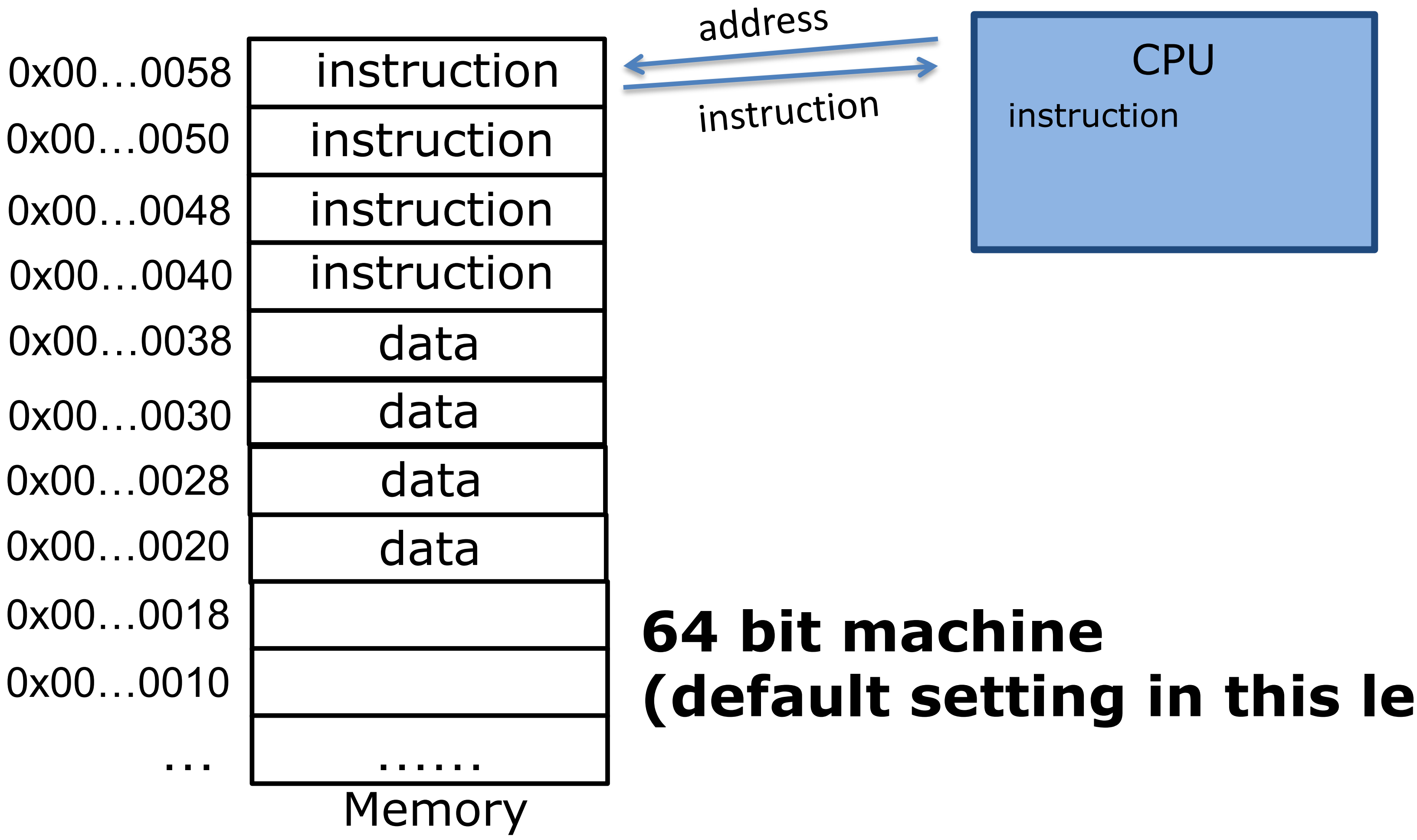
Machine Program: Basics

Shuai Mu

based on Tiger Wang's and Jinyang Li's slides

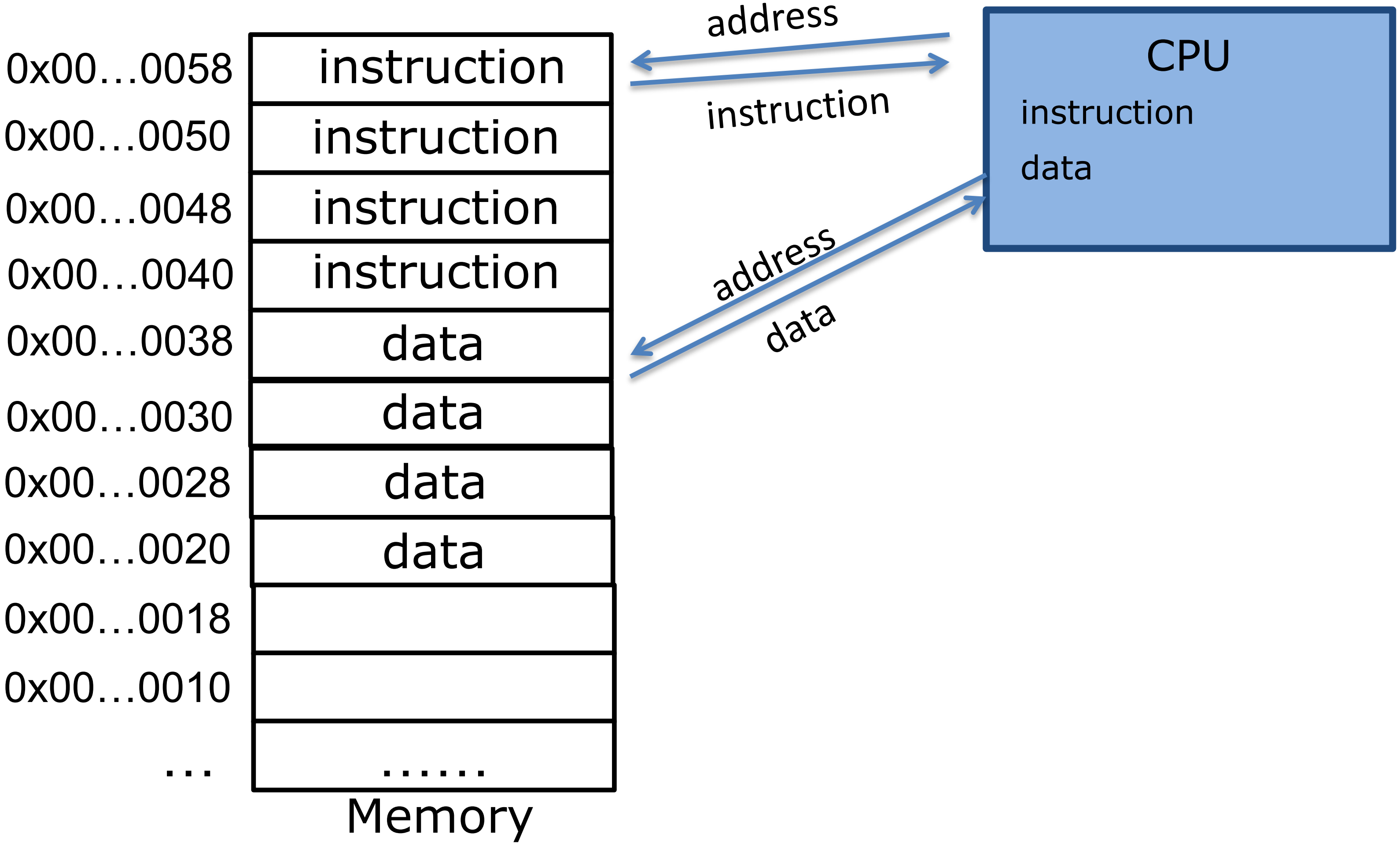
Your mental model



0x00…0058	instruction
0x00…0050	instruction
0x00…0048	instruction
0x00…0040	instruction
0x00…0038	data
0x00…0030	data
0x00…0028	data
0x00…0020	data
0x00…0018	
0x00…0010	
…	……

64 bit machine
(default setting in this lecture

Your mental model



0x00…0058	instruction
0x00…0050	instruction
0x00…0048	instruction
0x00…0040	instruction
0x00…0038	data
0x00…0030	data
0x00…0028	data
0x00…0020	data
0x00…0018	
0x00…0010	
…	……

Memory

address

instruction

CPU

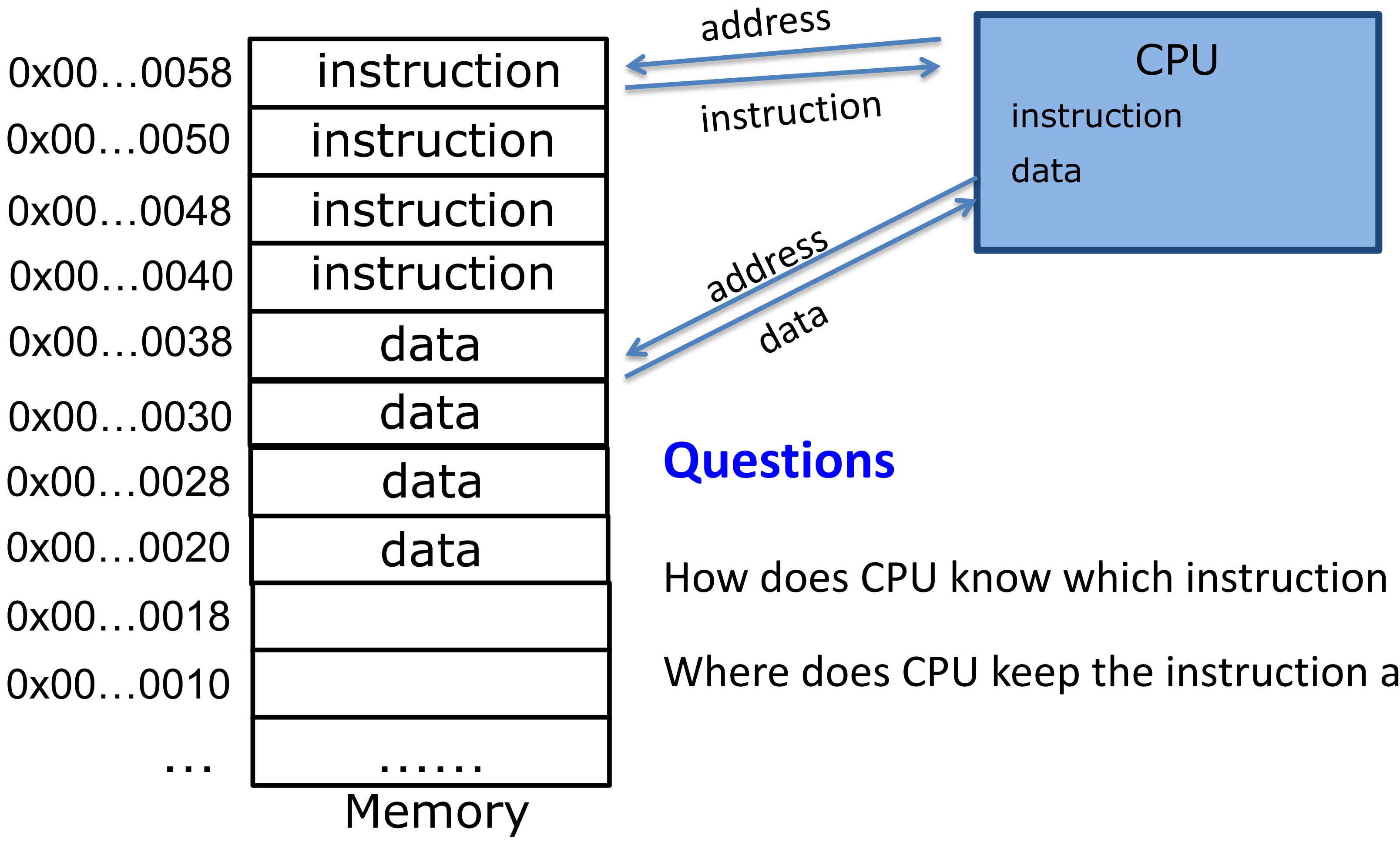
instruction

data

address

data

Your mental model



0x00…0058	instruction
0x00…0050	instruction
0x00…0048	instruction
0x00…0040	instruction
0x00…0038	data
0x00…0030	data
0x00…0028	data
0x00…0020	data
0x00…0018	
0x00…0010	
…	……

Memory

Questions

How does CPU know which instruction to fetch?

Where does CPU keep the instruction and data?

Register – temporary storage area built into a CPU

PC: Program counter

- Store memory address of next instruction
- Called "RIP" in x86_64

IR: instruction register

- Store the fetched instruction

General purpose registers:

- Store operands and pointers used by program

Program status and control register:

- Status of the program being executed
- Called "EFLAGS" in x86_64

Register – temporary storage area built into a CPU

PC: Program counter

- Store memory address of next instruction
- Also called “RIP” in x86_64

IR: instruction register

- Store the fetched instruction

General purpose registers:

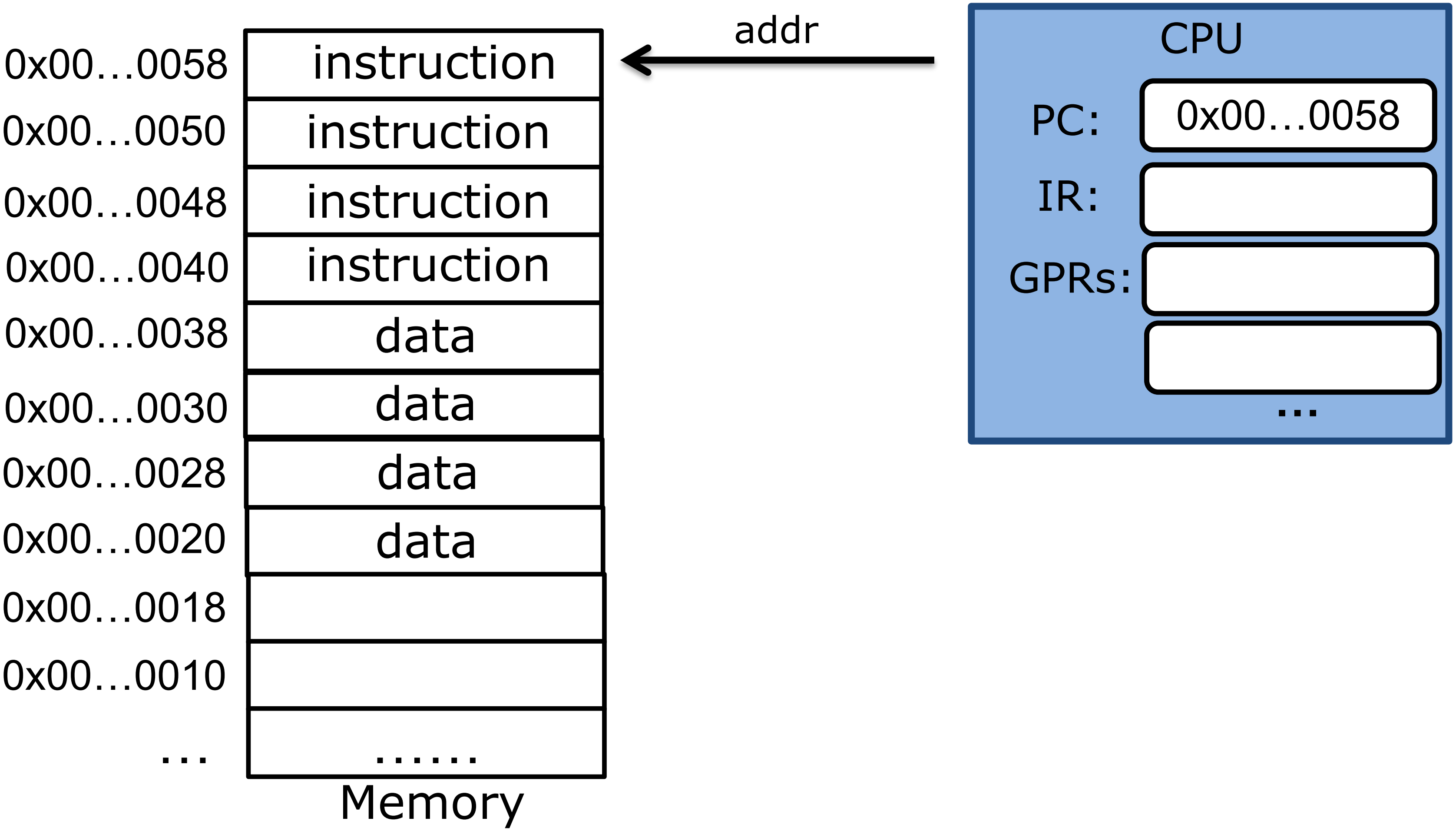
- Store operands and pointers used by program

Program status and control register:

- Status of the program being executed
- All called “EFLAGS” in x86_64

Visible to programmers

Your mental model



0x00…0058	instruction
0x00…0050	instruction
0x00…0048	instruction
0x00…0040	instruction
0x00…0038	data
0x00…0030	data
0x00…0028	data
0x00…0020	data
0x00…0018	
0x00…0010	
…	……

Memory

addr

CPU

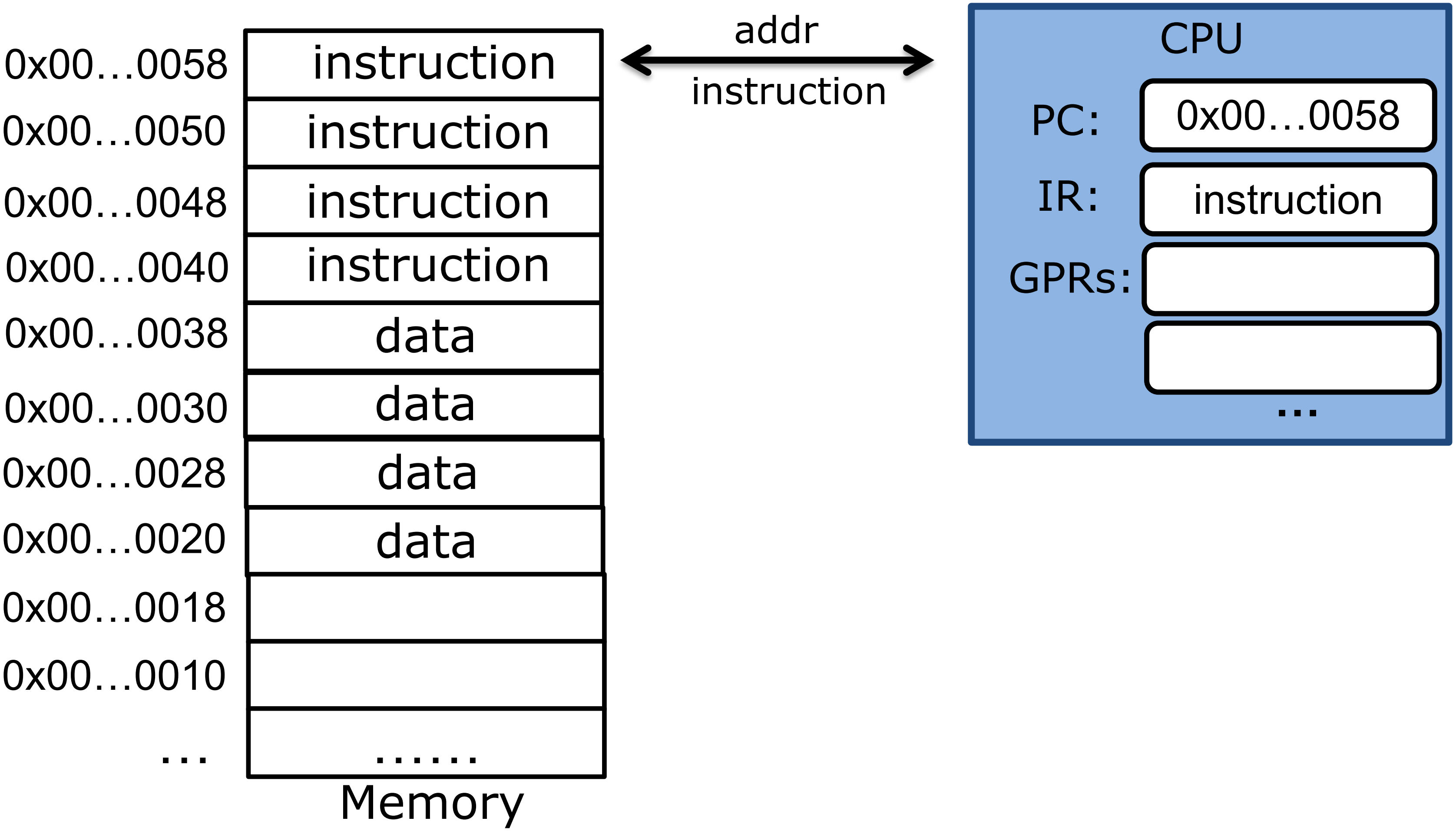
PC: 0x00…0058

IR:

GPRs:

…

Your mental model



0x00…0058	instruction
0x00…0050	instruction
0x00…0048	instruction
0x00…0040	instruction
0x00…0038	data
0x00…0030	data
0x00…0028	data
0x00…0020	data
0x00…0018	
0x00…0010	
…	……

Memory

addr

instruction

CPU

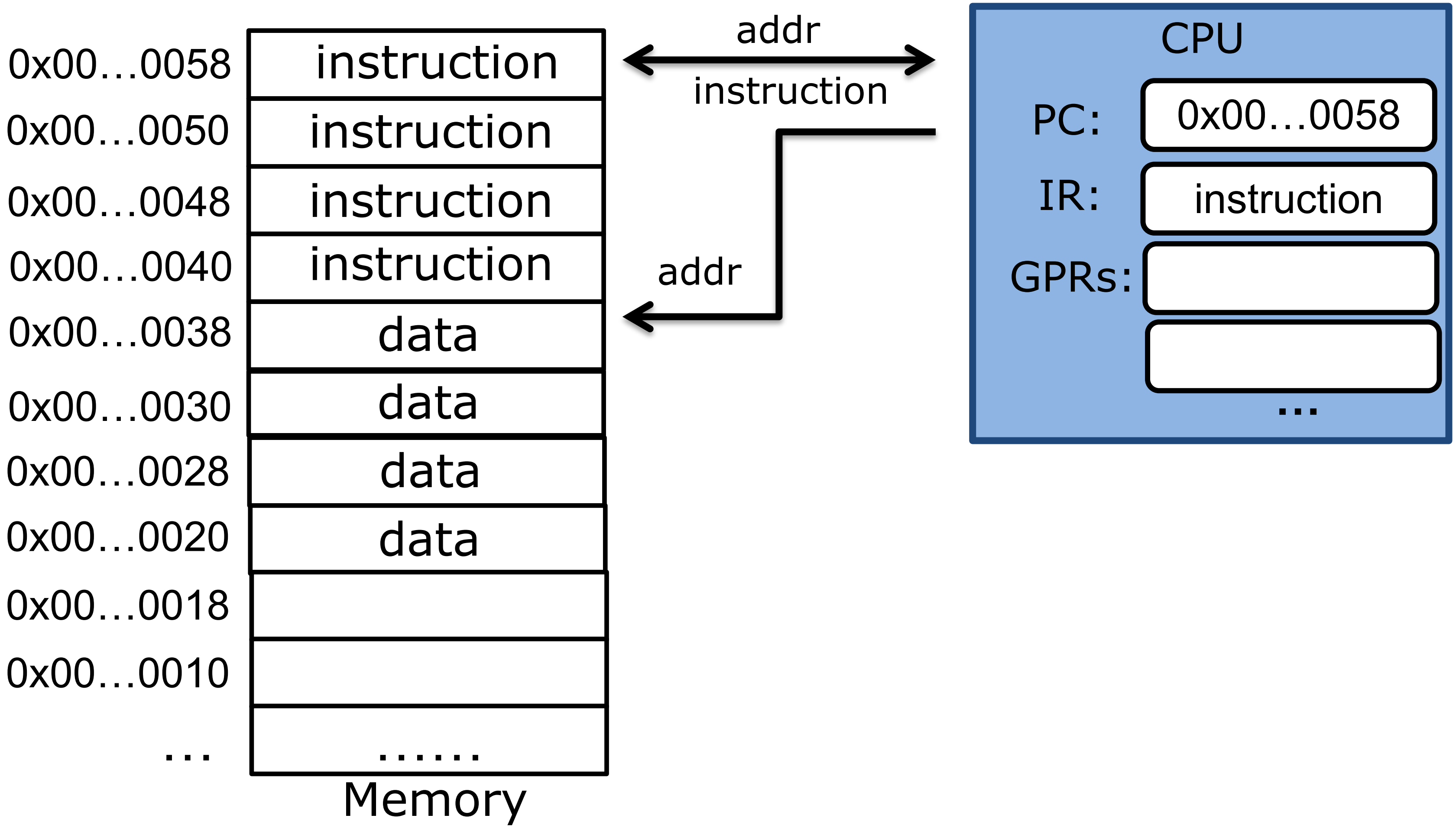
PC: 0x00…0058

IR: instruction

GPRs:

…

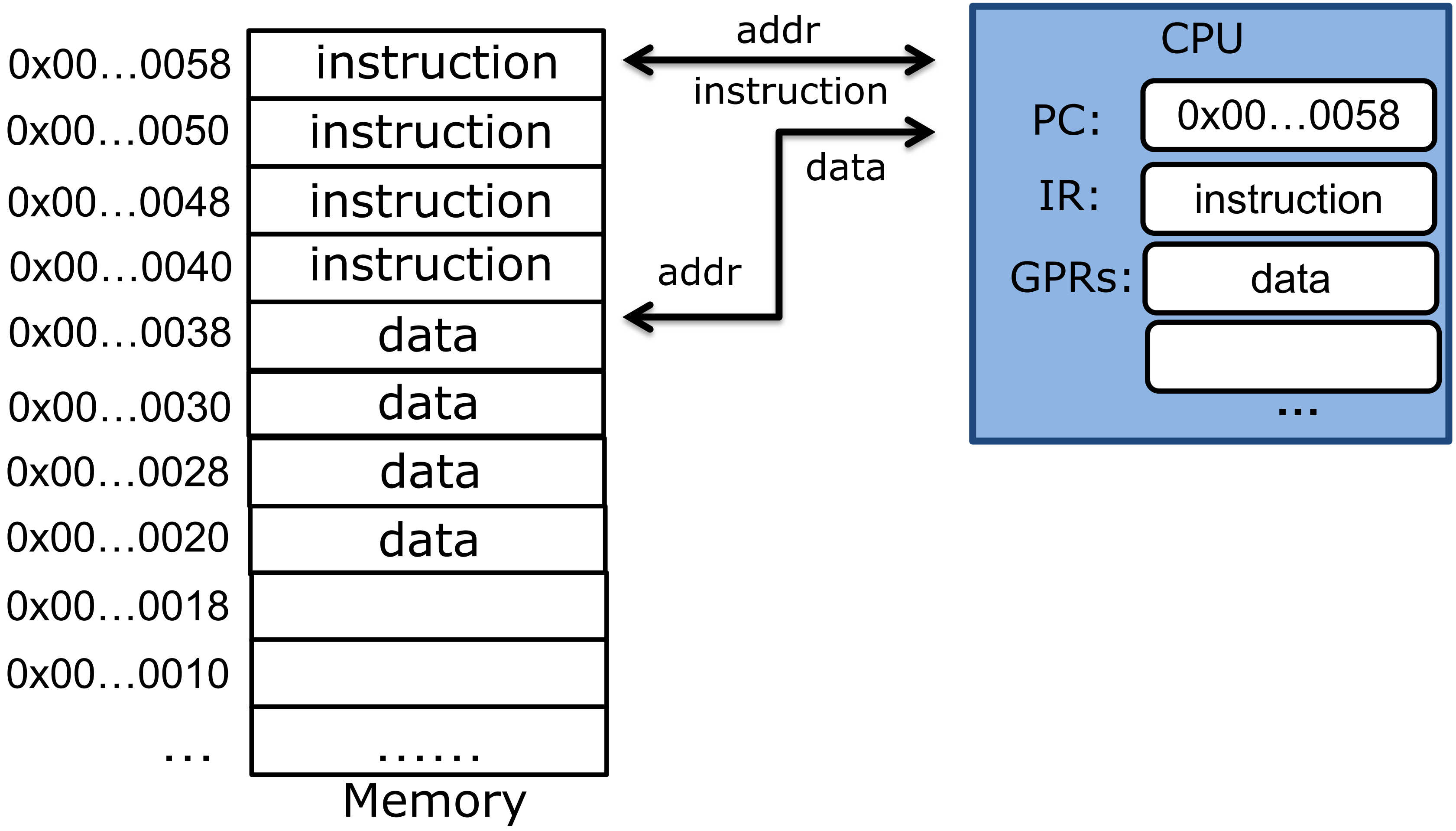
Your mental model



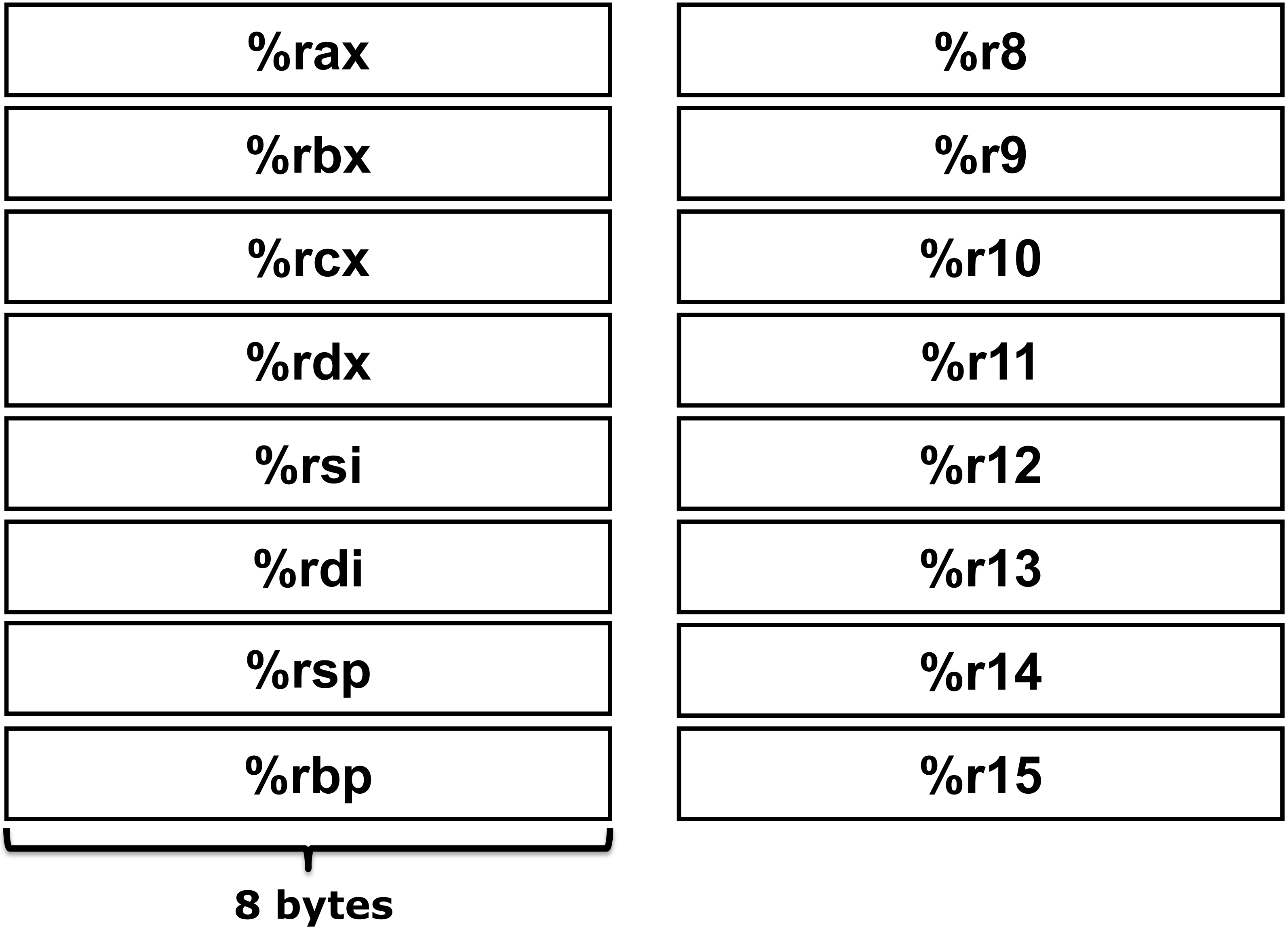
0x00…0058	instruction
0x00…0050	instruction
0x00…0048	instruction
0x00…0040	instruction
0x00…0038	data
0x00…0030	data
0x00…0028	data
0x00…0020	data
0x00…0018	
0x00…0010	
…	……

Memory

Your mental model



General Purpose Registers (intel x86-64)



