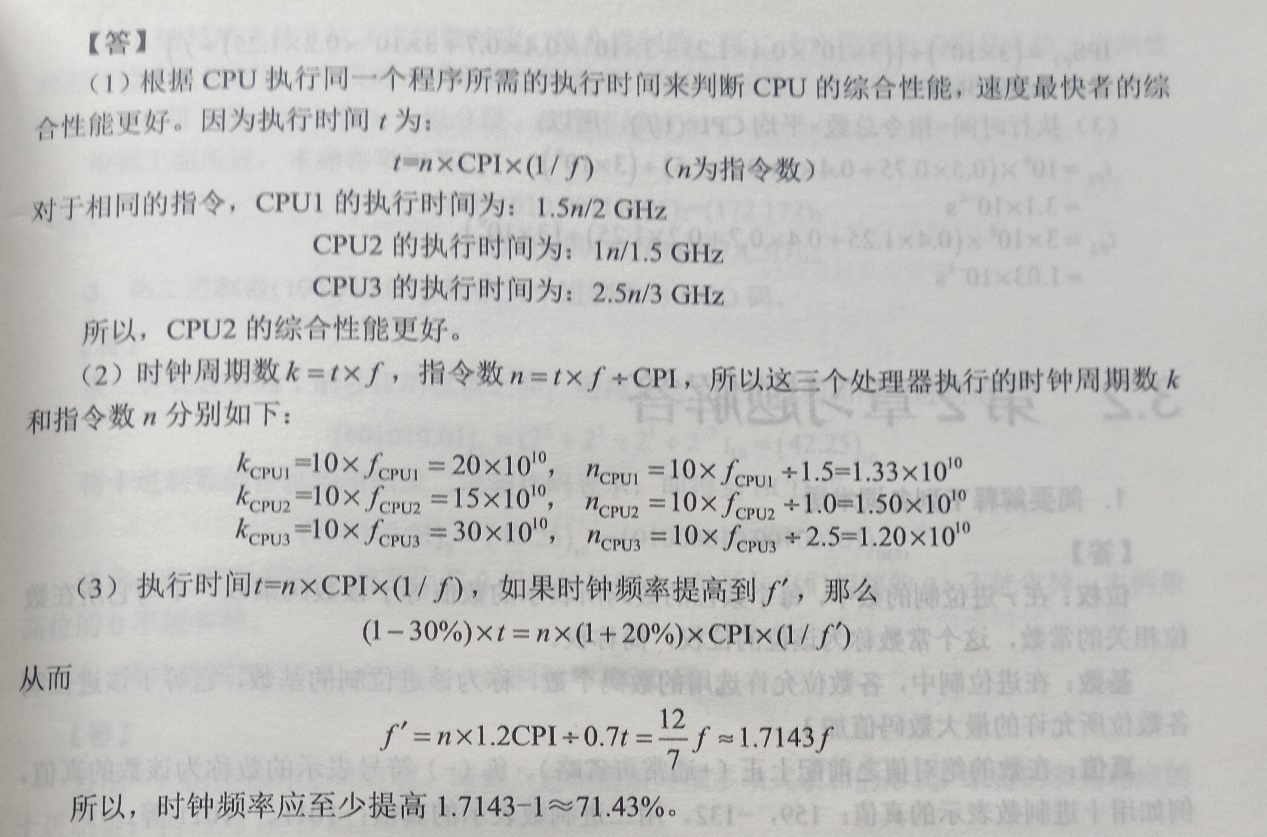


CPI： 平均每条指令执行周期数。

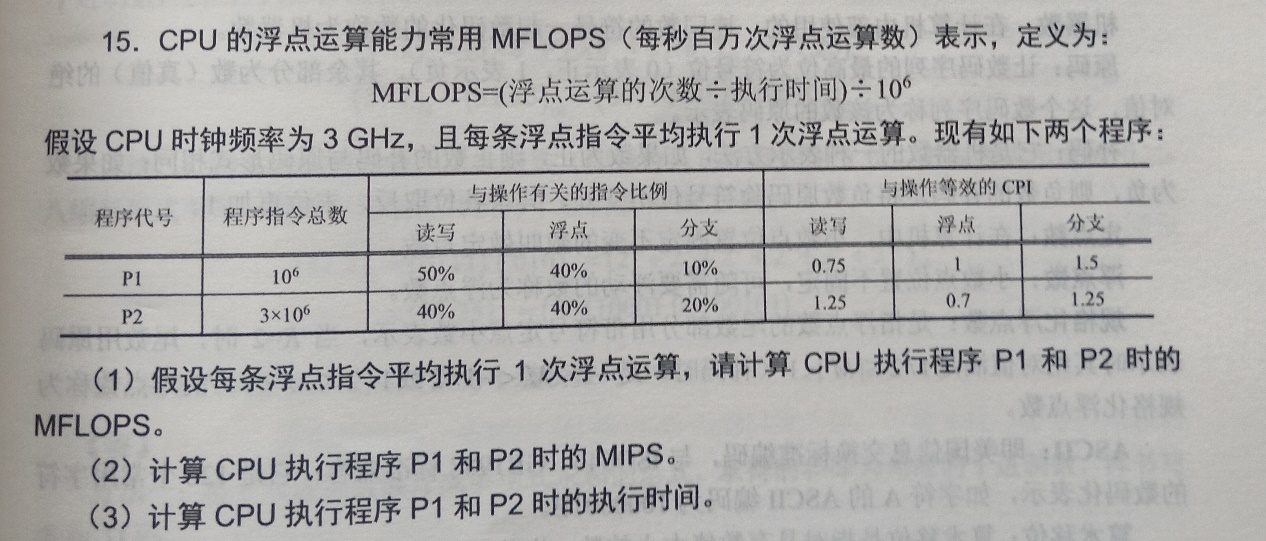
CPI/时钟频率=平均每条指令执行时间。该值越小，性能越好。

时钟周期数=时间\*每秒周期数（时钟频率） 该值除以CPI就是指令数



每秒执行指令数n=时钟频率/CPI 在CPI增加为原值1.2的情况下，要使得n变化为原值的1/0.7(执行时间为原来的0.7，单位时间里能执行的指令数为1/0.7)

1/0.7=X/1.2---X=1.2/0.7=1.7143



FLOPS:每秒执行浮点运算数，计算某个给定程序的该特征值，需要先计算程序总执行时间，再用浮点指令数除以总执行时间。

根据题干给出的指令比例，计算整个程序执行所需要的时钟周期数，乘以每个周期的时长（时钟频率的倒数）就是程序执行总的时间花费。

P1总的时钟周期数=1M\*0.5\*0.75+1M\*0.4\*1+1M\*0.1\*1.5=0.925M

（3）P1总的时间花费=0.925M/3G=3.1\*10-4S

P2总的时钟周期=3M\*0.4\*1.25+3M\*0.4\*0.7+3M\*0.2\*1.25=3.09M

（3）P2总的时间花费=3.09M/3G=1.03\*10-3S

1. P1的浮点指令数 0.4M 单位时间内执行的浮点指令数= 0.4M/(0.925M/3G)=1.2G/0.925=1.297GFLOPS=1297MFLOPS

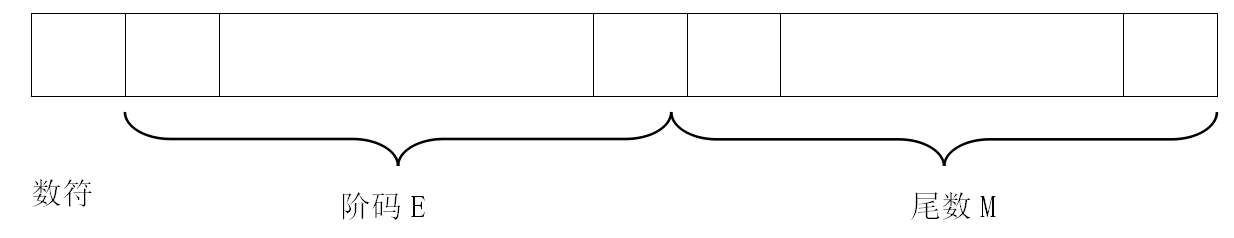
P2的浮点指令数 3\*0.4=1.2M 单位时间内执行的浮点指令数= 1.2M/(3.09M/3G)=3.6G/3.09=1.165GFLOPS=1165MFLOPS

1. MIPS为单位时间执行的百万条指令数

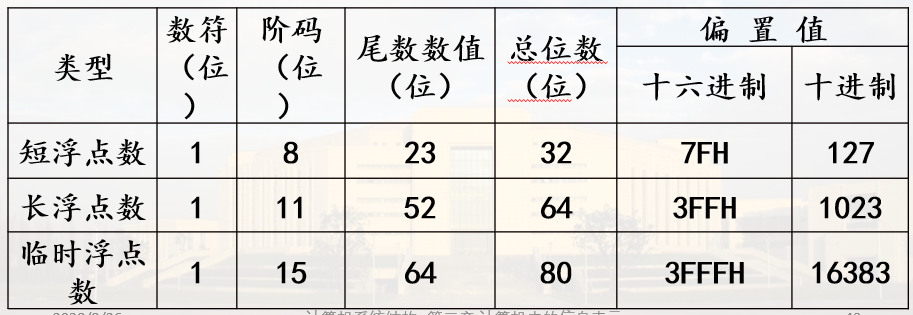
P1:1M/（0.925M/3G）=3243MIPS

P2:3M/（3.09M/3G）=2913MMIPS

浮点数：**IEEE754标准浮点格式**



阶码是增码表示无符号位。实际值需要减偏移修正。



**隐含尾数最高数位（20）**临时浮点数无隐含位

实际表达数是1.XX…XXX X个数为尾数位数

例：将(82.25)10 转换成短浮点数格式。

1）先将(82.25)10 转换成二进制数

(82.25)10 =(1010010.01)2

2）规格化二进制数(1010010.01)2

1010010.01=1.01001001×26

3）计算移码表示的阶码=偏置值+阶码真值：

(127+6)10=(133)10 =(10000101)2

4）以短浮点数格式存储该数

因此：符号位=0 表示该数为正数

阶码=10000101 由3）可得

尾数=01001001000000000000000 由2）可得   
 尾数为23位，不足在后面添15位0

所以，短浮点数代码为：

0；100 0010 1；010 0100 1000 0000 0000 0000

表示为十六进制代码为：42A48000H

例:IEEE754单精度浮点数:C0A00000H的十进制值是多少

(C0A00000)16 =(1 100,0000,1 010,0000, 0000, 0000,   
 0000,0000)2

可得, 符号位S是1

阶码位E是10000001→(10000001)2=(129)10

尾数M是: 010,0000, 0000,0000,0000,0000

→(0. 01000000000000000000000)2=(0.25)10

因此,由公式（-1）S ×1.M×2E-127得

（-1）1 ×1.25×2129-127=-1.25×4=-5

移位运算规则：

左移时右边最末位补0；右移时最高位补符号位。

左移有溢出问题：

正数首位为1，负数首位为0均会产生溢出，双符号位可解决。

双符号位溢出的判断：00正数无溢出 01正数正溢出 10 负数负溢出 11负数无溢出

首位表示数符，次位与首位不一致则溢出

舍入处理： 末位加1后右移； 新末位恒置1

n+1位定点整数/小数（1位符号位，n位数值位）的**补码**运算

**补码加减法：**

* 参与运算的操作数用补码表示，符号位作为数的一部分直接参与运算，所得运算结果即为补码表示形式。
* 若操作码为加，则两数直接相加。
* 若操作码为减，则将减数变补后再与被减数相加。

