计算机组成原理

Computer Organization

2022 . 秋

西安交通大学 计算机科学与技术学院

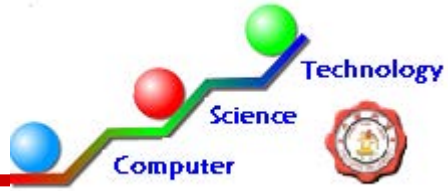
计算机组成原理课程组

http://corg.xjtu.edu.cn

计算机组成原理

第七章 控制器

目录



第七章 控制器

控制器的功能

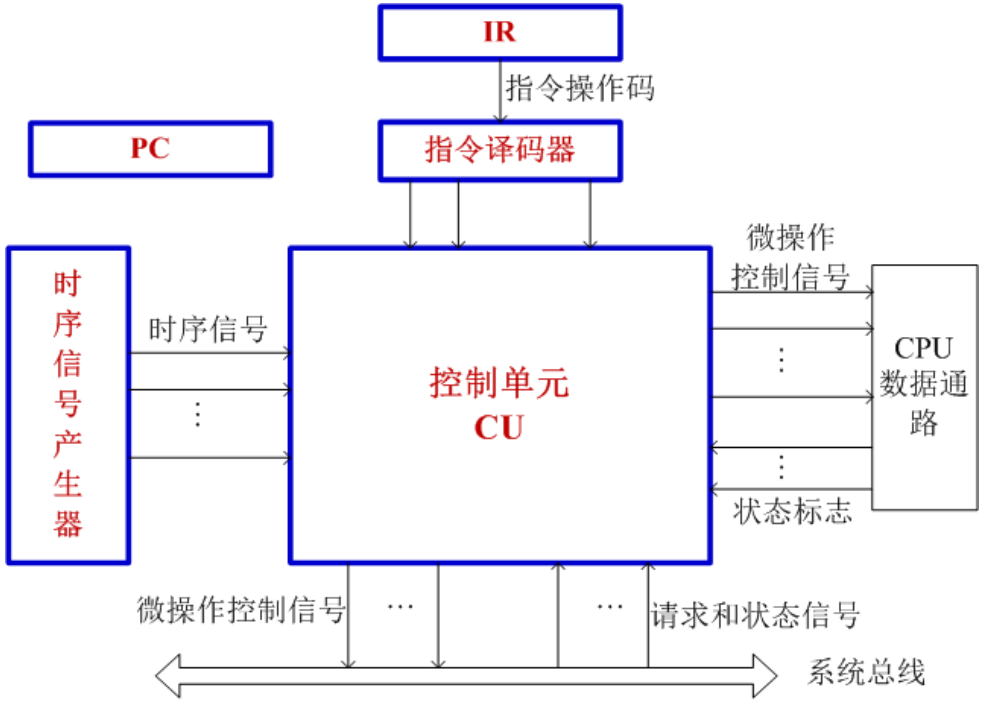
控制器的作用就是向计算机中的各个部件提供它们协调运行所需要的控制信号。

- **提供定时信号**。基本定时信号是主时钟信号，除此之外，根据需要可能对主时钟信号进行分频处理，产生其它不同作用的定时信号。
- **提供控制信号**。指令周期被划分为多个执行阶段，每个阶段通过执行一个微操作序列完成不同功能。控制器应在适当的时刻发出执行部件完成相应操作所需要的控制信号。
- **响应中断请求**。暂停现行程序的执行，进入中断周期（执行中断隐指令）。中断处理结束后，恢复执行被中断的程序。

控制器的组成

从功能角度以及部件间的关联度上，可以认为控制器由三个基本部分构成：

- **控制寄存器和译码器。**包括PC、IR和指令译码器ID。
- **时序信号产生器**
- **控制单元CU。**是控制器的核心，它发出整机运行所需要的全部控制信号。



控制器的设计方法

控制器设计的核心是控制单元的设计。控制单元的设计方法主要有三种，它们的区别在于如何产生微操作控制信号。

- 组合逻辑设计方法，或者称为硬布线设计方法
  - 控制单元被称为组合逻辑控制单元，或者硬布线控制单元。
  - 控制单元是一个组合逻辑网络，产生全机运行所需要的全部控制信号。
  - 特点
    - 设计思想简单，早期计算机设计中普遍使用；
    - 但设计过程非常繁琐，且一旦设计实现后就难以修改；
    - 随着CISC计算机指令系统的不断庞大，被微程序控制单元所替代。

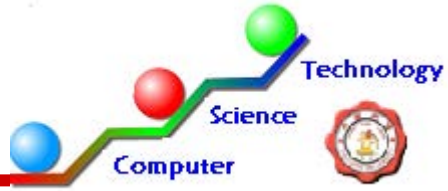
控制器的设计方法（续）

- 微程序设计方法，或者称为存储逻辑设计方法
  - 控制单元被称为微程序控制单元
  - 控制单元组成核心是控制存储器（CM）
  - 由微程序解释执行机器指令，产生指令运行所需要的全部控制信号
  - 特点
    - 一种设计复杂控制单元的技术，兴起于70年代；
    - 硬件结构简单；
    - 控制信号形成的速度慢；
    - 随着VLSI技术的发展，组合逻辑设计方法又重新受到重视，特别是在RISC计算机中普遍采用。

控制器的设计方法（续）

- 组合逻辑与微程序混合设计方法
  - 简单指令用组合逻辑实现，复杂指令用微程序实现。
  - 应用于传统的CISC计算机中
  - 特点
    - 通过加速常用操作提高程序的执行速度；
    - 复杂指令用微程序实现可以在很大程度上降低控制单元设计的复杂性。

目录



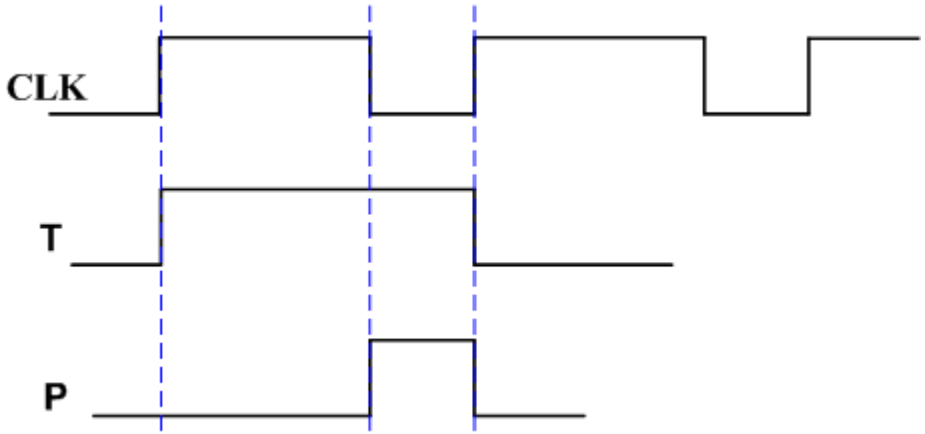
第七章 控制器

计算机中的时序系统

时序系统由一组作为时间基准的信号组成，称为时序信号 。

❒ 时序信号的基本体制

❍计算机中的主要器件是寄存器类的器件（状态单元），这类器件的特性决定了时序信号最基本的体制是**电位—脉冲**工作方式。



计算机中的时序系统（续）

❒ 三级时序体制

❍主状态周期

指令执行通常可以由几个相互较为独立的步骤来完成，把每一步所需要的时间称为一个主状态周期，或机器周期，或CPU工作周期（简称CPU周期）。

❍节拍电位

一个机器周期内可以执行多个相互独立的传送操作，可以将机器周期划分成若干个节拍电位。一个节拍宽度对应一个时钟周期，用来控制产生电位型操作控制信号。

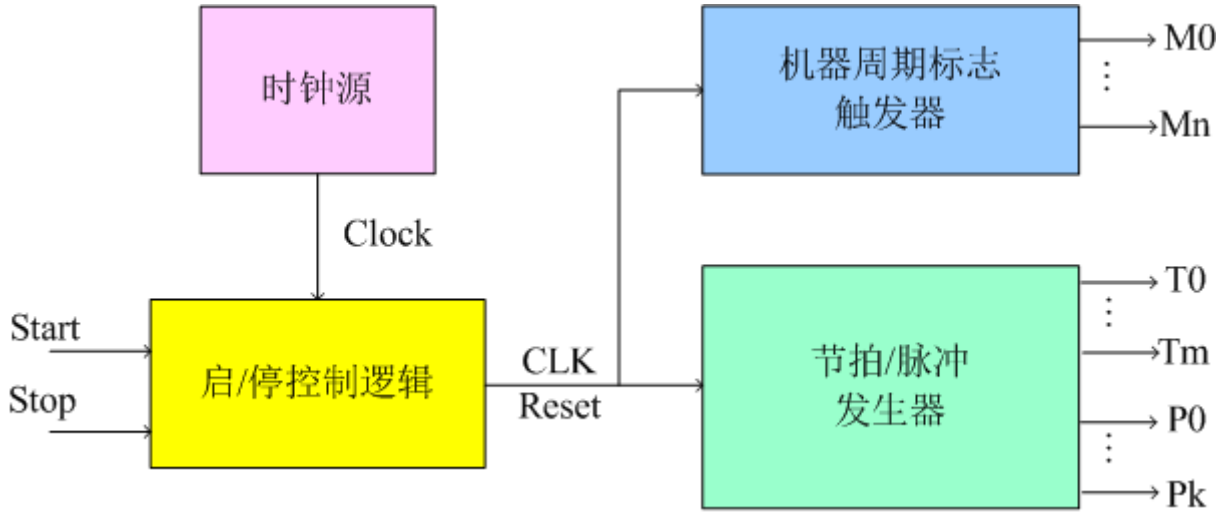
❍节拍脉冲

在一个节拍电位中可以包含多个节拍脉冲，用于产生脉冲型操作控制信号，作为寄存器或触发器同步打入、置位、复位的控制信号。

时序产生器

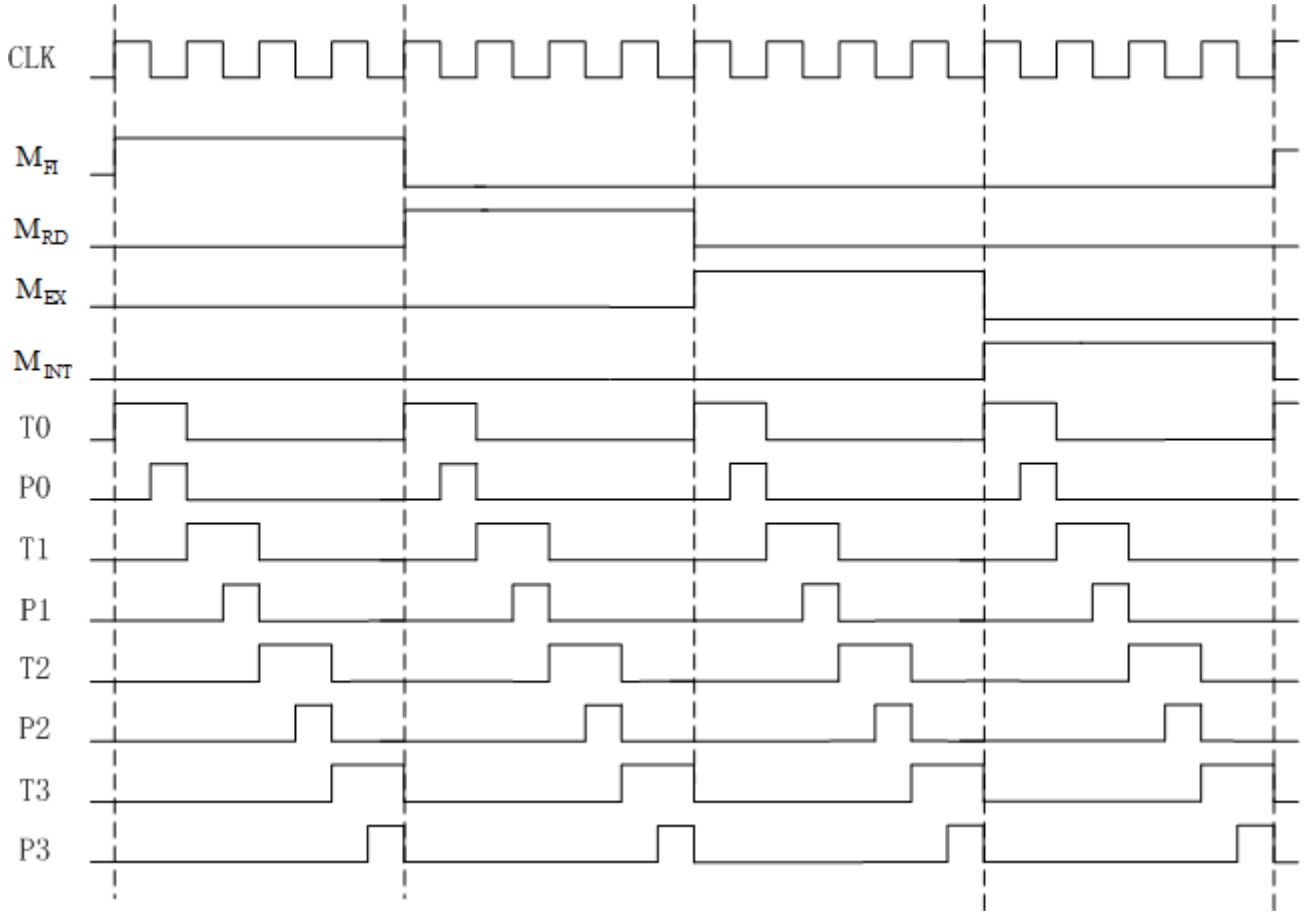
时序产生器为全机提供所需要的全部时序信号。

❒ 时序产生器组成（三级时序系统）



时序产生器（续）

❒ 时序信号波形举例



控制方式

控制不同操作序列时序的方法，称为控制器的控制方式。

- 基本控制方式
  - 同步控制
  - 异步控制
- 用有限状态机描述计算机
  - 指令执行过程中的每一步就是数据通路中线路状态的一次转换，状态转换的结果存放于寄存器（或存储器）。
  - 线路状态的不同组合值对应于有限个状态。CU发出的一组微命令也可以对应于有限个状态，触发状态改变的方式主要有两种：
    - 由时序信号来触发状态转换的方式称为同步控制方式；
    - 由握手机制来触发状态转换的方式称为异步控制方式。

