ISCAS NIST

循序渐进，学习开发一个 RISC-V 上的操作系统

第 1 章 计算机系统漫游

汪辰

2021/4

本章内容

- 计算机的硬件组成
- 程序的存储与执行
- 程序语言的设计和进化
- 存储设备的层次结构
- 操作系统

本章内容

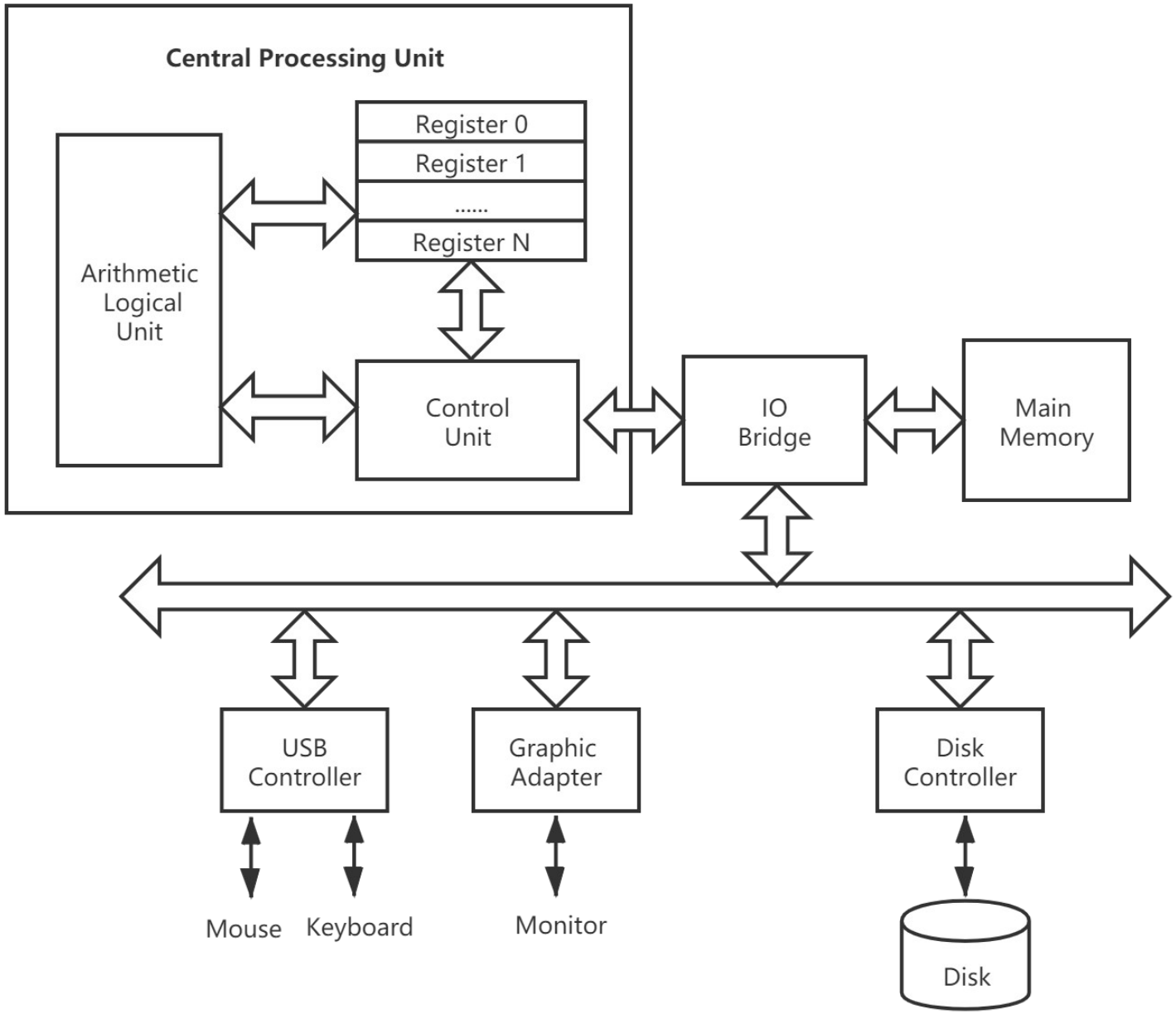
- 计算机的硬件组成
- 程序的存储与执行
- 程序语言的设计和进化
- 存储设备的层次结构
- 操作系统

Hello World!

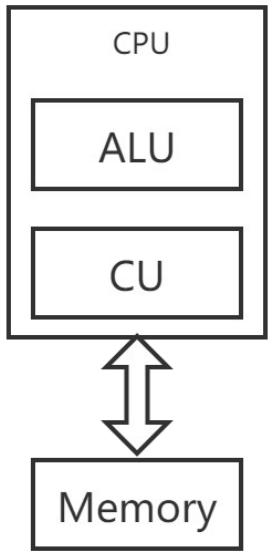
```
#include <stdio.h>

int main()
{
    printf( “hello world!\n” );
    return 0;
}
```

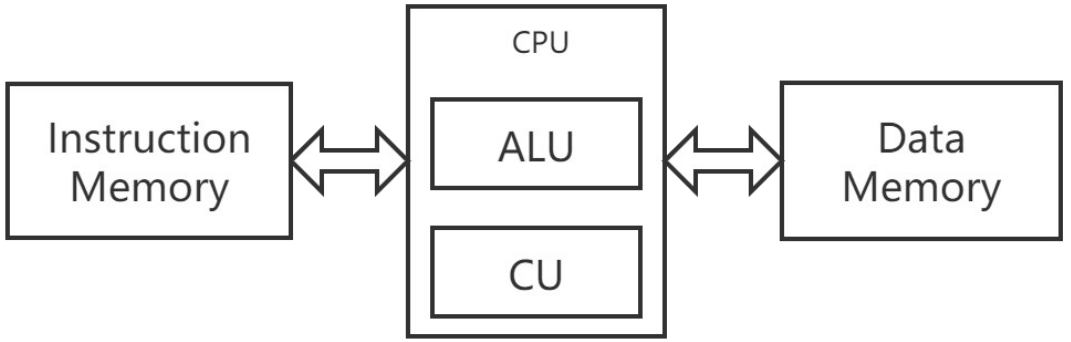
计算机的硬件组成


Central Processing Unit
Register 0
Register 1
.....
Register N
Arithmetic
Logical
Unit
Control
Unit
IO
Bridge
Main
Memory
USB
Controller
Graphic
Adapter
Disk
Controller
Mouse
Keyboard
Monitor
Disk

