191220154 张涵之 第 3 章作业

8. $C_1 = A_1 \cdot B_1 + A_1 \cdot C_0 + B_1 \cdot C_0$
   $C_2 = A_2 \cdot B_2 + A_2 \cdot C_1 + B_2 \cdot C_1$
   $C_3 = A_3 \cdot B_3 + A_3 \cdot C_2 + B_3 \cdot C_2$
   $C_4 = A_4 \cdot B_4 + A_4 \cdot C_3 + B_4 \cdot C_3$

10. 10 的原码和补码均为 001010，-6 的原码为 100110，补码 111010，6 均为 000110

（1）$[x + y]_{补} = [10]_{补} + [-6]_{补} \pmod{2^6} = 001010 + 111010 = 000100$，真值为 4

$[x - y]_{补} = [10]_{补} + [6]_{补} \pmod{2^6} = 001010 + 000110 = 010000$，真值为 16

（2）对原码 001010 和 100110 符号位有 $0 \oplus 1 = 1$，数值部分为 01010 和 00110

C	P	Y	说明
0	0 0 0 0 0	0 0 1 1 0	$P_0 = 0$
	+ 0 0 0 0 0		$Y_5 = 0$，不做加法（加 0）
0	0 0 0 0 0	0 0 1 1 0	C、P 和 Y 同时右移一位
0	0 0 0 0 0	0 0 0 1 1	得 P_1
	+ 0 1 0 1 0		$Y_4 = 1$，+X
0	0 1 0 1 0		C、P 和 Y 同时右移一位
0	0 0 1 0 1	0 0 0 0 1	得 P_2
	+ 0 1 0 1 0		$Y_3 = 1$，+X
0	0 1 1 1 1		C、P 和 Y 同时右移一位
0	0 0 1 1 1	1 0 0 0 0	得 P_3
	+ 0 0 0 0 0		$Y_2 = 0$，不做加法（加 0）
0	0 0 1 1 1	1 0 0 0 0	C、P 和 Y 同时右移一位
0	0 0 0 1 1	1 1 0 0 0	得 P_4
	+ 0 0 0 0 0		$Y_1 = 0$，不做加法（加 0）
0	0 0 0 1 1	1 1 0 0 0	C、P 和 Y 同时右移一位
0	0 0 0 0 1	1 1 1 0 0	得 P_5

可见$[x \times y]_{原} = 100000\ 111100$，是 12 位机器数原码表示，真值为-60

（3）$[x]_{补} = 001010$，$[y]_{补} = 111010$，$[-x]_{补} = 110110$

P	Y	y_{-1}	说明
0 0 0 0 0 0	1 1 1 0 1 0	0	设 $y_{-1} = 0$，$[P_0]_{补} = 0$
			$y_0 y_{-1} = 00$，P、Y 直接右移一位
0 0 0 0 0 0	0 1 1 1 0 1	0	得$[P_1]_{补}$
+ 1 1 0 1 1 0			$y_1 y_0 = 10$，$+[-x]_{补}$
1 1 0 1 1 0			P、Y 同时右移一位
1 1 1 0 1 1	0 0 1 1 1 0	1	得$[P_2]_{补}$
+ 0 0 1 0 1 0			$y_2 y_1 = 01$，$+[x]_{补}$
0 0 0 1 0 1			P、Y 同时右移一位

0 0 0 0 1 0	1 0 0 1 1 1	0	得$[P_3]_{补}$
+ 1 1 0 1 1 0			$y_3y_2 = 10$，$+[-x]_{补}$
1 1 1 0 0 0			P、Y 同时右移一位
1 1 1 1 0 0	0 1 0 0 1 1	1	得$[P_4]_{补}$
			$y_4y_3 = 11$，P、Y 直接右移一位
1 1 1 1 1 0	0 0 1 0 0 1	1	得$[P_5]_{补}$
			$y_5y_4 = 11$，P、Y 直接右移一位
1 1 1 1 1 1	0 0 0 1 0 0	1	得$[P_6]_{补}$

可见$[x \times y]_{补}$ = 111111 000100，是 12 位机器数补码表示，真值为-60

（4）$[x]_{原}$数值部分扩展为 10 位 00000 01010，$[y]_{原}$数值部分 00110，$[-y]_{补}$取 11010

A	Q	说明
0 0 0 0 0	0 1 0 1 0	开始 $R_0 = X$
+ 1 1 0 1 0		$R_1 = X - Y$
1 1 0 1 0	0 1 0 1 0	$R_1 < 0$，则 $q_6 = 0$，没有溢出
1 0 1 0 0	1 0 1 0 _	$2R_1$（R 和 Q 同时左移，空出一位商）
+ 0 0 1 1 0		$R_2 = 2R_1 + Y$
1 1 0 1 0	1 0 1 0 0	$R_2 < 0$，则 $q_5 = 0$
1 0 1 0 1	0 1 0 0 _	$2R_2$（R 和 Q 同时左移，空出一位商）
+ 0 0 1 1 0		$R_3 = 2R_2 + Y$
1 1 0 1 1	0 1 0 0 0	$R_3 < 0$，则 $q_4 = 0$
1 0 1 1 0	1 0 0 0 _	$2R_3$（R 和 Q 同时左移，空出一位商）
+ 0 0 1 1 0		$R_4 = 2R_3 + Y$
1 1 1 0 0	1 0 0 0 0	$R_4 < 0$，则 $q_3 = 0$
1 1 0 0 1	0 0 0 0 _	$2R_4$（R 和 Q 同时左移，空出一位商）
+ 0 0 1 1 0		$R_5 = 2R_4 + Y$
1 1 1 1 1	0 0 0 0 0	$R_5 < 0$，则 $q_2 = 0$
1 1 1 1 0	0 0 0 0 _	$2R_5$（R 和 Q 同时左移，空出一位商）
+ 0 0 1 1 0		$R_6 = 2R_5 + Y$
0 0 1 0 0	0 0 0 0 1	$R_6 > 0$，则 $q_1 = 1$

