第二章作业

1. 解：各自的原码、反码、补码如下：

整数值	原码	反码	补码
-35	1010 0011	1101 1100	1101 1101
-128	没法表示	没法表示	1000 0000
-127	1111 1111	1000 0000	1000 0001
-1	1000 0001	1111 1110	1111 1111

3. 解：（选做）浮点数的表示，即用来表示数的指数和尾数部分，有**一般格式**和 **IEEE754 格式**这两种区别。其中，一般格式指满足一般的二进制数机器码（包括定点整数和定点小数）的规定规则；而 IEEE754 格式则在一般格式上进一步做了一些约定，以便表示数时比较方便和高效。

下面以 32 位的浮点数表示作为例子来说明。

（1） 浮点表示的格式

一般格式如下：

其中，阶符 1 位，阶码 7 位；数符 1 位，尾数 23 位。如果阶码用移码表示，则阶符 0 表示负数，阶符 1 表示正数，这种符号特点正好与原码/补码的符号特点相反。如果把阶符和阶码合起来当作是阶码，即不额外看待阶符的话，那么用移码表示阶码时，它的范围是 0～255；这样看待时比较容易处理符号的问题，因此一般情况下对于阶码都特别地区别其符号位。

阶符用移码表示时，就是真实的指数 e 加上 128，即 E=e+128。这样的话，可以使得所有的指数都在 0～255 的范围内，这样在浮点加减进行对阶时有利于两个阶码的比较，这是因为正数比较容易用硬件电路实现。

数符 1 表示负数，数符 0 表示正数。通常数符独立于尾数，放在最高位。尾数可以用原码或补码纯小数表示，具体取决于应用需求。为了提高表示的效率，尾数用规格化数表示，即满足下列形式的为规格化数：

A、 尾数为原码时

正数：0.1xxxxx……x

负数：1.1xxxxx……x

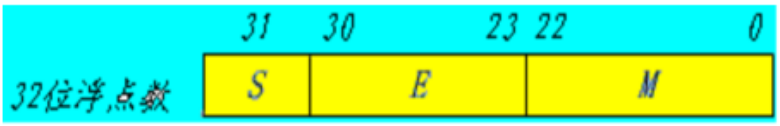
B、尾数为补码时

正数：0.1xxxxx……x

负数：1.0xxxxx……x

其中，最高位表示符号位，小数点后面表示真正的尾数的定点小数部分。

IEEE754 格式：



其中，主要是把一般格式中的数符挪到最前面（即最高比特位），符号位 1 个比特，阶码 E 占 8 个比特，尾数 M 占 23 个比特。

在一般格式的基础上，IEEE754 进一步做了一些约定：
（a）移码不再按照定义 E=e+128 进行，而是按照 E=e+127 进行计算，原因是把阶码为全 0 和全 1 这两种特殊情况排除掉了，因而只有 1～254 才表示真正的阶码；
（b）尾数用原码表示。为了进一步提高规格化数表示的效率，将规格化数规定为 1.M 的形式，即先不考虑符号位，而是通过对尾数进行适当的左移或右移，使得尾数的最高有效位总是出现在小数点的左边（对应一般格式的规格化数，最高有效位则出现在小数点的右边第 1 位）。这样，由于小数点的左边总是 1，因此在表示的时候，我们可以省略掉，只要将 1.M 中的“M”这些小数值放在 IEEE754 格式中的、23 位的“M”中即可。这样的好处是相当于多了一个比特（即隐含省去的那个 1）来表示尾数，从而使得尾数的表示效率大大增加。

在做题目的时候，究竟采用一般格式还是 IEEE754 格式，这个由题给条件来判断。如果题目中没明确声明是 IEEE754 格式，或者不能采用 IEEE754 格式，则采用一般格式，因为这通常是为了考察移码、规格化尾数的相关知识点。然而，需要注意的是，在现实应用的计算机系统中，采用的是 IEEE754 格式。

（2） 由于第 3 题中尾数用补码，而 IEEE754 格式中用的是原码，凭此可以判断不能直接用 IEEE754 格式。而对于 E 和 e 之间的关系也没有明确说明，因此采用一般格式。下面就以一般格式进行解答题目，即 e=E-128，尾数采用补码表示（规格化正数形式为 0.1xxx…x，负数形式为 1.0xxxx…x），具体的答案如下：