**补码乘法**：n+1步 加减法；n步移位。

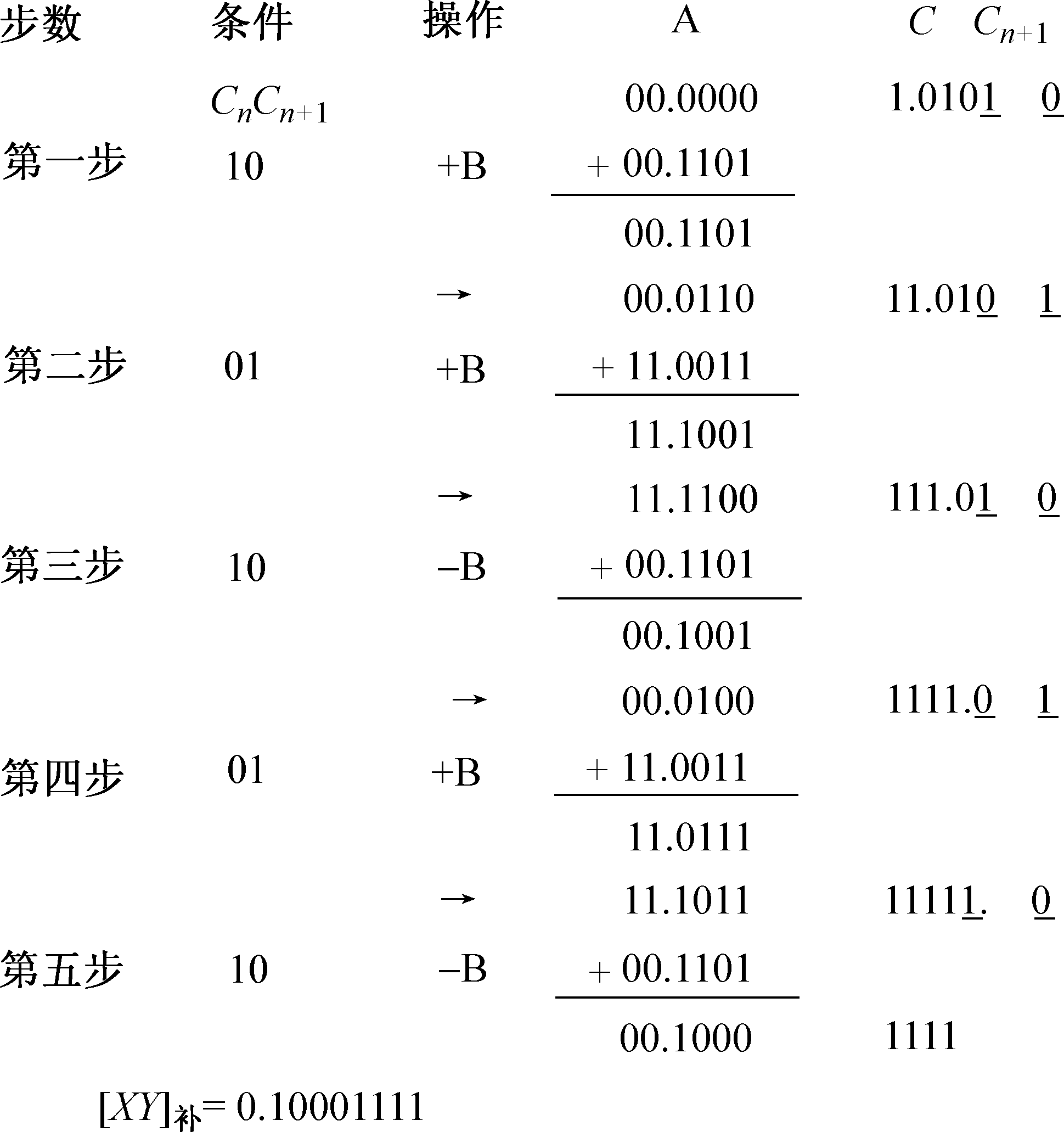
乘数最低位后加一个0；

C寄存器存放乘数Y **(Yn,Yn+1)**  00、11 不加不减（+0） 01+X补 10 -X补

n步移位 最后一步不移位—该步用于处理符号位

【例3-14】 *X* = 0.1101，*Y* = 0.1011，求[*XY*]补 = ？

设*A* = 00.0000，*B* = *X*补 = 11.0011，*B* = *X*补= 00.1101，*C* = *Y*补 = 1.0101。 C加上附加伟1.01010

 第一步是-B

则[*XY*]补=0.1000111。

**补码除法**：n步加减，n步移位

根据Sri和SY 同号商1 —> 2ri-Y 异号商0 —> 2ri +Y

最后求得的商需修正 +1.XX…1 X有n-1位

【例3-19】 *X*÷*Y* = 0.1000÷(0.1010)=？

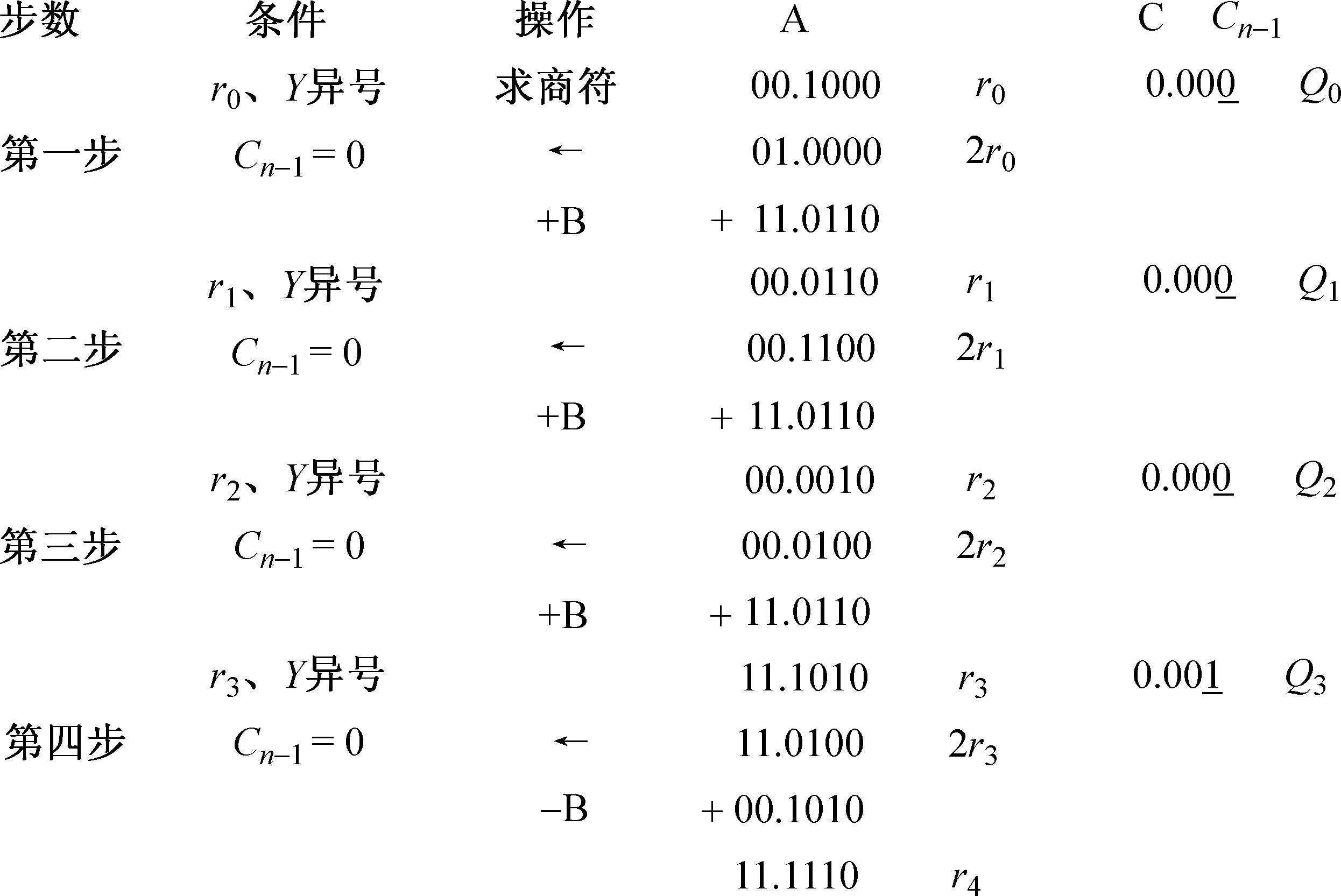
设A = *X*补= 00.1000，B = *Y*补= 11.0110，B = 00.1010，C = Q补= 0.0000。

1. 寄存器分配与符号位

用A寄存器存放被除数（X补码），以后存放余数，取双符号位。B寄存器存放除数（Y补码），双符号位，在算式中先写明B。C寄存器存放商，初始值为0（未考虑商符之前），单符号位。

1. 假商符

在第一步操作之前，先根据*r*0 (X)、*Y*符号比较确定假商符（与真商符相反）。

 最后一步r3、Y同号

假商= 0.001，真商= 0.001 + 1.0001 = 1.0011补 = 0.1101真值；

余数=24r4 = 1.11111110补 = 24×0.0010真值。

相关说明如下：

③ 基本操作

各步操作统一。根据假商符值决定第一步操作，并根据第一步操作结果决定第一位商值。如本例，假商符（*Q*0）为0，第一步做2*r*0+*B*，然后根据*r*1、*Y*异号上商（*Q*1）为0。