计算机的硬件组成



冯诺伊曼架构



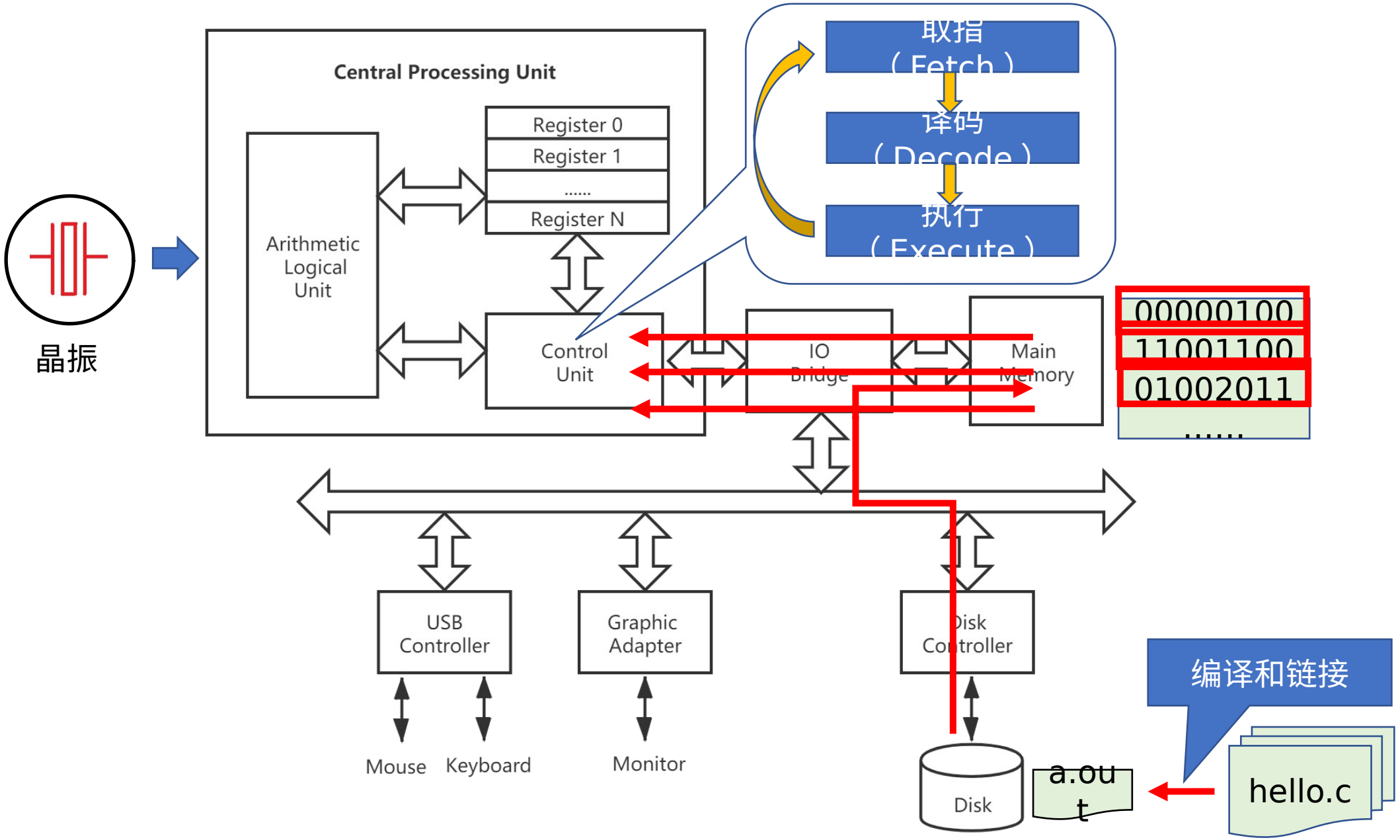
哈佛架构

- **冯．诺依曼架构（Von Neumann architecture）**：又称普林斯顿架构（Princetion architecture），特点是指令和数据不加区别地存储在存储器中，经由同一个总线传输。优点是总线开销小，控制逻辑实现更简单；缺点是执行效率较低。
- **哈佛架构（Harvard architecture）**：特点是将程序指令和数据分开存储。优点是执行效率较高，缺点是总线开销更大，控制逻辑实现更复杂。

本章内容

- 计算机的硬件组成
- **程序的存储与执行**
- 程序语言的设计和进化
- 存储设备的层次结构
- 操作系统

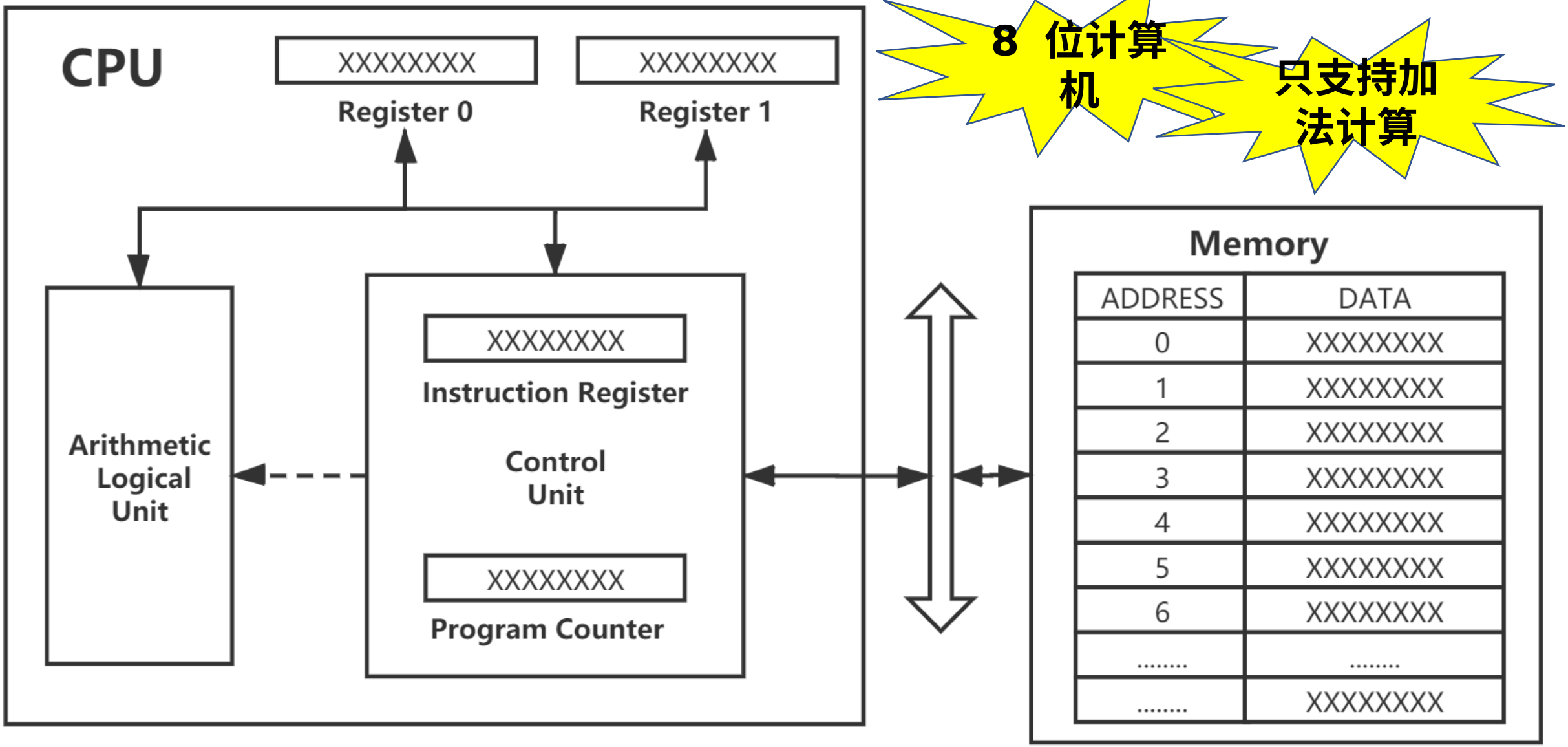
程序的存储与执行


晶振
Central Processing Unit
Arithmetic Logical Unit
Register 0
Register 1
.....
Register N
Control Unit
取指（Fetch）
译码（Decode）
执行（Execute）
IO Bridge
Main Memory
00000100
11001100
01002011
.....
USB Controller
Graphic Adapter
Disk Controller
Mouse
Keyboard
Monitor
Disk
a.out
编译和链接
hello.c

