第 3 章 运算方法和运算部件

3.1 教学目标和内容安排

主要教学目标：使学生掌握核心运算部件 ALU 以及计算机内部各种基本运算算法和运算部件，能够运用所学知识分析和解释高级语言和机器级语言程序设计中遇到的各种问题和相应的执行结果。

基本学习要求：

(1) 了解高级程序设计语言和低级程序设计语言中涉及的各种运算。

(2) 掌握定点数的逻辑移位、算术移位和扩展操作方法。

(3) 了解串行加法器和快速并行加法器的基本工作原理。

(4) 掌握补码加减运算方法，并能设计补码加减运算器。

(5) 了解原码加减运算的基本原理。

(6) 了解定点数乘法和除法运算的基本思想。

(7) 了解专用的阵列乘法器的基本思想。

(8) 理解为何在运算中会发生溢出，并掌握各类定点数运算的溢出判断方法。

(9) 掌握浮点数加减运算过程和方法。

(10) 理解 IEEE 754 标准对附加位的添加以及舍入模式等方面的规定。

(11) 了解浮点数乘法和除法运算的基本思想。

(12) 掌握算术逻辑单元 ALU 的功能和结构。

(13) 了解浮点数加减运算器的基本结构。

本章涉及各种类型数据的各种运算算法和运算电路，因此内容多而烦琐。特别是原码加减运算和各种类型的乘除运算，它们的基本原理都很简单，但实现起来非常复杂。在课堂教学中，这些内容往往占用很多课时，而且烦琐的步骤和一些算法规定也经常使得学生感觉枯燥无味。这些内容在本课程内容框架体系中不属于主干内容，对这些内容掌握的好坏与深浅程度基本不会影响学生对其他知识的学习。因此，对于原码加减运算，只要根据现实世界中十进制加减运算规则去理解，把原理讲清楚就行了，没有必要让学生死记硬背运算规则；对于乘法运算，只要将原码一位乘法和补码一位乘法(布斯乘法)的基本原理、两位乘算法的基本思想以及阵列乘法器的基本思想讲清楚就行了。对于除法运算，只要将原码除法运算的基本原理、补码除法运算的特点以及阵列除法器的基本思想讲清楚就行了。对于乘

除运算的学习，其主要目的不是学会怎样模拟计算机进行乘除运算，而是能够认识到乘除运算的算法复杂性和相对较大的时间开销，并且认识到不同实现方法所用时间开销是不同的，这种不同主要是由于运算部件控制方式的不同而造成的。

课时有限的情况下，有关原码加减运算、两位乘法运算、补码除法运算、阵列乘法器、浮点数乘除运算、浮点运算部件等加 * 的部分，都可以只用一两句话概括讲一下基本思想或提出问题以引起学生进一步思考并课后阅读，无须占用大量的课堂时间来介绍细节内容。

为了增强学生对计算机内部各类运算算法的认识，可以让学生亲自编写相关的程序，通过程序的执行结果来理解本章所学的知识。与本章内容相关的编程练习示例：①给定一组无符号数和带符号整数变量的值，对其进行移位操作后，查看其结果，并进行分析解释；②对于某个负数，如－4098，将该数赋值给一个 short 类型变量，然后将其转换为 unsigned short、int、unsigned int、float 等类型，查看其结果，并进行分析解释；③对于某个大的正整数，如 2147483647（即 2^{31}），将该数赋值给某个 int 类型变量，再将其转换为是 short、unsigned short、unsigned int、float 等类型，查看其结果，并进行分析解释；④对于某个有效位数多于 8 的浮点数，如 123456.789e5，将其定义为 float 型变量，然后转换为 double 型变量；再反过来将 double 型转换为 float 型，查看其结果，并进行分析解释；⑤对于各类运算，给出一些特殊的例子，以进行“溢出”“大数吃小数”等方面的验证，并分析结果以给出合理的解释。例如，对于无符号数（如 unsigned int），要求计算 1＋4294967295 和 1－4294967295；对于带符号整数（如 int），要求计算 2147483647＋1 和－2147483648－1；对于浮点数，要求分别计算 （1.0＋123456.789e30）＋（－123456.789e30） 和 1.0＋（123456.789e30＋（－123456.789e30）），查看两个结果是否一致，等等。

3.2 主要内容提要

1. 加法器

根据进位方式的不同，有三种基本加法器：行波进位加法器、进位选择加法器和先行（超前）进位加法器。行波进位加法器通过将多个一位全加器串行连接，各进位串行传递，速度慢；进位选择加法器通过选择两个分别带进位 0 和 1 的高位部分加法器的输出来实现高、低两部分的并行执行，使运算时间减半；先行（超前）进位加法器通过“进位生成”和“进位传递”函数来使各进位独立、并行产生，速度快。可用单级、两级或更多级先行进位方式连接。采用先行进位方式能加快加法器速度，目前多用这种方式。

2. 算术逻辑单元（ALU）

在先行进位加法器的基础上增加其他逻辑构成 ALU，以实现基本的加/减算术运算和基本逻辑运算。ALU 有两个操作数输入、一位进位输入、一个操作控制输入、一个结果输出、一位进位输出，以及零标志（ZF）和溢出标志（OF）等输出。

3. 定点运算及定点运算器

定点运算由专门的定点运算器实现，其核心部件是带先行进位加法器的 ALU，在控制逻辑的控制下，可进行各种逻辑运算和算术运算。除基本逻辑运算外，主要的运算包括以下 5 种。

（1）移位运算：包括逻辑移位、算术移位和循环移位。逻辑移位对无符号数进行，移位

时,在空出的位补0,左移时可根据移出位是否为1来判断溢出;算术移位对带符号整数进行,移位前后符号位保持不变,否则溢出;循环移位时不需要考虑溢出。左移一位,数值扩大一倍,相当于乘2操作;右移一位,数值缩小一半,相当于除2操作。

(2) 扩展运算:包括零扩展和符号扩展。零扩展对无符号数进行,高位补0;符号扩展对带符号整数进行,因为用补码表示,所以在高位直接补符号。

(3) 整数加减运算:包括补码加减、原码加减和移码加减运算。补码加减运算用于带符号整数,符号位和数值位一起运算。同号相加时,若结果的符号不同于加数的符号,则会发生溢出;因为IEEE 754标准用原码小数表示尾数,所以原码加减运算用于浮点数的尾数,符号位和数值位分开运算,同号数相加或异号数相减时做加法,同号数相减或异号数相加时做减法;因为IEEE 754标准用移码表示指数,所以移码加减运算用于浮点数的指数,移码的和、差等于和、差的补码,因此,可通过移码先求出和、差的补码,最后将符号取反,就能得到和、差的移码表示。减法运算电路只要在加法器基础上增加求补和溢出判断电路,并将进位输入端用于加减控制就可实现,因此,所有的减法运算都是用加法器实现的。

