计算机组成原理与系统结构

第七章 控制器

http://jpkc.hdu.edu.cn/computer/zcyl/dzkjdx/

杭州電子科技大学
HANGZHOU DIANZI UNIVERSITY

第七章　控制器

BACK

7.3 微程序控制器

一 微程序控制的基本概念和工作原理

二 简单微程序控制器的设计

三 微程序设计技术

四 微程序控制方式下模型机的设计实例

五 模型机微程序设计

六 微程序控制器与硬布线控制器的比较

一、微程序控制的基本概念和工作原理

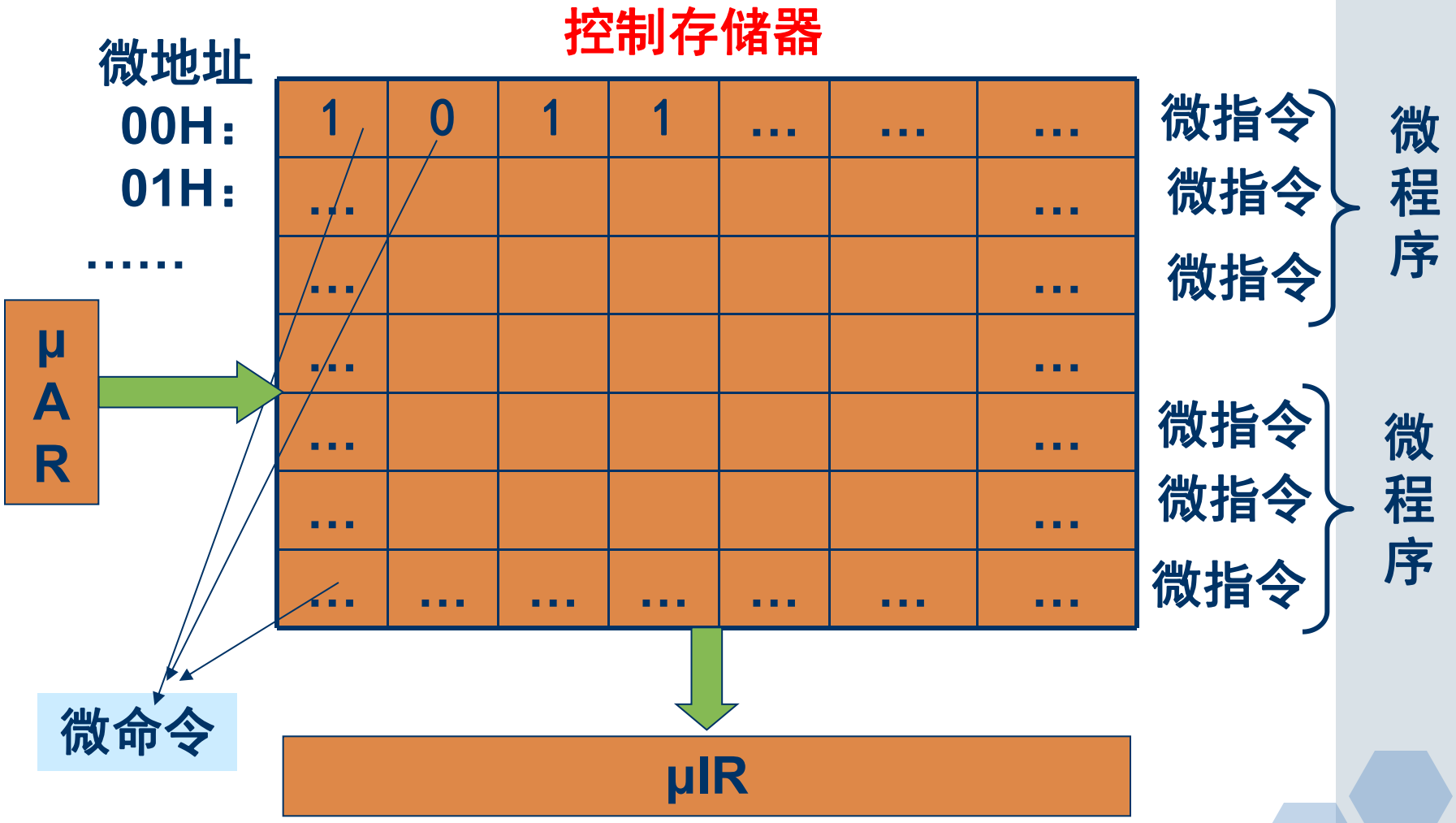
- 微程序设计思想：每条机器指令的功能都用一段相应的微程序来实现。
- 1、基本概念：
  - 微操作：指令执行时必须完成的基本操作。例如，PC→AR，PC+1→ PC，RAM→IR。
  - 微命令：是组成微指令的最小单位，也就是控制实现微操作的控制信号。一般用于控制数据通路上门的打开/关闭，或者功能选择。
  - 微指令：是一组微命令的集合，用于完成一个功能相对完整的操作。
  - 微程序：微指令的有序集合，用于实现机器指令的功能。

一、微程序控制的基本概念和工作原理

❖1、基本概念：

- 控制存储器：简称控存，用于存放所有指令的微程序，其中一个存储单元存放一条微指令。一般为ROM。
- 微地址：微指令在控存中的地址。
- 微地址寄存器μAR：存放微地址的寄存器。
- 微指令寄存器μIR：存放从控存取出的微指令的寄存器。
- 微周期：指从控存中取出并执行一条微指令所需要的时间，一般与一个机器周期相当。

微程序基本概念关系


控制存储器
微地址
00H：
01H：
……
μAR
1 0 1 1 … … …
微指令
微指令
微指令
微程序
微指令
微指令
微指令
微程序
微命令
μIR

一、微程序控制的基本概念和工作原理

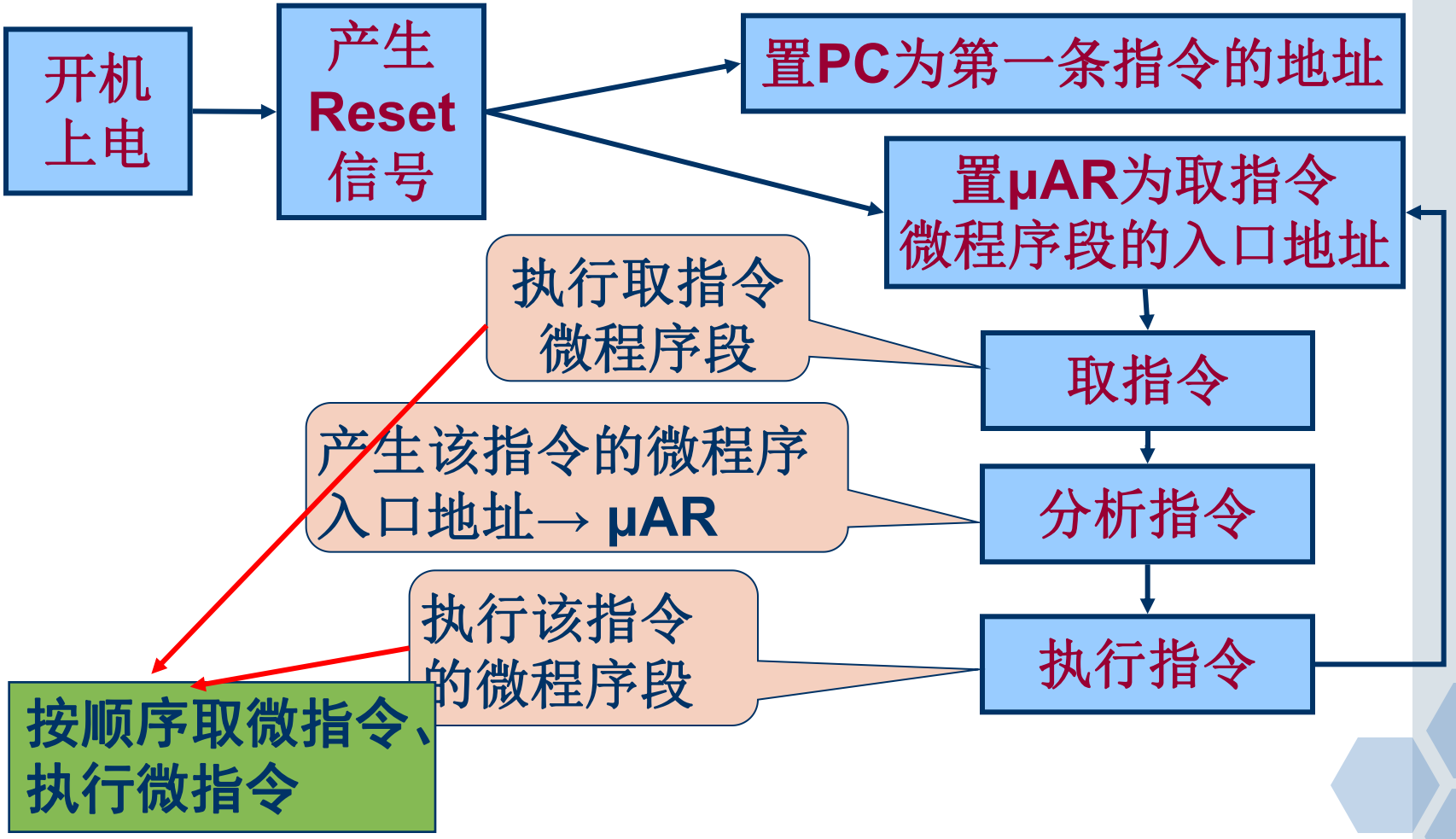
- 2、微程序控制器的工作原理：
- 一条机器指令由一段微程序来解释实现 。
- 控制存储器中包含：
  - 取指令的微程序段：公操作（所有指令共用）
  - 各条指令的微程序段

