第 二 章　习　题 答 案

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

本题要点：

(1) 数据要根据其类型要表示到足够的位数，如题（3），要写成 0000FFFA，而不是 FFFA

(2) 因为求机器数，所以负数要用补码表示

(3) 浮点数要用 IEEE 754 标准编码，要注意是 float 还是 double，分别编码：阶码位数、尾数位数、偏置常数各不相同。要表示到足够的位数。

(4) 16 进制数后面要写 H 标识，二进制数后面要写 B 标识

参考答案：

（1）FFFF8000H

解：-2^{15}=−1000 0000 0000 0000B，负数符号位为 1，int 型整数表示为 32 位，

故机器数为 1…1 1000 0000 0000 0000=FFFF8000H

（2）020AH

解：522=10 0000 1010B，正数符号位为 0，short 型整数表示为 16 位，

故机器数为 0000 0010 0000 1010=020AH

（3）0000FFFAH

解：65530=2^{16}-1-5=1111 1111 1111 1010B，unsigned 型整数表示为 32 位，无符号位（高位补 0），

故机器数为 0000FFFAH

（4）40H

解：'@'的 ASCII 码是 40H，char 型表示为 8 位

（5）BF8CCCCCH

解：−1.1=−1.00011 [0011]…B=−1.000 1100 1100 1100 1100 1100B，

float 型浮点数，阶码为 127+0=01111111（8 位），整数部 1 为隐藏位，负数符号位为 1，

故机器数为 1 01111111 000 1100 1100 1100 1100 1100=BF8CCCCCH

（6）40250000 00000000H

解：10.5=1010.1B=1.0101B×2^3（左规），

Double 型浮点数，阶码为 1023+3=100 0000 0010，整数部 1 为隐藏位，正数符号位为 0，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

本题要点

（1） 正确换算，结果最好表示成 10 进制

（2） 注意数据类型，准确区分有符号数和无符号数，有符号数最高位为 1 时为负数

（3） 正确解析浮点数

参考答案：

（1）−65530

解：FFFF0006H=1…1 0000 0000 0000 0110B，最高位为 1，所以为负数，int 型整数，所以真值为其余为取反加 1，故 x= −1111 1111 1111 1010B= −(65535−5)=−65530

（2）−8196

解：DFFCH=1101 1111 1111 1100B=−010 0000 0000 0100B，最高位为 1，所以为负数，short 型整数，所以真值为其余为取反加 1，故 y=−(8192+4)=−8196

（3）4294967290（或 $2^{32}-6$）

解：FFFFFFFAH=1…1 1010B，最高位虽然为 1，但为 unsigned 型整数，所以解析为正整数，直接展开换算，故 $z=2^{32}-6=4294967290$

（4）字符’*’

解：2AH=0010 1010B，故 c 的值是 42，且 c 表示字符，故 c 为字符’*’

（5）−800

解：C4480000H=1100 0100 0100 1000 0…0B，因为是 float 型浮点数数据，所以按照 IEEE 754 标准解析。符号位为 1，所以为负数；阶码为 10001000，减掉偏置常数 127 后有：阶为 136−127=9；尾数考虑隐藏位为：−1.1001B，故 $a=-1.1001B\times2^9=-11\ 0010\ 0000B=-800$

（6）-10.25

解：C024800000000000H=1100 0000 0010 0100 1000 0 0…0B，因为是 double 型浮点数，所以按照 IEEE 754 标准解析。符号位为 1，所以为负数；阶码为 100 0000 0010,，减掉偏置常数 1023 后有：阶为

