Computer Systems Principles

x86-64 Assembly (Part 1)

UMASSCS
SCHOOL OF COMPUTER SCIENCE

Objectives

- **x86-64 Assembly Language**
  - To understand what assembly is
  - To learn about registers and memory
  - To understand basic x86-64 assembly instructions
  - To learn about addressing modes
  - To learn about condition codes

WHAT IS ASSEMBLY?

Assembly and Instructions

```
subl    $104, %esp
movl    8(%ebp), %eax
movl    %eax, -76(%ebp)
movl    12(%ebp), %eax
movl    %eax, -80(%ebp)
movl    %gs:20, %eax
movl    %eax, -12(%ebp)
xorl    %eax, %eax
movl    -76(%ebp), %eax
movl    %eax, fp
movl    $8, (%esp)
call    malloc
movl    %eax, -68(%ebp)
movl    -68(%ebp), %eax
movl    -80(%ebp), %edx
movl    %edx, (%eax)
movl    -68(%ebp), %eax
movl    %eax, (%esp)
call    parse_lines
movl    -68(%ebp), %edx
movl    %eax, 4(%edx)
movl    token, %eax
cmpl    $3, %eax
je  .L2
movl    token, %eax
movl    %eax, 4(%esp)
leal    -62(%ebp), %eax
```

Assembly Program (.s)

Statement:

Assignment

id3 = id1 op id2

id2 = op id1

id2 = id1

Stack operation

push id

id = pop()

