东南大学计算机学院

计算机系统组成

主讲教师： 徐造林

第7章 中央处理器（CPU）

- CPU概念：控制并执行指令的部件，与其它功能部件进行信息交换，并控制它们的操作。
  - 包括控制器与运算器

- 中央处理器（CPU ）是整个计算机的核心。本章着重讨论CPU的功能与组成、控制器的工作原理与实现方法、微程序控制原理、基本控制单元的设计以及流水线技术。

7.1 CPU的结构与功能

7.1.1 CPU的组成与操作

- CPU的组成

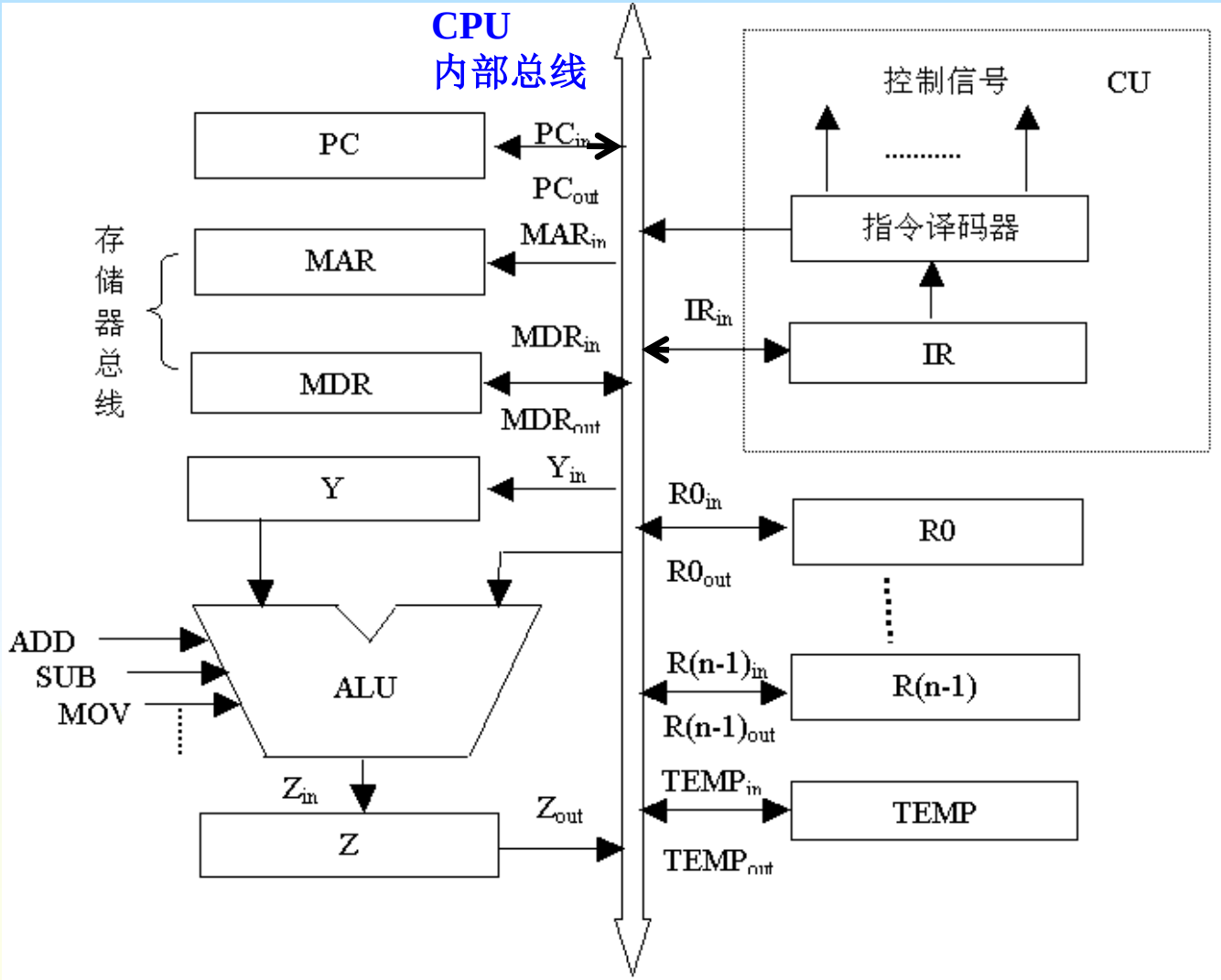


图7.1 单总线CPU内部结构

- 几个名词:

*指令控制：程序执行过程＝指令执行过程+循环

*操作控制：指令控制及指令约定功能实现的控制信号产生

*时间控制：操作控制的信号时序实现

*数据加工：指令约定的算术及逻辑运算功能实现

*中断处理：I/O操作的中断方式实现

- CPU的四种基本操作：
  - 存储器读：读取某一主存单元的内容，并将其装入某一个CPU寄存器；
  - 存储器写：把一个数据字从某一CPU寄存器存入给定的主存单元中；
  - 把一个数据字从某一CPU寄存器送到另一个寄存器或者ALU；
  - 进行一个算术运算或逻辑运算，将结果送入某一CPU寄存器或存储器 。

1. 存储器读

▲ CPU向主存发送地址和读控制信号；

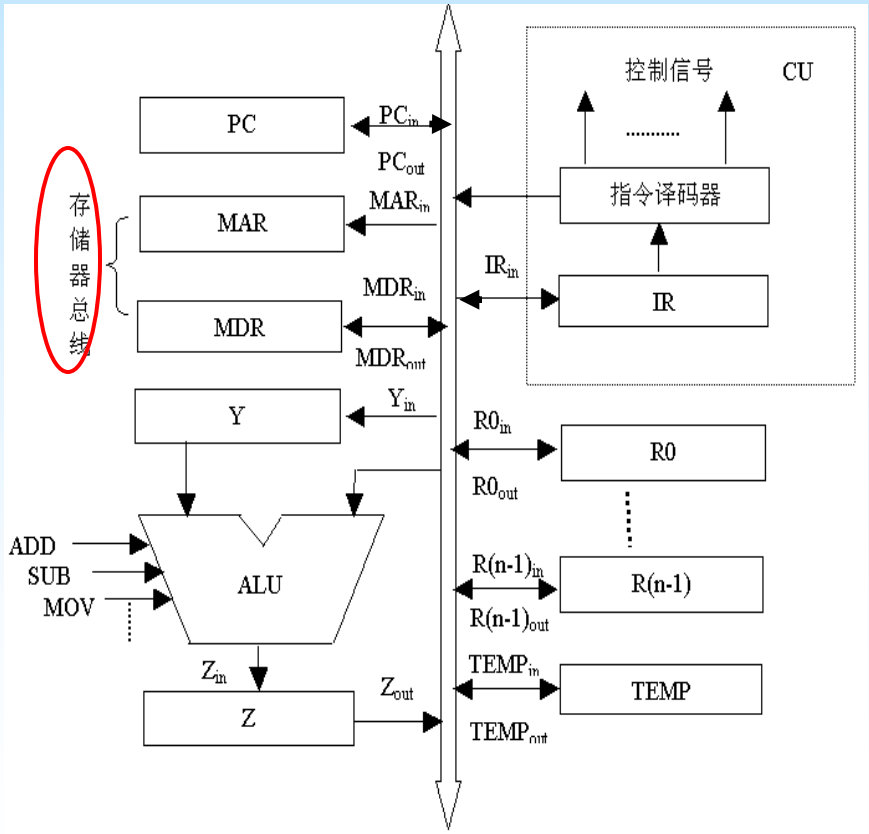
例：从内存读取一字((R1)→R2)

t1: MAR←[R1]

t2: Read

t3: WMFC ； 等待存储器操作完成信号

t4: R2←[MDR]



2. 存储器写

- 主存地址装入MAR，数据字装入MDR，然后向存储器发一个“写”信号。

例：把一个字写入主存，数据字放在寄存器R2中，主存单元地址放在R1中

t1: MAR←[R1]

t2: MDR←[R2]

t3: Write

t4: WMFC

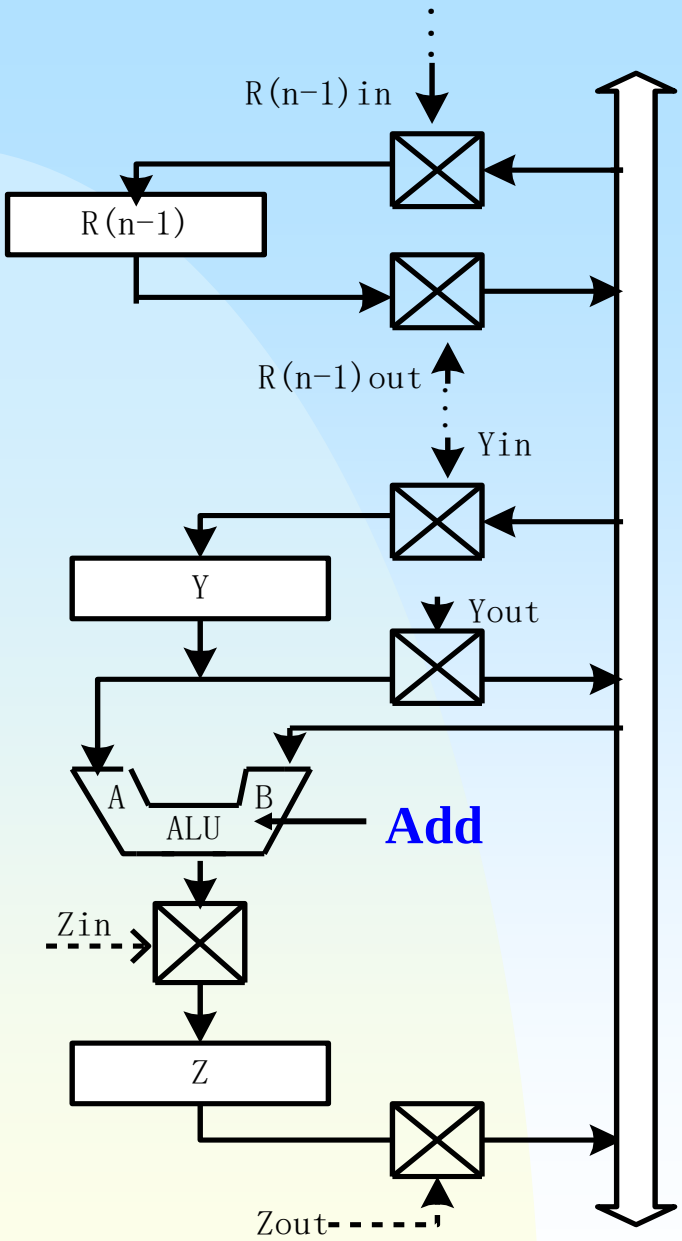


图7.2　寄存器的输入门和输出门

3. 通用寄存器之间传送数据

R1out，R4in

4. 完成算术、逻辑运算

t1: R1out，Yin

t2: R2out，ADD，Zin

t3: Zout，R3in

- 寄存器的1位输入、输出门的组成

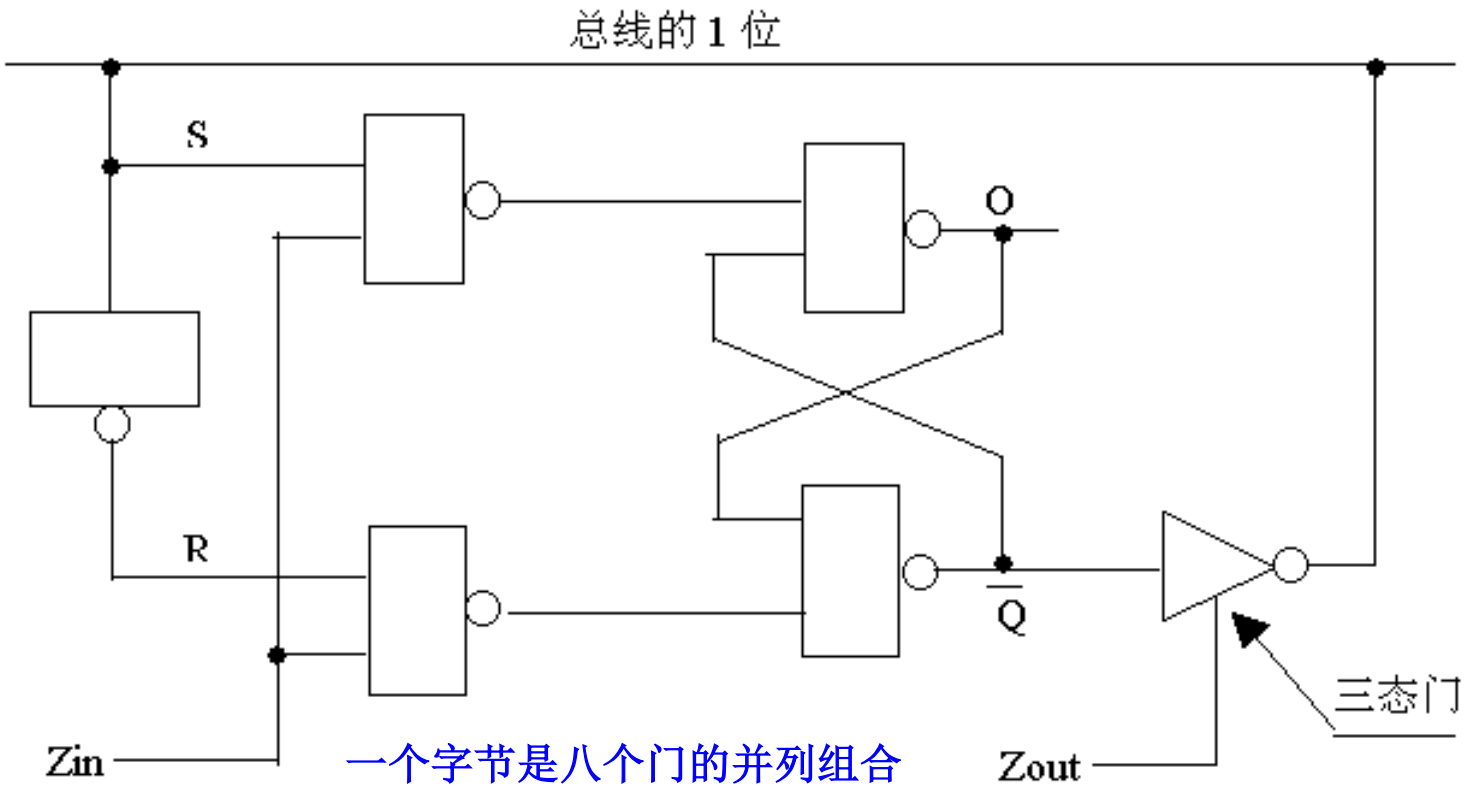


图7.3　寄存器的 1 位输入、输出门的组成

CPU内部数据传送时序

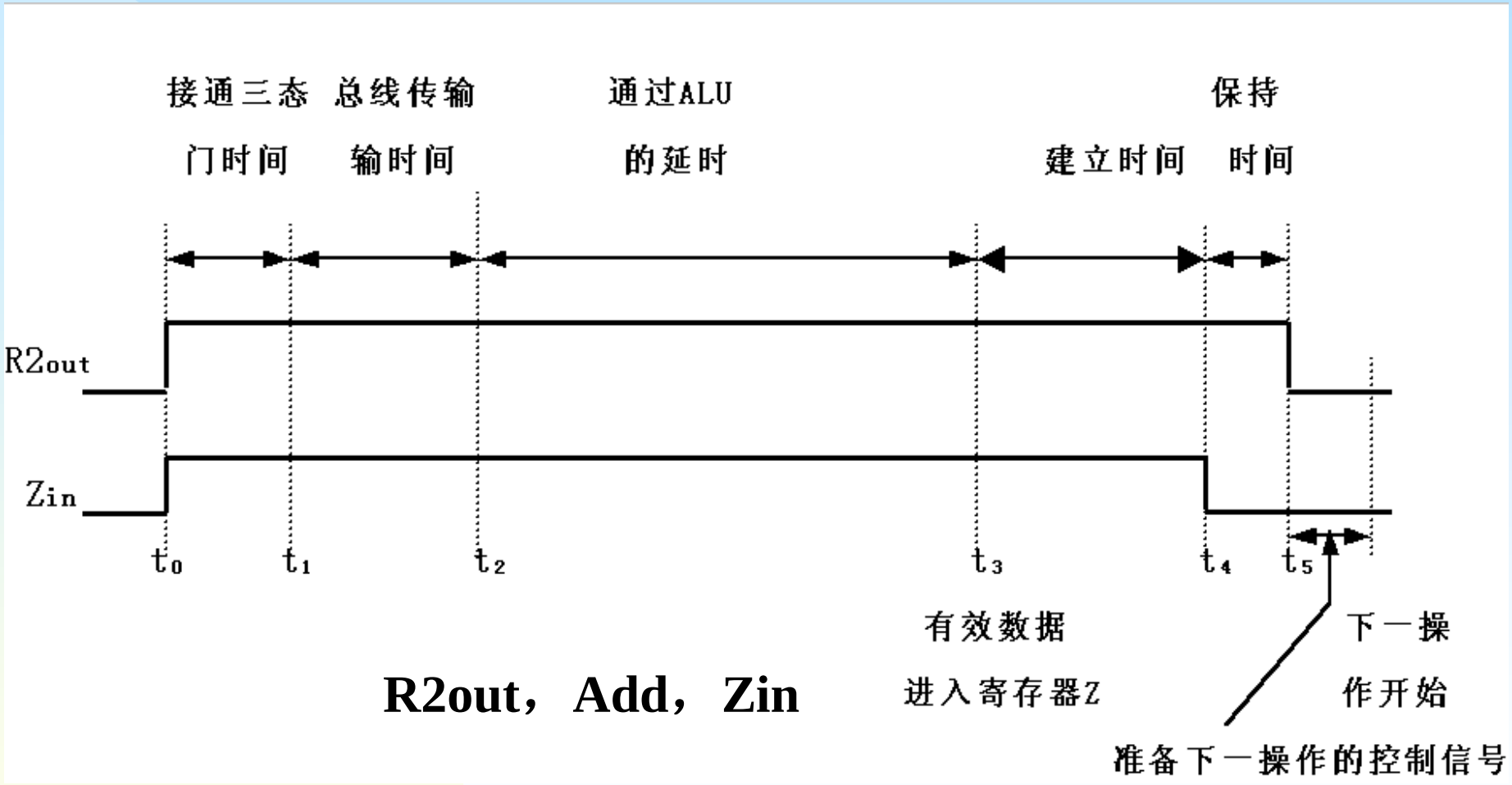


图7.4 加法操作期间控制信号的时序

7.1.2 寄存器组织

- 寄存器在CPU中用来保存运算和控制过程中的中间结果、最终结果以及控制、状态信息。

◆ CPU寄存器组的设计要考虑的主要原则：

1. 使用完全通用的寄存器还是完全专用的寄存器或者是两者结合；
2. 寄存器数量；
3. 寄存器长度；地址寄存器、数据寄存器长度的选取。

有用户可见寄存器和控制/状态寄存器两大类。

*用户可见寄存器：　--存放地址及数据

数据REG—存放操作数，长度＝CPU字长；

累加REG(AC)—同时用作ALU源和目标操作数的REG；

地址REG—存放操作数地址或指令地址，

长度＝程序逻辑地址的段号或段内地址位数；

通用REG—可用作数据REG和地址REG的REG；

→长度相同

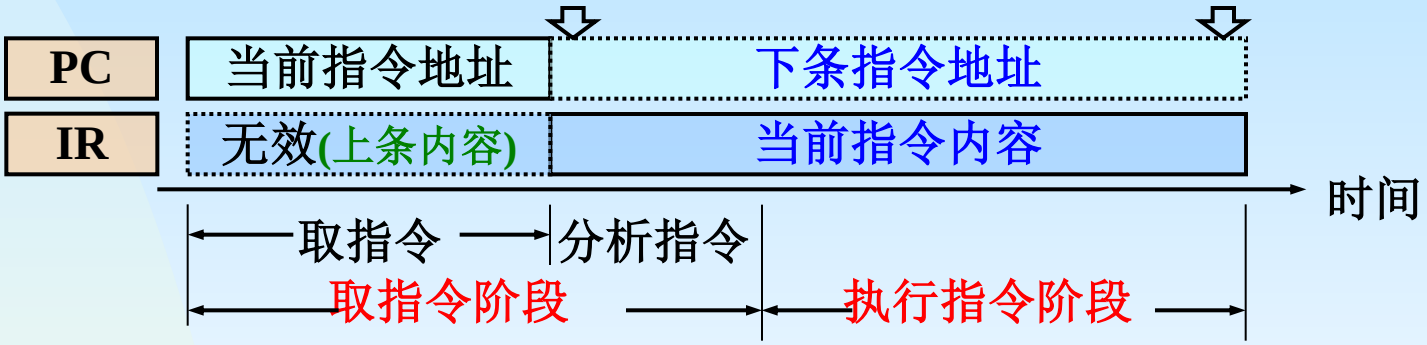
条件码REG—存放指令执行结果的状态(如C/V/Z/N)，可作为转移指令的测试条件

*控制/状态寄存器： --控制CPU的操作或运算

PC—存放指令地址，作循环变量使用，

通常具有计数功能； ←顺序型指令占绝大多数

IR—存放当前指令内容；



MAR—存放CPU所访问存储单元或I/O设备的地址；

MDR—存放CPU欲写出或已读入的信息(数据或指令)；

程序状态字REG(PSW)—存放程序执行状态；

状态REG—包含条件码和程序状态字的REG；

其他REG—如段REG、系统模式REG等

7.1.3 指令周期

- 指令周期：指一条指令的完整执行时间；包含取指令周期和执行周期。

◆ 四种典型指令的指令周期

1. 非访存指令；
2. 直接访存指令；
3. 间接访存指令；
4. 程序控制指令（决定于目的地址的寻址方式）。

7.2 控制器的功能和组成

7.2.1 功能和组成

- CPU的控制流程
- 控制器完成的基本任务
  - 排序：安排相应的微操作序列；
  - 执行：控制器发出各种微操作信号。

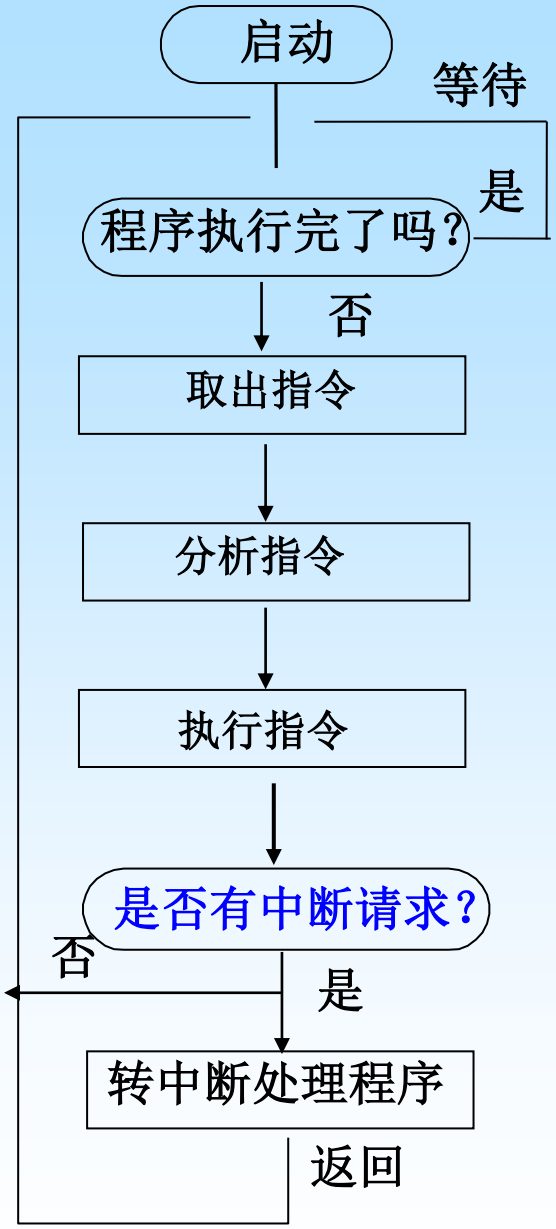
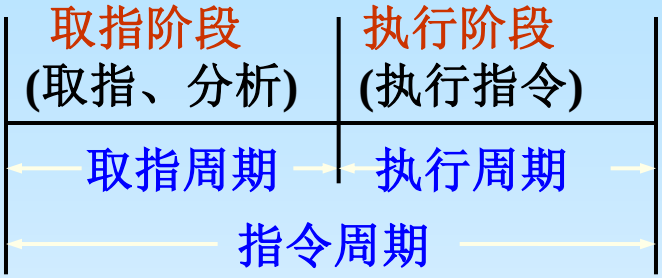


图7.5 CPU控制流程

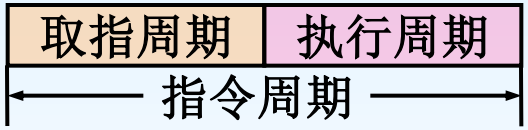
*指令周期：CPU取出并执行一条指令的时间。



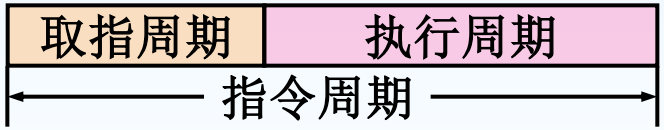
*指令周期的特征：

不同指令类型及寻址方式的指令周期可能不同。

加法指令：(直接寻址)



乘法指令：(直接寻址)



加法指令：(间接寻址)

控制器功能部件组成

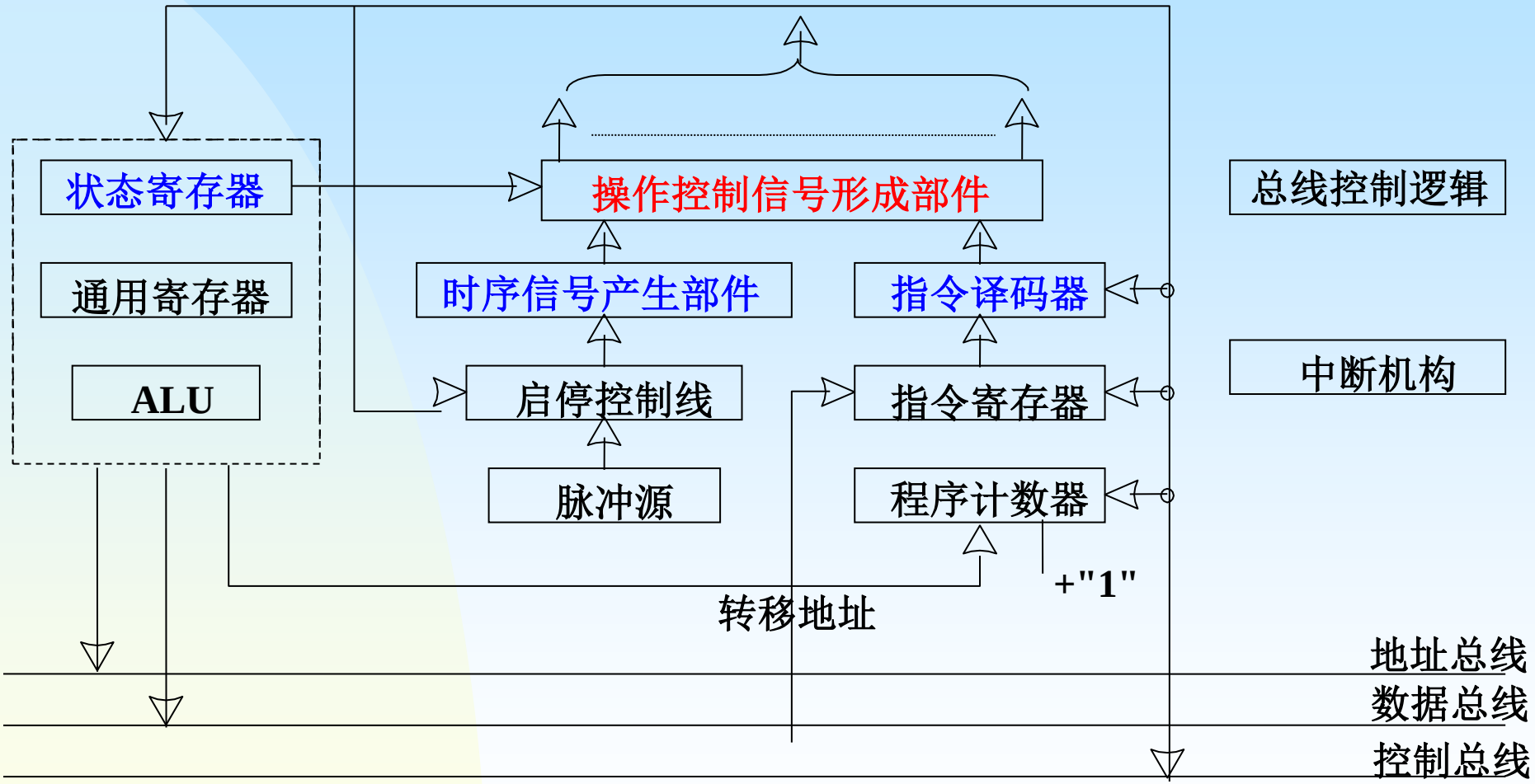


图7.6 控制器组成框图

7.2.2 CPU时序系统及控制方式

◆ 三级时序系统

1. 机器周期

- 机器周期（CPU周期）：指令执行过程中相对独立的阶段；常把一个指令周期划分为若干个机器周期；
- 指令的执行过程分为取指令、地址计算及取操作数、执行指令三个基本的工作周期；
- 以主存的工作周期为基础来确定CPU周期；
- 当机器运行在不同的机器周期时，其对应的标志触发器就被置1。