控制存储器

取指令微程序段
MOV指令的微程序段
ADD指令的微程序段
SUB指令的微程序段
……
HALT指令的微程序段

一、微程序控制的基本概念和工作原理

- 微程序控制的计算机工作过程：



一、微程序控制的基本概念和工作原理

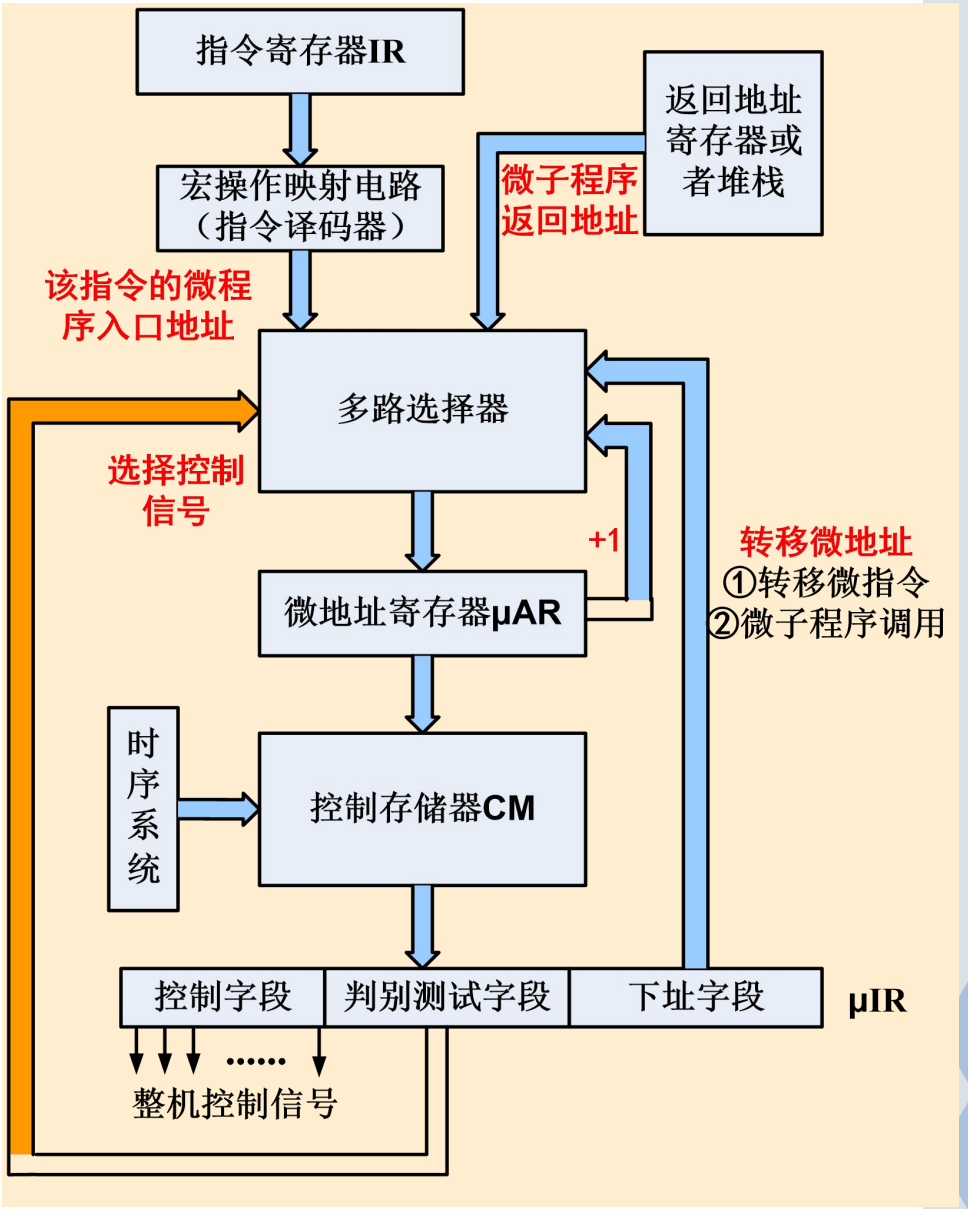
❖3、微程序控制器的组成

- 其他部件均等同于硬布线控制器
- 操作控制信号形成部件主要由以下3个部件构成：

①控制存储器CM

②微地址寄存器μAR

③微指令寄存器μIR



一、微程序控制的基本概念和工作原理

4、微指令的一般结构

- **控制字段：**包含了一组微命令信号，用于控制完成本条微指令的操作。
- **下址字段：**用于指出后继微地址（下条微指令地址）的相关信息。

控制字段	下址字段

一、微程序控制的基本概念和工作原理

❖ 后继微地址的生成方法：

① 自增1：后继微地址=当前微地址+1，用于顺序执行微程序的场合

② 下址字段产生：后继微地址由当前微指令的下址字段指定，用于微程序转移的场合。

- 条件/无条件转移、微子程序调用

③ 宏操作映射：由机器指令操作码产生该指令对应的微程序入口地址，主要用于指令译码。

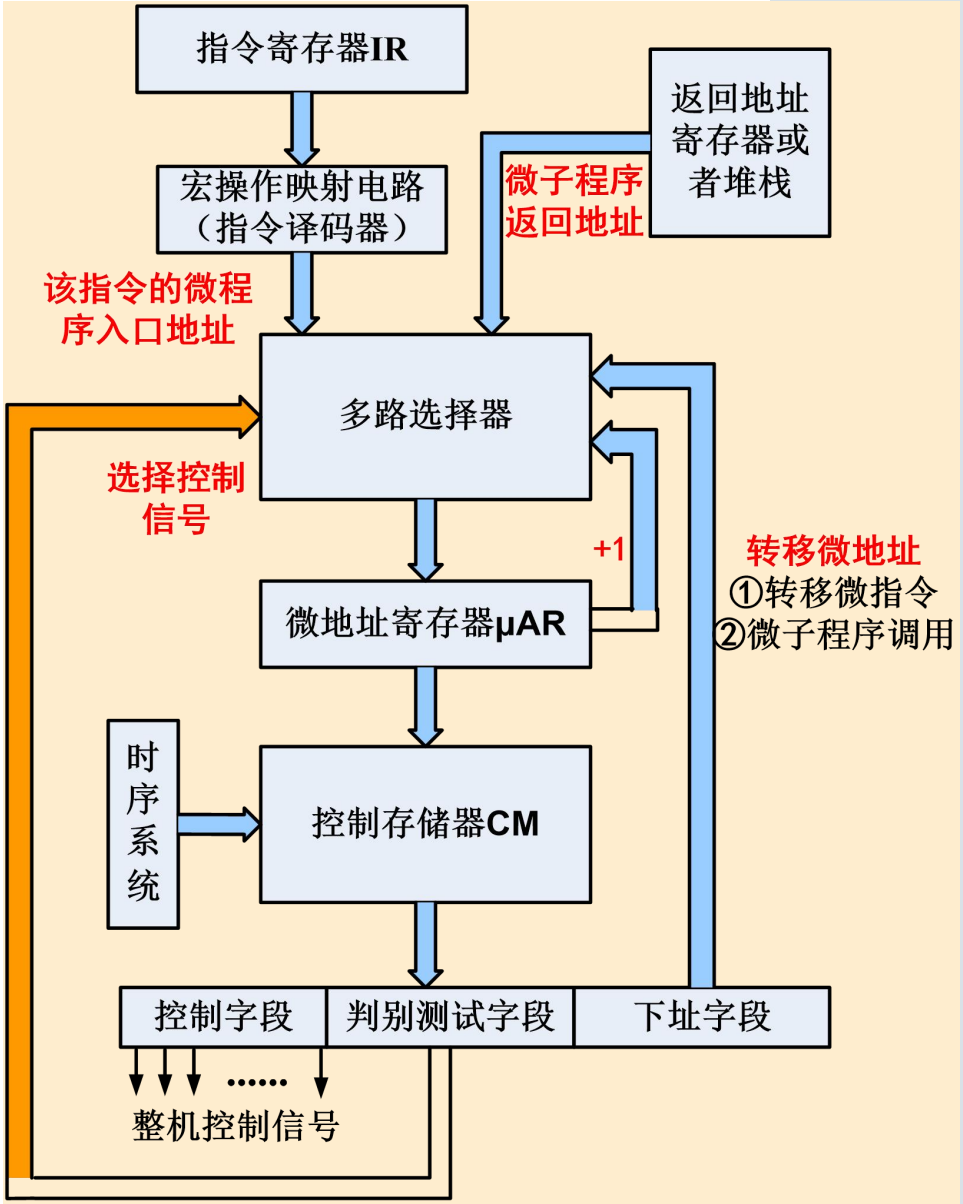
- 实现方法：MAPROM或者逻辑电路

④ 微子程序返回地址：由微子程序寄存器和堆栈产生微子程序的返回地址，用于微子程序的返回。

一、微程序控制的基本概念和工作原理

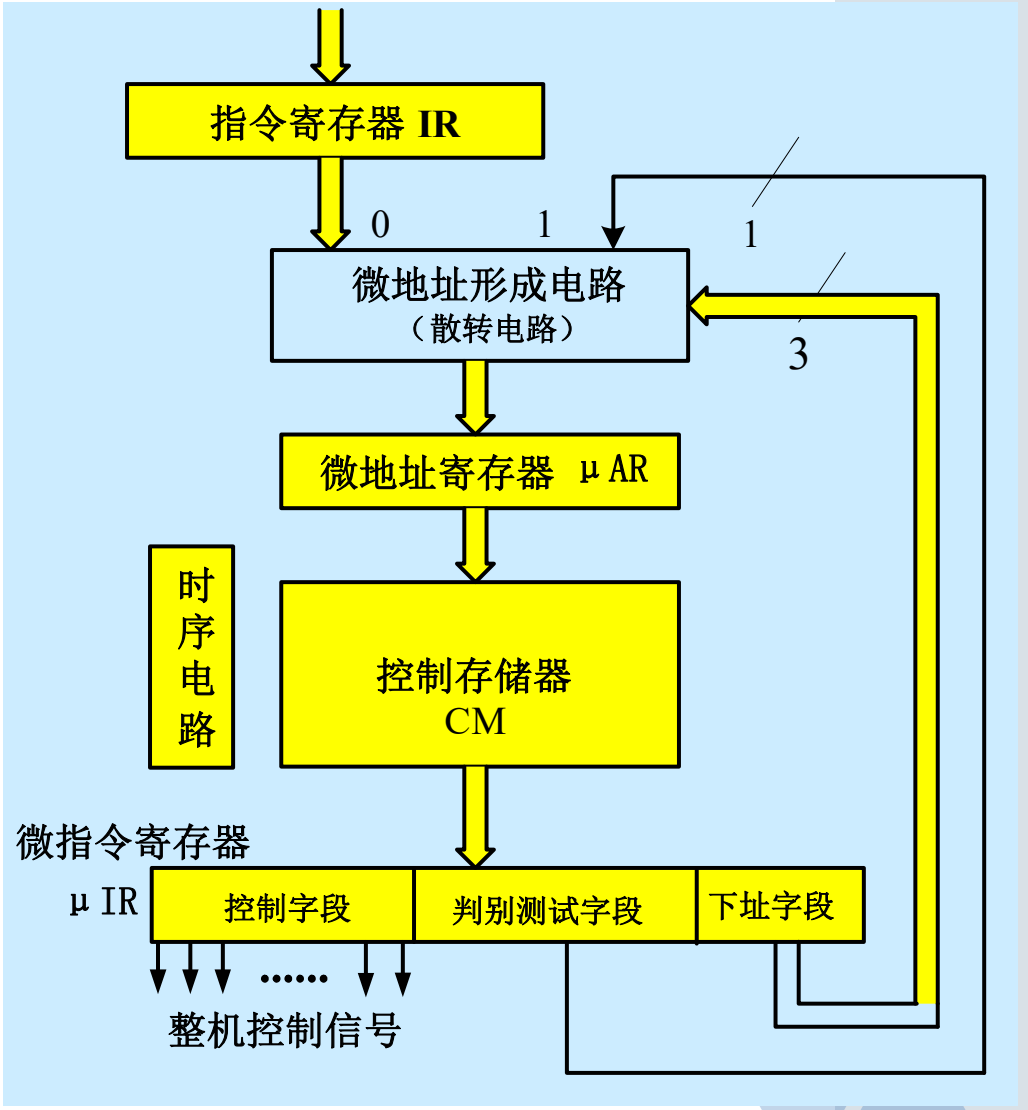
- 问题：如何确定下一条微指令的地址来源？
- 增加**判别测试字段**，用于指明**后继微地址**的来源。

控制字段	判别测试字段	下址字段



二、简单微程序控制器的设计

- 微程序控制器的设计主要完成两个任务：
  - 产生正确的微命令；
  - 产生正确的微指令序列（即上述CPU状态转换序列）。
- 怎样采用微程序控制的方法来设计CPU呢？



