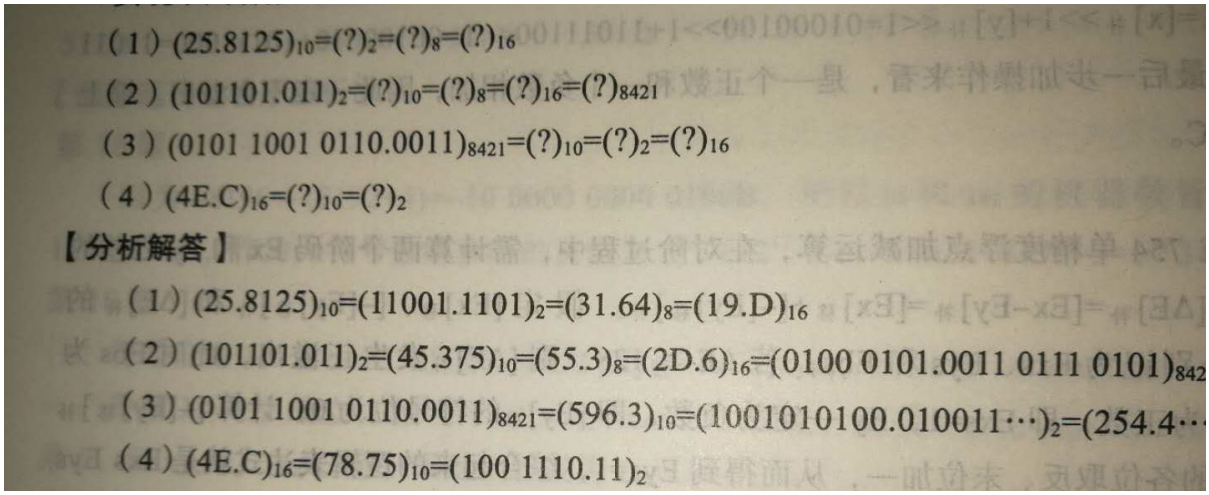
第 二 章　习　题 答 案

第 3 题



（1）$(25.8125)_{10}=(?)_2=(?)_8=(?)_{16}$

（2）$(101101.011)_2=(?)_{10}=(?)_8=(?)_{16}=(?)_{8421}$

（3）$(0101\ 1001\ 0110.0011)_{8421}=(?)_{10}=(?)_2=(?)_{16}$

（4）$(4E.C)_{16}=(?)_{10}=(?)_2$

【分析解答】

（1）$(25.8125)_{10}=(11001.1101)_2=(31.64)_8=(19.D)_{16}$

（2）$(101101.011)_2=(45.375)_{10}=(55.3)_8=(2D.6)_{16}=(0100\ 0101.0011\ 0111\ 0101)_{842}$

（3）$(0101\ 1001\ 0110.0011)_{8421}=(596.3)_{10}=(1001010100.010011\cdots)_2=(254.4\cdots$

（4）$(4E.C)_{16}=(78.75)_{10}=(100\ 1110.11)_2$

第 4 题

表 2.1　小数的原码和补码表示

数　值	原　码	补　码
+0.1001	0.1001000	0.1001000
−0.1001	1.1001000	1.0111000
+1.0	溢出	溢出
−1.0	溢出	1.0000000
+0.010100	0.0101000	0.0101000
−0.010100	1.0101000	1.1011000
+0	0.0000000	0.0000000
−0	1.0000000	0.0000000

第 5 题

表 2.2　整数的补码和移码表示

数　值	补　码	移码（偏置常数 =1 0000000）
+1001	0 0001001	1 0001001
−1001	1 1110111	0 1110111
+1	0 0000001	1 0000001
−1	1 1111111	0 1111111
+10100	0 0010100	1 0010100
−10100	1 1101100	0 1101100
+0	0 0000000	1 0000000
−0	0 0000000	1 0000000