④ 步数

本例求4位商（尾数），所以做4步，但假商中只取3位商，第4位商通过校正（恒置1）获得。

⑤ 假商校正

本例中假商为0.001，含一位假商符与三位商值。校正（加1.0001）后得到真商1.0011，商符由0校正为实际值1，末位商恒为1。

⑥ 结果表达

补码相除后，商与余数自带符号（算式中最后一步求得的余数符号就是实际的余数符号），可正可负，可以分别写出商与余数的补码或真值形式。注意余数的权。

n+1位定点整数/小数（1位符号位，n位数值位）的**原码**运算

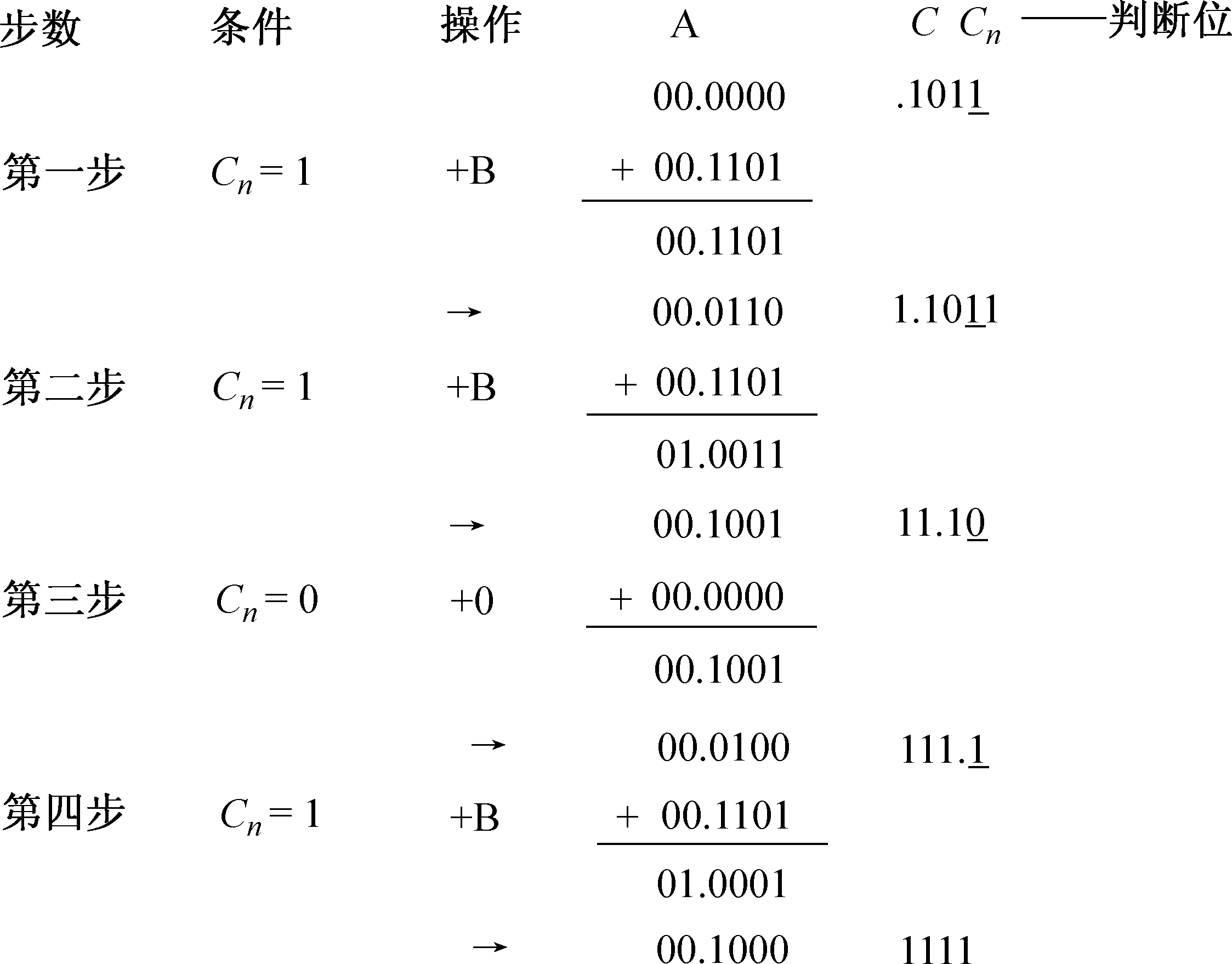
**原码乘法**： n次加法 n次移位

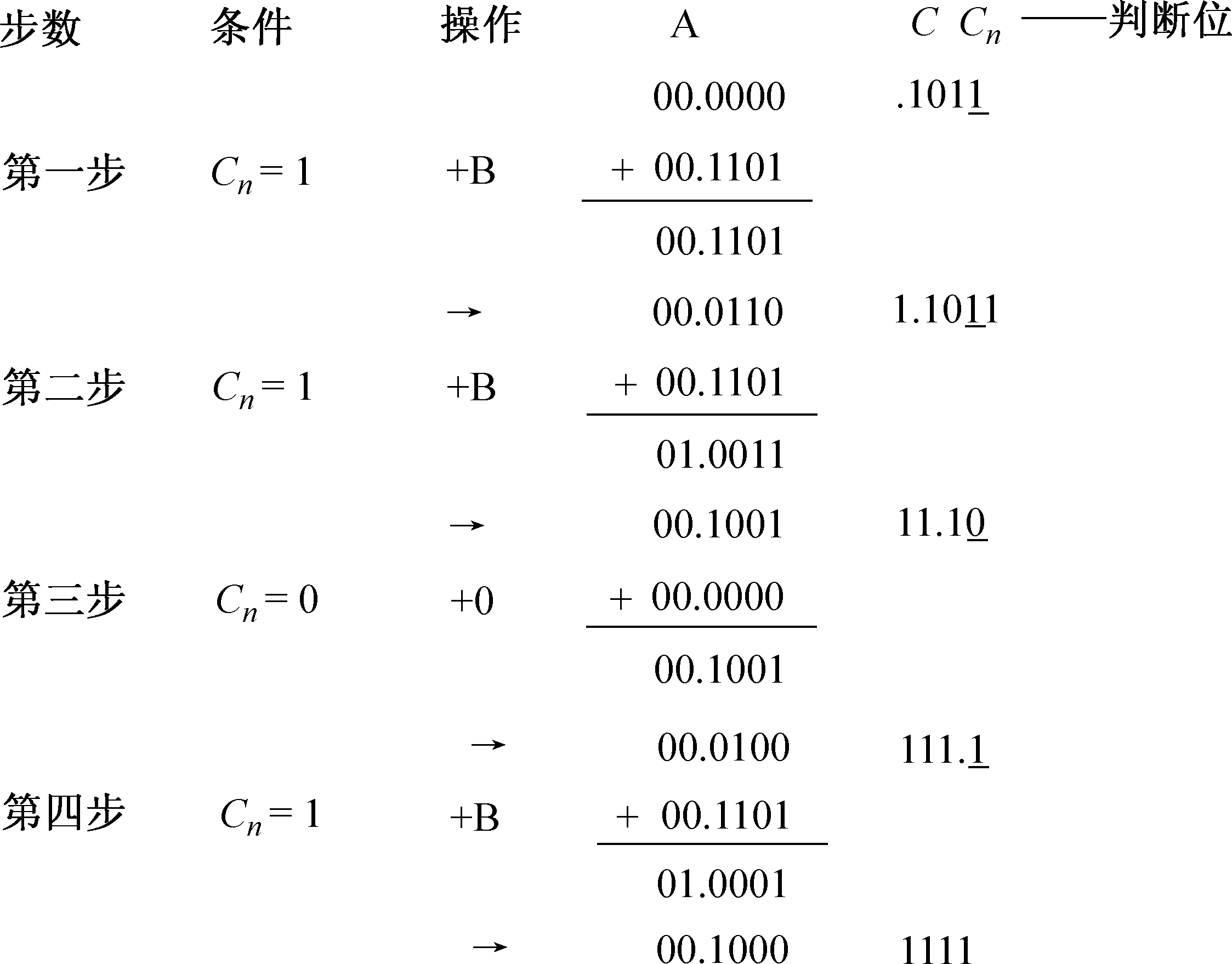
根据Cn 0 –> +0 1 –> +X 先加后右移

原码一位乘法是指：取两操作数的绝对值（即原码中的尾数）相乘，每次将一位乘数所对应的部分积与原部分积累加和相加，并右移一位。

【例3-13】 *X* = 0.1101，*Y* = 0.1011，求*XY* =？

设寄存器 *A* = 00. 0000，*B* =｜*X*｜= 00. 1101，*C* =｜*Y*｜= .1011。





加符号位，则乘积为1.10001111。

**原码除法**：n次移位，n或n+1次加减运算（恢复余数可能会增加一次加运算）

n位商 做n次运算 根据SA的符号决定加减 为0下一步2ri-Y、为1下一步2ri+Y，最后一步若为1，则需恢复余数rn+Y (n+1步加减)

① 取绝对值（原码尾数）相除，符号位单独处理。

② 对于定点小数除法，为使商不致溢出，要求被除数绝对值小于除数绝对值，即|*X*|＜|*Y*|。

③ 每步操作后，可根据余数*ri*符号判断是否够减：*ri*为正表明够减，上商*Qi*= 1；*ri*为负表明不够减，上商*Qi* = 0。

④ 基本操作可用通式描述为

 （3-14）

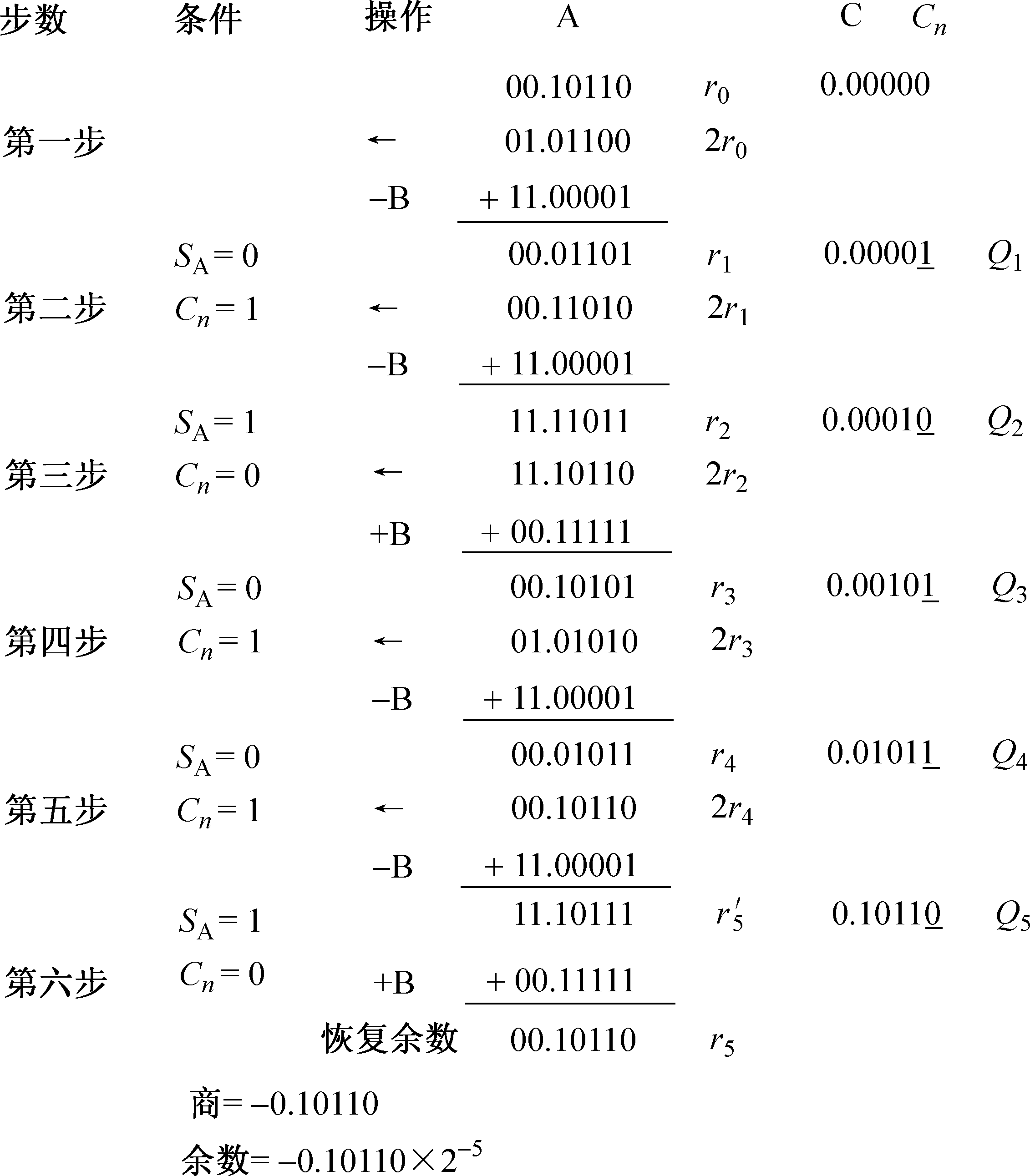
式（3-14）包含了两种情况：若第*i*步够减，*Qi* = 1，则第*i*+1步应做2*ri* -*Y*；若第*i*步不够减，*Qi* = 0，则第*i* +1步应做2*ri* + *Y*。

⑤ 原码除的思想是先当成正数相除，若最后一步所得余数为负，则应恢复余数，以保持*r*≥0。

##### （2）运算实例

【例3-16】 *X*÷*Y* = 0.10110÷0.11111 = ?

