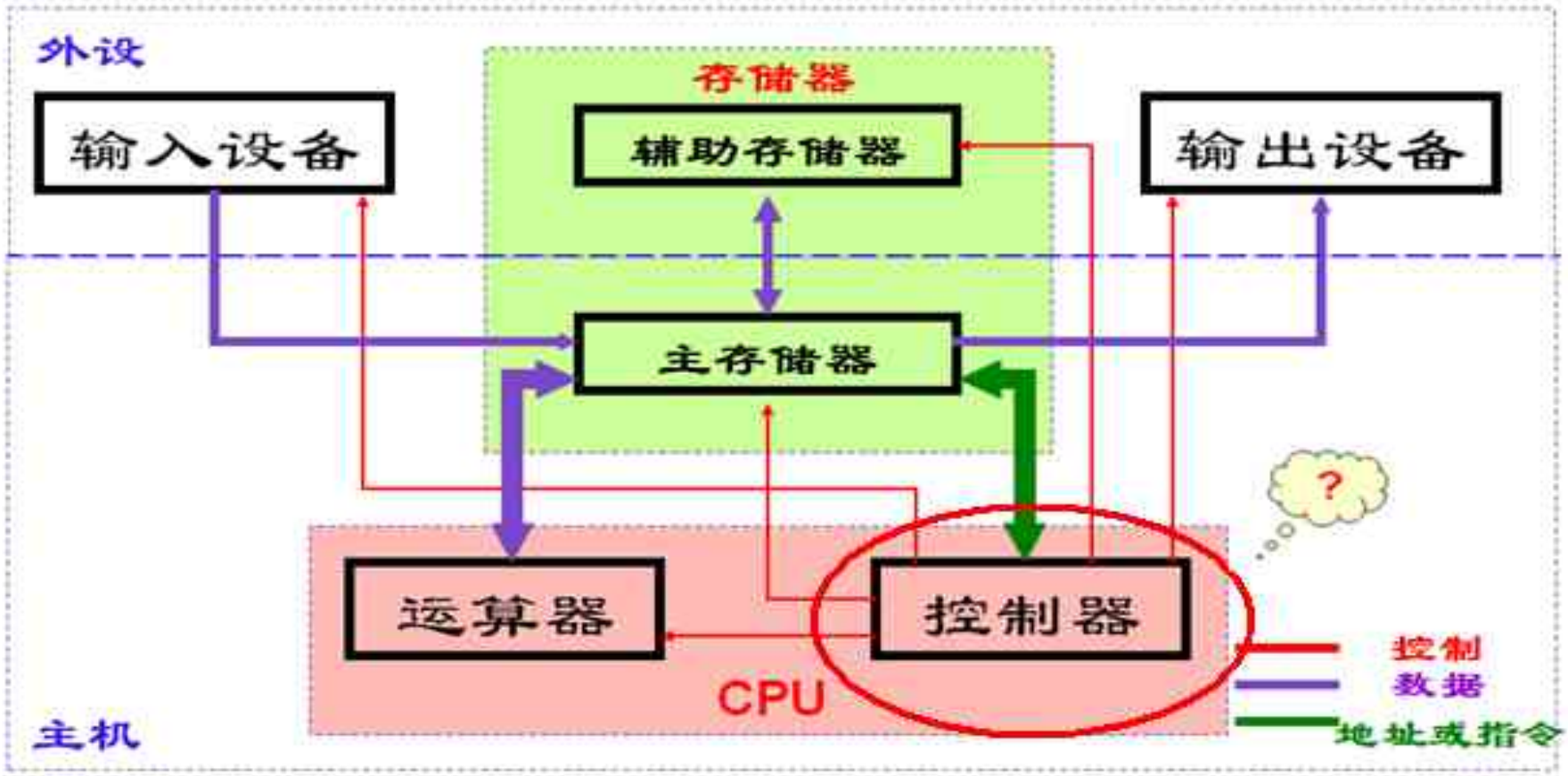
第6章 中央处理机组织

本章学习导读:

(1) CPU的结构、功能、时序控制方式及指令执行过程。

(2) CU的2种实现方式：组合逻辑和微程序控制 。

CPU概念：解释执行机器指令的部件，根据指令的要求指挥全机的工作——计算机系统的指挥中心。包含运算器和控制器两大部分。

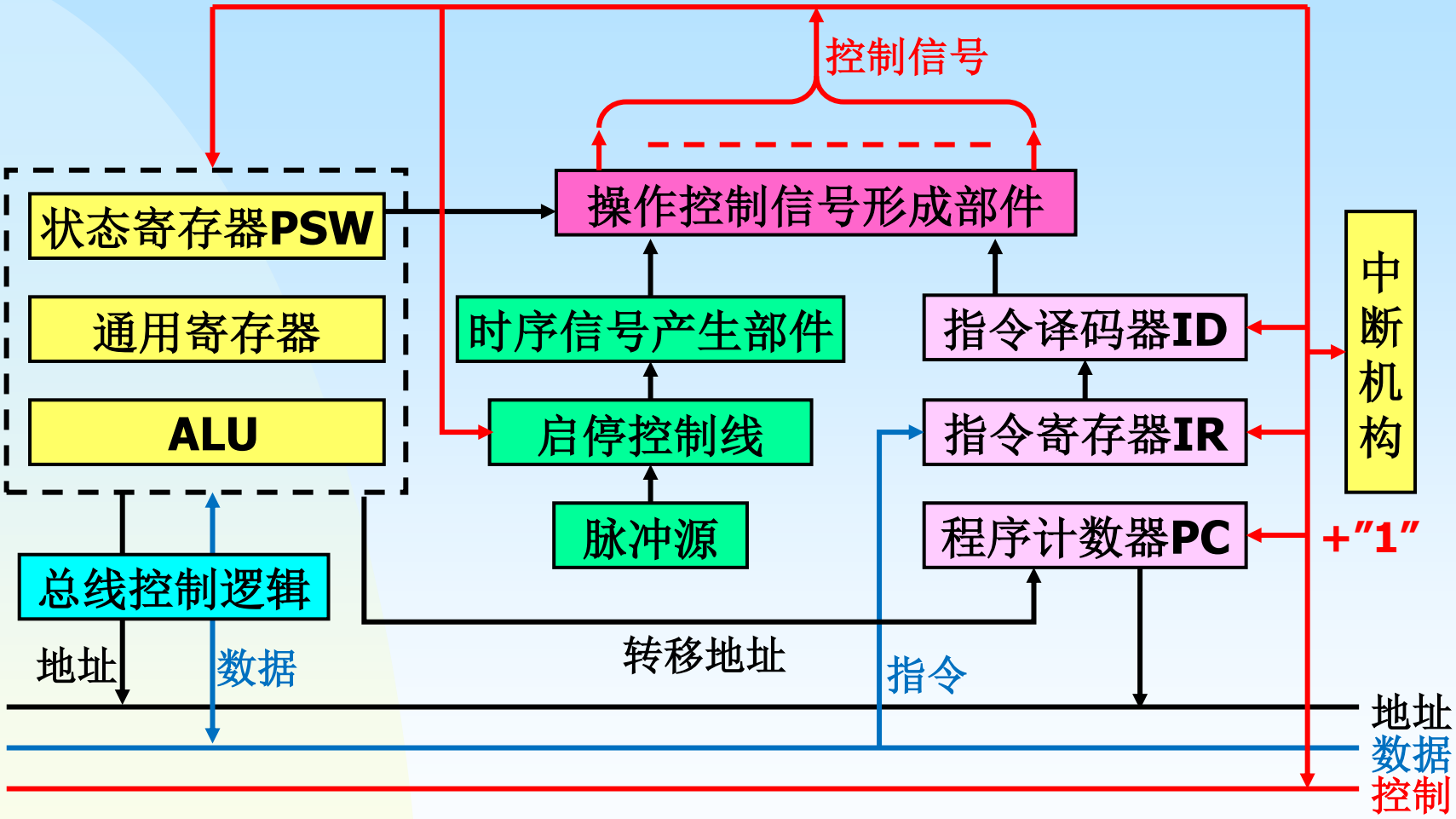


冯诺依曼结构

本质特征：

- 存储程序和指令驱动的顺序执行
- 目前的计算机没有能突破该特征的，都是对冯诺依曼结构的变种（如哈佛结构、并行结构等）

6.1 CPU的基本组成


控制信号
操作控制信号形成部件
状态寄存器PSW
通用寄存器
ALU
时序信号产生部件
启停控制线
脉冲源
指令译码器ID
指令寄存器IR
程序计数器PC
中断机构
+"1"
总线控制逻辑
地址
数据
转移地址
指令
地址
数据
控制

6.1.1 CPU的基本组成与功能

1. 运算部件 ——ALU
2. 寄存器设置 ——通用、专用寄存器
3. 控制器
   1) 指令部件：取指令、分析指令。
   2) 时序部件 ——产生脉冲、节拍等时间信号
   3) 操作控制信号形成部件 ——CU

6.1.2 寄存器设置

CPU内部设置多个寄存器，暂存数据信息和控制信息：

1. 用于处理的寄存器。
2. 用于控制的寄存器。
3. 用作主存接口的寄存器。

1. 用于处理的寄存器

通用寄存器组：

- 每个寄存器设有编号，可编程访问；
- 具有多种功能：提供操作数、存放运算结果、地址指针、变址寄存器、计数器等。

  存放操作数（满足各种数据类型）；

  两个寄存器拼接存放双倍字长数据；

  存放地址，其位数应满足最大的地址范围；

  用于特殊的寻址方式，段基值，栈指针等。

2. 用于控制的寄存器

指令寄存器IR：用户不可见

- 存放现行指令
- 输出用于产生控制信号序列

程序计数器PC： 用户可见

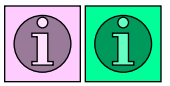
- 提供读取指令的地址(顺序,转移), 又叫指令指针IP
- 相对寻址，支持程序浮动。

指令译码器ID：

- 对IR中的操作码译码，用以产生控制信号。

程序状态字寄存器PSW： 内容表示现行程序的状态

用户可见



程序状态字寄存器PSW：

① 状态标志：记录程序执行状态。

- 一条指令执行后，根据运行结果自动修改标志位的有关内容，作为决定程序流向的因素之一；
- 包括进位C、溢出V、零Z、负N、奇偶位P等.

