3 考虑以下C语言程序代码：

```
int   func1(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，二进制补码表示int（其实题干应该说清楚int和unsigned的位数）。说明函数func1和func2的功能，并填表，给出对表中“异常”数据的说明。

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，因为先转换为有符号数，再左移24位（此时左边第一位变为原来的第25位），最后算术右移，高位补符号，即高24位都变成和原来第25位相同。（下表用16进制表示机器数）

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**

- 因为逻辑左移和算术左移的结果完全相同，所以，函数func1和func2中第一步左移24位得到的结果完全相同，所不同的是右移24位后带来的结果。
- 上述表中，蓝色数据是一些“异常”结果。当w=128和255时，第25位正好是1，因此函数func2执行的结果为一个负数，出现了“异常”。当w=256时，低8位为00，高24位为非0值，左移24位后使得有效数字被移出，因而发生了“溢出”，使得出现了“异常”结果0。

4 填表，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	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

①对于两个相同的机器数，作为无符号数进行乘法运算和作为带符号整数进行乘法运算，因为其所用的乘法算法不同，所以，乘积的机器数可能不同。但是，不同的仅是乘积中的高n位，而低n位完全一样。

②对于n位乘法运算，若截取2n位乘积的低n位作为最终结果，则都有可能结果溢出，即n位数字无法表示正确的乘积。带符号整数乘积截断后也可能溢出，例如，011×011=001001，截断后011×011=001，显然截断后的结果发生了溢出。

③表中蓝色是截断后发生溢出的情况。溢出判断方法：无符号整数——若乘积中高n位为全0，则截断后的低n位乘积不发生溢出，否则溢出；带符号整数——若高n位中的每一位都等于低n位中的第一位，则截断后的低n位乘积不发生溢出，否则溢出。

5 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;
          }
```

对于x，“x左移4位”即：x=16x，然后有一条“减法”指令，即x=16x–x=15x，根据源程序知M=15；

对于y，有一条“y右移2位”指令，即y=y/4，根据源程序知N=4。

但是，对于带符号整数来说，采用算术右移时，高位补符号，低位移出。
因此，当符号位为0时，与无符号整数相同，采用移位方式和直接相除得到的商完全一样。

当符号位为1时，若低位移出的是非全0，则说明不能整除。这种情况下，移位得到的商与直接相除得到的商不一样，需要进行校正（在右移前，先将*x*加上偏移量(2^k-1)，然后再右移*k*位）。例如，上述函数optarith中，在执行y>>2之前加了一条语句“if (y < 0) y+= 3;”，以对y进行校正。

6

- 设A_4~A_1和B_4~B_1分别是4位加法器的两组输入，C_0为低位来的进位。当加法器分别采用串行进位和先行进位时，写出4个进位C_4~C_1的逻辑表达式。

串行进位：

$C_1=A_1C_0+B_1C_0+A_1B_1$

$C_2=A_2C_1+B_2C_1+A_2B_2$

$C_3=A_3C_2+B_3C_2+A_3B_3$

$C_4=A_4C_3+B_4C_3+A_4B_4$

并行进位：

$C_1=A_1B_1+(A_1+B_1)C_0$

$C_2=A_2B_2+(A_2+B_2)A_1B_1+(A_2+B_2)(A_1+B_1)C_0$

$C_3=A_3B_3+(A_3+B_3)A_2B_2+(A_3+B_3)(A_2+B_2)A_1B_1+(A_3+B_3)(A_2+B_2)(A_1+B_1)C_0$

$C_4=A_4B_4+(A_4+B_4)A_3B_3+(A_4+B_4)(A_3+B_3)A_2B_2+(A_4+B_4)(A_3+B_3)(A_2+B_2)A_1B_1$

$+(A_4+B_4)(A_3+B_3)(A_2+B_2)(A_1+B_1)C_0$

7

（1）$[x]_{补}$=0101B， $[y]_{补}$=1101B， $[-y]_{补}$=0011B。

验证真值：x=+5， y=–3， x+y=2， x-y=8

$[x+y]_{补}$= $[x]_{补}$+$[y]_{补}$=0101B +1101B =(1)0010B， 因此， x+y=2。两个不同符号数相加，结果一定不会溢出。。