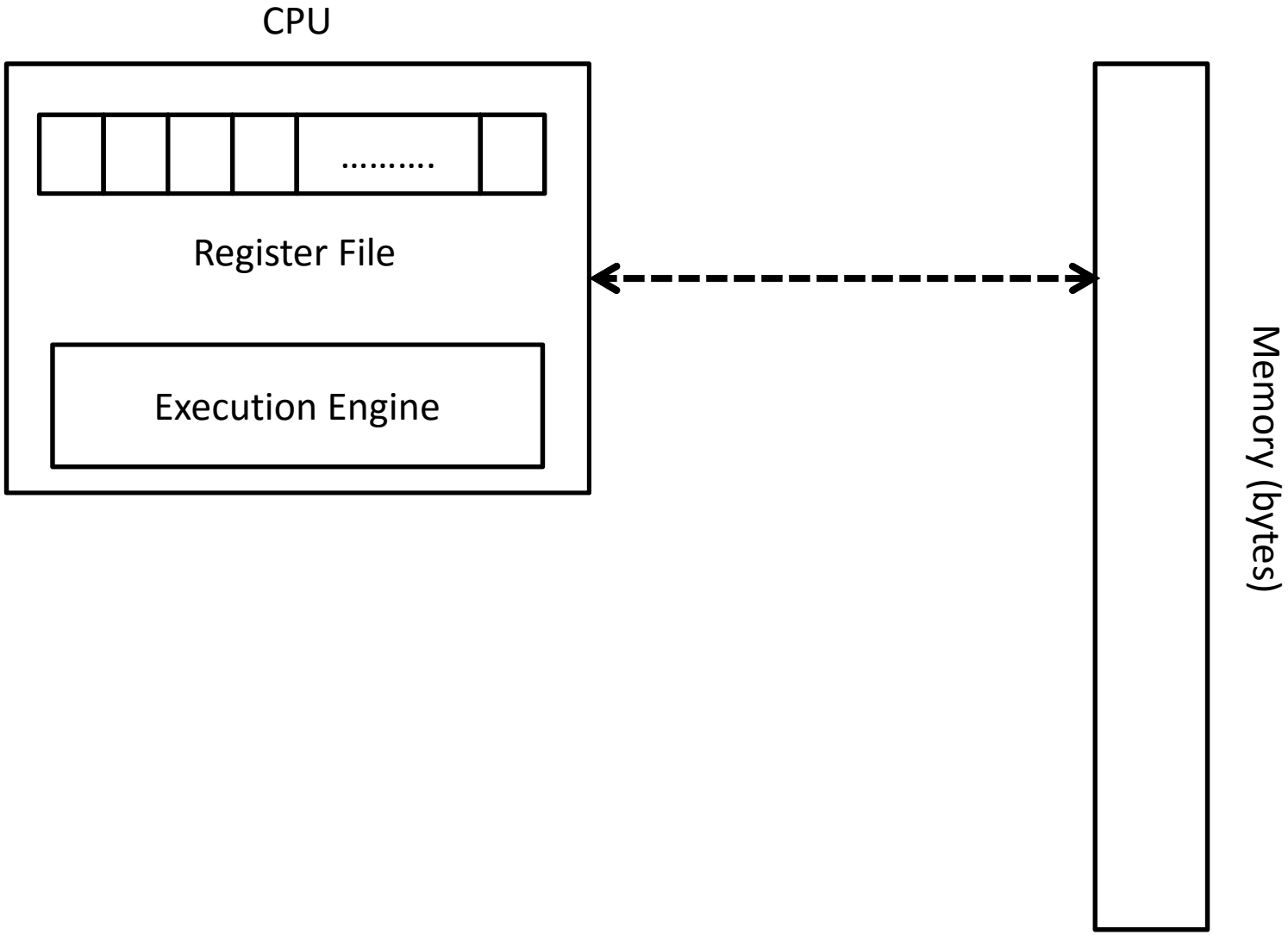
Jump

if id1 op id2 jump L

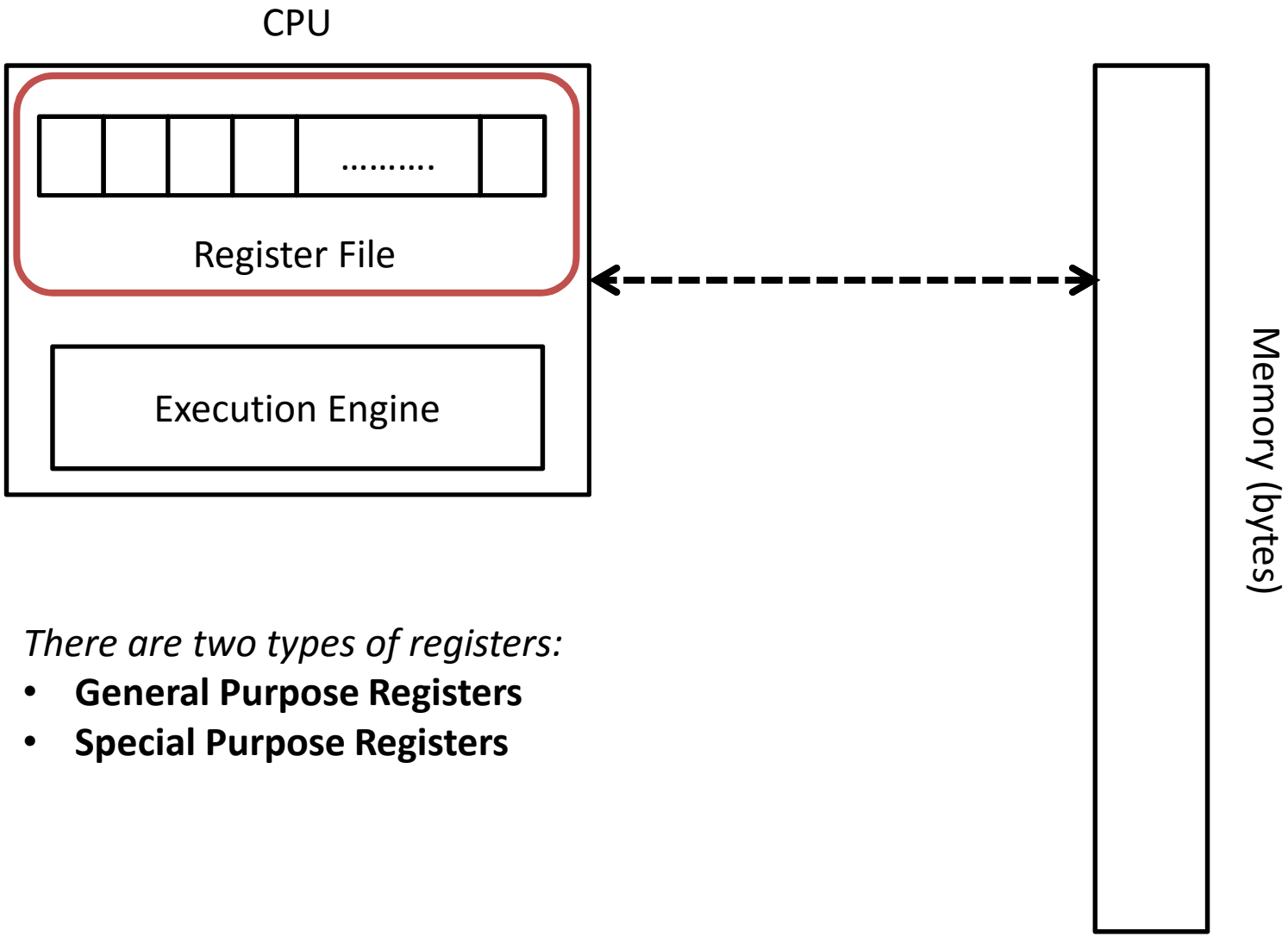
L

jump L

Machine Model



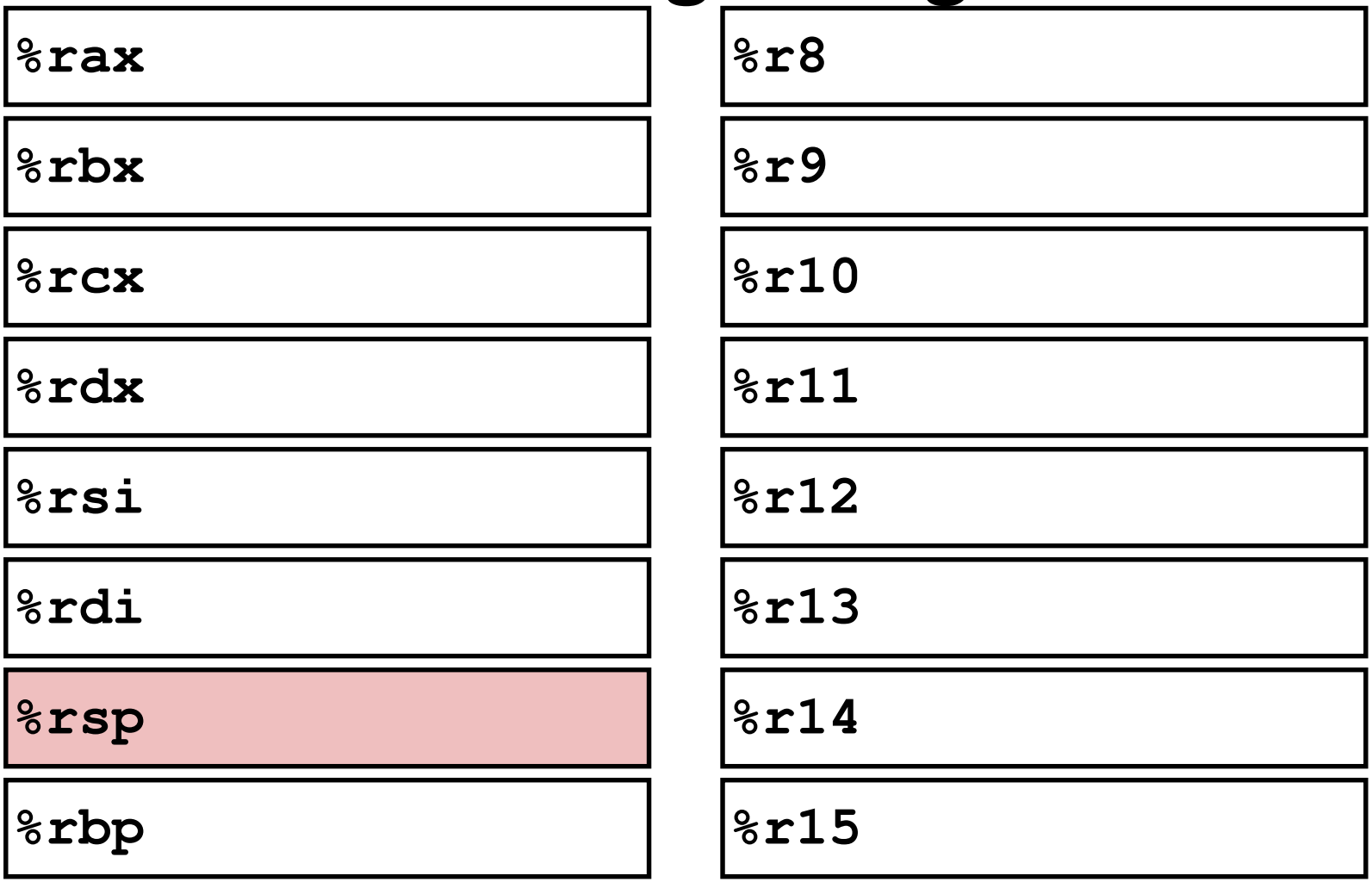
Machine Model



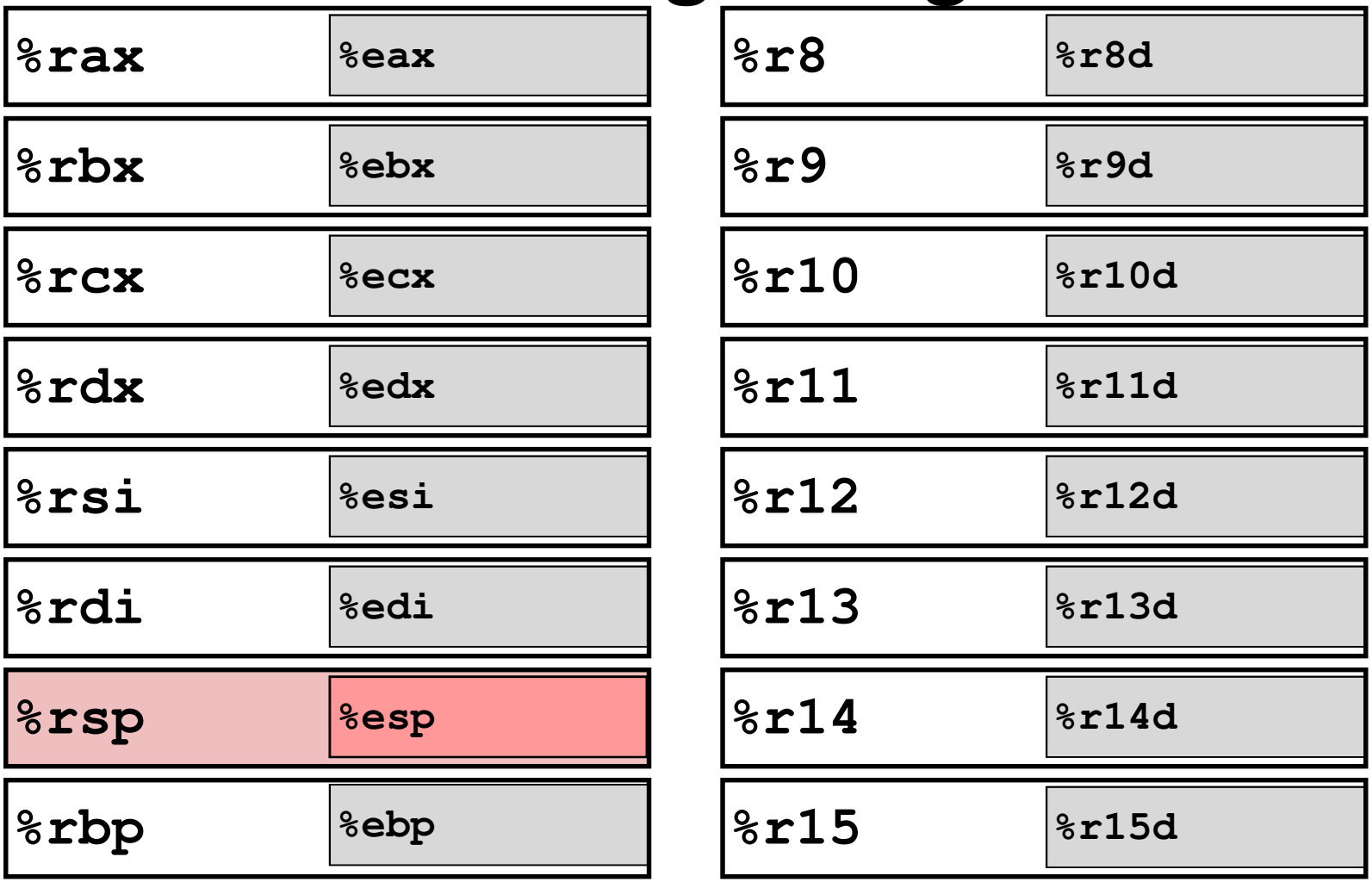
There are two types of registers:

- **General Purpose Registers**
- **Special Purpose Registers**

x86-64 Integer Registers

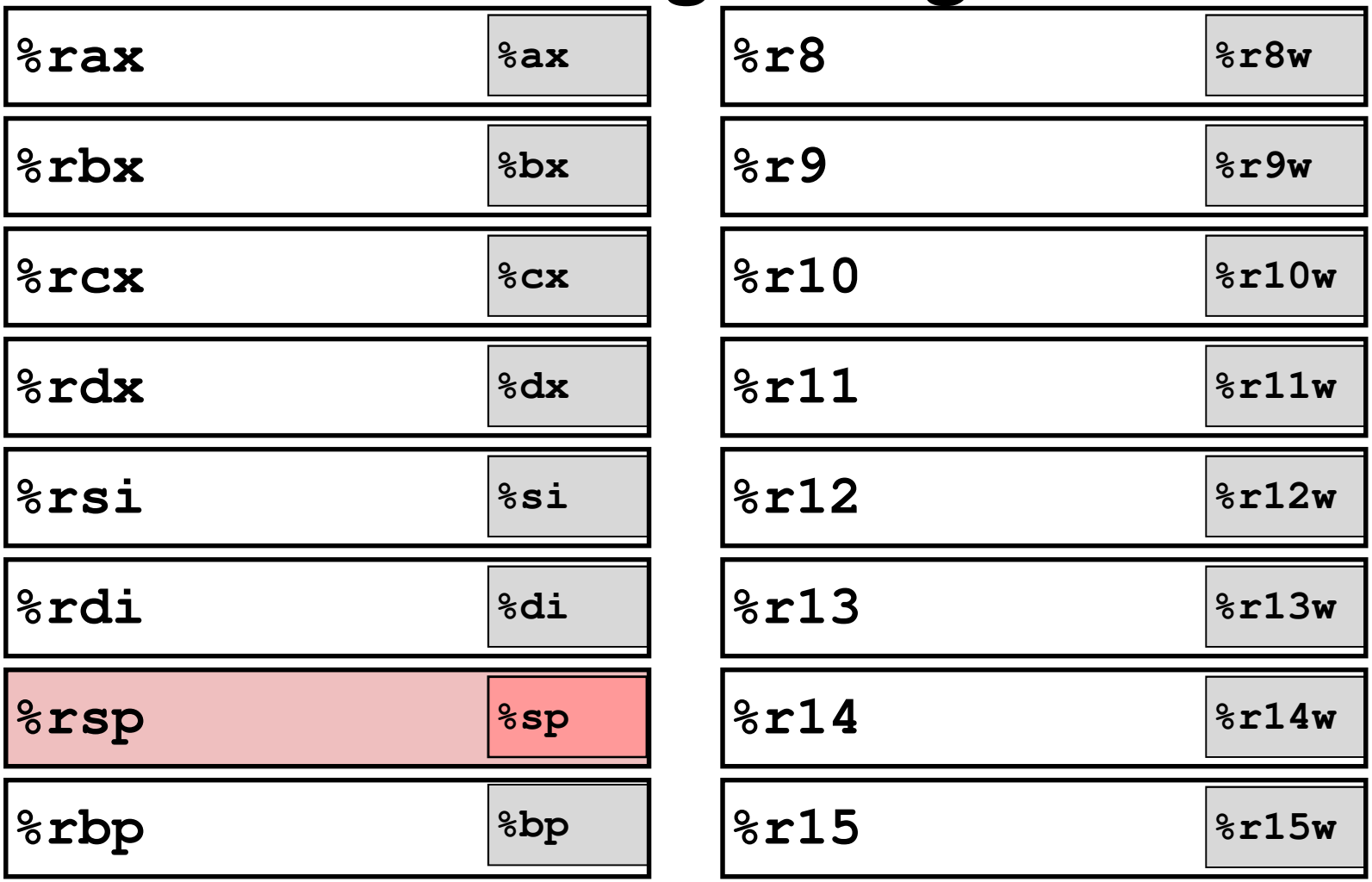


x86-64 Integer Registers



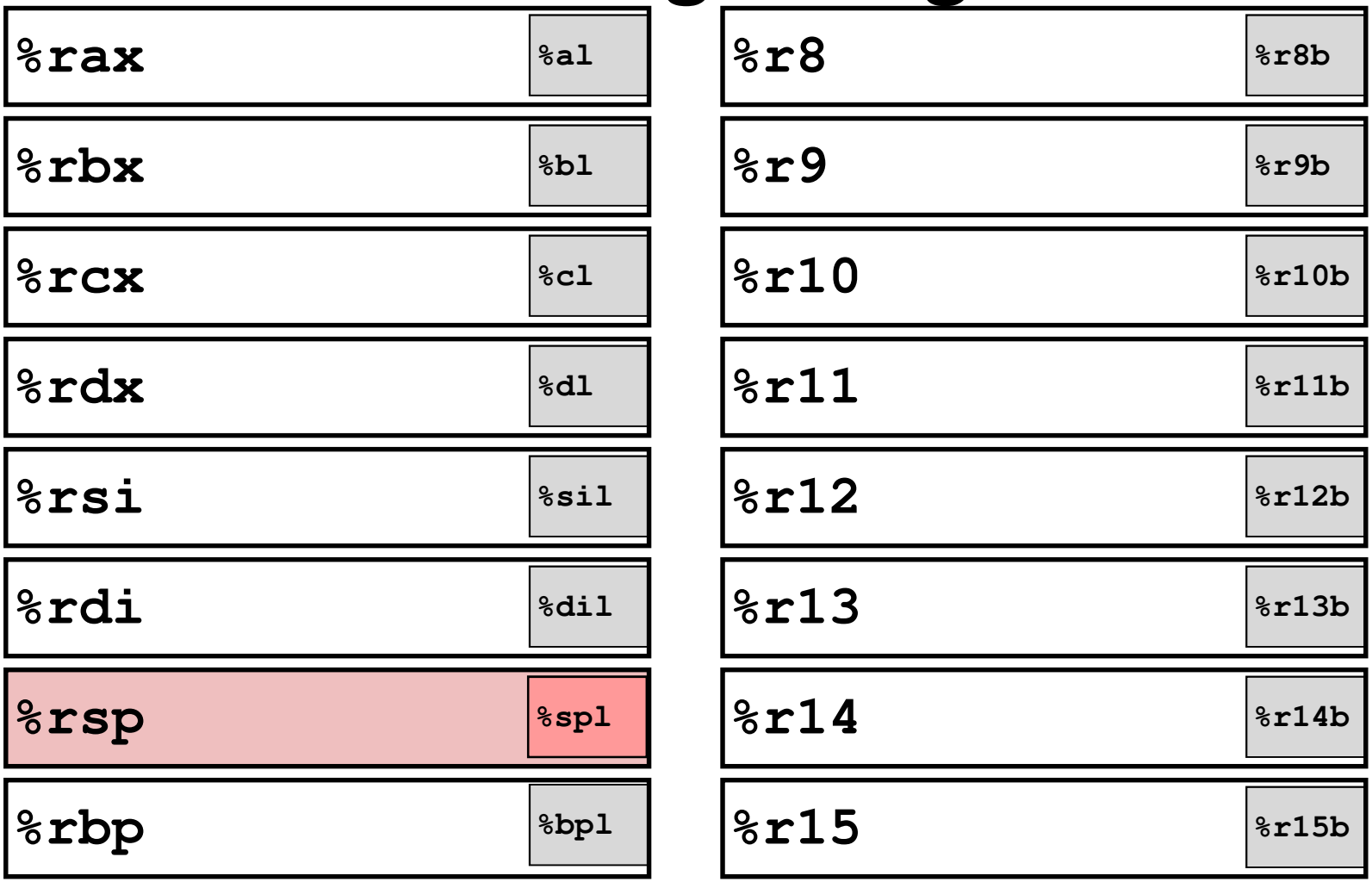
– Can reference low-order 4 bytes

x86-64 Integer Registers



– Can reference low-order 2 bytes

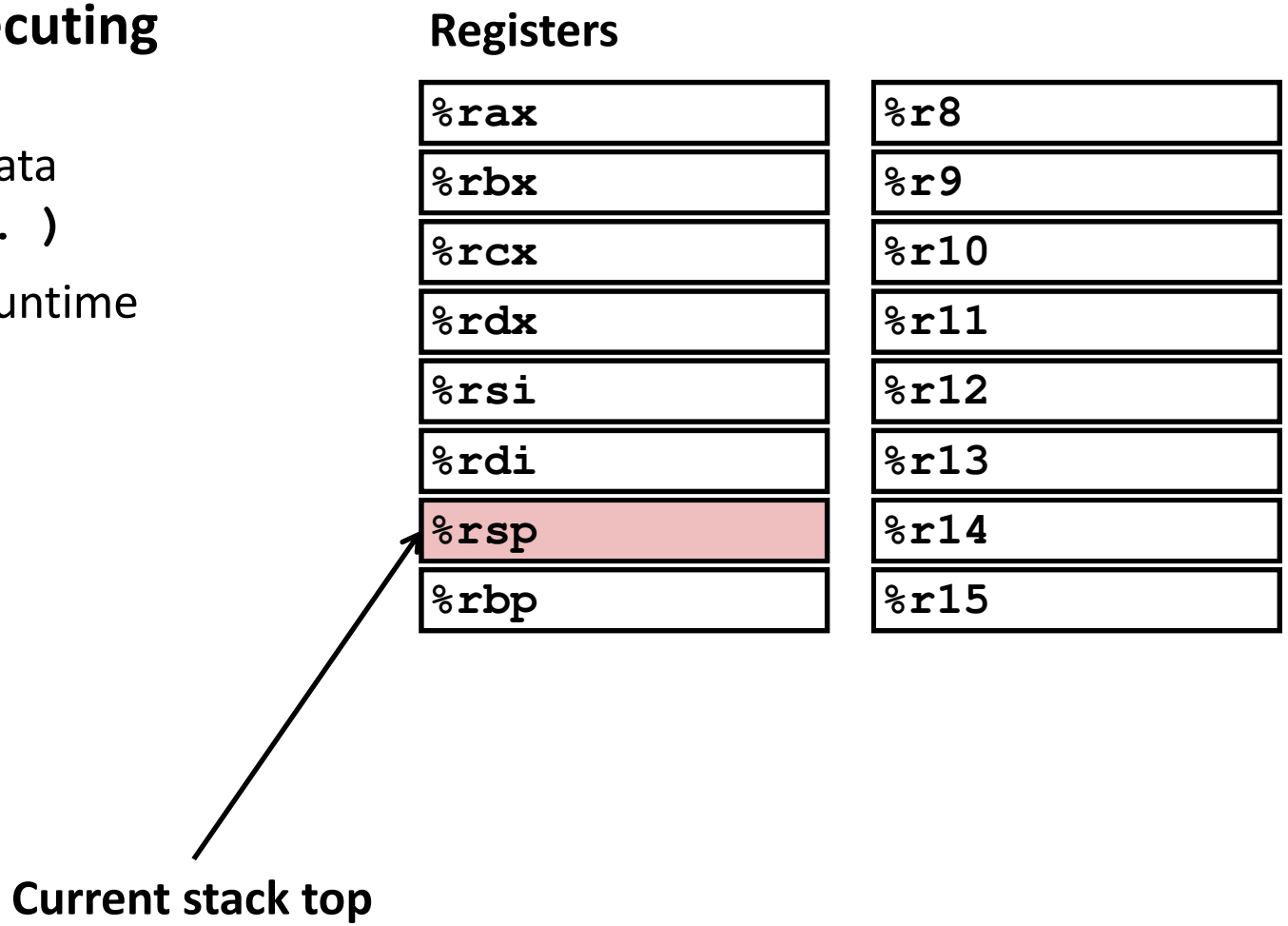
x86-64 Integer Registers



– Can reference low order byte

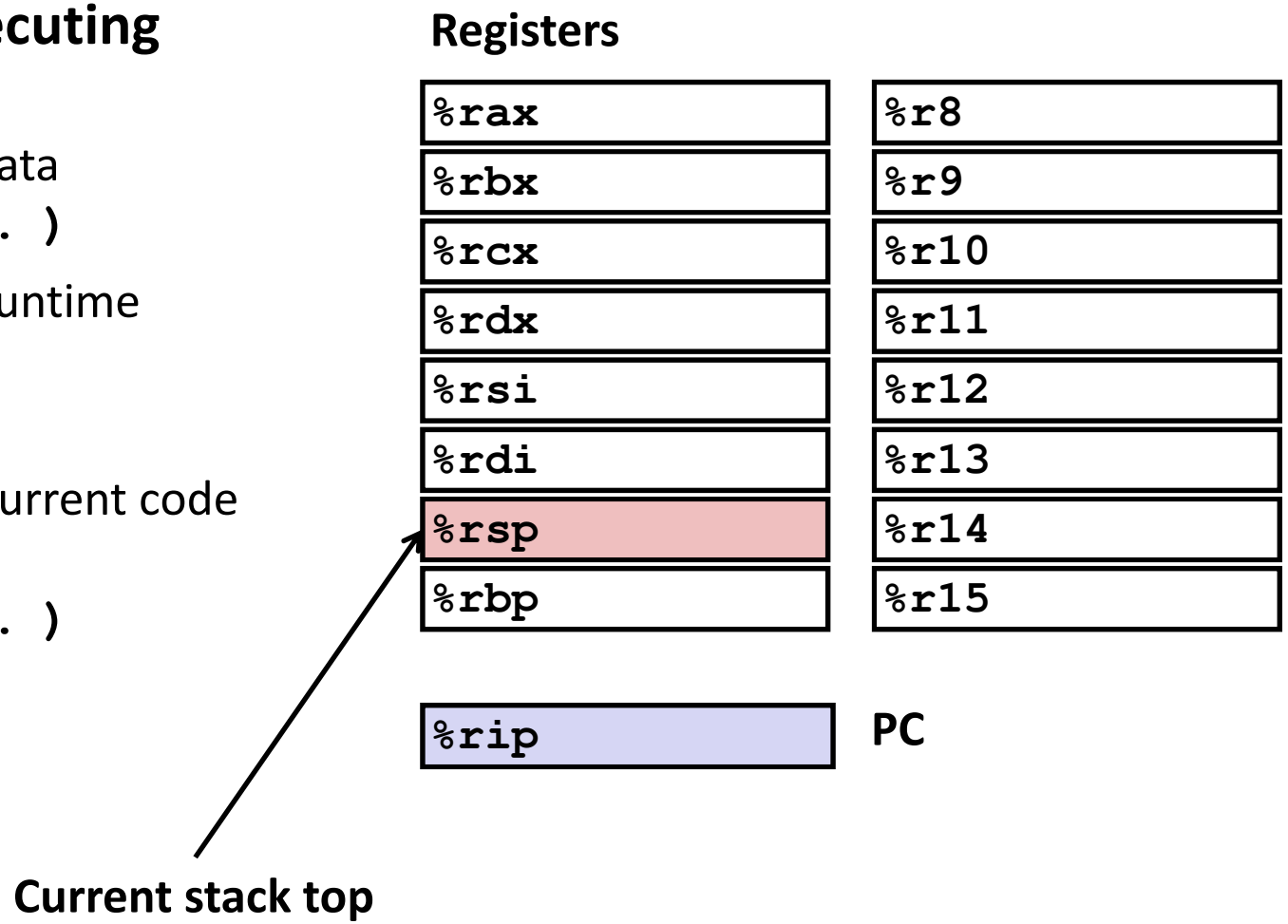
Processor State (x86-64, Partial)

- **Information about currently executing program**
  - Temporary data
    `( %rax, … )`
  - Location of runtime stack
    `( %rsp )`



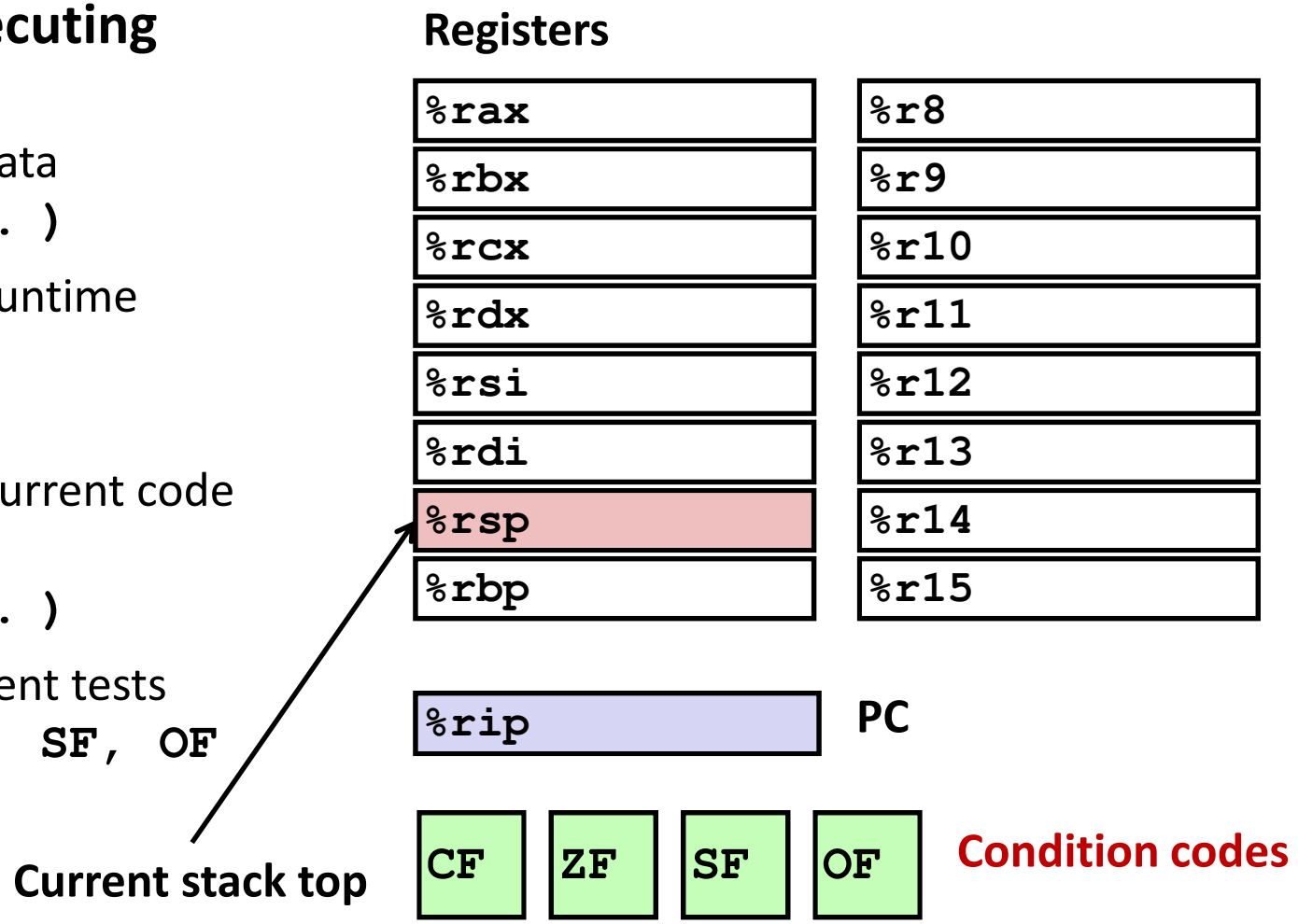
Processor State (x86-64, Partial)