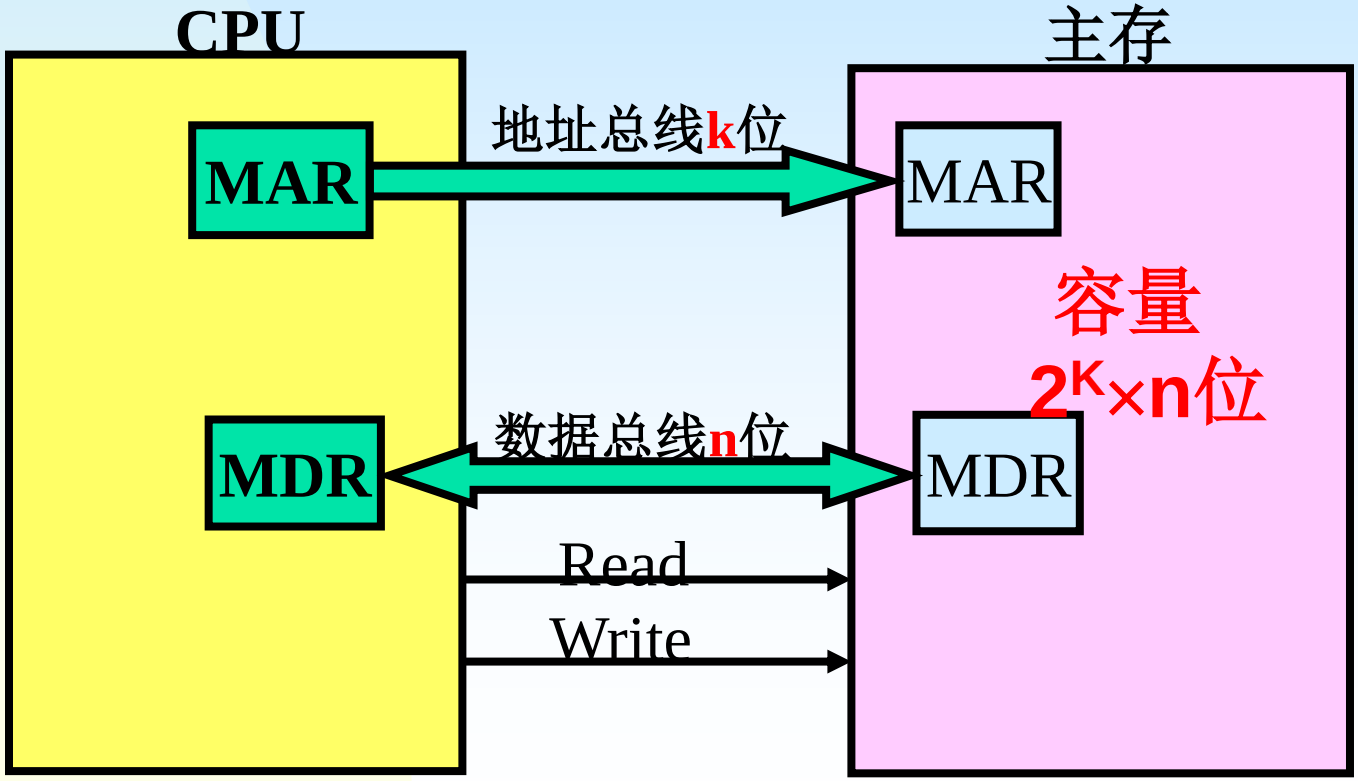
② 控制标志：

- 允许中断I：能否响应外部可屏蔽中断请求
- 陷阱标志T：方便程序调试，使处理器的执行进入单步方式。

3. 用作主存接口的寄存器

CPU访存时，先送出地址码，然后送出数据或接受数据；

设置两个寄存器，用户不能直接编程访问：



CPU基本功能：周而复始地读取并执行指令，包括

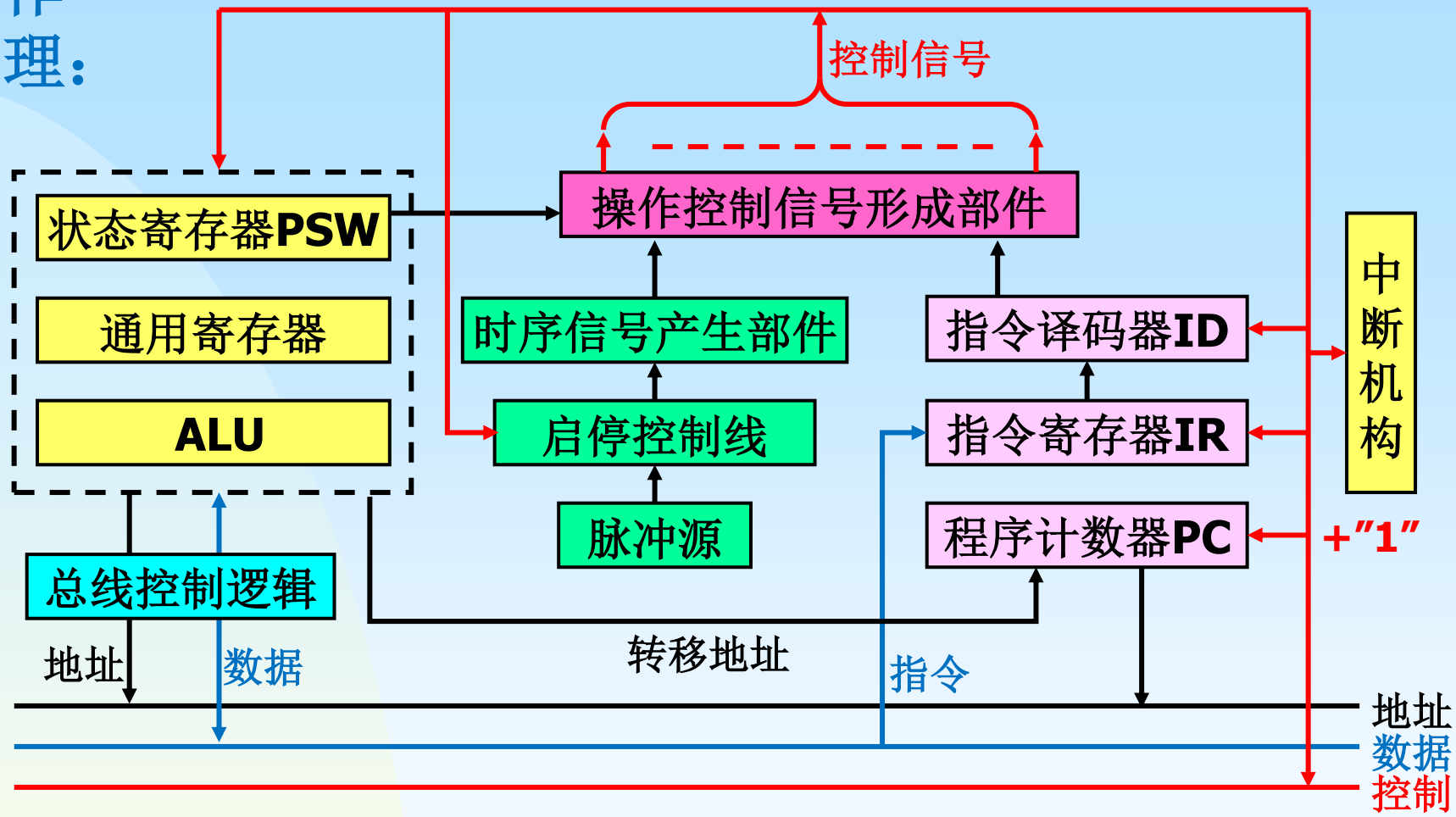
指令控制：内存中读取一条指令，并产生下一条指令在内存中的地址；

操作控制：指令译码，产生各种控制信号送往相应部件，以控制完成指令所要求的动作；

时序控制：对各种操作信号实施时间上的控制，按正确的时序产生操作控制信号，保证计算机有条不紊的连续自动工作；

数据加工：执行所有的算术运算和逻辑运算。

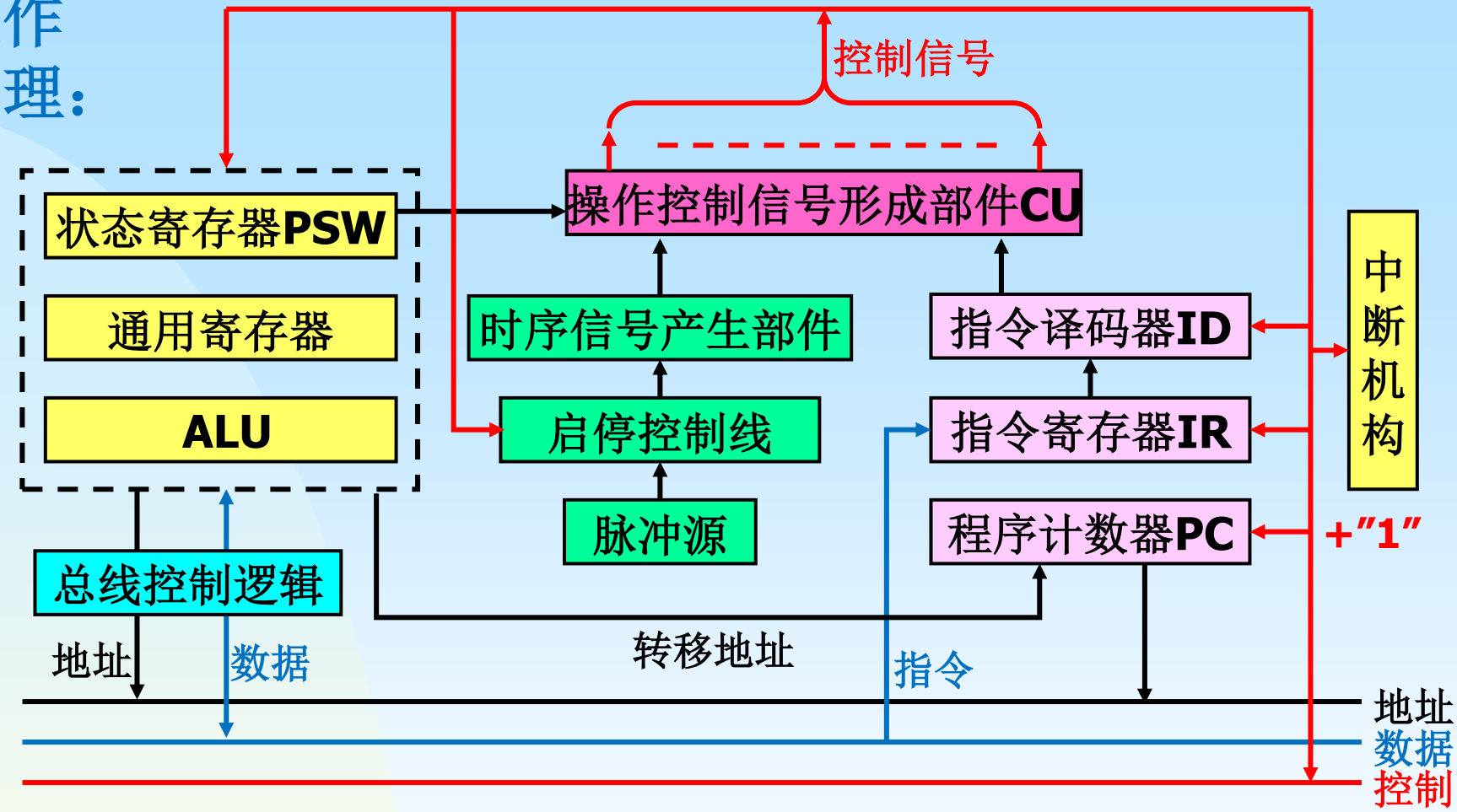
工作原理：



说明：时间、指令代码、状态等信息作为逻辑变量，经操作控制信号形成部件产生控制信号序列。

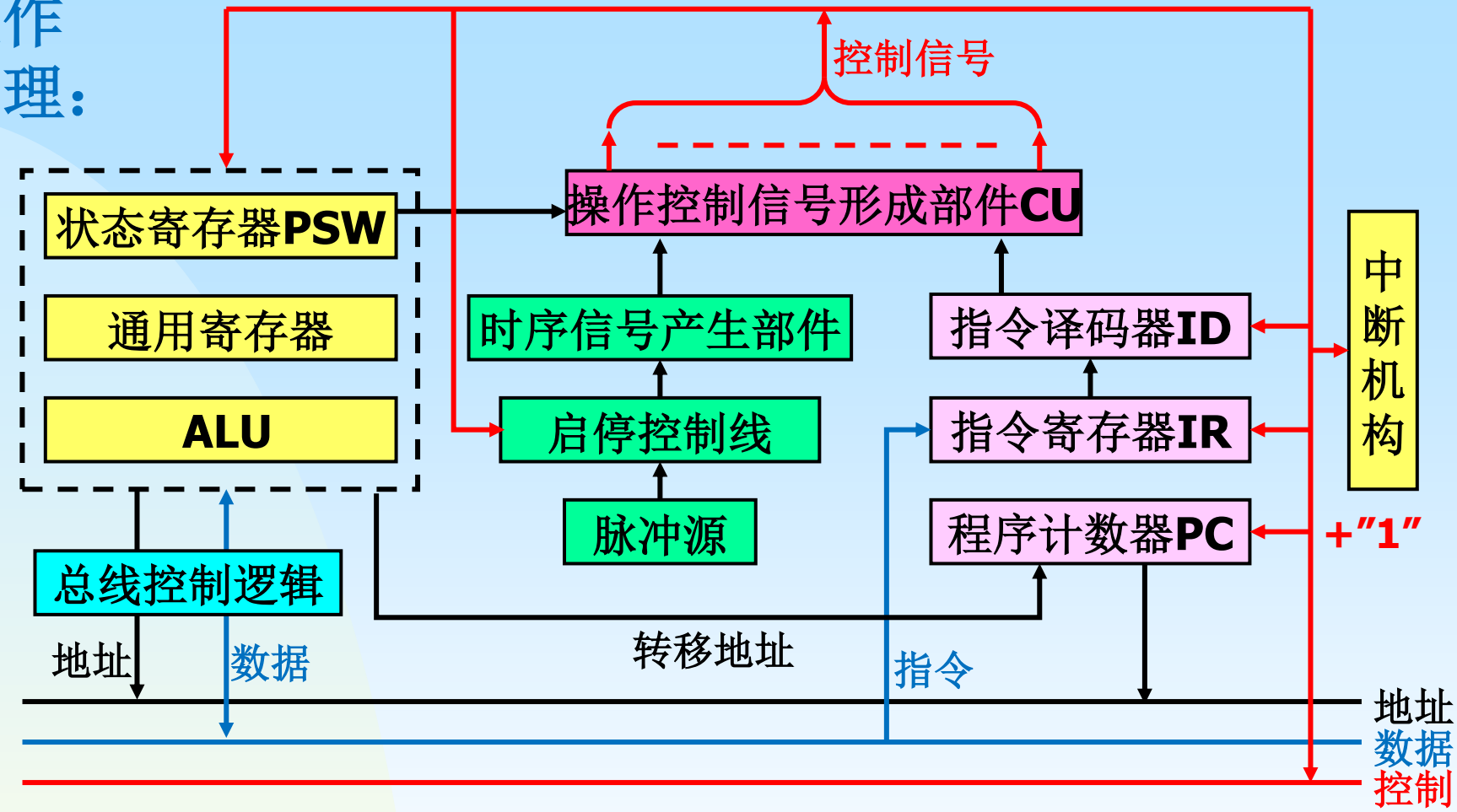
$$C = f(I, T, S)$$

工作原理：



1. 主存中读取的现行指令放在IR中；
2. 操作码经ID形成一些中间逻辑信号，送CU，作为产生控制信号的基本逻辑依据；
3. 控制信号的形成需考虑各种状态信息；

