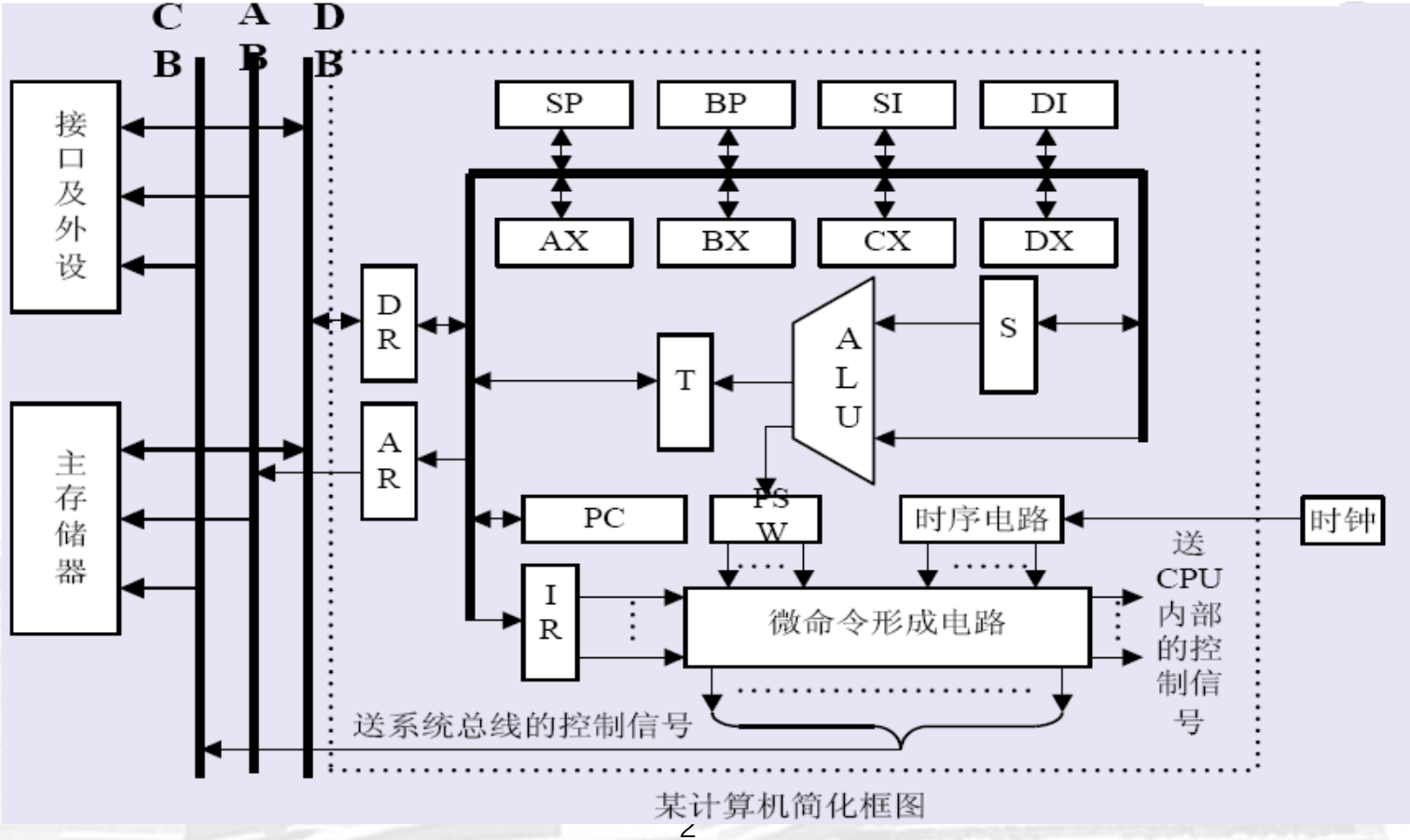
《计组I》
第五章　处理器设计2

计算机科学与技术学院

温故 —关于指令系统和CPU(1/2)

- **CPU内部的主要组成?**

1. **部件：寄存器、ALU、控制器、其它**
2. **数据通路(DataPath)：总线(BUS)、专用通路**
3. **外部交互：外部总线**



某计算机简化框图

深入CPU指令执行过程

- 取指令Fetch
  - PC->AR
  - AR->AB
  - AB->MAR
  - Read (MM->MDR)
  - MDR->DB
  - DB->DR
  - DR->IR
  - PC++

CS:IP指向下一条指令的地址

内存里的AR

内存里的DR

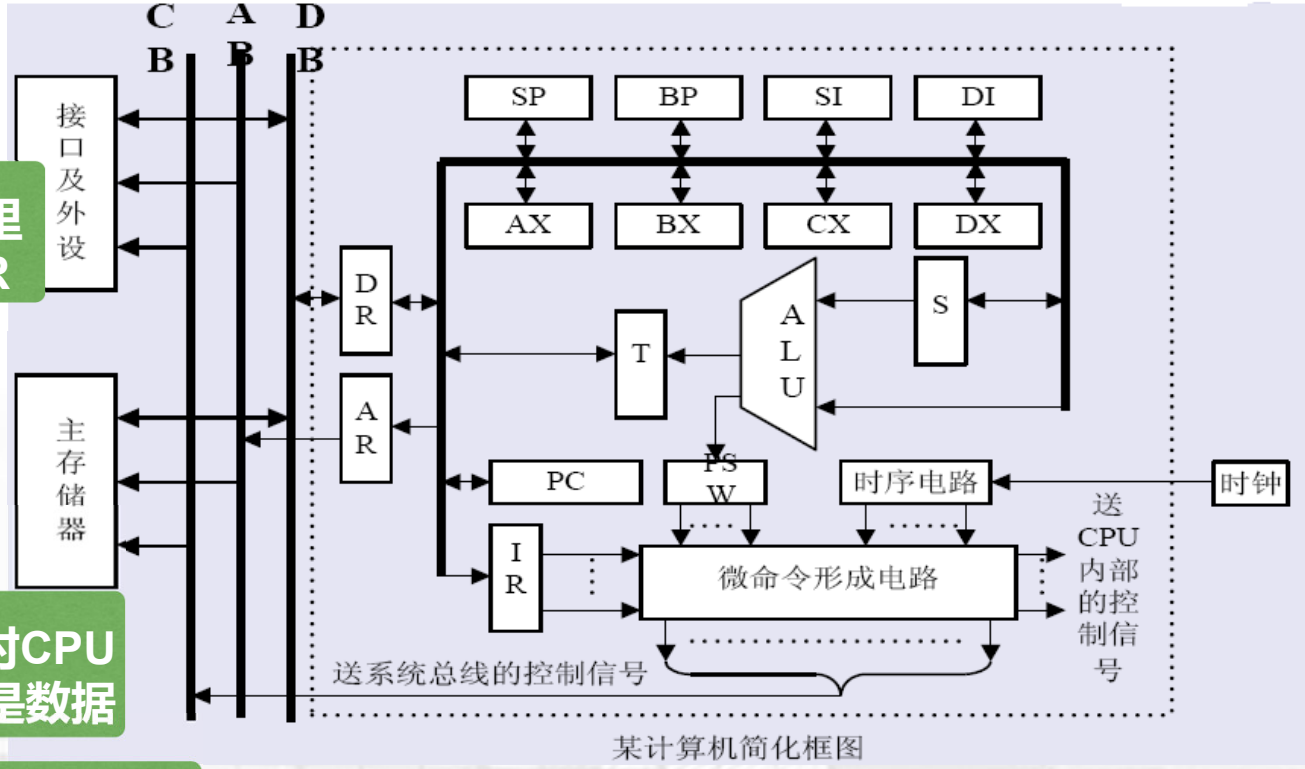
内存

取出的指令对CPU来说，就是数据

把“指令数据”导出IR才真的是看作指令

IP指针自加1

以指令add [2000h], ax为例



某计算机简化框图

温故 —关于指令系统和CPU

- 什么是微操作？什么是微命令？
- 微操作：CPU不可分解的操作，以含有一个寄存器传递操作为标志
- 微命令：完成微操作的控制信号
- CPU中的控制器什么用途？
- 发出满足一定时序关系的信号，
- 实现指令系统所规定的各条指令的功能，
- 保证计算机系统正常运行
- 时序信号的用途？
- 将各种控制信号严格定时，在时间上相互配合完成某一功能

温故 —关于指令系统和CPU

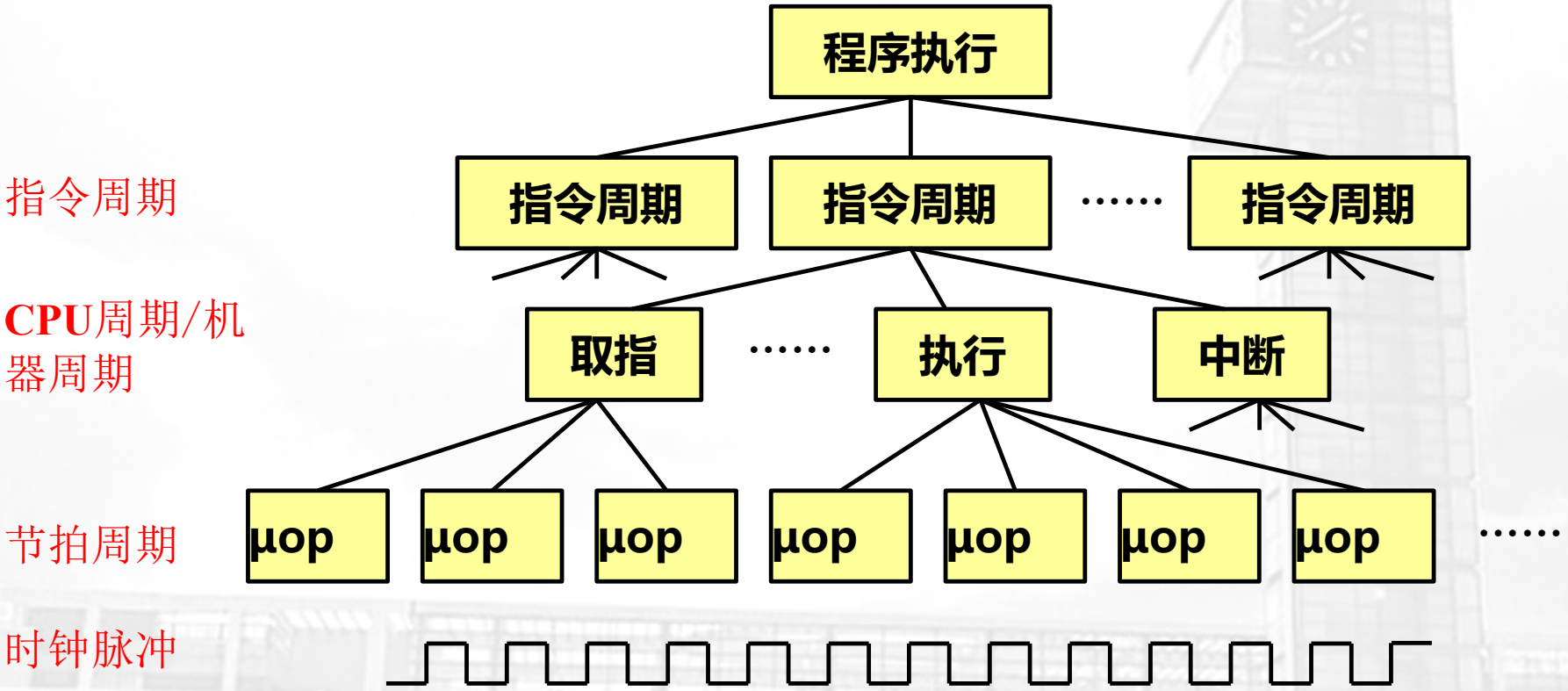
- **微操作**：处理器（CPU）的基本或原子操作。
  - CPU可以实现的、不可分解的操作动作
  - 以含有一个寄存器传递（移进、移出）操作为标志
- 每一个**微操作**是通过**控制器**将**控制信号**发送到相关部件上引起**部件动作**而完成的。
  - 这些控制微操作完成的**控制信号**称为**微命令**
  - **微命令**是由**控制器**产生的

$$\underbrace{AR \leftarrow PC;}_{\text{微操作}} \quad \underbrace{PC_{out},\ AR_{in}}_{\text{微命令}}$$

控制器的设计

- 揭晓这两种设计之前，想几个问题：
  - 控制单元，它的输出是什么？
  - 众多（汇编）指令对应的微操作需要存储，如何存储？
  - 如何根据时序信号和操作码产生输出？
    - 带有时序的逻辑关系如何表达出来？

- **指令周期、CPU周期、节拍周期、时钟脉冲的层次关系?**



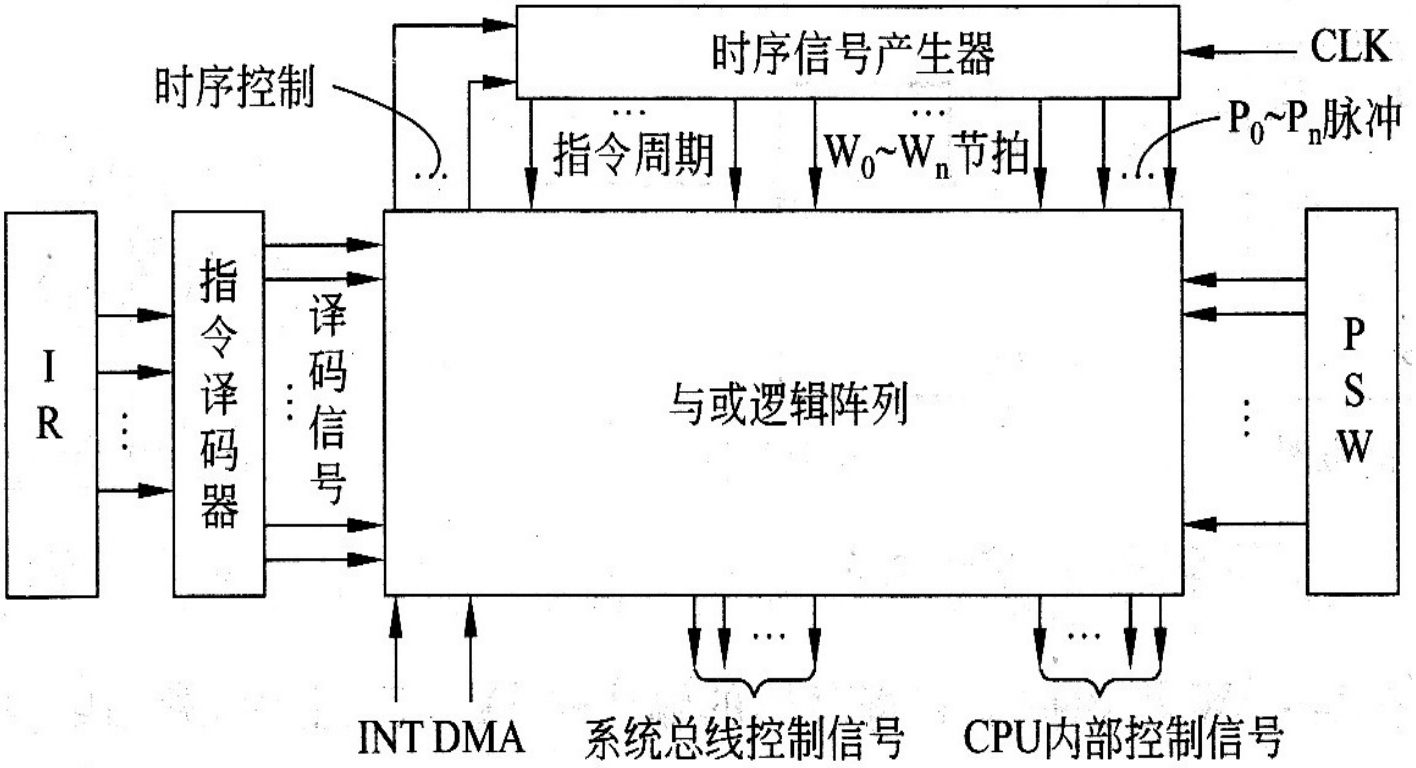
控制器的设计

- 控制器有两种实现方式
  - 组合逻辑控制器（硬布线/硬编码）
    - 硬，真的硬~
  - 微程序控制器

组合逻辑控制器

构成：

- **时序信号产生器**
  - **指令周期**
  - **节拍(W0, W1, W2 …)**
  - **触发脉冲(P0, P1, P2…)**
- **译码器**
- **PSW**
- **与或逻辑阵列**



逻辑阵列极端复杂：把各种指令的微指令序列/时序/控制/反馈都要硬编码在阵列里！

How ??~!

