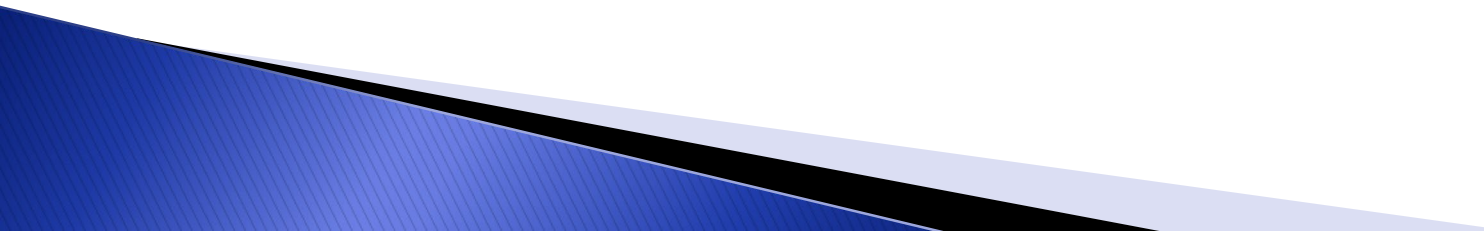# X86–64 Assembly Language Part 3: Control

Required reading: Section 3.6.6

Today

- Control: Condition codes
- **Conditional branches**
- Loops

Jumping

- jX Instructions
  - Jump to different part of code depending on condition codes

jX	Condition	Description
jmp	1	Unconditional
je	ZF	Equal / Zero
jne	~ZF	Not Equal / Not Zero
js	SF	Negative
jns	~SF	Nonnegative
jg	~(SF^OF)&~ZF	Greater (Signed)
jge	~(SF^OF)	Greater or Equal (Signed)
jl	(SF^OF)	Less (Signed)
jle	(SF^OF)|ZF	Less or Equal (Signed)
ja	~CF&~ZF	Above (unsigned)
jb	CF	Below (unsigned)

Conditional Branch Example (Old Style)

▸ C code example with assembly

```
long absdiff
  (long x, long y)
{
  long result;
  if (x > y)
      result = x-y;
  else
      result = y-x;
  return result;
}
```

```
absdiff:
    cmpq      %rsi, %rdi   # x:y
    jle       .L4
    movq      %rdi, %rax
    subq      %rsi, %rax
    ret
.L4:              # x <= y
    movq      %rsi, %rax
    subq      %rdi, %rax
    ret
```

Register	Use(s)
%rdi	Argument **x**
%rsi	Argument **y**
%rax	Return value

Expressing with Goto Code

- C allows `goto` statement
- Jump to position designated by label

```
long absdiff
   (long x, long y)
{
    long result;
    if (x > y)
        result = x-y;
    else
        result = y-x;
    return result;
}
```

```
long absdiff_j
   (long x, long y)
{
    long result;
    int ntest = x <= y;
    if (ntest) goto Else;
    result = x-y;
    goto Done;
 Else:
    result = y-x;
 Done:
    return result;
}
```

General Conditional Expression Translation (Using Branches)

C Code

```
val = Test ? Then_Expr : Else_Expr;
```

```
val = x>y ? x-y : y-x;
```

Goto Version

```
    ntest = !Test;
    if (ntest) goto Else;
    val = Then_Expr;
    goto Done;
Else:
    val = Else_Expr;
Done:
    . . .
```

- Create separate code regions for then & else expressions
- Execute appropriate one

Using Conditional Moves

▸ Conditional Move Instructions
  ◦ Instruction supports:

    if (Test) Dest ← Src
  ◦ Supported in post–1995 x86 processors
  ◦ GCC tries to use them
    • But, only when known to be safe
▸ Why?
  ◦ **Incorrectly predicted branches are very disruptive to instruction flow through pipelines**
  ◦ Conditional moves do not require control transfer
  ◦ On out–of–order processors, correctly predicted branches usually outperform conditional move

C Code

```
val = Test
    ? Then_Expr
    : Else_Expr;
```

Goto Version

```
result = Then_Expr;
eval = Else_Expr;
nt = !Test;
if (nt) result = eval;
return result;
```

Conditional Move Example

```c
long absdiff
  (long x, long y)
{
    long result;
    if (x > y)
        result = x-y;
    else
        result = y-x;
    return result;

}
```

Register	Use(s)
`%rdi`	Argument **x**
`%rsi`	Argument **y**
`%rax`	Return value

```
movq    %rdi, %rax  # x
subq    %rsi, %rax  # result = x-y
movq    %rsi, %rdx
subq    %rdi, %rdx  # eval = y-x
cmpq    %rsi, %rdi  # x:y
cmovle  %rdx, %rax  # if <=, result = eval
ret
```

Bad Cases for Conditional Move

Expensive Computations

```
val = Test(x) ? Hard1(x) : Hard2(x);
```

- ▸ Both values get computed
- ▸ Only makes sense when computations are very simple

Risky Computations

```
val = p ? *p : 0;
```

- Both values get computed
- May have undesirable effects

Computations with side effects

```
val = x > 0 ? x*=7 : x+=3;
```

- Both values get computed
- Must be side-effect free

Usage notes

- Conditional move has a restricted set of operands
  - Destination must be a register
  - Source is either memory or register, but not immediate

- If you need to conditional move from an immediate value, first move the immediate value to a register and then conditional move that.

- If you need to move to memory, either recode using a branch or conditional move to a register and move that to memory