

第三章 指令系统设计与执行

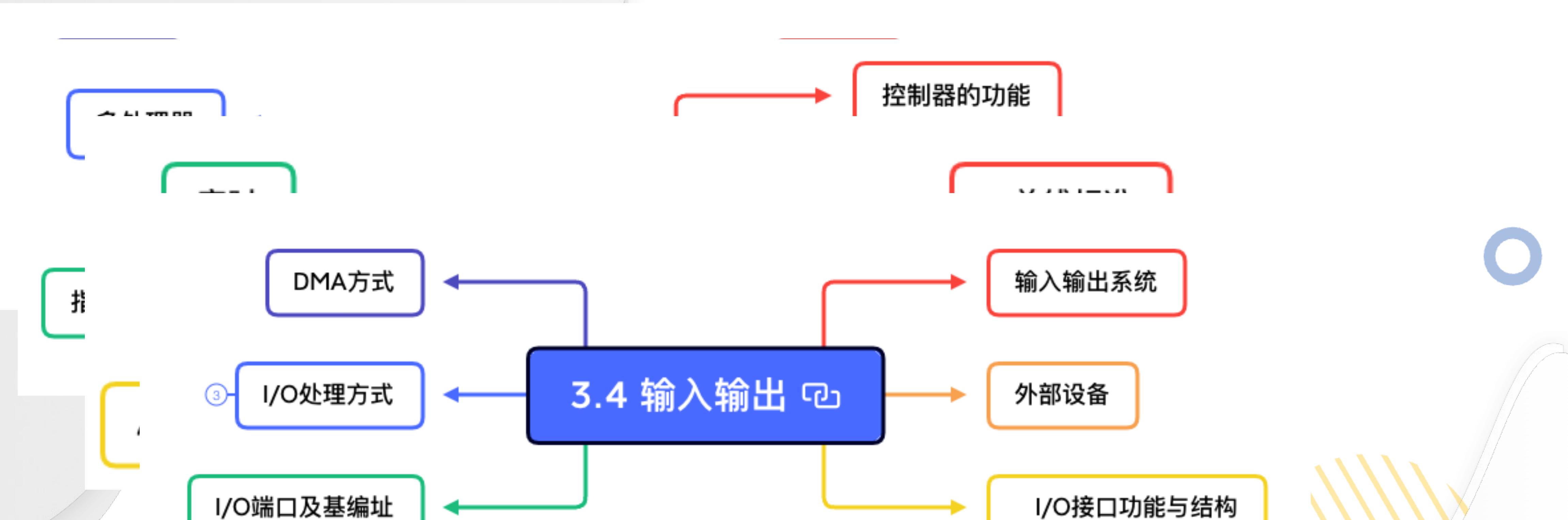
3. 指令系统设计与CPU运行控制

内容介绍

1. 指令系统设计与执行
2. CPU控制
3. 总线
4. 输入/输出



3. 指令系统设计与CPU运行控制 内容介绍





3. 指令系统设计与CPU运行控制

3.1 指令系统设计与执行

船说：计算机基础

3. 指令系统设计与CPU运行控制

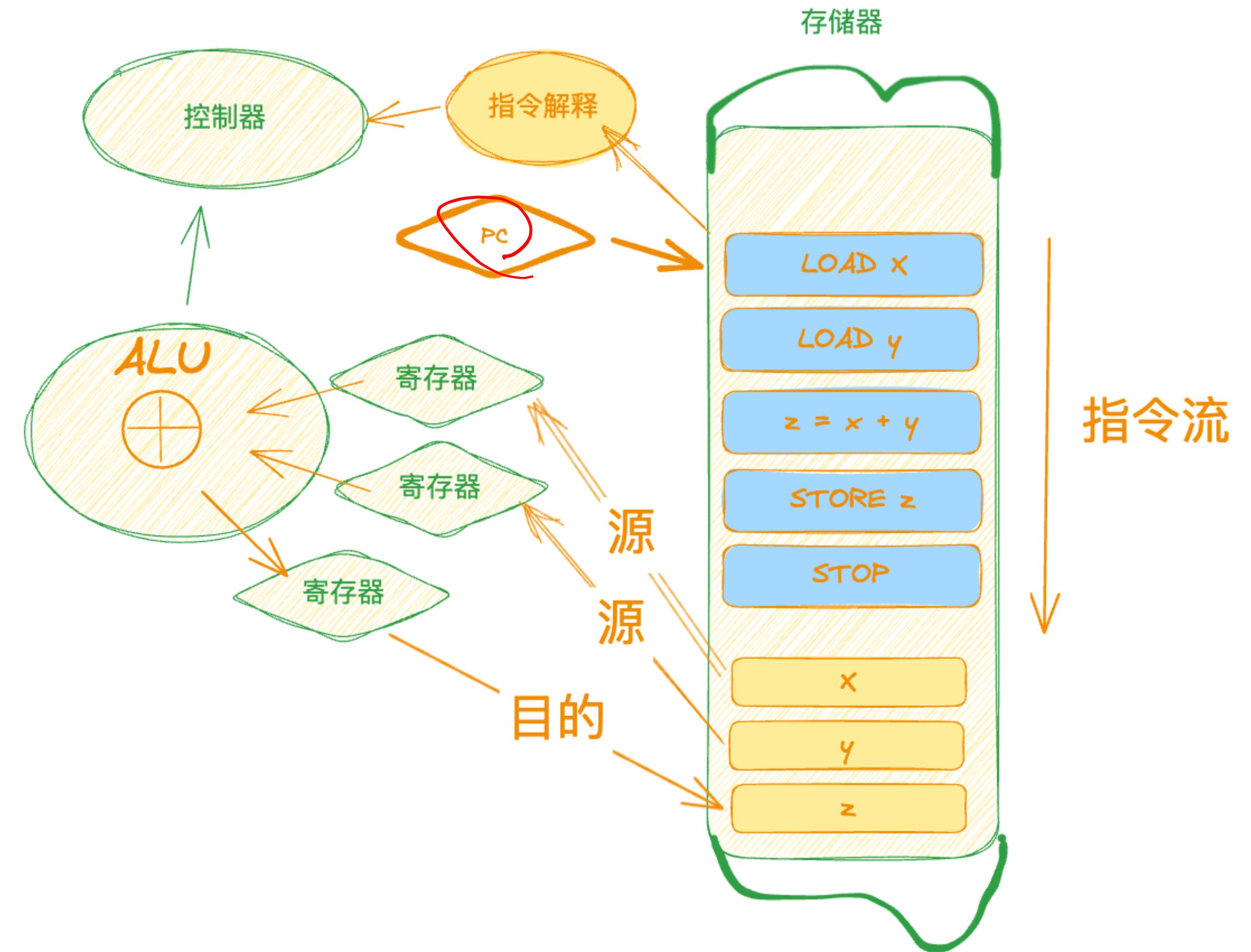
1 指令系统设计与执行

1. 指令格式
2. 指令操作类型
3. 指令寻址方式
4. CISC和RISC
5. 指令的执行
6. 数据通路



3. 指令系统设计与CPU运行控制

3.1.1 存储型计算机





3. 指令系统设计与CPU运行控制

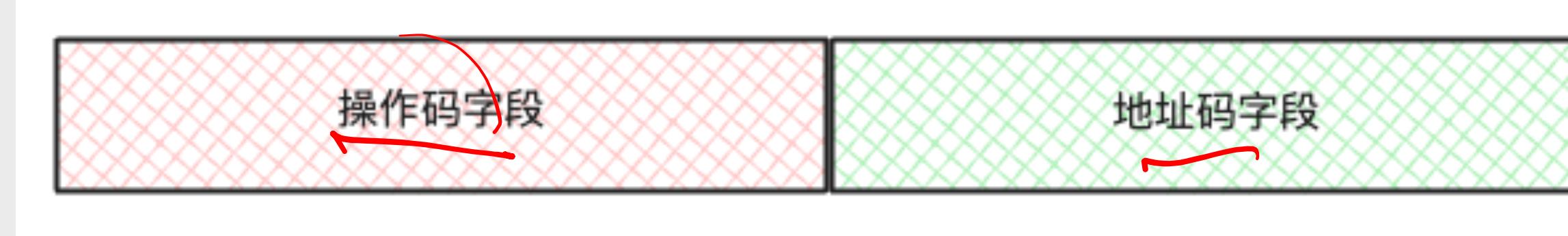
3.1.1 指令集体体系结构ISA

- 1、从汇编语言程序的角度描述计算机，并功能具备的功能
- 2、ISA说明了计算机能做什么，而计算机的组成说明了这些功能是如何完成的
- 3、ISA就是计算机能做事的一堆命令集合
- 4、指令：指示计算机执行某种操作的命令，是计算机运行的最小功能单位



3. 指令系统设计与CPU运行控制

3.1.1 完整指令格式设计



- 1、指令长度和机器字长没有固定关系
- 2、定长指令字结构：所有指令的长度都一样（单字长指令、半字长指令、双字长指令）
- 3、变长指令字结构：根据功能长度变化

010101

字长.8位

16—

32—
64—



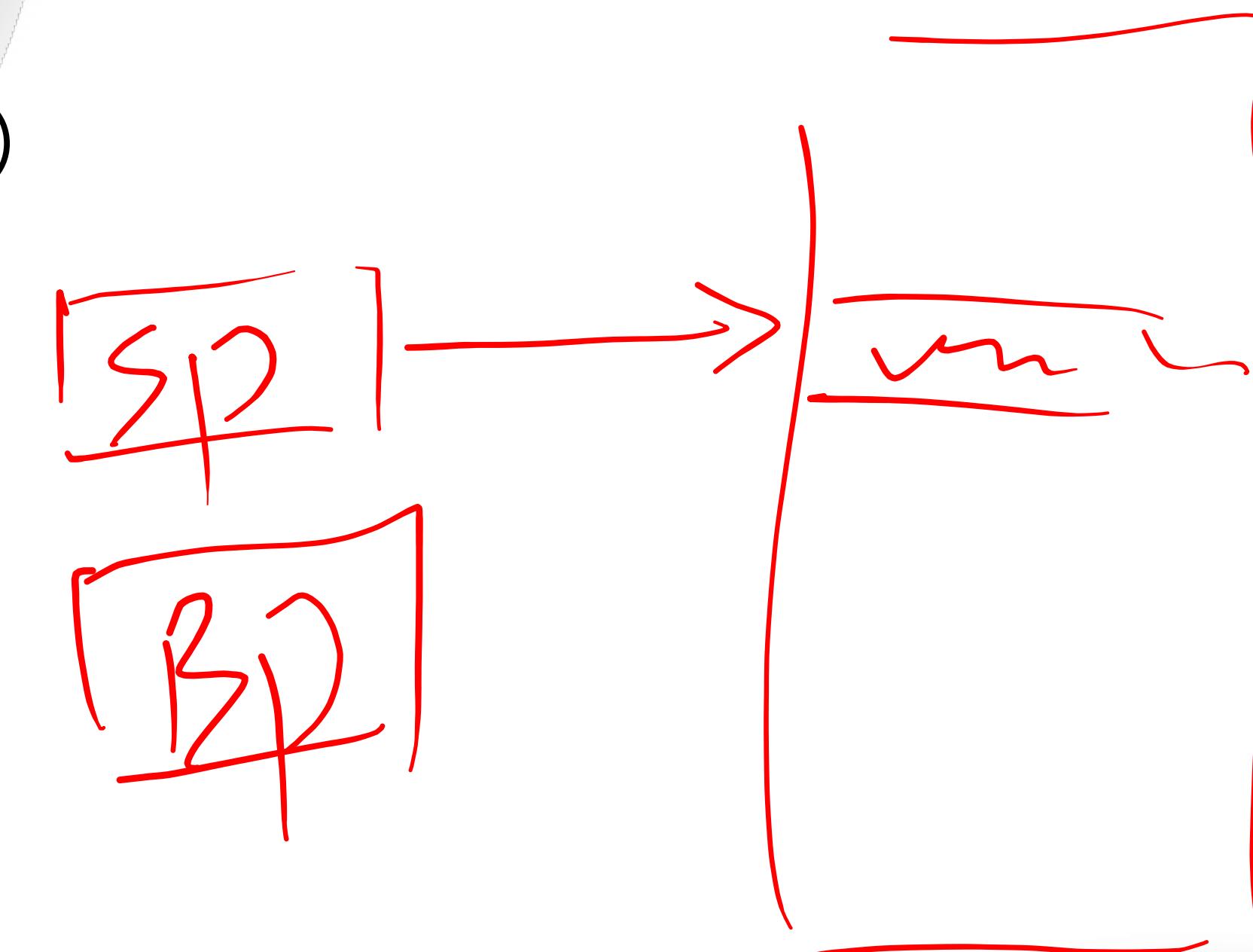
3. 指令系统设计与CPU运行控制

3.1.1 零地址指令

OP

- 1、只给出操作码（空、停机、关中断）
- 2、默认用栈空间的数据

push





3. 指令系统设计与CPU运行控制

3.1.1 一地址指令



- 1、在A1地址读取数据，OP操作完成后，再把结果存回到A1
- 2、隐含约定的地址的双操作数指令，一个数据来自A1，另一个数据来自ACC（累加器）
- 3、如果指令长度为32位，操作码占8位，地址占24位，则指令操作数据的直播寻址范围为 $2^{24} = 16M$



3. 指令系统设计与CPU运行控制

3.1.1 二地址指令

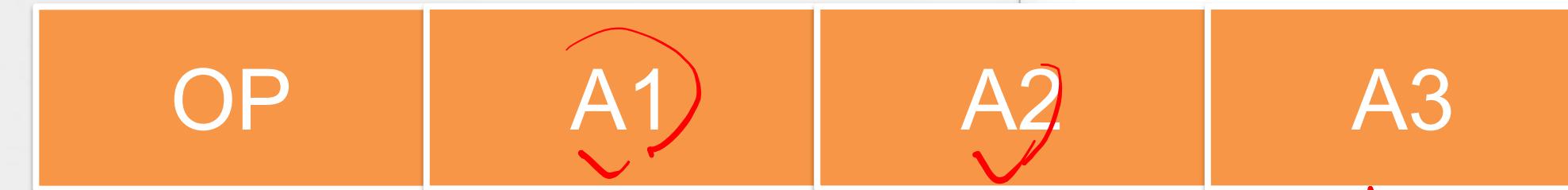


- 1、读取A1和A2地址数据，OP操作完成后，再把结果存回到A1
- 2、如果指令长度为32位，操作码占8位，两个地址各占12位，则指令操作数据的直播寻址范围为 $2^{12} = 4K$



3. 指令系统设计与CPU运行控制

3.1.1 三地址指令

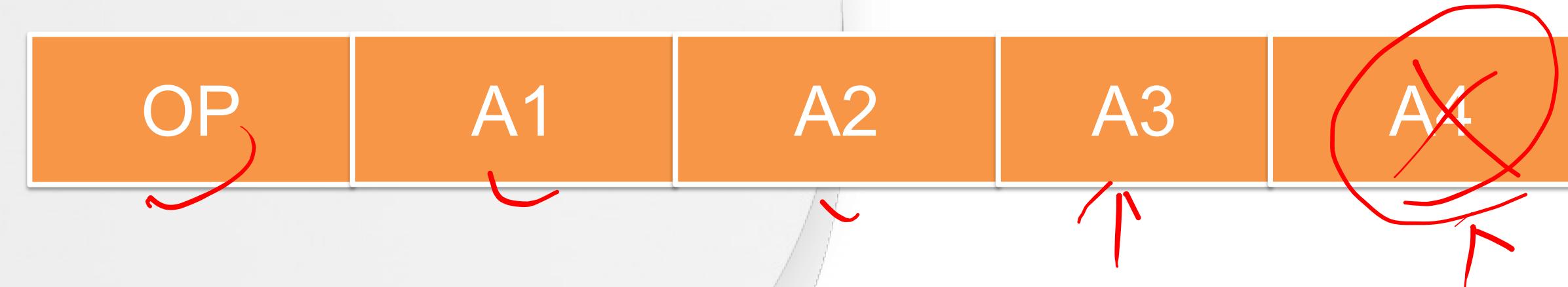


- 1、读取A1和A2地址数据，OP操作完成后，再把结果存回到A3
- 2、如果指令长度为32位，操作码占8位，三个地址各占8位，则
指令操作数据的直播寻址范围为 $2^8 = 256$ 字节
- 3、如果指令和所有数据都在主存中，三地址指令执行需要访问
主存4次。



3. 指令系统设计与CPU运行控制

3.1.1 四地址指令



1、读取A1和A2地址数据，OP操作完成后，再把结果存回到A3

2、A4存放的下一条要执行的指令地址。

如果指令长度为32位，操作码占8位，三个地址各占6位，则指令操作数据的直播寻址范围为 $2^6 = 64$

3、执行完本指令后，马上会执行A4指定的地址内的指令。

PC



3. 指令系统设计与CPU运行控制

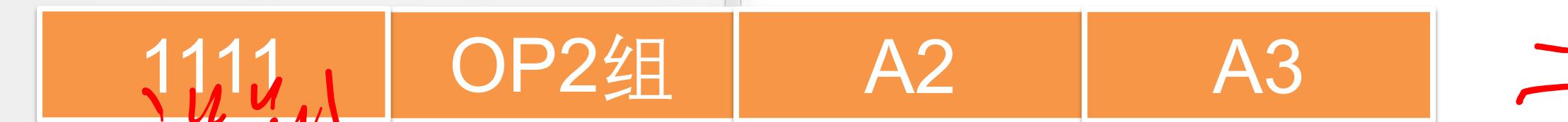
3.1.1 操作码设计：根据指令数分配段位

16位

4位操作码：



8位操作码：



12位操作码：



16位操作码：



总结：

- 1、使用频率高的指令，分配较短的操作码
- 2、操作码不能重复
- 3、扩展操作码增加了译码和分析难度，控制器设计很复杂

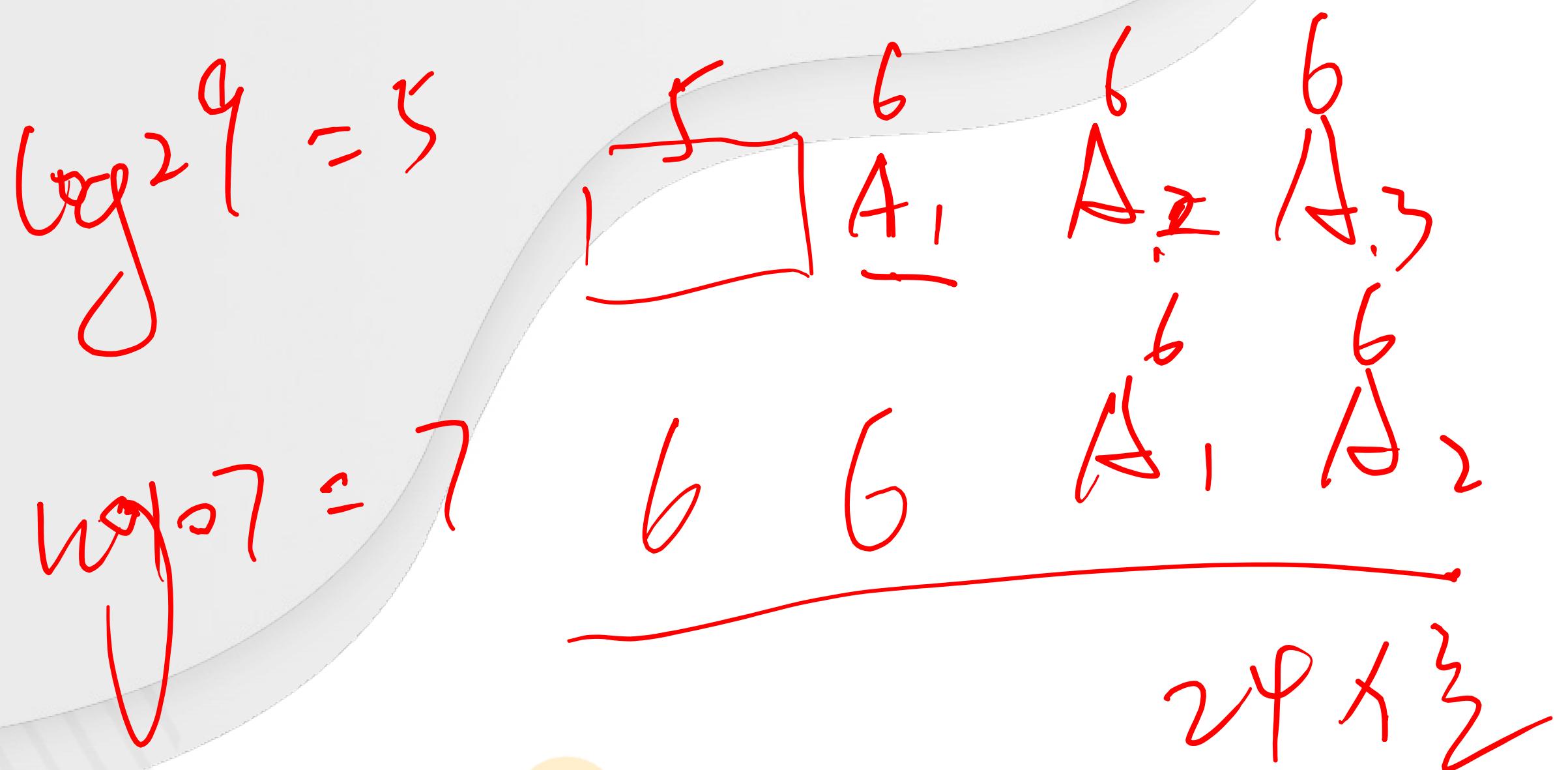


3. 指令系统设计与CPU运行控制

3.1.1 练习

例1【2017统考】：某计算机按字节编址，指令长度固定且只有
两种指令格式，其中三地址指令29条，二地址指令107条，每个
地址字段为6位，则指令字长至少应该是 A。

