## Special Arithmetic operations

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>imull</td>
<td>signed full multiply</td>
<td>S x R[%eax] --&gt; R[%edx]:R[eax]</td>
</tr>
<tr>
<td>mull</td>
<td>unsigned full multiply</td>
<td>S x R[%eax] --&gt; R[%edx]:R[eax]</td>
</tr>
<tr>
<td>idivl</td>
<td>signed divide</td>
<td>R[%edx]:R[eax] mod S --&gt; R[%edx]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R[%edx]:R[eax] ÷ S --&gt; R[%eax]</td>
</tr>
<tr>
<td>divl</td>
<td>unsigned divide</td>
<td>R[%edx]:R[eax] mod S --&gt; R[%edx]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R[%edx]:R[eax] ÷ S --&gt; R[%eax]</td>
</tr>
</tbody>
</table>

- Provide full 64-bit multiplication and division for both signed and unsigned numbers.
- The pair of registers %edx and %eax are viewed as forming a single 64-bit “quad word”.
- Remember that a word in IA32 is 16 bits (*4 = 64)
Special Arithmetic examples

// x at %ebp+8 and y at %ebp+12
movl 12(%ebp), %eax // put y in %eax
imull 8(%ebp) // multiply by x
movl %eax, (%esp) // store low-order 32 bits
movl %edx, 4(%esp) // store high-order 32 bits

Works for little-endian machine (i.e. low order bits at the top of the stack)

// x at %ebp+8 and y at %ebp+12
movl 8(%ebp), %edx // put x in %edx
movl %edx, %eax // copy x to %eax
sarl $31, %edx // sign extend x in %edx
idivl 12(%ebp) // divide by y
movl %eax, 4(%esp) // store x/y
movl %edx, (%esp) // store x % y
Control structures (in C)

Machine code provides two basic low-level mechanisms for implementing conditional behavior, tests data values then either
- Alters the control flow (conditional statement)
- Alters the data flow (conditional expression)

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

```c
int absdiff(int x, int y) {
    return x < y ? y - x : x-y;
}
```

```c
int gotodiff(int x, int y) {
    int result;
    if (x >= y) goto x_ge_y;
    result = y - x;
    goto done;
    x_ge_y:  result = x - y;
    done:    return result;
}
```

```c
int cmovdiff(int x, int y) {
    int tval = y-x;
    int rval = x-y;
    int test = x < y;
    if (test) rval = tval;
    return rval;
}
```
Using the JMP instruction, we may create a simple infinite loop that counts up from zero using the %eax register:

```assembly
MOVL $0, %eax
loop:    INCL %eax
        JMP loop
// unconditional jump
```

Loop to count %eax from 0 to 5:

```assembly
MOVL $0, %eax
loop:    INCL %eax
        CMPL $5, %eax
        JLE loop
// conditional jump
//if %eax <= 5 then go to loop
```

- The jmp label instruction causes the processor to execute the next instruction at the location given by the label (i.e., the %eip is set to label).
- Conditional jump instructions will only transfer control if to the target of the appropriate flags are set.