1026−1023=3；尾数考虑隐藏位为 1.01001B，故 b=-1.01001B×2^3= 1010.01B=-10.25

17. 以下是一个由反汇编器生成的一段针对某个小端方式处理器的机器级代码表示文本，其中，最左边是指令所在的存储单元地址，冒号后面是指令的机器码，最右边是指令的汇编语言表示，即汇编指令。已知反汇编输出中的机器数都采用补码表示，请给出指令代码中划线部分表示的机器数对应的真值。

```
80483d2: 81 ec b8 01 00 00      sub   &0x1b8, %esp
80483d8: 8b 55 08               mov   0x8(%ebp), %edx
80483db: 83 c2 14               add   $0x14, %edx
80483de: 8b 85 58 fe ff ff      mov   0xfffffe58(%ebp), %eax
80483e4: 03 02                  add   (%edx), %eax
80483e6: 89 85 74 fe ff ff      mov   %eax, 0xfffffe74(%ebp)
80483ec: 8b 55 08               mov   0x8(%ebp), %edx
80483ef: 83 c2 44               add   $0x44, %edx
80483f2: 8b 85 c8 fe ff ff      mov   0xfffffec8(%ebp), %eax
80483f8: 89 02                  mov   %eax, (%edx)
80483fa: 8b 45 10               mov   0x10(%ebp), %eax
80483fd: 03 45 0c               add   0xc(%ebp), %eax
8048400: 89 85 ec fe ff ff      mov   %eax, 0xfffffeec(%ebp)
8048406: 8b 45 08               mov   0x8(%ebp), %eax
8048409: 83 c0 20               add   $0x20, %eax
```

本题要点：

（1） 注意数据是小端方式，所以要注意字节的编排，如 5 字节：b8 01 00 00，表示的数是“000001b8”。

（2） 上述指令都是整数运算指令，所以所有的数据均按整数换算

（3） 正确换算，不要计算错了

参考答案：

（1）440

解：b8 01 00 00：4 字节，但考虑小端数据格式，所以机器数为 000001B8H，符号位为 0，所以为正整数，直接展开，真值为+1 1011 1000B = 440

（2）20

解：14：单字节，机器数即为 14H，符号位为 0，解析为正整数，所以真值为+1 0100B = 20

（3）−424

解：58 fe ff ff：4 字节，考虑小端数据格式，所以机器数为 FFFFFE58H，符号位为 1，所以为负数，真值为其余位取反加 1，所以真值为：−1 1010 1000B = −424

（4）−396

解：74 fe ff ff：4 字节，考虑小端数据格式，所以机器数为 FFFFFE74H，符号位为 1，所以为负数，真值为其余位取反加 1，所以真值为：−1 1000 1100B = −396

（5）68

解：44：单字节，机器数即为 44H，符号位为 0，解析为正整数，所以真值为+100 0100B=68

（6）−312

解：c8 fe ff ff：4 字节，考虑小端数据格式，所以机器数为 FFFFFEC8H，符号位为 1，所以为负数，真值为其余位取反加 1，所以真值为：−1 1000 1100B = −312

（7）16

解：10：单字节，机器数即为 10H，符号位为 0，解析为正整数，所以真值为+10000B=16

（8）12

解：0c：单字节，机器数即为 0CH，符号位为 0，解析为正整数，所以真值为：+1100B=12

（9）−276

解：ec fe ff ff：4 字节，考虑小端数据格式，所以机器数为 FFFFFEECH，符号位为 1，所以为负数，真值为其余位取反加 1，所以真值为：−1 0001 0100B = −276