第 6 题：

(1)　X=0001 1001 B= -25　(2) X= -1000 0000 B=-128

(3) X=0101 0010 B = 82　(4) X=-0010 1101 B=-45

第 7 题

(1) R1=0000 108BH，R2=8080 108B H

(2) R1=0000 108B H　R2= - 7F7FEF75 H

(3) R1：符号位为 0，阶码为 0000 0000 尾数：000 0000 0001 0000 1000 1011 R1=+ 1. 0000 0000 0010 0001 0001 0110（2） × 2（-127）

R2：符号位为 1，阶码为 0000 0001 尾数：000 0000 0001 0000 1000 1011 R2= - 1. 0000 0000 0010 0001 0001 0110（2） × 2（1-127）

第 8 题

关系 表达式	类型	结果	说明
0 = = 0U	无	1	00…0B = 00…0B
-1 < 0	带	1	11…1B (-1) < 00…0B (0)
-1 < 0U	无	0*	11…1B (2^{32}-1) > 00…0B(0)
2147483647 > -2147483647 - 1	带	1	011…1B (2^{31}-1) > 100…0B (-2^{31})
2147483647U > -2147483647 - 1	无	0*	011…1B (2^{31}-1) < 100…0B(2^{31})
2147483647 > (int) 2147483648U	带	1*	011…1B (2^{31}-1) > 100…0B (-2^{31})
-1 > -2	带	1	11…1B (-1) > 11…10B (-2)
(unsigned) -1 > -2	无	1	11…1B (2^{32}-1) > 11…10B (2^{32}-2)

9. 在 32 位计算机中运行一个 C 语言程序，在该程序中出现了以下变量的初值，请写出它们对应的机器数（用十六进制表示）。

(1) int x=-32768　(2) short y=522　(3) unsigned z=65530

(4) char c=’ @’　(5) float a=-1.1　(6) double b=10.5

参考答案：

(1) -2^{15}=-1000 0000 0000 0000B，故机器数为 1…1 1000 0000 0000 0000=FFFF8000H (常见错误：00008000H)

(2) 522=10 0000 1010B，故机器数为 0000 0010 0000 1010=020AH

(3) 65530=2^{16}-1-5=1111 1111 1111 1010B，故机器数为 0000FFFAH

(4) ’ @’ 的 ASCII 码是 40H

(5) -1.1=-1.00011 [0011]…B=-1.000 1100 1100 1100 1100 1100B，阶码为 127+0=01111111，故机器数为 1 01111111 000 1100 1100 1100 1100 1100=BF8CCCCCH

(6) 10.5=1010.1B=1.0101B×2^3，阶码为 1023+3=100 0000 0010，故机器数为 0 100 0000 0010 0101 [0000]=40250000 00000000H

10. 在 32 位计算机中运行一个 C 语言程序，在该程序中出现了一些变量，已知这些变量在某一时刻的

机器数（用十六进制表示）如下，请写出它们对应的真值。

（1）int x：FFFF0006H　（2）short y：DFFCH　（3）unsigned z：FFFFFFFAH

（4）char c：2AH　5）float a：C4480000H　（6）double b：C024800000000000H

参考答案：

（1）FFFF0006H=1…1 0000 0000 0000 0110B，故 x= -1111 1111 1111 1010B= -(65535-5)=-65530

（2）DFFCH=1101 1111 1111 1100B=-010 0000 0000 0100B

故 y=-(8192+4)=-8196

（3）FFFFFFFAH=1…1 1010B，故 $z=2^{32}-6$

（4）2AH=0010 1010B，故 c=42，若 c 表示字符，则 c 为字符’*’

（5）C4480000H=1100 0100 0100 1000 0…0B，阶码为 10001000，阶为 136-127=9，尾数为-1.1001B，

故 $a=-1.1001B\times2^{9}=$ -11 0010 0000B= -800

（6）C024800000000000H=1100 0000 0010 0100 1000 0 0…0B，阶码为 100 0000 0010，阶为 1026-1023=3，尾数为 1.01001B，故 $b=-1.01001B\times2^{3}=$ -1010.01B= -10.25