- **Information about currently executing program**
  - Temporary data
    `( %rax, … )`
  - Location of runtime stack
    `( %rsp )`
  - Location of current code control point
    `( %rip, … )`

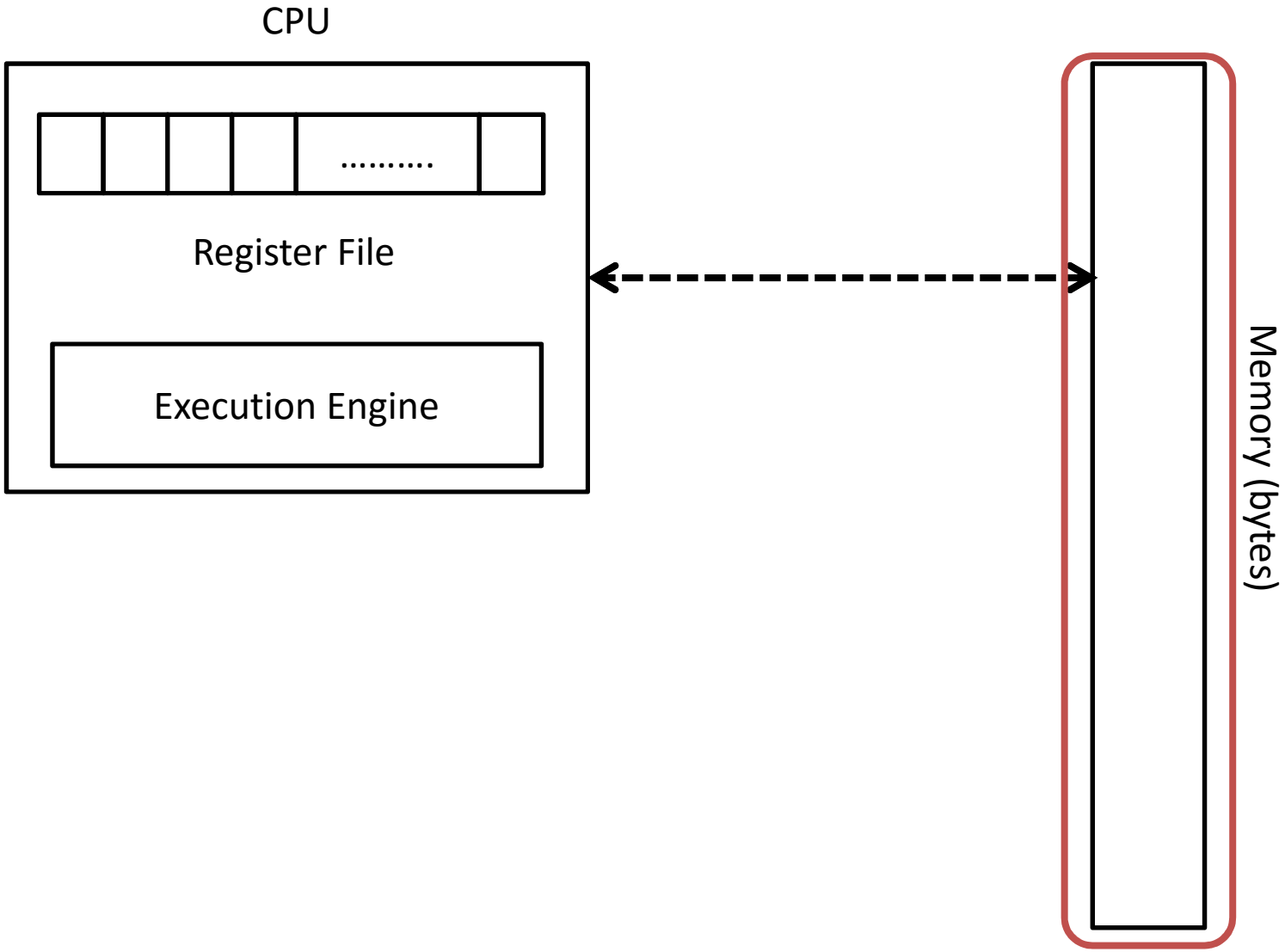


Processor State (x86-64, Partial)

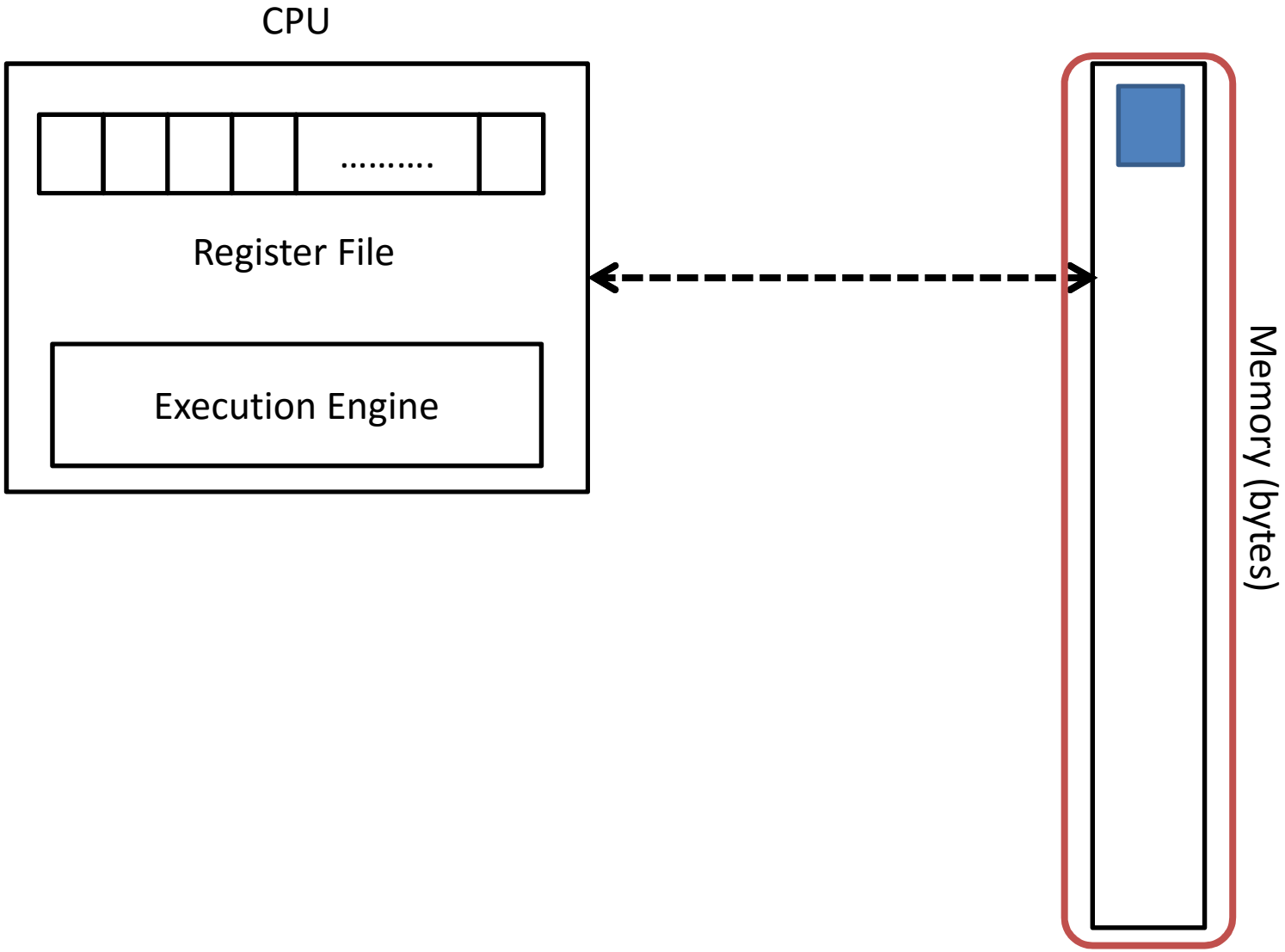
- **Information about currently executing program**
  - Temporary data
    (%rax, …)
  - Location of runtime stack
    (%rsp)
  - Location of current code control point
    (%rip, …)
  - Status of recent tests
    (CF, ZF, SF, OF)



