

计算机组成原理

中央处理器 (3)

王浩宇,教授

haoyuwang@hust.edu.cn

<https://howiepku.github.io/>

本节目录

- 微程序控制器
- 微程序设计
- 微指令格式
- 单周期MIPS CPU
- 多周期MIPS CPU

微程序控制器

1

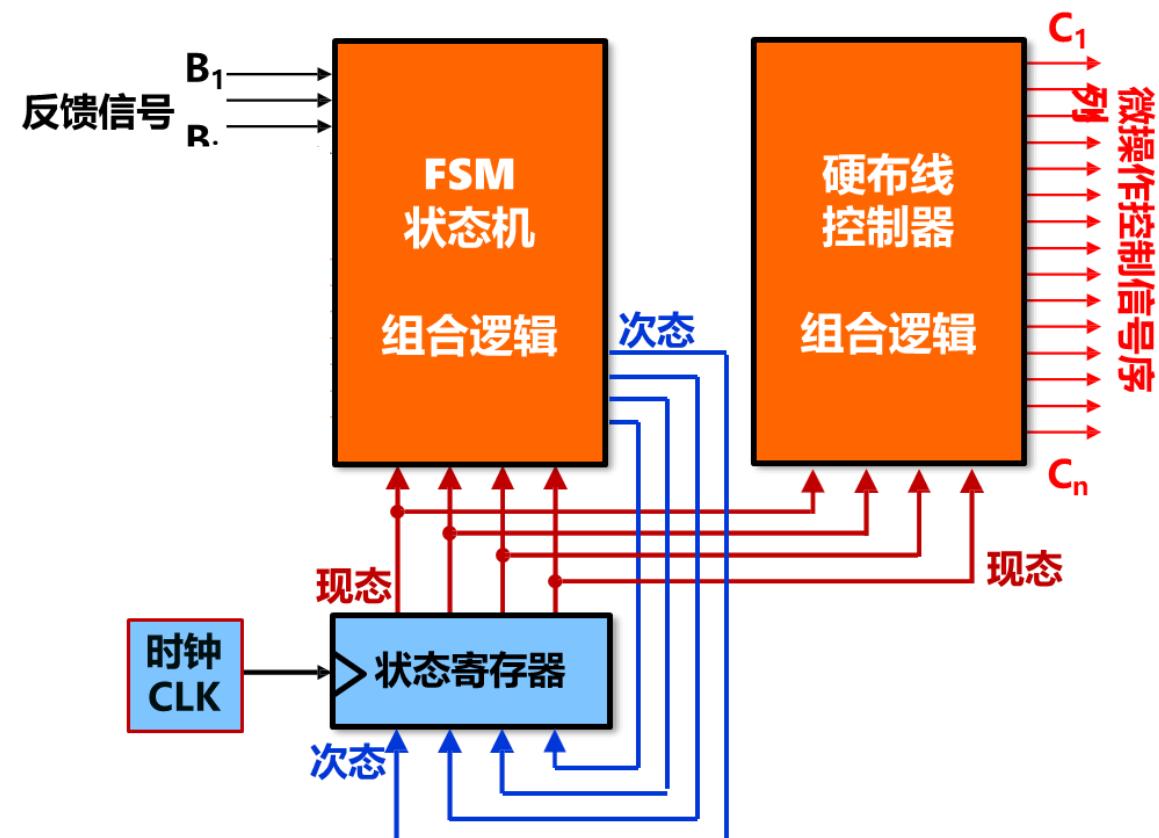
微程序控制器基本思想

■ 硬布线: 同步逻辑、繁, 快, 贵, 难改

- ◆ 一条指令多个时钟周期
- ◆ 一个时钟周期一个状态
- ◆ 一个状态对应一组并发信号

■ 微程序: 存储逻辑、简、慢、廉, 易改

- ◆ 将并发信号事先存储为微指令
- ◆ 一条指令对应多条微指令
- ◆ 状态等同与存储器地址



机器指令字 → 控制器信号序列

微程序控制器工作原理

- 微程序是利用软件方法来设计硬件的技术
- 将完成指令所需的控制信号按格式编写成微指令，存放到控制存储器
 - ◆ 一条机器指令对应一段微程序（多条微指令）
 - ◆ 指令取指执行 → 微程序的执行 → 执行多条微指令 → 依次生成控制信号
- 存储技术和程序设计相结合，回避复杂的同步时序逻辑设计

基本概念—微命令和微操作

■ 微命令

- 控制部件向执行部件发出的各种控制命令叫作微命令，它是构成控制序列的最小单位。
- 例如：打开或关闭某个控制门的电位信号、某个寄存器的打入脉冲等。微命令是控制计算机各部件完成某个基本微操作的命令

■ 微操作

- 执行部件接受微命令后进行的操作，最基本的操作。
- 微命令和微操作是一一对应的。
- 微命令是微操作的控制信号，微操作是微命令的操作过程。
微操作是执行部件中最基本的操作。

基本概念—微命令和微操作

- ~~微命令是控制信号最小，最基本的单位~~
 - 微命令带来的执行部件的动作称为微操作
-
- 由于数据通路的结构关系，微操作可分为相容的和互斥的两种：
 - 互斥的微操作：
 - 不能同时或不能在同一个节拍内并行执行的微操作，例如：+、-、M
 - 相容的微操作
 - 能够同时或在同一个节拍内并行执行的微操作

基本概念—微指令和微程序

- **微指令：**在机器的一个CPU周期中，一组实现一定操作功能的微命令的组合，构成一条微指令
- **把在同一CPU周期内并行执行的微操作控制信息，存储在控制存储器里，称为一条微指令**
 - 一条微指令通常至少包含两大部分信息：
 - **操作控制字段**，又称微操作码字段，用以产生某一步操作所需的各个微操作控制信号。
 - 某位为1，表明发微命令
 - 微指令发出的控制信号都是节拍电位信号，持续时间为一个CPU周期
 - 微命令信号还要引入时间控制
 - **顺序控制字段**，又称微地址码字段，用以控制产生下一条要执行的微指令地址。

基本概念—微指令和微程序

微程序

- 一系列微指令的有序集合就是微程序。
 - 一段微程序对应一条机器指令。
 - 微地址：存放微指令的控制存储器的单元地址

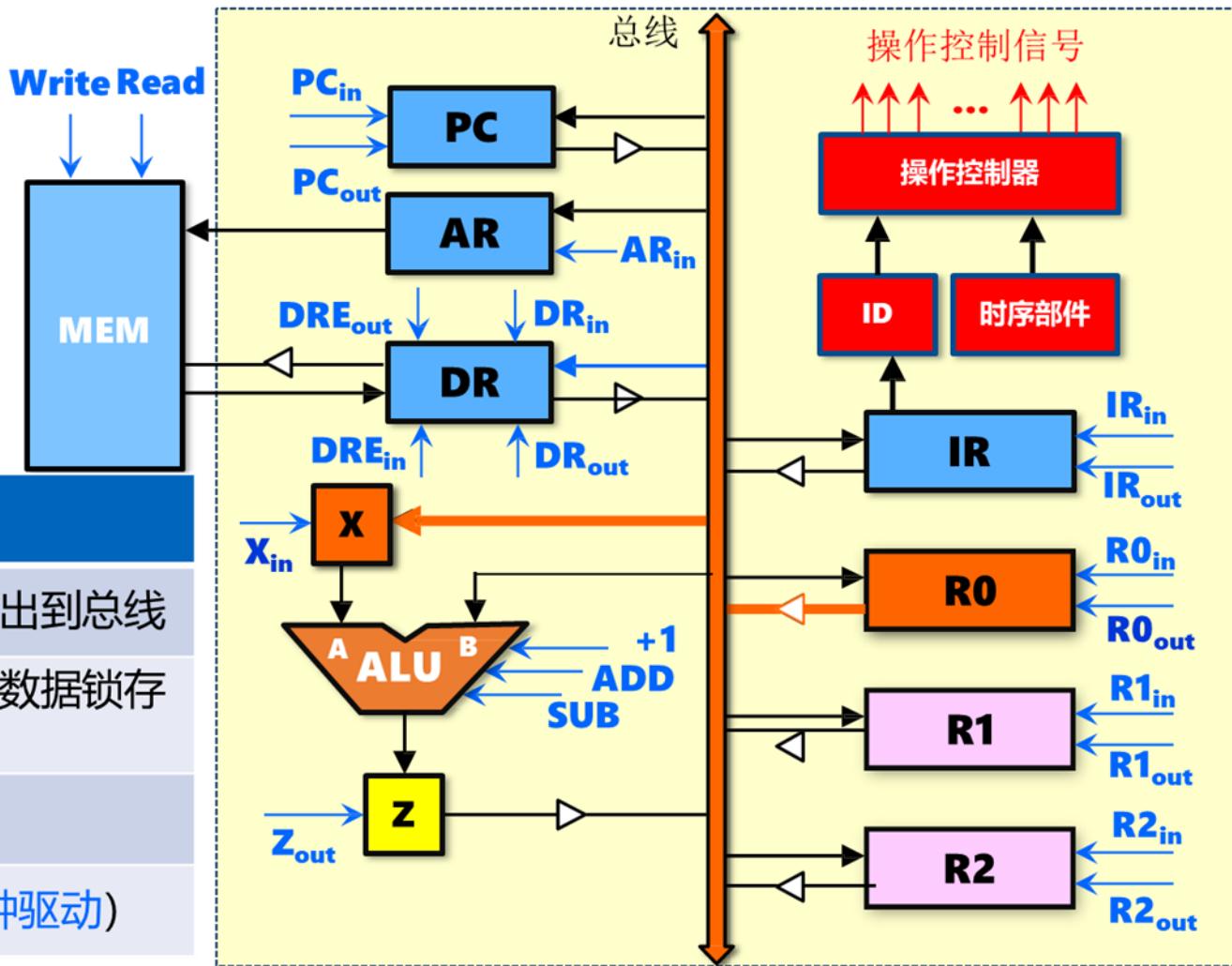
微命令->微指令->微程序

微程序控制器

单总线结构CPU

- 主要部件都连接在总线上
- 各部件间通过总线进行传输

控制信号	作用说明
$IR_{out}, PC_{out}, \dots R1_{out}$	控制三态门将寄存器值输出到总线
$IR_{in}, PC_{in}, \dots R1_{in}$	控制寄存器使能端将总线数据锁存 (时钟驱动)
+1、ADD、SUB	运算控制信号
Write、Read	内存读写控制信号 (时钟驱动)



