第五章　中央处理器

- 5.1 CPU功能和组成
- 5.2 指令周期
- 5.3 时序产生器和控制方式
- 5.4 微程序控制器
- 5.5 硬连线控制器
- 5.6 流水CPU
- 5.7 RISC CPU

5.1 CPU的功能和组成

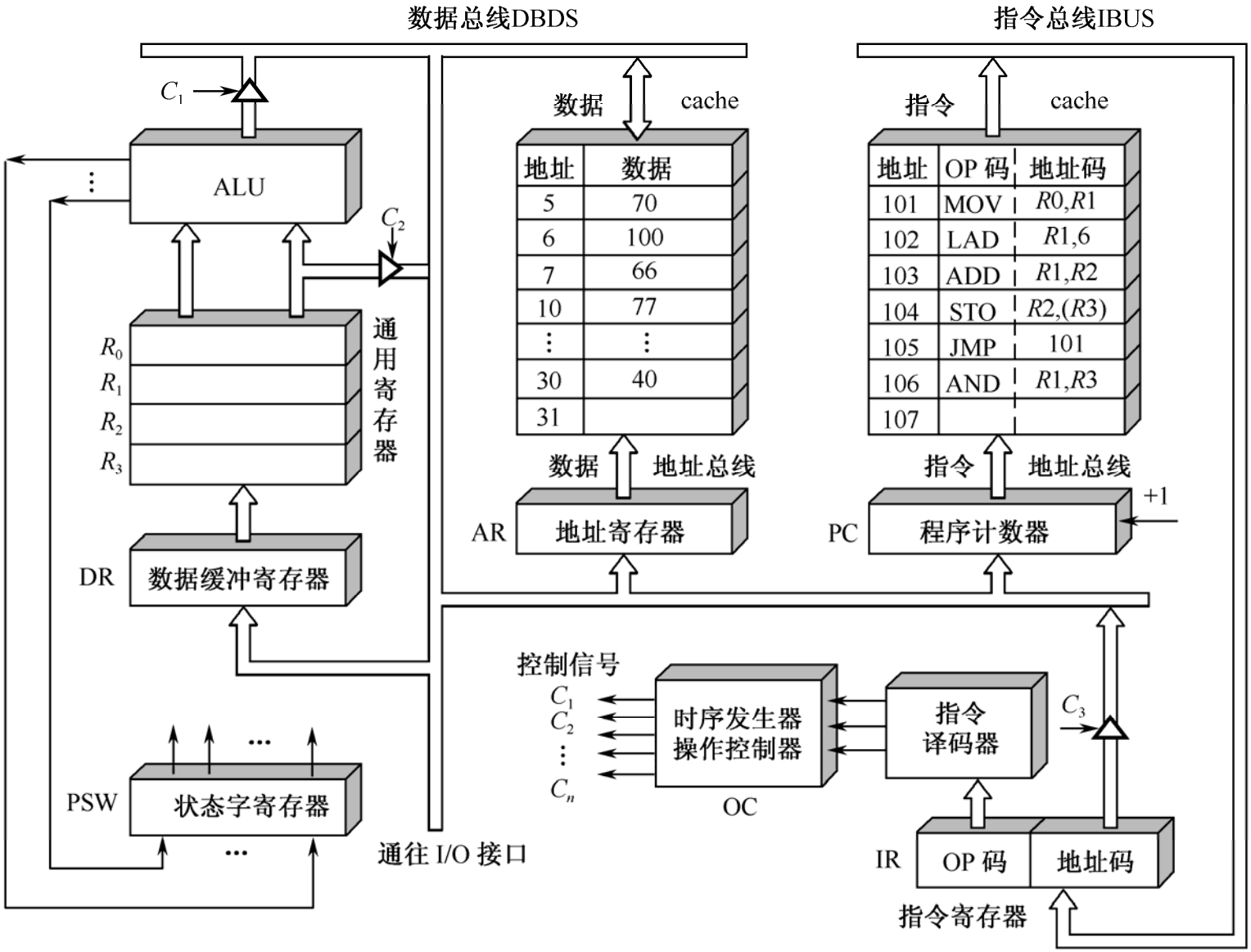
- 当代主流计算机所遵循的仍然是冯.诺依曼的“存储程序、程序控制”思想。
  - 程序告诉计算机：应该逐步执行什么操作；在什么地方找到用来操作的数据，结果存到何处等。
  - 中央处理器是控制计算机自动完成取出指令和执行指令任务的部件。它是计算机的核心部件，通常简称为CPU（Central Processing Unit）
  - CPU的工作过程：取指令→执行指令→取指令→.....

5.1.1 CPU的功能

- 4个基本功能：
  - 指令控制：程序的顺序控制
    - 程序是一个指令序列，这些指令的相互顺序不能任意颠倒，保证机器按规定的顺序执行程序是CPU的首要任务。
  - 操作控制：一条指令的功能由若干操作信号实现
    - 根据指令的功能发出若干个操作信号，把各种操作信号送往相应的部件，从而控制这些部件按指令的要求进行动作。
  - 时间控制：指令各个操作实施时间的定时
    - 每条指令的定时
    - 每个微操作的定时
  - 数据加工：算术运算和逻辑运算
    - 完成数据的加工处理，是CPU的根本任务。

5.1.2 CPU的基本组成

- 中央处理器CPU = 运算器 + cache + 控制器



5.1.2 CPU的基本组成

- 中央处理器CPU = 运算器 + cache + 控制器
  - **Cache**：
    - 指令cache：PC，**IBUS**
    - 数据cache：AR，**DBUS**
  - 运算器功能：
    - 执行所有的算数运算。
    - 执行所有的逻辑运算，并进行逻辑测试，如零值测试或两个值的比较。
  - 运算器组成：
    - ALU，通用寄存器：R0～R3，数据缓冲暂存器：DR
    - 状态字寄存器：PSW，保存运算过程和结果的特征：进位，溢出，符号，奇偶.....

5.1.2 CPU的基本组成

- 控制器由程序计数器(PC)、指令寄存器(IR)、指令译码器、时序产生器和操作控制器组成，它是发布命令的“决策机构”，即完成协调和指挥整个计算机系统的操作。
  - 控制器的主要功能有：
    - 从内存中取出一条指令，并指出下一条指令在内存中的位置。
    - 对指令进行译码或测试，并产生相应的控制信号。
    - 输出相应的控制信号，指挥并控制CPU，内存和I/O之间的数据流动的方向。
  - 程序计数器**PC**(Programming Counter)
    - 用来存放正在执行的指令的地址或接着将要执行的下一条指令的地址。
    - 顺序执行时，每执行一条指令，PC的值应加1
    - 要改变程序执行顺序的情况时，一般由转移类指令将转移目标地址送往PC ，可实现程序的转移。
  - 指令寄存器**IR**(Instruction Register)
    - 指令寄存器用来存放从存储器中取出的待执行的指令。
    - 在执行该指令的过程中，指令寄存器的内容不允许发生变化，以保证实现指令的全部功能。

5.1.3 CPU中的主要寄存器

- 寄存器：CPU保存数据的设备，每个寄存器可以保存一个字长类型的数据。
  - 数据缓冲寄存器(DR)：缓冲寄存器用来暂时存放由内存储器读出的一条指令或一个数据字；反之，当向内存存入一条指令或一个数据字时，也暂时将它们存放在缓冲寄存器中。
  - 指令寄存器(IR)：指令寄存器用来保存当前正在执行的一条指令。存放从内存中取出的指令；其中指令的操作码送到指令译码器，译码后输出控制信号。
  - 程序计数器(PC)：程序计数器中存放的是下一条指令在内存中的地址。 若程序顺序执行：PC ← PC+1；若程序有跳转：PC←PC+偏移地址。
  - 地址寄存器(AR)：地址寄存器用来保存当前CPU所访问的内存单元的地址。由于在内存和CPU之间存在着操作速度上的差别，所以必须使用地址寄存器来保持地址信息，直到内存的读/写操作完成为止。

5.1.3 CPU中的主要寄存器

- 寄存器：CPU保存数据的设备，每个寄存器可以保存一个字长类型的数据。
  - **累加寄存器(AC)**：累加寄存器AC通常简称为累加器，它的功能是：当运算器的算术逻辑单元（ALU）执行全部算术和逻辑运算时，为ALU提供一个工作区。累加寄存器是暂时存放ALU运算的结果信息。显然，运算器中至少要有一个累加寄存器。
  - **状态条件寄存器(PSW)**：状态条件寄存器保存由算术指令和逻辑指令运行或测试的结果建立的各种条件码内容，同时状态条件寄存器还保存中断和系统工作状态等信息，以便使CPU和系统能及时了解机器运行状态和程序运行状态。因此，状态条件寄存器是一个由各种状态条件标志拼凑而成的寄存器。
    - **OF** → 溢出　　**DF** → 方向　　**IF** → 中断允许
    - **TF** → 追踪　　**SF** → 符号　　**ZF** → 零
    - **AF** → 辅助进位　　**PF** → 奇偶　　**CF** → 进位

5.1.4 操作控制器与时序产生器

- 信息如何在各寄存器之间传送？数据的流动是由什么部件控制的？
  - **数据通道**：寄存器之间传送信息的通路。
  - **操作控制器**：根据指令操作码和时序信号，产生各种操作控制信号；正确地建立数据通路，从而完成取指令和执行指令的控制
  - **操作控制器分类**：
    - 时序逻辑型：硬布线控制器，采用时序逻辑技术来实现，由门电路组成的复杂树形网络。
    - 存储逻辑型：微程序控制器，采用存储逻辑技术来实现，把微操作信号代码化，使每条机器指令转化成为一段微程序并存入一个专门的存储器(控制存储器)中，微操作控制信号由微指令产生。
  - **时序产生器**：产生并发出计算机所需要的时序控制信号

5.1.4 操作控制器与时序产生器

- CPU中的寄存器总结：
  - 指令寄存器(IR)：用来保存当前正在执行的一条指令。
  - 程序计数器(PC)：用来确定下一条指令的地址。
  - 地址寄存器(AR)：用来保存当前CPU所访问的内存单元的地址。
  - 缓冲寄存器(DR)：

    <1>作为CPU和内存、外部设备之间信息传送的中转站。

    <2>补偿CPU和内存、外围设备之间在操作速度上的差别。

    <3>在单累加器结构的运算器中，缓冲寄存器还可兼作为操作数寄存器。
  - 通用寄存器(AC)：当运算器的算术逻辑单元（ALU）执行全部算术和逻辑运算时，为ALU提供一个工作区。
  - 状态条件寄存器(PSW)：保存由算术指令和逻辑指令运行或测试的结果建立的各种条件码内容。除此之外，还保存中断和系统工作状态等信息，以便使CPU和系统能及时了解机器运行状态和程序运行状态。

5.2 指令周期

- 程序的执行过程：

  1.从程序入口地址开始执行指令。

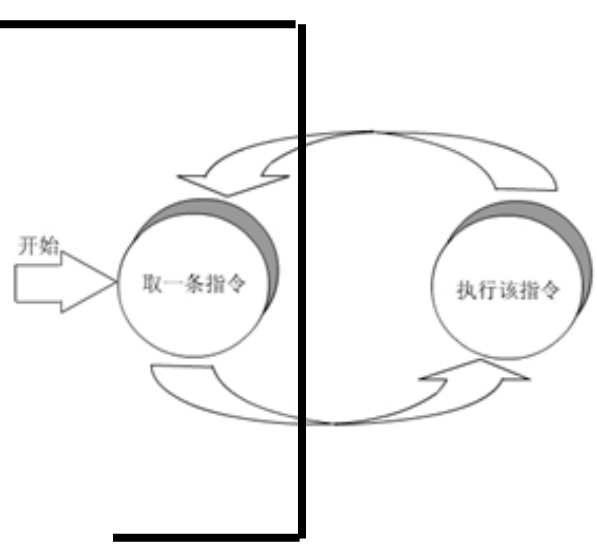
  2.取指令，分步执行取出的指令，并形成下条待执行指令的地址。

  3.自动地连续重复2，直到程序的最后一条指令。

- 指令的执行过程：

  - 读取指令
    - 指令地址送入主存地址寄存器
    - 读主存，读出内容送入指定的寄存器
  - 分析指令
    - 按指令规定内容执行指令
  - 检查有无中断请求
    - 若无，则转入下一条指令的执行过程

形成下一条指令地址



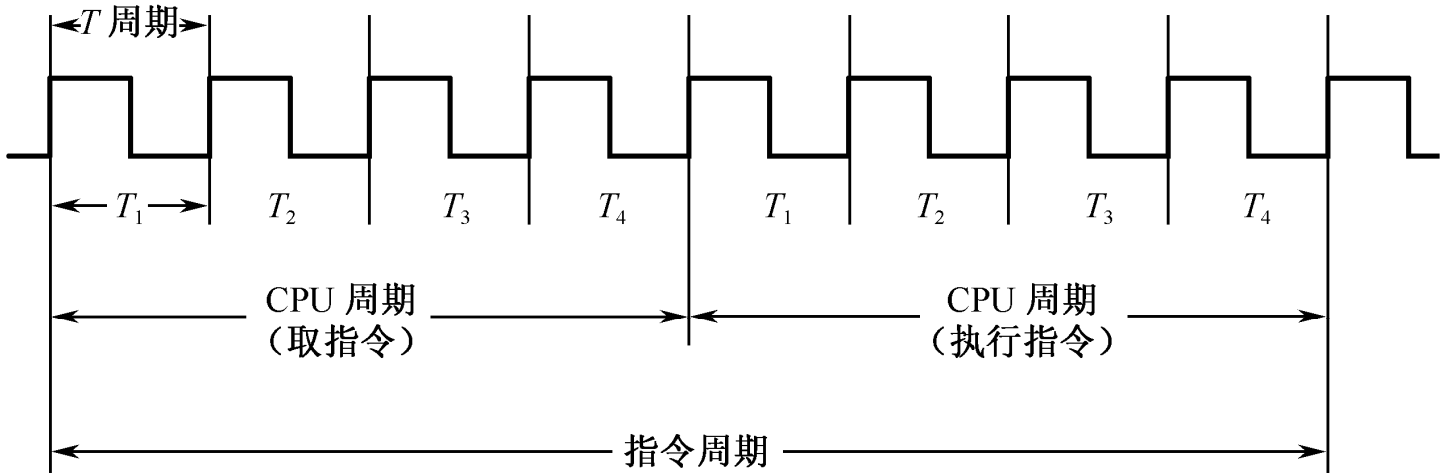
- 所有的执行过程都按严格的顺序进行，使用时间的控制实现顺序的要求。

5.2.1 指令周期的基本概念

- 指令周期：
  - CPU每取出并执行一条指令，都要完成一系列的操作，这一系列操作所需用的时间通常叫做一个指令周期。
    - 不同功能的指令指令周期不相同。
- 机器周期：
  - 把一条指令周期划分为若干个机器周期，每个机器周期完成一个基本操作。也称为CPU周期，时钟周期，总线周期。
  - 用CPU读取一个指令字的最短时间(存取周期)来规定CPU周期。
  - 不同的指令，可能包含不同数目的CPU周期。
    - 单周期：指令周期=机器周期
    - 多周期：指令周期=若干个机器周期

5.2.1 指令周期的基本概念

- T周期：
  - 在一个CPU周期内，要完成若干个微操作。微操作有的可以同时执行，有的需要按先后次序串行执行。因而需要把一个CPU周期分为若干个相等的时间段，每一个时间段称为一个节拍脉冲或T周期。时钟周期通常定义为机器主频的倒数。
  - T周期有T1，T2，T3，T4类型。
- 1个指令周期 = 若干个CPU周期，1个CPU周期 = 若干T周期



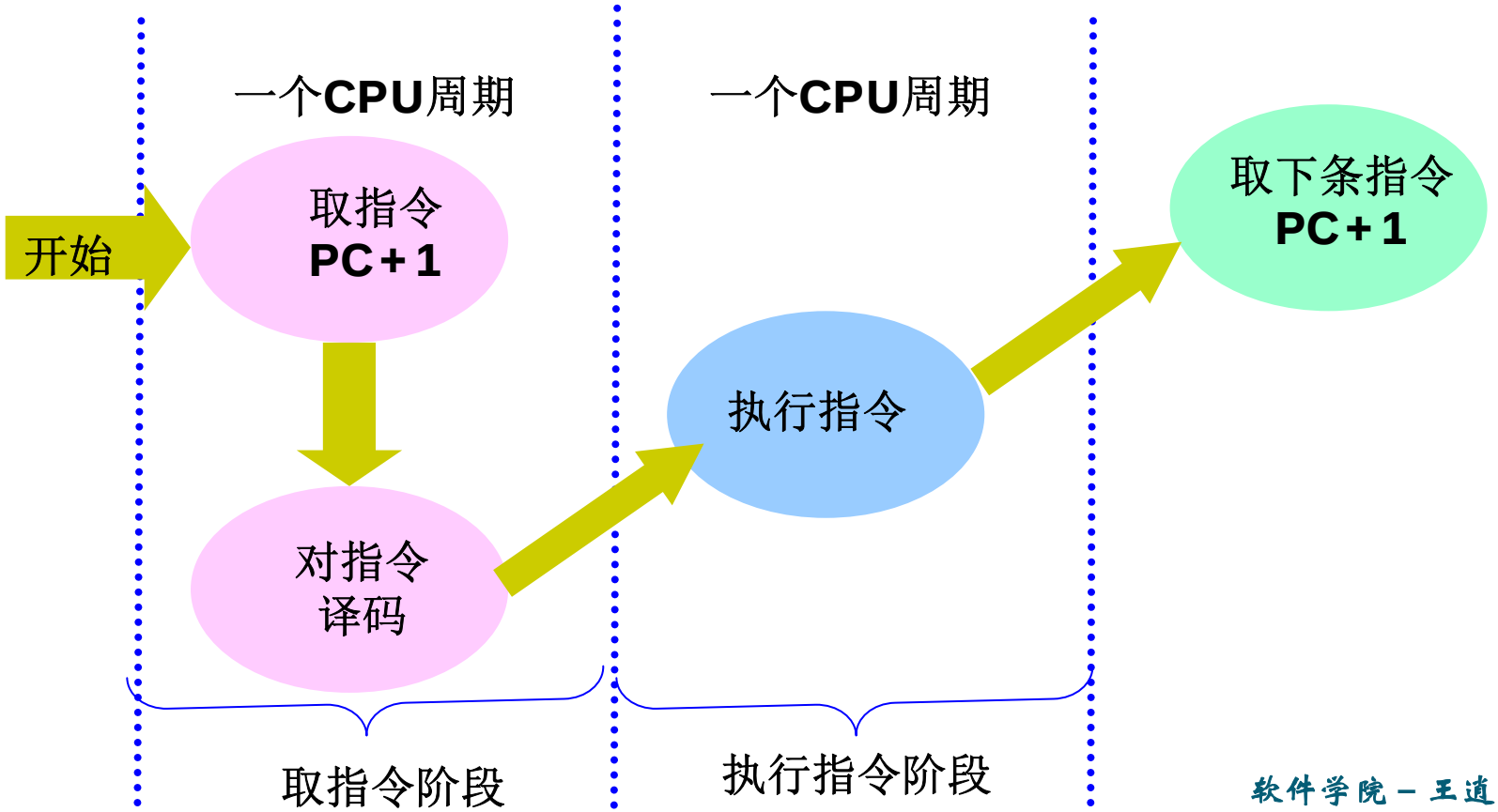
5.2.1 指令周期的基本概念

- 分析程序中6条指令的指令周期：

	内存地址	助记符指令	说明
指令存储器	100		初始值：(R0)=00,(R1)=10,(R2)=20,(R3)=30
	101	MOV R0,R1	MOV数据传送：(R1)→R0
	102	LAD R1,6	LAD取内存数：(6)→R1
	103	ADD R1,R2	ADD加法运算：(R1)+(R2)→(R2)
	104	STO R2,(R3)	STO存数：(R2)→(R3)
	105	JMP 101	JMP无条件转移：PC=101
	106	AND R3,R1	AND与运算： (R1) (R3)→R3
数据存储器	5	70	
	6	100	
	7	66	
	8	77	
	9	88	
	…	…	
	30	40/120	执行STO指令后，40→120

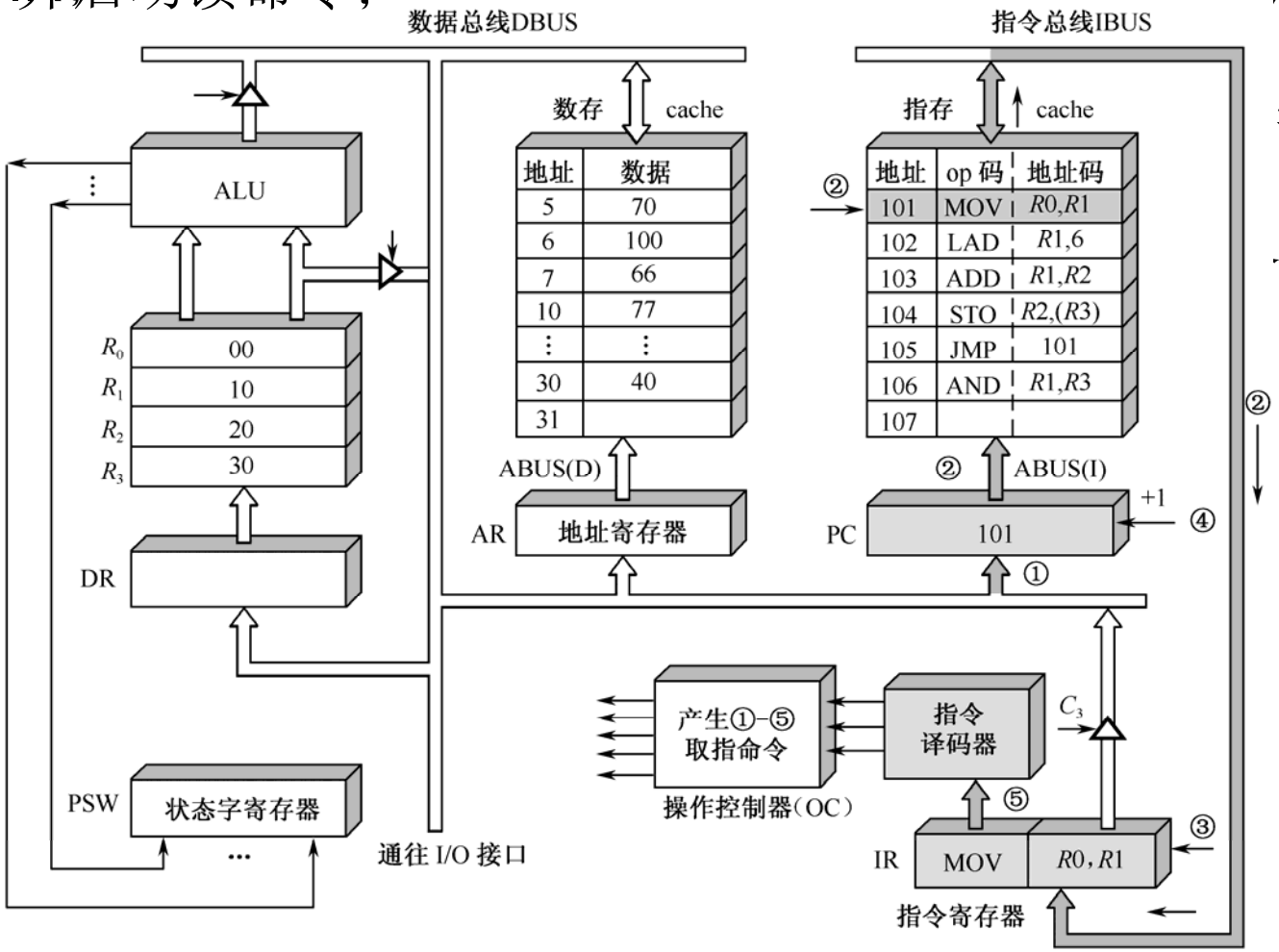
5.2.2 MOV指令的指令周期

- MOV R0,R1：RR型2地址指令。
  - MOV指令周期=2CPU周期=1取指CPU周期+1执行CPU周期
  - 取指周期：从指存中取指令字→PC++→指令字译码
  - 执行周期：实现指令的功能：(R1)→R0



