

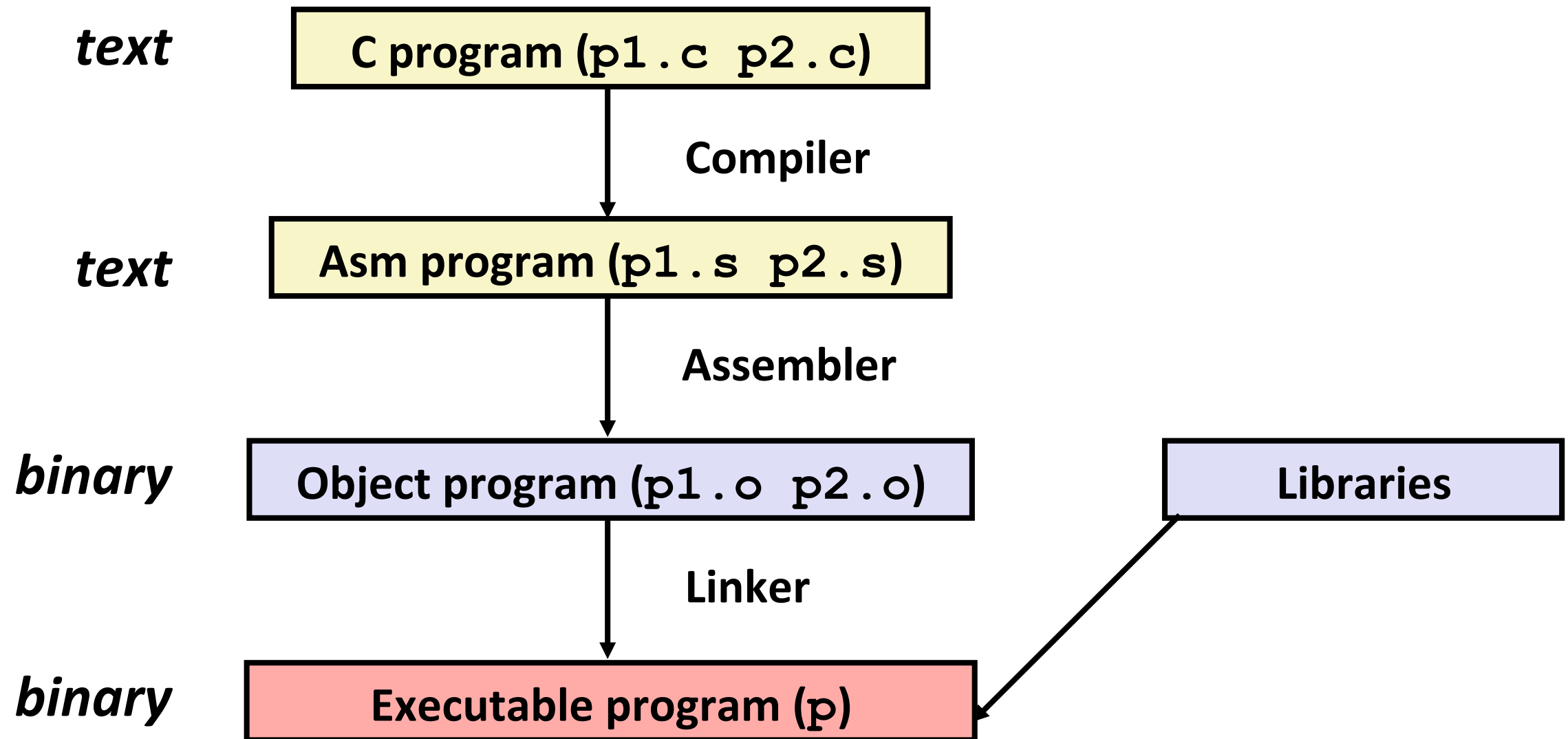
# Bowdoin

## x86-64 Basics

CSCI 2330



# From C to Executable Code



# Instruction Classes

- Data Movement (Accessing Information)
- Arithmetic and Logic Operations
- Control and Conditions
- Procedures

## MOV

Accessing Information  
(Data Movement)



