Control

if-else

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)

General purpose registers

Current stack top

Current stack frame

Instruction pointer

Condition codes

Condition Codes (Implicit Setting)

- Single bit registers
  - CF Carry Flag (unsigned)
  - ZF Zero Flag
  - SF Sign Flag
  - OF Overflow Flag (signed)
- Implicitly set by arithmetic operations
  - Example: $addl srcl,src2 \Rightarrow t = a+b$
  - $CF$ set if carry out from most significant bit (unsigned)
  - $ZF$ set if $t == 0$
  - $SF$ set if $t < 0$ (as signed)
  - $OF$ set if two's-complement (signed) overflow, i.e.,
    $(a>0 && b>0 && t<0) \lor (a<0 && b<0 && t>0)$

Condition Codes (Explicit Setting)

- Explicit setting by compare instruction
  - $cmp src2, src1$
    - (like computing $src1-src2$ without setting destination)
  - $ZF$ set if $src1 == src2$
  - $SF$ set if $src1 - src2 < 0$ (as signed)
  - $OF$ set if two's-complement (signed) overflow
    $(src1 > 0 \land src2 < 0 \land (src1-src2) < 0) \lor$
    $(src1 < 0 \land src2 > 0 \land (src1-src2) > 0)$

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 $src1$ & $src2 < 0$
  - $testl \%eax, \%eax$
    - What does this instruction do?
    - See whether $\%eax$ is negative, zero, or positive.

Jumping

- $jx$ instructions
  - Jump to different part of code depending on condition codes

<table>
<thead>
<tr>
<th>J</th>
<th>Description</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>jmp</td>
<td>Unconditional</td>
<td>1</td>
</tr>
<tr>
<td>je</td>
<td>Equal / Zero</td>
<td>ZF</td>
</tr>
<tr>
<td>jne</td>
<td>Not Equal / Not Zero</td>
<td>~ZF</td>
</tr>
<tr>
<td>js</td>
<td>Negative</td>
<td>OF</td>
</tr>
<tr>
<td>jns</td>
<td>Nonnegative</td>
<td>~OF</td>
</tr>
<tr>
<td>jg</td>
<td>Greater (Signed)</td>
<td>~SF^OF</td>
</tr>
<tr>
<td>jge</td>
<td>Greater or Equal (Signed)</td>
<td>~SF^OF1</td>
</tr>
<tr>
<td>jl</td>
<td>Less (Signed)</td>
<td>SF^OF</td>
</tr>
<tr>
<td>jle</td>
<td>Less or Equal (Signed)</td>
<td>SF^OF</td>
</tr>
<tr>
<td>ja</td>
<td>Above (unsigned)</td>
<td>~CF &amp; -ZF</td>
</tr>
<tr>
<td>jae</td>
<td>Above or Equal (unsigned)</td>
<td>~CF &amp; SF</td>
</tr>
<tr>
<td>jb</td>
<td>Below (unsigned)</td>
<td>CF</td>
</tr>
</tbody>
</table>
Conditional Branch Example

C Code

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

Goto Version

```c
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;
}
```

Goto Version (Cont.)

```asm
absdiff:
    pushl %ebp
    movl %esp, %ebp
    movl 8(%ebp), %edx
    movl 12(%ebp), %eax
    cmpl %eax, %edx
    jle .L6
    subl %edx, %eax
    jmp .L7
    .L6:
    subl %edx, %eax
    .L7:
    popl %ebp
    ret
```

Note that %eax contains the return value.