(4) 整数乘法运算:包括无符号数乘法、补码乘法和原码乘法,都可用加减及右移运算实现。无符号数乘法用于无符号整数;补码乘法用于带符号整数,符号位和数值位一起运算,可采用Booth算法或MBA算法;原码乘法用于浮点数尾数,符号位和数值位分开运算,数值部分用无符号数乘法实现。除了可以在定点运算部件中用加法和右移来实现乘法运算以外,也可用基于CSA的阵列乘法器、流水线乘法器、MBA+WT乘法器等实现。两个n位数相乘得到$2n$位数乘积,若结果只取低n位,则乘积高n位必须是0(无符号数乘法)或是符号(补码乘法),否则溢出。若采用一位乘法算法,则n位数相乘大约需要n次加减运算和n次右移运算。

(5) 整数除法运算:包括无符号数除法、补码除法和原码除法,都可用加减及左移运算实现。无符号数除法用于无符号整数,有恢复余数法和不恢复余数法两种;补码除法用于带符号整数,符号位和数值位一起运算,也有恢复余数法和不恢复余数法两种;原码除法用于浮点数尾数,符号和数值分开运算,数值部分用无符号数除法实现。因为除法运算无法事先确定做加法还是减法,所以无法实现流水线除法器。两个n位数相除时,需要将被除数扩展成$2n$位数,对于不恢复余数法,n位数除法大约需要n次加减运算和n次左移运算。

4. 浮点运算及浮点运算器

计算机中大多用IEEE 754标准表示浮点数,因此,浮点运算主要针对IEEE 754标准浮点数。浮点运算由专门的浮点运算器实现,因为一个浮点数由一个定点小数和一个定点整数组成,所以浮点运算器由定点运算部件构成,其核心部件还是带先行进位加法器的ALU。浮点运算包括浮点加减运算和浮点乘除运算。

(1) 浮点加减运算:按照对阶、尾数加减、规格化、舍入和溢出判断几个步骤完成。对阶时小阶向大阶看齐,阶小的那个数的尾数右移,直到两数阶码相同,右移时一般保留两位或三位附加位;尾数加减时用原码加减运算实现;规格化处理时根据结果的尾数形式的不同确定进行左规或右规操作;舍入操作有就近舍入、正向舍入、负向舍入和截去4种方式,默认的是就近舍入到偶数方式;溢出判断主要根据结果的阶码进行判断,当发生阶码上溢时,运算结果发生溢出,当发生阶码下溢时,运算结果近似为0。

(2) 浮点乘除运算:尾数用原码小数的乘/除运算实现,阶码用移码加减运算实现,需

要对结果进行规格化、舍入和溢出判断。

3.3 基本术语解释

逻辑移位(logical shift)

逻辑移位是对无符号数进行的移位,把无符号数看成一个逻辑数进行移位操作。左移时,高位移出,低位补0;右移时,低位移出,高位补0。

算术移位(arithmetic shift)

算术移位是对带符号整数进行的,移位前后符号位不变。移位时,符号位不动,只是数值部分进行移位。左移时,高位移出,末位补0,移出非符时,发生溢出。右移时高位补符,低位移出。移出时进行舍入操作。

循环(逻辑)移位(rotating shift)

循环移位是一种逻辑移位,移位时把高(低)位移出的一位送到低(高)位。即左移时,各位左移一位,并把最左边的位移到最右边;右移时,各位右移一位,并把最右边的位移到最左边。

扩展操作(extending)

在计算机内部,有时需要将一个取来的短数扩展为一个长数,此时要进行填充(扩展)处理。有"零扩展"和"符号扩展"两种。

零扩展(zero extending)

对无符号整数进行高位补0的操作称为"零扩展"。

符号扩展(sign extending)

对补码表示的带符号整数在高位直接补符的操作称为"符号扩展"。

扩展器(extender)

扩展器是进行扩展(填充)操作的部件,一般输入为 n 位,输出为 $2n$ 位。

半加器(half adder)

半加器是只考虑本位两个加数而不考虑低位进位来生成本位和的一位加法器。

全加器(full adder,(3,2) adder)

全加器是不仅考虑本位两个加数而且考虑低位进位来生成本位和的一位加法器。

加法器(adder)

加法器是能进行 n 位加法运算的部件。

行波进位(ripple carry)

行波进位是在进行 n 位加法运算时,低位向高位的进位采用像"行波"一样串行传递的方式。

行波进位加法器(ripple carry adder)

行波进位加法器也称为串行进位加法器,它通过 n 个全加器按照串行方式连起来实现。进位方式采用行波进位方式。

先行进位(Carry LookAhead,CLA)

先行进位通过引入进位生成和进位传递两个进位辅助函数,使得加法器的各个进位之间相互独立、并行产生。这种进位方式也称为并行进位方式。

成组先行进位(Block Carry LookAhead,BCLA)

成组先行进位将数据分成若干组,在每一组内,除了并行生成组内各位向前面的进位外,同时还通过组进位生成和组进位传递两个组进位辅助函数,使得各组的进位也能相互独立、并行产生。

先行进位加法器(CLA adder)

采用先行进位方式实现的加法器,也称为并行进位加法器。因为采用先行进位方式能够快速得到和数,故也称为快速加法器。

算术逻辑部件(Arithmetic Logic Unit,ALU)

算术逻辑部件是用于执行各种基本算术运算和逻辑运算的部件,其核心部件是加法器,有两个操作数输入端和低位进位输入端,一个运算结果输出端和若干标志信息(如零标志、溢出标志等)输出端。因为ALU能进行多种运算,因此,需要通过相应的操作控制输入端来选择进行何种运算。

零标志ZF,溢出标志OF,进位/借位标志CF,符号标志SF

ALU部件的输出除了运算结果外,还有一组状态标志信息。例如,ZF(Zero Flag)为1时表示结果为0;OF(Overflow Flag)为1时表示结果溢出;CF(Carry Flag)为1表示在最高位产生了进位或借位;SF(Sign Flag)和符号位保持一致,若为1则表示结果为负数。

布斯算法(Booth's algorithm)

布斯算法是一种一位补码乘法算法,用于带符号数的乘法运算,由布斯(Booth)提出。算法的基本思想是:在乘数的末位添加一个0,乘数中出现的连续0和连续1处不进行任何运算;出现10时,做减法;出现01时,做加法。每次只做一位乘法,因而每一步都右移一位。

改进布斯算法(Modified Booth's algorithm,MBA)

改进布斯算法也称为基-4 Booth算法,从布斯算法推导得到,采用两位一乘,根据乘数中连续三位的不同取值确定每一步相应的运算,每次部分积右移两位。

对阶(align exponent)

浮点数加/减运算时,在尾数相加/减之前所进行的操作称为对阶。对阶时,需要比较两个阶的大小。阶小的那个数的尾数右移,阶码增量。右移一次,阶码加1,直到两数的阶码相等为止。

溢出(overflow)

