厦門大學
XIAMEN UNIVERSITY

计算机组成原理课程作业

——第三次

黄勖 22920212204392

3.1

厦門大學 XIAMEN UNIVERSITY

解释下列名词。
全加器 半加器 进位生成函数 进位传递函数 算术移位 逻辑移位 阵列乘法器 原码恢复余数除法 原码不恢复余数法 阵列除法 串行进位 先行进位 对阶 规格化 保留附加位

1. 全加器（Full Adder）：一种用于加法运算的数字电路，能够将两个二进制数相加并输出其和以及进位。全加器有三个输入，即两个要相加的数字和上一个加法器的进位，以及两个输出，即它们的和以及新的进位。
2. 半加器（Half Adder）：一种用于加法运算的数字电路，只能将两个二进制数的低位相加，不能处理进位。半加器有两个输入，即两个要相加的数字和一个输出，即它们的和。
3. 进位生成函数：一种用于加法运算的逻辑函数，其输出表示在特定情况下是否会产生进位。进位生成函数有两个输入，即两个要相加的数字。
4. 进位传递函数：一种用于加法运算的逻辑函数，其输出表示在特定情况下是否会传递进位。进位传递函数有三个输入，即两个要相加的数字和上一个加法器的进位。
5. 算术移位：一种移位操作，用于将一个二进制数的位向左或向右移动，移动后的空位用0或符号位填充。
6. 逻辑移位：一种移位操作，用于将一个二进制数的位向左或向右移动，不考虑符号位，移动后的空位用0填充。
7. 阵列乘法器：一种数字电路，用于执行乘法运算。它使用一系列乘法器和加法器，将两个数字的每一位相乘，然后将结果相加。

3.1

厦門大學 XIAMEN UNIVERSITY

解释下列名词。

全加器 半加器 进位生成函数 进位传递函数 算术移位 逻辑移位 阵列乘法器 原码恢复余数除法 原码不恢复余数法 阵列除法 串行进位 先行进位 对阶 规格化 保留附加位

8. 原码恢复余数除法：一种用于除法运算的算法，可将两个有符号的二进制数相除，并输出商和余数。该算法使用除数的绝对值来进行运算。
9. 原码不恢复余数法：一种用于除法运算的算法，可将两个有符号的二进制数相除，并输出商，但不输出余数。该算法使用除数的绝对值来进行运算。
10. 阵列除法：一种数字电路，用于执行除法运算。它使用一系列乘法器和加法器，通过逐步减去除数的多个倍数来计算商和余数。
11. 串行进位：一种加法器的实现方式，其中进位信号需要依次从高位向低位传递，每个加法器都需要等待上一个加法器的进位信号才能开始计算。
12. 先行进位：一种加法器的实现方式，其中进位信号可以从低位向高位传递，允许多个加法器同时进行计算，从而提高运算速度。
13. 对阶：在进行浮点数运算时，需要将两个浮点数的小数点对齐，使它们可以进行运算。这个过程称为对阶。
14. 规格化：在浮点数中，规格化是将一个数表示成科学计数法的形式，即将小数点移动到最左边的非零数字的右侧，使其系数部分只有一位小数。
15. 保留附加位：在进行浮点数运算时，为避免精度损失，通常会保留一些额外的位数来进行计算。这些位数称为保留附加位。

(1) [2009]一个C语言程序在一台32位机器上运行，程序中定义了3个变量x、y、z,其中x和z是int型，y为short型。当x=127, y=-9时,执行赋值语句z=x+y后, x、y、z的值分别是（ D ）

A. x= 0000007FH, y= FFF9H, z= 00000076H

B. x= 0000007FH, y= FFF9H, z= FFFF0076H

C. x= 0000007FH, y= FFF7H, z= FFFF0076H

D. x= 0000007FH, y= FFF7H, z= 00000076H

在32位机器上，int型变量占用4个字节，short型变量占用2个字节。因此，变量x和z各占用4个字节，变量y占用2个字节。

在本题中，变量x的值为127，变量y的值为-9。由于变量y被转换成int型，因此参与运算的值为-9的32位补码，即0xFFFFFFF7。将x和y相加的结果为118，即127+(-9)。因此，执行完赋值语句z=x+y后，变量z的值为00000076H，变量x和y的值分别为0000007FH和FFF7H。

