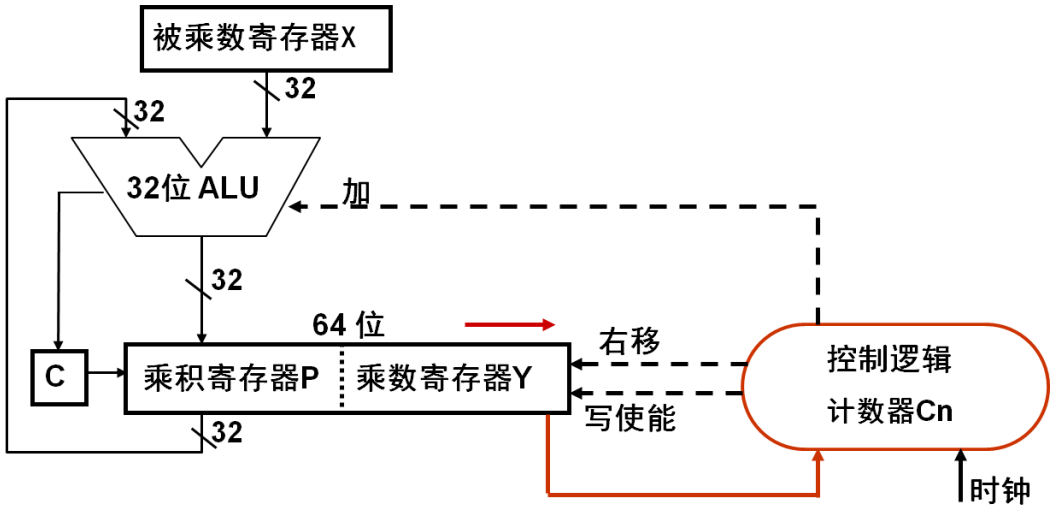
回顾第14次课

无符号数乘法：

“判断”、“加”+“右移”

(每次逻辑右移1位)

