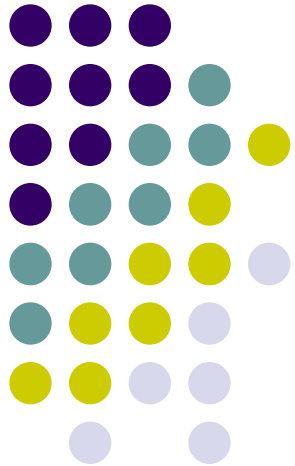
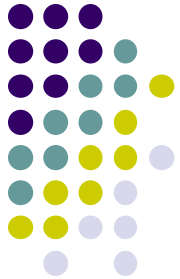
第五章　中央处理器



返回

第五章　中央处理器



5.1CPU功能和组成

5.2指令周期

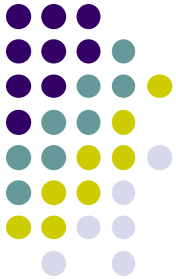
5.3时序产生器和控制方式

5.4微程序控制器

5.5硬连线控制器

5.6流水CPU

5.7RISC CPU



5.1 CPU的功能和组成

5.1.1 CPU的功能

5.1.2 CPU的基本组成

5.1.3 CPU中的主要寄存器

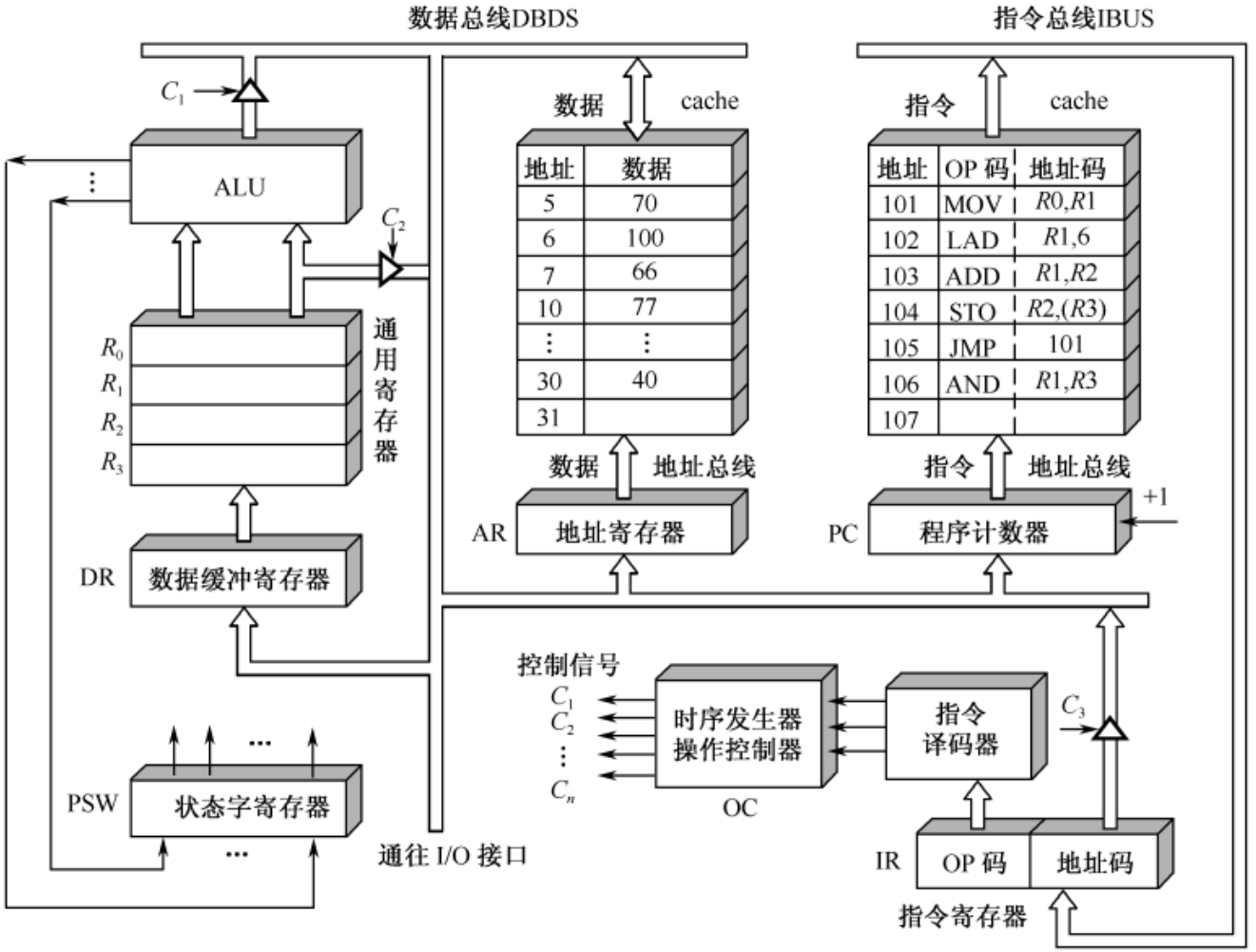
5.1.4 操作控制器与时序产生器

5.1 CPU的功能和组成

1、CPU的功能

- 指令控制（程序的顺序控制）
- 操作控制（一条指令有若干操作信号实现）
- 时间控制（指令各个操作实施时间的定时）
- 数据加工（算术运算和逻辑运算）

5.1.2 CPU的基本组成



5.1.2 CPU的基本组成

（1）中央处理器CPU = 运算器 + cache + 控制器

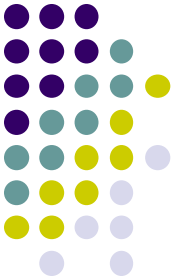
（2）运算器

- ALU
- 通用寄存器：R0～R3
- 暂存器：DR
- 状态字寄存器：PSW

（3）cache

- 指令cache：PC，IBUS
- 数据cache：AR，DBUS

5.1.2 CPU的基本组成



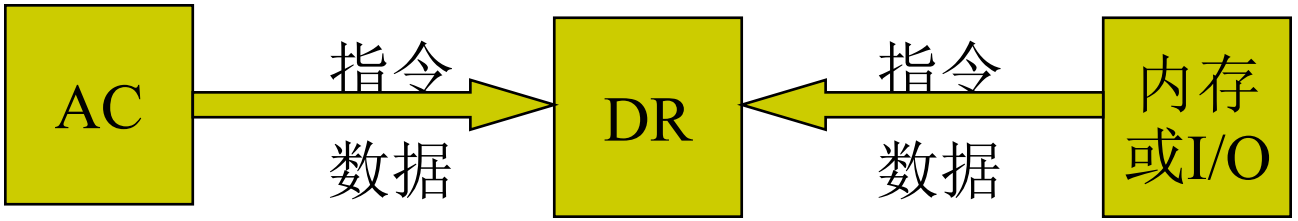
（3）控制器

控制器组成：程序计数器、指令寄存器、数据缓冲器、地址寄存器、通用寄存器、状态寄存器、时序发生器、指令译码器、总线（数据通路）

- 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

- 数据缓冲寄存器(DR)



  - 中转站
  - 补偿速度差别
- 指令寄存器(IR)
- 程序计数器(PC)
- 数据地址寄存器(AR)
- 通用寄存器(R_0～R_3)
- 状态字寄存器(PSW)

5.1.4 操作控制器和时序产生器

（1）数据通路

（2）操作控制器：为数据通路的建立提供各种操作信号。根据设计方法不同，可分为时序逻辑型和存储逻辑型：

- 硬布线控制器
- 微程序控制器

（3）时序产生器：提供定时和时序信号

其他功能部件：中断系统、总线接口等

5.2指令周期

5.2.1 指令周期的基本概念

5.2.2 MOV指令的指令周期

5.2.3 LAD指令的指令周期

5.2.4 ADD指令的指令周期

5.2.5 STO指令的指令周期

5.2.6 JMP指令的指令周期

5.2.7 用方框图语言表示指令周期

5.2.1 指令周期的基本概念

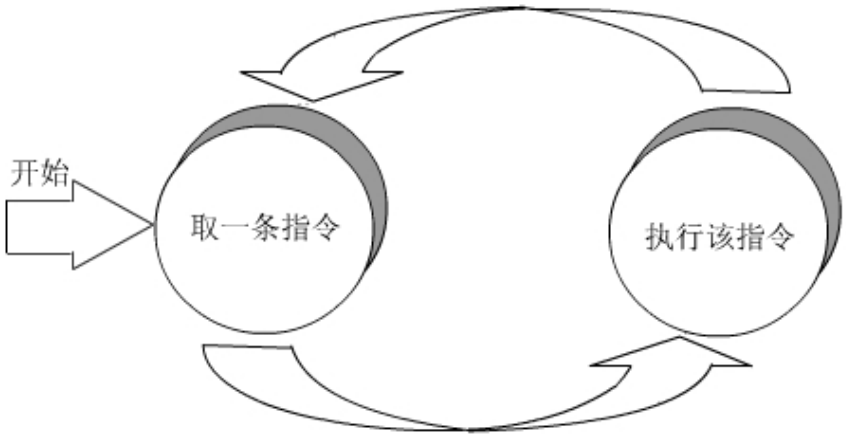
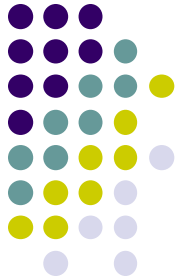


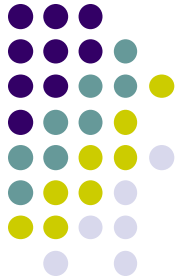
图5.2 取指令、执行指令周期序列

5.2.1 指令周期的基本概念



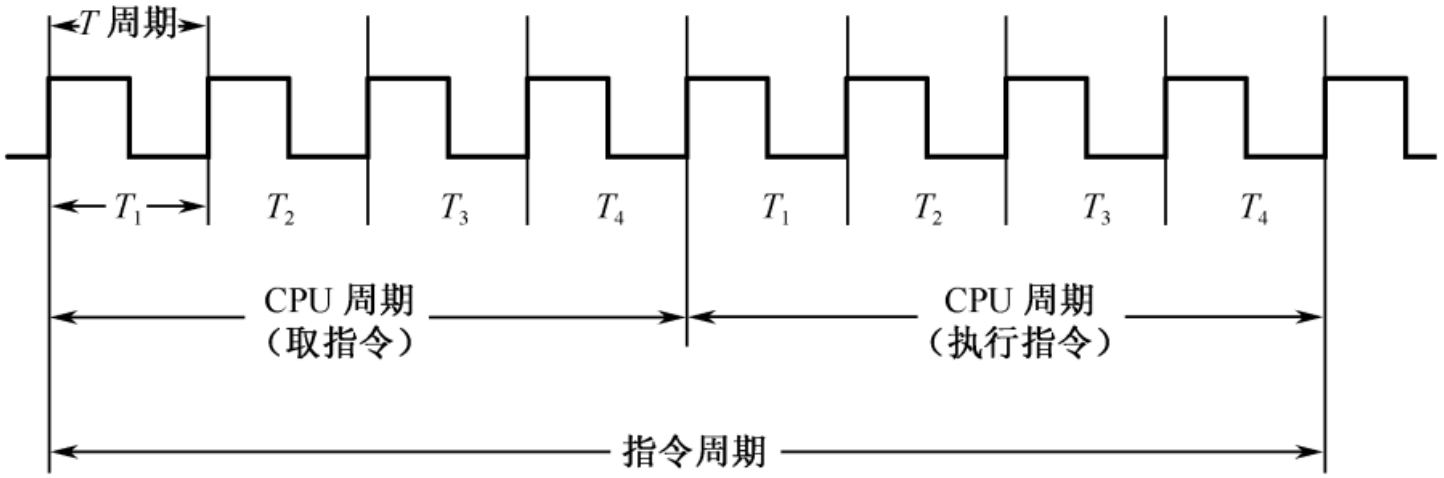
- 概念
  - 指令周期：指取指令、分析指令到执行完该指令所需的全部时间。
    - 各种指令的指令周期相同吗？为什么？
  - CPU周期通常又称时钟周期
    - 通常把一条指令周期划分为若干个机器周期，每个机器周期完成一个基本操作。
    - 主存的工作周期(存取周期)为基础来规定CPU周期，比如，可以用CPU读取一个指令字的最短时间来规定CPU周期
    - 不同的指令，可能包含不同数目的CPU周期。
    - 一个CPU周期中，包含若干个节拍脉冲（T周期）。
    - 单周期、多周期的概念

5.2.1 指令周期的基本概念



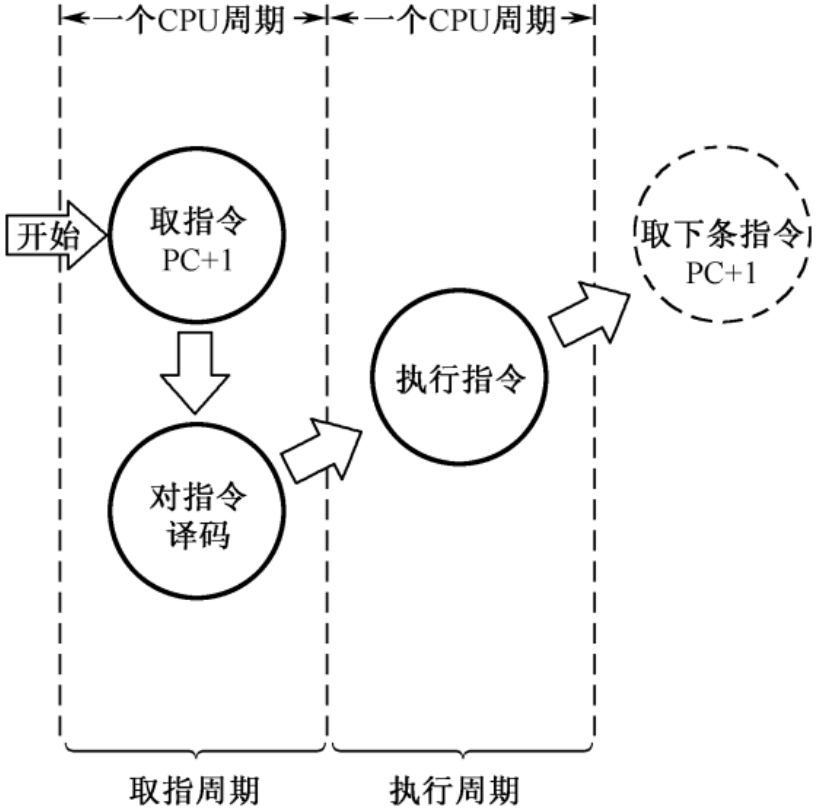
- 时钟周期
  - 在一个CPU周期内，要完成若干个微操作。这些微操作有的可以同时执行，有的需要按先后次序串行执行。因而需要把一个CPU周期分为若干个相等的时间段，每一个时间段称为一个节拍脉冲或T周期。
  - 时钟周期通常定义为机器主频的倒数。时钟周期不等于T周期。