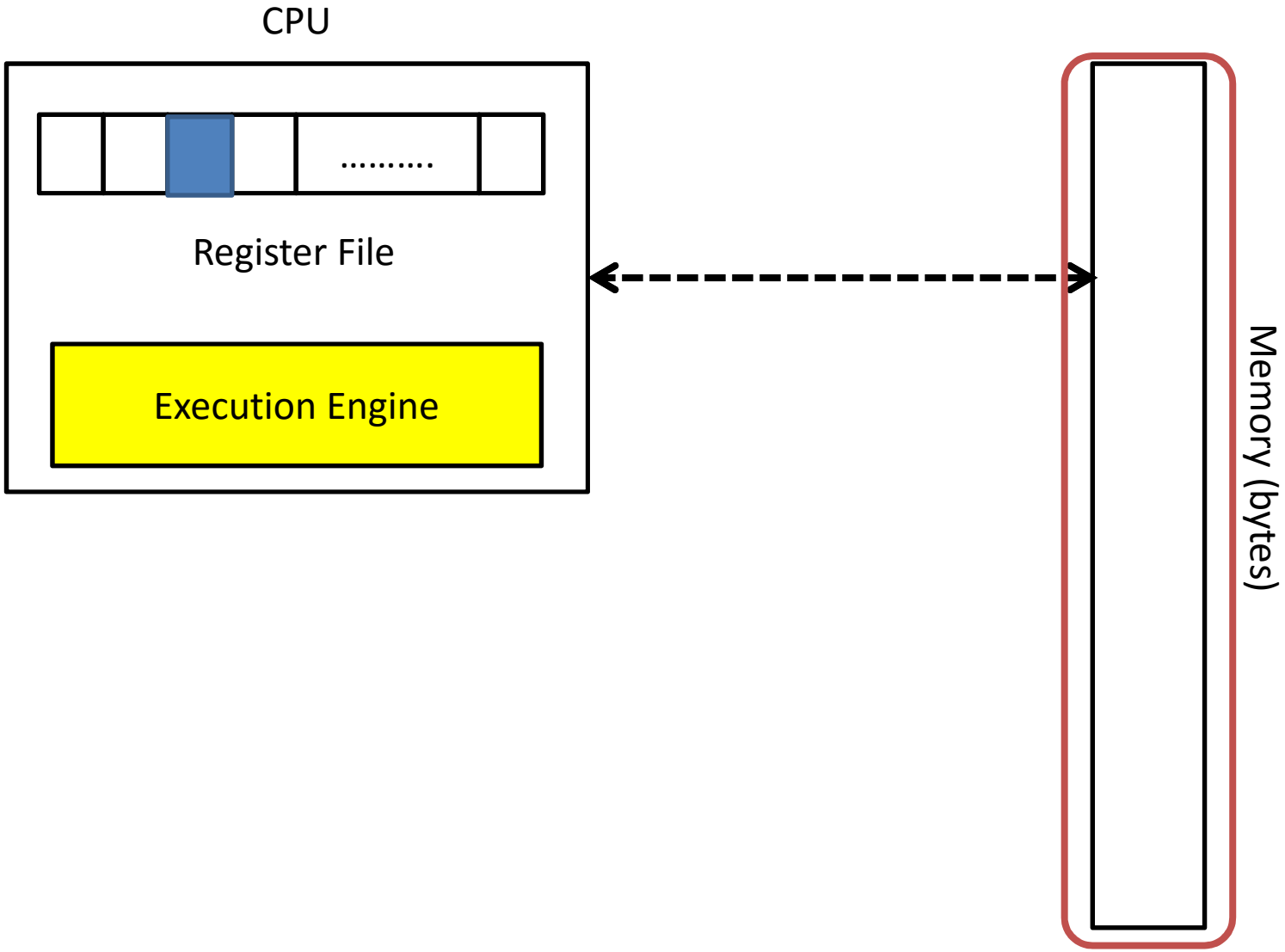
Machine Model



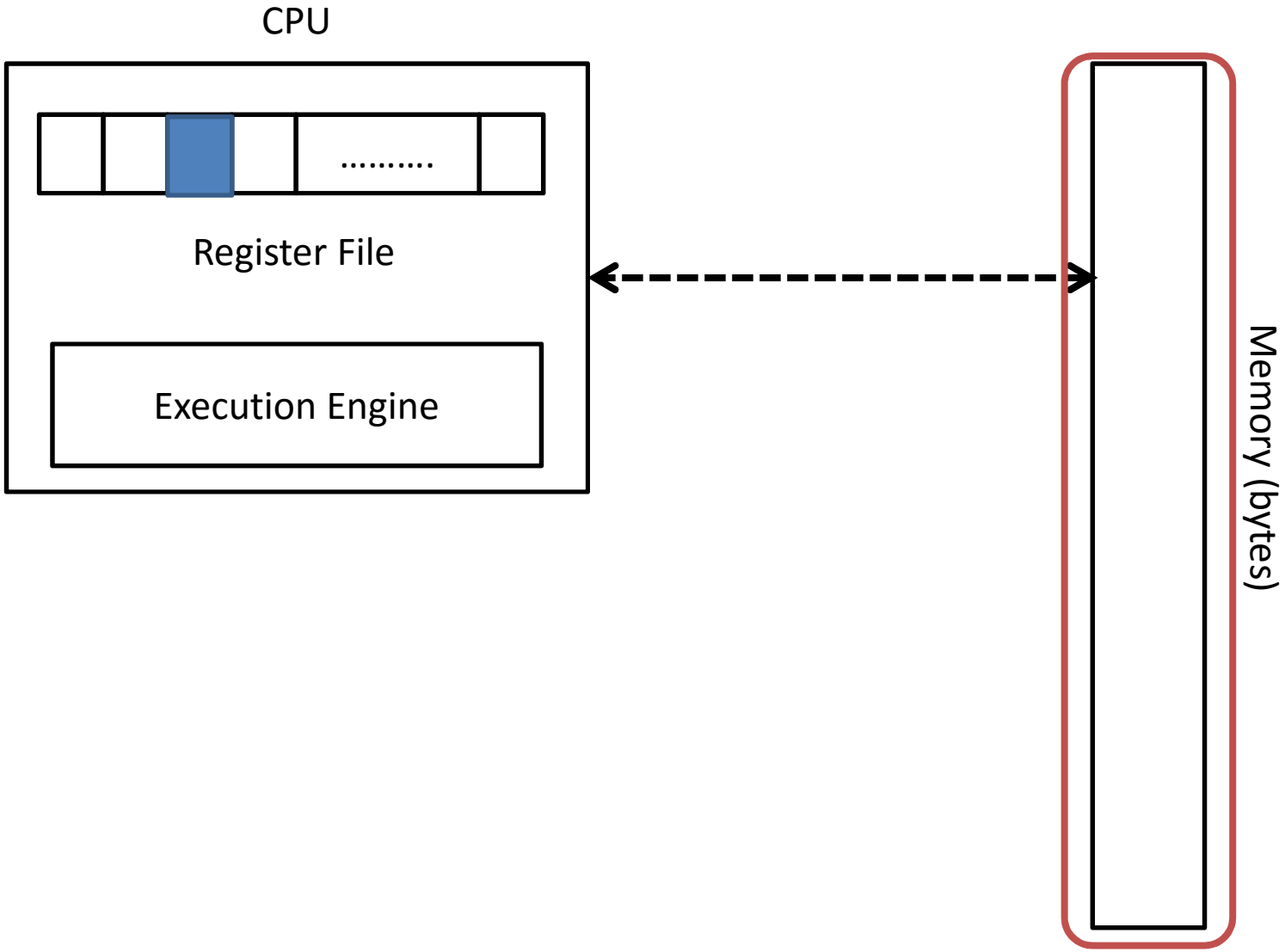
Machine Model



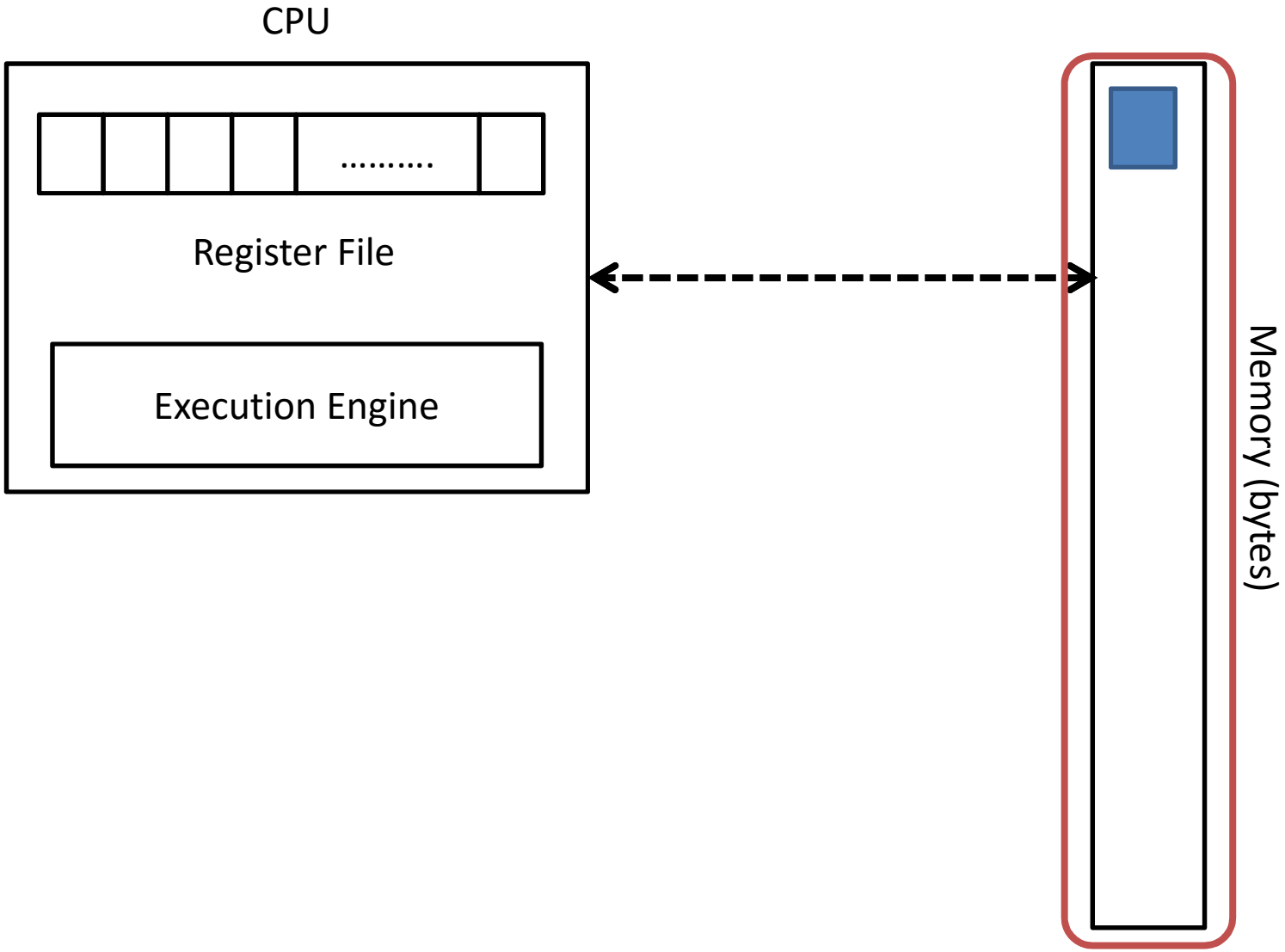
Machine Model



Machine Model



Machine Model



OPERAND SPECIFIER

General form of memory reference

- **Scaled index with base and immediate offset: Imm(r_b, r_i, s):**
  - Imm is displacement or offset
  - r_b is the base register
  - r_i is the index register (e.g., an array index)
  - s is the scale factor, which can only be 1, 2, 4 or 8

General form of memory reference

- **Scaled index with base and immediate offset: Imm(r_b, r_i, s):**
  - Imm is displacement or offset
  - r_b is the base register
  - r_i is the index register (e.g., an array index)
  - s is the scale factor, which can only be 1, 2, 4 or 8

General form of memory reference

- **Scaled index with base and immediate offset: Imm(r_b, r_i, s):**
  - Imm is displacement or offset
  - r_b is the base register
  - r_i is the index register (e.g., an array index)
  - s is the scale factor, which can only be 1, 2, 4 or 8
  - Computed as: Imm + R[r_b] + R[r_i] * s
  - Example: 9(%rbx,%rax,8)

General form of memory reference

- **Scaled index with base and immediate offset: Imm(r_b, r_i, s):**
  - Imm is displacement or offset
  - r_b is the base register
  - r_i is the index register (e.g., an array index)
  - s is the scale factor, which can only be 1, 2, 4 or 8
  - Computed as: Imm + R[r_b] + R[r_i] * s
  - Example: 9(%rbx,%rax,8)
  - Any of these can be omitted
  - Imm and register defaults to 0, scale factor to 1

What does it do?

- **Scaled index with base and displacement: 8(%ebp,%eax,4) … means:**

M[%ebp + (%eax*4) + 8]

What is it good for?