2. 节拍

- 把一个机器周期分为若干个相等的时间段，每一时间段对应一个电位信号，称为节拍电位。

▲节拍选取的3种方法：

1）统一节拍：节拍宽也以完成最繁杂的操作为标准；

2）分散节拍：按照每一个机器周期的实际需求来安排节拍数；

3）延长节拍：对于复杂机器周期，若基本节拍无法完成周期内的所有操作，则可以延长一到两个节拍。

3. 工作脉冲

- 在一个节拍里，某些操作需要同步定时脉冲以保证触发器可靠稳定地翻转。

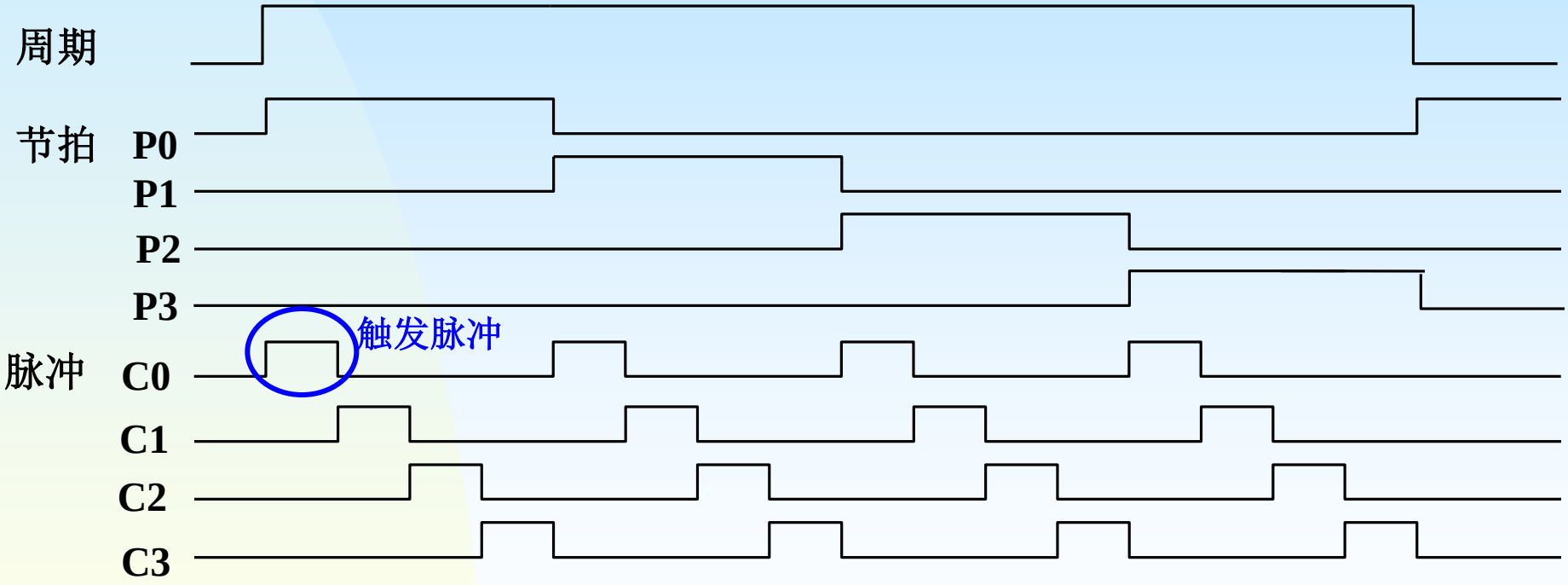
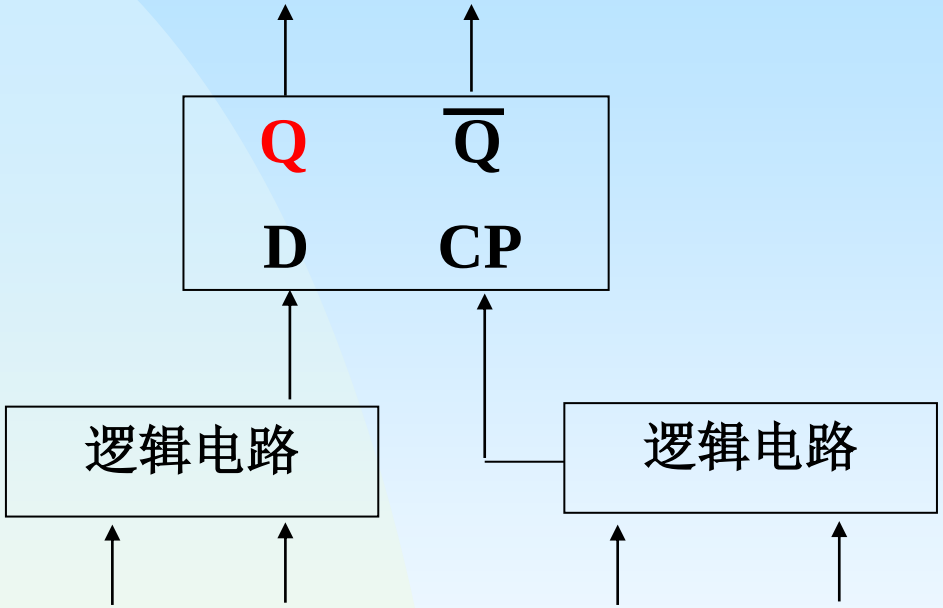
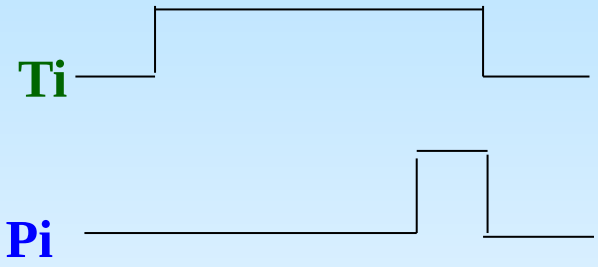


图7.7　三级时序系统





信息 节拍电位Ti　　打入条件 工作脉冲Pi

图7.8 节拍和脉冲的配合

- 时序系统中各种时序信号的控制方式：同步控制、异步控制和联合控制。

1. 同步控制方式

- 指令执行过程中每个操作的完成，都由确定的具有基准时标的时序信号来控制；
- 以最复杂指令和最复杂操作的时间作为统一的时序标准；
- 设计简单，实现容易；影响简单指令执行的速度。

2. 异步控制方式

- 按每条指令、每个操作的需要而占用时间的一种控制方式；
- 没有集中统一的时序信号形成和控制部件；
- 各操作之间是用“结束”或“就绪”—“起始”的方式衔接起来的；
- 机器效率高；但实现起来非常复杂。

3. 联合控制方式

- 同步和异步控制方式的结合；
- 实行部分统一、部分区别对待的方式；
- 在功能部件内部基本采用同步控制方式；
- 在功能部件之间采用异步控制方式，如CPU和主存、外设等交换数据时。

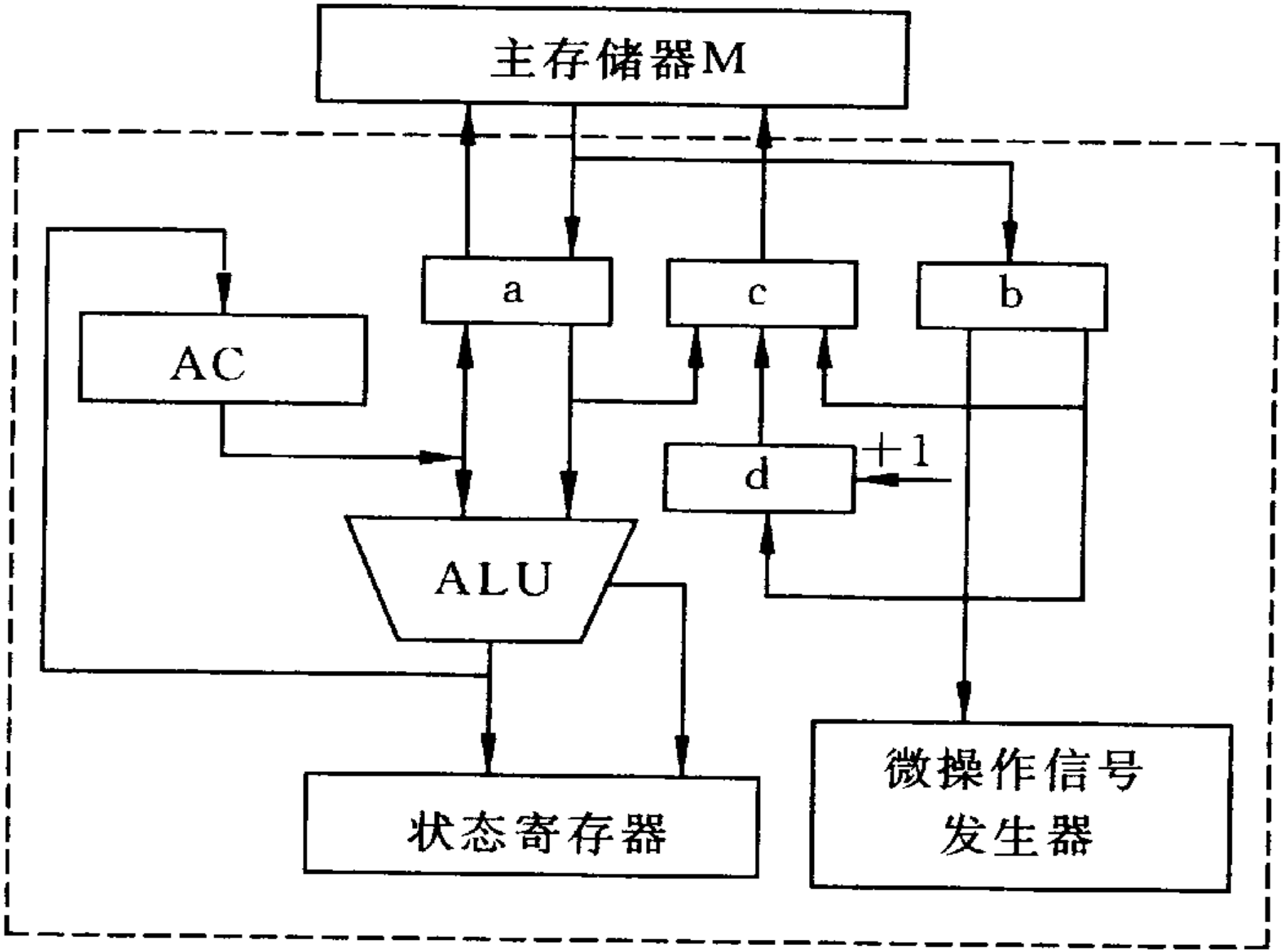
◆ 现代计算机大多采用同步控制方式或联合控制方式。

例1 CPU结构如后图，其中有一个累加寄存器AC、一个状态寄存器；各部分之间的连线表示数据通路，箭头表示信息转送方向。要求：

（1）表明图中a，b，c，d四个寄存器的名称。

（2）简述指令从主存储器取到控制器的数据通路。

（3）简述数据在运算器和主存之间进行存/取访问的数据通路。


主存储器M
a
c
b
AC
d
+1
ALU
状态寄存器
微操作信号
发生器

解：

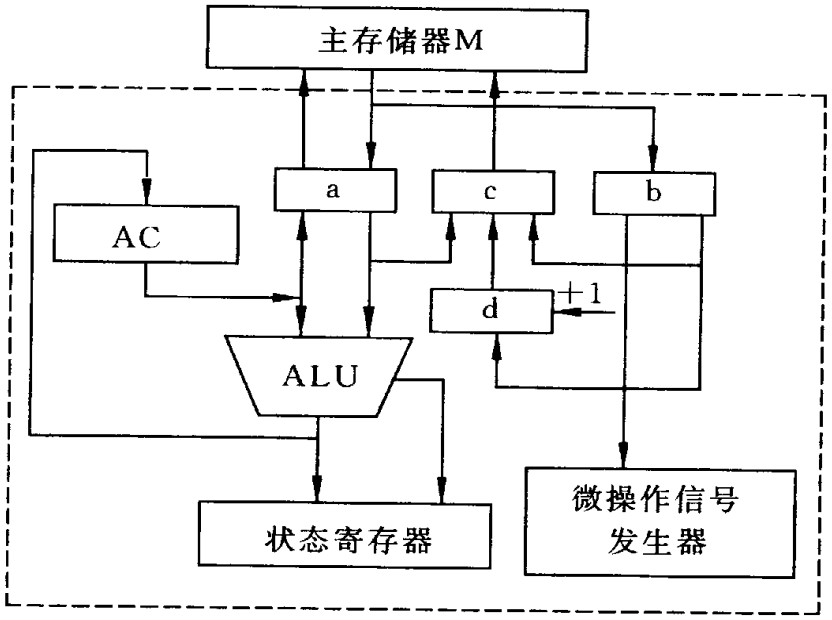
（1）a: 存储数据寄存器MDR；　　b: 指令寄存器IR；

　　c: 存储地址寄存器MAR；　　d: 程序计数器PC。

（2）M → IR → 控制器

（3）读：M → MDR → ALU → AC；

　　写：AC → MDR → M



7.2.3 指令执行过程（流程）

- 考虑一条比较简单的指令：

  Add （R3），R1　　((R3)) + (R1)=>R1

- 执行这条指令需要下列动作：

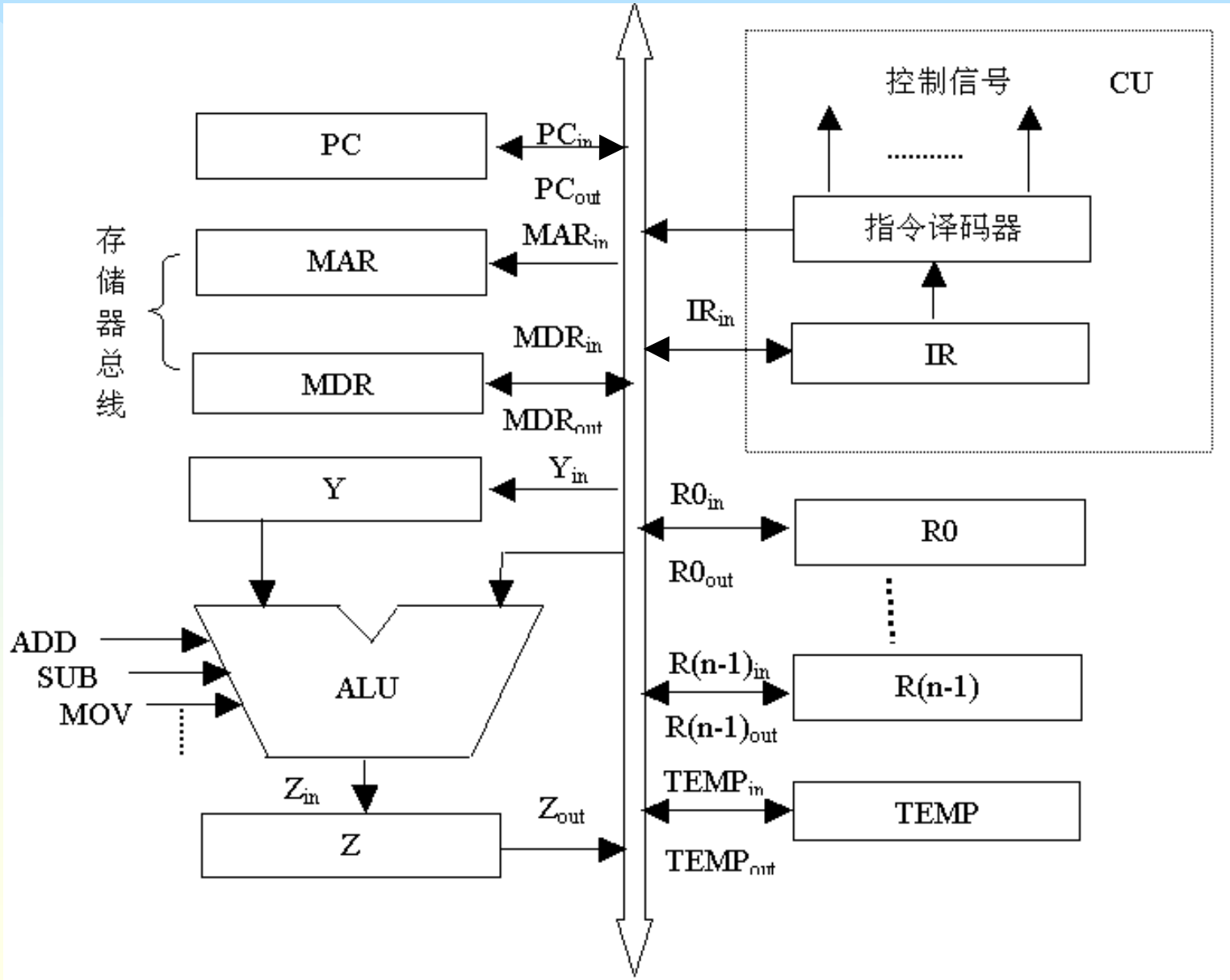
（1）取指令；

（2）取第一个操作数(由R3指出的存储单元的内容)；

（3）完成加法运算；

（4）结果存入R1中。

指令： Add （R3），R1 ((R3)) + (R1)=>R1



▲指令Add (R3)，R1的执行控制命令序列

步	动作	说明
1	PCout，MARin，Read	；取指，(PC)+1 => PC
2	WMFC	；
3	MDRout，IRin	；指令 => IR
4	R3out，MARin，Read	；取数据
5	R1out，Yin，WMFC	；(R1) => Y
6	MDRout，Add,Zin	；相加
7	Zout，R1in，End	；结果 => R1

- 无条件转移(相对)指令的控制命令序列

步　　动作

1　PCout，MARin，Read
2　WMFC
3　MDRout，IRin
4　PCout，Yin
5　（IR的偏移字段)out，Add，Zin　；（若是双字指令？）
6　Zout，PCin，End

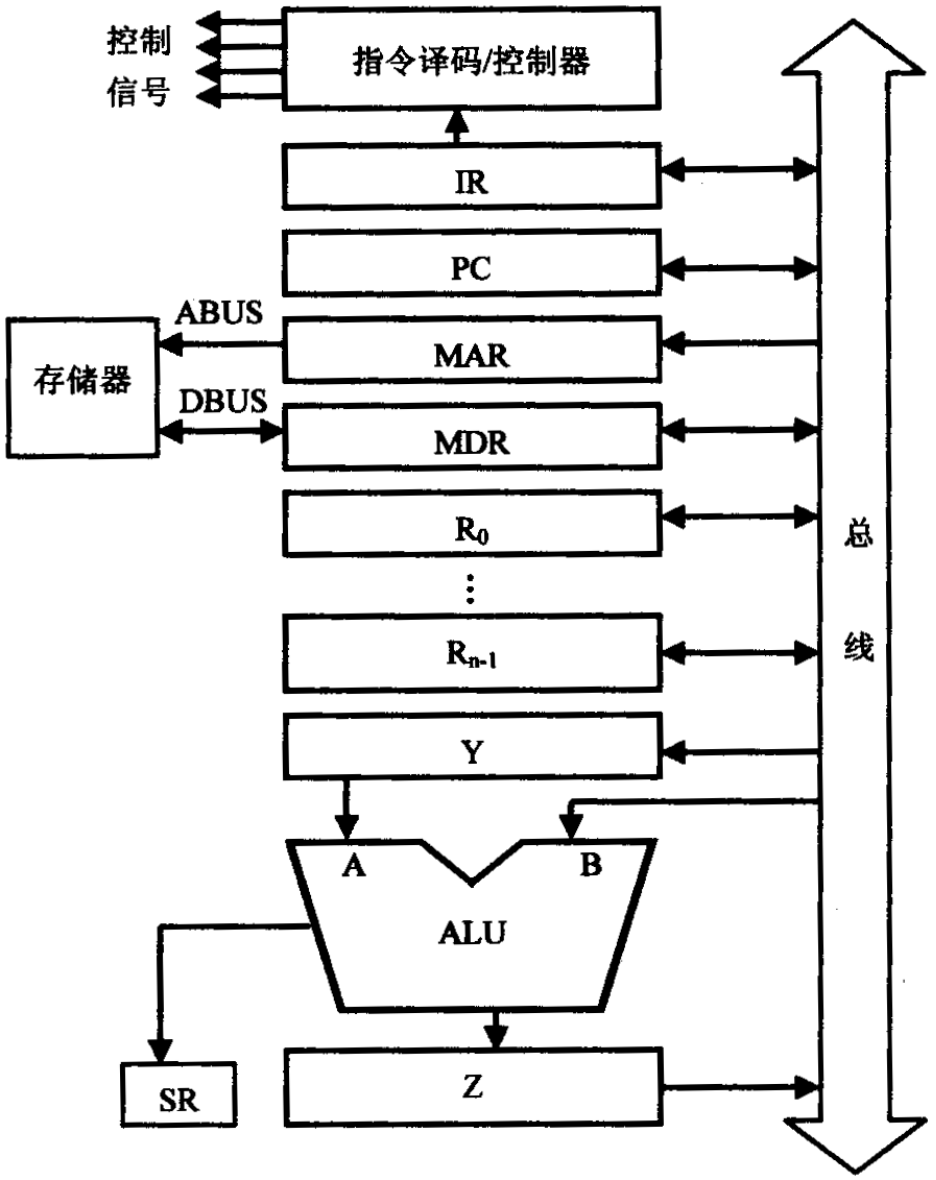
- 条件(比较)转移的情况

上述控制序列中第4步应改为：

If N=0 then End　　；N为符号位
If N=1 then PCout,Yin

例. 左图为单总线结构CPU结构图，所需的控制信号如下：

R0out：R0的输出控制
R0in： R0的输入控制
MDRout：MDR的输出控制
MDRin： MDR的输入控制
IR（地址段）out：
IRin： IR的输入控制
PCout：PC的输出控制
PCin：PC的输入控制
PC+1：PC计数更新控制
Yin：Y的输入控制
MARin：MAR的输入控制
Zout：Z的输出控制
Zin： Z的输入控制
ADD：ALU加法操作控制
ROL：循环左移控制信号
Read：存储器读信号
Write：存储器写信号



- 试分析以下几条指令的执行过程，写出指令执行流程。

（1）ADD Z，（MEM）；Z为累加器，MEM内存单元地址，结果保存在累加器中。

1 PCout，MARin，Read，Clear Y,1=>C0，Add，Zin ；取指，(PC)+1

2 Zout，PCin，WMFC ；(PC)+1 => PC

3 MDRout，IRin ；指令 => IR

4 IR(地址段)out，MARin，Read ；取数据

5 Zout，Yin， WMFC ；(MEM) => Y

6 MDRout，Add，Zin，End ；相加，结果 => Z

（2）ROL（MEM）；将内存中MEM单元的数据循环左移1位（设R1有循环左移功能）。

1 PCout, MARin, Read, Clear Y, 1=>C0, Add, Zin
；取指，(PC)+1

2 Zout, PCin, WMFC ；(PC)+1 => PC

3 MDRout, IRin ；指令 => IR

4 IR(地址段)out, MARin, Read ；取数据

5 WMFC ；（MEM）→DBUS→MDR

6 MDRout, R1in

7 ROL R1 ；左移结果 => R1

（3）MOV x（R0）,@ (R1)；其中，源操作数为变址寻址，目的操作数为间接寻址， x在该指令的下一字中 。

1 PCout, MARin, Read, Clear Y, 1=>C0, Add, Zin
；取指，(PC)+1

2 Zout, PCin, WMFC ；(PC)+1 => PC

3 MDRout, IRin ；指令 => IR

4 PCout, MARin, Read, Clear Y, 1=>C0, Add, Zin

5 Zout,PCin,WMFC ； (PC)+1 => PC

6 MDRout,Yin ； x => Y

7 R0out, ADD, Zin ; x +（R0） => Z

8 Zout, MARin, Read, WMFC

; （MEM）=>BUS=>MDR

9 R1out, MARin, Write, WMFC

; MDR=>BUS=>（MEM）

10 END

7.2.4 CPU性能设计

- 决定CPU性能三个因素：指令的功能强弱、时钟周期的长短、执行每条指令所需时钟周期数。

- CPU性能设计的技术措施

  - 流水线技术：指令流水执行过程中，采用相对简单的指令；
  - 提高时钟频率：时钟速度取决于所采用的电子电路的速度和功能部件（如ALU）的实现技术；
  - 微处理器芯片的时钟频率从几兆已经发展到几百兆、几千兆。

1. 多总线组织

- 每条指令的执行时钟周期尽量少，最好是一个时钟周期；

- 单总线只允许在一个时钟周期内传输一个数据；
- CPU内部采用多总线结构，一个时钟周期内可传输多个数据；
- Pentium处理器就采用了分层次多总线结构。

▲多总线结构的实现

- 采用存储元件阵列组成寄存器堆。

▲考虑以下三操作数指令的例子：

OP Rsrc1，

Rsrc2，

Rdst

- 只用一个机器周期

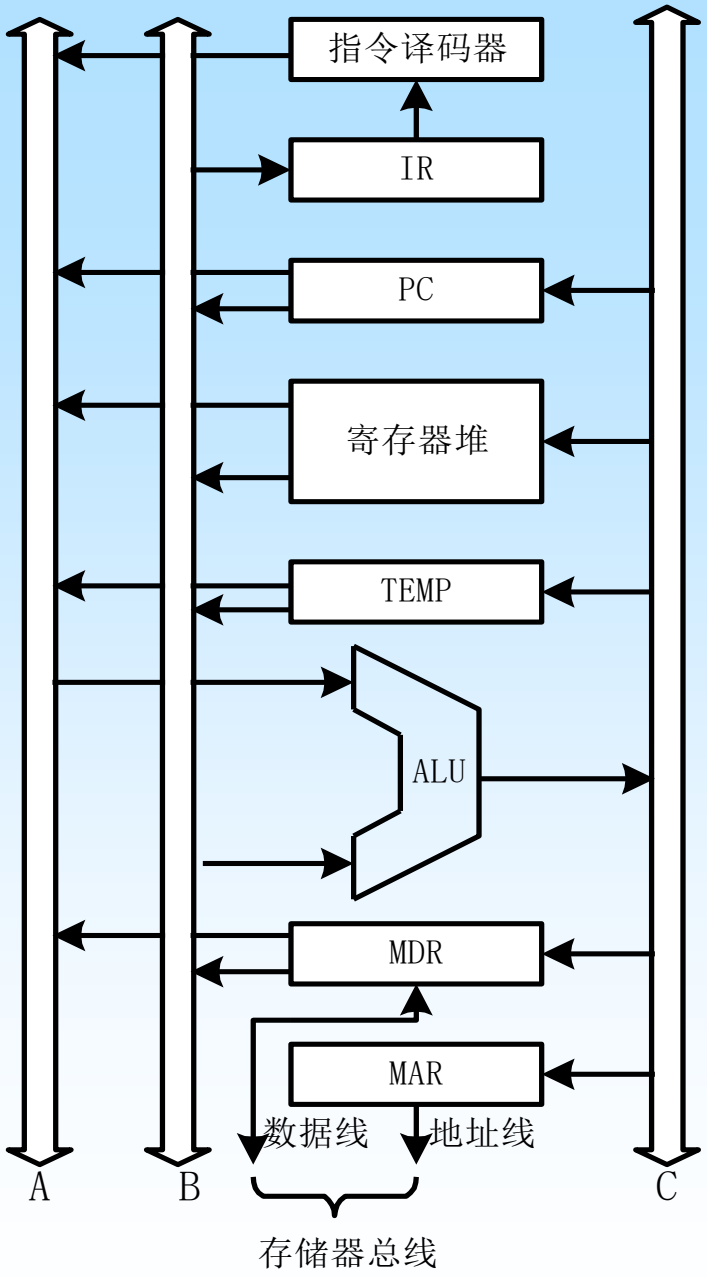


图7.9 CPU的三总线组织

2. 指令流水

- CPU能够将指令的取指周期和执行周期重叠起来，会极大地改善CPU性能。
- 一条指令流水线可由如下5段组成：(后图)

  S1，取指令(IF)：由cache或主存取指令；

  S2，指令译码(ID)：生成指令将要完成的操作；

  S3，取操作数(OF)：确定操作数地址，读取存储器操作数和寄存器操作数；

  S4，执行(EX)：对操作数完成指定操作；

  S5，写回(WB)：修改目标操作数。

- 后一指令的第i功能步与前一指令的第i+1功能步同时进行。

3. 指令发射与完成策略

- 指令发射：启动指令进入执行的过程；
- 指令发射策略：指令发射所使用的协议或规则；
- 指令的发射和完成策略：用于流水线的调度，充分利用指令级的并行度，提高处理器性能。

- 按序发射和乱序发射：

  将有(结构、控制、数据)冒险的指令推后发射，将后面的无冒险的指令提前发射，以改善流水线性能。

4. 动态执行技术

▲ 动态执行技术：通过预测程序流来调整指令的执行，且分析程序的数据流来选择指令执行的最佳顺序。

▲ 动态执行技术由三方面组成：

- 转移预测技术：(预测正确率高达90%)：允许程序的几个分支流同时在处理器中进行；
- 数据流分析：通过分析指令之间的数据相关性，产生优化的重排序的指令调度；
- 推测执行：将多个程序流向的指令序列，进行优化顺序送往处理器的执行部件去执行，尽量保持多端口多功能的执行部件始终为“忙”。

