南京大学
NANJING UNIVERSITY

运算方法和运算部件

殷亚凤
智能软件与工程学院
苏州校区南雍楼东区225
yafeng@nju.edu.cn，https://yafengnju.github.io/

运算方法和运算部件

- 高级语言和机器指令中的运算
- 基本运算部件
- 定点数运算
- 整数乘除运算
- 浮点数运算

高级语言和机器指令中的运算

- **C语言程序中涉及的运算**
- **算术运算（最基本的运算）**
  - 无符号数、带符号整数、浮点数的运算
- **按位运算**
  - 用途
    - 对一个**位串**实现“掩码”（mask）操作或相应的其他处理（主要用于对**多媒体数据**或**控制信息**进行处理）
  - 操作
    - **按位或：“|”**
    - **按位与：“&”**
    - **按位取反：“~”**
    - **按位异或：“^”**

问题：如何从一个16位采样数据y中提取高位字节，并使低字节为0？

可用“&”实现“掩码”操作：y & 0xFF00

例如，当y=0x2C0B时，通过掩码操作得到结果为：0x2C00

高级语言和机器指令中的运算

- **C语言程序中涉及的运算**
  - **逻辑运算**
    - 用途
      - 用于关系表达式的运算
      例如，if （x>y and i<100）then ……中的“and”运算
    - 操作
      - **“||”表示“或”运算**
      - **“&&”表示“与”运算**
        例如， if ((x>y) && (i<100)) then ……
      - **“!”表示“非”运算**
    - **与按位运算的差别**
      - 符号表示不同：& ~ && ；| ~ ||； ……
      - 运算过程不同：按位 ~ 整体
      - 结果类型不同：位串 ~ 逻辑值

高级语言和机器指令中的运算

- **C语言程序中涉及的运算**
- **移位运算**
  - 用途
    - 提取部分信息
    - 扩大或缩小数值的2、4、8…倍
  - 操作
    - 左移:：x<<k;　右移： x>>k
    - 不区分是逻辑移位还是算术移位，由x的类型确定
    - **无符号数**：逻辑左移、逻辑右移

      高（低）位移出，低（高）位补0

      问题：何时可能发生溢出？如何判断溢出？（**若高位移出的是1，则左移时发生溢出**）
    - **带符号整数**：算术左移、算术右移

      左移：高位移出，低位补0。（溢出判断：若移出的位不等于新的符号位，则溢出）

      右移：低位移出，高位补符，可能发生数据丢失。

南京大学 NANJING UNIVERSITY

高级语言和机器指令中的运算

- **C语言程序中涉及的运算**

- **位扩展和位截断运算**
  - 用途
    - 在进行类型转换时，可能需要数据的扩展或截断
  - 操作
    - 没有专门的操作运算符，根据类型转换前后数据长短来确定是扩展还是截断
    - “扩展”：短数转为长数；“截断”，长数转为短数
    - **扩展**

      无符号数：0扩展，即：前面补0

      带符号整数：符号扩展，即：前面补符号
    - **截断**

      强行将一个长数的高位丢弃，故可能会发生“溢出”

例1：在大端机上输出si, usi, i, ui的十进制和十六进制值是什么？

```
short si = -12345;
unsigned short usi = si;
int i = si;
unsigned ui = usi ;
```

si = -12345　　　CF C7
usi = 53191　　　CF C7
i = -12345　　**FF FF** CF C7
ui = 53191　　**00 00** CF C7

高级语言和机器指令中的运算

- **C语言程序中涉及的运算**

- **位扩展和位截断运算**
  - 用途
    - 在进行类型转换时，可能需要数据的扩展或截断
  - 操作
    - 没有专门的操作运算符，根据类型转换前后数据长短来确定是扩展还是截断
    - “扩展”：短数转为长数；“截断”，长数转为短数
    - **扩展**

      无符号数：0扩展，即：前面补0

      带符号整数：符号扩展，即：前面补符号
    - **截断**

      强行将一个长数的高位丢弃，故可能会发生“溢出”

例2：在大端机上执行后，i和j是否相等？

```
int i = 53191;
short si = (short)i;
int j = si;
```

不相等！

i = 53191	00 00 CF C7
si = -12345	CF C7
j = -12345	FF FF CF C7

原因：**对i截断时发生了“溢出”**，即：53191截断为16位数时，无法正确表示！

南京大学 NANJING UNIVERSITY

高级语言和机器指令中的运算

- **MIPS指令中涉及的运算**

表 3.1 MIPS 指令系统中涉及运算的部分指令

指令类型	指令名称	汇编形式举例	含义	所需运算
逻辑运算	and	and \$1, \$2, \$3	\$1= \$2 & \$3	按位与
	or	or \$1, \$2, \$3	\$1= \$2 | \$3	按位或
	nor	nor \$1, \$2, \$3	\$1=~(\$2 | \$3)	按位或非
	and immediate	andi \$1, \$2,100	\$1= \$2&100	按位与
	or immediate	ori \$1, \$2,100	\$1= \$2|100	按位或
	shift left logical	sll \$1, \$2,10	\$1= \$2<<10	逻辑左移
	shift right logical	srl \$1, \$2,10	\$1= \$2>>10	逻辑右移

涉及到的操作数：32/16位 逻辑数

涉及到的操作：按位与 / 按位或 / 按位或非 / 左移 / 右移

高级语言和机器指令中的运算

- **MIPS指令中涉及的运算**

定点算术运算*	shift right arithmetic	`sra $1,$2,10`	`$1=$2>>10`	算术右移
	add	`add $1,$2,$3`	`$1=$2+$3`	整数加(判溢出)
	subtract	`sub $1,$2,$3`	`$1=$2−$3`	整数减(判溢出)
	add immediate	`addi $1,$2,100`	`$1=$2+100`	符号扩展、整数加(判溢出)
	sub immediate	`subi $1,$2,100`	`$1=$2−100`	符号扩展、整数减(判溢出)
	add unsigned	`addu $1,$2,$3`	`$1=$2+$3`	整数加(不判溢出)
	subtract unsigned	`subu $1,$2,$3`	`$1=$2−$3`	整数减(不判溢出)
	add immediate unsigned	`addiu $1,$2,100`	`$1=$2+100`	0扩展、整数加(不判溢出)
	multiply	`mult $2,$3`	`Hi,Lo=$2×$3`	带符号整数乘
	multiply unsigned	`multu $2,$3`	`Hi,Lo=$2×$3`	无符号整数乘
	divide	`div $2,$3`	`Lo=$2÷$3`	带符号整数除
			`Hi=$2 mod $3`	Lo=商,Hi=余数
	divide unsigned	`divu $2,$3`	`Lo=$2÷$3`	无符号整数除
			`Hi=$2 mod $3`	Lo=商,Hi=余数

涉及到的操作数：32/16位 无符号数， 32/16位带符号数；

涉及到的操作：加 / 减 / 乘 / 除（有符号 / 无符号）。

高级语言和机器指令中的运算

- **MIPS指令中涉及的运算**

定点数据传送	load word	lw \$1,100(\$2)	\$1=mem[\$2+100]	符号扩展并整数加
	store word	sw \$1,100(\$2)	mem[\$2+100]=\$1	符号扩展并整数加
	load half unsigned	lhu \$1,100(\$2)	\$1=mem[\$2+100]	符号扩展并整数加,0 扩展
	store half	sh \$1,100(\$2)	mem[\$2+100]=\$1	符号扩展并整数加,符号扩展
	load byte unsigned	lbu \$1,100(\$2)	\$1=mem[\$2+100]	符号扩展并整数加,0 扩展
	store byte	sb \$1,100(\$2)	mem[\$2+100]=\$1	符号扩展并整数加,符号扩展
	load upper immediate	lui \$1,100	$\$1=100\times2^{16}$	逻辑左移 16 位

涉及到的操作数： 32/16位带符号数（偏移量可以是负数）

涉及到的操作：加 / 减 / 符号扩展 / 0扩展

高级语言和机器指令中的运算

- **MIPS指令中涉及的运算**

指令类型	指令名称	汇编形式举例	含义	所需运算
浮点算术运算	FP add single	add.s \$f2, \$f4, \$f6	\$f2=\$f4+\$f6	单精度浮点加
	FP subtract single	sub.s \$f2, \$f4, \$f6	\$f2=\$f4−\$f6	单精度浮点减
	FP multiply single	mul.s \$f2, \$f4, \$f6	\$f2=\$f4×\$f6	单精度浮点乘
	FP divide single	div.s \$f2, \$f4, \$f6	\$f2=\$f4÷\$f6	单精度浮点除
	FP add double	add.d \$f2, \$f4, \$f6	\$f2=\$f4+\$f6	双精度浮点加
	FP subtract double	sub.d \$f2, \$f4, \$f6	\$f2=\$f4−\$f6	双精度浮点减
	FP multiply double	mul.d \$f2, \$f4, \$f6	\$f2=\$f4×\$f6	双精度浮点乘
	FP divide double	div.d \$f2, \$f4, \$f6	\$f2=\$f4÷\$f6	双精度浮点除

- **涉及到的浮点操作数**：32位单精度 / 64位双精度浮点数
- **涉及到的浮点操作**：加 / 减 / 乘 / 除

- **MIPS提供专门的浮点数寄存器**：

32个32位单精度浮点数寄存器：$f0, $f1, ……, $f31

连续两个寄存器（一偶一奇）存放一个双精度浮点数

高级语言和机器指令中的运算

- **MIPS指令中涉及的运算**

浮点数据传送	load word corp.1 store word corp.1	lwcl \$f1,100(\$2) swcl \$f1,100(\$2)	\$f1=mem[\$2+100] mem[\$2+100]=\$f1	符号扩展并整数加 符号扩展并整数加

涉及到的浮点操作数： 32位单精度浮点数

涉及到的浮点操作：传送操作（与定点传送一样）

涉及到定点操作：加 / 减（用于地址运算）

例：实现将两个浮点数从内存取出相加后再存回到内存的指令序列为：

```
lwcl    $f1, x($s1)
lwcl    $f2, y($s2)
add.s  $f4, $f1, $f2
swlc    $f4, z(s3)
```

高级语言和机器指令中的运算

- **MIPS指令中涉及的运算**
  - **涉及到的操作数**：
    - 无符号整数、带符号整数
    - 逻辑数
    - 浮点数
  - **涉及到的运算**
    - 定点数运算
      - 带符号整数运算：取负 / 符号扩展 / 加 / 减 / 乘 / 除 / 算术移位
      - 无符号整数运算：0扩展 / 加 / 减 / 乘 / 除
    - 逻辑运算
      - 逻辑操作：与 / 或 / 非 / …
      - 移位操作：逻辑左移 / 逻辑右移
    - 浮点数运算：加、减、乘、除

实现MIPS定点运算指令的思路：

首先实现一个能进行基本算术运算（加/减）和基本逻辑运算（与/或/或非）、并能生成基本条件码（ZF/VF/CF/NF）的**ALU**，再由ALU和移位器实现乘除运算器。

运算方法和运算部件

- 高级语言和机器指令中的运算
- **基本运算部件**
- 定点数运算
- 整数乘除运算
- 浮点数运算

基本运算部件

- **全加器**：输入为加数、被加数和低位进位C_{in}，输出为和F、进位C_{out}

X_i	Y_i	C_{i-1}	F_i	C_i
A	B	Cin	F	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

真值表

$$F=\overline{A}\bullet\overline{B}\bullet \mathrm{Cin}+\overline{A}\bullet B\bullet\overline{\mathrm{Cin}}+A\bullet\overline{B}\bullet\overline{\mathrm{Cin}}+A\bullet B\bullet \mathrm{Cin}$$

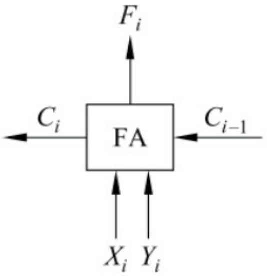
$$\mathrm{Cout}=\overline{A}\bullet B\bullet \mathrm{Cin}+A\bullet\overline{B}\bullet \mathrm{Cin}+A\bullet B\bullet\overline{\mathrm{Cin}}+A\bullet B\bullet \mathrm{Cin}$$