组合逻辑控制器

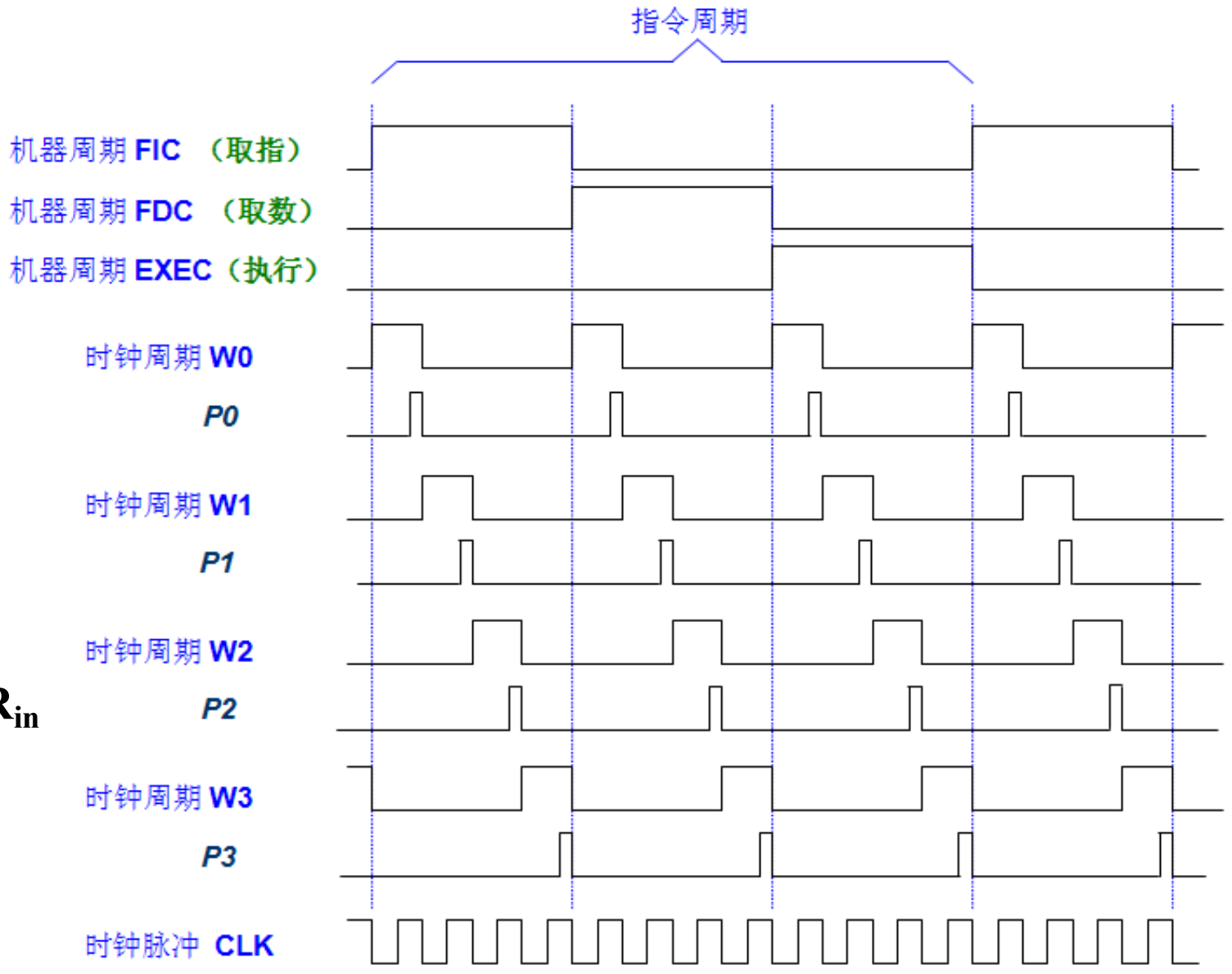
FIC 微操作命令

W_0　PC_{out}，AR_{in}

W_1　AR→AB，
RD，PC+1

W_2　MM→DB，DR_{in}

W_3　DR_{out}，IR_{in}



组合逻辑控制器

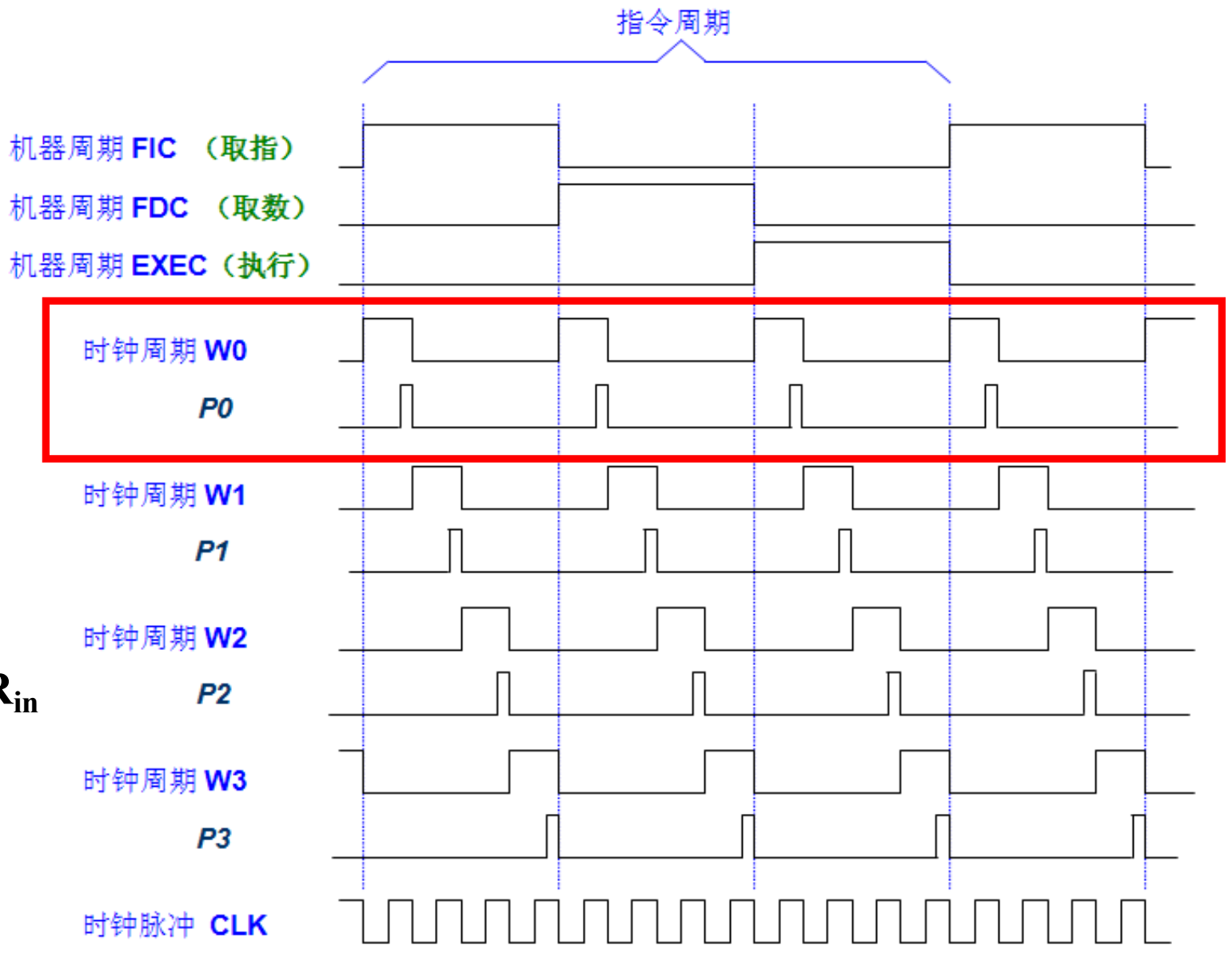
FIC 微操作命令

W_0　PC_{out}，AR_{in}

W_1　AR→AB，
RD，PC+1

W_2　MM→DB，DR_{in}

W_3　DR_{out}，IR_{in}



组合逻辑控制器

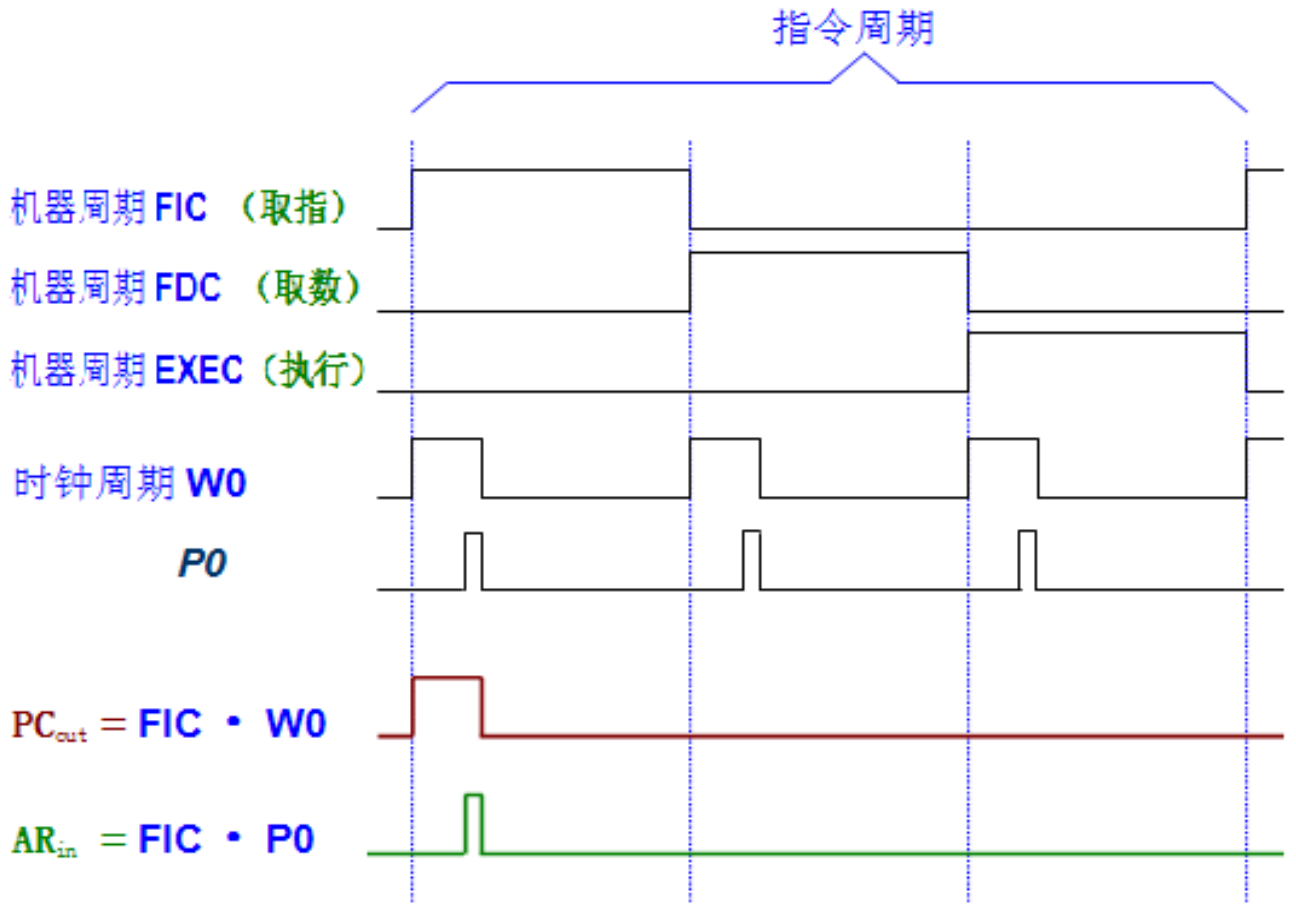
FIC 微操作命令

W_0　PC_{out}，AR_{in}

W_1　AR→AB，
RD，PC+1

W_2　MM→DB，DR_{in}

W_3　DR_{out}，IR_{in}



组合逻辑控制器

1. 取指周期中产生的控制信号相同。
2. 通常，一个控制信号在若干条指令的某些机器周期中都需要，应将它们组合起来。

   如：$PC_{out} = FIC \cdot W_0 + FDC \cdot$ 双字指令 $\cdot W_0 + \cdots\cdots$

所以，控制信号‘ + ’可以用两级门电路产生

—— 第一级为与门，第二级为或门。 “组合逻辑电路”

3. 逻辑式的化简以减少逻辑电路规模。
4. 构成与或逻辑阵列。

组合逻辑控制器

- PC_{out}出现在：
  - 取指周期的W0节拍
  - 指令JZ执行周期（假设为第2个CPU周期M2）的T1节拍
  - 指令CALL（X）执行周期的T4节拍
  - ……
- 生成PC_{out}的逻辑表达式为：

$$PC_{out} = FIC \cdot W0 + T1 \cdot JZ(\text{相对寻址}) \cdot (ZF==1) + T4 \cdot CALL(\text{间接寻址}) + \cdots\cdots$$

描述了“当x指令的y周期时的第z节拍…，发出PCout信号；

也就是说：组合逻辑控制器，是用非常机械的方式去描述其逻辑的：枚举所有微操作所需的发生的条件，将其描述为与或门逻辑。

组合逻辑控制器

- AR_{in}出现在：
  - 取指周期的T1节拍
  - 指令 MOV R0,X 和指令 MOV (R1),R0 执行周期的T1节拍
  - 指令 SUB R0,(X) 执行周期的T1和T3节拍
  - 指令 IN R0,P 和指令 OUT P,R0 执行周期的T1节拍
  - 指令 PUSH R0 执行周期的T2节拍
  - 指令 POP R0 执行周期的T1节拍
  - 指令 CALL (X) 执行周期的T2和T4节拍
  - 指令 RET 执行周期的T1节拍
  - ……
- 生成AR_{in}的逻辑表达式为：

组合逻辑控制器

- **生成AR_{in}的逻辑表达式为：**

  AR_{in}＝T1＋T4·MOV(源操作数直接寻址+目的操作数寄存器间接寻址)＋(T4+T6) ·SUB(源操作数间接寻址)＋T4·(IN(直接寻址)+OUT(直接寻址))＋T5·PUSH＋T4·POP＋(T5+T7)·CALL(间接寻址)＋T4·RET＋……

