Jump instructions (IA32)

- Help to implement instruction sequences that implement the control constructs of C
- Alters sequential execution
- Control passes to different location
  - Other than next instruction
  - Based on some test

Single-bit condition code registers
- Describes the attributes of the most recent arithmetic or logical operation
- Tested and used to perform conditional branches
- The most useful condition codes are:
  - CF: Carry Flag – carry out; detects overflow for unsigned ops
  - ZF: Zero flag – the most recent operation yielded zero
  - SF: Sign Flag – the most recent op yielded a negative value
  - OF: Overflow Flag – 2’s comp operation overflowed (pos or neg)
- The flags are SET TO 1 when true
EXAMPLE: \( t = a + b \) (pg 60 for the ADD)

- \( a = 1011 + b = 1000 = 10011 \)

- **CF** set when: // unsigned overflow
  - unsigned \( t < \) unsigned \( a \)
  - reminder: only positive values
  - carry-out == 1

- **ZF** set when: \( t == 0 \) // zero

- **SF** set when: \( t < 0 \) // negative

- **OF** set when: // signed overflow
  - \((a<0 == b<0) && (t<0 != a<0)\)
  - \( (a<0 && b<0 && t>=0) || (a>0 && b>0 && t<0)\)
Technically...

- **Arithmetic and logical operators set the EFLAGS**

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Effect</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>leal</td>
<td>S, D</td>
<td>&amp;S --&gt; D</td>
</tr>
<tr>
<td>INC</td>
<td>D</td>
<td>D+1 --&gt; D</td>
</tr>
<tr>
<td>DEC</td>
<td>D</td>
<td>D-1 --&gt; D</td>
</tr>
<tr>
<td>NEG</td>
<td>D</td>
<td>-D --&gt; D</td>
</tr>
<tr>
<td>NOT</td>
<td>D</td>
<td>~D --&gt; D</td>
</tr>
<tr>
<td>ADD</td>
<td>S, D</td>
<td>D + S --&gt; D</td>
</tr>
<tr>
<td>SUB</td>
<td>S, D</td>
<td>D - S --&gt; D</td>
</tr>
<tr>
<td>IMUL</td>
<td>S, D</td>
<td>D * S --&gt; D</td>
</tr>
<tr>
<td>XOR</td>
<td>S, D</td>
<td>D ^ S --&gt; D</td>
</tr>
<tr>
<td>OR</td>
<td>S, D</td>
<td>D</td>
</tr>
<tr>
<td>AND</td>
<td>S, D</td>
<td>D &amp; S --&gt; D</td>
</tr>
<tr>
<td>SAL</td>
<td>k, D</td>
<td>D &lt;&lt; k --&gt; D</td>
</tr>
<tr>
<td>SHL</td>
<td>k, D</td>
<td>D &lt;&lt; k --&gt; D</td>
</tr>
<tr>
<td>SAR</td>
<td>k, D</td>
<td>D &gt;&gt; k --&gt; A</td>
</tr>
<tr>
<td>SHR</td>
<td>k, D</td>
<td>D &gt;&gt; k --&gt; L</td>
</tr>
</tbody>
</table>

Leal does not alter any condition codes (since intended use is address computations – pg. 420)

Logical operations carry and overflow flags are set to 0 (ex. XOR pg. 845)

Shift operations, the carry flags is set to the last bit shifted out; the overflow flag is set to 0 (pg. 741)

INC/DEC set overflow and zero flags; and leave carry flag unchanged.

* Check ISA manual
Compare instruction

- These instructions set the condition codes without updating any other registers.
- CMPx S1, S2 → S2−S1
  - The x can be a b, w or l for byte, word or long.
- CMP acts like the SUB without updating the destination.
  - ZF set if a == b
  - SF set if (a−b) < 0
  - CF set if carry out from MSB = 1
  - OF set if 2’s comp overflow
    - (a>0 && b<0 && (a−b)<0 || (a<0 && b>0 && (a−b)>0)

See Handout
The TEST operation sets the flags CF and OF to zero. The SF is set to the **MSB** of the result of the **AND**. If the result of the **AND** is 0, the ZF is set to 1, otherwise set to 0.

TEST acts like the AND without updating the destination...

```
testx s1, s2 → s1 & s2

- ZF set when a&b == 0
- SF set when a&b < 0
- OF/CF are set to 0 (not used)

Example: same operand repeated to see whether the operand is negative, zero or positive
  - testl %eax, %eax
    - sets ZF to 1 if %eax == 0
    - sets SF to 1 if %eax < 0 (i.e. negative) and 0 if %eax > 0 (i.e. positive)

One of the operands is a mask indicating which bits should be tested
  - testl 0xFF, %eax
```