第五章　中央处理器

内容简介

- CPU的功能和组成
- 指令周期的概念
- 时序信号的作用、同步/异步控制方式
- 微程序控制器
- 硬连线控制器
- 传统CPU、流水CPU

重点内容

- 基本概念

  微命令、微操作、微指令、微程序

  指令流水线、算术流水线、处理机流水线
- CPU的功能
- 熟悉典型模型机结构、指令和数据的传输通路，分解指令周期流程图
- 指令周期、机器周期、时钟周期三级时序
- 微程序控制器原理及组成框图
- 流水线中的资源相关、数据相关、控制相关问题

课后作业

P183：2、3、10、13

5.1 CPU的功能和组成

- CPU的功能
- CPU的基本组成
- CPU中的主要寄存器
- 操作控制器与时序产生器

5.1.1 CPU的功能

（1）指令控制

控制程序的执行顺序；

由于程序是一个指令序列，这些指令的执行顺序不能任意颠倒，必须严格按照程序规定的顺序进行。

（2）操作控制

控制器产生取指令、执行指令所需要的全部操作控制信号（微命令），并依序送往相应的部件，从而控制这些部件按指令的要求完成规定的动作。

5.1.1 CPU的功能

（3）时间控制

对各种操作实施时间上的定时；

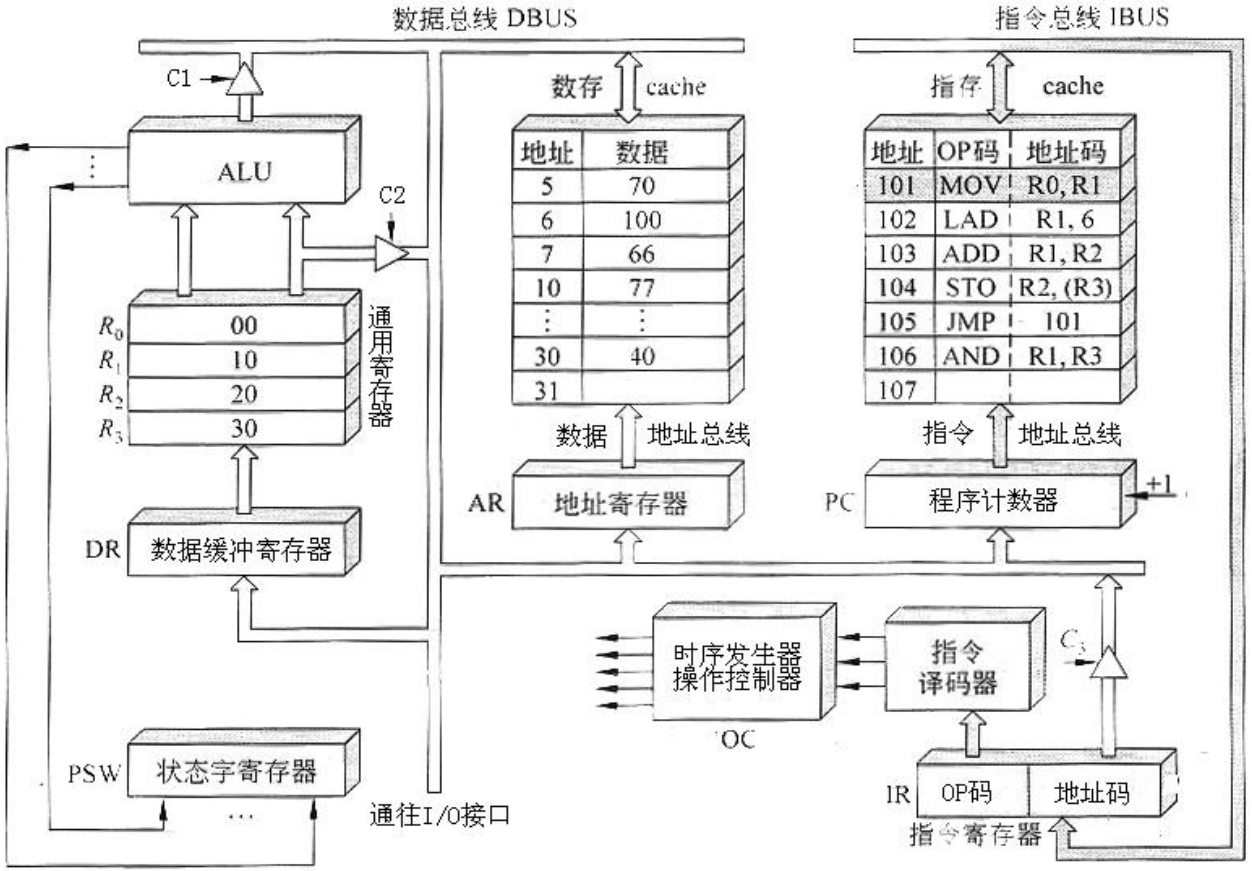
在计算机中，各种指令的操作信号和整个执行过程均受到时间的严格定时和事件先后顺序控制（应在规定的时间点开始，在规定的时间内结束），以保证计算机有条不紊地自动工作。

（4）数据加工

完成指令规定的算术运算、逻辑运算等操作。

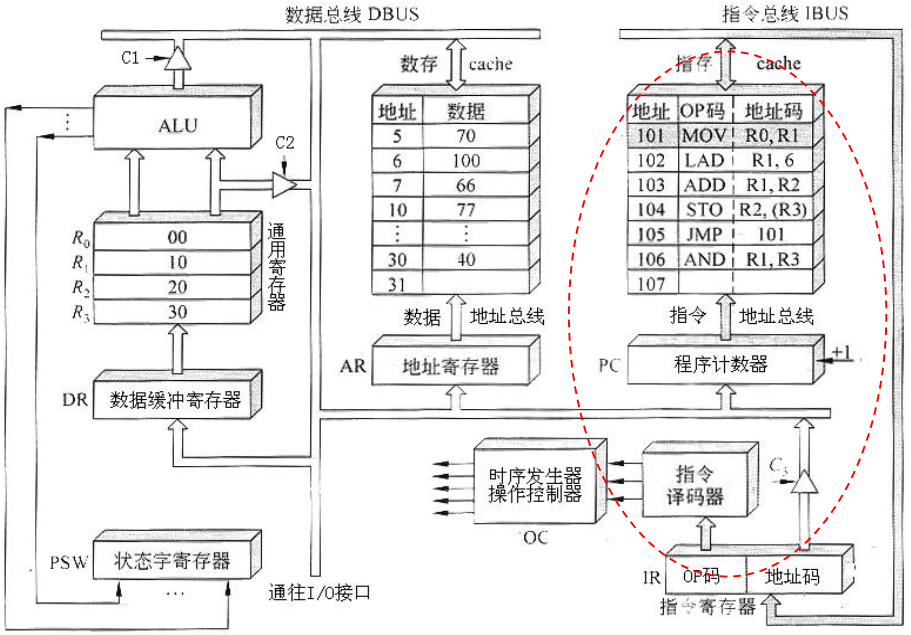
5.1.2 CPU的基本组成

—— 模型机



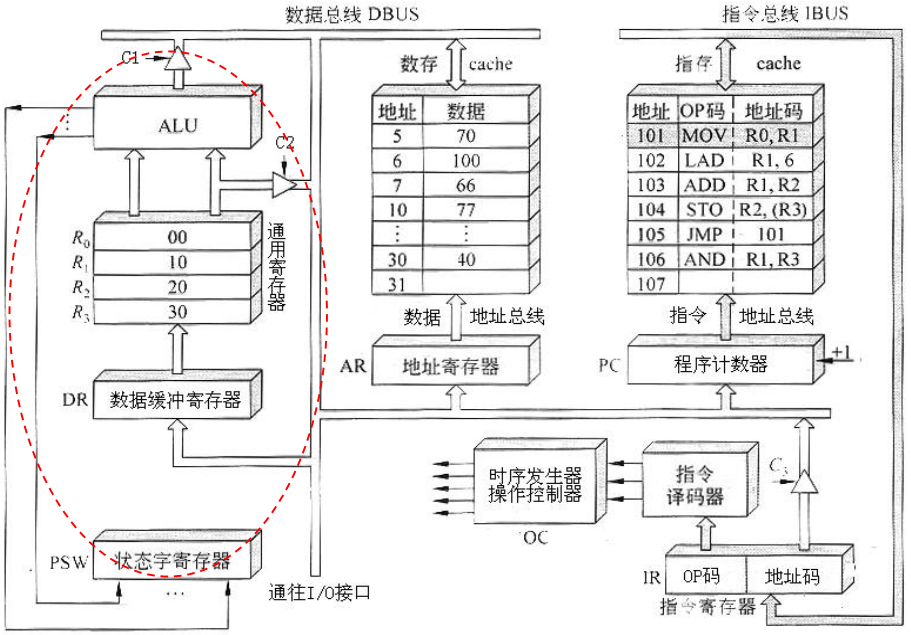
5.1.2 CPU的基本组成

控制部分：由指令cache、程序计数器PC、指令寄存器IR、指令译码器、时序产生器和操作控制器组成。负责取指令、解释指令、产生时序信号和操作控制信号，指挥计算机系统各个部件协调完成相应操作。



5.1.2 CPU的基本组成

执行部分：由算术逻辑单元ALU、通用寄存器R0-R3、地址寄存器AR、数据cache、数据缓冲寄存器DR和状态寄存器PSW组成。接受控制部分的命令，负责取操作数、运算、保存结果等操作。



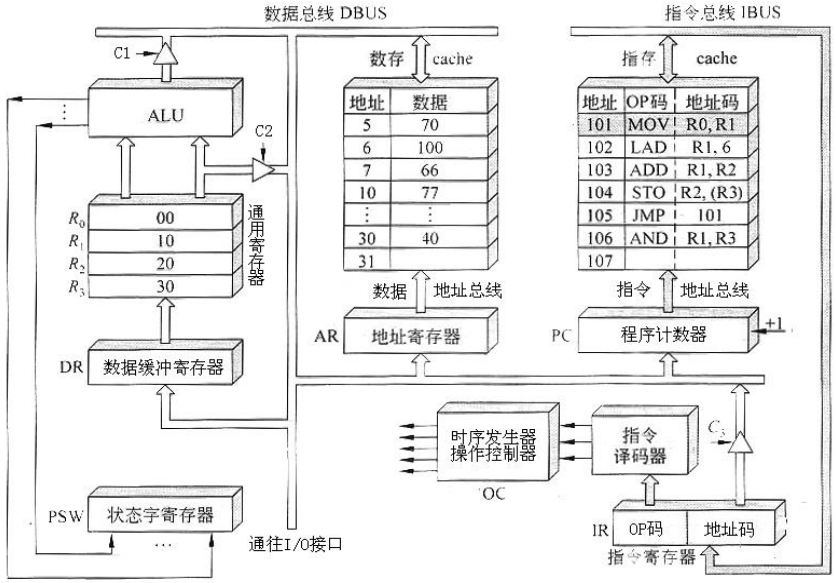
5.1.3 CPU中主要的寄存器

1、程序计数器PC

每次取指令cache，从PC获取地址。

取指令阶段结束后自动“加1”，保证程序顺序执行。

通过一定的策略改变PC，实现程序跳转执行。

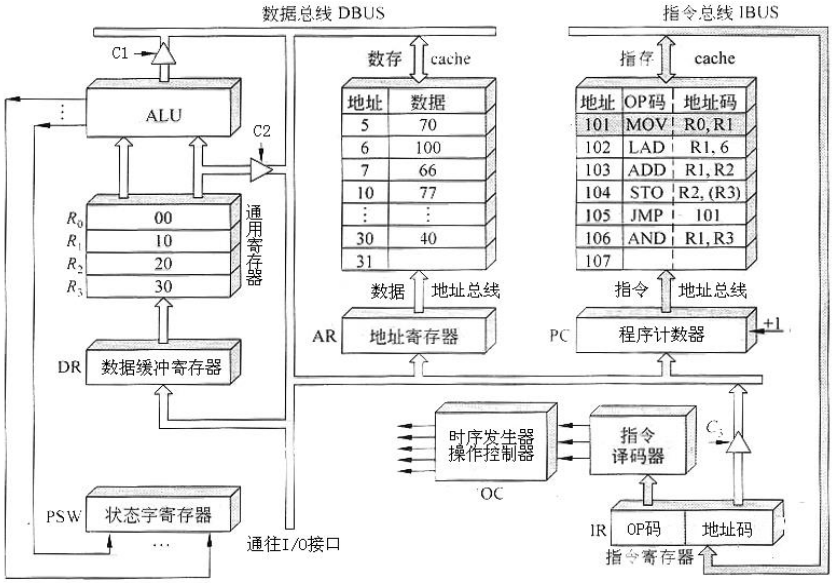


5.1.3 CPU中主要的寄存器

2、指令寄存器IR

存放从指令cache读取的一条指令，作为指令译码器的输入。

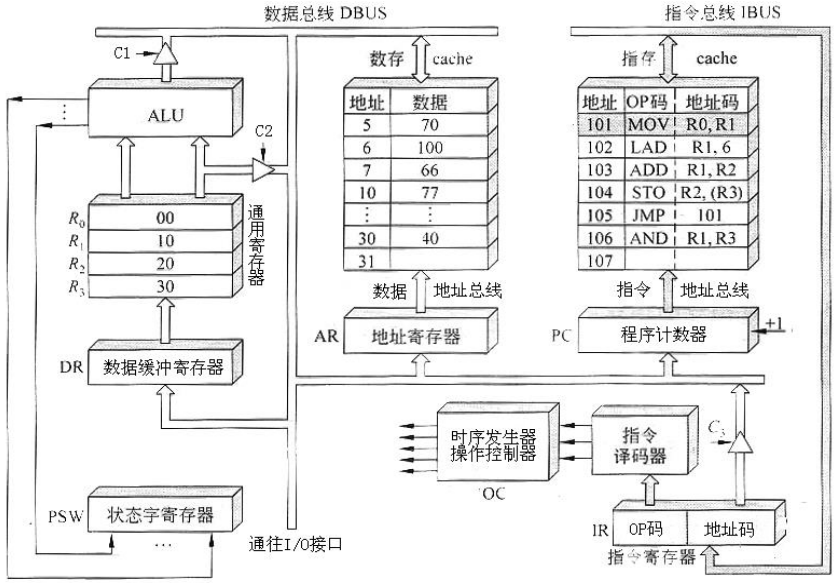
现代计算机已经发展成为指令队列，可以预取若干的指令，从而加快执行速度。



5.1.3 CPU中主要的寄存器

3、地址寄存器AR

访问数据cache时地址输入缓冲，在读写周期内维持地址信息不变。



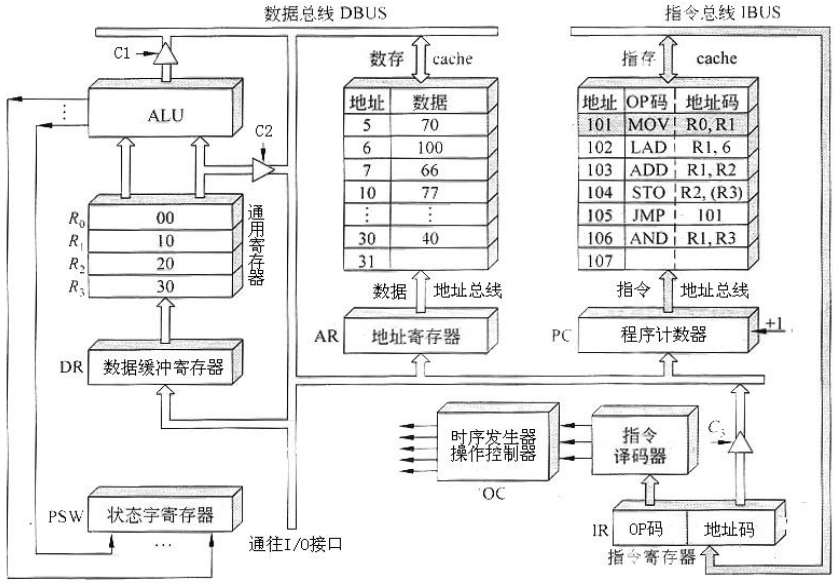
5.1.3 CPU中主要的寄存器

4、数据缓冲寄存器DR

数据cache的I/O缓冲。

ALU运算结果输出缓冲。

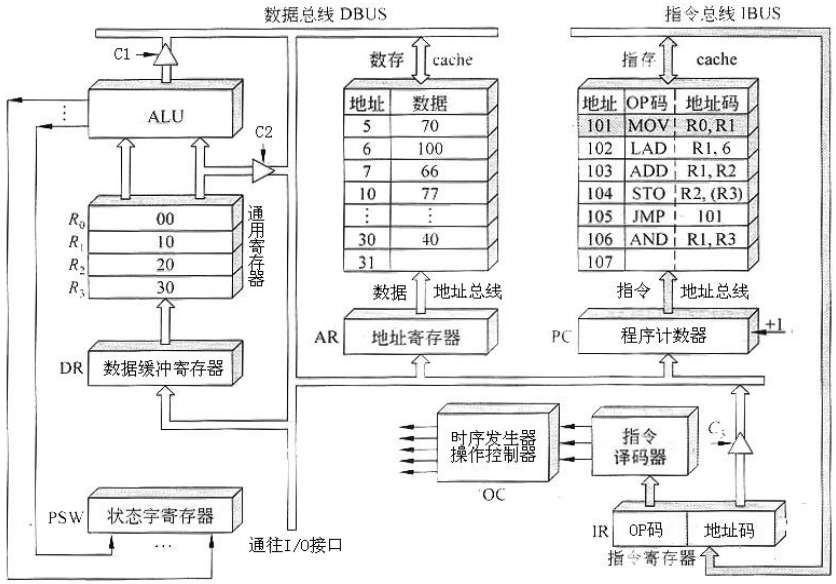
通用寄存器R0-R3的输入缓冲。



5.1.3 CPU中主要的寄存器

5、通用寄存器R0-R3

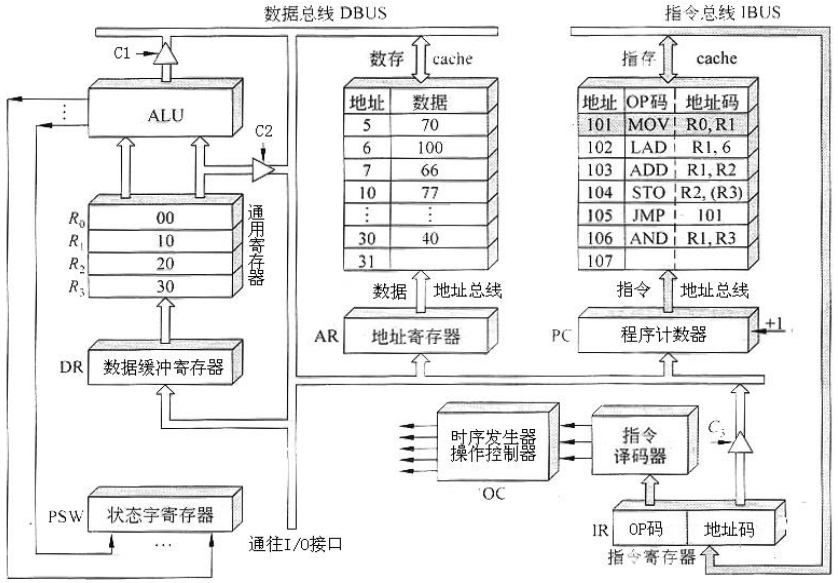
工作寄存器，保存ALU的操作数、运算结果。



5.1.3 CPU中主要的寄存器

6、状态字寄存器PSW

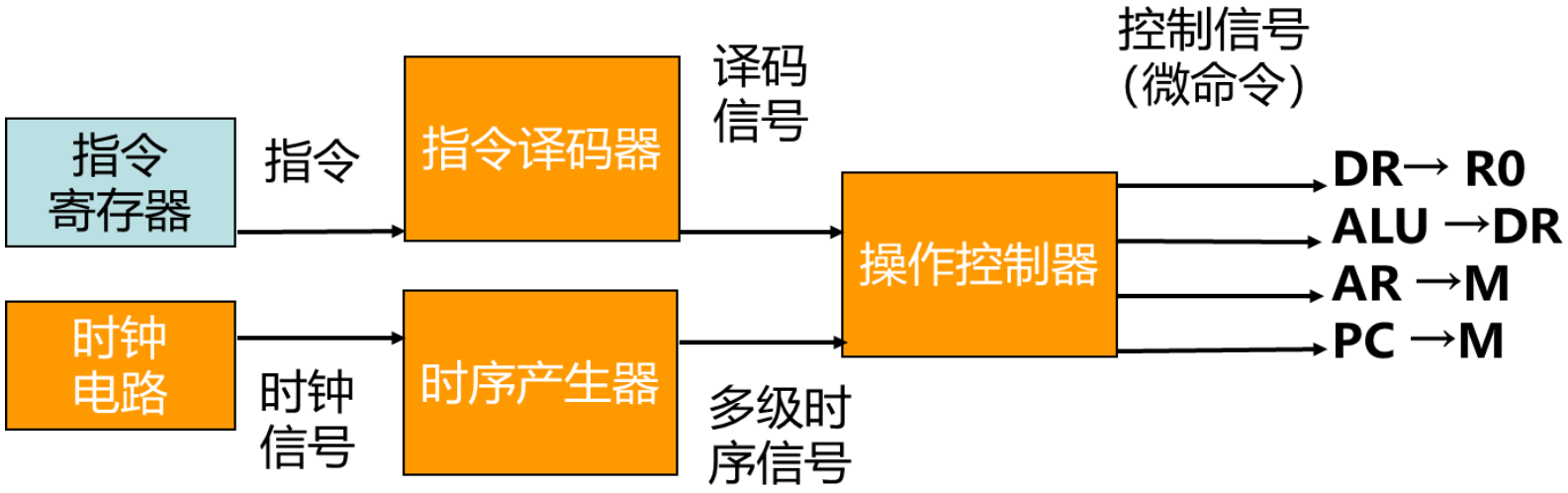
反映算术逻辑运算指令运算结果的各种状态，如进位标志CF、溢出标志OF、零标志ZF、符号标志SF等。