描述了“当x1指令的y1周期时的第z1节拍…，发出PCout信号；

或者

当x2指令的y2周期时的第z2节拍…，发出PCout信号；

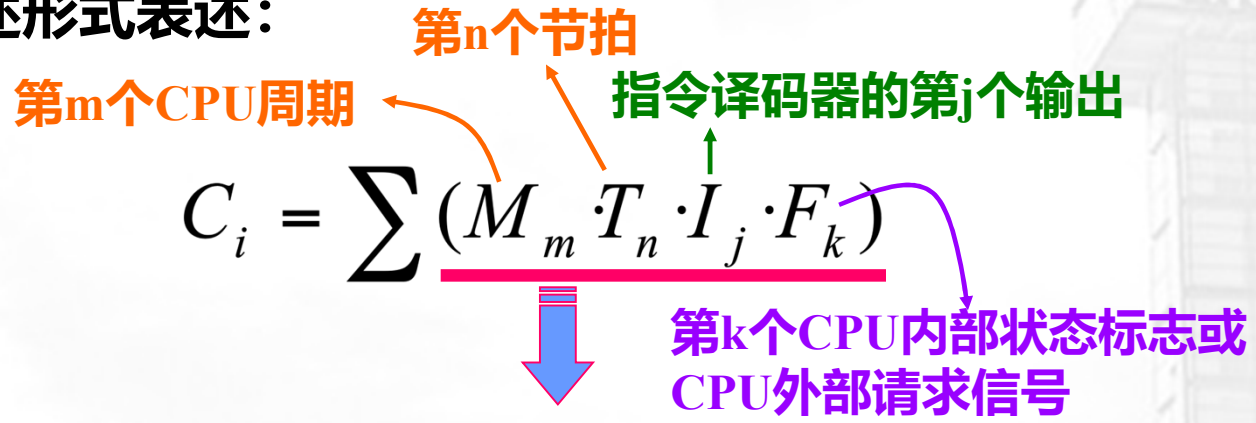
或者…

也就是说：组合逻辑控制器，是用非常机械的方式去描述其逻辑的：枚举所有微操作所需的发生的条件，将其描述为与或门逻辑。

组合逻辑控制器

- **组合逻辑控制器，它到底描述的是什么?**

由控制单元产生并加载到CPU内外的全部控制信号均可用下述形式表述：



在执行指令I_j时，若状态F_k满足要求，则在第m个机器周期M_m的第n个节拍T_n，控制单元发出C_i控制命令

所以称之为“基于组合逻辑的控制器”

组合逻辑控制器

- 组合逻辑控制单元，它的输出是什么?
  - 总线和内部的控制信号
- 众多（汇编）指令对应的微指令需要存储，如何存储?
  - 严格来说，组合逻辑控制单元并不单纯存储微程序;
- 如何根据时序信号和操作码产生输出?
  - 带有时序的逻辑关系如何表达出来?
    - 每一个控制信号的输出，由所有可能触发该信号的输入组合以“或”操作形式组合起来，每种输入组合内部是用“与”操作表达;
    - b.t.w“存储”了微程序和执行逻辑

组合逻辑控制器

小结：

- **每个控制信号的逻辑表达式就是一个与或逻辑方程式。**
- **将所有控制信号的与或逻辑电路组合在一起就构成了硬布线控制单元。**
- 时间信息、指令信息、状态信息是硬布线控制单元的输入，控制信号是硬布线控制单元的输出。

- **采用硬布线法设计控制器的特点：**
  - **一旦完成了控制器的设计，改变控制器行为的唯一方法就是重新设计控制单元 → 修改不灵活**
  - **使用PLD(PAL, PLA, GAL, FPGA)实现！**
  - **在现代复杂的处理器中，需要定义庞大的控制信号逻辑方程组 → 与或组合电路实现困难**
    - ➔ **微程序设计法**

控制器设计——微程序控制器

（基于）微程序（的）控制器

- **看两条机器指令的微操作执行过程：**
  - **横向切割、面向器件的操作模式——组合逻辑控制器**
  - **纵向切割、面向流程的模式——微程序控制器**

① ADD BX，[DI]

a) PC_{out}，AR_{in}

b) AR→AB，RD，PC+1

c) MD→DB，DR_{in}

d) DR_{out}，IR_{in}

e) DI_{out}，AR_{in}

f) AR→AB，RD

g) MD→DB，DR_{in}

h) DR_{out}，S_{in}

i) S→ALU

j) BX_{out}→ALU

k) ADD→T，T_{in}

l) T_{out}，BX_{in}

② INC [BX]

a) PC_{out}，AR_{in}

b) AR→AB，RD，PC+1

c) MD→DB，DR_{in}

d) DR_{out}，IR_{in}

e) BX_{out}，AR_{in}

f) AR→AB，RD

g) MD→DB，DR_{in}

h) DR_{out}，S_{in}

i) S→ALU

j) ALU+1→T，T_{in}

k) T_{out}，DR_{in}

l) DR→DB，WR

（基于）微程序（的）控制器

(1) 工作原理：

- 指导思想：将程序控制的思想引入控制信号的形成和控制。
- 基本思想：按照微程序顺序，产生所需的控制信号。

相当于把控制信号存储起来，因此又称存储控制逻辑方法。

- **几方面要学习的内容：**
  - **原理**
  - **微地址形成**
  - **微指令编码**
  - **微指令设计**
  - **微程序编程**

① ADD BX, [DI]

a) PC_{out}, AR_{in}

b) AR→AB, RD, PC+1

c) MD→DB, DR_{in}

d) DR_{out}, IR_{in}

e) DI_{out}, AR_{in}

f) AR→AB, RD

g) MD→DB, DR_{in}

h) DR_{out}, S_{in}

i) S→ALU

j) BX_{out}→ALU

k) ADD→T, T_{in}

l) T_{out}, BX_{in}

② INC [BX]

a) PC_{out}, AR_{in}

b) AR→AB, RD, PC+1

c) MD→DB, DR_{in}

d) DR_{out}, IR_{in}

e) BX_{out}, AR_{in}

f) AR→AB, RD

g) MD→DB, DR_{in}

h) DR_{out}, S_{in}

i) S→ALU

j) ALU+1→T, T_{in}

k) T_{out}, DR_{in}

l) DR→DB, WR

（基于）微程序（的）控制器

- 一个微指令序列称作微程序（microprogram）或固件（firmware）。
- 通过一组微指令产生的控制信号，使一条机器指令中的所有微操作得以实现。
- 机器指令、微程序、微指令的关系：

一条机器指令

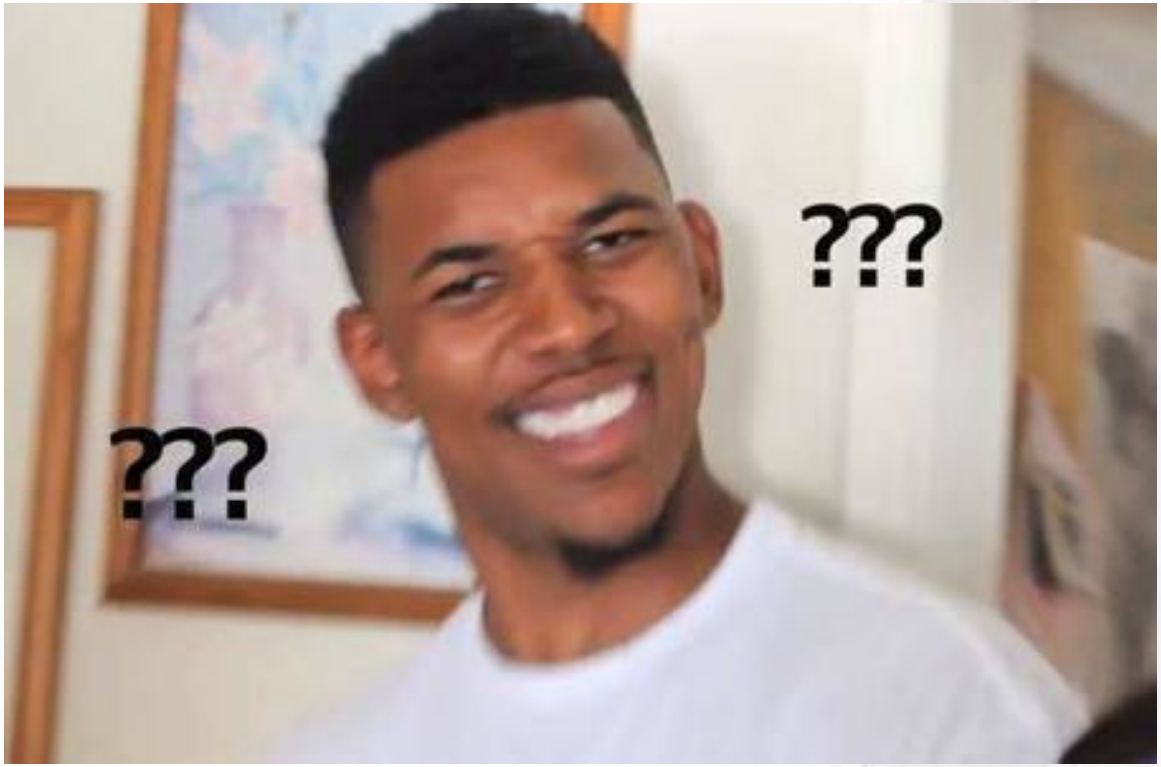
T_1：微操作1（命令1，命令2，…）
微操作2（命令1，命令2）
……
T_j：微操作i（命令1，命令2，…）
……
T_m：微操作n（命令1）

微指令1
微指令j
微指令m

一个微程序

程序由机器指令组成，存放在主存Memory中；
每一条机器指令，由一个微程序实现；
微程序由微指令组成，存放在CPU内部的控制存储器(Control Memory)中

- 微操作！~?
- 微命令！~?
- 微指令！~?
- 微程序！~?



微程序控制器：根据外部输入的信号，调用内部由微指令组成的微程序，输出微命令，完成一系列微操作。

- 微操作！~?
  - CPU中的一个原子的、基本的最小操作
- 微命令！~?
  - 触发/发起以上最小操作的命令/信号，由控制器发出（就是个电信号）
- 微指令！~?
  - 微程序控制器内，对微命令的指令化对应；
  - 经控制器尾部的译码器转换为微命令
- 微程序！~?
  - 一组微指令，就是一微程序

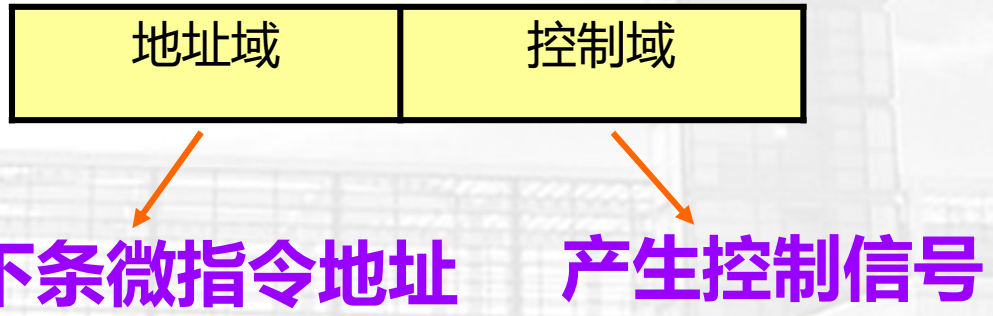
（基于）微程序（的）控制器

- 一条（机器）指令对应一个微程序，该微程序包含从取指令到执行指令一个完整微操作序列对应的全部微指令，它被存入一个称为控制存储器（control memory）的ROM中。
- CM中存放着指令系统中定义的所有指令的微程序。
- 微指令周期：一条微指令执行的时间（包括从控制存储器中取得微指令和执行微指令所用时间）。

- 指令系统？ <—> 微指令系统？ 各种指令系统的逻辑无脑套用！

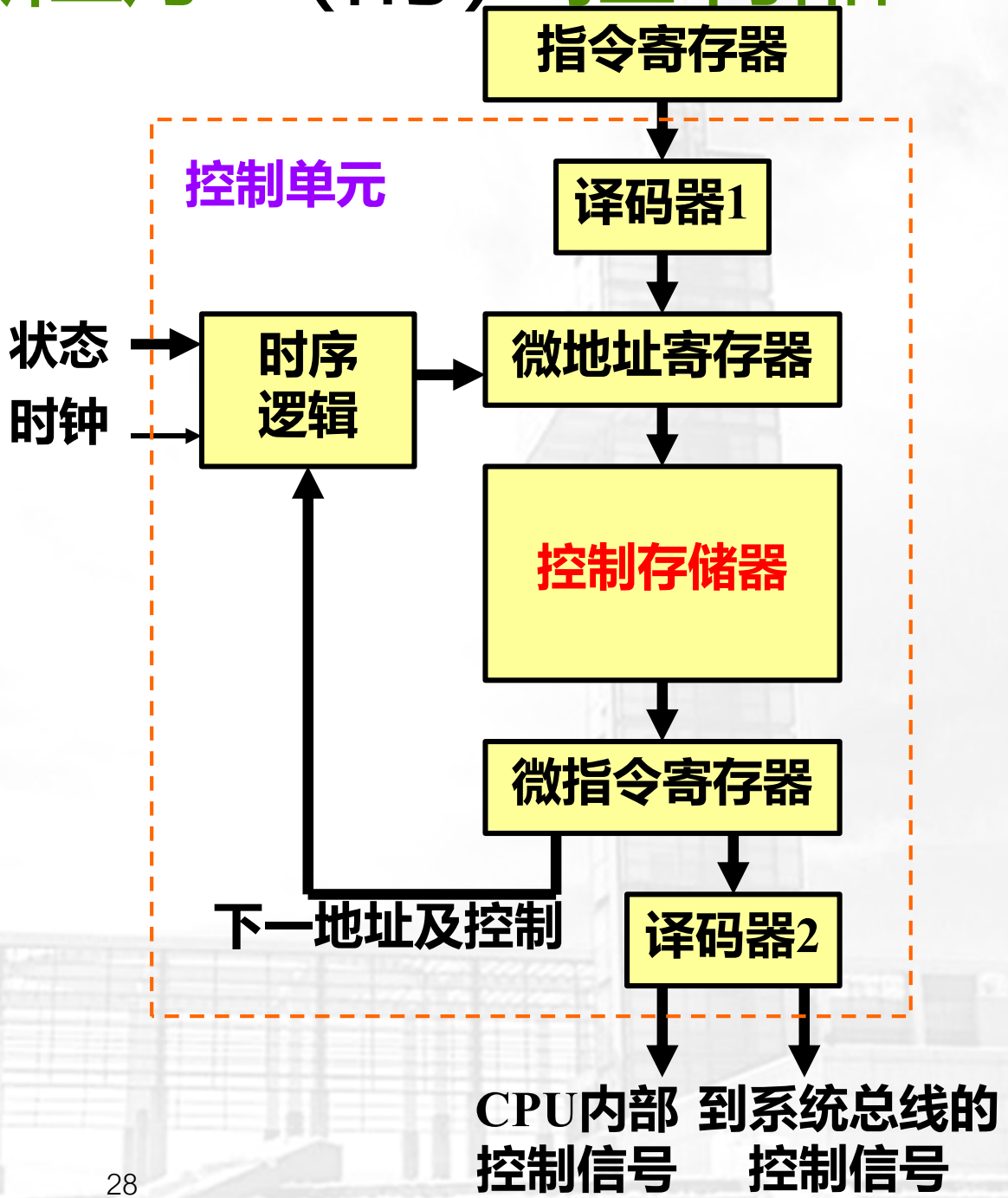
（基于）微程序（的）控制器

- 一条（机器）指令对应一个**微程序**，该微程序包含从取指令到执行指令一个**完整微操作序列**对应的全部**微指令**，它被存入一个称为**控制存储器**（control memory）的ROM中。
- CM中存放着指令系统中定义的**所有指令的微程序**。
- 微指令周期：一条微指令执行的时间（包括从控制存储器中**取得微指令**和**执行微指令**所用时间）。
- 微指令的一般格式：