控制方式（续）

❒ 同步控制方式

采用统一的时间基准信号为所有操作进行定时。具体实现方案：

❍完全统一的机器周期和节拍

所有机器周期具有相同的节拍电位数和节拍脉冲数。

❍不定长机器周期

将大多数操作安排在一个较短的机器周期内完成，对某些时间紧张的操作，延长其机器周期。

❍中央控制与局部控制结合

将大部分指令安排在固定的机器周期完成，称为中央控制，对少数复杂指令采用不同的时序，称为局部控制。

控制方式（续）

- 同步控制方式的特点
  - 通常，以最慢部件操作时间作为依据进行时序设计；
  - 每个操作在确定的时间启动和结束，不同操作之间自动衔接；
  - CPU设计简单，但效率低。
  - 同步控制的计算机常采用三级时序系统（机器周期、节拍、脉冲）。

控制方式（续）

❒ 异步控制方式

不需要统一的时间基准信号来定时。

- ❍每条指令、每个操作需要多少时间就占用多少时间。
- ❍当控制器发出某一操作控制信号后，等待执行部件完成该操作后送回“完成”信号，再开始新的操作。
- ❍可以充分发挥各部件的运行速度，时间利用率高，但控制复杂，技术上不易实现。

- ❍异步控制的计算机常采用二级时序系统（节拍、脉冲）。

控制方式（续）

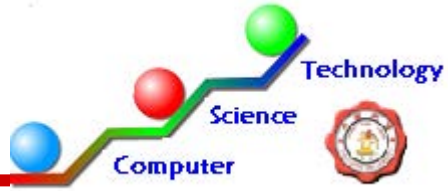
❒ 联合控制方式

同步控制和异步控制相结合的方式。

- ❍大部分操作安排在固定的机器周期中，某些时间难以确定的操作以“应答”方式工作。
- ❍根据其倾向性，可分为同步计算机和异步计算机。
- ❍以同步控制为主，以异步控制为辅，称为同步计算机。
- ❍以异步控制为主，以同步控制为辅，称为异步计算机。

- ❍ 例如在一台计算机中，CPU内部，常采用同步控制方式；
  主机和I/O间，常采用异步控制方式；
  CPU和内存间，采用同步（同步机）
  或异步（异步机）控制方式。

目录



第七章 控制器

硬布线控制器

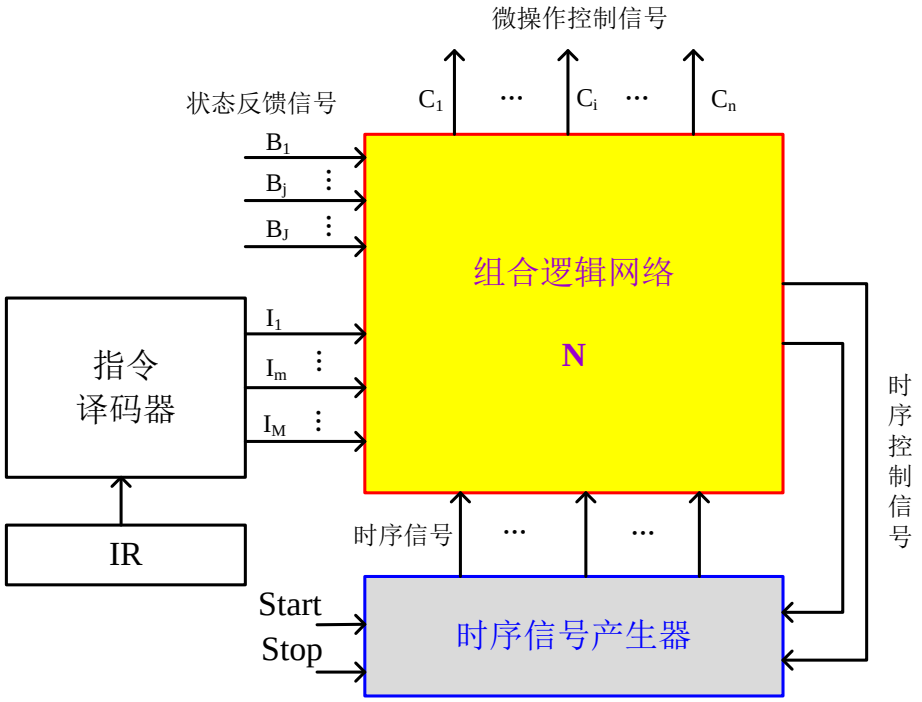
控制器由产生专门固定时序信号的逻辑电路和产生控制命令的组合逻辑网络实现，因此，控制器可看成是由门电路和触发器构成的复杂树形网络。

- 设计方法
  - 对系统进行总体设计

    确定指令系统以及

    数据通路和控制方式。
  - 设计控制器结构和

    时序系统
  - 设计控制单元



硬布线控制器（续）

❒ 控制单元设计步骤

- ❍在数据通路上排出每条指令的指令周期流程，并把流程中的每一步操作分解成微操作序列；
- ❍为每一个微操作分配时间，列出微操作对应的微命令时间表；
- ❍写出微命令的最简逻辑表达式，

$$Ci = f（Im, Bj, Ml, Tk , Pn）$$

即微命令信号是指令译码信号（*Im*）、机器状态（*Bj*）、机器周期（*Ml*）、节拍（*Tk*）和脉冲（*Pn*）的逻辑函数；

- ❍画出产生全部微命令的逻辑图；
- ❍用电路实现控制单元。

单周期控制单元设计实例

❒ 主控制单元功能描述

特点：单周期时序简化到主时钟控制

❍主控制单元功能表

输入/输出	信号名	R型	lw	sw	beq	j
输入	Op5	0	1	1	0	0
	Op4	0	0	0	0	0
	Op3	0	0	1	0	0
	Op2	0	0	0	1	0
	Op1	0	1	1	0	1
	Op0	0	1	1	0	0
输出	RegDst	1	0	X	X	X
	ALUSrc	0	1	1	0	X
	MemtoReg	0	1	X	X	X
	RegWr	1	1	0	0	0
	MemWr	0	0	1	0	0
	MemRd	0	1	0	0	0
	Branch	0	0	0	1	0
	Jump	0	0	0	0	1
	ALUOp1	1	0	0	0	X
	ALUOp0	0	0	0	1	X

单周期控制单元设计实例（续）

❍主控制单元控制信号逻辑表达式

$$\text{RegDst} = \overline{Op_0} \cdot \overline{Op_1} \cdot \overline{Op_2} \cdot \overline{Op_3} \cdot \overline{Op_4} \cdot \overline{Op_5}$$

$$\text{ALUScr} = Op_0 \cdot Op_1 \cdot \overline{Op_2} \cdot \overline{Op_3} \cdot \overline{Op_4} \cdot Op_5 + Op_0 \cdot Op_1 \cdot \overline{Op_2} \cdot Op_3 \cdot \overline{Op_4} \cdot Op_5$$

$$\text{MemtoReg} = Op_0 \cdot Op_1 \cdot \overline{Op_2} \cdot \overline{Op_3} \cdot \overline{Op_4} \cdot Op_5$$

$$\text{RegWr} = \overline{Op_0} \cdot \overline{Op_1} \cdot \overline{Op_2} \cdot \overline{Op_3} \cdot \overline{Op_4} \cdot \overline{Op_5} + Op_0 \cdot Op_1 \cdot \overline{Op_2} \cdot \overline{Op_3} \cdot \overline{Op_4} \cdot Op_5$$

$$\text{MemWr} = Op_0 \cdot Op_1 \cdot \overline{Op_2} \cdot Op_3 \cdot \overline{Op_4} \cdot Op_5$$

$$\text{MemRd} = Op_0 \cdot Op_1 \cdot \overline{Op_2} \cdot \overline{Op_3} \cdot \overline{Op_4} \cdot Op_5$$

$$\text{Branch} = \overline{Op_0} \cdot \overline{Op_1} \cdot Op_2 \cdot \overline{Op_3} \cdot \overline{Op_4} \cdot \overline{Op_5}$$

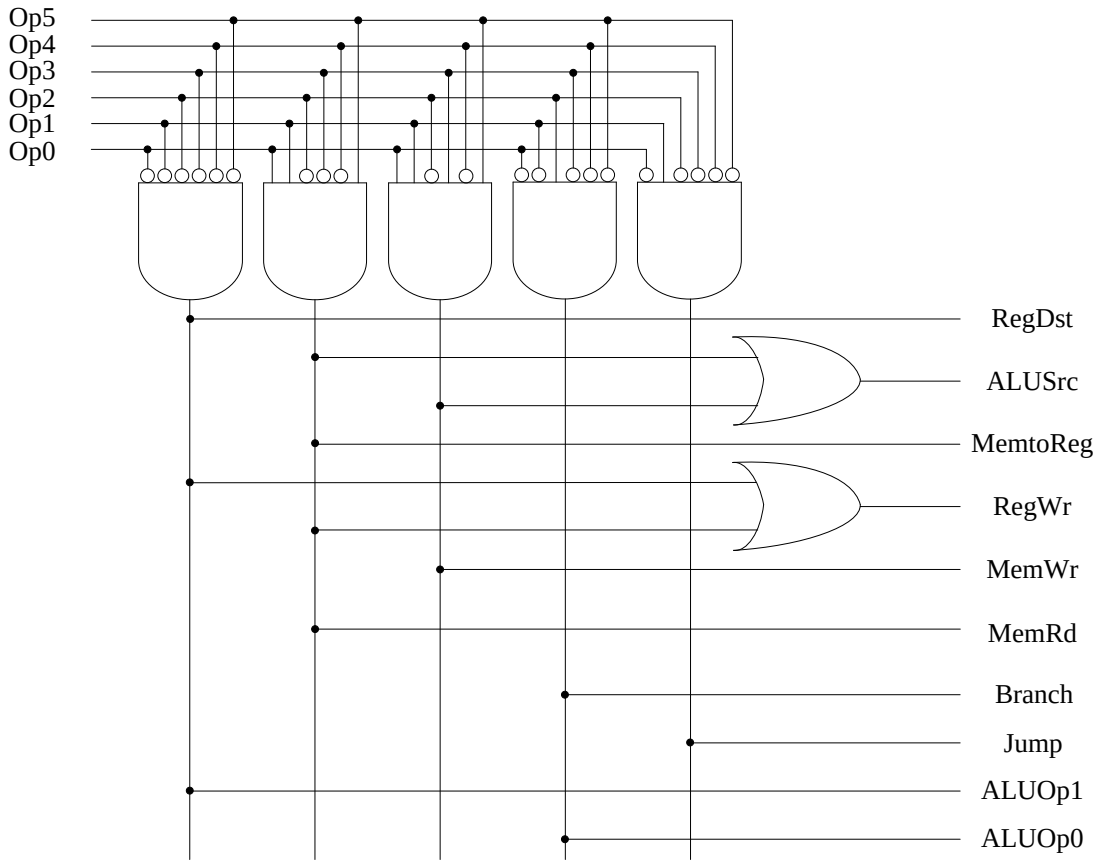
$$\text{Jump} = \overline{Op_0} \cdot Op_1 \cdot \overline{Op_2} \cdot \overline{Op_3} \cdot \overline{Op_4} \cdot \overline{Op_5}$$

$$\text{ALUOp}_1 = \overline{Op_0} \cdot \overline{Op_1} \cdot \overline{Op_2} \cdot \overline{Op_3} \cdot \overline{Op_4} \cdot \overline{Op_5}$$

$$\text{ALUOp}_0 = \overline{Op_0} \cdot \overline{Op_1} \cdot Op_2 \cdot \overline{Op_3} \cdot \overline{Op_4} \cdot \overline{Op_5}$$