简单微程序控制器的组成框图

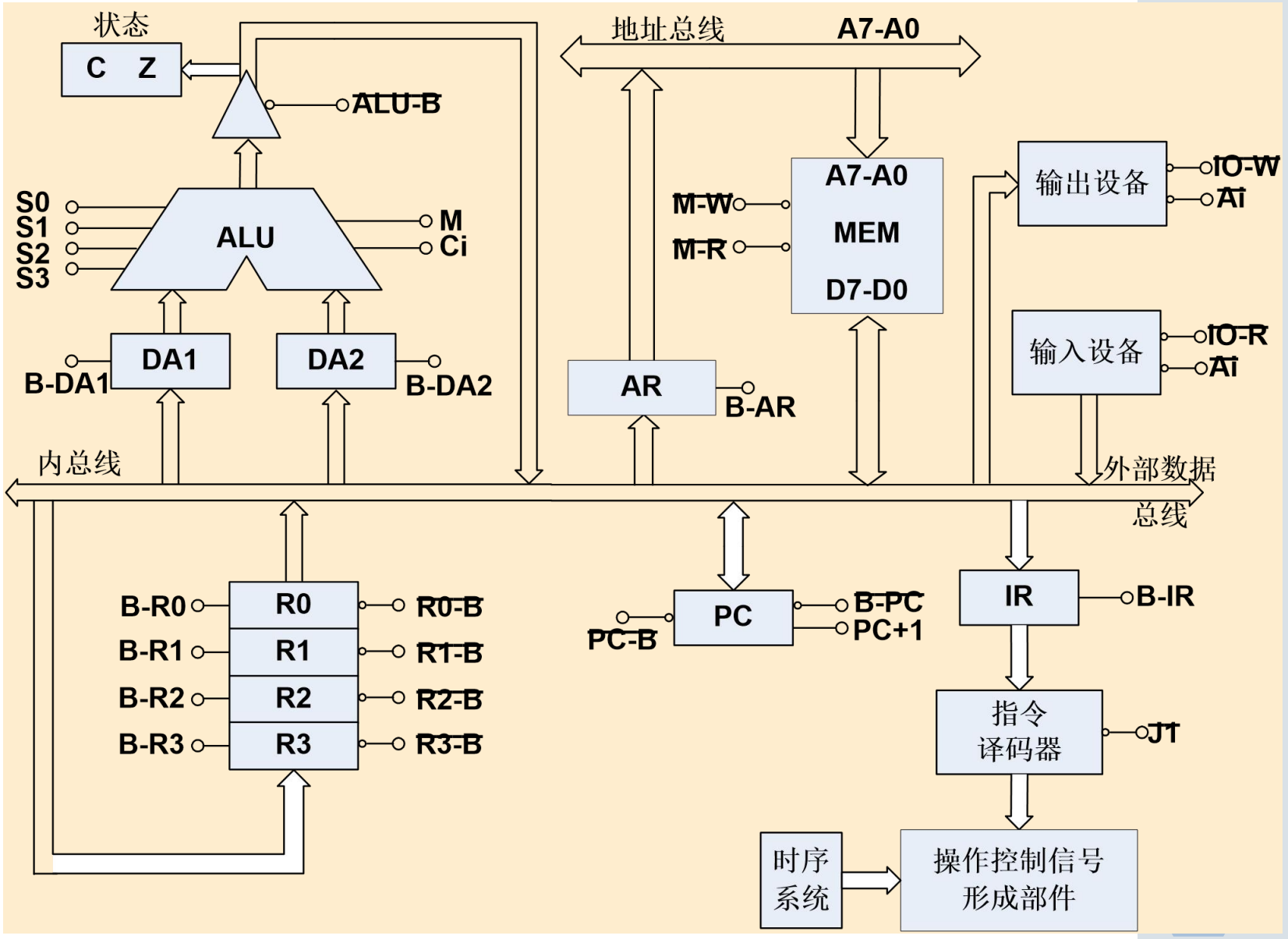
二、简单微程序控制器的设计

❖1.确定指令系统

助记符	格式	操作码OP	指令机器码	指令功能
ADD R_0, 06H	OP | ×× | DR 立即数	0101	0101 | 0000 0000 0110	（R_0）+ 06H→R_0
JMP 04H	OP | ×××× 转移地址	1000	1000 | 0000 0000 0100	04H→PC

二、简单微程序控制器的设计

2. 确定CPU的内部结构



控制字段—控制信号定义

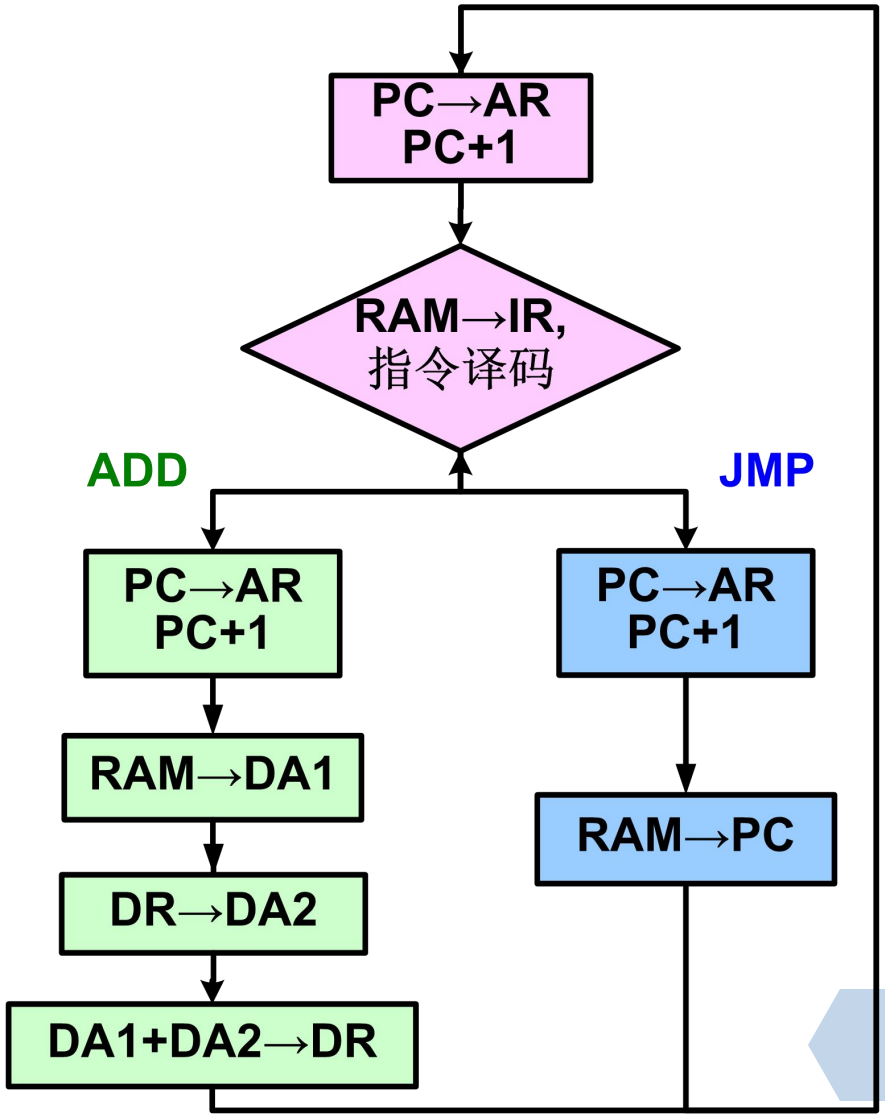
序号	控制信号	功能	序号	控制信号	功能
1	$\overline{\text{PC-B}}$	指令地址（PC）送总线	13	B-DA1	总线内容打入暂存器DA1
2	B-AR	总线内容打入地址寄存器	14	B-DA2	总线内容打入暂存器DA2
3	PC+1	程序计数器内容+1	15	$\overline{\text{ALU-B}}$	运算器ALU内容送总线
4	$\overline{\text{B-PC}}$	总线内容打入程序计数器	16	Ci	ALU进位输入
5	B-IR	总线内容打入指令寄存器	17	B-R0	总线内容打入R0寄存器
6	$\overline{\text{M-W}}$	存储器写	18	B-R1	总线内容打入R1寄存器
7	$\overline{\text{M-R}}$	存储器读	19	B-R2	总线内容打入R2寄存器
8	S_3	S_3- S_0选择ALU 16种运算之1	20	B-R3	总线内容打入R3寄存器
9	S_2	同上	21	$\overline{\text{R0-B}}$	R0寄存器内容送总线
10	S_1	同上	22	$\overline{\text{R1-B}}$	R1寄存器内容送总线
11	S_0	同上	23	$\overline{\text{R2-B}}$	R2寄存器内容送总线
12	M	M=1，ALU做逻辑运算 M=0，ALU做算术运算	24	$\overline{\text{R3-B}}$	R3寄存器内容送总线

二、简单微程序控制器的设计

- 3. 分析每条指令的执行过程，画出微程序流程图
- ADD指令：分为6个机器周期完成
  - M0： PC→AR，PC+1→PC；（取指令地址）
  - M1： RAM→IR, J1#；（取指令并译码）
  - ADD·M2： PC→AR，PC+1→PC；（取指令第二字地址）
  - ADD·M3： RAM→DA1；（取数据）
  - ADD·M4： DR→DA2；（送寄存器数据）
  - ADD·M5： DA1+DA2→DR；（计算并存结果）
- JMP指令：分为4个机器周期完成
  - M0： PC→AR，PC+1→PC；（取指令地址）
  - M1： RAM→IR,J1#；（取指令并译码）
  - JMP·M2： PC→AR，PC+1→PC；（取指令第二字地址）
  - JMP·M3： RAM→PC；（取转移地址并执行转移）

二、简单微程序控制器的设计

❖微程序流程图



二、简单微程序控制器的设计

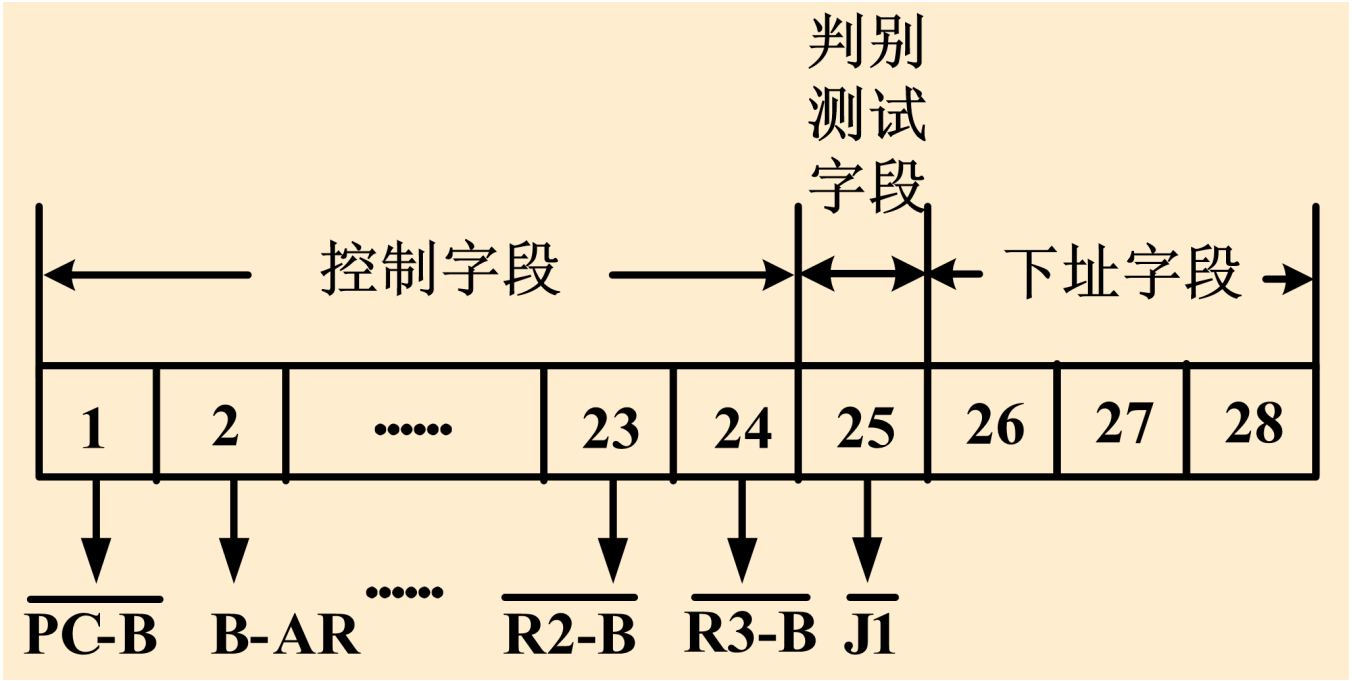
- 4. 写出每条微指令所发送的微操作控制信号序列
- 取指令公操作：
  - M0：PC-B#，B-AR，PC+1；
  - M1：M-R#，B-IR，J1#；
- ADD指令：
  - ADD·M2： PC-B#，B-AR，PC+1；
  - ADD·M3： M-R#，B-DA1；
  - ADD·M4：R0-B#，B-DA2；
  - ADD·M5：$ALU_{S3S2S1S0MCi}$（F=A加B），ALU-B#，B-R0；
- JMP指令：
  - JMP·M2： PC-B#，B-AR，PC+1；
  - JMP·M3： M-R#，B-PC#，PC+1；