5. 一个完整的CPU

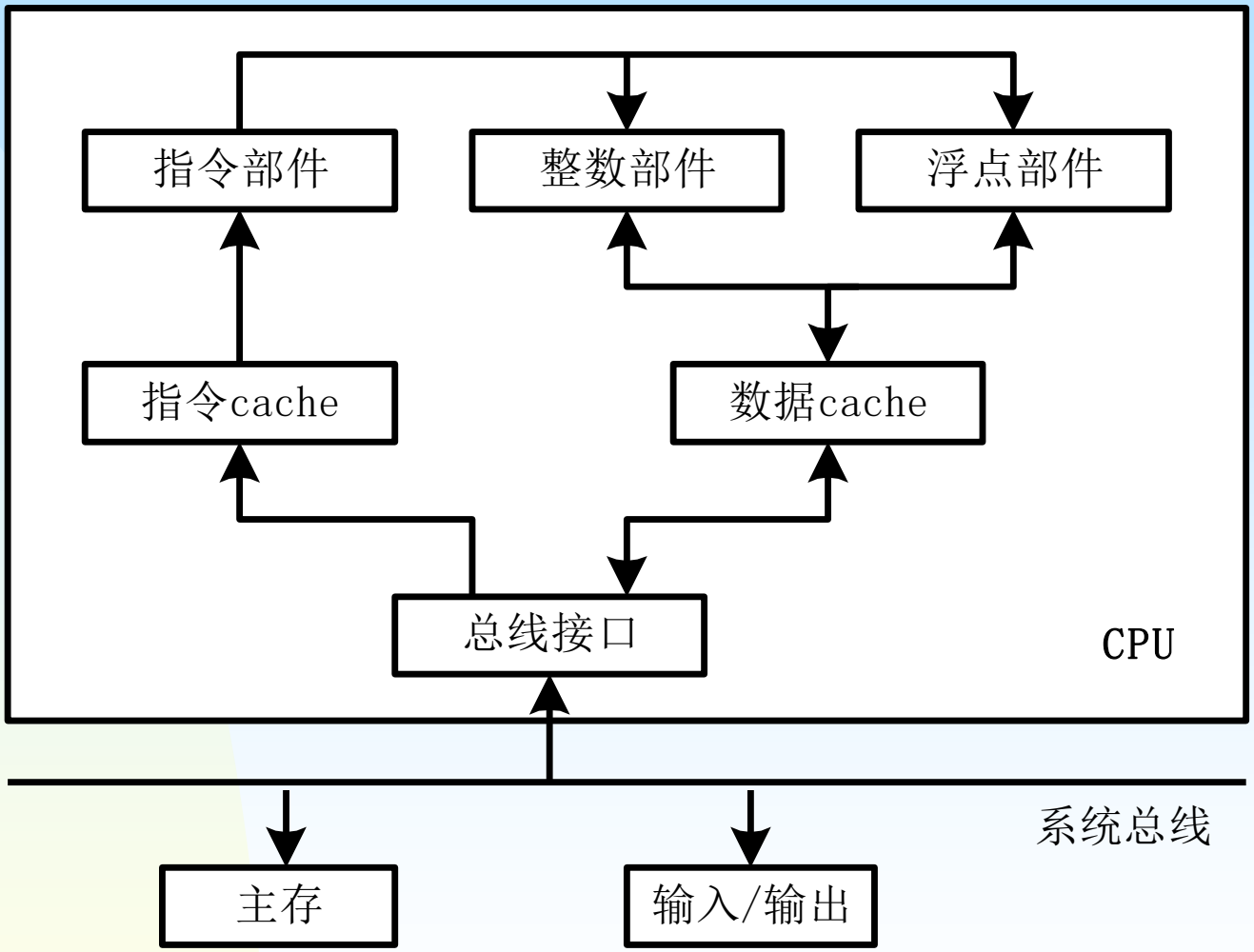


图7.10 一个完整的CPU

7.3 组合逻辑与PLA控制器

- 操作控制信号形成部件：产生指令所需要的操作控制信号序列，来控制计算机各部分的操作。

- 控制器的结构可分为：
  - 组合逻辑控制器
  - PLA控制器
  - 微程序控制器

  } 硬布线控制

- 微程序控制与硬布线控制的兴衰主要由机器的复杂性与集成电路的水平造成的。

（1）复杂指令的增加，将部分软件完成的操作交给硬件完成；

（2）使用复杂指令可减少程序的长度以及程序运行时访问存储器的次数，提高速度；

（3）复杂指令的使用，使控制器的设计极为复杂。

（4）在RISC中，大多数指令都能单周期执行；大多采用硬布线控制。

- 微程序控制与硬布线控制比较

（1）实现

- 微程序控制器：由控制存储器和存放当前正在执行的微指令的寄存器直接控制下实现的，
- 硬布线控制器由逻辑门组合实现。
- 微程序控制器电路较规整，易增加或修改，在CISC中广泛使用。

（2）性能

- 在相同的半导体工艺条件下，微程序控制比硬布线控制的速度低，在RISC中一般用硬布线控制。

7.3.1 组合逻辑控制器

- 组合逻辑控制部件：基于时钟信号CLK驱动的计数器来控制每个控制步。
  - 计数器的每个计数值，对应表5.1和表5.2的每一步。
- 操作控制信号的产生由下列因素确定：
  - 控制步计数器的内容（时序信号）；
  - 指令寄存器的内容；
  - 条件码和其它状态标志的内容。
  - 状态标志是指CPU中各部分状态以及连到各控制部件的信号，如MFC信号等。

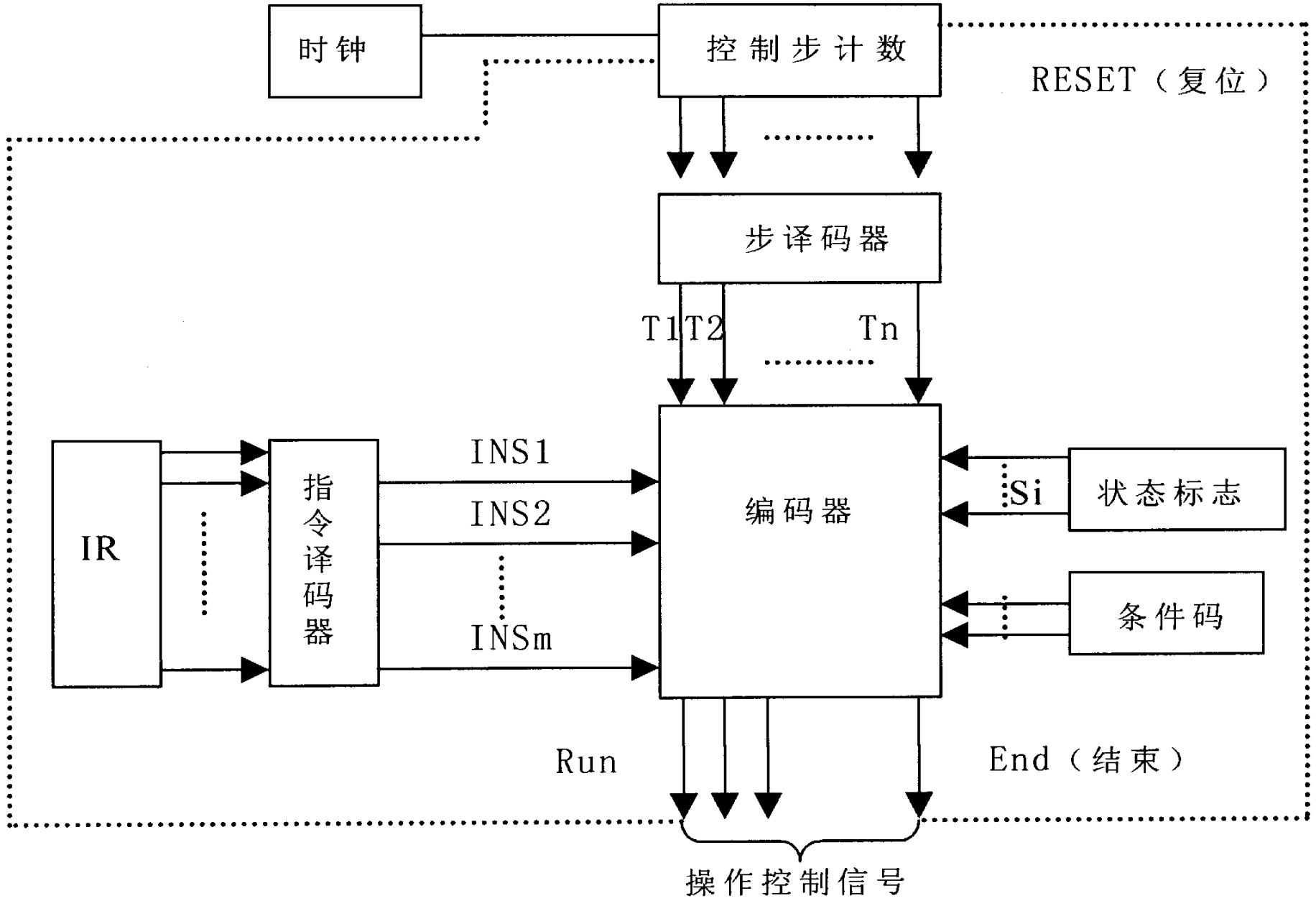


图7.11 简化的组合逻辑控制器

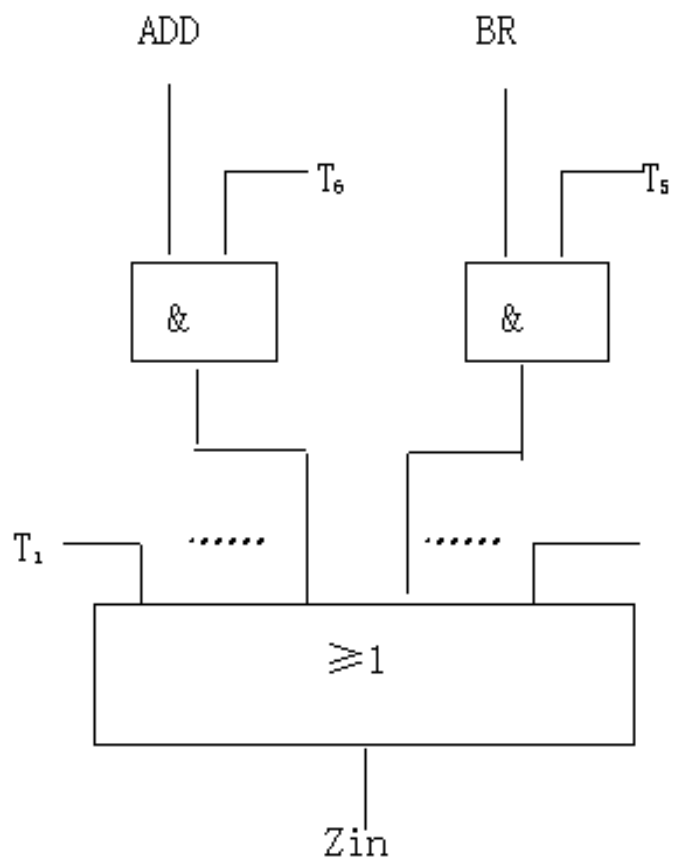


图7.12 Zin信号的产生

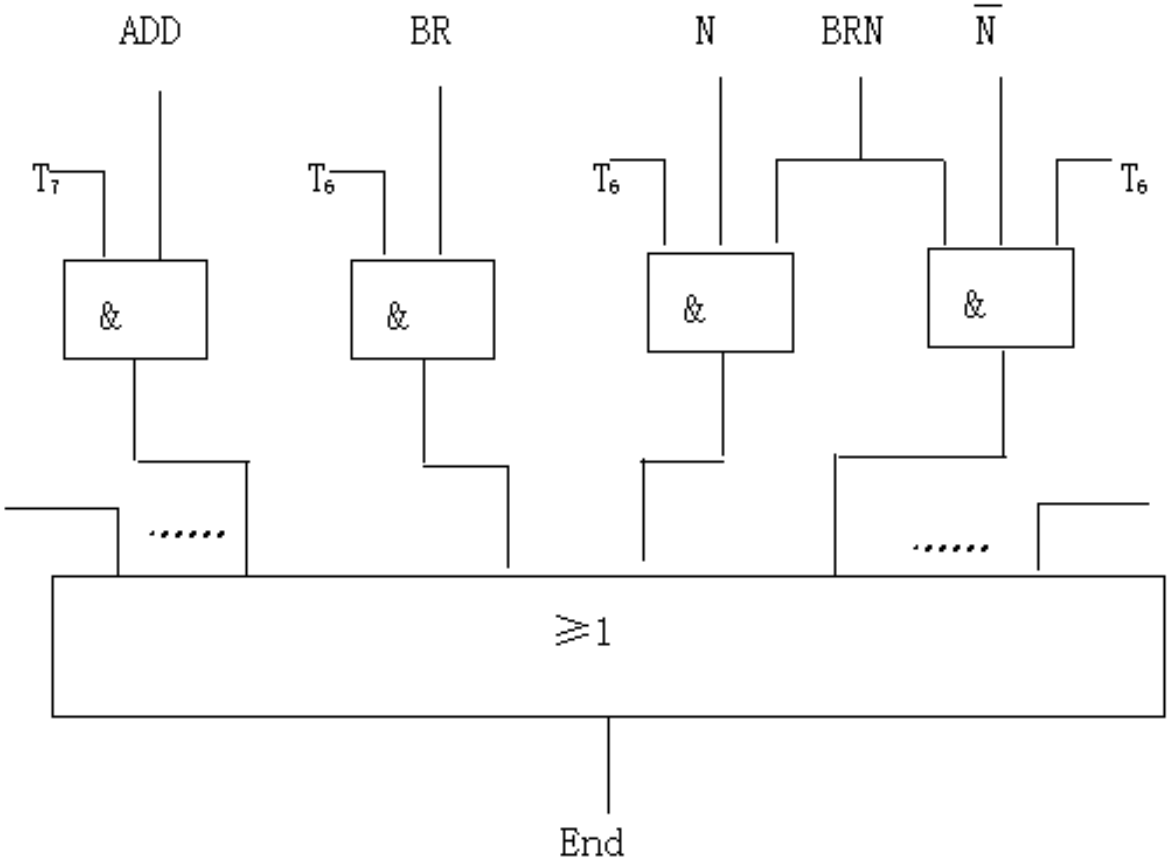


图7.13 End信号的产生

- 图7.12是编码器结构电路的一个例子。电路实现逻辑功能：
  Zin=T1+T6·ADD+T5·BR+…
- 图7.13给出了End控制信号产生电路，其逻辑功能：
  End=T7·ADD+T6·BR+（T6·N+T4·$\overline{N}$）·BRN+…

- 组合逻辑控制器的设计步骤：

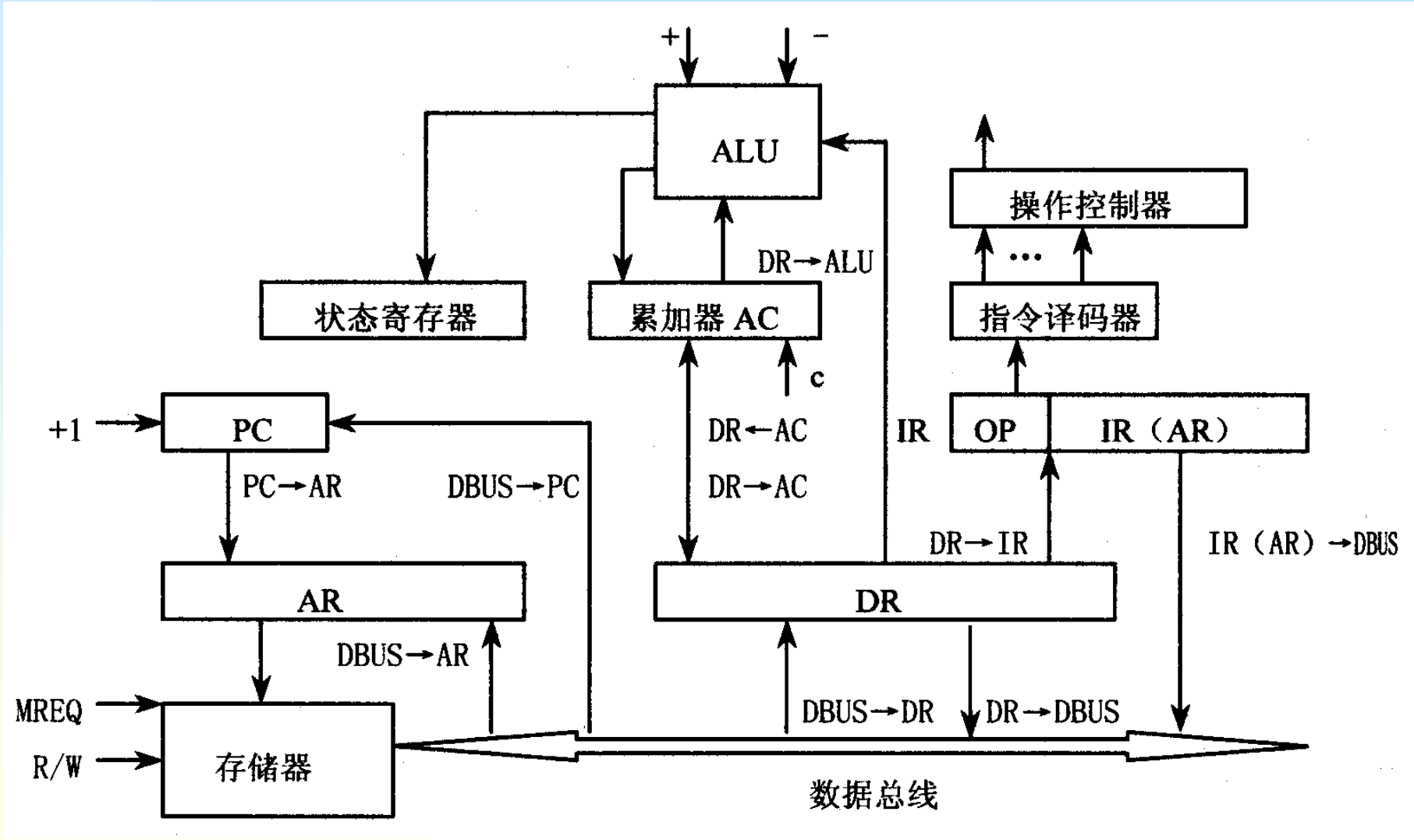
(1) 由CPU数据通路和指令功能，排列出每条指令的操作控制步序列(微操作序列)；

(2) 选择控制方式和控制时序，确定机器的状态周期、节拍与工作脉冲；

(3) 列出每个操作控制信号的逻辑表达式；

- 表达式由指令操作码、时序状态以及状态条件信息（允许有空缺）等因子组成；
- 只须节拍电位控制的信号不用考虑脉冲。

例. CPU结构框图如下图所示，设计以下几条指令的组合逻辑控制器



CLA ； 清AC

ADD I D ； I=0 为直接寻址，即(AC) + (D) => AC

； I=1 为间接寻址，即(AC) + ((D)) => AC

STA I D ； I=0 为直接寻址，即(AC) => (D)

； I=1 为间接寻址，即(AC) => ((D))

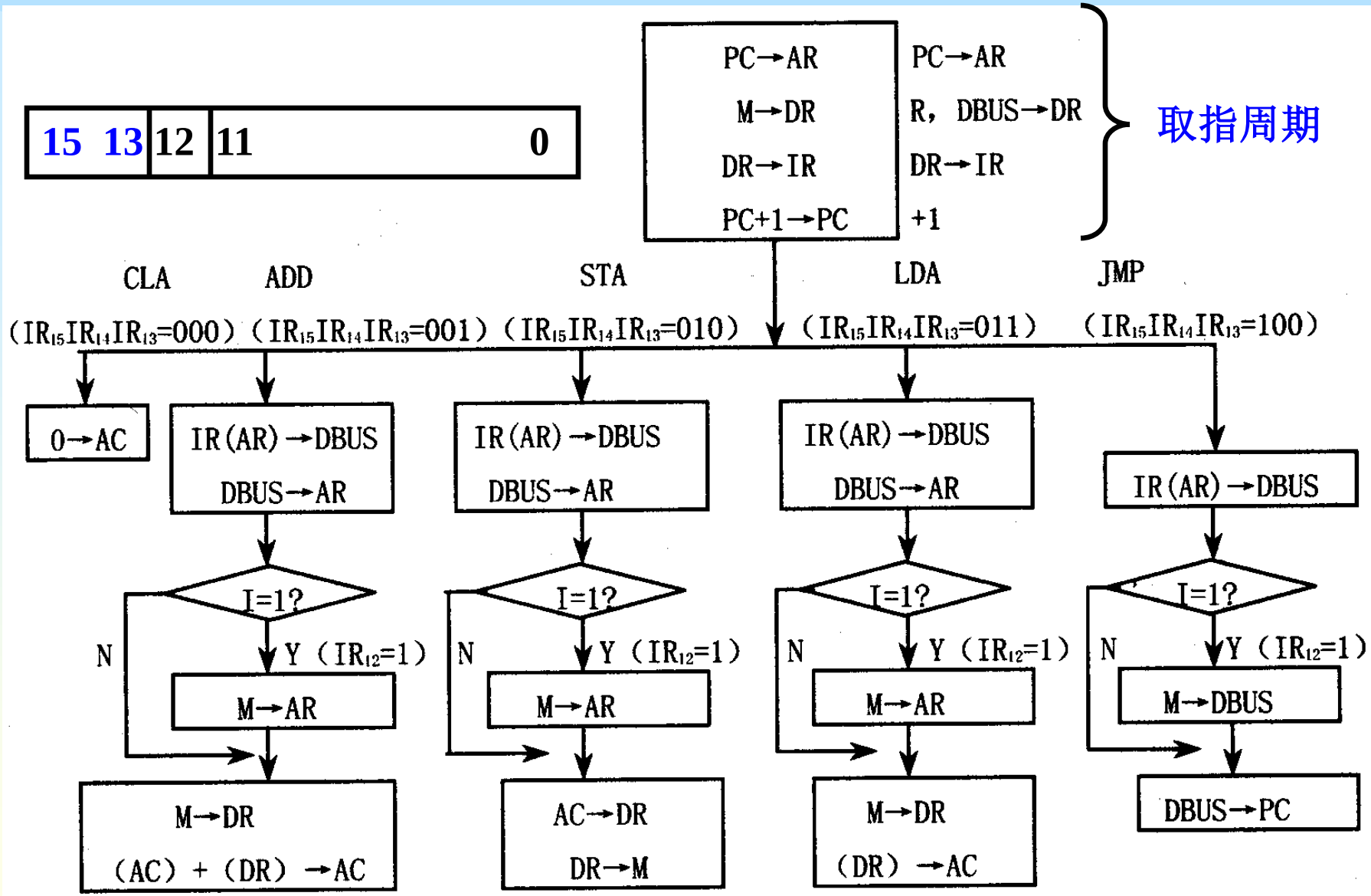
LDA I D ； I=0 为直接寻址，即(D) => AC

； I=1 为间接寻址，即((D)) => AC

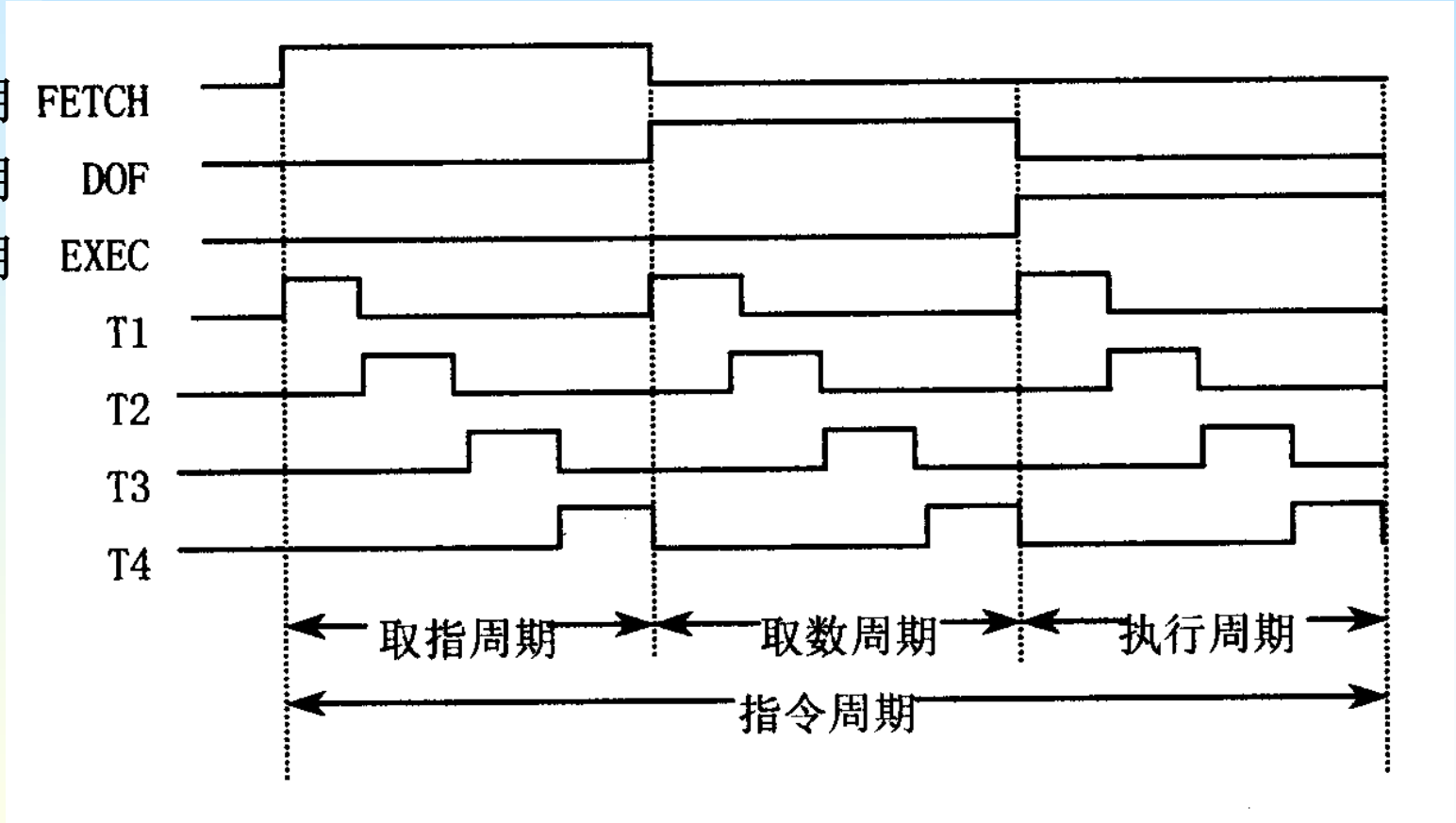
JMP I D ； I=0 为直接寻址，即(D) => (PC)

； I=1 为间接寻址，即((D)) => (PC)

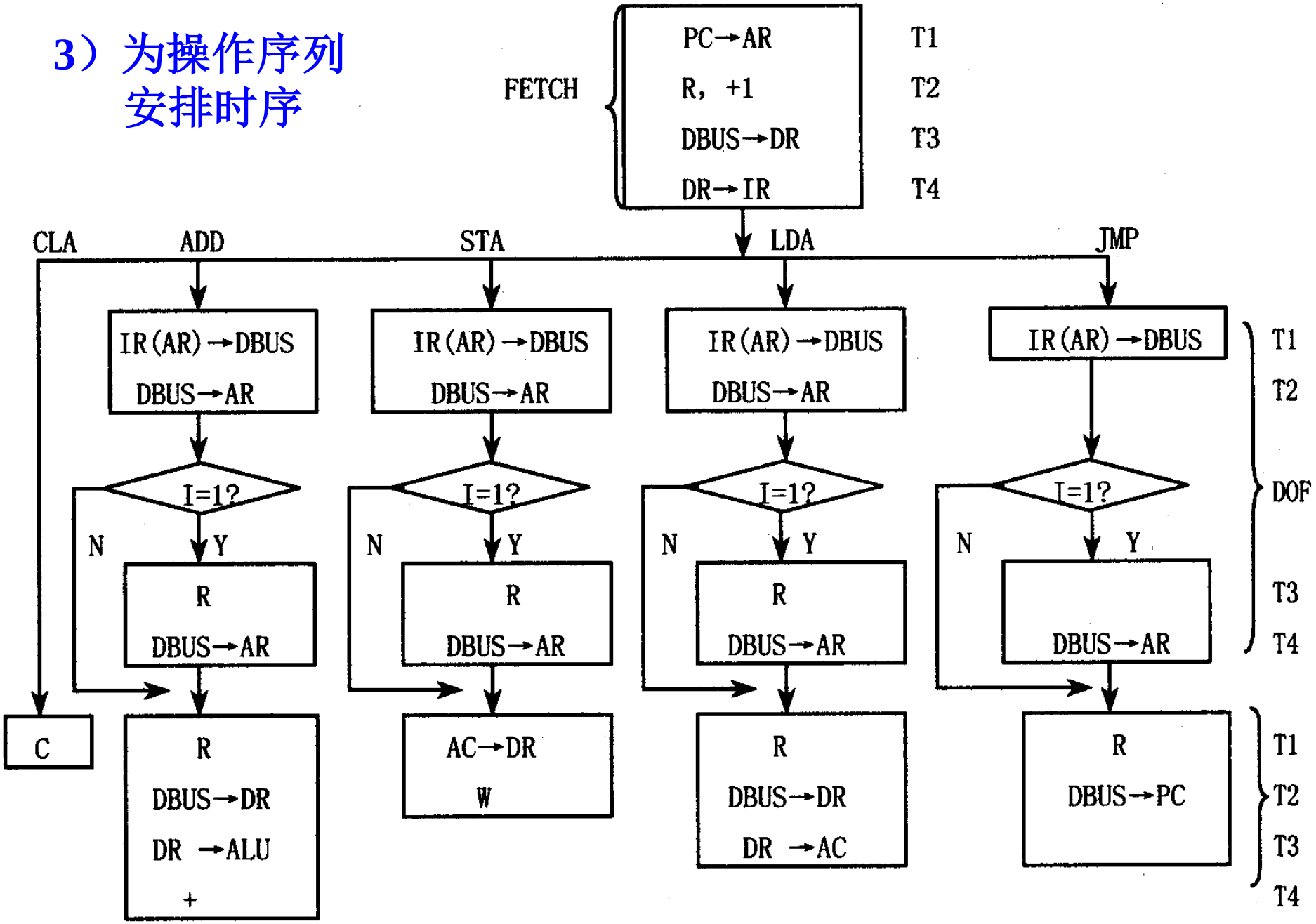
解： 1）根据CPU结构写出指令操作流程图



2）选择同步控制方式的二级时序



3）为操作序列安排时序


FETCH
PC→AR
R，+1
DBUS→DR
DR→IR
T1
T2
T3
T4
CLA
ADD
STA
LDA
JMP
IR(AR)→DBUS
DBUS→AR
I=1?
N
Y
R
DBUS→AR
C
AC→DR
W
DBUS→PC
T1
T2
T3
T4
DOF

4）操作时间列于表中

微操作	FETCH				DOF				EXEC			
	T1	T2	T3	T4	T1	T2	T3	T4	T1	T2	T3	T4
PC→AR	ALL											
R/W=1		ALL 所有指令					$\overline{CLA}\cdot\overline{JMP}\cdot I$		ADD+LDA+JMP			
R/W=0										STA		
MREQ=0		ALL					$\overline{CLA}\cdot\overline{JMP}\cdot I$		ADD+LDA+JMP	STA		
DR→IR				ALL								
+1		ALL										
C									CLA			
IR(AR)→DBUS					$\overline{CLA}$							
DBUS→AR						$\overline{CLA}\cdot\overline{JMP}$		$\overline{CLA}\cdot I$				
DBUS→DR		ALL	ALL							ADD+LDA		
DBUS→PC										JMP		
DR →ALU											ADD	
DR →AC											LDA	
AC →DR									STA			
+												ADD