32位无符号乘法运算的硬件实现



原码乘法：符号和数值分开运算，用于浮点数尾数乘法运算。

一位乘法：每次取乘数的最右一位进行判断，无符号数乘法实现

两位乘法：每次取乘数的最右两位进行判断，需要对无符号数乘法实现进行一些修改（用补码实现减法、每次算术右移2位，等）

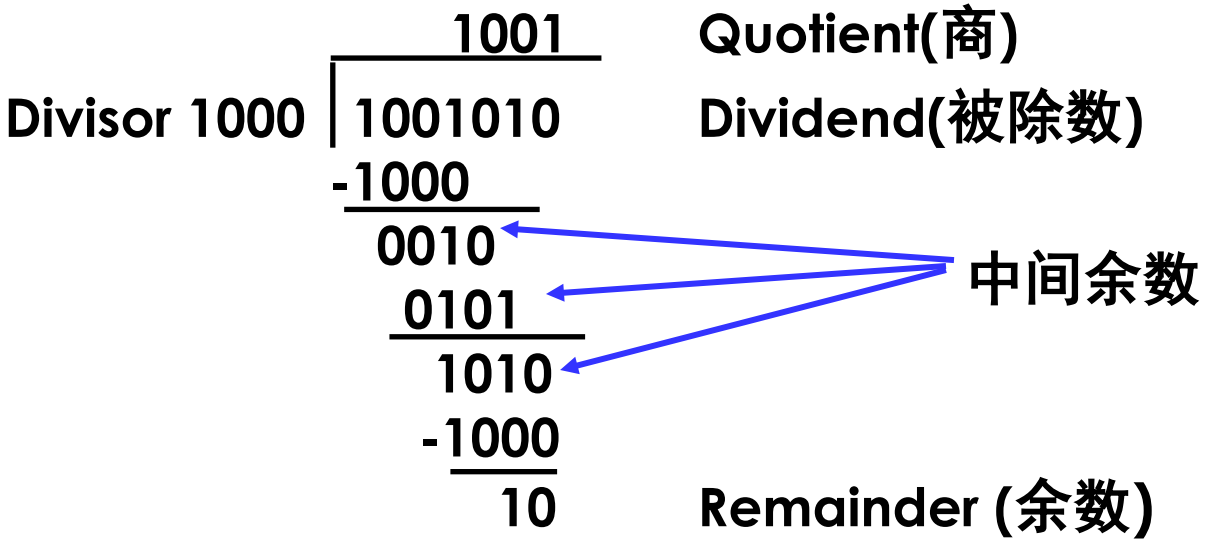
补码乘法（布斯乘法）：符号和数值一起运算，算术右移

回顾第14次课（续）

- **判断溢出的方法**
  - **硬件**
  - **编译时**
  - **高级语言**

运算	X	Y	X×Y	P	溢出否
无符号乘	0110	1010	0011 1100	1100	
带符号乘	0110	1010	1101 1100	1100	
无符号乘	1000	0010	0001 0000	0000	
带符号乘	1000	0010	1111 0000	0000	
无符号乘	1101	1110	1011 0110	0110	
带符号乘	1101	1110	0000 0110	0110	
无符号乘	0010	1100	0001 1000	1000	
带符号乘	0010	1100	1111 1000	1000	

除法Divide: Paper & Pencil



- 手算除法的基本要点

（1）被除数减去除数（以除数的位数为准对齐）

够减则上商为1；不够减则上商为0。得到的差为中间余数

（2）除数右移一位，然后用中间余数减去除数或0

够减则上商1；否则上商0。得到的差仍为中间余数，

重复执行本步骤。

（3）直到求得的商的位数足够为止。

定点除法运算

- 除前预处理

什么时候做？谁在做？　　软件or硬件

①若被除数=0且除数≠0，或定点整数除法|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

③若被除数和除数都为0，则有些机器产生一个不发信号的NaN，即“quiet NaN”

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
  - 基本操作为减法（用加法实现）和移位，与乘法用同一套硬件