5.2.1 指令周期的基本概念

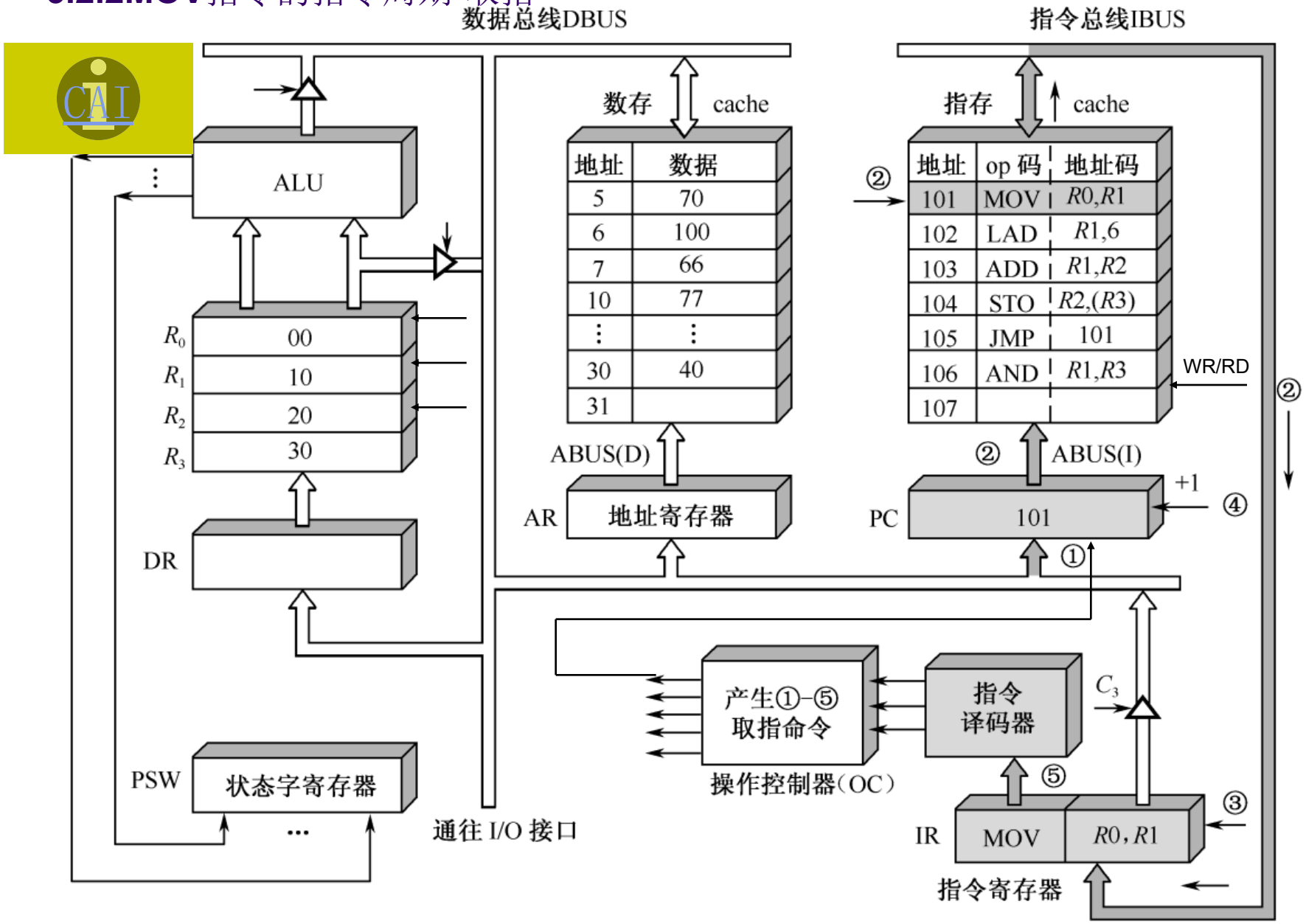


5.2.2MOV指令的指令周期

- 取指周期
- 执行周期

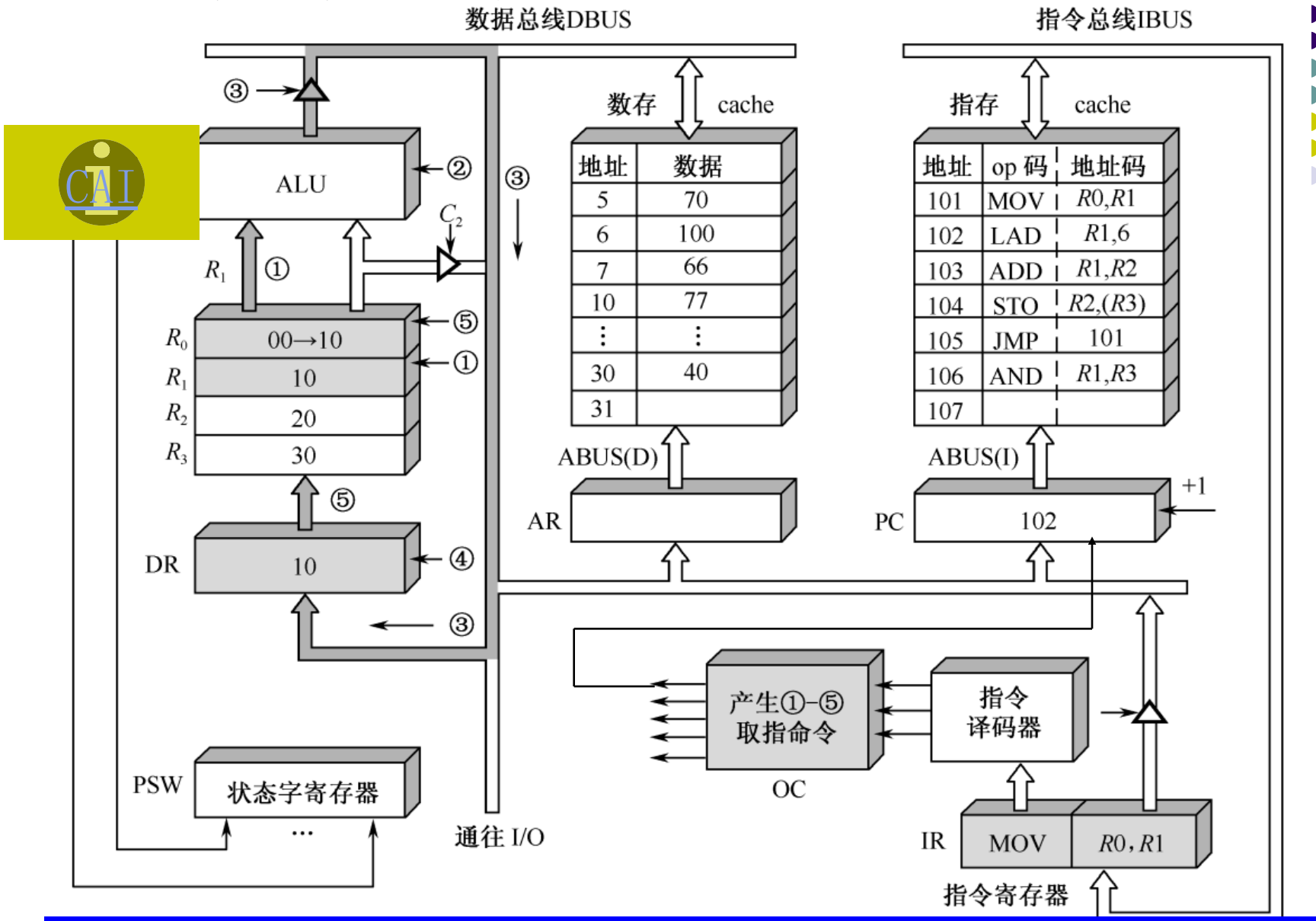


5.2.2MOV指令的指令周期-取指



②④ 程序i（⑥ CPU识别出是MOV指令，至此，取指周期即告结束。 存器IR；

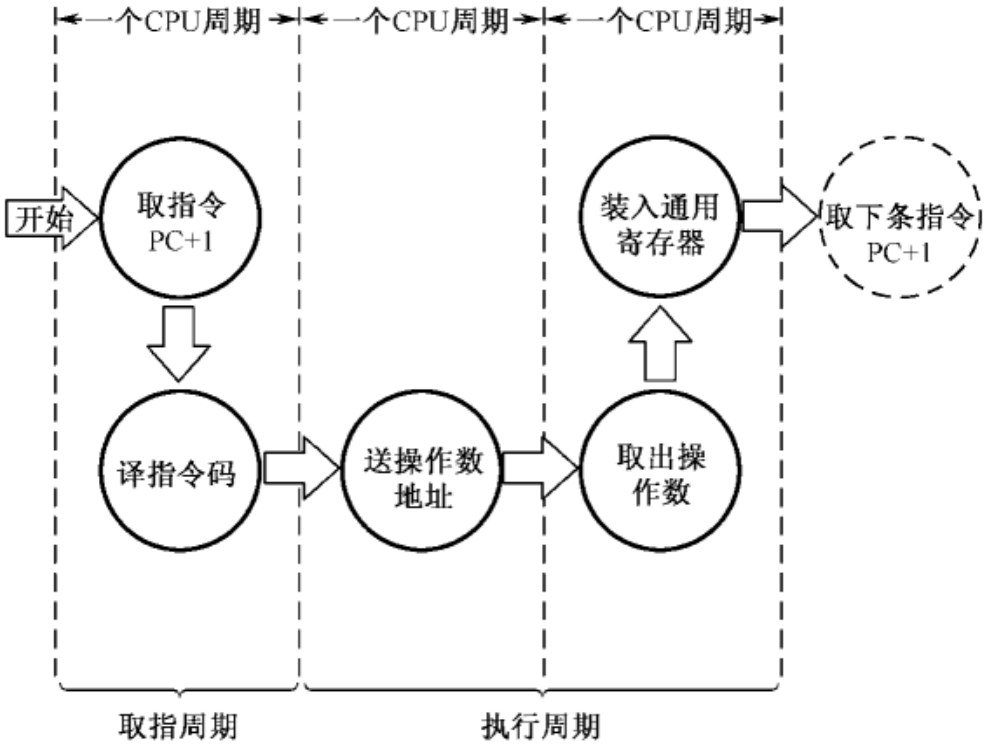
5.2.2 MOV指令的指令周期——执行



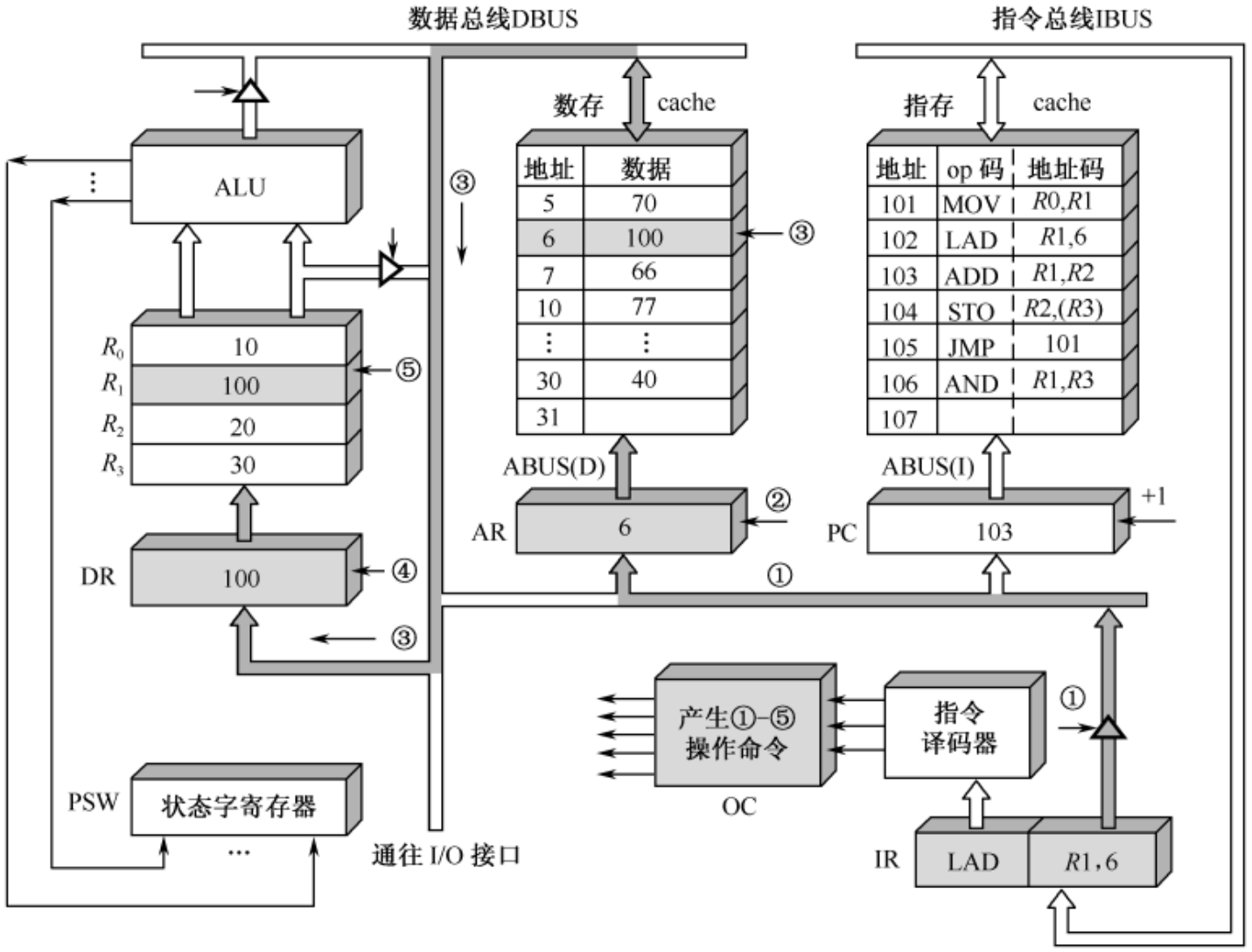
⑤ OC送出控制信号，将DR中的数据10打入到目标寄存器R0，R0的内容由00变为10。至此，MOV指令执行结束。