工作原理：



4. 控制信号分时产生，由时序系统提供时序信号(节拍)。

5. 后继指令地址形成：①顺序执行时，PC增量计数②需要转移时，IR地址段信息产生转移地址，送入PC，使程序发生转移。

6.1.3 打断程序正常执行的事件

有时CPU会遇到一些特殊情况，而无法继续执行当前程序：

内部异常(exception)：如ALU运算的结果发生溢出，或除法指令的除数为0，无法继续执行程序。

外部中断(interrupt)：如打印机缺纸。

键盘缓冲区已满，如“翻页”，“退出”等。

CPU必须具有程序正常执行被打断时的处理机制——中断机制——中断机构

CPU转到操作系统中预先设定的，与所发生事件相关的，处理程序执行。

6.2 数据通路基本结构和工作原理

6.2.1 数据通路基本结构

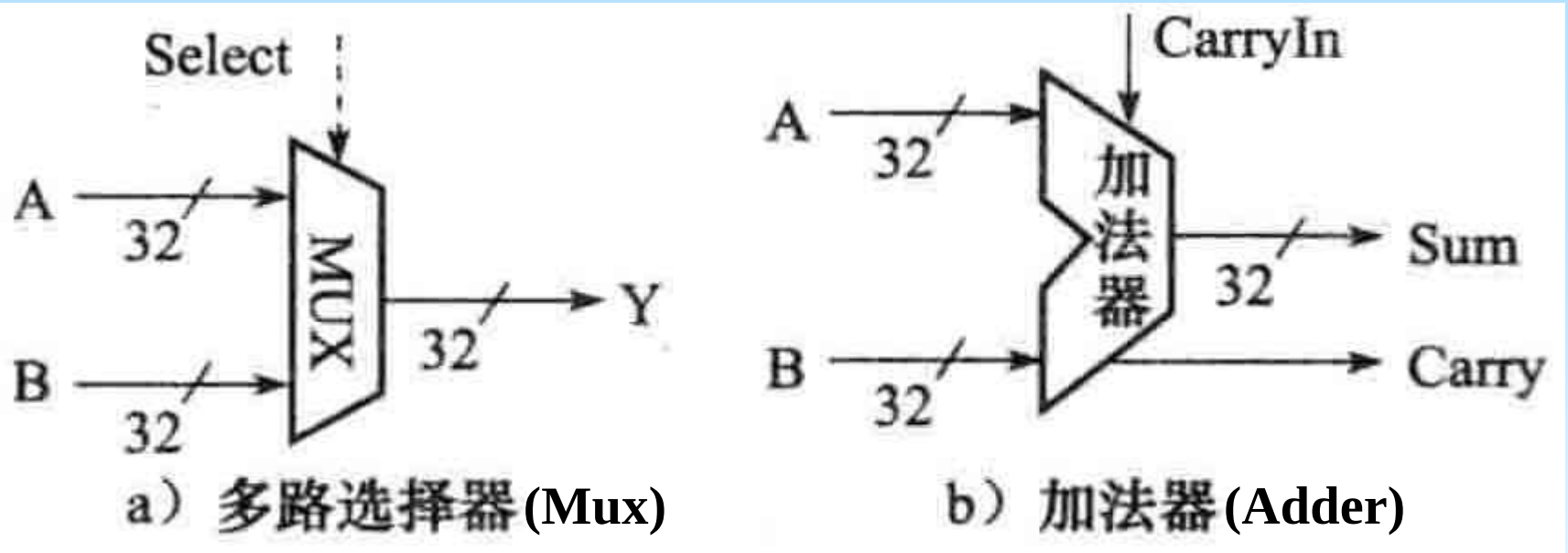
指令执行用到两类元件：

1) 组合逻辑元件（操作元件）
2) 时序逻辑元件（状态元件/存储元件）

数据通路：由操作元件和存储元件连接而成的，数据存储、处理和传送的路径。包括ALU，通用寄存器、Cache等。

指令的执行 ⇔ 数据在通路中的流动

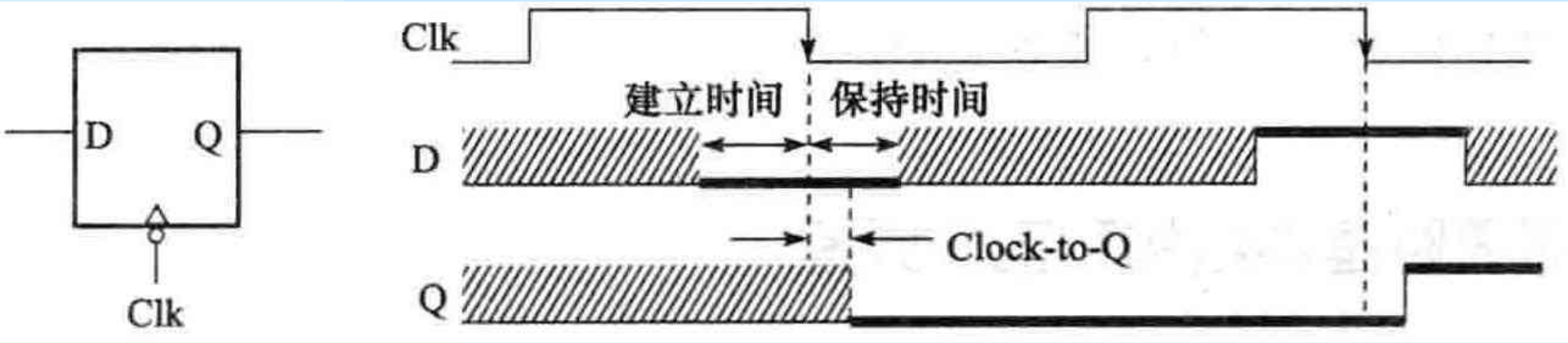
数据通路中的常用组合逻辑元件



时序逻辑元件（状态元件/存储元件）

D触发器：最简单的状态单元。

存储功能：输入状态在时钟控制下写到电路中，**保持输出值**到下一个时钟。



必须满足时间约束，

输出状态能正确的随着输入状态改变。

寄存器：状态存储元件。

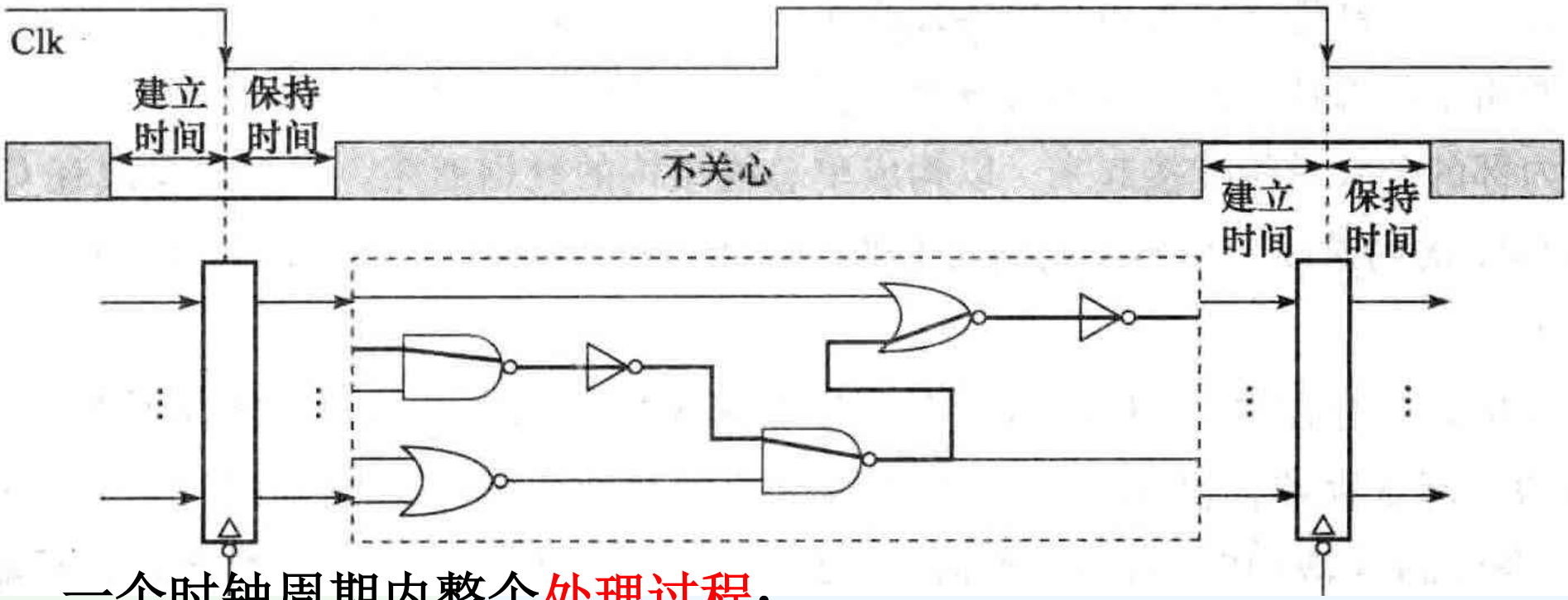
6.2.2 数据通路的时序控制

时钟信号：是整个数据通路的定时信号。

一个时钟周期 = 一个节拍

数据通路的基本结构：交替组合

…——状态元件——操作元件——状态元件…



一个时钟周期内整个处理过程：

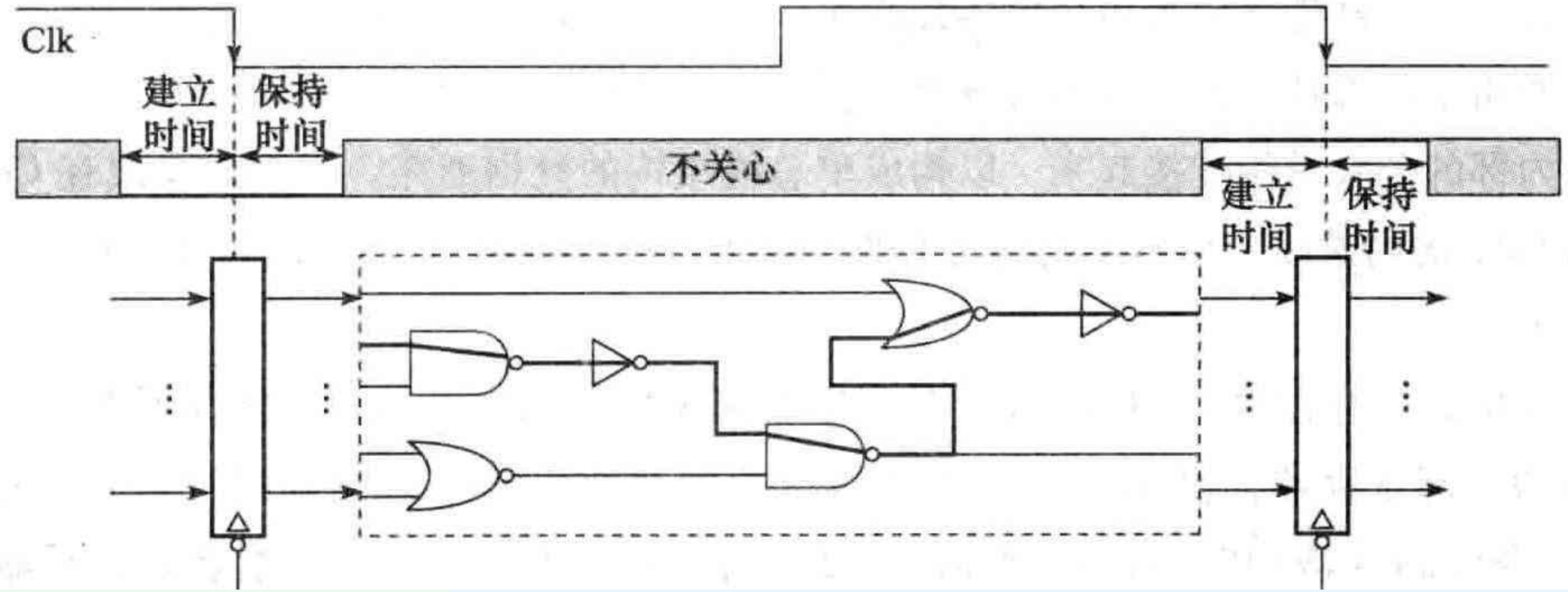
经Clk-to-Q时间，前一周期生成的信号被写入状态元件，

并输出到随后的操作元件进行处理，门延迟，处理结果送到下一级状态元件输入端，

稳定一段时间（建立时间），开始下一个周期，

下降沿后保持一段时间（保持时间）。

数据通路的时钟周期（宽度的确定）



Longest Delay：所有各级操作元件中, 最长的操作延迟时间。

时钟偏移（时钟扭斜）：工艺、走线等原因。

数据通路的时钟周期：

Cycle Time = Clk-to-Q + Longest Delay +建立时间 + 时钟偏移

6.2.3 单周期数据通路

指令周期：读取并执行一条指令所需的时间。

单周期处理器：所有指令的指令周期都为一个时钟周期。

单周期数据通路：单周期处理器中的数据通路。

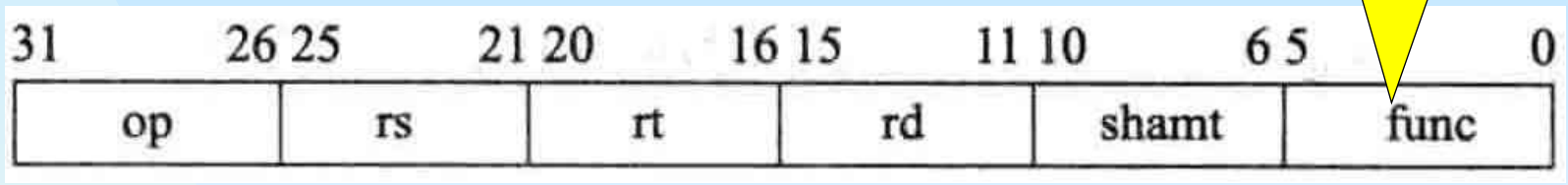
在数据通路中，取指令、指令译码、取操作数、执行运算、存结果，更新PC等所有操作都需在一个时钟周期内完成。

以MIPS指令系统为例。

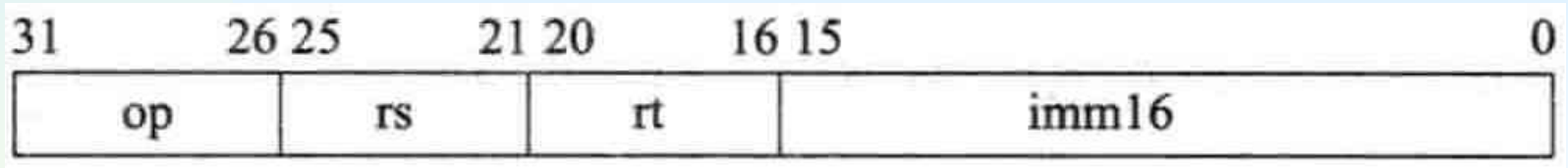
MIPS指令系统

32位定长指令，按字节编址，共3种指令格式：

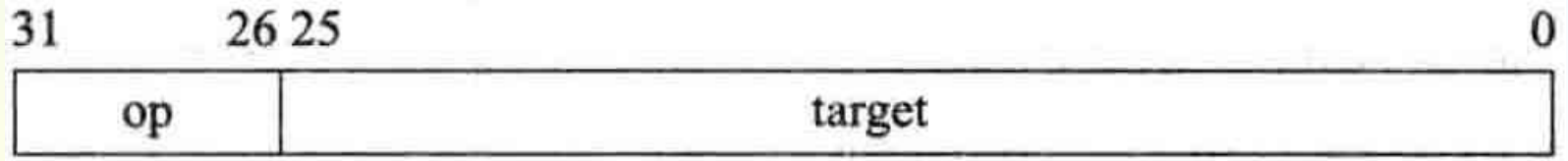
R型指令：



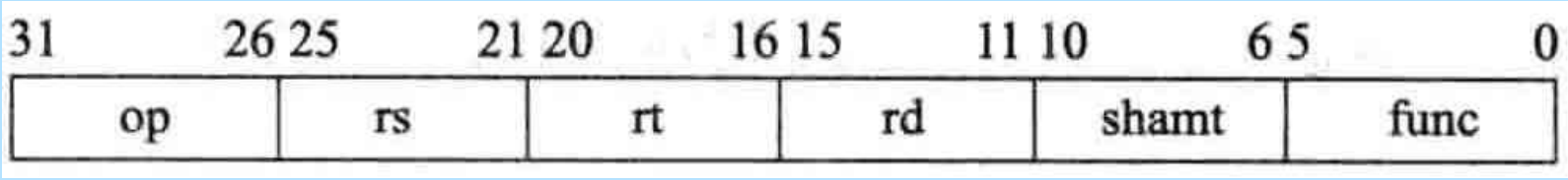
I型指令：



J型指令：



R型指令



ALU运算

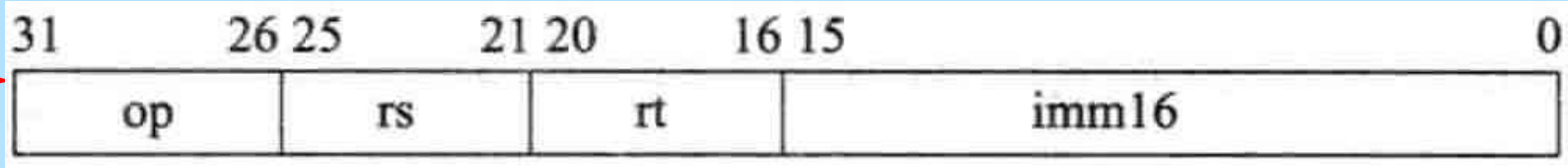
汇编形式：op(func) rd, rs, rt

功能：R[rd] ← R[rs] op(func) R[rt]

如，sub $8, $9, $10

R[8] ← R[9] - R[10]

I型指令

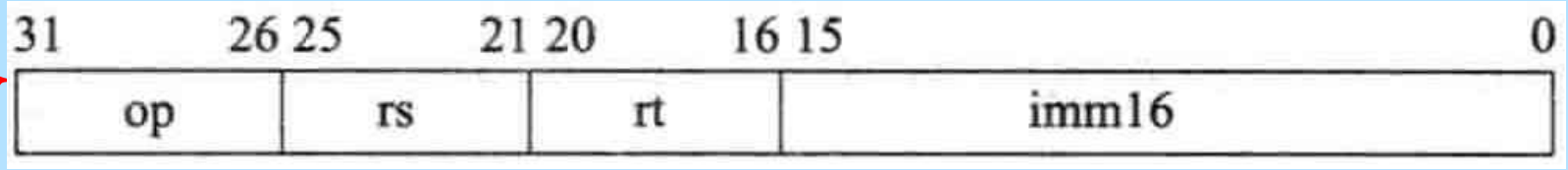


1) 带立即数的运算指令：

汇编形式如：**addi rt, rs, imm16**

功能：对16位立即数imm16进行符号扩展或零扩展，
与rs内容进行运算，
ALU运算结果送rt。

I型指令

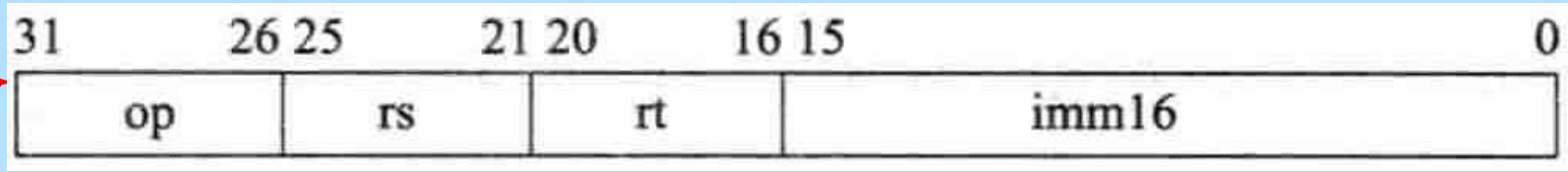


2) lw指令

汇编形式：**lw rt, imm16(rs)**

功能： **R[rt] ← M[R[rs] + SignExt(imm16)]**

I型指令



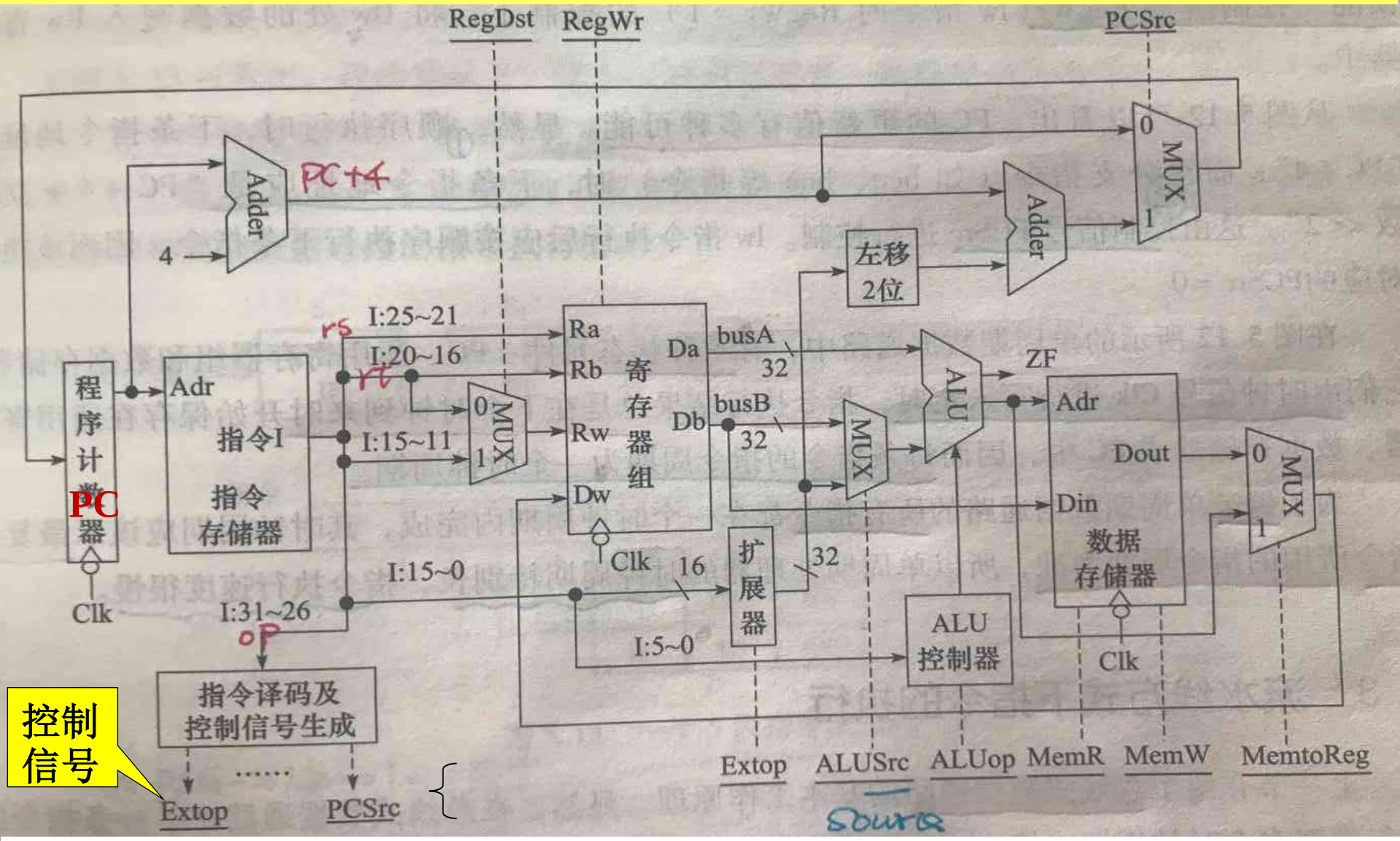
3) beq指令（分支指令/条件转移指令）

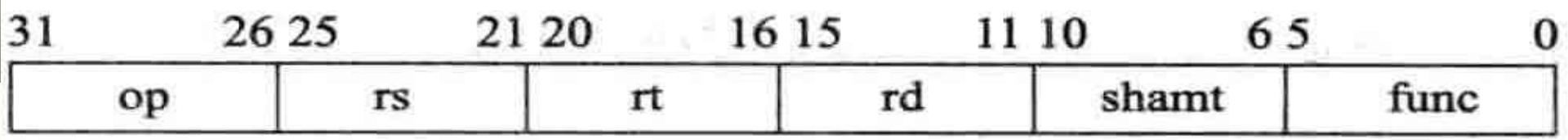
汇编形式：**beq rs, rt, imm16**

功能： 若相等则转移

```
if R[rs]=R[rt]
    PC ← PC + 4 + 4×imm16
else
    PC ← PC + 4
```

