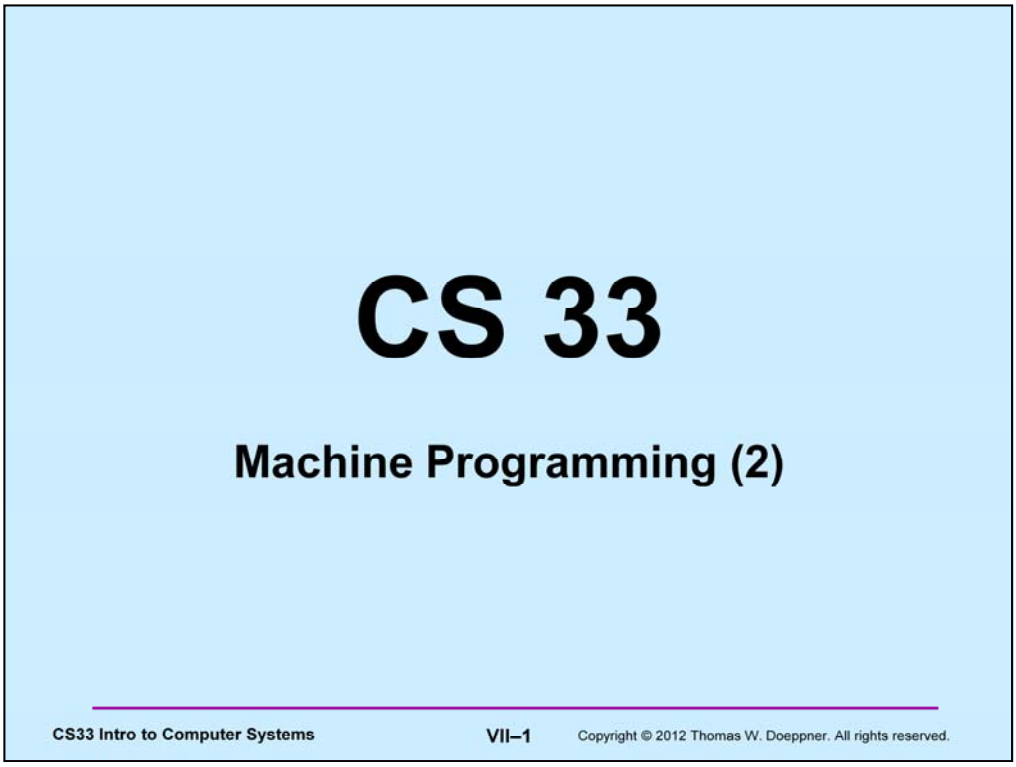
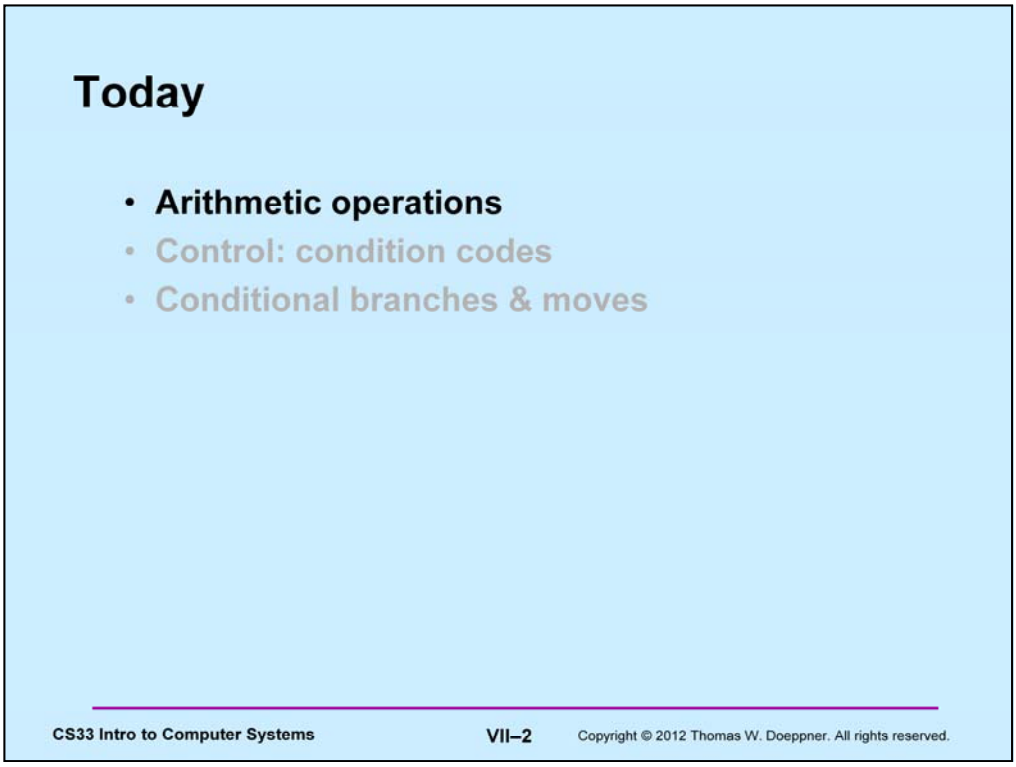


All of the slides in this lecture are either from or adapted from slides provided by the authors of the textbook "Computer Systems: A Programmer's Perspective," 2nd Edition and are provided from the website of Carnegie-Mellon University, course 15-213, taught by Randy Bryant and David O'Hallaron in Fall 2010. These slides are indicated "Supplied by CMU" in the notes section of the slides.



Supplied by CMU.

Some Arithmetic Operations

- **Two-operand instructions:**

Format	*Computation*		
`addl`	*Src,Dest*	Dest = Dest + Src	
`subl`	*Src,Dest*	Dest = Dest – Src	
`imull`	*Src,Dest*	Dest = Dest * Src	
`sall`	*Src,Dest*	Dest = Dest << Src	*Also called shll*
`sarl`	*Src,Dest*	Dest = Dest >> Src	*Arithmetic*
`shrl`	*Src,Dest*	Dest = Dest >> Src	*Logical*
`xorl`	*Src,Dest*	Dest = Dest ^ Src	
`andl`	*Src,Dest*	Dest = Dest & Src	
`orl`	*Src,Dest*	Dest = Dest | Src	

  - watch out for argument order!
  - no distinction between signed and unsigned int (why?)