MOV指令的取指周期

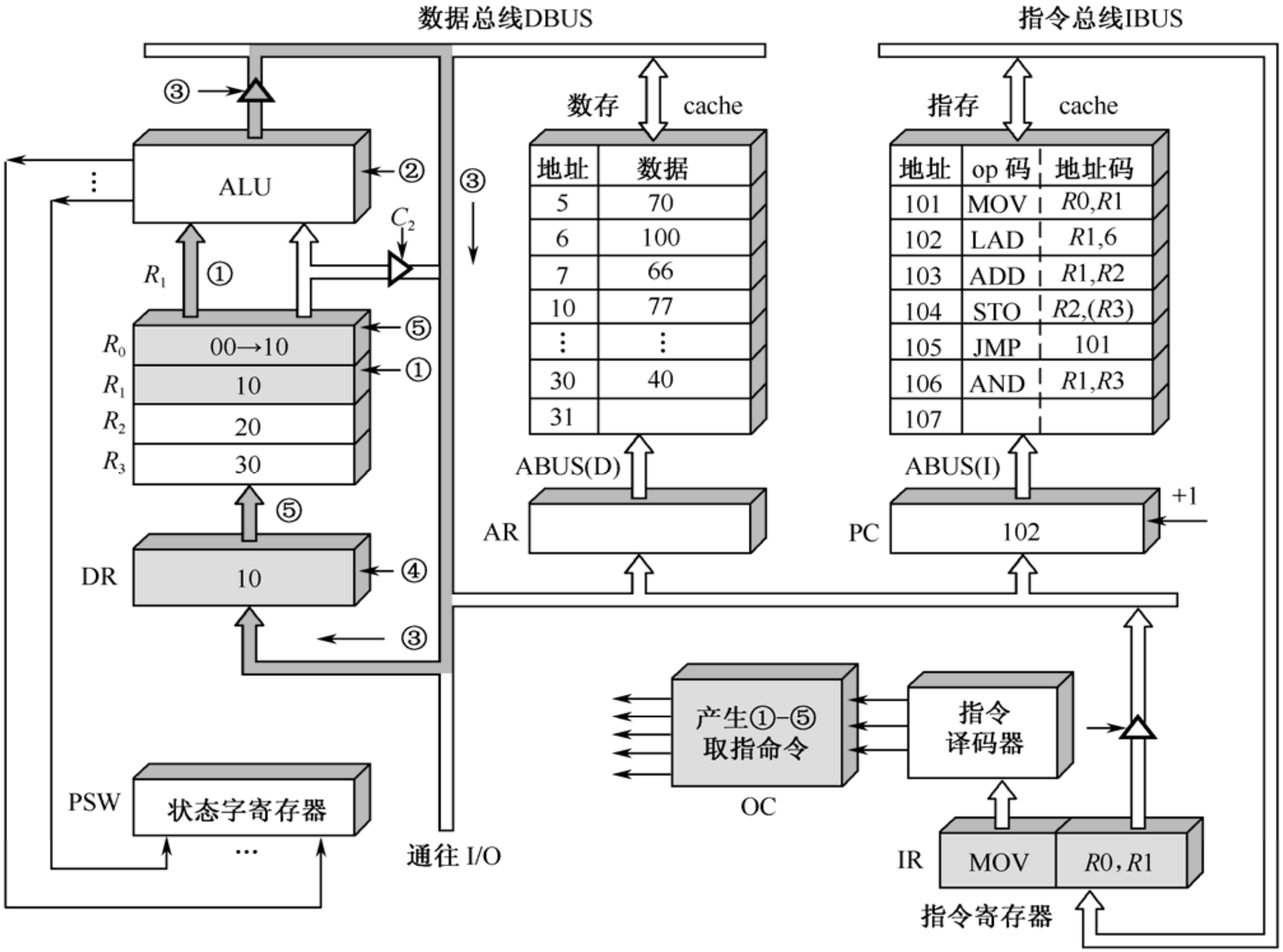
- CPU的动作：取指周期



MOV指令的执行周期

- CPU的动作：执行周期

[illegible]有一个数据。



输出送到数据总线

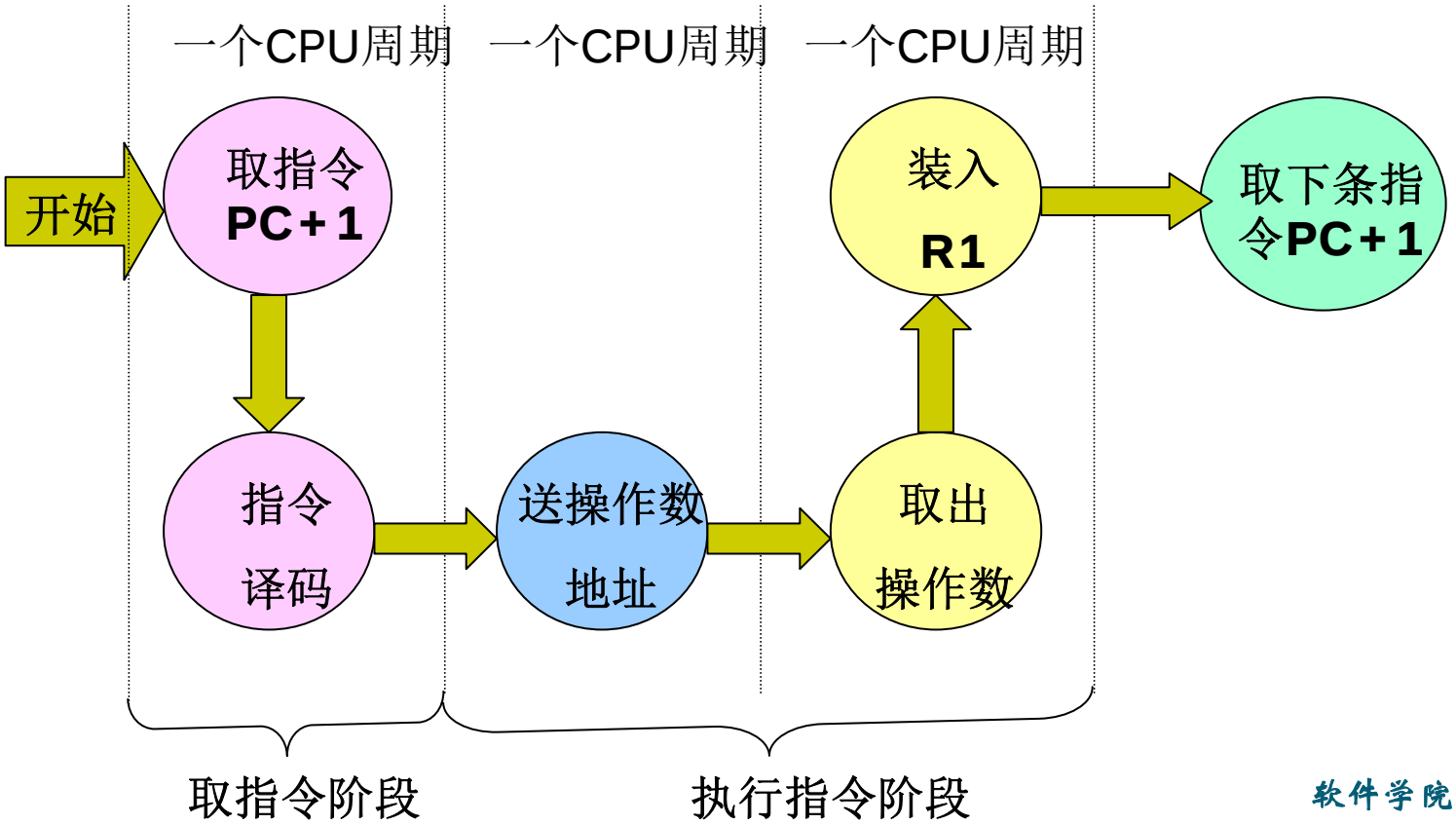
。

缓冲寄存器DR

存器R0，R0的内

5.2.3 LAD指令的指令周期

- LAD R1,6：RS型2地址指令。
  - LAD指令周期＝3CPU周期＝1取指CPU周期＋2执行CPU周期
  - 取指周期：从指存中取指令字→PC＋＋→指令字译码
  - 执行周期：实现指令的功能(6)→R1；



LAD指令的取指周期

- CPU的动作：取指周期
  - 类似于MOV指令的取指周期；

① 程序计数器PC中装入第一条指令地址**102**（八进制）；

② PC的内容被放到指令地址总线ABUS（I）上，对指存进行译码，并启动读命令；

③ 从102号地址读出的LAD指令通过指令总线IBUS装入指令寄存器IR；

④ 程序计数器内容加1，变成**103**，为取下一条指令做好准备；

⑤ 指令寄存器中的操作码（OP）被译码；

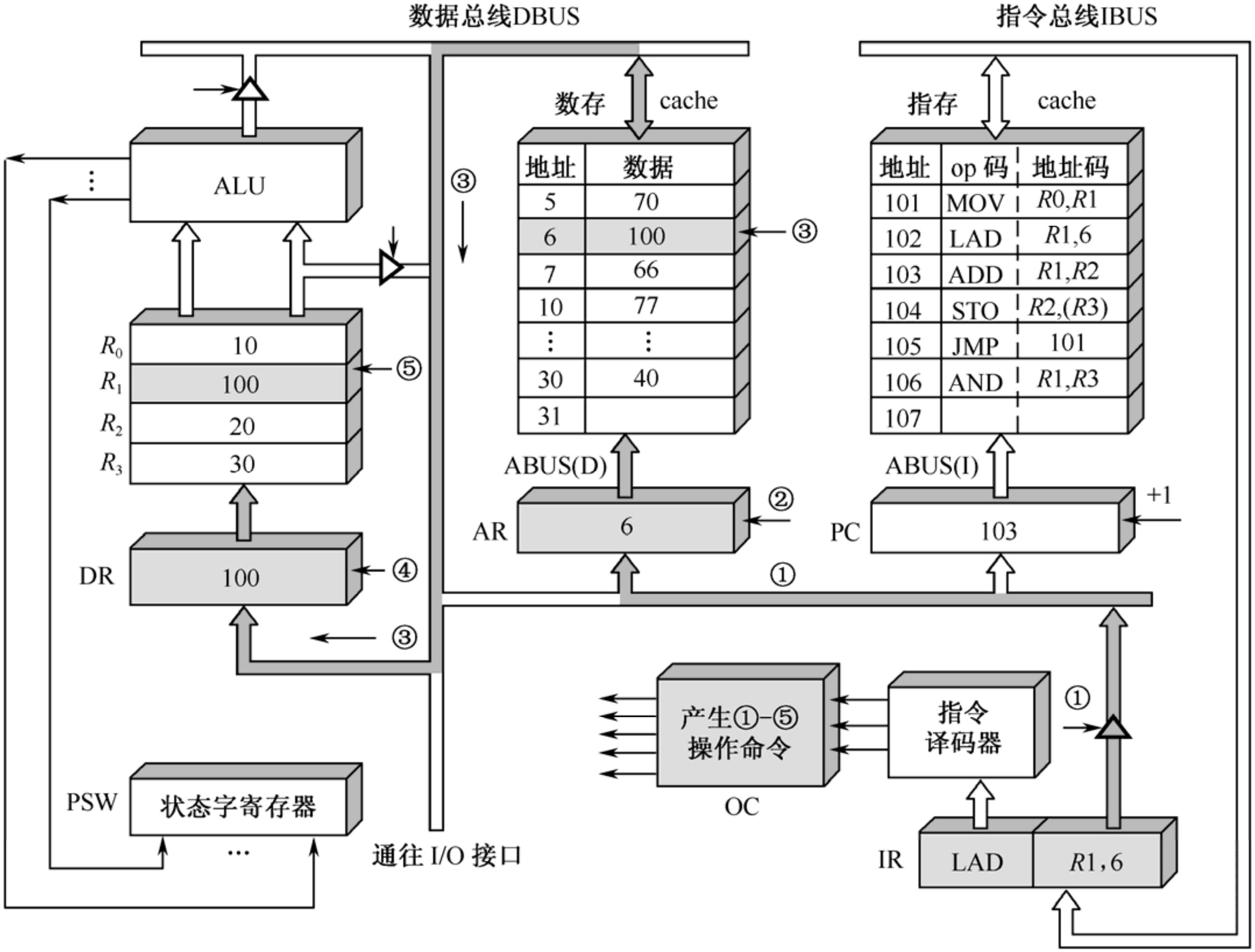
⑥ CPU识别出是LDA指令，至此，取指周期即告结束。

LAD指令的执行周期

- CPU的动作：执行周期

① 操作控制器OC发出控制命令打开IR输出三态门，将指令中的

指令执行周期结束。



存器AR；

出到DBUS上；

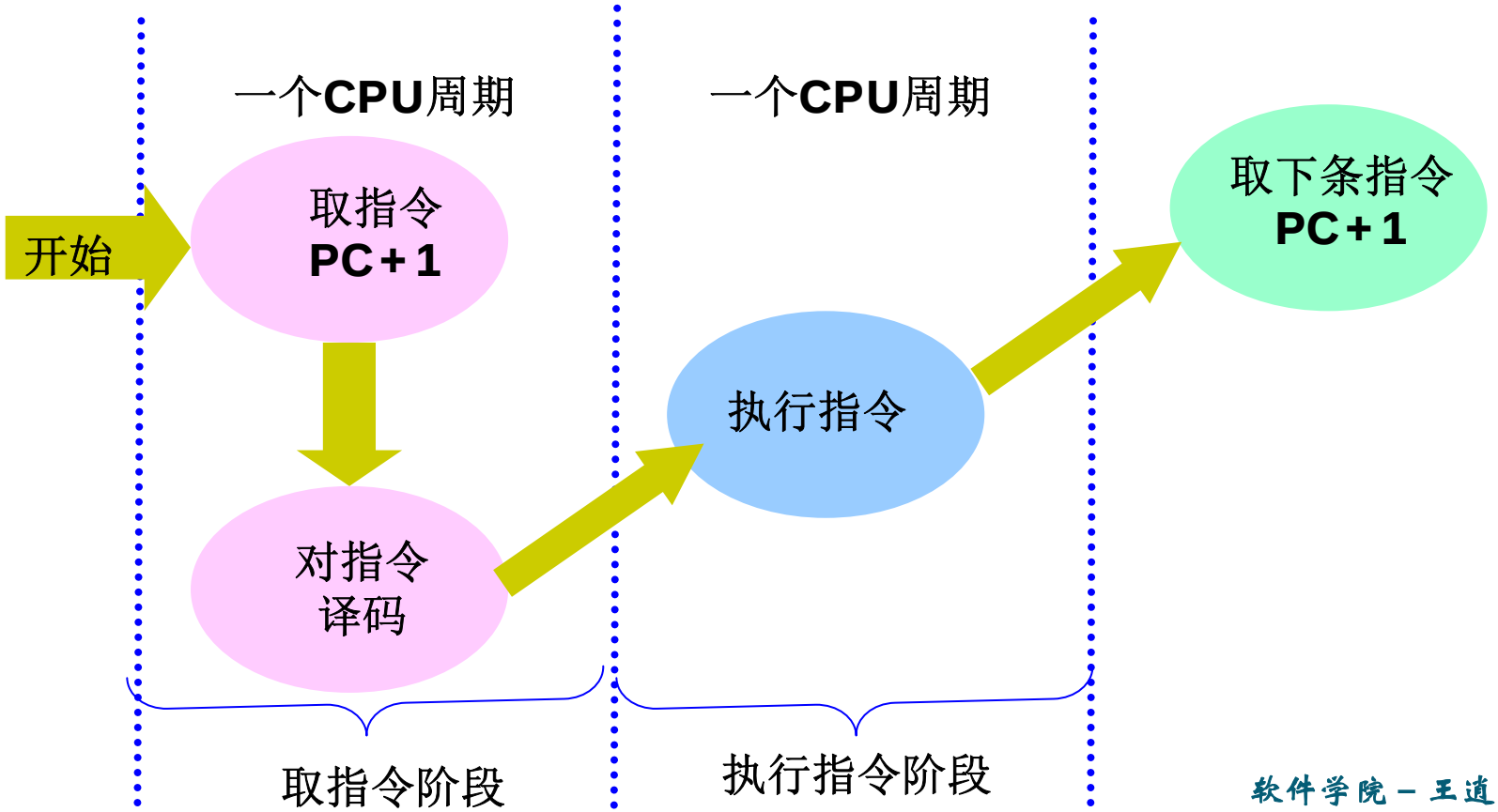
寄存器DR；

器R存1，原来R

结束。

5.2.4 ADD指令的指令周期

- ADD R1,R2：RR型2地址指令。
  - LAD指令周期＝2CPU周期＝1取指CPU周期＋1执行CPU周期
  - 取指周期：从指存中取指令字→PC＋＋→指令字译码
  - 执行周期：实现指令的功能(R1)＋(R2)→R2；



ADD指令的取指周期

- CPU的动作：取指周期
  - 类似于MOV指令的取指周期；

① 程序计数器PC中装入第一条指令地址**103**（八进制）；

② PC的内容被放到指令地址总线ABUS（I）上，对指存进行译码，并启动读命令；

③ 从103号地址读出的ADD指令通过指令总线IBUS装入指令寄存器IR；

④ 程序计数器内容加1，变成**104**，为取下一条指令做好准备；

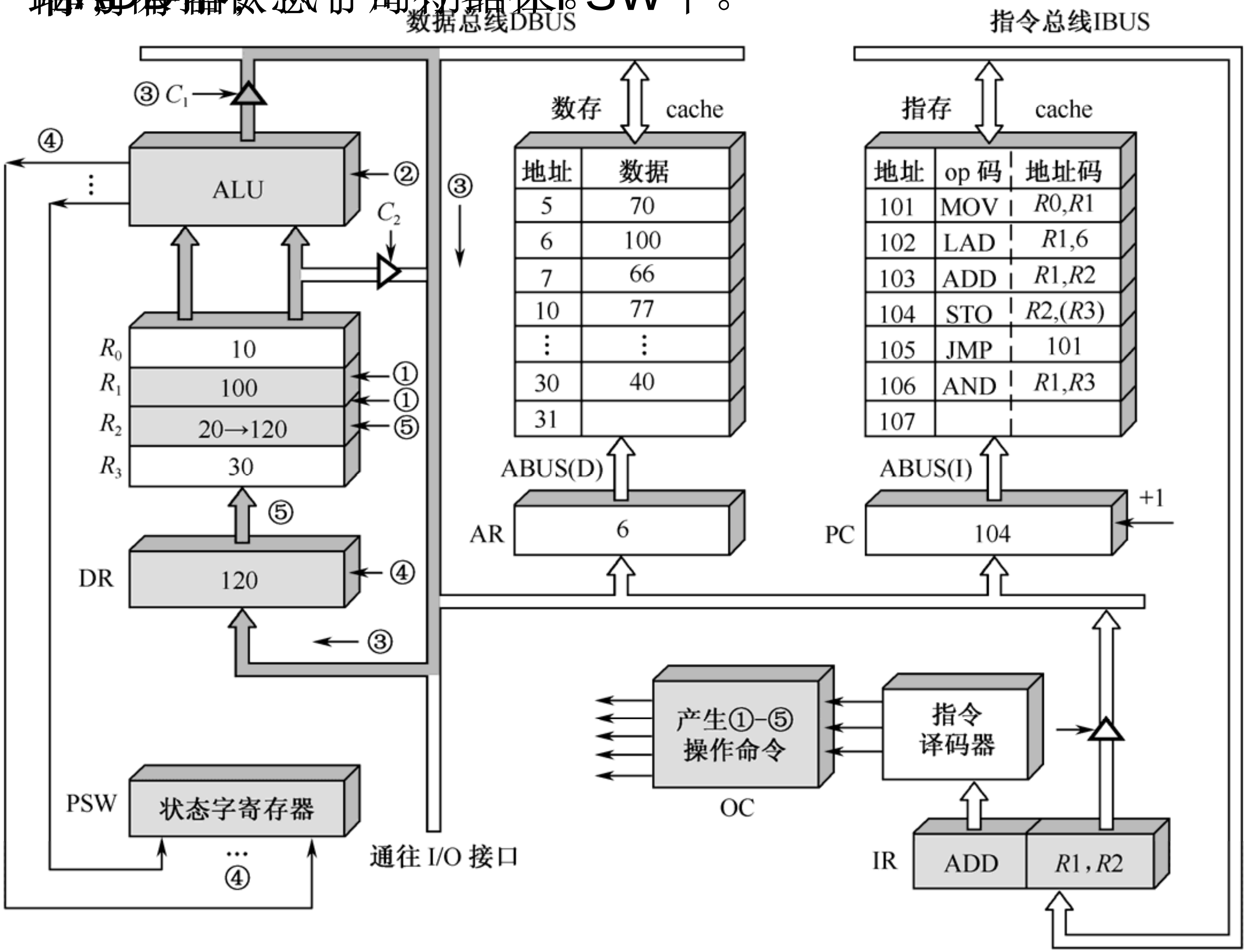
⑤ 指令寄存器中的操作码（OP）被译码；

⑥ CPU识别出是ADD指令，至此，取指周期即告结束。

ADD指令的执行周期

- CPU的动作：执行周期

PSW中。



）和R2（20）

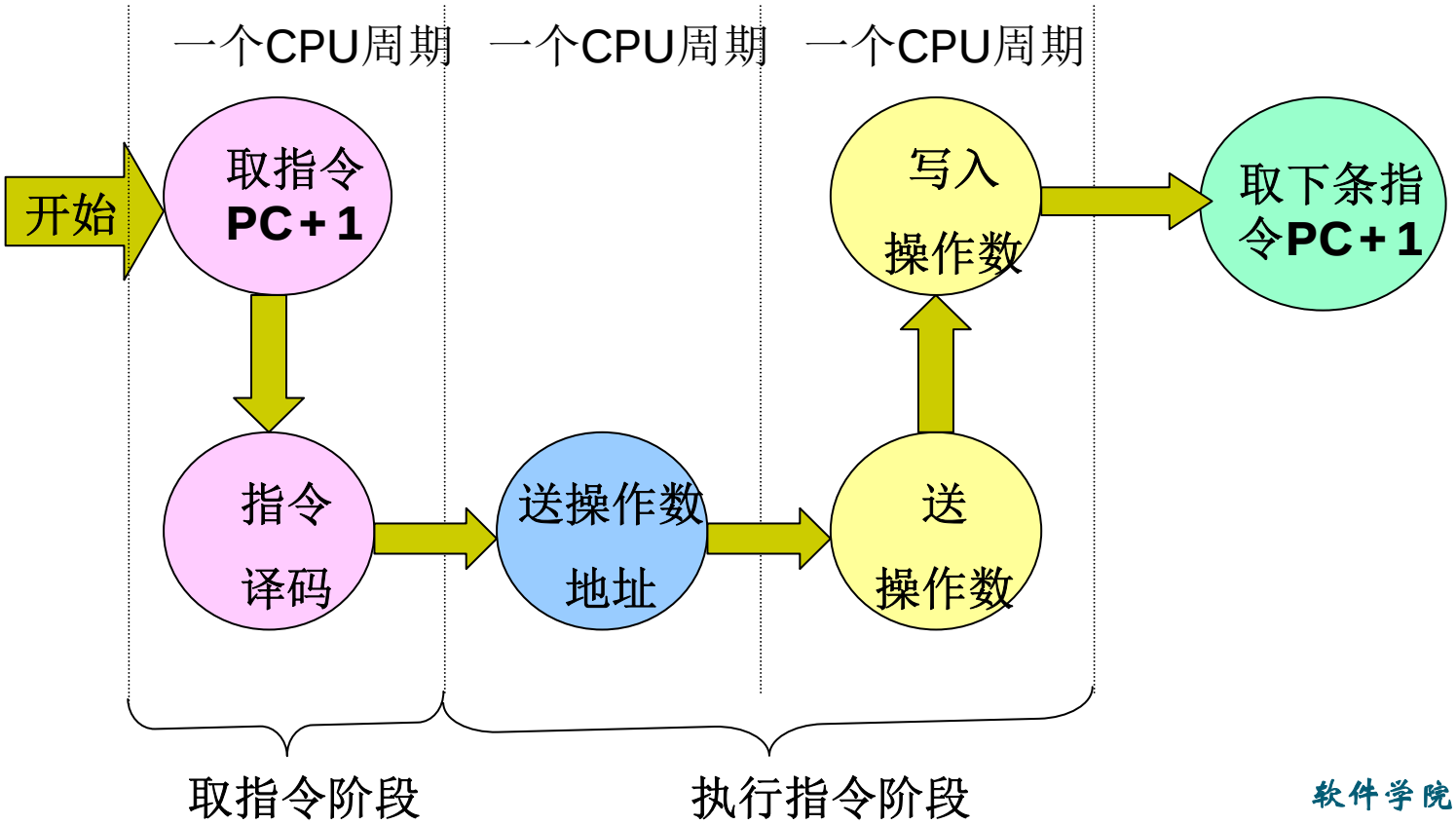
结果120放到

存器DR；ALU

中原来的内容

5.2.5 STO指令的指令周期

- STO R2,(R3)：RS型2地址指令。
  - LAD指令周期＝3CPU周期＝1取指CPU周期＋2执行CPU周期
  - 取指周期：从指存中取指令字→PC＋＋→指令字译码
  - 执行周期：实现指令的功能(R3)→(R3)；



STO指令的取指周期

- CPU的动作：取指周期
  - 类似于MOV指令的取指周期；

① 程序计数器PC中装入第一条指令地址**104**（八进制）；

② PC的内容被放到指令地址总线ABUS（I）上，对指存进行译码，并启动读命令；

③ 从104号地址读出的STO指令通过指令总线IBUS装入指令寄存器IR；

④ 程序计数器内容加1，变成**105**，为取下一条指令做好准备；

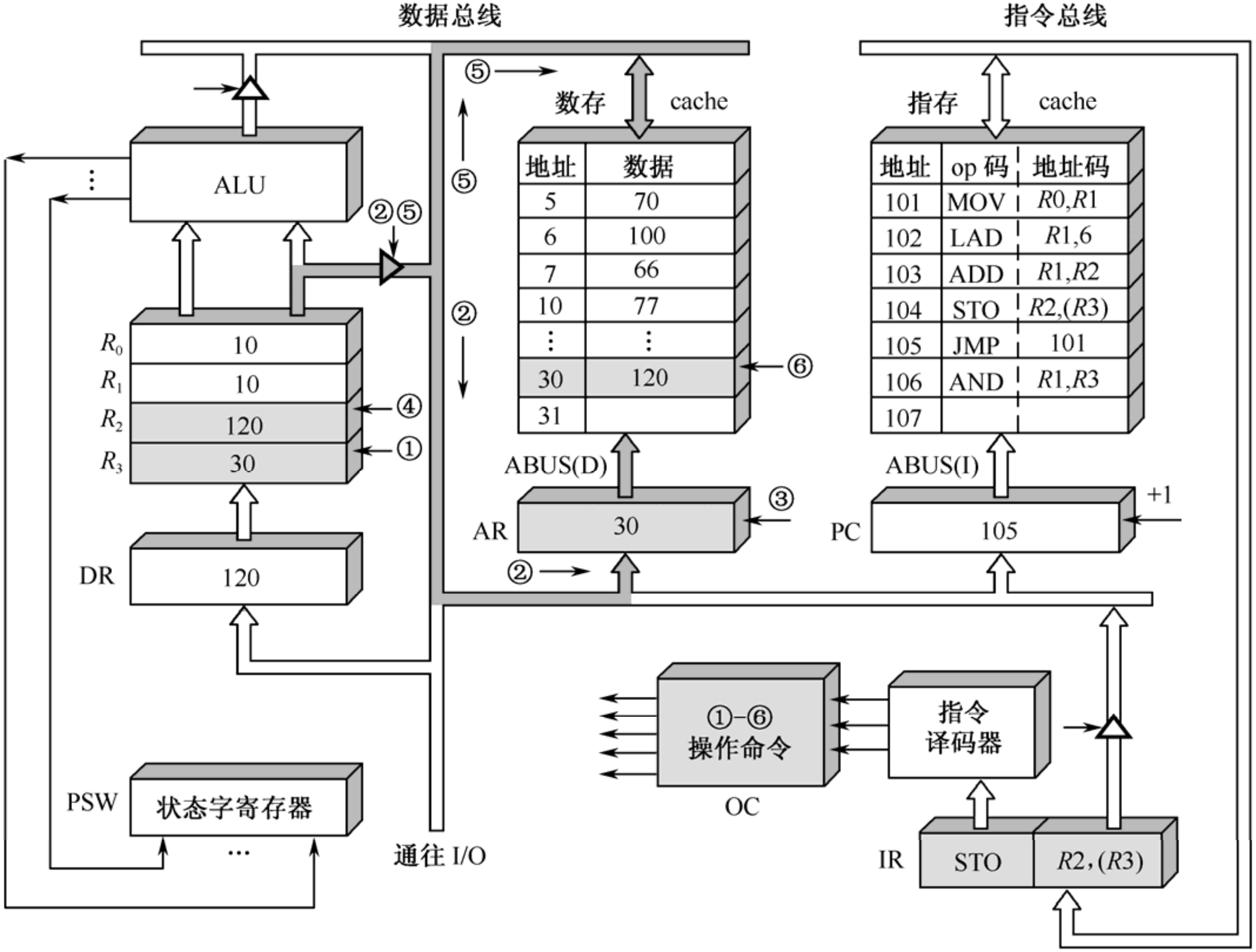
⑤ 指令寄存器中的操作码（OP）被译码；

⑥ CPU识别出是STO指令，至此，取指周期即告结束。

STO指令的执行周期

- CPU的动作：执行周期

[illegible]（将地址30放到DBUS上）…… STO指令执行周期结束。



]（不经ALU以

数存地址译

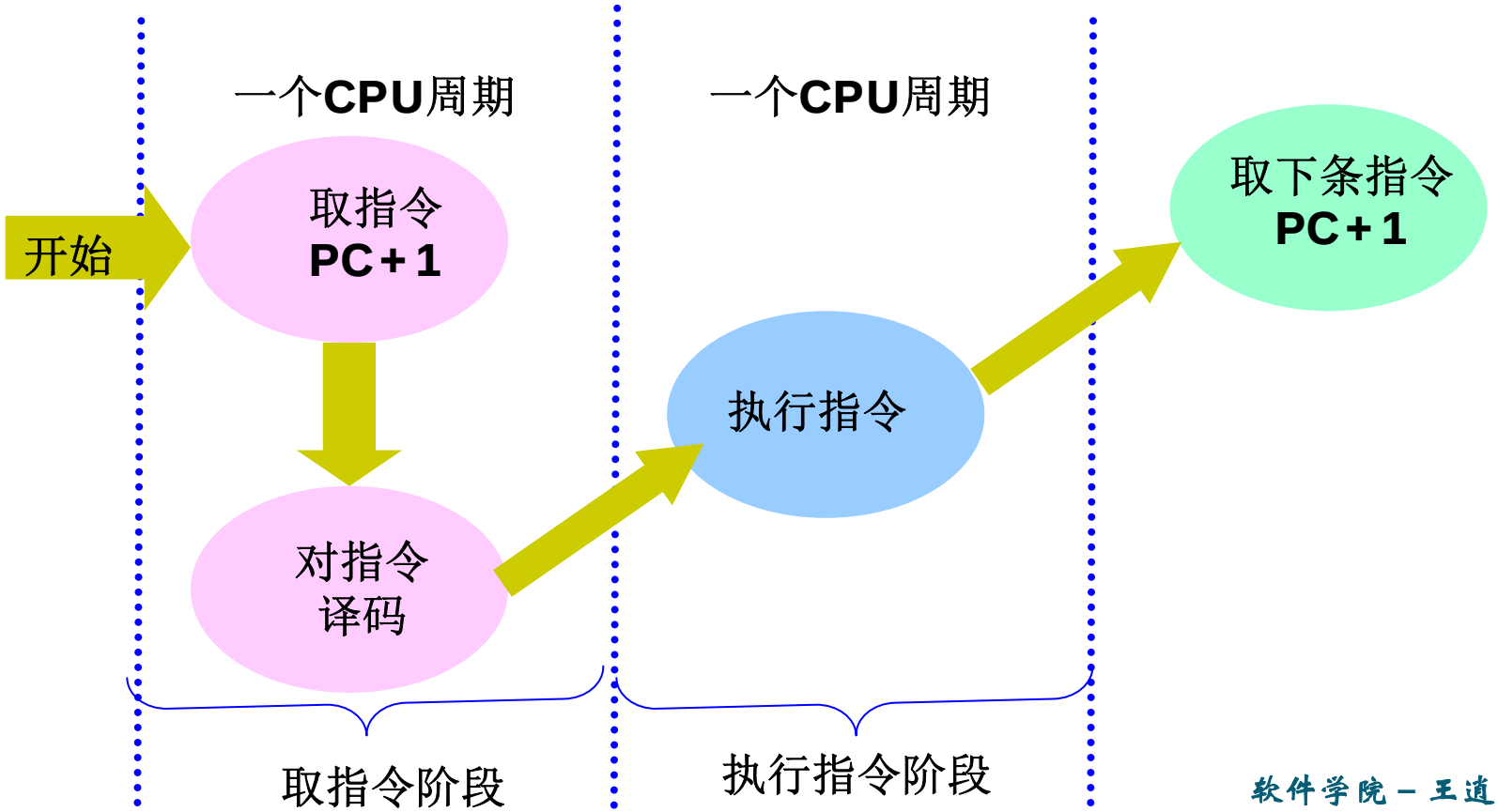
=120，作为数

]，将数据120

单元，它原先的

5.2.6 JMP指令的指令周期

- JMP 101：立即数型1地址指令。
  - JMP指令周期=2CPU周期=1取指CPU周期+1执行CPU周期
  - 取指周期：从指存中取指令字→PC++→指令字译码
  - 执行周期：实现指令的功能101→PC；