$[x-y]_{补}$= $[x]_{补}$+$[-y]_{补}$=0101B +0011B = (0)1 000B， 因此， x–y=–8。

两个正数相加结果为负，发生了溢出。

验证：8>最大可表示数7， 故溢出。

（2）$[x]_{原}$=0101B，$[y]_{原}$=1101B。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**无符号数乘法算法**计算101×101的乘积。共循环3次，最终得到一个8位无符号数表示的乘积1 0011001B。（自动补齐8位，最高位数值位直接添加0）

C	P	Y	说明
0	000	101	$P_0=0$
	+101		$y_0=1$，$+X$
0	101	101	C,P 和 Y 同时右移一位
0	010	110	得 P_1
	+000		$y_1=0$，$+0$
0	010	110	C,P 和 Y 同时右移一位
0	001	011	得 P_2
	+101		$y_2=1$，$+X$
0	110	011	C,P 和 Y 同时右移一位
0	011	001	得 P_3

符号位为1，因此，$[x \times y]_{原}$=1 0011001，因此，x×y=–25。

若结果取4位原码1 001，则因为乘积数值部分高3位为011，是一个非0数，所以，结果溢出。验证：4位原码的表示范围为-7～+7，显然乘积–25不在其范围内，结果应该溢出。

（3）$[x]_{补}$=0 101B，$[-x]_{补}$=1011B， $[y]_{补}$=1101B。

采用**MBA（基4布斯）算法**时，符号和数值部分一起参加运算，在乘数后面添0，初始部分积为0，并在部分积前加一位补充符号位0（C那一列）。每次循环先根据乘积寄存器中最低3位决定执行+X、+2X、–X、–2X、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**一起算术右移两位**。–X和–2X分别采用$+[-x]_{补}$和$+2[-x]_{补}$的方式进行。共循环两次。最终得到一个8位补码表示的乘积1111 0001 B。

$[x\times y]_{补}$=1111 0001，因此，x×y= –15。 **如果仅保留低4位，则溢出**

C	P	Y Y_{-1}	说明
0	0000	1101 0	$P_0=0$
+0	0101		$y_1y_0y_{-1}$=010，$+X$
0	0101	1101	C,P 和 Y 同时右移两位
0	0001	0111 01	得 P_1
+1	1011		$y_3y_2y_1$=110，$-X$
1	1100	0111 01	C,P 和 Y 同时右移两位
1	1111	0001 11	得 P_2

（4）$[x]_{原}$=0101B，$[y]_{原}$=1101B。将符号和数值部分分开处理。

将符号和数值部分分开处理。商的符号为$0\oplus 1=1$，数值部分采用无符号数除法算法计算101B和101B的商和余数。

最高位需添加符号位，

整数除法，所以被除数高位补0

为了实现补码加减，余商都补足4位，但其实商只有最后三位是有意义的。第一个绿色的0代表不溢出。第一个红色的0相当于符号位，而且肯定是0

验证：0101 / 0101 = 1 余 0　　　　–D = 1011

R：被除数（中间余数）；D：除数

	R: 0000 0101
R = R–D	1011
	1011 0101
sl R, 0	R: 0110 1010
R = R+D	0101
	1011 1010
sl R, 0	R: 0111 0100
R = R+D	0101
	1100 0100
sl R, 0	R: 1000 1000
R = R+D	0101
	1101 1000
sl R, 0	R: 1011 0000
R = R+D	0101
	0000 $\boxed{0001}$