本章内容

- 计算机的硬件组成
- 程序的存储与执行
- **程序语言的设计和进化**
- 存储设备的层次结构
- 操作系统

一个 mini 的计算机


CPU
XXXXXXXX
Register 0
XXXXXXXX
Register 1
Arithmetic Logical Unit
XXXXXXXX
Instruction Register
Control Unit
XXXXXXXX
Program Counter
8 位计算机
只支持加法计算
Memory
ADDRESS
DATA
0
XXXXXXXX
1
XXXXXXXX
2
XXXXXXXX
3
XXXXXXXX
4
XXXXXXXX
5
XXXXXXXX
6
XXXXXXXX
.....
.....
.....
XXXXXXXX

针对 mini 计算机的机器指令设计

Operation Description	OPCODE	OPRANDS	INSTRUCTION
LOAD data from ADDRESS where stores the first value to REGISTER_0	LOAD	REGISTER_0 ADDRESS	
LOAD data from ADDRESS where stores the second value to REGISTER_1	LOAD	REGISTER_1 ADDRESS	
ADD REG0 and REG1, store result in REGISTER_0	ADD	REGISTER_0 REGISTER_1	
STORE value of REGISTER_0 to ADDRESS	STORE	REGISTER_0 ADDRESS	

针对 mini 计算机的机器指令设计

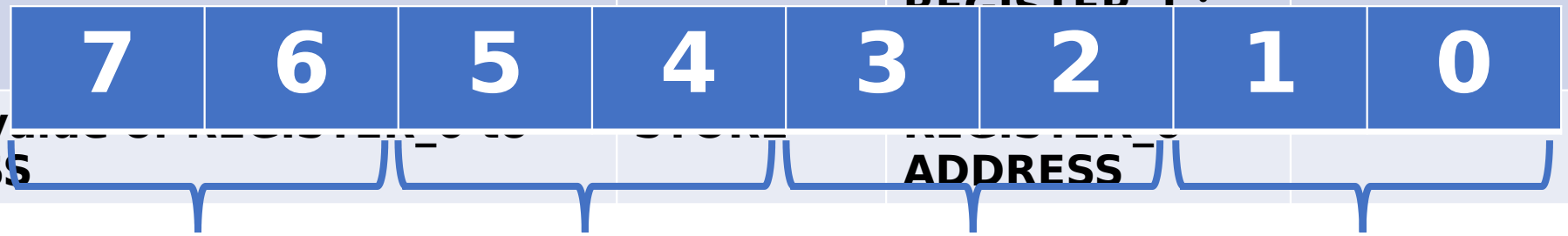
Operation Description	OPCODE	OPRANDS	INSTRUCTION
LOAD data from ADDRESS where stores the first value to REGISTER_0	LOAD : 01	REGISTER_0 : 00 ADDRESS : XXXX	XXXX-00-01
LOAD data from ADDRESS where stores the second value to REGISTER_1	LOAD	REGISTER_1 ADDRESS	
ADD REG0 and REG1, store result in REGISTER_0	ADD	REGISTER_0 REGISTER_1	
STORE value of REGISTER_0 to ADDRESS	STORE	REGISTER_0 ADDRESS	

针对 mini 计算机的机器指令设计

Operation Description	OPCODE	OPRANDS	INSTRUCTION
LOAD data from ADDRESS where stores the first value to REGISTER_0	LOAD : 01	REGISTER_0 : 00 ADDRESS : XXXX	XXXX-00-01
LOAD data from ADDRESS where stores the second value to REGISTER_1	LOAD : 01	REGISTER_1 : 01 ADDRESS : XXXX	XXXX-01-01
ADD REG0 and REG1, store result in REGISTER_0	ADD	REGISTER_0 REGISTER_1	
STORE value of REGISTER_0 to ADDRESS	STORE	REGISTER_0	

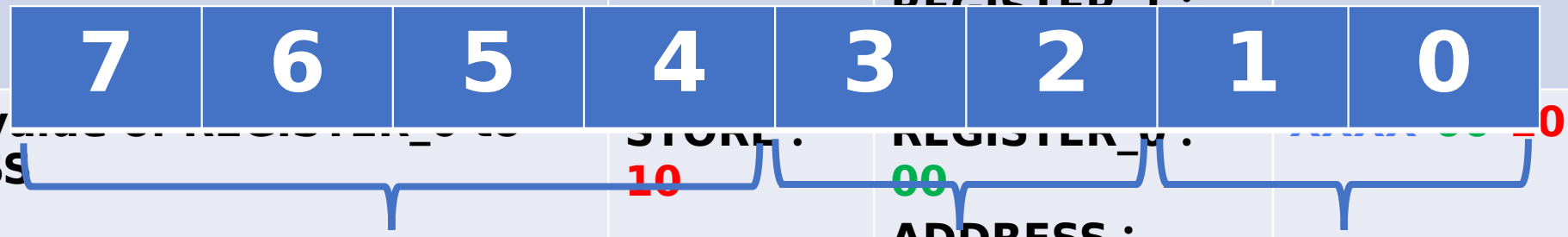
针对 mini 计算机的机器指令设计

Operation Description	OPCODE	OPRANDS	INSTRUCTION
LOAD data from ADDRESS where stores the first value to REGISTER_0	LOAD : 01	REGISTER_0 : 00 ADDRESS : XXXX	XXXX-00-01
LOAD data from ADDRESS where stores the second value to REGISTER_1	LOAD : 01	REGISTER_1 : 01 ADDRESS : XXXX	XXXX-01-01
ADD REG0 and REG1, store result in REGISTER_0	ADD : 11	REGISTER_0 : 00 REGISTER_1 :	NN-01-00-11
STORE value of REGISTER_0 to ADDRESS		ADDRESS	