JMP指令的取指周期

- CPU的动作：取指周期
  - 类似于MOV指令的取指周期；

① 程序计数器PC中装入第一条指令地址**105**（八进制）；

② PC的内容被放到指令地址总线ABUS（I）上，对指存进行译码，并启动读命令；

③ 从105号地址读出的JMP指令通过指令总线IBUS装入指令寄存器IR；

④ 程序计数器内容加1，变成**106**，为取下一条指令做好准备；

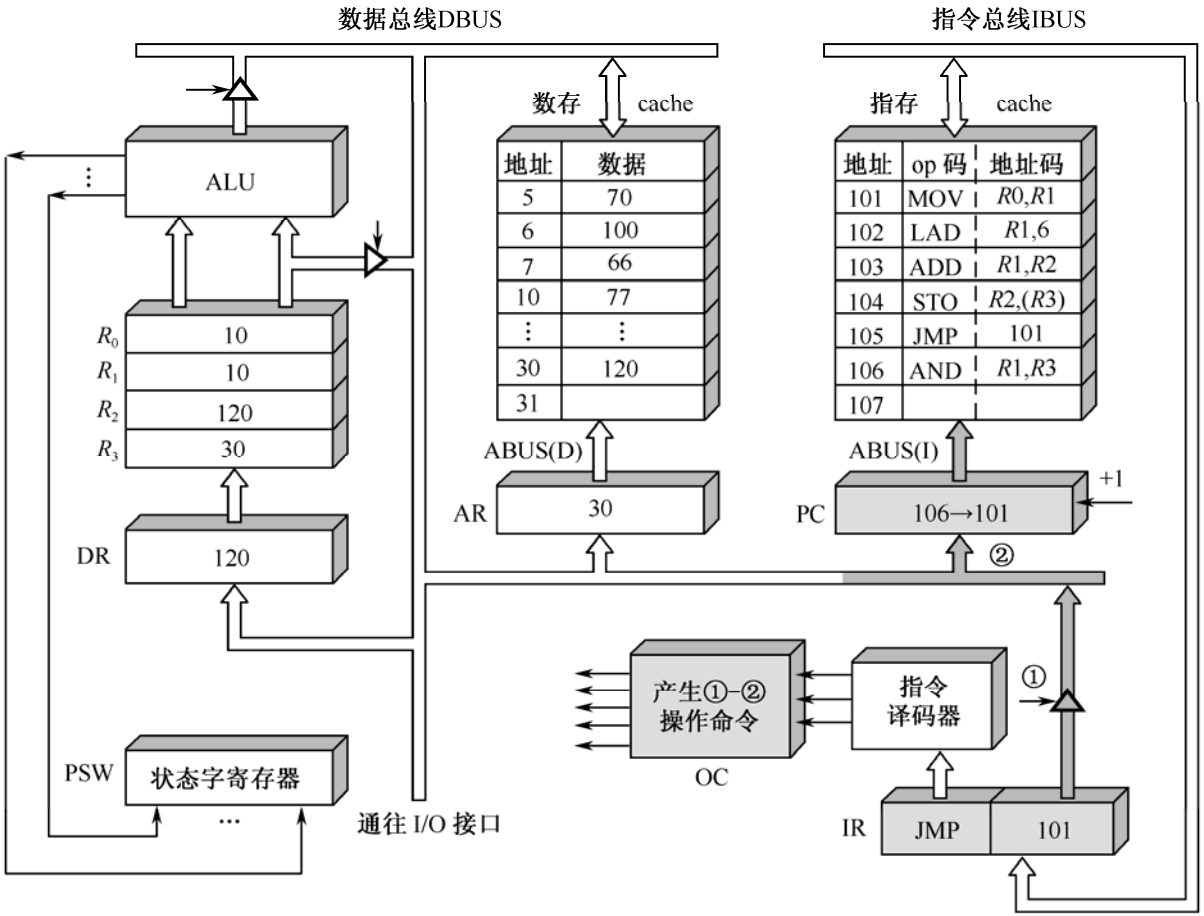
⑤ 指令寄存器中的操作码（OP）被译码；

⑥ CPU识别出是JMP指令，至此，取指周期即告结束。

JMP指令的执行周期

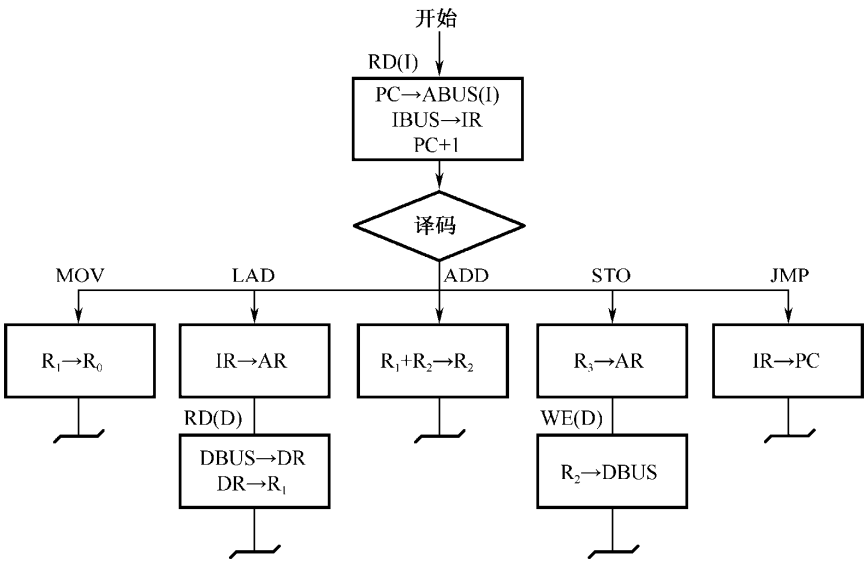
- CPU的动作：执行周期

- ② CC发出操作控制命令，将DBUS上的地址码101打入到程序计数器PC中，于是下一条指令不是从106号单元取出，而是转移到101号单元取出。至此JMP指令执行周期结束。



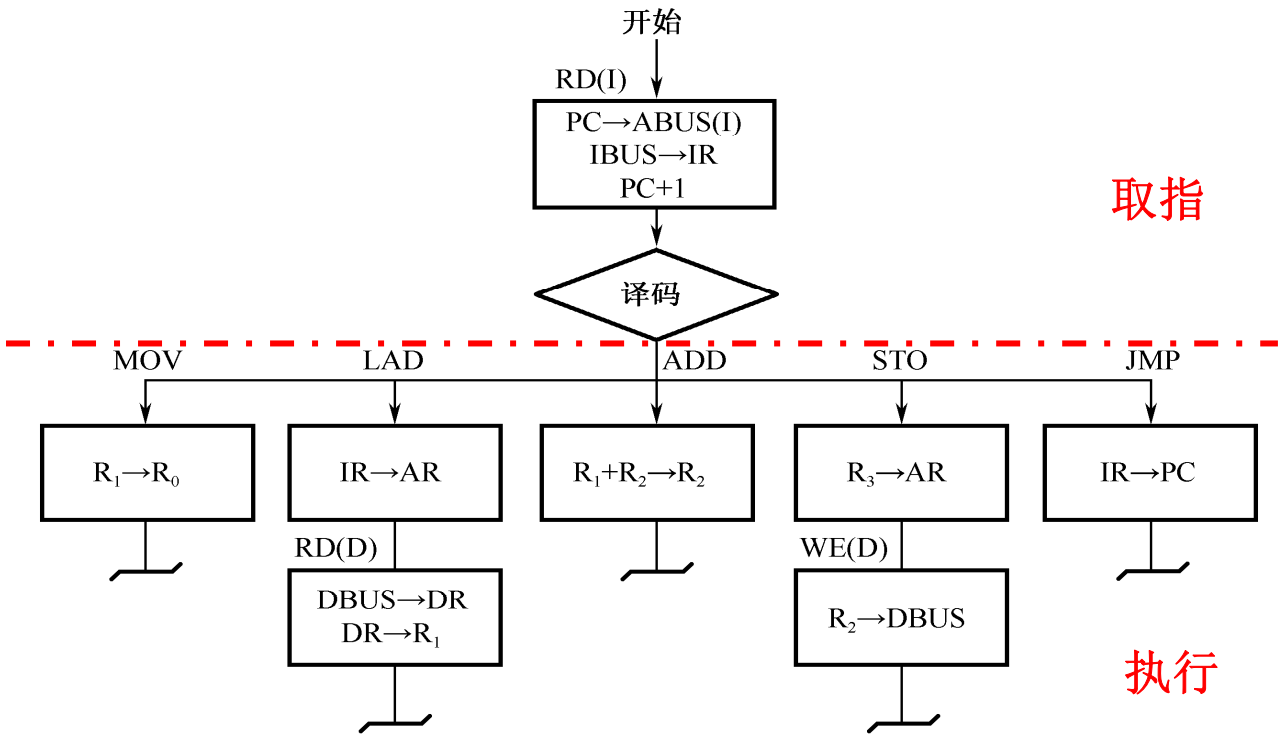
5.2.7 用方框图语言表示指令周期

- 在进行计算机设计时，可以采用方框图语言来表示指令的指令周期，实现计算机指令系统的设计。
  - 方框：代表一个CPU周期，方框中的内容表示数据通路的操作或某种控制操作。
  - 菱形：通常用来表示某种判别或测试，时间上它依附于紧接它的上一个方框的CPU周期，不单独占用一个CPU周期。
  - ~符号：公操作符号，表示当前指令执行完毕，转入公操作。
  - 公操作：取下一条指令(取指周期)或中断处理，通道处理等。
    - 所有指令的取值周期相同，是一种公操作。

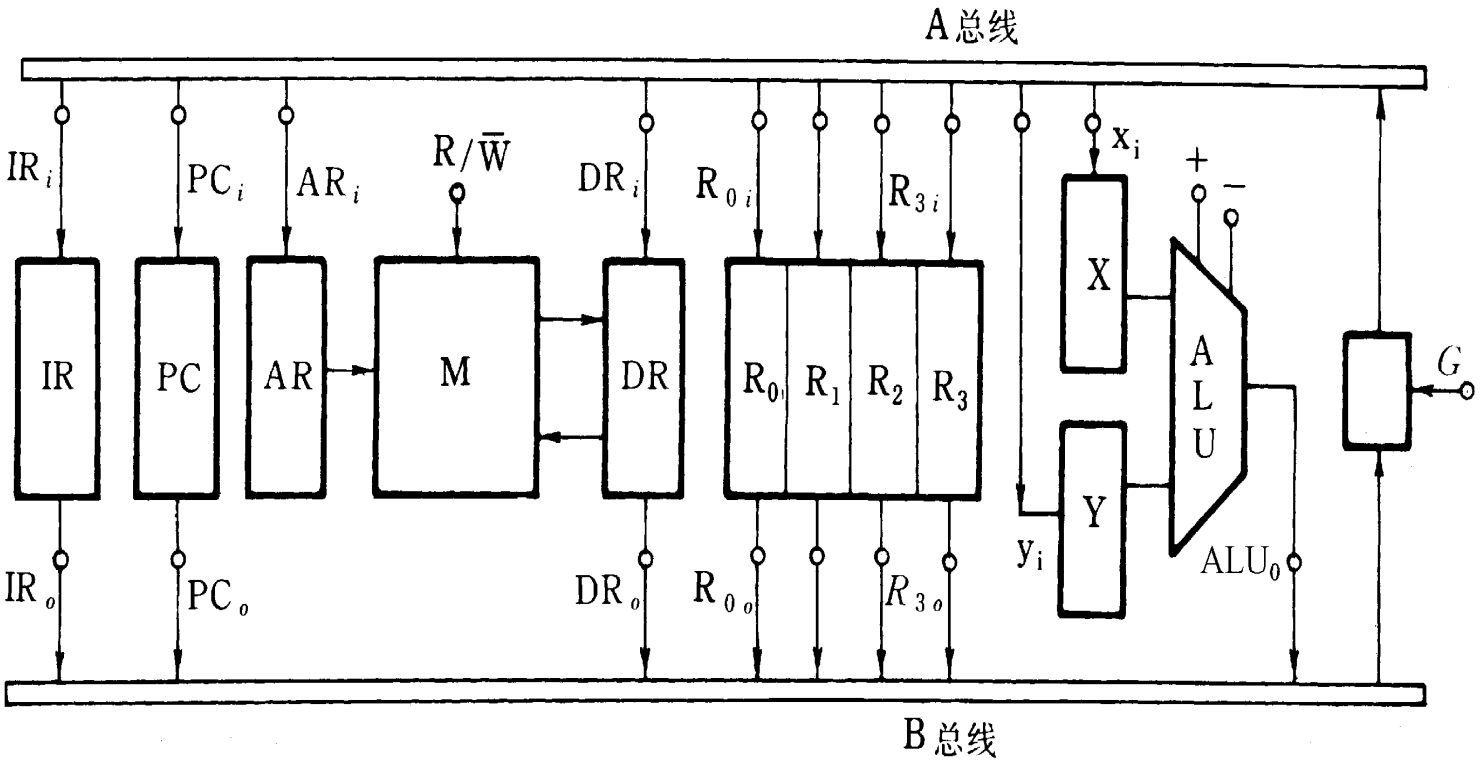


5.2.7 用方框图语言表示指令周期

- 方框图语言来表示表5-1P143指令的指令周期：
  - **取指周期**：所有指令的取指周期相同，一个CPU周期。
  - **执行周期**：MOV，ADD，JMP指令是1个CPU周期。LAD，STO指令是2个CPU周期。
  - **DBUS**：数据总线；**ABUS(D)**：数存地址总线；**ABUS(I)**：指存地址总线；**RD(D)**：数存读；**RD(W)**：数存写；**RD(I)**：指存读；



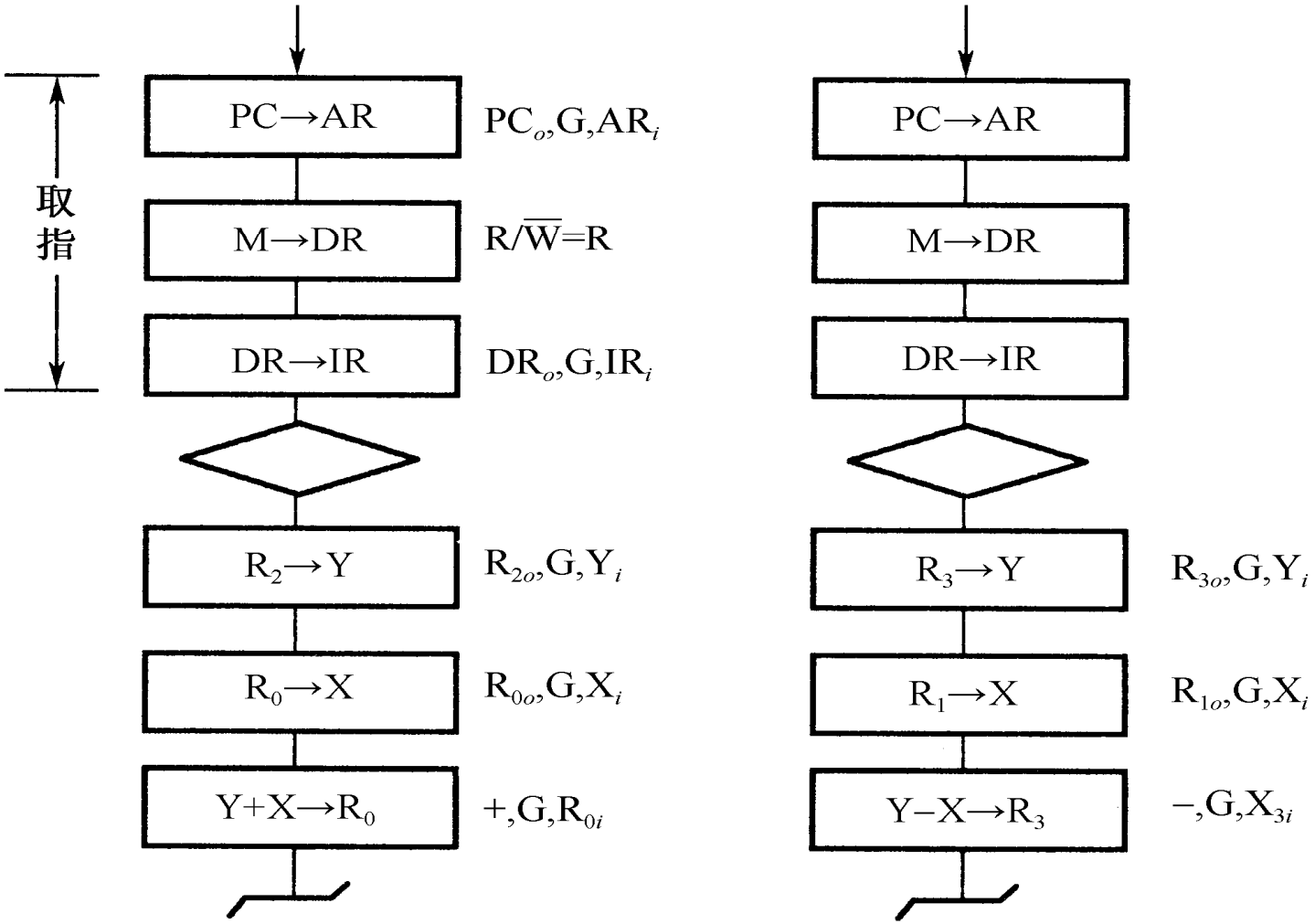
- 例：P151,双总线结构的数据通道：
  - A总线：输入总线，输入数据传输总线；
  - B总线：输出总线，输出数据传输总线；
  - 微操作控制信号：实现某个功能涉及到的设备需要的控制信号；
    - 下标i：输入控制；
    - 下标o：输出控制；



数据缓

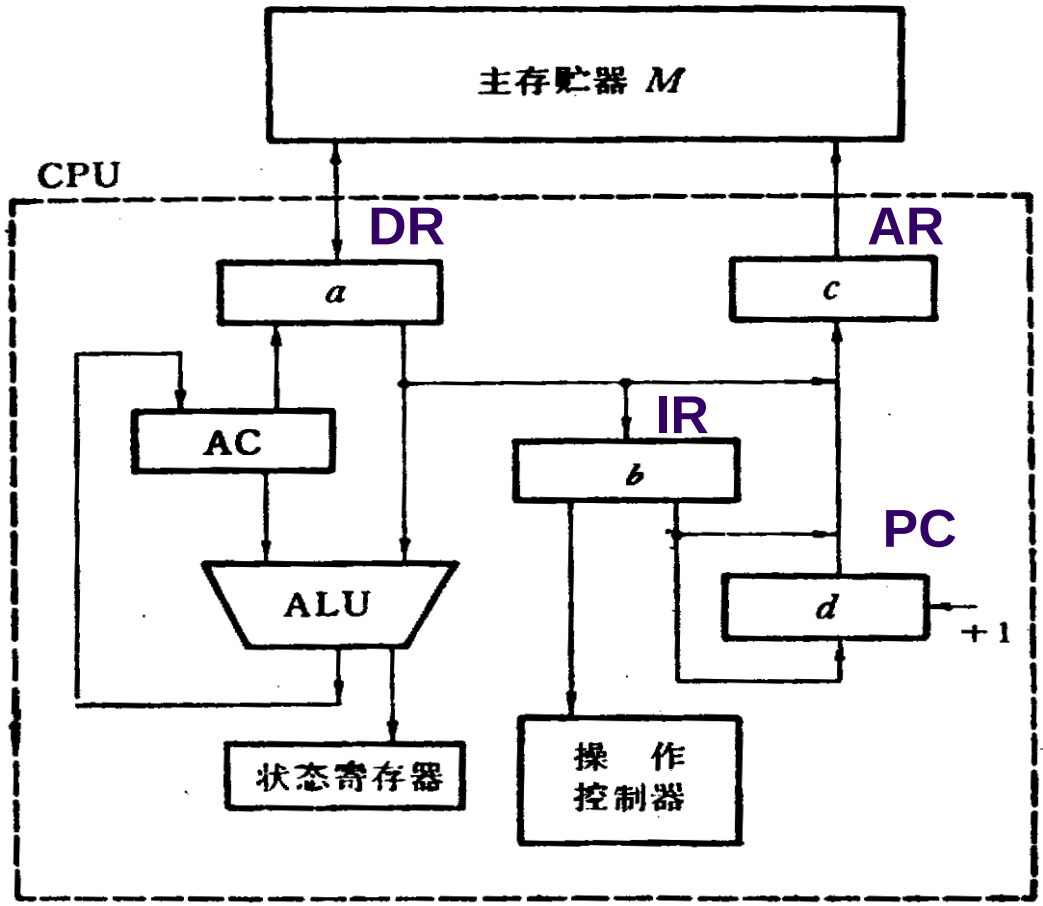
空制信号

- 例：P151，双总线结构的数据通道：



(a) 加法　　(b) 减法

- 小结：
  - 一条指令周期包括一个取指令周期和至少一个执行周期组成
  - 指令周期是指取出并执行一条指令的时间，指令周期常常用若干个CPU周期数来表示，CPU周期也称为机器周期，而一个CPU周期又包含若干个时钟周期（也称为节拍脉冲或T周期）。
  - a，b，c，d寄存器分别是哪个寄存器？



5.3 时序产生器和控制方式

- 一个问题：
  - 用二进制码表示的指令和数据都放在内存里，那么CPU是怎样识别出它们是数据还是指令呢?
    - 从时间上来说：取指令事件发生在指令周期的第一个CPU周期中，即发生在“取指令”阶段，而取数据事件发生在指令周期的后面几个CPU周期中，即发生在“执行指令”阶段。
    - 从空间上来说：如果取出的代码是指令，那么一定经DR送往指令寄存器IR，如果取出的代码是数据，那么一定送往运算器。
  - 不同的CPU周期中，涉及到的2进制数据表示不同的含义，CPU应该实现不同放入操作，所有的操作应该有严格的时间先后顺序，每个操作应该有严格的时间控制，在规定的时间开始，在规定的时间结束。