单周期控制单元设计实例（续）

❍主控制单元逻辑电路



门阵列控制器

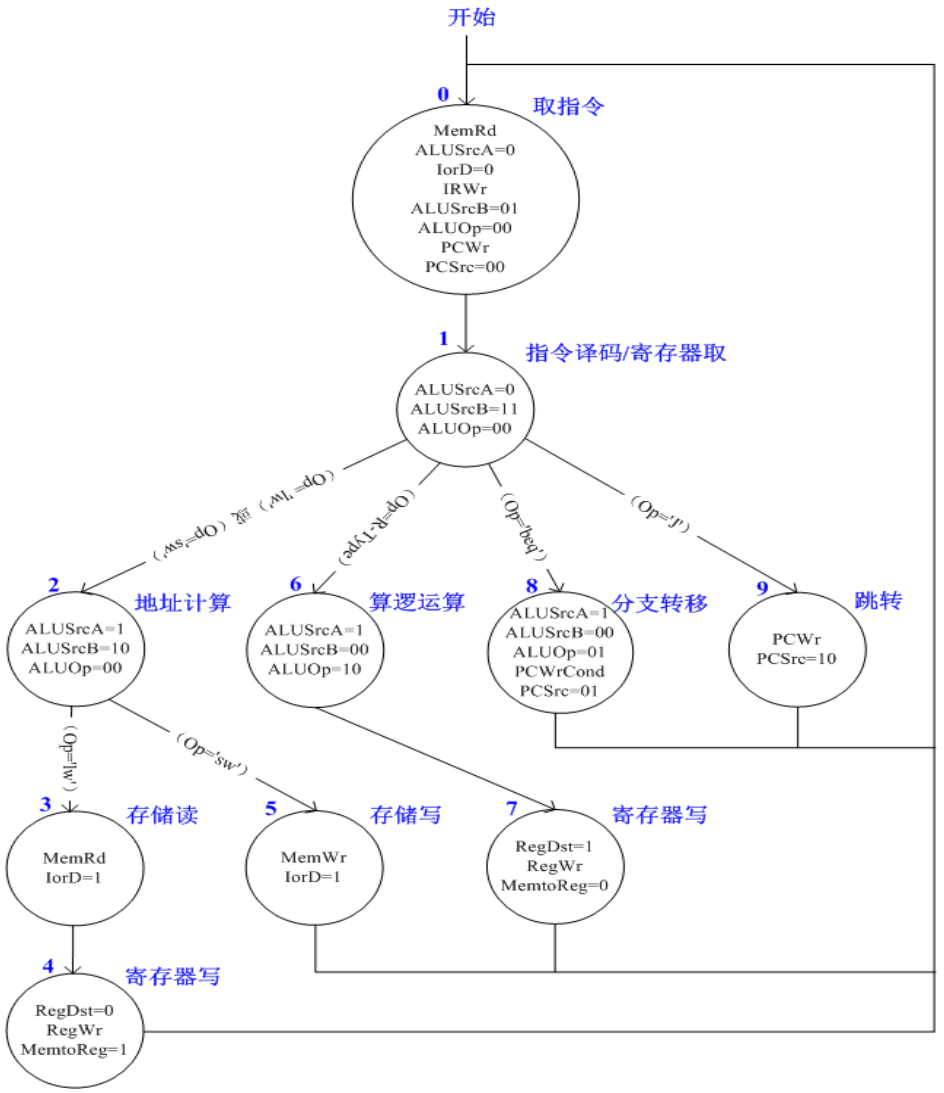
通用可编程逻辑器件由大量的与门、或门阵列等电路构成，简称门阵列器件。由门阵列器件设计的控制器称为**门阵列控制器**。

- 设计方法
  - 类似硬布线控制器
  - 用门阵列芯片实现控制逻辑

多周期同步控制单元设计实例

- 同步控制状态转换图

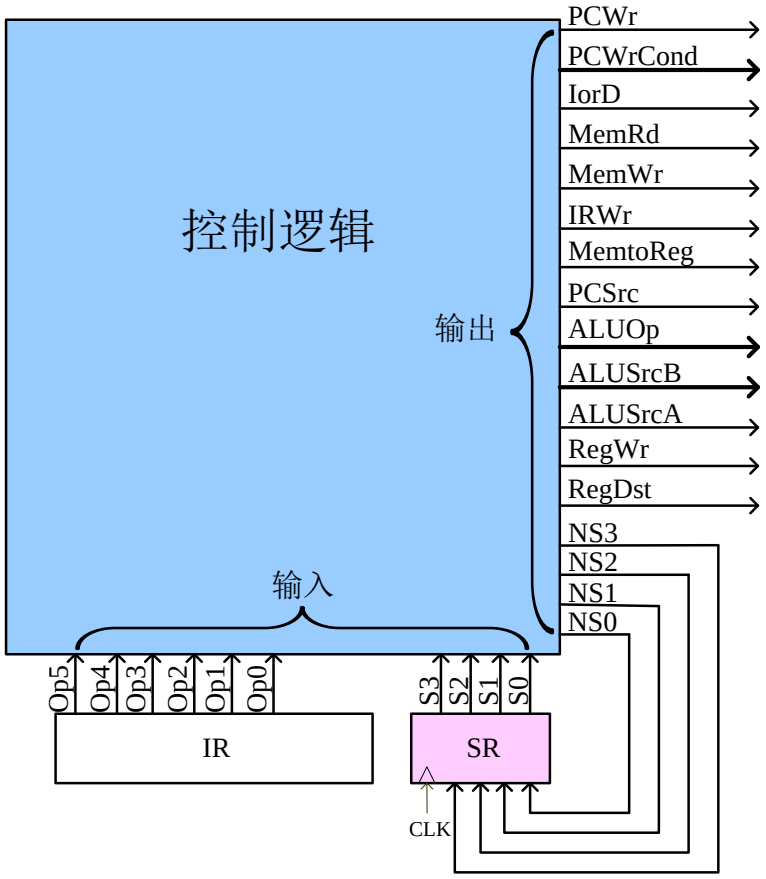
可以用一个有限状态机（FSM）描述多周期CPU控制单元。控制信号的不同组合值对应于有限个状态，分别用数字编号表示；由主时钟信号CLK触发状态改变。



多周期同步控制单元设计实例（续）

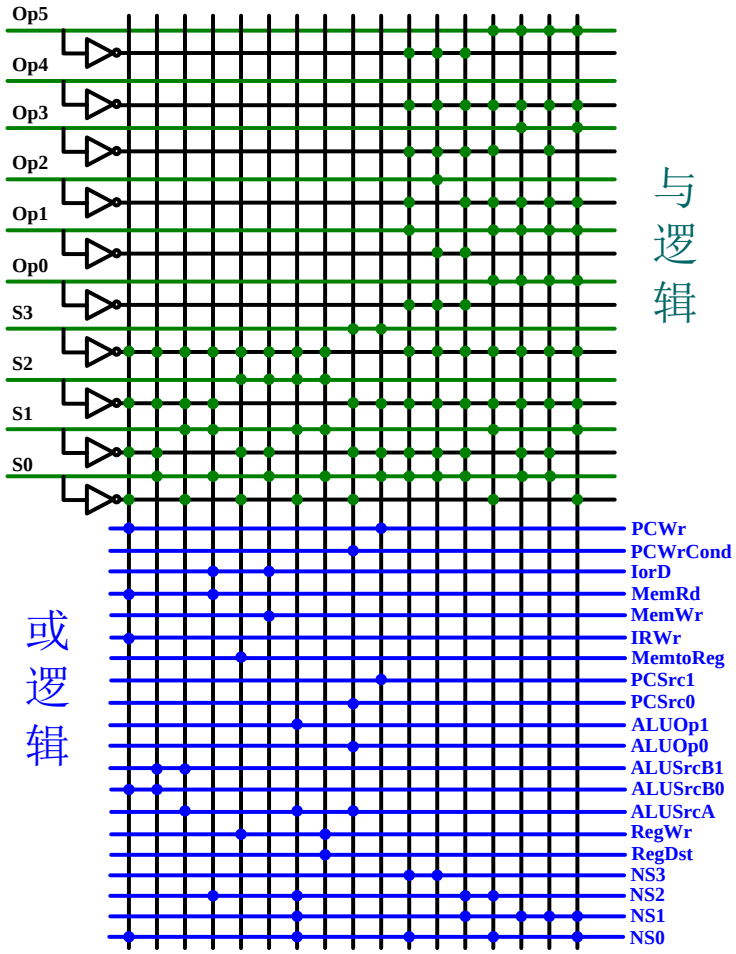
❒ 同步控制单元框图

控制单元由一个组合逻辑模块和一个保存当前状态的寄存器（SR）组成。



多周期同步控制单元设计实例（续）

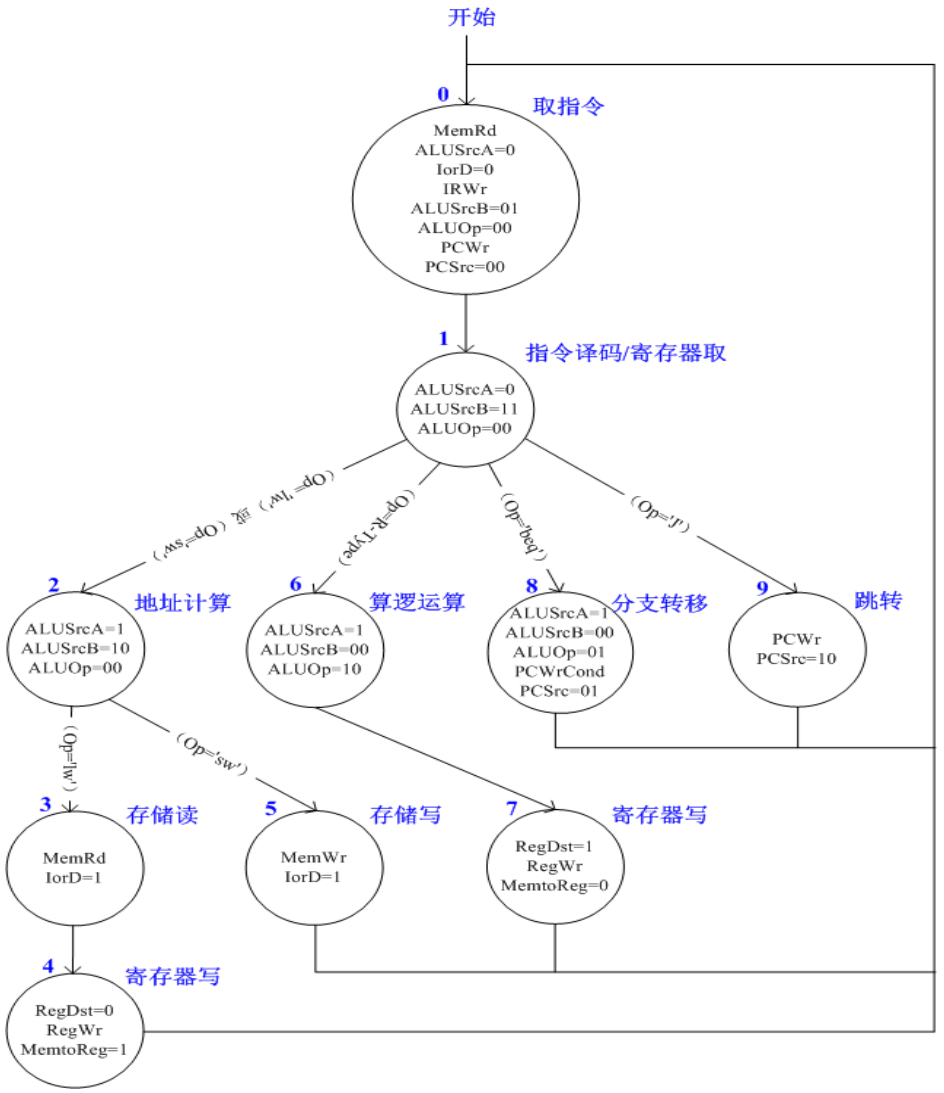
- 用PLA实现的控制单元



多周期同步控制单元设计实例

- 同步控制状态转换图

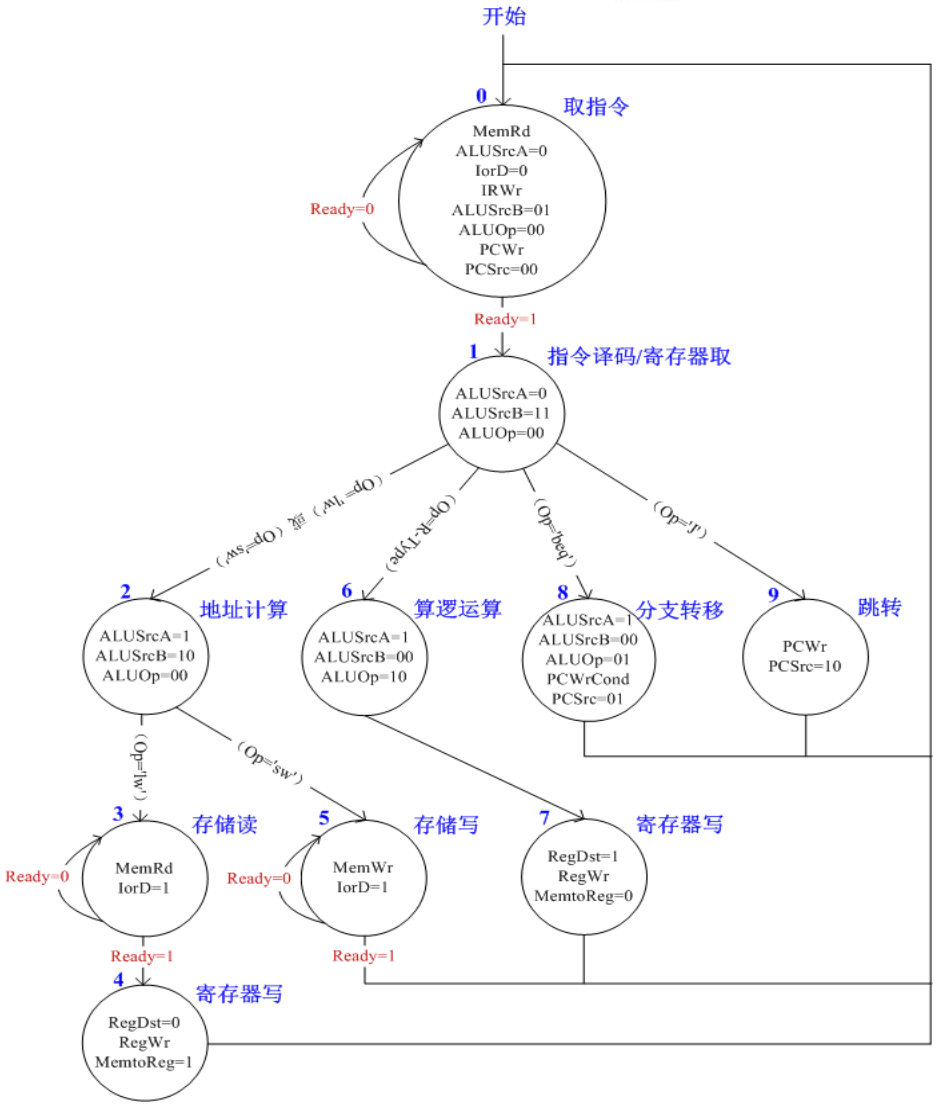
可以用一个有限状态机（FSM）描述多周期CPU控制单元。控制信号的不同组合值对应于有限个状态，分别用数字编号表示；由主时钟信号CLK触发状态改变。