化简后：

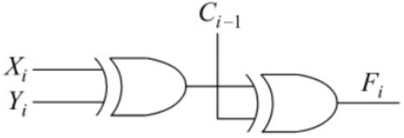
$$F=A\oplus B\oplus \mathrm{Cin}$$

$$\mathrm{Cout}=A\bullet B+A\bullet \mathrm{Cin}+B\bullet \mathrm{Cin}$$

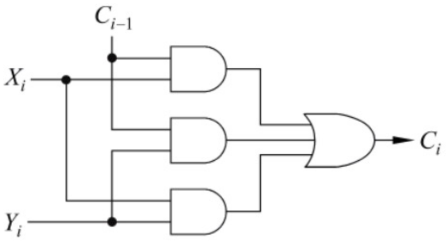
逻辑符号



全加和Fi的生成



全加进位Ci的生成



基本运算部件

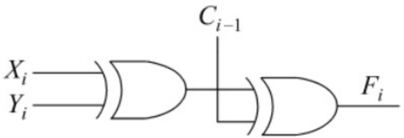
- **串行进位加法器**

全加和、全加进位：

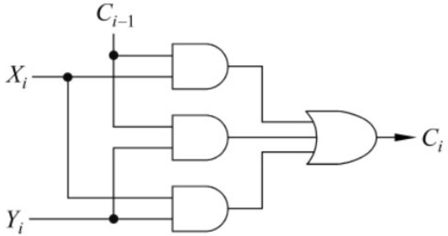
$$F=A\oplus B\oplus \mathrm{Cin}$$

$$\mathrm{Cout}=A\bullet B+A\bullet \mathrm{Cin}+B\bullet \mathrm{Cin}$$

全加和Fi的生成



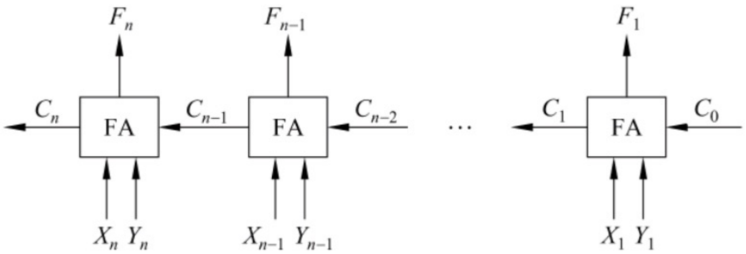
全加进位Ci的生成



- 求和Sum延迟为6ty；进位Carryout延迟为 2 ty

  (假定一个与门/或门延迟为1ty，异或门的延迟则为 3 ty)

n位串行进位加法器



串行加法器的缺点：

进位按串行方式传递，速度慢！

问题：n位串行加法器从C0到Cn的延迟时间为多少？ **2n级门延迟！**

最后一位和数的延迟时间为多少？ **2n+1级门延迟！**

NANJING UNIVERSITY

基本运算部件

- **并行进位加法器**
- **为什么用先行进位方式？**

串行进位加法器采用串行逐级传递进位，电路延迟与位数成正比关系。因此，现代计算机采用一种**先行进位(Carry look ahead)方式**。

- **如何产生先行进位？**

定义辅助函数：$G_i=X_iY_i$…进位生成函数

$P_i=X_i+Y_i$…进位传递函数

通常把实现上述逻辑的电路称为进位生成/传递部件

- **全加逻辑方程：$F_i=X_i\oplus Y_i\oplus C_{i-1}$　$C_{i+1}=X_iY_i+(X_i+Y_i)C_i=G_i+P_iC_i$ $(i=0,1,\ldots n)$**

设n=4,则：$C_1=G_0+P_0C_0$

$C_2=G_1+P_1C_1=G_1+P_1G_0+P_1P_0C_0$

$C_3=G_2+P_2C_2=G_2+P_2G_1+P_2P_1G_0+P_2P_1P_0C_0$

$C_4=G_3+P_3C_3=G_3+P_3G_2+P_3P_2G_1+P_3P_2P_1G_0+P_3P_2P_1P_0C_0$

由上式可知: **各进位之间无等待**，相互独立并同时产生。通常把实现上述逻辑的电路称为4位先行进位部件（4位CLU）

基本运算部件

• 并行进位加法器

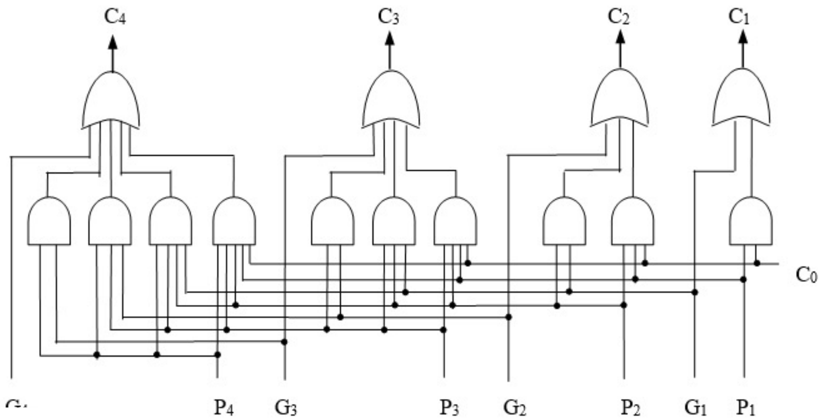
全先行进位加法器(CLA)：
所有进位独立并同时生成

$C_1=G_1+P_1C_0$

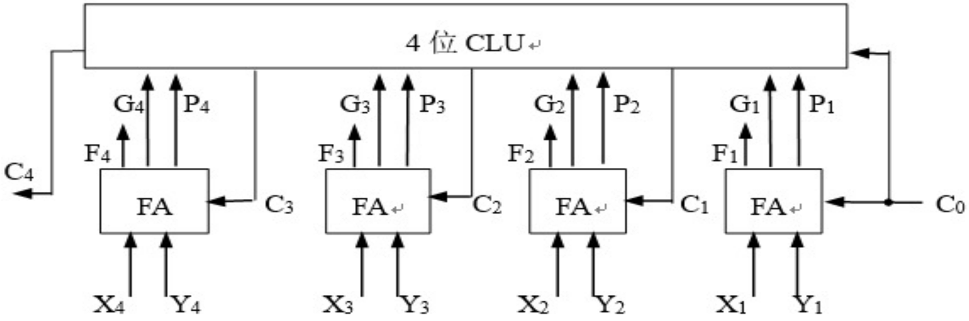
$C_2=G_2+P_2C_1=G_2+P_2G_1+P_2P_1C_0$

$C_3=G_3+P_3C_2=G_3+P_3G_2+P_3P_2G_1+P_3P_2P_1C_0$

$C_4=G_4+P_4C_3=G_4+P_4G_3+P_4P_3G_2+P_4P_3P_2G_1+P_4P_3P_2P_1C_0$



4位先行进位部件(CLU)



4位全先行进位加法器CLA

$G_i=X_iY_i$

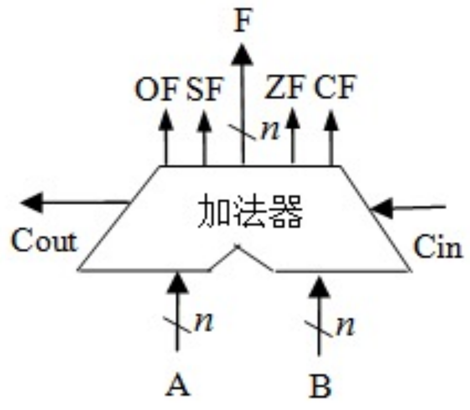
$P_i=X_i+Y_i$ (或 $P_i=X_i\oplus Y_i$)

$F_i=X_i\oplus Y_i\oplus C_{i-1}$

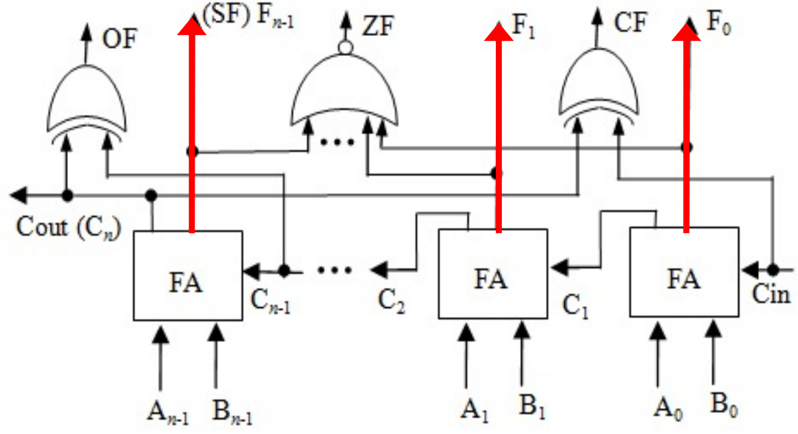
NANJING UNIVERSITY

基本运算部件

- **带标志加法器**
  - n位加法器无法用于两个n位带符号整数（补码）相加，无法判断是否溢出
  - 程序中经常需要比较大小，通过（在加法器中）做减法得到的标志信息来判断



带标志加法器符号



带标志加法器的逻辑电路

溢出标志OF：OF=$C_n \oplus C_{n-1}$

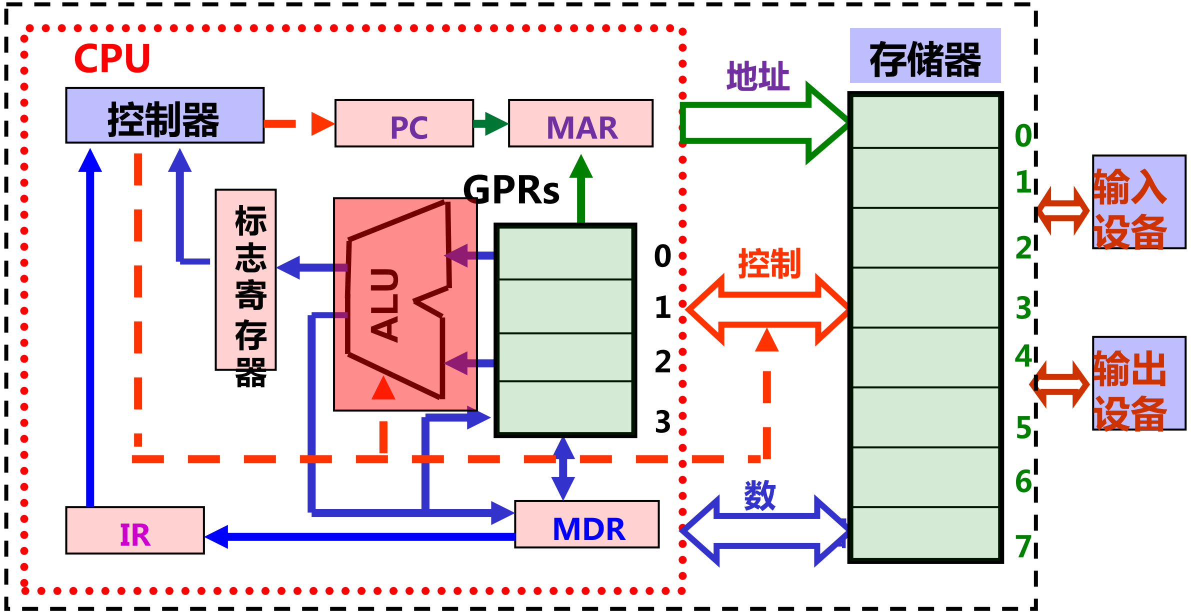
符号标志SF：SF=F_{n-1}

零标志ZF=1，当且仅当F=0；

进位/借位标志CF：CF=$C_{out} \oplus C_{in}$

基本运算部件

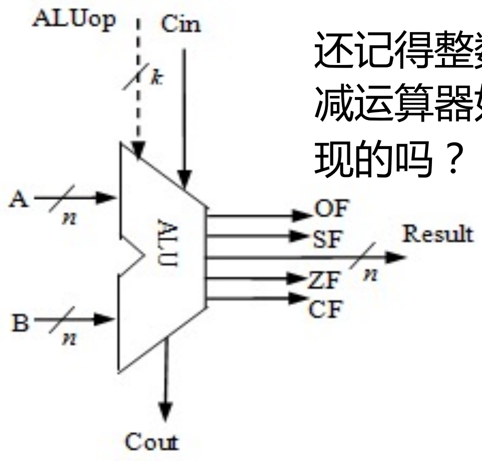
- 算术逻辑部件



CPU：中央处理器；
PC：程序计数器；
MAR：存储器地址寄存器
ALU：算术逻辑部件；
IR：指令寄存器；
MDR：存储器数据寄存器
GPRs：通用寄存器组
（由若干通用寄存器组成）

