## **《计算机系统基础习题解答》第二章**

3.实现下列各数的转换

（1）(25.8125)10=(？)2=(?)8=(?)16

（2）(101101.011)2=(?)10=(?)8=(?)16=(?)8421

（3）(0101 1001 0110.0011)8421=(？)10=(?)1=(?)16

（4）(4E.C)16=(?)10=(?)2

**[分析解答]**

（1）(25.8125)10=(11001.1101)2=(31.64)8=(19.D)6

（2）(101101.011)2=(45.375)10=(55.3)8=(2D.6)6=(0100 0101.0011 0111 0101)8421

（3）(0101 1001 0110.0011)8421=(596.3)10=(1001 0101 00.010011…)2=(254.4…)16

（4）(4E.C)16=(78.75)10=(100 1110.11)2

8421码是BCD码的一种。

如何将十进制整数与二进制之间转换？

如何将十进制小数与二进制之间转换？

4.假定机器数8位（1位符号，7位数值），写出下列各二进制的源码。0.1001，-0.1001，+1.0，-1.0，+0.010100，-0.010100，+0，-0

**[分析解答]：**

|  |  |
| --- | --- |
| 数值 | 原码 |
| +0.1001 | 0.1001000 |
| -0.1001 | 1.1001000 |
| +1.0 | overflow |
| -1.0 | overflow |
| +0.010100 | 0.0101000 |
| -0.010100 | 1.0101000 |
| +0 | 0.0000000 |
| -0 | 1.0000000 |

5.假定机器数8位（1位符号，7位数值），写出下列各二进制的补码和移码。

+1001，-1001，+1，-1，+10100，-10100，+0，-0

**[分析解答]：**

|  |  |  |
| --- | --- | --- |
| 数值 | 补码 | 移码（偏置常数 1 0000000） |
| +1001 | 0 0001001 | 10001001 |
| -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 |

如何从补码求原码？依然可以用取反加1.

6.已知补码求原码。

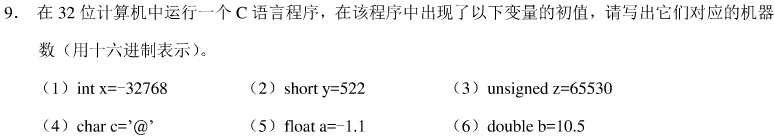
（1）[X]补=1110 0001 （2）[X]补=1000 0000

（3）[X]补=0111 1111 （4）[X]补=1111 1111

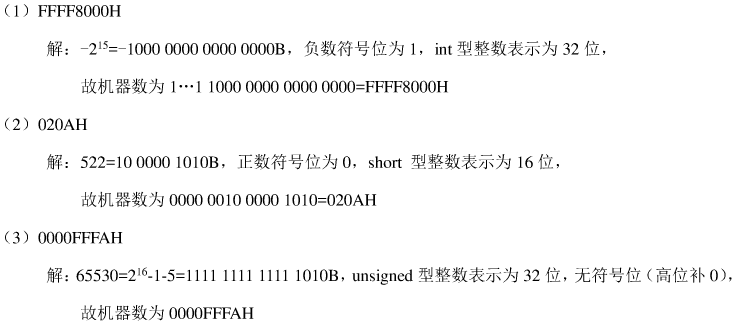
**[分析解答]：**

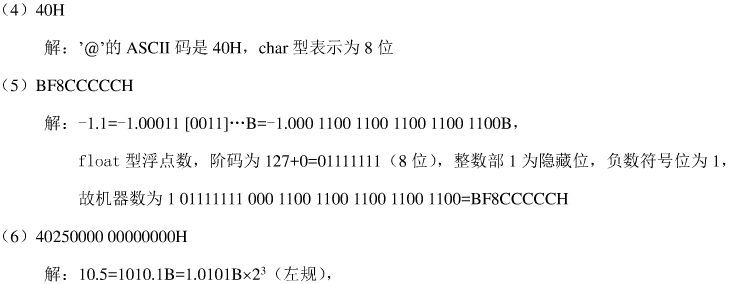
（1）X=-1 1111B=-31 （2）X=-1000 0000**B**=-128

