西安电子科技大学
人工智能学院

计算机组成与体系结构

第6章 中央处理器(CPU)

本章第3次课重点

- 微程序控制思想与微指令
- 微程序控制器一般结构与工作原理
- 微指令地址域设计
  - ✓ 两地址格式
  - ✓ 单地址格式
  - ✓ 可变格式
- 微指令控制域设计
  - ✓ 直接表示法
  - ✓ 译码法
  - ✓ 字段译码法
- 微程序设计

微程序控制器设计

——微程序控制思想与微指令

6.3.1 微程序控制原理

一、微程序控制基本思想

- 指导思想：用软件方法组织和控制数据处理系统的信息传送，并最终用硬件实现。
- 基本思想：依据微程序顺序产生一条指令执行时所需的全部控制信号。
- 相当于把控制信号存储起来，因此又称存储控制逻辑方法。

6.3.1 微程序控制原理

二、微指令

- 对在一个时间单位（节拍）内出现的一组微操作进行描述的语句称作微指令（microinstruction）。
- 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，使一条指令中的所有微操作得以实现，从而实现一条指令的功能。
- 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）
　　……　　} 微指令1
T_j：微操作i（命令1，命令2，…）　　微指令j
　　……
T_m：微操作n（命令1）　　微指令m ⎫ 一个微程序

6.3.1 微程序控制原理

二、微指令

- 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CPU内部一个称为控制存储器（control memory）的ROM中。
- 在控制存储器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）→节拍周期。

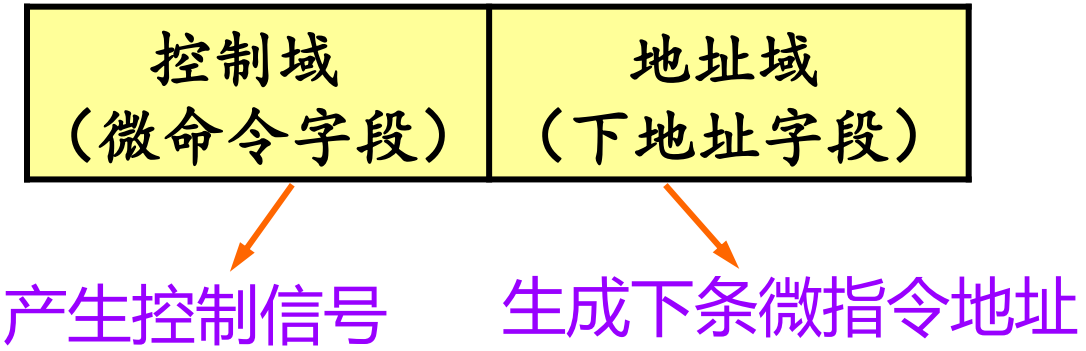
6.3.1 微程序控制原理

二、微指令

➢ 微指令格式

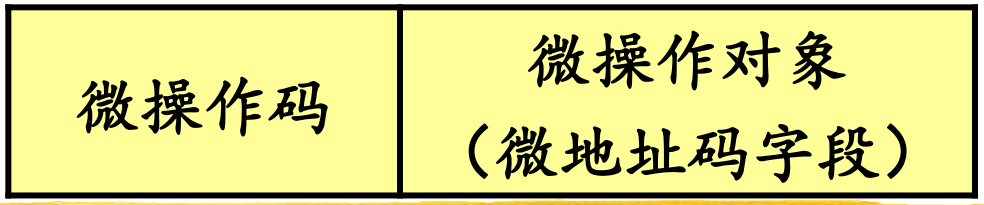
（1）水平型微指令（horizontal microinstruction）

多个控制信号同时有效 → 多个微操作同时发生。



（2）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。



微程序控制器设计

—— 一般结构与工作原理

6.3.1 微程序控制原理

三、微程序控制器的一般结构和工作原理

- 控制存储器（Control Memory,CM）
  - 微指令长度
  - 微程序占用的存储单元数
- 微指令寄存器μIR（控制缓冲寄存器）、微地址寄存器μAR
- 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。