（10）32

解：20：单字节，机器数即为 20H，符号位为 0，解析为正整数，所以真值为+00100000B=32

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;
    }
```

本题要点：

直接用 C 语言写的源代码和反编译后的代码形式上不一致，所以要从功能的角度分析，而不能只从字面看。

参考答案：

答案：M=15，N=4

解：（1）对照源代码语句中的“x*M”和反编译代码中的“int t = x; x << = 4; x-=t;”，可以分析出，反编译代码是将 x 乘以 4（x << = 4），然后减去 x 的原值（t 是 x 的备份，x-=t 相当于减去 x 的原值），所以这三条语句实现的功能是：x 乘 15。因此，M 等于 15；

（2）对照源代码语句中的“y/N”与反编译代码中的“if (y < 0) y += 3; y>>2;”，可以分析出，“y>>2”实现了 y 除以 4 的功能，因此 N 是 4。

而比较迷惑的是第一句“if (y < 0) y += 3;”什么意思呢：

其意图主要在于：对 y= –1 时进行调整。若不调整，则–1>>2= –1（-1 的补码表示是 FFFFFFFFH，移位后还是 FFFFFFFFH，其值还是-1），而–1/4=0（数学定义），所以两者不等，不符合数学定义；故做上述调整，调整后 –1+3=2，2>>2=0，两者相等，正确。

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？

本题要点：

（1） 按照 32 位补码整数和 IEEE 754 单精度浮点格式准确表示 4098 的值，尤其是浮点数位串不要搞错了。

（2） 找出位串相同的部分，为什么相同主要是 4098 数比较小，在 32 位补码整数（且为正数）和 23 位浮点尾数部分都能精确表示（没有舍入），所以对应相等（除了浮点数最高位，因为浮点数有 1 个隐藏位）。

参考答案：

解：4098 = +1 **0000 0000 0010**B（整数） = +1. 0000 0000 001 × 2^{12}（规格化浮点数）

（1）32 位 2-补码形式为：0000 0000 0000 0000 0001 **0000 0000 0010**（即 00001002H）

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

对照（1）和（2）的位串，粗体部分的 12 位位串为相同部分。

注：浮点数中粗体部分为除隐藏位外的有效数字，只考虑这 12 位，阶码中最后一个 1 不考虑。

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。

本题要点：

（1） 正确区分大端、小端数据表示，**注意字节顺序**。

（2） 将 *x*、y 和 i 根据其类型正确换算成相应的位串（机器数），尤其是浮点数，要会转换成二进制。注意：长度不同，x 和 y 是 32 位，i 是 16 位。

参考答案：

解：

首先，将 *x*、*y* 和 i 换算成正确的机器数

（1）–0.125 = –0.001B = -1.0×2^{-3}，负浮点数，符号位为 1， 阶码为-3（加偏置常数 127 后为 124），尾数为.0，所以，x 在机器内部的机器数为：1 01111100 00…0 (BE00 0000H)

（2）7.5= +111.1B= $+1.111 \times 2^{2}$，正浮点数，阶码为 2（加偏置常数 127 后为 129），尾数为.111，所以，y 在机器内部的机器数为：0 10000001 11100…0 (40F0 0000H)

（3）100=64+32+4=1100100B，正整数，直接转换，所以 i 在 32 位的机器内部表示的机器数为：0000 0000 0110 0100（0064H）

所以 *x*、*y* 和 i 在主存中的情况如下：（注意三个数的起始地址）

	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

29. 对于图 2.6，假设 n=8，机器数 X 和 Y 的真值分别是 x 和 y。请按照图 2.6 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											

本题要点：

（1） 注意位长只有 8 位，所以无符号数超过 255 或有符号数小于-128 或大于+127 将进位和溢出

（2） 正确计算有符号数和无符号数的值

（3） 正确使用模运算

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 位无符号数表示，即结果应有进位，CF 应为 1。减 256，得 60，验证正确。

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

（3）带符号整数−80+(−116)=−196，无法用 8 位有符号数表示，即结果溢出，OF 应为 1，加 256，得 60。验证正确。

（4）带符号整数−80−(−116)=36，可用 8 位有符号数表示，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 位无符号数表示，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 位无符号数表示，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 位有符号数表示，即结果溢出，OF 应为 1。减 256，得-37，验证正确。

（8）带符号整数 126−93=33，可用 8 位有符号数表示，即结果不溢出，OF 应为 0。验证正确。

注：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．对于第 2.7.5 节中例 2.31 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。

本题要点：

(1) 主要是传递给 malloc 的参数 arraysize 的值超过 unsigned int 所致。

(2) 不能通过重写 malloc 函数，将其参数类型更改为 unsigned long long 来解决，因为 C 语言函数库里的函数不是普通用户能修改的——那是开发 C 语言及其编译器本身的人做的事。你所要做的是在调用 malloc 分配空间前，**主动检查 arraysize** 的值是不是超过 size_t 的表示范围，如果是，就不要调用 malloc，直接退出。而只有合法大小的数量才为之申请空间。

参考答案：

上述改动无法消除整数溢出漏洞。

分析：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;   /*将 arraysize 强制转换成 size_t*/
7      if (myarraysize!=arraysize)  /*然后比较转换后的 myarraysize 和 arraysize，如果二
                                      者不等，这表示转换过程有数据丢失，原数据 arraysize 在
                                      size_t 型范围内溢出，此时不能申请空间，直接退出*/
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 }
```