%rax

%rbx

%rcx

%rdx

%rsi

%rdi

%rsp

%rbp

8 bytes

%r8

%r9

%r10

%r11

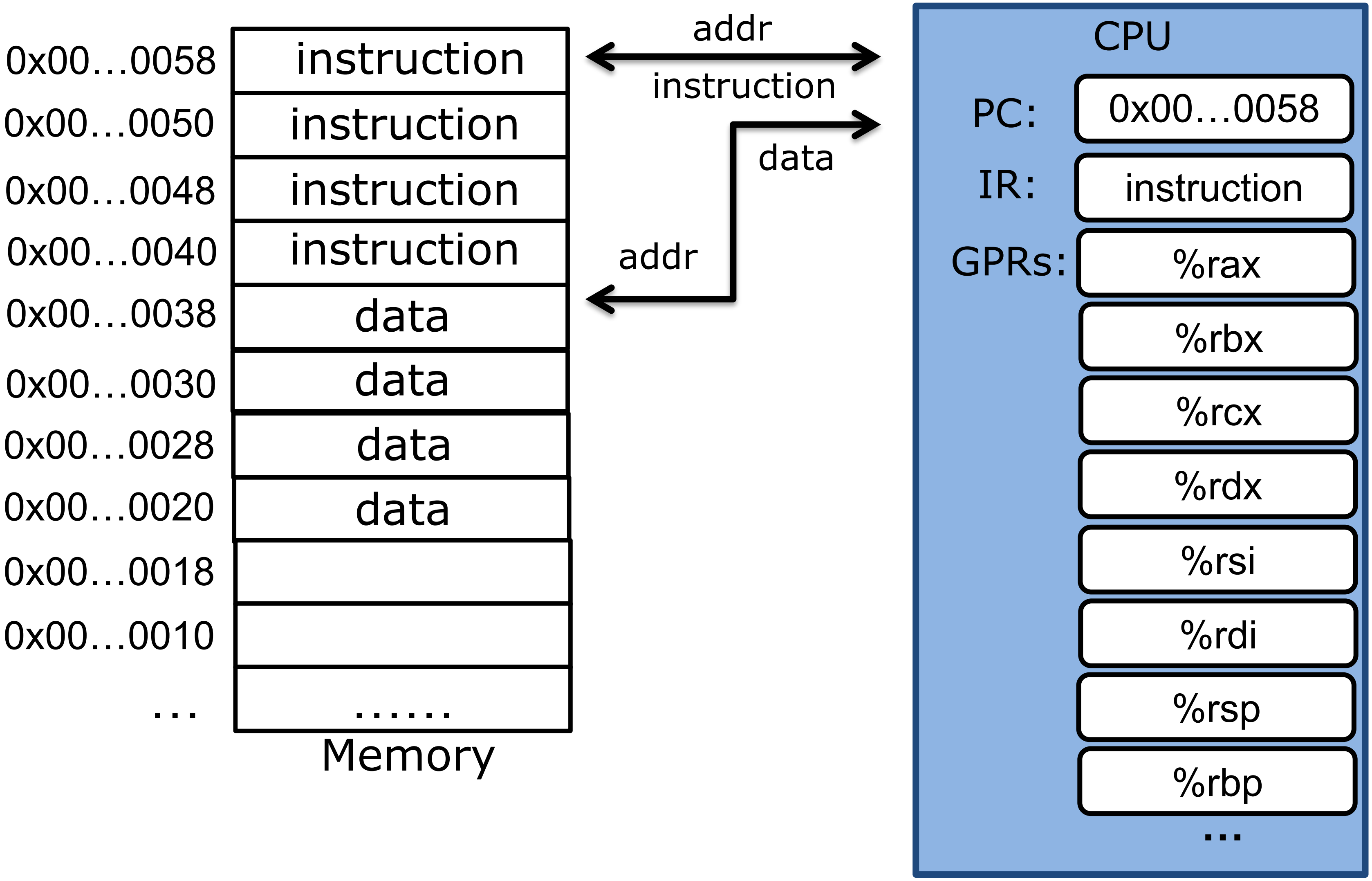
%r12

%r13

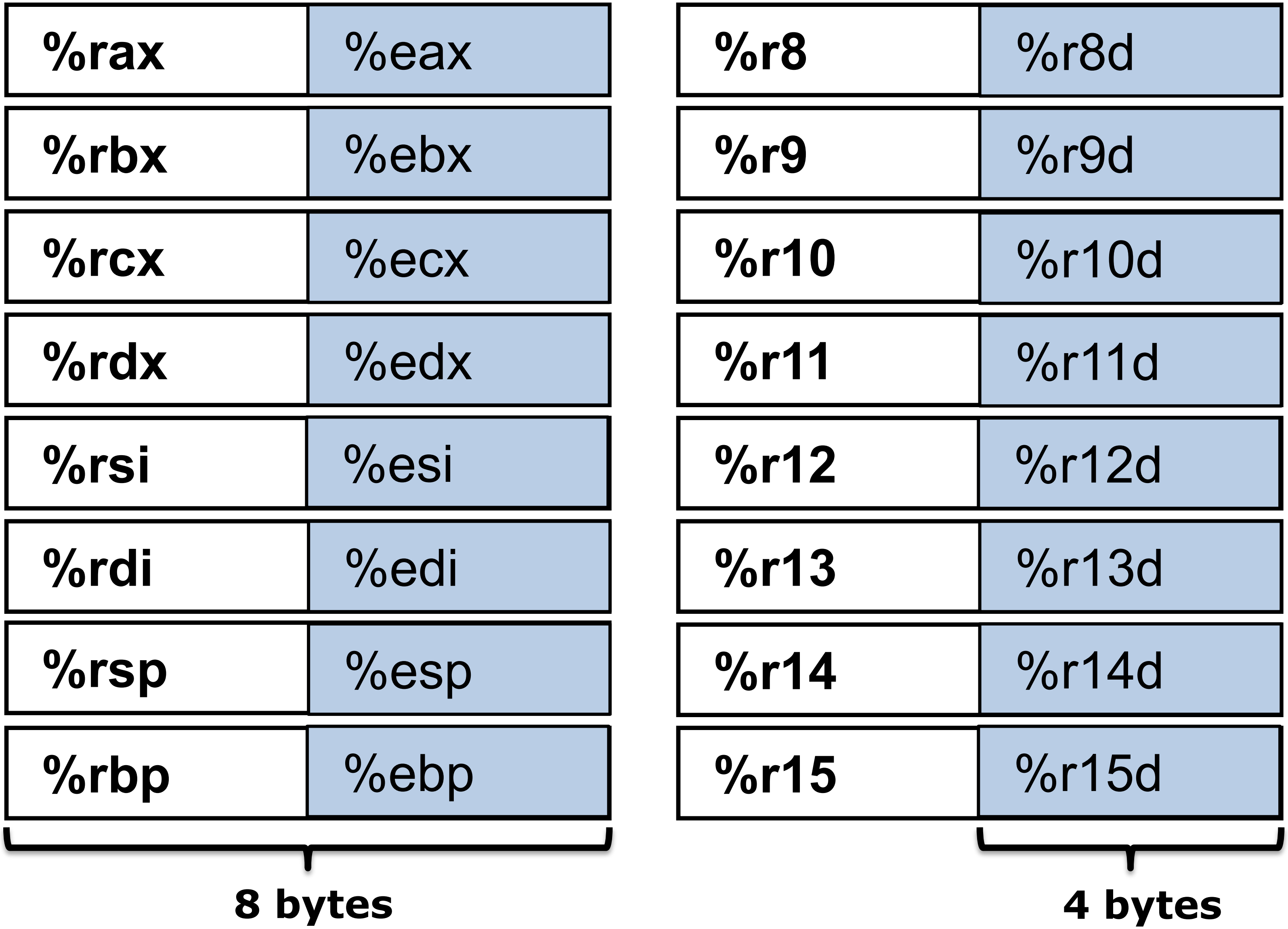
%r14

%r15

Your mental model



General Purpose Registers (intel x86-64)

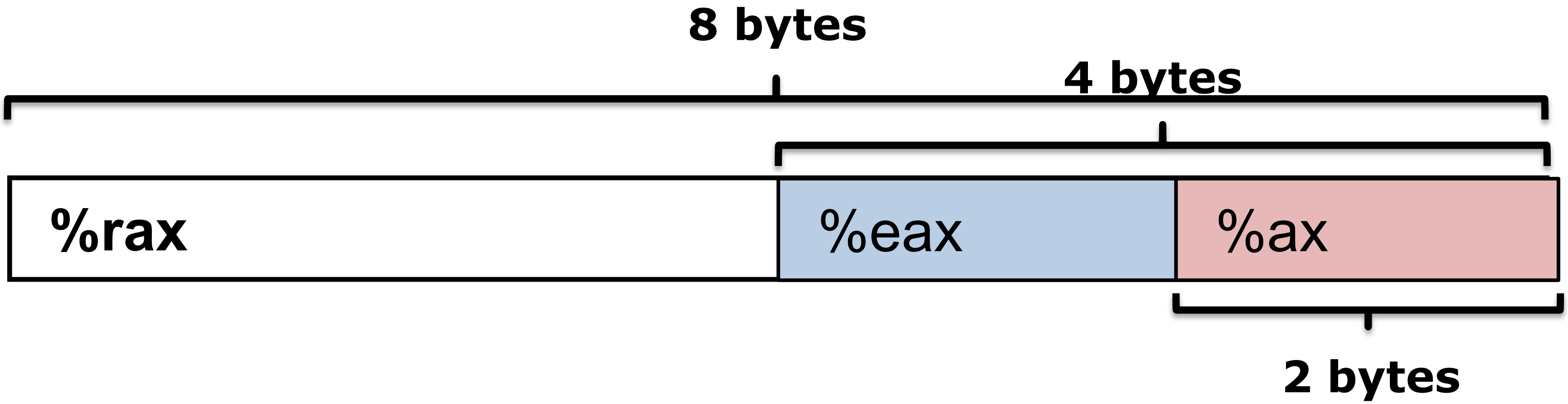


%rax	%eax	%r8	%r8d
%rbx	%ebx	%r9	%r9d
%rcx	%ecx	%r10	%r10d
%rdx	%edx	%r11	%r11d
%rsi	%esi	%r12	%r12d
%rdi	%edi	%r13	%r13d
%rsp	%esp	%r14	%r14d
%rbp	%ebp	%r15	%r15d

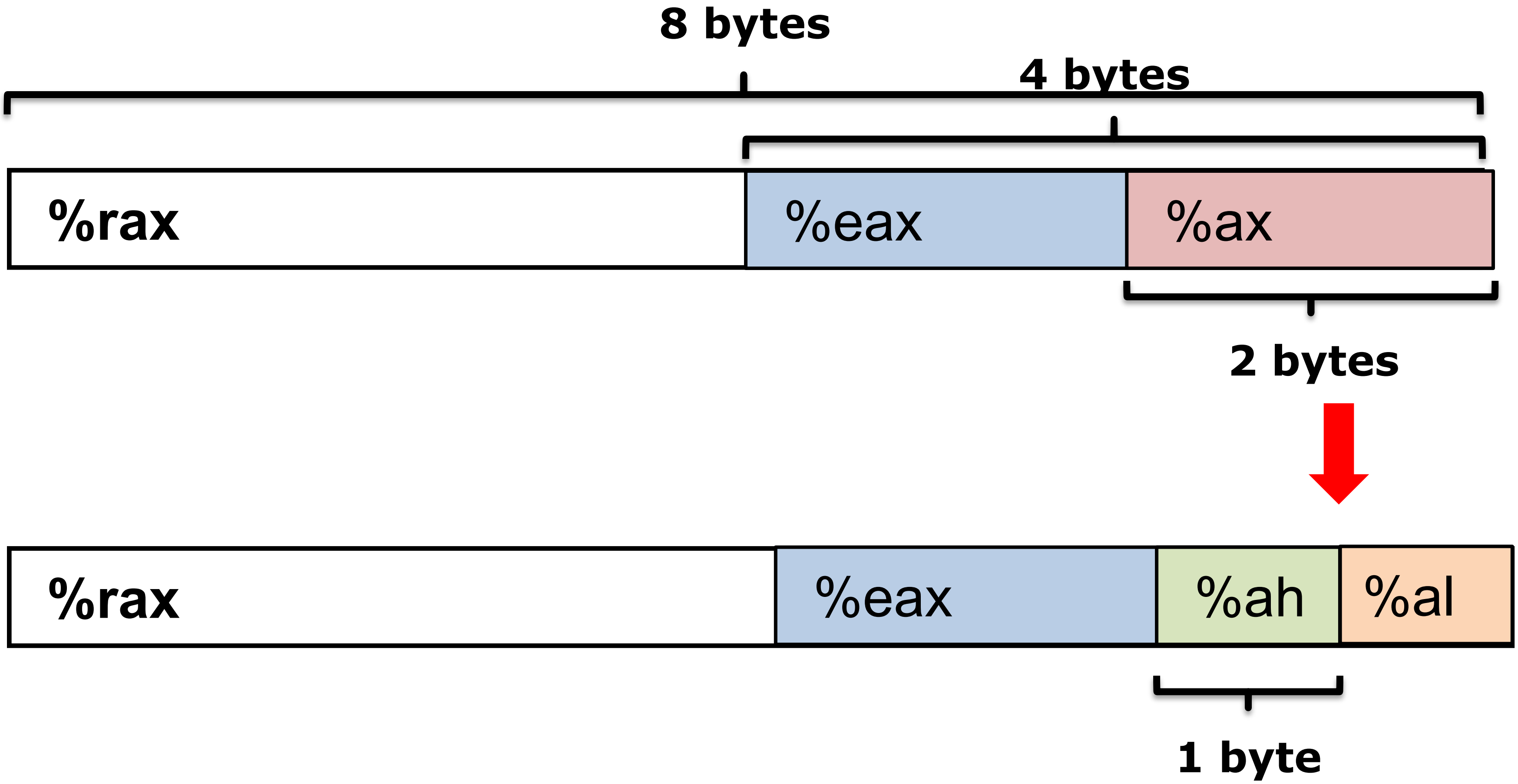
8 bytes

4 bytes

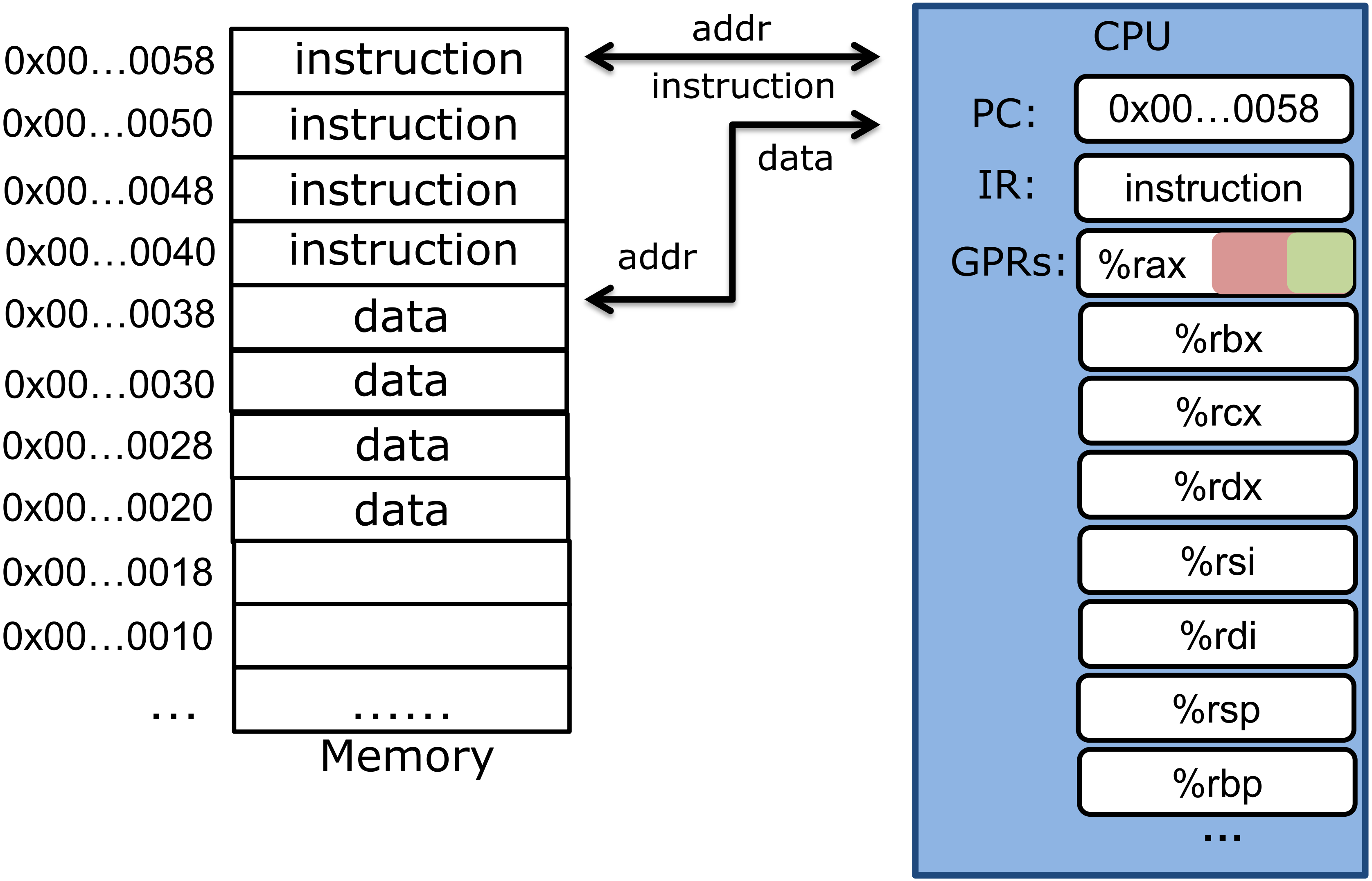
Use %rax and %rbx as examples



Use %rax as an example



Your mental model (intel x86-64)



Steps of execution

1. PC contains the instruction’s address

2. Fetch the instruction into IR

3. Execute the instruction

Instruction Set Architecture (ISA)

An abstract model of a computer

X86_64 is the ISA implemented by Intel/AMD CPUs

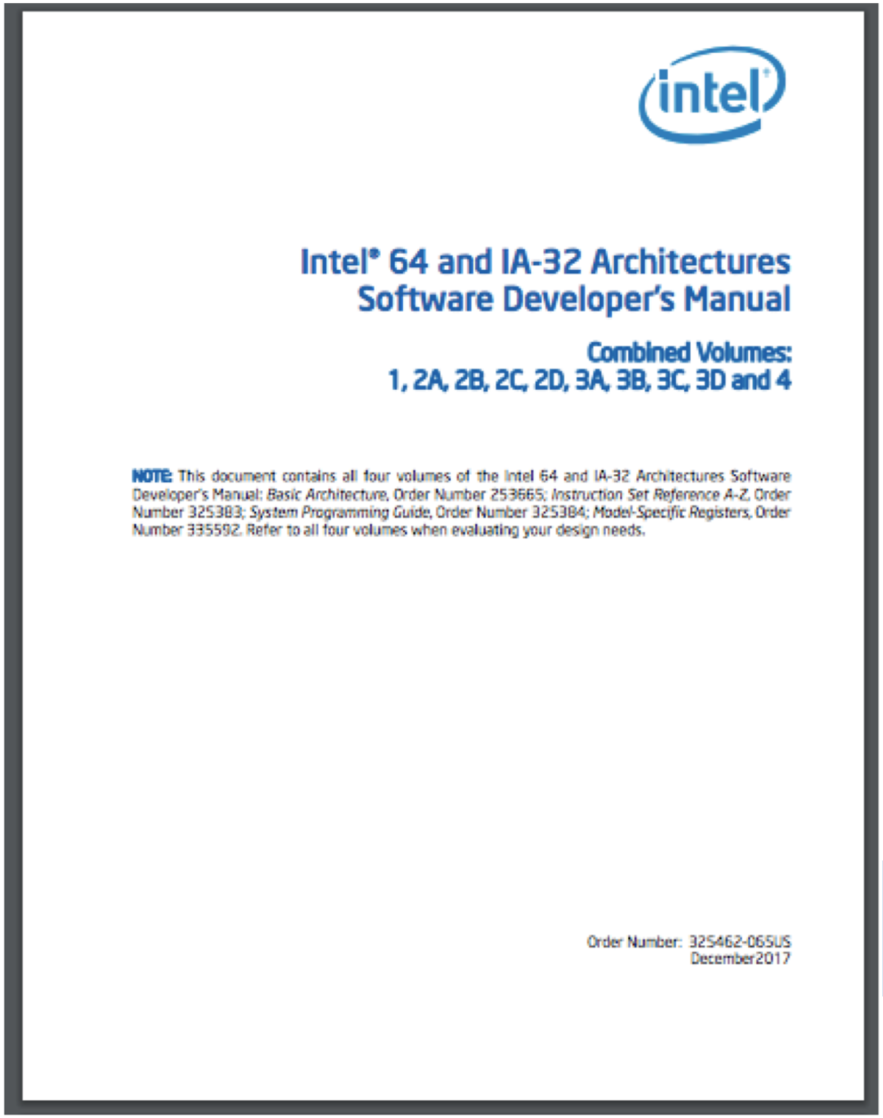
- 64-bit version of x86

this class' focus

ARM is another common ISA

- Phones, tablets, Raspberry Pi

X86 ISA



intel

Intel® 64 and IA-32 Architectures
Software Developer's Manual

Combined Volumes:
1, 2A, 2B, 2C, 2D, 3A, 3B, 3C, 3D and 4

NOTE: This document contains all four volumes of the Intel 64 and IA-32 Architectures Software Developer's Manual: *Basic Architecture*, Order Number 253665; *Instruction Set Reference A-Z*, Order Number 325383; *System Programming Guide*, Order Number 325384; *Model-Specific Registers*, Order Number 335592. Refer to all four volumes when evaluating your design needs.

Order Number: 325462-065US
December2017

A must-read for
compiler and OS writers

https://software.intel.com/en-us/articles/intel-sdm#combined

Moving data

movq *Source*, *Dest*

- Copy a quadword (64 bits) from the source operand (first operand) to the destination operand (second operand).

Moving data

suffix

movq *Source*, *Dest*

- Copy a quadword (64 bits) from the source operand (first operand) to the destination operand (second operand).

Suffix	Name	Size (byte)
B	Byte	1
W	Word	2
L	Long	4
Q	Quadword	8

Why using a size suffix?

movq *Source*, *Dest*

- Copy a quadword (64 bits) from the source operand (first operand) to the destination operand (second operand).
- In the Intel x86 world , a word = 16 bits.
  - 8086 uses 16 bits as a word
  - Support **full backward compatibility**
    - New processor can run the same binary file compiled for older processors

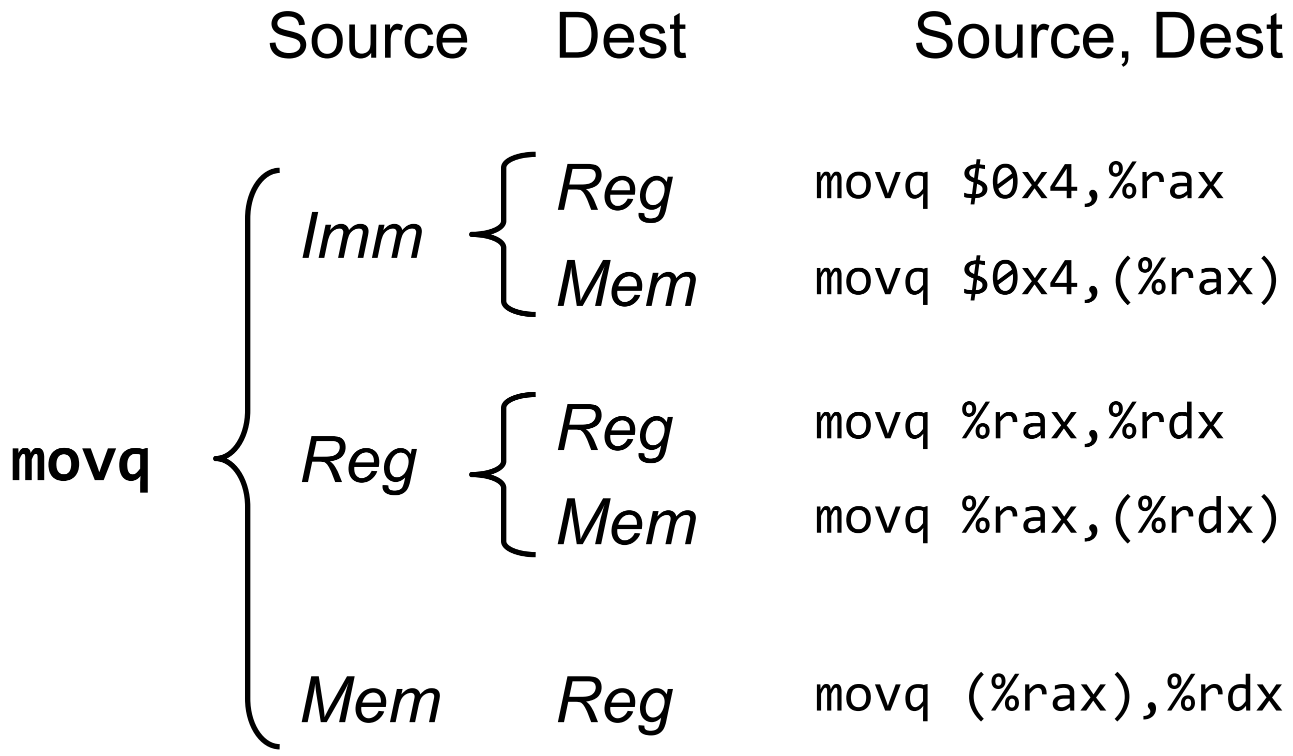
Moving data

movq *Source*, *Dest*

Operand Types

- ***Immediate:*** Constant integer data
  - Prefixed with $
  - Example: **`$0x400`**, **`$-533`**
- ***Register:*** One of general purpose registers
  - Example: **`%rax, %rsi`**
- ***Memory:*** 8 consecutive bytes of memory
  - Indexed by register with various “address modes”
  - Simplest example: **`(%rax)`**

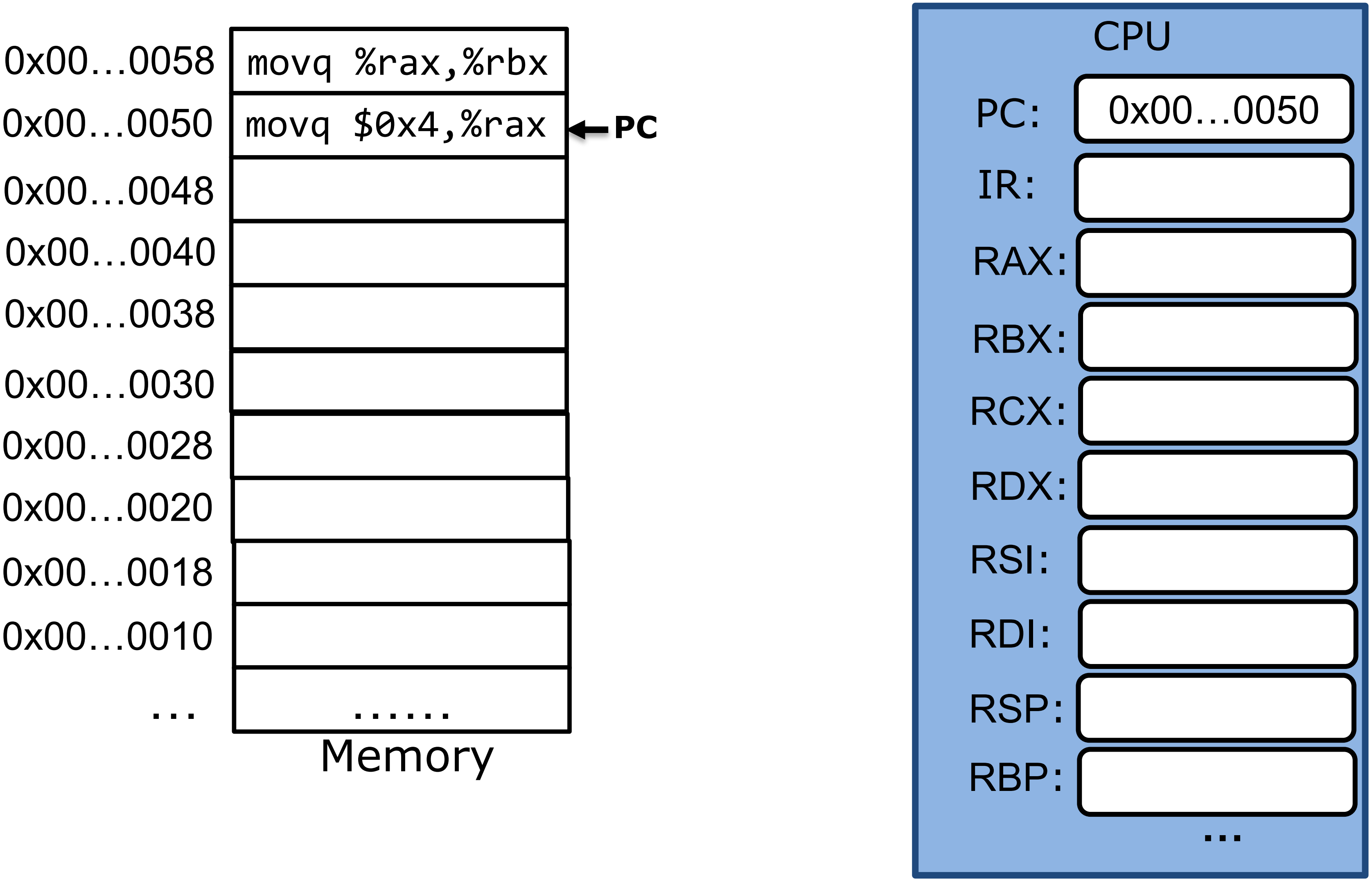
movq Operand combinations



	Source	Dest	Source, Dest
movq	*Imm*	*Reg*	`movq $0x4,%rax`
		Mem	`movq $0x4,(%rax)`
	Reg	*Reg*	`movq %rax,%rdx`
		Mem	`movq %rax,(%rdx)`
	Mem	*Reg*	`movq (%rax),%rdx`

1. Immediate can only be *Source*
2. Cannot do memory-memory transfer with a single instruction

movq *Imm, Reg*



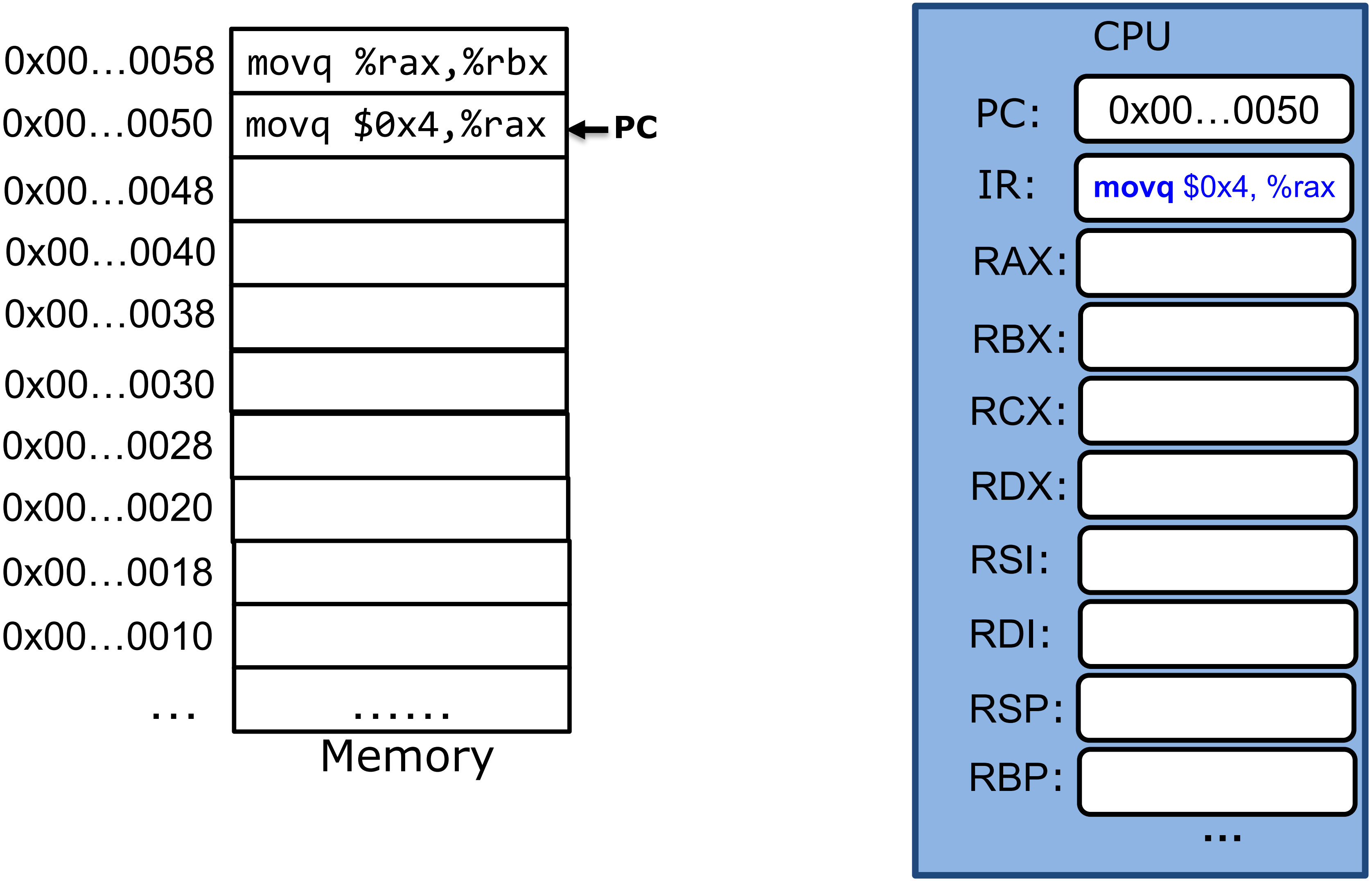
0x00…0058	`movq %rax,%rbx`
0x00…0050	`movq $0x4,%rax` ← **PC**
0x00…0048	
0x00…0040	
0x00…0038	
0x00…0030	
0x00…0028	
0x00…0020	
0x00…0018	
0x00…0010	
…	……

Memory

CPU

PC:	0x00…0050
IR:	
RAX:	
RBX:	
RCX:	
RDX:	
RSI:	
RDI:	
RSP:	
RBP:	
	…

movq *Imm, Reg*



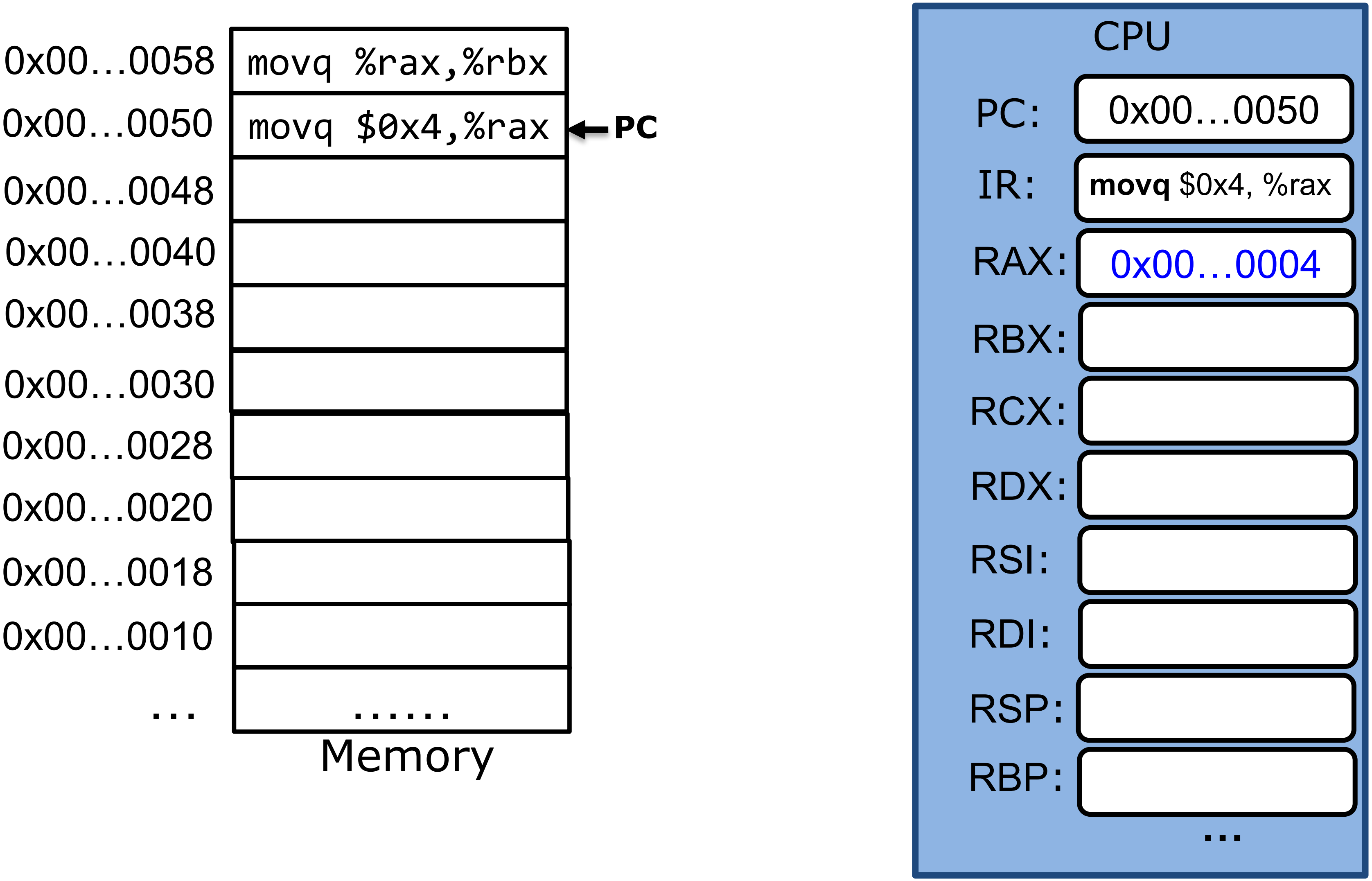
0x00…0058	movq %rax,%rbx
0x00…0050	movq $0x4,%rax ← PC
0x00…0048	
0x00…0040	
0x00…0038	
0x00…0030	
0x00…0028	
0x00…0020	
0x00…0018	
0x00…0010	
…	……

Memory

CPU

PC:	0x00…0050
IR:	**movq** $0x4, %rax
RAX:	
RBX:	
RCX:	
RDX:	
RSI:	
RDI:	
RSP:	
RBP:	
	…

movq *Imm, Reg*



0x00…0058	`movq %rax,%rbx`
0x00…0050	`movq $0x4,%rax` ← **PC**
0x00…0048	
0x00…0040	
0x00…0038	
0x00…0030	
0x00…0028	
0x00…0020	
0x00…0018	
0x00…0010	
…	……

Memory

CPU

PC:	0x00…0050
IR:	**movq** $0x4, %rax
RAX:	0x00…0004
RBX:	
RCX:	
RDX:	
RSI:	
RDI:	
RSP:	
RBP:	
	…