5.2.3LAD指令的指令周期

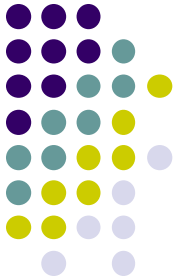
- 取指周期
- 执行周期



5.2.3LAD指令的指令周期

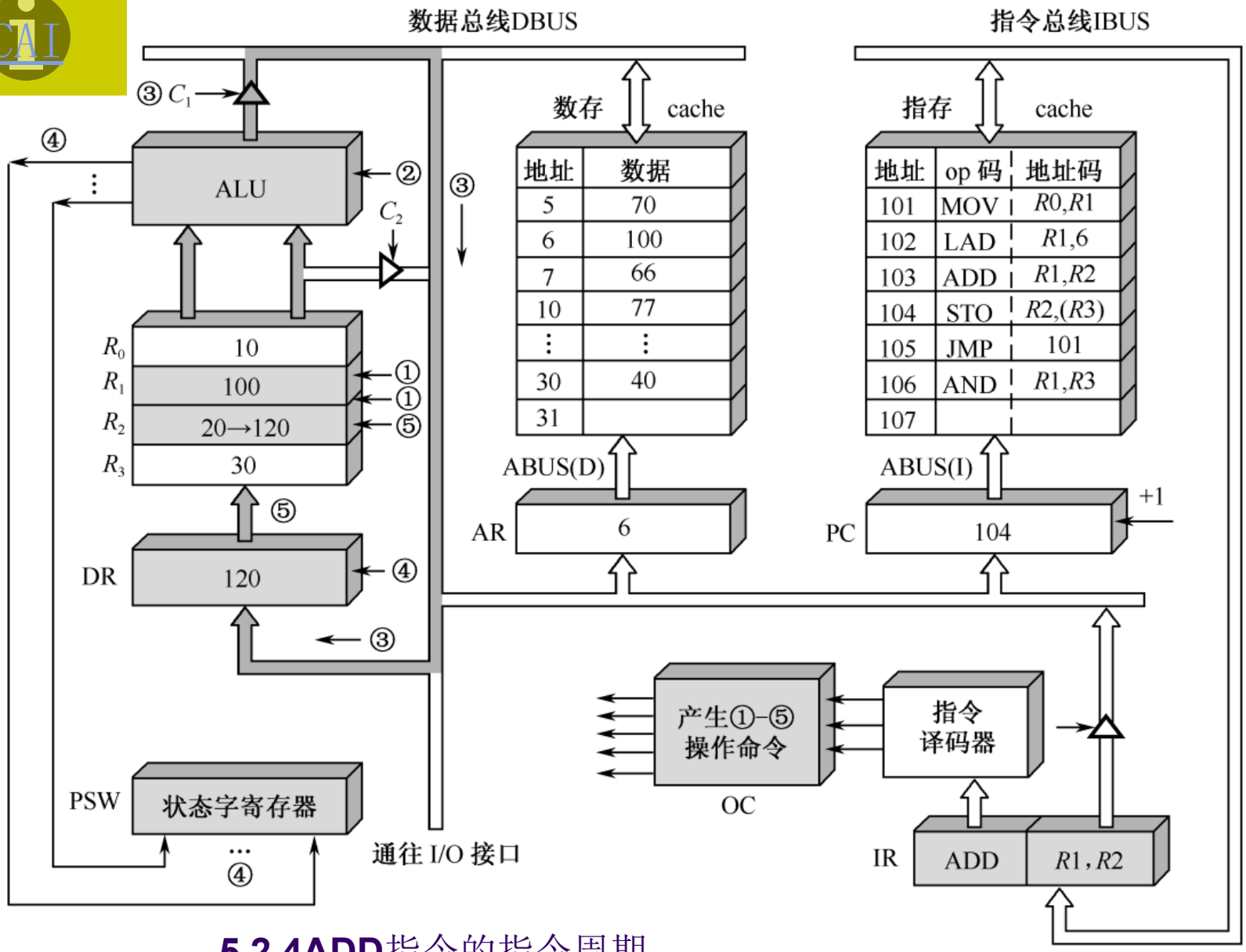


5.2.4 ADD指令的指令周期



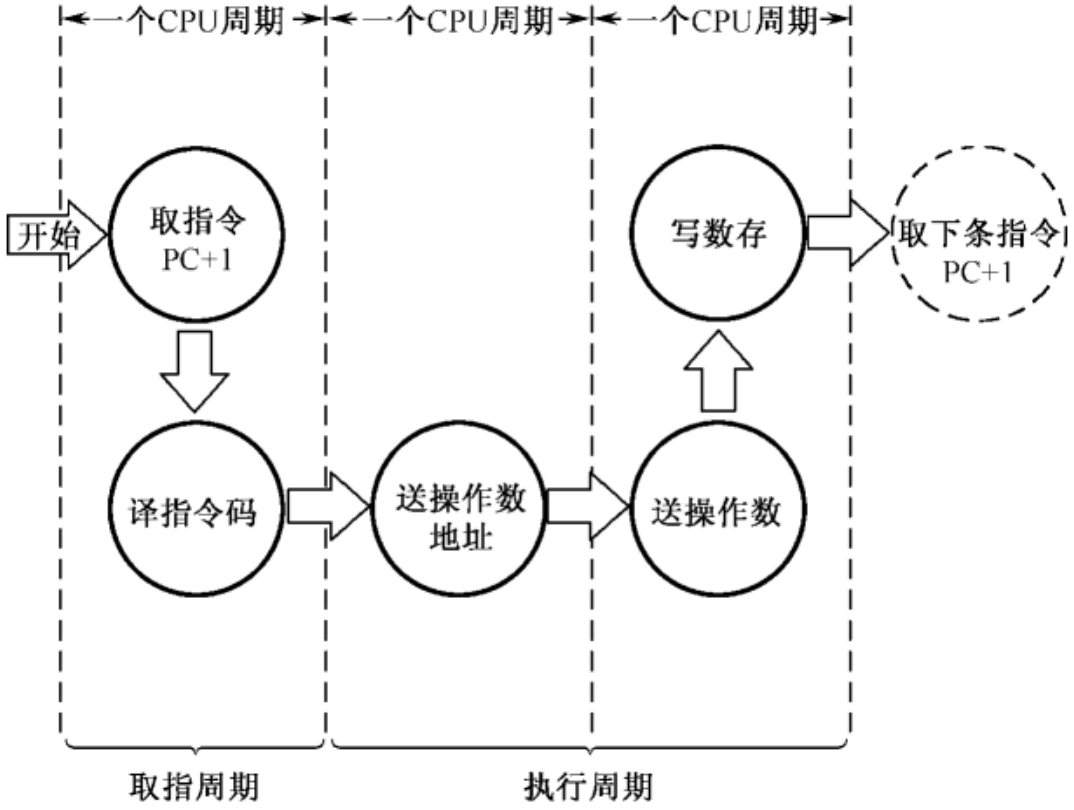
第1个CPU周期取ADD指令（指令cache）

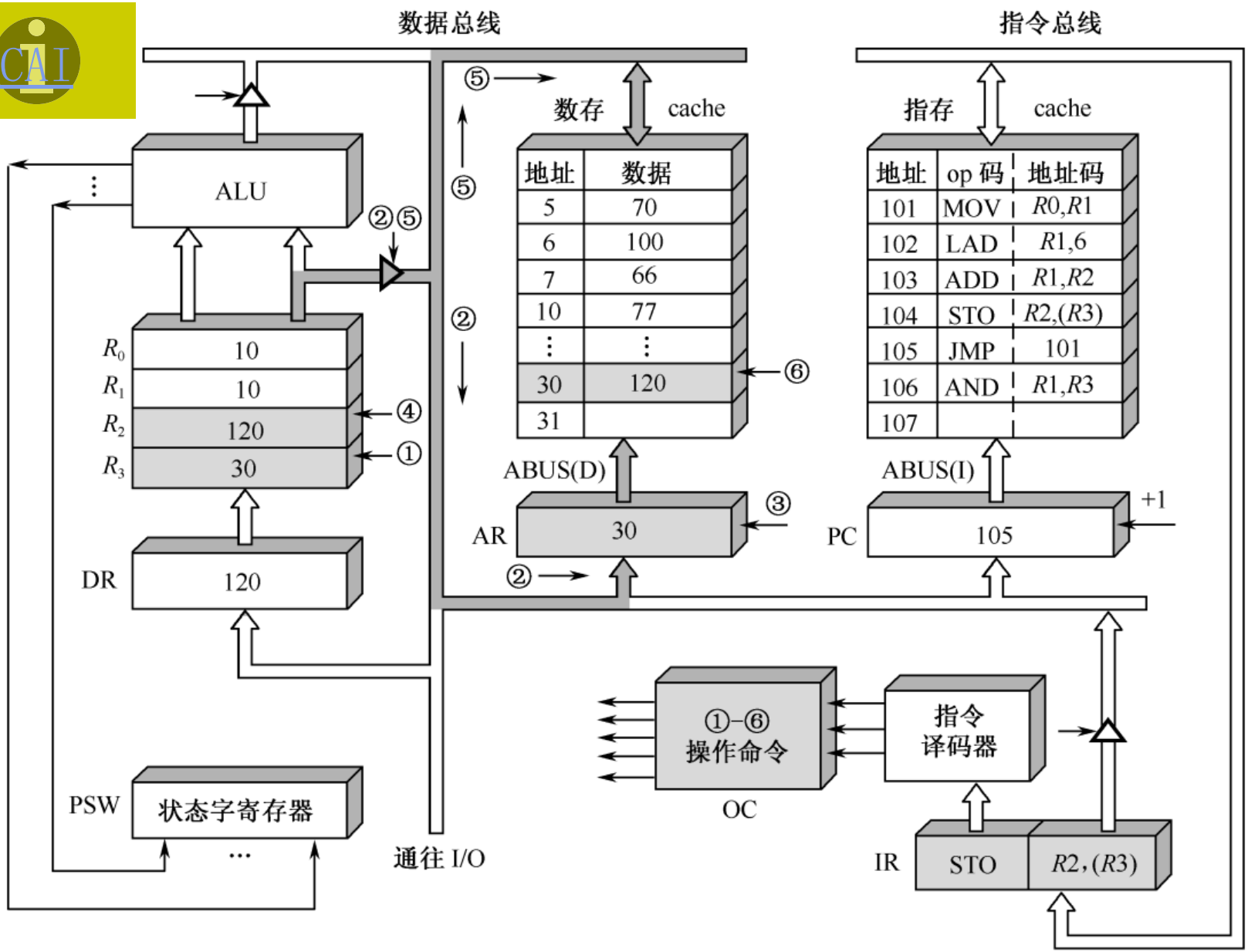
第2个CPU周期执行加法运算（运算器）



5.2.4ADD指令的指令周期

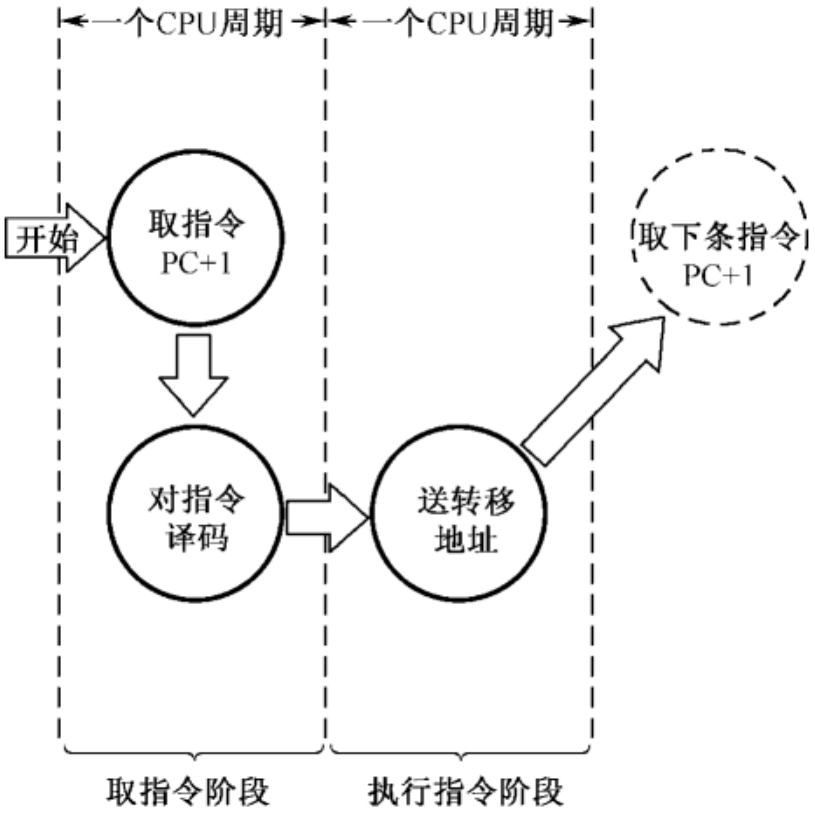
5.2.5STO指令的指令周期

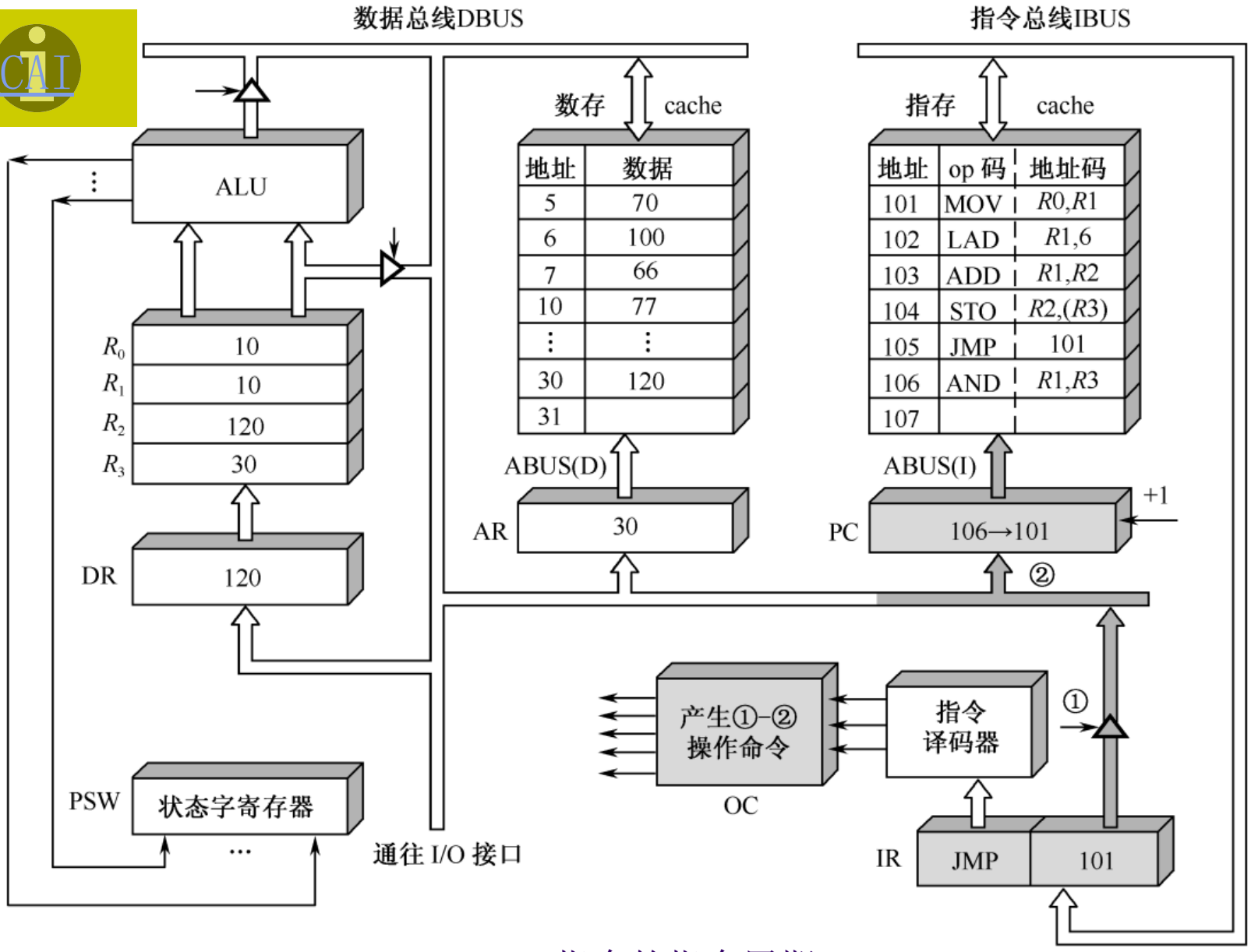




5.2.5STO指令的指令周期

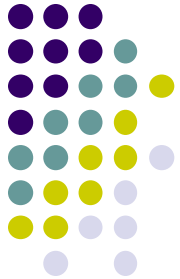
5.2.6JMP指令的指令周期





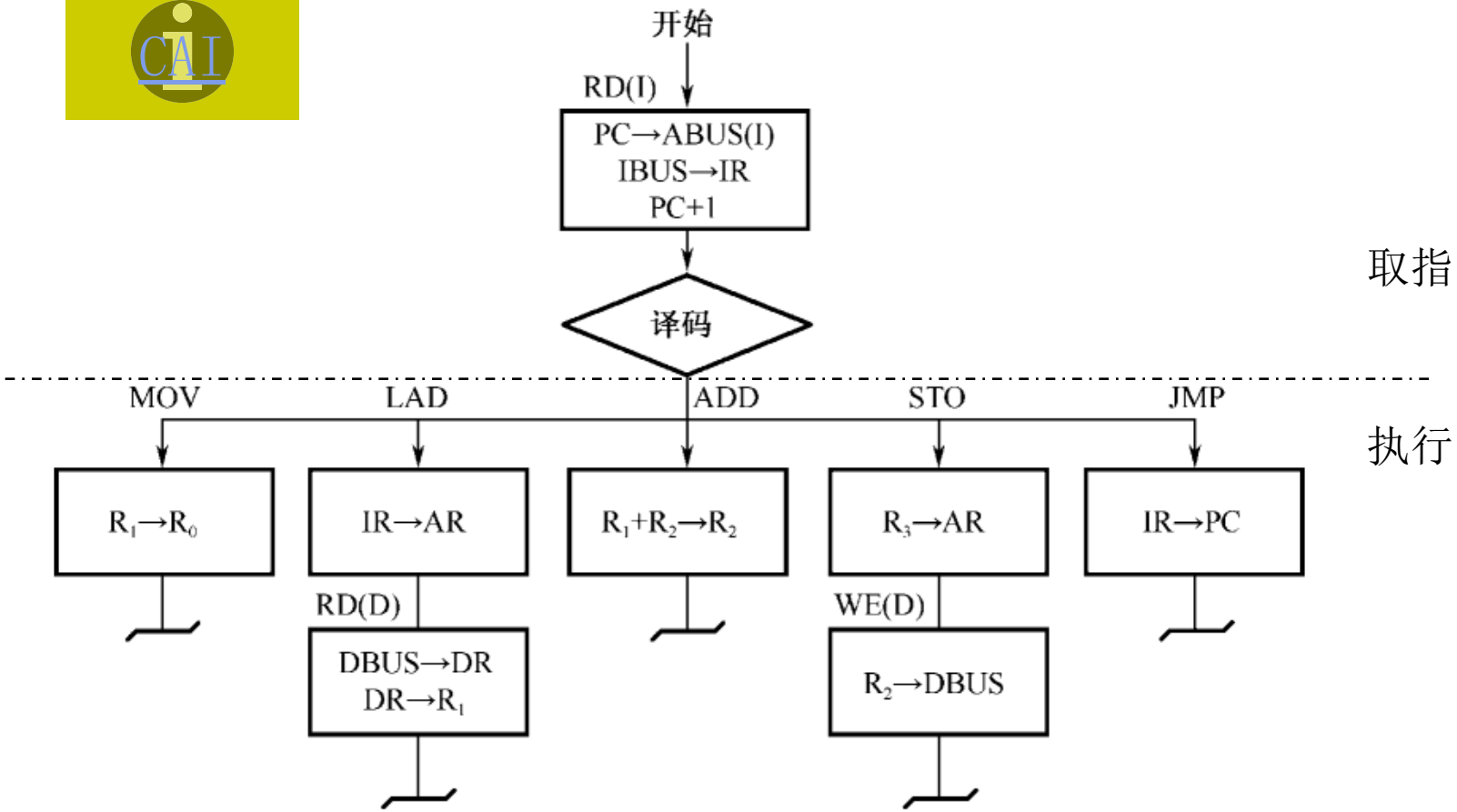
5.2.6JMP指令的指令周期

5.2.7用方框图语言表示的指令周期



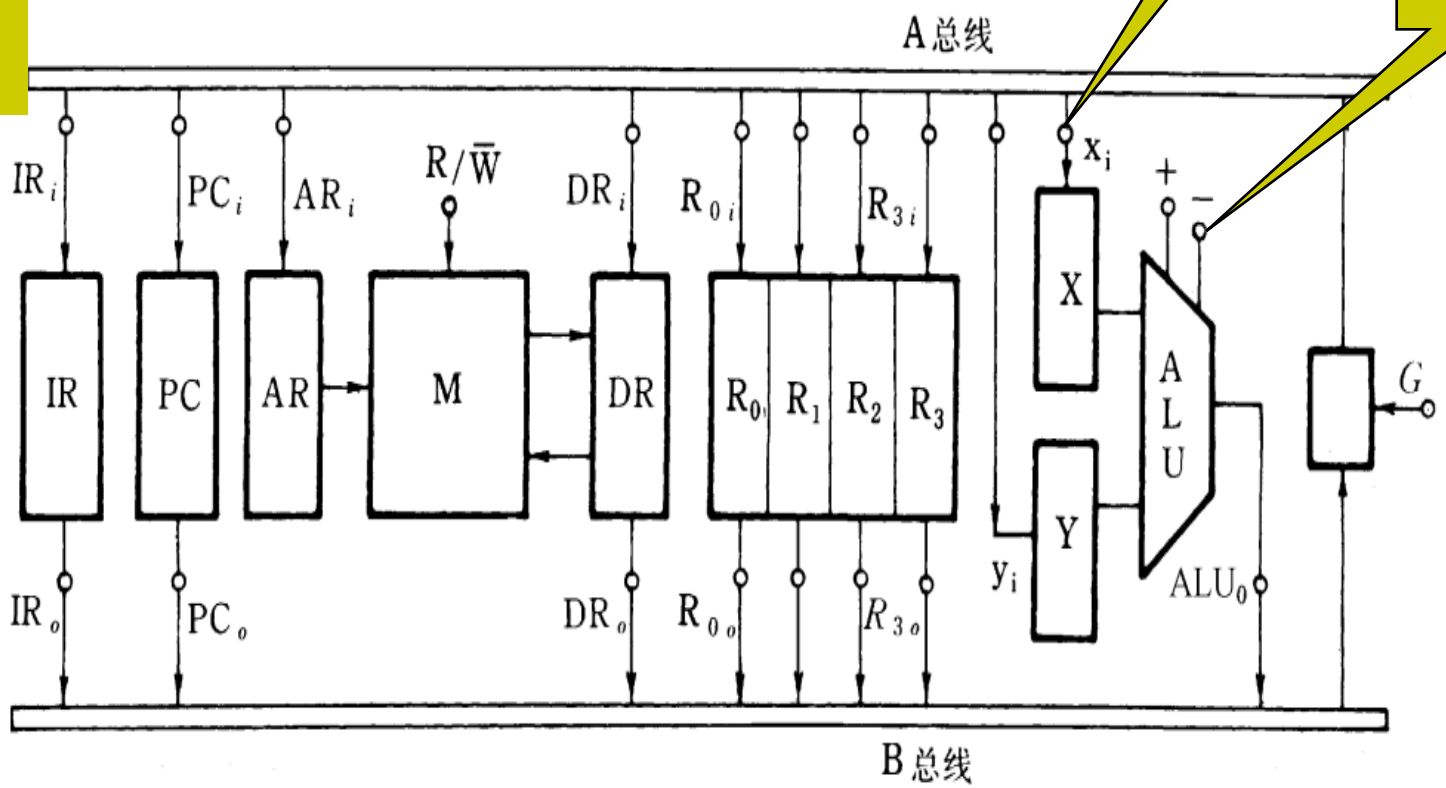
- 引入目的主要是为了教学目的（控制器设计）
- 方法：
  - 指令系统设计（模型机的五指令系统）
  - 方框——按CPU周期
  - 方框内内容——数据通路操作或控制操作
  - 菱形符号——判别或测试
  - ~——公操作
  - 前边所讲述的5种操作的框图描述

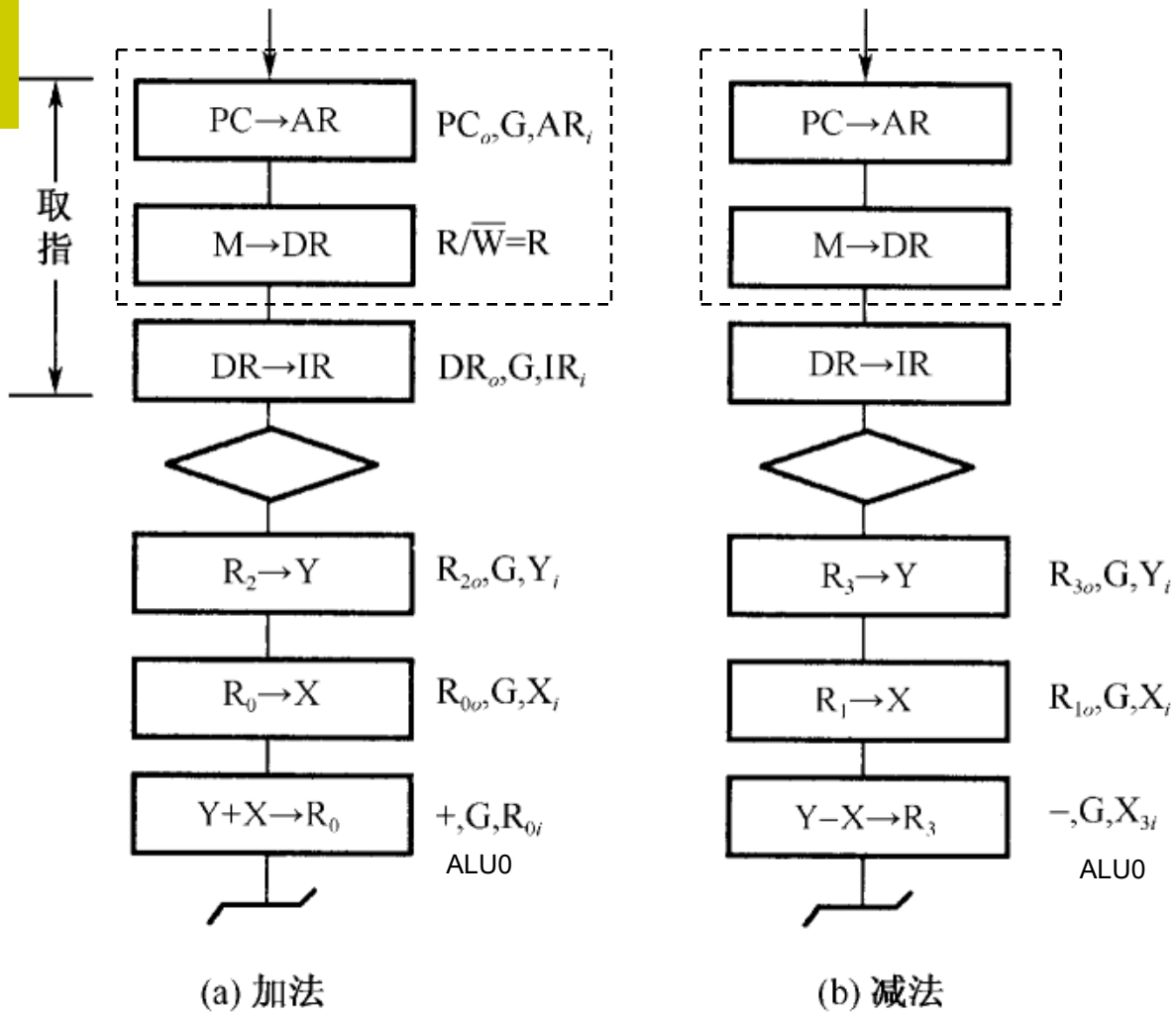
5.2.7用方框图语言表示指令周期



5.2.7用方框图语言表示指令周期

[例1] 双总线结构机器的数据通路图





注意微操作控制信号（右边）

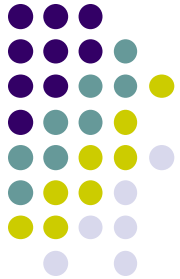
5.3 时序产生器和控制方式

5.3.1时序产生器作用和体制

5.3.2时序信号产生器

5.3.3控制方式

5.3.1、时序产生器作用和体制



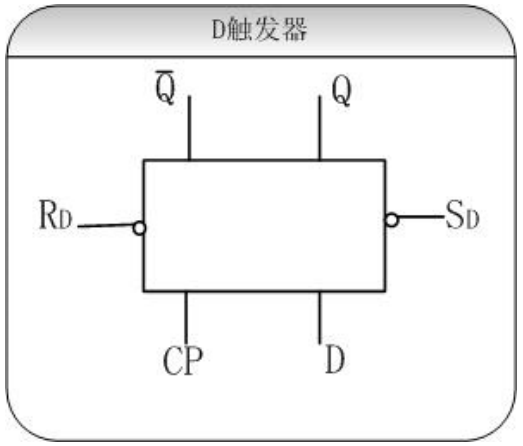
1. 作用：
   - CPU中的控制器用它指挥机器的工作
   - CPU可以用时序信号/周期信息来辨认从内存中取出的是指令（取指）还是数据（执行）
   - 一个CPU周期中时钟脉冲对CPU的动作有严格的约束
   - 操作控制器发出的各种信号是时间（时序信号）和空间（部件操作信号）的函数。