- A. 24位
- B. 26位
- C. 28位
- D. 32位





3. 指令系统设计与CPU运行控制

3.1.1 ARM指令集

32bit RISC 1足矣

3 3 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1 1 9 8 7 6 5 4 3 2 1 0	Cond	0 0 1	Opcode	S	Rn	Rd	Operand 2			
1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0	Cond	0 0 0 0 0 0 A S			Rd	Rn	Rs	1 0 0 1		Rm
	Cond	0 0 0 0 1 U A S			RdHi	RdLo	Rn	1 0 0 1		Rm
	Cond	0 0 0 1 0 B 0 0			Rn	Rd	0 0 0 0 1 0 0 0 1			Rm
	Cond	0 0 0 1 0 0 1 0 1 1 1 1 1 1 1 1 1 1 1 0 0 0 1								Rn
	Cond	0 0 0 P U 0 W L			Rn	Rd	0 0 0 0 1 S H 1			Rm
	Cond	0 0 0 P U 1 W L			Rn	Rd	Offset	1 S H 1		Offset
	Cond	0 1 I P U B W L			Rn	Rd	Offset			
	Cond	0 1 1					1			
	Cond	1 0 0 P U S W L			Rn	Register List				
	Cond	1 0 1 L				Offset				
	Cond	1 1 0 P U N W L			Rn	CRd	CP#	Offset		
	Cond	1 1 1 0 CP Opc			CRn	CRd	CP#	CP	0	CRm
	Cond	1 1 1 0 CP Opc L			CRn	Rd	CP#	CP	1	CRm
	Cond	1 1 1 1			Ignored by processor					
3 3 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1 1 9 8 7 6 5 4 3 2 1 0										
1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0										

Figure 4-1: ARM instruction set formats



3. 指令系统设计与CPU运行控制

3.1.2 ARM指令操作类型

主存 \leftrightarrow 寄存器

1. 数据传送:

1. LDR/STR: 用于加载和存储操作, 可以将数据从内存传送到寄存器, 或者从寄存器传递到内存。
2. MOV: 将一个寄存器的值移动到另一个寄存器。
立即数
3. LDM/STM: 加载和存储多个寄存器, 用于处理多数据传送。

2. 算术和逻辑运算:

1. ADD/SUB: 用于加法和减法运算。
2. MUL: 用于乘法运算。
3. AND/ORR/EOR: 逻辑与、或、异或运算。
4. CMP: 比较两个寄存器的值。

3. 移位操作:

1. LSL/LSR: 逻辑左移和逻辑右移。
2. ASR: 算术右移。
3. ROR: 循环右移。

4. 转移操作:

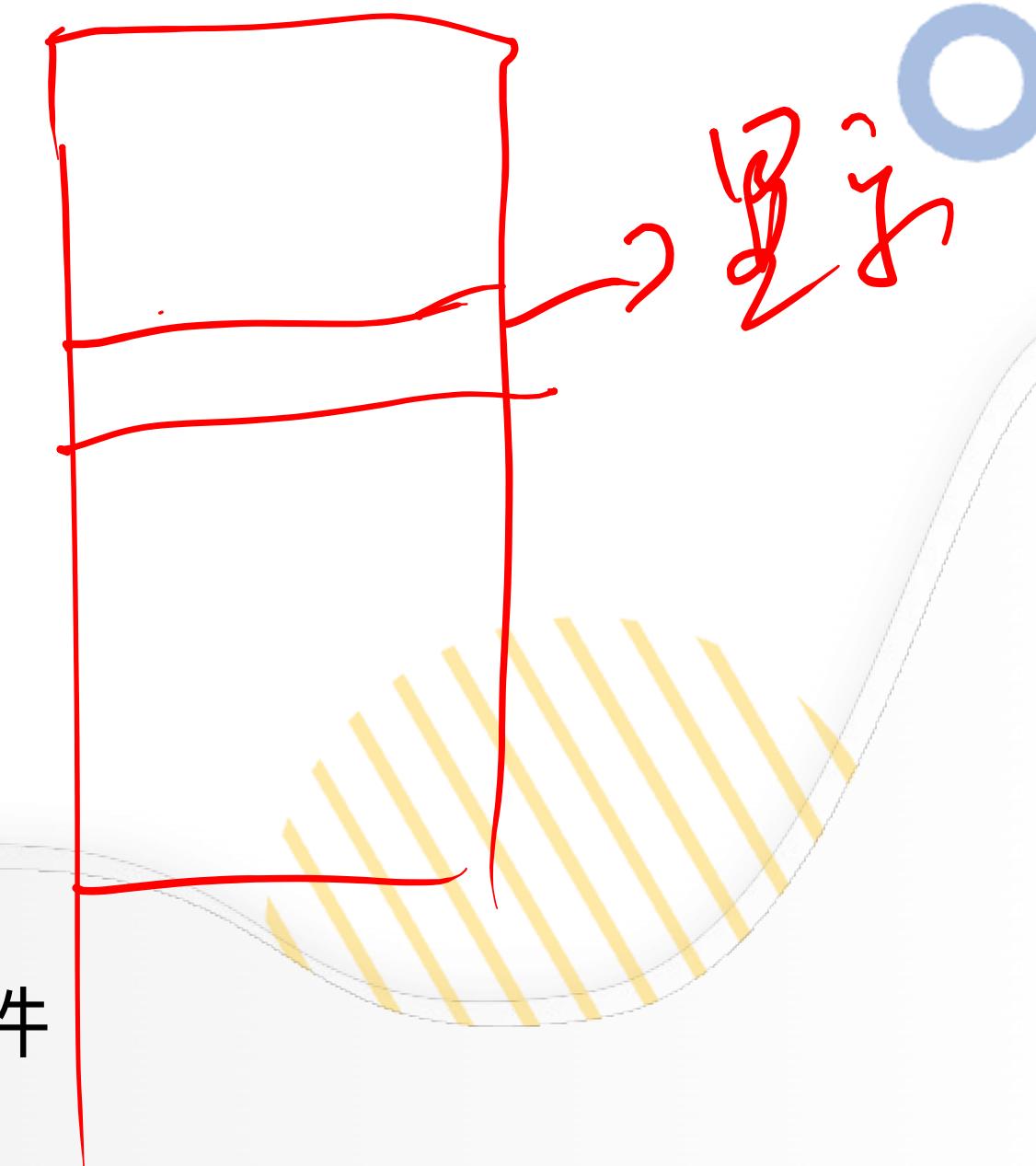
1. B/BL: 无条件跳转, BL还会保存返回地址。
2. BX: 通过寄存器值跳转。
3. 条件转移指令, 如 BEQ (等于时跳转) 、 BNE (不等于时跳转) 等。

5. 输入输出操作:

1. ARMv7指令集本身不直接提供输入输出指令, 这些操作通常通过操作系统或特定的硬件接口来实现。但是, 可以使用LDR和STR指令与内存映射的I/O接口进行数据交换。

PC \rightarrow 指令!

2
3





3. 指令系统设计与CPU运行控制

3.1.2 ARM标记位

1. 零标志 (Z flag) :

- 如果比较结果为零 (即两个比较数值相等)，零标志被设置 (Z=1)。
- 如果结果非零 (不相等)，零标志被清除 (Z=0)。

2. 负标志 (N flag) :

- 如果比较结果为负，负标志被设置 (N=1)。
- 如果结果为正数或零，负标志被清除 (N=0)。

3. 进位标志 (C flag) :

- 在无符号运算中，如果发生借位，进位标志被设置 (C=1)。
- 如果没有发生借位，进位标志被清除 (C=0)。
- 在ARM中，进位标志的解释可能会根据具体的指令略有不同。

4. 溢出标志 (V flag) :

- 如果在有符号运算中发生溢出，溢出标志被设置 (V=1)。
- 如果没有发生溢出，溢出标志被清除 (V=0)。

正 + 正 = 正

负 + 负 = 正

CPSR 32bit

subs r0, r0, r0
movs r0, #0



3. 指令系统设计与CPU运行控制

3.1.2 指令语法前/后缀

条件码	条件码助记符	含义	CPSR中条件标志位值
0000	eq	相等	Z=1
0001	ne	不相等	Z=0
0010	cs/hs	无符号数大于/等于	C=1
0011	cc/lo	无符号数小于	C=0
0100	mi	负数	N=1
0101	pl	非负数	N=0
0110	vs	上溢出	V=1
0111	vc	没有上溢出	V=0
1000	hi	无符号数大于	C=1且Z=0
1001	ls	无符号数小于/等于	C=0且Z=1
1010	ge	带符号数大于/等于	N=1且V=1 或 N=0且V=0
1011	lt	带符号数小于	N=1且V=0 或 N=0且V=1
1100	gt	带符号数大于	Z=0且N=V
1101	le	带符号数小于/等于	Z=1或N!=V
1110	al	无条件执行	
1111	nv	该指令从不执行	

Sub ne

Sub ne



3. 指令系统设计与CPU运行控制

3.1.2 X86指令操作类型

PC



1. 数据传送:

1. MOV: 将数据从一个地方移动到另一个地方, 例如从寄存器到寄存器, 从内存到寄存器, 或反之。
2. PUSH/POP: 用于堆栈操作, 将数据推送到堆栈或从堆栈弹出。
3. LEA (Load Effective Address) : 加载有效地址到寄存器。

2. 算术和逻辑运算:

1. ADD/SUB: 加法和减法运算。
2. MUL/IMUL: 无符号和有符号乘法。
3. DIV/IDIV: 无符号和有符号除法。
4. AND/OR/XOR/NOT: 逻辑运算。

3. 移位操作:

1. SHL/SAL: 逻辑左移。
2. SHR: 逻辑右移。
3. SAR: 算术右移。
4. ROL/ROR: 循环左移和循环右移。

4. 转移操作:

1. JMP: 无条件跳转。
2. 条件跳转指令, 如 JE (跳转如果等于) 、 JNE (跳转如果不等于) 等。
3. CALL: 调用子程序。
4. RET: 从子程序返回。

5. 输入输出操作:

1. IN: 从端口读取数据到寄存器。
2. OUT: 将数据从寄存器写入端口。
3. 这些指令用于与外部设备进行数据交换, 通常在更高级别的系统编程中使用。

单片机



3. 指令系统设计与CPU运行控制

3.1.2 X86标记位

1. 零标志 (ZF, Zero Flag) :

1. 如果比较的两个值相等，即结果为零，零标志被设置 ($ZF=1$)。
2. 如果不相等，零标志被清除 ($ZF=0$)。

2. 符号标志 (SF, Sign Flag) :

1. 如果比较结果为负数，符号标志被设置 ($SF=1$)。
2. 如果结果为正数或零，符号标志被清除 ($SF=0$)。

3. 进位标志 (CF, Carry Flag) :

1. 在无符号运算中，如果发生借位，进位标志被设置 ($CF=1$)。
2. 如果没有发生借位，进位标志被清除 ($CF=0$)。

4. 溢出标志 (OF, Overflow Flag) :

1. 如果在有符号运算中发生溢出，溢出标志被设置 ($OF=1$)。
2. 如果没有发生溢出，溢出标志被清除 ($OF=0$)。

5. 辅助进位标志 (AF, Auxiliary Carry Flag) :

1. 主要用于BCD（二进制编码的十进制）运算。如果在低四位中发生借位，辅助进位标志被设置 ($AF=1$)。
2. 否则被清除 ($AF=0$)。

7 4 3 2
↓ ↓
1 0



3. 指令系统设计与CPU运行控制

3.1.3 编址与寻址

两种编址方式：

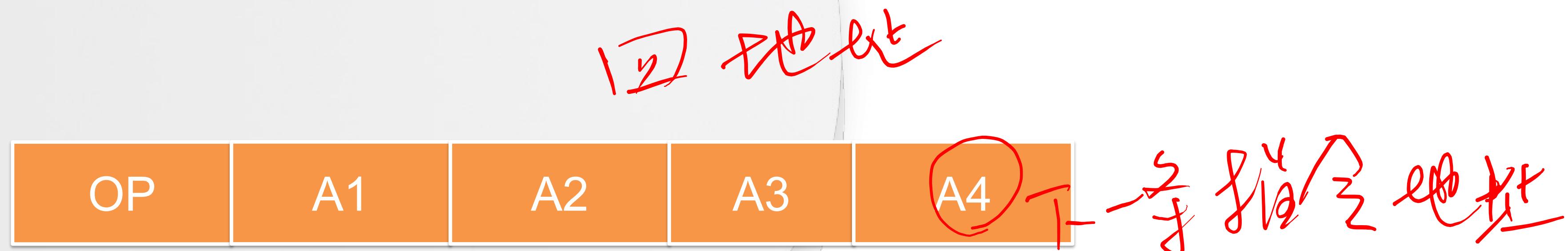
- 1、统一编址：I/O与内存统一编址，如ARM，访问外设如同访问内存
- 2、独立编址：I/O与内存分开编址，如X86，专门的CPU指令