设*A* = |*X*| = 00.10110，*B* = |*Y*| = 00.11111，则*B* = 11.00001，*C* = |*Q*| = 0.00000。



则商= 0.10110，余数= 0.10110×25。

2．具有输入锁存器的运算器

如图3-26所示，CPU内部总线是一组双向传送的数据线，而寄存器组采用小规模高速存储器结构，每次由控制器发命令可选中某一单元，相当于选中一个寄存器进行读或写，也就是说，每次只能向ALU提供一个操作数。为了对双操作数进行运算操作，就需要在ALU输入端前设置一级锁存器，用来暂存操作数。例如，要实现R0+R1→R0操作，可通过内部总线先将R0的数据送入锁存器1，再通过内部总线将R1的数据送入锁存器2，或直接送加法器，然后相加，并将结果经内部总线送入R0。

寄存器组中也可包含若干暂存器，用来与系统总线相连接。例如，通过系统总线访问主存获得的数据可先暂时存放于暂存器之中，再通过内部总线送入ALU；运算结果也可先暂存于暂存器中，再通过系统总线送入主存储器。具体的结构有多种变化，可结合具体机器分析。

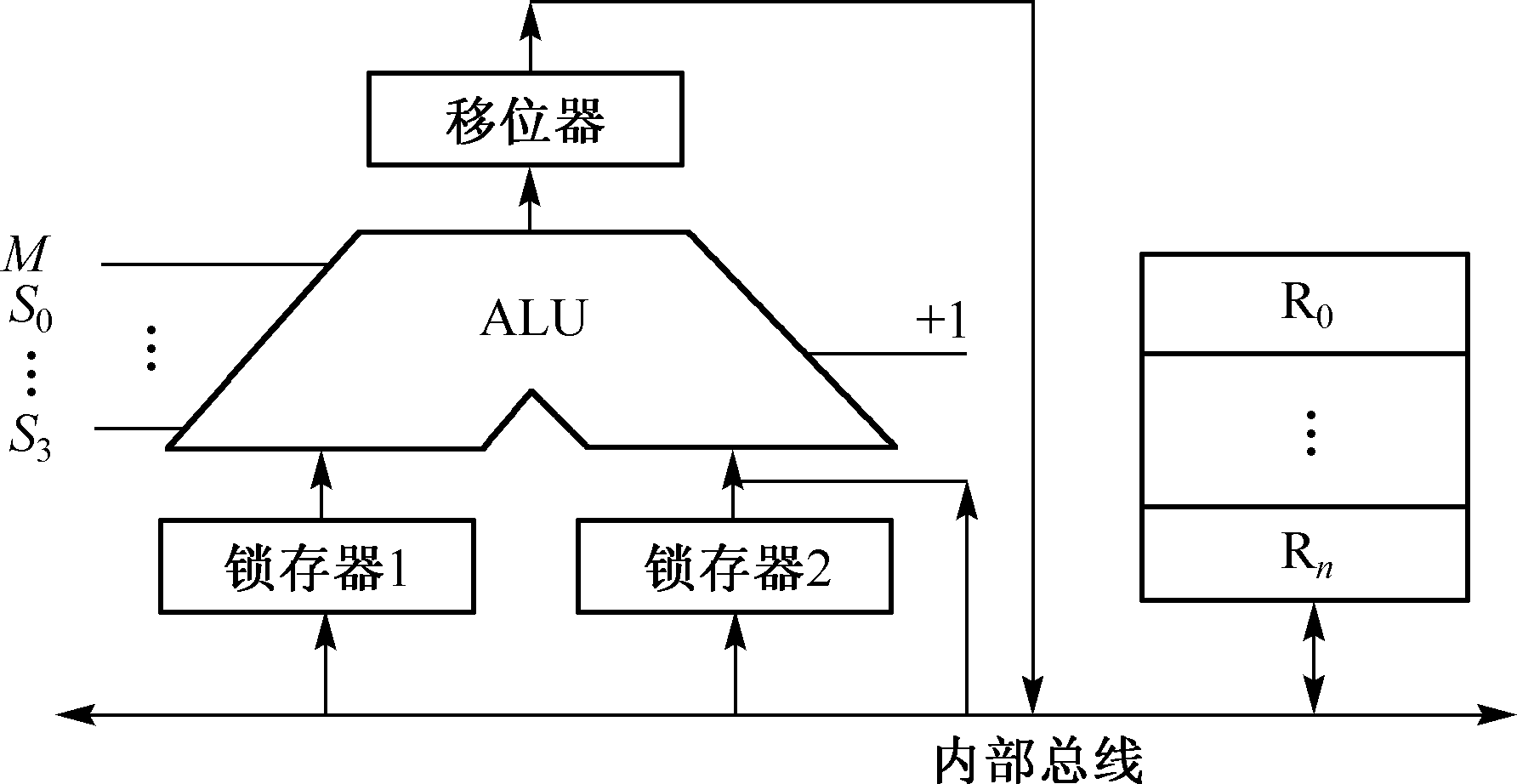


图3-26 具有输入锁存器的运算器组成

**变址寻址方式的应用例子**

假设要将主存200号~203号单元内容复制到500~503单元，主存按字编址。计算机即教材第三章的模型机。请用模型机指令系统编制实现上述复制功能的程序。

解 假设通用寄存器R2和R3分别放着主存地址B和A，则可用如下程序来实现将200~203号单元内容复到500~503单元。

|  |  |
| --- | --- |
| EOR *R*0，*R*0 | 实现 |
| A：MOV *X*(*R*0)，*X*(*R*0)； | 第一次执行时，将200单元内容传送到500单元。  第二次执行时，将201单元内容传送到501单元。  本指令在程序中执行4次，复制工作就全部完成。 |
| 1NC *R*0 ； | *R*0内容加1，即修改变址量（即位移量），计A地址指令执行次数 |
| MOV *R*1 *R*0 ； | 将R0传到R1保存 （下一条指令*R*0被修改） |
| SUB *R*0，(*PC*)+ ； | 相减，(*PC*)+是立即寻址，立即数是4。若相减后(R0)=0，则置特征位Z=1，否则Z=0 |
| JMP *R*2，Z ； | 判A单元指令是否执行了4次，是4次，Z为1转B，否则顺序执行 |
| MOV *R*0 *R*1 ； | 复制工作未全部完成，将A地址指令已执行次数送R0 |
| JMP *R*3 ； | 转A继续复制 |
| B：………………………； | 复制完成，执行下面程序 |

在程序中，MOV *X*(*R*0)，*X*(*R*0)，和INC *R*0两条指令是复制工作的核心，它们后面的几条指令是用于判别复制工作是否全部结束。

在模型机指令系统中，由于指令字长较短，放不下变址寻址的形式地址、立即寻址的立即数和直接寻址的直接地址，因此，将它们分别放在该指令的下一个单元。故本程序中MOV *X*(*R*0)，*X*(*R*0)，和SUB *R*0，(*PC*)+指令在主存中分别占用三个和两个字单元，示意如下：

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| A | MOV | *R*0号 | *X*(*R*) | *R*0号 | *X*(*R*) |  | SUB | *R*0号 | *R* | *PC*号 | (*R*)+ |
|  | 200（源寻址的形式地址） | | | | |  | 4（立即数） | | | | |
|  | 500（目的寻址的形式地址） | | | | |  |  | | | | |

首次执行A单元指令时，源寻址的有效地址=200+(*R*0)=200+0=200

目的寻址的有效地址=500+(*R*0)=500+0=500

于是第一次执行A单元指令时便将200号单元内容复制到500号单元，然后执行INC *R*0指令，*R*0内容加1，（即*R*0内容为1）。当第二次执行MOV *X*(*R*0)，*X*(*R*0)时，就进行201到501单元的复制工作，依此类推直到复制全部结束。

某**定点小数**字长16位，含一位符号，**原码**表示，分别写出下列典型值的二进制代码与十进制真值。

（1）非零最小正数 （2）最大正数

（3）绝对值最小负数 （4）绝大值最大负数

解：四种典型值用表3.2.2表示。

表3.2.2 定点小数原码典型值

|  |  |  |  |
| --- | --- | --- | --- |
|  | 原码 | 二进制真值 | 十进制真值 |
| (1)非零最小正数 | 0.000000000000001 | 0.000000000000001 |  |
| (2)最大正数 | 0.111111111111111 | 0.111111111111111 |  |
| (3)绝对值最小负数 | 1.000000000000001 | －0.000000000000001 |  |
| (4)绝对值最大负数 | 1.111111111111111 | －0.111111111111111 |  |

某**定点小数**字长16位，含一位符号，**补码**表示，分别写出下列典型值的二进制代码与十进制真值。

（1）非零最小正数 （2）最大正数

（3）绝对值最小负数 （4）绝大值最大负数

解：四种典型值用表3.2.3表示。

表3.2.3 定点小数补码典型值

|  |  |  |  |
| --- | --- | --- | --- |
|  | 补码 | 二进制真值 | 十进制真值 |
| (1)非零最小正数 | 0.000000000000001 | 0.000000000000001 |  |
| (2)最大正数 | 0.111111111111111 | 0.111111111111111 |  |
| (3)绝对值最小负数 | 1.111111111111111 | －0.000000000000001 |  |
| (4)绝对值最大负数 | 1.000000000000000 | －1 |  |

某**定点整数**字长16位，含一位符号，**补码**表示，分别写出下列典型值的二进制代码与十进制真值。

（1）非零最小正数 （2）最大正数

（3）绝对值最小负数 （4）绝大值最大负数

解：四种典型值用表3.2.4表示。

3.2.4 定点整数补码典型值

|  |  |  |  |
| --- | --- | --- | --- |
| 典型值 | 补码 | 二进制真值 | 十进制真值 |
| (1)非零最小正数 | 0000000000000001 | 1 | 1 |
| (2)最大正数 | 0111111111111111 | 111111111111111 |  |
| (3)绝对值最小负数 | 1111111111111111 | －1 |  |
| (4)绝对值最大负数 | 1000000000000000 | －1000000000000000 |  |

若采用图所示的**IEEE754短浮点数格式**，请将十进制数37.25写成浮点数，并且写出其二进制代码序列。

IEEE754短浮点数格式为：

31 30 23 22 0

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| S0 |  | …… |  |  | …… |  |

数符 阶码E 尾数M

解：将十进制数37.25转换为二进制数100101.01，按IEEE754标准的短实数浮点格式要求，将100101.01表示为1.0010101×25，故浮点数阶码的真值e=5。于是，按IEEE754标准，得到：

数符S0=0，阶码（移码表示）E=（e+127）10=（5+127）10=（132）10=（10000101）2，M=001010100000…00。

最后得到32位浮点数的二进制数代码序列为：

01000010100101010000000000000000

某主存储器部分单元的地址码与存储器内容对应关系如下：

|  |  |
| --- | --- |
| 地址码 | 存储内容 |
| 1000H | A307H |
| 1001H | 0B3FH |
| 1002H | 1200H |
| 1003H | F03CH |
| 1004H | D024H |

（1）若采用寄存器间址方式读取操作数，指定寄存器*R*0的内容为1002H，则操作数是多少？

答：操作数是1200H。

（2）若采用自增型寄存器间址方式(R0)+读取操作数，*R*0内容为1000H，则操作数是多少？指令执行后*R*0的内容是多少？

答：操作数是A307H，指令执行后*R*0的内容变为1001H

（3）若采用自减型寄存器间址方式读取操作数，R1内容为1003H，则操作数是多少？指令执行后R1的内容是多少？

答：操作数是1200H，指令执行后R1的内容为1002H。

（4）若采用变址寻址方式读取操作数，指令中给出形式地址d=3H，变址寄存器R2内容为1000H，则操作数是多少？

答：操作数为F03CH。

用**双符号位补码**计算X补+Y补=？并指出是否有溢出。

（1）X补=00.110011 Y补=00.101101

（2）X补=00.010110 Y补=00.100101

（3）X补=11.110011 Y补=11.101101

（4）X补=11.001101 Y补=11.010011

解：作补码加法时，将两个操作数直接相加。对于双符号位补码而言，溢出=第一符号位⊕第二符号位，因此可以根据运算结果的两个符号位相同或相异，判断是否发生了溢出以及溢出的类型。

