

第四章 中央处理器

本章主要内容

课前思考：

- 1) 计算机如何能够自动执行？
- 2) 运行中的程序在哪里存储？
- 3) 控制器怎么逐条取出指令，分析、执行？

本章讲述CPU的基本组成、基本功能、硬布线控制器的组成原理与实现方法、微程序控制器的组成原理与实现方法、流水线基本原理、控制器的控制方式等内容。要求重点掌握两种控制器的设计思路及实现方法、流水线基本原理。

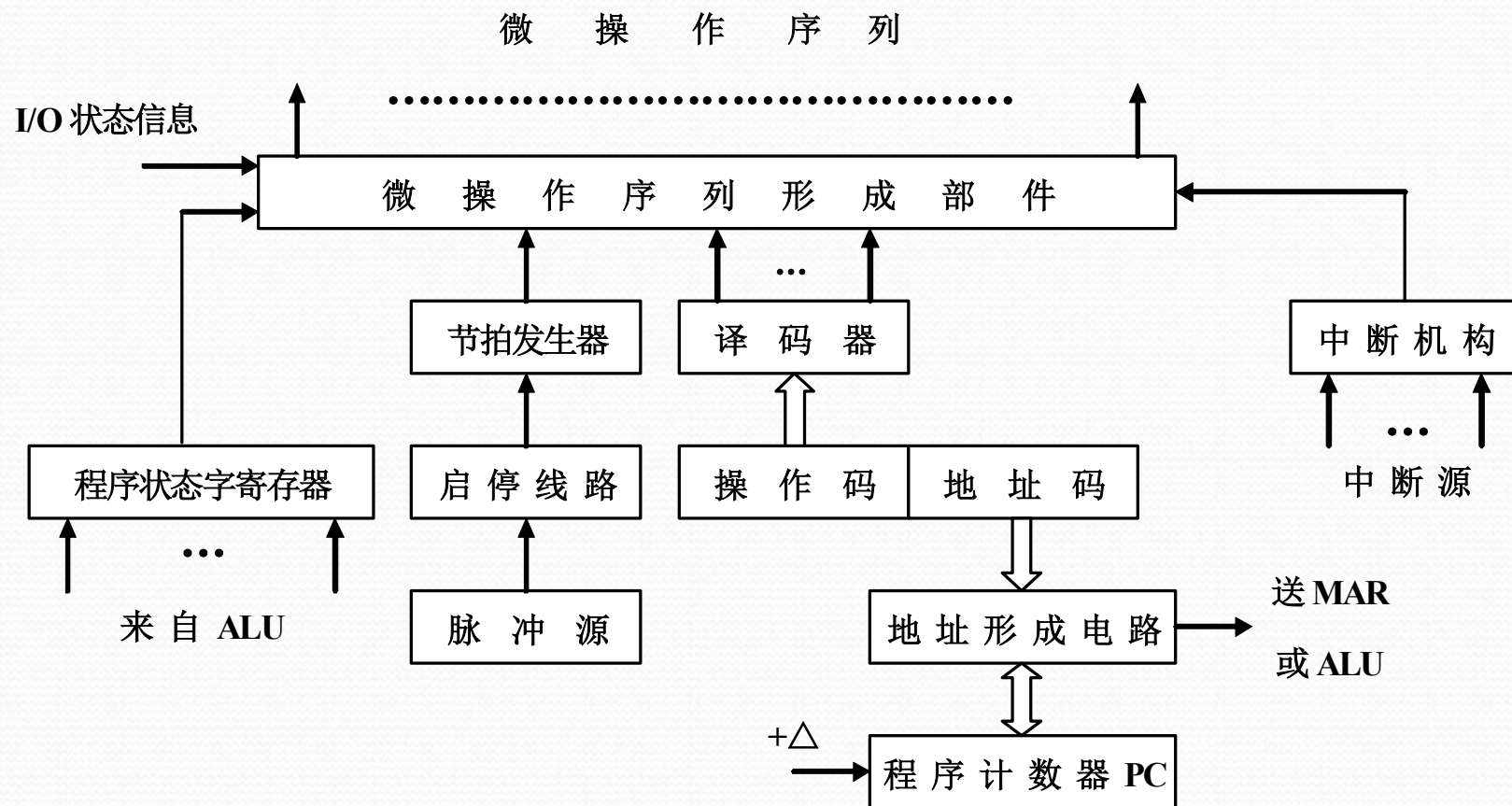
第四章 中央处理器

- 4.1 CPU的基本功能及结构
- 4.2 指令的执行过程
- 4.3 硬布线控制器
- 4.4 微程序控制器
- 4.5 流水线原理
- 4.6 控制器的控制方式

CPU的功能

- 1). 取指令
- 2). 分析指令
- 3). 执行指令
- 4). 控制程序和数据的输入与结果输出
- 5). 随机事件和某些特殊请求的处理

控制器的基本组成



控制器基本组成框图

指令部件

完成取指令并分析指令。包括以下部分：

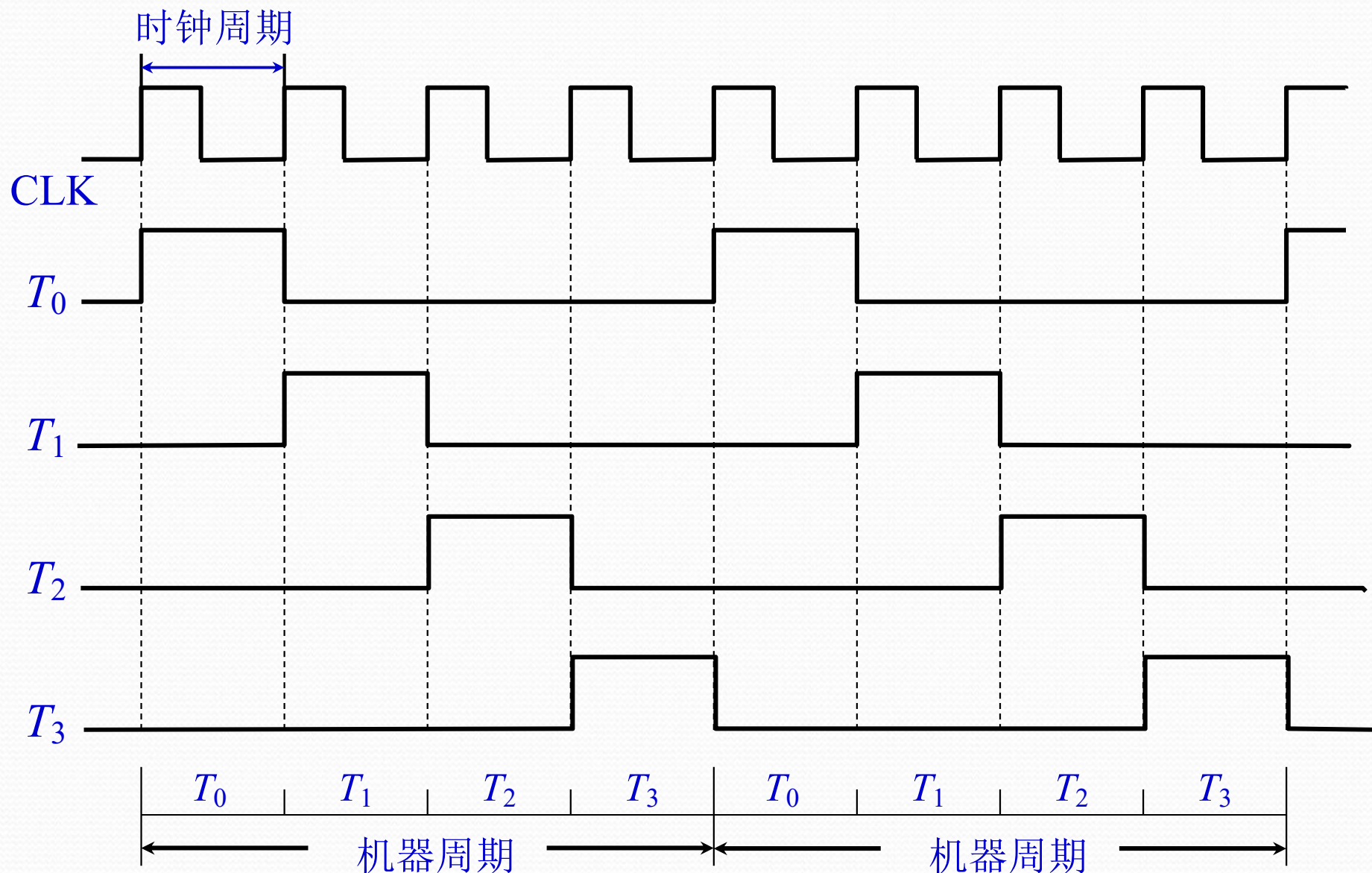
- (1) **程序计数器PC**：程序计数器又称指令计数器，用来存放当前指令或接下来要执行的指令的地址。指令地址的形成分两种：对于顺序执行的情况，PC的值应不断的增量（ $+\Delta$ ），增量的功能可由指令计数器自己完成，也可由运算器完成；对于非顺序执行的程序，一般由转移类指令将转移目标地址送往程序计数器。
- (2) **指令寄存器IR**：指令寄存器用来存放从存储器中取出的指令。
- (3) **指令译码器ID**：指令译码器又称指令功能分析解释器，对指令寄存器中的指令操作码进行分析、解释，并产生相应的控制信号送给微操作形成部件。
- (4) **地址形成部件**：根据指令寻址方式，形成操作数有效地址。
- (5) **存储数据寄存器MDR**：用来存放将被读取或写入的数据。
- (6) **存储地址寄存器MAR**：用来存放将被访问的存储单元的地址。

时序部件

时序部件能产生一定的时序信号，以保证计算机的各功能部件有节奏的运行。计算机工作的过程就是不断地执行指令的过程，执行一条指令的全部时间称为**指令周期**，通常将指令周期划分为若干个**机器周期**，每个机器周期又分为若干个**节拍**，每个节拍中又包含一个或几个**工作脉冲**。包括以下部分：

- (1) **脉冲源**:用来产生一定频率和宽度的时钟脉冲信号作为整个机器的基准时序脉冲(也称为机器的主脉冲)，实际上，它是具有一定频率的方波或窄脉冲信号发生器，其工作频率称为“计算机主频”。
- (2) **启停线路**:计算机加电后，立即产生一定频率的主脉冲，但这并不意味着计算机已经开始工作，而是要根据计算机的需要，在启停线路的作用下可靠地开放或封锁脉冲，实现对计算机安全可靠的启动和停机。
- (3) **节拍信号发生器**:又称脉冲分配器，其主要功能是按时间先后次序，周而复始地发出各个机器周期中的节拍信号，用来控制计算机完成每一步微操作。常用计数器、移位器等多种线路构成。

时钟周期(节拍、状态)



微操作信号发生器

微操作是一个指令周期中最基本的不可再分割的操作，不同的机器指令具有不同的微操作序列。微操作信号发生器就是用来产生微操作序列，根据微操作序列的形成方式不同，控制器可分为硬布线控制器和微程序控制器。不管是哪种控制器，微操作序列的形成都主要与下列信号有关：

- (1) 指令寄存器中的操作码经过指令译码器后所产生的译码信号
- (2) 时序部件提供的时序信号
- (3) 程序状态字寄存器PSWR中的状态字所提供的状态信号
- (4) 中断机构输出的信号

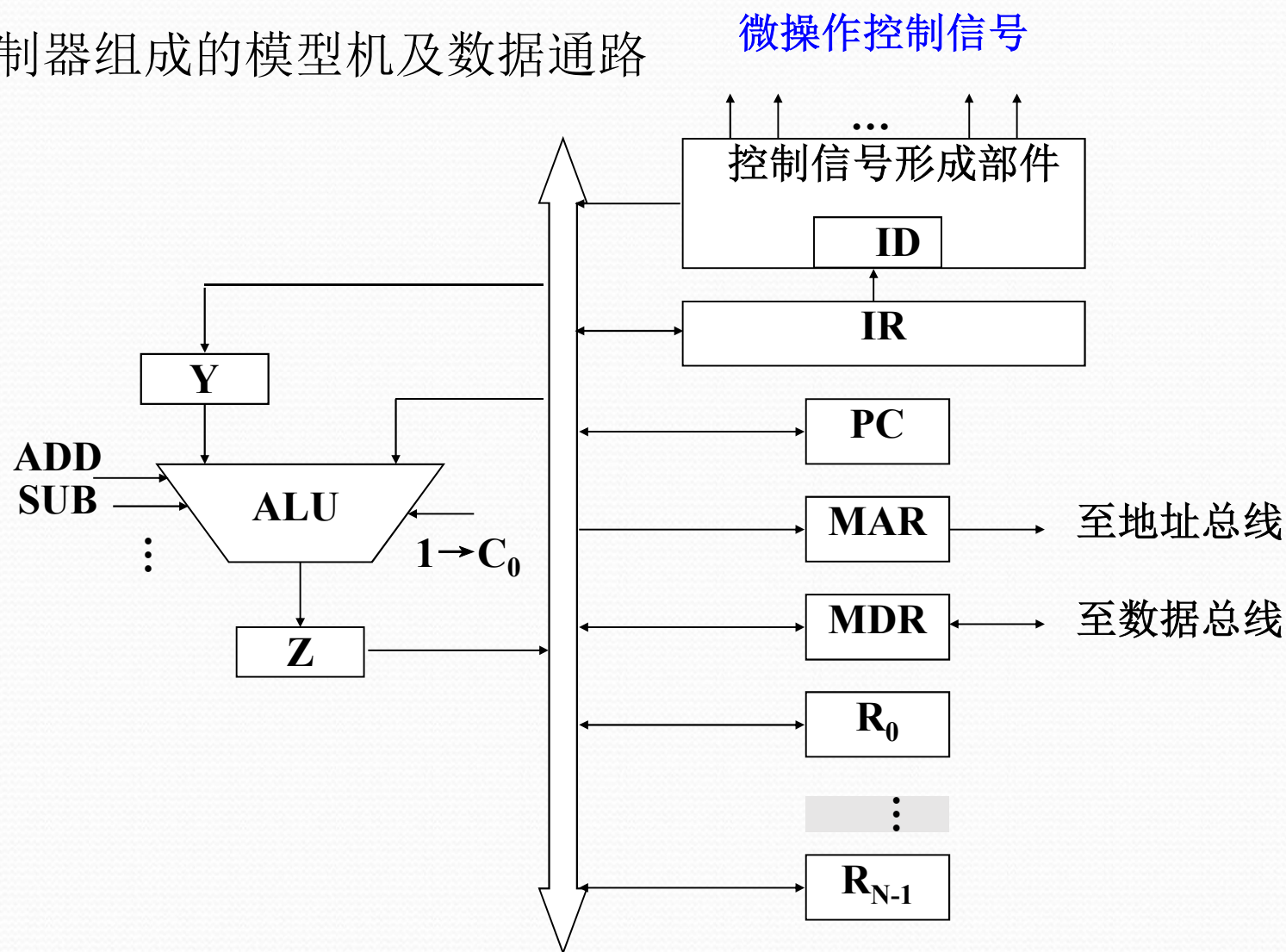
中断机构

响应和处理中断的逻辑线路称为中断机构，负责处理异常情况和特殊请求。

由于大规模集成电路的发展，目前计算机的体系结构发展很快，所有计算机都使用如指令预取、流水线等技术，以提高计算机系统的性能。

控制器结构示例

例：控制器组成的模型机及数据通路



第四章 中央处理器

- 4.1 CPU的基本功能及结构
- 4.2 指令的执行过程
- 4.3 硬布线控制器
- 4.4 微程序控制器
- 4.5 流水线原理
- 4.6 控制器的控制方式

时序系统

时序系统是控制器的核心，为指令的执行提供各种定时信号。主要包括以下部分：

1. 指令周期
2. 机器周期
3. 节拍
4. 工作脉冲

指令周期

计算机之所以能自动地工作，是因为CPU能从存储器中取一条指令并执行这条指令，紧接着又是取指令、执行指令，如此周而复始，构成了一个封闭的循环。

CPU每取出并执行一条指令，都要完成一系列的操作，这一系列操作所需要的时间叫做一个指令周期。更简单地说，指令周期是指执行一条指令（包括取指令、分析指令和执行指令）所需的全部时间，不同指令的指令周期是不同的。机器周期 通常它是主存储器的一个访问周期。

机器周期

按照指令执行的各项不同任务，把一个指令周期划分成若干个时间段，每个时间段完成一个基本操作，这样的时间段称为**机器周期**。在采用混合控制方式的现代计算机中，一般的做法是定义几种基本的机器周期，例如：取指令机器周期，读存储器机器周期，写存储器机器周期和执行运算机器周期等。一个指令周期可包括不同数量、不同类型的几个机器周期，但是任何一条指令的第一个机器周期必须是取指令机器周期。

机器周期又可称作**CPU周期**，通常它是主存储器的一个访问周期。

节拍

- 在每个机器周期内要执行若干个微操作，这些操作可能需要分成几步完成，例如按变址方式读取操作数，先要进行变址运算才能访存读取。所以应把一个机器周期分为若干个相等的时间段，以便对应一个机器周期中规定的微操作，这样的每个时间段都对应一个电位信号，称为节拍电位信号。“节拍”是计算机操作的最小时间单位，又可称作时钟周期或T周期。
- 节拍的宽度取决于CPU执行一次微操作所需要的时间，不同的机器周期内所包括的节拍数可以是固定不变的，也可以是可变的。

节拍选取方法

(1) **统一节拍法**：这种方式一律以最复杂指令的机器周期为标准，确定节拍数，即这种指令有多少微操作，机器周期就包括多少个节拍。**每个节拍的长度为最复杂的微操作所需的时间**。所以，这种方式仅在各种指令执行时间相差不大的情况下才比较合适。

(2) **分散节拍法**：根据每条指令某个机器周期的实际需要安排节拍数，**需要多少节拍**，控制器中的节拍发生器**就发出多少节拍**。

(3) **延长节拍法**：这种方式**在照顾多数机器周期要求的情况下**，选取适当的节拍数作为基本节拍，然后根据指令的需要，每拍都可以**适当的延长一或两拍**。

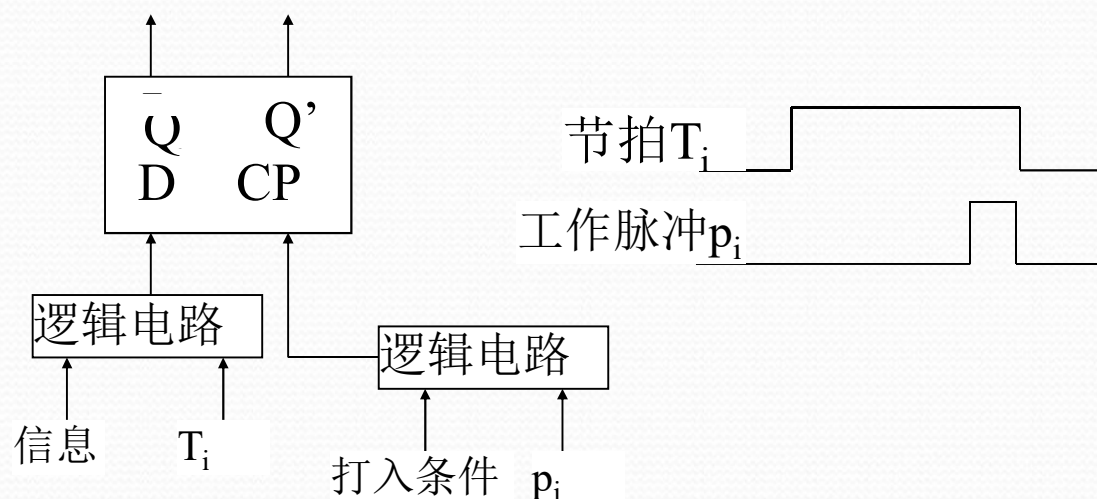
(4) **时钟周期插入**：在一些微型机中，时序信号中不设置节拍，而直接使用时钟周期信号，即**一个机器周期中含有若干个时钟周期且还可以不断插入等待周期**。

工作脉冲

节拍电位信号提供了一项操作所需要的时间分段，它是信息的载体，在数据通路传输中起着开门或关门的作用。但在一个节拍中，有的操作还需要严格的定时脉冲，如将稳定的运算结果打入寄存器、机器周期状态切换等。所以在节拍之内往往还需要设置几个工作脉冲，作为各种同步脉冲的来源。

工作脉冲

通常触发器使用电位-脉冲工作方式，节拍电位控制信息送到D触发器的D输入端，工作脉冲CP将输入端的数据打入D触发器。两者的配合关系如下图所示。



节拍电位和工作脉冲的配合关系

指令的执行过程

指令的执行通常都可以分为三个阶段：

- 1) 取指令
- 2) 分析指令
- 3) 执行指令

取指令

任何一条指令的执行，都必须经过取指令阶段，该阶段主要是将指令从主存中取出放入CPU内部的指令寄存器中。

(1) 将程序的启动地址，即第一条指令的地址置于程序计数器PC中

(2) 将PC中的内容送至主存的地址寄存器MAR，并送地址总线AB

(3) 向存储器发读命令。读取指令时CPU是空闲的，利用这段时间完成 $PC + \Delta$ 的操作，为指令的连续运行做准备

(4) 从主存中取出的指令经过主存的数据寄存器MDR，再经过数据总线进入CPU中的指令寄存器中

分析指令

取出指令后指令译码器对保存在IR中的指令操作码进行译码，产生译码信号并送微操作序列形成部件，进而产生微操作序列送运算器、存储器、外设及控制器本身。对于无操作数指令，只要识别出是哪一条指令，便可转入其执行指令阶段。而对于带操作数指令，则要根据具体的寻址方式去指令中、寄存器中或存储单元中寻找操作数，若操作数位于存储单元中，则首先要计算出操作数的物理地址，计算方法也得根据具体的寻址方式而定。