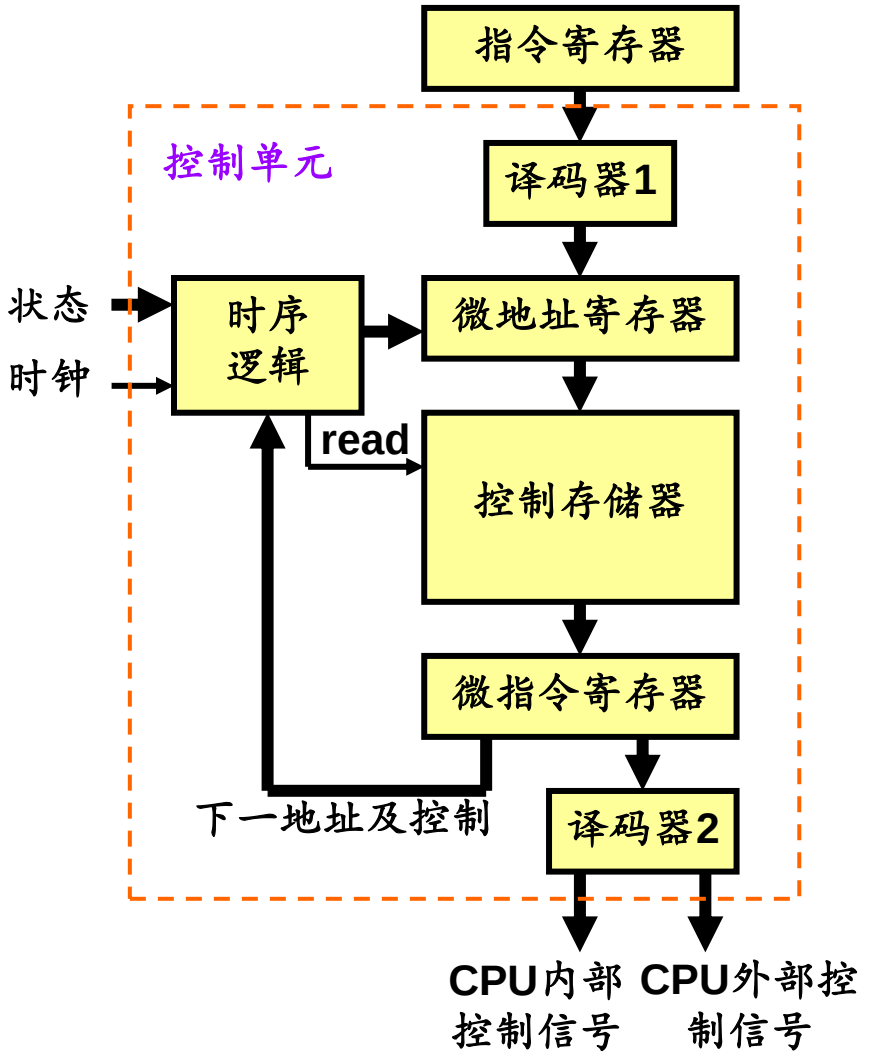


图6.10 微程序控制器的一般结构

6.3.1 微程序控制原理

三、微程序控制器的一般结构和工作原理

- 微程序控制器在一个时钟周期(节拍)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。

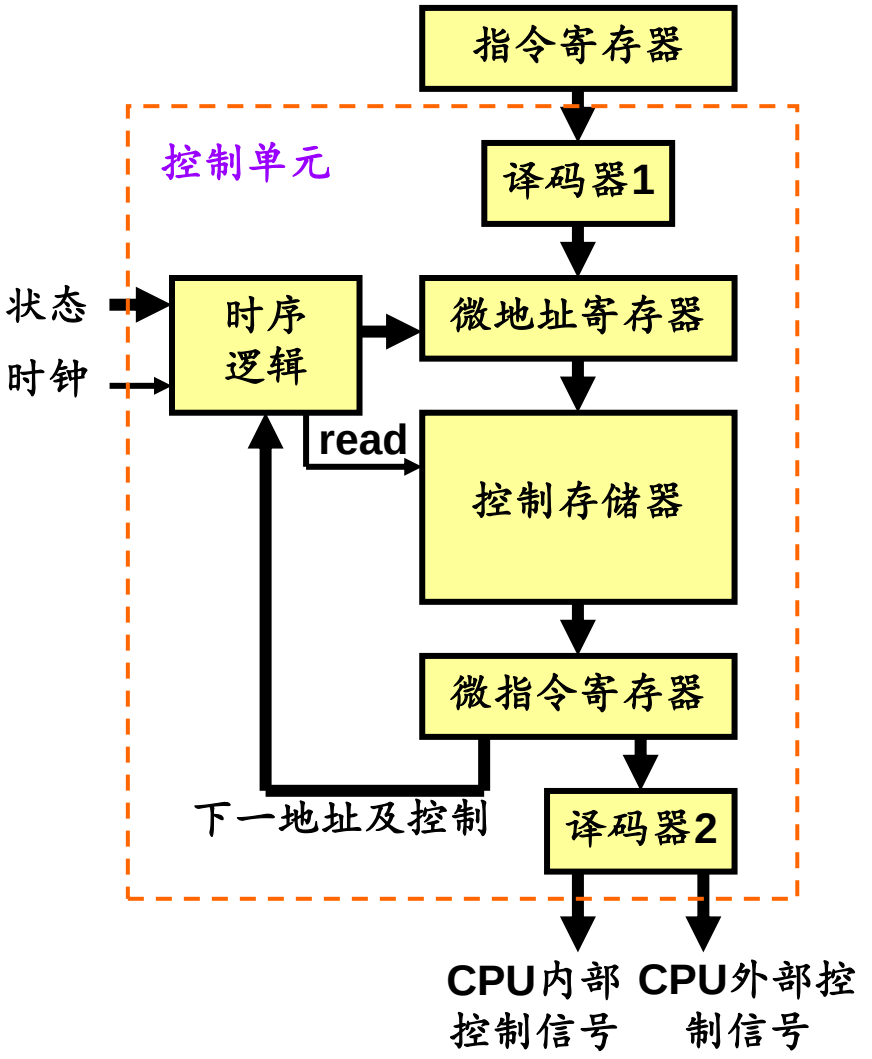


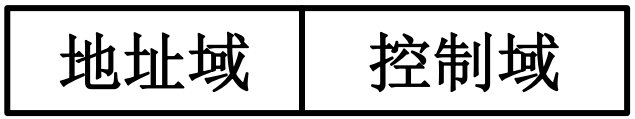
图6.10 微程序控制器的一般结构

微程序控制器设计

——微指令地址域设计与控制器结构

6.3.2 微指令设计

- 微指令设计影响微程序控制器硬件结构和微程序结构，是微程序控制器设计的核心
- 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
- 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