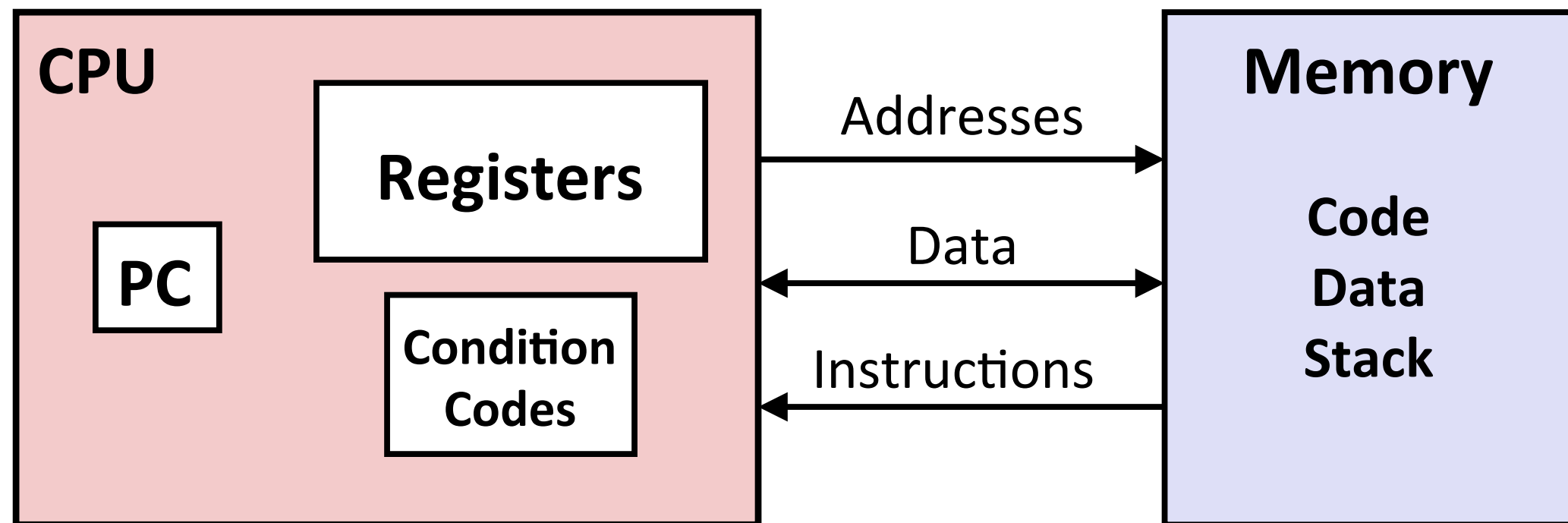
# MOV

**mov\_\_ source, destination**

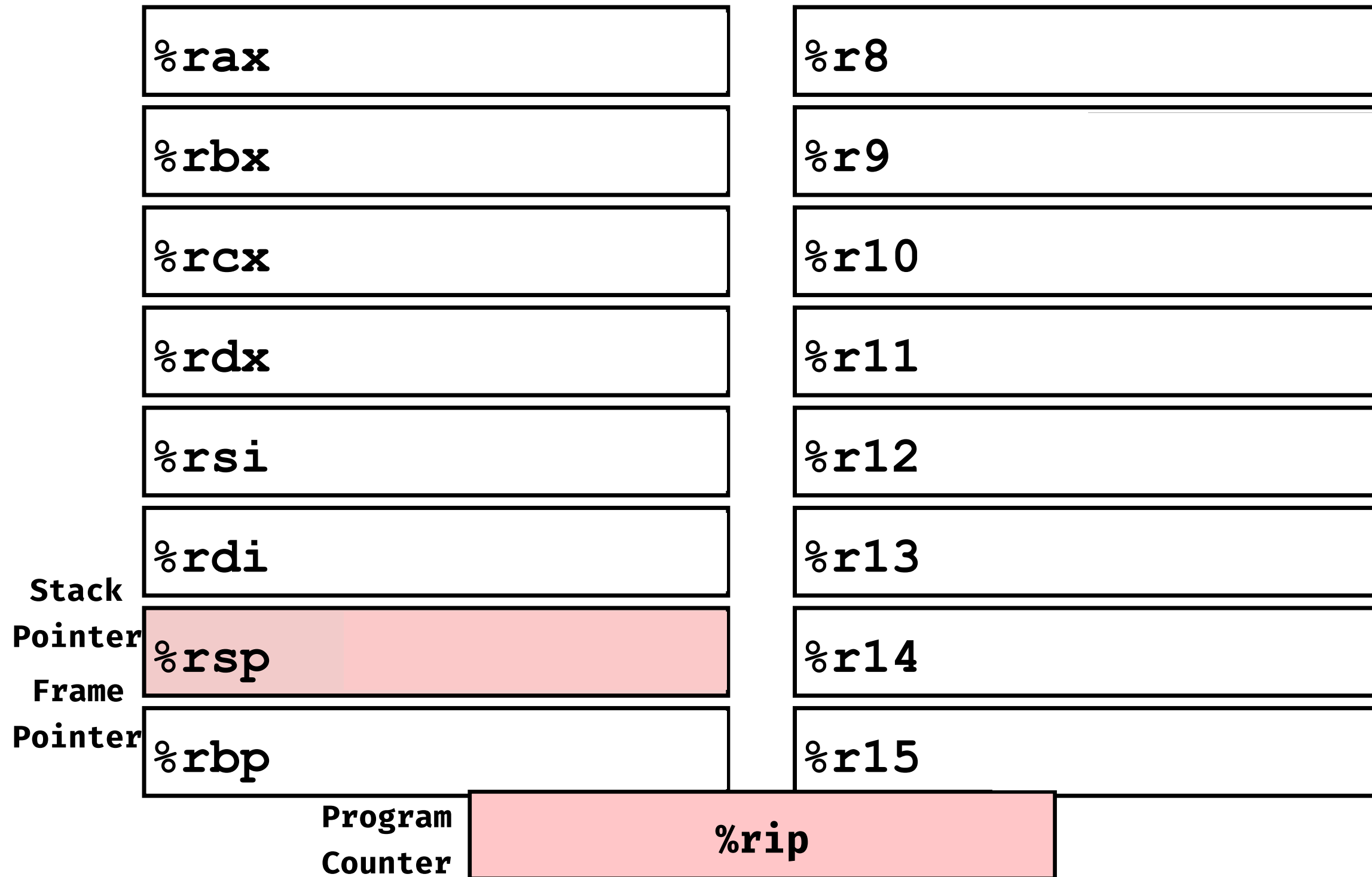
Source and destination can be..