执行指令

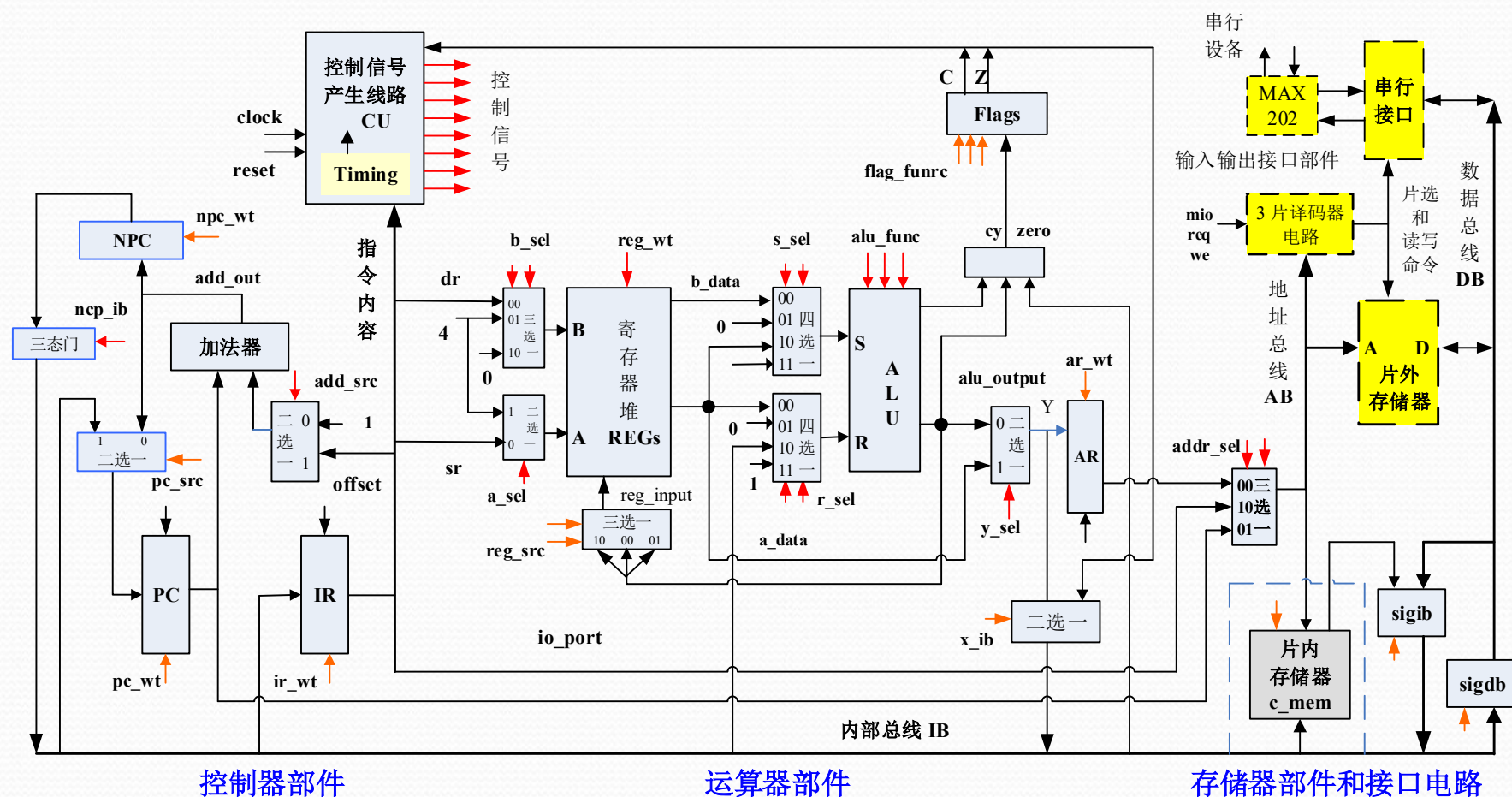
根据分析指令阶段所产生的微操作序列，控制运算器、存储器、外设及控制器本身完成指令规定的各种操作。

控制器在实现一条指令的功能时，总是把每条指令分解成一系列时间上先后有序的最基本、最简单的微操作，即微操作序列。

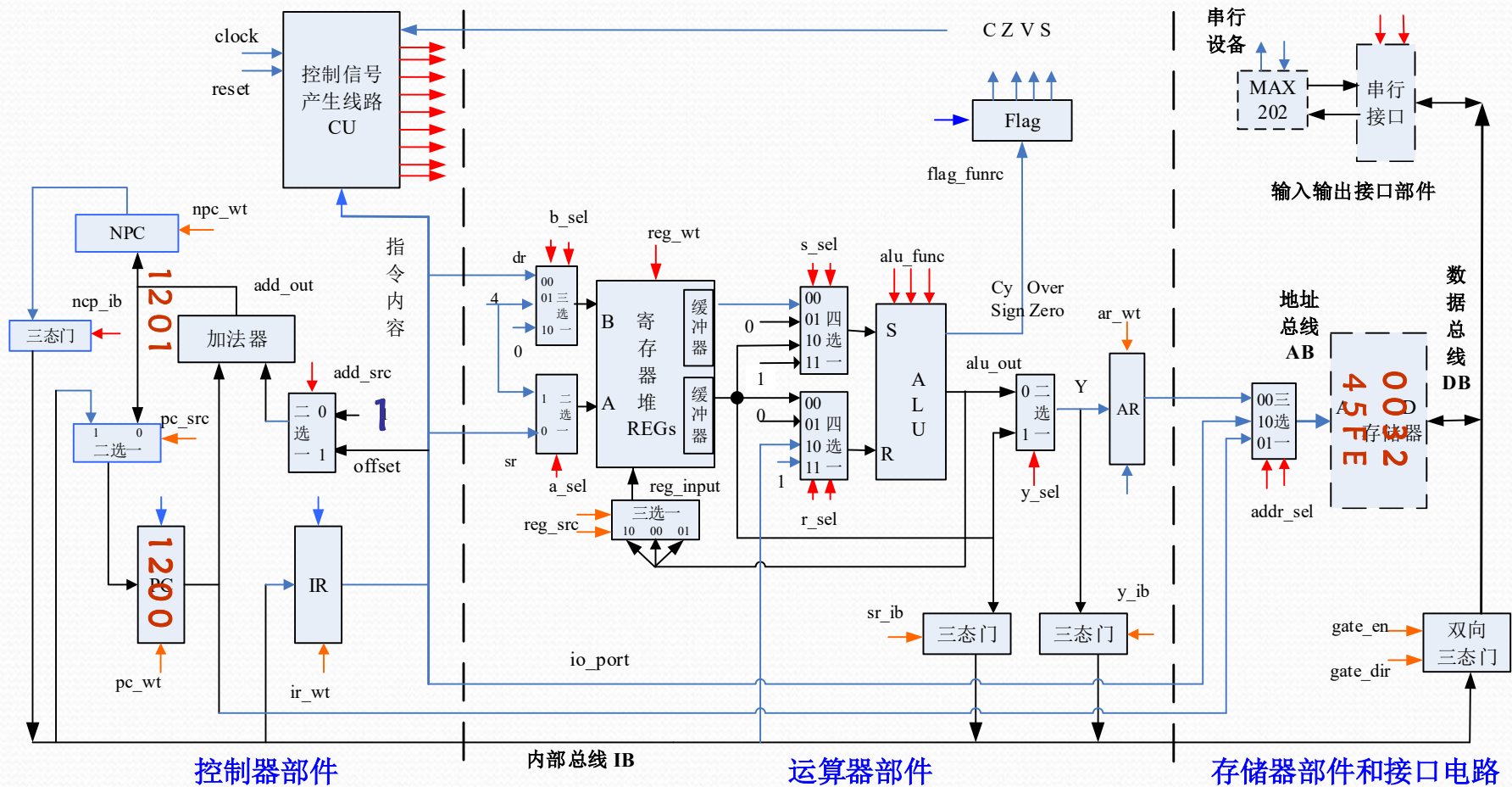
下面通过两个简单的模型机来看具体指令的执行过程，即指令的微操作序列。

例1. 简化TEC-2000 整机系统组成框图。

简化TEC-2000 整机系统组成框图



下面看一看这套指令系统的几条指令的执行步骤，并通过这张整机框图观察其信息传送的时空关系。



取指操作:在取指周期，以PC作地址读内存，读出指令送IR，计算下一条指令地址

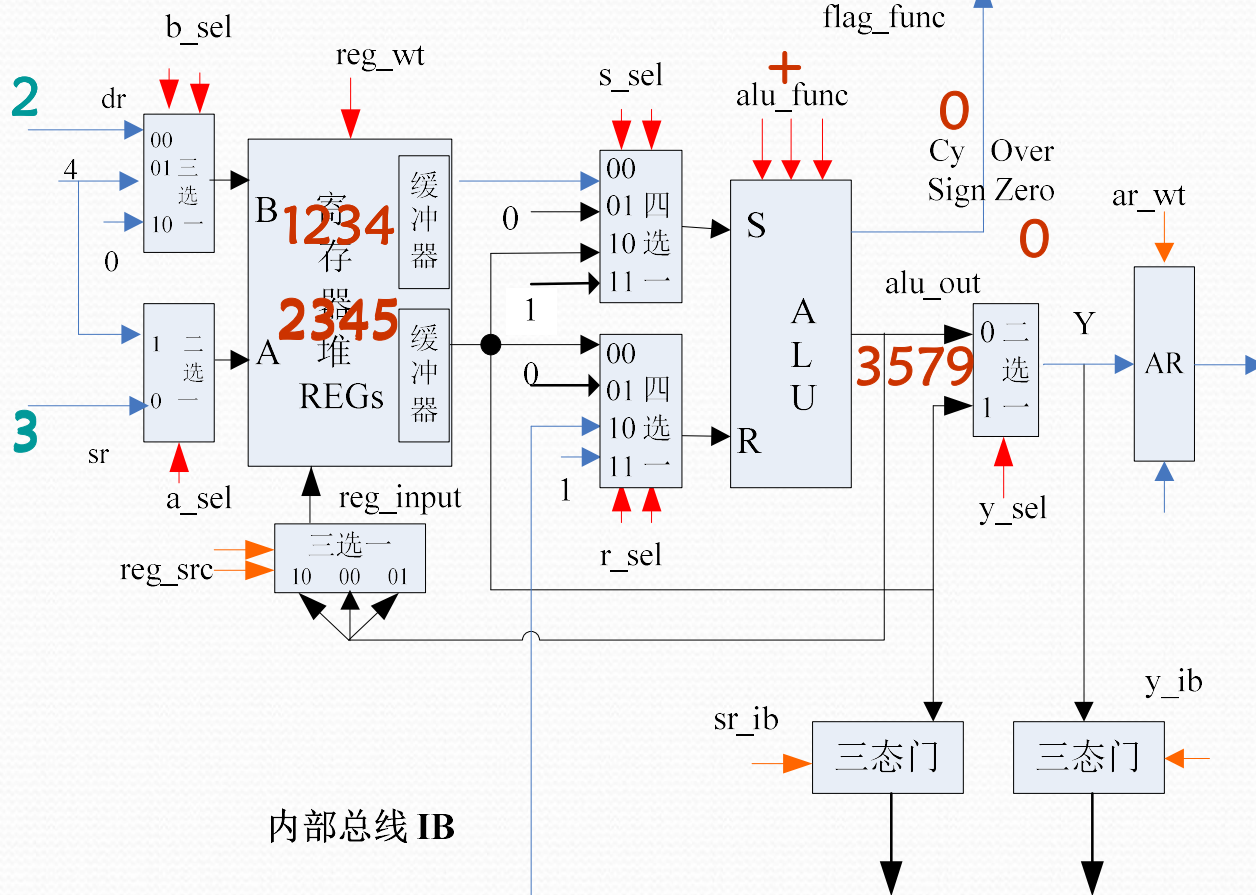
假设PC的内容为1200，内存1200单元的内容为0023，IR的内容未定

实现功能： $pc \rightarrow AB$ ， $mem[AB] \rightarrow ir$ ， $pc+1 \rightarrow pc$

控制信号： $addr_sel=01$ ， $gate_en=1$ ， $mio=1$ ， $ir_wt=1$ ， $pc_wt=1$

ADD R3, R2

00000000 0011 0010



动画演示

REGs(3) + REGs(2)

→ REG(3)

假设R2的内容为1234

R3的内容为2345

控制信号:

b_sel 选00

a_sel 选00

s_sel 选00

r_sel 选00

alu_func 选000

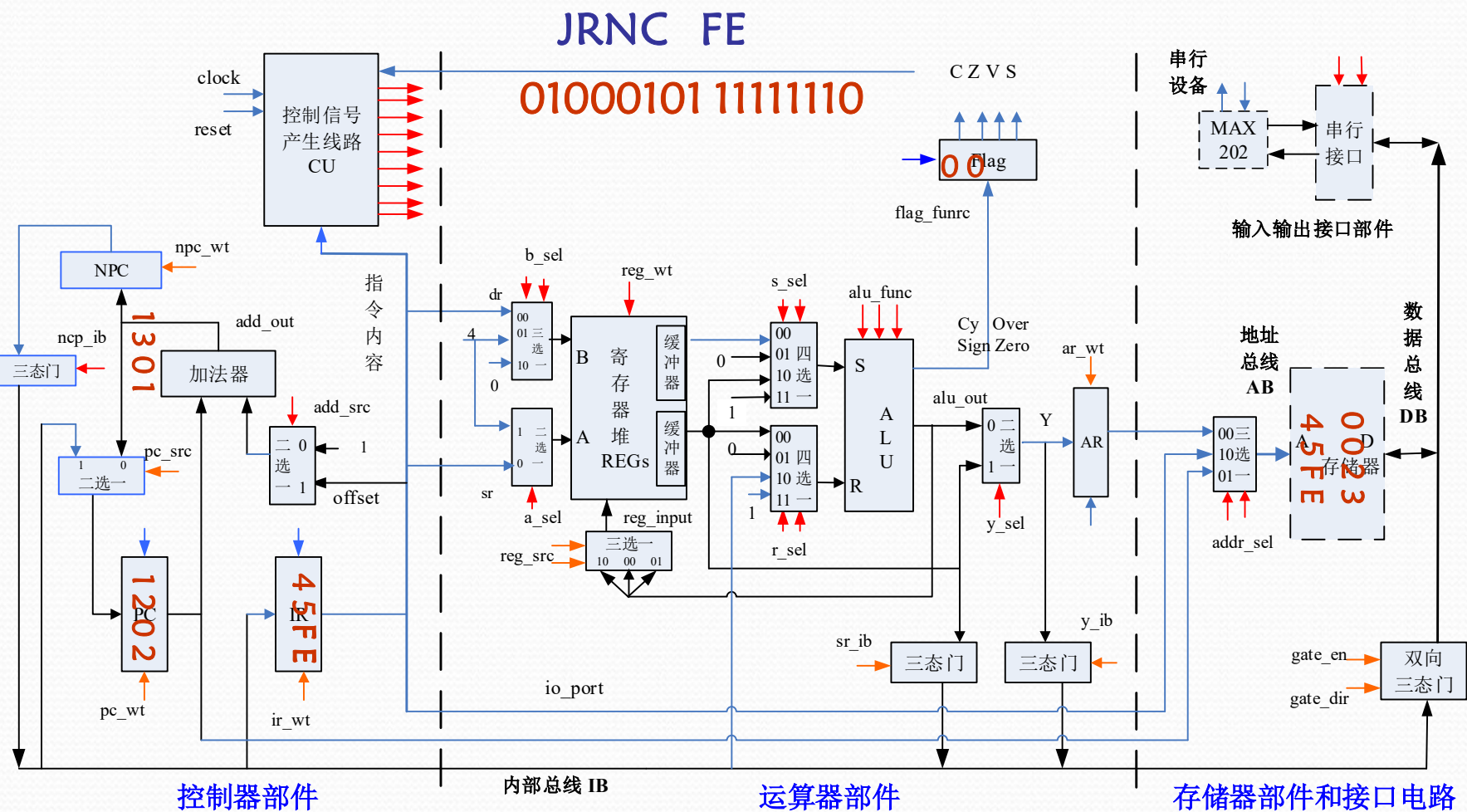
reg_src 选00

reg_wt 选1

flag_func 选001

运算器部件

在执行周期，完成在取指周期读出来的加法指令的具体运算过程

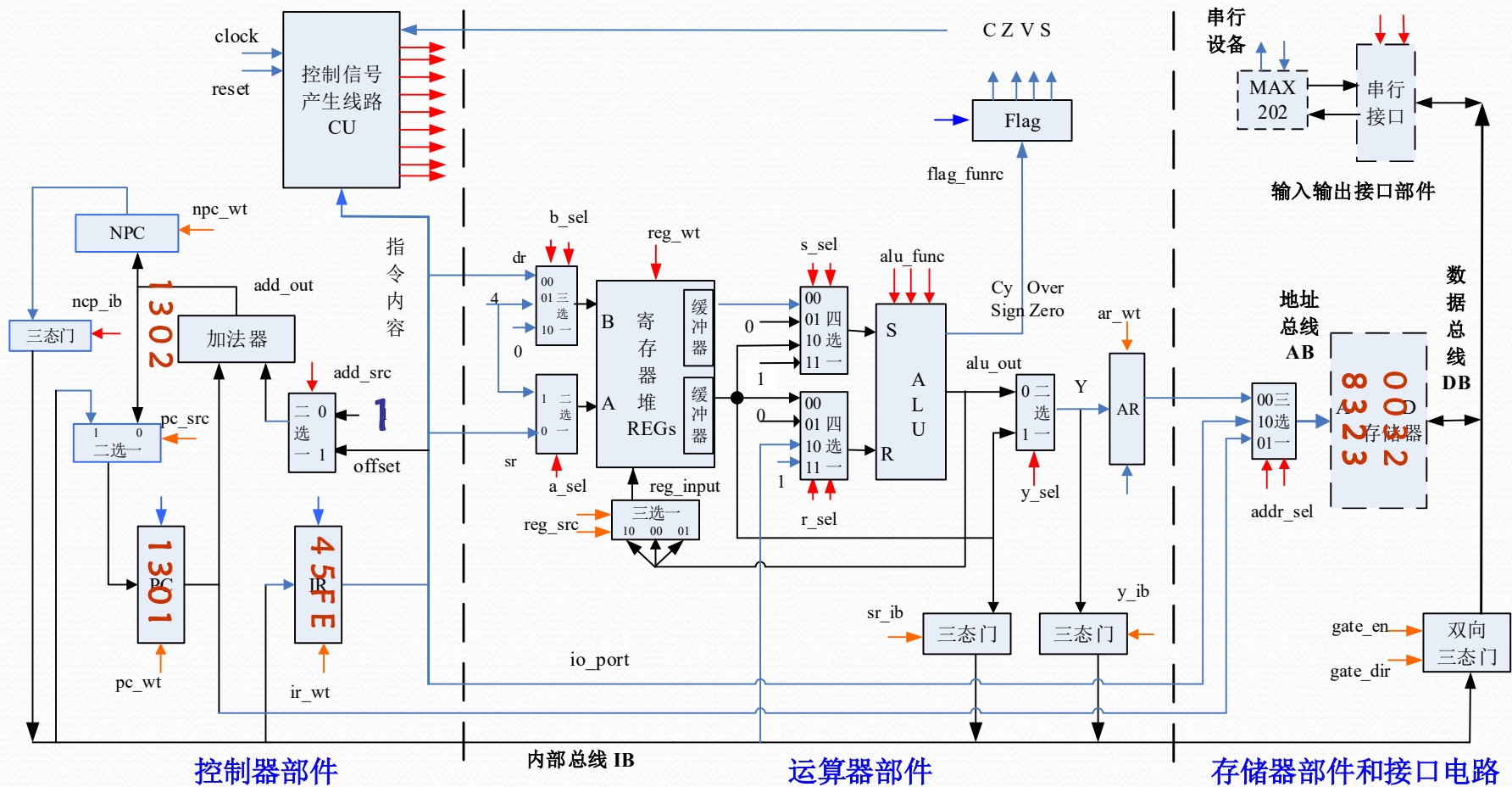


相对转移指令：标志位C为0时，转移到1200地址，否则顺序执行

在执行周期，由控制器部件计算转移指令地址，若C=0则把转移地址送入PC

实现功能：if nc then pc+offset→pc

控制信号：add_sel=1, pc_wt=1



取指操作:在取指周期，以PC作地址读内存，读出指令送IR，计算下一条指令地址

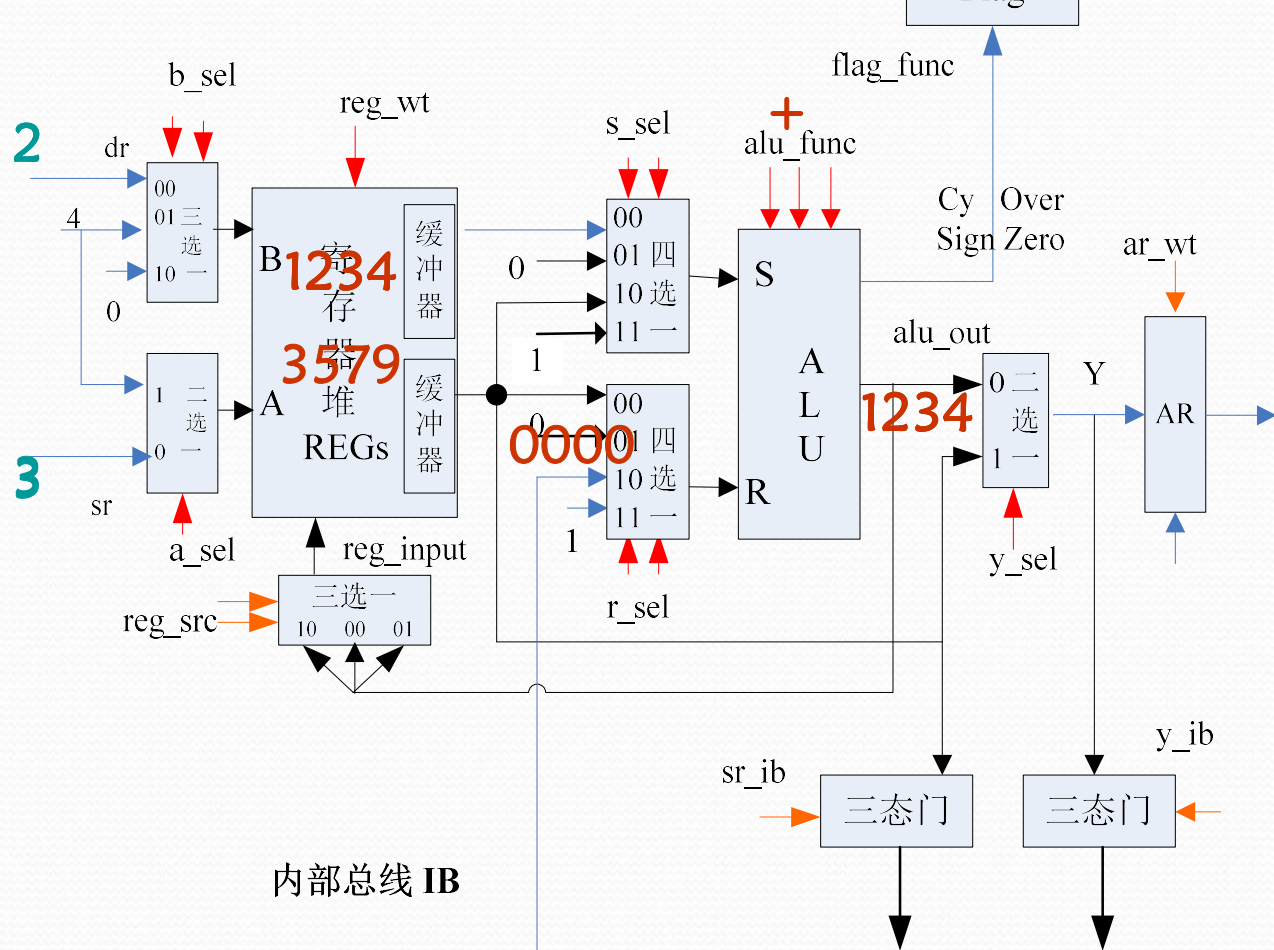
此时PC的内容为1301，IR的原内容45FE，假设内存1301单元的内容为8323