11. 以下给出的是一些字符串变量在内存中存放的字符串机器码，请根据 ASCII 码定义写出对应的字符串。指出代码 0AH 和 00H 对应的字符的含义。

（1）char *mystring1：68H 65H 6CH 6CH 6FH 2CH 77H 6FH 72H 6CH 64H 0AH 00H

（2）char *mystring2：77H 65H 20H 61H 72H 65H 20H 68H 61H 70H 70H 79H 21H 00H

参考答案：

字符串由字符组成，每个字符在内存中存放的是对应的 ASCII 码，因而可根据表 2.5 中的 ASCII 码和字符之间的对应关系写出字符串。

（1）mystring1 指向的字符串为：hello,world\n

（2）mystring2 指向的字符串为：we are happy!

其中，ASCII 码 00001010B=0AH 对应的是“换行”字符’\n’（LF)。每个字符串在内存存放时最后都会有一个“空”字符’\0’（NUL)，其 ASCII 码为 00H。

12. 以下给出的是一些字符串变量的初值，请写出对应的机器码。

（1）char *mystring1="./myfile"　（2）char *mystring2="OK, good!"

参考答案：

（1）mysring1 指向的存储区存放内容为：2EH 2FH 6DH 79H 66H 69H 6CH 65H 00H

（2）mysring2 指向的存储区存放内容为：4FH 4BH 2CH 67H 6FH 6FH 64H 21H 00H

13.

第一步结束：x 和 y 指向的内存单元内容是 a 和 a^b

第二步结束：x 和 y 指向的内存单元内容是 b 和 a^b

第三步结束：x 和 y 指向的内存单元内容是 b 和 a

16.

（1）（x>>(n-8)）<< (n-8)

(2) x &0xFF

(3) ((~x)>>8) <<8

(4) X| 0XFF

17.

b8 01 00 00 机器数为 0000 01B8 H， 真值为 440

14 机器数为 14H， 真值为 20

58 fe ff ff 机器数为 ff ff fe 58H， 真值为-424

74 fe ff ff 机器数为 ff ff fe 74 H， 真值为-396

44 机器数为 44 H， 真值为 68

c8 fe ff ff 机器数为 ff ff fe c8H， 真值为-312

10 机器数为 10 H， 真值为 16

0c 机器数为 0c H， 真值为 12

ec fe ff ff 机器数为 ff ff fe ec H， 真值为-276

20 机器数为 20 H， 真值为 32

18. 假设以下 C 语言函数 compare_str_len 用来判断两个字符串的长度，当字符串 *str*1 的长度大于 *str*2 的长度时函数返回值为 1，否则为 0。

```
1   int compare_str_len(char *str1, char *str2)
2   {
3       return strlen(str1) - strlen(str2) > 0;
4   }
```

已知 C 语言标准库函数 strlen 原型声明为“size_t strlen(const char *s);”，其中，size_t 被定义为 unsigned int 类型。请问：函数 compare_str_len 在什么情况下返回的结果不正确？为什么？为使函数正确返回结果应如何修改代码？

参考答案：

因为 size_t 被定义为 unsigned int 类型，因此，库函数 strlen 的返回值为无符号整数。函数 compare_str_len 中的返回值是 strlen(str1) - strlen(str2) > 0，这个关系表达式中>号左边是两个无符号数相减，其差还是无符号整数，因而总是大于等于 0，也即，在 str1 的长度小于 str2 的长度时结果也为 1。显然，这是错误的。

只要将第 3 行语句改为以下形式即可：

```
3       return strlen(str1) > strlen(str2) ;
```

19.