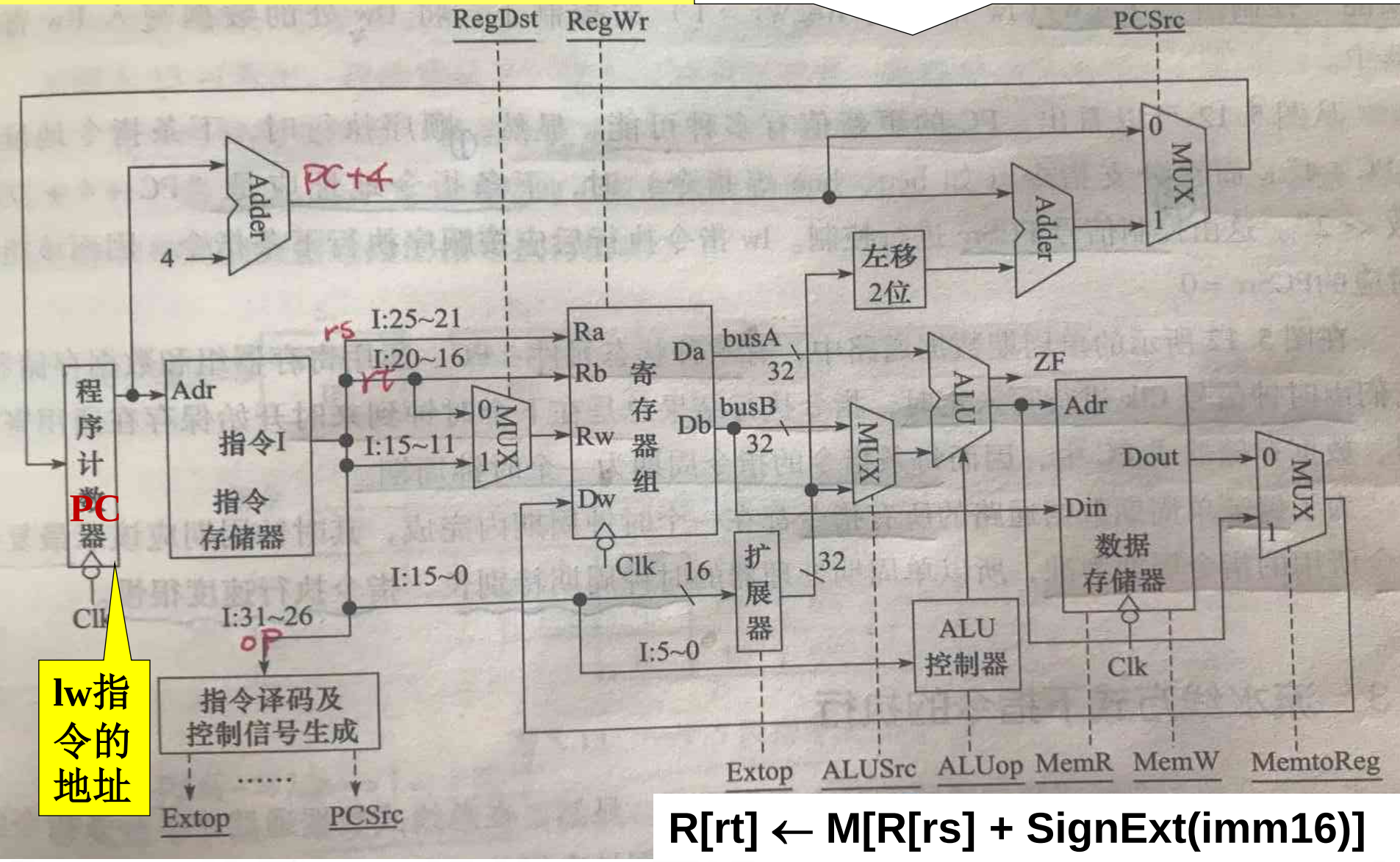
执行MIPS指令的单周期数据通路



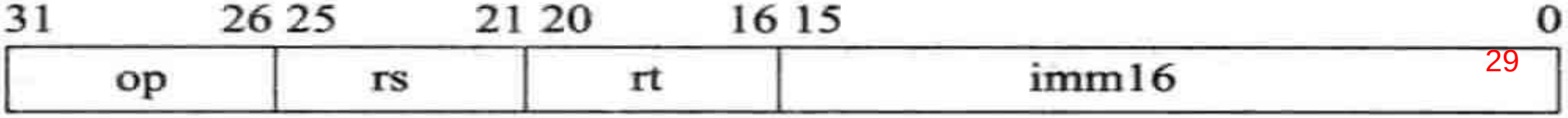


例：lw指令的执行过程

1. 取指令并执行PC+4。

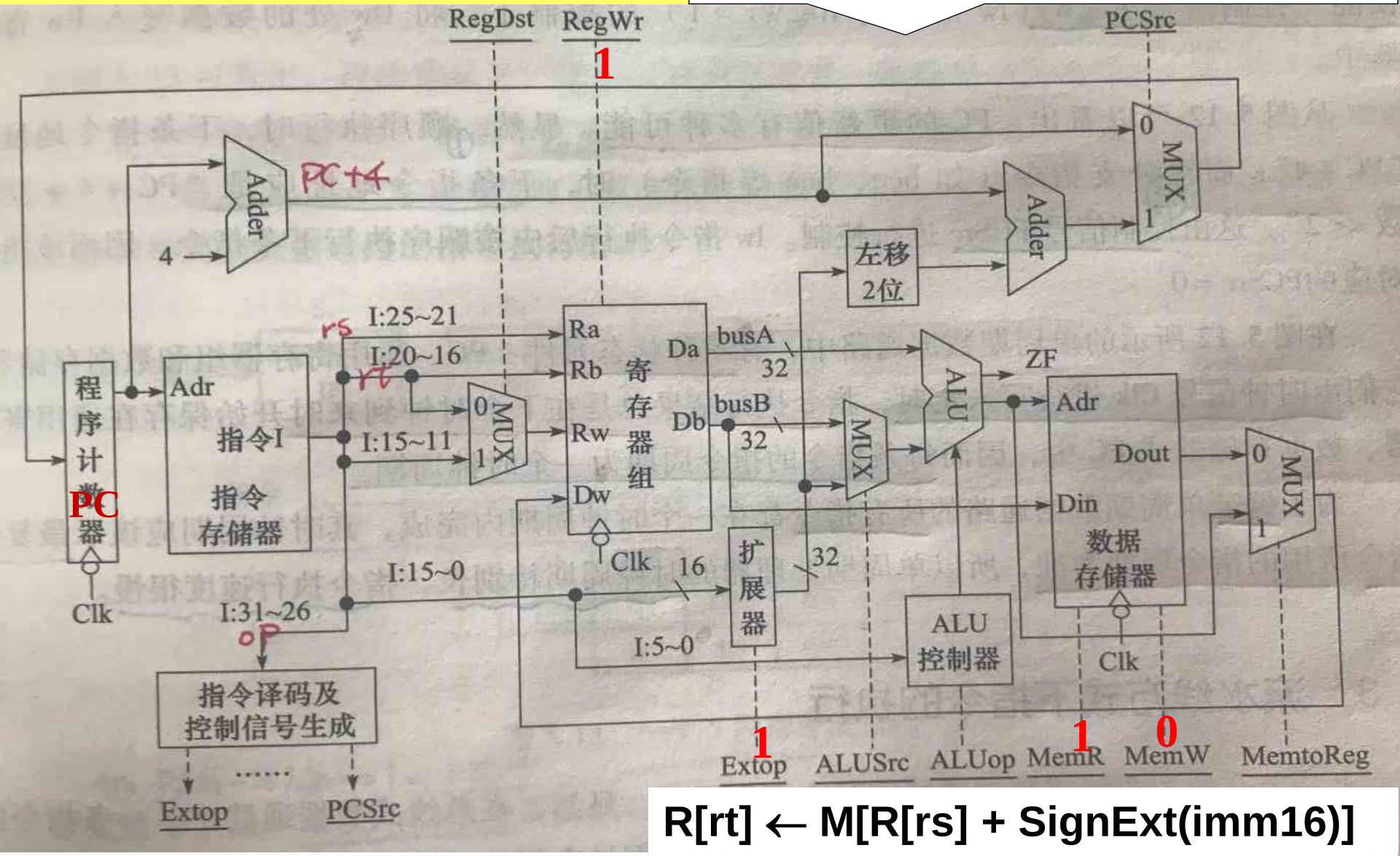


R[rt] ← M[R[rs] + SignExt(imm16)]

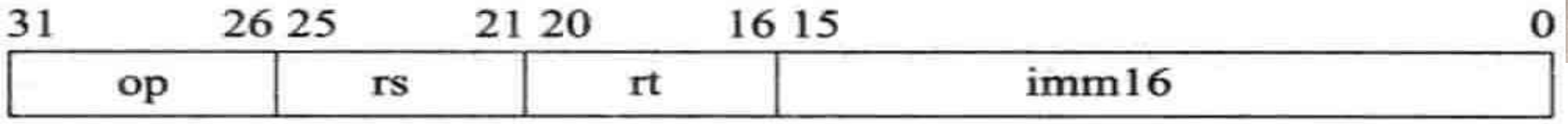


例：lw指令的执行过程

2. 指令译码并取操作数。

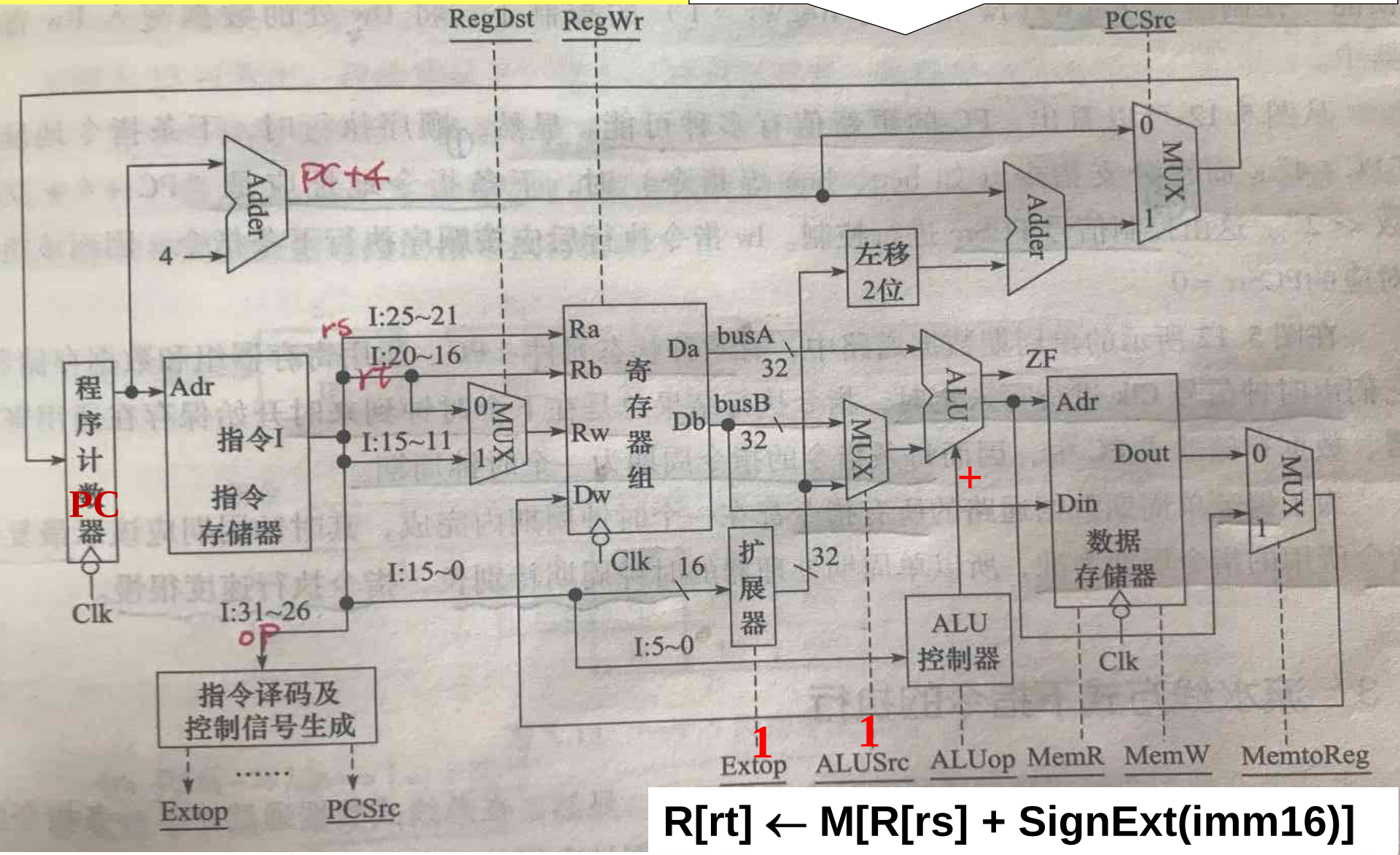


R[rt] ← M[R[rs] + SignExt(imm16)]

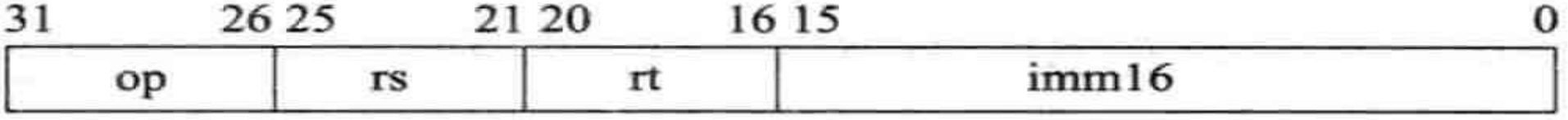


例：lw指令的执行过程

3. 计算操作数地址。

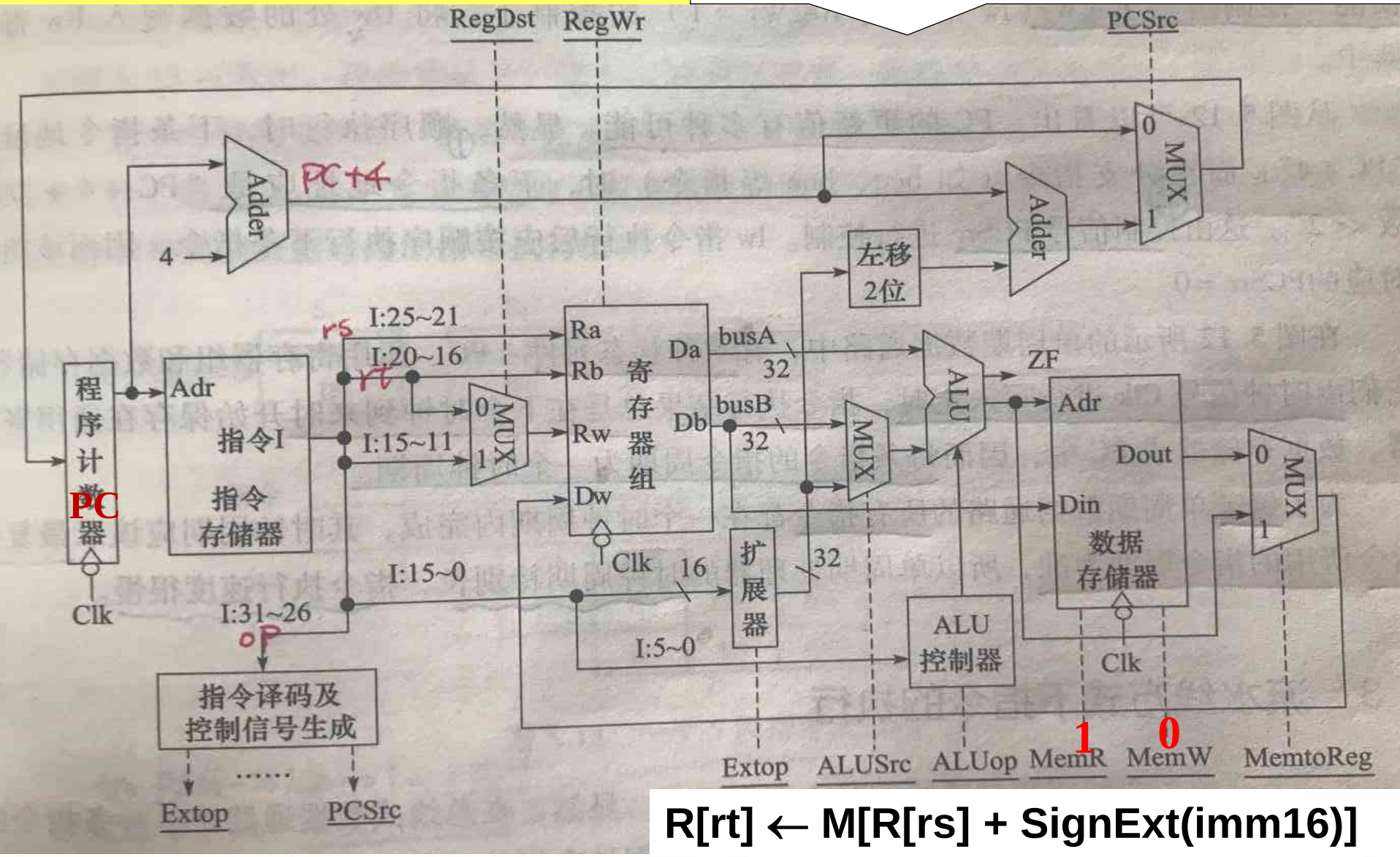


R[rt] ← M[R[rs] + SignExt(imm16)]