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

本题要点：

（1）x、y 是有符号数，ux、uy 是无符号数。在机器码层面，$[x]_{补}$=ux，$[y]_{补}$=uy，所以 x 和 y 的运算，机器级等价于 ux 和 uy 的运算，然后再把结果解析成为带符号数。

（2）x、y 向 ux、uy 赋值，所以，如果 x、y 是正数，则 ux、uy 的值直接等于 x、y。如果 x、y 是负数，则 ux、uy 的值等于 x、y 的补码（然后被当作无符号正整数看待）。

（3）对题目中各表达式的判别，主要是看运算表达式的值有没有溢出，有的话，可能正变负或负变正，这时候使得条件不能永真。

（4）特别注意-2 147 483 648 机器码表示 的特殊性

（5）不管是有符号还是无符号数，都是按照统一的规则做运算：补码运算

（5）本题只为各式判定是否永真（在任何情况下都为真），而不能简单表示为“对”或“错”

参考答案：

（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（这是因为在运算器上不去分有符号还是无符号，直接计算），符号位为 0，因而 x−1>0。 此时，(x-1<0) 和 x>0 两者都不成立。

（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。

（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**（机器数），而不是 2 147 483 648。所以仍是最小负数，和任意其它数比还是小。

（8）(int) (ux−uy) ==−(y−x)

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

（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。

（12）x*y==ux*uy

永真。根据第 2.7.5 节 P65 页内容可知，x*y 的低 32 位和 ux*uy 的低 32 位是完全一样的位序列。

（14）x*~y+ux*uy==−x

永真。这里~y 是按位取反，不是负运算。所以，$[-y=\sim y+1]_{补}$，即$[\sim y=-y-1]_{补}$。而根据（12），ux*uy=x*y，因此，等式左边为 x*(−y−1)+x*y=−x。

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：

（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

本题要点：

（1） 注意 IEEE 754 运算规则

（2） 整型数向 float 转换可能会发生数据丢失，应为 float 的尾数只有 23+1 位

（3） 浮点数运算存在舍入问题

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。

（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。

（5）dx*dy*dz == dz*dy*dx

非永真。浮点乘可能产生舍入，使得浮点乘不满足交换律等。

36. 在 IEEE 754 浮点数运算中，当结果的尾数出现什么形式时需要进行左规，什么形式时需要进行右规？如何进行左规，如何进行右规？

本题要点：

（1）本题旨在考查浮点数左规和右规的规则，照其基本规则回答即可。

（2）本题对于理解浮点数左规和右规的规则非常重要，重点关注

参考答案：

（1）对于结果为±1x. xx…x 的情况，需要进行右规。右规时，尾数右移一位，阶码加 1。右规操作可以表示为：$M_b \leftarrow M_b \times 2^{-1}$，$E_b \leftarrow E_b+1$。右规时注意以下两点：

① 尾数右移时，最高位“1”被移到小数点前一位作为隐藏位，最后一位移出时，要考虑舍入。

② 阶码加 1 时，直接在末位加 1。

（2）对于结果为±0.00…01x…x 的情况，需要进行左规。左规时，数值位逐次左移，阶码逐次减 1，直到将第一位“1”移到小数点左边。假定 k 为结果中“±”和最左边第一个 1 之间连续 0 的个数，则左规操作可以表示为：$M_b \leftarrow M_b \times 2^k$，$E_b \leftarrow E_b-k$。左规时注意以下两点：

① 尾数左移时数值部分最左边 k 个 0 被移出，因此，相对来说，小数点右移了 k 位。因为进行尾数相加时，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。

② 执行 $E_b \leftarrow E_b-k$ 时，每次都在末位减 1，一共减 k 次。

39. 采用 IEEE 754 单精度浮点数格式计算下列表达式的值。

（1）0.75+(– 65.25)　　（2）0.75–(– 65.25)

