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的相关知识，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) 了解原码加减运算的基本原理。

(4) 掌握补码加减运算方法，并能设计补码加减运算器。

(5) 了解定点数乘法和除法运算的基本思想。

(6) 了解专用的阵列乘法器和阵列除法器的基本思想。

(7) 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。

(8) 掌握浮点数加减运算的过程和方法。

(9) 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。

(10) 了解浮点数乘法和除法运算的基本思想。

(11) 掌握算术逻辑单元 ALU 的功能和结构。

(12) 了解串行加法器和快速并行加法器的基本工作原理。

(13) 了解定点运算器(即定点运算数据通路)的基本结构。

(14) 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，对这些内容掌握得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、两位一乘算法

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、阵列除法器、浮点数乘除运算、定点运算部件、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留作课后阅读，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如下。(1)给定一组无符号数和有符号整数变量的值，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($2^{31}-1$)，将该数赋值给某个 int 类型变量，再将其转换为 short、unsigned short、unsigned int、float 等类型，查看其结果，并进行分析解释；(4)对于某个十进制有效位数多于 8 的实数，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)，计算“1＋4294967295”、“1－4294967295”的值。对于带符号整数(如 int)，计算“2147483647＋1”、“－2147483648－1”的值。对于浮点数，分别计算“(1.0＋123456.789e30)＋(－123456.789e30)”和“1.0＋(123456.789e30＋(－123456.789e30))”的值，并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 3 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可构成 ALU，以实现基本的加/减算术运算和基本逻辑运算。ALU 的输入有：两个操作数、一位低位来的进位、一组操作控制信号，ALU 的输出有：一个结果、一位向高位的进位，以及零标志(ZF)和溢出标志(OF)等。

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。除基本的与或非等逻辑运算外，主要的运算包括以下几种。

(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) 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) 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)或符号(补码乘法)，否则溢出。对于一位乘法，n位数相乘大约需要n次加减运算和n次右移运算。

(5) 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) 加减运算：按照对阶、尾数加减、规格化、舍入和溢出判断步骤完成。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) 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)

对补码整数在高位直接补符的操作，称为“符号扩展”。

扩展器(Extender)

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

半加器(Half Adder)

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)

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)

能进行 n 位加法运算的部件。

行波进位(Ripple Carry)

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接起来实现。进位方式采用行波进位方式。

先行(超前)进位(Carry LookAhead,CLA)

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)

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这两个组进位辅助函数,促使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)

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)

是一种一位补码乘法算法,用于带符号数的乘法运算,由 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,基-4 Booth 算法(Modified Booth's Algorithm,MBA)

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(Exponent Underflow)

在浮点数运算中,当运算的结果其指数(阶码)比最小允许值还小,此时,运算结果发生阶码下溢,也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(Exponent Overflow)

在浮点数运算中,当运算的结果其指数(阶码)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-∞。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,它们是添加的附加位。增设附加位后,能保证运算结果具有一定的精度,但最终必须将附加位去掉,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)

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 Register Set,GRS)

CPU中提供了若干个通用寄存器,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)

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。

桶形移位器(Barrel Shifter)

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3种基本实现方式。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。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。并行进位加法器也称为快速加法器。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。

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。也即采用模运算方式。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的负数的补码得到。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、末尾加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:现代计算机中浮点数采用IEEE 754标准表示,因此在进行两个浮点数加减运算时,必须考虑原码的加减运算,因为,IEEE 754规定浮点数的尾数都用原码表示。

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增加保护位和舍入吗?

答:不需要。整数运算的结果还是整数,没有误差,无须考虑增加保护位,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为:"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为:"若结果的符号与两个加数的符号不同,则发生溢出"。规则2为:

“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 0101B右移一位后的值为(　　)。

   A. 0100 1010B　　B. 0100 1011B

   C. 1000 1010B　　D. 1100 101B

2. 8位补码定点整数1001 0101B右移一位后的值为(　　)。

   A. 0100 1010B　　B. 0100 1011B

   C. 1000 1010B　　D. 1100 1010B

3. 8位补码定点整数1001 0101B左移一位后的值为(　　)。

   A. 1010 1010B　　B. 0010 1010B

   C. 0010 1011B　　D. 溢出

4. 8位补码定点整数1001 0101B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 0101B扩展8位后的值为(　　)。

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

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

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，不考虑线延迟，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. 74181 ALU 的功能是(　　)。

A. 实现 16 种 4 位算术运算

B. 实现 16 种 4 位逻辑运算

C. 实现 16 种 4 位算术和逻辑运算

D. 实现 4 位乘法运算和 4 位除法运算

11. 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

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，此时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 0101B、0010 0110B、0　　B. 0100 0101B、0010 0110B、1

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

13. 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，此时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 0101B、0010 0110B、0　　B. 0100 0101B、1101 1001B、1

C. 0100 0101B、1101 1010B、0　　D. 0100 0101B、1101 1010B、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，此时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 1011B、1101 1010B、0　　B. 1011 1011B、1101 1010B、1

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

15. 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，此时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 1011B、1101 1010B、0　　B. 1011 1011B、1101 1010B、1

C. 1011 1011B、0010 0110B、0　　D. 1011 1011B、0010 0101B、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. 1FH、0　　B. 20H、0

C. 1FH、1　　D. 20H、1

17. 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

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

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. 32H、0　　B. 32H、1

C. FCH、0　　D. FCH、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 $x-2y$ 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

22. 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

23. 假定有两个整数用8位补码分别表示为 $r1=$F5H，$r2=$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. $r1+r2$　　B. $r1-r2$

C. $r1\times r2$　　D. $r1/r2$

24. 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