《计算机组成原理》第四次作业

网络空间安全学院 信息安全 陆皓喆 2211044

3.13

题目

使用与图3-6类似的表格，按照图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。

解答

可以看出该计算方式是改进后的乘法运算,我们进行列表格计算即可。

根据二进制转化，我们得到：

$$62_{16} = 01100010_2$$

$$12_{16} = 00010010_2$$

迭代轮数	步骤	被乘数	积/乘数
0	初始化	01100010	00000000/00010010
1	乘数最低位为0，无操作	01100010	00000000/00010010
1	乘数右移，积左移	01100010	000000000/0001001
2	乘数最低位为1，积=积+被乘数	01100010	011000100/0001001
2	乘数右移，积左移	01100010	0011000100/000100
3	乘数最低位为0，无操作	01100010	0011000100/000100
3	乘数右移，积左移	01100010	00011000100/00010
4	乘数最低位为0，无操作	01100010	00011000100/00010
4	乘数右移，积左移	01100010	000011000100/0001
5	乘数最低位为1，积=积+被乘数	01100010	011011100100/0001
5	乘数右移，积左移	01100010	0011011100100/000
6	乘数最低位为0，无操作	01100010	0011011100100/000
6	乘数右移，积左移	01100010	00011011100100/00
7	乘数最低位为0，无操作	01100010	00011011100100/00
7	乘数右移，积左移	01100010	000011011100100/0
8	乘数最低位为0，无操作	01100010	000011011100100/0
8	乘数右移，积左移	01100010	0000011011100100/

可以看出，经过8轮迭代，我们获得了最后的结果0000011011100100，转化成十六进制的话是

$$0000011011100100_2 = 06E4_{16}$$

所以最后的结果是，转化为十六进制答案为06E4

3.17

题目

正如书中讨论的，一种增强性能的办法是用一次移位和加法来代替一次实际的乘法。给出用移位和加/减法来计算

$$0x33 \times 0x55$$

的最好的方法。假设输入的都是8位无符号整数。

解答

我们先将这两个数转换为二进制数

$$33_{16} = 110011_2$$

$$55_{16} = 1010101_2$$

所以根据位数的大小，我们选择用二进制的33去乘上55

具体方法是：首先，我们将结果设置为0，并存储在一个寄存器中；然后，我们将55左移5位，加到结果中；再将55左移4位，加到结果中；再将55左移1位，加到结果中；再将55加到寄存器中，我们就能获得最后的结果，这种方法是最优的方法。

3.19

题目

使用类似于图3-10中的表格，按照图3-11中的硬件结构计算74除以21，需要给出每一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稍微不同的方法。

解答

我们尝试使用改进后的除法运算，该运算方式通过将余数和商放在一起，除数保持不变，通过余数和商的左右移动来解决问题。首先我们将两个数转换成二进制数，来进行进一步的运算。

$$74_8 = 111100_2$$

$$21_8 = 010001_2$$

我们通过列表格来进行计算。

迭代	步骤	除数	余数/商
0	初始化	010001	000000111100/
1	余数=余数-除数，余数小于0，商为0，加回去	010001	000000111100/
1	余数左移一位	010001	00000111100/0
2	余数=余数-除数，余数小于0，商为0，加回去	010001	00000111100/0
2	余数左移一位	010001	0000111100/00
3	余数=余数-除数，余数小于0，商为0，加回去	010001	0000111100/00
3	余数左移一位	010001	000111100/000
4	余数=余数-除数，余数小于0，商为0，加回去	010001	000111100/000

迭代	步骤	除数	余数/商
4	余数左移一位	010001	00111100/0000
5	余数=余数-除数，余数小于0，商为0，加回去	010001	00111100/0000
5	余数左移一位	010001	0111100/00000
6	余数=余数-除数，余数大于0，商为1	010001	0011010/00001
6	余数左移一位	010001	011010/000010
7	余数=余数-除数，余数大于0，商为1	010001	001001/000011

所以，根据表格的内容，我们可以得出结论：

$$\frac{74_8}{21_8} = 3_8 \cdots 11_8$$

3.28

题目

惠普2114、2115和2116采用这样一种格式，其最左边16位以补码形式存储着尾数，紧跟着的另一个16位字段里，左边8位是尾数的扩展（使尾数达到24位宽），右边8位表示指数。然而，作为一种有趣的交叉，指数以“符号-数值”的形式存储且符号位在最右端！写出这种格式下

$$-1.5625 \times 10^{-1}$$

的二进制位模式。没有隐含1。通过与IEEE 754标准单精度的比较，评估这个32位位模式的范围和精确度。

解答

首先，我们对原数进行转化，将其转化为二进制。

$$-1.5625 \times 10^{-1} = -0.00101_2$$

因为没有隐含1，所以，我们将其规范化，得到

$$-0.00101_2 = -0.101 \times 2^{-2}$$

所以，指数位是代表-2，尾数是-0.10100000...(一共是24位尾数)