减法计算完之后发现不够减怎么办？

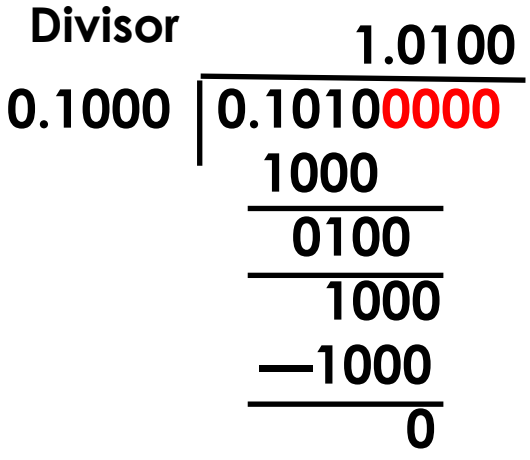
说明这次本不该减！要恢复余数！

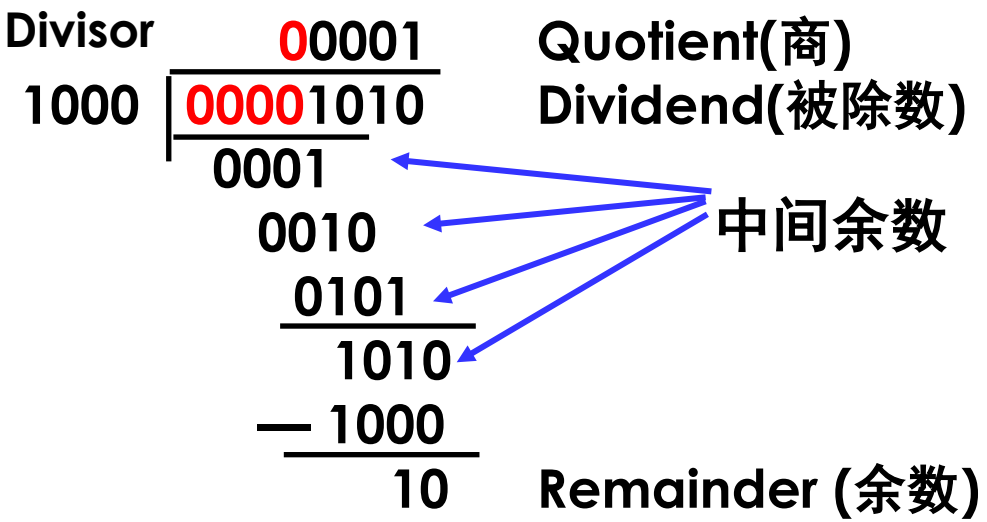
定点除法运算（两个n位正数相除的情况）

(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





n位除法需 n+1步

手算中的“除数右移”改为“被除数（中间余数）左移”

第一次试商为1时的情况

问题：第一次试商为1，说明什么?

商有n+1位数，因而溢出!

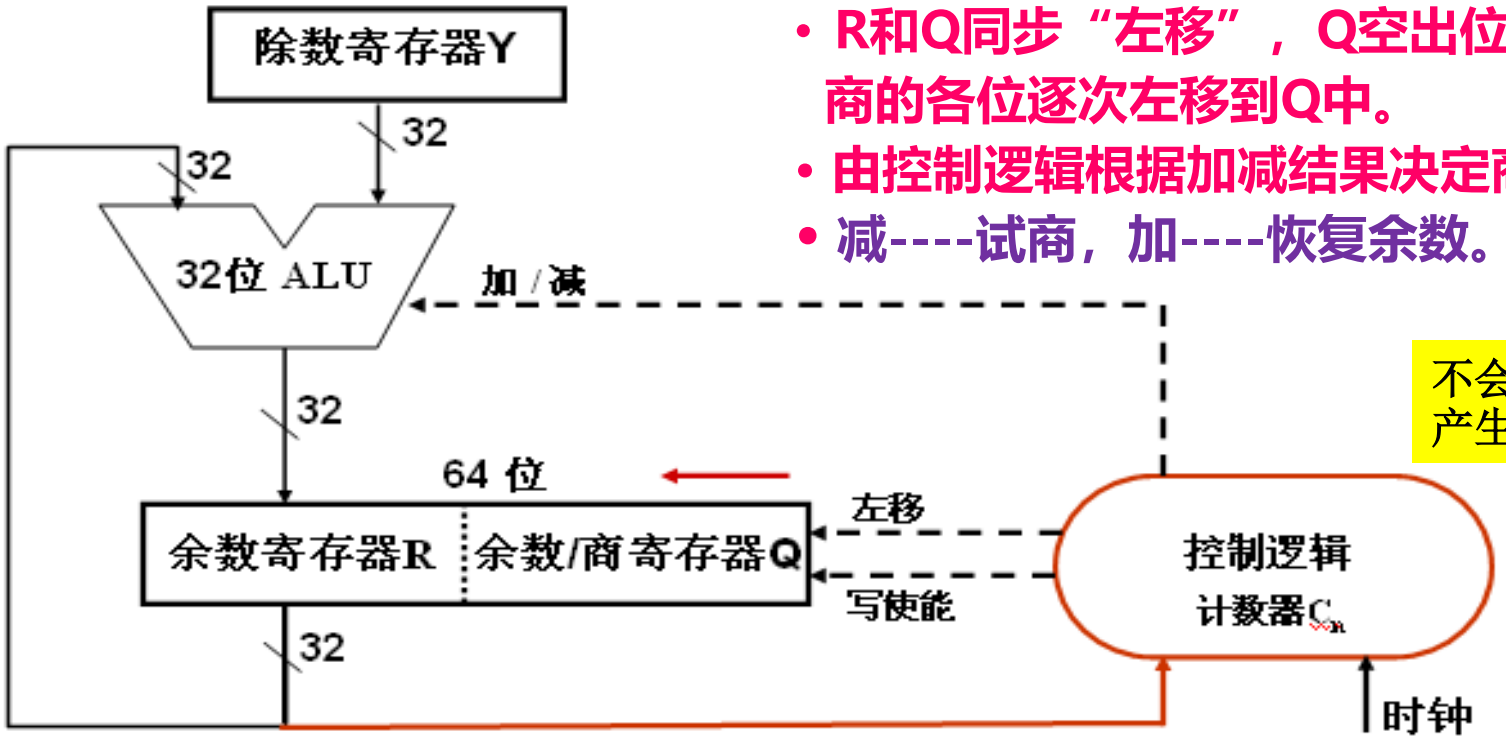
若是2n位除以n位的无符号整数运算，则说明将会得到多于n+1位的商，因而结果“溢出”（即：无法用n位表示商）。

