计算机与网络技术

第3讲　指令系统（续）、逻辑电路模块

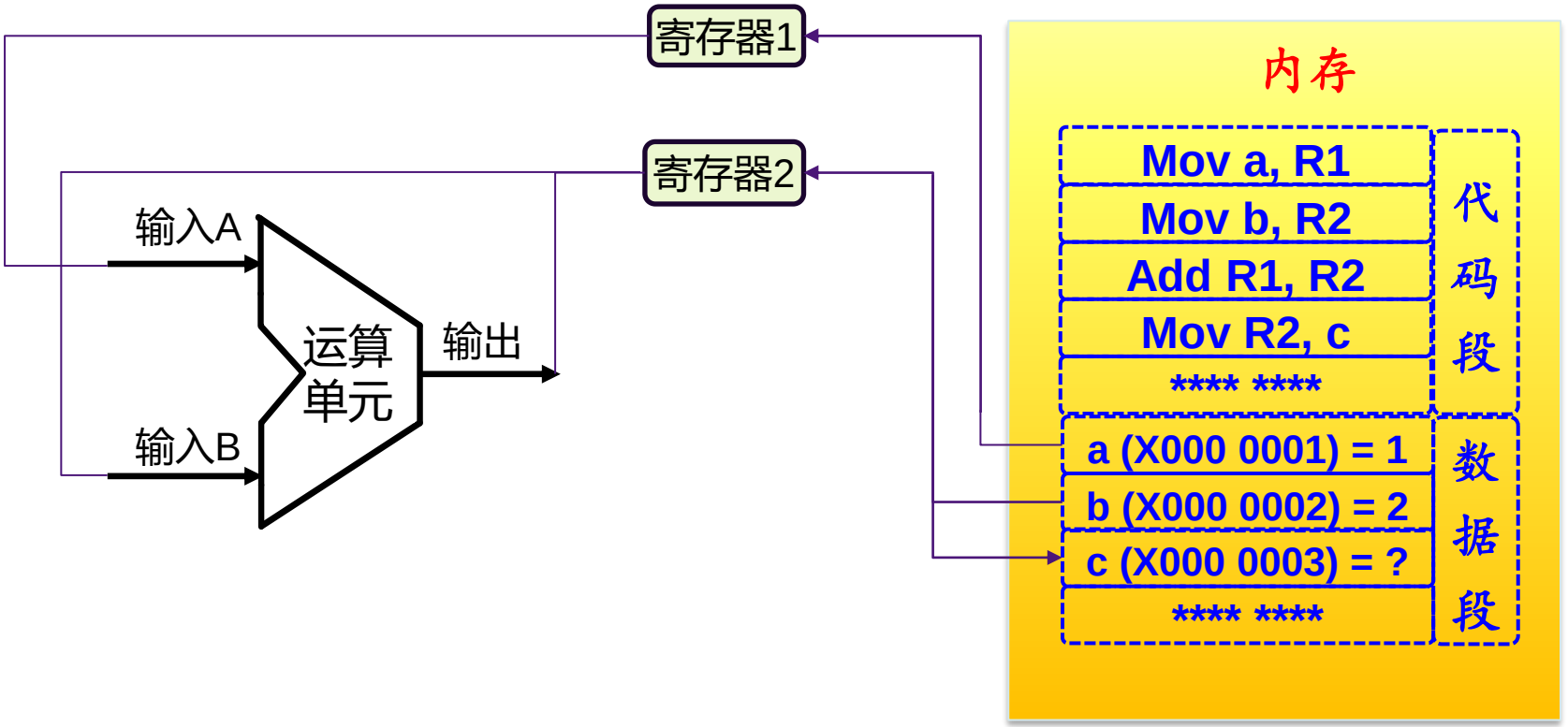
提纲

- **计算机指令系统（MIPS）**
  - 指令系统
  - 寻址方式

- **计算机逻辑电路模块**
  - 计算机微体系结构
  - 运算器（ALU）
  - 多路开关、译码器
  - 寄存器、指令指针（程序计数器）与取指单元
  - 存储器、I/O接口（微处理器外部）

计算机工作流程

例： “a+b=?”的运算



操作数放在哪里？ （操作数寻址方式）

- 立即数：在操作指令中直接给定
- 寄存器：操作数放在寄存器中
- 内存： 操作数放在存储单元中

指令系统

常用指令类型

序号	分类	基本指令
1	算术运算、关系运算	加、减、乘、除、比较、扩展
2	逻辑操作	与、或、非、异或、测试
3	移位指令	逻辑移位、算术移位、循环移位
4	数据传送	将存储器中的数据读到寄存器，将寄存器中的数据写入存储器
5	跳转、分支转移	条件转移、条件延迟、分支、子程序
6	系统控制	各种系统寄存器的存取、设置

寻址方式

指令格式

操作码　　操作数

1) 由操作码决定寻址方式

2)由寻址特征位（X）指定寻址方式

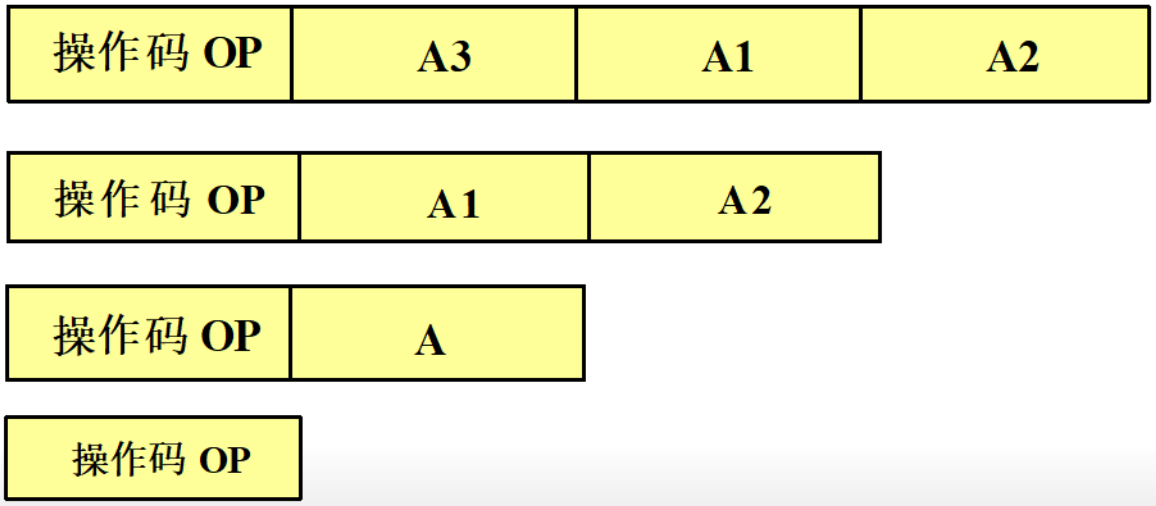
寻址方式

寻址方式（Addressing Mode）

根据物理地址去去指令和数据。如何获得指令和数据的物理地址？

寻找指令和操作数有效地址的方法

- 指令设计、编写：指明操作数来源的方式
- 指令执行：确定本条指令所需要的操作数的地址，确定下一条要执行指令地址



寻址方式

寻址方式（Addressing Mode）

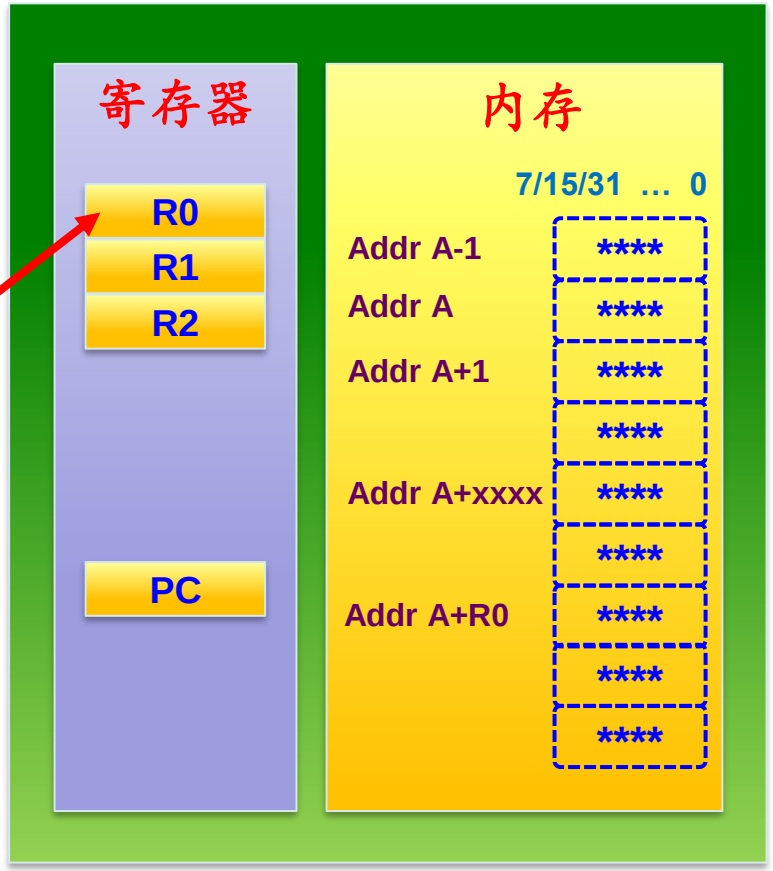
- 常见操作数寻址方式：
  - 立即数寻址（立即寻址）
  - 寄存器寻址
  - 直接寻址
  - 寄存器间接寻址、（存储器）间接寻址
  - 变址寻址
  - 隐含寻址（见本讲义附录）
- 指令的寻址方式：
  - 顺序寻址
  - 跳跃寻址

寻址方式

立即数寻址

- 指令中操作数字段就是操作数的数值本身
- 取来指令立即可得到操作数的数值，故称该操作数为立即数
- 指令执行速度快
- 通常用#等特殊符号作为立即数的前缀，用以和直接寻址进行区分
- 立即数寻址常用来给变量赋初值
- 数据大小受到操作数字段位数限制

- 示例：mov #Imm,R0

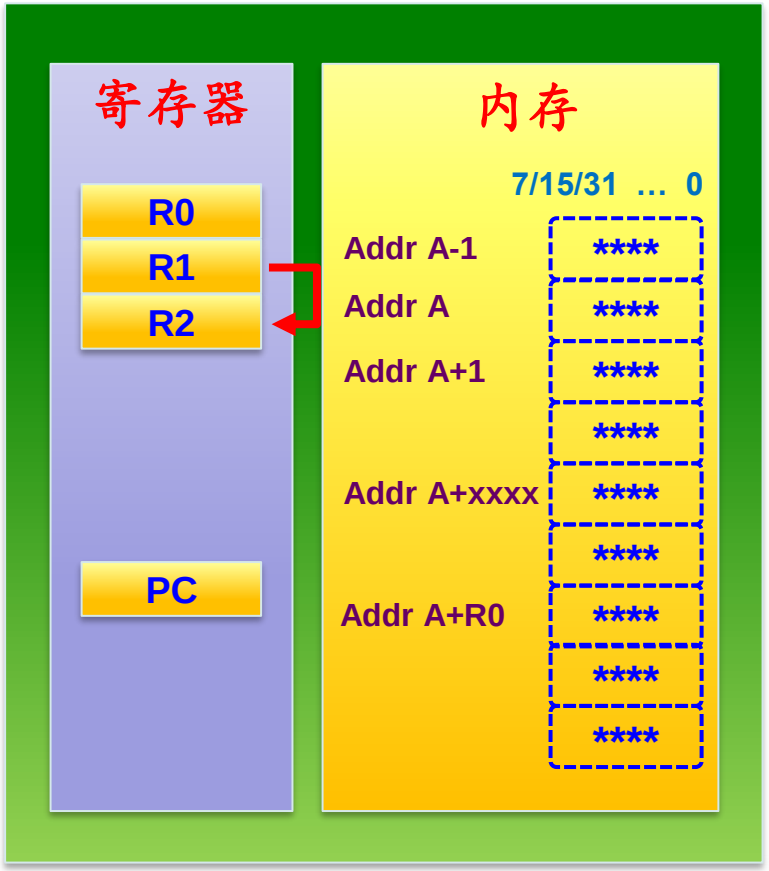


寻址方式

寄存器寻址

- 指令中的操作数给出了寄存器编号，该寄存器中保存了操作数的实际数据
- 寄存器读写比访问内存更快速，是RISC指令集最常用的寻址方式（例如OMAP4430芯片几乎所有指令使用这种方式，除了直接读取内存的LDR和STR指令外）
- 操作数字段位数限制的是通用寄存器的数量，并不限制数据的大小

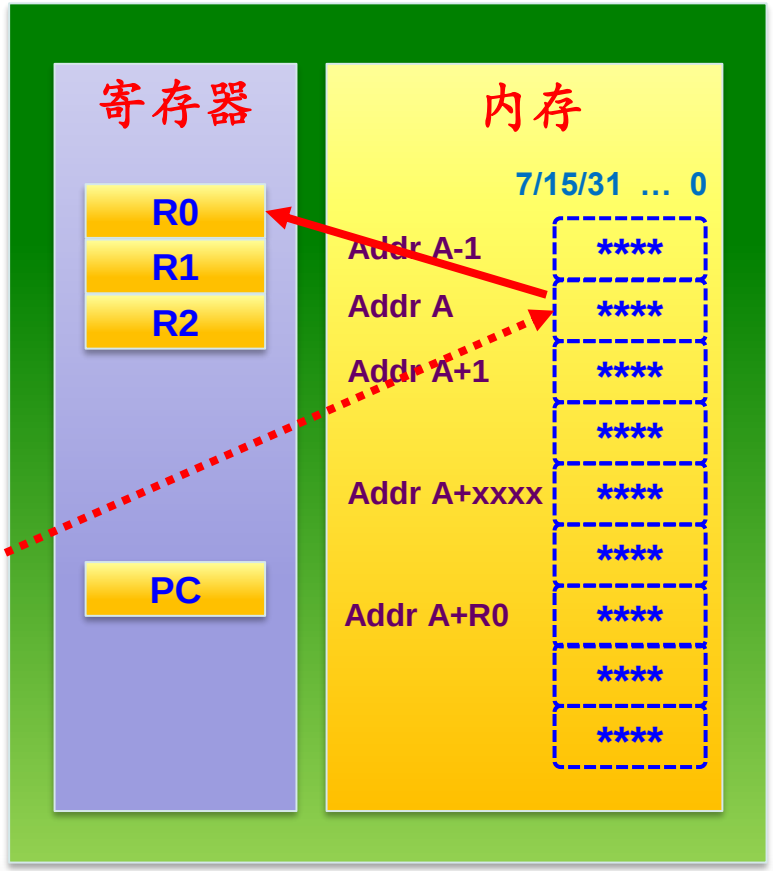
- 示例：mov R1, R2



寻址方式

直接寻址

- 提供访问主存的操作
- 由于要访问主存，指令的执行速度相对慢
- 指令中的操作数给出了实际数据所在的内存地址，根据该地址得到操作数的实际数值
- 直接寻址常用来访问全局变量、硬件外设寄存器（I/O端口等）
- 指令中操作数字段的位数，决定了可访问主存空间的大小



- 示例：mov &Addr A,R0

有什么区别？ mov #Addr A,R0

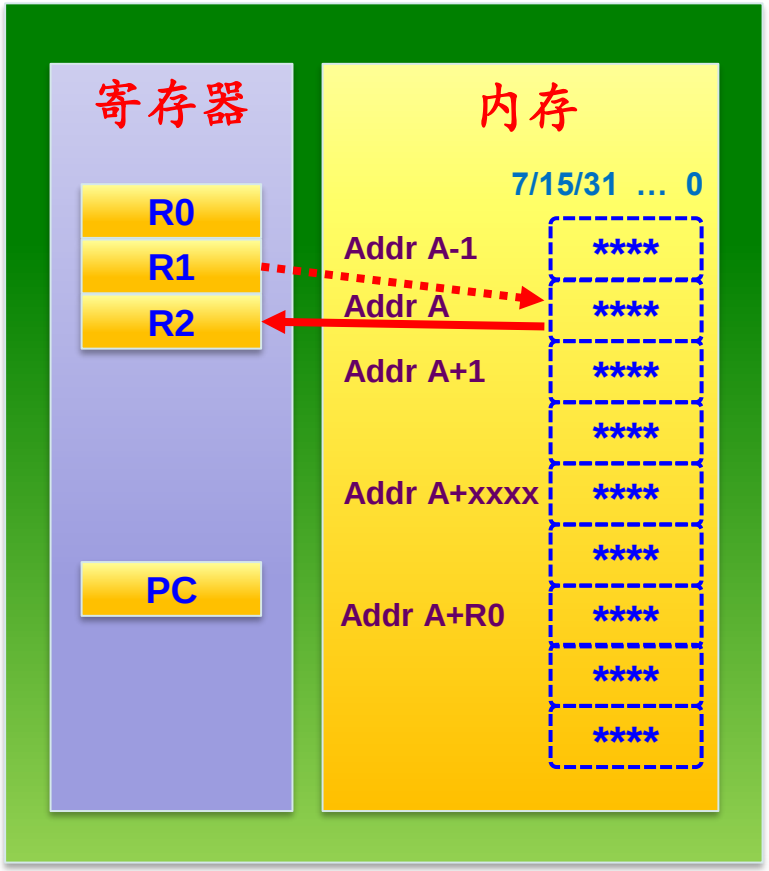
寻址方式

寄存器间接寻址

- 为了解决直接寻址访存范围受限的问题
- 指令中的操作数给出了寄存器编号，该寄存器中保存了实际数据所在的内存地址
- 通常用@、()、[ ]等特殊符号作为寄存器编号的前缀，用来和寄存器寻址进行区分

- 示例：**mov @(R1),R2**

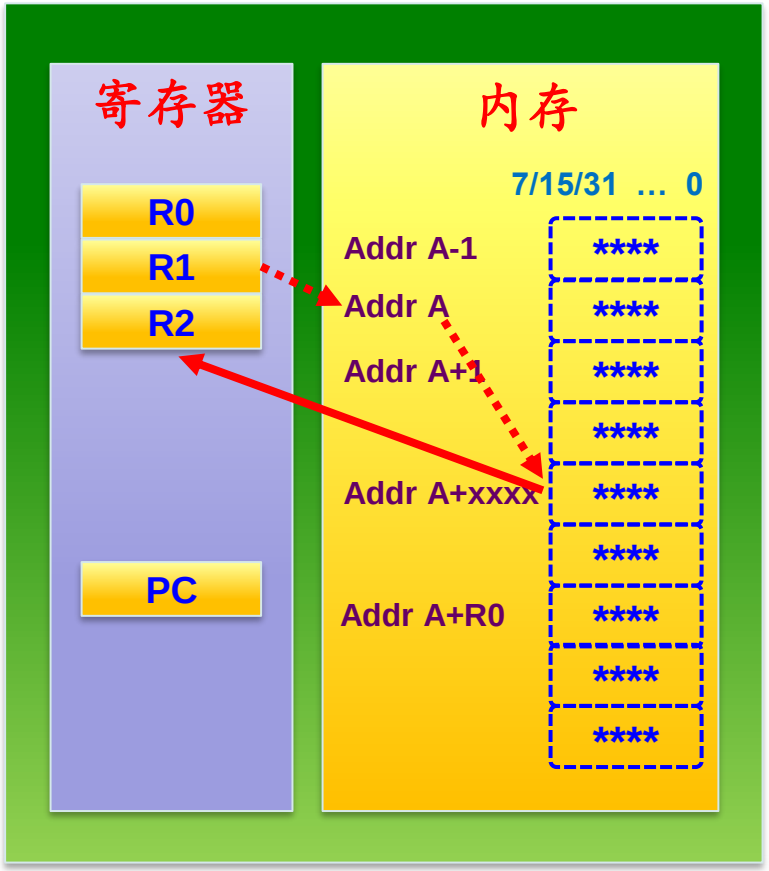
有什么区别？ **mov R1,R0**



寻址方式

变址寻址

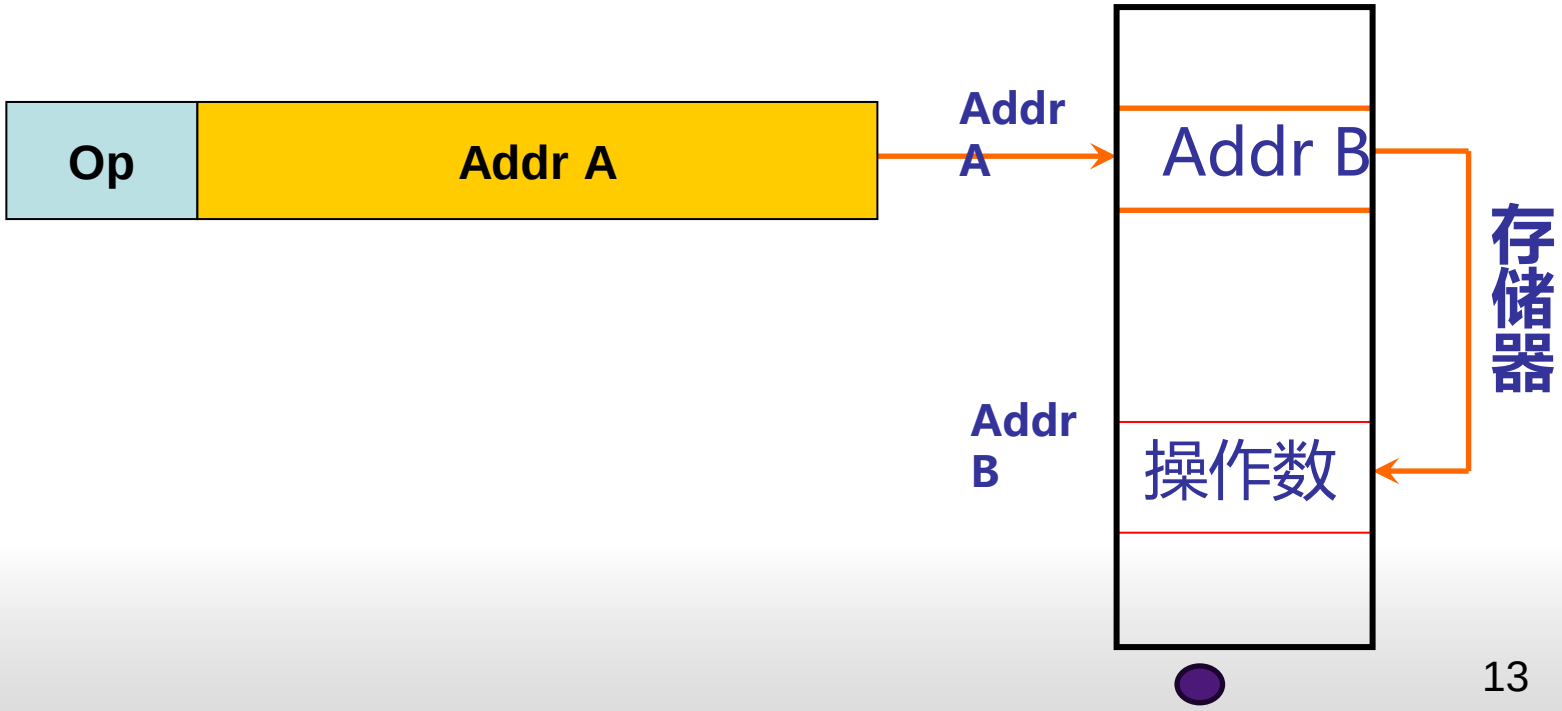
- 指令中的操作数给出了寄存器编号，以及地址偏移量的立即数，该寄存器中的数值与立即数相加的结果，才是实际数据所在的内存地址
- 变址寻址主要用来访问数组变量
- 示例：mov @(R1,xxxx),R2