实现功能： $pc \rightarrow AB$ ， $mem[AB] \rightarrow ir$ ， $pc+1 \rightarrow pc$

控制信号： $addr_sel=01$ ， $gate_en=1$ ， $mio=1$ ， $ir_wt=1$ ， $pc_wt=1$

STRR [R2], R3

10000011 0010 0011



动画演示

在执行周期，在运算器部件中完成把R2的内容送地址寄存器AR的操作

R2的内容为1234

R3的内容为3579

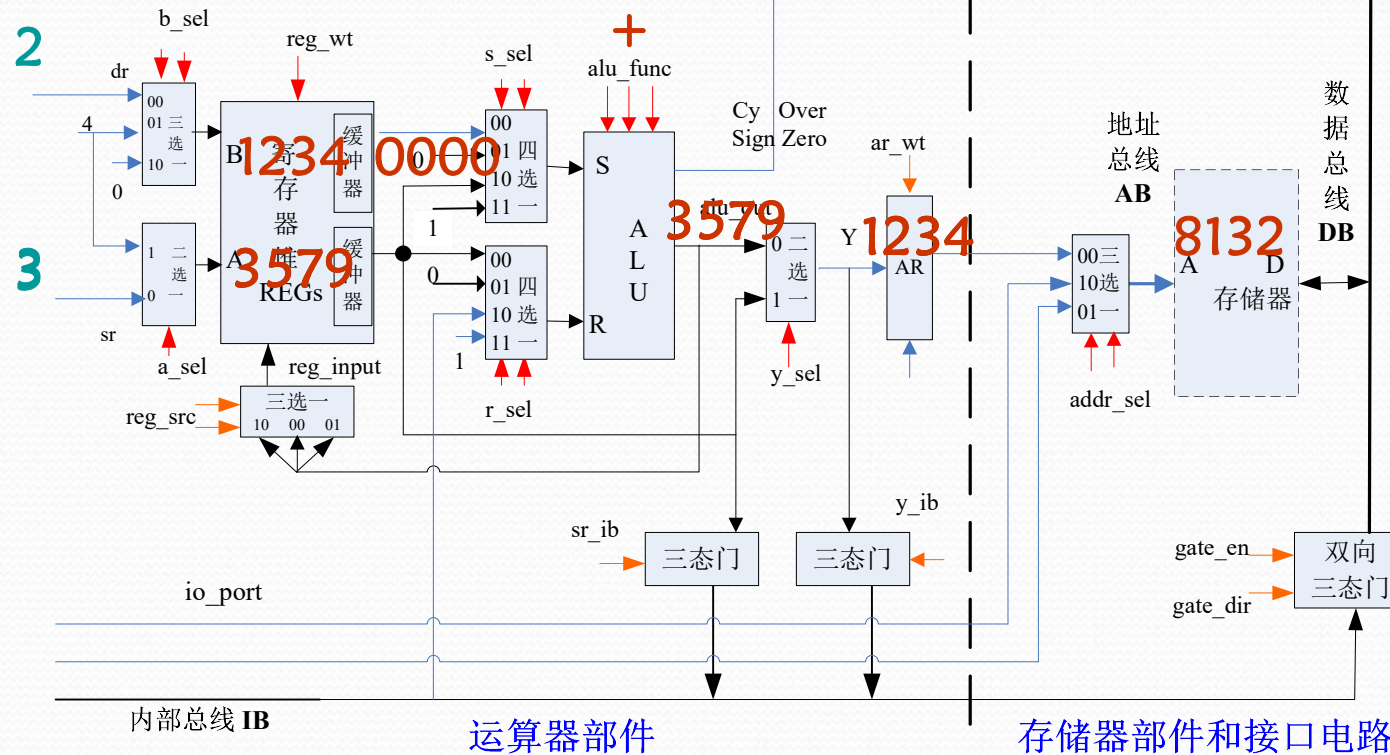
请注意，此步骤中是计算数据在内存中的单元地址，通过 $R2 + 0 \rightarrow AR$ 实现，不影响C、Z等标志位；不使用R3。

运算器部件

在执行周期，完成读内存指令的数据地址计算与传送操作

STRR [R2], R3

10000011	0010	0011
----------	------	------



动画演示

在存储器读写周期，完成把寄存器堆中的一个寄存器 (R3) 的内容写入存储器的由 AR 指定地址的一个存储单元

写内存指令，在存储器读写周期，完成把寄存器R3的内容写入存储器的选定单元

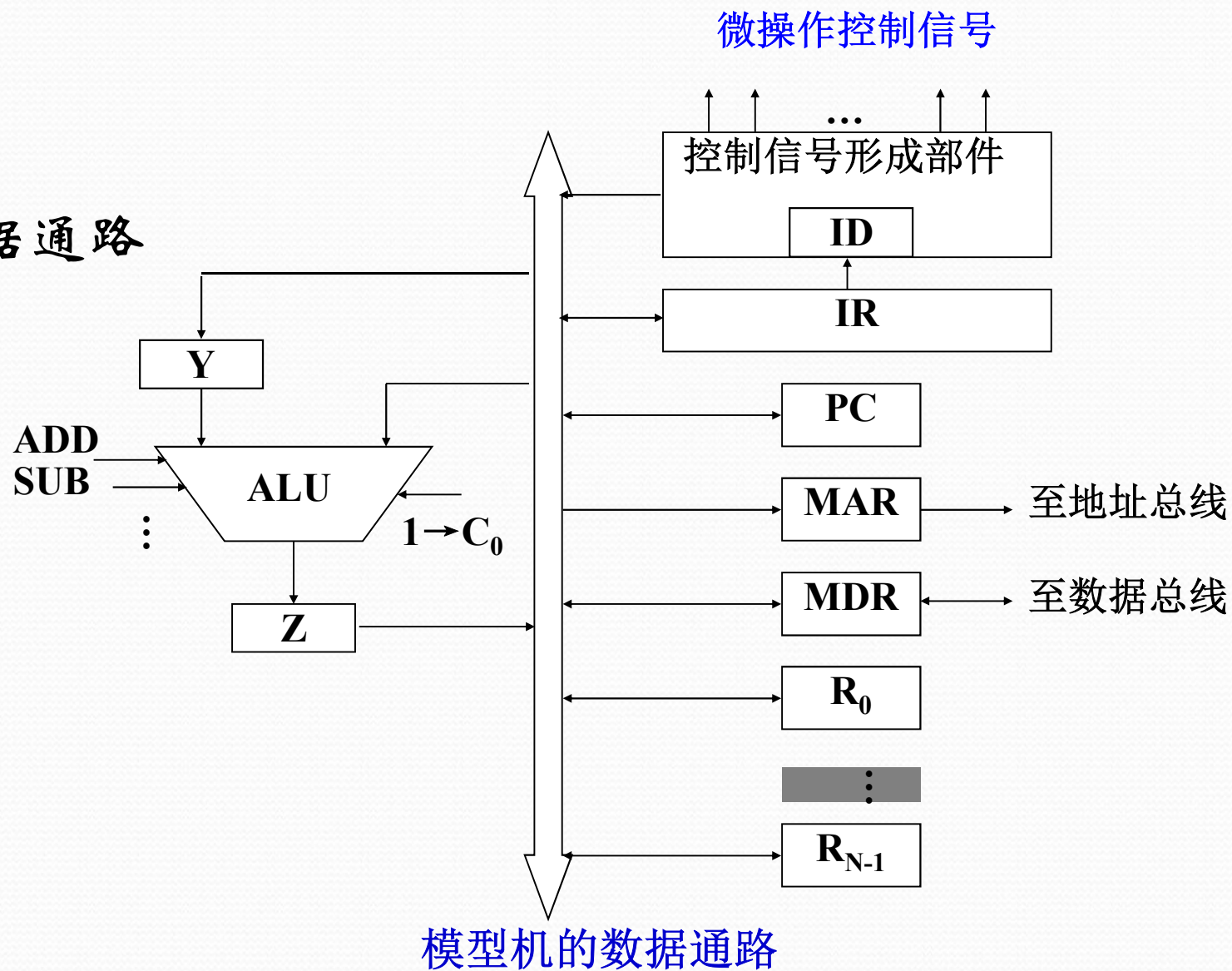
实现功能：Reg[R3] → Mem[AR]，运算器中通过 R3+0 执行数据计算

控制信号：addr_sel=00,s_sel=01,r_sel=00, gate_en=1,gate_dir=1,mio=1,we=1

指令的执行过程举例

例2. 假设某机的数据通路如下图所示。其中**ALU**为运算器，**Y**为其输入端的一个暂存器，**Z**用来保存其输出结果；**PC**为程序计数器；**IR**、**ID**分别为指令寄存器和指令译码器；**MAR**、**MDR**分别为存储地址寄存器和存储数据寄存器；**R₀**至**R_{N-1}**为模型机的**N**个寄存器。

1. 数据通路



2. 加法指令ADD R1, [NUM]

即实现: $R1 \leftarrow (R1) + (NUM)$

相应的微操作如下:

- (1) PC_{out} 、 MAR_{in} 、READ、在数据没取出的间隙进行 $PC+1(0 \rightarrow Y, 1 \rightarrow C_0, ADD, Z_{in})$;
- (2) Z_{out} 、 PC_{in} 、WMFC(等待存储功能完成);
- (3) MDR_{out} 、 IR_{in} ; 取指令完成, 分析指令(略)
- (4) $IR(D)_{out}$ 、 MAR_{in} 、READ; 取存储器取操作数
- (5) $R1_{out}$ 、 Y_{in} 、WMFC; 将另一个操作数送入ALU一个输入端
- (6) MDR_{out} 、ADD、 Z_{in} ; 执行加法
- (7) Z_{out} 、 $R1_{in}$; 存放结果
- (8) END。

3. 转移指令JZ A

若上次运算结果为0 ($ZF=1$)，就转移，转移地址为A；
若上次运算结果不为0 ($ZF=0$)，就顺序执行下一条指令。
相应的微操作序列如下：

(1) PC_{out} 、 MAR_{in} 、 $READ$ 、在数据没取出的间隙进行

$PC+1$ ($0 \rightarrow Y, 1 \rightarrow C_0, ADD, Z_{in}$)；

(2) Z_{out} 、 PC_{in} 、 $WMFC$ (等待存储功能完成)；

(3) MDR_{out} 、 IR_{in} ；完成取指令，对于某种具体模型机都是这3步

(4) IF $ZF=1$ THEN $IR(D)_{out}$ 、 PC_{in}

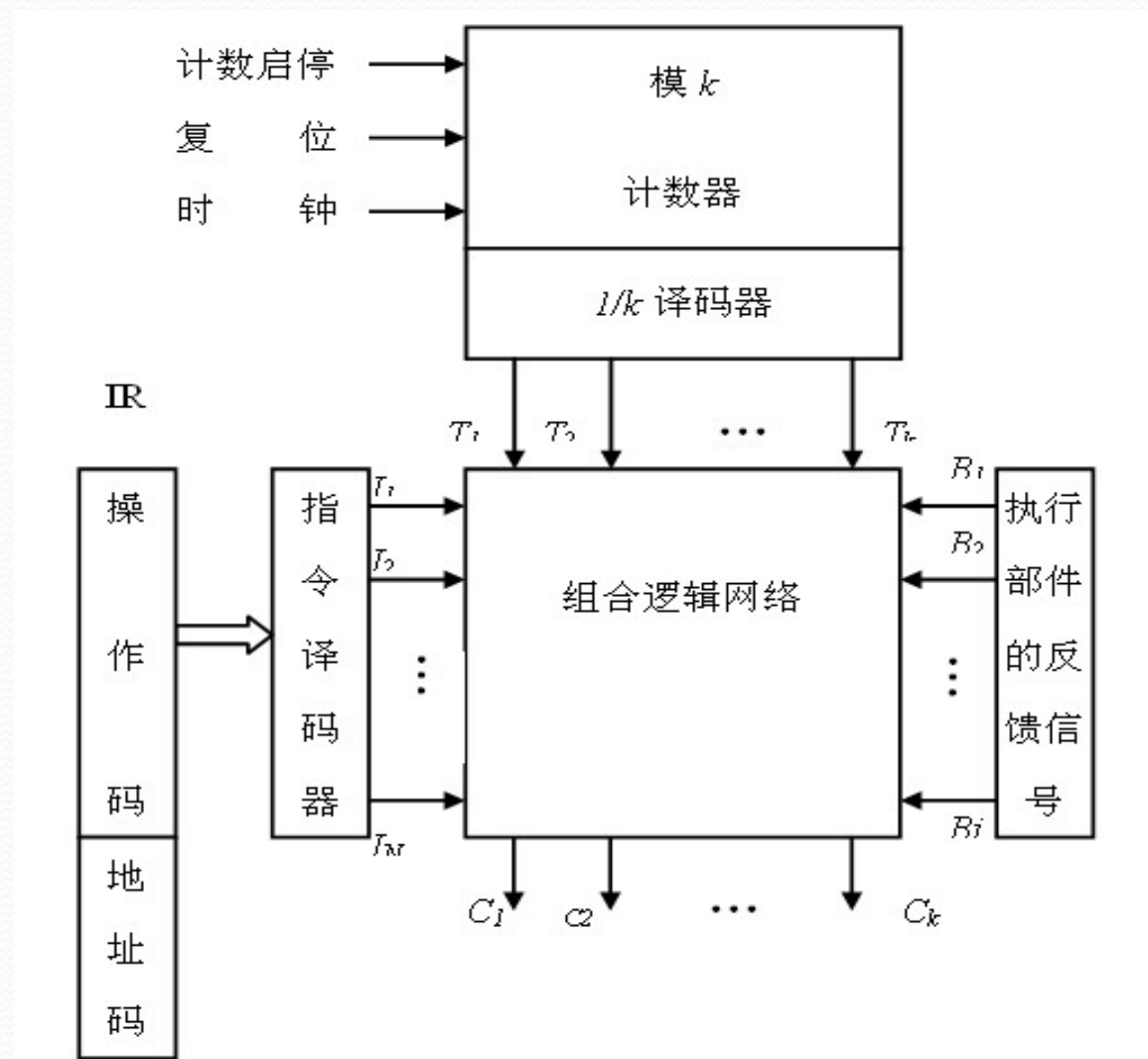
ELSE END；

(5) END

第四章 中央处理器

- 4.1 CPU的基本功能及结构
- 4.2 指令的执行过程
- 4.3 硬布线控制器
- 4.4 微程序控制器
- 4.5 流水线原理
- 4.6 控制器的控制方式

硬布线控制器的组成原理



硬布线控制器的组成原理

逻辑网络的输入信号来源有三个：

- (1) 指令操作码译码器的输出 I_n ；
- (2) 来自时序发生器的节拍电位信号 T_k ；
- (3) 来自执行部件的反馈信号 B_j 。

逻辑网络的输出信号就是微操作控制信号，用来对执行部件进行控制。

显然，硬布线控制器的基本原理可叙述为：某一微操作控制信号 C_m 是指令操作码译码器的输出 I_n 、时序信号（节拍电位信号 T_k ）和状态条件信号 B_j 的逻辑函数。即

$$C_m = f(I_n, T_k, B_j)$$

用这种方法设计控制器，需要根据每条指令的要求，让节拍电位和时序脉冲有步骤地去控制机器的各有关部分，一步一步地依次执行指令所规定的微操作，从而在一个指令周期内完成一条指令所规定的全部操作。

硬布线控制器设计步骤

一般来说，硬布线控制器的设计步骤如下：

1. 绘制指令流程图

以指令为线索，按指令类型分类，将每条指令归纳成若干微操作，然后根据操作的先后次序画出流程图。

2. 安排指令操作时间表

指令流程图的进一步具体化，把每一条指令的微操作序列分配到各个机器周期的各个时序节拍信号上。要求尽量多地安排公共操作，避免出现互斥。

3. 安排微命令表

以微命令为依据，表示在哪个机器周期的哪个节拍有哪些指令要求这些微命令。

硬布线控制器设计步骤

4. 进行微操作逻辑综合

根据微操作时间表，将执行某一微操作的所有条件（哪条指令、哪个机器周期、哪个节拍和脉冲等）都考虑在内，加以分类组合，列出各微操作产生的逻辑表达式，并加以简化。

5. 实现电路

根据上面所得逻辑表达式，用逻辑门电路的组合或PLA电路来实现。

硬布线控制器设计实例

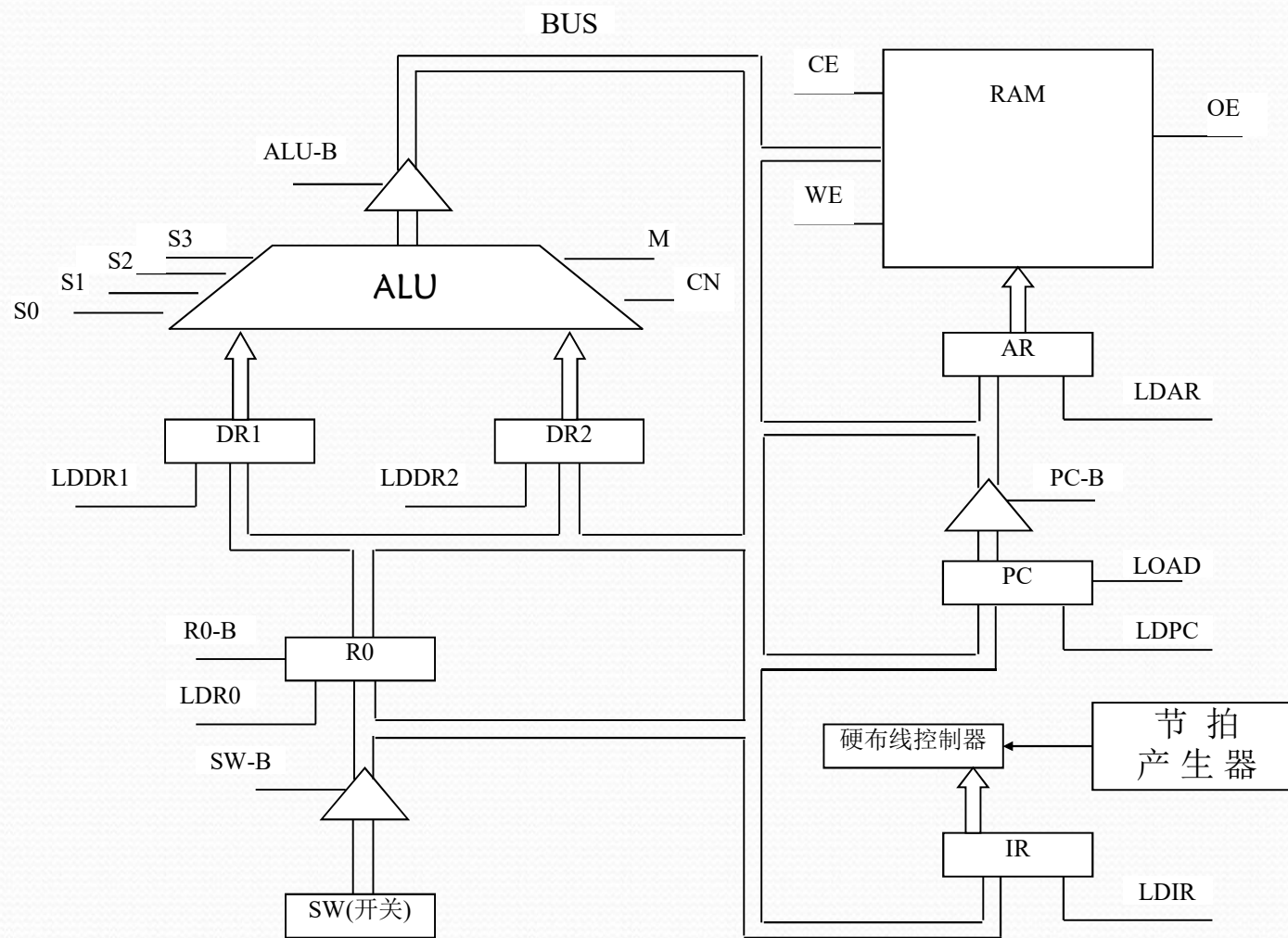
这一部分将以一个简单的模型机为例来讨论硬布线控制器的设计。为了突出重点，减少篇幅，模型机的指令系统仅具有最常见的基本指令和寻址方式，且逻辑结构、时序安排、操作过程安排等方面尽量规整、简单，使初学者比较容易掌握，以帮助大家建立整机概念。分为以下几个部分介绍：