5.1.4 操作控制器与时序产生器

操作控制器：根据指令操作码和时序信号，产生各种操作控制信号。操作控制信号应在规定的时间点开始有效、在规定的时间范围内失效。

时序产生器：根据时钟周期信号，产生各种时间定时信号（指令周期、机器周期、时钟周期），相当于设置各种定时时间的闹钟。



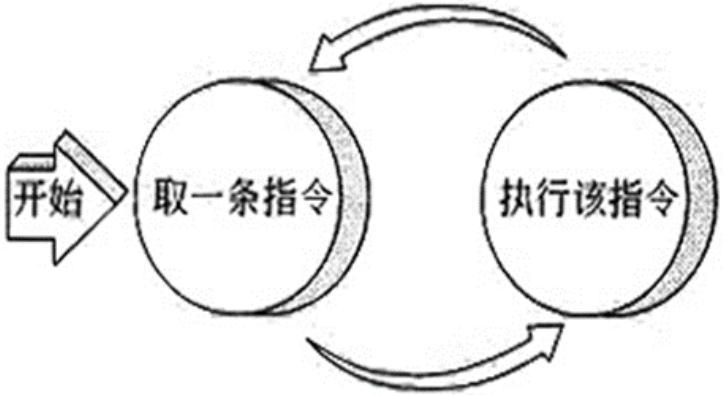
5.2 指令周期

- 指令周期的基本概念
- MOV指令的指令周期
- LAD指令的指令周期
- ADD指令的指令周期
- STO指令的指令周期
- JMP指令的指令周期
- 用方框图语言表示指令周期

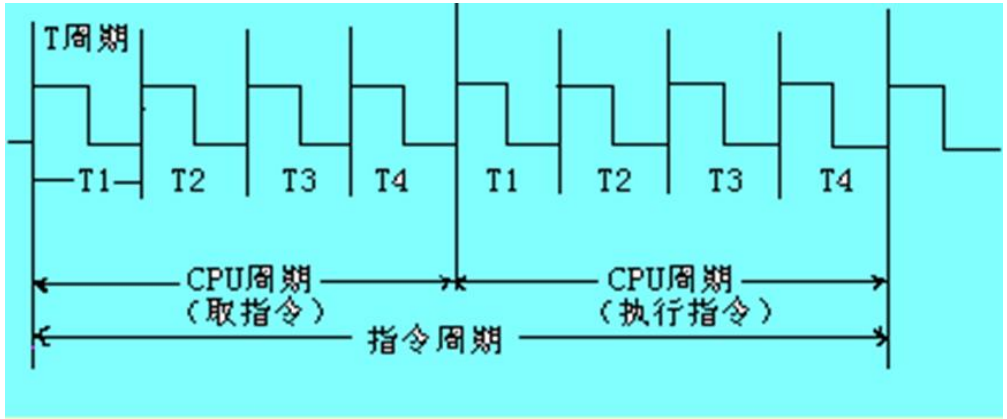
5.2.1 指令周期的基本概念

指令周期：取出一条指令并执行这条指令所需要的时间。

CPU周期：常称为机器周期，每条指令的执行过程可划分为若干个阶段，如取指令、取源操作数、取目的操作数、执行运算、保存结果等阶段，每个阶段所对应的时间。



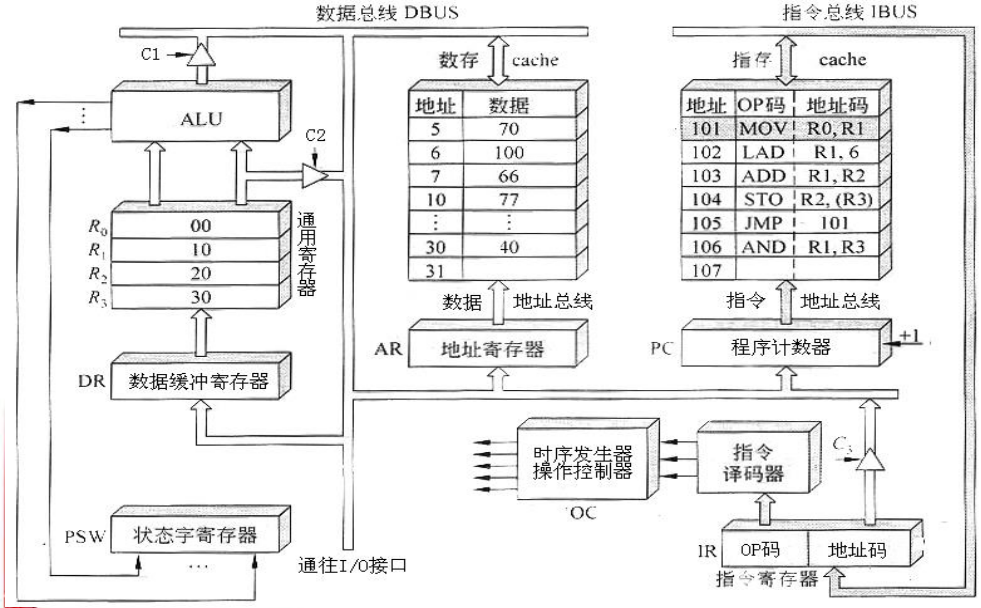
取指令-执行指令序列

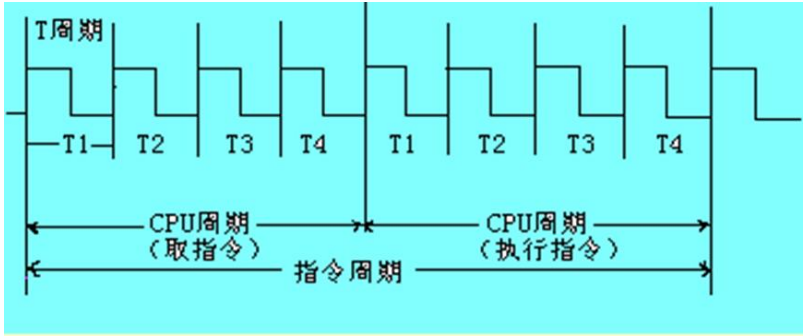


指令周期时序关系

5.2.1 指令周期的基本概念

时钟周期：每个阶段由若干有序的、不可再分的基本操作过程构成，如从数据cache中取一个源操作数送到R0，包括地址→AR、AR→M 、M→DR 、DR→R0等基本操作过程，每个操作过程所对应的时间。





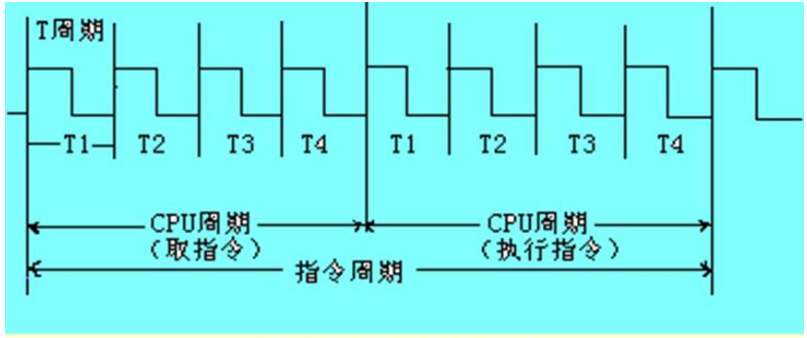
指令周期时序关系

5.2.1 指令周期的基本概念

三级时序：由指令周期、机器周期、时钟周期构成计算机系统的三级时序，相当于有三种定时时间的闹钟。

每个基本操作、每个阶段、每条指令都应在规定的时间开始，在规定的时间结束。

当然，这些规定是在指令系统设计时定义好的。



指令周期时序关系

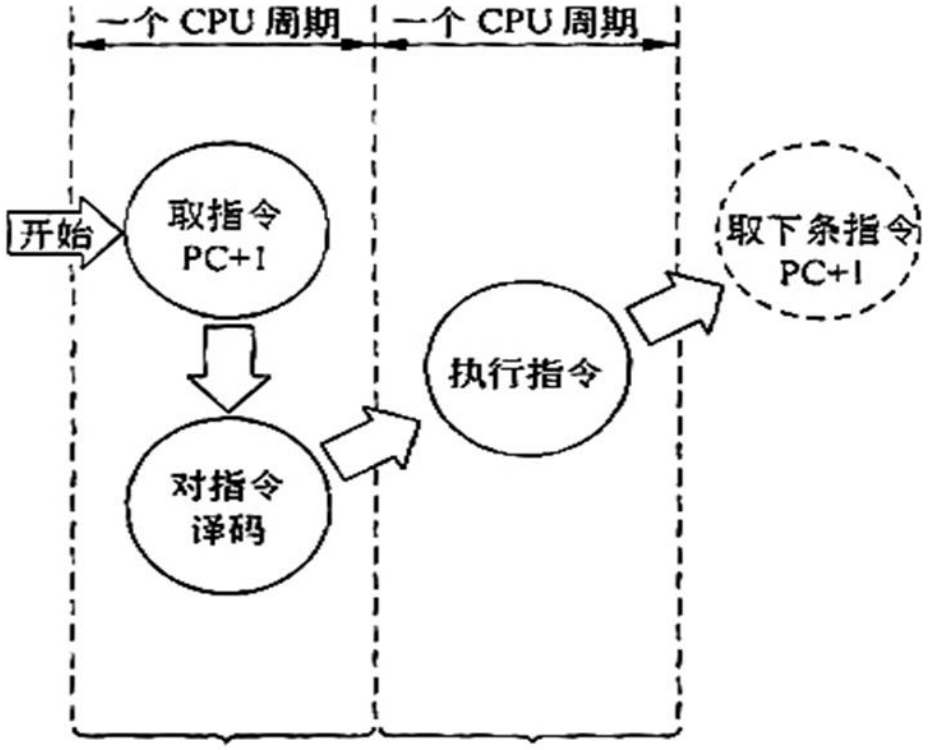
5.2.1 指令周期的基本概念

6 条典型指令组成的一个简单程序

	八进制地址	指令助记符	说　　明
指令存储器	100		① 程序执行前(R0)=00,(R1)=10,(R2)=20,(R3)=30;
	101	MOV R0,R1	② 传送指令 MOV 执行(R1)→R0;
	102	LAD R1,6	③ 取数指令 LAD 从数存 6 号单元取数(100)→R1;
	103	ADD R1,R2	④ 加法指令 ADD 执行(R1)+(R2)→R2,结果为(R2)=120;
	104	STO R2,(R3)	⑤ 存数指令 STO 用(R3)间接寻址,(R2)=120 写入数存 30 号单元;
	105	JMP 101	⑥ 转移指令 JMP 改变程序执行顺序到 101 号单元;
	106	AND R1,R3	⑦ 逻辑乘 AND 指令执行(R1)·(R3)→R3;

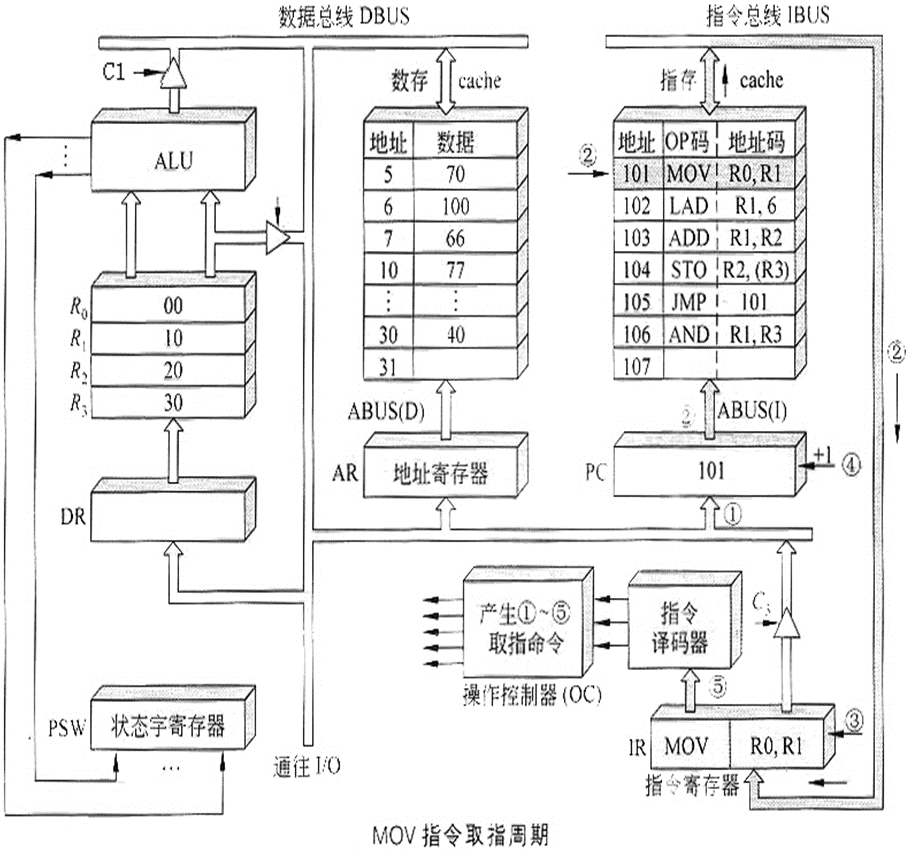
	八进制地址	八进制数据	说　　明
数据存储器	5	70	
	6	100	执行 LAD 指令后,数存 6 号单元的数据 100 仍保存在其中;
	7	66	
	10	77	
	⋮	⋮	
	30	40(120)	执行 STO 指令后,数存 30 号单元的数据由 40 变为 120;