5.3.1、时序产生器作用和体制

2. 体制

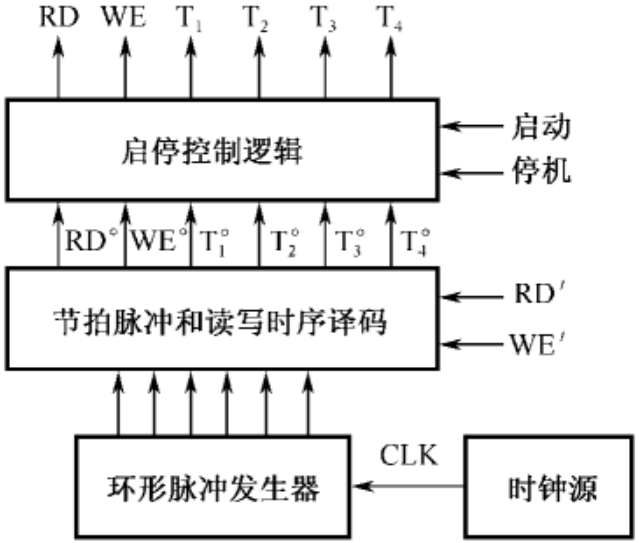
- 组成计算机硬件的器件特性决定了时序信号的基本体制是电位—脉冲制（以触发器为例）
- D为电位输入端，CP（Clock Pulse）为脉冲输入端
- R,S为电位输入端
- 特性方程如下
  - D=0时，CP上升沿到来时，D触发器状态置0
  - D=1时，CP上升沿到来时，D触发器状态置1
- 硬布线控制器：主状态周期—节拍电位—节拍脉冲三级体制

  微程序控制器：节拍电位—节拍脉冲二级体制



5.3.2、时序信号产生器

1. 功能：产生时序信号
   - 各型计算机产生时序电路不相同
   - 大、中型计算机的时序电路复杂，微型计算机的时序电路简单
2. 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



5.3.2 时序信号产生器

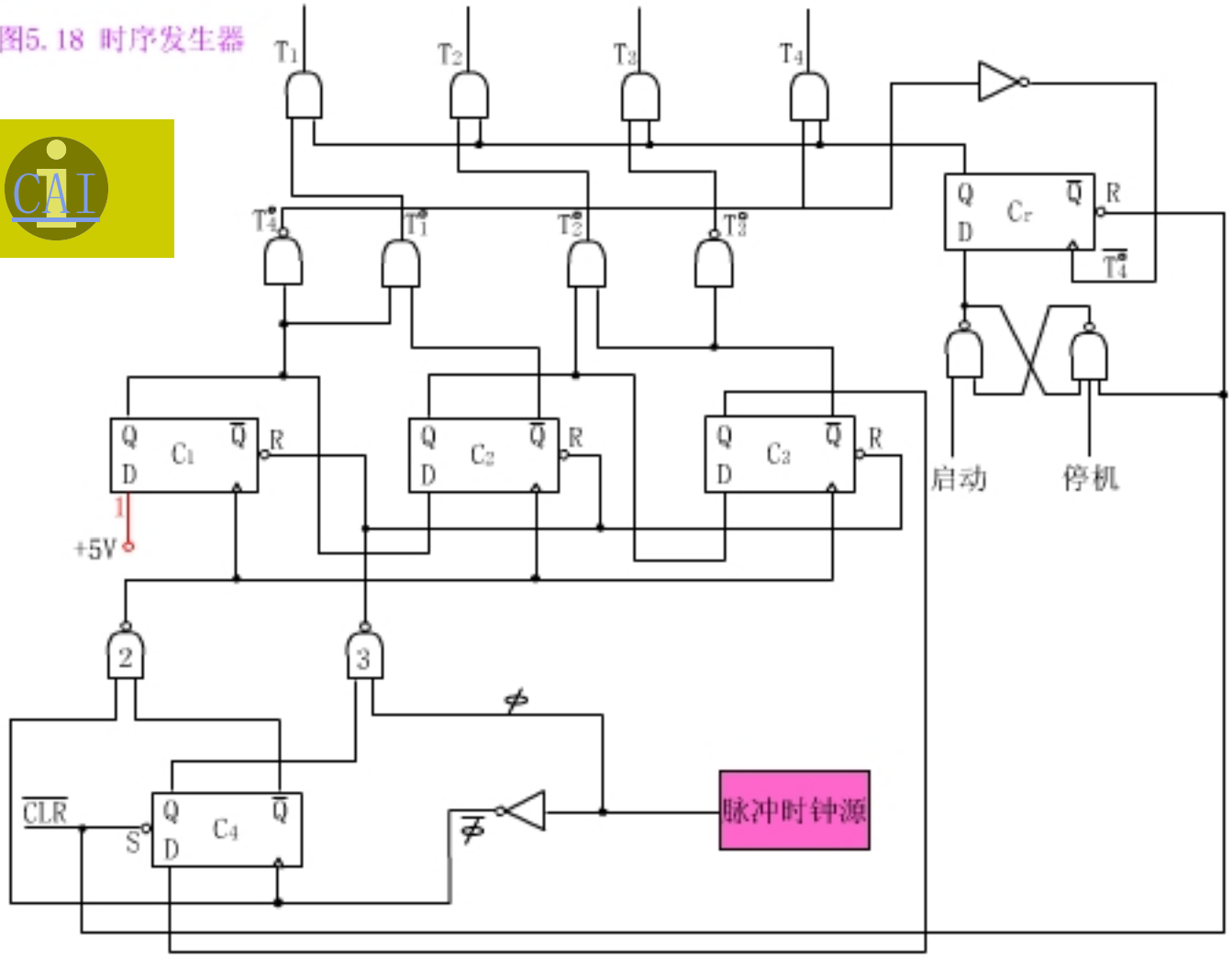
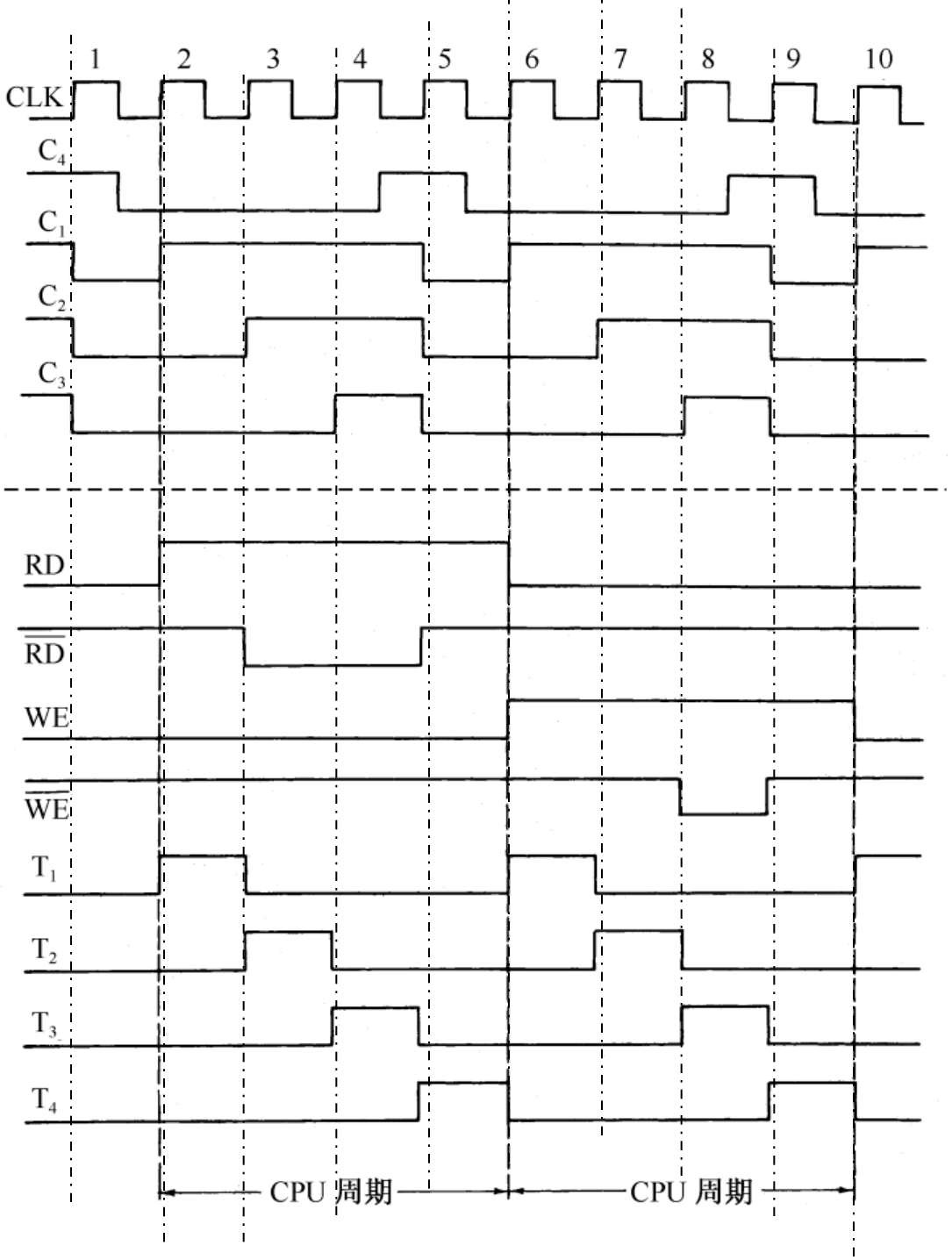


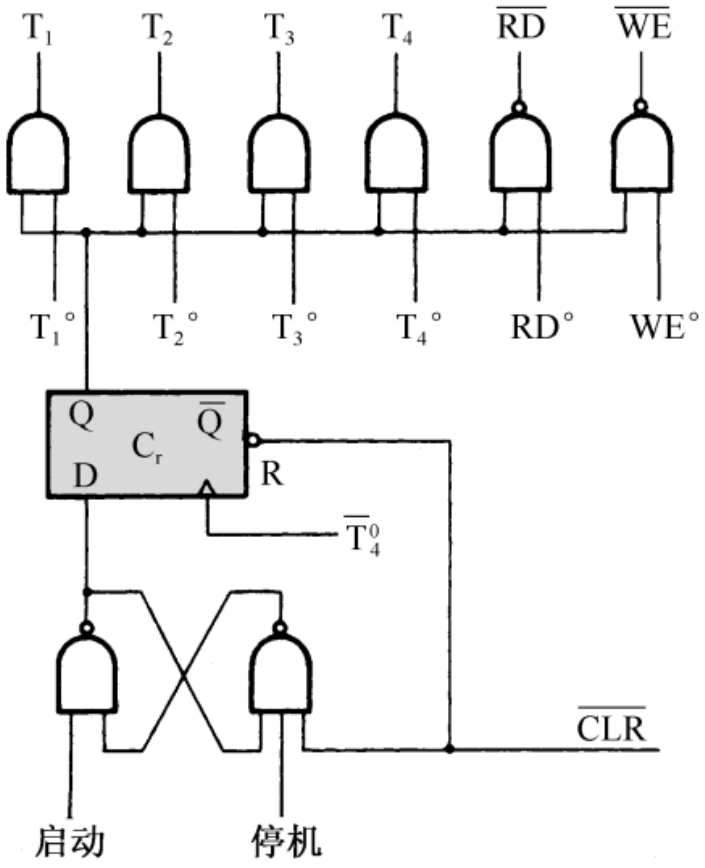
图**5.18** 节拍电位与节拍脉冲时序关系



5.3.2 时序信号产生器启停控制逻辑

启停控制逻辑

- 启动、停机是随机的，对读/写时序信号也需要由启停逻辑加以控制。
- 当运行触发器为“1”时，打开时序电路。当计算机启动时，一定要从第1个节拍脉冲前沿开始工作。
- 当运行触发器“0”时，关闭时序产生器。停机时一定要在第4个节拍脉冲结束后关闭时序产生器。

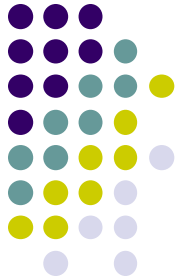


5.3.3 控制方式

- 机器指令所包含的CPU周期数反映了指令的复杂程度，不同CPU周期的操作信号的数目和出现的先后次序也不相同。
- 控制方式：控制不同操作序列时序信号的方法。
- 分为以下几种：
  - 同步控制方式
  - 异步控制方式
  - 联合控制方式

5.3.3 控制方式

- 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

5.4.1 微程序控制原理

5.4.2 微程序设计技术

5.4 微程序控制器

基本思想：仿照解题的方法，把操作控制信号编制成微指令，存放到控制存储器里，运行时，从控存中取出微指令，产生指令运行所需的操作控制信号。从上述可以看出，微程序设计技术是用软件方法来设计硬件的技术。

5.4.1 微程序控制原理

1、微命令和微操作
2、微指令和微程序
3、微程序控制器原理框图
4、微程序举例
5、CPU周期与微指令周期的关系
6、机器指令与微指令的关系

5.4.1微程序控制原理

1、微命令和微操作

1. 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。
2. 微操作：是微命令的操作过程。
   - 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。

5.4.1微程序控制原理

1、微命令和微操作

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：

1. 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。可以编码
2. 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。必须各占一位

- 举一个例子看一下：见下图

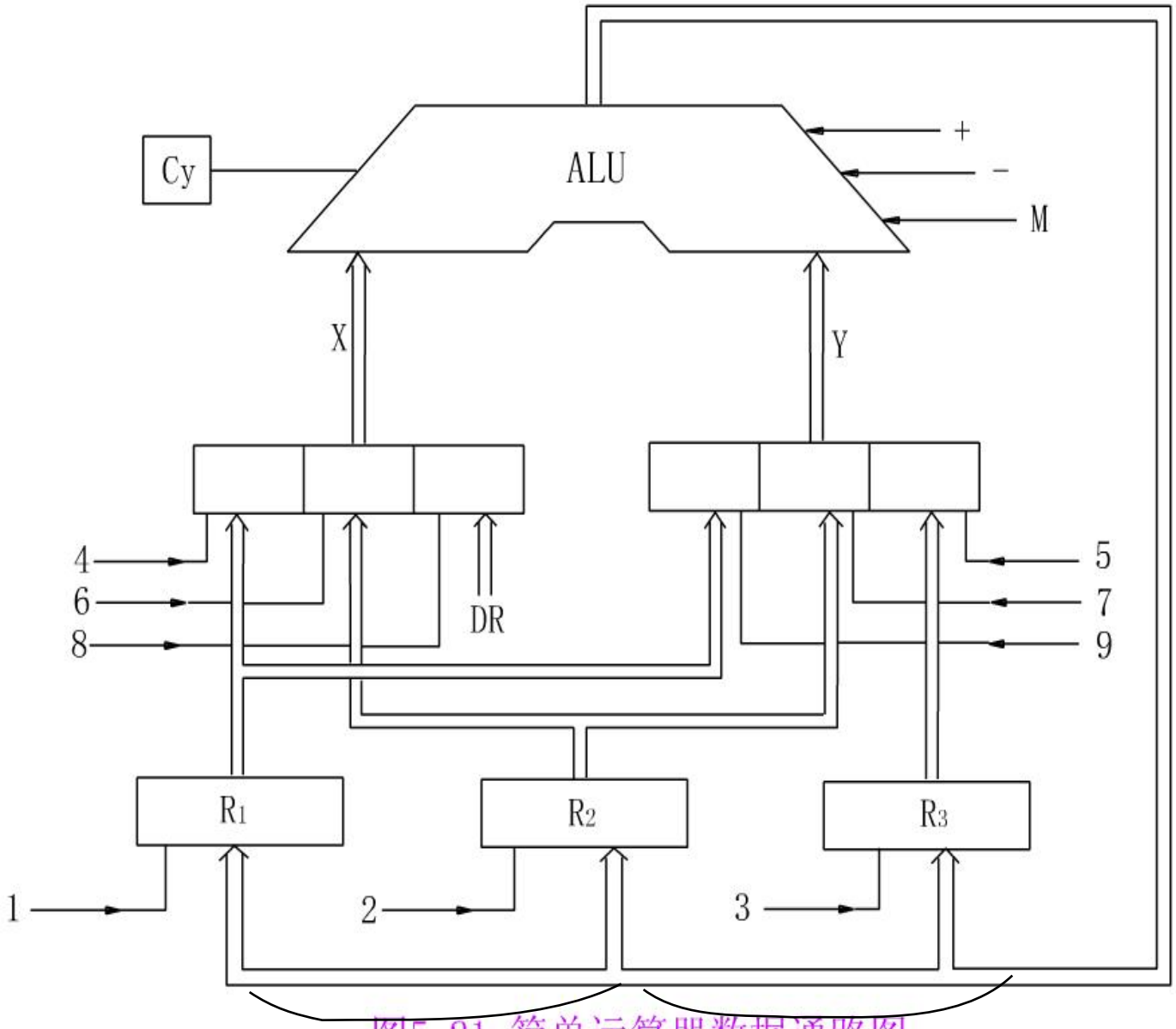
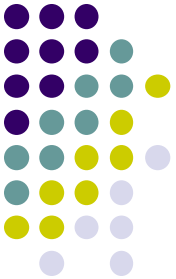


图5.21 简单运算器数据通路图

5.4.1微程序控制原理



2、微指令和微程序

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
    - 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。