多周期异步控制单元设计实例

❒ 异步控制状态转换图

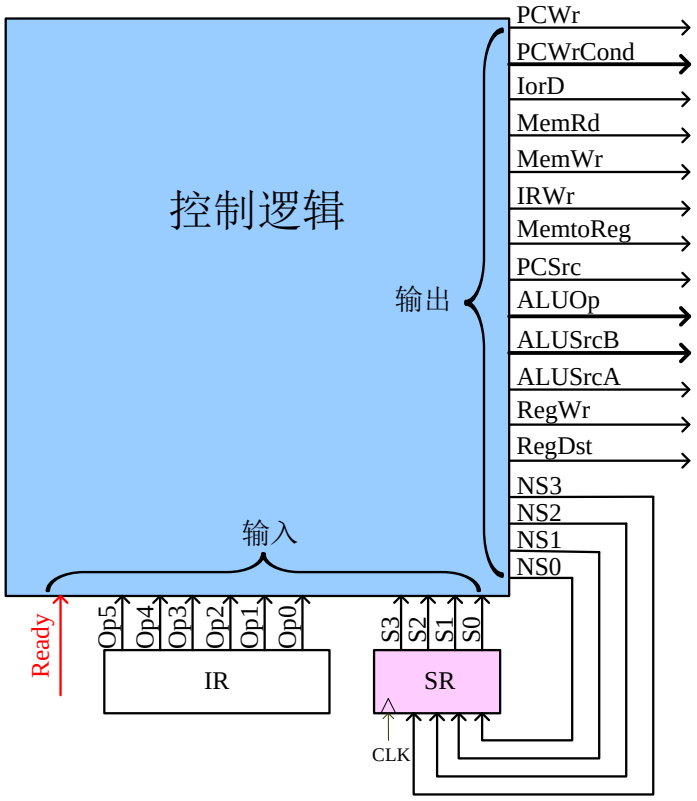
假设存储器和CPU之间按照异步方式工作。CPU向存储器发送MemRd或MemWr控制命令，存储器向CPU发送Ready信号作为应答。



多周期异步控制单元设计实例（续）

❒ 异步控制单元框图

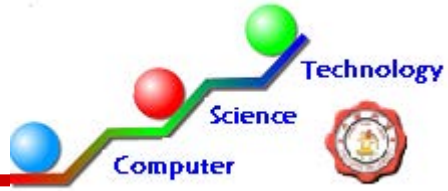
控制单元由一个组合逻辑模块和一个保存当前状态的寄存器（SR）组成。



组合逻辑控制单元的特点

- 设计方法简单，早期计算机中普遍使用；
- 设计过程繁琐，且修改困难。后来逐步被微程序控制单元所替代设计；
- 控制信号形成速度快，提高了整机处理速度。随着VLSI技术的发展，其技术又重新受到重视，在RISC技术中普遍采用。

目录



第七章 控制器

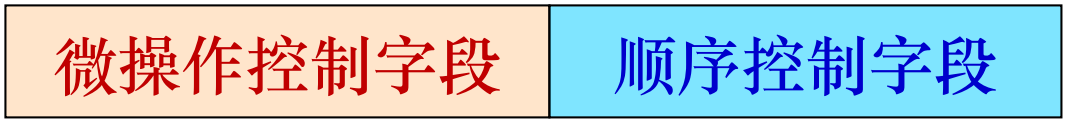
微程序控制器设计思想

❒ 微程序设计思想

- ❍将微命令二进制代码化，即用一位二进制信息1、0来表示一个微命令的有、无；
- ❍将一段时间内所有微命令位组织在一起，形成一个控制字，称为“微指令”；
- ❍将一条机器指令执行过程中需要的全部微命令，用若干条微指令编程实现，称为“微程序”；
- ❍将机器全部微程序存储在一个特定的存储器中，称为“控制存储器CM”，通过执行微程序实现全部控制功能。

微指令格式

❒ 微指令基本格式



❍微操作控制字段（ μOP ）

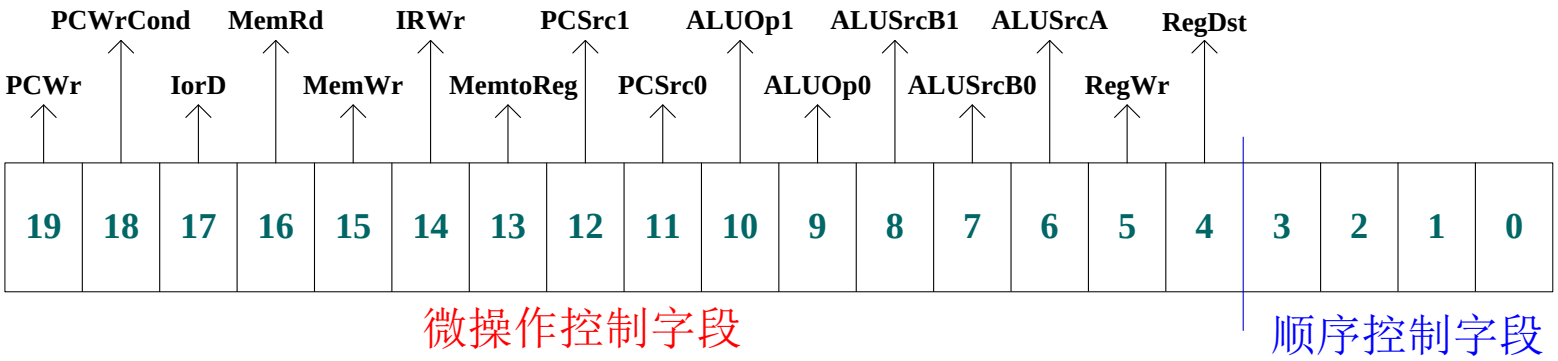
指出该条微指令的操作性质，即由该字段能产生全部相应微操作控制信号（微命令）。

❍顺序控制字段（ μAddr ）

指出下一条微指令在控存的地址，即对微程序级的执行顺序进行控制。

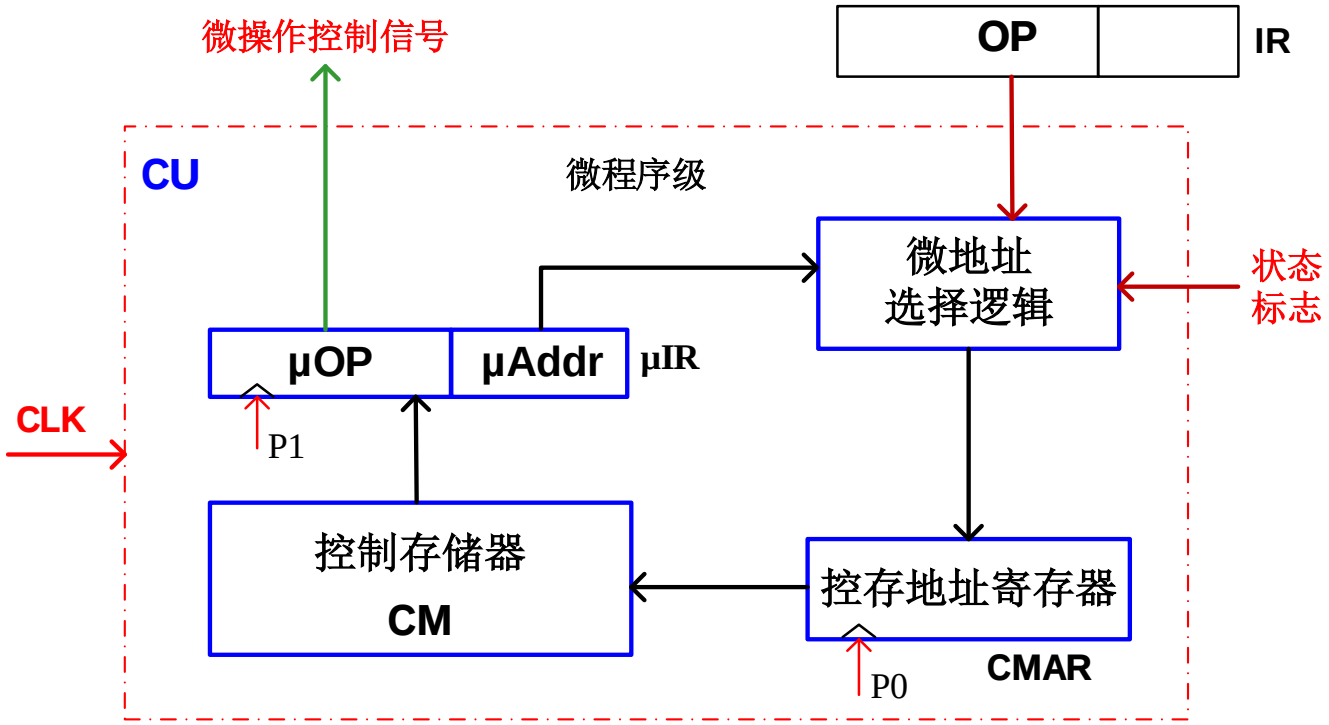
微指令格式（续）

❒ 微指令格式举例



微程序控制单元的基本结构

❒ 微程序控制单元基本组成



微程序控制单元的基本结构（续）

❒ 微程序控制单元基本组成

- ❍控制存储器CM，也称为微程序存储器μPM：用来存放实现全部指令系统对应的微程序。
- ❍微指令寄存器μIR：用来存放由控制存储器读出的一条微指令。
- ❍微地址选择逻辑：形成下一条微指令的地址。
- ❍微地址寄存器μAR，也称为控存地址寄存器CMAR：暂存将要访问的下一条微指令在控存中的地址。

微指令格式设计

❒ 微程序控制单元设计内容

- ❍设计微指令格式及相应的逻辑结构
- ❍设计控制存储器的结构
- ❍编制微程序

其中：

- ❍控存结构设计采用存储技术完成
- ❍微程序编制采用程序设计技术完成

因此，微程序控制单元的设计主要解决微指令结构的设计问题，即微程序设计技术。

微指令格式设计（续）

- 微指令结构设计的基本原则
  - 有利于缩短微指令长度；
  - 有利于减小控存容量；
  - 有利于提高微程序的执行速度；
  - 有利于对微指令进行修改；
  - 有利于微程序设计的灵活性。

微指令格式设计（续）

❒ 微指令结构分类

❍水平型微指令

在一条微指令中定义执行多个微操作的微命令。

❍垂直型微指令

在一条微指令只提供1~2种微命令。微指令格式与指令相似。

❒ 两种微指令结构比较

❍水平型微指令

并行操作能力强、微指令长但微程序短、难使用。

❍垂直型微指令

并行操作能力弱、微指令短但微程序长、易使用。

微指令格式设计（续）

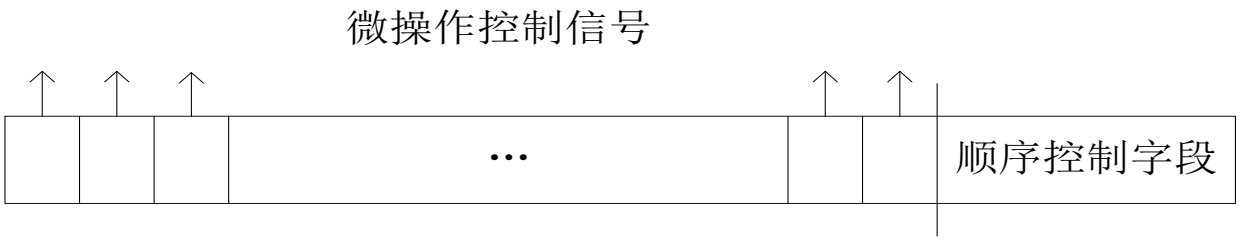
❒ 微指令格式设计——以水平型微指令格式为例

❍微命令编码方式

主要解决微指令的操作控制字段的格式安排。

✧直接编码方式（直接控制、不译法）

微操作控制字段的每一位表示一个微命令，“0”表示无效（一种状态），“1”表示有效（另一种状态）。



特点：简单直观，速度快，但微指令字长较长，适用于结构较简单的机器。

微指令格式设计（续）

- 微命令之间的关系
  - 相容性微命令

    能在同时或同一个时间段内有效的微命令。

    例如：存储器读（MemRd）、指令寄存器IR写（IRWr）、ALU加法（add）和PC写（PCWr）等微命令。
  - 互斥性微命令

    不能在同时或在同一个时间段内有效的微命令。

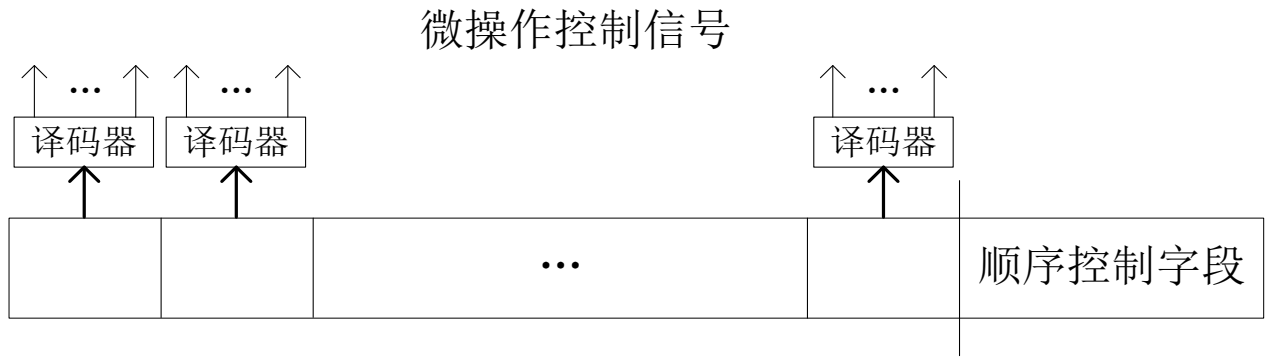
    例如：存储器读（MemRd）和存储器写（MemWr）等微命令。

微指令格式设计（续）

✧字段直接编码方式（分段直接编译法）

将微操作控制字段分为若干小字段，把一组互斥微命令组织在一起，用一个小字段编码表示，将相容的微命令安排在不同字段内。在某一时间段，每个字段通过译码产生一条微命令，不同字段可以发出多条微命令。