5） 综合微操作表达式如下：

$PC \rightarrow AR = FETCH.T1$

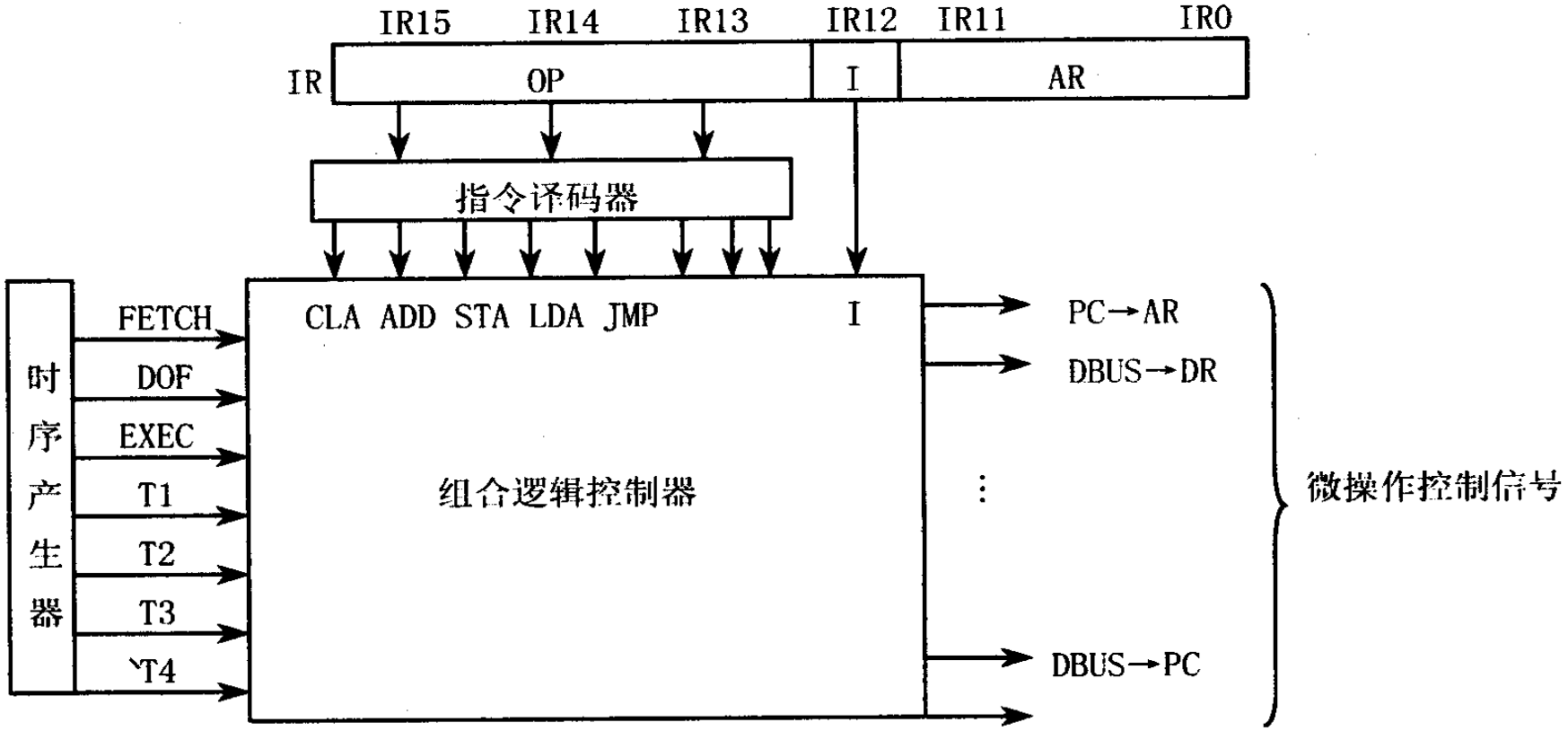
$R/W = FETCH.T2 + DOF.T3.\overline{CLA}.\overline{JMP}.I + EXEC.T1.(ADD + LDA + JMP)$

$R/W = \overline{STA \bullet EXEC \bullet T2}$

$$MREQ = \overline{FETCH.T2 + DOF.T3.\overline{CLA}.\overline{JMP}.I + EXEC.T1.(ADD + LDA + JMP) + \overline{STA \bullet EXEC \bullet T2}}$$

⋮ ⋮

6） 逻辑电路框图



7.3.2 PLA控制

- 可编程逻辑阵列（PLA）产生指令执行操作控制信号

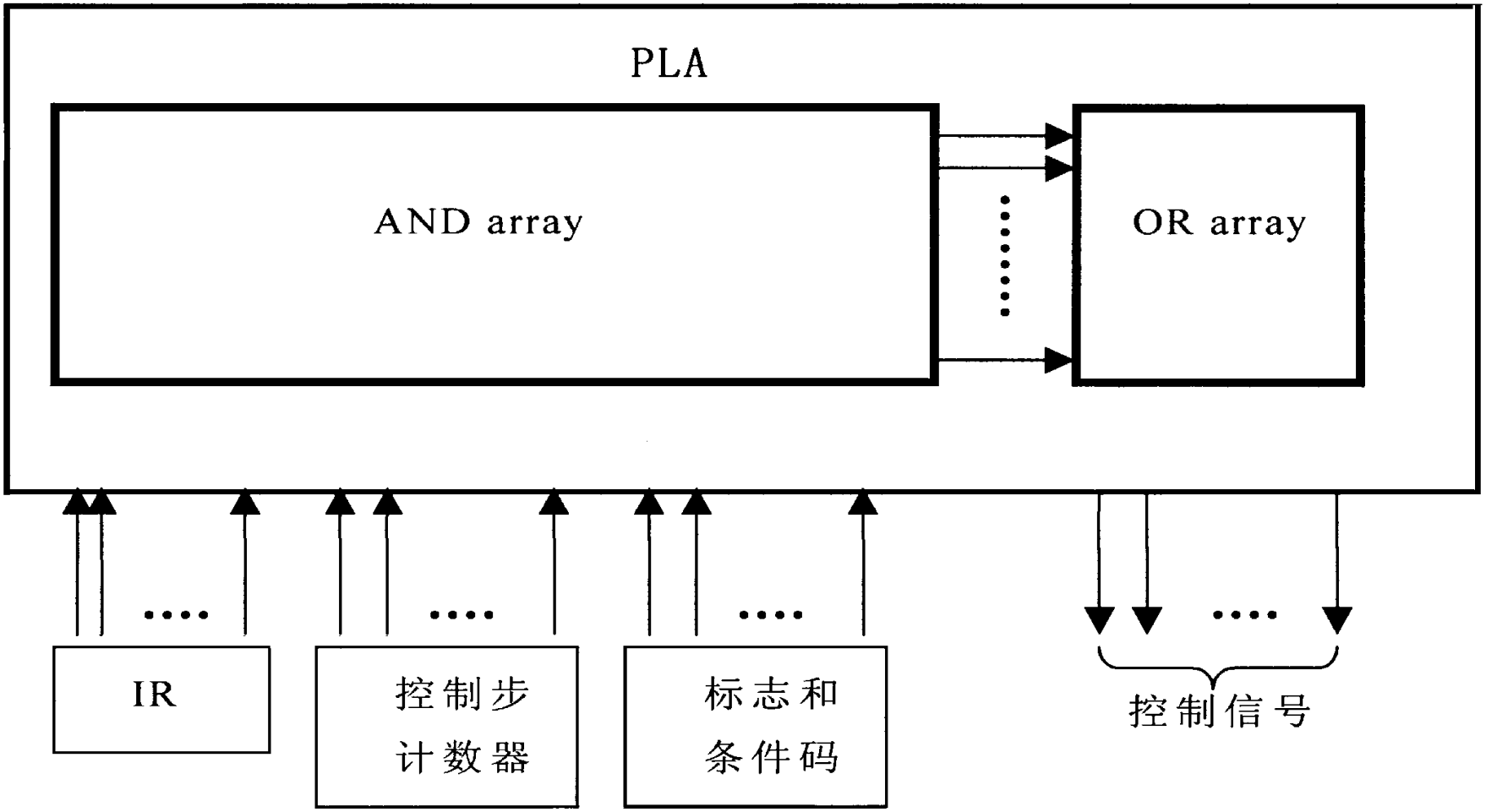


图7.14 在一个VLSI芯片（PLA）上实现控制信号序列

一个简单的PLA模型

$$F1 = a\bar{b}\bar{c}d + ab\bar{c}\bar{d} + bc\bar{d} + \bar{a}\bar{b}\bar{c}d \qquad F2 = a\bar{b}\bar{c}d + abc$$

$$F3 = ab\bar{c}\bar{d} + \bar{a}\bar{b}\bar{c}d + bd \qquad F4 = bc\bar{d} + abc + bd$$

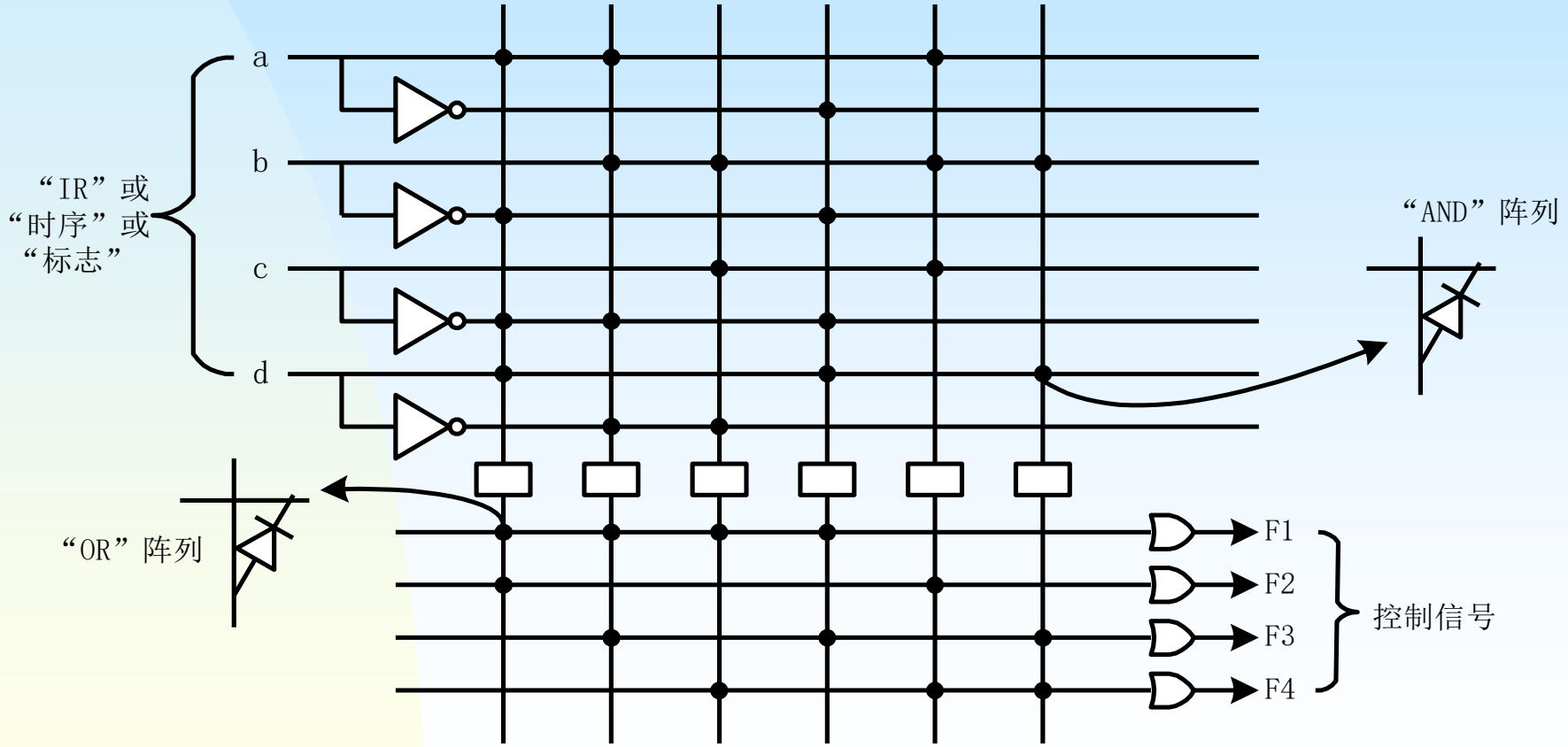


图7.15 PLA模型

7.4 微程序控制器

7.4.1 Wilkes微程序控制器

- **Wilkes提出的微程序控制器设计方案**

- 减少了控制器的复杂性和非标准化程度，把纯硬件的用量限制在狭小范围内。
- 主要缺点：它要比相同或相近半导体技术的硬布线式控制器(如PLA方式)慢一些。

- 微程序控制的基本思想：把机器指令的每一操作控制步编成一条微指令。当执行机器指令时，只要从控制存储器中顺序取出这些微指令，即可按所要求的次序产生相应的操作控制信号。

- 微程序存储器（控制存储器）：存放计算机指令系统所对应的所有微程序的一个专门存储器。

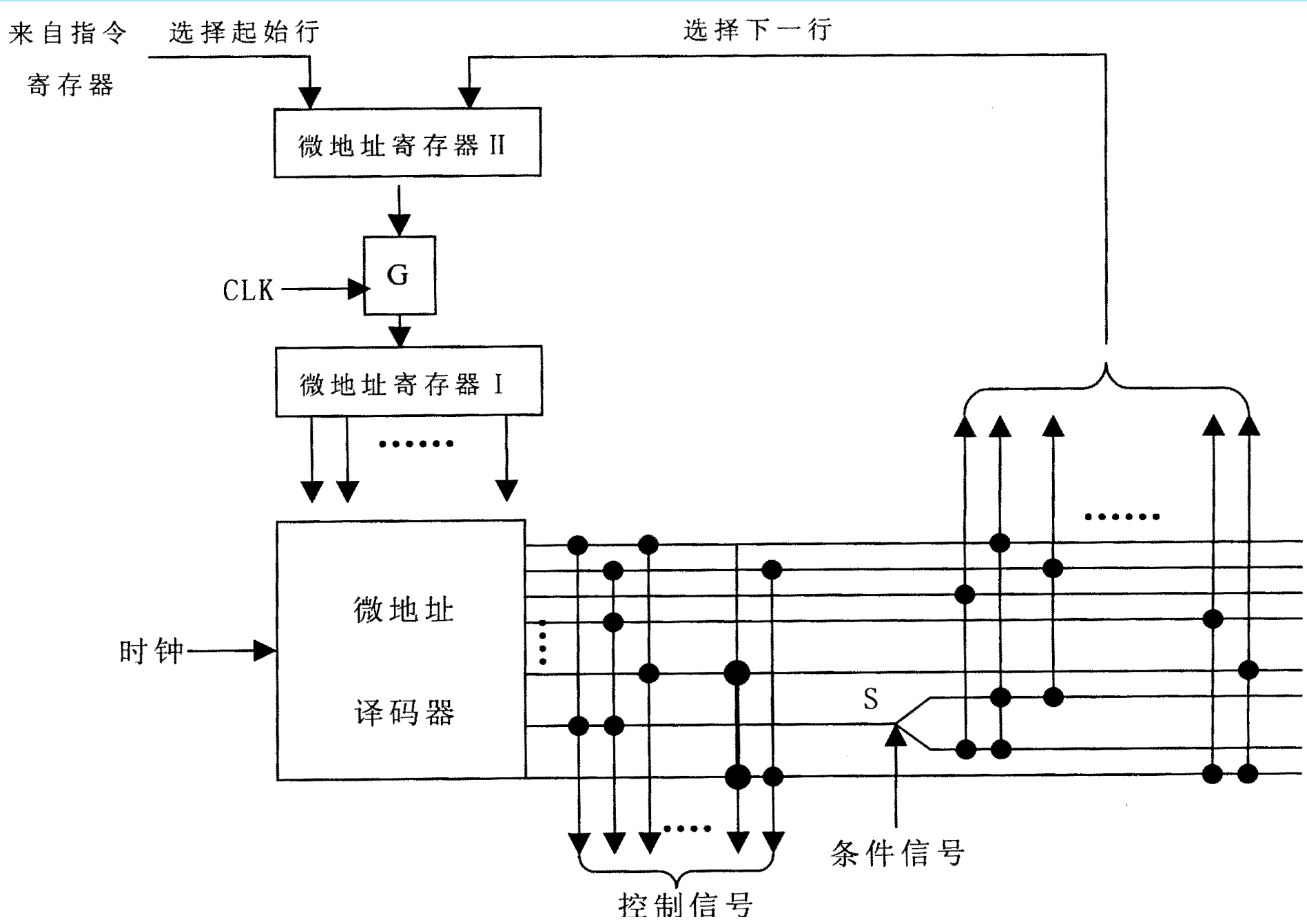
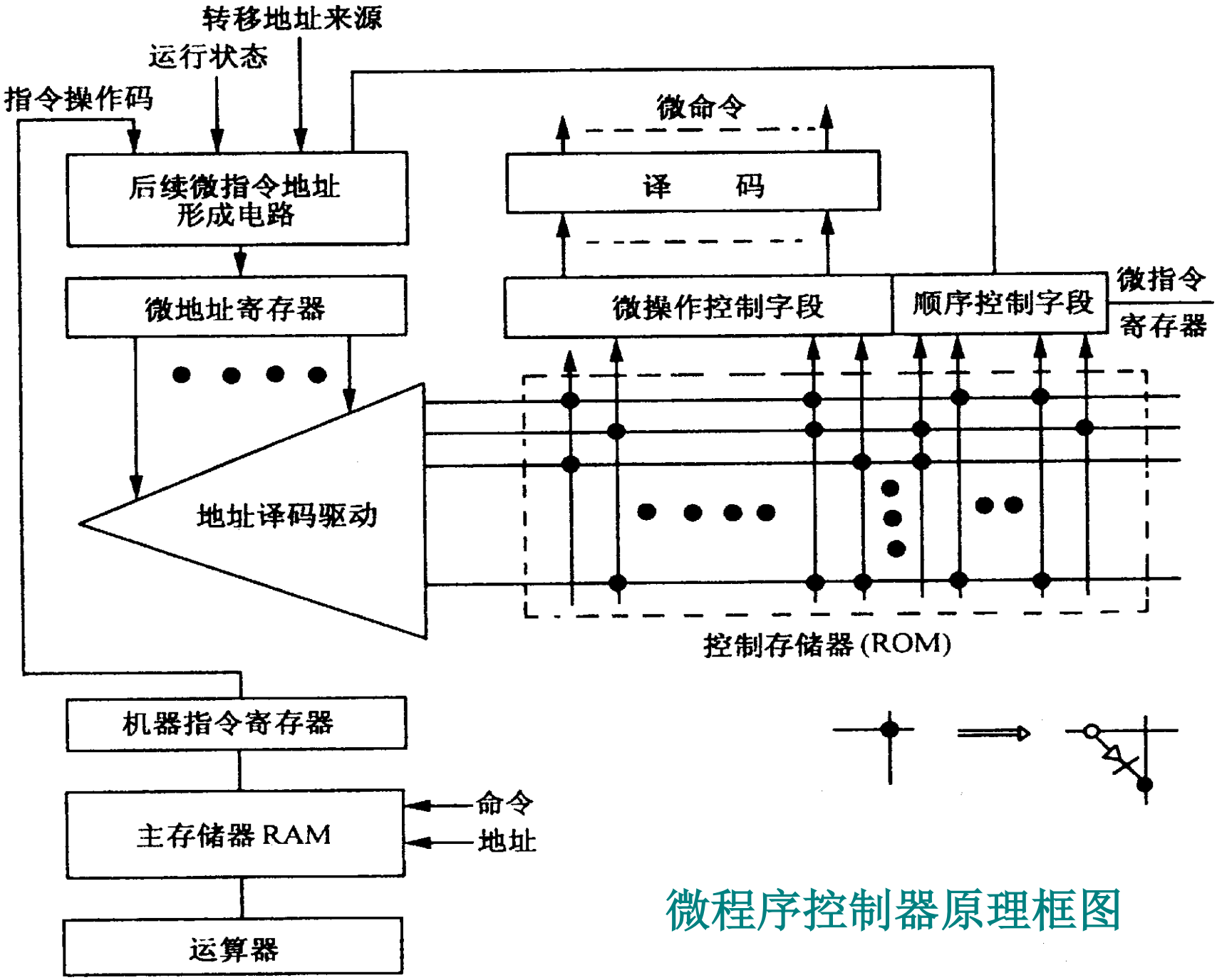


图7.16 Wilkes微程序控制器模型


转移地址来源
运行状态
指令操作码
微命令
后续微指令地址
形成电路
译　　码
微地址寄存器
微操作控制字段
顺序控制字段
微指令
寄存器
地址译码驱动
控制存储器 (ROM)
机器指令寄存器
主存储器 RAM
命令
地址
运算器

微程序控制器原理框图

7.4.2 基本概念

1. 基本术语

- 一条指令的功能是通过执行一系列操作控制步完成的；这些控制步中的基本操作称为微操作。
- 微命令：微操作的控制信号，而微操作是微命令的操作内容。
- 微指令：可同时执行的一组微命令组成一条微指令，完成一个基本运算或传送功能。
- 微程序：完成指定任务的微指令序列称为微程序；一条机器指令其功能可由一段微程序解释完成。

指令Add (R3), R1的执行控制序列

微指令	…	PCin	PCout	MARin	Read	MDRout	IRin	Yin	Add	Zin	Zout	R1out	R1in	R3out	WMFC	End	…	
1		0	1	1	1	0	0	0	0	0	0	0	0	0	0	0		PCout，MARin，Read
2		0	0	0	0	0	0	0	0	0	0	0	0	0	1	0		WMFC
3		0	0	0	0	1	1	0	0	0	0	0	0	0	0	0		MDRout，IRin
4		0	0	1	1	0	0	0	0	0	0	0	0	1	0	0		R3out，MARin，Read
5		0	0	0	0	0	0	1	0	0	0	1	0	0	1	0		R1out，Yin，WMFC
6		0	0	0	0	1	0	0	1	1	0	0	0	0	0	0		MDRout，Add,Zin
7		0	0	0	0	0	0	0	0	0	1	0	1	0	0	1		Zout，R1in，End

表7.1 对应于机器指令Add (R3), R1的微程序的例子

2. 微程序控制器

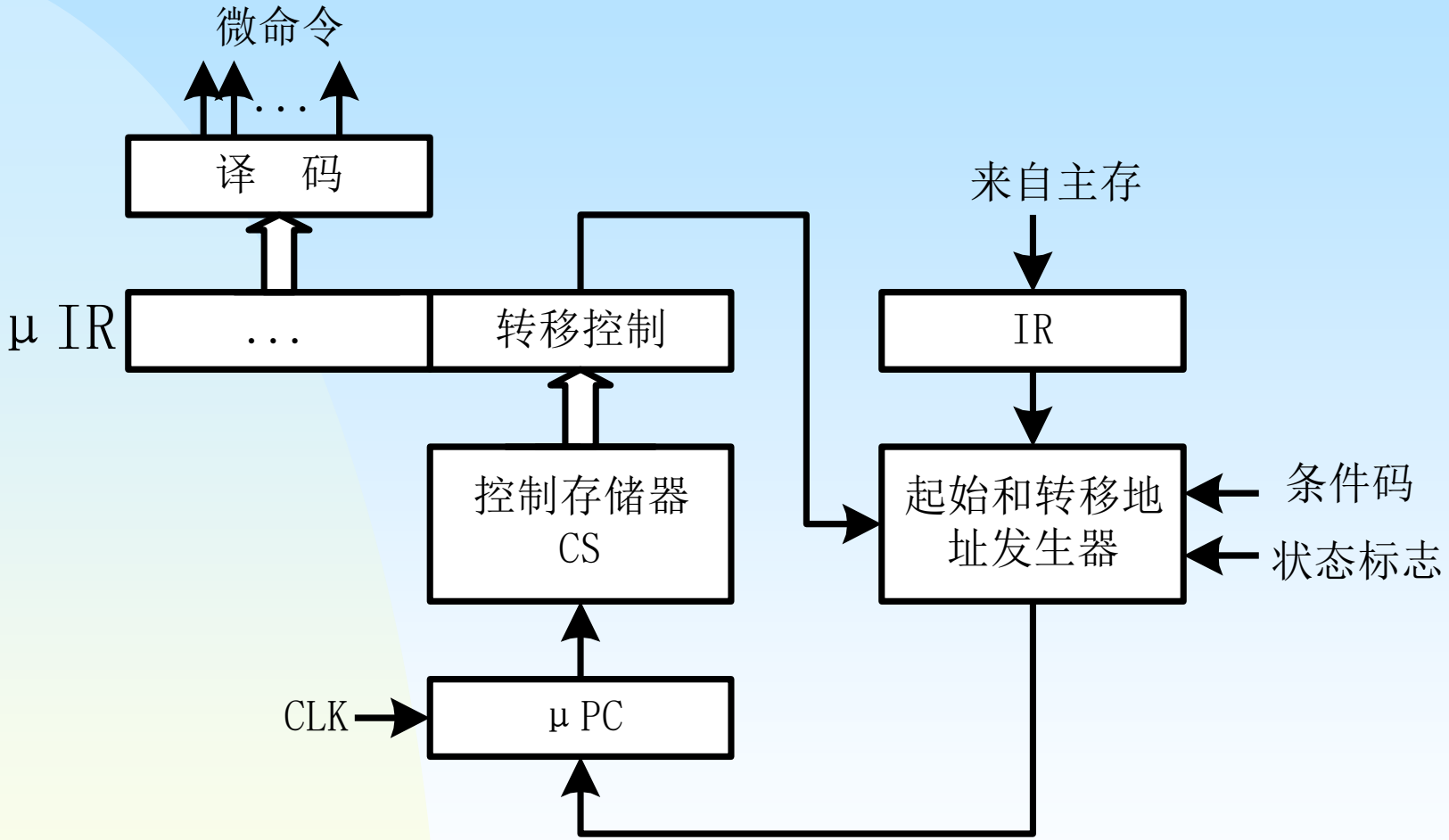


图7.17 微程序控制器的基本组织 组织

▲ 小结：

（1）微程序定义了计算机的指令系统；可以借助改变微程序存储器的内容来改变指令系统。

（2）用只读存储器ROM充当微程序存储器。

（3）任何机器指令的执行都将多次访问控制存储器，控制存储器的速度起着主要的作用。

3. 微程序控制器工作流程

- 不断地执行取指令的微程序和执行相应功能指令的微程序。

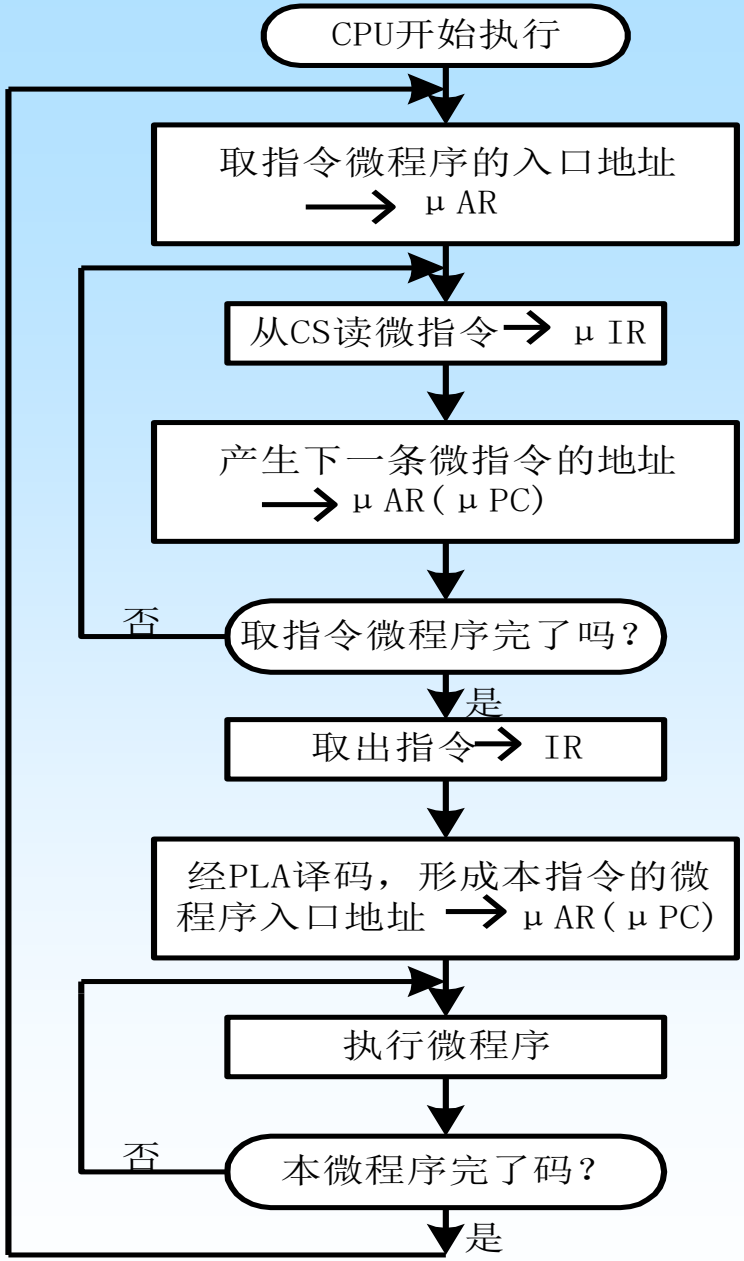


图7.18 微程序控制器工作流程

5.4.3 微指令的格式与编码

- 格式：水平型微指令和垂直型微指令。

▲ 水平型微指令

- 能最大限度地表示微操作的并行性；
- 使用较长的代码；
- 适用于要求较高速度的场合；
- 水平型微指令的码空间利用率较低，编制最佳水平微程序难度较大。

▲ 垂直型微指令

- 采用短格式，一条微指令只能控制一、二个微操作；
- 设有微操作码字段，由微操作码确定微指令的功能；
- 用地址码来指定微操作数所在的寄存器地址或微指令转移地址。
- 编写垂直微程序的方法和传统的程序设计方法更为接近；
- 垂直型微指令面向算法描述而水平型微指令面向处理机内部控制逻辑的描述。