基本运算部件

- **算术逻辑部件**
- 进行**基本算术运算与逻辑运算**
  - 无符号整数加、减
  - 带符号整数加、减
  - 与、或、非、异或等逻辑运算
- 核心电路是整数加/减运算部件
- 输出除和/差等，还有标志信息
- 有一个操作控制端（ALUop），用来决定ALU所执行的处理功能。ALUop的位数k决定了操作的种类例如，当位数k为3时，ALU最多只有2^3=8种操作。



ALU 符号

还记得整数加/减运算器如何实现的吗？

ALUop	Result	ALUop	Result	ALUop	Result	ALUop	Result
0 0 0	A加B	0 1 0	A与B	1 0 0	A取反	1 1 0	A
0 0 1	A减B	0 1 1	A或B	1 0 1	A⊕B	1 1 1	未用

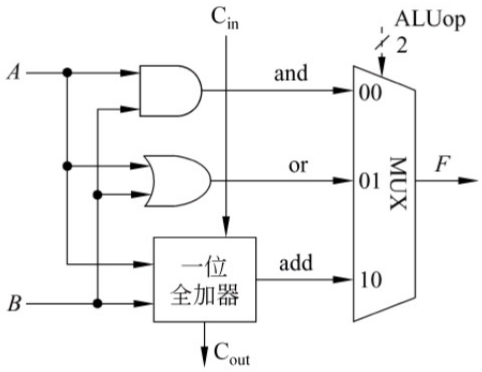


图 3.8　一位 ALU 结构

运算方法和运算部件

- 高级语言和机器指令中的运算
- 基本运算部件
- **定点数运算**
- 整数乘除运算
- 浮点数运算

定点数运算——补码加减运算

- **补码加减运算公式**
  - $[X+Y]_补 = [X]_补 + [Y]_补$（MOD 2^n）
  - $[X-Y]_补 = [X]_补 + [-Y]_补$（MOD 2^n）
- **补码加减运算要点和运算部件**
  - 加、减法运算统一采用加法来处理
  - 符号位(最高有效位MSB)和数值位一起参与运算
  - 直接用ALU实现两个数的加运算（模运算系统）

问题：模是多少？

- 运算结果的高位丢弃，保留低n位，相当于对和数取模2^n
- 实现减法的主要工作在于：求$[-Y]_补$

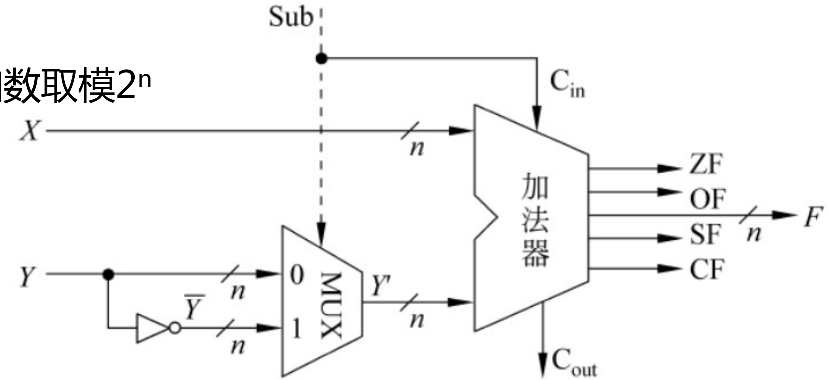
问题：补码加减运算的用途是什么？

用于实现带符号整数的加减运算！

问题：如何求$[-Y]_补$？

$[-B]_补 = \overline{B} + 1$

当控制端Sub为1时，做减法，实现A–B当控制端Sub为0时，做加法，实现A+B

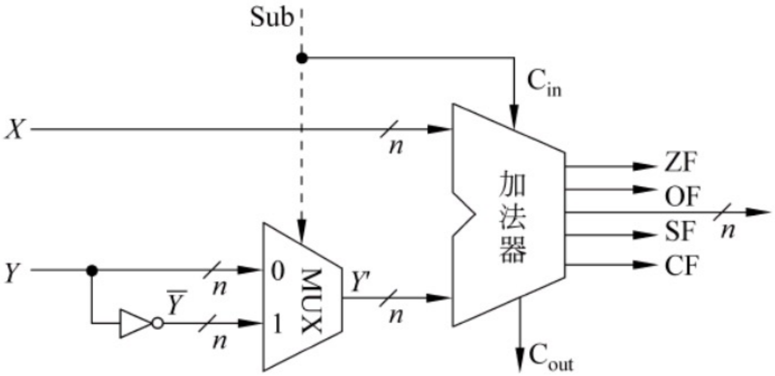


补码加减运算部件

定点数运算——补码加减运算

- 利用带标志加法器，可构造整数加/减运算器，进行以下运算：
  - ➢ 无符号整数加、无符号整数减
  - ➢ 带符号整数加、带符号整数减

当Sub为1时，做减法
当Sub为0时，做加法



- **ZF=1**，表示结果F=0；
- **OF=1**，表示溢出（**仅指带符号运算，**
  $OF=C_n \oplus C_{n-1}$， $OF = X_{n-1}Y_{n-1}\overline{F}_{n-1} + \overline{X}_{n-1}\overline{Y}_{n-1}F_{n-1}$
- **SF=1**，表示结果的符号/F最高位
  （**仅指带符号运算**）；
- **CF=1**，表示进/借位
  （**仅指无符号运算，$CF=Sub \oplus C_{out}$**）

定点数运算——补码加减运算

例1: -7 - 6 = -7 + (- 6) = +3 X

```
 1   0
     1   0   0   1
+    1   0   1   0
------------------
     0   0   1   1
```

-3 - 5 = - 3 + (- 5) = - 8 √

```
 1   1   1   1
     1   1   0   1
+    1   0   1   1
------------------
     1   0   0   0
```

溢出现象：(1) 最高位和次高位的进位不同；(2) 和的符号位和加数的符号位不同

例2: 用8位补码求 107 和 46的“和”，结果错误: 107 + 46 = -103.

```
          11  1 11
107₁₀ = 0110 1011₂
 46₁₀ = 0010 1110₂
------------------
       0 1001 1001
```

溢出时，符号位的进位是真正的符号：+153

问题：若采用**变形补码**则结果怎样？有何好处？

结果的值为01 0011001，**左边第一位为真正的符号，数值部分进到了右边符号位上。**

变形补码时的“溢出”判断条件：结果的两个符号位不同。

定点数运算——原码加减运算（不要求）

- 用于浮点数尾数运算
- 符号位和数值部分分开处理
- 仅对数值部分进行加减运算，符号位起判断和控制作用
- 规则如下：
  - **比较两数符号，对加法实行“同号求和，异号求差”，对减法实行“异号求和，同号求差”**。
  - **求和**：数值位相加，若最高位产生进位，则结果溢出。和的符号取被加数（被减数）的符号。
  - **求差**：被加数（被减数）加上加数（减数）的补码。分二种情况讨论：

    a) 最高数值位产生进位，表明加法结果为正，所得数值位正确。

    b) 最高数值位没有产生进位，表明加法结果为负，得到的是数值位的补码形式，需对结果求补，还原为绝对值形式的数值位。
  - **差的符号位**：a) 情况下，符号位取被加数（被减数）的符号

    b) 情况下，符号位为被加数（被减数）的符号取反

定点数运算——原码加减运算（不要求）

例1：已知 [X]原 = 1.0011，[Y]原 = 1.1010，要求计算[X+Y]原

解：根据原码加减运算规则，知：两数同号，用加法求和，和的符号同被加数符号。

- 和的数值位为：0011 + 1010 = 1101 （ALU中无符号数加）
- 和的符号位为：1
- [X+Y]原 = 1.1101

求和：直接加，有进位则溢出，符号同被

例2 ：已知 [X]原 = 1.0011，[Y]原 = 1.1010，要求计算[X–Y]原

解：根据原码加减运算规则，知：两数同号，用减法求差（补码减法）

- 差的数值位为：0011+(1010)补 = 0011 + 0110 = 1001
- 最高数值位没有产生进位，表明加法结果为负，需对1001求补，还原为绝对值形式的数值位：（1001）补 = 0111
- 差的符号位为[X] 原的符号位取反，即：0
- [X–Y]原 = 0.0111

求差：加补码，不会溢出，符号分情况

定点数运算——无符号数的乘法运算

假定：$[X]_原=x_0.x_1\ldots x_n$，$[Y]_原=y_0.y_1\ldots y_n$ ，求$[x\times Y]_原$

数值部分 $z_1\ldots z_{2n} = (0.x_1\ldots x_n) \times (0.y_1\ldots y_n)$

(小数点位置约定，不区分小数还是整数)

- **手算乘法例子**:

被乘数	1000
乘数	1001
	1000
	0000
	0000
	1000
积	0.1001000

$$X\times Y=\sum_{i=1}^{4}(X\times y_i\times 2^{-i})$$

$X\times y_4\times 2^{-4}$　　n=4

$X\times y_3\times 2^{-3}$

$X\times y_2\times 2^{-2}$

$X\times y_1\times 2^{-1}$

整个运算过程中用到两种操作：加法 + 左移

因而，可用ALU和移位器来实现乘法运算

定点数运算——无符号数的乘法运算

- **手算乘法的特点：**

  ① 每步计算：$X \times y_i$，若$y_i = 0$，则得0；若$y_i = 1$，则得X

  ② 把①求得的各项结果$X \times y_i$ 逐次左移，可表示为$X \times y_i \times 2^{-i}$

  ③ 对②中结果求和，即 $\sum (X \times y_i \times 2^{-i})$，这就是两个无符号数的乘积

- **计算机内部稍作以下改进：**

  ① 每次得$X \times y_i$后，与前面所得结果累加，得到P_i，称之为部分积。因为不等到最后一次求和，减少了保存各次相乘结果$X \times y_i$的开销。

  ② 每次得$X \times y_i$后，不将它左移与前次部分积P_i相加，而将部分积P_i右移后与$X \times y_i$相加。因为加法运算始终对部分积中高n位进行。故用n位加法器可实现二个n位数相乘。

  ③ 对乘数中为“1”的位执行加法和右移，对为“0”的位只执行右移，而不执行加法运算。

定点数运算——无符号数的乘法运算

- 上述思想可写成如下**数学推导过程**：

$$X\times Y = X \times (0.y_1 y_2 \ldots y_n)$$
$$= X\times y_1\times 2^{-1} + X\times y_2\times 2^{-2} + \ldots\ldots + X\times y_{n-1}\times 2^{-(n-1)} + X\times y_n\times 2^{-n}$$
$$= 2^{-n}\times X\times y_n + 2^{-(n-1)}\times X\times y_{n-1} + \ldots\ldots + 2^{-2}\times X\times y_2 + 2^{-1}\times X\times y_1$$
$$=\underbrace{2^{-1}(2^{-1}(2^{-1}\ldots 2^{-1}(2^{-1}}_{n\text{个}2^{-1}}(0 + X\times y_n) + X\times y_{n-1}) + \ldots\ldots + X\times y_2) + X\times y_1)$$

- 上述推导过程**具有明显的递归性质**，因此，无符号数乘法过程可归结为循环计算下列算式的过程：设$P_0 = 0$，每步的乘积为：

$$P_1 = 2^{-1}(P_0 + X\times y_n)$$
$$P_2 = 2^{-1}(P_1 + X\times y_{n-1})$$
$$\ldots\ldots\quad\ldots\ldots$$
$$P_n = 2^{-1}(P_{n-1} + X\times y_1)$$

