

# Conditional Branch Example

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

absdiff:		
pushl	%ebp	} Setup
movl	%esp, %ebp	
movl	8(%ebp), %edx	} Body1
movl	12(%ebp), %eax	
cmpl	%eax, %edx	
jle	.L7	
subl	%eax, %edx	
movl	%edx, %eax	
.L8:		
leave		} Finish
ret		
.L7:		
subl	%edx, %eax	} Body2
jmp	.L8	

## Conditional Branch Example (Cont.)

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

```
int goto_ad(int x, int y)
{
    int result;
    if (x <= y) goto Else;
    result = x-y;
Exit:
    return result;
Else:
    result = y-x;
    goto Exit;
}
```

- C allows “goto” as means of transferring control
  - Closer to machine-level programming style
- Generally considered bad coding style

# Conditional Branch Example (Cont.)

```
int goto_ad(int x, int y)
{
    int result;
    if (x <= y) goto Else;
    result = x-y;
Exit:
    return result;
Else:
    result = y-x;
    goto Exit;
}
```

```
int x          %edx
int y          %eax
```

```
absdiff:
    pushl    %ebp
    movl     %esp, %ebp
    movl     8(%ebp), %edx
    movl     12(%ebp), %eax
    cmpl     %eax, %edx
    jle      .L7
    subl     %eax, %edx
    movl     %edx, %eax
.L8:
    leave
    ret
.L7:
    subl     %edx, %eax
    jmp      .L8
```

# Conditional Branch Example (Cont.)

```

int goto_ad(int x, int y)
{
    int result;
    if (x <= y) goto Else;
    result = x-y;
Exit:
    return result;
Else:
    result = y-x;
    goto Exit;
}

```

int x	%edx
int y	%eax

```

absdiff:
    pushl    %ebp
    movl     %esp, %ebp
    movl     8(%ebp), %edx
    movl     12(%ebp), %eax
    cmpl     %eax, %edx
    jle      .L7
    subl     %eax, %edx
    movl     %edx, %eax
.L8:
    leave
    ret
.L7:
    subl     %edx, %eax
    jmp      .L8

```

# Conditional Branch Example (Cont.)

```

int goto_ad(int x, int y)
{
    int result;
    if (x <= y) goto Else;
    result = x-y;
Exit:
    return result;
Else:
    result = y-x;
    goto Exit;
}

```

int x	%edx
int y	%eax

```

absdiff:
    pushl    %ebp
    movl     %esp, %ebp
    movl     8(%ebp), %edx
    movl     12(%ebp), %eax
    cmpl     %eax, %edx
    jle      .L7
    subl     %eax, %edx
    movl     %edx, %eax
.L8:
    leave
    ret
.L7:
    subl     %edx, %eax
    jmp      .L8

```

# Conditional Branch Example (Cont.)

```

int goto_ad(int x, int y)
{
    int result;
    if (x <= y) goto Else;
    result = x-y;
Exit:
    return result;
Else:
    result = y-x;
    goto Exit;
}

```

int x	%edx
int y	%eax

```

absdiff:
    pushl    %ebp
    movl     %esp, %ebp
    movl     8(%ebp), %edx
    movl     12(%ebp), %eax
    cmpl     %eax, %edx
    jle      .L7
    subl     %eax, %edx
    movl     %edx, %eax
.L8:
    leave
    ret
.L7:
    subl     %edx, %eax
    jmp      .L8

```

# Conditional Branch Example (Cont.)

```

int goto_ad(int x, int y)
{
    int result;
    if (x <= y) goto Else;
    result = x-y;
Exit:
    return result;
Else:
    result = y-x;
    goto Exit;
}

```

int x	%edx
int y	%eax

```

absdiff:
    pushl    %ebp
    movl     %esp, %ebp
    movl     8(%ebp), %edx
    movl     12(%ebp), %eax
    cmpl     %eax, %edx
    jle      .L7
    subl     %eax, %edx
    movl     %edx, %eax
.L8:
    leave
    ret
.L7:
    subl     %edx, %eax
    jmp      .L8

```

# General Conditional Expression Translation

## C Code

```
val = Test ? Then-Expr : Else-Expr;
```

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

```
if (Test)
    val = Then-Expr;
else
    val = Else-Expr;
```

## Goto Version

```
nt = !Test;
if (nt) goto Else;
val = Then-Expr;
Done:
. . .
Else:
    val = Else-Expr;
    goto Done;
```

- *Test* is expression returning integer  
= 0 interpreted as false  
≠ 0 interpreted as true
- Create separate code regions for then & else expressions
- Execute appropriate one
- How might you make this more efficient?



# Conditionals: x86-64

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


```
absdiff: # x in %edi, y in %esi
    movl    %edi, %eax    # eax = x
    movl    %esi, %edx    # edx = y
    subl    %esi, %eax    # eax = x-y
    subl    %edi, %edx    # edx = y-x
    cmpl    %esi, %edi    # x:y
    cmovle  %edx, %eax    # eax=edx if <=
    ret
```

## ■ Conditional move instruction

- **cmovC** src, dest
- Move value from src to dest if condition **C** holds
- More efficient than conditional branching (simple control flow)
- But overhead: both branches are evaluated

# PC Relative Addressing

0x100	cmp	r2, r3	0x1000
0x102	je	0x70	0x1002
0x104	...		0x1004
...	...		...
0x172	add	r3, r4	0x1072



- PC relative branches are relocatable
- Absolute branches are not