- 水平型微指令编码设计

1）直接表示法

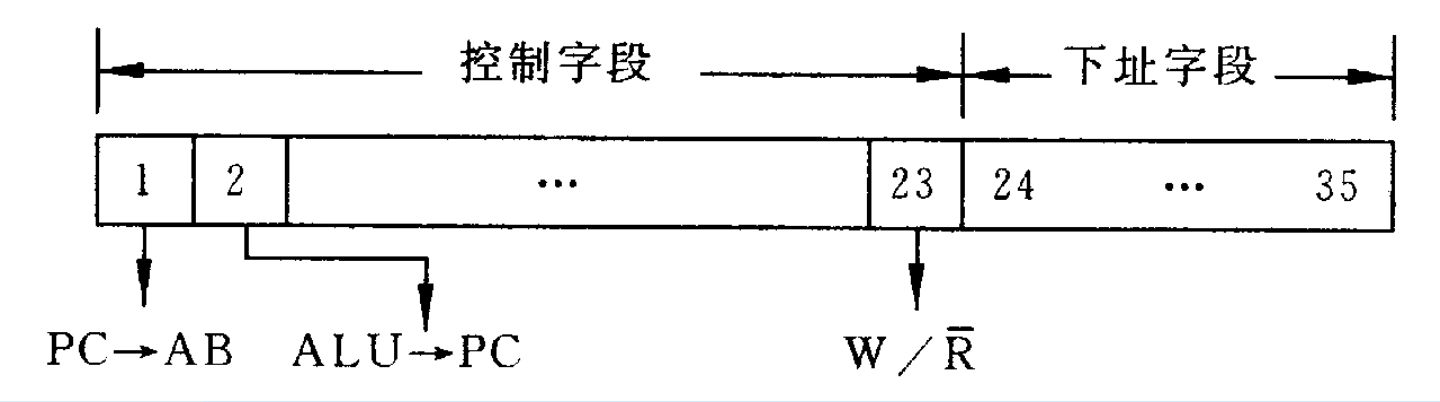
▲ 在微指令的微命令字段中，每个二进制位表示一个微命令。

- 这种方法直观、不必译码、控制电路简单、速度快；
- 通常微命令的个数很多，使得微指令字长多达几百位，实现上非常困难；
- 计算机一般不采用直接表示法。

▲ 控制信号（微命令）列表

序号	控制信号	功　能	序号	控制信号	功　能
1	PC→AB	指令地址送地址总线	13	+	ALU 进行加法运算
2	ALU→PC	转移地址送 PC	14	−	ALU 进行减法运算
3	PC+1	程序计数器加 1	15	∧	ALU 进行逻辑乘运算
4	imm(disp)→ALU	立即数或位移量送 ALU	16	∨	ALU 进行逻辑加运算
5	DB→IR	取指到指令寄存器	17	ALU→GR	ALU 运算结果送通用寄存器
6	DB→DR	数据总线上的数据送数据寄存器	18	ALU→DR	ALU 运算结果送数据寄存器
7	DR→DB	数据寄存器中的数据送数据总线	19	ALU→AR	ALU 计算得的有效地址送地址寄存器
8	rs1→GR	寄存器地址送通用寄存器	20	AR→AB	地址寄存器内容送地址总线
9	rs,rd→GR	寄存器地址送通用寄存器	21	ADS	地址总线上地址有效
10	(rs1)→ALU	寄存器内容送 ALU	22	M/$\overline{IO}$	访问存储器或 I/O
11	(rs)→ALU	寄存器内容送 ALU	23	W/$\overline{R}$	写或读
12	DR→ALU	数据寄存器内容送 ALU			

▲直接表示法微指令格式(ADD x(R1), R2, R3)



PC→AB，PC+1，DB→IR，READ

imm→ALU，rs1→GR，(rs1)→ALU，ADD，ALU→AR

DB →DR，AR→AB，ADS，READ

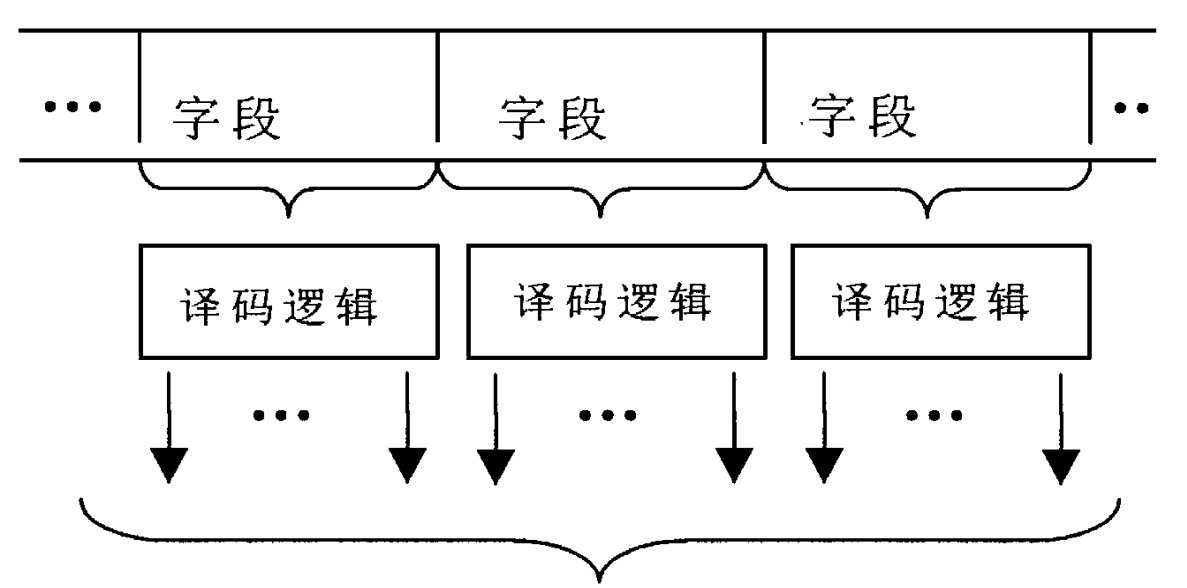
rs2→GR，(rs2)→ALU，DR→ALU，ADD，ALU→GR

▲加法指令的微指令编码

	控制位																							下址												
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35
取指	1		1		1																1	1	0	×	×	×	×	×	×	×	×	×	×	×	×	×
计算地址				1				1		1			1						1							1								1		
取数						1														1	1	1	0			1								1	1
加法运算									1		1	1	1				1									1										

2）分段直接编码法

- 微指令中相容的微命令分配在不同字段；
- 微指令中相斥的微命令组合在一起，编成一个字段；



（a）直接编译

- 它为多数微程序控制的计算机所采用。

微指令

F1	F2	F3	F4	F5

F1（4bit）	F2（3bit）	F3（3bit）	F4（4bit）	F5（2bit）
0000: No transfer	000: No transfer	000: No transfer	0000: Add	00: No action
0001: PCout	001: PCin	001: MARin	0001: Sub	01: Read
0010: MDRout	010: IRin	010: MDRin	⋮	10: Write
0011: Zout	011: Zin	011: TEMPin		
0100: R0out	100: R0in	100: Yin	1111: XOR	
0101: R1out	101: R1in		16 ALU	
0110: R2out	110: R2in		function	
0111: R3out	111: R3in			
1010: TEMPout				
1011: Addressout				

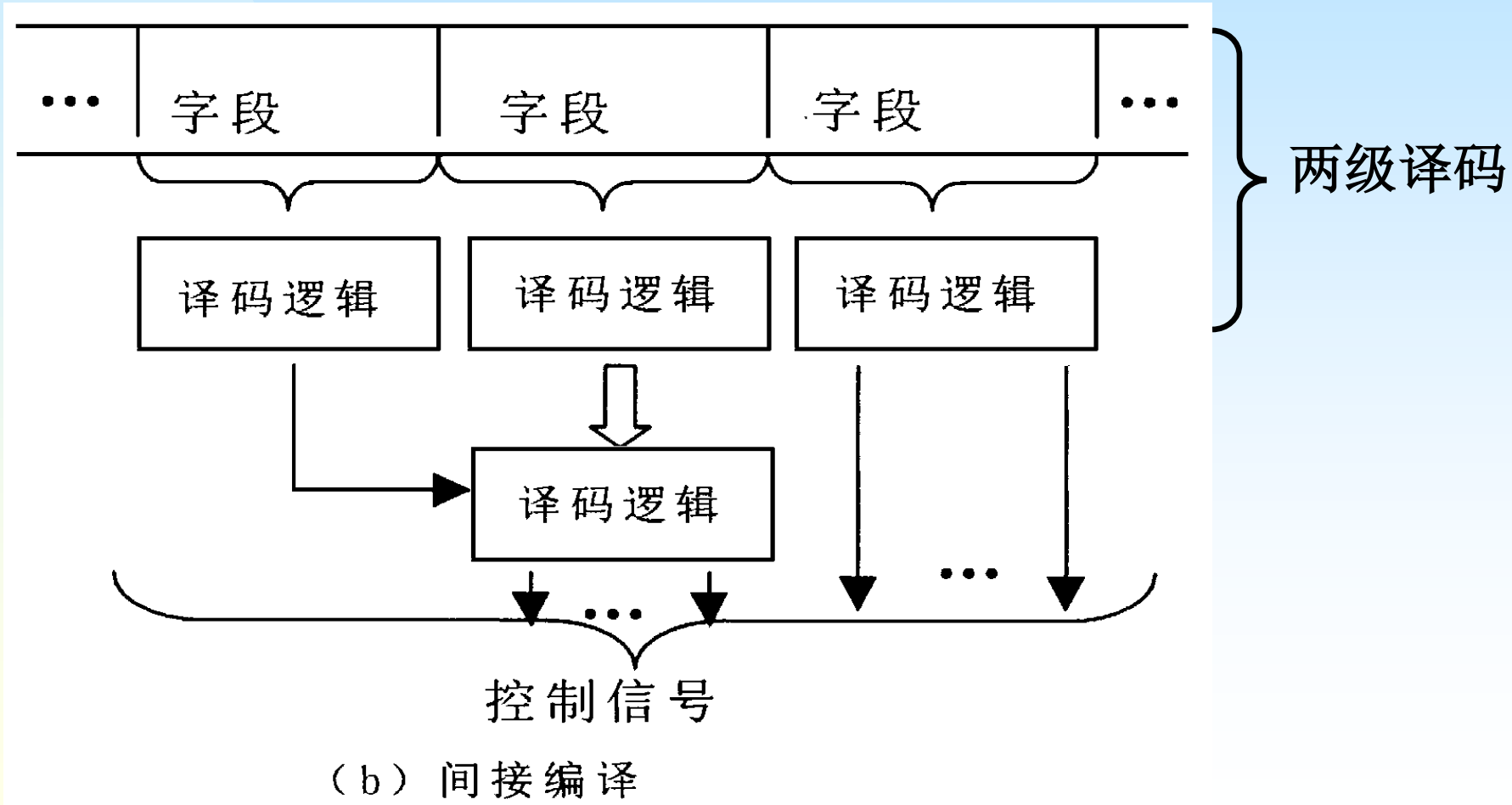
F6	F7	F8	F9	·····

F6（1bit）	F7（1bit）	F8（1bit）	F9（1bit）
0: No action	0: 0→Co	0: No action	0: Continue
1: Clear Y	1: 1→Co	1: WMFC	1: End

图7.19

3）字段间接编码法

- 某一小字段可以表示多个微命令组，到底代表哪一组微命令，则由另一小字段的二进制码确定。



（b）间接编译

4）水平型微指令的直接编码和分段编码的差别

▲ 直接编码

- 采用一位表示一个控制信号将导致微指令长度很长；
- 造成控制存储器的空间极大地浪费。

▲ 分段编码

- 控制信号分组，将那些互斥的信号分在同一组，而相容信号分在不同组；
- 在一条微指令中每一组只能有一个微操作信号有效。

7.4.4 微指令排序（微指令地址的生成）

- 获得下一条将要执行的微指令地址有三种情况：
  - 由指令寄存器IR确定
  - 下一顺序地址
  - 转移
  - 根据当前微指令、条件标志和指令寄存器内容，产生下一微指令的控制存储器地址。
- 微指令地址生成技术有计数器法和下地址字段法两种。

1）计数器法

- 按顺序执行微指令，由μPC负责生成下一微指令的地址(μPC＋1→μPC)；
- 微程序的转移由专门的“转移微指令”来实现，通过微地址发生器以及相应的条件码和标志，生成转移地址送μPC；
- 将机器指令的操作码通过PLA翻译成对应的微程序入口地址，这时微指令可写为：μPC←（PLA）。
- 当存在着较大量的转移微指令时，这将严重影响计算机的工作速度。

2）下址字段法

- 在微指令中设置一个专门的地址字段用以指出下一条微指令的地址；
- 优点是消除了专门的转移微指令，而且在给微指令分配地址时简直没有什么限制；
- 缺点是增加了微指令的长度，有时还会对控制存储器的设计带来影响。

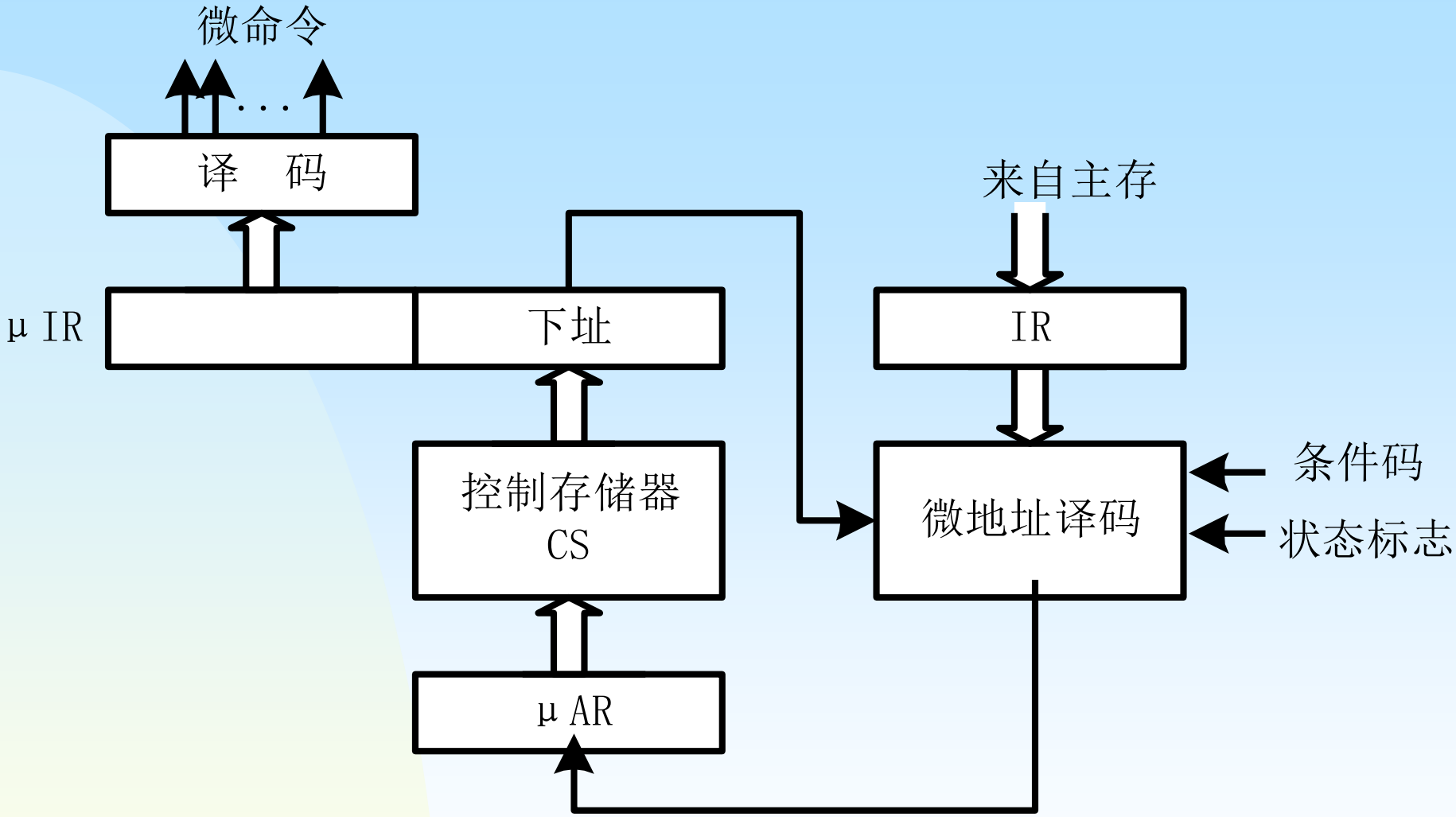


图7.20　采用下址字段法的微程序控制器组织

7.4.5 微程序设计

1. 微指令的执行方式

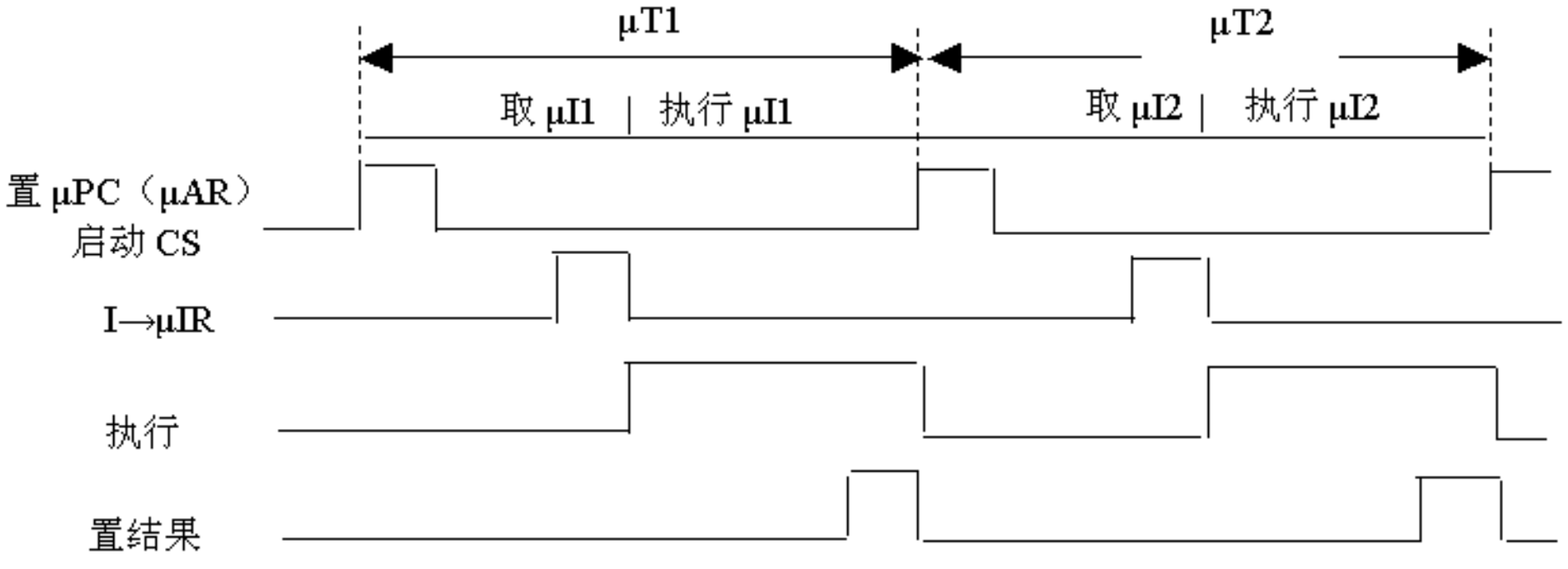


图7.21 微指令的串行执行

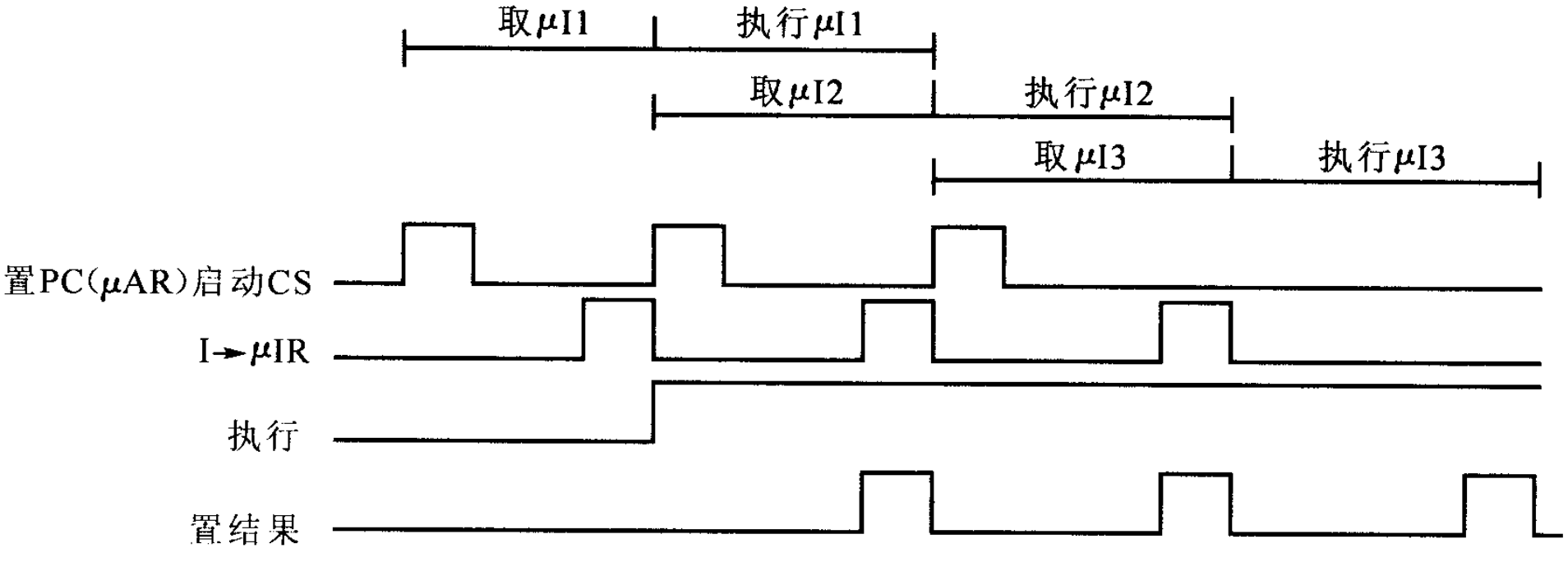


图7.22　微指令的并行执行

2. 动态微程序设计

- 静态微程序设计：实现指令系统的微程序固定不变。
- 动态微程序设计：使计算机的指令系统按需要改变，采用这种技术的机器能更灵活、更有效地适应不同的应用场合，而无须改变或更换硬件。
- 利用部分未定义的扩充操作码，用户可自定义指令；将编写好所定义指令的微程序写入控制存储器（WCS）；成为指令系统的一部分。

7.4.6 微程序控制器设计

- 设计步骤：

1）确定微指令格式与执行方式

- 根据具体情况决定采用水平微指令格式还是垂直微指令格式，微指令是按串行方式还是按并行方式执行等。

2）定义微命令集、确定微命令编码方式和微指令排序方式

- 根据机器指令所需要的所有微操作信号拟定微命令集，并决定微命令编码方式及字段的划分，选择微指令排序方法。

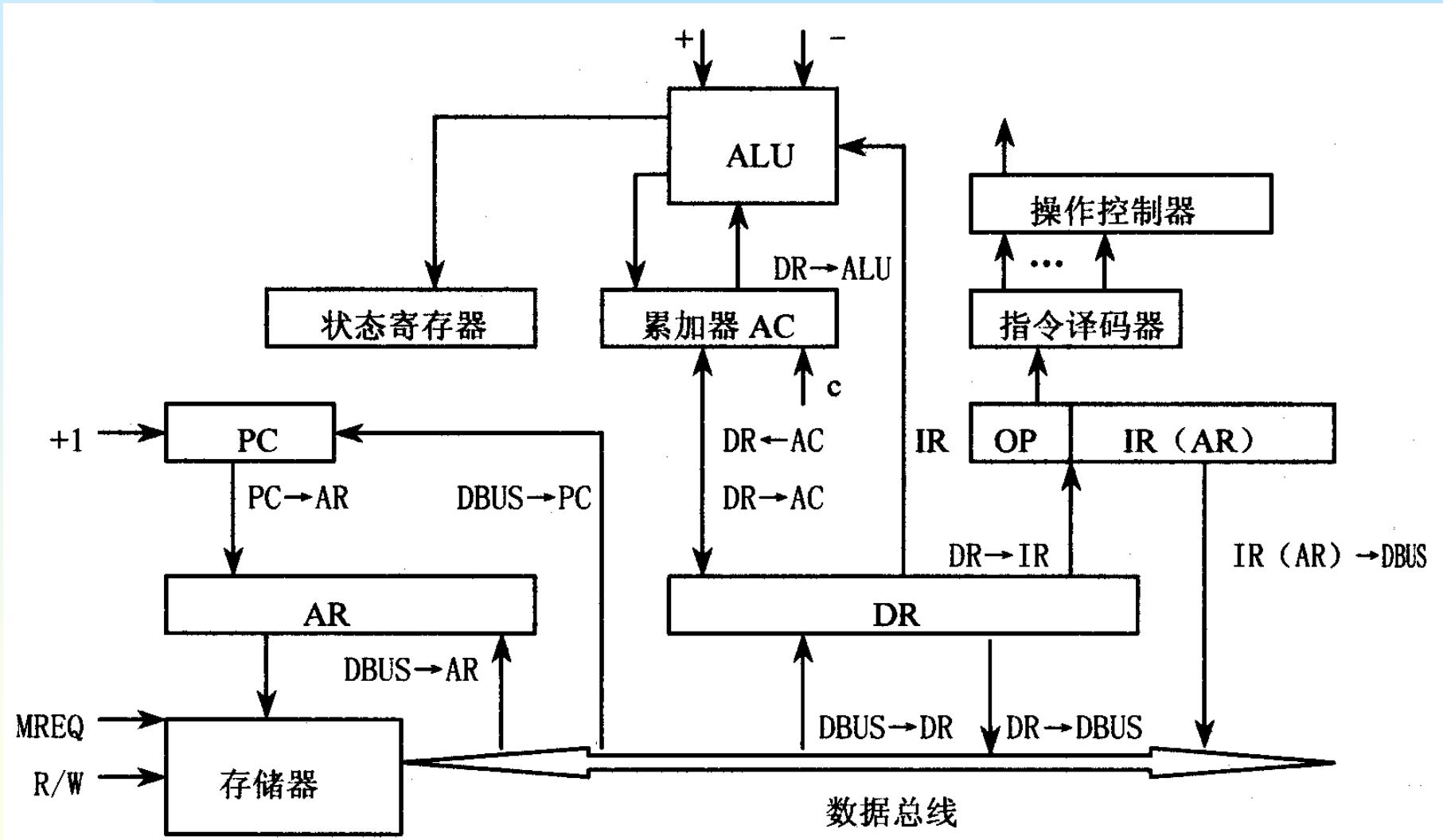
3）编制微程序

- 列出机器指令对应的全部微命令节拍安排；
- 按已定的微指令格式编写出微程序，直到所有机器指令所对应的微程序全部编制完成；
- 进行微程序的优化和代码化（将微程序转换成二进制代码）。