溢出是指一个数比给定的格式所能表示的最大值还要大或比最小值还要小的现象。因为无符号数、带符号整数和浮点数的位数是有限的,所以,都有可能发生溢出,但判断溢出的具体方法不同。

阶码下溢(underflow)

在浮点数运算中,当运算的结果其指数(阶)比最小允许值还小,此时,运算结果发生阶码下溢,即运算结果的实际值位于0和绝对值最小的可表示数之间。通常机器会把阶码下溢时浮点数的值置为0。因此,这种情况下结果并没有发生错误,只是得到了一个近似于0的值,因而无须进行溢出处理。

阶码上溢(overflow)

在浮点数运算中,当运算的结果其指数(阶)超过了最大允许值,此时,浮点数发生了上溢。即向∞方向溢出。如果结果是正数,则发生正上溢,有的机器把值置为+∞;如果是负

数,则发生负上溢,有的机器把值置为$-\infty$。这种情况为软件故障,通常要引入溢出故障处理程序来处理。

规格化数(normalized number)

为了使浮点数中能尽量多地表示有效位数,一般要求运算结果用规格化数形式表示。规格化浮点数的尾数小数点后的第一位一定是个非零数。因此,对于原码编码的尾数来说,只要看尾数的第一位是否为1就行;对于补码表示的尾数,只要看符号位和尾数最高位是否相反。

左规(left normalize)

在浮点数运算中,当一个尾数的数值部分的高位出现0时,尾数为非规格化形式。此时,进行"左规"操作:尾数左移一位,阶码减1,直到尾数为规格化形式为止。

右规(right normalize)

在浮点数运算中,当尾数最高有效位有进位时,发生尾数溢出。此时,进行"右规"操作:尾数右移一位,阶码加1,直到尾数为规格化形式为止。右规过程中,要判断是否发生溢出。此时,只要阶码不发生上溢,那么浮点数就不会溢出。

舍入(rounding)

舍入是指数值数据右部的低位数据需要丢弃时,为保证丢弃后数值误差尽量小而考虑的一种操作。例如,定点整数"右移"时、浮点加/减运算中某数"对阶"时、浮点运算结果"右规"时都会涉及舍入。

保护位(guard bit)和舍入位(rounding bit)

为了使浮点数的有效数据位在右移时最大限度地保证不丢失,一般在运算过程中得到的中间值后面增加若干数据位,这些位用来保存右移后的有效数据,因此,是添加的附加位。增设附加位后,能保证运算结果具有一定的精度,但最终附加位可能被去掉,以得到规定格式的浮点数,此时要考虑舍入。在IEEE 754标准中规定,浮点运算的中间结果可以额外多保留两位附加位,这两位分别称为保护位和舍入位。

粘位(sticky bit)

IEEE 754中规定,为了更进一步提高计算精度,可以在舍入位右边再增加一位,称为"粘位",只要舍入位的右边还有任何非零数位,则粘位为1,否则为0。

运算器(operational unit)

运算器即运算部件,通常指用ALU以及为了完成ALU的运算而必须与之共同工作的各种寄存器、多路选择器和实现数据传送的总线等构成的部件。根据功能不同有定点运算器和浮点运算器,它们是数据通路中的核心部件。

通用寄存器组(General-Purpose Register Set,GPRS)

CPU中提供了若干通用寄存器,这些寄存器可以用来存放指令操作的对象,需要在指令中明确给出寄存器的编号,所有通用寄存器合起来构成一个通用寄存器组,也称为寄存器堆或寄存器文件(register file)。通常,通用寄存器组有两个读口和一个写口。

多路选择器(multiplexer)

多路选择器是在多个输入数据中根据控制信号选择其一作为输出的部件。

桶型移位器(barrel shifter)

桶型移位器是由大量多路选择器实现的快速移位器,可一次左移或右移多位,移动位数

由控制输入端给出。

Q 乘商寄存器(Q multiplier-quotient Register)

Q 乘商寄存器用于乘除运算。在乘法中该寄存器用来存放乘数,在除法中用来存放商,并可以和另外的移位器共同完成左移(用于除法)或右移(用于乘法)操作。

3.4 常见问题解答

1. 无符号加法器如何实现?

答:计算机中,最基本的加法器是无符号加法器。根据进位方式的不同,有三种不同的基本实现方式:①串行进位加法器(行波进位加法器):通过 n 个全加器按照串行方式连起来实现;②进位选择加法器:通过选择两个分别带进位 0 和 1 的高位部分加法器的输出来实现高、低两部分的并行执行;③并行进位加法器(先行进位加法器):通过引入进位生成函数和进位传递函数,使得进位之间相互独立,并行产生。也称为快速加法器。

2. 补码加法器如何实现?

答:两个 n 位补码进行加法运算的规则是:两个 n 位补码直接相加,并将结果中最高位的进位丢掉。即采用模运算方式。显然,可用一个 n 位无符号加法器来生成各位的和。最终的结果是否正确,取决于结果是否溢出,只要结果不溢出,则结果一定是正确的。因此,补码加法器只要在无符号加法器的基础上再增加"溢出判断电路"即可。

3. 在补码加法器中,如何实现补码减法运算?

答:补码减法的规则是:两个数差的补码可用第一个数的补码加上另一数负数的补码得到。由此可见,减法运算可在加法器中运行。只要在加法器的第二个输入端输入减数的负数的补码。求一个数的负数的补码电路称为"负数求补电路"。可以通过"各位取反、末位加 1"来实现"负数求补电路"。

4. 现代计算机中是否要考虑原码加/减运算?

答:因为现代计算机中浮点数采用 IEEE 754 标准,浮点数的尾数都用原码表示,所以在进行两个浮点数加减运算时,必须考虑原码的加减运算。

5. 加法器的运算速度取决于什么?

答:在门电路延迟一定的情况下,加法器的速度主要取决于进位方式,先行进位方式比串行进位方式的速度快。

6. 定点整数运算要考虑加保护位和舍入吗?

答:不需要。整数运算的结果还是整数,没有误差,无须考虑加保护位,也无须考虑舍入。但运算结果可能会"溢出"。

7. 如何判断带符号整数运算结果是否溢出?