6.3.2 微指令设计　一、微指令地址的生成

➢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

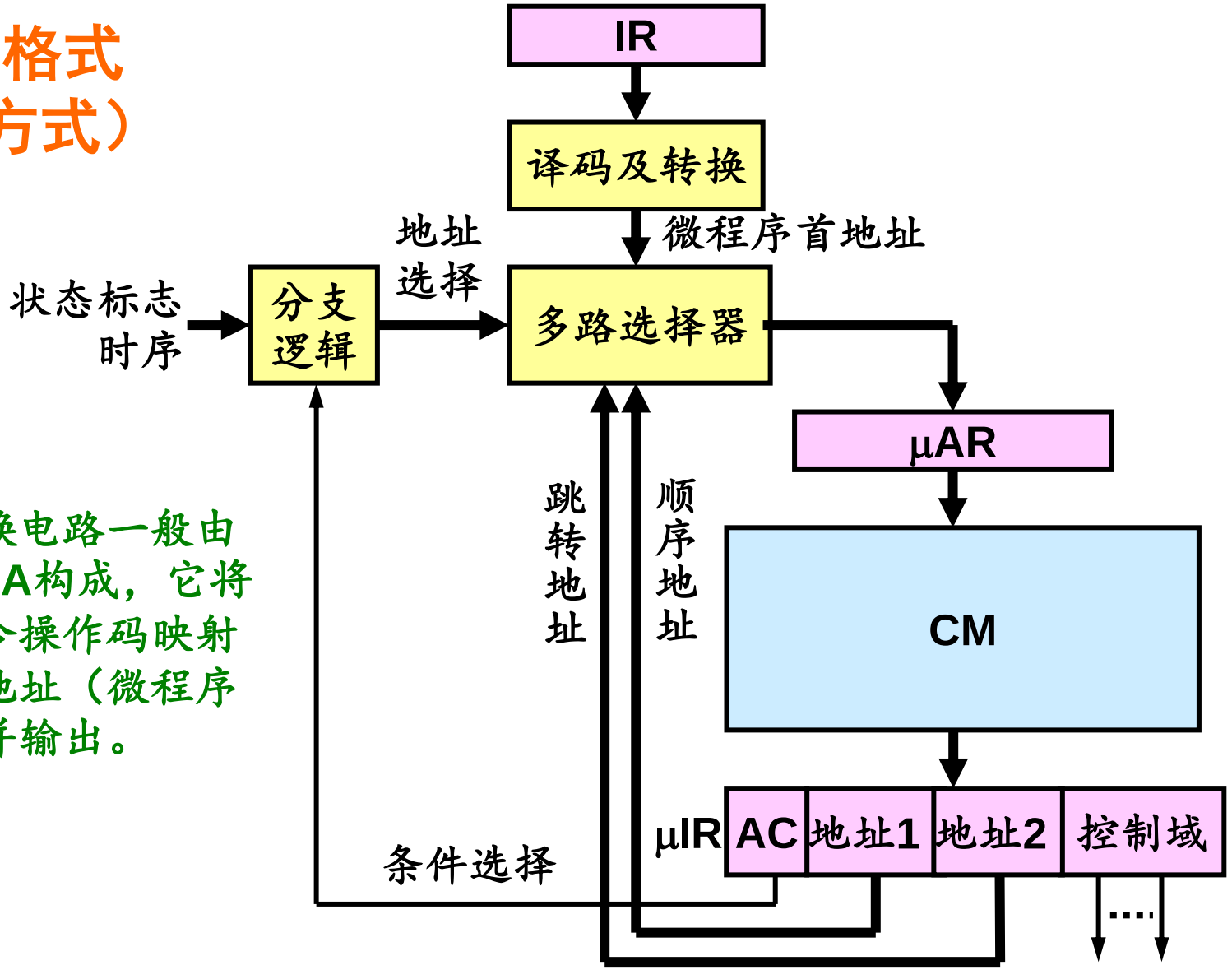
③分支跳转地址

- 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

6.3.2 微指令设计　一、微指令地址的生成

1. 两地址格式（断定方式）

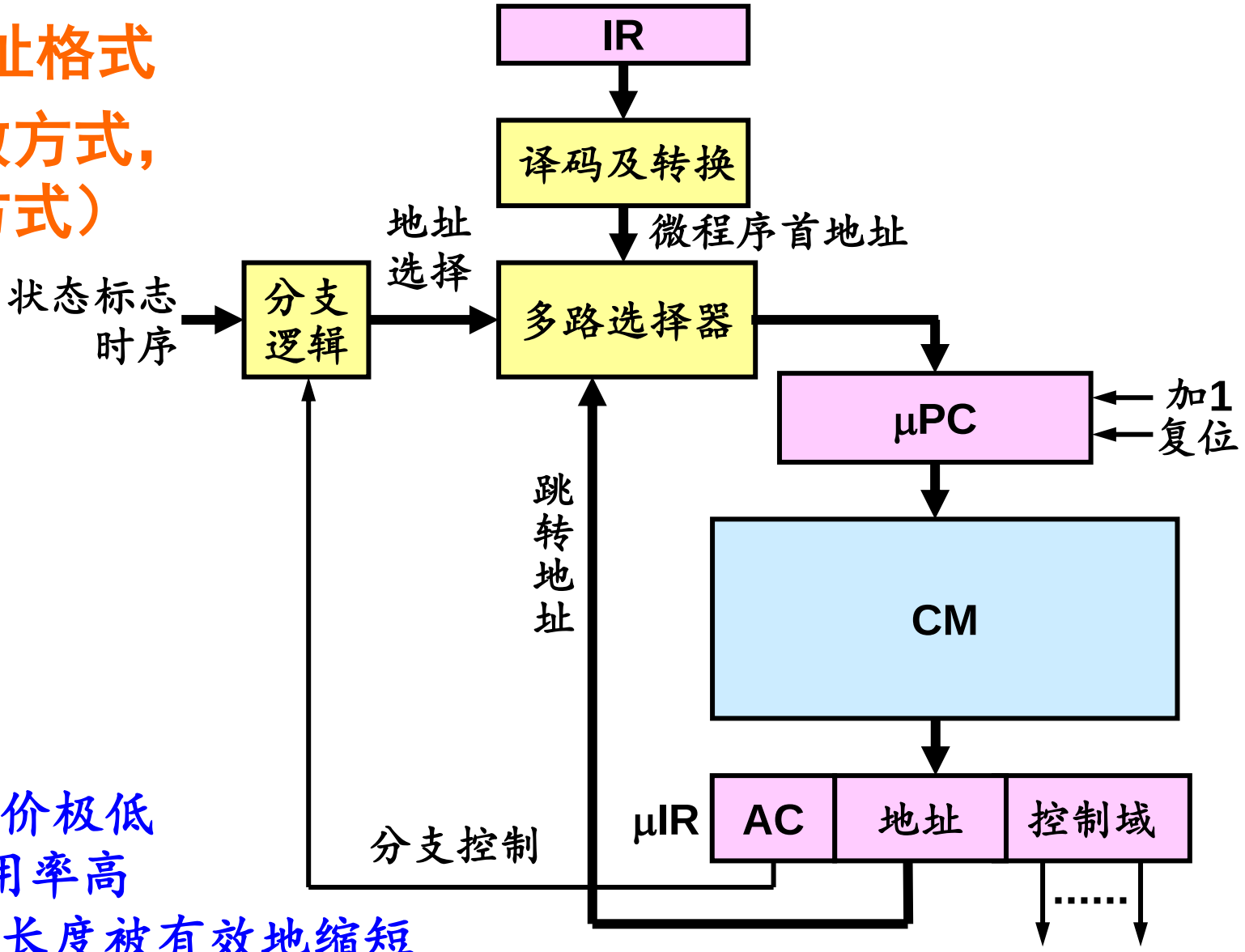


译码及转换电路一般由ROM或PLA构成，它将输入的指令操作码映射为微指令地址（微程序首地址）并输出。

图6.11　两地址格式的分支控制逻辑

6.3.2 微指令设计　一、微指令地址的生成

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

图6.12 单地址格式的分支控制逻辑

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

标识S	控制域

  - ✓ 转移微指令
    S＝1

标识S	分支控制	地址字段

- 控制存储器存储单元的位数L应设计为：

  L＝max{Lc，Lj}

  Lc ＝控制微指令长度，Lj＝转移微指令长度

3. 可变格式

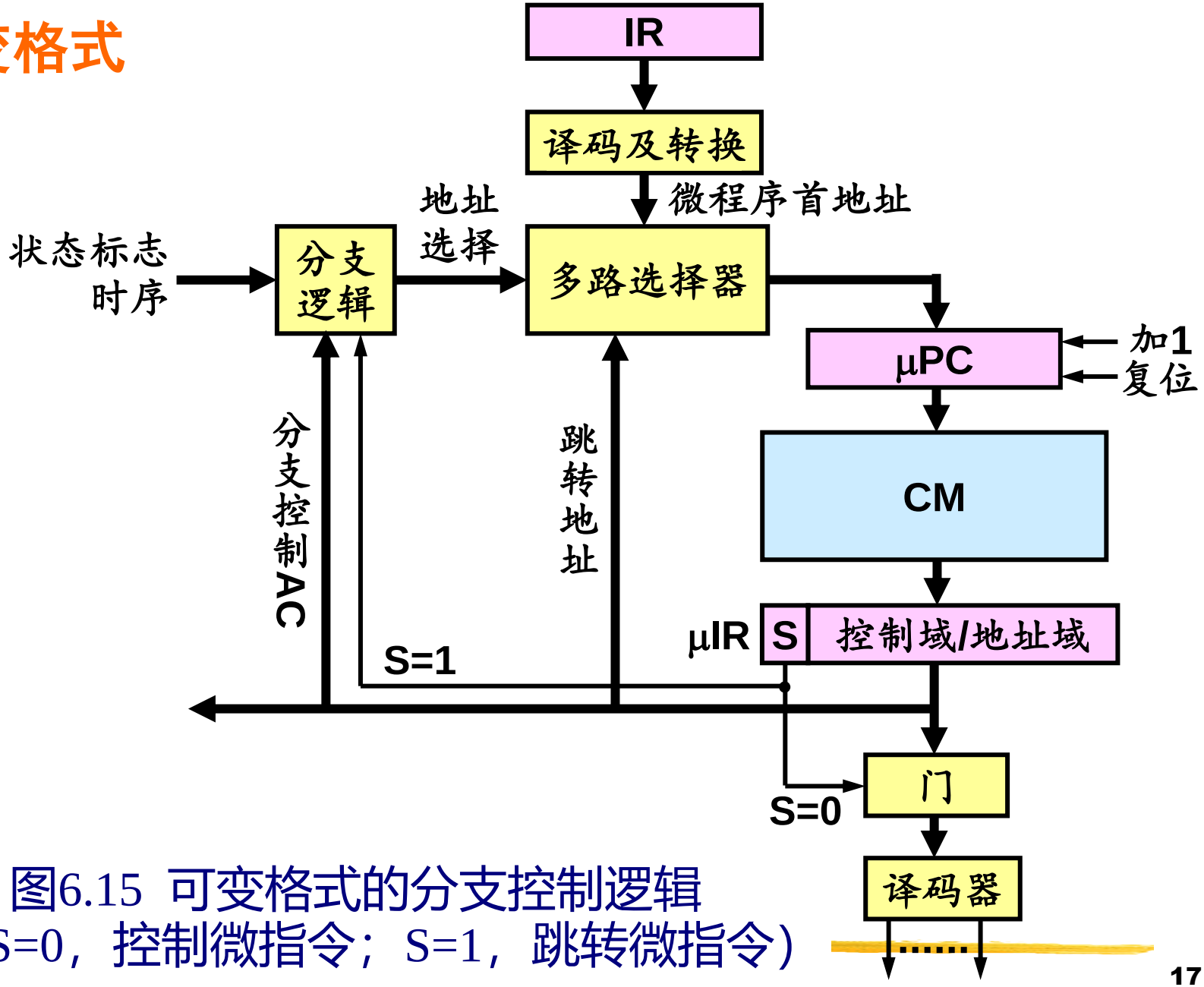


图6.15 可变格式的分支控制逻辑
（设S=0，控制微指令；S=1，跳转微指令）

4. 三种地址域格式的比较

- **两地址格式**
  - ✓ 分支逻辑较简单，下条微指令地址可以快速生成
  - ✓ 地址域较长，微指令较长，控存单元需要较多的位数