3.2

插入一段文本插入

(2) [2010]假定有4个整数用8位补码分别表示r_1 = FEH, r_2= F2H，r_3 = 90H，r_4= F8H,若将运算结果存放在-一个8位的寄存器中，则下列运算会发生溢出的是（B）

A. r1 x r2

B. r2 x r3

C. r1 x r4

D. r2 x r4

3.2

8位补码的表示范围是-128~127

然后进行乘法运算时，如果结果超出了8位补码的表示范围，就会发生溢出。

对于给定的4个整数，它们的补码表示如下：

r1 = FEH = -2（补码）

r2 = F2H = -14（补码）

r3 = 90H = -112（补码）

r4 = F8H = -8（补码）

A. r1 x r2 = (-2) x (-14) = 28，未发生溢出。

B. r2 x r3 = (-14) x （-112）= 1568，发生溢出，因为1568不在8位补码的表示范围内。

C. r1 x r4 = (-2) x (-8) = 16，未发生溢出。

D. r2 x r4 = (-14) x (-8) = 112，未发生溢出。

因此，只有选项B会发生溢出，答案为B。

(3) [2013]某字长为8位的计算机中，已知整型变量x、y的机器数分别为$[x]_{补}$=11110100 $[y]_{补}$=10110000。若整型变量z=2x + y/2，则z的机器数为（ A ）

A.11000000

B.00100100

C.10101010

D.溢出

首先，将给定的机器数转换为十进制数：

[x]补 = 11110100（补码）= -12（十进制）

[y]补 = 10110000（补码）= -80（十进制）

Z=2x+y/2=-24-40=-64=（1100 0000）B

3.2

(4) [2018]假定带符号整数采用补码表示，若int型变量x和y的机器数分别是FFFF FFDFH和0000 0041H，则x、y的值以及x-y的机器数分别是（C）

A. x=-65, y=41, x- y的机器数溢出

B. x=-33，y=65, x- y的机器数为FFFF FF9DH

C. x=-33, y=65, x- y的机器数为FFFF FF9EH

D. x=-65，y=41, x-y的机器数为FFFF FF96H

（FFFF FFDF）H=-33

（0000 0041）H=65

x-y=-33-65=-98=(FFFF FF9E)H

3.2

3.2

(5) [2018]整数x的机器数为1101 1000. 分别对x进行逻辑右移1位和算术右移1位操作，得到的机器数各是（ B ）

A. 1110 1100、1110 1100

B. 0110 1100、1110 1100

C. 1110 1100、0110 1100

D. 0110 1100、0110 1100

逻辑移位不考虑符号位，移动后的空位用0填充。

算数移位

	码　制	添 补 代 码
正数	原码、补码、反码	0
负数	原码	0
	补码	左移添 0
		右移添 1
	反码	1

3.2

(6) [2019]浮点数加减运算过程一般包括对阶、尾数运算、规格化、舍入和判断溢出等步骤。设浮点数的阶码和尾数均采用补码表示，且位数分别为5位和7位(均含2位符号位)。若有两个数X=2^7 x 29/32，Y=2^5 x 5/8，则用浮点加法计算X + Y的最终结果是（ D ）

A.0011 1110 0010 B. 0011 1010 0010 C.0100 0001 0001 D.发生溢出

X的浮点数格式为00,111;00,11101(分号前为阶码,分号后为尾数)

Y的浮点数格式为00,101;00,10100

第一步:对阶。X、Y阶码相减,即00,111-00,101=00,010,可知X的阶码比Y的价码大2。根据小阶向大阶看齐的原则,将Y的阶码加2,尾数右移2位,Y变为00,111:00,00101。