答:带符号整数用补码表示,对于单符号补码(即 2-补码)和双符号补码(即 4-补码,变形补码),其溢出判断方式不同。变形补码运算的溢出判断规则是:"当结果的两个符号位不同时,发生溢出";单符号补码运算时,异号数相加不会溢出,而对于同号数相加,则有两种判断规则。规则 1 是:"若结果的符号与两个加数的符号不同,则发生溢出"。规则 2 是:"若最高位的进位和次高位的进位不同,则发生溢出"。

8. 在计算机中，乘法和除法运算如何实现？

答：乘法和除法运算是通过加、减运算和左、右移位运算来实现的。只要用加法器和移位寄存器在控制逻辑的控制下就可以实现乘除运算。也可用专门的乘法器和除法器实现。

9. 浮点数如何进行舍入？

答：舍入方法选择的原则是：①尽量使误差范围对称，使得平均误差为0，即有舍有入，以防误差积累；②方法要简单，以加快速度。

IEEE 754有4种舍入方式：①就近舍入：舍入为最近可表示的数，若结果值正好落在两个可表示数的中间，则一般选择舍入结果为偶数；②正向舍入：朝$+\infty$方向舍入，即取右边的那个数；③负向舍入：朝$-\infty$方向舍入，即取左边的那个数；④截去：朝0方向舍入，即取绝对值较小的那个数。

10. 在C语言程序中，为什么以下程序段最终的 f 值为0，而不是2.5？

```
float f=2.5+1e10;
f=f -1e10;
```

答：首先，float类型采用IEEE 754单精度浮点数格式表示，因此，最多有24位二进制有效位数。因为$1e10=10^{10}=10\times10^3\times10^6$，在数量级上大约相当于$2^3\times2^{10}\times2^{20}=2^{33}$，而2.5的数量级为$2^1$，因此，在计算2.5+1e10进行对阶时，两数阶码的差为32。也就是说，2.5的尾数要向右移32位，从而使得24位有效数字全部丢失，尾数变为全0，再与1e10的尾数相加时结果就是1e10的尾数，因此f=2.5+1e10的运算结果仍为1e10，这样，再执行f=f-1e10时结果就为0。这个例子就是典型的“大数吃小数”的例子。

3.5 单项选择题

1. 8位无符号整数1001 0101右移一位后的值为(　　)。

A. 0100 1010　　B. 0100 1011　　C. 1000 1010　　D. 1100 101

2. 8位补码定点整数1001 0101右移一位后的值为(　　)。

A. 0100 1010　　B. 0100 1011　　C. 1000 1010　　D. 1100 1010

3. 8位补码定点整数1001 0101左移一位后的值为(　　)。

A. 1010 1010　　B. 0010 1010　　C. 0010 1011　　D. 溢出

4. 8位补码定点整数1001 0101扩展8位后的值用十六进制表示为(　　)。

A. 0095H　　B. 9500H　　C. FF95H　　D. 95FFH

5. 原码定点小数1.1001 0101扩展8位后的值为(　　)。

A. 1.0000 0000 1001 0101　　B. 1.1001 0101 0000 0000

C. 1.1111 1111 1001 0101　　D. 1.1001 0101 1111 1111

6. 考虑以下C语言代码：

```
short si=-8196;
int i=si;
```

执行上述程序段后，i 的机器数表示为(　　)。

A. 0000 9FFCH　　B. 0000 DFFCH　　C. FFFF 9FFCH　　D. FFFF DFFCH

7. CPU中能进行算术和逻辑运算的最基本运算部件是(　　)。

A. 多路选择器　　B. 移位器　　C. 加法器　　D. ALU

8. ALU的核心部件是(　　)。

A. 多路选择器　　B. 移位器　　C. 加法器　　D. 寄存器

9. 假定T表示一级门延迟,一个异或门的延迟为3T,则8位全先行进位加法器的关键路径延迟为(　　)。

A. 6T　　B. 8T　　C. 16T　　D. 17T

10. 在补码加/减运算部件中,无论采用双符号位还是单符号位,必须有(　　)电路,它一般用异或门来实现。

A. 译码　　B. 编码　　C. 溢出判断　　D. 移位

11. 某计算机字长为8位,其CPU中有一个8位加法器。已知无符号数$x=69$,$y=38$,现要在该加法器中完成$x+y$的运算,则该加法器的两个输入端信息和输入的低位进位信息分别为(　　)。

A. 0100 0101、0010 0110、0　　B. 0100 0101、0010 0110、1

C. 0100 0101、1101 1010、0　　D. 0100 0101、1101 1010、1

12. 某计算机字长为8位,其CPU中有一个8位加法器。已知无符号数$x=69$,$y=38$,现要在该加法器中完成$x-y$的运算,则该加法器的两个输入端信息和输入的低位进位信息分别为(　　)。

A. 0100 0101、0010 0110、0　　B. 0100 0101、1101 1001、1

C. 0100 0101、1101 1010、0　　D. 0100 0101、1101 1010、1

13. 某计算机字长为8位,其CPU中有一个8位加法器。已知带符号整数$x=-69$,$y=-38$,现要在该加法器中完成$x+y$的运算,则该加法器的两个输入端信息和输入的低位进位信息分别为(　　)。

A. 1011 1011、1101 1010、0　　B. 1011 1011、1101 1010、1

C. 1011 1011、0010 0101、0　　D. 1011 1011、0010 0101、1

14. 某计算机字长为8位,其CPU中有一个8位加法器。已知带符号整数$x=-69$,$y=-38$,现要在该加法器中完成$x-y$的运算,则该加法器的两个输入端信息和输入的低位进位信息分别为(　　)。

A. 1011 1011、1101 1010、0　　B. 1011 1011、1101 1010、1

C. 1011 1011、0010 0101、1　　D. 1011 1011、0010 0110、1

15. 某8位计算机中,假定x和y是两个带符号整数变量,用补码表示,$x=63$,$y=-31$,则$x+y$的机器数及其相应的溢出标志OF分别是(　　)。

A. 1FH、0　　B. 20H、0　　C. 1FH、1　　D. 20H、1

16. 某8位计算机中,假定x和y是两个带符号整数变量,用补码表示,$x=63$,$y=-31$,则$x-y$的机器数及其相应的溢出标志OF分别是(　　)。

A. 5DH、0　　B. 5EH、0　　C. 5DH、1　　D. 5EH、1

17. 某8位计算机中,假定带符号整数变量x和y的机器数用补码表示,$[x]_{补}=F5H$,$[y]_{补}=7EH$,则$x+y$的值及其相应的溢出标志OF分别是(　　)。

A. 115、0　　B. 119、0　　C. 115、1　　D. 119、1

18. 某8位计算机中，假定带符号整数变量 x 和 y 的机器数用补码表示，$[x]_{补}=F5H$，$[y]_{补}=7EH$，则 $x-y$ 的值及其相应的溢出标志 OF 分别是(　　)。

A. 115、0　　B. 119、0　　C. 115、1　　D. 119、1

19. 某8位计算机中，假定 x 和 y 是两个带符号整数变量，用补码表示，$[x]_{补}=44H$，$[y]_{补}=DCH$，则 $x+2y$ 的机器数以及相应的溢出标志 OF 分别是(　　)。

A. 32H、0　　B. 32H、1　　C. FCH、0　　D. FCH、1

20. 某8位计算机中，假定 x 和 y 是两个带符号整数变量，用补码表示，$[x]_{补}=44H$，$[y]_{补}=DCH$，则 $x-2y$ 的机器数以及相应的溢出标志 OF 分别是(　　)。

