Computer Organization and Design

Arithmetic for Computers Part II Multiplication & Division

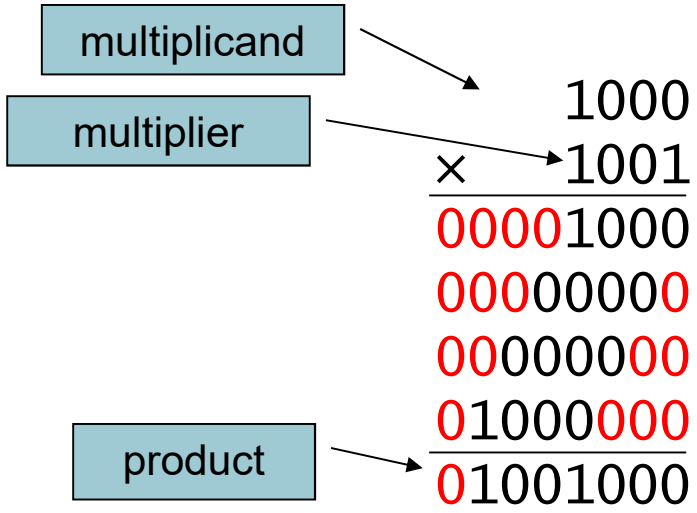
Jiang Zhong

zhongjiang@cqu.edu.cn

Multiplication

- 先回顾二进制乘法的人工计算
- 在计算机中实现面临哪些问题?

```
      1000
×     1001
------------
      1000
     0000
    0000
   1000
------------
   1001000
```



Length of product is the sum of operand lengths

Multiplication

- Start with long-multiplication approach

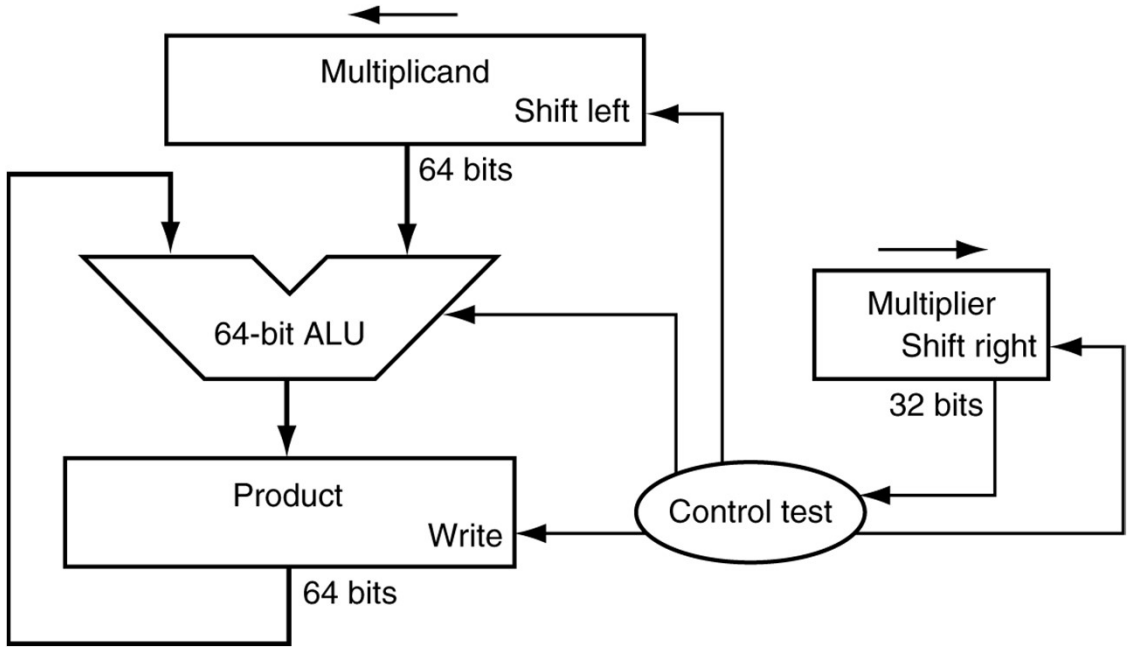
multiplicand

multiplier

product

```
  1000
× 1001
------
  1000
 0000
0000
1000
------
01001000
```

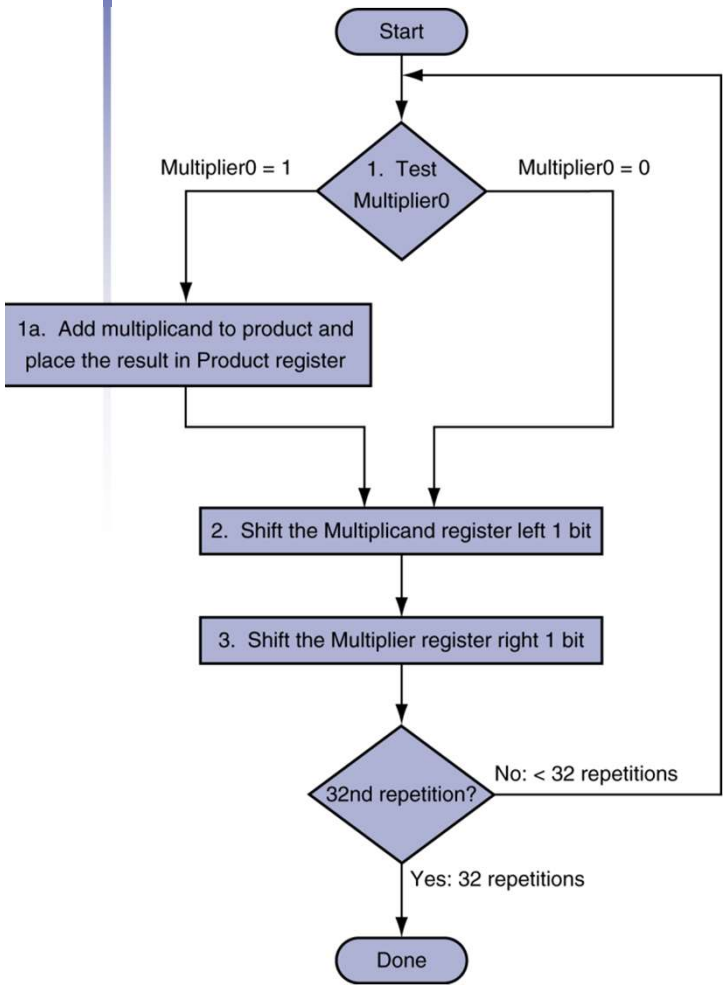
Length of product is the sum of operand lengths

