8. 设A4~A1和B4~B1分别是4位加法器的两组输入，C0为低位来的进位。当加法器分别采用串行进位和先行进位（即并行进位）时，写出4个进位C4 ~C1的逻辑表达式。

**【分析解答】**

串行进位：

C1=A1C0+B1C0+A1B1

C2=A2C1+B2C1+A2B2

C3=A3C2+B3C2+A3B3

C4=A4C3+B4C3+A4B4

并行进位：

C1=A1B1+(A1+B1)C0

C2=A2B2+(A2+B2)A1B1+(A2+B2)(A1+B1)C0

C3=A3B3+(A3+B3)A2B2+(A3+B3)(A2+B2) A1B1+(A3+B3)(A2+B2)(A1+B1)C0

C4=A4B4+(A4+B4)A3B3+(A4+B4)(A3+B3)A2B2+(A4+B4)(A3+B3)(A2+B2)A1B1

+(A4+B4) (A3+B3)(A2+B2)(A1+B1)C0

10. 已知*x* = 10，*y* = – 6，采用6位机器数表示。请按如下要求计算，并把结果还原成真值。

1. 求[*x* + *y*]补，[*x*–*y*]补。
2. 用原码一位乘法计算[*x*×*y*]原。
3. 用布斯乘法计算[*x*×*y*]补。
4. 用不恢复余数法计算[*x*/*y*]原的商和余数。

**【分析解答】**

先将x和y转换为二进制数。x=10=+01010B，y = –6 = –00110B。

（1）[x]补=0 01010 B，[y]补=1 11010 B，[–y]补=0 00110 B。

[x+y]补= [x]补+[y]补=0 01010 B + 1 11010 B =0 00100 B，因此，x+y=4。

[x–y]补= [x]补+[–y]补=0 01010 B + 0 00110 B = 0 10000 B，因此，x–y=+16。

（2）[x]原 = 0 01010 B，[y]原=1 00110 B。

将符号和数值部分分开处理。乘积的符号为0⊕1=1，数值部分采用无符号整数乘法算法计算01010×00110的乘积。原码一位乘法过程描述如下：初始部分积为0，在乘积寄存器前增加一个进位位。每次循环首先根据乘数寄存器中最低位决定+X还是+0，然后将得到的新进位、新部分积和乘数寄存器中的部分乘数一起逻辑右移一位。共循环5次，最终得到一个10位无符号数表示的乘积00001 11100 B。所以，[x×y]原=1 00001 11100 B，因此，x×y= –60。若结果取6位原码，则因为高5位00001是一个非0数，所以，结果溢出，即[x×y]原≠1 11100。验证：6位原码的表示范围为-31～+31，显然乘积–60不在其范围内，结果应该溢出。（过程略）

（3）[x]补=0 01010 B，[–x]补=1 10110 B， [y]补=1 11010 B。

采用Booth算法时，符号和数值部分一起参加运算，最初在乘数后面添0，初始部分积为0。每次循环先根据乘积寄存器中最低两位决定执行+X、–X还是+0操作，然后将得到的新的部分积和乘数寄存器中的部分乘数一起算术右移一位。–X用+[–x]补实现。共循环6次。最终得到一个12位补码表示的乘积111111 000100 B，所以，[x×y]补=111111 000100 B，因此，x×y= –60。若结果取6位补码，则根据乘积低6位000100的符号位为0，而高6位为111111，不等于全0，说明结果溢出，即[x×y]补≠0 00100。验证：6位补码的表示范围为-32～+31，显然乘积–60不在其范围内，结果应该溢出。（过程略）

（4）[x]原 =0 01010 B，[y]原=1 00110 B。

将符号和数值部分分开处理。商的符号为0⊕1=1，数值部分采用无符号整数除法算法计算01010 B和00110 B的商和余数。无符号整数不恢复余数除法过程描述如下：初始中间余数为0 00000 01010 0，其中，最高位为添加的符号位，用于判断余数是否大于等于0；最后一位0为第一次上的商，该位商只是用于判断结果是否溢出，不包含在最终的商中。因为结果肯定不溢出，所以该位商可以直接上0，并先做一次–Y操作得到第一次中间余数，然后进入循环。每次循环首先将中间余数和商一起左移一位，然后根据上一次上的商（或余数的符号）决定执行+Y还是–Y操作，以得到新的中间余数，最后根据中间余数的符号确定上商为0还是1。–Y采用+[–y]补的方式进行。整个循环内执行的要点是“正、1、减；负、0、加”。共循环5次。最终得到一个6位无符号数表示的商0 00001和余数00100，其中第一位商0必须去掉，添上符号位后得到最终的商的原码表示为1 00001，余数的原码表示为0 0100。因此，x/y的商为–1，余数为4。（过程略）