寻址方式

存储器间接寻址（目前较少使用）

- 指令的操作数字段给出的内容既不是具体的操作数，也不是操作数的地址，而是操作数（或指令）地址的地址，这被称为存储器间接寻址方式，多一次读存储器的操作

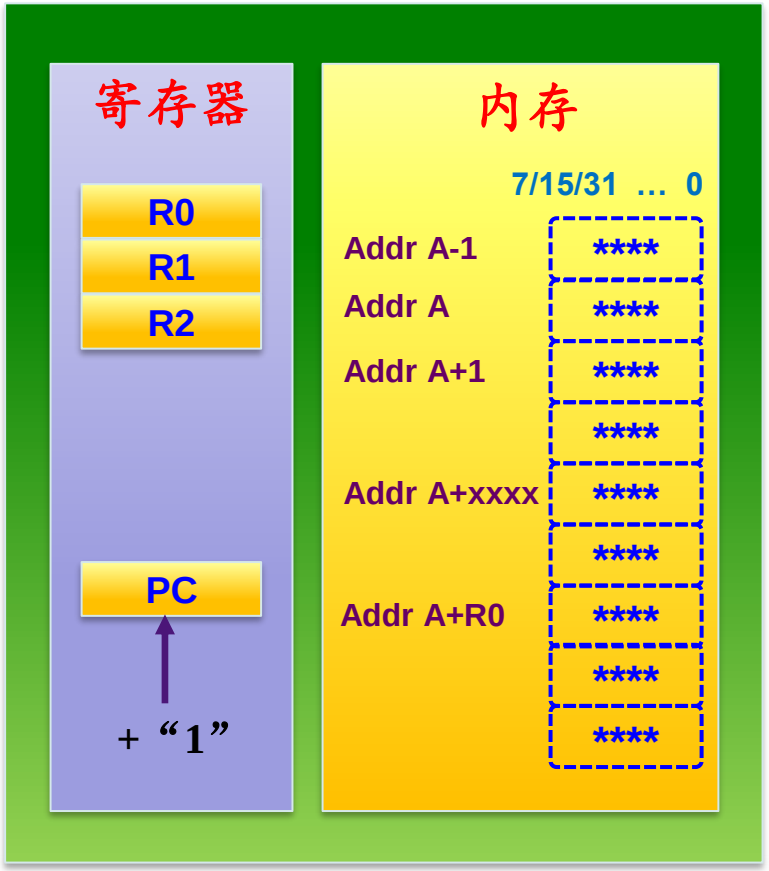


寻址方式

指令的顺序寻址

- 执行时从第一条指令开始，逐条取出并逐条执行
- CPU中设置程序计数器（PC）对指令的顺序号进行计数。PC开始时存放程序首地址，每执行一条指令，PC加“1”，指出下条指令的地址，直至程序结束

“1”：存储一条指令所占用的字节单元数目

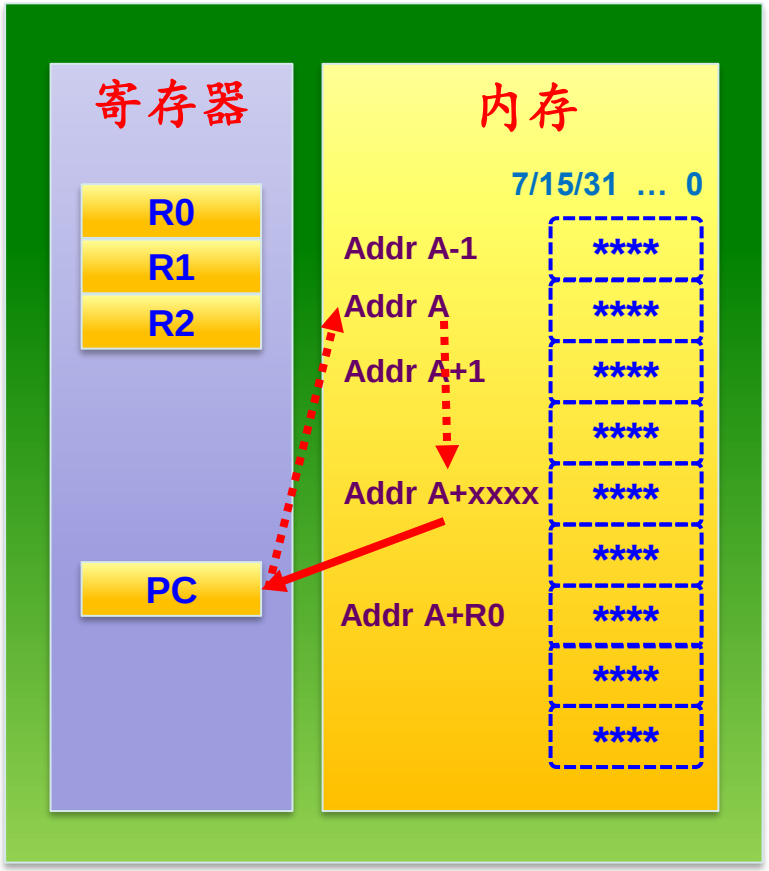


寻址方式

指令的跳跃寻址

相对寻址、基址寻址、基址+变址寻址

- 指令中的操作数给出了地址偏移量的立即数，PC或其他特定寄存器中的数值与立即数相加的结果，才是实际数据或者目标代码所在的内存地址
- PC相对寻址主要用于指令跳转



MIPS指令概览

MIPS（Microprocessor without interlocked piped stages）

- 是80年代初由斯坦福大学教授约翰·轩尼诗与他的团队创立；
- 属于精简指令集计算机RISC
- MIPS指令集有MIPS I, MIPS II, MIPS III, MIPS IV, MIPS V, MIPS32, MPIS64多个版本；
- 早期主要用于嵌入式系统，如Windows CE设备、路由器、家用网关和视频游戏机，现在已经在PC机、服务器中得到广泛应用
- 2007年8月16日，“龙芯”获得MIPS处理器IP的全部专利和总线、指令集授权

MIPS指令系统（32位）

MIPS基本指令

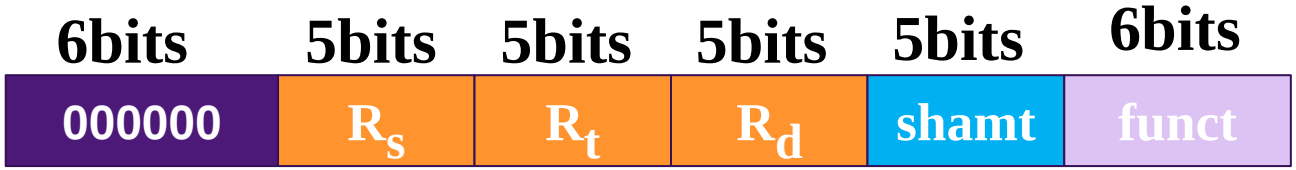
序号	分类	基本指令
1	算术运算	add, sub, addi, addu, mul, mulu, div, divu
2	逻辑、移位操作	and, or, andi, ori, sll, srl, lui
3	数据传送	lw, sw, lb, lbu, sb
4	分支转移	beq, bne, bnez, slt, slti, sltu,sltiu
5	无条件跳转	j, jr, jal

指令后缀： i 表示立即数；u 表示无符号；w 表示字(Word)操作；b 表示字节(Byte)操作

MIPS指令系统（32位）

- 只有三种指令格式

R型指令



寄存器型

- Rs， Rt 分别为第一、二源操作数；Rd为目标操作数

I型指令

立即数型

- 双目、Load/Store: Rs和立即数是源操作数， Rt为目标操作数
- 条件转移： Rs, Rt均为源操作数

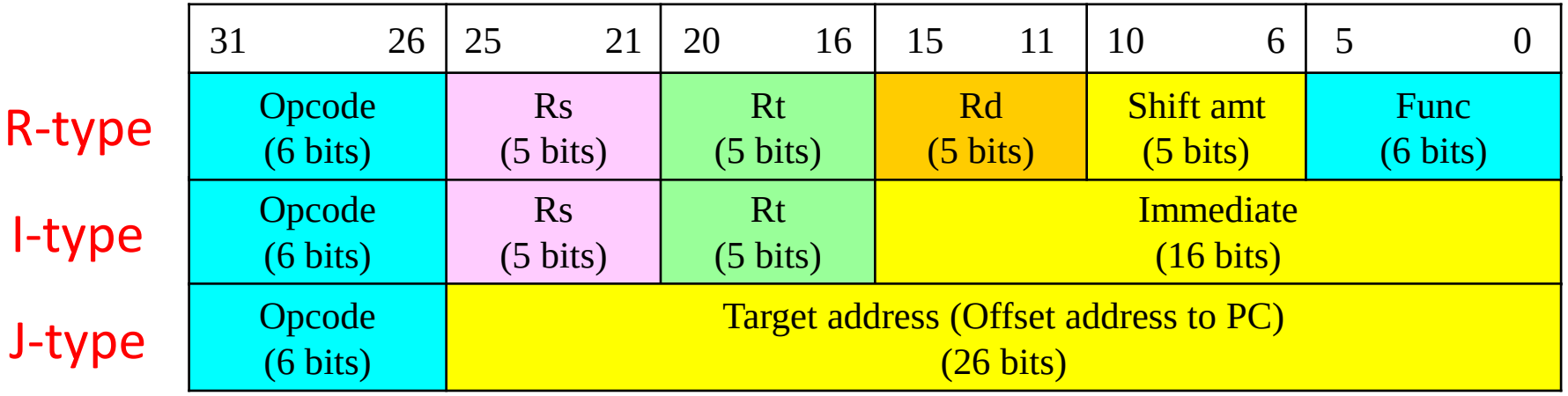
J型指令

跳转

- 26位立即数作为跳转目标地址的部分地址

MIPS指令系统（32位）

32位MIPS系统指令格式



- Opcode: partially specifies what instruction it is; **Equal to 0 for all R-Format instructions**
- Funct: combined with opcode, this number exactly specifies the instruction
- Rs (Source Register): generally used to specify register containing first operand
- Rt (Target Register): generally used to specify register containing second operand
- Rd (Destination Register): generally used to specify register which will receive result of Opcode
- Shamt: This field contains the amount a shift instruction will shift by.
- Immediate: address offset or immediate value
- Target address (Offset address to PC): target address of the jump instruction

MIPS指令系统（32位）

32位MIPS系统指令特点

1. 指令定长（32 bits）
   - 操作码长度固定（6位），在指令格式中位置固定
   - 寄存器编码长度固定（5位），可寻址32个寄存器，在指令格式中位置固定
   - 立即数长度固定（16位），在指令格式中位置固定
2. 由操作码决定寻址方式
3. 运算操作主要基于寄存器/立即数

MIPS指令系统（32位）

MIPS系统寄存器

寄存器名称	编码	用途
$zero	0	Constant 0
$at	1	Reserved for assembler
$v0 ~ $v1	2~3	Expression evaluation and results of a function
$a0 ~ $a3	4~7	Arguments 13
$t0 ~ $t7	8~15	Temporary (not preserved across call)
$s0 ~ $s7	16~23	Saved temporary (preserved across call)
$t8 ~ $t9	24~25	Temporary (not preserved across call)
$k0 ~ $k1	26~27	Reserved for OS kernel
$gp	28	Pointer to global area
$sp	29	Stack pointer
$fp	30	Frame pointer
$ra	31	Return address (used by function call)

MIPS指令系统（32位）

MIPS基本指令

- R-type指令

31 26	25 21	20 16	15 11	10 6	5 0
Opcode (6 bits)	Rs (5 bits)	Rt (5 bits)	Rd (5 bits)	Shift amt (5 bits)	Func (6 bits)
0	18	19	17	0	32
0000 00	10 010	1 0011	1000 1	000 00	10 0000
add $s1, $s2, $s3 # $s1=$s2+$s3, R-type 寄存器寻址					

31 26	25 21	20 16	15 11	10 6	5 0
Opcode (6 bits)	Rs (5 bits)	Rt (5 bits)	Rd (5 bits)	Shift amt (5 bits)	Func (6 bits)
0	18	19	17	0	34
0000 00	10 010	1 0011	1000 1	000 00	10 0010
sub $s1, $s2, $s3 # $s1=$s2-$s3, R-type 寄存器寻址					

MIPS指令系统（32位）

MIPS基本指令

- R-type指令

31 26	25 21	20 16	15 11	10 6	5 0
Opcode (6 bits)	Rs (5 bits)	Rt (5 bits)	Rd (5 bits)	Shift amt (5 bits)	Func (6 bits)
0	18	19	17	0	36
0000 00	10 010	1 0011	1000 1	000 00	10 0100
and $s1, $s2, $s3	# $s1=$s2 AND $s3, R-type				寄存器寻址

31 26	25 21	20 16	15 11	10 6	5 0
Opcode (6 bits)	Rs (5 bits)	Rt (5 bits)	Rd (5 bits)	Shift amt (5 bits)	Func (6 bits)
0	18	19	17	0	37
0000 00	10 010	1 0011	1000 1	000 00	10 0101
or $s1, $s2, $s3	# $s1=$s2 OR $s3, R-type				寄存器寻址

MIPS指令系统（32位）

MIPS基本指令

- R-type指令

31 26	25 21	20 16	15 11	10 6	5 0
Opcode (6 bits)	Rs (5 bits)	Rt (5 bits)	Rd (5 bits)	Shift amt (5 bits)	Func (6 bits)
0	18	19	17	0	42
0000 00	10 010	1 0011	1000 1	000 00	10 1010
slt $s1, $s2, $s3 # set if less than $s2<$s3, $s1=1; else $s1=0, R-type 寄存器寻址					

MIPS指令系统（32位）

MIPS基本指令

- R-type指令

31 26	25 21	20 16	15 11	10 6	5 0
Opcode (6 bits)	Rs (5 bits)	Rt (5 bits)	Rd (5 bits)	Shift amt (5 bits)	Func (6 bits)
0	0	18	17	10	2
0000 00	00 000	1 0010	1000 1	010 10	00 0010
srl $s1, $s2, 10	#$s1=shift($s2) right logical 10 bits, R-type 寄存器寻址				

0	0	18	17	10	0
0000 00	00 000	1 0010	1000 1	010 10	00 0000
sll $s1, $s2, 10	# $s1=shift($s2) left logical 10 bits, R-type 寄存器寻址				

MIPS指令系统（32位）

MIPS基本指令

- R-type 指令

0	31	0	0	0	8
0000 00	11 111	0 0000	0000 0	000 00	00 1000
jr $ra # jump register $ra, R-type					寄存器寻址

寄存器提供要转移的目标地址

MIPS指令系统（32位）

MIPS基本指令

■ I-type指令

31 26	25 21	20 16	15 11	10 6	5 0
Opcode (6 bits)	Rs (5 bits)	Rt (5 bits)	Address/Immediate (16 bits)		
8	18	17	100		
0010 00	10 010	1 0001	0000 0000 0110 0100		
addi $s1, $s2, 100 # $s1=$s2+100, I-type					

12	18	17	100
0011 00	10 010	1 0001	0000 0000 0110 0100
andi $s1, $s2, 100 # $s1=$s2 AND 100, I-type			

13	18	17	100
0011 01	10 010	1 0001	0000 0000 0110 0100
ori $s1, $s2, 100 # $s1=$s2 OR 100, I-type			

寄存器寻址，立即数寻址

MIPS指令系统（32位）

MIPS基本指令

- I-type 指令

31 26	25 21	20 16	15 11	10 6	5 0
Opcode (6 bits)	Rs (5 bits)	Rt (5 bits)	Address/Immediate (16 bits)		
35	18	17	100		
1000 11	10 010	1 0001	0000 0000 0110 0100		
lw $1, 100($2) # load memory word $1=Memory[$2+100], I-type 变址寻址					

43	18	17	100
1010 11	10 010	1 0001	0000 0000 0110 0100
sw $s1, 100($s2) # save memory word $Memory[$s2+100]=$s1, I-type 变址寻址			

MIPS指令系统（32位）

MIPS基本指令　I-type指令

R：通用寄存器
PC：指令指针寄存器

31 26	25 21	20 16	15 11	10 6	5 0
Opcode (6 bits)	Rs (5 bits)	Rt (5 bits)	Address/Immediate (16 bits)		
15	0	17	100		
0011 11	00 000	1 0001	0000 0000 0110 0100		
lui $s1, 100 # load & shift left unsigned immediate $s1=100«16, I-type					

寄存器寻址
立即数寻址

MIPS指令系统（32位）

MIPS基本指令

I-type指令

R：通用寄存器
PC：指令指针寄存器

4	17	18	25
0001 00	10 001	1 0010	0000 0000 0001 1001
beq $s1, $s2, 25 # branch if $s1=$s2, goto PC+4+25*4, I-type			

寄存器寻址
相对寻址

5	17	18	25
0001 01	10 001	1 0010	0000 0000 0001 1001
bne $s1, $s2, 25 # branch if $s1≠$s2, goto PC+4+25*4, I-type			

MIPS指令系统（32位）

MIPS基本指令

New PC = { PC[31..28], target address, 00 }
= { 1010, 00000000000000100111000100, 00 }
32 bit address = { 4 bits , 26 bits , 2 bits }

- J-type 指令

31 26	25 21	20 16	15 11	10 6	5 0
Opcode (6 bits)	Target address (26 bits)				
2	2500				
0000 10	00 0000 0000 0000 1001 1100 0100				
j 2500 # jump to 2500, I-type					

相对寻址?
直接寻址?