（1） 00.110011

+ 00.101101 结果的两个符号位相异，表明发生了溢出；又由于第一

01.100000 符号位为0表示结果为正，所以发生了正溢出。

（2） 00.010110

+ 00.100101 结果的两个符号位相同，为00，表明未发生溢出，结

00.111011 果为正。

（3） 11.110011

+ 11.101101 结果的两个符号位相同，为11，表明未发生溢出，结

11.100000 果为负。

（4） 11.001101

+ 11.010011 结果的两个符号位相异，表明发生了溢出；又由于第一

10.100000 符号位为1表示结果为负，所以发生了负溢出。

用**双符号位补码**计算X补-Y补=？并指出是否有溢出。

（1）X补=00.100011 Y补=00.101101

（2）X补=00.110110 Y补=11.010011

（3）X补=11.100011 Y补=00.110100

（4）X补=11.101101 Y补=11.010011

解：作补码减法时，将减数Y补变补，再与被减数X补相加。需要注意的是，变补是将Y补的尾数连同符号位一起变补。

（1）将Y补变补，即-Y补=11.010011

00.100011

+ 11.010011 结果的两个符号位相同，为11，表明未发生溢出，结

11.110110 果为负。

（2）-Y补=00.101101

00.110110

+ 00.101101 结果的两个符号位相异，表明发生了溢出；又由于第一

01.100011 符号位为0表示结果为正，所以发生了正溢出。

（3）-Y补=11.001100

11.100011

+ 11.001100 结果的两个符号位相异，表明发生了溢出；又由于第一

10.101111 符号位为1表示结果为负，所以发生了负溢出。

（4）-Y补=00.101101

11.101101

+ 00.101101 结果的两个符号位相同，为00，表明未发生溢出，结

00.011010 果为正。

**浮点加减运算流程**

浮点加减运算流程包括判操作数是否为0、对阶、尾数加减、结果规格化等几个阶段，其中，对阶和结果规格化是重要的基本概念。

（1）对阶

需要掌握的内容有：对阶的定义、原因、原则、操作等。

●什么是对阶？

对阶是指让两个浮点数的阶码相同。

●为什么要对阶？

对阶用在浮点数的加减运算中。两个浮点数相加减，是将它们的尾数相加减，但必须首先保证其小数点的位置对齐，即它们对应的数位的权值要相同。例如，两个浮点数22×0.1001和23×0.1101，它们的小数点的实际位置是：

10.01

110.1

也就是说，两个浮点数的阶码不同，其尾数的小数点的实际位置是没有对齐的，尾数对应位的权值不同，因而不能直接加减。对齐小数点的位置，使它们成为：

010.01

110.1

写成浮点数形式，即23×0.0101和23×0.1101。这样，两数阶码相同，尾数便可以直接加减了。

●如何对阶？

对阶是将大阶码减小到与小阶码相同？还是将小阶码增大到与大阶码相同？或者任意方式均可？这是一个原则问题。阶码增大或减小，相应地尾数应右移或左移，才能使浮点数的值不变。由于小阶增大，尾数右移，舍去的是尾数的低位，误差很小；而大阶减小，尾数左移，舍去的是尾数的高位，误差太大，因而是不允许的。所以，对阶的原则是：小阶向大阶对齐。对阶操作是：将小阶加1，尾数右移一位，直到小阶与大阶相等。

（2）结果规格化

需要掌握的内容有：规格化的定义、左规和右规的判断、左规和右规操作等。

●什么是规格化？

规格化是针对浮点数尾数而言。浮点数尾数的绝对值大于或等于1/2而小于1，这样的浮点数称为规格化浮点数。从代码形式来看，即尾数最高位的绝对值为1。对于正数补码（包括正、负数原码），尾数最高位为1；对于负数补码，尾数最高位为0。

●在什么情况下进行左规或右规？

当尾数的绝对值小于1/2，即尾数的符号位和最高位同为0.0或1.1时，进行左移规格化，将尾数左移一位，阶码减1，直到尾数的绝对值大于或等于1/2。当尾数的绝对值大于1，即尾数的两个符号位不同（01或10）时，进行右移规格化，将尾数右移一位，第二符号位中存放的进位移至尾数最高位，使尾数的绝对值小于1，并且阶码加1。

用74181和74182芯片构成一个**64位ALU**，采用分级分组并行进位链结构。画出逻辑图，并注明输入、输出信号。

解：首先计算74181芯片的数量。由于1片74181实现4位数运算，所以64位数需要16片74181。

其次计算74182芯片的数量。1片74182只能接受4个小组的进位辅助函数，并同时输出4个组间进位信号。因此将16片74181分为4组，每组4片，组内共用1片74182，同时产生组内各74181的初始进位。4片74182再共用另1片74182，同时产生4个组间进位信号。所以一共需要5片74182。

ALU逻辑图如下：

74182

74182

74182

74182

74182

74181

74181

74181

74181

74181

74181

74181

74181

·· ·· ·· ··

**图3.3.1 ALU逻辑图**

用若干片SN74181可以方便地构成更多位数的ALU部件。片内已实现组内并行进位，如果采取组间串行进位结构，只需将几片SN74181简单级连，即将各片的进位输出*Cn*+4送往高位芯片的进位输入端*Cn*，如图3-23所示。

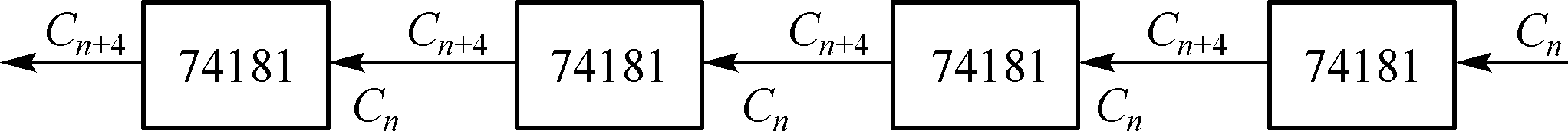


图3-23 组间串行进位的ALU

如果采用组间并行进位结构，需增加并行进位链芯片SN74182。图3-24提供了一个16位ALU连接实例。74181输出的小组进位产生函数*G*与进位传递函数*P*可作为并行进位链74182的输入，而74182则向各74181提供组间进位信号。74182的输出（图中的、）还可支持更高一级的并行进位链，从而可构造更长位数的ALU。

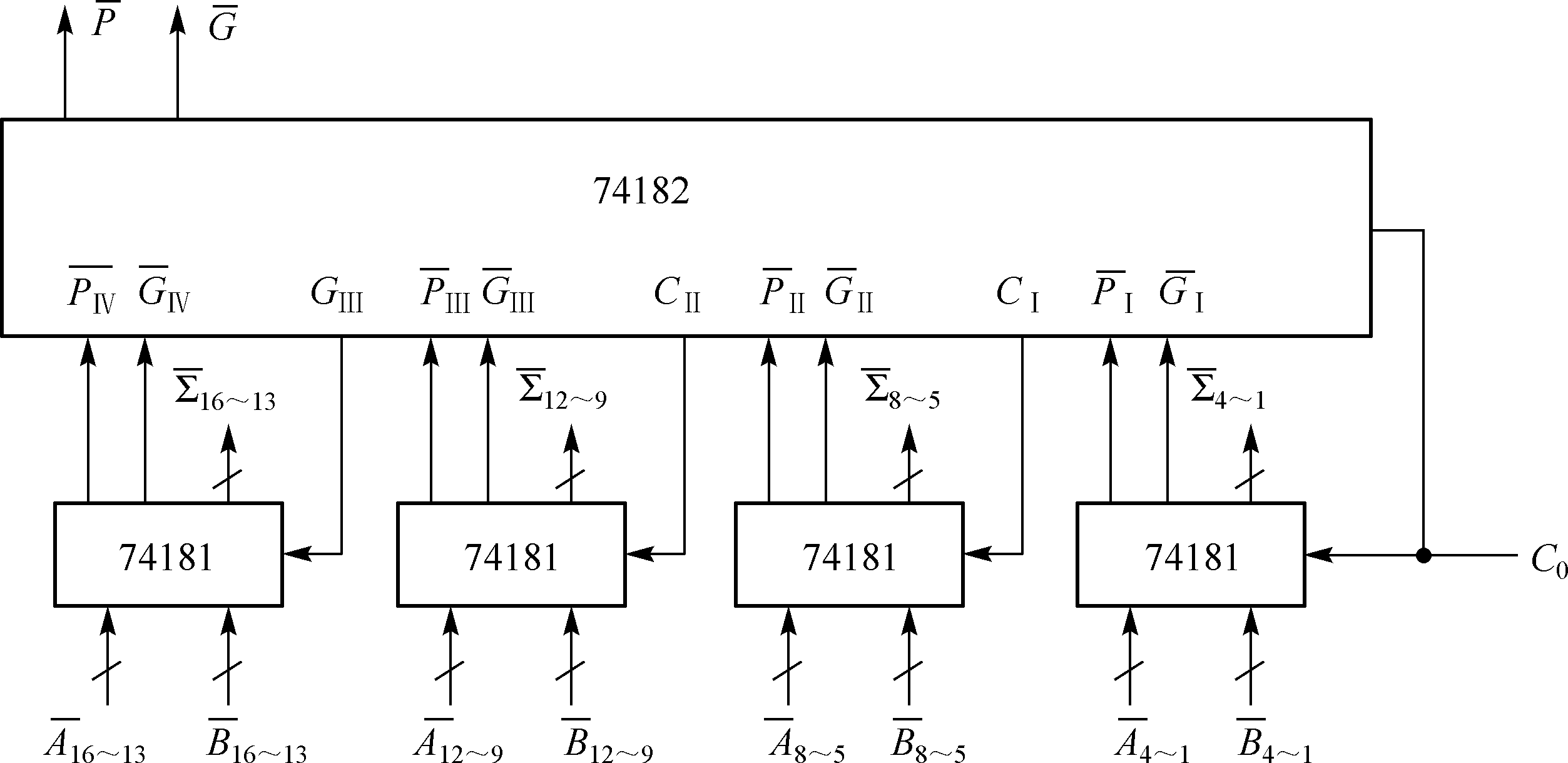


图3-24 组间并行进位的ALU

习题3-6、根据模型机数据通路结构，拟定MOV指令流程在ST2、ST3、ST4中的操作时间表。

解：按图3-36 MOV指令流程图，ST2完成一次内部数据通路操作，根据ALU输入选择、ALU功能选择、输出移位选择、结果分配等四段操作设置相应的微命令。ST2结束后，若寻址方式为自增型寄存器间址，则进入目的周期DT或执行周期ET；若寻址方式为自增型双间址或变址，则继续ST，进入ST3，完成一次内部数据通路操作，然后进入ST4，完成一次从存储器读出，并经内部数据通路传送的操作。ST2、ST3、ST4中的操作时间表如下。

**表3.3.3 MOV指令源周期操作时间表**

电位型微命令 脉冲型微命令

ST2： Ri→A CP Ri

SP→A CP SP

PC→A CP PC

A+1 CP T (P)