1. 模型机的数据通路
2. 模型机的指令系统
3. 绘制指令流程图
4. 安排指令的操作时间表
5. 安排指令的微命令表
6. 进行微操作信号综合
7. 实现电路

模型机的数据通路

模型机的数据通路如图所示，全机采用单总线结构。ALU是运算器，由74LS181（正逻辑）构成，能完成16种算术运算和16种逻辑运算，M是状态控制端，当M=H时执行逻辑运算，M=L时执行算术运算， $S_3 \sim S_0$ 是运算选择控制端，决定电路执行哪种算术运算或哪种逻辑运算， C_N 是ALU的最低位进位输入，CN=H无进位，CN=L有进位； DR_1 和 DR_2 是ALU输入端的两个暂存器，均由74273构成，LDDR1、LDDR2分别为 DR_1 、 DR_2 的装数信号；R0是一个寄存器，由74374构成，LDR0是装数信号，R0-B是数据输出信号；RAM的地址由地址寄存器AR提供，LDAR是AR的装数信号；输入设备为开关SW，SW-B是控制开关状态输出的三态门的选通信号；输出设备为总线上所连的二极管指示灯；PC是程序计数器，具有计数功能，LDPC=1，LOAD=1时PC+1，LDPC=1时，PC装数；IR是指令寄存器，LDIR是装指令信号。

模型机的数据通路



模型机的数据通路

模型机的指令系统

模型机的指令系统包括5条指令，由单字长和双字长指令构成，包括输入/输出指令IN、OUT，算术指令ADD，数据传送指令STA和控制转移指令JMP，其指令格式如下：

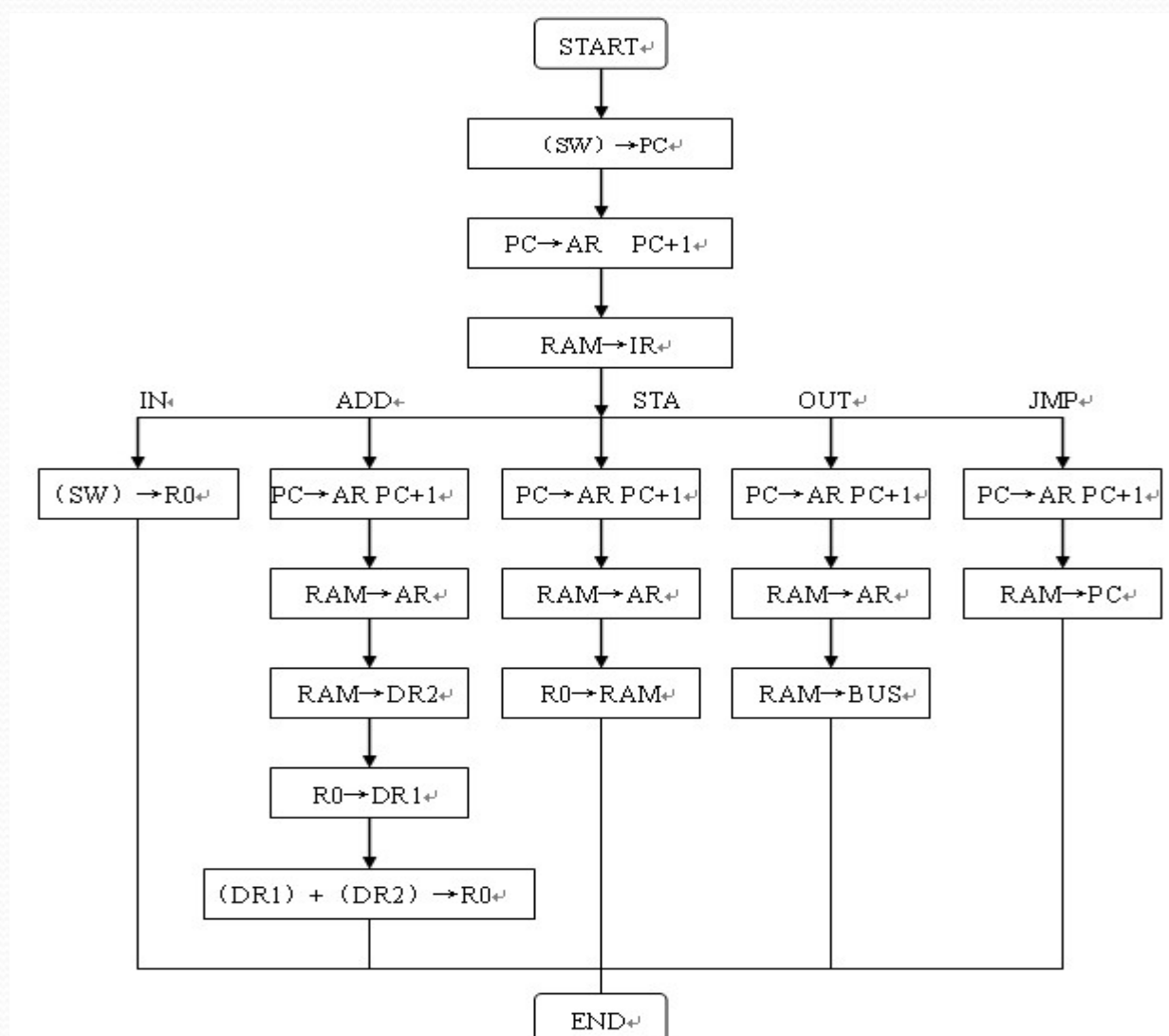
助记符	机器指令码	地址字段	说明
IN	0000 0000		开关状态→R0
ADD addr	0001 0000	×××× ××××	R0+[addr]→R0
STR addr	0010 0000	×××× ××××	R0→[addr]
OUT addr	0011 0000	×××× ××××	[addr]→BUS
JMP addr	0100 0000	×××× ××××	addr→PC

其中IN为单字节，其余为双字节指令，×××× ××××为addr对应的二进制地址码。

绘制指令流程图

根据模型机的**5**条指令所完成的功能及模型机的数据通路，绘制出指令的流程图。假设指令的执行采用统一节拍法，即无论简单还是复杂，都采用**8**个节拍来完成。

指令流程图



安排指令的操作时间表

下表是指令的操作时间表，实际是将指令流程图进一步细化，详细列出每个节拍应做的操作，要求尽量多地安排公共操作。

指令的操作时间表						
时序	公共操作	IN	ADD	STA	OUT	JMP
T0	SW-B、LDPC					
T1	PC-B、LDAR、LDPC、LOAD					
T2	CE、OE、LDIR					
T3	PC-B、LDAR、LDPC、LOAD					
T4	CE、OE、LDAR					
T5			CE、OE、LDDR2	R0-B、CE、WE	CE、OE	
T6			R0-B、LDDR1			
T7		SW-B、LDR0	+, ALU-B、LDR0			CE、OE、LDPC

安排指令的微命令表

指令的微命令表

微命令	T0	T1	T2	T3	T4	T5	T6	T7
SW-B	ALL							IN
LDPC	ALL	ALL		ALL				JMP
LOAD		ALL		ALL				JMP
LDAR		ALL		ALL	ALL			
CE			ALL		ALL	ADD+STA+OUT		JMP
OE			ALL		ALL	ADD+OUT		JMP
WE						STA		
LDIR			ALL					
PC-B		ALL		ALL				
LDDR1							ADD	
LDDR2						ADD		
LDR0								IN+ADD
R0-B						STA	ADD	
+								ADD
ALU-B								ADD

进行微操作信号综合

在画出全部指令的流程图、微操作时间表和微命令表后，即可对它们进行综合分析、归纳，根据微操作时间表可以写出各微操作控制信号的**逻辑表达式**。表达式一般包括下列因素：

微操作控制信号=机器周期×节拍×脉冲×操作码×机器状态条件

由于本模型机的指令系统比较简单，且指令的执行只以节拍为依据，所以微操作控制信号的逻辑表达式比较简单，下面根据微命令表列出了模型机的部分微操作信号的逻辑表达式。如：

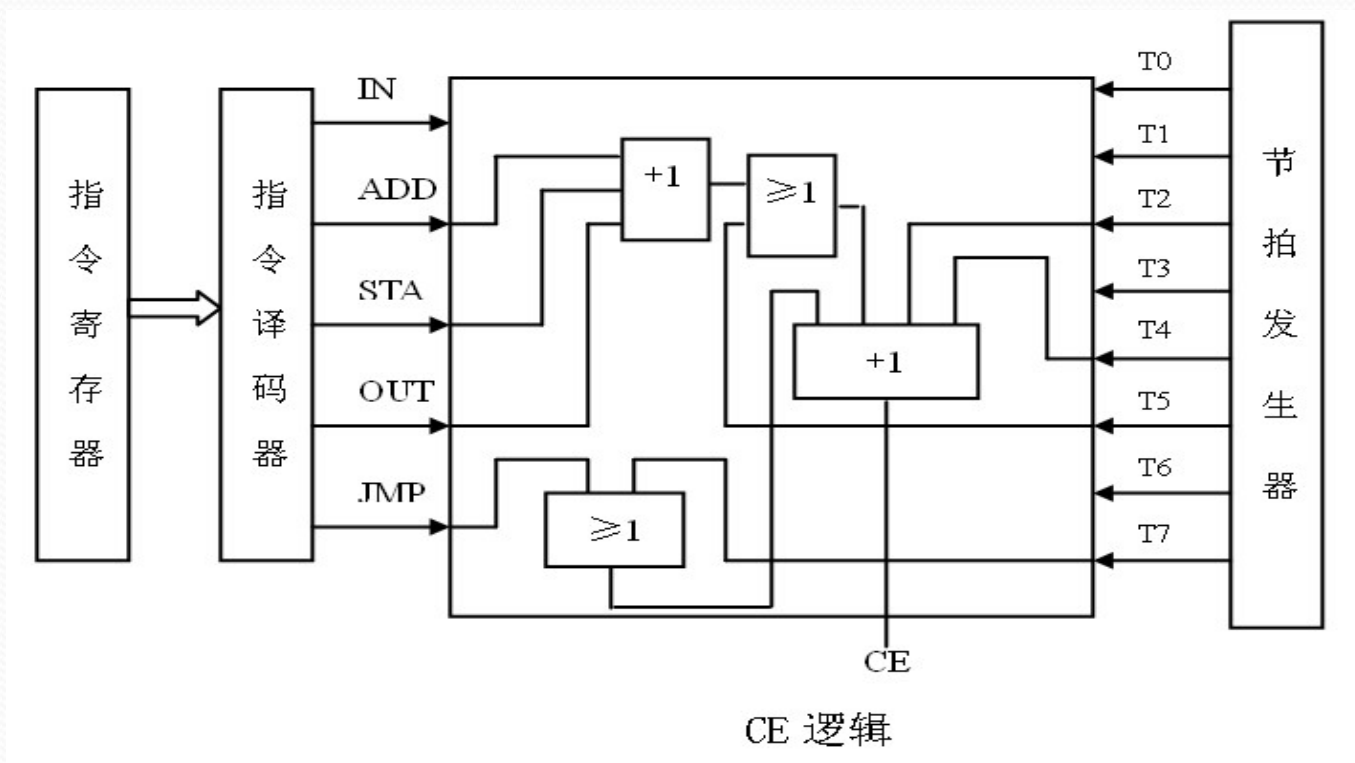
$$SW-B=T_0+T_7 \cdot IN$$

$$LDPC=T_0+T_1+T_3+T_7 \cdot JMP$$

$$CE=T_2+T_4+T_5 \cdot (ADD+STA+OUT) +T_7 \cdot JMP$$

实现电路

根据上述列出的各微操作控制信号的逻辑表达式，可画出相应的逻辑电路图，并用逻辑门电路实现之。例如，**CE**逻辑如图所示。



第四章 中央处理器

- 4.1 CPU的基本功能及结构
- 4.2 指令的执行过程
- 4.3 硬布线控制器
- 4.4 微程序控制器
- 4.5 流水线原理
- 4.6 控制器的控制方式

微程序控制器的基本原理

微程序控制器的设计思想是由英国剑桥大学的教授威尔克斯(Wilkes)于1951年首先提出来的，即将机器指令的操作(从取指令到执行)分解成若干个更基本的微操作序列，并将有关控制信号(微命令)按照一定的格式编成微指令，存放到一个只读存储器中，当机器运行时，一条一条地读出这些微指令，从而产生全机所需要的各种操作控制信号，使相应部件执行所规定的操作。

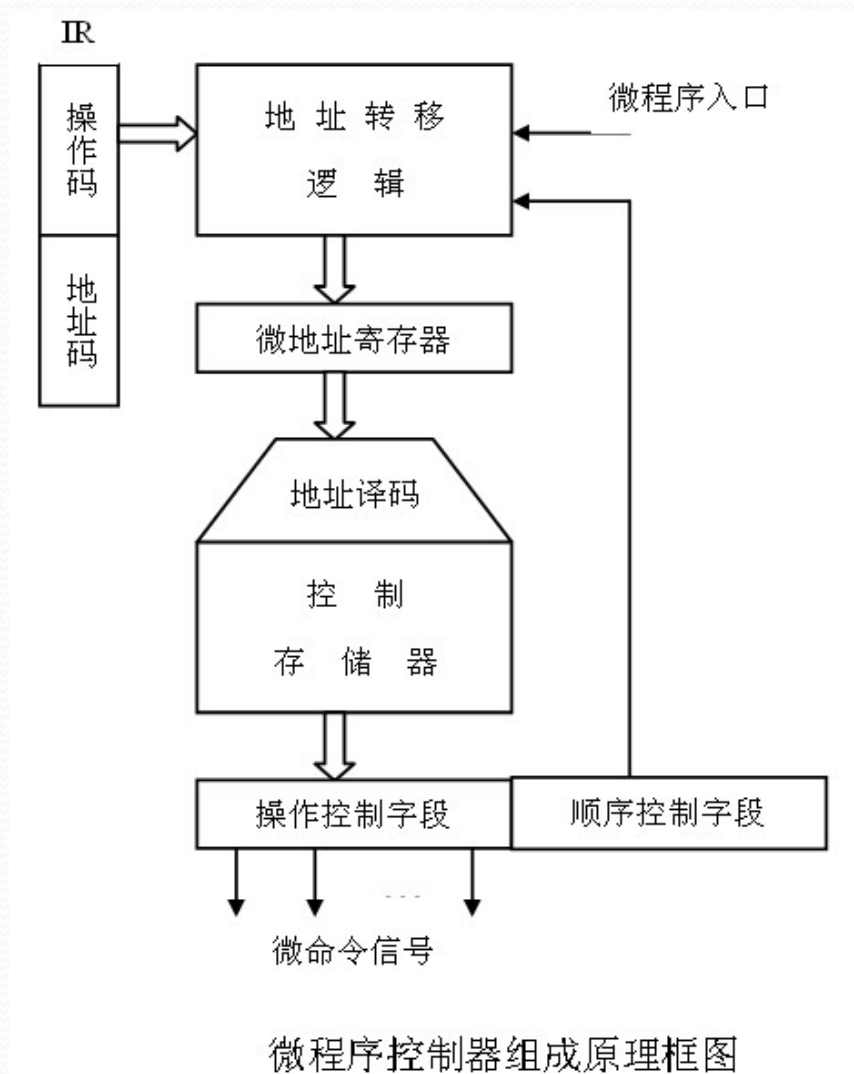
微程序控制器同硬布线控制器相比较，具有规整性、灵活性、可维护性等优点，因而在计算机设计中逐渐取代了早期采用的硬布线控制器，并已被广泛地应用。在计算机系统中，微程序设计技术是利用软件方法来设计硬件的一门技术。

有关的术语和概念

- (1) **微命令**：构成控制信号序列的最小单位。
- (2) **微操作**：由微命令控制实现的最基本的操作。
- (3) **微指令**：一组实现一定操作功能的用二进制编码表示的微命令的组合。
- (4) **微周期**：从控制存储器读取一条微指令并执行相应的微操作所需的时间。
- (5) **微程序**：一系列微指令的有序集合。

组成原理框图

微程序控制器原理框图如图所示。它主要由控制存储器、微地址寄存器、微命令寄存器和地址转移逻辑几部分组成。



组成原理框图

(1) 控制存储器

控制存储器用来实现整个指令系统的所有微程序。一般计算机的指令系统是固定的，所以实现指令系统的微程序也是固定的，因此控制存储器通常由高速半导体只读存储器构成，其存储容量视机器指令系统而定，即取决于微程序的数量，其字长就是微指令字的长度。

(2) 微指令寄存器

微指令寄存器用来存放从控制存储器读出的当前微指令。微指令中包含两个字段：微操作控制字段和微地址字段，微操作控制字段将操作控制信号送到控制信号线上，并提供判别测试字段，微地址字段用于控制下一条微指令地址的形成。

组成原理框图

(3) 微地址寄存器

存放将要访问的下一条微指令的地址。

(4) 地址转移逻辑

地址转移逻辑用来形成即将要执行的微指令的地址，形成方式一般有以下几种：取指令公共操作所对应的微程序一般从控制存储器的0号单元开始存放，所以微程序的入口地址0是由硬件强制规定的；当微程序不出现分支，则微指令从控制存储器读出后直接给出下一条微指令的地址；当微程序出现分支时，通过判别测试字段、微地址字段和执行部件的反馈信息形成后继微地址，包括根据操作码转移的情况。

执行过程

(1) 从控制存储器中逐条取出“取机器指令”用的微指令，执行取指令公共操作，执行完后，从主存中取出的机器指令就已存入指令寄存器中了。一般取指令微程序的入口地址为控制存储器的0号单元；

(2) 根据指令寄存器中的操作码，经过微地址形成部件，得到这条指令对应的微程序入口地址，并送入微地址寄存器；

(3) 从控制存储器中逐条取出对应的微指令并执行之；

(4) 执行完对应于一条机器指令的一段微程序后又回到取指微程序的入口地址，继续第(1)步，以完成取下一条机器指令的公共操作

微指令设计的技术问题

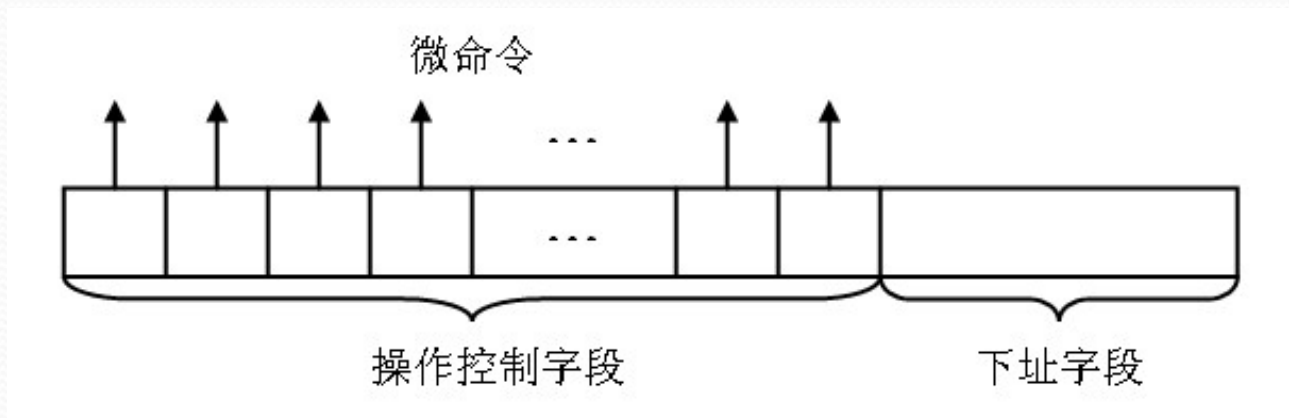
微指令的结构、微程序的顺序控制方式及微指令的执行方式直接影响微程序控制器的结构和控制过程，它们是微程序控制器设计要解决的关键技术问题。

微指令可以分为操作控制字段和下址字段两大部分。这里所说的微指令编码法就是操作控制字段的编码法，通常有几种方法。

- 1) 直接控制法
- 2) 最短字长编码
- 3) 分段直接编码
- 4) 分段间接编码

直接控制法

在微指令的操作控制字段中，**每一个微命令都用一位信息表示**。是否发出某个微命令，只要将控制字段中表示该命令的相应位设置成“1”或“0”，就可打开或关闭某个控制门，因此，微命令的产生不必经过译码，可从微操作控制字段直接得到，故有时称为不译码法。

