中 央 处 理 器

- CPU的功能和组成
- 指令周期
- 时序产生器和控制方式
- 微程序控制器
- 硬布线控制器
- CPU举例

CPU的功能

中央处理器，简称CPU (Central Processing Unit)，具有如下四方面的基本功能：

指令控制：程序是一个指令序列，这些指令的相互顺序不能任意颠倒，必须严格按程序规定的顺序进行。指令的顺序寻址和跳跃寻址就是实现指令控制的手段。

操作控制：完成一条指令的功能需要一系列操作控制信号。CPU控制器产生这些操作控制信号并送往机器的相应部件，从而控制这些部件按指令的要求进行动作。

时间控制：各种指令的操作控制信号以及一条指令的整个执行过程必须严格定时。

数据加工：数据加工就是对数据进行算术运算和逻辑运算处理。

CPU的基本组成

传统的CPU包括运算器和控制器两部分。大型机中，由于这两部分都比较复杂，在逻辑上和组装上常采取分离的模式。微型机中，随着VLSI技术的发展，早期放在CPU外部的一些逻辑功能部件，如浮点运算器、Cache也移入CPU内。

控制器：由程序计数器、指令寄存器、指令译码器、时序产生器和操作控制器组成。它的主要功能有：

(1) 从内存中取出一条指令，并指出下一条指令在内存中的位置；

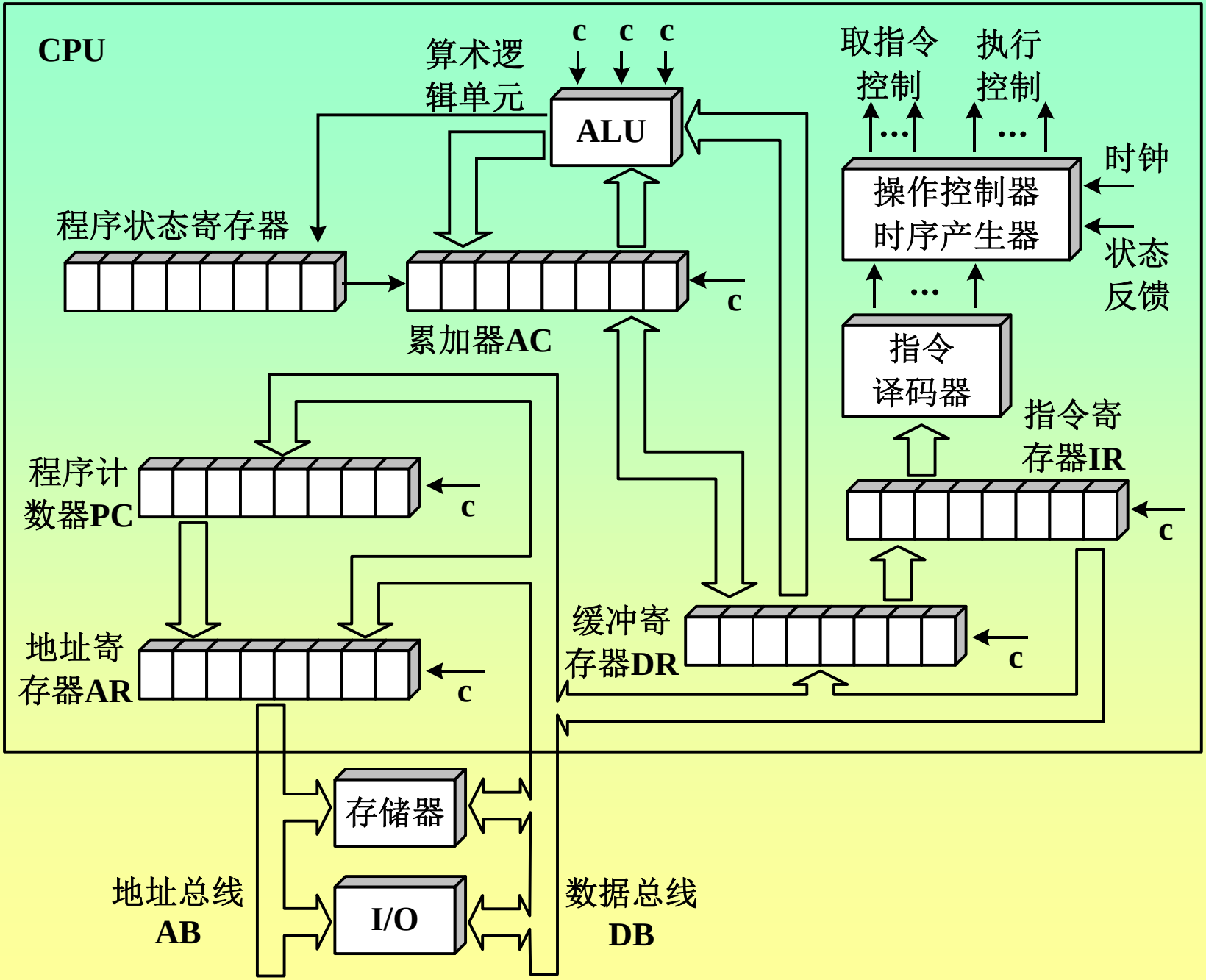
(2) 对指令进行译码或测试，并产生相应的操作控制信号，以便启动规定的动作；

(3) 指挥并控制CPU、内存和输入/输出设备之间数据流动的方向。

运算器：由算术逻辑单元（ALU)、累加寄存器、数据缓冲寄存器和程序状态寄存器组成，它是数据加工处理部件。运算器有两个主要功能：

(1) 执行所有的算术运算；

(2) 执行所有的逻辑运算，并进行逻辑测试，如零值测试或两个值的比较。


CPU
算术逻辑单元
c c c
ALU
取指令控制
执行控制
时钟
操作控制器
时序产生器
状态反馈
程序状态寄存器
c
累加器AC
指令译码器
指令寄存器IR
程序计数器PC
c
c
地址寄存器AR
c
缓冲寄存器DR
c
存储器
地址总线
AB
I/O
数据总线
DB

CPU中的主要寄存器

CPU中至少要有六类寄存器。 这些寄存器用来暂存一个计算机字。根据需要，可以扩充其数目。

数据缓冲寄存器（DR）：用来暂时存放由内存储器读出的一条指令或一个数据字；当向内存存入一个数据字时，也暂时将它们存放在数据缓冲寄存器中。缓冲寄存器的作用是：

(1) 作为CPU和内存、外部设备之间信息传送的中转站；

(2) 补偿CPU和内存、外围设备之间在操作速度上的差别；

(3) 在单累加器结构的运算器中，数据缓冲寄存器可兼作为操作数寄存器。

指令寄存器（IR）：用来保存当前正在执行的一条指令。当执行一条指令时，先把它从内存取到缓冲寄存器中，然后再传送至指令寄存器。指令划分为操作码和地址码字段，由二进制数字组成。为了执行任何给定的指令，必须对操作码进行测试，以便识别所要求的操作。指令译码器就是做这项工作的。指令寄存器中操作码字段的输出就是指令译码器的输入。操作码一经译码后，即可向操作控制器发出具体操作的特定信号。

程序计数器（PC）：程序计数器用来确定下一条指令的地址。在程序开始执行前，必须将程序的第一条指令所在的内存单元地址送入PC。当执行指令时，CPU将自动修改PC的内容，以便使其存放的总是将要执行的下一条指令的地址。由于大多数指令都是按顺序来执行的，所以修改的过程通常只是简单的对PC加1。但是，当遇到转移指令如JMP指令时，那么后继指令的地址必须从指令的地址段取得。因此程序计数器的结构应当是具有寄存信息和计数两种功能的结构。

地址寄存器（AR）：地址寄存器用来保存当前CPU所访问的内存单元的地址。由于在内存和CPU之间存在着操作速度上的差别，所以必须使用地址寄存器来保持地址信息，直到内存的读/写操作完成为止 。

当CPU和内存进行信息交换，即CPU向内存存/取数据时，或者CPU从内存中读出指令时，都要使用地址寄存器和数据缓冲寄存器。同样，如果把外围设备的设备地址作为像内存的地址单元那样来看待，那么，当CPU和外围设备交换信息时，同样要使用地址寄存器和数据缓冲寄存器。

累加寄存器（AC）：通常简称为累加器，它是一个通用寄存器。其功能是：当运算器的算术逻辑单元（ALU)执行算术或逻辑运算时，为ALU提供一个工作区。累加寄存器暂时存放ALU运算的结果信息。

目前CPU中的累加寄存器，多达16个，32个，甚至更多。其中任何一个可存放源操作数，也可存放结果操作数。在这种情况下，需要在指令格式中对寄存器号加以编址。

程序状态寄存器（Program State Word, PSW）：保存由算术指令和逻辑指令运行或测试的结果建立的各种条件码内容，如运算结果进位标志(C)，运算结果溢出标志（V)，运算结果为零标 志(Z)，运算结果为负标志(N)等等。这些标志位通常分别由 1位触发器保存。

除此之外，状态条件寄存器还保存中断和系统工作状态等信息，以便使CPU和系统能及时了解机器运行状态和程序运行状态。因此，状态条件寄存器是一个由各种状态条件标志拼凑而成的寄存器。

操作控制器

操作控制器的功能，是根据指令操作码和时序信号，产生各种操作控制信号，以便正确地建立数据通路，从而完成取指令和执行指令的控制。

根据设计方法不同，操作控制器可分为时序逻辑型、存储逻辑型、时序逻辑与存储逻辑结合型三种。

1. 硬布线控制器

是采用时序逻辑技术来实现的；

2. 微程序控制器

是采用存储逻辑来实现的；

3. 前两种方式的组合

CPU的主要技术参数

1. **字长。**CPU可同时处理的二进制数据的位数。CPU按照其字长可以分为：8位CPU、16位CPU、32位CPU以及64位CPU等。

2. **内部工作频率。**又称为内频或主频，它是衡量CPU速度的重要参数。内部工作频率的倒数是时钟周期，这是CPU中最小的时间元素。

8086和8088执行一条指令平均需要12个时钟周期；80286和80386每条指令大约要4.5个时钟周期；80486每条指令大约2个时钟周期；Pentium具有双指令流水线，使得每个时钟周期执行1到2条指令；而Pentium Ⅱ/III每个时钟周期可以执行3条或更多的指令。

3. **外部工作频率。**是由主板为CPU提供的基准时钟频率。在早期，CPU的内频就等于外频。目前，CPU的内频越来越高，如果外频设计得跟内频同步，则主存都将无法跟上CPU的速度，从而出现了所谓的内部倍频技术：内频＝外频×倍频。

4. 片内Cache的容量和速度。过去的CPU一般没有片内Cache，而近年的CPU普遍均设有片内Cache。片内Cache的运行速度与内频相同或接近，容量为几十KB～几百KB。

5. 地址总线宽度。决定了CPU可以访问的最大的物理地址空间。

6. 数据总线宽度。决定了CPU与外部Cache、主存以及输入输出设备之间进行一次数据传输的信息量。

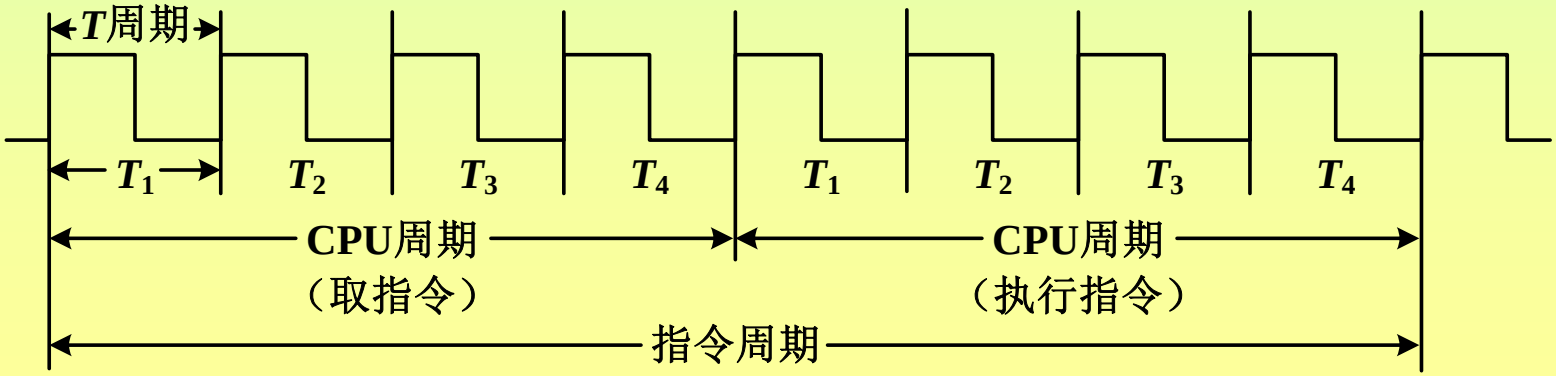
7. 制造工艺。线宽是指芯片上的最基本功能单元—门电路的宽度，因为实际上门电路之间连线的宽度与门电路的宽度相同，所以可以用线宽来描述制造工艺。Pentium Ⅲ的线宽是0.25微米，晶体管数达到9.5M个；Pentium 4的线宽是0.18微米，晶体管数达到42M个。

指令周期

指令周期：取出并完成一条指令的时间。由于各种指令的功能不同，其指令周期也不尽相同。一个指令周期包含若干个CPU周期。

CPU周期：也称**机器周期**。由于CPU内部速度块，而访问主存所花的时间较长，因此通常用主存中读取一个指令字的最短时间规定CPU周期。访问主存也就是一次总线传送，因此微型机中也称**总线周期**。一个指令周期至少需要两个CPU周期：取指周期和执行周期。

时钟周期：通常称为**节拍脉冲**或***T*周期**。是处理操作的最基本单位。一个CPU周期包含若干个时钟周期。

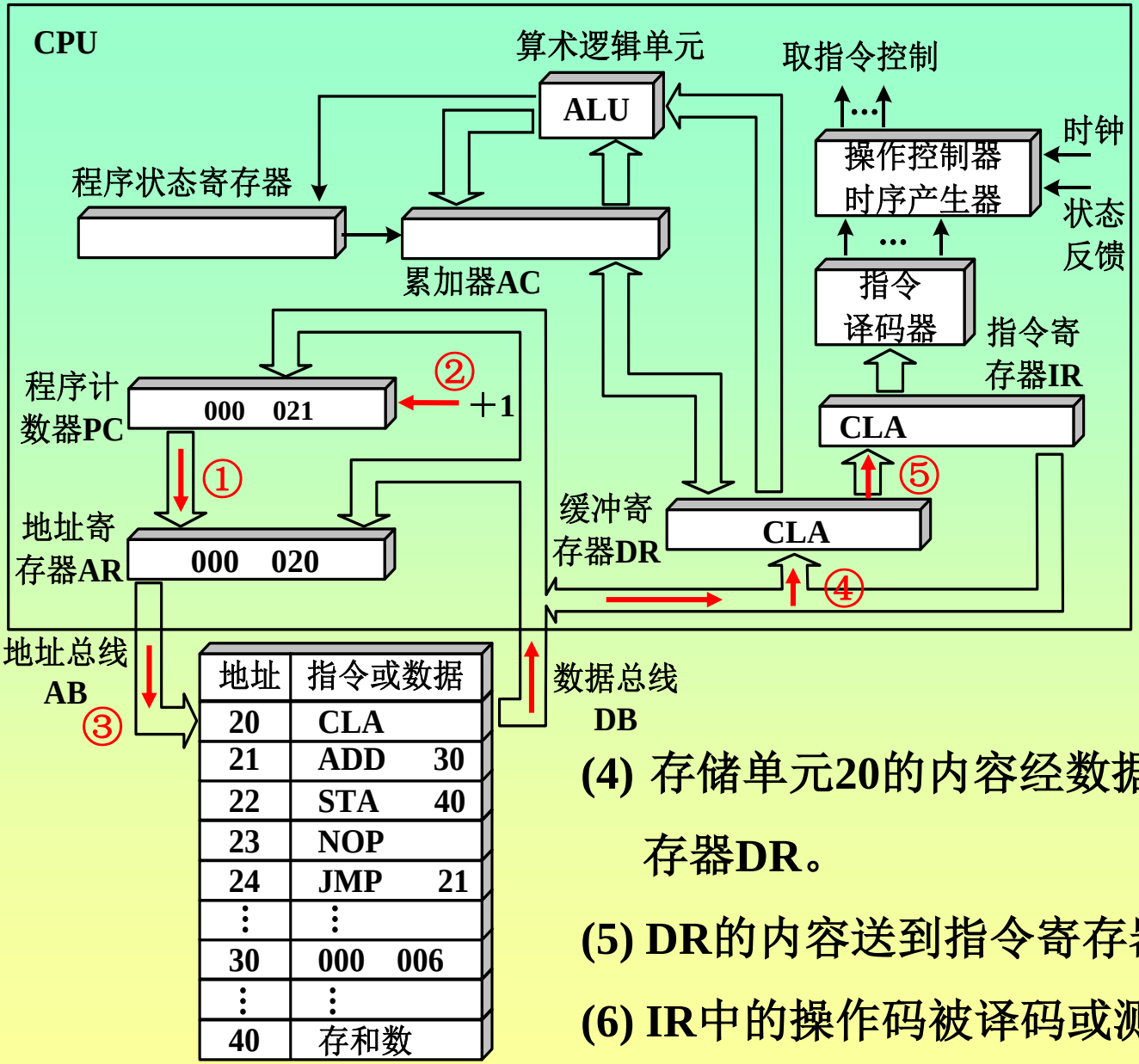


典型指令的指令周期

五条典型指令组成的程序

八进制地址	八进制内容	助　记　符
020	250　000	CLA
021	030　030	ADD　30
022	020　040	STA　40
023	000　000	NOP
024	140　021	JMP　21
⋮	⋮	
030	000　006	数据
031	000　040	
⋮	⋮	
040	和数存储单元	