- **单地址格式**
  - ✓ 减少了指令的长度，控制存储器的容量大为减小
  - ✓ 微程序计数器加1的速度决定了顺序地址产生的时间
- **可变格式**
  - ✓ 长度最短，要求控存单元的位数最少
  - ✓ 专用的跳转微指令：微程序的长度增加，控存单元数量增加，机器指令执行时间增长
  - ✓ 下条微指令地址的生成时间与单地址格式基本一致

随堂练习

某控制器中的CM在微指令格式采用两地址时，容量为2K×36b。若微指令格式改用一地址或可变格式时，CM单元数量和微指令格式中的控制域位数保持不变，请填写下表中各字段的位数。

两地址微指令

控制域	AC	地址1	地址2
[填空1]	1	[填空2]	[填空3]

一地址微指令

控制域	AC	地址1
[填空4]	1	[填空5]

可变格式微指令

标识S	控制/地址域
1	[填空6]

答案解析

CM在微指令格式采用两地址时，容量为2K×36b，则微指令长度=36b

CM单元数量=2K，则地址位数=11b。

两地址微指令

控制域	AC	地址1	地址2
13	1	11	11

一地址微指令

控制域	AC	地址1
13	1	11

可变格式微指令

标识S	控制/地址域
1	13

微程序控制器设计

——微指令控制域设计

二、微指令控制域编码

- 对微指令控制域采用不同的设计方法，微指令就有不同的分类方法。一种较通用的分类方法是根据产生控制信号的方式将微指令分为
- 水平型微指令(Horizontal Microinstruction)
  - ✓ 水平型微指令可以使多个控制信号同时有效，达到使多个微操作同时发生的效果。
- 垂直型微指令(Vertical Microinstruction)
  - ✓ 垂直型微指令类似于机器指令，通常一条微指令实现一个微操作。