5.3.1 时序信号的作用和体制

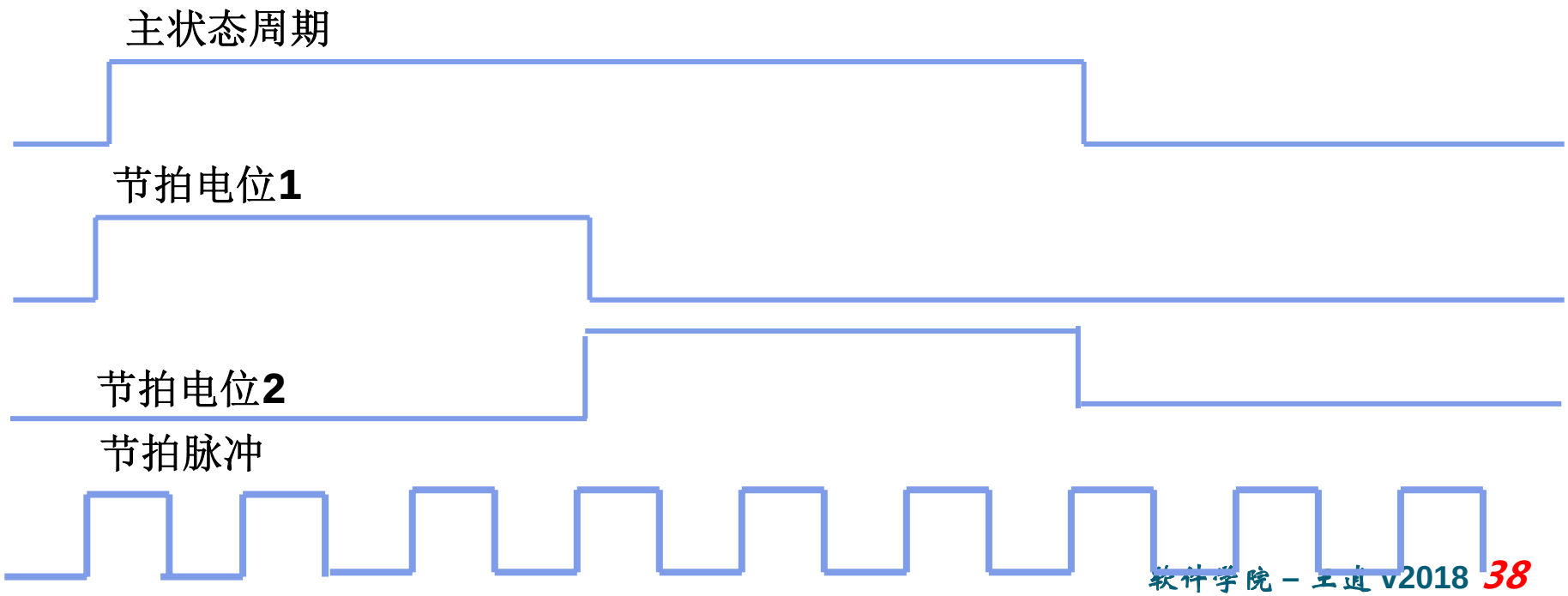
- 时序信号与时间标志：
  - 协调，指挥计算机各部分准确、迅速、有条不紊工作需要时间标志。
  - 时间标志：利用定时脉冲和不同的脉冲间隔，规定计算机当某个脉冲达到的时候，在脉冲间隔规定的时间内该去完成什么操作？
  - 实现一个程序的功能，需要按一定的顺序提供一系列的时间标志，一系列时间标志构成了时序信号。实现信号提供时间标志。

5.3.1 时序信号的作用和体制

- 时序信号作用：
  - CPU中的控制器用它指挥机器的工作；
  - CPU可以用时序信号/周期信息来辨认从内存中取出的是指令（取指）还是数据（执行）；
  - 一个CPU周期中时钟脉冲对CPU的动作有严格的约束；
  - 操作控制器发出的各种信号是时间（时序信号）和空间（部件操作信号）的函数；
- 时序信号体制：
  - 基本体制：组成计算机硬件的器件特性决定了时序信号的基本体制是电位(0/1)—脉冲制(触发)。（表示的问题）
  - 根据设计方法的不同，产生时序信号的操作控制器分两类：
    - 时序逻辑型：硬连线控制器，三级体制：主状态周期----节拍电位----节拍脉冲；
    - 存储逻辑型：微程序控制器，二级体制：节拍电位----节拍脉冲；

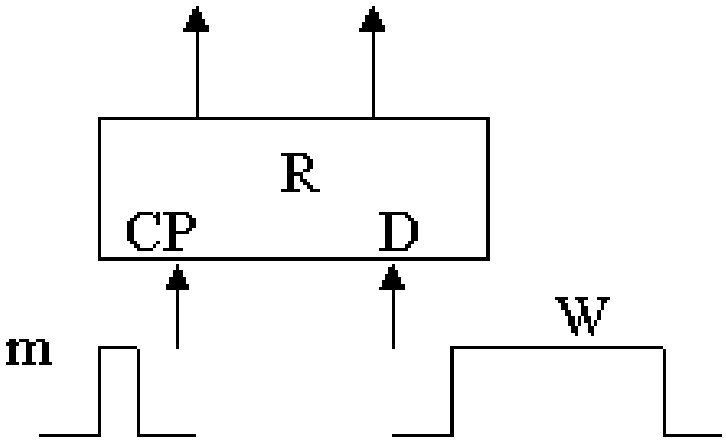
5.3.1 时序信号的作用和体制

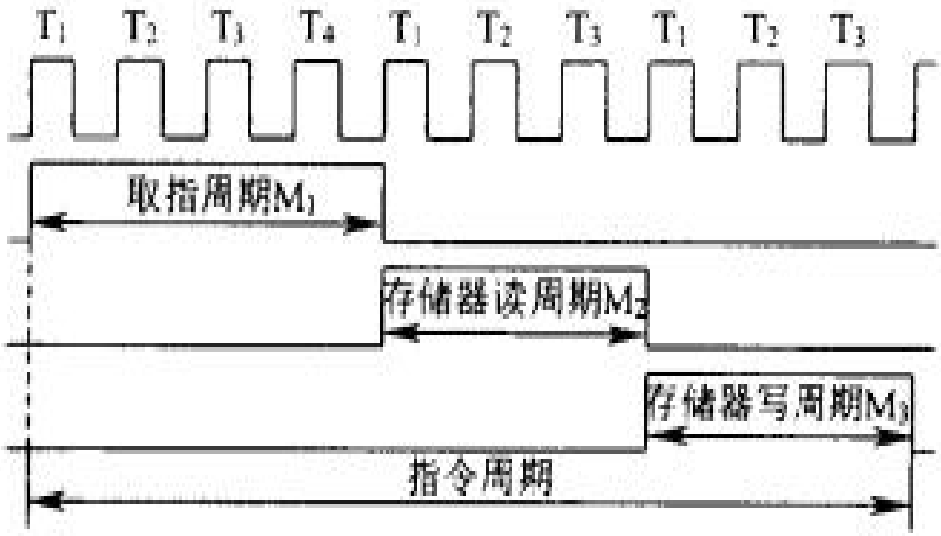
- 三级体制：硬布线控制器中，时序信号往往采用主状态周期-节拍电位-节拍脉冲三级体制。
  - 主状态周期（指令周期）：包含若干个节拍周期，可以用一个触发器的状态持续时间来表示；
  - 节拍电位（机器周期）：表示一个CPU 周期的时间，包含若干个节拍脉冲；
  - 节拍脉冲（时钟周期）：表示较小的时间单位；



5.3.1 时序信号的作用和体制

- 二级体制：微程序控制器中，时序信号则一般采用节拍电位/节拍脉冲二级体制。
  - 数据准备好后，以电位的方式送触发器
  - 控制信号来到后，用一个脉冲信号把数据装入触发器

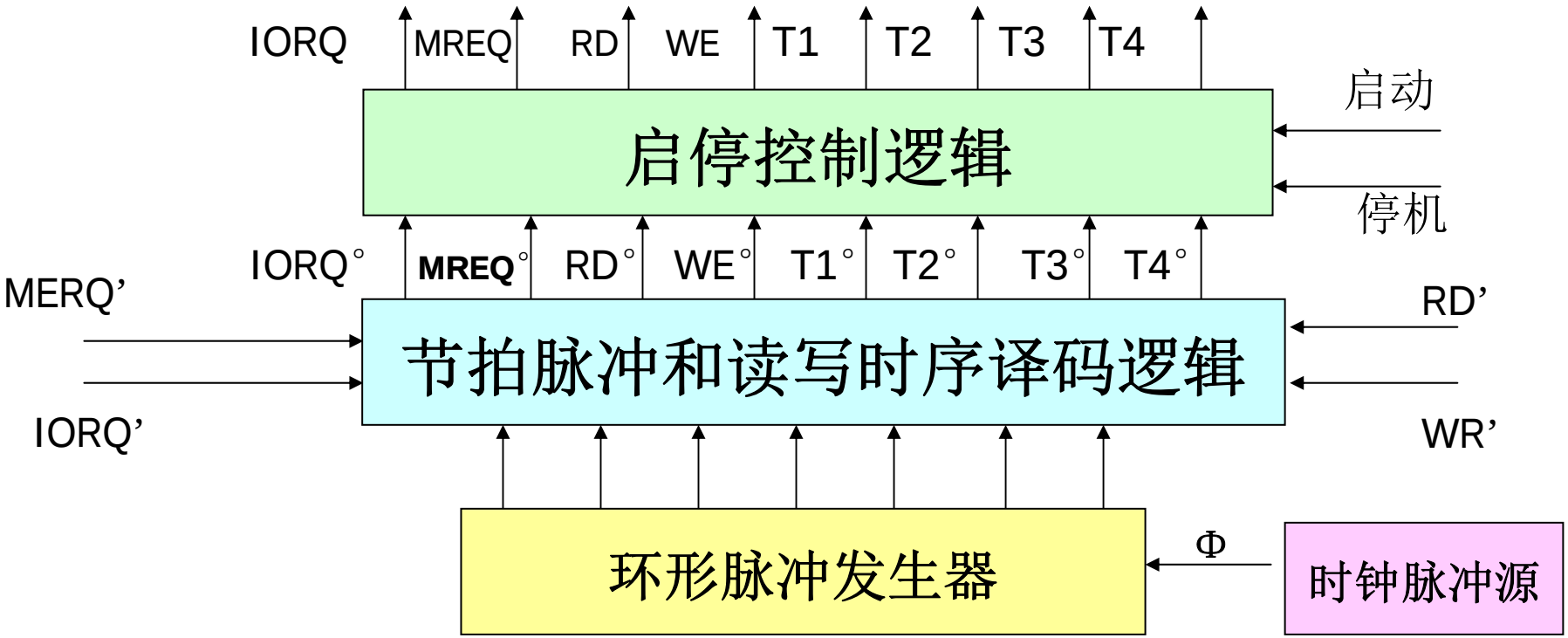




控制信号：脉冲　　数据：电位

5.3.2 时序信号产生器

- 产生时序信号，对各种操作实施时间上的控制。
  - 功能：产生时序信号
  - 构成：时钟源，环形脉冲发生器，节拍脉冲和读写时序译码逻辑，启停控制逻辑



5.3.3 控制方式

- 机器指令包含的CPU周期数反映了指令的复杂程度，不同CPU周期操作信号的数目和出现的先后次序也不相同。
- 控制方式：控制产生不同操作序列时序信号的方法。
  - 同步控制方式（指令的机器周期和时钟周期数不变）
    - 完全统一的机器周期执行各种不同的指令
    - 采用不定长机器周期
    - 中央控制于局部控制的结合
  - 异步控制方式
    - 每条指令需要多长时间就占多长时间
  - 联合控制方式
    - 大部分指令在固定的周期内完成，少数难以确定的操作采用异步方式
    - 机器周期的节拍脉冲固定，但是各指令的机器周期数不固定（微程序控制器采用）

5.4 微程序控制器

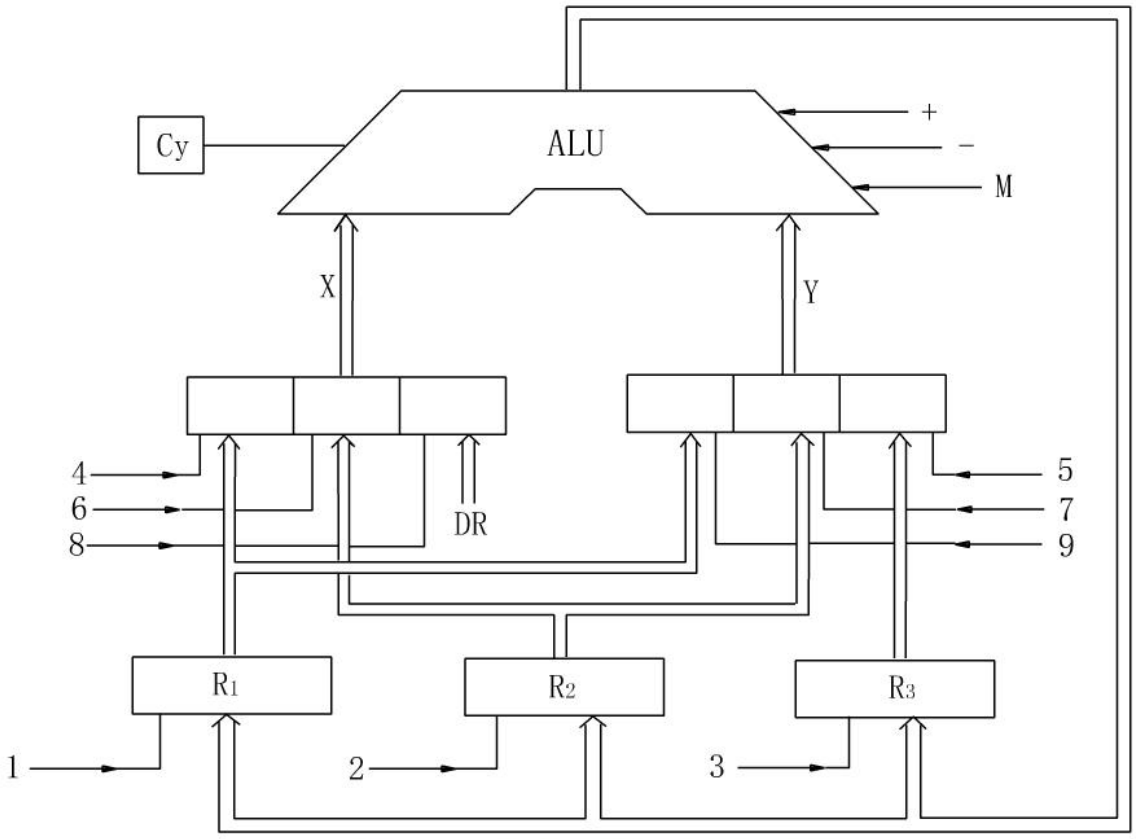
- 操作控制器：产生实现指令功能所需要的控制信号(时序信号)到方法。
  - 微程序控制器与硬布线控制器
  - 微程序控制器同硬布线控制器相比较，具有规整性、灵活性、可维护性等一系列优点。它利用软件方法（微程序设计技术）来设计硬件
- 微程序控制的基本思想：把执行指令需要的操作控制信号编成所谓的“微指令”，存放到一个只读存储器里（控制存储器CM）。当机器运行时，一条又一条地读出这些微指令，从而产生全机所需要的各种操作控制信号，控制相应部件执行所规定的操作。

5.4.1 微程序控制器原理

- CPU内部可以分为：控制部件和执行部件，使用控制线连接
  - 控制部件：提供执行程序需要的控制信号。
  - 执行部件：在控制信号的控制下依次实现程序中指令的功能。
  - 执行程序时，涉及到两组2信息：控制信息和执行信息。
- 基本概念：
  - 微命令：控制部件通过控制线向执行部件发出各种控制命令，通常把这种控制命令称为微命令。是控制序列的最小单位。
    - 例如：打开或关闭控制门的电位信号、某个寄存器的输入脉冲等。
    - 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所进行的操作，称为微操作。
    - 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。
  - 微操作可分为相容的和互斥的两种：
    - 互斥的微操作：是指不能同时或不能在同一个节拍内并行执行的微操作。可以使用不同的微操作编码来表示；
    - 相容的微操作：是指能够同时或在同一个节拍内并行执行的微操作。可以在一个微操作编码在使用使用不同位的值表示；

5.4.1 微程序控制器原理

- 运算器模型：寄存器R1，R2，R3中的数据可以使用多路开关X，Y传送到ALU实现+，-，M(传送)运算。
  - 微命令包括：1,2,3,4,5,6,7,8,9，+，-，M
  - 互斥的微操作： +，-，M； 4，6，8； 5，7，9等；
  - 相容的微操作： 1，2，3；(4，6，8)与(5，7，9)等

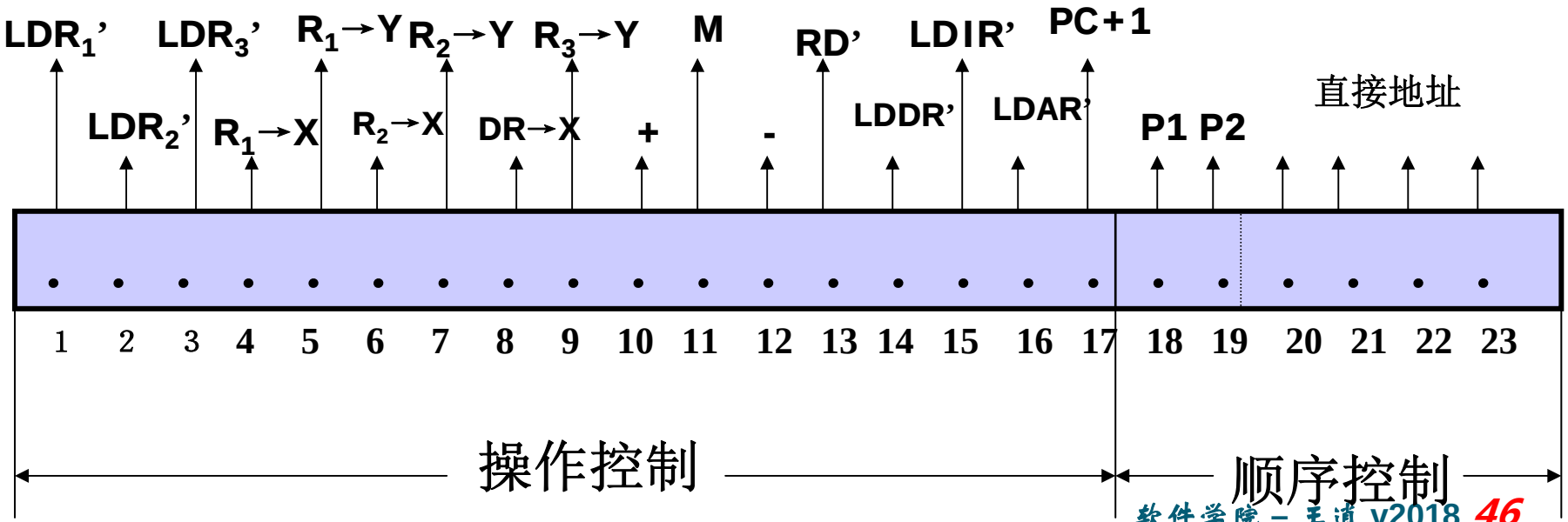


5.4.1 微程序控制器原理

- 基本概念：
  - 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（Microinstruction）。
    - 它是微命令的组合，微指令存储在控制器中的控制存储器中。
  - 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，表明发微指令，为0，不发出；
      - 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，实现信号同步；
    - 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。
  - 微程序：一系列微指令的有序集合就是微程序。
    - 一条机器指令的功能由若干条微指令序列实现，对应一个微程序。所有微程序的总和对应整个指令系统。

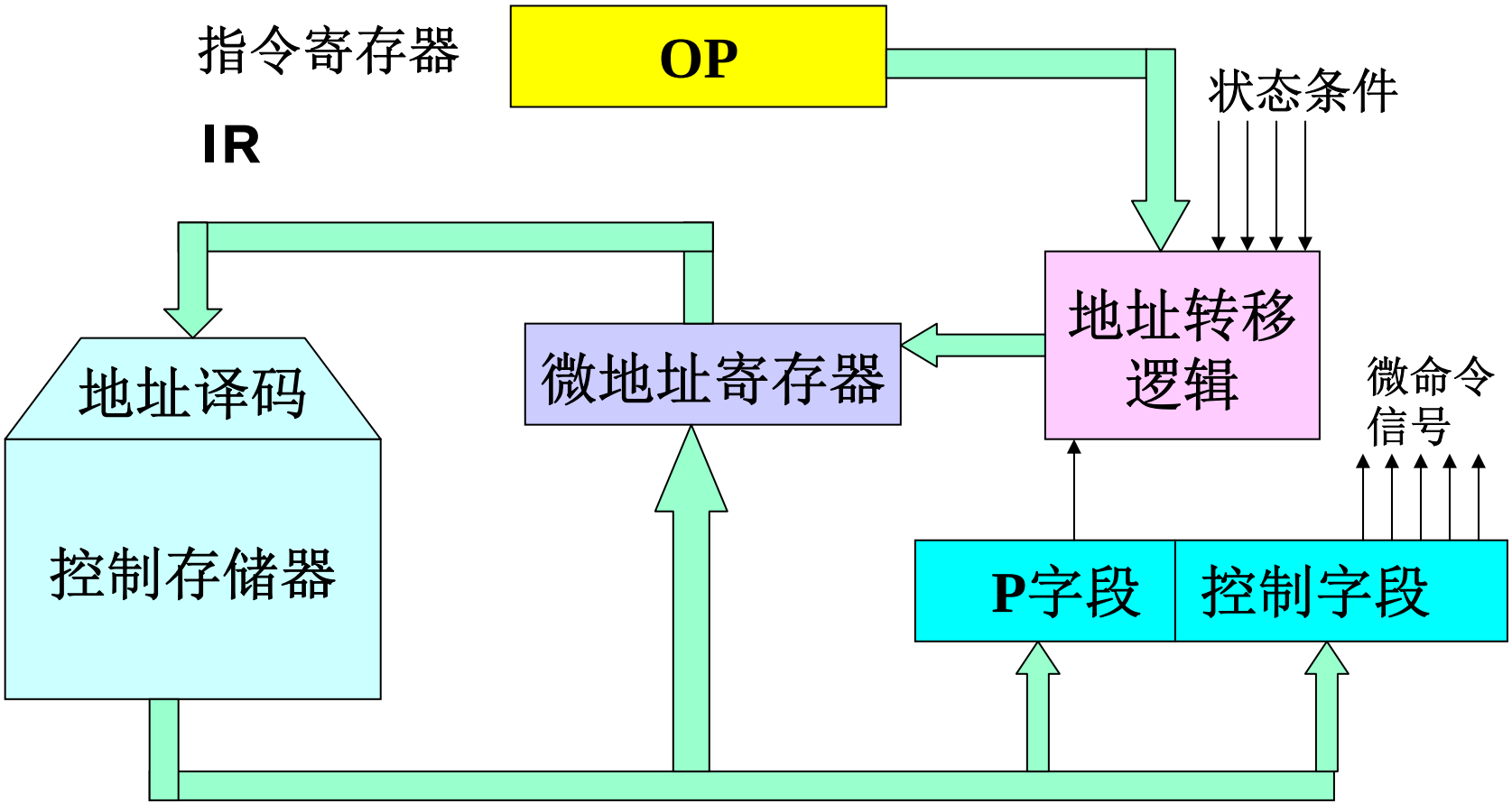
5.4.1 微程序控制器原理

- 微指令举例：
  - 微指令：字长23位
  - 操作控制字段17位，提供17个微命令；取值为1，发出对应的微命令，实现规定的微操作。取值为0，不发出微操作。
  - 顺序控制字段6位：确定下一条微指令的微地址。由两位判别测试位P1P2和4位地址位组成。
    - P1P2=00，4位地址位是正确的微地址。
    - P1P2≠00，需要4位地址位进行修改形成正确的微地址。



5.4.1 微程序控制器原理

- 微程序控制器的原理框图：
  - 组成：控制寄存器，微指令寄存器＝微地址寄存器＋微命令寄存器，地址转移逻辑。



微程序控制器原理框图

5.4.1 微程序控制器原理

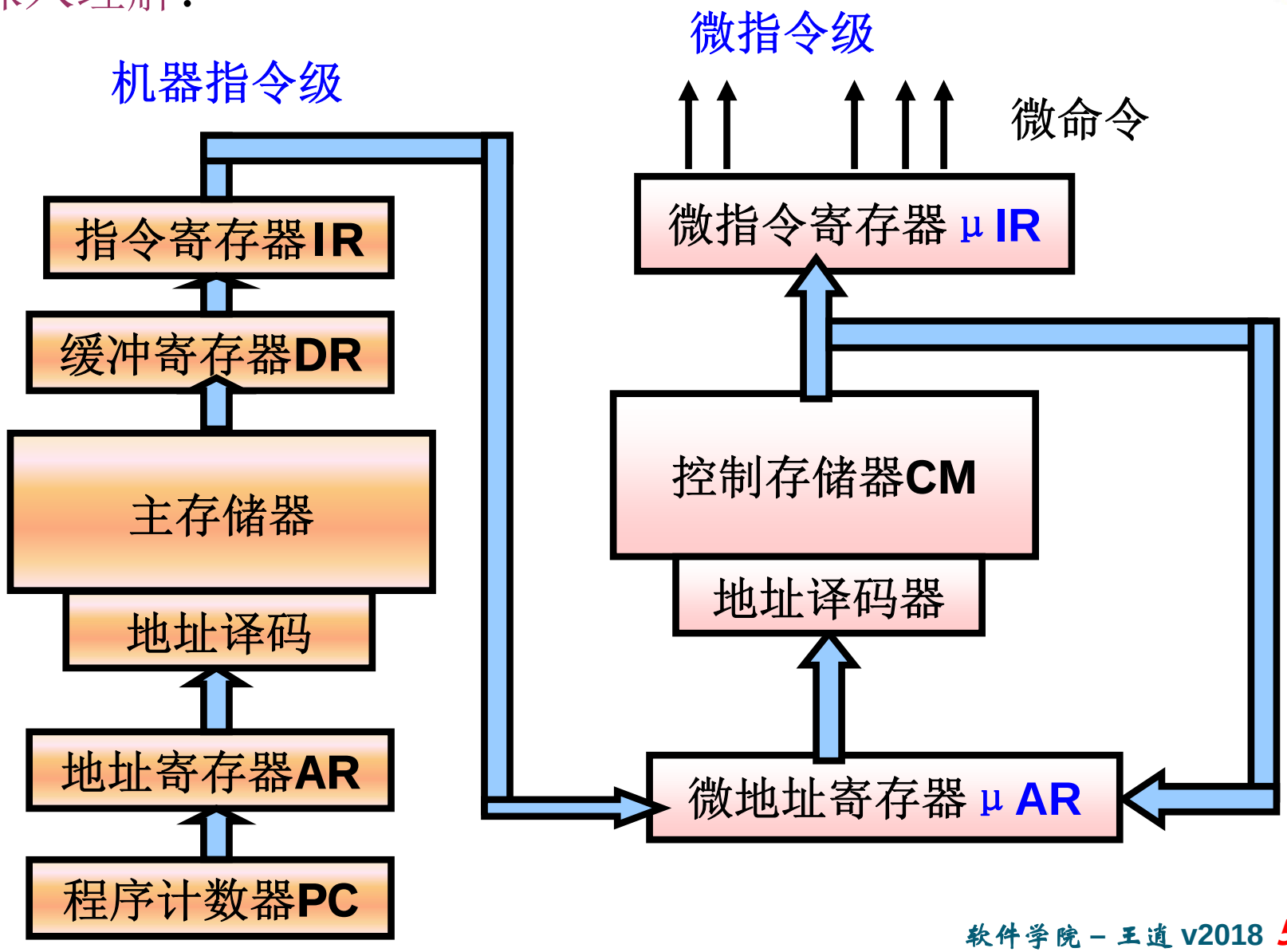
- 基本概念：
  - **微地址**：用于在控制寄存器CM中寻址某条微指令的地址。内存地址？
  - **控制存储器CM**：控存，微程序控制器的核心部件，用来存放微程序。是一种高速只读存储器。
  - **微指令寄存器**：用来存放从CM取出的正在执行的微指令，它的位数同微指令字长相等。
  - **地址转移逻辑**：用来产生初始微地址和后继微地址，以保证微指令的连续执行。
  - **微地址寄存器**：保存下一条微指令的微地址。
  - **微程序控制器的工作过程**：取微指令→执行微指令→取位置→……