二、简单微程序控制器的设计

即8条微指令，控存8个单元

- 5. 设计微指令格式
- CPU的有限状态机只有8个状态，可能产生8个下址。

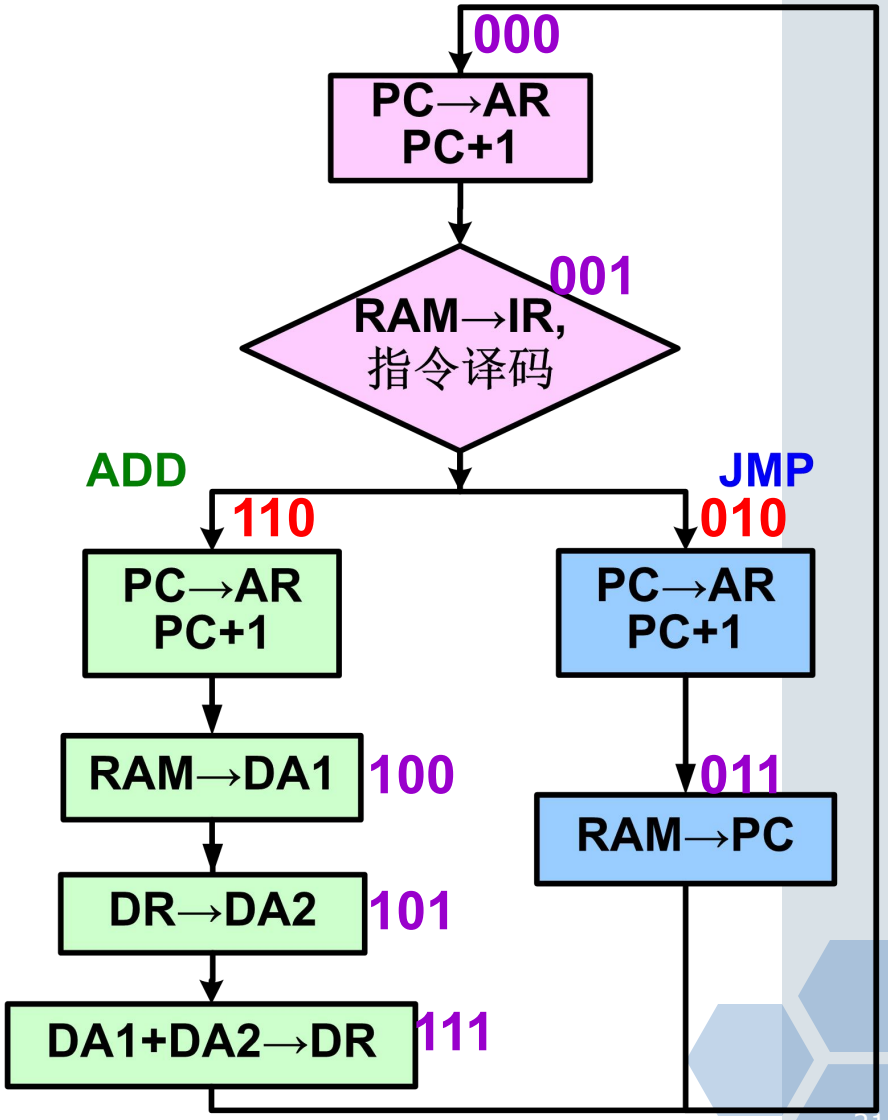
控制字段（24位）	判别测试字段（1位）	下址字段（3位）



- J1#=0：后继微地址由指令译码器产生（该条指令的微程序入口地址）
- J1#=1：后继微地址由当前微指令的下址字段产生

二、简单微程序控制器的设计

- 6.分配微地址，并编写微指令代码
- 指令译码器译码原理：
  - 输入：指令操作码 $OP=I_7I_6I_5I_4$；
  - 输出：该指令的微程序入口地址=I_4 10
- 所以
  - ADD（OP=0101）入口=110
  - JMP（OP=1000）入口=010



二、简单微程序控制器的设计

❖6. 分配微地址，并编写微指令代码

微地址	微指令（状态）	判别测试字段（J1#）	下址字段
000	M0：PC→AR, PC+1	1	001
001	M1：RAM→IR,译码	0	×××
010	JMP•M2：PC→AR, PC+1	1	011
011	JMP•M3：RAM→PC	1	000
100	ADD•M3：RAM→DA1	1	101
101	ADD•M4：Rd→DA2	1	111
110	ADD•M2：PC→AR, PC+1	1	100
111	ADD•M5：DA1+DA2→Rd	1	000

二、简单微程序控制器的设计

❖6. 分配微地址，并编写微指令代码

微地址	微指令发出的微操作信号	判别测试字段（J1#）	下址字段
000	M0： PC-B#,B-AR,PC+1	1	001
001	M1： M-R# ,B-IR,J1#	0	×××
010	JMP•M2： PC-B#,B-AR,PC+1	1	011
011	JMP•M3： M-R#, B-PC#,PC+1	1	000
100	ADD•M3： M-R#, B-DA1	1	101
101	ADD•M4： R0-B#,B-DA2	1	111
110	ADD•M2： PC-B#,B-AR,PC+1	1	100
111	ADD•M5： $S_3S_2S_1S_0MC_i$=100101, ALU-B#,B-R0	1	000

二、简单微程序控制器的设计

❖6. 分配微地址，并编写微指令代码

微地址	微指令代码	判别测试字段（J1#）	下址字段
000	01110110000000100000111 1	1	001
001	10011100000000100000111 0	0	×××
010	01110110000000100000111 1	1	011
011	10100100000000100000111 1	1	000
100	10010100000010100000111 1	1	101
101	10010110000001100000011 1	1	111
110	01110110000000100000111 1	1	100
111	10010111001000011000111 1	1	000

二、简单微程序控制器的设计

❖7. 微指令代码装入控制存储器的相应单元

微地址	微指令代码
000	76020F9 H
001	9C020E0 H
010	76020FB H
011	A4020F8 H
100	940A0FD H
101	960607F H
110	76020FC H
111	97218F8 H

三、微程序设计技术

- 研究与应用微程序设计技术的目的：
  1. 缩短微指令字长；
  2. 减少控制存储器的容量；
  3. 加快微程序的执行速度；
  4. 便于微程序的修改与扩充；
  5. 提高微程序设计的灵活性。
- 微指令由两部分构成：控制字段和下址字段
- 研究方法：
  - 控制字段设计技术：微指令的编译法
  - 下址字段设计技术

三、微程序设计技术

1. 微指令的编译法
2. 微指令下址字段设计方法
3. 微指令格式的类型
4. 控制存储器和动态微程序设计
5. 毫微程序设计

1、微指令的编译法

（1）直接控制法

（2）全译码方式

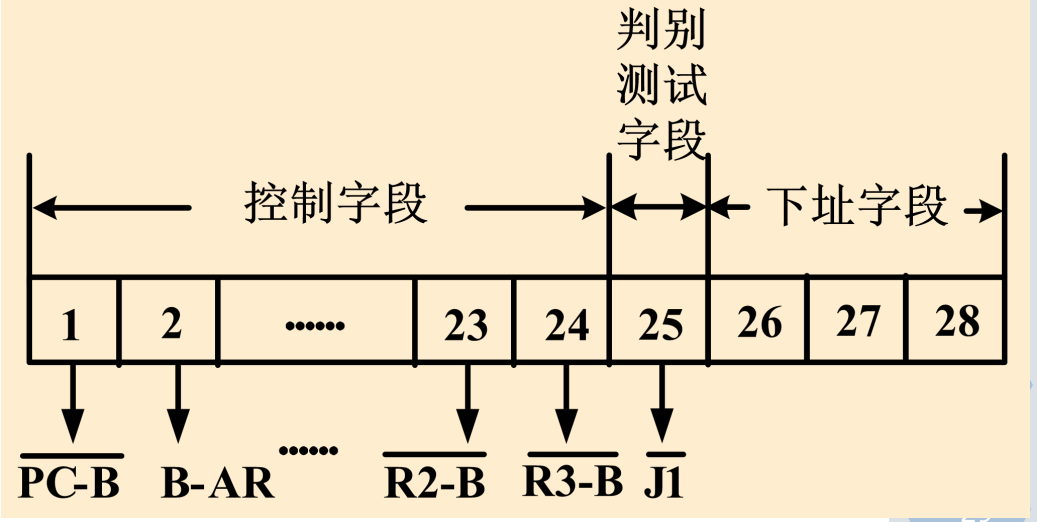
（3）字段直接编译法

（4）字段间接编译法

（1）直接控制法

- 控制字段的编码方法：每一位代表一个微命令（控制信号），编写微指令方法：**直接控制**
  - 如果要发出某个微命令：将控制字段中对应位，置为有效，即打开其控制门
  - 如果不要发出某个微命令：将控制字段中对应位，置为无效，即关闭其控制门

- 优点：无需译码，执行速度快；微程序较短。
- 缺点：微指令字长很长，占用控存容量大。



（2）全译码方式

- 控制字段的编码方法：将所有的控制信号进行编码，作为控制字段。在执行微指令时，译码产生各个微命令。
  - 每条微指令只能发送1～2个微命令。
- 优点：微指令字长很短。
- 缺点：并行操作能力弱，微程序很长，执行速度慢
- 一般用于垂直微指令格式。