迭代过程从乘数最低位y_n和P_0=0开始，经n次“判断–加法–右移”循环，直到求出P_n为止

- 其递推公式为：$P_{i+1} = 2^{-1}(P_i + X\times y_{n-i})$ （ $i = 0, 1, 2, 3, \ldots, n-1$ ）
- 最终乘积$P_n = X\times Y$

南京大学 NANJING UNIVERSITY

定点数运算——原码乘法运算

- 用于浮点数尾数乘运算
- 符号与数值分开处理：积符用两个符号异或得到，数值用无符号乘法运算

例：设$[x]_原$=0.1110 ，$[y]_原$=1.1101，计算 $[X\times Y]_原$

解：数值部分用无符号数乘法算法计算：1110×1101= 1011 0110,

符号位：0 ⊕ 1=1，所以： $[X\times Y]_原$=1. 10110110

左侧是**原码一位乘法**：每次只取乘数中的一位进行判断，需n次循环，速度相对较慢。

原码两位乘法的思想：对乘数的每两位取值进行判断，每步求出对应两位的部分积。

- **原码两位乘法的操作的递推公式：**

$00 — P_{i+1}=2^{-2} P_i$

$01 — P_{i+1}=2^{-2} (P_i +X)$

$10 — P_{i+1}=2^{-2} (P_i +2X)$

$11 — P_{i+1}=2^{-2} (P_i +3X)=2^{-2} (P_i +4X-X)$

$=2^{-2} (P_i -X) +X$

3X时，本次-X，下次+X！

采用两位一乘，运算速度提高多少？

运算次数减少一半，速度提高一倍！

y_{i-1} y_i T	操　作	迭代公式
0 0 0	$0\rightarrow T$	$2^{-2}(P_i)$
0 0 1	+X　$0\rightarrow T$	$2^{-2}(P_i+X)$
0 1 0	+X　$0\rightarrow T$	$2^{-2}(P_i+X)$
0 1 1	+2X　$0\rightarrow T$	$2^{-2}(P_i+2X)$
1 0 0	+2X　$0\rightarrow T$	$2^{-2}(P_i+2X)$
1 0 1	–X　$1\rightarrow T$	$2^{-2}(P_i-X)$
1 1 0	–X　$1\rightarrow T$	$2^{-2}(P_i-X)$
1 1 1	$1\rightarrow T$	$2^{-2}(P_i)$

T触发器用来记录下次是否要执行“+X”，其中“–X”运算用“$+[-X]_补$”实现！

定点数运算——原码乘法运算

已知 $[X]_{原}$=0.111001， $[Y]_{原}$= 0.100111，用原码两位乘法计算$[X\times Y]_{原}$

解： 先采用无符号数乘法计算111001× 100111的乘积，原码两位乘法过程如下：

采用补码右移

数据为模8补码形式（三位符号位），为什么？

若采用模4补码，则进行P和Y同时右移2位操作时，按照补码右移规则，得到的P3是负数，显然，两个正数相乘，乘积不可能是负数

$[|X|]_{补}=000\ 111001$，$[-|X|]_{补}=111\ 000111$

P		Y	T	说明
000	000000	100111	0	开始，$P_0=0$，T=0
+111	000111			$y_5y_6T=110$，−X，T=1
111	000111			P 和 Y 同时右移 2 位
111	110001	11 1001	1	得 P_1
+001	110010			$y_3y_4T=011$，+2X，T=0
001	100011			P 和 Y 同时右移 2 位
000	011000	1111 10	0	得 P_2
+001	110010			$y_1y_2T=100$，+2X，T=0
010	001010			P 和 Y 同时右移 2 位
000	100010	101111	0	得 P_3

加上符号位，得 $[X\times Y]_{原}=0.100010101111$

速度快，但代价也大

定点数运算——补码乘法运算

- 用于对什么类型的数据计算？已知什么？求什么？
- 带符号整数！如C语句：int x=-5，y=-4，z=x*y;

问题：已知$[x]_{补}$和$[y]_{补}$，求$[x*y]_{补}$

- 因为$[x*y]_{补} \neq [x]_{补}*[y]_{补}$，故不能直接用无符号整数乘法计算。
- 例如，若x=-5，求x*x=？：$[-5]_{补}$=1011
- **$[x*x]_{补}$：$[25]_{补}$=0001 1001**
- **$[x]_{补}*[x]_{补}$：$[-5]_{补}*[-5]_{补}$=1111 1001**

思路：根据$[y]_{补}$求y，将y的计算公式代入$[x*y]_{补}$

令：$[y]_{补}= y_{n-1}y_{n-2}\cdots\cdots y_1y_0$

则：$y= -y_{n-1}\cdot 2^{n-1}+y_{n-2}\cdot 2^{n-2}+ \cdots\cdots y_1\cdot 2^1+ y_0\cdot 2^0$

因为$[A+B]_{补}= [A]_{补}+[B]_{补}$，只要将$[x*y]_{补}$转换为对若干数的和求补即可

定点数运算——补码乘法运算

布斯（Booth）乘法推导如下：

假定：$[X]_{补}=x_{n-1}x_{n-2}\cdots\cdots x_1x_0$，$[Y]_{补}=y_{n-1}y_{n-2}\cdots\cdots y_1y_0$，求：$[X\times Y]_{补}=?$

基于补码定义：$Y=-y_{n-1}\cdot 2^{n-1}+y_{n-2}\cdot 2^{n-2}+\cdots\cdots y_1\cdot 2^1+y_0\cdot 2^0$

令：$y_{-1}=0$，则：

当n=32时，$Y=-y_{31}\cdot 2^{31}+y_{30}\cdot 2^{30}+\cdots\cdots y_1\cdot 2^1+y_0\cdot 2^0+y_{-1}\cdot 2^0$

$$\downarrow$$

$$\underline{-y_{31}\cdot 2^{31}+(y_{30}\cdot 2^{31}}-y_{30}\cdot 2^{30})+\cdots\cdots+(y_0\cdot 2^1\underline{-y_0\cdot 2^0)+y_{-1}\cdot 2^0}$$

$$\downarrow$$

$$(y_{30}-y_{31})\cdot 2^{31}+(y_{29}-y_{30})\cdot 2^{30}+\cdots\cdots+(y_0-y_1)\cdot 2^1+(y_{-1}-y_0)\cdot 2^0$$

$$2^{-32}\cdot[X\times Y]_{补}=(y_{30}-y_{31})X\cdot 2^{-1}+(y_{29}-y_{30})X\cdot 2^{-2}+\cdots\cdots+(y_0-y_1)X\cdot 2^{-31}+(y_{-1}-y_0)X\cdot 2^{-32}$$

$$=2^{-1}(2^{-1}\ldots(2^{-1}(y_{-1}-y_0)X)+(y_0-y_1)X)+\ldots+(y_{30}-y_{31})X)$$

部分积公式：$P_i=2^{-1}(P_{i-1}+(y_{i-1}-y_i)X)$

符号与数值统一处理

南京大學 NANJING UNIVERSITY

定点数运算——补码乘法运算

布斯（Booth）算法实质：

结束　　运行中　　开始

0 1 1 1 1 0

当前位	右边位	操作	举例
1	0	减被乘数	0001111000
1	1	加0 (不操作)	0001111000
0	1	加被乘数	0001111000
0	0	加0 (不操作)	0001111000

- 在“**1串**”中，第一个1时做减法，最后一个1做加法，其余情况只要移位。
- 最初提出这种想法是因为在Booth的机器上移位操作比加法更快！

同前面算法一样，将乘积寄存器右移一位。（这里是算术右移）

定点数运算——补码乘法运算

布斯（Booth）算法举例：

已知$[X]_补$ = 1 101，$[Y]_补$ = 0 110，计算$[X\times Y]_补$　　$[-X]_补$ = 0011

X=-3，Y=6，X×Y=-18，$[X\times Y]_补$应等于11101110或结果溢出

P	Y	y_{-1}		说明
0000	0110	0		设 $y_{-1}=0$，$[P_0]_补=0$
			→ 1	$y_0\,y_{-1}=00$，P、Y 直接右移一位
0000	0011	0		得$[P_1]_补$
+0011				$y_1\,y_0=10$，$+[-X]_补$
0011			→ 1	P、Y 同时右移一位
0001	1001	1		得$[P_2]_补$
			→ 1	$y_2\,y_1=11$，P、Y 直接右移一位
0000	1100	1		得$[P_3]_补$
+1101				$y_3\,y_2=01$，$+[X]_补$
1101			→ 1	P、Y 同时右移一位
1110	1110	0		得$[P_4]_补$

如何判断结果是否溢出？

高4位是否全为符号位！

验证：当X×Y取8位时，结果 -0010010B=-18；取4位时，结果溢出

南京大学 NANJING UNIVERSITY

定点数运算——补码两位乘法

- **补码两位乘可用布斯算法推导如下**：
  - $[P_{i+1}]_补 = 2^{-1}\,(\,[P_i]_补 + (\,y_{i-1} - y_i\,)\,[X]_补)$
  - $[P_{i+2}]_补 = 2^{-1}\,(\,[P_{i+1}]_补 + (\,y_i - y_{i+1})\,[X]_补)$

$$= 2^{-1}\,(2^{-1}\,(\,[P_i]_补 + (\,y_{i-1} - y_i\,)\,[X]_补) + (\,y_i - y_{i+1})\,[X]_补)$$

$$= 2^{-2}\,(\,[P_i]_补 + (y_{i-1} + y_i - 2y_{i+1})\,[X]_补)$$

y_{i+1}	y_i	y_{i-1}	操　作	迭 代 公 式
0	0	0	0	$2^{-2}[P_i]_补$
0	0	1	$+[X]_补$	$2^{-2}\{[P_i]_补+[X]_补\}$
0	1	0	$+[X]_补$	$2^{-2}\{[P_i]_补+[X]_补\}$
0	1	1	$+2[X]_补$	$2^{-2}\{[P_i]_补+2[X]_补\}$
1	0	0	$+2[-X]_补$	$2^{-2}\{[P_i]_补+2[-X]_补\}$
1	0	1	$+[-X]_补$	$2^{-2}\{[P_i]_补+[-X]_补\}$
1	1	0	$+[-X]_补$	$2^{-2}\{[P_i]_补+[-X]_补\}$
1	1	1	0	$2^{-2}[P_i]_补$

- 开始置附加位y_{-1}为0，乘积寄存器最高位前面添加一位附加符号位0。
- 最终的乘积高位部分在乘积寄存器P中，低位部分在乘数寄存器Y中。
- 因为字长总是8的倍数，所以补码的位数n应该是偶数，因此总循环次数为n/2。

定点数运算——补码两位乘法

- 已知 $[X]_{补}$ = 1 101， $[Y]_{补}$ = 0 110，用补码两位乘法计算$[X\times Y]_{补}$。
- 解：$[-X]_{补}$= 0 011，**用补码二位乘法计算**$[X\times Y]_{补}$的过程如下。

P_n	P	Y	y_{-1}	说明
0	0 0 0 0	0 1 1 0	0	开始，设y_{-1} = 0，$[P_0]_{补}$ = 0
+ 0	0 1 1 0			$y_1y_0y_{-1}$ =100，+2$[-X]_{补}$
0	0 1 1 0		$\rightarrow$ 2	P和Y同时右移二位
0	0 0 0 1	1 0 0 1	1	得$[P_2]_{补}$
+ 1	1 0 1 0			$y_3y_2y_1$ = 011，+2$[X]_{补}$
1	1 0 1 1		$\rightarrow$ 2	P和Y同时右移二位
1	1 1 1 0	1 1 1 0		得$[P_4]_{补}$

因此 [X × Y]补=1110 1110 ，与一位补码乘法（布斯乘法）所得结果相同，但循环次数减少了一半。

验证：-3 ×6=-18 （-10010B）

定点数运算——快速乘法器（不作要求）

- **前面介绍的乘法部件的特点**
  - 通过一个ALU多次做“加/减+右移”来实现
    - 一位乘法：约n次“加+右移”
    - 两位乘法：约n/2次“加+右移”