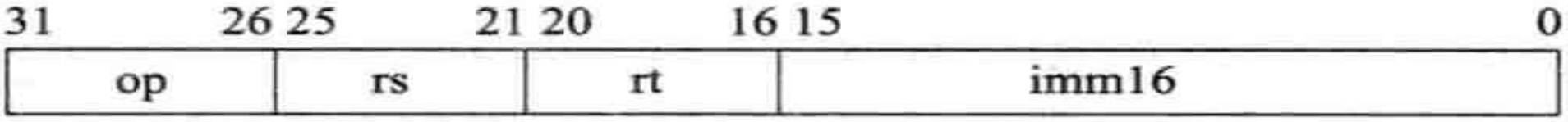


例：lw指令的执行过程

4. 读出操作数。

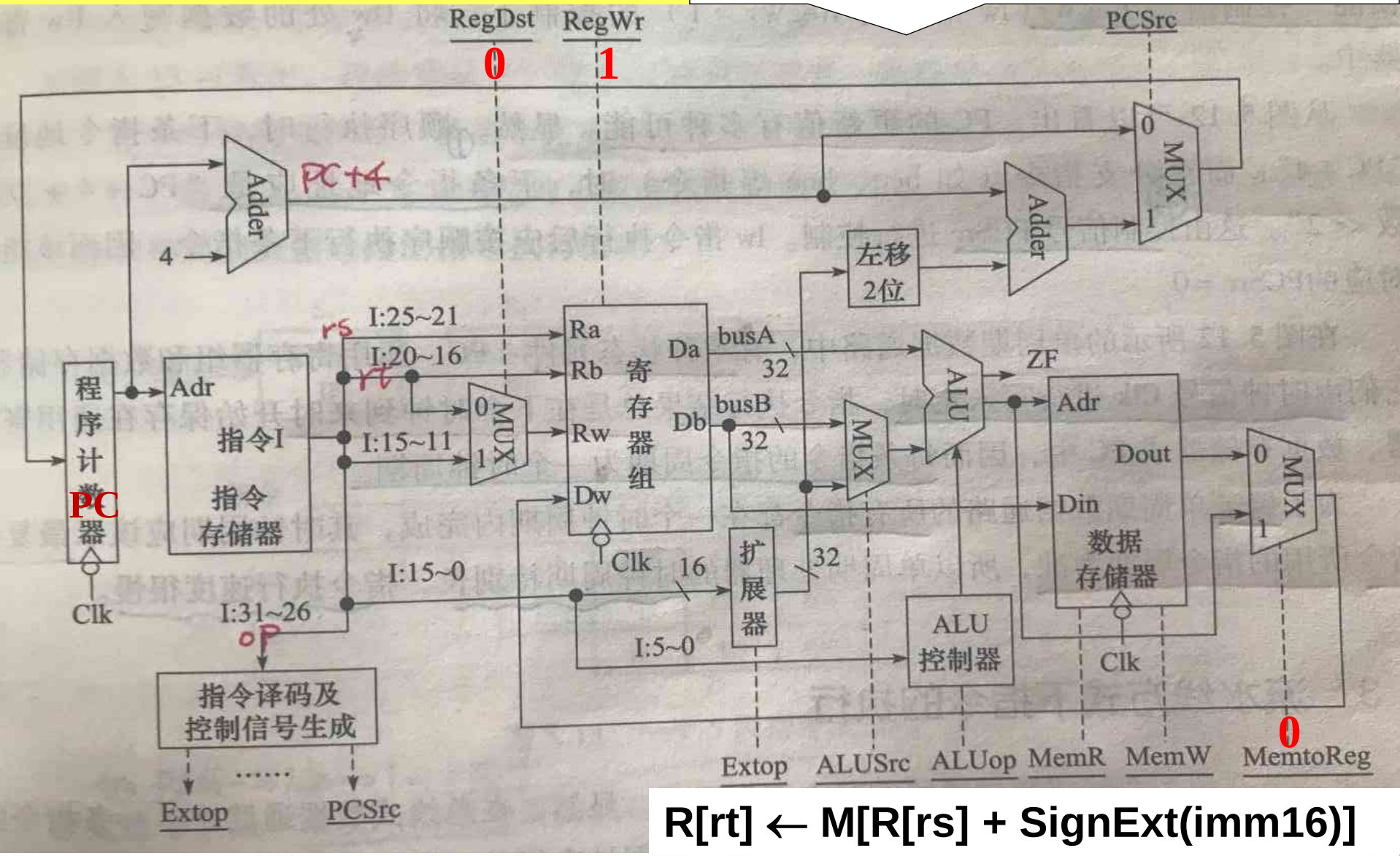


R[rt] ← M[R[rs] + SignExt(imm16)]

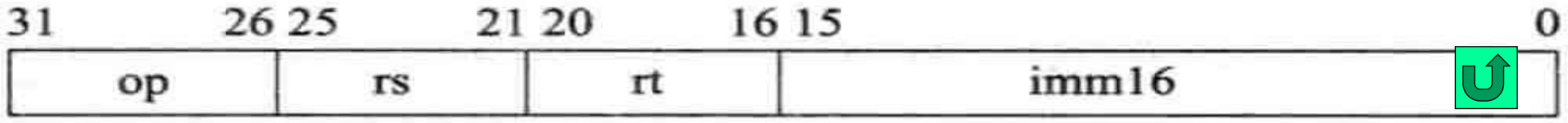


例：lw指令的执行过程

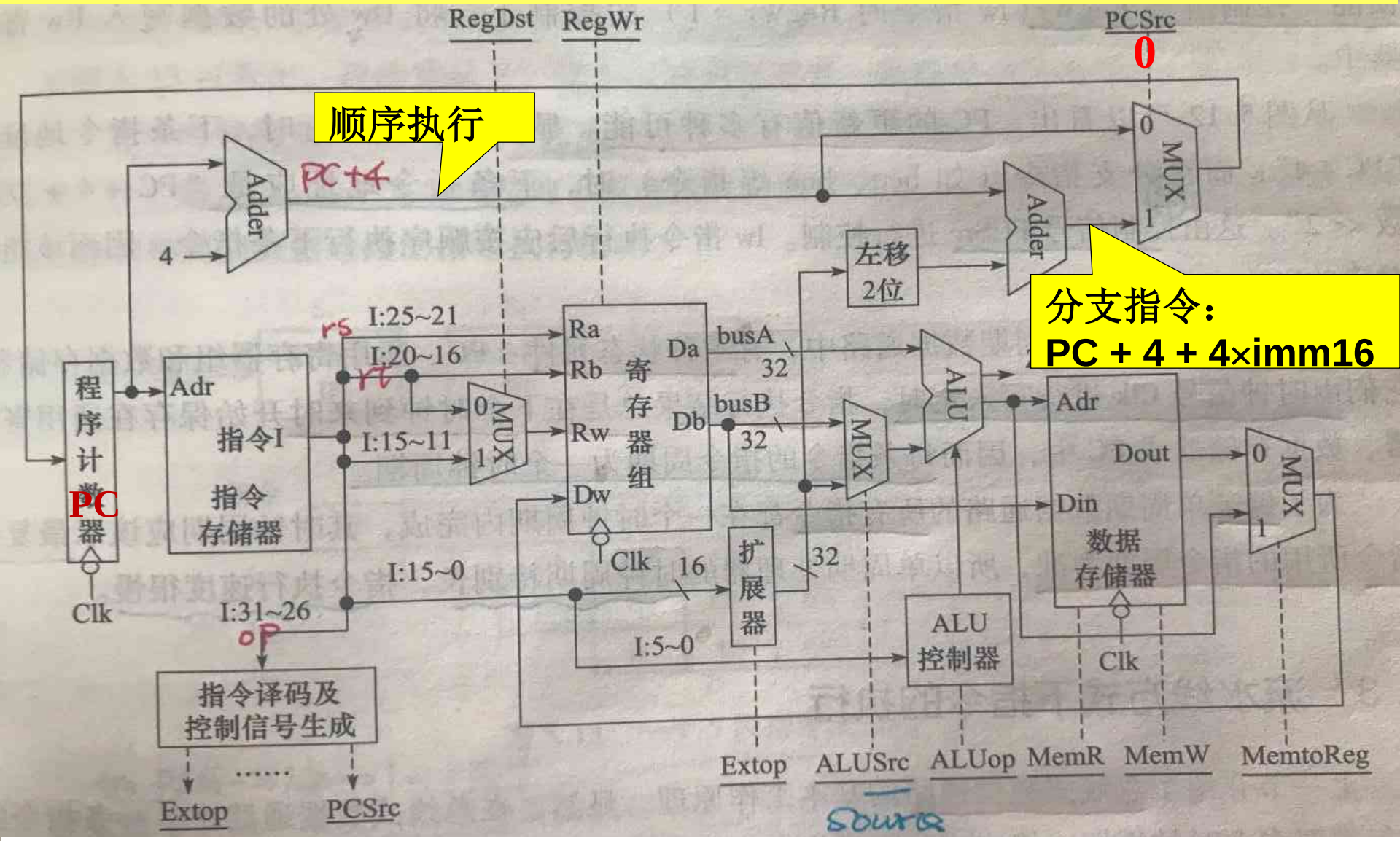
5. 写结果。



R[rt] ← M[R[rs] + SignExt(imm16)]