第二步:尾数相加。即00,11101+00,00101=01,00010,尾数相加结果符号位为01,故需右规。

第三步:规格化。将尾数右移1位,阶码加1,得X+Y为01,000;00,10001。

第四步:判溢出。阶码符号位为01,说明发生溢出。

(7) [2015]下列有关浮点数加减运算的叙述中，正确的是（D）

I.对阶操作不会引起阶码上溢或下溢

II.右规和尾数舍入都可能引起阶码上溢

III.左规时可能引起阶码下溢

IV.尾数溢出时结果不一定溢出

A.仅II. III　　B.仅1、II、IVC.仅1、III、IV　　D. I. II. III、IV

对阶是较小的阶码对齐至较大的阶码，I 正确。

右规和尾数舍入过程，阶码加1而可能上溢，II 正确，同理III也正确。

尾数溢出时可能仅产生误差，结果不一定溢出，IV 正确。

3.2

3.4已知x和y，用变形补码计算x+y,并判断结果是否溢出。
(3)x=-0.10111. y= -0.11000。
符号位最终为10，负溢出

3.4

$[X]_{补} = 11.01001$

$[Y]_{补} = 11.01000$

$$\begin{array}{r} 11.01001 \\ +\ 11.01000 \\ \hline 110.10001 \end{array}$$

3.4

3.5已知x和y， 用变形补码计算x-y,并判断结果是否溢出。
(3)x= -0.11111 y=-0.11001。
未溢出

3.5

$[X]_{补}=11.0000$

$[Y]_{补}=11.00111$　　$[-Y]_{补}=00.11001$

$$\begin{array}{r} 11.0000 \\ +\ 00.11001 \\ \hline 11.11001 \end{array}$$

3.6用原码一位乘法计算x·y。
(2) x=-0.11010, y= -0.01011。

Tips:
- 乘数的符号位不参与运算，可以省略
- 原码一位乘可以只用单符号位
- 答题时最终结果最好写为原码机器数

原码一位乘法：机器字长n+1，数值部分占n位
符号位通过异或确定；数值部分通过被乘数和乘数绝对值的n轮加法、移位完成
根据当前乘数中参与运算的位确定(ACC)加什么。若当前运算位=1，则(ACC)+$[|x|]_{原}$；
若=0，则(ACC)+0。
每轮加法后ACC、MQ的内容统一逻辑右移

3.6

$|x| = 0.11010$ $|y| = 0.01011$

	(高位部分积)	(低位部分积/乘数)		说明
	0.00000	0101⊥	丢失位	起始情况
+|x|	0.11010			$C_S=1$ 则 +|x|
	0.11010			
右移	0.01101	0010⊥	1	右移部分积与乘数
+|x|	0.11010			$C_S=1$ 则 +|x|
	1.00111			
右移	0.10011	1001⊥	11	右移部分积与乘数
+0	0.00000			$C_S=0$ 则 +0
	0.10011			
右移	0.01001	1100⊥	011	右移部分积与乘数
+|x|	0.11010			$C_S=1$ 则 +|x|
	1.00011			
右移	0.10001	11100	1011	右移部分积与乘数
+0	0.00000			$C_S=0$ 则 +0
	0.10001			
右移	0.01000	11110	01011	右移部分积和乘数，乘数全部移出。

