**3.6 [5] <3.2>假定185和122是无符号8位十进制整数。计算185-122。是否有上溢或者下溢?**

解：

（含义 185转化为2进制位10111001，解释为无符号8位十进制整数）

185 -> 10111001

122 -> -01111010

00111111 -> 63 无溢出

**3.7 [5] <3.2>假定185和122是带符号8位十进制整数且以符号-数值形式存放。计算185 + 122。是否有上溢或者下溢?**

解：

（含义 185转化为2进制位10111001，但解释为符号-值，所以实际代表十进制-57）

185 -> 1（符号位）0111001（数值） -> -57 -> 11000111（补码）

11000111

+01111010

（1）01000001 -> 65 无溢出

**3.12 [20] <3.3>使用图3-3所示的硬件描述计算八进制无符号6位整数62和12的乘积，并给出一个类似于图3-6中的表。必须给出每个步骤中每个寄存器的内容。**

解：

（50）10转成十进制对应的值

001 010 –〉12 110 010 –〉62

（10）10

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 迭代次数 | 步骤 | 乘数 | 被乘数 | 乘积 |
| 0 | 初始值 | 001 010 | 000000 110010 | 000000 000000 |
| 1 | 1：0=>无操作 | 001 010 | 000000 110010 | 000000 000000 |
| 2：左移被乘数 | 001 010 | 000001 100100 | 000000 000000 |
| 3：右移乘数 | 000 101 | 000001 100100 | 000000 000000 |
| 2 | 1a：1=>乘积=乘积+被乘数 | 000 10**1** | 000001 100100 | 000001 100100 |
| 2：左移被乘数 | 000 101 | 000011 001000 | 000001 100100 |
| 3：右移乘数 | 000 010 | 000011 001000 | 000001 100100 |
| 3 | 1：0=>无操作 | 000 01**0** | 000011 001000 | 000001 100100 |
| 2：左移被乘数 | 000 010 | 000110 010000 | 000001 100100 |
| 3：右移乘数 | 000 001 | 000110 010000 | 000001 100100 |
| 4 | 1a：1=>乘积=乘积+被乘数 | 000 001 | 000110 010000 | 000111 110100 |
| 2：左移被乘数 | 000 001 | 001100 100000 | 000111 110100 |
| 3：右移乘数 | 000 000 | 001100 100000 | 000111 110100 |
| 5 | 1：0=>无操作 | 000 000 | 001100 100000 | 000111 110100 |
| 2：左移被乘数 | 000 000 | 011001 000000 | 000111 110100 |
| 3：右移乘数 | 000 000 | 011001 000000 | 000111 110100 |
| 6 | 1：0=>无操作 | 000 000 | 110010 000000 | 000111 110100 |
| 2：左移被乘数 | 000 000 | 110010 000000 | 000111 110100 |
| 3：右移乘数 | 000 000 | 110010 000000 | 000111 110100 |

62（8）= 50（10） 12（8）=10（10）

000111 110100（2） -> 500（10）（=50\*10）

**3.13 [20] <3.3>使用与图3-6相同的一张表，使用图3-5所示的硬件描述计算十六进制无符号8位整数62和12的乘积。必须给出每个步骤中每个寄存器的内容。**

（18）10

（98）10

解：

注意62和12 为8位十六进制数 0110 0010-〉62 0001 0010-〉12

|  |  |  |  |
| --- | --- | --- | --- |
| 迭代次数 | 步骤 | 被乘数 | 乘积/乘数 |
| 0 | 初始值 | 0110 0010 | 00000000 00010010 |
| 1 | 1：0=>无操作 | 0110 0010 | 00000000 00010010 |
| 2：右移乘积 | 0110 0010 | 00000000 00001001 |
| 2 | 1a：1=>乘积=乘积+被乘数 | 0110 0010 | 01100010 00001001 |
| 2：右移乘积 | 0110 0010 | 00110001 00000100 |
| 3 | 1：0=>无操作 | 0110 0010 | 00110001 00000100 |
| 2：右移乘积 | 0110 0010 | 00011000 10000010 |
| 4 | 1：0=>无操作 | 0110 0010 | 00011000 10000010 |
| 2：右移乘积 | 0110 0010 | 00001100 01000001 |
| 5 | 1a：1=>乘积=乘积+被乘数 | 0110 0010 | 01101110 01000001 |
| 2：右移乘积 | 0110 0010 | 00110111 00100000 |
| 6 | 1：0=>无操作 | 0110 0010 | 00110111 00100000 |
| 2：右移乘积 | 0110 0010 | 00011011 10010000 |
| 7 | 1：0=>无操作 | 0110 0010 | 00011011 10010000 |
| 2：右移乘积 | 0110 0010 | 00001101 11001000 |
| 8 | 1：0=>无操作 | 0110 0010 | 00001101 11001000 |
| 2：右移乘积 | 0110 0010 | 00000110 11100100 |