Supplied by CMU.

Some Arithmetic Operations

- **One-operand Instructions**

`incl`	*Dest = Dest + 1*
`decl`	*Dest = Dest – 1*
`negl`	*Dest = – Dest*
`notl`	*Dest = ~Dest*

- **See book for more instructions**

Supplied by CMU.

Arithmetic Expression Example

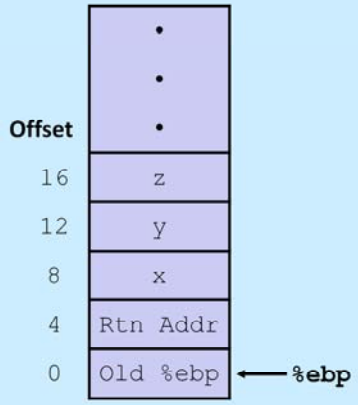
```
int arith(int x, int y, int z)
{
  int t1 = x+y;
  int t2 = z+t1;
  int t3 = x+4;
  int t4 = y * 48;
  int t5 = t3 + t4;
  int rval = t2 * t5;
  return rval;
}
```

```
arith:
  pushl  %ebp              } Set
  movl   %esp, %ebp        } Up

  movl   8(%ebp), %ecx     }
  movl   12(%ebp), %edx    }
  leal   (%edx,%edx,2), %eax }
  sall   $4, %eax          }
  leal   4(%ecx,%eax), %eax } Body
  addl   %ecx, %edx        }
  addl   16(%ebp), %edx    }
  imull  %edx, %eax        }

  popl   %ebp              } Finish
  ret                      }
```


Supplied by CMU.

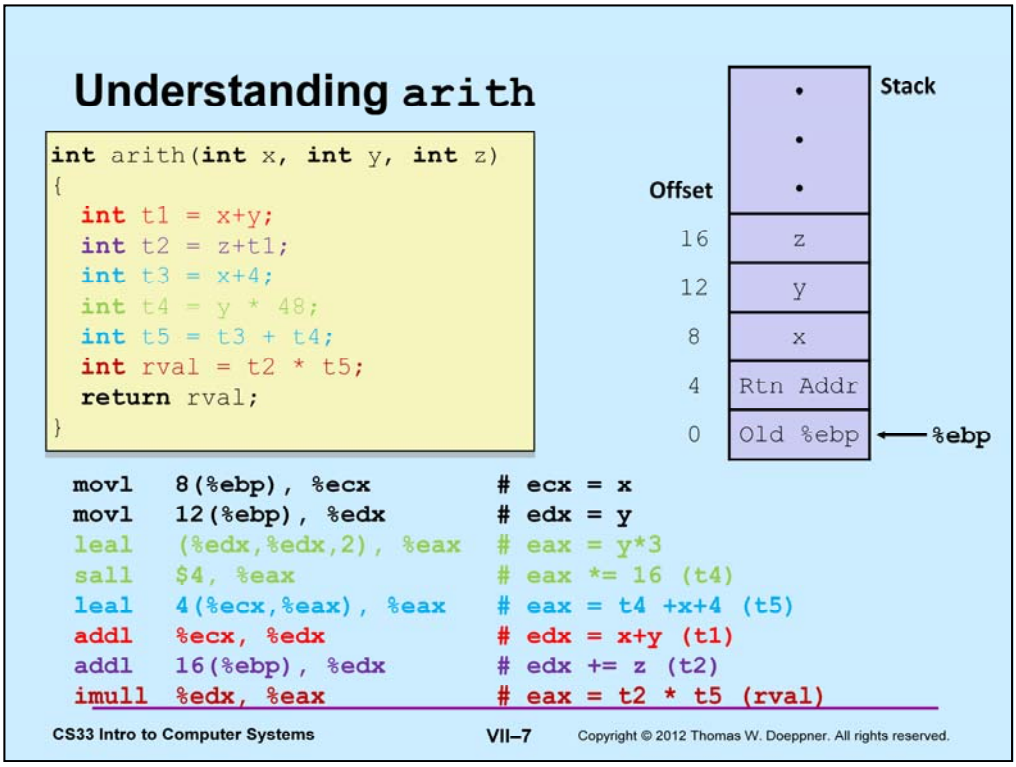
Understanding `arith`

```
int arith(int x, int y, int z)
{
  int t1 = x+y;
  int t2 = z+t1;
  int t3 = x+4;
  int t4 = y * 48;
  int t5 = t3 + t4;
  int rval = t2 * t5;
  return rval;
}
```




```
  movl   8(%ebp), %ecx
  movl   12(%ebp), %edx
  leal   (%edx,%edx,2), %eax
  sall   $4, %eax
  leal   4(%ecx,%eax), %eax
  addl   %ecx, %edx
  addl   16(%ebp), %edx
  imull  %edx, %eax
```


Supplied by CMU.



Supplied by CMU.