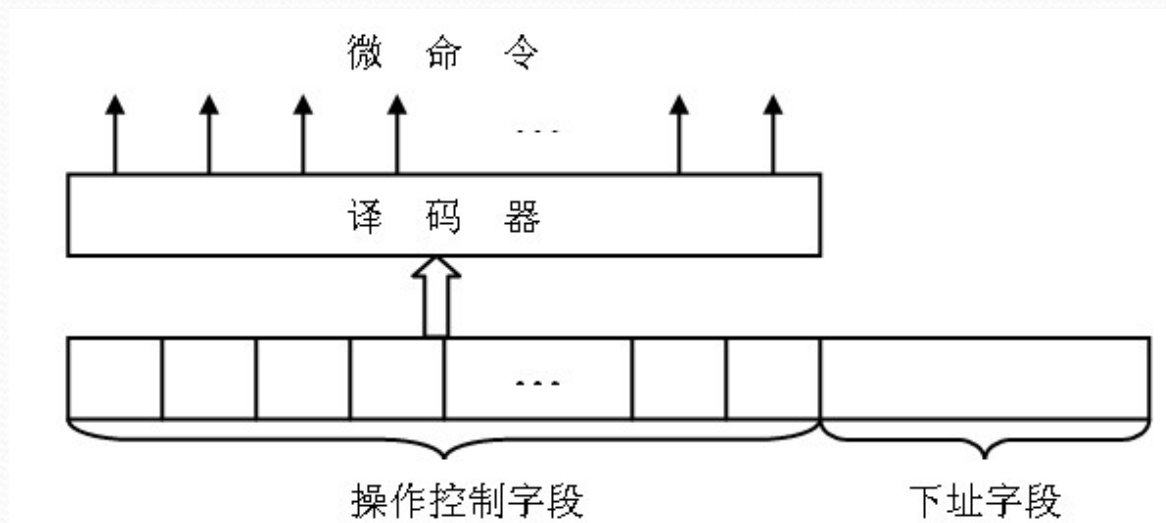


优点：控制简单、直观，操作并行性最好，从而可以提高速度。

缺点：微指令字太长，控制存储器的容量过大且微指令字利用效率很低。因此这种编码方法只适用于结构简单或速度要求很高的高速数字控制部件。

最短字长编码

将所有的微命令进行统一的二进制编码，用不同的码点去表示不同的微命令，通过译码器产生微操作控制信号，如图所示。

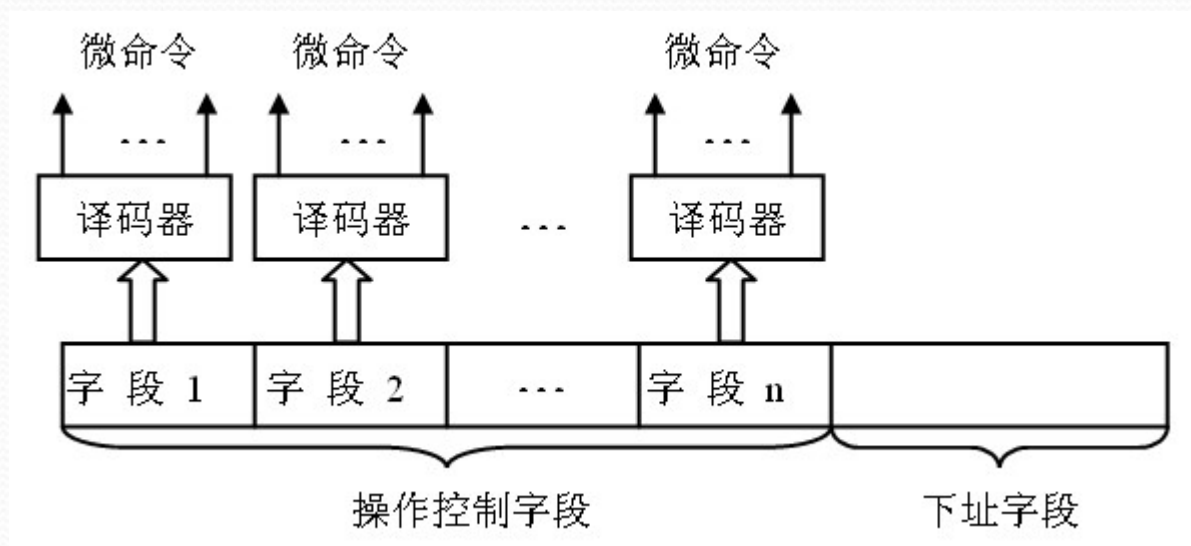


优点：微指令字长很短

缺点：因为它每次只能产生一个微命令，所有微命令均不能够并行，难以提高微命令的执行效率，故在实际应用中很少采用。

分段直接编码

将微操作控制字段划分为若干个小字段，每个小字段独立译码，每个码点表示一个微命令，其微指令结构如图所示。

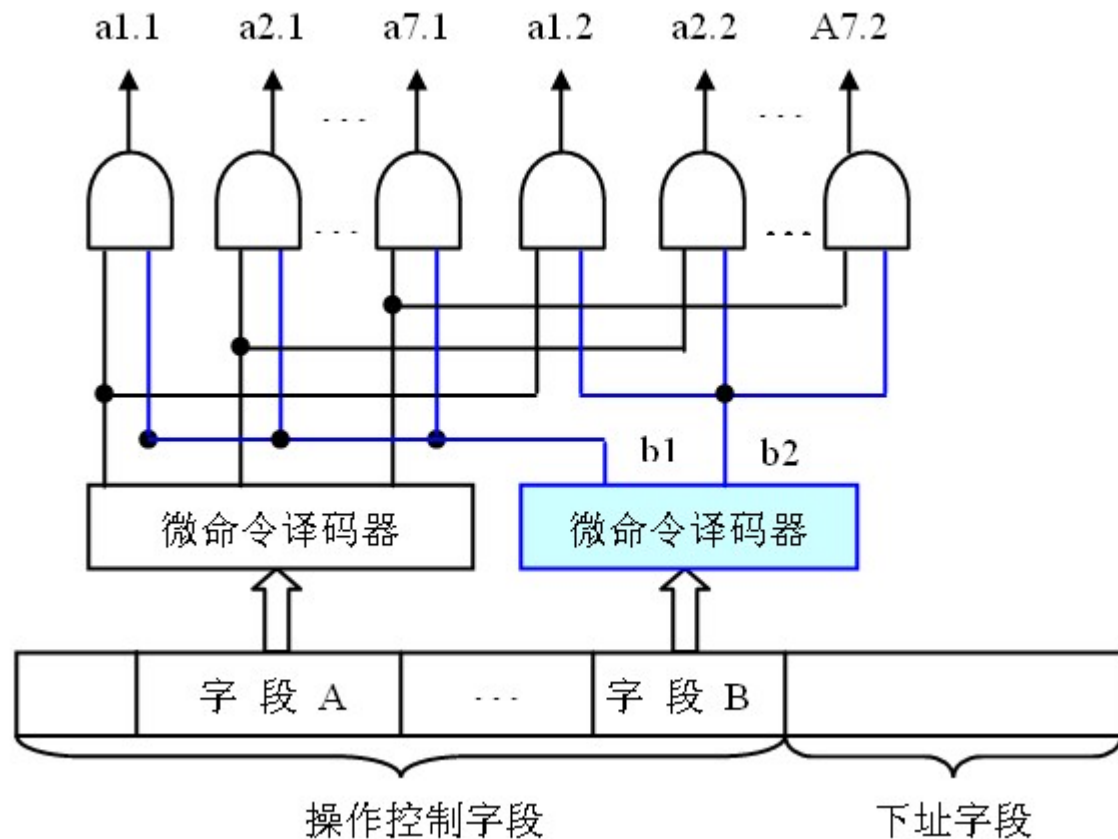


特点：吸收了直接控制编码和最短字长编码两种方法的优点，既能缩短微指令字长，又有较高的并行性，执行速度比较快，因此得到了广泛的应用。

分段间接编码

分段间接编码是在分段直接编码的基础上，进一步缩短微指令字长的一种编码方法。在这种编码方法中，某些参与编码的微指令**不是由一个控制字段直接定义**，而**需要两个或两个以上的控制字段来定义**。

或者说，一个控制字段的微操作需要另外一个控制字段来解释才能确定。这种方法进一步减少了微指令的长度，但通常可能会削弱微指令的并行控制能力，且译码电路相应地较复杂，因此，它只作为字段直接编码法的一种补充。



微指令格式

微指令格式的设计是微程序控制器设计的主要工作，它直接影响微程序控制器的结构和微程序的编制，也直接影响计算机的速度和控制存储器的容量。不同机器有不同的微指令格式，就其共性来说，大致可归纳为两大类，即水平型微指令和垂直型微指令。

(1) 水平型微指令

一次能定义多个微命令（控制执行多个微操作）的微指令。一般来说，它具有如下特征：

- 微指令字较长（一般为几十位到100位左右，有的长达200多位），增加了控制存储器的横向容量。
- 微指令中的微操作有高度的并行性，因而能充分发挥数据通路的并行操作能力。
- 微指令和机器指令的差别很大，设计者只有熟悉了数据通路，才有可能编制出理想的微程序，一般用户不易掌握。

微指令格式

(2)垂直型微指令

一次只能执行一个微命令的微指令。其特征是：

微指令字短，一般为10~20位左右。

微指令的并行操作能力有限，一般只能实现一个微操作，控制一两个信息传送通路，效率低，执行一条机器指令所需的微指令数目多，执行时间长。

微指令与机器指令很相似，所以容易掌握和利用，编程比较简单，不必过多地了解数据通路的细节。

(3)混合型微指令

从上面的讨论可以看出，水平型微指令和垂直型微指令各有其优缺点。实际使用中，常常兼顾两者的优缺点，设计出一种混合型微指令，采用不太长的字长，又具有一定的并行控制能力，可高效地去实现机器的指令系统。

微地址的形成方式

从两个方面讨论后继微地址的确定方法。一是如何产生每条机器指令所对应的微程序的入口地址，另一个是在微程序内如何产生后继微地址。

(1) 微程序入口地址的确定

每一种机器指令对应着一段微程序。当“取指令”微程序把一条机器指令取到指令寄存器后，就要根据其操作码经地址转移逻辑形成微程序入口地址，地址转移逻辑通常有以下两种：

当操作码的位数与位置固定时，可直接将操作码与微地址码的部分对应。这时，不需要专门的硬件，只需要用连线将对应位直接相连即可。若微地址码长度大于操作码长度，则对应位以下的低位一般填0，对应位以上的高位可以固定填某个二进制常数。

在操作码的位数或位置不固定的情况下，需用专门的硬件来实现操作码到微地址的变换。通常是以查表的方法来实现，即用ROM实现一个表格，该表格中存放有每个操作码所对应的微程序入口地址。

微地址的形成方式

(2)后继微地址的产生

每条微指令执行后都必须根据要求产生后继微指令的地址，一种常用的产生后继微地址的方法是通过下址字段直接给出后继微地址，这种方法也称为**断定方式**。此外，后继微地址的产生主要有两种方式：

a) 计数器方式

这种方式和机器指令的控制方式很相似，它也有顺序执行和转移执行之分。顺序执行时，后继微地址就是现行微地址加上一个增量（通常为“1”），转移时，由微指令的下址字段产生转移微地址。

优点：简单、易于掌握，编制微程序容易；

缺点：这种方式不能实现两路以上的并行微程序转移，不利于提高微程序的执行速度。

微指令的形成方式

b) 多路转移方式

一条微指令具有多个转移分支的能力称为多路转移，这种方式的后继微地址可由微程序设计者指定，或者根据微指令所规定的测试结果直接决定后继微地址的全部或部分值。

这种方式的微地址一般分为两部分，一部分为非测试地址，可由程序设计者直接指定，占高位部分；另一部分为测试地址，即由测试结果确定其地址值，占低位部分。测试地址的位数确定了转移的并行度：1位为2路转移，2位为4路转移，N位为 2^N 路转移。

优点：能以较短的顺序控制字段配合，实现多路并行转移，灵活性好，速度较快；

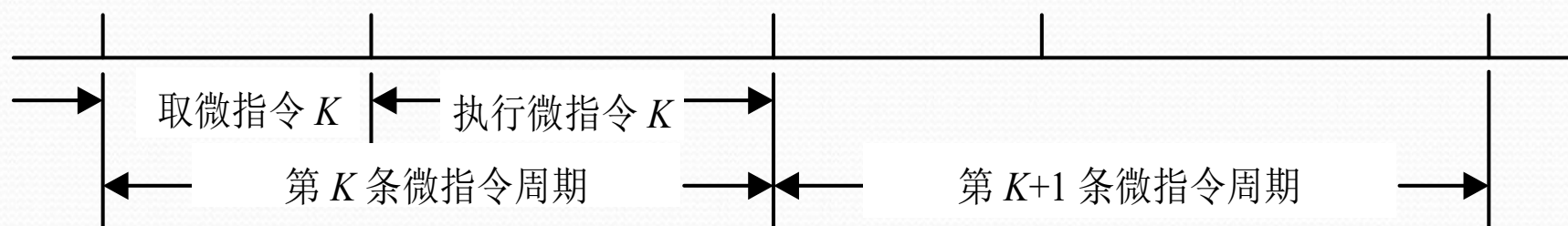
缺点：后继微地址码的生成机构比较复杂。

微指令的执行方式

微指令的执行过程与机器指令的执行过程很类似，任何一条微指令的执行过程均可分为取微指令和执行微指令两个阶段，其执行方式可分为串行和并行两种方式。

(1) 串行方式

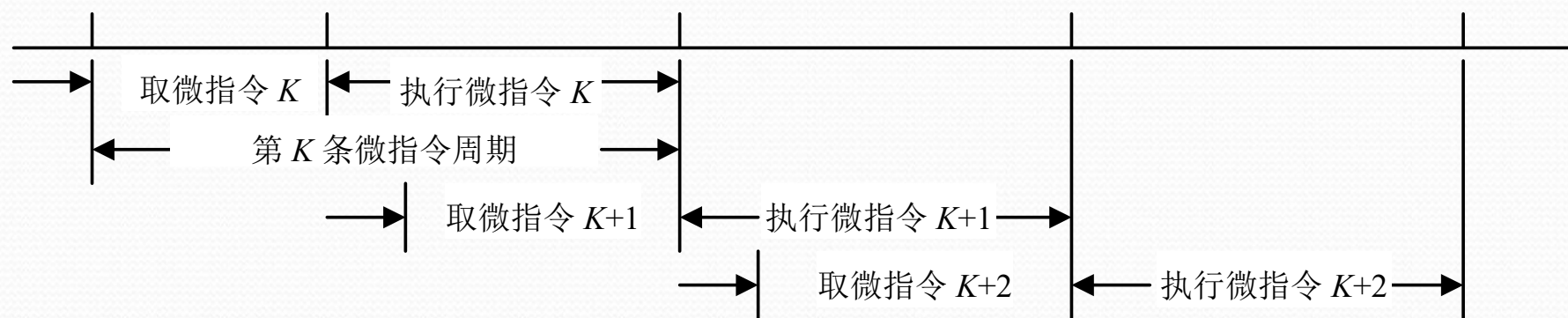
取微指令和执行微指令顺序进行。在这种方式里，取微指令和执行微指令是顺序进行的，在一条微指令取出并执行之后，才能取下一条微指令，微指令周期如下图所示。



微指令的串行执行过程

微指令的执行方式

为了提高微指令的执行速度，可以将取微指令和执行微指令的操作重叠起来，从而缩短微周期。因为这两个操作是在两个完全不同的部件中执行的，所以完全可在执行第K条微指令的同时去取第K+1条微指令，如图所示。



微指令的并行执行过程

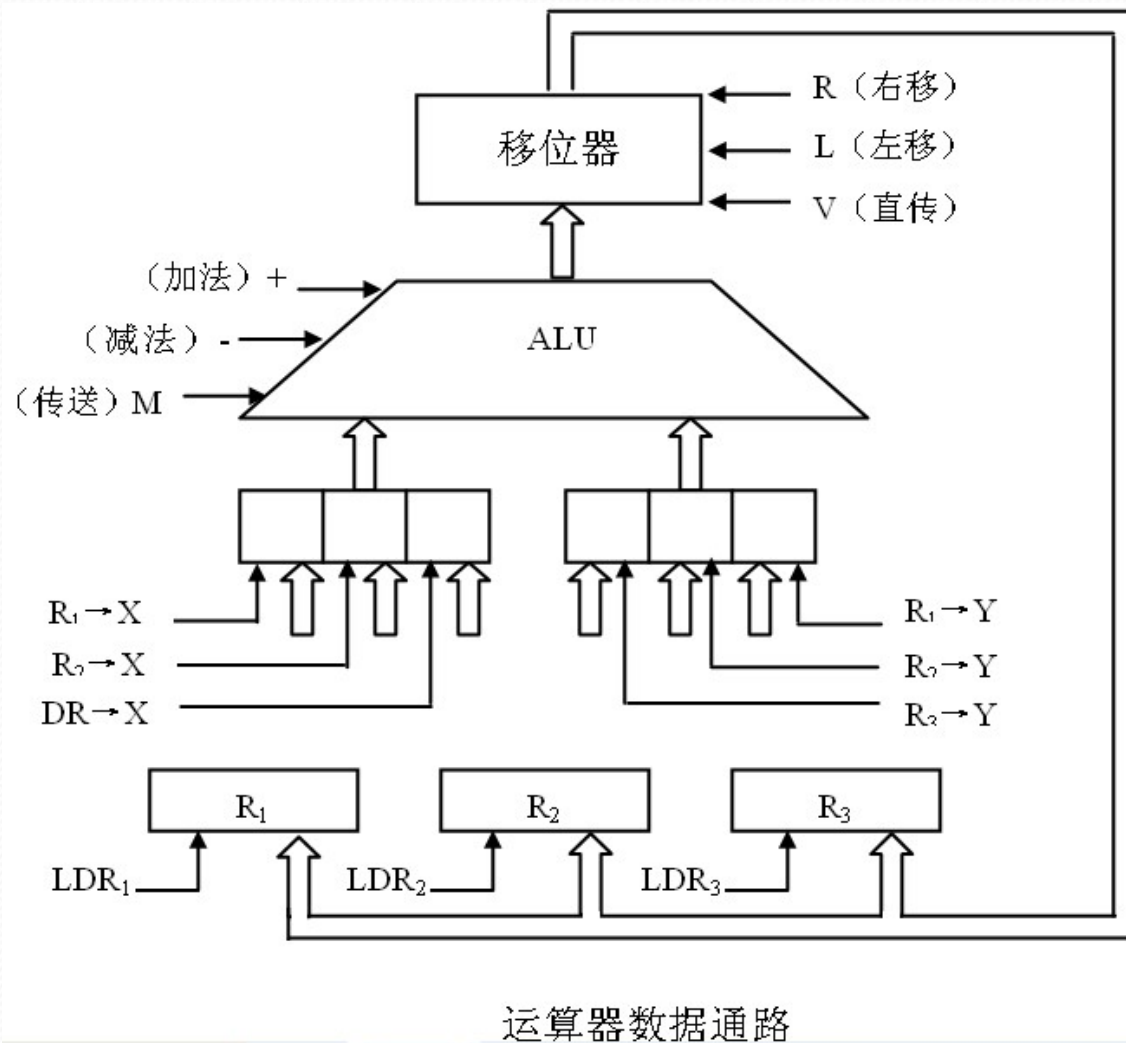
优点：缩短了微指令周期，但是为了不影响本条微指令的正确执行，需要增加一个微指令寄存器，用以暂存下一条微指令。

微指令格式设计举例

设某运算器数据通路如图所示。

(1)指出相容性和相斥性的微操作

(2)设计适合此运算器的微指令格式 (要求微指令字长17位, 其中下址字段6位)。



微指令格式设计举例

解：

相斥性的微操作有：

- (i) 移位器的三个微操作R、L、V；
- (ii) ALU的三个微操作+、-、M；
- (iii) X输入端的三个微操作 $R1 \rightarrow X$ 、 $R2 \rightarrow X$ 、 $DR \rightarrow X$ ；Y输入端的三个微操作 $R1 \rightarrow Y$ 、 $R2 \rightarrow Y$ 、 $R3 \rightarrow Y$ ；

相容性的微操作有：

- (i) LDR1、LDR2、LDR3相容；
- (ii) X输入端的任一微操作与Y输入端的任一微操作相容。

微指令格式设计举例

(2) 根据题意，微指令字长为17位，微指令字中下址字段为6位，则应采用11位来实现数据通路中的15个微命令信号。显然，只用直接控制法无法完成，因此我们采用直接控制与字段直接译码法相结合的方法来完成。