特点：能有效压缩微指令的长度，但由于译码稍影响速度。



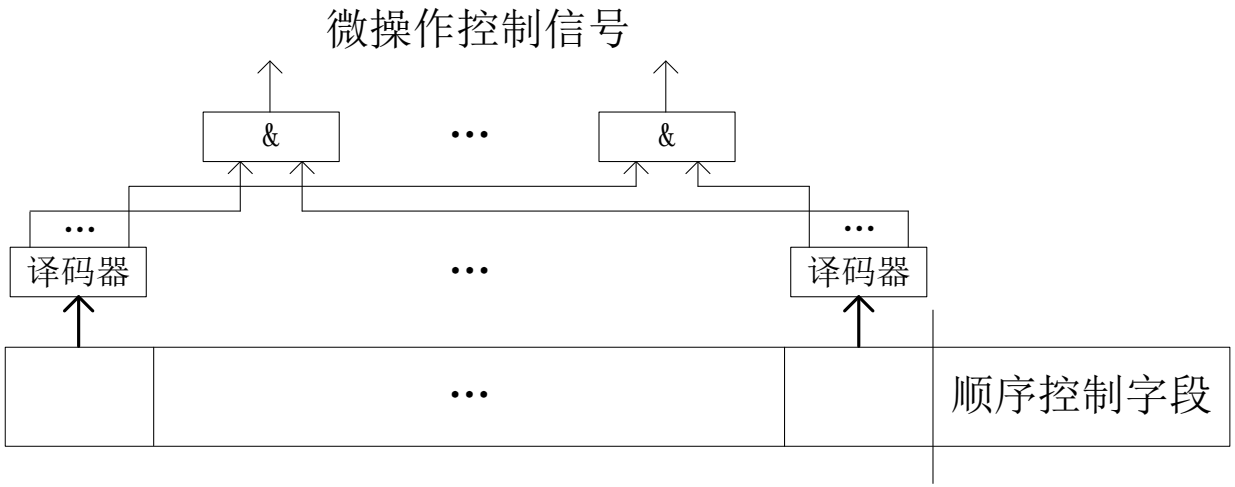
注意：为每个字段分配编码时，应考虑无操作的情况，即 n 位通常仅能安排 2^n-1 个微命令。

微指令格式设计（续）

✧字段间接编码方式

在字段直接编码方式的基础上，若规定一个字段的某些微命令，要兼由另一个字段中的某些微命令来解释，称为字段间接编码方式。

特点：在字段直接编码法的基础上，进一步缩短了微指令的长度，但可能消减微指令的并行控制能力。通常只作为直接编码法的一种辅助手段。



微指令格式设计（续）

✧混合编码方式

当分段编码方式（直接或间接）中某些小字段的位数少到只有一位时，就可认为是直接编码方式和字段直接（间接）编码方式的混合编码方式。

特点：可改善分段编码方式的灵活性和执行速度。

微指令格式设计（续）

❍微命令编码方式举例

某机的微指令格式中，共有8个控制字段，每个字段可分别激活5、8、3、16、1、7、25、4种控制信号。分别采用直接编码和字段直接编码方式设计微指令的操作控制字段，并说明两种方式的操作控制字段各取几位。

解：

(1) 采用直接编码方式时，直接用微指令的操作控制字段的一位表示1个微命令，则操作控制字段的位数等于微命令个数，即

总位数 = 5+8+3+16+1+7+25+4 = 69位

微指令格式设计（续）

(2) 采用字段直接编码方式时，每个小字段中的微命令应为互斥的。此时为每个小字段分配位数时要在有效微命令个数的基础上再加1种无操作状态，则每个小字段需给出的状态数分别为6、9、4、17、2、8、26、5种，据此设计的微指令的操作控制字段格式如下：

3位	4位	2位	5位	1位	3位	5位	3位

共分8个小字段，微指令操作控制字段的总位数为：

总位数=3+4+2+5+1+3+5+3=26位

与直接编码法相比，压缩了：69-26 = 43位

可见，字段直接编码法可有效地缩短微指令字长。

微指令格式设计（续）

- 微地址的形成方法

  在微程序执行的不同阶段获得下一条微指令地址（后继微地址）的方法不同。

  - 直接表示方式

    微指令的顺序控制字段直接给出后继微地址。

特点：简单，但微指令字长较长。

微指令格式设计（续）

✧增量方式（计数器方式）

– 硬件上 CMAR 增加计数功能（μPC）

– 两种基本格式：顺序微指令和转移微指令

（a）顺序型微指令

（b）转移型微指令

– 特点：微指令字长较短，但转移灵活性差

微指令格式设计（续）

✧增量与下址字段相结合方式

– 硬件上 CMAR 增加计数功能（μPC）

– 仅有一种微指令格式，每条微指令都具有转移功能

– 特点：指令格式统一，但微指令字长较长

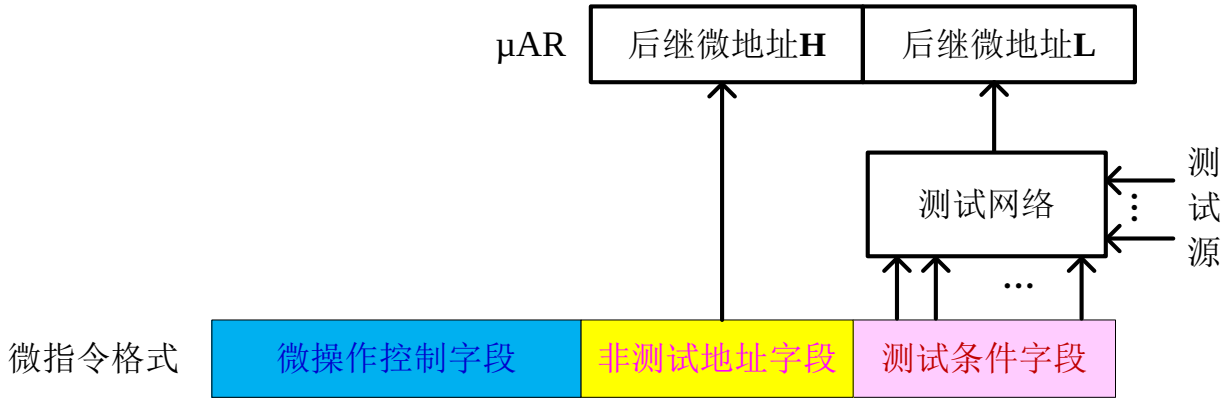
注意：

✧增量与下址字段相结合方式只能实现两路分支转移。

微指令格式设计（续）

✧断定方式

- 后继微地址只能由微指令的顺序控制字段产生
- 每一条微指令都具有转移能力，CMAR 无计数功能
- 微指令中给出后继微地址的**部分位**，其余位由机器运行状态和测试条件来断定。



- **特点：** 可实现微程序的多路分支转移，转移路数由微地址被断定的位数决定。

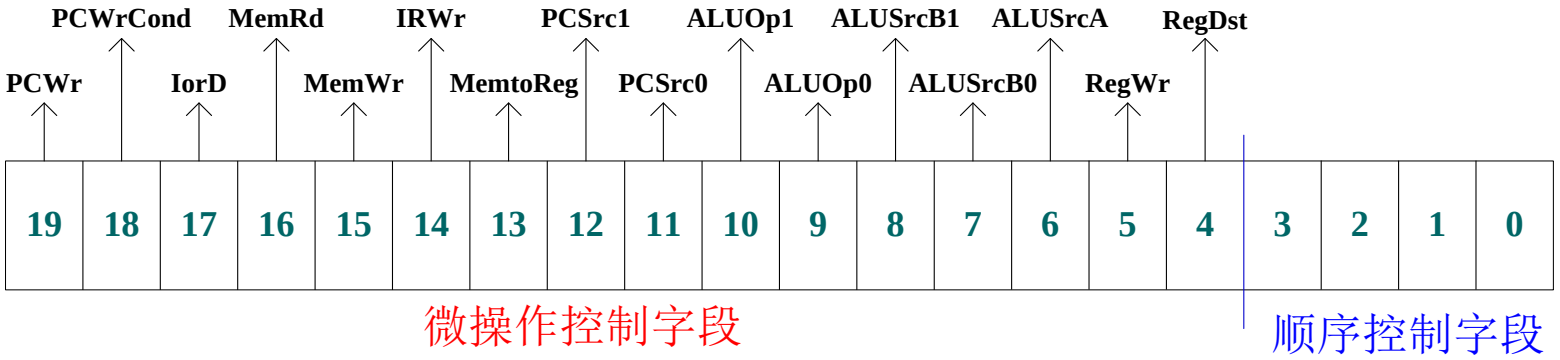
微入口技术

- 前述的微地址形成方法主要解决微程序执行过程中下一条微指令地址（后继微地址）的获得方法，但如何进入相应的微程序段执行则需要另外的技术。
- 根据特定机器指令进入对应的微程序执行的方法通常称为“功能转移”。功能转移按照机器指令操作码转移到相应微程序的入口。
- 功能转移属于微程序多路分支的情况，必须外加相应硬件逻辑实现，具体方法有：硬件查表法（ROM或PLA）和转移地址产生逻辑。

微指令格式和微程序设计实例

以MIPS多周期（分散互连）控制单元设计为例，采用水平型微指令结构，介绍微指令格式和微程序的基本设计方法。

- 设计实例一
  - 微命令编码方式：直接编码方式
  - 微地址形成方法：直接表示方式
  - 微入口：二级功能转移（指令操作码译码，LW/SW）
  - 微指令格式



微指令格式和微程序设计实例

❍微程序

微指令名称	微指令地址（二进制）	微指令（二进制代码）																		
		操作控制															顺序控制			
		PCWr			MemRd	15	IRWr			11	10	9	8	ALUSrcB0	6	5	4	3	2	1	0
Fetch1 Fetch2	0000	1	0	0	1	0	1	0	0	0	0	0	0	1	0	0	0				
	0001	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	X	X	X	X
LW/SW	0010	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	Y	Y	Y	Y
LW1 LW2	0011	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0			
	0100	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0			
SW	0101	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R-type1 R-type2	0110	0	0	1	1	0	0	0	0	0	1	0	0	0	1	0	0	0	1	1	1
	0111	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0
BEQ	1000	0	1	0	0	0	0	0	0	1	0	1	0	0	1	0	0	0	0	0	0
JUMP	1001	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0

功能转移1

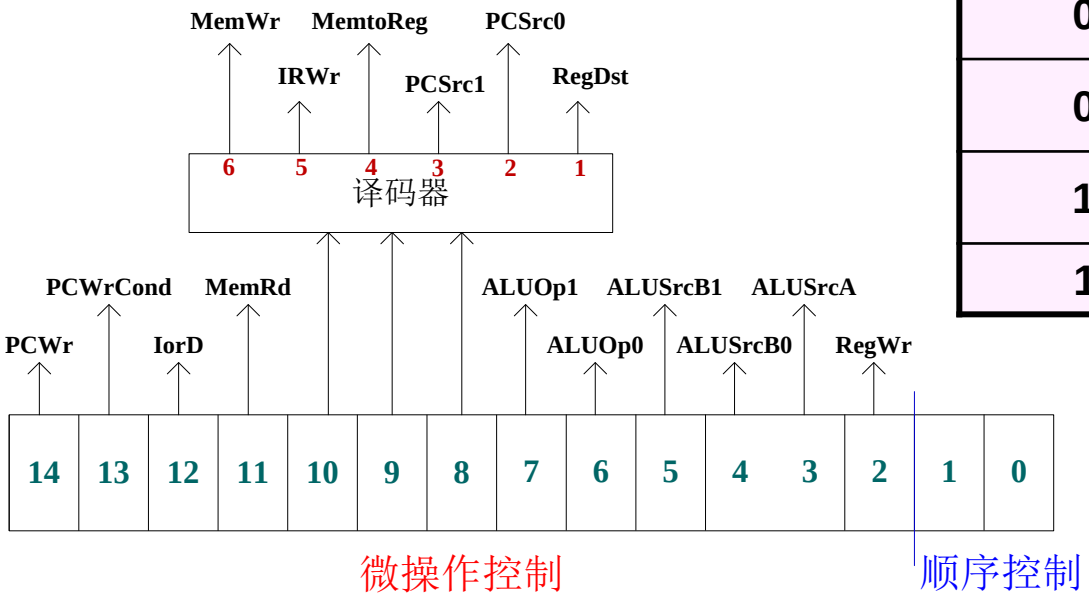
功能转移2

微指令格式和微程序设计实例

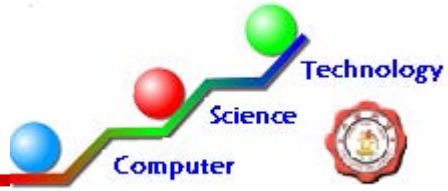
设计实例二

- 微命令编码方式：混合编码方式
- 微地址形成方法：编码选择微地址
- 微指令格式

顺序控制	微地址操作
00	转取指令（0号单元）
01	功能转移1
10	功能转移2
11	顺序执行（+1）



微指令格式和微程序设计实例



❍微程序

回上页 转下页

微指令名称	微指令地址（二进制）	微指令（二进制代码）																		
		操作控制															顺序控制			
		19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Fetch1 Fetch2	0000	1	0	0	1	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	1
	0001	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	X	X	X	X
LW/SW	0010	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	Y	Y	Y	Y
LW1 LW2	0011	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
	0100	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0
SW	0101	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R-type1 R-type2	0110	0	0	1	1	0	0	0	0	0	1	0	0	0	1	0	0	0	1	1	1
	0111	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0
BEQ	1000	0	1	0	0	0	0	0	0	1	0	1	0	0	1	0	0	0	0	0	0
JUMP	1001	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0