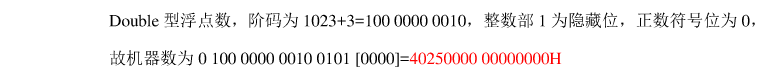
（3）X=111 1111B=31 （4）X=-0000 0001B=-1



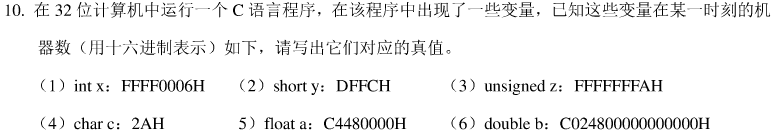
**[分析解答]：**



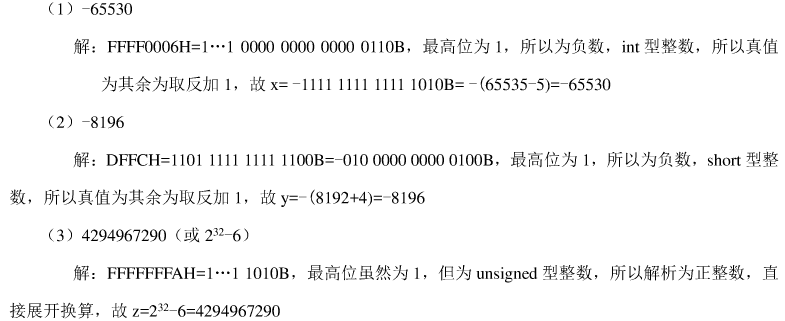


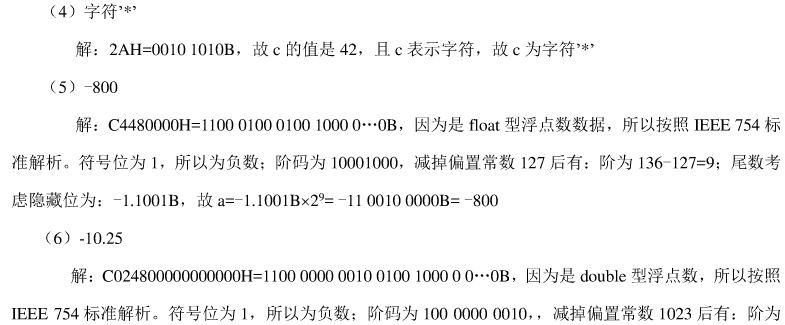


记忆方法：单精度阶码8位、双精度阶码11位，除了符号位之外其他都是尾数。偏置常数分别是127、1023，即2^7-1和2^10-1。



**[分析解答]：**







1. 已知C语言中的的按位异或运算XOR用“^”表示。对于任意一个位序a，存在a^a=0。C语言程序可以利用这个特性来实现两个数值交换的功能。以下是实现该功能的C语言函数：

Void xor\_swap(int \*x, int \*y)

{

\*y=\*x^\*y;

\*x=\*x^\*y;

\*y=\*x^\*y;

}

假定执行该函数时\*x和\*y的初始值分别为a和b，即\*x=a且\*y=b，请说明每个步执行结束后x和y各自指向的内存单元中的内容分别是什么？

**[分析解答]：**

第一步结束后，x和y指向的内存单元内容各为a和a^b；

第二步结束后，x和y指向的内存单元内容各为b和a^b；

第三步结束后，x和y指向的内存单元内容各为b和a。

备注：A XOR B XOR C=A XOR （B XOR C）；

这种交换的好处是节省内存空间。

1. 【？】假定某个实现组元素倒置的函数reverse\_array调用了第13题给出的xor\_swap函数：

Void reverse\_array(int a[], int len)

{

Int left, right=len-1;

For(left=0; left<=right;left++;right)

Xor\_swap(&a[left],&a[right]);