5.4.1 微程序控制器原理

- 深入理解：
  - 微程序控制器的工作过程实质上就是在微程序控制器的控制下，计算机执行机器指令的过程。
  - 从控存中取出一段“取机器指令”用的微程序，称为取指微程序，这是一段公用的微操作，其首址通常放在“0”号微地址单元。该微程序完成从主存中读取机器指令并送往指令寄存器。
  - 机器指令操作码通过微地址形成部件，产生对应的微程序入口地址，并送入微地址寄存器。
  - 逐条取出对应的微指令，每一条微指令提供一个微命令序列，控制有关的微操作。
  - 执行完对应于一条机器指令的一段微程序后，返回到取指微程序的入口，以便取出下一条机器指令。不断重复，直至程序执行完毕。

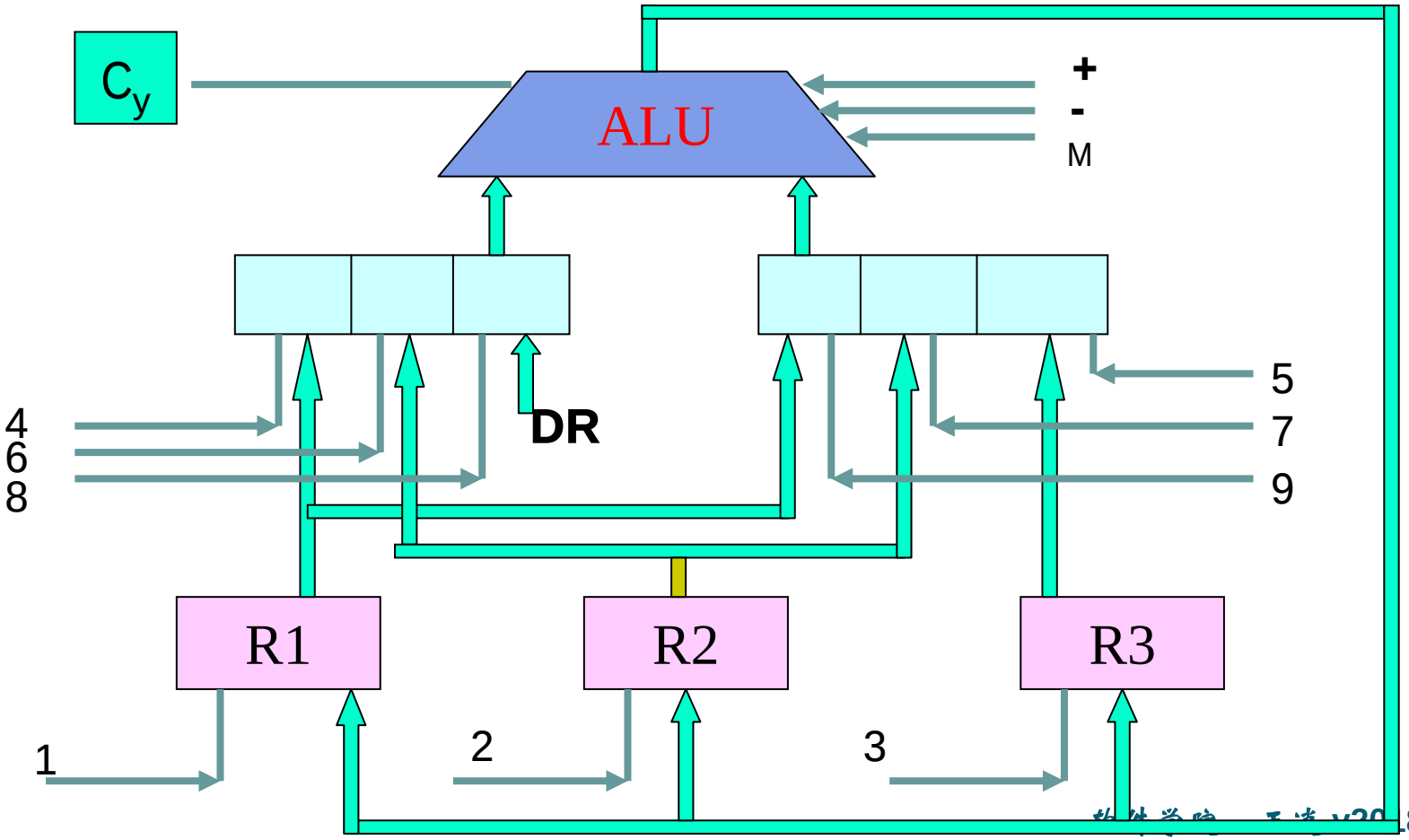
5.4.1 微程序控制器原理

- 深入理解：



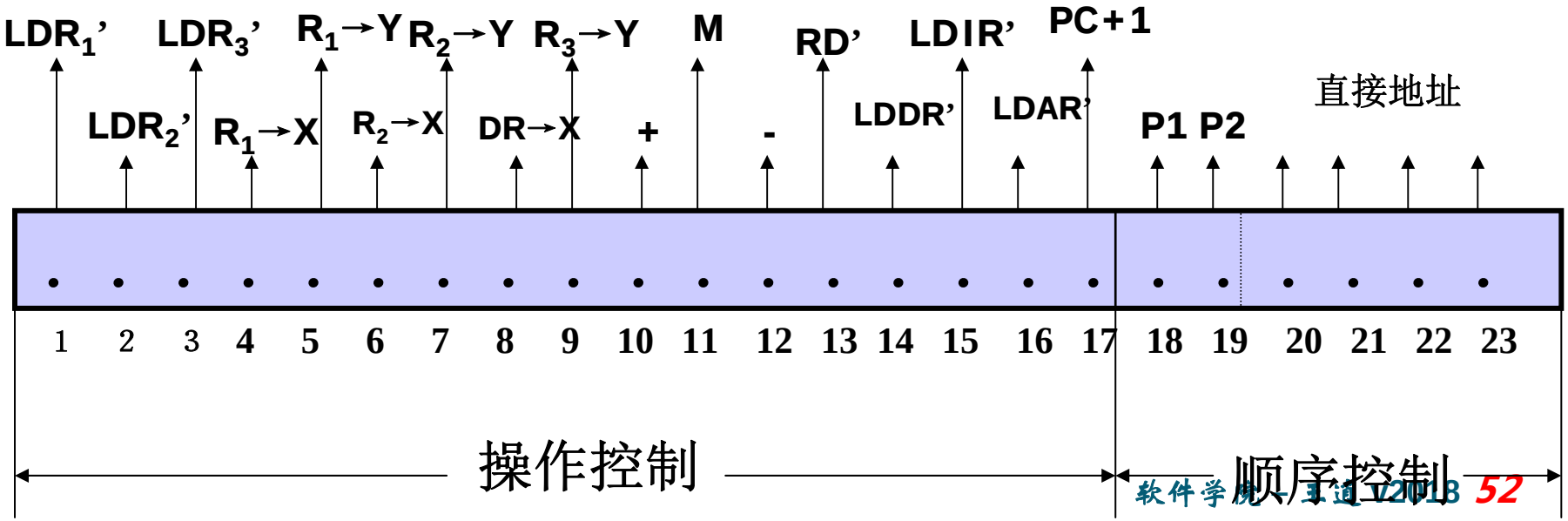
5.4.1 微程序控制器原理

- 微程序举例：
  - R1=3 R2=5 R1+R2->R3
  - 控4=1，R1->多路开关->X；控7=1，R2->多路开关->Y
  - 控+=1，ALU完成3+5；控3=1 ，3+5->R3



5.4.1 微程序控制器原理

- 微指令格式：字长23位
  - 操作控制字段17位，提供17个微命令；取值为1，发出对应的微命令，实现规定的微操作。取值为0，不发出微操作。
  - 顺序控制字段6位：确定下一条微指令的微地址。由两位判别测试位P1P2和4位地址位组成。
    - P1P2=00，4位地址位是正确的微地址。
    - P1=1，判别测试操作码OP形成正确的微地址。
    - P2=1，判别测试标志位形成正确的微地址。

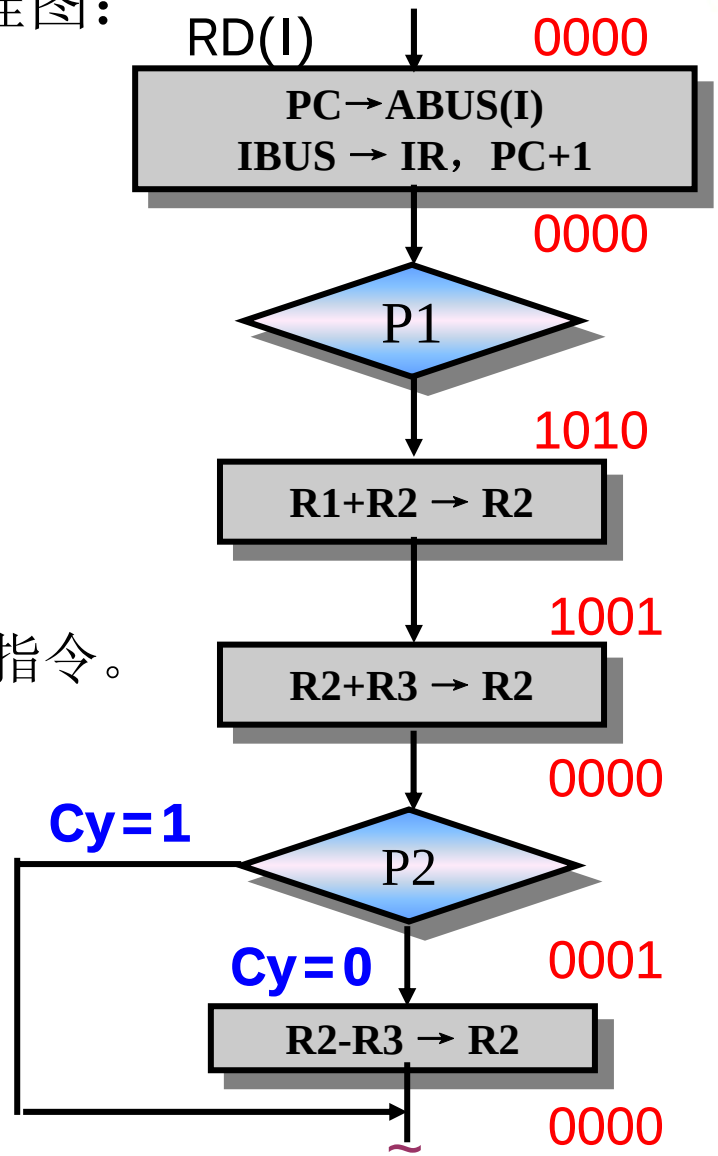


5.4.1 微程序控制器原理

- 设计实现”**10进制加法**”指令的微程序
  - 运算规则：
    - 参加运算的数据用BCD码表示；
    - 和大于9时，便产生进位，要进行加6修正；
    - 和小于等于9时，结果是正确的。
  - 运算实现：
    - 假定数据a和b已存放在R1和R2两寄存器中，数6存放在R3寄存器中。
    - 算法要求先进行a+b+6运算，然后判断结果有无进位；
    - Cy=1不减6，当Cy=0，减6，从而可以获得正确的结果。
  - 其他：
    - 数据通道：P144 图5.5；
    - 运算器：P156 图5.20；

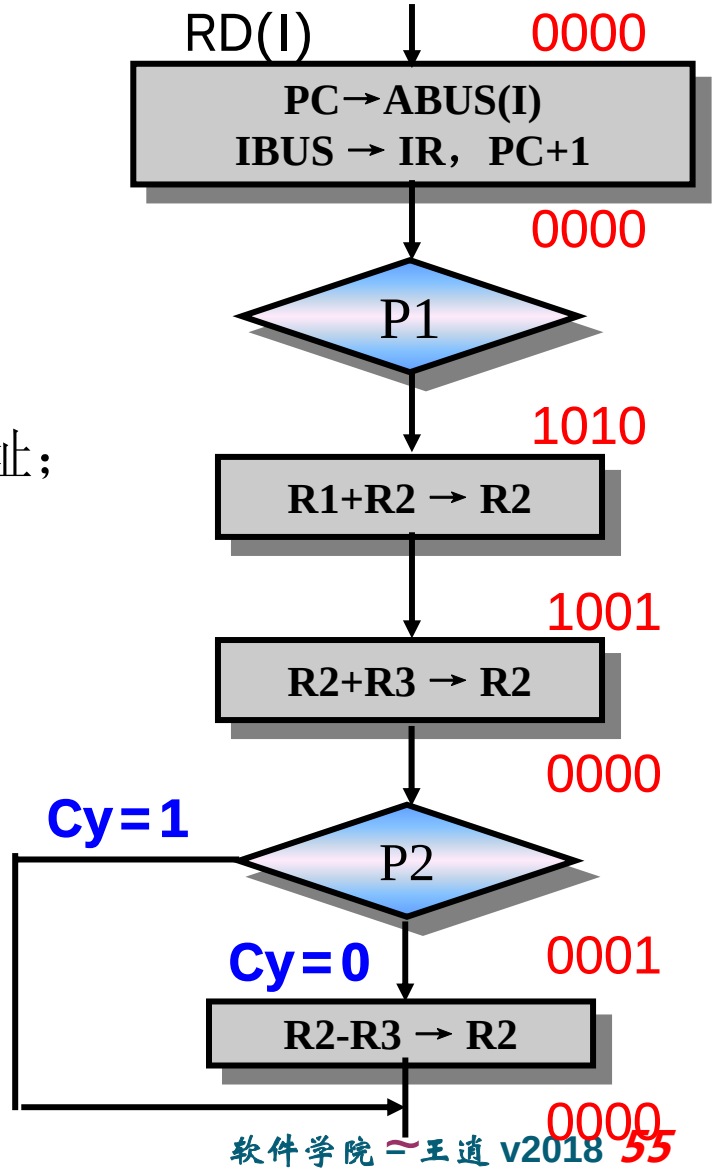
5.4.1 微程序控制器原理

- ”10进制加法”指令的微程序流程图：
  - 4条伪指令，方框表示。
  - 两次判断，菱形表示。
    - 微指令1：取指周期；
    - 微指令2：R1+R2→R2；
    - 微指令3：R2+R3→R2；
    - 微指令4：R2-R3→R2；
    - 微地址：方框右上角数字。
    - 判别测试操作依附于第1条微指令。



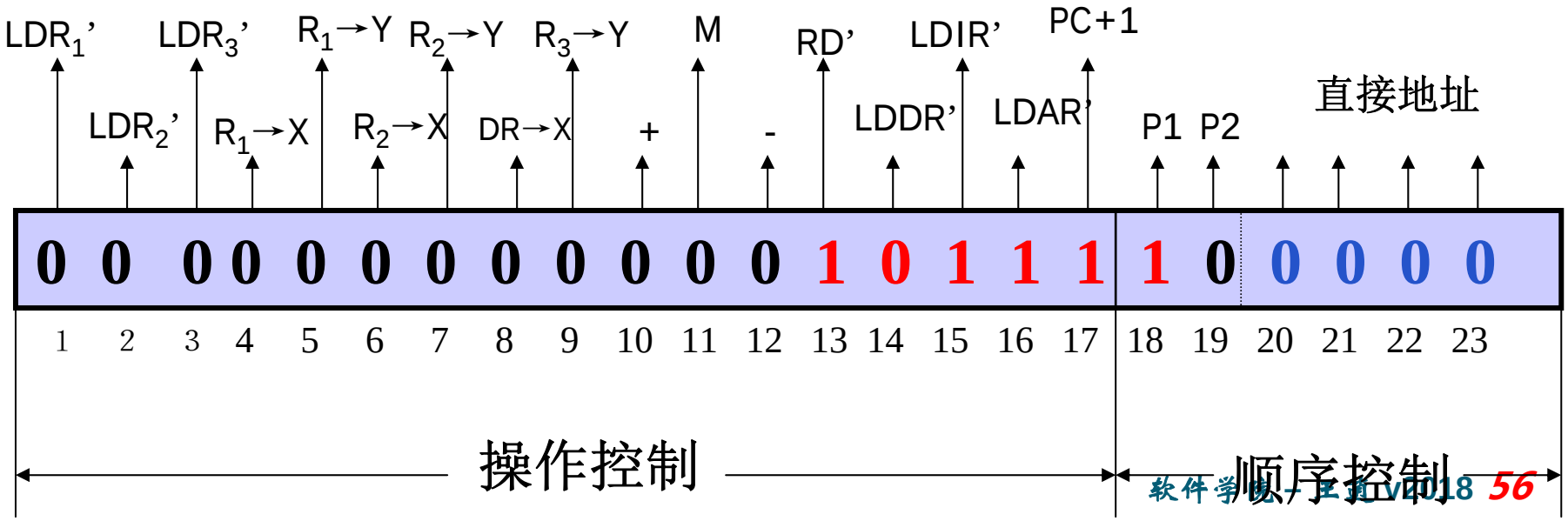
5.4.1 微程序控制器原理

- 微指令1：取指周期，取要执行的机器指令；
  - 3个任务：
    - 任务1：根据PC值取“10进制加法”机器指令→IR寄存器；
    - 任务2：PC++；
    - 任务3：对机器指令的OP用P1进行判断测试，给出下一条微指令的地址；



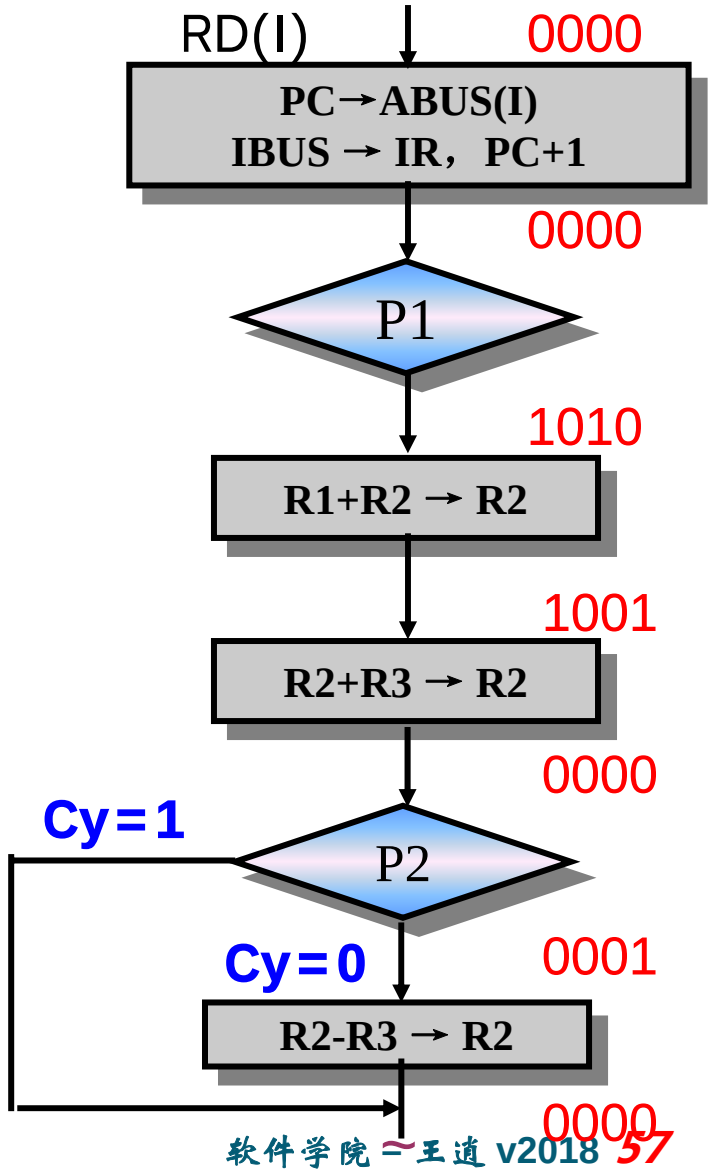
5.4.1 微程序控制器原理

- 微指令1：微指令编码；
  - 操作控制字段有五个微命令：
    - 第16位**LDAR’=1**，将PC→ABUS(I)；
    - 第13位**RD’=1**，执行指存读操作，取出“十进制加法”机器指令→IBUS；
    - 第15位**LDIR’=1**，将IBUS上的机器指令→指令寄存器IR；
    - 第17位**PC+1=1**，程序计数器加1，做好取下一条机器指令；
    - 第18位**P1=1**，判别测试机器指令的OP，将下一条微指令的地址从0000修改为1010（假定）；



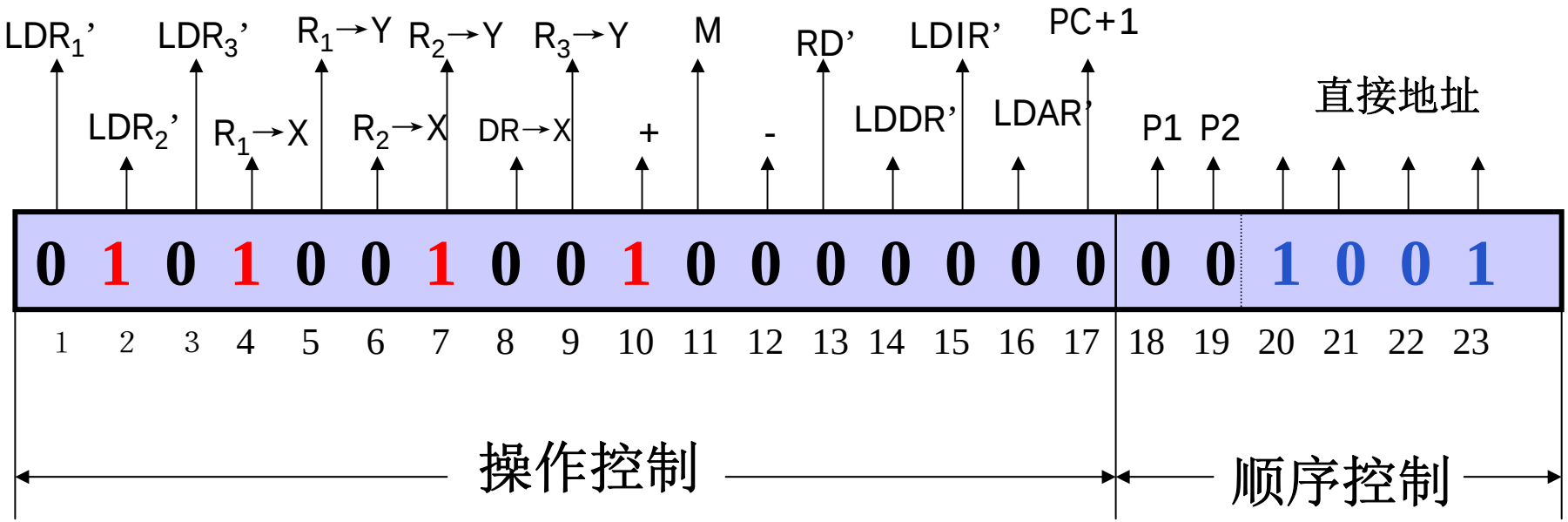
5.4.1 微程序控制器原理

- 微指令2：R1+R2→R2；



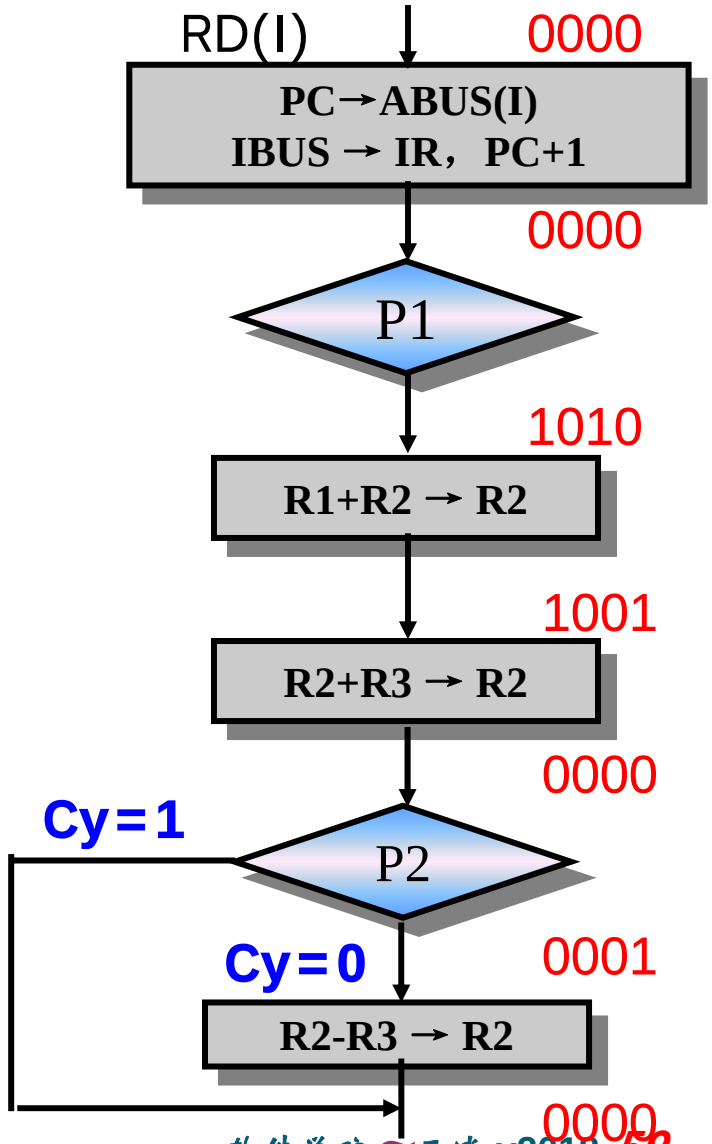
5.4.1 微程序控制器原理

- 微指令2：微指令编码；
  - 操作控制字段有4个微命令：
    - 第4位**R1→X=1**，将R1→X；
    - 第7位**R2→Y=1**，将R2→Y；
    - 第10位**+=1**，将X+Y即(R1+R2)；
    - 第2位**LDR2=1**，将X+Y →R2；
    - 第18位19位**P1P2=0**，下一条微指令地址1001。