Observations about `arith`

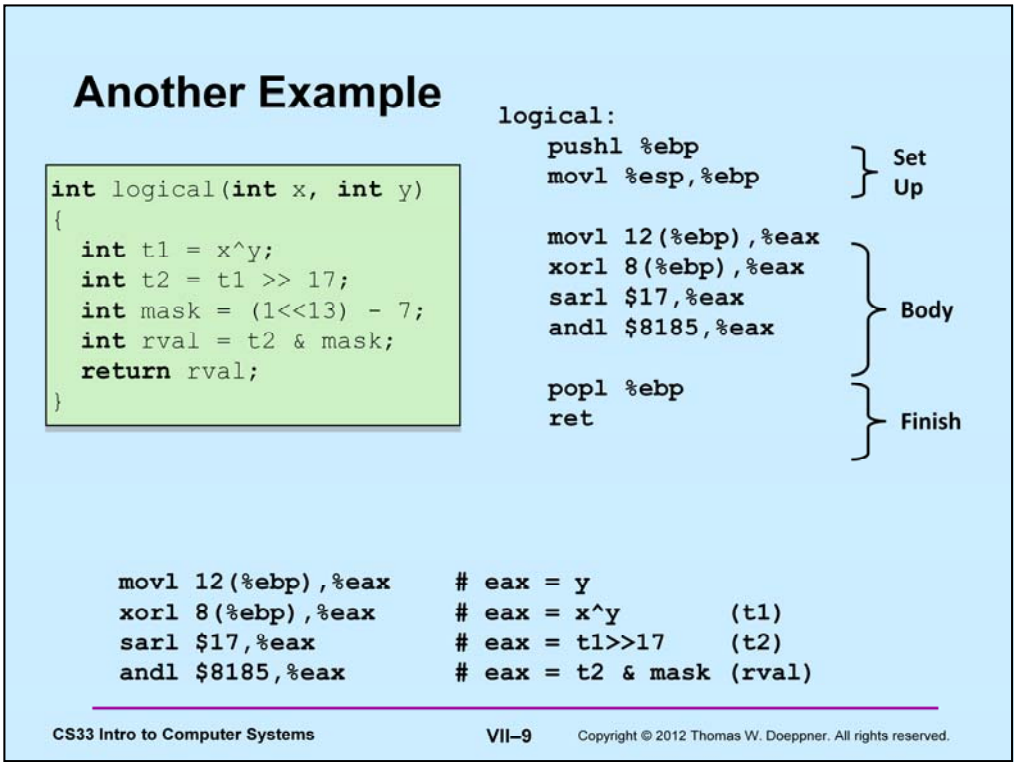
```
int arith(int x, int y, int z)
{
  int t1 = x+y;
  int t2 = z+t1;
  int t3 = x+4;
  int t4 = y * 48;
  int t5 = t3 + t4;
  int rval = t2 * t5;
  return rval;
}
```

- Instructions in different order from C code
- Some expressions require multiple instructions
- Some instructions cover multiple expressions
- Get exact same code when compile:

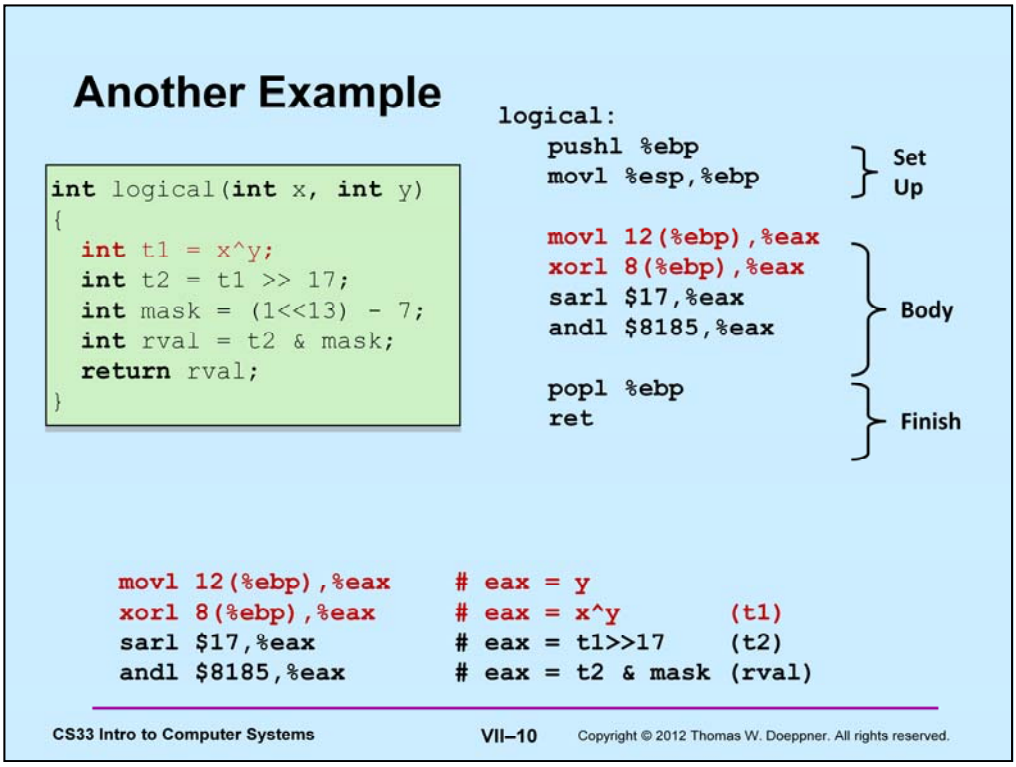
  `(x+y+z)*(x+4+48*y)`

```
movl   8(%ebp), %ecx          # ecx = x
movl   12(%ebp), %edx         # edx = y
leal   (%edx,%edx,2), %eax    # eax = y*3
sall   $4, %eax               # eax *= 16 (t4)
leal   4(%ecx,%eax), %eax     # eax = t4 +x+4 (t5)
addl   %ecx, %edx             # edx = x+y (t1)
addl   16(%ebp), %edx         # edx += z (t2)
imull  %edx, %eax             # eax = t2 * t5 (rval)
```