- Immediate; `$0x03`, `$-2`, ...
- Register; `%rbp`, `%eax`, ...
- Memory; `0x500`, `-3(%rax, %rbp, 2)`, ...

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



# Data Size Suffixes

Suffix	Size	Description
b	8 bits	byte
w	16 bits	word (historical)
l	32 bits	long word
q	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;

# Exercise

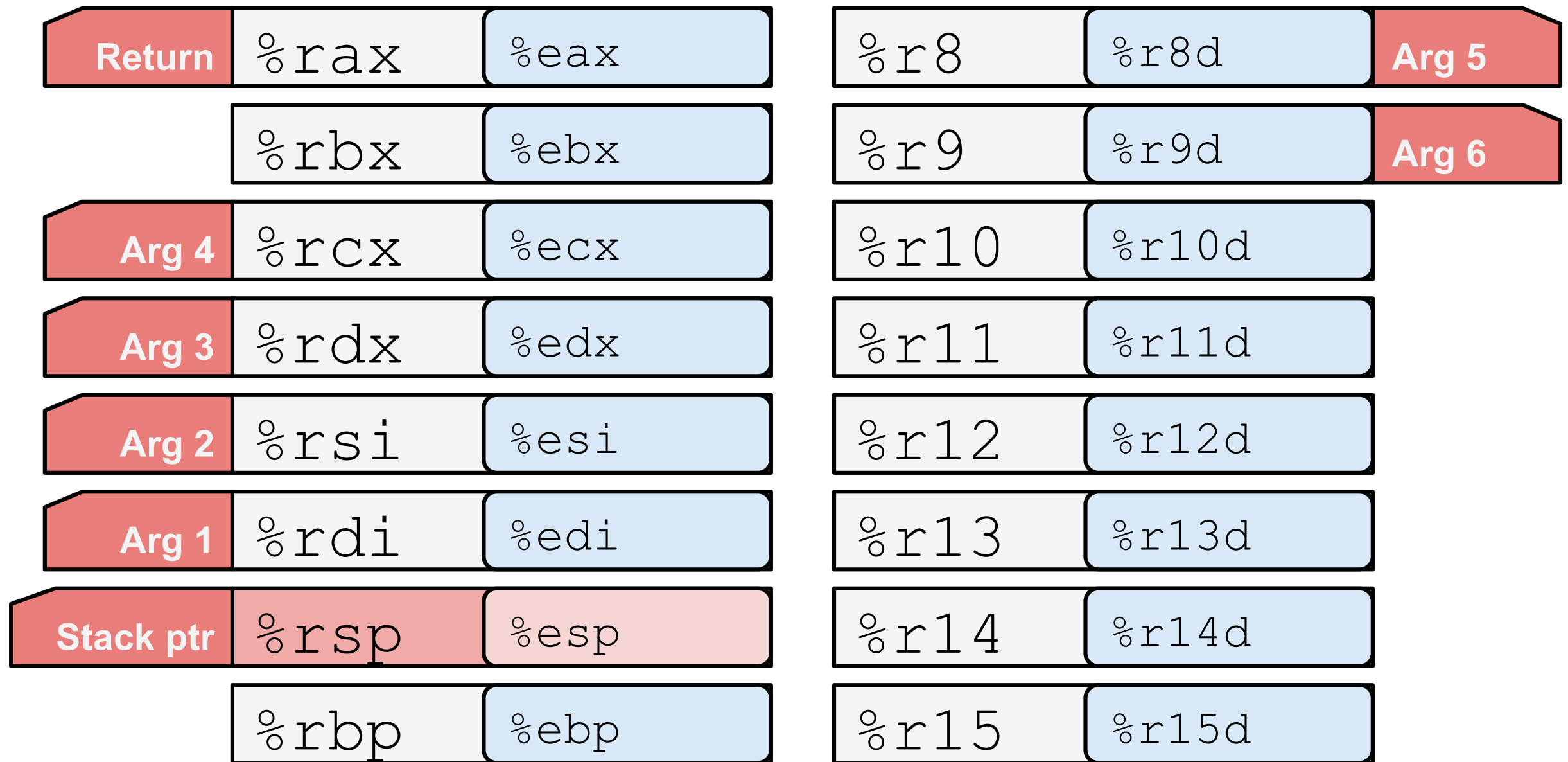
**“Copy K bytes from  
[val N/addr N/reg N] to [addr M/reg M]”**