Steps

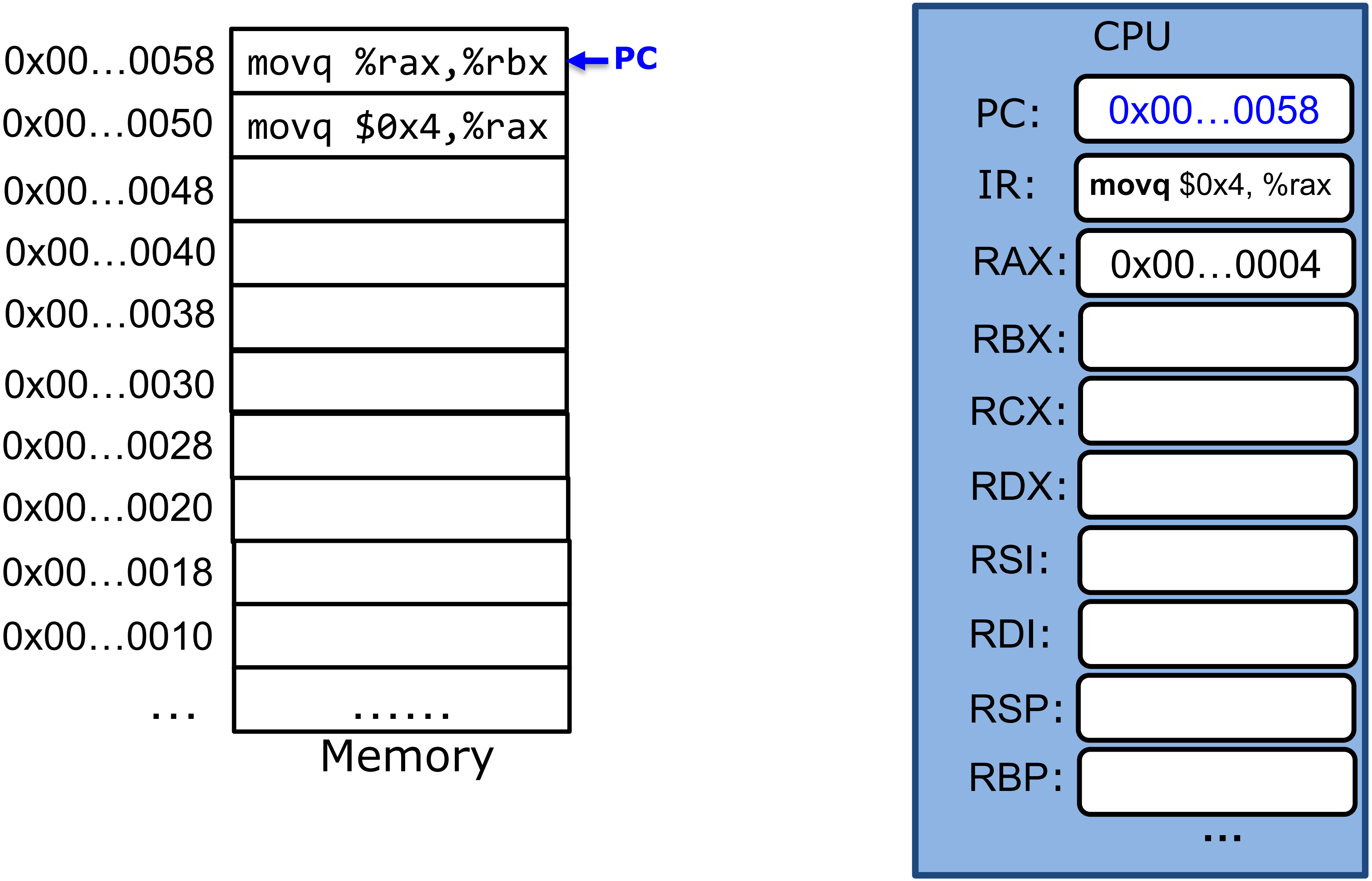
1. PC contains the instruction’s address

2. Load the instruction into IR

3. Execute the instruction

4. CPU automatically updates PC after current instruction finishes (is retired).

movq *Reg, Reg*



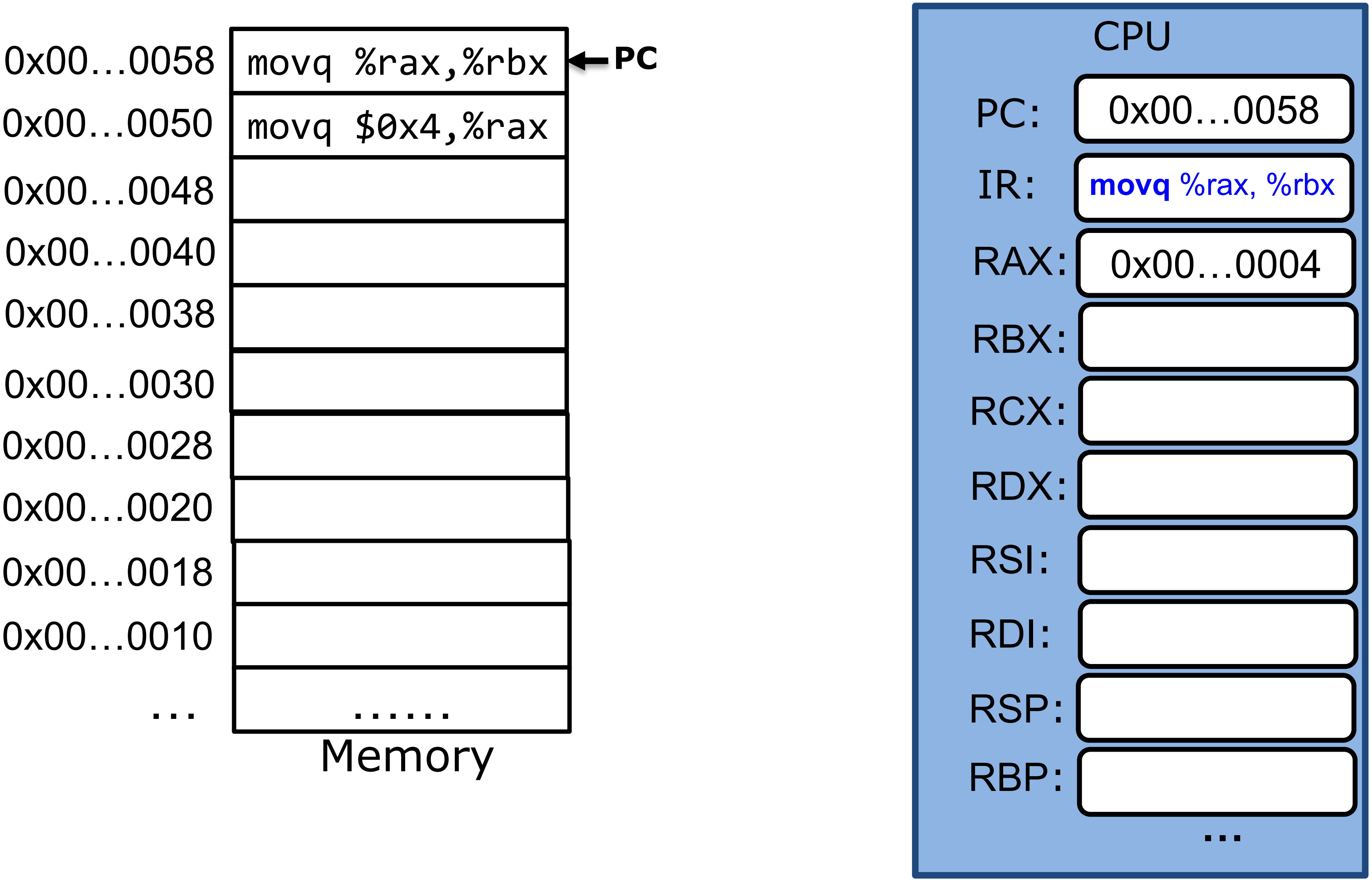
0x00…0058	`movq %rax,%rbx` ← PC
0x00…0050	`movq $0x4,%rax`
0x00…0048	
0x00…0040	
0x00…0038	
0x00…0030	
0x00…0028	
0x00…0020	
0x00…0018	
0x00…0010	
…	……

Memory

CPU

PC:	0x00…0058
IR:	**movq** $0x4, %rax
RAX:	0x00…0004
RBX:	
RCX:	
RDX:	
RSI:	
RDI:	
RSP:	
RBP:	
	…

movq *Reg, Reg*



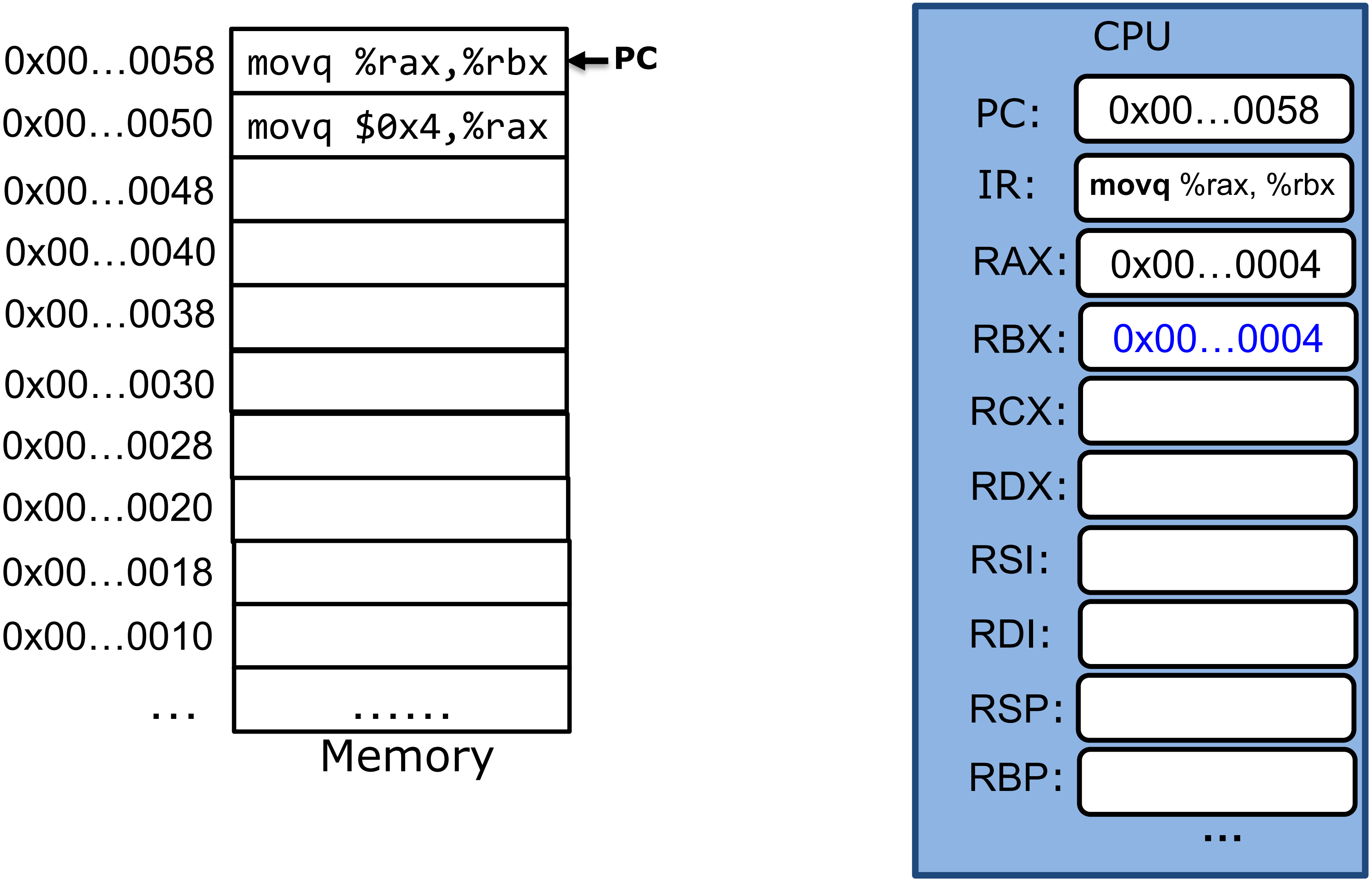
0x00…0058	movq %rax,%rbx	← PC
0x00…0050	movq $0x4,%rax	
0x00…0048		
0x00…0040		
0x00…0038		
0x00…0030		
0x00…0028		
0x00…0020		
0x00…0018		
0x00…0010		
…	……	

Memory

CPU

PC:	0x00…0058
IR:	**movq** %rax, %rbx
RAX:	0x00…0004
RBX:	
RCX:	
RDX:	
RSI:	
RDI:	
RSP:	
RBP:	
	…

movq *Reg, Reg*



0x00…0058	movq %rax,%rbx	← PC
0x00…0050	movq $0x4,%rax	
0x00…0048		
0x00…0040		
0x00…0038		
0x00…0030		
0x00…0028		
0x00…0020		
0x00…0018		
0x00…0010		
…	……	

Memory

CPU

PC:	0x00…0058
IR:	**movq** %rax, %rbx
RAX:	0x00…0004
RBX:	0x00…0004
RCX:	
RDX:	
RSI:	
RDI:	
RSP:	
RBP:	
	…

movq *Mem, Reg*

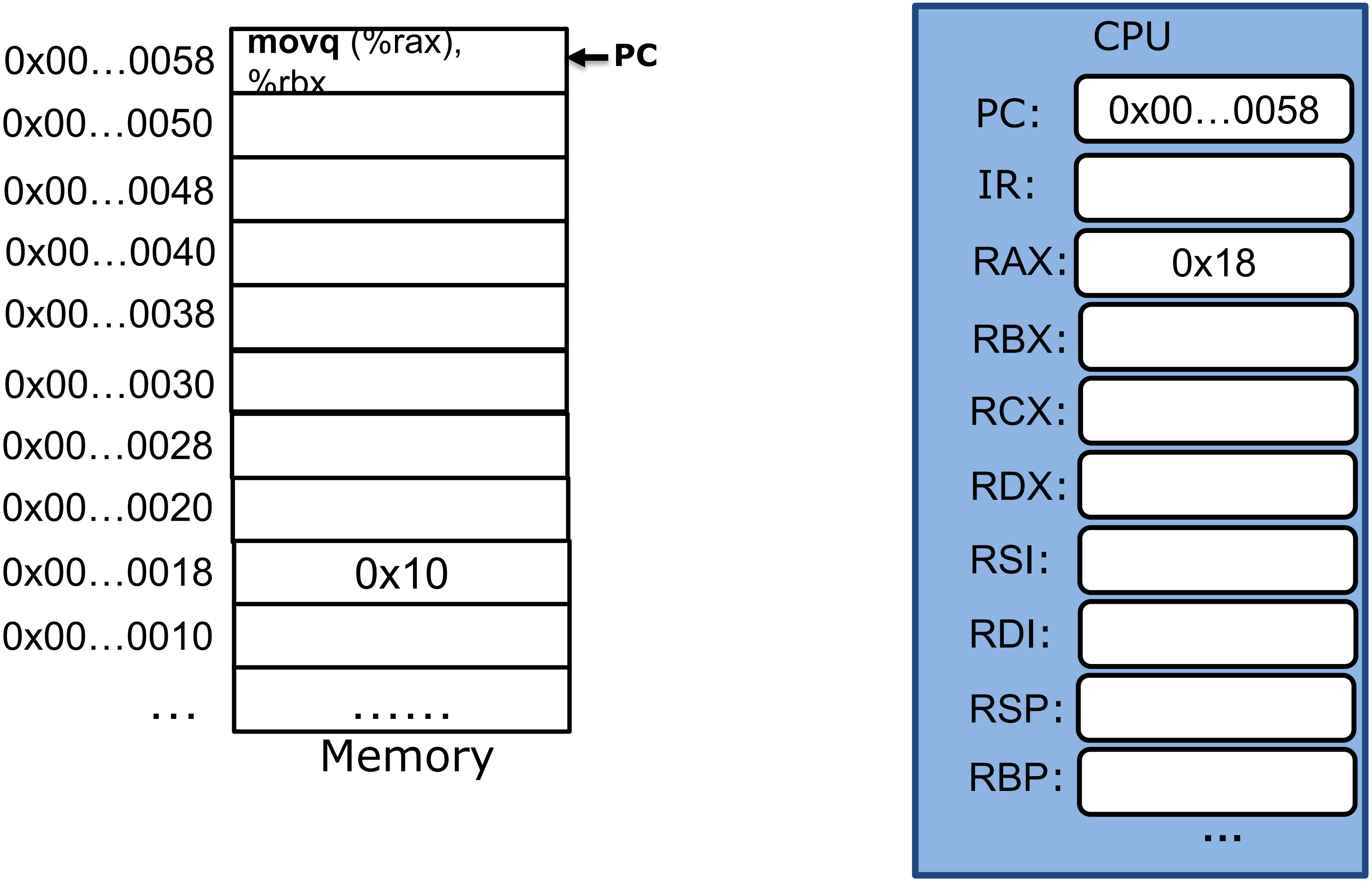
How to represent a “memory” operand?

Direct addressing: use registers to index the memory

(Register)

- The content of the register specifies memory address
- movq (%rax), %rbx

movq *(%rax), %rbx*



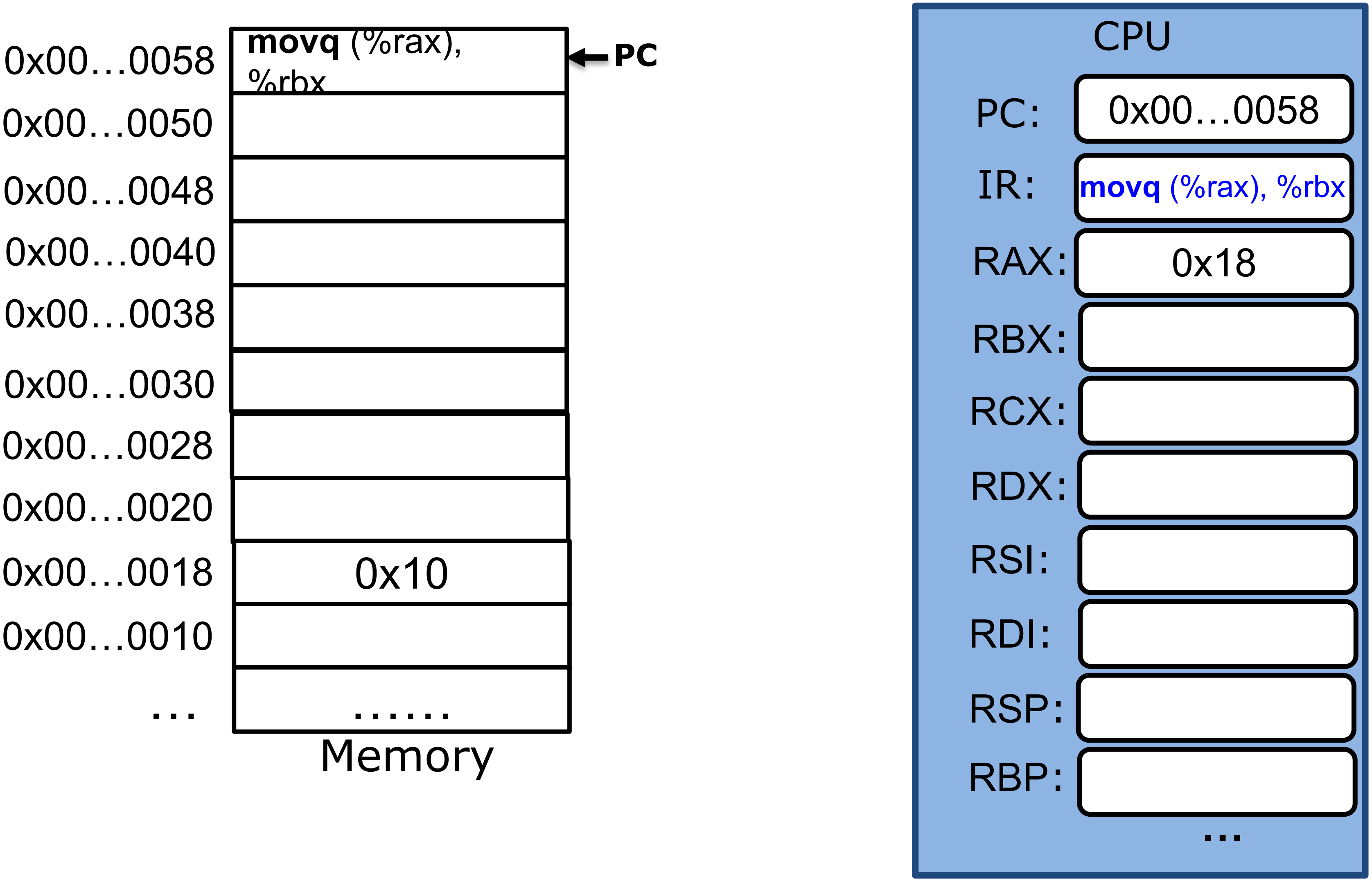
0x00…0058	**movq** (%rax), %rbx	← **PC**
0x00…0050		
0x00…0048		
0x00…0040		
0x00…0038		
0x00…0030		
0x00…0028		
0x00…0020		
0x00…0018	0x10	
0x00…0010		
…	……	

Memory

CPU

PC:	0x00…0058
IR:	
RAX:	0x18
RBX:	
RCX:	
RDX:	
RSI:	
RDI:	
RSP:	
RBP:	
	…

movq *(%rax), %rbx*



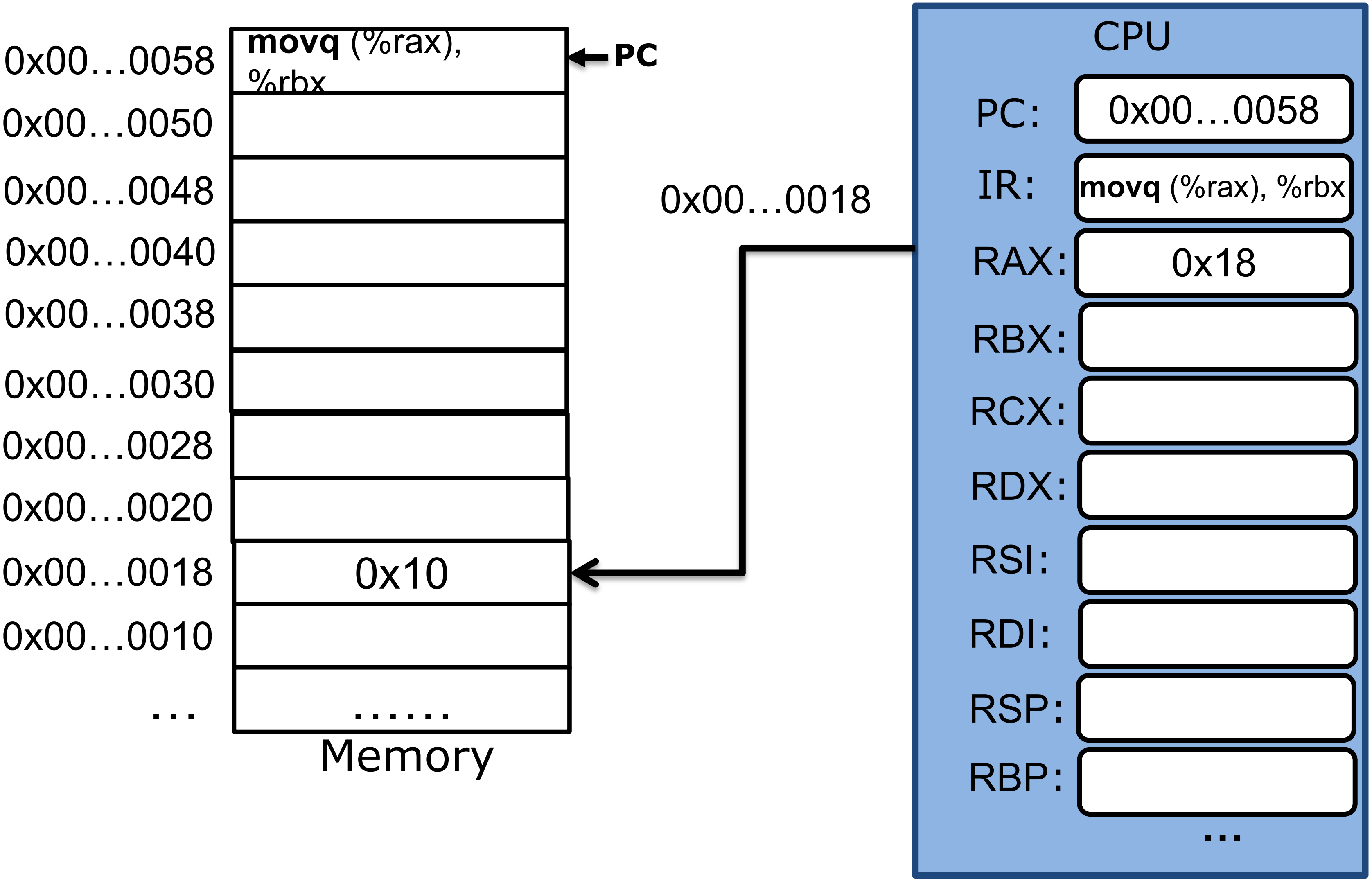
0x00…0058	**movq** (%rax), %rbx ← **PC**
0x00…0050	
0x00…0048	
0x00…0040	
0x00…0038	
0x00…0030	
0x00…0028	
0x00…0020	
0x00…0018	0x10
0x00…0010	
…	……

Memory

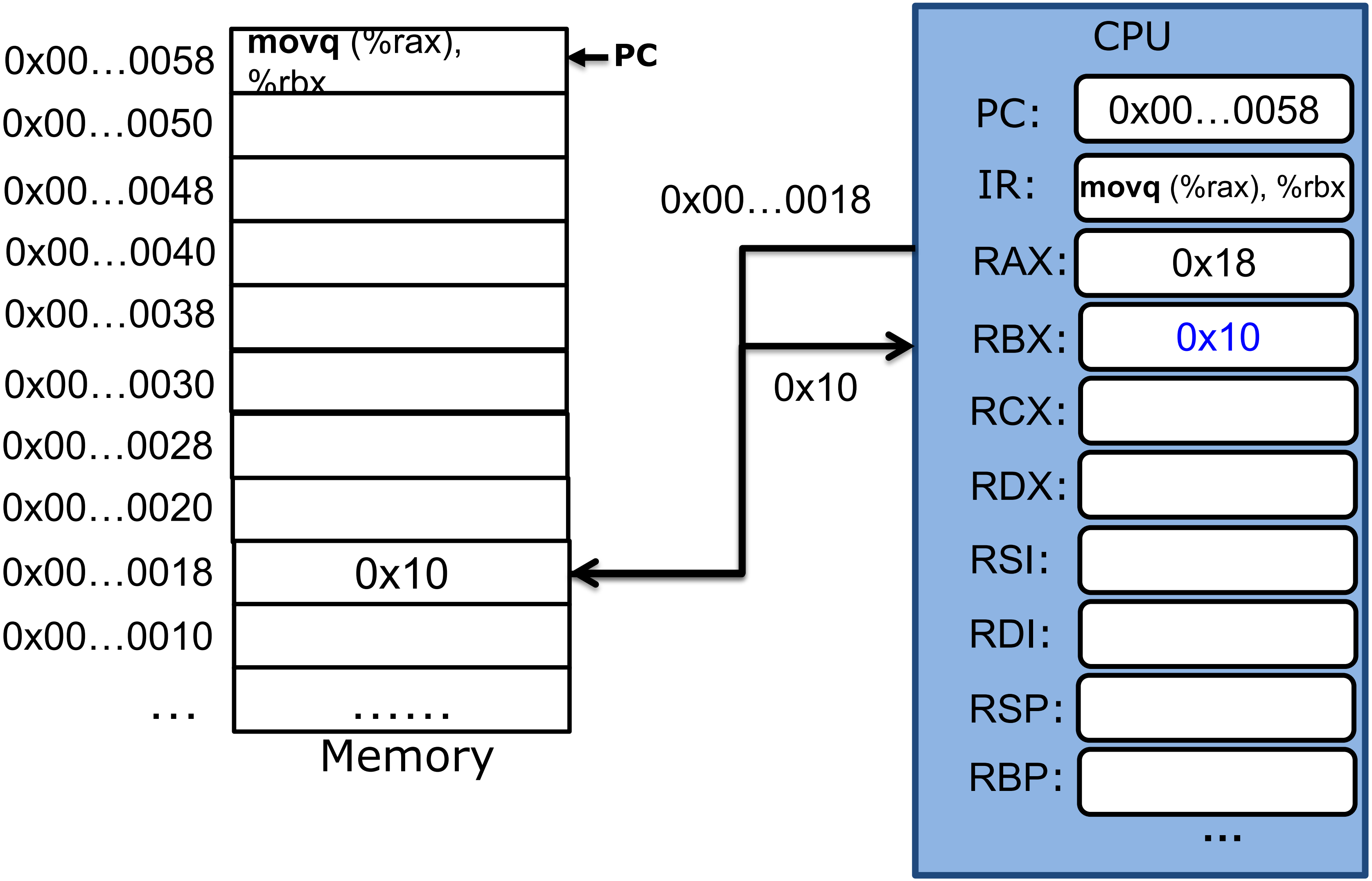
CPU

PC:	0x00…0058
IR:	**movq** (%rax), %rbx
RAX:	0x18
RBX:	
RCX:	
RDX:	
RSI:	
RDI:	
RSP:	
RBP:	
	…

movq *(%rax), %rbx*


0x00…0058
movq (%rax), %rbx
PC
0x00…0050
0x00…0048
0x00…0040
0x00…0038
0x00…0030
0x00…0028
0x00…0020
0x00…0018
0x10
0x00…0010
…
……
Memory
0x00…0018
CPU
PC:
0x00…0058
IR:
movq (%rax), %rbx
RAX:
0x18
RBX:
RCX:
RDX:
RSI:
RDI:
RSP:
RBP:
…

movq *(%rax), %rbx*


0x00…0058
movq (%rax), %rbx
← **PC**
0x00…0050
0x00…0048
0x00…0040
0x00…0038
0x00…0030
0x00…0028
0x00…0020
0x00…0018
0x10
0x00…0010
…
……
Memory
0x00…0018
0x10
CPU
PC:
0x00…0058
IR:
movq (%rax), %rbx
RAX:
0x18
RBX:
0x10
RCX:
RDX:
RSI:
RDI:
RSP:
RBP:
…

swap function

```
void
swap(long *a, long* b) {

    long tmp = *a;
    *a = *b;
    *b = tmp;

}
```

GCC –S –O3 swap.c

```
swap:
```

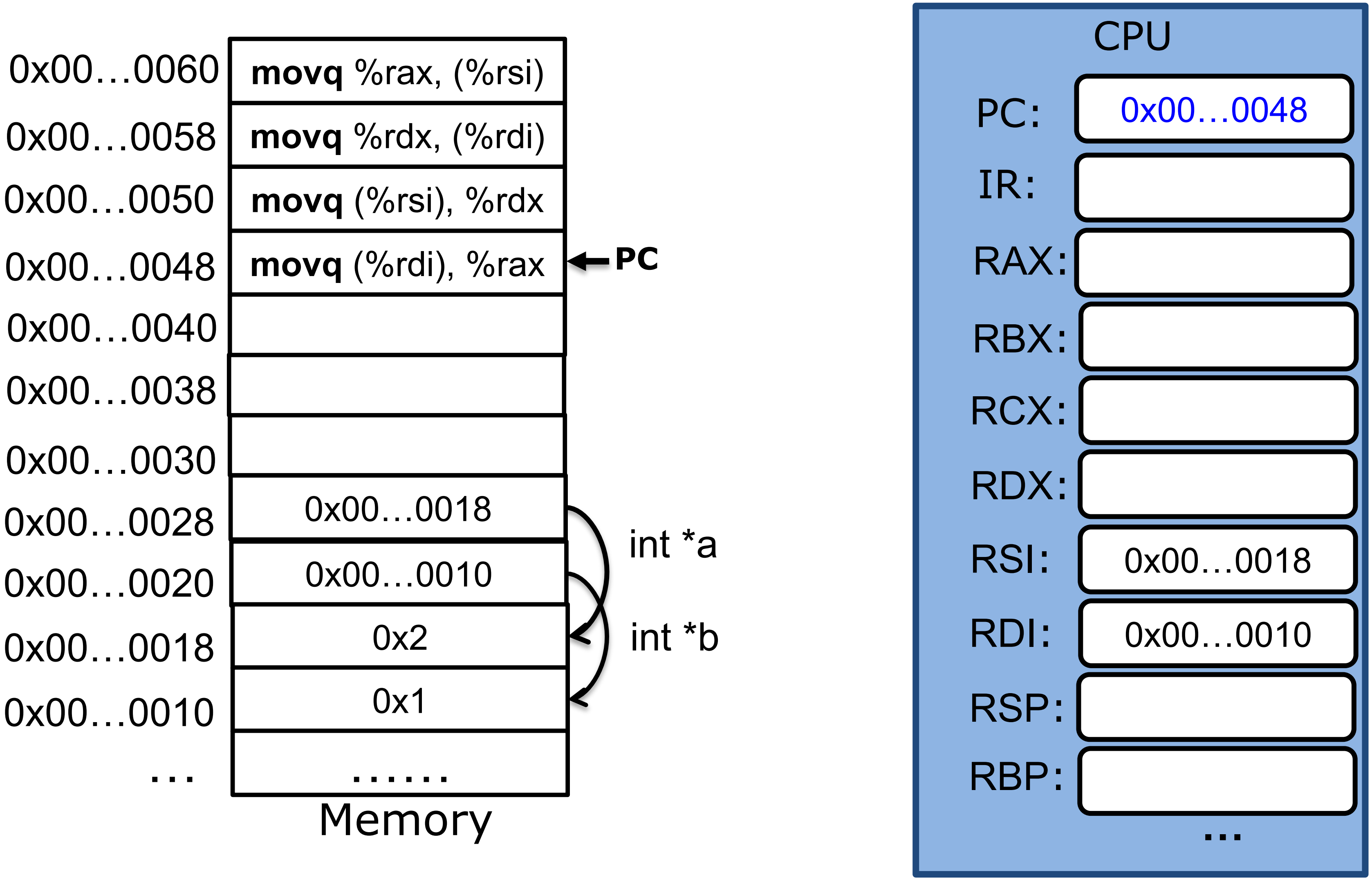
swap function

```
void
swap(long *a, long* b) {

   long tmp = *a;
   *a = *b;
   *b = tmp;

}
```

GCC –S –O3 swap.c

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

swap func

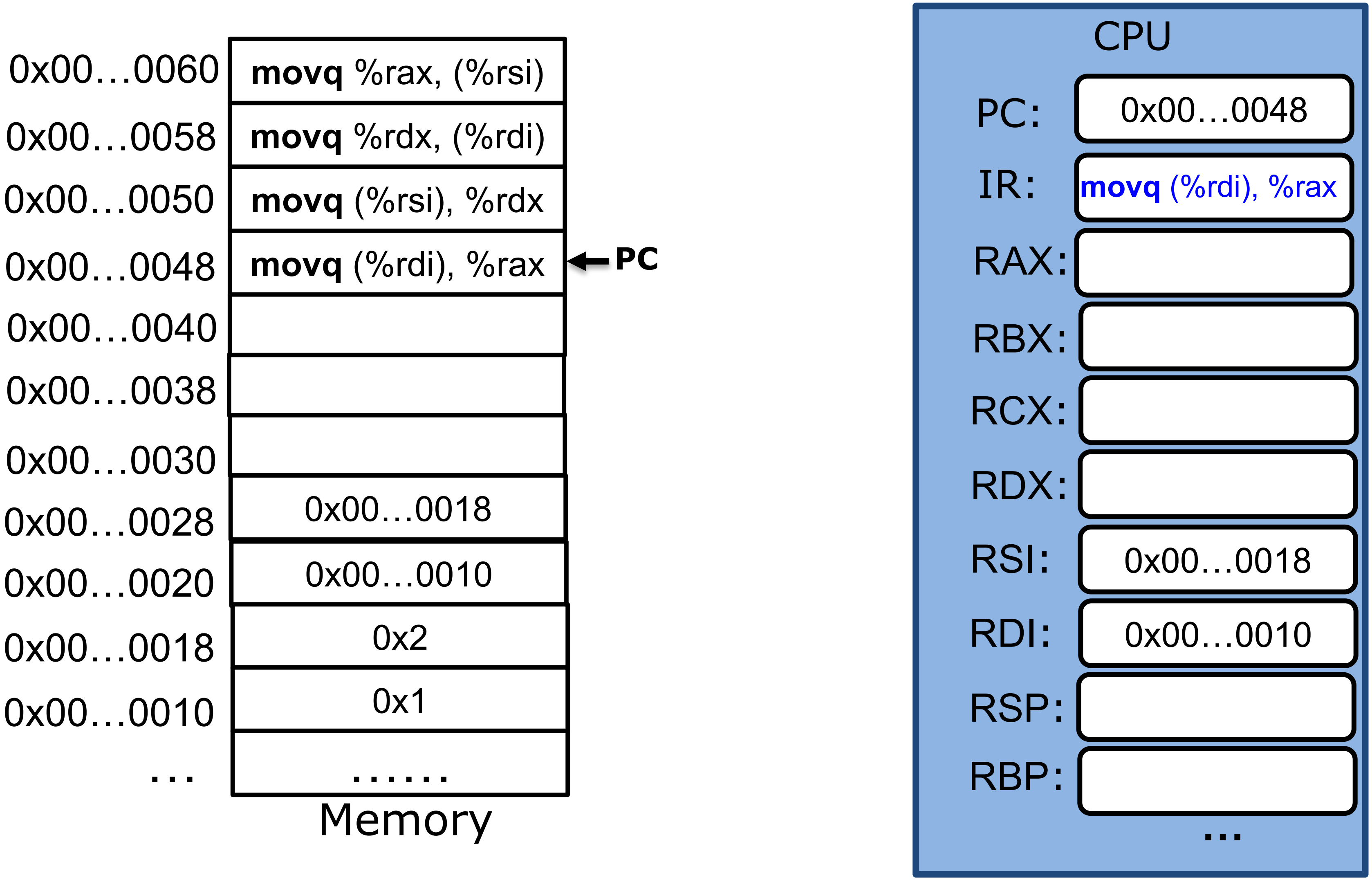


0x00…0060	**movq** %rax, (%rsi)
0x00…0058	**movq** %rdx, (%rdi)
0x00…0050	**movq** (%rsi), %rdx
0x00…0048	**movq** (%rdi), %rax
0x00…0040	
0x00…0038	
0x00…0030	
0x00…0028	0x00…0018
0x00…0020	0x00…0010
0x00…0018	0x2
0x00…0010	0x1
…	……

