第 6 章　运算方法和运算部件

作业：习题 3、4、5、6、7

补充题目（复习第一章的内容）：

考虑下列 C 语言程序代码：

```
int i =65535;
short si = (short)i;
int j = si;
```

假定上述程序段在某 32 位机器上执行，sizeof (int)=4，则变量 i、si 和 j 的值分别是多少？为什么？

【分析解答】

在一台 32 位机器上执行上述代码段时，i 为 32 位补码表示的定点整数，第 2 行要求强行将一个 32 位带符号数截断为 16 位带符号整数，65535 的 32 位补码表示为 0000 FFFFH，截断为 16 位后变成 FFFFH，它是–1 的 16 位补码表示，因此 si 的值是–1。再将该 16 位带符号整数扩展为 32 位时，就变成了 FFFF FFFFH，它是–1 的 32 位补码表示，因此 j 的值也为–1。也就是说，i 的值原来为 65535，经过截断、再扩展后，其值变成了–1。

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，sizeof (int)=4。说明函数 func1 和 func2 的功能，并填写表 6.2，给出对表中“异常”数据的说明。

表 6.2　题 3 用表

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 位都变成和原来第 25 位相同。

根据程序执行的结果填表如下表，表中机器数用十六进制表示。

题 3 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**

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 位后的结果不同。

上述表中，加粗数据是一些“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. 填写表 6.3，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。