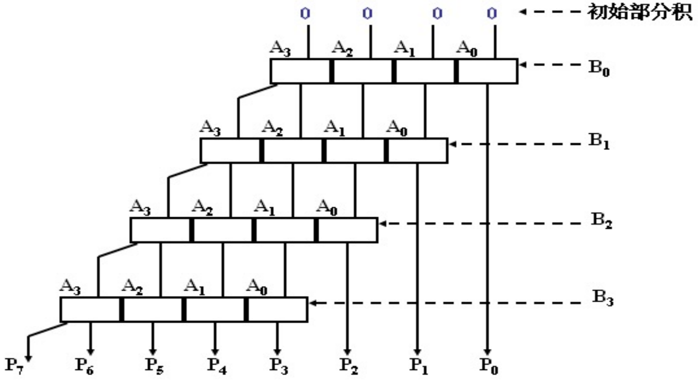
  所需时间随位数增多而加长，由时钟和控制电路控制

- **设计快速乘法部件的必要性**
  - 大约1/3是乘法运算

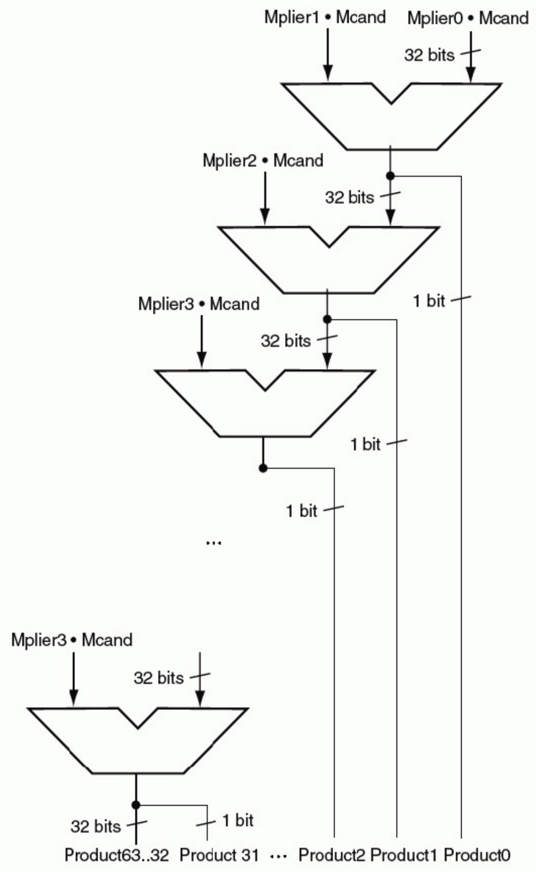
- **快速乘法器的实现**（由特定功能的组合逻辑单元构成）
  - 流水线方式
  - 硬件叠加方式（如：阵列乘法器）

定点数运算——快速乘法器（不作要求）

流水线方式的快速乘法器



组合逻辑电路！
无需控制器控制

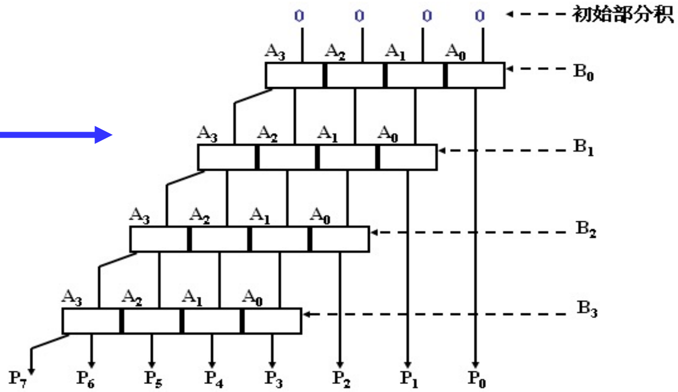


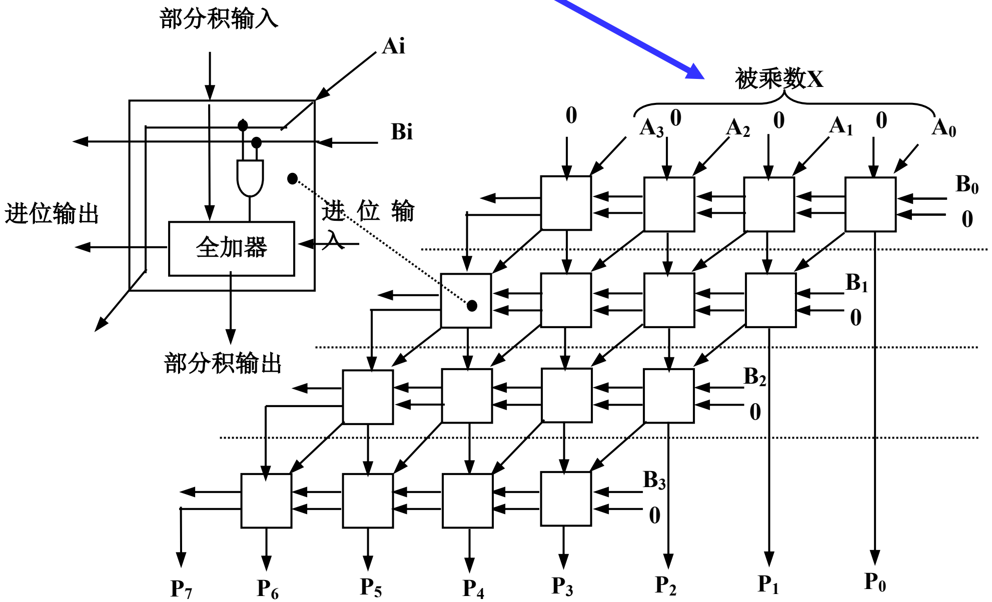
- 为乘数的每位提供一个n位加法器
- 每个加法器的两个输入端分别是：
  - 本次乘数对应的位与被乘数相与的结果（即：0或被乘数）
  - 上次部分积
- 每个加法器的输出分为两部分：
  - 和的最低有效位(LSB)作为本位乘积
  - 进位和高31位的和数组成一个32位数作为本次部分积

定点数运算——快速乘法器（不作要求）

阵列乘法器的实现

- 手算乘法过程
- 阵列乘法器





速度仅取决于逻辑门和加法器的传输延迟

无符号阵列乘法器

增加符号处理电路、乘前及乘后求补电路，即可实现带符号数乘法器。

定点数运算——除法运算

手算除法：

```
                 1001      商
除数    1000 ) 1001010     被除数
              -1000
               ------
                  10
                  101
               ------
                  1010
                 -1000
                 ------
                    10     余数
```

中间余数

- 手算除法的基本要点
  - ① **被除数与除数相减**，够减则上商为1；不够减则上商为0。
  - ② 每次得到的**差为中间余数**，将除数右移后与上次的中间余数比较。用中间余数减除数，够减则上商为1；不够减则上商为0。
  - ③ 重复执行第②步，直到求得的**商的位数足够为止**。

定点数运算——定点除法运算

- **除前预处理**

  ①若被除数=0且除数≠0，或定点整数除法时|被除数|<|除数|，则商为0，不再继续

  ②若被除数≠0、除数=0，则发生“除数为0”异常（浮点数时为∞）

  （若浮点除法被除数和除数都为0，则有些机器产生一个不发信号的NaN，即“quiet NaN”

  ③只有当被除数和除数都≠ 0，且商≠ 0时，才进一步进行除法运算。

- **计算机内部无符号数除法运算**
  - 与手算一样，通过被除数（中间余数）减除数来得到每一位商
    - 够减上商1；不够减上商0（从msb→lsb得到各位商）
  - 基本操作为减（加）法和移位，故可与乘法合用同一套硬件

两个n位数相除的情况：

(1)**定点正整数（即无符号数 ）相除**：在被除数的高位添n个0

(2)**定点正小数（即原码小数）相除**：在被除数的低位添加n个0

这样，就将所有情况都统一为：一个2n位数除以一个n位数

南京大學 NANJING UNIVERSITY

定点数运算——定点除法运算

问题：第一次试商为1，说明什么？ **商有n+1位数，因而溢出！**

若是2n位除以n位的无符号整数运算，则说明将会得到多于n位的商，因而结果“溢出”（即：无法用n位表示商）。

例：1111 1111/1111 = 1 0001

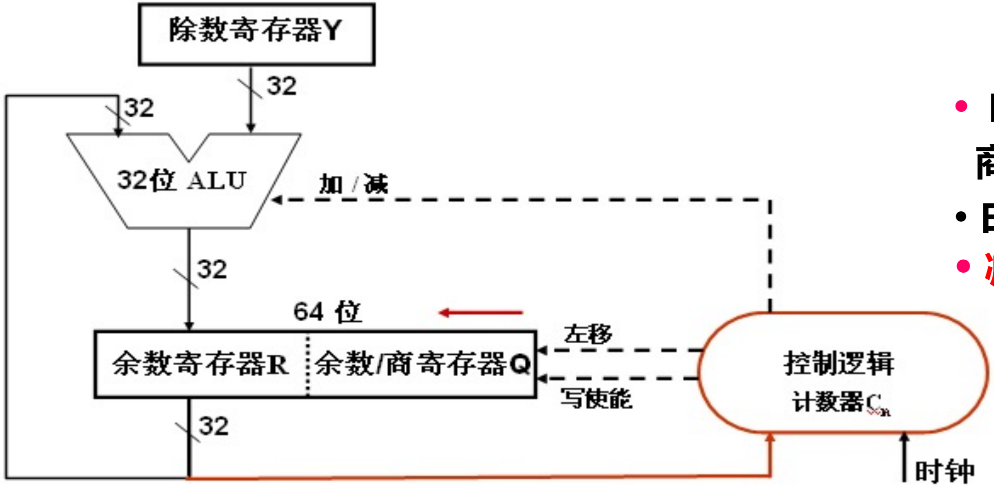
若是两个n位数相除，则第一位商为0，且肯定不会溢出，为什么？

最大商为:0000 1111/0001=1111

若是浮点数中尾数原码小数运算，第一次试商为1，则说明尾数部分有“溢出”，可通过浮点数的“右规”消除“溢出”。所以，在浮点数运算器中，第一次得到的商“1”要保留。

例：0.11110000/0.1000=+1.1110

定点数运算——定点除法运算



- R和Q同步“左移”，Q空出位上“商”，商的各位逐次左移到Q中。
- 由控制逻辑根据加减结果决定商为0还是1
- 减----试商，加----恢复余数。

- **除数寄存器Y**：存放除数。
- **余数寄存器R**：初始时高位部分为高32位被除数；结束时是余数。
- **余数/商寄存器Q**：初始时为低32位被除数；结束时是32位商。
- **循环次数计数器C_n**：存放循环次数。初值是32，每循环一次，Cn减1，当Cn=0时，除法运算结束。
- **ALU**：除法核心部件。在控制逻辑控制下，对于寄存器R和Y的内容进行“加/减”运算，在“写使能”控制下运算结果被送回寄存器R。

定点数运算——定点除法运算

验证：7 / 2 = 3 余 1　　–D = 1110

R：被除数（中间余数）；　D：除数

	D: 0010		R: 0000 0111
Shl R	D: 0010		R: 0000 1110
R = R–D	D: 0010	+1110	R: 1110 1110
+D, sl R, 0	D: 0010	+0010	R: 0001 1100
R = R–D	D: 0010	+1110	R: 1111 1100
+D, sl R, 0	D: 0010	+0010	R: 0011 1000
R = R–D	D: 0010	+1110	R: 0001 1000
sl R, 1	D: 0010		R: 0011 0001
R = R–D	D: 0010	+1110	R: 0001 0001
sl R, 1	D: 0010		R: 0010 0011
Shr R(rh)	D: 0010		R: $\boxed{0001}$ 0011

这里是**两个n位无符号数相除**，肯定不会溢出，故余数先左移而省略判断溢出过程。

从例子可看出：

每次上商为0时，需做加法以“恢复余数”。所以，称为**“恢复余数法”**.