A. 68H、0　　B. 68H、1　　C. 8CH、0　　D. 8CH、1

21. 某8位计算机中，假定 x 和 y 是两个带符号整数变量，用补码表示，$[x]_{补}=44H$，$[y]_{补}=DCH$，则 $x/2+2y$ 的机器数以及相应的溢出标志 OF 分别是(　　)。

A. CAH、0　　B. CAH、1　　C. DAH、0　　D. DAH、1

22. 假定有两个整数用8位补码分别表示为 $r_1=F5H$，$r_2=EEH$。若将运算结果存放在一个8位寄存器中，则下列运算中会发生溢出的是(　　)。

A. r_1+r_2　　B. r_1-r_2　　C. $r_1\times r_2$　　D. r_1/r_2

23. 以下关于原码一位乘法算法要点的描述中，错误的是(　　)。

A. 符号位和数值位分开运算，符号位可由一个异或门生成

B. 通过循环执行“加法”和“移位”操作得到乘积

C. ALU 中是否进行部分积与被乘数的加法运算由乘数最低位决定

D. 移位时，将进位位、部分积和乘积部分一起进行算术右移

24. 假定一次 ALU 运算用1个时钟周期，移位一次用1个时钟周期，则最快的32位原码一位乘法所需的时钟周期数大约为(　　)。

A. 32　　B. 64　　C. 96　　D. 100

25. 以下关于 Booth 补码一位乘法算法要点的描述中，错误的是(　　)。

A. 符号位和数值位一起参加运算，无须专门的符号生成部件

B. 通过循环执行“加/减”和“移位”操作得到乘积

C. 由乘数最低两位决定对部分积和被乘数进行何种运算

D. 移位时，将进位位、部分积和乘积部分一起进行算术右移

26. 以下关于乘法运算部件的叙述中，错误的是(　　)。

A. 补码乘法部件可用于带符号整数的乘法运算

B. 原码乘法部件可用于浮点数中尾数相乘运算

C. 快速阵列乘法器中的基本部件包含有移位器

D. 两位乘法运算比一位乘法运算速度约快一倍

27. 对于两个 n 位无符号整数除法运算，以下关于不恢复余数算法要点的描述中，错误的是(　　)。

A. 起始时被除数在高位扩展 n 位0，以将其扩展为 $2n$ 位无符号整数

B. 为判断中间余数的正/负，需在余数寄存器的最高位前增加一位符号位

C. 至少需 $n+1$ 次循环执行“加/减”和“左移”操作才能得到 n 位商

D. 运算结果一定不会发生溢出，故无须通过得到最高位商来判断溢出

28. 对于IEEE 754单精度浮点加减运算,在对阶过程中,需计算两个阶码E_x和E_y之差的补码$[\Delta E]_{补}$。若$\Delta E \geqslant 128$或$\Delta E \leqslant -129$,则$[\Delta E]_{补}$发生溢出。假定$[E_x]_{移}$、$[-[E_y]_{移}]_{补}$和$[\Delta E]_{补}$的最高有效位分别记为E_{xs}、E_{ys}和E_{bs},则相应的溢出判断方程为(　　)。

A. $\text{Overflow}=\overline{E}_{xs}\overline{E}_{ys}E_{bs}+E_{xs}E_{ys}\overline{E}_{bs}$

B. $\text{Overflow}=\overline{E}_{xs}\overline{E}_{ys}\overline{E}_{bs}+E_{xs}\overline{E}_{ys}\overline{E}_{bs}$

C. $\text{Overflow}=\overline{E}_{xs}E_{ys}E_{bs}+E_{xs}\overline{E}_{ys}E_{bs}$

D. $\text{Overflow}=\overline{E}_{xs}\overline{E}_{ys}\overline{E}_{bs}+E_{xs}E_{ys}E_{bs}$

29. IEEE 754单精度浮点数加减运算的对阶过程中,需要计算两个阶码E_x和E_y之差的补码$[\Delta E]_{补}$。假设两个浮点数分别记为$[x]_{浮}$和$[y]_{浮}$,$[E_x]_{移}$、$[E_y]_{移}$和$[\Delta E]_{补}$的最高有效位分别记为E_{xs}、E_{ys}和E_{bs},当$[\Delta E]_{补}$发生溢出时,正确的处理方式是(　　)。

A. 中止当前程序的执行,调出相应的"溢出"异常处理程序执行

B. 当E_{xs}为1时置最终结果为$[x]_{浮}$;当E_{xs}为0时置最终结果为$[y]_{浮}$

C. 当E_{ys}为1时置最终结果为$[x]_{浮}$;当E_{ys}为0时置最终结果为$[y]_{浮}$

D. 当E_{bs}为0时置最终结果为$[x]_{浮}$;当E_{bs}为1时置最终结果为$[y]_{浮}$

30. 若两个float型变量(用IEEE 754单精度浮点格式表示)x和y的机器数分别表示为$x=$40E8 0000H,$y=$C204 0000H,则在计算$x+y$时,第一步对阶操作的结果$[\Delta E]_{补}$为(　　)。

A. 0000 0111　　B. 0000 0011　　C. 1111 1011　　D. 1111 1101

31. 对于IEEE 754单精度浮点数加减运算,只要对阶时得到的两个阶码之差的绝对值$|\Delta E|$大于或等于(　　),就无须继续进行后续处理,此时,运算结果直接取阶大的那个数。

A. 24　　B. 25　　C. 126　　D. 128

32. IEEE 754标准提供了以下4种舍入模式,其中平均误差最小的是(　　)。

A. 就近舍入(中间值时强迫为偶数)

B. 正向舍入(即朝$+\infty$方向舍入)

C. 负向舍入(即朝$-\infty$方向舍入)

D. 截断舍入(即朝0方向舍入)

【参考答案】

1. A	2. D	3. D	4. C	5. B	6. D	7. D	8. C	9. A	10. C
11. A	12. B	13. A	14. C	15. B	16. B	17. A	18. D	19. C	20. D
21. C	22. C	23. D	24. B	25. D	26. C	27. C	28. D	29. B	30. D
31 .B	32. A								

【部分题目的答案解析】

第20题

已知$[y]_{补}=$DCH$=$11011100B,所以$[-y]_{补}=$00100100B,$[x-2y]_{补}=[x]_{补}+[-2y]_{补}=[x]_{补}+[-y]_{补}<<1=01000100+00100100<<1=01000100+01001000=10001100=$8CH,从最后一步加操作来看,是两个正数相加,结果为负数,故溢出标志OF为1。综上所述,答案为选项D。

第 21 题

$[x/2+2y]_{补}=[x]_{补}>>1+[y]_{补}<<1=01000100>>1+11011100<<1=00100010+10111000=11011010=$DAH，从最后一步加操作来看，是一个正数和一个负数相加，因此一定不会溢出。综上所述，答案为 C。