【分析解答】

函数 func1 的功能是把无符号数高 24 位清零（左移 24 位再逻辑右移 24 位），结果一定是正的带符号整数；而函数 func2 的功能是把无符号数的高 24 位都变成和第 25 位一样，因为左移 24 位后左边第一位变为原来的第 25 位，然后进行算术右移，高位补符号，即高 24 位都变成和原来第 25 位相同。

根据程序执行的结果填表（见表 2.8），其中机器数用十六进制表示。

表 2.8　题 15 中填入结果后的表

w		func1(w)		func2(w)	
机器数	值	机器数	值	机器数	值
0000007FH	127	0000007FH	+127	0000007FH	+127
00000080H	128	00000080H	+128	FFFFFF80H	−128
000000FFH	255	000000FFH	+255	FFFFFFFFH	−1
00000100H	256	00000000H	0	00000000H	0

21．以下是两段 C 语言代码，函数 arith( )是直接用 C 语言写的，而 optarith( )是对 arith( )函数以某个确定的 *M* 和 *N* 编译生成的机器代码反编译生成的。根据 optarith( )，可以推断函数 arith( ) 中 *M* 和 *N* 的值各是多少?

```
#define  M
#define  N
int   arith     (int x,   int y)
{
      int   result = 0 ;
      result = x*M + y/N;
      return result;
}

int optarith ( int x,   int y)
{
      int   t = x;
      x << = 4;
      x – = t;
      if ( y < 0 )   y += 3;
      y>>2;
      return x+y;
}
```

参考答案:

可以看出 x*M 和“int t = x; x << = 4;x-=t;”三句对应，这些语句实现了 x 乘 15 的功能（左移 4 位相当于乘以 16，然后再减 1），因此，M 等于 15；

y/N 与“if (y < 0) y += 3; y>>2;”两句对应，第二句“y 右移 2 位”实现了 y 除以 4 的功能，

因此 N 是 4。而第一句“if (y < 0) y += 3;”主要用于对 y= –1 时进行调整，若不调整，则–1>>2= –1 而–1/4=0，两者不等；调整后 –1+3=2，2>>2=0，两者相等。

24．设一个变量的值为 4098，要求分别用 32 位补码整数和 IEEE 754 单精度浮点格式表示该变量（结果用十六进制形式表示），并说明哪段二进制位序列在两种表示中完全相同，为什么会相同?

参考答案：

4098 = +1 0000 0000 0010B = +1. 0000 0000 001 × 2^{12}

32 位 2-补码形式为：0000 0000 0000 0000 0001 0000 0000 0010 （00001002H)

IEEE754 单精度格式为：0 10001011 0000 0000 0010 0000 0000 000 （45801000H)

粗体部分为除隐藏位外的有效数字，因此，在两种表示中是相同的位序列。

28．假定在一个程序中定义了变量 *x*、*y* 和 *i*，其中，*x* 和 *y* 是 float 型变量（用 IEEE754 单精度浮点数表示），*i* 是 16 位 short 型变量（用补码表示）。程序执行到某一时刻，*x*= –0.125、*y*=7.5、*i*=100，它们都被写到了主存（按字节编址），其地址分别是 100，108 和 112。请分别画出在大端机器和小端机器上变量 *x*、*y* 和 *i* 中每个字节在主存的存放位置。

参考答案：

–0.125 = –0.001B = –1.0 × 2^{-3}

x 在机器内部的机器数为：1 01111100 00...0 (BE00 0000H)

7.5= +111.1B= +1.111 × 2^{2}

y 在机器内部的机器数为：0 10000001 11100...0 (40F0 0000H)

100=64+32+4=1100100B

i 在机器内部表示的机器数为：0000 0000 0110 0100（0064H)

	大端机	小端机
地址	内容	内容
100	BEH	00H
101	00H	00H
102	00H	00H
103	00H	BEH
108	40H	00H
109	F0H	00H
110	00H	F0H
111	00H	40H
112	00H	64H
113	64H	00H

（错误：地址为 100、99、...，每个单元只存放 4 位)

29． 对于图 2.8，假设 *n*=8，机器数 *X* 和 *Y* 的真值分别是 *x* 和 *y*。请按照图 2.8 的功能填写表 2.17，并给出对每个结果的解释。要求机器数用十六进制形式填写，真值用十进制形式填写。