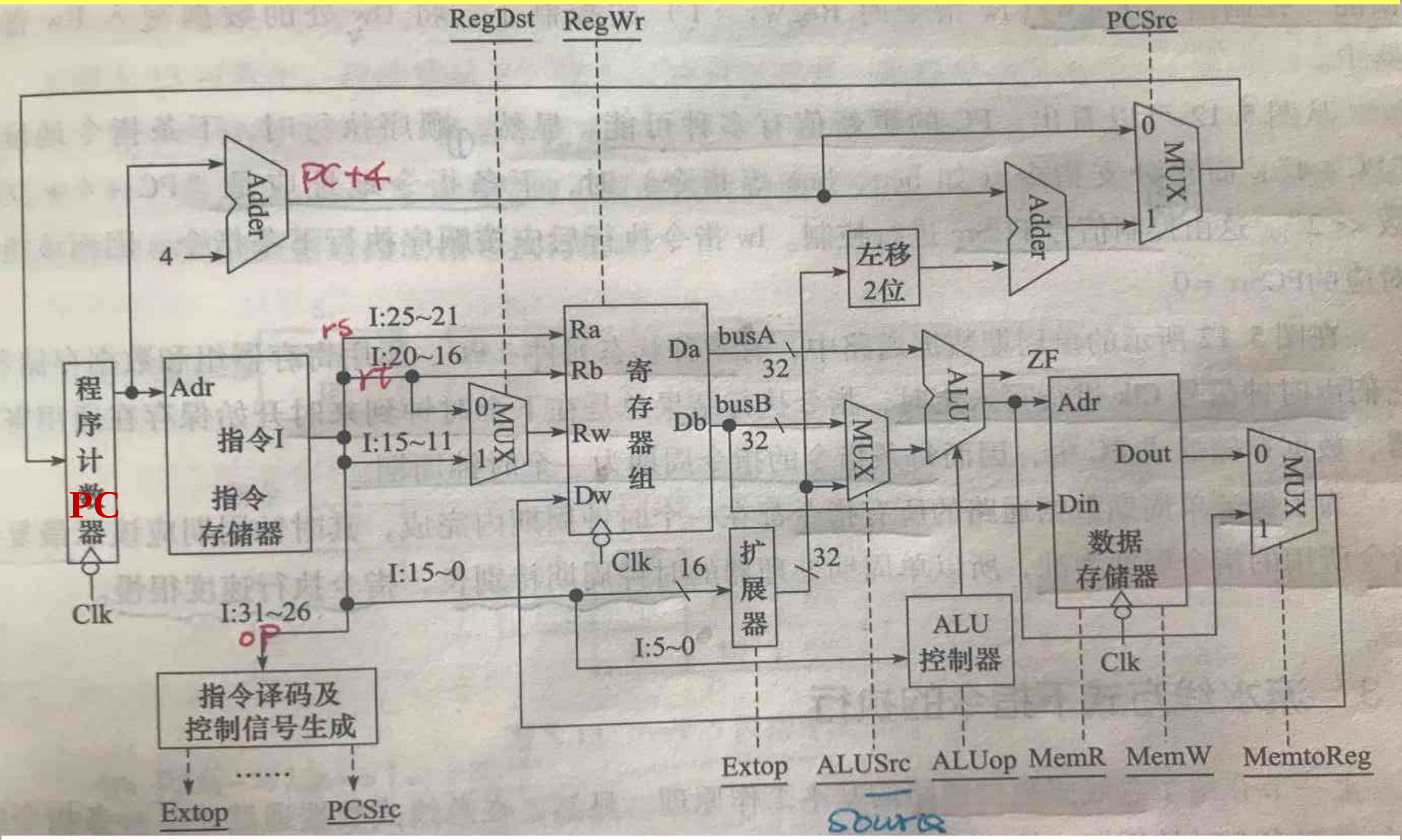


例：lw指令的执行过程



31 26	25 21	20 16	15 11	10 6	5 0
op	rs	rt	rd	shamt	func

执行MIPS指令的单周期数据通路



三种状态元件: 统一的时钟信号Clk进行写入定时, 下个下降沿写入。

每条指令的指令周期都为一个时钟周期。

6.3 流水线方式下指令的执行

串行执行指令：没有充分利用执行部件的并行性，指令执行效率低。

流水线方式执行指令：多条指令的执行相互重叠起来，提高CPU执行指令的效率。

6.3.1 指令流水线的基本原理

一条指令的执行可被分成若干个阶段：

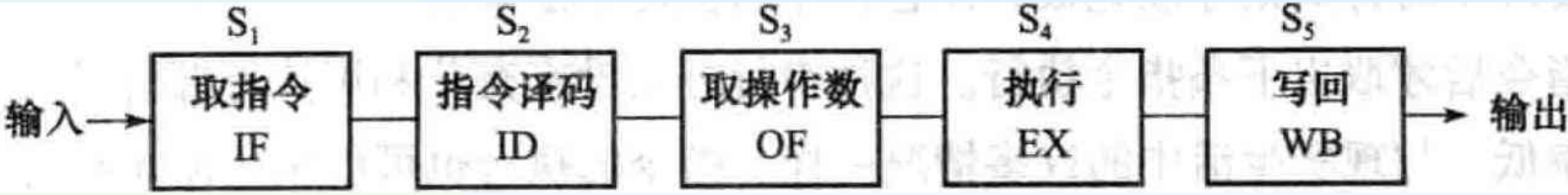
取指令(IF)：根据PC值从存储器取出指令。

指令译码(ID)：产生指令执行所需的控制信号。

取操作数(OF)：读取存储器操作数或寄存器操作数。

执行(EX)：对操作数完成指定操作。

写回(WB)：将操作结果写入存储器或寄存器。

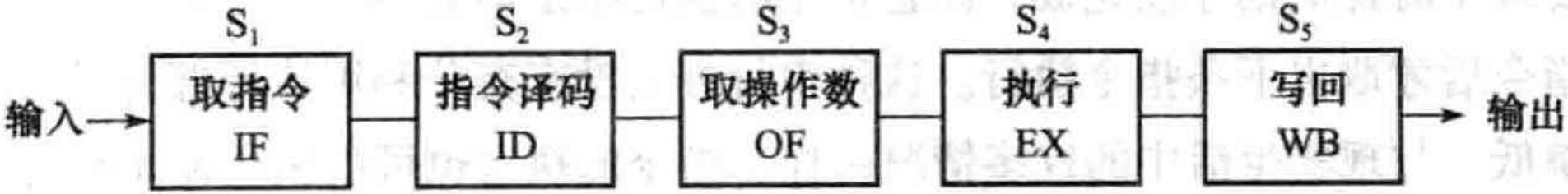


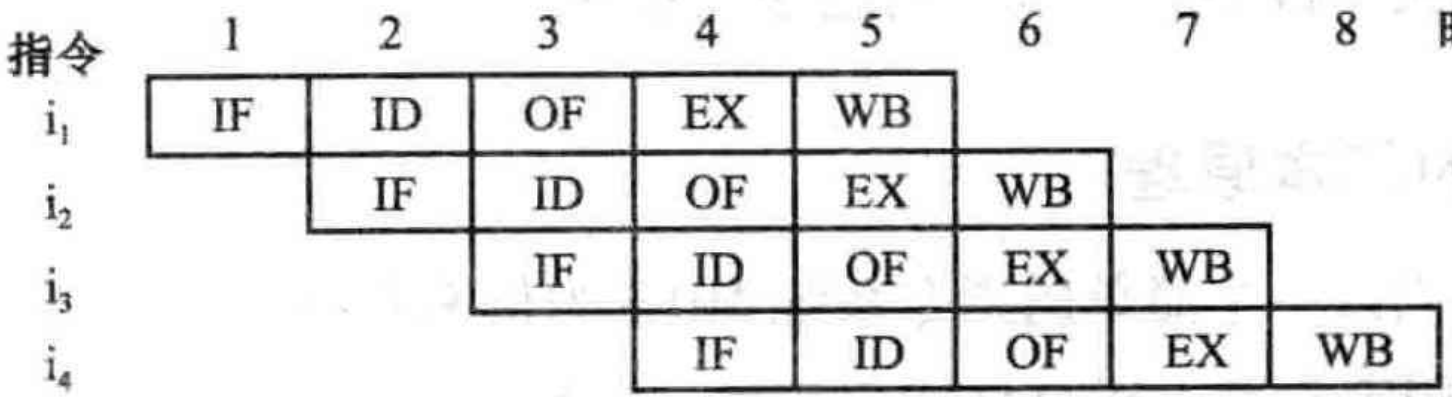
每个阶段都在相应的功能部件中完成，

5个流水段，5个部件，

指令的执行过程——指令流水线

一个5段指令流水线





完成4条指令：

流水执行：__8__个周期

串行执行：__20__个周期

理想情况：

CPI = 1

一个5段指令流水线

设指令系统中最复杂的指令需用5个阶段完成：

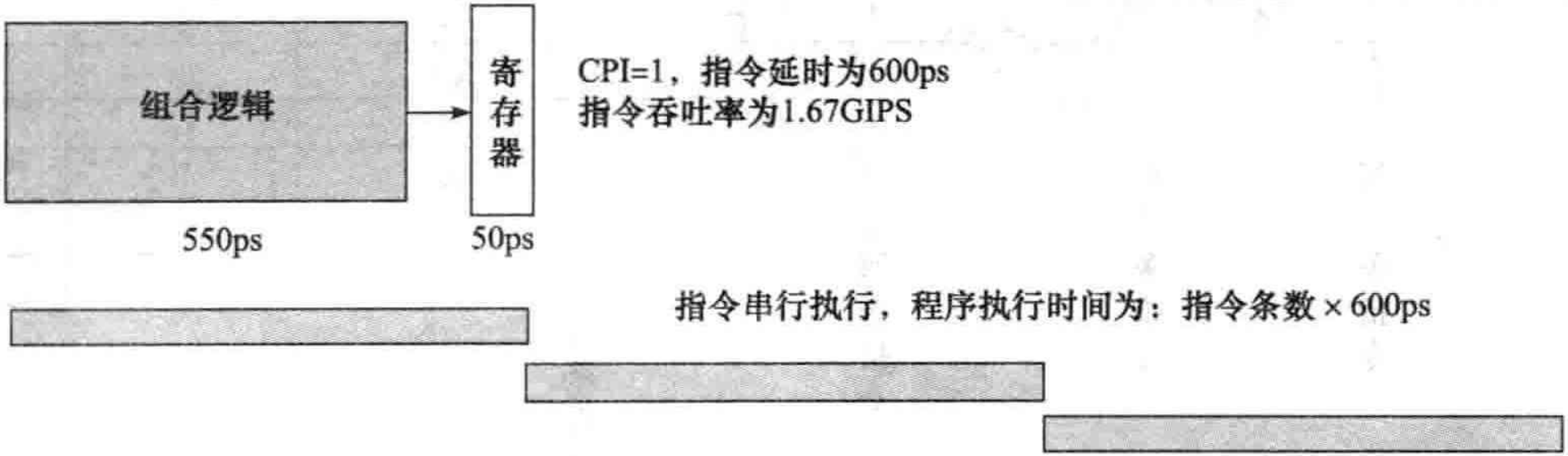
①取指令: 200ps　　②译码和读操作数: 50ps

③ALU操作: 100ps　　④读存储器: 200ps

⑤ 结果写寄存器: 50ps

该指令总执行时间：200+50+100+200+50=600ps

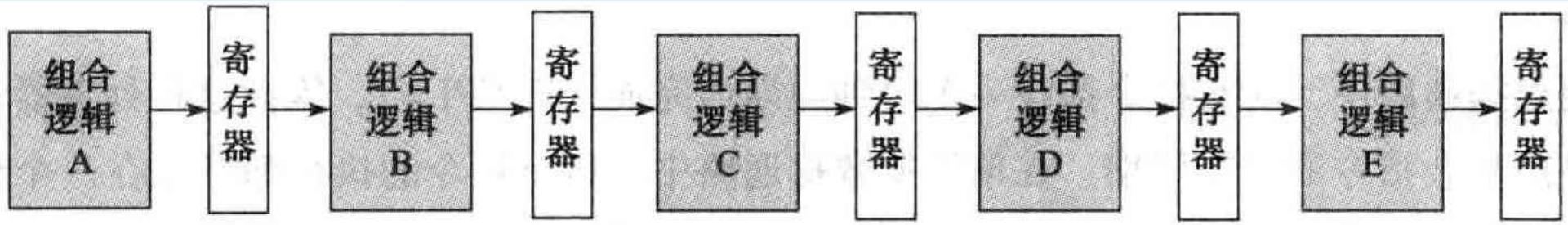
单周期数据通路的简化结构及指令执行过程：



一个5段指令流水线

5个阶段，5个功能部件完成 ⇒ 5个流水段：

每个流水段：一个组合逻辑 + 一个寄存器



流水线数据通路设计原则

流水段个数：最复杂指令的功能段个数。

流水段长度：最复杂功能段的操作所用的时间。

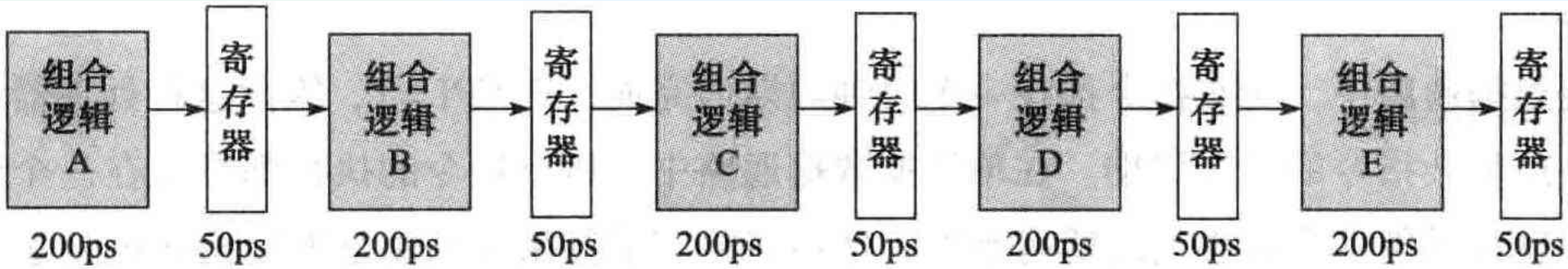
上述指令系统的流水线数据通路：

最复杂指令：5个功能段

最复杂功能段时间：200ps

时钟周期：200+50 = 250ps

指令周期: 250×5=1.25ns



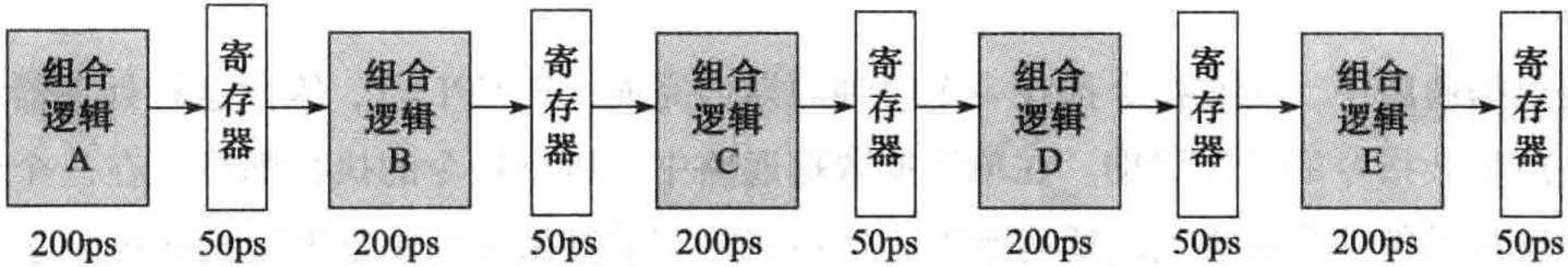
CPI=1，指令延时为1.25ns，指令吞吐率为4GIPS

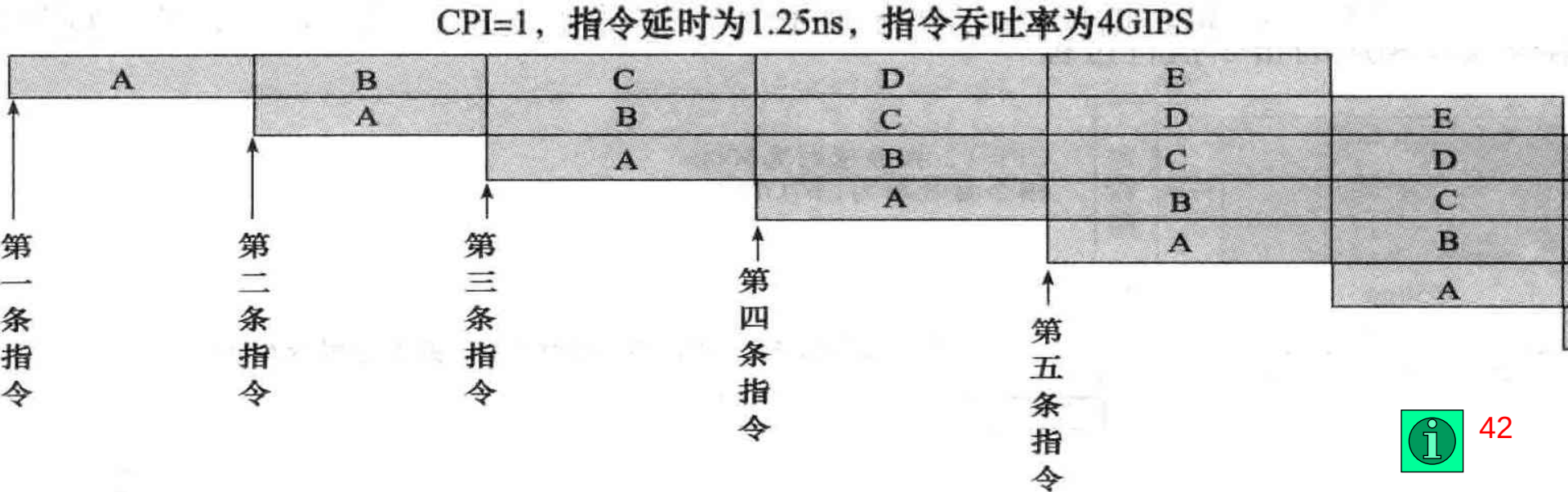
一个5段指令流水线

时钟周期: 200+50 = 250ps　　指令周期: 250×5=1.25ns

统一的时序控制，顺畅流动，各段完全并行工作。

CPI=1　　指令吞吐率=1/(250×10^{-12}) = 4GIPS





6.3.2 适合流水线的指令集特征

流水线方式执行指令的关键：

所有指令的执行过程都分成相同数目的功能段，

每个功能段的执行时间都相同。

适合流水线的指令集特征：

1、指令长度应尽量一致。简化取指、下址计算、指令译码。

2、指令格式应尽量规整，尽量保证源操作数寄存器的位置相同。可同时进行指令译码和读取寄存器内容。

3、采用load/store型指令风格。

load/store型指令风格: 指令集中只有load指令和store指令能访问存储器，其它指令一律不能访问。

6.3.2 适合流水线的指令集特征

load/store型指令风格:

每条指令的功能强弱比较一致，

load/store指令的地址计算和运算指令的执行规整在同一流水段。

其它指令执行阶段都不访问存储器，利于减少步骤，规整流水线。

IA-32非load/store型风格，运算指令的操作数可以是存储器，指令流程（链接原来的），

简单指令、复杂指令功能段（节拍）数量差距大，不利于流水线规划。

6.3.2 适合流水线的指令集特征

4、为了便于以流水线方式执行指令，数据和指令在存储器中要“对齐”存放。

利于减少访存次数，一个流水段内得到所需数据。

（之前课件的图）

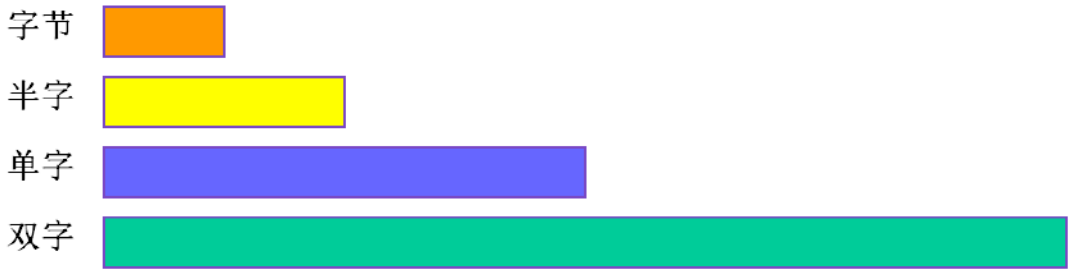
规整、简单和一致的指令系统有利于流水线执行。

主存储器的连接与控制

数据在主存中的存放

1．内存中对齐问题

在采用字节编址的情况下，数据在主存储器中的三种不同存放方法。假设，存储字为64位（8个字节），读/写的数据有四种不同长度，它们分别是字节（8位）、半字（16位）、单字（32位）和双字（64位）

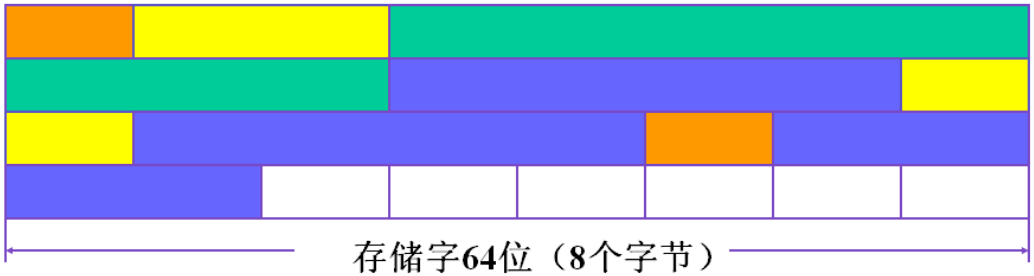


现有一批数据，它们依次为：字节、半字、双字、单字、半字、单字、字节、单字。

(1)不浪费存储器资源的存放方法

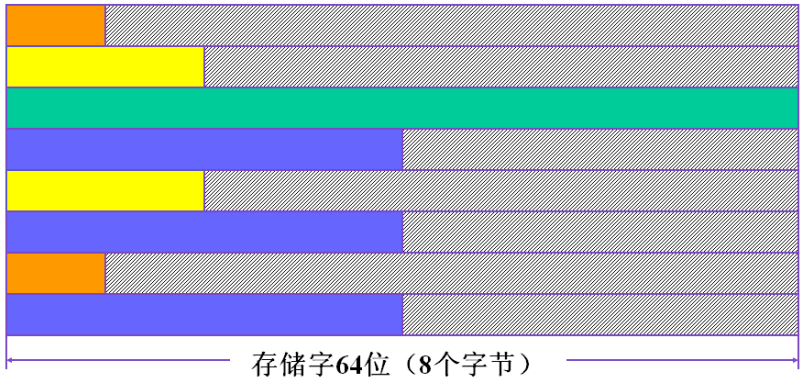
四种不同长度的数据一个紧接着一个存放。优点是不浪费宝贵的主存资源，但存在的问题是：当访问的一个双字、单字或半字跨越两个存储字时，存储器的工作速度降低了一倍，而且读写控制比较复杂。