指令格式见P147表4.7。CLA和NOP是非访存指令，ADD和STA是访存指令，JMP是程序控制指令。



1. 取指令阶段

(1) 程序计数器PC的内容20装入地址寄存器AR。

(2) PC内容加1，为取下一条指令做好准备。

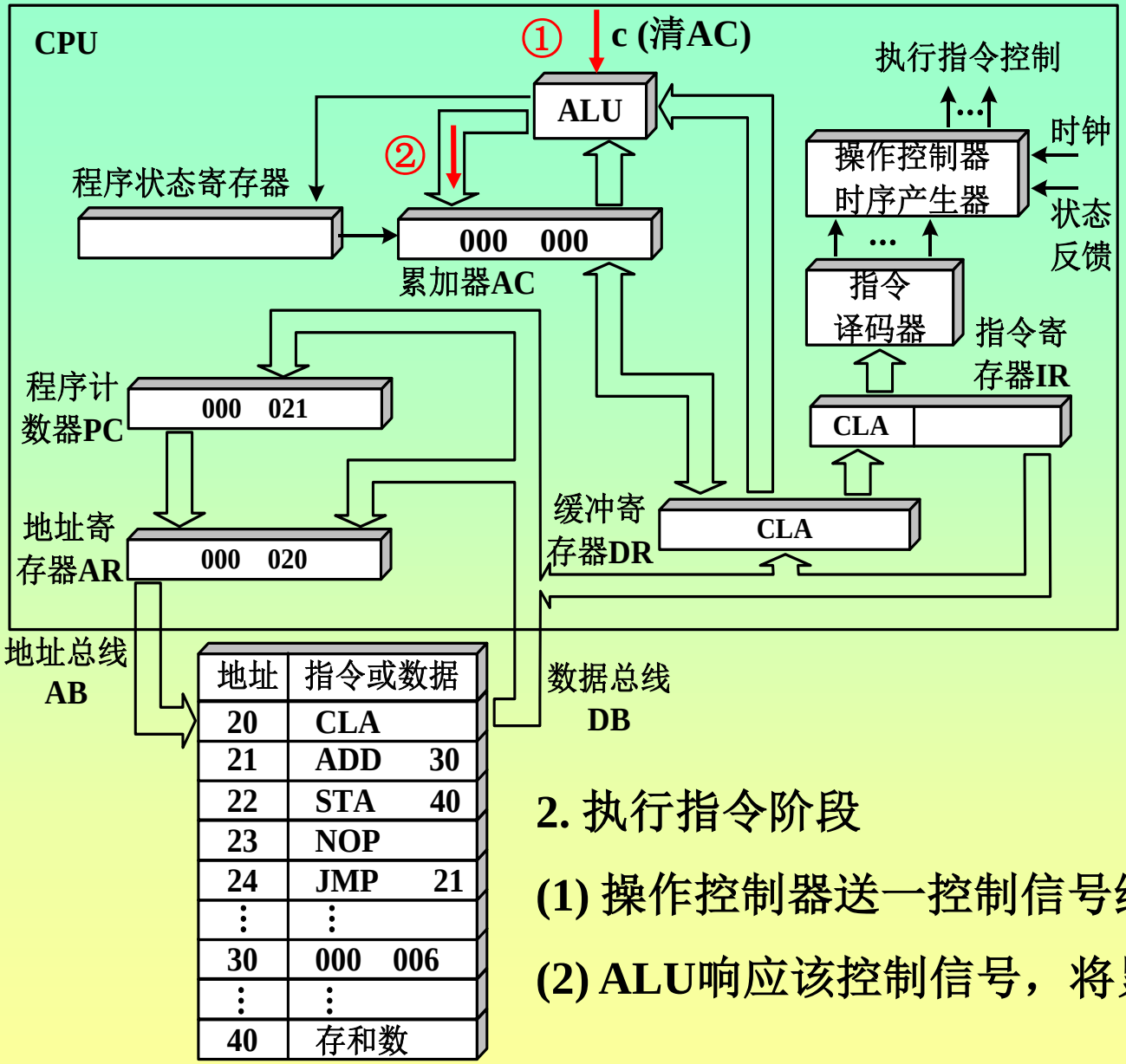
(3) AR内容放到地址总线上。

(4) 存储单元20的内容经数据总线传到数据缓冲寄存器DR。

(5) DR的内容送到指令寄存器IR。

(6) IR中的操作码被译码或测试。

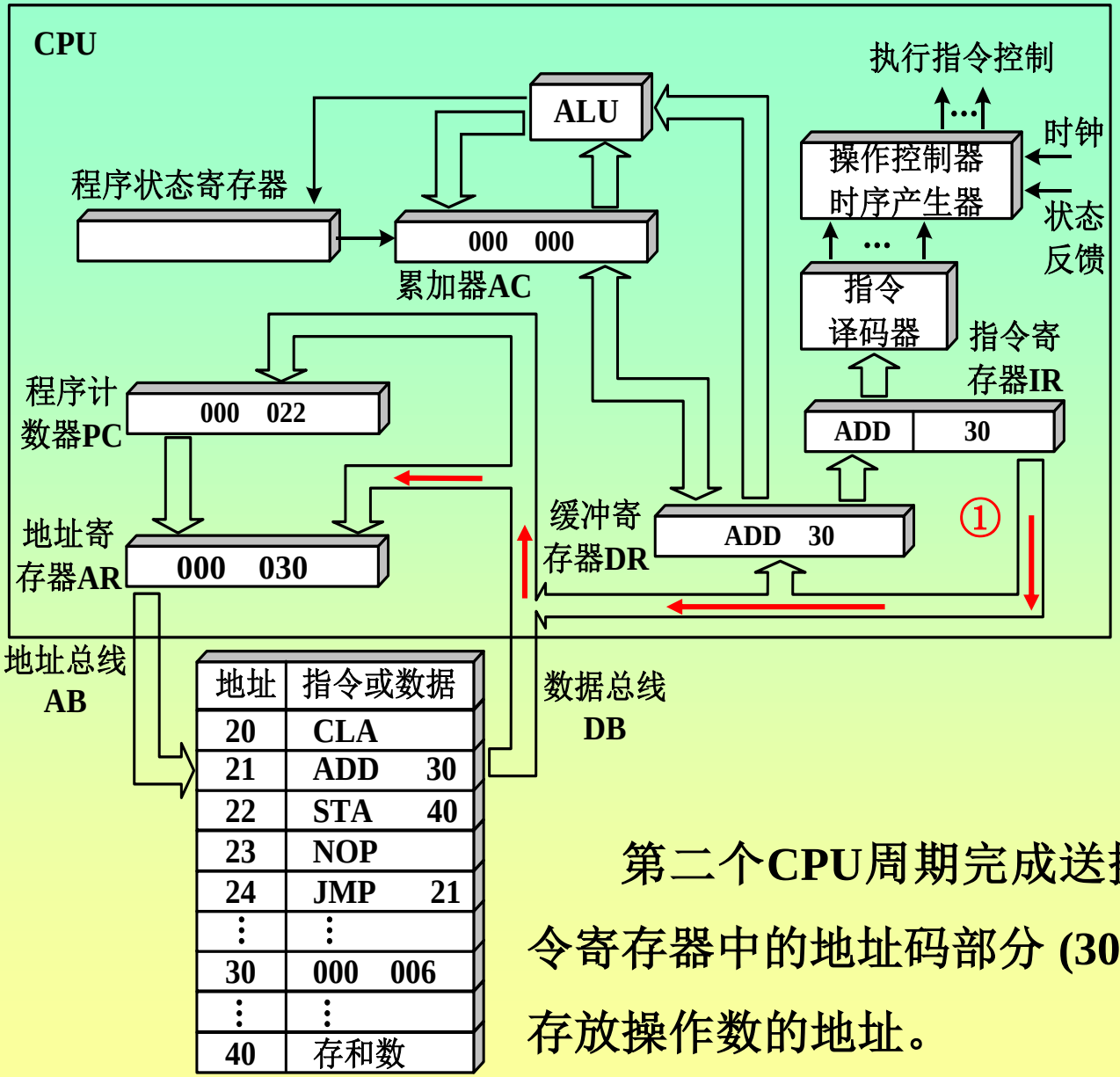
(7) CPU识别出是指令CLA。



2. 执行指令阶段

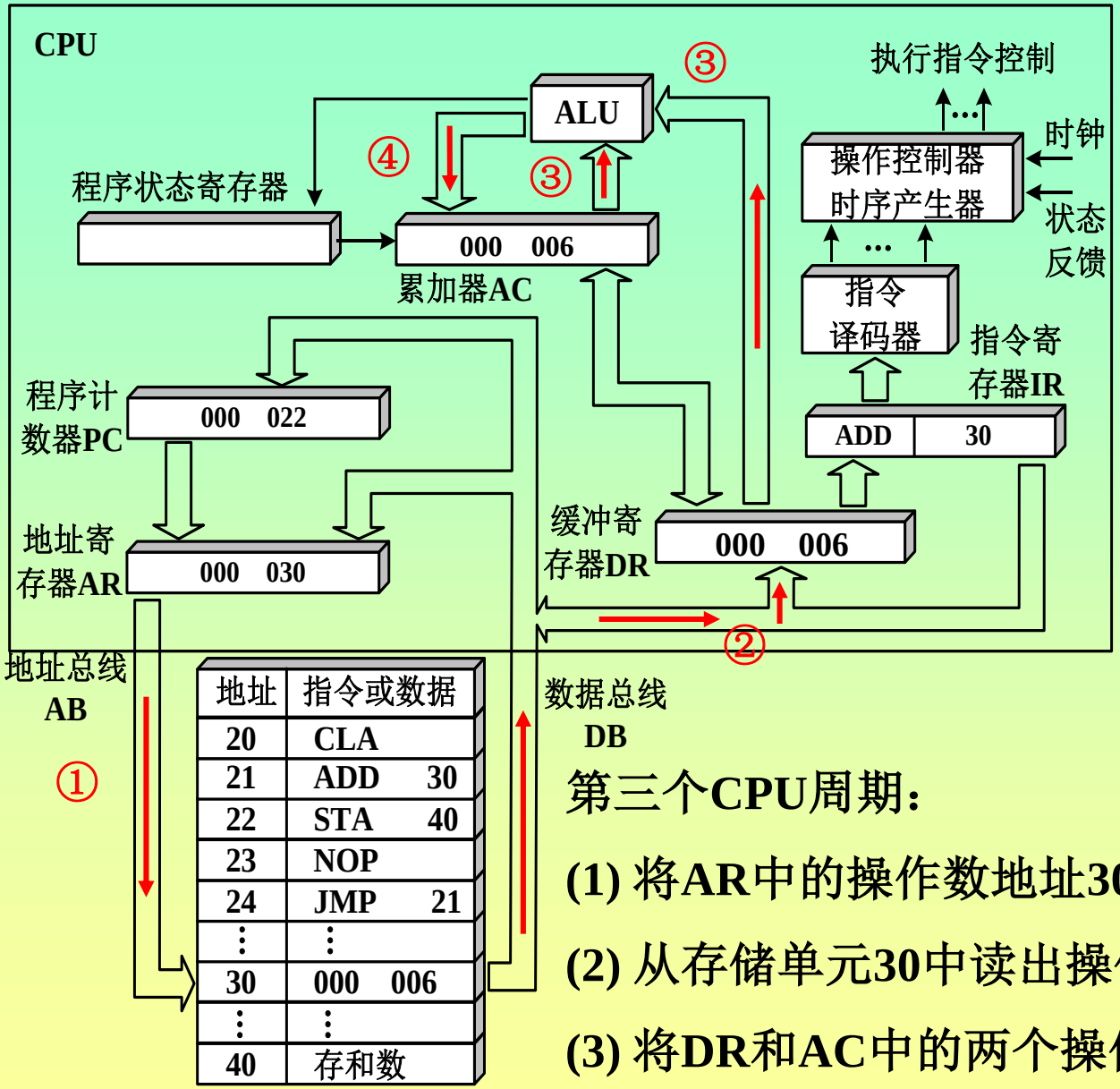
(1) 操作控制器送一控制信号给算术逻辑单元ALU。

(2) ALU响应该控制信号，将累加寄存器AC清零。



ADD指令的指令周期包括三个CPU周期。第一个CPU周期为取指令阶段，与CLA指令相同。第二和第三个CPU周期是执行指令阶段。

第二个CPU周期完成送操作数地址。CPU把指令寄存器中的地址码部分 (30) 装入AR，30是主存中存放操作数的地址。



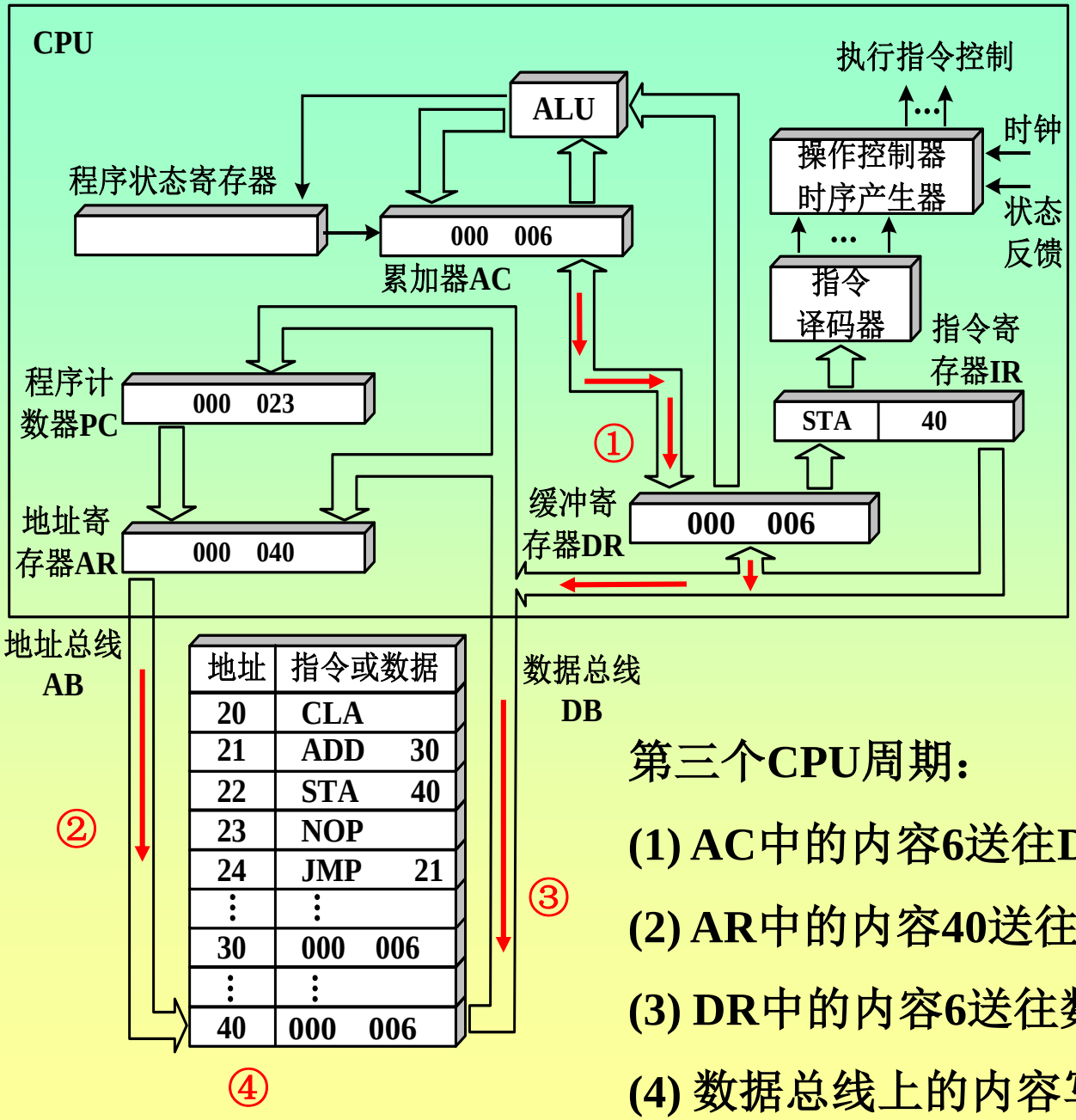
第三个CPU周期：

(1) 将AR中的操作数地址30发送到地址总线上。

(2) 从存储单元30中读出操作数6送到DR。

(3) 将DR和AC中的两个操作数送往ALU相加。

(4) 结果存入AC。



STA指令的指令周期包括三个CPU周期。第一个CPU周期为取指令阶段，第二和第三个CPU周期是执行指令阶段。第二个CPU周期将IR中的地址码40送往AR。

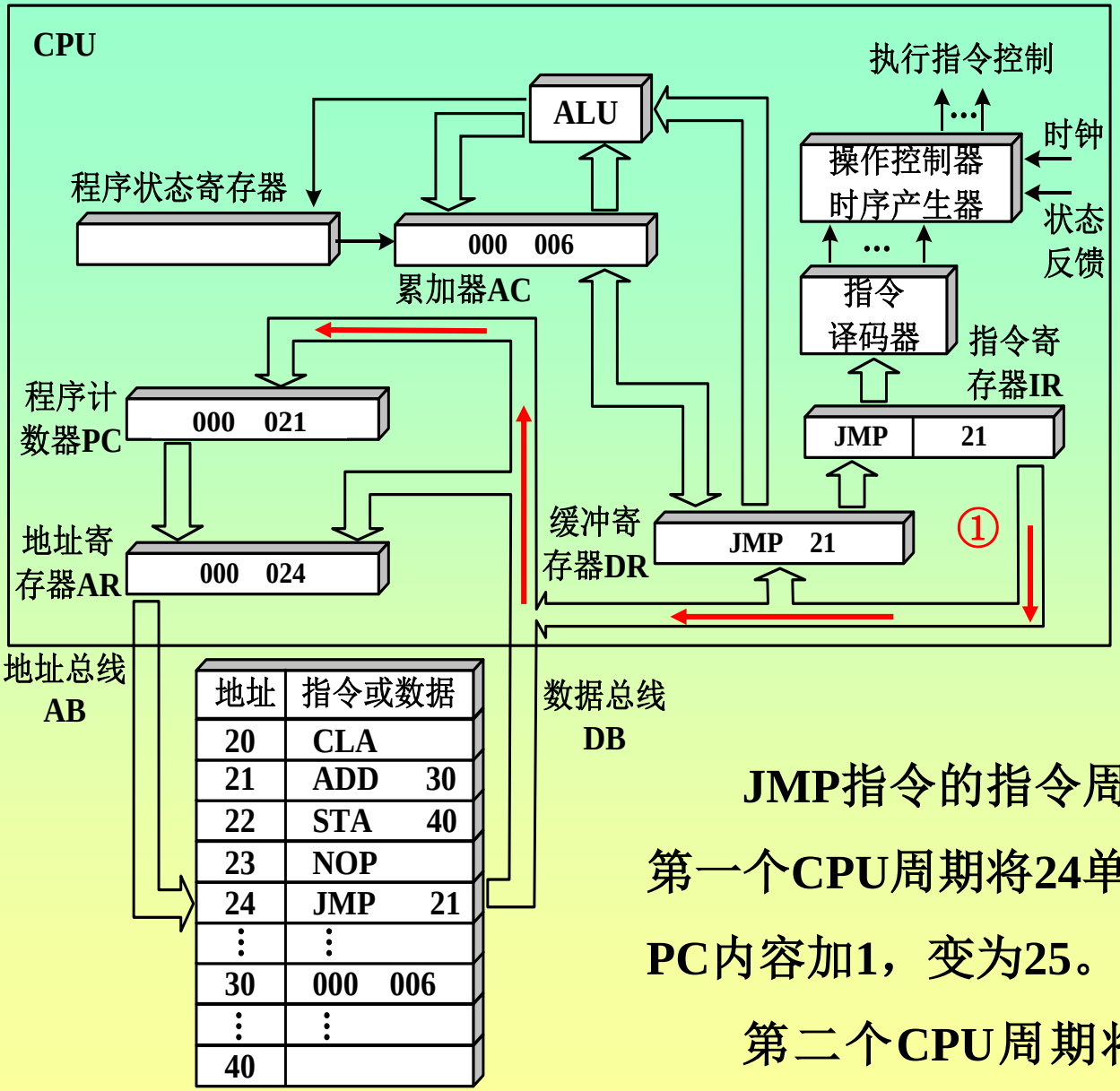
第三个CPU周期：

(1) AC中的内容6送往DR。

(2) AR中的内容40送往地址总线。

(3) DR中的内容6送往数据总线。

(4) 数据总线上的内容写入选定的存储单元。

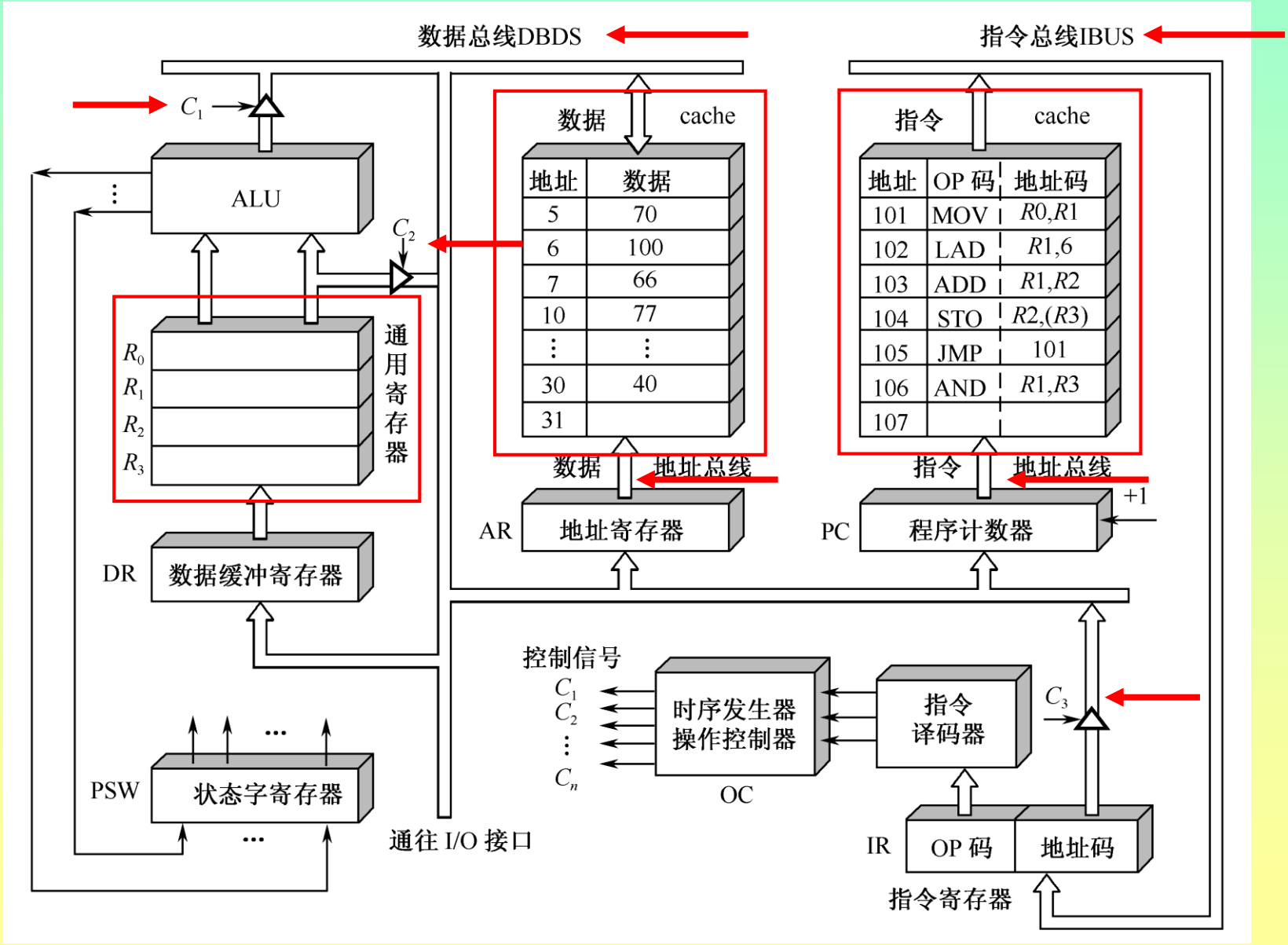


NOP指令的指令周期包括两个CPU周期。第一个CPU周期将23单元的指令送入IR。第二个CPU周期操作控制器不发出任何控制信号。

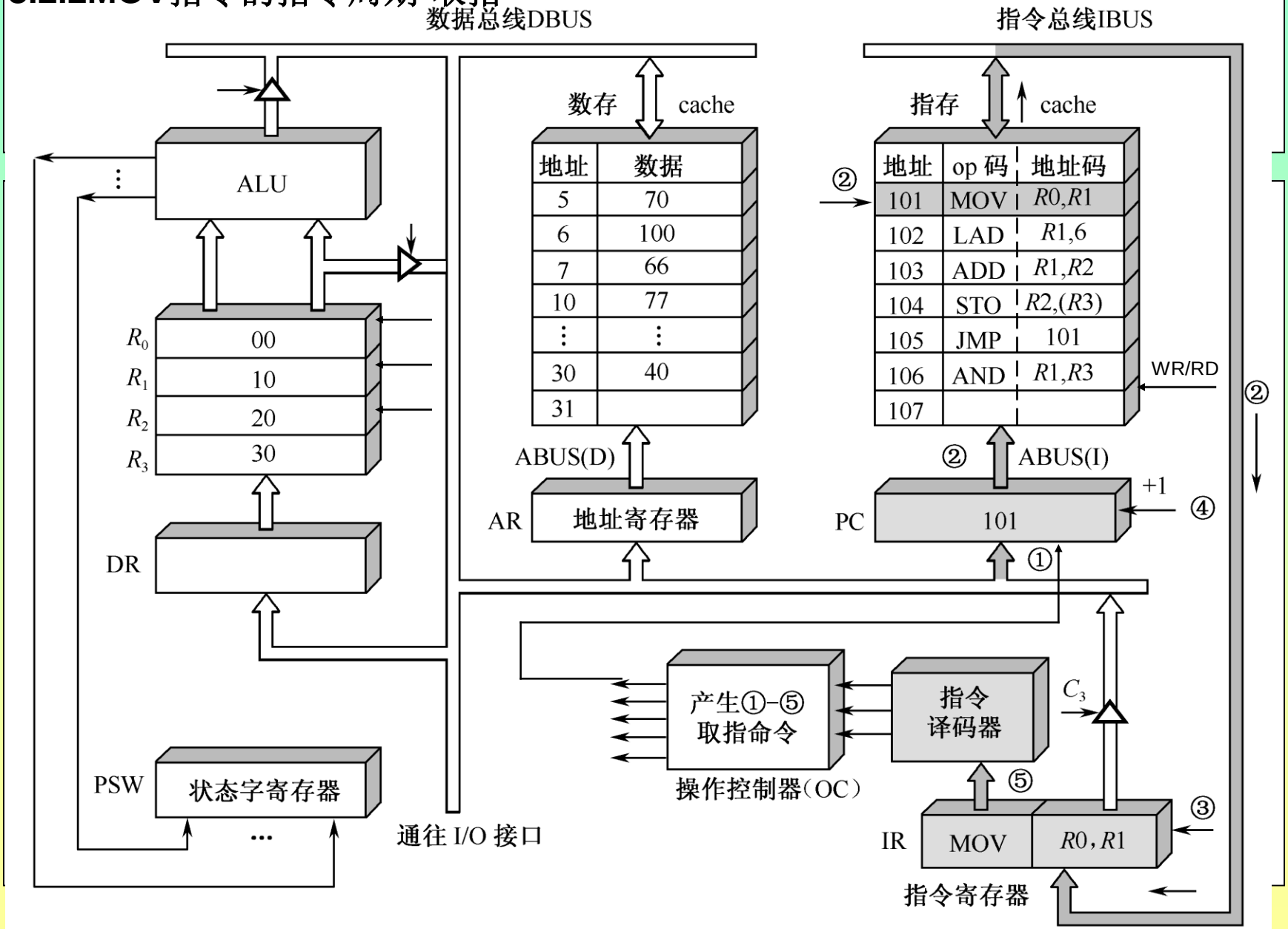
JMP指令的指令周期包括两个CPU周期。第一个CPU周期将24单元的指令送入IR，然后PC内容加1，变为25。

第二个CPU周期将IR中的地址码21送入PC，替代原来的内容25。

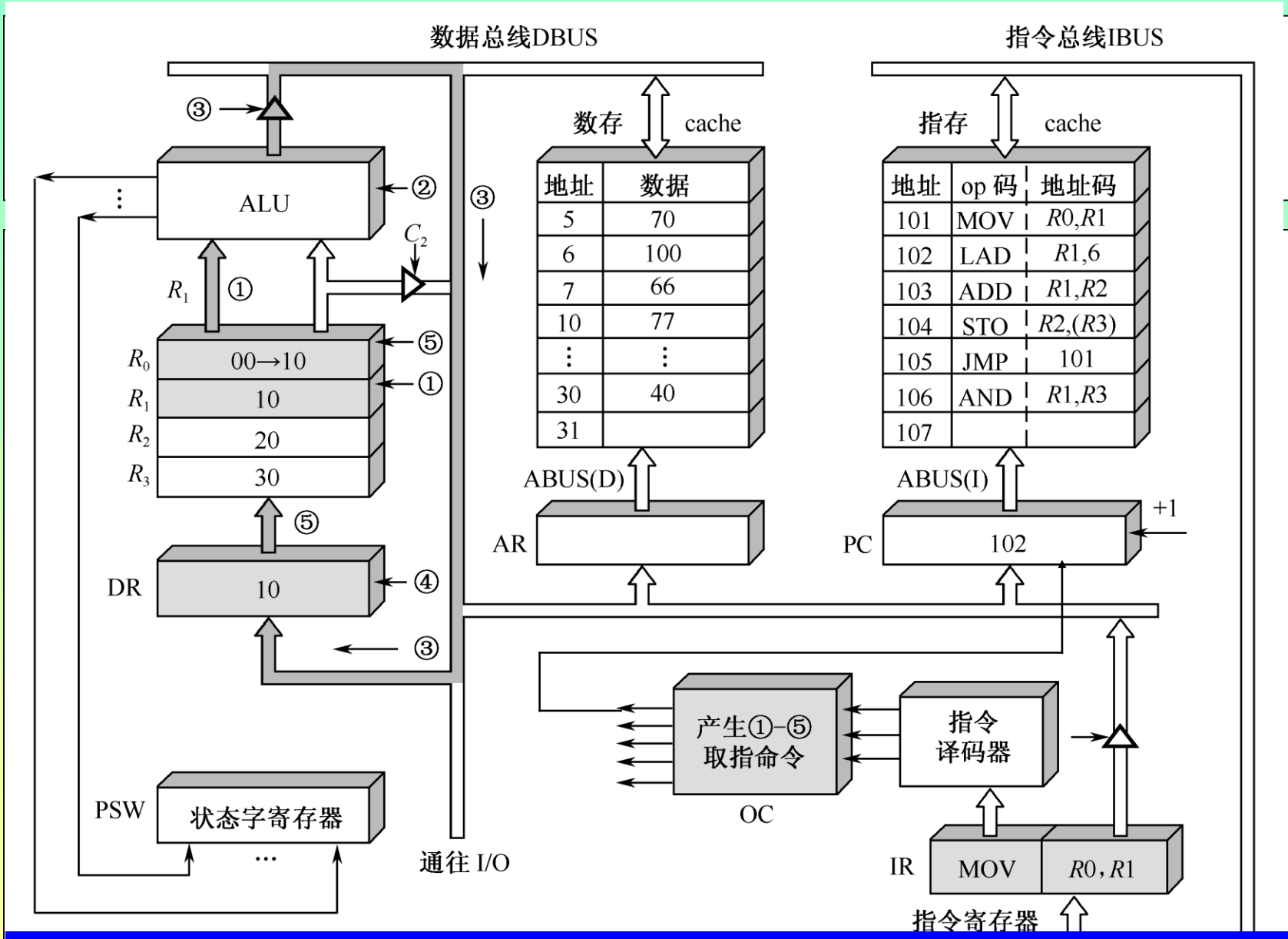
带cache以及通用寄存器的CPU模型



5.2.2MOV指令的指令周期-取指



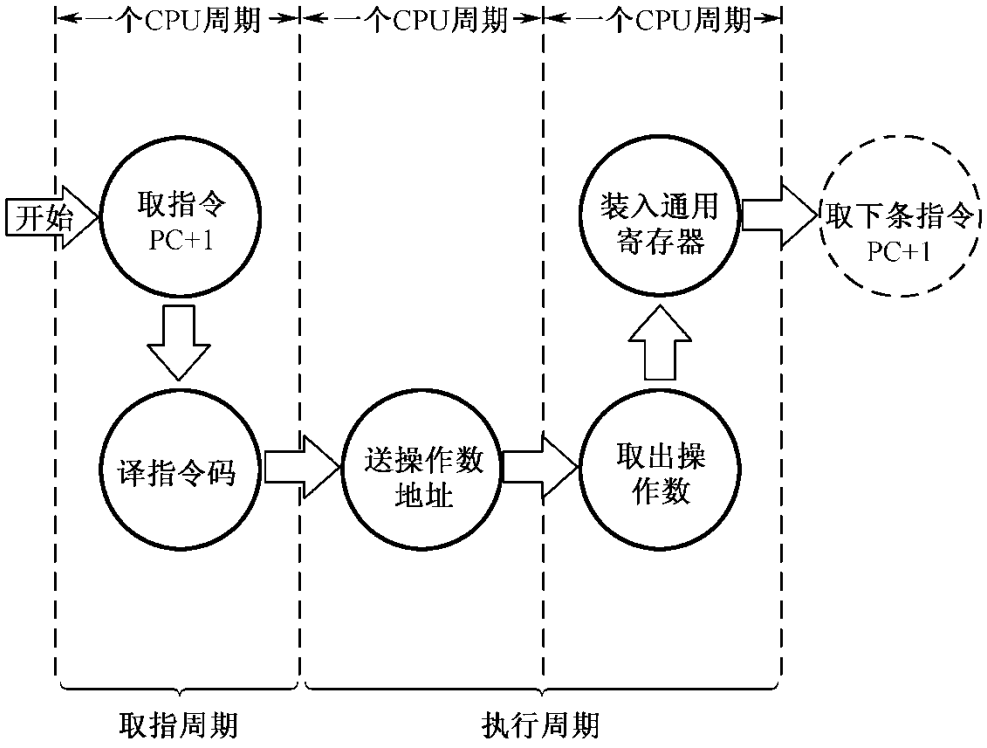
② ④ 程序计 ⑥ CPU识别出是MOV指令，至此，取指周期即告结束。 寄存器IR； ；



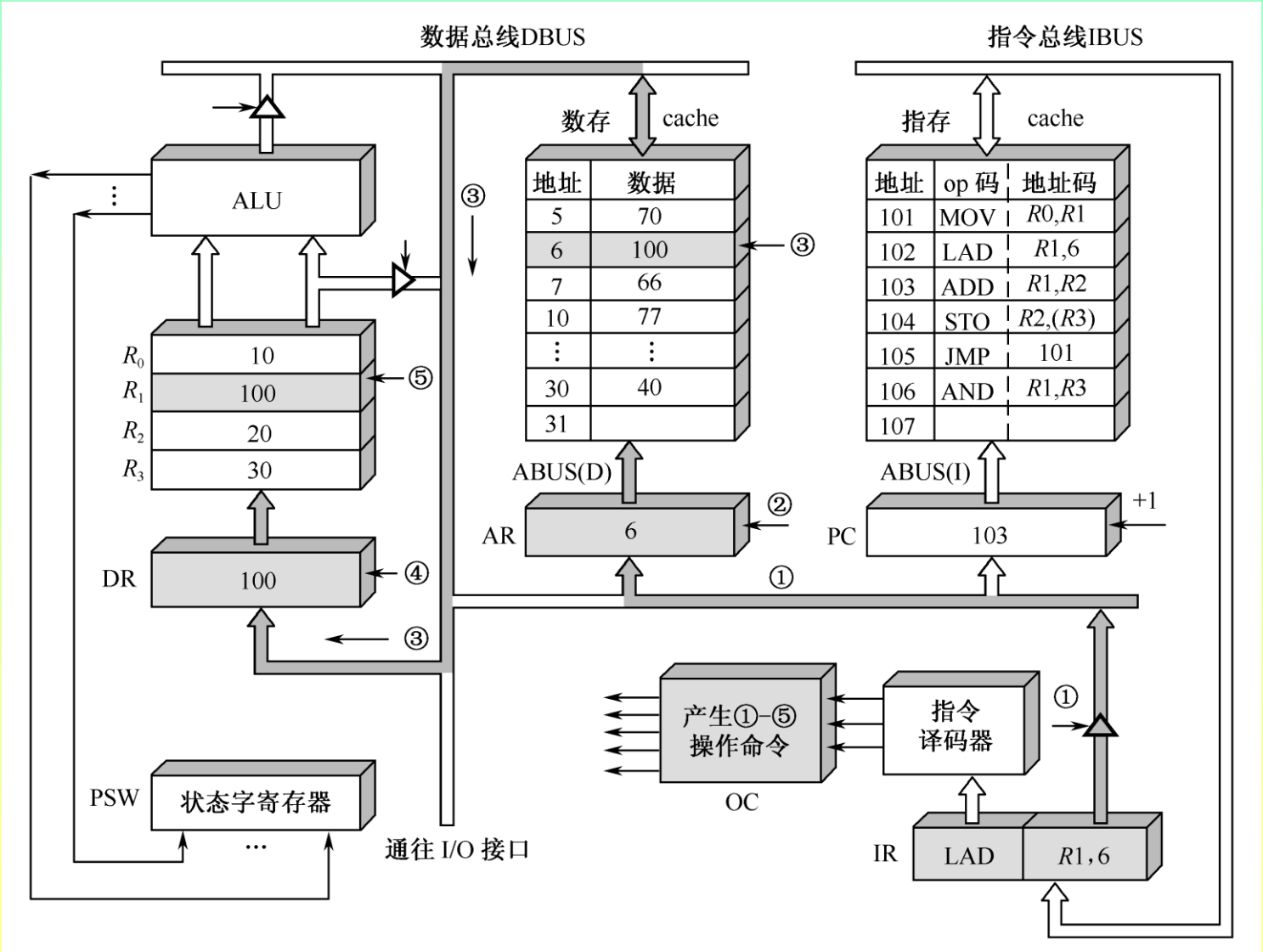
⑤ OC送出控制信号，将DR中的数据10打入到目标寄存器R0，R0的内容由　　上。00变为10。至此，MOV指令执行结束。　　；

5.2.3LAD指令的指令周期

- 取指周期
- 执行周期



5.2.3LAD指令的指令周期



5.2.5ADD指令的指令周期

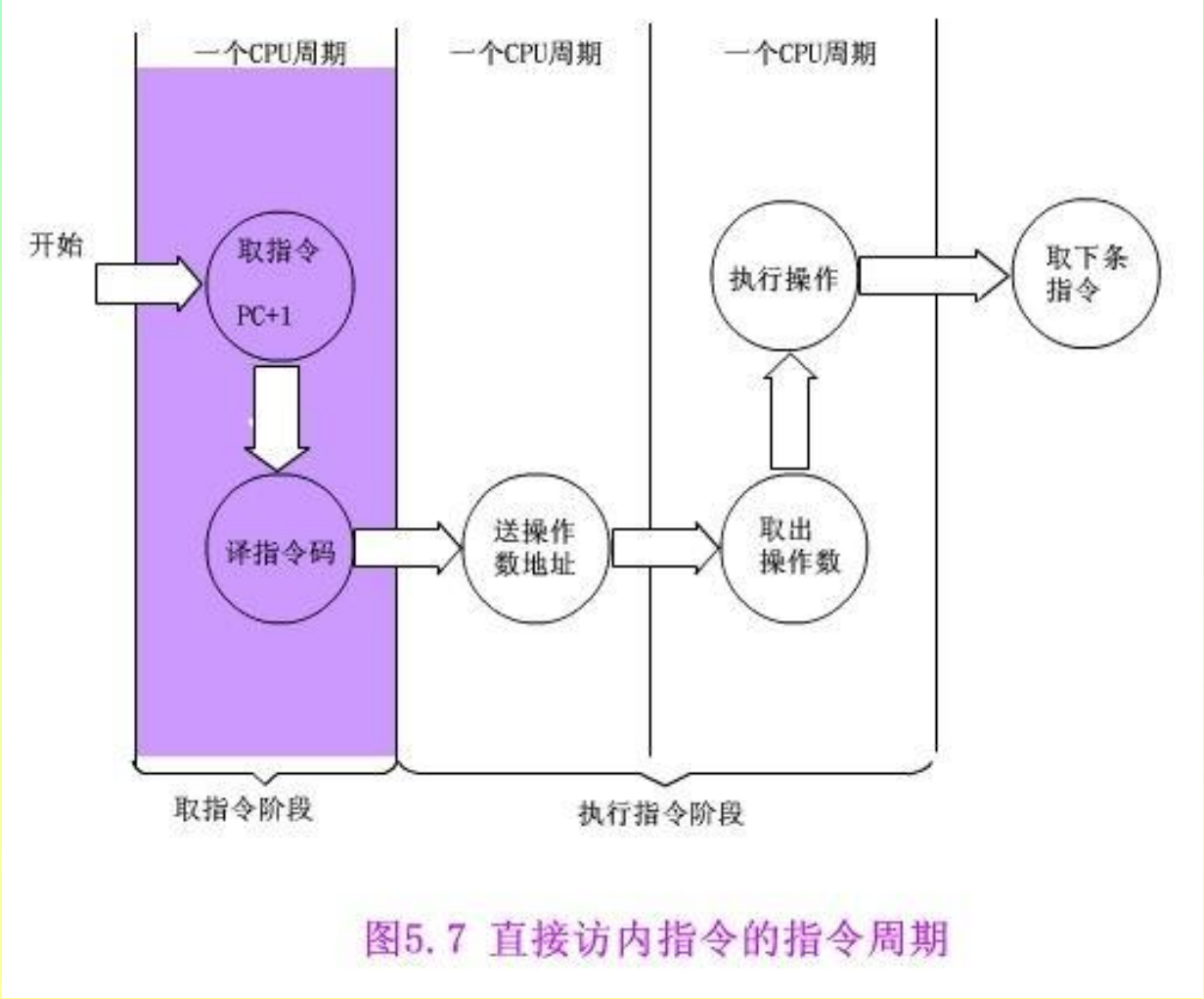
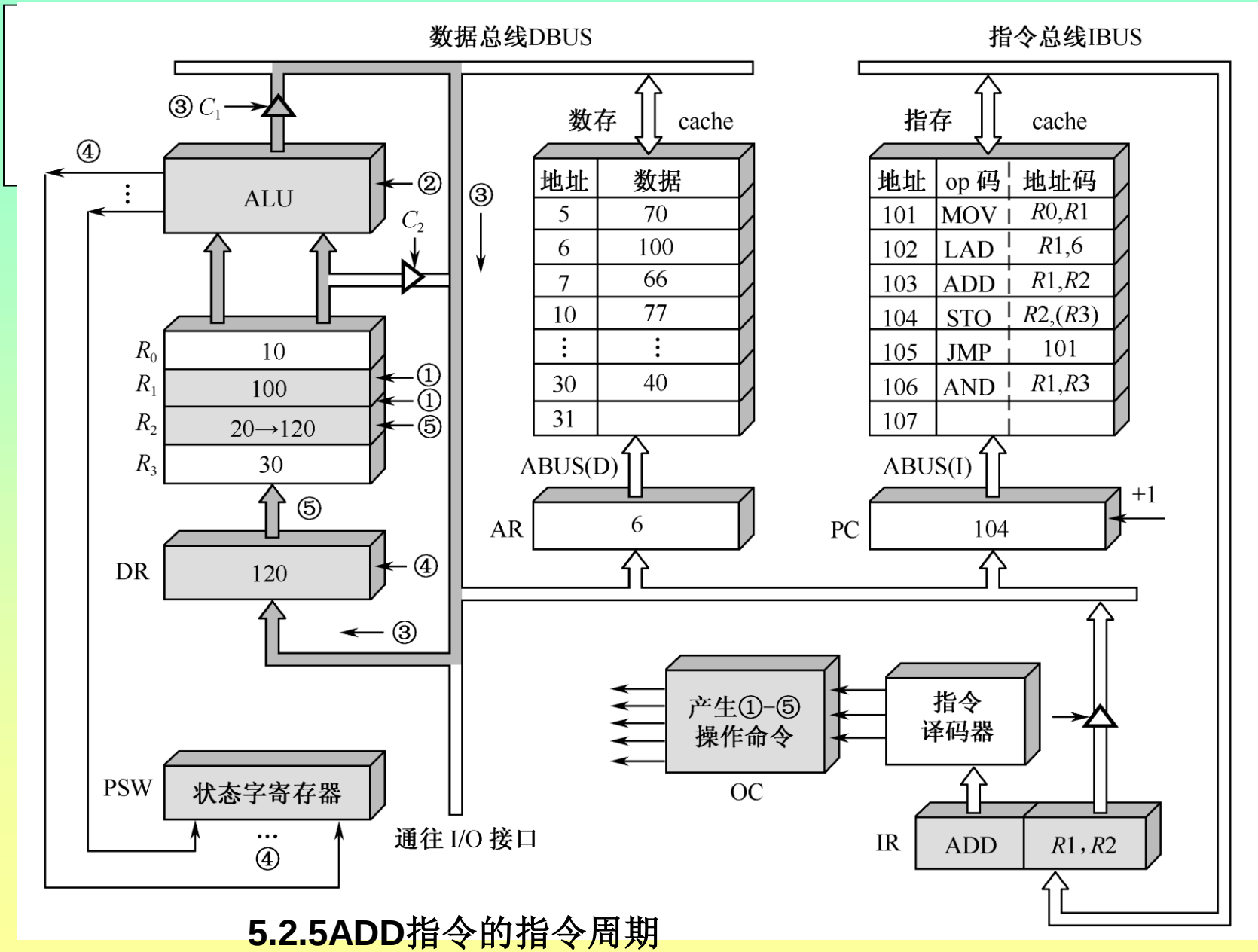
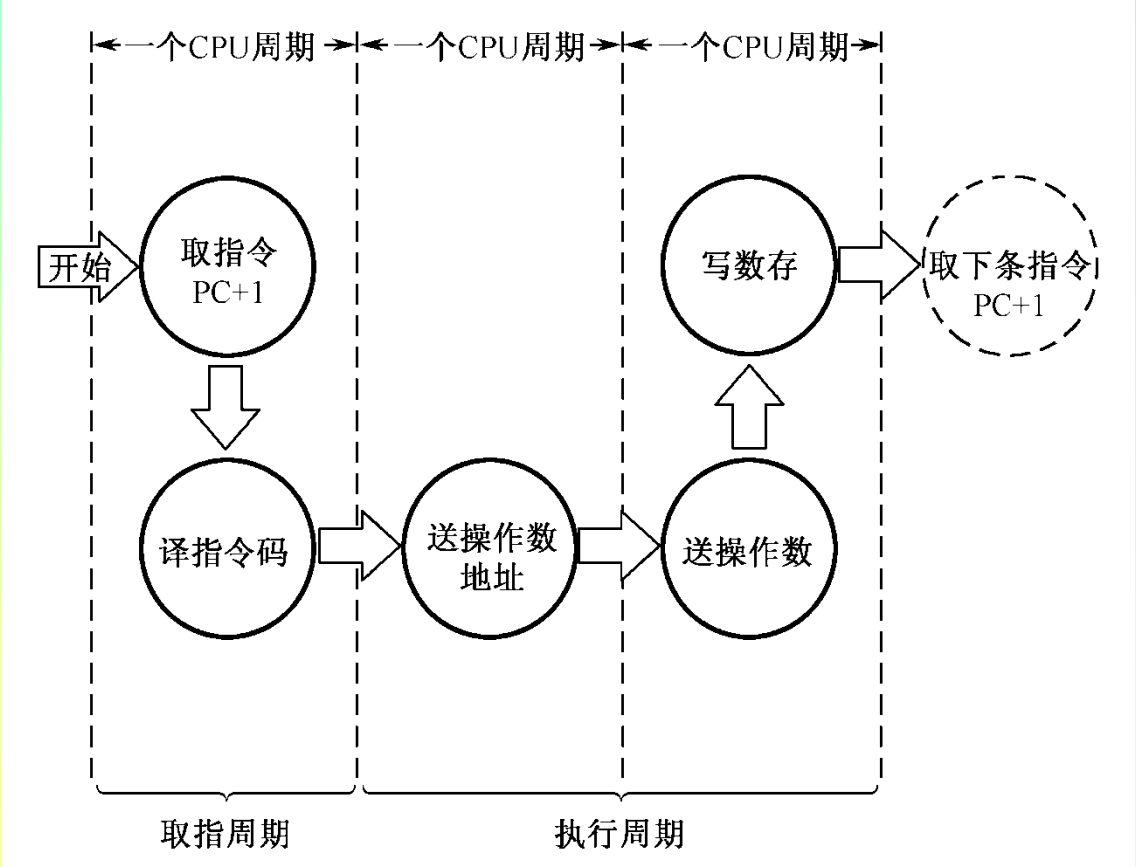