（基于）微程序（的）控制器

- 核心结构：
  - 控制存储器（CM）
    - 微指令长度
    - 微程序占用的存储单元数
  - 微指令寄存器μIR、微地址寄存器μAR
  - 微地址形成电路
  - 时序逻辑
    - 依据时钟按节拍为控制存储器提供读出控制信号。
    - 在微程序运行时依据CPU内外状态（ALU标志、中断请求、DMA请求等）和当前微指令地址域的信息生成下一条微指令地址，并将其装入到微地址寄存器中。



（基于）微程序（的）控制器

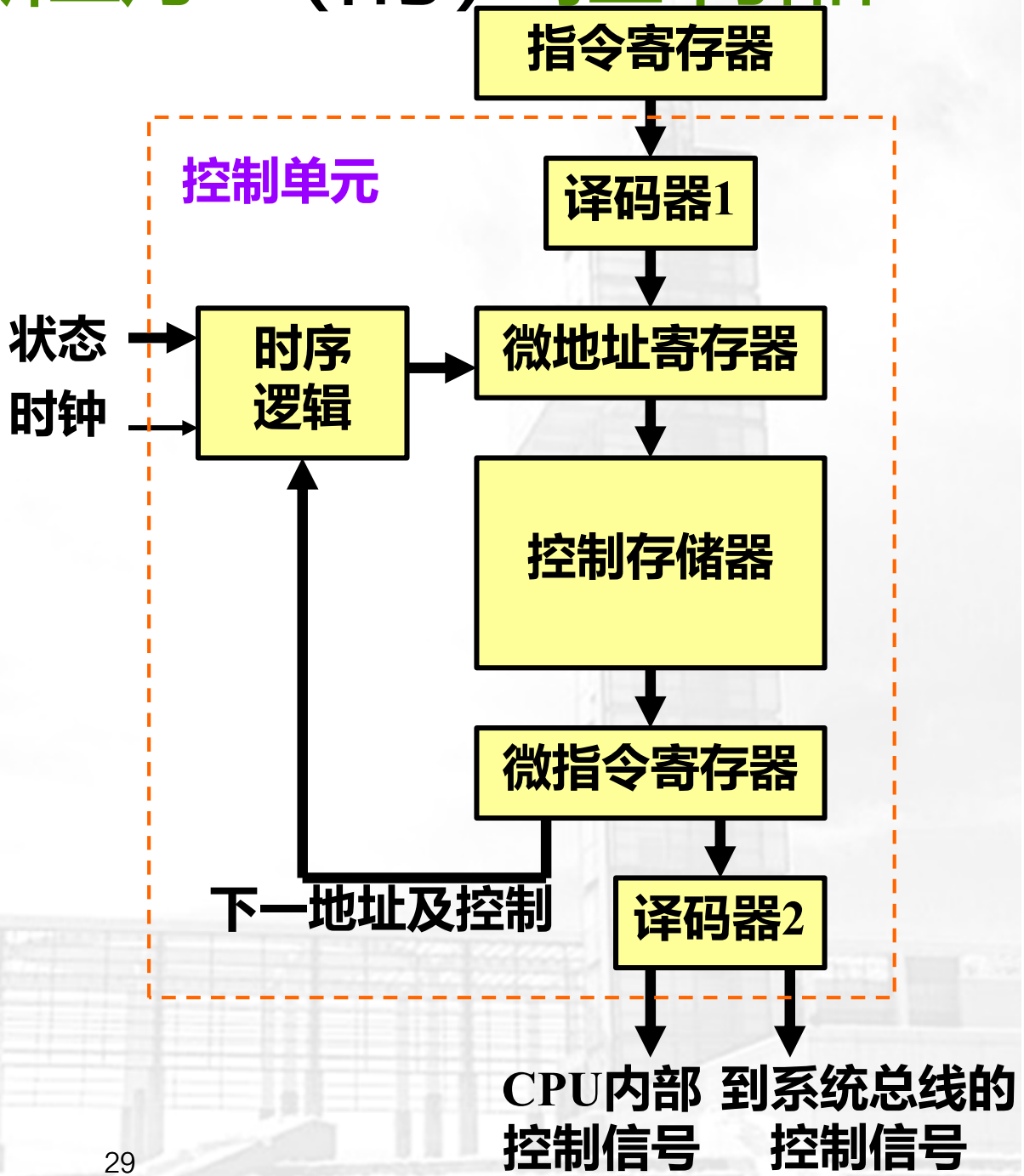
■ 微程序控制器**在一个时钟周期内完成如下工作：**

① **时序逻辑电路给控制存储器发出read命令；**

② **从微地址寄存器μAR指定的控存单元读出微指令，送入微指令寄存器μIR；**

③ **根据微指令寄存器的内容，产生控制信号，给时序逻辑提供下条微地址信息；**

④ **时序逻辑根据来自微指令寄存器的下条微地址信息和CPU内外状态，给微地址寄存器加载一个新的微地址。**



（基于）微程序（的）控制器

- 微指令的一般格式：
  - **地址域**：决定如何取得微指令
  - **控制域**：微指令的执行
- 设计微指令需要从两方面考虑：
  - 微指令的**长度** → 减少控制器占CPU集成芯片的面积
  - 微指令的**执行时间** → 提高CPU的工作速度

（基于）微程序（的）控制器：地址形成

暂时，先不去操心微指令的内容，先想想微指令的运行时顺序如何控制…

- **当机器指令在内存时…**
  - **指令寻址**
    - **默认（PC自加1）**
    - **段内/段间转移**
- **在CPU内部**
  - **有PC、IR、AR、DR四个寄存器用于定位和缓存指令**