表 2.17 题 29 用表

表示	*X*	*x*	*Y*	*y*	*X*+*Y*	*x*+*y*	OF	SF	CF	*X*-*Y*	*x*-*y*	OF	SF	CF
无符号	0xB0		0x8C											
带符号	0xB0		0x8C											

无符号	0x7E		0x5D											
带符号	0x7E		0x5D											

参考答案：

表 2.17 题 29 用表

表示	X	x	Y	y	$X+Y$	$x+y$	OF	SF	CF	$X-Y$	$x-y$	OF	SF	CF
无符号	0xB0	176	0x8C	140	0x3C	60	1	0	1	0x24	36	0	0	0
带符号	0xB0	-80	0x8C	-116	0x3C	60	1	0	1	0x24	36	0	0	0
无符号	0x7E	126	0x5D	93	0xDB	219	1	1	0	0x21	33	0	0	0
带符号	0x7E	126	0x5D	93	0xDB	-37	1	1	0	0x21	33	0	0	0

(1) 无符号整数 176+140=316，无法用 8 位表示，即结果应有进位，CF 应为 1。验证正确。

(2) 无符号整数 176-140=36，可用 8 位表示，即结果没有进位，CF 应为 0。验证正确。

(3) 带符号整数-80+(-116)=-316，无法用 8 位表示，即结果溢出，OF 应为 1。验证正确。

(4) 带符号整数-80-(-116)=36，可用 8 位表示，即结果不溢出，OF 应为 0。验证正确。

(5) 无符号整数 126+93=219，可用 8 位表示，即结果没有进位，CF 应为 0。验证正确。

(6) 无符号整数 126-93=33，可用 8 位表示，即结果没有进位，CF 应为 0。验证正确。

(7) 带符号整数 126+93=219，无法用 8 位表示，即结果溢出，OF 应为 1。验证正确。

(8) 带符号整数 126-93=33，可用 8 位表示，即结果不溢出，OF 应为 0。验证正确。

无符号整数的加减运算的结果是否溢出，通过进位/借位标志 CF 来判断，而带符号整数的加减运算结果是否溢出，通过溢出标志 OF 来判断。

31．第 2.7.5 节中例 2.31 如下：

例 2.31 以下程序段实现数组元素的复制，将一个具有 *count* 个元素的 int 型数组复制到堆中新申请的一块内存区域中，请说明该程序段存在什么漏洞，引起该漏洞的原因是什么。

```
1   /* 复制数组到堆中，count 为数组元素个数 */
2   int copy_array(int *array, int count) {
3       int i;
4    /* 在堆区申请一块内存 */
5       int *myarray = (int *) malloc(count*sizeof(int));
6       if (myarray == NULL)
7           return -1;
8       for (i = 0; i < count; i++)
9           myarray[i] = array[i];
10      return count;
11  }
```

对于存在的整数溢出漏洞，如果将其中的第 5 行改为以下两个语句：

```
unsigned long long arraysize=count*(unsigned long long)sizeof(int);
```

int *myarray = (int *) malloc(arraysize);

已知 C 语言标准库函数 malloc 的原型声明为“void *malloc(size_t size);”，其中，size_t 定义为 unsigned int 类型，则上述改动能否消除整数溢出漏洞？若能则说明理由；若不能则给出修改方案。

参考答案：

上述改动无法消除整数溢出漏洞，这种改动方式虽然使得 arraysize 的表示范围扩大了，避免了 arraysize 的溢出，不过，当调用 malloc 函数时，若 arraysize 的值大于 32 位的 unsigned int 的最大可表示值，则 malloc 函数还是只能按 32 位数给出的值去申请空间，同样会发生整数溢出漏洞。