DM CP FT (P)

1→ST [@(R)+ ∨ X(R)] CP ST (P)

T+1 [@(R)+ ∨ X(R)] CP DT (P)

1→DT [ (R)+] DR CP ET (P)

1→ET [ (R)+] DR

ST3: C→A CP MAR

Ri→B CP T (P)

PC→B

输出A [@(R)+]

A+B [X(R)]

DM

T+1

ST4： EMAR CP C

R CP T (P)

SMDR CP FT (P)

MDR→B CP ST (P)

输出B CP DT (P)

DM CP ET (P)

1→DT 或

1→ET

注：上述操作时间表中基本上未考虑逻辑条件。

**习题3-7**、拟出下述指令流程及操作时间表。

（1）MOV（R0），（SP）+；

（2）MOV（R1）+，X（R0）；

（3）MOV R2，（PC）+；

（4）MOV -（SP），（R3）；

（5）ADD R1，X（R0）；

（6）SUB（R1）+，（R2）；

（7）AND -（R0），R1；

（8）OR R2，（R0）+；

（9）EOR（R0），（R1）；

（10）INC X（PC）；

（11）DEC（R0）；

（12）COM（R1）+

（13）NEG -（R2）；

（14）SL R0；

（15）SR R3；

（16）JMP SKP；

（17）JMP R0；

（18）JMP X（PC）；

（19）RST（SP）+；

（20）JSR（R1）；

解：拟定指令流程关键是要了解模型机各类指令功能，掌握模型机各种寻址方式，熟悉模型机数据通路结构。此外，还需清楚指令中源地址和目的地址的位置，教材中约定，源在后，目的在前。

（1）MOV（R0），（SP）+；

该指令完成出栈操作。需要注意的是，在操作时间表中，当进行周期转换时，5个打入命令都发，以便简化逻辑条件。另外，为简单起见，没有分开列出电位型微命令和脉冲型微命令。