主存储器的连接与控制



(2)从存储字的起始位置开始存放方法

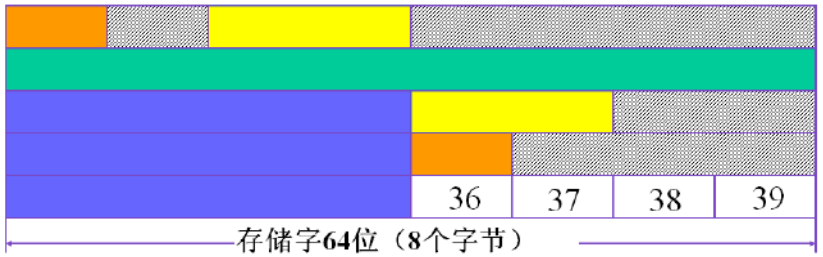
无论要存放的是字节、半字、单字或双字，都必须从存储字的起始位置开始存放，而空余部分浪费不用。优点是：无论访问一个字节、半字、单字或双字都可以在一个存取周期内完成，读写数据的控制比较简单。缺点是：浪费了宝贵的存储器资源。



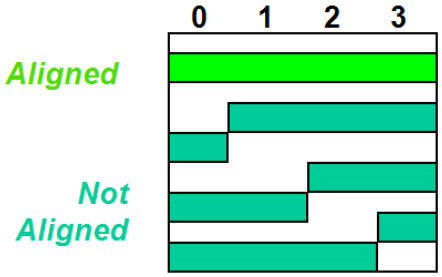
主存储器的连接与控制

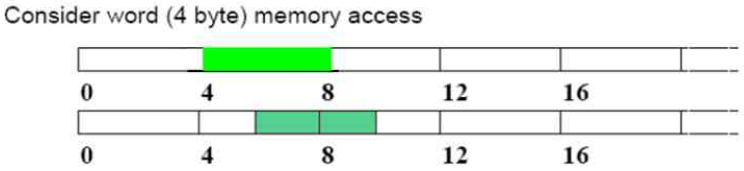
(3)边界对齐的数据存放方法

双字地址的最末三个二进制位必须为000，单字地址的最末两位必须为00，半字地址的最末一位必须为0。它能够保证无论访问双字、单字、半字或字节，都在一个存取周期内完成，尽管存储器资源仍然有浪费，但是浪费比第(2)种存放方法要少得多。



e.g for a word length of 32-bit(4 bytes), aligned words begin at byte address 0, 4, 8… (地址的最后两位为“00”)





6.3.3 指令流水线的实现

令MIPS指令系统的执行微结构采用如下的流水线数据通路，5个流水功能段：

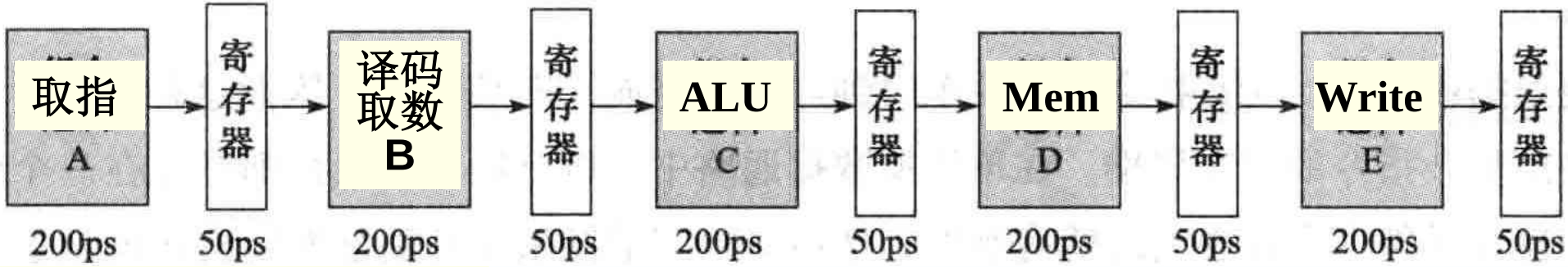
①取指: 组合逻辑A中有指令存储器，该阶段实现取指，PC+4。

②译码/取数(Reg/Dec): B中有寄存器堆，该阶段读取寄存器(rs, rt)内容并对指令操作码译码，产生指令执行所需的控制信号。

③执行(ALU): 组合逻辑C中有ALU，该阶段实现算术逻辑运算。

④访存(Mem): D中有数据存储器, 该阶段实现存储器读写操作。

⑤写回(Write):组合逻辑E中有多路选择器, 该阶段选择存入寄存器的结果，并将其保持一段建立时间。



典型MIPS指令在5段流水线数据通路中的执行过程

R型指令：如, sub rd, rs, rt

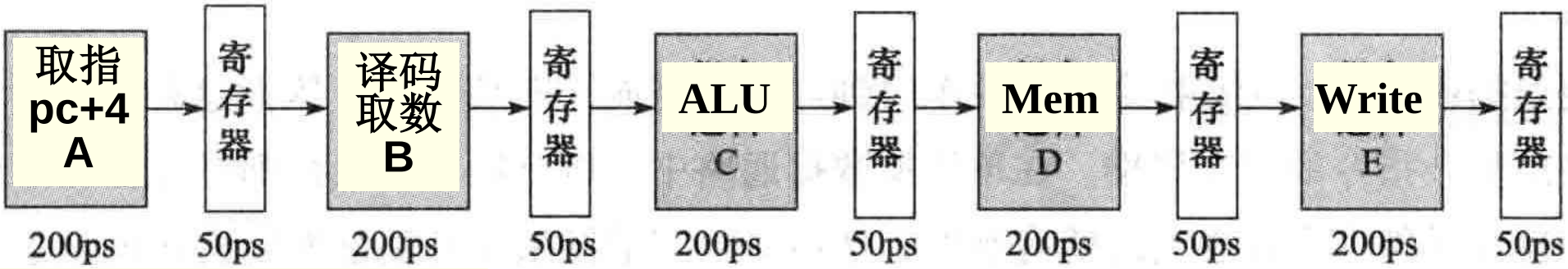
A、B两个公共功能段后,

C：ALU段在ALU中计算;

D：Mem段为空操作;

E：Write段将ALU运算结果写rd。

I型带立即数的运算指令(如：addi rt, rs, imm16)：功能段划分与R型指令相同。只是操作码译码产生的控制信号不同，如Extop, ALUSrc, ALUop等 。



典型MIPS指令在5段流水线数据通路中的执行过程

lw指令： 如，lw rt, imm16(rs)

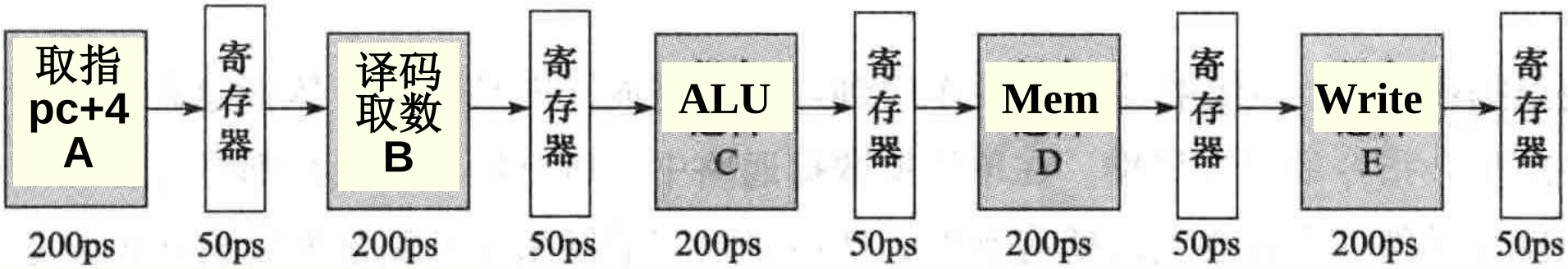
R[rt] ← M[R[rs] + SignExt(imm16)]

A、B两个公共功能段后

C：ALU段在ALU中计算存储单元地址；

D：Mem段从存储单元读数据；

E：Write段将数据写rt。



典型MIPS指令在5段流水线数据通路中的执行过程

beq指令：如，beq rs, rt, imm16

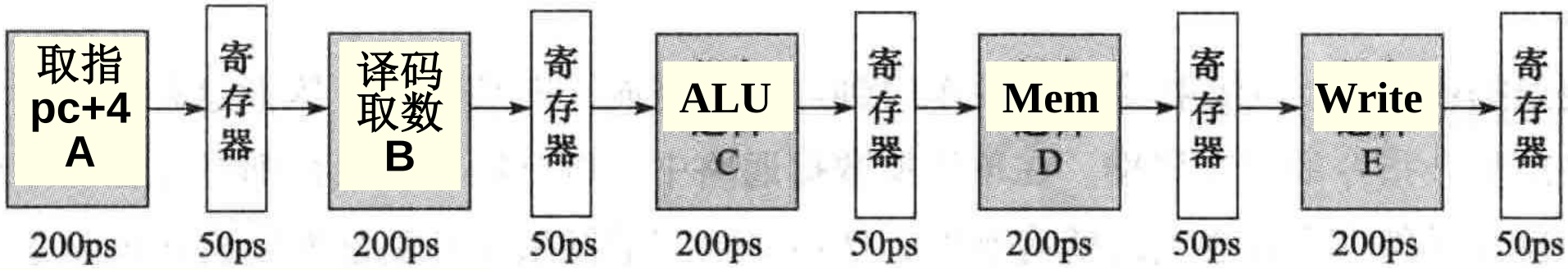
A、B两个公共功能段后，

C： ALU段在ALU中做R[rs]-R[rt], 同时计算转移目标地址

PC+4+4×imm16；

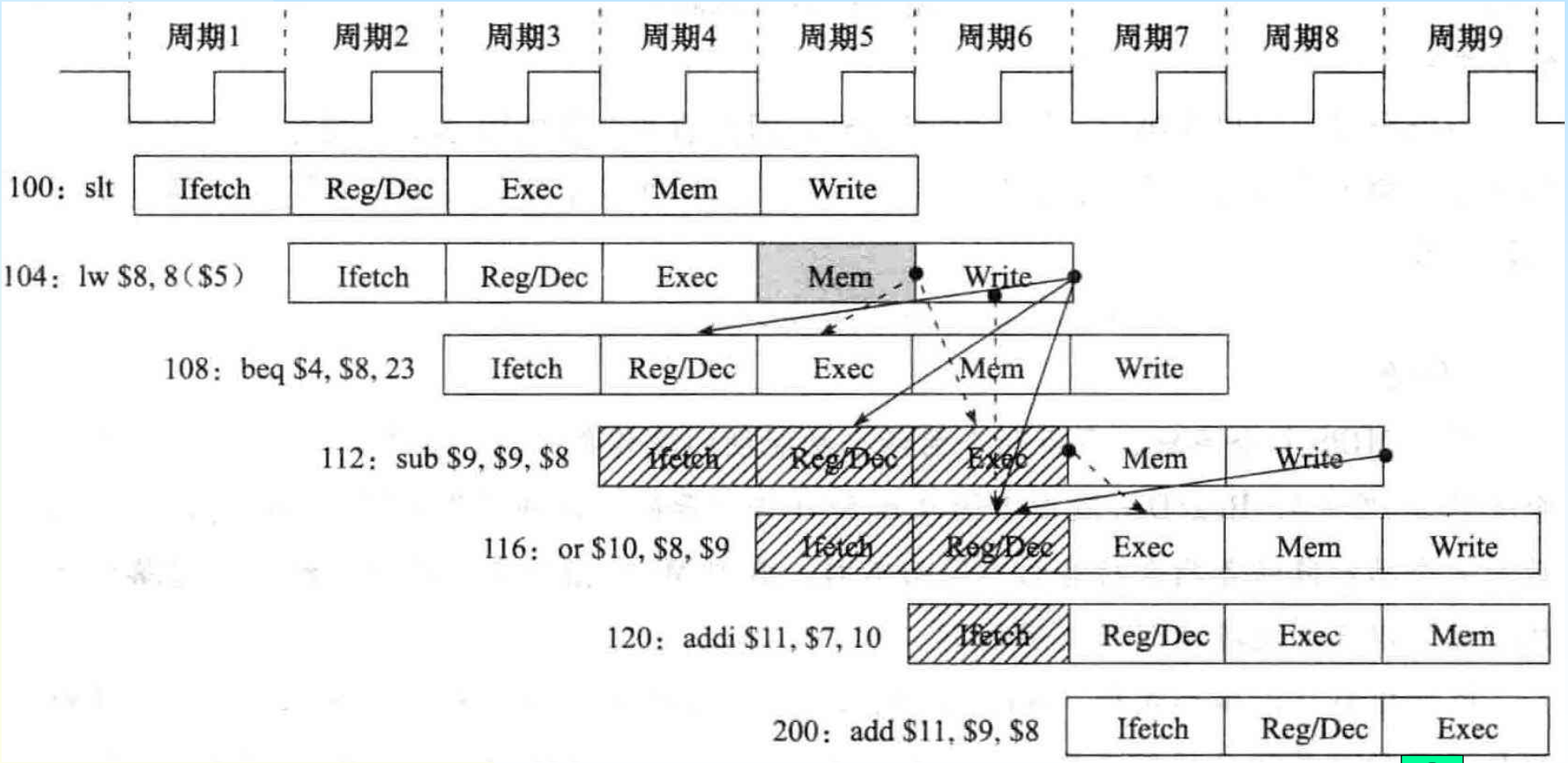
D：Mem段根据减的结果来确定将哪个地址送PC;