8(%rbp,%rax,4)

- **Typically an array of int starting at %rbp+8**
  - %rax is the index
- **The various scale factors are suitable for char (1), short (2), int (4), long long (8), float (4), double (8)**

Special cases of Imm(r_b, r_i, s)

- **Absolute: 0x2100 – *Note: no $!* … means M[0x2100], that is, the specific *address* is in the instruction**

Special cases of Imm(r_b, r_i, s)

- **Register indirect: (%rax) … means M[%rax] where %rax is the value in %rax**
  - M[i] means the byte(s) at location i in main memory, i.e., think of memory as a big array, M

Special cases of Imm(r_b, r_i, s)

- **Register indirect: (%rax) … means M[%rax] where %rax is the value in %rax**
  - M[i] means the byte(s) at location i in main memory, i.e., think of memory as a big array, M
- **Base + Register displacement: 4(%rbp) … means M[%rbp+4] where %rbp is the value in %rbp**

Group Activity

Assume the following values are stored at the indicated memory address and registers:

Address	Value	Register	Value
0x100	0x01	%rax	0x100
0x104	0x02	%rcx	0x1
0x108	0x03	%rdx	0x3
0x10C	0x04		

Operand	Value
$Imm	Imm
Imm	M[Imm]
$t	R[$t]
($t)	M[R[$t]]
Imm($t)	M[R[$t]+Imm]
Imm($t1, $t2)	M[R[$t1]+R[$t2]+Imm]
Imm(,$t2,s)	M[R[$t2]*s+Imm]
Imm($t1,$t2,s)	M[R[$t1]+R[$t2]*s+Imm]

Show the value for the indicated operands:

%rax, 0x104, $0x108, (%rax), 4(%rax), 9(%rax,%rdx), 260(%rcx,%rdx), 0xFC(,%rcx,4), (%rax,%rdx,4)

Group Activity

Assume the following values are stored at the indicated memory address and registers:

Address	Value	Register	Value
0x100	0x01	%rax	0x100
0x104	0x02	%rcx	0x1
0x108	0x03	%rdx	0x3
0x10C	0x04		

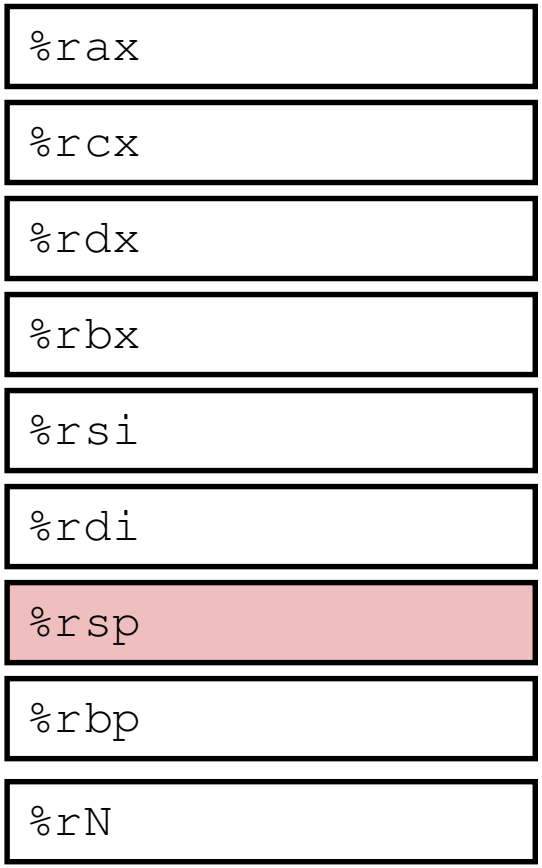
Operand	Value
$Imm	Imm
Imm	M[Imm]
$t	R[$t]
($t)	M[R[$t]]
Imm($t)	M[R[$t]+Imm]
Imm($t1, $t2)	M[R[$t1]+R[$t2]+Imm]
Imm(,$t2,s)	M[R[$t2]*s+Imm]
Imm($t1,$t2,s)	M[R[$t1]+R[$t2]*s+Imm]

Show the value for the indicated operands:

%rax, 0x104, $0x108, (%rax), 4(%rax), 9(%rax,%rdx), 260(%rcx,%rdx), 0xFC(,%rcx,4), (%rax,%rdx,4)

Sol: 0x100, 0x02, 0x108, 0x01, 0x02, 0x04, 0x03, 0x01, 0x04

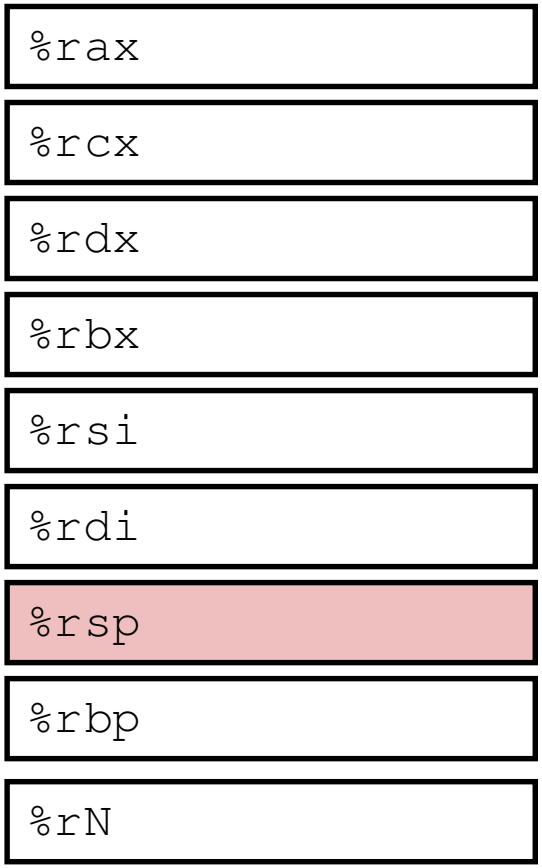
Moving Data: x86-64



Moving Data: x86-64

- **Moving Data**

  `movq` ***Source, Dest***

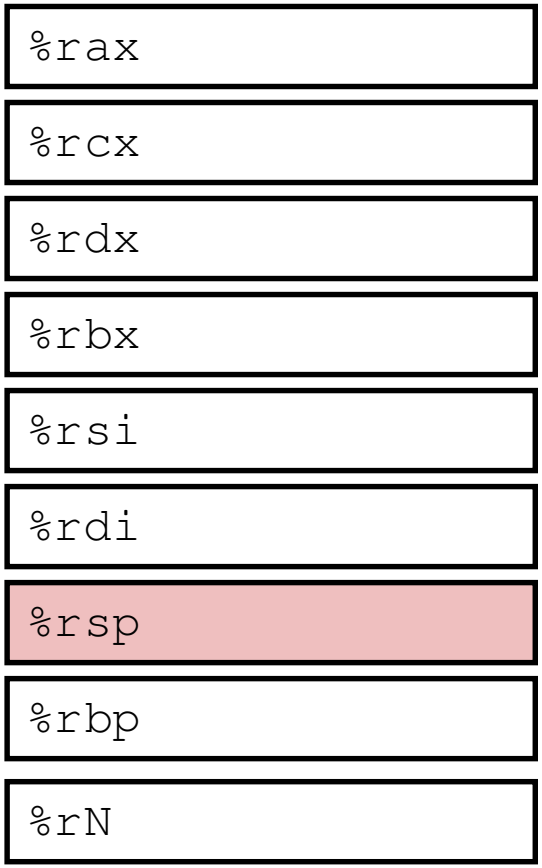


Moving Data: x86-64

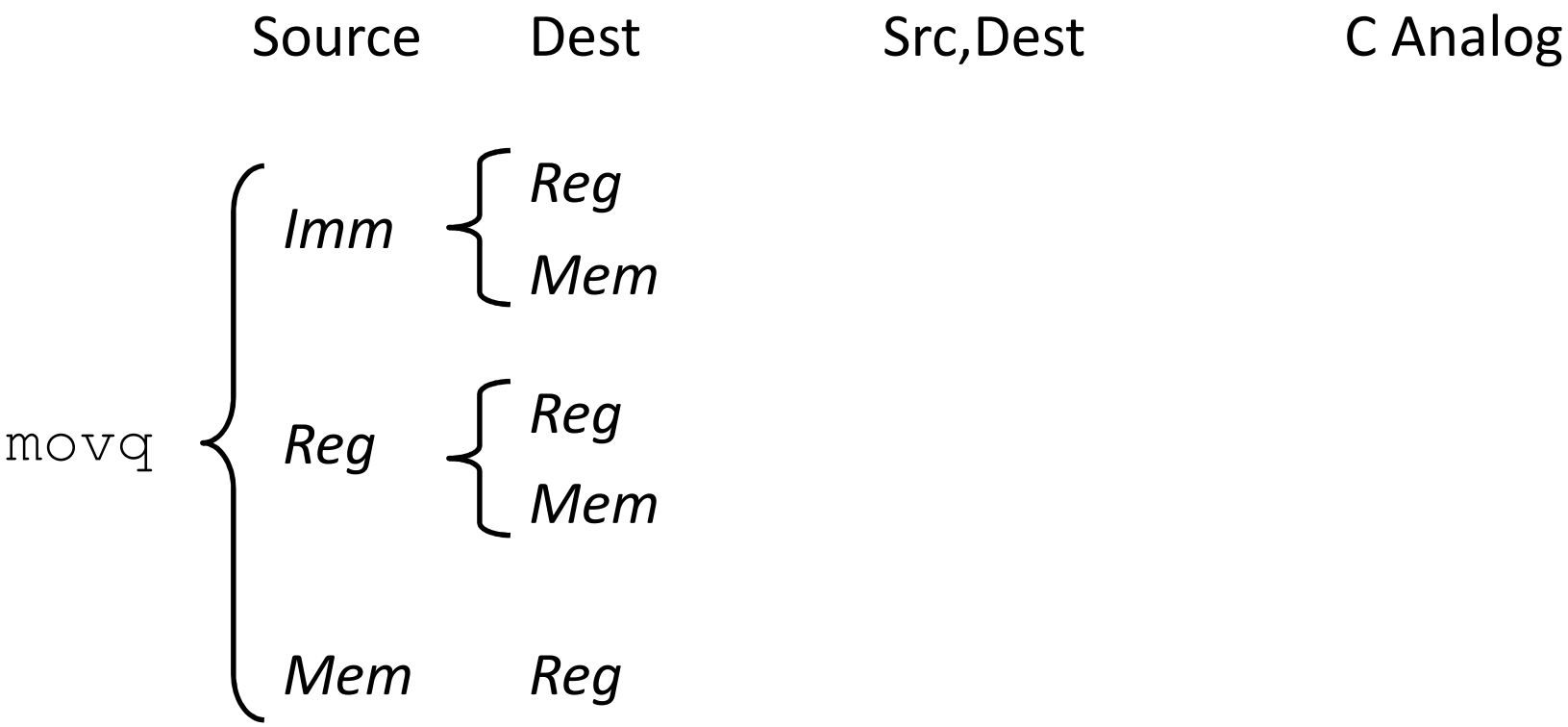
- **Moving Data**

  `movq` *Source*, *Dest*:

- **Operand Types**
  - ***Immediate:*** Constant integer data
    - Example: **`$0x400`, `$-533`**
    - Like C constant, but prefixed with **`'$'`**
    - Encoded with 1, 2, or 4 bytes
  - ***Register:*** One of 16 integer registers
    - Example: **`%rax`**, **`%rdx`**
    - But **`%rsp`** reserved for special use
    - Others have special uses for particular instructions
  - ***Memory:*** 8 consecutive bytes of memory at address given by register
    - Simplest example: **`(%rax)`** (Note: (reg) is like *reg in C.)