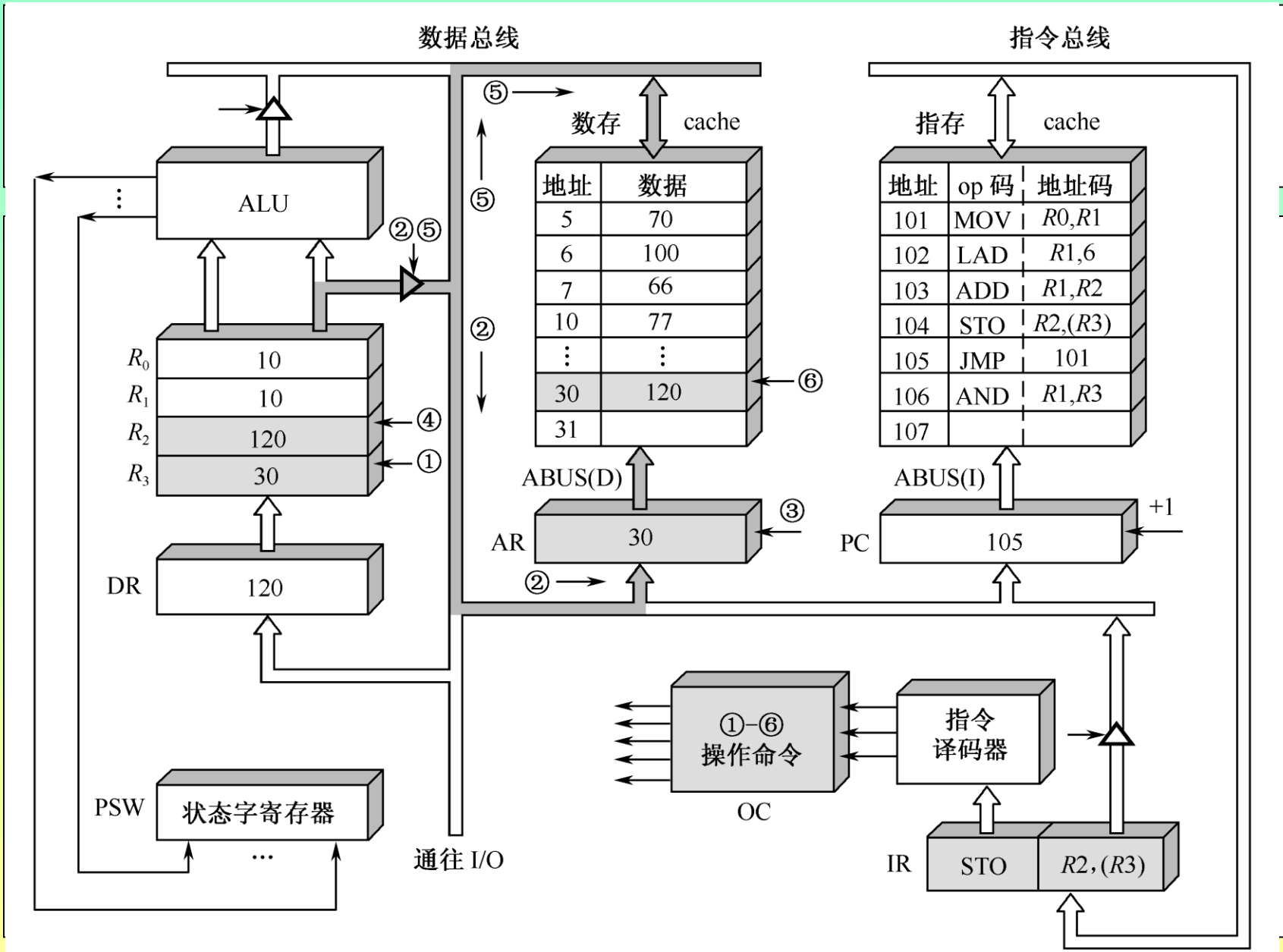
图5.7 直接访内指令的指令周期



5.2.5ADD指令的指令周期

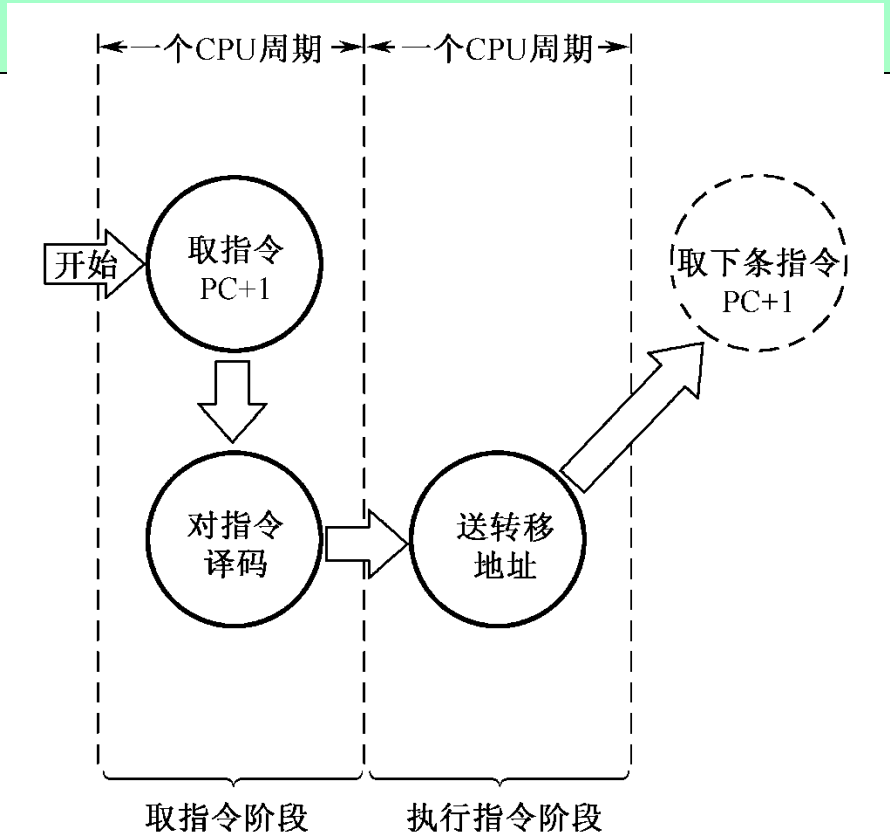
5.2.5STO指令的指令周期

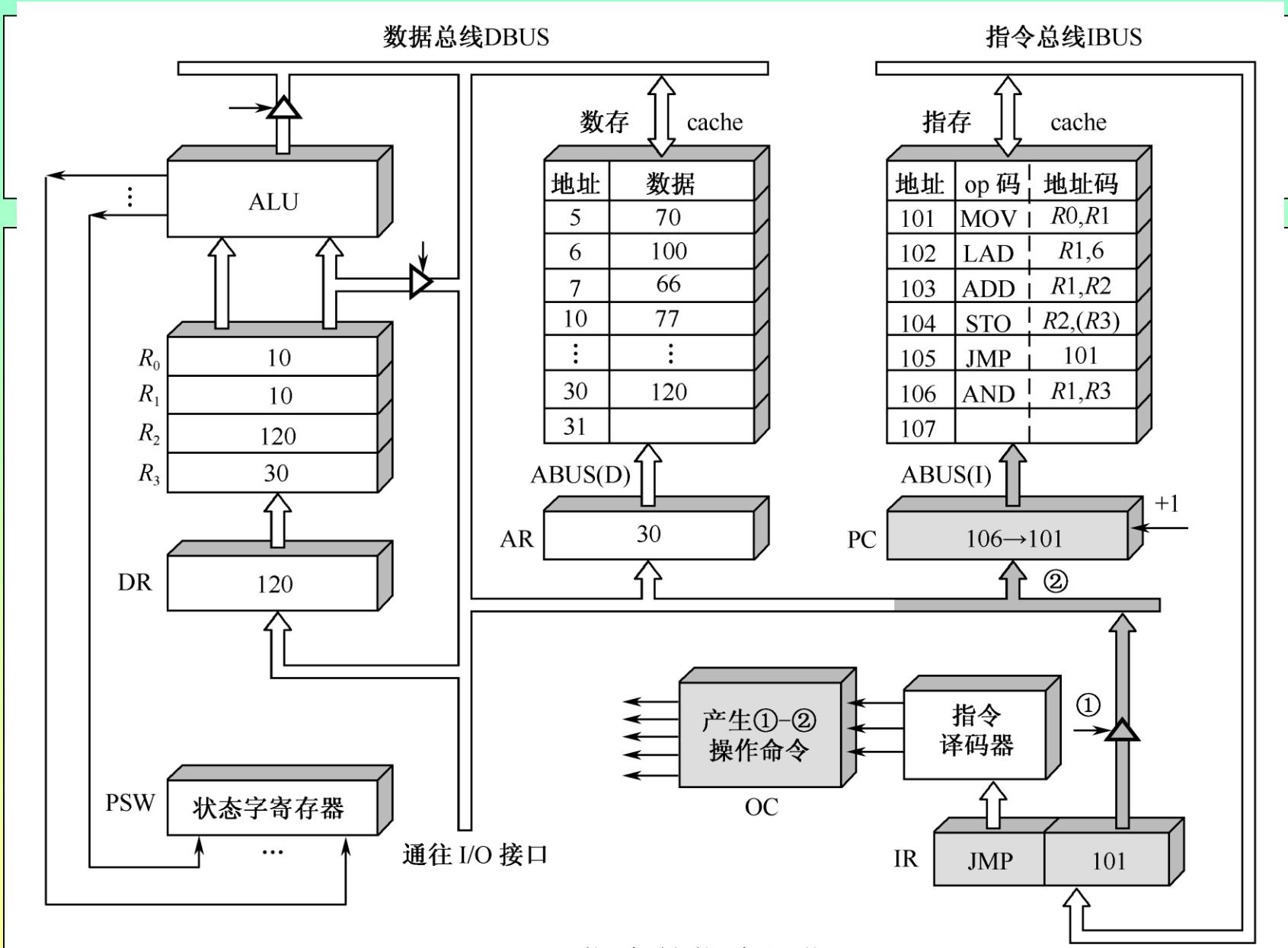




5.2.5STO指令的指令周期

5.2.6JMP指令的指令周期





5.2.6JMP指令的指令周期

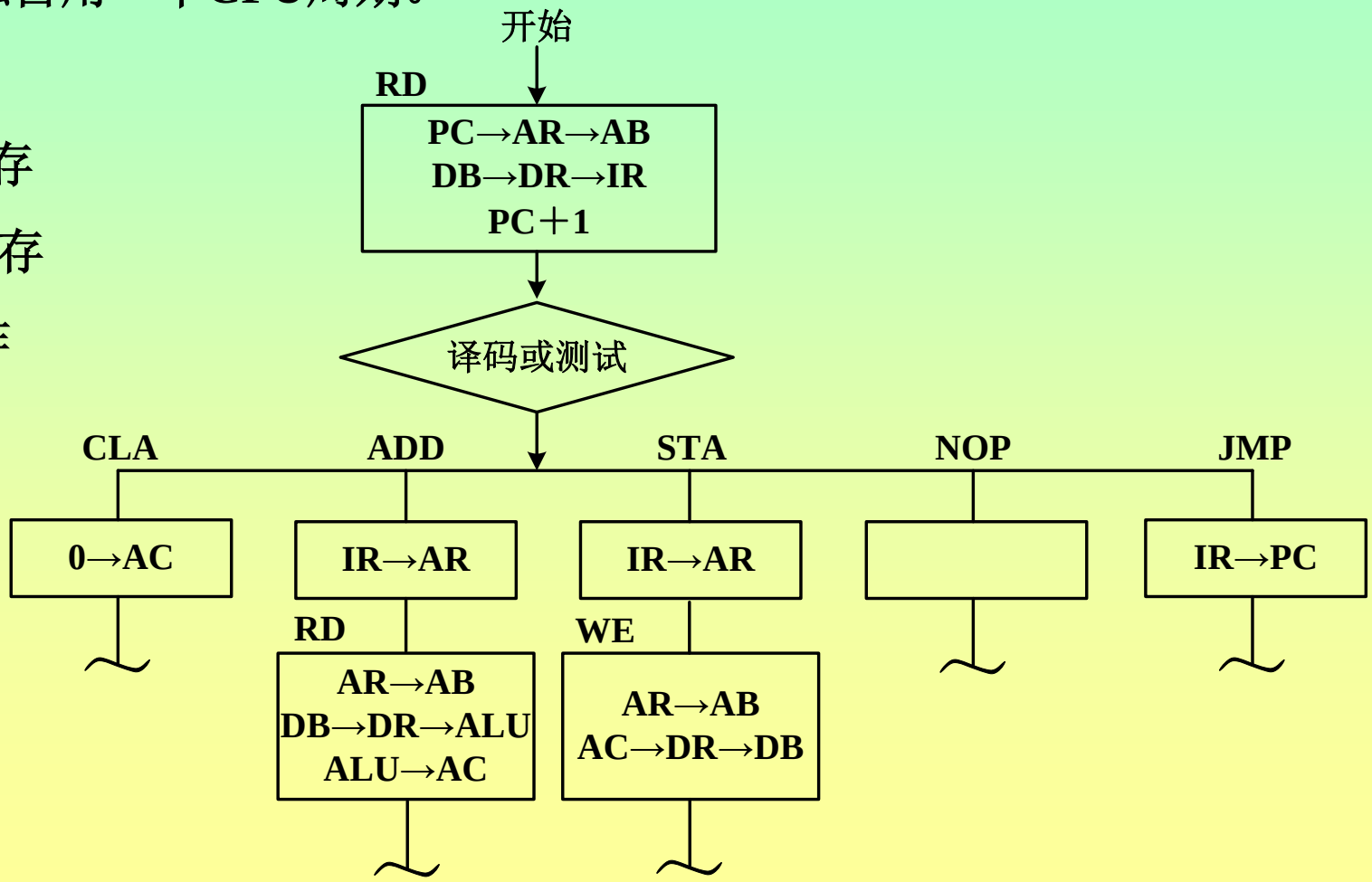
方框图

一个方框代表一个CPU周期，方框中的内容代表该CPU周期内完成的操作。菱形框表示某种判别或测试，在时间上它依附于前面一个方框的CPU周期，不单独占用一个CPU周期。

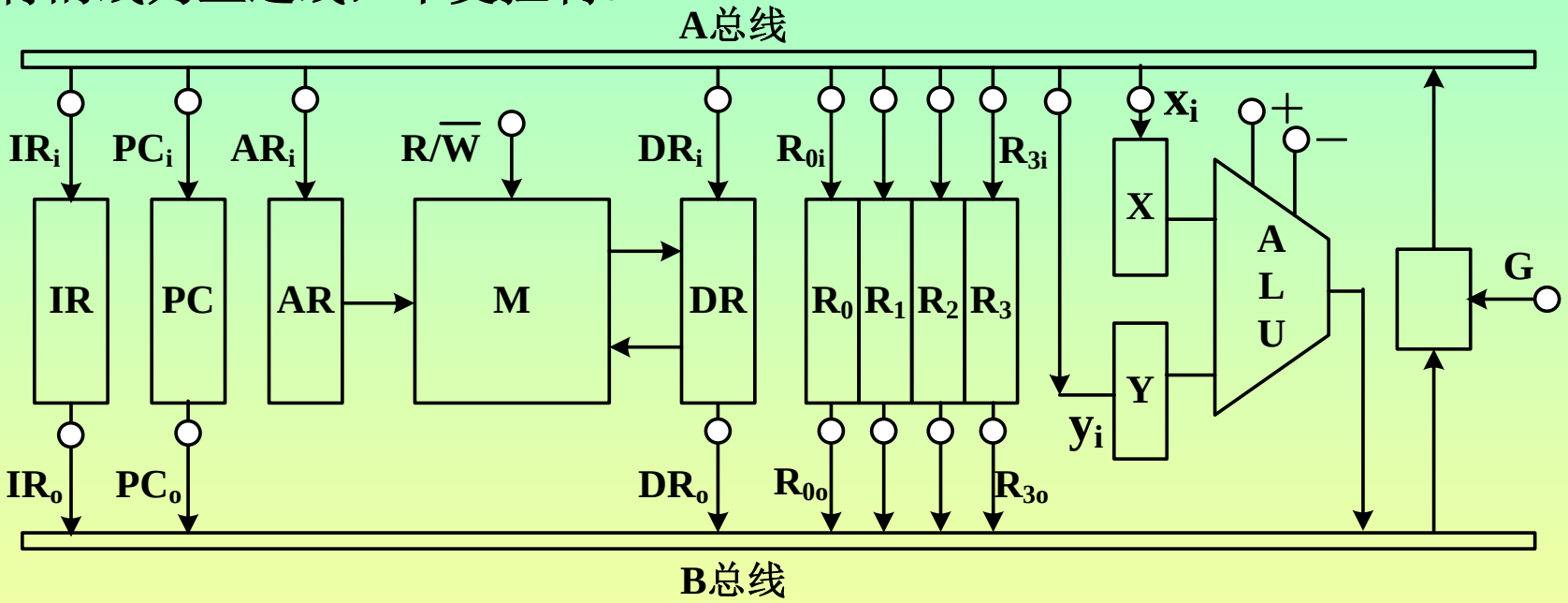
RD：读主存

WE：写主存

～：公操作

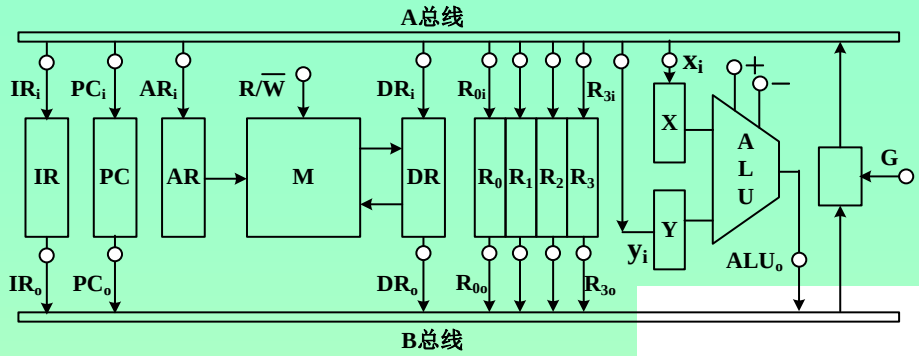


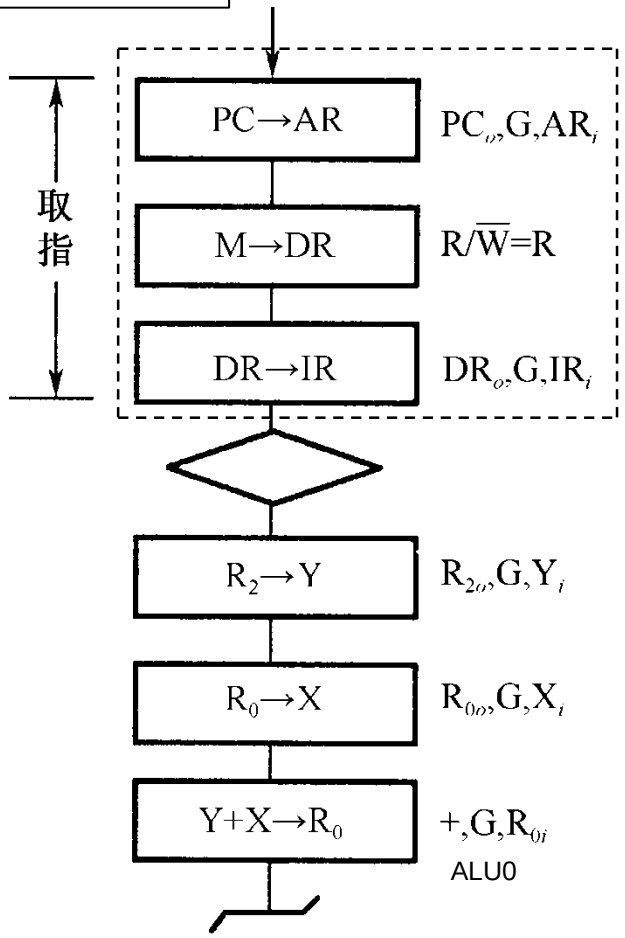
例：双总线结构机器的数据通路如图。主存M受R/$\overline{W}$信号控制，ALU受加减信号控制，信号G控制的是一个门电路。线上标注小圈表示有控制信号，例如y_i表示y寄存器的输入控制信号，R_{1o}表示寄存器R_1的输出控制信号。未标字符的线为直通线，不受控制。



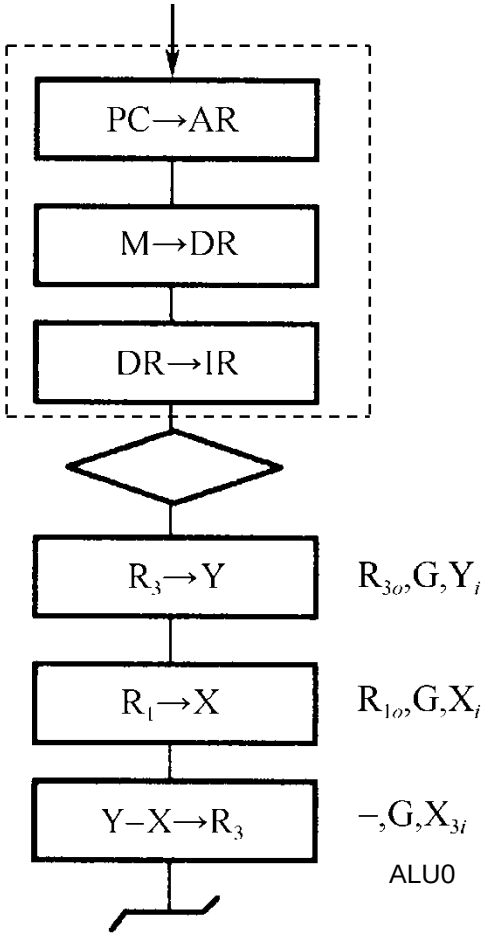
(1) ADD R_2, R_0指令完成$(R_0)+(R_2)\rightarrow R_0$功能。画出指令周期流程图，列出微操作控制信号序列。

(2) SUB R_1, R_3指令完成$(R_3)-(R_1)\rightarrow R_3$功能。画出指令周期流程图，列出微操作控制信号序列。





(a) 加法



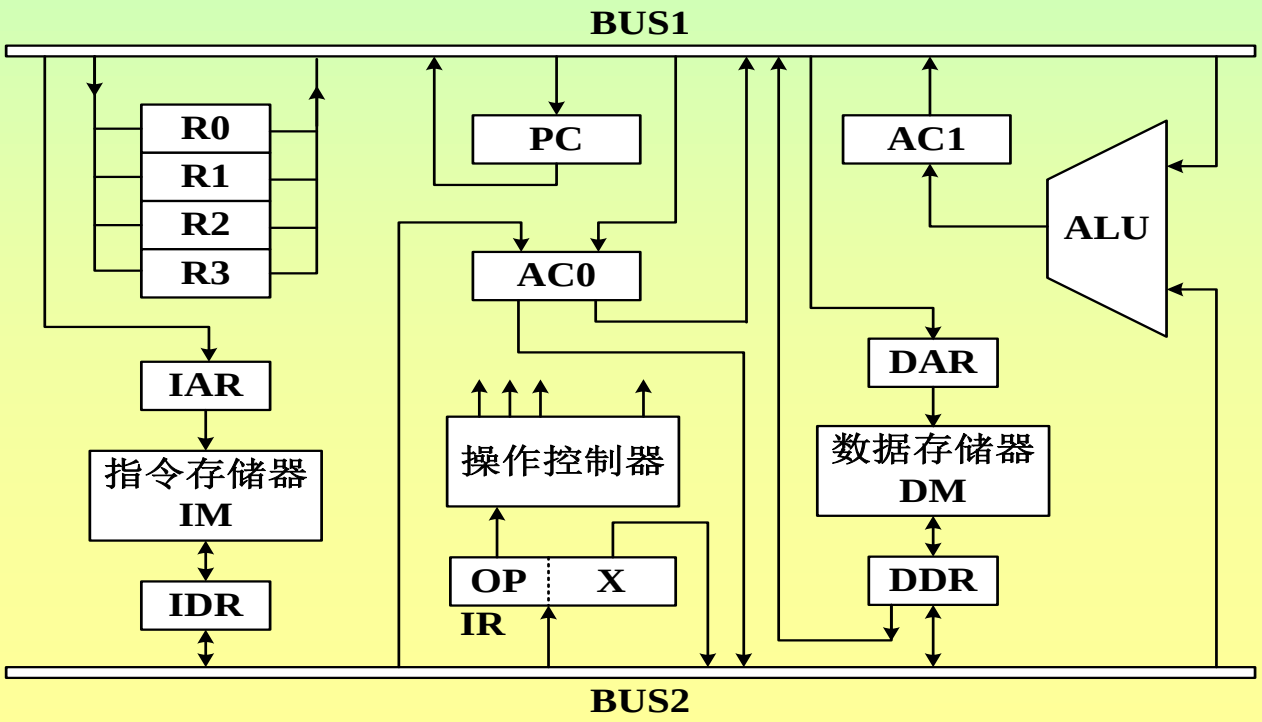
(b) 减法

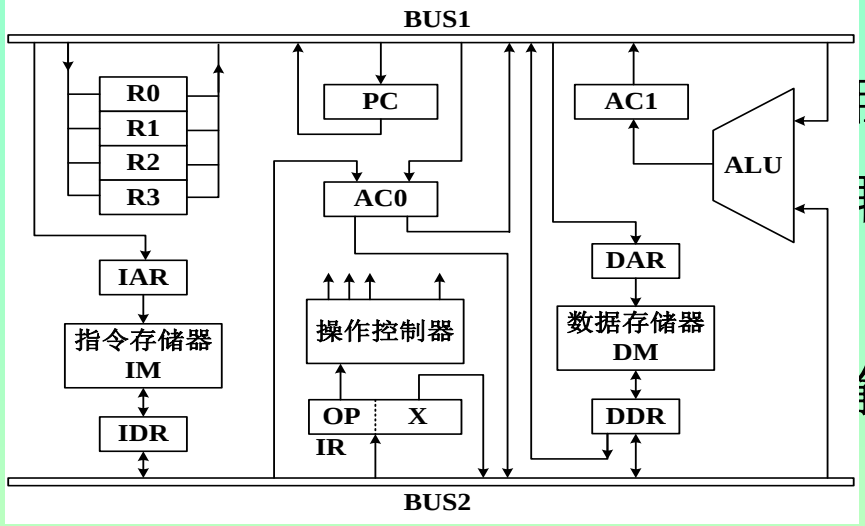
注意微操作控制信号（右边）

例：双总线结构计算机的数据通路如图。有两个独立的存储器，所有寄存器都有“打入”和“送出”控制命令，例如IAR_{in}为IAR的打入命令，DDR_{out2}为DDR到BUS2的送出命令，X_{out}为IR中X字段的送出命令；ALU有加减控制命令；指令存储器和数据存储器均有读写控制命令。设指令格式为：

OP	X

加法指令“ADD X(R1)”的功能是(AC0) ＋ ((R1)＋X) → AC1，即加法运算的两个操作数，一个来源于AC0，另一个来源于数据存储器中地址为(R1)＋X的存储单元。

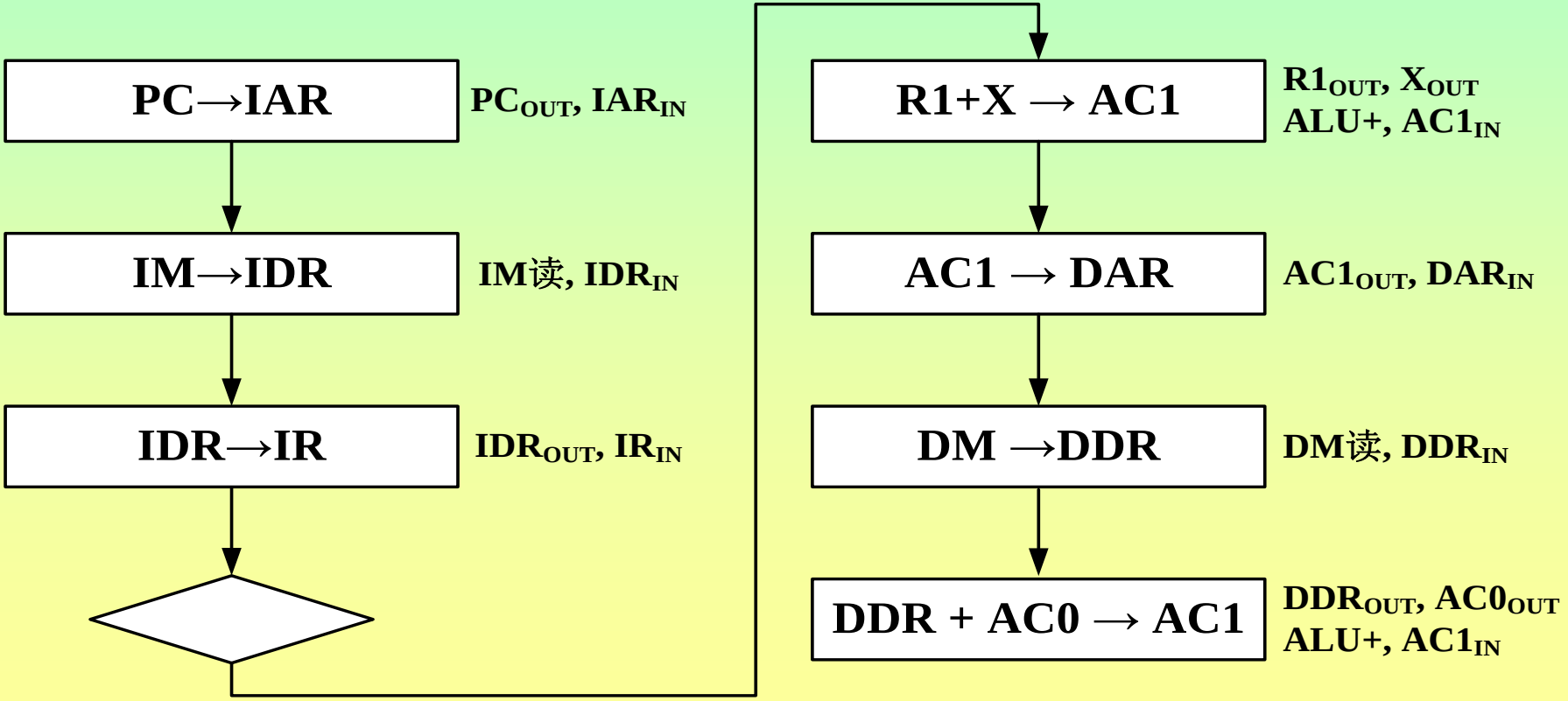




用何种寻址方式？ R、S

取指令到执行结束的操作序列，并说明每

输出端直接连接到BUS1？为什么？ 不行

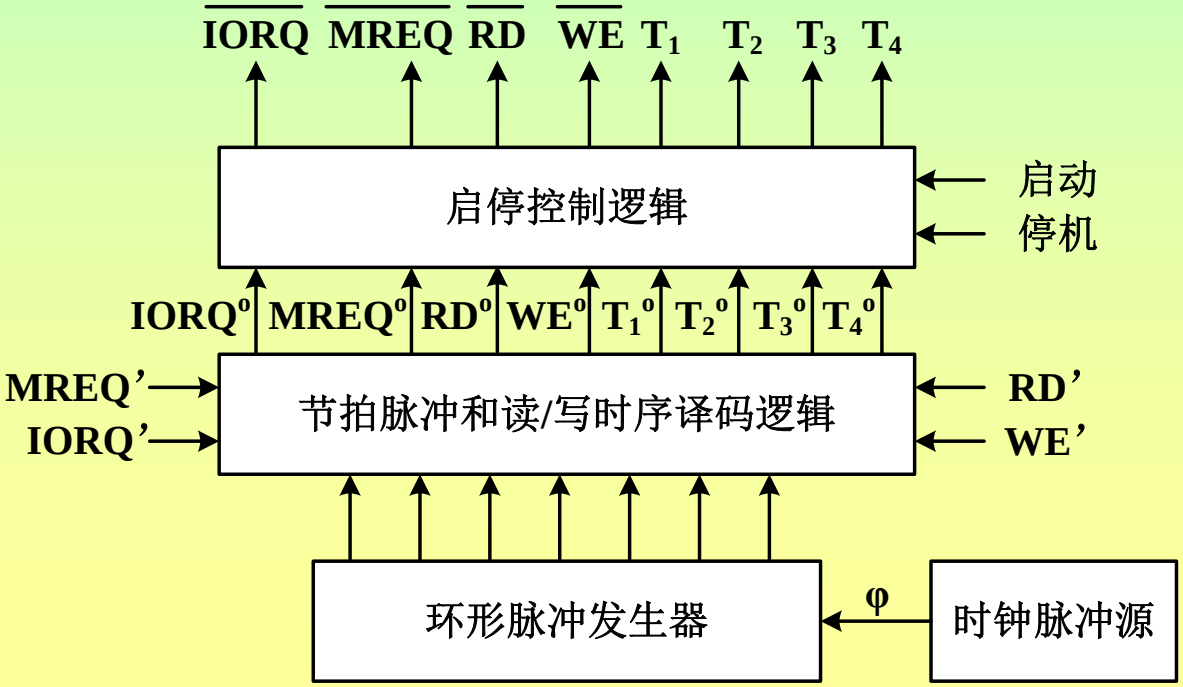


时序信号产生器

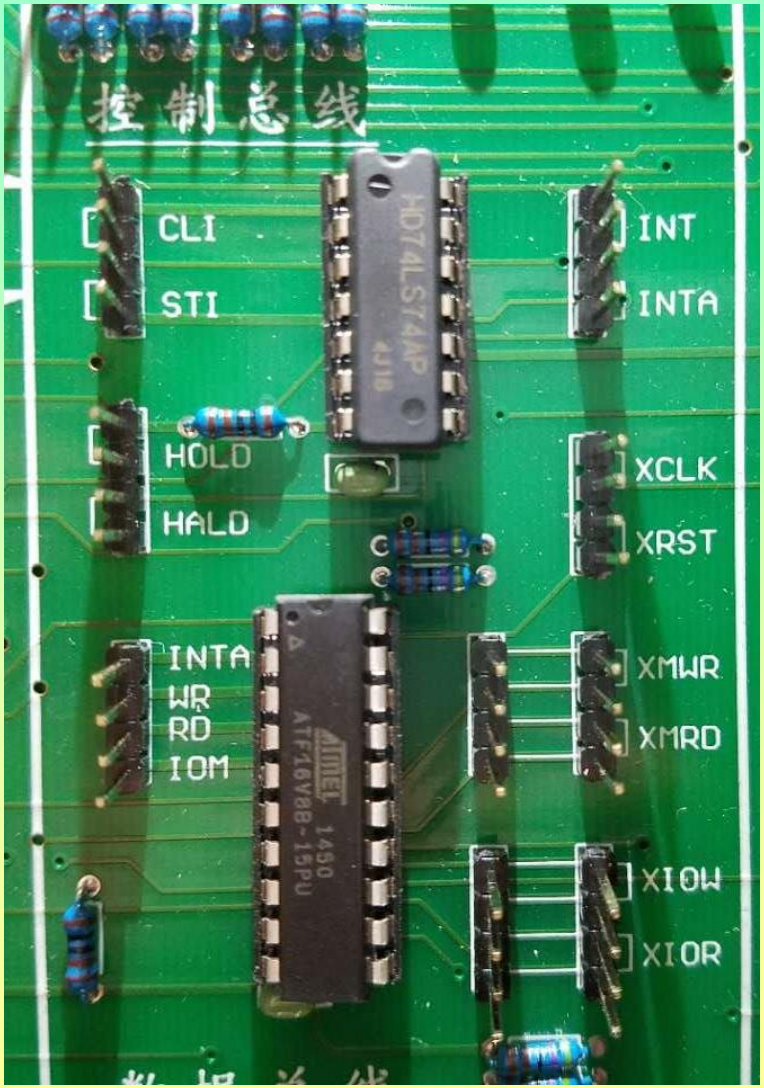
时序信号产生器用来产生具有规定顺序和时间间隔的时序信号，操作控制器利用这些时序信号指挥计算机各部件的协同工作。