5.2.2 MOV指令的指令周期



MOV 指令的指令周期

5.2.2 MOV指令的指令周期



MOV 指令取指周期

取指周期：

指令地址，（PC）=101

OC→PC，（PC）→指令cache

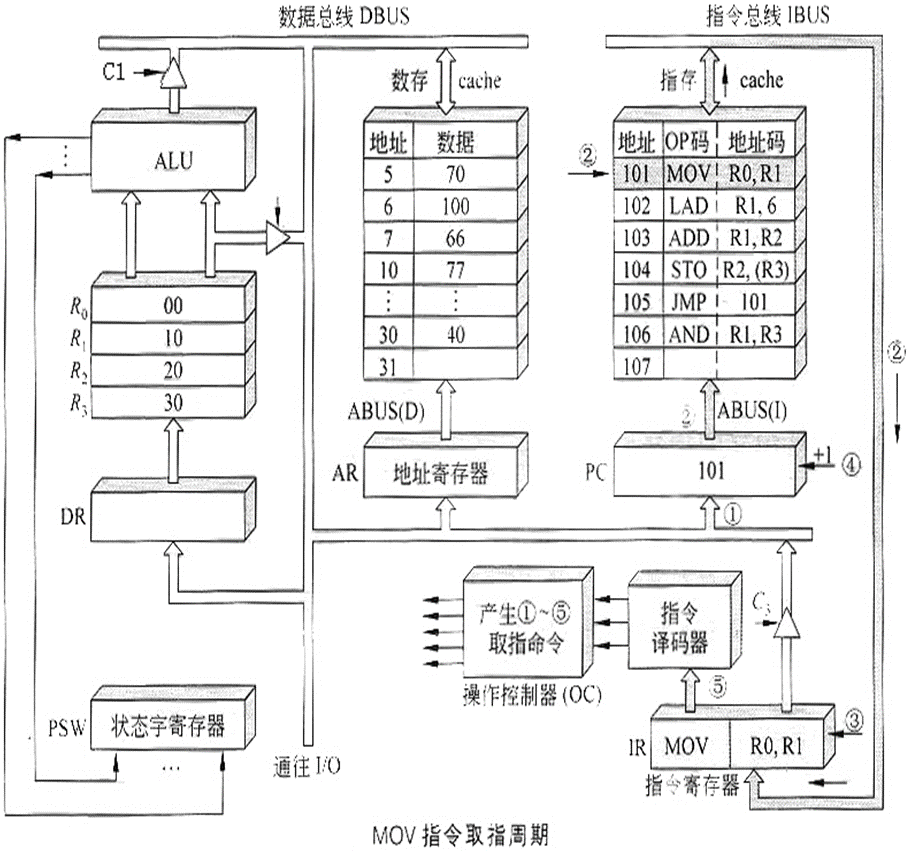
OC→IR，（101）→IBUS→IR

OC→PC，（PC）+1→(PC) =102

OC→指令译码器，（IR）→指令译码器→操作控制器

OC表示操作控制信号

5.2.2 MOV指令的指令周期



MOV 指令取指周期

执行周期：

OC→通用寄存器，R1→ALU

OC→ALU，通知ALU传送操作

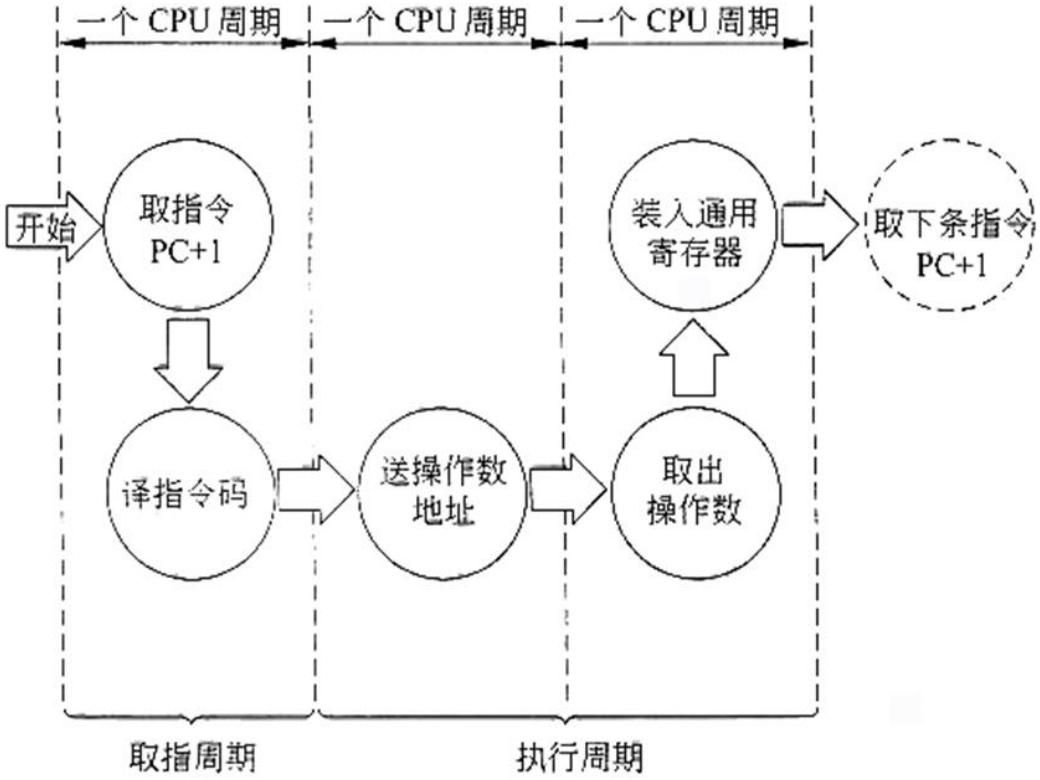
OC→C1，ALU→ DBUS

OC→DR，DBUS→DR

OC→通用寄存器，DR→R0

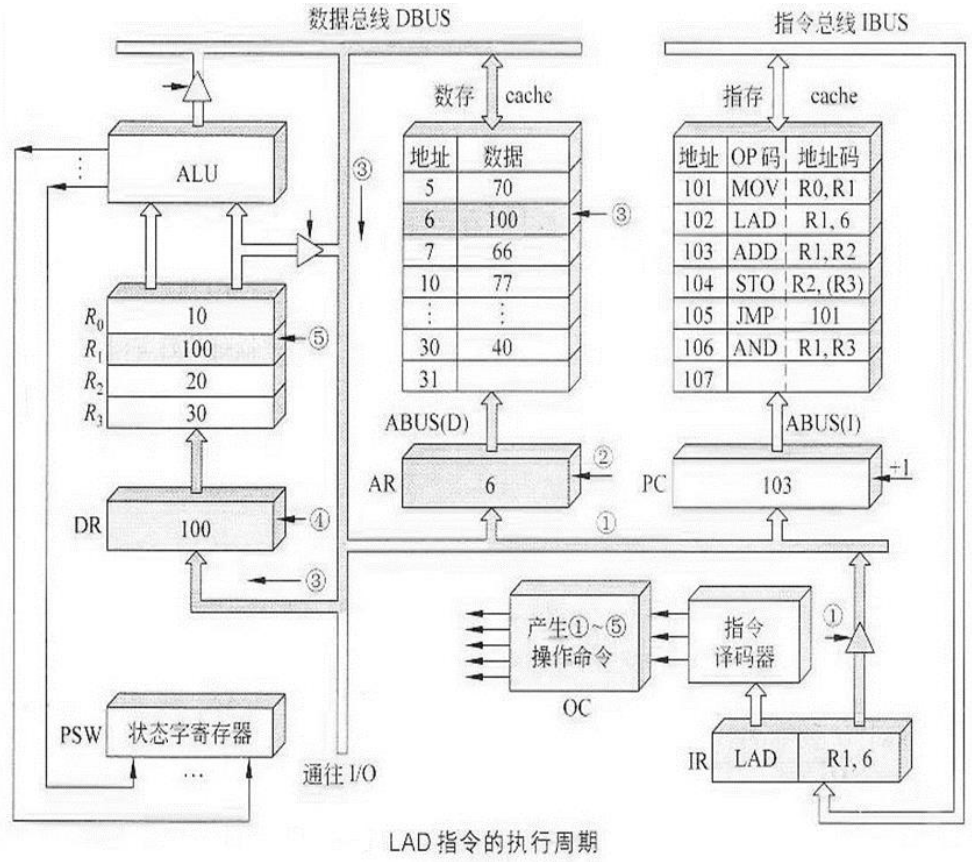
OC表示操作控制信号

5.2.3 LAD指令的指令周期



LAD 指令的指令周期

5.2.3 LAD指令的指令周期



执行周期（CPU周期1）：

OC→C3，IR（地址码）→DBUS

OC→AR，DBUS →AR

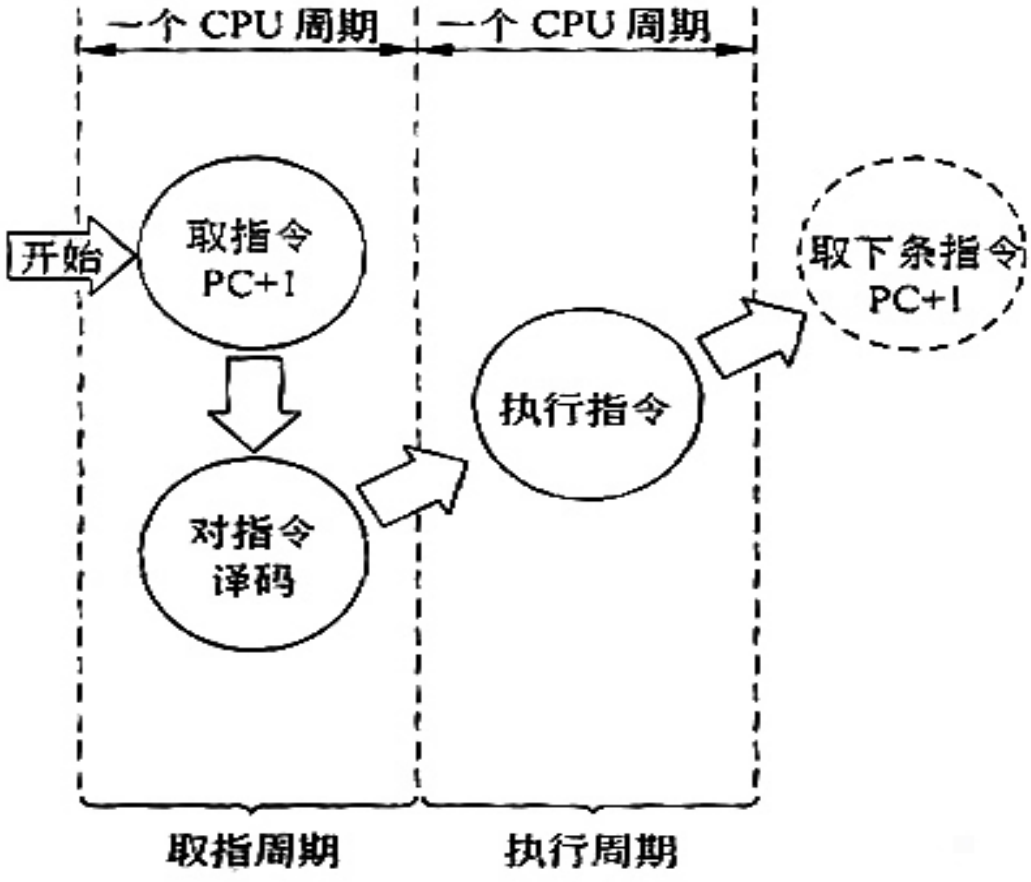
执行周期（CPU周期2）：

OC→读数存，数存→ DBUS

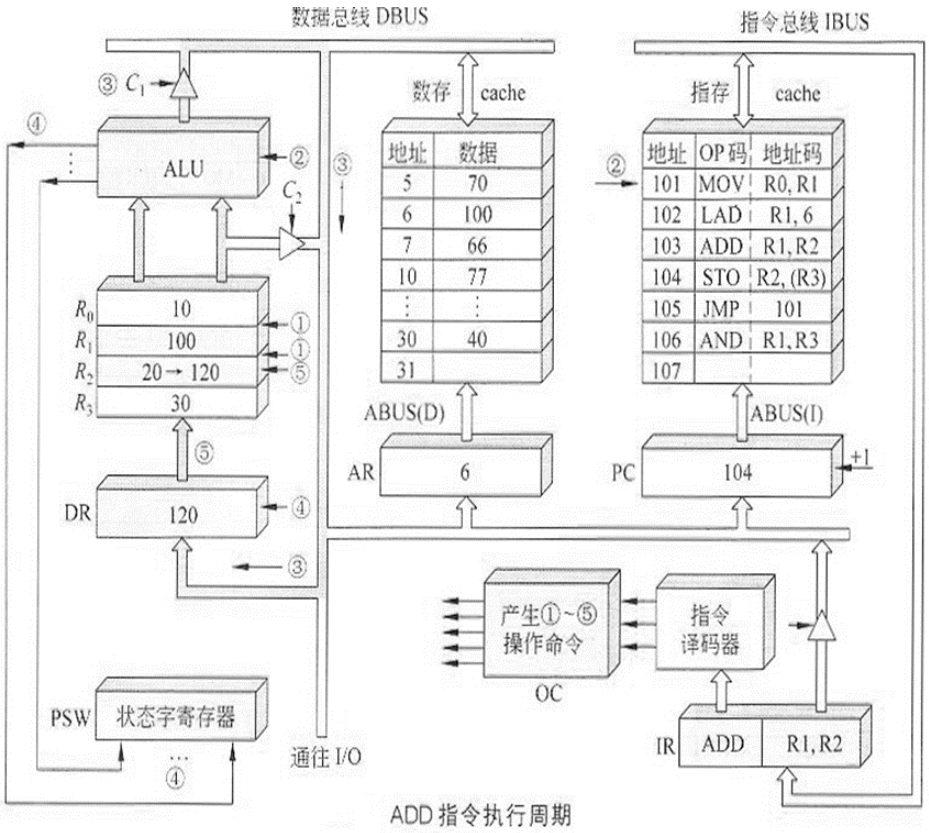
OC→DR，DBUS→DR

OC→通用寄存器，DR→R1

5.2.4 ADD指令的指令周期



5.2.4 ADD指令的指令周期



执行周期：

OC→通用寄存器，R0,R1→ALU

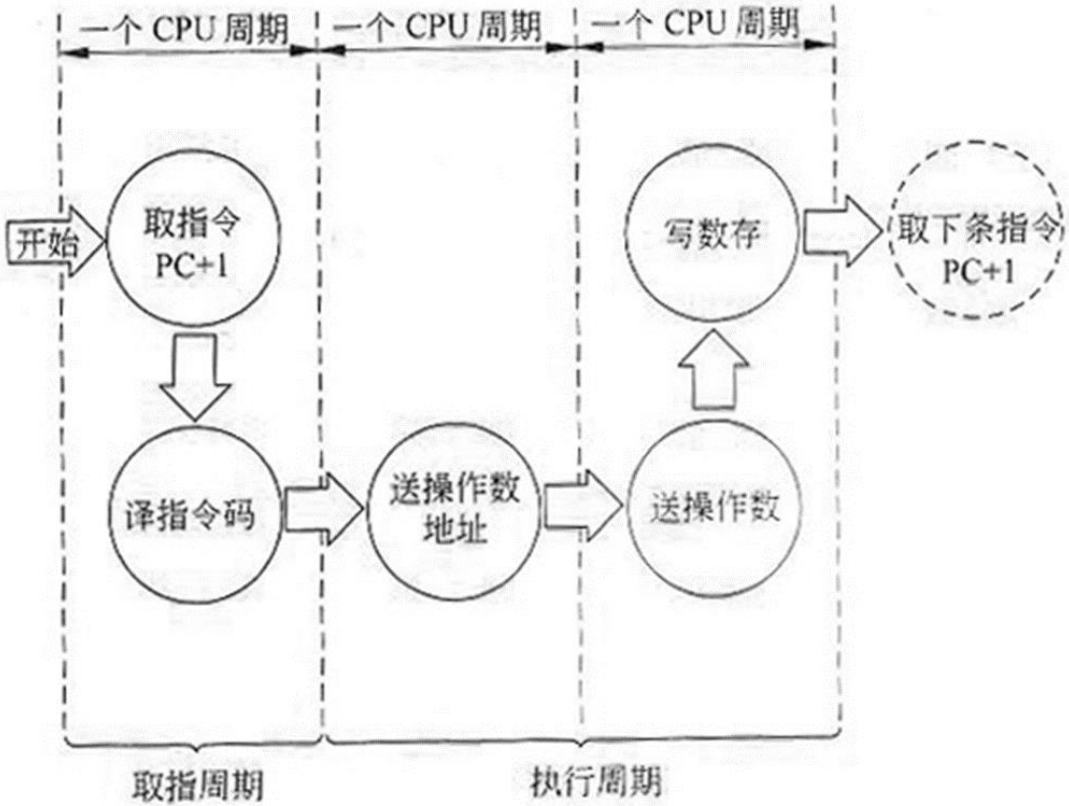
OC→ALU，通知ALU加法操作

OC→C1，ALU→ DBUS,ALU→PSW

OC→DR，DBUS→DR

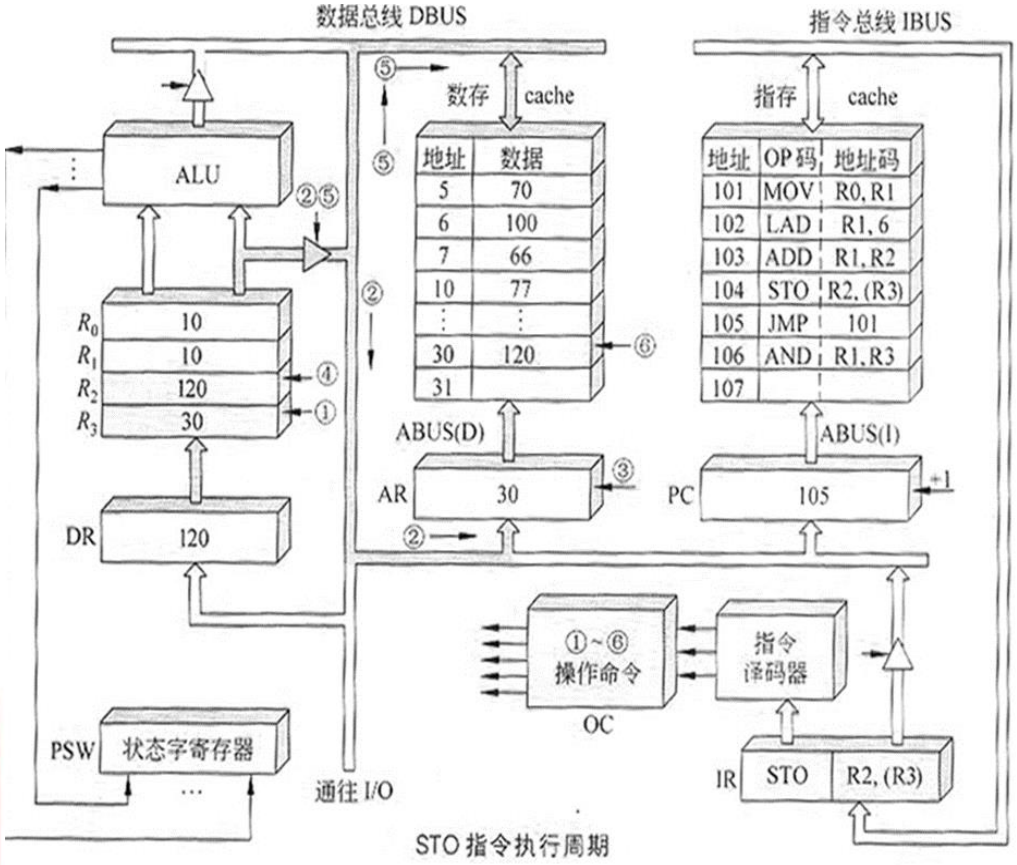
OC→通用寄存器，DR→R2

5.2.5 STO指令的指令周期



STO 指令的指令周期

5.2.5 STO指令的指令周期



执行周期（CPU周期1）：

OC→通用寄存器，选择R3

OC→C2，R3 →DBUS

OC→AR，DBUS→AR

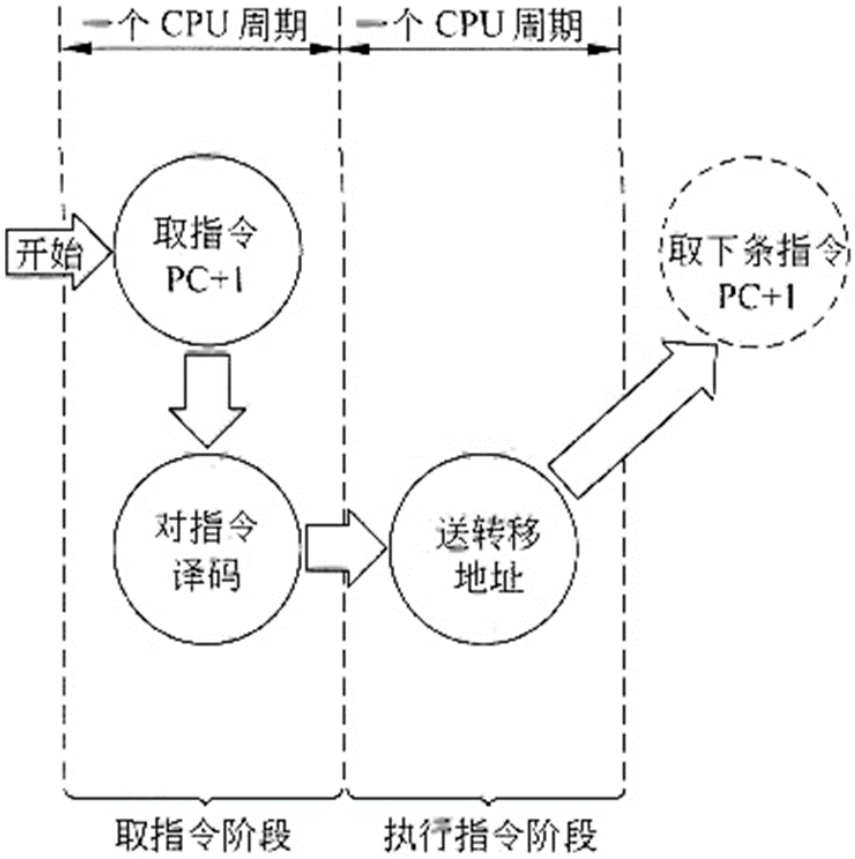
执行周期（CPU周期2）：

OC→通用寄存器，选择R2

OC→C2，R2 →DBUS

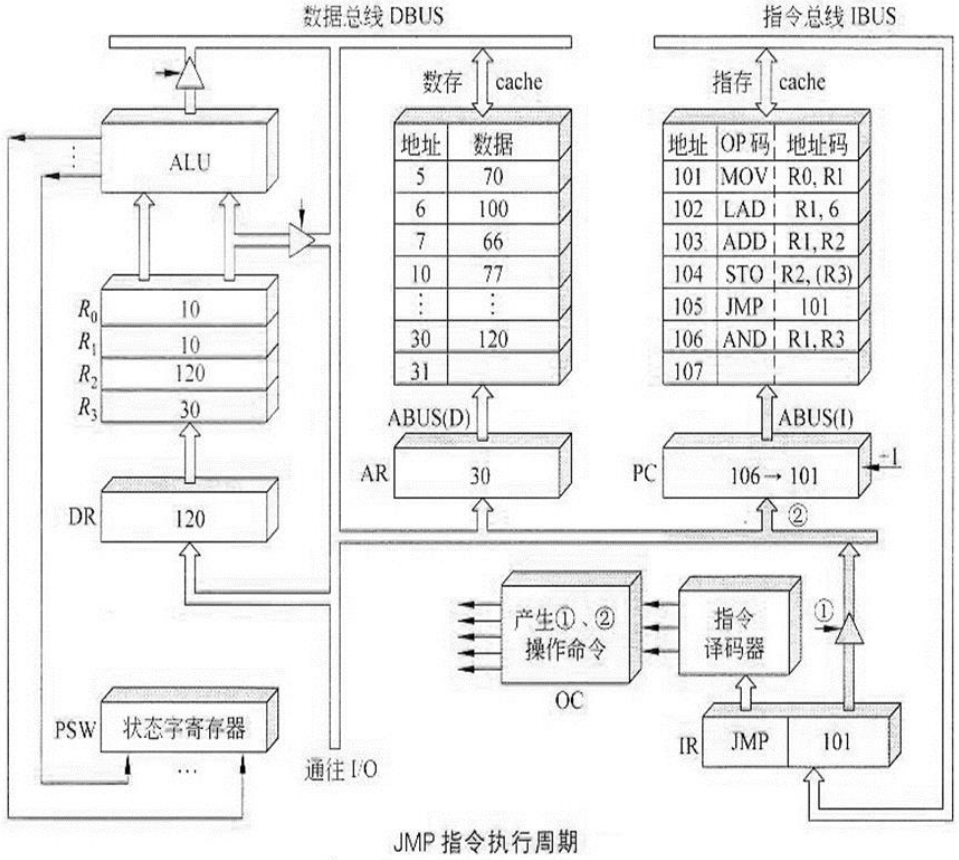
OC→写数存，DBUS→数存

5.2.6 JMP指令的指令周期



JMP 指令的指令周期

5.2.6 JMP指令的指令周期



JMP 指令执行周期

执行周期：

OC→C3，IR（地址码）→DBUS

OC→PC，DBUS →PC=101

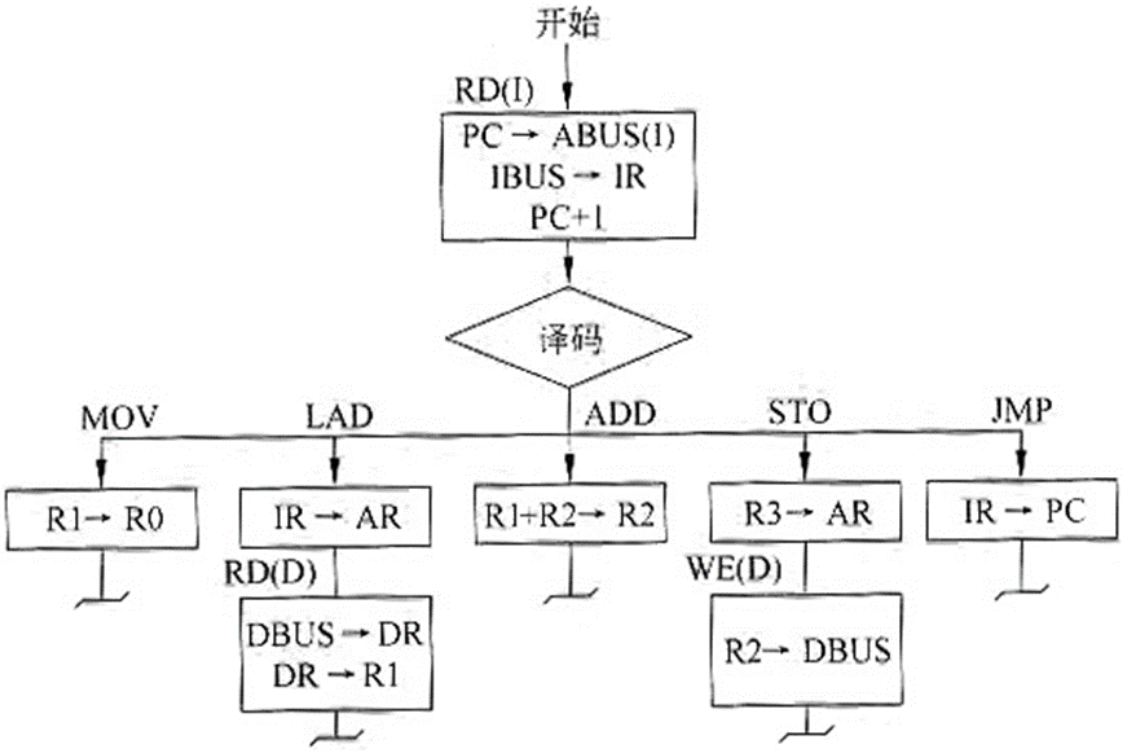
5.2.7 用方框图语言表示指令周期

```
MOV   R0,R1
LAD   R1,6
ADD   R1,R2
STO   R2,(R3)
JMP   101
AND   R1,R3
```