开始余数先左移了一位，故最后余数需向右移一位

定点数运算——定点除法运算

不恢复余数除法（加减交替法）：在下一步运算时把当前多减的除数补回来

根据恢复余数法(设B为除数，R_i为第i次中间余数)，有：

- 若$R_i<0$,则商上“0”，并做加法恢复余数，即：

  $R_{i+1}=2(R_i+2^n|B|)-2^n|B|=2R_i+2^n|B|$ （“负，0，加”）

- 若$R_i>=0$,则商上“1”，不需恢复余数，即：

  $R_{i+1}=2R_i-2^n|B|$ （“正，1，减”）

省去了恢复余数的过程

注意：最后一次上商为“0”的话，需要“纠余”处理，即把试商时被减掉的除数加回去，恢复真正的余数。

不恢复余数法也称为加减交替法

定点数运算——带符号数除法

- **原码除法**
  - **商符和商值分开处理**
    - 商的数值部分由无符号数除法求得
    - 商符由被除数和除数的符号确定：同号为0，异号为1
  - **余数的符号同被除数的符号**
- **补码除法**
  - 方法1：**同原码除法一样**，先转换为正数，先用无符号数除法，然后修正商和余数。
  - 方法2：**直接用补码除法**，符号和数值一起进行运算，商符直接在运算中产生。

若是两个n位补码整数除法运算，则被除数进行符号扩展。

若被除数为2n位，除数为n位，则被除数无需扩展。

定点数运算——原码除法举例

已知 $[X]_原 = 0.1011$，$[Y]_原 = 1.1101$

用**恢复余数法**计算$[X/Y]_原$？

解：商的符号位：$0 \oplus 1 = 1$

减法操作用补码加法实现，是否够减通过中间余数的符号来判断，所以中间余数要加一位符号位。

$[|X|]_补 = 0.1011$

$[|Y|]_补 = 0.1101$

$[-|Y|]_补 = 1.0011$

小数在低位扩展0

思考：若实现无符号数相除，即1011除以1101，则有何不同？结果是什么？

被除数高位补0，1011除以1101，结果等于0

余数寄存器 R	余数/商寄存器 Q	说明
01011	0000□	开始 $R_0=X$
+10011		$R_1=X-Y$
11110	00000	$R_1<0$，则 $q_4=0$
+01101		恢复余数：$R_1=R_1+Y$
01011		得 R_1
10110	0000□	$2R_1$（R 和 Q 同时左移，空出一位商）
+10011		$R_2=2R_1-Y$
01001	00001	$R_2>0$，则 $q_3=1$
10010	0001□	$2R_2$（R 和 Q 同时左移，空出一位商）
+10011		$R_3=2R_2-Y$
00101	00011	$R_3>0$，则 $q_2=1$
01010	0011□	$2R_3$（R 和 Q 同时左移，空出一位商）
+10011		$R_3=2R_2-Y$
11101	00110	$R_4<0$，则 $q_1=0$
+01101		恢复余数：$R_4=R_4+Y$
01010	00110	得 R_4
10100	0110□	$2R_4$（R 和 Q 同时左移，空出一位商）
+10011		$R_5=2R_4-Y$
00111	01101	$R_5>0$，则 $q_0=1$

商的最高位为0，说明没有溢出，商的数值部分为：1101。

所以，$[X/Y]_原=1.1101$（最高位为符号位），余数为 0.0111×2^{-4}。

用于判断是否溢出

若求$[Y/X]_原$结果如何？

定点数运算——原码除法举例

已知 $[X]_原$ = 0.1011，$[Y]_原$ = 1.1101

用加减交替法计算$[X/Y]_原$

解：$[|X|]_补$ = 0.1011

$[|Y|]_补$ = 0.1101

$[-|Y|]_补$ = 1.0011

加减交替法的要点：

负、0、加

正、1、减

得到的结果与恢复余数法一样！

用被除数（中间余数）减除数试商时，怎样确定是否“够减”？

中间余数的符号！（正数-正数）

余数寄存器R	余数/商寄存器Q	说明
01011	0000□	开始 $R_0=X$
+10011		$R_1=X-Y$
11110	00000	$R_1<0$，则 $q_4=0$，没有溢出
11100	0000□	$2R_1$（R和Q同时左移，空出一位商）
+01101		$R_2=2R_1+Y$
01001	00001	$R_2>0$，则 $q_3=1$
10010	0001□	$2R_2$ （R和Q同时左移，空出一位商）
+10011		$R_3=2R_2-Y$
00101	00011	$R_3>0$，则 $q_2=1$
01010	0011□	$2R_3$ （R和Q同时左移，空出一位商）
+10011		$R_3=2R_2-Y$
11101	00110	$R_4<0$，则 $q_1=0$
11010	0110□	$2R_4$ （R和Q同时左移，空出一位商）
+01101		$R_5=2R_4+Y$
00111	01101	$R_5>0$，则 $q_0=1$

补码除法能否这样来判断呢？

不能，因为符号可能不同！

南京大学 NANJING UNIVERSITY

定点数运算——补码除法运算

- **补码除法判断是否“够减”的规则**

（1）当被除数（或中间余数）与除数**同号**时，做**减法**，若新余数的符号与被除数符号**一致**（**正(负)-正(负)=正(负)**）**表示够减**，否则为不够减

（2）当被除数（或中间余数）与除数**异号**时，做**加法**，若新余数的符号与被除数符号**一致** **表示够减**（**正(负)+负(正)=正(负)**），否则为不够减

总结：余数变号不够减，不变号够减

上述判断规则归纳如下：

中间余数R的符号	除数Y的符号	同号：新中间余数=R–Y（同号为正商）		异号：新中间余数=R+Y（异号为负商）	
		0	1	0	1
0	0	够减	不够减		
0	1			够减	不够减
1	0			不够减	够减
1	1	不够减	够减		

定点数运算——补码恢复余数法

- **两个n位带符号整数相除算法要点**：

(1) 操作数的预置：

除数装入除数寄存器Y，**被除数经符号扩展**后装入余数寄存器R和余数/商寄存器Q

(2) R和Q同步串行左移一位。

(3) 若R与Y同号，则R = R–Y；否则R = R+Y，并按以下规则确定商值q_0 ：

① 若中间余数R=0或R操作前后符号未变，表示够减，则q_0置 1 ，转下一步；

② 若操作前后R的符号已变，表示不够减，则q_0置0，恢复R值后转下一步；

(4) 重复第(2)和第(3)步，直到取得n位商为止。

(5) 若被除数与除数同号，则Q中就是真正的商；否则，将Q求补后是真正的商。

（即：若商应为负数时，则需要“各位取反，末位加1”来得到真正的商）

(6) 余数在R中。

问题：如何恢复余数？通过“做加法”来恢复吗？

无符号数（或原码）除法通过“做加法”恢复余数，但补码不是!

补码：若原来是 R = R–Y，则执行R = R+Y 来恢复余数；

若原来是 R = R+Y，则执行R = R–Y 来恢复余数。

定点数运算——补码恢复余数法

举例：7/3=?

被除数： 0000 0111 除数 0011

	A	Q	M=0011
	0000	0111	
←	0000	1110	
+	1101		减（同号）
	1101	1110	变号
+	0011		恢复(加)商0
	0000	1110	
←	0001	1100	
+	1101		减
	1110	1100	变号
+	0011		恢复(加)商0
	0001	1100	
←	0011	1000	
+	1101		减
	0000	1000	符同，商1
←	0001	0001	
+	1101		减
	1110	0001	变号
+	0011		恢复(加)商0
	0001	0010	

余:0001/商:0010 验证：7/3 = 2，余数为1

(-7)/3=?

被除数： 1111 1001 除数 0011

	A	Q	M=0011
	1111	1001	
←	1111	0010	
+	0011		加（异号）
	0010	0010	变号
+	1101		恢复(减)商0
	1111	0010	
←	1110	0100	
+	0011		加
	0001	0100	变号
+	1101		恢复(减)商0
	1110	0100	
←	1100	1000	
+	0011		加
	1111	1000	符同，商1
←	1111	0001	
+	0011		加
	0010	0001	变号
+	1101		恢复(减)商0
	1111	0010	

商应为负数，需求补：0010→1110

余:1111/商:1110

验证：-7/3 =- 2，余数为-1

定点数运算——补码不恢复余数法

- **算法要点：**

(1) 操作数的预置：

除数装入除数寄存器Y，被除数经符号扩展后装入余数寄存器R和余数/商寄存器Q。

(2) 根据以下规则求第一位商q_n：

若被除数X与Y同号，则R1=X–Y；否则R1 =X+Y，并按以下规则确定商值q_n：

① 若新的中间余数R1与Y同号，则q_n置1，转下一步；

② 若新的中间余数R1与Y异号，则q_n置0，转下一步；

q_n用来判断是否溢出，而不是真正的商。以下情况下会发生溢出：

若X与Y同号且上商q_n=1，或者，若X与Y异号且上商q_n = 0。

(3) 对于i =1到n ，按以下规则求出n位商：

① 若Ri与Y同号，则q_{n-i}置1，Ri+1 = 2Ri –[Y]补，i = i +1；

② 若Ri与Y异号，则q_{n-i}置0，Ri+1 =2Ri+[Y]补，i = i +1；

(4) 商的修正：最后一次Q寄存器左移一位，将最高位q_n移出，最低位置上商q_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。

商已经是“反码”

(5) 余数的修正：若余数符号同被除数符号，则不需修正，余数在R中；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。

判断是否同号与恢复余数法不同，不是新老余数之间！而是余数和除数之间！

补码不恢复余数法也有一个六字口诀“同、1、减；异、0、加”。其运算过程也呈加/减交替方式，因此也称为“加减交替法”。

定点数运算——补码不恢复余数法

举例：-9/2

将X=-9和Y=2分别表示成5位补码形式为：

[X]补 = 1 0111

[Y]补 = 0 0010

被除数符号扩展为：

[X]补=11111 10111

[–Y] 补 = 1 1110

同、1、减

异、0、加

X/Y= – 0100B = – 4

余数为 –0001B = –1

将各数代入公式:

“除数×商+余数= 被除数”进行验证，得

2×(–4) +(–1) = –9

余数寄存器 R	余数/商寄存器 Q	说明
11111	10111	开始 R_0=[X]
+00010		R_1=[X]+[Y]
00001	10111	R_1 与[Y]同号，则 q_5=1
00011	01111	$2R_1$（R 和 Q 同时左移，空出一位上商 1）
+11110		R_2=$2R_1$+[–Y]
00001	01111	R_2 与[Y]同号，则 q_4=1，
00010	11111	$2R_2$（R 和 Q 同时左移，空出一位上商 1）
+11110		R_3=$2R_2$+[–Y]
00000	11111	R_3 与[Y]同号，则 q_3=1
00001	11111	$2R_3$（R 和 Q 同时左移，空出一位上商 1）
+11110		R_4=$2R_3$+[–Y]
11111	11111	R_4 与[Y]异号，则 q_2=0
11111	11110	$2R_4$ （R 和 Q 同时左移，空出一位上商 0）
+00010		R_5=$2R_4$+[Y]
00001	11110	R_5 与[Y]同号，则 q_1=1，
00011	11101	$2R_5$ （R 和 Q 同时左移，空出一位上商 1）
+11110		R_6=$2R_5$+[–Y]
00001	11011	R_6 与[Y]同号，则 q_0=1，Q 左移，空出一位上商 1
+11110	+ 1	商为负数，末位加 1；减除数以修正余数
11111	11100	

所以，[X/Y]补=11100。余数为 11111。

运算方法和运算部件

- 高级语言和机器指令中的运算
- 基本运算部件
- 定点数运算
- **整数乘除运算**
- 浮点数运算

整数乘除运算——整数的乘运算

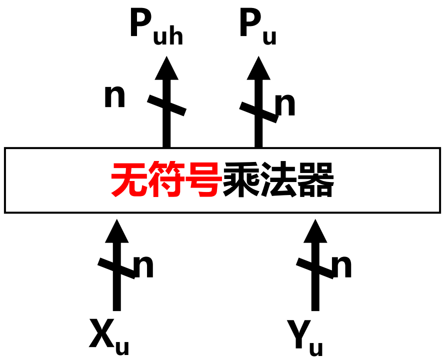
- 假定**两个n位无符号**整数x_u和y_u对应的机器数为X_u和Y_u，$p_u=x_u\times y_u$，p_u为**n位无符号整数**且对应的机器数为P_u；
- **两个n位带符号**整数x_s和y_s对应的机器数为X_s和Y_s，$p_s=x_s\times y_s$，p_s为**n位带符号整数**且对应的机器数为P_s。
- 若$X_u=X_s$且$Y_u=Y_s$，则$P_u=P_s$。