A、 最大数的二进制表示

0 1111 1111 1111 1111 1111 1111 1111 111

即数值 $2^{255-128}\times(1-2^{-23})=2^{127}\times(1-2^{-23})$

B、 最小数的二进制表示

1 1111 1111 0000 0000 0000 0000 0000 000

即数值 $2^{255-128}\times(-1.0)=-2^{127}$

C、 规格化数所能表示的范围

最大正数：0 1111 1111 1111 1111 1111 1111 1111 111

即数值 $2^{255-128}\times(1-2^{-23})=2^{127}\times(1-2^{-23})$

最小正数：0 0000 0000 1000 0000 0000 0000 0000 000

（原因为需要符合正数补码的一般规格化形式，见上面描述）

即数值为 $2^{0-128}\times 2^{-1}=2^{-128}\times 2^{-1}$

最大负数：1　0000 0000　0111　1111　1111　1111　1111　111

（即为负数，但绝对是最小的。由于需要符合补码负数的一般规格化形式，在符号位为 1 的时候，最高位有效数字必须是 0。这时，后面 22 个全都是 1 的时候，取反加 1 变成原码时的值才会最小；后面 22 个比特是其它组合的时候都不可能更小。这时尾数对应的原码是 1000　0000　0000　0000　0000　001）.

即数值 $2^{0-128}\times(2^{-1}+2^{-23})=2^{-128}\times(2^{-1}+2^{-23})$

最小负数：1　1111 1111　0000　0000　0000　0000　0000　000

（即为负数，但绝对是最小的。由于需要符合补码负数的一般规格化形式，在符号位为 1 的时候，最高位有效数字必须是 0。根据补码的表示范围可知，定点纯小数补码的最小值是-1，表示形式是符号为 1，尾数 23 位全为 0。）

即数值为 $2^{255-128}\times(-1)=2^{127}\times(-1)$

综上可知，规格化数所能表示的范围是：

$[2^{127}*(-1),-2^{-128}*(2^{-1}+2^{-23})]\cup[2^{-128}*2^{-1},2^{127}*(1-2^{-23})]$

5. 解：$[x+y]_{补}=[x]_{补}+[y]_{补}$

（1）$[x]_{补}$=00 11011, $[y]_{补}$=00 00011

$[x]_{补}$ =	00 11011
+ $[y]_{补}$ =	00 00011
$[x+y]_{补}$ =	00 11110

结果没有溢出，x+y=+11110。

（2）$[x]_{补}$=00 11011, $[y]_{补}$=11 01011

$[x]_{补}$ =	00 11011
+ $[y]_{补}$ =	11 01011
$[x+y]_{补}$ =	00 00110

结果没有溢出，x+y=00110。

（3）$[x]_{补}$=11 01010, $[y]_{补}$=11 11111

$[x]_{补}$ =	11 01010
+ $[y]_{补}$ =	11 11111
$[x+y]_{补}$ =	11 01001

结果没有溢出，x+y=-10111。

6．解：$[x-y]_{补}=[x]_{补}+[-y]_{补}$

（1）$[x]_{补}$=00 11011, $[-y]_{补}$=00 11111

$[x]_{补}$	=	00 11011
+ $[-y]_{补}$	=	00 11111
$[x-y]_{补}$	=	01 11010

结果有正溢出，x-y=-00110。如果考虑变形补码的表示范围，溢出可以纠正为+111010。

（2）$[x]_{补}$=00 10111，$[-y]_{补}$=11 00101

$[x]_{补}$	=	00 10111
+ $[-y]_{补}$	=	11 00101
$[x-y]_{补}$	=	11 11100

结果没有溢出，x−y=−00100。

（3）$[x]_{补}$=00 11011，$[-y]_{补}$=00 10011

$[x]_{补}$	=	00 11011
+ $[-y]_{补}$	=	00 10011
[x-y]补	=	01 01110

结果有正溢出，x-y=-10010。如果考虑变形补码的表示范围，溢出可以纠正为+101110。

7. （1）**用原码阵列乘法器：**

✧ 假定最高位为符号为，则 x 和 y 的原码为：[x]原=0 11011，[y]原=1 11111；

✧ 先求结果的符号：Sf=0⊕1=1

✧ 考虑完符号后，用各自的绝对值进行相乘，即考虑|x|×|y|=11011 × 11111，运算如下：

					1	1	0	1	1
				×)	1	1	1	1	1
					1	1	0	1	1
				1	1	0	1	1	
			1	1	0	1	1		
		1	1	0	1	1			
	1	1	0	1	1				
1	1	0	1	0	0	0	1	0	1