第 23 题

关于原码一位乘法算法要点的描述中，错误的是 D 选项中的说法："移位时，将进位位、部分积和乘积部分一起进行算术右移"。因为原码一位乘法算法中，数值部分和符号分开处理，数值部分通过无符号乘的算法实现，在将进位位、部分积和乘积部分右移时，采用的是逻辑右移，即高位补 0 的办法，而不是采用算术右移方式。

第 24 题

假定一次 ALU 运算用 1 个时钟周期，移位一次用 1 个时钟周期，则最快的 32 位原码一位乘法所需的时钟周期数大约为 64。因为 32 位原码一位乘法的循环次数为 32，每次循环中，控制逻辑根据当前乘数寄存器的最低位确定是否在 ALU 中进行加法运算，这需要一个时钟周期；然后进行右移操作，这需要一个时钟周期。因此，每次循环需要两个时钟周期，一共需要大约 64 个时钟周期。

第 25 题

关于布斯补码一位乘法算法要点的描述中，错误的是 D 选项中的说法："移位时，将进位位、部分积和乘积部分一起进行算术右移"。因为补码一位乘法的算法中，并没有专门的进位位，而符号位作为部分积的一部分进行处理，所以，选项 D 的描述中提到进位位和部分积、乘积部分一起算术右移是错误的。

第 27 题

对于两个 n 位无符号整数除法运算，最大的商为 111…11/000…01=111…11，显然，运算结果没有产生溢出。因此，两个 n 位无符号整数除法运算肯定不会溢出，无须通过得到最高位商(第 n 位商 q_n)来判断溢出，也即只要有 n 次循环执行"加/减"和"左移"操作，就能得到第 $n-1\sim0$ 位的 n 位商($q_{n-1}\sim q_0$)。因此，选项 C 的说法是错误的。

第 28 题

对于 IEEE 754 单精度浮点加减运算，在对阶过程中，需要计算两个阶码 E_x 和 E_y 之差的补码$[\Delta E]_{补}$，$[\Delta E]_{补}=[E_x-E_y]_{补}=[E_x]_{移}+[-[E_y]_{移}]_{补}$。假定$[E_x]_{移}$、$[-[E_y]_{移}]_{补}$和$[\Delta E]_{补}$的最高有效位分别记为 E_{xs}、E_{ys}和 E_{bs}，若 $\Delta E\geqslant128$，则$[\Delta E]_{补}$发生正溢出，此时 E_{bs}为 1，而 E_x 一定为正数，即 $E_{xs}=1$，E_y 一定为负数，即$[E_y]_{移}$的符号位为 0，计算$[-[E_y]_{移}]_{补}$时，对$[E_y]_{移}$的各位取反、末位加 1，从而得到 $E_{ys}=1$，综合起来的逻辑表达式就是 $E_{xs}\,E_{ys}\,E_{bs}$；若 $\Delta E\leqslant-129$，则$[\Delta E]_{补}$发生负溢出，此时 E_{bs}为 0，而 E_x 一定为负数，即 $E_{xs}=0$，E_y 一定为正数，即$[E_y]_{移}$的符号位为 1，计算$[-[E_y]_{移}]_{补}$时，对$[E_y]_{移}$的各位取反、末位加 1，从而得到 $E_{ys}=0$，综合起来的逻辑表达式就是 $\overline{E}_{xs}\overline{E}_{ys}\overline{E}_{bs}$，因此，Overflow=$\overline{E}_{xs}\overline{E}_{ys}\overline{E}_{bs}+\overline{E}_{xs}\overline{E}_{ys}\overline{E}_{bs}$。

第 29 题

假设浮点数 x 和 y 的机器数分别记为$[x]_{浮}$和$[y]_{浮}$，$[\Delta E]_{补}=[E_x-E_y]_{补}=[E_x]_{移}+[-[E_y]_{移}]_{补}$，将$[E_x]_{移}$、$[E_y]_{移}$和$[\Delta E]_{补}$的最高有效位分别记为 E_{xs}、E_{ys}和 E_{bs}。当$[\Delta E]_{补}$

发生溢出时,若 $\Delta E \geqslant 128$,说明 x 的阶比 y 的阶至少大 128,y 的尾数至少要向右移 128 位,因而 y 被 x"吃掉",结果应该取 x,此时 $E_{xs}=1$;若 $\Delta E \leqslant -129$,说明 y 的阶比 x 的阶至少大 129,x 的尾数至少要向右移 129 位,因而 x 被 y"吃掉",结果应该取 y,此时 $E_{xs}=0$。因此,选项 B 是正确的。

第 30 题

x 和 y 的机器数分别表示为 $x=$40E8 0000H$=$0100 0000 1…,$y=$C204 0000H$=$1100 0010 0…,因此,$[E_x]_{移}=$1000 0001,$[E_y]_{移}=$1000 0100,$[\Delta E]_{补}=[E_x-E_y]_{补}=[E_x]_{移}+[-[E_y]_{移}]_{补}=$1000 0001$+$0111 1100$=$1111 1101。

第 31 题

对于 IEEE 754 单精度浮点数加减运算,若对阶时得到的两个阶码之差的绝对值 $|\Delta E|=24$,则说明阶小的那个数的尾数右移 24 位,进行尾数加减运算时,虽然其结果的前 24 位直接取阶大的那个数的相应位,但是,由于可以保留附加位,阶小的那个数右移后的尾数可能会在舍入时向前面一位进 1。例如,$1.00\cdots01\times2^{1}+1.10\cdots00\times2^{-23}=1.00\cdots01\times2^{1}+0.00\cdots00\mathbf{11}\times2^{1}=1.00\cdots01\mathbf{11}\times2^{1}$。其中,加粗的两位为保留的附加位,最终需要根据这两位进行舍入,显然,舍入后的结果为 $1.00\cdots10\times2^{1}$,并不等于阶大的那个数。若 $|\Delta E|=25$,则保留的附加位中,最左边第 1 位一定是 0,采用就近舍入时,这些附加位完全被丢弃。因此,$|\Delta E| \geqslant 25$ 时,可以使运算结果直接取阶大的那个数。

3.6 分析应用题

1. 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。

2. 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 3.1，给出对表中“异常”数据的说明。

表 3.1　题 2 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表 3.2 所示，表中机器数用十六进制表示。

表 3.2　题 2 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。

表 3.2 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 00H，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。

3. 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,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 $-\infty$ 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。

4. 设 $A_4 \sim A_1$ 和 $B_4 \sim B_1$ 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 $C_4 \sim C_1$ 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:$C_1=A_1B_1+(A_1+B_1)C_0$

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

$C_3=A_3B_3+(A_3+B_3)A_2B_2+(A_3+B_3)(A_2+B_2)\ A_1B_1+(A_3+B_3)(A_2$

$$+B_2)(A_1+B_1)C_0$$

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

5. 某字长为 8 位的计算机中，x 和 y 为无符号整数，已知 $x=68$，$y=80$，x 和 y 分别存放在寄存器 A 和 B 中。请回答下列问题(要求最终用十六进制表示二进制序列)。