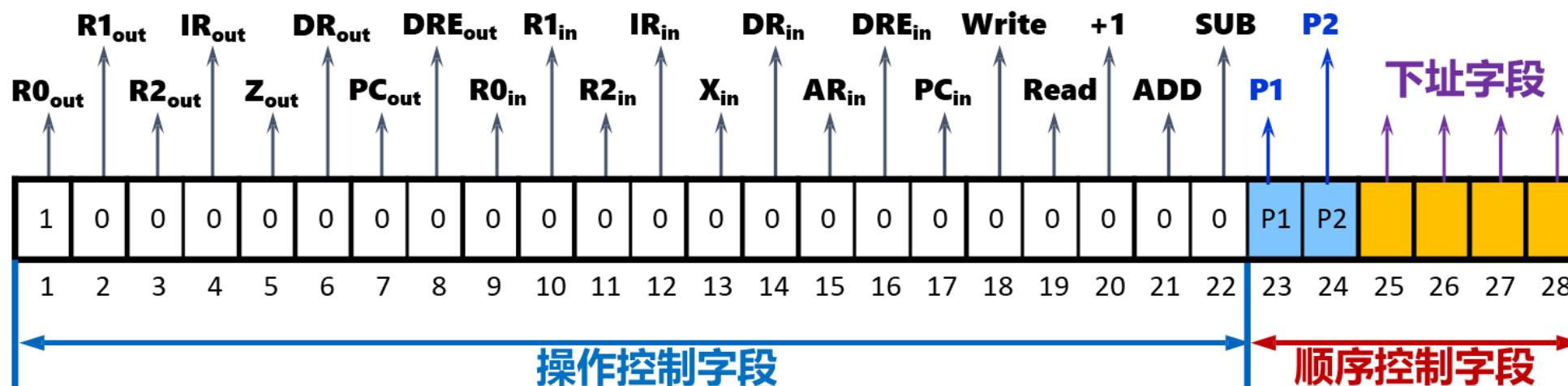
单总线CPU微指令构造

■ 操作控制字段： 存储操作控制信号

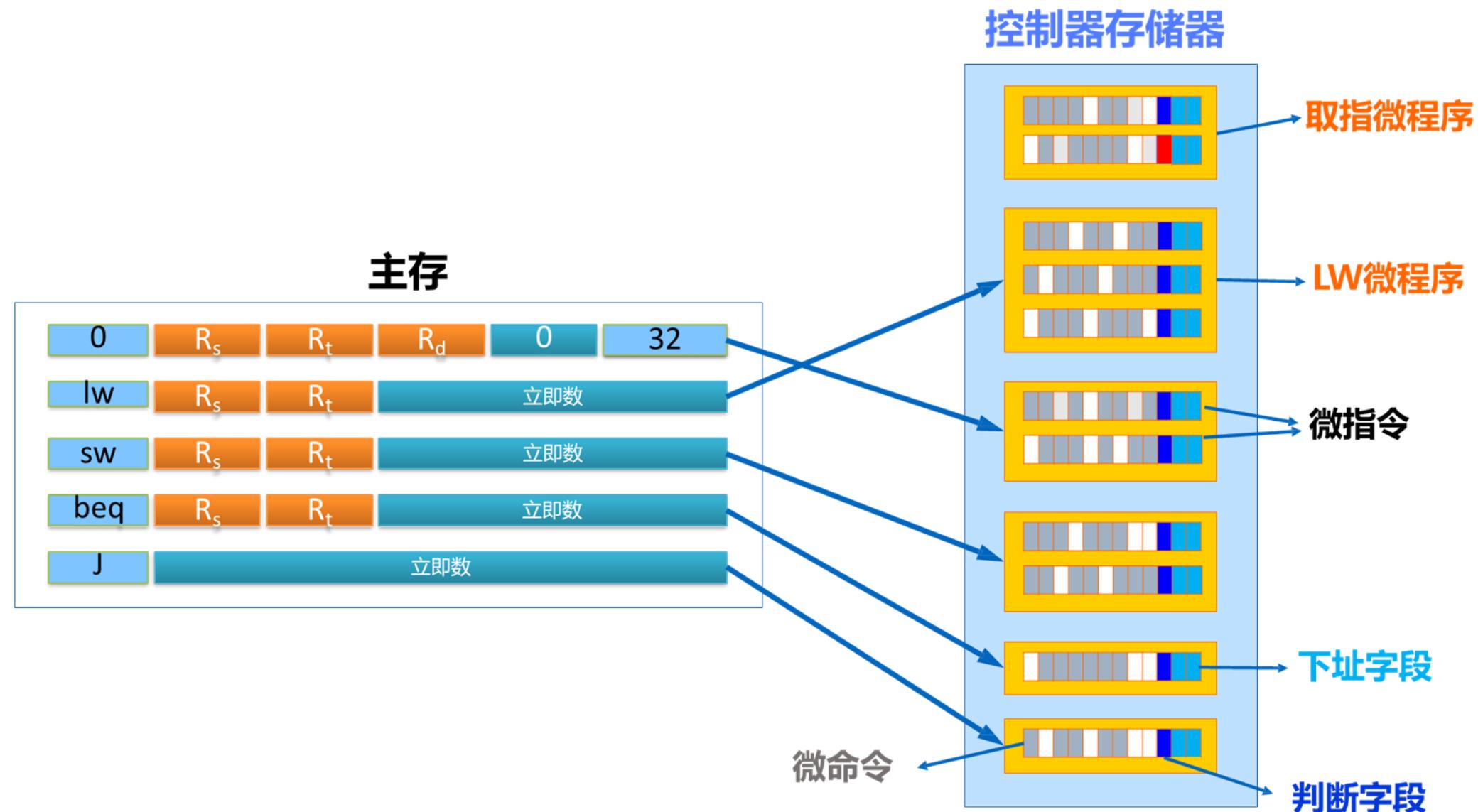
◆ 每一位对应一个控制信号，也称微命令，可同时给出多个操作信号

■ 顺序控制字段： 用于控制微程序的执行顺序

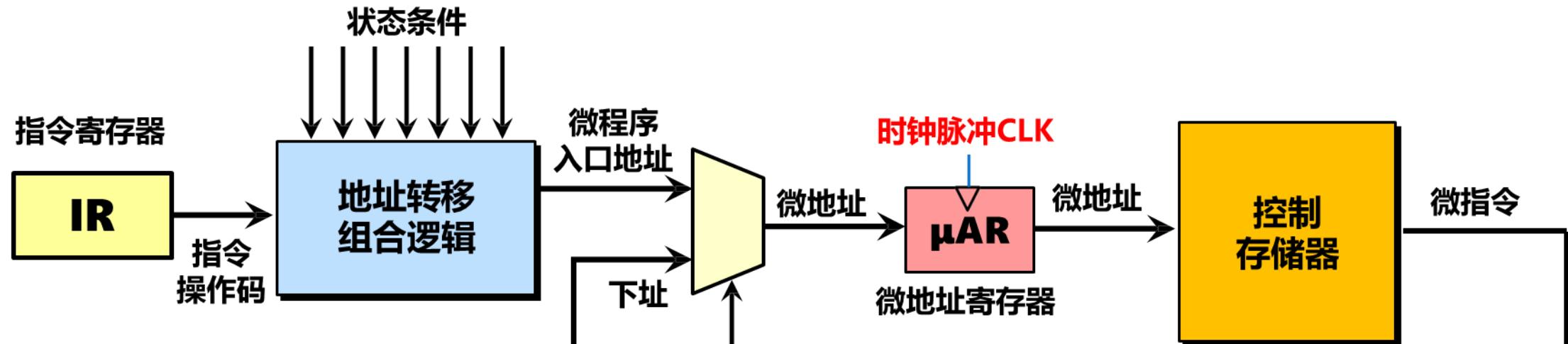
◆ 判别逻辑为零，下一条微指令地址从下址字段获取，否则按约定规则生成



程序、微程序、指令、微指令对应关系



微程序控制器组成原理框图（下址字段，断定法）

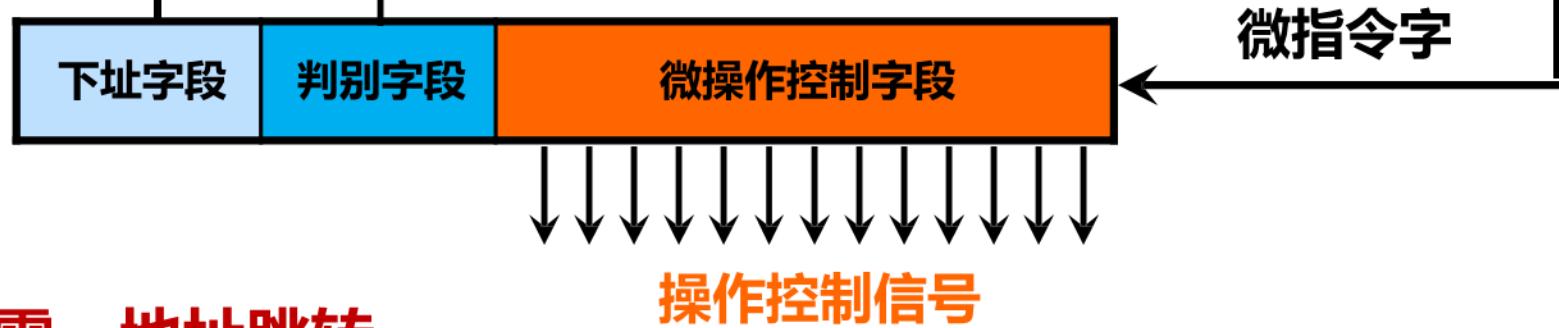


■ 初始化: $\mu AR=0$

■ 0号地址为取指微程序入口

■ 顺序执行取指微程序

■ 最后一条微指令判别字段非零, 地址跳转



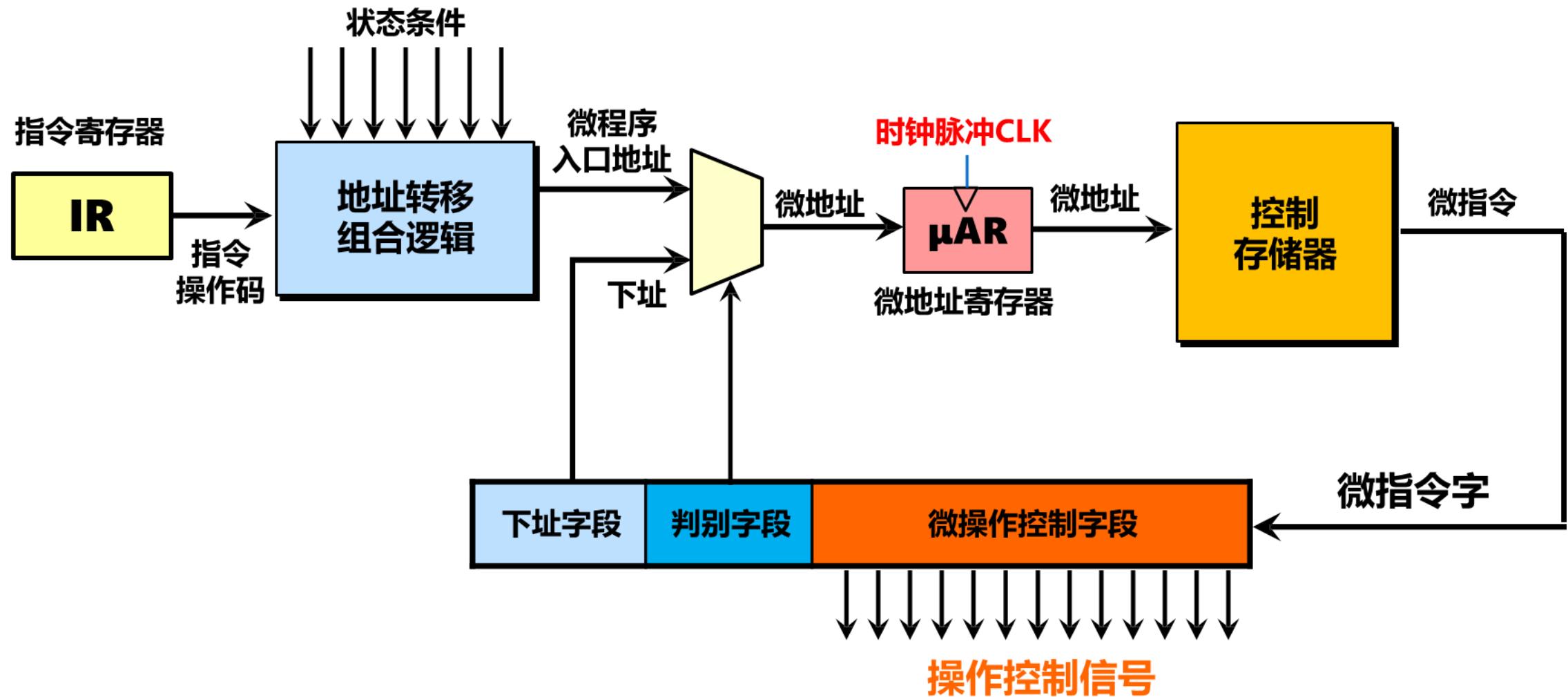
微程序优势与劣势

- 速度慢 访存频繁、成本低廉
- 设计规整，设计简单，易于修改、扩展指令系统功能
 - ◆ 适合**CISC**等功能较复杂的系列机 **X86、IBM S/360、DEC VAX**
 - ◆ 可写控存方便修复出厂故障 **Intel Core 2、Intel Xeon**
- 硬布线控制器执行速度快，但设计复杂，代价昂贵，不便于修改
 - 适合**RISC**计算机，如**MIPS、ARM**

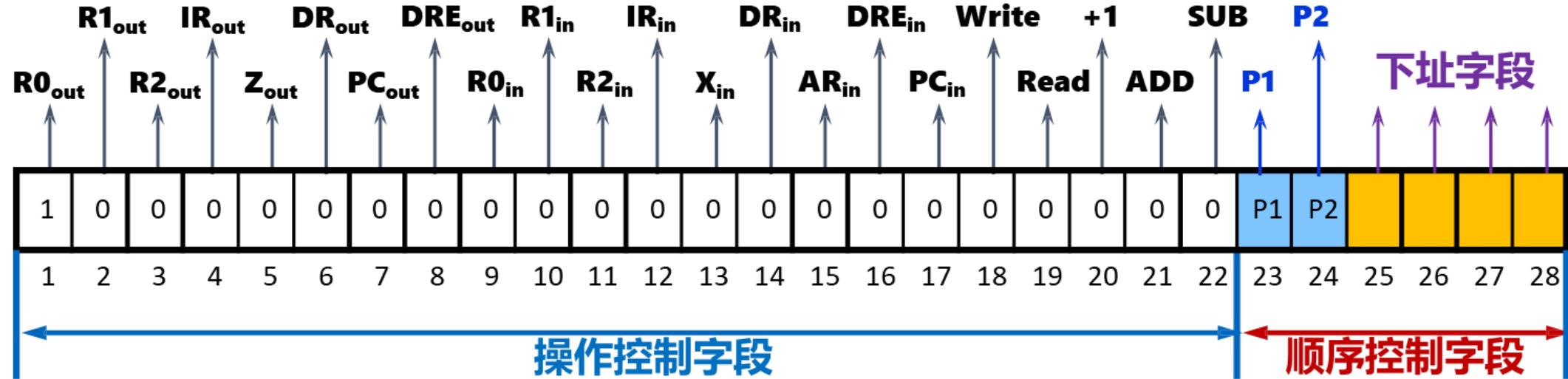
本节目录

- 微程序控制器
- 微程序设计
- 微指令格式
- 单周期MIPS CPU
- 多周期MIPS CPU

微程序控制器组成原理框图



微指令格式



- 一条微指令对应一个时钟周期
- 微指令操作控制字段的信号在该时钟周期内有效
- 指令需要多少时钟周期就包括多少微指令

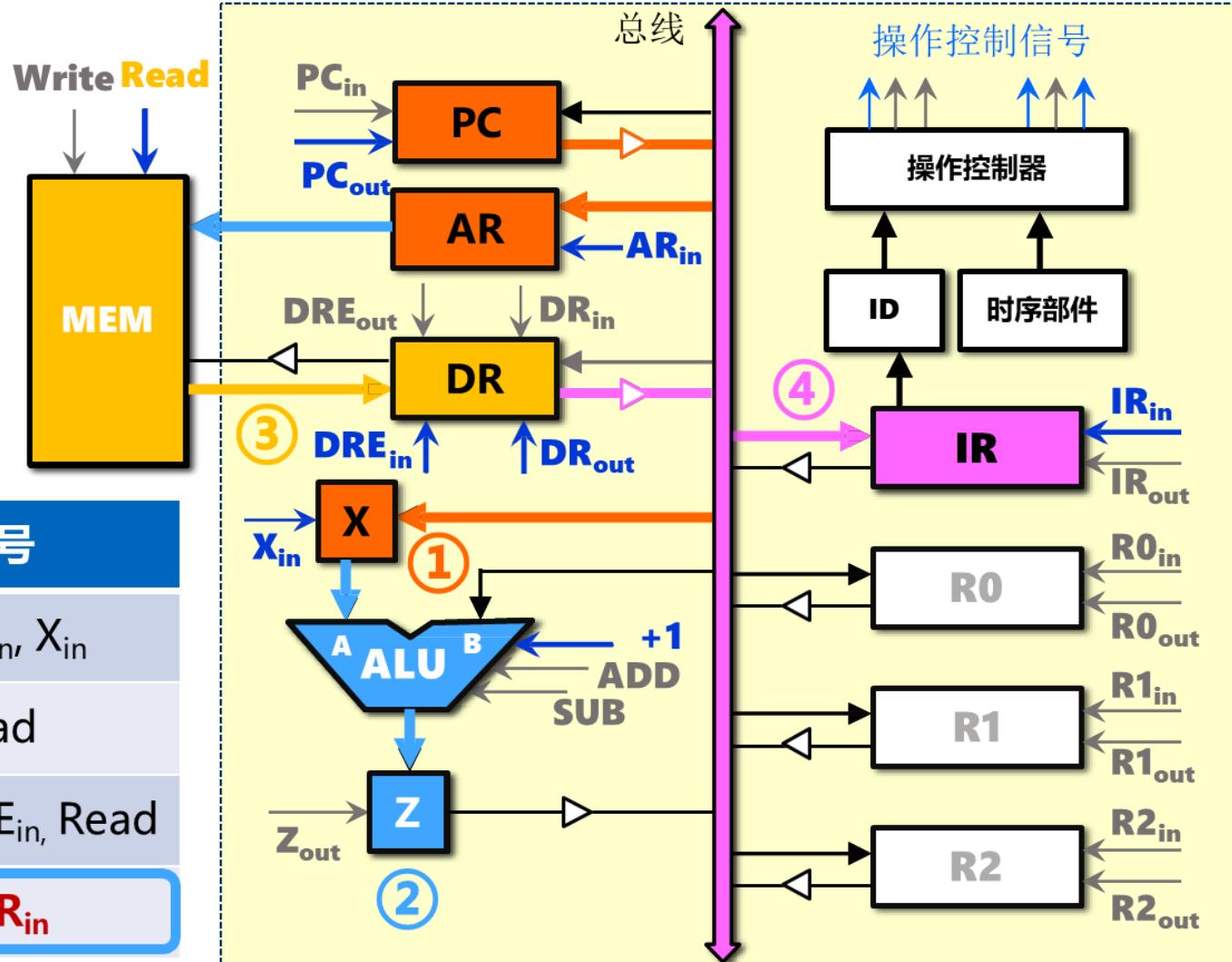
微程序设计

取指令数据通路

Mem[PC++] → IR

- 4个时钟周期
- 四条微指令

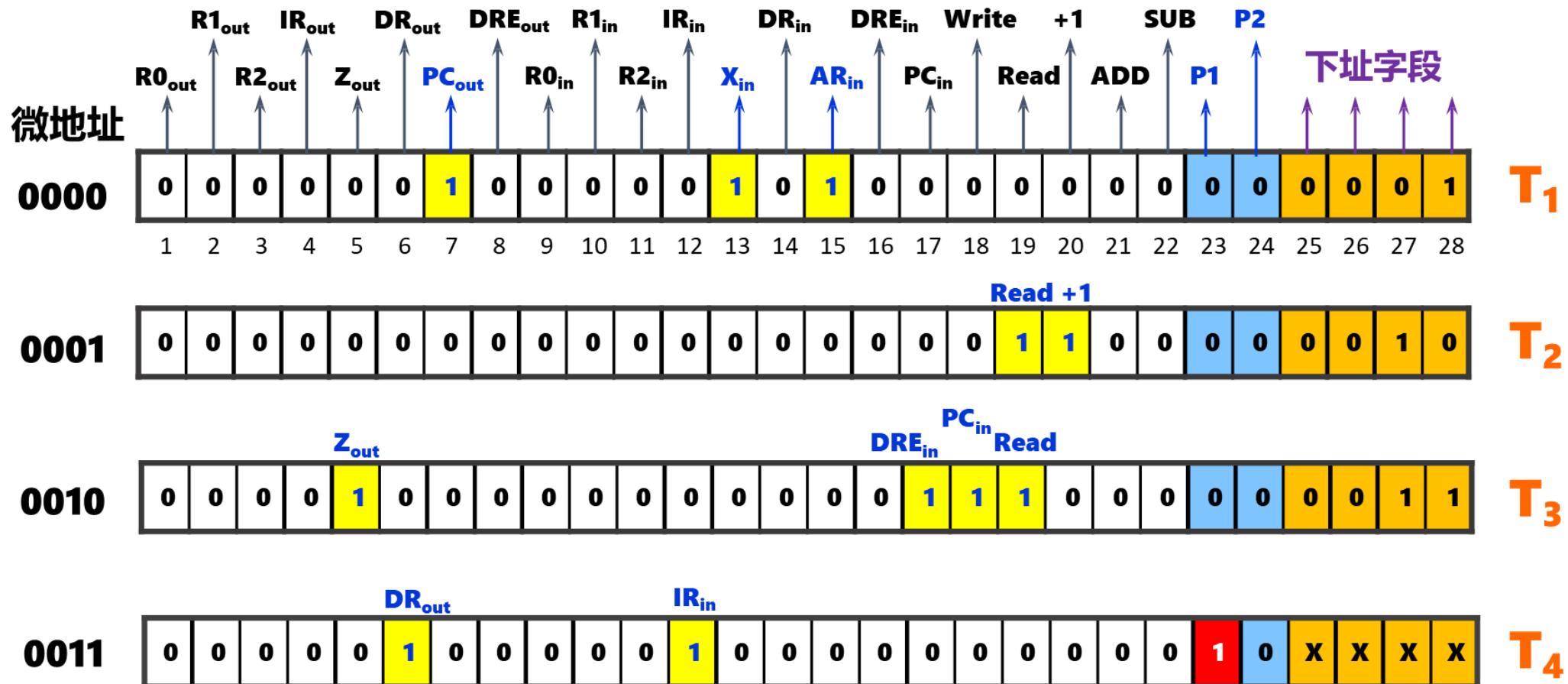
节拍	数据通路	控制信号
T1	(PC)→AR, (PC)→X	PC _{out} , AR _{in} , X _{in}
T2	(X)+1→Z	+1, Read
T3	(Z)→PC, Mem[AR]→DR	Z _{out} , PC _{in} , DRE _{in} , Read
T4	(DR)→IR	DR_{out}, IR_{in}



微程序设计

取指令微程序

节拍	取指令数据通路	控制信号
T1	(PC)→AR, (PC)→X	PC _{out} , AR _{in} , X _{in}
T2	(X)+1→Z	+1, Read
T3	(Z)→PC, Mem[AR]→DR	Z _{out} , PC _{in} , DRE _{in} , Read
T4	(DR)→IR	DR _{out} , IR _{in}