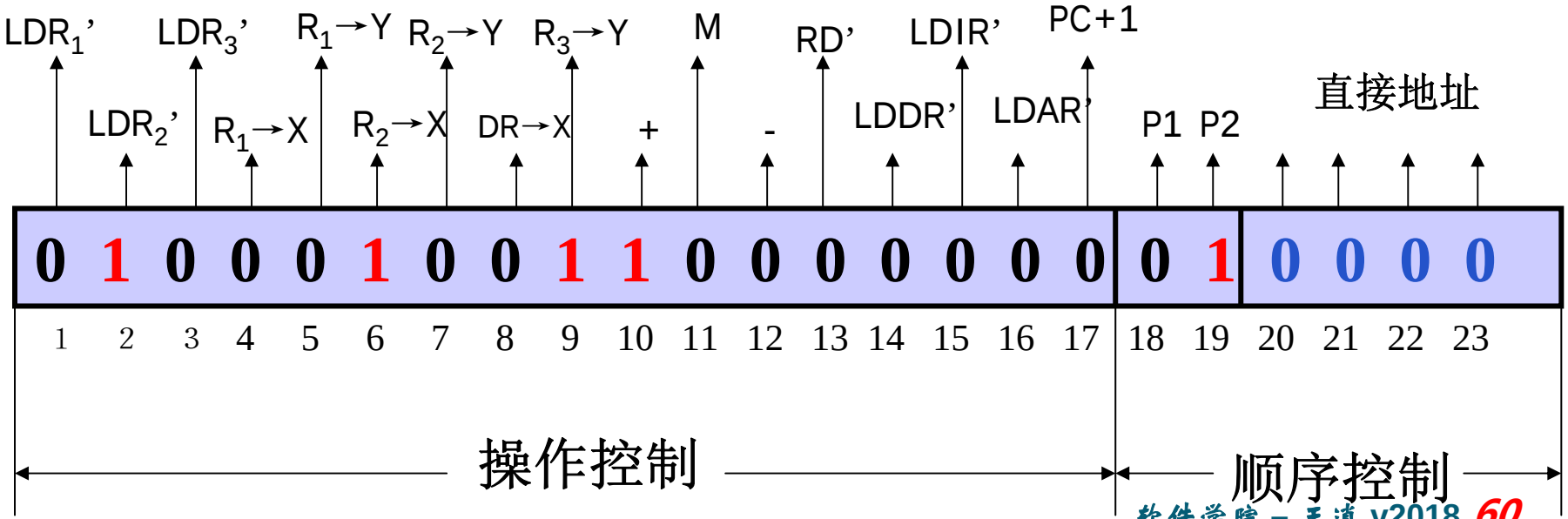
5.4.1 微程序控制器原理

- 微指令3：R2+R3→R2；



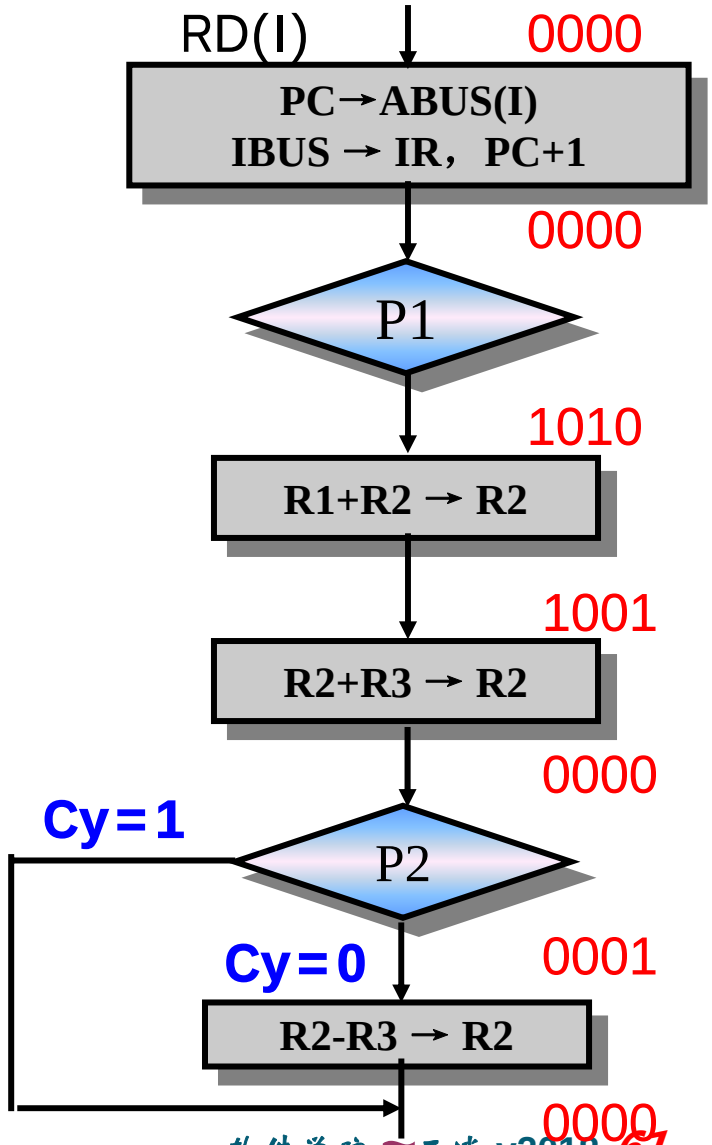
5.4.1 微程序控制器原理

- 微指令3：微指令编码；
  - 操作控制字段有5个微命令：
    - 第6位**R2→X=1**，将R2→X；
    - 第9位**R3→Y=1**，将R3→Y；
    - 第10位**+=1**，将X+Y即(R2+R3)；
    - 第2位**LDR2=1**，将X+Y →R2；
    - 第19位**P2=1**， 判别测试Cy， Cy=0，下一条微指令的地址=0001， Cy=1，下一条微指令的地址=0000（假定）。



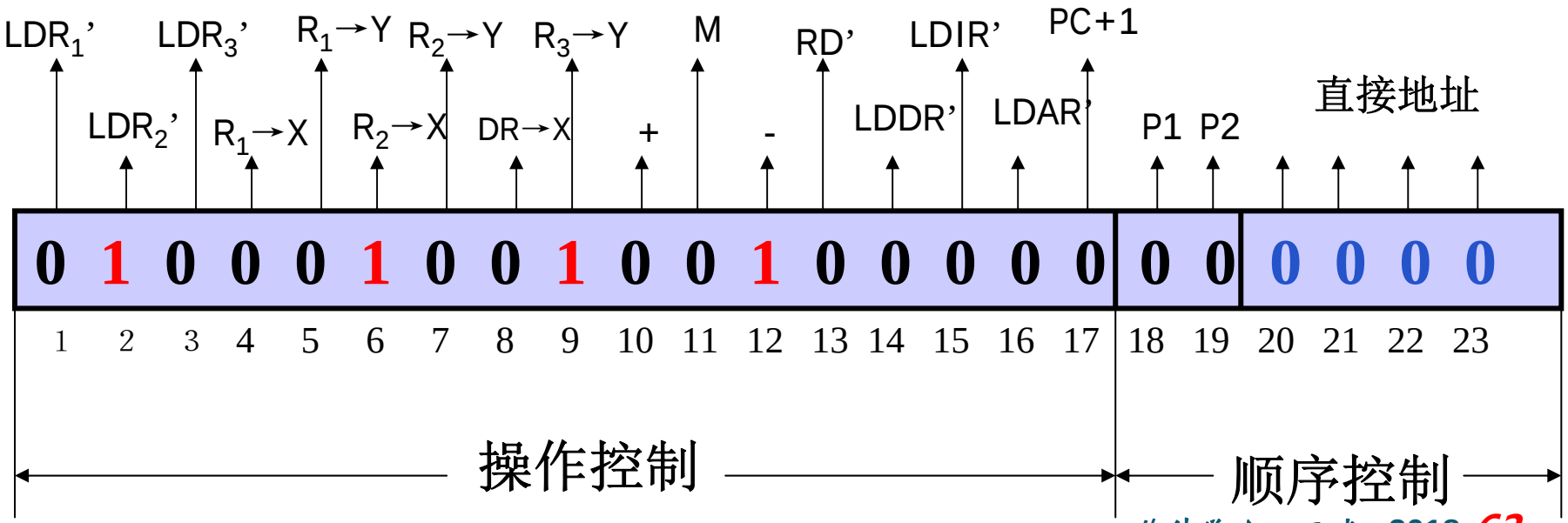
5.4.1 微程序控制器原理

- 微指令4：R2-R3→R2；



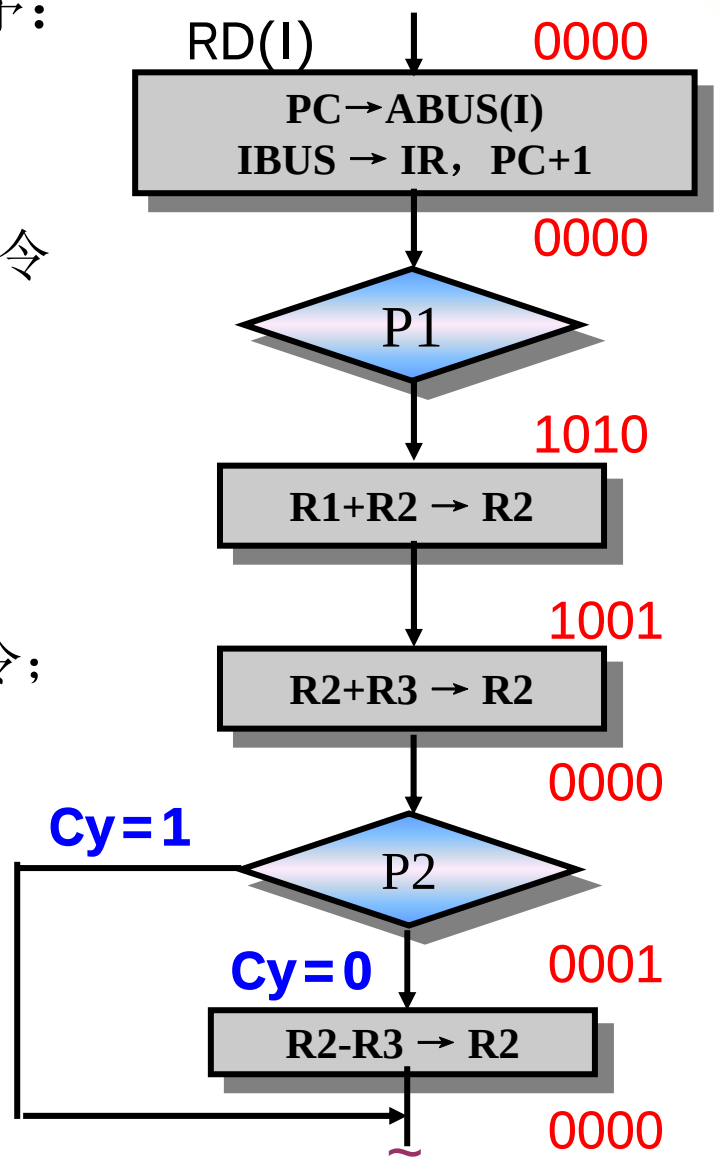
5.4.1 微程序控制器原理

- 微指令4：微指令编码；
  - 操作控制字段有4个微命令：
    - 第6位**R2→X=1**，将R2→X；
    - 第9位**R3→Y=1**，将R3→Y；
    - 第12位**-=1**，将X-Y即(R2-R3)；
    - 第2位**LDR2=1**，将X-Y →R2；
    - 第18位19位**P1P2=0**，下一条微指令地址0000。



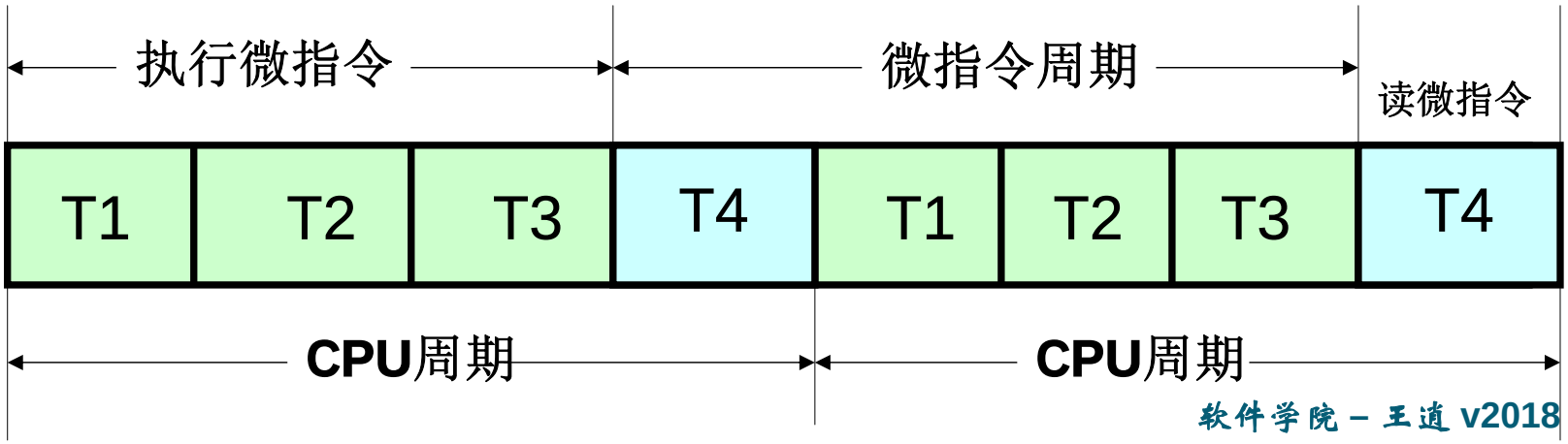
5.4.1 微程序控制器原理

- 执行"10进制加法"指令的微程序：
  - 从微地址=0000开始执行；
    - 执行0000中的取指微指令；
    - 判别P1，执行1010中的微指令
      R1+R2→R2；
    - 执行1001中的微指令
      R2+R3→R2；
    - 判别P2：
    - Cy=0，执行0001中的微指令；
  - 进入下一个CPU周期，
    从微地址=0000开始执行；



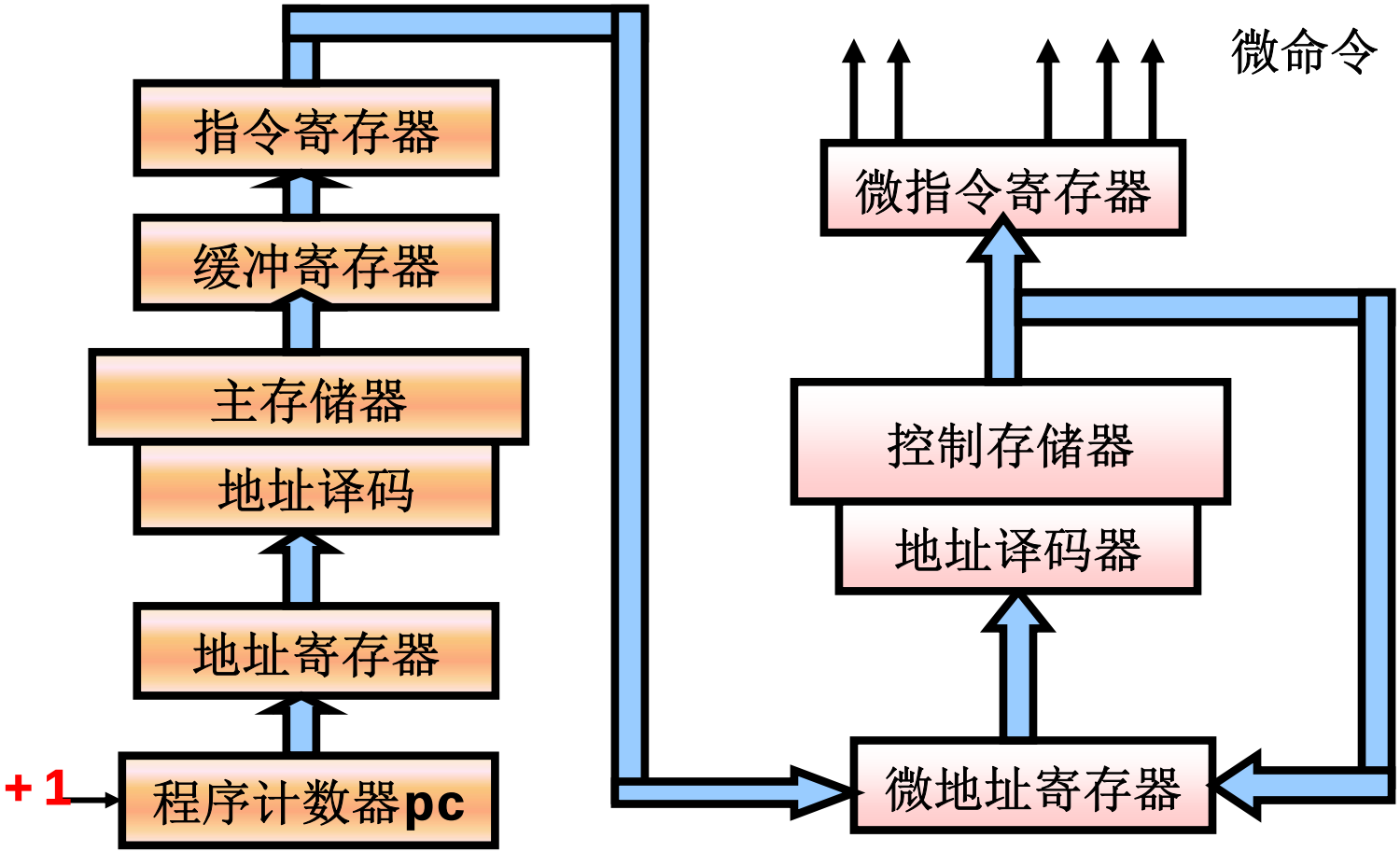
5.4.1 微程序控制器原理

- CPU周期与微指令周期
  - CPU周期： CPU读取一个指令字的最短时间(存取周期)来规定CPU周期；
  - 微指令周期：读微指令的时间与执行微指令时间之和。
  - 一般微指令周期=CPU周期；时间相等，不是概念相等；
    - 在T4节拍脉冲期间（200ns）:取微指令；在T1的上升沿：微指令已经取好，在T1—T3期间（600ns）：执行微操作。所以：在一个CPU周期内（800ns）：取址指令：占1/4CPU周期；执行微指令：占3/4CPU周期。



5.4.1 微程序控制器原理

- 机器指令与微指令



5.4.1 微程序控制器原理

- 机器指令与微指令

1.一条机器指令对应一个微程序，这个微程序是由若干条微指令序列组成的。因此，一条机器指令的功能是由若干条微指令组成的序列来实现的。简言之，一条机器指令所完成的操作划分成若干条微指令来完成，由微指令进行解释和执行。

2.从指令与微指令，程序与微程序，地址与微地址的一一对应关系来看，前者与内存储器有关，后者与控制存储器有关。

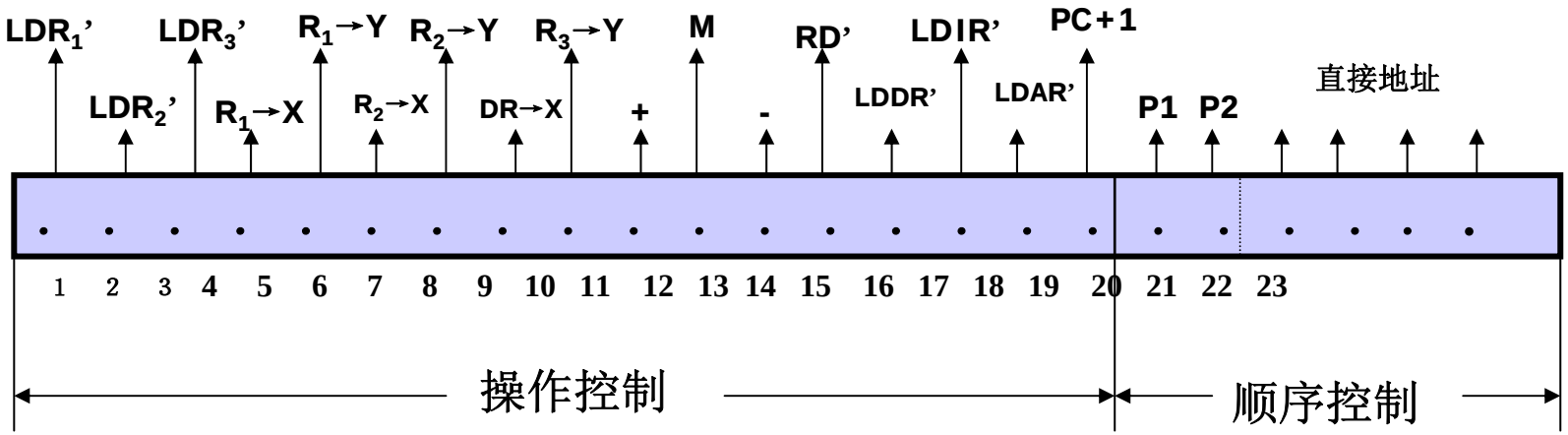
3. 每一个CPU周期对应一条微指令。

5.4.2 微程序设计技术

- 设计微指令结构应当追求的目标
  - 有利于缩短微指令的长度
  - 有利于缩小CM的容量
  - 有利于提高微程序的执行速度
  - 有利于对微指令的修改
  - 有利于提高微程序设计的灵活性

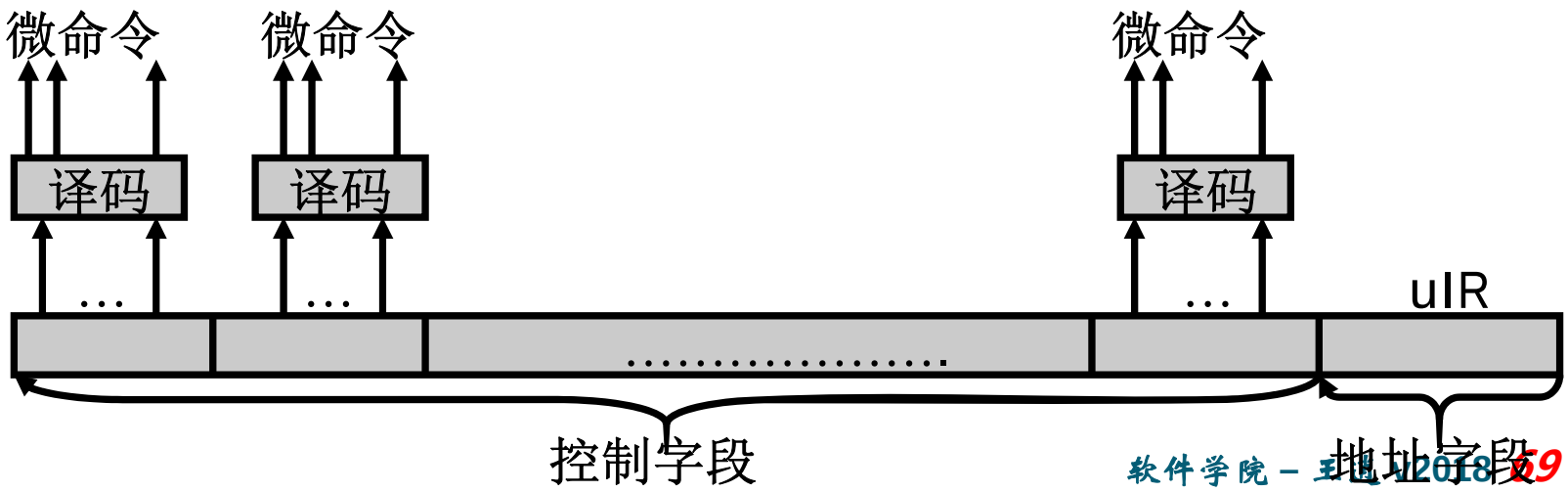
5.4.2 微程序设计技术

- 微指令编码：如何表示某条微指令要实现的操作？
- 直接表示法、编码表示法、混合表示法
  - 直接表示法：在微指令的操作控制字段中每一个微命令都用一位信息表示，对应于一种微操作。设计微指令时，选用或不选用某个微命令，只要将表示该微命令的相应位设置成“1”或“0”就可以了。因此，微命令的产生不必经过译码，所需的控制信号直接送到相应的控制点。
  - 特点：直观、不必译码、速度快，微指令的长度太长，不好记忆，占用空间大。



5.4.2 微程序设计技术

- 微指令编码：
  - 编码表示法：将微指令的控制字段分为若干个小字段，每个字段分别编码，每种编码代表一种微命令。把一组相斥性的微命令信号组成一个小组(即一个字段)，然后通过小组(字段）译码器对每一个微命令信号进行译码，译码输出作为操作控制信号，其微指令结构如下图所示。
  - 特点：采用字段译码的编码方法，可以用较小的二进制信息位表示较多的微命令信号。3位二进位译码后可表示7个微命令，4位二进制位译码后可表示15个微命令(字段全零无意义)。但由于增加译码电路，使微程序的执行速度稍稍减慢。



5.4.2 微程序设计技术

- 微指令编码：
  - 混合表示法：把直接表示法与编码表示法混合使用，以便能综合考虑指令字长、灵活性、执行微程序速度等方面的要求。另外，在微指令中还可附设一个常数字段。该常数可作为操作数送入ALU运算，也可作为计数器初值用来控制微程序循环次数。

- 例：某机器指令系统总共需要240个微命令，采用直接编码法，微指令的操作控制字段需240位，采用编码表示法，如将控制字段分成4位一段，共16段，每个字段经一个译码器输出，可获得15个微命令，总共16段就可获得240个微命令。微指令的操作控制字段仅64位。

5.4.2 微程序设计技术

- 微地址形成方法：
- 微程序入口地址：每条机器指令对应一段微程序，当公用的取指微程序从主存中取出机器指令之后，由机器指令的操作码字段指出各段微程序的入口地址，这是一种多分支(或多路转移)的情况。
  - 根据机器指令操作码找到相应机器指令所对应的微程序的入口地址以及后续微指令地址。
  - 计数器方式
  - 多路转移方式

5.4.2 微程序设计技术

- 微地址形成方法：
  - 计数器方式：设置一个微程序计数器μPC，在顺序执行微指令时，后继微指令地址由现行微地址加上一个增量来实现。遇到转移时，由微指令给出转移微地址。
  - 方法：
    - 微程序顺序执行时，其后继微地址就是现行微地址加上一个增量(通常为1)；
    - 当微程序遇到转移或转子程序时，由微指令的转移地址段来形成转移微地址。
    - 一般情况下都是将微地址寄存器μMAR作为μPC。
  - 特点：
    - 优点是简单、易于掌握，编制微程序容易
    - 缺点是这种方式不能实现两路以上的并行微程序转移，因而不利于提高微程序的执行速度。

5.4.2 微程序设计技术

- 微地址形成方法：
  - 多路转移方式：一条微指令具有多个转移分支的能力称为多路转移。
- 在多路转移方式中：
  - 当微程序不产生分支时，后继微地址直接由微指令的顺序控制字段给出；
  - 当微程序出现分支时，有若干“后选”微地址可供选择：即按顺序控制字段的“判别测试”标志P1和“状态条件”信息P2来选择其中一个微地址。“状态条件”有n位标志，可实现微程序2^n路转移，涉及微地址寄存器的n位 。
  - P1：用来确定微程序的入口；
  - P2：实现微程序的转移结构；
- 多路转移方式的特点：
  - 能以较短的顺序控制字段配合，实现多路并行转移，灵活性好，速度较快，但转移地址逻辑需要用组合逻辑方法设计。

- 例：微地址寄存器有6位(μA5-μA0)，当需要修改其内容时，可通过某一位触发器的强置端S将其置“1”。现有三种情况：
  (1)执行“取指”微指令，微程序按IR的OP字段(IR3-IR0)进行16路分支；
  (2)执行条件转移指令微程序，按进位标志C的状态进行2路分支；
  (3)执行控制台指令微程序时，按IR4，IR5的状态进行4路分支。
  - 请按多路转移方法设计微地址转移逻辑。
- 解：按所给设计条件，微程序有三种判别测试，分别为P1，P2，P3。 由于修改μA5-μA0内容具有很大灵活性，现分配如下：
  (1)用P1和IR3-IR0修改μA3-μA0；
  (2)用P2和C修改μA0；
  (3)用P3和IR5，IR4修改μA5， μA4。
  - 另外还要考虑时间因素T4(假设CPU周期最后一个节拍脉冲)，故转移逻辑表达式如下：