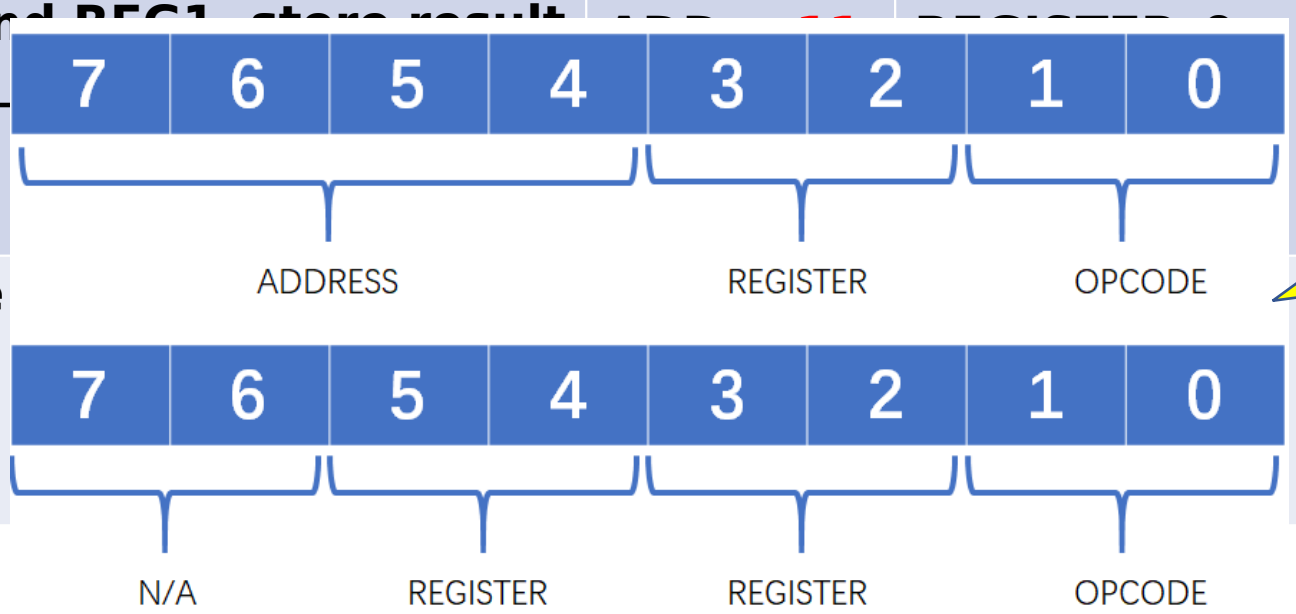
针对 mini 计算机的机器指令设计

Operation Description	OPCODE	OPRANDS	INSTRUCTION
LOAD data from ADDRESS where stores the first value to REGISTER_0	LOAD : 01	REGISTER_0 : 00 ADDRESS : XXXX	XXXX-00-01
LOAD data from ADDRESS where stores the second value to REGISTER_1	LOAD : 01	REGISTER_1 : 01 ADDRESS : XXXX	XXXX-01-01
ADD REG0 and REG1, store result in REGISTER_0	ADD : 11	REGISTER_0 : 00 REGISTER_1 :	NN-01-00-11
STORE value of REGISTER_0 to ADDRESS	STORE : 10	REGISTER_0 : 00 ADDRESS : XXXX	XXXX-00-10



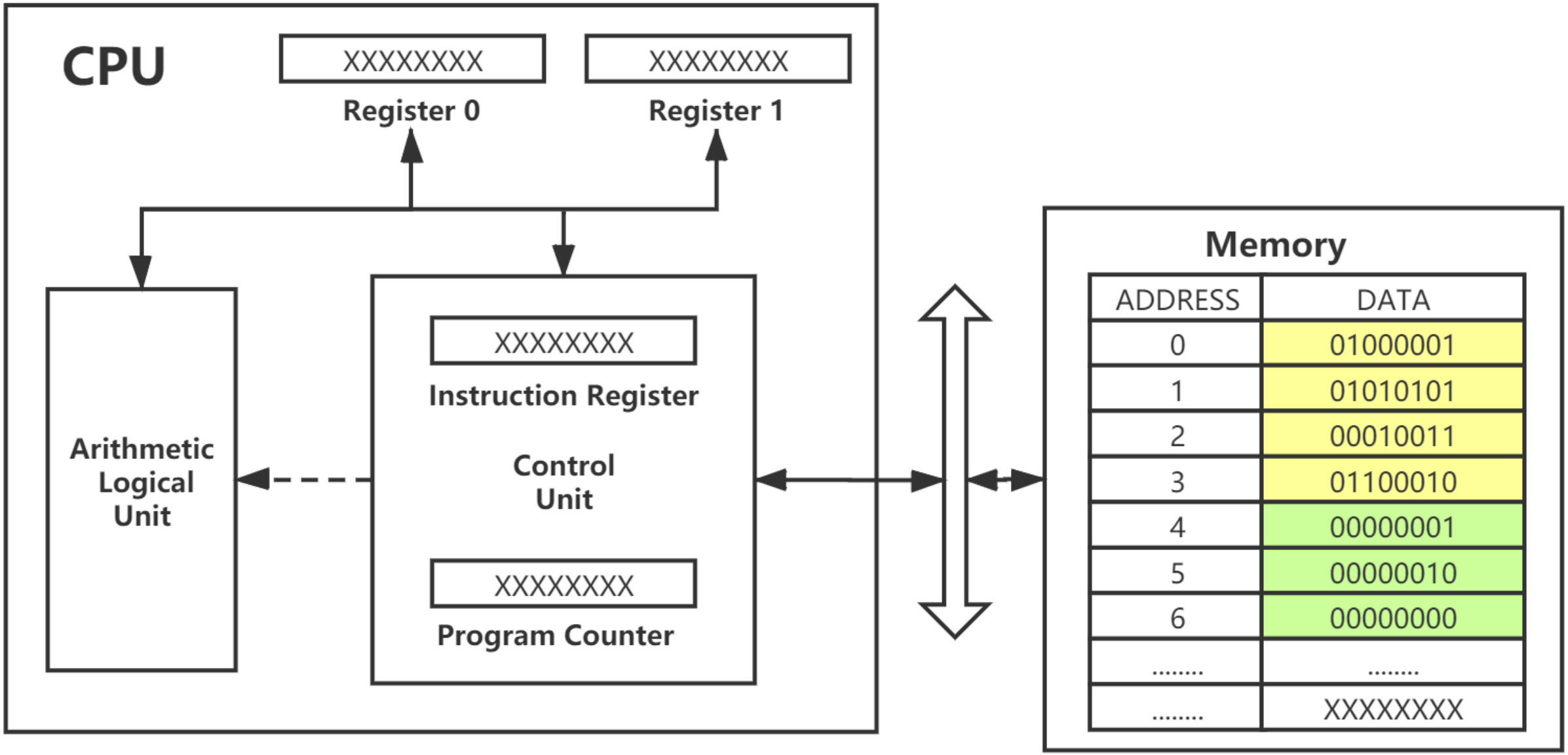
针对 mini 计算机的机器指令设计

Operation Description	OPCODE	OPRANDS	INSTRUCTION
LOAD data from ADDRESS where stores the first value to REGISTER_0	LOAD : 01	REGISTER_0 : 00 ADDRESS : XXXX	XXXX-00-01
LOAD data from ADDRESS where stores the second value to REGISTER_1	LOAD : 01	REGISTER_1 : 01 ADDRESS : XXXX	XXXX-01-01
ADD REG0 and REG1, store result in REGISTER_			N
STORE value ADDRESS			