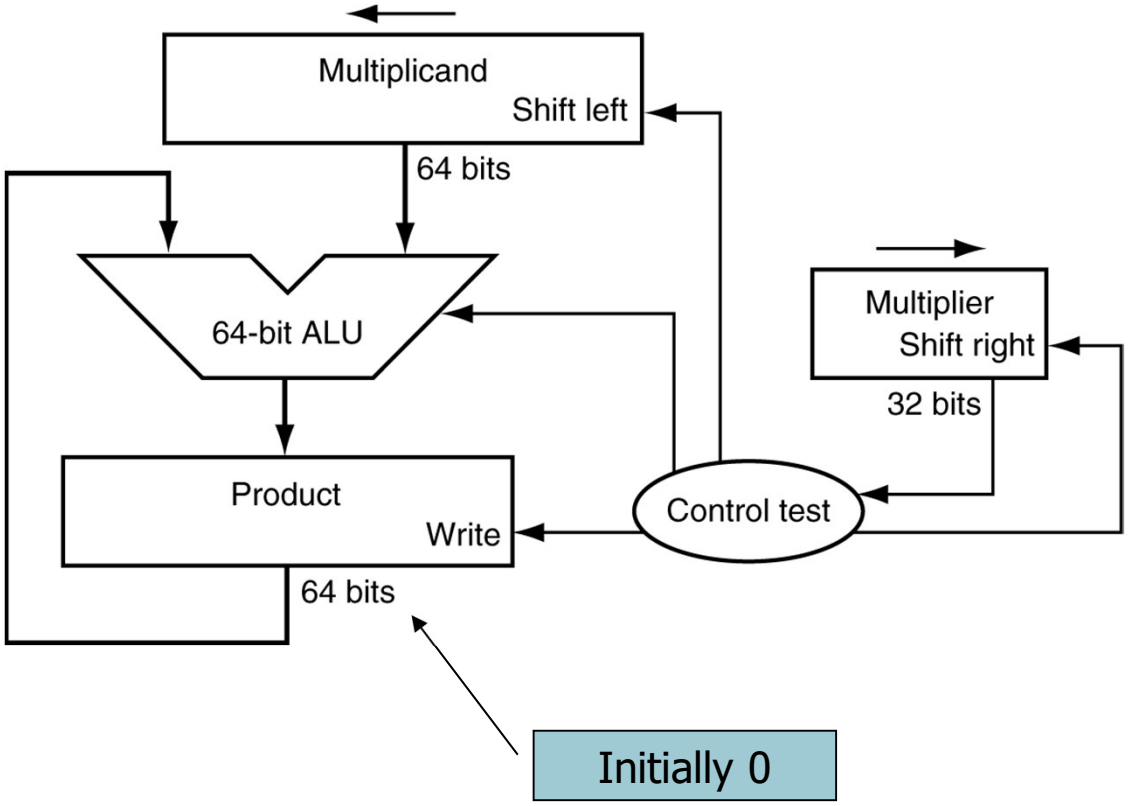


Multiplication Hardware

Sequential Implementation

~100 cycles for multiplying 2 32-bit values





定点小数的乘法计算过程(与定点整数的结论一致)

Let x,y are fractional fixed-point both x and y less then 1.

$x=0.x_1x_2.....x_n$ <1

$y=0.y_1y_2.....y_n$ <1

The product of x and y is

$x \cdot y = x(0.y_1y_2.....y_n)$

$= x(y_1 2^{-1} + y_2 2^{-2} + ... + y_n 2^{-n})$

$= 2^{-1}(y_1 x + 2^{-1}(y_2 x + 2^{-1}(... + 2^{-1}(y_{n-1} x + 2^{-1}(y_n x + 0))...)))$

the recurrence formula for product

let z_i denotes the ith product, we could get the recurrence formula

$$
\begin{aligned}
z_0 &= 0, \\
z_1 &= 2^{-1}(y_n x + z_0) \\
z_2 &= 2^{-1}(y_{n-1} x + z_1) \\
&\vdots \\
z_i &= 2^{-1}(y_{n-i+1} x + z_{i-1}) \\
&\vdots \\
z_n &= x \cdot y = 2^{-1}(y_1 x + z_{n-1})
\end{aligned}
$$

特点：每次只需要相加两个数，然后右移一位。且相加的两个数（部分积和位积）都只有n位，因而不需要2n位的加法器。

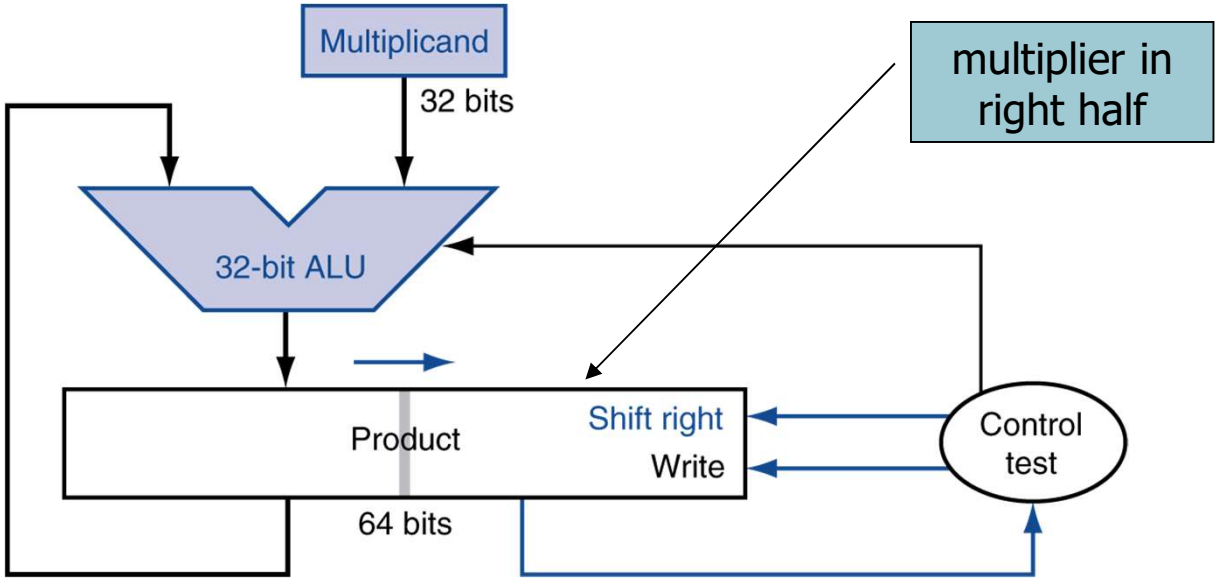
例：x=0.1101，y=0.1011，求 x·y 。

部分积	乘数	说明
0 0.0 0 0 0	y_f 1 0 1 1	$z_0=0$
+ 0 0.1 1 0 1		$y_4=1$，+x
0 0.1 1 0 1		
→ 0 0.0 1 1 0	1 y_f 1 0 1	右移，得z_1
+ 0 0.1 1 0 1		$y_3=1$，+x
0 1.0 0 1 1		
→ 0 0.1 0 0 1	1 1 y_f 1 0	右移，得z_2
+ 0 0.0 0 0 0		$y_2=0$，+0
0 0.1 0 0 1		
→ 0 0.0 1 0 0	1 1 1 y_f 1	右移，得z_3
+ 0 0.1 1 0 1		$y_1=1$，+x
0 1.0 0 0 1		
→ 0 0.1 0 0 0	1 1 1 1 y_f	右移，得$z_4=x\cdot y$

