# General Memory Addressing

- Most General Form

**D(Rb,Ri,S)** **Mem[D + Reg[Rb]+S*Reg[Ri]]**

- D: Constant "displacement"
- Rb: Base register
- Ri: Index register
- S: Scale: 1, 2, 4, or 8

- Special cases

| | |
|---|---|
| (Rb,Ri) | Mem[Reg[Rb]+Reg[Ri]] |
| D(Rb,Ri) | Mem[D + Reg[Rb]+Reg[Ri]] |
| (Rb,Ri,S) | Mem[Reg[Rb]+S*Reg[Ri]] |
| (,Ri,S) | Mem[S*Reg[Ri]] |
| D(,Ri,S) | Mem[D + S*Reg[Ri]] |

# Arithmetic Operations

| | | | |
|---|---|---|---|
| `leaq` | *Src,Dest* | Dest = Src-Expr | ***No memory access!*** |
| `addq` | *Src,Dest* | Dest = Dest + Src | |
| `subq` | *Src,Dest* | Dest = Dest - Src | |
| `imulq` | *Src,Dest* | Dest = Dest * Src | |
| `sarq` | *Src,Dest* | Dest = Dest >> Src | ***Arithmetic*** |
| `shrq` | *Src,Dest* | Dest = Dest >> Src | ***Logical*** |
| `salq` | *Src,Dest* | Dest = Dest << Src | ***Also called shlq*** |
| `xorq` | *Src,Dest* | Dest = Dest ^ Src | |
| `andq` | *Src,Dest* | Dest = Dest & Src | |
| `orq` | *Src,Dest* | Dest = Dest \| Src | |
| `incq` | *Dest* | *Dest = Dest + 1* | |
| `decq` | *Dest* | *Dest = Dest - 1* | |
| `negq` | *Dest* | *Dest = -Dest* | |
| `notq` | *Dest* | *Dest = ~Dest* | |

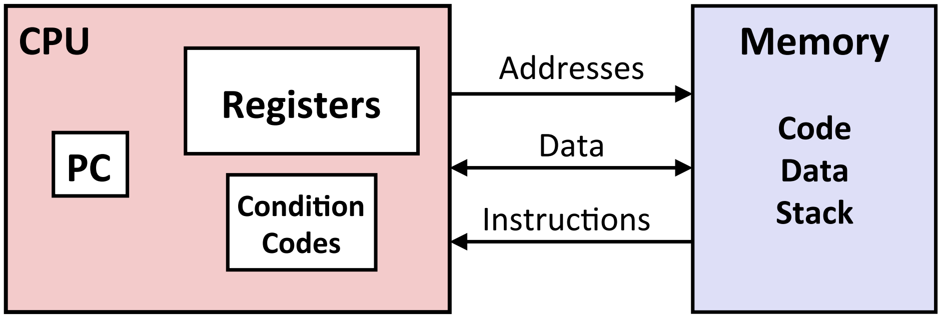
# Arithmetic Example

```
(x,y,z) -> (%rdi,%rsi,%rdx)
```

```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
  addq    %rdx, %rax
  leaq    (%rsi,%rsi,2), %rdx
  salq    $4, %rdx
  leaq    4(%rdi,%rdx), %rcx
  imulq   %rcx, %rax
  ret
```

# Assembly View of the Machine



# Condition Codes

CF ZF SF OF Condition codes

CF: Carry flag (for unsigned)

ZF: Zero flag

SF: Sign flag (for signed)

OF: Overflow flag (for signed)

# Reading Condition Codes

| SetX | Condition | Description |
|---|---|---|
| `sete` | `ZF` | Equal / Zero |
| `setne` | `~ZF` | Not Equal / Not Zero |
| `sets` | `SF` | Negative |
| `setns` | `~SF` | Nonnegative |
| `setg` | `~(SF^OF)&~ZF` | Greater (Signed) |
| `setge` | `~(SF^OF)` | Greater or Equal (Signed) |
| `setl` | `(SF^OF)` | Less (Signed) |
| `setle` | `(SF^OF)\|ZF` | Less or Equal (Signed) |
| `seta` | `~CF&~ZF` | Above (unsigned) |
| `setb` | `CF` | Below (unsigned) |

# Single-Byte Virtual Registers

| | | | |
|---|---|---|---|
| %rax | %al | %r8 | %r8b |
| %rbx | %bl | %r9 | %r9b |
| %rcx | %cl | %r10 | %r10b |
| %rdx | %dl | %r11 | %r11b |
| %rsi | %sil | %r12 | %r12b |
| %rdi | %dil | %r13 | %r13b |
| %rsp | %spl | %r14 | %r14b |
| %rbp | %bpl | %r15 | %r15b |

# Example: Greater Than

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

| Register | Use(s) |
|---|---|
| **%rdi** | Argument **x** |
| **%rsi** | Argument **y** |
| **%rax** | Return value |

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

# Goto

```
#include <stdio.h>

int main() {

  /* local variable definition */
  int a = 10;

  /* do loop execution */
  LOOP:do {

       if (a == 15) {
         /* skip the iteration */
         a = a + 1;
         goto LOOP;
       }

       printf("value of a: %d\n", a);
       a++;

     } while (a < 20);

     return 0;
}
```

# Jumping

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

# Example: absdiff

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

```
absdiff:
   cmpq    %rsi, %rdi  # x:y
   jle     .L4
   movq    %rdi, %rax
   subq    %rsi, %rax
   ret
.L4:       # x <= y
   movq    %rsi, %rax
   subq    %rdi, %rax
   ret
```

| Register | Use(s) |
|---|---|
| %rdi | Argument **x** |
| %rsi | Argument **y** |
| %rax | Return value |

# absdiff with Goto

```
absdiff:
   cmpq    %rsi, %rdi  # x:y
   jle     .L4
   movq    %rdi, %rax
   subq    %rsi, %rax
   ret
.L4:       # x <= y
   movq    %rsi, %rax
   subq    %rdi, %rax
   ret
```

```
long absdiff_j
  (long x, long y)
{
    long result;
    int ntest = x <= y;
    if (ntest) goto Else;
    result = x-y;
    goto Done;
 Else:
    result = y-x;
 Done:
    return result;
}
```