硬布线控制器的时序信号采用主状态周期－节拍电位－节拍脉冲三级体制。节拍电位表示一个CPU周期的时间，一个节拍电位包含若干个节拍脉冲，主状态周期包含若干个节拍电位。

微程序控制器一般采用节拍电位－节拍脉冲二级体制。



曾经见过……


控制总线
CLI
STI
HOLD
HALD
INTA
WR
RD
IOM
INT
INTA
XCLK
XRST
XMWR
XMRD
XIOW
XIOR
HD74LS74AP
ATMEL
ATF16V8B-15PU

时序对读写控制的影响

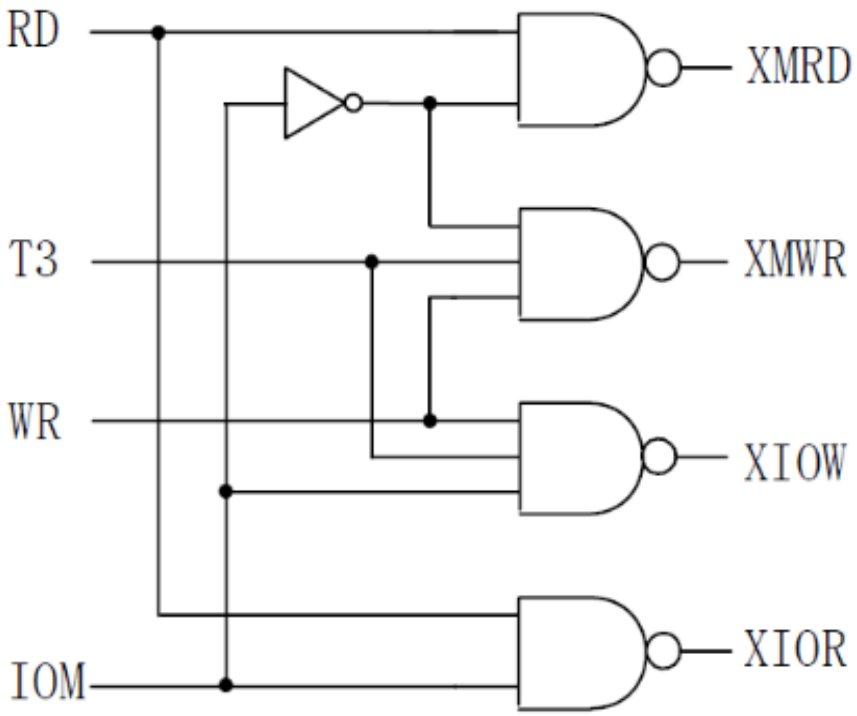
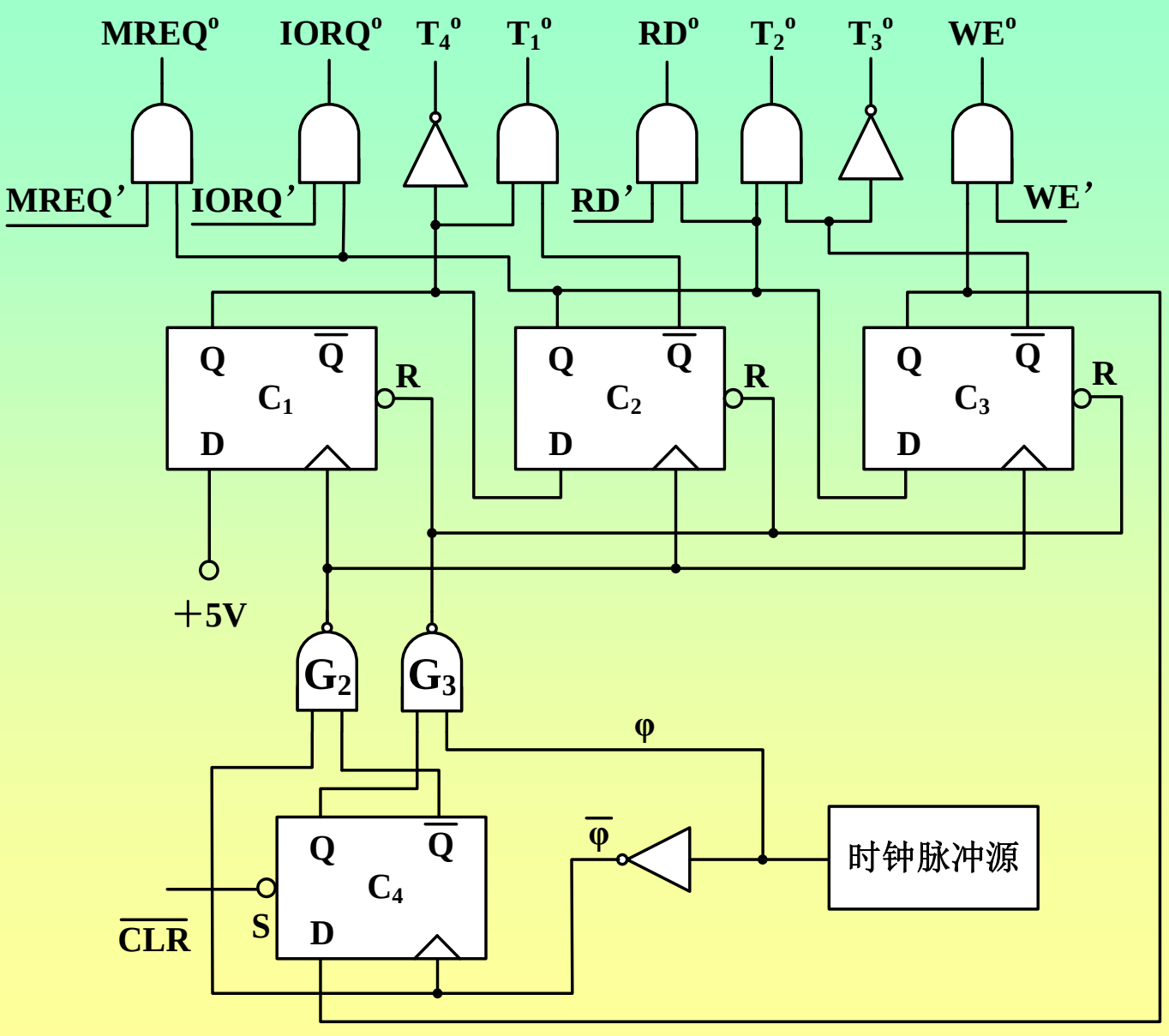


图 2-1-2　读写控制逻辑

曾经见过……


HYDZ
REMOVE
SEAL
AFTER
WASHING
合格
SPK
CLKO
SN74LS390N
时序与操作台
MODE
TS4
TS1
300HZ
30HZ
3HZ
SN74LS245N
编程
校验
运行
主存
控存
置数
减1
加1
KK3
KK4
KK5
S1
KK
单步
连续
TS4 TS3 TS2 TS1
S1- S1+
Q
KK+
KK-
系统总线
主存及

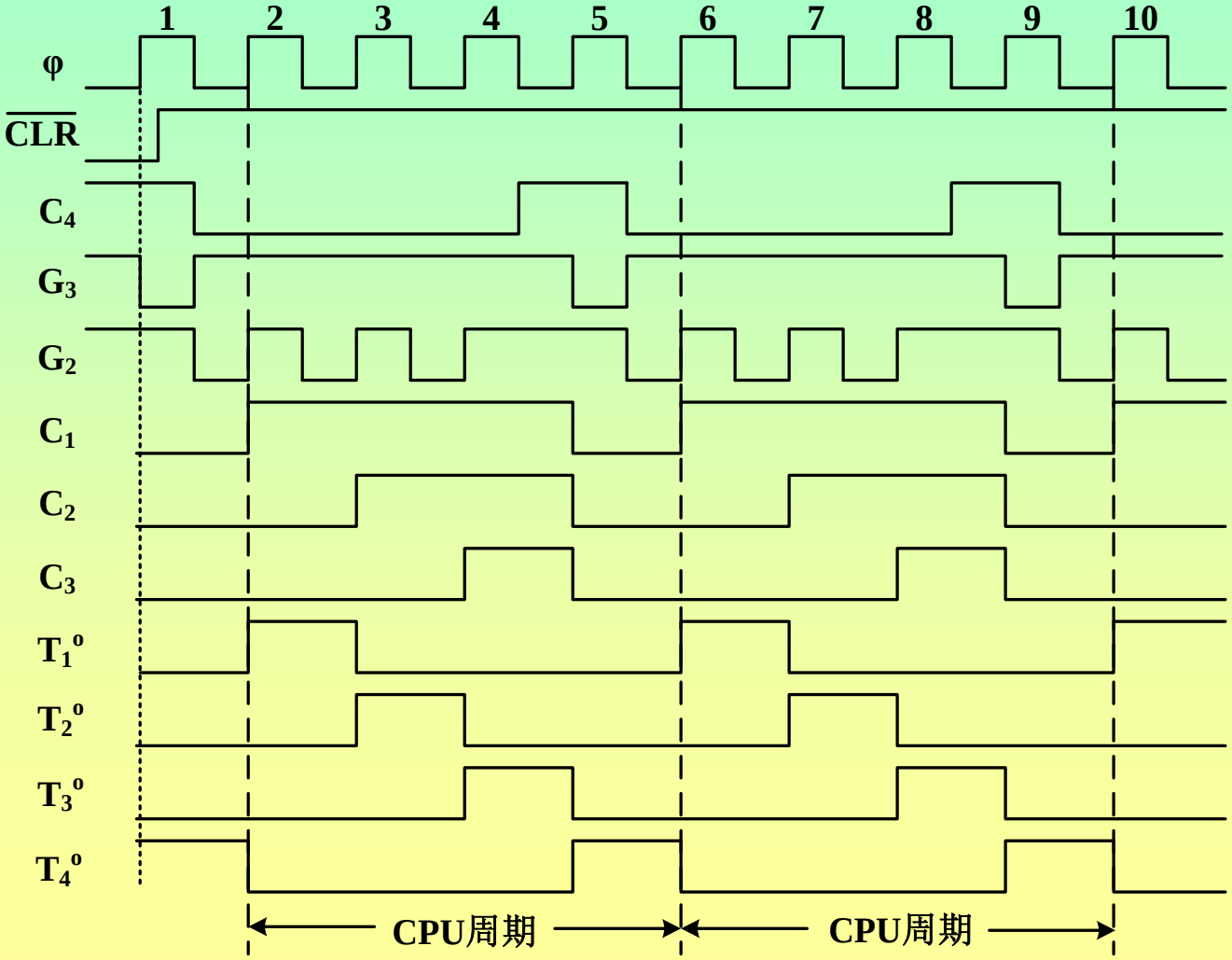
环形脉冲发生器和译码逻辑



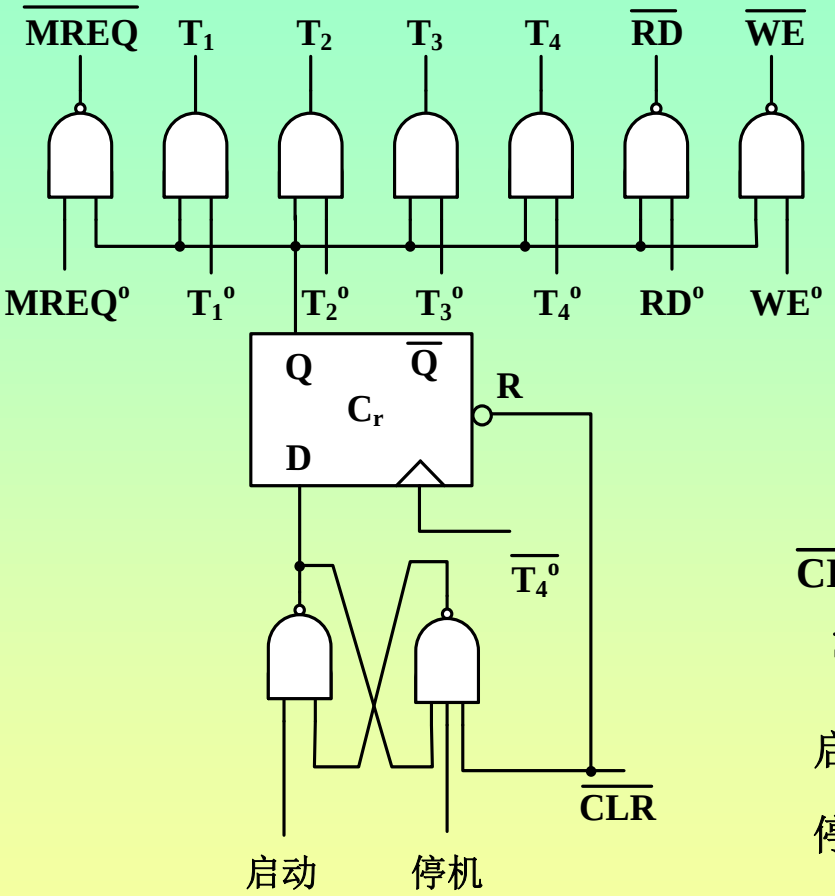
节拍脉冲和读/写时序的译码

$$T_1^o = C_1 \cdot \overline{C_2} \qquad T_2^o = C_2 \cdot \overline{C_3} \qquad T_3^o = C_3 \qquad T_4^o = \overline{C_1}$$

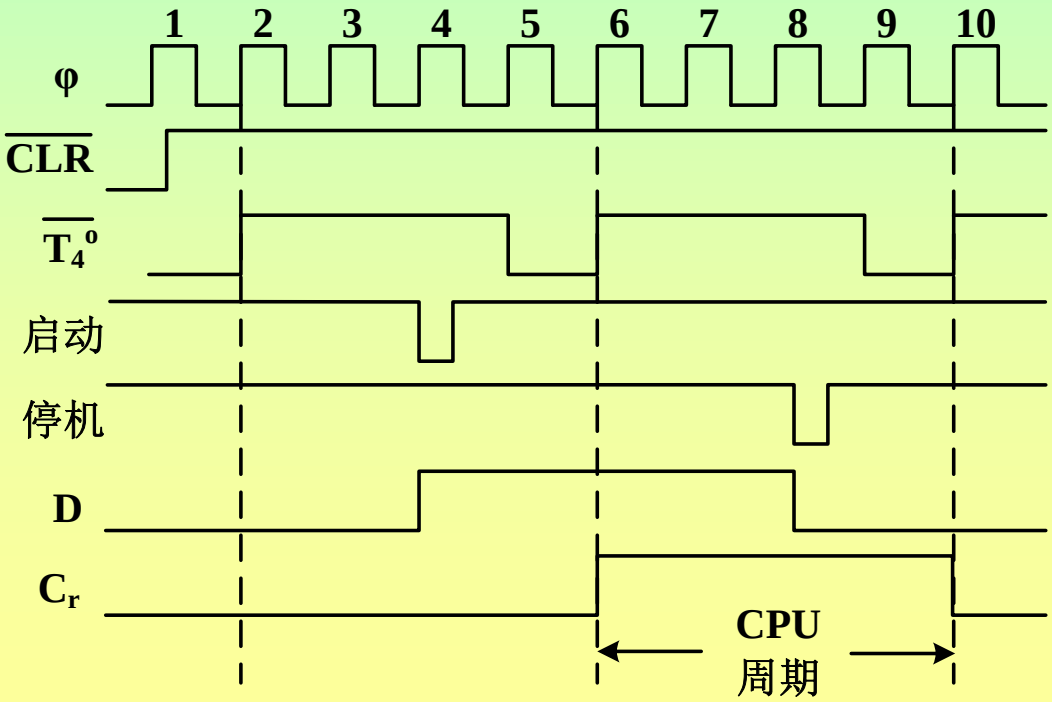
$$RD^o = C_2 \cdot RD' \qquad WE^o = C_3 \cdot WE' \qquad MREQ^o = C_2 \cdot MREQ' \qquad IORQ^o = C_2 \cdot IORQ'$$



启停控制逻辑



计算机的启动和停机都是随机的。当计算机启动时，必须从第一个节拍脉冲前沿开始工作；停机时必须在第四个节拍脉冲结束后关闭时序产生器。只有这样，才能使每个CPU周期都是完整的。



控制方式

一条指令的执行由许多微操作组成。控制器的控制方式，是指形成微操作序列时序的方式。

1. **同步控制方式。**系统有一个统一的时钟，所有控制信号都来自这个统一的时钟。选取最长的操作时间作为统一的时间间隔标准。所有部件都在这个时间间隔内启动并完成操作。

根据指令周期和机器周期的长度是否固定，同步控制方式分为以下几种：

(1) 定长指令周期。所有指令的执行时间都相同。指令的繁简相差很大，规定统一的指令周期，会造成很多的时间浪费。

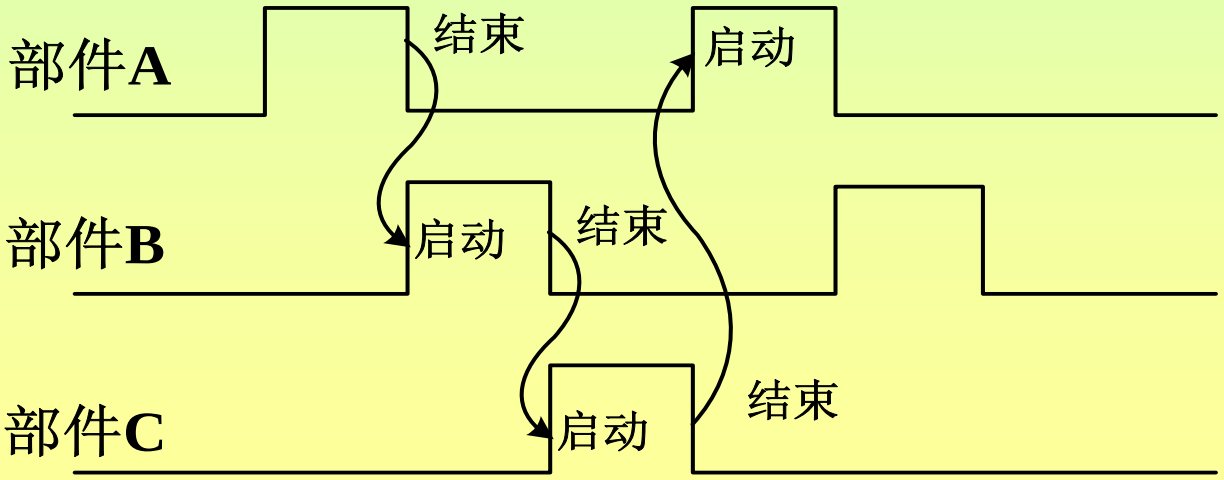
(2) 定长机器周期。机器周期长度固定，一般等于主存的存取周期，指令周期长度不固定，等于机器周期的整数倍。

(3) 变长机器周期、定长节拍脉冲。指令周期和机器周期长度都不固定，每个机器周期含有的节拍脉冲数根据需要而定。不会造成时间的浪费。

CPU内部操作均采用同步控制。

2. 异步控制方式。各项操作不采用统一的时钟信号控制，而是根据指令或部件的具体情况决定，需要多少时间，就占用多少时间。

这是一种“应答”式控制方式。各操作之间的衔接由“结束—起始”信号实现。由前一项操作的“结束”信号或下一项操作的“准备好”信号作为下一项操作的起始信号。在未收到“结束”或“准备好”信号之前不开始新的操作。例如读存储器时，CPU向存储器发出读命令（起始信号），启动存储器的读操作，此时CPU处于等待状态。存储器操作结束后，向CPU发出结束信号，作为下一项操作的起始信号。



3. 联合控制方式。现代计算机中几乎没有完全采用同步或异步控制方式，都是采用联合控制方式。其思想是在功能部件内部采用同步方式或以同步为主的控制方式，在功能部件之间采用异步方式。

联合控制方式中，CPU内部采用同步方式，CPU与主存或其它I/O设备交换数据时，转入异步方式。CPU给出起始信号，主存和I/O设备按自己的时序安排操作；操作结束后向CPU发出结束信号。

CPU并不是随时对来自主存或I/O的信号作出反应，而是在一个节拍脉冲的结束。例如PC/XT机的基本机器周期包括4个节拍脉冲T_1、T_2、T_3和T_4，在T_3和T_4之间可插入若干个等待状态T_w，直到主存或I/O设备的结束信号到达为止。

联合控制方式是CPU进行主存读写或I/O数据传送时通常采用的方式，较好地解决了同步和异步的衔接问题。

习　　题

P186

4. 10.

微程序控制器

微程序是利用软件方法设计硬件的技术。微程序控制的基本思想，就是把操作控制信号编成所谓的“微指令”，存放到一个只读的控制存储器里。机器运行时，一条一条地读出这些微指令，从而产生机器所需要的各种操作控制信号，使相应部件执行规定的操作。

微程序设计的概念和原理是由M. V. Wilkes教授提出来的：一条机器指令可以分解为许多基本的微命令序列。并且首先把这种思想用于计算机控制器的设计。但是由于当时还不具备制造专门存放微程序的控制存储器的技术，所以在十几年时间内实际上并未真正使用。直到1964年，IBM公司在IBM360系列机上成功地采用了微程序设计技术，解决了指令系统的兼容问题。20世纪70年代以来，由于VLSI技术的发展，推动了微程序设计技术的发展和应用，目前，大多数计算机都采用微程序设计技术。

微命令和微操作

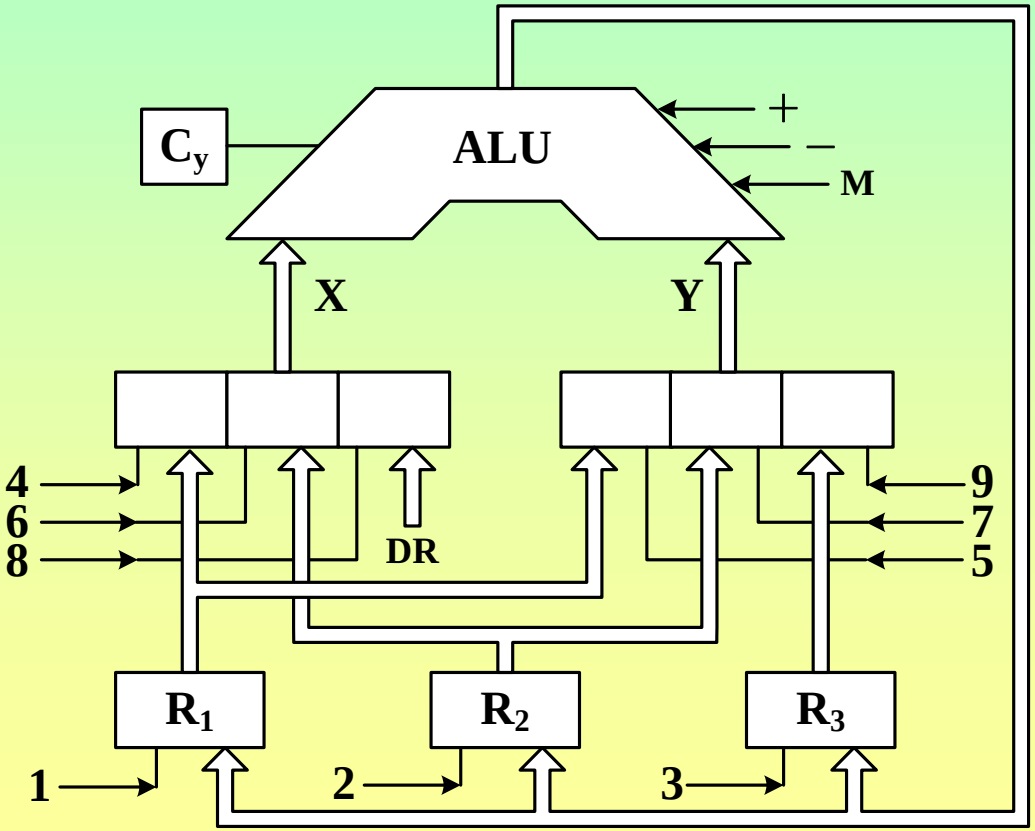
控制部件向执行部件发出的各种控制命令，称为微命令。执行部件接受微命令后所进行的操作，称为微操作。

一条机器指令可以分解成一个微操作序列，这些微操作是计算机中最基本的、不可再分解的操作。微命令和微操作是一一对应的，它是构成控制序列的最小单位。例如打开或关闭某个控制门的电位信号、某个寄存器的时钟脉冲等。

微操作有相容性和相斥性之分。相容性微操作可以同时或在同一个CPU周期内并行执行。相斥性微操作不能同时或不能在同一个CPU周期内并行执行。一个微操作可以和一些微操作相容，和另一些微操作相斥。对于单独一个微操作，谈论其相容和相斥是没有意义的。

下图是一个运算器模型。ALU是算术逻辑单元，R_1～R_3是三个寄存器。1号～3号微命令作为三个寄存器的时钟，以便在ALU运算完毕而输出公共总线上电平稳定时，将结果打入到某一个寄存器。4号～9号微命令作为6个控制门的控制信号，可以将三个寄存器和数据缓冲寄存器DR的内容分别经过X输入端或Y输入端送到ALU。ALU有＋、－、M（传送）三种操作。C_y为进位触发器。

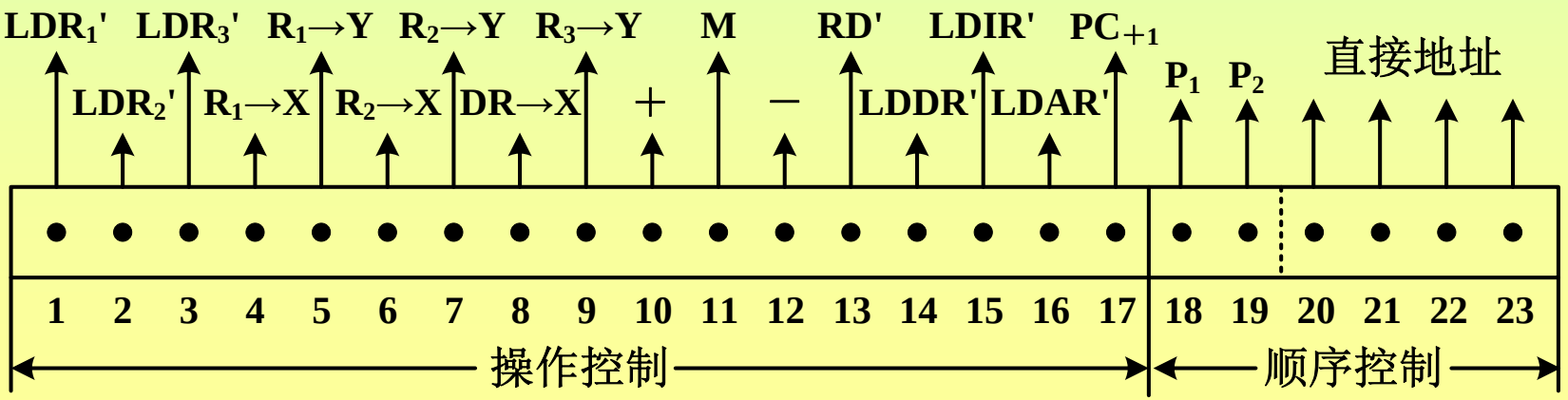
＋、－、M三个微操作是相斥的，4、6、8是相斥的，5、7、9是相斥的。1、2、3是相容的。4、6、8中的一个和5、7、9中的一个是相容的。



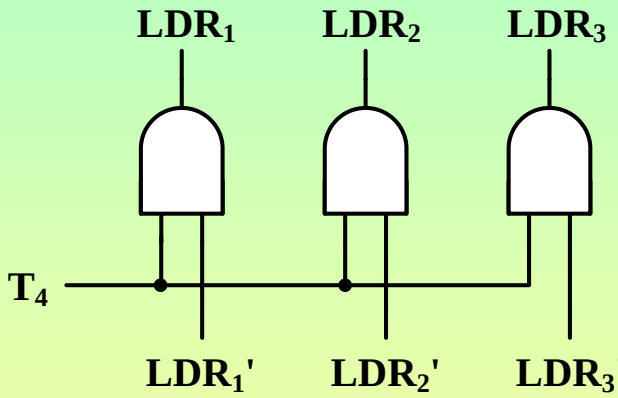
微指令和微程序

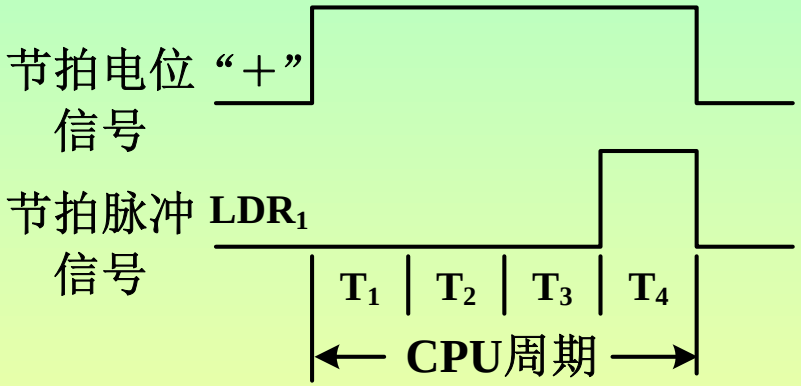
一个CPU周期中，一组实现一定操作功能的微命令的组合，称为微指令。

下图是一个具体的微指令结构，字长23位，包括操作控制和顺序控制两部分。操作控制部分用来发出管理和指挥全机工作的控制信号，共17位，每一位表示一个微命令。某一位为1时，表示发出微命令；为0表示不发出微命令。例如，微指令第1位为1时，发出LDR_1'微命令，运算器执行ALU→R_1的微操作，把公共总线上的信息送入寄存器R_1。微指令第10位为1时，向ALU发出“＋”的微命令，ALU执行“＋”的微操作。



微指令给出的控制信号都是节拍电位信号，持续时间都是一个CPU周期。某些微命令还应加入时间控制。例如前三个微命令应该和节拍脉冲T_4相与，得到LDR_1～LDR_3，送给运算器的其它9个微命令都是节拍电位信号。这样，保证运算器在前3个节拍脉冲内进行运算，第4个节拍脉冲把运算结果打入相应的寄存器。



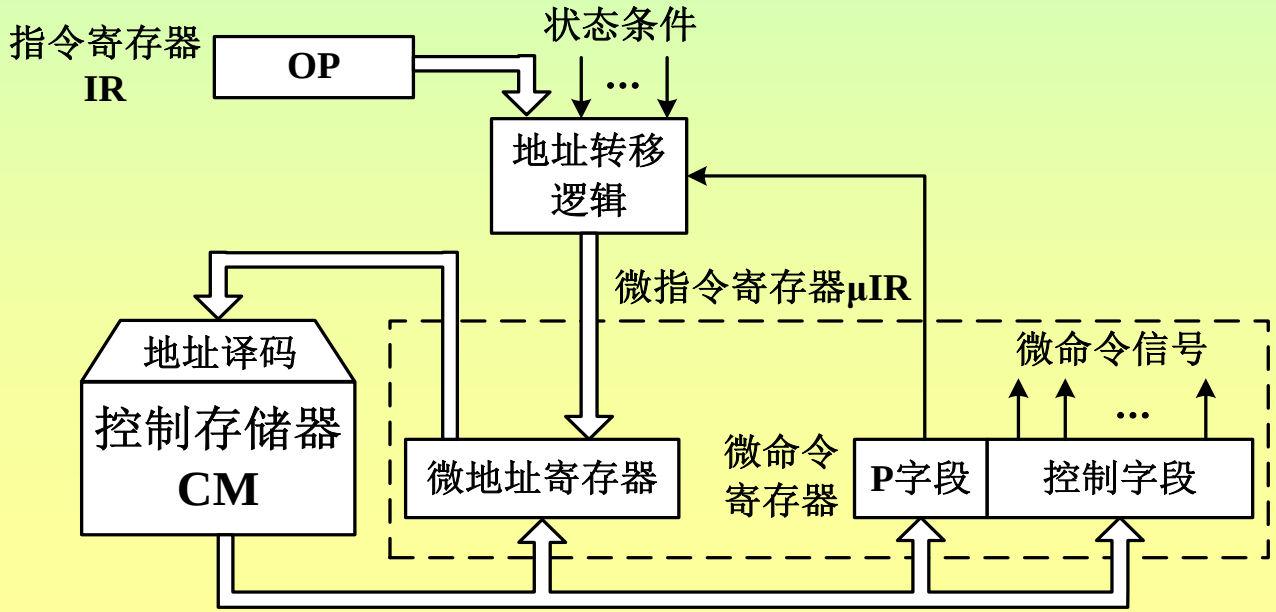


一条机器指令的功能由许多条微指令组成的序列来实现，这个微指令序列称为微程序。执行当前微指令时，必须指出后继微指令的地址。微指令格式中的顺序控制部分就是用来产生后继微指令的地址。P_1和P_2为0时，直接地址就是下一条微指令的地址；P_1或P_2为1时，要进行判别测试，根据测试结果对直接地址进行修改。