微指令格式和微程序设计实例

❍微程序

微程序名称	微指令地址（二进制）	微指令（二进制代码）													
		操作控制												顺序控制	
		14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Fetch1 Fetch2	0000	1	0	0	1	1	0	1	0	0					1	1
	0001	0	0	0	0	0	0	0	0	0	1	1	0	0	0	1
LW/SW	0010	0	0	0	0	0	0	0	0	0					1	0
LW1 LW2	0011	0	0	1	1	0	0	0	0	0	0	0	0	0	1	1
	0100	0	0	0	0	1	0	0	0	0					0	0
SW	0101	0	0	1	0	1	1	0	0	0					0	0
R-type1 R-type2	0110	0	0	1	1	0	0	0	1	0	0	0	1	0	1	1
	0111	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0
BEQ	1000	0	1	0	0	0	1	0	1	1	0	0	1	0	0	0
JUMP	1001	1	0	0	0	0	1	1	0	0	0	0	0	0	0	0

功能转移1

功能转移2

转取指令

+1

微指令格式和微程序设计实例

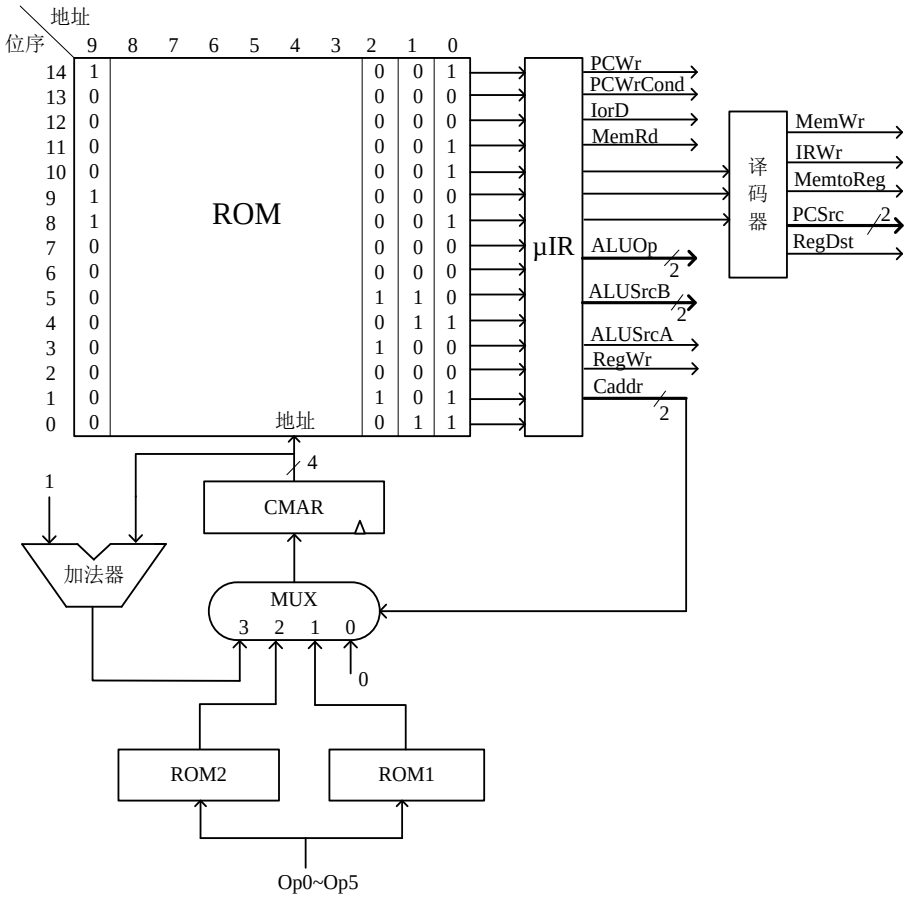
❍微程序控制单元框图

ROM1（功能转移1）

指令操作码（Op5~Op0）	指令名称	后继微地址
000000	R-type	0110
000010	jump	1001
000100	beq	1000
100011	lw	0010
101011	sw	0010

ROM2（功能转移2）

指令操作码（Op5~Op0）	指令名称	后继微地址
100011	lw	0011
101011	sw	0101



微指令格式和微程序设计实例

❒ 设计实例三

- ❍采用异步内存方式：存储器以Ready应答MemRd、MemWr命令
- ❍微命令编码方式：混合编码方式
- ❍微地址形成方法：条件选择+编码选择微地址
- ❍微指令格式

条件选择	操作	Ready	微地址
0	不测试		下址字段
1	测Ready	0	保持不变
		1	下址字段

微指令格式和微程序设计实例

❍微程序

微程序名称	微指令地址（二进制）	微指令（二进制代码）														
		微操作码												条件	下地址	
		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Fetch1 Fetch2	0000	1	0	0	1	1	0	1	0	0	0	1	0	0	1	1	1
	0001	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
LW/SW	0010	0	0	0	0	0	0	0	0	0	1	0	1	0	0	1	0
LW1 LW2	0011	0	0	1	1	0	0	0	0	0	0	0	0	0	1	1	1
	0100	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0
SW	0101	0	0	1	0	1	1	0	0	0	0	0	0	0	1	0	0
R-type1 R-type2	0110	0	0	1	1	0	0	0	1	0	0	0	1	0	0	1	1
	0111	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0
BEQ	1000	0	1	0	0	0	1	0	1	1	0	0	1	0	0	0	0
JUMP	1001	1	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0

测Ready

微指令格式和微程序设计实例

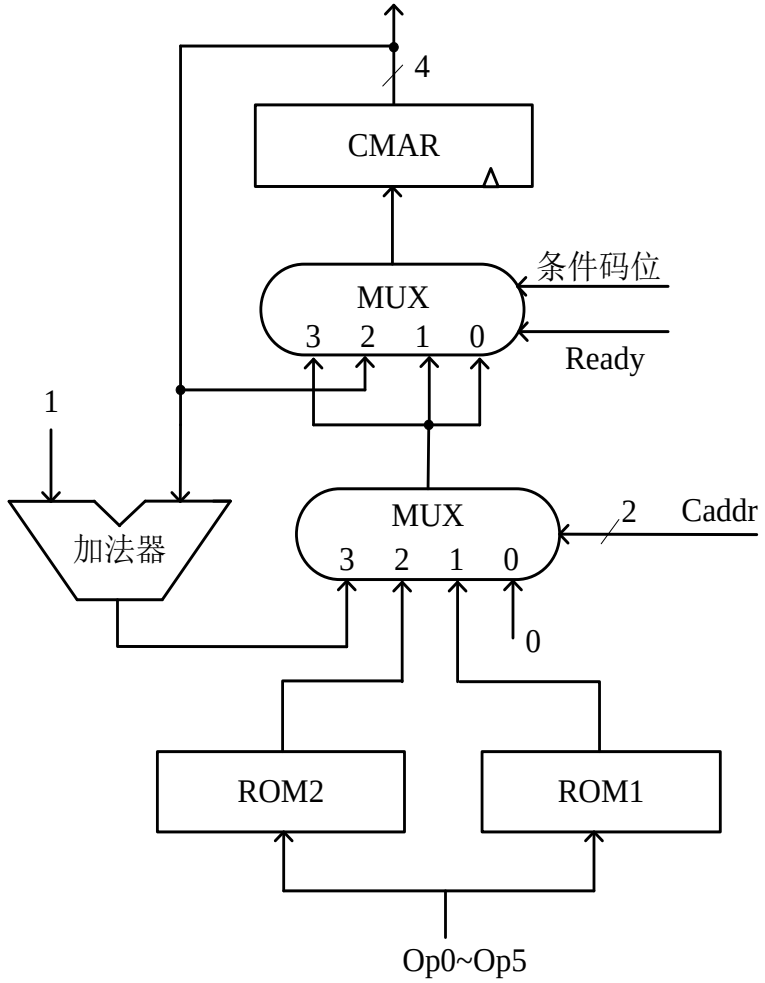
❍微地址选择逻辑

ROM1

指令操作码（Op5~Op0）	指令名称	后继微地址
000000	R-type	0110
000010	jump	1001
000100	beq	1000
100011	lw	0010
101011	sw	0010

ROM2

指令操作码（Op5~Op0）	指令名称	后继微地址
100011	lw	0011
101011	sw	0101



微指令顺序控制字段设计举例1

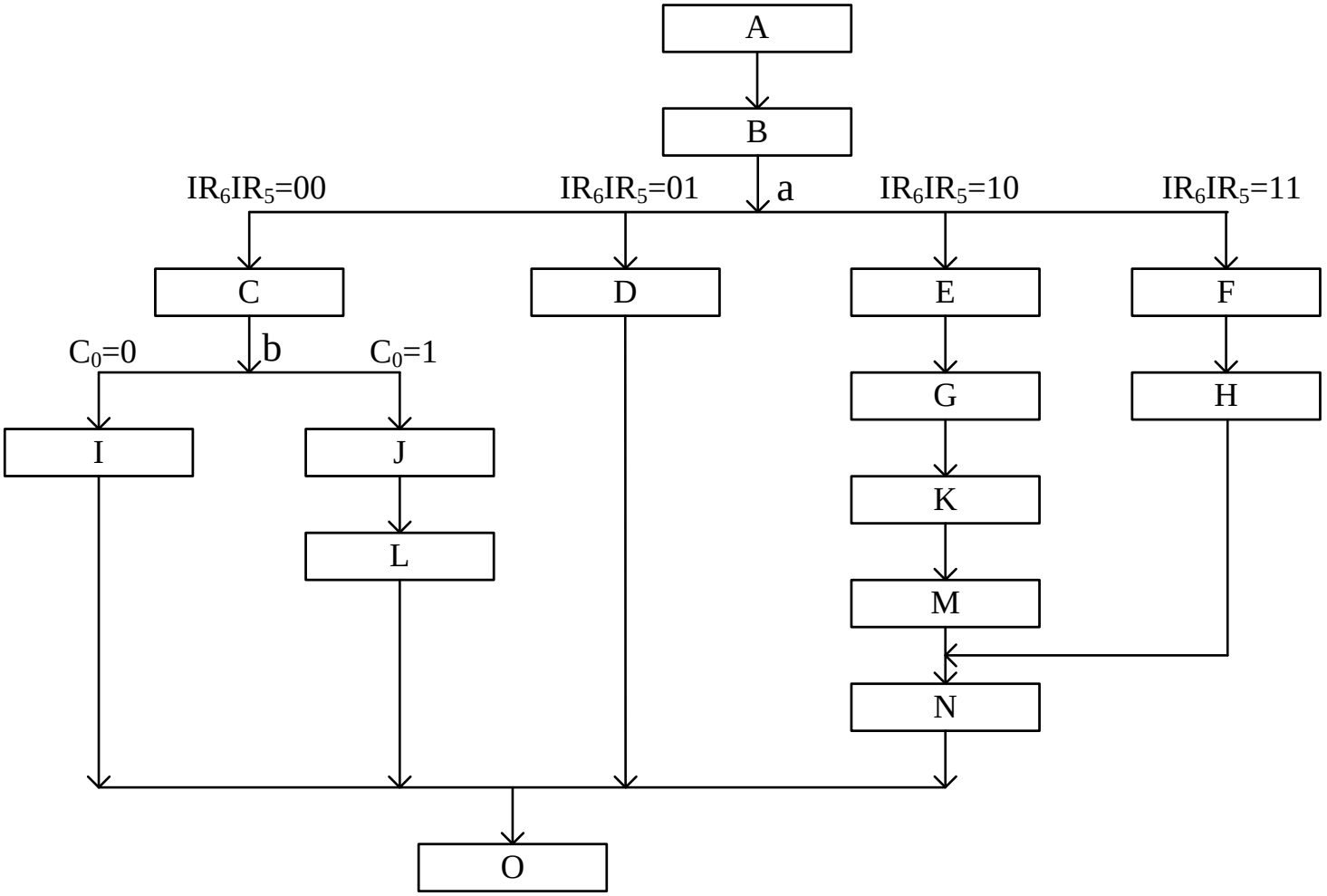
- 下图给出了某微程序控制计算机的部分微指令序列，图中每一框代表一条微指令。分支点a由指令寄存器IR的第5、6两位决定，分支点b由条件码C_0决定。已知微指令地址寄存器长度为8位。请采用增量与下址字段相结合方式实现微程序的顺序控制。具体要求如下：