(1) 寄存器 A 和 B 中的内容分别是什么？

(2) 若 x 和 y 相加后的结果存放在寄存器 C 中，则寄存器 C 中的内容是什么？运算结果是否正确？加法器最高位的进位 Cout 是什么？零标志 ZF 和进位标志 CF 各是什么？

(3) 若 x 和 y 相减后的结果存放在寄存器 D 中，则寄存器 D 中的内容是什么？运算结果是否正确？加法器最高位的进位 Cout 是什么？零标志 ZF 和借位标志 CF 各是什么？

(4) 无符号整数加/减运算时，加法器最高位进位 Cout 的含义是什么？它与进/借位标志 CF 的关系是什么？

(5) 无符号整数一般用来表示什么信息？为什么通常不对无符号整数的运算结果判断溢出？

【分析解答】

(1) $x=68=0100\ 0100\ B=44H$；$y=80=0101\ 0000\ B=50H$。所以，寄存器 A 和寄存器 B 中的内容分别是 44H 和 50H。

(2) $x+y=0100\ 0100+0101\ 0000=(0)\ 1001\ 0100=94H$，所以，寄存器 C 中的内容为 94H，对应的真值为 148，运算结果正确。加法器最高位的进位 Cout 为 0。因为结果不为 0，所以 ZF=0；进位标志 CF=Cout=0。

(3) $x-y=x+[-y]_{补}=0100\ 0100+1011\ 0000=(0)\ 1111\ 0100=F4H$，所以，寄存器 D 中的内容为 F4H，对应的真值为 244，运算结果不正确，这是因为相减结果为负数造成的。加法器最高位的进位 Cout 为 0。因为结果不为 0，所以 ZF=0；借位标志为 CF=Cout$\oplus$1=1。

(4) 在加法器中进行无符号整数加法运算时，若加法器最高位进位 Cout=1，则表示实际结果大于最大可表示数 255；在加法器中进行无符号整数减法运算时，若加法器最高位进位 Cout=1，则表示被减数大于减数，反之被减数小于减数。因此，在无符号数相加时，CF 就等于 Cout，表示进位；在无符号数相减时，通常将最高进位 Cout 取反来作为借位标志 CF。也就是说，无符号整数相减时，CF=$\overline{\text{Cout}}$，CF=1 表示有借位。

(5) 无符号整数一般用来表示地址(指针)信息，当两个地址相加结果大于最大地址而取低位地址时，相当于取模，即采用地址循环运算。因此，通常不需要判断其运算结果是否溢出，即不考虑溢出标志 OF。

6. 假设某字长为 8 位的计算机中，带符号整数采用补码表示，$x=-68$，$y=-80$，x 和 y 分别存放在寄存器 A 和寄存器 B 中。请回答下列问题(要求最终用十六进制表示二进制序列)。

(1) 寄存器 A 和寄存器 B 中的内容分别是什么？

(2) 若 x 和 y 相加后的结果存放在寄存器 C 中，则寄存器 C 中的内容是什么？运算结果是否正确？加法器最高位的进位 Cout 是什么？溢出标志 OF、符号标志 SF 和零标志 ZF 各是什么？

(3) 若 x 和 y 相减后的结果存放在寄存器 D 中,则寄存器 D 中的内容是什么?运算结果是否正确?此时,加法器最高位的进位 Cout 是什么?溢出标志 OF、符号标志 SF 和零标志 ZF 各是什么?

(4) 对于带符号整数的减法运算,能否直接根据 CF 的值对两个带符号整数的大小进行比较?

【分析解答】

(1) $[-68]_{补}=[-1000100]_{补}=1011\ 1100B=BCH$。$[-80]_{补}=[-1010000]_{补}=1011\ 0000B=B0H$。所以,寄存器 A 和寄存器 B 中的内容分别是 BCH 和 B0H。

(2) $[x+y]_{补}=[x]_{补}+[y]_{补}=1011\ 1100B+1011\ 0000B=(1)\ 0110\ 1100B=6CH$,最高位前面的一位 1 被丢弃,因此,寄存器 C 中的内容为 6CH,对应的真值为+108,结果不正确。加法器最高位向前面的进位 Cout 为 1。溢出标志位 OF 可采用以下任意一条规则判断得到。规则 1:若两个加数的符号位相同,但与结果的符号位相异,则溢出;规则 2:若最高位上的进位和次高位上的进位不同,则溢出。对于本题,通过这两个规则都判断出结果溢出,因此溢出标志 OF 为 1,说明寄存器 C 中的内容不是正确的结果。$x+y$ 的正确结果应是$-68+(-80)=-148$,而运算的结果为 108,两者不等。其原因是因为 $x+y$ 的值(即-148)小于 8 位补码可表示的最小值(即-128),即结果发生了溢出;结果的第一位(最高位)0 为符号标志位 SF,即 SF=0,表示结果为正数;因为结果不为 0,所以零标志 ZF=0。

(3) $[x-y]_{补}=[x]_{补}+[-y]_{补}=1011\ 1100B+0101\ 0000B=(1)\ 0000\ 1100B=0CH$,最高位前面的一位 1 被丢弃,因此,寄存器 D 中的内容为 0CH,对应的真值为+12,结果正确。加法器最高位向前面的进位 Cout 为 1。两个加数的符号位相异一定不会溢出,因此溢出标志 OF=0,说明寄存器 D 中的内容是真正的结果;结果的第一位(最高位)0 为符号标志位 SF,即 SF=0,表示结果为正数;因为结果不为 0,所以零标志 ZF=0。

(4) 对于带符号整数的减法运算,无法直接根据 CF 的值判断两个带符号整数的大小。例如,对于 $x=-68$,$y=80$,$[x-y]_{补}=[x]_{补}+[-y]_{补}=1011\ 1100B+1011\ 0000B=(1)\ 0110\ 1100B$,得到的 Cout 为 1,因此,CF=Cout⊕1=0,表示没有借位,推断出被减数应该大于减数,即$-68>80$,显然这是不正确的。因此,带符号运算中不考虑 CF 标志。

7. 某计算机标志寄存器包含 4 个标志位:CF 为进/借位标志;OF 为溢出标志;SF 为符号标志;ZF 为零标志。请说明在无符号数和带符号整数两种情况下,以下各种比较运算的逻辑判断表达式。

(1) 等于　(2) 大于　(3) 小于　(4) 大于或等于　(5) 小于或等于

【分析解答】

要比较两个数的大小,通常对这两个数先做减法,根据相减的结果生成相应的标志位,最后根据标志位判断大小。在无符号数相减时,一般不考虑 SF 和 OF 标志;在带符号整数相减时,一般不考虑 CF 标志。

