Understanding arith

```c
long arith (long x, long y, long z) {
    long t1 = x+y;
    long t2 = z+t1;
    long t3 = x+4;
    long t4 = y * 48;
    long t5 = t3 + t4;
    long rval = t2 * t5;
    return rval;
}
```

```
arith:
leaq (%rdi,%rsi), %rax # t1
addq %rdx, %rax # t2
leaq (%rsi,%rsi,2), %rdx
salq $4, %rdx # t4
leaq 4(%rdi,%rdx), %rcx # t5
imulq %rcx, %rax # rval
ret
```

<table>
<thead>
<tr>
<th>Register</th>
<th>Use(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>%rdi</td>
<td>Argument x</td>
</tr>
<tr>
<td>%rsi</td>
<td>Argument y</td>
</tr>
<tr>
<td>%rdx</td>
<td>Argument z</td>
</tr>
<tr>
<td>%rax</td>
<td>t1, t2, rval</td>
</tr>
<tr>
<td>%rdx</td>
<td>t4</td>
</tr>
<tr>
<td>%rcx</td>
<td>t5</td>
</tr>
</tbody>
</table>

Condition Codes (Explicit Setting: Compare)

- **Single-bit registers**
  - **CF** Carry Flag (for unsigned)
  - **ZF** Zero Flag
  - **SF** Sign Flag (for signed)
  - **OF** Overflow Flag (for signed)

- **Explicit Setting by Compare Instruction**
  - `cmpq %rdx,%rsi`
  - `cmpq %rdx,%rsi`
    - like computing `a-b` without setting destination

- **CF set** if carry out from most significant bit (used for unsigned comparisons)
- **ZF set** if `a == b`
- **SF set** if `(a-b) < 0` (as signed)
- **OF set** if two's complement (signed) overflow
  - `(a>0 && b<0 && (a-b)<0) || (a<0 && b>0 && (a-b)>0)`
Condition Codes (Explicit Setting: Test)

- Single-bit registers
  - CF  Carry Flag (for unsigned)
  - SF  Sign Flag (for signed)
  - ZF  Zero Flag
  - OF  Overflow Flag (for signed)

- Explicit Setting by Test instruction
  - `testq`Src2,Src1
  - `testq b, a` like computing `a & b` without setting destination
  - Sets condition codes based on value of `Src1 & Src2`
  - Useful to have one of the operands be a mask
  - **ZF** set if `a&b == 0`
  - **SF** set if `a&b < 0`

- `testq %rax, %rax`
  - Sets SF and ZF, check if rax is +,0,-

Reading Condition Codes (Cont.)

- **SetX Instructions:**
  - Set single byte to 0 or 1 based on combination of condition codes
  - Operand is one of the addressable byte registers (eg. `al`, `dl`)

- **SetX instruction does not alter remaining bytes in register**
  - Typically use `movzbl` to finish job
  - Sets upper 32 bits to zero
  - Aside: Other 32-bit instructions also set upper 32 bits to zero

```c
int gt (long x, long y) {
    return x > y;
}
```

<table>
<thead>
<tr>
<th>Register</th>
<th>Use(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>%rdi</td>
<td>Argument x</td>
</tr>
<tr>
<td>%rsi</td>
<td>Argument y</td>
</tr>
<tr>
<td>%rax</td>
<td>Return value</td>
</tr>
</tbody>
</table>

```c
cmpq %rsi, %rdi  # Compare x:y
setg %al        # al = x > y
movzbl %al, %eax # Zero rest of %rax
ret
```