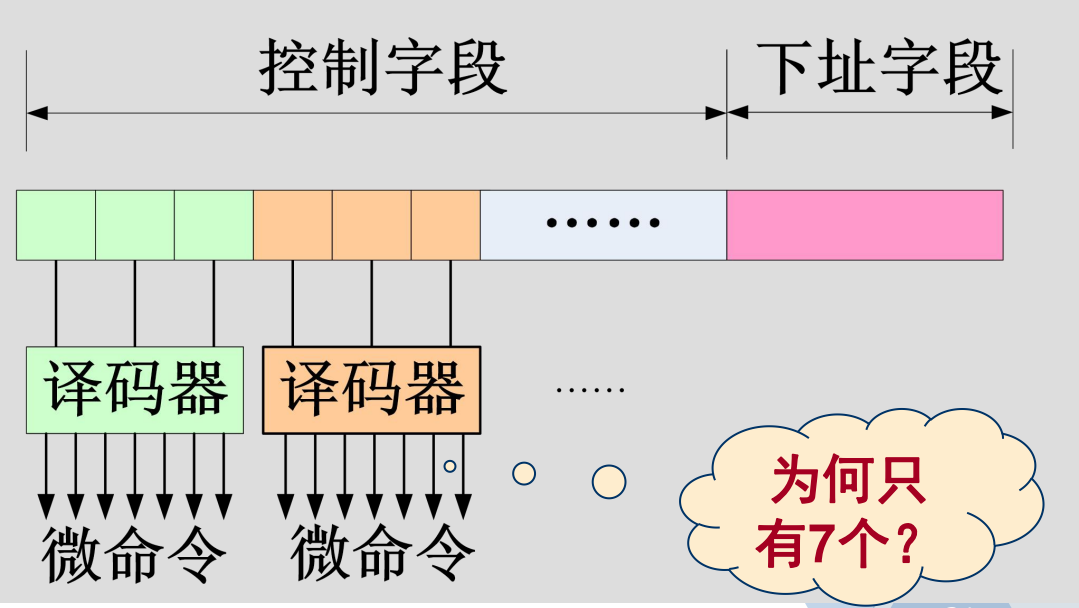
（3）字段直接编译法

可以提高信息位的利用率，缩短微指令字长

有利于实现并行操作，加快指令的执行速度

- 法：将控制字段分成若
  个控制信号。
- 优点：并行能力较强，字长较短
- 字段直接编译法的基本分段原则是：
  - 相斥性微命令分在同一字段内，相容性微命令分在不同字段内。

- 相斥性微命令：指在同一个微周期中不可能同时出现的微命令。
- 相容性微命令：指在同一个微周期中可以同时出现的微命令



用字段直接编译法，重新设计微指令格式

- 24个控制信号中，将总线数据送目的部件的7个控制信号组成1个字段（BTO）编码和译码，将部件数据送总线的4个控制信号组成1个字段（OTB）编码和译码
- 11个控制信号：从直接控制的11位缩短到了字段直接译码的6位。
- 去掉了B-R1、 B-R2 、B
- 微指令字长从28位缩短到

M_{16}~M_{14}	M_{13}~M_{11}	M_{10}	M_9
BTO	OTB	PC+1	S_3

编码+译码	BTO	OTB
000		
001	B-DA1	ALU-B#
010	B-DA2	PC-B#
011	B-IR	R0-B#
100	B-AR	M-R#
101	B-R0	
110	M-W#	
111	B-PC#	

字段直接编译法设计的微程序

微地址	微指令发出的微操作信号	判别测试字段（J1#）	下址字段
000	M0： PC-B#, B-AR, PC+1	1	001
001	M1： M-R# , B-IR, J1#	0	×××
010	JMP•M2： PC-B#, B-AR, PC+1	1	011
011	JMP•M3： M-R#, B-PC#, PC+1	1	000
100	ADD•M3： M-R#, B-DA1	1	101
101	ADD•M4： R0-B#, B-DA2	1	111
110	ADD•M2： PC-B#, B-AR, PC+1	1	100
111	ADD•M5： $S_3S_2S_1S_0MC_i$=100101, ALU-B#, B-R0	1	000

字段直接编译法设计的微程序

微地址	M_{16}~M_{14}	M_{13}~M_{11}	M_{10}	M_9	M_8	M_7	M_6	M_5	M_4	M_3	M_2~M_0
	BTO	OTB	PC+1	S_3	S_2	S_1	S_0	M	C_i	J1#	下址
00H	100	010	1	0	0	0	0	0	0	1	001
01H	011	100	0	0	0	0	0	0	0	0	***
02H	100	010	1	0	0	0	0	0	0	1	011
03H	111	100	1	0	0	0	0	0	0	1	000
04H	001	100	0	0	0	0	0	0	0	1	101
05H	010	011	0	0	0	0	0	0	0	1	111
06H	100	010	1	0	0	0	0	0	0	1	100
07H	101	001	0	1	0	0	1	0	1	1	000

（4）字段间接编译法

- 编码方法：某字段的编码含意，除了其本身的编码外，还需要由另一字段来加以解释。也就是说，某一字段所产生的微命令，是和另一字段的代码联合定义出来的。
- **优点：**进一步缩短微指令字长的一种编译法。
- 举例：用字段间接编译法，重新定义微指令格式

直接控制法

M_{15}~M_{13}	M_{12}~M_{11}	M_{10}	M_9	M_8	M_7	M_6	M_5	M_4	M_3	M_2~M_0
BTO	OTB	FUNC	FS	S_3	S_2	S_1	S_0	M	C_i	下址

字段直接编译法

字段间接编译法

用字段间接编译法，重新设计微指令

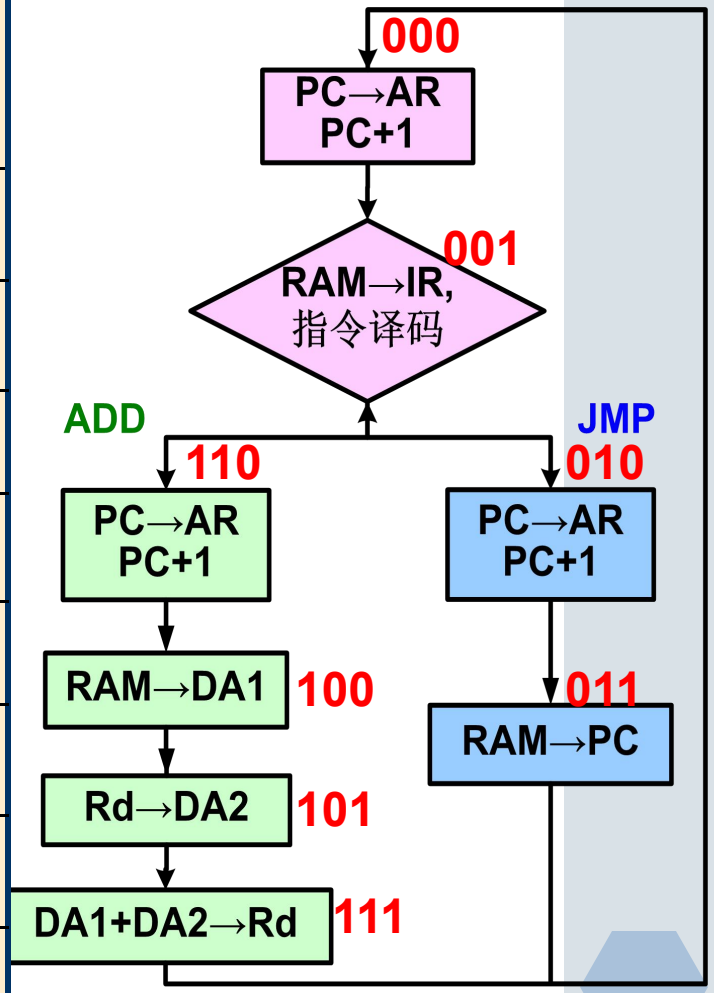
BTO 编码	BTO 信号
000	
001	B-DA1
010	B-DA2
011	B-IR
100	B-AR
101	B-R0
110	M-W#
111	B-PC

OTB 编码	OTB 信号
00	
01	ALU-B#
10	PC-B#
11	M-R#

FUNC 编码	FS=0	FS=1
0	PC+1	
1	J1#	R0-B#

字段间接编译法设计的微程序

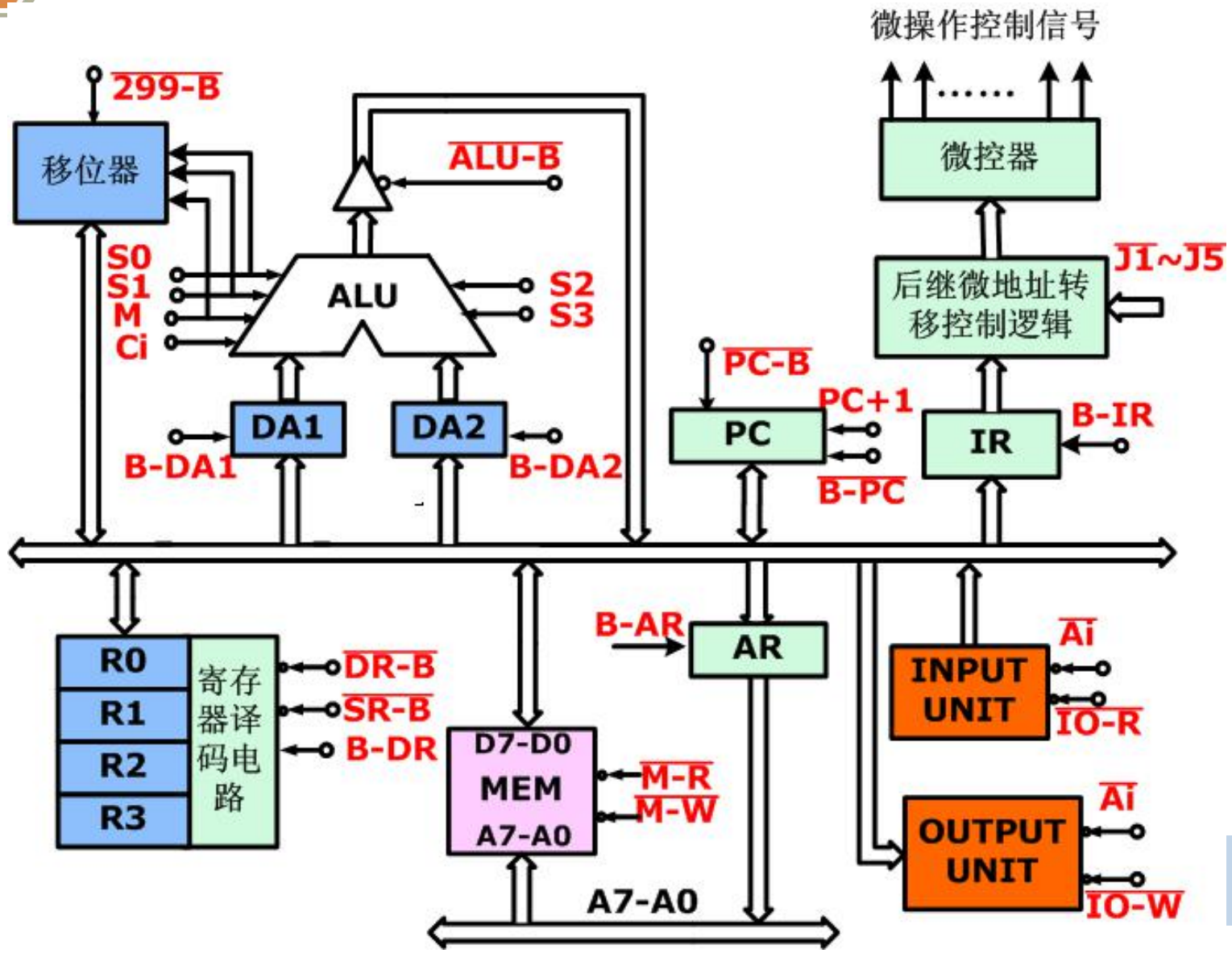
微地址	微指令发出的微操作信号	下址字段
000	M0： PC-B#,B-AR,PC+1	001
001	M1： M-R# ,B-IR,J1#	×××
010	JMP•M2： PC-B#,B-AR,PC+1	011
011	JMP•M3： M-R#, B-PC#,PC+1	000
100	ADD•M3： M-R#, B-DA1	101
101	ADD•M4： R0-B#,B-DA2	111
110	ADD•M2： PC-B#,B-AR,PC+1	100
111	ADD•M5： $S_3S_2S_1S_0MC_i$=100101, ALU-B#,B-R0	000