微指令编码的一般规则：

a) 使直接编码占用的操作数量同译码方法占用的操作数量基本平衡

b) 将容易引起互斥的微命令分在同一段，采用译码的方法进行
因此，在对题目中的微命令的操作码控制字段进行编码时就要考虑这些问题。

对下址字段的编码方法是任意的，本题不作考虑。

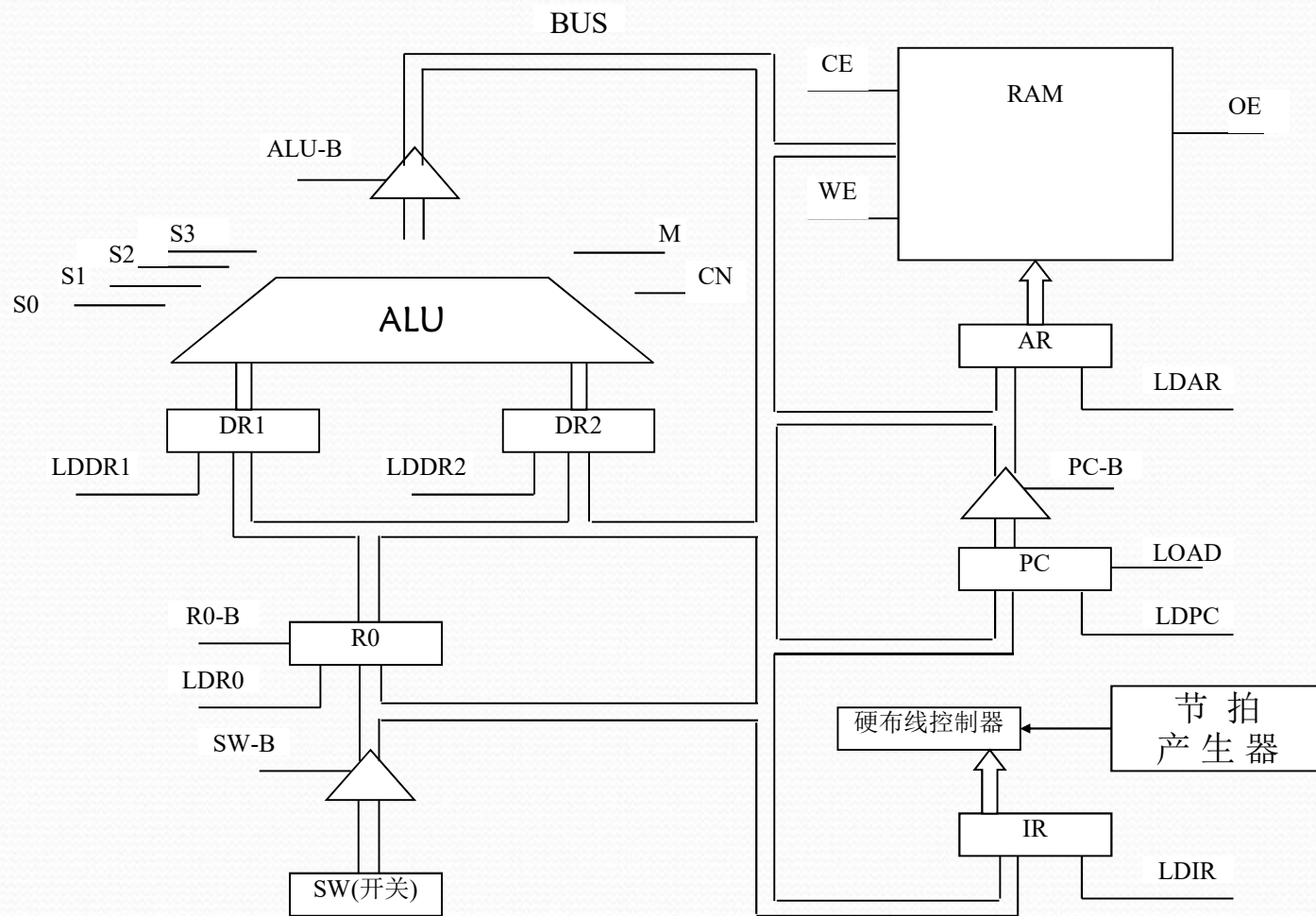
微指令格式设计举例

字段译码法要求将相斥的微命令放在一个字段中。因此可分别将（R、L、V）、（+、-、M）、（R1→X、R2→X、DR→X）、（R1→Y、R2→Y、R3→Y）放在一个字段中译码，每个字段长2位，译码产生4个输出；LDR1、LDR2、LDR3各占1位，用直接表示法表示。据此，设计的微指令格式如下表：

运算器的微指令格式							
微操作控制字段							下址字段
2 位	2 位	2 位	2 位	1 位	1 位	1 位	6 位
00 无	00 无	00 无	00 无	0 无	0 无	0 无	*****
01 R	01 +	01 R ₁ →X	01 R ₁ →Y	1 LDR ₁	1 LDR ₂	1 LDR ₃	*****
10 L	10 -	10 R ₂ →X	10 R ₂ →Y				*****
11 V	11 M	11 DR→X	11 R ₃ →Y				*****

微程序控制器的设计实例

模型机的数据通路



模型机的数据通路

模型机的指令系统

模型机的指令系统包括5条指令，包括输入/输出指令IN和OUT，算术指令ADD，数据传送指令STA和控制转移指令JMP，其指令格式如下：

助记符	机器指令码	地址字段	说明
IN	0000 0000		开关状态→R0
ADD addr	0001 0000	×××× ××××	R0+[addr]→R0
STR addr	0010 0000	×××× ××××	R0→[addr]
OUT addr	0011 0000	×××× ××××	[addr]→BUS
JMP addr	0100 0000	×××× ××××	addr→PC

其中IN为单字节，其余为双字节指令，×××× ××××为addr对应的二进制地址码。

绘制微程序流程图

根据模型机的指令系统和数据通路，绘制出微程序流程图[如图所示](#)，当拟定“取指”微指令时，该微指令的判别测试字段为P(1)测试，并根据P(1)的测试结果出现多路分支，由于操作码的位数固定，所以可直接将操作码与微地址码的部分对应。本模型机用指令寄存器（ $IR_7 \sim IR_0$ ）的3位（ $IR_6 \sim IR_4$ ）与微地址码的后3位对应，而微地址码的高三位固定为001，据此，出现了5路分支，占用5个固定的微地址单元。

设计微指令格式

根据模型机的数据通路和控制存储器的要求，设计出微指令格式。

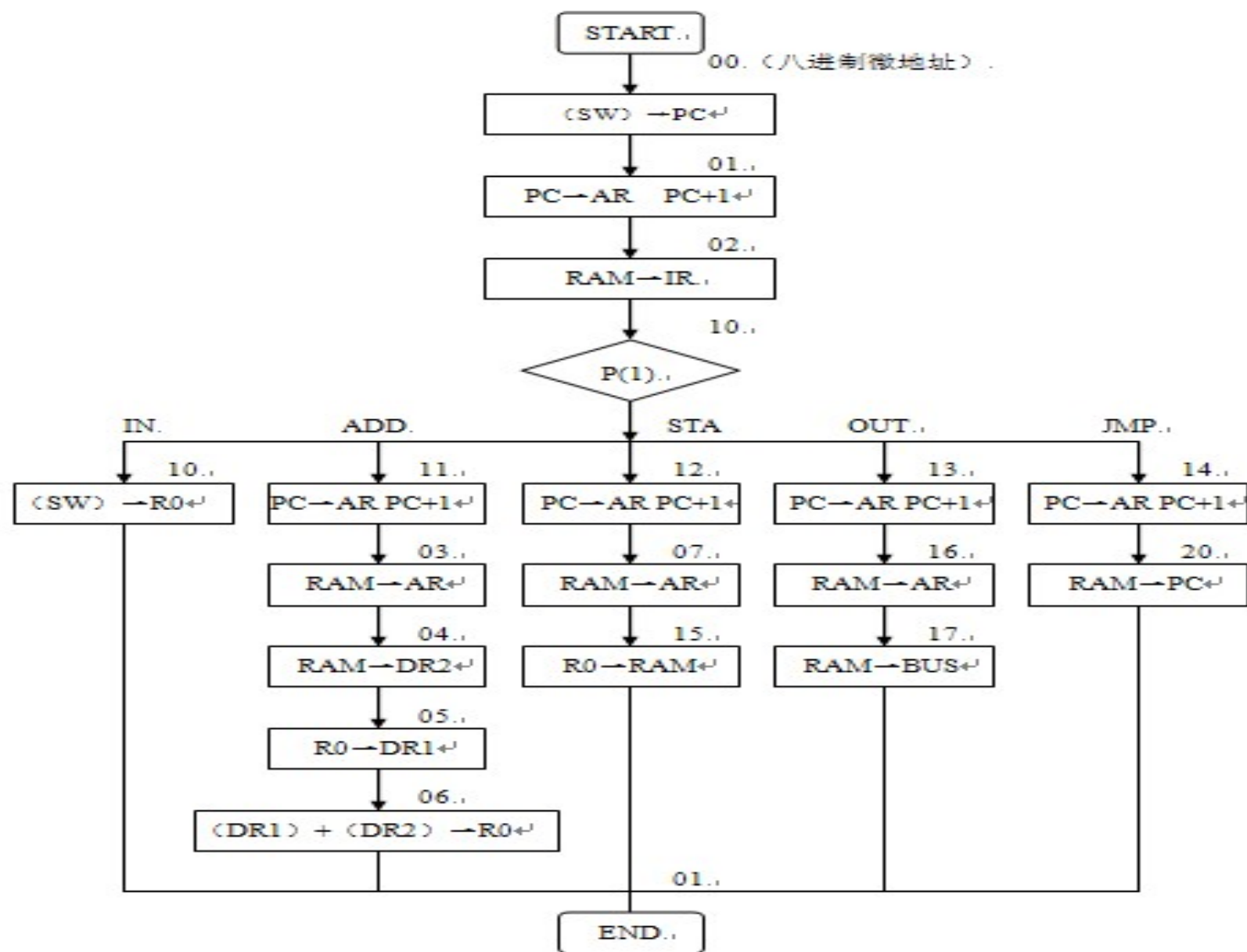
微指令格式

24	23	22	21	20	19	18	17	16	15	14 13 12	11 10 9	8 7	6	5	4	3	2	1
S3	S2	S1	S0	M	CN	CE	WE	OE	LDPC	A	B	C	uA5~uA0					

A 字段			
0	0	0	
0	0	1	LDR0
0	1	0	LDDR1
0	1	1	LDDR2
1	0	0	LDIR
1	0	1	LOAD
1	1	0	LDAR

B 字段			
0	0	0	
0	0	1	R0-B
0	1	0	ALU-B
0	1	1	SW-B
1	0	0	PC-B

C 字段		
0	0	
0	1	P (1)



微程序代码表

微地址	24	23	22	21	20	19	18	17	16	15	14~12	11~9	8 7	6~1
	S3	S2	S1	S0	M	CN	CE	WE	OE	LDPC	A	B	C	uA5~uA0
00	0	0	0	0	0	0	0	0	0	1	101	011	00	000001
01	0	0	0	0	0	0	0	0	0	1	110	100	00	000010
02	0	0	0	0	0	0	1	0	1	0	100	000	01	100000
03	0	0	0	0	0	0	1	0	1	0	110	000	00	000100
04	0	0	0	0	0	0	1	0	1	0	011	000	00	000101
05	0	0	0	0	0	0	0	0	0	0	010	001	00	000110
06	1	0	0	1	0	1	0	0	0	0	001	010	00	000001
07	0	0	0	0	0	0	1	0	1	0	110	000	00	001101
10	0	0	0	0	0	0	0	0	0	0	001	010	00	000001
11	0	0	0	0	0	0	0	0	0	1	110	100	00	000011
12	0	0	0	0	0	0	0	0	0	1	110	100	00	000111
13	0	0	0	0	0	0	0	0	0	1	110	100	00	001110
14	0	0	0	0	0	0	0	0	0	1	110	100	00	010000
15	0	0	0	0	0	0	1	1	0	0	000	001	00	000001
16	0	0	0	0	0	0	1	0	1	0	110	000	00	001111
17	0	0	0	0	0	0	1	0	1	0	000	000	00	000001
20	0	0	0	0	0	0	1	0	1	1	101	000	00	000001

第四章 中央处理器

- 4.1 CPU的基本功能及结构
- 4.2 指令的执行过程
- 4.3 硬布线控制器
- 4.4 微程序控制器
- 4.5 流水线原理
- 4.6 控制器的控制方式

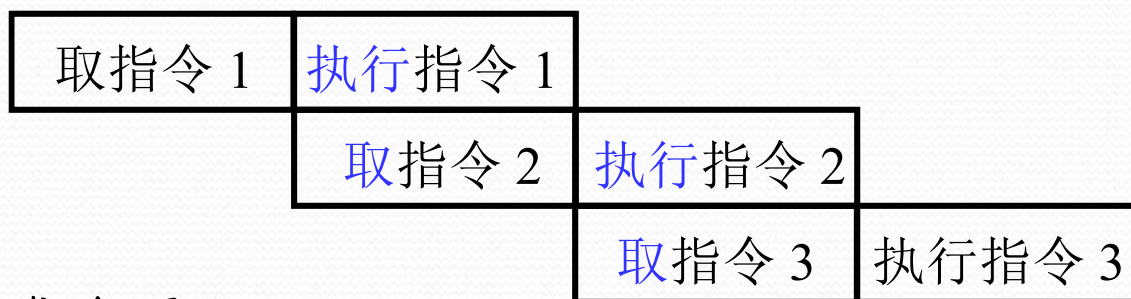
指令的串行执行与重叠执行

1. 指令的串行执行



取指令 取指令部件 完成 总有一个部件 空闲
执行指令 执行指令部件 完成

2. 指令的二级流水



指令预取

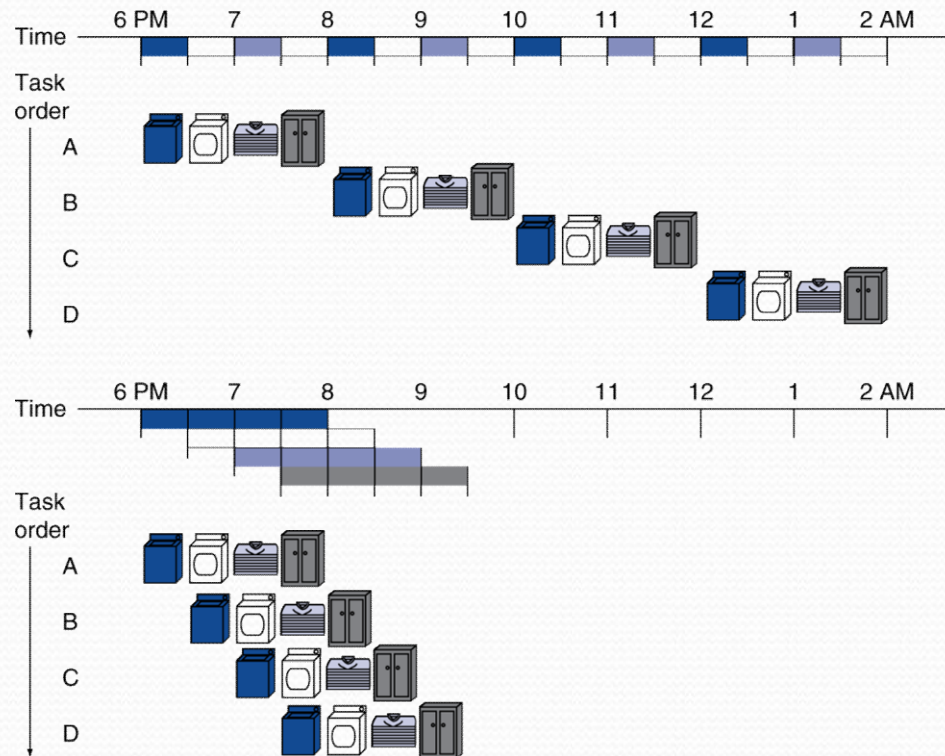
若 取指 和 执行 阶段时间上 完全重叠

指令周期 减半 速度提高 1 倍

Pipelining Analogy

Pipelined laundry: overlapping execution

➤ Parallelism improves performance



■ Four loads:

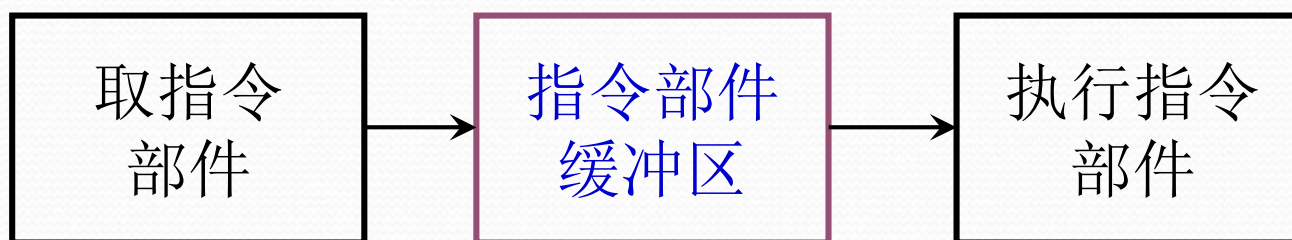
■ Speedup
 $= 8 / 3.5 = 2.3$

■ Non-stop:

■ Speedup
 $= 2n / (0.5n + 1.5) \approx 4$
 $= \text{number of stages}$

影响指令流水效率的因素

(1) 执行时间 > 取指时间



(2) 条件转移指令 对指令流水的影响

必须等 上条 指令执行结束，才能确定 下条 指令的地址，
造成时间损失

解决办法 ？

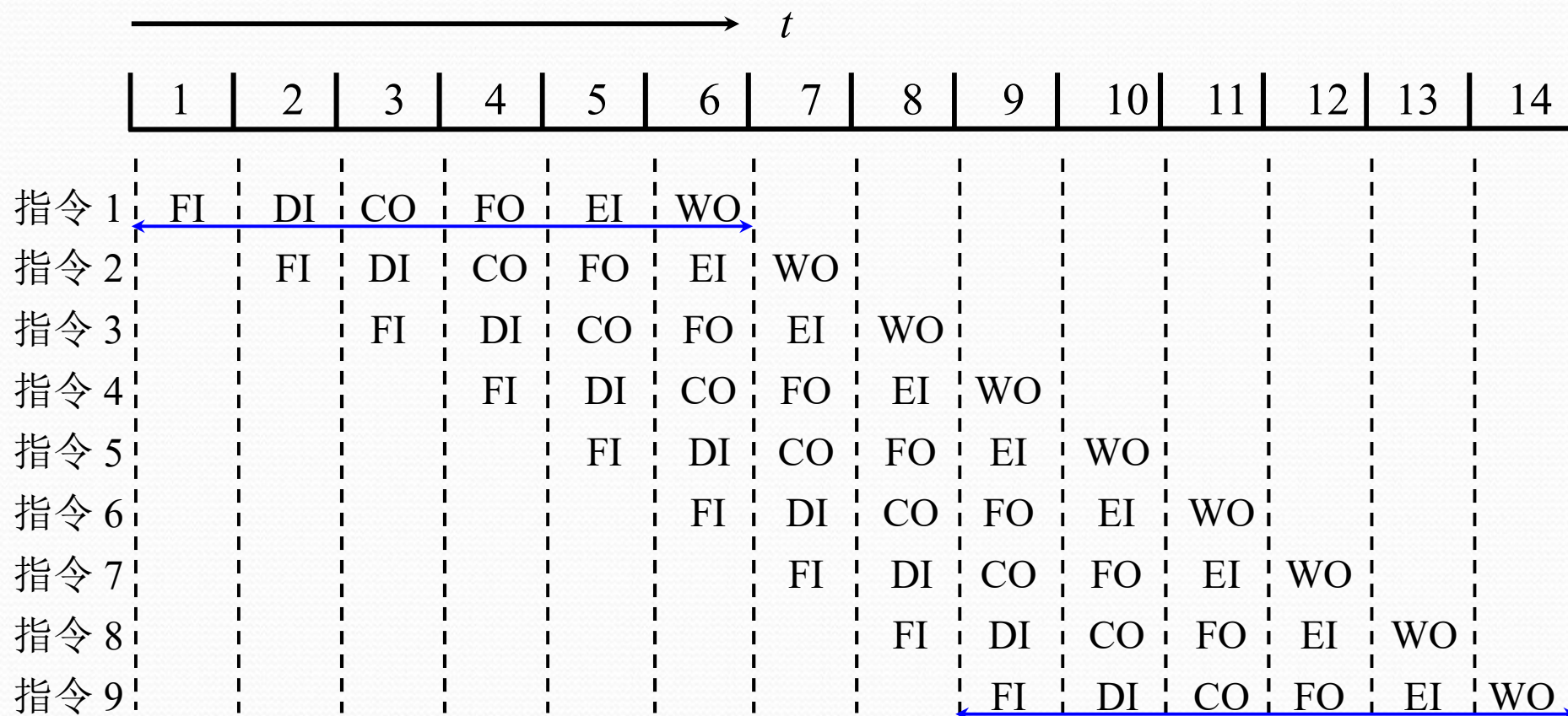
猜测法

指令的六级流水

为了进一步提高处理速度，可以将指令的处理过程分解为更细的几个阶段：

- 取指令(FI)：从存储器中取出一条指令
- 指令译码(DI)：确定操作性质和操作数地址形式
- 计算操作数地址(CO)：计算操作数EA
- 取操作数(FO)：取操作数
- 执行指令(EI)：执行指令所需要的操作
- 写操作结果(WO)：将结果写回