因此，x×y 的机器码为 x×y=1 1101000101，真值为 x×y=-1101000101。

用补码乘法阵列：

✧ 假定最高位为符号为，则 x 和 y 的补码为：[x]补=0 11011，[y]补=1 00001；

✧ 先求结果的符号：Sf=0⊕1=1

✧ 算前求补： x 的算前求补器的输出为|x|=11011，y 的算前求补器的输出为|y|=11111；

✧ 算前求补器输出相乘：即求如下的运算|x|×|y|=11011 × 11111，具体运算过程与上面的相

同，故此略去；

✧ 进行算后求补：乘积符号为 1，进行算后求补的结果为 1 0010111011。

因此，x×y 采用补码乘法阵列得到的结果，机器码为 x×y=1 0010111011，真值为 x×y=-1101000101。

（2）用原码阵列乘法器：

✧ 假定最高位为符号为，则 x 和 y 的原码为：[x]原=1 11111，[y]原=1 11011；

✧ 先求结果的符号：Sf=1⊕1=0

✧ 考虑完符号后，用各自的绝对值进行相乘，即考虑|x|×|y|=11011 × 11111，运算如下：

					1	1	0	1	1
				×)	1	1	1	1	1
					1	1	0	1	1
				1	1	0	1	1	
			1	1	0	1	1		
		1	1	0	1	1			
	1	1	0	1	1				
1	1	0	1	0	0	0	1	0	1

因此，x×y 结果的机器码为 x×y=0 1101000101，真值为 x×y=+1101000101。

用补码乘法阵列：

✧ 假定最高位为符号为，则 x 和 y 的补码为：[x]补=1 00001，[y]补=1 00101；

✧ 先求结果的符号：Sf=1⊕1=0

✧ 算前求补： x 的算前求补器的输出为|x|=11111，y 的算前求补器的输出为|y|=11011；

✧ 算前求补器输出相乘：即求如下的运算|x|×|y|=11111 ×11011，具体运算过程与上面的相同，故此略去；

✧ 进行算后求补：乘积符号为 0，进行算后求补的结果为 0 1101000101。

因此，x×y 采用补码乘法阵列得到的结果，机器码为 x×y=0 1101000101，真值为 x×y=+1101000101。

9. 解答如下：（注：为简便起见，未考虑乘数、被乘数在内存中如何存储，仅仅是运用浮点加减法的计算方法进行运算而已。）

（1）假定阶码和尾数均采用双符号补码，则题给的两个定点乘数的阶码、尾数分别为：

[Ex]补= 11 101, [Mx]补 = 00.100101

[Ey]补= 11 110, [My]补 = 11.100010

则具体的运算如下：

- 为非零操作数
- 求阶差并对阶：△E=[Ex-Ey]补=[Ex]补+[-Ey]补=11 101 + 00 010 = 11 111
  即△E=-1，x 的阶码小，应使 Ex 加 1，同时 Mx 右移 1 位，即完成对阶后的 x 的阶码和为数分别为：
  [Ex]补= 11 110, [Mx]补 = 00.010010 (1)
- 尾数求和及求差
  [Mx+My]补：

[Mx]补	0	0	0	1	0	0	1	0	(1)
+[My]补	1	1	1	0	0	0	1	0	
和	1	1	1	1	0	1	0	0	(1)

[Mx-My]补：

[Mx]补	0	0	0	1	0	0	1	0	(1)
+[-My]补	0	0	0	1	1	1	1	0	
和	0	0	1	1	0	0	0	0	(1)