CPU

PC:	0x00…0048
IR:	
RAX:	
RBX:	
RCX:	
RDX:	
RSI:	0x00…0018
RDI:	0x00…0010
RSP:	
RBP:	
	…

swap func



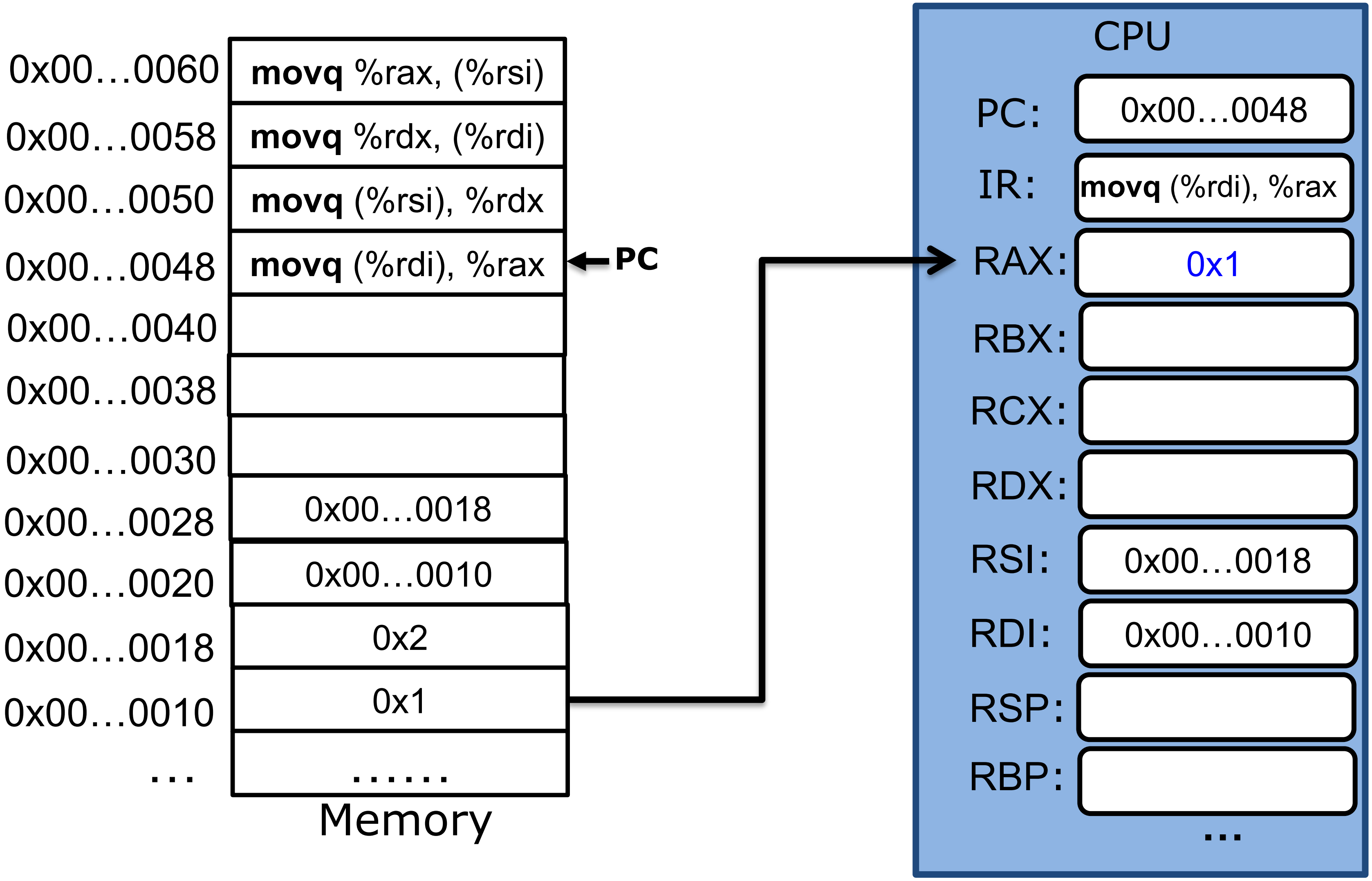
0x00…0060	**movq** %rax, (%rsi)
0x00…0058	**movq** %rdx, (%rdi)
0x00…0050	**movq** (%rsi), %rdx
0x00…0048	**movq** (%rdi), %rax ← **PC**
0x00…0040	
0x00…0038	
0x00…0030	
0x00…0028	0x00…0018
0x00…0020	0x00…0010
0x00…0018	0x2
0x00…0010	0x1
…	……

Memory

CPU

PC:	0x00…0048
IR:	**movq** (%rdi), %rax
RAX:	
RBX:	
RCX:	
RDX:	
RSI:	0x00…0018
RDI:	0x00…0010
RSP:	
RBP:	
	…

swap func



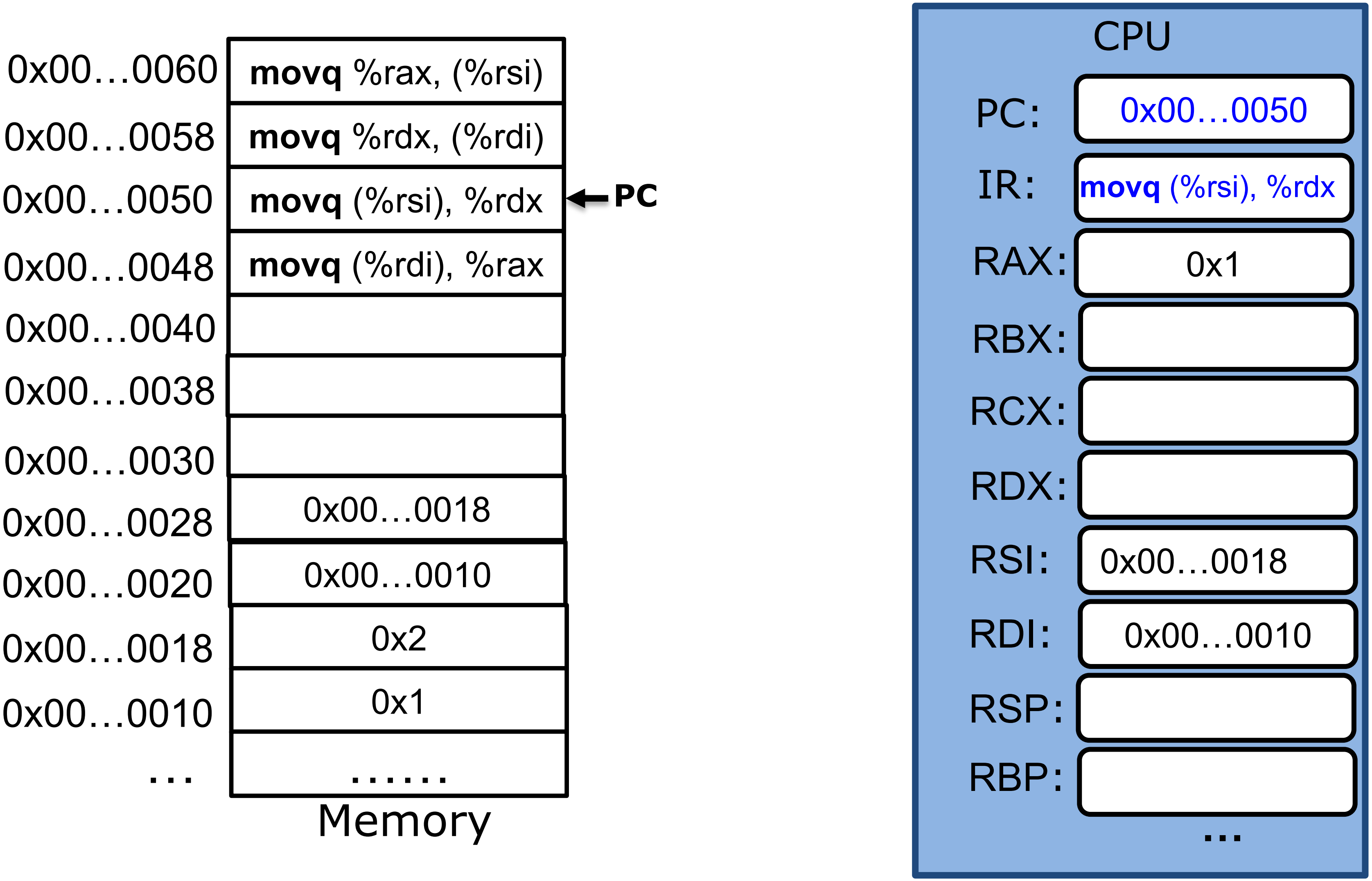
0x00…0060	**movq** %rax, (%rsi)
0x00…0058	**movq** %rdx, (%rdi)
0x00…0050	**movq** (%rsi), %rdx
0x00…0048	**movq** (%rdi), %rax ← **PC**
0x00…0040	
0x00…0038	
0x00…0030	
0x00…0028	0x00…0018
0x00…0020	0x00…0010
0x00…0018	0x2
0x00…0010	0x1
…	……

Memory

CPU

PC:	0x00…0048
IR:	**movq** (%rdi), %rax
RAX:	0x1
RBX:	
RCX:	
RDX:	
RSI:	0x00…0018
RDI:	0x00…0010
RSP:	
RBP:	
	…

swap func



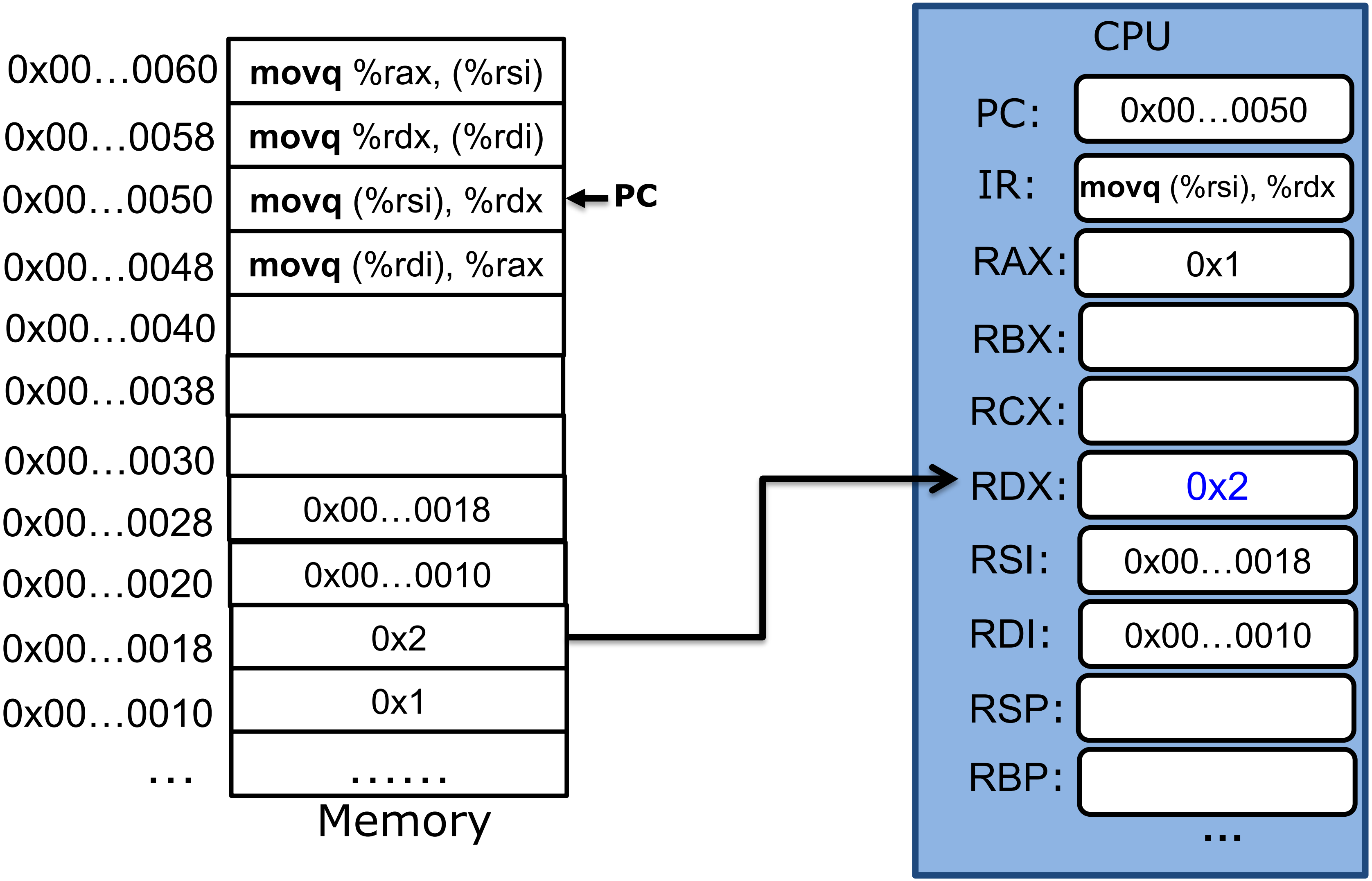
0x00…0060	**movq** %rax, (%rsi)
0x00…0058	**movq** %rdx, (%rdi)
0x00…0050	**movq** (%rsi), %rdx ← **PC**
0x00…0048	**movq** (%rdi), %rax
0x00…0040	
0x00…0038	
0x00…0030	
0x00…0028	0x00…0018
0x00…0020	0x00…0010
0x00…0018	0x2
0x00…0010	0x1
…	……

Memory

CPU

PC:	0x00…0050
IR:	**movq** (%rsi), %rdx
RAX:	0x1
RBX:	
RCX:	
RDX:	
RSI:	0x00…0018
RDI:	0x00…0010
RSP:	
RBP:	
	…

swap func



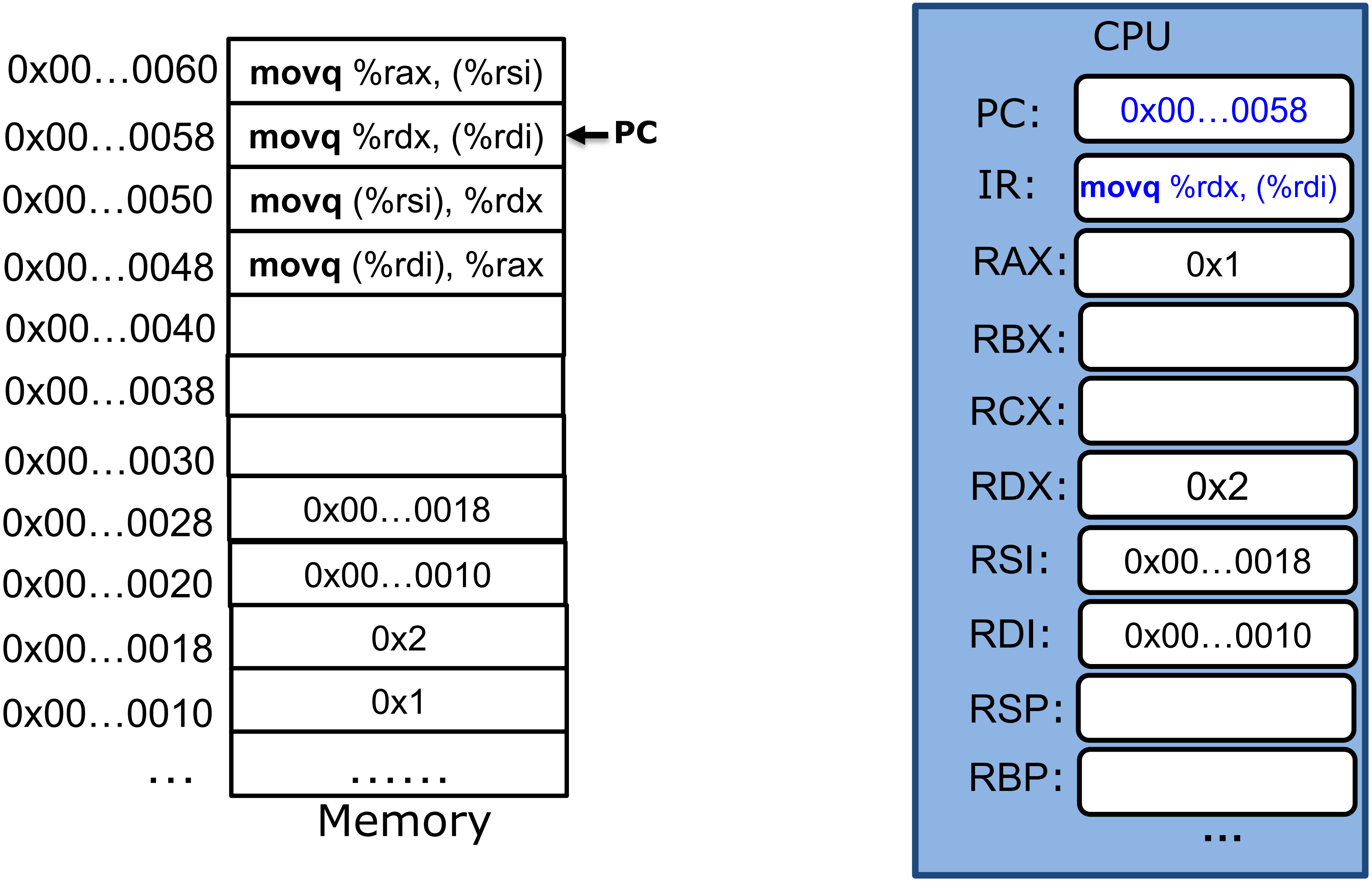
0x00…0060	**movq** %rax, (%rsi)
0x00…0058	**movq** %rdx, (%rdi)
0x00…0050	**movq** (%rsi), %rdx ← **PC**
0x00…0048	**movq** (%rdi), %rax
0x00…0040	
0x00…0038	
0x00…0030	
0x00…0028	0x00…0018
0x00…0020	0x00…0010
0x00…0018	0x2
0x00…0010	0x1
…	……

Memory

CPU

PC:	0x00…0050
IR:	**movq** (%rsi), %rdx
RAX:	0x1
RBX:	
RCX:	
RDX:	0x2
RSI:	0x00…0018
RDI:	0x00…0010
RSP:	
RBP:	
…	

swap func



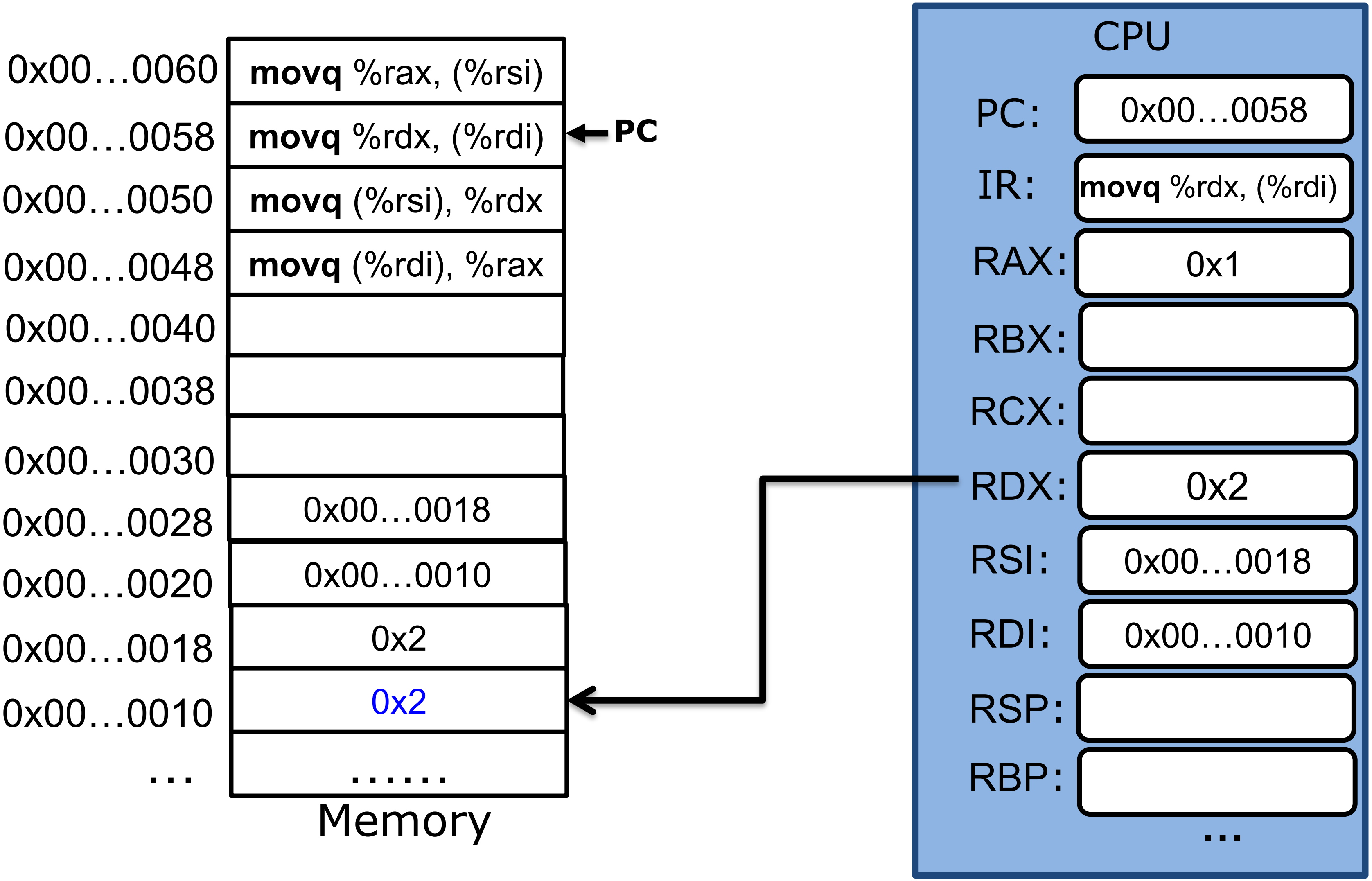
0x00…0060	**movq** %rax, (%rsi)
0x00…0058	**movq** %rdx, (%rdi) ← **PC**
0x00…0050	**movq** (%rsi), %rdx
0x00…0048	**movq** (%rdi), %rax
0x00…0040	
0x00…0038	
0x00…0030	
0x00…0028	0x00…0018
0x00…0020	0x00…0010
0x00…0018	0x2
0x00…0010	0x1
…	……

Memory

CPU

PC:	0x00…0058
IR:	**movq** %rdx, (%rdi)
RAX:	0x1
RBX:	
RCX:	
RDX:	0x2
RSI:	0x00…0018
RDI:	0x00…0010
RSP:	
RBP:	
	…

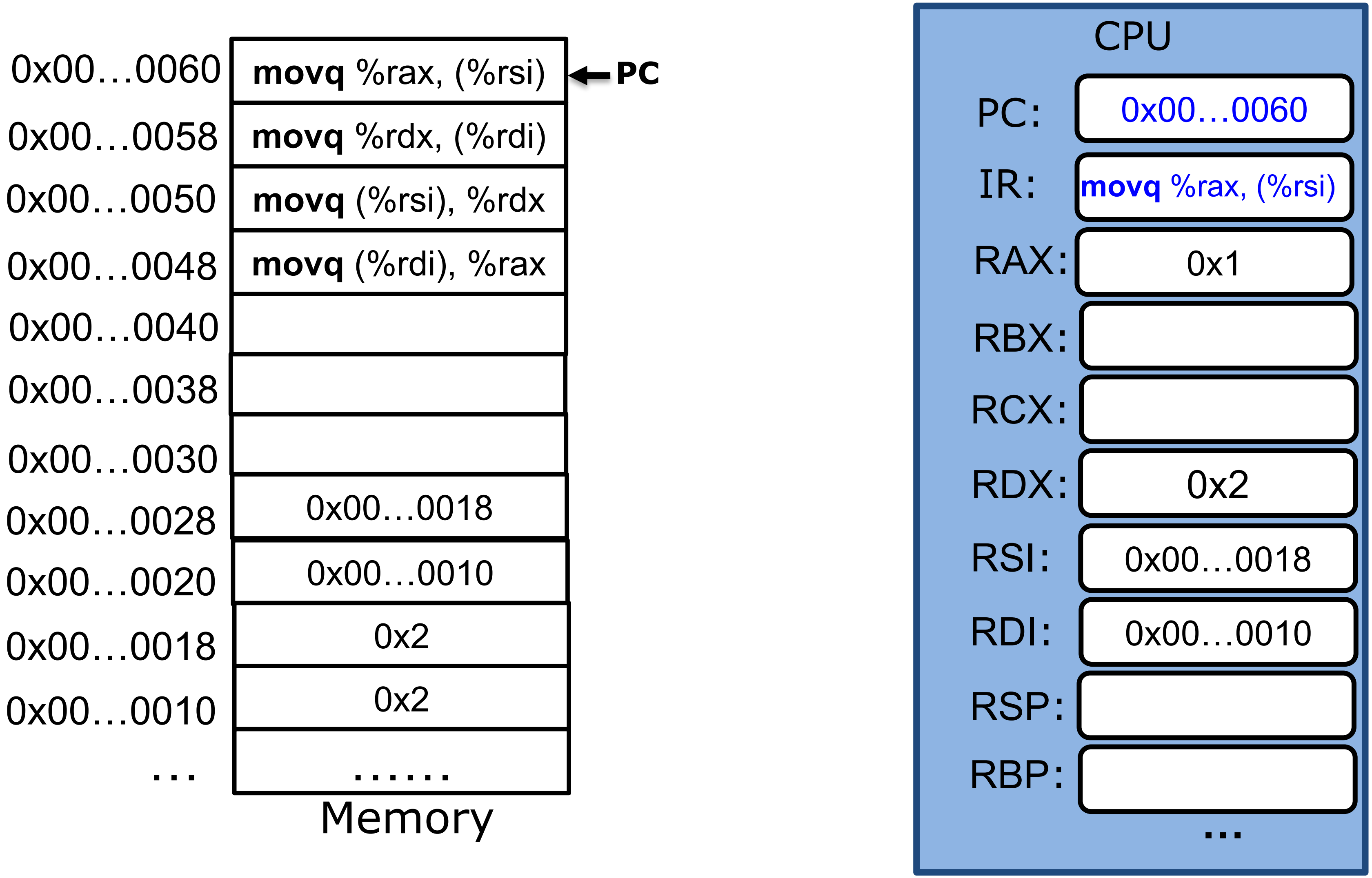
swap func



0x00…0060	**movq** %rax, (%rsi)
0x00…0058	**movq** %rdx, (%rdi)
0x00…0050	**movq** (%rsi), %rdx
0x00…0048	**movq** (%rdi), %rax
0x00…0040	
0x00…0038	
0x00…0030	
0x00…0028	0x00…0018
0x00…0020	0x00…0010
0x00…0018	0x2
0x00…0010	0x2
…	……

PC:	0x00…0058
IR:	**movq** %rdx, (%rdi)
RAX:	0x1
RBX:	
RCX:	
RDX:	0x2
RSI:	0x00…0018
RDI:	0x00…0010
RSP:	
RBP:	
…	

swap func



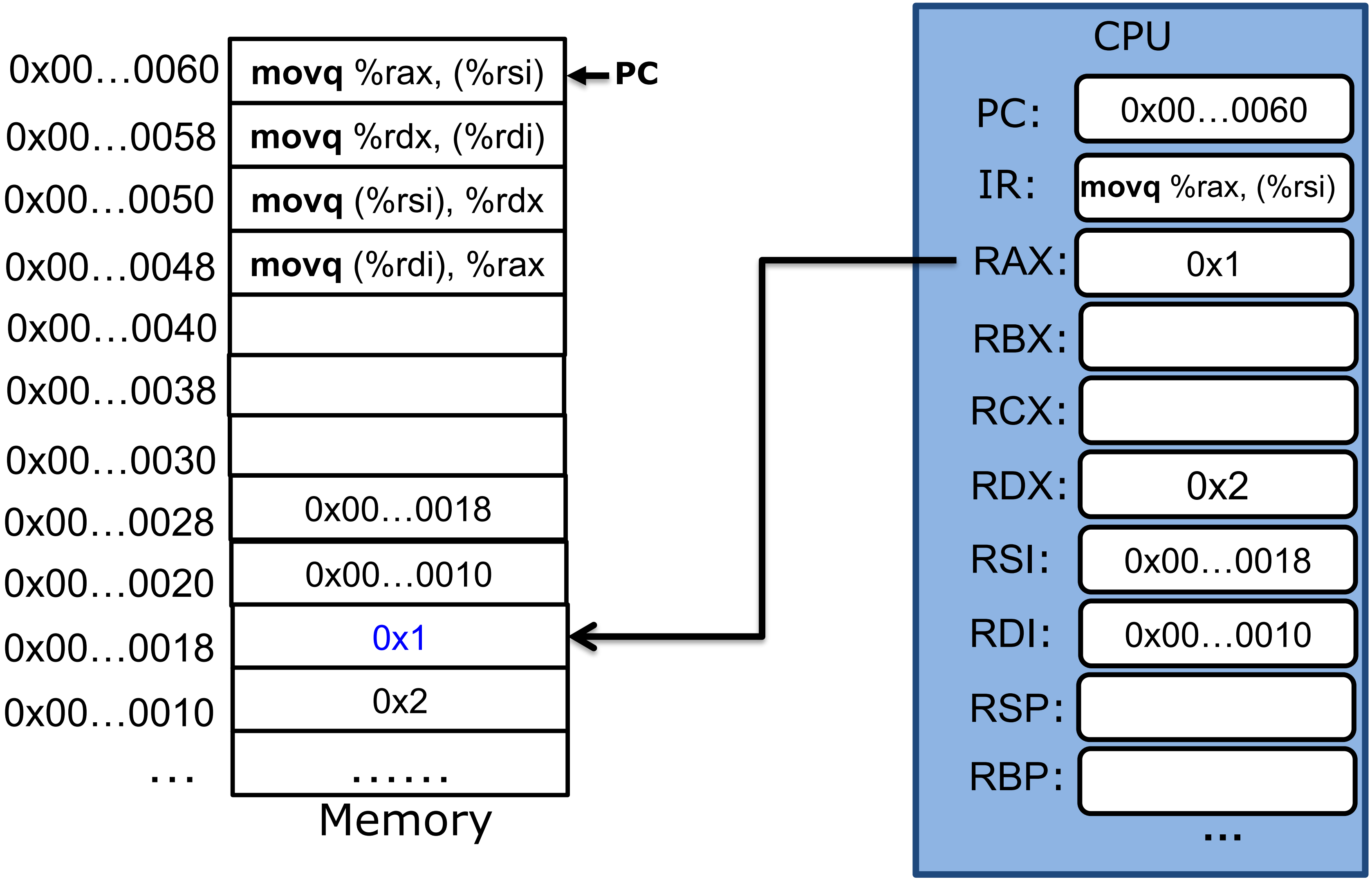
0x00…0060	**movq** %rax, (%rsi) ← **PC**
0x00…0058	**movq** %rdx, (%rdi)
0x00…0050	**movq** (%rsi), %rdx
0x00…0048	**movq** (%rdi), %rax
0x00…0040	
0x00…0038	
0x00…0030	
0x00…0028	0x00…0018
0x00…0020	0x00…0010
0x00…0018	0x2
0x00…0010	0x2
…	……

Memory

CPU

PC:	0x00…0060
IR:	**movq** %rax, (%rsi)
RAX:	0x1
RBX:	
RCX:	
RDX:	0x2
RSI:	0x00…0018
RDI:	0x00…0010
RSP:	
RBP:	
	…

swap func



0x00…0060	**movq** %rax, (%rsi) ← **PC**
0x00…0058	**movq** %rdx, (%rdi)
0x00…0050	**movq** (%rsi), %rdx
0x00…0048	**movq** (%rdi), %rax
0x00…0040	
0x00…0038	
0x00…0030	
0x00…0028	0x00…0018
0x00…0020	0x00…0010
0x00…0018	0x1
0x00…0010	0x2
…	……

Memory

CPU

PC:	0x00…0060
IR:	**movq** %rax, (%rsi)
RAX:	0x1
RBX:	
RCX:	
RDX:	0x2
RSI:	0x00…0018
RDI:	0x00…0010
RSP:	
RBP:	
…	

Limitation of direct addressing

Issue: the address must be calculated and stored in the register before each memory access.