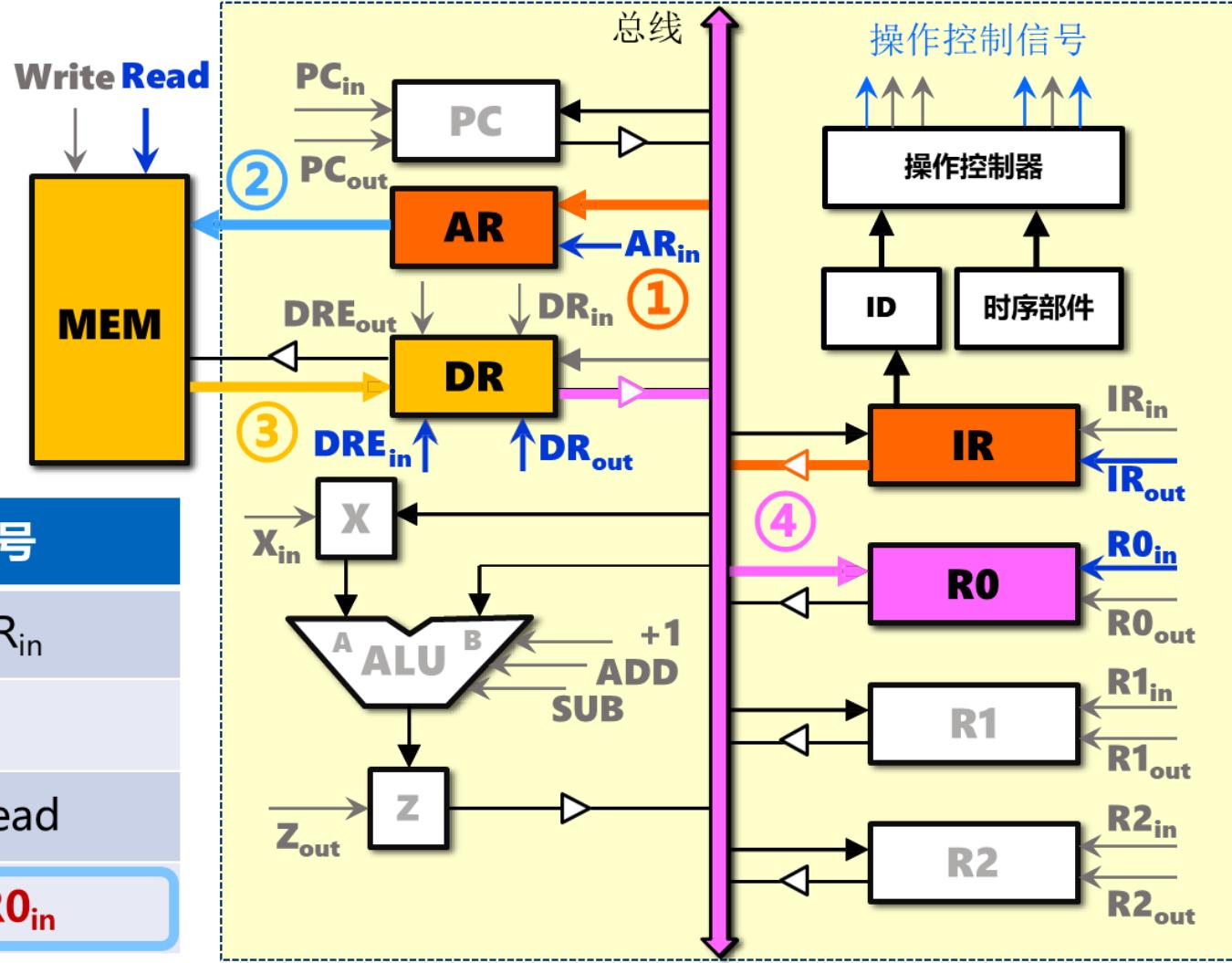
微程序设计

LOAD指令执行数据通路

LOAD R0,6#

Mem[IR_A] → Reg

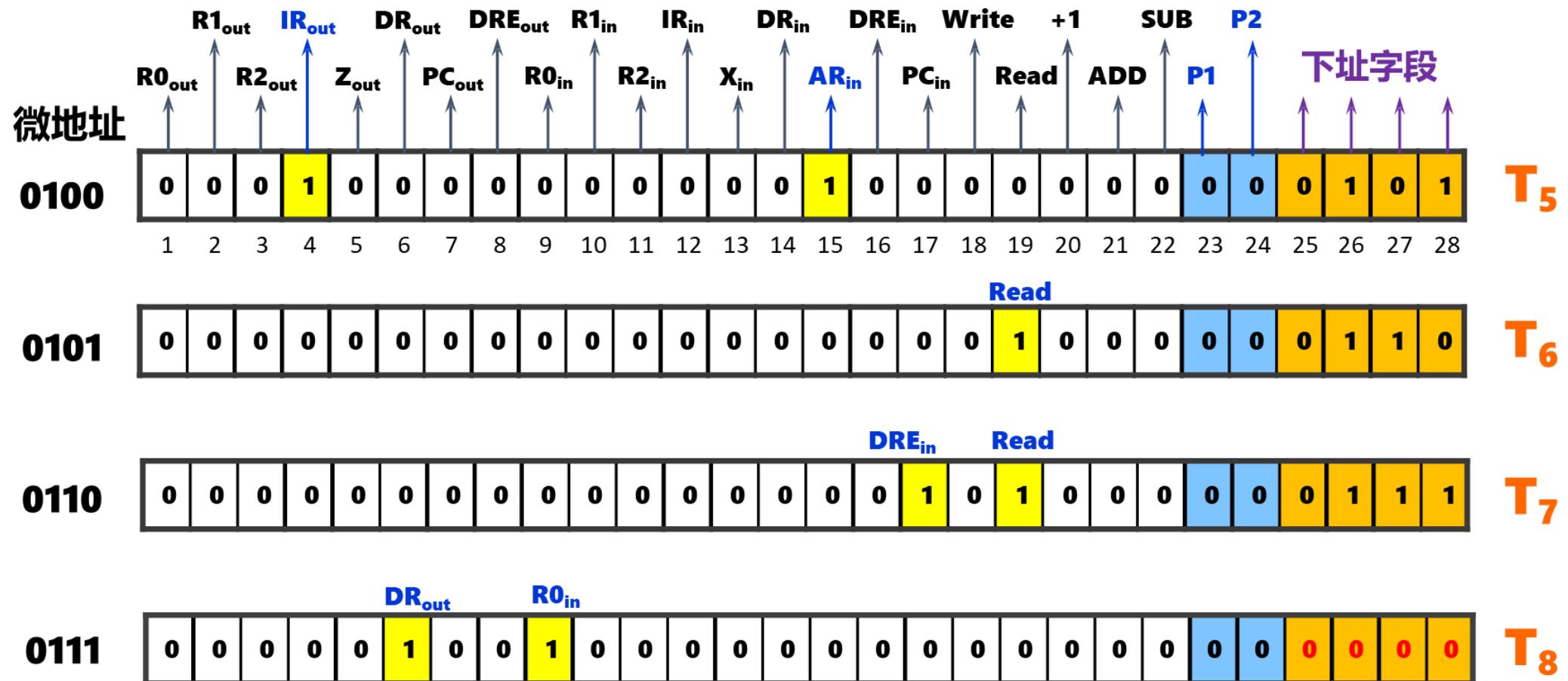
节拍	数据通路	控制信号
T1	(IR _A)→AR, (PC)→X	IR _{out} , AR _{in}
T2		Read
T3	Mem[AR]→DR	DRE _{in} , Read
T4	(DR)→R0	DR_{out}, R0_{in}



微程序设计

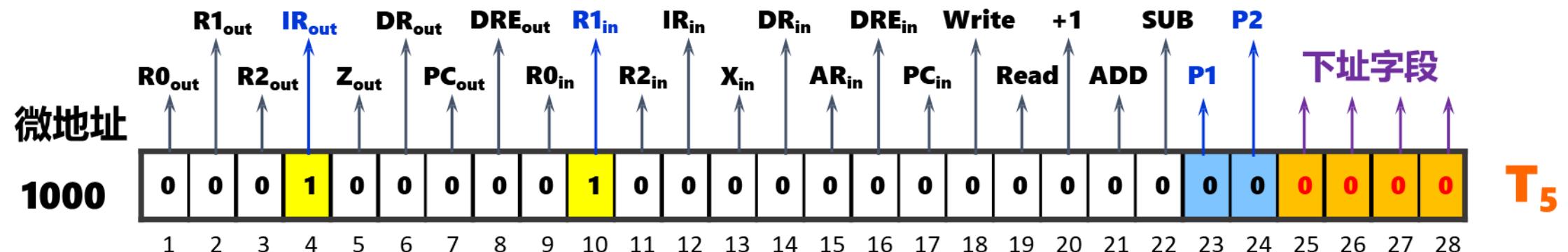
LOAD指令微程序

节拍	取指令数据通路	控制信号
T1	$(IR_A) \rightarrow AR, (PC) \rightarrow X$	IR_{out}, AR_{in}
T2		Read
T3	$Mem[AR] \rightarrow DR$	$DRE_{in}, Read$
T4	$(DR) \rightarrow R0$	$DR_{out}, R0_{in}$



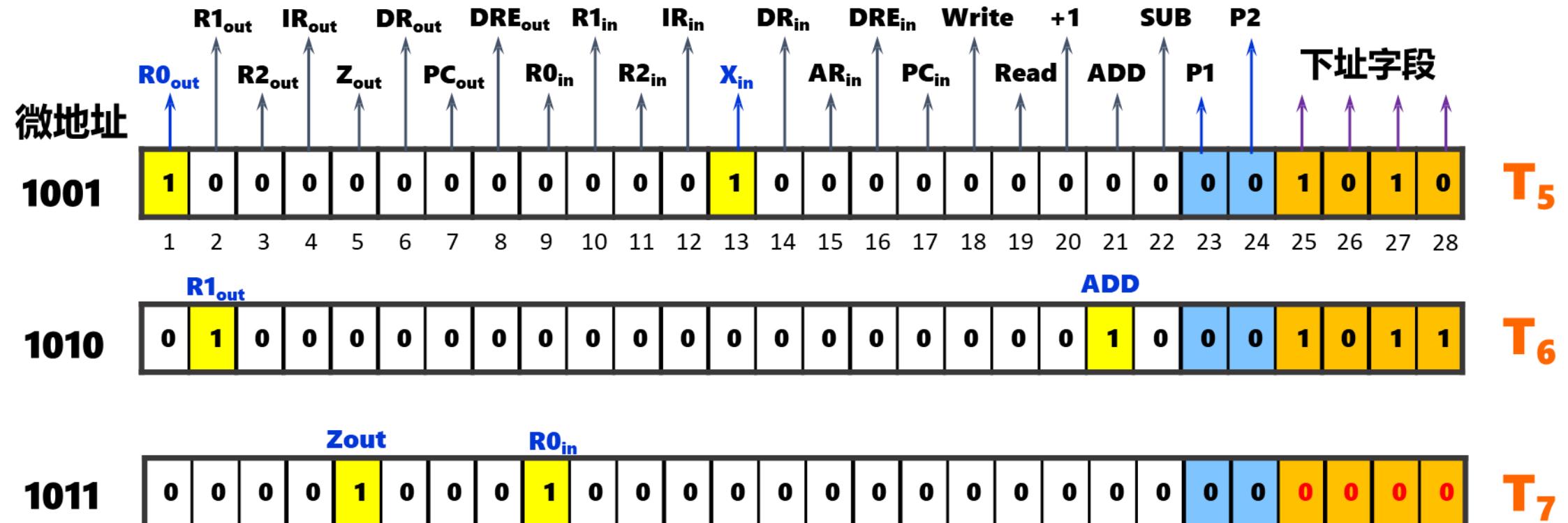
微程序设计

MOVE指令微程序



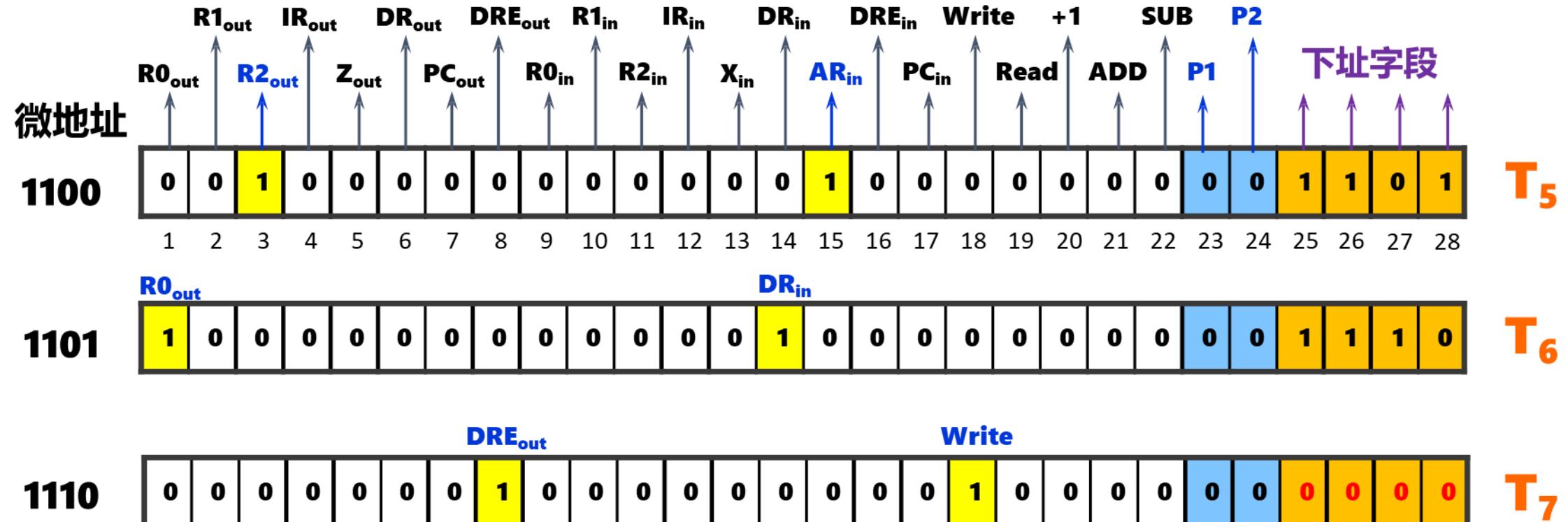
微程序设计

ADD指令微程序



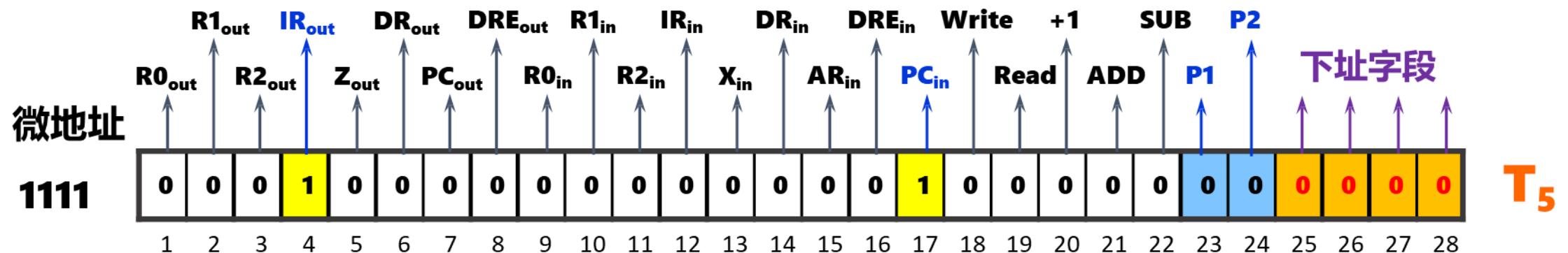
微程序设计

STORE指令微程序



微程序设计

JMP指令微程序



微程序设计

单总线CPU微程序

状态	微地址	操作控制字段																顺序控制字段								
		0	0	0	0	0	0	1	0	0	0	0	1	0	1	0	0	0	0	0	0	1				
S0	0000	0	0	0	0	0	0	1	0	0	0	0	1	0	1	0	0	0	0	0	0	1				
S1	0001	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	1	0			
S2	0010	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	1	1		
S3	0011	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	1	0	X	X	X	X	
S4	0100	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	1	
S5	0101	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	1	0
S6	0110	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	1	1	1
S7	0111	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
S8	1000	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
S9	1001	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	1	0	
S10	1010	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	1	1	
S11	1011	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
S12	1100	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	1	0	1	
S13	1101	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	1	1	0	
S14	1110	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	
S15	1111	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	