- **当微指令在控制内存时，如何得到下一条微指令?**
- **微指令寻址!**

（基于）微程序（的）控制器：地址形成

- 下一条微指令的地址有三种可能：

① 由指令寄存器确定的微程序首地址：
每一个指令周期仅出现一次，且仅出现在刚刚获取一条指令之后。

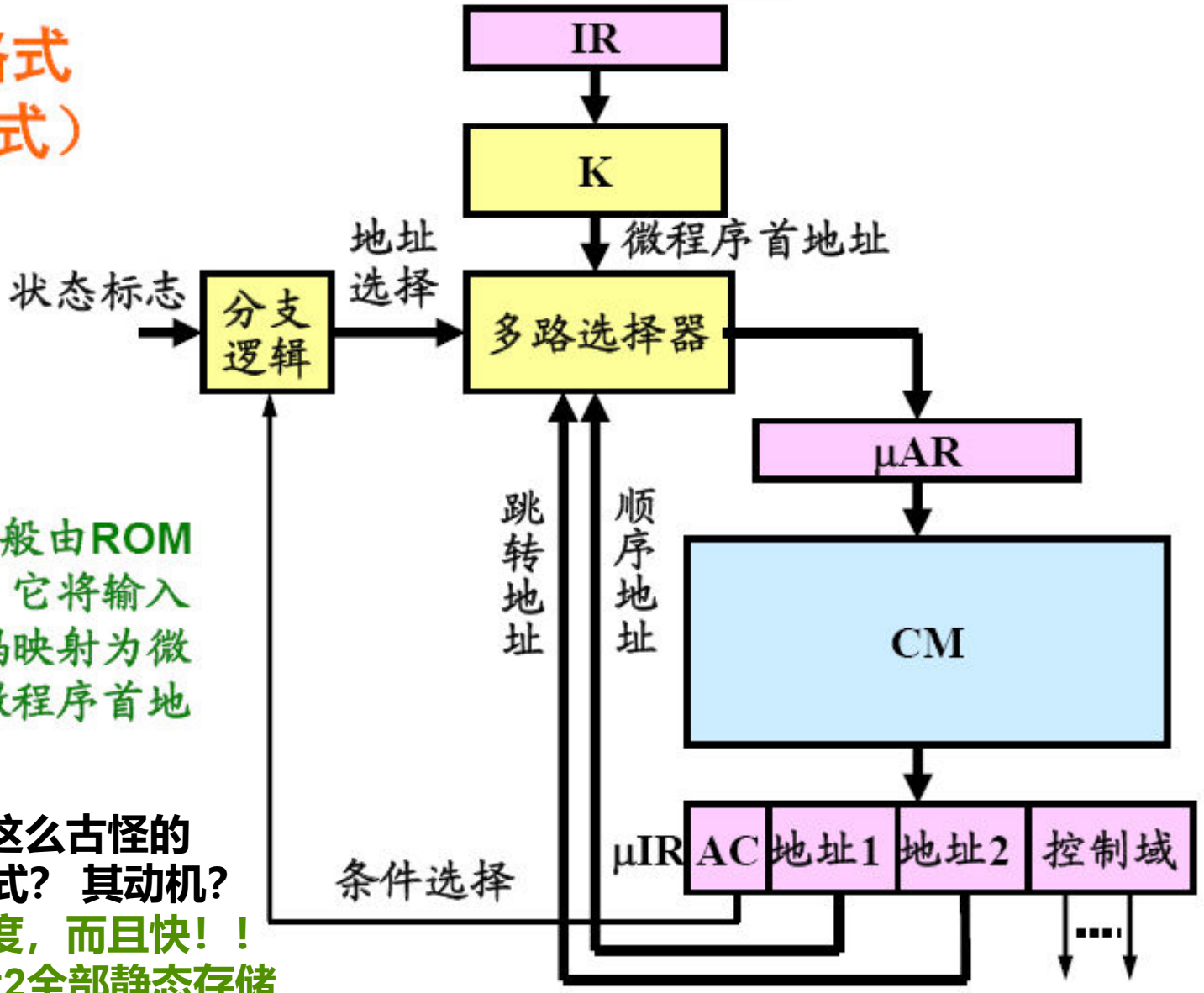
② 下一条顺序地址
下一条微指令地址＝当前微指令地址＋1

③ 分支跳转地址
  - 无条件和条件跳转
  - 两分支和多分支跳转

- 两地址格式（断定方式）
- 单地址格式（计数方式，增量方式）
- 可变格式

（基于）微程序（的）控制器：地址形成

1. 两地址格式（断定方式）



两地址格式的分支控制逻辑

逻辑电路K一般由ROM或PLA构成，它将输入的指令操作码映射为微指令地址（微程序首地址）并输出。

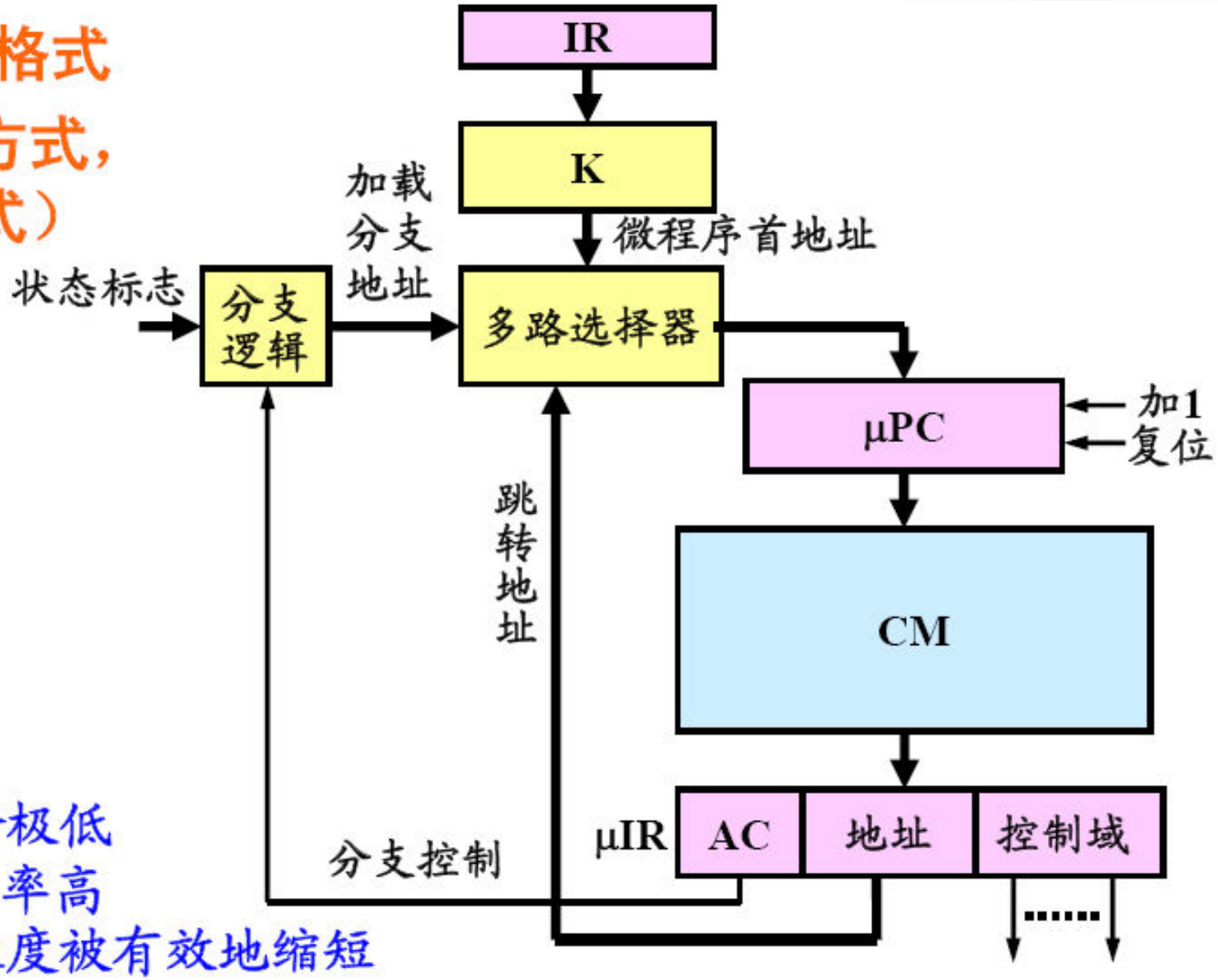
为什么会有这么古怪的Addressing方式？ 其动机？

降低控制器复杂度，而且快！！（AC/地址1/地址2全部静态存储，只有多路选择器一个逻辑部件；代价只是CM大一些而已）

（基于）微程序（的）控制器：地址形成

2. 单地址格式

（计数方式，增量方式）



- 硬件代价极低
- μPC利用率高
- 微指令长度被有效地缩短

单地址格式的分支控制逻辑

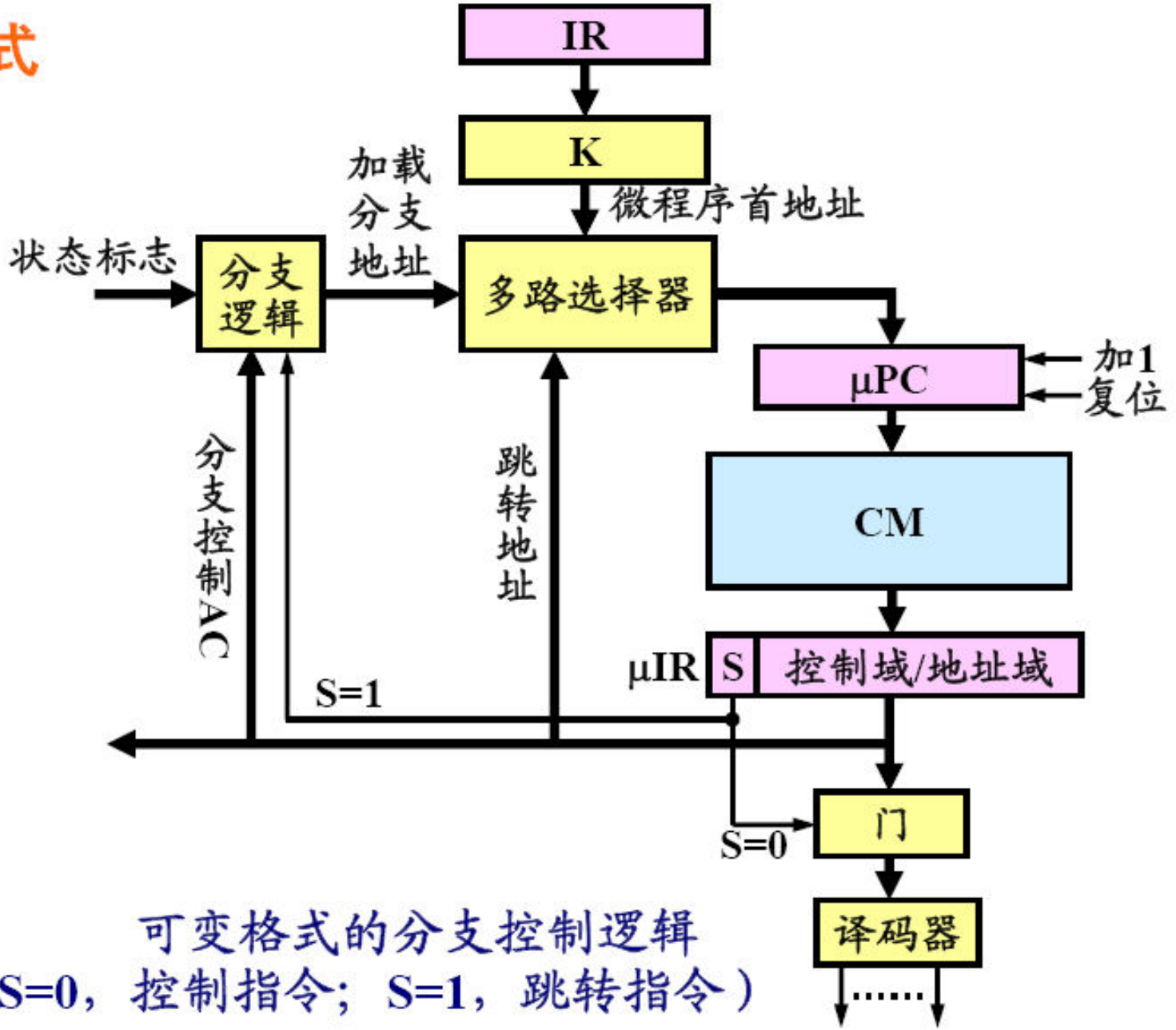
（基于）微程序（的）控制器：地址形成

3. 可变格式

- 使任何微指令执行时不存在无用信息：让微指令在顺序执行时只提供控制信号的产生，需要分支时再提供跳转地址。→ 可变格式微指令
- 两种微指令格式
  - 控制微指令
    S = 0
  - 转移微指令
    S = 1

（基于）微程序（的）控制器：地址形成

3. 可变格式



可变格式的分支控制逻辑
（设S=0，控制指令；S=1，跳转指令）

（基于）微程序（的）控制器：控制信号编码

- **水平型微指令（horizontal microinstruction）**

  多个控制信号同时有效 → 多个微操作同时发生。

- **垂直型微指令（vertical microinstruction）**

  类似于机器指令，利用微操作码的不同编码来表示不同的微操作功能。

（基于）微程序（的）控制器：控制信号编码

1. 水平型微指令控制域的编码

（1）直接表示法（水平编码）

- 可以在同一个时间有效的控制信号称为相容信号，具有相容性；
- 不能在同一个时间有效的控制信号称为互斥信号，具有互斥性。

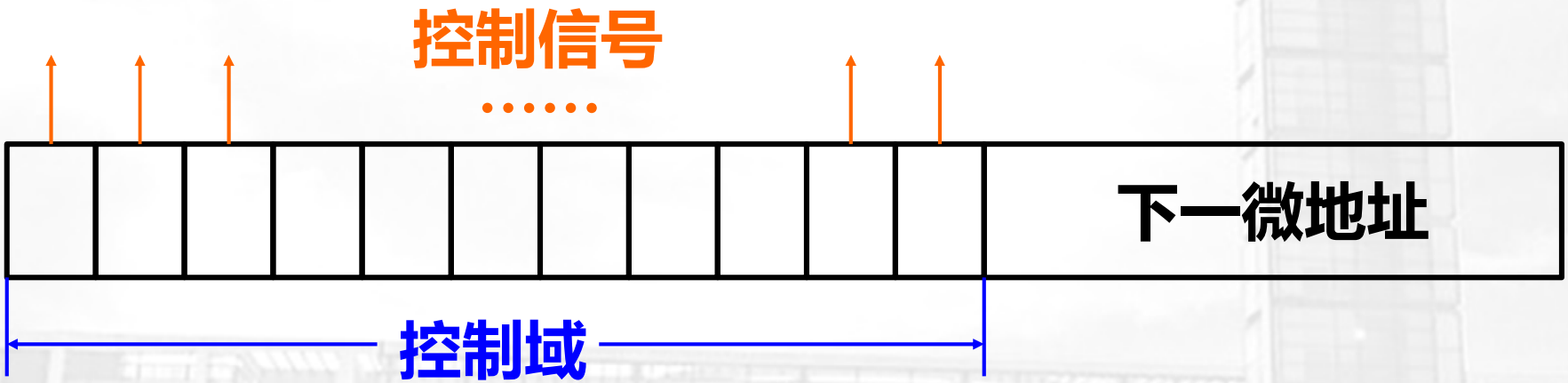


图6.15 直接表示法

（基于）微程序（的）控制器：控制信号编码

1. 水平型微指令控制域的编码

（2）字段译码法（字段编码）

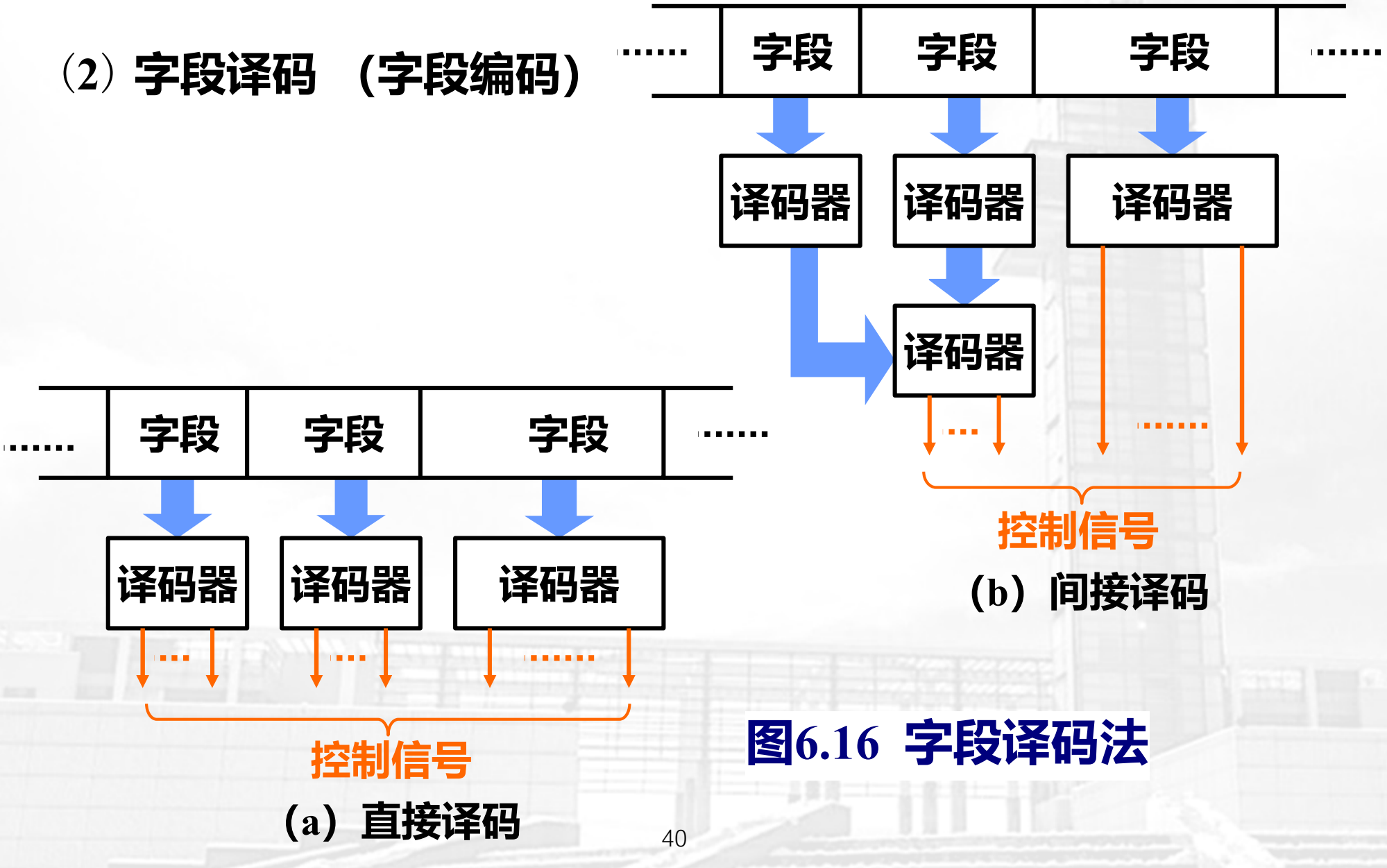
将控制域分为若干字段，字段内垂直编码，字段间水平编码

- ☞ **互斥的信号放在同一字段**
- ☞ **相容的信号放在不同字段**
- **若各字段的编码相互独立，则通过各字段独立译码就可以获得计算机系统的全部控制信号，这被称作直接译码方式。**
- **若某些字段的编码相互关联，则关联字段要通过两级译码才能获得相关的控制信号，这被称作间接译码方式。**

（基于）微程序（的）控制器：控制信号编码

1. 水平型微指令控制域的编码

（2）字段译码 （字段编码）



（b）间接译码

（a）直接译码

图6.16 字段译码法

（基于）微程序（的）控制器：控制信号编码

1. 水平型微指令控制域的编码

（2）字段译码法（字段编码）

- 每个字段中要设计一个无效控制信号的编码
- 若控制域的某字段有m位，则可以提供2^m-1个控制信号的编码
- 字段组织的有效方法：
  - 按**功能**组织：把功能类同的各控制信号放在同一字段中。
  - 按**资源**组织：把加载到同一部件上的各控制信号放在同一字段中。

（基于）微程序（的）控制器：控制信号编码

1. 水平型微指令控制域的编码

(2) 字段译码法（字段编码）

按功能	按功能	按资源	按资源	按功能	按资源	按资源	
字段1 (4位)	字段2 (4位)	字段3 (2位)	字段4 (3位)	字段5 (4位)	字段6 (2位)	字段7 (2位)	字段8
NOP 0000 $R0_{in}$ 0001 $R1_{in}$ 0010 …… $R7_{in}$ 1000 IR_{in} 1001 Y_{in} 1010 AR_{in} 1011 DRI_{in} 1100 DRS_{in} 1101	NOP 0000 $R0_{out}$ 0001 $R1_{out}$ 0010 …… $R7_{out}$ 1000 IR_{out} 1001 Z_{out} 1010 AR_{out} 1011 DRI_{out} 1100 DRS_{out} 1101	NOP 00 PC_{in} 01 PC_{out} 10 PC+1 11	NOP 000 SP_{in} 001 SP_{out} 010 SP+1 011 SP-1 100	NOP 0000 ADD 0001 SUB 0010 AND 0011 OR 0100 SHL 0101 SHR 0110 ROL 0111 ROR 1000	NOP 00 Mread 01 Mwrite 10	NOP 00 IOread 01 IOwrite 10	其他信号
*NOP为无效控制信号							

（基于）微程序（的）控制器：控制信号编码

1. 水平型微指令控制域的编码

按功能	按功能	按功能/资源	按资源	
字段1（4位）	字段2（4位）	字段3（4位）	字段4（3位）	字段5
NOP 0000	NOP 0000	NOP 0000	NOP 000	其他信号
$R0_{in}$ 0001	$R0_{out}$ 0001	ADD 0001	Mread 001	
$R1_{in}$ 0010	$R1_{out}$ 0010	SUB 0010	Mwrite 010	
……	……	AND 0011	IOread 011	
$R7_{in}$ 1000	$R7_{out}$ 1000	OR 0100	IOwrite 100	
IR_{in} 1001	IR_{out} 1001	SHL 0101		
Y_{in} 1010	Z_{out} 1010	SHR 0110		
AR_{in} 1011	AR_{out} 1011	ROL 0111		
DRI_{in} 1100	DRI_{out} 1100	ROR 1000		
DRS_{in} 1101	DRS_{out} 1101	PC+1 1001		
PC_{in} 1110	PC_{out} 1110	SP+1 1010		
SP_{in} 1111	SP_{out} 1111	SP-1 1011		

优化后的字段组织和编码

*NOP为无效控制信号

（基于）微程序（的）控制器：控制信号编码

1. 水平型微指令控制域的编码

（2）字段译码法（字段编码）

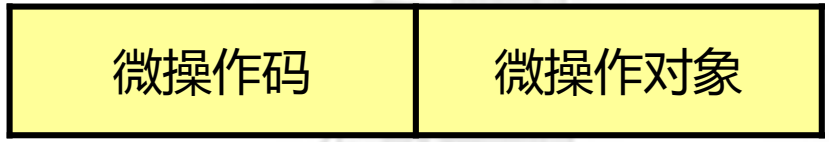
也可以对字段进行关联设计，使一个域用于解释另一个域。

表6.3 采用间接译码方式的字段编码

……	字段i（2位）	字段i+1（2位）	……
	NOP 00 算术 01 逻辑 10 移位 11	ADD 00 SUB 01 AND 00 OR 01 SHL 00 SHR 01 ROL 10 ROR 11	

（基于）微程序（的）控制器：控制信号编码

2. 垂直型微指令控制域的编码



- **采用与机器指令相似的格式**
  - **微操作码：指示作何种微操作**
    固定长度、可变长度
  - **微操作对象：**
    为微操作提供所需的操作数（常量或地址），一个、多个
- **特点：**
  - **控制域紧凑、短小**
  - **并行能力差，微程序长，执行速度减慢**
  - **可扩展性强**

（基于）微程序（的）控制器：控制信号编码

3. 水平型与垂直型微指令的比较

水平型微指令特性：

- 需要较长的微指令控制域；
- 可以表示高度并行的控制信号；
- 对控制域提供的控制信息只需较少的译码电路，甚至不需要译码。

垂直型微指令特性：

- 需要较短的微指令控制域；
- 并行微操作的表示能力有限；
- 对控制信息必须译码。

（基于）微程序（的）控制器

6、微程序控制器与组合逻辑（硬布线）控制器的比较

- 微程序控制器
  - 比硬布线控制器速度慢
  - 设计简单化、规范化
  - 功能可修改、可扩充
  - 实现成本低，出错概率小
  - 常用于CISC处理器控制器的实现
- 硬布线控制器
  - 速度快
  - 当计算机系统复杂时，设计困难
  - 一旦实现，不可修改和扩充
  - 常用于RISC处理器控制器的实现

CPU内部指令系统设计

- 假设我们正在造一个CPU
  - 假设我们正在研发核心的指令系统….