1. 水平型微指令控制域的编码

地址域	控制域

(1) 直接表示法

- ✓ 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。

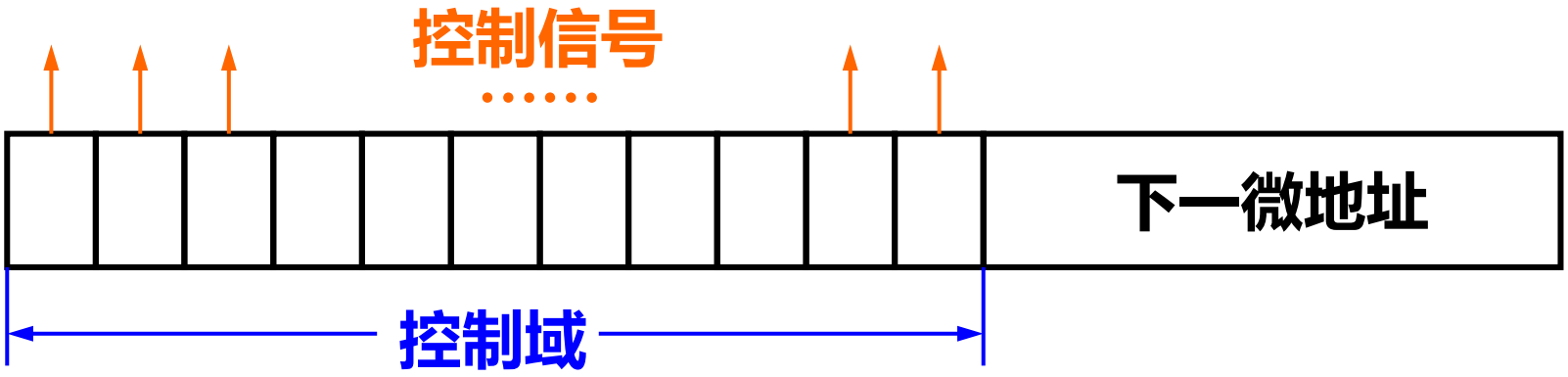


图6.16 直接表示法

1. 水平型微指令控制域的编码

(2) 译码法

- ✓ 采用编码的方法表示控制信号。
- ✓ 可以极大地缩短微指令控制域的长度。
- ✓ 各控制信号需要通过不同的微指令在不同时间来产生，所以各控制信号是相斥的，这也被称为垂直编码。
- ✓ 不能实现一个节拍提供多个控制信号的任务，从而使指令周期的节拍数增多，微程序中包含的微指令数量增多，（机器）指令执行时间增长。