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

无符号数除法算法的硬件实现



- 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。

不会因为加法而产生最高进位C

- 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Cn：存放循环次数。初值是32（不包括第一次试商），每循环（移位）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。

无符号数除法例子

4位无符号数（但数值范围只能是1-7，因为用补码实现减法，最高位需留做符号位），最后商是4位。

整数，所以R初始在高位扩展0

验证：7 / 2 = 3 余 1

+D = 0010
–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	R: 1110 1110
+D, sl R, 0	D: 0010	R: 0001 1100
R = R–D	D: 0010	R: 1111 1100
+D, sl R, 0	D: 0010	R: 0011 1000
R = R–D	D: 0010	R: 0001 1000
sl R, 1	D: 0010	R: 0011 0001
R = R–D	D: 0010	R: 0001 0001
sl R, 1	D: 0010	R: 0010 0011
Shr R(rh)	D: 0010	R: 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

最后为了上商，把余数也左移了一位（共移了5次），故最后余数需向右移一位

不恢复余数除法(加减交替法)

恢复余数法可进一步简化为“加减交替法”

根据恢复余数法(设D为除数，$R_i=2R_{i-1}-D$为第i次中间余数)，有：

- 若$R_i<0$,则商上“0”，做加法恢复余数，即：
- $R_{i+1}=2(R_i+D)-D=2R_i+D$ （“负，左移，上商0，加”）
- 若$R_i>=0$,则商上“1”，不需恢复余数，即：
- $R_{i+1}=2R_i-D$ （“正，左移，上商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

Divide Algorithm example

验证：7 / 2 = 3 余 1　　–D = 1110

R：被除数（中间余数）；D：除数

R: 0000 0111

R = R–D　　1110

1110

sl R, 0　　R: 1100 1110

R = R+D　　0010

1110

sl R, 0　　R: 1101 1100

R = R+D　　0010

1111

sl R, 0　　R: 1111 1000

R = R+D　　0010

0001

sl R, 0　　R: 0011 0001

R = R-D　　1110

0001 0011

第1次上商为“试商”