两种寻址：

- 1、指令寻址：下一条将要执行的指令在哪里？
- 2、数据寻址：本条指令执行要用的数据在哪里？

3. 指令系统设计与CPU运行控制

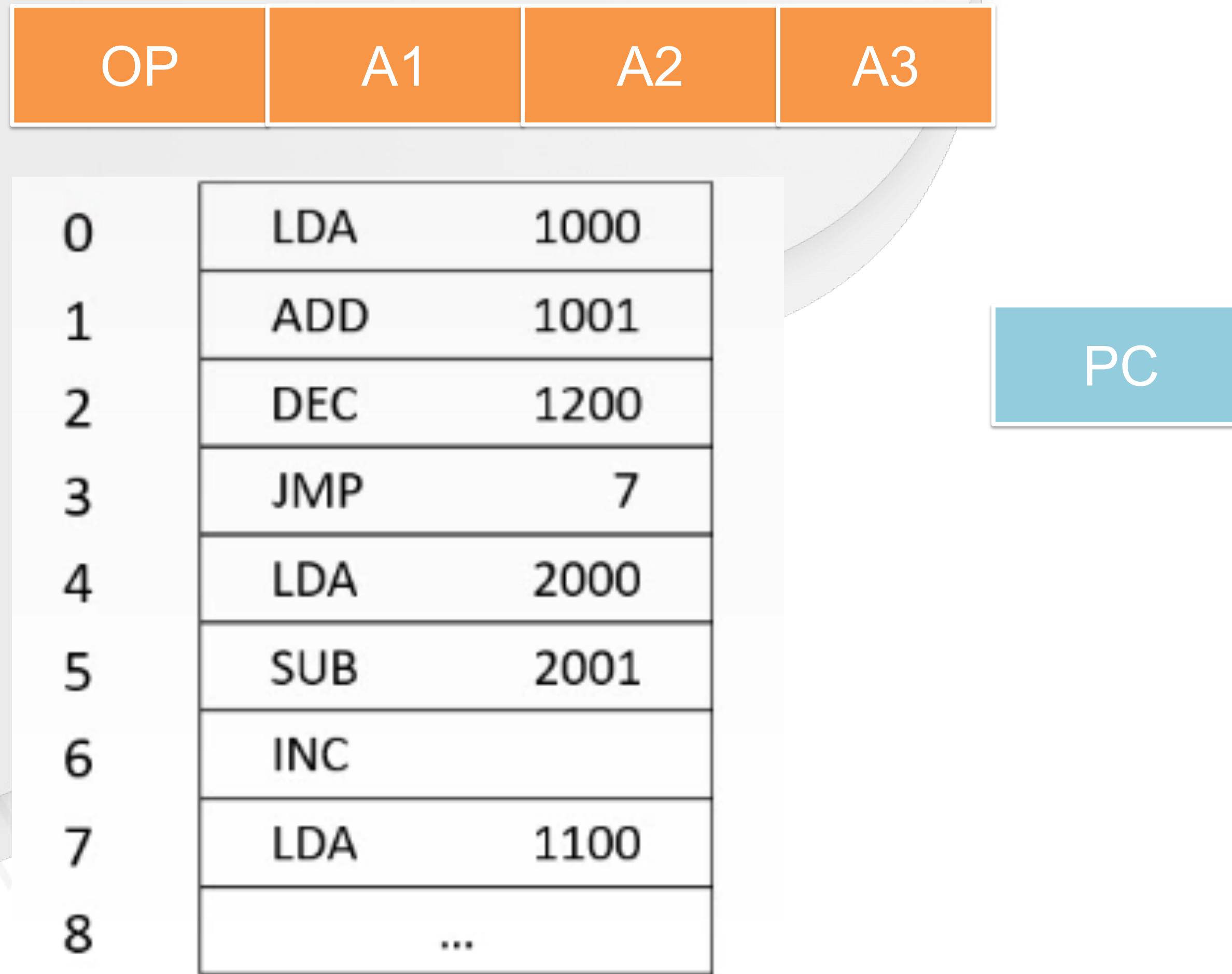
3.1.3 指令寻址方式：顺序寻址





3. 指令系统设计与CPU运行控制

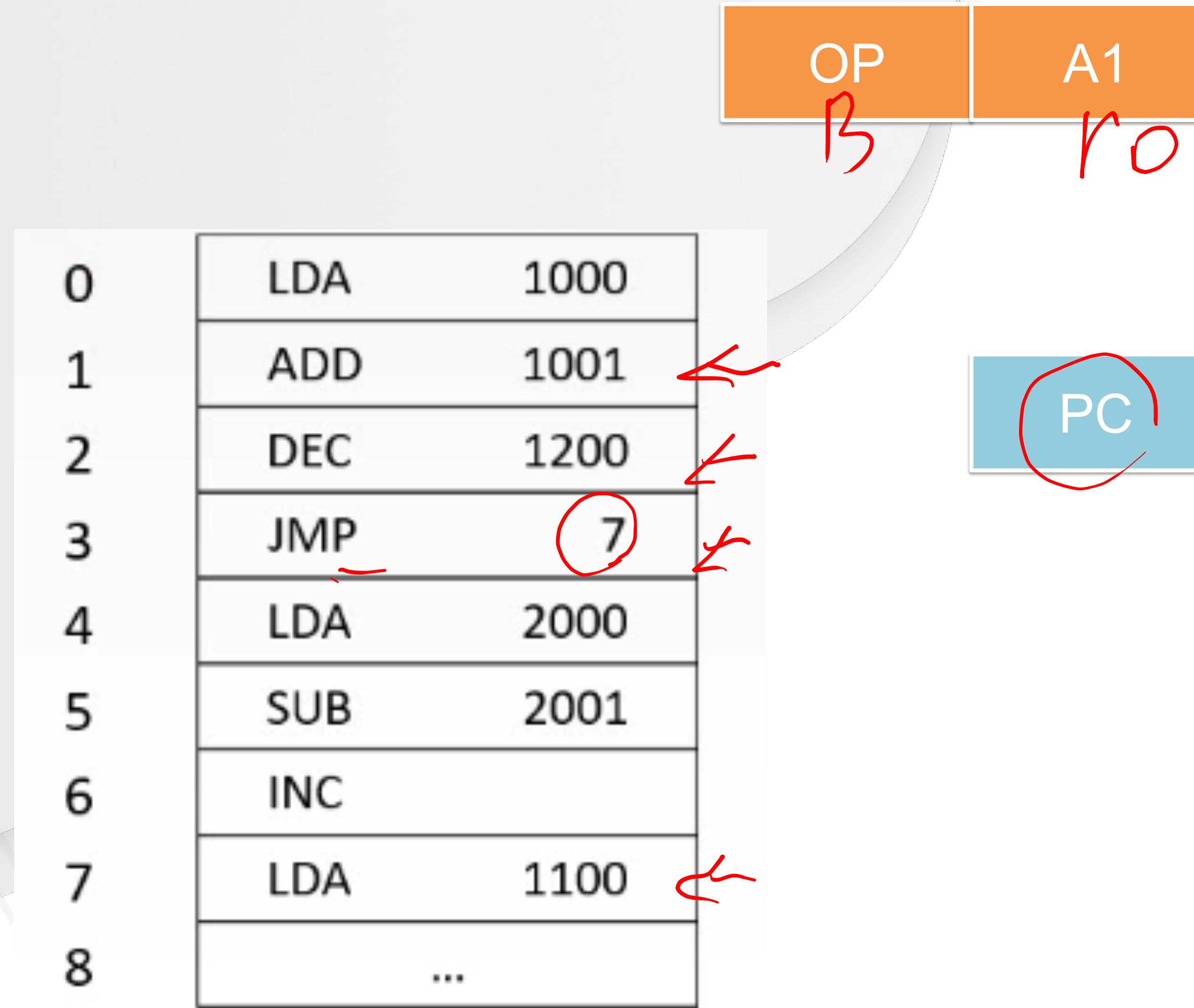
3.1.3 指令寻址方式：顺序寻址





3. 指令系统设计与CPU运行控制

3.1.3 指令寻址方式：跳转执行



B
BL.
JMP
Call



3. 指令系统设计与CPU运行控制

3.1.3 数据寻址方式

OP

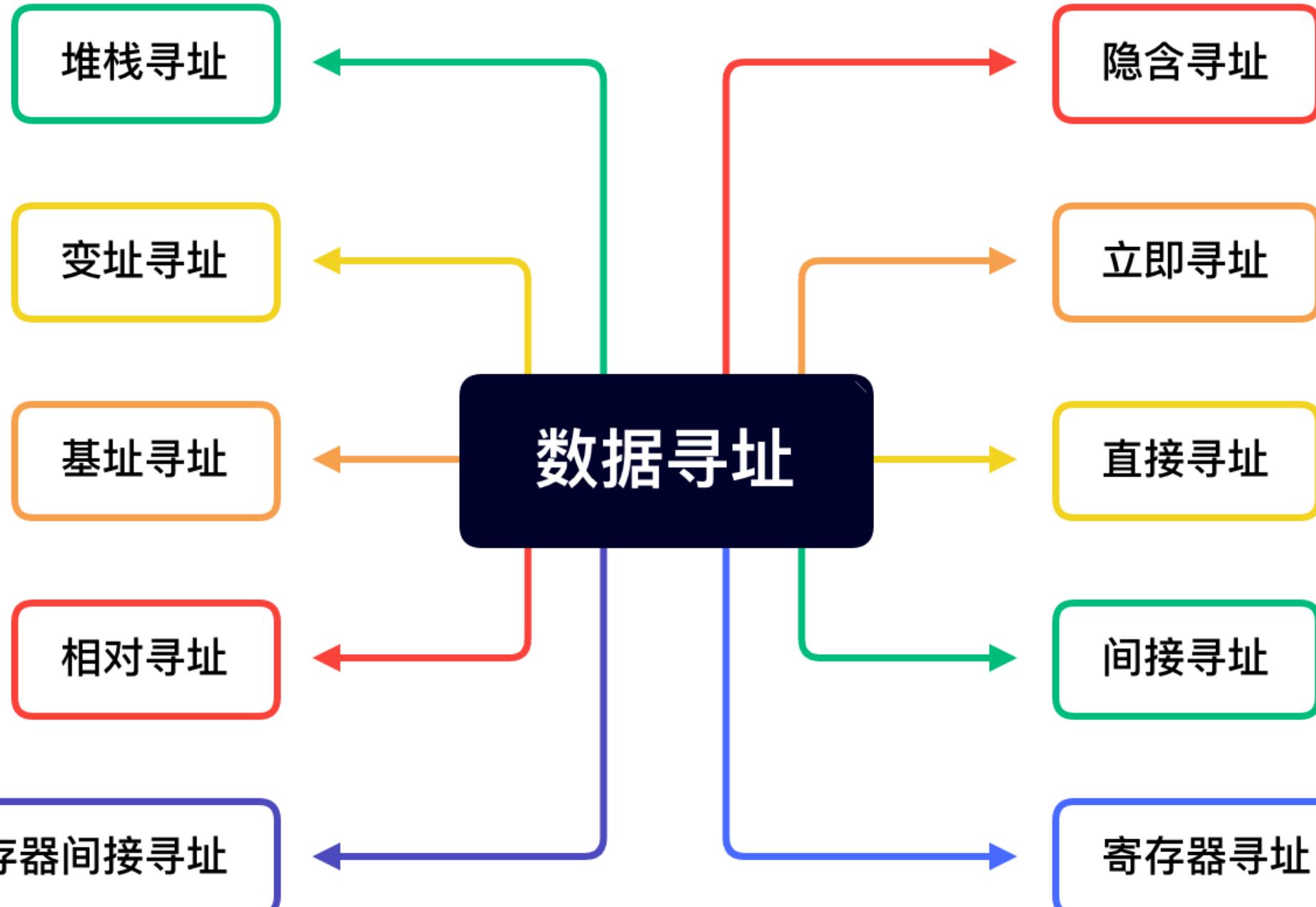
A1

A2

A3

①

寻址特征和形式地址求出来的真正对应到存储器的地址称之为有效地址



0000

操作码

寻址特征

形式地址



3. 指令系统设计与CPU运行控制

3.1.3 数据寻址方式：隐含寻址



3. 指令系统设计与CPU运行控制

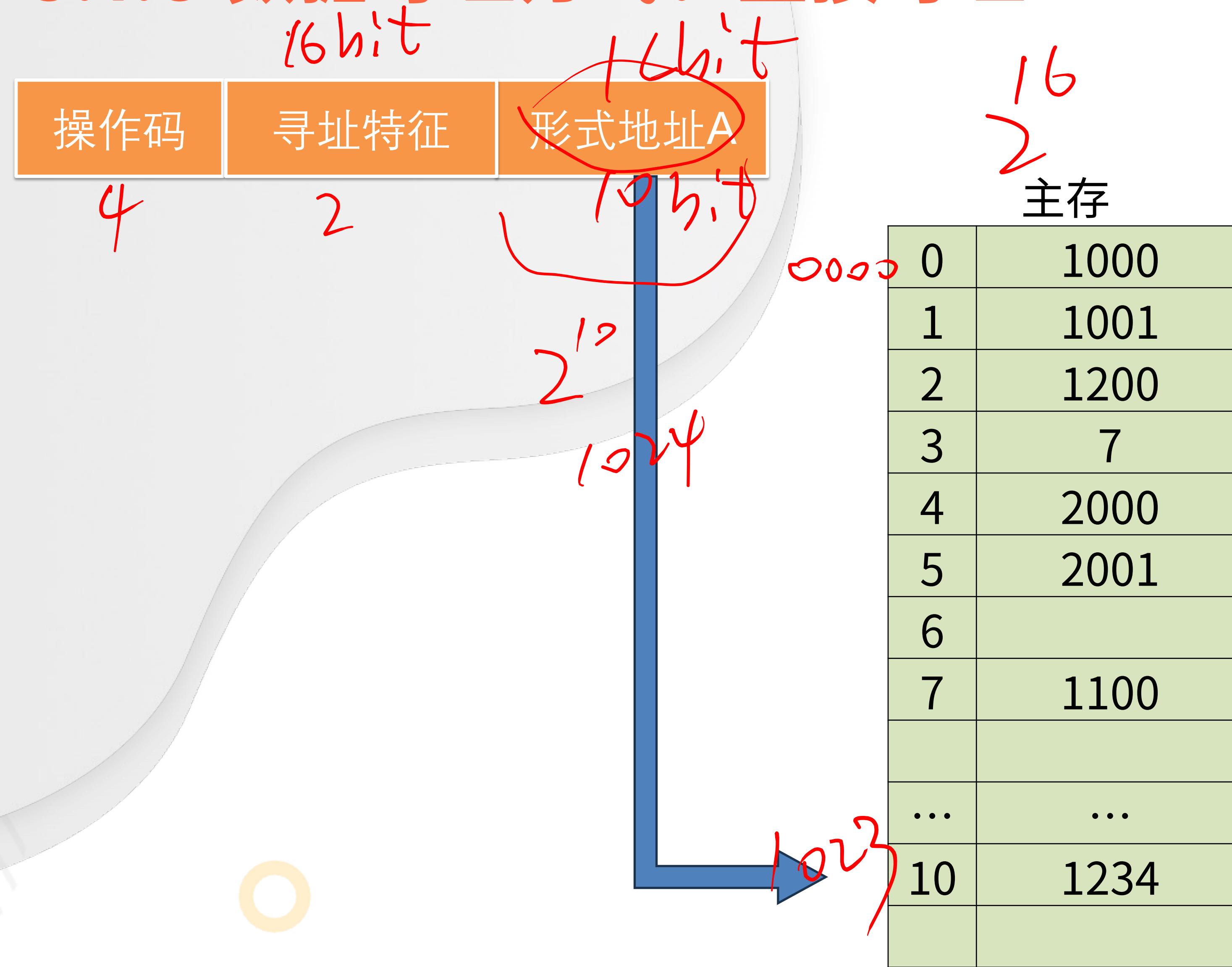
3.1.3 数据寻址方式：立即寻址



mov R0, #10

3. 指令系统设计与CPU运行控制

3.1.3 数据寻址方式：直接寻址



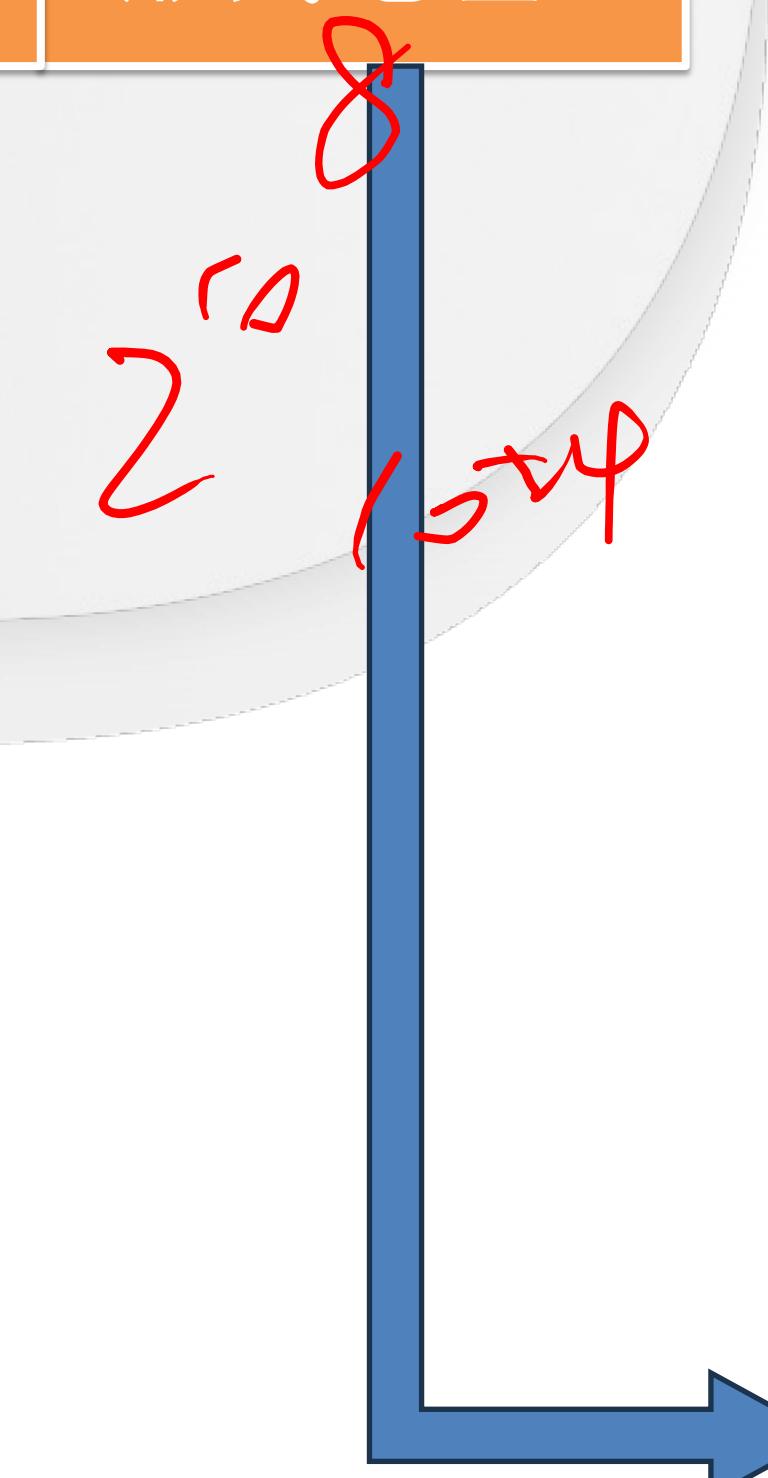
特点：

- 1、简单
- 2、访问一次主存
- 3、位数有限，范围有限
- 4、不可变



3. 指令系统设计与CPU运行控制

3.1.3 数据寻址方式：间接寻址



32bit

主存

0	1000
1	1001
2	1200
3	7
4	2000
5	2001
6	
7	1100
8	10
...	...
10	1234

特点：

- 1、寻址范围扩大
- 2、多次访存
- 3、不常用

一次

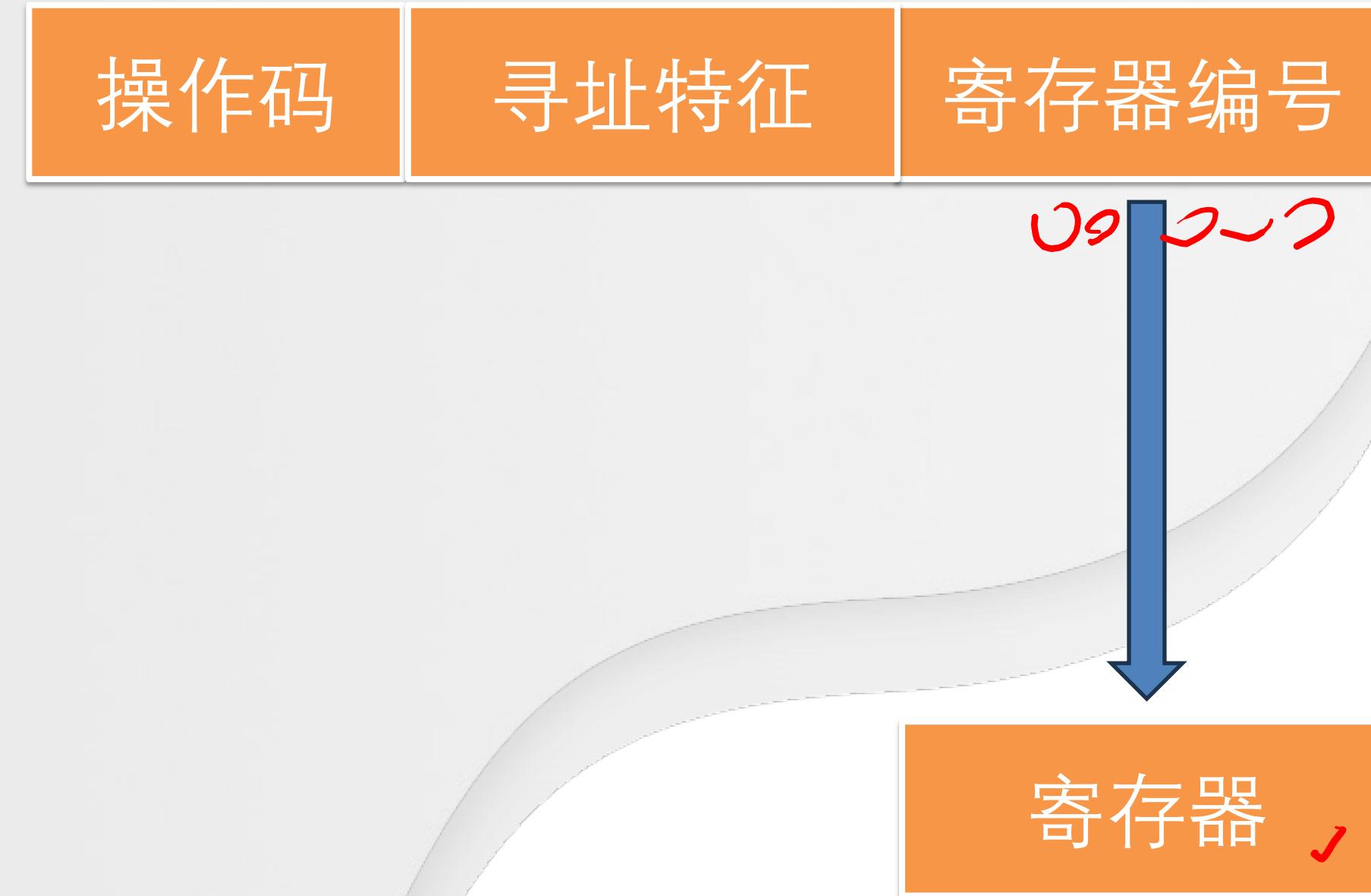
一

二次



3. 指令系统设计与CPU运行控制

3.1.3 数据寻址方式：寄存器寻址



Sub R0 R1 R2

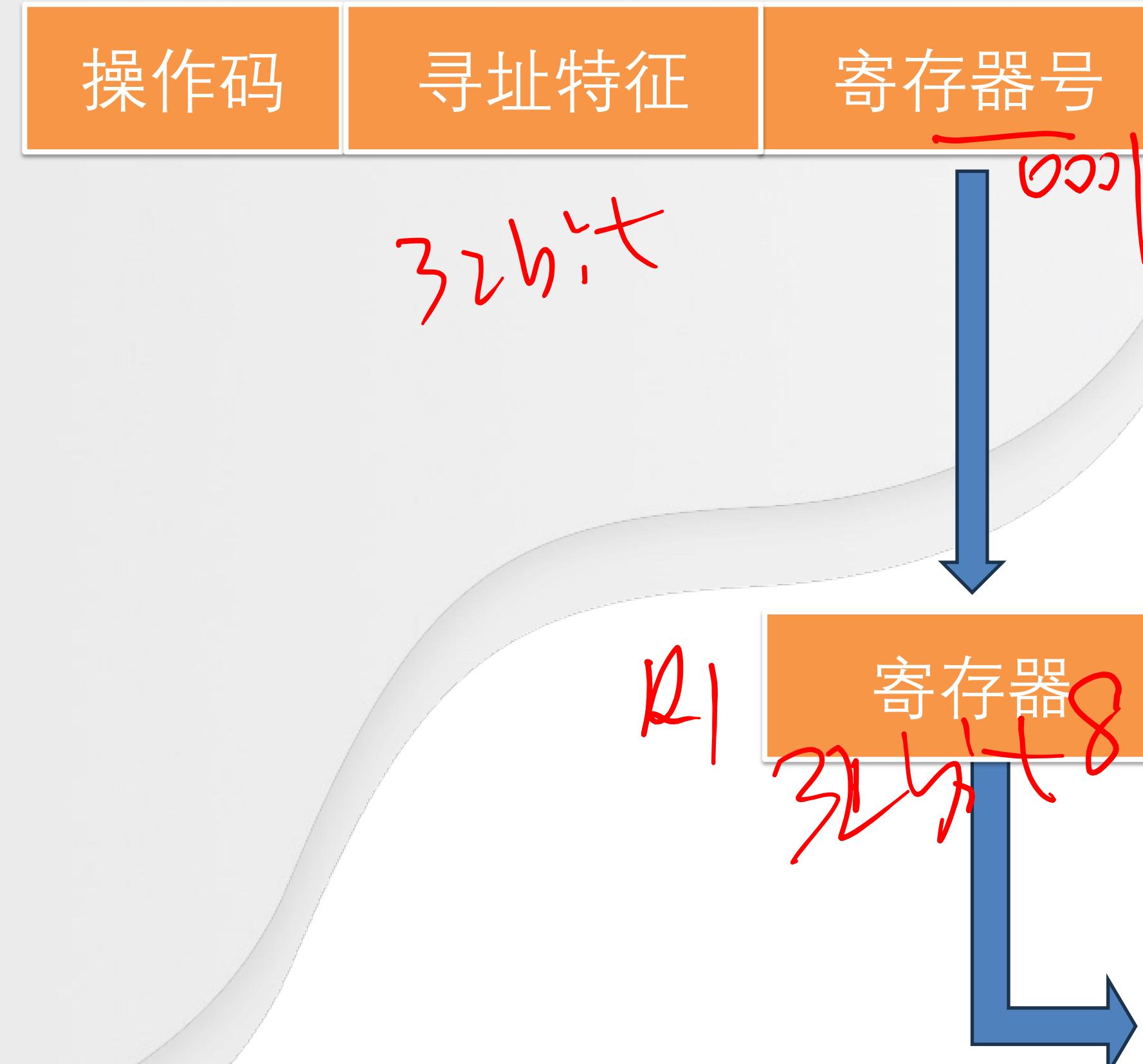
--特点：

- 1、速度快
- 2、容量小
- 3、重得利用
- 4、可能要备份数据到内存

压栈
出栈

3. 指令系统设计与CPU运行控制

3.1.3 数据寻址方式：寄存器间接寻址



主存	
0	1000
1	1001
2	1200
3	7
4	2000
5	2001
6	
7	1100
8	10
...	...
10	1234

特点：

- 1、寻址范围扩大
- 2、一次访存
- 3、速度比间接寻址快
- 4、常用



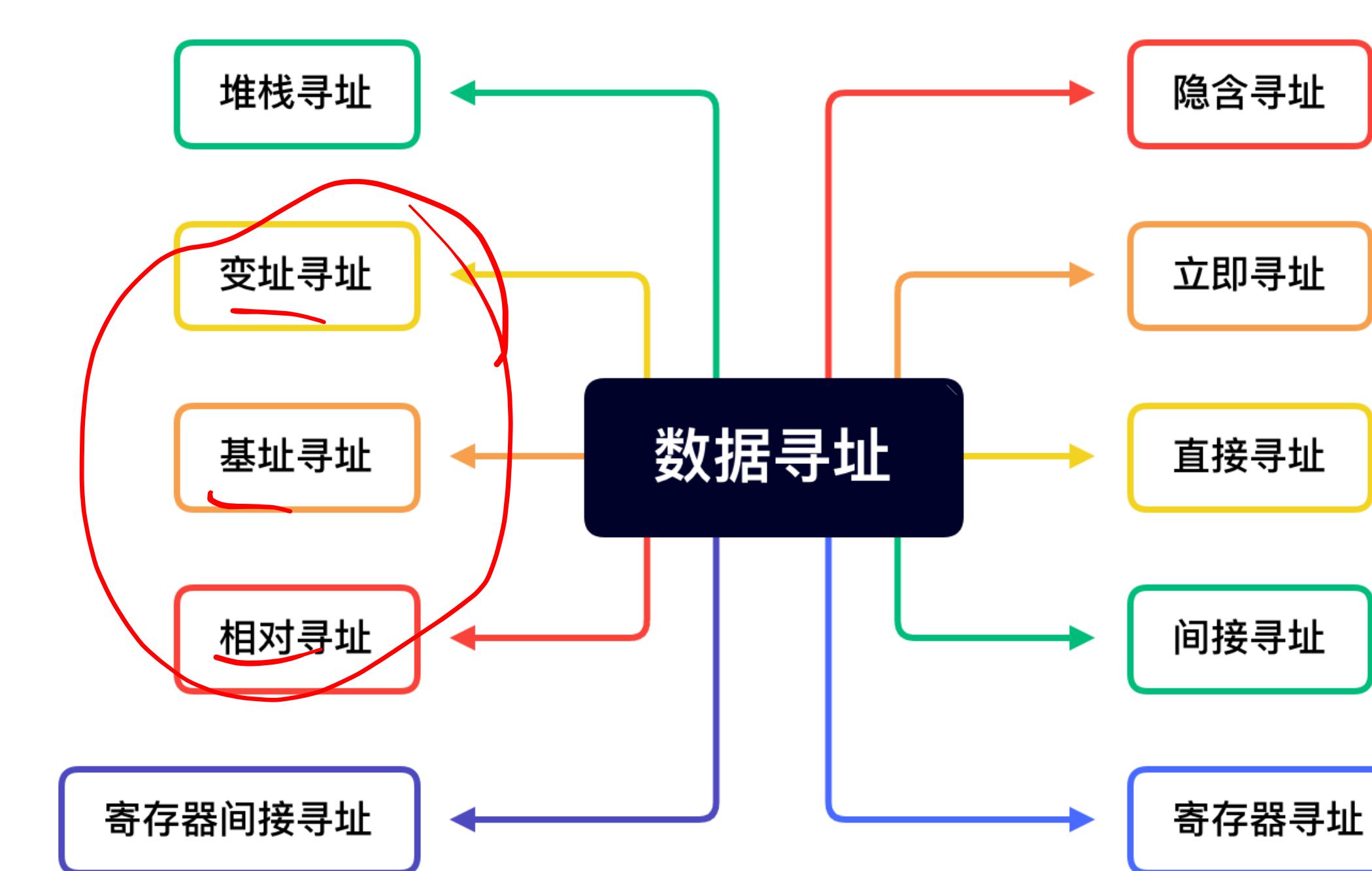
3. 指令系统设计与CPU运行控制

3.1.3 数据寻址方式：偏移寻址

$$EA = (IX) + A$$

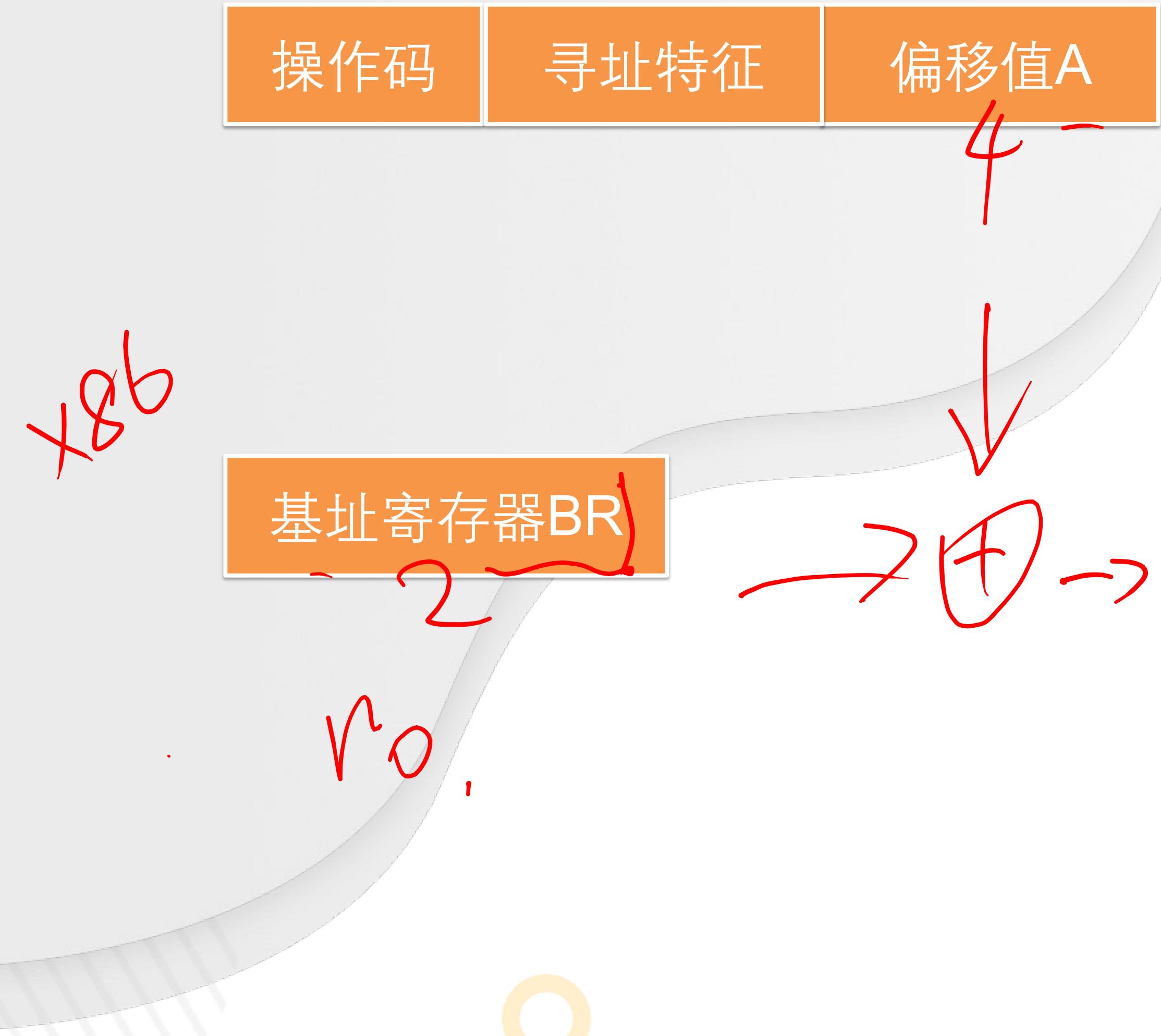
$$EA = (BR) + A$$

$$EA = (PC) + A$$



3. 指令系统设计与CPU运行控制

3.1.3 数据寻址方式：基址寻址



主存	
0	...
1	100
2	101
3	102
4	103
5	104
6	105
7	106
8	107
...	...

特点：

- 1、寻址范围扩大
- 2、用于切换数据区域
- 3、常由操作系统或管理使用
- 4、也可使用通用寄存器代替BR

3. 指令系统设计与CPU运行控制

3.1.3 数据寻址方式：变址寻址



主存

0	...
1	100
2	101
3	102
4	103
5	104
6	105
7	106
8	107
...	...

int [特点: 基址]

- 1、寻址范围扩大
- 2、用于用户空间连续空间访问
- 3、指令不变

3. 指令系统设计与CPU运行控制

3.1.3 数据寻址方式：相对寻址



程序计数器PC

主存

0	...
1	100
2	101
3	102
4	103
5	104
6	105
7	106
8	107
...	...

特点：

- 1、目标地址不固定，随指令执行而且变化
- 2、广泛应用于跳转

13

JMP.

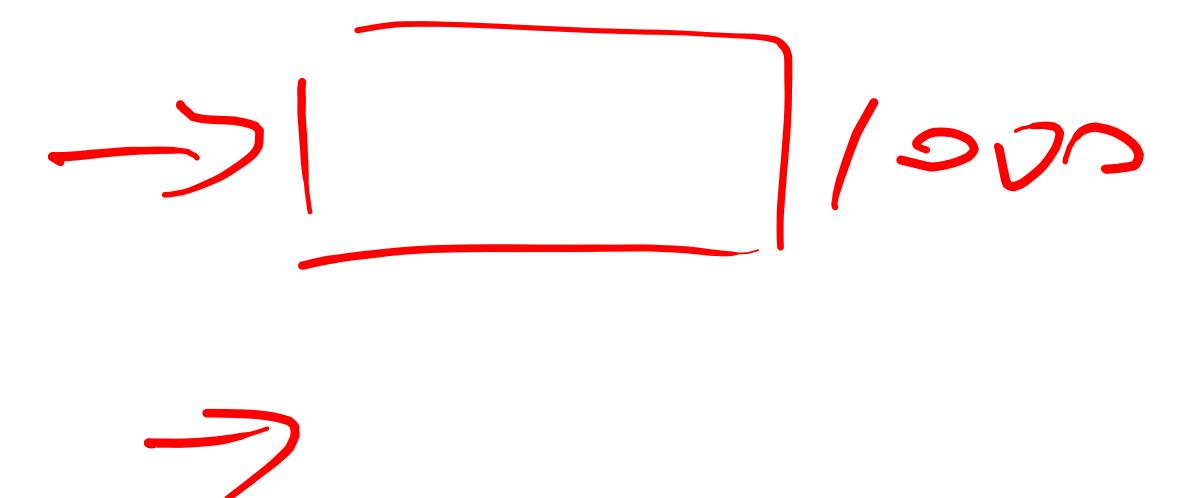


3. 指令系统设计与CPU运行控制

3.1.3 数据寻址方式：堆栈寻址

压栈、出栈操作，不指定操作内容地址，默认使用 sp 寄存器的地址

POP. r_0, r_1, r_2 .





3. 指令系统设计与CPU运行控制

3.1.3 数据寻址总结

寻址方式	有效地址	访存次数
隐含	指令直接指定	0
立即	指令中的A为操作数	0
直接	$EA = A$	1
一次间接	$EA = (A)$	2
寄存器	$EA = Rj$	0
寄存器间接一次	$EA = (Rj)$	1
相对	$EA = (PC) + A$	1
基址	$EA = (BR) + A$	1
变址	$EA = (IX) + A$	1

3. 指令系统设计与CPU运行控制

3.1.3 练习

例2：设机器字长32位，一个容量为16MB的存储器，CPU按半字寻址，寻址单元数是 ()。

- A. 2^{24}
- B. 2^{23}
- C. 2^{22}
- D. 2^{21}

$$\begin{array}{l} \text{KB} \\ 2^{10} \\ \hline 2^{24} \text{ 半字} \end{array}$$

16M

2²⁴/2²

$$\begin{array}{r} 2^{24} \\ \hline 2^2 \end{array}$$

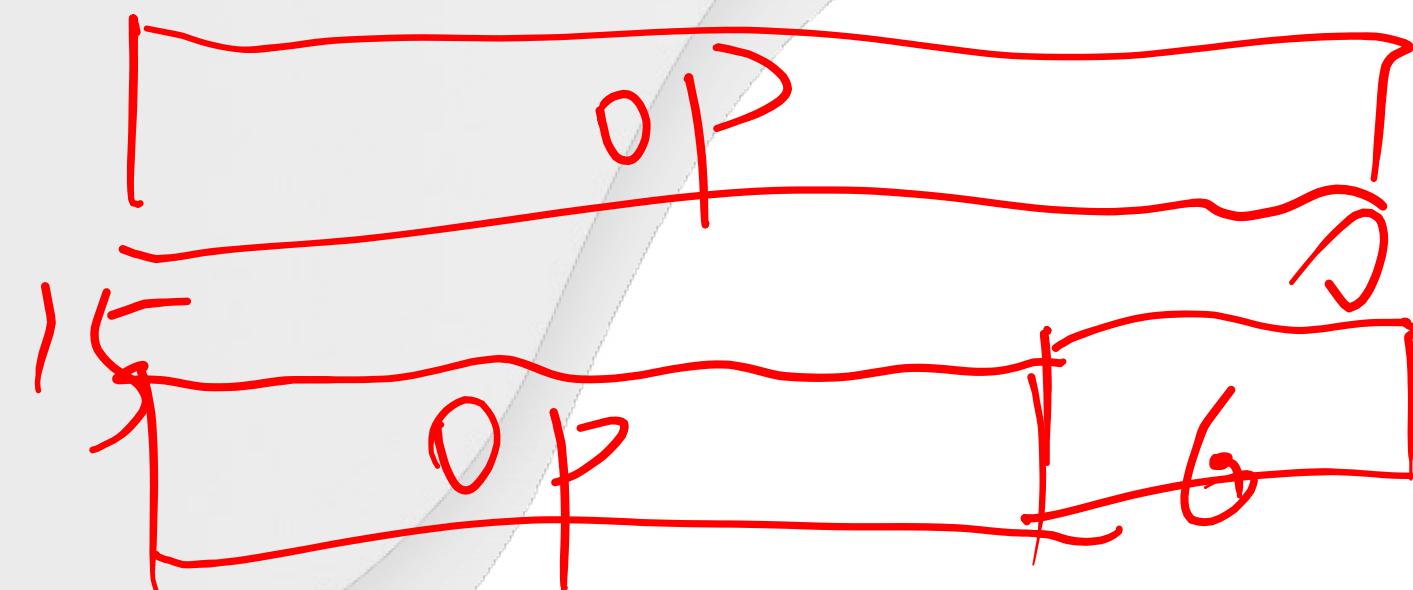


3. 指令系统设计与CPU运行控制

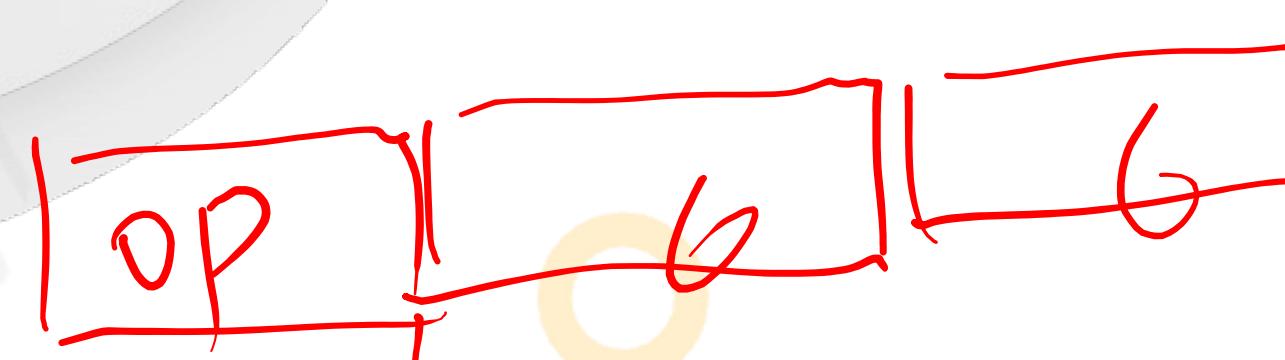
3.1.3 练习

例3【2022统考】：设计某指令系统时，假设采用16位定长指令字格式，操作码使用扩展编码方式，地址码为6位，包含零地址、一地址、二地址3种格式指令，若二地址指令有12条，一地址指令有254条，则零地址指令的条数最多为 ()。

- A. 0
- B. 2
- C. 64
- D. 128



$$\begin{aligned} & \frac{2^{16}}{254 \times 2^6} - 12 \times 2^{12} = 128 \\ & 2^{16} \\ & 254 \times 2^6 \\ & 12 \times 2^{12} \end{aligned}$$





3. 指令系统设计与CPU运行控制

3.1.3 练习

2⁶
6位

例4：某模型机共有64种操作码，位数固定，且具有以下特点：

- 1) 采用一地址或二地址格式。
特地2位
- 2) 有寄存器寻址、直接寻址、相对寻址（位移量为-128 ~ 127）
3种
- 3) 有16个通用寄存器，算术运算和逻辑运算的操作数均在寄存器中，
结果也在寄存器中。
2⁴
- 4) 取数/存数指令在通用寄存器和存储器之间传递数据。
- 5) 存储器容量为1MB，按字节编址。
2²⁰ *20 bit*

要求设计算术逻辑指令、取数/存数指令和相对转移指令的格式，并简述理由。