例如：系统中有200个控制信号，用直接表示法，控制域位数=200；用垂直编码法，控制域位数=8

1. 水平型微指令控制域的编码

(3) 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。

1. 水平型微指令控制域的编码

(3) 字段译码法
（字段编码）

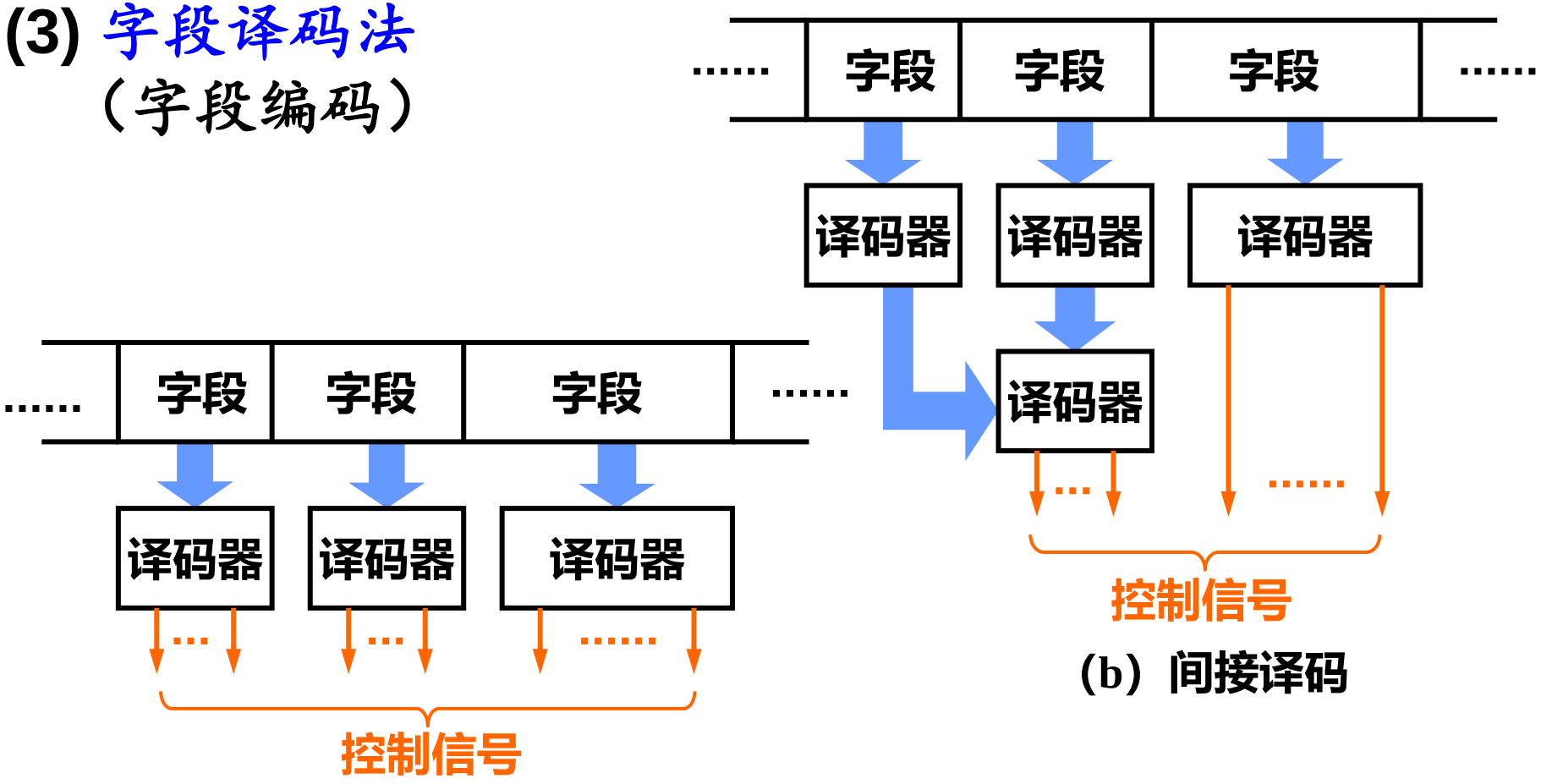


图6.17 字段译码法

1. 水平型微指令控制域的编码

(3) 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。

1. 水平型微指令控制域的编码

图6.3

图6.4

(3) 字段译码法（字段编码）

按功能	按功能	按资源	按资源	按功能/资源	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	NOP 0000	NOP 00	NOP 000	NOP 0000	NOP 00	NOP 00	其他信号
$R0_{in}$ 0001	$R0_{out}$ 0001	PC_{in} 01	SP_{in} 001	ADD 0001	Mread 01	IOread 01	
$R1_{in}$ 0010	$R1_{out}$ 0010	PC_{out} 10	SP_{out} 010	SUB 0010	Mwrite 10	IOwrite 10	
……	……	PC+1 11	SP+1 011	AND 0011			
$R7_{in}$ 1000	$R7_{out}$ 1000		SP-1 100	OR 0100			
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						

*NOP为无效控制信号

表6.3 一种控制域字段的组织和编码

1. 水平型微指令控制域的编码

图6.3

图6.4

(3) 字段译码法（字段编码）

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	PC_{out}	1110	PC+1	1001			
PC_{in}	1110	SP_{out}	1111	SP+1	1010			
SP_{in}	1111			SP-1	1011			
				DRS_{out}	1100			

表6.4 优化后的字段组织和编码

*NOP为无效控制信号

1. 水平型微指令控制域的编码

(3) 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6.5 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	

2. 垂直型微指令控制域的编码

微操作码	微操作数

- 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

  改进：在计算机系统中大量引入并行机制，使得少量的控制信号就可以引起较多的微操作同时完成，例：三总线结构的ALU

微指令设计——微指令控制域编码

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。

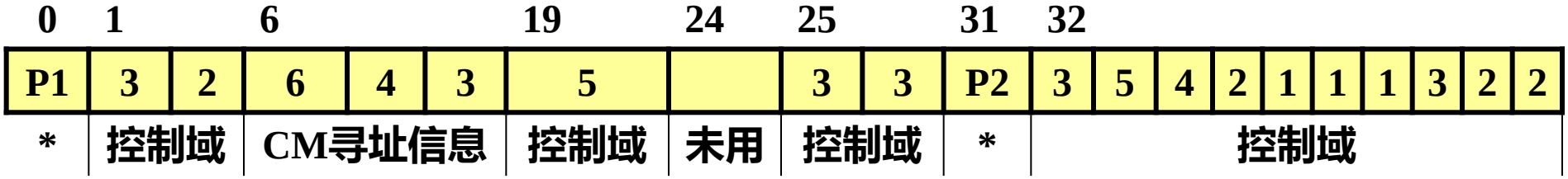
速度快

- **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。

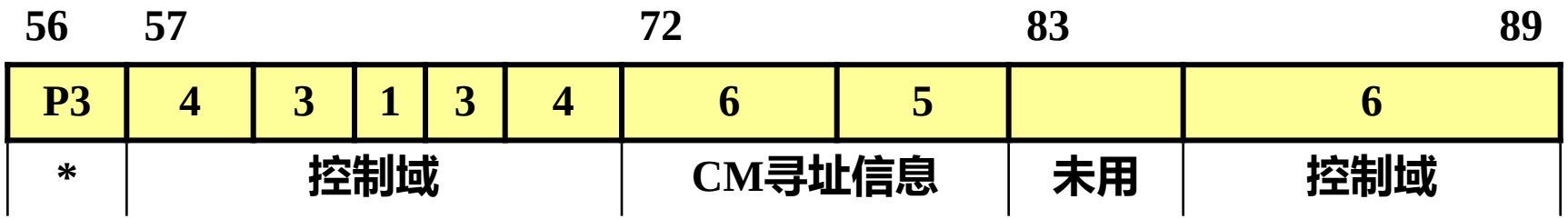
速度慢

4. 微指令控制域编码设计实例

- IBM system/360 Model 50的微指令：由90位构成，其中有21个字段的控制域、5个字段的地址域和3个校验位。



*P1：1～30位的校验；　P2：32～55位的校验



*P3：57～89位的校验

图6.19　IBM system/360 Model 50的微指令格式

4. 微指令控制域编码设计实例

- IBM system/370 Model 145的微指令：由32位构成
  - ✓ 微操作码：指定应完成的微操作
  - ✓ 微操作数：如CPU寄存器的地址
  - ✓ 下一条微指令地址的信息