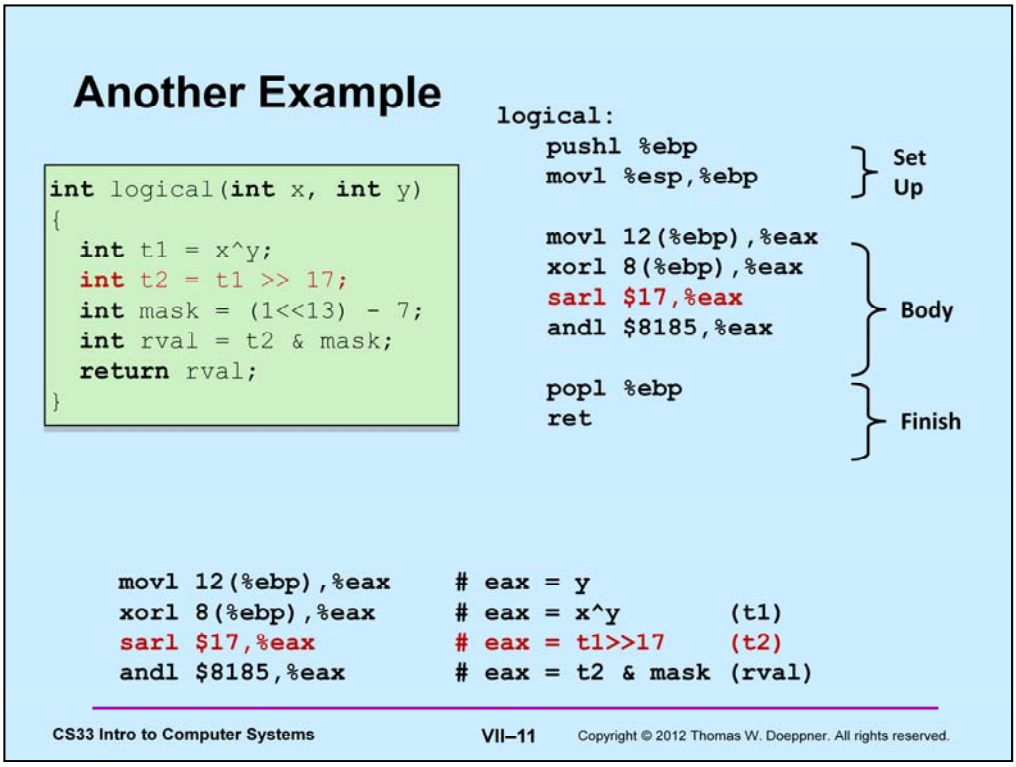

Supplied by CMU.



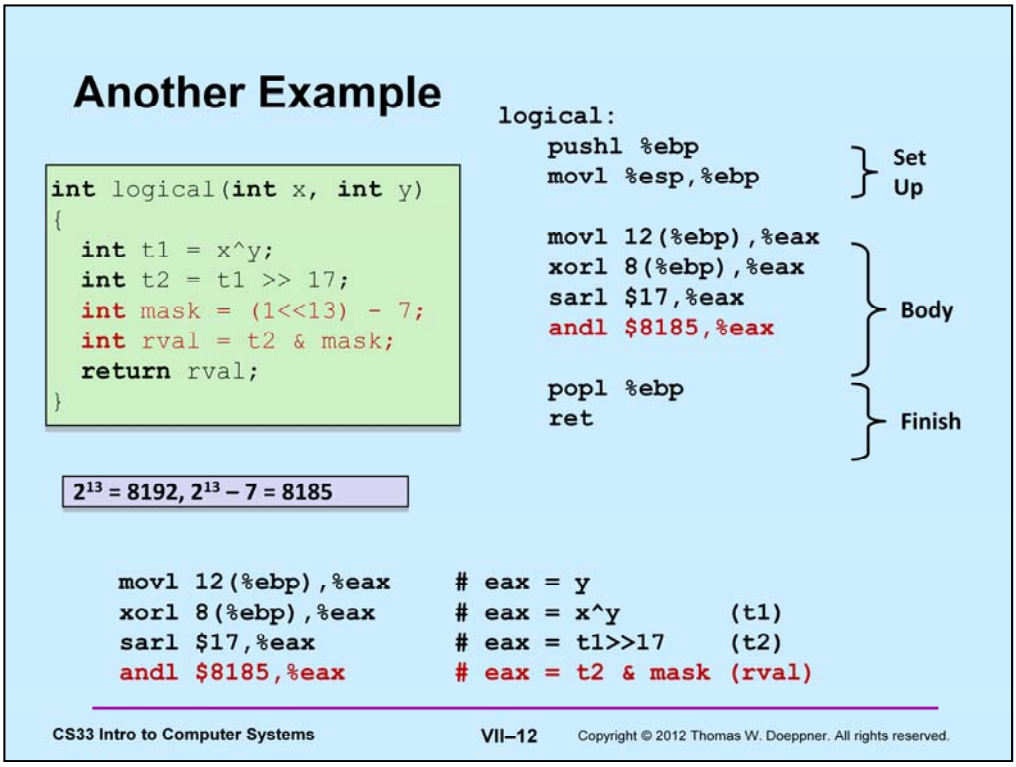
Supplied by CMU.



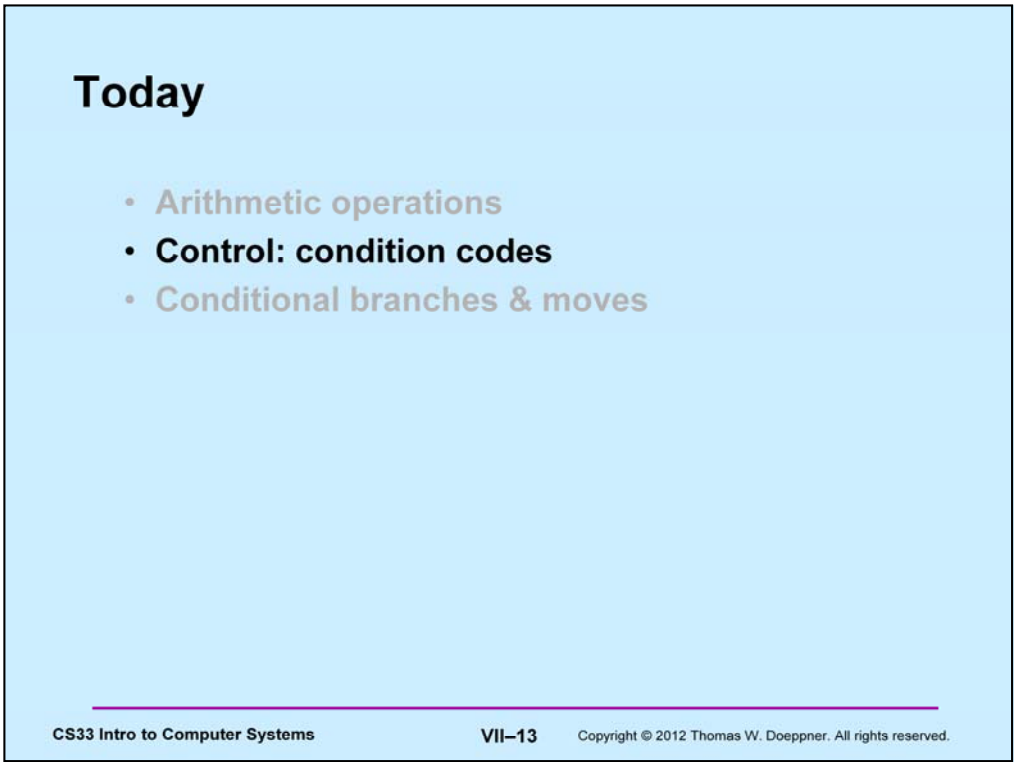
Supplied by CMU.