  (1) 设计实现该微指令序列的微指令顺序控制字段格式；

  (2) 给出每条微指令的二进制编码地址；

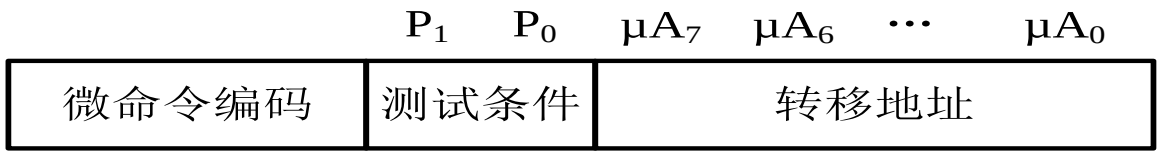
  (3) 设计微地址转移逻辑。

微指令顺序控制字段设计举例1（续）



微指令顺序控制字段设计举例1（续）

解：（1）该微指令顺序控制字段格式如下：



- P_1P_0=00，表示顺序执行；
- P_1P_0=01，表示对C_0进行测试，C_0=0，顺序执行，C_0=1，按转移地址跳转；
- P_1P_0=10，表示按照机器指令码中IR_5、IR_6的值实现功能转移；
- P_1P_0=11，表示按照转移地址无条件转移。

微指令顺序控制字段设计举例1（续）

微指令	微指令地址	微指令编码									
		μOP	P_1	P_0	μA_7	μA_6	μA_5	μA_4	μA_3	μA_2	μA_1	μA_0
A	00000000	—	0	0	X	X	X	X	X	X	X	X
B	00000001	—	1	0	1	0	0	0	0	0	0	0
C	10000000	—	0	1	1	0	0	0	0	0	1	0
D	10100000	—	1	1	0	0	0	0	0	0	1	1
E	11000000	—	0	0	X	X	X	X	X	X	X	X
F	11100000	—	0	0	X	X	X	X	X	X	X	X
G	11000001	—	0	0	X	X	X	X	X	X	X	X
H	11100001	—	1	1	1	1	0	0	0	1	0	0
I	10000001	—	1	1	0	0	0	0	0	0	1	1
J	10000010	—	0	0	X	X	X	X	X	X	X	X
K	11000010	—	0	0	X	X	X	X	X	X	X	X
L	10000011	—	1	1	0	0	0	0	0	0	1	1
M	11000011	—	0	0	X	X	X	X	X	X	X	X
N	11000100	—	1	1	0	0	0	0	0	0	1	1
O	00000011	—	—		—							

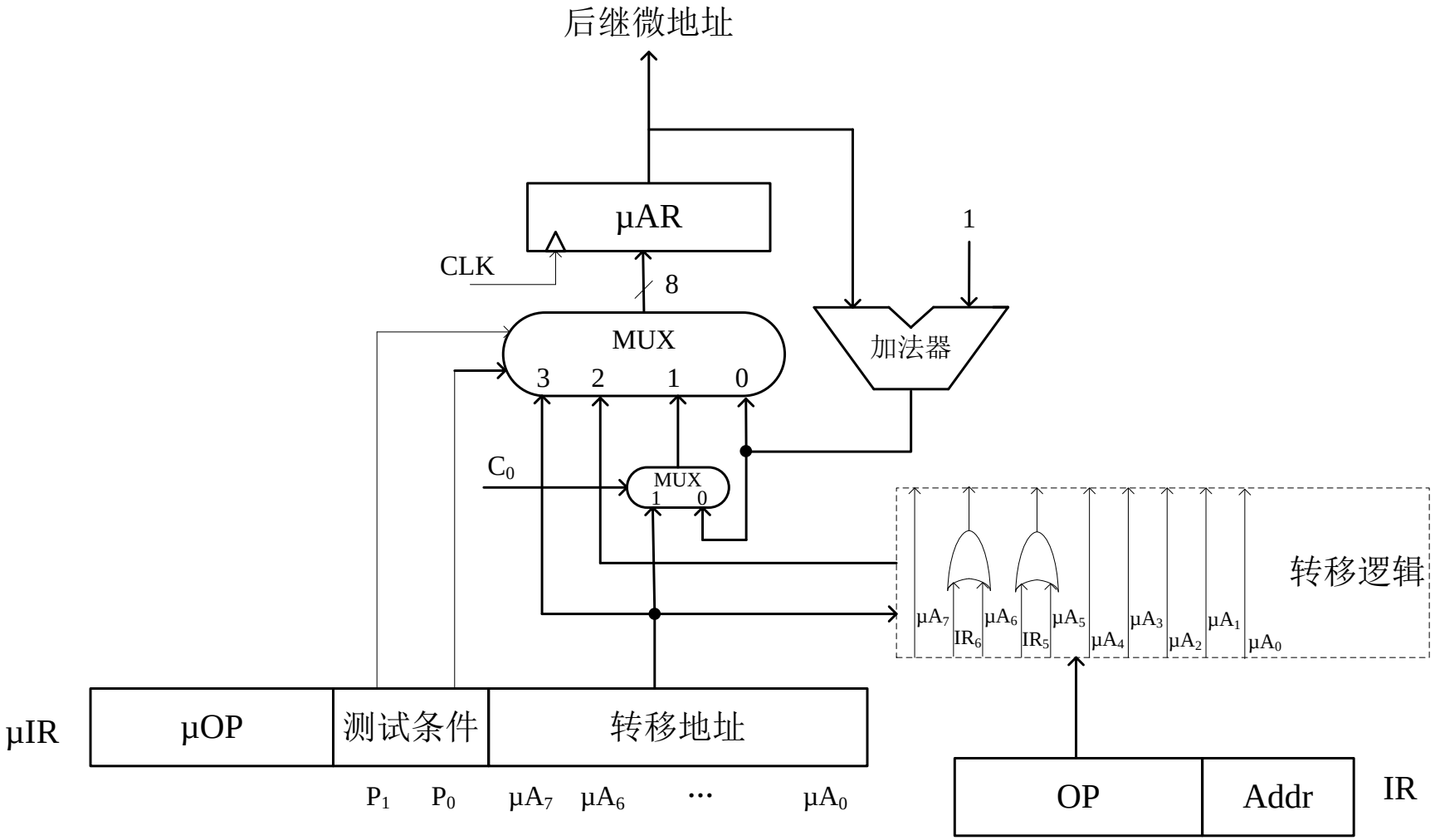
注：“—”表示不是该题目要求设计的内容。“X”表示其值任意的1位二进制代码，

微指令顺序控制字段设计举例1（续）

（3）后继微地址设计方案

- 分支a处是功能转移，需要由微指令顺序控制字段中给出的转移地址和转移逻辑共同产生后继微地址。一种可能的实现方案是：采用指令操作码相应位IR_6、IR_5直接作为后继微地址位μAR_6、μAR_5的取值。则分支a处的地址转移逻辑表达式为：
  - $\mu AR_6=IR_6$
  - $\mu AR_5=IR_5$
- 分支b处是两路分支转移，C_0=0时顺序执行，C_0=1时由转移地址字段直接给出后继微地址。
- 顺序执行时，后继微地址由μAR+1（μPC+1）产生。
- 后继微地址形成逻辑框图如下：

微指令顺序控制字段设计举例1（续）



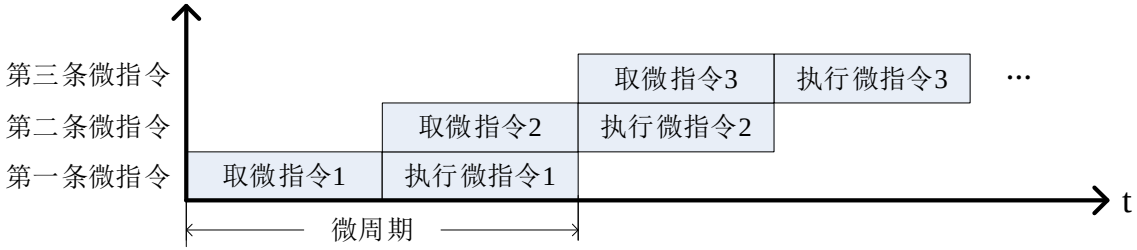
微程序控制单元的操作定时

- 微指令周期（微周期）
  - 取微指令：即将微指令从控存中取出
  - 执行微指令：完成微指令所规定的操作，即发出微命令。
- 微操作定时方式
  - 串行方式：按照先取微指令再执行微指令的顺序进行。
  - 重叠方式：将执行本条微指令与取下一条微指令在时间上重迭起来。