结果的绝对值

符号位 $P_S = x_S \oplus y_S = 1 \oplus 1 = 0$ 得 $x \times y = 0.01000111110$

3.6

3.7用补码一位乘法计算xy。

(2) x= -0.011010, y= -0.011101。

3.7 $[x]_{补}=1.100110$ $[-x]_{补}=0.011010$ $[y]_{补}=1.100011$

Booth算法：

n轮加法、算数右移，加法规则如下：

辅助位 - MQ中最低位 = 1时，(ACC)+$[x]_{补}$

辅助位 - MQ中最低位 = 0时，(ACC)+0

辅助位 - MQ中最低位 = -1时，(ACC)+$[-x]_{补}$

补码的算数右移：

符号位不动，数值位右移，正数右移补0，负数右移补1（符号位是啥就补啥）

(高位部分积)	(低位部分积/乘数)	说明
0.000000	1.100011 0	初始情况
+$[-x]_{补}$ 0.011010	↑辅助位	$y_4y_5=10$，$y_5-y_6=-1$，+$[-x]_{补}$
0.011010		
右移 0.001101	01.100011	右移部分积与乘数
+0 0.000000		$y_5-y_6=0$，+0
0.001101		
右移 0.000110	101.10001	右移部分积与乘数
+$[x]_{补}$ 1.100110		$y_4y_5=01$，$y_5-y_6=1$，+$[x]_{补}$
1.101100		
右移 1.110110	0101.1000	右移部分积与乘数
+0 0.000000		$y_5-y_6=0$，+0
1.110110		