取指令微程序

LOAD微程序

MOVE微程序

ADD 微程序

STORE微程序

JMP 微程序

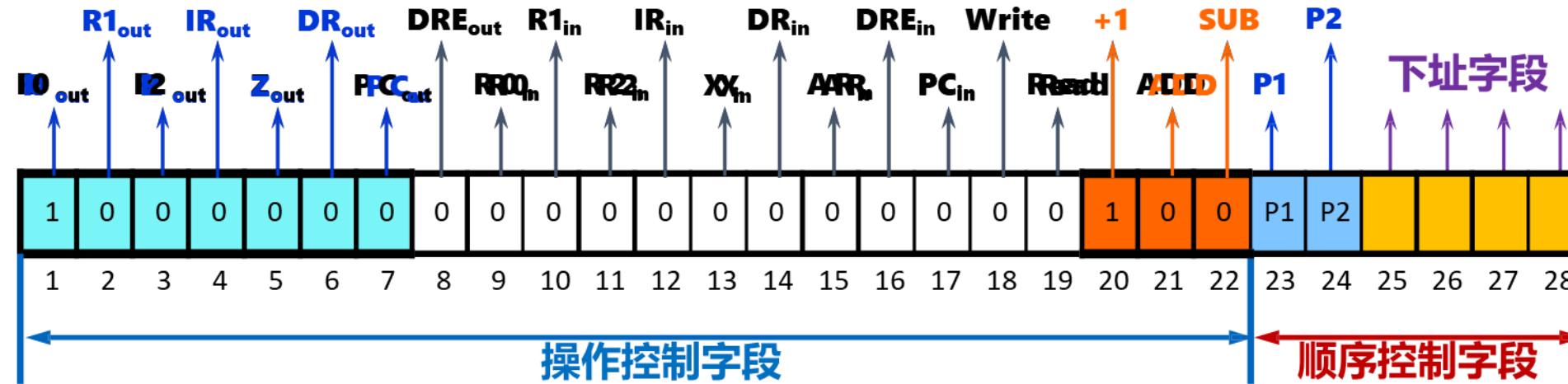
本节目录

- 微程序控制器
- 微程序设计
- 微指令格式
- 单周期MIPS CPU
- 多周期MIPS CPU

微指令设计原则

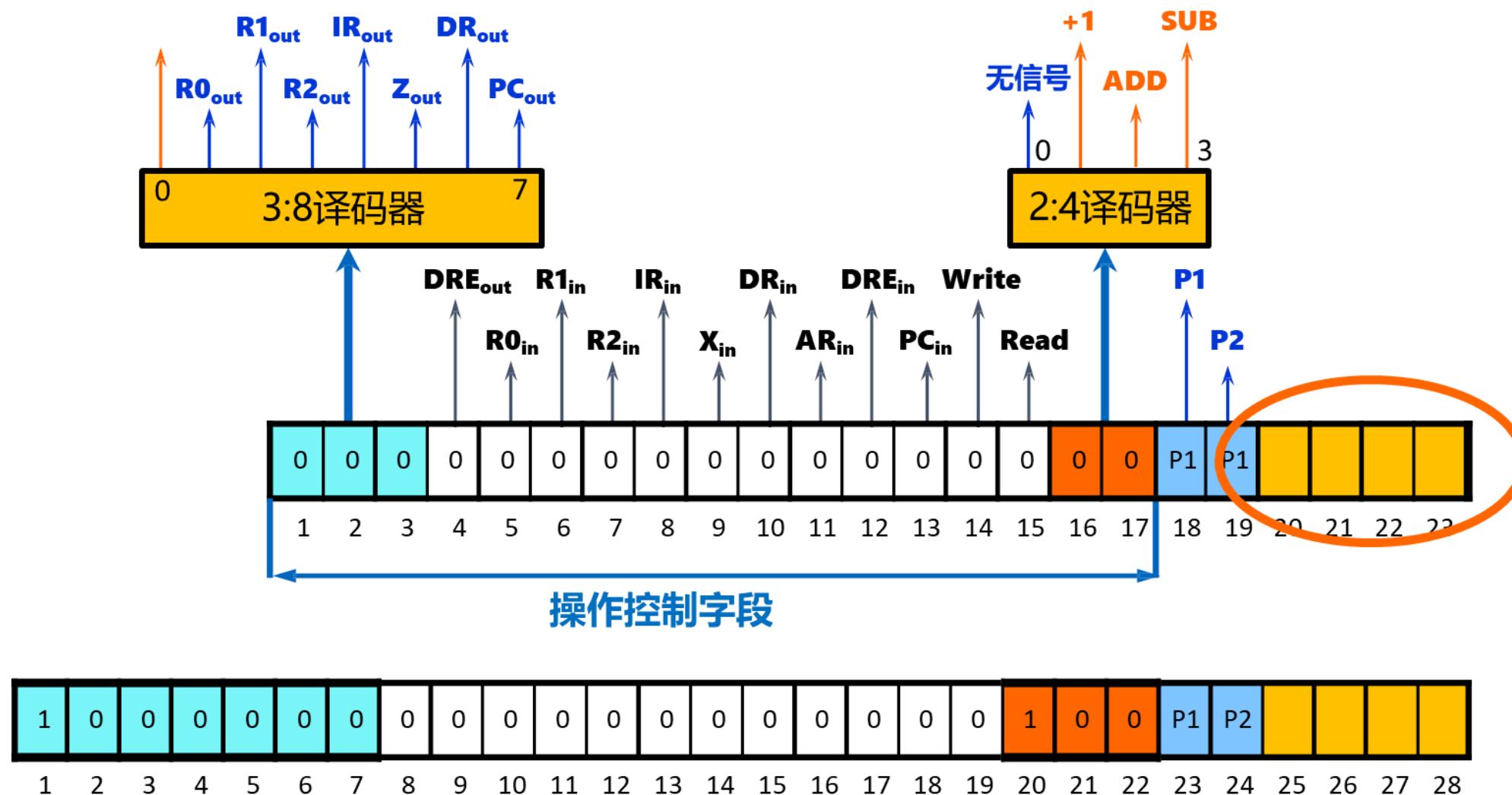
- 有利于缩短微指令字长度
- 有利于减少控制存储器容量
- 有利于提高微程序执行速度
- 有利于对微指令进行修改
- 有利于提高微程序设计的灵活性

微指令格式 (直接表示法)



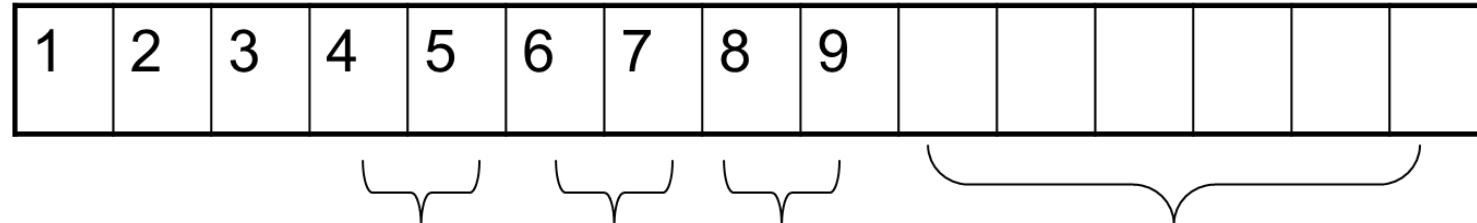
- 简单直观，便于输出控制，微指令长度太长，控存容量大，如何压缩微指令长度？
 - 改直接表示为编码表示（压缩互斥性微指令）
 - 去掉下址字段，采用 $\mu\text{PC} = \mu\text{PC} + 1$ 的方式生成微指令地址
 - 改水平型微指令为垂直型微指令（牺牲并行性）

微指令格式 (编码表示法)



混合表示法

- 将前两种结合在一起，兼顾两者特点。以便能够综合考虑微指令字长、灵活性、执行微程序速度等方面的要求



4、5:	6、7:	8、9:	顺序控制
00 无操作	00 无操作	00 无操作	
01 R1→X	01 R3→Y	01 +	
10 R2 → X	10 R2 → Y	10 -	
11 DR → X	11 R1 → Y	11 M	

混和表示法

- 1、2、3位为直接表示法
- 4、5、6、7 8、9位为编码表示法

编码表示方法

- 编码注意几点：字段编码法中操作控制字段并非是任意的，必须要遵循如下的原则：
 - ①把互斥性的微命令分在同一段内，兼容性的微命令分在不同段内。这样不仅有助于提高信息的利用率，缩短微指令字长，而且有助于充分利用硬件所具有的并行性，加快执行的速度。
 - ②应与数据通路结构相适应。
 - ③每个小段中包含的信息位不能太多，否则将增加译码线路的复杂性和译码时间。
 - ④一般每个小段还要留出一个状态，表示本字段不发出任何微命令。因此当某字段的长度为三位时，最多只能表示七个互斥的微命令，通常用000表示不操作。

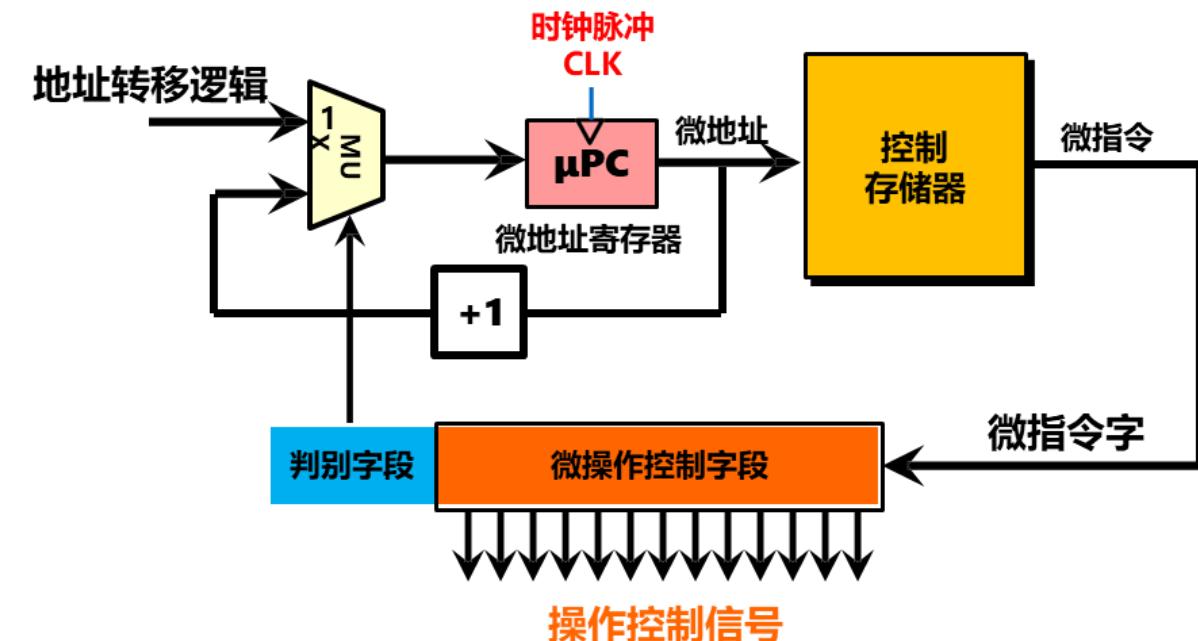
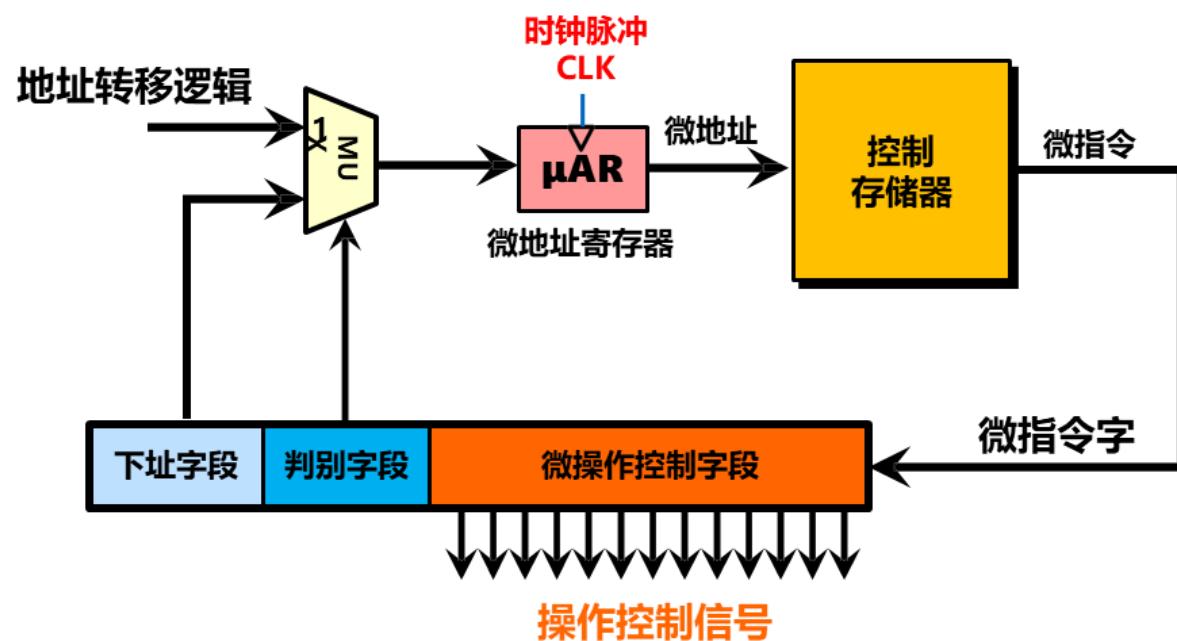
微指令地址形成方法

■ 下地址字段法

微指令长，控存容量大

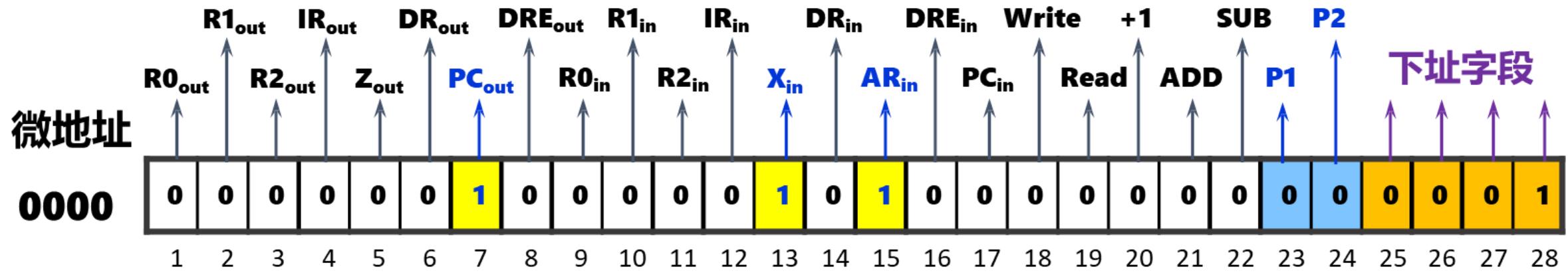
■ 计数器法 μ PC

微指令短，需要加法器



水平型微指令编码效率

微地址	操作控制字段																顺序控制字段						
	0	0	0	0	0	0	0	1	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0
0000	0	0	0	0	0	0	0	1	0	0	0	0	1	0	1	0	0	0	0	0	0	0	1
0001	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	1
0010	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	1
0011	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	1	0	X	X	X
0100	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0
0101	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	1
0110	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	1	1
0111	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1000	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1001	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	1
1010	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	1
1011	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1100	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	1	0
1101	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	1	1
1110	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
1111	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0



垂直型微指令

■ 数据通路分类

- ◆ 寄存器传输 **MOV reg1, reg2** **MOV AR, PC**
- ◆ 运算类型 **ALU_OP reg** **INC、ADD R1**
- ◆ 访存类型 **Mem** **LOAD、STORE**
- ◆ 顺序控制 **Branch** **Branch P1**

如采用扩展操作码，需8位字长

MOV	源寄存器(3)	目的寄存器(3)
INC		无操作数
ADD	源寄存器编号 (3位)	
Mem		Read/Write
Branch		判断条件

节拍	取指令	LOAD	MOVE	ADD	STORE	JMP
T1	$(PC) \rightarrow AR, (PC) \rightarrow X$	$(IR_A) \rightarrow AR, (PC) \rightarrow X$	$(IR_A) \rightarrow R[0]$	$(R0) \rightarrow X$	$(R2) \rightarrow AR$	$(IR_A) \rightarrow PC$
T2	$(X) + 1 \rightarrow Z$ Read Mem	Read Mem		$(X) + (R1) \rightarrow Z$	$(R0) \rightarrow DR$	
T3	$(Z) \rightarrow PC$, Mem[AR] $\rightarrow DR$	Mem[AR] $\rightarrow DR$		$(Z) \rightarrow R0$	$(DR) \rightarrow Mem[AR]$	
T4	$(DR) \rightarrow IR$	$(DR) \rightarrow R0$				

垂直型微指令实例

LOAD指令微程序容量：

■ 直接表示法 + 下址字段

◆ 28位*8=224位

■ 编码表示法 + μPC

◆ 19位*8=152位

■ 垂直微指令

◆ 8位*14=112位

节拍	LOAD指令	#	LOAD	信号
T1	(PC)→AR, (PC)→X	C1	MOV AR,PC	PC _{out} , AR _{in}
		C2	MOV X,PC	PC _{out} , X _{in}
T2	(X)+1→Z Read Mem	C3	LOAD	DRE _{in} , Read
		C4	LOAD	DRE _{in} , Read
		C5	INC	+1
T3	(Z)→PC, Mem[AR]→DR	C6	MOV PC,Z	Z _{out} , PC _{in}
T4	(DR)→IR	C7	MOV IR,DR	DR _{out} , IR _{in}
		C8	Branch P1	P1
T5	(IR _A)→AR, (PC)→X	C9	MOV AR,IR	IR _{out} , AR _{in}
		C10	MOV X, PC	PC _{out} , X _{in}
T6	Read Mem	C11	LOAD	DRE _{in} , Read
T7	Mem[AR]→DR	C12	LOAD	DRE _{in} , Read
T8	(DR)→R0	C13	MOV R0, DR	DR _{out} , R0 _{in}
		C14	Branch	

水平型与垂直型微指令对比

■ 水平型微指令

- ◆ 并行操作能力强，效率高，灵活性强，
- ◆ 微指令字较长，微程序短，控存容量大，性能佳

■ 垂直型微指令

- ◆ 字长短，微程序长，控存容量小，性能差
- ◆ 垂直型与指令相似，易于掌握
- ◆ 基本被淘汰

本节目录

- 微程序控制器
- 微程序设计
- 微指令格式
- 单周期MIPS CPU
- 多周期MIPS CPU

MIPS CPU控制器设计

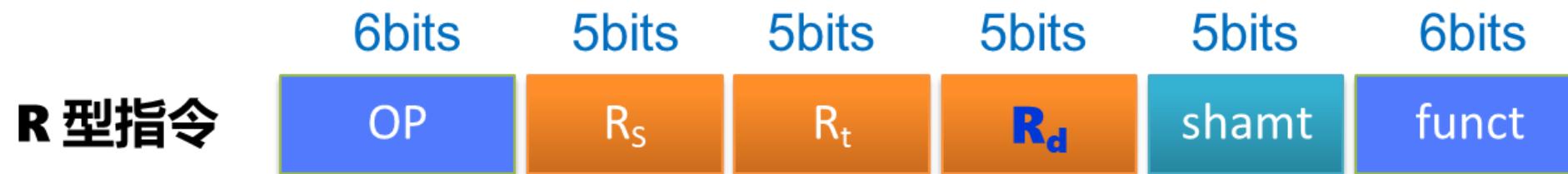
■ 定长指令周期：单周期实现

- ◆ 所有指令均在一个时钟周期内完成， $CPI=1$
- ◆ 性能取决于最慢的指令，时钟周期过长

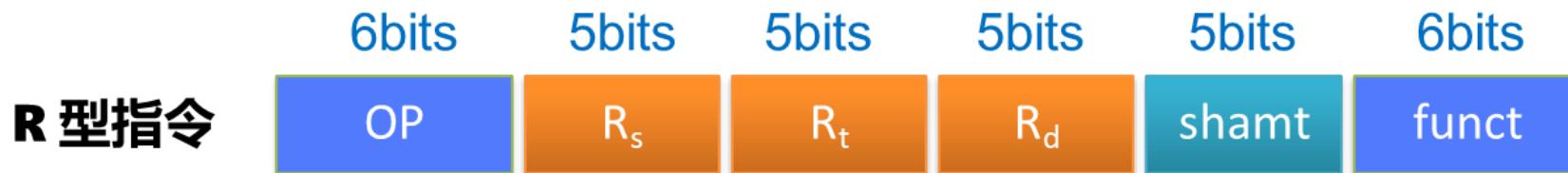
■ 变长指令周期：多周期实现

- 缩短时钟周期，复用器件或数据通路
- 可支持流水操作，提升性能

MIPS指令格式



R型指令格式



■ **add \$s1,\$s2,\$s3**



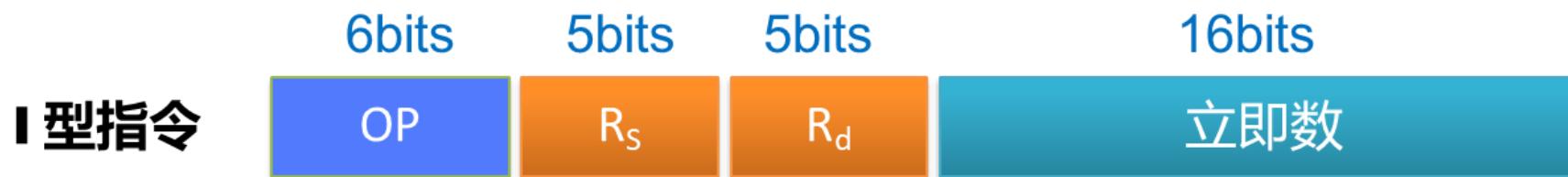
■ **sub \$s0,\$s1,\$s2**



■ **sll \$s0,\$s1,2**



I型指令格式



■ addi \$s1,\$s2,200



■ lw \$s1,300(\$s2)