1）指令流程 2）操作时间表

FT： M→IR FT：EMAR、R、SIR、

PC+1→PC PC→A、A+1、DM、CPPC、1→ST、CPT（P）、CPFT（P）、

CPST（P）、CPDT（P）、CPET（P）；

ST： SP→MAR ST：SP→A、输出A、DM、CPMAR、T+1、CPT（P），

M→MDR→C EMAR、R、SMDR、MDR→B、输出B、DM、CPC、T+1、CPT（P），

SP+1→SP SP→A、A+1、DM、CPSP、1→DT、CPT（P）、CPFT（P）、

CPST（P）、CPDT（P）、CPET（P）；

DT： R0→MAR DT：R0→A、输出A、DM、CPMAR、1→ET、CPT（P）、CPFT（P）、

CPST（P）、CPDT（P）、CPET（P）；

ET： C→MDR ET：C→A、输出A、DM、CPMDR、T+1、CPT（P），

MDR→M EMAR、W、T+1、CPT（P），

PC→MAR PC→A、输出A、DM、CPMAR、1→FT、CPT（P）、CPFT（P）、

CPST（P）、CPDT（P）、CPET（P）。

（2）MOV（R1）+，X（R0）；

该指令将采用变址方式获得的源操作数送入目的单元。由于模型机指令字长有限，变址所需的形式地址放在现行指令所在单元的下一个单元中，取指后由PC指示。

1）指令流程 2）操作时间表

FT： M→IR FT：微命令同前；

PC+1→PC

ST： PC→MAR ST：PC→A、输出A、DM、CPMAR、T+1、CPT（P），

M→MDR→C EMAR、R、SMDR、MDR→B、输出B、DM、CPC、T+1、CPT（P），

C+ R0→MAR C→A、R0→B、A+B、DM、CPMAR、T+1、CPT（P），

M→MDR→C EMAR、R、SMDR、MDR→B、输出B、DM、CPC、T+1、CPT（P），

PC+1→PC PC→A、A+1、DM、CPPC、1→DT、CPT（P）、CPFT（P）、

CPST（P）、CPDT（P）、CPET（P）。

DT： R1→MAR DT：R1→A、输出A、DM、CPMAR、T+1、CPT（P），

R1+1→R1 R1→A、A+1、DM、CP R1、1→ET、CPT（P）、CPFT（P）、

CPST（P）、CPDT（P）、CPET（P）；

ET： C→MDR ET：微命令同前，

MDR→M 微命令同前，

PC→MAR 微命令同前。

（3）MOV R2，（PC）+；

该条指令的源采用立即寻址，指令的功能是将立即数送入R2。与变址中的形式地址一样，16位立即数存放在现行指令所在单元的下一个单元中，取指后由PC指示。

1）指令流程 2）操作时间表

FT： M→IR FT：微命令同前；

PC+1→PC

ST： PC→MAR ST：PC→A、输出A、DM、CPMAR、T+1、CPT（P），

M→MDR→C EMAR、R、SMDR、MDR→B、输出B、DM、CPC、T+1、CPT（P），

PC+1→PC PC→A、A+1、DM、CPPC、1→ET、CPT（P）、CPFT（P）、

CPST（P）、CPDT（P）、CPET（P）；

ET： C→R2 ET：C→A、输出A、DM、CPR2、T+1、CPT（P），

PC→MAR PC→A、输出A、DM、CPMAR、1→FT、CPT（P）、CPFT（P）、

CPST（P）、CPDT（P）、CPET（P）。

（4）MOV -（SP），（R3）；

该指令执行入栈操作。

1）指令流程 2）操作时间表

FT： M→IR FT：微命令同前；

PC+1→PC

ST： R3→MAR ST：R3→A、输出A、DM、CPMAR、T+1、CPT（P），

M→MDR→C EMAR、R、SMDR、MDR→B、输出B、DM、CPC、1→DT、

CPT（P）、CPFT（P）、CPST（P）、CPDT（P）、CPET（P）；

DT：SP-1→SP、MAR DT：SP→A、A-1、DM、CPSP、CPMAR、1→ET、CPT（P）、

CPFT（P）、CPST（P）、CPDT（P）、CPET（P）；

ET： C→MDR ET：微命令同前，

MDR→M 微命令同前，

PC→MAR 微命令同前。

（5）ADD R1，X（R0）；

该条指令的功能是将变址获得的源操作数与目的操作数相加，结果存放在目的地址R1中。

1）指令流程 2）操作时间表

FT： M→IR FT：微命令同前；

PC+1→PC

ST： PC→MAR ST：微命令同前，

M→MDR→C 微命令同前，

C+ R0→MAR 微命令同前，

M→MDR→C 微命令同前，

PC+1→PC PC→A、A+1、DM、CPPC、1→ET、CPT（P）、CPFT（P）、

CPST（P）、CPDT（P）、CPET（P）；

ET： C+ R1→R1 ET：C→A、R1→B、A+B、DM、CPR1、T+1、CPT（P），

PC→MAR PC→A、输出A、DM、CPMAR、1→FT、CPT（P）、CPFT（P）、

CPST（P）、CPDT（P）、CPET（P）。

（6）SUB（R1）+，（R2）；

该指令的功能是用源操作数减去目的操作数，结果存放在目的单元中。

1）指令流程 2）操作时间表

FT： M→IR FT：微命令同前；

PC+1→PC

ST： R2→MAR ST：R2→A、输出A、DM、CPMAR、T+1、CPT（P），

M→MDR→C EMAR、R、SMDR、MDR→B、输出B、DM、CPC、1→DT、

CPT（P）、CPFT（P）、CPST（P）、CPDT（P）、CPET（P）；

DT： R1→MAR DT：R1→A、输出A、DM、CPMAR、T+1、CPT（P），

M→MDR→D EMAR、R、SMDR、MDR→B、输出B、DM、CPD、T+1、CPT（P），

R1+1→R1 R1→A、A+1、DM、CPR1、1→ET、CPT（P）、CPFT（P）、

CPST（P）、CPDT（P）、CPET（P）；

ET： C-D→MDR ET：C→A、D→B、A-B、DM、CPMDR、T+1、CPT（P），

MDR→M EMAR、W、T+1、CPT（P），

PC→MAR PC→A、输出A、DM、CPMAR、1→FT、CPT（P）、CPFT（P）、

CPST（P）、CPDT（P）、CPET（P）。

（7）AND -（R0），R1；

该指令将两数相与。由于源采用寄存器寻址，取指后进入目的周期。

1）指令流程 2）操作时间表

FT： M→IR FT：只需将1→ST改为1→DT，其余微命令同前；

PC+1→PC

DT： R0-1→R0、MAR DT：R0→A、A-1、DM、CPR0、CPMAR、T+1、CPT（P），

M→MDR→D EMAR、R、SMDR、MDR→B、输出B、DM、CPD、1→ET、

CPT（P）、CPFT（P）、CPST（P）、CPDT（P）、CPET（P）；

ET： R1∧ D→MDR ET：R1→A、D→B、A∧B、DM、CPMDR、T+1、CPT（P），

MDR→M EMAR、W、T+1、CPT（P），

PC→MAR 微命令同前。

（8）OR R2，（R0）+；

1）指令流程 2）操作时间表

FT： M→IR FT：微命令同前；

PC+1→PC

ST： R0→MAR ST：R0→A、输出A、DM、CPMAR、T+1、CPT（P），

M→MDR→C EMAR、R、SMDR、MDR→B、输出B、DM、CPC、T+1、CPT（P），

R0+1→R0 R0→A、A+1、DM、CPR0、1→ET、CPT（P）、CPFT（P）、

CPST（P）、CPDT（P）、CPET（P）；

ET： C∨R2→R2 C→A、R2→B、A∨B、DM、CPR2、T+1、CPT（P），

PC→MAR 微命令同前。

（9）EOR（R0），（R1）；

1）指令流程 2）操作时间表

FT： M→IR FT：微命令同前；

PC+1→PC

ST： R1→MAR ST：R1→A、输出A、DM、CPMAR、T+1、CPT（P），

M→MDR→C EMAR、R、SMDR、MDR→B、输出B、DM、CPC、1→DT、

CPT（P）、CPFT（P）、CPST（P）、CPDT（P）、CPET（P）；

DT： R0→MAR DT：R0→A、输出A、DM、CPMAR、T+1、CPT（P），

M→MDR→D EMAR、R、SMDR、MDR→B、输出B、DM、CPD、1→ET、

CPT（P）、CPFT（P）、CPST（P）、CPDT（P）、CPET（P）；

ET： C EOR D→MDR ET：C→A、D→B、A EOR B、DM、CPMDR、T+1、CPT（P），

MDR→M 微命令同前，

PC→MAR 微命令同前。

（10）INC X（PC）；

该加1指令采用相对寻址，位移量放在现行指令所在单元的下一单元中，取指后由PC指示。

1）指令流程 2）操作时间表

FT： M→IR FT：EMAR、R、SIR、PC→A、A+1、DM、CPPC、1→DT、

PC+1→PC CPT（P）、CPFT（P）、CPST（P）、CPDT（P）、CPET（P）；

DT：PC→MAR DT：PC→A、输出A、DM、CPMAR、T+1、CPT（P），

M→MDR→D EMAR、R、SMDR、MDR→B、输出B、DM、CPD、T+1、CPT（P），

PC+D→MAR PC→A、D→B、A+B、DM、CPMAR、T+1、CPT（P），

M→MDR→D EMAR、R、SMDR、MDR→B、输出B、DM、CPD、T+1、CPT（P），

PC+1→PC PC→A、A+1、DM、CPPC、1→ET、CPT（P）、CPFT（P）、

CPST（P）、CPDT（P）、CPET（P）；

ET： D+1→MDR ET：D→A、A+1、DM、CPMDR、T+1、CPT（P），

MDR→M 微命令同前，

PC→MAR 微命令同前。

注意，在模型机的相对寻址中，基准地址PC是指向位移量，而不是指向下条指令，因此在DT中，PC+1→PC的操作应在变址计算PC+D→MAR的操作之后进行。

（11）DEC（R0）；

1）指令流程 2）操作时间表

FT： M→IR FT：微命令同前；

PC+1→PC

DT： R0→MAR DT：R0→A、输出A、DM、CPMAR、T+1、CPT（P），

M→MDR→D EMAR、R、SMDR、MDR→B、输出B、DM、CPD、1→ET、

CPT（P）、CPFT（P）、CPST（P）、CPDT（P）、CPET（P）；

ET： D-1→MDR ET：D→A、A-1、DM、CPMDR、T+1、CPT（P），

MDR→M 微命令同前，

PC→MAR 微命令同前。

（12）COM（R1）+

1）指令流程 2）操作时间表

FT： M→IR FT：微命令同前；

PC+1→PC

DT： R1→MAR DT：R1→A、输出A、DM、CPMAR、T+1、CPT（P），

M→MDR→D EMAR、R、SMDR、MDR→B、输出B、DM、CPD、T+1、CPT（P），

R1+1→R1 R1→A、A+1、DM、CPR1、1→ET、CPT（P）、CPFT（P）、

CPST（P）、CPDT（P）、CPET（P）；

ET： D→MDR ET：D→A、输出A、DM、CPMDR、T+1、CPT（P），

MDR→M 微命令同前，

PC→MAR 微命令同前。

（13）NEG -（R2）；

1）指令流程 2）操作时间表

FT： M→IR FT：微命令同前；

PC+1→PC

DT： R2-1→R2、MAR DT：R2→A、A-1、DM、CPR2、CPMAR、T+1、CPT（P），

M→MDR→D EMAR、R、SMDR、MDR→B、输出B、DM、CPD、1→ET、

CPT（P）、CPFT（P）、CPST（P）、CPDT（P）、CPET（P）；

ET： D+1→MDR ET：D→A、A+1、DM、CPMDR、T+1、CPT（P），

MDR→M 微命令同前，

PC→MAR 微命令同前。

（14）SL R0；

1）指令流程 2）操作时间表

FT： M→IR FT：将1→DT改为1→ET，其余微命令同前；

PC+1→PC

ET： R0→R0 ET：R0→A、输出A、DM、CPR0、T+1、CPT（P），

PC→MAR 微命令同前。

（15）SR R3；

1）指令流程 2）操作时间表

FT： M→IR FT：微命令同前；

PC+1→PC

ET： R3→R3 ET：R3→A、输出A、DM、CPR3、T+1、CPT（P），

PC→MAR 微命令同前。

（16）JMP SKP；

1）指令流程 2）操作时间表

FT： M→IR FT：微命令同前；

PC+1→PC

ET： PC+1→PC、MAR ET：PC→A、A+1、DM、CPPC、CPMAR、1→FT、CPT（P）、

CPFT（P）、CPST（P）、CPDT（P）、CPET（P）。

（17）JMP R0；

1）指令流程 2）操作时间表

FT： M→IR FT：微命令同前；

PC+1→PC

ET： R0→PC、MAR ET：R0→A、输出A、DM、CPPC、CPMAR、1→FT、CPT（P）、

CPFT（P）、CPST（P）、CPDT（P）、CPET（P）。

（18）JMP X（PC）；

1）指令流程 2）操作时间表

FT： M→IR FT：微命令同前；

PC+1→PC

ET： PC→MAR ET：PC→A、输出A、DM、CPMAR、T+1、CPT（P），

M→MDR→C EMAR、R、SMDR、MDR→B、输出B、DM、CPC、T+1、CPT（P），

PC+C→PC、MAR PC→A、C→B、A+B、DM、CPPC、CPMAR、1→FT、CPT（P）、

CPFT（P）、CPST（P）、CPDT（P）、CPET（P）。

注意，转移指令采用相对寻址时，在作了变址计算后，不再作PC+1→PC的操作。这是因为转移指令执行后要转到转移地址去执行目标指令，而不是顺序执行紧跟转移指令的下条指令。

（19）RST（SP）+；

1）指令流程 2）操作时间表

FT： M→IR FT：微命令同前；

PC+1→PC

ET： SP→MAR ET：SP→A、输出A、DM、CPMAR、T+1、CPT（P），

SP+1→SP SP→A、A+1、DM、CPSP、T+1、CPT（P），

M→MDR→PC、MAR EMAR、R、SMDR、MDR→B、输出B、DM、CPPC、CPMAR、

1→FT、CPT（P）、CPFT（P）、CPST（P）、CPDT（P）、

CPET（P）。

（20）JSR（R1）；

转子指令在取指后进入源周期获取子程序入口，再进入执行周期保存断点，并转到子程序入口。

1）指令流程 2）操作时间表

FT： M→IR FT：将1→ET改为1→ST，其余微命令同前；

PC+1→PC

ST： R1→MAR ST：R1→A、输出A、DM、CPMAR、T+1、CPT（P），

M→MDR→C EMAR、R、SMDR、MDR→B、输出B、DM、CPC、1→ET、

CPT（P）、CPFT（P）、CPST（P）、CPDT（P）、CPET（P）；

ET：SP-1→SP、MAR ET：SP→A、A-1、DM、CPSP、CPMAR、T+1、CPT（P），

PC→MDR PC→A、输出A、DM、CPMDR、T+1、CPT（P），

MDR→M EMAR、W、T+1、CPT（P），

C→PC、MAR C→A、输出A、DM、CPPC、CPMAR、1→FT、CPT（P）、

CPFT（P）、CPST（P）、CPDT（P）、CPET（P）。

该题也可以采用另一种作法，即取指后进入执行周期，先保存断点，再获取子程序入口并转该入口。相应的指令流程如下：

FT： M→IR

PC+1→PC

ET： SP-1→SP、MAR

PC→MDR

MDR→M

R1→MAR

M→MDR→PC、MAR

习题3-8、拟出中断周期IT中各拍的操作时间表。

解：假设采用模型机中断机制。当CPU响应中断后，向中断控制器发出中断响应信号INTA，并进入中断周期。中断控制器收到INTA后，将被批准的中断源的中断号送往数据总线，以便CPU接收。在中断周期中，CPU关中断，保存断点，将中断号乘以2，作为向量地址访问向量表，获得中断服务程序入口地址后，转入服务程序。

中断周期流程和操作时间表如下：

1）IT流程 2）操作时间表

IT： 关中断 IT：0→I、

SP-1→SP、MAR SP→A、A-1、DM、CPSP、CPMAR、T+1、CPT（P），

PC→MDR PC→A、输出A、DM、CPMDR、T+1、CPT（P），

MDR→M EMAR、W、T+1、CPT（P），

8259→MDR→MAR IOR、SMDR、MDR→B、输出B、左移、CPMAR、T+1、CPT（P），

M→MDR→PC、MAR EMAR、R、SMDR、MDR→B、输出B、DM、CPPC、CPMAR、

1→FT、CPT（P）、CPFT（P）、CPST（P）、CPDT（P）、

CPET（P）。

说明：在关中断操作中，发0→I命令，将模型机程序状态字PSW中的允许中断位I清零。在8259→MDR→MAR操作中，发IOR命令，从8259读出中断号，经MDR、ALU传送，并将ALU输出的结果左移一位，即获得向量地址（中断号乘以2）。

习题3-9、编写取目的地址微子程序（从60H单元开始）。

解：取目的地址微子程序与取源操作数微子程序类似，所不同的是，前者只取出目的地址，而不取目的操作数。微子程序如下表。

**表3.3.4 取目的地址微子程序**

微地址 操作 微命令

取目的地址 60 按目的寻址方式分支，SC=0110

R 61 返回，SC=1000

（R） 62 Rj→MAR Rj→A，输出A，DM，CPMAR，返回，SC=1000

-（R） 63 Rj-1→Rj Rj→A，A-1，DM，CPRj，SC=0000

64 转62，SC=0001

（R）+ 65 Rj→MAR Rj→A，输出A，DM，CPMAR，SC=0000

66 Rj+1→Rj Rj→A，A+1，DM，CPRj，返回，SC=1000

@（R）+ 67 Rj→MAR Rj→A，输出A，DM，CPMAR，SC=0000

68 M→MDR→MAR EMAR，R，SMDR，MDR→B，输出B，DM，CPMAR，SC=0000

69 转66，SC=0001

X（R） 6A PC→MAR PC→A，输出A，DM，CPMAR，SC=0000

6B PC+1→PC PC→A，A+1，DM，CPPC，SC=0000

6C M→MDR→D EMAR，R，SMDR，MDR→B，输出B，DM，CPD，SC=0000

6D Rj+D→MAR Rj→A，D→B，A+B，DM，CPMAR，返回，SC=1000

习题3-12、如果将CPU时钟周期与访存周期分开设置，一个访存周期占用四个时钟周期，试重新设计模型机指令流程。

解：一个访存周期占用四个时钟周期，可在第一个时钟周期中传送地址，即CPU将地址从MAR送往存储器；在第二个时钟周期中向存储器发读令或写令；第三个时钟周期从存储器读出数据或向存储器写入数据；第四个时钟周期结束存储器访问，即CPU撤消地址、命令、数据。

下面，以加法指令ADD（R1），R0；为例，重新拟定模型机指令流程。

FT0：MAR→M ；进入FT，将现行指令地址经地址总线送往存储器。

FT1：读令→M ；将读命令经控制总线送往存储器。

FT2：M→IR ；从存储器读出指令，经数据总线送往IR。

FT3：PC+1→PC ；撤消总线上的地址、命令、指令，并修改PC，结束FT。

DT0：R1→MAR ；进入DT，将R1内容送往MAR。

DT1：MAR→M ；将目的地址经地址总线送往存储器。

DT2：读令→M ；将读命令经控制总线送往存储器。

DT3：M→MDR ；从存储器读出目的操作数，经数据总线送往MDR。

DT4：MDR→D ；撤消总线上的地址、命令、数据，并将目的操作数从MDR送

；往暂存器D，结束DT。

ET0：R0+D→MDR ；进入ET，将源操作数和目的操作数相加，结果送往MDR。

ET1：MAR→M ；将目的地址经地址总线送往存储器。

ET2：写令→M ；将写命令经控制总线送往存储器。

ET3：MDR→M ；将运算结果经数据总线写入存储器。

ET4：PC→MAR ；撤消总线上的地址、命令、数据，并将下条指令的地址送

；往MAR。

（13分）某计算机字长16位，采用16位定长指令字结构，部分数据通路结构如图4-6所示，图中所有控制信号为1时表示有效、为0时表示无效，例如控制信号MDRinE为1表示允许数据从DB打入MDR，MDRin为1表示允许数据从内总线打入MDR。假设MAR的输出一直处于使能状态。加法指令“ADD（R1），R0”的功能为（R0）+（（R1））→（R1），即将R0中的数据与R1的内容所指主存单元的数据相加，并将结果送入R1的内容所指主存单元中保存。