✓ **可用无符号乘来实现带符号乘，但高n位无法得到，故不能判断溢出。**

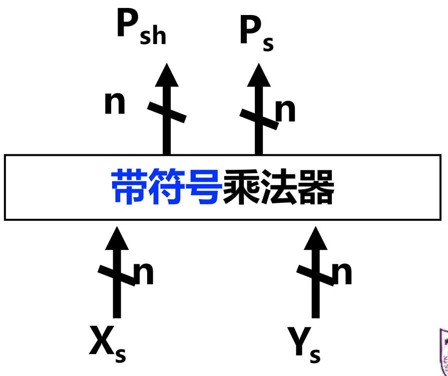
✓ **无符号：若$P_{uh}=0$，则不溢出**

✓ **带符号：若P_{sh}每位都等于P_s的最高位，则不溢出**

$p_u=x_u*y_u;$



$P_{uh} \neq P_{sh}$



$p_s=x_s*y_s;$

整数乘除运算——整数的乘运算

- **X*Y的高n位可以用来判断溢出**，规则如下：
  - **无符号**：若高n位全0，则不溢出，否则溢出
  - **带符号**：若高n位全0或全1且等于低n位的最高位，则不溢出。

运算	x	X	y	Y	x×y	X×Y	p	P	溢出否
无符号乘	6	0110	10	1010	60	0011 1100	12	1100	溢出
带符号乘	6	0110	–6	1010	–36	1101 1100	–4	1100	溢出
无符号乘	8	1000	2	0010	16	0001 0000	0	0000	溢出
带符号乘	–8	1000	2	0010	–16	1111 0000	0	0000	溢出
无符号乘	13	1101	14	1110	182	1011 0110	6	0110	溢出
带符号乘	–3	1101	–2	1110	6	0000 0110	6	0110	不溢出
无符号乘	2	0010	12	1100	24	0001 1000	8	1000	溢出
带符号乘	2	0010	–4	1100	–8	1111 1000	–8	1000	不溢出

整数乘除运算——整数的乘运算

在计算机内部，一定有$x^2 \geq 0$吗?

若x是**带符号整数**，则不一定！

如x是浮点数，则一定！

例如，当 n=4 时, $5^2=-7<0$！

```
int mul(int x, int y)
{
        int z=x*y;
        return z;
}
```

若x、y和z都改成unsigned类型，则判断方式为

乘积的高n位为全0，则不溢出

```
         0101
    ×    0101
   ------------
        0101
   +   0101
   ------------
     00011001
```

结果溢出

只取低4位，值为-111B=-7

- **高级语言程序如何判断z是正确值？**

当 !x || z/x==y 为真时

- **编译器如何判断？**

当 $-2^{n-1} \leq x*y < 2^{n-1}$（不溢出）时

即：乘积的高n位为全0或全1，并等于低n位的最高位！

即：乘积的高n+1位为全0或全1

整数乘除运算——常量的乘除运算

变量与常数之间的除运算

- 不能整除时，采用**朝零舍入**，即**截断**方式
  - **无符号数、带符号正整数**：移出的低位直接丢弃
  - **带符号负整数**：**加偏移量(2^k-1)**，然后再右移k位 ，低位截断（这里K是右移位数）

举例：

无符号数　14/4=3：0000 1110>>2=0000 0011

带符号负整数　-14/4=-3

若直接截断，则 1111 0010 >>2=1111 1100=-4≠-3

应先纠偏，再右移: k=2, 故(-14+2^2-1)/4=-3

即： 1111 0010+0000 0011=1111 0101

1111 0101>>2=1111 1101=-3

-9/2: 10111+00001=11000

11000>>1=11100=-4

整数乘除运算——常量的乘除运算

- 假设*x*为一个int型变量，请给出一个用来计算*x*/32的值的函数div32。要求不能使用除法、乘法、模运算、比较运算、循环语句和条件语句，**可以使用右移、加法以及任何按位运算。**

解: **若x为正数，则将x右移k位得到商；若x为负数，则x需要加一个偏移量(2^k-1)后再右移k位得到商。**因为32=2^5，所以 k=5。

即结果为: **(x>=0 ? x : (x+31))>>5**

但题目要求不能用比较和条件语句，因此要找一个计算偏移量b的方式

这里，x为正时b=0，x为负时b=31. 因此，可以从x的符号得到b

x>>31 得到的是32位符号，取出最低5位，就是偏移量b。

```
int div32(int x)
{  /* 根据x的符号得到偏移量b */
    int b=(x>>31) & 0x1F;
    return (x+b)>>5;
}
```

运算方法和运算部件

- 高级语言和机器指令中的运算
- 基本运算部件
- 定点数运算
- 整数乘除运算
- **浮点数运算**

浮点数运算——浮点数加减运算

设两个规格化浮点数分别为 $A=M_a \cdot 2^{Ea}$　$B=M_b \cdot 2^{Eb}$,则：

$A \pm B = (M_a \pm M_b \cdot 2^{-(Ea-Eb)}) \cdot 2^{Ea}$　（假设Ea>=Eb）

$A*B = (M_a * M_b) \cdot 2^{Ea+Eb}$

$A/B = (M_a / M_b) \cdot 2^{Ea-Eb}$

上述运算结果可能出现以下几种情况：

阶码上溢：一个正指数超过了最大允许值 =〉+∞/-∞/溢出

阶码下溢：一个负指数比最小允许值还小 =〉+0/-0

尾数溢出：最高有效位有进位 =〉右规

非规格化尾数：数值部分高位为0 =〉左规

右规或对阶时，右段有效位丢失 =〉尾数舍入

IEEE建议实现时为每种异常情况提供一个自陷允许位。若某异常对应的位为1，则发生相应异常时，就调用一个特定的异常处理程序执行。

浮点数运算——浮点数加减运算

① 无效运算（无意义）

- 运算时有一个数是非有限数，如：加 / 减∞、0 x ∞、 ∞/∞等
- 结果无效，如：源操作数是NaN、0/0、x REM 0、 ∞ REM y 等

② 除以0（即：无穷大）

③ 数太大（阶码上溢）：对于单精度浮点数，阶码 E >1111 1110 （阶大于127）

④ 数太小（阶码下溢）：对于单精度浮点数，**阶码 E < 0000 0001（阶小于-126-23 ）**

⑤ 结果不精确（舍入时引起），例如1/3，1/10等不能精确表示成浮点数

上述情况硬件可以捕捉到，因此这些异常可设定让硬件处理，也可设定让软件处理。让硬件处理时，称为硬件陷阱。

注：硬件陷阱：事先设定好是否要进行硬件处理（即挖一个陷阱），当出现相应异常时，就由硬件自动进行相应的异常处理（掉入陷阱）。

浮点数运算——浮点数加减运算

- **十进制科学计数法的加法例子**

  $1.123 \times 10^5 + 2.560 \times 10^2$

  其计算过程为：

  $1.123 \times 10^5 + 2.560 \times 10^2 = 1.123 \times 10^5 + 0.002560 \times 10^5$

  $= (1.123 + 0.00256) \times 10^5 = 1.12\boxed{556} \times 10^5$

  $= 1.126 \times 10^5$

  进行尾数加减运算前，必须“对阶”！最后还要考虑舍入

  计算机内部的二进制运算也一样！

- **“对阶”操作**：目的是使两数阶码相等
  - 小阶向大阶看齐，阶小的那个数的尾数右移，右移位数等于两个阶码差的绝对值
  - **IEEE 754尾数右移时，要将隐含的“1”移到小数部分**，高位补0，移出的低位保留到特定的“附加位”上

浮点数运算——浮点数加减运算

问题：如何对阶？

通过计算$[\Delta E]_{补}$来判断两数的阶差：

$[\Delta E]_{补} = [Ex-Ey]_{补} = [Ex]_{移} + [-[Ey]_{移}]_{补} \pmod{2^n}$

问题：在ΔE为何值时无法根据$[\Delta E]_{补}$来判断阶差？　**溢出时！**

例：4位移码，Ex=7，Ey=-7，则$[\Delta E]_{补}$=1111+1111=1110，ΔE<0,错

问题：对IEEE754 SP格式来说，|ΔE|大于多少时，结果就等于阶大的那个数（即小数被大数吃掉）？　**24!**

1.xx…x → 0.00…01xx…x(右移24位后，尾数变为0)

问题：IEEE754 SP格式的偏置常数是127，这会不会影响阶码运算电路的复杂度？

对计算$[Ex-Ey]_{补}$（mod 2^n）没有影响

$[\Delta E]_{补} = 256+Ex-Ey = 256+127+Ex-(127+Ey)$

$= 256 + [Ex]_{移} - [Ey]_{移} = [Ex]_{移} + [-[Ey]_{移}]_{补} \pmod{256}$

但$[Ex+Ey]_{移}$和$[Ex-Ey]_{移}$的计算会变复杂！浮点乘除运算涉及之。

浮点数运算——浮点数加减运算

（假定：Xm、Ym分别是X和Y的尾数， Xe和Ye 分别是X和Y的阶码 ）

(1) **求阶差**：$\Delta e = Ye - Xe$ (若Ye > Xe，则结果的阶码为Ye)

(2) **对阶**：将Xm右移Δe位，尾数变为 $Xm*2^{Xe-Ye}$（保留右移部分附加位）

(3) **尾数加减**： $Xm*2^{Xe-Ye} \pm Ym$

(4) **规格化**：

0.00...0001x2^-126

当尾数高位为0，则需左规：尾数左移一次，阶码减1，直到MSB为1或阶码为00000000（-126，非规格化数）

每次阶码减1后要判断阶码是否下溢（比最小可表示的阶码还要小）

当尾数最高位有进位，需右规：尾数右移一次，阶码加1，直到MSB为1

每次阶码加1后要判断阶码是否上溢（比最大可表示的阶码还要大）

阶码溢出异常处理：阶码上溢，则结果溢出；阶码下溢到无法用非规格化数表示，则结果为0

(5) 如果尾数比规定位数长（有附加位），则需考虑舍入（有多种舍入方式）

(6) 若运算结果尾数是0，则需要将阶码也置0。为什么？

尾数为0说明结果应该为0（阶码和尾数为全0）。

浮点数运算——浮点数加减运算

例：用二进制浮点数形式计算 0.5 +(– 0.4375) = ?

0.4375=0.25+0.125+0.0625=0.0111B

解：$0.5 = 1.000 \times 2^{-1}$， $- 0.4375 = -1.110 \times 2^{-2}$

对　阶: $-1.110 \times 2^{-2} \rightarrow -0.111 \times 2^{-1}$

加　减: $1.000 \times 2^{-1} + (-0.111 \times 2^{-1}) = 0.001 \times 2^{-1}$

左　规: $0.001 \times 2^{-1} \rightarrow 1.000 \times 2^{-4}$

判溢出: 无

结果为： $1.000 \times 2^{-4} = 0.0001000 = 1/16 = 0.0625$

问题：为何IEEE 754 加减运算右规时最多只需一次？

因为即使是两个最大的尾数相加，得到的和的尾数也不会达到4，故尾数的整数部分最多有两位，保留一个隐含的“1”后，最多只有一位被右移到小数部分。

浮点数运算——浮点数加减运算

在计算机内部执行上述运算时，必须解决哪些问题？

(1) 如何表示？　**用IEEE 754标准！**

(2) 如何判断阶码的大小？求$[\Delta E]_{补}$=？

(3) 对阶后**尾数的隐含位**如何处理？　**右移到数值部分，高位补0，保留移出低位部分**

(4) 如何进行**尾数加减**？　**隐藏位还原后，按原码进行加减运算，附加位一起运算**

(5) 何时需要规格化，如何**规格化**？

±1x .xx......x 形式时，则右规: 尾数右移1位, 阶码加1

± 0.0...01x...x 形式时，则左规: 尾数左移k位, 阶码减k

(6) 如何**舍入**？

最终须把附加位去掉，此时需考虑舍入（IEEE 754有四种舍入方式）

(7) 如何判断**溢出**？

若最终阶码为全1，则上溢；若尾数为全0，则下溢

浮点数运算——浮点数加减运算

已知x=0.5, y=-0.4375, 求x+y=? (用IEEE 754标准单精度格式计算)