表 6.3　题 4 用表

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】

根据无符号数乘法运算和补码乘法运算算法，填写表 6.3 后得到下表。

题 4 中填入结果后的表

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

对上表中结果分析如下：

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，从表中看出，截断后的乘积是一样的，也即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，无论是无符号数乘法还是带符号整数乘法，若截取 2n 位乘积的低 n 位作为最终的乘积，则都有可能结果溢出，即 n 位数字无法表示正确的乘积。虽然表中给出的带符号整数乘积截断后都没有发生溢出，但实际上还是存在溢出的情况，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 表中加粗的地方是截断后发生溢出的情况。可以看出，对于无符号整数乘法，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；对于带符号整数乘法，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;
}
```

【分析解答】

对反编译结果进行分析，可知：对于 x，指令机器代码中有一条“x 左移 4 位”指令，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，当 y<0 时，对于有些 y，执行 y>>2 后的值并不等于 y/4。例如，当 y= –1 时，在反编译函数 optarith 中执行 y>>2 时，因为–1 的机器数为全 1，左移两位后还是全 1，也即–1>>2= –1，结果为–1；而原函数 arith 中执行 y/4 时，因为–1/4=0，得到结果为 0。

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。例如，对于−3/2，假定补码位数为 4，则进行算术右移操作 1101>>1=1110.1B（小数点后面部分移出）后得到的商为−2，而精确商是−1.5，即整数商应为−1。显然，算术右移后得到的商比精确商少了 0.5，相当于朝−∞方向进行了舍入，而不是朝零方向舍入。因此，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。

校正的方法是，对于带符号整数 x，若 x<0，则在右移前，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。

【分析解答】

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：$C_1=A_1B_1+(A_1+B_1)C_0$

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

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

$C_4=A_4B_4+(A_4+B_4)A_3B_3+(A_4+B_4)(A_3+B_3)A_2B_2+(A_4+B_4)(A_3+B_3) (A_2+B_2)A_1B_1$
$\quad +(A_4+B_4) (A_3+B_3)(A_2+B_2)(A_1+B_1)C_0$

7．请按如下要求计算，并把结果还原成真值。

（1） 设［x］$_{补}$= 0101、［y］$_{补}$=1101，求［x+y］$_{补}$，［x–y］$_{补}$。

（2） 设［x］$_{原}$= 0101、［y］$_{原}$=1101，用原码一位乘法计算［x×y］$_{原}$。

（3） 设［x］$_{补}$= 0101、［y］$_{补}$=1101，用 MBA（基 4 布斯）乘法计算［x×y］$_{补}$。

（4） 设［x］$_{原}$= 0101、［y］$_{原}$=1101，用不恢复余数法计算［x/y］$_{原}$的商和余数。

（5） 设［x］$_{补}$= 0101、［y］$_{补}$=1101，用不恢复余数法计算［x/y］$_{补}$的商和余数。

【分析解答】

（1）[x]$_{补}$=0 101B，[y]$_{补}$=1 101B，[–y]$_{补}$=0 011B。

[x+y]$_{补}$= [x]$_{补}$+[y]$_{补}$=0 101B + 1 101B =(1)0 010B，因此，x+y=2。

两个不同符号数相加，结果一定不会溢出。验证：x=+101B=5，y=–011B=–3，x+y=2。

[x–y]$_{补}$= [x]$_{补}$+[–y]$_{补}$ =0 101B + 0 011B = (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：5–(–3)=8>最大可表示数 7，故溢出。

（2）[x]$_{原}$ = 0 101B，[y]$_{原}$=1 101B。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。

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 3 次，最终得到一个 8 位无符号数表示的乘积 1 0011001B。

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

符号位为 1，因此，［x×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 0011，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，$[-x]_{补}$=1 011B， $[y]_{补}$=1 101B。

采用 MBA 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符号位 0。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 3 次。最终得到一个 8 位补码表示的乘积 1111 0001 B。

C	P	Y	Y_{-1}	说明
0	0 0 0 0	1 1 0 1	0	$P_0=0$
+0	0 1 0 1			$y_1y_0y_{-1}=010$，$+X$
0	0 1 0 1			C, P 和 Y 同时右移两位
0	0 0 0 1	0 1 1 1	0	得 P_1
+1	1 0 1 1			$y_3y_2y_1=110$，$-X$
1	1 1 0 0			C, P 和 Y 同时右移一位
1	1 1 1 1	0 0 0 1		得 P_2

$[x \times y]_{补}$ =1111 0001，因此，x×y= –15。

（4）$[x]_{原}$ = 0 101B，$[y]_{原}$=1 101B。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⊕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。无符号数不恢复余数除法过程描述如下：初始中间余数为 0 000 101 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 3 次。最终得到一个 3 位无符号数表示的商 0001 和余数 0000，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 1 001，余数的原码表示为 0 000。因此，x/y 的商为–1，余数为 0。

余数寄存器 R	余数/商寄存器 Q	说　明
0　0 0 0	1 0 1 □	开始 $R_0=X$
+ 1　0 1 1		$R_1=X-Y$
1　0 1 1	1 0 1　0	$R_1<0$，故 $q_3=0$，没有溢出
0　1 1 1	0 1 0 □	$2R_1$（R 和 Q 同时左移，空出一位商）
+ 0　1 0 1		$R_2=2R_1+Y$
1　1 0 0	0 1 0 0	$R_2<0$，则 $q_2=0$
1　0 0 0	1 0 0 □	$2R_2$ （R 和 Q 同时左移，空出一位商）

+0 101		$R_3=2R_2+Y$
1 101	1000	$R_3<0$，则 $q_1=0$
1 011	000 □	$2R_3$ （R 和 Q 同时左移，空出一位商）
+0 101		$R_4=2R_3-Y$
0 000	0001	$R_4>0$，则 $q_0=1$

商的最高位为 0，说明没有溢出，商的数值部分为 001。所以，［x/y］$_{原}$=1 001（最高位为符号位），余数为 0。

（5）$[x]_{补}=0\ 101B$，$[y]_{补}=1\ 101B$。将符号和数值部分分开处理。

补码不恢复余数除法过程描述如下：初始中间余数为 0000 0101，整个循环内执行的要点是“同、1、减；异、0、加”。共循环 3 次。最终得到一个 3 位无符号数表示的商 0001 和余数 0000，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 1 001，余数的原码表示为 0 000。因此，x/y 的商为–1，余数为 0。

余数寄存器 R	余数/商寄存器 Q	说　　明
0 0 0 0	0 1 0 1	开始 $R_0 = X$
+1 1 0 1		被除数和除数异号，做加法
1 1 0 1	0 1 0 1	同、1、减
1 0 1 0	1 0 1 1	$2R_1$（R 和 Q 同时左移，空出一位商）
+0 0 1 1		$R_2 = 2R_1 - Y$
1 1 0 1	1 0 1 1	同、1、减
1 0 1 1	0 1 1 1	$2R_2$（R 和 Q 同时左移，空出一位商）
+0 0 1 1		$R_3 = 2R_2 - Y$
1 1 1 0	0 1 1 1	同、1、减
1 1 0 0	1 1 1 1	$2R_3$（R 和 Q 同时左移，空出一位商）
+0 0 1 1		$R_4 = 2R_3 - Y$
1 1 1 1	1 1 1 1	同、1、减
1 1 1 1	1 1 1 1	$2R_4$（R 和 Q 同时左移，空出一位商）
0 0 1 1		$R_5 = 2R_3 - Y$
0 0 1 0	1 1 1 0	异、0、加（最高位商 1 去掉）

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。