最后结果：　$x\cdot y=0.10001111$

Optimized Multiplier

- Perform steps in parallel: add/shift



- One cycle per partial-product addition
  - That's ok, if frequency of multiplications is low
  - ~32 cycles for multiplying 2 32-bit values

主观题 10分

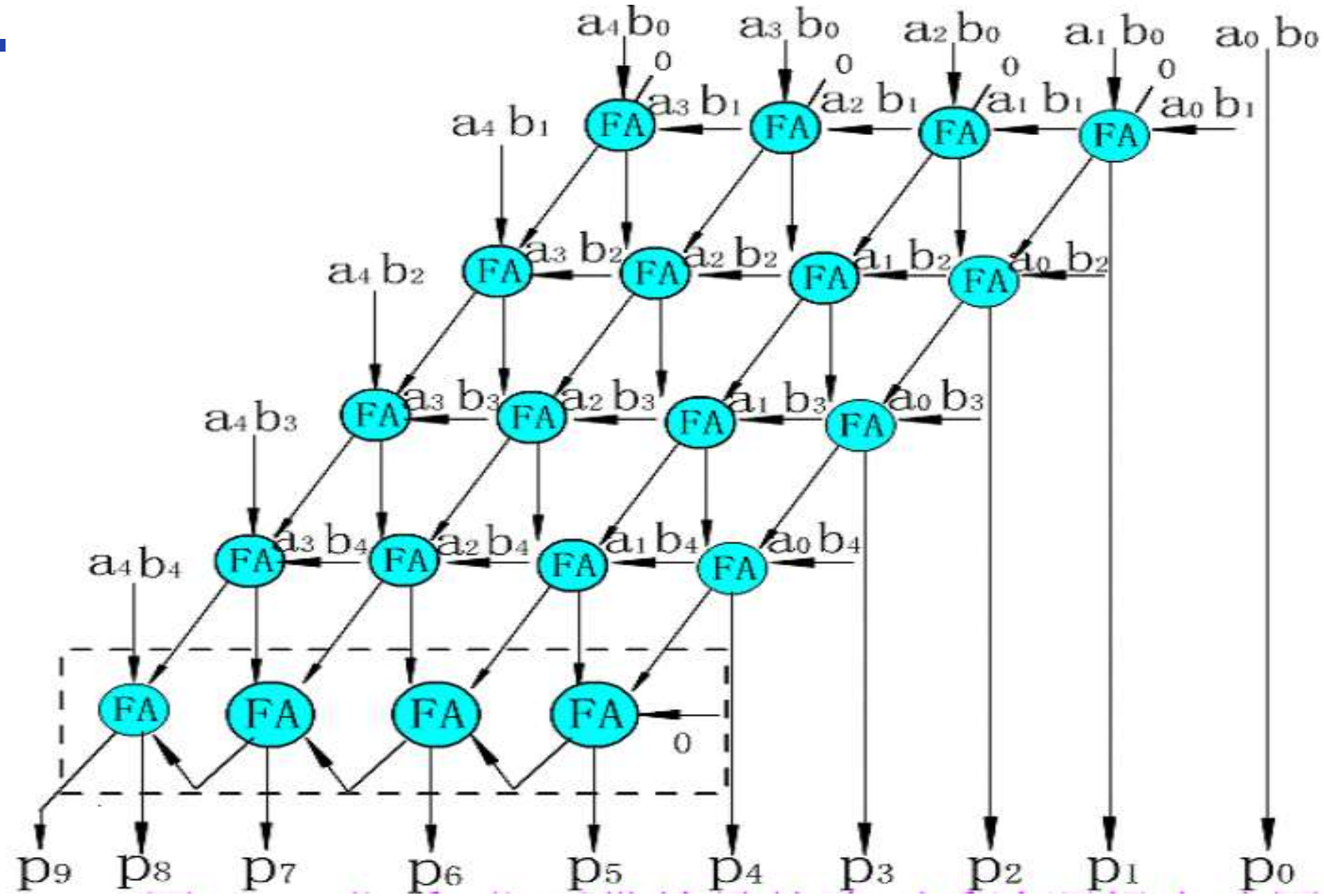
设置

计算：$1.110_{10} * 10^{10} * 9.200_{10} * 10^{-5}$

正常使用主观题需2.0以上版本雨课堂

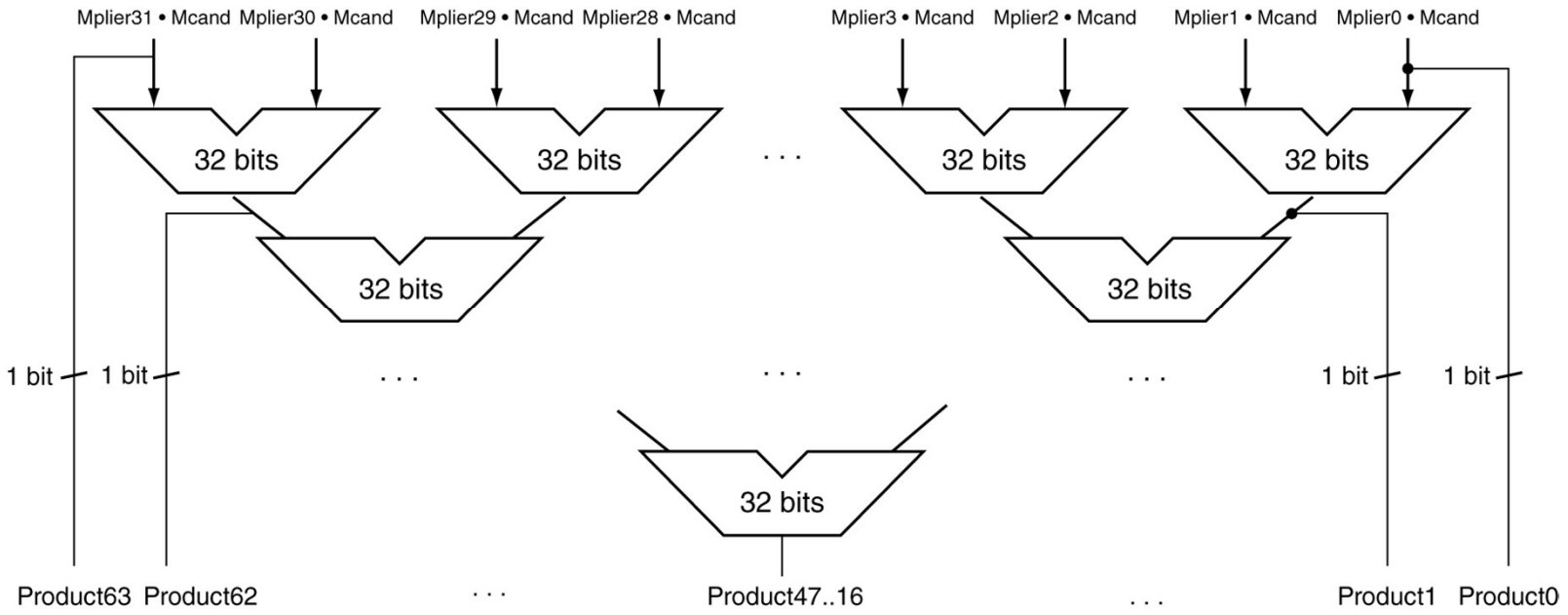
作答

Even Faster Multiplier



Even Faster Multiplier

- Uses multiple adders
  - Cost/performance tradeoff
    - 31 adders → $\log_2(31)$ = ~5 cycles for multiplying 2 32-bit values



- Can be pipelined
  - Several multiplication performed in parallel

MIPS Multiplication

- Two 32-bit registers for product
  - HI: most-significant 32 bits
  - LO: least-significant 32-bits
- Instructions
  - `mult rs, rt` / `multu rs, rt`
    - 64-bit product in HI/LO
  - `mfhi rd` / `mflo rd`
    - Move from HI/LO to rd
    - Can test HI value to see if product overflows 32 bits
  - `mul rd, rs, rt`
    - Least-significant 32 bits of product –> rd

主观题 10分

设置

请计算 0.1011 *0. 1101

正常使用主观题需2.0以上版本雨课堂

作答

3.4 Division

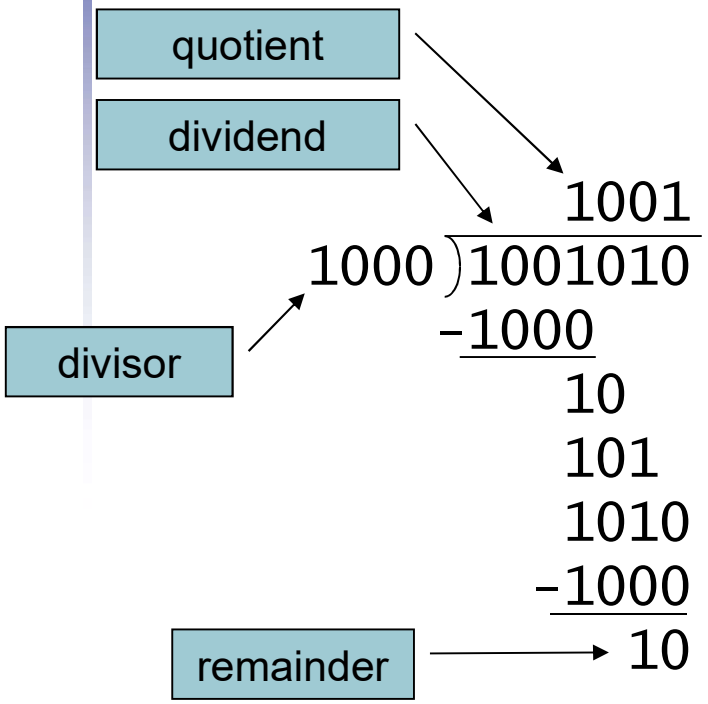
- **十进制整数除法回顾**
- 二进制整数除法
- 原码1位除法思想
- 恢复余数除法及其优化
- 加减交替除法
- 定点小数除法
- 快速除法概况

3.4 Division-- Decimal Division

```
    0 2 1
  ---------
6 ) 1 2 8
    0
  -------