第1次上商为“试商”，仅仅表示不溢出

不恢复余数法、加减交替法

负，0，加

正，1，减

第1位商为绿色0，在最后（第5次）被左移出去，并加上最后一位商

且最后一次上商1，余数无需恢复就是正确的。

可以省略第一步，也就是第一个绿色的0直接加入，
并把初始的余商0000 0101一起左移一次变成0000 1010

验证：0101 / 0101 = 1 余 0　　　　–D = 1011

R：被除数（中间余数）；D：除数

	R: 0000 1010
R = R–D	1011
	1011 1010
sl R, 0	R: 0111 0100
R = R+D	0101
	1100 0100
sl R, 0	R: 1000 1000
R = R+D	0101
	1101 1000
sl R, 0	R: 1011 0000
R = R+D	0101
	0000 $\boxed{0001}$

第1次直接上商0并左移，因为肯定不溢出

不恢复余数法、加减交替法

负，0，加

正，1，减

第1位商为绿色0，在最后（第4次）被左移出去，并加上最后一位商

且最后一次上商1，余数无需恢复就是正确的。

还可以这样做：为了实现补码加减，余数补足4位，商仅用3位
最后获得三位商均为数值位。第一个绿色的0代表不溢出。

验证：0101 / 0101 = 1 余 0　　　　–D = 1011

R：被除数（中间余数）；D：除数

	R: 0000 101
R = R–D	1011
	1011 101
sl R, 0	R: 0111 010
R = R+D	0101
	1100 010
sl R, 0	R: 1000 100
R = R+D	0101
	1101 100
sl R, 0	R: 1011 000
R = R+D	0101
	0000 000
sl R, 0	R: 0000 001

第1次上商为“试商”，仅仅表示不溢出

不恢复余数法、加减交替法

负，0，加

正，1，减

第1位商为绿色0，在最后（第4次）被左移出去，并加上最后一位商

且最后一次上商1，余数无需恢复就是正确的。

（5）$[x]_{补}$=0101B，$[y]_{补}$=1101B。

+y 即为+[1101]， -y 即为+[0011]

补码不恢复余数除法过程描述如下：**初始中间余数（被除数）高位补0**后为0000 0101，整个循环内执行的要点是“同、1、减；异、0、加”。共循环4次，得到商1110和余数0010。最终根据情况需要对商和余数进行修正。（**Y符号为1，每次判断中间余数和1的同异**，同则商为1，然后左移，然后减法。 异号则商为0，然后左移，然后加法。）

余数寄存器 R	余数/商寄存器 Q	
0000	0101	开始 $R_0=X$
+1101		被除数和除数异号，做加法
1101	0101	同、1、减
1010	1011	$2R_1$（R 和 Q 同时左移，Q4=1）
+0011		$R_2=2R_1-Y$
1101	1011	同、1、减
1011	0111	$2R_2$（R 和 Q 同时左移，Q3=1）
+0011		$R_3=2R_2-Y$
1110	0111	同、1、减
1100	1111	$2R_3$ （R 和 Q 同时左移，Q2=1）
+0011		$R_4=2R_3-Y$
1111	1111	同、1、减
1111	1111	$2R_4$（R 和 Q 同时左移，Q1=1）
+0011		$R_5=2R_3-Y$
0010	1111	异、0
0010	1110	最后一次 Q 寄存器左移（最高位商 1 去掉，Q0=0）

商的修正：最后一次Q寄存器左移一位，将最高位q_n移出，最低位置商q_0=0。若被除数与除数同号，Q中就是真正的商；否则，将Q中商的末位加1。故商为1110+1=1111B。

余数的修正：若余数符号同被除数符号，则不需修正；否则，按下列规则进行修正：当被除数和除数符号相同时，最后余数加除数；否则，最后余数减除数。故余数为0010B。

商的为1111（-1），余数为0010（2）。验证：“除数×商+余数=被除数”进行验证，得(-3)×(-1)+2 =5。