无条件转移指令 $\$PC \leftarrow (\$PC+4)_{H4} \cup (\text{Target address} << 2)$

PC+4后的高4位 26位目标地址左移2位

指令系统

指令系统设计实验

- 指令字长：16位
- 数据字长：16位
- 通用寄存器数量：4个
- 寻址空间：256字节
- 功能：15个

功能			操作码
算术运算	不带进位加减	不带进位加	0010
		不带借位减	0011
		不带进位立即数加	1010
	带进位加减	带进位加	0110
		带借位减	0101
关系运算	无符号数比较	无符号数比较	0100
逻辑运算	寄存器之间的与或运算	与	0000
		或	0001
	寄存器与立即数与或运算	立即数与	1000
		立即数或	1001
数据传输	将存储器中的数据读到寄存器	读存储器	1011
	将寄存器中的数据写入存储器	写存储器	1100
跳转	有条件跳转（根据比较结果）	相等时跳转	1101
		不等时跳转	1110
	无条件跳转	无条件跳转	0111

指令系统

指令系统设计实验

R：通用寄存器
PC：指令指针寄存器

	操作名称	操作码	汇编语言格式指令	执行操作
R型指令	与	0000	AND Rd, Rs, Rt	Rd←Rs and Rt; PC ← PC + 1
	或	0001	OR Rd, Rs, Rt	Rd←Rs or Rt; PC ← PC +1
	不带进位加	0010	ADD Rd, Rs, Rt	Rd←Rs+ Rt; PC ← PC + 1
	不带借位减	0011	SUB Rd, Rs, Rt	Rd←Rs– Rt; PC ← PC + 1
	带进位加	0110	ADDC Rd, Rs, Rt	Rd←Rs+ Rt+C; PC ← PC + 1
	带借位减	0101	SUBC Rd, Rs, Rt	Rd←Rs– Rt-(1-C); PC ← PC + 1
	无符号数比较	0100	SLT Rd,Rs,Rt	If Rs<Rt, Rd=1 else Rd=0; PC ← PC + 1
I型指令	立即数与	1000	ANDI Rt, Rs, imm	Rt←Rs and imm; PC ← PC +1
	立即数或	1001	ORI Rt, Rs, imm	Rt←Rs or imm; PC ←PC +1
	立即数加	1010	ADDI Rt, Rs, imm	Rt←Rs+ imm; PC ← PC +1
	读存储器	1011	LW Rt, Rs, imm	Rt←MEM[Rs+imm]; PC ←PC +1
	写存储器	1100	SW Rt, Rs, imm	MEM[Rs+imm] ←Rt; PC ← PC +1
	相等时跳转	1101	BEQ Rs, Rt, imm	If Rt=Rs, PC←PC+imm+1 else PC←PC+1
	不等时跳转	1110	BNE Rs, Rt, imm	If Rt!=Rs, PC←PC+imm+1 else PC←PC+1
J型指令	无条件跳转	0111	JMP imm	PC ← imm

指令系统设计实验

根据功能与性能要求，仿照MIPS指令格式进行设计

R型指令

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Op				Rs		Rt		Rd		---					

I型指令

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Op				Rs		Rt		Imm							

J型指令

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Op				----				Imm							

Op：指令操作码
Rs：第1个操作数的寄存器号
Rt：第2个操作数的寄存器号
Imm：立即数
Rd：第3个操作数的寄存器号
---：表示任意值，编码时通常取0

指令系统

指令系统设计实验

1. 指令定长（16 bits）
   - 操作码长度固定（4位），在指令格式中位置固定
   - 寄存器编码长度固定（2位），可寻址4个寄存器，在指令格式中位置固定
   - 立即数长度固定（8位），在指令格式中位置固定
2. 由操作码决定寻址方式
3. 运算操作主要基于寄存器/立即数

指令系统设计实验

指令对应寻址方式

R：通用寄存器
PC：指令指针寄存器

操作名称	操作码	汇编语言格式指令	执行操作	
与	0000	AND Rd, Rs, Rt	Rd←Rs and Rt; PC ← PC + 1	寄存器寻址
或	0001	OR Rd, Rs, Rt	Rd←Rs or Rt; PC ← PC +1	
不带进位加	0010	ADD Rd, Rs, Rt	Rd←Rs+ Rt; PC ← PC + 1	
不带借位减	0011	SUB Rd, Rs, Rt	Rd←Rs– Rt; PC ← PC + 1	
带进位加	0110	ADDC Rd, Rs, Rt	Rd←Rs+ Rt+C; PC ← PC + 1	
带借位减	0101	SUBC Rd, Rs, Rt	Rd←Rs– Rt-(1-C); PC ← PC + 1	
无符号数比较	0100	SLT Rd,Rs,Rt	If Rs<Rt, Rd=1 else Rd=0; PC ← PC + 1	
立即数与	1000	ANDI Rt, Rs, imm	Rt←Rs and imm; PC ← PC +1	立即数寻址
立即数或	1001	ORI Rt, Rs, imm	Rt←Rs or imm; PC ←PC +1	
立即数加	1010	ADDI Rt, Rs, imm	Rt←Rs+ imm; PC ← PC +1	
读存储器	1011	LW Rt, Rs, imm	Rt←MEM[Rs+imm]; PC ←PC +1	变址寻址
写存储器	1100	SW Rt, Rs, imm	MEM[Rs+imm] ←Rt； PC ← PC +1	
相等时跳转	1101	BEQ Rs, Rt, imm	If Rt=Rs, PC←PC+imm+1 else PC←PC+1	相对寻址
不等时跳转	1110	BNE Rs, Rt, imm	If Rt!=Rs, PC←PC+imm+1 else PC←PC+1	
无条件跳转	0111	JMP imm	PC ← imm	直接寻址

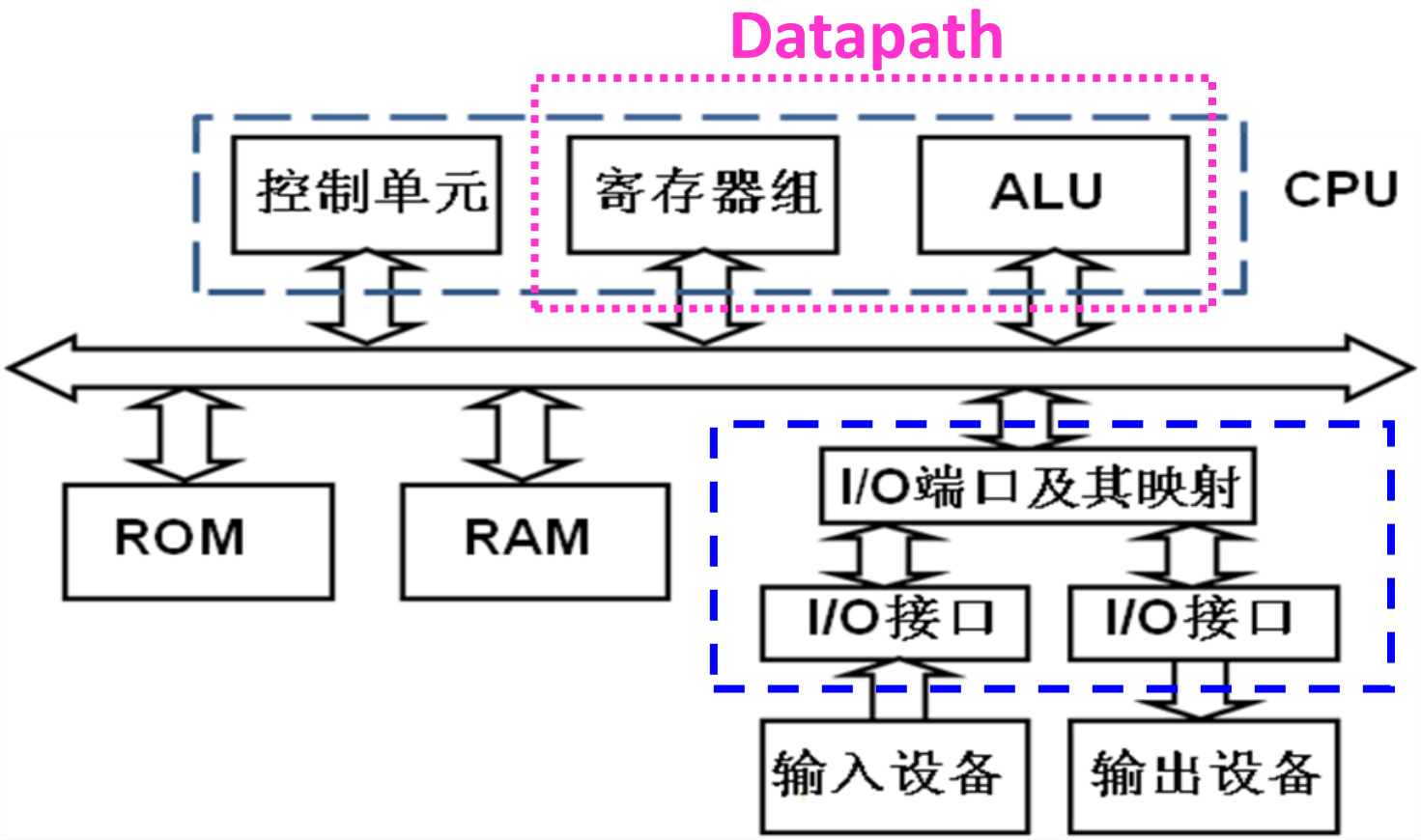
提纲

- 计算机指令系统（MIPS）
  - 指令系统
  - 寻址方式

- **计算机逻辑电路模块**
  - 计算机微体系结构
  - 运算器（ALU）
  - 多路开关、译码器
  - 寄存器、指令指针（程序计数器）与取指单元
  - 存储器（微处理器外部）

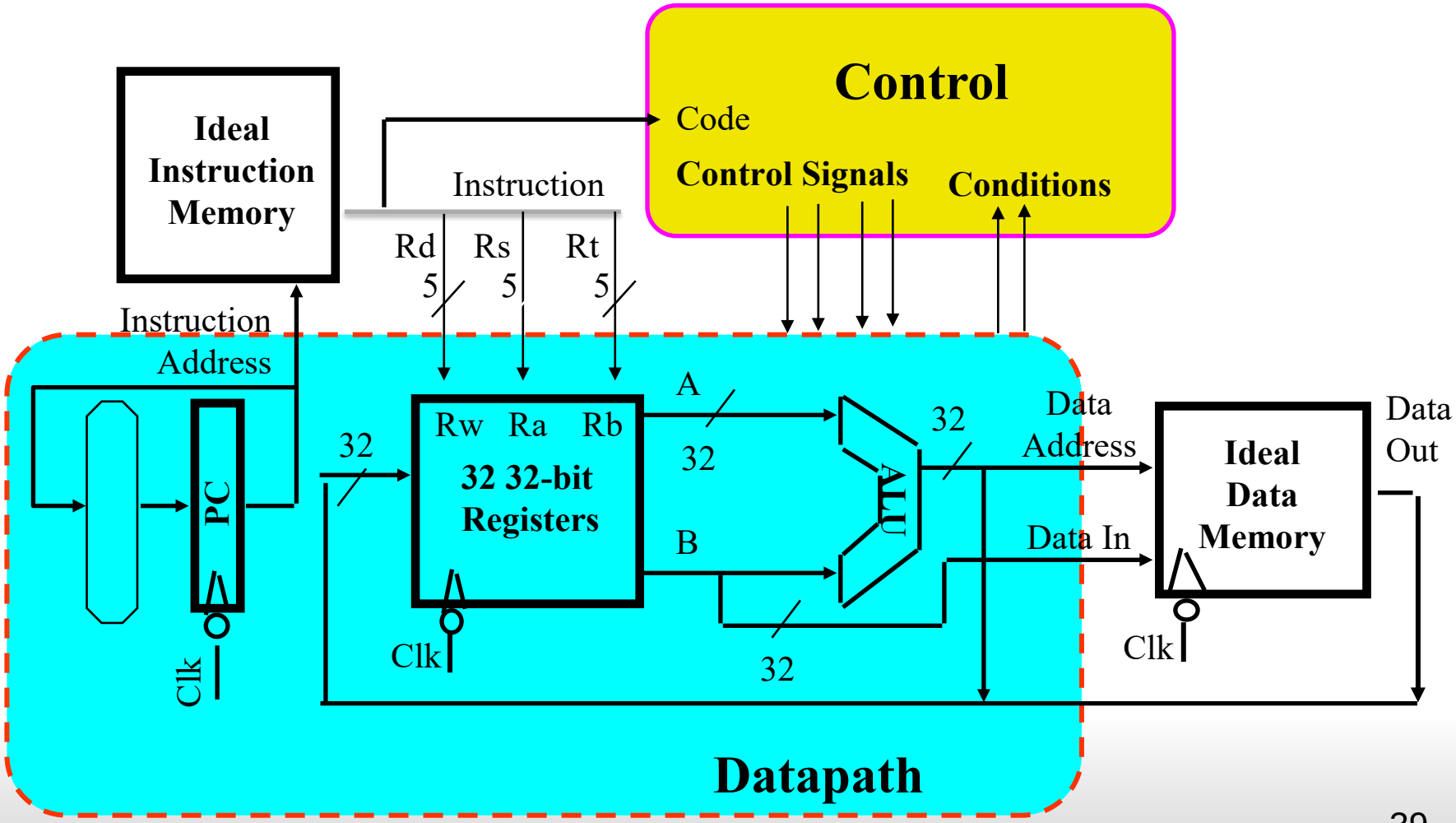
计算机逻辑电路模块

微体系结构核心：逻辑电路控制 + 数据通路（Datapath）



计算机逻辑电路模块

微体系结构核心：32位MIPS典型结构

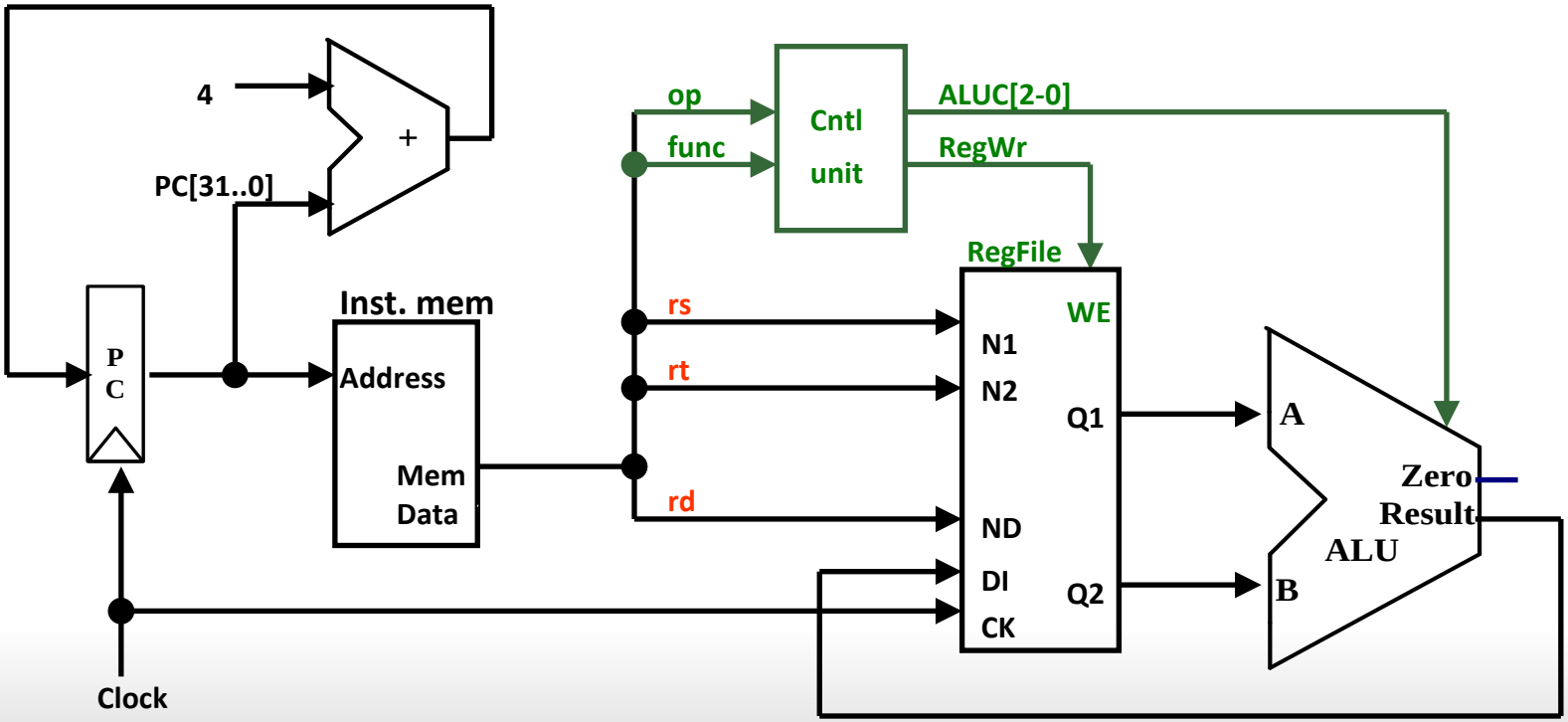


计算机逻辑电路模块

微体系结构核心：32位MIPS指令数据通路

R-format add/sub/and/or/slt rs, rt, rd # rd = rs *op* rt

Opcode (6 bits)	Rs (5 bits)	Rt (5 bits)	Rd (5 bits)	Shift amt (5 bits)	Func (6 bits)



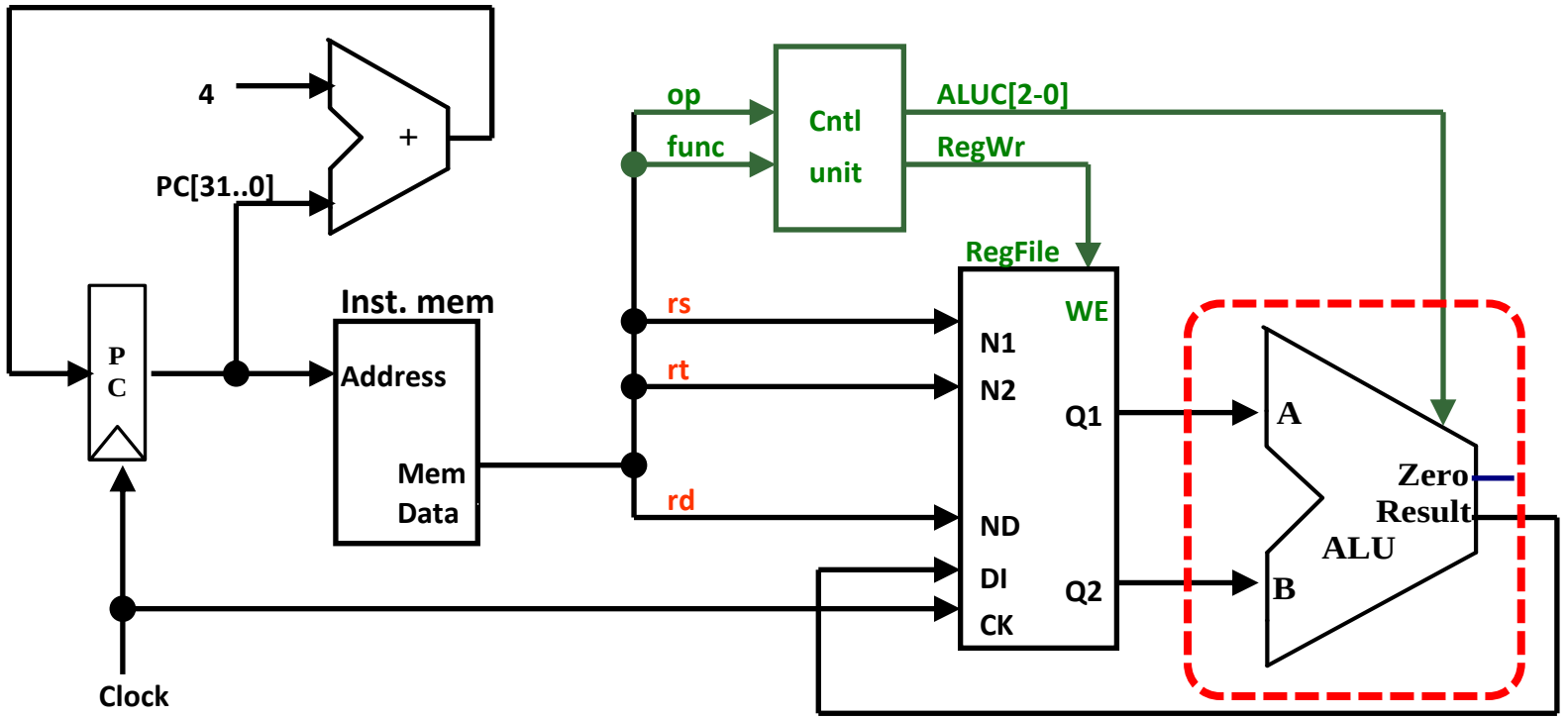
计算机逻辑电路模块

计算机常用逻辑电路模块

- 运算器（ALU）
  - 加法器、减法器、乘法器、除法器
  - 逻辑运算器、比较器（关系运算）
- 多路开关/数据选择器
- 译码器（指令译码、地址译码）
- 寄存器与寄存器组
- 指令指针（程序计数器）与取指单元
- 存储器（微处理器外部）
- I/O接口（微处理器内&外部）