微程序控制器

微程序控制器由控制存储器、微指令寄存器、地址转移逻辑三部分组成，其中微指令寄存器包括微地址寄存器和微命令寄存器。

1. **控制存储器。**用来存放实现全部指令系统的微程序，是一种只读存储器。读出一条微指令并执行微指令的时间总和称为一个微指令周期。在串行方式的微程序控制器中，微指令周期就是只读存储器的工作周期。控制存储器的字长就是微指令字的长度。



2. **微指令寄存器。**用来存放由控制存储器读出的一条微指令信息。其中微地址寄存器决定将要访问的下一条微指令的地址，而微命令寄存器则保存一条微指令的操作控制字段和判别测试字段的信息。

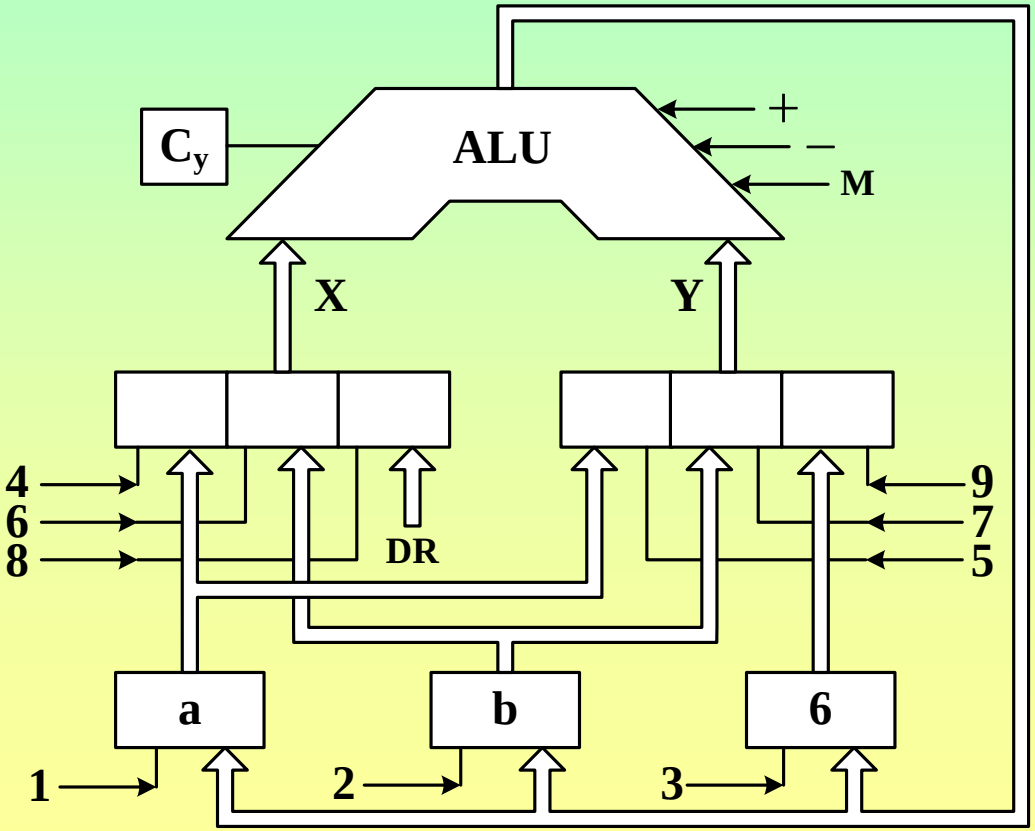
3. **地址转移逻辑。**一般情况下，微指令由控制存储器读出后直接给出下一条微指令的地址，简称微地址，这个微地址信息就存放在微地址寄存器中。如果微程序不出现分支，那么下一条微指令的地址就直接由微地址寄存器给出。当微程序出现分支时，意味着微程序出现条件转移。在这种情况下，通过判别测试字段P和执行部件的“状态条件”反馈信息，去修改微地址寄存器的内容，并按改好的内容去读下一条微指令。地址转移逻辑就承担自动完成修改微地址的任务。

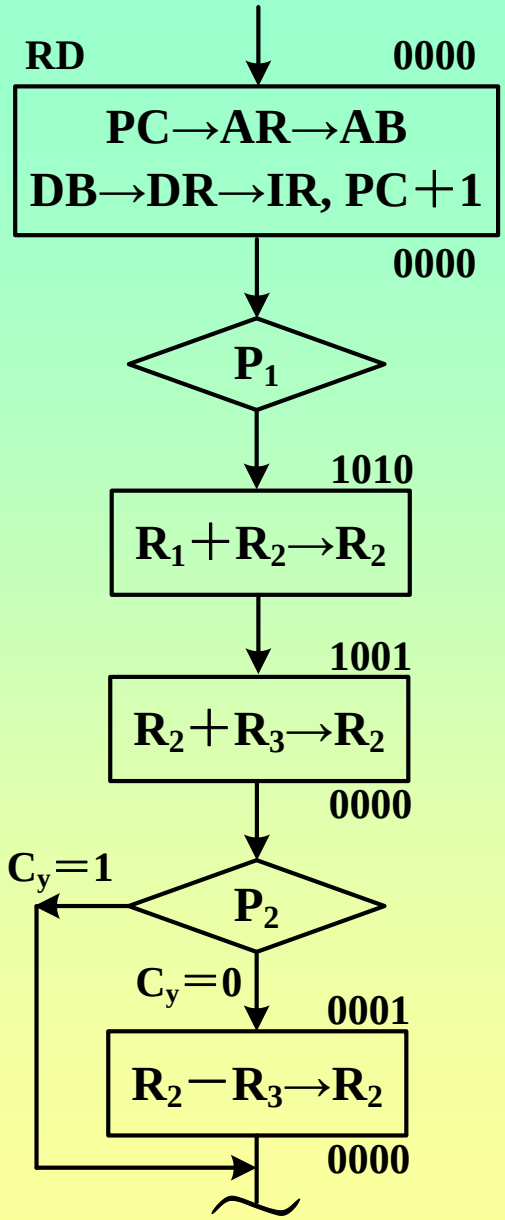
微程序举例

“十进制加法”指令的微程序。

“十进制加法”指令的功能是用BCD码进行十进制加法。两数相加的结果小于等于9时，结果是正确的；结果大于9时，要进行加6修正。

假定数a和b已经存放在寄存器R_1和R_2中，数6存放在R_3中。完成十进制加法的过程是：先计算a＋b＋6，然后判断有无进位。当进位标志C_y＝1时，不减6；当C_y＝0时，减去6。





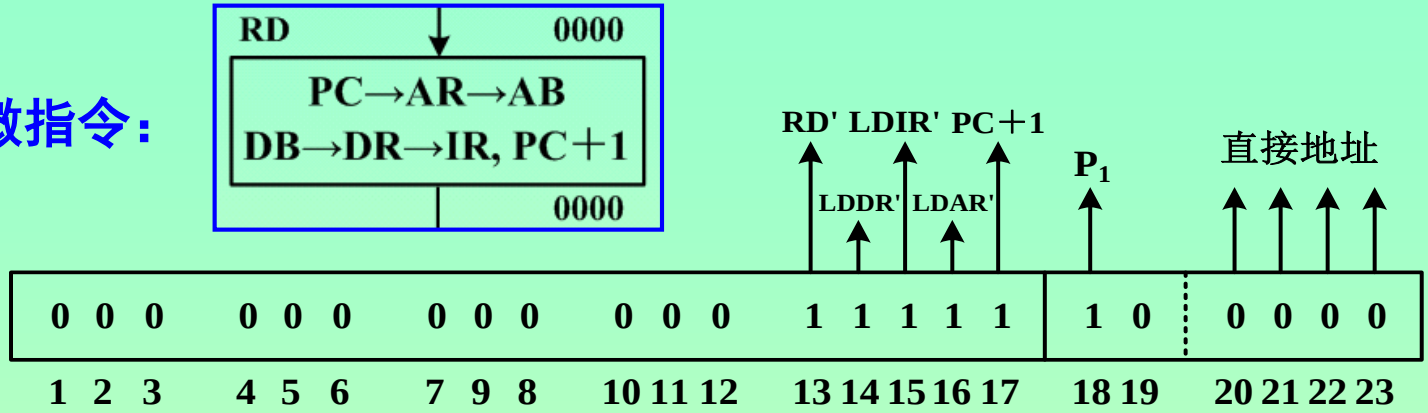
该微程序包括4条微指令，每条微指令用一个矩形框表示。

第一条微指令是“取指”微指令，其任务是：从主存中取出机器指令放入IR；PC加1；对机器指令的操作码用P_1进行判别测试，然后修改微地址寄存器内容，给出下一条微指令地址。

第二条微指令完成a＋b运算，第三条微指令完成a＋b＋6，同时进行判别测试。P_2用来测试进位标志C_y。根据测试结果，微程序或者转向公操作，或者转向第四条微指令进行减6运算。

矩形框右上角表示该微指令的地址，右下角表示该微指令内包含的直接地址。菱形符号代表判别测试，它的动作依附于上一条微指令。

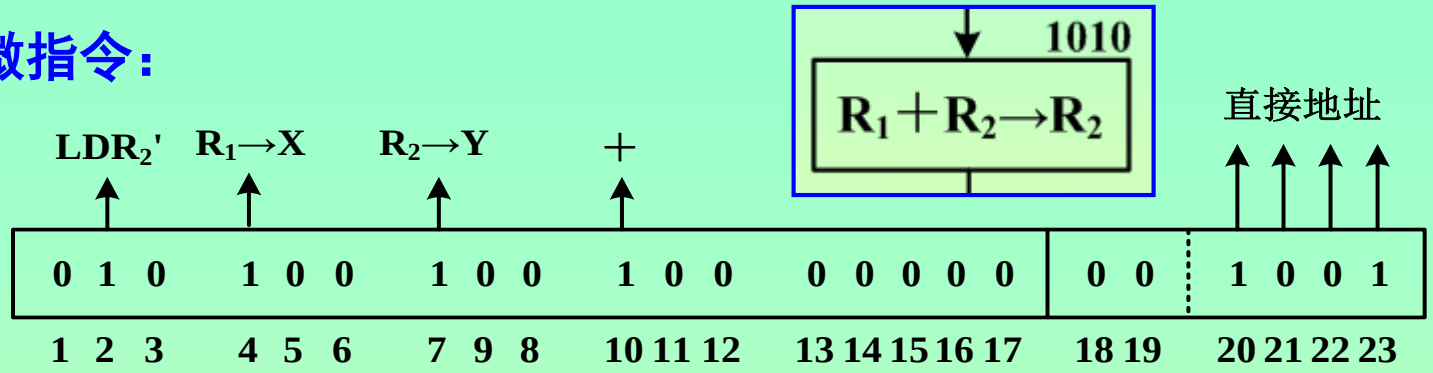
第一条微指令：



操作控制字段包括5个微命令。第16位LDAR’将PC内容送到地址寄存器AR；第13位RD’发出主存读命令；第14位LDDR’将主存中读出的“十进制加法”机器指令送数据缓冲器DR；第15位LDIR’将DR中内容送指令寄存器IR。假定“十进制加法”指令的操作码是1010，则IR的OP字段现在是1010。这四个微命令都应加入时间控制，和节拍脉冲相与。第17位将PC加1。

微指令的顺序控制字段指明下一条微指令的地址是0000，但第18位为1，表明进行P_1测试。P_1测试的状态条件是IR中的操作码字段，即用OP字段作为下一条微指令的地址，于是微地址寄存器的内容修改成1010。

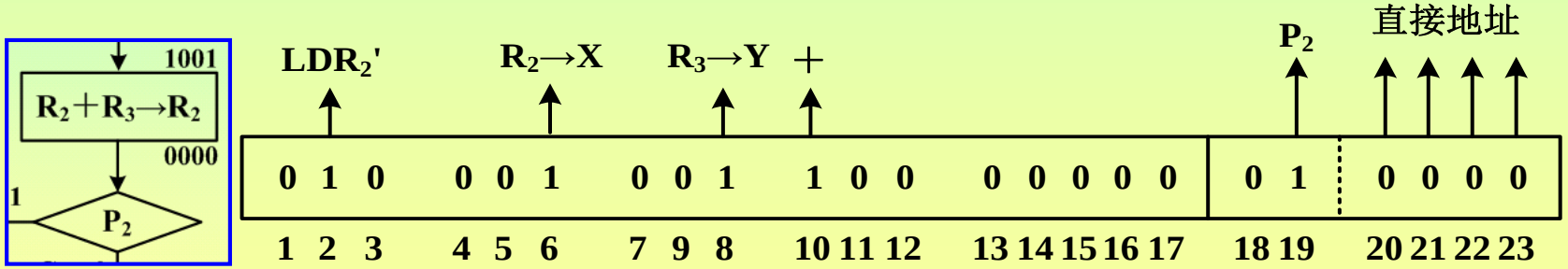
第二条微指令：



操作控制字段包括4个微命令。$R_1 \rightarrow X$， $R_2 \rightarrow Y$，＋，LDR_2'，运算器完成$R_1 + R_2 \rightarrow R_2$操作。

P_1和P_2均为0，表示不进行判别测试，直接给出下一条微指令地址1001。

第三条微指令：

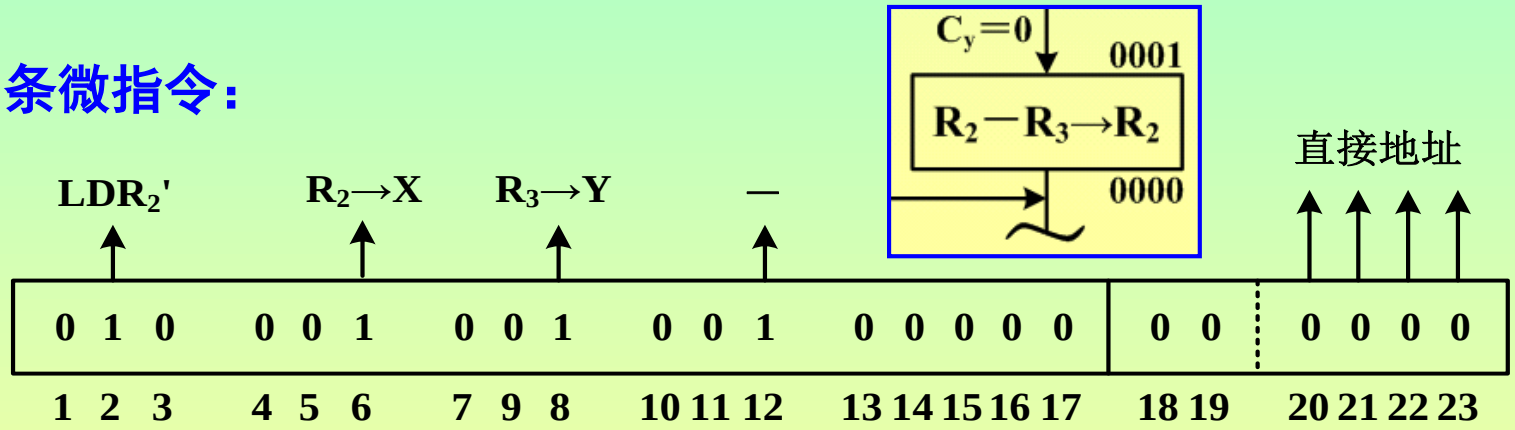


操作控制字段包括4个微命令。$R_2 \rightarrow X$， $R_3 \rightarrow Y$，＋，LDR_2'，运算器完成$R_2 + R_3 \rightarrow R_2$操作。

第19位为1，表明进行P_2测试。P_2测试的状态条件是进位标志C_y。假定用C_y修改微地址寄存器的最后一位：当C_y＝0时，下一条微指令地址为0001；当C_y＝1时，下一条微指令地址为0000，转向“取指”指令。

假定C_y＝0，按照地址0001读出第四条微指令。

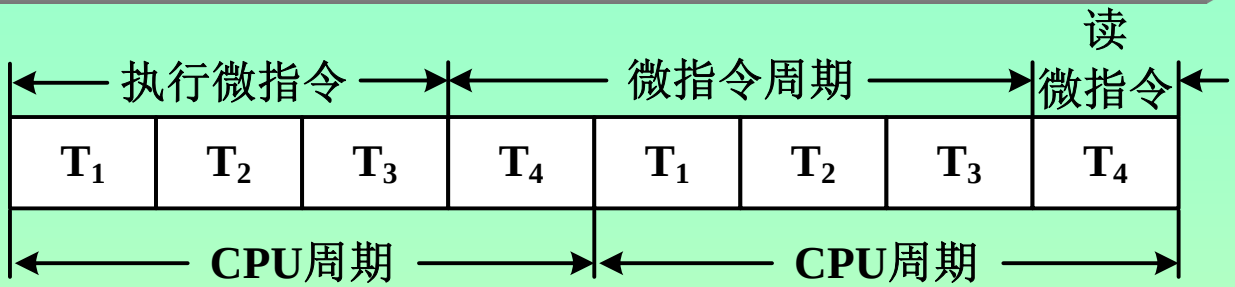
第四条微指令：



操作控制字段包括4个微命令。$R_2 \rightarrow X$， $R_3 \rightarrow Y$，－，LDR_2’，运算器完成$R_2 - R_3 \rightarrow R_2$操作。

顺序控制部分直接给出下一条微指令地址0000，即“取指”指令。

CPU周期与微指令周期的关系



串行方式的微程序控制器中，微指令周期等于读出微指令的时间加上执行该微指令的时间。为了使机器的控制信号同步，可以将一个微指令周期设计得恰好和CPU周期相等。

每个微指令周期包括上一个CPU周期的T_4（用于读微指令）和下一个CPU周期的T_1、T_2、T_3（用于执行微指令）。例如，在T_1～T_3时间内运算器进行运算，T_3的末尾运算器已经运算完毕，利用T_4上升沿将运算结果打入寄存器。与此同时利用T_4的时间取下一条微指令，并利用T_1的上升沿打入微指令寄存器。一条微指令的微命令信号从T_1上升沿开始有效，到下一条微指令打入微指令寄存器为止，其保持时间恰好为一个CPU周期。

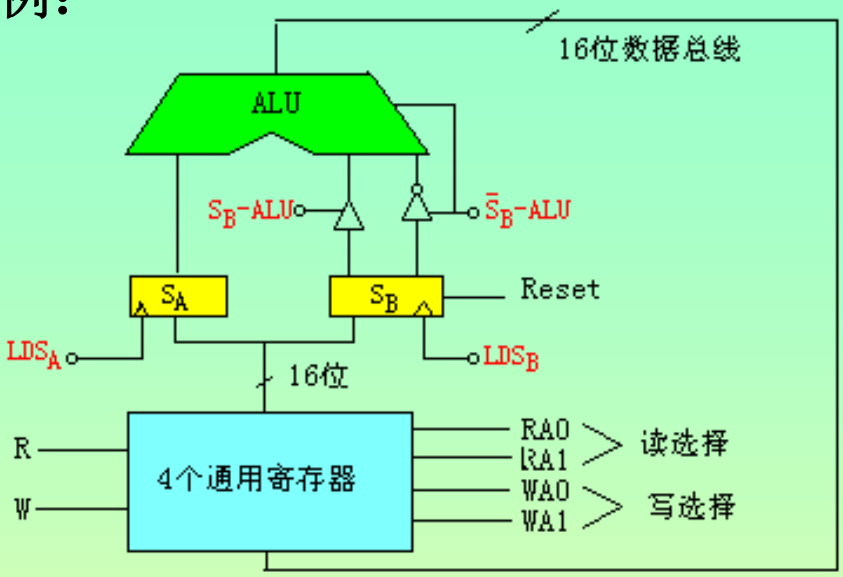
机器指令与微指令的关系

1、一条机器指令对应一个微程序；一条机器指令所完成的操作划分为若干条微指令完成；

2、指令存放在存储器中；微指令存放在控制存储器中

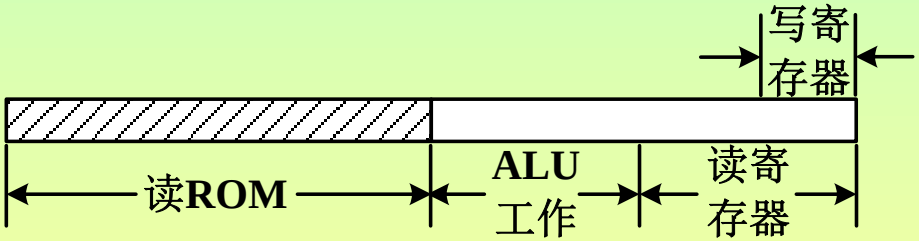
3、一个CPU周期对应一条微指令，一个指令周期至少包含两个CPU周期

例：

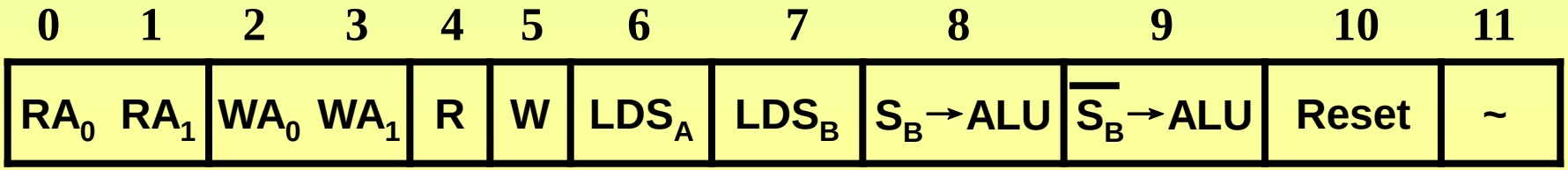


读控制				写控制			
R	RA_0	RA_1	选择	W	WA_0	WA_1	选择
1	0	0	R_0	1	0	0	R_0
1	0	1	R_1	1	0	1	R_1
1	1	0	R_2	1	1	0	R_2
1	1	1	R_3	1	1	1	R_3
0	×	×	不读出	0	×	×	不写入

微指令周期：



微指令格式：

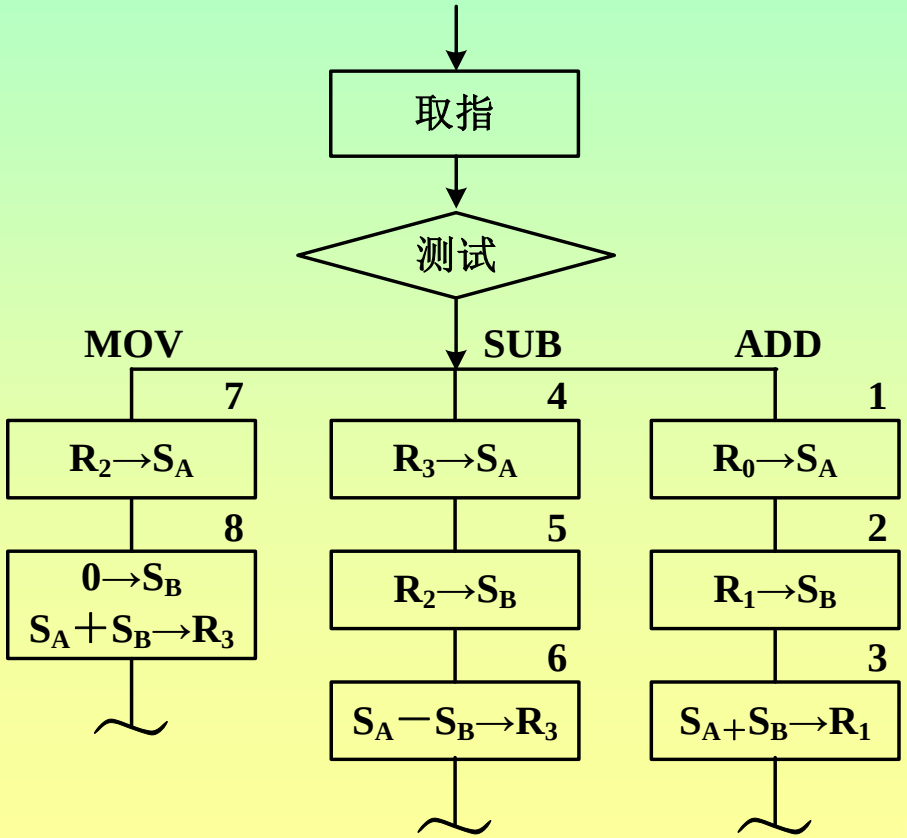


写出下列指令的微程序：

（1）ADD R_0, R_1指令，即$(R_0)+(R_1)\rightarrow R_1$

（2）SUB R_2, R_3指令，即$(R_3)-(R_2)\rightarrow R_3$

（3）MOV R_2, R_3指令，即$(R_2)\rightarrow R_3$



指令	微程序代码
ADD	1. 00××10100000 2. 01××10010000 3. ××0101001001
SUB	4. 11××10100000 5. 10××10010000 6. ××1101000101
MOV	7. 10××10100000 8. ××1101001011

0	1	2	3	4	5	6	7	8	9	10	11
RA_0	RA_1	WA_0	WA_1	R	W	LDS_A	LDS_B	$S_B\rightarrow ALU$	$\overline{S_B}\rightarrow ALU$	Reset	~

微程序设计技术

如何确定微指令结构是微程序设计的关键。

设计微程序结构追求的目标：

1、有利于缩短微指令的长度

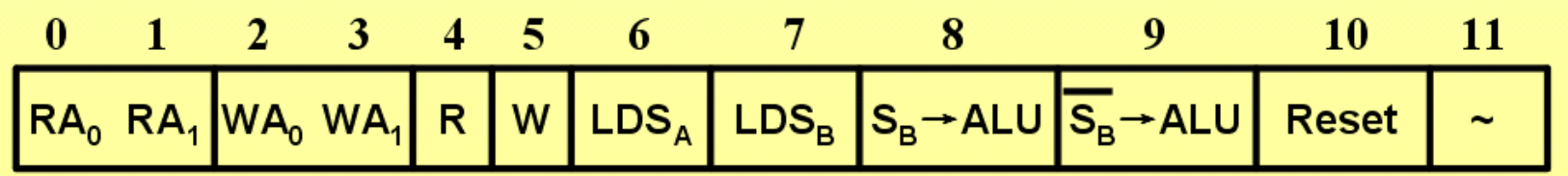
2、有利于减小控制存储器的容量

3、有利于提高微程序的执行速度

4、有利于对微指令的修改

5、有利于提高微程序设计的灵活性

P178 例2中的微指令结构：

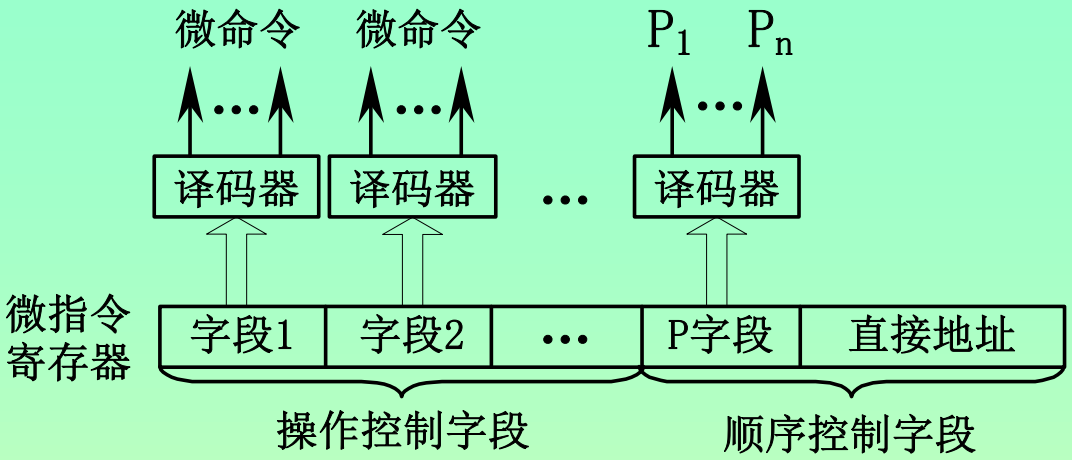


微命令编码

微命令编码就是对微指令中的操作控制字段采用的表示方法。

1. **直接表示法。**操作控制字段中的每一位代表一个微命令。这种方法简单直观，其输出直接用于控制，操作速度快，但是微指令字太长。若微命令的总数为N个，则微指令字的操作控制字段就要有N位。在某些计算机中，微命令的总数多达几百个，这使微指令的长度达到难以接受的地步。另外，在N个微命令中，有许多是互斥的，不允许并行操作，将它们安排在一条微指令中是毫无意义的，只会使信息的利用率下降。

2. **编码表示法。**把一组相斥性的微命令信号组成一个小组(即一个字段)，然后通过小组(字段)译码器对每一个微命令信号进行译码，译码输出作为操作控制信号。以时间换取空间。



编码表示法中操作控制字段的分组原则：

① 把相斥性的微命令分在同一组内，相容性的微命令分在不同组内。

② 每个小组中包含的位数不能太多，否则将增加译码时间。

③ 每个小组还要留出一个状态，表示小组不发出任何微命令。因此当某小组的长度为三位时，最多只能表示七个互斥的微命令。

3. 混合表示法。把直接表示法与字段编码法混合使用，以便能综合考虑指令字长、灵活性、执行微程序速度等方面的要求。

另外，在微指令中还可附设一个常数字段。该常数可作为操作数送入ALU运算，也可作为计数器初值用来控制微程序循环次数。

微地址的形成方式

微指令执行的顺序控制问题，实际上是如何确定下一条微指令地址的问题。通常，产生后继微地址有两种方法：

1. **计数器方式。**这种方法和用程序计数器来产生机器指令地址的方法类似，需要在微程序控制器中设置一个微程序计数器μPC，取代微地址寄存器。在顺序执行微指令时，后继微地址由现行微地址加上一个增量来产生，为此，顺序执行的微指令序列就必须安排在控制存储器的连续单元中；在非顺序执行微指令时，由转移微指令实现转移。转移微指令的顺序控制字段包括两部分：转移控制字段和转移地址字段。当转移条件满足时，将转移地址字段作为下一个微地址送μPC；否则直接从μPC中取出下一个微地址。

计数器方式的基本特点是：微指令的顺序控制字段较短，微地址产生机构简单。但是多路并行转移功能较弱，速度较慢，灵活性较差。

2. 多路转移方式。一条微指令具有多个转移分支的能力称为多路转移。在多路转移方式中，当微程序不产生分支时，后继微地址直接由微指令的顺序控制字段给出；当微程序出现分支时，有若干“候选”微地址可供选择：即按顺序控制字段的“判别测试”标志和“状态条件”信息来选择其中一个微地址。“状态条件”有n位标志，可实现微程序2^n路转移，涉及微地址寄存器的n位 。

多路转移方式的特点是：能与较短的顺序控制字段配合，实现多路并行转移，灵活性好，速度较快，但转移地址逻辑需要用组合逻辑方法设计。

例：微地址寄存器有6位($\mu A_5 - \mu A_0$)，当需要修改其内容时，可通过某一位触发器的强置端S将其置“1”。现有三种情况：(1) 执行“取指”微指令后，微程序按IR的OP字段($IR_3 - IR_0$)进行16路分支；(2) 执行条件转移指令微程序时，按进位标志C的状态进行2路分支；(3) 执行控制台指令微程序时，按IR_4，IR_5的状态进行4路分支。请按多路转移方法设计微地址转移逻辑。

