \text{notq}
  - **Logical instructions:** \( \text{and}^*, \text{or}^*, \text{xor}^* \)
    - **Examples:** \text{andq}, \text{orq}, \text{xorq}
  - **Shift instructions:** \( \text{sal}^*, \text{sar}^*, \text{shr}^* \)
    - **Examples:** \text{salq}, \text{sarq}, \text{shrq}
  - **ATT format:** Watch out for argument order!
Today’s Menu

- Introduction
  - C program -> assembly code -> machine level code
  - Assembly language basics: data, move operation
    - Memory addressing modes
- Operation leaq and Arithmetic & logical operations
- Conditional Statement
- Loop
- Function/procedure call – Stack – Recursion
- Array
- Floating-point operations
Program Control Overview

- We can change the execution flow of a program
  1. Based on a condition
  2. Unconditionally

  in C
  - Control statements
    - if/else
    - switch
    - loops
  - function calls

  in x86-64 assembly
  - Branching:
    - cmp* instruction (compare)
    - jX instructions (jump)

  call and ret
Conditional statement: if/else

in C:
void func(long x, long y) {
    if (x < y) {
        // stmts true
    } else {
        // stmts false
    }
    return;
}

in assembly:
func:
    cmpq %rsi, %rdi # x – y
    jge else #
    ... # stmts true
    jmp endif #
else: ...
endif: ret #

We branch (jump) when the condition is false
-> this method is called coding the false condition first
**cmp** instruction (compare)

- **Syntax:** `cmpq Src2, Src1`

- **Meaning (Effect):** `cmpq b, a`
  - Like computing `a - b` without setting destination
test* instruction (test)

- **Syntax:** `testq Src2, Src1`

- **Meaning (Effect):** `testq b, a`
  - Like computing `a & b` without setting destination
```markdown
## jx Instructions (Branching)

- Jump to different part of code depending result of previous instructions (i.e., condition codes)

<table>
<thead>
<tr>
<th>jx</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>jmp</td>
<td>Unconditional</td>
</tr>
<tr>
<td>je</td>
<td>Equal / Zero</td>
</tr>
<tr>
<td>jne</td>
<td>Not Equal / Not Zero</td>
</tr>
<tr>
<td>js</td>
<td>Negative</td>
</tr>
<tr>
<td>jns</td>
<td>Nonnegative</td>
</tr>
<tr>
<td>jg</td>
<td>Greater (Signed)</td>
</tr>
<tr>
<td>jge</td>
<td>Greater or Equal (Signed)</td>
</tr>
<tr>
<td>jl</td>
<td>Less (Signed)</td>
</tr>
<tr>
<td>jle</td>
<td>Less or Equal (Signed)</td>
</tr>
<tr>
<td>ja</td>
<td>Above (unsigned)</td>
</tr>
<tr>
<td>jb</td>
<td>Below (unsigned)</td>
</tr>
</tbody>
</table>
```
Example – int abs(int x)

in C:

```c
int abs(int x){
    if ( x < 0 )
        x = -x;
    return x;
}
```

in assembly:

```assembly
# x in edi, result in eax
movl %edi, %eax  # eax <- x
    #
    # ret if x >= 0
    # x = -x
endif:
    ret
```
In C, we can change the execution flow of a program

1. Conditionally
   - Conditional statements: if/else, switch
   - Iterative statements: loops

2. Unconditionally
   - Functions calls

In x86-64 assembly, we can also change the execution flow of a program

- `cmp*` instruction (compare)
- `jX` instructions (jump)
- `call` and `ret` instructions

How conditional statement `if/else` can be implemented in x86-64 assembly
Next lecture

- Introduction
  - C program -> assembly code -> machine level code
  - Assembly language basics: data, move operation
    - Memory addressing modes
  - Operation leaq and Arithmetic & logical operations
  - Conditional Statement – Condition Code + cmovX
- Loop
- Function/procedure call – Stack – Recursion
- Array
- Floating-point operations