    1
    1 2
    1 2
  -------
      0
      0 8
        6
  -------
        2
```

```
      0 2
  -------
62 ) 1 2 8
     0 0
   -------
     1 2
     1 2 8
     1 2 4
   -------
         4
```

Division -- Binary Division



n-bit operands yield *n*-bit quotient and remainder

计算机实现n位二进制除需解决的问题：

1）余数（部分余数）与除数如何对齐?

固定的位数2*n（n）位

2）如何进行试商?

通过比较余数和除数大小(减法)

3）计算（次数?）结束的条件?

设置固定n+1次迭代

4）如何处理符号位?

单独处理符号位

3.4 Division

- 十进制整数除法回顾
- 二进制整数除法
- 原码1位除法思想
- 恢复余数除法及其优化
- 加减交替除法
- 定点小数除法
- 快速除法概况

（1）原码1位除法

设有两个n位定点整(小)数：

被除数：X，其原码为 $[x]_{原} = x_f . x_{n-1} \cdots x_1 x_0$

除数　：Y，其原码为 $[y]_{原} = y_f . y_{n-1} \cdots y_1 y_0$

商　　：Q = X/Y,其原码为

$$[Q]_{原} = (x_f \oplus y_f) . (x_{n-1} \cdots x_1 x_0 / y_{n-1} \cdots y_1 y_0)$$

原码1位除的思想为：

1）商的符号运算 $q_f = x_f \oplus y_f$

2）商的数值运算，实质是两个正整数求商（余）的过程

恢复余数法和加减交替法

主观题 2分

设置

设计二进制正整数求商和余数的算法

假设X= $x_{n-1}\cdots x_1x_0$,Y= $y_{n-1}\cdots y_1y_0$,

X/Y的商为 Q= $q_{n-1}\cdots q_i\cdots q_1q_0$

请同学们设计一个算法来计算商Q中的每一位取值q_i？

提示1：令 X= Y*Q + R = Y * （$q_{n-1}\cdots q_1q_0$）+ R ， R为余数

X= q_{n-1} *Y*2^{n-1} +… q_i *Y*2^i +… q_1 *Y*2 + q_0 *Y + R

提示2：如果X>=Y * 2^{n-1} ,那么 q_{n-1}=1 ， 否则为q_{n-1}=0

正常使用主观题需2.0以上版本雨课堂

作答

正整数求商算法

假设X=Y * Q + R，其中Q为商，R为余数

算法：正整数求商和余数算法

输入：n位的二进制被除数X和除数Y

输出：商Q和余数R

step 0: 令 t = n-1, Q =0; R=X

step 1. R=R- Y * 2^t

if R>=0

q_t=1;