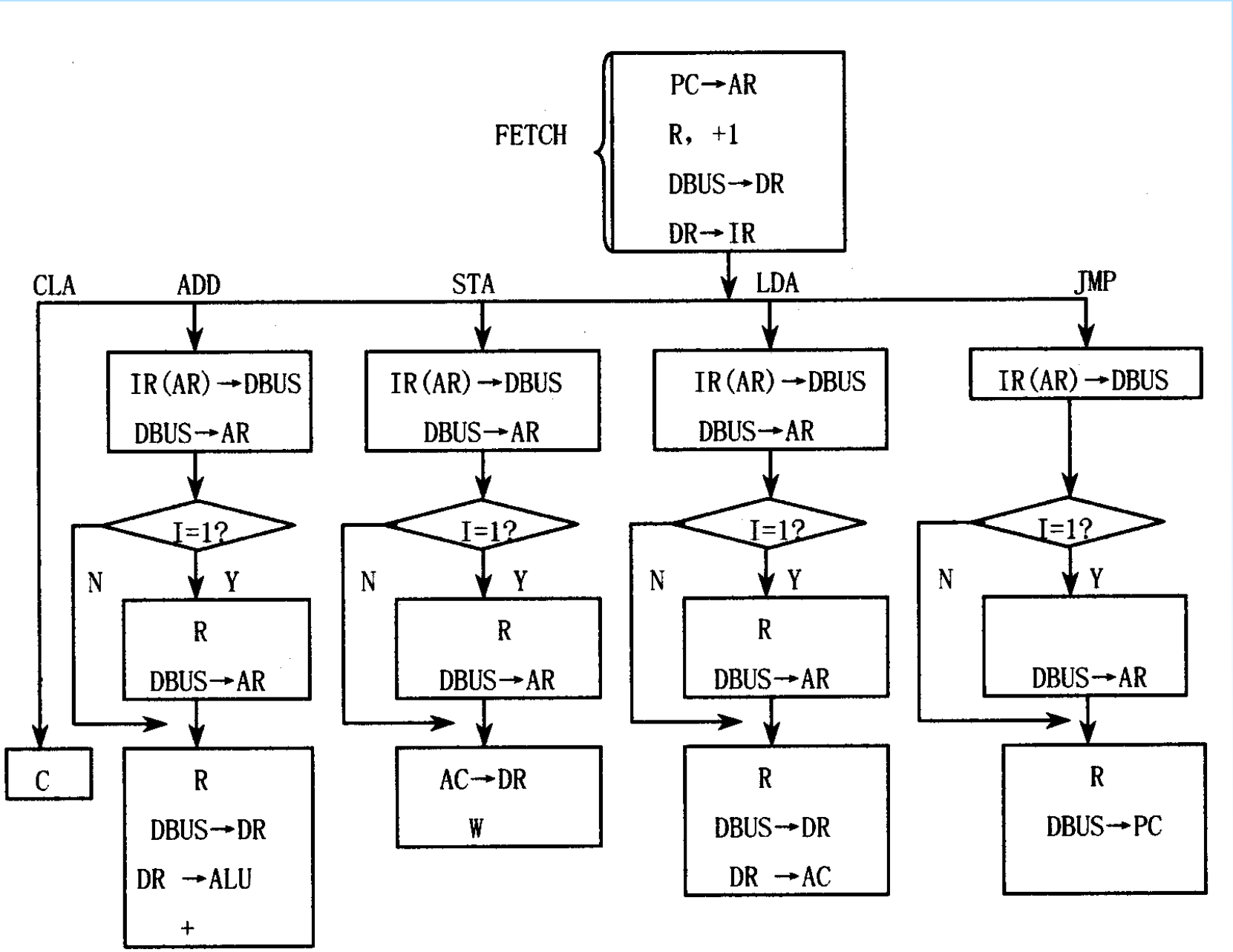
4）写入控制存储器

- 将二进制表示的全部微程序写入控制存储器。

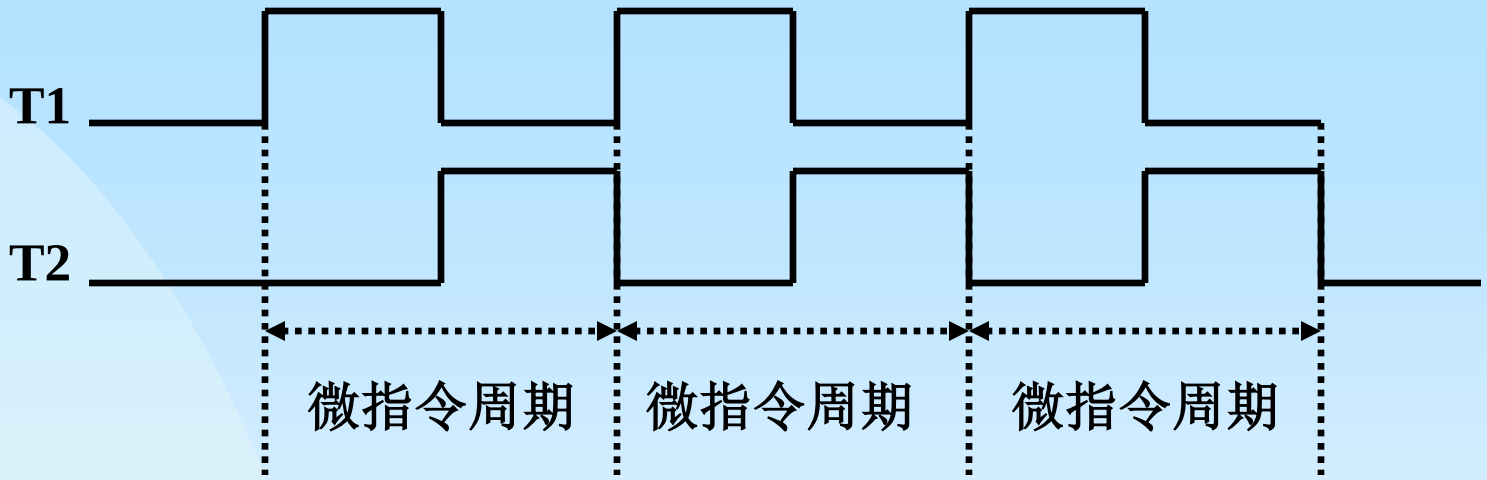
例. CPU结构框图如下图所示，设计以下几条指令的微程序控制器。



解：1）根据CPU结构框图写出指令的微操作流程图。



2）选用同步方式，采用二相控制时序。



3）对微命令进行编码。

xx	xx	x	x	x	x
00:不操作 01:DR→IR 10:DR→AC 11:DR→ALU	00:不操作 01:DBUS→AR 10:DBUS→DR 11:DBUS→PC	0:不操作 1:PC→AR	0:不操作 1:IR(AR) →DBUS	0:不操作 1:AC→DR	0:不操作 1:+1

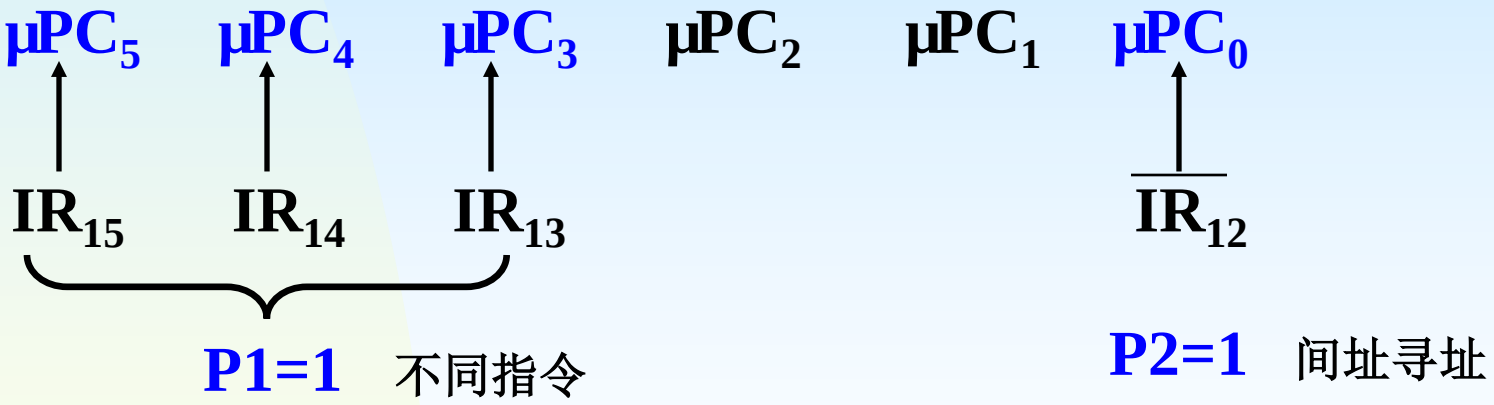
x	x	x	x	x
0:不操作 1:C	0:不操作 1:+	0:MREQ=0 1:MREQ=1	0:R/W=0 1:R/W=1	0:微命令 1:转移微指令

3）安排微指令地址

计数器法

- 地址尽量连续，在分支处加一转移微指令，有两种转移情况，用P1、P2来控制。
- 转移地址修改方案(转移微指令)：

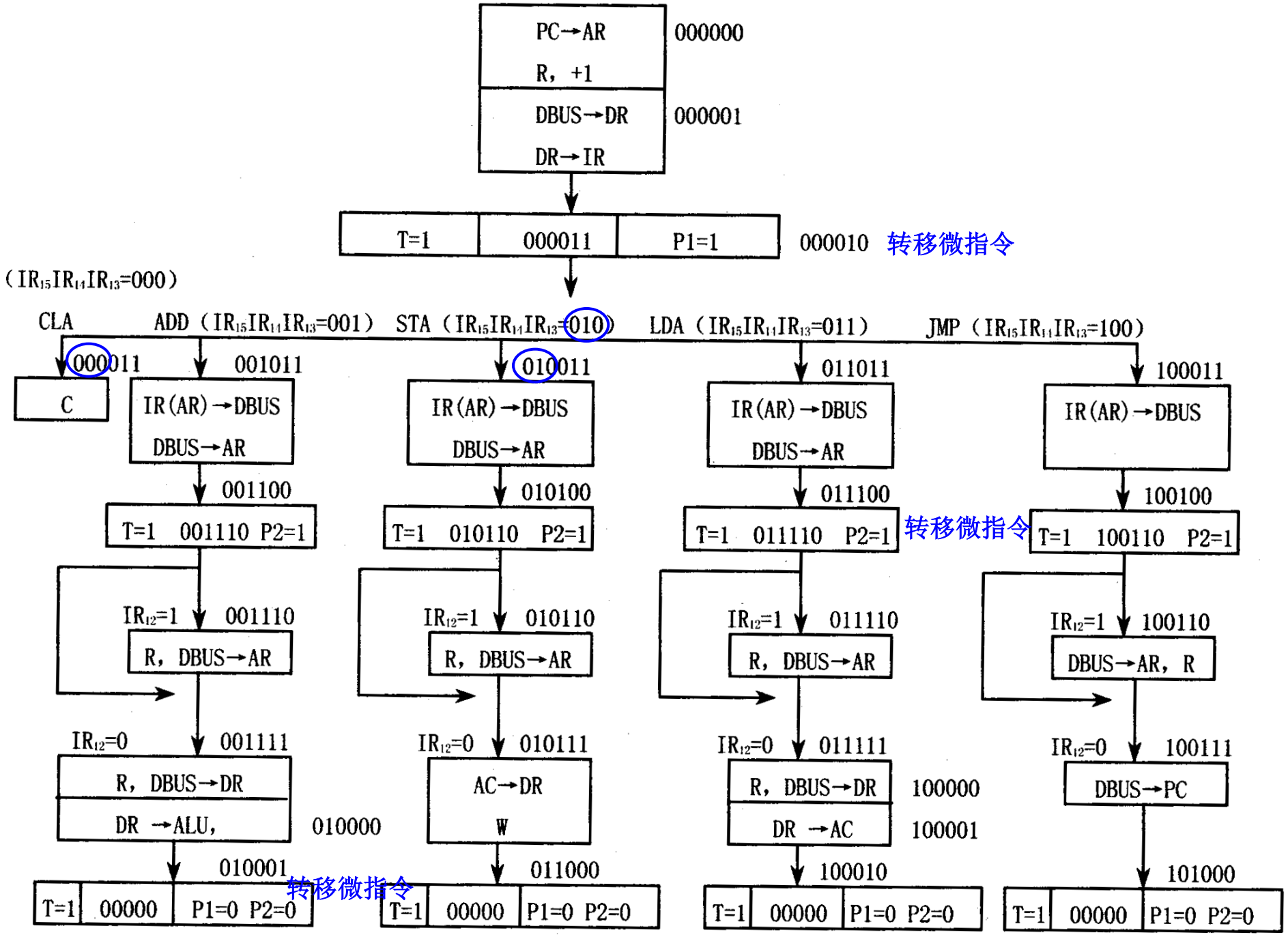
转移微指令标志T	转移地址($A_5A_4A_3A_2A_1A_0$)	转移控制(P1 P2)



- 地址转移逻辑表达式为：

$\mu PC_5 = IR_{15}\,P1T$　　$\mu PC_4 = IR_{14}\,P1T$　　$\mu PC_3 = IR_{13}\,P1T$

$\mu PC_0 = \overline{IR}_{12}P2T$


PC→AR
R, +1
000000
DBUS→DR
DR→IR
000001
T=1
000011
P1=1
000010
转移微指令
(IR15IR14IR13=000)
CLA
ADD (IR15IR11IR13=001)
STA (IR15IR14IR13=010)
LDA (IR15IR11IR13=011)
JMP (IR15IR11IR13=100)
000011
C
001011
IR(AR)→DBUS
DBUS→AR
001100
T=1 001110 P2=1
IR12=1
001110
R, DBUS→AR
IR12=0
001111
R, DBUS→DR
DR →ALU,
010000
010001
T=1 00000 P1=0 P2=0
转移微指令
010011
IR(AR)→DBUS
DBUS→AR
010100
T=1 010110 P2=1
IR12=1
010110
R, DBUS→AR
IR12=0
010111
AC→DR
W
011000
T=1 00000 P1=0 P2=0
011011
IR(AR)→DBUS
DBUS→AR
011100
T=1 011110 P2=1
转移微指令
IR12=1
011110
R, DBUS→AR
IR12=0
011111
R, DBUS→DR
DR →AC
100000
100001
100010
T=1 00000 P1=0 P2=0
100011
IR(AR)→DBUS
100100
T=1 100110 P2=1
IR12=1
100110
DBUS→AR, R
IR12=0
100111
DBUS→PC
101000
T=1 00000 P1=0 P2=0

0 0 1 1 0 0 1 0 0 1 0 0 0 0

1 0 0 0 0 0 0 0 0 1 0 0 0 1

转移微指令 R/W MREQ + C +1 AC→DR IR(AR)→DBUS PC→AR DBUS→AR DR→IR

DBUS→DR DR→AC

DBUS→PC DR→ALU

2 1 00011 1 0 0000

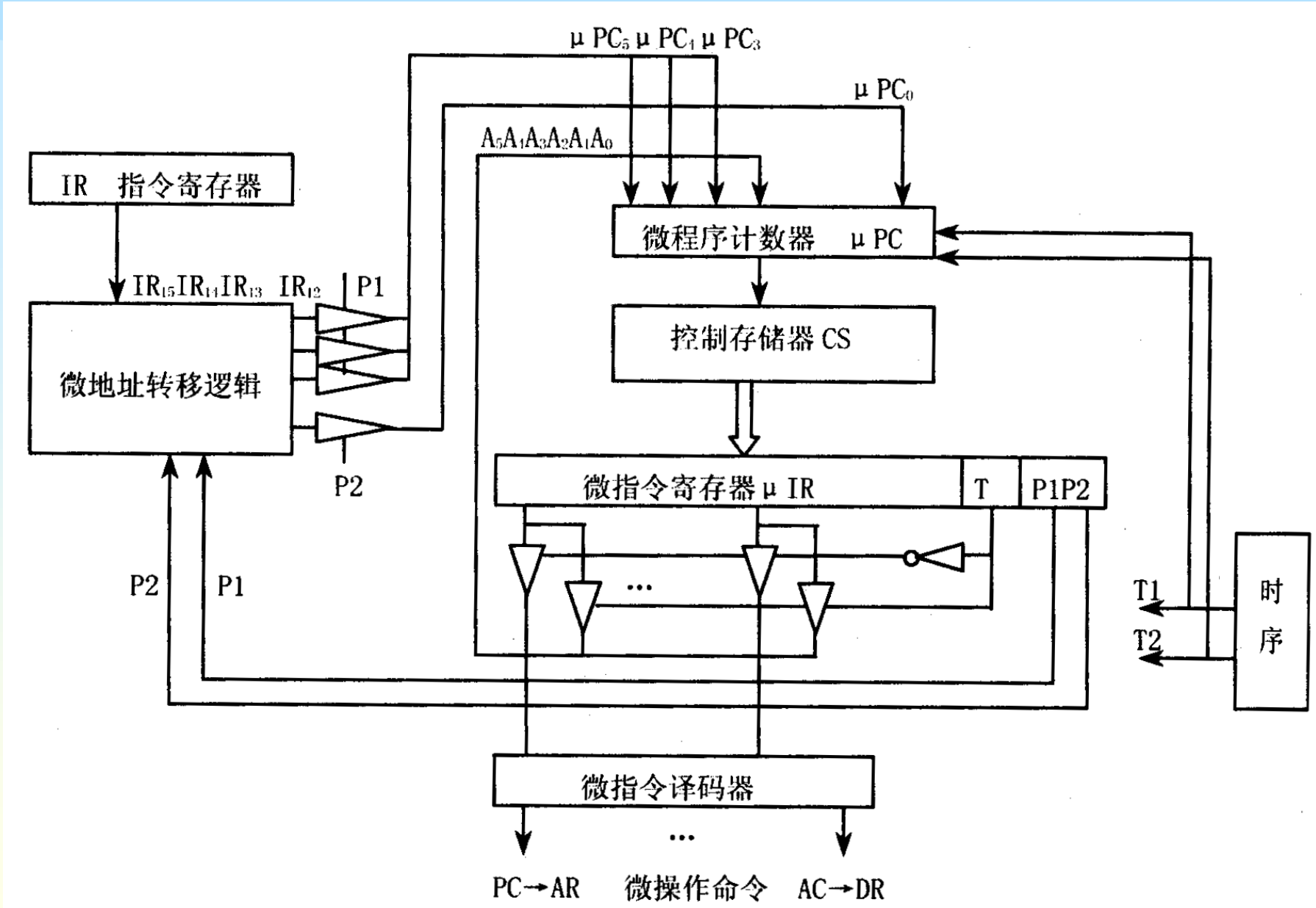
转移微指令标志T 转移地址(A5 A4 A3 A2A1 A0) 转移控制(P1 P2) 0000

3 0 0 0 0 1 0 0 0 0 1 0 0 1

11 0 0 0 0 0 0 0 1 0 0 1 0 0

12 1 001110 0 1 0000

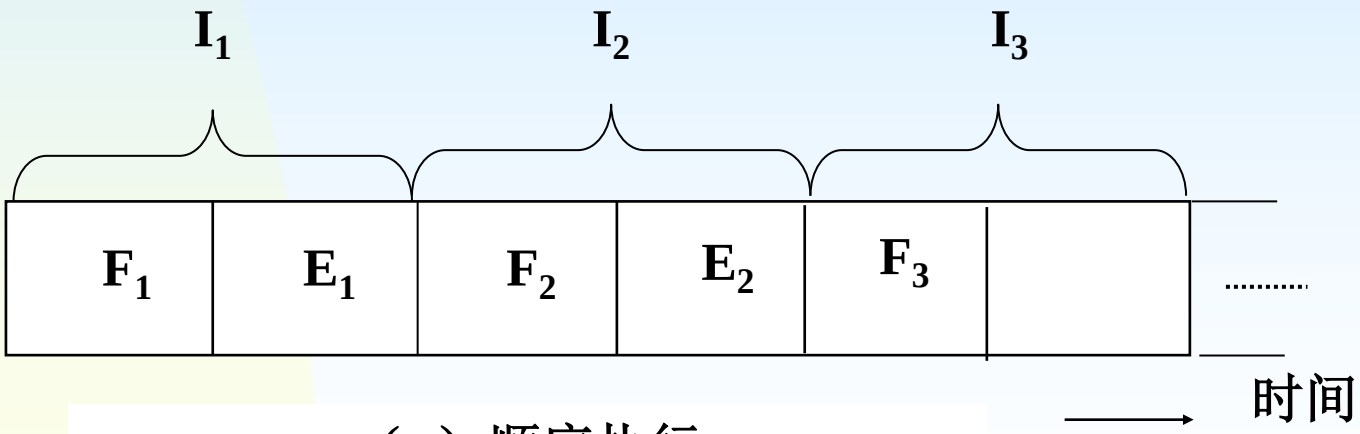
…


μ PC5 μ PC1 μ PC3
μ PC0
A5A1A3A2A1A0
IR 指令寄存器
微程序计数器 μ PC
IR15IR14IR13 IR12
P1
微地址转移逻辑
控制存储器 CS
P2
微指令寄存器 μ IR
T
P1P2
P2
P1
T1
T2
时序
微指令译码器
PC→AR
微操作命令
AC→DR

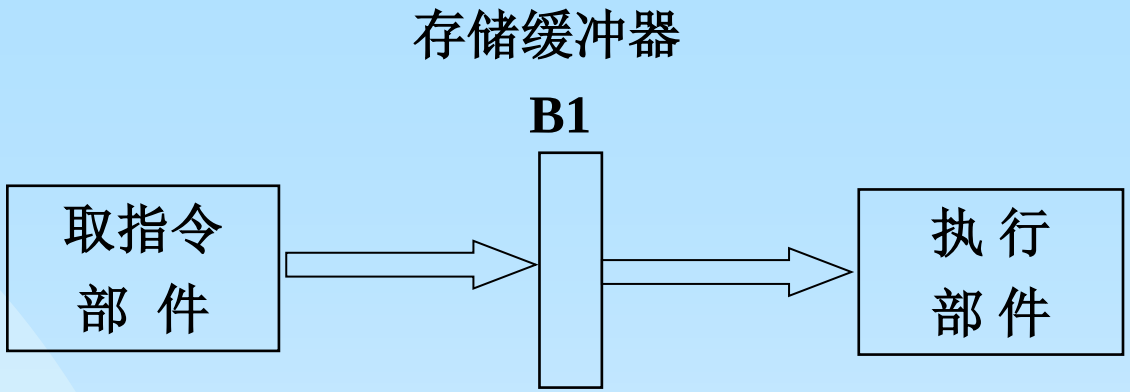
7.5 流水线处理器

- 流水线技术：将功能部件分离、执行时间重叠的一种技术；可有效地提高CPU性能；
- 指令流水与操作流水 。

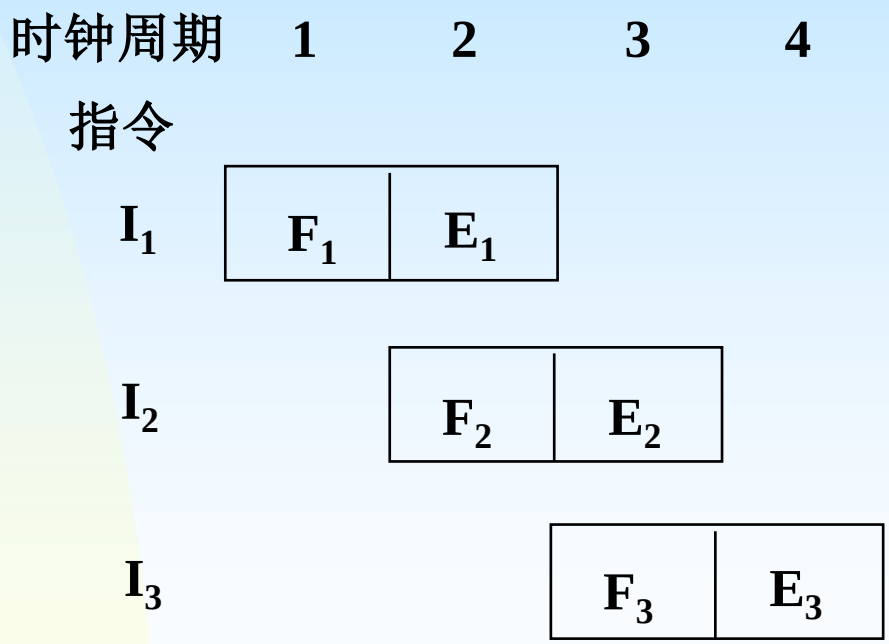
7.5.1 基本概念



（a）顺序执行

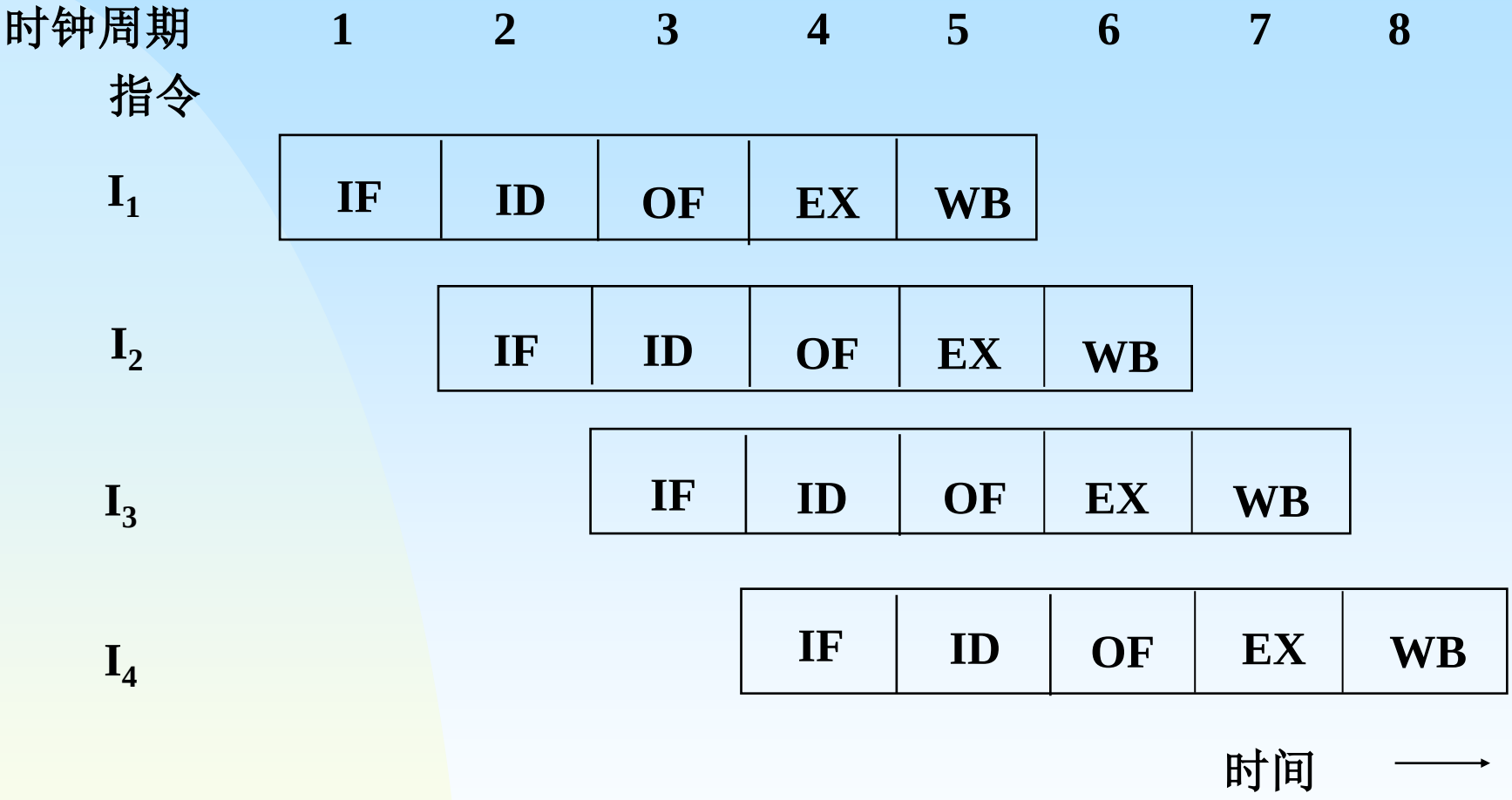


（b）硬件组织

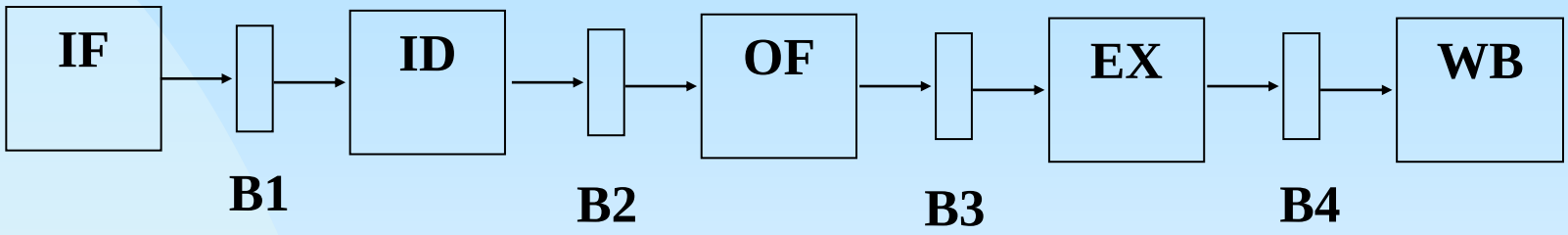


(c)流水线执行

图7.24 指令流水线的基本思想


时钟周期
1
2
3
4
5
6
7
8
指令
I_1
IF
ID
OF
EX
WB
I_2
IF
ID
OF
EX
WB
I_3
IF
ID
OF
EX
WB
I_4
IF
ID
OF
EX
WB
时间

(a) 流水线执行



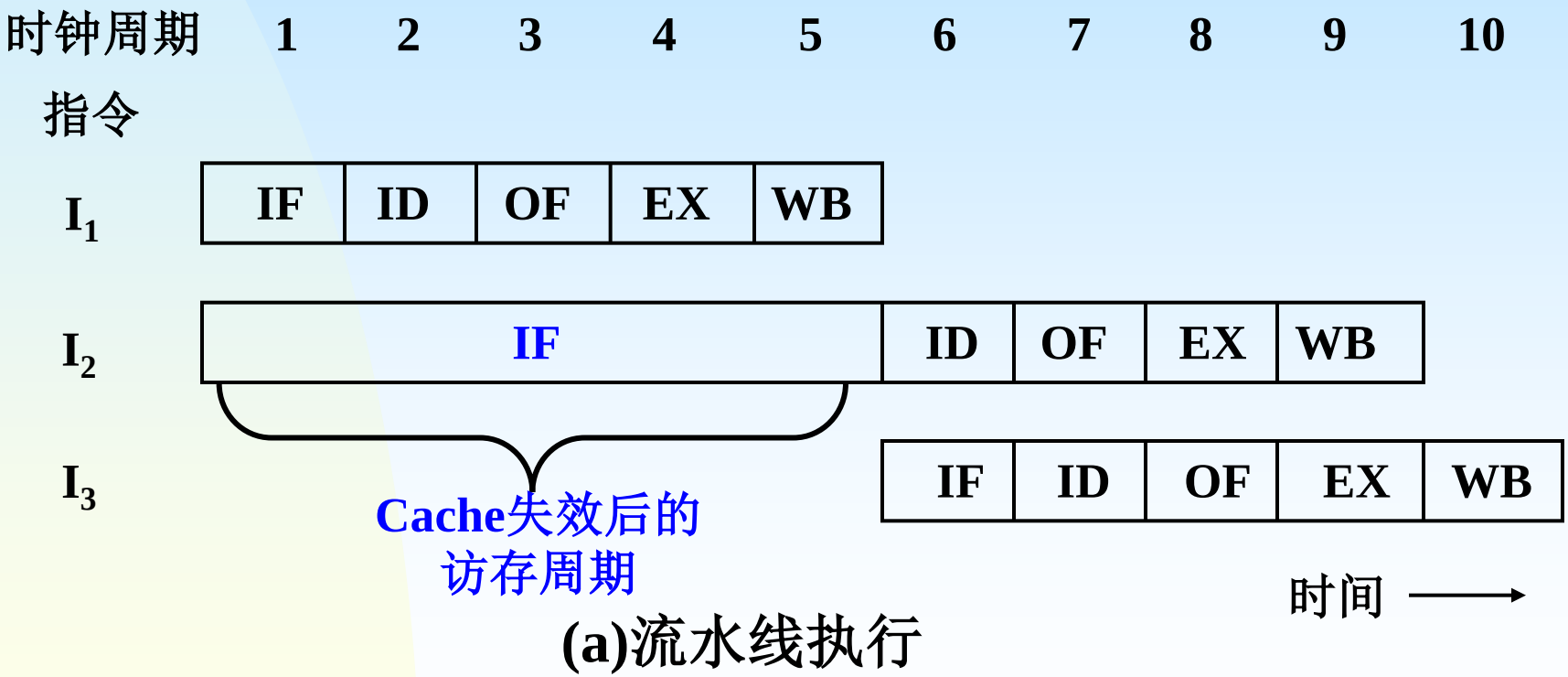
(b)硬件组织

图5.25 一个5段指令流水线

- 任何时候流水线上的某一段不能在一个时钟周期内完成，称流水线停顿。如访存和转移指令。

常用的改善技术

1. Cache的作用



(a)流水线执行

时钟周期 段	1	2	3	4	5	6	7	8	9	10
IF：	IF_1	IF_2	IF_2	IF_2	IF_2	IF_3				
ID：		ID_1	idle	idle	idle	ID_2	ID_3			
OF：			OF_1	idle	idle	idle	OF_2	OF_3		
EX：				EX_1	idle	idle	idle	EX_2	EX_3	
WB：					WB_1	idle	idle	idle	WB_2	WB_3