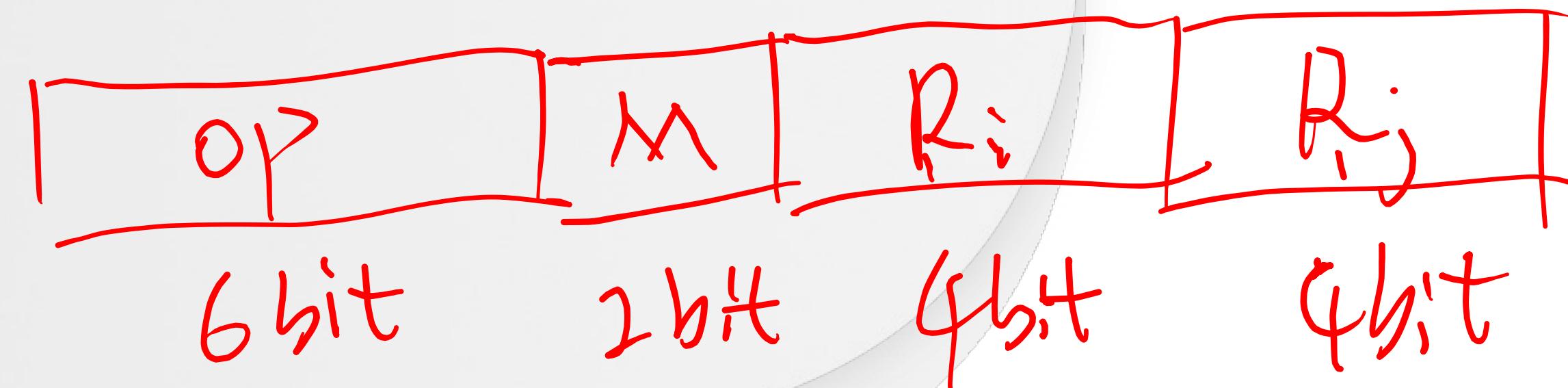
PC



3. 指令系统设计与CPU运行控制

3.1.3 练习

设计算术逻辑指令格式

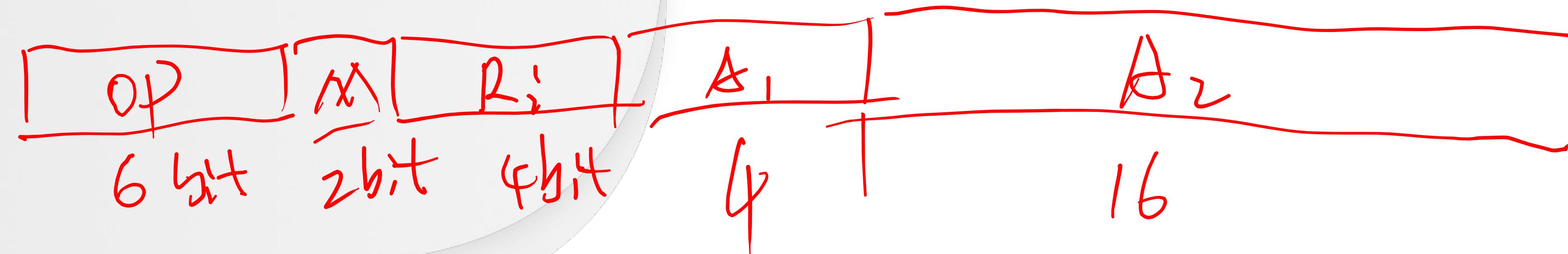




3. 指令系统设计与CPU运行控制

3.1.3 练习

设计取数/存数指令格式





3. 指令系统设计与CPU运行控制

3.1.3 练习

设计相对转移指令的格式





3. 指令系统设计与CPU运行控制



3.1.4 CISC与RISC

100%

CISC (Complex Instruction Set Computing) 、 RISC (Reduced Instruction Set Computing)

复杂

精简

特性	CISC	RISC
指令集复杂度	复杂，指令数量多，功能丰富	简单，指令数量少，功能单一
执行时间	指令执行时间不一，从一个时钟周期到多个时钟周期不等	大部分指令在一个时钟周期内完成
内存访问	指令可以直接操作内存	主要使用加载/存储指令操作内存
硬件与软件的角色	更依赖于硬件，硬件更复杂	依赖于软件（编译器优化），硬件更简单
典型应用	通用计算机，如桌面和服务器	嵌入式系统和移动设备
性能优化	通过复杂的硬件和微码优化	通过简化硬件和提高指令执行效率优化
能源效率	通常功耗更高	通常更节能
历史和发展	起源于存储器昂贵时代，减少程序大小	随存储器价格下降和编译器技术进步而发展

MIPS

PowerPC



3. 指令系统设计与CPU运行控制

3.1.5 指令的执行

X86处理器的主要寄存器

32bit

31	16 15	8 7	0
E	AH	AL	
E	BH	BL	
E	CH	CL	
E	DH	DL	
	ESI		
	EDI		
	EBP		
	ESP		

16bit	32bit	用途说明
AX	EAX	累加器 Acc
BX	EBX	基地址寄存器
CX	ECX	计数寄存器
DX	EDX	数据寄存器
	ESI	变址寄存器
	EDI	变址寄存器
	EBP	堆栈基指针
	ESP	堆栈寄存器
	EFLAGS	标志寄存器

2F 9F .FF FF



3. 指令系统设计与CPU运行控制

3.1.5 指令的执行

ARM处理器的主要寄存器

32bit



寄存器类别	寄存器在汇编中的名称	各模式下实际访问的寄存器						
		用户	系统	管理	中止	未定义	中断	快中断
通用寄存器和程序计数器	R0(a1)				R0			
	R1(a2)				R1			
	R2(a3)				R2			
	R3(a4)				R3			
	R4(v1)				R4			
	R5(v2)				R5			
	R6(v3)				R6			
	R7(v4)				R7			
	R8(v5)			R8			R8_fiq	
	R9(SB,v6)			R9			R9_fiq	
	R10(SL,v7)			R10			R10_fiq	
	R11(FP,v8)			R11			R11_fiq	
	R12(IP)			R12			R12_fiq	
状态寄存器	R13(SP)	R13	R13_svc	R13_abt	R13_und	R13_irq	R13_fiq	
	R14(LR)	R14	R14_svc	R14_abt	R14_und	R14_irq	R14_fiq	
状态寄存器	R15(PC)			R15				
	CPSR			CPSR			CH2	
	SPSR	无	SPSR_abt	SPSR_abt	SPSR_und	SPSR_irq	SPSR_fiq	



3. 指令系统设计与CPU运行控制

3.1.5 AT&T 和 Intel 两种汇编格式

- 1、AT&T指令只能使用小写字母，Intel格式大小写不敏感
- 2、AT&T中第一个为源操作数，第二个为目的操作数，Intel反的
- 3、AT&T中寄存器要加前缀“%”，立即数要加“\$”，Intel不需要
- 4、AT&T内存寻址用“()”，Intel使用“[]”
- 5、处理复杂寻址方式时，AT&T格式中“8(%edx, %eax, 2)”表示M[R[edx]+R[eax]*2+8]，Intel格式表示为 “[edx + eax*2 + 8]
- 5、AT&T在数据后面加字母表示长度b表示byte (字节)，w表示word (字)l表示long (双字)，Intel是在操作码后面注明 byte ptr, word ptr, dword ptr

伪指令

MOV R1, R2

8 64 1~8
8 32 64



3. 指令系统设计与CPU运行控制

3.1.5 AT&T 和 Intel 两种汇编格式

AT&T格式	Intel格式	含义
<code>mov \$100, %eax</code>	<code>mov eax, 100</code>	$100 \rightarrow R[\text{eax}]$
<code>mov %eax, %ebx</code>	<code>mov ebx, eax</code>	$R[\text{eax}] \rightarrow R[\text{ebx}]$
<code>mov %eax, (%ebx)</code>	<code>mov [ebx], eax</code>	$R[\text{eax}] \rightarrow M[R[\text{ebx}]]$
<code>mov %eax, -8(%ebp)</code>	<code>mov [ebp-8], eax</code>	$R[\text{eax}] \rightarrow M[R[\text{ebp}-8]]$
<code>lea 8(%edx, %eax, 2), %eax</code>	<code>lea eax, [edx+eax*2+8]</code>	$R[\text{edx}]+R[\text{eax}]*2+8 \rightarrow R[\text{eax}]$
<code>movl %eax, %ebx</code>	<code>mov dword ptr ebx, eax</code>	长度为4字节的 $R[\text{eax}] \rightarrow R[\text{ebx}]$



3. 指令系统设计与CPU运行控制

3.1.5 常见指令

X86处理器：

- 1、数据传送：mov, push, pop, load, store
- 2、算术与逻辑运算: add, sub, inc, dec, imul, idiv, and, or, xor, not, neg, shl, shr
- 3、控制流: jmp, j+条件码 (je, jne, jz, jg, jge, jl, jle), cmp, test, call, ret, int ~~80~~
- 4、输入输出类: in, out

ARM处理器：

- 1、数据传送: mov, mvn
- 2、算术与逻辑运算: add, adc, sub, subc, cmp, cmn, and, orr, eor, bic, tst, teq,
mul, mla, umull, umlal, smull, smlal
- 3、控制流: b, bl, blx, bx
- 4、程序状态寄存器处理: msr mrs ^{CPSR}
- 5、加载存储: ldr, str, ldrb, strb, ldm, stm, swp