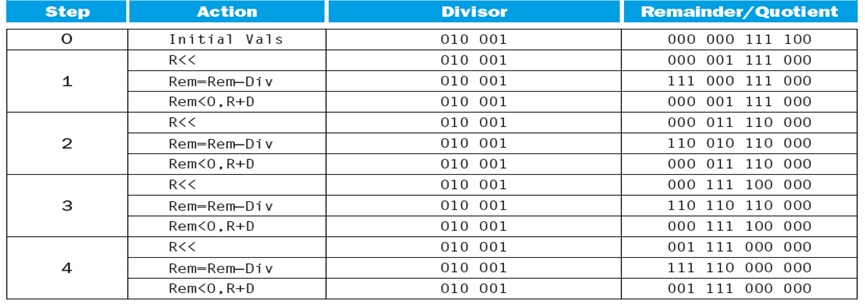
12（16）=18（10） 64（16）=98（10）

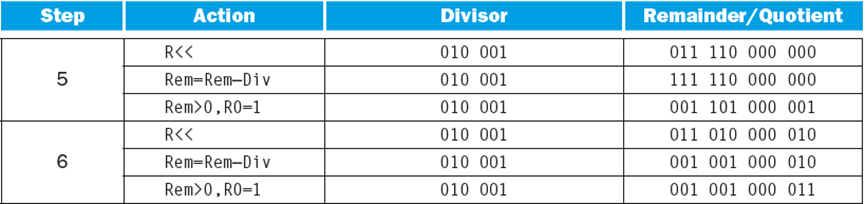
11011100100（2）=1764（10）（=98\*18）

**3.19 [30] <3.4>用图3-11中的硬件结构计算74除以21,并给出一个类似于图3- 10中的表。你需要给出每步中各个寄存器的值。假设A和B都是6位无符号整数。这个算法使用一个和图3-9中稍微不同的方法。这个算法你可能会认为很难，做-次或者 两次试验， 或者去网上寻找办法来让它正确工作。（提示:一种可能的解决方案是利用图3-11中暗示的那个余数寄存器既可右移也可左移的事实。)**

**原答案：**

解： 010 001-〉21 111 100-〉74





21（16）=17（10） 74（16）=60（10）

001 001（2）=9（10）（60/17=3余9）

**更新后的答案**

解： 010 001-〉21 111 100-〉74

|  |  |  |  |
| --- | --- | --- | --- |
| **Step** | **Action** | **Divisor** | **Remainder/Quotient** |
| 0 | Initial Vals | 010 001 | 000 000 111 100 |
| 1 | R<< | 010 001 | 000 001 111 000 |
| Rem=Rem-Div | 010 001 | 11**0** 000 111 000 |
| Rem<0,R+D | 010 001 | 000 001 111 000 |
| 2 | R<< | 010 001 | 000 011 110 000 |
| Rem=Rem-Div | 010 001 | 110 010 110 000 |
| Rem<0,R+D | 010 001 | 000 011 110 000 |
| 3 | R<< | 010 001 | 000 111 100 000 |
| Rem=Rem-Div | 010 001 | 110 110 110 000 |
| Rem<0,R+D | 010 001 | 000 111 100 000 |
| 4 | R<< | 010 001 | 001 111 000 000 |
| Rem=Rem-Div | 010 001 | 111 110 000 000 |
| Rem<0,R+D | 010 001 | 001 111 000 000 |
| 5 | R<< | 010 001 | 011 110 000 000 |
| Rem=Rem-Div | 010 001 | **00**1 1**01** 000 000 |
| Rem<0,R+D | 010 001 | 001 101 000 001 |
| 6 | R<< | 010 001 | 011 010 000 010 |
| Rem=Rem-Div | 010 001 | 001 001 000 010 |
| Rem<0,R+D | 010 001 | 001 001 000 011 |