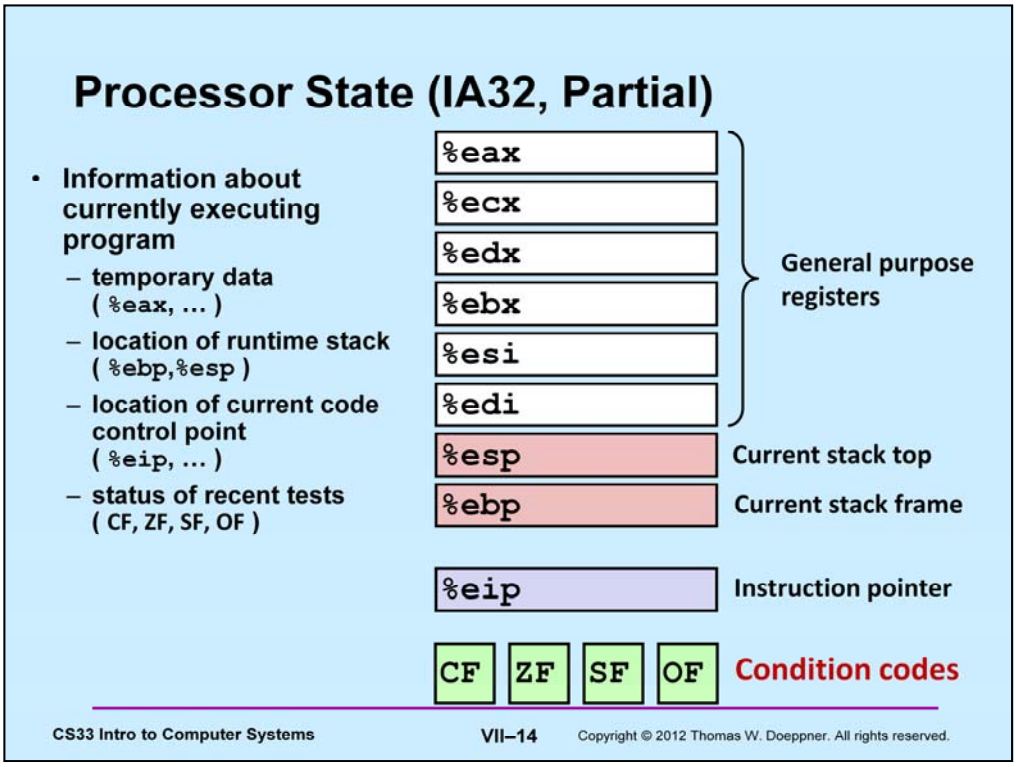
Supplied by CMU.



Supplied by CMU.



Supplied by CMU.



Supplied by CMU.

Condition Codes (Implicit Setting)

- **Single bit registers**

CF	carry flag (for unsigned)	SF	sign flag (for signed)
ZF	zero flag	OF	overflow flag (for signed)

- **Implicitly set (think of it as side effect) by arithmetic operations**

  example: `addl/addq` *Src*,*Dest* ↔ `t = a+b`

  CF set if carry out from most significant bit (unsigned overflow)

  ZF set if `t == 0`

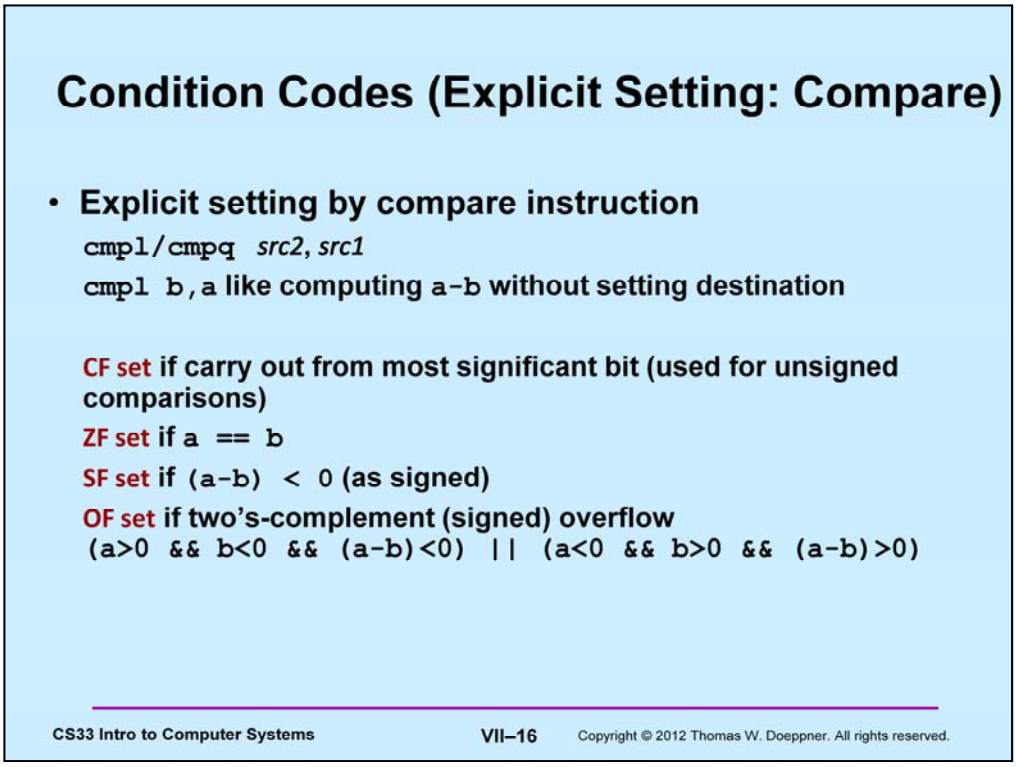
  SF set if `t < 0` (as signed)

  OF set if two's-complement (signed) overflow

```
(a>0 && b>0 && t<0) || (a<0 && b<0 && t>=0)
```

- **Not set by `lea` instruction**

Supplied by CMU.



