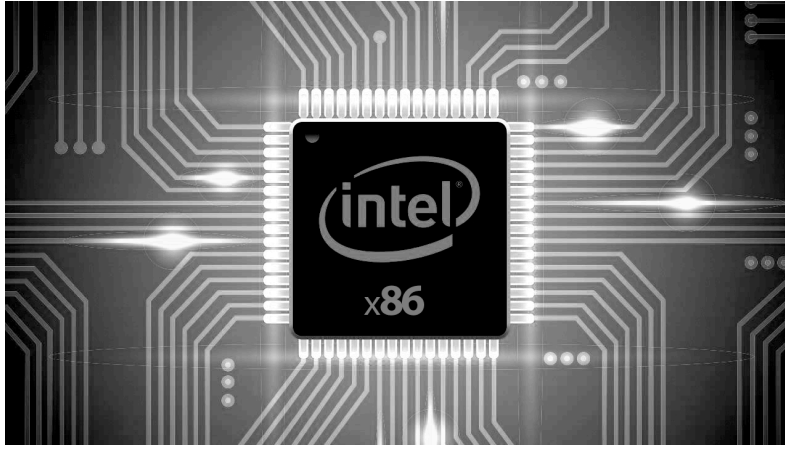
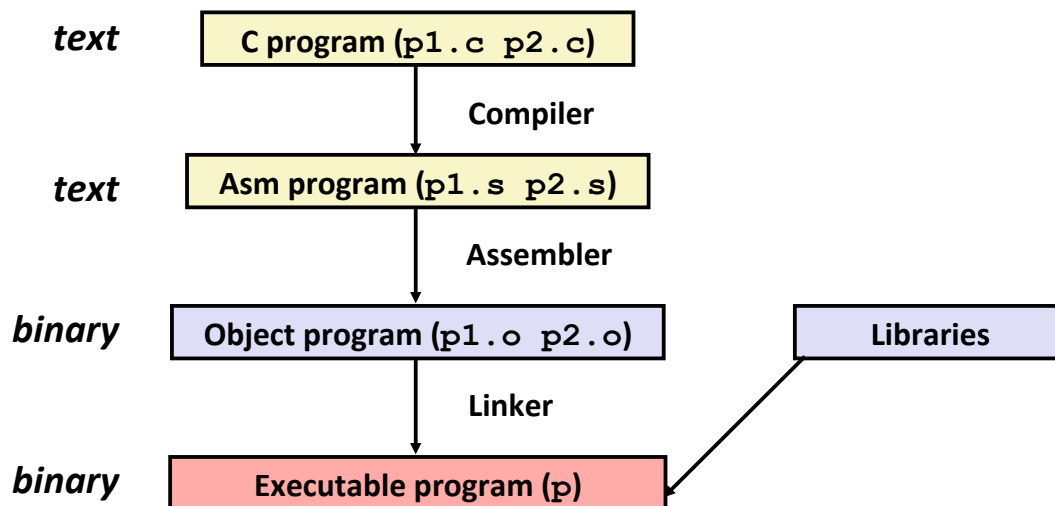