（b）各段完成的时序

图5.26　流水线停顿现象

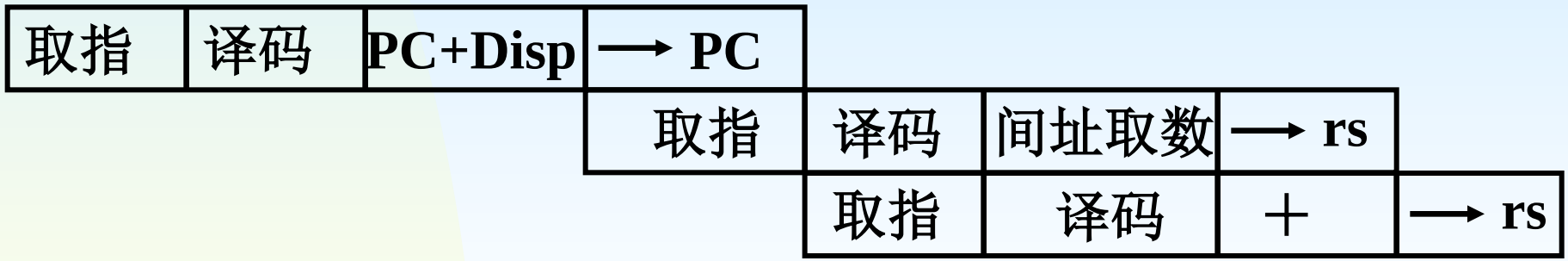
例2　某计算机采用4级流水线（取指、译码、执行、送结果），其中译码可同时完成从寄存器取数的操作，假设存储器的读/写操作（允许同时取指和取数）可在一个机器周期内完成，问顺序执行下面三条指令，共需多少周期？

（1）JMP　　Disp　　(相对寻址）

（2）LOAD　　rs　　@rs1　　（间接寻址）

（3）ADD　　rs　　rs1　　（寄存器寻址）

解：无实现动态转移预测



共8个周期

2. 指令预取

- 转移指令也会造成流水线停顿；
- 指令预取技术：提前将指令取出并放入指令队列中；指令发射部件将队列头上的指令发送到空闲的执行部件去执行。

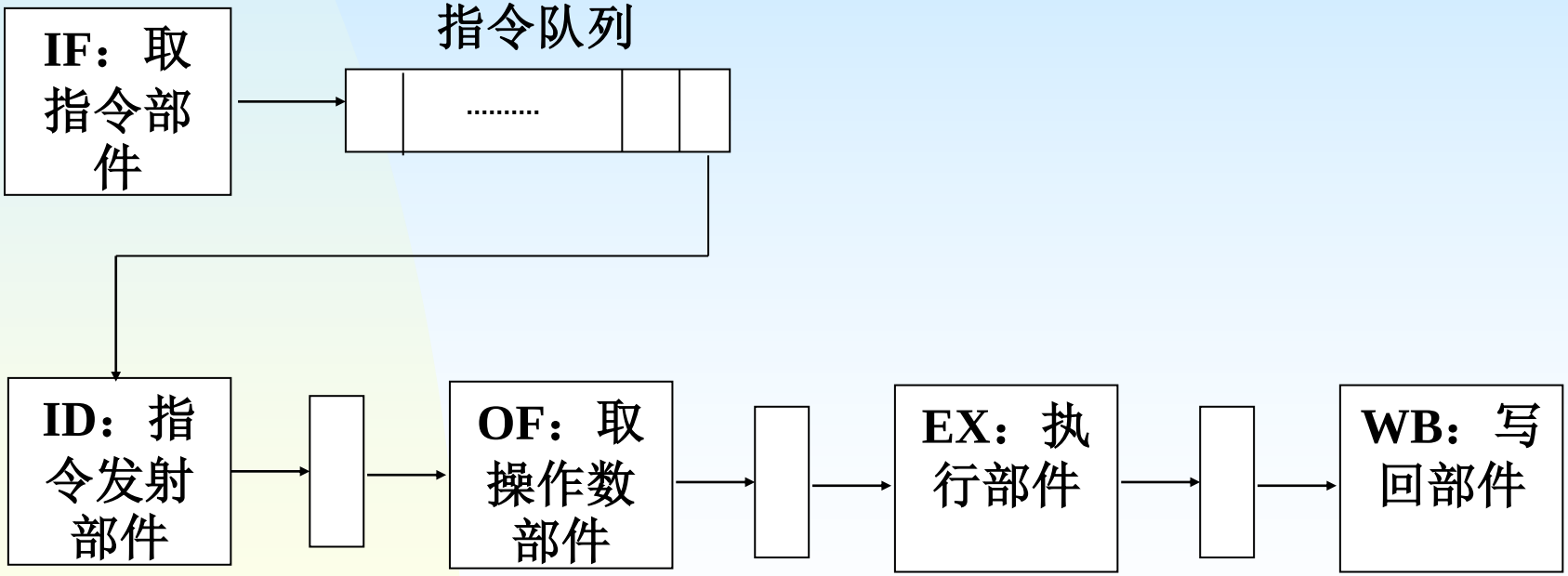


图7.27 带有指令队列的流水线硬件组织

7.5.2 访存冲突和相关处理

1. 访存冲突

- 在流水线上，取指令、取操作数都要访问主存；要求CPU必须能够同时访问主存的两个单元。

◆ 解决访存办法

1）设置分别存放指令和操作数的两个独立编址的主存；

2）采用多体交叉存储器，使两条相邻指令的操作数存放在不同的存储体内；

3）采用指令预取（指令缓冲）技术。

2. 控制相关

- 在流水线处理器中，当遇到条件转移指令时，只有前一条指令流出流水线时，转移条件才能形成，而此时后面的若干条指令早已流入流水线。

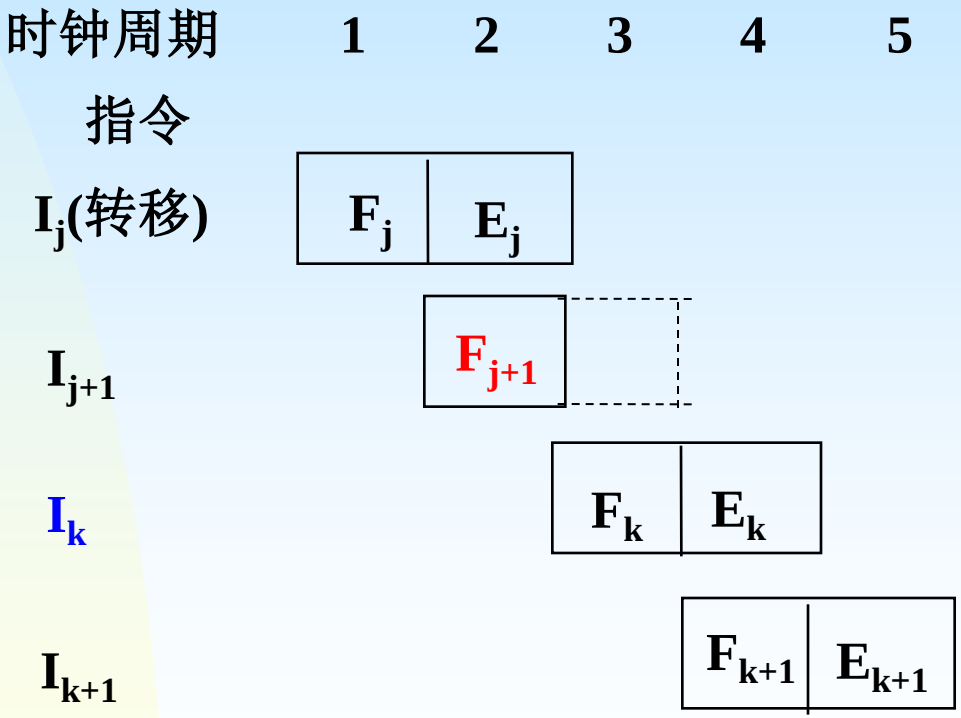


图7.28 由一条转移指令引起的空闲周期

转移相关技术

- 无条件转移指令，指令队列和高速Cache技术可以很好地解决控制相关问题。

1）延迟转移

- 通常将转移指令后面的指令位置称为转移延迟槽；
- 延迟槽中的指令总是被取出，且被完整地执行；延迟槽里可以安排一些有用的指令或用无操作指令（NOP）填充；
- 延迟转移技术的性能取决于能否有效地对指令序列进行重新排序；大部分编译程序只使用一个延迟槽。

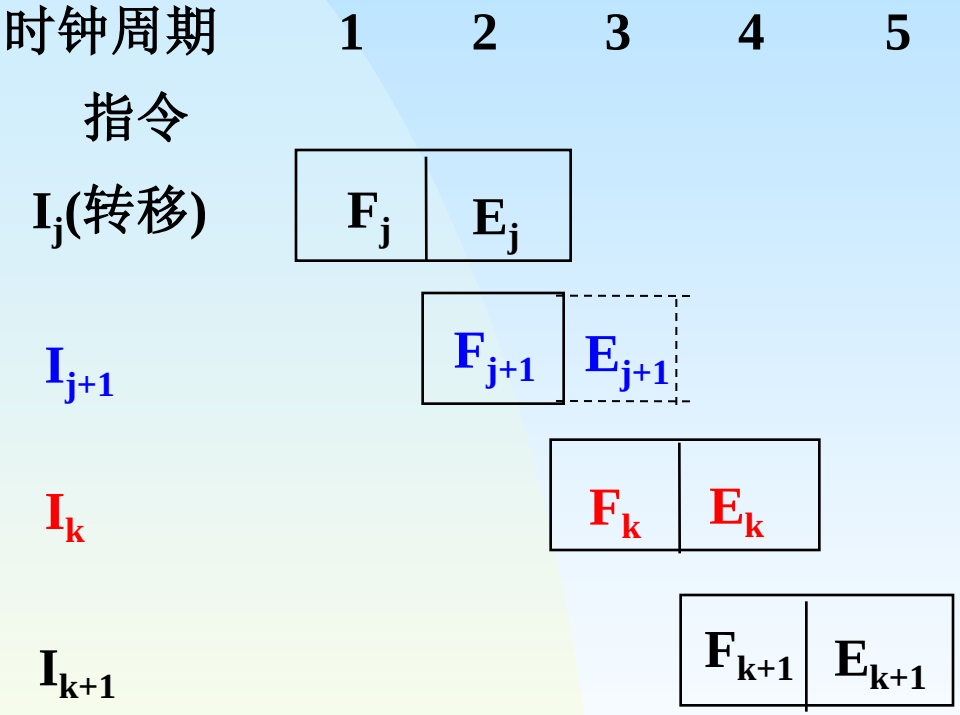


图7.28 由一条转移指令引起的空闲周期

```
LOOP    SL    R1 ；R1左移1位
        Dec   R2 ； R2←(R2-1)
        BNZ   LOOP ；不为零，转移
NEXT    Add   R1+R3
```

(a) 原始程序段

```
LOOP    Dec   R2
        BNZ   LOOP
        SL    R1
NEXT    Add   R1+R3
```

(b) 指令重排序后

图7.29 用于延迟转移的指令重排序

2）转移预测

- 转移预测技术试图预先判断一个转移是否会发生。

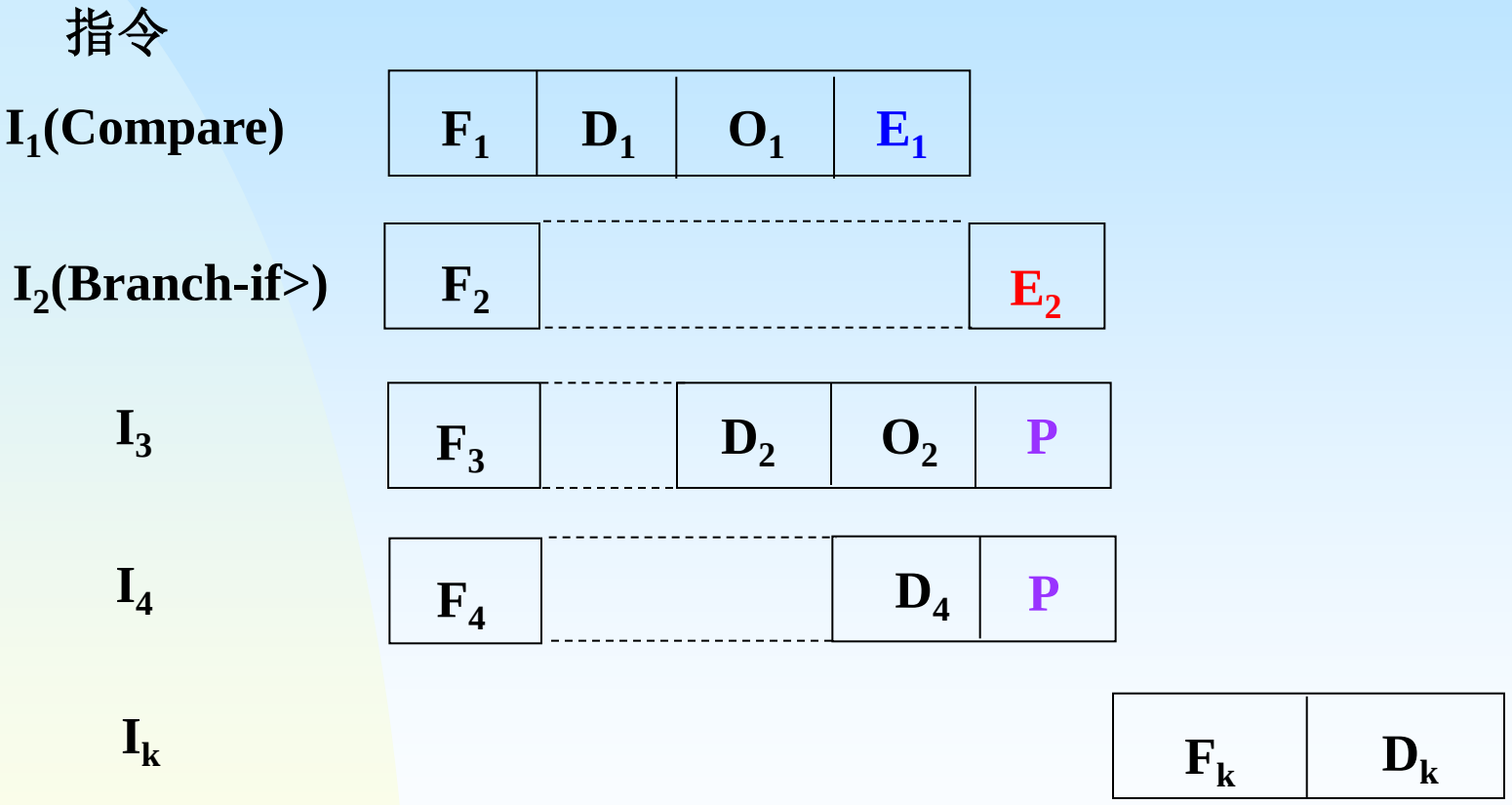


图7.30 在4段流水线中使用转移预测的例子

转移预测的实现方法

- 编译程序来完成，此时预测被编码在转移指令中。指令的操作码字段指明转移被预测为发生还是不发生。称为静态转移预测。
- 通过处理器硬件来判断一个转移指令每次发生的可能性。它可以记录上次转移预测的结果，并以此为依据来预测下次转移是否会发生。
- 转移目标缓冲器**BTB（Branch Target Buffer）**来保存更多的信息以提高预测的准确度和程序执行的性能。同样的转移指令在不同的场合可能有不同的预测结果。称为动态转移预测。

3．数据相关

- 一条指令的源操作数来自于先前执行的指令结果；
- 数据相关(数据冒险或数据依赖)。指程序有先后顺序的几条指令对共享变量(同一个寄存器或存储单元)的读、写操作；由于操作顺序上的原因，流水线可能停顿或输出错误的结果。
- 数据相关会造成流水线停顿（共用同一个寄存器）

I_1：　Add　R1，R2，R3
I_2：　SL　　R3

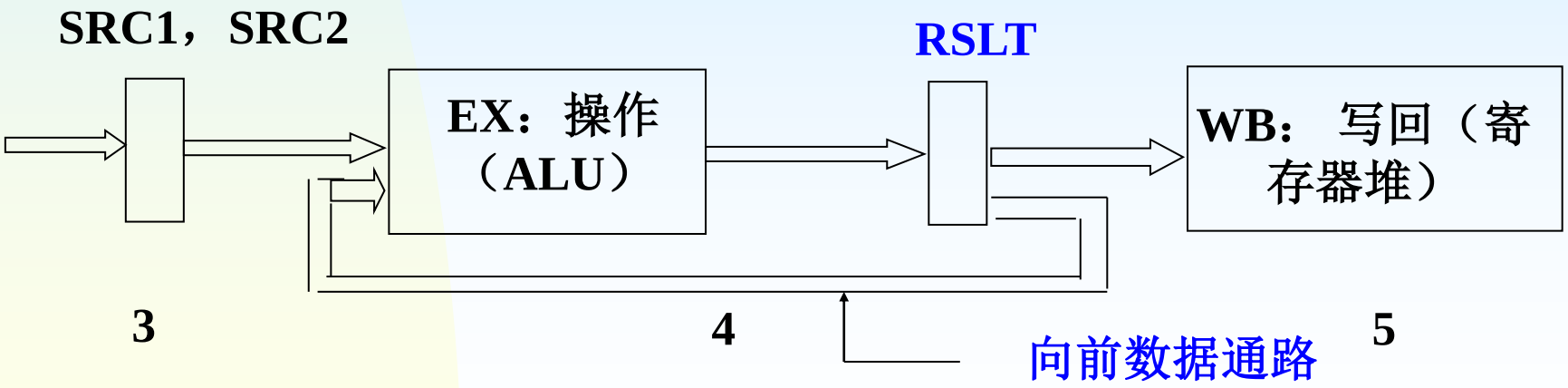


图7.31 源、目的寄存器在流水线中的位置

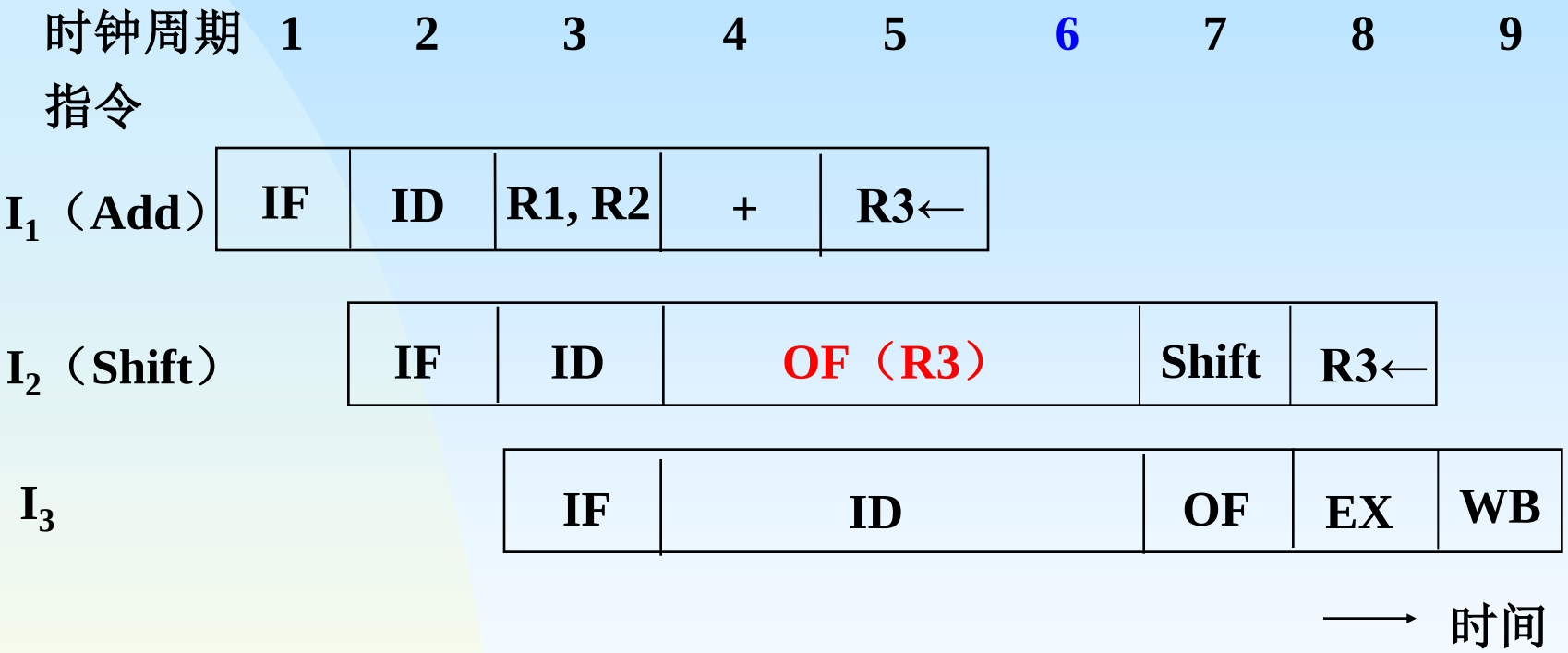


图7.32　数据相关引起流水线中断

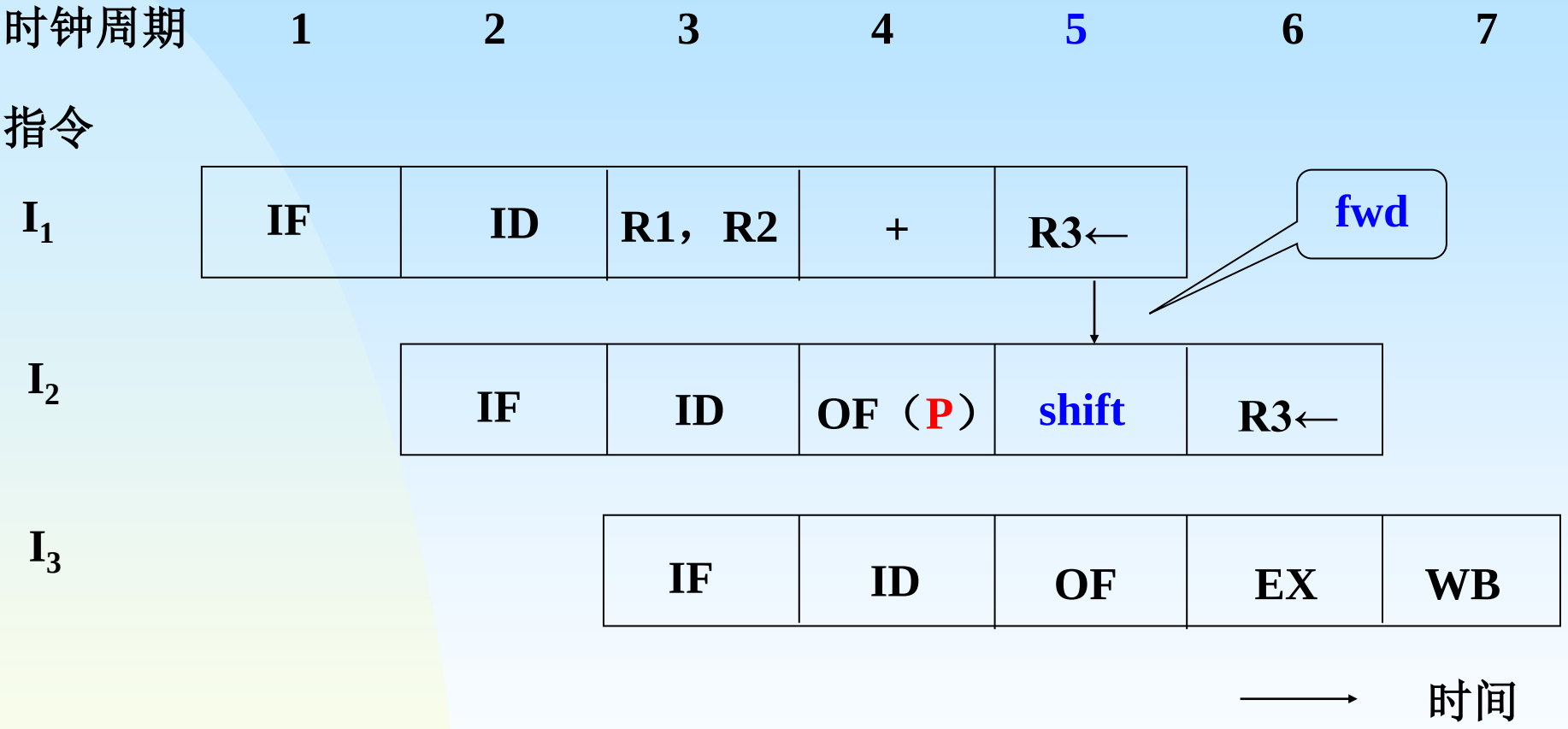


图7.33 使用向前数据通路解决数据相关

- 操作数来自于主存时发生的数据相关

I_1：　Load　（R1），R2　；　（（R1））→R2

I_2：　Add　R2，R3，R4　；　R2+R3→R4

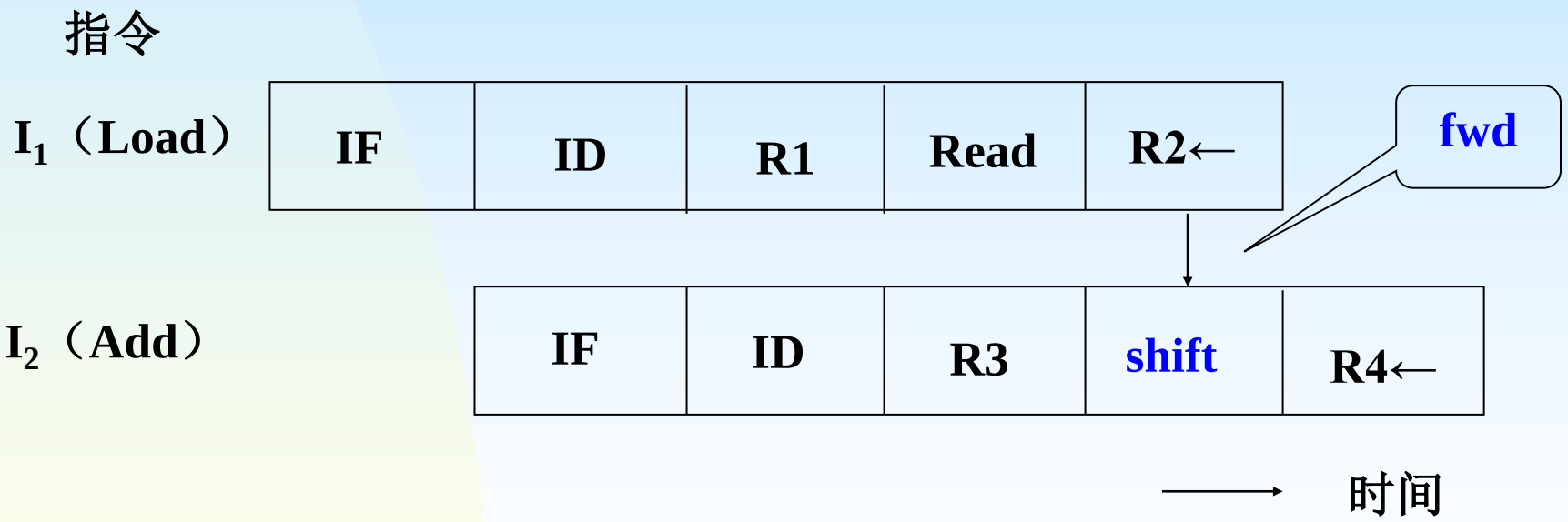


图7.34　在读操作中将操作数前移

- 有三类数据相关 :读后写 (RAW) (Read Ahead of Write) 相关 、写后读(WAR)相关 、写后写(WAW)相关 。
- 这三类数据相关(冒险)如下图所示:

- 在指令流水线中很容易出现RAW冒险, 而WAR和WAW两类冒险在指令流水线中一般不会出现。

7.6 超标量处理器

- 超标量处理器是指在处理器中安排多个指令执行部件，多条指令可以被同时启动和独立执行。

7.6.1 概述

- 在超标量处理器中配备了多个功能部件和指令译码电路，还有多个寄存器端口和总线，以便同时执行多条指令。
- 在超标量流水线中，并行执行的流水线条数称为超标度。
- 编译程序如何安排哪些指令可以并行执行对于充分发挥超标量结构的优势也是非常重要的。