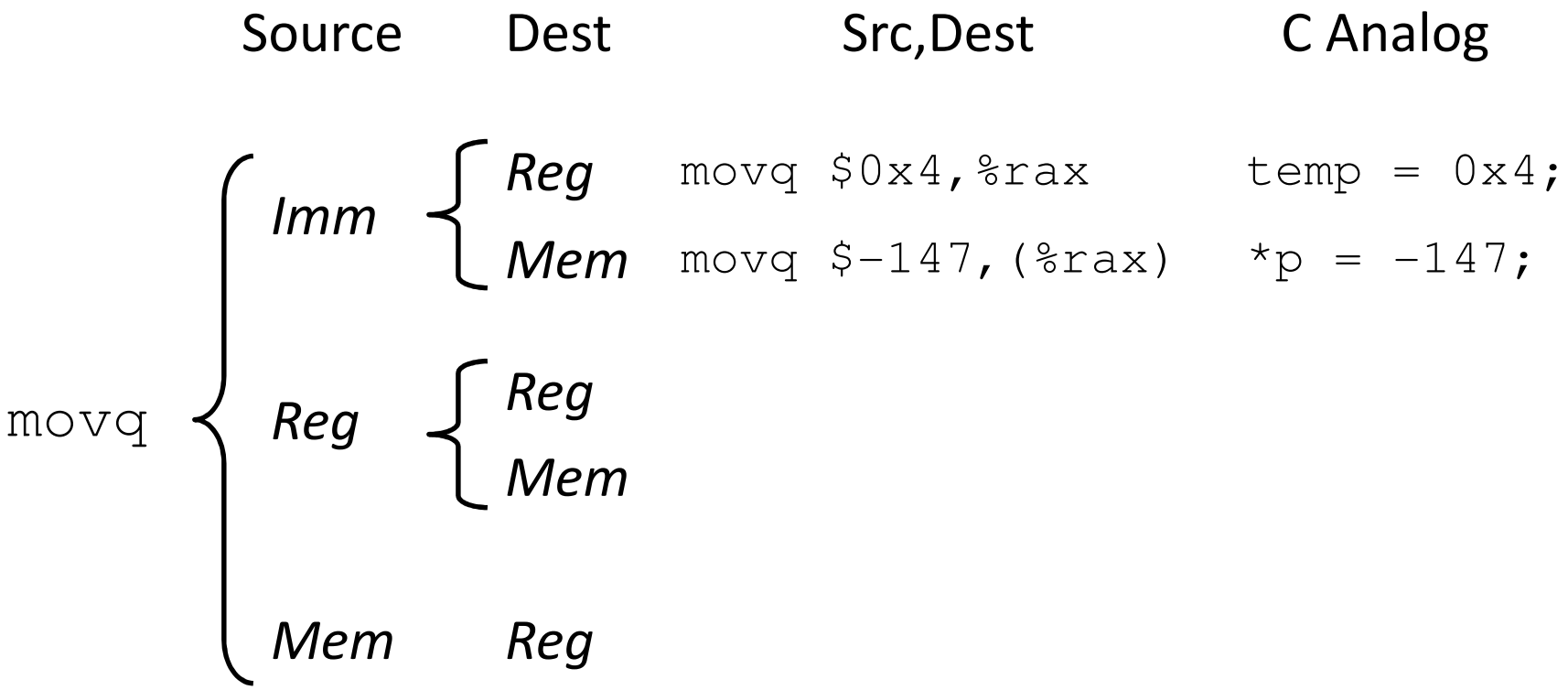


`movq` Operand Combinations



Cannot do memory-memory transfer with a single instruction

`movq` Operand Combinations



Cannot do memory-memory transfer with a single instruction

`movq` 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*		

Cannot do memory-memory transfer with a single instruction

`movq` 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;`

Cannot do memory-memory transfer with a single instruction

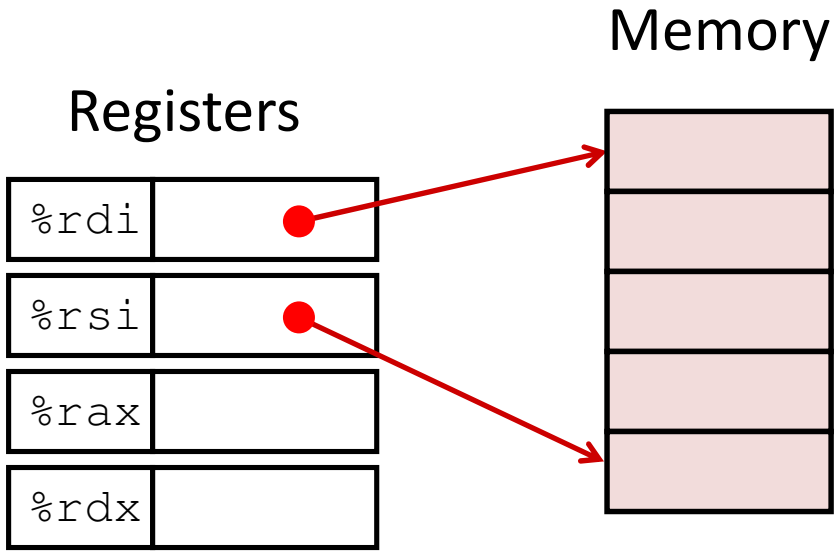
Example of Simple Addressing Modes

```
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()`

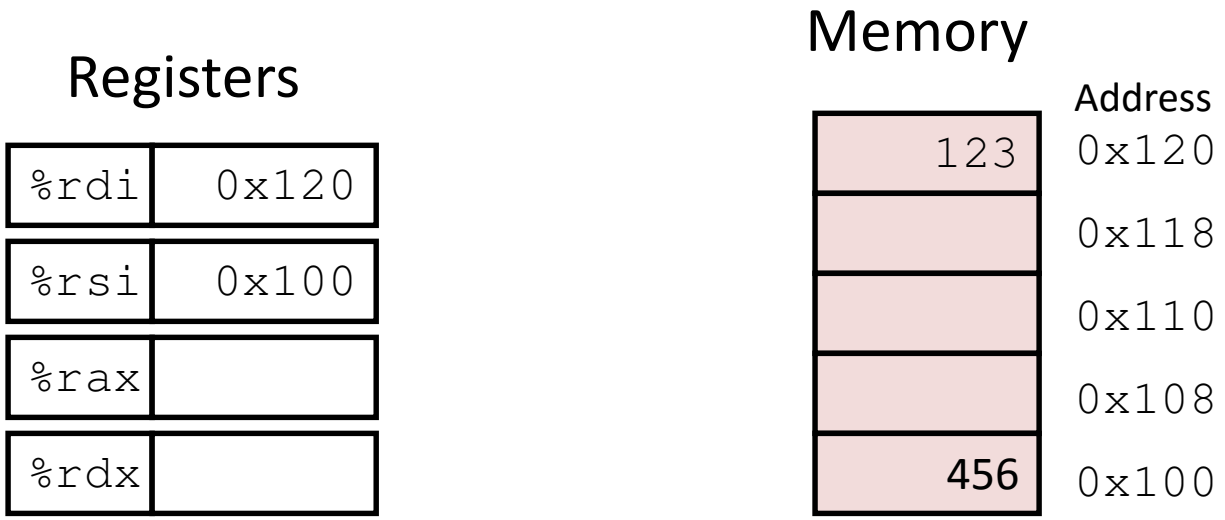
```
void swap
    (long *xp, long *yp)
{
  long t0 = *xp;
  long t1 = *yp;
  *xp = t1;
  *yp = t0;
}
```



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

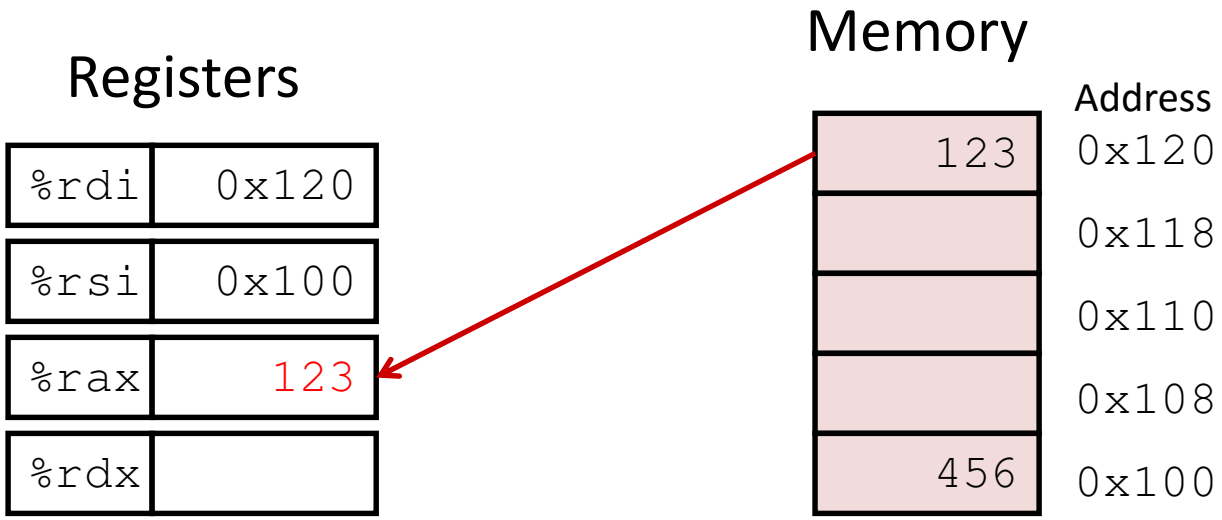
```
swap:
    movq    (%rdi), %rax  # t0 = *xp
    movq    (%rsi), %rdx  # t1 = *yp
    movq    %rdx, (%rdi)  # *xp = t1
    movq    %rax, (%rsi)  # *yp = t0
    ret
```

Understanding `Swap()`



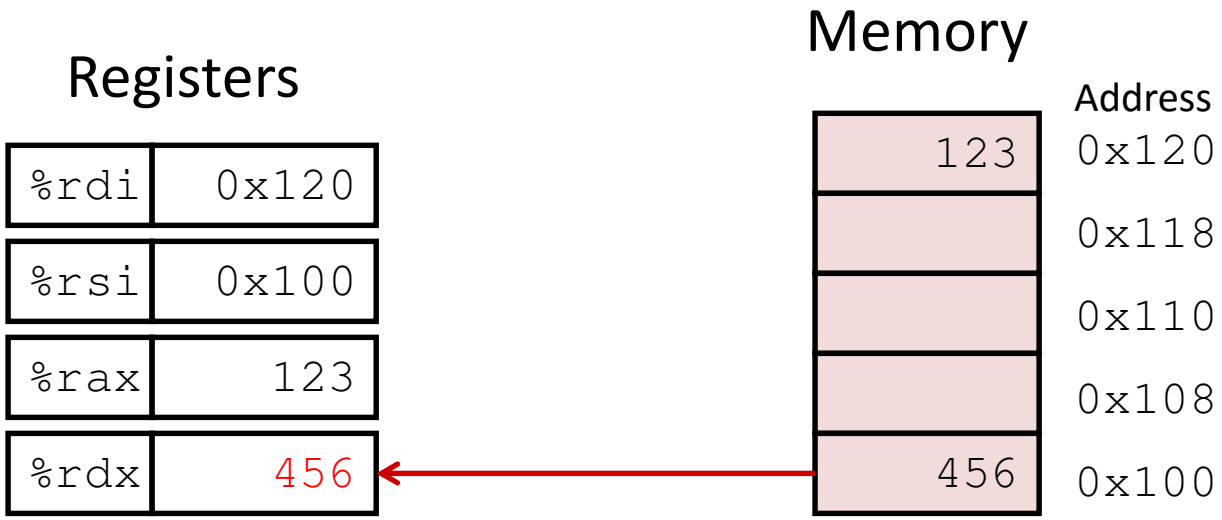

