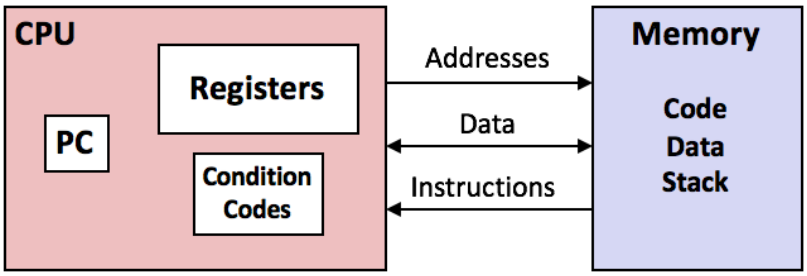
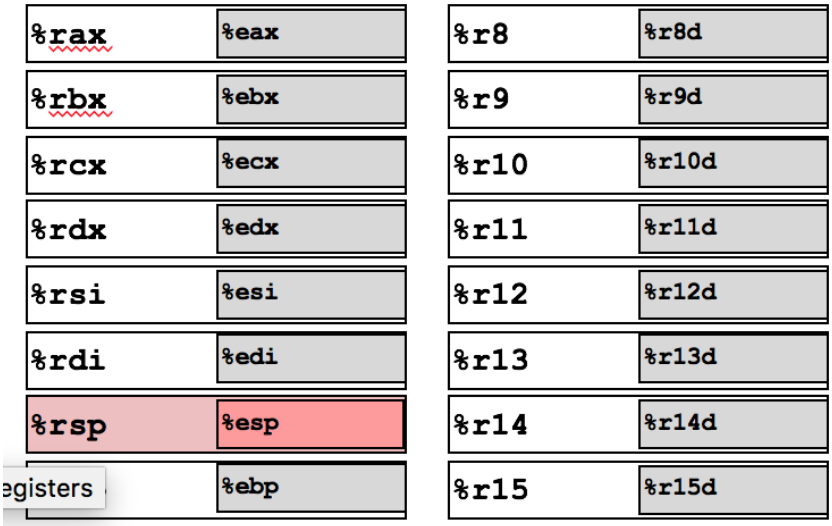
Machine Prog: Basic & Control

1、 machine — basics

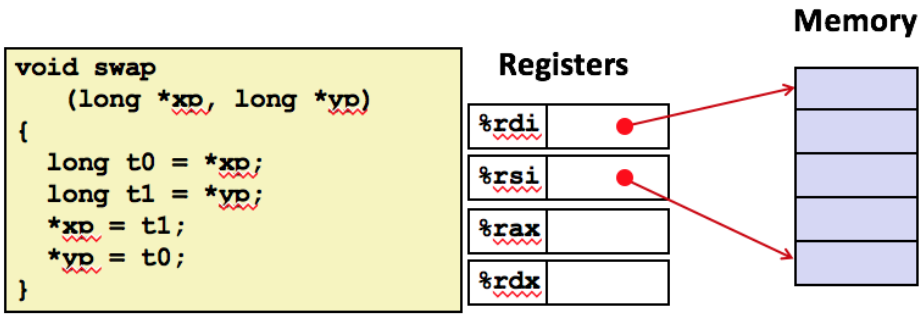
- Assembly Basics: Registers, operands, move



- x86-64 Integer Registers



example:

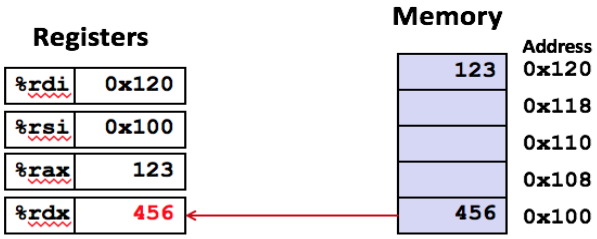



```
void swap
    (long *xp, long *yp)
{
  long t0 = *xp;
  long t1 = *yp;
  *xp = t1;
  *yp = t0;
}
```

Register	Value
%rdi	xp
%rsi	yp
%rax	t0
%rdx	t1

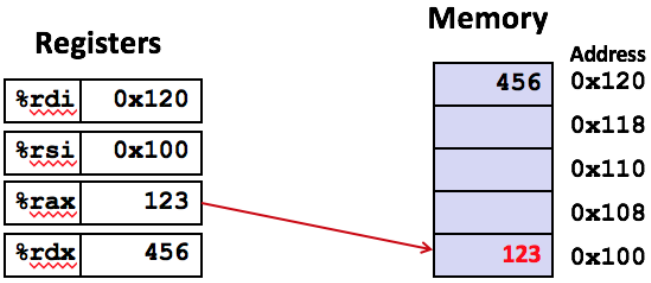
```
swap:
    movq    (%rdi), %rax  # t0 = *xp
    movq    (%rsi), %rdx  # t1 = *yp
    movq    %rdx, (%rdi)  # *xp = t1
    movq    %rax, (%rsi)  # *yp = t0
    ret
```

是一个 mem 到 res 的过程，下面是第二步的过程，第一步同理




```
swap:
    movq    (%rdi), %rax  # t0 = *xp
    movq    (%rsi), %rdx  # t1 = *yp
    movq    %rdx, (%rdi)  # *xp = t1
    movq    %rax, (%rsi)  # *yp = t0
    ret
```