字段间接编译法设计的微程序

微地址	M_{15}~M_{13}	M_{12}~M_{11}	M_{10}	M_9	M_8	M_7	M_6	M_5	M_4	M_3	M_2~M_0
	BTO	OTB	FUNC	FS	S_3	S_2	S_1	S_0	M	C_i	下址
00H	100	10	0	0	0	0	0	0	0	0	001
01H	011	11	1	0	0	0	0	0	0	0	***
02H	100	10	0	0	0	0	0	0	0	0	011
03H	111	11	0	0	0	0	0	0	0	0	000
04H	001	11	0	1	0	0	0	0	0	0	101
05H	010	00	1	1	0	0	0	0	0	0	111
06H	100	10	0	0	0	0	0	0	0	0	100
07H	101	01	0	1	1	0	0	1	0	1	000

实验模型机系统结构


微操作控制信号
299-B
移位器
ALU-B
S0
S1
M
Ci
ALU
S2
S3
DA1
DA2
B-DA1
B-DA2
微控器
J1~J5
后继微地址转移控制逻辑
PC-B
PC+1
PC
B-PC
IR
B-IR
R0
R1
R2
R3
寄存器译码电路
DR-B
SR-B
B-DR
B-AR
AR
D7-D0
MEM
A7-A0
M-R
M-W
Ai
INPUT UNIT
IO-R
Ai
OUTPUT UNIT
IO-W
A7-A0

2. 模型机控制信号（微命令表）

序号	控制信号	功能	序号	控制信号	功能
1	PC-B#	指令地址（程序计数器）送总线	15	ALU-B##	运算器 ALU 内容送总线
2	B-AR	总线内容打入地址寄存器	16	Ci	ALU 进位输入
3	PC+1	程序计数器内容加一			
4	B-PC	总线内容打入程序计数器			
5	B-IR	总线内容打入指令寄存器			
6	M-W##	存储器写			
7	M-R##	存储器读			
8	S_3	S_3- S_0 选择 ALU16 种运算之一			
9	S_2	同上			
10	S_1	同上			
11	S_0	同上	25	I/O-W##	写（输出）I/O 端口
12	M	M 为“1”选择 ALU 做逻辑运算，M 为“0”选择 ALU 做算数运算	26	I/O-R##	读（输入）I/O 端口
13	B-DA1	总线内容打入暂存器 DA1	27	Ai##	端口地址线
14	B-DA2	总线内容打入暂存器 DA2	28	J1#	指令译码器工作

2. 模型机控制信号

17	SR-B#	源寄存器内容送到总线
18	DR-B	目标寄存器的内容送到总线
19	SP-B#	堆栈指示器内容送到总线
20	B_SP	总线上数据打入堆栈指示器
21	SI-B#	变址寄存器内容送到总线
22	B-DR	总线上的数据打入目标寄存器
23	B-SI	总线上的数据打入变址寄存器
24	J2#	指令操作码2译码控制信号
29	J3#	进位和零标志译码控制信号
30	J4#	控制台译码控制信号
31	J5#	中断控制译码控制信号
32	CyCn#	进位控制信号
33	CyNCn#	不带进位控制信号
34	INT_R#	中断请求信号
35	INT_E#	中断允许信号

2、模型机系统结构

- (1)运算器

① ALU：两片74LS181串连构成，16种算术运算和16种逻辑运算。

② 暂存器DA1和DA2：各由一片74LS273构成，暂存送到ALU运算的数据。

③ ALU输出缓冲器：一片74LS245构成，控制ALU运算结果是否送总线。

④ 状态寄存器：指示ALU运算结果的状态FC和FZ。

⑤ 移位器：一片74299构成，有四种移位功能和置数功能。

⑥ 寄存器：4个8位寄存器，由74LS273构成。

寄存器译码电路

- 对于控制器来说，指令MOV R0,R1和MOV [illegible]，还是2条不同的指令？
- [illegible]行的是同一段微程序，微[illegible]的控制信号。但如何区分不同的寄存器[illegible]？

B-R0、B-R1、B-R2、B-R3、 R0-B#、R1-B#、R2-B#、R3-B#）

I1 I0

I3 I2

方法是通过寄存[illegible]译码电路，依据指令的DR和SR字段，将微控器发出的统一的寄存器控制信号，翻译为具体的不同的寄存器控制号。

B-DR、DR-B#、SR-B#、SI-B#、SP-B#

寄存器译码电路

- 输入信号有：

① B-DR、DR-B#、SR-B#、SI-B#、SP-B#：来自微指令译码器。

② 指令码I3-I2：来自指令寄存器的SR字段

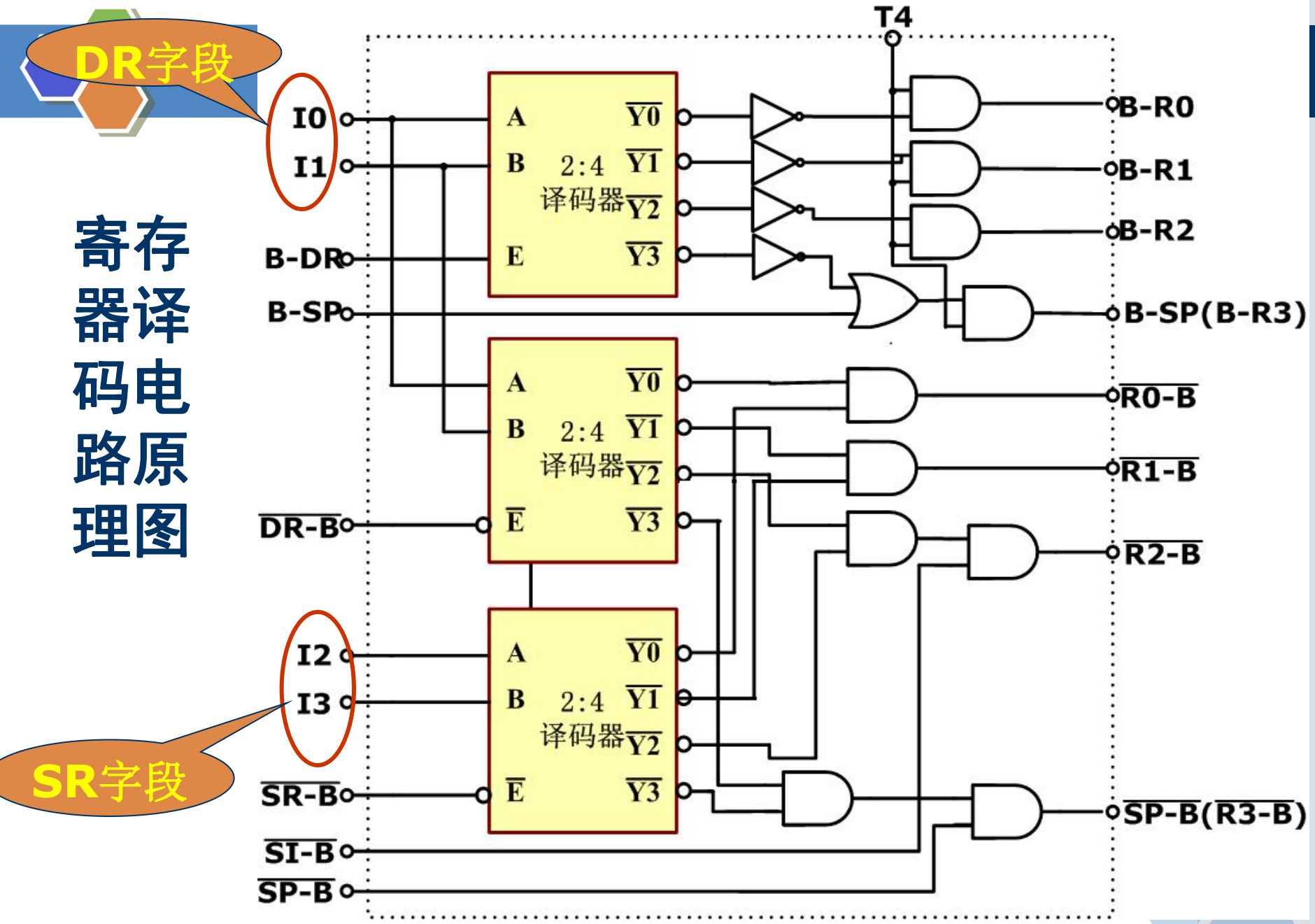
③ 指令码I1-I0：来自指令寄存器的DR字段

① 输出信号为：（送至寄存器单元REG UNIT）

① 寄存器打入脉冲：B-R0、B-R1、B-R2、B-SP（B-R3）。

② 寄存器输出控制：R0-B#、R1-B#、R2-B#、SP-B#（R3-B#）。

寄存器译码电路原理图


DR字段
SR字段
T4
I0
I1
B-DR
B-SP
I2
I3
SR-B
SI-B
SP-B
DR-B
A
B
E
2:4
译码器
Y0
Y1
Y2
Y3
B-R0
B-R1
B-R2
B-SP(B-R3)
R0-B
R1-B
R2-B
SP-B(R3-B)