取指周期

执行周期1

执行周期2

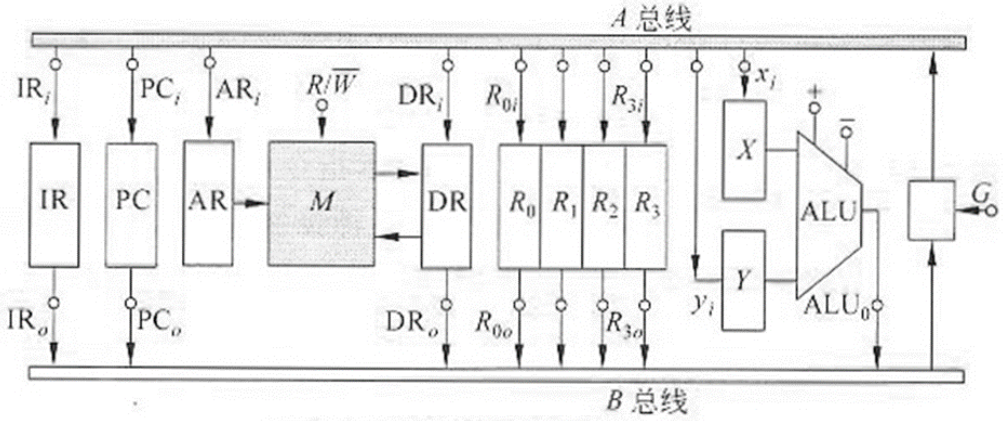


用方框图语言表示指令周期

指令周期流程图

5.2.7 用方框图语言表示指令周期

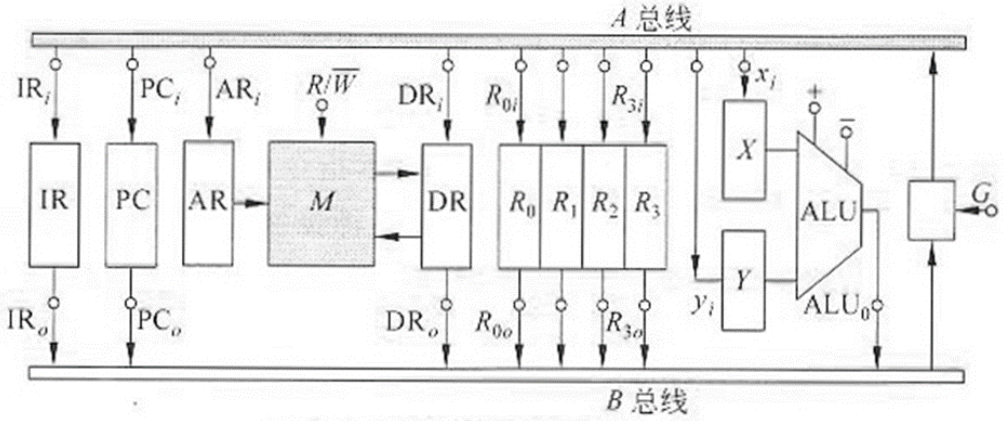
【例】教材图5.15所示为双总线结构机器的数据通路，IR为指令寄存器，PC为程序计数器(具有自增功能)，M为主存(受R/$\overline{W}$信号控制)，AR为地址寄存器，DR为数据缓冲寄存器， ALU由加、减控制信号决定完成何种操作，控制信号G控制的是一个门电路。另外，线上标注有小圈表示有控制信号，例中Yi表示Y寄存器的输入控制信号，R1o为寄存器R1的输出控制信号，未标字符的线为直通线，不受控制。



双总线结构机器的数据通路

5.2.7 用方框图语言表示指令周期

【例】教材图5.15所示为双总线结构机器的数据通路，IR为指令寄存器，PC为程序计数器(具有自增功能)，M为主存(受R/$\overline{W}$信号控制)，AR为地址寄存器，DR为数据缓冲寄存器， ALU由加、减控制信号决定完成何种操作，控制信号G控制的是一个门电路。另外，线上标注有小圈表示有控制信号，例中Yi表示Y寄存器的输入控制信号，R1o为寄存器R1的输出控制信号，未标字符的线为直通线，不受控制。

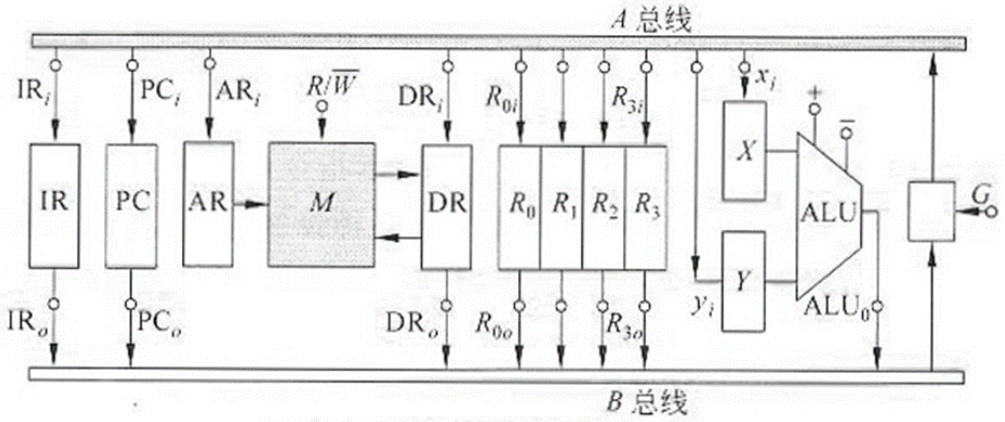


双总线结构机器的数据通路

5.2.7 用方框图语言表示指令周期

【例】(1) “ADD R2，R0”指令完成(R0)+(R2)→R0的功能操作，画出其指令周期流程图，假设该指令的地址已放入PC中。并列出相应的微操作控制信号序列。

(2) “SUB R1，R3”指令完成(R3)-(R1)→R3的操作，画出其指令周期流程图，并列出相应的微操作控制信号序列。



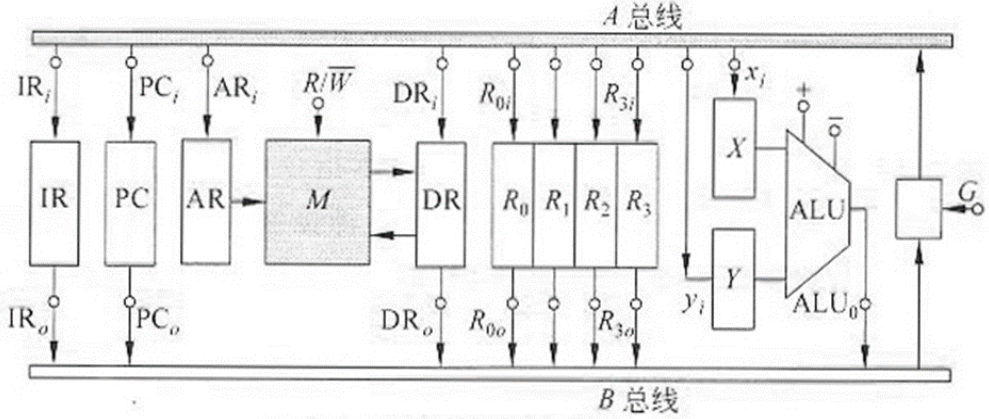
双总线结构机器的数据通路

5.2.7 用方框图语言表示指令周期

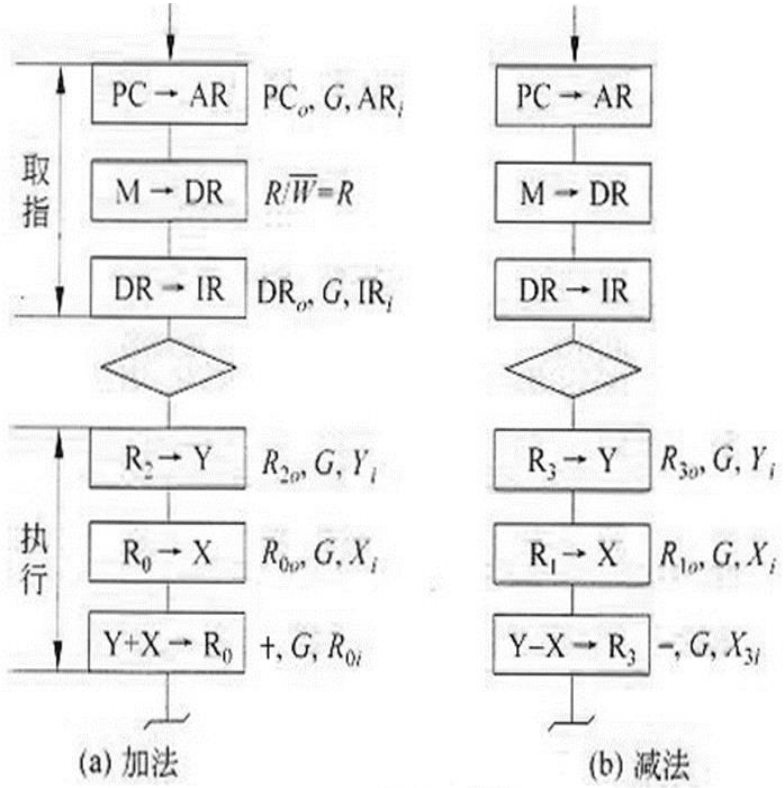
【解】

(1) “ADD R2，R0”；　(R0)+(R2)→R0

(2) “SUB R1，R3”；　(R3)- (R1)→R3



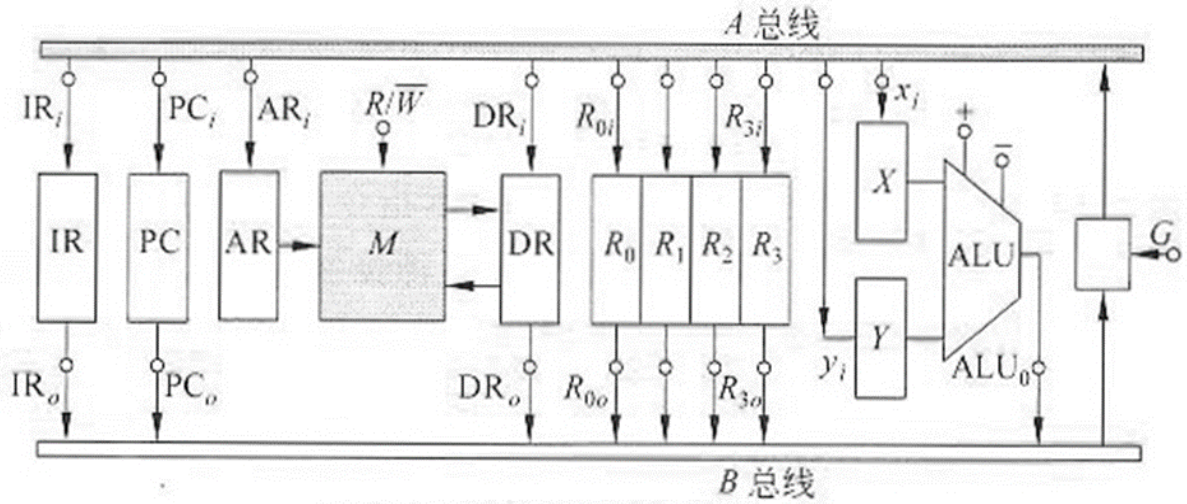
双总线结构机器的数据通路



加法和减法指令周期流程图

5.2.7 用方框图语言表示指令周期

【拓展】 ADD R0，（R1+R2）



双总线结构机器的数据通路

5.3 时序产生器和控制方式

- 时序信号的作用和体制
- 控制方式

5.3.1 时序信号的作用和体制

1、日常生活中的时间控制

在日常生活中，我们学习、工作和休息都有一个严格的作息时间表。

如在教学活动中，每个老师和学生都必须严格遵守作息时间和课表，在规定的时间里上课，在规定的时间里下课，否则就难以保证正常的教学秩序。

5.3.1 时序信号的作用和体制

2、计算机中的时间控制

CPU中也有一个类似“作息时间”的东西，它称为时序信号，使计算机可以准确、有条不紊地工作。

操作控制器发出不同时间间隔的各种定时信号（闹钟响），有条不紊地指挥各部件的动作，规定在这个信号到来时做什么，在那个信号到来时又做什么，给计算机各部分提供工作所需的各种时间标志（闹钟）。

5.3.1 时序信号的作用和体制

3、多级时序体制

前面讲过，指令周期对应一条指令，CPU周期对应一个阶段，时钟周期对应一个基本操作过程。

因此，时间应按指令周期、CPU周期、时钟周期进行分级控制，这就是三级时序体制。

指令周期时间到，说明一条指令执行完毕；CPU周期时间到，说明一个阶段执行完毕；时钟周期时间到，说明一个基本操作过程执行完毕。

5.3.2 控制方式

1、同步控制方式

在任何情况下，已定的指令在执行时所需要的机器周期数、时钟周期数固定不变，即任何指令的指令周期是确定的、可预知的。

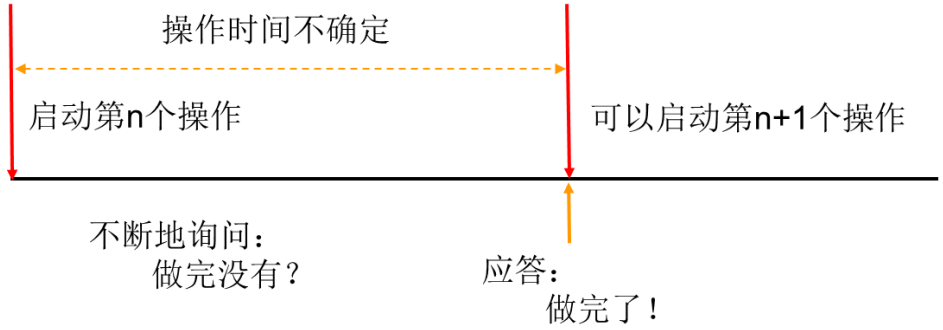
对已定指令，它的每个阶段、每个操作，都是在时间信号（闹钟）的同步下，在规定的时间点开始，在规定的时间点结束。

5.3.2 控制方式

2、异步控制方式

每条指令、每个阶段、每个操作需要多少时间就占用多少时间，没有同步的时间信号（闹钟）。

通常通过“询问”、“回答”方式，确定一个操作是否完成，因此其执行时间是不可预知的。



5.4 微程序控制器

- 微程序控制原理
- 微程序设计技术

5.4.1 微程序控制原理

【例】ADD R2,R0； (R0)+(R2)→ (R0)

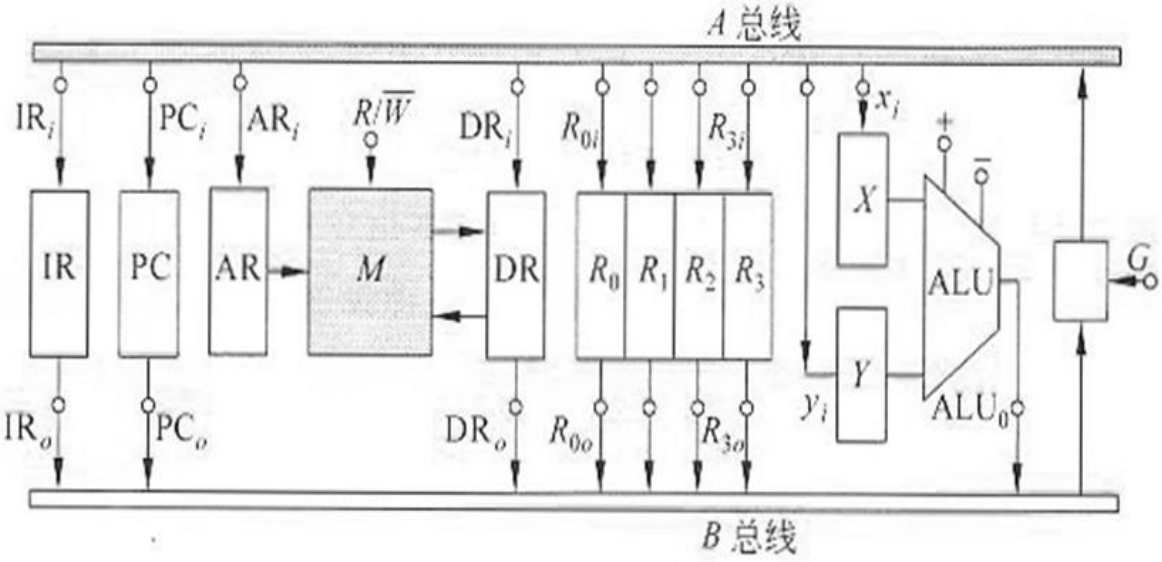
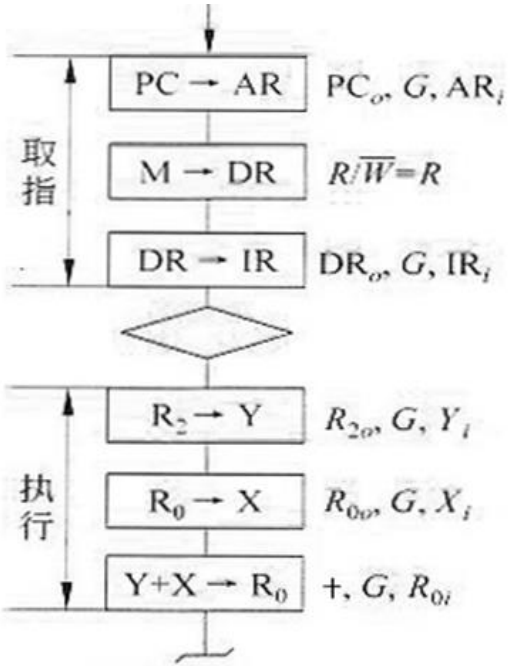
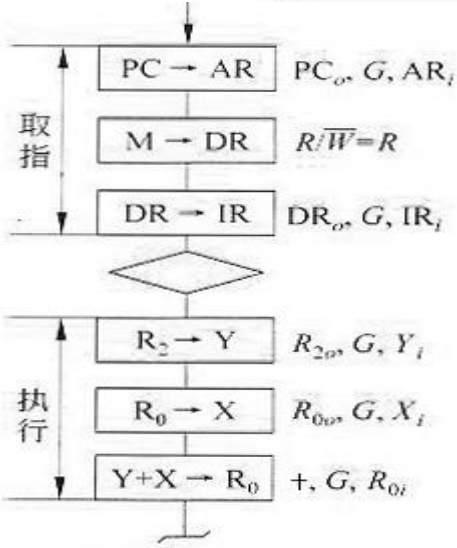


图 5.15 双总线结构机器的数据通路



5.4.1 微程序控制原理

【例】ADD R2,R0； (R0)+(R2)→ (R0)



位输出，直接接控制信号端。

↑ PC_o	↑ G	↑ AR_i	↑ R/W	↑ DR_o	↑ IR_i	↑ $R2_o$	↑ $R0_o$	↑ Y_i	↑ X_i	↑ +	↑ $R0_i$
1	1	1	0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	0	0	0	0
0	1	0	0	1	1	0	0	0	0	0	0
0	1	0	0	0	0	1	0	1	0	0	0
0	1	0	0	0	0	0	1	0	1	0	0
0	1	0	0	0	0	0	0	0	0	1	1