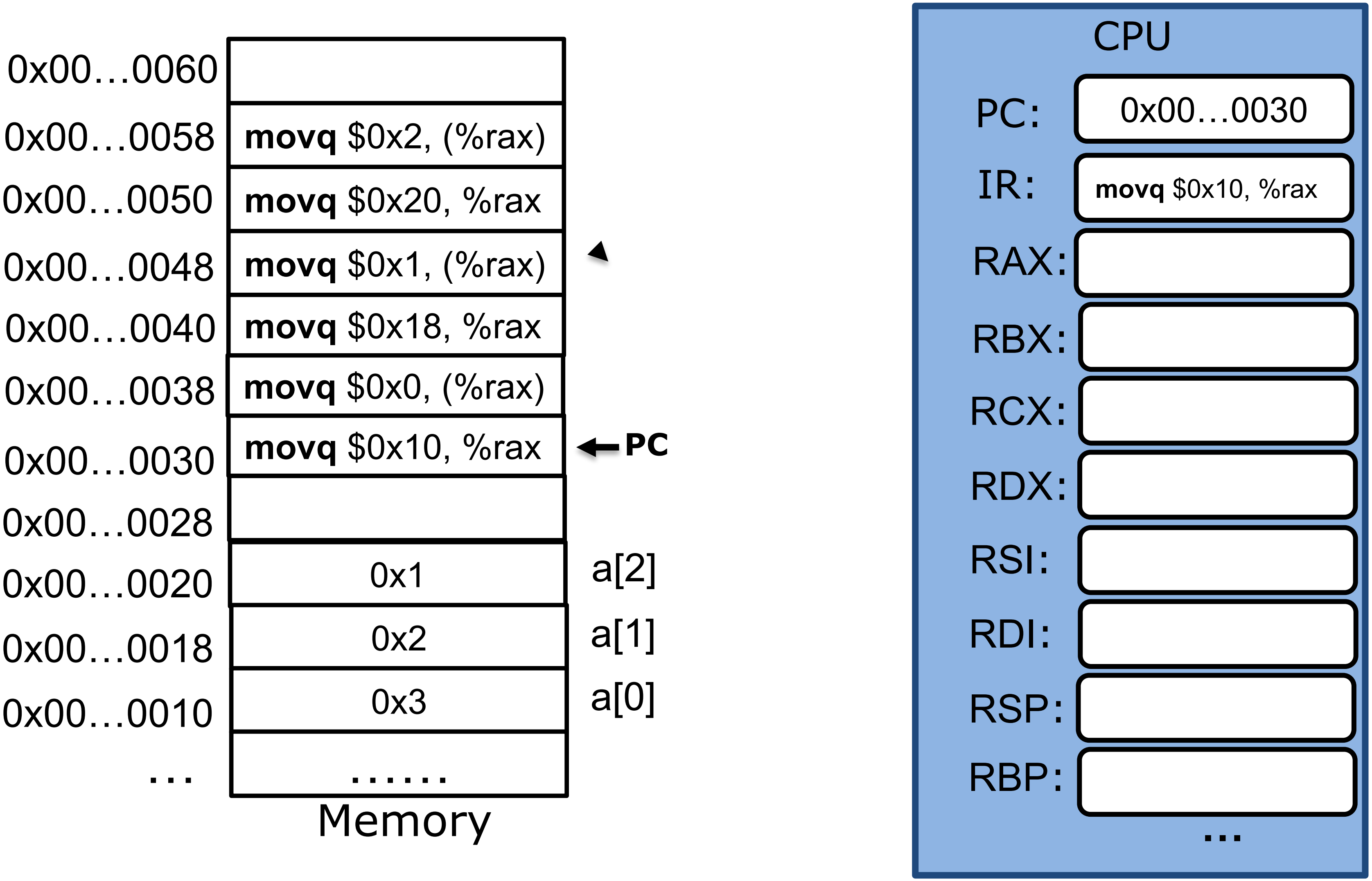
Issue

Issue: the address must be calculated and stored in the register before each memory access.

```
long a[] = {3, 2, 1};
for(int i = 0; i < 3; i++) {
    a[i] = i;
}
```

```
long a[] = {1, 2, 3};
for(int i = 0; i < 3; i++) {
   1. put &a[i] into reg
   2. mov $i, (reg)
}
```

Example



0x00…0060		
0x00…0058	**movq** $0x2, (%rax)	
0x00…0050	**movq** $0x20, %rax	
0x00…0048	**movq** $0x1, (%rax)	
0x00…0040	**movq** $0x18, %rax	
0x00…0038	**movq** $0x0, (%rax)	
0x00…0030	**movq** $0x10, %rax	← **PC**
0x00…0028		
0x00…0020	0x1	a[2]
0x00…0018	0x2	a[1]
0x00…0010	0x3	a[0]
…	……	

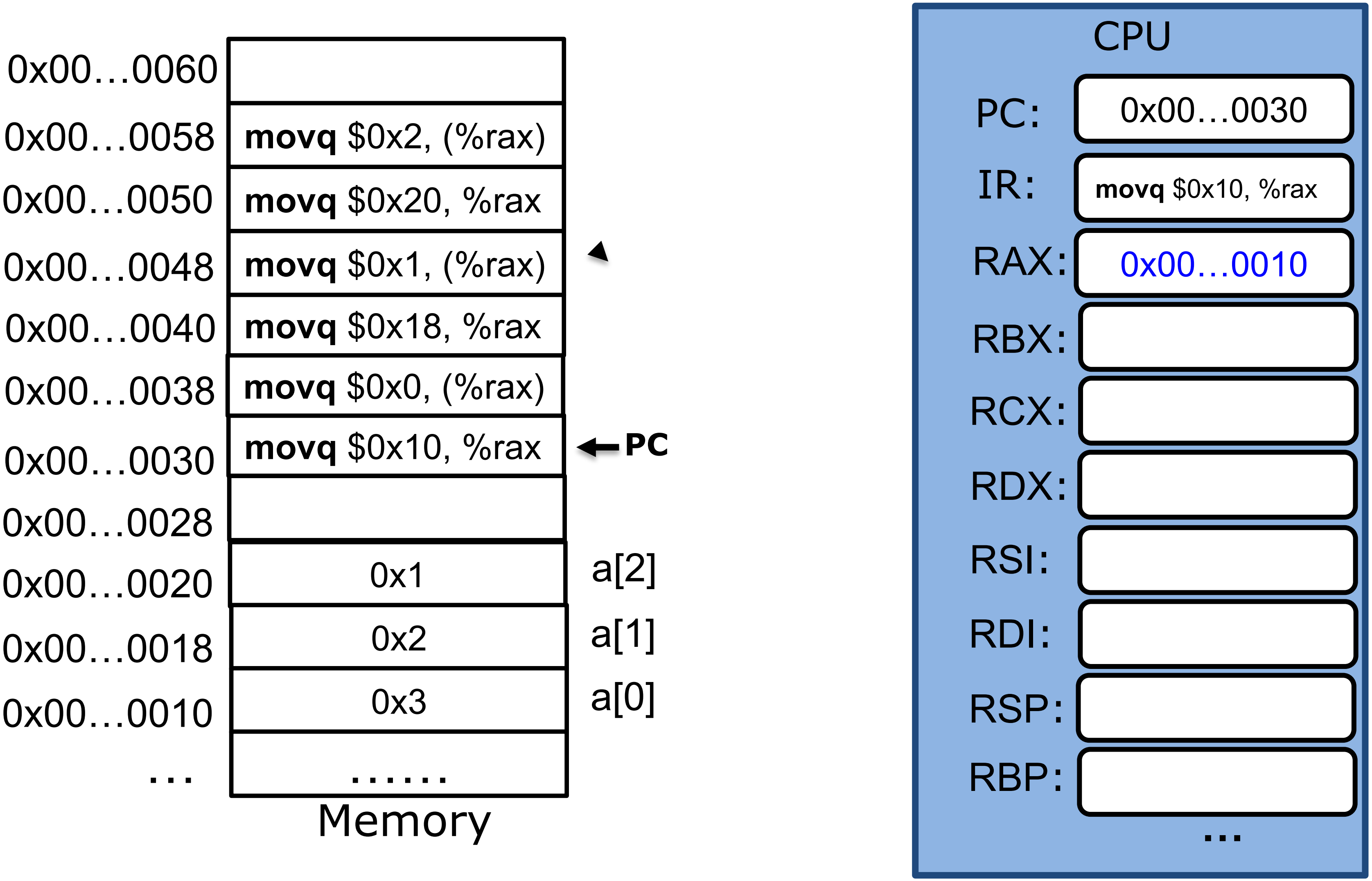
Memory

CPU

PC:	0x00…0030
IR:	**movq** $0x10, %rax
RAX:	
RBX:	
RCX:	
RDX:	
RSI:	
RDI:	
RSP:	
RBP:	

…

Example

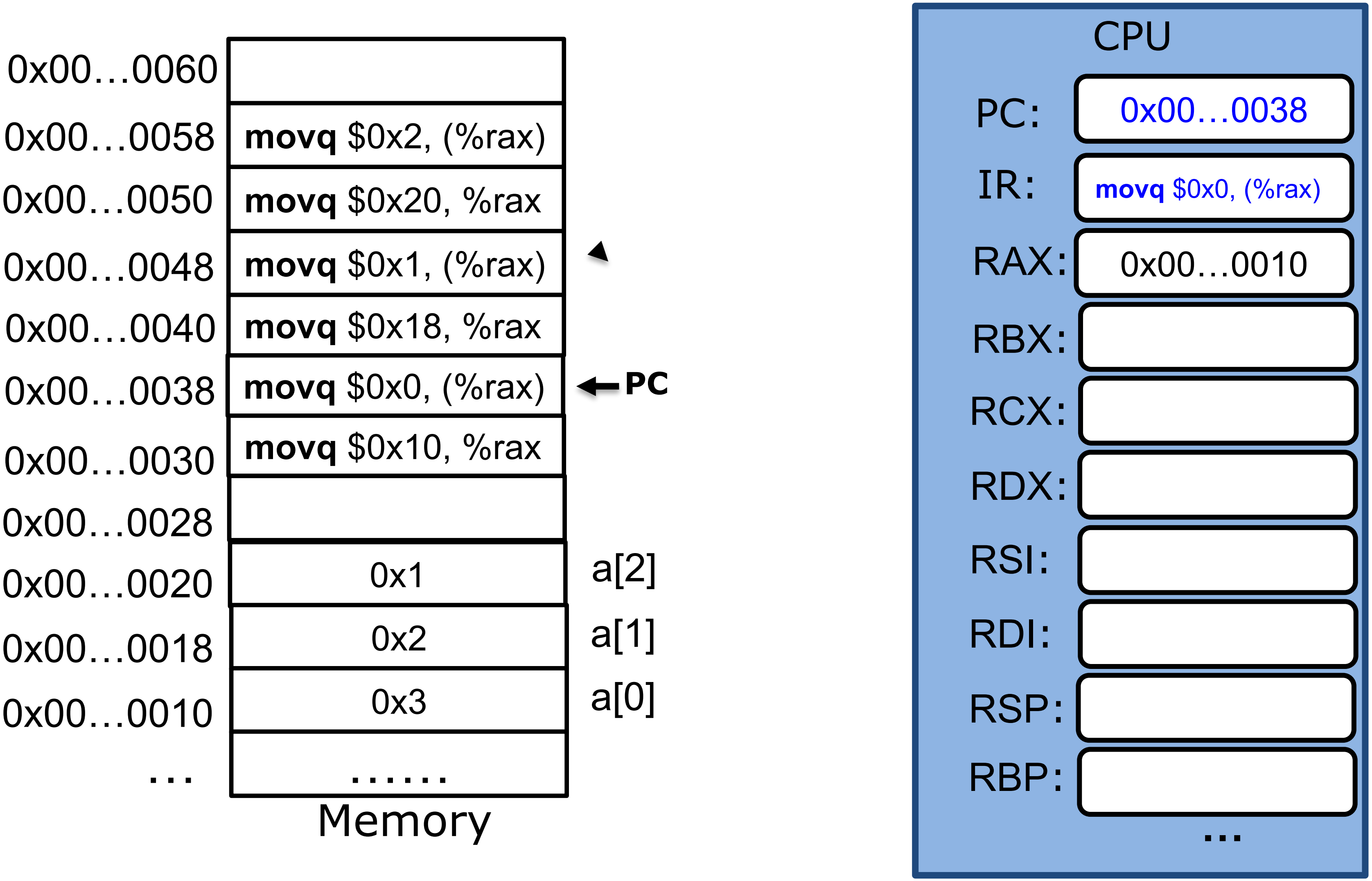


	Memory	
0x00…0060		
0x00…0058	**movq** $0x2, (%rax)	
0x00…0050	**movq** $0x20, %rax	
0x00…0048	**movq** $0x1, (%rax)	
0x00…0040	**movq** $0x18, %rax	
0x00…0038	**movq** $0x0, (%rax)	
0x00…0030	**movq** $0x10, %rax	← **PC**
0x00…0028		
0x00…0020	0x1	a[2]
0x00…0018	0x2	a[1]
0x00…0010	0x3	a[0]
…	……	

CPU

PC:	0x00…0030
IR:	**movq** $0x10, %rax
RAX:	0x00…0010
RBX:	
RCX:	
RDX:	
RSI:	
RDI:	
RSP:	
RBP:	
	…

Example



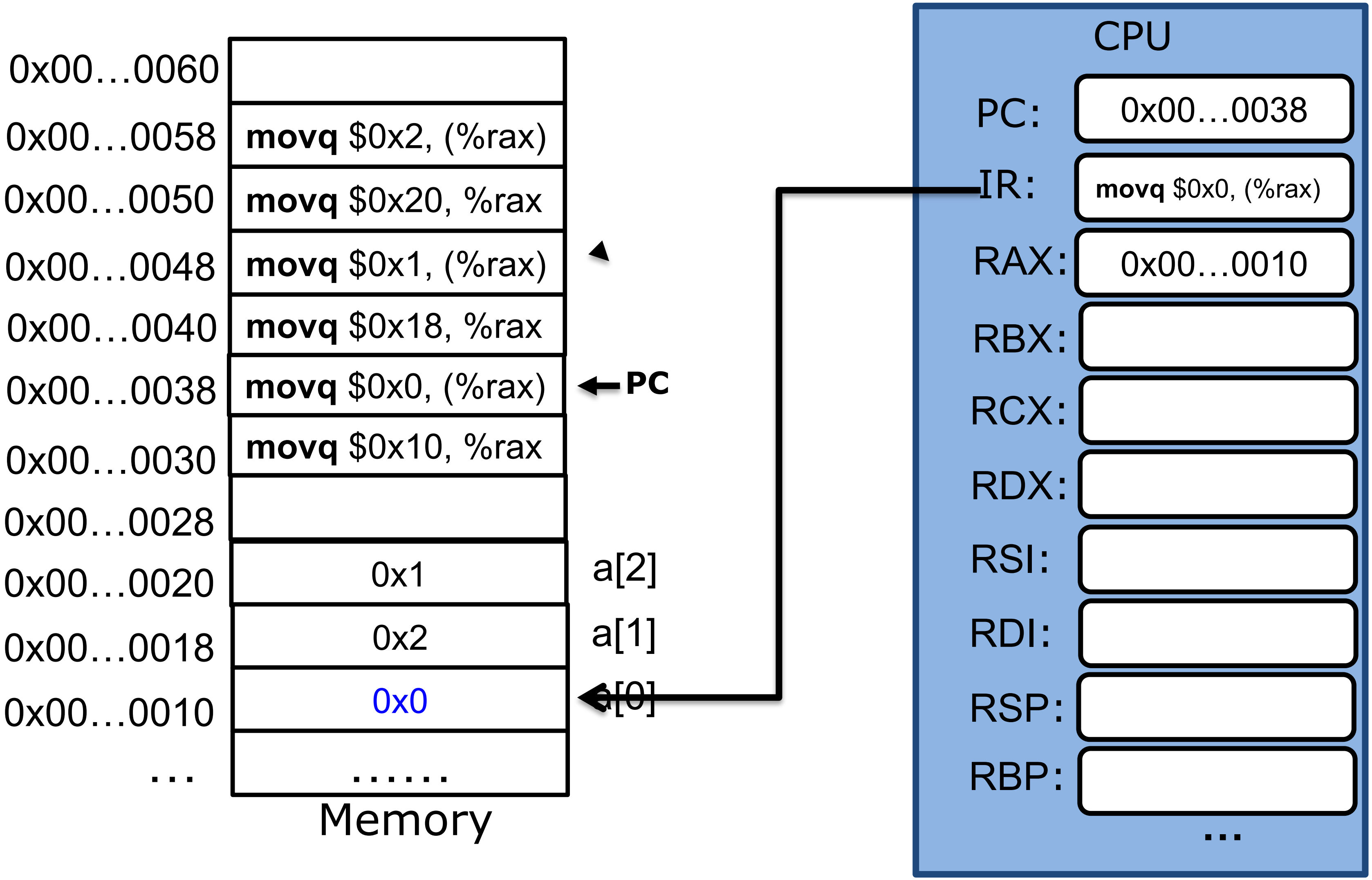
0x00…0060		
0x00…0058	**movq** $0x2, (%rax)	
0x00…0050	**movq** $0x20, %rax	
0x00…0048	**movq** $0x1, (%rax)	
0x00…0040	**movq** $0x18, %rax	
0x00…0038	**movq** $0x0, (%rax)	← **PC**
0x00…0030	**movq** $0x10, %rax	
0x00…0028		
0x00…0020	0x1	a[2]
0x00…0018	0x2	a[1]
0x00…0010	0x3	a[0]
…	……	

Memory

CPU

PC:	0x00…0038
IR:	**movq** $0x0, (%rax)
RAX:	0x00…0010
RBX:	
RCX:	
RDX:	
RSI:	
RDI:	
RSP:	
RBP:	
	…

Example



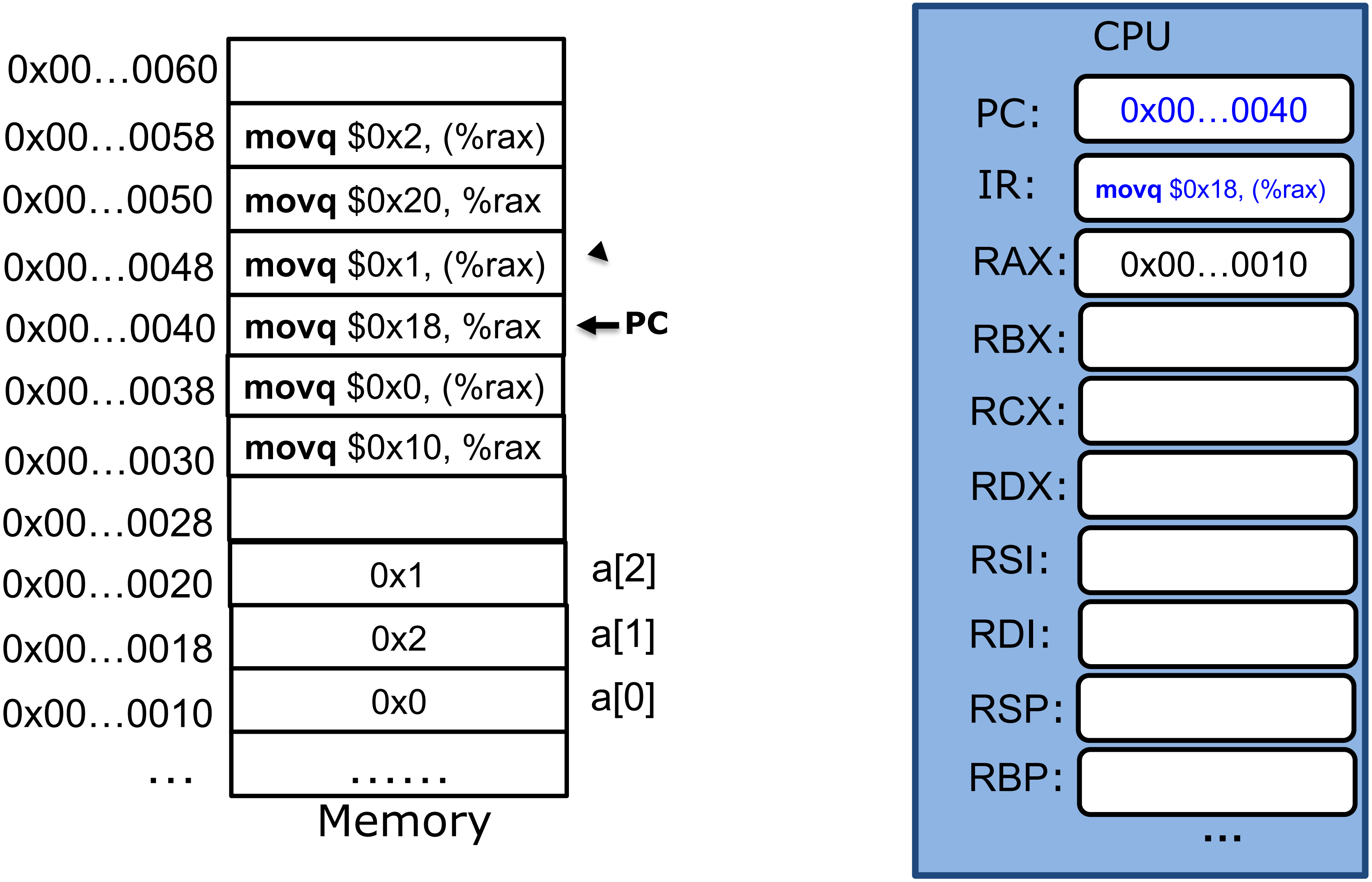
0x00…0060		
0x00…0058	**movq** $0x2, (%rax)	
0x00…0050	**movq** $0x20, %rax	
0x00…0048	**movq** $0x1, (%rax)	
0x00…0040	**movq** $0x18, %rax	
0x00…0038	**movq** $0x0, (%rax)	← **PC**
0x00…0030	**movq** $0x10, %rax	
0x00…0028		
0x00…0020	0x1	a[2]
0x00…0018	0x2	a[1]
0x00…0010	0x0	a[0]
…	……	

Memory

CPU

PC:	0x00…0038
IR:	**movq** $0x0, (%rax)
RAX:	0x00…0010
RBX:	
RCX:	
RDX:	
RSI:	
RDI:	
RSP:	
RBP:	
	…

Example



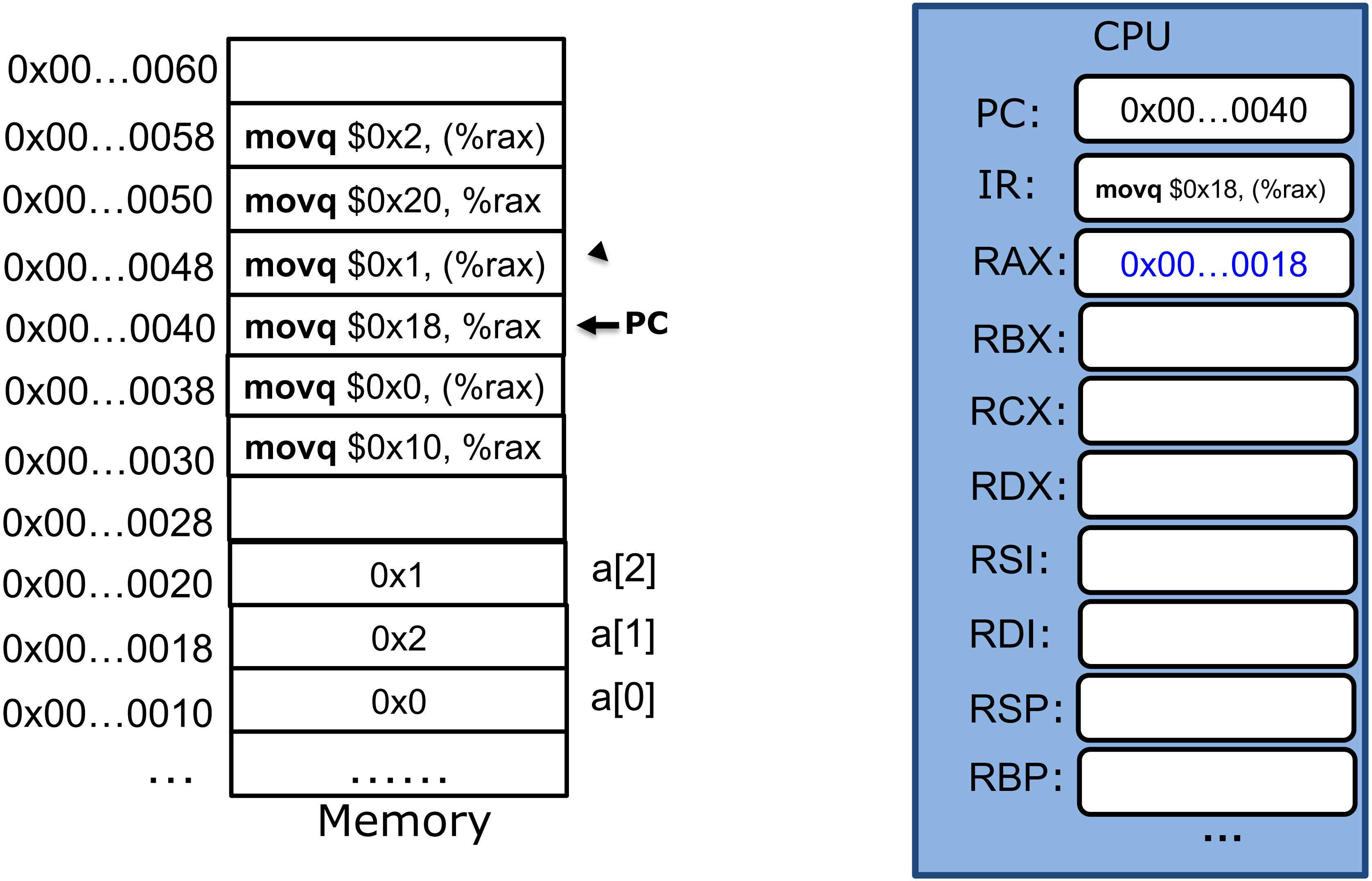
0x00…0060		
0x00…0058	**movq** $0x2, (%rax)	
0x00…0050	**movq** $0x20, %rax	
0x00…0048	**movq** $0x1, (%rax)	
0x00…0040	**movq** $0x18, %rax	← **PC**
0x00…0038	**movq** $0x0, (%rax)	
0x00…0030	**movq** $0x10, %rax	
0x00…0028		
0x00…0020	0x1	a[2]
0x00…0018	0x2	a[1]
0x00…0010	0x0	a[0]
…	……	

Memory

CPU

PC:	0x00…0040
IR:	**movq** $0x18, (%rax)
RAX:	0x00…0010
RBX:	
RCX:	
RDX:	
RSI:	
RDI:	
RSP:	
RBP:	
	…

Example



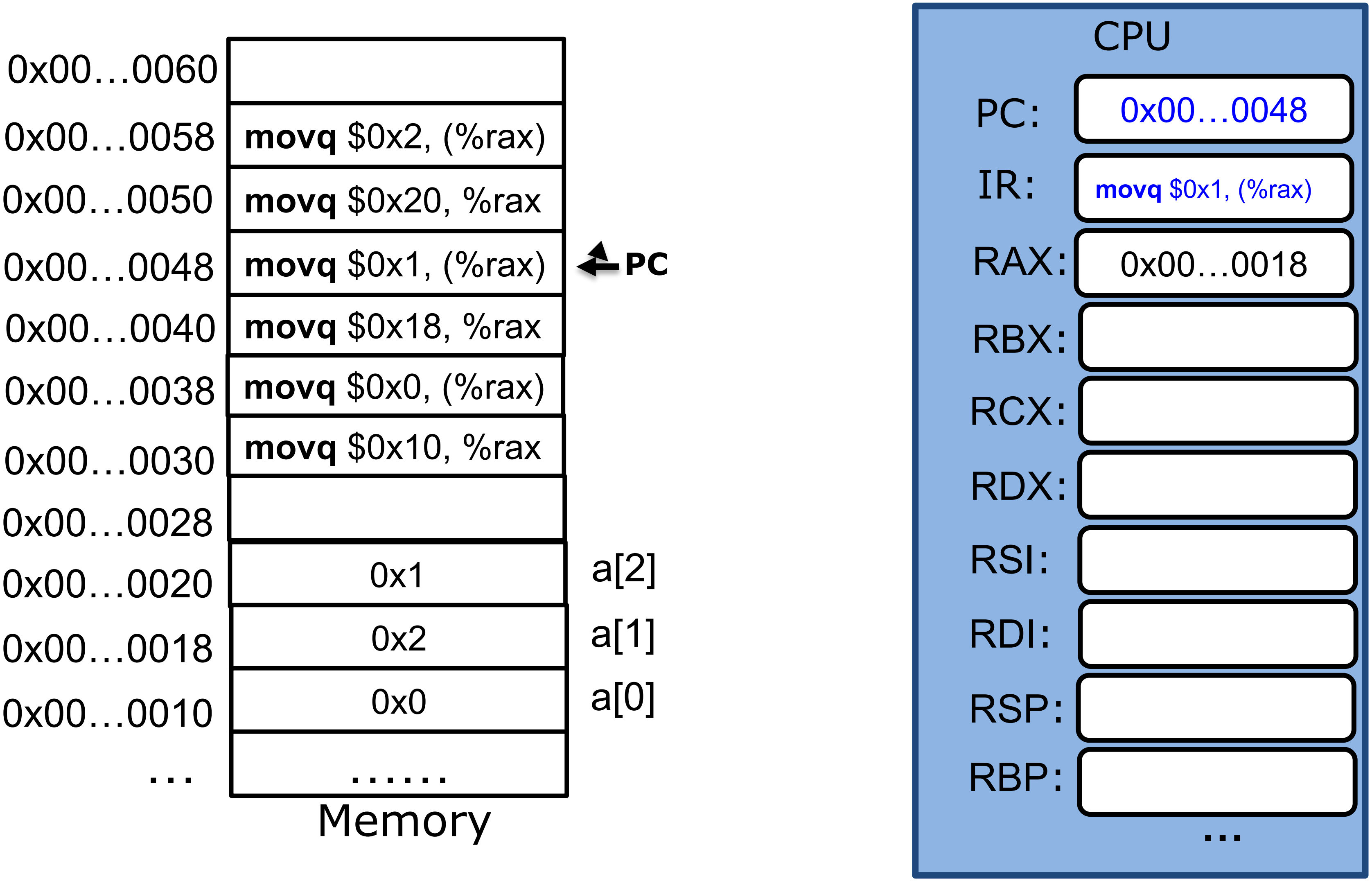
0x00…0060		
0x00…0058	**movq** $0x2, (%rax)	
0x00…0050	**movq** $0x20, %rax	
0x00…0048	**movq** $0x1, (%rax)	
0x00…0040	**movq** $0x18, %rax	← **PC**
0x00…0038	**movq** $0x0, (%rax)	
0x00…0030	**movq** $0x10, %rax	
0x00…0028		
0x00…0020	0x1	a[2]
0x00…0018	0x2	a[1]
0x00…0010	0x0	a[0]
…	……	

Memory

CPU

PC:	0x00…0040
IR:	**movq** $0x18, (%rax)
RAX:	0x00…0018
RBX:	
RCX:	
RDX:	
RSI:	
RDI:	
RSP:	
RBP:	
	…

Example



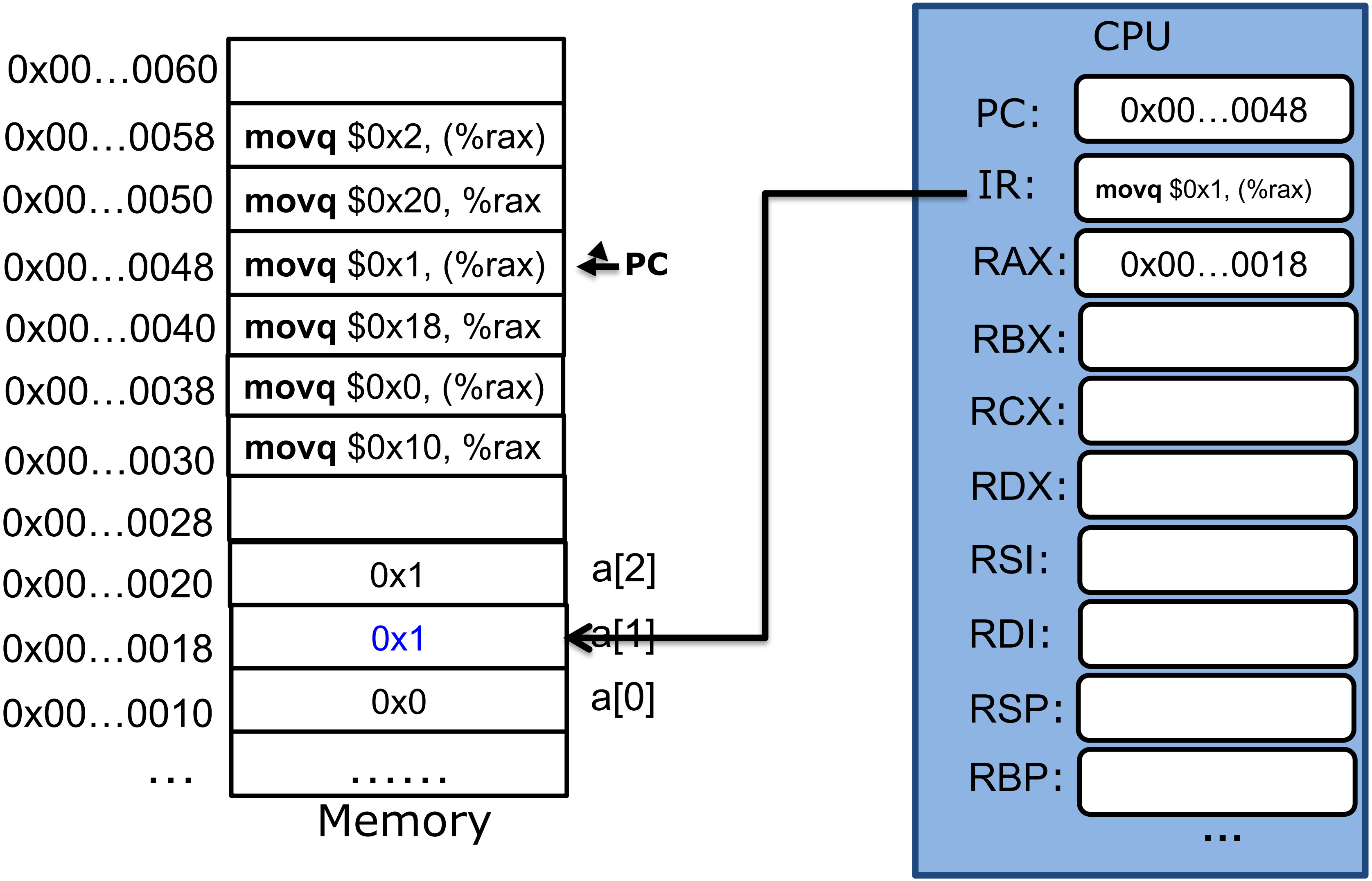
Address	Memory	
0x00…0060		
0x00…0058	**movq** $0x2, (%rax)	
0x00…0050	**movq** $0x20, %rax	
0x00…0048	**movq** $0x1, (%rax)	← **PC**
0x00…0040	**movq** $0x18, %rax	
0x00…0038	**movq** $0x0, (%rax)	
0x00…0030	**movq** $0x10, %rax	
0x00…0028		
0x00…0020	0x1	a[2]
0x00…0018	0x2	a[1]
0x00…0010	0x0	a[0]
…	……	

Memory

CPU

PC:	0x00…0048
IR:	**movq** $0x1, (%rax)
RAX:	0x00…0018
RBX:	
RCX:	
RDX:	
RSI:	
RDI:	
RSP:	
RBP:	
	…

Example



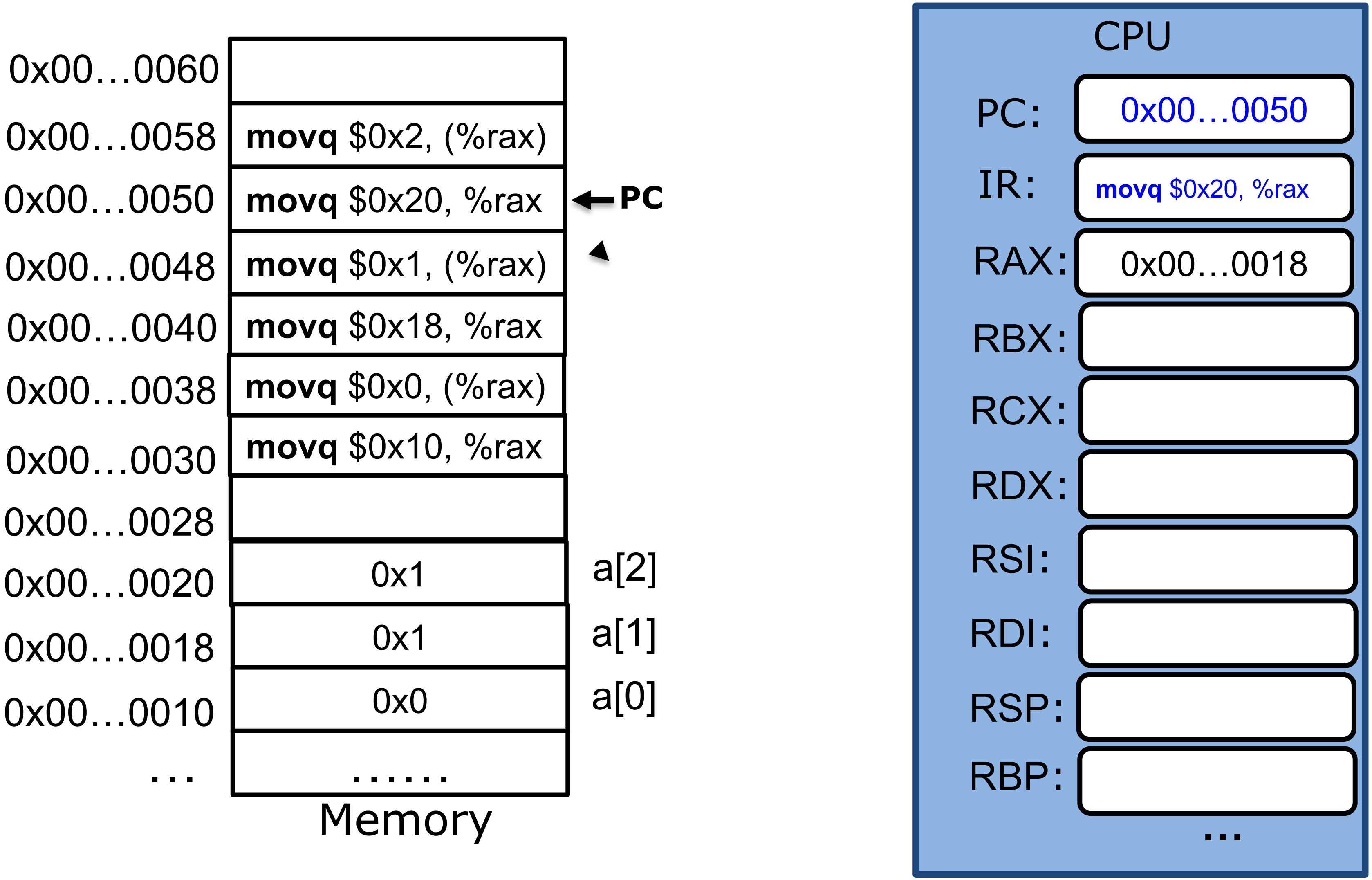
0x00…0060		
0x00…0058	**movq** $0x2, (%rax)	
0x00…0050	**movq** $0x20, %rax	
0x00…0048	**movq** $0x1, (%rax)	**PC**
0x00…0040	**movq** $0x18, %rax	
0x00…0038	**movq** $0x0, (%rax)	
0x00…0030	**movq** $0x10, %rax	
0x00…0028		
0x00…0020	0x1	a[2]
0x00…0018	0x1	a[1]
0x00…0010	0x0	a[0]
…	……	