Supplied by CMU.

Condition Codes (Explicit Setting: Test)

- **Explicit setting by test instruction**

  `testl`/`testq` *src2*, *src1*
  `testl b,a` like computing `a&b` without setting destination

  – sets condition codes based on value of *Src1* & *Src2*
  – useful to have one of the operands be a mask

  ZF set when `a&b == 0`
  SF set when `a&b < 0`

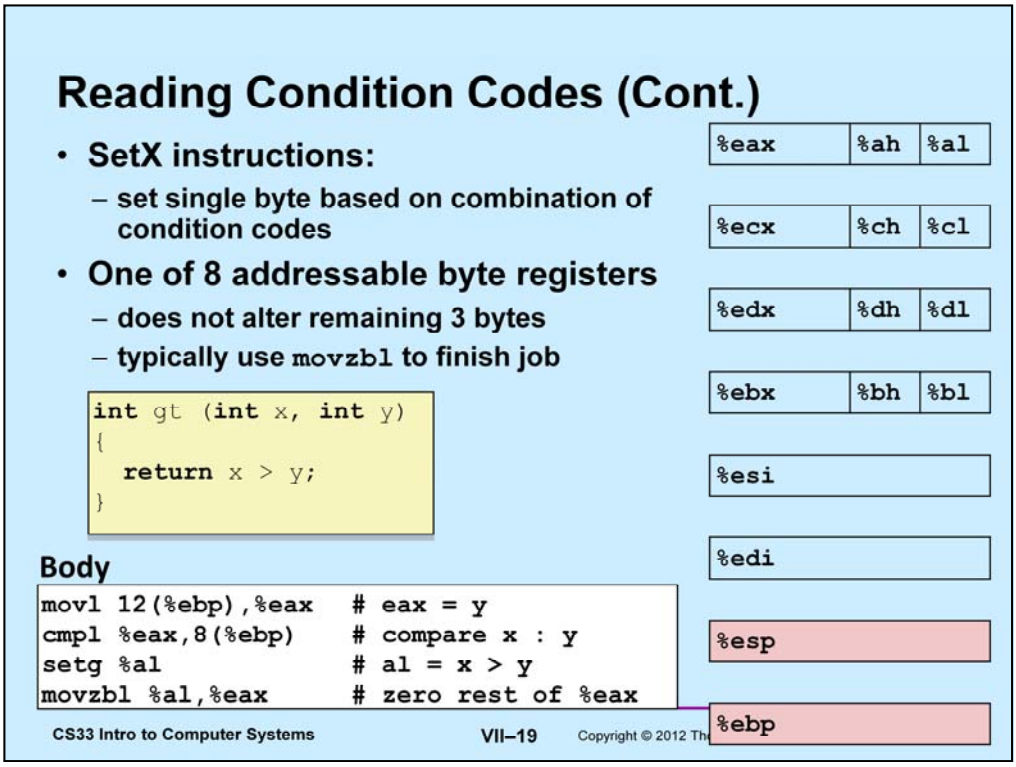
Supplied by CMU.

Reading Condition Codes

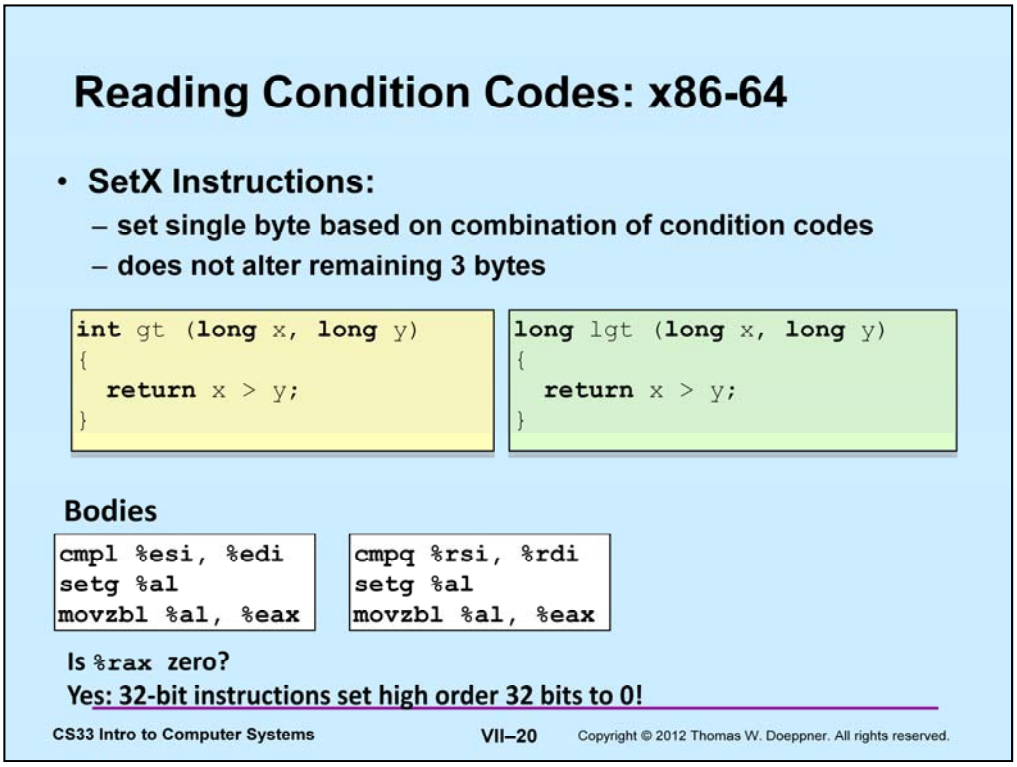
- **SetX instructions**
  - set single byte based on combinations of condition codes