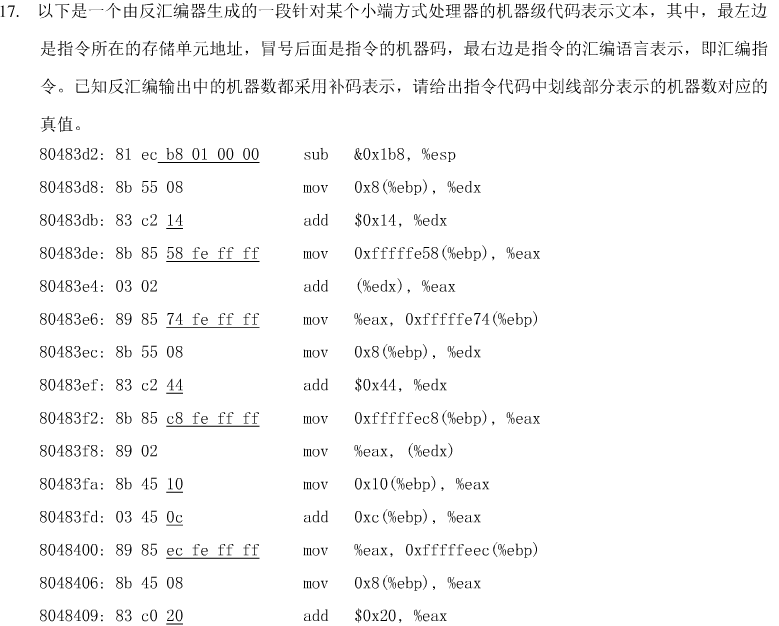
}

当len为偶数时，reverse\_array函数的执行没问题。但是，当len为奇数时，函数的执行结果不正确。请问，当len为奇数时会出现什么问题？最后一次循环中的left和right各取什么值？最后一次循环中调用xor\_swap函数后的返回值是什么？对reverse\_array函数做怎样的改动就可以消除该问题？

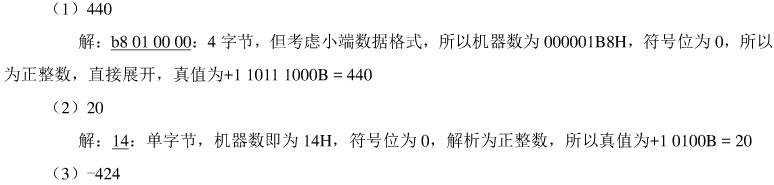
**[分析解答]：**

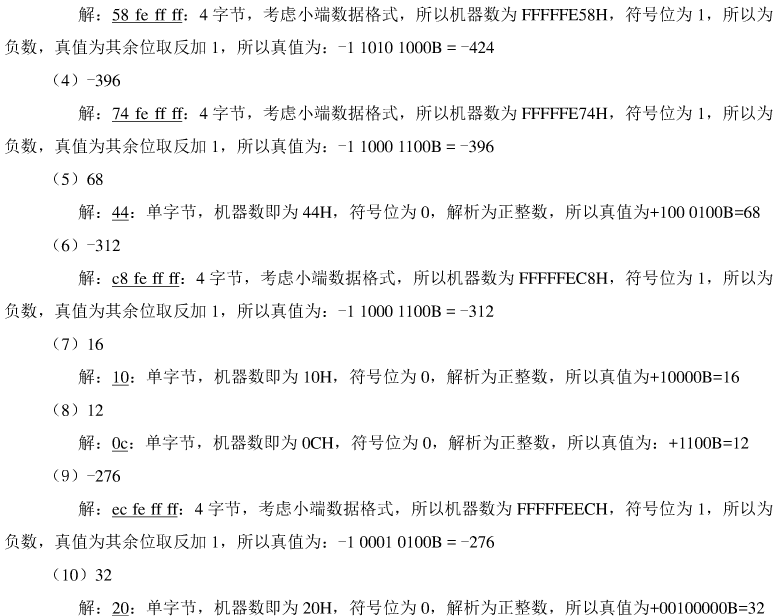
当len为奇数时，最后一次循环执行的是将最中间的数与自己进行交换，即left和right都指向最中间的数组元素，因而在调用xor\_swap函数过程中的每一步执行\*x^\*y时结果都是0，并将0写入最中间的数组元素，从而改变了原来的数值。

可以将for循环中的终止条件改为“left<right”，这样，当len为奇数时，最中间的数组元素不动。



**[分析解答]：**





1. 考虑以下C语言程序代码：

Int fun1(unsigned word)

{

Return (int)((word<<24)>>24);

}

Int func2(unsigned word)

{

Return ((int)word<<24)>>24;

}

假设在一个32位机器上执行这些函数，sizeof(int)=4。说明函数func1和func2的功能，填写下表，并给表中的“异常”数据做出说明。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| w | | Func1(w) | | Func2(w) | |
| 机器数 | 值 | 机器数 | 值 | 机器数 | 值 |
|  | 127 |  |  |  |  |
|  | 128 |  |  |  |  |
|  | 255 |  |  |  |  |
|  | 256 |  |  |  |  |

**[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位都变成和原来的第25位相同。

根据程序执行的结果填，其中机器数用十六进制表示。

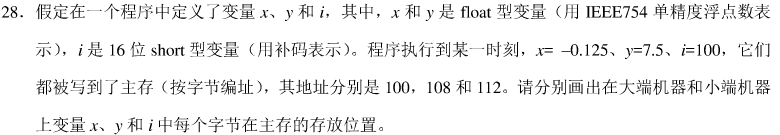
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| w | | Func1(w) | | Func2(w) | |
| 机器数 | 值 | 机器数 | 值 | 机器数 | 值 |
| 0000 007FH | 127 | 0000 007FH | +127 | 0000 007FH | +127 |
| 0000 0080H | 128 | 0000 0080H | +128 | FFFF FF80H | -128 |
| 0000 00FFH | 255 | 0000 00FFH | +255 | FFFF FFFFH | -1 |
| 0000 0100H | 256 | 0000 0000H | 0 | 0000 0000H | 0 |