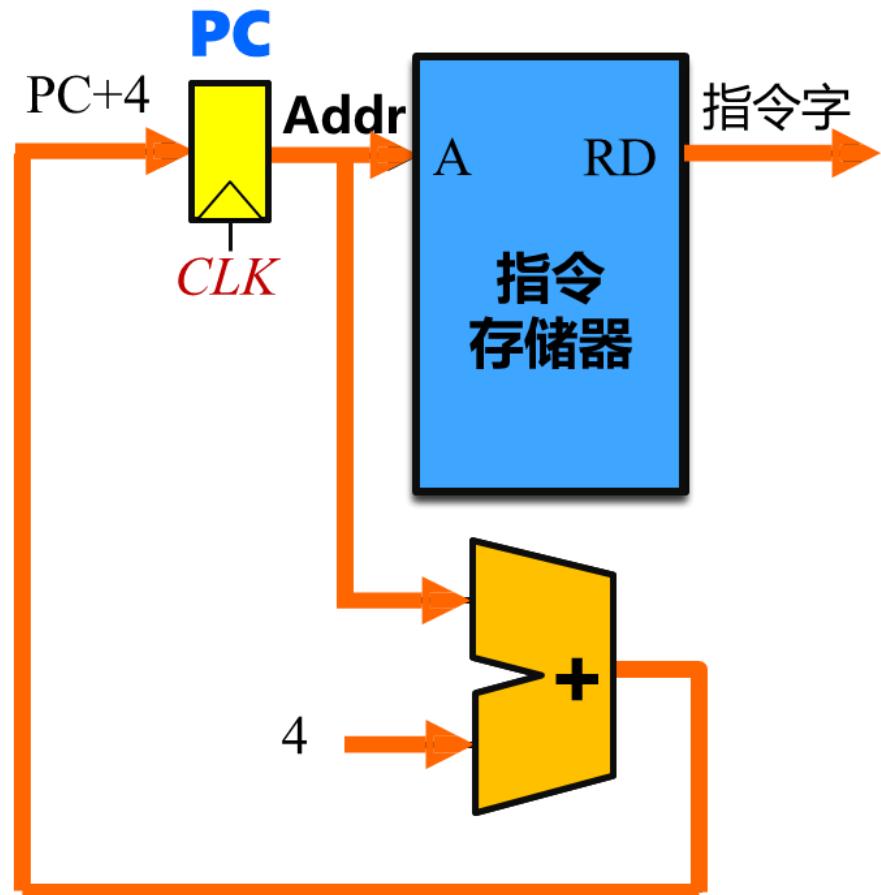


■ beq \$s1,\$s2,400



单周期MIPS CPU

取指令数据通路

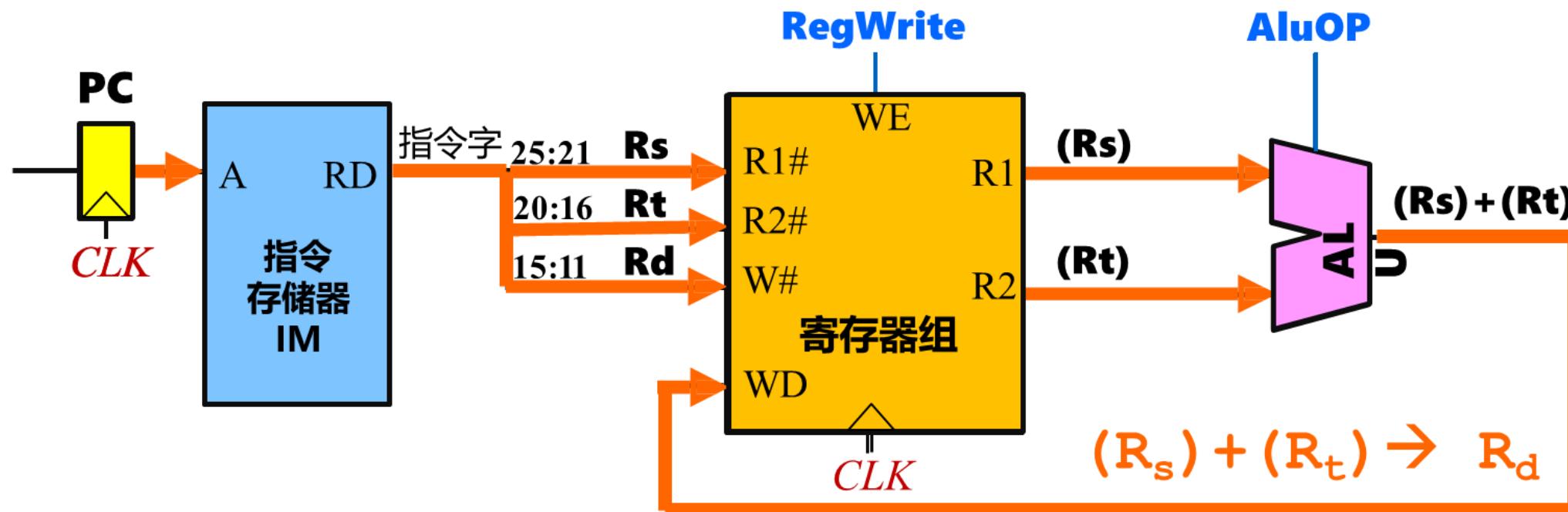


Mem[PC++] → IR

- 单周期不能设置AR, DR, IR寄存器
- 程序和数据分开存放---哈佛结构
 - ◆ 指令存储器 数据存储器
 - ◆ 指令cache 数据cache
- 运算器和PC累加器分离

单周期MIPS CPU

R型指令数据通路

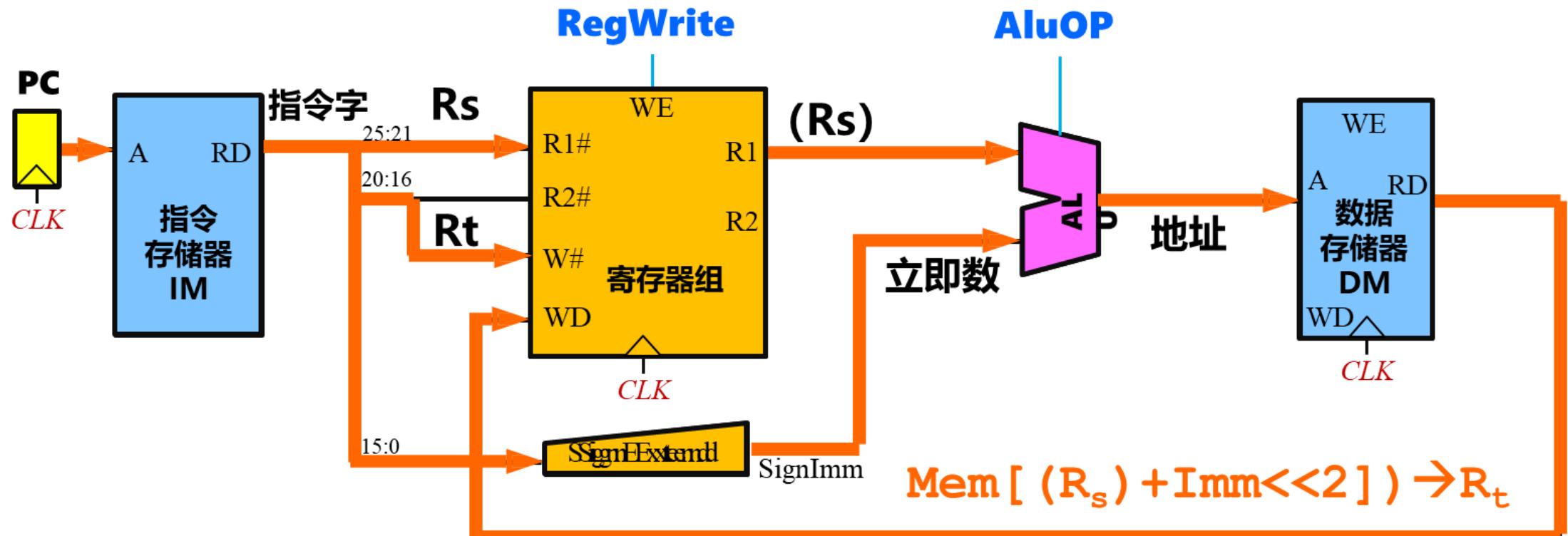


add \$s0,\$s1,\$s2



单周期MIPS CPU

Iw指令数据通路



Iw \$s0, 32(\$s1)