按所给设计条件，微程序有三种判别测试，分别为P1，P2，P3。 由于修改μA5-μA0内容具有很大灵活性，现分配如下：

(1)用P1和IR3-IR0修改μA3-μA0；

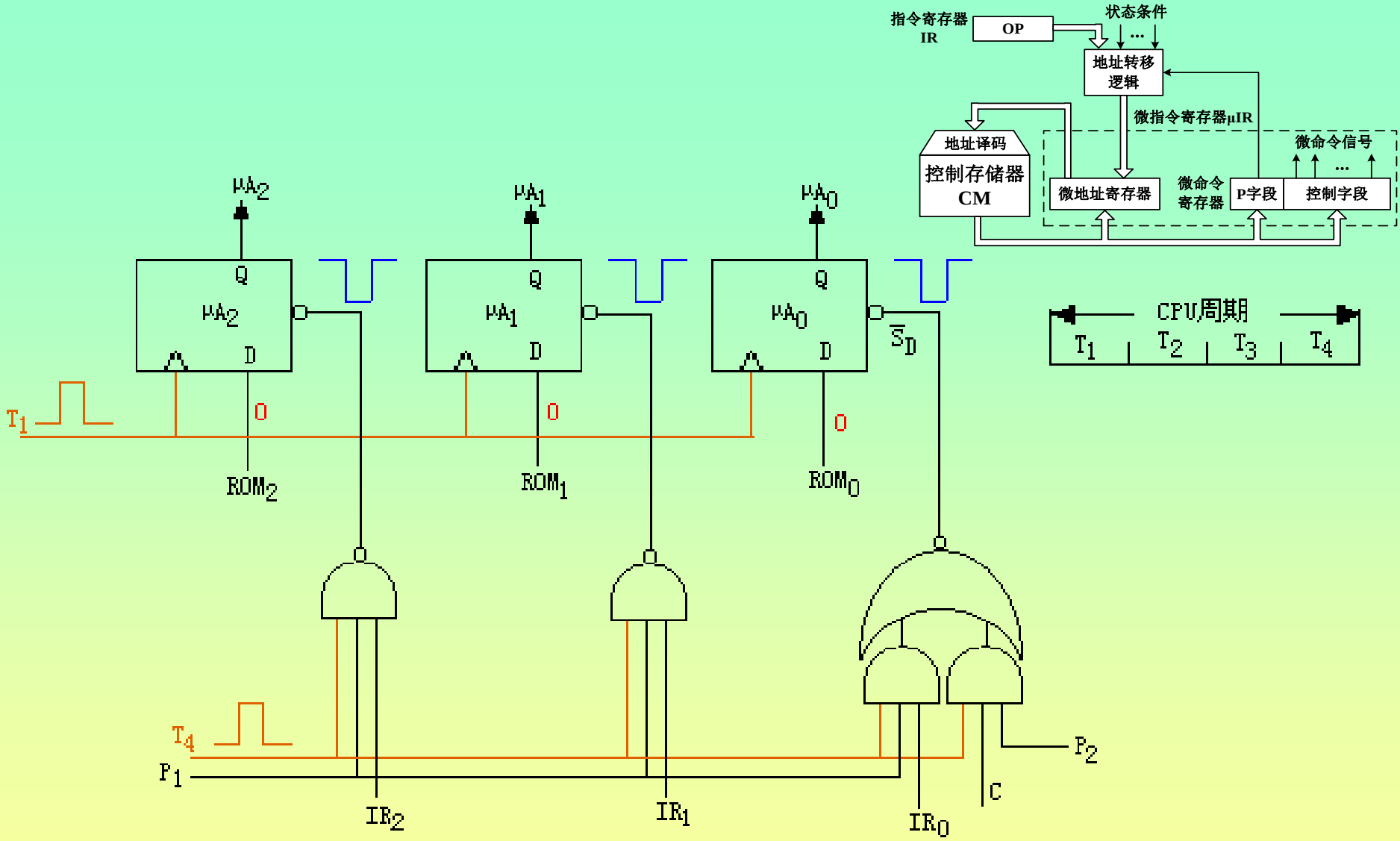
(2)用P2和C修改μA0；

(3)用P3和IR5，IR4修改μA5，μA4。

另外还要考虑时间因素T4(假设CPU周期最后一个节拍脉冲)，故转移逻辑表达式如下：

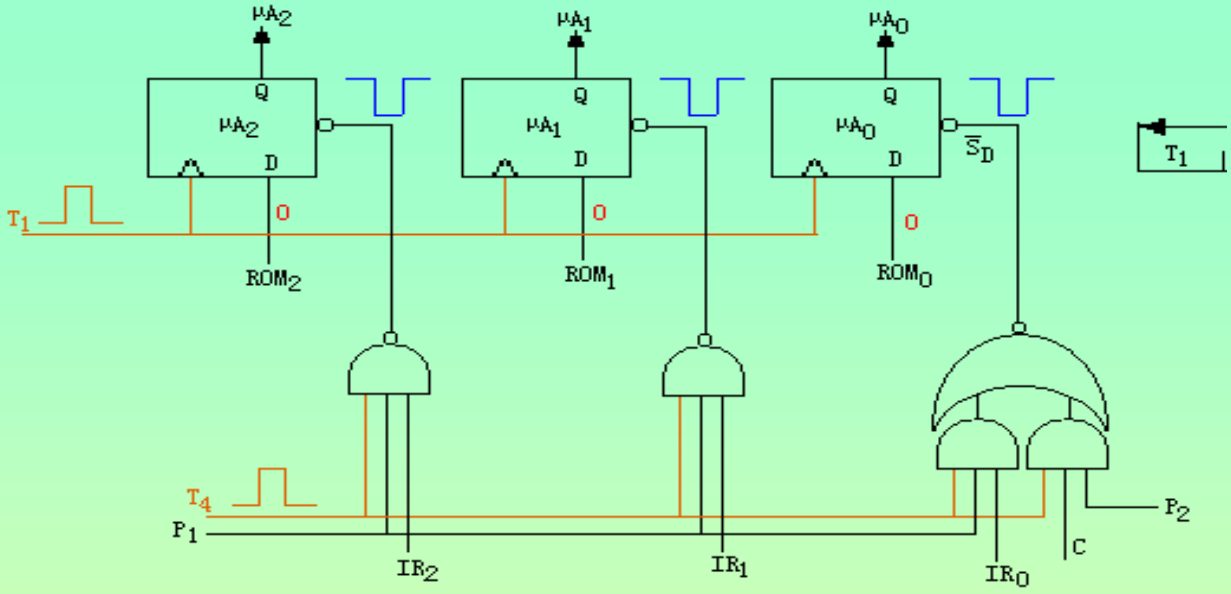
$$\mu A5 = P3 \cdot IR5 \cdot T4$$
$$\mu A4 = P3 \cdot IR4 \cdot T4$$
$$\mu A3 = P1 \cdot IR3 \cdot T4$$
$$\mu A2 = P1 \cdot IR2 \cdot T4$$
$$\mu A1 = P1 \cdot IR1 \cdot T4$$
$$\mu A0 = P1 \cdot IR0 \cdot T4 + P2 \cdot C \cdot T4$$

由于从触发器强置端修改，故前5个表达式可用“与非”门实现，最后一个用“与或非”门实现。
下图仅画出了μA2、μA1、μA0触发器的微地址转移逻辑图。

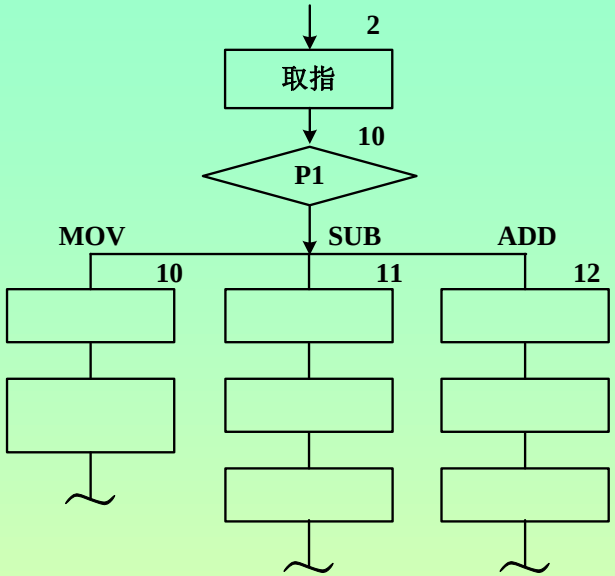


注意：IR0~IR2是指令寄存器的后三位，其中存放的是指令；

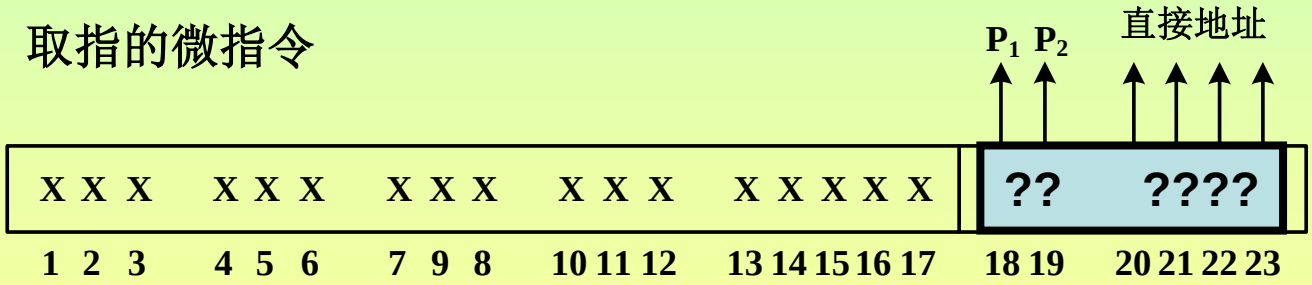
ROM0~ROM2是微地址的后三位，其中存放的是微地址。



地址以8进制方式表示



取指的微指令



当机器指令码为0001时，是否进行P1测试？取指之后，转移到哪条微指令？ SUB

当机器指令码为0010时呢？为0000时呢？

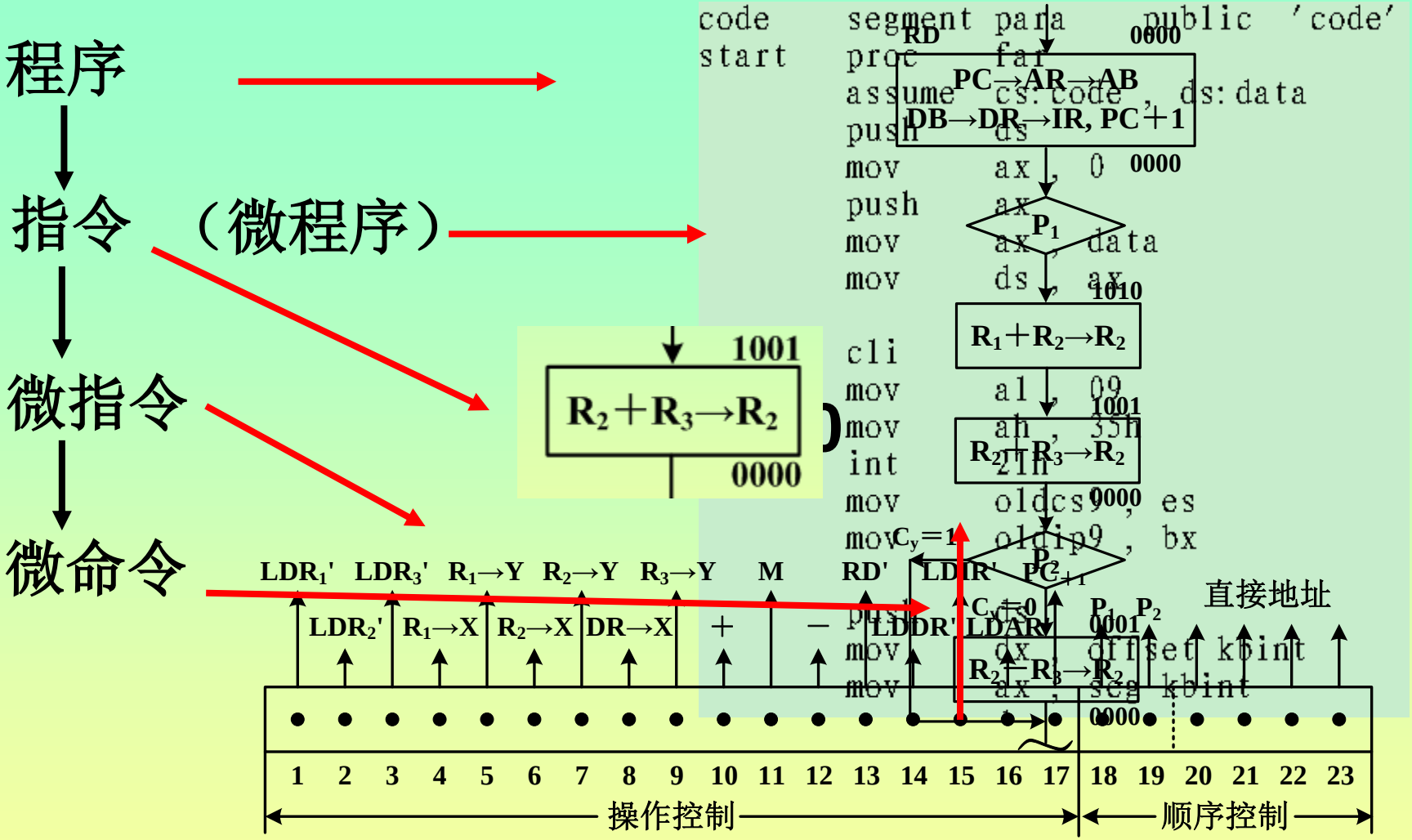
ADD MOV

- 为什么指令的OP字段不同，后续执行的操作不同？

- 取指令后，进行判别测试，跳转不同的分支

请区分以下概念定义

- 微命令
- 微操作
- 微指令
- 微程序
- 程序
- 机器指令
- 控制存储器
- 内存（主存）
- 微指令寄存器
- 指令寄存器
- 微程序计数器
- 程序计数器
- 指令周期
- 机器周期
- 时钟周期


程序
指令
（微程序）
微指令
微命令
code segment para public 'code'
start proc far
assume cs:code, ds:data
push ds
mov ax, 0
push ax
mov ax, data
mov ds, ax
cli
mov al, 09
mov ah, 35h
int 21h
mov oldcs9, es
mov oldip9, bx
RD 0000
PC→AR→AB
DB→DR→IR, PC+1
P1
1010
R1+R2→R2
1001
R2+R3→R2
0000
Cy=1
P2
PC+1
Cy=0
LDAR
LDDR
0001
直接地址
LDR1' LDR3' R1→Y R2→Y R3→Y M RD' LDIR'
LDR2' R1→X R2→X DR→X + −
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
操作控制
顺序控制

微指令格式

微指令的格式大体分成两类：水平型微指令和垂直型微指令。

1. **水平型微指令。**一次能定义并执行多个并行操作微命令的微指令，叫做水平型微指令。 其一般格式如下：

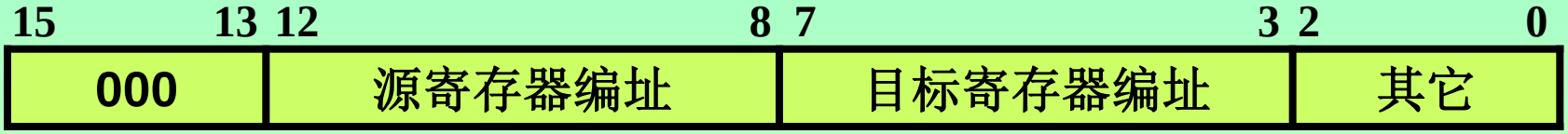
控 制 字 段	判别测试字段	下地址字段

按照控制字段的编码方法不同，水平型微指令又分为三种：全水平型(不译法)微指令，字段译码法水平型微指令，以及直接和译码相混合的水平型微指令。

2. **垂直型微指令。**微指令中设置微操作码字段，由微操作码规定微指令的功能 ，称为垂直型微指令。其结构类似于机器指令的结构。它有操作码，在一条微指令中只有1—2个微操作命令，每条微指令的功能简单，实现一条机器指令的微程序要比水平型微指令编写的微程序长得多。它是采用较长的微程序结构去换取较短的微指令结构。

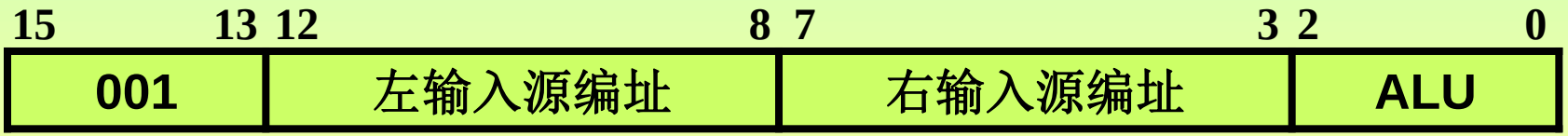
下面举4条垂直型微指令的微指令格式加以说明。设微指令字长为16位，微操作码3位。

(1) 寄存器—寄存器传送型微指令



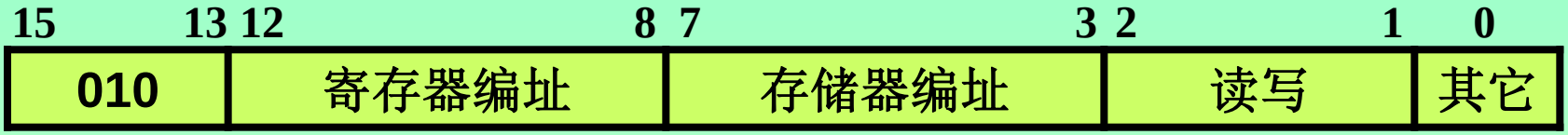
其功能是把源寄存器数据送目标寄存器。13－15位为微操作码(下同)，源寄存器和目标寄存器编址各5位，可指定31个寄存器。

(2) 运算控制型微指令



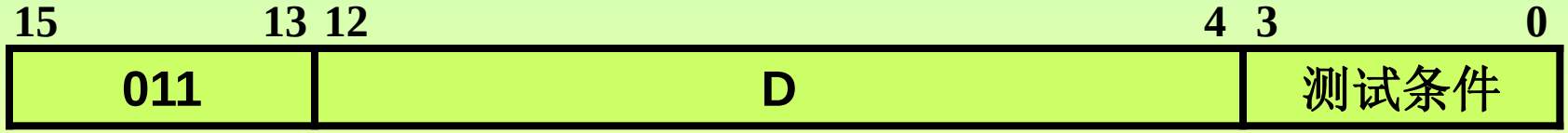
其功能是选择ALU的左、右两输入源信息，按ALU字段所指定的运算功能(8种操作)进行处理，并将结果送入暂存器中。左、右输入源编址可指定31种信息源之一。

（3）访问主存微指令



其功能是将主存中一个单元的信息送入寄存器或者将寄存器的数据送往主存。存储器编址是指按规定的寻址方式进行编址。第1，2位指定读操作或写操作(取其之一)。

（4）条件转移微指令



其功能是根据测试对象的状态决定是转移到D所指定的微地址单元，还是顺序执行下一条微指令。9位D字段不足以表示一个完整的微地址，但可以用来替代现行μPC的低位地址。 测试条件字段有4位，可规定16种测试条件。

水平型微指令与垂直型微指令的比较：

(1) 水平型微指令并行操作能力强，效率高，灵活性强，垂直型微指令则较差。

(2) 水平型微指令执行一条指令的时间短，垂直型微指令执行时间长。

(3) 由水平型微指令解释指令的微程序，有微指令字较长而微程序短的特点。垂直型微指令则相反。

(4) 水平型微指令用户难以掌握，而垂直型微指令与指令比较相似，相对来说，比较容易掌握。

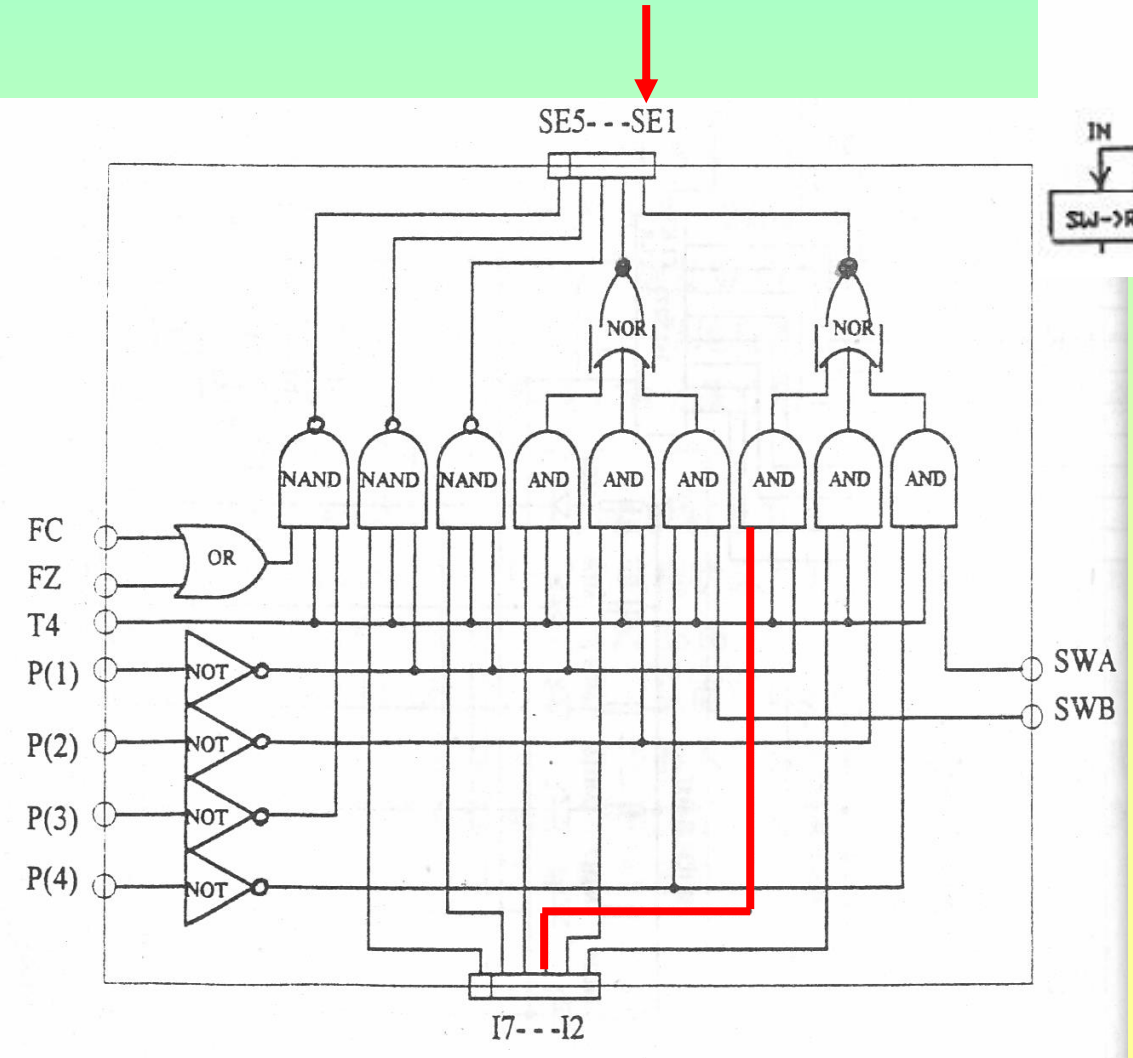
- 例：设计算机有微操作控制信号48个，分为4个相斥的微命令组A、B、C、D，各组分别包含5个、8个、15个和20个微命令，需要用到P1、P2两个判别测试条件，微指令字长24位。请设计该机器的水平型编码表示法微指令格式，并计算控制存储器允许的最大容量。 若采用直接表示控制方式，微指令的操作控制字段有多少位。

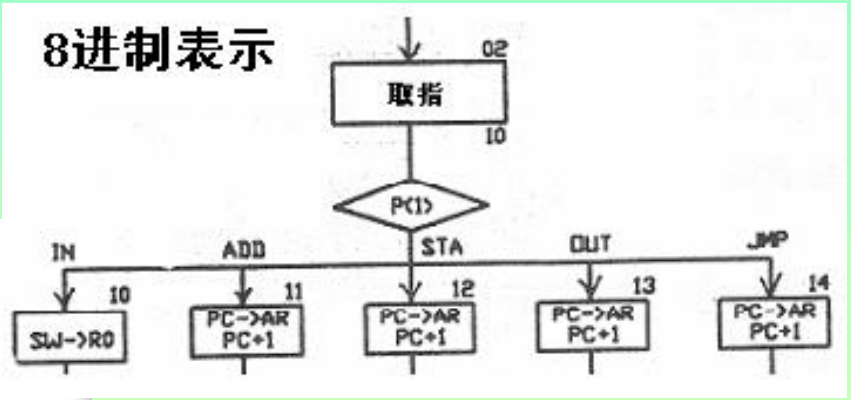
A	B	C	D	判别测试	下地址
3位	4位	4位	5位	2位	6位

控制存储器容量 $2^6 \times 24$位＝192 B

48位

设有地址转移逻辑如下图所示，I2~I7为指令的第2位到第7位，指令最后一位为I0，SE1~SE5输出为0时，微指令后继地址相应位被修改为1，如SE1为0时，后继地址的最后一位被修改为1，微程序分支如图所示，请说出给出的各个机器指令码表示的含义。





I4=1

机器指令码	助记符
0000 0000	IN
0001 0000	ADD addr
0010 0000	STA addr
0011 0000	OUT addr
0100 0000	JMP addr

请回答并**理解**：

1、计算机执行指令的过程。

2、不同指令之间的本质区别是什么。

3、为什么不同指令能够执行不同的操作。

4、微指令与指令的区别与联系。

动态微程序设计

微程序设计技术有静态微程序设计和动态微程序设计之分。

1. **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之后，一般无须改变而且也不好改变，这种微程序设计技术称为静态微程序设计。

2. **动态微程序设计。**当采用EPROM作为控制存储器时，还可以通过改变微指令和微程序来改变机器的指令系统，这种微程序设计技术称为动态微程序设计。采用动态微程序设计时，微指令和微程序可以根据需要加以改变，因而可在一台机器上实现不同类型的指令系统。

硬布线控制器

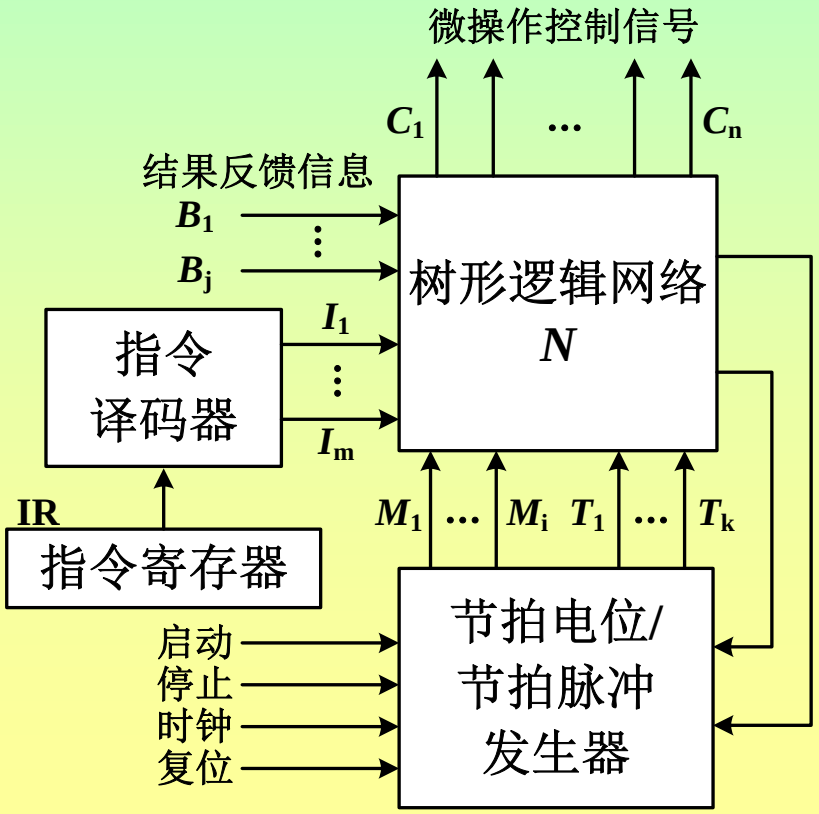
这种方法是把控制部件看做产生专门固定时序控制信号的逻辑电路，而此逻辑电路以使用最少元件和取得最高操作速度为设计目标。一旦控制部件构成后，除非重新设计和物理上对它重新布线，否则要想增加新的控制功能是不可能的。这种逻辑电路是一种由门电路和触发器构成的复杂树形逻辑网络，故称之为硬布线控制器。

逻辑网络输入信号的来源有三个：

（1）指令译码器的输出I_m；

（2）执行部件的反馈信息B_j；

（3）来自时序产生器的时序信号，包括节拍电位信号M_i（机器周期信号）和节拍脉冲信号T_k（时钟周期信号）。

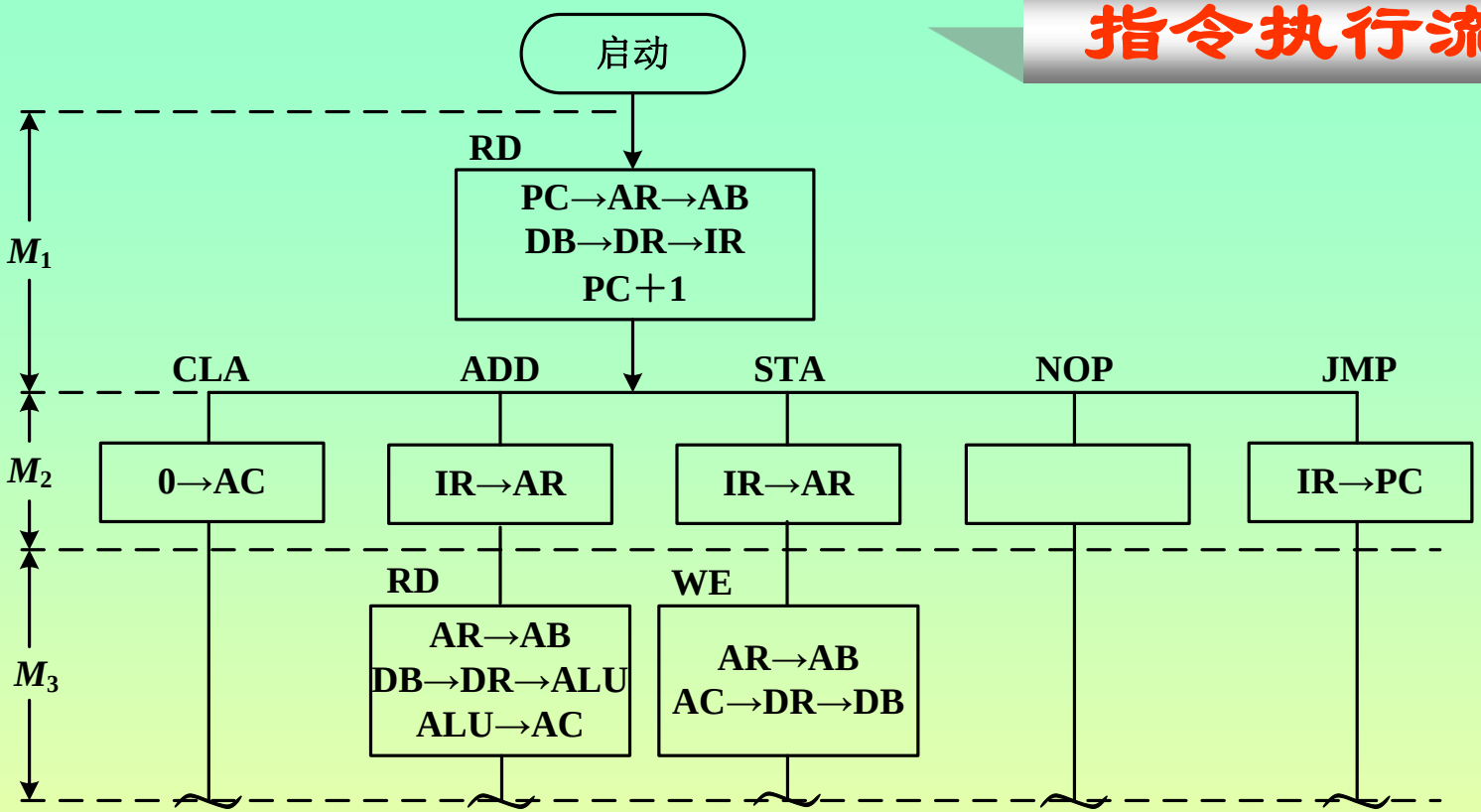


逻辑网络N的输出信号就是微操作控制信号，用来对执行部件进行控制。某一微操作控制信号C是指令译码器输出I_m、时序信号（节拍电位M_i，节拍脉冲T_k）和状态条件信号B_j的逻辑函数：

$$C=f(I_m, M_i, T_k, B_j)$$

与微程序控制器相比，硬布线控制的速度较快。因为微程序控制器中每条指令都要读取控制存储器，而硬布线控制主要取决于电路延迟。近年来在某些超高速计算机中，又选用了硬布线控制器 。

指令执行流程



时序产生器产生节拍电位和节拍脉冲信号。所有指令的取指放在M_1节拍，执行阶段由M_2、M_3两个节拍完成。为简化节拍控制，指令的执行控制可采用同步工作方式，即各指令的执行阶段均采用最长节拍数M_3来考虑。

为避免时间浪费，对CLA、NOP、JMP等短指令，可在执行M_2节拍后跳过M_3节拍而返回M_1节拍。

微操作控制信号的产生

在微程序控制器中，微操作控制信号由微指令产生。在硬布线控制器中，某一微操作控制信号由布尔代数表达式描述的输出函数产生。

设计微操作控制信号的方法和过程是，根据所有机器指令流程图，寻找出产生同一个微操作信号的所有条件，并与适当的节拍电位和节拍脉冲组合，从而写出其布尔代数表达式并进行简化，然后用门电路或可编程器件来实现。要特别注意控制信号是电位有效还是脉冲有效，如果是脉冲有效，必须加入节拍脉冲信号进行相“与”。