微程序

5.4.1 微程序控制原理

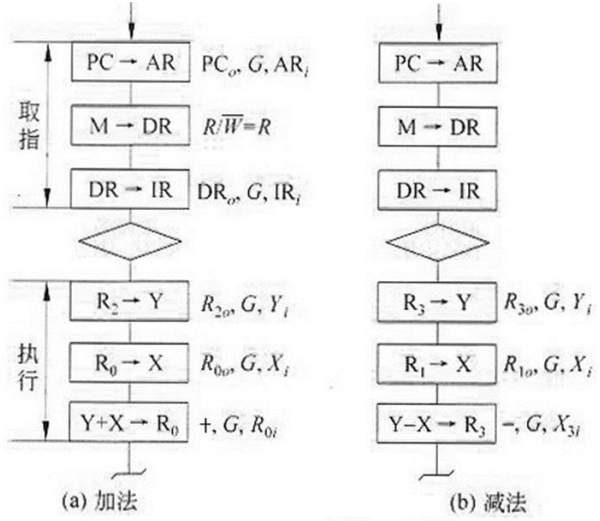
1、微命令、微操作

微命令：控制器通过控制线向部件发送的各种控制信号/操作命令。

微操作：部件接收微命令以后所完成的操作，微操作是执行部件中最基本的、不可再分解的操作。

微操作：每个方框中定义的操作。

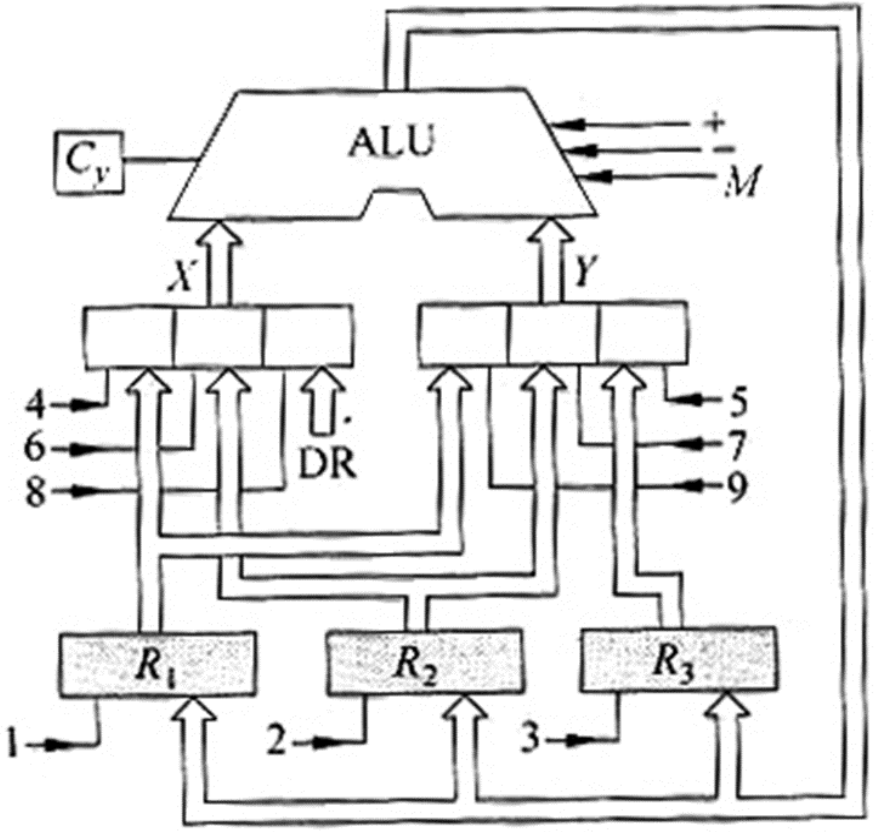
微命令：每个方框旁边的控制信号。



5.4.1 微程序控制原理

1、微命令、微操作

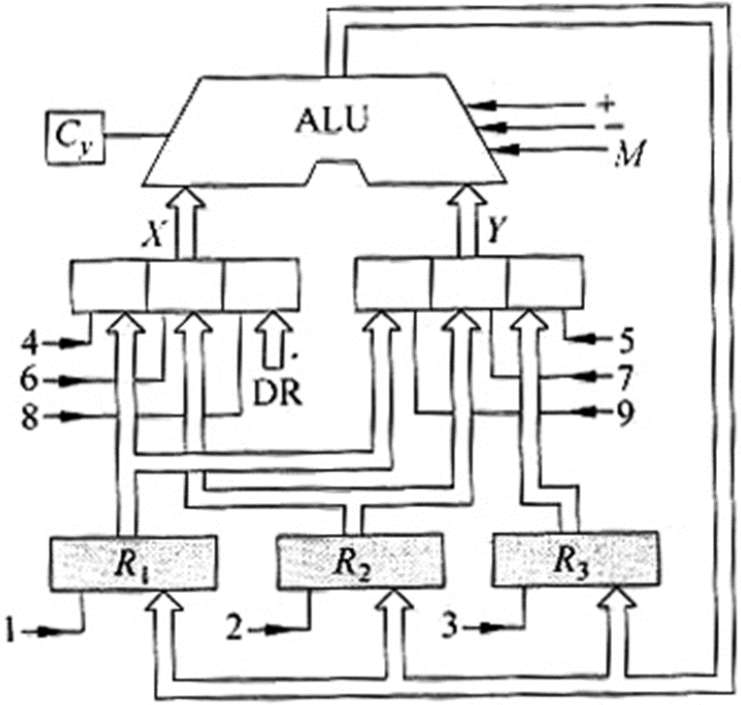
微命令	微操作
1	ALU→ R1
2	ALU→ R2
3	ALU→ R3
4	RI →X
6	R2→X
8	DR →X
5	R1 →Y
7	R2 →Y
9	R3 →Y
+	ADD
-	SUB
M	MOV



简单运算器数据通路图

5.4.1 微程序控制原理

2、相容性微命令/微操作、相斥性微命令/微操作



简单运算器数据通路图

相容性微命令/微操作：

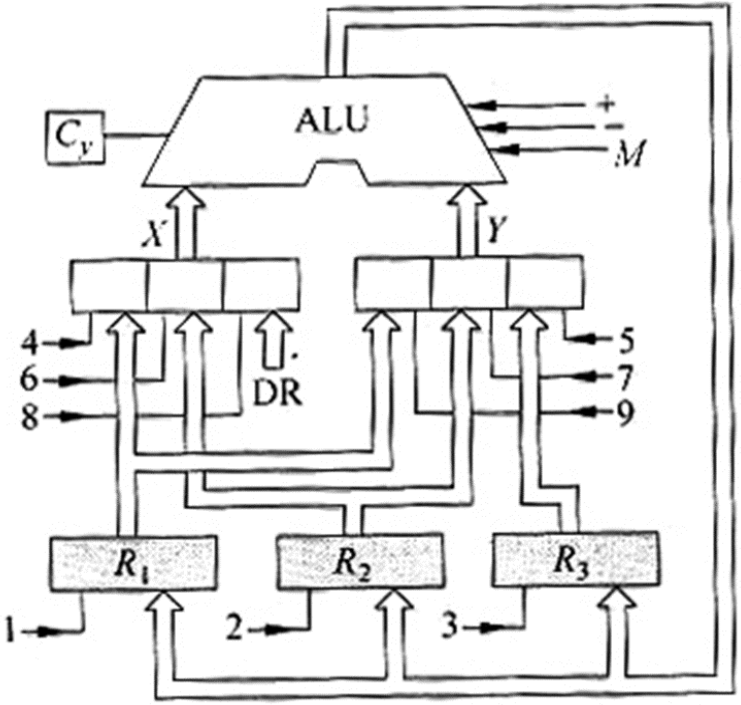
同时、或同一个时间范围内可以并行执行的微操作。

ALU→R1、ALU→R2 ALU→ R3

相应地微命令1、2、3可同时有效。

5.4.1 微程序控制原理

2、相容性微命令/微操作、相斥性微命令/微操作



简单运算器数据通路图

相斥性微命令/微操作：

不能同时、或在同一个时间范围内并行执行的微操作。

微操作：R1→X、R2→X、DR→X

微命令：4、6、8

微操作：ADD、SUB、MOV

微命令：+、-、M

5.4.1 微程序控制原理

3、微指令、微程序

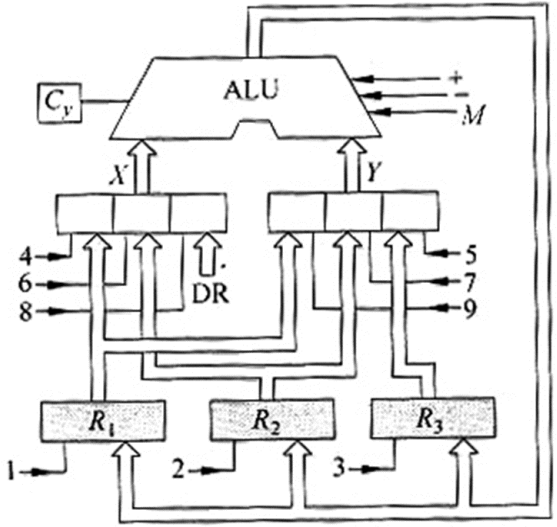
一组实现一定操作功能的微命令的组合形式，称为微指令。由操作控制和顺序控制两大部分组成。

微指令是CPU控制器设计人员使用的编程指令，而用户所能接触到的最低级的是机器指令。显然微指令比机器指令的层次还低，因此叫“微”。

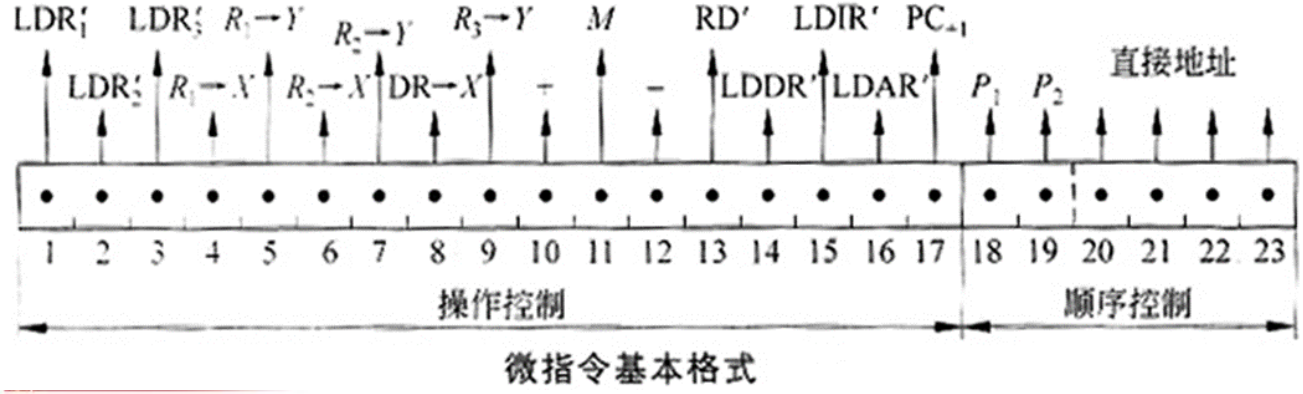
5.4.1 微程序控制原理

3、微指令、微程序

操作控制部分：用来发出管理和指挥部件工作的控制信号（微命令），这些微命令是带时间信号的（什么时候有效、出现的先后顺序、维持多长时间）。



简单运算器数据通路图

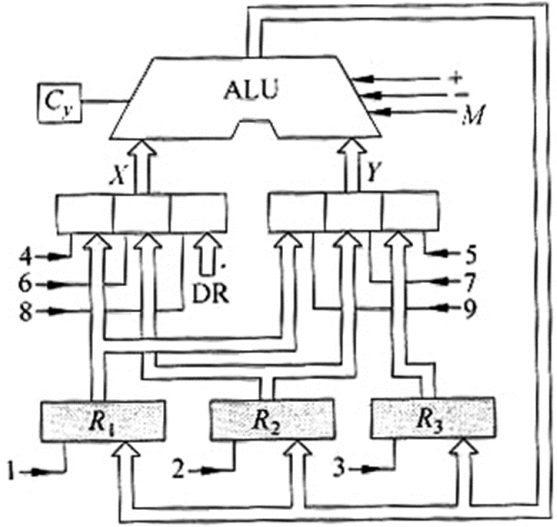


微指令基本格式

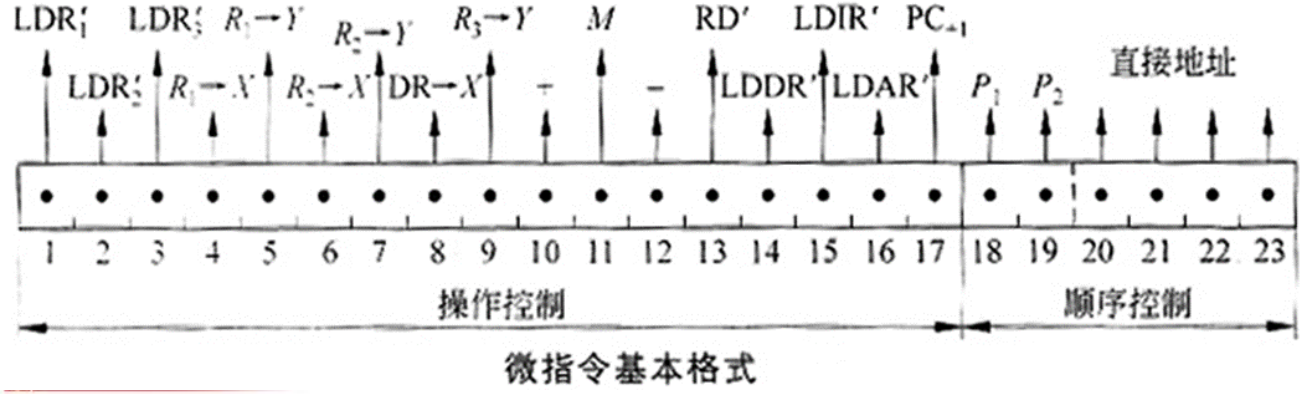
5.4.1 微程序控制原理

3、微指令、微程序

顺序控制部分：用来产生下一条待取微指令的地址，包括直接给出下条微指令的地址（P1P2=00表示无转移时），根据P1、P2的判别测试条件可能发生的转移。



简单运算器数据通路图



微指令基本格式

5.4.1 微程序控制原理

3、微指令、微程序

一条机器指令的功能是由多条微指令组成的序列来实现的，这个微指令序列称为微程序。

微程序控制器通过逐条执行微程序（软件），产生操作所需要的控制信号。

↑	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑
PC_o	G	AR_i	R/W	DR_o	IR_i	$R2_o$	$R0_o$	Y_i	X_i	+	$R0_i$
1	1	1	0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	0	0	0	0
0	1	0	0	1	1	0	0	0	0	0	0
0	1	0	0	0	0	1	0	1	0	0	0
0	1	0	0	0	0	0	1	0	1	0	0
0	1	0	0	0	0	0	0	0	0	1	1

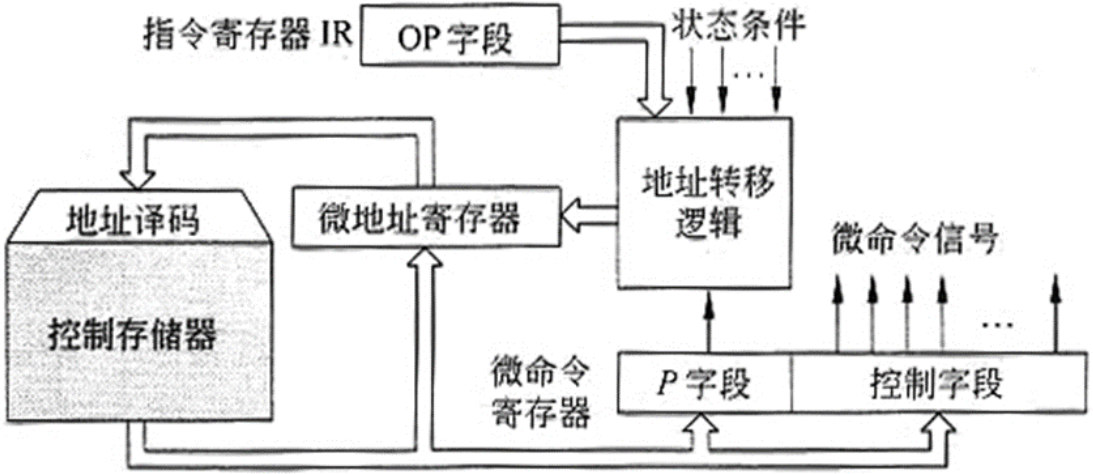
5.4.1 微程序控制原理

4、微程序控制器原理框图

控制存储器ROM：存放全部指令系统的微程序；

微地址寄存器uPC：具有自动增量功能，给出顺序执行的下条微指令地址；

微命令寄存器uIR：存放由控制存储器读出的一条微指令。

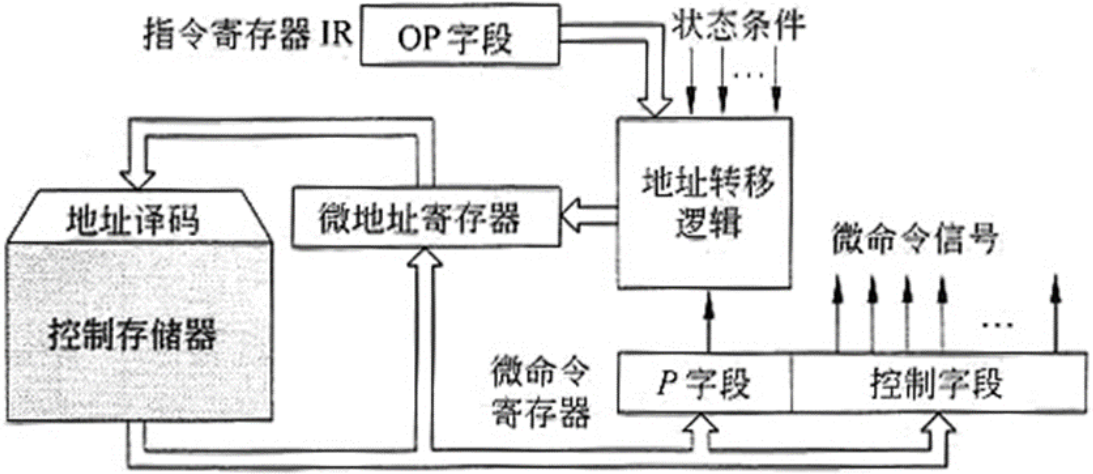


微程序控制器原理框图

5.4.1 微程序控制原理

4、微程序控制器原理框图

地址转移逻辑：①根据指令寄存器IR的操作码，定位到该指令对应的微程序段，uPC 初值；②如果判断条件P/状态条件=FALSE，则 uPC=uPC +1，顺序执行；③如果判断条件P/状态条件=TRUE，则uPC=根据策略形成新的微指令地址，程序转移。

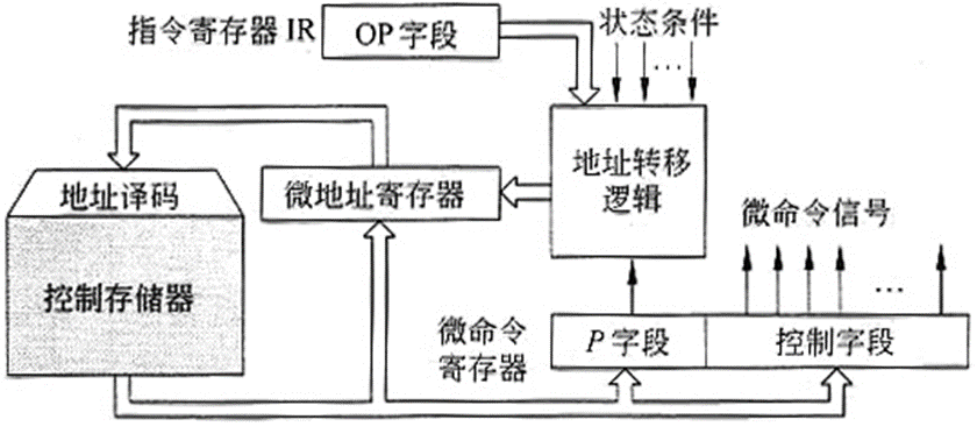


微程序控制器原理框图

5.4.1 微程序控制原理

5、微程序控制器原理

设计层面：首先，根据CPU的数据通路结构、指令操作定义等，画出每条指令的指令周期流程图（具体到每个时钟周期、微操作、微命令）。然后，根据微指令格式、指令周期流程图编写每条指令的微程序。最后，把整个指令系统的微程序（其中取指令的微程序段是公用的）固化到控制存储器中。