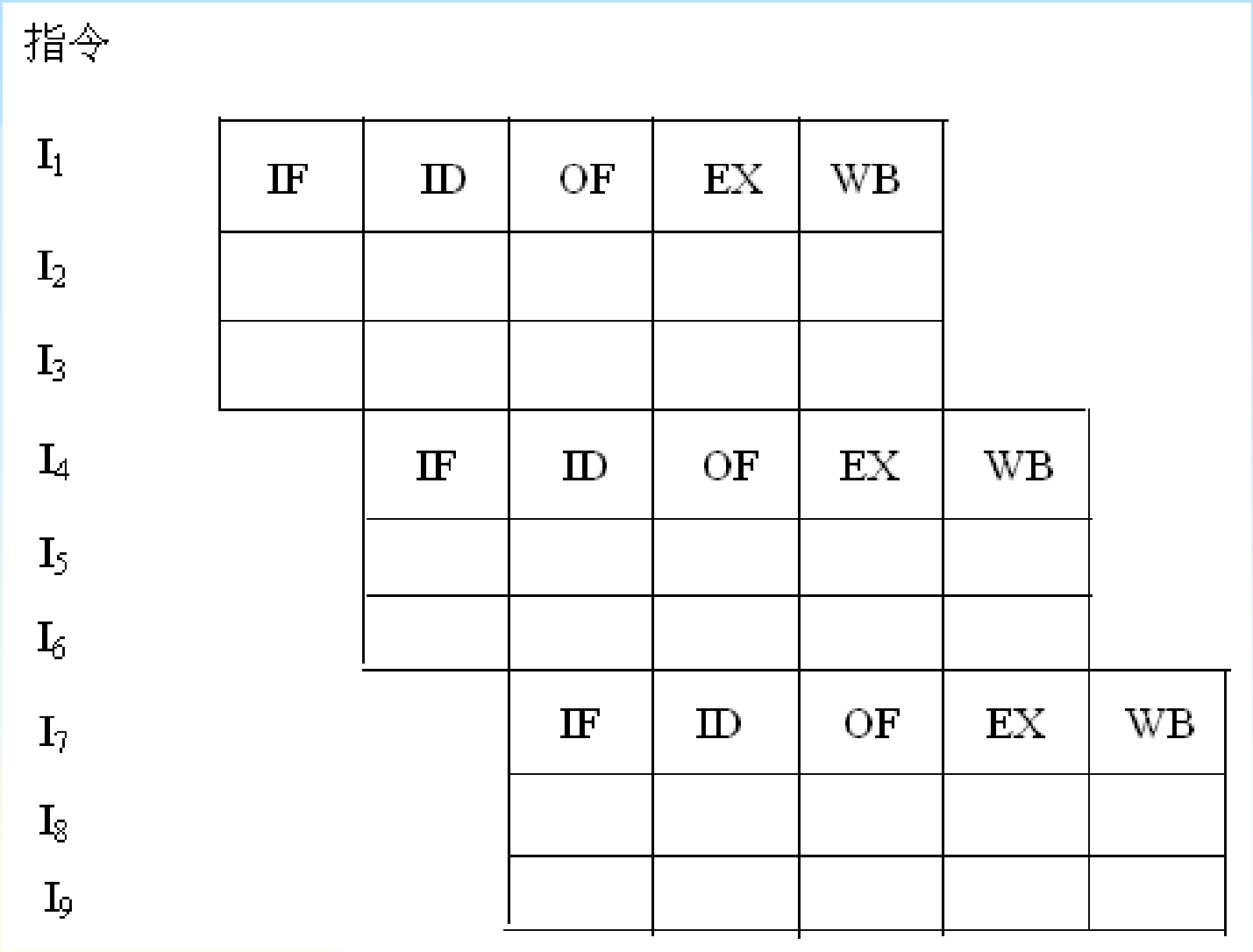


图7.35　超标量处理器

1. 超流水线技术

- 可以在一个基本时钟周期内分时发射多条指令，这种技术被称为超流水技术；
- 在一个时钟周期内也有多条指令流入流水线；
- 超流水线处理器对硬件速度要求较高，即以时间换取空间；
- 将超标量技术和超流水技术结合起来，形成超标量超流水线处理器。

2. 超长指令字技术

- 超长指令字技术与超标量技术类似，都采用多个功能部件并行处理多条指令；
- **VLIW是由编译器经过优化策略，将多条能够并行执行的指令合并成一条具有多个操作码的超长指令；**
- **VLIW相对于超标量具有更高的并行能力，对编译技术有很高的要求，且希望Cache的容量尽量大。**

7.6.2 指令发射策略

- 指令发射是指启动指令执行处理器功能的过程；发射指令所采用的协议或规则称为指令发射策略。
- 本质就是处理器试图在当前执行点前面查找能进入流水线并执行的指令。
- 指令发射策略依赖于指令的排序，有三类排序：

1）取指令顺序；

2）执行指令顺序；

3）寄存器和存储器位置改变的顺序。

- 超标量发射策略也有三类：

  1）按序发射，按序完成；

  2）按序发射，无序完成；

  3）无序发射，无序完成。

1. 按序发射按序完成

- 按序发射按序完成策略是严格地按能实现顺序执行的顺序发射指令，并以同样顺序写结果。

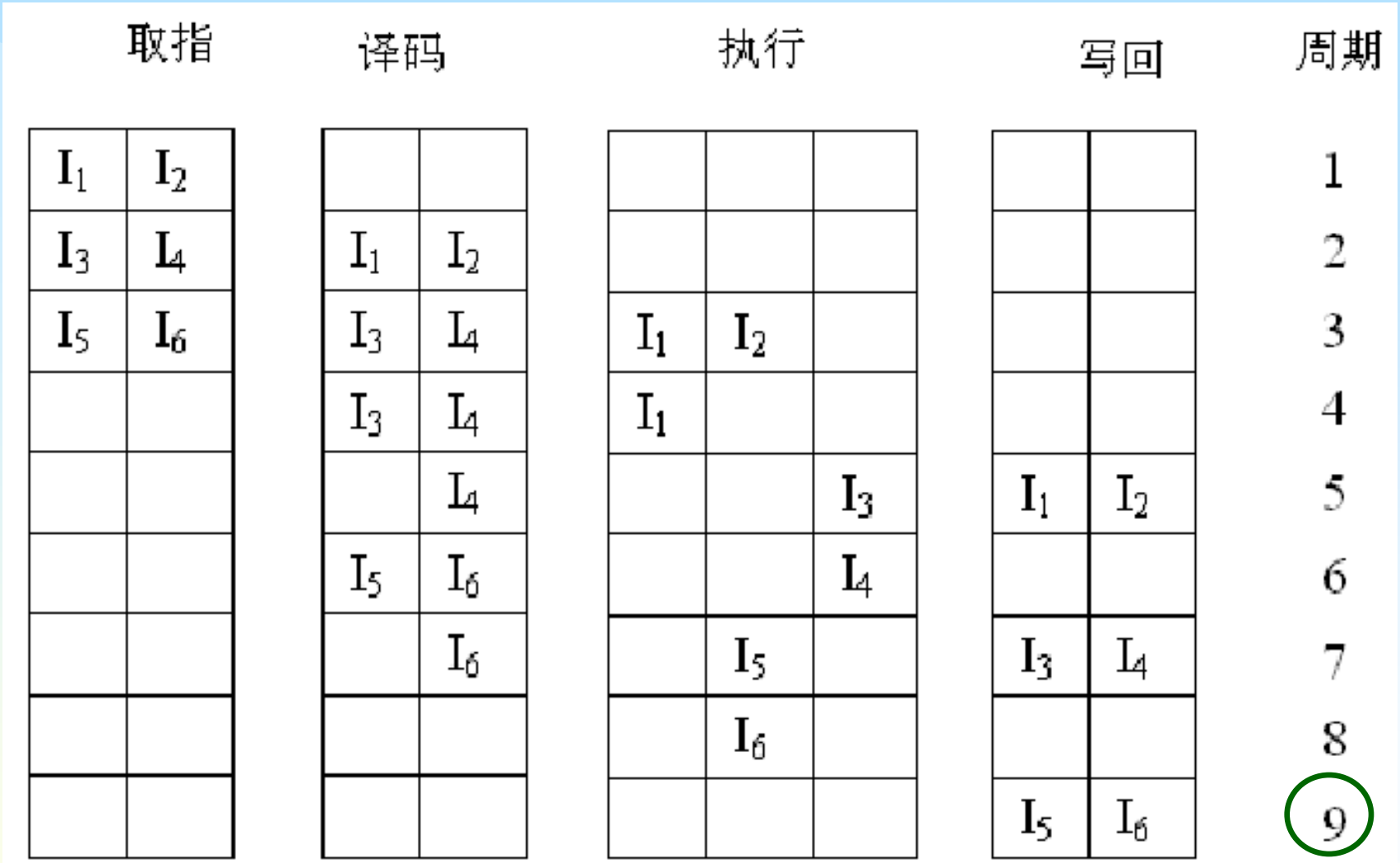


图7.36 按序发射按序完成

2. 按序发射无序完成

- 在标量RISC机器中，常采用无序完成来改善那些需要多个周期的指令性能。

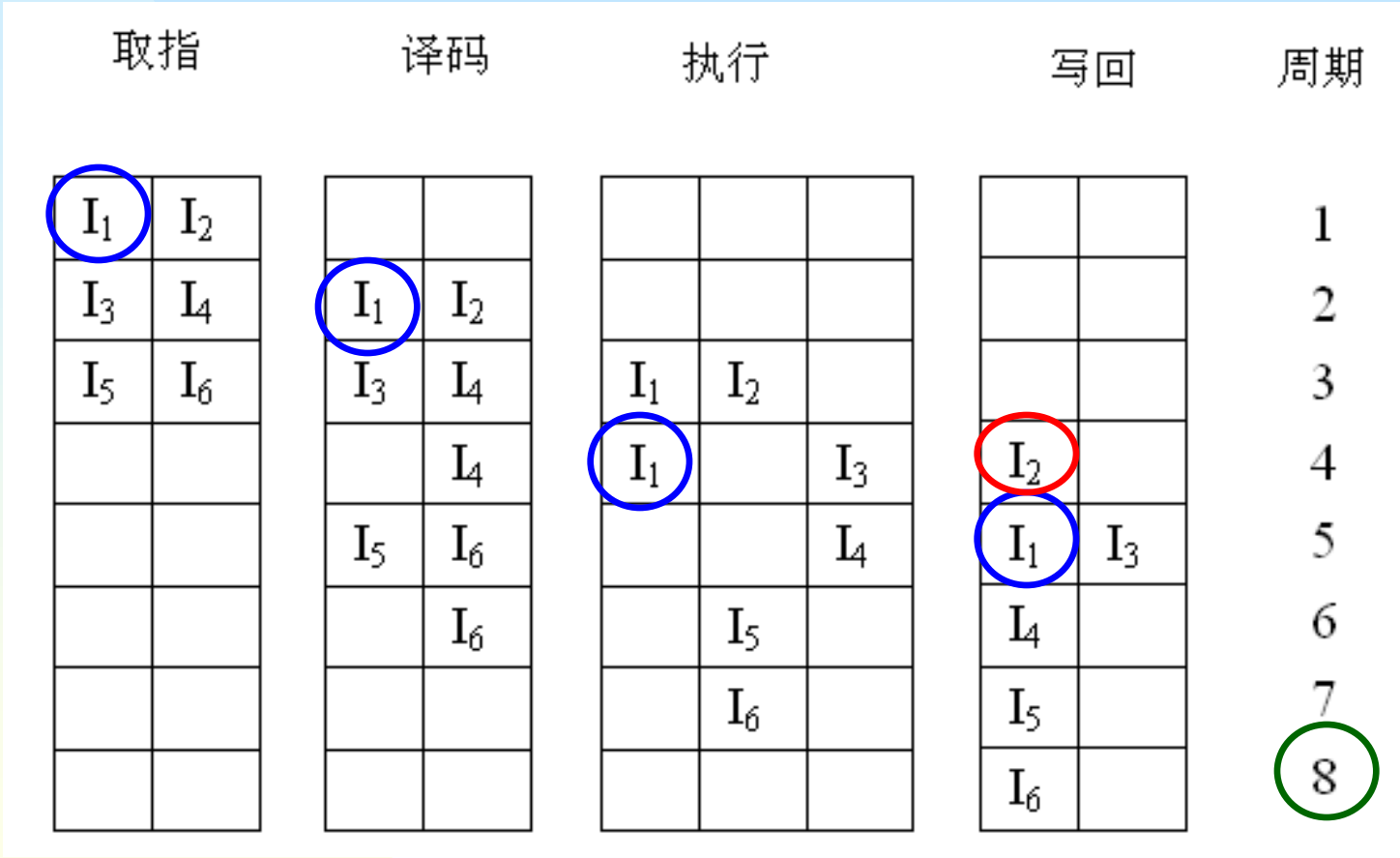


图7.37 按序发射无序完成

3. 无序发射无序完成

- 后续指令中可能存在与流水线中的指令无冲突的指令，它们可被引入流水线中；称无序发射。

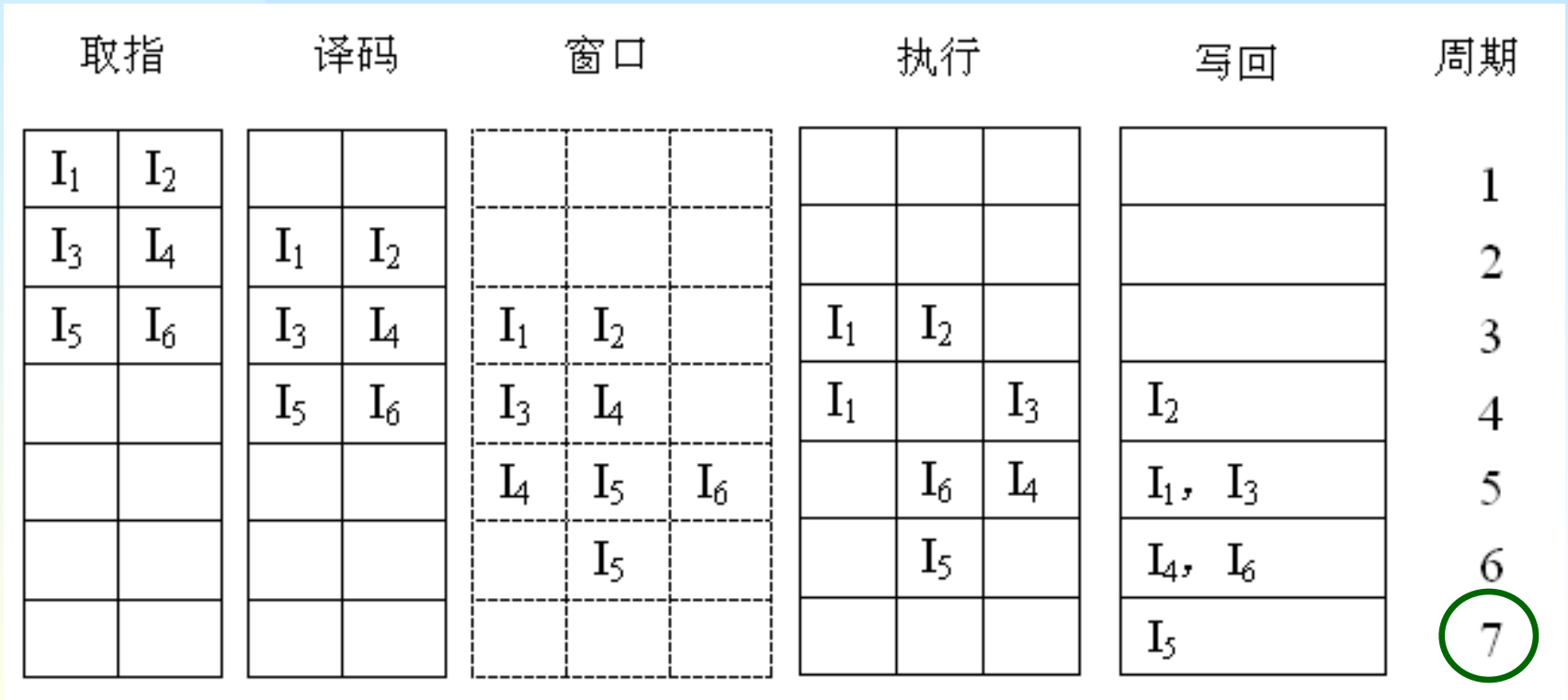


图7.38 无序发射无序完成

7.7 Pentium和PowerPC

- **Pentium处理器代表了复杂指令集计算机（CISC）多年的设计经验的结晶。**
- **PowerPC是目前市场上基于RISC的功能最强大、设计最好的系统之一。**

7.7.1 Pentium

1. Pentium处理器

- Intel公司x86系列的第五代微处理器；
- 主要寄存器为32位宽，是一个32位微处理器；
- 连接存储器的外部数据总线宽度为64位，每次可同时传输8个字节；
- Pentium外部地址总线宽度是36位，一般使用32位，故物理地址空间为4GB。
- Pentium采用U，V两条指令流水线的超标量结构，内部有分立的8KB指令cache和8KB数据cache，外部还可接256～512KB的二级cache；

- 大多数简单指令是用硬连接技术实现，一个时钟周期执行完；
- Pentium的CISC特征是主要的，属于CISC结构处理器。

▲ 以BTB实现动态转移预测

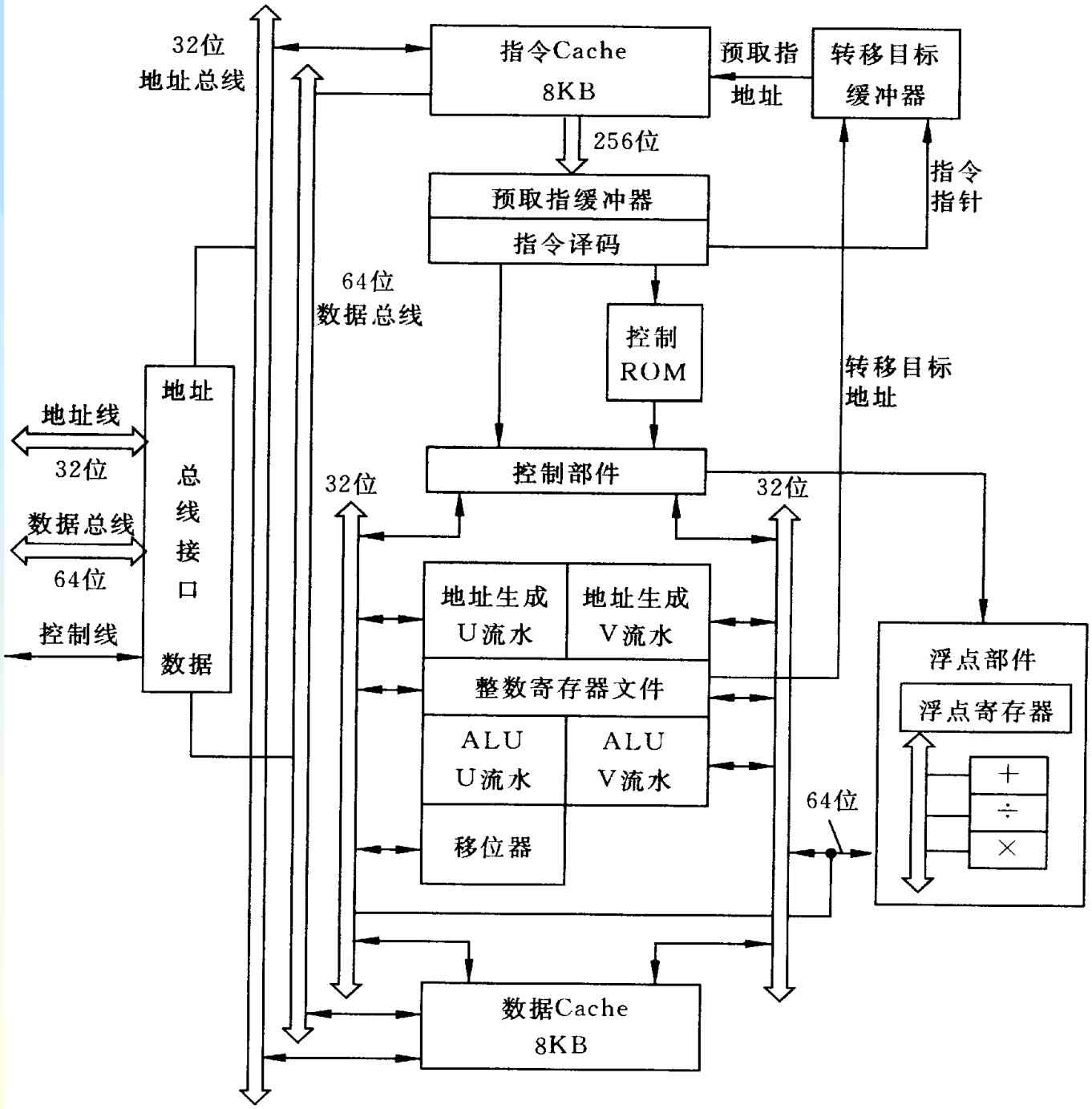
- 减少由于过程相关性引起的流水线性能损失。

▲ 后续产品Pentium MMX、Pentium Pro、Pentium II、Pentium III集成了MMX技术

- 增强了音频、视频和图形等多媒体应用的处理能力，加速数据加密和数据压缩与解压的过程；

▲ 多处理器技术、虚拟分页扩展技术、单指令多数据流（SIMD）技术等融入处理器芯片。

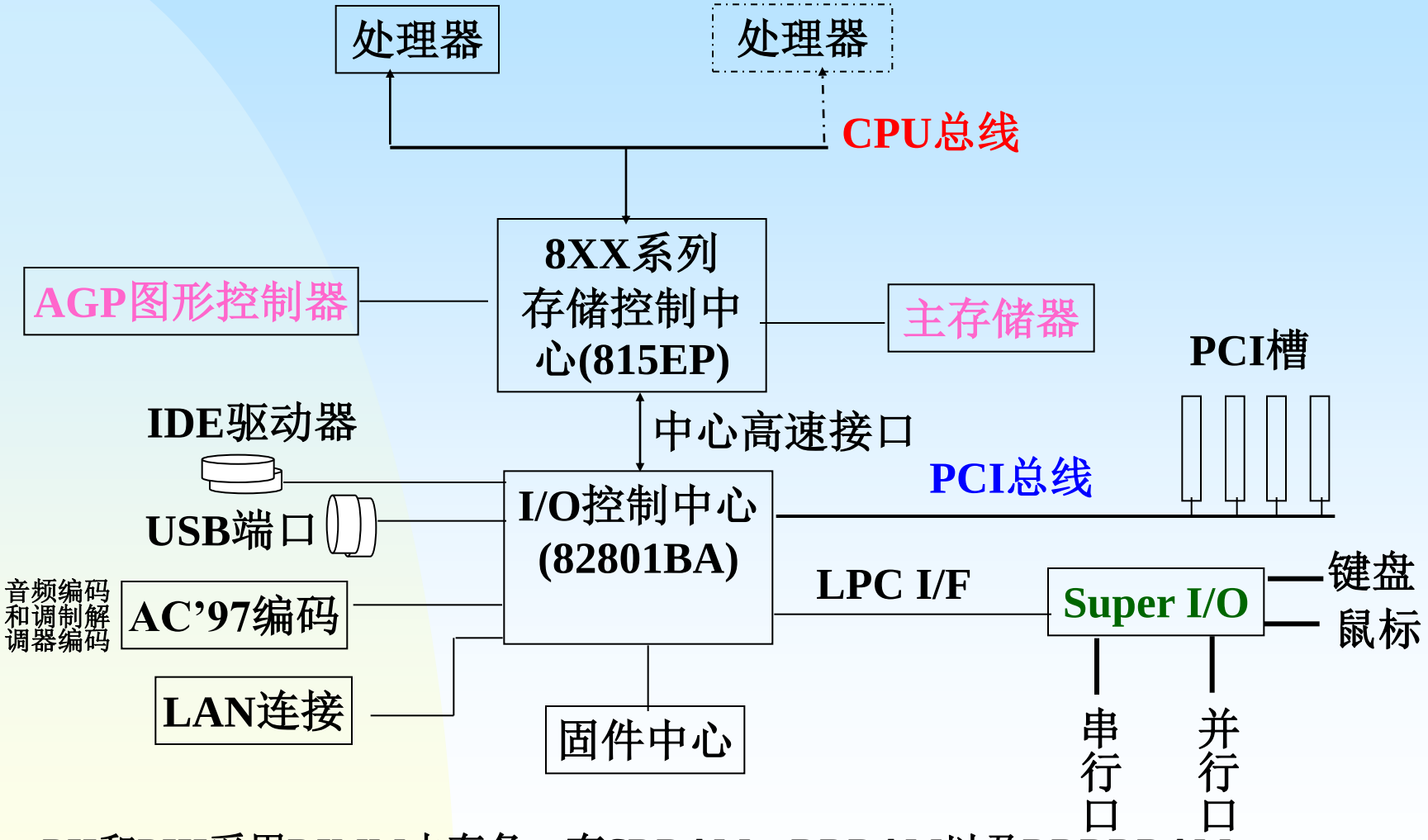
Pentium微处理器逻辑图



▲ Pentium有四类寄存器组：

- 基本结构寄存器组
  通用寄存器
  段寄存器
  指令指针
  标志寄存器
- 浮点部件寄存器组
  数据寄存器堆
  控制寄存器
  状态寄存器
  标记字
  事故寄存器
- 系统级寄存器组
  系统地址寄存器
  控制寄存器
- 调试和测试寄存器组

中心结构的Pentium III微机基本结构



- PII和PIII采用DIMM内存条，有SDRAM、RDRAM以及DDRDRAM

7.7.2 PowerPC

- PowerPC是IBM联合Motorola和Apple于20世纪90年代推出的一个RISC超标量结构的系列计算机。
- 两个特点：

1）高性能、低价格，并且很容易仿真CISC指令集；

2）是可扩展性强，从嵌入式设备、PC到大规模超级计算机等，PowerPC都能够提供整体解决方案。

1）指令队列和发送单元：存放从cache取来的指令。线宽为256位，一个周期可将8条指令(32位)装入指令队列。发送单元对队列中前4条指令进行分析，并发送到不同的指令执行单元。

2）分支处理单元：执行全部转移指令—用静态分析预测法，总是假定其中一支经常发生，而另一支不发生。此单元被设计成多数情况下转移不影响其它单元的执行步调，即零周期转移（zero-cycle）。

3）整数单元：进行算术与逻辑运算，可以在一个周期内完成加、减、移位、逻辑运算等简单指令，复杂指令(如乘除)需要多个周期。内设32×32位寄存器堆。

4）浮点单元：进行单/双精度的浮点运算。内含32个×64位寄存器，存放运算数据，并有反映运算结果的状态寄存器FPSCR。

5）存储管理单元：具有52位虚地址和32位实地址的寻址能力，它支持面向段、页及块的地址变换功能。

6）cache：统一的32KB的数据/指令cache，采用页组映象技术。

7）总线接口单元：定义了32位地址和32位数据的模式。总线协议支持流水、非流水和单独数据传送。

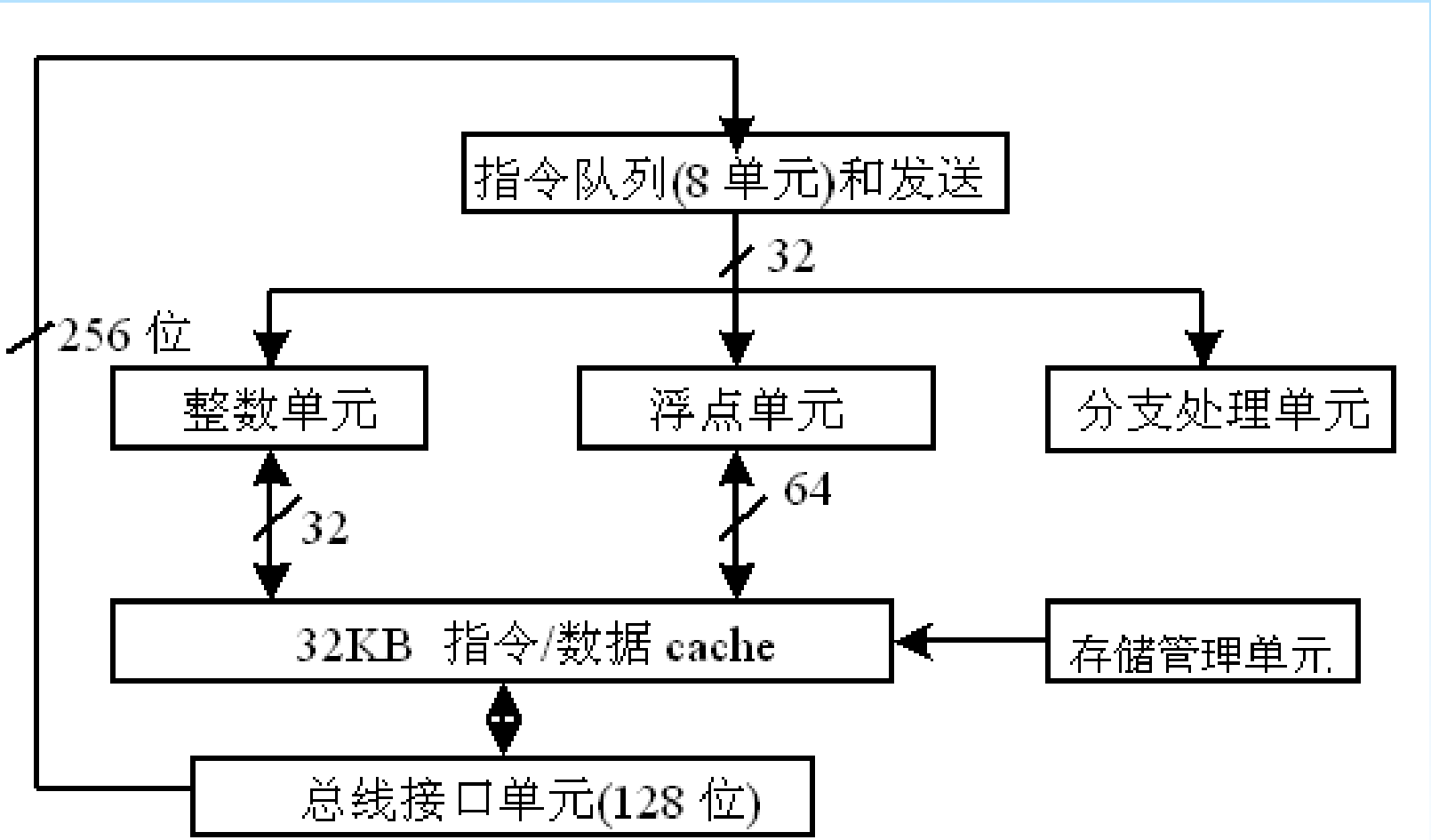


图7.39 PowerPC 601 的结构图

作业：P220—10、11、13、19、21