不恢复余数法、加减交替法

负，0，加

正，1，减

第1位商为0，表示结果不溢出，最后（第5次）左移出去，并加上最后一位商

且最后一次上商1，余数无需恢复就是正确的。

这里的最后 一步，余数保持不变，没有和商一起左移，所以也不用右移了

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。

原码除法举例（小数）

已知 $[X]_{原} = 0.1011$

$[Y]_{原} = 1.1101$

用恢复余数法计算$[X/Y]_{原}$

解：$[|X|]_{补} = 0.1011$

$[|Y|]_{补} = 0.1101$

$[-|Y|]_{补} = 1.0011$

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。

小数在低位扩展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

所以，$[X/Y]_{原}$=1.1101(最高位为符号位)，余数为

用于判断是否溢出

若求$[Y/X]_{原}$结果溢出

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

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：$[|X|]_{补} = 0.1011$

$[|Y|]_{补} = 0.1101$

$[-|Y|]_{补} = 1.0011$

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 $q_0=1$

“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-够减）

补码除法能否这样来判断呢？

补码除法

- 补码除法判断是否“够减”的规则

（1）当被除数（或当前余数）与除数同号时，做减法，得到新余数

（2）当被除数（或当前余数）与除数异号时，做加法，得到新余数

若新余数的符号与当前余数符号一致表示够减，否则为不够减；

当前余数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		

即：余数不变号够减、变号不够减

经过分析归纳（过程略），得到接下来的不恢复余数法

补码（n位，包括1位符号位）不恢复余数法

- **算法要点：**

判断是否同号（决定加or减、上商）不是新老余数之间！而是余数和除数Y之间

(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R1与Y同号，则q_n置1，转下一步；

② 若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+1 ，按以下规则求出q_n和接下来的n位商（i=n+1时，只需置商）：

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

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

同、1、减

异、0、加

(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X与Y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X符号，则不需修正，余数在R中；否则，按下列规则进行修正：当X和Y符号相同时，最后余数加Y；否则，最后余数减Y。

其运算过程也呈加/减交替方式，因此也称为“加减交替法”。

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

除以2^k的快速处理——右移k位

- 无符号整数：逻辑右移，高位补0，低位丢弃
- 带符号整数：算术右移，高位补符，低位丢弃

举例：

unsigned 16/4=4 ： 0001 0000>>2 = 0000 0100

signed -16/4=-4 ： 1111 0000>>2 = 1111 1100

提醒：
N位被除数扩充为2N位，在高还是低位补？补0还是补符？
列竖式的时候N到底是几位（几位数值位，几位符号位）？

整数除法的近似处理（除以2^k）

- 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（地板）：移出的低位直接丢弃
  - 带符号负整数（天板）：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（错）

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

变量与常数之间的除运算—举例

- 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 + 0) : (x+31))>>5

但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。

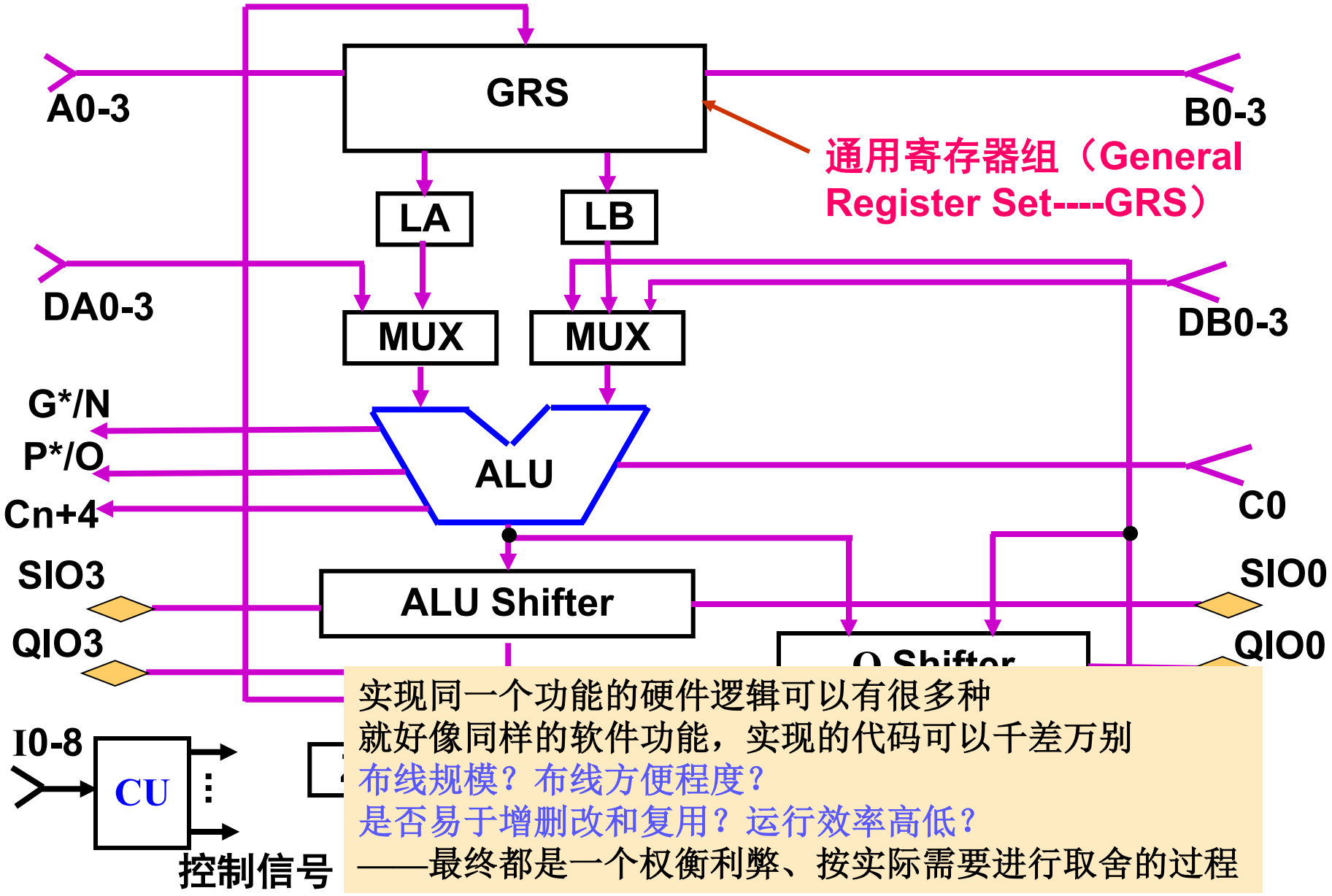
```
int div32(int x)
{  /* 根据x的符号得到偏移量b */
    int b=(x>>31) & 0x1F;
    return (x+b)>>5;
}
```