3.7

3.7用补码一位乘法计算xy。

(2) x= -0.011010, y= -0.011101。

n轮加法、算数右移，加法规则如下：

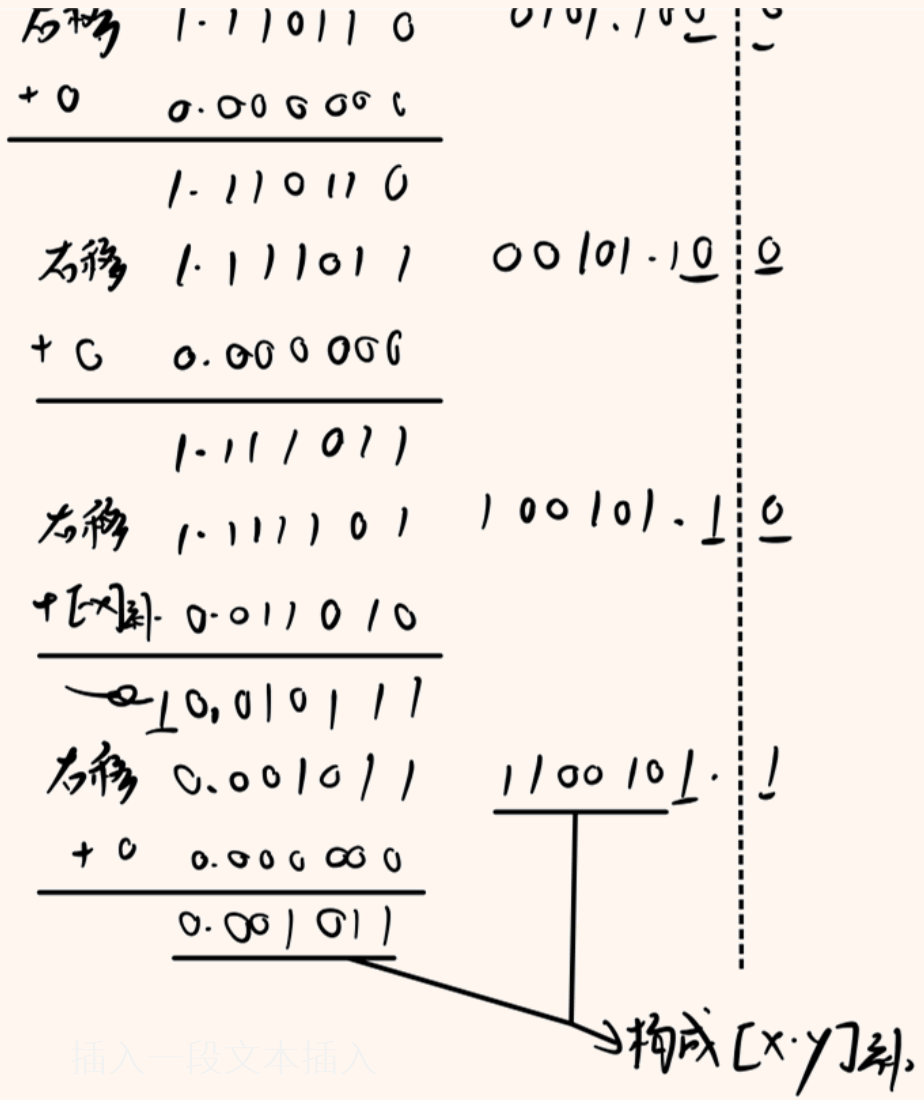
辅助位 - MQ中最低位 = 1时，$(ACC)+[x]_{补}$

辅助位 - MQ中最低位 = 0时，$(ACC)+0$

辅助位 - MQ中最低位 = -1时，$(ACC)+[-x]_{补}$

补码的算数右移：

符号位不动，数值位右移，正数右移补0，负数右移补1（符号位是啥就补啥）



右移部分积与乘数

$Y_5 - Y_4 = 0$，+0

右移部分积与乘数

$Y_5 - Y_4 = 0$，+0

右移部分积与乘数

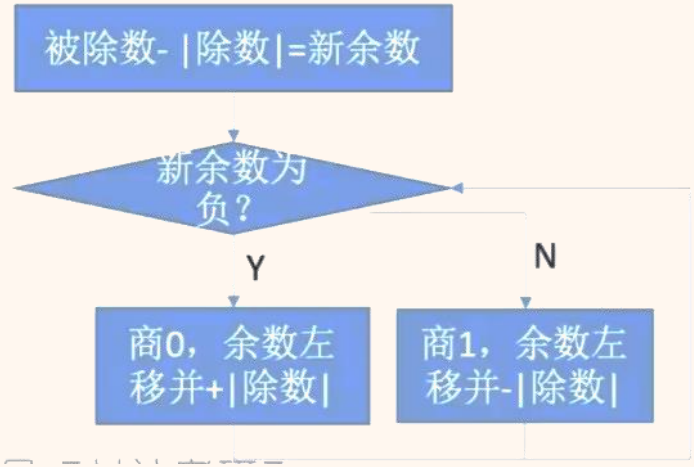
$Y_4 Y_3 = 10$，$Y_5 - Y_4 = -1$，$+[-x]_{补}$

右移部分积与乘数

$Y_5 - Y_4 = 0$，+0

3.7

3.8用原码不恢复余数法计算x÷y。
(2) x= -0.10101. y= 0.11000



加/减n+1次，每次加减确定一位商；
左移n次（最后一次加减完不移位）
最终可能还要再多一次加

3.8

$|x| = 0.10101$ $|y| = 0.11000$ $[|y|_{补}] = 0.11000$ $[-|y|_{补}] = 1.01000$

被除数/余数	商	说明
0.10101		
$+[-|y|_{补}]$ 1.01000		
1.11101	0	新余数负，余数左移 +|除数|
左移 1.11010		
$+[|y|_{补}]$ 0.11000		
0.10010	01	新余数正，余数左移 -|除数|
左移 1.00100		
$+[-|y|_{补}]$ 1.01000		
0.01100	011	新余数正，余数左移 -|除数|
左移 0.11000		
$+[-|y|_{补}]$ 1.01000		
0.00000	0111	新余数零，除尽了
左移 0.00000		
+0 0.00001		
0.00000	01110	$Q_S = x_S \oplus y_S = 1$

得 $x/y = -0.1110$，余数为 $0 \times 2^{-5} = 0$

3.9设数的阶码为3 位，尾数为6位(均不包括符号位) ,按机器补码浮点运算规则完成下列$[x+y]_{补}$运算。

(2)$x=2^{-101}x(-0.100010)$， $y=2^{-100}\ x(-0.010110)$

3.9

① 补码形式表示：

$[X]_{补}$ = 11 011 , 11.0 1110

$[Y]_{补}$ = 11 100, 11.1010 0

② 对阶，x(阶码-101)向y(阶码-100)对齐，x尾数为移1位

$[X]_{补}$ = 11 100 , 11.1011 1 0

③ 尾数运算

$$\begin{array}{r} 11100\ ,\ 11.1011\ 10 \\ +\ \ 11100\ ,\ 11.1010\ 0 \\ \hline 11100\ ,\ 11.0110\ 01\ 0 \end{array}$$