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. `movl (%rax), %ebx`
10. `movb $1, (%rdx)`

# Some Notes about MOV

- Cannot have both operands be memory  
e.g. **mov (%rbx),(%rax)** is not allowed.
- Base and index registers must be 64-bit in  
**movxx I(%base, %index, scale)**
- **movl** src, %reg will set high order bytes to  
**0x000000**
- **movz ..** and **movs ..** extend with zero or sign  
bit. e.g. **movzbw \$0x01, %eax**

# Procedure Call Registers



# Addressing Example

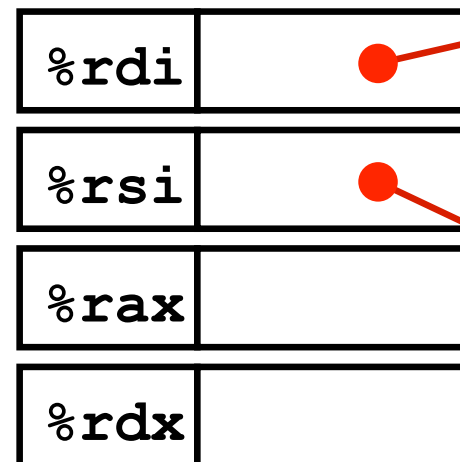
```
void swap(long *xp, long *yp) {  
    long t0 = *xp;  
    long t1 = *yp;  
    *xp = t1;  
    *yp = t0;  
}
```

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

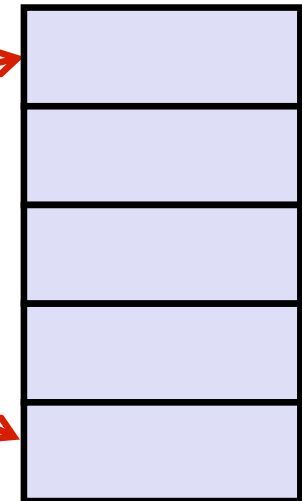
# Understanding Swap

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

Registers



Memory



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

swap:

movq

movq

movq

movq

ret

(%rdi), %rax # t0 = \*xp

(%rsi), %rdx # t1 = \*yp

%rdx, (%rdi) # \*xp = t1

%rax, (%rsi) # \*yp = t0

# General Memory Addressing

- General Form:

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

- **D** Constant “displacement”

- **Rb** Base register

- **Ri** Index register

- **S** Scale constant: 1, 2, 4, or 8

**Special Cases:**

**$Mem[Reg[rb]]$**

**$Mem[D + Reg[rb]]$**

**$(Rb, Ri)$        $Mem[Reg[Rb] + Reg[Ri]]$**

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

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



# Arithmetic Operations

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 Right Shift</b>
shrq	Src, Dest	Dest = Dest >> Src	<b>Logical Right Shift</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 Src, Dest      Dest = Src (as expr)**

**No memory access!**

# Arithmetic 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;
}
```

$(x, y, z) \rightarrow (\%rdi, \%rsi, \%rdx)$

```
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
```