程序应该在调用 malloc 函数之前检测所申请的空间大小是否大于 32 位无符号整数的可表示范围，若是，则返回-1，表示不成功；否则再申请空间并继续进行数组复制。修改后的程序如下：

```
1   /* 复制数组到堆中，count 为数组元素个数 */
2   int copy_array(int *array, int count) {
3       int i;
4    /* 在堆区申请一块内存 */
5       unsigned long long arraysize=count*(unsigned long long)sizeof(int);
6       size_t myarraysize=(size_t) arraysize;
7       if (myarraysize!=arraysize)
8           return -1;
9       int *myarray = (int *) malloc(myarraysize);
10      if (myarray == NULL)
11          return -1;
12      for (i = 0; i < count; i++)
13          myarray[i] = array[i];
14      return count;
15  }
```

(错误：程序中可以直接写 “2^{32}-1”吗？)

34. 无符号整数变量 *ux* 和 *uy* 的声明和初始化如下：

unsigned ux=x;

unsigned uy=y;

若 sizeof(int)=4，则对于任意 int 型变量 *x* 和 *y*，判断以下关系表达式是否永真。若永真则给出证明；若不永真则给出结果为假时 *x* 和 *y* 的取值。

(1) (x*x) >= 0

(2) (x-1<0) || x>0

(3) x<0 || -x<=0

(4) x>0 || -x>=0

(5) x&0xf!=15 || (x<<28)<0

(6) x>y==(-x<-y)

(7) ~x+~y==~(x+y)

(8) (int) (ux-uy) == -(y-x)

(9) ((x>>2)<<2) <= x

(10) x*4+y*8==(x<<2)+(y<<3)

(11) x/4+y/8==(x>>2)+(y>>3)

(12) x*y==ux*uy

(13) x+y==ux+uy　　(14) x*~y+ux*uy==-x

参考答案:

(以下所有的判断都是通过 CMP 指令实现的，即在整数加减运算器中做减法生成标志来判断的!)

(1) (x*x)>=0

非永真。例如，x=65 534 时，则 $x*x=(2^{16}-2)*(2^{16}-2)=2^{32}-2*2*2^{16}+4 \pmod{2^{32}}=-(2^{18}-4)=-262140$。x 的机器数为 0000FFFEH，x*x 的机器数为 FFFC0004H。

(2) (x-1<0) || x>0

非永真。当 x=-2 147 483 648 时，显然，x<0，机器数为 80000000H，x-1 的机器数为 7FFFFFFFH，符号位为 0，因而 x-1>0。此时，(x-1<0) 和 x>0 两者都不成立。

(3) x<0 || -x<=0

永真。若 x>0，x 符号位为 0 且数值部分为非 0（至少有一位是 1），从而使-x 的符号位一定是 1，即则-x<0；若 x=0，则-x=0。综上，只要 x<0 为假，则-x<=0 一定为真，因而是永真。

(4) x>0 || -x>=0

非永真。当 x=-2 147 483 648 时，x<0，且 x 和-x 的机器数都为 80000000H，即-x<0。此时，x>0 和-x>=0 两者都不成立。

(5) x&0xf!=15 || (x<<28)<0

非永真。这里!=的优先级比&（按位与）的优先级高。因此，若 x=0，则 x&0xf!=15 为 0，(x<<28)<0 也为 0，所以结果为假。

(6) x>y==(-x<-y)

非永真。当 x=-2 147 483 648、y 任意（除-2 147 483 648 外），或者 y=-2 147 483 648、x 任意（除-2 147 483 648 外）时不等。因为 int 型负数-2 147 483 648 是最小负数，该数取负后结果仍为-2 147 483 648，而不是 2 147 483 648。

(7) ~x+~y==~(x+y)

永假。$[-x]_{补}=\sim[x]_{补}+1$，$[-y]_{补}=\sim[y]_{补}+1$，故$\sim[x]_{补}+\sim[y]_{补}=[-x]_{补}+[-y]_{补}-2$。

$[-(x+y)]_{补}=\sim[x+y]_{补}+1$，故$\sim[x+y]_{补}=[-(x+y)]_{补}-1=[-x]_{补}+[-y]_{补}-1$。

由此可见，左边比右边少 1。

(8) (int) (ux-uy) ==-(y-x)