E：Write段为空操作。



流水线在理想情况下的正常执行过程

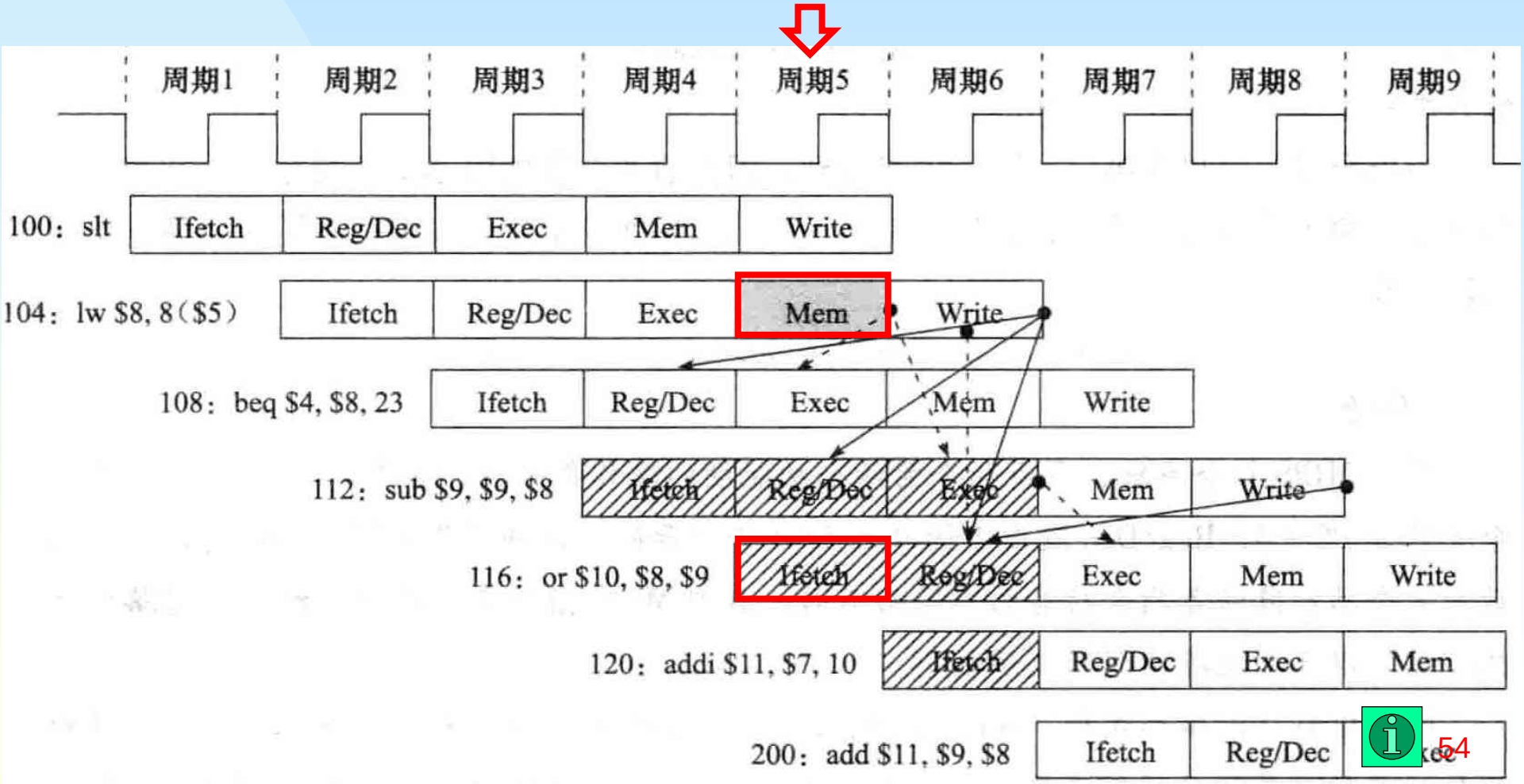
设某段时间执行的指令序列为slt, lw, beq, sub, or, …



流水线冒险:可能会遇到一些情况,破话流水线的执行。

1、结构冒险(硬件资源冲突)：同一个部件同时被不同指令使用。

若指令存储器与数据存储器是同一个存储器，则可能访存冲突。

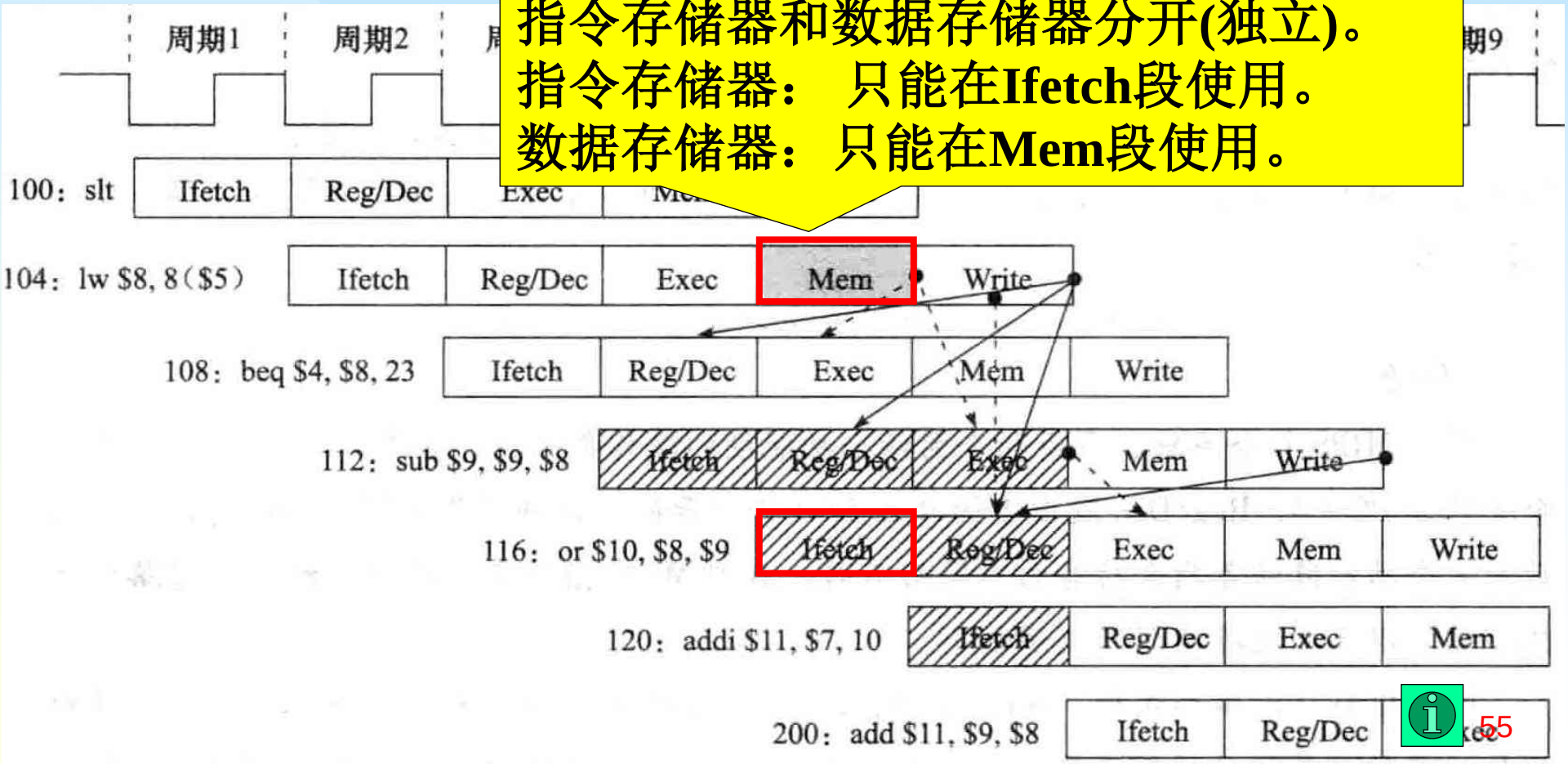


解决结构冒险的策略：

① 规定一个部件每条指令只能使用一次，且只能在特定阶段使用。

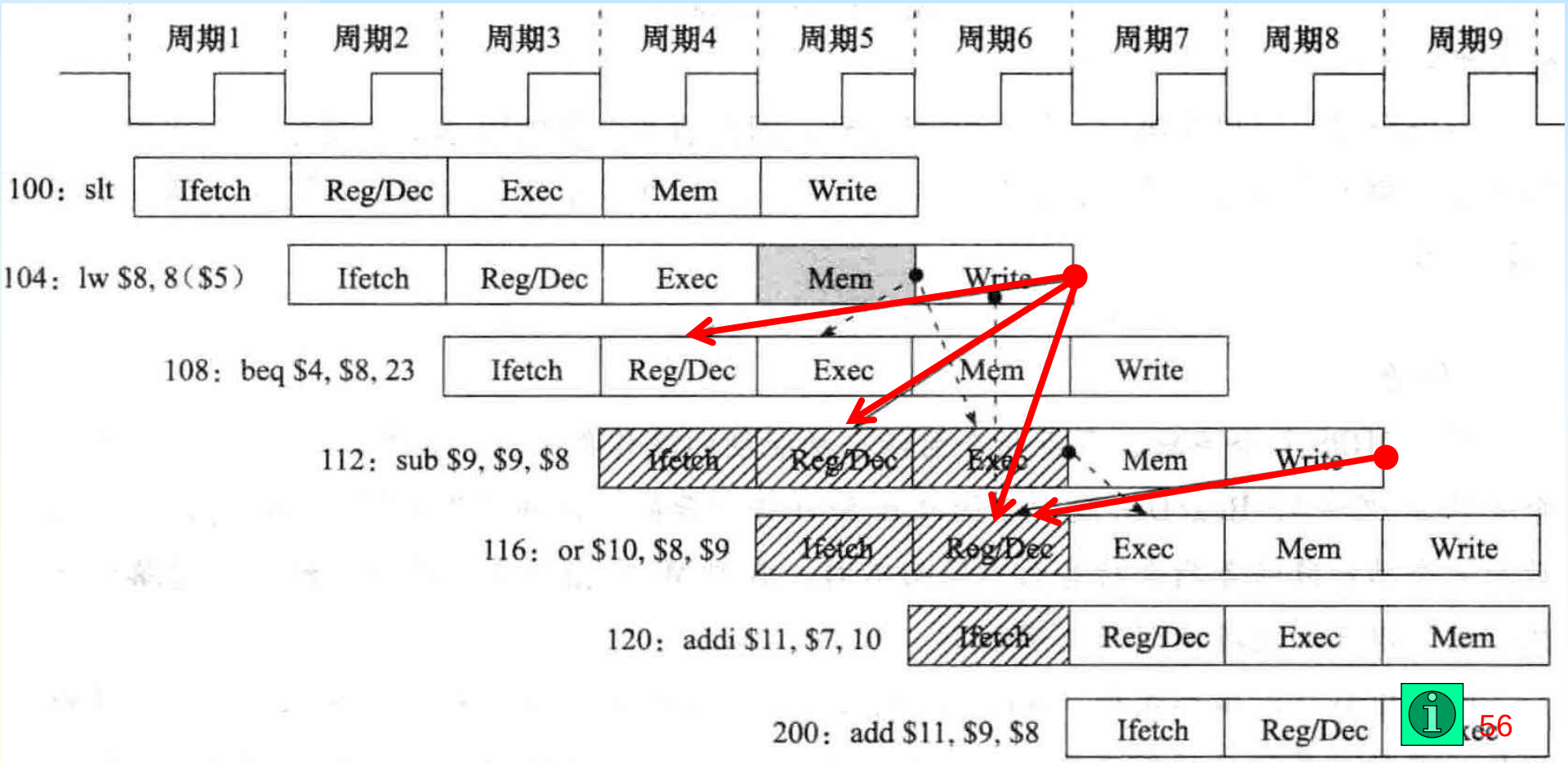
② 设置多个独立的部件避免资源冲突。

访存冲突：
指令存储器和数据存储器分开(独立)。
指令存储器： 只能在Ifetch段使用。
数据存储器：只能在Mem段使用。



流水线冒险:可能会遇到一些情况,破话流水线的执行。

2、数据冒险(数据相关)：后面指令用到前面指令的运算结果时，前面指令的结果还没产生。如，

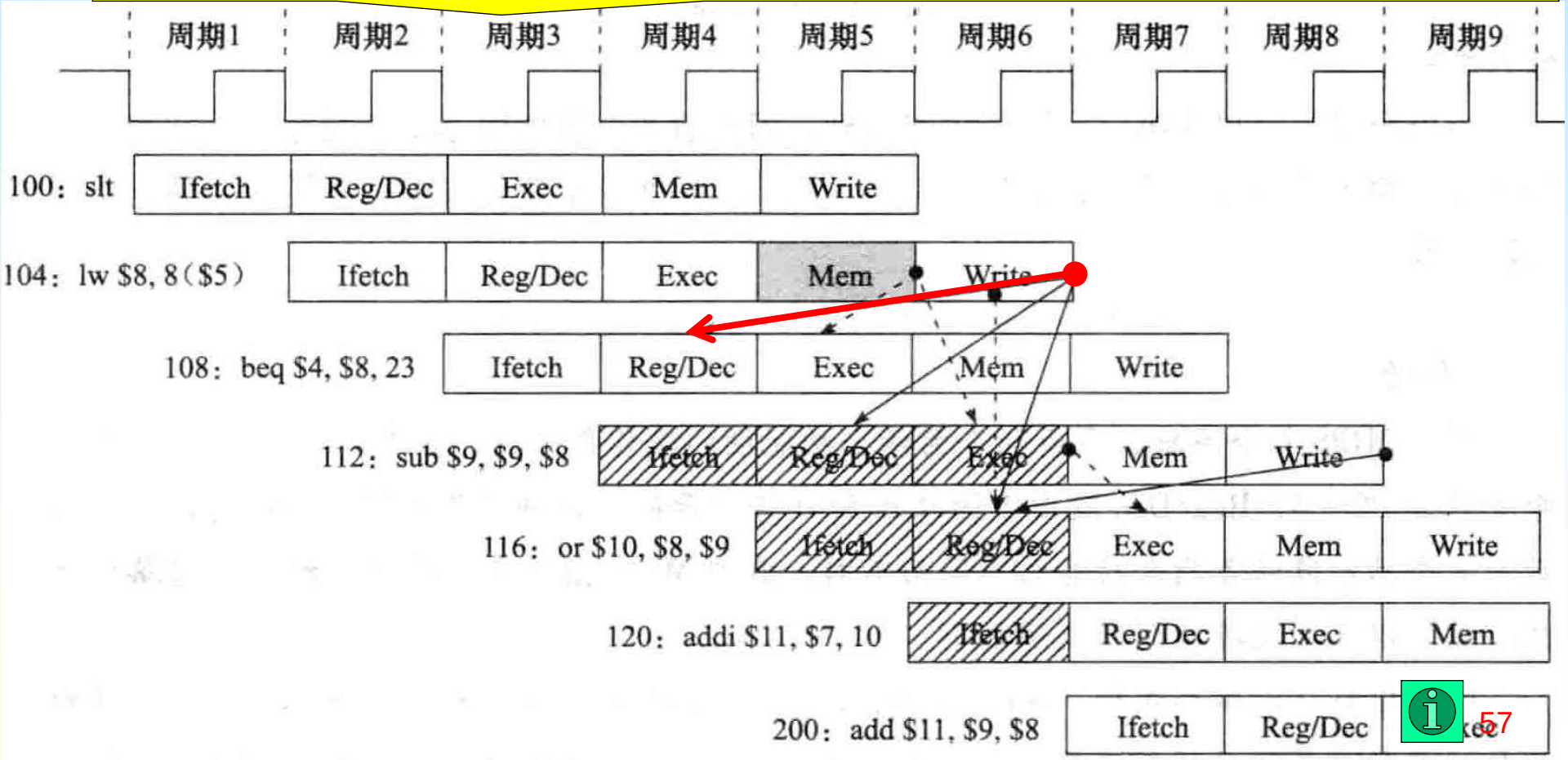


解决数据冒险的策略：

① 由编译器在数据相关的指令间加若干nop指令。

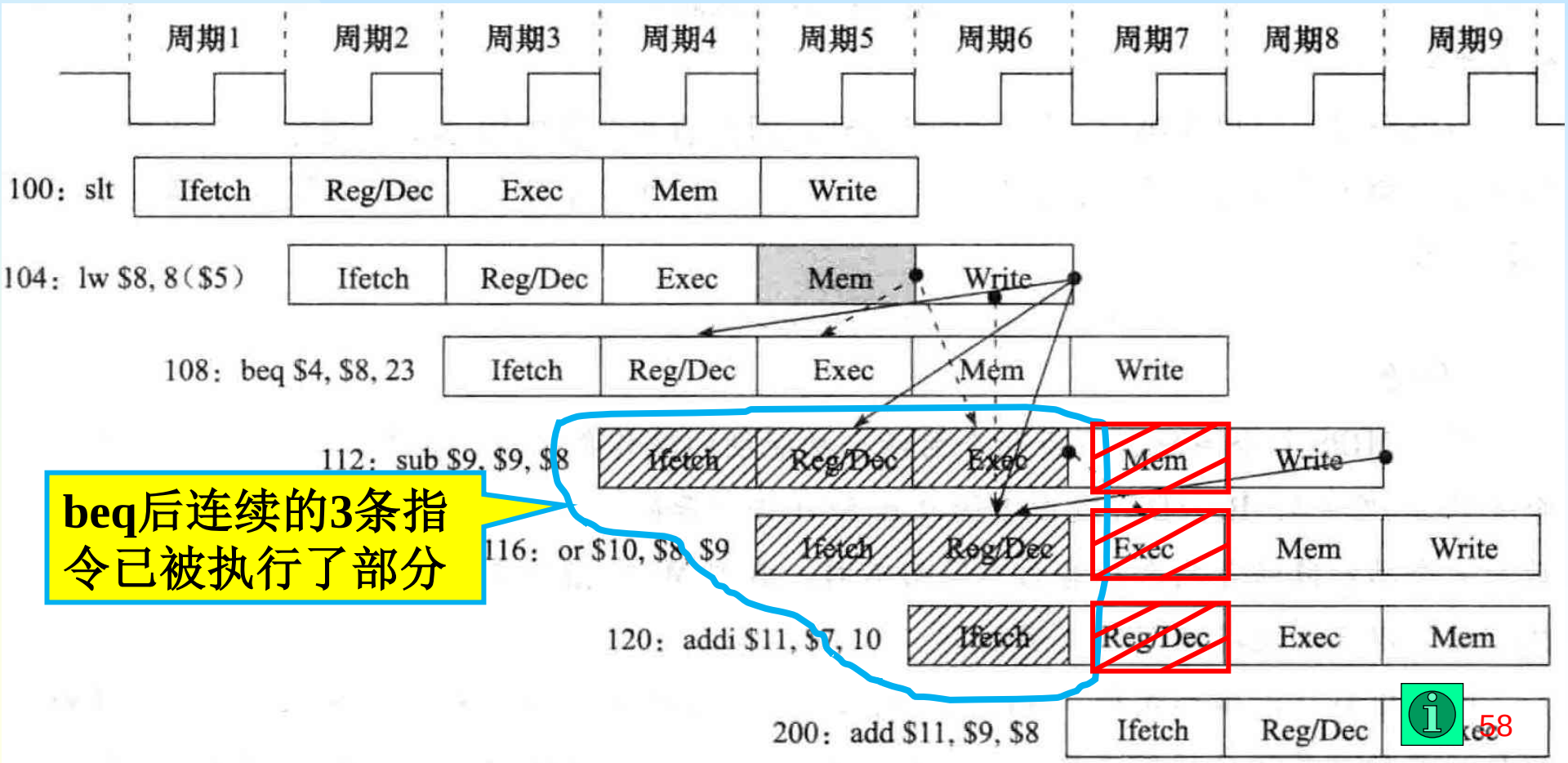
② 数据转发、通用寄存器读写操作特殊处理等

如在beq指令前加3条nop指令。



流水线冒险:可能会遇到一些情况,破话流水线的执行。

3、控制冒险：各种转移类指令，异常、中断等事件，会改变指令执行顺序，引起流水线阻塞。如，



解决控制冒险的策略：

①编译时beq指令后填入C(3)条nop指令。

② 分支预测等。