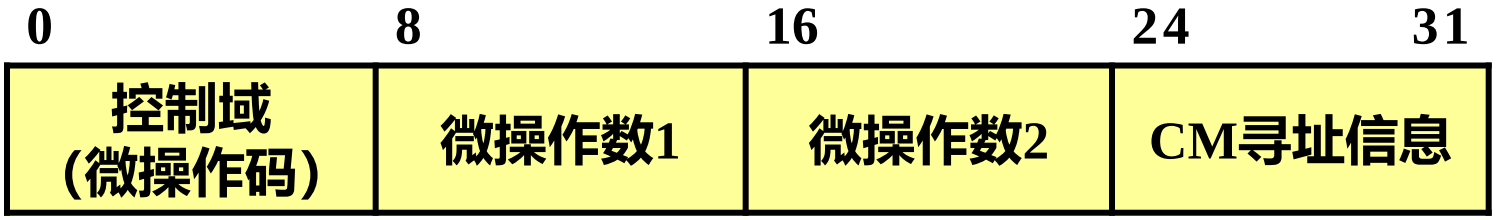
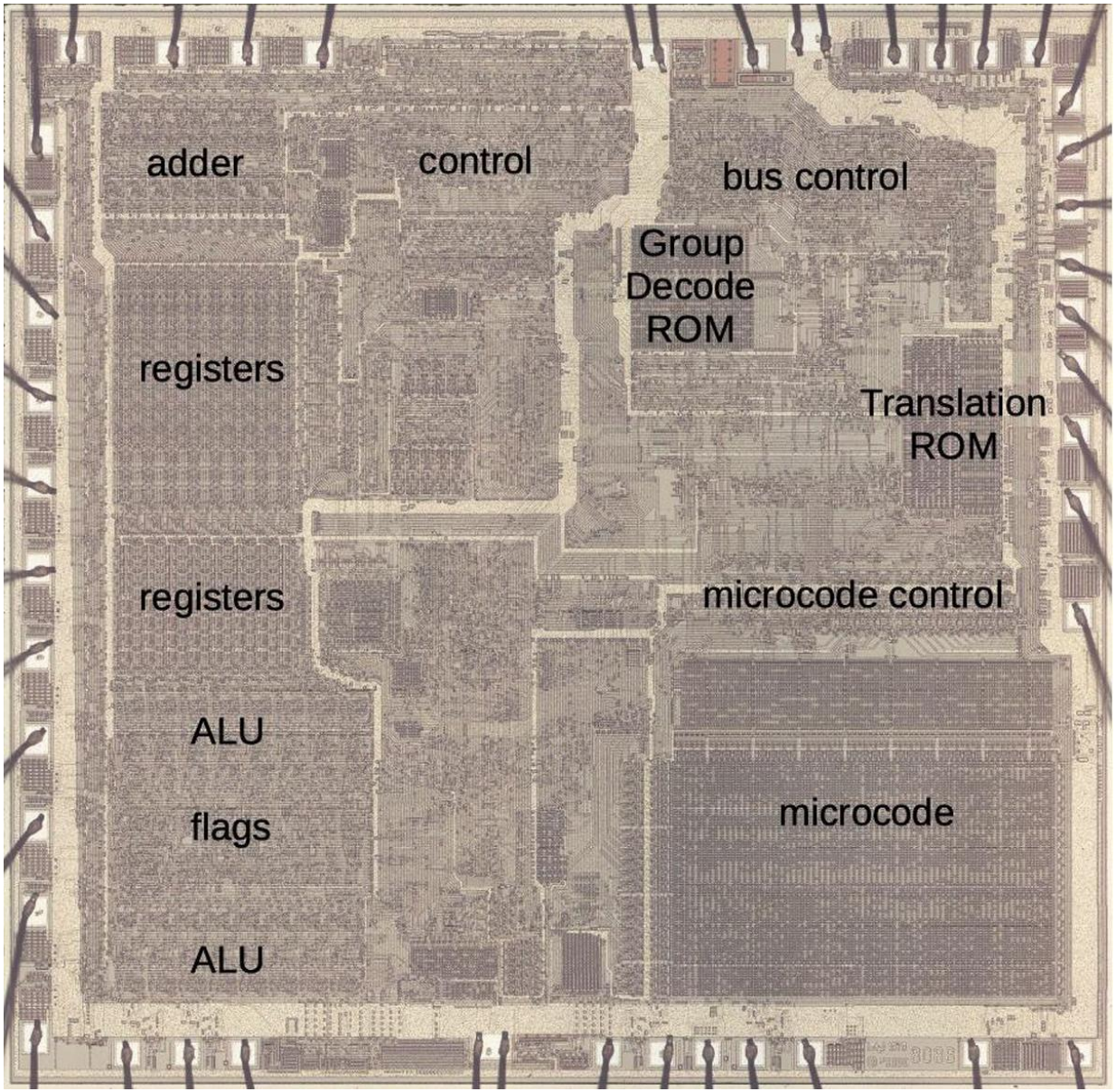