- 两大步骤：
  - 给指令编码（设计编码规则）
  - 给指令解码（让机器按规则去识别编码）

Designing a dummy CPU~

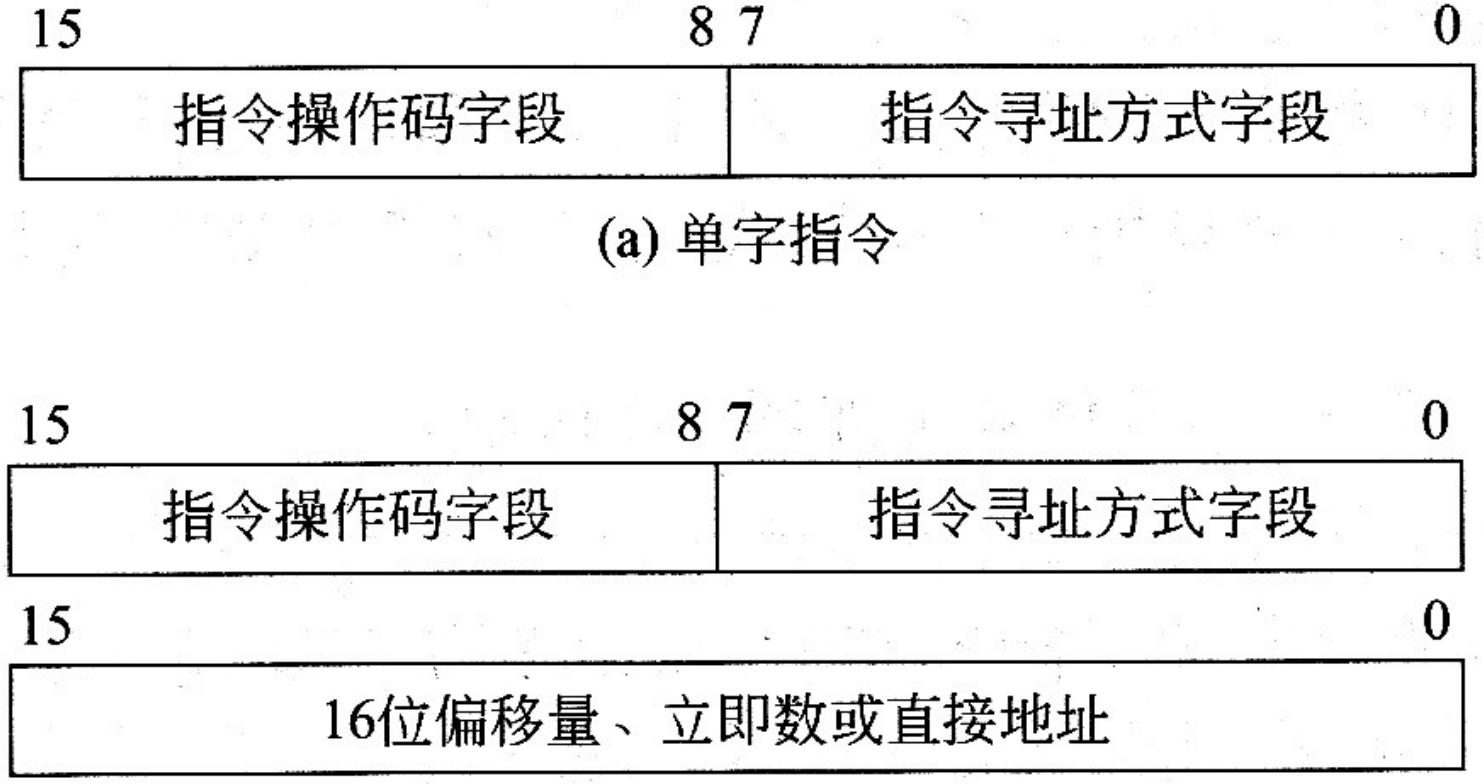


图 5-3　指令码构成格式

Designing a dummy CPU~

指令的设置

- 双操作数指令： 11条

  MOV、ADD、SUB、ADC、SBC、CMP、MUL、DIV、AND、OR、XOR

- 单操作数指令： 13条

  INC、DEC、PUSH、POP、NOT、SHL、SHR、SAR、ROL、ROR、RCL 、RCR、CALL

- 转移指令： 9条

  JMP、JZ、JNZ、JC、JNC、JG、JGE、JA、JAE

- 无操作数指令： 7条

  RET、IRET、NOP、CLI、STI、SWI、DAA

Designing a dummy CPU~

操作码字段的编排

Designing a dummy CPU~

- 双操作数指令： 11条

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

0	0	0	0	X	X	X	X	双操作数指令
0	0	0	0	0	0	0	0	MOV
				0	0	0	1	ADD
				0	0	1	0	SUB
				0	0	1	1	ADC
				0	1	0	0	SBC
				0	1	0	1	CMP
				0	1	1	0	MUL
				0	1	1	1	DIV
				1	0	0	0	AND
				1	0	0	1	OR
				1	0	1	0	XOR
				1	0	1	1	

Designing a dummy CPU~

- 单操作数指令： 13条

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	X	X	X	X	双操作数指令							
0	0	0	0	1	1	1	0	X	X	X	X	单操作数指令			
0	0	0	0	1	1	1	0	0	0	0	0	CALL			
								0	0	0	1	INC			
								0	0	1	0	DEC			
								0	0	1	1	PUSH			
								0	1	0	0	POP			
								0	1	0	1	NOT			
								0	1	1	0	SHL			
								0	1	1	1	SHR			
								1	0	0	0	SAR			
								1	0	0	1	ROL			
								1	0	1	0	ROR			
								1	0	1	1	RCL			
								1	1	0	0	RCR			
								1	1	0	1				
								1	1	1	0				
								1	1	1	1				

Designing a dummy CPU~

- 转移指令： 9条

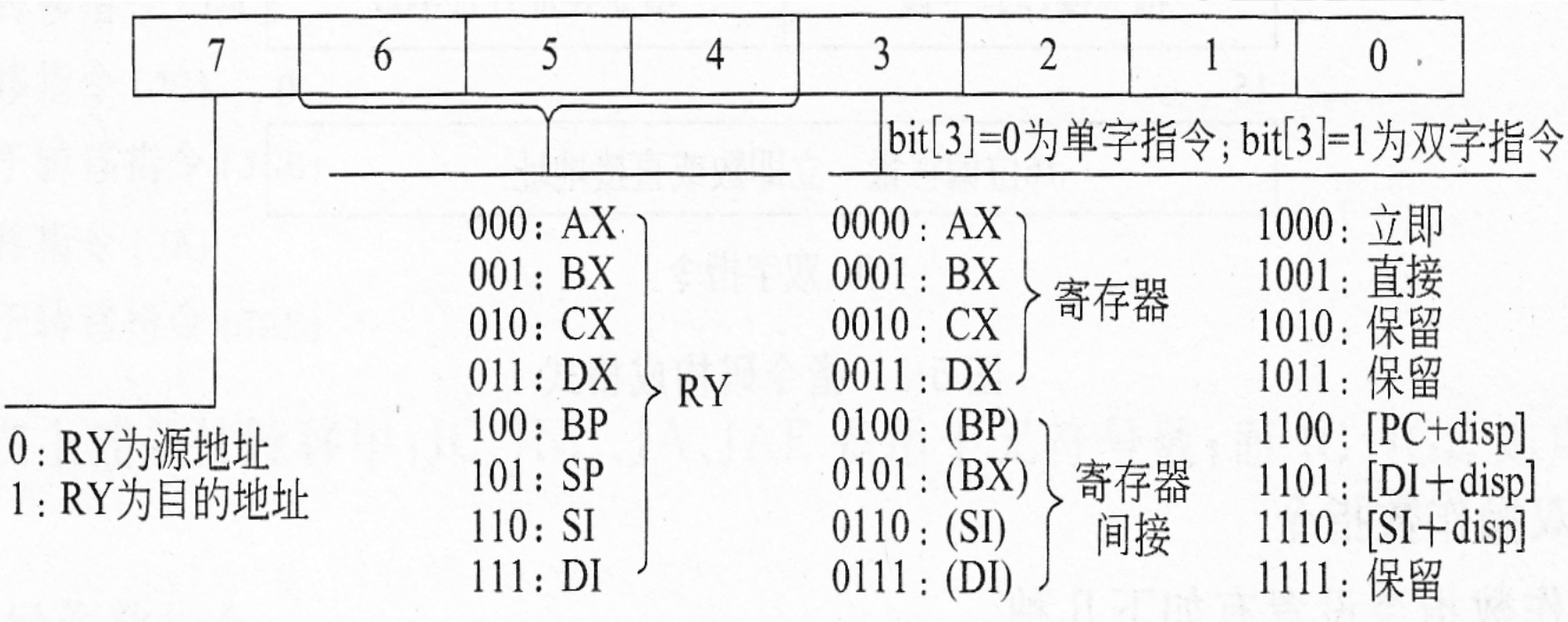
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	0	0	X	X	X	X									双操作数指令
0	0	0	0	1	1	1	0	X	X	X	X					单操作数指令
0	0	0	0	1	1	1	1	X	X	X	X					转移指令
0	0	0	0	1	1	1	1	0	0	0	0					JMP
								0	0	0	1					JZ
								0	0	1	0					JNZ
								0	0	1	1					JC
								0	1	0	0					JNC
								0	1	0	1					JG
								0	1	1	0					JGE
								0	1	1	1					JA
								1	0	0	0					JAE
								1	0	0	1					

Designing a dummy CPU~

- 无操作数指令： 7条

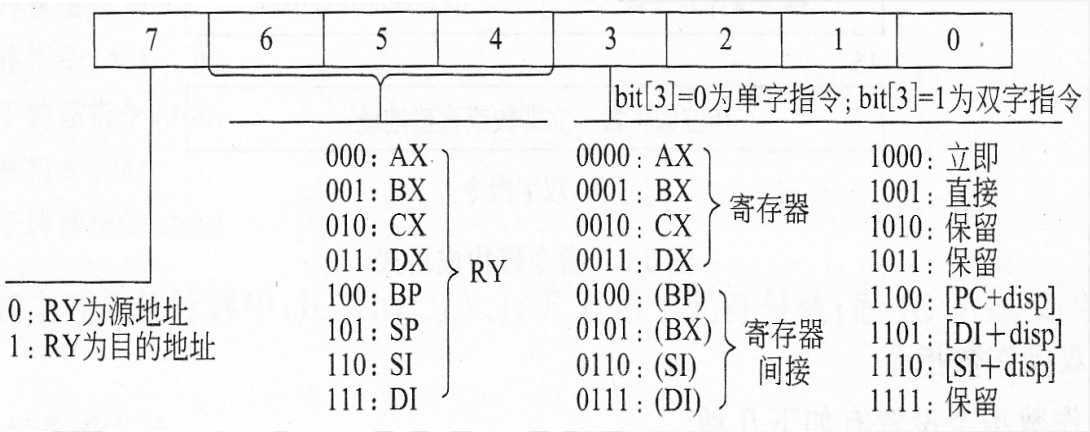
Designing a dummy CPU~

设计一个简单的寻址方式编码



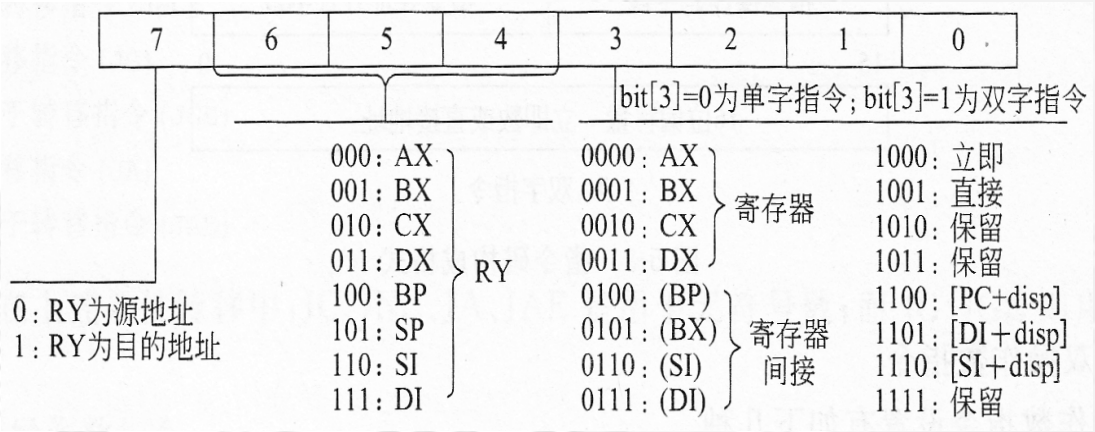
Designing a dummy CPU~

- 根据以上指令规范，我们可以为下面的指令给出操作码编码：
  - ADD AX [SI]
    - ?
  - Mov AX [2000H]
    - ?
  - INC [BX]
    - ?



Designing a dummy CPU~

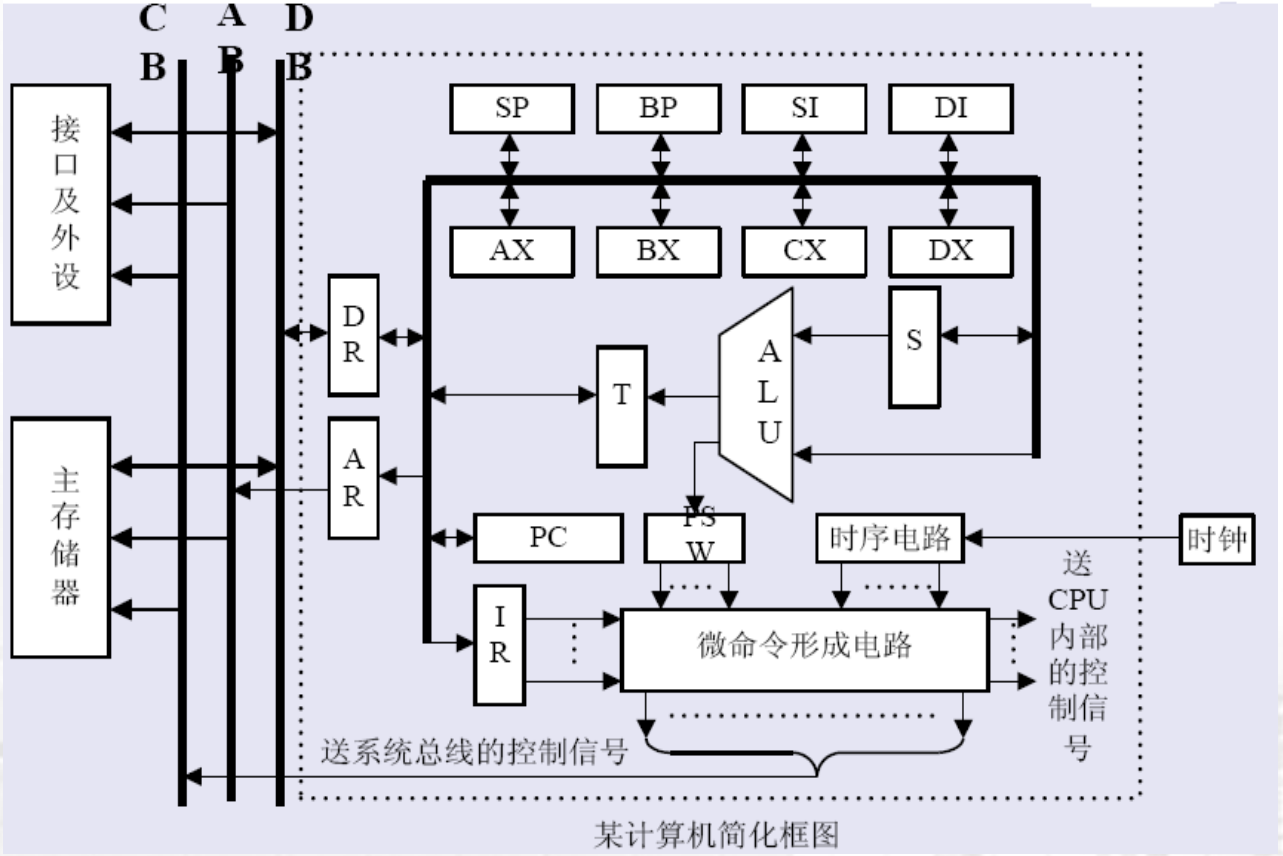
- 根据以上指令规范，我们可以为下面的指令给出操作码编码：
  - ADD AX [SI]
    - 01 86 (单字)
  - Mov AX [2000H]
    - 00 89 20 00 (双字)
  - INC [BX]
    - 0E 15 (单字)