6bits
OP

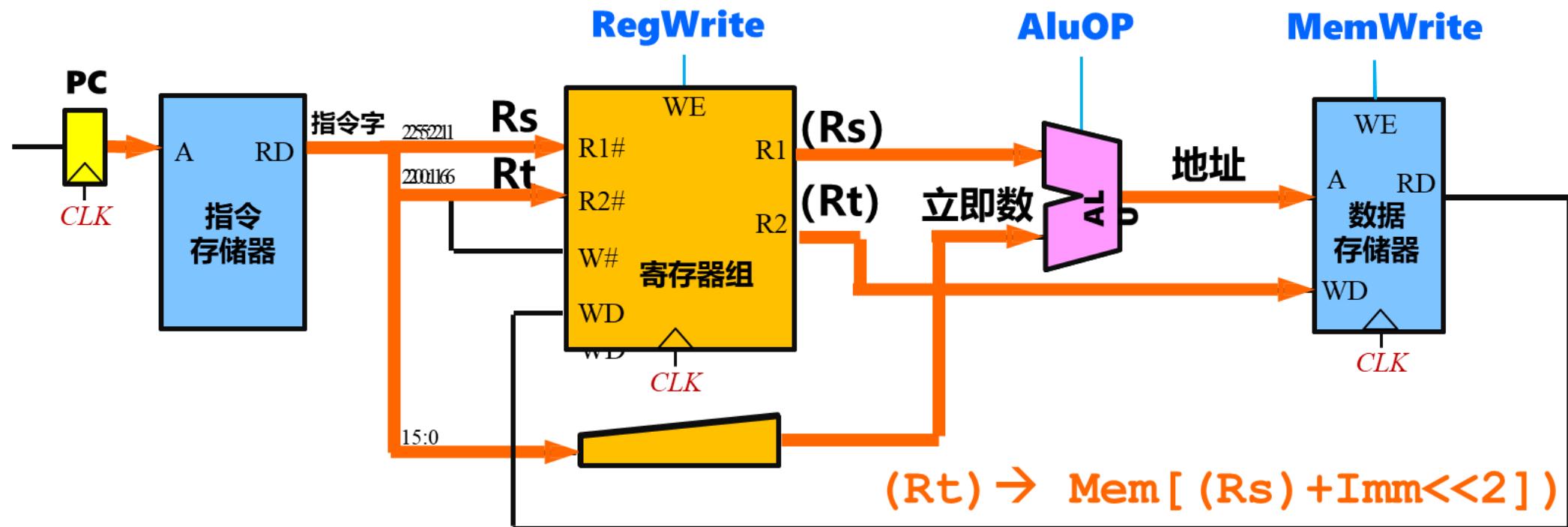
5bits
R_s

5bits
R_t

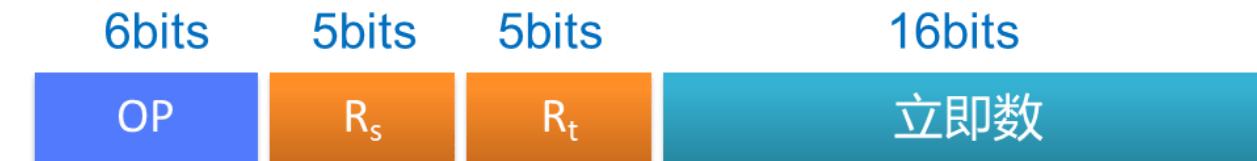
16bits
立即数

单周期MIPS CPU

sw指令数据通路

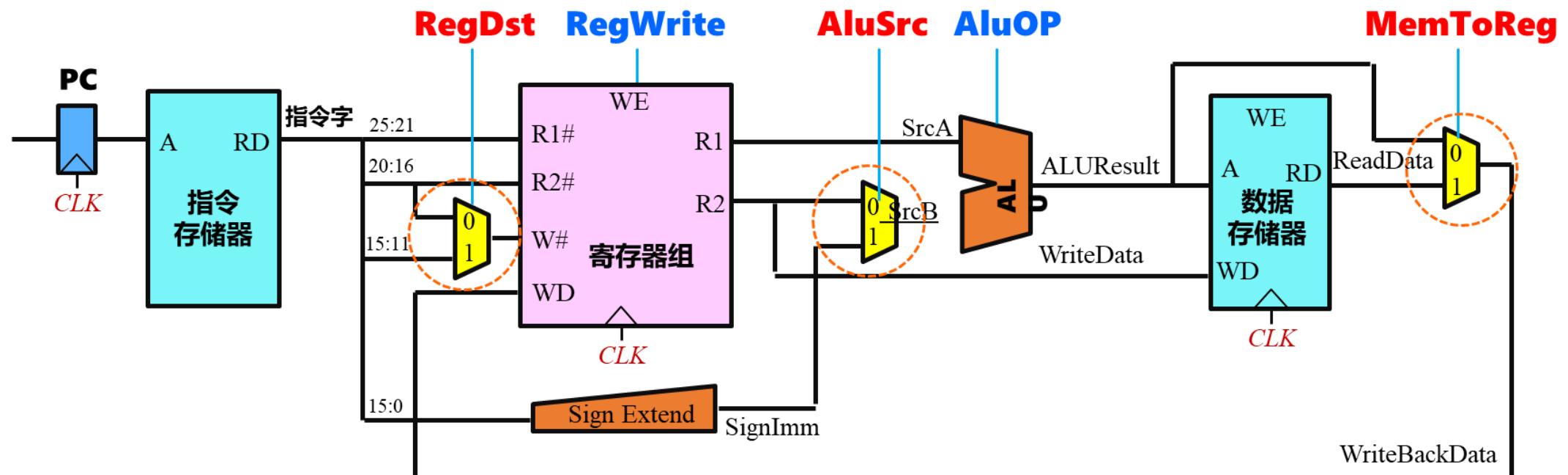


■ **sw \$s0 , 32(\$s1)**



单周期MIPS CPU

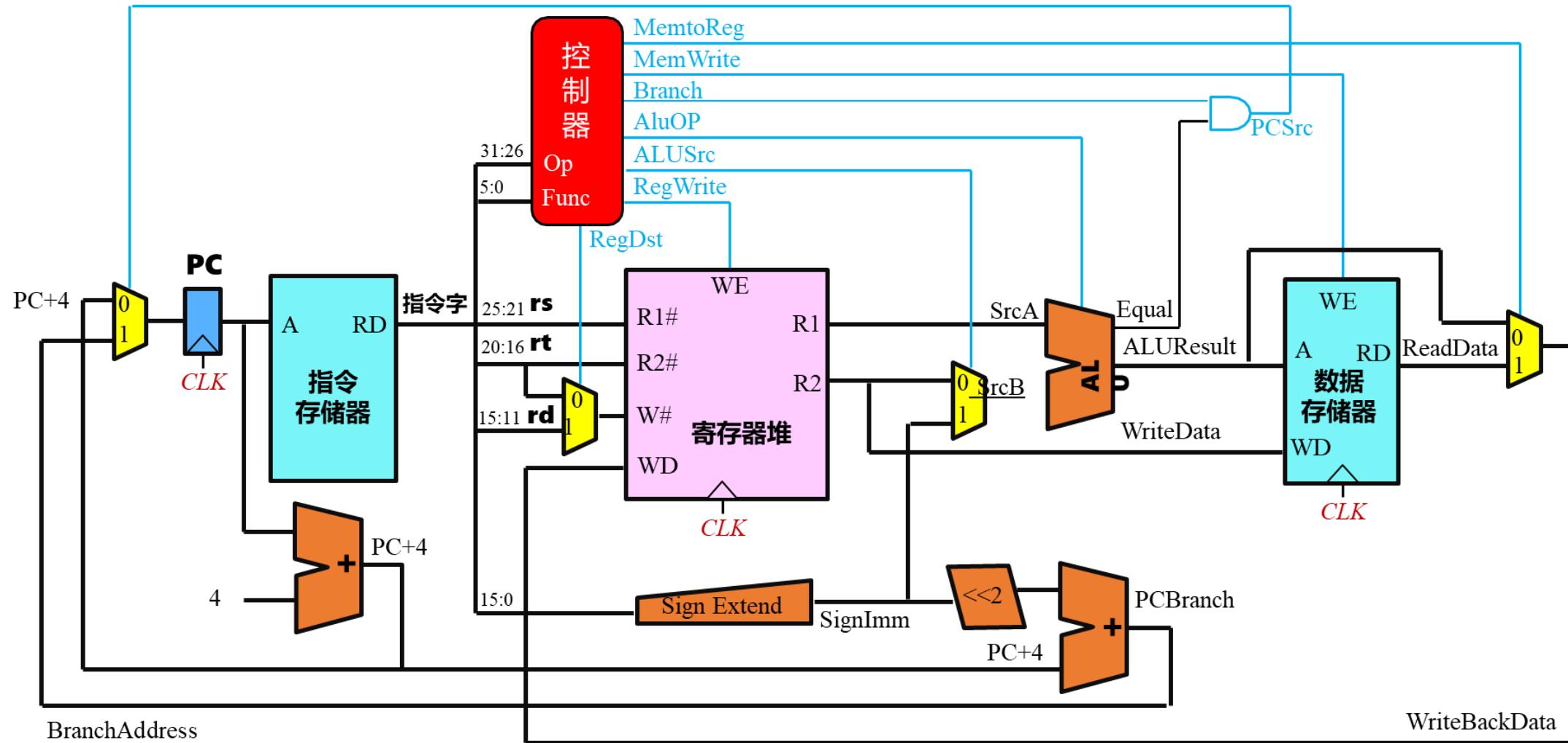
数据通路综合



凡是多个输入来源的，增加MUX，引入控点

单周期MIPS CPU

单周期MIPS数据通路

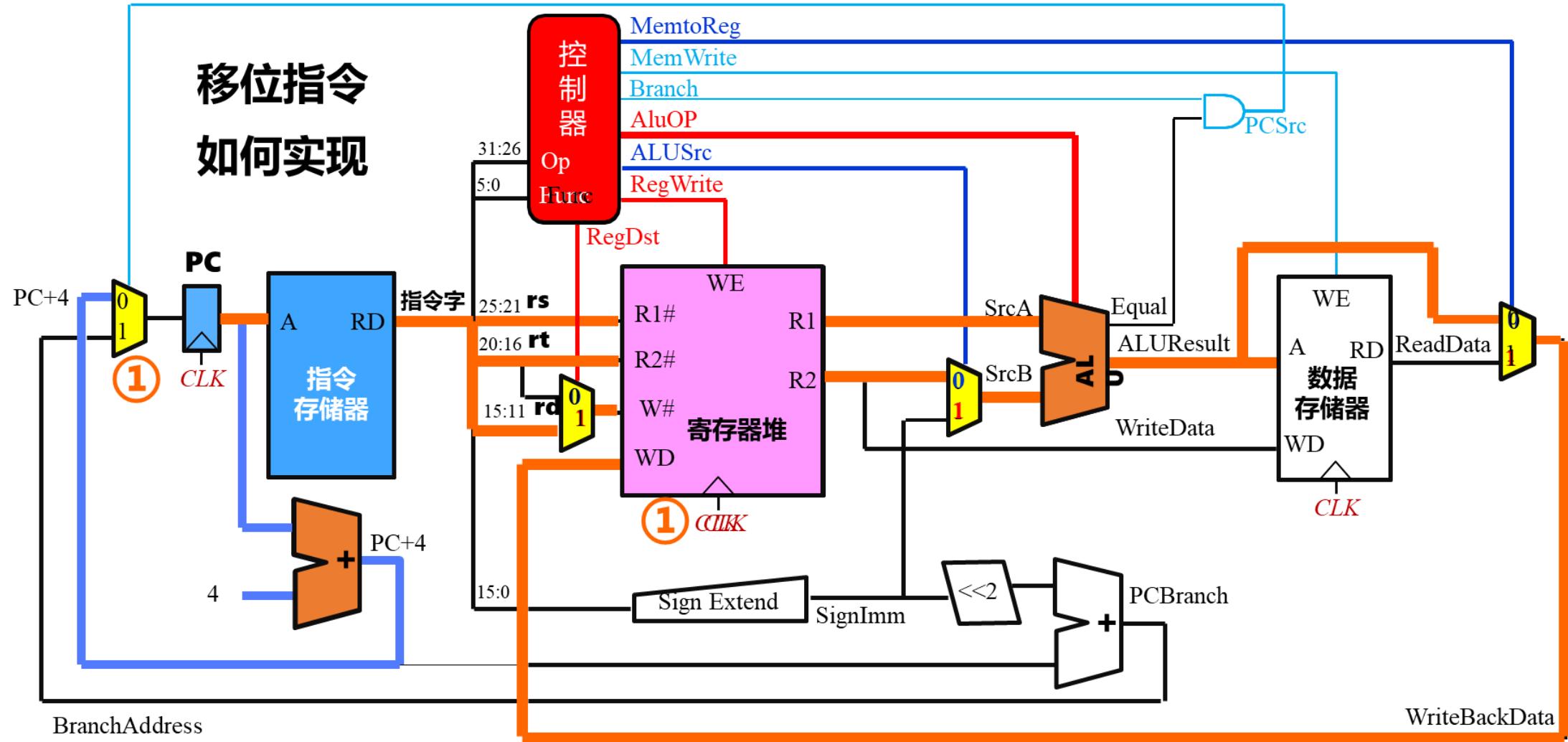


单周期MIPS CPU

R型指令数据通路建立过程



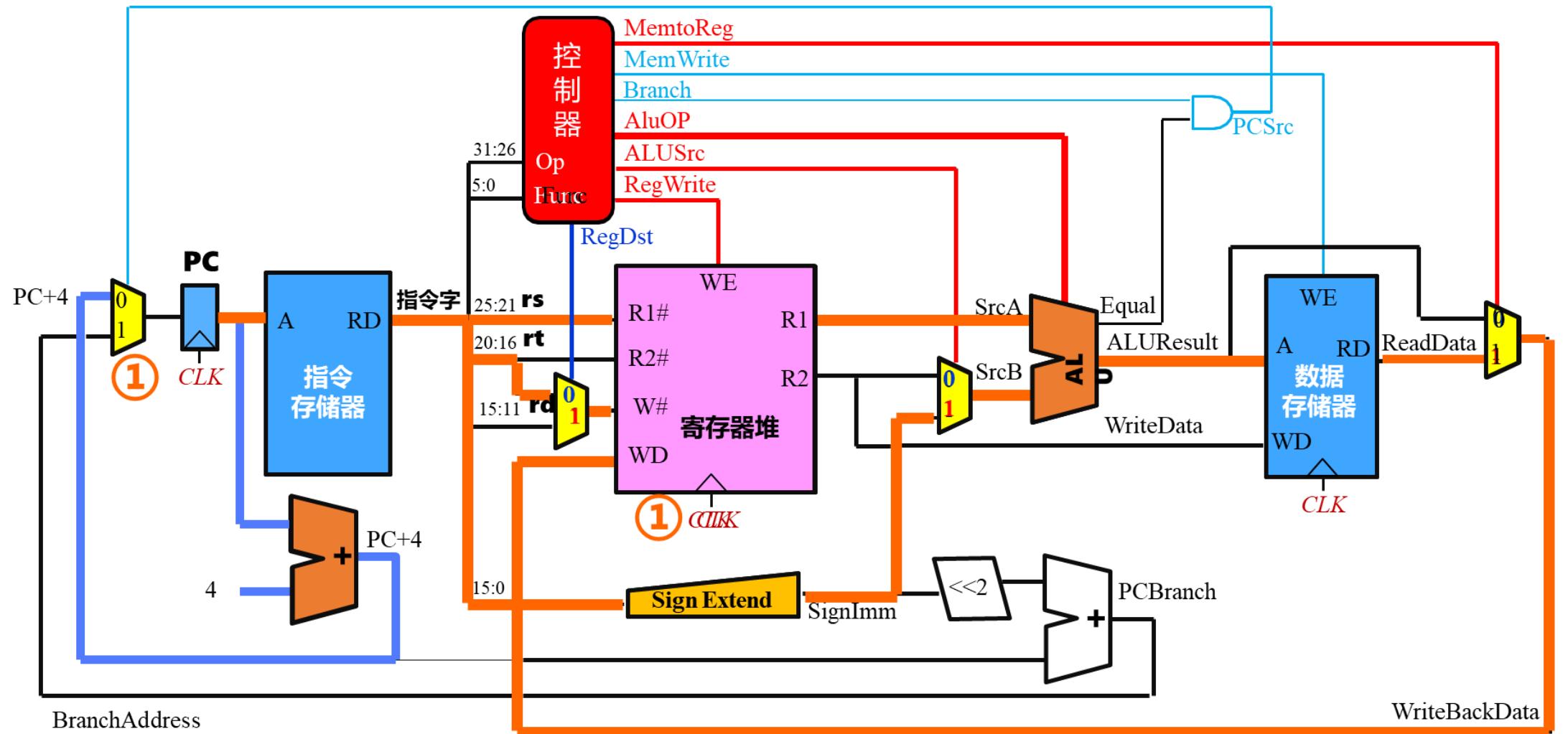
移位指令
如何实现



单周期MIPS CPU

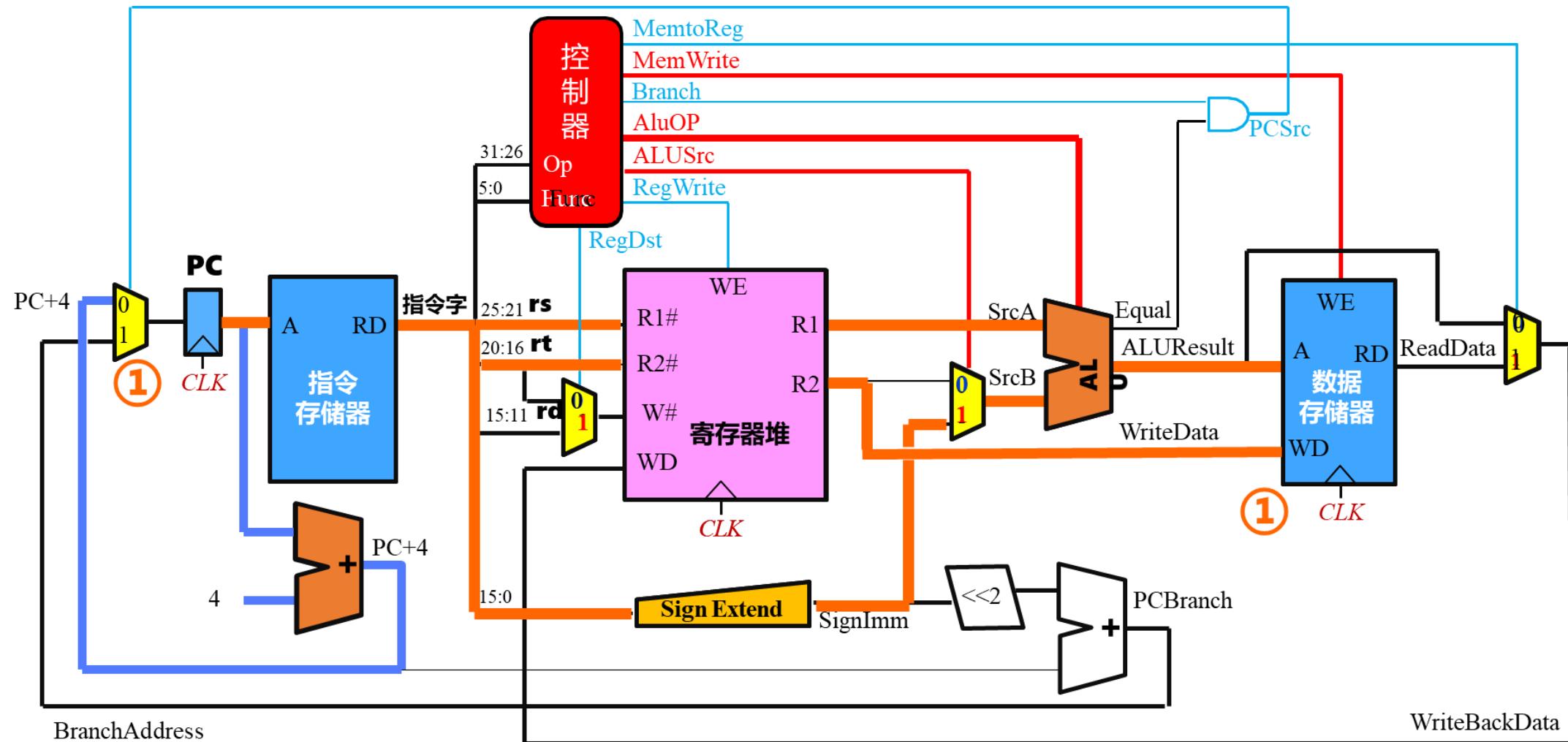
LW指令数据通路建立过程

6bits 5bits 5bits 16bits
OP R_s R_t 立即数



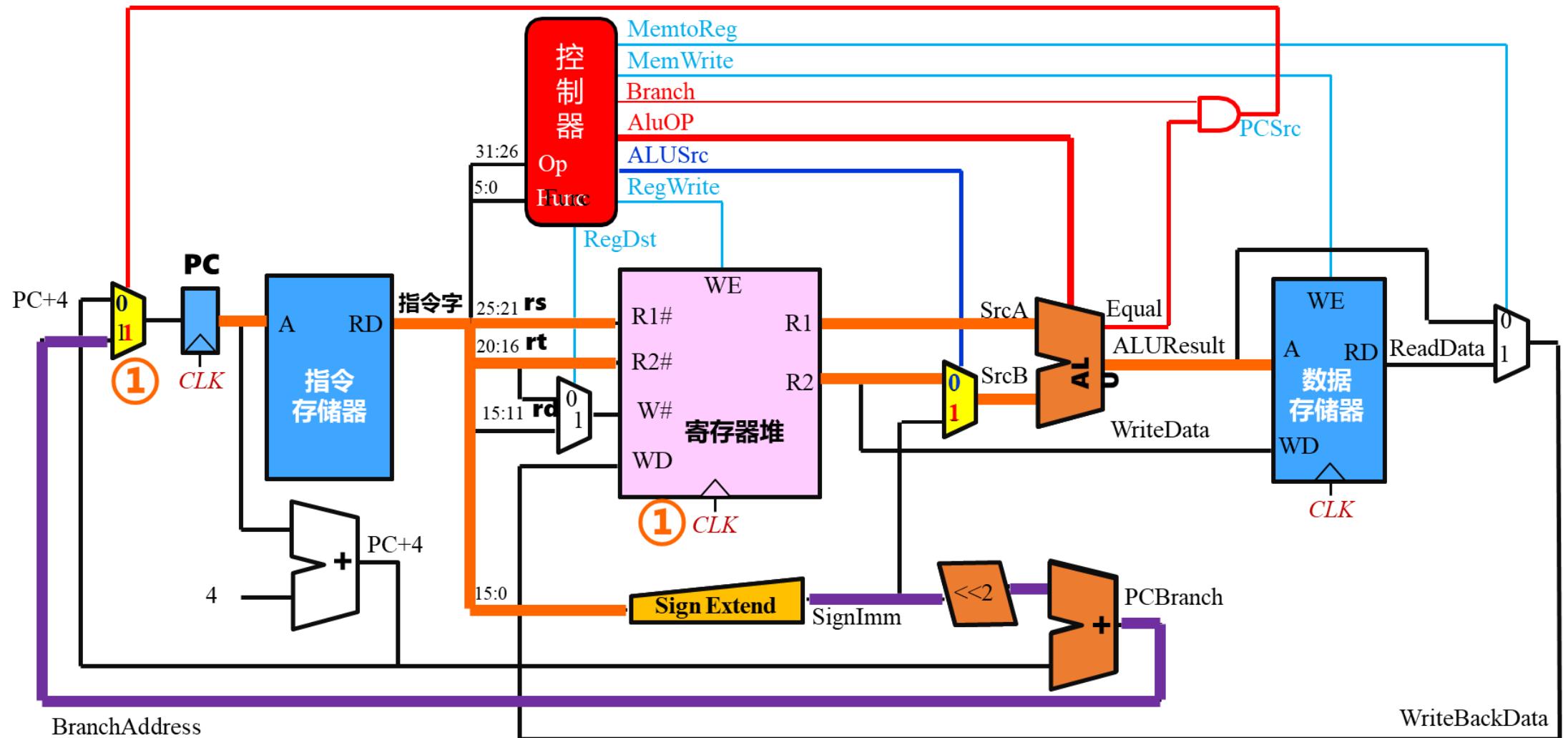
单周期MIPS CPU

SW指令数据通路建立过程



单周期MIPS CPU

BEQ指令数据通路建立过程



单周期MIPS CPU

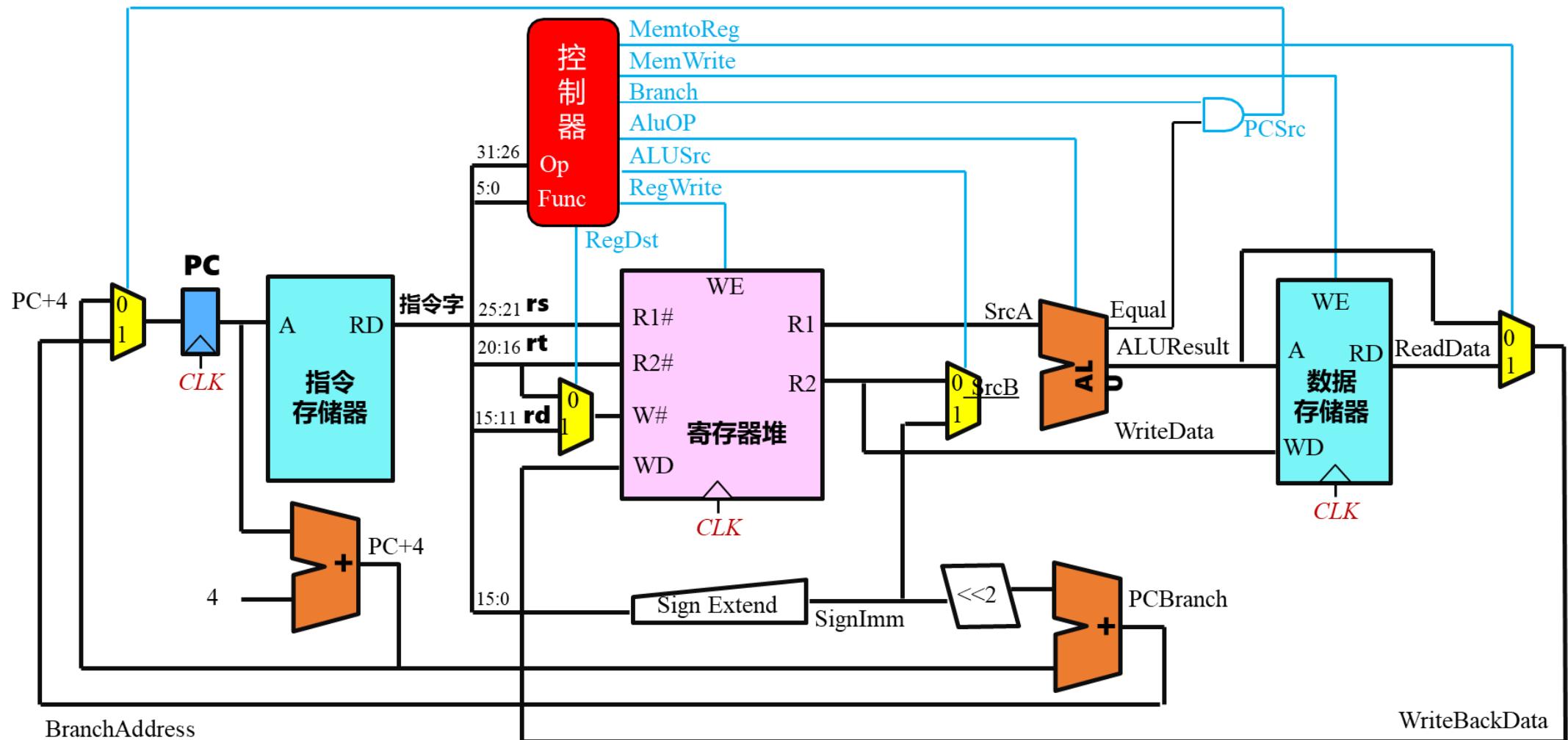
J指令数据通路建立?