3. 指令系统设计与CPU运行控制

3.1.5 指令演示

```
1 int add(int x, int y) {  
2     return x + y;  
3 }  
  
4  
5 int caller(){  
6     int temp1 = 125;  
7     int temp2 = 80;  
8     int sum = add(temp1,temp2);  
9     return sum;  
10 }
```

x86 gcc 1.27 Compiler options

```
1 add:  
2     pushl %ebp  
3     movl %esp,%ebp  
4     movl 8(%ebp),%eax  
5     addl 12(%ebp),%eax  
6     jmp .L1  
7 .L1:  
8     leave  
9     ret  
10 caller:  
11     pushl %ebp  
12     movl %esp,%ebp  
13     subl $12,%esp  
14     movl $125,-4(%ebp)  
15     movl $80,-8(%ebp)  
16     pushl -8(%ebp)  
17     pushl -4(%ebp)  
18     call add  
19     movl %eax,-12(%ebp)  
20     movl -12(%ebp),%eax  
21     jmp .L2  
22 .L2:  
23     leave  
24     ret
```

AII



3. 指令系统设计与CPU运行控制

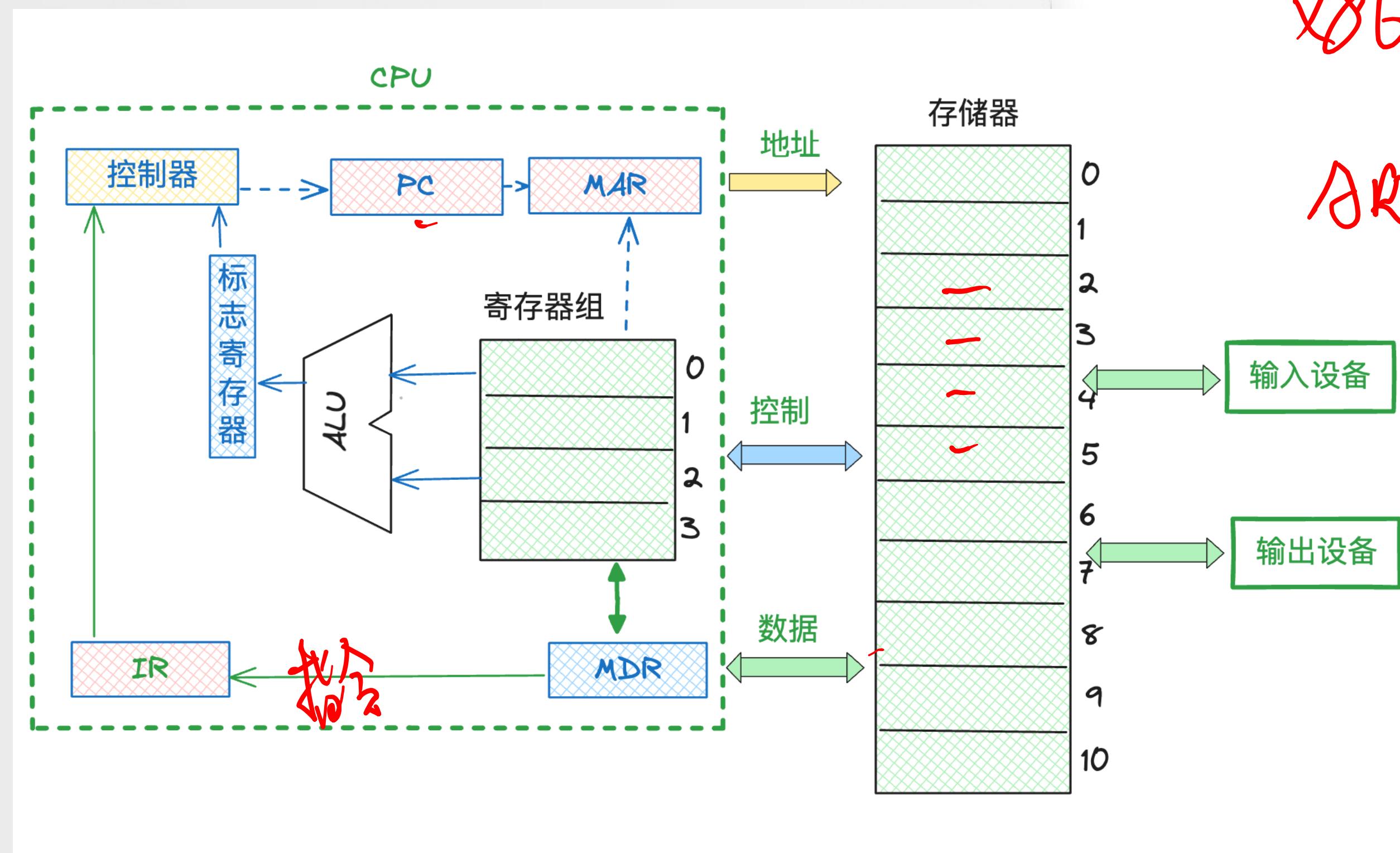
3.1.5 指令演示

```
x86 gcc 1.27          Compiler options...  
A Output... Filter... Libraries Overrides  
1 add:  
2     pushl %ebp  
3     movl %esp,%ebp  
4     movl 8(%ebp),%eax  
5     addl 12(%ebp),%eax  
6     jmp .L1  
7 .L1:  
8     leave  
9     ret  
10 caller:  
11    pushl %ebp  
12    movl %esp,%ebp  
13    subl $12,%esp  
14    movl $125,-4(%ebp)  
15    movl $80,-8(%ebp)  
16    pushl -8(%ebp)  
17    pushl -4(%ebp)  
18    call add  
19    movl %eax,-12(%ebp)  
20    movl -12(%ebp),%eax  
21    jmp .L2  
22 .L2:  
23    leave  
24    ret
```

```
x86 gcc 1.27 (Editor #1)          Compiler options...  
A Output... Filter... Libraries Overrides + Add new  
10 caller:  
11    55  
12    push    ebp  
13    89 e5  
14    mov     ebp,esp  
15    83 ec 0c  
16    sub     esp,0xc  
17    c7 45 fc 7d 00 00 00  
18    mov     DWORD PTR [ebp-0x4],0x7d  
19    c7 45 f8 50 00 00 00  
20    mov     DWORD PTR [ebp-0x8],0x50  
21    ff 75 f8  
22    push    DWORD PTR [ebp-0x8]  
23    ff 75 fc  
24    push    DWORD PTR [ebp-0x4]  
25    e8 fc ff ff ff  
26    call    2b <caller+0x1b>  
27    R_386_PC32 add  
28    89 45 f4  
29    mov     DWORD PTR [ebp-0xc],eax  
30    8b 45 f4  
31    mov     eax,DWORD PTR [ebp-0xc]  
32    eb 00  
33    jmp    37 <caller+0x27>  
34    c9  
35    leave
```

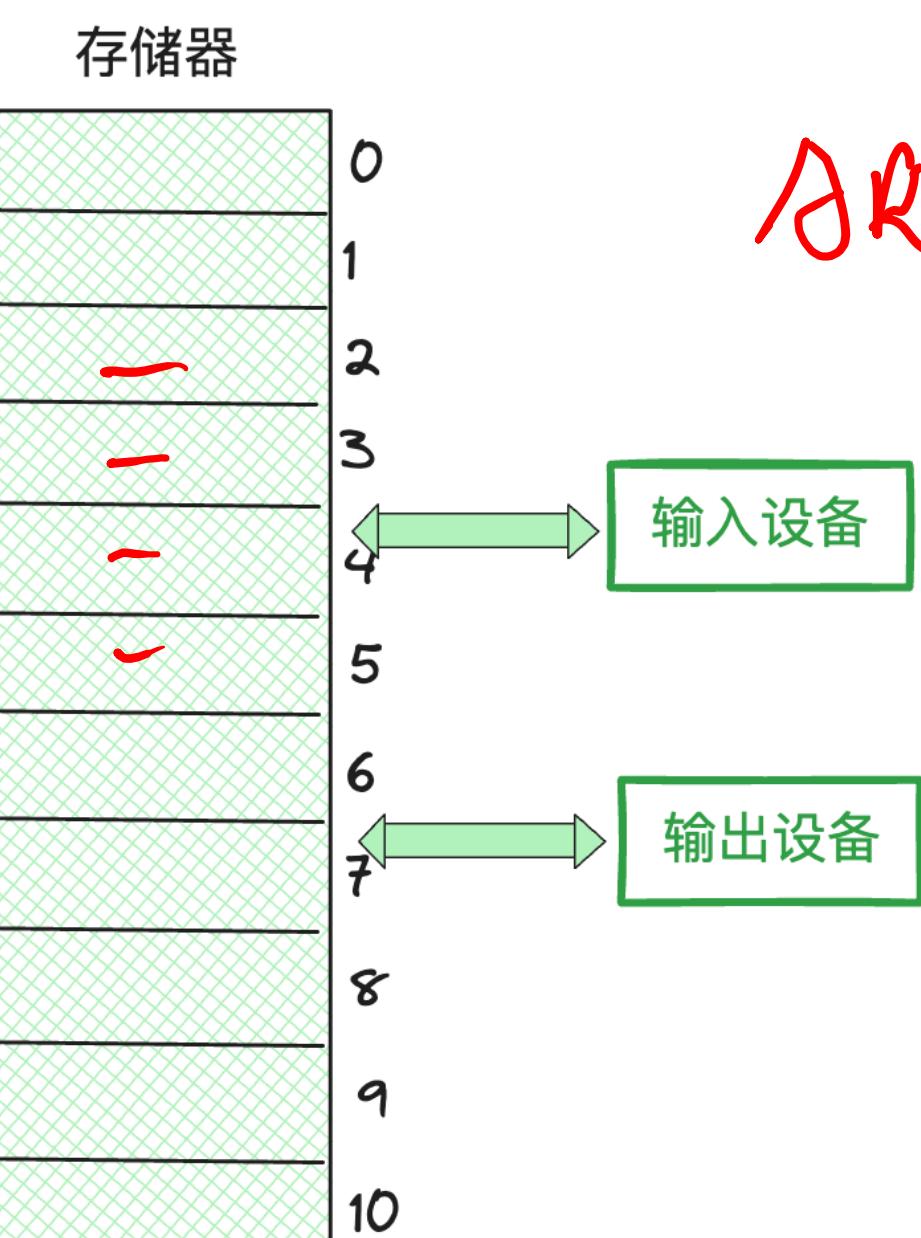
3. 指令系统设计与CPU运行控制

3.1.5 指令执行



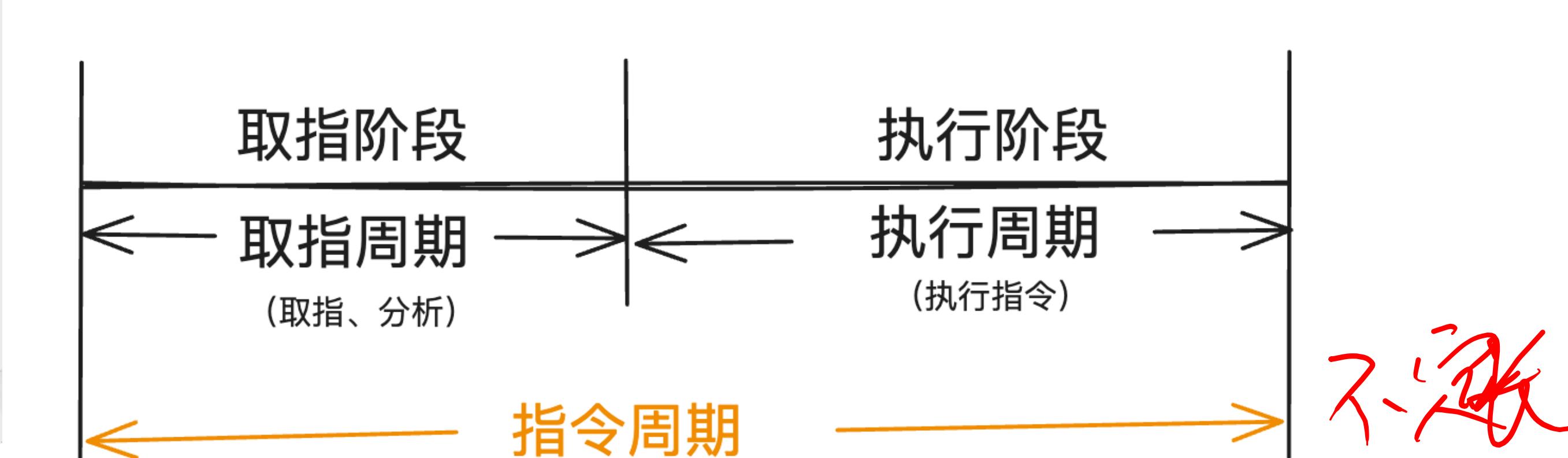
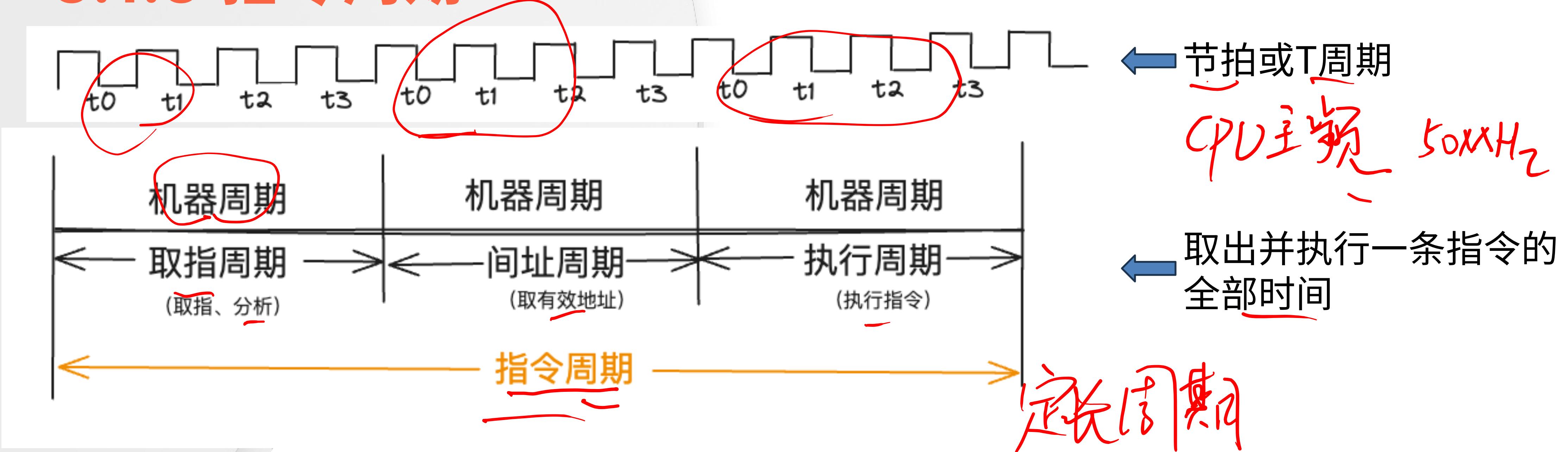
X86.

IN OUT
ARM ldr. str.



3. 指令系统设计与CPU运行控制

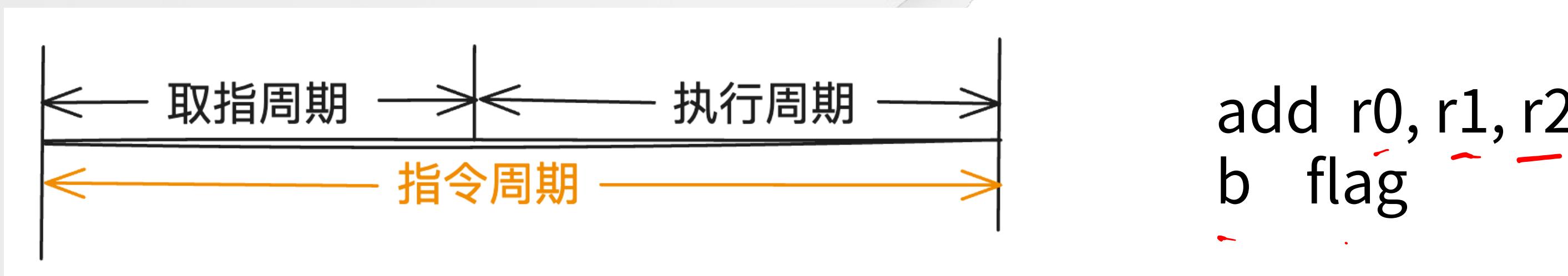
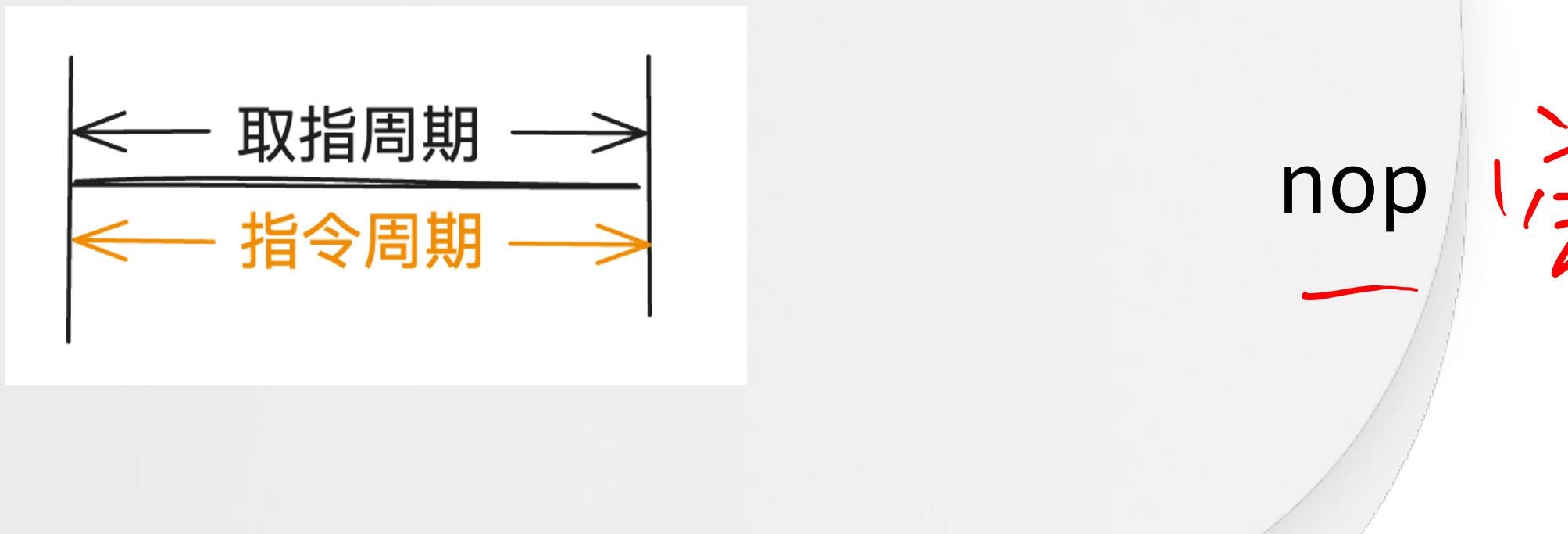
3.1.5 指令周期