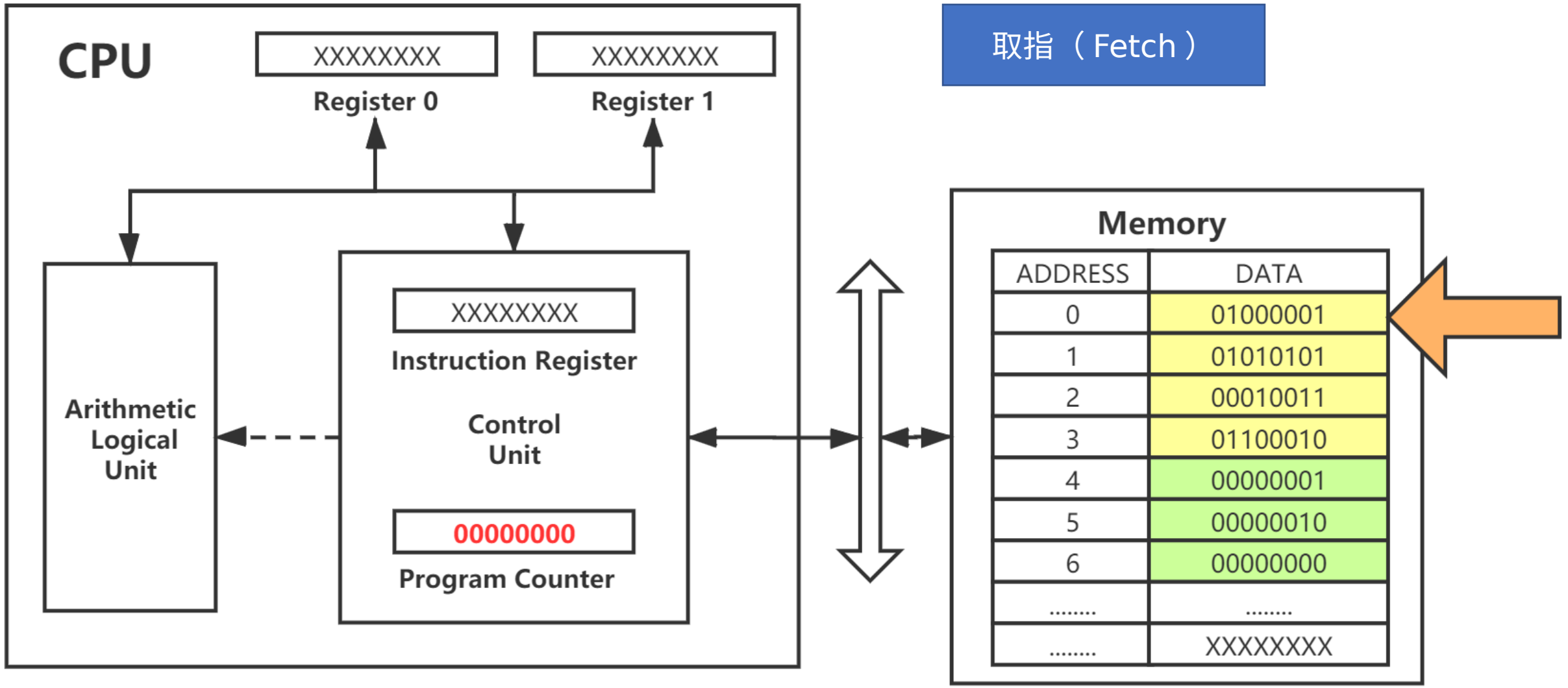


指令的编码格式

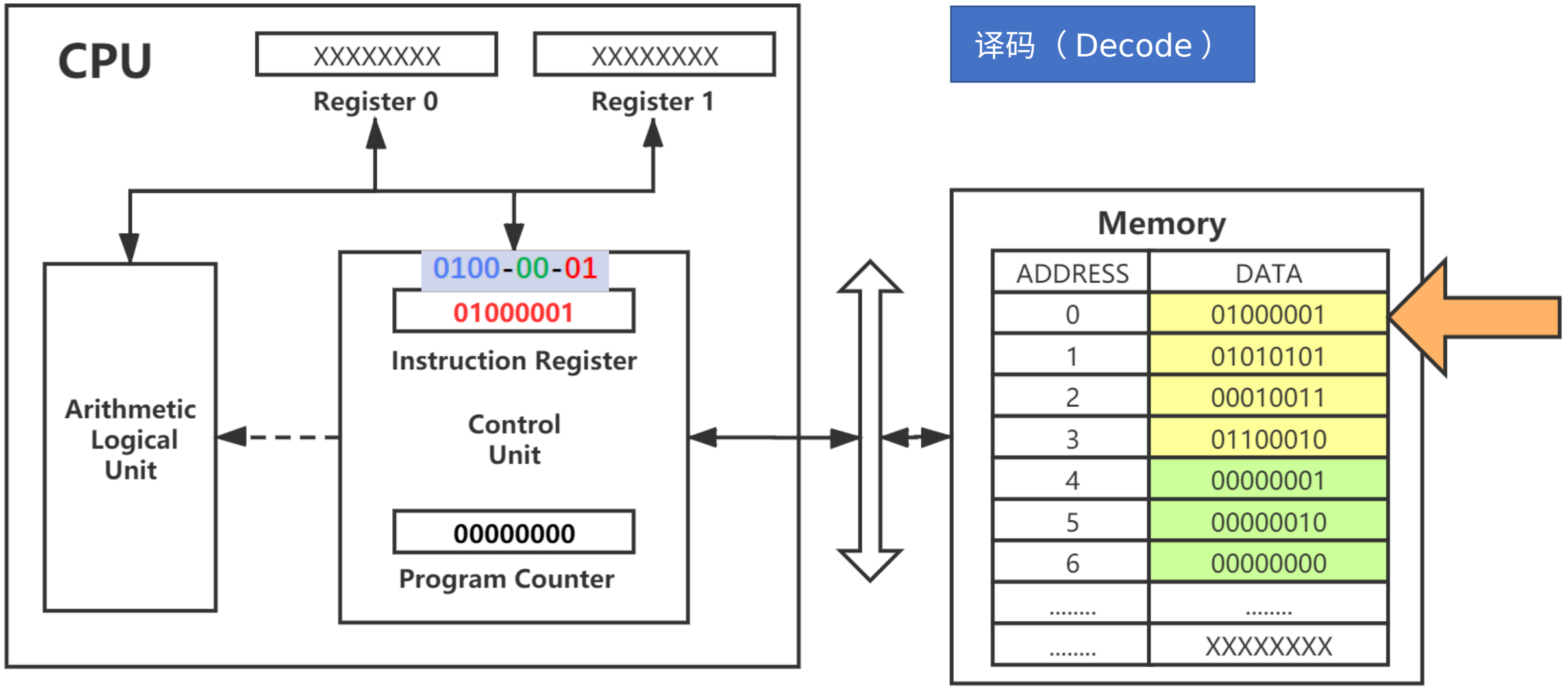
针对 mini 计算机的机器指令执行



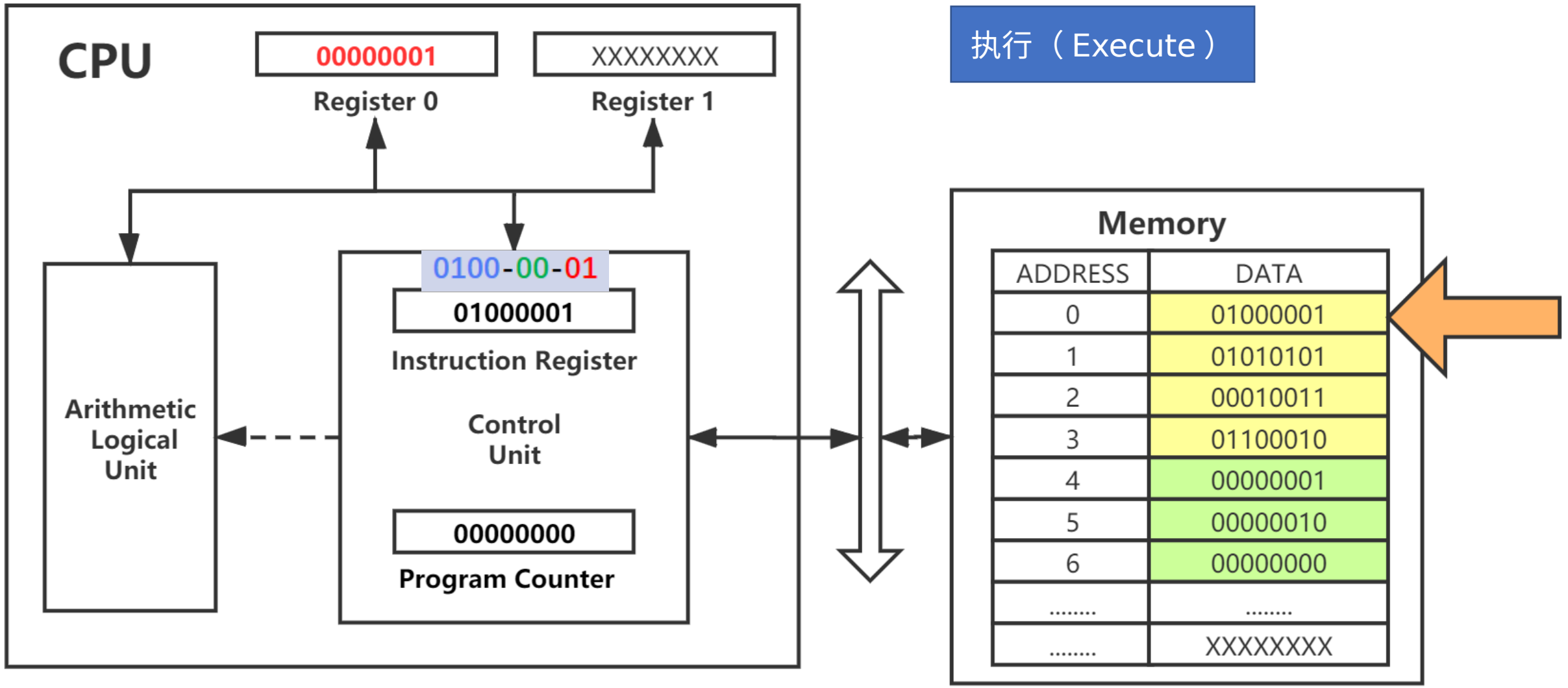
针对 mini 计算机的机器指令执行



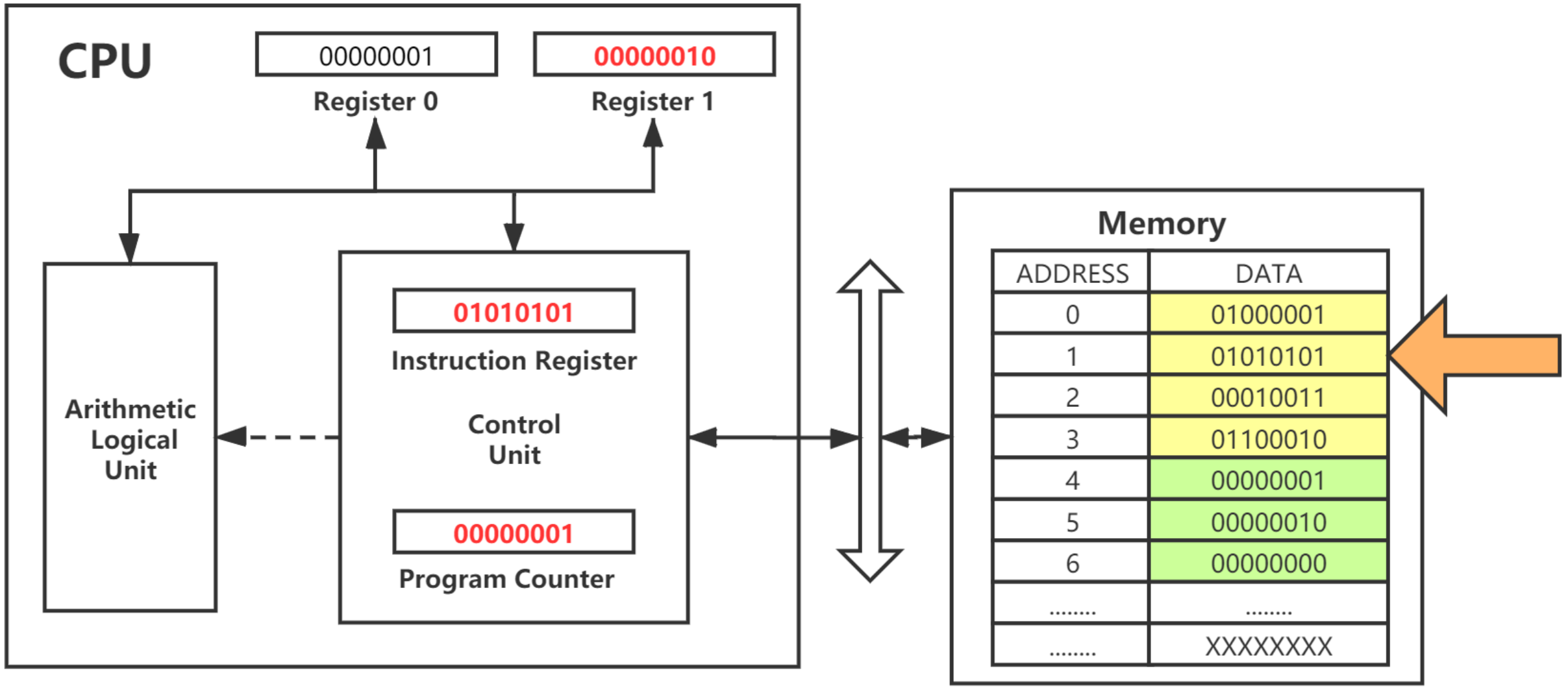
针对 mini 计算机的机器指令执行



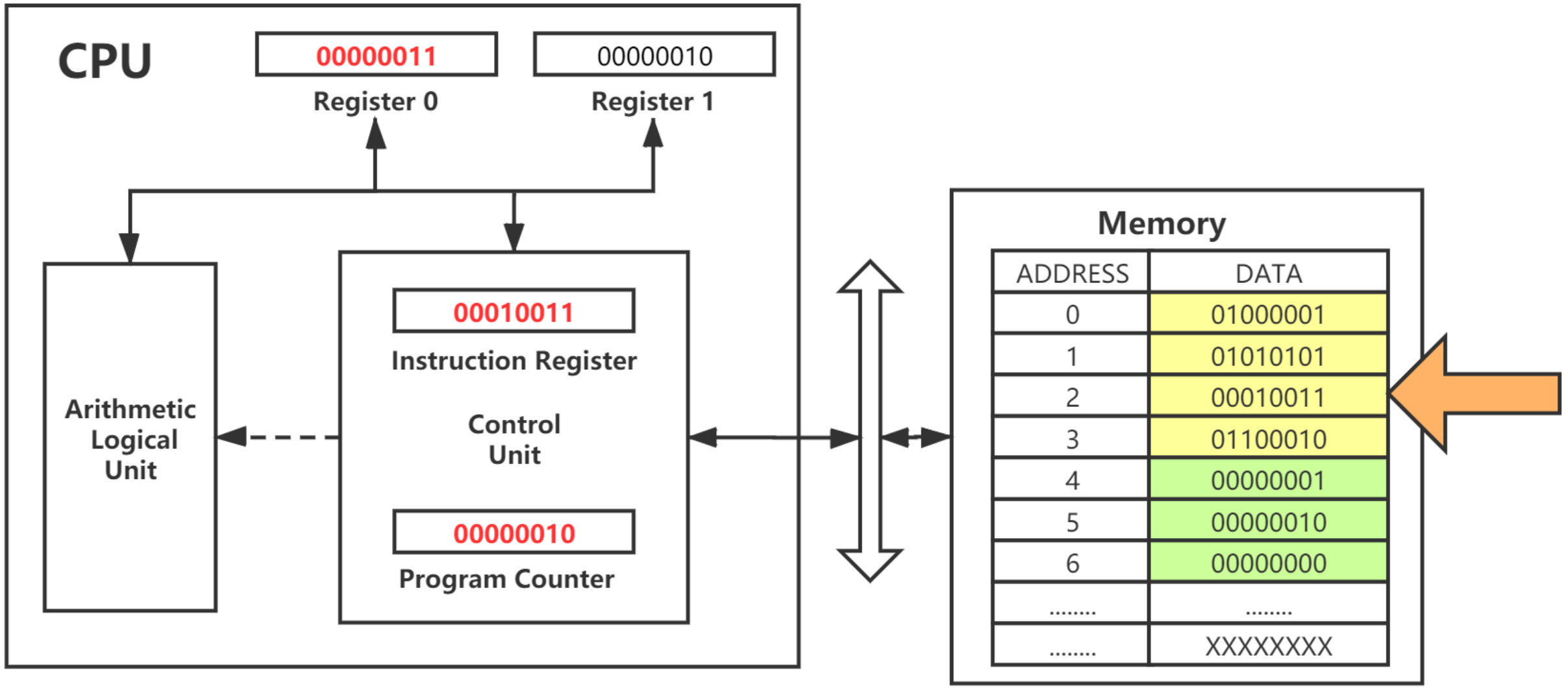
针对 mini 计算机的机器指令执行



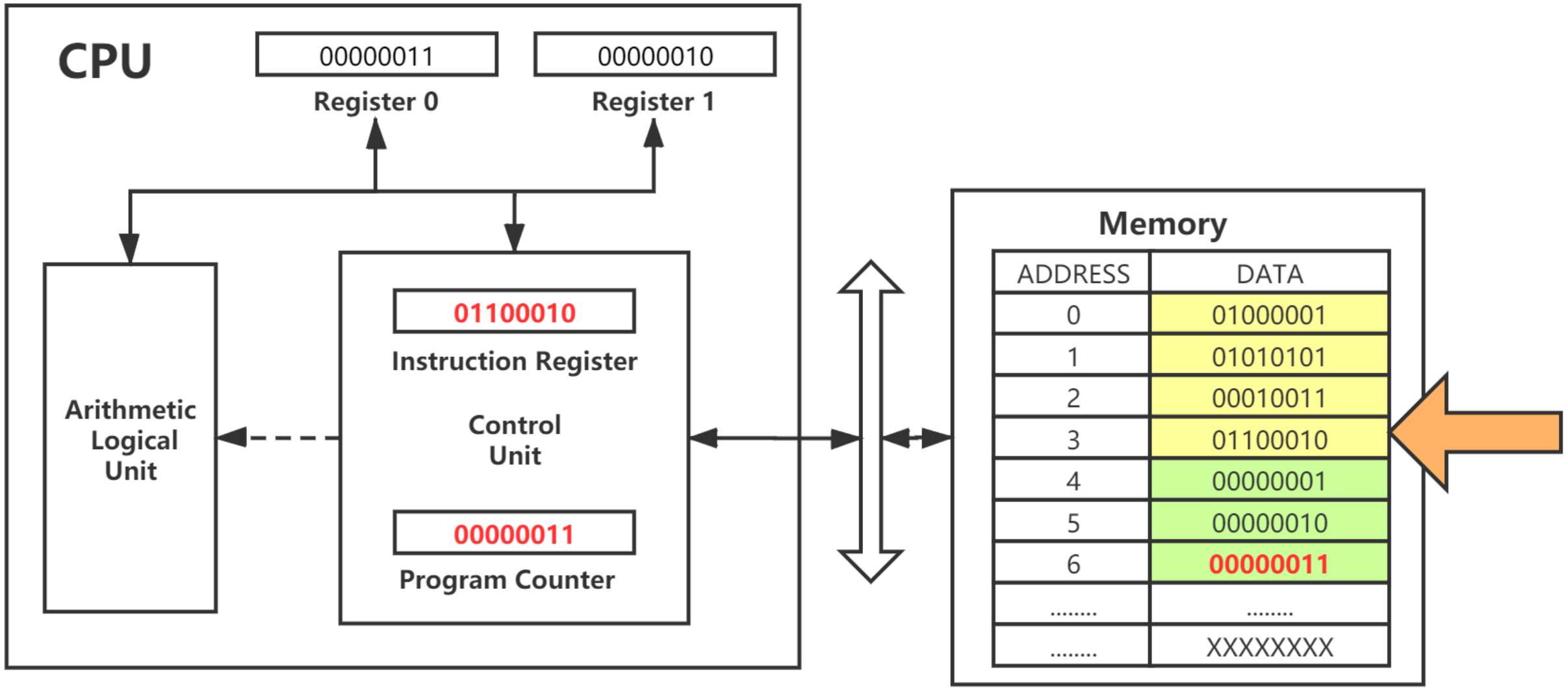
针对 mini 计算机的机器指令执行



针对 mini 计算机的机器指令执行



针对 mini 计算机的机器指令执行