指令的六级流水



完成一条指令

6 个时间单位

串行执行

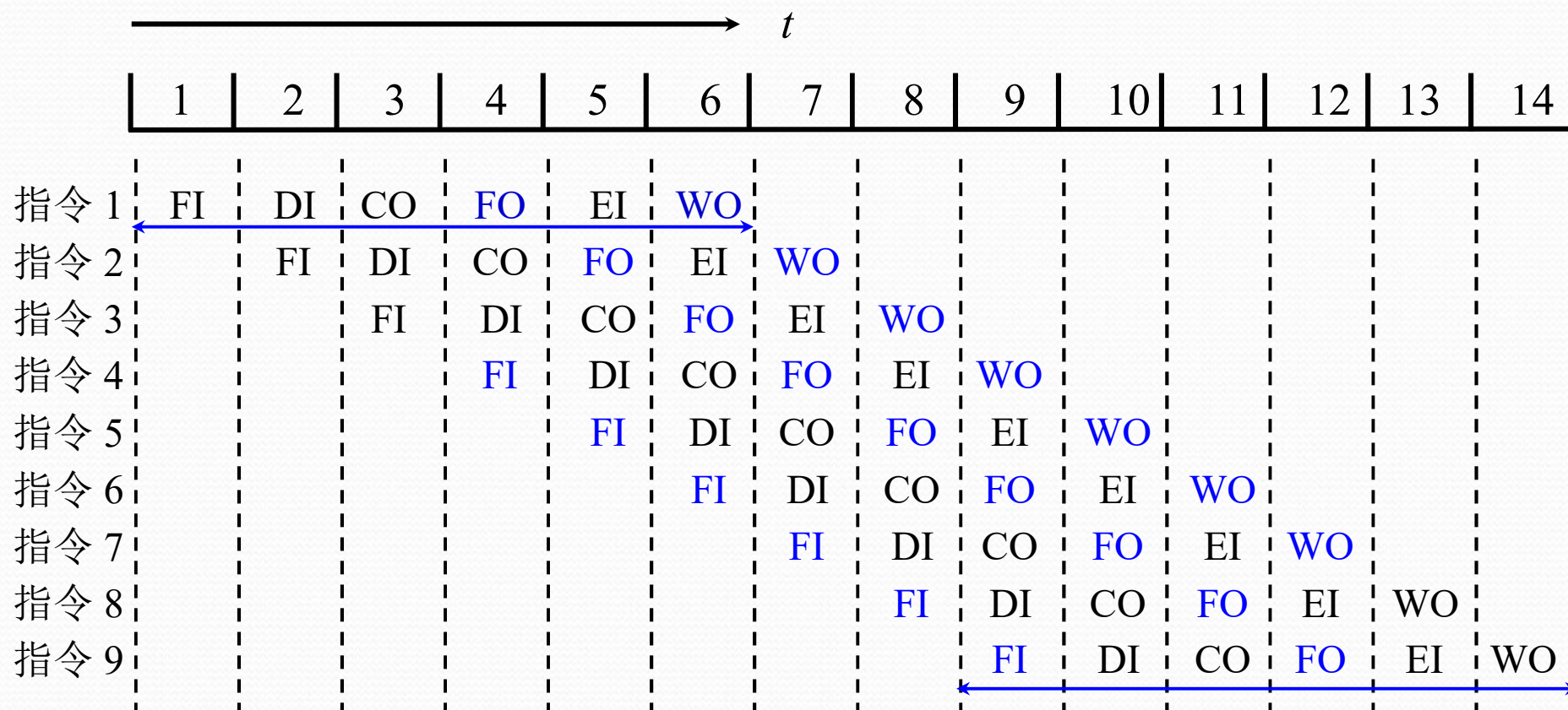
$6 \times 9 = 54$ 个时间单位

六级流水

14 个时间单位

结构相关

不同指令争用同一功能部件产生资源冲突



解决办法

- 停顿
- 指令存储器和数据存储器分开
- 指令预取技术（适用于访存周期短的情况）

数据相关

不同指令因重叠操作，可能改变操作数的 读/写访问顺序

- 写后读相关(RAW)

SUB R_1 , R_2 , R_3 ; $(R_2) - (R_3) \rightarrow R_1$

ADD R_4 , R_5 , R_1 ; $(R_5) + (R_1) \rightarrow R_4$

- 读后写相关(WAR)

STA M, R_2 ; $(R_2) \rightarrow M$ 存储单元

ADD R_2 , R_4 , R_5 ; $(R_4) + (R_5) \rightarrow R_2$

- 写后写相关(WAW)

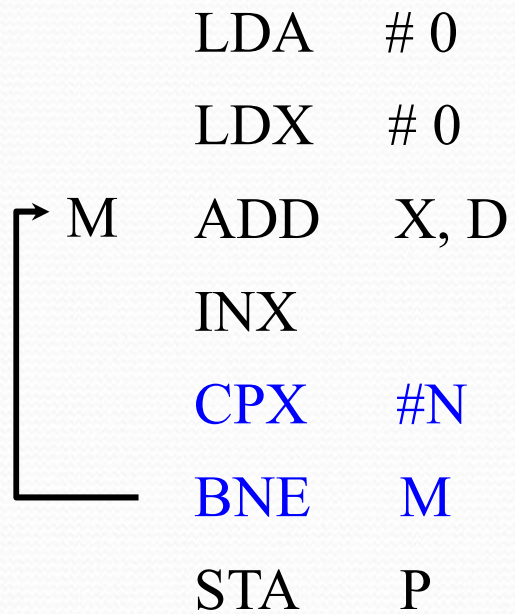
MUL R_3 , R_2 , R_1 ; $(R_2) \times (R_1) \rightarrow R_3$

SUB R_3 , R_4 , R_5 ; $(R_4) - (R_5) \rightarrow R_3$

解决办法 后推法 采用 旁路技术

控制相关

由转移指令引起



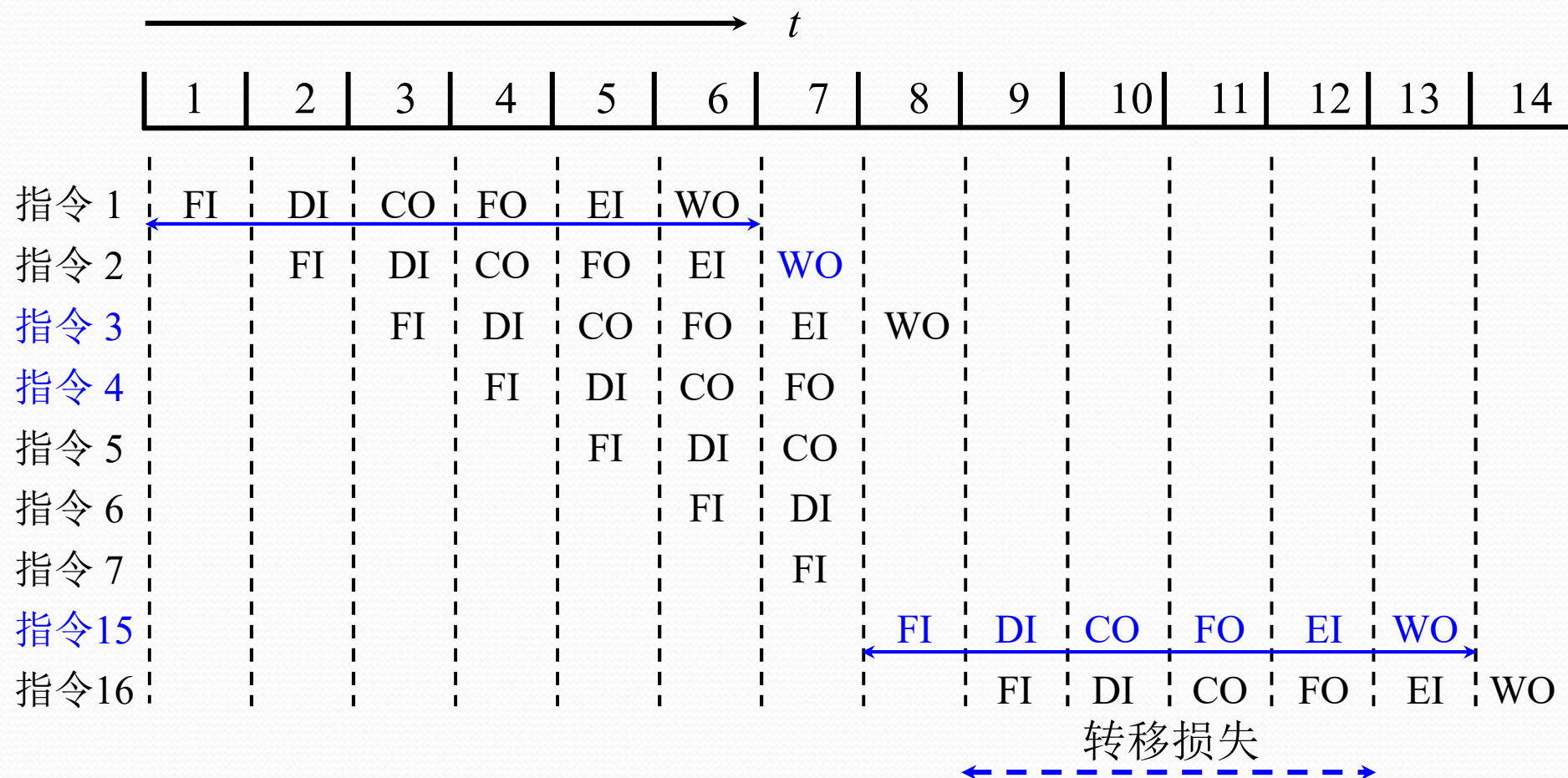
```
graph TD; LDA[LDA #0] --> LDX[LDX #0]; LDX --> ADD[ADD X,D]; ADD --> INX[INX]; INX --> CPX[CPX #N]; CPX --> BNE[BNE M]; BNE --> STA[STA P];
```

LDA # 0
LDX # 0
M ADD X, D
INX
CPX #N
BNE M
STA P

BNE 指令必须等
CPX 指令的结果
才能判断出
是转移
还是顺序执行

控制相关

设 指令3 是转移指令



流水线的吞吐率

单位时间内 流水线所完成指令 或 输出结果 的 数量

设 m 段的流水线各段时间为 Δt

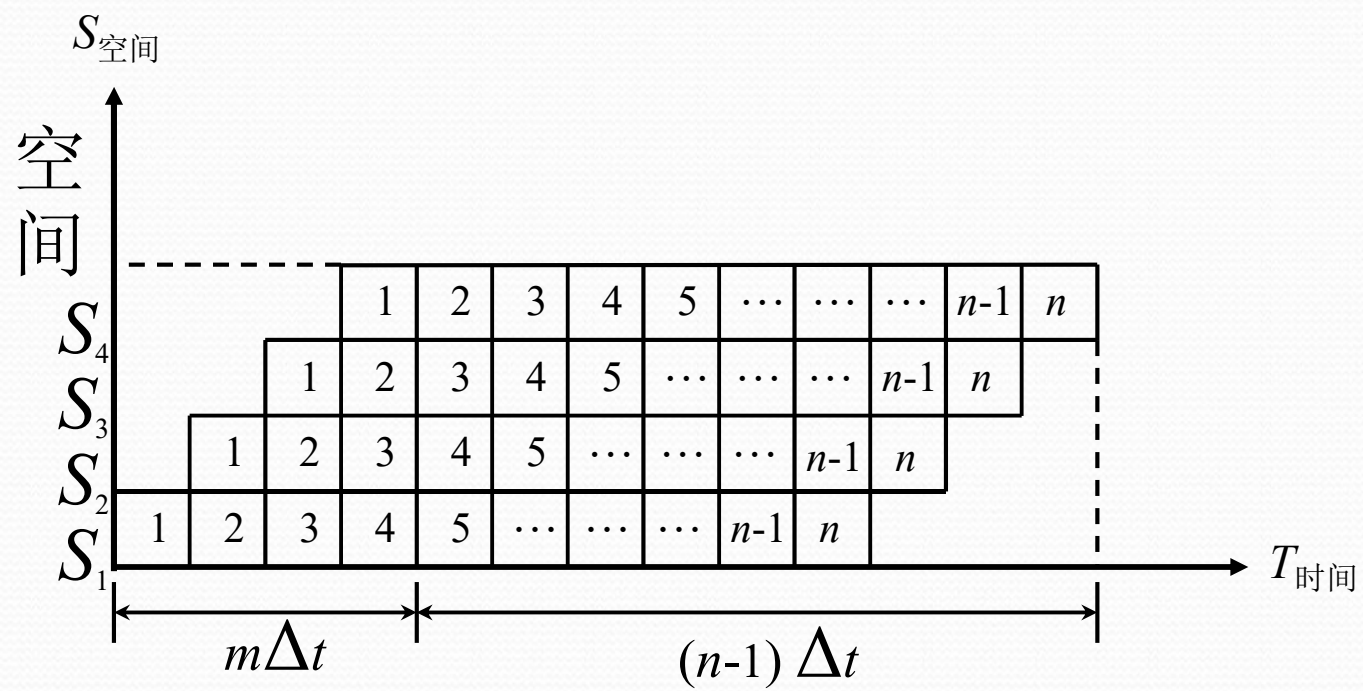
- 实际吞吐率

连续处理 n 条指令的吞吐率为

$$T_p = \frac{n}{m \cdot \Delta t + (n-1) \cdot \Delta t}$$

- 最大吞吐率

$$T_{pmax} = \frac{1}{\Delta t}$$



流水线的加速比

m 段的流水线的速度与等功能的非流水线的速度之比

设流水线各段时间为 Δt

完成 n 条指令在 m 段流水线上共需

$$T = m \cdot \Delta t + (n-1) \cdot \Delta t$$

完成 n 条指令在等效的非流水线上共需

$$T' = nm \cdot \Delta t$$

则

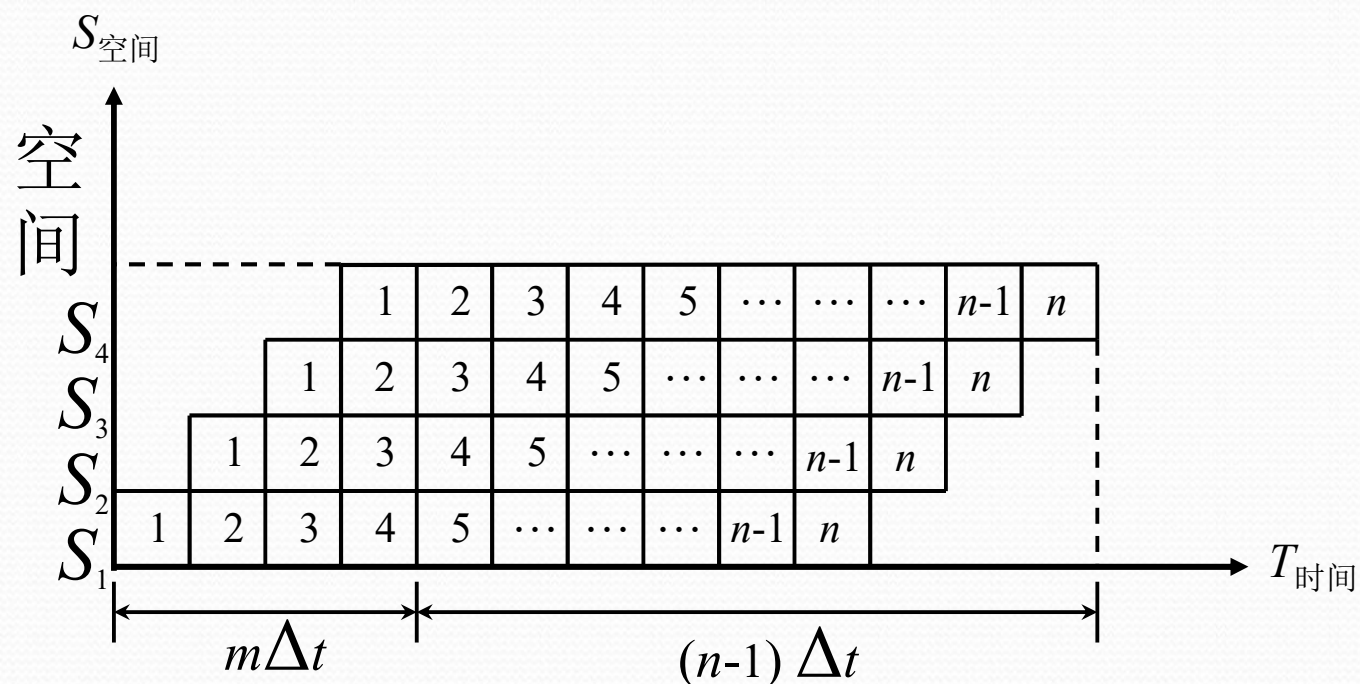
$$S_p = \frac{nm \cdot \Delta t}{m \cdot \Delta t + (n-1) \cdot \Delta t} = \frac{nm}{m + n - 1}$$

流水线的效率

流水线中各功能段的利用率

由于流水线有建立时间和排空时间

因此各功能段的设备不可能一直处于工作状态

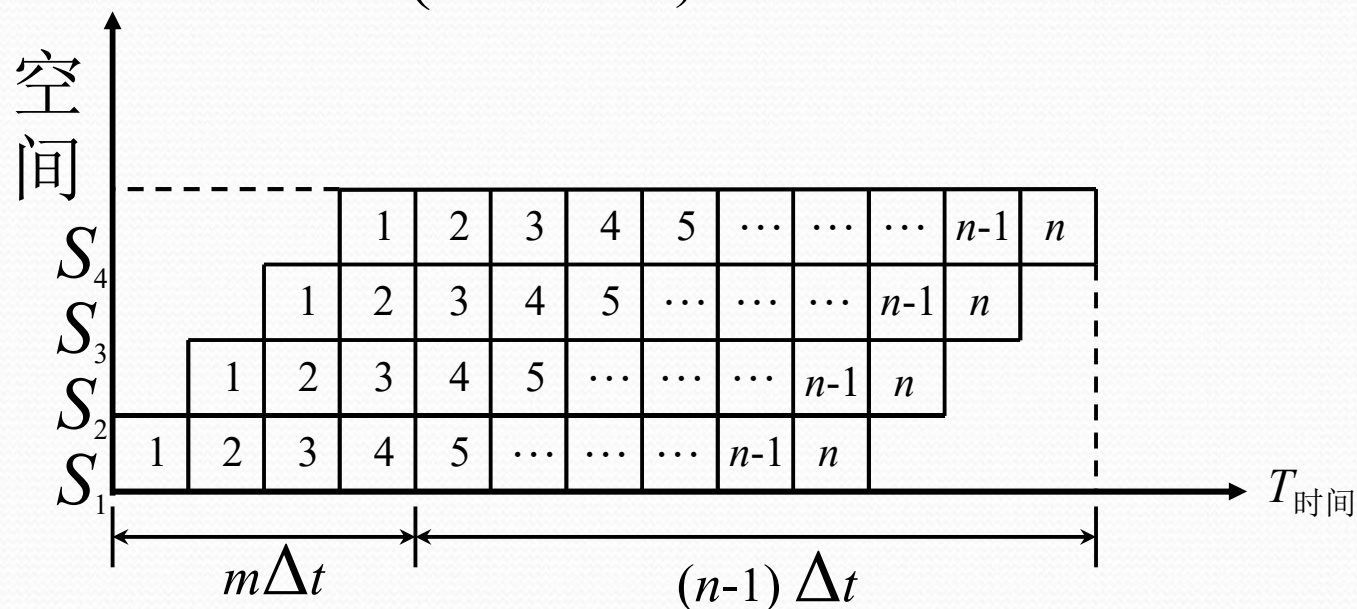


流水线效率

流水线中各功能段的利用率

效率 = $\frac{\text{流水线各段处于工作时间的时空区}}{\text{流水线中各段总的时空区}}$

$$S_{\text{空间}} = \frac{mn\Delta t}{m(m+n-1)\Delta t}$$



例：某指令流水线有取指令(IF)、译码(ID)、执行(EX)、写回寄存器堆(WB)4个过程段，共有100条指令连续输入此流水线，设时钟周期为50ns且不考虑数据相关等情况。

1. 试画出流水线的时空图；

2. 求流水线的实际吞吐率；

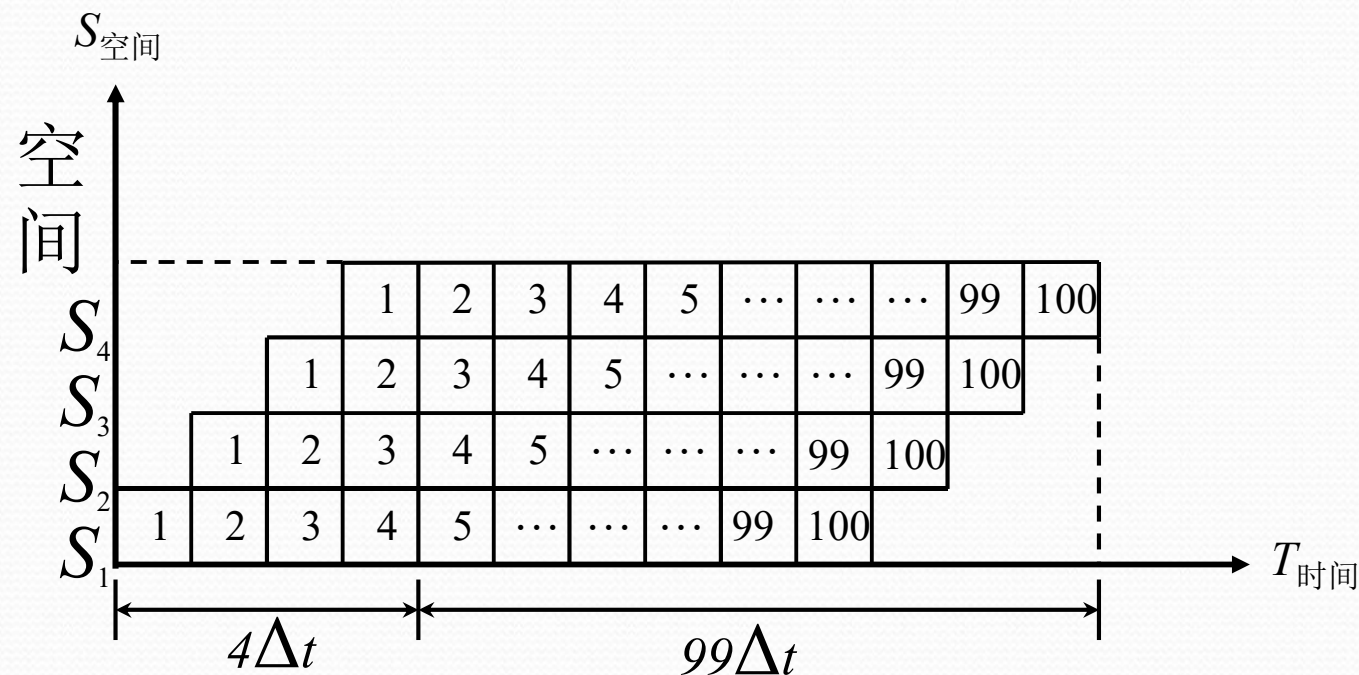
$$\text{吞吐率 } T = 100 / (103 * 50 * 10^{-9}) = 1.94 * 10^7$$