- 规格化处理：
  - 对于[Mx+My]补=11.110100(1)，先判断是否有溢出。变形补码符号位为 11，因此无溢出，不需要右规。
    再判断是否需要左规。最高位符号为 1，规格后的数据小数点右边第一位必须为 0，但结果不是这样。因此需要进行左规，以便小数点右边第 1 位是 0，这时需要左移 2 位，得到[Mx+My]补=11.010010(0)。相应地，阶码减 2，即阶码变为 11 100；
  - 对于[Mx-My]补=00.110000(1)，先判断是否有溢出。可以看到变形补码的符号位为 00，因此无溢出，不需要右规。
    再判断是否需要左规。最高位符号为 0，规格后的数据小数点右边第一位必须为 1。结果已经符合规格化要求，因此不再需要进行左规。
- 舍入处理，采用 0 舍 1 入法，则处理结果如下：
  - 对于[Mx+My]补=11.010010(0)，括号里的 0 舍去，即结果为[Mx+My]补=11.010010
  - 对于[Mx-My]补=00.110000(1)，括号里的 1 未超过最低有效位数值的一半，应舍去，即结果为[Mx-My]补=00.110000
- 最后的结果为：
  - x+y 的指数补码为 11100，则真值为-100。因此，$x + y = -0.101110 \times 2^{-100}$
  - x-y 的指数补码为 11110，则真值为-010。因此，$x - y = +0.110000 \times 2^{-010}$

（2）假定阶码和尾数均采用变形补码，则题给的两个定点乘数的解码、尾数分别为：

[Ex]补= 11 011,　[Mx]补 = 11.101010

[Ey]补= 11 100,　[My]补 = 00.010110

则具体的运算如下：

- 为非零操作数
- 求阶差并对阶：△E=[Ex-Ey]补=[Ex]补+[-Ey]补=11 011 + 00 100 = 11 111
  即△E=-1，x 的阶码小，应使 Ex 加 1，同时 Mx 右移 1 位，即完成对阶后的 x 的阶码和为数分别为（负数补码右移时，符号位不改变，右移方法则与正数补码的相同）：

  [Ex]补= 11 100,　[Mx]补 = 11.110101 (0)
- 尾数求和及求差

[Mx+My]补：

[Mx]补	1	1	1	1	0	1	0	1	(0)
+[My]补	0	0	0	1	0	1	1	0	
和	0	0	0	0	1	0	1	1	(0)

[Mx-My]补：

[Mx]补	1	1	1	1	0	1	0	1	(0)
+[-My]补	1	1	1	0	1	0	1	0	
和	1	1	0	1	1	1	1	1	(0)

- 规格化处理：
  - 对于[Mx+My]补=00.001011(0)，先判断是否有溢出。符号位为 00，因此无溢出，不需要右规。
    再判断是否需要左规。最高位符号为 0，规格后的数据小数点右边第一位必须为 1，但结果不是这样。因此需要进行左规，即需要左移 2 位以保证小数点右边第 1 位为 1，得到[Mx+My]补=00.101100(0)。相应地，阶码减 2，即阶码变为 11 010；
  - 对于[Mx-My]补=11.011111(0)，先判断是否有溢出。可以看到符号位为 11，因此无溢出，不需要右规。
    再判断是否需要左规。最高位符号为 1，规格后的数据小数点右边第一位必须为 0。结果已经符合规格化要求，因此不需要左规。
- 舍入处理，采用 0 舍 1 入法，则处理结果如下：
  - 对于[Mx+My]补=0.101100(0)，括号里的 0 舍去，即结果为[Mx+My]补=0.101100；
  - 对于[Mx-My]补=1.011111(0)，括号里的 0 舍去，即结果为[Mx-My]补=1.011111
- 最后的结果为：
  - x+y 的指数补码为 11010，则真值为-110。因此，$x + y = +0.101100 \times 2^{-110}$
  - x-y 的指数补码为 11100，则真值为-100。因此，$x - y = -0.1000001 \times 2^{-100}$

10. 解答如下：

（1）$(13)_{10}=(1101)_2$，$16=2^4$，因此 13/16 相当于将$(1101)_2$左移 4 位，同时考虑尾数 6 位的要求，得到$(0.110100)_2$。类似地，9/16 为$(0.100100)_2$。因此，题给的两个定点乘数相应的指数变形补码、尾数原码分别为：