6bits

OP

26bits

立即数



单周期MIPS CPU

单周期MIPS控制器设计

- 单周期控制器无时序逻辑，纯组合逻辑电路

- 输入信号

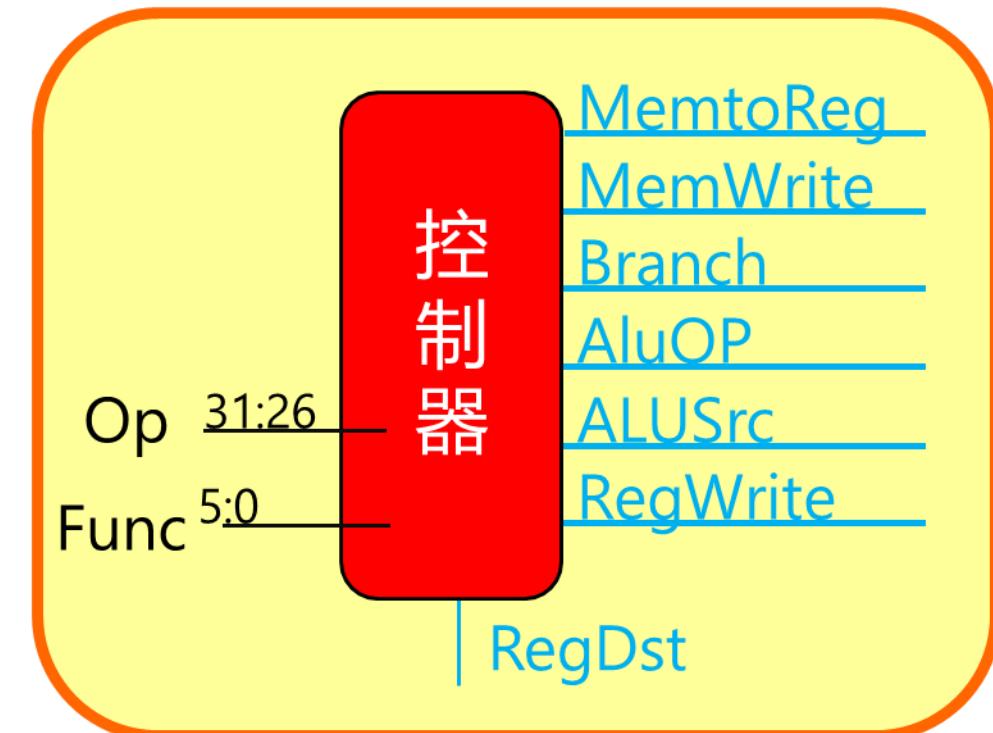
- 指令字**Opcode**, **Func**字段（12位）

- 输出信号

- 多路选择器选择信号

- 内存访问控制信号

- 寄存器写使能信号

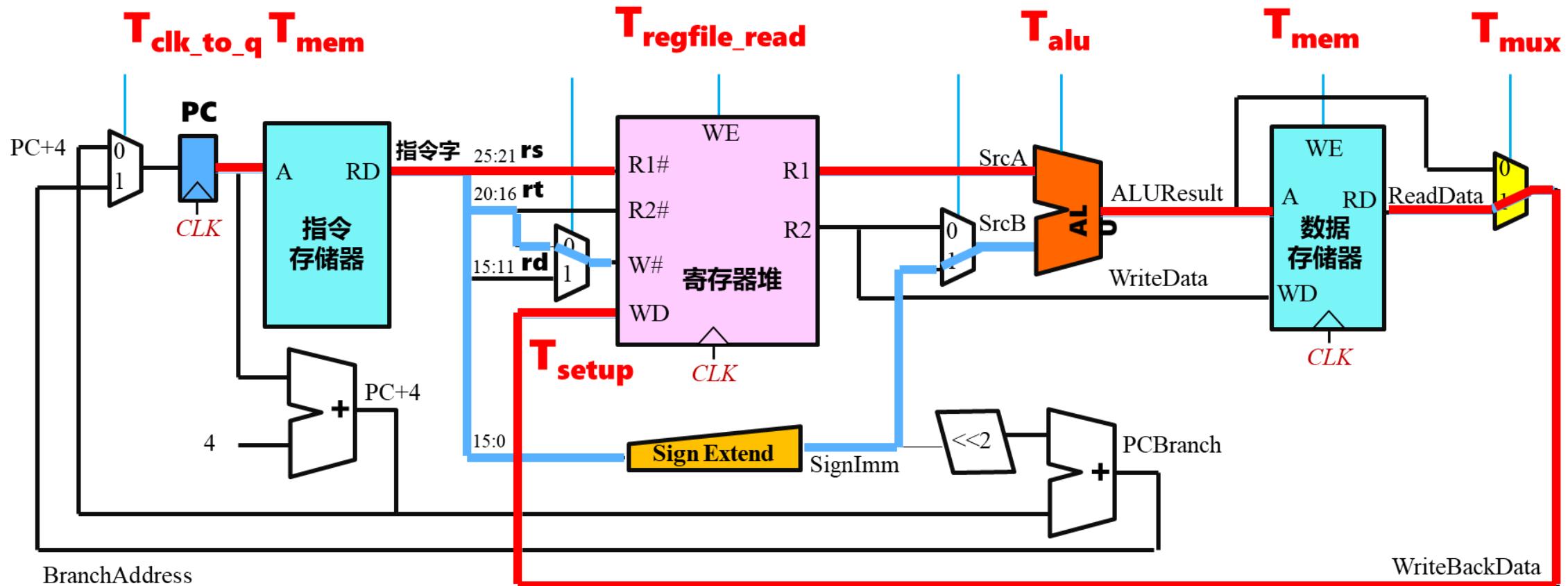


本节目录

- 微程序控制器
- 微程序设计
- 微指令格式
- 单周期MIPS CPU
- 多周期MIPS CPU

多周期MIPS CPU数据通路

单周期MIPS关键路径 LW指令

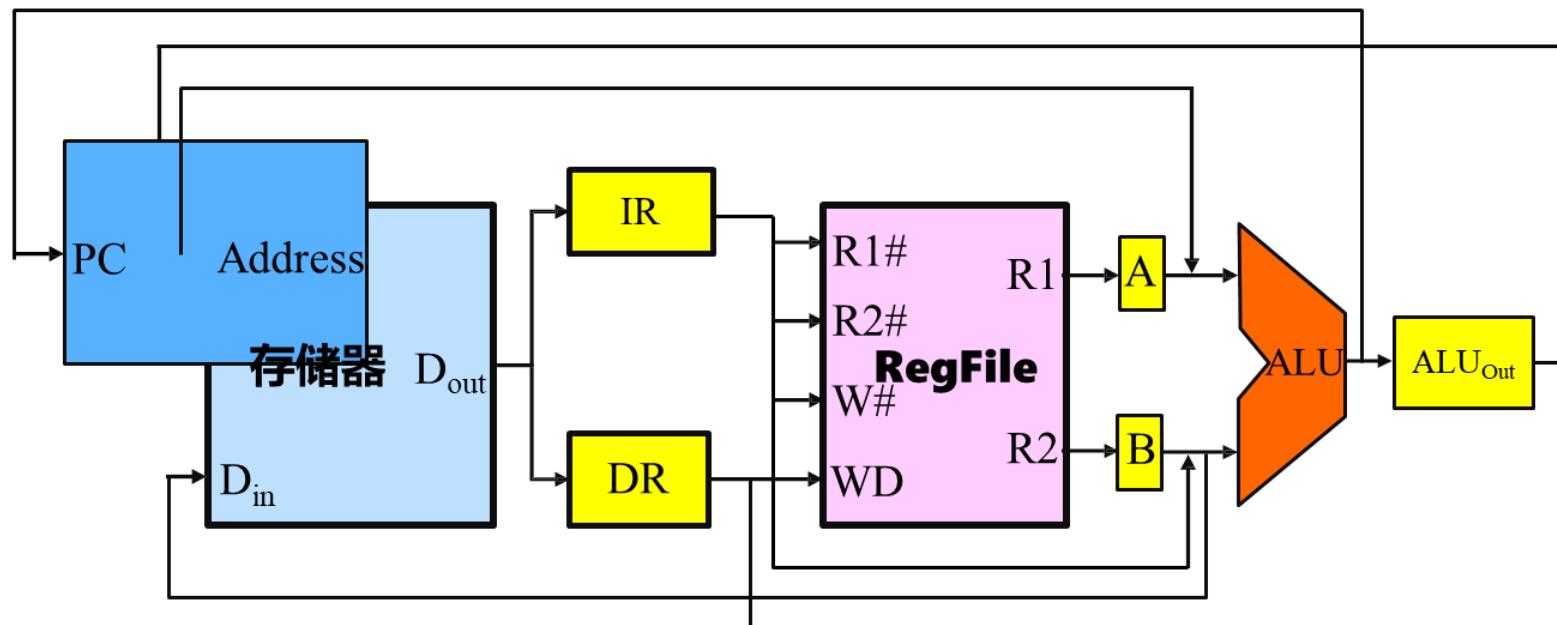


性能取决于最慢的指令，时钟周期过长

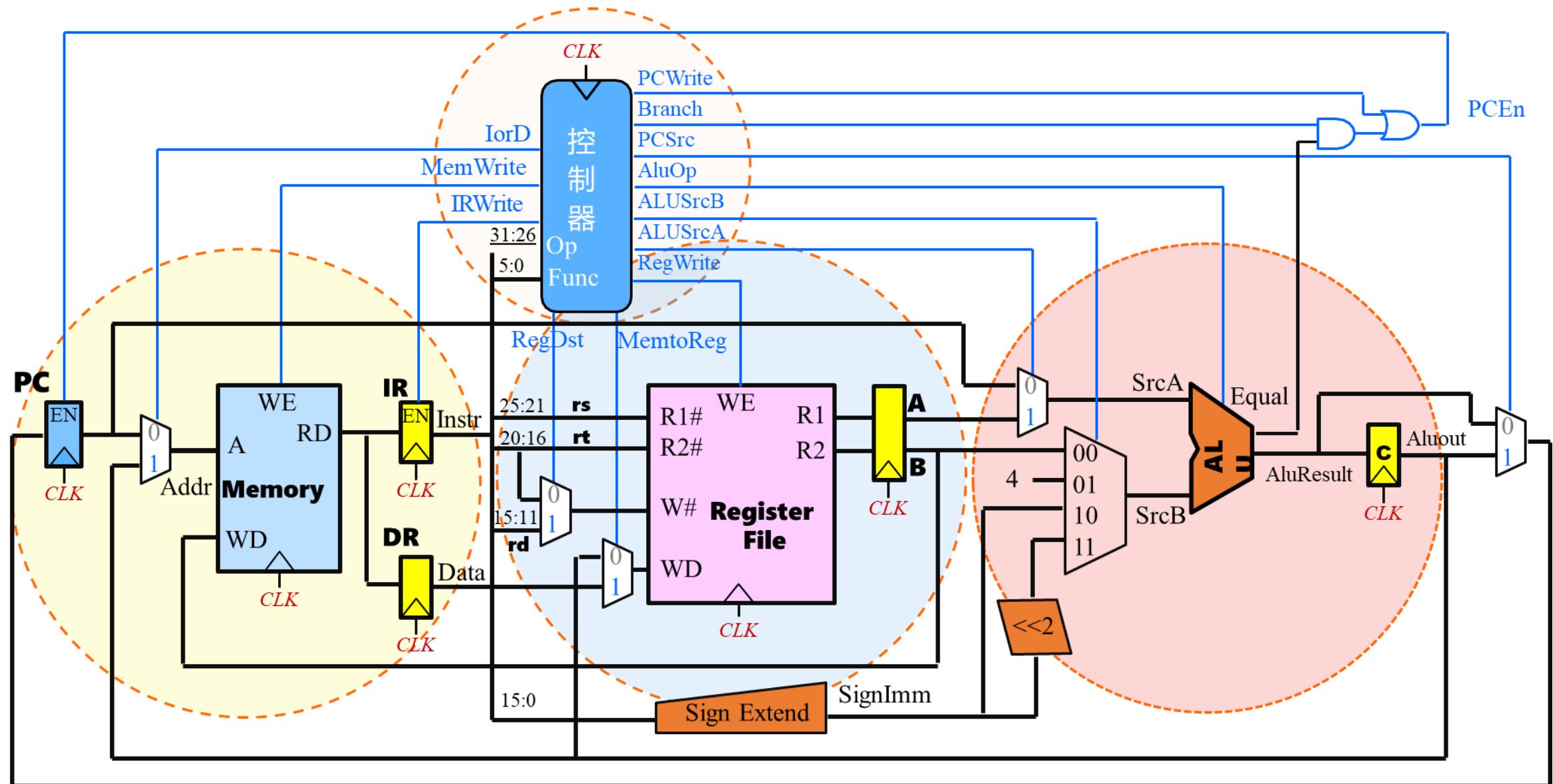
多周期MIPS CPU数据通路

多周期MIPS数据通路特点

- 不再区分指令存储器和数据存储器，分时使用部分功能部件