5.4.1微程序控制原理

2、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
- 下面我们举一个十进制加法指令为实例。

5.4.1微程序控制原理

2、微指令和微程序

- 微指令基本格式

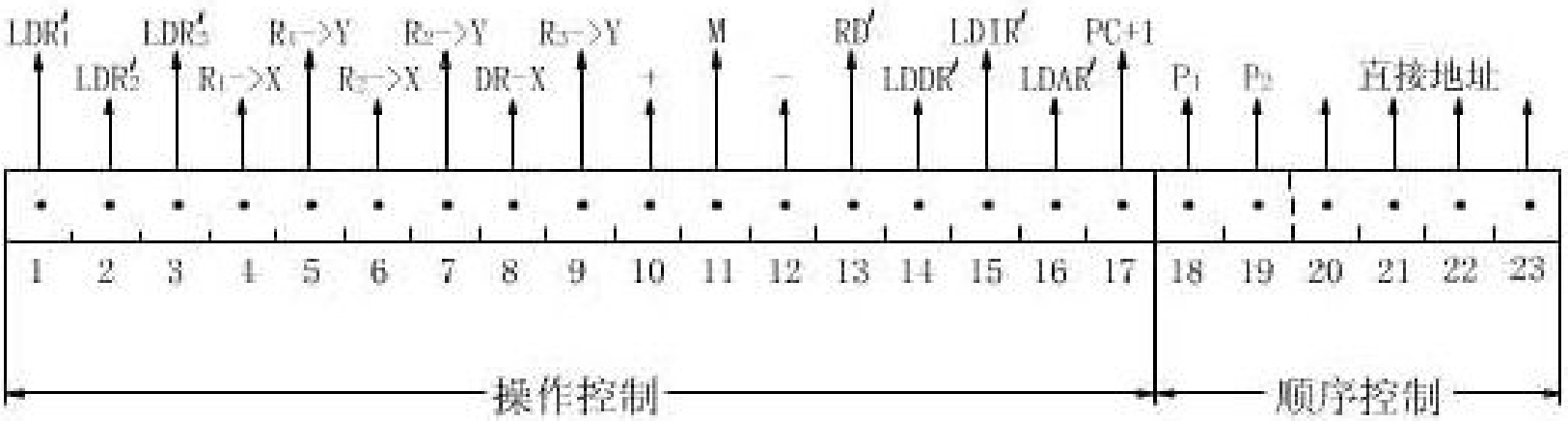
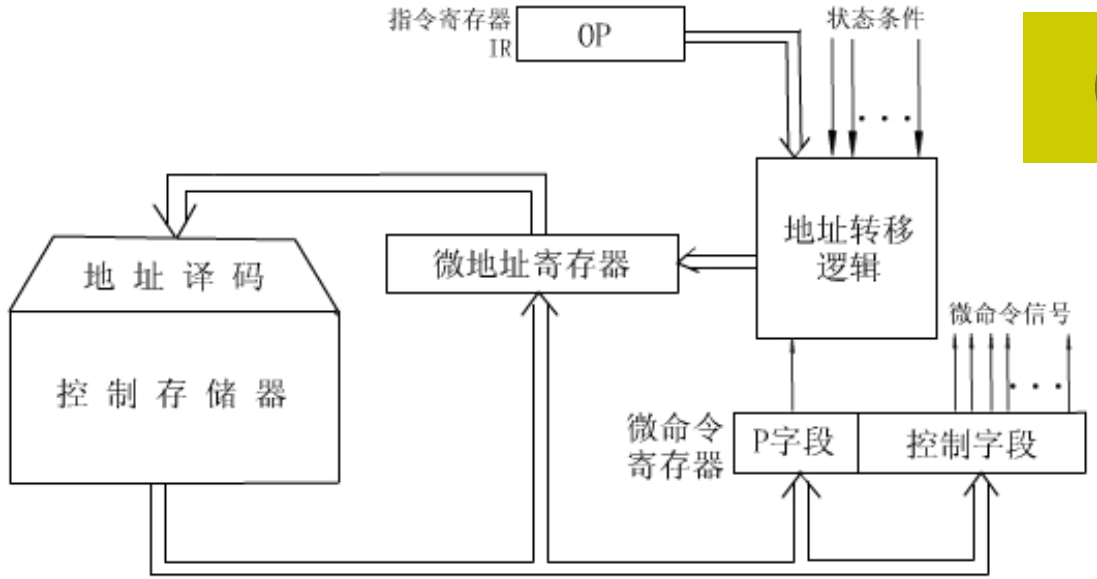


图5.22 微指令基本格式

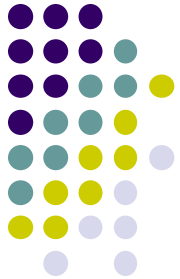
5.4.1微程序控制原理

3、微程序控制器原理框图



- 控制存储器(μCM)。
  - 这是微程序控制器的核心部件，用来存放微程序。其性能(包括容量、速度、可靠性等)与计算机的性能密切相关。

5.4.1微程序控制原理



- 微指令寄存器(μIR)
  - 用来存放从μCM取出的正在执行的微指令，它的位数同微指令字长相等。
- 微地址形成部件
  - 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(μMAR)
  - 它接受微地址形成部件送来的微地址，为下一步从μCM中读取微指令作准备。

5.4.1微程序控制原理

4、微程序举例

- 以十进制加法指令流程

数据通路图

操作流程图

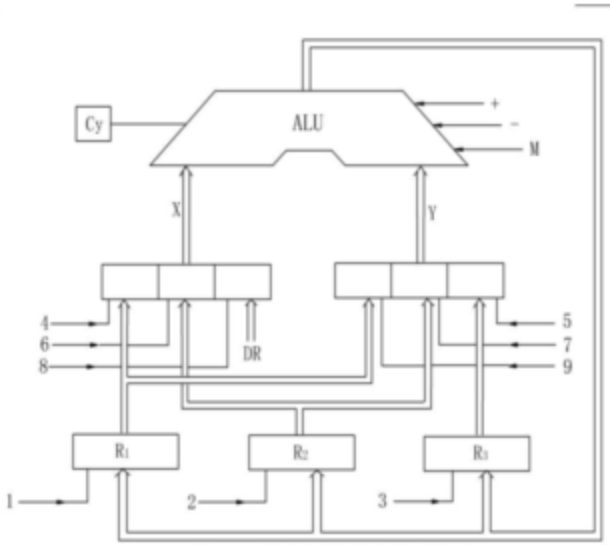
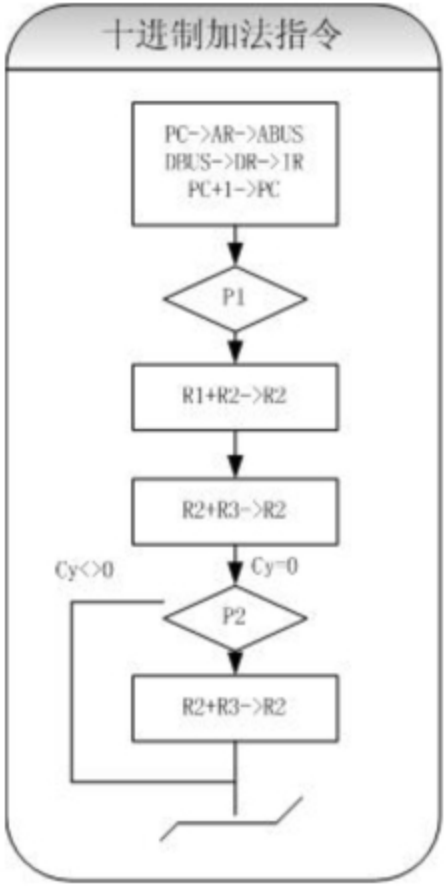


图5.21 简单运算器数据通路图



5.4.1微程序控制原理

4、微程序举例

- 四条微指令如下

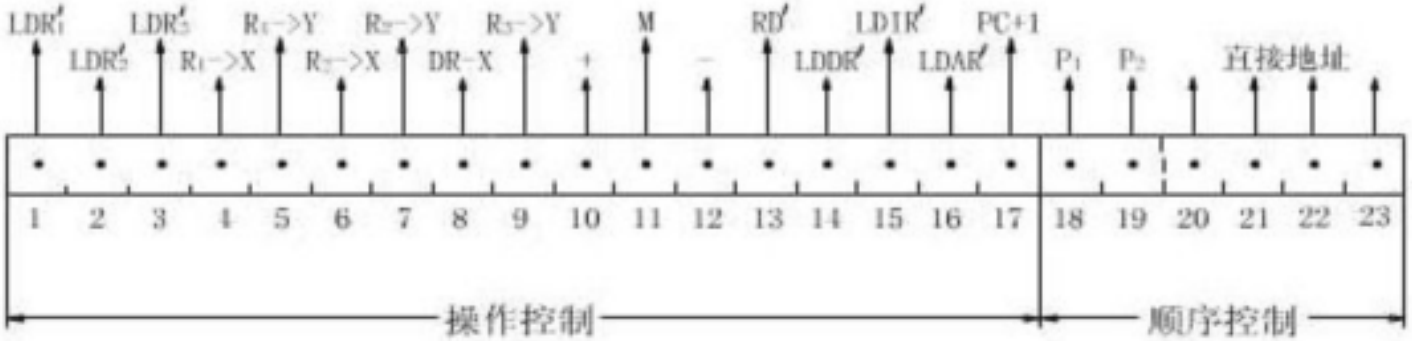
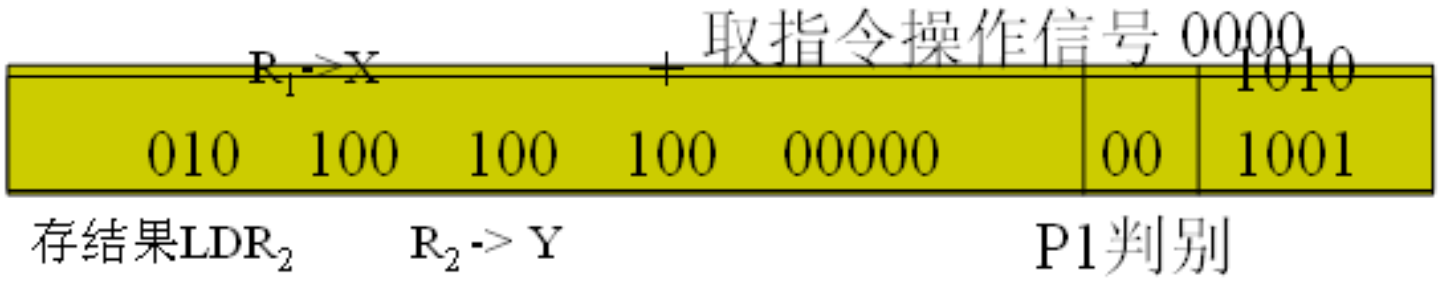


图5.22 微指令基本格式



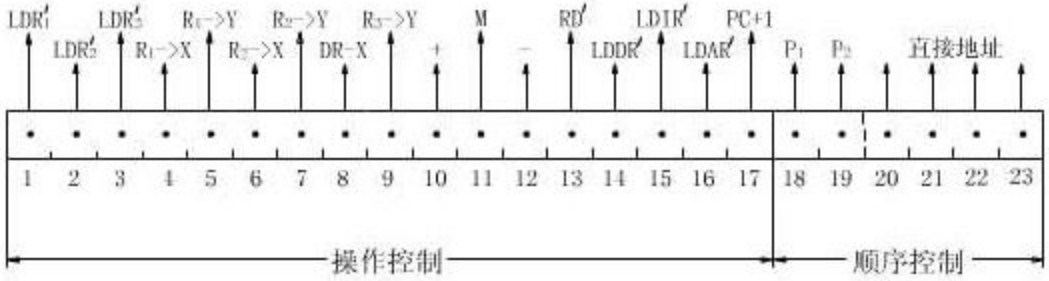
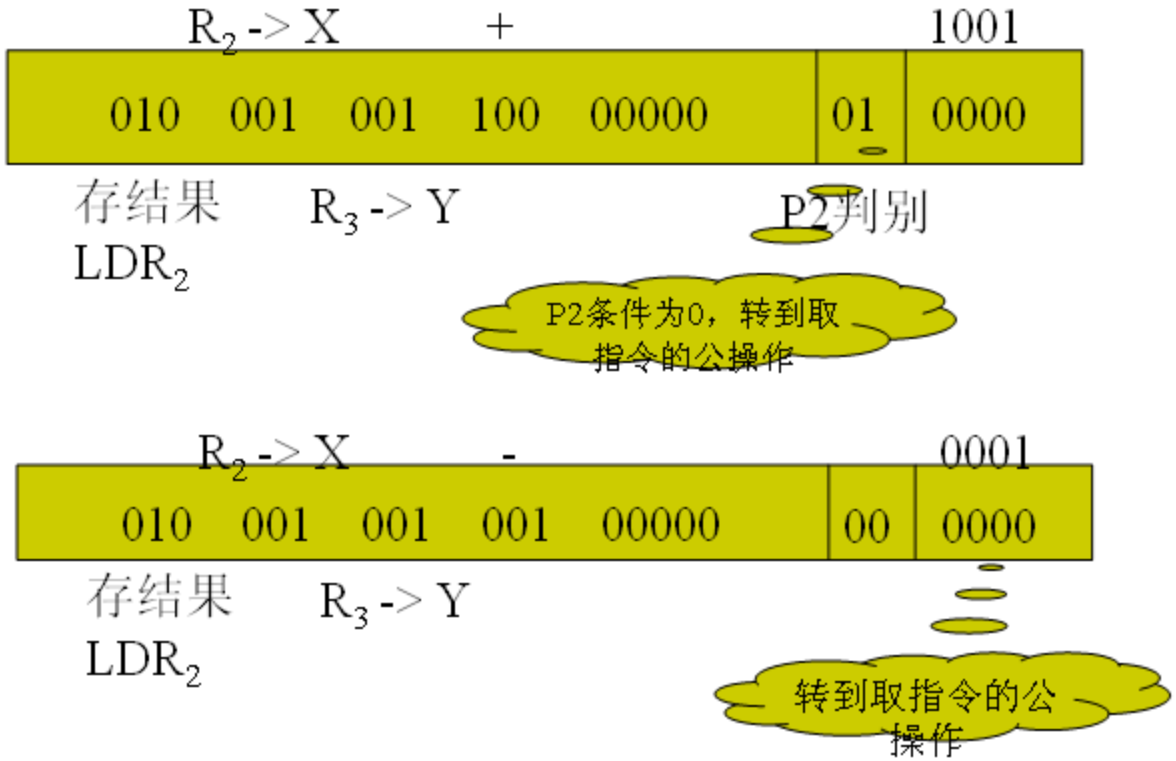
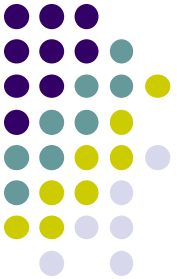


图5.22 微指令基本格式



5.4.1微程序控制原理



- 微程序控制器的工作过程

  (1)执行取指令的公共操作。取指令的公共操作通常由一段取指微程序来完成，在机器开始运行时，自动将取指微程序的入口微地址送μMAR，并从μCM中读出相应的微指令送入μIR。微指令的操作控制字段产生有关的微命令，用来控制实现取机器指令的公共操作。取指微程序的入口地址一般为μCM的0号单元，当取指微程序执行完后，从主存中取出的机器指令就已存人指令寄存器IR中了。

  (2)由机器指令的操作码字段通过微地址形成部件产生出该机器指令所对应的微程序的入口地址，并送入μMA

  (3)从μCM中逐条取出对应的微指令并执行之，每条微指令都能自动产生下一条微指令的地址。

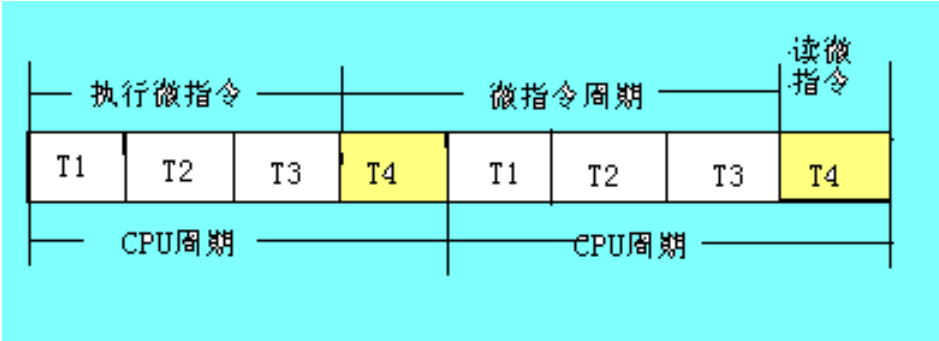
5.4.1微程序控制原理

(4)一条机器指令对应的微程序的最后一条微指令执行完毕后，其下一条微指令地址又回到取指微程序的人口地址，从而继续第(1)步，以完成取下条机器指令的公共操作。

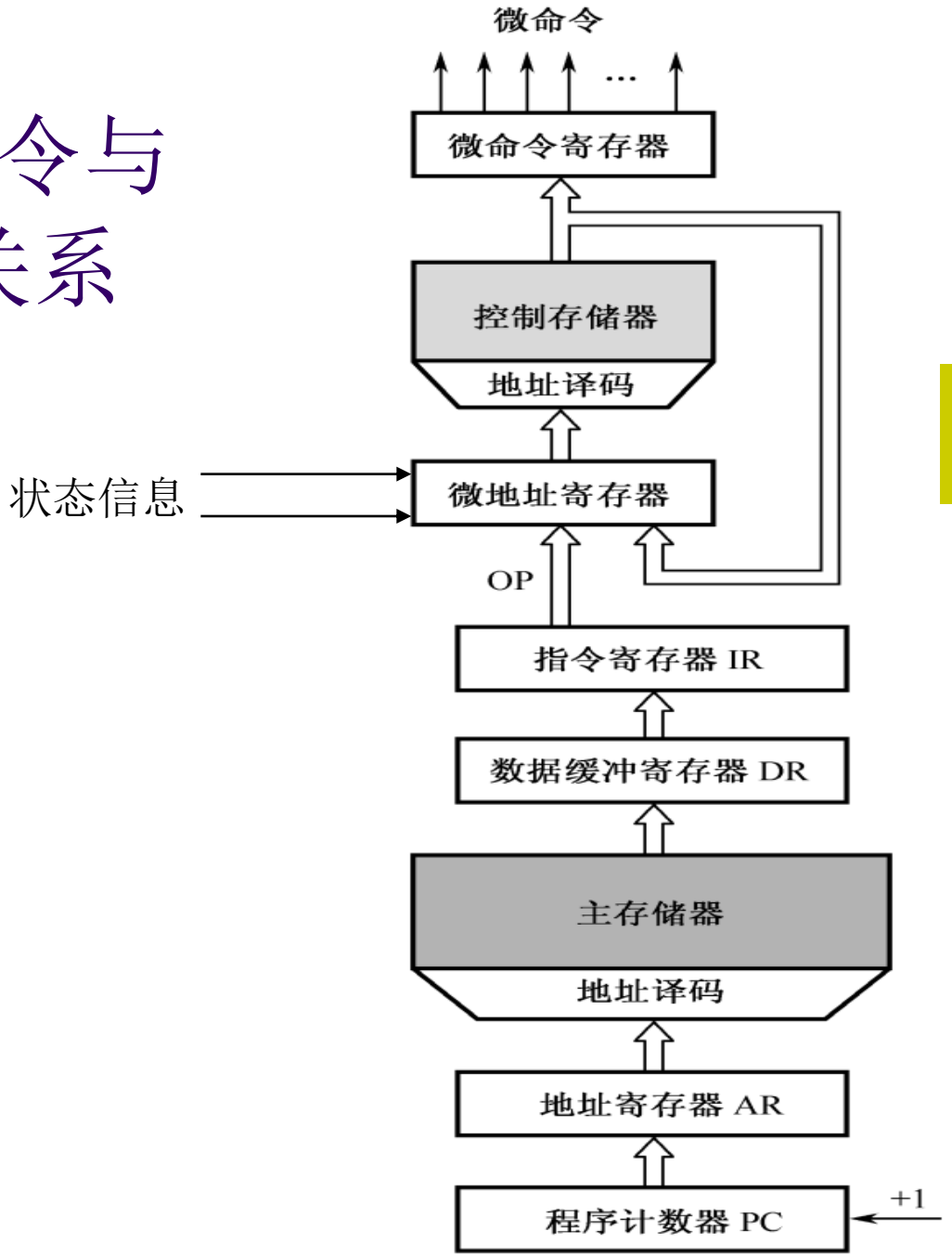
- 以上是一条机器指令的执行过程，如此周而复始，直到整个程序的所有机器指令执行完毕。