编程语言的进化

```
// Advanced Language
byte array[3] = {1, 2, 0};
function add()
{
        byte a = array[0];
        byte b = array[1];
        array[2] = a + b;
}
```

高级语言

```
# Assemble Language
        .global add     # Define entry
add:
        load r0, 0x04   # Load r0 from address 0x04
        load r1, 0x05   # Load r0 from address 0x05
        add r0, r1      # Add r0 and r1
                        # and store result in r0
        store r0, 0x06  # Store r0 to address 0x06
array:
.byte   0x01
.byte   0x02
.byte   0x00
        .end            # End of file
```

机器语言

ADDRESS	DATA
0	01000001
1	01010101
2	00010011
3	01100010
4	00000001
5	00000010
6	00000000

机器语言

Hello World!

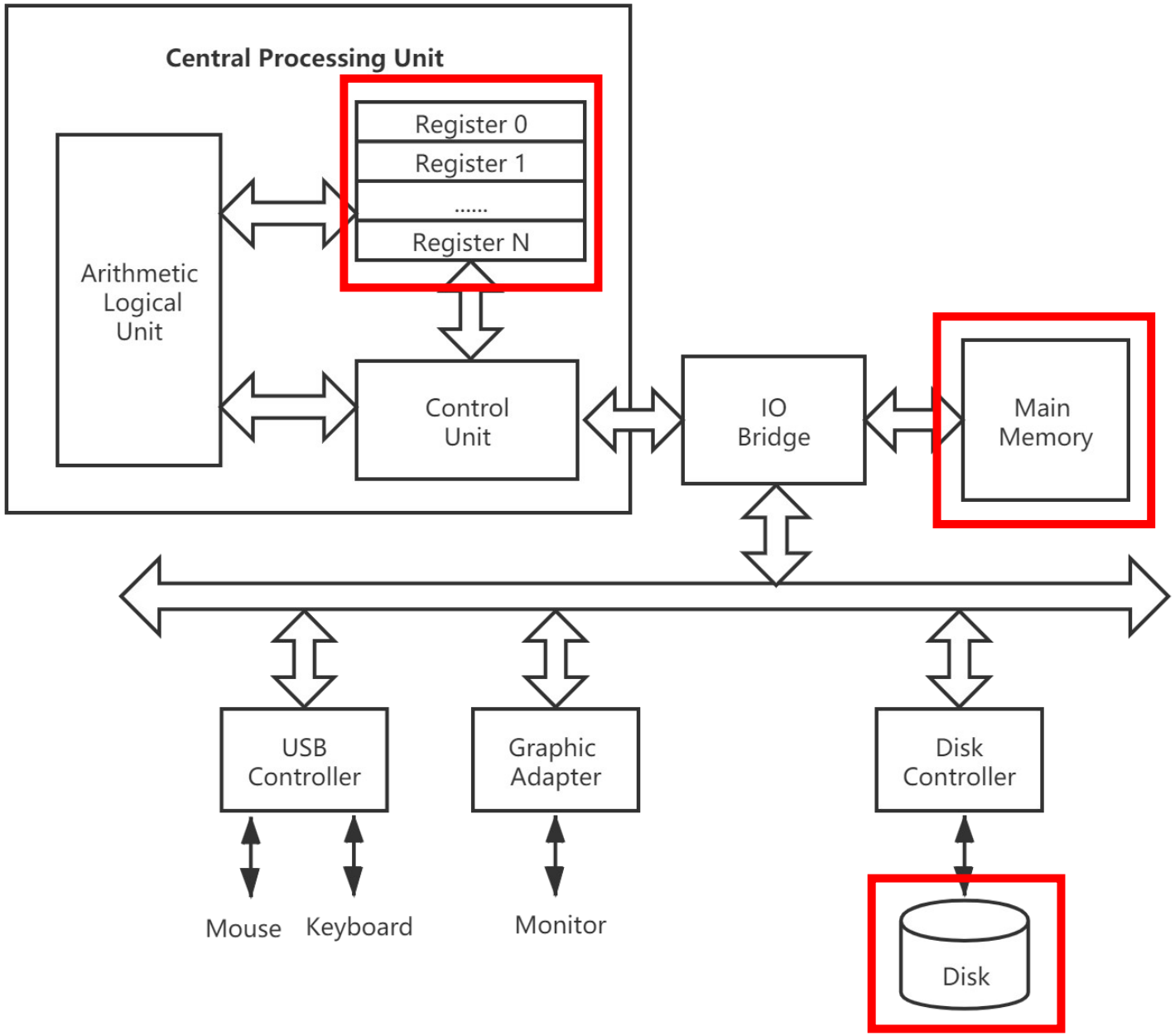
```
#include <stdio.h>

int main()
{
    printf( “hello world!\n” );
    return 0;
}
```