第二部分是 res 到 mem，下面是第二步，第一步同理




```
swap:
    movq    (%rdi), %rax  # t0 = *xp
    movq    (%rsi), %rdx  # t1 = *yp
    movq    %rdx, (%rdi)  # *xp = t1
    movq    %rax, (%rsi)  # *yp = t0
    ret
```

- Arithmetic & logical operations

  ■ **Two Operand Instructions:**

Format	*Computation*		
addq	*Src,Dest*	Dest = Dest + Src	
subq	*Src,Dest*	Dest = Dest – Src	
imulq	*Src,Dest*	Dest = Dest * Src	
salq	*Src,Dest*	Dest = Dest << Src	*Also called shlq*
sarq	*Src,Dest*	Dest = Dest >> Src	*Arithmetic*
shrq	*Src,Dest*	Dest = Dest >> Src	*Logical*
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*

Example

```
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
```

Register	Use(s)
%rdi	Argument x
%rsi	Argument y
%rdx	Argument z, t4
%rax	t1, t2, rval
%rcx	t5

example：

leaq: address computation

2、 machine — control

- Condition codes

Example: **addq** *Src,Dest* ↔ **t = a+b**

CF set if carry out from most significant bit (unsigned overflow)

ZF set if **t == 0**

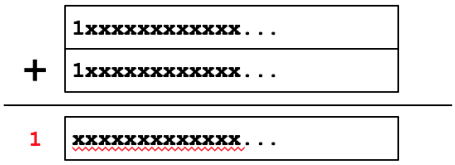
SF set if **t < 0** (as signed)

OF set if two's-complement (signed) overflow

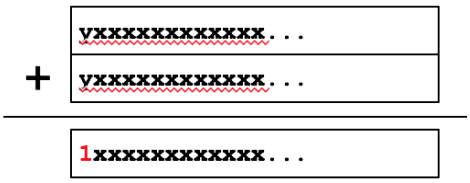
(a>0 && b>0 && t<0) || (a<0 && b<0 && t>=0)

example:

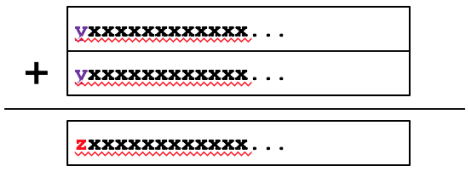
CF set when



SF set when



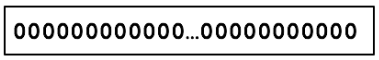
OF set when



z = ~y

ZF set when

OF set when




```
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
```

第三步：结果存入%al

第四步：前面多余补 0（%rax > %eax > %al）.

(有可能是这样，我也没太理解)

- Loops
  - Do—while（条件成立循环）

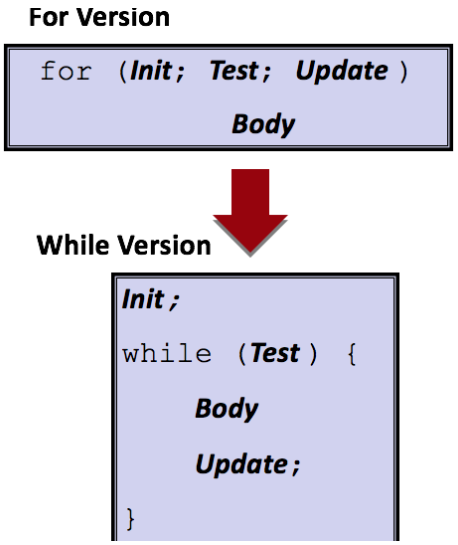
C Code

```
long pcount_do
  (unsigned long x) {
  long result = 0;
  do {
    result += x & 0x1;
    x >>= 1;
  } while (x);
  return result;
}
```

```
   movl    $0, %eax    #  result = 0
.L2:                   # loop:
   movq    %rdi, %rdx
   andl    $1, %edx    #  t = x & 0x1
   addq    %rdx, %rax  #  result += t
   shrq    %rdi        #  x >>= 1
   jne     .L2         #  if (x) goto loop
   rep; ret
```

Register	Use(s)
%rdi	Argument x
%rax	result

- For




```
for (i = 0; i < WSIZE; i++)
```

基本同理，只是写法不同。

- Switch case

```
switch(x) {
  case val_0:
    Block 0
  case val_1:
    Block 1
    • • •
  case val_n-1:
    Block n–1
  default :
     ......
}
```

Setup:

```
switch_eg:
    movq    %rdx, %rcx
    cmpq    $6, %rdi    # x:6
    ja      .L8
    jmp     *.L4(,%rdi,8)
```

Register	Use(s)
%rdi	Argument x
%rsi	Argument y
%rdx	Argument z
%rax	Return value

ja.L8 是跳到了 case0，4？？？？

Jump table

```
.section    .rodata
  .align 8
.L4:
  .quad     .L8   # x = 0
  .quad     .L3   # x = 1
  .quad     .L5   # x = 2
  .quad     .L9   # x = 3
  .quad     .L8   # x = 4
  .quad     .L7   # x = 5
  .quad     .L7   # x = 6
```

```
switch(x) {
case 1:       // .L3
    w = y*z;
    break;
case 2:       // .L5
    w = y/z;
    /* Fall Through */
case 3:       // .L9
    w += z;
    break;
case 5:
case 6:       // .L7
    w -= z;
    break;
default:      // .L8
    w = 2;
}
```

疑问：Lx 是怎么确定的？

好像是系统自己生成的？

我第一次写这么长的知识点整理笔记，如果写的不好希望大家多多包涵，谢谢。