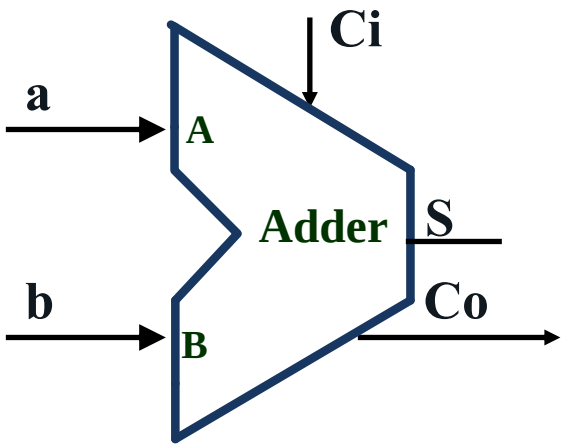
计算机逻辑电路模块

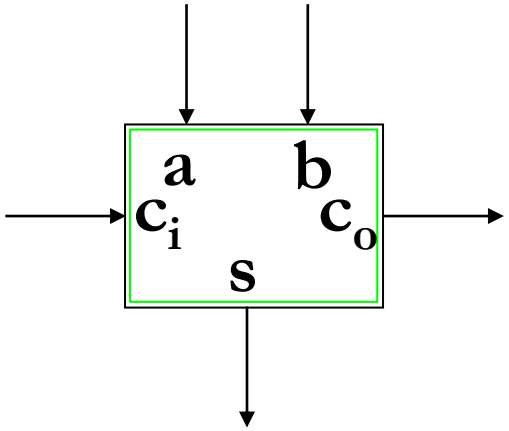
运算器（Arithmetic Logic Unit, ALU 算术逻辑单元）



计算机逻辑电路模块

运算器（ALU）：1位全加器





input a, b, c_i

// c_i : carry in（进位标志）

output c_o, s

// c_o: carry out（进位标志）

// s: sum

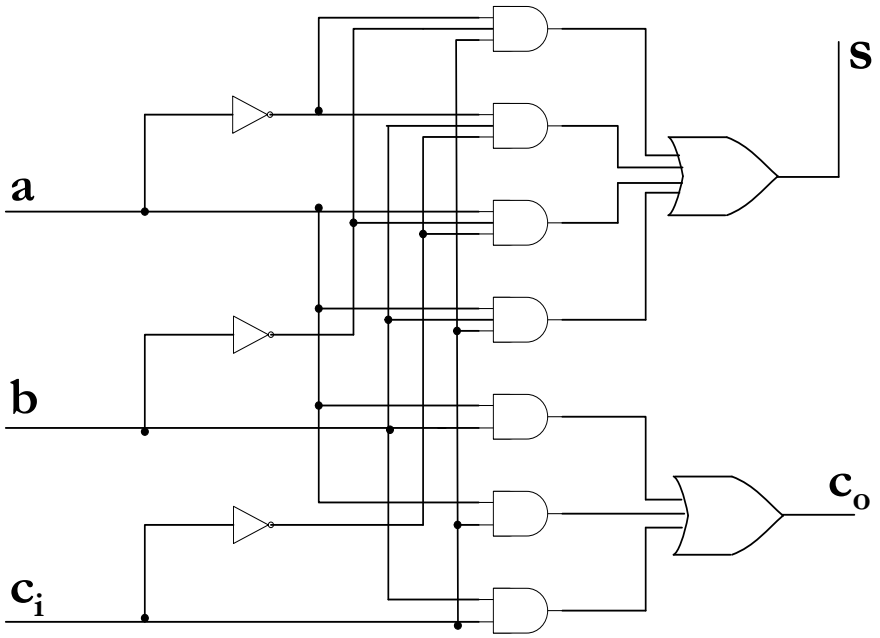
计算机逻辑电路模块

运算器（ALU）：1位全加器

a	b	c_i	c_o	s	Comments
0	0	0	0	0	0+0+0 = 0 0
0	0	1	0	1	0+0+1 = 0 1
0	1	0	0	1	0+1+0 = 0 1
0	1	1	1	0	0+1+1 = 1 0
1	0	0	0	1	1+0+0 = 0 1
1	0	1	1	0	1+0+1 = 1 0
1	1	0	1	0	1+1+0 = 1 0
1	1	1	1	1	1+1+1 = 1 1

$$s = \bar{a}\bar{b}c_i + \bar{a}b\bar{c}_i + a\bar{b}\bar{c}_i + abc_i$$

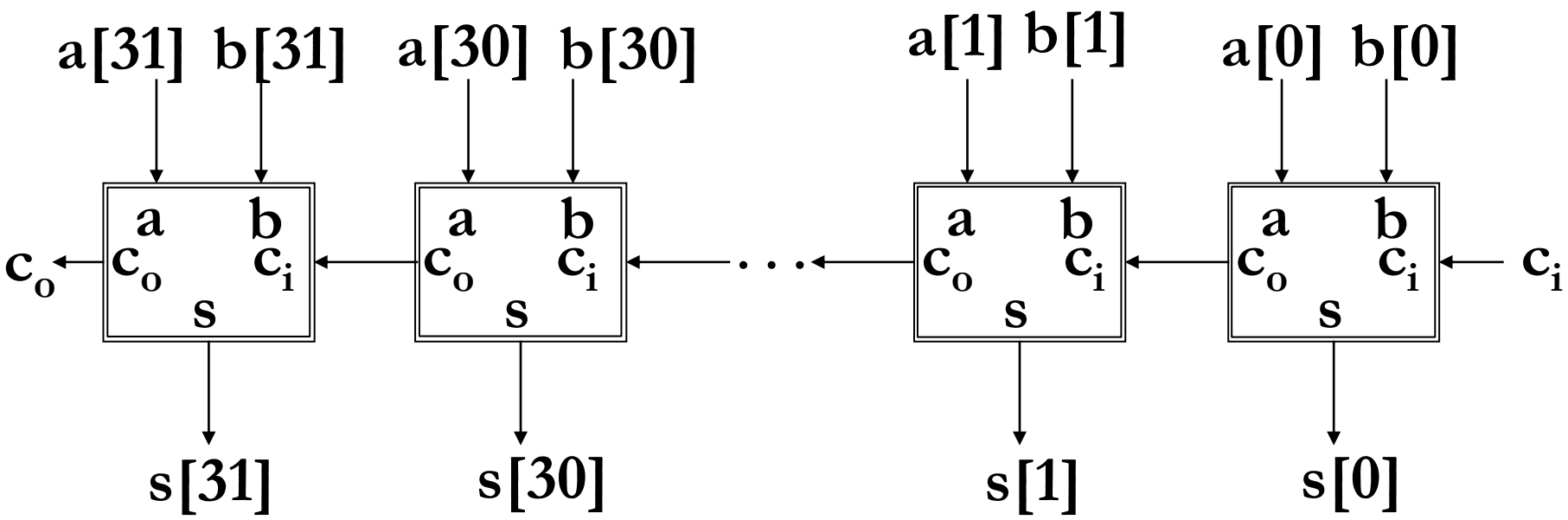
$$c_o = ab + ac_i + bc_i$$



组合逻辑电路

计算机逻辑电路模块

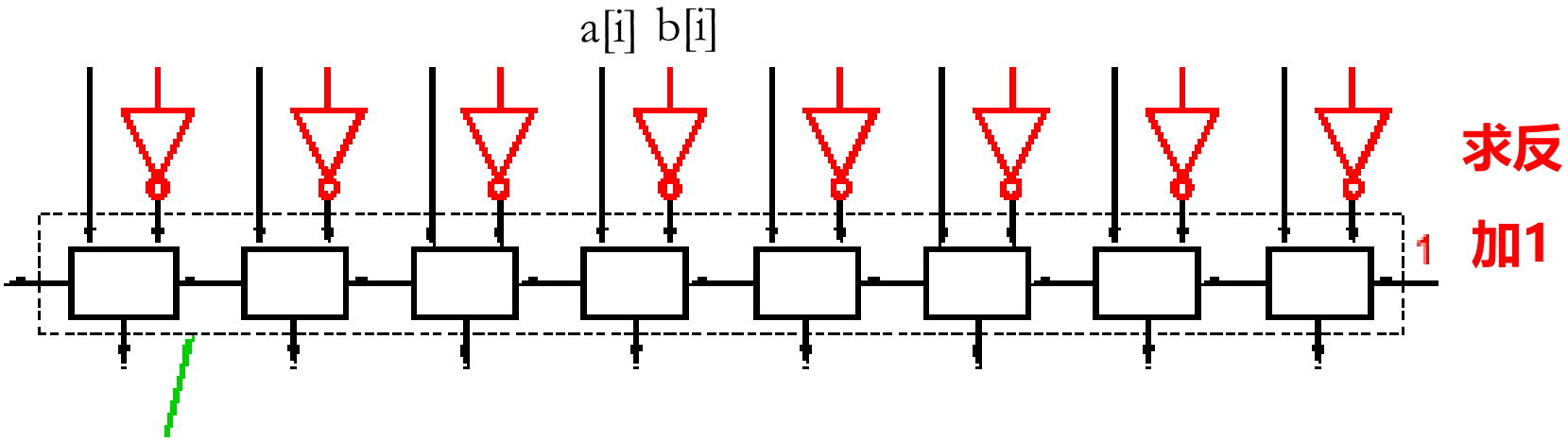
运算器（ALU）：32位全加器



32位全加器 = 32个1位全加器级联

计算机逻辑电路模块

运算器（ALU）：减法器

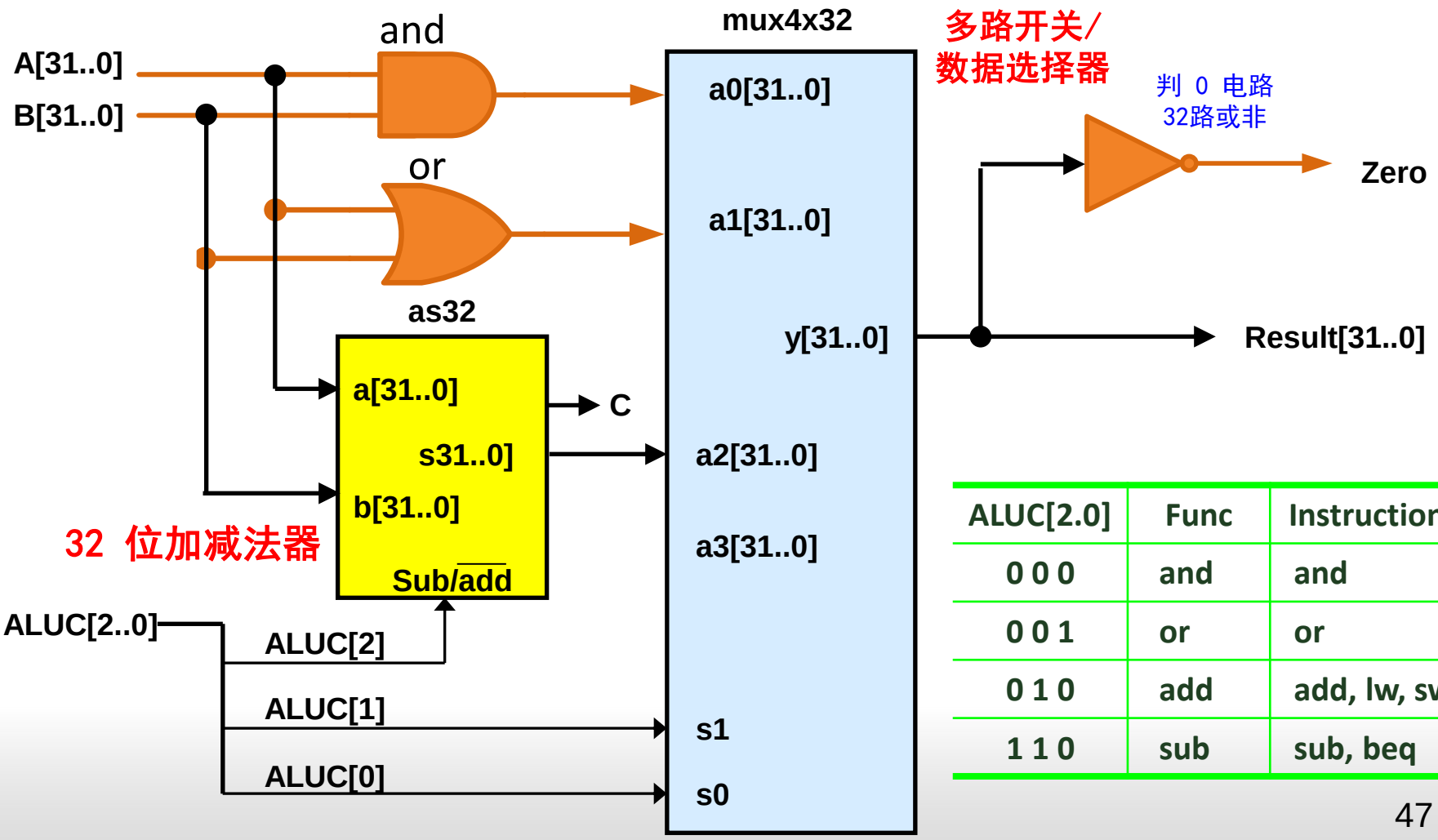


补码 = 求反加1　　$-x = \overline{x} + 1$

$$a - b = a + (-b) = a + \overline{b} + 1$$

计算机逻辑电路模块

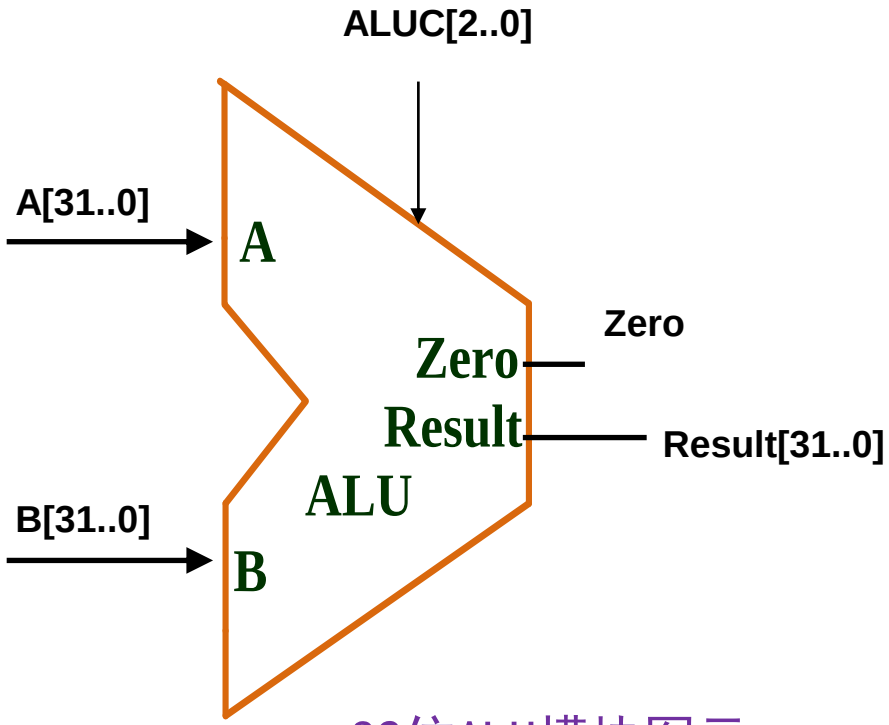
运算器（ALU）：32位ALU结构



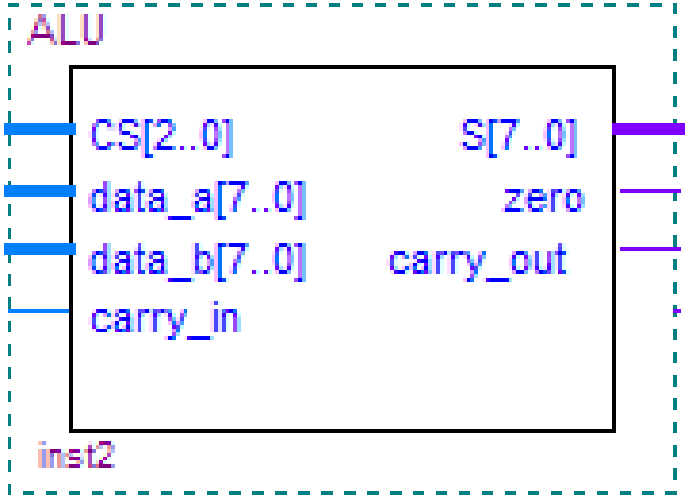
ALUC[2.0]	Func	Instructions
0 0 0	and	and
0 0 1	or	or
0 1 0	add	add, lw, sw
1 1 0	sub	sub, beq

计算机逻辑电路模块

运算器（ALU）：ALU图符



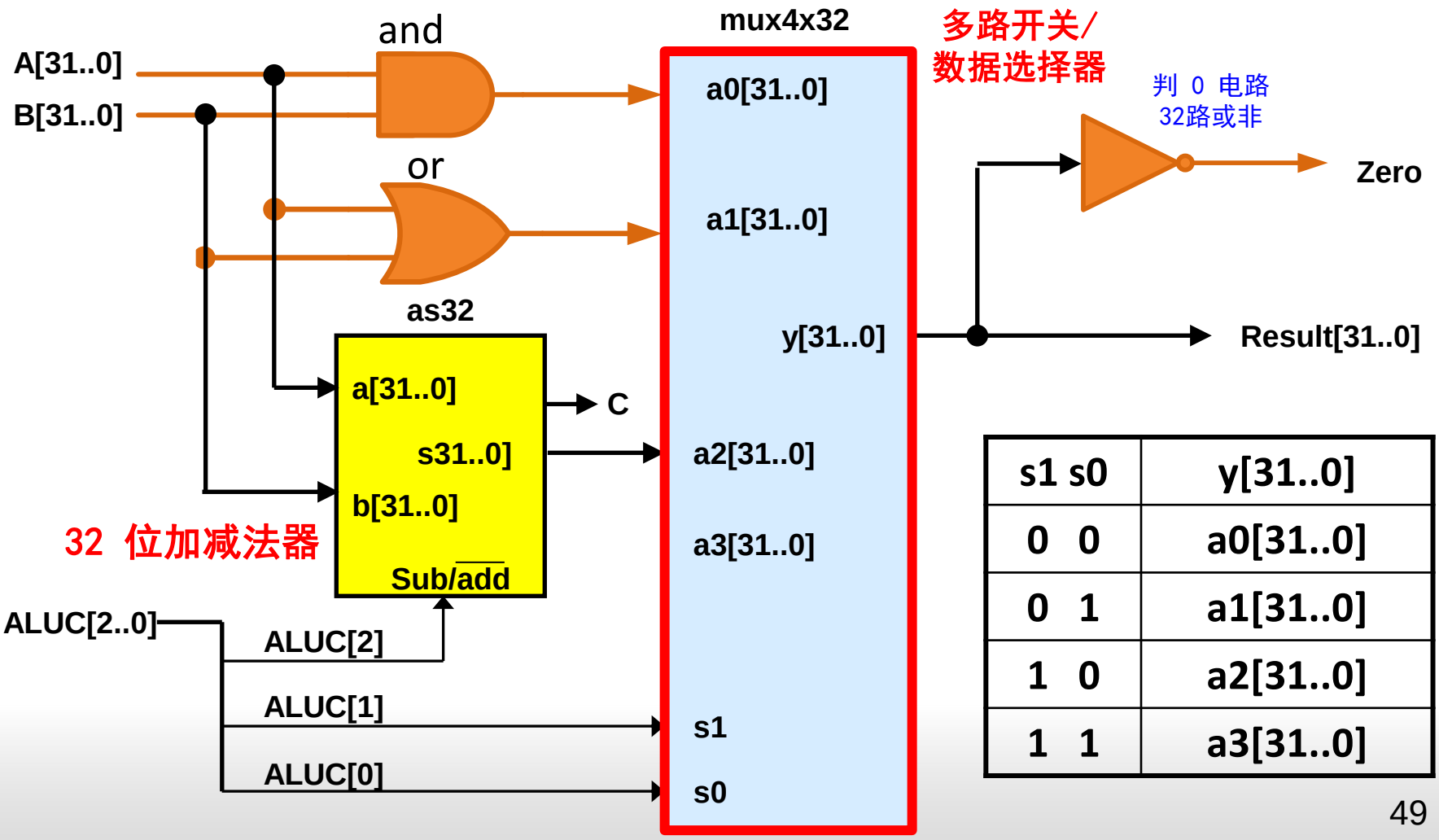
32位ALU模块图示



8位ALU模块封装图

计算机逻辑电路模块

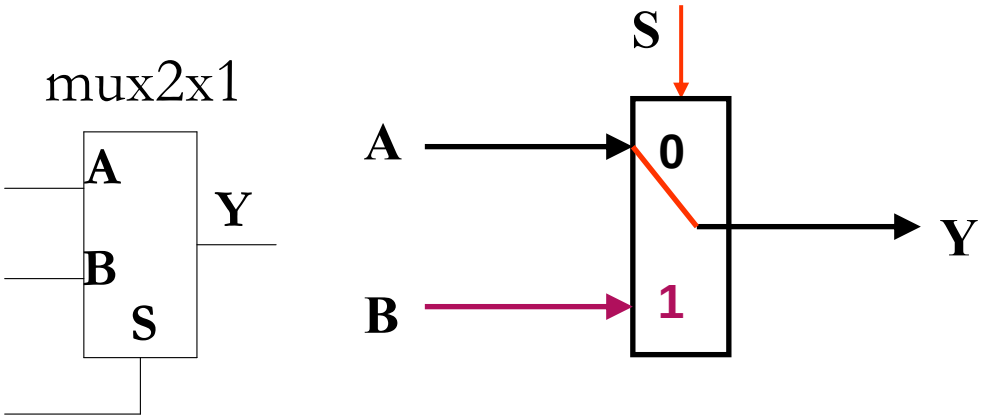
多路开关/数据选择器（Multiplexer）

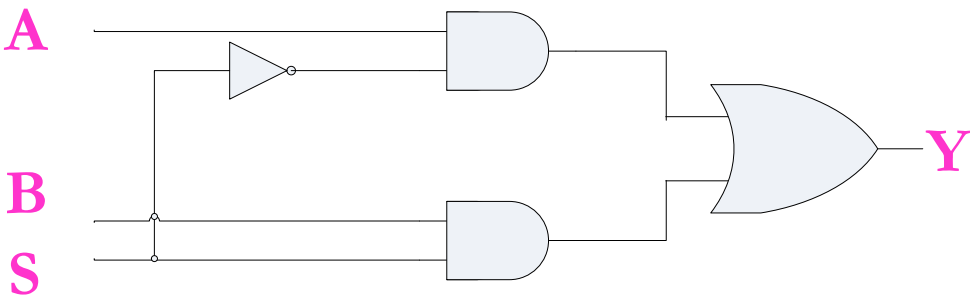


s1 s0	y[31..0]
0 0	a0[31..0]
0 1	a1[31..0]
1 0	a2[31..0]
1 1	a3[31..0]