11. 若一次加法需要1ns，一次移位需要0.5ns。请分别计算用一位乘法、两位乘法计算两个8位无符号二进制数乘积时所需的时间。

**【分析解答】**

对于8位无符号二进制数相乘，一位乘法需8次右移，8次加法，共计12ns；二位乘法需4次右移，4次加法，共计6ns。

16. 对于3.4.1节中例3.8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。

**【分析解答】**

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. }

17. 假设一次整数加法、一次整数减法和一次移位操作都只需一个时钟周期，一次整数乘法操作需要10个时钟周期。若x为一个整型变量，现要计算55\*x，请给出一种计算表达式，使得所用时钟周期数最少。

**【分析解答】**

根据表达式55\**x*=(64-8-1)\**x*=64\**x*-8\**x*-*x*可知，完成55\**x*只要两次移位操作和两次减法操作，共4个时钟周期。若将55分解为32+16+4+2+1，则需要4次移位操作和4次加法操作，共8个时钟周期。上述两种方式都比直接执行一次乘法操作所用的时钟周期数少。

21. 分别给出不能精确用IEEE 754单精度和双精度格式表示的最小正整数。

**【分析解答】**

不能精确用IEEE 754单精度和双精度格式表示的最小正整数分别是224+1、253+1。

（不超过24位的正整数都可以用float类型精确表示。

224+1=1.0000 0000 0000 0000 0000 0001×224）

22. 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？

**【分析解答】**

IEEE 754浮点数加、减、乘、除运算结果的尾数不可能大于等于4，因而尾数溢出的情况只可能是±1x .xx…x的形式。

(1) 对于结果为±1x .xx…x的情况，需进行右规。即尾数右移一位，阶码加1。右规操作可以表示为Mb←Mb×2-1，Eb←Eb+1。

右规时注意以下两点：a) 尾数右移时，最高位“1”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。b) 阶码加1前，先判断阶码是否为全1或1…10，若是，则发生阶码上溢导致结果溢出；否则，直接在阶码末位加1。

(2) 对于结果为±0.0…01x…x的情况，需进行左规。即数值位逐次左移，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“±”和左边第一个1之间连续0的个数，则左规操作可以表示为Mb←Mb×2k，Eb←Eb–k。

左规时注意以下几点：a) 尾数左移时数值部分最左k个0被移出，因此，相对来说，小数点右移了k位。b) 每次减1前，先判断阶码是否为全0，若是全0，则发生阶码下溢，不再进行左规操作，结果为非规格化数（尾数不为0时）或0（尾数为0时）；否则，通过执行+11…1进行减1操作。c) 减1操作最多k次。

25. 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）

**【分析解答】**

x=0.75=0.11B=(1.10…0)2×2-1，y= –65.25= –1000001.01B = (–1.00000101…0) 2 ×26。

用IEEE 754标准单精度格式表示为[x]浮=0 01111110 10…0，[y]浮=1 10000101 000001010…0。假定用Ex、Ey分别表示[x]浮和[y]浮中的阶码，Mx、My分别表示[x]浮和[y]浮中的尾数，即Ex=0111 1110，Mx=0(1).10…0，Ey=1000 0101，My=1(1).000001010…0。尾数Mx和My的小数点前面有两位，第一位为数符，第二位加了括号，是隐藏位“1”。以下是机器中浮点数加/减运算过程（假定保留两位附加位：保护位和舍入位）。

① 对阶。

[ΔE]补=Ex+[–E y]补=0111 1110 + 0111 1011=1111 1001(mod 28)，ΔE= –7，故需对x进行对阶，结果为Ex=Ey=1000 0101，Mx=00.000000**1**10…0 **00**，即将x的尾数Mx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，最后移出的高两位保留。

② 尾数相加。

Mb=Mx+My=00.000000**1**10…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（加法运算规则为“同号求和，异号求差”，最后结果可能需要求补），得：00.000000**1**10…0 **00**+11.000001010…0 **00**=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，尾数最后两位是附加位。（0.00000011+0.11111011=0.11111110，没有进位，需求补，求补后为1.00000010）

③ 规格化。

根据所得尾数的形式，数值部分最高位为1，所以不需要进行规格化。

④ 舍入。

将结果的尾数Mb中最后两位附加位舍入，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：Mb = 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。

最终结果为Eb=1000 0101，Mb=1(1).00000010…0，即结果为(–1.0000001)2×26 = –64.5。