Memory

CPU

PC:	0x00…0048
IR:	**movq** $0x1, (%rax)
RAX:	0x00…0018
RBX:	
RCX:	
RDX:	
RSI:	
RDI:	
RSP:	
RBP:	
	…

Example



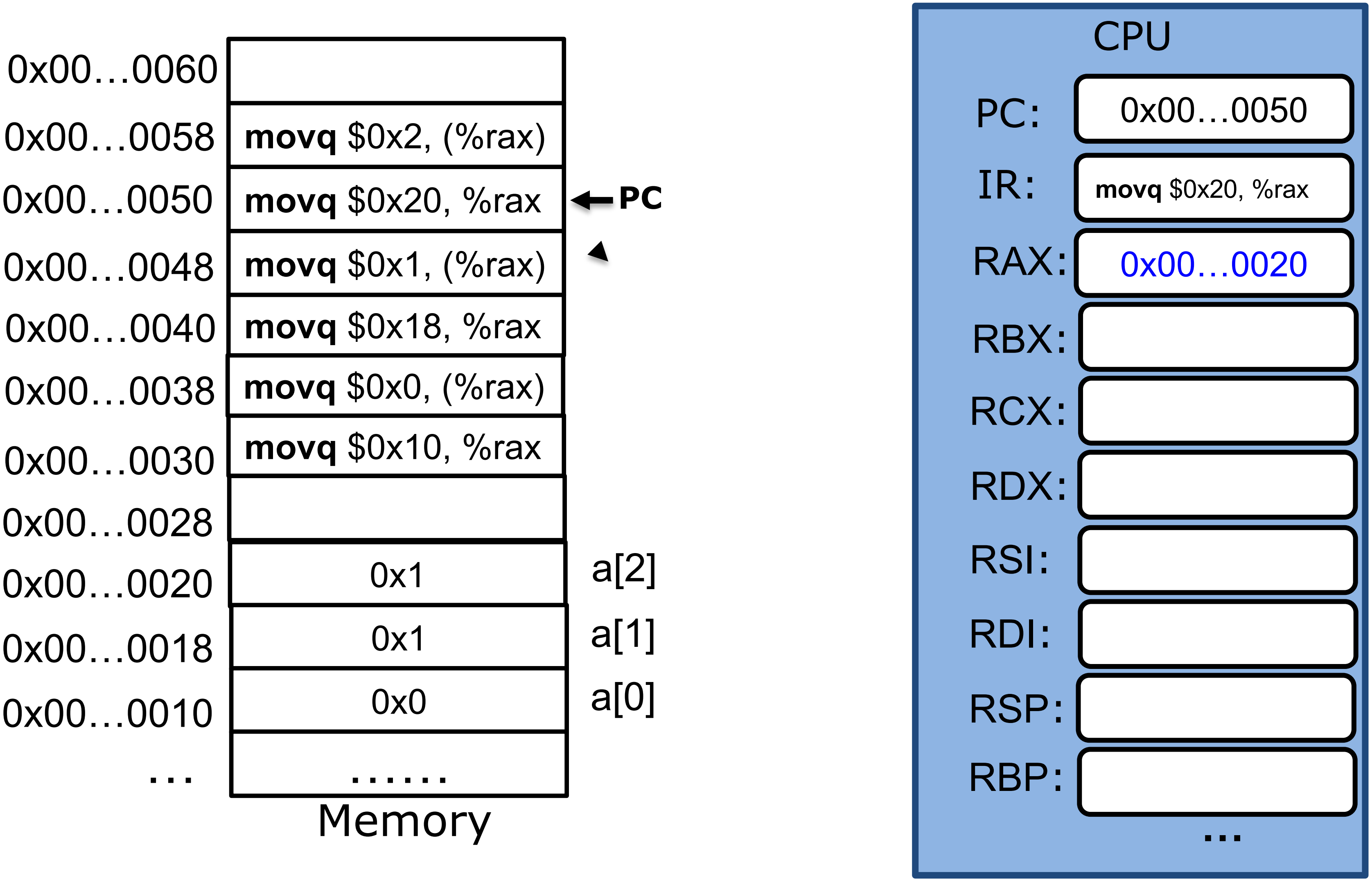
0x00…0060		
0x00…0058	**movq** $0x2, (%rax)	
0x00…0050	**movq** $0x20, %rax	← **PC**
0x00…0048	**movq** $0x1, (%rax)	
0x00…0040	**movq** $0x18, %rax	
0x00…0038	**movq** $0x0, (%rax)	
0x00…0030	**movq** $0x10, %rax	
0x00…0028		
0x00…0020	0x1	a[2]
0x00…0018	0x1	a[1]
0x00…0010	0x0	a[0]
…	……	

Memory

CPU

PC:	0x00…0050
IR:	**movq** $0x20, %rax
RAX:	0x00…0018
RBX:	
RCX:	
RDX:	
RSI:	
RDI:	
RSP:	
RBP:	
	…

Example



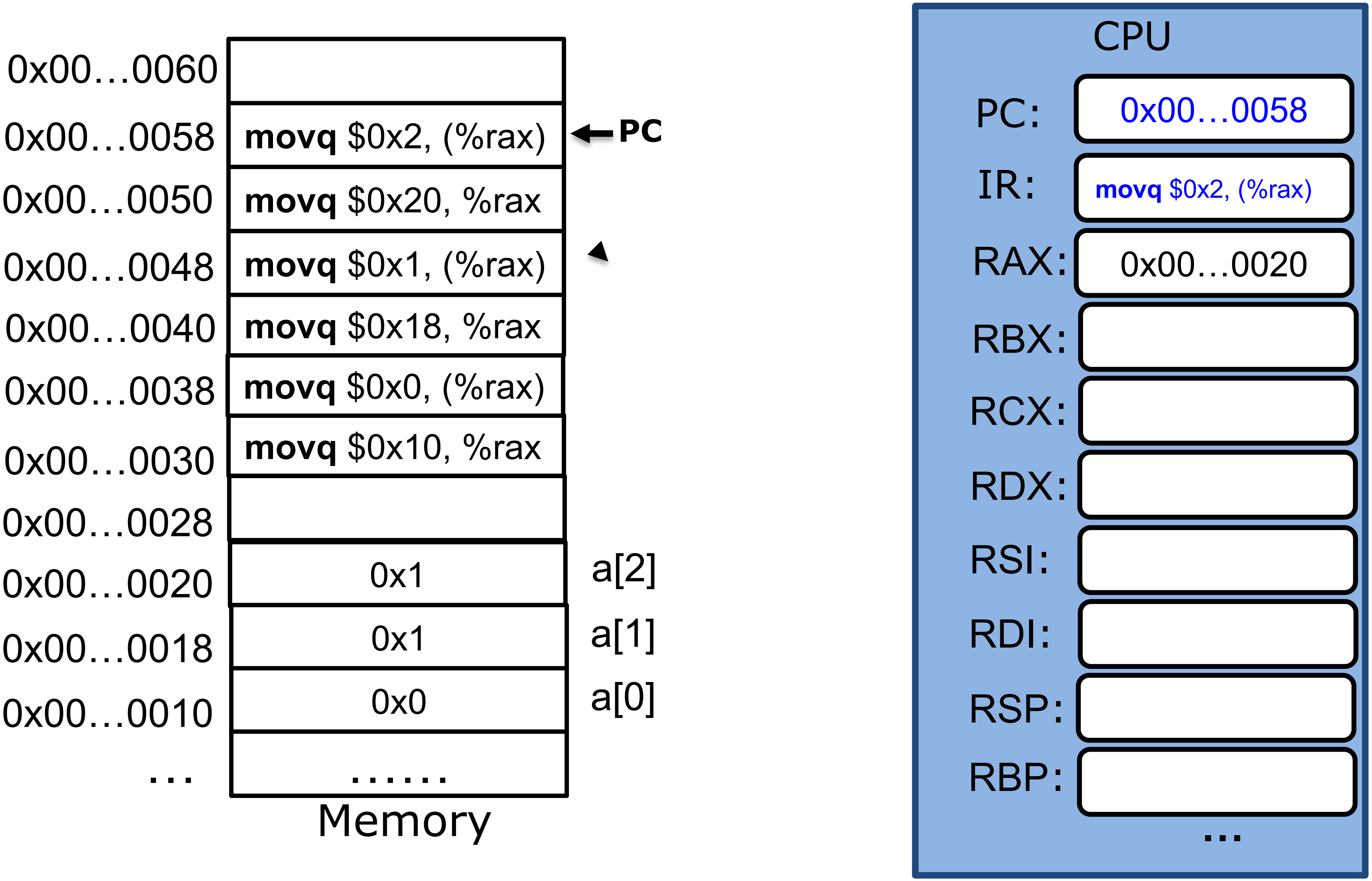
0x00…0060		
0x00…0058	**movq** $0x2, (%rax)	
0x00…0050	**movq** $0x20, %rax	← **PC**
0x00…0048	**movq** $0x1, (%rax)	
0x00…0040	**movq** $0x18, %rax	
0x00…0038	**movq** $0x0, (%rax)	
0x00…0030	**movq** $0x10, %rax	
0x00…0028		
0x00…0020	0x1	a[2]
0x00…0018	0x1	a[1]
0x00…0010	0x0	a[0]
…	……	

Memory

CPU

PC:	0x00…0050
IR:	**movq** $0x20, %rax
RAX:	0x00…0020
RBX:	
RCX:	
RDX:	
RSI:	
RDI:	
RSP:	
RBP:	
	…

Example



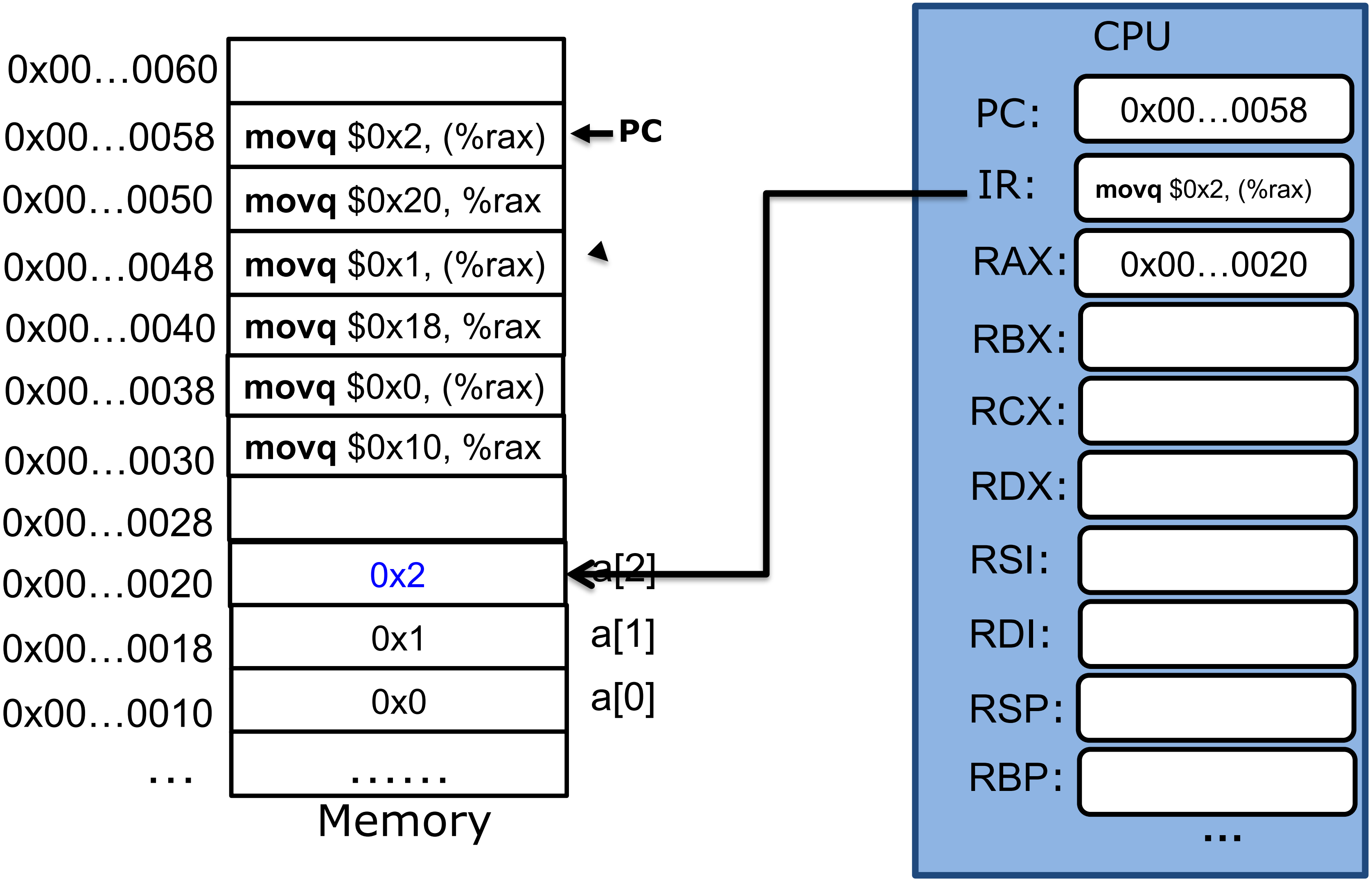
0x00…0060		
0x00…0058	**movq** $0x2, (%rax)	← **PC**
0x00…0050	**movq** $0x20, %rax	
0x00…0048	**movq** $0x1, (%rax)	
0x00…0040	**movq** $0x18, %rax	
0x00…0038	**movq** $0x0, (%rax)	
0x00…0030	**movq** $0x10, %rax	
0x00…0028		
0x00…0020	0x1	a[2]
0x00…0018	0x1	a[1]
0x00…0010	0x0	a[0]
…	……	

Memory

CPU

PC:	0x00…0058
IR:	**movq** $0x2, (%rax)
RAX:	0x00…0020
RBX:	
RCX:	
RDX:	
RSI:	
RDI:	
RSP:	
RBP:	
	…

Example



0x00…0060		
0x00…0058	**movq** $0x2, (%rax)	PC
0x00…0050	**movq** $0x20, %rax	
0x00…0048	**movq** $0x1, (%rax)	
0x00…0040	**movq** $0x18, %rax	
0x00…0038	**movq** $0x0, (%rax)	
0x00…0030	**movq** $0x10, %rax	
0x00…0028		
0x00…0020	0x2	a[2]
0x00…0018	0x1	a[1]
0x00…0010	0x0	a[0]
…	……	

Memory

CPU

PC:	0x00…0058
IR:	**movq** $0x2, (%rax)
RAX:	0x00…0020
RBX:	
RCX:	
RDX:	
RSI:	
RDI:	
RSP:	
RBP:	
	…

Observation

```
long a[] = {3, 2, 1};
for(int i = 0; i < 3; i++) {
    a[i] = i;
}
```

a[0], a[1] and a[2] have the same base address:&a[0]

- &a[0]: &a[0] + 0
- &a[1]: &a[0] + 1
- &a[2]: &a[0] + 2

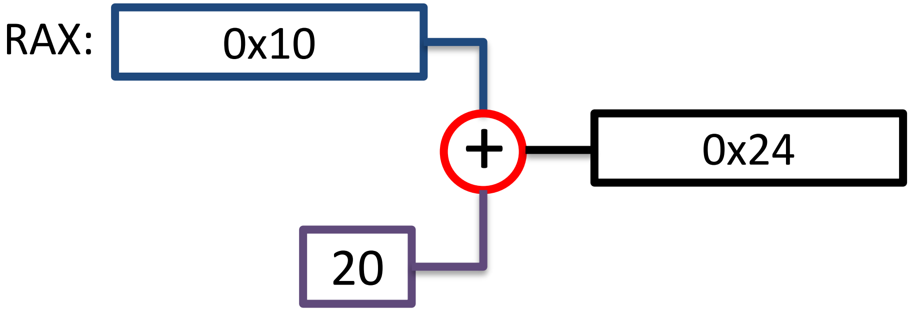
Address mode with displacement

D(Register): val(Register) + D

- Register specifies the start of the memory region
- Constant D specifies the offset

RAX: 0x10

20(RAX)



Address mode with displacement

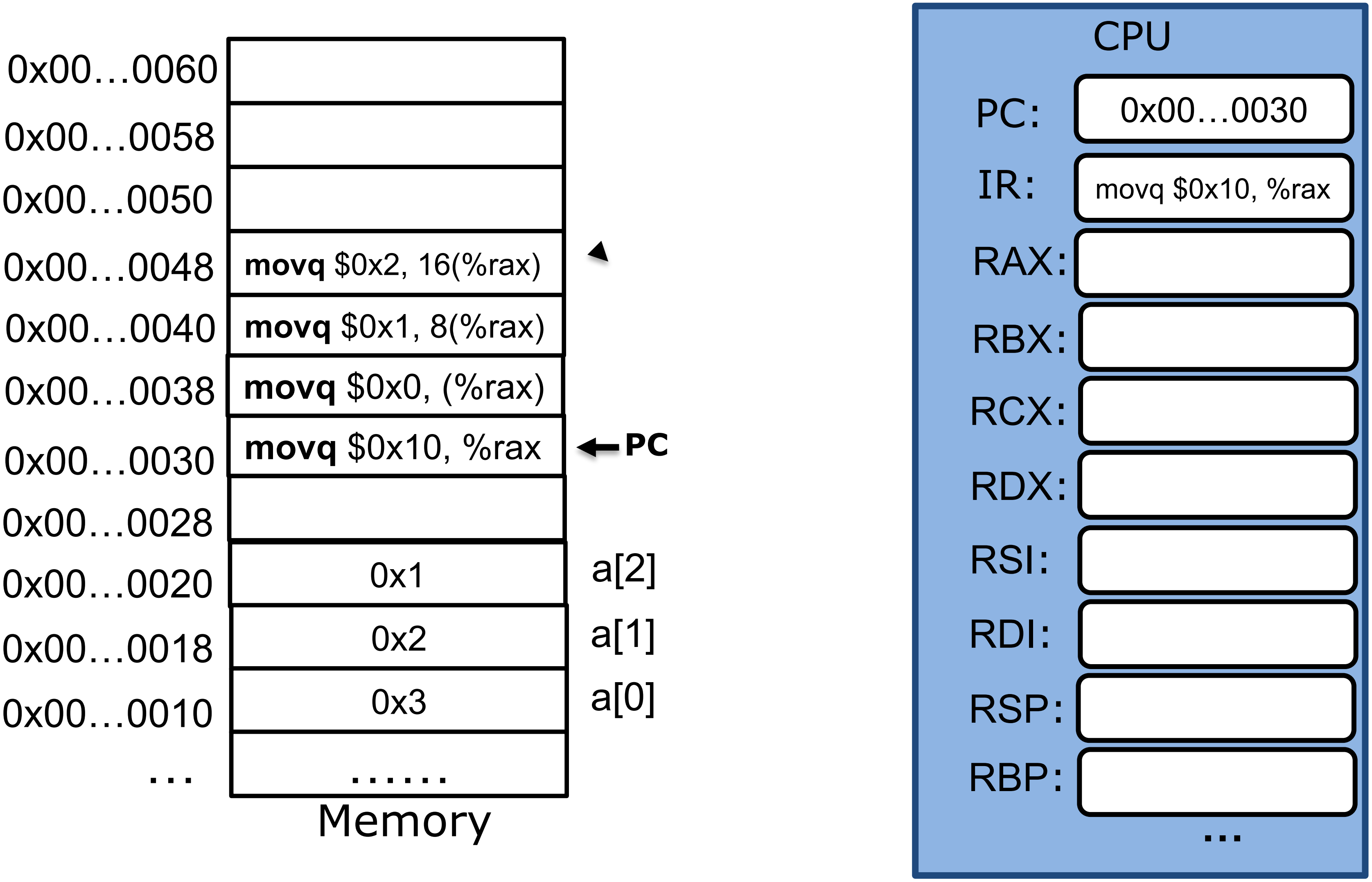
D(Register): val(Register) + D

- Register specifies the start of the memory region
- Constant D specifies the offset

```
long a[] = {3, 2, 1};
for(int i = 0; i < 3; i++) {
    a[i] = i;
}
```

```
long a[] = {1, 2, 3};
for(int i = 0; i < 3; i++) {
    mov $i, D(reg); // D = i * 8, reg = &a[0]
}
```

Example



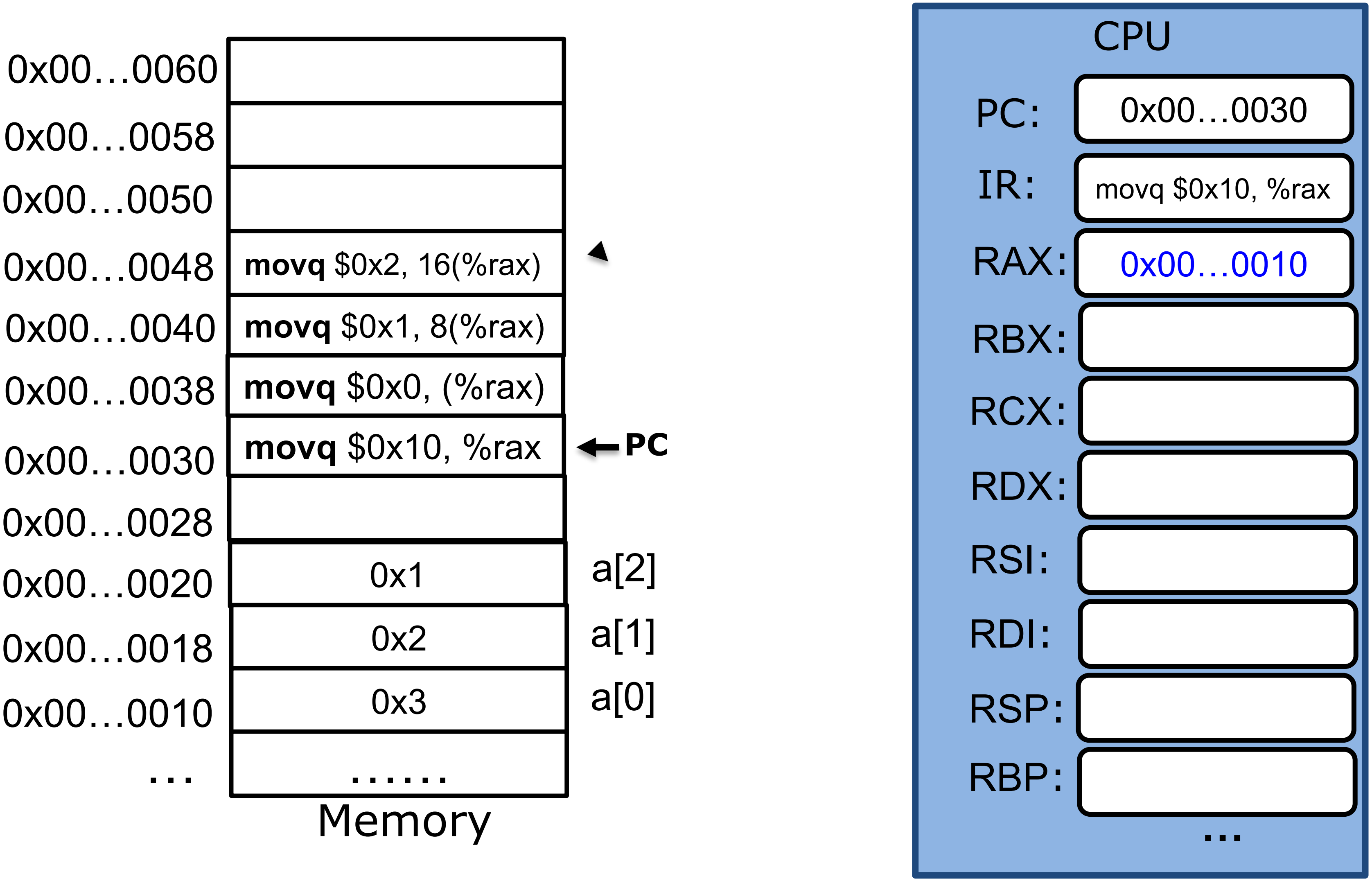
0x00…0060		
0x00…0058		
0x00…0050		
0x00…0048	**movq** $0x2, 16(%rax)	
0x00…0040	**movq** $0x1, 8(%rax)	
0x00…0038	**movq** $0x0, (%rax)	
0x00…0030	**movq** $0x10, %rax	← **PC**
0x00…0028		
0x00…0020	0x1	a[2]
0x00…0018	0x2	a[1]
0x00…0010	0x3	a[0]
…	……	

Memory

CPU

PC:	0x00…0030
IR:	movq $0x10, %rax
RAX:	
RBX:	
RCX:	
RDX:	
RSI:	
RDI:	
RSP:	
RBP:	
	…

Example



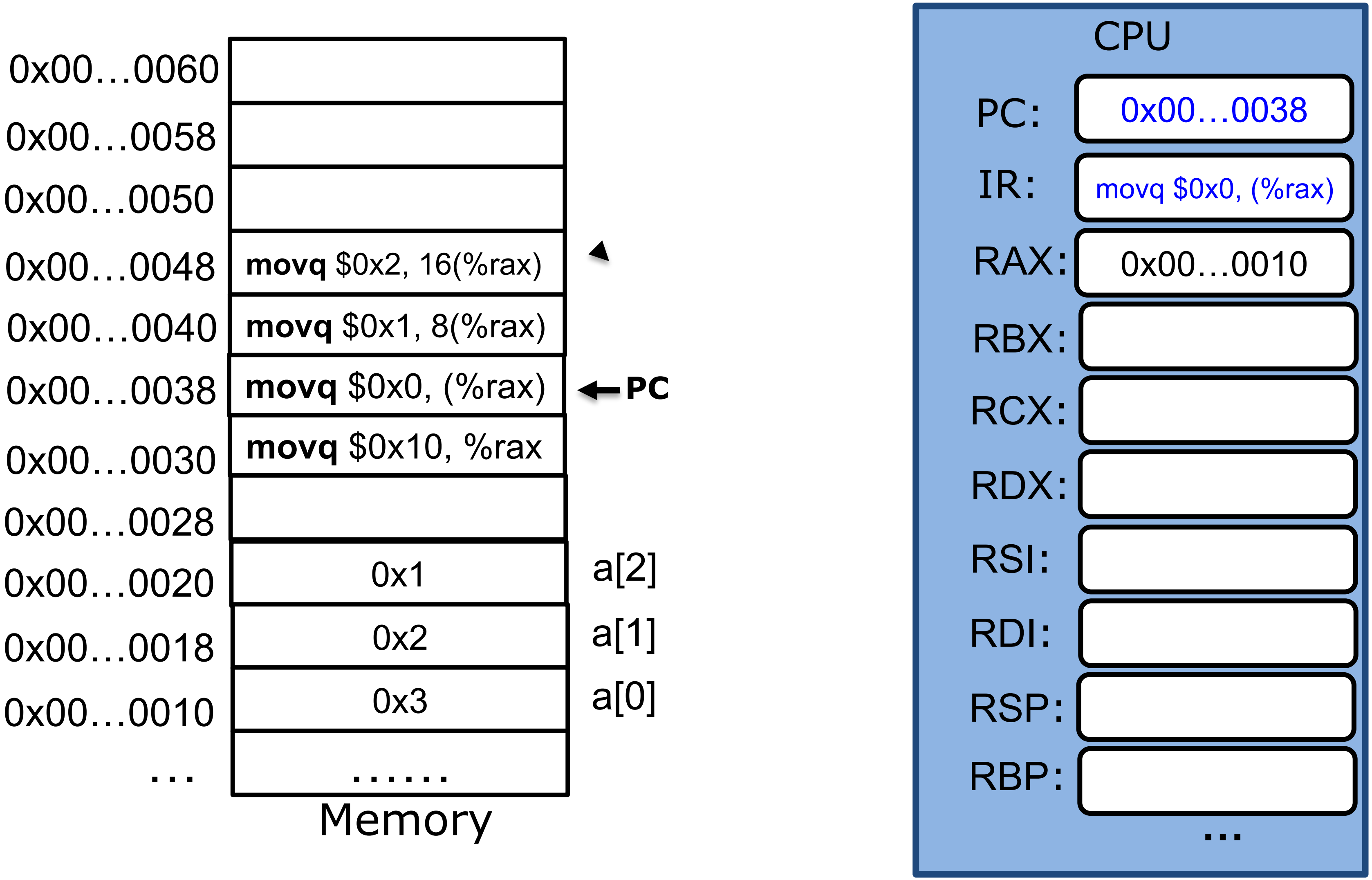
0x00…0060		
0x00…0058		
0x00…0050		
0x00…0048	**movq** $0x2, 16(%rax)	
0x00…0040	**movq** $0x1, 8(%rax)	
0x00…0038	**movq** $0x0, (%rax)	
0x00…0030	**movq** $0x10, %rax	← **PC**
0x00…0028		
0x00…0020	0x1	a[2]
0x00…0018	0x2	a[1]
0x00…0010	0x3	a[0]
…	……	

Memory

CPU

PC:	0x00…0030
IR:	movq $0x10, %rax
RAX:	0x00…0010
RBX:	
RCX:	
RDX:	
RSI:	
RDI:	
RSP:	
RBP:	
	…

Example



0x00…0060		
0x00…0058		
0x00…0050		
0x00…0048	**movq** $0x2, 16(%rax)	
0x00…0040	**movq** $0x1, 8(%rax)	
0x00…0038	**movq** $0x0, (%rax)	← **PC**
0x00…0030	**movq** $0x10, %rax	
0x00…0028		
0x00…0020	0x1	a[2]
0x00…0018	0x2	a[1]
0x00…0010	0x3	a[0]
…	……	

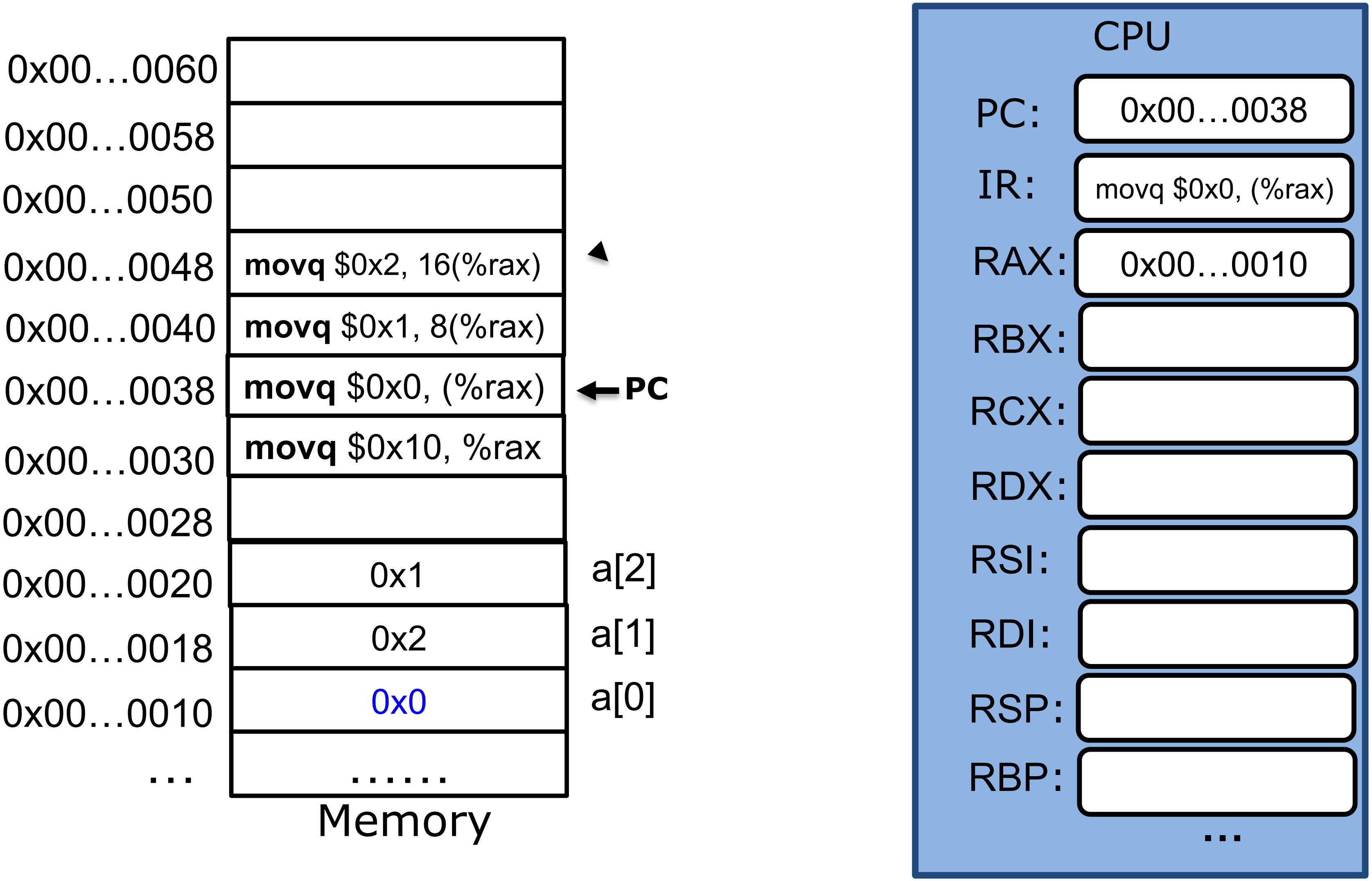
Memory

CPU

PC:	0x00…0038
IR:	movq $0x0, (%rax)
RAX:	0x00…0010
RBX:	
RCX:	
RDX:	
RSI:	
RDI:	
RSP:	
RBP:	

…

Example



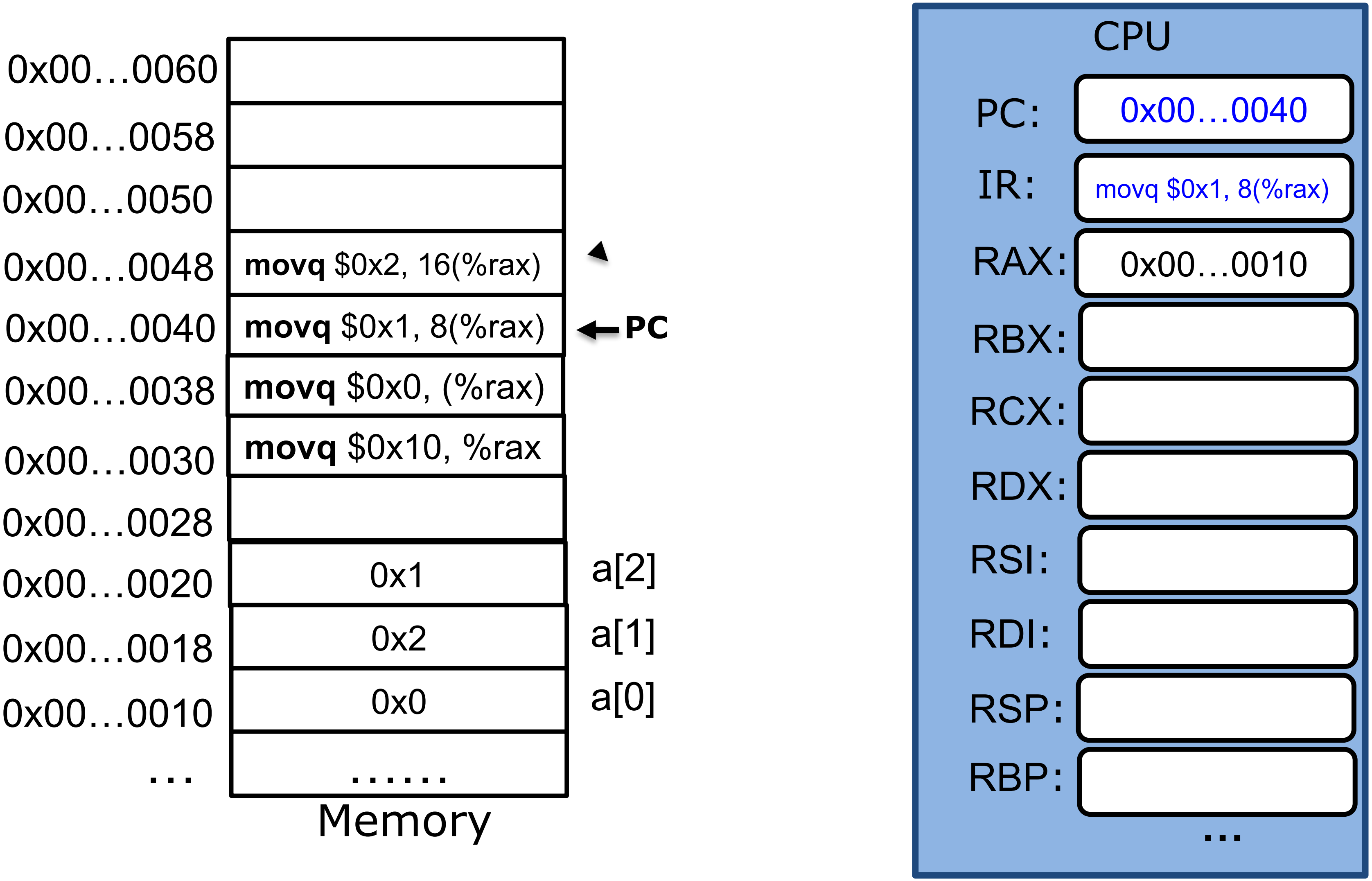
0x00…0060		
0x00…0058		
0x00…0050		
0x00…0048	**movq** $0x2, 16(%rax)	
0x00…0040	**movq** $0x1, 8(%rax)	
0x00…0038	**movq** $0x0, (%rax)	← **PC**
0x00…0030	**movq** $0x10, %rax	
0x00…0028		
0x00…0020	0x1	a[2]
0x00…0018	0x2	a[1]
0x00…0010	0x0	a[0]
…	……	