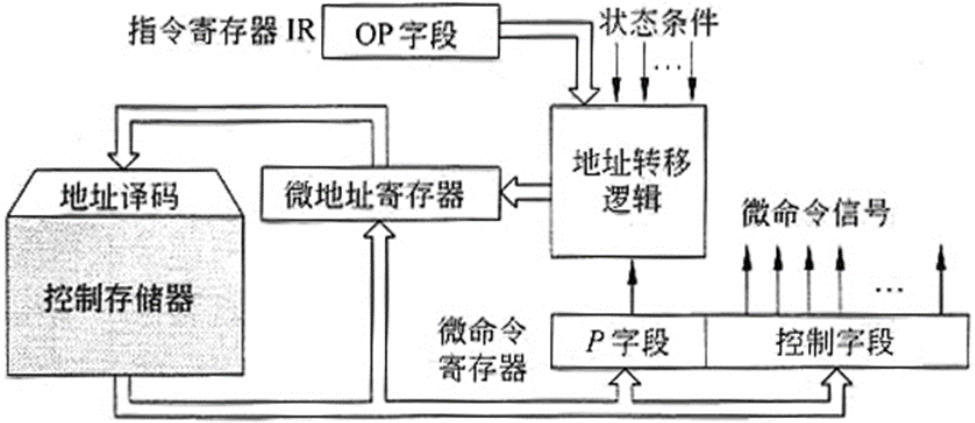


微程序控制器原理框图

5.4.1 微程序控制原理

5、微程序控制器原理

运行层面：首先，逐条执行取指令公用微程序段，控制取指令操作。然后，根据指令的操作码字段，经过变换，找到该指令所对应的特定微程序段，从控制存储器中逐条取出微指令，根据微操作控制字段，直接或经过译码产生微命令（控制信号），控制相关部件完成指定的微操作。一条微指令执行以后，根据微地址字段取下一条微指令。

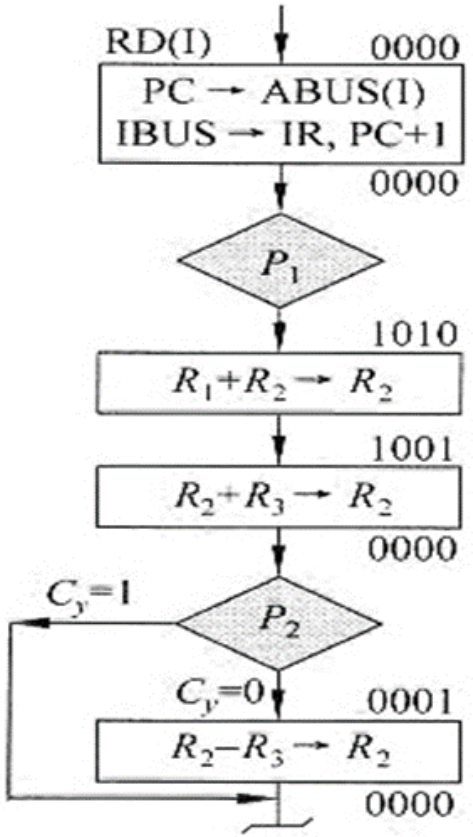


微程序控制器原理框图

5.4.1 微程序控制原理

【例】R1+R2→R2，R1、R2中均是BCD码，要求BCD码加法运算以后结果是合法的BCD码，设（R3）=6。

算法：首先做R1+R2+R3，如果有进位则表示R1+R2 >1001，应该加6，否则应从结果中减6。



十进制加法微程序

5.4.1 微程序控制原理

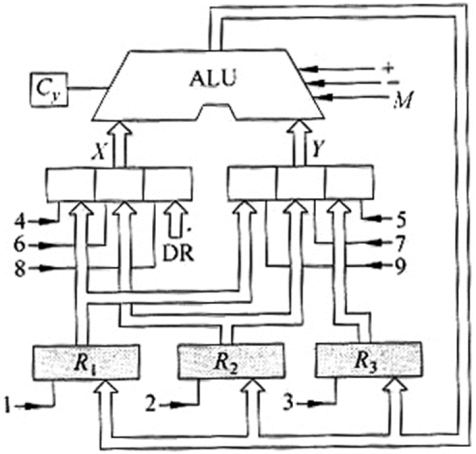
【解】

PC →AR:　　LDAR

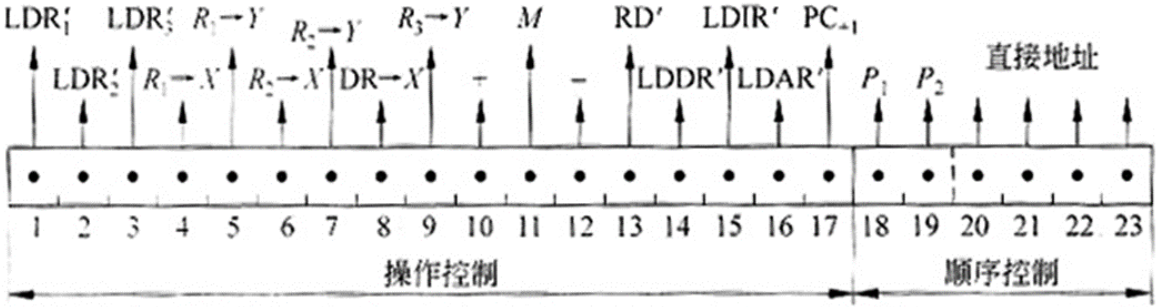
M →DR:　　RD,LDDR

DR →IR:　　LDIR

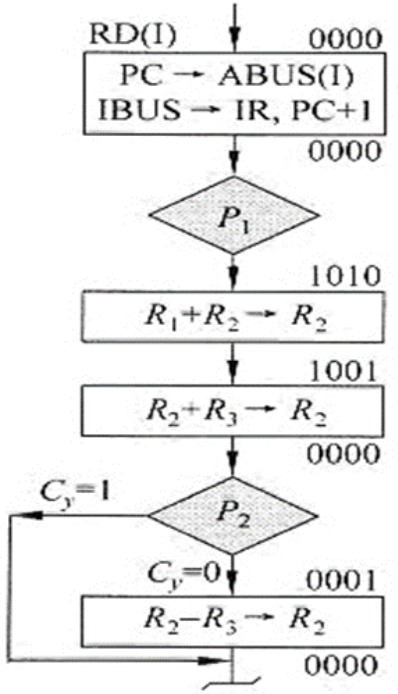
PC+1 →PC:　　PC_{+1}



简单运算器数据通路图



微指令基本格式



十进制加法微程序

本条地址	操作字段	顺序字段	下条地址
0000	000 00000000011111	10 0000	P1条件：判断是否本条指令。P1=0，顺序 0000；P1=1，转移1010

5.4.1 微程序控制原理

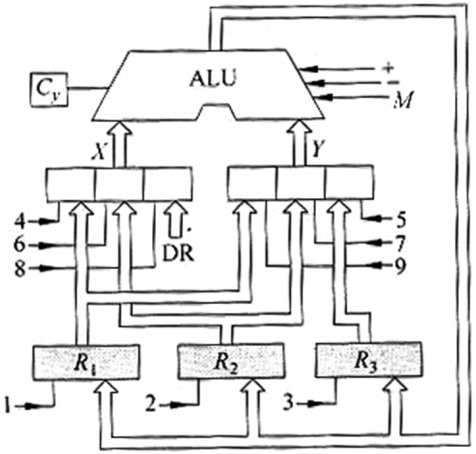
【解】

R1 →X:　　R1 →X

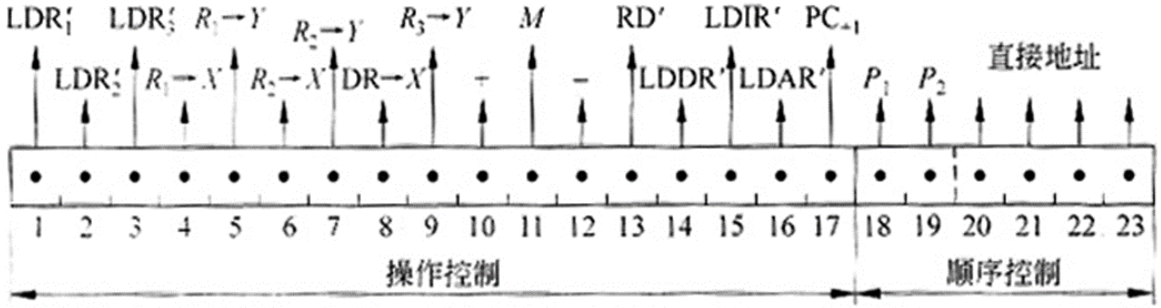
R2 →Y:　　R2 →Y

+ →ALU:　　+

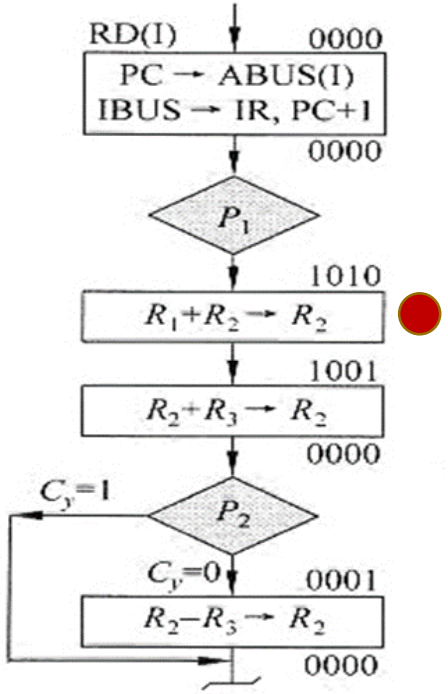
ALU →R2:　　LDR2



简单运算器数据通路图



微指令基本格式



十进制加法微程序

本条地址	操作字段	顺序字段	下条地址
1010	1010010010000000	00 1001	顺序 1001

5.4.1 微程序控制原理

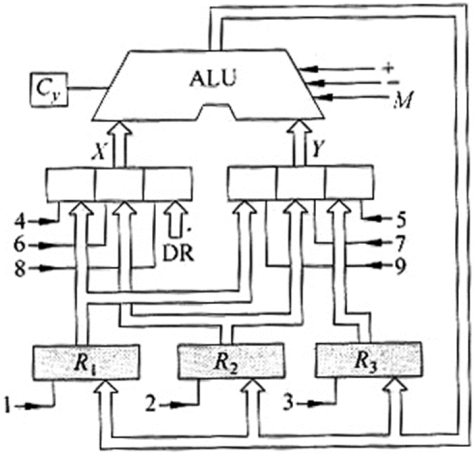
【解】

R2→X:　　R2 →X

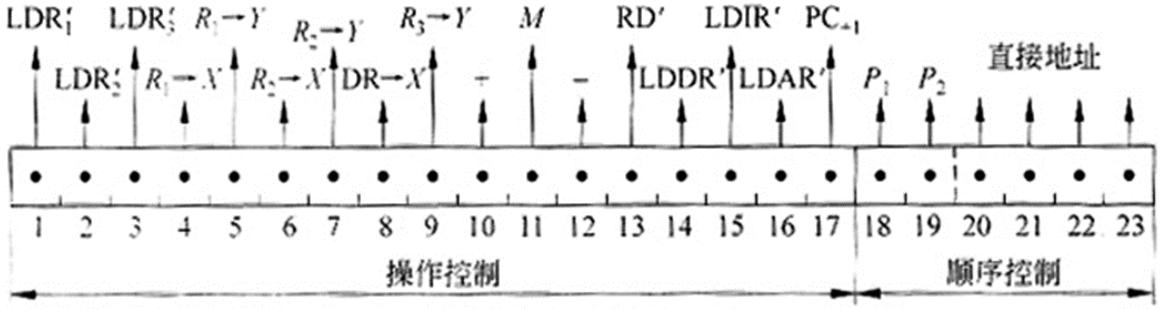
R3 →Y:　　R3 →Y

+ →ALU:　　+

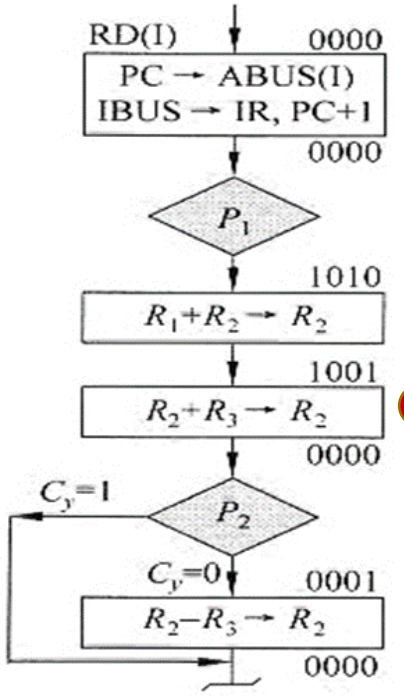
ALU →R2:　　LDR2



简单运算器数据通路图



微指令基本格式



十进制加法微程序

本条地址	操作字段	顺序字段	下条地址
1001	1000100110000000	01 0001	P2条件：判断是否有进位。P2=0，顺序 0001；P2=1，转移0000

5.4.1 微程序控制原理

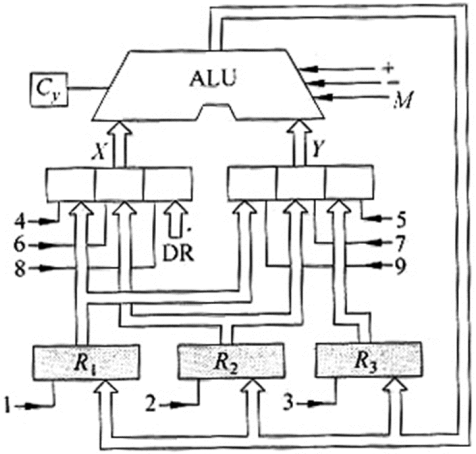
【解】

R2→X:　　R2 →X

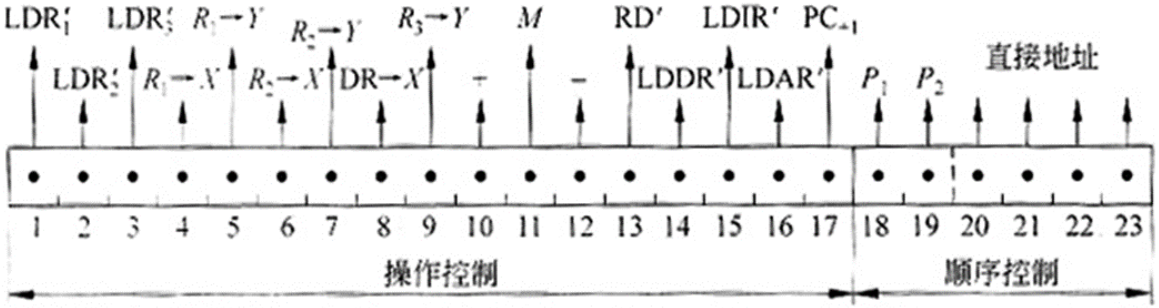
R3 →Y:　　R3 →Y

- →ALU:　　-

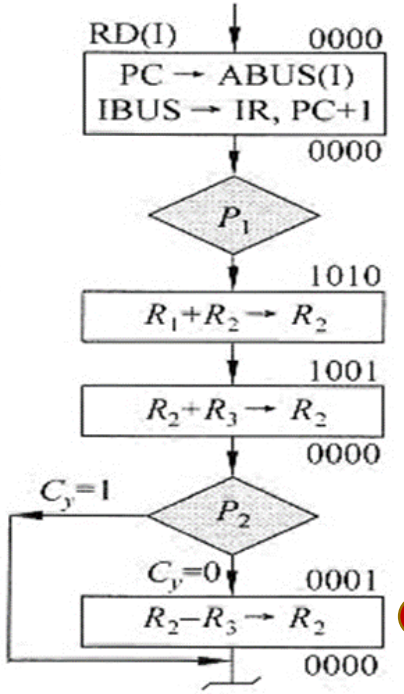
ALU →R2:　　LDR2



简单运算器数据通路图



微指令基本格式

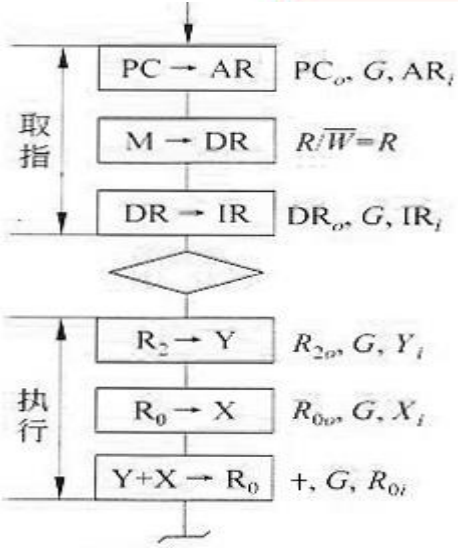


十进制加法微程序

本条地址	操作字段	顺序字段	下条地址
0001	1000100100100000	00 0000	顺序0000

5.4.1 微程序控制原理

【例】ADD R2,R0； (R0)+(R2)→ (R0)

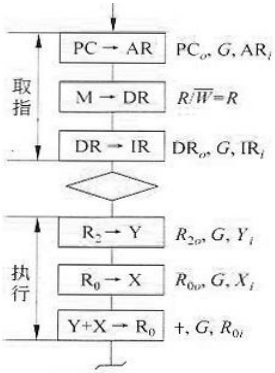


位输出，直接接控制信号端。

↑ PC_o	↑ G	↑ AR_i	↑ R/W	↑ DR_o	↑ IR_i	↑ $R2_o$	↑ $R0_o$	↑ Y_i	↑ X_i	↑ +	↑ $R0_i$
1	1	1	0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	0	0	0	0
0	1	0	0	1	1	0	0	0	0	0	0
0	1	0	0	0	0	1	0	1	0	0	0
0	1	0	0	0	0	0	1	0	1	0	0
0	1	0	0	0	0	0	0	0	0	1	1

微程序

5.4.1 微程序控制原理

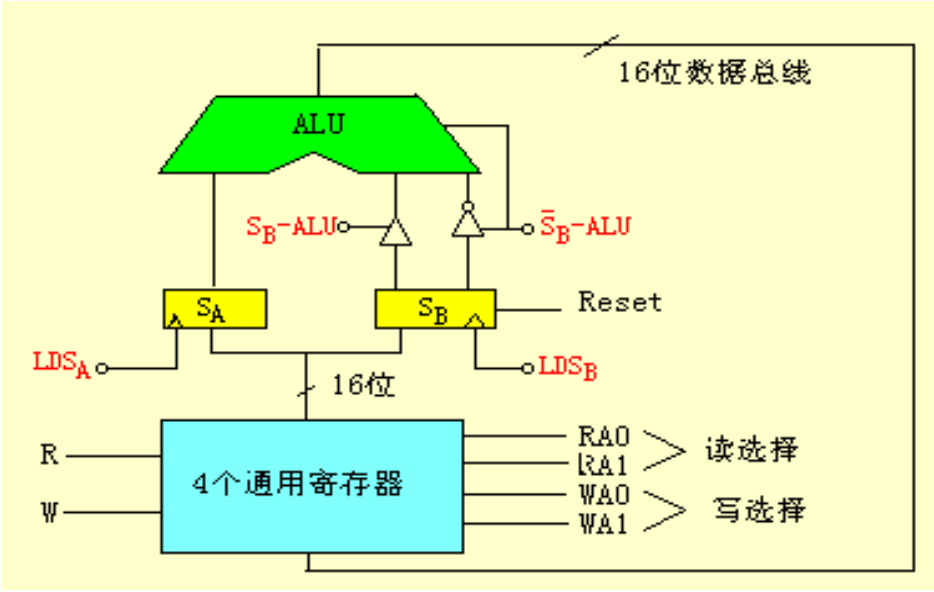


↑ PC_o	↑ G	↑ AR_i	↑ R/W	↑ DR_o	↑ IR_i	↑ $R2_o$	↑ $R0_o$	↑ Y_i	↑ X_i	↑ +	↑ $R0_i$
1	1	1	0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	0	0	0	0
0	1	0	0	1	1	0	0	0	0	0	0
0	1	0	0	0	0	1	0	1	0	0	0
0	1	0	0	0	0	0	1	0	1	0	0
0	1	0	0	0	0	0	0	0	0	1	1

本条地址	操作字段	顺序字段	下条地址
0100	111000000000	0 0101	顺序 0101
0101	000100000000	0 0110	顺序 0110
0110	010011000000	1 0100	P1条件：判断是否本条指令。P1=0，顺序 0100；P1=1，转移0110
1100	010000101000	0 1101	顺序 1101
1101	010000010100	0 1110	顺序 1110
1110	010000000011	0 0100	顺序0100

5.4.1 微程序控制原理

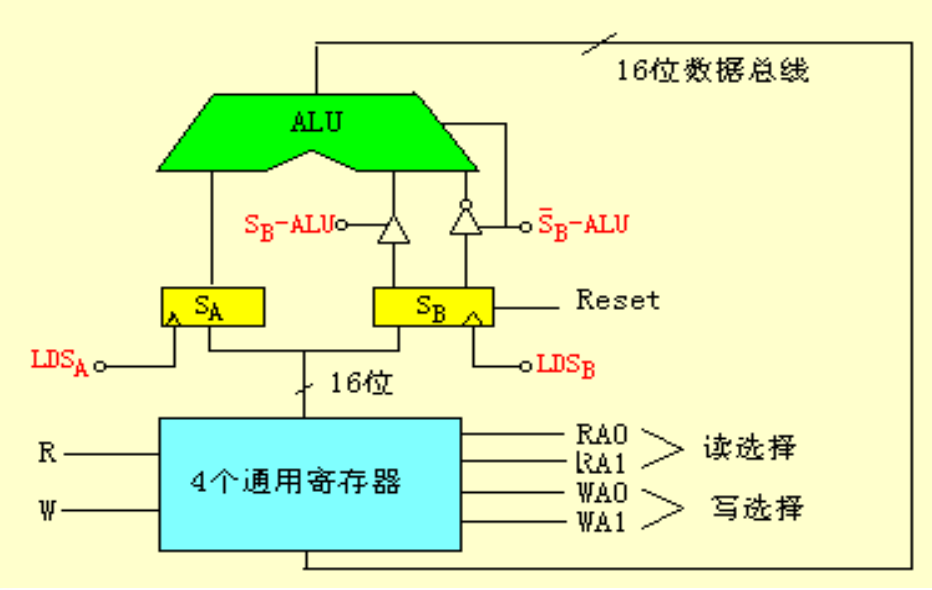
【例】设某计算机运算器框图如图所示，其中ALU为16位的加法器(高电平工作)，SA,SB为16位暂存器。4个通用寄存器的读写控制功能、微指令格式（未考虑顺序控制）如下表所示。



读控制				写控制			
R	RA0	RA1	选择	W	WA0	WA1	选择
1	0	0	R0	1	0	0	R0
1	0	1	R1	1	0	1	R1
1	1	0	R2	1	1	0	R2
1	1	1	R3	1	1	1	R3
0	*	*	不读出	0	*	*	不写入

5.4.1 微程序控制原理

【例】设某计算机运算器框图如图所示，其中ALU为16位的加法器(高电平工作)，SA,SB为16位暂存器。4个通用寄存器的读写控制功能、微指令格式（未考虑顺序控制）如下表所示。



RA_0RA_1	WA_0WA_1	R	W	LDS_A	LDS_B	S_B-ALU	$\overline{S}_B$-ALU	Reset	~

RA_0RA_1：读R0-R3的选择控制

WA_0WA_1：写R0-R3的选择控制

R：寄存器读命令

W：寄存器写命令

LDS_A：打入SA的控制信号

LDS_B：打入SB的控制信号

S_B-ALU：传送S_B的控制信号

$\overline{S}_B$-ALU：传送$\overline{S}_B$的控制信号，并使加法器最低位加1.