SetX	Condition	Description
sete	ZF	Equal / Zero
setne	~ZF	Not Equal / Not Zero
sets	SF	Negative
setns	~SF	Nonnegative
setg	~(SF^OF)&~ZF	Greater (Signed)
setge	~(SF^OF)	Greater or Equal (Signed)
setl	(SF^OF)	Less (Signed)
setle	(SF^OF)|ZF	Less or Equal (Signed)
seta	~CF&~ZF	Above (unsigned)
setb	CF	Below (unsigned)

Supplied by CMU.



Supplied by CMU.



Supplied by CMU.

Today

- Arithmetic operations
- x86-64
- Control: condition codes
- **Conditional branches & moves**

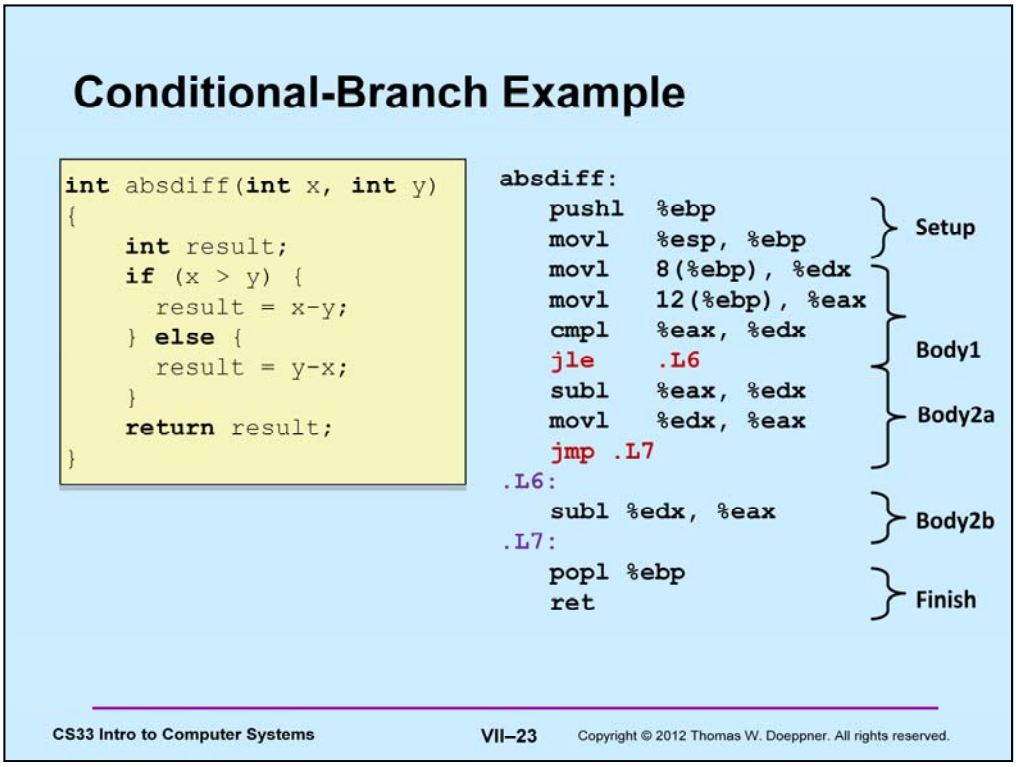
Supplied by CMU.

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)

Supplied by CMU.



Supplied by CMU.

Conditional-Branch Example (Cont.)

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

- C allows "goto" as means of transferring control
  - closer to machine-level programming style
- Generally considered bad coding style

```
absdiff:
    pushl   %ebp                    } Setup
    movl    %esp, %ebp              }
    movl    8(%ebp), %edx           } Body1
    movl    12(%ebp), %eax          }
    cmpl    %eax, %edx              }
    jle     .L6                     }
    subl    %eax, %edx              } Body2a
    movl    %edx, %eax              }
    jmp .L7                         }
.L6:
    subl %edx, %eax                 } Body2b
.L7:
    popl %ebp                       } Finish
    ret                             }
```


Supplied by CMU.

Conditional-Branch Example (Cont.)

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

```
absdiff:
    pushl  %ebp                    } Setup
    movl   %esp, %ebp              }
    movl   8(%ebp), %edx           } Body1
    movl   12(%ebp), %eax          }
    cmpl   %eax, %edx              }
    jle    .L6                     }
    subl   %eax, %edx              } Body2a
    movl   %edx, %eax              }
    jmp .L7                        }
.L6:
    subl %edx, %eax                } Body2b
.L7:
    popl %ebp                      } Finish
    ret                            }
```


Supplied by CMU.

Conditional-Branch Example (Cont.)

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

```
absdiff:
    pushl  %ebp                    } Setup
    movl   %esp, %ebp              }
    movl   8(%ebp), %edx           } Body1
    movl   12(%ebp), %eax          }
    cmpl   %eax, %edx              }
    jle    .L6                     }
    subl   %eax, %edx              } Body2a
    movl   %edx, %eax              }
    jmp .L7                        }
.L6:
    subl %edx, %eax                } Body2b
.L7:
    popl %ebp                      } Finish
    ret                            }
```


Supplied by CMU.

Conditional-Branch Example (Cont.)

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

```
absdiff:
    pushl   %ebp                    } Setup
    movl    %esp, %ebp              }
    movl    8(%ebp), %edx           } Body1
    movl    12(%ebp), %eax          }
    cmpl    %eax, %edx              }
    jle     .L6                     }
    subl    %eax, %edx              } Body2a
    movl    %edx, %eax              }
    jmp .L7                         }
.L6:
    subl %edx, %eax                 } Body2b
.L7:                                }
    popl %ebp                       } Finish
    ret                             }
```


Supplied by CMU.