定点运算部件

- **综合考虑各类定点运算算法后，发现：**
  - **所有运算都可通过“加”和“移位”操作实现**
- **以一个或多个ALU（或加法器）为核心，加上移位器、存放中间临时结果的寄存器组，在相应控制逻辑的控制下， 通过多路选择器和实现数据传送的总线等，即可以实现各种运算——也就是构成了一个运算数据通路。**
  - 可用专门运算器芯片实现（如：4位运算器芯片AM2901）
  - 可用若干芯片级联实现（如4个AM2901构成16位运算器）
  - **现代计算机把运算数据通路和控制器都做在CPU中，为实现高级流水线，CPU中有多个运算部件，通常称为“功能部件”或“执行部件”。**

“运算器（Operate Unit）”、“运算部件（Operate Unit）”、“功能部件（Function Unit）”、“执行部件（Execution Unit）”和“数据通路（DataPath）”的含义基本上一样，只是强调的侧重不同

定点运算器芯片举例-AM2901A（不要求）



实现同一个功能的硬件逻辑可以有很多种
就好像同样的软件功能，实现的代码可以千差万别
布线规模？布线方便程度？
是否易于增删改和复用？运行效率高低？
——最终都是一个权衡利弊、按实际需要进行取舍的过程

RISC-V中整数的乘、除运算处理

ISA要素：指令+数据类型+寄存器设计等

- 乘法指令: mul，mulh，mulhu，mulhsu
  - mul rd, rs1, rs2：将低32位乘积存入结果寄存器rd
  - mulh、mulhu：将两个乘数同时按带符号整数（mulh）、同时按无符号整数（mulhu）相乘，高32位乘积存入rd中
  - mulhsu：将两个乘数分别作为带符号整数和无符号整数相乘后得到的高32位乘积存入rd中
  - 得到64位乘积需要两条连续的指令，其中一定有一条是mul指令，实际执行时只有一条指令
  - 两种乘法指令都不检测溢出，而是直接把结果写入结果寄存器。由软件根据结果寄存器的值自行判断和处理溢出
- 除法指令: div ，divu，rem，remu
  - div / rem：按带符号整数做除法，得到商 / 余数
  - divu / remu：按无符号整数做除法，得到商 / 余数