计算机逻辑电路模块

多路开关/数据选择器（Multiplexer）：2选1

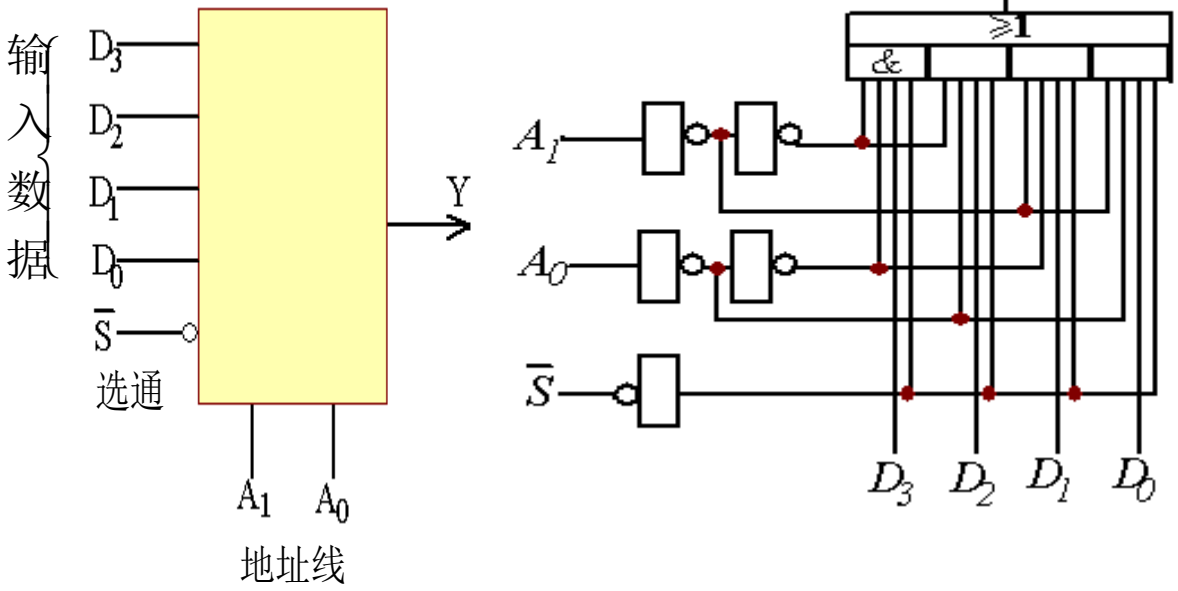




S	A	B	Y
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

计算机逻辑电路模块

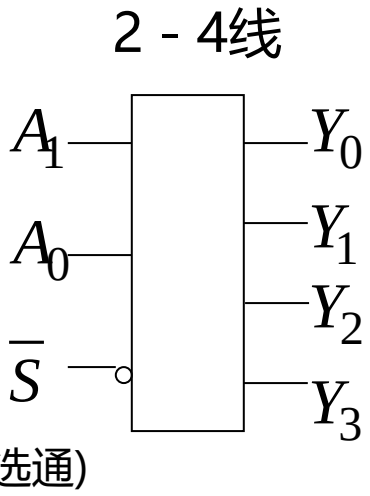
多路开关/数据选择器（Multiplexer）：4选1

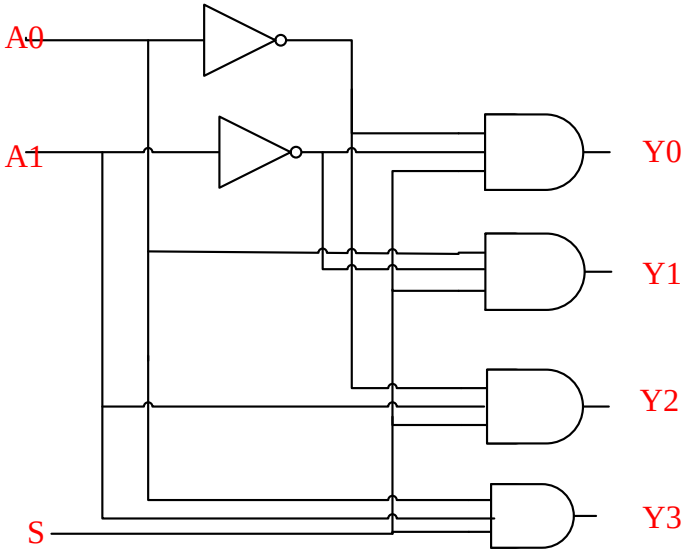


$\overline{S}$	A_1	A_2	Y
1	X	X	0
0	0	0	D_0
0	0	1	D_1
0	1	0	D_2
0	1	1	D_3

计算机逻辑电路模块

译码器：2-4译码器（n >> 2^n）



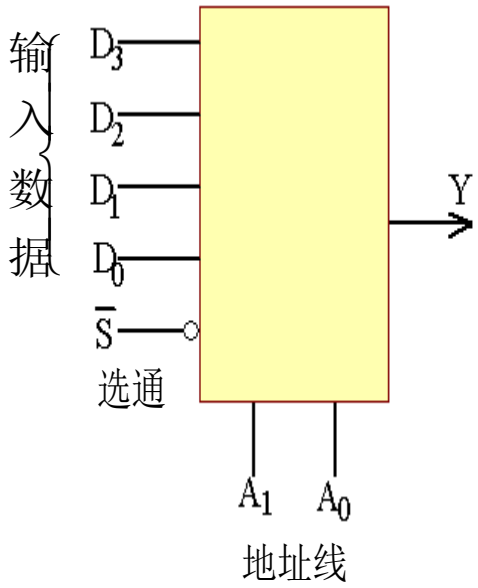


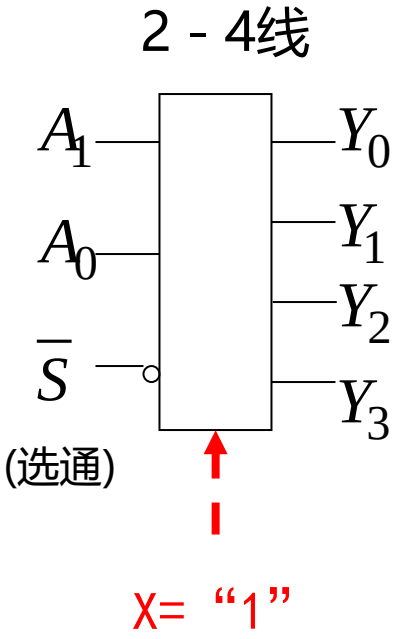
输出Y：通常作为下一级模块的选择、使能等信号

S	A_1	A_2	Y_i
1	X	X	0000
0	0	0	0001
0	0	1	0010
0	1	0	0100
0	1	1	1000

计算机逻辑电路模块

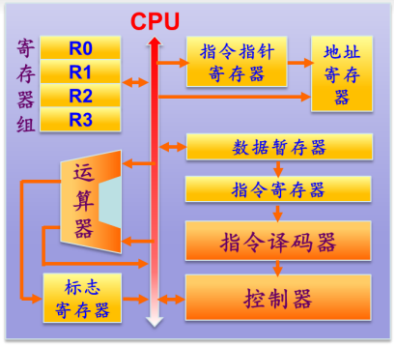
思考：多路开关/数据选择器、译码器的区别？



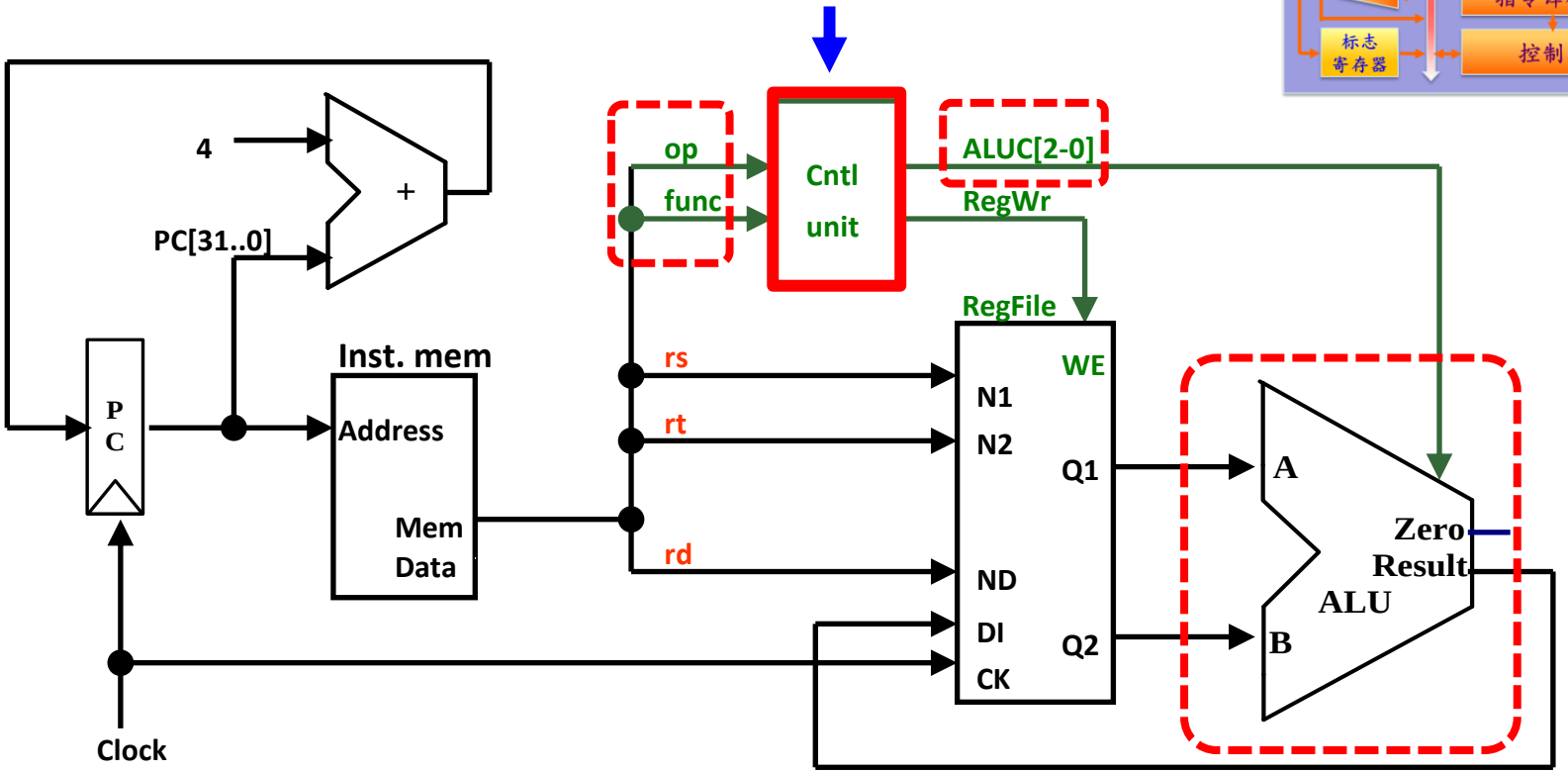


计算机逻辑电路模块

译码器：用于指令译码

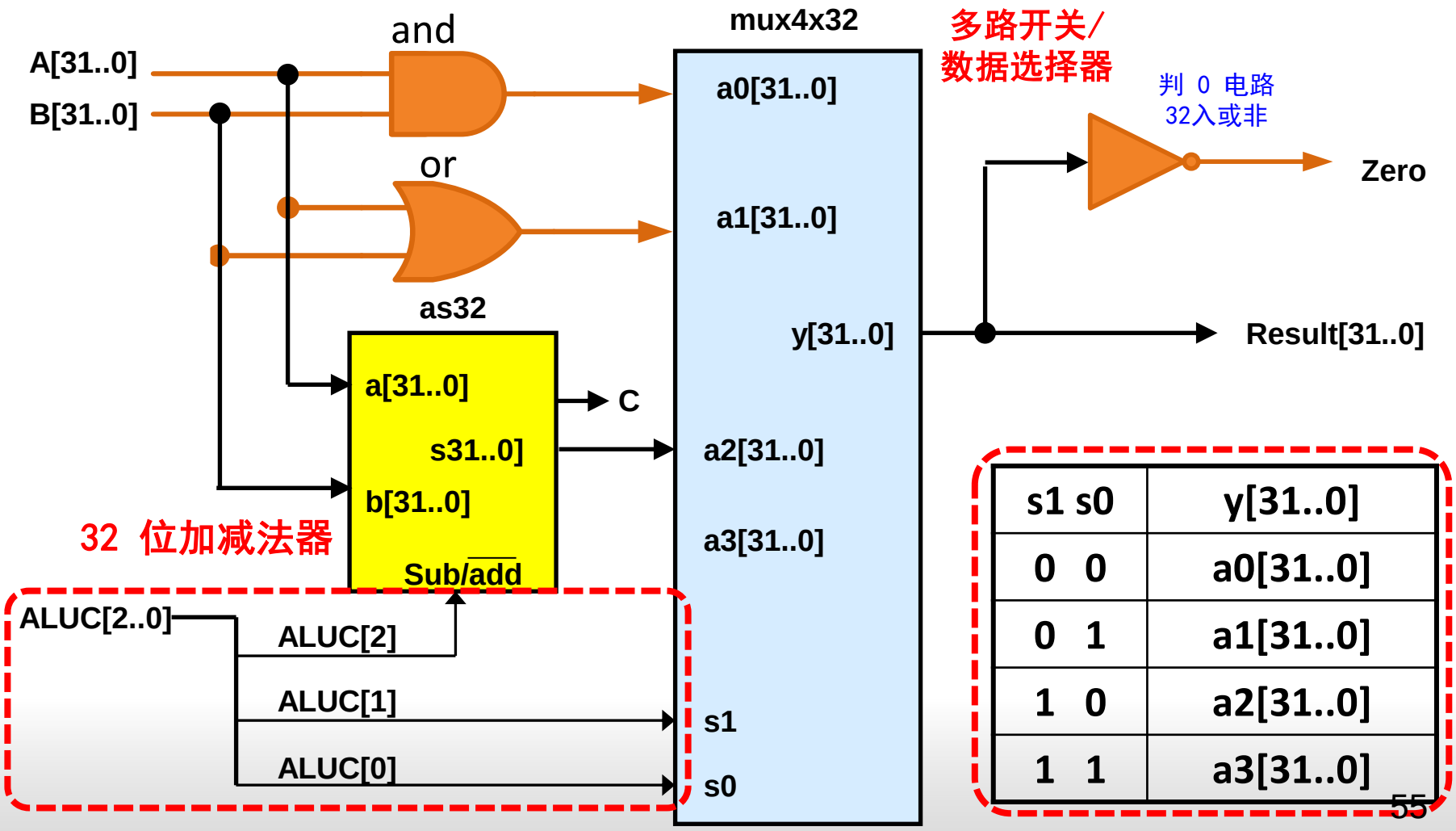


控制器：内含指令译码器



计算机逻辑电路模块

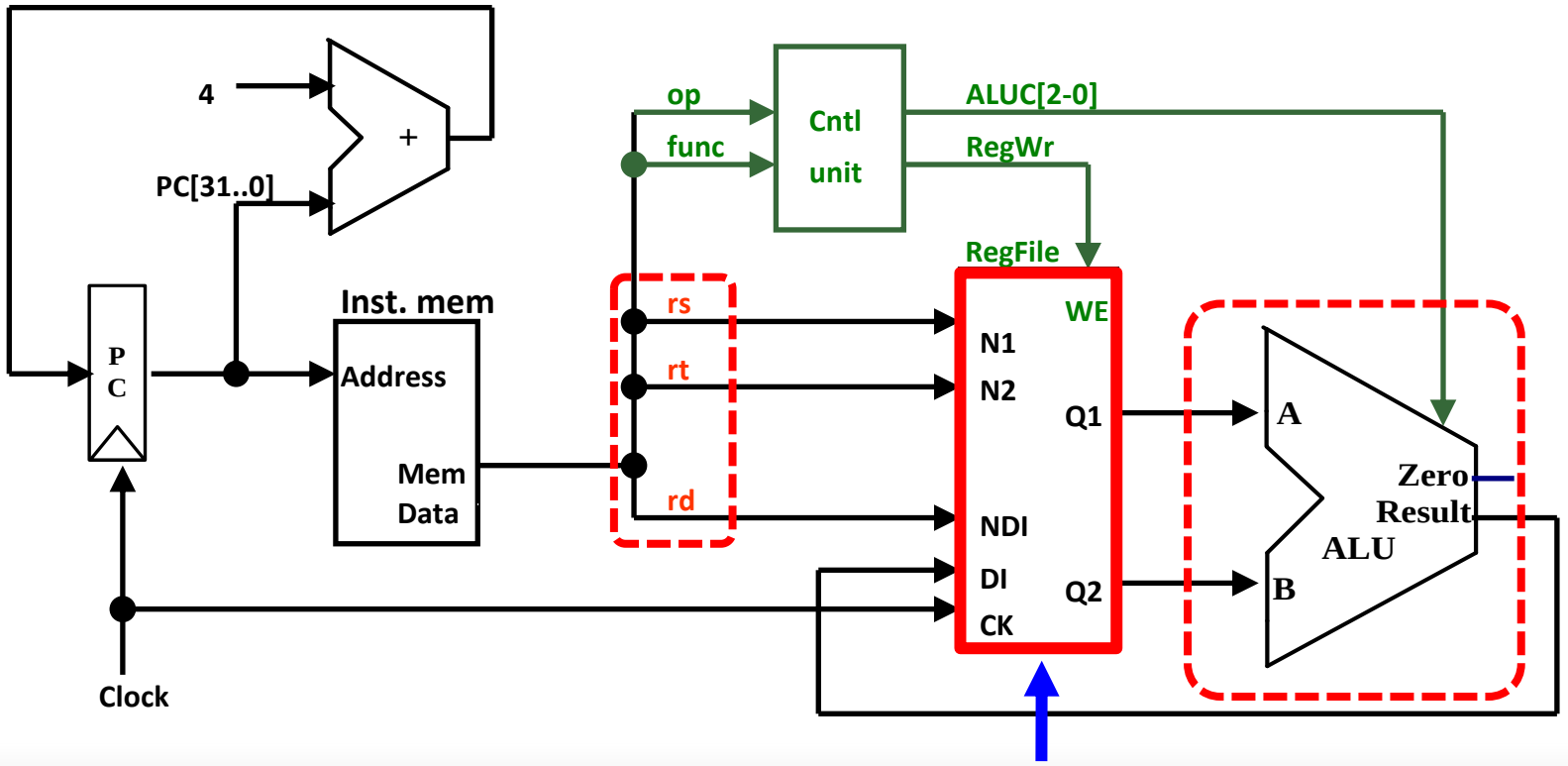
译码器：用于指令译码



s1 s0	y[31..0]
0 0	a0[31..0]
0 1	a1[31..0]
1 0	a2[31..0]
1 1	a3[31..0]