Simulating a dummy CPU~

a) 取指令过程：

- 计算机简单框图



某计算机简化框图

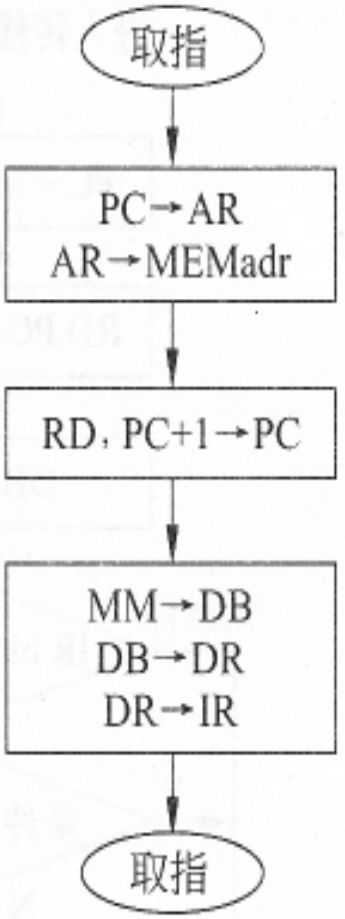
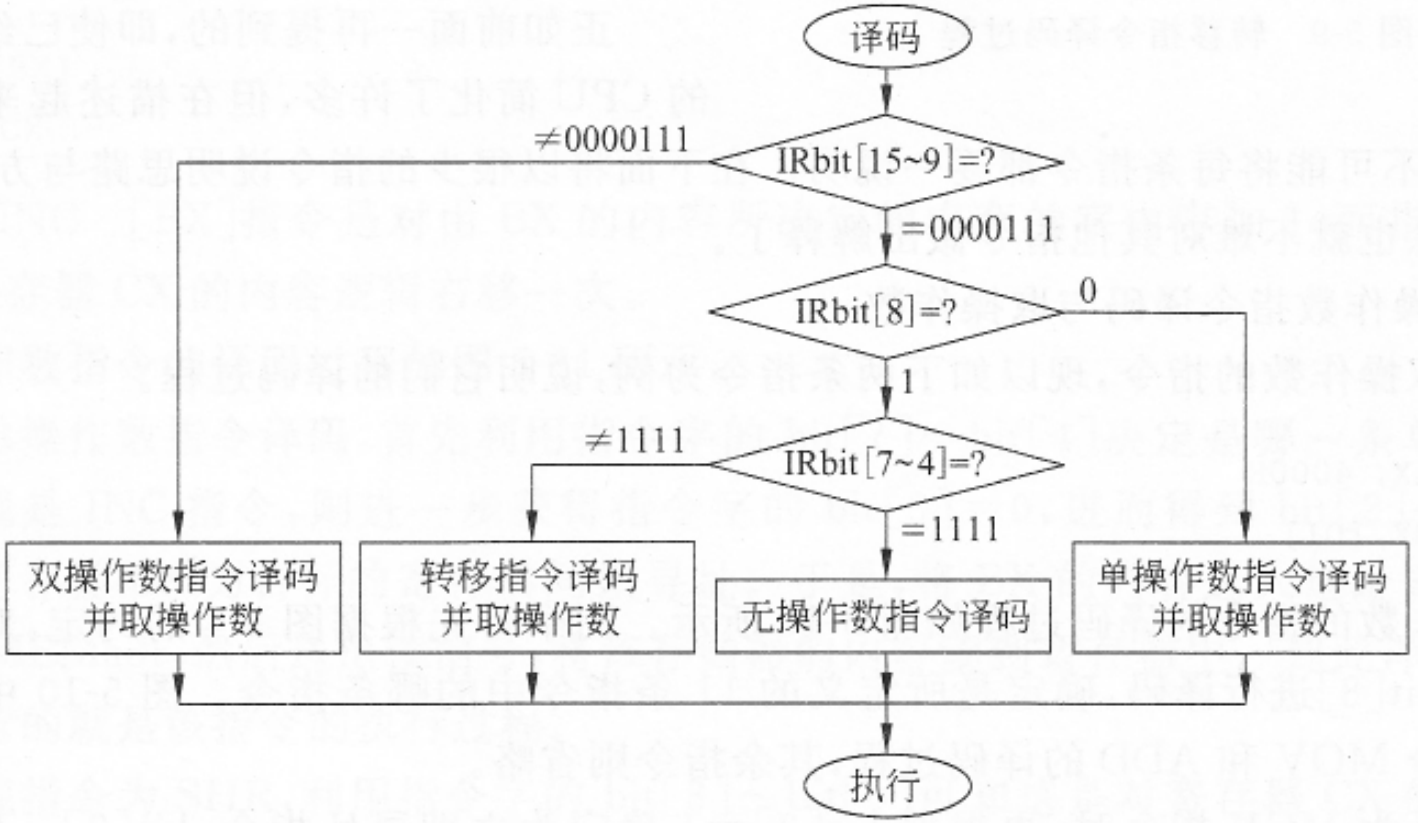


图 5-7 取指过程

Simulating a dummy CPU~

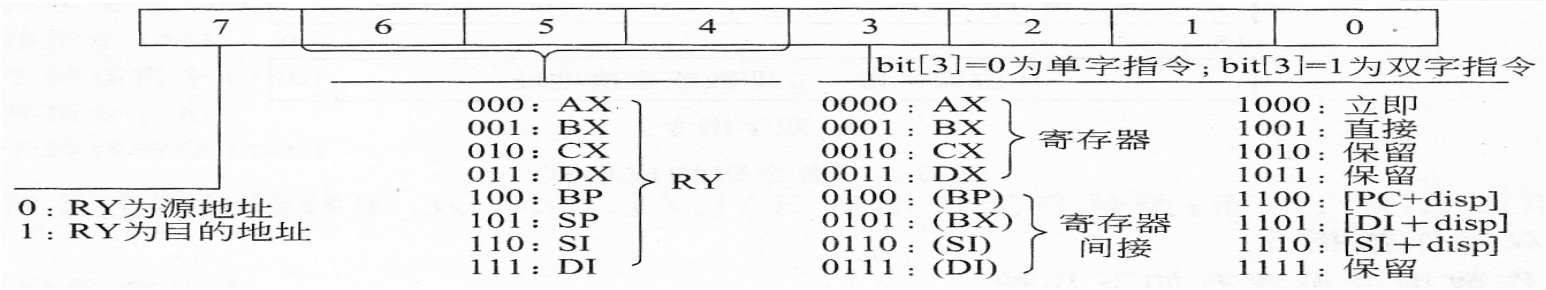
b) 译码取操作码过程：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	0	0	X	X	X	X	双操作数指令								
0	0	0	0	1	1	1	0	X	X	X	X	单操作数指令				
0	0	0	0	1	1	1	1	X	X	X	X	转移指令				
0	0	0	0	1	1	1	1	1	1	1	1	X	X	X	X	无操作数指令



Simulating a dummy CPU~

- 双操作数指令译码取操作数过程（续）：



- **ADD AX [SI]**
  - **01 86 (单字)**
- **Mov AX [2000H]**
  - **00 89 20 00 (双字)**
- **INC [BX]**
  - **0E 15 (单字)**

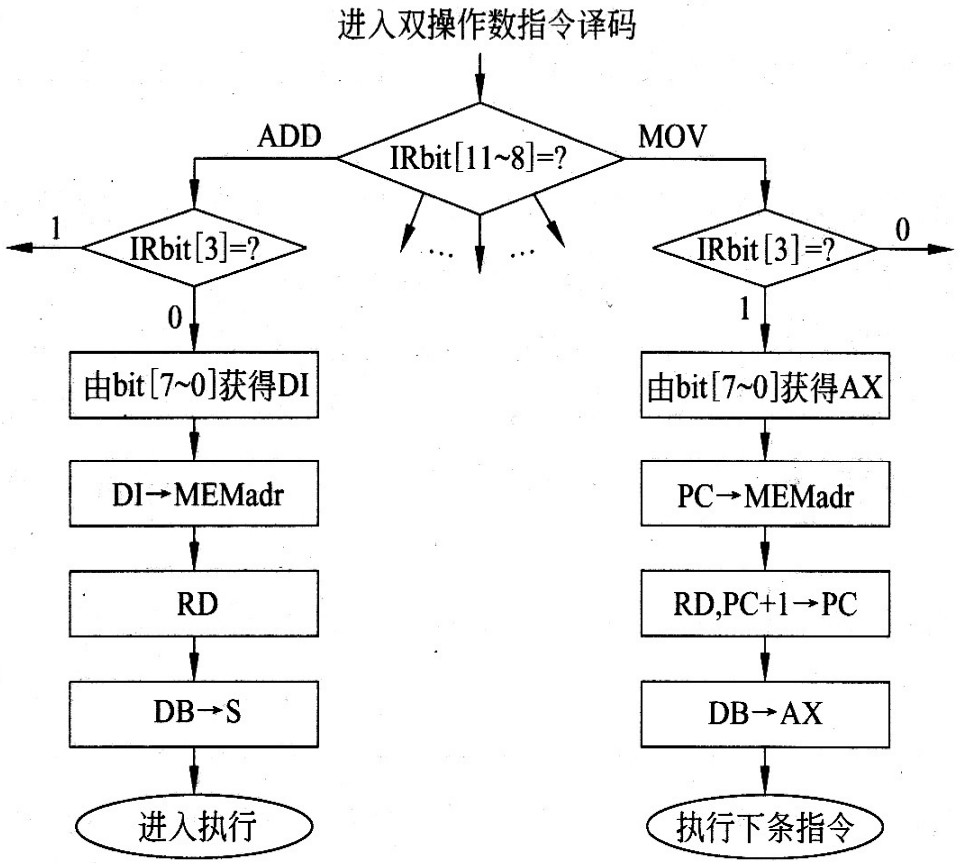


图 5-10　双操作数指令译码过程

Simulating a dummy CPU~

- 单操作数指令译码取操作数过程：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	0	0	1	1	1	0	X	X	X	X					单操作数指令
0	0	0	0	1	1	1	0	0	0	0	0					CALL
								0	0	0	1					INC
								0	0	1	0					DEC
								0	0	1	1					PUSH
								0	1	0	0					POP
								0	1	0	1					NOT
								0	1	1	0					SHL
								0	1	1	1					SHR
								1	0	0	0					SAR
								1	0	0	1					ROL
								1	0	1	0					ROR
								1	0	1	1					RCL
								1	1	0	0					RCR
								1	1	0	1					
								1	1	1	0					
								1	1	1	1					

- ADD AX [SI]
  - 01 86 (单字)
- Mov AX [2000H]
  - 00 89 40 00 (双字)
- INC [BX]
  - 0E 15 (单字)

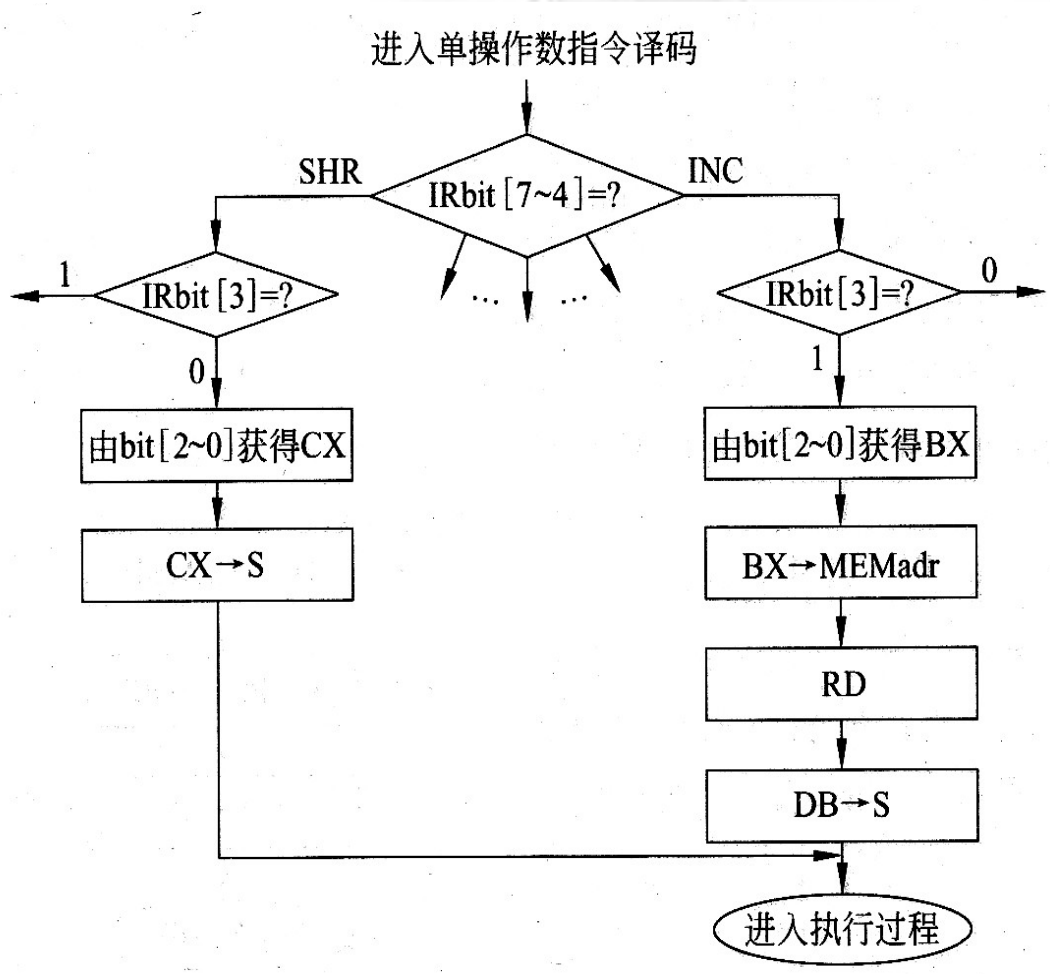


图 5-11　单操作数指令译码过程

Simulating a dummy CPU~

- 无操作数指令译码取操作数过程：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	0	0	1	1	1	1	1	1	1	1	X	X	X	X	无操作数指令
0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	RET
												0	0	0	1	IRET
												0	0	1	0	NOP
												0	0	1	1	CLI
												0	1	0	0	STI
												0	1	0	1	SWI
												0	1	1	0	DAA
												0	1	1	1	
												1	1	1	1	