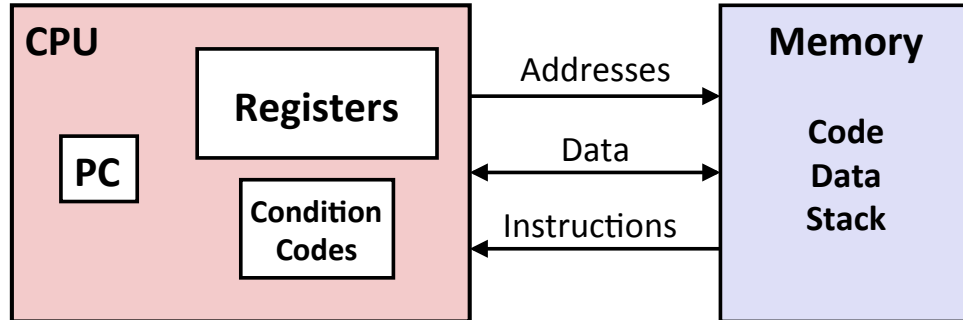
# Machine Code



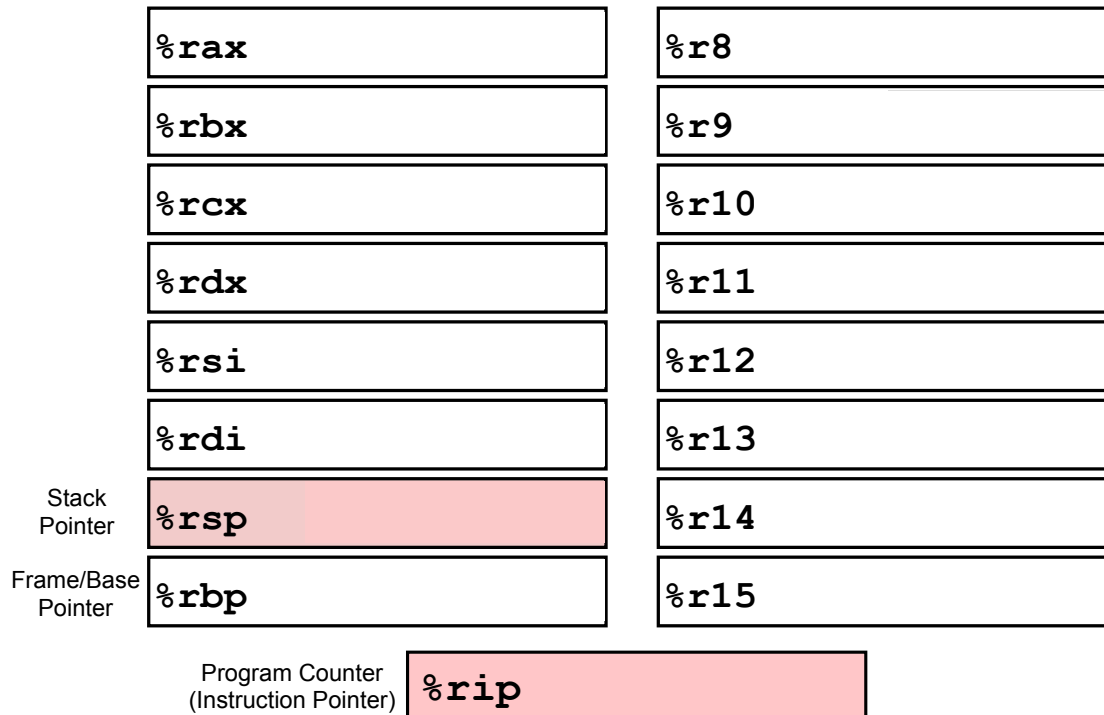
# From C to Executable Code



# Assembly View of the Machine



## x86-64 Integer Registers



# x86-64 Virtual Registers

64-Bit Register	Lowest 32 Bits	Lowest 16 Bits	Lowest 8 Bits
<code>%rax</code>	<code>%eax</code>	<code>%ax</code>	<code>%al</code>
<code>%rbx</code>	<code>%ebx</code>	<code>%bx</code>	<code>%bl</code>
<code>%rcx</code>	<code>%ecx</code>	<code>%cx</code>	<code>%cl</code>
<code>%rdx</code>	<code>%edx</code>	<code>%dx</code>	<code>%dl</code>
<code>%rsi</code>	<code>%esi</code>	<code>%si</code>	<code>%sil</code>
<code>%rdi</code>	<code>%edi</code>	<code>%di</code>	<code>%dil</code>
<code>%rbp</code>	<code>%ebp</code>	<code>%bp</code>	<code>%bpl</code>
<code>%rsp</code>	<code>%esp</code>	<code>%sp</code>	<code>%spl</code>
<code>%r8</code>	<code>%r8d</code>	<code>%r8w</code>	<code>%r8b</code>
<code>%r9</code>	<code>%r9d</code>	<code>%r9w</code>	<code>%r9b</code>
<code>%r10</code>	<code>%r10d</code>	<code>%r10w</code>	<code>%r10b</code>
<code>%r11</code>	<code>%r11d</code>	<code>%r11w</code>	<code>%r11b</code>
<code>%r12</code>	<code>%r12d</code>	<code>%r12w</code>	<code>%r12b</code>
<code>%r13</code>	<code>%r13d</code>	<code>%r13w</code>	<code>%r13b</code>
<code>%r14</code>	<code>%r14d</code>	<code>%r14w</code>	<code>%r14b</code>
<code>%r15</code>	<code>%r15d</code>	<code>%r15w</code>	<code>%r15b</code>