微程序控制单元的操作定时（续）

❒ 两种微指令执行方式基本时空关系

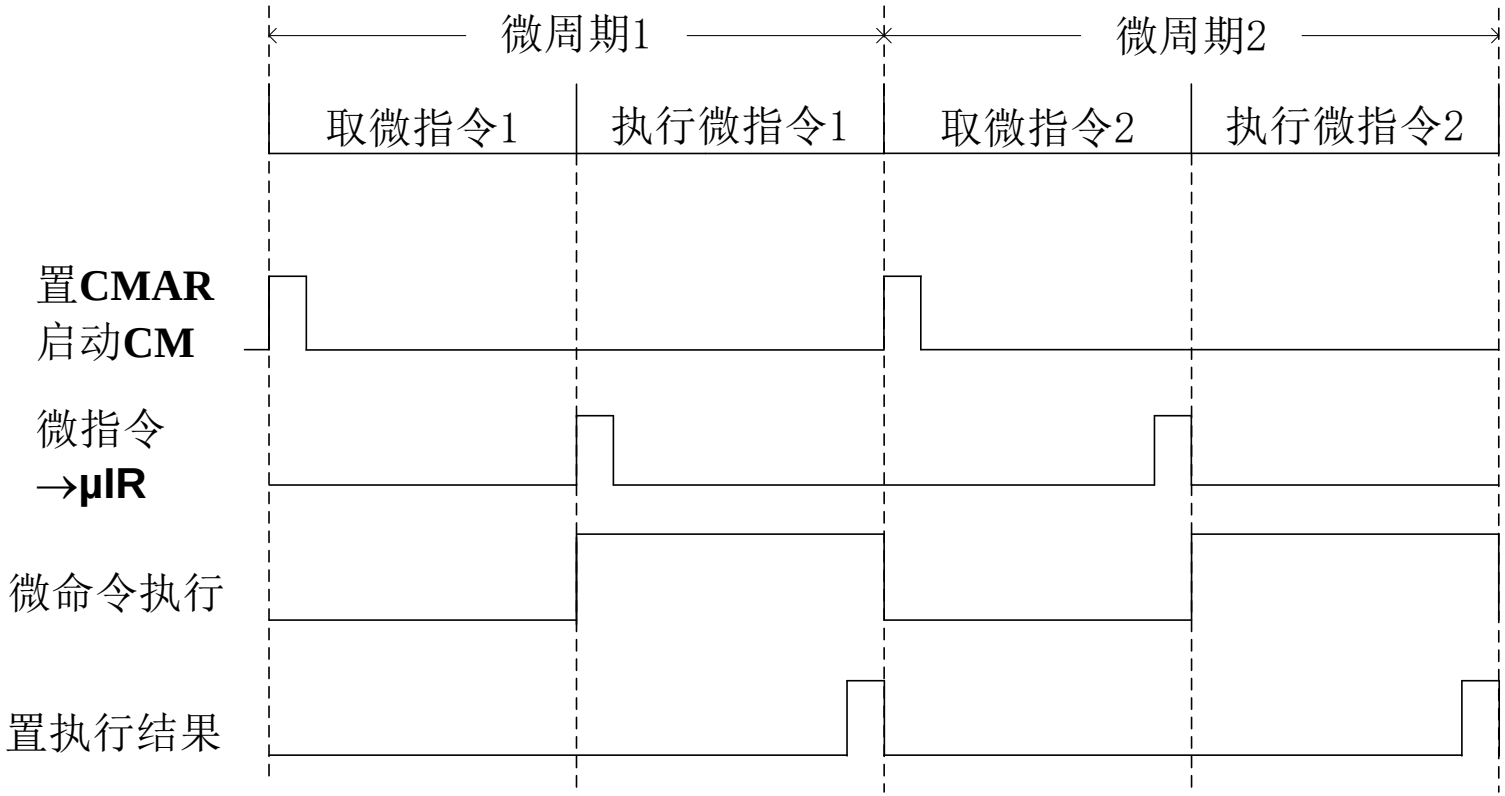
（a）串行方式



（b）重叠方式

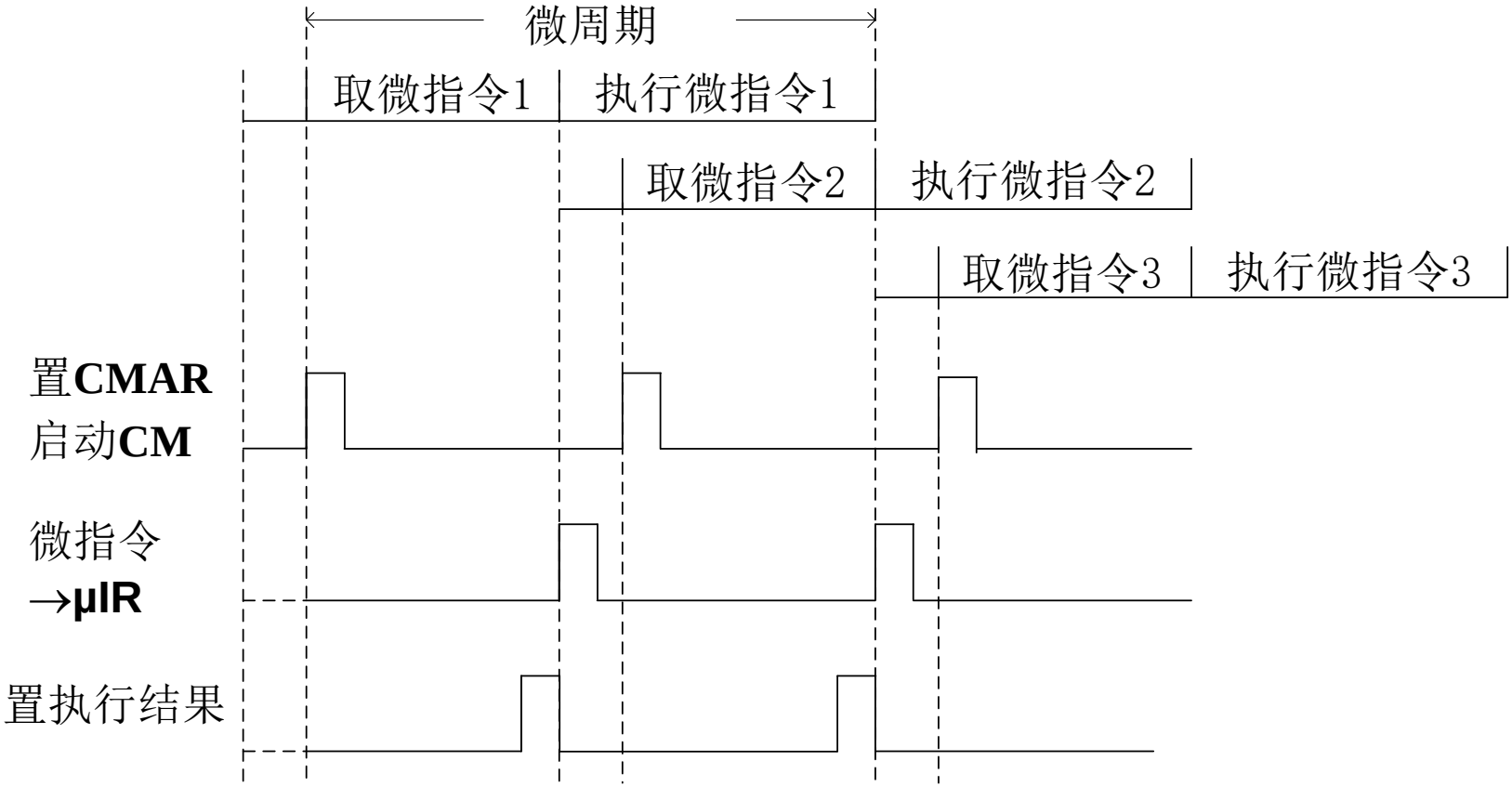
微程序控制单元的操作定时（续）

❒ 串行微指令操作时序（三相脉冲）

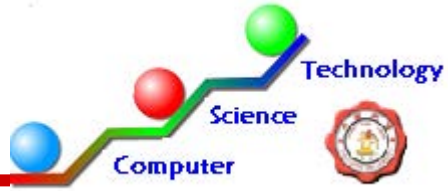


微程序控制单元的操作定时（续）

❒ 重叠微指令操作时序（三相脉冲）



目录



第七章 控制器

流水线控制器

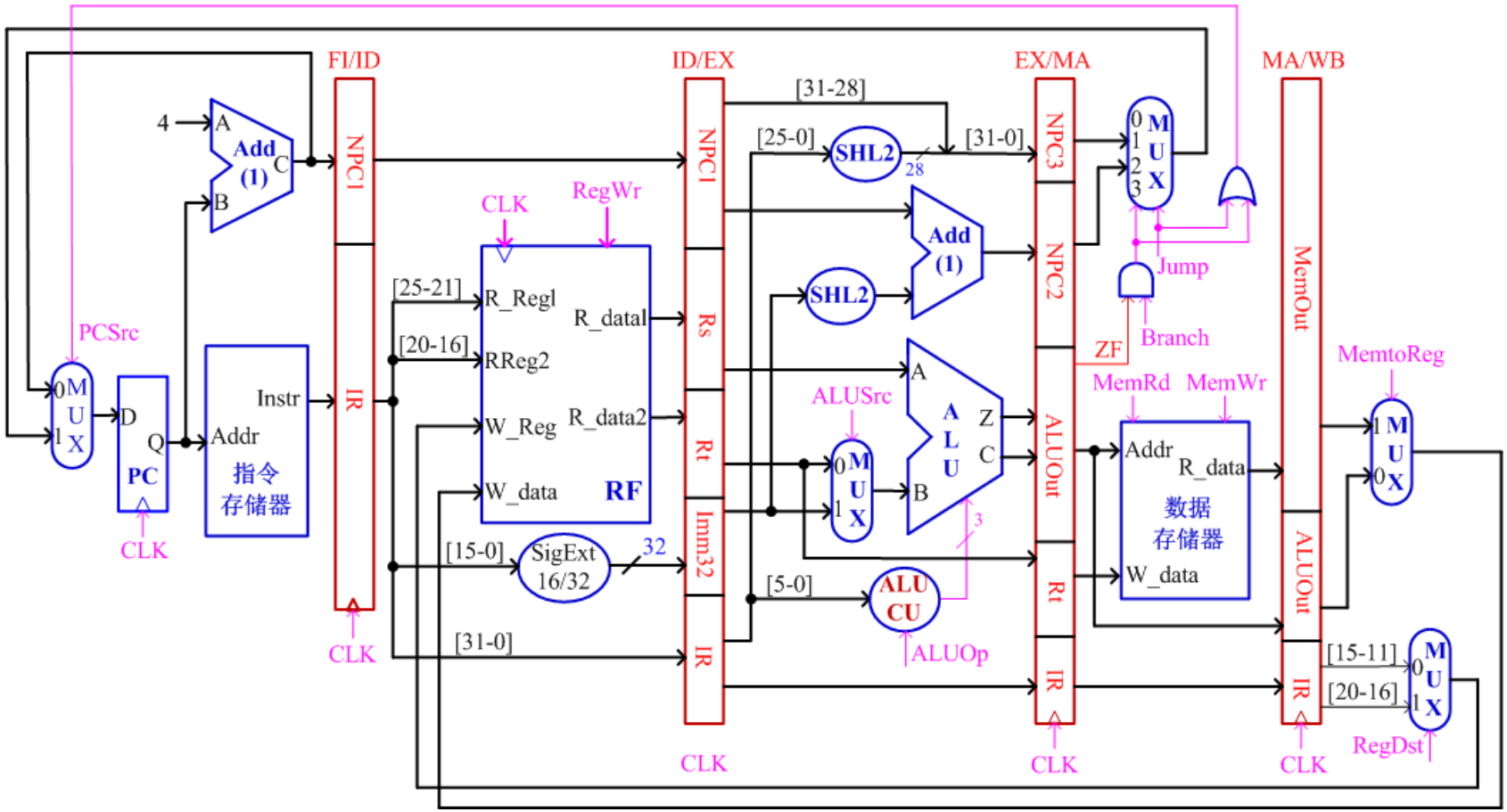
控制器对流水线实施的控制，就是在每一个流水步骤中设定相应控制信号的值。

❒ 流水线控制器基本结构

根据每个流水段的具体操作，分析每个流水段需要哪些控制信号。

- ❍取指令（FI）：执行读指令和修正PC
- ❍指令译码（ID）：指令译码和读取寄存器堆
- ❍执行指令（EX）：指令执行和地址计算
- ❍存储器访问（MA）：数据读/写以及转移判断
- ❍数据写回（WB）：将数据写入寄存器。

流水CPU基本数据通路



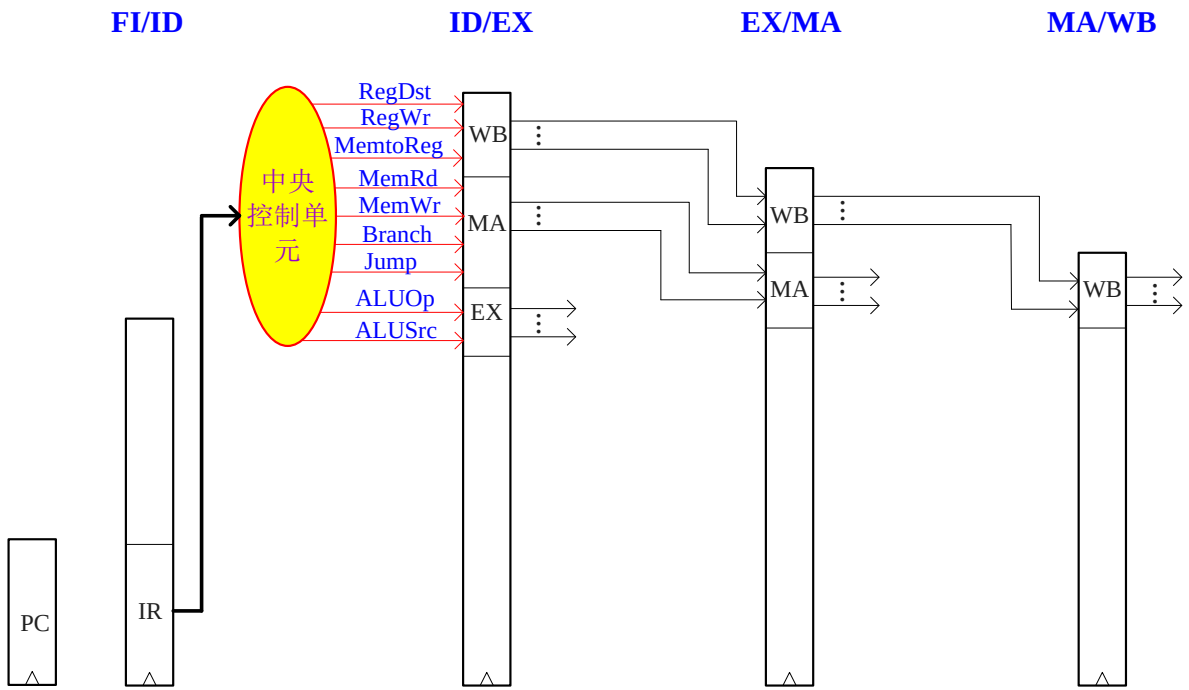
流水线控制器（续）

- 各种指令在不同流水段的控制信号
  - 五段流水线中第一（FI）和第二段（ID）控制信号完全相同；
  - 将第三（EX）、第四（MA）、第五（WB）流水段需要的控制信号分成三组。

指令	EX段控制信号				MA段控制信号				WB段控制信号	
	RegDst	ALUOp1	ALUOp0	ALUSrc	Jump	Branch	MemRd	MemWr	RegWr	MemtoReg
R型	1	1	0	0	0	0	0	0	1	0
lw	0	0	0	1	0	0	1	0	1	1
sw	X	0	0	1	0	0	0	1	0	X
j	X	X	X	X	1	0	0	0	0	X
beq	X	0	1	0	0	1	0	0	0	X

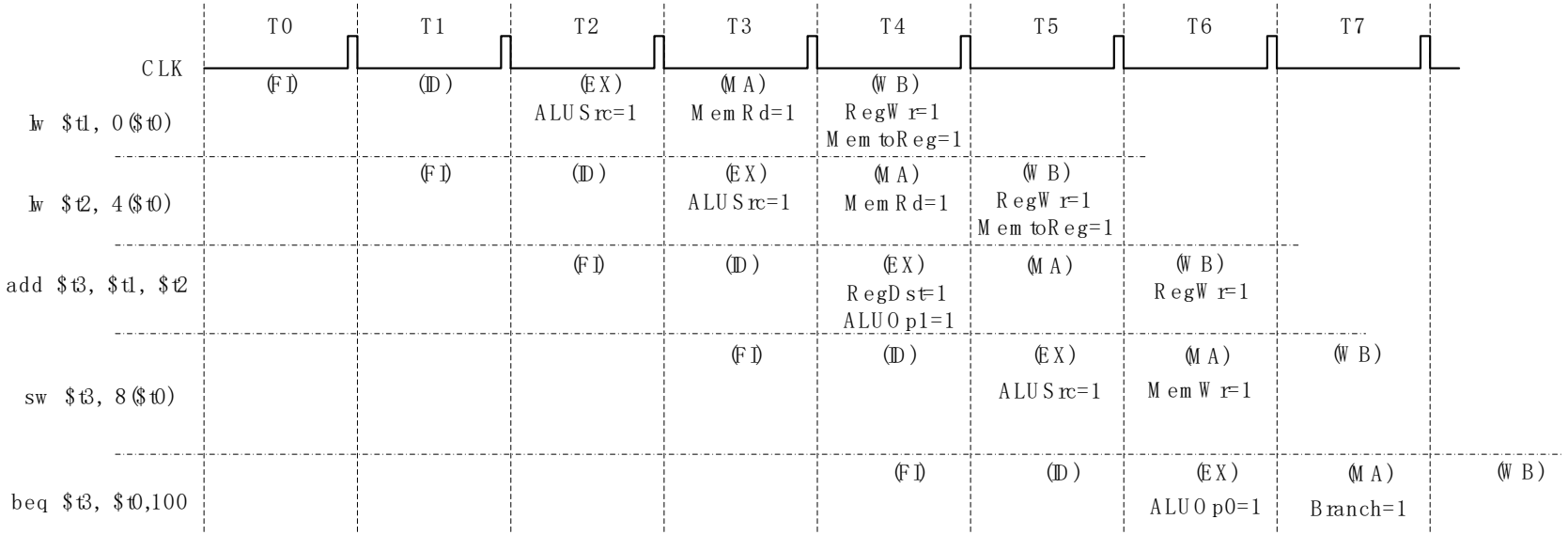
流水线控制器（续）

❒ 控制信号的存储与分发



流水线控制器（续）

❒ 流水线控制信号举例



本章作业（总第11次作业）

- 7.10、7.16、 7.18