3. 指令系统设计与CPU运行控制

3.1.5 指令周期差异



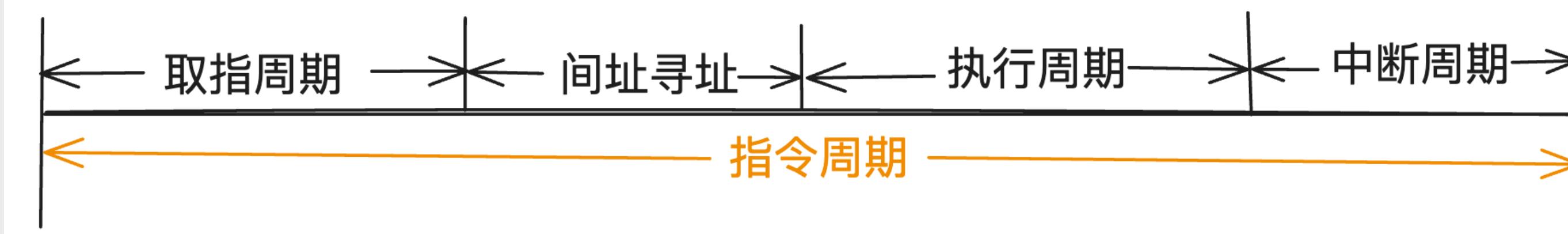


3. 指令系统设计与CPU运行控制

3.1.5 指令周期差异



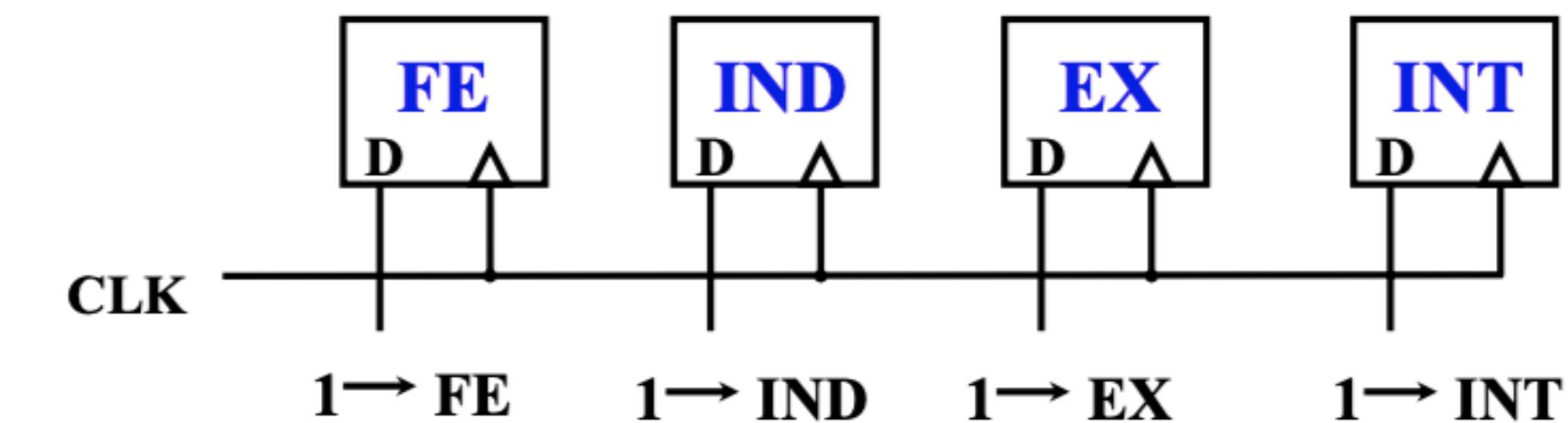
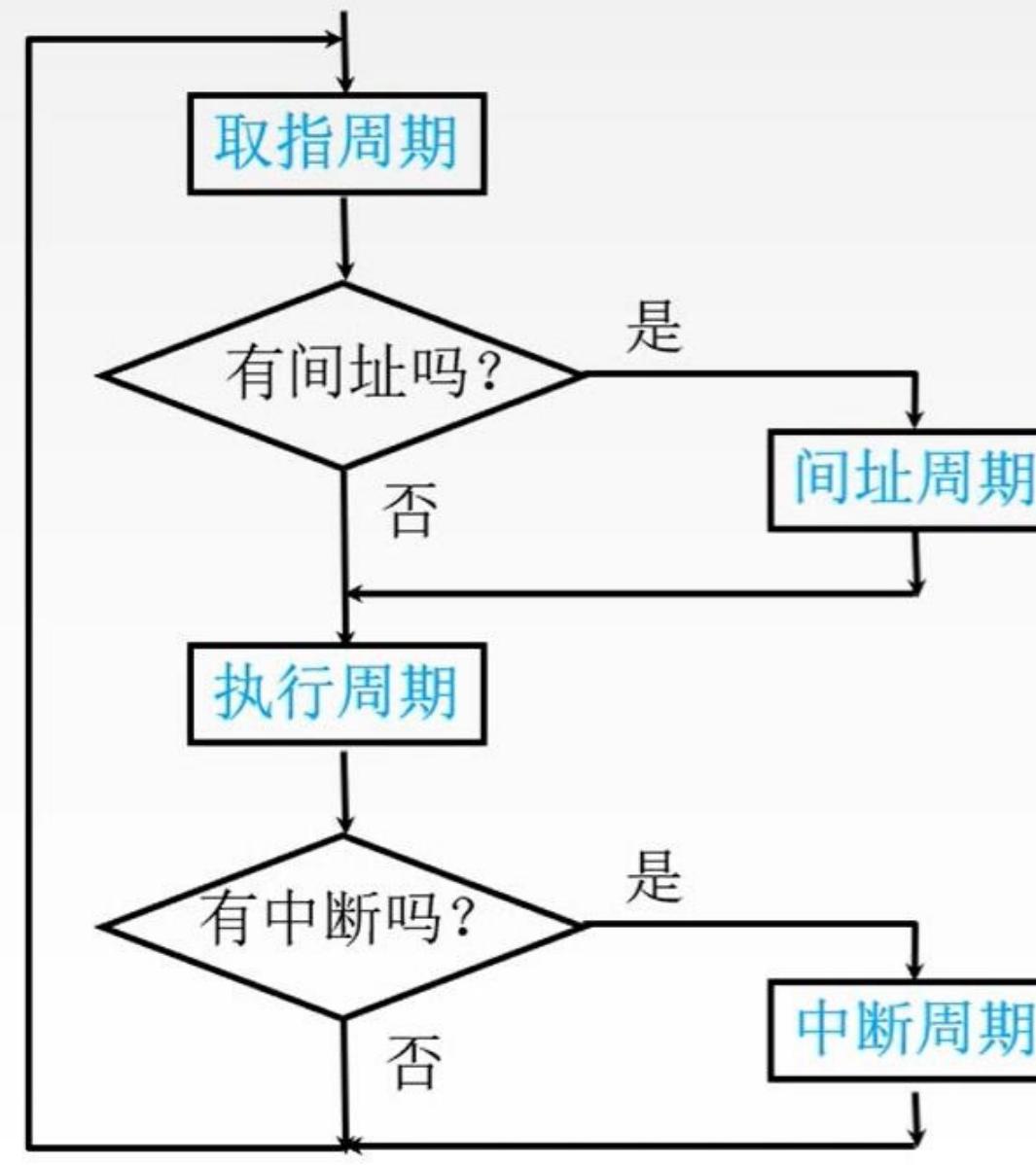
add r0,r1,(r2)





3. 指令系统设计与CPU运行控制

3.1.5 指令周期流程与控制



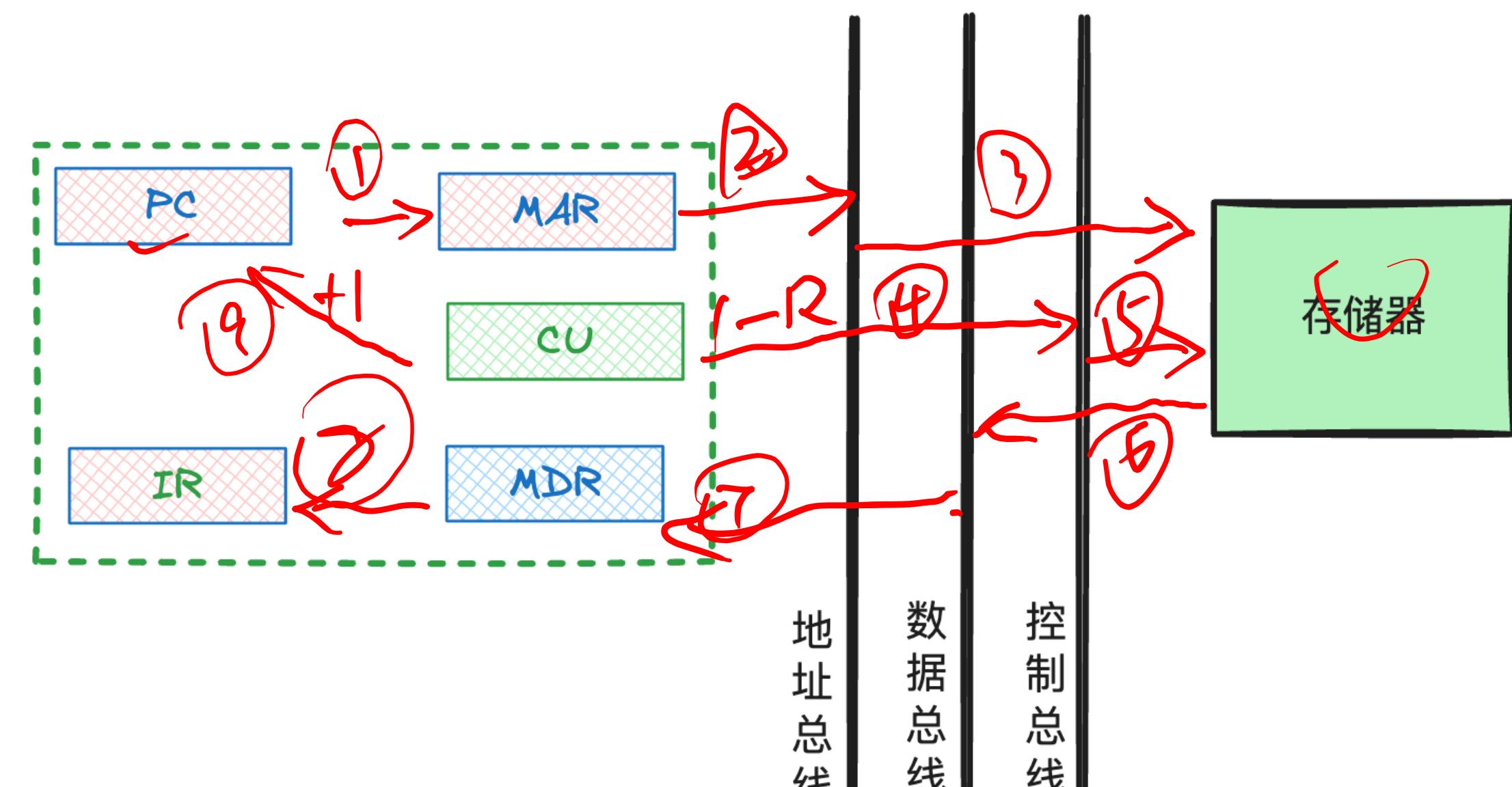
	FE	IND	EX	INT
取指周期	1	0	0	0
间址周期	0	1	0	0
执行周期	0	0	1	0
中断周期	0	0	0	1



3. 指令系统设计与CPU运行控制

3.1.6 取指周期的数据流

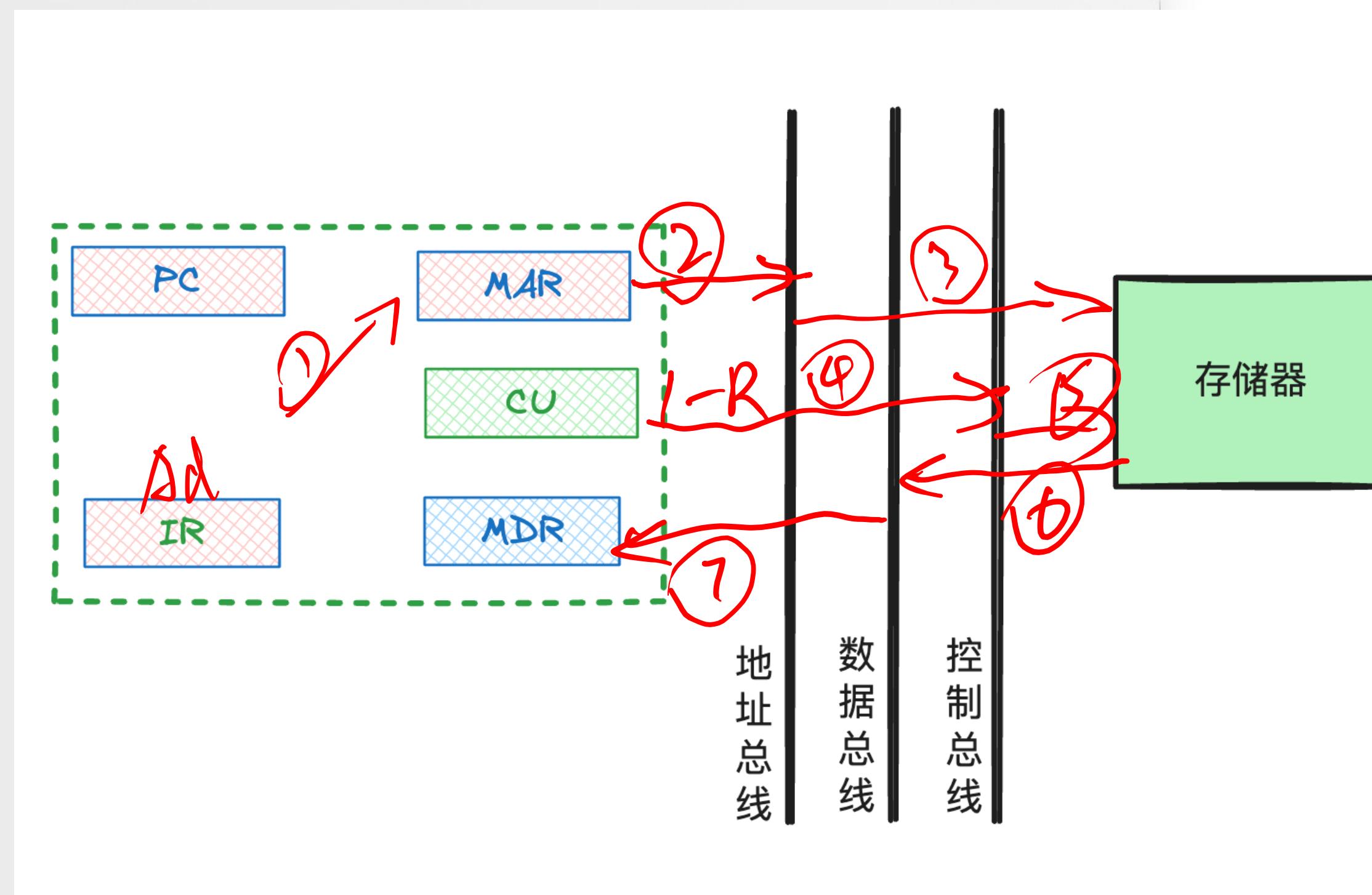
读





3. 指令系统设计与CPU运行控制

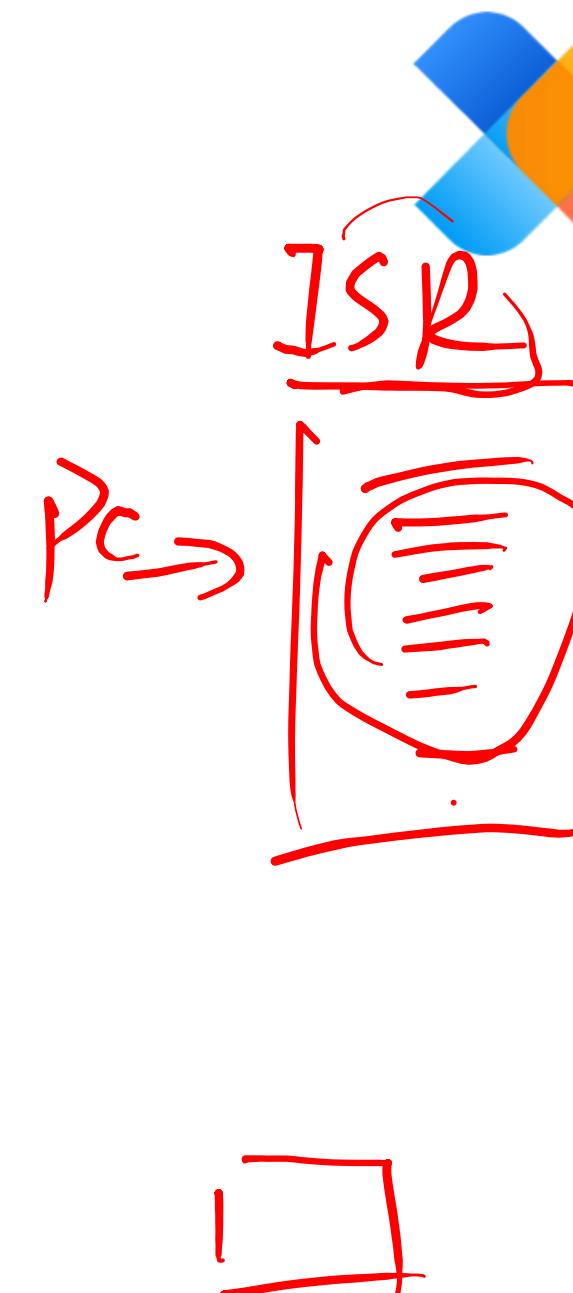
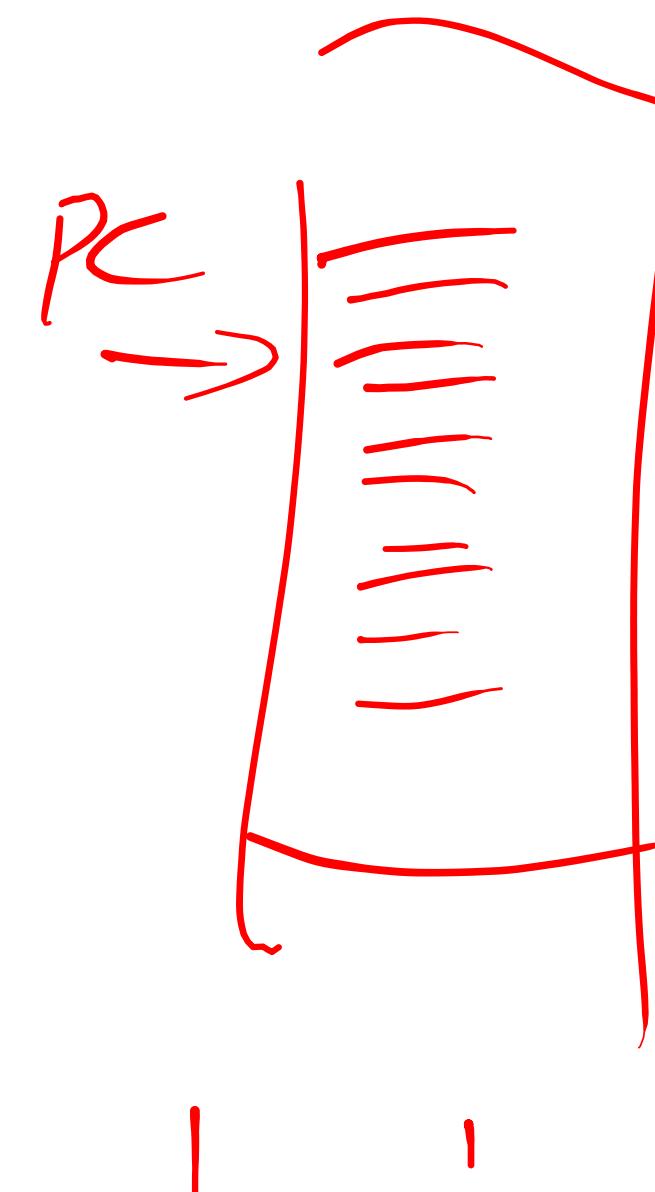
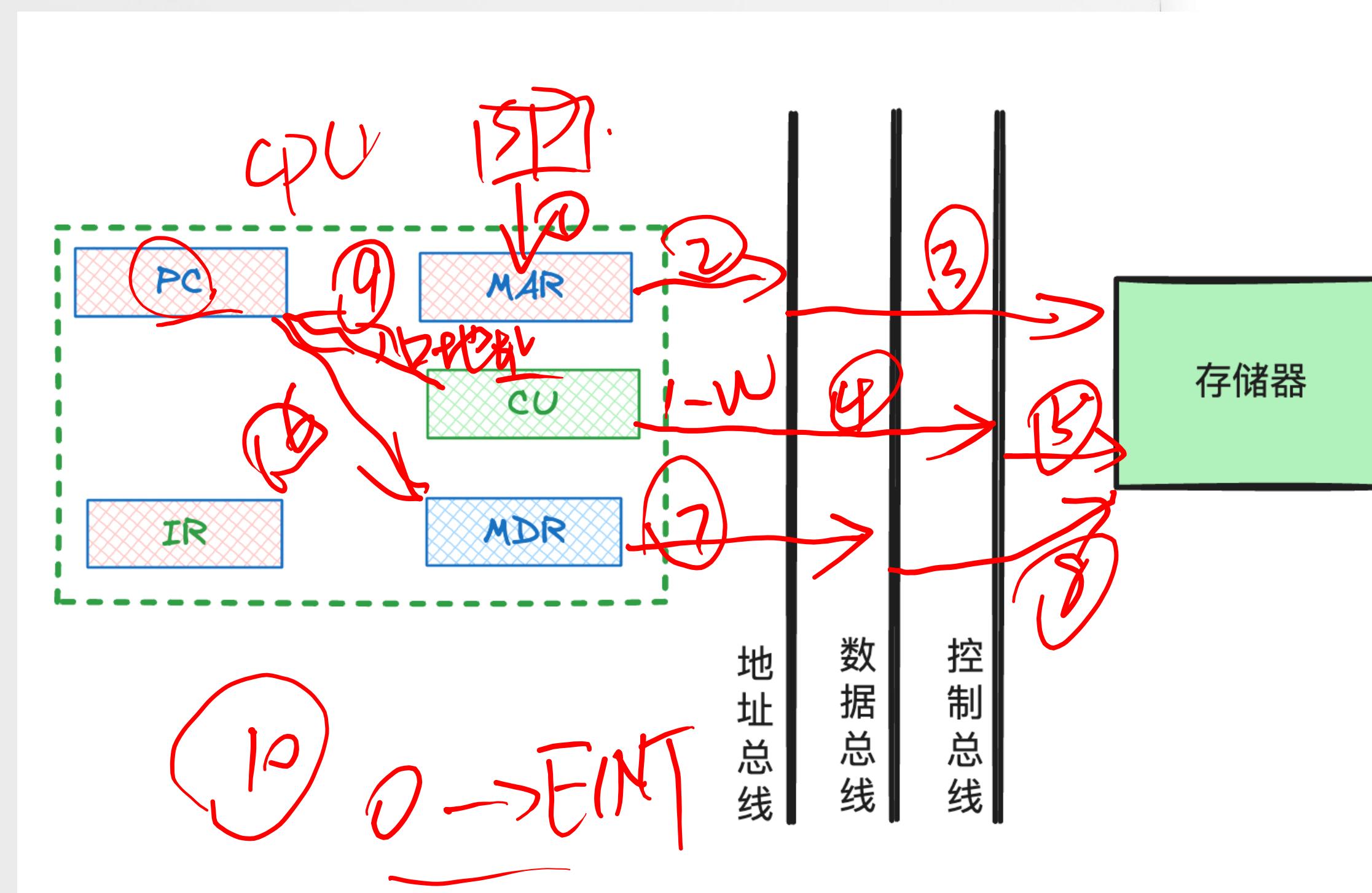
3.1.6 间址周期的数据流