④ 无舍入处理 ⑤ 溢出判断：无溢出

运算结果：$[x+y]_{补}$ = 11 100 , 11.0110 0

3.9

厦門大學 XIAMEN UNIVERSITY

3.10采用IEEE754单精度浮点数格式计算下列表达式的值。

(2) 0.625- (-12.25)

IEEE754单精度浮点数：32位　　32浮点数 *float*

阶码的真值=E-127；尾数=1.M

0.625 = 0 01111110（126） 01000000000000000000000=$1.01*2^{-1}$

-12.25 = 1 10000010（130） 10001000000000000000000=$-1.10001*2^{3}$

（1）对阶：X、Y 的阶码不同，小对大，0.625变为$0.000101*2^{3}$

（2）尾数相减：0.000101+1.10001=1.100111

（3）结果规格化：尾数相减后为规格化数

（4）舍入处理：没有舍入处理

（5）溢出判断：阶码正常，无溢出

得到结果：0.625-（-12.25）＝（0 10000010 10011100000000000000000）B

＝414E0000H

3.10

3.11假定在一个8位字长的计算机中运行如下C语言程序段。

```
unsigned int x=134; .
unsigned int y=246;
int m=x; int n=y; .
unsigned int z1=x-y;
unsigned int z2=x+y;
int k1=m-n;
int k2=m+n;
```

若编译器编译时将8个8位寄存器R1 ~ R8分别分配给变量x、y、m、n、z1、z2、k1和k2。请回答下列问题(提示:带符号整数用补码表示)。

(1)执行上述程序段后，寄存器R1、R5和R6中的内容分别是什么?(用十六进制表示)

(2)执行上述程序段后，变量m和k1的值分别是多少?(用十进制表示)

（1）(R1)=134=1000 0110B=86H，(R5)=-112=10010000B=90H，(R6)=380（溢出）=（1）0111 1100B=7CH

（2）m=（1000 0110）补 = -122，n = （1111 0110）补 = -10，k1=-122+10=-112。

3.11

插入一段文本插入

unsigned int x=134; .unsigned int y=246;
int m=x; int n=y; .unsigned int z1=x-y;unsigned int z2=x+y;
int k1=m-n;int k2=m+n;

若编译器编译时将8个8位寄存器R1 ~ R8分别分配给变量x、y、m、n、z1、z2、k1和k2。请回答下列问题(提示:带符号整数用补码表示)。

(3)上述程序段涉及带符号整数加减、无符号整数加减运算，这4种运算能否利用同一个加法器及辅助电路实现?简述理由。

(4)计算机内部如何判断带符号整数加减运算的结果是否发生溢出?上述程序段中，哪些带符号整数运算语句的执行结果会发生溢出?

（3）能。在理论上，可以使用同一个加法器及辅助电路实现带符号整数加减和无符号整数加减。实现的关键在于如何处理带符号整数的符号位。

一种常见的实现方式是使用补码来表示带符号整数，并在加法器中进行补码加法运算。对于无符号整数，其符号位为0，直接进行加法运算即可。对于带符号整数，需要特殊处理符号位，具体做法是将符号位视为最高位参与加法运算，处理后再舍弃最高位得到正确结果。

因此，可以使用同一个加法器及辅助电路实现这四种运算。

（4）计算机判断带符号整数加减运算是否发生溢出的方法是检查运算结果的符号位进位和操作数的最高进位是否相同，若相同则无溢出，否则则发生了溢出。

int k2=m+n;（1000 0110 + 1111 0110）在执行加法操作时，计算机会对这两个带符号的二进制数进行加法运算。由于这两个二进制数的最高位都是1，它们在加法运算时会产生一个进位。这个进位会使得结果超出了这个变量类型所能表示的最大值范围，导致溢出。

3.11

ABOUT

感谢观看

THANKS FOR WATCHING

---汇报人：黄勖