本题要点：

（1） 本题旨在考查手工模拟浮点数加、减过程，而不是只写出最后的答案，否则就没有意义了。

（2） 浮点数加、减过程中，要有**对阶、尾数运算、规格化、舍入**等关键步骤，所以回答本题时应予体现，否则，等于白做了

参考答案：

首先，将两个浮点数表示成规格化的 IEEE 754 格式的机器数。

（1） x = 0.75　= 0.110...0B　　$= (1.10...0)_2 \times 2^{-1}$，

y = – 65.25 = – 1000001.01000...0B = $(-1.00000101...0)_2 \times 2^6$

用 IEEE 754 标准单精度格式表示为：

$[x]_{浮}$ = 0 01111110 10...0 ：阶码 E_x= 01111110，尾数 M_x= 0(1). 1...0

$[y]_{浮}$ = 1 10000101 000001010...0 ：阶码 E_y= 10000101，尾数 M_y =1 (1).000001010...0

尾数 M_x 和 M_y 用原码表示，其中小数点前写了 2 位，第一位是符号位，第二位是隐藏位“1”，加了括号。

然后，详细描述计算机中进行浮点数加减运算的过程：（假定保留 **2 位附加位**：保护位和舍入位）

（1）0.75+ (– 65.25）

① 对阶：$[\Delta E]_{补}=E_x+[-E_y]_{补} \pmod{2^n}$ = 0111 1110 + 0111 1011 = 1111 1001，即 Δ E = –7。

根据对阶规则可知需对 x 进行对阶，结果为：$E_x = E_y$ = 10000101，M_x = 00.000000110...000

M_x 右移 7 位，符号不变，数值高位补 0，**隐藏位右移到小数点后面**，最后移出的 **2 位保留**

② 尾数相加：$M_b = M_x + M_y$ = 00. 000000110...0**00**+ 11. 000001010 ...0**00** （注意小数点在隐藏位后）

根据**原码加/减法运算规则**，得：00.000000110...000+ 11.000001010...000 = 11.000000100…0**00**

上式尾数中最左边第一位是符号位，其余都是数值部分，尾数后面两位是附加位（加粗）。

③ 规格化：尾数数值部分最高位为 1，因此不需要进行规格化。

④ 舍入：把结果的尾数 M_b 中最后两位附加位舍入掉，从本例来看，不管采用什么舍入法，结果都一样，都是把最后两个 0 去掉，得：M_b = 11.000000100…0

⑤ 溢出判断：在上述阶码计算和调整过程中，没有发生“阶码上溢”和“阶码下溢”的问题。因此，阶码 E_b = 10000101。

最后结果为 E_b = 10000101，M_b = 1(1).00000010…0，即：– 64.5。

（2） 0.75–(– 65.25）

① 对阶：$[\Delta E]_{补} = E_x + [-E_y]_{补} \pmod{2^n}$ = 0111 1110 + 0111 1011 = 1111 1001， Δ E = -7。

根据对阶规则可知需要对 x 进行对阶，结果为：$E_x = E_y$ = 10000110，M_x = 00.000000110...000

M_x 右移一位，符号不变，数值高位补 0，隐藏位右移到小数点后面，最后移出的位保留

② 尾数相加：$M_b = M_x – M_y$ = 00. 000000110...000 – 11. 000001010...000 （注意小数点在隐藏位后）

根据原码加/减法运算规则，得：00.000000110...000 – 11.000001010...000=01.00001000…000

上式尾数中最左边第一位是符号位，其余都是数值部分，尾数后面两位是附加位（加粗）。

③ 规格化：尾数数值部分最高位为 1，不需要进行规格化。

④ 舍入：把结果的尾数 M_b 中最后两位附加位舍入掉，从本例来看，不管采用什么舍入法，结果都一样，都是把最后两个 0 去掉，得：M_b = 01.00001000…0

⑤ 溢出判断：在上述阶码计算和调整过程中，没有发生“阶码上溢”和“阶码下溢”的问题。因此，阶码 E_b = 10000101。

最后结果为 E_b = 10000101，M_b = 0(1).00001000…0，即：+66。