Memory

CPU

PC:	0x00…0038
IR:	movq $0x0, (%rax)
RAX:	0x00…0010
RBX:	
RCX:	
RDX:	
RSI:	
RDI:	
RSP:	
RBP:	
	…

Example



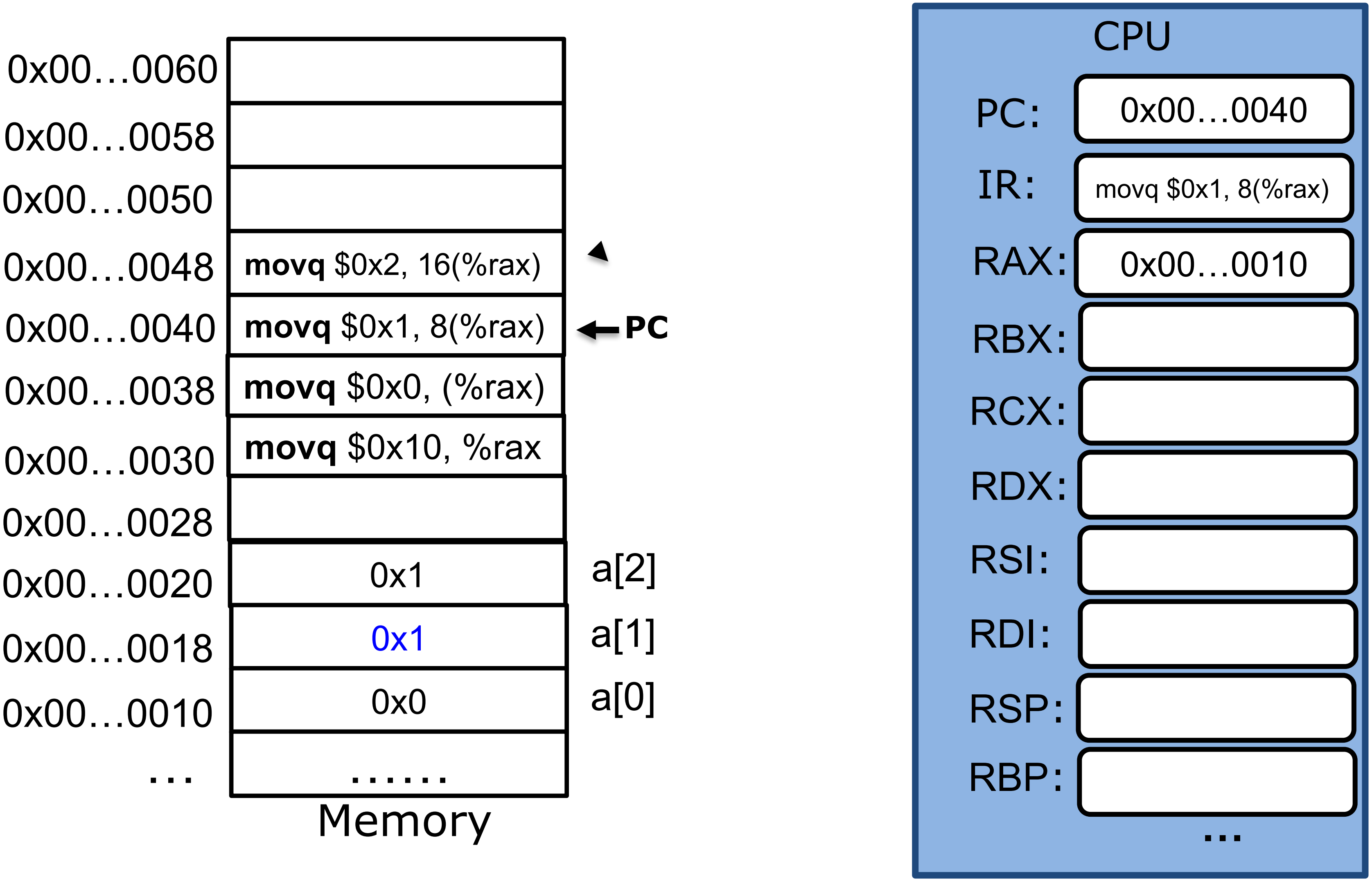
Address	Memory	
0x00…0060		
0x00…0058		
0x00…0050		
0x00…0048	**movq** $0x2, 16(%rax)	
0x00…0040	**movq** $0x1, 8(%rax)	← **PC**
0x00…0038	**movq** $0x0, (%rax)	
0x00…0030	**movq** $0x10, %rax	
0x00…0028		
0x00…0020	0x1	a[2]
0x00…0018	0x2	a[1]
0x00…0010	0x0	a[0]
…	……	

Memory

CPU

PC:	0x00…0040
IR:	movq $0x1, 8(%rax)
RAX:	0x00…0010
RBX:	
RCX:	
RDX:	
RSI:	
RDI:	
RSP:	
RBP:	
	…

Example



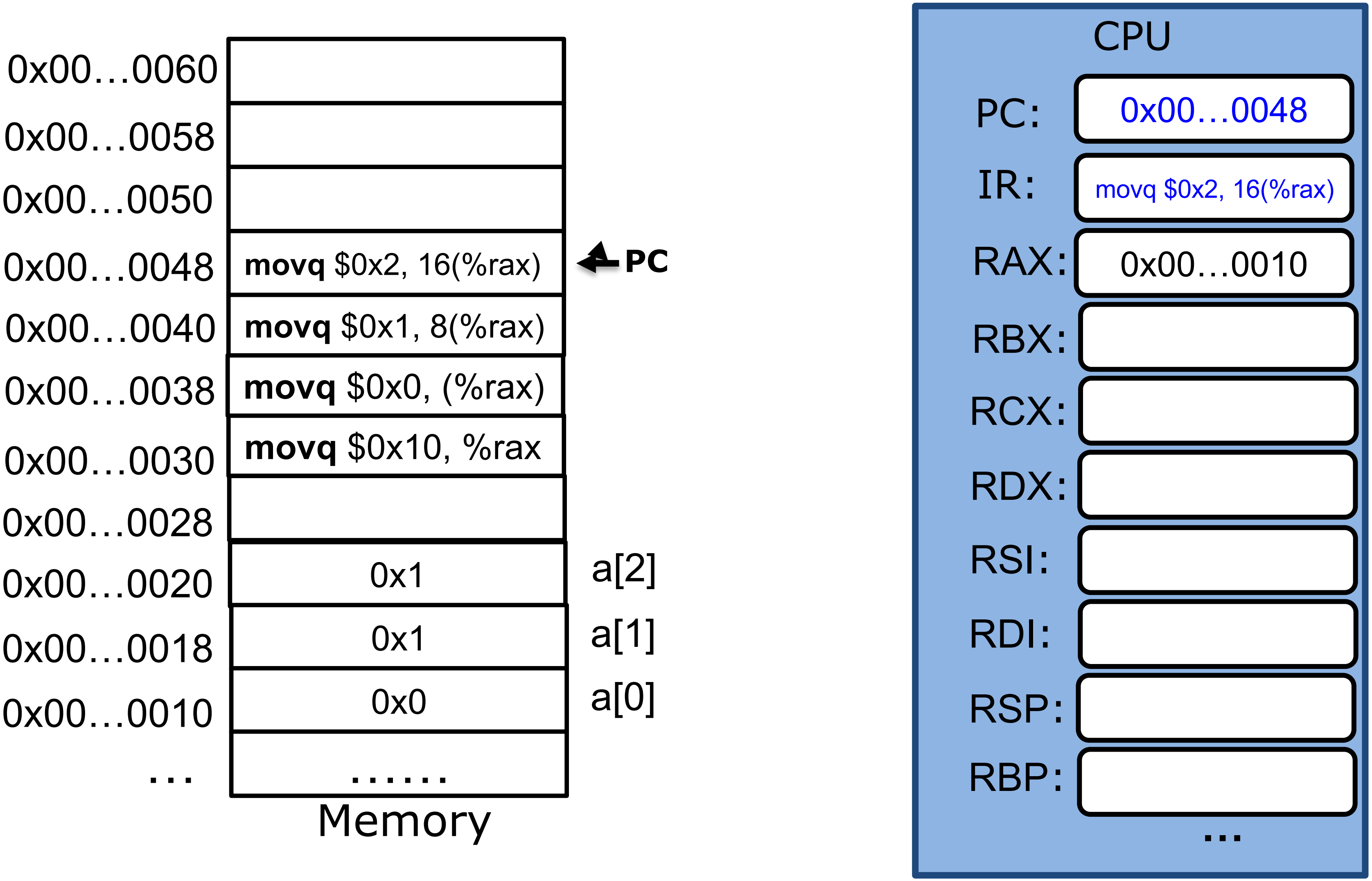
0x00…0060		
0x00…0058		
0x00…0050		
0x00…0048	**movq** $0x2, 16(%rax)	
0x00…0040	**movq** $0x1, 8(%rax)	← **PC**
0x00…0038	**movq** $0x0, (%rax)	
0x00…0030	**movq** $0x10, %rax	
0x00…0028		
0x00…0020	0x1	a[2]
0x00…0018	0x1	a[1]
0x00…0010	0x0	a[0]
…	……	

Memory

CPU

PC:	0x00…0040
IR:	movq $0x1, 8(%rax)
RAX:	0x00…0010
RBX:	
RCX:	
RDX:	
RSI:	
RDI:	
RSP:	
RBP:	
	…

Example



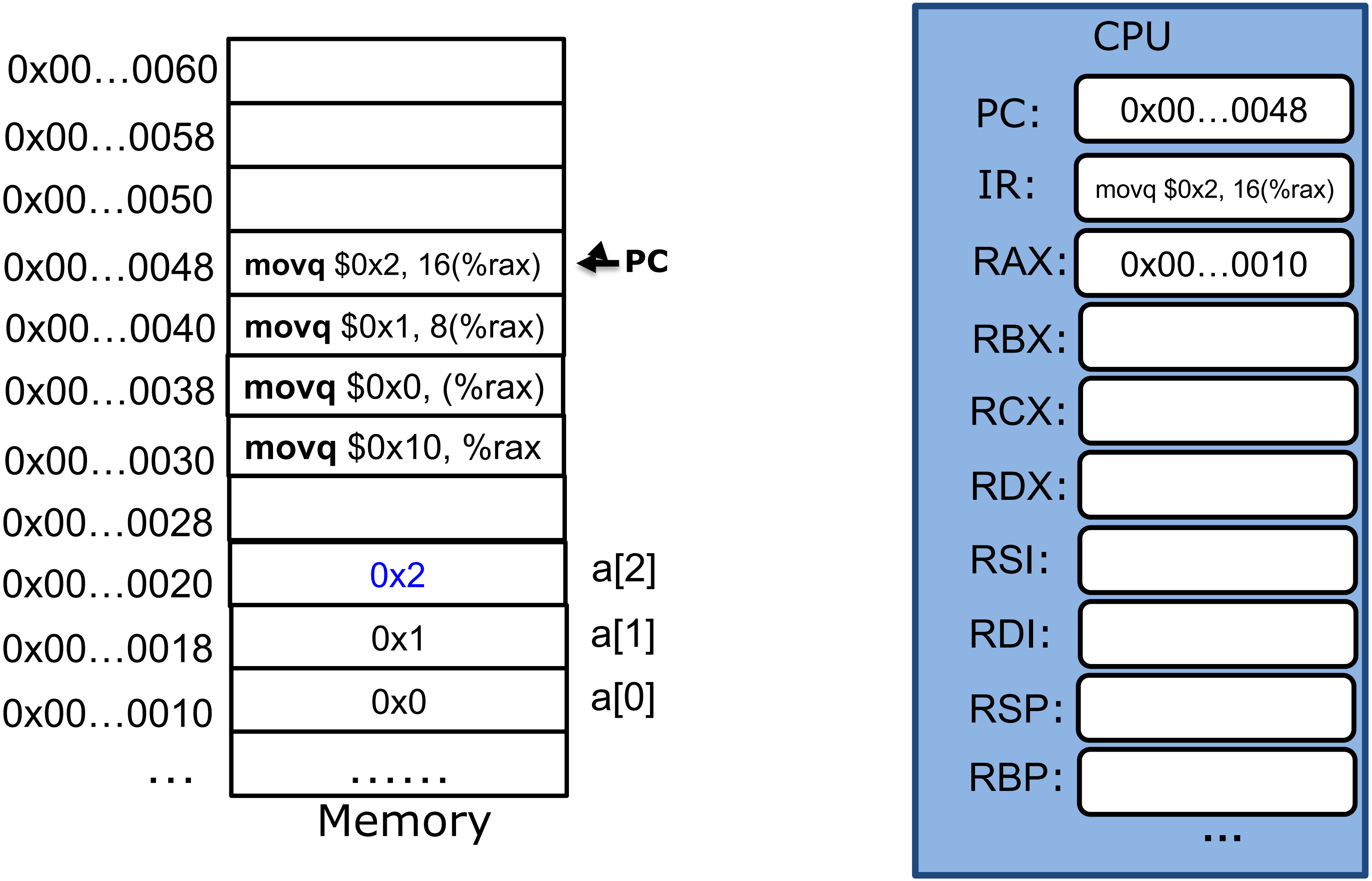
0x00…0060		
0x00…0058		
0x00…0050		
0x00…0048	**movq** $0x2, 16(%rax)	← **PC**
0x00…0040	**movq** $0x1, 8(%rax)	
0x00…0038	**movq** $0x0, (%rax)	
0x00…0030	**movq** $0x10, %rax	
0x00…0028		
0x00…0020	0x1	a[2]
0x00…0018	0x1	a[1]
0x00…0010	0x0	a[0]
…	……	

Memory

CPU

PC:	0x00…0048
IR:	movq $0x2, 16(%rax)
RAX:	0x00…0010
RBX:	
RCX:	
RDX:	
RSI:	
RDI:	
RSP:	
RBP:	
	…

Example



0x00…0060			
0x00…0058			
0x00…0050			
0x00…0048	**movq** $0x2, 16(%rax)		← **PC**
0x00…0040	**movq** $0x1, 8(%rax)		
0x00…0038	**movq** $0x0, (%rax)		
0x00…0030	**movq** $0x10, %rax		
0x00…0028			
0x00…0020	0x2	a[2]	
0x00…0018	0x1	a[1]	
0x00…0010	0x0	a[0]	
…	……		

Memory

CPU

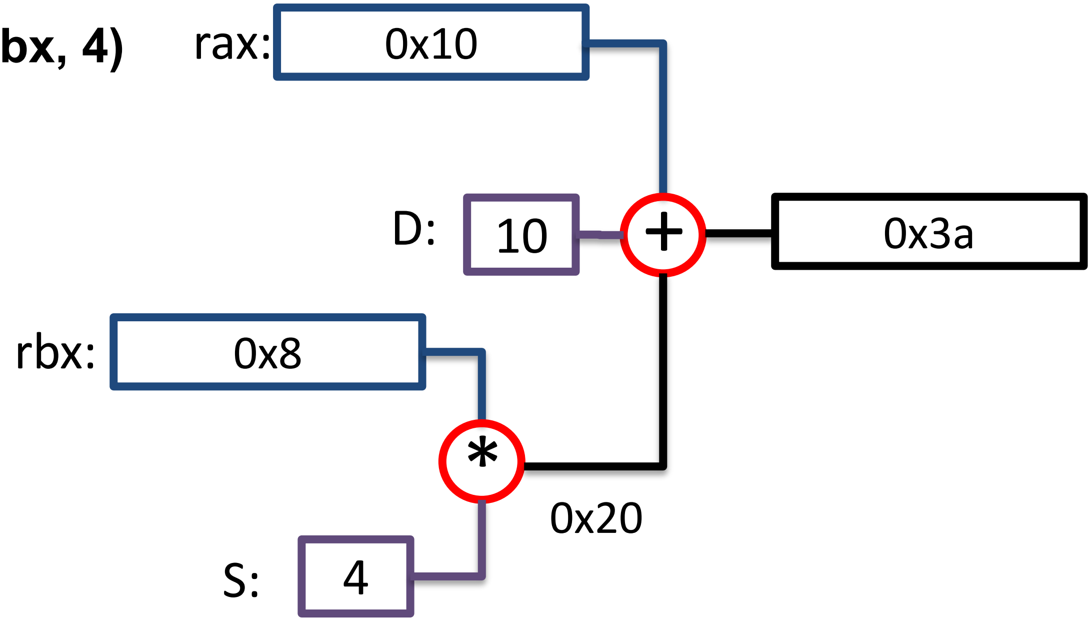
PC:	0x00…0048
IR:	movq $0x2, 16(%rax)
RAX:	0x00…0010
RBX:	
RCX:	
RDX:	
RSI:	
RDI:	
RSP:	
RBP:	
	…

Complete Memory Addressing Mode

D(Rb, Ri, S): val(Rb) + S * val(Ri) + D

- Rb: Base register
- D: Constant “displacement”
- Ri: Index register (not `%rsp`)
- S: Scale: 1, 2, 4, or 8

10(%rax, %rbx, 4)



Complete Memory Addressing Mode

D(Rb, Ri, S): val(Rb) + S * val(Ri) + D

- D: Constant “displacement”
- Rb: Base register
- Ri: Index register (not `%rsp`)
- S: Scale: 1, 2, 4, or 8

If S is 1 or D is 0, we can just get rid of them

- (Rb, Ri): val(Rb) + val(Ri)
- D(Rb, Ri): val(Rb) + val(Ri) + D
- (Rb, Ri, S): val(Rb) + S * val(Ri)

Address Computation Examples

`%rdx`	`0xf000`
`%rcx`	`0x100`

Expression	Address Computation	Address
`0x8(%rdx)`		
`(%rdx,%rcx)`		
`(%rdx,%rcx,4)`		
`0x80(,%rdx,2)`		

Address Computation Examples

%rdx	0xf000
%rcx	0x100

Expression	Address Computation	Address
0x8(%rdx)	0xf000 + 0x8	0xf008
(%rdx,%rcx)		
(%rdx,%rcx,4)		
0x80(,%rdx,2)		

Address Computation Examples

`%rdx`	`0xf000`
`%rcx`	`0x100`

Expression	Address Computation	Address
`0x8(%rdx)`	`0xf000 + 0x8`	`0xf008`
`(%rdx,%rcx)`	`0xf000 + 0x100`	`0xf100`
`(%rdx,%rcx,4)`		
`0x80(,%rdx,2)`		

Address Computation Examples

`%rdx`	`0xf000`
`%rcx`	`0x100`

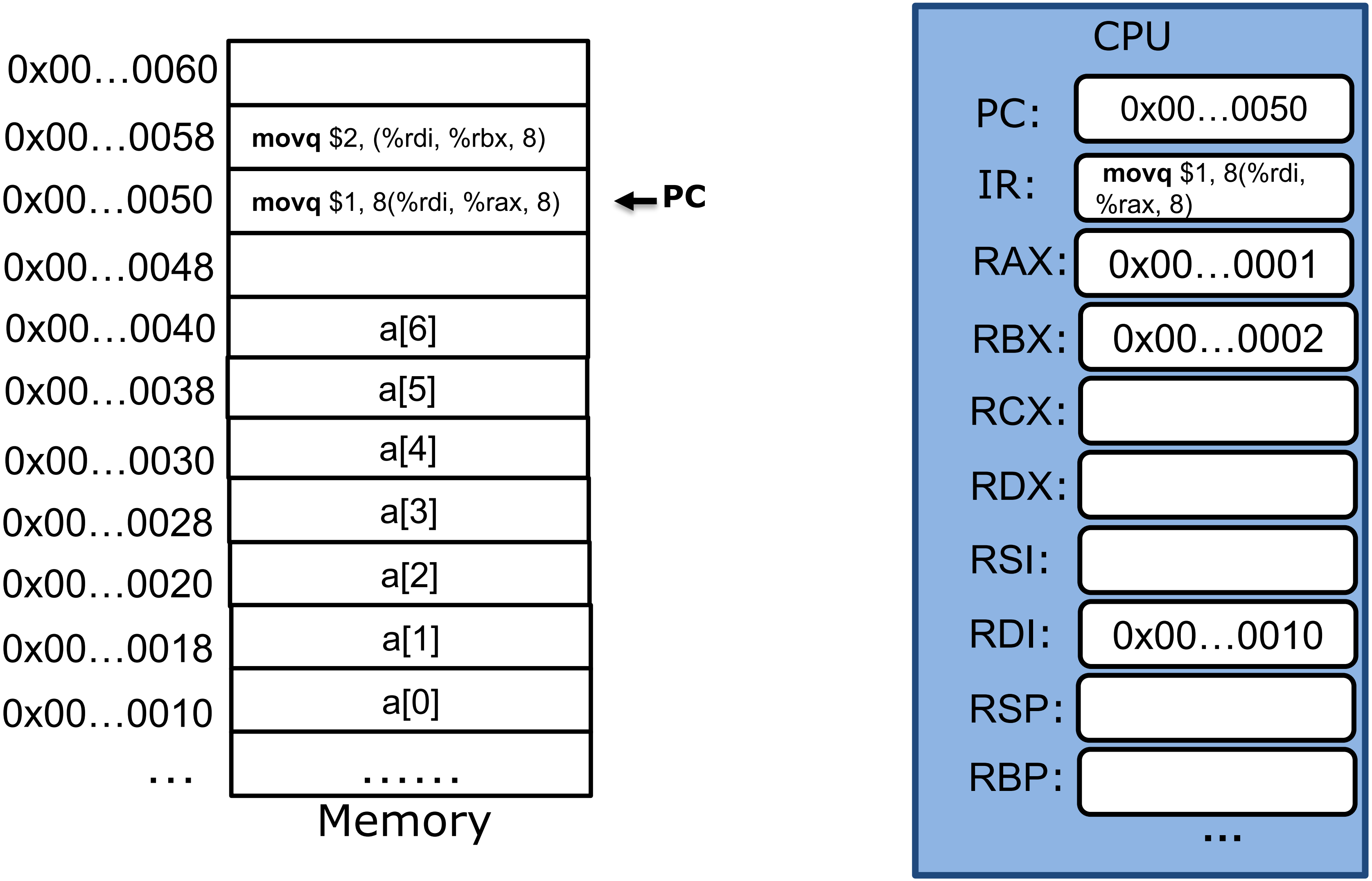
Expression	Address Computation	Address
`0x8(%rdx)`	`0xf000 + 0x8`	`0xf008`
`(%rdx,%rcx)`	`0xf000 + 0x100`	`0xf100`
`(%rdx,%rcx,4)`	`0xf000 + 4*0x100`	`0xf400`
`0x80(,%rdx,2)`		

Address Computation Examples

`%rdx`	`0xf000`
`%rcx`	`0x100`

Expression	Address Computation	Address
`0x8(%rdx)`	`0xf000 + 0x8`	`0xf008`
`(%rdx,%rcx)`	`0xf000 + 0x100`	`0xf100`
`(%rdx,%rcx,4)`	`0xf000 + 4*0x100`	`0xf400`
`0x80(,%rdx,2)`	`2*0xf000 + 0x80`	`0x1e080`

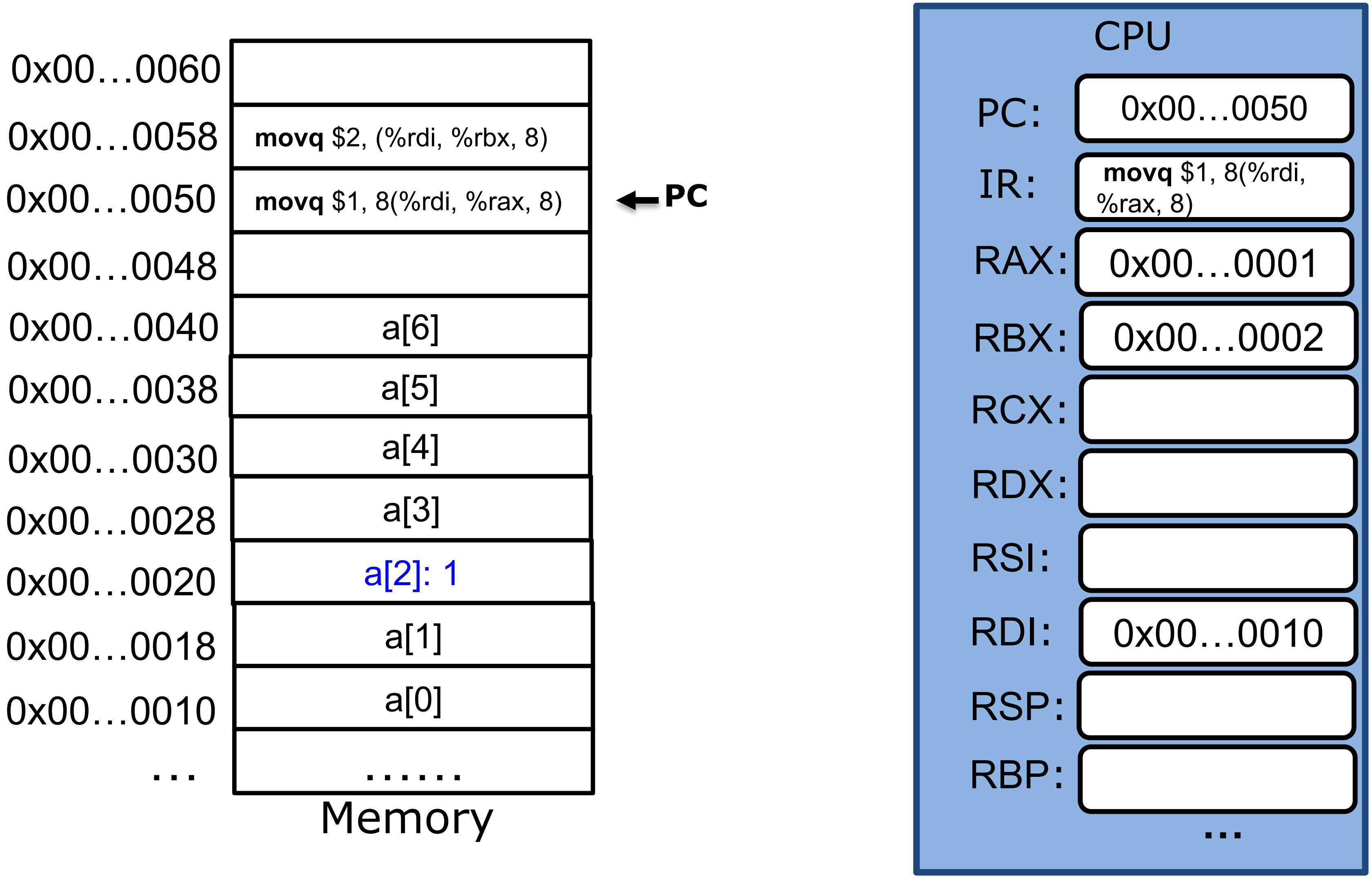
Example



	Memory
0x00…0060	
0x00…0058	**movq** $2, (%rdi, %rbx, 8)
0x00…0050	**movq** $1, 8(%rdi, %rax, 8) ← **PC**
0x00…0048	
0x00…0040	a[6]
0x00…0038	a[5]
0x00…0030	a[4]
0x00…0028	a[3]
0x00…0020	a[2]
0x00…0018	a[1]
0x00…0010	a[0]
…	……

CPU	
PC:	0x00…0050
IR:	**movq** $1, 8(%rdi, %rax, 8)
RAX:	0x00…0001
RBX:	0x00…0002
RCX:	
RDX:	
RSI:	
RDI:	0x00…0010
RSP:	
RBP:	
	…

Example



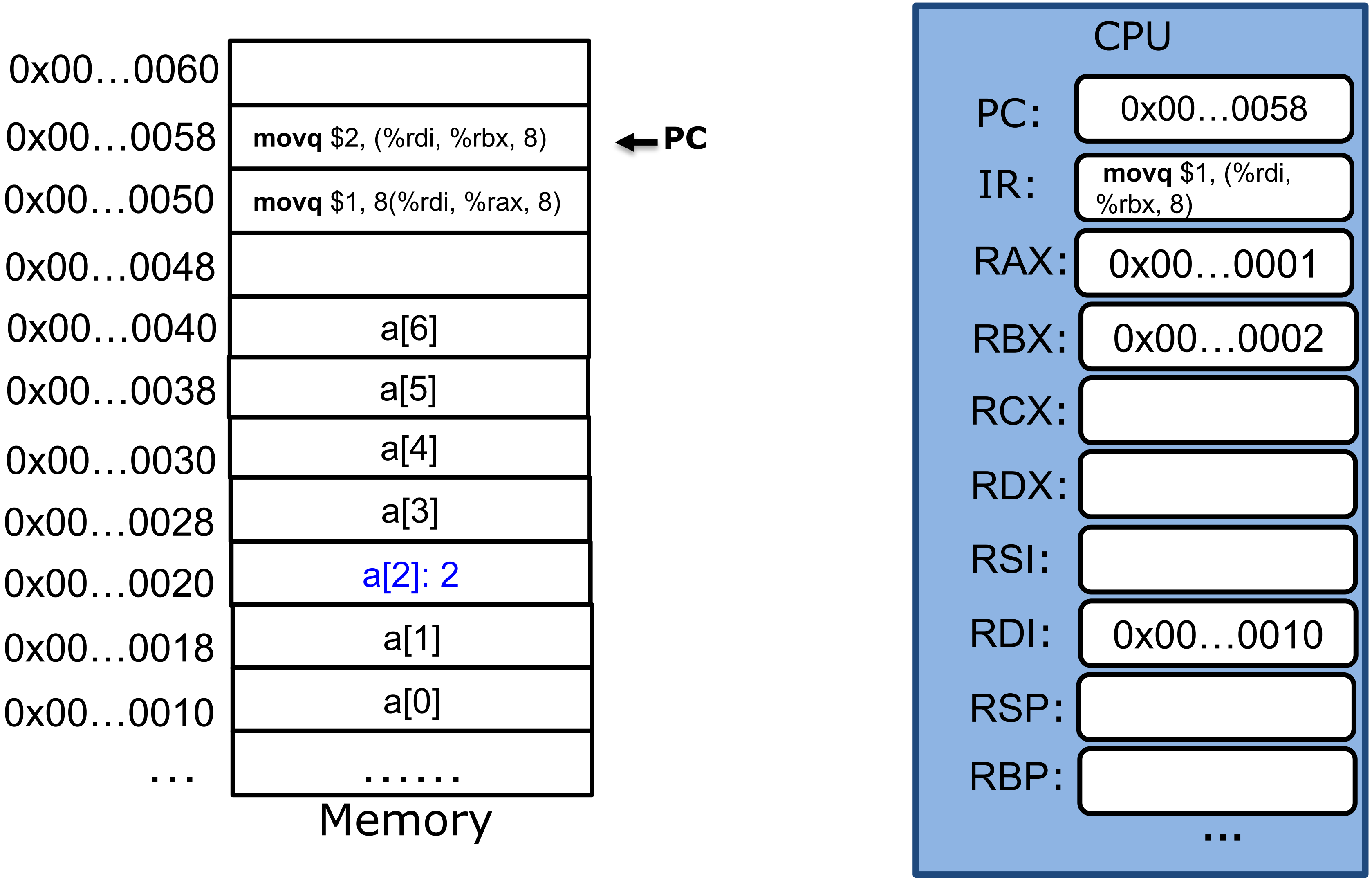
0x00…0060	
0x00…0058	**movq** $2, (%rdi, %rbx, 8)
0x00…0050	**movq** $1, 8(%rdi, %rax, 8)
0x00…0048	
0x00…0040	a[6]
0x00…0038	a[5]
0x00…0030	a[4]
0x00…0028	a[3]
0x00…0020	a[2]: 1
0x00…0018	a[1]
0x00…0010	a[0]
…	……

Memory

CPU

PC:	0x00…0050
IR:	**movq** $1, 8(%rdi, %rax, 8)
RAX:	0x00…0001
RBX:	0x00…0002
RCX:	
RDX:	
RSI:	
RDI:	0x00…0010
RSP:	
RBP:	
	…

Example



0x00…0060	
0x00…0058	**movq** $2, (%rdi, %rbx, 8) ← **PC**
0x00…0050	**movq** $1, 8(%rdi, %rax, 8)
0x00…0048	
0x00…0040	a[6]
0x00…0038	a[5]
0x00…0030	a[4]
0x00…0028	a[3]
0x00…0020	a[2]: 2
0x00…0018	a[1]
0x00…0010	a[0]
…	……

Memory

CPU

PC:	0x00…0058
IR:	**movq** $1, (%rdi, %rbx, 8)
RAX:	0x00…0001
RBX:	0x00…0002
RCX:	
RDX:	
RSI:	
RDI:	0x00…0010
RSP:	
RBP:	
…	

mov{bwlq}

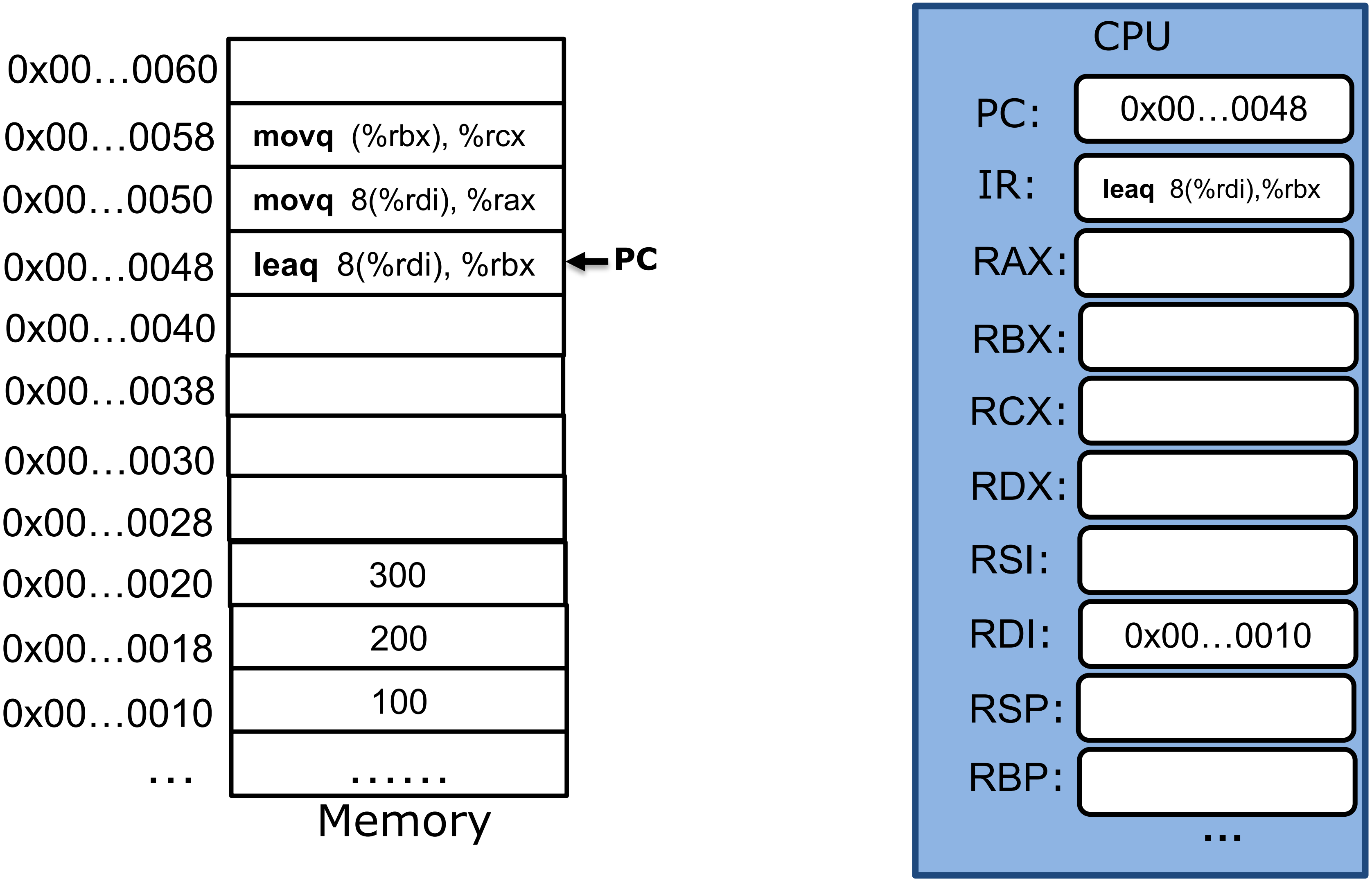
movb src, dest	Copy a **byte** from the source operand to the destination operand. e.g., **movb** %al, %bl
movw src, dest	Copy a **word** from the source operand to the destination operand. e.g., **movw** %ax, %bx
movl src, dest	Copy a **long** (32 bits) from the source operand to the destination operand. e.g., **movl** %eax, %ebx
movq src, dest	Copy a **quadword** from the source operand to the destination operand. e.g., **movq** %rax, %rbx