Reset：清暂存器SB为零的信号

~：一段微程序结束，转入取机器指令的控制信号

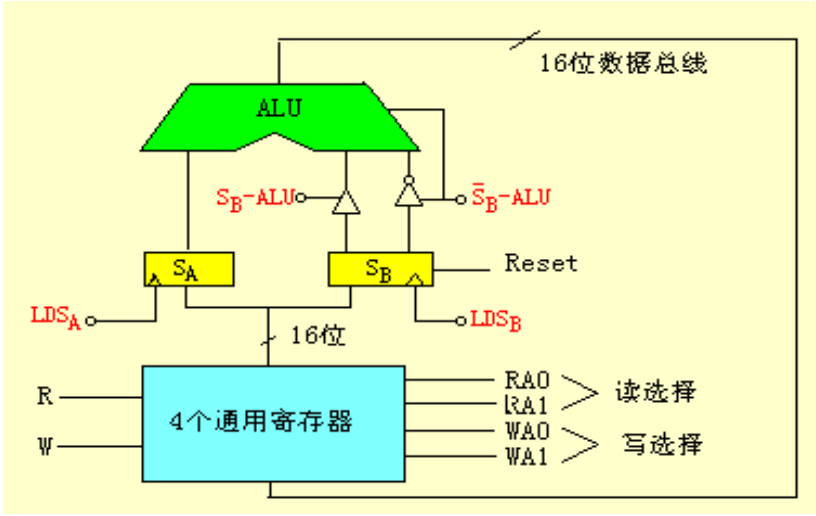
5.4.1 微程序控制原理

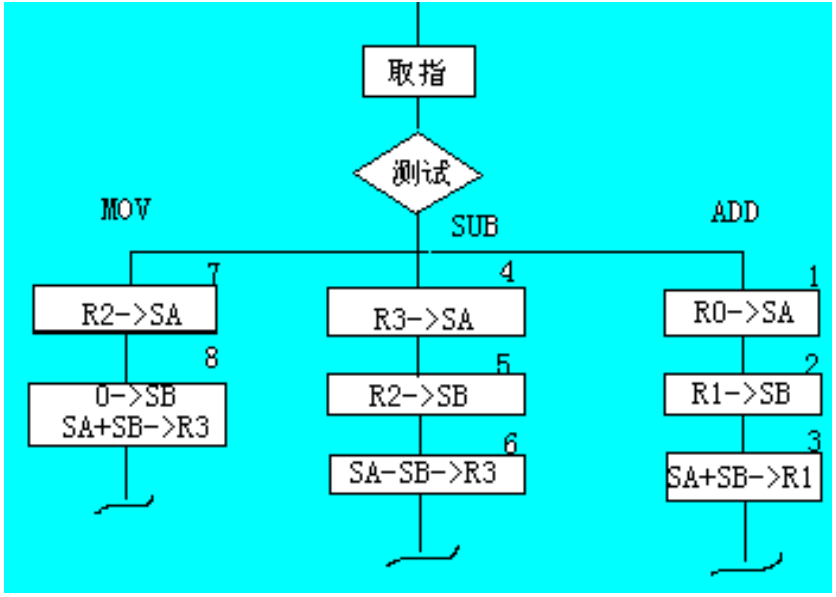
【例】要求：用二进制代码写出如下指令的微程序：

(1) “ADD R0，R1”指令，即(R0)+(R1)→R1

(2) “SUB R2，R3”指令，即(R3)-(R2)→R3

(3) “MOV R2，R3”指令，即(R2)→(R3)





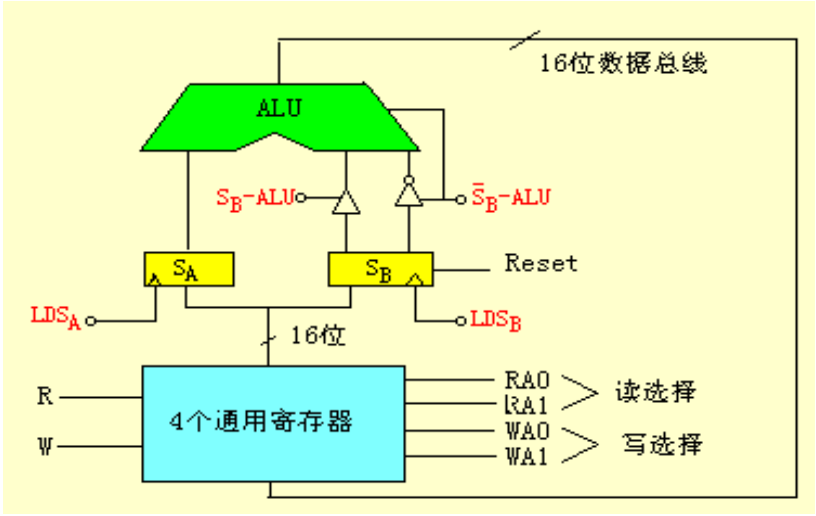
5.4.1 微程序控制原理

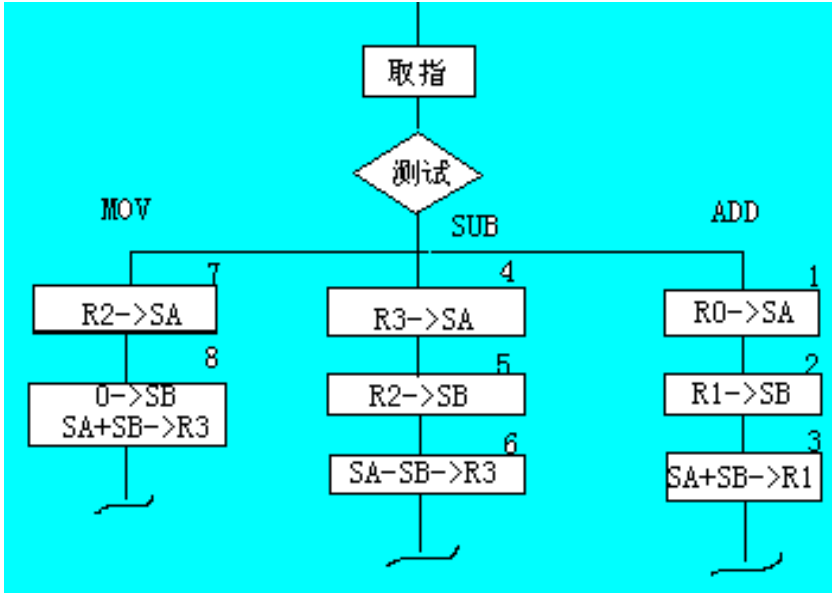
【例】要求：用二进制代码写出如下指令的微程序：

(1) “ADD R0，R1”指令，即(R0)+(R1)→R1

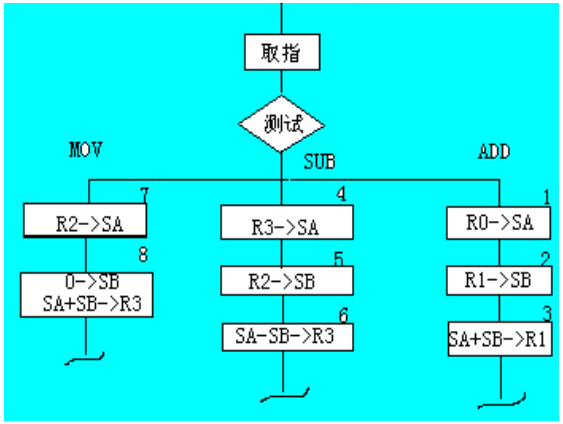
(2) “SUB R2，R3”指令，即(R3)-(R2)→R3

(3) “MOV R2，R3”指令，即(R2)→(R3)





5.4.1 微程序控制原理



微程序

指令	微指令										
ADD	00 **	1010	0000								
	01**	1001	0000								
	**01	0100	1001								
SUB	11**	1010	0000								
	10**	1001	0000								
	**11	0100	0101								
MOV	10**	1010	0000								
	**11	0100	1011								

RA_0RA_1	WA_0WA_1	R	W	LDS_A	LDS_B	S_B-ALU	$\overline{S}_B$-ALU	Reset	~

RA_0RA_1：读R0-R3的选择控制

WA_0WA_1：写R0-R3的选择控制

R：寄存器读命令

W：寄存器写命令

LDS_A：打入SA的控制信号

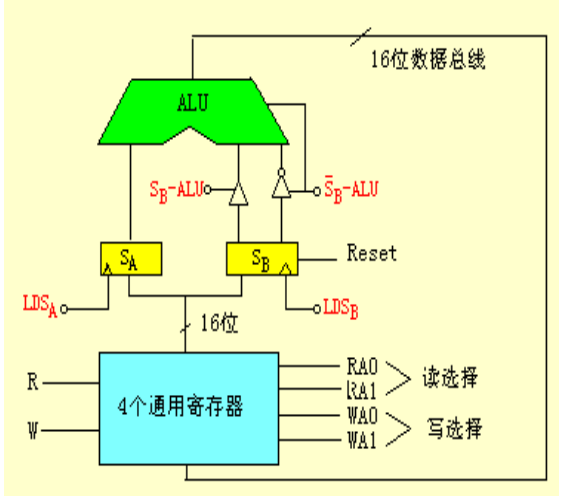
LDS_B：打入SB的控制信号

S_B-ALU：传送S_B的控制信号

$\overline{S}_B$-ALU：传送$\overline{S}_B$的控制信号,并使加法器最低位加1.

Reset：清暂存器SB为零的信号

~：一段微程序结束，转入取机器指令的控制信号

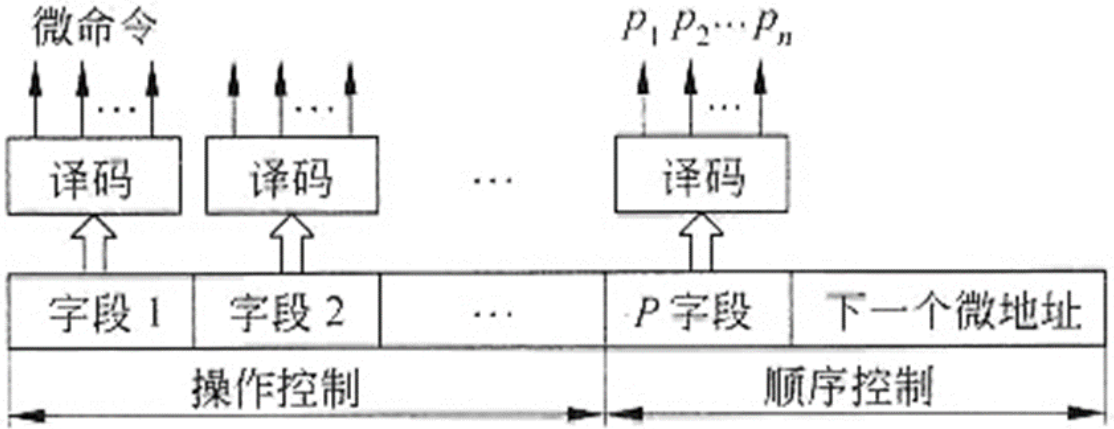


5.4.2 微程序设计技术

1、微命令编码

直接表示法：控制字段的每一位就直接代表一个微命令；

编码表示法：把一组相斥性的微命令组成一个字段，然后通过译码器进行译码。



字段直接译码法

5.4.2 微程序设计技术

2、微地址的形成方法

计数器方式：微地址寄存器uPC实现顺序控制，转移时通过策略改变uPC；

直接地址方式：微指令中直接给出顺序取的地址，若转移取则通过策略形成新的转移地址；

多路转移方式：一条微指令具有多个转移分支的能力，类似高级语言中的CASE语句。即根据判别测试标志、状态条件码等，依据一定的策略机制，形成新的多路转移微地址。

5.4.2 微程序设计技术

3、微指令格式

水平型微指令：一次能定义并执行多个并行操作微命令的微指令，其并行操作能力强、效率高、灵活性强。显然微指令格式较长、而一条指令对应的微程序较短；

垂直型微指令：每条微指令功能简单，一般控制一个基本的微操作，其并行操作能力弱、效率低，显然微指令格式较短、而一条指令对应的微程序较长。

5.4.2 微程序设计技术

4、静态、动态微程序设计

静态微程序设计：对应于一台计算机的机器指令只有一组微程序，而且这一组微程序设计好以后，一般无须改变而且也不好改变；

动态微程序设计：可以通过改变微指令和微程序来改变机器的指令系统，因而可以在一台机器上实现不同类型的指令系统。常用于计算机仿真。

5.5 硬连线控制器

- 基本思想
- 设计方法

5.5.1 基本思想

硬连线控制器由硬件方法（逻辑电路）实现。

一旦硬连线控制器构成后，除非重新设计和物理上重新布线，否则想增加功能是不可能的，因此逐渐被微程序控制器取代。

但是，由于硬件方法实现速度快、超大规模集成电路技术发展等因素，硬连线控制器思想又逐步得到重视，通常是两种方法结合使用。

5.5.1 基本思想

逻辑网络的输入信号来源：

(1)来自指令译码器的输出Im

(2)来自执行部件的反馈信息Bj

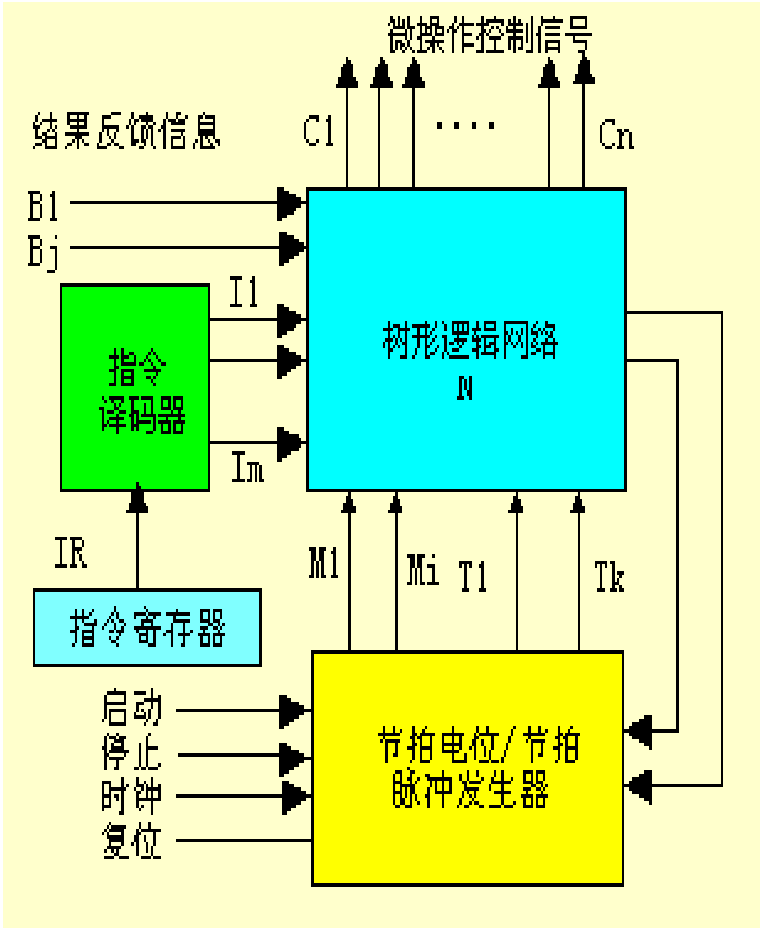
(3)来自时序产生器的时序信号Mi、Tk

逻辑网络N的输出信号Cn：

$Cn = f(Im, Bj, Mi, Tk)$

$= \sum (Mi.Tk.Bj \sum Im)$

即：哪些指令，在哪些CPU周期、时钟周期中，控制信号Cn应该有效。



5.5.2 设计方法

（1）确定指令系统、CPU内部部件构成、数据通路结构、时序信号和控制信号构成；

（2）根据数据通路结构，画出指令系统中所有指令的执行流程，必须定位到每个CPU周期、每个时钟周期中要做的微操作，需要的相关控制信号；

（3）根据指令执行流程，写出每个微命令的逻辑表达式；

（4）根据选用的元器件，对逻辑表达式化简；

（5）利用逻辑电路实现。

5.5.2 设计方法

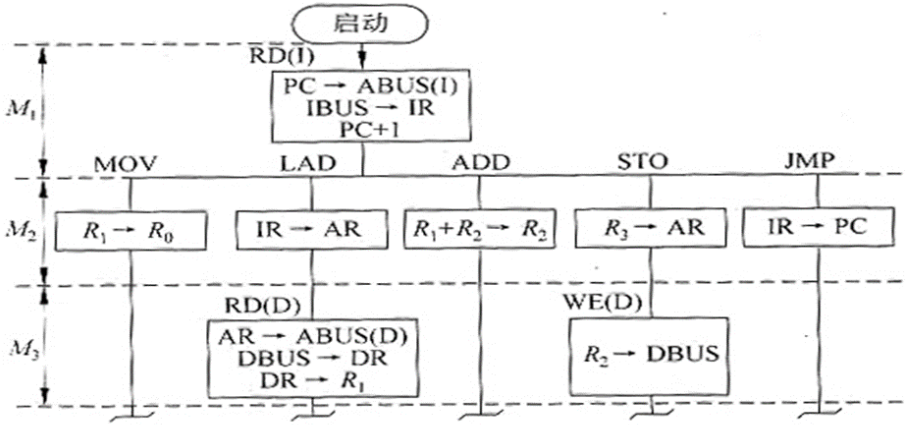
【例】指存读命令 RD(I)= M1.T2

数存读命令 RD(D)= M3.T2.LAD

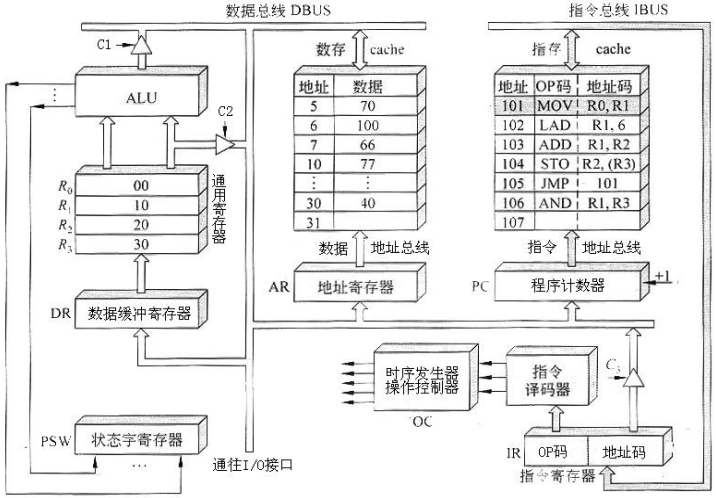
数存写命令 WE(D)= M3.T3.LDA

打入程序计数器命令 LDPC = M1.T4 + M2.T4.JMP

打入指令寄存器命令 LDIR = M1.T4



硬连线控制器的指令周期流程图



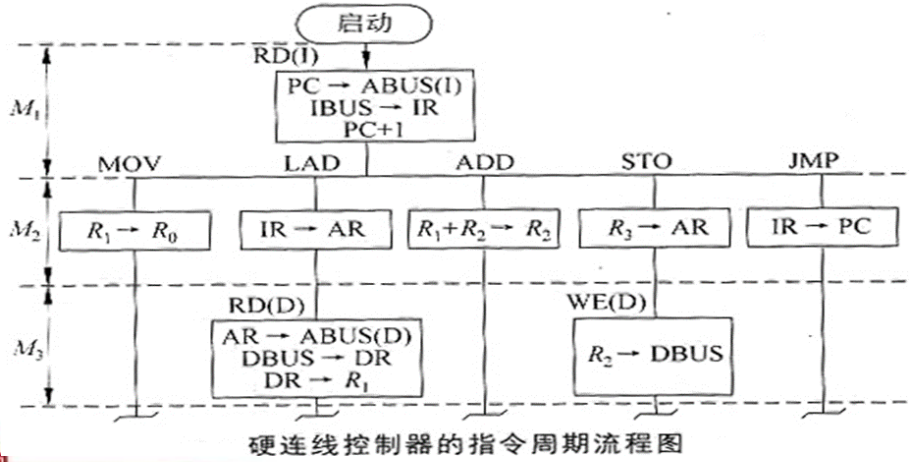
5.5.2 设计方法

【例】打入数存地址寄存器命令 LDAR = M2.T4.(LAD+STO)