例：五条指令的微操作控制信号举例。

$LDAR = M_1 \cdot T_2 + M_2(ADD+STA) \cdot T_2$

$LDDR = M_1 \cdot T_3 + M_3(ADD+STA) \cdot T_3$

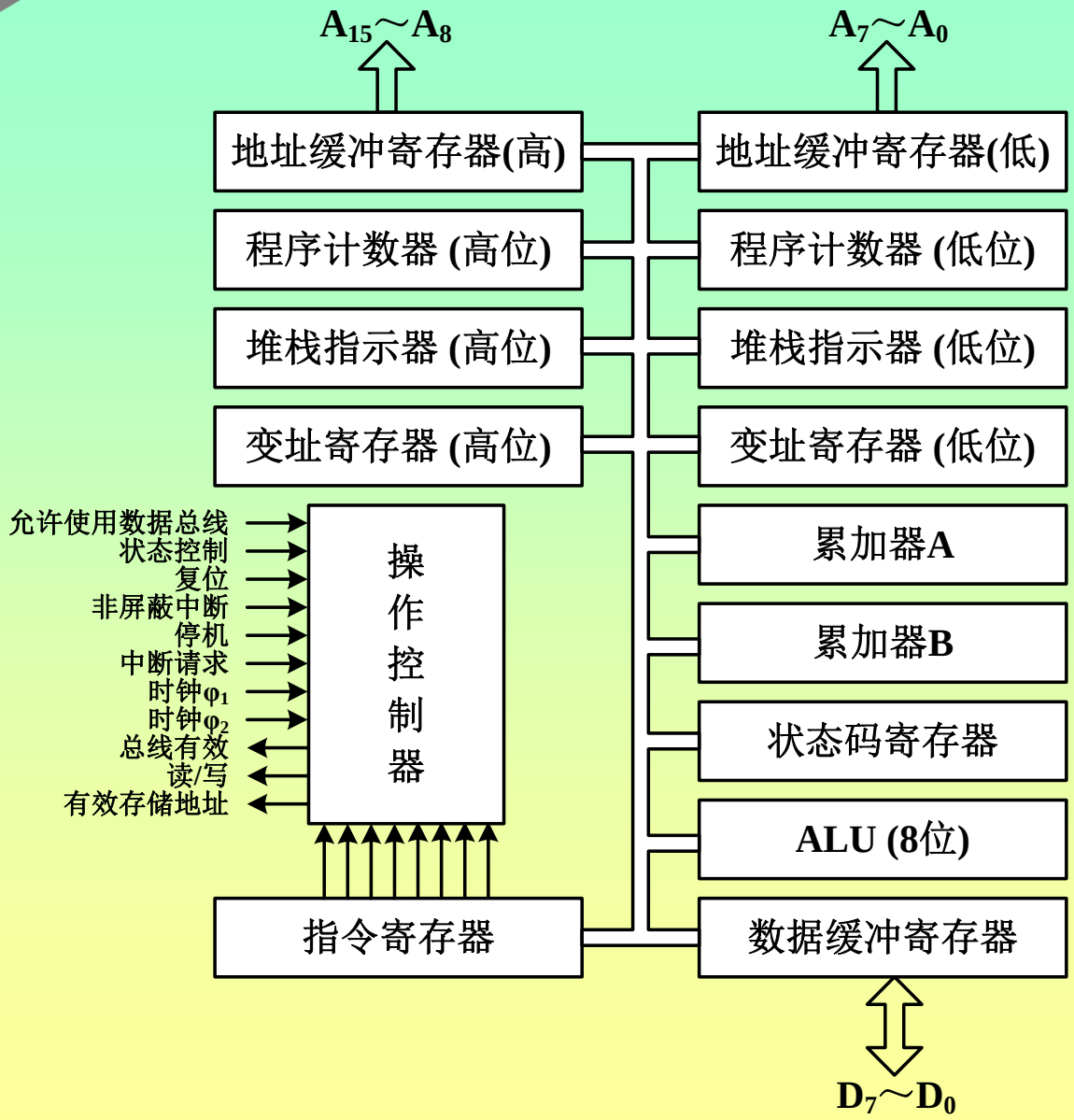
$LDIR = M_1 \cdot T_4$

$RD = M_1 + M_3 \cdot ADD$　　$WE = M_3 \cdot STA$

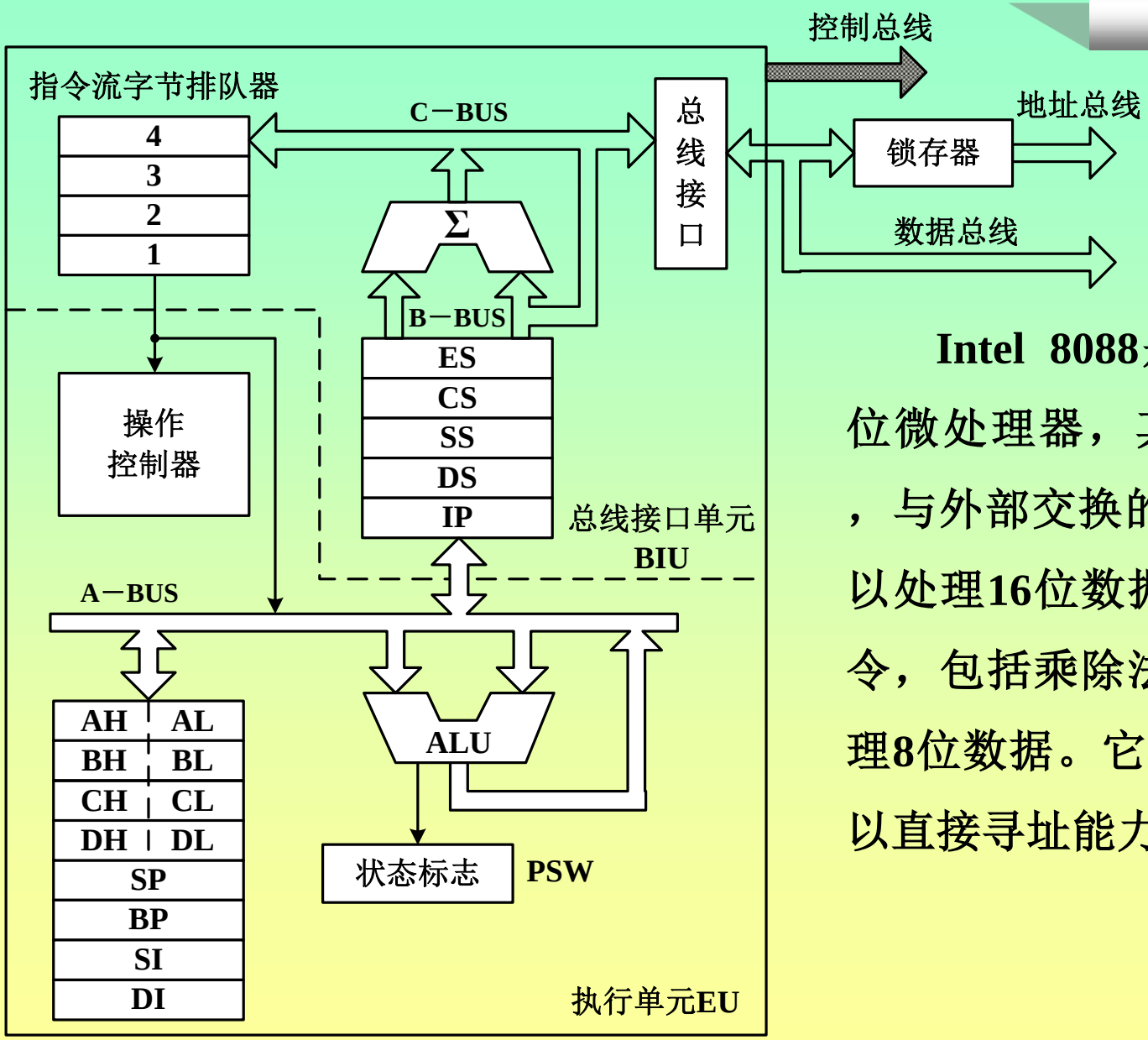
其中M_1、M_2、M_3是三个节拍电位信号；T_2、T_3、T_4为节拍脉冲信号；ADD、STA是指令OP字段译码器的输出信号。

MC6800 CPU

MC6800 CPU是一个单总线结构的8位微处理器，主要包括8位的ALU、16位的程序计数器、16位的堆栈指示器和16位的变址寄存器，两个8位的累加器和一个8位的状态条件码寄存器，一个8位的指令寄存器以及指令译码与控制部件(即操作控制器)。此外还有一个8位的数据缓冲寄存器和一个16位 的地址缓冲寄存器。



8088 CPU



Intel 8088是一种通用的准16位微处理器，其内部结构为16位，与外部交换的数据为8位。它可以处理16位数据(具有16位运算指令，包括乘除法指令)，也可以处理8位数据。它有20条地址线，所以直接寻址能力达到1M字节。

8088 CPU从功能上来说分成两大部分:

总线接口单元BIU：负责与存储器和外围设备接口;

执行单元EU：负责指令的执行。

通用寄存器为16位，其中AX为累加器，BX、CX、DX用于存放操作数。这四个寄存器的高8位和低8位可分开使用。堆栈指针SP用来指示堆栈操作时堆栈在主存的位置，但是SP必须与堆栈段寄存器SS一起使用 。另外三个16个寄存器BP (基数指针)、SI (源变址)、DI (目的变址)用来增加几种寻址方式，从而能更灵活的寻找操作数。

指令指针IP的功能相当于一般机器的程序计数器PC，但是IP要与代码段寄存器CS相配合才能形成真正的物理地址。

状态寄存器PSW由九个标志位组成，以反映操作结果的某些状态或机器运行状态。

四个16位的段寄存器用来存放主存段地址（代码段CS，数据段DS，堆栈段SS，附加段ES)。 通过把某个段寄存器左移4位低位补零后与16位偏移地址相加可形成20位长度的实际地址，从而可使主存具有1MB的寻址能力。

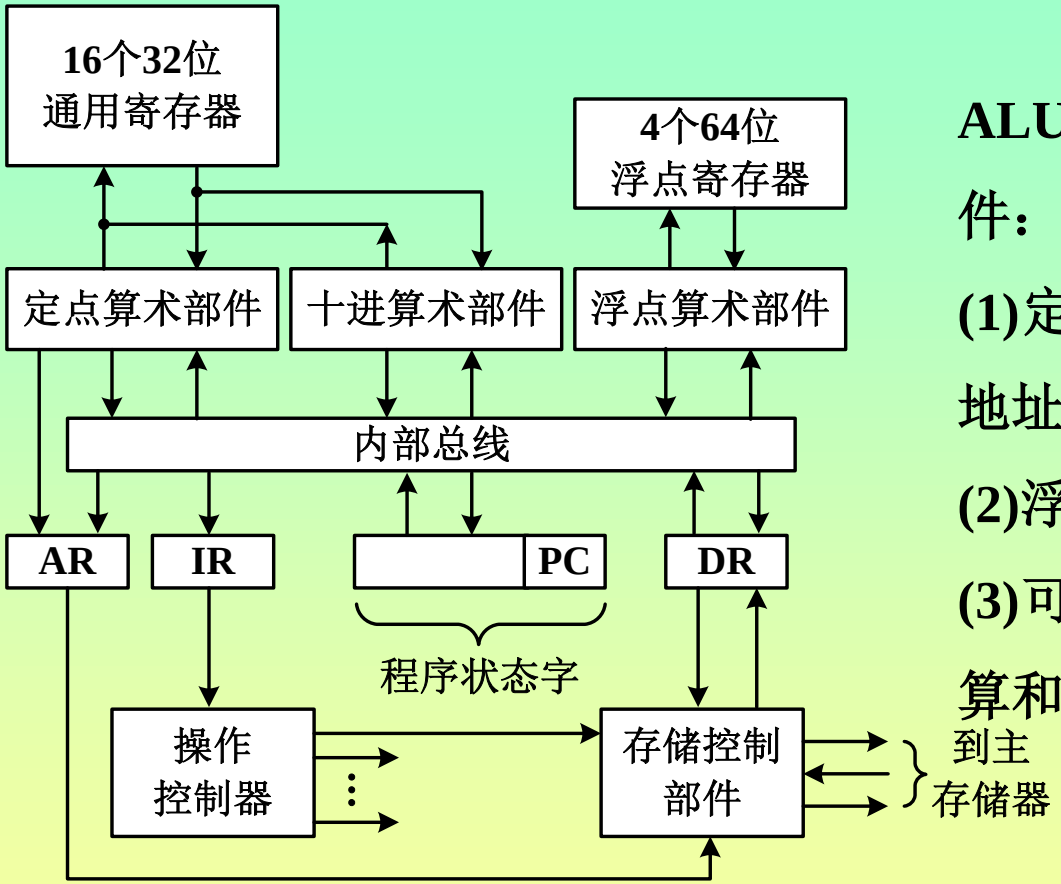
取指令时，CPU自动选择代码段寄存器CS，再加上由IP所决定的16位位移量，便得到所取指令的20位主存物理地址。

进行堆栈操作时，CPU自动选择堆栈段寄存器SS，再加上SP所决定的16位偏移量，便得到堆栈操作所需要的20位物理地址。

涉及到操作数时，CPU自动选择数据段寄存器DS或附加段寄存器ES，再加上16位偏移量，便得到操作数的20位物理地址。此处的16位偏移量，可以是包含在指令中的直接地址，也可以是某一个16位地址寄存器的值，又可以是指令中的偏移量加上16位地址寄存器的值等等，这要取决于指令的寻址方式。

在不改变段寄存器值的情况下，寻址的最大范围是64KB。

IBM 370系列CPU



IBM 370系列机字长32位。ALU部件按功能不同分为三个子部件：

(1)定点运算，包括整数计算和有效地址的计算；

(2)浮点运算；

(3)可变长运算，包括十进制算术运算和字符串操作。

16个32位通用寄存器用来存放操作数和运算结果，并且也可用作变址寄存器。4个浮点寄存器用于浮点运算。数据寄存器DR、地址寄存器AR、指令寄存器IR都是标准化的。

程序状态字PSW存放在专用寄存器中，指明程序运行的状态，可用于CPU响应中断的情况及指明下一条执行指令的地址，PSW主要是为处理中断而使用的。CPU通过将现行的PSW存入主存储器，并取出新的PSW的方式来响应中断。新的PSW指出为处理中断而应执行的程序。一旦该程序执行完毕，CPU可从主存储器取回旧的PSW，再继续执行原来被中断了的程序。

在370系统中，任何时刻CPU都只能处于几种控制状态中的一种。当它在执行操作系统的一段程序时，操作系统明确地控制着CPU，这时我们说CPU处于管理状态(简称管态)。某些指令只允许在这个状态下执行。当CPU在执行用户程序时，则认为处于正常的解题状态(简称目态)。CPU在任何时刻的状态都是由它的PSW来说明的。

为了进行存储保护，PSW寄存器还包含一个存储键。主存储器按每2K字节分成若干块，每块都配置一个存储键。存储键规定了可允许存取的类别，如只允许读、可读可写、不可读写等。 对每块中的信息，只有当该块的存储键与PSW寄存器中的现行键相符时，才可以进行存取操作 。

Intel 80486 CPU

Intel 80486是32位的CPU，其主要特点如下：

(1)通过采用流水技术以及微程序控制和硬布线逻辑控制相结合的方式，进一步缩短可变长指令的译码时间，达到基本指令可以在一个时钟周期内完成。

(2)486芯片内部包含一个8KB的数据和指令混合性cache，为频繁访问的指令和数据提供快速的内部存储，从而使系统总线有更多的时间用于其他控制。

(3)486芯片内部包含了增强型80387协处理器，称为浮点运算部件(FPU)。由于FPU功能扩充且放在CPU内部，使引线缩短，速度比80387提高了3－5倍。

(4)486CPU的内部数据总线宽度为64位，这也是它缩短指令周期的一个原因。而外部数据总线的宽度也可以自动转换。

(5)地址信号线扩充到32位，可以处理4GB的物理存储空间。如果利用虚拟存储器，其存储空间达64TB。

(6)486CPU采用单倍的时钟频率，而在CLK端加入的时钟频率，就是它内部CPU的时钟频率， 因此大大增加了电路的稳定性。

Intel 486的内部结构包含如下九个功能部件：总线接口部件、小容量cache、指令预取部件、指令译码器、段管理部件、页管理部件、定点运算部件ALU、浮点运算部件FPU及操作控制部件。

总线接口部件主要用来产生访问外部存储器和I/O口所需要的地址、数据、命令信号。

段管理部件用来把指令指定的逻辑地址(程序中指定的虚拟地址)变成线性地址。

页管理部件的功能是把线性地址换算成物理地址。

指令预取部件中包含了32字节的预取队列寄存器，可以存放多条指令，因而是一种流水线结构。

ALU中包含了通用寄存器组以及各种算术逻辑运算操作。

FPU则完成浮点数运算、二进制整数运算、十进制数串运算等。

操作控制部件采用微程序控制和硬布线控制相结合的方式。

并行处理技术

并行性的两种含义：

同时性：指两个以上事件在同一时刻发生；

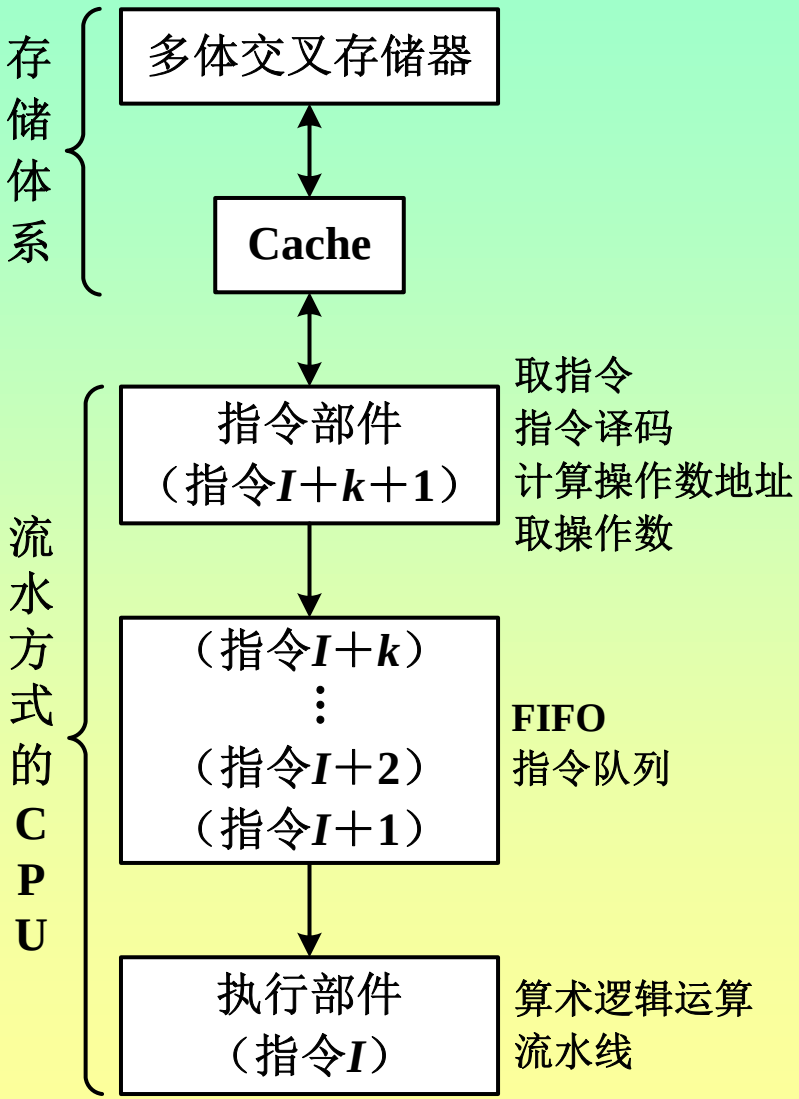
并发性：指两个以上事件在同一时间间隔内发生。

1. **时间并行**：指时间重叠，让多个处理过程在时间上相互错开，轮流重叠地使用同一套硬件设备的各个部分，以加快硬件周转而赢得速度。时间并行性概念的实现方式就是采用流水处理部件。这是一种非常经济而实用的并行技术，能保证计算机系统具有较高的性能价格比。

2. **空间并行**：指资源重复，以“数量取胜” 为原则来大幅度提高计算机的处理速度。大规模和超大规模集成电路的迅速发展为空间并行技术带来了巨大生机，因而成为目前实现并行处理的一个主要途径。

3. **时间并行+空间并行**：时间重叠和资源重复的综合应用 ，既采用时间并行性又采用空间并行性。

流水计算机的系统组成



CPU按流水线方式组织，由三部分组成：指令部件、指令队列、执行部件。这三个功能部件可以组成一个3级流水线。

为了使存储器的存取时间能与流水线的其他各过程段的速度相匹配，一般都采用多体交叉存储器。Cache用于弥补主存和CPU速度上的差异。

指令部件本身又构成一个流水线，由取指令、指令译码、计算操作数地址、取操作数等几个过程段组成。

指令队列是FIFO的寄存器栈，存放经过译码的指令和取来的操作数。

执行部件可以具有多个算术逻辑运算部件，这些部件本身又采用流水线方式构成。

执行段的速度匹配问题：通常采用并行的运算部件以及部件流水线的工作方式来解决。方法包括：

(1) 将执行部件分为定点执行部件和浮点执行部件两个可并行执行的部分， 分别处理定点运算指令和浮点运算指令；

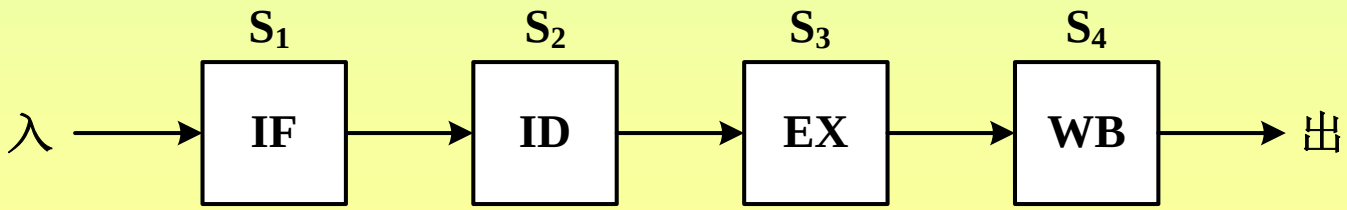
(2) 在浮点执行部件中，又有浮点加法部件和浮点乘/除部件，它们也可以同时执行不同的指令；

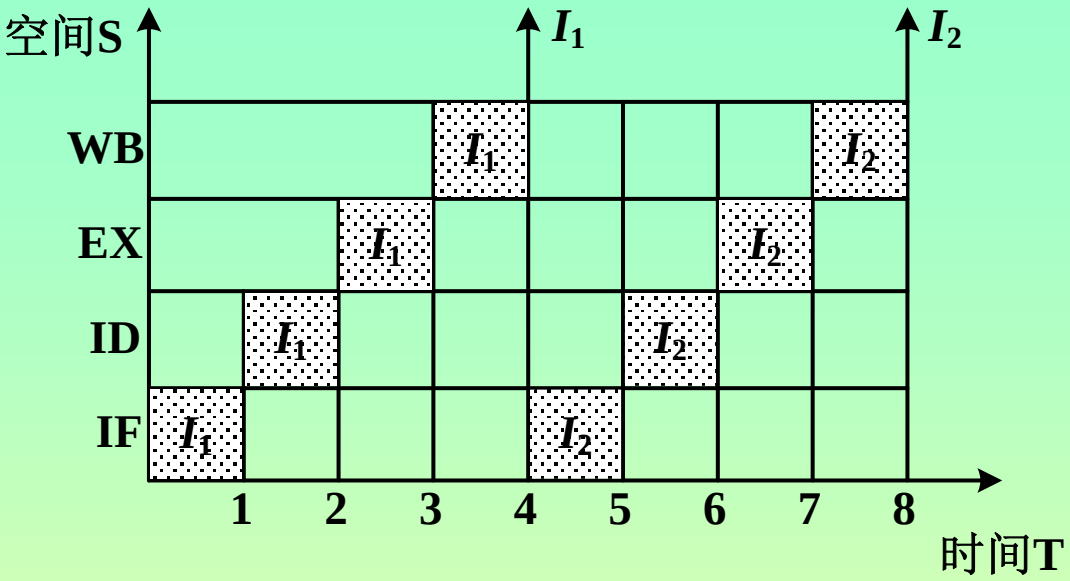
(3) 浮点运算部件都以流水线方式工作。

流水CPU的时空图

为了实现流水，首先把输入的任务(或过程)分割为一系列子任务，并使各子任务能在流水线的各个阶段并发地执行。当任务连续不断地输入流水线时，在流水线的输出端便连续不断地吐出执行结果，从而实现了子任务级的并行性。

假设指令周期包含四个子过程：取指令 (IF)，指令译码 (ID)，执行运算(EX)，结果写回 (WB)，每个子过程称为一个过程段 (S_i)，各过程段之间设有高速缓存，以暂时保存上一过程段子任务处理的结果。在统一的时钟信号控制下，数据从一个过程段流向相邻的过程段。



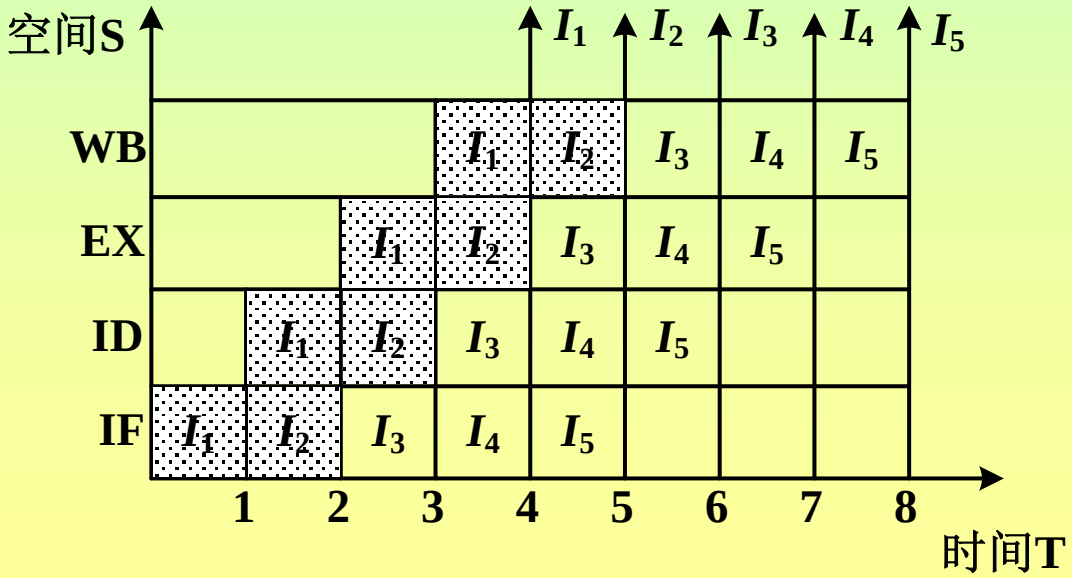


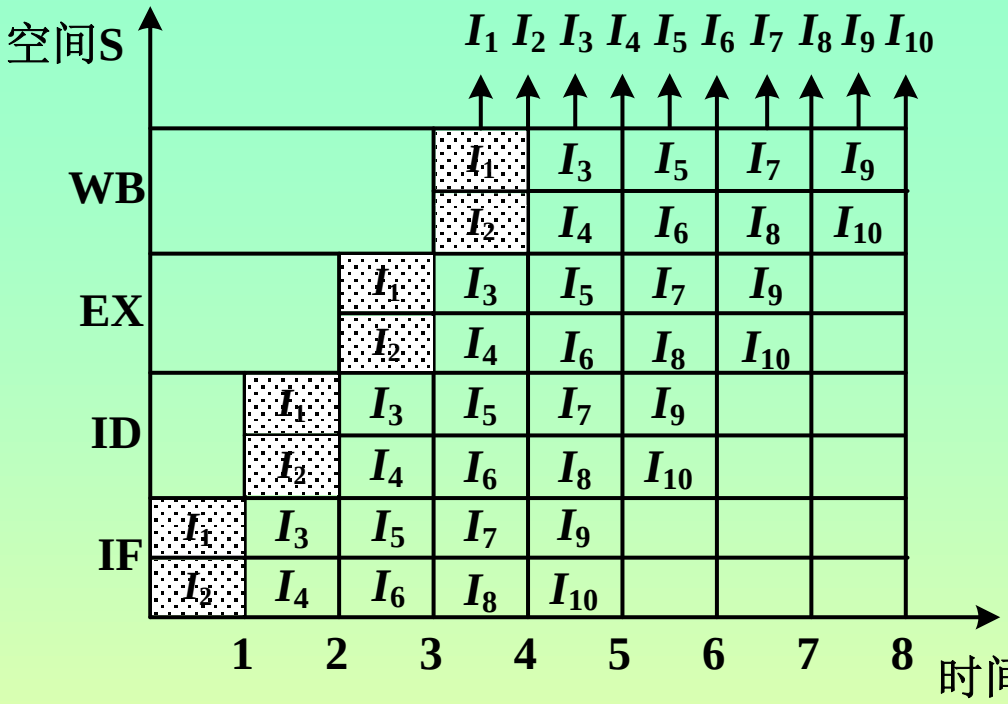
非流水计算机的时空图。

上一条指令的四个子过程全部执行完毕才能开始下一条指令。每隔4个时钟周期有一个输出结果。

流水计算机的时空图。

上一条指令与下一条指令的四个子过程在时间上可以重叠执行。当流水线满载时，每一个时钟周期就可以输出一个结果。





超标量流水计算机的时空图。

只有一条指令流水线的计算机称为标量计算机。超标量流水是指有两条以上的指令流水线。超标量流水计算机是时间并行技术和空间并行技术的综合应用。每个时钟周期可完成多条指令。

流水线的分类

指令流水线：指指令步骤的并行。将指令流的处理过程划分为取指令、译码、执行、写回等几个并行处理的过程段。几乎所有的高性能计算机都采用了指令流水线。

算术流水线：指运算操作步骤的并行。如流水加法器、流水乘法器、流水除法等。 现代计算机中已广泛采用了流水的算术运算器。

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。数据流从第一台处理机输入，经处理后被送入与第二台处理机相联的缓冲存储器中。第二台处理机从该存储器中取出数据进行处理，然后传送给第三台处理机，如此串联下去。随着高档微处理器芯片的出现， 构造处理机流水线变得容易了。处理机流水线应用在多机系统中。

流水线中的主要问题

流水线中会出现以下三种相关冲突，使流水线断流。

1. **资源相关：**多条指令进入流水线后在同一个时钟周期内争用同一个功能部件所发生的冲突。

指令 \ 时钟	1	2	3	4	5	6	7	8
I_1	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

设一条指令流水线包括5段：取指(IF)、指令译码(ID)、计算有效地址或执行(EX)、访存取数(MEM)、结果写寄存器堆(WB)。I_1的MEM段和I_4的IF段都要访问存储器，发生争用存储器资源的冲突。解决办法是I_4停顿一拍再启动，或者将指令和数据存放在两个存储器中。

2. 数据相关：如果必须等前一条指令执行完毕后，才能执行后一条指令，这两条指令就是数据相关的。当后继指令的操作数是前一条指令的运算结果时，就会发生数据相关冲突。例如：

ADD　R_1，R_2，R_3；　　$(R_2) + (R_3) \rightarrow R_1$

SUB　R_4，R_1，R_5；　　$(R_1) - (R_5) \rightarrow R_4$

AND　R_6，R_1，R_7；　　$(R_1) \wedge (R_7) \rightarrow R_6$

时钟 / 指令	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指令在时钟5写寄存器R_1，而SUB指令在时钟3、AND指令在时钟4读R_1，产生数据相关冲突。

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

3. 控制相关：控制相关冲突是由转移指令引起的。当执行转移指令时，依据转移条件的产生结果，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，从而使流水线发生断流。

为了减小转移指令对流水线性能的影响，常用以下两种转移处理技术：

延迟转移法：由编译程序重排指令序列来实现。基本思想是“先执行再转移”，即发生转移时并不排空指令流水线，而是让紧跟在转移指令I_b之后已进入流水线的少数几条指令继续完成。如果这些指令是与I_b结果无关的有用指令，那么延迟损失时间片正好得到了有效的利用。

转移预测法：用硬件方法来实现，依据指令过去的行为来预测将来的行为。通过使用转移取和顺序取两路指令预取队列器以及目标指令cache，可将转移预测提前到取指阶段进行，以获得良好的效果。

例：流水线中有三类数据相关冲突：写后读 (RAW)相关；读后写 (WAR)相关；写后写 (WAW)相关。判断以下三组指令各存在哪种类型的数据相关。

(1) I1: ADD R_1，R_2，R_3；　　$(R_2)+(R_3)\rightarrow R_1$　　RAW

I2: SUB R_4，R_1，R_5；　　$(R_1)-(R_5)\rightarrow R_4$

(2) I3: STA M(x)，R_3；　　$(R_3)\rightarrow$M(x)，M(x)是存储器单元

I4: ADD R_3，R_4，R_5；　　$(R_4)+(R_5)\rightarrow R_3$　　WAR

(3) I5: MUL R_3，R_1，R_2；　　$(R_1)\times(R_2)\rightarrow R_3$

I6: ADD R_3，R_4，R_5；　　$(R_4)+(R_5)\rightarrow R_3$　　WAW

Pentium CPU

Pentium是Intel公司生产的超标量流水处理器，早期使用5V工作电压，后期使用3.3V工作电压。CPU的主频是片外主总线时钟频率(60MHz或66MHz)的倍频，有120，166，200MHz等多种。

CPU内部的主要寄存器宽度为32位，故认为它是一个32位微处理器。但它通向存储器的外部数据总线宽度为64位，每次总线操作可以同时传输8个字节。以主总线(存储器总线)时钟频率66MHz计算，64位数据总线可使CPU与主存的数据交换速率达到528MB/s。CPU支持多种类型的总线周期，其中一种称猝发模式，在此模式下，可在一个总线周期内读出或写入256位(32字节)的数据。

CPU外部地址总线宽度是36位，但一般使用32位宽，故物理地址空间为4096MB(4GB)。虚拟地址空间为64TB，分页模式除支持4KB页面外(与486相同)，还支持2MB和4MB页面。其中2MB页面的分页模式必须使用36位地址总线。

CPU内部分别设置指令cache和数据cache，外部还可接L2 cache。CPU采用U、V两条指令流水线，能在一个时钟周期内发射两条简单的整数指令，也可发射一条浮点指令。操作控制器采用硬布线控制和微程序控制相结合的方式。大多数简单指令用硬布线控制实现，在一个时钟周期内执行完毕。对微程序实现的指令，也在2－3个时钟周期内执行完毕。