商的最高位为 0，说明没有溢出，数值部分为 00001，符号位为 $0 \oplus 1 = 1$
则$[x/y]_{原}$的商为 100001 原码表示，真值为-1，余数为 000100，真值为 4

11. 用一位乘法计算要 8 * 1ns + 8 * 0.5ns = 12ns，两位乘法要 4 * 1ns + 4 * 0.5ns = 6ns

16. 不能，尽管使用 unsigned long long 之后，实参 arraysize 的表示范围增大了，但由于标注库函数函数 malloc 的形参定义为 unsigned int，则实际分配空间时，arraysize 仍会被转换成 unsigned int 传入，则按例 3.8 中取 count = 2^{30} + 1，arraysize 超出 unsigned int 能表示的最大范围，malloc 函数还是只会分配 4 个字节的空间，同样造成整数溢出。
标准库函数不能修改，则程序员要么手动实现自己的形参定义为 unsigned long long 的与 malloc 功能相同的函数用于内存分配。如果还要使用 C 语言提供的 malloc，则应该

在调用该函数前检查 arraysize 是否超出 unsigned int 表示范围，若超出，则输出提示信息告知用户数组过大，元素复制失败，并终止程序，否则再进行正常复制。
具体实现为将例 3.8 中第五行改为：

```
unsigned long long arraysize = count* (unsigned long long) sizeof (int);
unsigned int myarraysize = (unsigned int) arraysize;
if (myarraysize != arraysize) {
    printf("Failure: array size too large\n");
    return -1;
}
int myarray = (int *) malloc (myarraysize);
```

17. 使用移位（左移 n 位相当于乘以 2^n）和加减来实现乘法比直接进行乘法操作更合算。
$55 \times x = 64 \times x – 9 \times x = 64 \times x – 8 \times x – x$
$55 \times x = 32 \times x + 23 \times x = 32 \times x + 16 \times x + 8 \times x – x$
则前一种更合算，进一步可表示为 $55 \times x = x << 6 – x << 3 – x$
只需要进行两次位移、两次减法操作即可，共需要 4 个时钟周期就可以完成

21. IEEE 754 标准单精度和双精度浮点数格式的尾数分别为 23 位和 52 位
则加上隐藏位，能表示的最大有效位数分别为 24 位和 53 位
则不能精确表示的最小正整数分别为 $2^{24} + 1$ 和 $2^{53} + 1$

22. ①对于结果形如+/-1b.bb…b 的情况，需要进行右归：尾数右移一位，阶码加 1。注意右移后得到的隐藏位为 1。最后一位移出时，要考虑舍入。
②对于结果形如+/-0.00…01bb..b 的情况，需要进行左归：尾数逐次左移，每移一位价码减 1，直到阶码全为 0 或第一位 1 移到小数点左边。其中当阶码全为 0 时，尾数不再左移，结果为非规格化数形式。进行尾数相加时，默认小数点位置在第一个数值位（即隐藏位）之后，所以小数点右移 k 位后被移到了第一位 1 后面，这个 1 就是隐藏位。

25. $x = 0.75 = 0.11B = 1.1B \times 2^{-1}$，机器数为 0 0111 1110 100 0000 0000 0000 0000 0000
$y = -65.25 = 1000001.01B = 1.00000101B \times 2^6$，
机器数为 1 1000 0101 000 0010 1000 0000 0000 0000。
（1） 对阶。$[E_x]_{移}$ = 0111 1110，$[E_y]_{移}$ = 1000 0101，$[E_x – E_y]_{补} = [E_x]_{移} + [-[E_y]_{移}]_{补}$ = 0111 1110 + 0111 1011 = 1111 1001，$E_x – E_y$ = -111B = -7，x 的尾数右移 7 位，对阶后 x 的阶码为 1000 0101，尾数为 0.000 000**1** 1000 0000 0000 0000 **00**，其中粗体“1”为右移的隐藏位，最低两位粗体“0”为移出时保留的附加位。
（2） 尾数相加。0.000 0001 1000 0000 0000 0000 00 – 1.000 0010 1000 0000 0000 0000 = -1.000 0001 0000 0000 0000 0000 00。
（3） 尾数规格化。尾数相加后的结果已经是规格化结果。
（4） 舍入。针对第（3）步得到的结果，对小数点右边第 23 位后的数字进行舍入，得到最终的尾数部分。此处舍去的两位数字均为 0，直接丢弃即可。
x + y 的机器数为 1 1000 0101 000 0001 1000 0000 0000 0000
x + y 的真值为$-1.0000001B \times 2^6 = -1000000.1B = -64.5$

（1） 对阶，同上，阶码为 1000 0101。

（5） 尾数相加。0.000 0001 1000 0000 0000 0000 00 + 1.000 0010 1000 0000 0000 0000 = -1.000 0100 0000 0000 0000 0000 00。

（6） 尾数规格化。尾数相加后的结果已经是规格化结果。

（7） 舍入。针对第（3）步得到的结果，对小数点右边第 23 位后的数字进行舍入，得到最终的尾数部分。此处舍去的两位数字均为 0，直接丢弃即可。

x + y 的机器数为 0 1000 0101 000 0100 0000 0000 0000 0000

x + y 的真值为 $1.00001B \times 2^6 = 1000010B = 66$