计算机逻辑电路模块

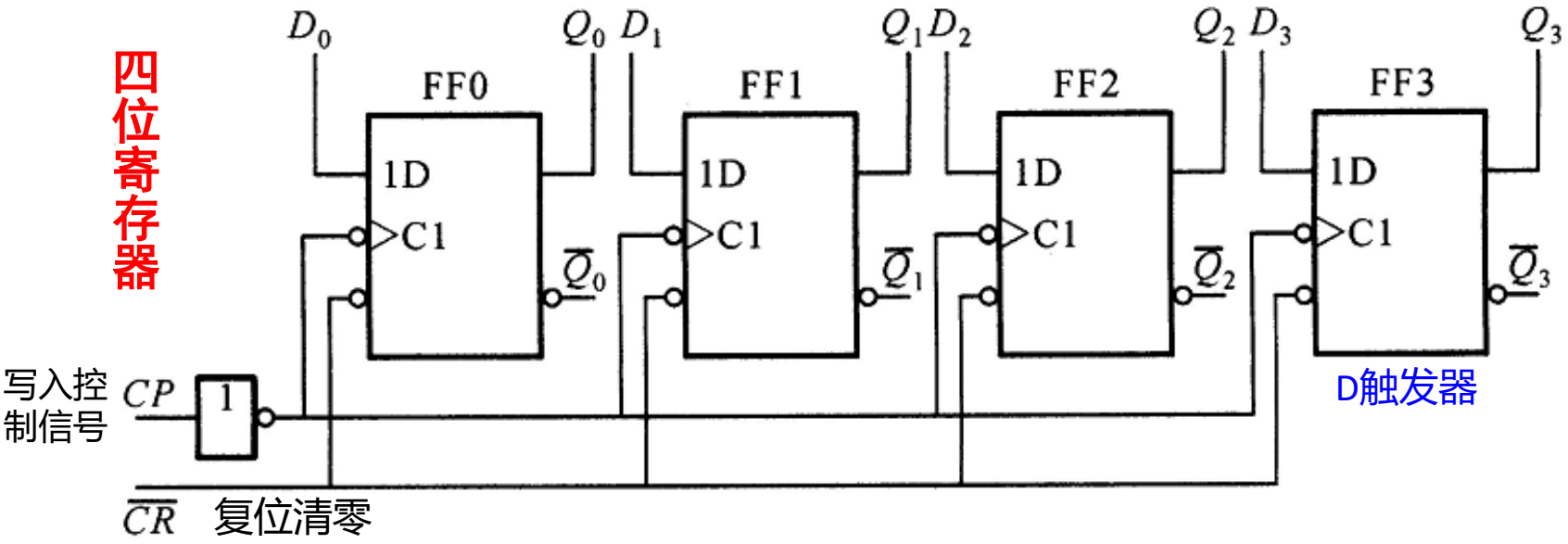
译码器：用于地址译码



寄存器组：内含地址译码器

计算机逻辑电路模块

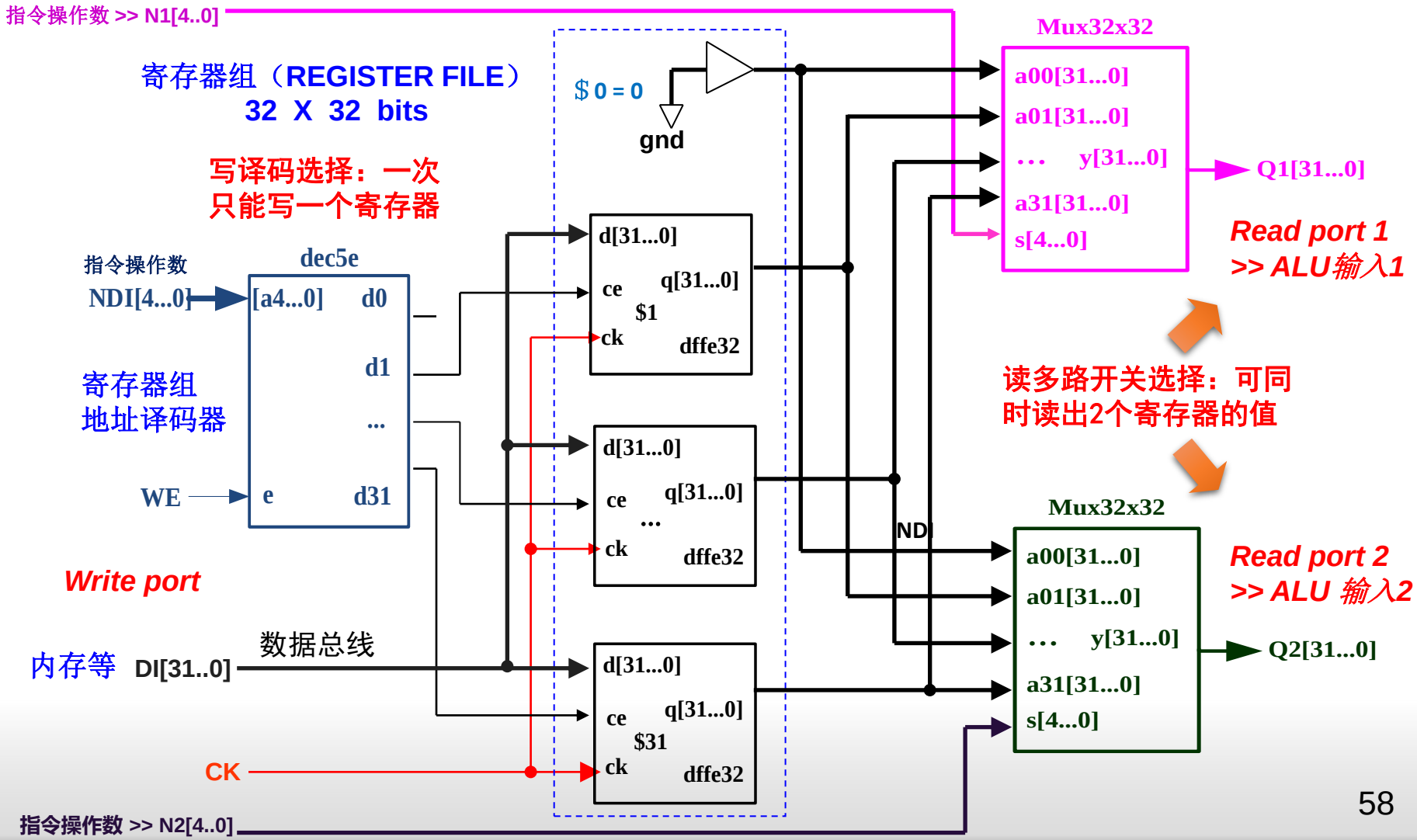
寄存器（Register）：4位寄存器



寄存器： CPU的重要组成部分，用来暂存指令、数据和地址

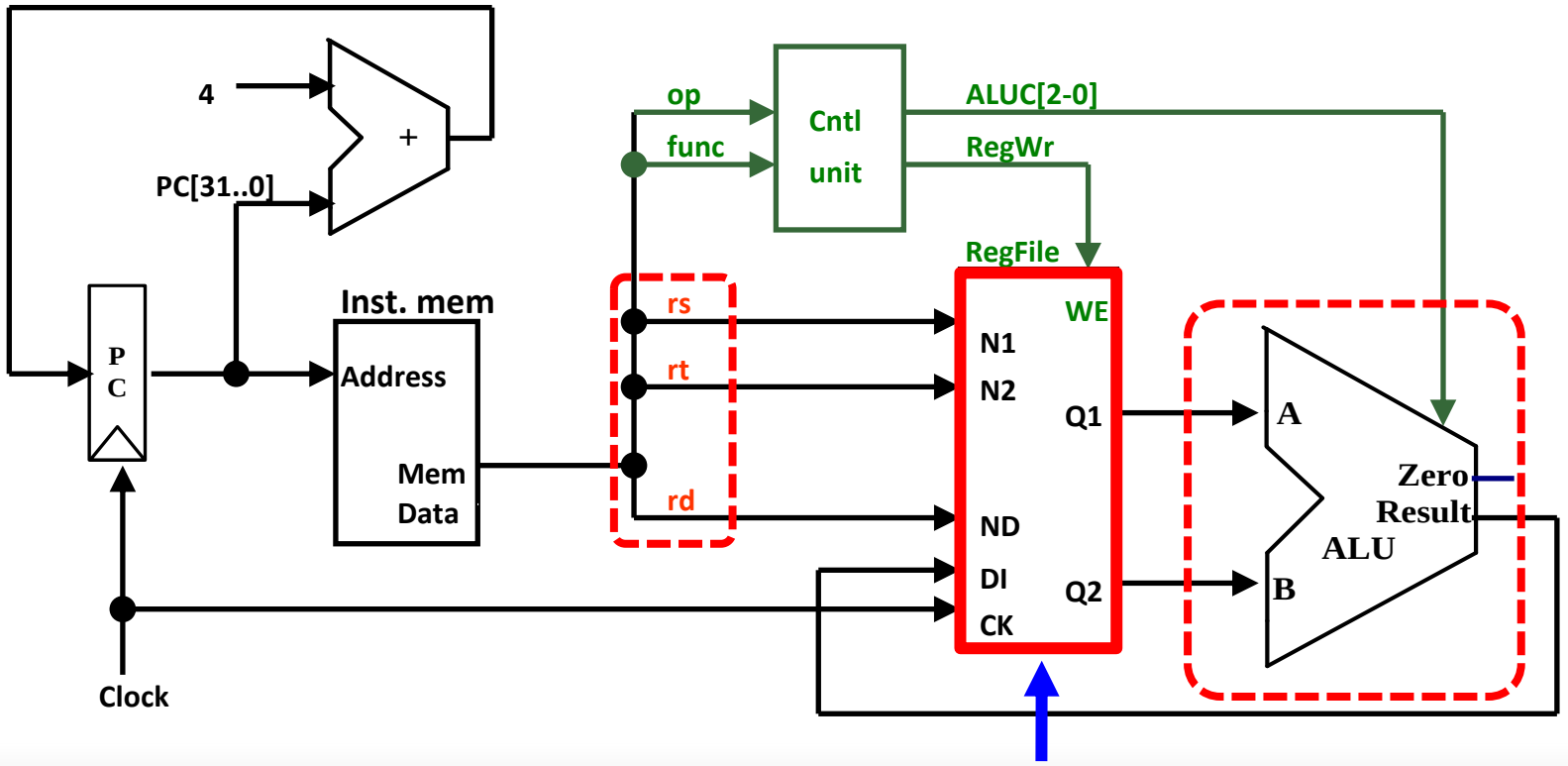
计算机逻辑电路模块

寄存器（Register）：寄存器组



计算机逻辑电路模块

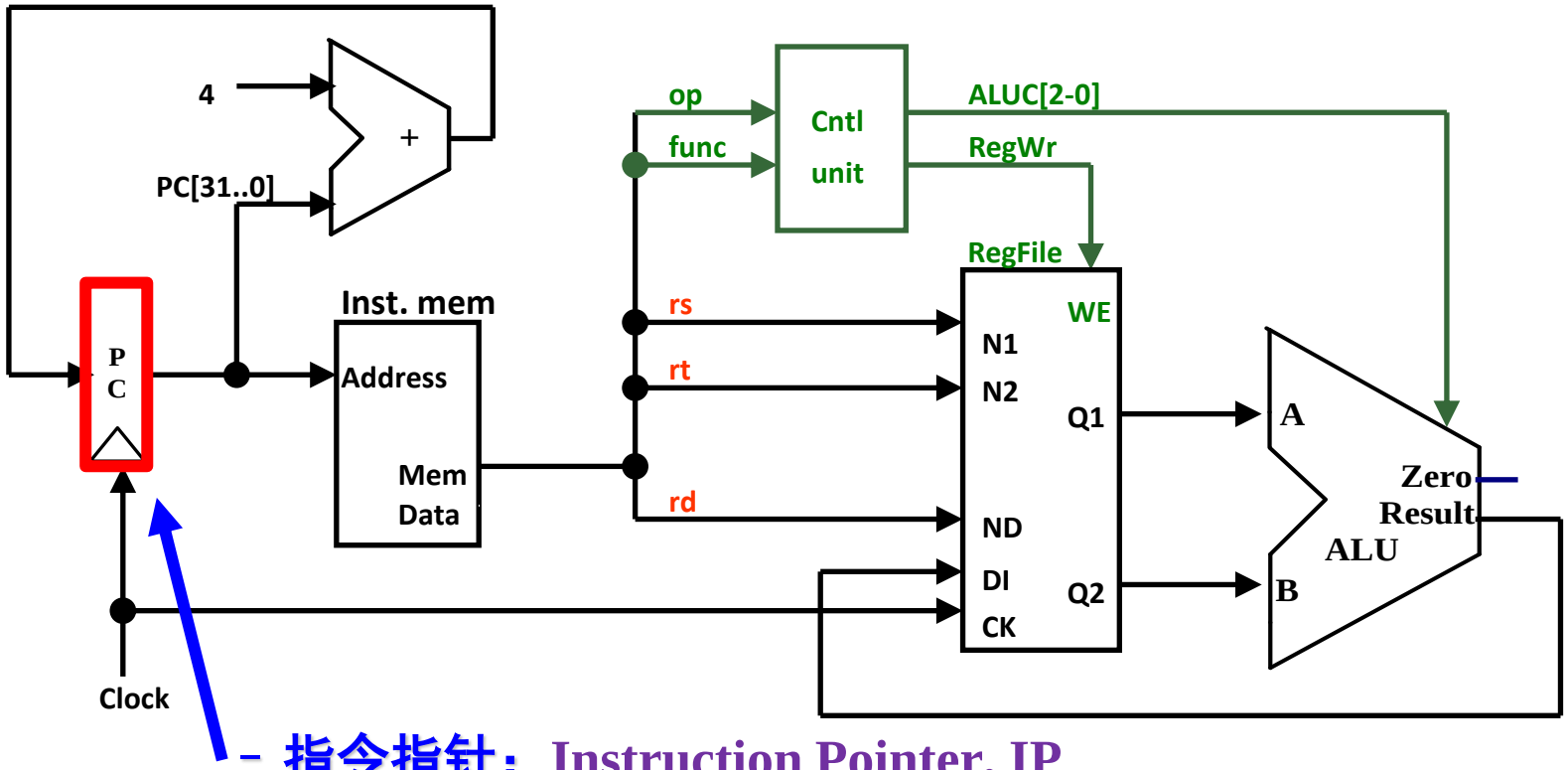
译码器：用于地址译码



寄存器组：内含地址译码器

计算机逻辑电路模块

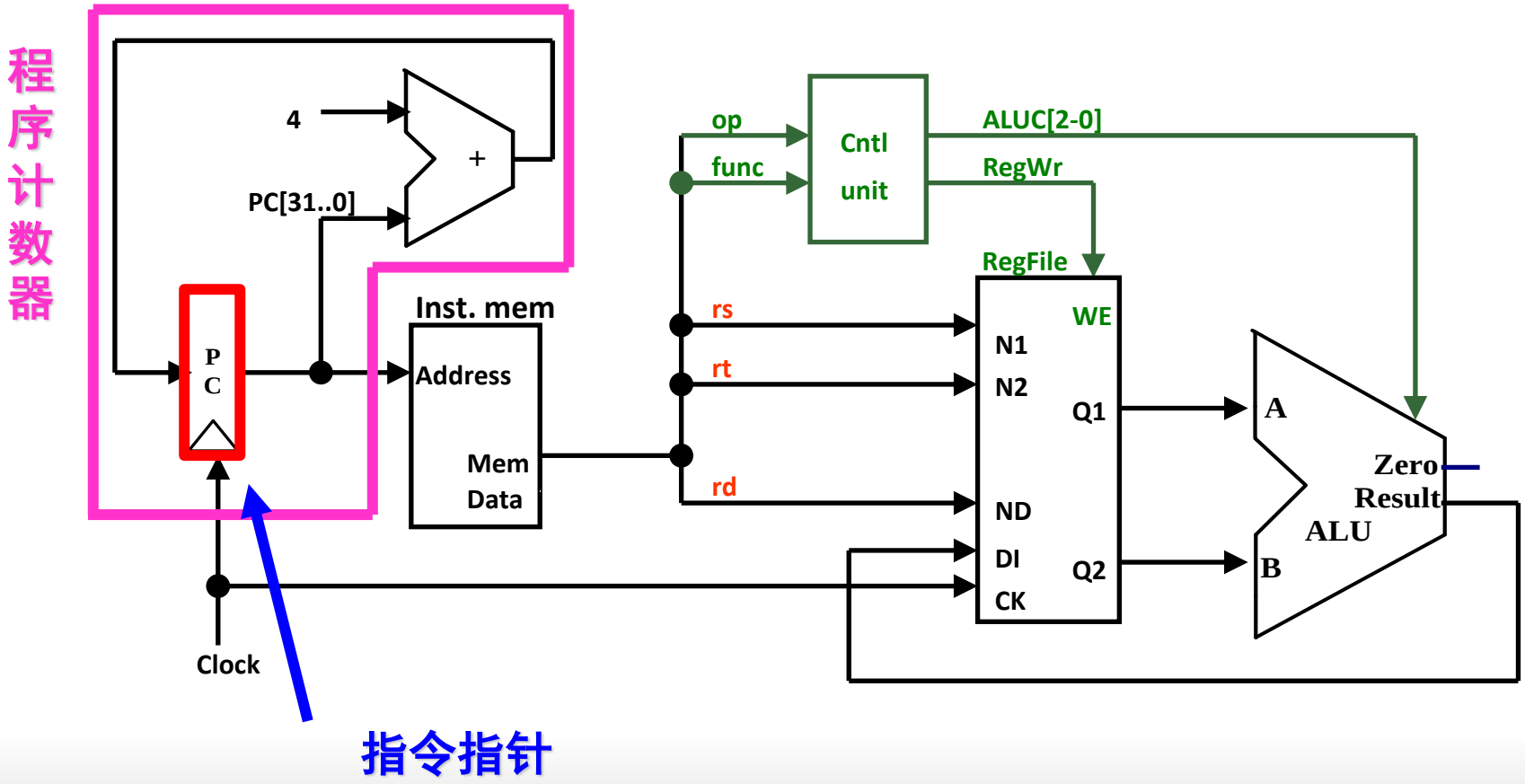
指令指针（程序计数器）与取指单元



- 指令指针：Instruction Pointer, IP
- 程序计数器：Program Counter, PC

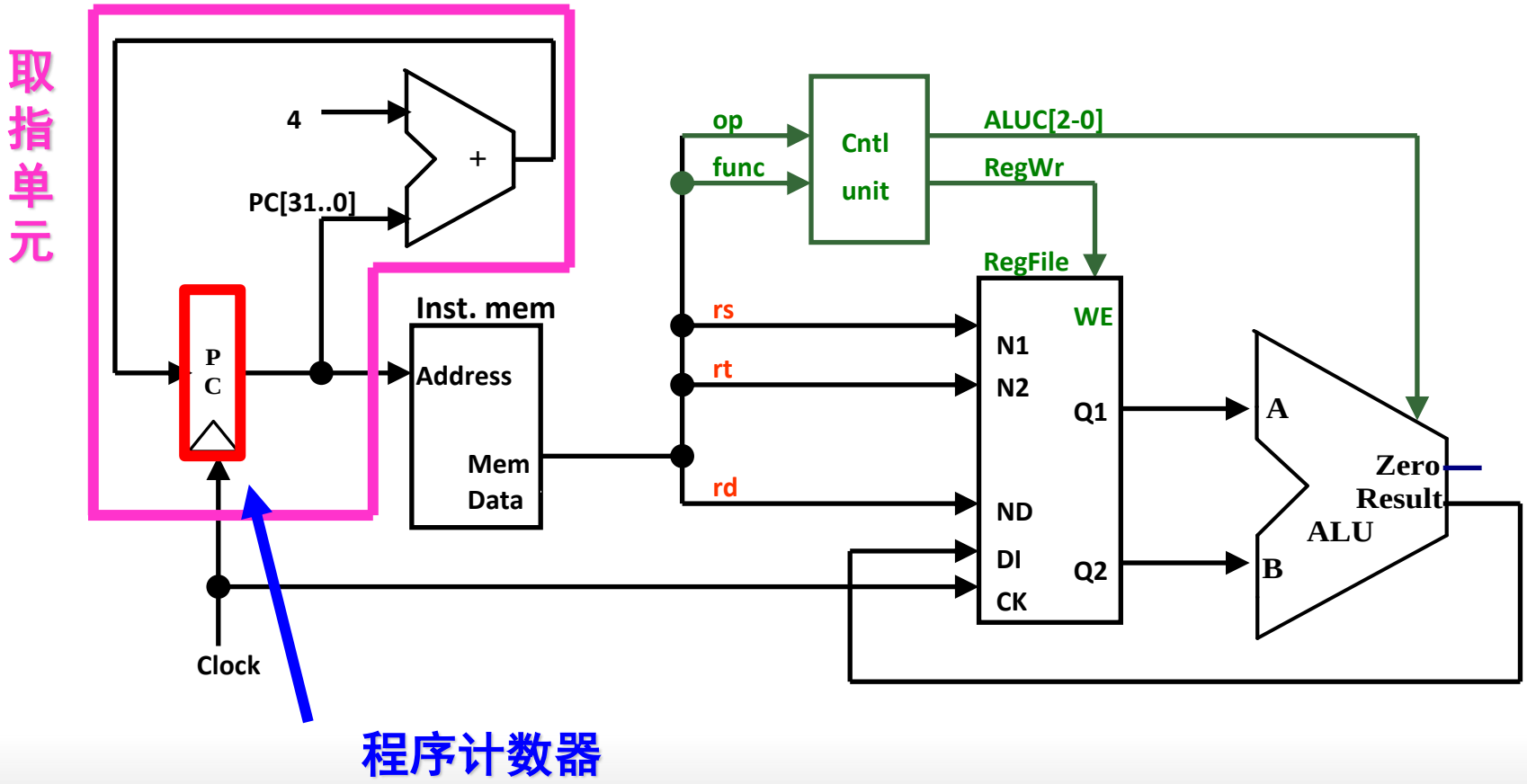
计算机逻辑电路模块

指令指针（程序计数器）与取指单元



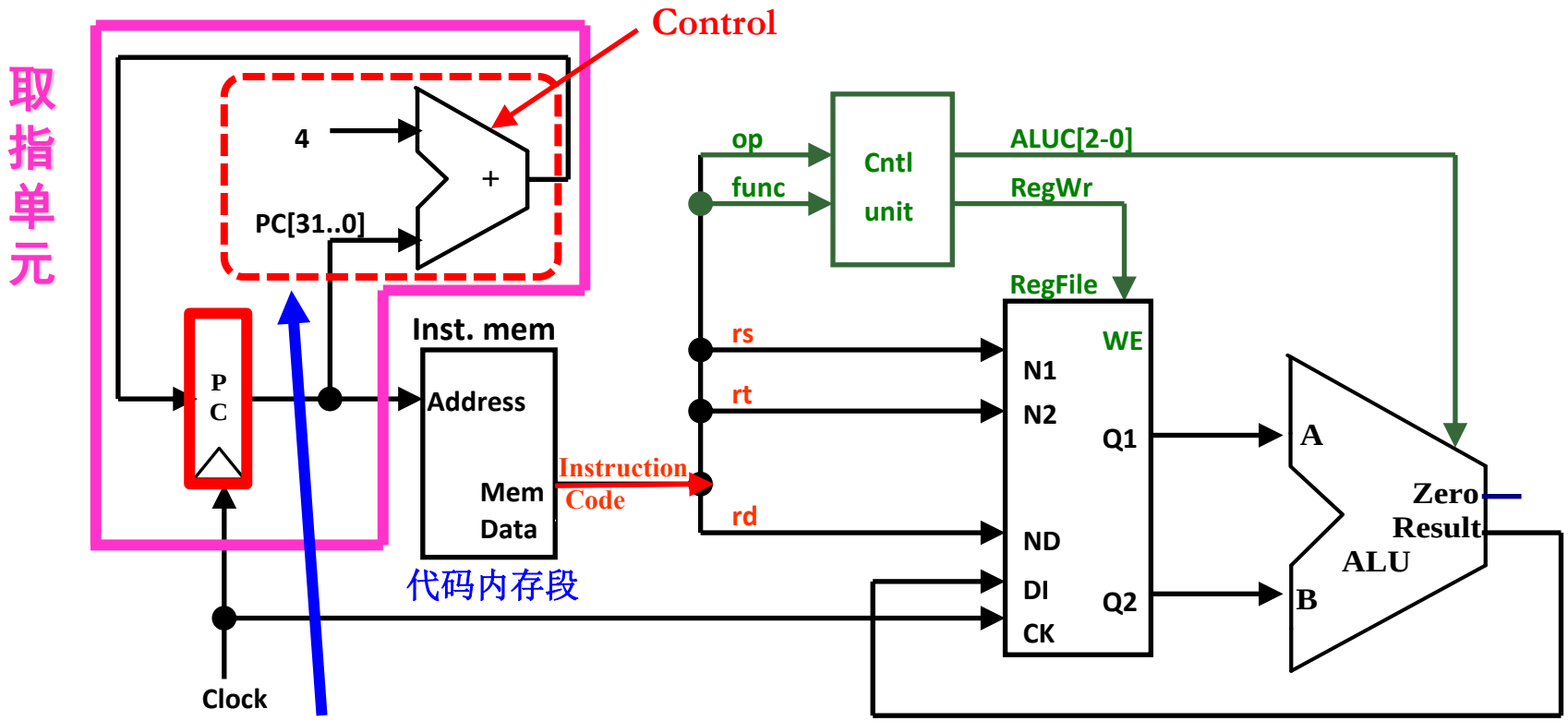
计算机逻辑电路模块

指令指针（程序计数器）与取指单元



计算机逻辑电路模块

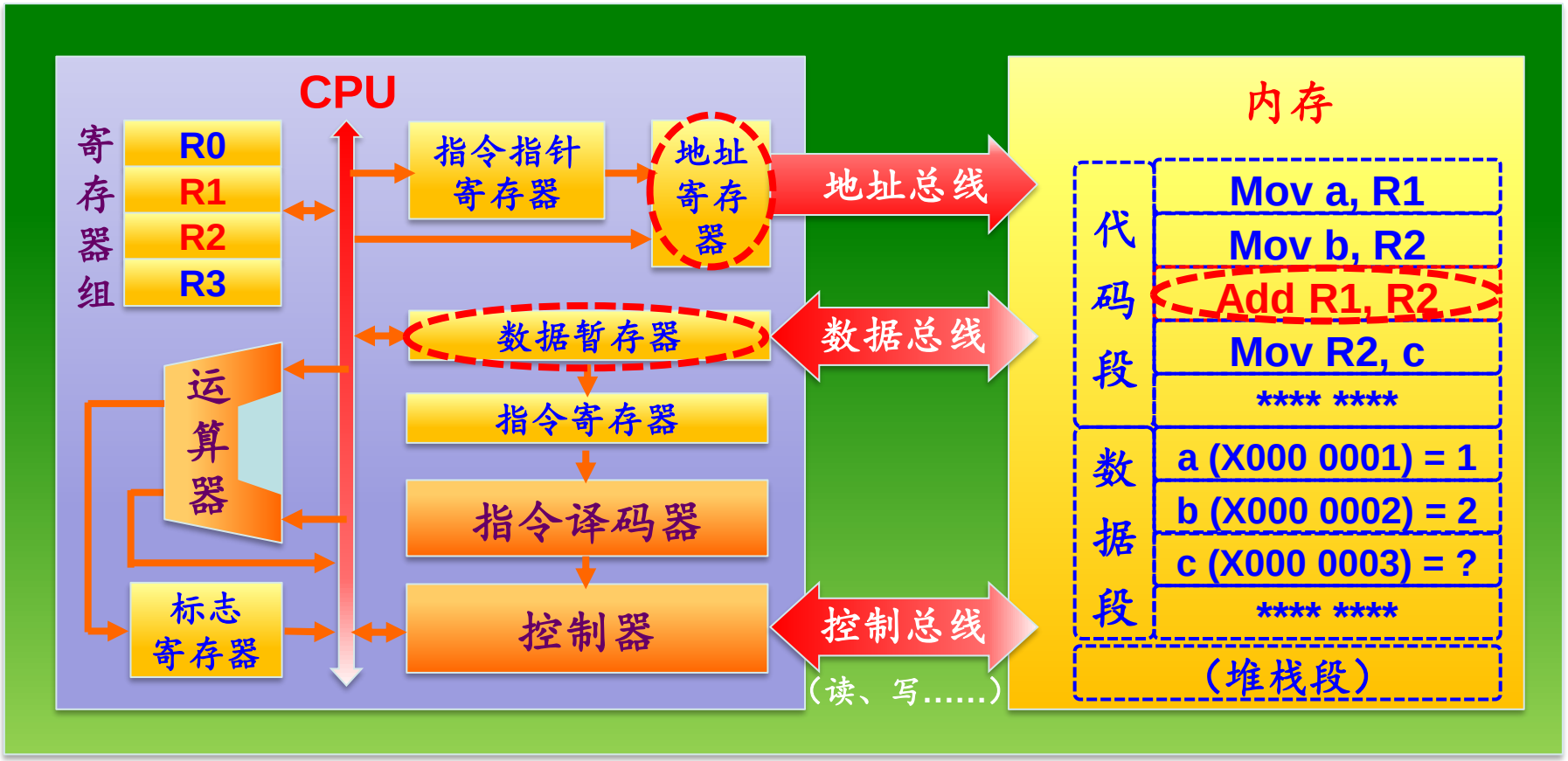
指令指针（程序计数器）与取指单元



PC+4：指向下一条指令（32位）

CPU微观工作流程

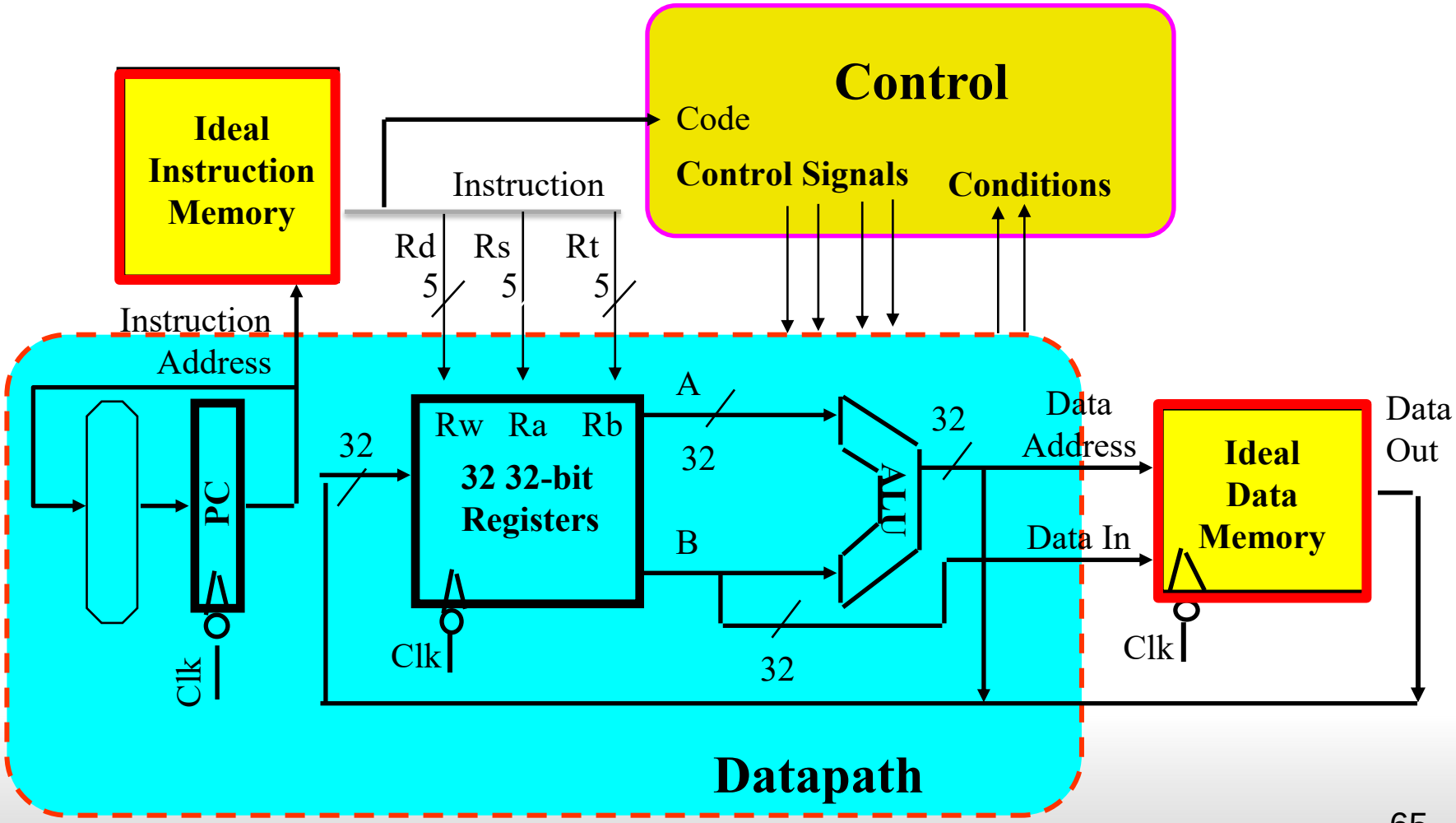
单条指令执行过程分解：取指令 → 译码 → 执行



数据通路（Datapath）：指令执行过程中，数据所经过的路径，包括路径中的指令执行部件
指令控制部件（Control）：对执行部件发出控制信号，包括对指令进行译码，生成指令对应的控制信号，控制数据通路的动作

