CMPT 295

Unit - Machine-Level Programming

Lecture 12 – Assembly language – Program Control – Conditional Statements

Last Lecture

- Demo
  - Observation: C compiler will figure out different instruction combinations to carry out the computations in our C code

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 – Condition Code + `cmovX`
- Loops
- Function call – Stack
- Array
- Buffer Overflow
- Floating-point operations

Completing our Demo

1. ✓ `gcc` uses `leaq` for addition -> `sum_store.c`
2. ✓ Writing our own assembly code (`arith.s`) using arithmetic instructions of x86-64 assembly language
3. `makefile`
   - when we compile our own `*.s` files with `*.c` files
   - when we compile only `*.c` files
4. How would `gcc` compile our `arith.c` into `arith.s`?

Program Control Overview

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

in C

- Control statements

Conditional statements
- `if/else`
- `switch`

Iterative statements
- `for` loop
- `while` and `do while` loops

- function calls → **`call`** and **`ret`**

in x86-64 assembly

Branching:
- **`cmp*`** instruction (compare)
- **`jx`** instructions (jump)

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:   ...            # stmts false
endif:  ret            #
```

label

label

A **label** is a memory address

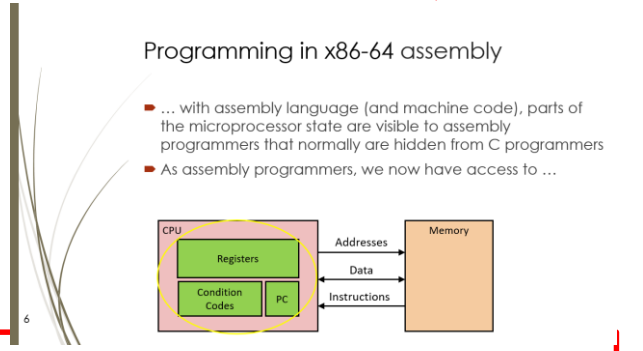
We branch (jump) when the condition is false

-> This technique is called *"coding the false condition first"*

```
* -> Size designator
q -> long  64
l -> int   32
w -> short 16
b -> char   8
```

comparison instructions

Remember in Lecture 9, we saw …



Syntax	Meaning/Effect	Example
cmp* ***Src2, Src1***	**Src1 – Src2 -> > 0? -> Src1 > Src2** **= 0? -> Src1 == Src2** **< 0? -> Src1 < Src2** without saving the result in the destination operand (no **Dest**)	`cmpq %rsi,%rdi`

- Sets ***condition codes*** based on value of **Src1 – Src2**

test* ***Src2, Src1***	**Src1 & Src2 ->** without saving the result in the destination operand (no **Dest**)	`testq %rax,%rax`

- Sets ***condition codes*** based on value of **Src1 & Src2**
- Useful when one of the operands is a ***bit mask***

`jx` jump family instructions (branching)

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

`jx`	Description
`jmp`	Unconditional
`je`	Equal / Zero
`jne`	Not Equal / Not Zero
`js`	Negative
`jns`	Nonnegative
`jg`	Greater (Signed)
`jge`	Greater or Equal (Signed)
`jl`	Less (Signed)
`jle`	Less or Equal (Signed)
`ja`	Above (unsigned)
`jb`	Below (unsigned)

Example – `int abs(int x)`

in C:

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

in assembly:

```
    # x in edi, result in eax
abs:
    movl %edi, %eax # eax <- x
                    #
                    # ret if x >= 0
                    # x = -x
endif:
    ret
```

int max(int x, int y)- Homework

in C:

```
int max(int x, int y){
  int result = x;
  if (y > x)
    result = y;
  return result;
}
```

in assembly:

```
    # x in edi, y in esi, result in eax
max:
    movl %edi, %eax # result = x




endif:
    ret
```

Summary

- In C, we can change the execution flow of a program
  1. Conditionaly
     - 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

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`
- Loops
- Function call – Stack
- Array
- Buffer Overflow
- Floating-point operations