图6.20 IBM system/370 Model 145的微指令格式

6.3.2 微指令设计



8086 CPU
显微照片

微程序控制器设计

——微程序设计

6.3.3 微程序设计　　1. 微程序结构

(1) 一条指令对应一段完整的微程序

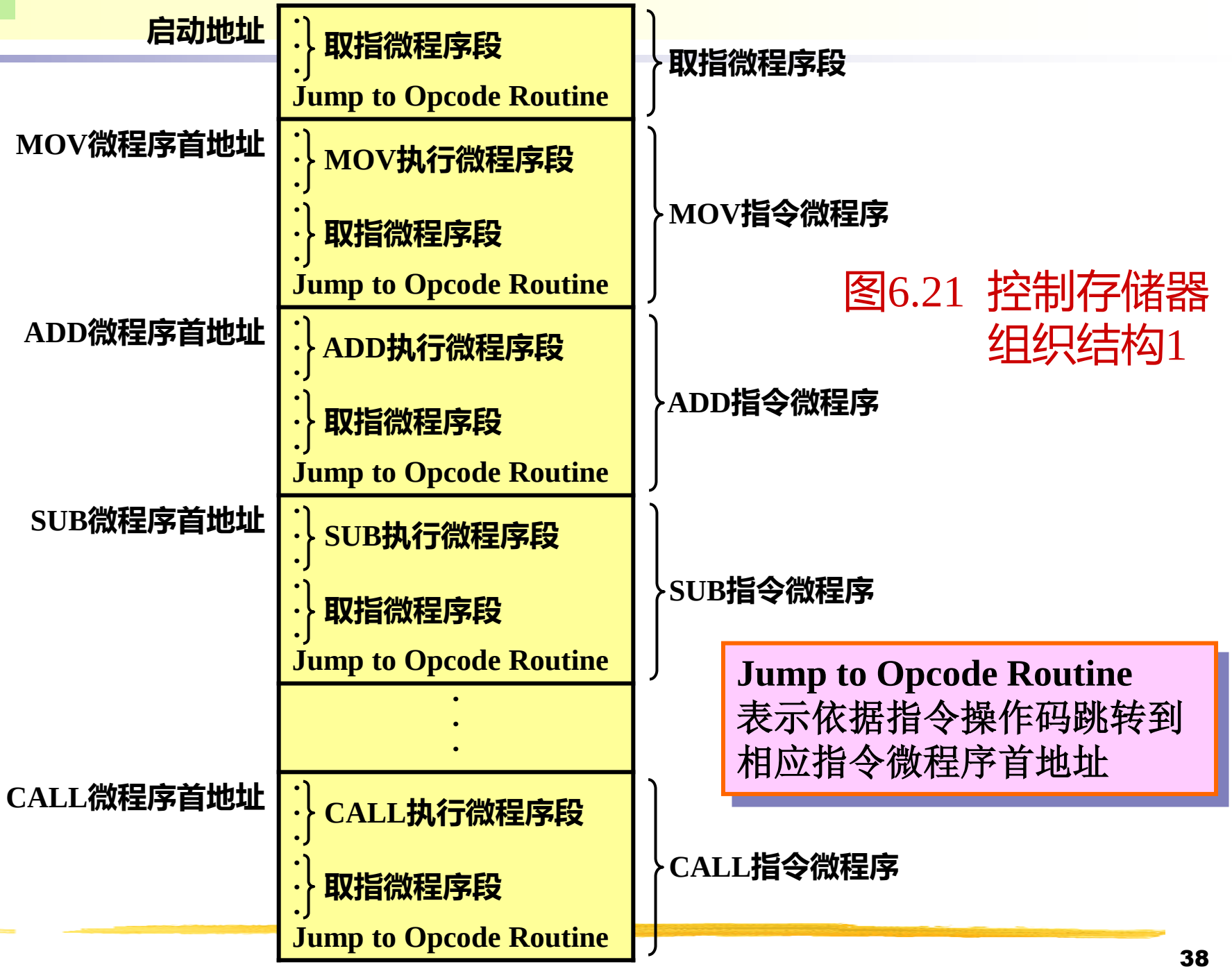


图6.21 控制存储器组织结构1

Jump to Opcode Routine
表示依据指令操作码跳转到相应指令微程序首地址

(2) 将微程序中的公共部分设计成微子程序进行公共调用

图6.22 控制存储器组织结构2

6.3.3 微程序设计　　2. 编写微程序

- 编写微程序要做两件事：

1. 按照设计好的微指令格式，将指令微操作（微命令）序列按每节拍一条微指令写出每条微指令的具体编码；
2. 按照选定的微程序结构，将微指令组织成微程序或微子程序。

- 微指令格式：

控 制 域				地址域
字段1(4)	字段2(4)	字段3(4)	字段4(3)	AC(1)

表6.6 微程序首地址的生成

微程序首地址	指令操作码(4)	指令寻址方式(4)	JA(4)
控存启动地址	0	0	0
MOV R0, X	1	6	0
ADD R1, R0	2	1	0
SUB R0, (X)	3	7	0
…	…	…	…

6.3.3 微程序设计

2. 编写微程序

表6.4

图6.3

图6.4

AC=0，下条顺序微指令地址由μPC提供；
AC=1，根据指令操作码和寻址方式信息跳转到微程序首地址。

表6.7 微程序段示例

微程序名	微地址	微指令					节拍	微操作	微命令
取指	000H	1011	1110	0000	000	0	T1	AR←PC	PC_{out}，AR_{in}
	001H	1101	1011	0000	001	0	T2	DR←Memory[AR]	AR_{out}，Mread，DRS_{in}
	002H	1001	1100	1001	000	1	T3	PC←PC+I, IR←DR	PC+1，DRI_{out}，IR_{in}
MOV	160H	1011	1001	0000	000	0	T1	AR←IR（地址字段）	IR_{out}，AR_{in}
	161H	1101	1011	0000	001	0	T2	DR←Memory[AR]	AR_{out}，Mread，DRS_{in}
	162H	0001	1100	0000	000	0	T3	R0←DR	DRI_{out}，$R0_{in}$
	163H	1011	1110	0000	000	0	T4	AR←PC	PC_{out}，AR_{in}
	164H	1101	1011	0000	001	0	T5	DR←Memory[AR]	AR_{out}，Mread，DRS_{in}
	165H	1001	1100	1001	000	1	T6	PC←PC+I, IR←DR	PC+1，DRI_{out}，IR_{in}
ADD	210H	1010	0001	0000	000	0	T1	Y←R0	$R0_{out}$，Y_{in}
	211H	0000	0010	0001	000	0	T2	Z←R1+Y	$R1_{out}$，ADD
	212H	0010	1010	0000	000	0	T3	R1←Z	Z_{out}，$R1_{in}$
	213H	1011	1110	0000	000	0	T4	AR←PC	PC_{out}，AR_{in}
	214H	1101	1011	0000	001	0	T5	DR←Memory[AR]	AR_{out}，Mread，DRS_{in}
	215H	1001	1100	1001	000	1	T6	PC←PC+I, IR←DR	PC+1，DRI_{out}，IR_{in}

图6.3

图6.4

表6.7 微程序段示例（续）

微程序名	微地址	微指令					节拍	微操作	微命令
SUB	370H	1011	1001	0000	000	0	T1	AR←IR（地址字段）	IR_{out}，AR_{in}
	371H	1101	1011	0000	001	0	T2	DR←Memory[AR]	AR_{out}，Mread，DRS_{in}
	372H	1011	1100	0000	000	0	T3	AR←DR	DRI_{out}，AR_{in}
	373H	1101	1011	0000	001	0	T4	DR←Memory[AR]	AR_{out}，Mread，DRS_{in}
	374H	1010	0001	0000	000	0	T5	Y←R0	$R0_{out}$，Y_{in}
	375H	0000	1100	0010	000	0	T6	Z←Y－DR	DRI_{out}，SUB
	376H	0001	1010	0000	000	0	T7	R0←Z	Z_{out}，$R0_{in}$
	377H	1011	1110	0000	000	0	T8	AR←PC	PC_{out}，AR_{in}
	378H	1101	1011	0000	001	0	T9	DR←Memory[AR]	AR_{out}，Mread，DRS_{in}
	379H	1001	1100	1001	000	1	T10	PC←PC+I，IR←DR	PC+1，DRI_{out}，IR_{in}
· · ·	· · ·	· · ·					· · ·	· · ·	· · ·

6.3.4 微程序控制器设计

- 微程序控制器设计的基本原则：
  - ✓ 速度快
    - 快速产生下条微指令地址
    - 快速获得微指令
    - 快速产生控制信号
  - ✓ 体积小
    - 控制器中使用的器件数量
    - 控制存储器的规模

6.3.4 微程序控制器设计

- 微程序控制器设计的基本步骤：
  1. 指令分析：得到指令微操作（微命令）序列
  2. 微命令分析：归类，相容性/互斥性检查
  3. 微指令及控制器结构设计
     - ① 确定微指令地址域格式：两地址/单地址/可变格式
     - ② 确定微指令控制域格式：水平/垂直/字段编码
  4. 微程序与控制存储器设计

     控制存储器容量＝微指令长度×(平均微程序长度(微指令数)/一条指令)×指令数
     - ① 选择微指令和微程序结构：结构1、结构2
     - ② 确定微程序入口地址(首地址)的生成方法
  5. 微程序控制器实现
     - 设计硬件逻辑电路并实现
     - 编写微程序并存储到控制存储器中

两种控制器设计方法比较

6.4 微程序控制器与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