[Ex]补= 00011，　[Mx]原 = 0.110100

[Ey]补= 00100，　[My]原 = 1.100100

则具体的运算如下：

- 非零操作数
- 指数相加：[Ez]补=[Ex]补+[Ey]补= 00011+00100=00111
- 尾数相乘的符号位：Sf = 0⊕1=1
- 尾数的绝对值进行原码相乘，即|Mx|×|My|=0.110100 × 0.100100：

```
        0 1 1 0 1 0 0
    ×   0 1 0 0 1 0 0
  -------------------
        0 1 1 0 1
      0 0 0 0 0
    0 0 0 0 0
  0 1 1 0 1
0 0 0 0 0
-------------------
0 0 1 1 1 0 1 0 1 0 0
```

- 结果为 0.0111010100，因此需要进行左规，左移 1 位后得到 0.1110101000；阶码减 1 为 00111-00001=00110；
- 舍入处理：按题目要求，保留 6 位小数位，则 1000 需要进行舍入处理。现采用的是原

码，没指出舍入处理方法的时候，缺省按 0 舍 1 入法。因舍入处理部分 1000 的最高位为 1，因此进行进位，得到舍入处理后的尾数为 0.111011。

- 考虑到符号位为 1，则最后的结果为-0.111011×2^6。

（2）因除法未作介绍，故此略去。

12. 解：依题意，采用 IEEE754 标准，其 32 位格式如下：

1 位	8 位	23 位
S	E	M

其中 S=0 代表整数，S=1 为负数；E=e+127 采用译码表示，e 为真值的指数；尾数采用 1.M 形式，但存储时只存储以原码表示的 M。

（1）$(-5)_{10} = (-1.01\times 2^2)_2$，因此其 32 位的 IEEE754 格式为：

符号：S=1

阶码：E=e+127=2+127=129

尾数：M=01000. . .0，1 后面共有 21 个“0”

1 位	8 位	23 位
1	1000 0001	01000. . .0

写成十六进制形式为：$(C0A00000)_{16}$

（2）$(-1.5)_{10} = (-1.1\times 2^0)_2$，因此其 32 位的 IEEE754 格式为：

符号：S=1

阶码：E=e+127=0+127=127

尾数：M=1000. . .0，1 后面共有 22 个“0”

1 位	8 位	23 位
1	0111 1111	1000. . .0

写成十六进制形式为：$(BFC00000)_{16}$

（3）$(384)_{10} = (1.1\times 2^8)_2$，因此其 32 位的 IEEE754 格式为：

符号：S=0

阶码：E=e+127=8+127=135

尾数：M=1000. . .0，1 后面共有 22 个“0”

1 位	8 位	23 位
0	1000 0111	1000. . .0

写成十六进制形式为：$(43C00000)_{16}$

（4）$(1/16)_{10} = (1.0\times 2^{-4})_2$，因此其 32 位的 IEEE754 格式为：

符号：S=0

阶码：E=e+127=-4+127=123

尾数：M=0000. . .0，共有 23 个“0”

1 位	8 位	23 位
0	0111 1011	000. . .0

写成十六进制形式为：$(3D800000)_{16}$

（5）$(-1/32)_{10}$ = $(-1.0\times 2^{-5})_2$，因此其 32 位的 IEEE754 格式为：

符号：S=1

阶码：E=e+127=-5+127=122

尾数：M=0000. . .0，共有 23 个“0”

1 位	8 位	23 位
1	0111 1010	000. . .0

写成十六进制形式为：$(BD000000)_{16}$

13. 解：IEEE754 标准中，32 位的格式如下：

1 位	8 位	23 位
S	E	M

其中 S=0 代表整数，S=1 为负数；E=e+127 采用译码表示，e 为真值的指数；尾数采用 1.M 形式，但存储时只存储以原码表示的 M。

（1）由题给数值可知：

- ✧ 符号 S=1，为负数
- ✧ 指数 e=E-127=131-127=4
- ✧ 尾数 =1+0.M=1+0.11=1.11

因此，所代表的十进制真值为：$(-1.11\times 2^4)_2=(-28)_{10}$

（2）由题给数值可知：

- ✧ 符号 S=0，为正数
- ✧ 指数 e=E-127=126-127=-1
- ✧ 尾数 =1+0.M=1+0.101=1.101

因此，所代表的十进制真值为：$(1.101\times 2^{-1})_2=(0.8125)_{10}$