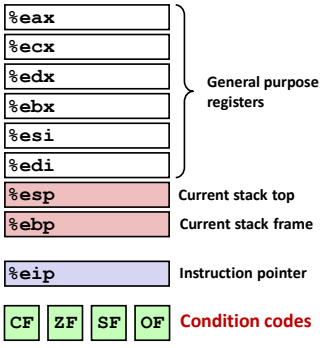
Lecture 13

Control

CPSC 275
Introduction to Computer Systems

Processor State

- Information about currently executing program
  - Temporary data (`%eax`, …)
  - Location of runtime stack (`%ebp`,`%esp`)
  - Location of current code control point (`%eip`, …)
  - Status of recent tests (**CF, ZF, SF, OF**)



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 by arithmetic operations

  Example: `addl` *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)`
- Flags are *not* set by `lea`, `inc`, or `dec` instructions.

Condition Codes (Explicit Setting)

- Explicit Setting by Compare Instruction

  `cmpl` ***src2, src1***

  (like computing *src1-src2* without setting destination)

  CF set if carry out from most significant bit (used for unsigned comparisons)

  ZF set if *src1* == *src2*

  SF set if *src1* - *src2* < 0 (as signed)

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

  (*src1* > 0 && *src2* < 0 && (*src1-src2*) < 0) ||

  (*src1* < 0 && *src2* > 0 && (*src1-src2*) > 0)

Condition Codes (Explicit Setting)

- Explicit Setting by Test instruction

  `testl` ***src2, src1***

  –Sets condition codes based on value of *src1* & *src2*

  –Like computing *src1* `&` *src2* without setting destination

  –Useful to have one of the operands be a *mask*

  ZF set when src1 & src2 == 0

  SF set when src1 & src2 < 0

  `testl %eax,%eax` (??)

  See whether `%eax` is negative, zero, or positive.

Jumping

- *j*x Instructions
  - Jump to different part of code depending on condition codes

*j*x	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

C Code

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

    return result;
}
```

Goto Version

```
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

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                             }
```

Note that `%eax` contains the return value.

"Do-While" Loop Example

C Code

```
int pcount_do(unsigned x)
{
  int result = 0;
  do {
    result += x & 0x1;
    x >>= 1;
  } while (x);
  return result;
}
```

Goto Version

```
int pcount_do(unsigned x)
{
  int result = 0;
loop:
  result += x & 0x1;
  x >>= 1;
  if (x)
    goto loop;
  return result;
}
```

"Do-While" Loop Compilation

Goto Version

```
int pcount_do(unsigned x) {
  int result = 0;
loop:
  result += x & 0x1;
  x >>= 1;
  if (x)
    goto loop;
  return result;
}
```

- Registers:
  - %edx x
  - %ecx result

```
    movl   $0, %ecx      #   result = 0
.L2:                     # loop:
    movl   %edx, %eax
    andl   $1, %eax      #   t = x & 1
    addl   %eax, %ecx    #   result += t
    shrl   %edx          #   x >>= 1
    jne    .L2           #   If !0, goto loop
```

"While" Loop Example

C Code

```
int pcount_while(unsigned x) {
  int result = 0;
  while (x) {
    result += x & 0x1;
    x >>= 1;
  }
  return result;
}
```

Goto Version

```
int pcount_do(unsigned x) {
  int result = 0;
  if (!x) goto done;
loop:
  result += x & 0x1;
  x >>= 1;
  if (x)
    goto loop;
done:
  return result;
}
```

General "While" Translation

While version

```
while (Test)
  Body
```

Do-While Version

```
  if (!Test)
    goto done;
  do
    Body
    while(Test);
done:
```

Goto Version

```
  if (!Test)
    goto done;
loop:
  Body
  if (Test)
    goto loop;
done:
```

Practice Problems

- Read CSaPP Sec. 3.6.1-3.6.5 and try the following problems:

  3.16, 3.17, 3.18, 3.22, 3.23