第二讲小结

逻辑运算、移位运算、扩展运算等电路简单

主要考虑算术运算

- **定点运算涉及的对象**

  无符号数；带符号整数(补码)；原码小数；移码整数

- **定点运算：(ALU实现基本算术和逻辑运算，ALU+移位器实现其他运算)**

  补码加/减：符号位和数值位一起运算，减法用加法实现。同号相加时可能溢出

  原码加/减：符号位和数值位分开运算，用于浮点数尾数加/减运算 （等到浮点数运算时介绍）

  移码加减：移码的和、差等于和、差的补码，用于浮点数阶码加/减运算（等到浮点数运算时介绍）

第二讲小结

乘法运算：

无符号数乘法：“加”+“右移”

原码（一位/两位）乘法：符号和数值分开运算，数值部分用无符号数乘法实现，用于浮点数尾数乘法运算。

补码（一位/两位）乘法：符号和数值一起运算，采用Booth算法。

快速乘法器：流水化乘法器、阵列乘法器

除法运算：

无符号数除法：用“加/减”+“左移”，有恢复余数和不恢复余数两种。

原码除法：符号和数值分开，数值部分用无符号数除法实现，用于浮点数尾数除法运算。

补码除法：符号位和数值位一起。有恢复余数和不恢复余数两种。

- **定点运算部件**

本章总结（1）

定点数运算：由ALU + 移位器实现各种定点运算

- 移位运算
  - 逻辑移位：对无符号数进行，左（右）边补0，低（高）位移出
  - 算术移位：对带符号整数进行，移位前后符号位不变，编码不同，方式不同。
  - 循环移位：最左（右）边位移到最低（高）位，其他位左（右）移一位。
- 扩展运算
  - 零扩展：对无符号整数进行高位补0
  - 符号扩展：对补码整数在高位直接补符
- 加减运算
  - 补码加/减运算：用于整数加/减运算。符号位和数值位一起运算，减法用加法实现。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。
  - 原码加/减运算：用于浮点数尾数加/减运算。符号位和数值位分开运算，同号相加，异号相减；加法直接加；减法用加负数补码实现。
- 乘法运算：用加法和右移实现。
  - 补码乘法：用于整数乘法运算。符号位和数值位一起运算。采用Booth算法。
  - 原码乘法：用于浮点数尾数乘法运算。符号位和数值位分开运算。数值部分用无符号数乘法实现。
- 除法运算：用加/减法和左移实现。
  - 补码除法：用于整数除法运算。符号位和数值位一起运算。
  - 原码除法：用于浮点数尾数除法运算。符号位和数值位分开运算。数值部分用无符号数除法实现。

本章总结（2）

- 浮点数运算：由多个ALU + 移位器实现
  - 加减运算
    - 对阶 、尾数相加减、规格化处理、舍入、判断溢出
  - 乘除运算
    - 尾数用定点原码乘/除运算实现，阶码用定点数加/减运算实现。
  - 溢出判断
    - 当结果发生阶码上溢时，结果发生溢出，发生阶码下溢时，结果为0。
  - 精确表示运算结果
    - 中间结果增设保护位、舍入位、粘位
    - 最终结果舍入方式：就近舍入 / 正向舍入 / 负向舍入 / 截去四种方式。
- **ALU的实现**
  - **算术逻辑单元ALU：实现基本的加减运算和逻辑运算。**
  - **加法运算是所有定点和浮点运算（加/减/乘/除）的基础，加法速度至关重要**
  - **进位方式是影响加法速度的重要因素**
  - **并行进位方式能加快加法速度**
  - **通过“进位生成”和“进位传递”函数来使各进位独立、并行产生**
- 作业：习题3、4、5、6、7（11月19号晚上24:00之前交）