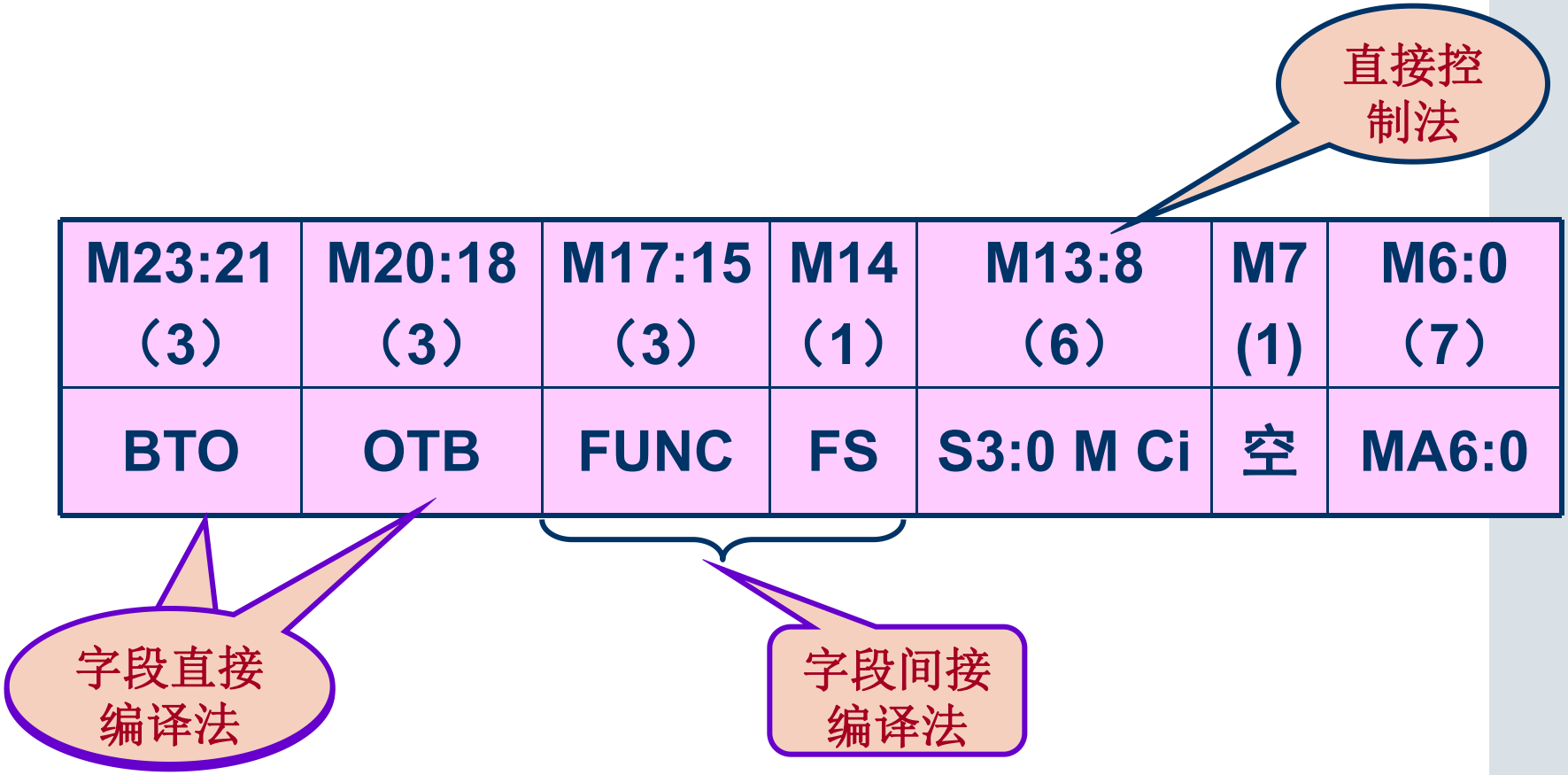
寄存器控制信号的定义

信号	输入/输出	解释
I1 I0	输入	指令码中目的寄存器DR的编码字段
I3 I2	输入	指令码中源寄存器SR的编码字段
B-DR	输入	目的寄存器DR的装载控制信号，与指令码I1I0一起译码产生B-R[0:3]四个信号
DR-B#	输入	目的寄存器DR内容送总线的控制信号，与指令码I1I0一起译码产生R[0:3]-B#四个信号
SR-B#	输入	源寄存器SR内容送总线的控制信号，与指令码I3I2一起译码产生R[0:3]-B#四个信号
RI-B#	输入	变址寄存器SI内容送总线的控制信号，即R2-B#
B-SP	输入	堆栈指针寄存器SP的装载控制信号，即B-R3
SP-B#	输入	SP内容送总线的控制信号，即R3-B#
B-R[0:3]	输出	寄存器R0、R1、R2、R3从总线上装入数据
R[0:3]-B#	输出	将寄存器R0、R1、R2、R3的内容送至总线上

寄存器控制信号的产生逻辑

输入	输出	输入	输出
B-DR=1，且I1I0=00	B-R0=1	SR-B#=0，且I3I2=00	R0-B#=0
B-DR=1，且I1I0=01	B-R1=1	SR-B#=0，且I3I2=01	R1-B#=0
B-DR=1，且I1I0=10	B-R2=1	SR-B#=0，且I3I2=10	R2-B#=0
B-DR=1，且I1I0=11	B-R3=1	SR-B#=0，且I3I2=11	R3-B#=0
DR-B#=0，且I1I0=00	R0-B# =0	SI-B#=0	R2-B#=0
DR-B#=0，且I1I0=01	R1-B# =0	B-SP=1	B-R3=1
DR-B#=0，且I1I0=10	R2-B#=0	SP-B#=0	R3-B#=0
DR-B#=0，且I1I0=11	R3-B#=0		

实验模型机的微指令格式定义(24位)



实验模型机微指令字段编码表

字段间接编译法

编码+译码	BTO	OTB	FUNC	
			FS=1	FS=0
000	空	空	PC+1	空
001	B-DA1(t4)	ALU-B# (t2)	J1# (t2)	M-W# (t3)
010	B-DA2(t4)	299-B# (t2)	J2# (t2)	M_R# (t2)
011	B-IR(t3)	SR-B# (t2)	J3# (t2)	I/O-W# (t3)
100	B-DR(t4)	DR-B# (t2)	J4# (t2)	I/O_R# (t2)
101	B-SP(t4)	SI-B# (t2)*	J5# (t2)	INT_R# (t2)
110	B-AR(t3)	SP-B# (t2)*	CyCn# (t2)	INT_E# (t2)
111	B-PC# (t4)	PC-B# (t2)	CyNCn# (t2)	

字段直接编译法

2、微指令下址字段设计方法

（1）
微程序入口地址的产生

（2）
下址字段的设计方法

（1）微程序入口地址的产生

- 微程序入口地址的产生：由指令译码器实现
  - 微程序控制器中的指令译码器的功能：就是根据指令操作码来产生该指令的微程序入口地址。
- 实现方法：
  - 映射存储器（MAPROM）：把机器指令的操作码当做MAPROM的地址，读出的MAPROM的数据就是该指令对应的微程序入口地址。（时序逻辑电路）
  - 逻辑电路：逻辑电路的输入是机器指令的操作码，输出就是其微程序入口地址。（组合逻辑电路）

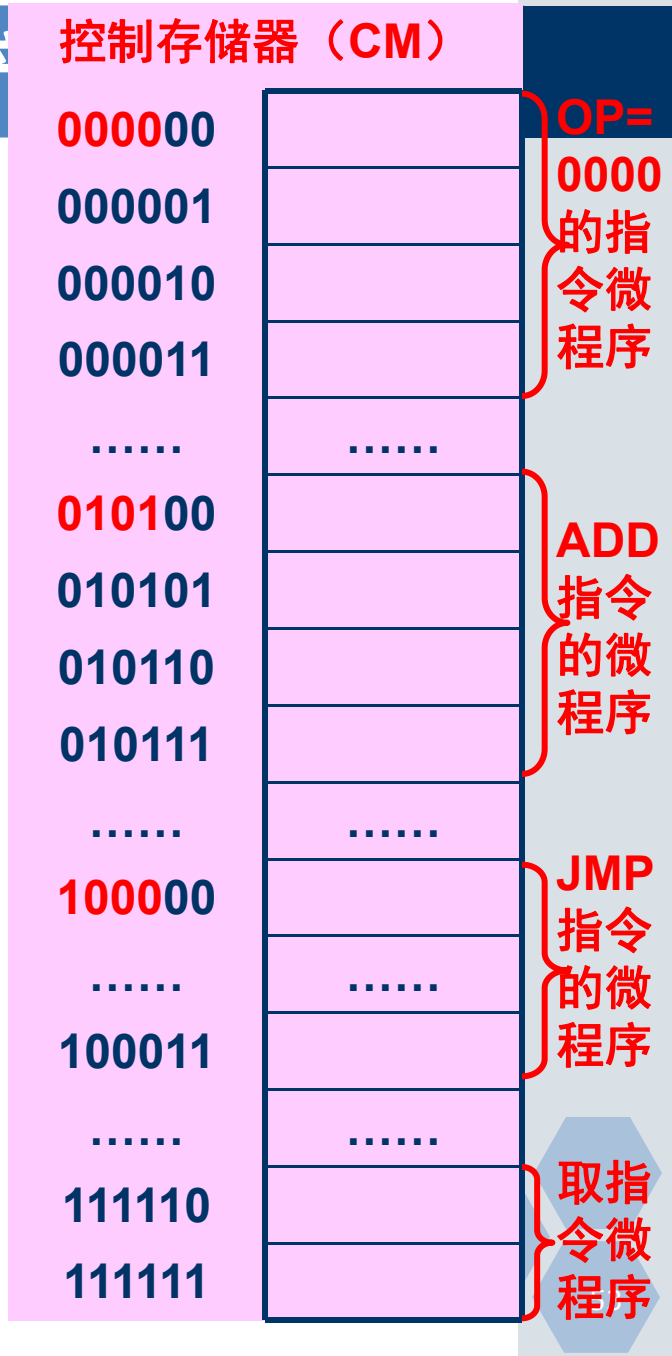
（1）微程序入口地址的产生

- 举例：前述的包含两条指令的指令系统
  - ①采用MAPROM方法：
    - ADD的操作码=0101，入口地址=110，则在MAPROM的5号单元地址中写入6 即可；
    - JMP的操作码=1000，入口地址=010，则在MAPROM的8号单元地址中写入2 即可；
  - 可以扩展指令条数到16条

MAPROM	
0000	
0001	
0010	
0011	
0100	
0101	110
0110	
0111	
1000	010
……	…….
1111	

（1）微程序入口地址的产

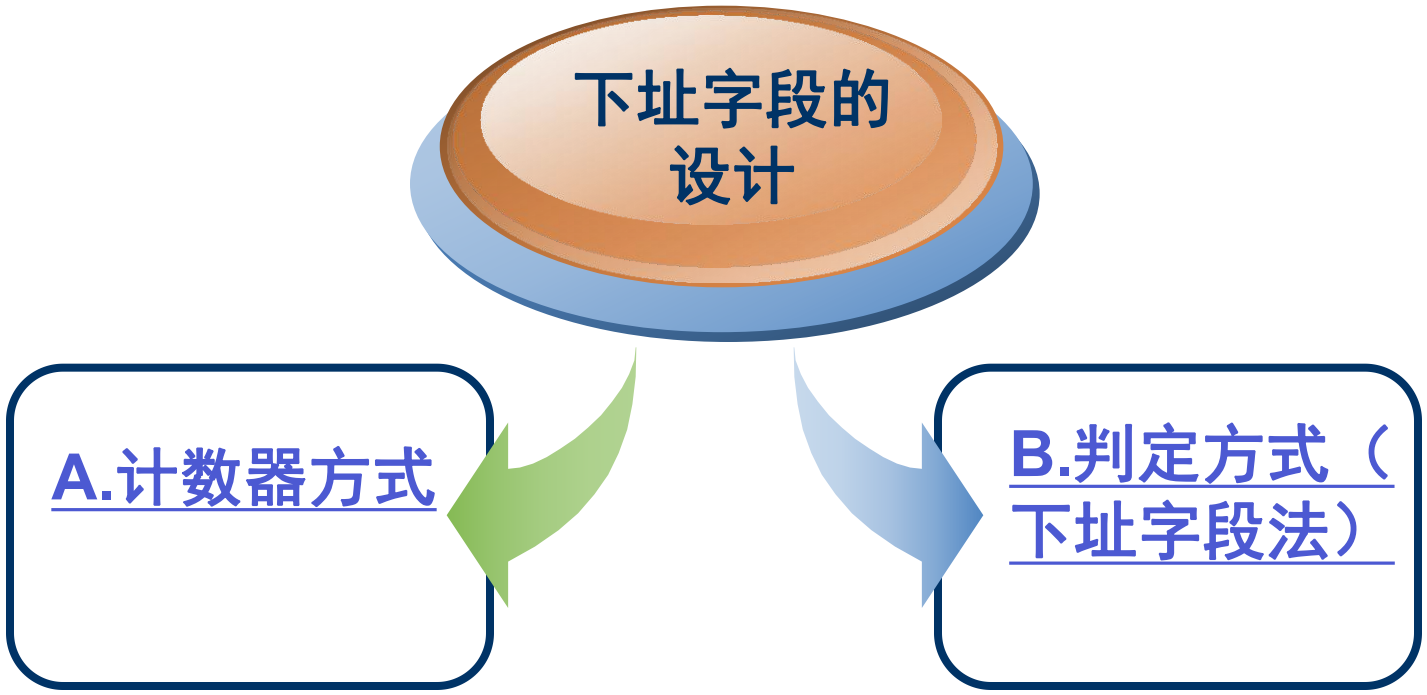
- ②采用逻辑电路方法：
  - 输入：指令操作码OP=$I_7I_6I_5I_4$；
  - 输出：该指令的微程序入口地址=I_4 10
- 只能包含2条指令，如果需要扩充到16条呢？
  - 输入：指令操作码OP=$I_7I_6I_5I_4$
  - 输出：该指令的微程序入口地址=$I_7I_6I_5I_4$ 00