3. 求流水处理器的加速比；

$$\text{加速比 } S_p = 100 * 4 / 103$$

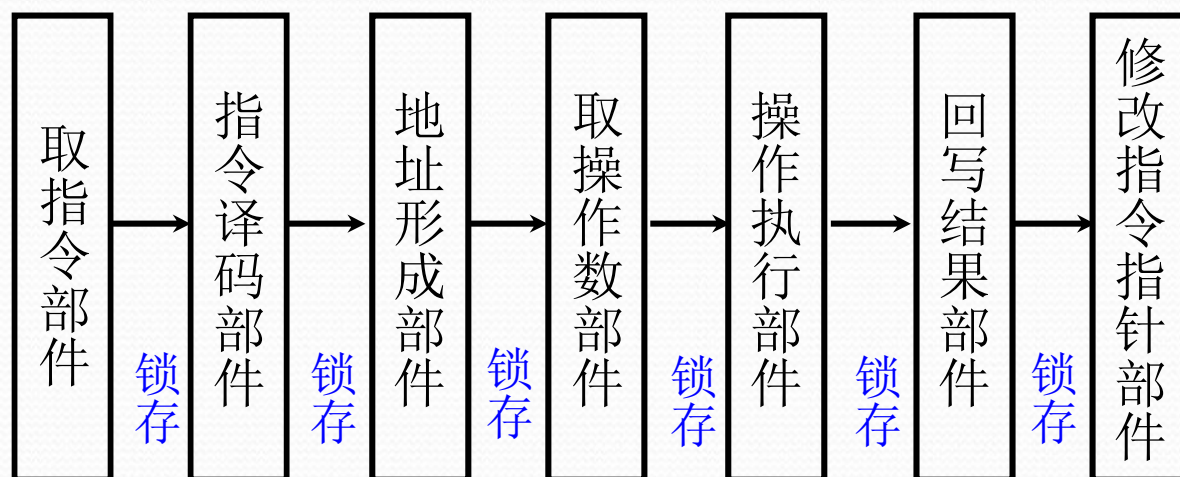
4. 求流水线的效率。

$$\text{效率} = 100 * 4 / (103 * 4)$$



指令流水线的结构

完成一条指令分 7 段，每段需一个时钟周期



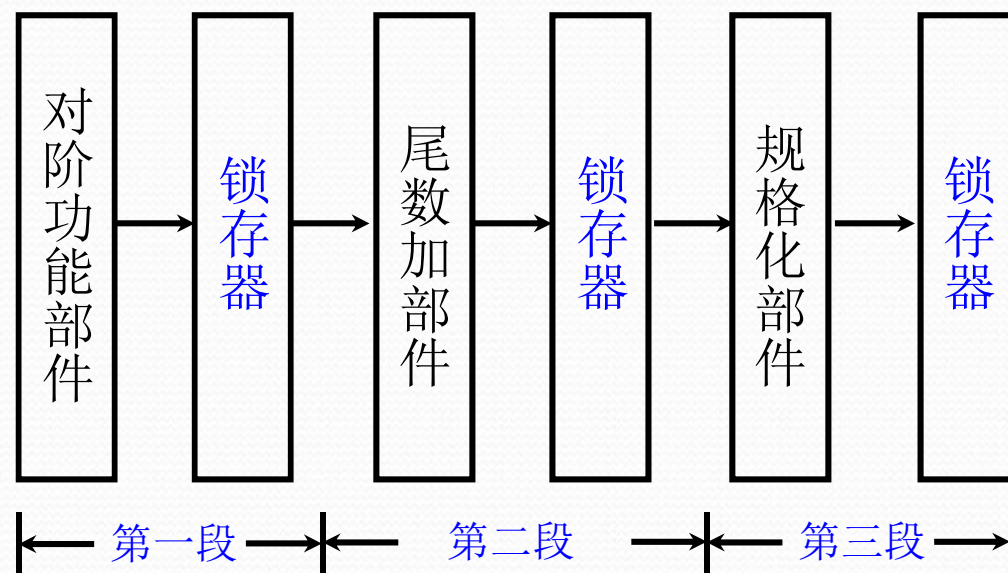
若 流水线不出现断流 1 个时钟周期出 1 结果

不采用流水技术 7 个时钟周期出 1 结果

理想情况下，7 级流水 的速度是不采用流水技术的 7 倍

运算流水线的结构

完成 浮点加减运算，可分对阶、尾数求和、规格化三段



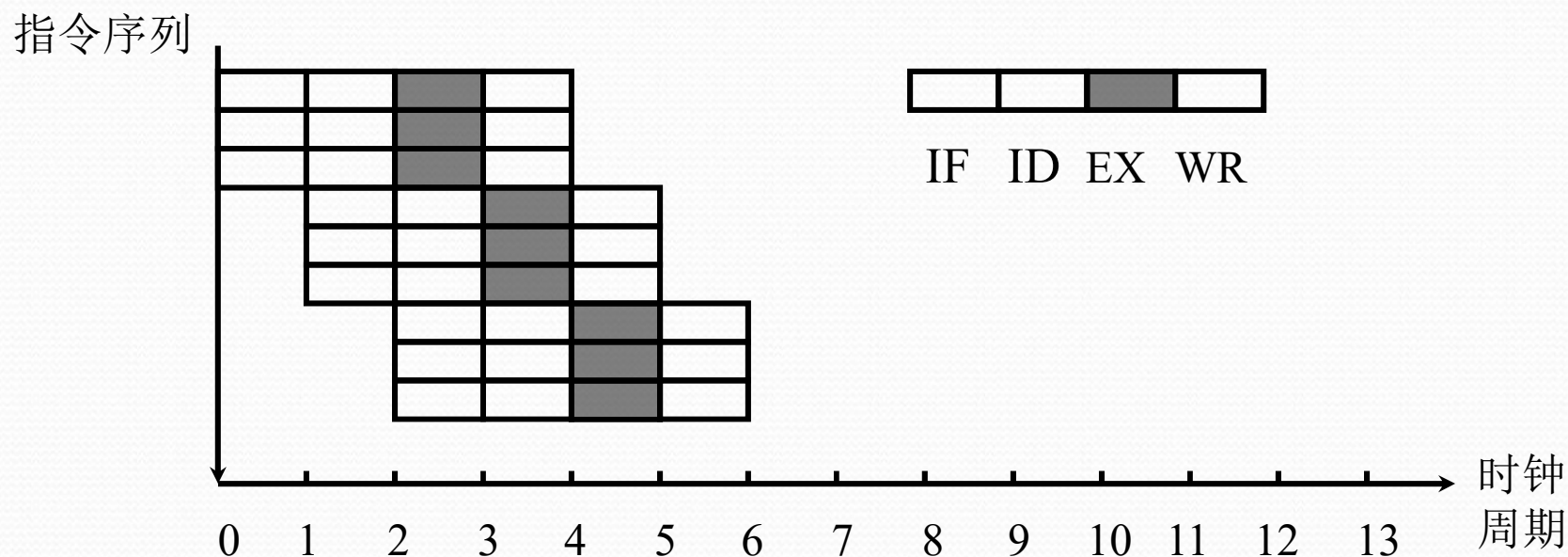
分段原则 每段操作时间尽量一致

流水线的多发技术*

- 超标量技术
- 超流水技术
- 超长指令字技术

超标量技术

- 每个时钟周期内可 并发多条独立指令
配置多个功能部件
- 不能调整 指令的 执行顺序
- 通过编译优化技术，把可并行执行的指令搭配起来



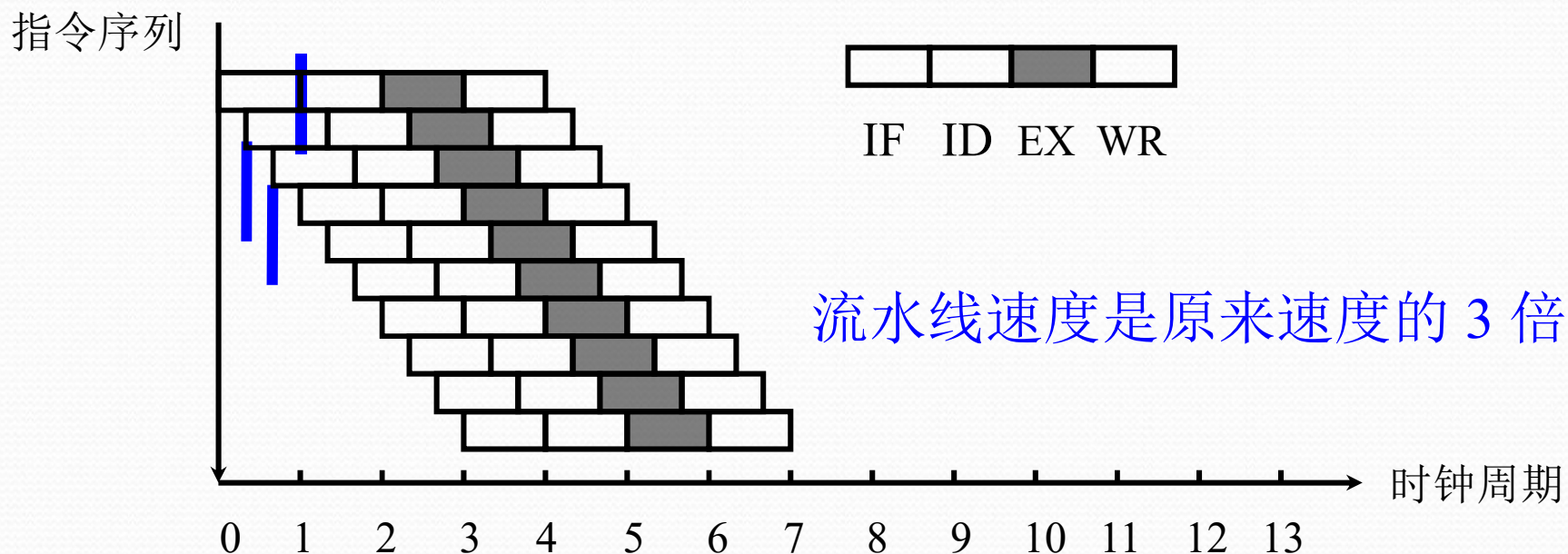
超流水技术

- 在一个时钟周期内再分段（3段）

在一个时钟周期内一个功能部件使用多次（3次）

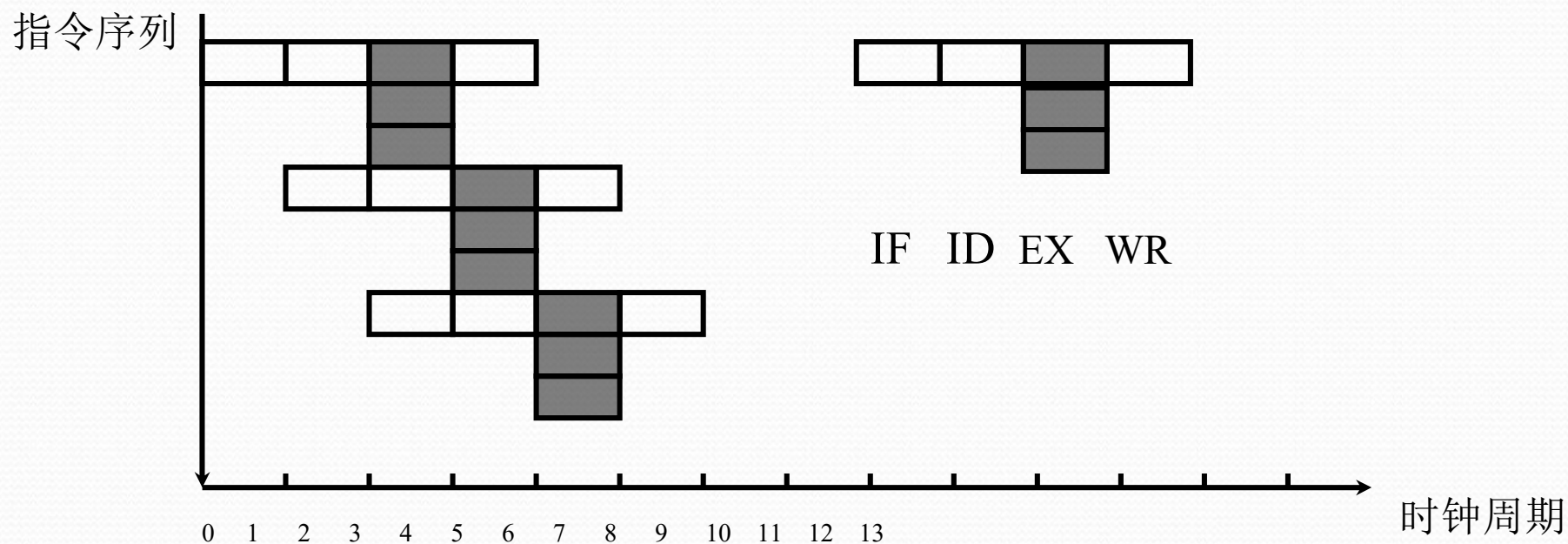
- 不能调整指令的执行顺序

靠编译程序解决优化问题



超长指令字技术

- 由编译程序 挖掘 出指令间 潜在的 并行性，
将 多条 能 并行操作 的指令组合成 一条
具有 多个操作码字段 的 超长指令字（可达几百位）
- 采用 多个处理部件

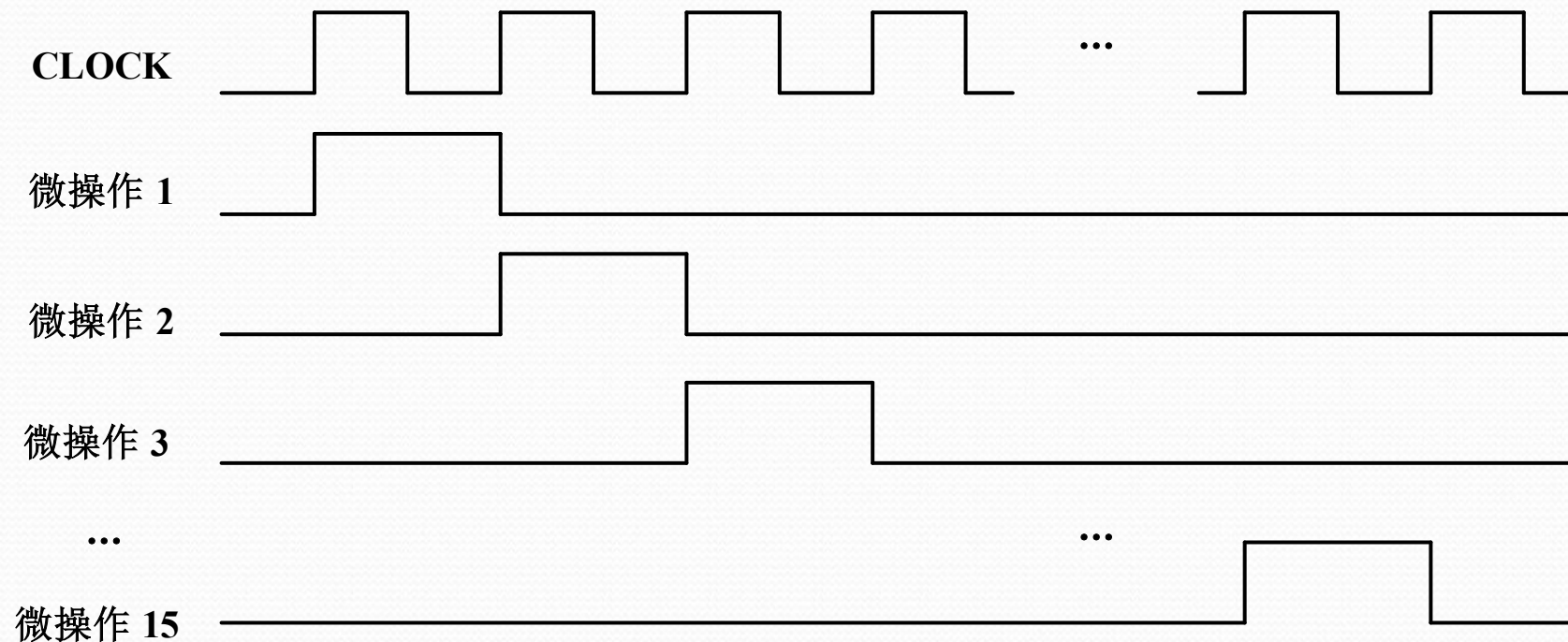


第四章 中央处理器

- 4.1 CPU的基本功能及结构
- 4.2 指令的执行过程
- 4.3 硬布线控制器
- 4.4 微程序控制器
- 4.5 流水线原理
- 4.6 控制器的控制方式

同步控制方式

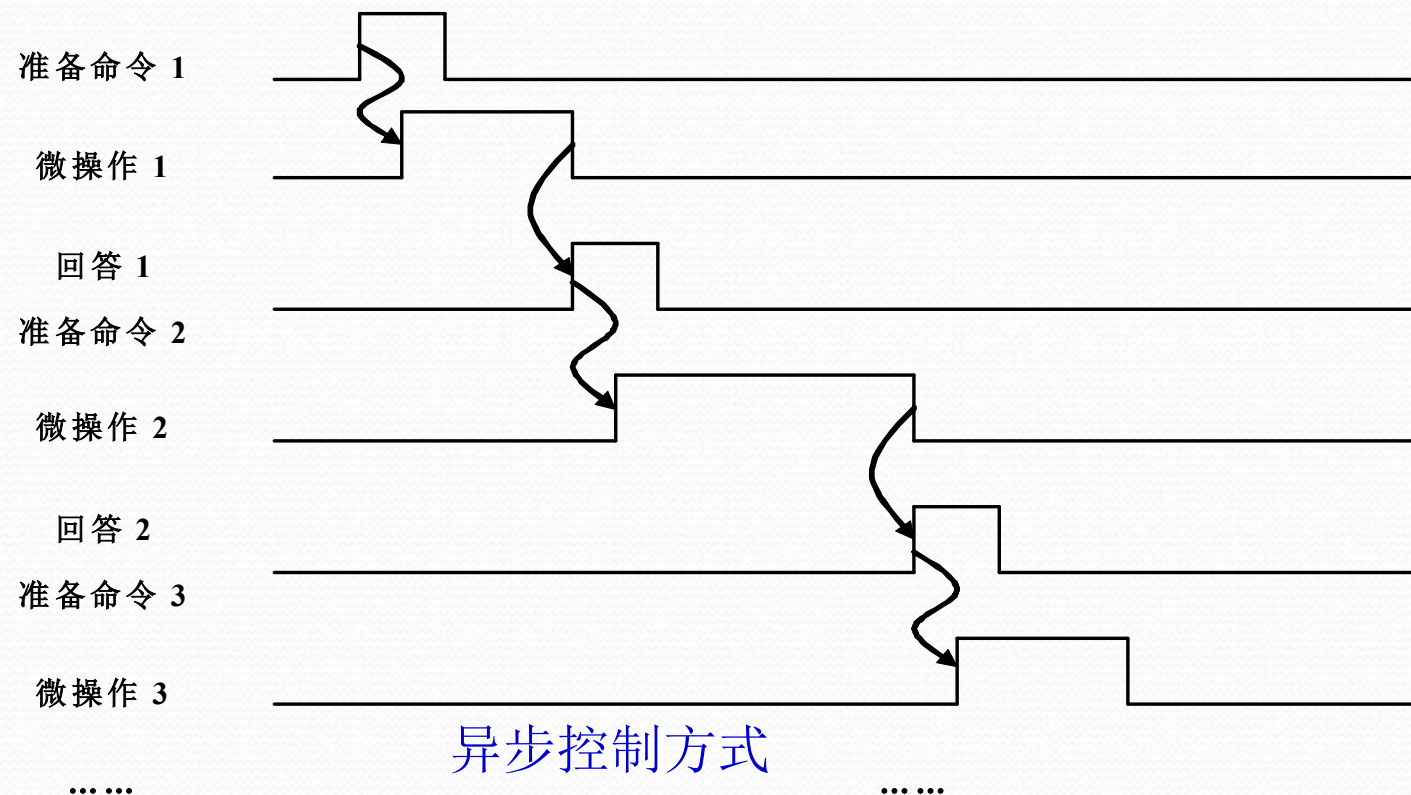
同步控制方式又被称为统一控制方式、集中控制方式或中央控制方式，是指机器有统一的时钟信号，所有的微操作控制信号都与时钟信号同步，且机器周期具有完全相同的执行时间。



同步控制方式

异步控制方式

异步控制方式又可以称为可变时序控制方式、分散控制方式或局部控制方式。是指各项操作不采用统一的时序信号控制，而根据指令或部件的具体情况决定，需要多少时间就安排多少时间。微操作控制信号采用“起始—微操作—结束”方式进行工作。



联合控制方式

集中以上两种控制方式的优点构成。

基本做法是将指令周期分成多个机器周期，每个机器周期中再分成多个节拍，于是各类机器指令可取不同的机器周期数作为各自的指令周期。这种控制方式不浪费很多时间，控制上又不很复杂，成为现代计算机中广泛采用的控制方式。

小结

1. 控制器内部构成的主要构件
2. 组合逻辑控制器的构成原理
3. 微程序控制器原理
4. 流水线技术
5. 控制方式

作业

P.370: 1, 2, 11, 24

P.393: 1, 4, 6

P.420: 2, 21, 22