Pentium具有非固定长度的指令格式，9种寻址方式，191条指令，但是在每个时钟周期又能执行两条指令。因此它具有CISC和RlSC两者的特性，不过具有的CISC特性更多一些，因此被看成为一个CISC结构的处理器。以CISC结构实现超标量流水线，并有BTB (Branch Target Buffer)方式的转移预测能力，堪称为当代CISC机器的经典之作。

（1）超标量流水线：由U和V两条指令流水线构成，每条流水线都有自己的ALU、地址生成电路、与数据cache的接口。两个指令预取缓冲器，每个都是32字节，负责由指令cache或主存取指令并放入其中。

（2）指令cache和数据cache：各8KB，都是2路组相联结构，每行32字节。每个cache都有一个后援缓冲器TLB (Translation Lookaside Buffer)，负责将TLB命中的线性地址转换成32位物理地址。

（3）浮点运算部件：奔腾CPU内部包含了一个8段的流水浮点运算器。支持*IEEE* 754标准的单、双精度格式的浮点数，另外还使用一种称为临时实数的80位浮点数。

（4）动态转移预测技术：转移目标缓冲器BTB (Branch Target Buffer)是一个小容量的cache。当一条指令导致程序转移时，BTB便记录这条指令及其转移目标地址。以后遇到这条转移指令时，BTB会依据前后转移发生的历史来预测该指令这次是转移取还是顺序取。

RISC CPU

RISC（精简指令系统计算机）的三个要素是：

(1)一个有限的简单的指令集；(2)CPU配备大量的通用寄存器；(3)强调对指令流水线的优化。

基于三要素的RISC机器的特征是：

(1)使用等长指令，目前的典型长度是4个字节。

(2)寻址方式少且简单，一般为2－3种，最多不超过4种，绝不出现存储器间接寻址方式。

(3)只有取数指令、存数指令访问存储器。指令中最多出现RS型指令，绝不出现SS型指令。

(4)指令集中的指令数目一般少于100种，指令格式一般少于4种。

(5)指令功能简单，控制器多采用硬布线方式，以期更快的执行速度。

(6)平均而言，所有指令的执行时间为一个处理时钟周期。

(7)指令格式中用于指派整数寄存器的个数不少于32个，用于指派浮点数寄存器的个数不少于16个。

(8)强调通用寄存器资源的优化使用。

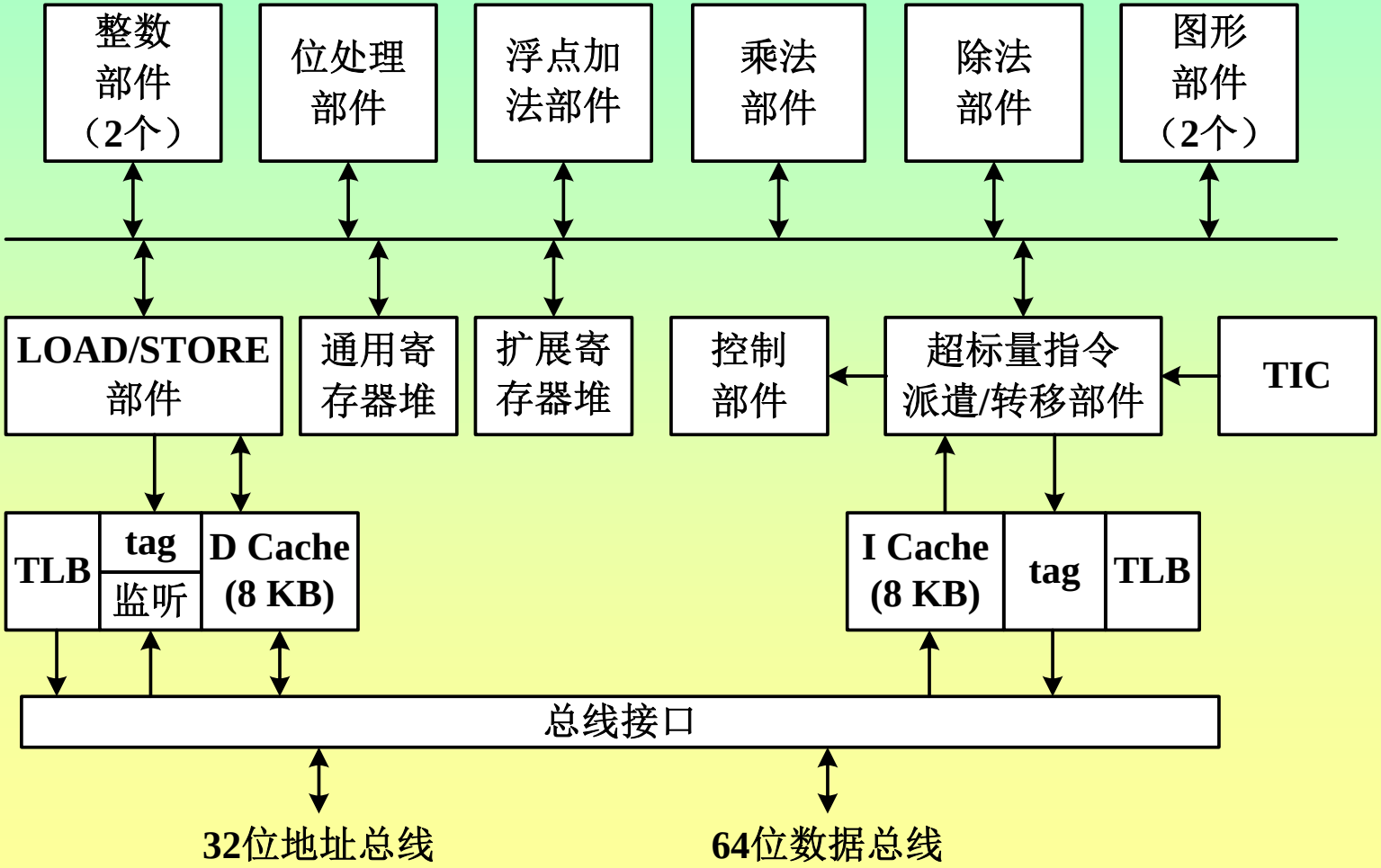
(9)支持指令流水并强调指令流水的优化使用。

(10)编译程序复杂，因此软件系统开发时间比CISC机器长。

比较内容	CISC	RISC
指令系统	复杂，庞大	简单，精简
指令数目	一般大于200	一般小于100
指令格式	一般大于4	一般小于4
寻址方式	一般大于4	一般小于4
指令字长	不固定	等长
可访存指令	不加限制	只有LOAD/STORE指令
各种指令使用频率	相差很大	相差不大
各种指令执行时间	相差很大	绝大多数在一个周期内完成
优化编译实现	很难	较容易
程序源代码长度	较短	较长
控制器实现方式	绝大多数为微程序控制	绝大多数为硬布线控制
软件系统开发时间	较短	较长

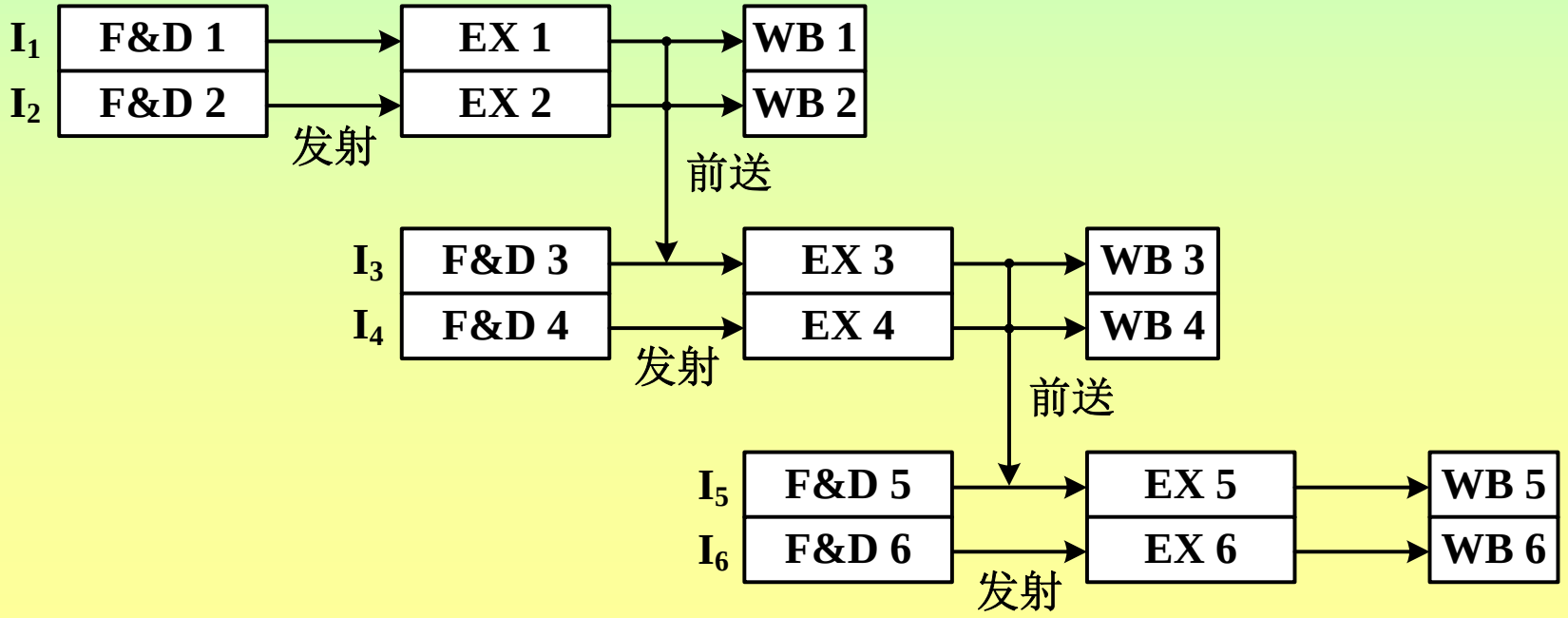
MC88110 CPU

MC 88110 CPU是Motorola公司生产的RISC处理器。包括12个执行功能部件，3个Cache，两个寄存器堆和一个控制部件。

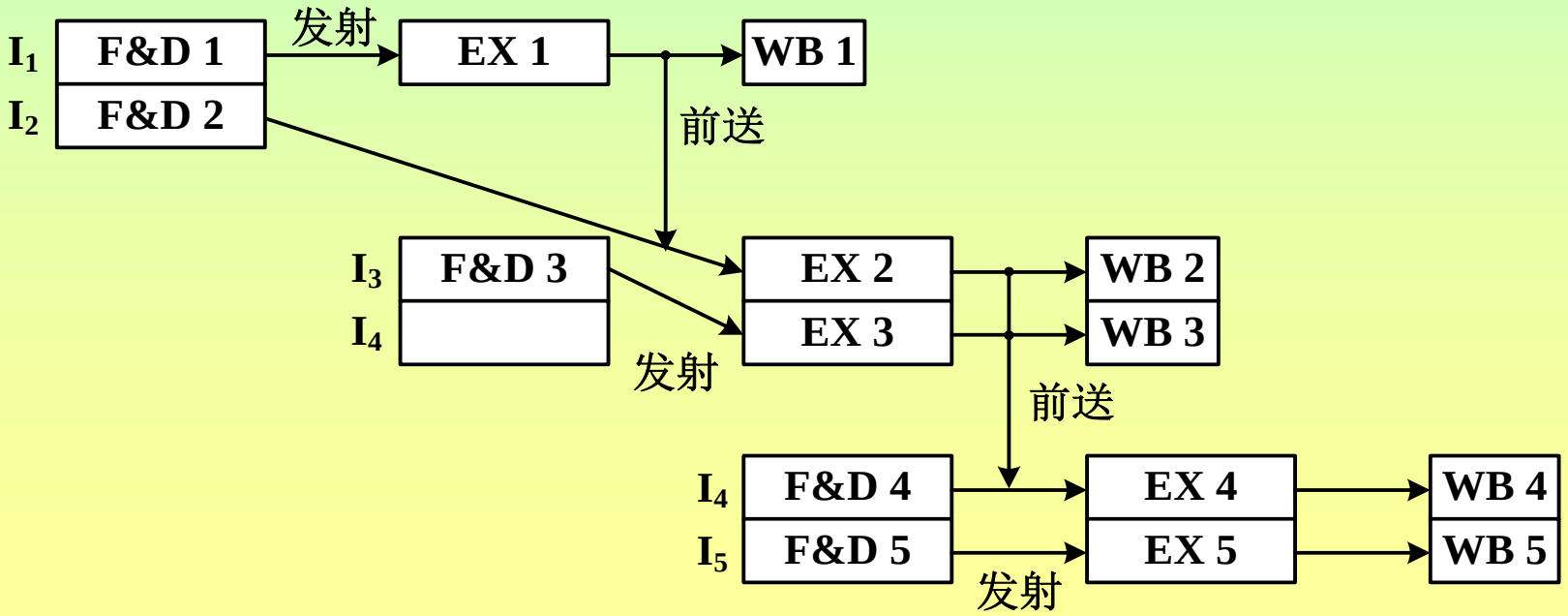


MC 88110 是超标量流水CPU，指令流水线在每个时钟周期可完成两条指令。流水线分三段：取指和译码 (F&D)段、执行 (EX)段、写回 (WB)段。

F&D段需一个时钟周期，完成由指令Cache取出一对指令并译码，并从寄存器堆取操作数。若无相关冲突则把指令发射给EX段。EX段对大多数指令只需一个时钟周期。EX段的执行结果在WB段写回寄存器堆，WB段只需半个时钟周期。为解决数据相关冲突，EX段的执行结果一方面写回寄存器堆，另一方面经定向传送电路提前传送给ALU，供当前进入EX段的指令使用。



MC 88110采用按序发射、按序完成的指令动态调度策略。指令派遣单元总是发出单一地址，然后从指令cache取出此地址及下一地址的两条指令。若这对指令的第一条指令由于资源冲突或数据相关冲突，则这一对指令都不发射，两条指令在F＆D段停顿，等待资源的可用或数据相关的消除。若是第一条指令能发射第二条指令不能发射，则只发射第一条指令，而第二条指令停顿并与新取的指令之一进行配对等待发射，此时原第二条指令作为配对的第一条指令对待。可见，这样实现的方式是按序发射。



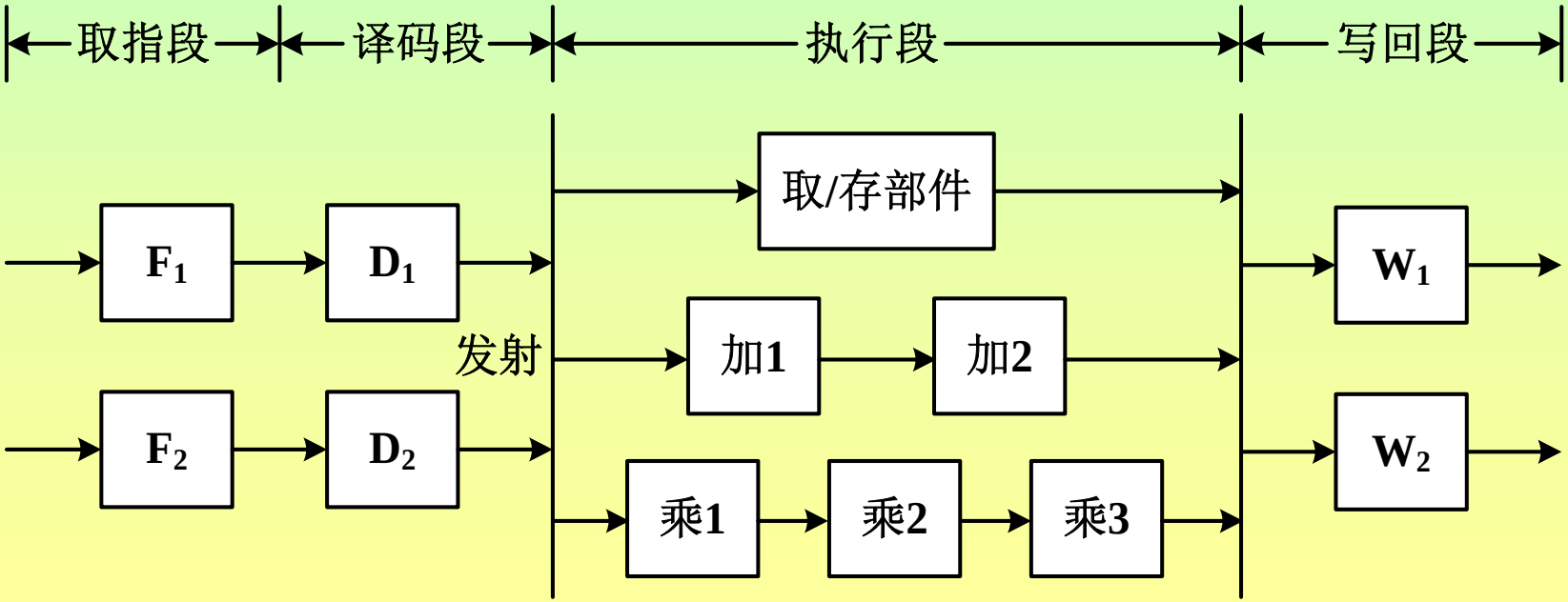
为了判定能否发射指令，88110使用了计分牌方法。计分牌是一个位向量，寄存器堆中每个寄存器都有一个相应位。每当一条指令发射时，它预约的目的寄存器在位向量中的相应位上置“1”，表示该寄存器“忙”。当指令执行完毕并将结果写回此目的寄存器时，该位被清除。判定是否发射一条指令时，一个必须满足的条件是：该指令的所有目的寄存器、源寄存器在位向量中的相应位都已被清除。否则，指令必须停顿等待这些位被清除。

为实现按序完成，88110提供了一个FIFO指令执行队列，称为历史缓冲器。每当一条指令发射出去，它的副本就被送到FIFO队尾。当它到达队首并执行完毕后才离开队列。

对于转移处理，88110使用延迟转移法和目标指令Cache (TIC)法。若采用延迟转移，则跟随在转移指令后的指令将被发射；否则，在转移指令发射之后的转移延迟时间片内不发射任何指令。延迟转移通过编译程序来调度。

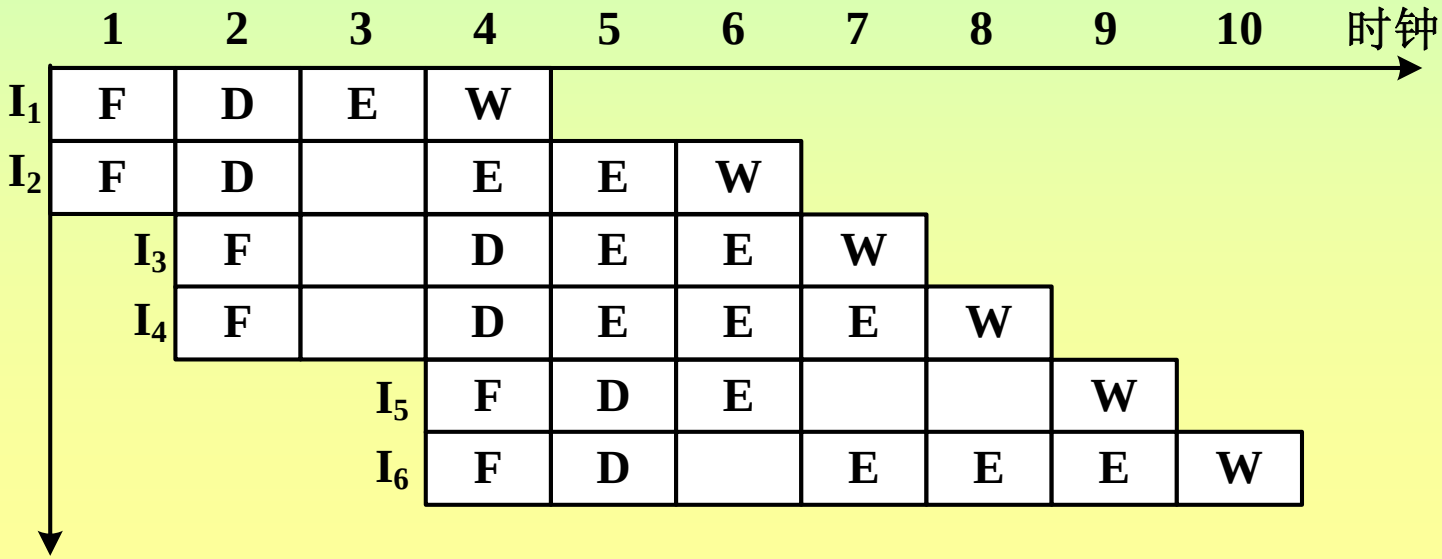
TIC是一个32项的全相联Cache，每项能保存转移目标路径的前两条指令。当一条转移指令译码并命中Cache时，能同时由TIC取来它的目标路径的前面两条指令。

例：超标度为2的超标量流水线结构模型如图所示。它分为4个段，即取指(F)段、 译码(D)段、执行(E)段和写回(W)段。F，D，W 段只需1个时钟周期完成。E段有多个功能部件，其中LOAD/STORE部件完成数据Cache访问，只需一个时钟周期；加法器完成需2个时钟周期，乘法器需3个时钟周期，它们都已流水化。F段和D段要求成对输入。E段有内部数据定向传送，结果生成即可使用。



现有如下6条指令序列，其中I_1、I_2有RAW相关，I_3、I_4有WAR相关，I_5、I_6有WAW相关和RAW相关。

I_1	LAD R_1，A	；M(A)→R_1，M(A)是存储器单元
I_2	ADD R_2，R_1	；(R_2) ＋ (R_1) →R_2
I_3	ADD R_3，R_4	；(R_3) ＋ (R_4) →R_3
I_4	MUL R_4，R_5	；(R_4)×(R_5) →R_4
I_5	LAD R_6，B	；M(B)→R_6，M(B)是存储器单元
I_6	MUL R_6，R_7	；(R_6)×(R_7) →R_6



说明：

(1) 由于I_1和I_2存在RAW相关，I_2要等到I_1执行段结束后才能发射。

(2) 在时钟3，由于I_2仍处在译码段，I_3和I_4无法进入译码段，要等到时钟4才能译码；同样，在时钟3，由于I_3和I_4仍处在取指段，I_5和I_6要等到时钟4才能取指。

(3) I_3和I_4存在WAR相关，但可并行操作，不会导致错误。

(4) I_5在时钟6已执行完成，但为保证按序完成，要等到时钟9才写回。

(5) I_5和I_6存在RAW相关，在时钟7，I_5已执行完成，可通过定向传送电路为I_6提供操作数，使I_6启动执行操作。I_5和I_6还存在WAW相关，只要I_6的完成在I_5之后，就不会出错。

多媒体 CPU

MMX是一种多媒体扩展结构技术，它极大提高了计算机在多媒体和通信应用方面的功能。带有MMX技术的CPU特别适合于数据量很大的图形、图像数据处理。

MMX技术集成到新一代Pentium CPU时，主要体现在：

①采用4种新的数据类型，②使用8个64 位宽的MMX寄存器，③增设57条新指令。

1. MMX数据类型：MMX技术定义了三种打包的数据类型及一种64位字长的数据类型。打包数据类型中的每个元素以及64位数都是带符号或不带符号的定点整数 (字节、字、双字、四字)。四种数据类型定义如下：

紧缩字节类型：8个字节打包成一个64位数据

紧缩字类型：4个字打包成一个64位数据

紧缩双字类型：两个32位的双字打包成一个64位数据

四字类型：一个64位数。

2. MMX寄存器：8个MMX寄存器MM0－MM7的宽度为64位，但它们没有单独设置，而是借用浮点处理单元中的8个(80位)数据寄存器，它是通过使用“别名”的办法来实现的。即浮点单元的8个数据寄存器被浮点指令看成ST0－ST7，被MMX指令看成是MM0－MM7。

这样，8个字节或4个字或2个双字被打包装入一个64位的MMX寄存器，一旦执行一条MMX指令时，将所有这些8个、4个或2个的数据同时取出，进行数学运算或逻辑操作，最后结果写入MMX寄存器。事实上，这种运算处理过程是一种并行处理过程，故称为SIMD(单指令多数据)的并行处理。

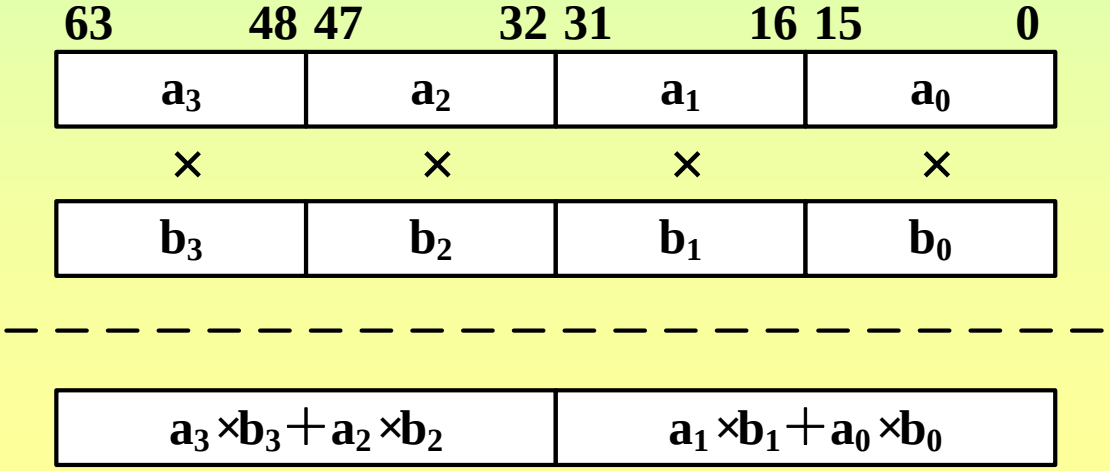
3. MMX指令集：增加了57条MMX指令。

MMX指令的先进型体现在：

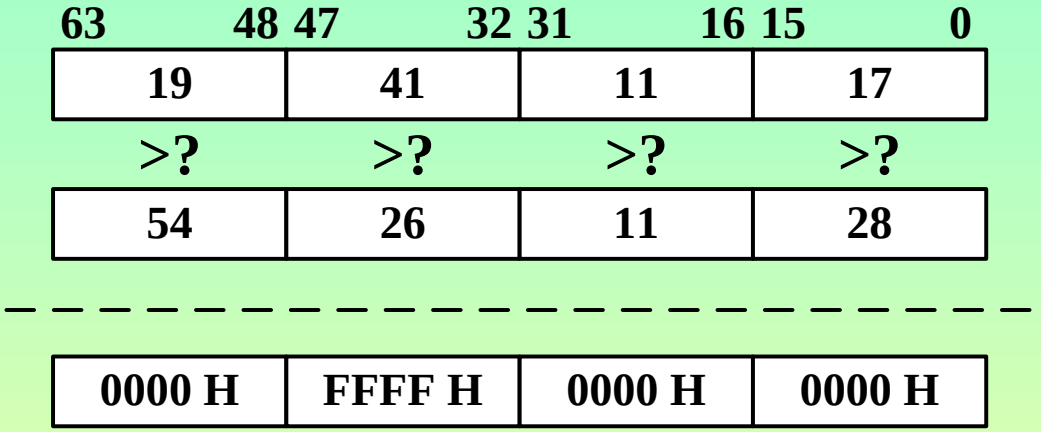
SIMD结构：MMX指令充分利用CPU64位带宽的处理能力，一次可以并行处理8个8位数据，或4个16位数据，或2个32位数据，因而成为提高计算性能的最基本因素。

饱和运算方式：这是运算发生溢出时使用的处理方法。如果运算结果超过最大值，则将此值按最大值处理，低于最小值时按最小值处理。由于不需要进行溢出处理， 所以提高了处理能力。饱和运算适合于面向像素数据的处理。

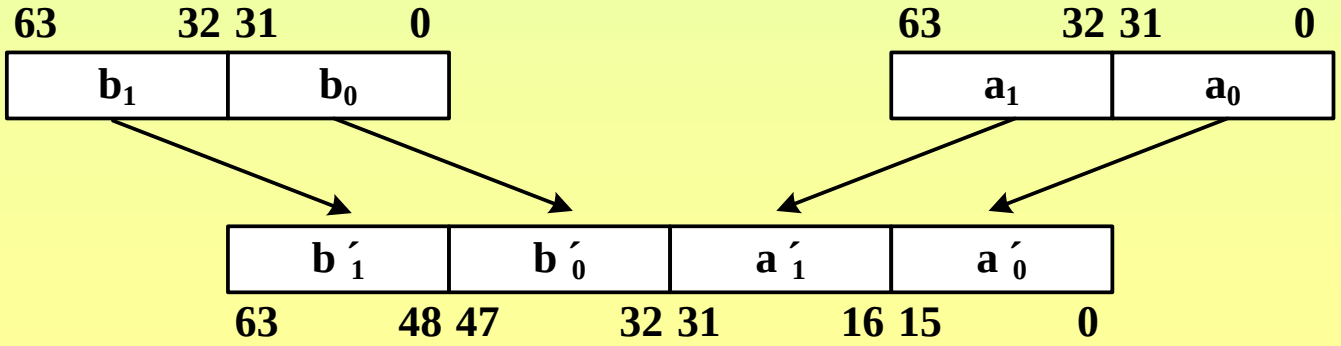
积和运算方式：PMADDWD指令是一条关键指令，它具有乘法－累加操作功能，特别适合于向量计算与矩阵计算。



比较指令特点：MMX的比较指令不建立标志位，而是建立真假条件屏蔽字，后跟一个逻辑操作，从不同的输入中选择所需要的元素，取消了转移指令。比较结果为全“0”表示假条件，全“1”表示真条件。



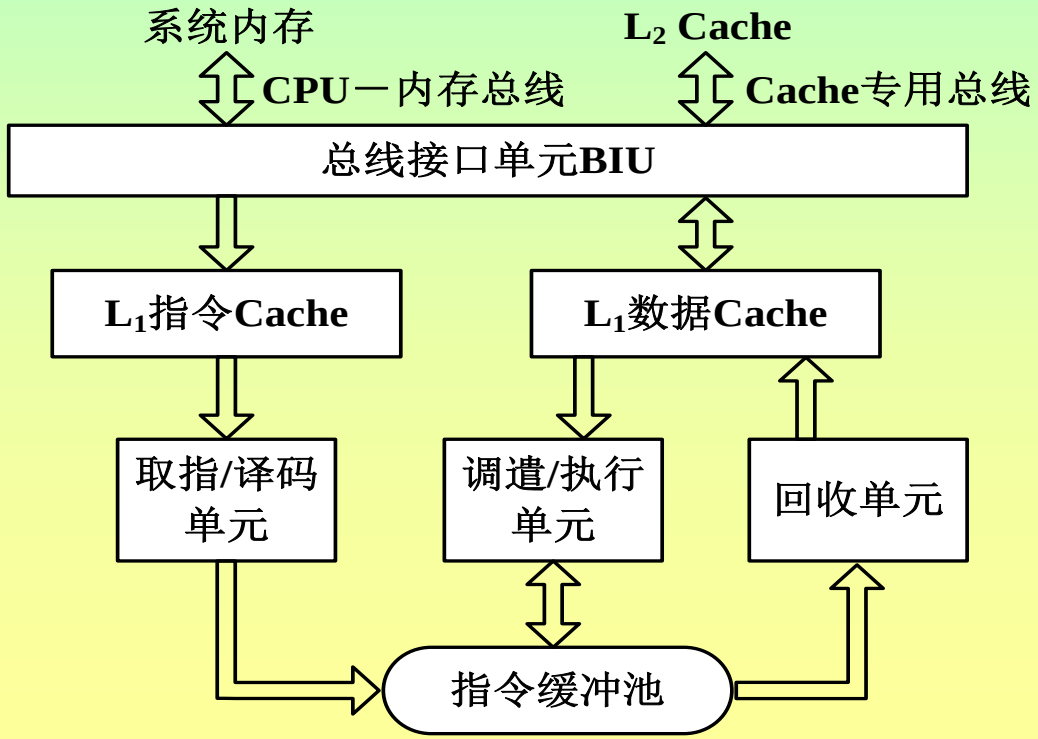
转换指令特点：紧缩或解紧缩指令能方便地完成各种精度的数据转换，其中紧缩指令带有饱和操作。以PACKSSDW紧缩指令为例，它取出4个32位数，将其紧缩为4个16位的数。如果某个数比16位数大，则执行饱和操作。



动态执行技术

动态执行技术是指通过预测程序流来调整指令的执行，并分析程序的数据流来选择指令执行的最佳顺序。它非常利于MMX指令的加速执行。

实现动态执行的关键：取消传统的“取指”和“执行”两阶段之间指令需要线性排列的限制，而使用一个指令缓冲池以开辟一个较长的指令窗口，以便允许执行单元能在一个较大的范围内调遣和执行已译码过的程序指令流。



取指/译码单元：从指令Cache读取程序指令流，将其译码成相应的微操作系列，以指明该指令流所需的数据流。遇到转移指令，通过转移目标缓冲器BTB来预测是否发生转移。它有三个并行的指令译码器ID，故一个CPU周期能向指令缓冲池同时送入3个微操作。

调遣/执行单元：从数据Cache接收数据流，根据数据的相关性和资源可用性来规划微操作的执行，并暂存推测执行的结果。这个过程并不严格按照程序中原来的顺序执行微操作，因此是一个无序完成的过程。调遣/执行单元在一个CPU周期内最多能执行5个微操作，但一般是执行3个微操作。

回收单元：检查指令缓冲池中的微操作状态，找出那些已被执行完的微操作，并且按原始顺序对它们重新排序。如果一条指令的全部微操作均已完成，则按原始顺序逐个回收，将它们保存在回收寄存器RRF中，并删除指令缓冲池中该指令的全部微操作。虽然调遣/执行单元以无序方式执行指令微操作，而回收单元保证最终能得到符合程序要求的指令执行正确结果。回收单元能够在一个CPU周期内同时回收3个微操作。