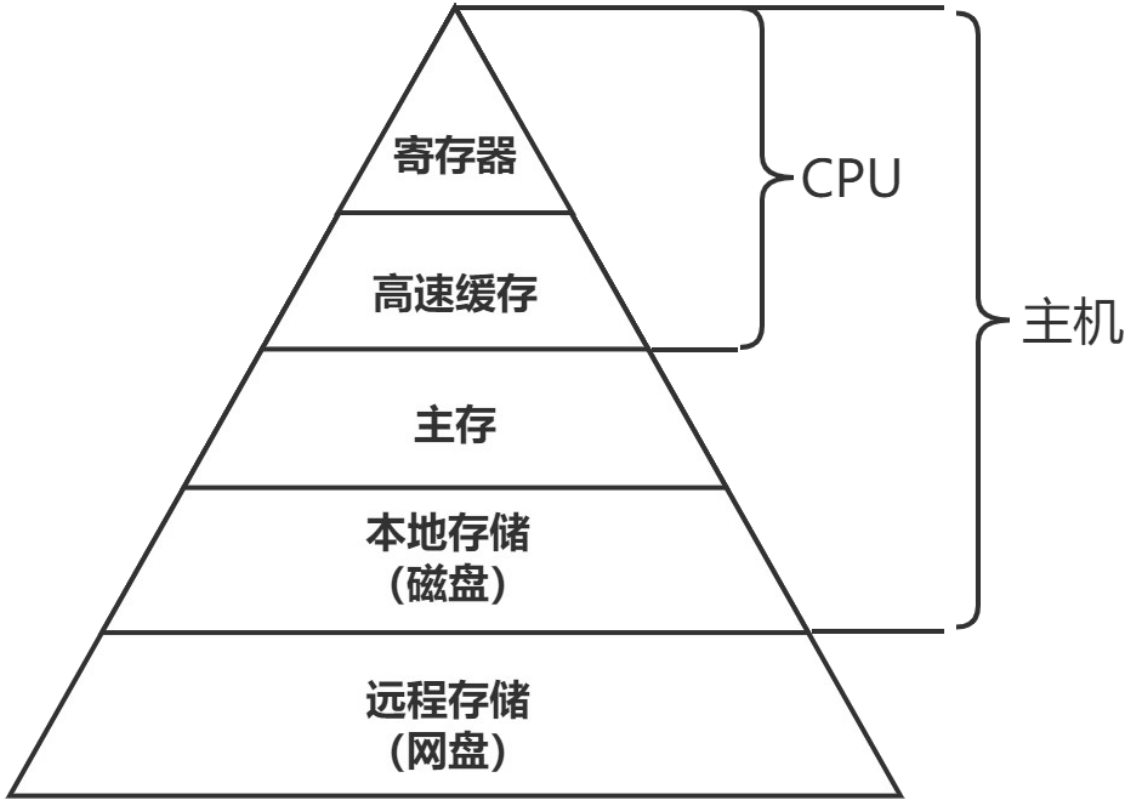
本章内容

- 计算机的硬件组成
- 程序的存储与执行
- 程序语言的设计和进化
- **存储设备的层次结构**
- 操作系统

存储设备的层次结构


Central Processing Unit
Register 0
Register 1
.....
Register N
Arithmetic Logical Unit
Control Unit
IO Bridge
Main Memory
USB Controller
Graphic Adapter
Disk Controller
Mouse
Keyboard
Monitor
Disk

存储设备的层次结构



速度　容量　成本

快　小　高

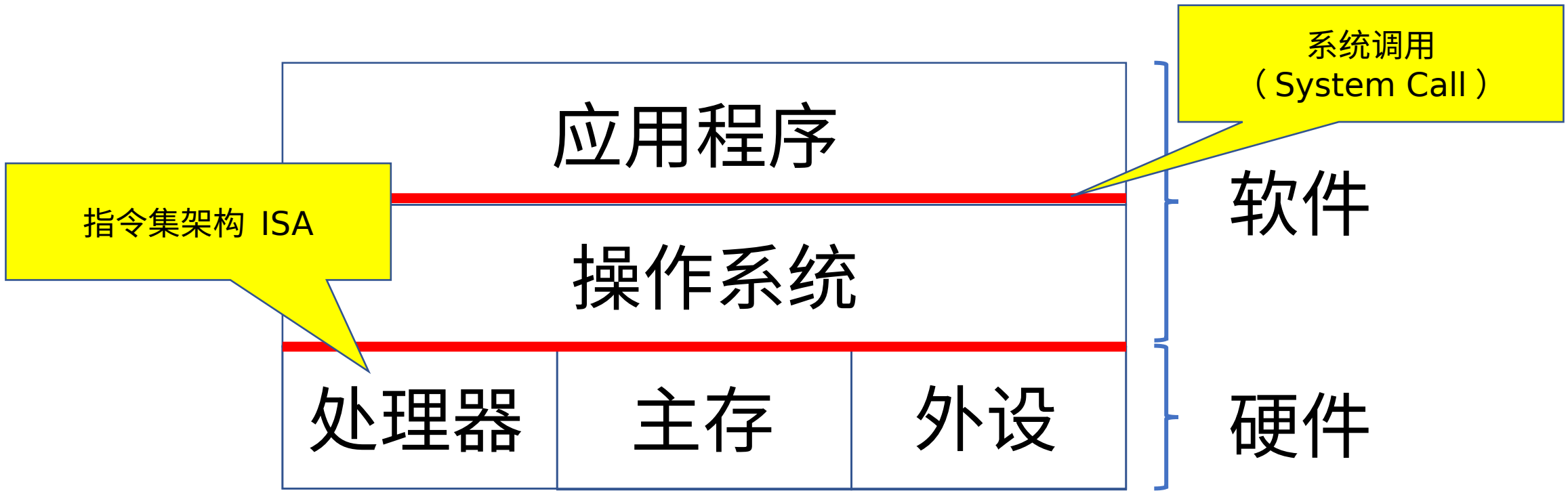
↓　↓　↓

慢　大　低

本章内容

- 计算机的硬件组成
- 程序的存储与执行
- 程序语言的设计和进化
- 存储设备的层次结构
- **操作系统**

操作系统



- **保护硬件被失控的软件应用程序滥用**
- **向应用程序提供简单一致的 *抽象接口* 来控制复杂的多种外设硬件。**

谢 谢

欢迎交流合作