$\mu A5 = P3 \cdot IR5 \cdot T4$　　$\mu A4 = P3 \cdot IR4 \cdot T4$

$\mu A3 = P1 \cdot IR3 \cdot T4$　　$\mu A2 = P1 \cdot IR2 \cdot T4$

$\mu A1 = P1 \cdot IR1 \cdot T4$　　$\mu A0 = P1 \cdot IR0 \cdot T4 + P2 \cdot C \cdot T4$

  - 由于从触发器强置端修改，故前5个表达式可用“与非”门实现，最后一个用“与或非”门实现。

5.4.2 微程序设计技术

- 微指令格式：水平型微指令和垂直型微指令
  - 水平型微指令：一次能定义并执行多个并行操作微命令的微指令，叫做水平型微指令。
    - 全水平型(不译法)伪指令，字段译码法水平伪指令，直接和译码相混合水平型微指令。
  - 基本特征：
    - 微指令字较长；
    - 一条微指令能控制数据通路中多个功能部件并行操作；
    - 微命令的编码简单，尽可能使微命令与控制门之间具有直接对应关系；

控 制 字 段	判别测试字段	直接地址字段

5.4.2 微程序设计技术

- 微指令格式：水平型微指令和垂直型微指令
  - 垂直型微指令：在微指令中设置微操作码字段，采用为操作码编译法，由操作码规定为微指令的功能，在一条微指令中只有一、两个微操作命令称为垂直型微指令。
  - 基本特征：
    - 微指令字短。
    - 微指令的并行操作能力有限，一条微指令只能控制数据通路中的一、两个信息传送。
    - 微指令编码比较复杂，全部微命令组成一个微操作码字段，经过完全译码，微指令的各个二进制位与数据通路的各个控制点之间完全不存在直接对应关系。

5.4.2 微程序设计技术

- 垂直型微指令：设微指令字长为16位，微操作码3位。
  - ①寄存器—寄存器传送型微指令：把源寄存器数据送目标寄存器。13-15位为微操作码，源寄存器和目标寄存器编址各5位，可指定31个寄存器。

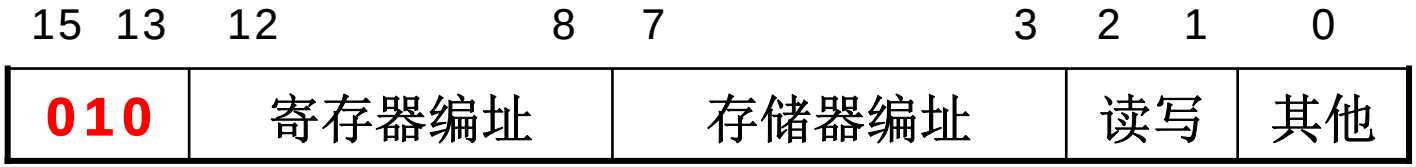
15 13	12 8	7 3	2 0
000	源寄存器编址	目的寄存器编址	其他

  - ②运算控制型微指令：其功能是选择ALU的左、右两输入源信息，按ALU字段所指定的运算功能(8种操作)进行处理，并将结果送入暂存器中。左、右输入源编址可指定31种信息源之一。

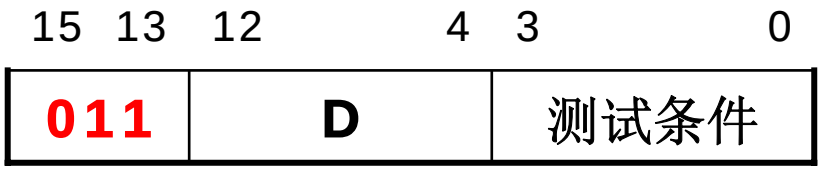
15 13	12 8	7 3	2 0
001	左输入源编址	右输入源编址	ALU

5.4.2 微程序设计技术

- 垂直型微指令：设微指令字长为16位，微操作码3位。
  - ③访问主存微指令：功能是将主存中一个单元的信息送入寄存器或将寄存器的数据送往主存。存储器编址是指按规定的寻址方式进行编址。



  - ④条件转移微指令：功能是根据测试对象的状态决定是转移到D所指定的微地址单元，还是顺序执行下一条微指令。



5.4.2 微程序设计技术

- 水平型与垂直型微指令比较：

(1)水平型微指令并行操作能力强，效率高，灵活性强，垂直型微指令则较差。

(2)水平型微指令执行一条指令的时间短，垂直型微指令执行时间长。

(3)由水平型微指令解释指令的微程序，有微指令字较长而微程序短的特点。垂直型微指令则相反。

(4)水平型微指令用户难以掌握，而垂直型微指令与指令比较相似，相对来说，比较容易掌握。

5.4.2 微程序设计技术

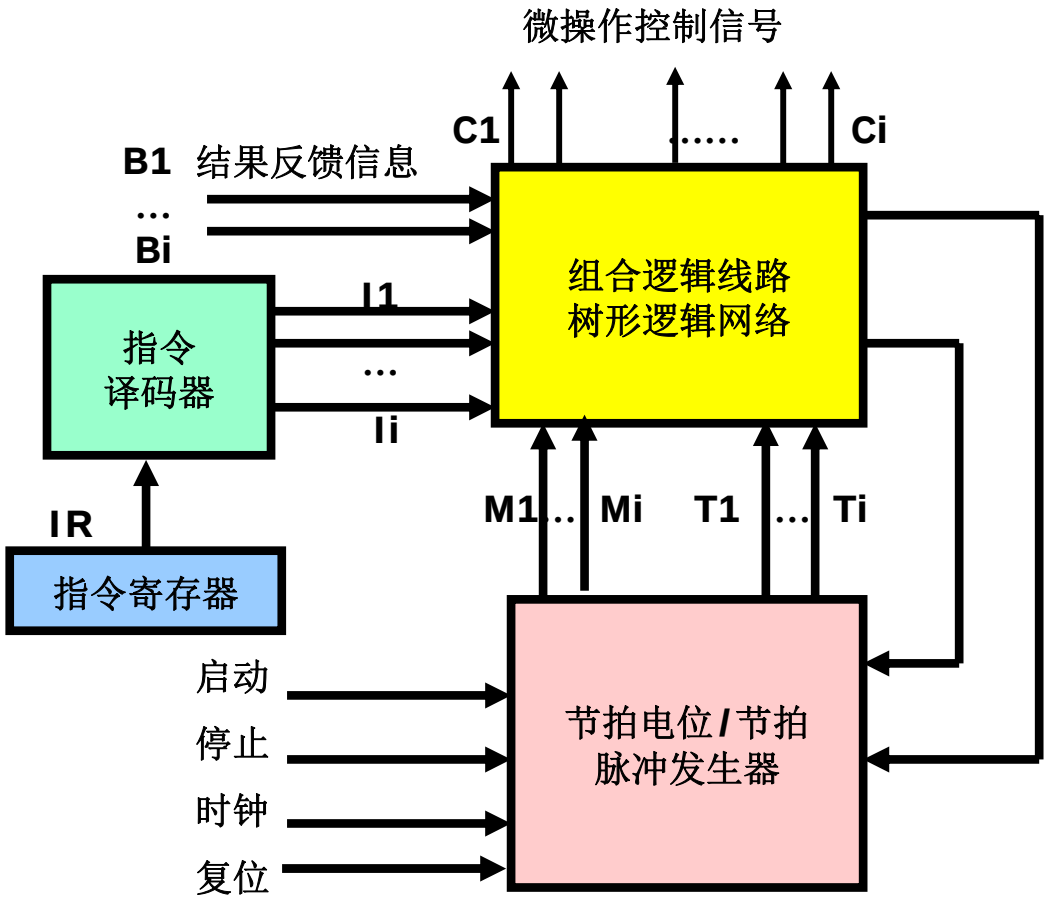
- 动态微程序设计：
  - 对应于一台计算机的机器指令只有一组微程序，这一组微程序设计好之后，一般无须改变而且也不好改变，这种微程序设计技术称为静态微程序设计。
  - 采用EPROM作为控制存储器，可以通过改变微指令和微程序来改变机器的指令系统，这种微程序设计技术称为动态微程序设计。

5.5 硬连线 / 布线控制器

- 基本思想：
  - 把控制部件看成作为生产专门固定时序控制信号的逻辑电路，而此逻辑电路以使用最少元件和取得最高操作速度为设计目标。
  - 这种逻辑电路是一种由门电路和触发器构成的复杂树形网络，故称为硬布线控制器。由于门电路多是组合逻辑电路所以也称为组合逻辑控制器。
  - 硬布线控制的优点：速度较快；缺点：不容易修改添加新功能
  - 微程序控制的优点：具有规整性、灵活性、可维护性等；缺点：采用存储程序原理，需要执行多条微指令，速度较慢

5.5 硬连线/布线控制器

- 硬布线控制器的结构图：



5.5 硬连线/布线控制器

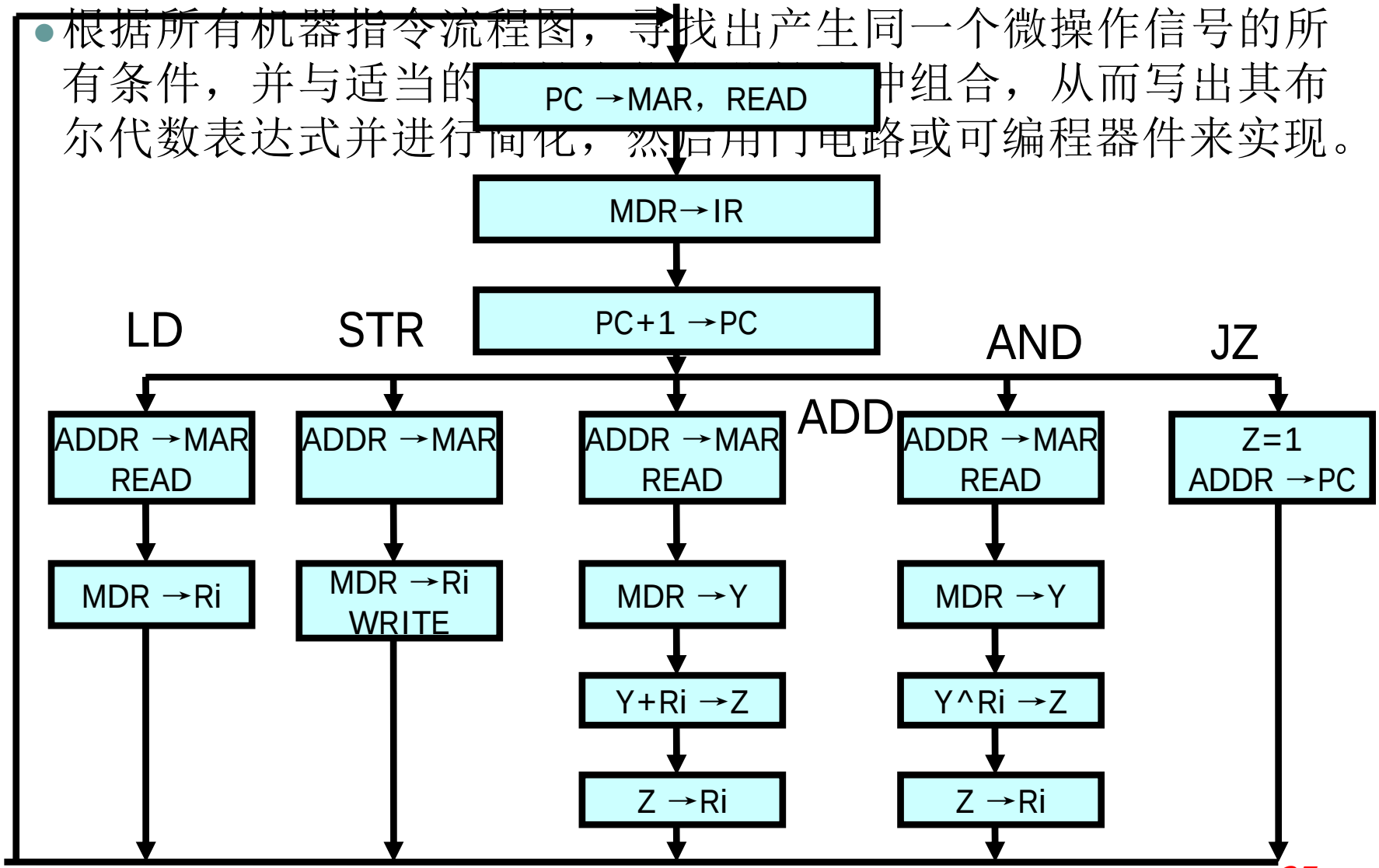
- 硬布线控制器输入信号：
  - 来自指令操码译码器的输出I_m；
  - 来自执行部件的反馈信息B_j；
  - 来自时序产生器的时序信号，包括节拍电位信号M和节拍脉冲信号T。
- 硬布线控制器输出信号：
  - 微操作控制信号，它用来对执行部件进行控制。
- 硬布线控制器基本原理：
  - $C=f(I_m,M_i,T_k,B_j)$
  - 实现：每一个操作控制信号与指令，时序，条件都有一个逻辑关系，用逻辑表达式描述。

5.5 硬连线/布线控制器

- 硬布线控制器输入信号：
  - 来自指令操码译码器的输出I_m；
  - 来自执行部件的反馈信息B_j；
  - 来自时序产生器的时序信号，包括节拍电位信号M和节拍脉冲信号T。
- 硬布线控制器输出信号：
  - 微操作控制信号，它用来对执行部件进行控制。
- 硬布线控制器基本原理：
  - $C=f(I_m,M_i,T_k,B_j)$
  - 实现：每一个操作控制信号与指令，时序，条件都有一个逻辑关系，用逻辑表达式描述。

5.5 硬连线 / 布线控制器

- 硬布线控制器设计：
  - 根据所有机器指令流程图，寻找出产生同一个微操作信号的所有条件，并与适当的[illegible]种组合，从而写出其布尔代数表达式并进行简化，然后用门电路或可编程器件来实现。



5.6 流水CPU

- 追求计算机有很高的处理速率，促使计算机性能提高的因素除了提高器件性能外还有哪些方式呢？？
  - 硬件工艺上的提升
  - 并行工作
    - 双端口访问和多模块交叉
    - 空间并行和时间并行
  - 采用分层的存储系统
    - Cache
    - 虚拟存储系统
- 流水线技术：把一个重复的过程分解为若干个子过程，每个子程序可以与其他子过程同时进行。

5.6.1 并行处理技术

- 并行性(Parallelism)：在同一时刻或是同一时间间隔内完成两种或两种以上性质相同或不相同的工作。
  - 同时性(Simultaneity)：同一时刻发生的并行性
  - 并发性(Concurrency)：同一个时间间隔内发生的并行性
- 3中并行实现：

1.时间并行

- 时间并行指时间重叠，在并行性概念中引入时间因素，让多个处理过程在时间上相互错开，轮流重叠地使用同一套硬件设备的各个部分，以加快硬件周转而赢得速度。
- 实现方式：流水技术；

2.空间并行

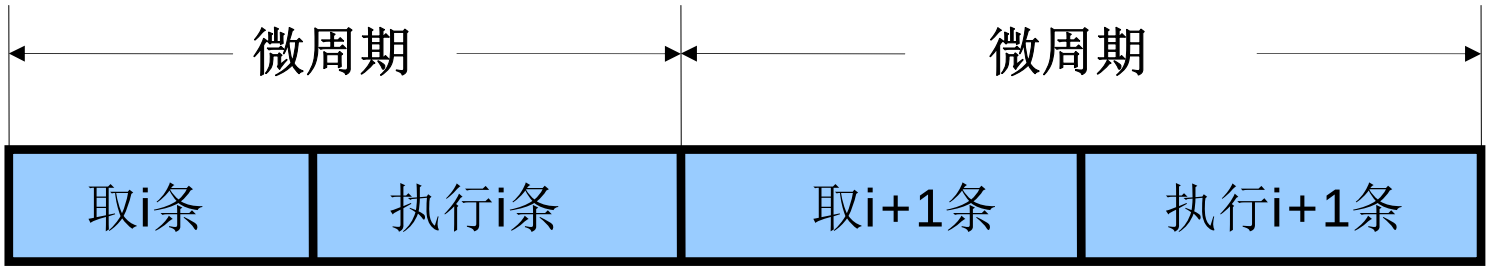
- 空间并行指资源重复，在并行性概念中引入空间因素，以“数量取胜”为原则来大幅度提高计算机的处理速度。
- 实现方式：多处理器，多计算机系统

3.时间并行+空间并行

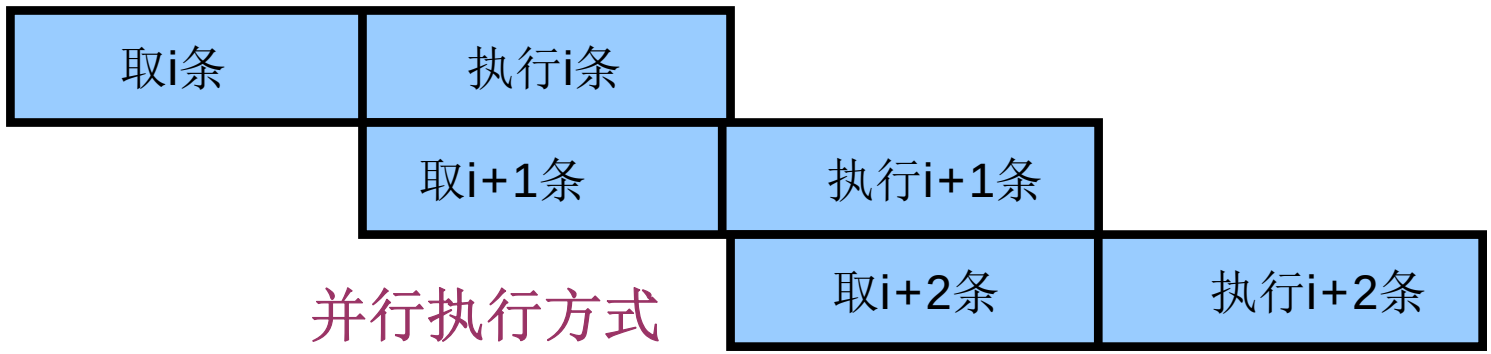
- 指时间重叠和资源重复的综合应用，既采用时间并行性又采用空间并行性。显然，第三种并行技术带来的高速效益是最好的。
- Pentium中采用了超标量流水技术。

5.6.1 并行处理技术

- 并行性应用：
  - 指令内部并行：微操作之间 / 指令级并行（ILP：Instruction Level Parallel）/ 线程级并行（TLP：Thread Level Parallel ）/ 程序级并行 /系统级并行：分布式系统、多机系统、机群系统



串行执行方式



并行执行方式

5.6.2 流水CPU的结构

- 流水CPU的系统组成：
  - 存储器体系：

    主存采用多体交叉存储器；

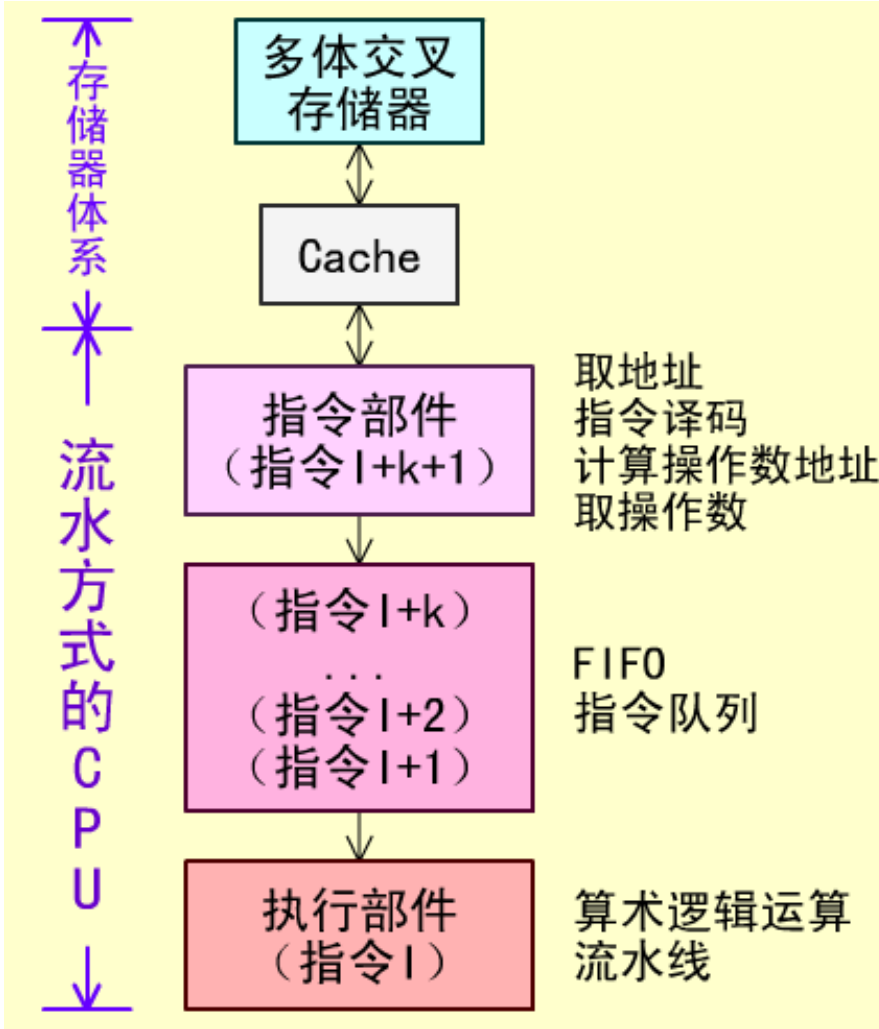
    Cache
  - 流水方式CPU：

    指令部件、指令队列、执行部件
    - 指令流水线
    - 指令队列：FIFO
    - 执行部件：

      可以由多个采用流水线方式构成的算术逻辑部件构成，可以将定点运算部件和浮点运算部件分开。
  - 3级流水结构：

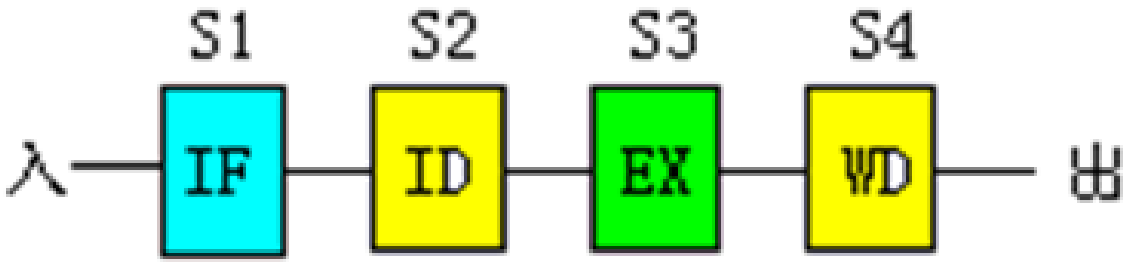
    多体交叉存储器(存储器流水)

    取指流水(CPU流水)，执行流水(ALU流水)



5.6.2 流水CPU的结构

- 流水CPU的时空图：描述流水线的工作，最常用的方法是时间-空间图（时空图）
  - 横坐标：表示时间，即各个任务在流水线中所经过的时间；
  - 纵坐标：表示空间，即流水线的各个子过程，也称为级、段、流水线深度(Stage)
  - 4级：
    - IF（Instruction Fetch取指）
    - ID（Instruction Decode指令译码）
    - EX（Execution执行）
    - WB（Write Back写回）