永真。(int) $ux-uy=[x-y]_{补}=[x]_{补}+[-y]_{补}=[-y+x]_{补}=[-(y-x)]_{补}$

(9) ((x>>2)<<2) <= x

永真。因为右移总是向负无穷大方向取整。

(10) x*4+y*8==(x<<2)+(y<<3)

永真。因为带符号整数 x 乘以 2^k 完全等于 x 左移 k 位，无论结果是否溢出。

(11) x/4+y/8==(x>>2)+(y>>3)

非永真。当 x=-1 或 y=-1 时，x/4 或 y/8 等于 0，但是，因为-1 的机器数为全 1，所以，x>>2 或 y>>3 还是等于-1。此外，当 x 或 y 为负数且 x 不能被 4 整除或 y 不能被 8 整除，则 x/4 不等于 x>>2，y/8 不等于 y>>3。

(12) x*y==ux*uy

永真。根据第 2.7.5 节内容可知，x*y 的低 32 位和 ux*uy 的低 32 位是完全一样的位序列。

(13) x+y==ux+uy

永真。根据第 2.7.4 节内容可知，带符号整数和无符号整数都是在同一个整数加减运算部件中进行运算的，x 和 ux 具有相同的机器数，y 和 uy 具有相同的机器数，因而 x+y 和 ux+uy 具有完全一样的位序列。

(14) x*~y+ux*uy==-x

永真。-y=~y+1，即~y=-y-1。而 ux*uy=x*y，因此，等式左边为 x*(-y-1)+x*y=-x。

35. 变量 *dx*、*dy* 和 *dz* 的声明和初始化如下：

double dx = (double) x;

double dy = (double) y;

double dz = (double) z;

若 float 和 double 分别采用 IEEE 754 单精度和双精度浮点数格式，sizeof(int)=4，则对于任意 int 型变量 *x*、*y* 和 *z*，判断以下关系表达式是否永真。若永真则给出证明；若不永真则给出结果为假时 *x* 和 *y* 的取值。

(1) dx*dx >= 0

(2) (double)(float) x == dx

(3) dx+dy == (double) (x+y)

(4) (dx+dy)+dz == dx+(dy+dz)

(5) dx*dy*dz == dz*dy*dx

(6) dx/dx == dy/dy

参考答案：

(1) dx*dx >= 0

永真。double 型数据用 IEEE 754 标准表示，尾数用原码小数表示，符号和数值部分分开运算。不管结果是否溢出都不会影响乘积的符号。

(2) (double)(float) x == dx

非永真。当 int 型数据 x 的有效位数比 float 型可表示的最大有效位数 24 更多时，x 强制转换为 float 型数据时有效位数丢失，而将 x 转换为 double 型数据时没有有效位数丢失。也即等式左边可能是近似值，而右边是精确值。

(3) dx+dy == (double) (x+y)

非永真。因为 x+y 可能会溢出，而 dx+dy 不会溢出。

(4) (dx+dy)+dz == dx+(dy+dz)

永真。因为 dx、dy 和 dz 是由 32 位 int 型数据转换得到的，而 double 类型可以精确表示 int 类型数据，并且对阶时尾数移位位数不会超过 52 位，因此尾数不会舍入，因而不会发生大数吃小数的情况。

但是，如果 dx、dy 和 dz 是任意 double 类型数据，则非永真。

(5) dx*dy*dz == dz*dy*dx

非永真。相乘的结果可能产生舍入。

(6) dx/dx == dy/dy

非永真。dx 和 dy 中只要有一个为 0、另一个不为 0 就不相等。

36.参考答案：

(1) 当结果为 1x.xxx…x 时，需要右规，即尾数右移一位，价码加 1

（2）当结果为 0.00xxx...01xxx 时，需要左规，即数值位逐次左移，阶码逐次减 1，直到将第一个 1 移到小数点左边。

39．参考答案：

（1）E_b=1000 0101, M_b=1(1).0000001，即 （-1.0000001）2×2^6=-64.5

（2）E_b=1000 0101, M_b=0(1).00001000...0，即 （+1.00001）2×2^6=c+66