else q_t=0; R=R + Y * 2^t

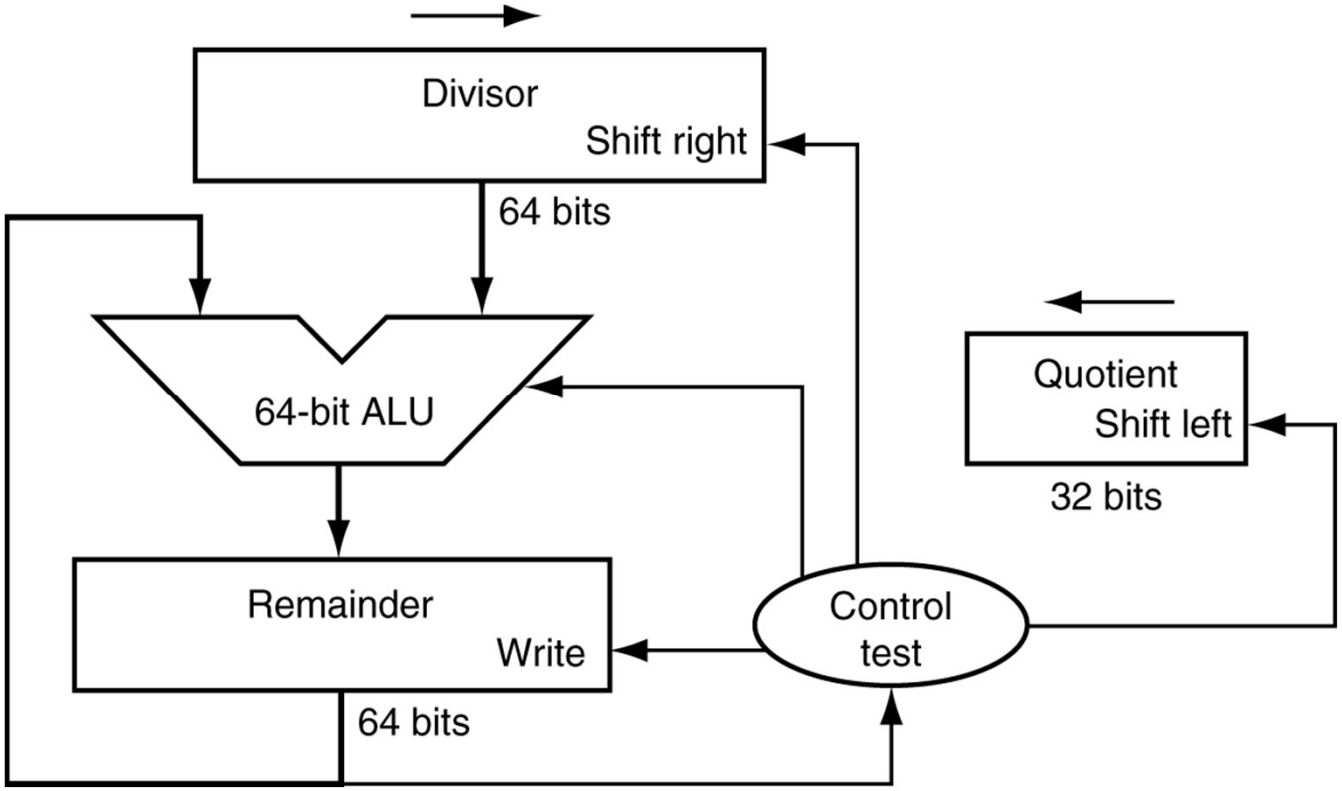
step 2. t=t-1

step 3 if t>=0 goto step1

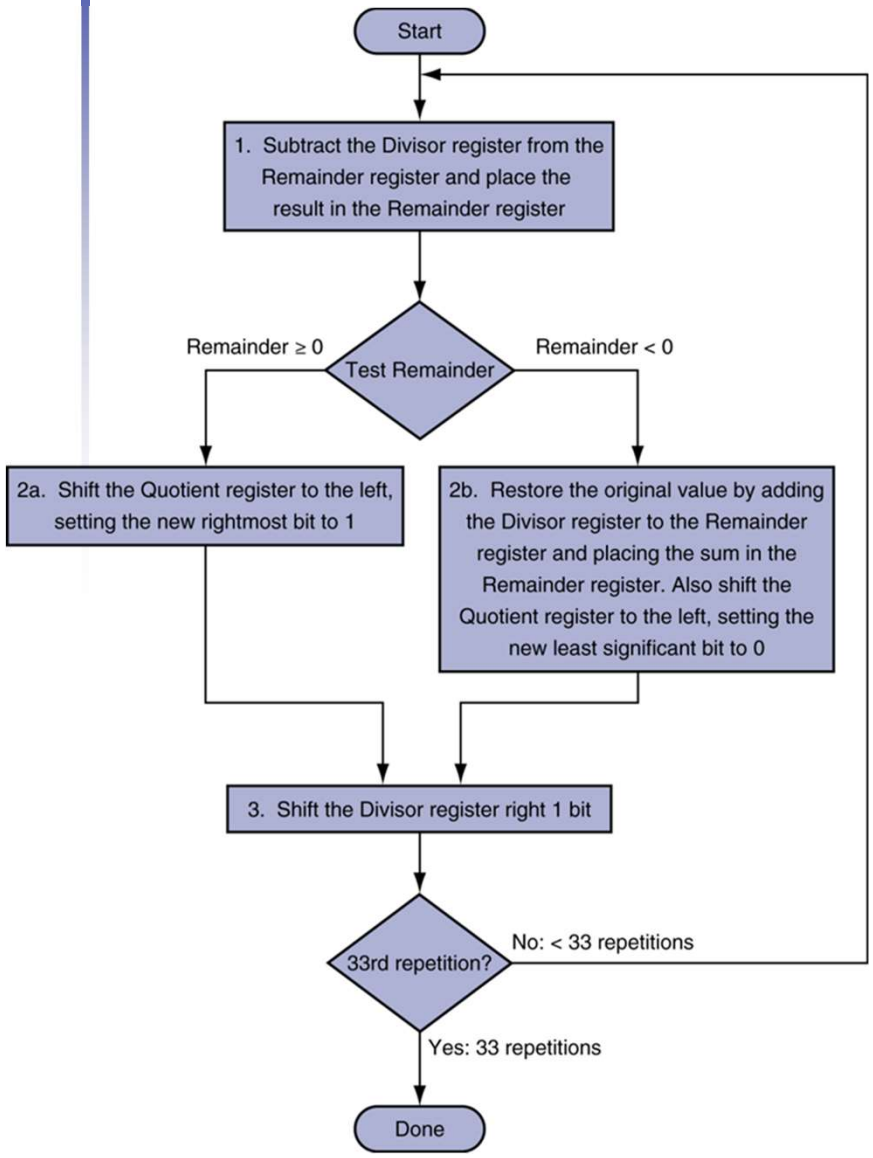
step 4 R=X

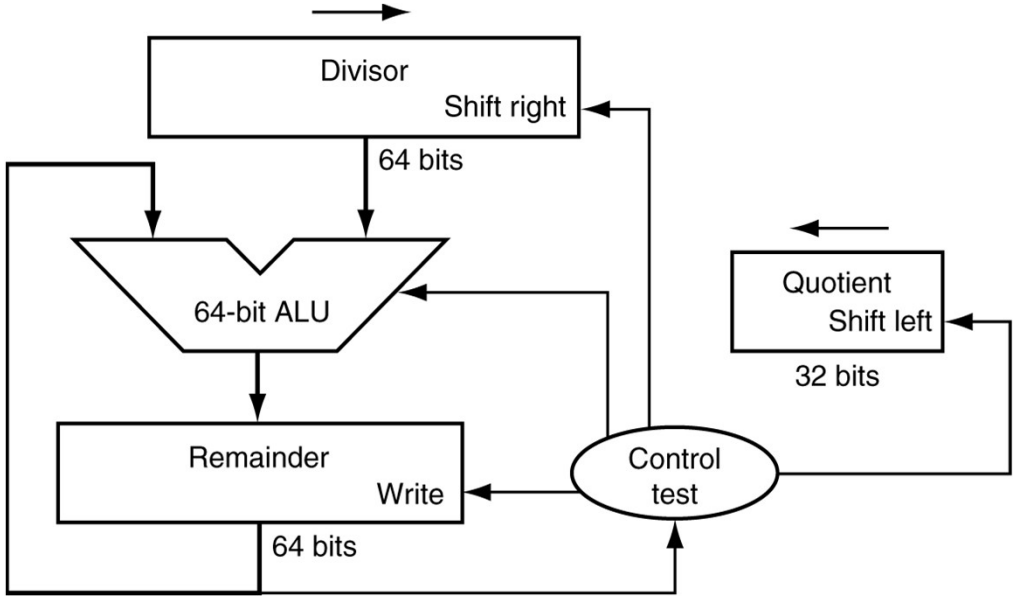
step 5 输出R, Q

Integer Division Hardware