5.6.2 流水CPU的结构

- 非流水CPU的时空图：

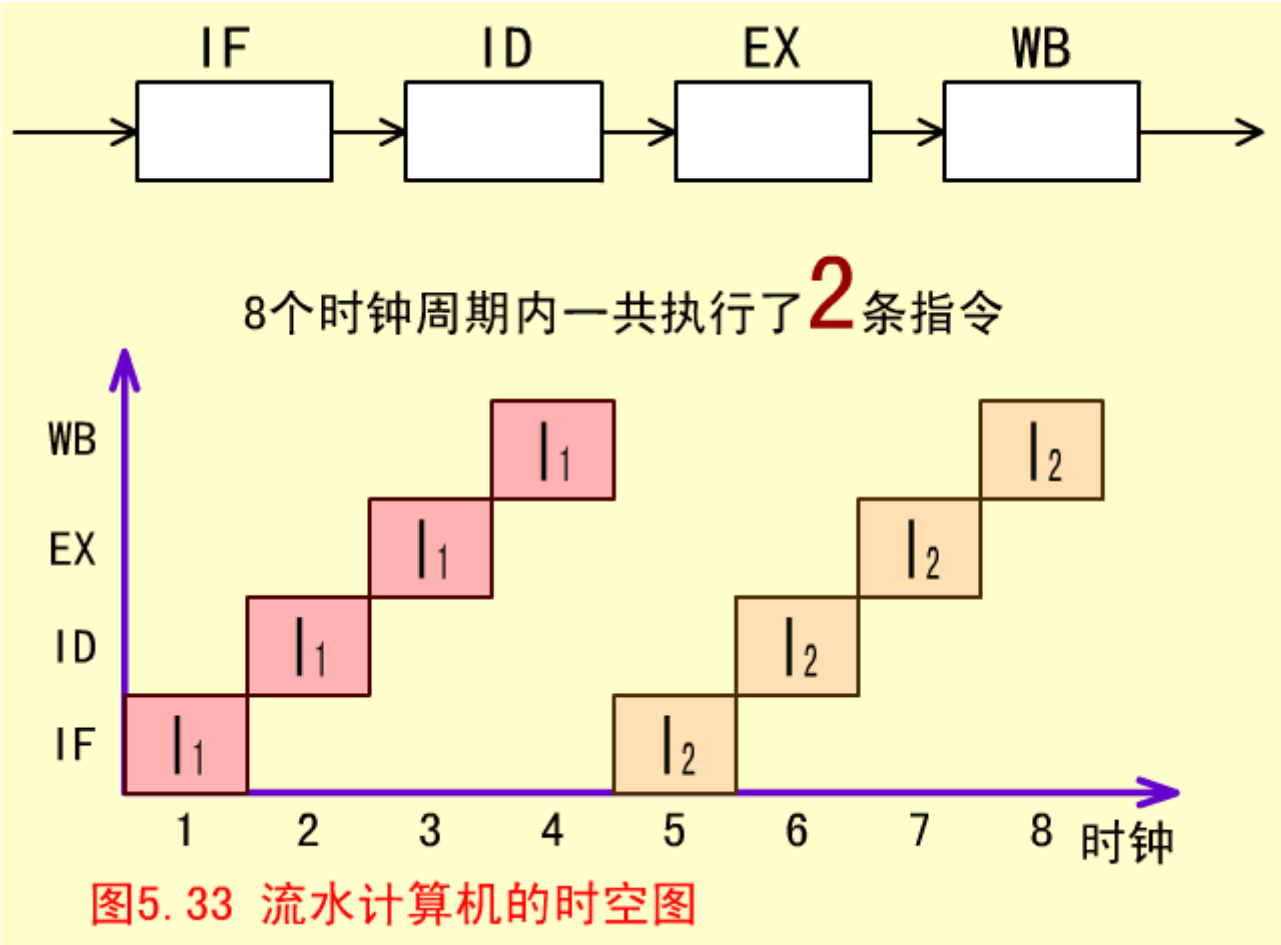


图5.33 流水计算机的时空图

5.6.2 流水CPU的结构

- 标量流水CPU的时空图：

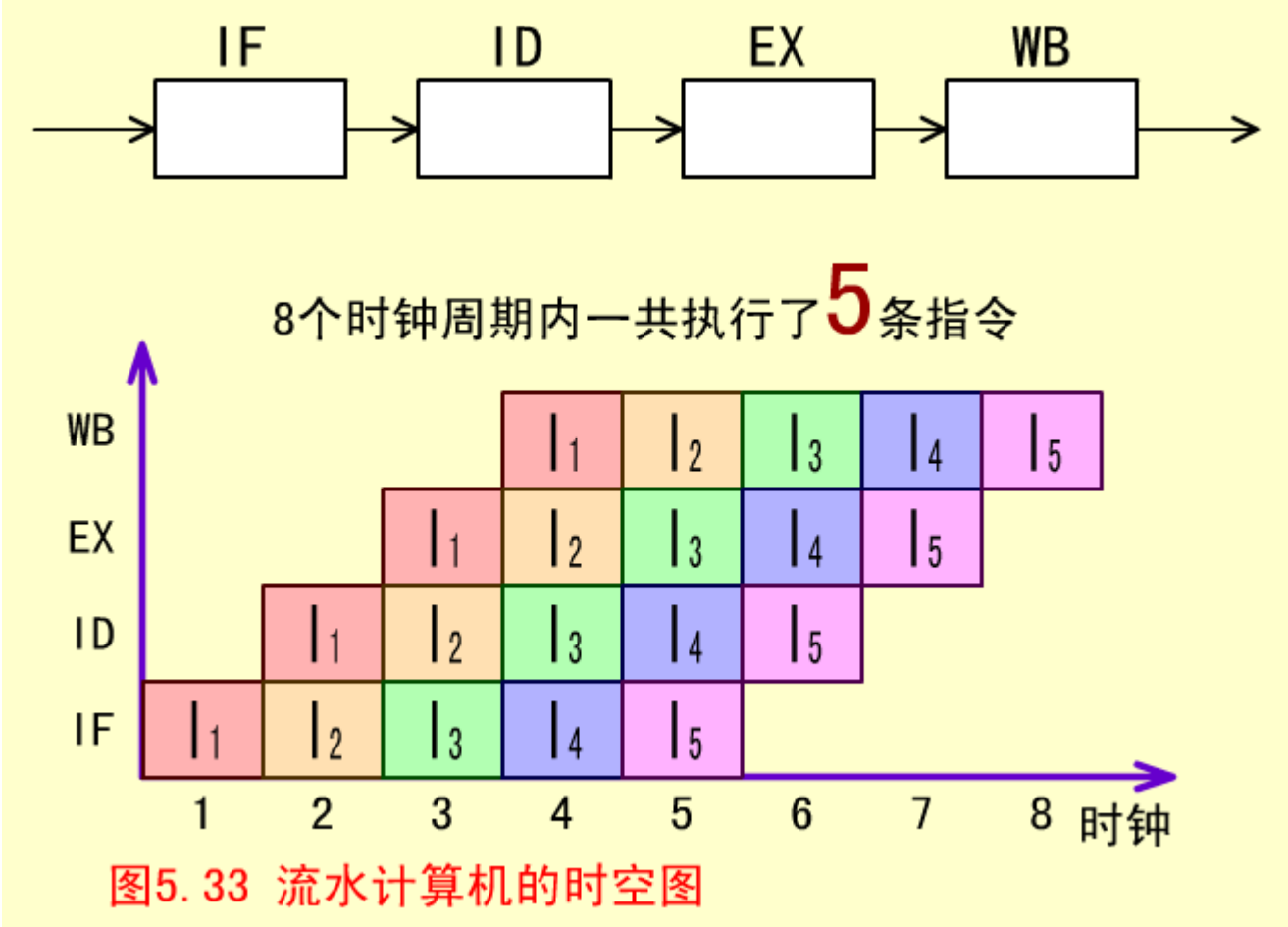


图5.33 流水计算机的时空图

5.6.2 流水CPU的结构

- 超标量流水CPU的时空图：
- 超标量流水
  - 具有两条以上的指令流水线，上图中流水线满载时，每一个时钟周期可以执行2条指令，采用时间和空间并行技术
  - PS. 此时钟周期不同于彼“时钟周期”：一个是流水理论中的概念
  - 一个是指令周期、CPU周期中的概念

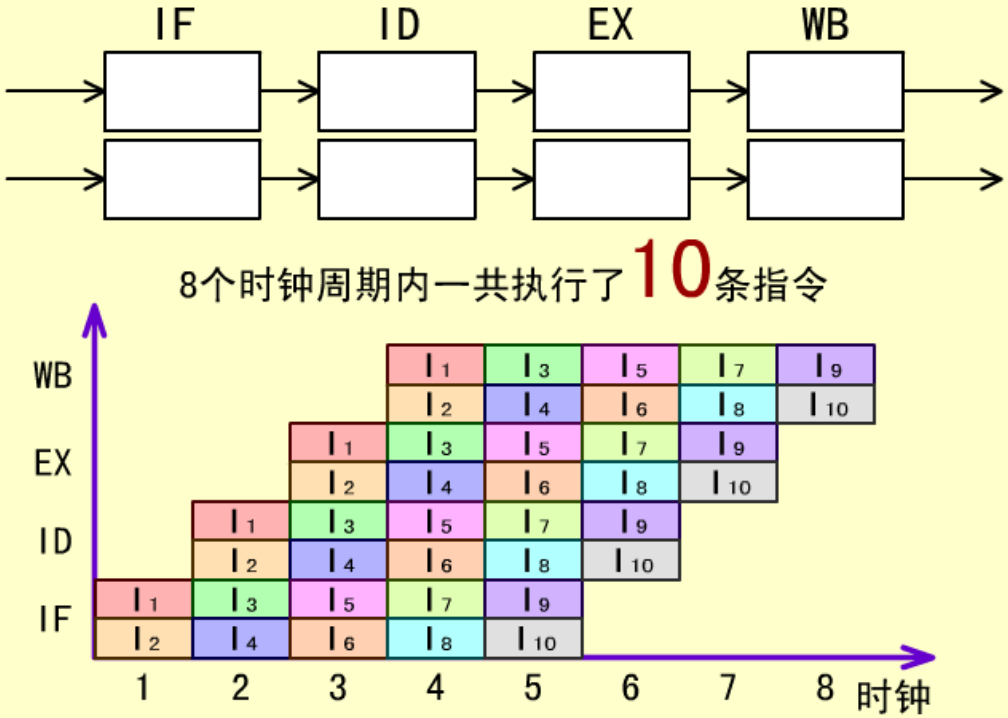


图5.33 流水计算机的时空图

5.6.2 流水CPU的结构

- 流水线分类：按种类分为
  - 指令流水线：指令步骤并行
  - 算术流水线：运算步骤并行
  - 存储流水线：多体交叉并行访存
  - 处理机流水线（宏流水线）：处理机级联
- 流水线特点：
  - 流水线实际上是把一个功能部件分解成多个独立的子功能部件（一个任务也就分成了几个子任务，每个子任务由一个子功能部件完成），并依靠多个子功能部件并行工作来缩短所有任务的执行时间
  - 流水线有助于提高整个程序（所有任务）的吞吐率，但并没有减少每个指令（任务）的执行时间
  - 流水线各个功能段所需时间应尽量相等。否则，时间长的功能段将成为流水线的“瓶颈”，会造成流水线的**“阻塞”（Stall）**
  - 流水线开始需要**“通过时间” (Fill)**和最后需要**“排空时间”(Drain)**。流水线只有处理连续不断的任务才能发挥其效率

5.6.3 流水线中主要问题

- 问题：流水线中存在一些相关(冲突、冒险Hazard，相关、依赖Dependence，竞争Competition)的情况，它使得下一条指令无法在设计的时钟周期内执行。这些相关将降低流水线性能。
- 断流问题：因为出现了相关冲突
  - 主要有三种类型的相关（冲突）
    - 结构相关（资源冲突）：当指令重叠执行过程中，硬件资源满足不了指令重叠执行的要求
    - 数据相关（数据冲突）：在同时执行的多条指令中，一条指令依赖前一条指令的执行结果(数据)却无法得到
    - 控制相关（控制冲突）：流水线遇到分支指令或其他改变PC值的指令

5.6.3 流水线中主要问题

- 资源相关：多条指令进入流水线后，在同一机器时钟周期内争用同一个功能部件所发生的冲突。
  - 解决办法：后边指令拖一拍再推进；或增设一个存储部件，指令数据分存。
  - 例：假定一条指令流水线由五段组成，且仅有IF过程和MEM过程需要访问存储器
    - I1与I4两条指令在时钟4争用存储器资源的相关冲突

时钟 / 指令	1	2	3	4	5	6	7	8
I1 (Load)	IF	ID	EX	MEM	WB			
I2		IF	ID	EX	MEM	WB		
I3			IF	ID	EX	MEM	WB	
I4				IF	ID	EX	MEM	WB
I5					IF	ID	EX	MEM

5.6.3 流水线中主要问题

- 数据相关：
  - **RAW**(Read After Write)：后面指令用到前面指令所写的数据。
  - **WAW**(Write After Write)：两条指令写同一个单元，在简单流水线中没有此类相关，因为不会乱序执行。
  - **WAR**(Write After Read)：后面指令覆盖前面指令所读的单元，在简单流水线中没有此类相关。
  - 解决办法：
    - 可以推后后继指令对相关单元的读操作
    - 设置相关的直接通路（Forwarding）

5.6.3 流水线中主要问题

- 例：数据相关：
  - ADD R1, R2, R3 ;R2+R3-->R1
  - SUB R4, R1, R5 ;R1-R5-->R4
  - AND R6, R1, R7 ;R1^R7-->R6
  - 两条指令ADD/SUB发生数据相关冲突RAW(Read After Write)：ADD结果保存到R1，SUB再读R1。AND也存在同一问题。

指令 \ 时钟	1	2	3	4	5	6	7	8
ADD	IF	ID	EX	MEM	WB			
SUB		IF	ID	EX	MEM	WB		
AND			IF	ID	EX	MEM	WB	

- 例：数据相关：
  - 流水线中有三类数据相关冲突：写后读（RAW）相关；读后写（WAR）相关；写后写（WAW）相关。判断以下三组指令各存在哪种类型的数据相关：

```
I1    ADD R1，R2，R3        ;(R2)+(R3)→R1
I2    SUB R4，R1，R5        ;(R1)-(R5) → R4

I3    STO M（x），R3        ;(R3) → M(x)，M(x)是存储器单元
I4    ADD R3，R4，R5        ;(R4)+(R5) → R3

I5    MUL R3，R1，R2        ;(R1)×(R2) → R3
I6    ADD R3，R4，R5        ;(R4)+(R5) → R3
```

- 解：
  - I1指令运算结果应先写入R1，然后在I2指令中读出R1内容。由于I2指令进入流水线，变成I2指令在I1指令写入R1前就读出R1内容，发生**RAW**相关。
  - I3指令应先读出R3内容并存入存储单元M（x），然后在I4指令中将运算结果写入R3。但由于I4指令进入流水线，变成I4指令在I3指令读出R3内容前就写入R3，发生**WAR**相关。
  - 如果I6指令的加法运算完成时间早于I5指令的乘法运算时间，变成指令I6在指令I5写入R3前就写入R3，导致R3的内容错误，发生**WAW**相关。

5.6.3 流水线中主要问题

- 控制相关：
  - 引起原因：转移指令
    - 当前指令有跳转，但流水已经开启后续指令处理过程。
  - 解决办法：
    - 延迟转移法，让跳转的指令接在最后流水入口
    - 转移预测法，用硬件预测将来的行为，提前让转移指令进流水。

5.6.4 Pentium CPU

- Pentium CPU （第一代）
  - 1989年初0.8um工艺，310万晶体管
  - 有60M和66MHz外频两种版本
  - 5V电压，功耗20W
  - 超标量流水线结构
    - 486有一条流水线
    - Pentium有U和V两条指令流水线
      - U流水线可以执行所有的整数和浮点指令
      - V流水线可以执行简单的整数和FXCH浮点指令
  - 双重分离式Cache，减少了等待和搬移数据时间
  - 32位CPU，外部数据总线宽度为64位，外部地址总线宽度为36位
  - 非固定长度指令格式，9种寻址方式，191条指令，兼具有RISC和CISC特性，不过我们还是将其看成CISC
  - SL电源管理技术
  - 提供了更加灵活的存储器寻址结构，可以支持传统的4k大小的页面，也可以支持4M大小的页面
  - 动态转移预测技术

5.6.4 Pentium CPU

- 80486 CPU 5级指令流水线，每级1个时钟周期

① PF——指令预取（prefetch）

② D1——指令译码1（decode stage 1）

对所有操作码和寻址方式信息进行译码

③ D2——指令译码2（decode stage 2）

将操作码扩展为ALU的控制信号，存储器地址计算

④EX——指令执行（execute）

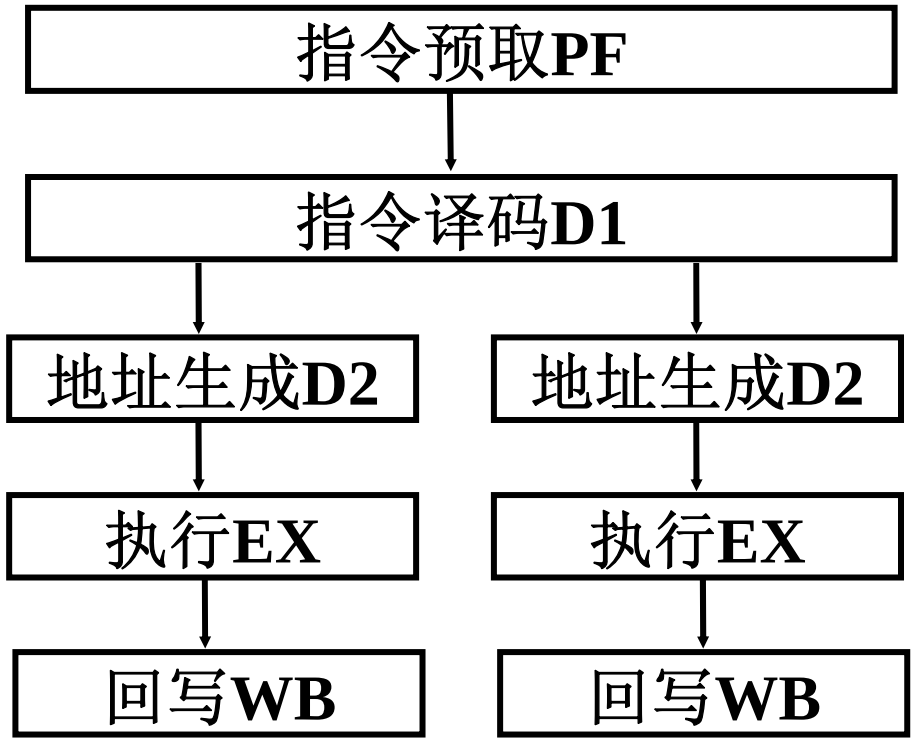
完成ALU操作和Cache存取

⑤WB——回写（write back）

更新在EX步骤得到的寄存器数据和状态标志

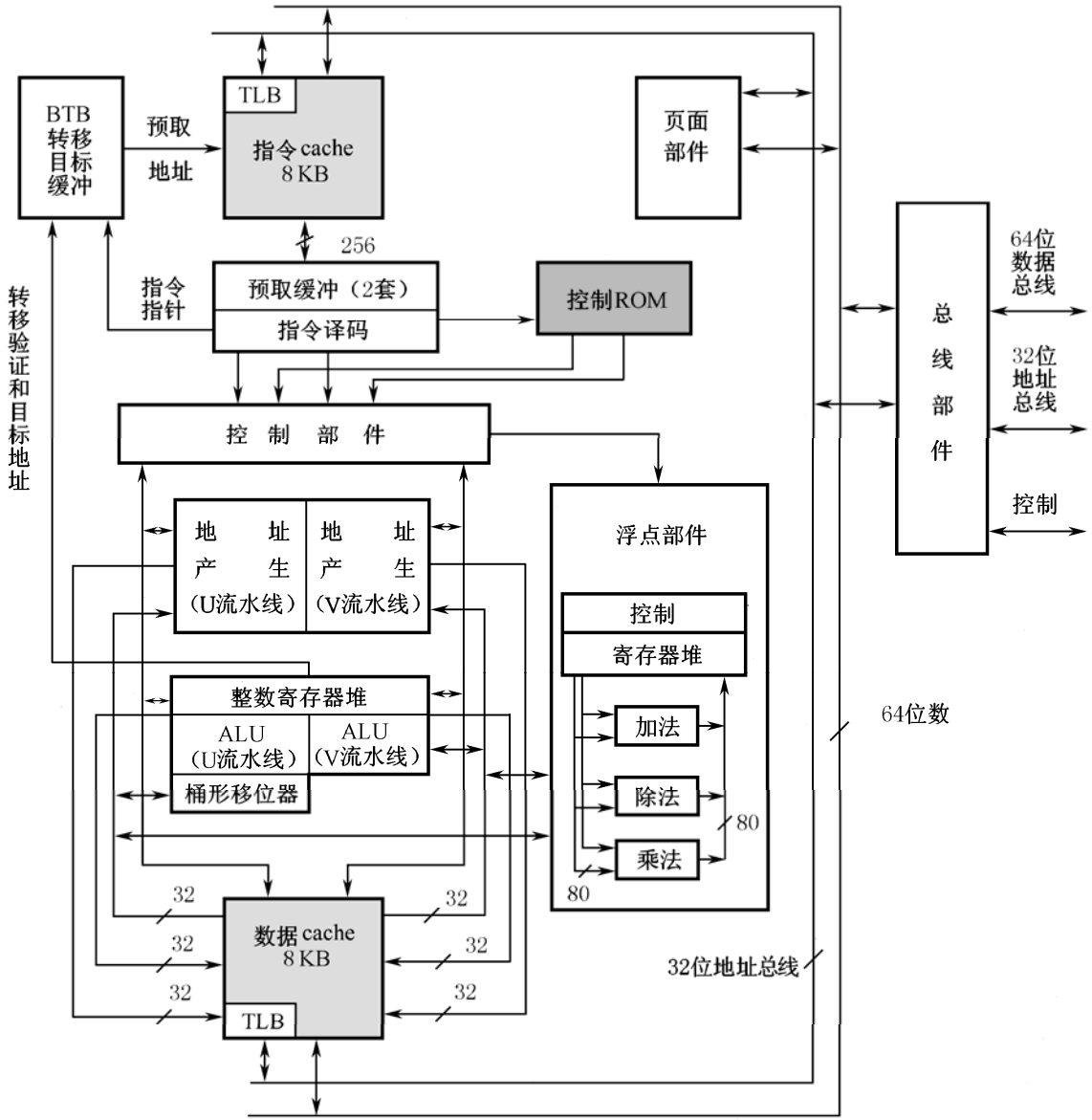
5.6.4 Pentium CPU

- Pentium的超标量流水线
  - 类似80486的5级流水线，后3级可以在两个流水线同时进行
  - 指令预取PF和指令译码D1步骤可以并行取出、译码2条简单指令，然后分别发向U和V流水线
  - 在满足指令配对的条件下，Pentium可以每个时钟周期执行完2条指令



5.6.4 Pentium CPU

- Pentium结构图
  - MESI（Modified
  - BTB（Branch Targ
  - TLB（Translation



5.7 RISC CPU

- **RISC的三个要素**
  - 一个有限的简单的指令集
  - CPU配备大量的通用寄存器
  - 强调对指令流水线的优化

5.7.1 RISC机器的特点

- 特点
  - 等长指令，典型长度是4个字节(32位）
  - 寻址方式少且简单，一般为2～3种
  - 只有取数指令和存数指令访问存储器
  - 指令数目一般少于100种，指令格式一般少于4种
  - 指令功能简单，控制器多采用硬布线方式
  - 指令的执行时间为一个处理时钟周期
  - 整数寄存器的个数不少于32个
  - 强调通用寄存器资源的优化使用
  - 支持指令流水并强调指令流水的优化使用
  - RISC技术的编译程序复杂

5.7.1 RISC机器的特点

- CISC与RISC比较

比较内容	CISC	RISC
指令系统	复杂，庞大	简单，精简
指令数目	一般大于200	一般小于100
指令格式	一般大于4	一般小于4
寻址方式	一般大于4	一般小于4
指令字长	不固定	等长
可访存指令	不加限制	只有LOAD/STORE指令
各种指令使用频率	相差很大	相差不大
各种指令执行时间	相差很大	绝大多数在一个周期内完成
优化编译实现	很难	较容易
程序源代码长度	较短	较长
控制器实现方式	绝大多数为微程序控制	主要采用硬布线控制
软件系统开发时间	较短	较长

5.7.2 RISC CPU的实例

- **MC88110**
  - 12个执行功能部件，3个Cache（指令，数据和目标指令），两个寄存器堆（通用寄存器堆、扩展寄存器堆），六条80位宽的内部总线

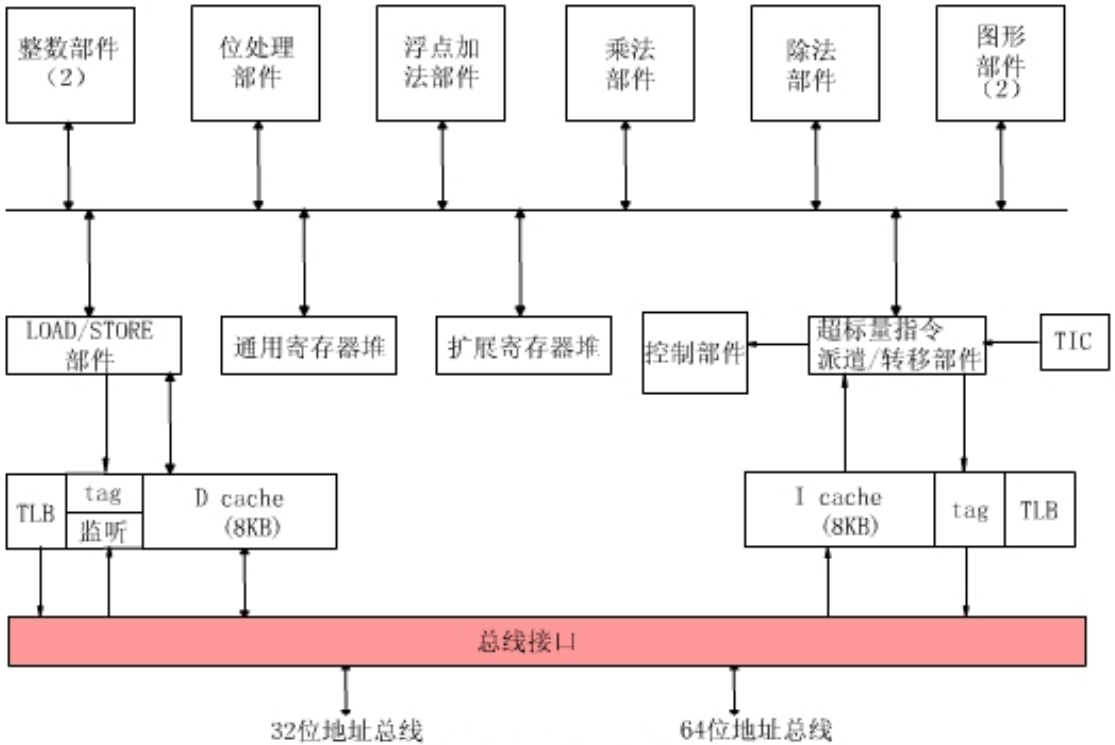
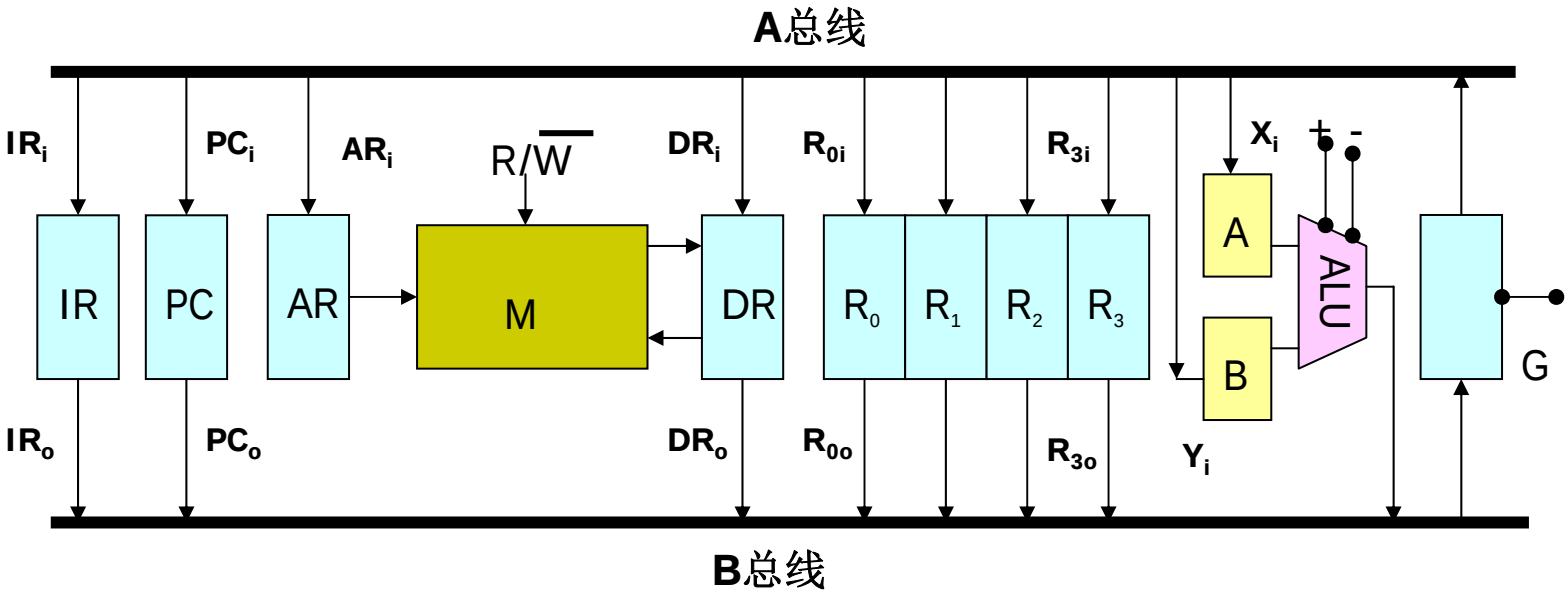


图5.41 88110处理器结构框图

5.7.2 RISC CPU的实例

- **MC88110**的指令流水线
  - 超标量流水线CPU
  - F&D：取指和译码段需要一个时钟周期，
  - EX：执行段，大都只需要一个时钟周期，
  - WB：写回段，只需要时钟周期的一半
  - 采用了直接通路（Forwarding）技术

- 例：画出指令周期流程图并列出相应的微操作控制信号序列。
  - **STA R1,(R2)** ；R1→(R2)
  - **LDA (R3),R0** ；(R2)→R0



- 例：画出指令周期流程图并列出相应的微操作控制信号序列。
  - **STA R1,(R2)** ；R1→(R2)
  - **LDA (R3),R0** ；(R2)→R0