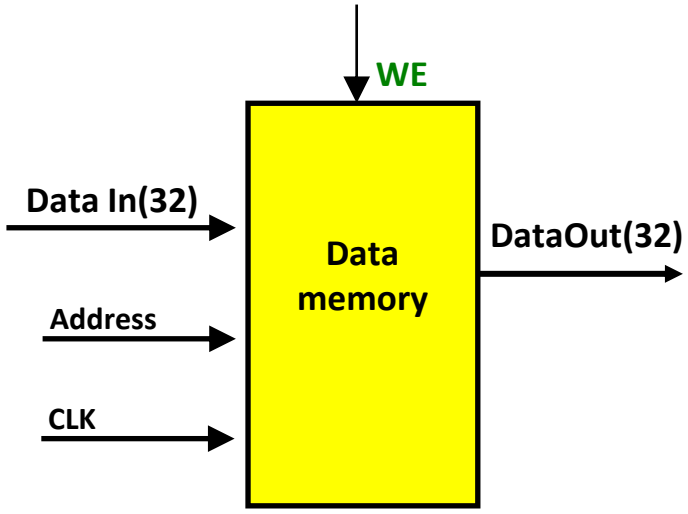
计算机逻辑电路模块

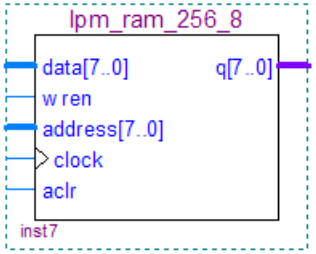
存储器（Memory）

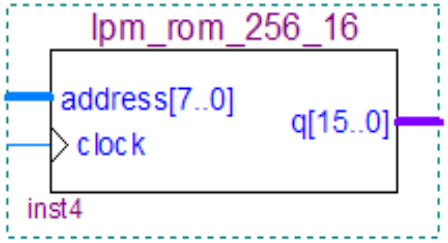


计算机逻辑电路模块

存储器（Memory）：~= 寄存器组 + 地址译码器

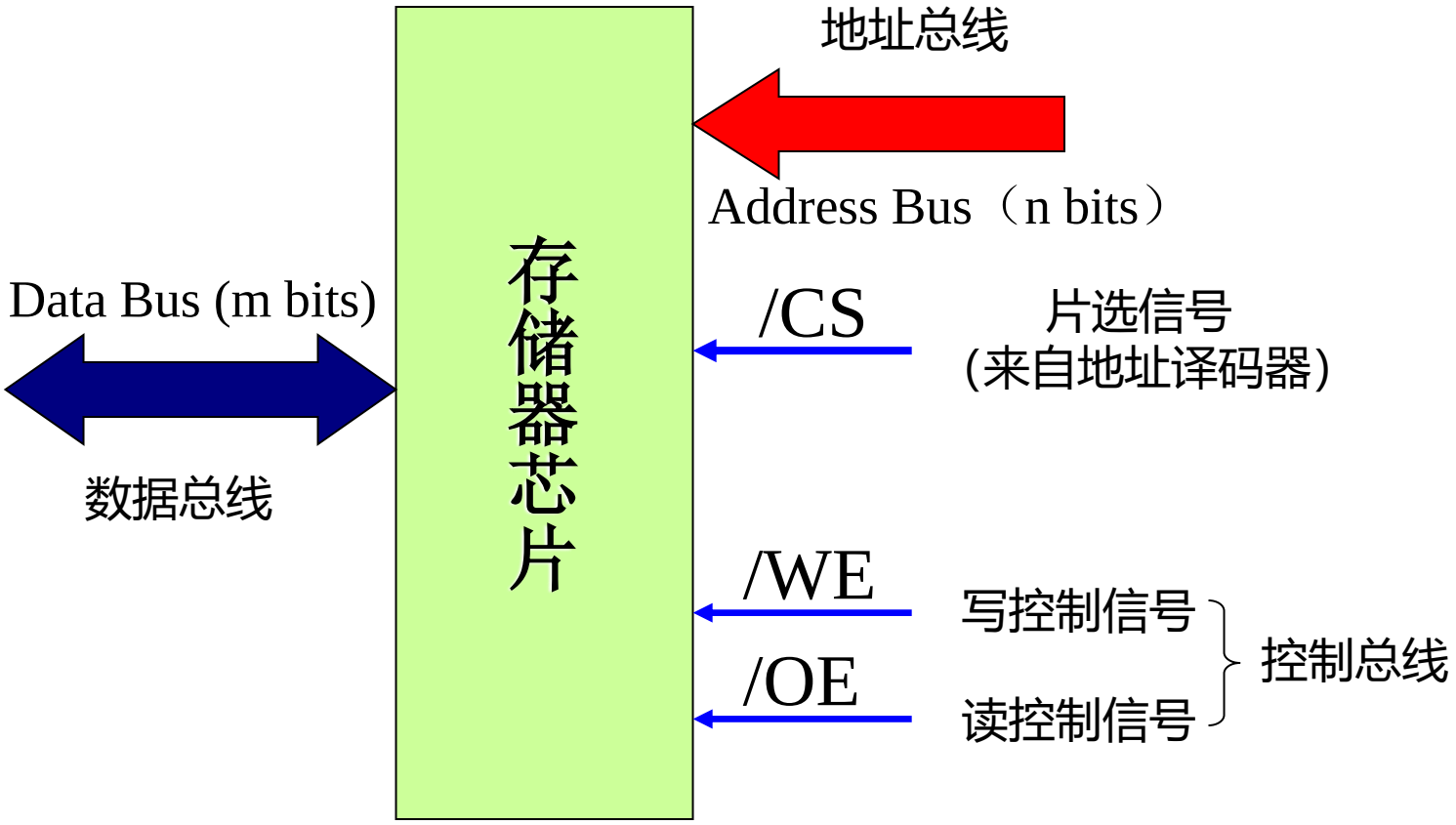






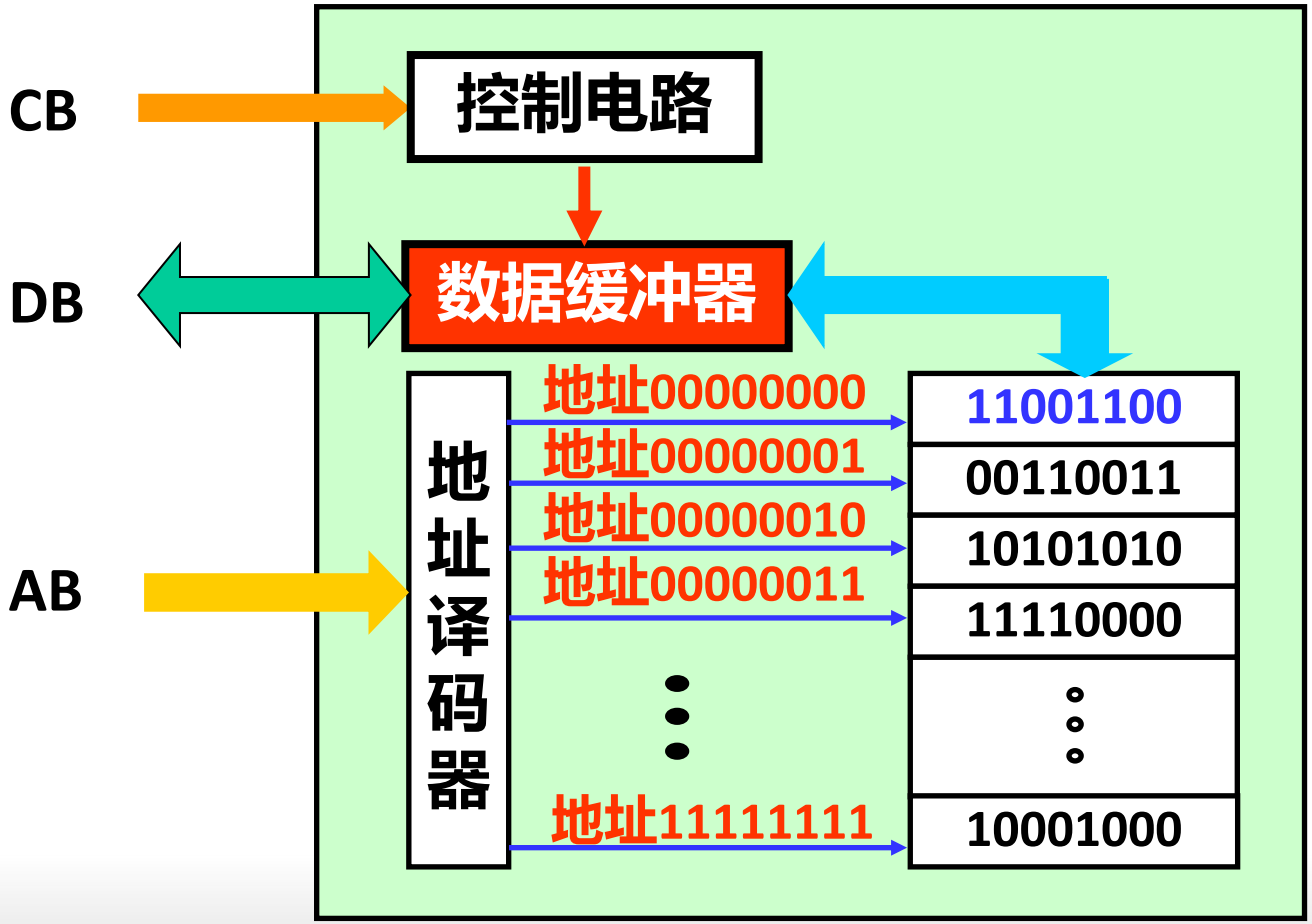
- Memory (idealized)
  - One input bus: Data In
  - One output bus: Data Out
- Memory word is selected by:
  - Write Enable = 1: address selects the memory data to be written via the Data In bus
  - Address selects the data to put on Data Out
- Clock input (CLK)
  - The CLK input is a factor ONLY during write operation
  - During read operation, behaves as a combinational logic block:
    - Address valid => Data Out valid after “access time.”

存储器总线连接

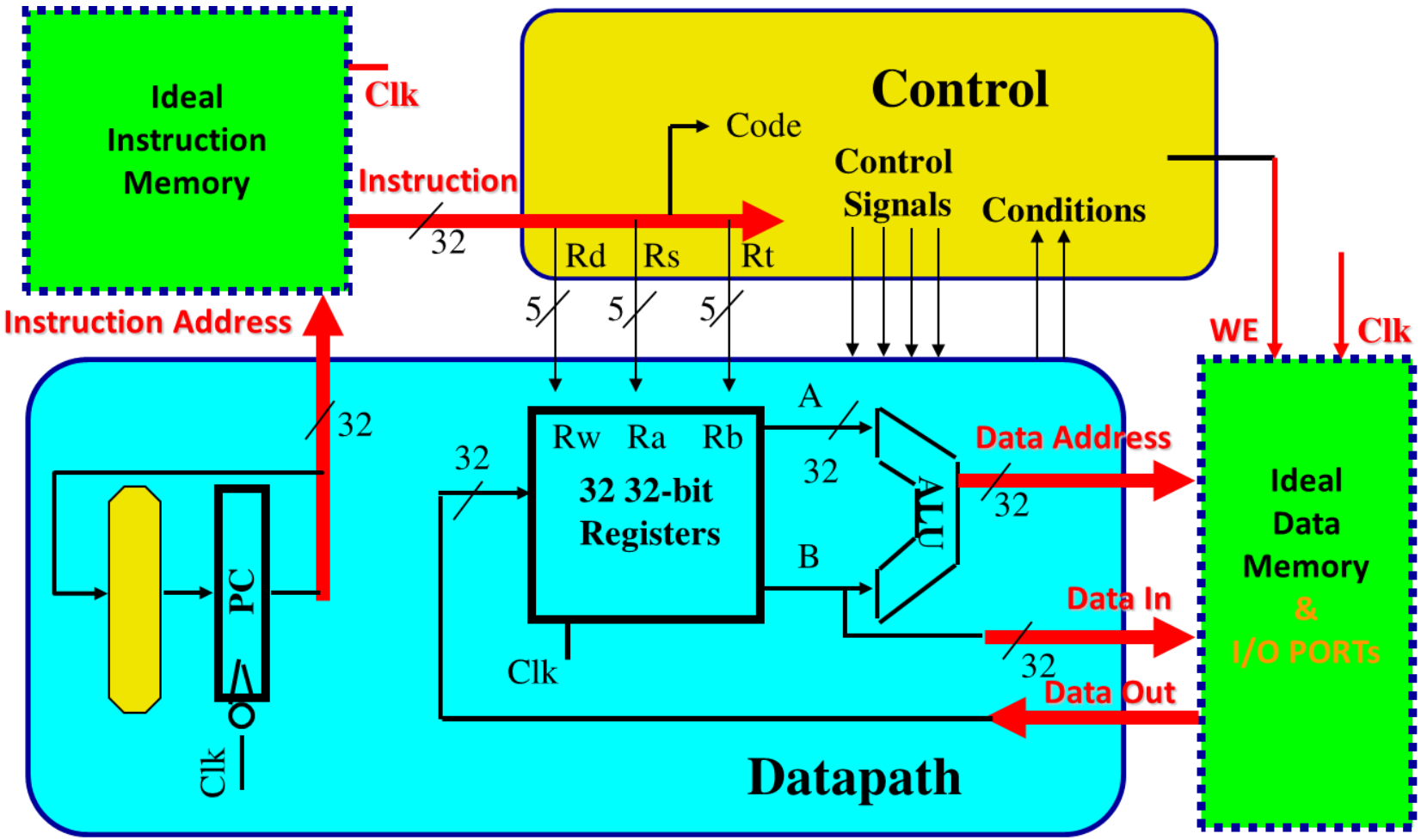

地址总线
Address Bus（n bits）
Data Bus (m bits)
存储器芯片
/CS
片选信号
(来自地址译码器)
数据总线
/WE
写控制信号
/OE
读控制信号
控制总线