General Conditional-Expression Translation

C Code

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

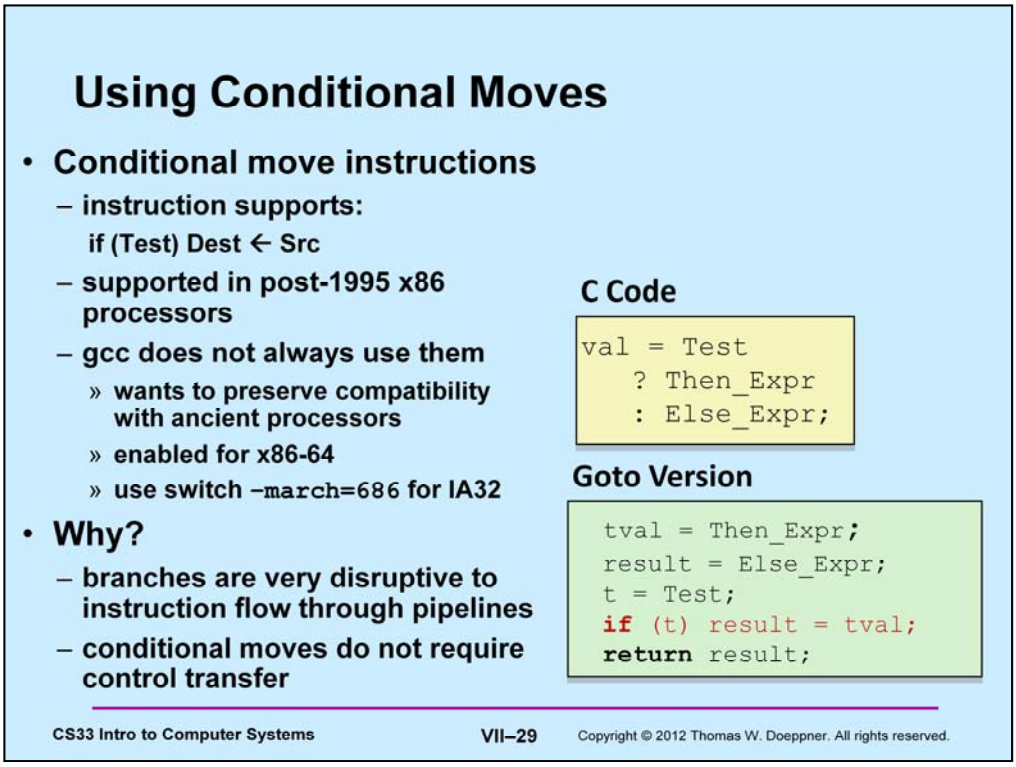
```
val = x>y ? x-y : y-x;
```

Goto Version

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

Supplied by CMU.



Supplied by CMU.

Conditional Move Example: x86-64

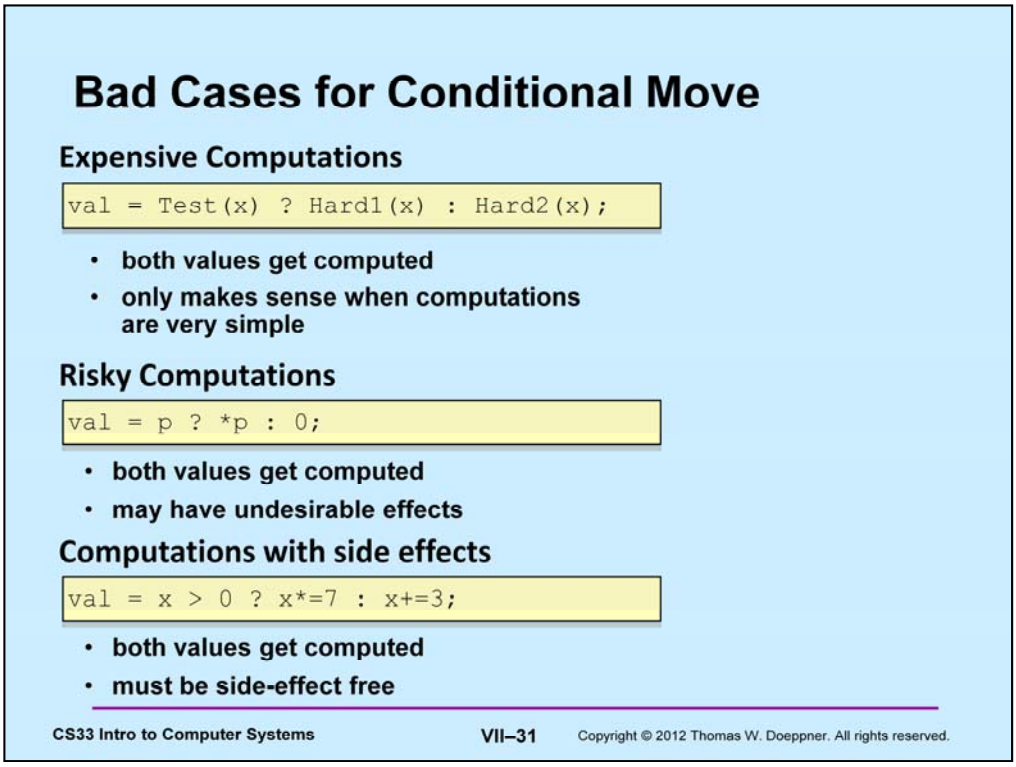
```
int absdiff(int x, int y) {
    int result;
    if (x > y) {
        result = x-y;
    } else {
        result = y-x;
    }
    return result;
}
```

x in %edi

y in %esi

```
absdiff:
  movl   %edi, %edx
  subl   %esi, %edx   # tval = x-y
  movl   %esi, %eax
  subl   %edi, %eax   # result = y-x
  cmpl   %esi, %edi   # compare x:y
  cmovg  %edx, %eax   # if >, result = tval
  ret
```


Supplied by CMU.



Supplied by CMU.