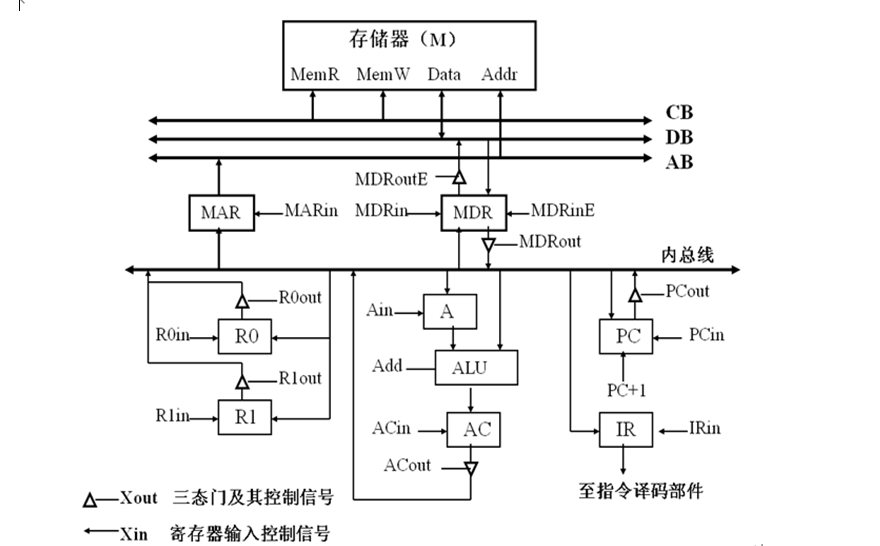
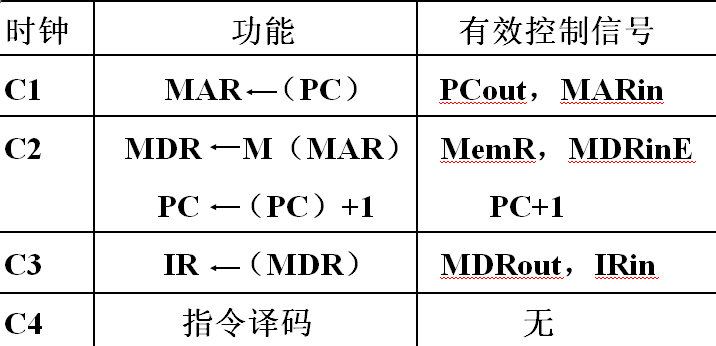


图4-6 某机部分数据通路结构

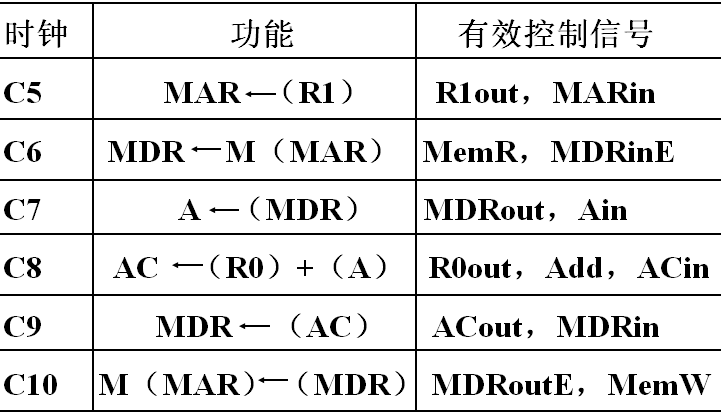
表4-1给出了上述指令取指和译码阶段每个节拍（时钟周期）的功能和有效控制信号，请按表中描述方式用表格列出指令执行阶段每个节拍的功能和有效控制信号。

表4-1 取指和译码阶段每个节拍的功能和有效控制信号

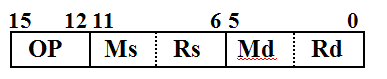


答案：如表4-2所示。

表4-2 指令执行阶段每个节拍的功能和有效控制信号



例、某计算机字长为16位, 主存地址空间大小为128KB,按字编址。采用单字长指令格式, 指令各字段定义如下：



转移指令采用相对寻址, 相对偏移量用补码表示。寻址方式定义如表4-3：

表4-3 寻址方式定义

|  |  |  |  |
| --- | --- | --- | --- |
| **Ms/Md** | **寻址方式** | **助记符** | **含义** |
| **000B** | **寄存器直接** | **Rn** | **操作数=(Rn)** |
| **001B** | **寄存器间接** | **(Rn)** | **操作数=((Rn))** |
| **010B** | **寄存器间接、自增** | **(Rn)+** | **操作数=((Rn)),**  **(Rn)+1→ Rn** |
| **011B** | **相对** | **D(Rn)** | **转移目标地址=**  **(PC)+ (Rn)** |

请回答以下问题：

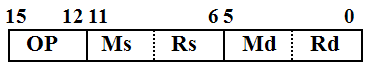
（1）该指令系统最多可有多少条指令, 该计算机最多有多少个通用寄存器, 存储器地址寄存器(MAR)和存储器数据寄存器至少各需要多少位(MDR)？

（2）转移的目标地址范围是多少？

（3） 若0010B表示加法操作(助记符add), 寄存器R4和R5的编号分别为100B和101B, R4的内容为1234H, R5的内容为5678H, 地址1234H中的内容为5678H, 地址5678H中的内容为1234H, 则汇编指令“add (R4), (R5)+”(逗号前为原操作数, 逗号后为目标操作数)对应的机器码是什么(用十六进制表示)? 该指令执行后, 哪些寄存器和存储单元的内容会改变？改变后的内容是什么?

解答：

（1）根据指令格式：



因为操作码为4位, 所以最多有16条指令。因为用来表示寄存器的位数是3位, 所以, 最多有8个通用寄存器。

存储空间按字编址, 因此存储单元数量为128KB/2=64KB。需要16位地址, 所以地址寄存器MAR为16为。因为字长16位, 所以数据寄存器MDR为16位。

（2）因为地址位数和字长都是16位, 所以PC和通用寄存器的位数均为16位, 所以转移目标地址位数为16位, 因此转移目标地址范围0000H~FFFFH。

（3）指令“add (R4), (R5)+”操作码为0010, 源操作数采用间接寻址, 编码001, 目标操作数采用间接自增寻址, 编码010, R4的编码为100, R5的编码为101B, 所以对应的机器码为 0010 001 100 010 101 (2315H)。

指令执行后, 结果存入5678H单元, 因此由原来的1234H变为68ACH, R5的内容加1后变为5679H。

4、组合逻辑控制方式和微程序控制方式

组合逻辑控制方式和微程序控制方式是**产生微命令的两种不同方式**，前者是直接通过硬连逻辑电路产生微命令，后者则是通过执行微指令产生微命令。对于这两种控制方式，需掌握它们的基本思想、优缺点及应用场合。

（1）组合逻辑控制方式

●基本思想

综合化简产生微命令的条件，形成逻辑式，用组合逻辑电路实现；执行指令时，由组合逻辑电路(微命令发生器)在相应时间发出所需微命令，控制有关操作。

由于组合逻辑电路反映了产生微命令的条件与微命令之间的逻辑关系，因而只要条件（逻辑条件如指令信息、状态信息和时间条件）满足，在组合逻辑电路的输出端就能获得所需微命令，用来控制相应操作。

●优缺点

主要优点：产生微命令的速度较快。

主要缺点：有两个主要缺点，其一，设计不规整，设计效率较低。这是因为不同的微命令可能需要不同的条件，这就使得产生微命令的逻辑电路零乱、不规整，难于实现设计自动化。其二，

不易修改、扩展指令系统功能。因为逻辑电路固定连接，修改起来非常麻烦。机器一旦制作出来，硬件功能就固定了，很难进一步扩充功能。

●应用场合

用于高速计算机，或小规模计算机。

（2）微程序控制方式

●基本思想

微程序控制方式是针对组合逻辑控制方式的主要缺点提出来的，它将程序技术和存储逻辑引入CPU的构成级，较好地解决了设计的规整性和功能的可修改性问题。其基本思想可归结为以下几点。

①将一步操作所需的若干微命令编写在一条微指令中，控制实现一步操作；

②用若干微指令组成一段微程序，解释执行一

条机器指令；

③将微程序事先存放在控制存储器中，执行机器指令时再取出。

●优缺点

主要优点：设计规整，设计效率高；易于修改、扩展指令系统功能；结构规整、简洁，可靠性高；性价比高。

主要缺点：其一，速度较慢，因访存频繁且转移较多；其二，执行效率不高，因未充分发挥数据通路本身所具有的并行操作能力。

●应用场合

用于速度要求不高、功能较复杂的机器中，特别适用于系列机。

14、根据表3-13微程序，以微地址序列形式（如00-01-02-0C…），拟出下述指令的读出与执行过程。

（1）MOV（R0），（SP）+；

（2）MOV（R1）+，X（R0）；

（3）ADD X（R0），R1；

（4）SUB（R1）+，（R2）；

（5）NEG -（R2）；

（6）JMP（R0）；

（7）JSR R1；

解：注意MOV指令和双操作数指令中源地址与目的地址的位置，源地址在后，目的地址在前。

（1）MOV（R0），（SP）+；

00-01-02-03-4C-52-53-54-4F-04-60-62-05-06-07-08-09-00

解释微地址序列：先取指（00-01-02），再转MOV指令入口（03），然后转“取源操作数”入口（4C），进入自增型寄存器间址（52-53-54-4F），返回（04），再转“取目的地址”入口（60），进入寄存器间址（62），返回（05），最后执行传送操作并转取指入口（06-07-08-09-00）。

（2）MOV（R1）+，X（R0）；

00-01-02-03-4C-59-5A-5B-5C-5D-4F-04-60-65-66-05-06-07-08-09-00

解释微地址序列：取指（00-01-02），再转MOV指令入口（03），然后转“取源操作数”入口（4C），进入变址（59-5A-5B-5C-5D-4F），返回（04），再转“取目的地址”入口（60），进入自增型寄存器间址（65-66），返回（05），最后执行传送操作并转取指入口（06-07-08-09-00）。

（3）ADD X（R0），R1；

00-01-02-0C-4C-4D-0D-60-6A-6B-6C-6D-0E-0F-10-11-07-08-09-00

解释微地址序列：ADD是双操作数指令，因此取指后进入其入口（0C），转“取源操作数”入口（4C），进入寄存器寻址（4D），返回（0D），再转“取目的地址”入口（60），进入变址（6A-6B-6C-6D），返回（0E-0F），最后执行加法操作并转取指入口（10-11-07-08-09-00）。

（4）SUB（R1）+，（R2）；

00-01-02-0C-4C-4E-4F-0D-60-65-66-0E-0F-14-15-07-08-09-00

解释微地址序列：取指（00-01-02），进入双操作数指令入口（0C），转“取源操作数”入口（4C），进入寄存器间址（4E-4F），返回（0D），再转“取目的地址”入口（60），进入自增型寄存器间址（65-66），返回（0E-0F），最后执行减法操作并转取指入口（14-15-07-08-09-00）。

（5）NEG -（R2）；

00-01-02-24-60-63-64-62-25-26-2B-2C-07-08-09-00

解释微地址序列：NEG是单操作数指令，取指后进入其入口（24），转“取目的地址”入口（60），进入自减型寄存器间址（63-64-62），返回（25-26），最后执行求补操作并转取指入口（2B-2C-07-08-09-00）。

（6）JMP（R0）；

00-01-02-3F-43-4C-4E-4F-44-45-08-09-00

解释微地址序列：JMP是转移指令，取指后进入其入口（3F-43），转“取源操作数”入口（4C），进入寄存器间址（4E-4F），返回（44-45），将转移地址送入MAR并转取指入口（08-09-00）。

（7）JSR R1；

00-01-02-3F-46-48-49-4A-4B-47-43-4C-4D-44-45-08-09-00

解释微地址序列：JSR是转子指令，取指后进入其入口（3F-46），转“压栈”入口，将转子时的返回地址压入堆栈保存（48-49-4A-4B），返回（44-45），将子程序入口地址送入MAR并转取指入口（08-09-00）。