```
swap:
    movq    (%rdi), %rax  # t0 = *xp
    movq    (%rsi), %rdx  # t1 = *yp
    movq    %rdx, (%rdi)  # *xp = t1
    movq    %rax, (%rsi)  # *yp = t0
    ret
```

Understanding `Swap()`



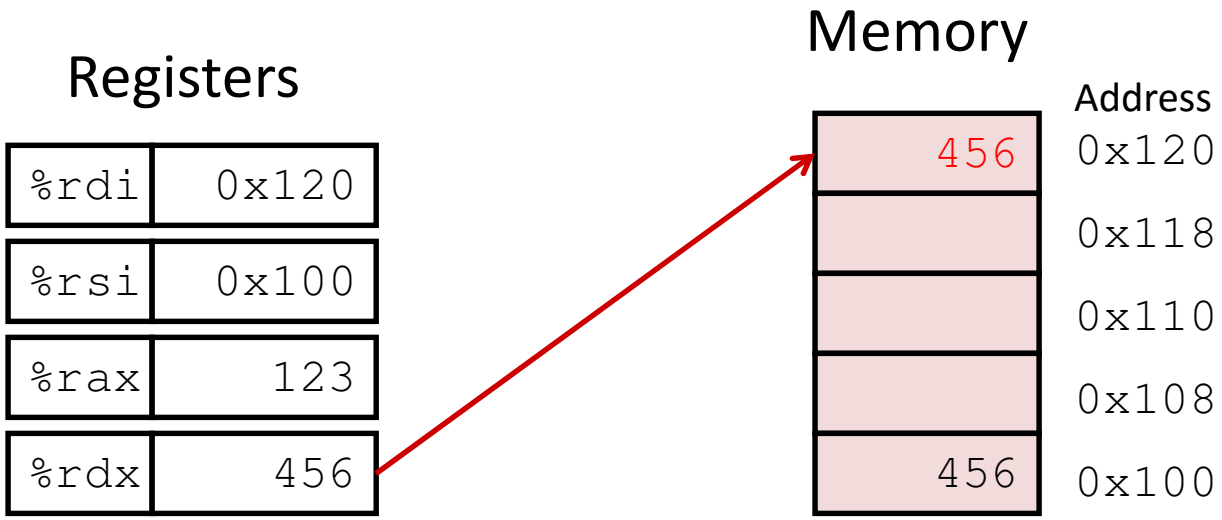

```
swap:
    movq    (%rdi), %rax  # t0 = *xp
    movq    (%rsi), %rdx  # t1 = *yp
    movq    %rdx, (%rdi)  # *xp = t1
    movq    %rax, (%rsi)  # *yp = t0
    ret
```

Understanding `Swap()`



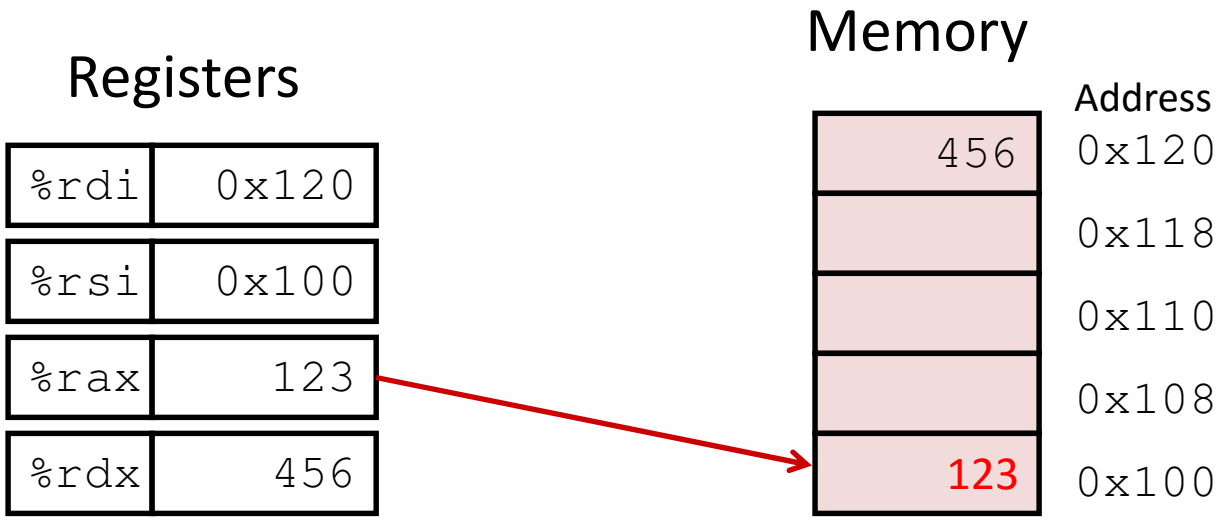

```
swap:
    movq    (%rdi), %rax  # t0 = *xp
    movq    (%rsi), %rdx  # t1 = *yp
    movq    %rdx, (%rdi)  # *xp = t1
    movq    %rax, (%rsi)  # *yp = t0
    ret
```

Understanding `Swap()`




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

Understanding `Swap()`



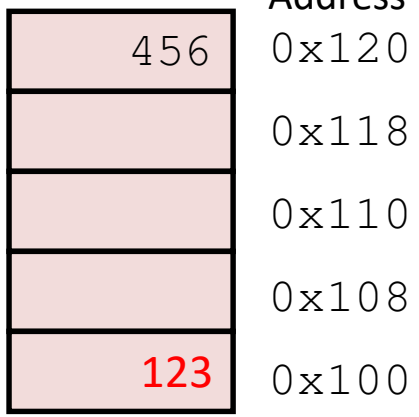

```
swap:
    movq    (%rdi), %rax  # t0 = *xp
    movq    (%rsi), %rdx  # t1 = *yp
    movq    %rdx, (%rdi)  # *xp = t1
    movq    %rax, (%rsi)  # *yp = t0
    ret
```

Understanding `Swap()`

Registers

%rdi	0x120
%rsi	0x100
%rax	123
%rdx	456

Memory




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

ARITHMETIC & LOGICAL OPERATION

Address Computation Instruction

- **`leaq Src, Dst`**
  - *Src is address mode expression*
  - *Set Dst to address denoted by expression*

Address Computation Instruction

- **`leaq Src, Dst`**
  - *Src is address mode expression*
  - *Set Dst to address denoted by expression*
- **`Uses`**
  - *Computing addresses without a memory reference*
    - *E.g., translation of p = &x[i];*

Address Computation Instruction

- **`leaq Src, Dst`**
  - *Src is address mode expression*
  - *Set Dst to address denoted by expression*
- **`Uses`**
  - *Computing addresses without a memory reference (v.s. **mov**)*
    - *E.g., translation of p = &x[i];*

Address Computation Instruction

- **`leaq Src, Dst`**
  - *Src is address mode expression*
  - *Set Dst to address denoted by expression*
- **`Uses`**
  - *Computing addresses without a memory reference (v.s. **mov**)*
    - *E.g., translation of p = &x[i];*
  - *Computing arithmetic expressions of the form x + k*y*
    - *k = 1, 2, 4, or 8*

Address Computation Instruction

- **`leaq Src, Dst`**
  - *Src is address mode expression*
  - *Set Dst to address denoted by expression*
- **`Uses`**
  - *Computing addresses without a memory reference*
    - *E.g., translation of p = &x[i];*
  - *Computing arithmetic expressions of the form x + k*y*
    - *k = 1, 2, 4, or 8*
- **`Example`**

```
long m12(long x)
{
  return x*12;
}
```

Converted to Assembly by compiler:

```
leaq (%rdi,%rdi,2), %rax # t <- x+x*2
salq $2, %rax            # return t<<2
```

iClicker question

Assuming %rbp holds 0x7fff0410 and %rsi holds 3, what value does this instruction load into %rdx?

leaq -0x110(%rbp,%rsi,4),%rdx

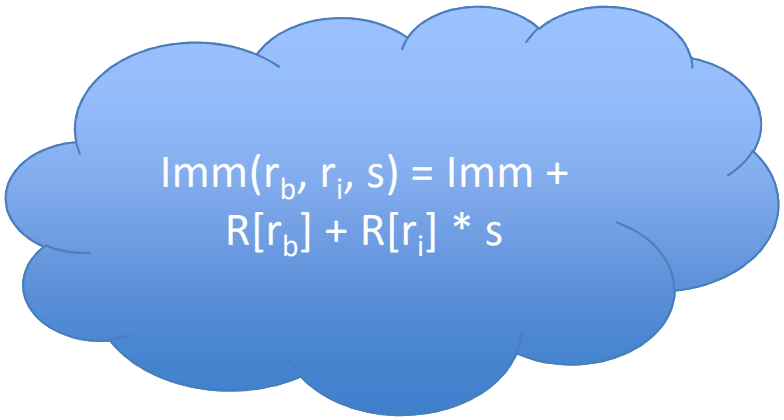
A) 0x7fff0300

B) 0x7fff030c

C) 0x7fff041c

D) 0x7fff0303

E) Something else



iClicker question

Assuming %rbp holds 0x7fff0410 and %rsi holds 3, what value does this instruction load into %rdx?

leaq -0x110(%rbp,%rsi,4),%rdx Sol: B

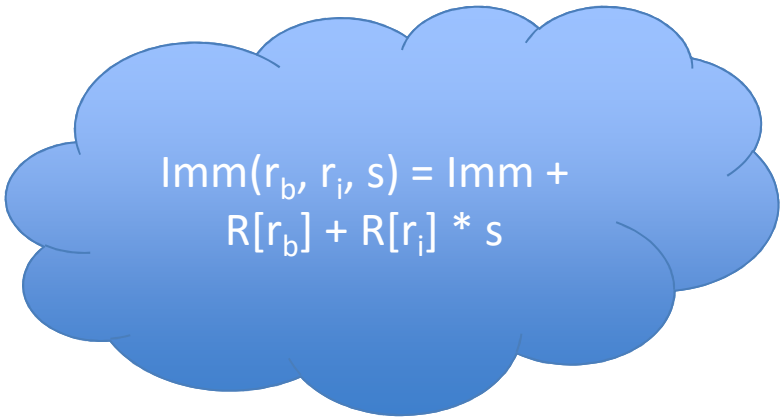
A) 0x7fff0300

B) 0x7fff030c

C) 0x7fff041c

D) 0x7fff0303

E) Something else



Some Arithmetic 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

Some Arithmetic 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

- **Watch out for argument order!**

Some Arithmetic 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

- **Watch out for argument order!**
- **No distinction between signed and unsigned int (why?)**

Some Arithmetic Operations

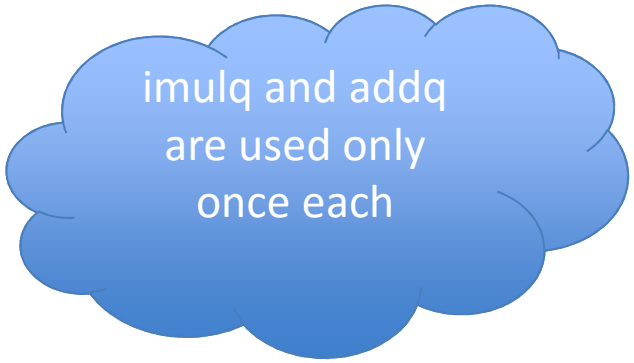
- **One Operand Instructions**

`incl`	*Dest*	*Dest = Dest + 1*
`decl`	*Dest*	*Dest = Dest – 1*
`negl`	*Dest*	*Dest = – Dest*
`notl`	*Dest*	*Dest = ~Dest*

Understanding Arithmetic Expression 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
```



Understanding Arithmetic Expression 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**
%rax	**t1**, **t2**, **rval**
%rdx	**t4**
%rcx	**t5**

Understanding Arithmetic Expression 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**
%rax	**t1**, **t2**, **rval**
%rdx	**t4**
%rcx	**t5**

Understanding Arithmetic Expression 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**
%rax	**t1**, **t2**, **rval**
%rdx	**t4**
%rcx	**t5**

Understanding Arithmetic Expression 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**
%rax	**t1**, **t2**, **rval**
%rdx	**t4**
%rcx	**t5**

Understanding Arithmetic Expression 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**
%rax	**t1**, **t2**, **rval**
%rdx	**t4**
%rcx	**t5**

Understanding Arithmetic Expression 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**
%rax	**t1**, **t2**, **rval**
%rdx	**t4**
%rcx	**t5**