解:　$x=0.5=1/2=(0.100...0)_2=(1.00...0)_2 x 2^{-1}$

$y=-0.4325=(-0.01110...0)_2=(-1.110..0)_2 x 2^{-2}$

[x]浮=0 01111110,00...0　[y]浮=1 01111101,110...0

对阶: [ΔE]补=0111 1110 + 1000 0011=0000 0001，ΔE=1

故对y进行对阶：[y]浮=1 0111 1110 1110...0(高位补隐藏位)

尾数相加：01.0000...0 + (10.1110...0) = 00.00100...0

(原码加法，最左边一位为符号位，符号位分开处理)

左规：$+(0.00100...0)_2 x 2^{-1}=+(1.00...0)_2 x 2^{-4}$

(阶码减3，实际上是加了三次11111111)

[x+y]浮=0 0111 1011 00...0

$x+y=(1.0)_2 x 2^{-4}=1/16=0.0625$

问题：尾数加法器最多需要多少位？

1+1+23+3=28位

浮点数运算——浮点运算的精度和舍入

在浮点数运算中，保留多少附加位才能保证运算精度？ **无法给出准确答案！**

保留附加位可以得到比不保留附加位更高的精度。

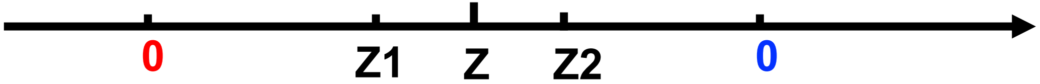
- IEEE754规定: 中间结果须在右边至少加2个附加位（guard & round）
- **Guard bit(保护位/警戒位)**：在浮点数尾数右边的位
- **Rounding bit(舍入位)**：在保护位右边的位
- **Sticky(粘位)**: 舍入位右边任何非0数字，粘位置1，否则置0

附加位的作用：用以保护对阶时右移的位或运算的中间结果。

附加位的处理： ①左规时被移到尾数中; ② 作为舍入的依据。

浮点数运算——浮点运算的精度和舍入

IEEE 754的舍入方式



（Z1和Z2分别是结果Z的最近的可表示的左、右两个数）

(1) 就近舍入： 舍入为最近可表示的数

非中间值：0舍1入；

中间值：强迫结果为偶数

例如：附加位为
01：舍
11：入
10：(强迫结果为偶数)

例：1.110111 → 1.1110;　1.110101 → 1.1101;
1.110110 → 1.1110;　1.111110 → 10.0000;

(2) 朝+∞方向舍入: 舍入为Z2(正向舍入)

(3) 朝-∞方向舍入: 舍入为Z1(负向舍入)

(4) 朝0方向舍入: 截去。正数：取Z1; 负数：取Z2

浮点数运算——浮点运算的精度和舍入

以下情况下，可能会导致阶码溢出

- **左规（阶码 - 1）时**
  - 左规（- 1）时：先判断阶码是否为全0，若是，则直接置阶码下溢；否则，阶码减1后判断阶码是否为全0，若是，则阶码下溢。
- **右规（阶码 +1）时**
  - 右规（+ 1）时，先判断阶码是否为全1，若是，则直接置阶码上溢；否则，阶码加1后判断阶码是否为全1，若是，则阶码上溢。

问题：机器内部如何减1？ $+[-1]_{补} = +\ 11\ldots1$

浮点数运算——浮点运算的精度和舍入

以下情况下，可能会导致阶码溢出（续）

- **乘法运算求阶码的和时**
  - 若Ex和Ey最高位皆1，而Eb最高位是0或Eb为全1，则阶码上溢
  - 若Ex和Ey最高位皆0，而Eb最高位是1或Eb为全0，则阶码下溢
- **除法运算求阶码的差时**
  - 若Ex的最高位是1，Ey的最高位是0，Eb的最高位是0或Eb为全1，则阶码上溢。
  - 若Ex的最高位是0，Ey的最高位是1，Eb的最高位是1或Eb为全0，则阶码下溢。

例：若Eb = 0000 0001，则左规一次后，结果的阶码 Eb = ?

解：Eb=Eb+[-1]$_{补}$=0000 0001 + 1111 1111 = 0000 0000　阶码下溢！

例：若Ex=1111 1110，Ey=1000 0000，则乘法运算时结果的阶码 Eb=?

解：Eb=Ex+Ey+129=1111 1110+1000 0000+1000 0001=1111 1111

阶码上溢！

浮点数运算——浮点数乘/除运算

- 浮点数乘法：$A*B = (M_a * M_b) \cdot 2^{Ea+Eb}$
- 浮点数除法：$A/B = (M_a / M_b) \cdot 2^{Ea-Eb}$

浮点数**尾数采用原码乘/除**运算

浮点数乘 / 除法步骤

（Xm、Ym分别是X和Y尾数原码， Xe和Ye 分别是X和Y阶移码 ）

(1) 求阶： Xe $\pm$ Ye $\mp$ 127

(2) 尾数相乘除： Xm */Ym （两个形为1.xxx的数相乘/除）

(3) 两数符号相同，结果为正；两数符号相异，结果为负；

(4) **当尾数高位为0，需左规；当尾数最高位有进位，需右规。**

(5) 如果尾数比规定的长，则需考虑舍入。

(6) 若尾数是0，则需要将阶码也置0。

(7) 阶码溢出判断

问题1：**乘法**运算结果最多左规几次？最多右规几次？ **不需左规！最多右规1次！**

问题2：**除法**呢？ **左规次数不定！不需右规！**

浮点数运算——浮点数乘/除运算

- **求阶码的和与差**

设Ex和Ey分别是两个操作数的阶码，Eb是结果的阶码，则：

- 阶码加法公式为： $Eb \leftarrow Ex+Ey+129 \pmod{2^8}$

$$[E1+E2]_{移} = 127 + E1 + E2 = 127 + E1 + 127 + E2 - 127$$

$$= [E1]_{移} + [E2]_{移} - 127$$

$$= [E1]_{移} + [E2]_{移} + [-127]_{补}$$

$$= [E1]_{移} + [E2]_{移} + 10000001B \pmod{2^8}$$

- 阶码减法公式为： $Eb \leftarrow Ex+[-Ey]_{补}+127 \pmod{2^8}$

$$[E1-E2]_{移} = 127 + E1 - E2 = 127+E1-(127+E2)+127$$

$$= [E1]_{移}-[E2]_{移} + 127$$

$$= [E1]_{移}+[-[E2]_{移}]_{补}+01111111B \pmod{2^8}$$

设Ex和Ey分别是两个操作数阶码的移码，Eb是结果阶码的移码表示

例：若两个阶码分别为10和-5，求10+(-5)和10-(-5)的移码。

解：Ex = 127+10 =137=1000 1001B

Ey = 127+ (–5) = 122 = 0111 1010B

[–Ey]补= 1000 0110B

将Ex和Ey代入上述公式，得：

Eb = Ex+Ey +129 = 1000 1001 + 0111 1010 +1000 0001

= 1000 0100B = 132 （mod 2^8）

其阶码的和为132 – 127 = 5，正好等于10 + (–5) = 5。

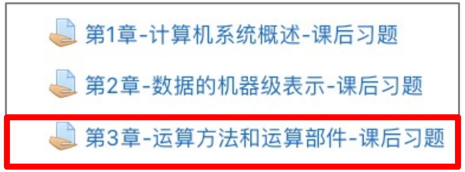
Eb = Ex+[–Ey]补+127 = 1000 1001 + 1000 0110 +0111 1111

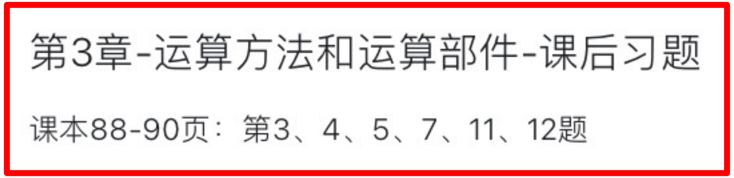
= 1000 1110B = 142 （mod 2^8）

其阶码的差为142–127 = 15，正好等于10 – (–5) = 15。

课程习题（作业）——截止日期：9月30日晚23:59

- **课本88-90页**：第3、4、5、7、11、12题
- 提交方式：https://selearning.nju.edu.cn/（教学支持系统）





- 命名：学号+姓名+第*章。
- 若提交遇到问题请及时发邮件或在下一次上课时反馈。

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，该机器使用二进制补码表示带符号整数。无符号数采用逻辑移位，带符号整数采用算术移位。请填写下表，并说明函数 func1 和 func2 的功能。

w		func1(w)		func2(w)	
机器数	值	机器数	值	机器数	值
	127				
	128				
	255				
	256				

课程习题（作业）——截止日期：9月30日晚23:59

4. 填写下表，注意对比无符号整数和带符号整数(用补码表示)的乘法结果，包括截断操作前后的结果。

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
无符号整数	110		010					
带符号整数	110		010					
无符号整数	001		111					
带符号整数	001		111					
无符号整数	111		111					
带符号整数	111		111					

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()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;
}
```

课程习题（作业）——截止日期：9月30日晚23:59

7. 已知 $x=10$，$y=-6$，采用 6 位机器数表示。请按如下要求计算，并把结果还原成真值。

(1) 求$[x+y]_{补}$，$[x-y]_{补}$。

(2) 用原码一位乘法计算$[x\times y]_{原}$。

(3) 用 MBA(基 4 布斯算法)计算$[x\times y]_{补}$。

(4) 用不恢复余数法计算$[x/y]_{原}$的商和余数。

(5) 用不恢复余数法计算$[x/y]_{补}$的商和余数。

课程习题（作业）——截止日期：9月30日晚23:59

11. 假设浮点数格式为：阶码是 4 位移码，偏置常数为 8，尾数是 6 位补码（采用双符号位），用浮点运算规则分别计算以下表达式在不采用任何附加位和采用 2 位附加位（保护位、舍入位）两种情况下的值（假定采用就近舍入到偶数方式）。

(1) $(15/16)\times2^7+(2/16)\times2^5$　　(2) $(15/16)\times2^7-(2/16)\times2^5$

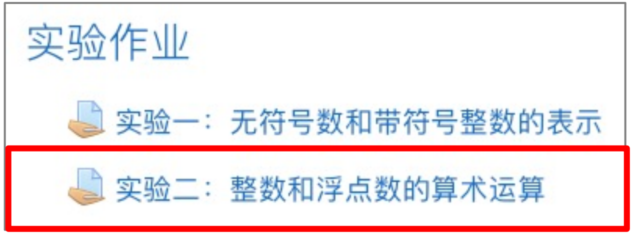
(3) $(15/16)\times2^5+(2/16)\times2^7$　　(4) $(15/16)\times2^5-(2/16)\times2^7$

12. 采用 IEEE 754 单精度浮点数格式计算下列表达式的值。

(1) $0.75+(-65.25)$　　(2) $0.75-(-65.25)$

课程实验——截止日期：10月14日晚23:59

- 提交方式：https://selearning.nju.edu.cn/（教学支持系统）



- 命名：学号+姓名+实验*。
- **提交：文件打包，提交ZIP压缩文件。**

南京大学 NANJING UNIVERSITY

课程实验——截止日期：10月14日晚23:59

实验二 整数和浮点数的算术运算

实验目的: 通过检查高级语言中数据运算的不同结果，进一步理解机器代码在 CPU 中的执行过程，从而为更好地学习指令系统设计和 CPU 设计打下良好的基础。

实验要求: 编程计算下列表达式的值:

(1) unsigned int 型数据: 1+4294967295=?;1-4294967295=?

(2) int 型数据: 2147483647+1=?;-2147483648-1=?

(3) float 型数据: (1.0 + 123456.789e30) + (-123456.789e30) = ?;

1.0 + (123456.789e30 + (- 123456.789e30)) = ?

要求分别用十进制和十六进制形式显示各种操作的结果。

实验报告:

1. 给出源程序(文本文件)和执行结果。
2. 分别给出每个运算结果的解释说明

提问

Q & A

殷亚凤
智能软件与工程学院
苏州校区南雍楼东区225
yafeng@nju.edu.cn，https://yafengnju.github.io/