3. 指令系统设计与CPU运行控制

3.1.6 中断周期的数据流





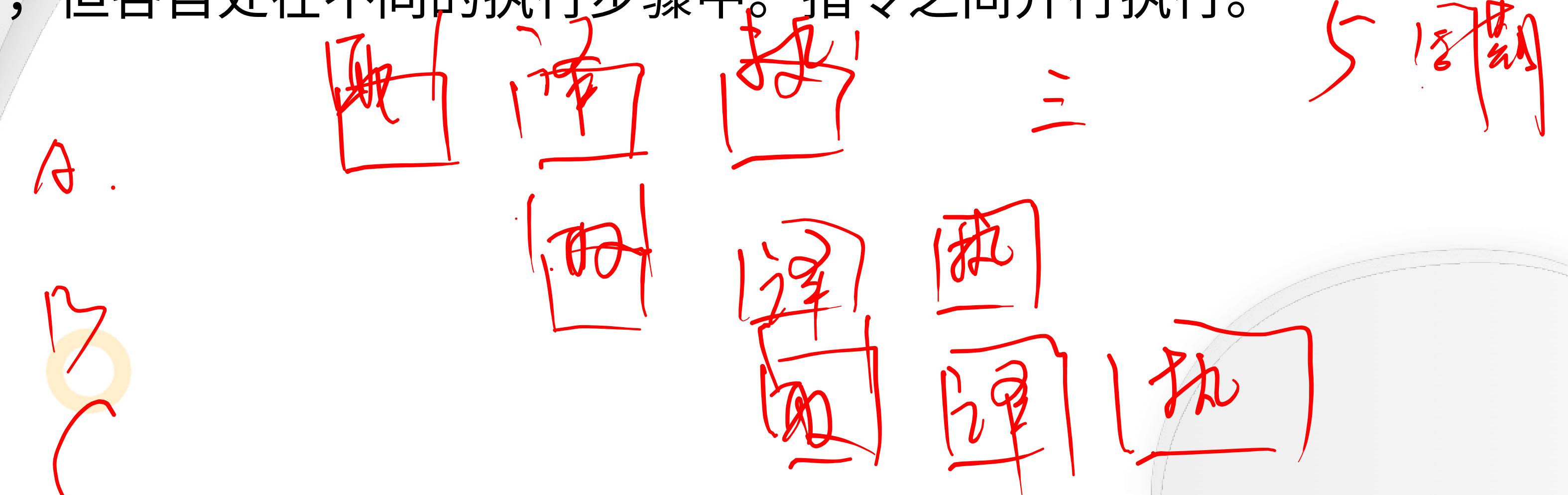
3. 指令系统设计与CPU运行控制

3.1.6 指令执行方案

1) 单指令周期：所有指令都选用相同的执行时间完成，指令串行执行，时长取决于执行时间最长的指令

2) 多指令周期：不同类型的指令选用不同的执行步骤来完成。指令之间串行执行，可选用不同个数的时钟周期来完成不同指令的执行过程

3) 流水线方案：在每一个时钟周期启动一条指令，尽量让多条指令同时运行，但各自处在不同的执行步骤中。指令之间并行执行。





3. 指令系统设计与CPU运行控制

3.1.6 指令执行控制方式

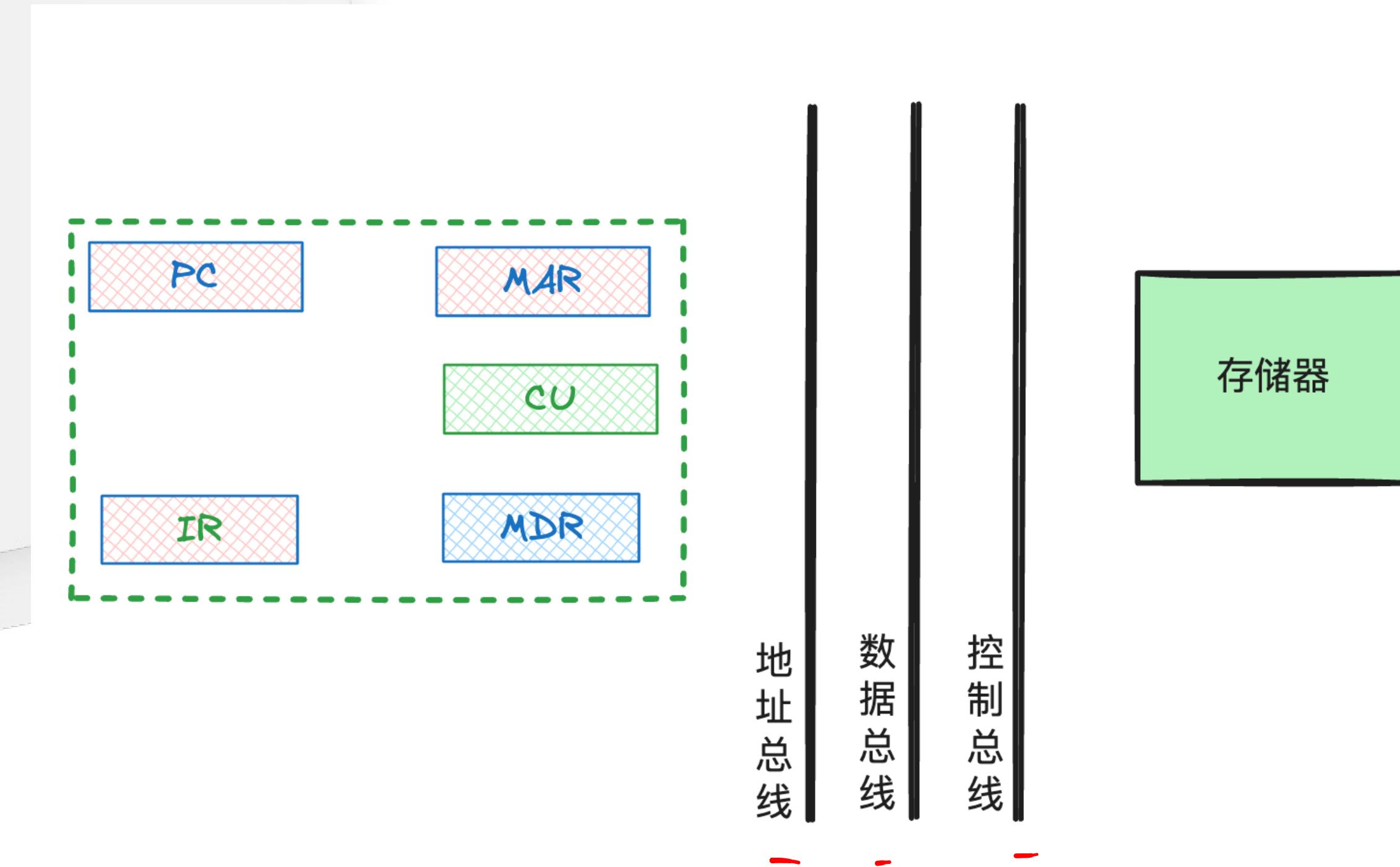
- 1) 同步控制：所有指令的执行都是由统一的基准时标的时序信号所控制，具体有完全统一节拍（定长）、不同节拍（不定长）、中央控制与局部控制相结合
- 2) 异步控制：没有基准时标，没有固定节拍和时钟同步，由专门的应答信号控制
- 3) 联合控制：同步和异步混合使用



3. 指令系统设计与CPU运行控制

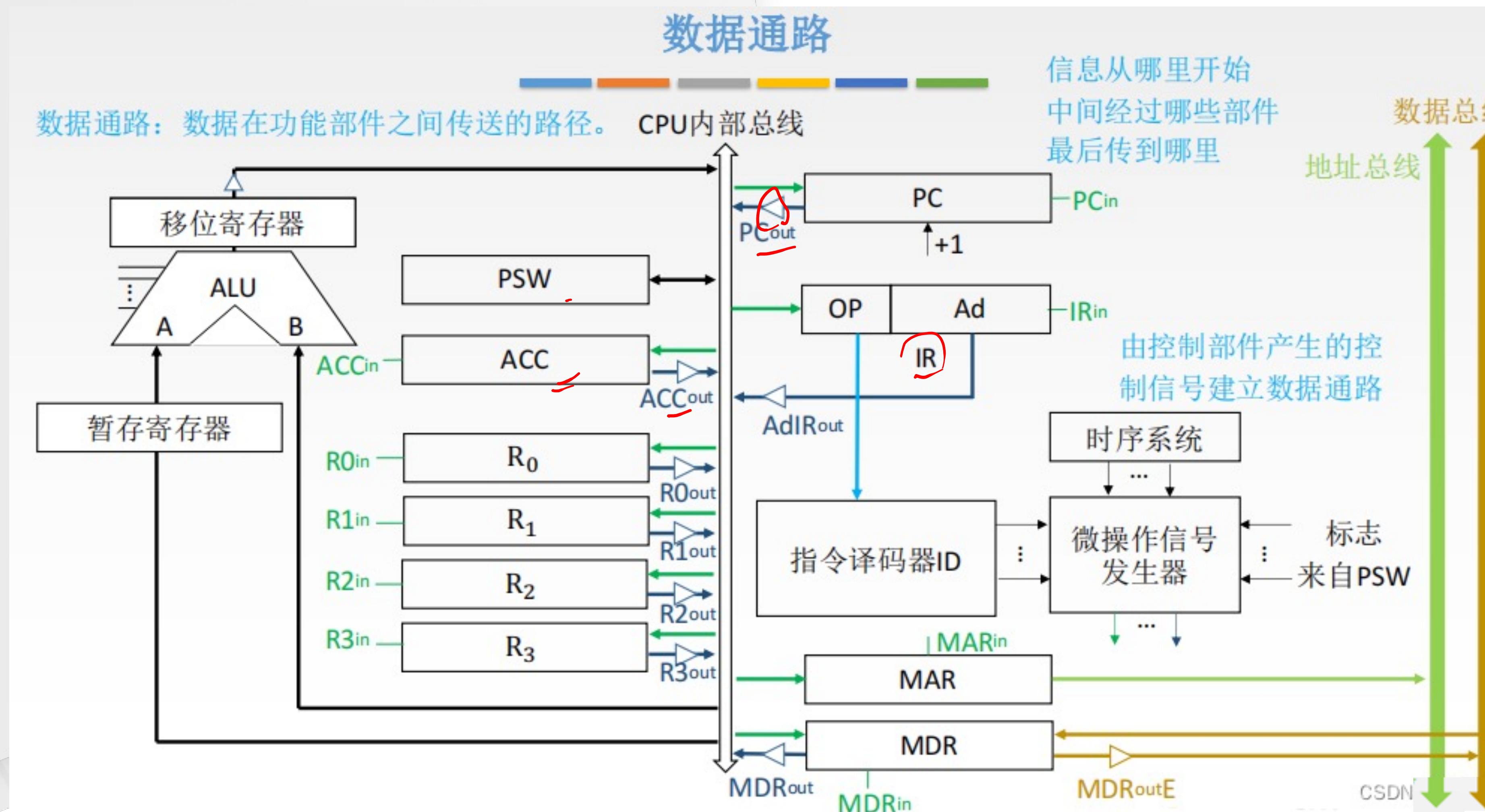
3.1.6 数据通路

1. CPU内部单总线方式
2. CPU内部多总线方式
3. 专用数据通路方式



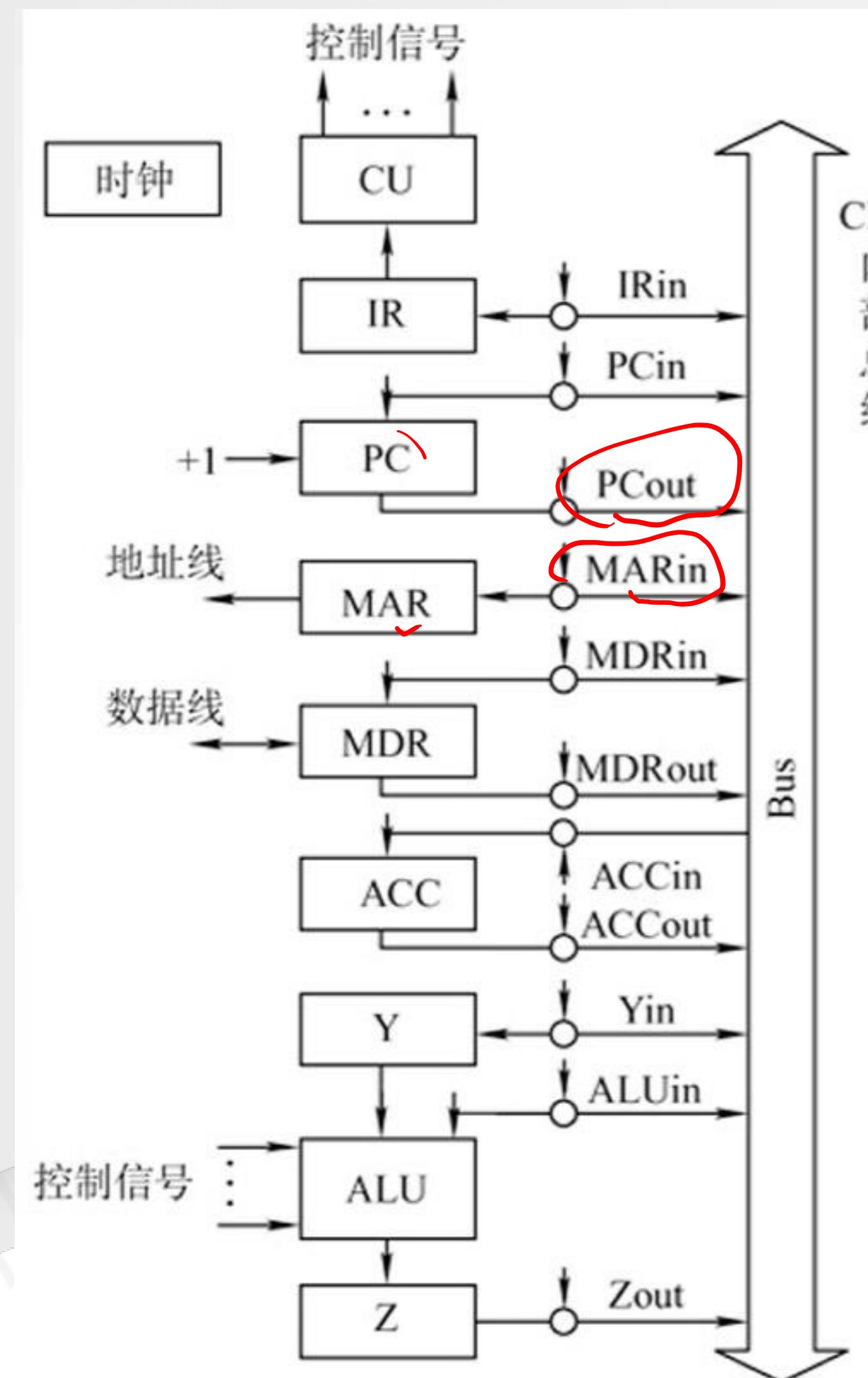
1. 内部总线：同一部件内各寄存器及运算部件之间的数据总线
2. 系统总线：同一台计算机系统的各部件互相连接的总线

3.1.6 CPU内部总线的数据通路与控制信号



3. 指令系统设计与CPU运行控制

3.1.6 CPU内部总线的数据通路与控制信号



寄存器之间数据传送

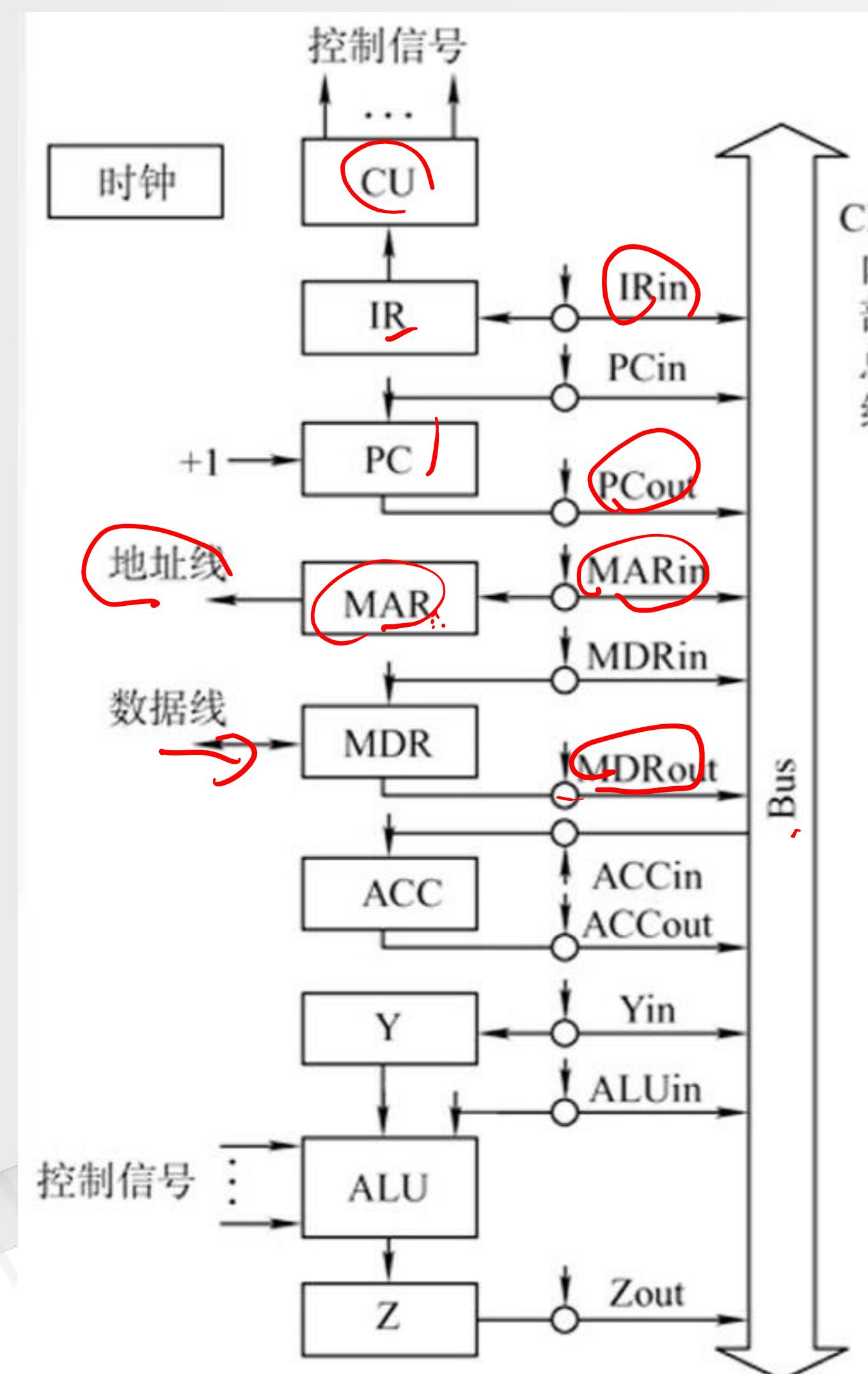
把 PC 内容送至 MAR, 实现传送数据的流程及控制信号为:

1. (PC) → Bus
2. Bus → MAR



3. 指令系统设计与CPU运行控制

3.1.6 CPU内部总线的数据通路与控制信号



主存和CPU之间数据传送

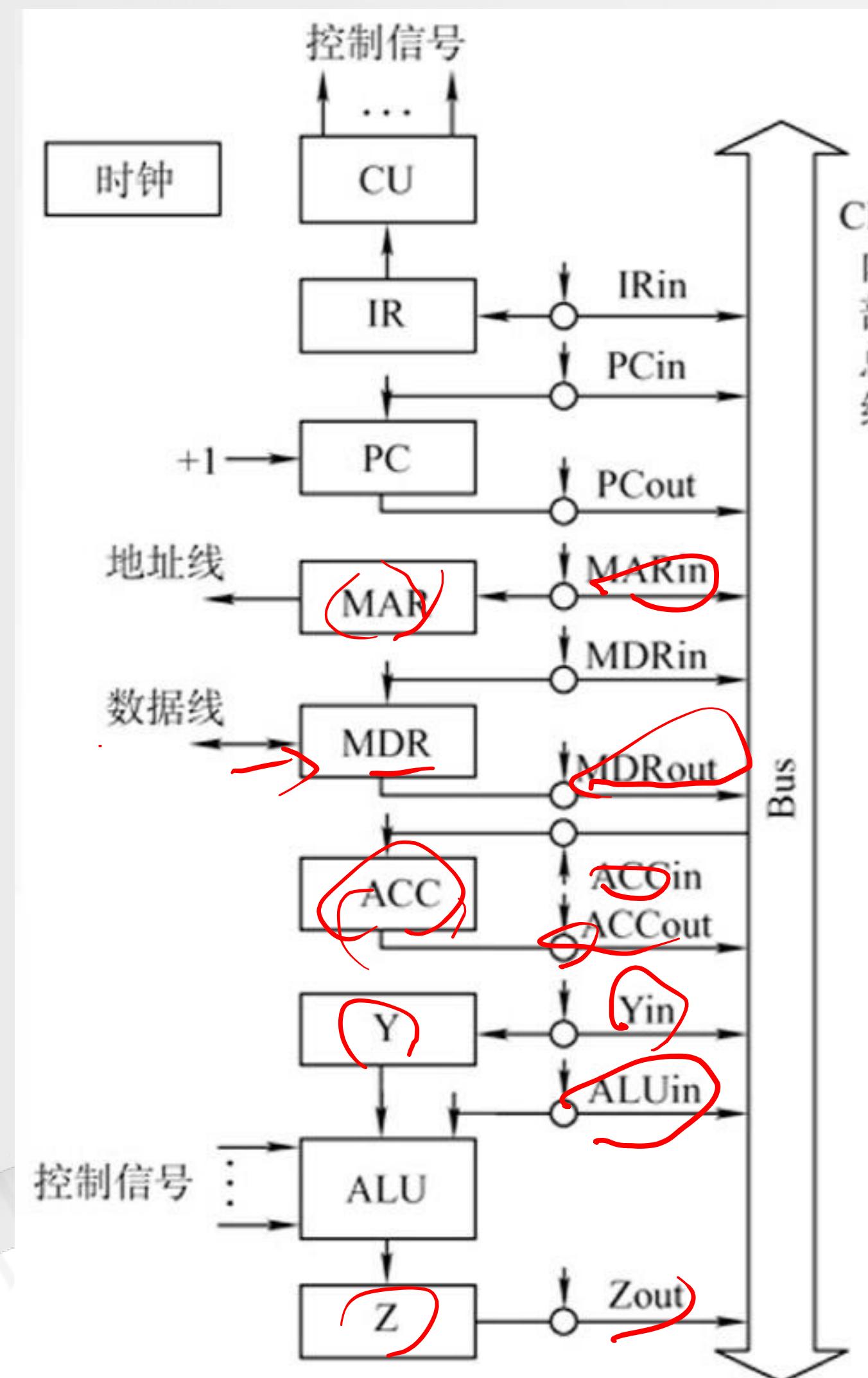
CPU从主存读取指令, 实现传送操作的流程

及控制信号为:

1. (PC) → Bus → MAR
2. 1 → R *CU 读*
3. M(MAR) → MDR
4. MDR → Bus → IR

3. 指令系统设计与CPU运行控制

3.1.6 CPU内部总线的数据通路与控制信号



算术或逻辑运算之间数据传送

一条加法指令,微操作序列及控制信号为:

