Announcement

- Assignment 3
  - Due Monday, October 20
  - Simulating an Accumulator Machine – Part II
  - Assignment 2 to be returned on Friday.

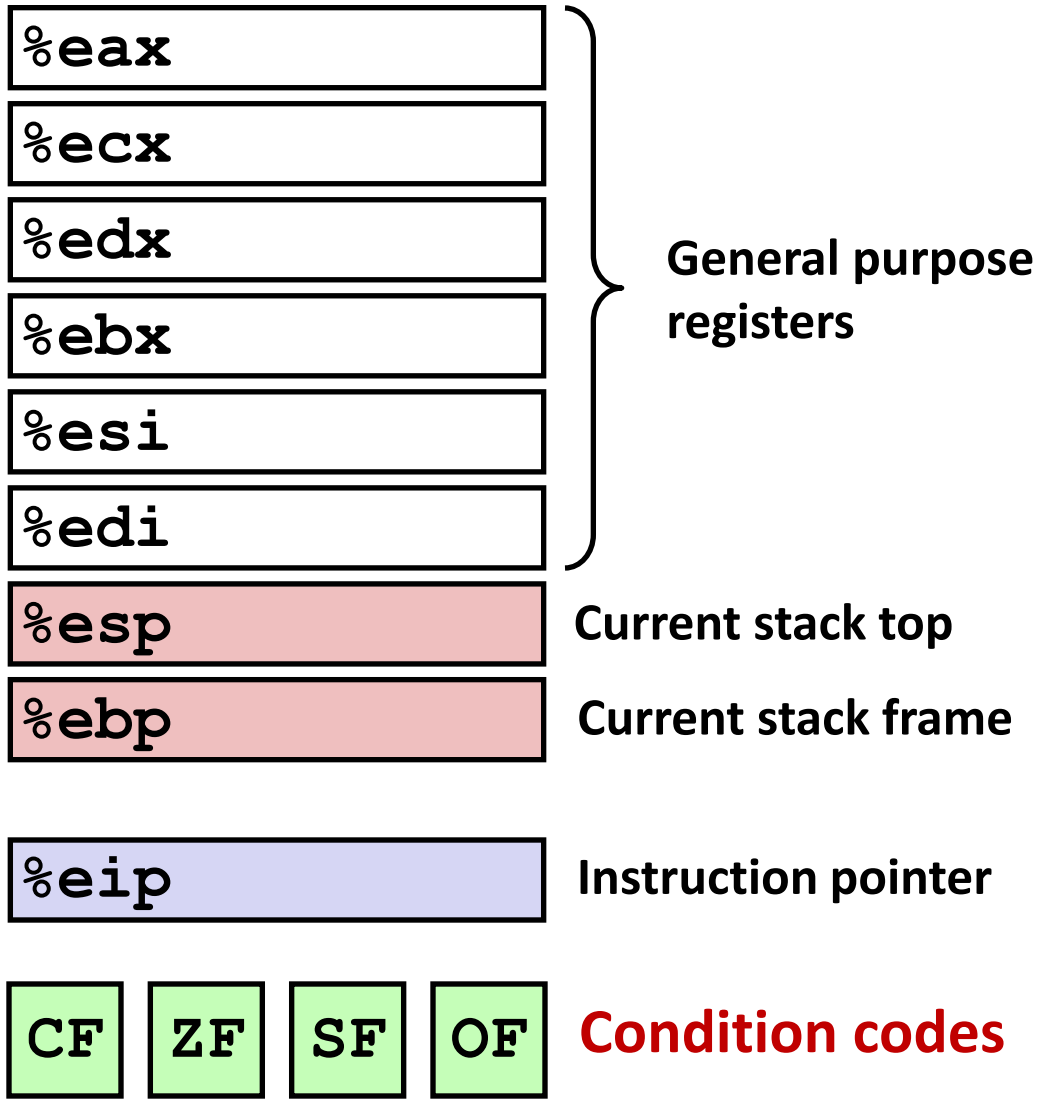
Lecture 16

Control: Selection

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**)



Control Flow

- Programs normally execute code *sequentially*.
- Program constructs such as if statements, loops, and switches require *conditional execution*.
- At the machine level, conditional behavior is achieved through *tests* and *changes* in control or data flow.
- *Data-dependent control flow* determines which instructions execute based on test results.
- *Jump* instructions can alter the normal execution order, either conditionally or unconditionally.
- *Compilers* translate high-level control structures into low-level *tests* and *jumps* that implement program logic.

Condition Codes (Implicit Setting)

- Single-bit registers

  CF Carry Flag (unsigned) **SF** Sign Flag

  ZF Zero Flag **OF** Overflow Flag (signed)

- Implicitly set by arithmetic operations

  e.g. **`addl`** *src,dest* ↔ **`t = a+b`**

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

  ZF set if `t == 0`

  SF set if `t < 0` (signed)

  OF set if two's-complement (signed) overflow, i.e.,

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

Condition Codes (Explicit Setting)

- Explicit setting by comparison instruction

  `cmpl` *src2*, *src1*

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

  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 the 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          # What does this instruction do?
```

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

Jumping

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

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

Conditional Branch Example

C Code

```
int diff(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;
}
```

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;
}
```

%edx	x
%eax	y

```
diff:
    pushl   %ebp                    } Setup
    movl    %esp,%ebp               }
    movl    8(%ebp),%edx
    movl    12(%ebp),%eax
    cmpl    %eax,%edx               } Body1
    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.