计算机逻辑电路模块

存储器芯片结构：



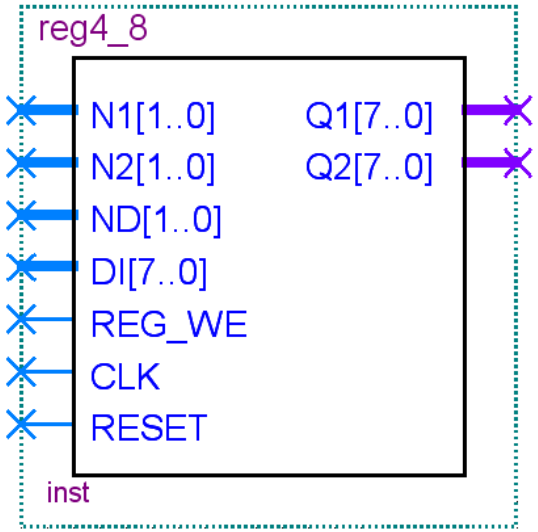
计算机逻辑电路模块：I/O接口



实验：简单计算机系统基本模块设计

简单计算机系统基本模块设计

1）寄存器组模块：



输入信号：

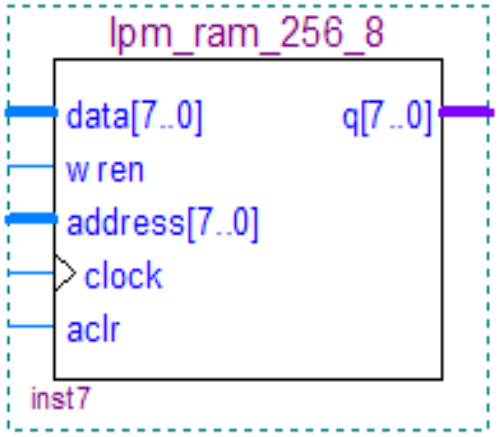
N1[1..0]：读通道1的寄存器号
N2[1..0]：读通道2的寄存器号
ND[1..0]：写通道的寄存器号
DI[7..0]：写通道的输入数据
CLK：时钟脉冲信号，上升沿有效
REG_WE：写允许，为1时，在CLK上升沿，将数据DI写入ND指定的寄存器；为“0”时，禁止对寄存器组进行写操作
RST： 异步复位信号，清空所有寄存器的内容

输出信号：

Q1[7..0]：输出N1[1..0]指定寄存器的内容
Q2[7..0]：输出N2[1..0]指定寄存器的内容

简单计算机系统基本模块设计

2）RAM数据存储系统模块：



输入信号：

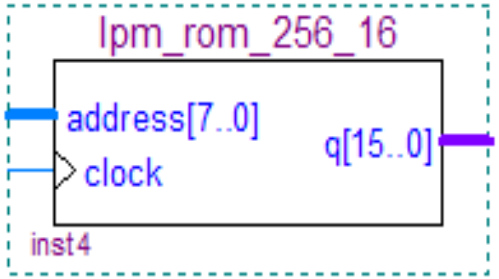
address[7..0]：需要进行读/写操作的RAM单元地址
data[7..0]：待写入RAM中的数据
wren：写允许信号，为1时写；为0则读
clock：时钟脉冲信号，上升沿将数据写入或读出
aclr： 异步复位信号，对q[7..0]进行清0

输出信号：

q[7..0]：输出address [7..0]指定的RAM单元的内容

简单计算机系统基本模块设计

3）ROM程序存储模块：



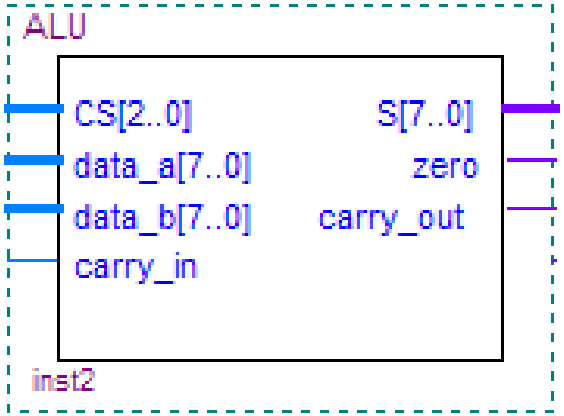
输入信号：
address[7..0]：ROM单元的地址
clock：时钟脉冲信号，在时钟的上升沿，将选中的ROM单元内容输出

输出信号：
输出address [7..0]指定的ROM单元的内容

实验说明：
1）设计ROM模块过程中会提示是否要指定ROM模块的初始化内容，选择“是”，并且指定一个用于存放程序的16进制编码格式文件，可以加载自定义的代码程序
2）ROM模块的检测，可将8个拨码开关接address，按键rst接clock，q[15~8]送到数码管显示，q[7..0]接到8个LED灯上，根据数码管上的显示进行检测

简单计算机系统基本模块设计

4）8位算术逻辑运算器ALU模块：



CS[2..0]	运算名称	运算操作
	与	S[7..0] = data_a[7..0] AND data_b[7..0]
	或	S[7..0] = data_a[7..0] OR data_b[7..0]
	不带进位加	S[7..0] = data_a[7..0] + data_b[7..0]
	不带借位减	S[7..0] = data_a[7..0] - data_b[7..0]
	带进位加	S[7..0] = data_a[7..0] + data_b[7..0] + Carry_in
	带借位减	S[7..0] = data_a[7..0] - data_b[7..0] – (1-Carry_in)
	比较	If data_a[7..0]<data_b[7..0], S[7..0]=1; else S[7..0]=0

输入信号：

data_a[7..0]、data_b[7..0]：参与运算的两个8位二进制数

CS[2..0]：存放7种运算操作的编码，编码规则自定

Carry_in: 参与运算的进/借位值

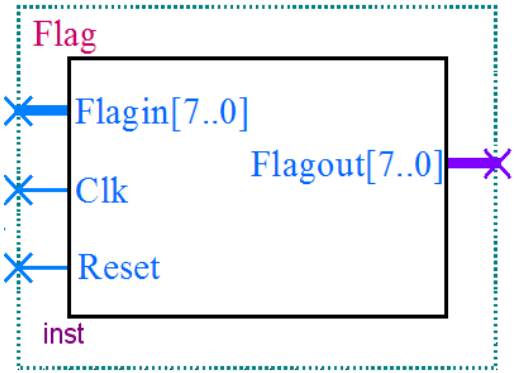
输出信号：

S[7..0]：操作后的8位结果

Zero：零标志， 当s[7..0]为0时，zero=1; 否则zero=0；

Carry_out: 进位/借位标志，当加法运算过程最高位有进位时，Carry_out=1；否则Carry_out=0；当减法运算最高位产生借位时，Carry_out=0；否则，Carry_out=1

5）标志寄存器模块：



输入信号：

Flagin[7..0]：写入标志寄存器的数据，如Carry、Zero等信息，每个标志占1位，共2位，其他6位暂时无用。

CLK：写时钟脉冲信号，上升沿时，将Flagin上的数据写入Flagout中

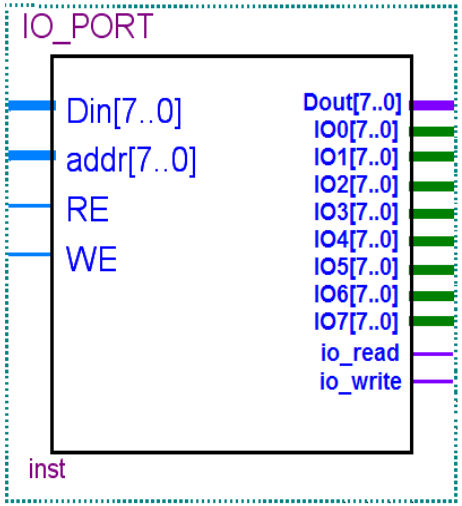
Reset：异步复位信号，对标志寄存器内容清零 flagout[7..0]=0

输出信号：

flagout[7..0]：标志寄存器的数据

简单计算机系统基本模块设计

6）I/O端口及其映射模块：



输入信号：

addr[7..0]：地址输入信号

Din[7..0]：写入I/O端口中的数据信号

RE： 读控制信号，高电平有效

WE： 写控制信号，高电平有效

输出信号：

Dout[7..0]：读I/O端口时输出的数据信号

io_read： I/O端口读控制信号，高电平有效，读0x00~0x07存储单元时有效

io_write：I/O端口写控制信号，高电平有效，写0x00~0x07存储单元时有效

输入/输出信号

IO0[7..0]~ IO7[7..0]：双向I/O端口0~端口7的数据信号

课后作业：

复习：

《计算机组成原理》第7章《指令系统》、第4章《组合逻辑电路》

预习：

《计算机组成原理》第8章《中央处理器》

思考题：

《计算机组成原理》 P188页习题 7.6、7.8、7.9、7.10、7.13题

谢 谢

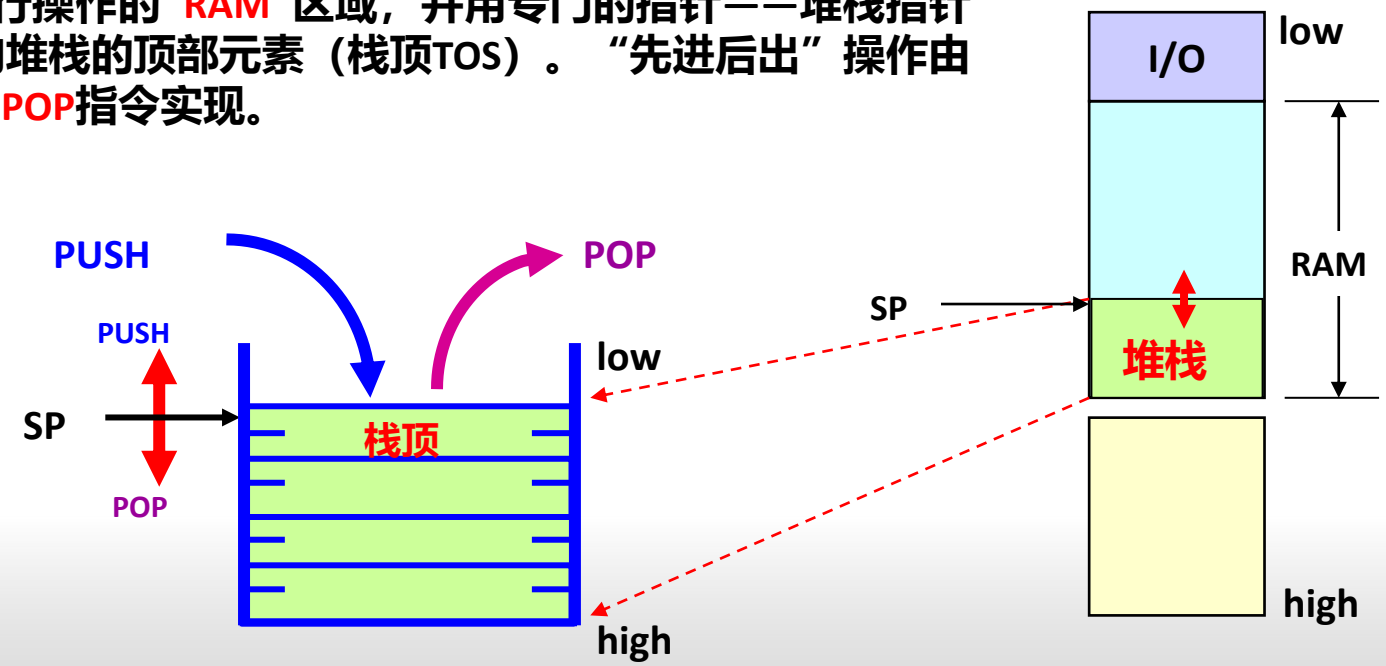
请不要将课件上传到公共网络平台上~~

寻址方式

隐含寻址

- 指令中不给出操作数地址，通常操作数约定在某个特定的或者堆栈中
- 示例：push、pop

堆栈（stack）是一片以“先进后出/后进先出”（FILO/LIFO）方式进行操作的 RAM 区域，并用专门的指针——堆栈指针 SP 指向堆栈的顶部元素（栈顶TOS）。“先进后出”操作由PUSH和POP指令实现。

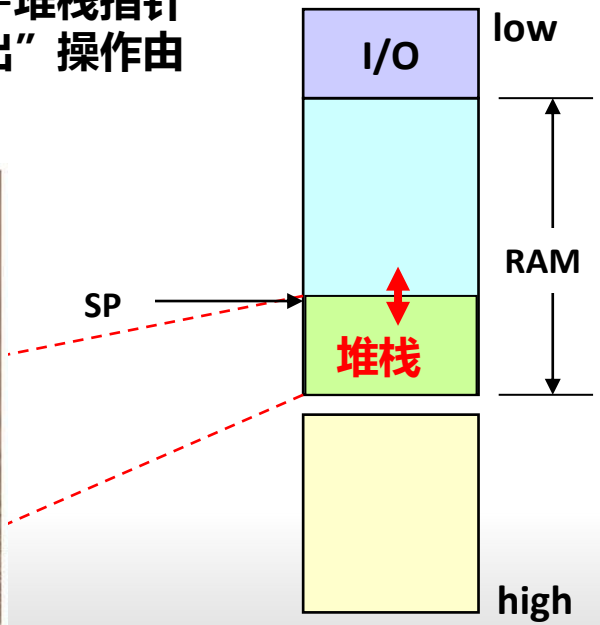


寻址方式

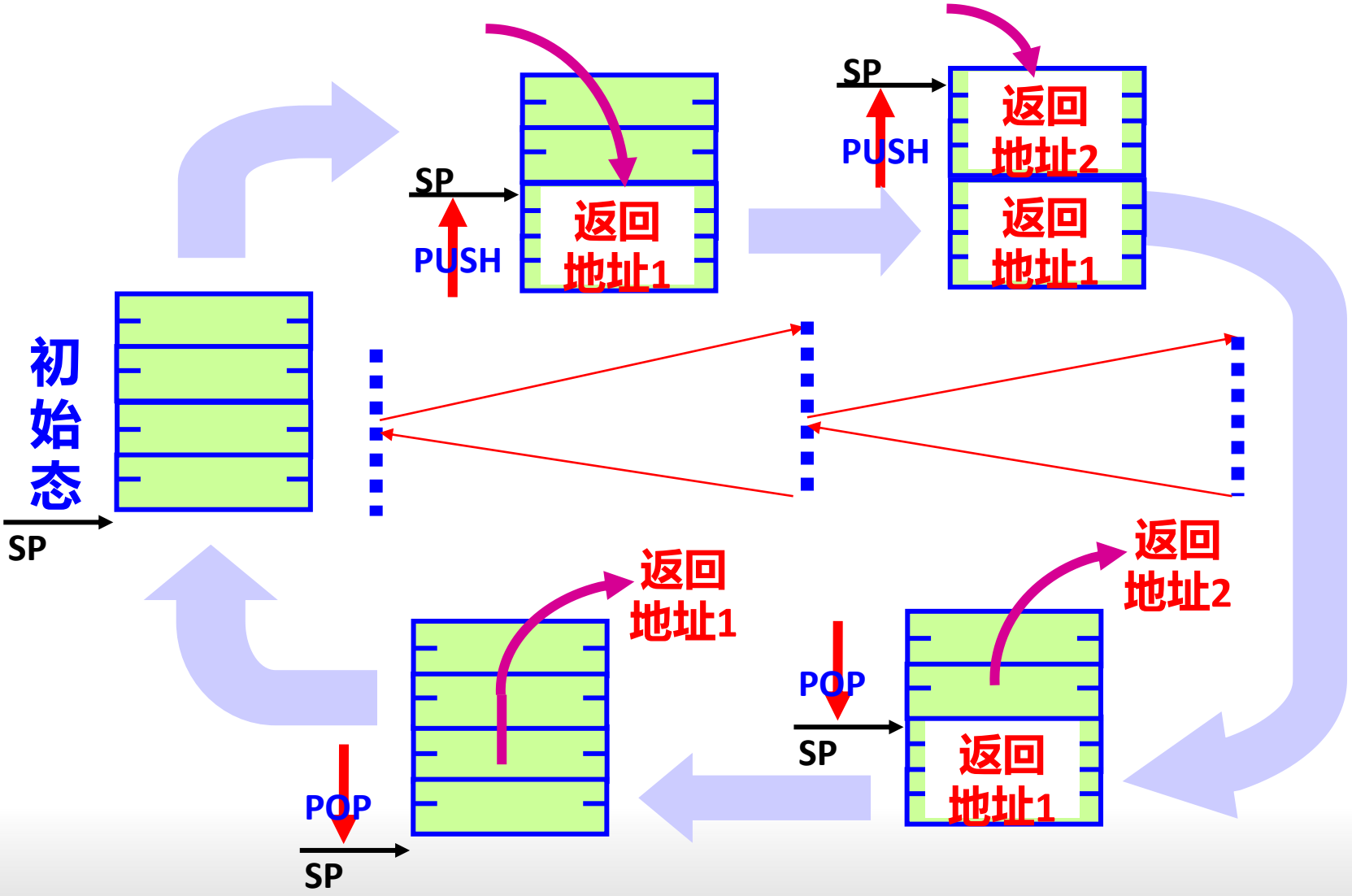
隐含寻址

- 指令中不给出操作数地址，通常操作数约定在某个特定的或者堆栈中
- 示例：push、pop

堆栈（stack）是一片以“先进后出/后进先出”（FILO/LIFO）方式进行操作的 RAM 区域，并用专门的指针——堆栈指针 SP 指向堆栈的顶部元素（栈顶TOS）。“先进后出”操作由PUSH和POP指令实现。



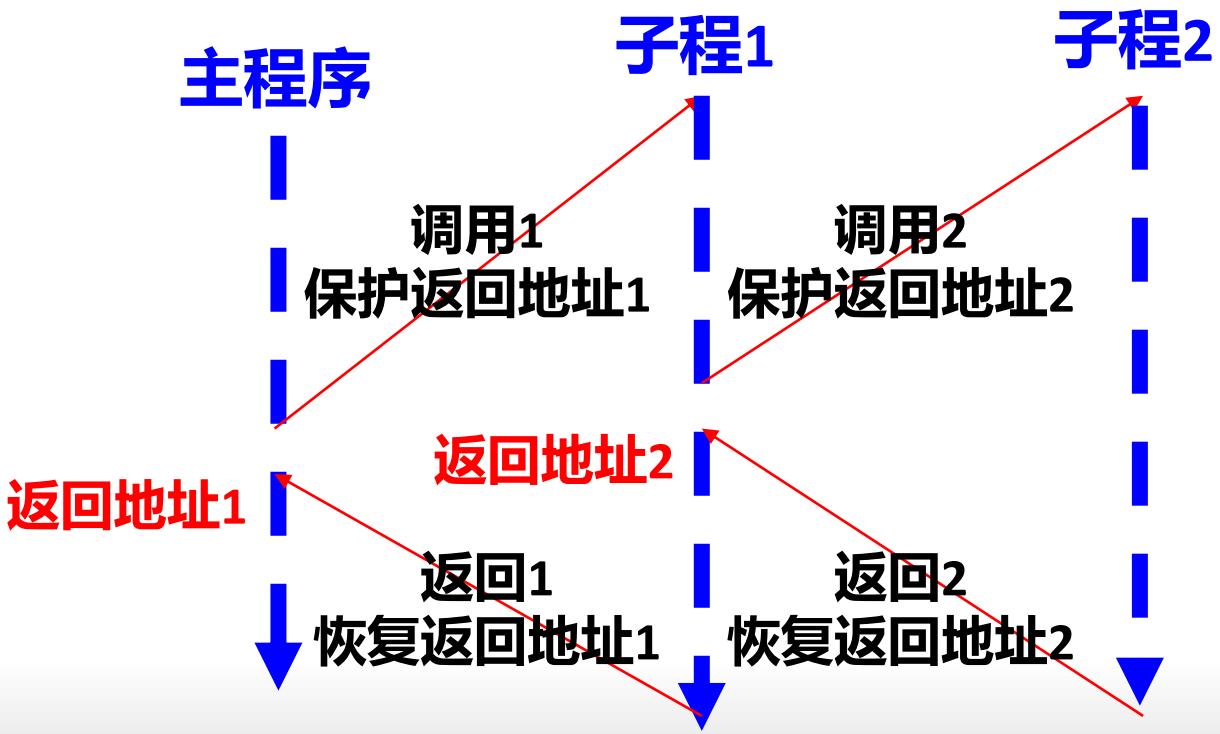
寻址方式

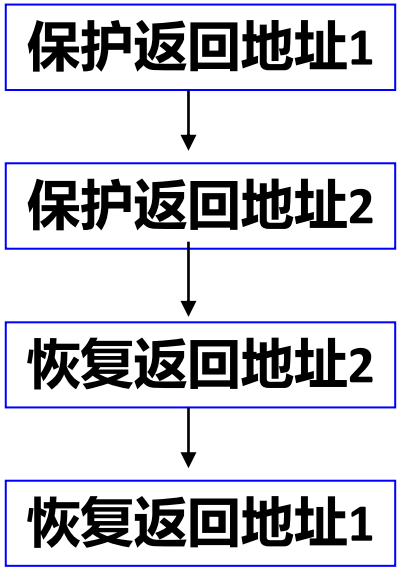

初始态
SP
SP
PUSH
返回地址1
SP
PUSH
返回地址2
返回地址1
返回地址2
POP
SP
返回地址1
返回地址1
POP
SP

寻址方式

堆栈用途：

1. 普通子程和中断子程的现场和返回地址的保存和恢复
2. C 语言程序中局部变量、形参的保存和释放





先进后出