- 主要功能单元输出端增加寄存器锁存数据
- 传输通路延迟变小，时钟周期变短

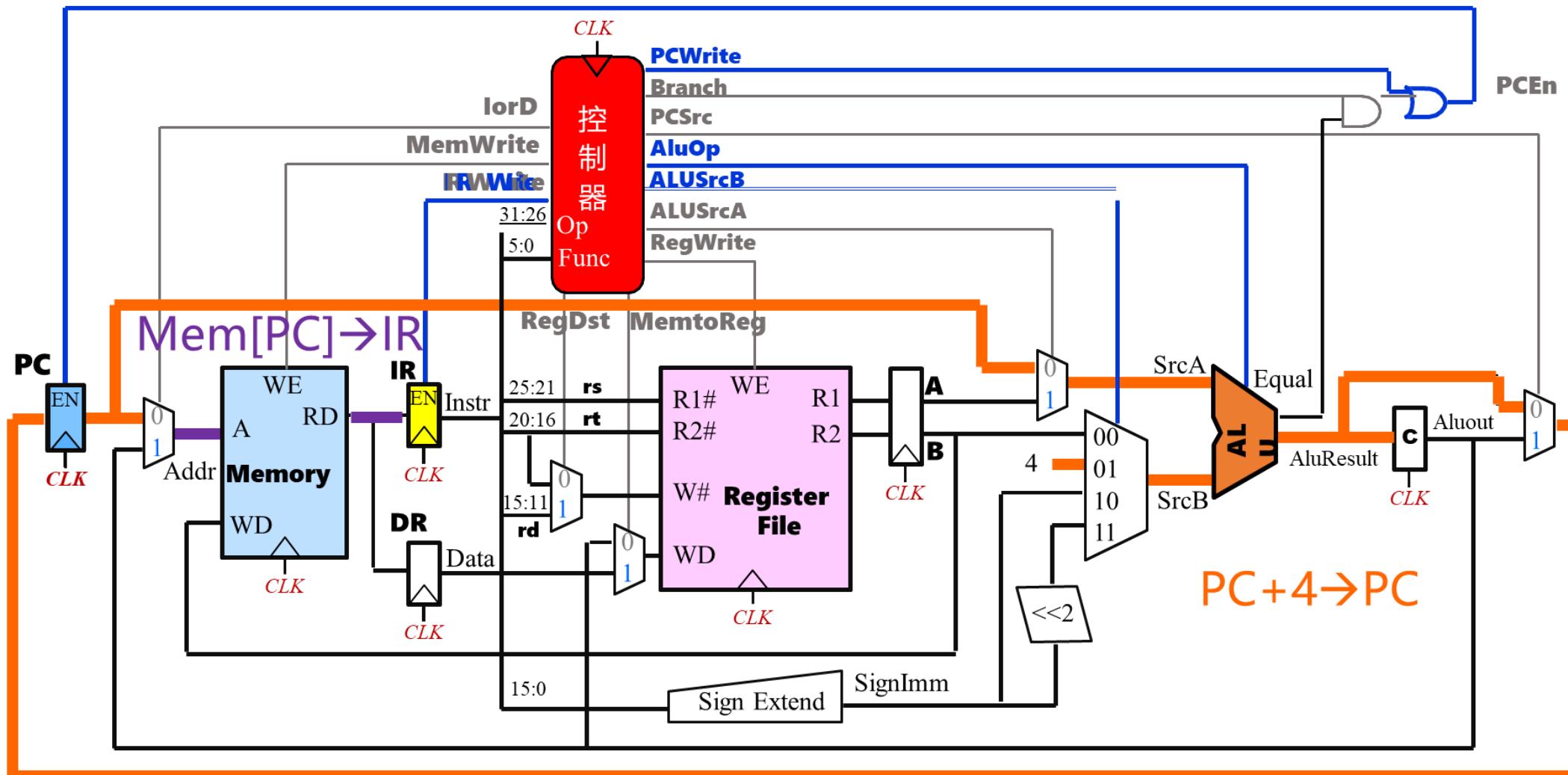


多周期MIPS CPU数据通路



多周期MIPS CPU数据通路

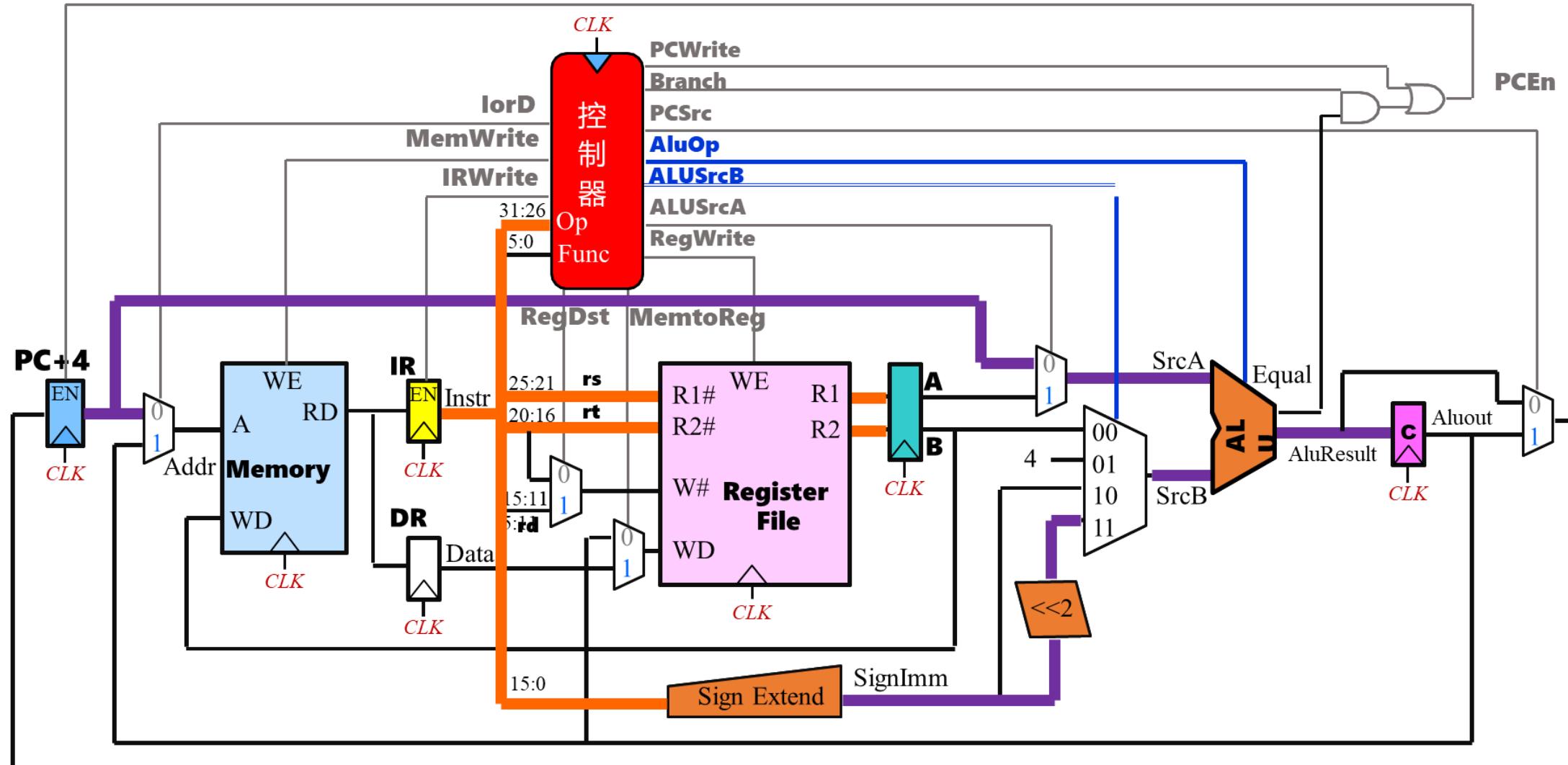
多周期MIPS取指令阶段T1 Mem[PC]→IR PC+4→PC



多周期MIPS CPU数据通路

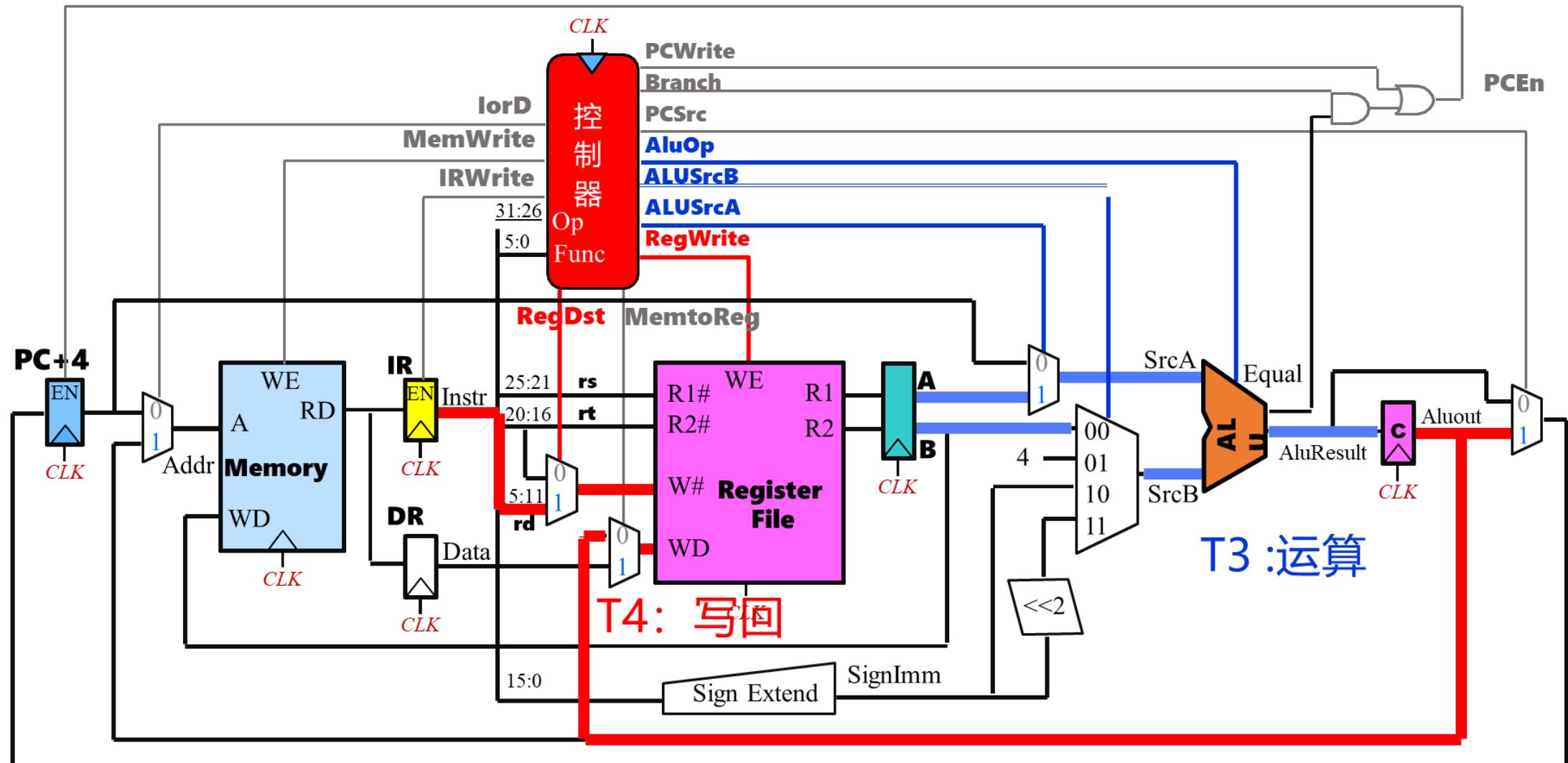
多周期MIPS取指令阶段T2

译码、Reg→A、B、PC+4+Imm16<<2→C



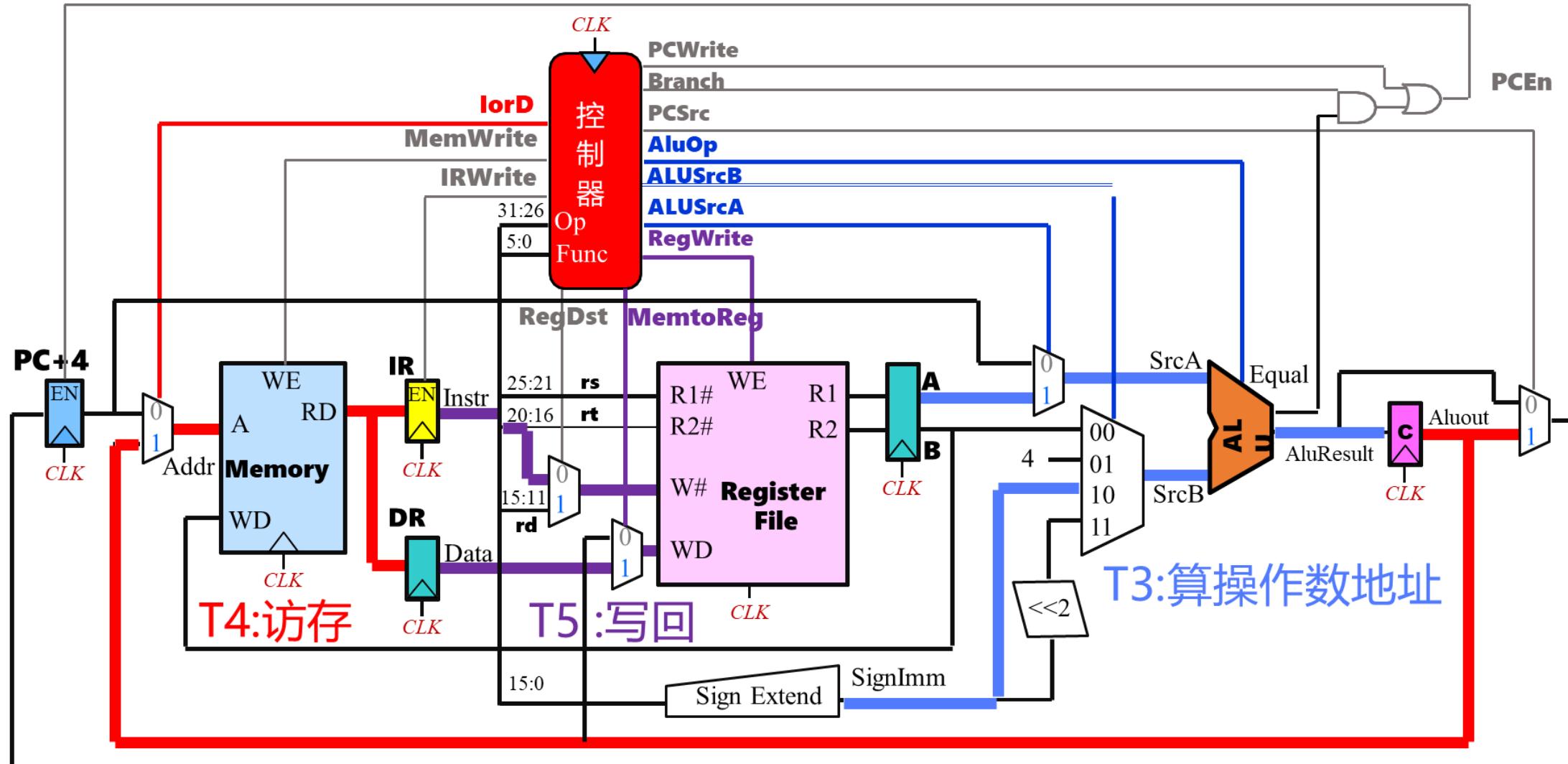
多周期MIPS CPU数据通路

R型指令执行状态周期T3~T4



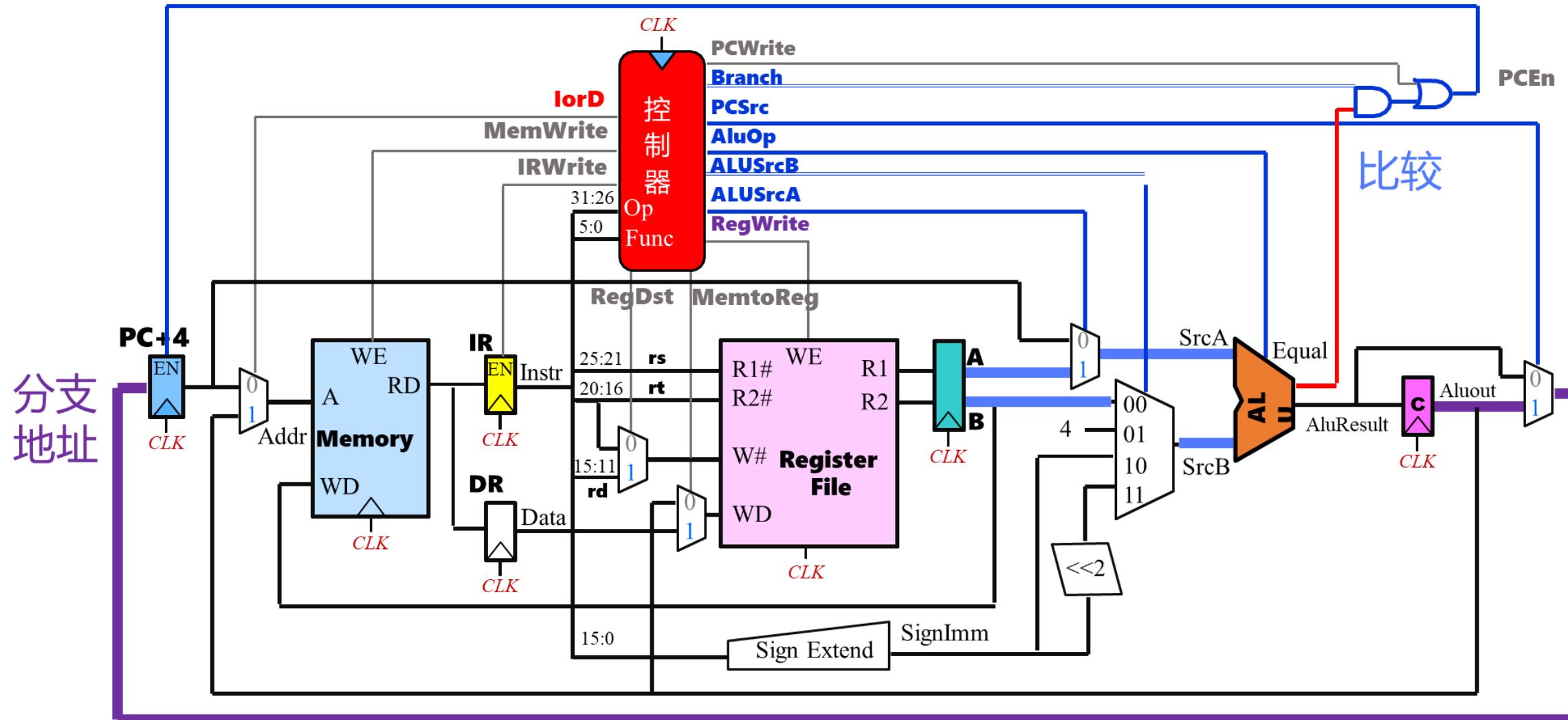
多周期MIPS CPU数据通路

LW指令执行状态周期T3~T5



多周期MIPS CPU数据通路

Beq指令执行状态周期T3



多周期MIPS CPU数据通路

多周期状态转换图