IRET

0F F1

NOP

0F F2

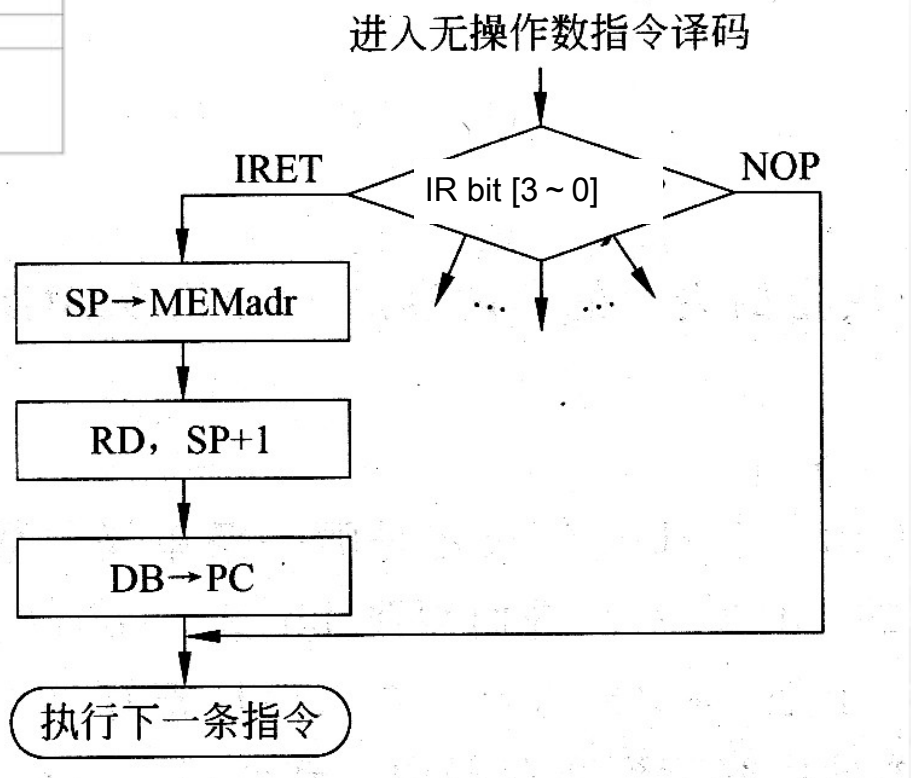
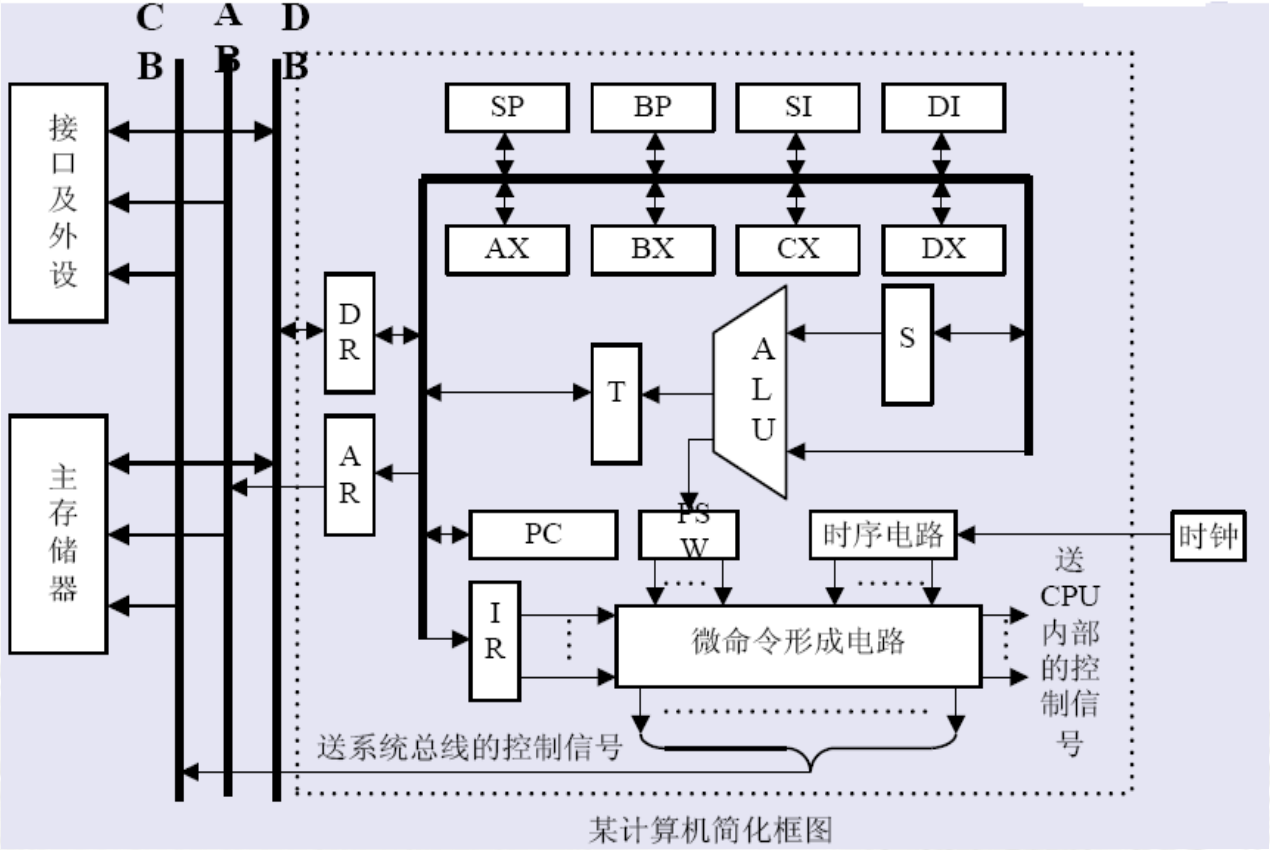


图 5-12　无操作数指令的译码过程

Simulating a dummy CPU~

c) 指令执行过程：

- 计算机简单框图



某计算机简化框图

JMP disp

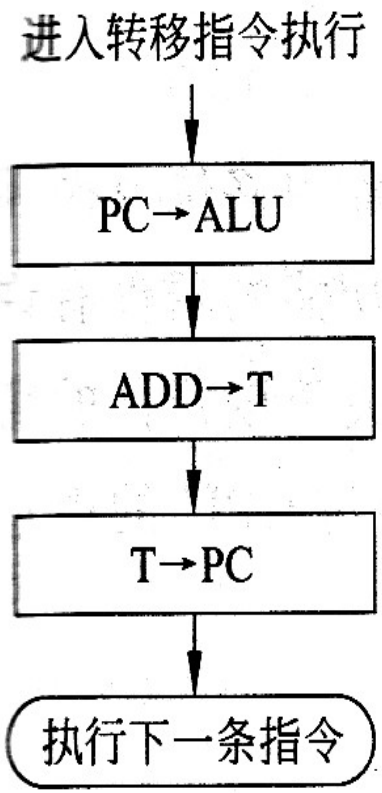


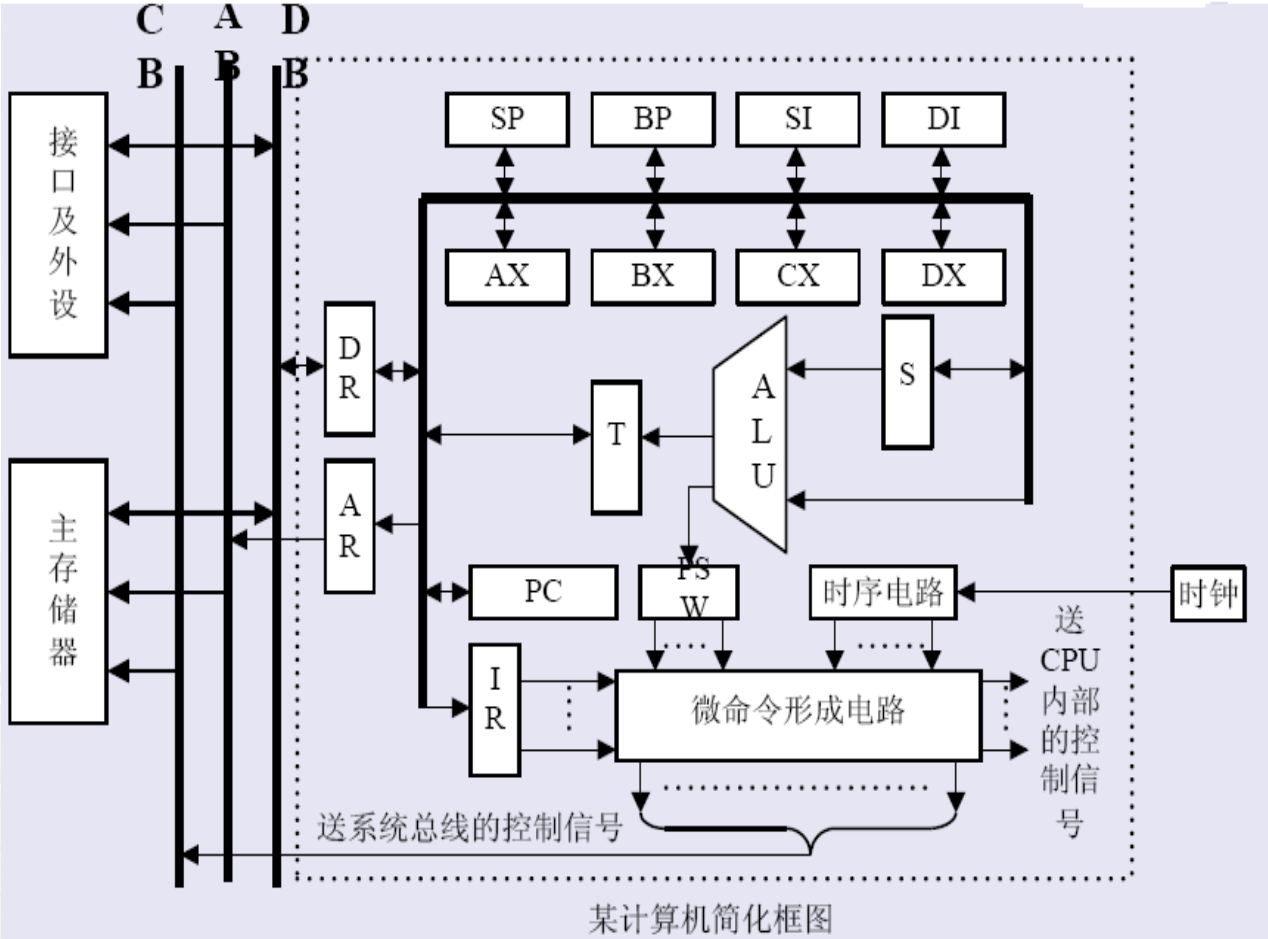
图 5-13 转移指令执行

Simulating a dummy CPU~

c) 指令执行过程(续):

ADD BX, [DI]

- 计算机简单框图



某计算机简化框图

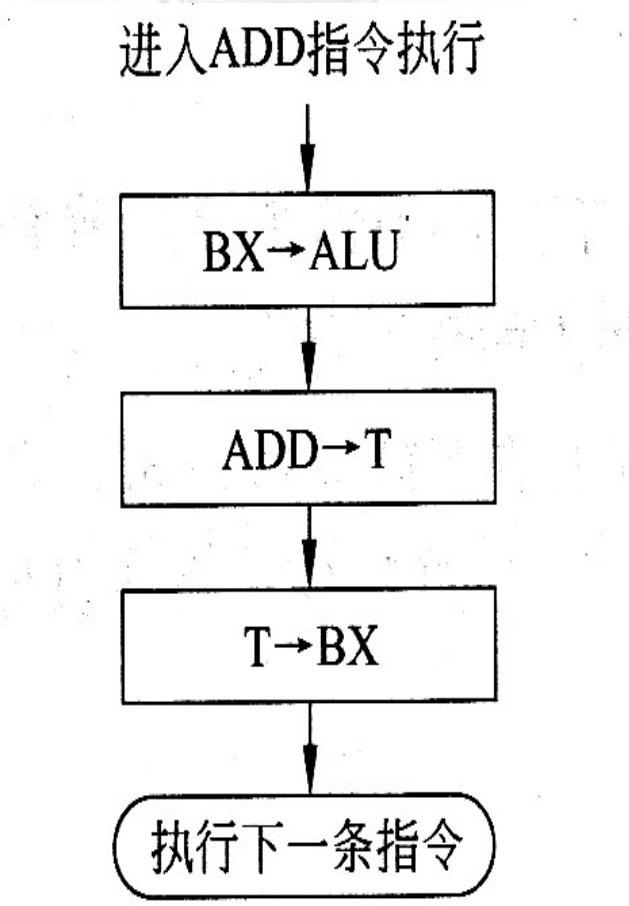
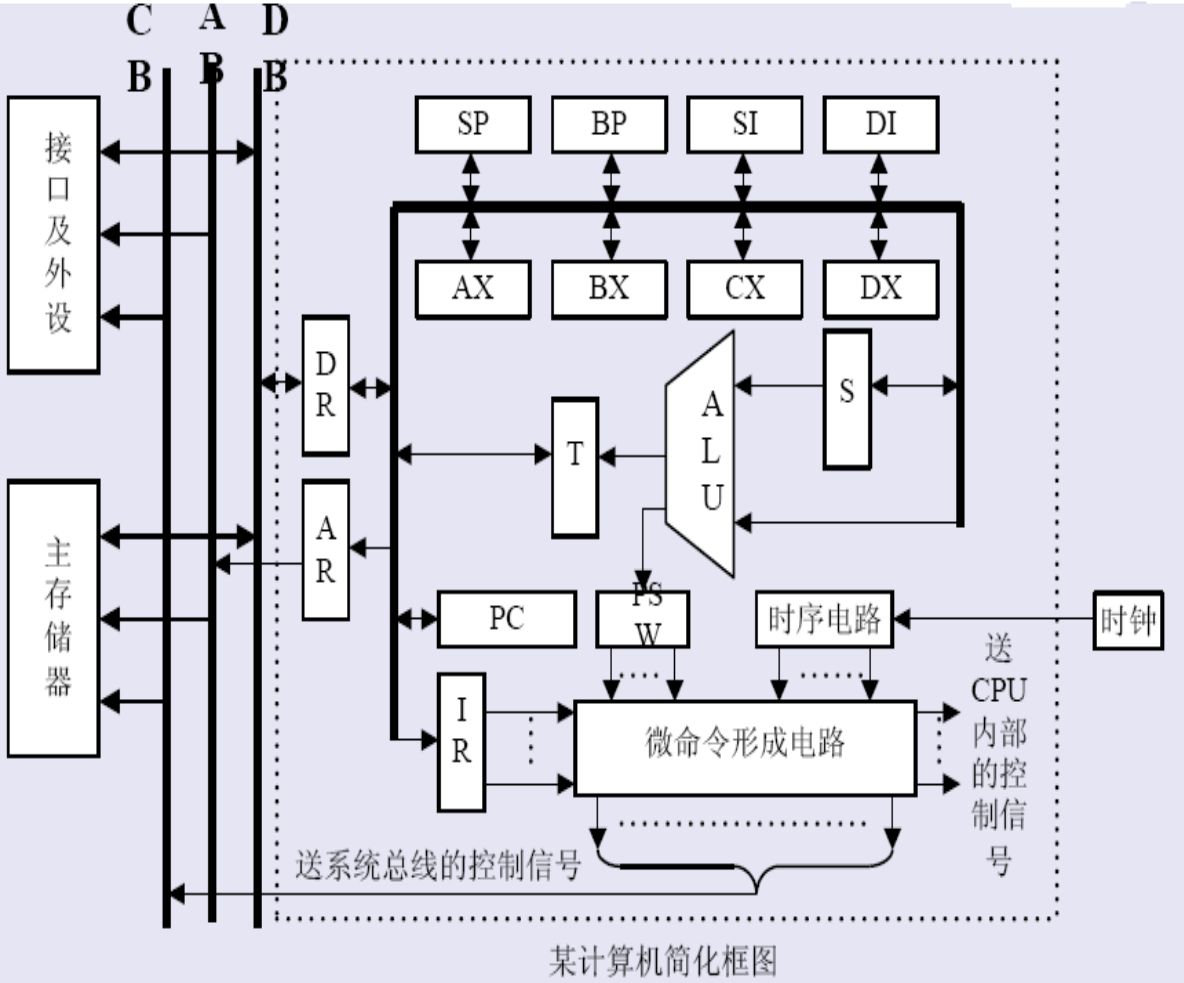


图 5-14 ADD 指令执行

Simulating a dummy CPU~

c) 指令执行过程(续):

- 计算机简单框图



某计算机简化框图

INC [BX]

SHR CX