5.4.1微程序控制原理

5、CPU周期和微指令周期的关系



6、机器指令与微指令的关系



5.4.2 微程序设计技术

设计微指令应当追求的目标

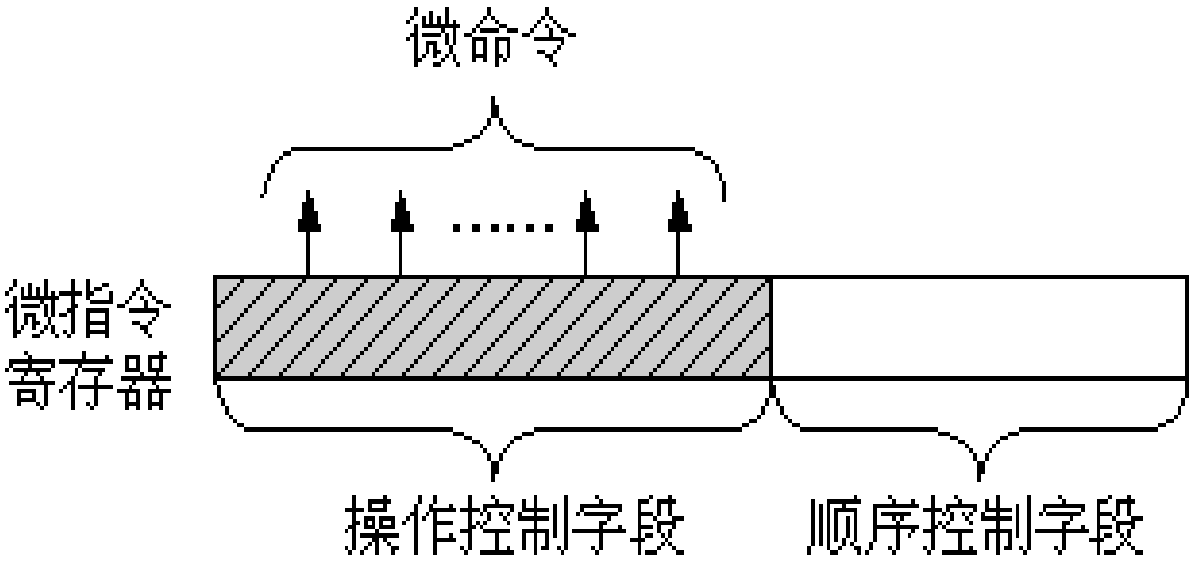
- 有利于缩短微指令的长度
- 有利于缩小CM的容量
- 有利于提高微程序的执行速度
- 有利于对微指令的修改
- 有利于提高微程序设计的灵活性

5.4.2 微程序设计技术

1、微命令的编码方法

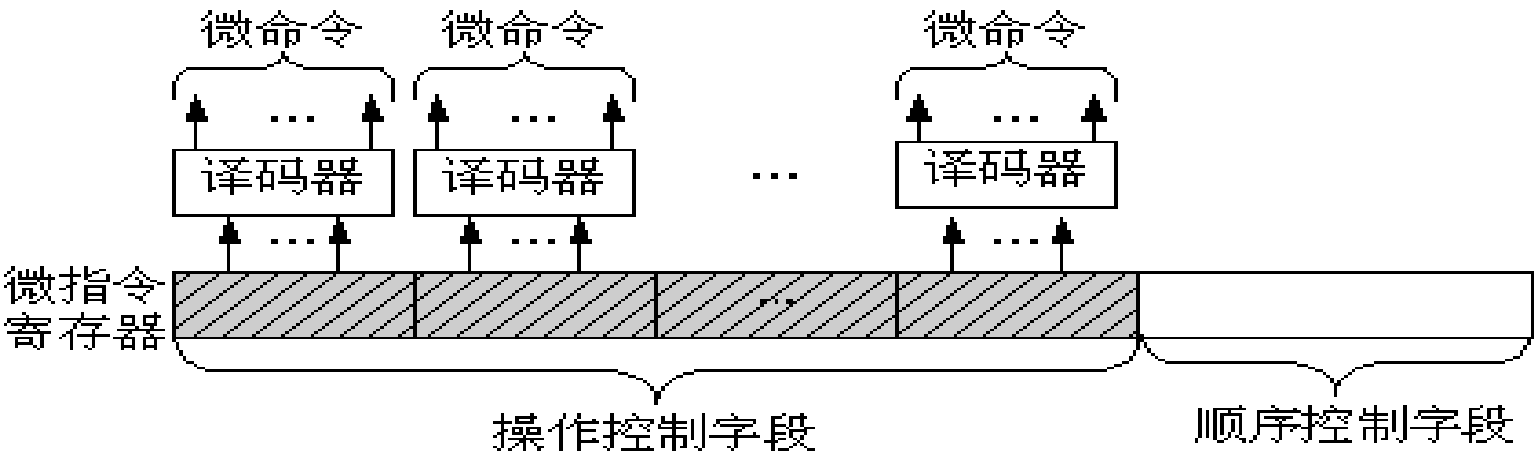
编码有三种方法：直接表示法/编码表示法/混合表示法

- 直接表示法：操作控制字段中的各位分别可以直接控制计算机，不需要进行译码。



5.4.2 微程序设计技术

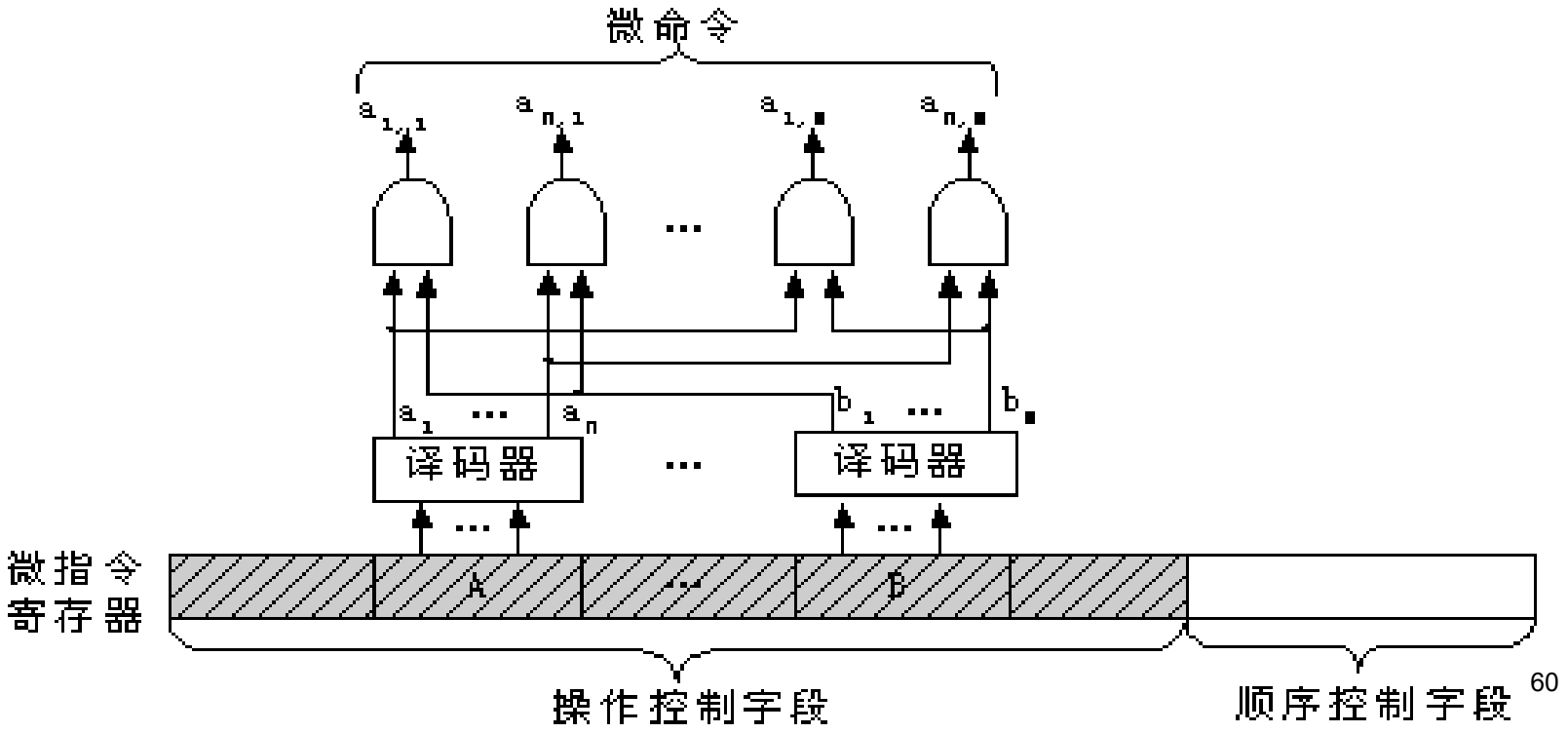
编码表示法：将操作控制字段分为若干个小段，每段内采用最短编码法，段与段之间采用直接控制法。



编码表示法特点：可以避免互斥，使指令字大大缩短，但增加了译码电路，使微程序的执行速度减慢

5.4.2 微程序设计技术

混合表示法：将前两种结合在一起，兼顾两者特点。一个字段的某些编码不能独立地定义某些微命令，而需要与其他字段的编码来联合定义，如例2：F1与RW



5.4.2 微程序设计技术

2、微地址的形成方法

- 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。
- 机器指令的操作码转换成初始微地址的方式主要有两种。
  - 计数器的方式
  - 多路转移的方式

5.4.2 微程序设计技术

2、微地址的形成方法

（1）计数器的方式

- 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。
  - 在微程序控制器中也有一个微程序计数器μPC，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

2、微地址的形成方法

（2）多路转移的方式

- 根据条件转移如图
- 条件：状态条件/测试/微指令中微地址/操作码

【例2】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”微指令后，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时，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。

[例2] 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：

μA5=P3·IR5·T4

μA4=P3·IR4·T4

μA3=P1·IR3·T4

μA2=P1·IR2·T4

μA1=P1·IR1·T4

μA0=P1·IR0·T4+P2·C·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

3、微指令格式

分为两类：水平型微指令和垂直型微指令

（1）水平型微指令

- 水平型微指令是指一次能定义并能并行执行多个微命令的微指令。
- 格式如下

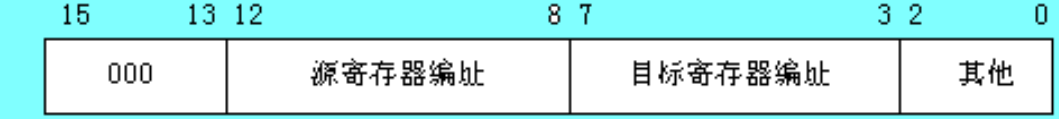
控制字段	判别测试字段	下地址字段

5.4.2 微程序设计技术

3、微指令格式

（2）垂直型微指令：采用编码方式。

- 设置微操作控制字段时，一次只能执行一到二个微命令的微指令称为垂直型微指令。



5.4.2 微程序设计技术

水平型微指令和垂直型微指令的比较

(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

4、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 硬连线控制器

1、基本思想

（1）实现方法

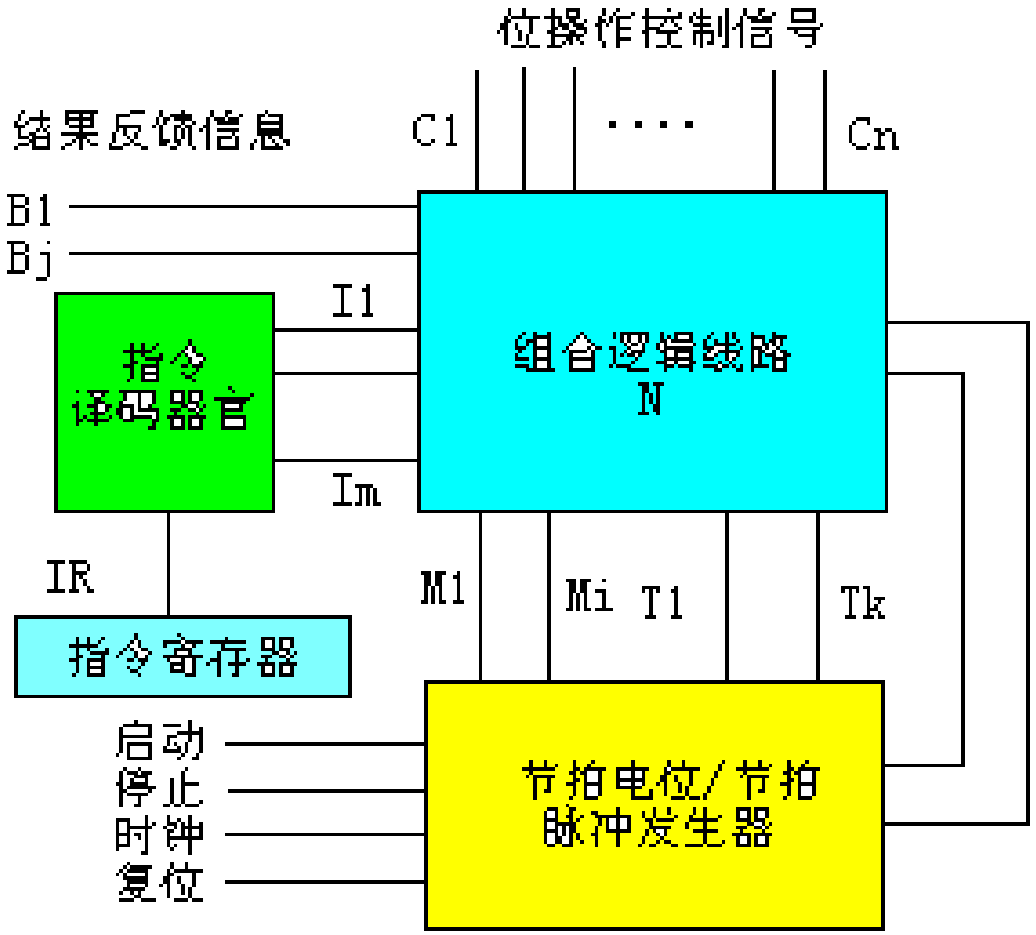
- 通过逻辑电路直接连线而产生的，又称为组合逻辑控制方式

（2）设计目标

- 使用最少元件（复杂的树形网络）
- 速度最高

5.5 硬连线控制器

图**5.28** 硬连线控制器结构方框图



5.5 硬连线控制器

（3）微操作控制信号产生

- 在微程序控制器中，微操作控制信号由微指令产生，并且可以重复使用。
- 在硬联线控制器中，某一微操作控制信号由布尔代数表达式描述的输出函数产生。
- 设计微操作控制信号的方法和过程是，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，并与适当的节拍电位和节拍脉冲组合，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。

5.5 硬连线控制器

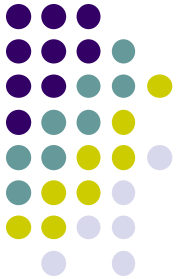
例3根据图5.29，写出以下操作控制信号RD（I）、RD（D）、WE（D）、LDPC、LDIR、LDAR、LDDR、PC+1、LDR2的逻辑表达式。其中每个操作控制信号的含义是：

- RD（I）—指存读命令
- RD（D）—数存读命令
- WE（D）——数存写命令
- LDPC—打入程序计数器
- LDIR—打入指令寄存器
- LDAR—打入数存地址寄存器
- LDDR—打入数据缓冲寄存器
- PC+1—程序计数器加1
- LDR2—打入R1寄存器

5.5 硬连线控制器

[例3] 解：设M_1、M_2、M_3为节拍电位信号，T_1、T_2、T_3、T_4为一个CPU周期中的节拍脉冲信号，MOV、LAD、ADD、STO、JMP分别表示对应机器指令的OP操作码译码输出信号，则有如下逻辑表达式：

$RD(I)=M_1$ (电位信号) , $RD(D)=M_3LAD$ (电位信号) ,
$WE(D)=M_3T_3LDA$ (脉冲信号) , $LDPC=M_1T_4+M_2T_4JMP$,
$LDIR=M_1T_4$, $LDAR=M_2T_4(LAD+STO)$,
$LDDR=M_2T_3(MOV+ADD)+M_3T_3LDA$, $PC+1=M_1$,
$LDR_2=M_2T_4ADD$



5.6 流水CPU

5.6.1并行处理技术

5.6.2流水CPU的结构

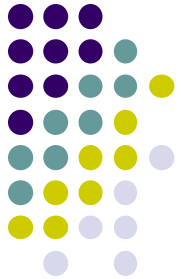
5.6.3流水线中的主要问题

5.6.4 奔腾 CPU

5.6.1并行处理技术

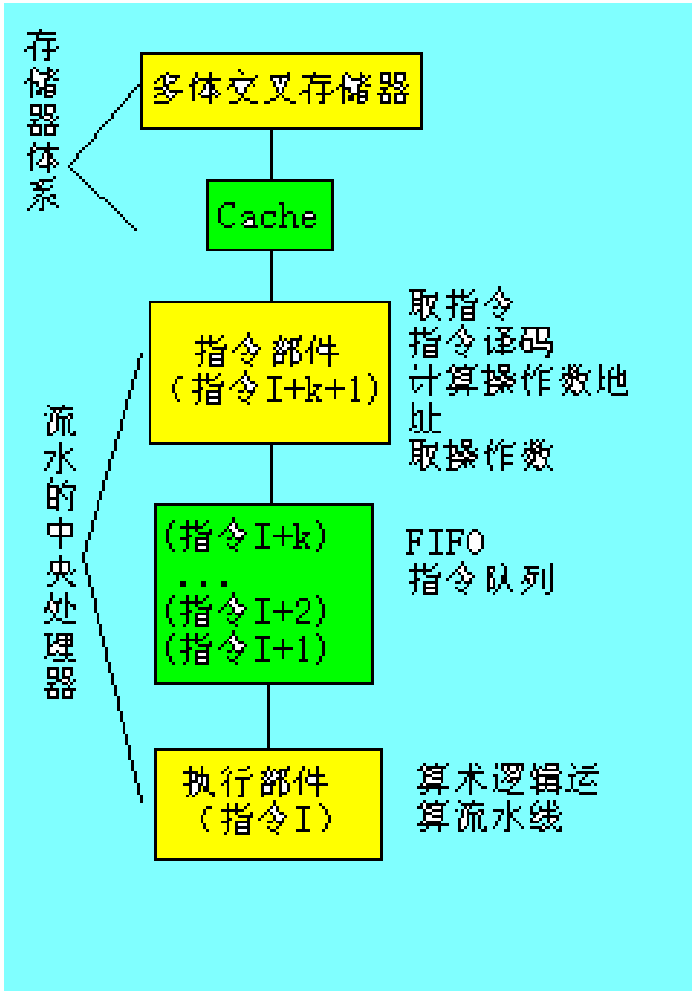
- 并行性（Parrelism）概念
  - 问题中具有可以同时进行运算或操作的特性
  - 例：在相同时延的条件下，用n位运算器进行n位并行运算速度几乎是一位运算器进行n位串行运算的n倍（狭义）
- （广义）含义
  - 只要在同一时刻（同时性）或在同一时间间隔内（并发性）完成两种或两种以上性质相同或不同的工作，他们在时间上相互重叠，都体现了并行性