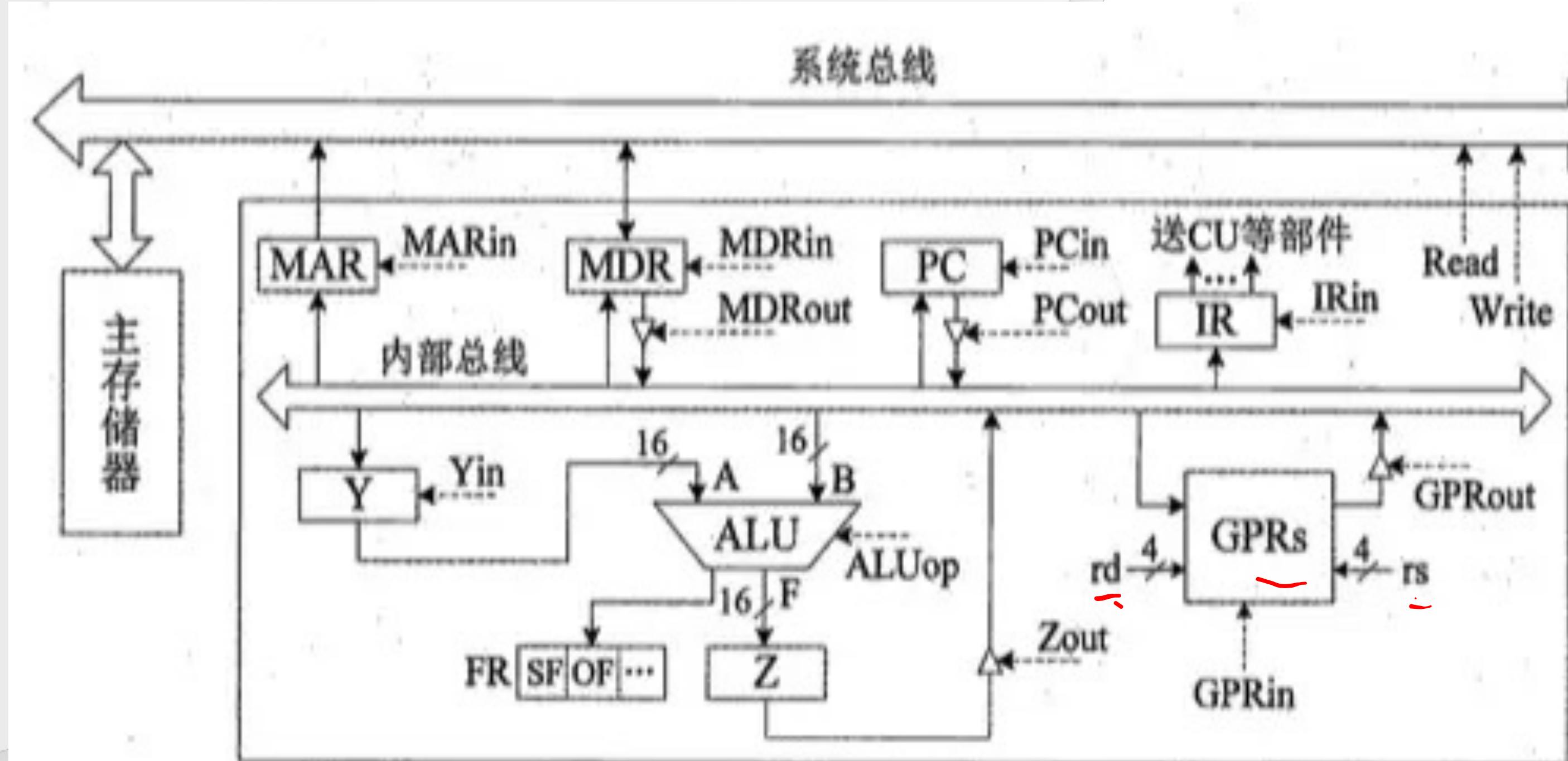
1. (MDR) → Bus → MAR
2. 1 → R
3. M(MAR) → BUS → MDR
4. MDR → Bus → Y
5. (ACC) + (Y) → Z
6. Z → ACC



3. 指令系统设计与CPU运行控制

3.1 练习

例5【2022真题】：某CPU中部分数据通路如图所示，其中，GPRs为通用寄存器组；FR为标志寄存器，用于存放ALU产生的标志信息；带箭头虚线表示控制信号，如控制信号Read、Write分别表示主存读、主存写，MDRin表示内部总线上数据写入MDR，MDRout表示MDR的内容送内部总线。





3. 指令系统设计与CPU运行控制



3.1 练习

- (1) ALU的输入端A、B及输出端F的最高位分别为 A15、B15 及 F15，FR中的符号标志和溢出标志分别为SF和OF，则SF的逻辑表达式是什么？A加B、A减B时OF的逻辑表达式分别是什么？要求逻辑表达式的输入变量为A15、B15 及 F15。
- (2) 为什么要设置暂存器Y和Z？
- (3) 若GPRs的输入端rs、rd分别为所读、写的通用寄存器的编号，则GPRs中最多有多少个通用寄存器？rs和rd来自图中的哪个寄存器？已知GPRs内部有一个地址译码器和一个多路选择器，rd应该连接地址译码器还是多路选择器？
- (4) 取指令阶段（不考虑PC增量操作）的控制信号序列是什么？若从发出主存读命令到主存读出数据并传送到MDR共需5个时钟周期，则取指令阶段至少需要几个时钟周期？
- (5) 图中控制信号由什么部件产生？图中哪些寄存器的输出信号会连到该部件的输入端？



3. 指令系统设计与CPU运行控制

3.1 练习

(1) ALU的输入端A、B及输出端F的最高位分别为 A₁₅、B₁₅ 及 F₁₅，FR中的符号标志和溢出标志分别为SF和OF，则SF的逻辑表达式是什么？A加B、A减B时OF的逻辑表达式分别是什么？要求逻辑表达式的输入变量为A₁₅、B₁₅ 及 F₁₅。

$$SF = F_{15}$$

加： $A_{15} + B_{15} = \bar{F}_{15}$
负： $\bar{A}_{15} + \bar{B}_{15} = \bar{F}_{15}$

$$OF = A_{15} \cdot B_{15} \cdot \bar{F}_{15} + \\ A_{15} \cdot \bar{B}_{15} \cdot \bar{F}_{15}$$

减

$$\text{正} - \text{负} = \text{负}, \quad \text{负} - \text{正} = \text{正}$$

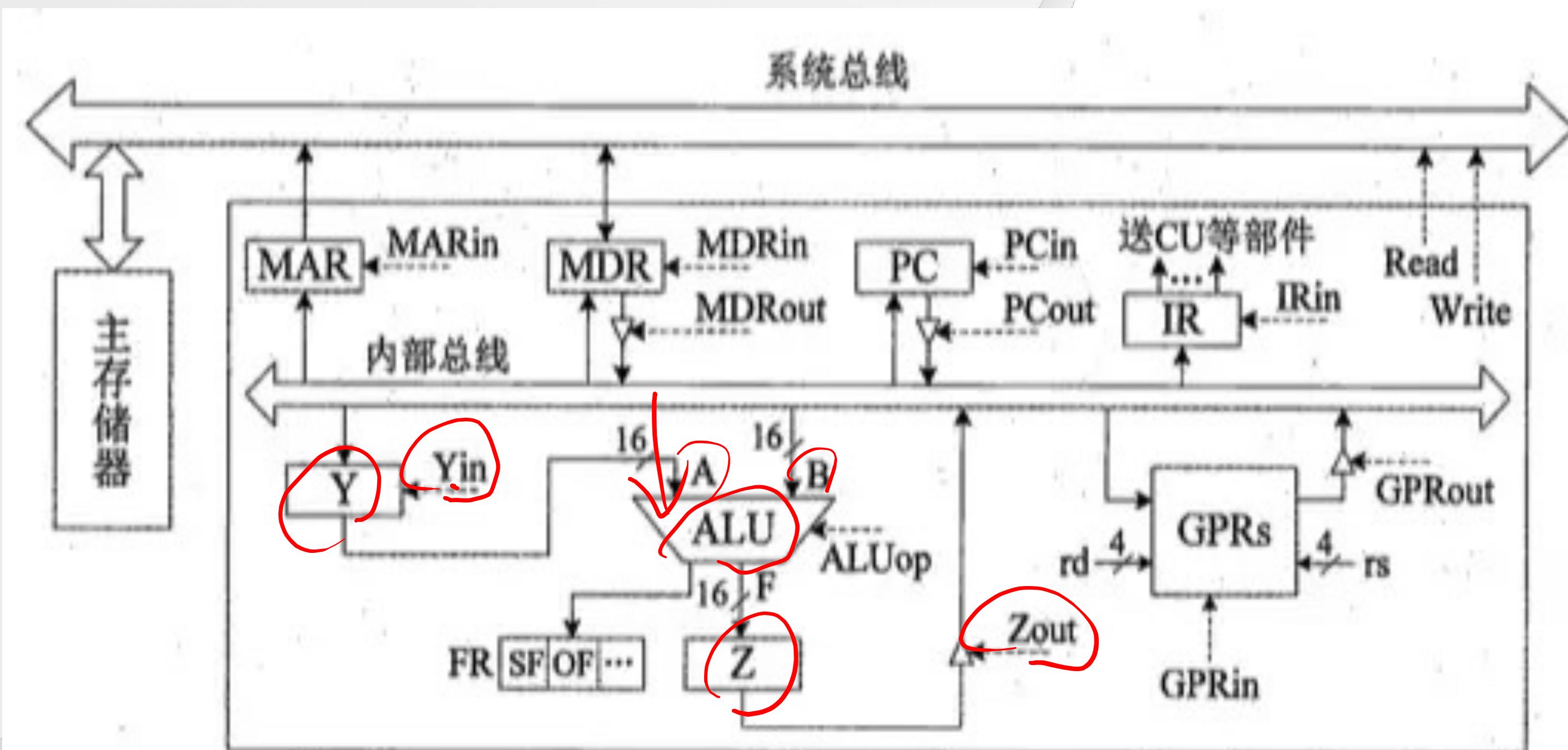
$$OF = \bar{A}_{15} \cdot B_{15} \cdot \bar{F}_{15} + A_{15} \cdot \bar{B}_{15} \cdot F_{15}$$



3. 指令系统设计与CPU运行控制

3.1 练习

(2) 为什么要设置暂存器Y和Z?

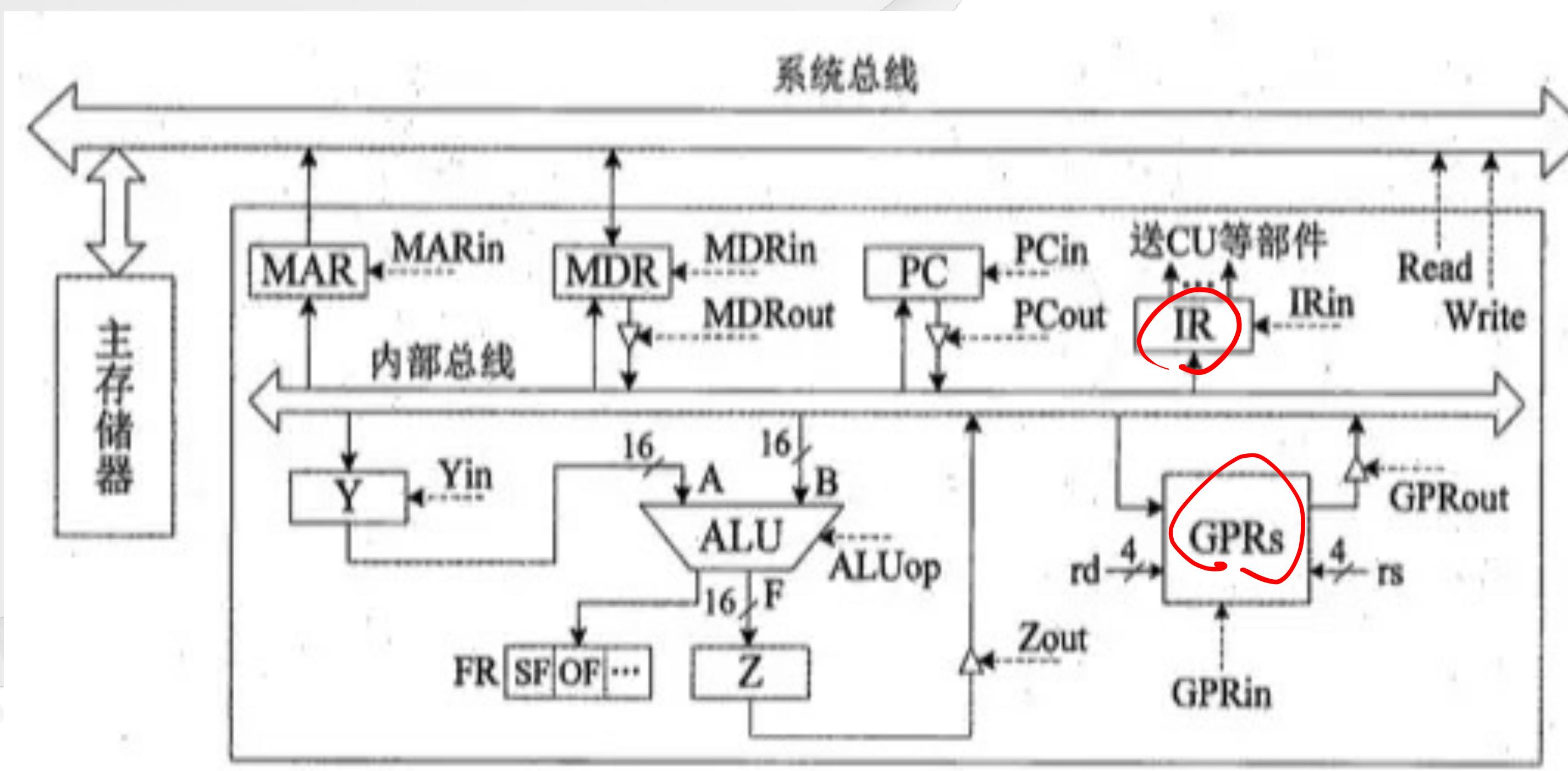




3. 指令系统设计与CPU运行控制

3.1 练习

(3) 若GPRs的输入端rs、rd分别为所读、写的通用寄存器的编号，则GPRs中最多有16个通用寄存器？rs和rd来自图中的哪个寄存器? 已知GPRs内部有一个地址译码器和一个多路选择器，rd应该连接地址译码器还是多路选择器？



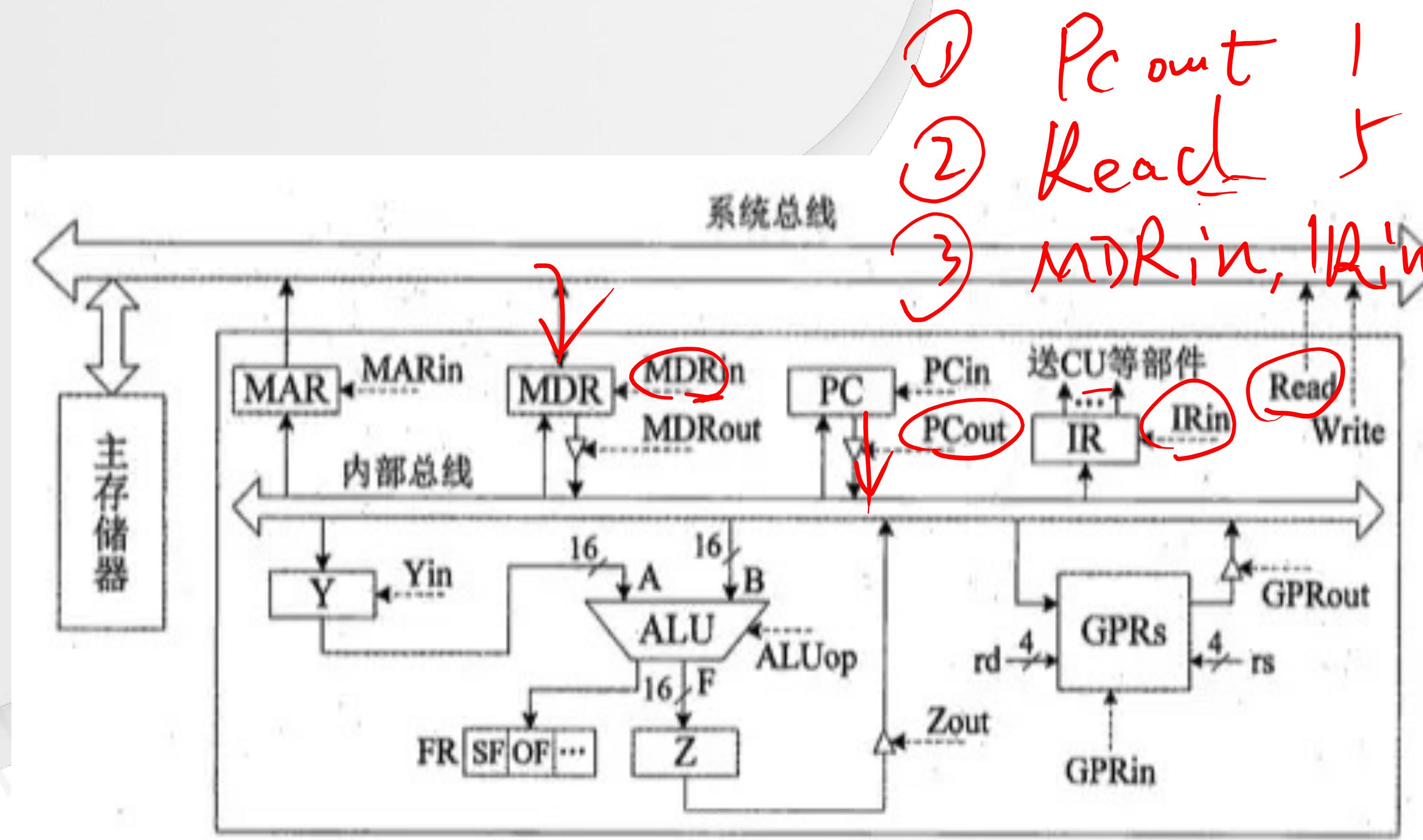
$$2^4 = 16$$



3. 指令系统设计与CPU运行控制

3.1 练习

(4) 取指令阶段（不考虑PC增量操作）的控制信号序列是什么？若从发出主存读命令到主存读出数据并传送到MDR共需5个时钟周期，则取指令阶段至少需要几个时钟周期？





3. 指令系统设计与CPU运行控制

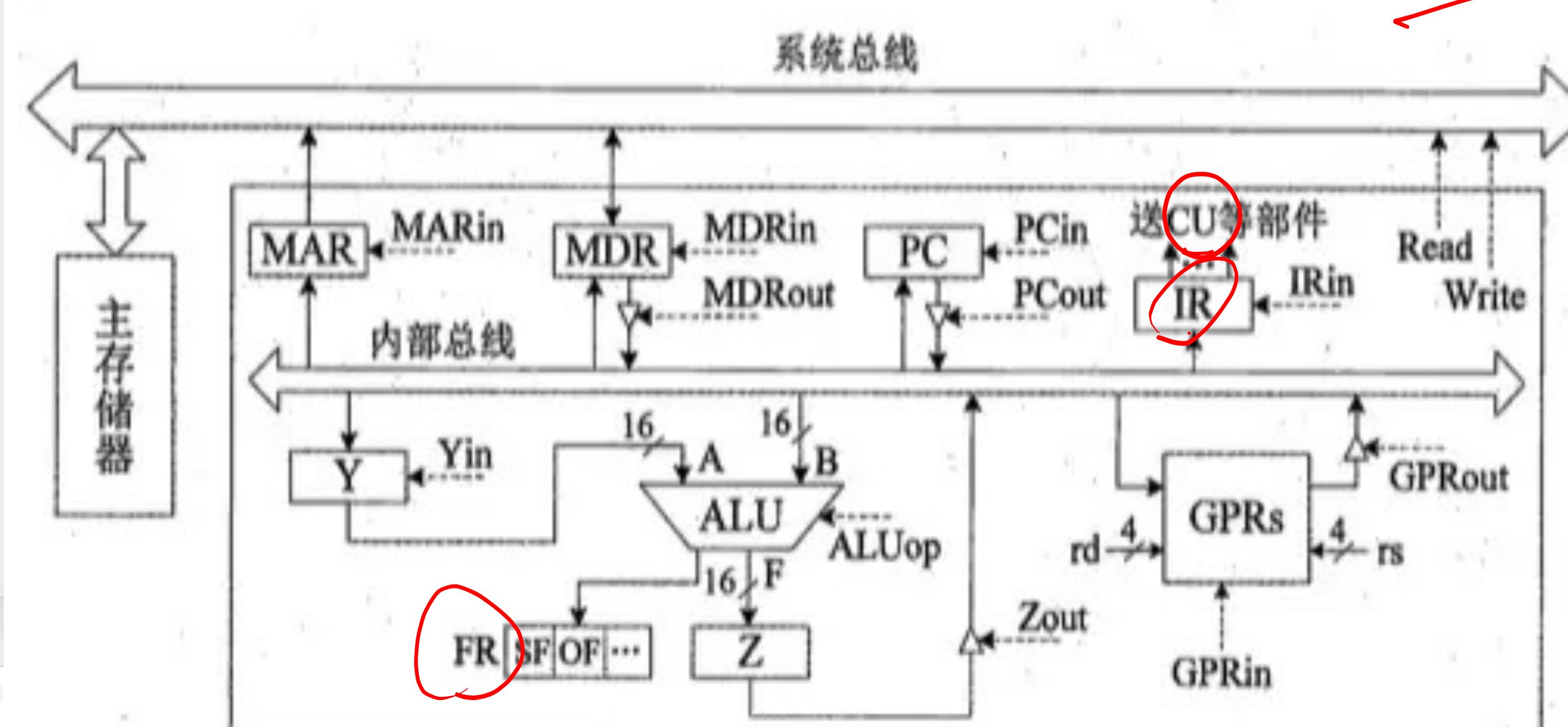
3.1 练习

(5) 图中控制信号由什么部件产生？图中哪些寄存器的输出信号会连到该部件的输入端？

CU

IR ,

FR, Z, C, N, O.





3. 指令系统设计与CPU运行控制

3.1 本节总结

1. 指令中包含让CPU完成任务的全部信息
2. 指令处理中的数据来源取决于数据寻址方式
3. 指令执行可能会分为取指，取数，运算，中断处理
4. 各阶段完成功能不一样，数据流也不一样

欢迎参与学习

WELCOME FOR YOUR JOINING

船说：计算机基础