（1）微程序入口地址的产生

- 举例：实验模型机的7位微程序入口地址产生方法：
  - 采用逻辑电路方法：在发送J1#的那条微指令周期的T4节拍进行指令译码
  - 输入：指令操作码6位　I7 I6 I5 I4 I3 I2
  - 输出：
    - 当I7 I6≠11时，微程序入口地址=MA6 MA5 MA4 I7 I6 I5 I4；
    - 当I7 I6=11时，微程序入口地址的逻辑表达式是MA6 1 MA4 I5 I4 I3 I2。
    - MA6~ MA0是发送J1#的那条微指令的下址字段

（2）下址字段的设计



A. 计数器方式

- 在微程序控制器中设置一个微程序计数器μPC；由μPC来提供后继微地址
  - 在顺序执行微指令时， μPC自动+1。
  - 遇到转移微指令时，由微指令给出转移微地址，置入μPC。
- 微指令的格式有两种：

非转移类的微指令格式

0	控制字段

转移类的微指令格式

1	转移类型	下址字段（转移微地址）

A. 计数器方式

- **优点**：微指令字较短，便于编写微程序，后继微地址产生机构比较简单；
- **缺点**：微程序较长，执行速度相对较慢

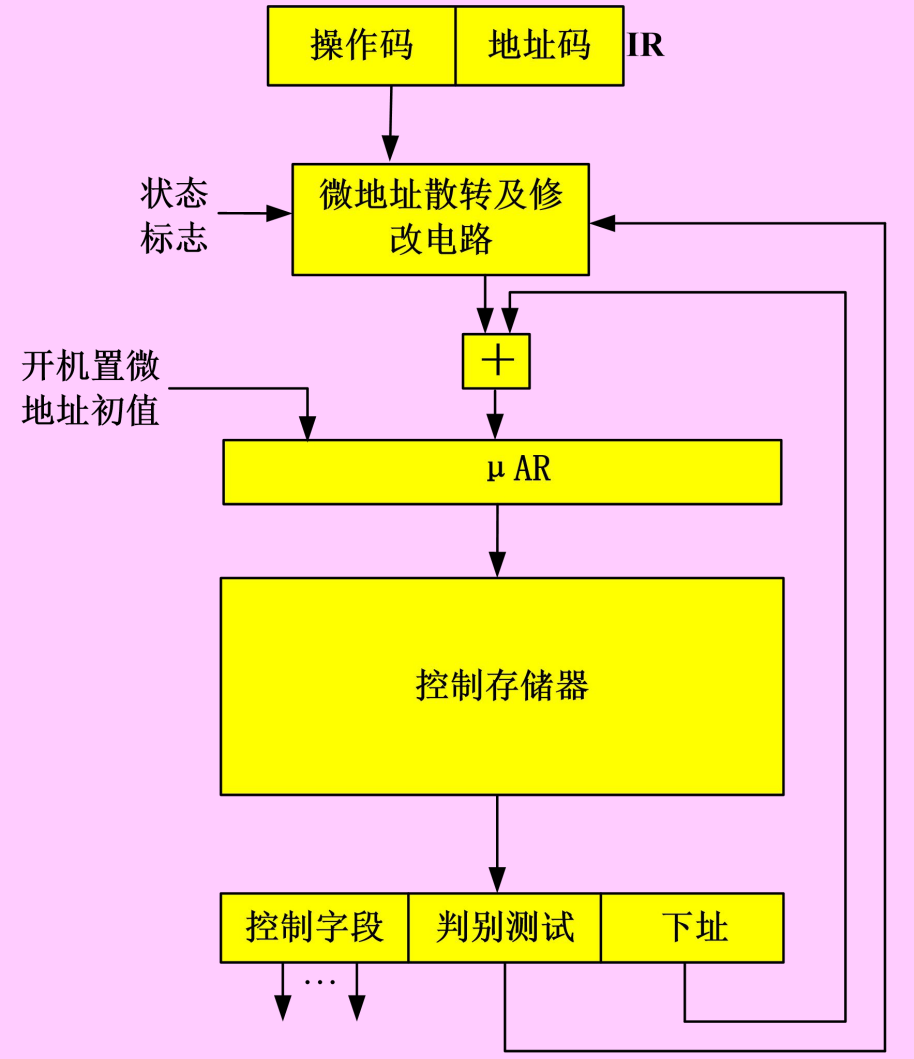
B. 判定方式（下址字段法）

- 微指令格式中设置一个字段用来指明下一条要执行的微指令地址，所以也称为下址字段法。
- 每一条微指令至少都是一条无条件转移微指令，因此不必设置专门的转移微指令。
  - 当微程序不产生分支时，后继微指令地址直接由微指令的下址字段给出；
  - 当微程序出现分支时，按判别测试字段和状态条件通过逻辑电路来形成后继微地址。
- 微指令格式：

控制字段	判别测试字段	下址字段

判定方式产生后继微地址的原理图

- **优点**：可以实现快速多路分支，以提高微程序的执行速度，微程序在控制存储器中的物理分配方便，微程序设计灵活；
- **缺点**：微指令字加长，形成后继微地址的结构比较复杂。



3、微指令格式的类型

① 水平型微指令

- 基本特征是：一条微指令能控制数据通路中多个功能部件并行操作。
- 优点：一条微指令可同时发许多个微命令，且微指令控制字段直接控制，微指令执行效率高，速度快，灵活，各部件执行操作的并行能力强；编制的微程序比较短。
- 缺点：微指令字太长，明显地增加了控制存储器的横向容量。
- 控制字段一般采用直接控制法和字段直接控制法。

3、微指令格式的类型

② 垂直型微指令

- 采用**完全编码**的方法，将一套微命令代码化构成微指令。因此，一条微指令只能控制**1～2种**微操作，
- 垂直型微指令的格式

微操作码	源地址	目标地址	其他

- **优点**：比较直观，容易掌握和便于使用；微指令字短，减少了横向控制存储器的容量。
- **缺点**：微指令要经过译码才能发出微命令，微指令的执行效率低，**并行操作性比较差**，增加了纵向微程序容量。

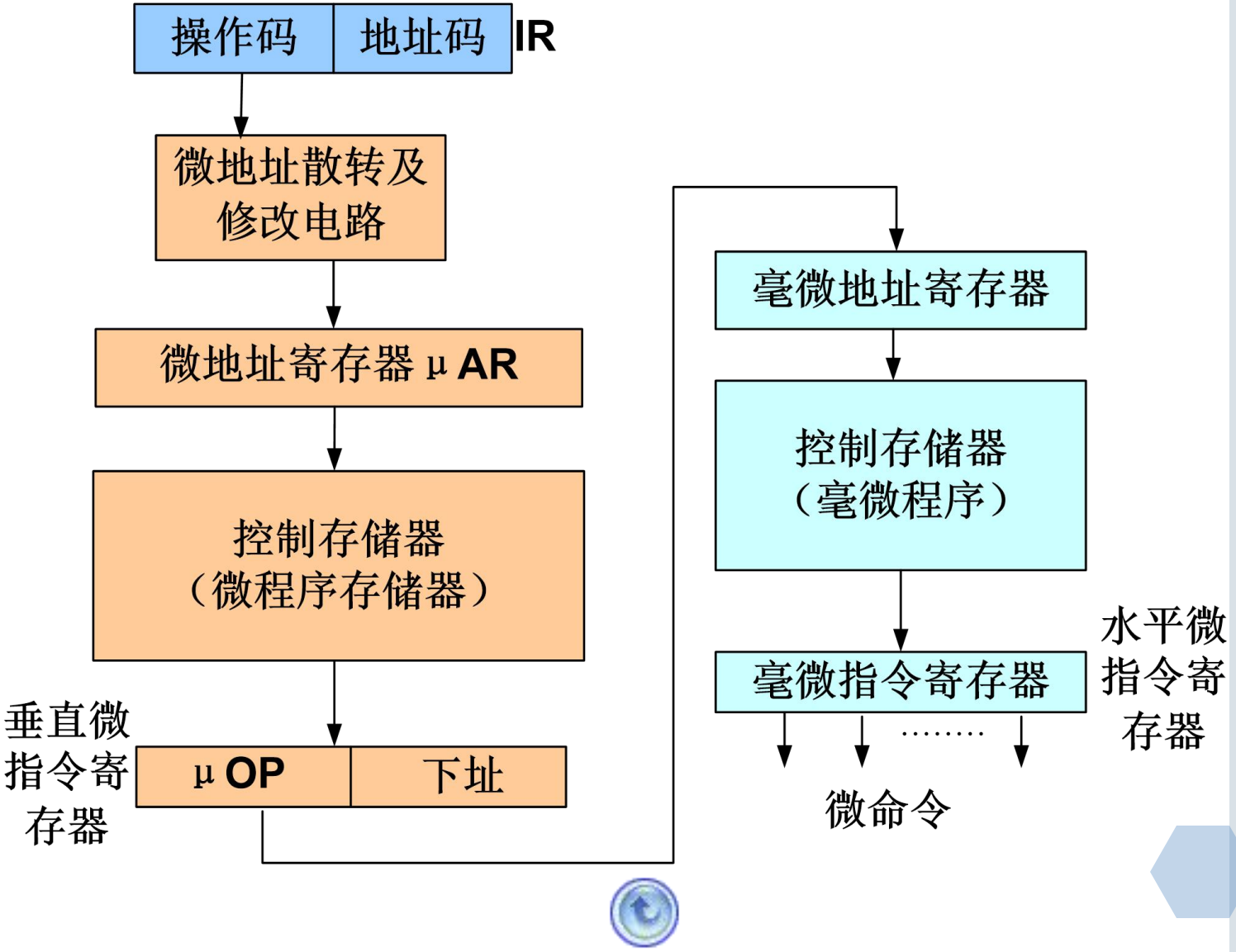
4、控制存储器和动态微程序设计

- **控制存储器：一般由ROM构成，**因为指令系统一般是固定的，微程序是解释执行指令的，因此**微程序一般也是固定的，**所以使用只读存储器来存放微程序。
- **动态微程序设计：**在一台微程序控制的计算机中，假如**能根据用户的要求改变微程序，**那么这台机器就具有动态微程序设计功能。
  - 具有动态微程序设计功能的控制器的**CM必须是可改写的存储器，**如RAM或者E^2PROM。
  - 动态微程序设计可以通过修改微程序来**实现不同的指令系统，**或者**实现指令系统的扩充或调整。**
  - 动态微程序设计的目的是使计算机能更灵活、更有效地适应于各种不同的应用场合。

5、毫微程序设计

- 将垂直型微指令设计和水平型微指令设计结合起来，采用两级微程序来实现指令系统：
  - 第一级为垂直微程序：用来解释机器指令，称为微程序并存放在称为微程序存储器的控存（一级控存）中。
  - 第二级为水平微程序，用来解释垂直微指令，并产生相应微命令，实现数据通路的控制。由于它是解释微程序的微程序，所以称为毫微程序，存放在称为毫微程序存储器的控存（二级控存）中。
  - 毫微程序设计使得微程序流的控制和微命令发出完全分离，微程序流控制由微程序级实现（垂直型微指令），而微命令则由毫微程序产生（水平型微指令）。

毫微程序控制器结构


操作码
地址码
IR
微地址散转及修改电路
微地址寄存器 μAR
控制存储器（微程序存储器）
垂直微指令寄存器
μOP
下址
毫微地址寄存器
控制存储器（毫微程序）
毫微指令寄存器
.....
水平微指令寄存器
微命令

作业

- P347：2、3、5、8、9、11、12（1）、15

The End!