因为指数部分末尾需要有一位来表示正负，所以需要7位来表示数值部分，是0000010，加上末尾的1，指数部分就变成了00000101

同理，尾数部分前16位变成了0101000000000000，取反码为1010111111111111，加上1，为1011000000000000

所以，最后的二进制位模式为10110000000000000000000101

相比单精度来说，尾数使用了24位，但是IEEE754只使用了23位作为尾数，所以该方法的精度是要高于754的；但是从指数角度来看，该方法的位数是7位，而754方法的指数位是8位，所以综合来看，两个方法各有优点：**此方法的精确度高一些，但是范围来看，还是IEEE754的方法要大一些。**

3.39

题目

手算

$$(1.666015625 \times 10^0 \times 1.9760 \times 10^4) + (1.666015625 \times 10^0 \times (-1.9744) \times 10^4)$$

设每个数值都以练习题3.27中提到的16位半精度格式存储，假设有一位保护位、一位舍入位和一位粘贴位，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答案。

解答

首先，我们需要将这三个数转为二进制来表示

$$1.666015625 \times 10^0 = 1.101010101 \times 2^0$$

$$1.9760 \times 10^4 = 1.0011010011 \times 2^{14}$$

$$(-1.9744) \times 10^4 = -1.001101001 \times 2^{14}$$

然后，我们首先进行乘法运算，先将指数次加起来，为14次，然后将两个规范化的数进行相乘，得到结果

$$1.666015625 \times 10^0 \times 1.9760 \times 10^4 = 1.101010101 \times 2^0 \times 1.0011010011 \times 2^{14} = 1.00000001001100001111 \times 2^{15}$$

同理，我们进行下一个乘法运算，还是按照前面的算法，我们计算得

$$1.666015625 \times 10^0 \times (-1.9744) \times 10^4 = 1.101010101 \times 2^0 \times (-1.001101001 \times 2^{14}) = 1.00000000111111011101 \times 2^{15}$$

我们根据有效位进行舍入，得到结果为

$$1.0000000101 \times 2^{15}$$

$$-1.0000000100 \times 2^{15}$$

所以，最终我们需要将两个数相减，得到最后的结果

$$1.0000000101 \times 2^{15} - 1.0000000100 \times 2^{15} = 0.0000000001 \times 2^{15} = 2^5 = 32$$

所以，最终的结果为32.

3.47

题目

下面的C代码实现了一个4阶FIR滤波器，其输入为数组sig_in。假设有的数组元素为16位定点数。

```
for(i=3;i<128;i++){
  sig_out[i]=sig_in[i-3]*f[0]+sig_in[i-2]*f[1]+sig_in[i-1]*f[2]+sig_in[i]*f[3];
}
```

假设你要面向一个具有SIMD指令集且有128位寄存器的处理器，使用汇编语言对该代码进行优化。在不知道指令集的情况下，简要介绍一下你该怎样实现该代码，最大限度地使用子字并行操作，并且使寄存器和存储器间的数据传送量最少。阐明你对使用的指令集的假设。

解答

我们给出指令集的假设：

1.使用8通道16位乘法

2.四个最重要的16位值的总和约简

3.移位和逐位操作

4.128位、64位、32位的最高有效位的加载与存储

我们首先声明寄存器的位数限制

```
load register F[bits 127:0] = f[3..0] & f[3..0] (64-bit load)
load register A[bits 127:0] = sig_in[7..0] (128-bit load)
```

然后我们开始实现上面的语句

```
for i = 0 to 15 do
    load register B[bits 127:0] = sig_in[(i*8+7..i*8] (128-bit load)
    for j = 0 to 7 do
        (1) eight-lane multiply C[bits 127:0] = A*F(eight 16-bit multiplies)
        (2) set D[bits 15:0] = sum of the four 16-bit values in C[bits 63:0] (reduction of four
16-bit values)
        (3) set D[bits 31:16] = sum of the four 16-bit values in C[bits 127:64] (reduction of four
16-
bit values)
        (4) store D[bits 31:0] to sig_out (32-bit store)
        (5) set A = A shifted 16 bits to the left
        (6) set E = B shifted 112 shifts to the right
        (7) set A = A OR E
        (8) set B = B shifted 16 bits to the left
    end for
end for
```

上面的语句就很好地实现了在完成任务的情况下，使用了子节并行操作，使寄存器和存储器间的数据传送量最少。