5.6.1并行处理技术



- 三种形式
  - 时间并行（重叠）：让多个处理过程在时间上相互错开，轮流使用同一套硬件设备的各个部件，以加快硬件周转而赢得速度，实现方式就是采用流水处理部件
  - 空间并行（资源重复）：以数量取胜
    - 它能真正的体现同时性
    - LSI和VLSI为其提供了技术保证
  - 时间+空间并行
    - Pentium中采用了超标量流水线技术

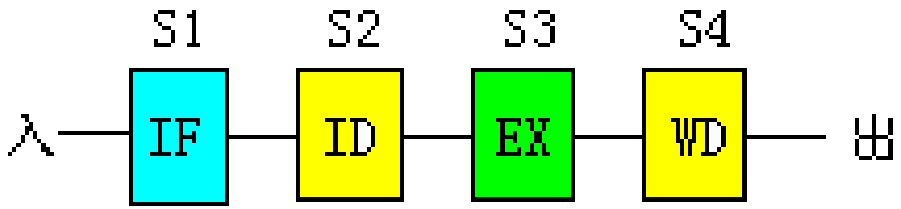
5.6.2流水CPU的结构



- 流水计算机的系统组成
  - 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：可以有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。

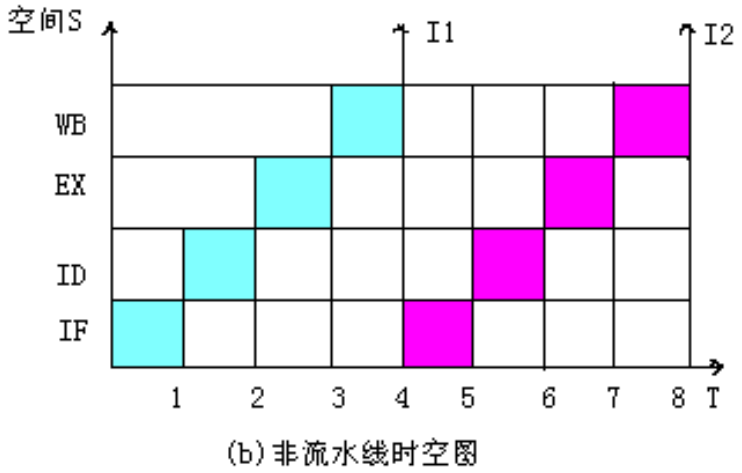
5.6.2流水CPU的结构

- 流水线CPU时空图
  - IF（Instruction Fetch取指）
  - ID（Instruction Decode指令译码）
  - EX（Execution执行）



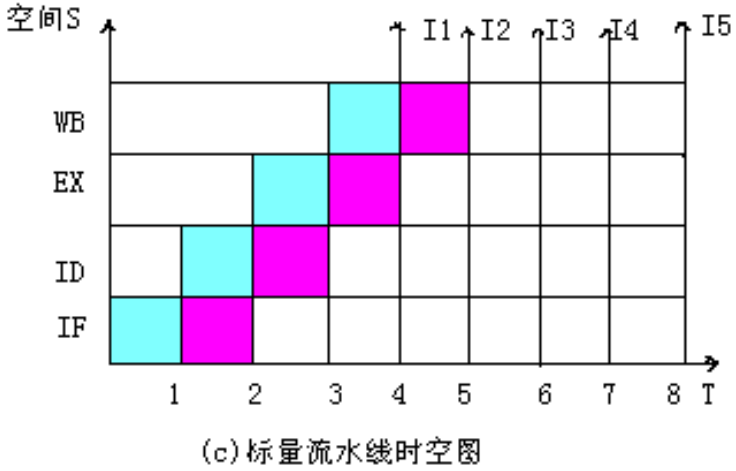
(a)一个指令流水线过程段

5.6.2流水CPU的结构



(b)非流水线时空图

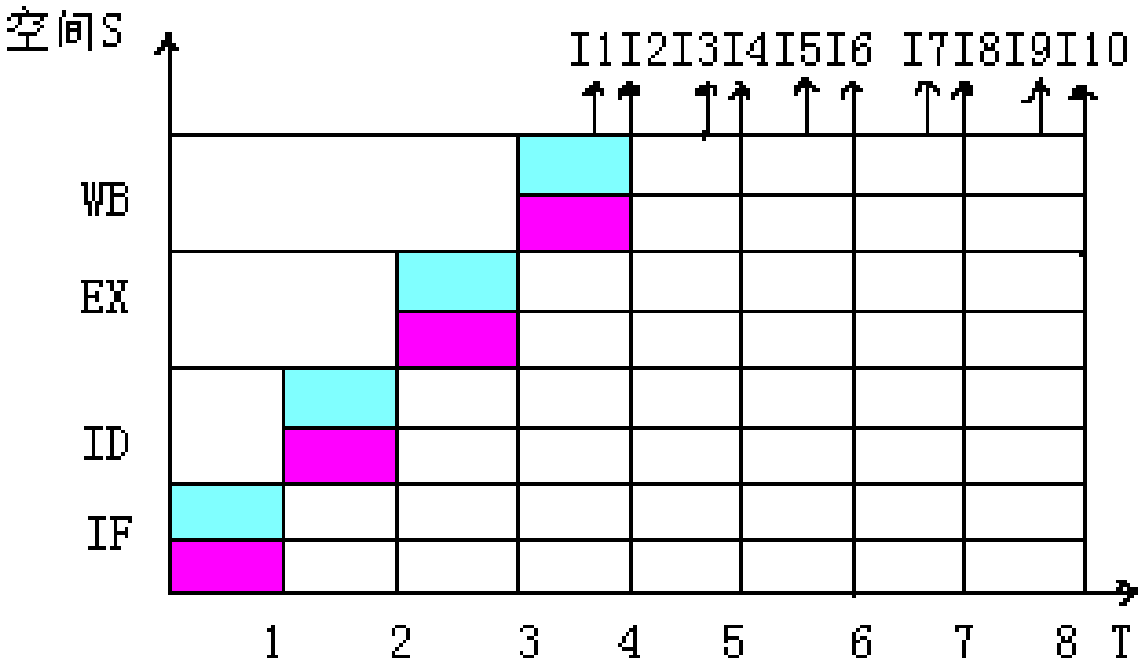
非流水CPU



(c)标量流水线时空图

流水CPU

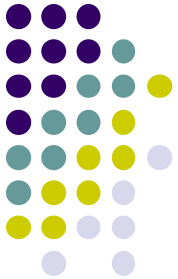
5.6.2流水CPU的结构



(d)超标量流水线时空图

- 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

5.6.2流水CPU的结构



- 流水线（Pipelining）的分类
  - 按级别分为
    - 指令流水线
    - 算术流水线
    - 处理机流水线（宏流水线）

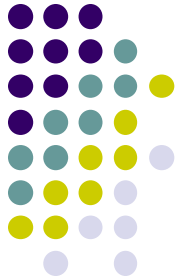
5.6.3流水线中的主要问题

- 三种相关冲突：资源相关、数据相关、控制相关

- 资源相关：多条指令进入流水线后在同一时钟周期内争用同一功能部件。
  - 解决办法：后边指令拖一拍再推进；增设一个功能部件

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流水线中的主要问题



- 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）

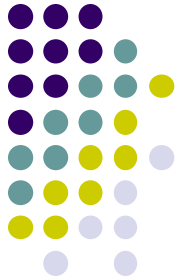
5.6.3流水线中的主要问题

例：两条指令发生数据相关冲突RAW(Read After Write)

```
ADD R1,    R2,    R3              R2+R3-->R1
SUB        R4,    R1,    R5              R1-R5-->R4
AND R6,    R1,    R7              R1^R7-->R6
```

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	

5.6.3流水线中的主要问题



- 控制相关
  - 引起原因：转移指令
  - 解决办法：延迟转移法，转移预测法

【例4】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。

```
（1）I1     ADD R1，R2，R3        ；（R2）+（R3）->R1
     I2     SUB R4，R1，R5        ；（R1）-（R5）->R4
（2）I3     STO M（x），R3        ；（R3）->M(x)，M(x)是存储器单元
     I4     ADD R3，R4，R5        ；（R4）+（R5）->R3
（3）I5     MUL R3，R1，R2        ；（R1）×（R2）->R3
     I6     ADD R3，R4，R5        ；（R4）+（R5）->R3
```

解：

- 第（1）组指令中，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。
- 第（2）组指令中，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。
- 第（3）组指令中，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。

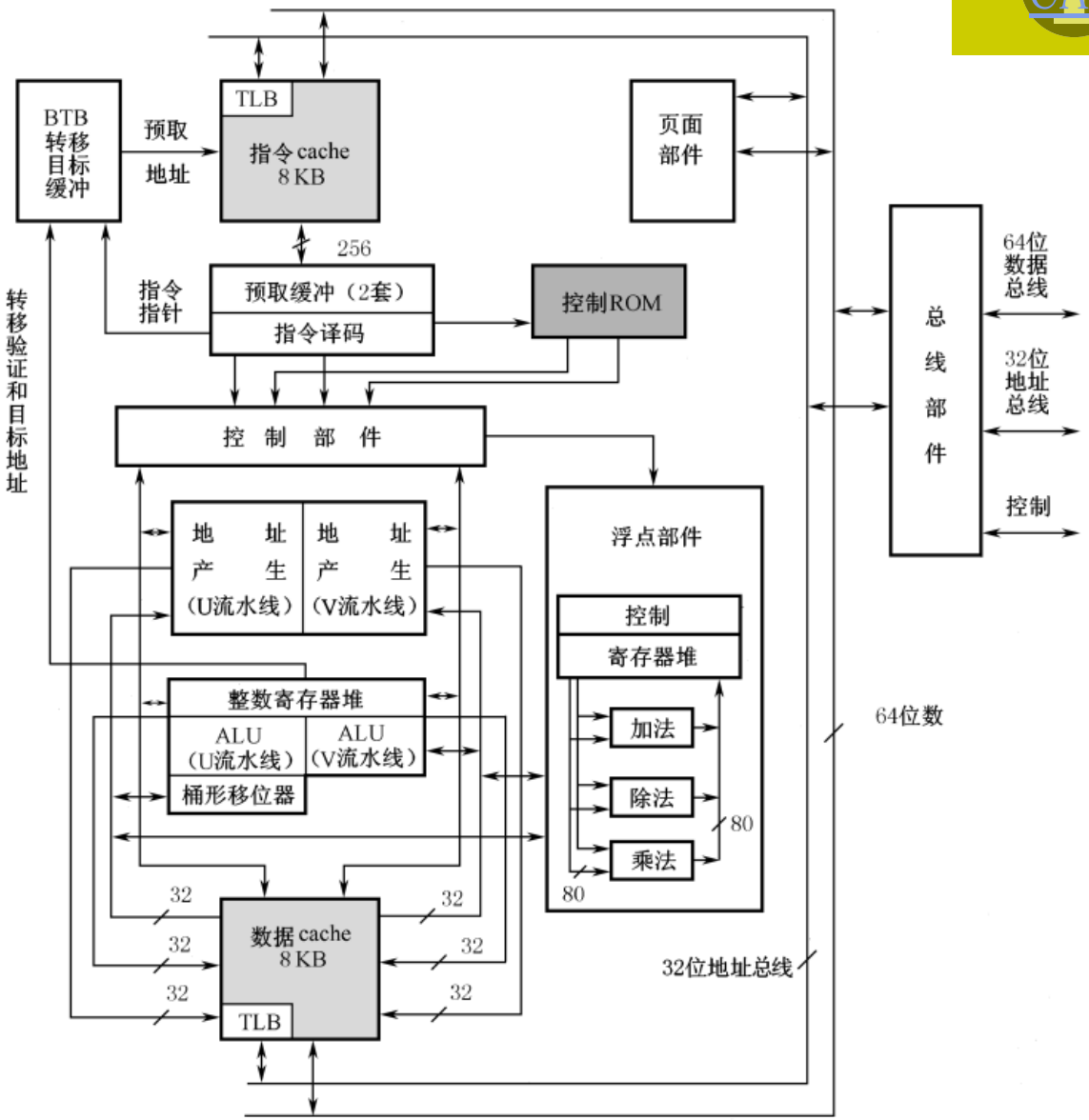
5.6.4 奔腾CPU

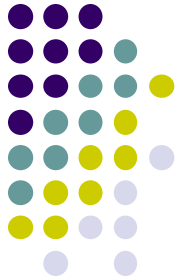
- Pentium CPU （第一代）
  - 1989年初0.8um工艺，310万晶体管
  - 有60M和66MHz外频两种版本
  - 5V电压，功耗20W
  - 超标量流水线结构
    - 486有一条流水线
    - Pentium有U和V两条指令流水线
      - U流水线可以执行所有的整数和浮点指令
      - V流水线可以执行简单的整数和FXCH浮点指令
  - 双重分离式Cache，减少了等待和搬移数据时间
  - 32位CPU，外部数据总线宽度为64位，外部地址总线宽度为36位

5.6.4 奔腾CPU

- 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
- Pentium结构图
  - MESI（Modified　Exclusion　Share Invalid）
  - BTB（Branch Target Buffer）
  - TLB（Translation Lookaside Buffer）

图5.34 Pentium CPU结构框图





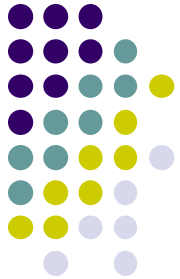
5.7 RISC CPU

5.7.1 RISC机器的特点

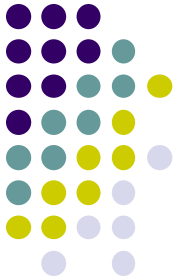
5.7.2 RISC CPU实例

5.7.3 动态流水线调度

5.7.1 RISC机器的特点



- 特点
  - （采用流水线技术）
  - 简单而统一格式的指令译码；
  - 大部分指令可以单周期执行
  - 只有LOAD/STORE可以访问存储器
  - 简单的寻址方式
  - 采用延迟转移技术
  - 采用LOAD延迟技术
  - 三地址指令格式
  - 较多的寄存器
  - 对称的指令格式
  - 其他



5.7.2 RISC CPU实例

- 实例 MC88110
  - CPU结构框图（见下图）
    - 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

MC88110 CPU结构框图

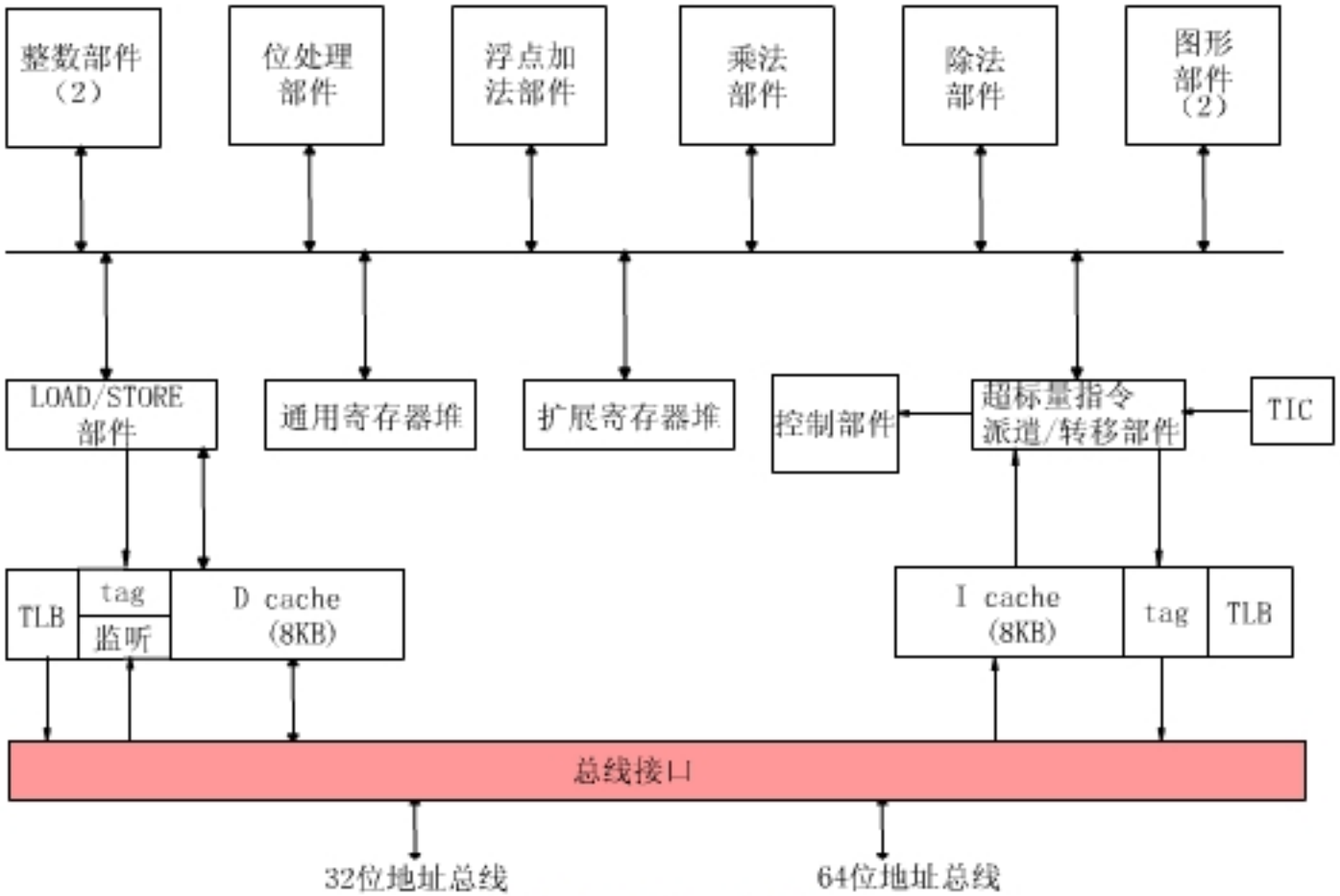
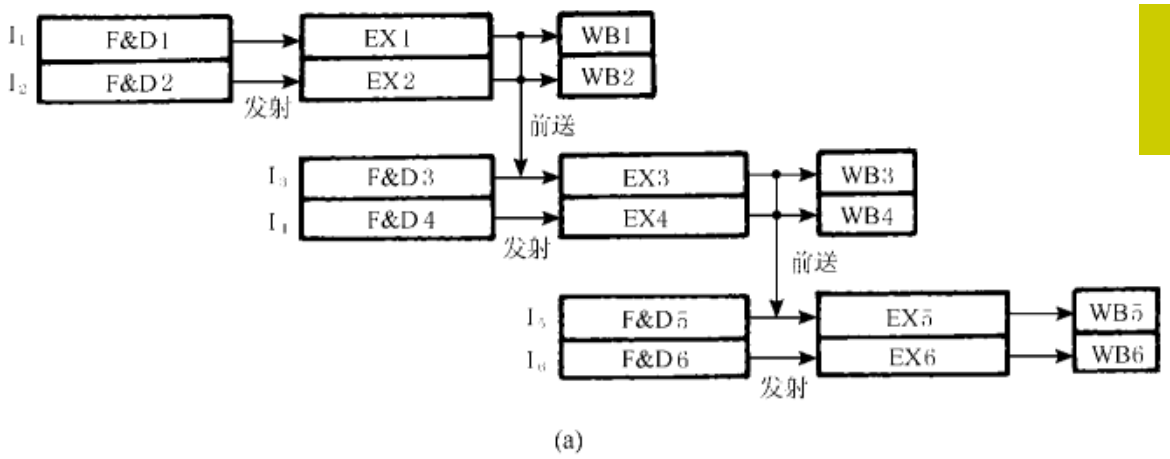