计算机中恢复余数算法流程





恢复余数法的基本特点

- **对齐方式**：除数和被除（余数）均为2*n位

  除数左移n位扩展到2n位，被除数则通过无符号扩展

- **用1试商**: 每次试商1，即采用余数-除数

  根据余数符号位判断是否有错。若有错误需要恢复余数。

- **移位处理**：除数右移1位，商左移1位置最低位值

- **运算次数**：n位运算需要执行n+1次计算

Demo

令

$|x| = [0111]_2$　$|y| = [0010]_2$

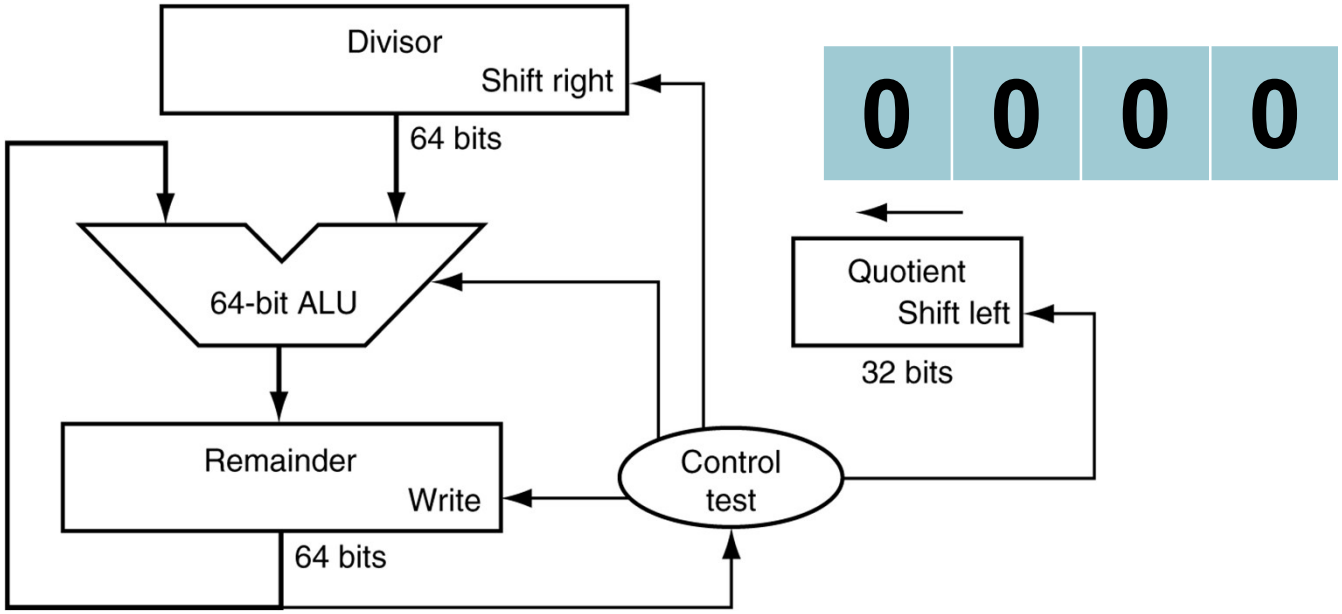
求解　$|x/y|$

Integer Division Example

$|x| = [0111]_2 \quad |y| = [0010]_2$

初始化

0	0	1	0	0	0	0	0

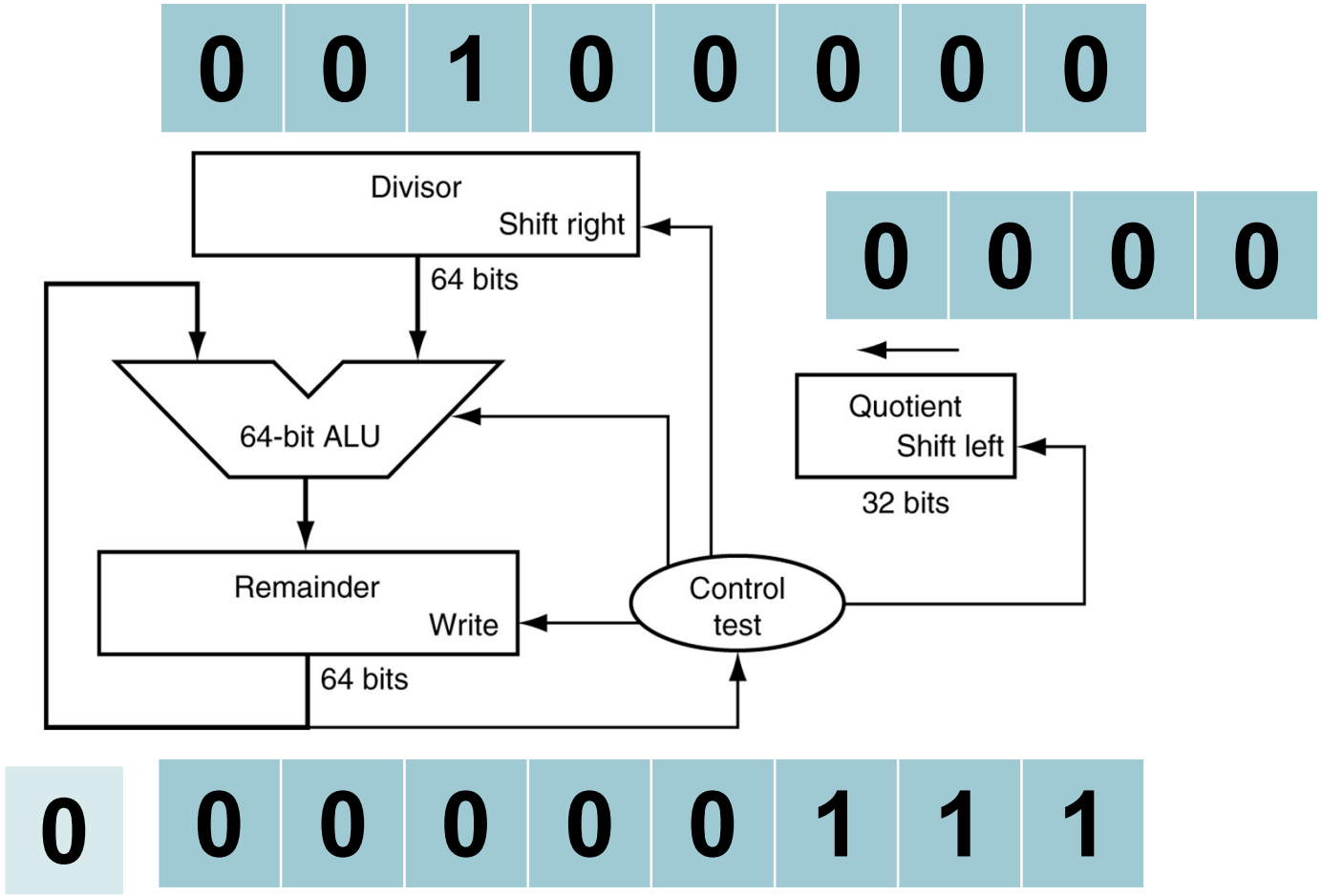


0	0	0	0

0	0	0	0	0	0	1	1	1

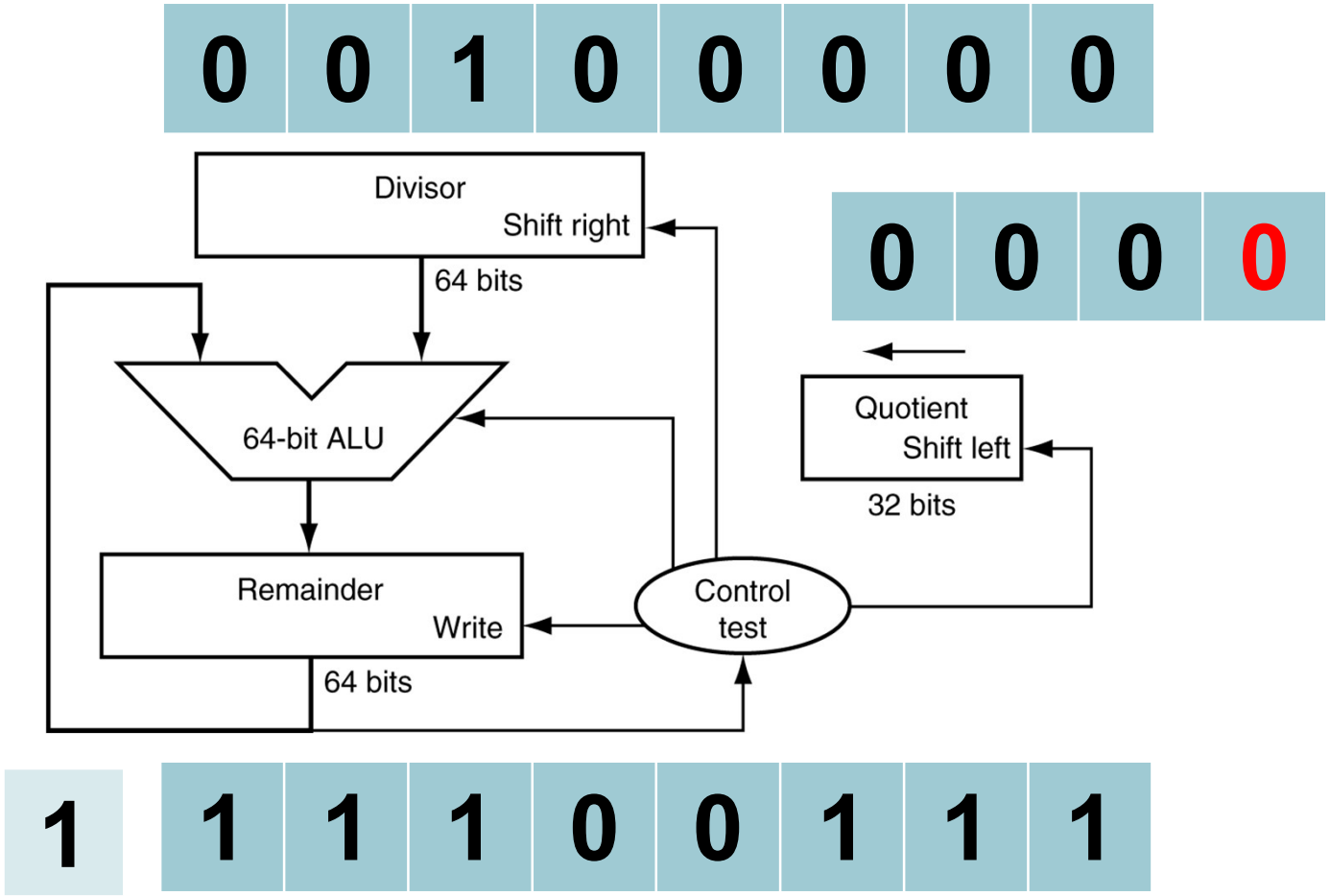