The lea instruction

leaq *Source, Dest*

- load effective address: set *Dest* to the address denoted by *Source* address mode expression

Example



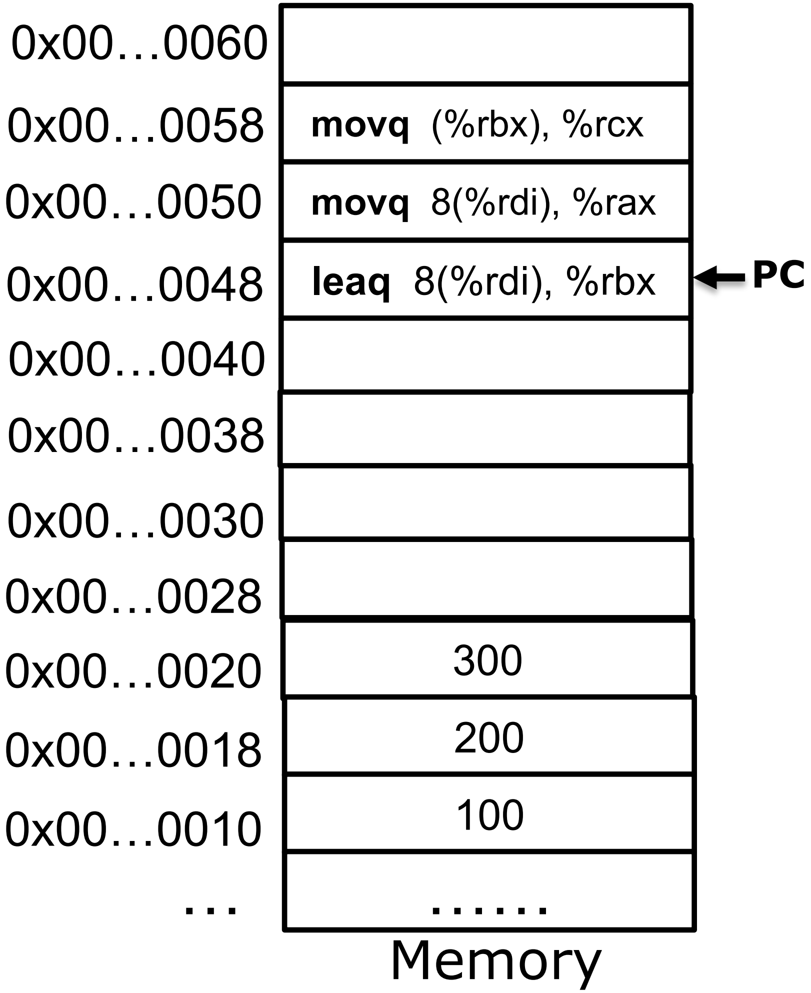
0x00…0060	
0x00…0058	**movq** (%rbx), %rcx
0x00…0050	**movq** 8(%rdi), %rax
0x00…0048	**leaq** 8(%rdi), %rbx ← **PC**
0x00…0040	
0x00…0038	
0x00…0030	
0x00…0028	
0x00…0020	300
0x00…0018	200
0x00…0010	100
…	……

Memory

CPU

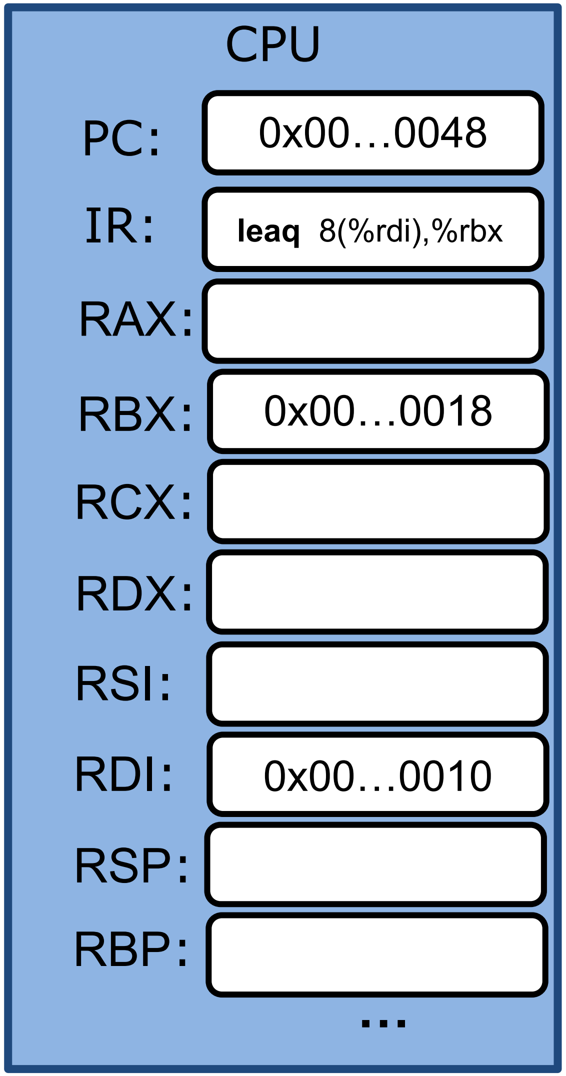
PC:	0x00…0048
IR:	**leaq** 8(%rdi),%rbx
RAX:	
RBX:	
RCX:	
RDX:	
RSI:	
RDI:	0x00…0010
RSP:	
RBP:	
	…

Example



0x00…0060	
0x00…0058	**movq** (%rbx), %rcx
0x00…0050	**movq** 8(%rdi), %rax
0x00…0048	**leaq** 8(%rdi), %rbx
0x00…0040	
0x00…0038	
0x00…0030	
0x00…0028	
0x00…0020	300
0x00…0018	200
0x00…0010	100
…	……

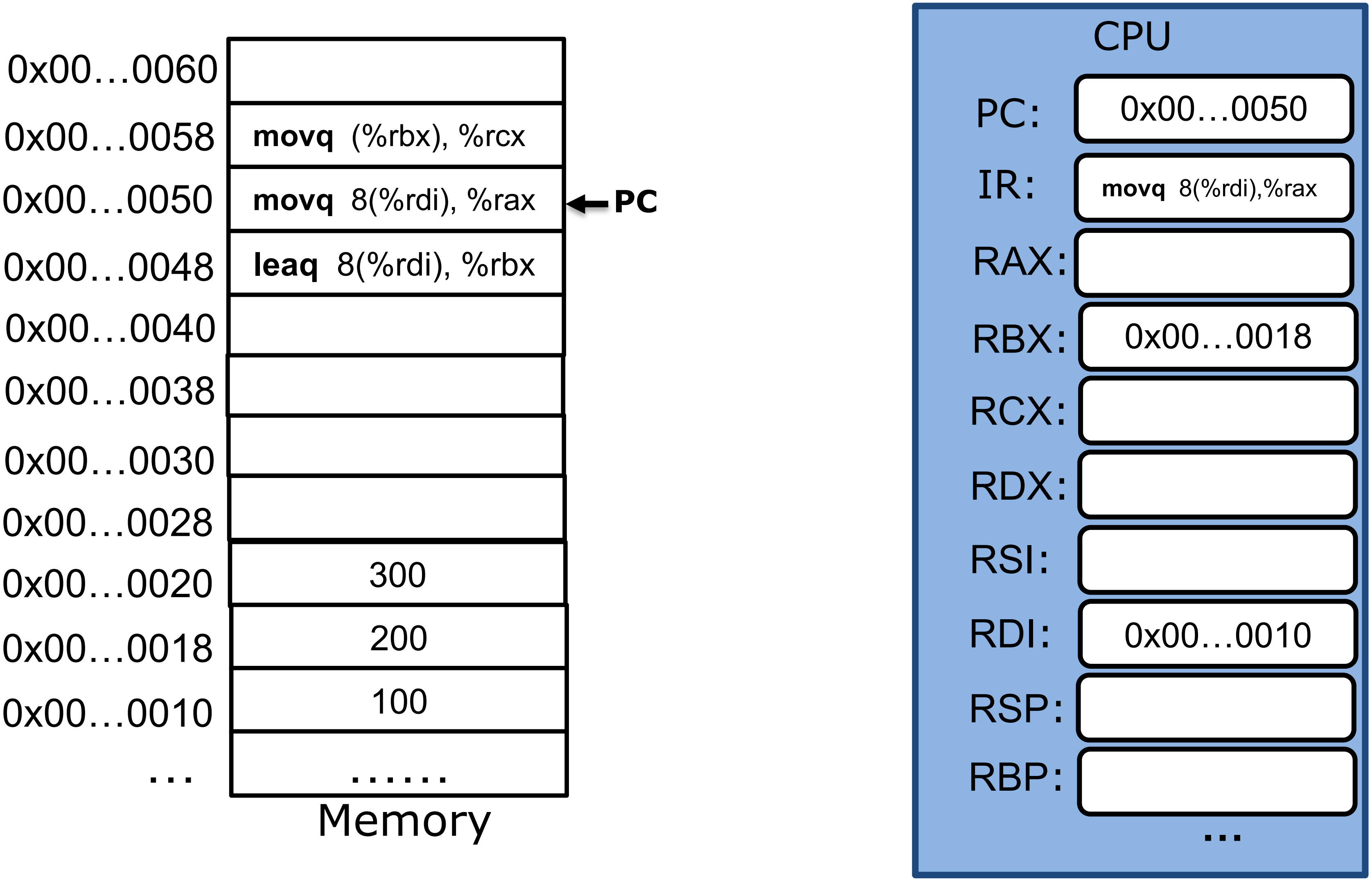
Memory



CPU

PC:	0x00…0048
IR:	**leaq** 8(%rdi),%rbx
RAX:	
RBX:	0x00…0018
RCX:	
RDX:	
RSI:	
RDI:	0x00…0010
RSP:	
RBP:	
	…

Example



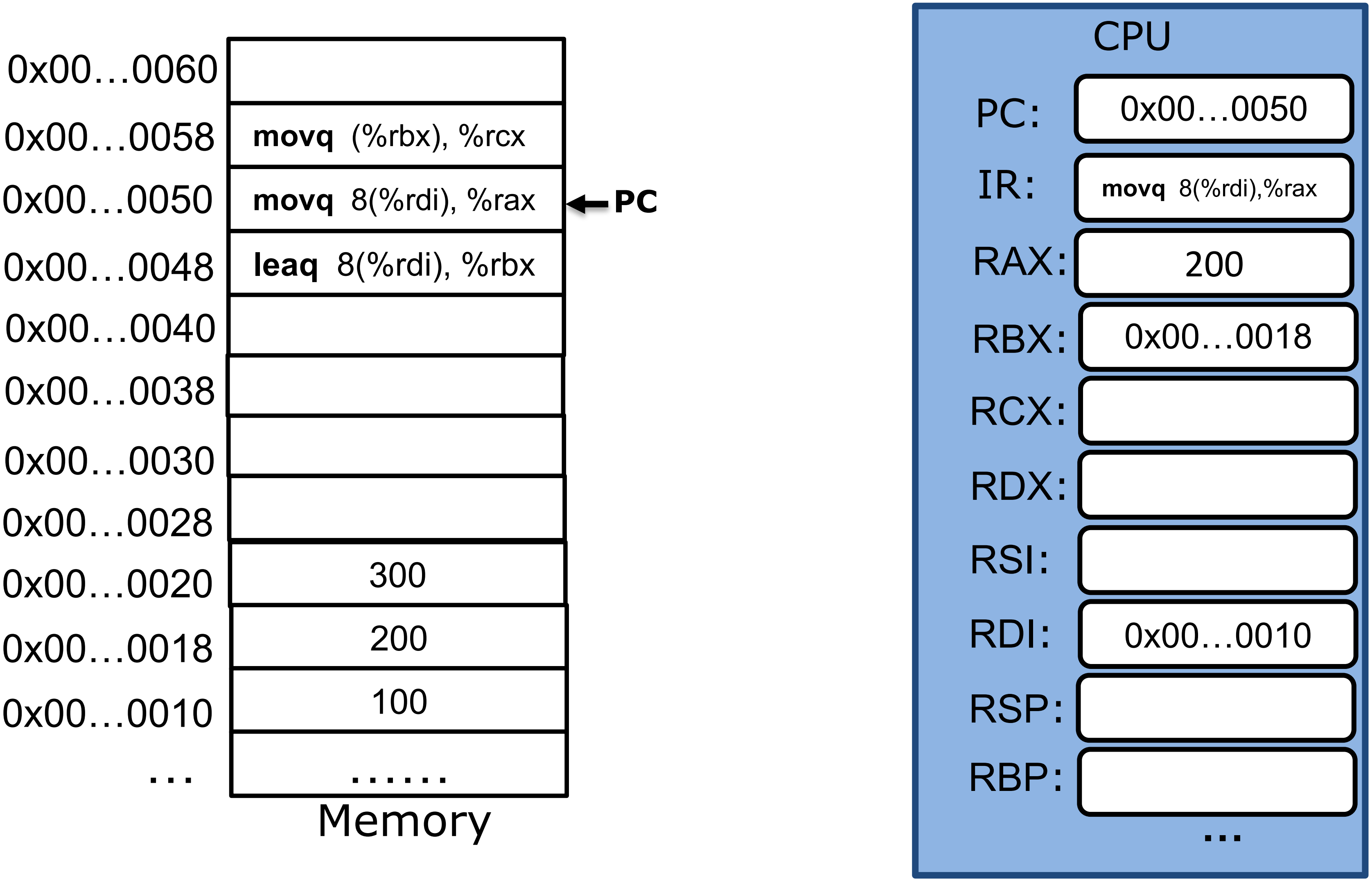
0x00…0060	
0x00…0058	**movq** (%rbx), %rcx
0x00…0050	**movq** 8(%rdi), %rax ← **PC**
0x00…0048	**leaq** 8(%rdi), %rbx
0x00…0040	
0x00…0038	
0x00…0030	
0x00…0028	
0x00…0020	300
0x00…0018	200
0x00…0010	100
…	……

Memory

CPU

PC:	0x00…0050
IR:	**movq** 8(%rdi),%rax
RAX:	
RBX:	0x00…0018
RCX:	
RDX:	
RSI:	
RDI:	0x00…0010
RSP:	
RBP:	
	…

Example



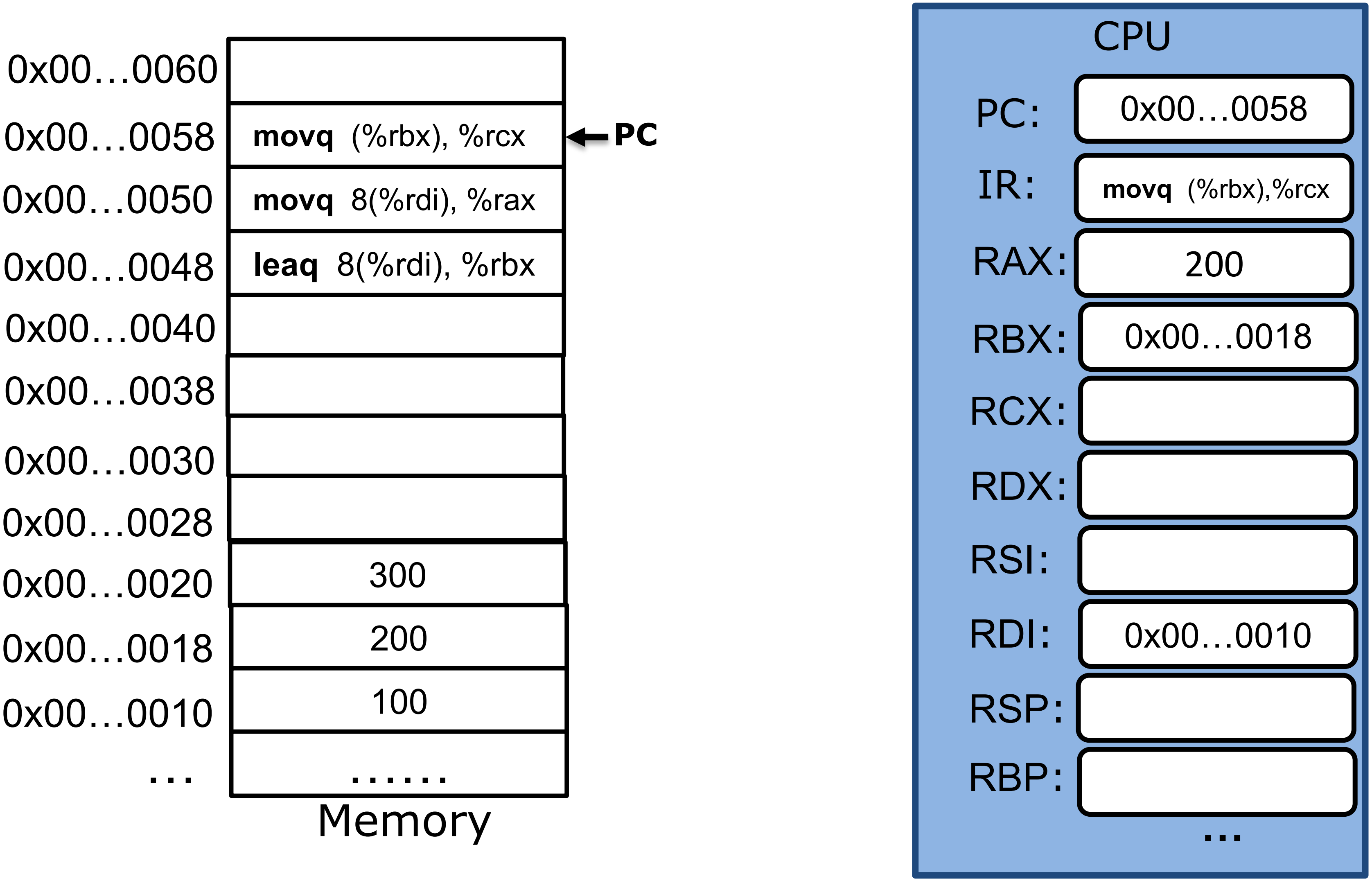
0x00…0060	
0x00…0058	**movq** (%rbx), %rcx
0x00…0050	**movq** 8(%rdi), %rax ← **PC**
0x00…0048	**leaq** 8(%rdi), %rbx
0x00…0040	
0x00…0038	
0x00…0030	
0x00…0028	
0x00…0020	300
0x00…0018	200
0x00…0010	100
…	……

Memory

CPU

PC:	0x00…0050
IR:	**movq** 8(%rdi),%rax
RAX:	200
RBX:	0x00…0018
RCX:	
RDX:	
RSI:	
RDI:	0x00…0010
RSP:	
RBP:	
	…

Example



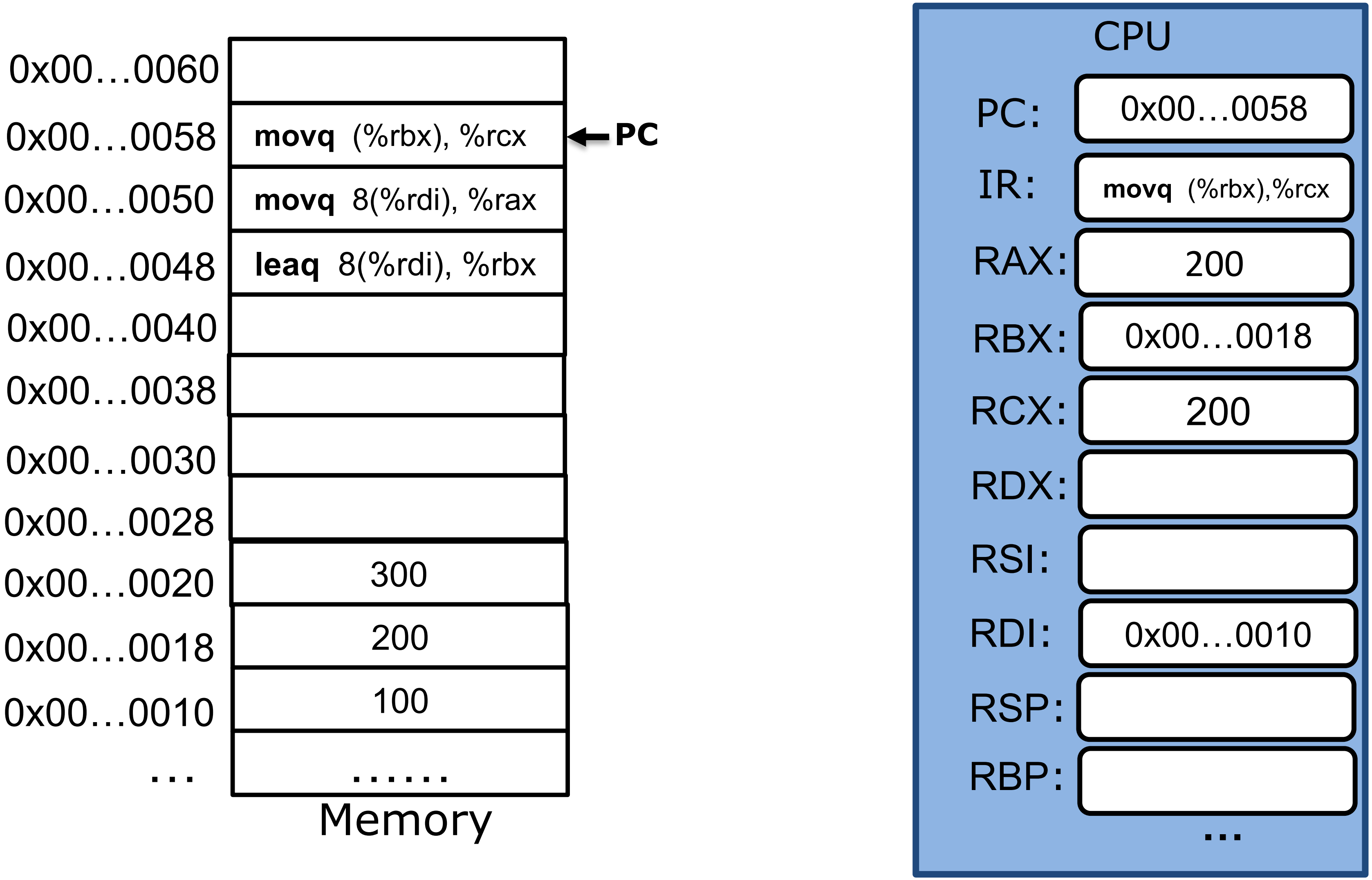
0x00…0060		
0x00…0058	**movq** (%rbx), %rcx	← **PC**
0x00…0050	**movq** 8(%rdi), %rax	
0x00…0048	**leaq** 8(%rdi), %rbx	
0x00…0040		
0x00…0038		
0x00…0030		
0x00…0028		
0x00…0020	300	
0x00…0018	200	
0x00…0010	100	
…	……	

Memory

CPU

PC:	0x00…0058
IR:	**movq** (%rbx),%rcx
RAX:	200
RBX:	0x00…0018
RCX:	
RDX:	
RSI:	
RDI:	0x00…0010
RSP:	
RBP:	
	…

Example



0x00…0060	
0x00…0058	**movq** (%rbx), %rcx ← **PC**
0x00…0050	**movq** 8(%rdi), %rax
0x00…0048	**leaq** 8(%rdi), %rbx
0x00…0040	
0x00…0038	
0x00…0030	
0x00…0028	
0x00…0020	300
0x00…0018	200
0x00…0010	100
…	……

Memory

CPU

PC:	0x00…0058
IR:	**movq** (%rbx),%rcx
RAX:	200
RBX:	0x00…0018
RCX:	200
RDX:	
RSI:	
RDI:	0x00…0010
RSP:	
RBP:	
	…

Other usage of leaq

Computing arithmetic expressions of the form x + k*y + d (k = 1, 2, 4, or 8)

```
long m3(long x)
{
  return x*3;
}
```

→

Assume %rdi has the value of x

Other usage of leaq

Computing arithmetic expressions of the form x + k*y + d (k = 1, 2, 4, or 8)

```
long m3(long x)
{
  return x*3;
}
```

→

```
leaq (%rdi, %rdi,2), %rax
```

Assume %rdi has the value of x

Arithmetic Expression Puzzle

Suppose %rdi, %rsi, %rax contains variable x, y, s respsectively

```
leaq    (%rdi,%rsi,2), %rax
leaq    (%rax,%rax,4), %rax
```

```
long f(long x, long y)
{
    long s = ??;
    return s;
}
```

Arithmetic Expression Puzzle

Suppose %rdi, %rsi, %rax contains variable x, y, s respsectively

```
leaq    (%rdi,%rsi,2), %rax
leaq    (%rax,%rax,4), %rax
```

```
long f(long x, long y)
{
    long s = 5(x + 2y);
    return s;
}
```

Some Arithmetic Operations

Two Operand Instructions:

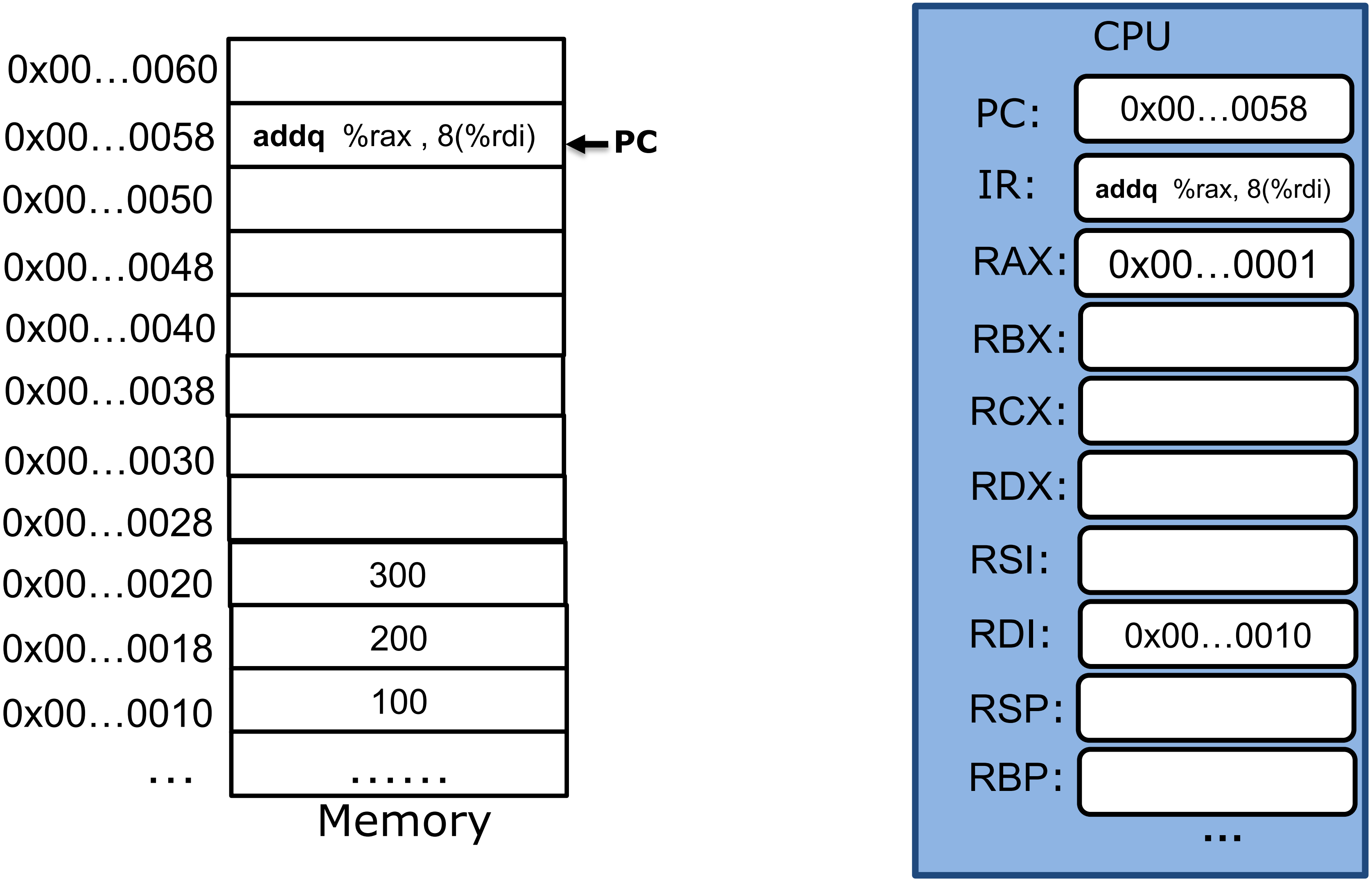
addq	Src,Dest	Dest = Dest + Src	
subq	Src,Dest	Dest = Dest – Src	
imulq	Src,Dest	Dest = Dest * Src	
salq	Src,Dest	Dest = Dest << Src	Also called shlq
sarq	Src,Dest	Dest = Dest >> Src	Arithmetic
shrq	Src,Dest	Dest = Dest >> Src	Logical
xorq	Src,Dest	Dest = Dest ^ Src	
andq	Src,Dest	Dest = Dest & Src	
orq	Src,Dest	Dest = Dest | Src	

Some Arithmetic Operations

One Operand Instructions

incq	Dest	Dest = Dest + 1
decq	Dest	Dest = Dest – 1
negq	Dest	Dest = – Dest
notq	Dest	Dest = ~Dest

Example



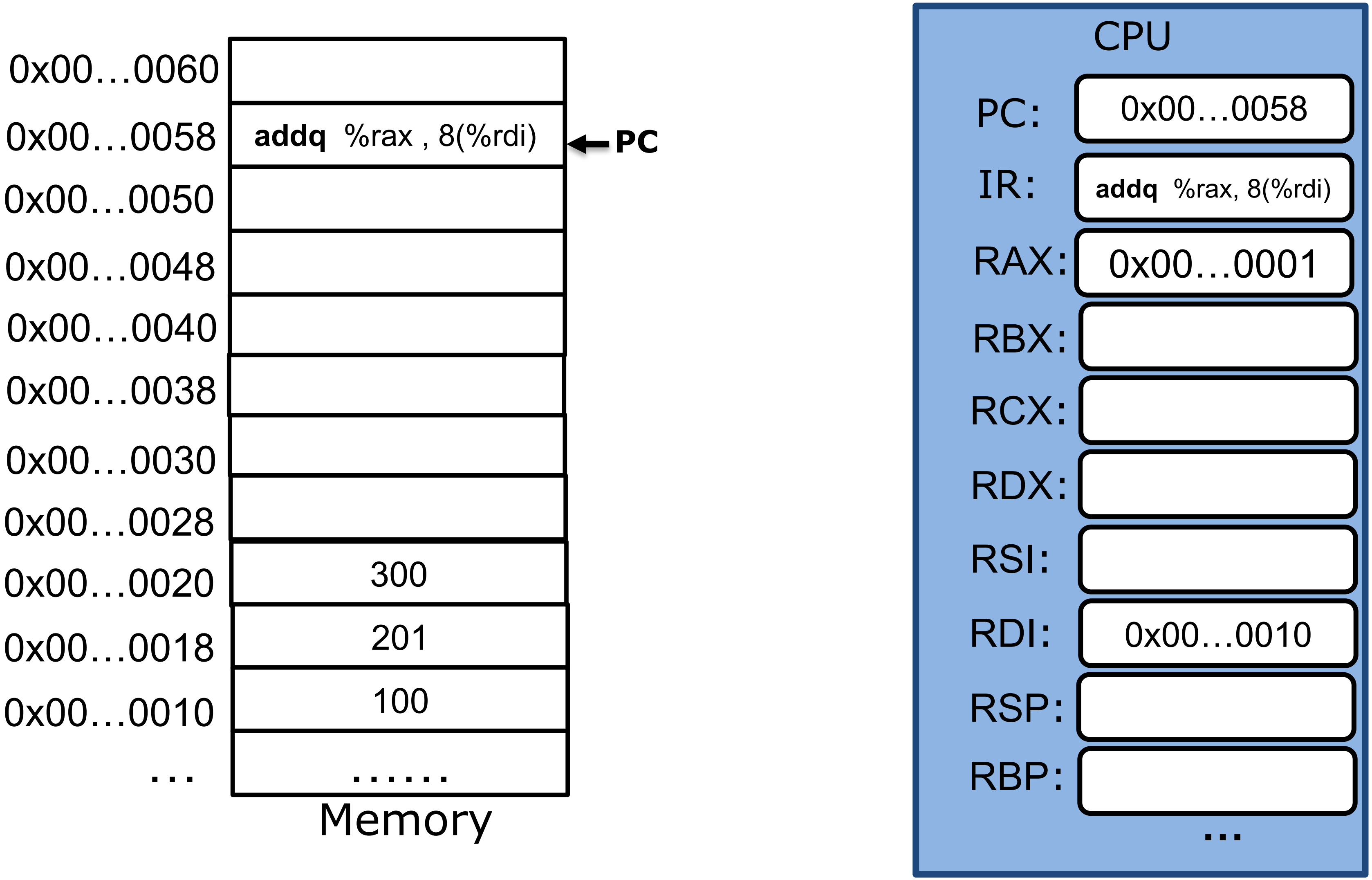
0x00…0060	
0x00…0058	**addq** %rax , 8(%rdi) ← **PC**
0x00…0050	
0x00…0048	
0x00…0040	
0x00…0038	
0x00…0030	
0x00…0028	
0x00…0020	300
0x00…0018	200
0x00…0010	100
…	……

Memory

CPU

PC:	0x00…0058
IR:	**addq** %rax, 8(%rdi)
RAX:	0x00…0001
RBX:	
RCX:	
RDX:	
RSI:	
RDI:	0x00…0010
RSP:	
RBP:	
	…

Example



0x00…0060	
0x00…0058	**addq** %rax , 8(%rdi) ← **PC**
0x00…0050	
0x00…0048	
0x00…0040	
0x00…0038	
0x00…0030	
0x00…0028	
0x00…0020	300
0x00…0018	201
0x00…0010	100
…	……

Memory

CPU

PC:	0x00…0058
IR:	**addq** %rax, 8(%rdi)
RAX:	0x00…0001
RBX:	
RCX:	
RDX:	
RSI:	
RDI:	0x00…0010
RSP:	
RBP:	
	…

Some Arithmetic Operations

Two Operand Instructions:

add{bwlq}	Src,Dest	Dest = Dest + Src	
sub{bwlq}	Src,Dest	Dest = Dest – Src	
imul{bwlq}	Src,Dest	Dest = Dest * Src	
sal{bwlq}	Src,Dest	Dest = Dest << Src	Also called shlq
sar{bwlq}	Src,Dest	Dest = Dest >> Src	Arithmetic
shr{bwlq}	Src,Dest	Dest = Dest >> Src	Logical
xor{bwlq}	Src,Dest	Dest = Dest ^ Src	
and{bwlq}	Src,Dest	Dest = Dest & Src	
or{bwlq}	Src,Dest	Dest = Dest | Src	

Some Arithmetic Operations

One Operand Instructions

inc{bwlq}	Dest	Dest = Dest + 1
dec{bwlq}	Dest	Dest = Dest – 1
neg{bwlq}	Dest	Dest = – Dest
not{bwlq}	Dest	Dest = ~Dest