## Data Size Suffixes

Suffix	Size	Description
<b>b</b>	8 bits	byte
<b>w</b>	16 bits	word (historical)
<b>l</b>	32 bits	long word
<b>q</b>	64 bits	quad word

# Operand Combinations

	Source	Dest	Src, Dest	C Analog
movq	Imm	Reg	movq \$0x4,%rax	temp = 0x4;
		Mem	movq \$-147, (%rax)	*p = -147;
	Reg	Reg	movq %rax,%rdx	temp2 = temp1;
		Mem	movq %rax, (%rdx)	*p = temp;
	Mem	Reg	movq (%rax), %rdx	temp = *p;

# Data Movement Examples

“Copy K bytes from [val N/addr N/reg N] to [addr M/reg M]”

1. movq %rax, %rbx
2. movw %ax, %bx
3. movq \$5, %rcx
4. movq \$-12, (%rcx)
5. movl \$0xFF, %eax
6. movb %al, (%rbx)
7. movl 5, %eax
8. movw %ax, 30
9. movq (%rax), %rbx
10. movl (%rax), %ebx

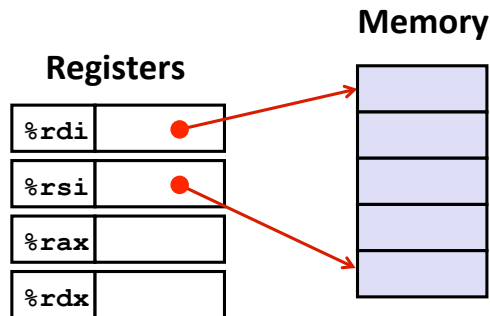
# Assembly Translation Example

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

```
swap:  
    movq    (%rdi), %rax  
    movq    (%rsi), %rdx  
    movq    %rdx, (%rdi)  
    movq    %rax, (%rsi)  
    ret
```

# Understanding Swap

```
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
```

# General Memory Addressing

## General Form:

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

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

$5(\%rax, \%rbx, 8) \quad Mem[5 + \%rax + 8 * \%rbx]$

## Special Cases:

(Rb)	(%rax)	Mem[Reg[Rb]]
D(Rb)	5(%rax)	Mem[D + Reg[Rb]]
(Rb, Ri)	(%rax, %rbx)	Mem[Reg[Rb] + Reg[Ri]]
D(Rb, Ri)	5(%rax, %rbx)	Mem[D + Reg[Rb] + Reg[Ri]]
(Rb, Ri, S)	(%rax, %rbx, 8)	Mem[Reg[Rb] + S * Reg[Ri]]
(, Ri, S)	(, %rbx, 8)	Mem[S * Reg[Ri]]
D(, Ri, S)	5(, %rbx, 8)	Mem[D + S * Reg[Ri]]

# Arithmetic Instructions

addq	Src, Dest	Dest = Dest + Src	
subq	Src, Dest	Dest = Dest - Src	
imulq	Src, Dest	Dest = Dest * Src	
sarq	Src, Dest	Dest = Dest >> Src	<b>Arithmetic RShift</b>
shrq	Src, Dest	Dest = Dest >> Src	<b>Logical RShift</b>
salq	Src, Dest	Dest = Dest << Src	<b>Also called shlq</b>
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	

$leaq \quad Src, Dest \quad Dest = Src \text{ (as expr)} \quad \text{No memory access!}$

# Arithmetic Example

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

```
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:

```
1. leaq    (%rdi,%rsi), %rax
2. addq    %rdx, %rax
3. leaq    (%rsi,%rsi,2), %rdx
4. salq    $4, %rdx
5. leaq    4(%rdi,%rdx), %rcx
6. imulq   %rcx, %rax
ret
```

# Procedure Call Registers



# Procedure Call Example

```
long x = add(3, 5);  
doSomething(x);
```

```
long add(long x, long y) {  
    return x + y;  
}
```

```
movq $3, %rdi  
movq $5, %rsi  
callq add  
movq %rax, %rdi  
callq doSomething
```

```
add:  
    movq %rdi, %rax  
    addq %rsi, %rax  
    ret
```