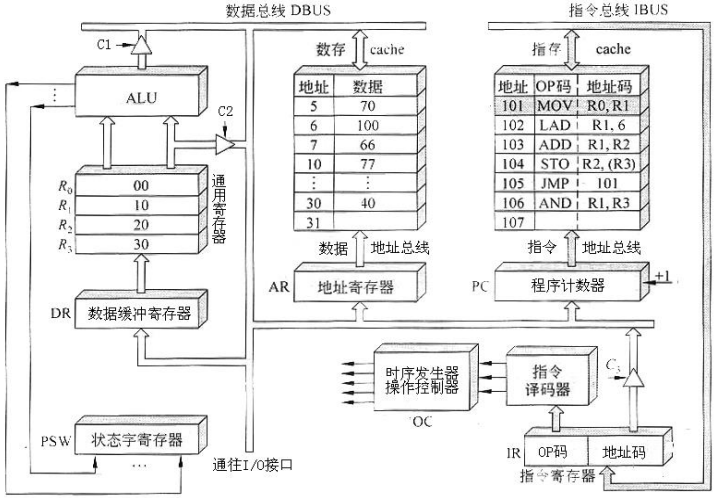
打入数据缓冲寄存器命令 LDDR =M2.T3.(MOV+ADD)+M3.T3.LDA

程序计数器加1命令 PC+1= M1.T4

打入R1寄存器命令 LDR2 = M2.T4.ADD

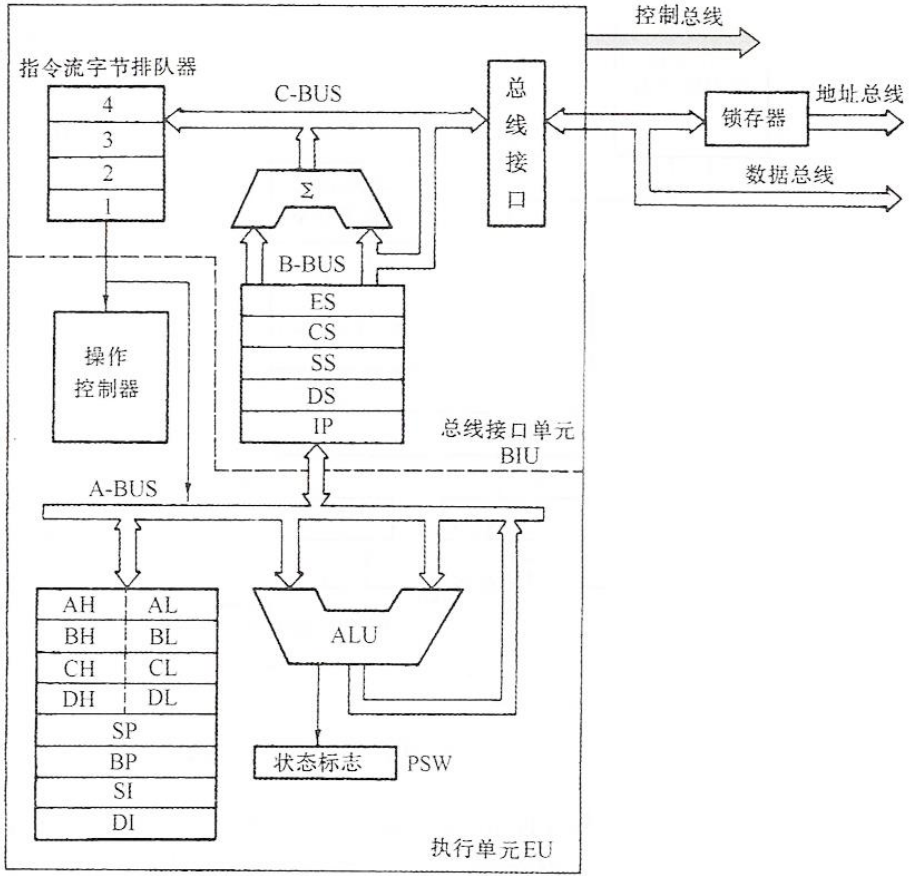


硬连线控制器的指令周期流程图



5.6 传统CPU

8086/8088CPU



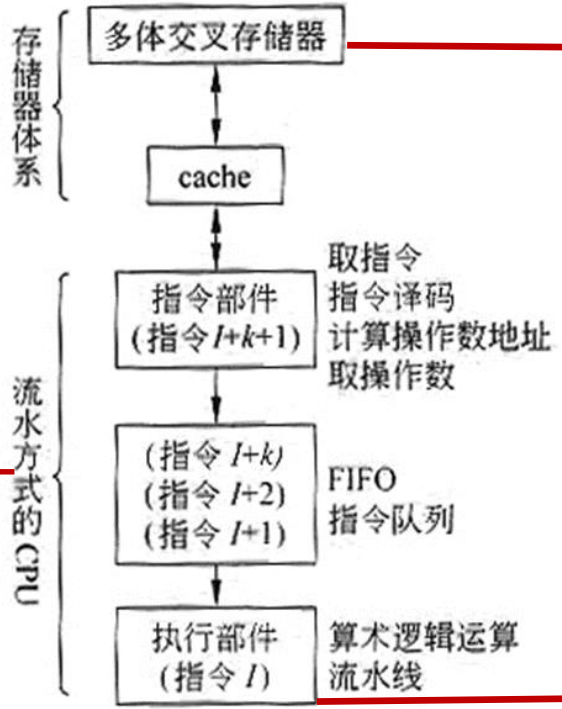
5.7 流水CPU

- 流水CPU的结构
- 流水线中的主要问题
- 奔腾CPU

5.7.1 流水CPU的结构

1、流水计算机系统的组成

流水线存取的多体交叉存储器



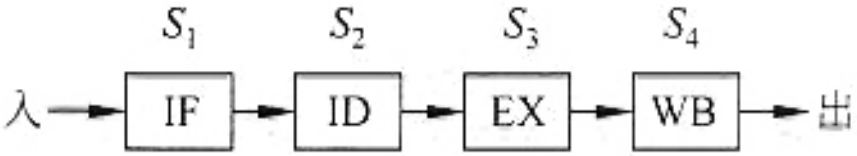
流水计算机系统组成原理示意图

多个过程段构成的流水线指令执行部件

流水线执行部件，如浮点运算器；多种、多个运算部件构成流水线。

5.7.1 流水CPU的结构

2、流水CPU的时空图

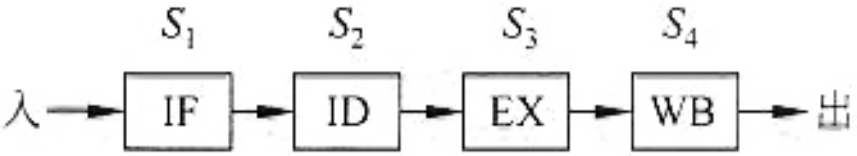


(a) 一个指令流水线过程段

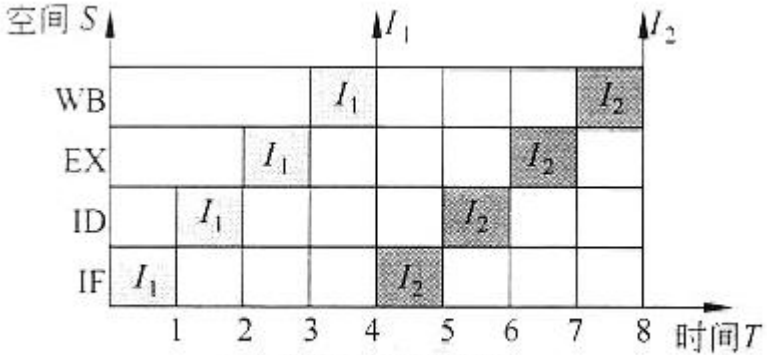
设一个指令周期分解为取指令IF、指令译码ID、执行运算EX、结果写回WB四个过程段，组成四个过程段的流水线。

5.7.1 流水CPU的结构

2、流水CPU的时空图



(a) 一个指令流水线过程段

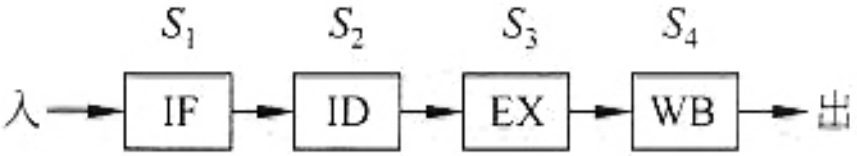


(b) 非流水线时空图

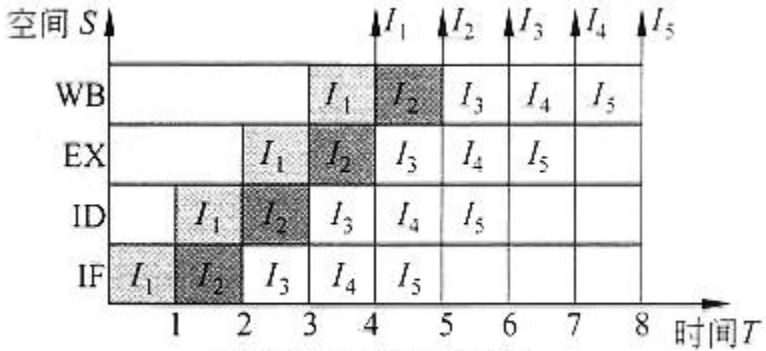
非流水线时空图

5.7.1 流水CPU的结构

2、流水CPU的时空图



(a) 一个指令流水线过程段

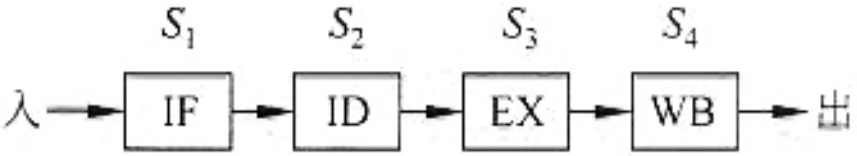


(c) 标量流水线时空图

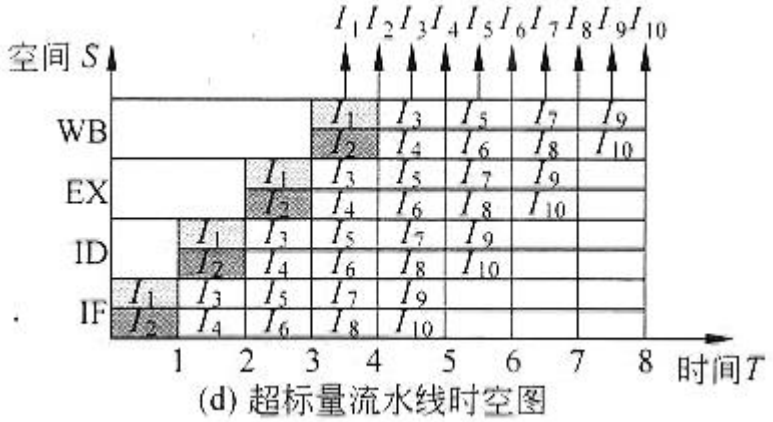
一条流水线（标量流水线）时空图

5.7.1 流水CPU的结构

2、流水CPU的时空图



(a) 一个指令流水线过程段



(d) 超标量流水线时空图

二条及以上流水线（超标量流水线）时空图

5.7.1 流水CPU的结构

4、流水线分类

指令流水线：指指令执行步骤的并行。将指令流的处理过程划分为取指令、指令译码、执行、写结果等几个并行处理的过程段。

算术流水线：是指运算操作步骤的并行，例如流水乘法器、流水除法器、流水浮点运算器等。

处理机流水线：又称为宏流水线，是指程序步骤的并行。由一串级联的处理机构成流水线的各个过程段，每台处理机负责某一特定的任务。

5.7.2 流水线中的主要问题

要使流水线具有良好的性能，必须使流水线畅通流动、不发生断流。

但由于流水过程中会出现资源相关、数据相关、控制相关三种相关冲突，可能使流水线断流，达不到理想的时间 k+(n-1) 个流水线时钟周期T。

5.7.2 流水线中的主要问题

1、资源相关

是指多条指令进入流水线后，在同一流水线时钟周期内争用同一个功能部件所发生的冲突。

两条指令同时访存发生资源相关冲突

指令 \ 时钟	1	2	3	4	5	6	7	8
I_1(LAD)	IF	ID	EX	MEM	WB			
I_2		IF	ID	EX	MEM	WB		
I_3			IF	ID	EX	MEM	WB	
I_4				IF	ID	EX	MEM	WB
I_5					IF	ID	EX	MEM

解决资源相关冲突的办法：

一是第I4条指令停顿一个时钟周期后再启动。

二是增设一个存储器，将指令和数据分别放在两个存储器中。

5.7.2 流水线中的主要问题

2、数据相关

在一个程序中，如果必须等前一条指令执行完毕以后，才能执行后一条指令，那么这两条指令就是数据相关的。

两条指令会发生数据相关冲突

指令 \ 时钟	1	2	3	4	5	6	7	8
ADD	IF	ID	EX	MEM	WB			
SUB		IF	ID	EX	MEM	WB		
AND			IF	ID	EX	MEM	WB	

```
ADD   R1, R2, R3    ;  (R2) + (R3) → (R1)
SUB   R4, R1, R5    ;  (R1) - (R5) → (R4)
AND   R6, R1, R7    ;  (R1) . (R7) → (R6)
```

5.7.2 流水线中的主要问题

2、数据相关

在一个程序中，如果必须等前一条指令执行完毕以后，才能执行后一条指令，那么这两条指令就是数据相关的。

两条指令会发生数据相关冲突

指令 \ 时钟	1	2	3	4	5	6	7	8
ADD	IF	ID	EX	MEM	WB			
SUB		IF	ID	EX	MEM	WB		
AND			IF	ID	EX	MEM	WB	

解决数据相关冲突的办法：

在流水CPU的运算器中设置若干个运算结果缓冲寄存器，暂时保留运算结果，以便于后继指令提前直接使用，这称为“向前”或定向传送技术。

5.7.2 流水线中的主要问题

3、控制相关

控制相关冲突是由转移类指令引起的。当执行转移类指令时，可能为顺序取下条指令；也可能转移到新的目标地址取指令。

如果流水线顺序取指令，而程序却需要转移时，进入流水线的指令并不是将要执行的指令，或者转移的目标指令可能还没有进入流水线，从而使流水线发生断流。

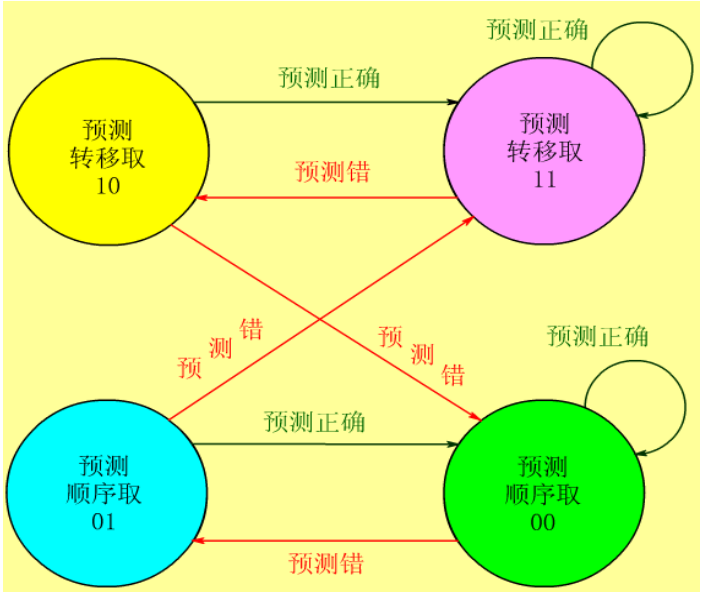
延迟转移法：基本思想是“先执行再转移”，即发生转移取时并不排空指令流水线，而是让紧跟在转移指令之后已进入流水线的指令继续完成。若与转移指令无关，可能有用。

5.7.2 流水线中的主要问题

3、控制相关

转移预测法：硬件方法来实现，依据指令过去的行为来预测将来的行为。即在取指令阶段，碰到一条转移类指令，则提前预测是顺序取、还是转移取。

从这个状态转换图可以看到：如果连续两次预测错误，则必定会改变预测方向。

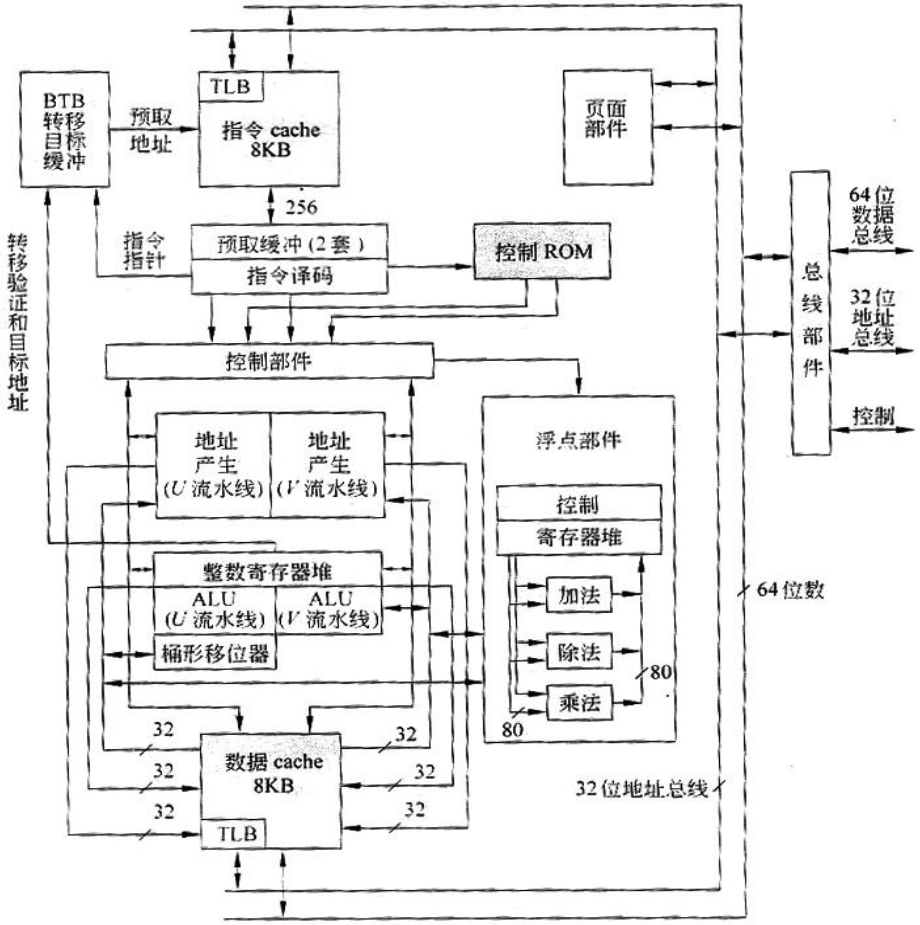


5.7.3 奔腾CPU

1、32位微处理器；

2、外部数据总线64位、地址总线32位；

3、片内指令和数据cache，采用组相联结构；



5.7.3 奔腾CPU

4、U、V两条指令流水线，涉及到指令调度发射策略；

5、八段的流水浮点运算器，支持IEEE754标准中32/64位浮点数、80位扩展浮点数；

6、采用动态转移预测技术实现指令预取，BTB记录过去的转移行为。