20.填写下表，注意对比无符号数和带符号整数的乘法结果，以及截断操作前、后的结果。

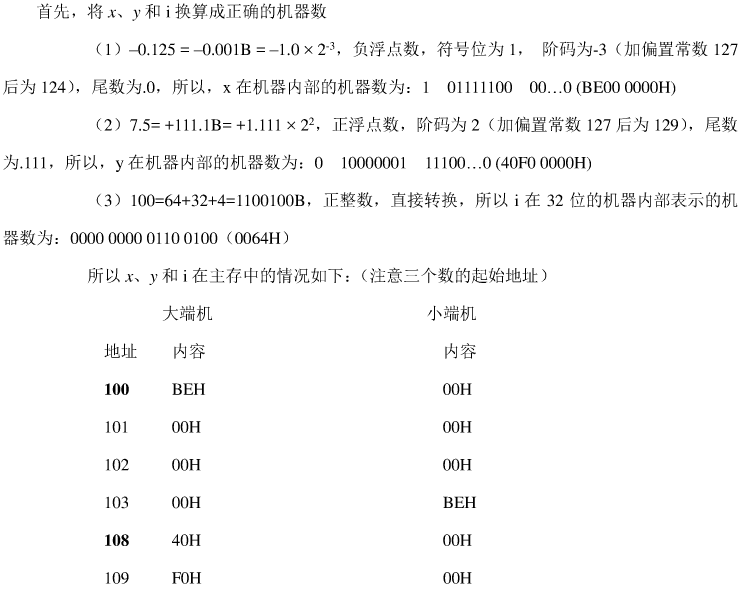
|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 模式 | x | | y | | X\*y截断前 | | X\*y截断后 | |
| 机器数 | 值 | 机器数 | 值 | 机器数 | 值 | 机器数 | 值 |
| 无符号数 | 110 |  | 010 |  |  |  |  |  |
| 二进制补码 | 110 |  | 010 |  |  |  |  | - |
| 无符号数 | 001 |  | 111 |  |  |  |  |  |
| 二进制补码 | 001 |  | 111 |  |  |  |  |  |
| 无符号数 | 111 |  | 111 |  |  |  |  |  |
| 二进制补码 | 111 |  | 111 |  |  |  |  |  |

**[分析解答]：**根据无符号乘法运算和补码乘法运算算法，填表如下：

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 模式 | x | | y | | X\*y截断前 | | X\*y截断后 | |
| 机器数 | 值 | 机器数 | 值 | 机器数 | 值 | 机器数 | 值 |
| 无符号数 | 110 | 6 | 010 | 2 | 001100 | 12 | 100 | 4 |
| 二进制补码 | 110 | -2 | 010 | +2 | 111100 | -4 | 100 | -4 |
| 无符号数 | 001 | 1 | 111 | 7 | 000111 | 7 | 111 | 7 |
| 二进制补码 | 001 | +1 | 111 | -1 | 111111 | -1 | 111 | -1 |
| 无符号数 | 111 | 7 | 111 | 7 | 110001 | 49 | 001 | 1 |
| 二进制补码 | 111 | -1 | 111 | -1 | 000001 | +1 | 001 | +1 |

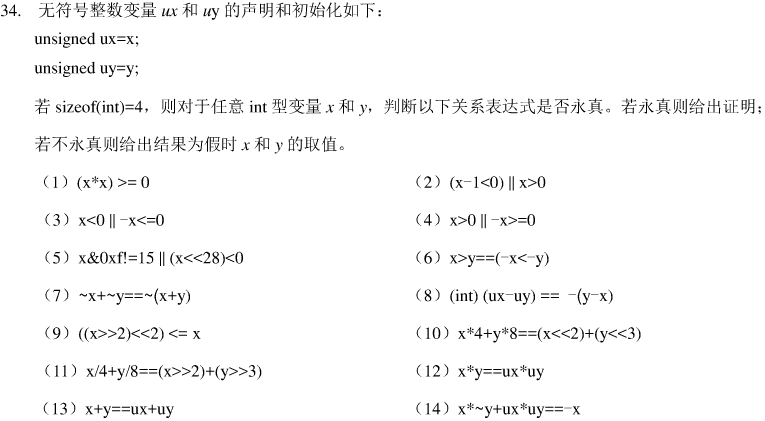


**[分析解答]：**









**[分析解答]：**