图5.41 88110处理器结构框图

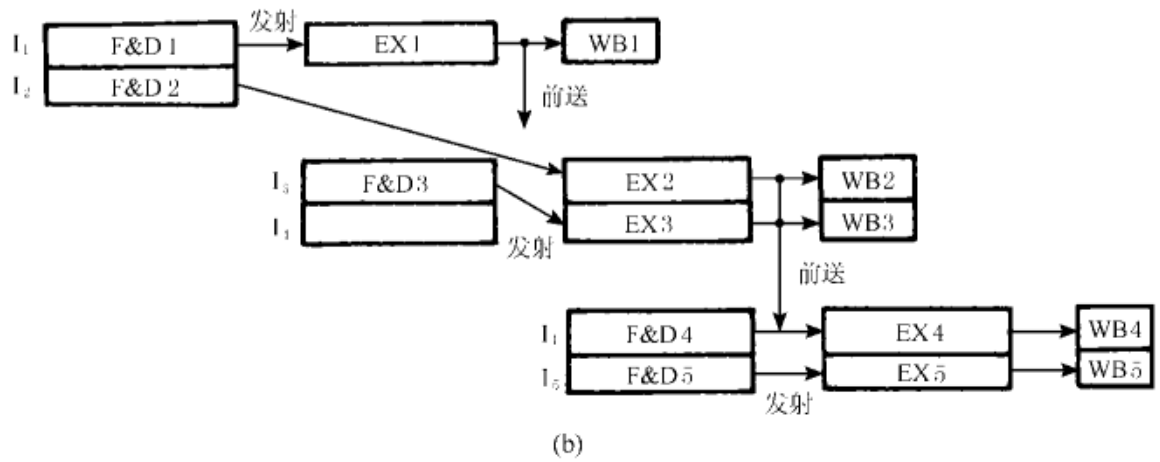
MC88110的指令流水线

- 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

- 指令动态调度策略
  - 按序发射
    - 取两条指令，配对发送，一个周期可以有两条指令执行完毕
    - 如下图：



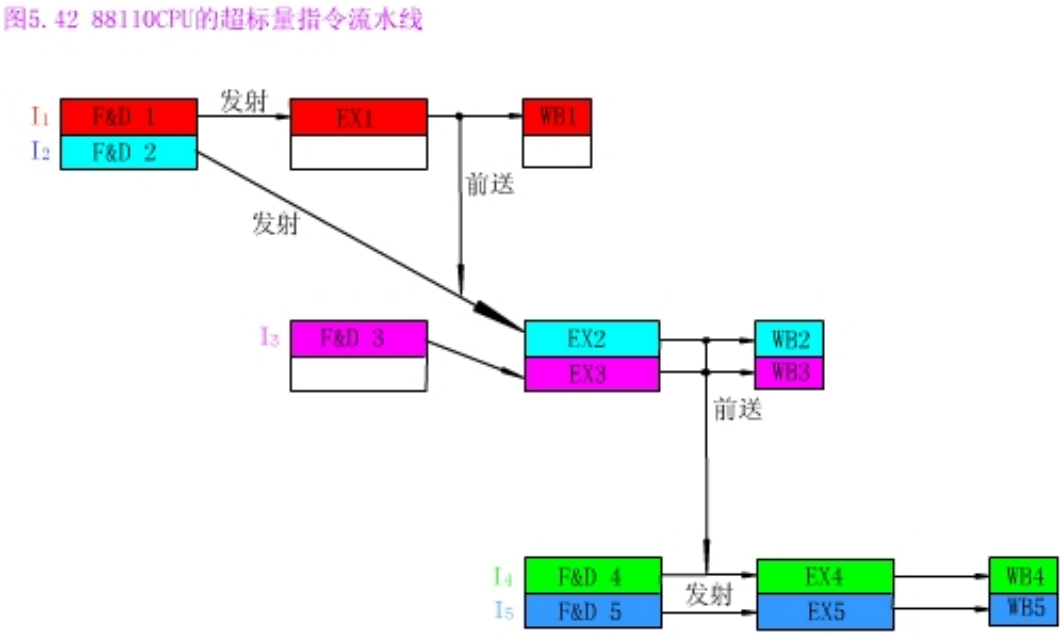
(a)



(b)

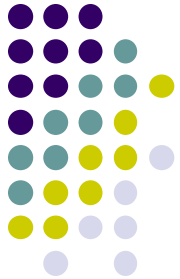
指令动态调度策略

- 第一条指令由于资源相关或数据相关,则这两条指令都不发射
- 若第一条指令能发射,第二条不能发射,只发射第1条指令到EX段,第二条指令等待并新取一条指令与之配对等待发射



超标量流水线指令配对情况

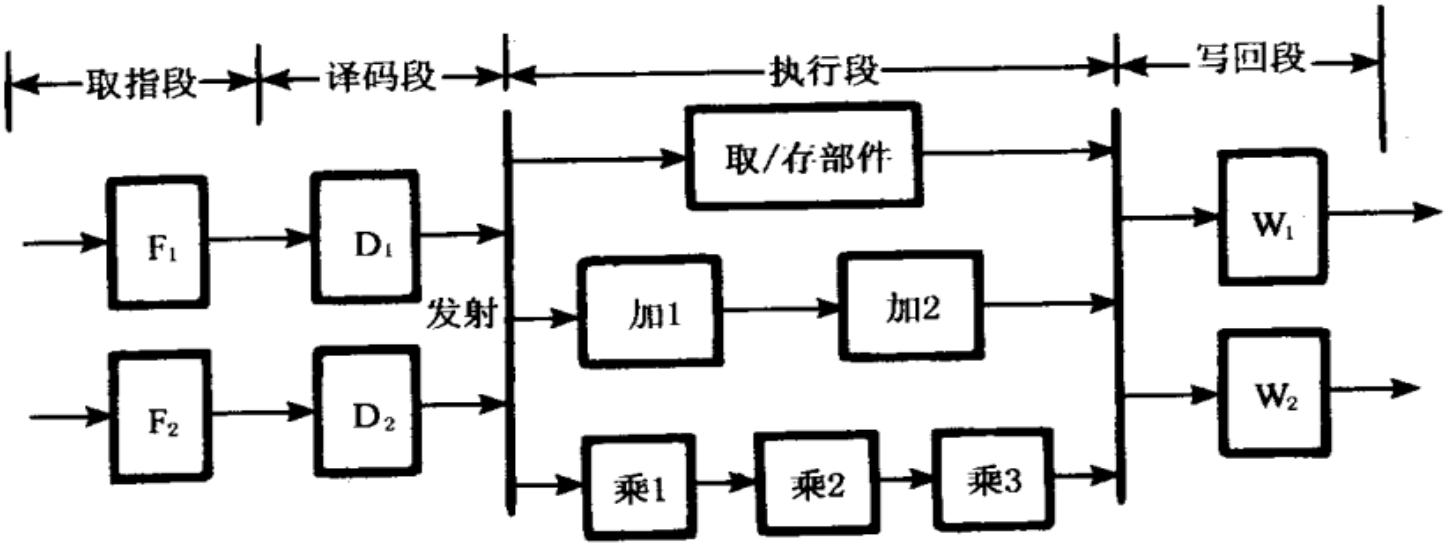
5.7.2 RISC CPU实例



- 几个问题：
  - 怎样判断能否发射呢?
    - 可以采用计分牌的方法
  - 如何保证按序完成?
    - FIFO指令队列
  - 如何对待控制相关（转移指令）？
    - 采用延迟转移法和目标指令cache法

5.7.2 RISC CPU实例

- 例5 超标量流水线结构如下



（a）超标量流水模型结构

5.7.2 RISC CPU实例

```
I1    LDA    R1,    A
I2    ADD    R2,    R1
I3    ADD    R3,    R4
I4    MUL    R4,    R5
I5    LDA    R6,    B
I6    MUL    R6,    R7
```

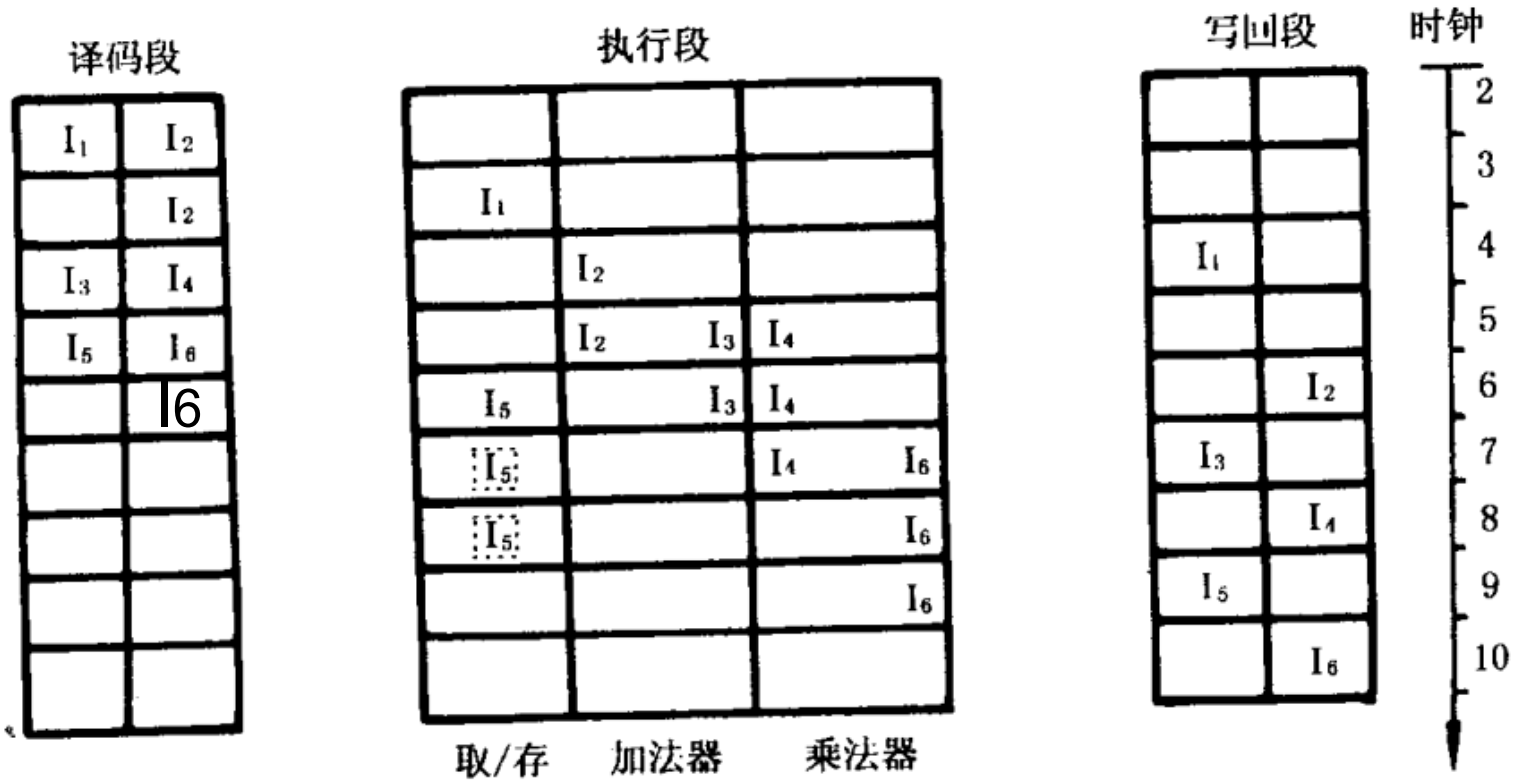
RAW

WAR

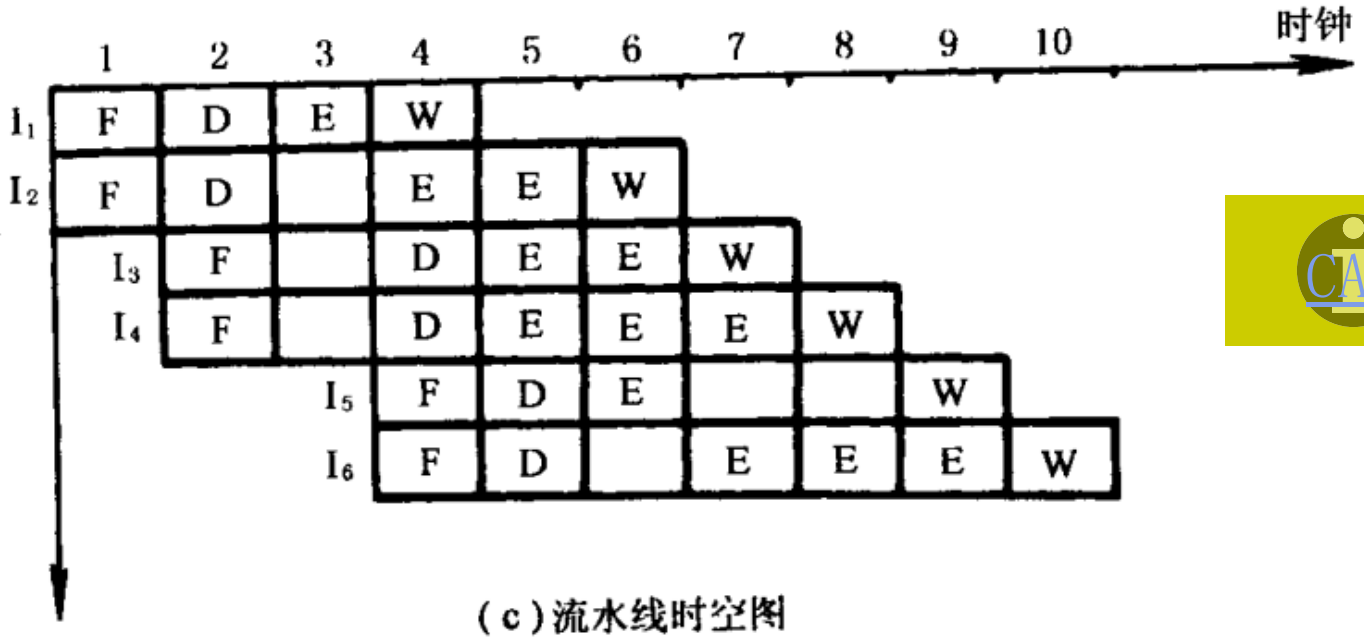
WAW

- 画出按序完成各段推进情况图
- 画出按序完成流水线时空图

5.7.2 RISC CPU实例



5.7.2 RISC CPU实例



（c）流水线时空图

5.7.3 动态流水线调度

动态流水线调度是指对指令重新排序以避免处理器阻塞的硬件支持。

流水线3个主要单元：
1. 指令发射单元（一个）
2. 功能单元（多个）
3. 指令完成单元（一个）

图**5.38** 动态流水线调度模型

第五章小结

- CPU是计算机的中央处理部件，具有指令控制、操作控制、时间控制、数据加工等基本功能。早期的CPU由运算器和控制器两大部分组成。随着高密度集成电路技术的发展，当今的CPU芯片变成运算器、cache和控制器三大部分，其中还包括浮点运算器、存储管理部件等。
- CPU中至少要有如下六类寄存器：指令寄存器、程序计数器、地址寄存器、数据缓冲寄存器、通用寄存器、状态条件寄存器。CPU从存储器取出一条指令并执行这条指令的时间和称为指令周期。CISC中，由于各种指令的操作功能不同，各种指令的指令周期是不尽相同的。划分指令周期，是设计操作控制器的重要依据。

第五章小结

- RISC中，由于流水执行，大部分指令在一个机器周期完成。时序信号产生器提供CPU周期（也称机器周期）所需的时序信号。操作控制器利用这些时序信号进行定时，有条不紊地取出一条指令并执行这条指令。
- 微程序设计技术是利用软件方法设计操作控制器的一门技术，具有规整性、灵活性、可维护性等一系列优点，因而在计算机设计中得到了广泛应用。但是随着ULSI技术的发展和对机器速度的要求，硬连线逻辑设计思想又得到了重视。
- 硬连线控制器的基本思想是：某一微操作控制信号是指令操作码译码输出、时序信号和状态条件信号的逻辑函数，即用布尔代数写出逻辑表达式，然后用门电路、触发器等器件实现。

第五章小结

- 从简单到复杂，举出一个CPU模型以及Intel8088、IBM370CPU等传统CPU的结构，目的在于使读者由浅入深地理解教学内容，也使读者了解了计算机技术的发展历程。这对于建立整机概念是十分重要的。不论微型机还是超级计算机，并行处理技术已成为计算机技术发展的主流。
- 并行处理技术可贯穿于信息加工的各个步骤和阶段。概括起来，主要有三种形式：①时间并行；②空间并行；③时间并行+空间并行。
- 流水CPU是以时间并行性为原理构造的处理机，是一种非常经济而实用的并行技术。目前的高性能微处理机几乎无一例外地使用了流水技术。流水技术中的主要问题是资源相关、数据相关和控制相关，为此需要采取相应的技术对策，才能保证流水线畅通而不断流。

第五章小结

- RISC CPU是继承CISC的成功技术，并在克服CISC机器缺点的基础上发展起来的。RISC机器的三个基本要素是：①一个有限的简单指令集，②CPU配备大量的通用寄存器，③强调指令流水线的优化。RISC机器一定是流水CPU，但流水CPU不一定是RISC机器。如奔腾CPU是流水CPU，但奔腾机是CISC机器。
- 多媒体CPU是带有MMX技术的处理器。MMX是一种多媒体扩展结构技术，特别适合于图像数据处理，极大地提高了计算机在多媒体和通信应用方面的功能。多媒体CPU以新一代奔腾CPU为代表，开始采用单指令流多数据流的新型结构。