21（16）=17（10） 74（16）=60（10）

001 001（2）=9（10）（60/17=3余9）

**3.23 [10] <3.5>写出十进制数63.25的二进制表达。设采用IEEE 754单精度格式。**

解：

符号位：0（正），指数：127+5=132（101+1111111=10000100）

精度格式：0 1000 0100 1111 1010 0000 0000 0000 000=0x427D0000

**3.24 [10] <3.5>写出十进制数63. 25的二进制表达。设采用IEEE 754双精度格式。**

解：

符号为：0（正），指数：1023+5=1028（101+111111111=10000000100）

0 100 0000 0100 1111 1010 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000=0x404FA00000000000

**3.27 [20] <3.5> IEEE 754-2008包含一种“半精度”格式，只有16位宽。最左边仍为符号位，指数有5位宽且以余-16（excess-16）的形式存储。尾数有10位宽。具有隐含1。写出-1.5625x10-1的这种格式的二进制位模式。同IEEE 754标准的单精度比较，评估这个16位位模式的范围和精确度。**

解：

|  |  |  |
| --- | --- | --- |
| 1位符号 | 5位指数 | 10位尾数 |

指数：-3+15=12（01100）小数：-.0100000000

结果:1 01100 0100000000

范围：（00000与11111保留）

最小值： 指数：00001 – 偏移量15 = -14

尾数：0000000000 实际值≈1.0 即±1.0\*2-14 ≈ 6.1\*10-5

最大值： 指数：11110 – 偏移量15 = 13

尾数：1111111111 实际值≈2.0 即±2.0\*213 ≈ 16384

精度和范围都比IEEE 754标准的单精度低。

**3.29 [20]<3.5>手算2.6125\*101和4.150390625\*10-1的和，设A和B以练习题3.27中提到的16位半精度格式存储。假设有1位保护位、1位舍入位和1位粘贴位，并采用向最靠近的偶数舍入的模式。给出所有步骤。**

解：

符号位：0（正），指数：15+4=19，则指数e=19 （10011）

16位半精度格式：0 10011 1010100011（向偶数舍入）

**3.36 [30]<3.9>手算3.41796875\*10-3\*（6.34765625\*10-3\*****1.05625\*102），设每个值都以练习题3.27中提到的16位半精度格式存储（书中也有介绍）。假设有1位保护位、1位舍入位和1位粘贴位，并采用向最靠近的偶数舍入的模式。给出所有步骤，并以16位浮点格式和十进制格式给出你的答案。**

解：

A：3.41796875\*10-3 = 1.1100000000\*2-9

B：6.34765625\*10-3 = 1.0001000000\*2-8

C：1.05625\*102 = 1.1010011010\*26

先计算B\*C：新的指数 -8+6 = -2 符号都为正数，结果也为正数

(B) 1.0001000000

(C) \* 1.1010011010

00000000000

10001000000

00000000000

10001000000

10001000000

00000000000

00000000000

10001000000

00000000000

10001000000

10001000000

1.110000001110100000000

1.1100000011 10（G,R）100000000 = 1.1100000011 101（G，R，S）

结果为1.1100000011 101 介于 1.1100000100 和1.1100000011之间

由于当中的值为1.110000011 100 所以计算结果1.1100000011 101大于该当中值

所以向上进位，取1.1100000 100为结果，则D=B\*C = 1.1100000100\*10-2

接下来计算A\*D:新的指数-9-2 = -11 符号都为正数，结果也为正数

(A) 1.1100000000

(D) \* 1.1100000100

11100000000

11100000000

11100000000

11100000000

11.00010001110000000000 标准化 指数+1 = -11+1 = -10

1.1000100011 10（G,R） 0000000000

额外的位(G,R,S100)正好是最小有效位(0)的一半，

即1.1000100011 10（G,R） 0介于1.1000100011 1.1000100100当中

所以取偶数。

则A\*(B\*C)=1.1000100100\*2-10

= 0.00000000011000100100\*20 = 0.001499176025390625 = 1.499176025390625\*10-3

符号位：0（正），指数：15-10=5，则指数e=5 （00101） 16位半精度格式：0 00101 1000100100