假设被减数的机器数为 X,减数的机器数为 Y,则在加法器中计算两数的差时,计算公式为 $X-Y=X+(-Y)_{补}$。以下举两个例子来说明。

假定 $X=1001$,$Y=1100$,则在 4 位加法器中执行以下运算:$1001-1100=1001+0100=(0)1101$。若是无符号数比较,则是 9 和 12 相比,显然,ZF=0,CF=1;若是带符号整数(补码表示),则是-7 和-4 比较,显然,ZF=0,OF=0,SF=1。

假定 $X=1001$，$Y=0100$，则在 4 位加法器中执行以下运算：1001－0100＝1001＋1100＝(1)0101。若是无符号数比较，则是 9 和 4 相比，显然，ZF＝0，CF＝0；若是带符号整数，则是－7 和 4 比较，显然，ZF＝0，OF＝1，SF＝0。

以下分别说明无符号数和带符号整数两种情况下各种比较运算的逻辑判断表达式。

在无符号数的情况下：

(1) 等于：相减后结果为零，即 $F=\mathrm{ZF}$。

(2) 大于：没有借位且相减后不为 0，即 $F=\overline{\mathrm{CF}}\cdot\overline{\mathrm{ZF}}=\overline{\mathrm{CF}+\mathrm{ZF}}$。

(3) 小于：有借位且相减后不为 0，即 $F=\mathrm{CF}\cdot\overline{\mathrm{ZF}}$。

(4) 大于或等于：没有借位或相减后结果为 0，即 $F=\overline{\mathrm{CF}}+\mathrm{ZF}$。

(5) 小于或等于：有借位或相减后结果为 0，即 $F=\mathrm{CF}+\mathrm{ZF}$。

在带符号整数的情况下：

(1) 等于：相减后结果为零，即 $F=\mathrm{ZF}$。

(2) 大于：相减后结果不为 0，并且，不溢出时为正，溢出时为负，即 $F=\overline{\mathrm{ZF}}\cdot\overline{(\mathrm{SF}\oplus\mathrm{OF})}$。

(3) 小于：相减后结果不为 0，并且，不溢出时为负，溢出时为正，即 $F=\overline{\mathrm{ZF}}\cdot(\mathrm{SF}\oplus\mathrm{OF})$。

(4) 大于或等于：相减后结果为 0，或者，不溢出时为正，溢出时为负，即 $F=\mathrm{ZF}+\overline{(\mathrm{SF}\oplus\mathrm{OF})}$。

(5) 小于或等于：相减后结果为 0，或者，不溢出时为负，溢出时为正，即 $F=\mathrm{ZF}+(\mathrm{SF}\oplus\mathrm{OF})$。

可以对照上述判断表达式，验证上述例子。无符号整数 9 和 12 是小于关系，相减后得到的标志 ZF＝0，CF＝1，故 $F=\mathrm{CF}\cdot\overline{\mathrm{ZF}}=1$。无符号整数 9 和 4 是大于关系，相减后得到的标志 CF＝0，ZF＝0，故 $F=\overline{\mathrm{CF}}\cdot\overline{\mathrm{ZF}}=1$。带符号整数－7 和－4 是小于关系，相减后得到的标志 ZF＝0，OF＝0，SF＝1，故 $F=\overline{\mathrm{ZF}}\cdot(\mathrm{SF}\oplus\mathrm{OF})=1$。带符号整数－7 和 4 是小于关系，相减后得到的标志 ZF＝0，OF＝1，SF＝0，故 $F=\overline{\mathrm{ZF}}\cdot(\mathrm{SF}\oplus\mathrm{OF})=1$。

8. 填写表 3.3，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、截断操作后的结果。

表 3.3　题 8 用表

模　式	x		y		$x\times y$(截断前)		$x\times y$(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3.4。

表3.4 题8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

对表3.4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,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3.4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。

9. 考虑以下C语言程序代码:

```
int func1(unsigned short si)
{
    return(si * 256);
}
int func2(unsigned short si)
{
    return(si/256);
}
int func3(unsigned short si)
{
    return(((short)si * 256)/256);
}
int func4(unsigned short si)
{
    return(short)((si * 256)/256);
}
```

请回答下列问题：

(1) 假设计算机硬件不提供乘除运算功能，能否用移位运算实现上述函数功能？函数 func1、func2、func3 和 func4 得到的结果各有什么特征？

(2) 填写表 3.5(要求机器数用十六进制表示)，并对表中的“异常”数据进行分析。

表 3.5　题 9 用表

si		func1(si)		func2(si)		func3(si)		func4(si)	
机器数	值	机器数	值	机器数	值	机器数	值	机器数	值
	127								
	128								
	255								
	256								
	65 535								

(3) 对于函数 func1 来说，用一位乘运算所花的时间开销大约是用移位运算的多少倍？对于函数 func2 来说，用不恢复余数除法所花的开销大约是用移位运算的多少倍？

【分析解答】

(1) 编译器在处理变量与常数相乘时，往往以移位、加法和减法的组合运算来代替乘法运算。例如，对于表达式 $x * 20$，编译器可以利用 $20=16+4=2^4+2^2$，将 $x * 20$ 转换为 $(x<<4)+(x<<2)$，这样，一次乘法转换成了两次移位和一次加法。不管是无符号整数还是带符号整数的乘法，即使乘积溢出时，利用移位和加减运算组合的方式得到的结果都是和采用直接相乘的结果是一样的。

为了缩短除法运算的时间，编译器在处理一个变量与一个 2 的幂次形式的整数相除时，常采用右移运算来实现。无符号整数除法采用逻辑右移方式，带符号整数除法采用算术右移方式。两个整数相除，结果也一定是整数，在不能整除时，其商采用朝零方向舍入的方式，也就是截断方式，即将小数点后的数直接去掉，例如，7/3=2，−7/3=−2。

对于无符号整数来说，采用逻辑右移时，高位补 0，低位移出，因此，移位后得到的商的值只可能变小而不会变大，即商朝零方向舍入。因此，不管是否能够整除，采用移位方式和直接相除得到的商完全一样。但是，对于带符号整数 x 来说，如题 3 的分析解答中所说的那样，当计算 $x/2^k$ 时，若 $x<0$，则不能直接将 x 算术右移 k 位，而应该先将 x 加上偏移量 (2^k-1)，然后再算术右移 k 位。

因为 $256=2^8$，所以题目给出的函数中的乘、除运算可以分别用左、右移运算来实现。可用“左移 8 位”代替“乘 256”的操作，用“右移 8 位”代替“除以 256”的操作。func1(si)相当于将 si 逻辑左移 8 位，结果的最后 8 位都为 0；func2(si)相当于将 si 逻辑右移 8 位，结果的范围是 0～255；func3(si)相当于将 si 先算术左移 8 位，再算术右移 8 位，所以结果的范围是−128～127；func4(si)相当于将 si 先逻辑左移 8 位，再逻辑右移 8 位，最后以带符号整数类型返回。因为最后是逻辑右移，高位补 0，所以，返回的总是正数，结果的范围是 0～255。

(2) 函数 func1、func2、func3 和 func4 的执行结果填表如表 3.6 所示。