Integer Division Example

Step 1.1 余数-除数,试商



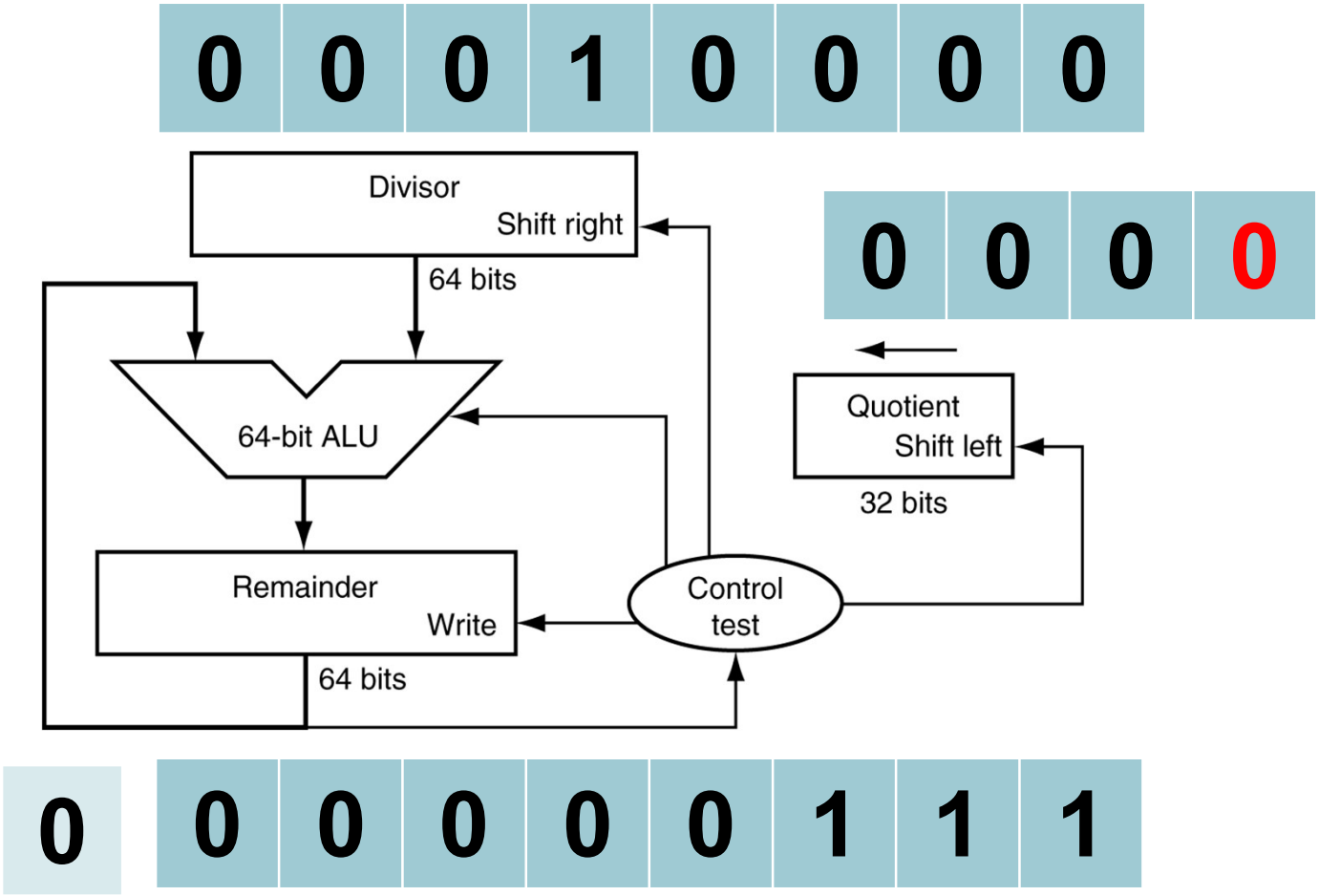
Integer Division Example

Step 1.2 余数<0,商左移1位，置0



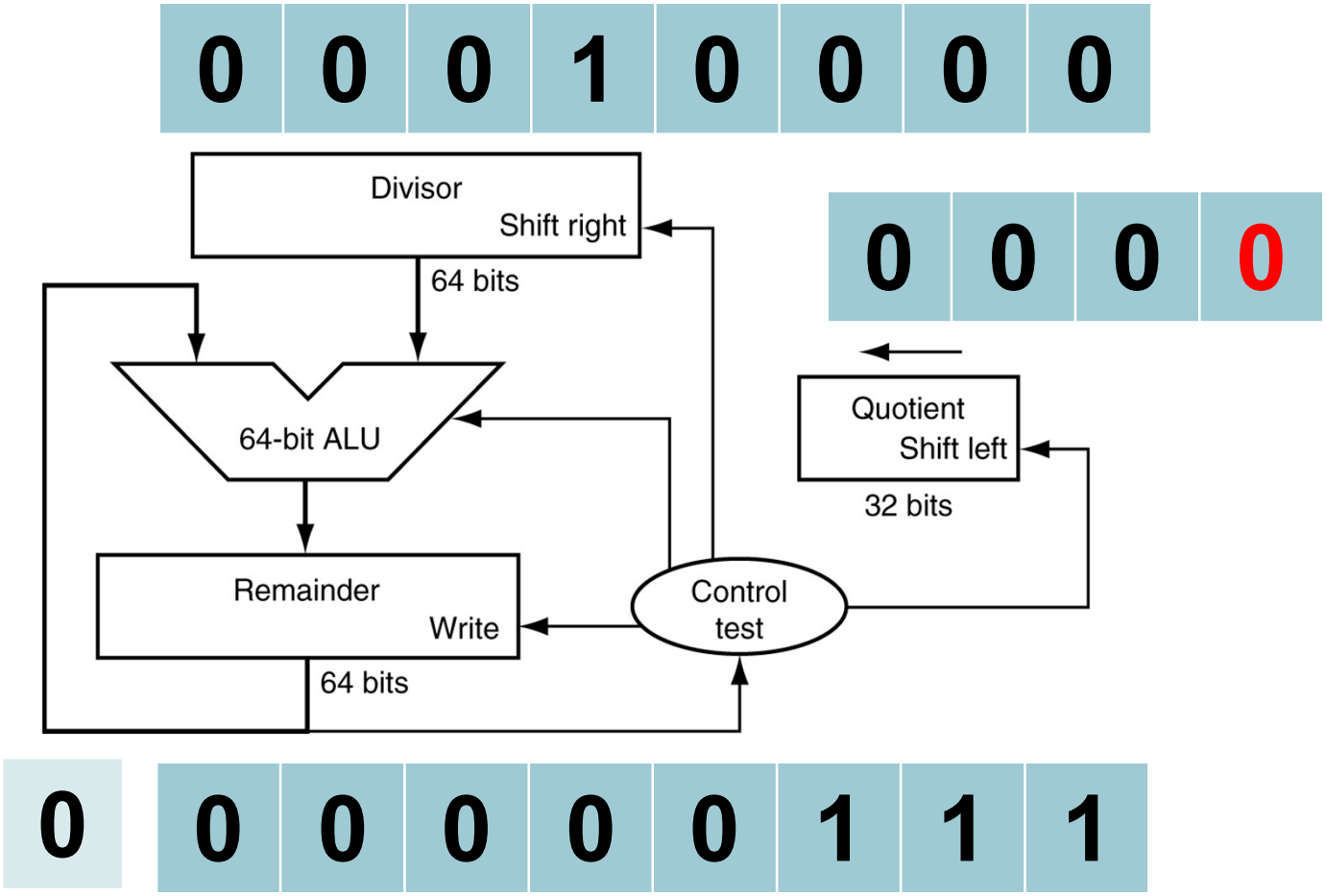
Integer Division Example

Step 1.3 恢复余数，除数右移1位



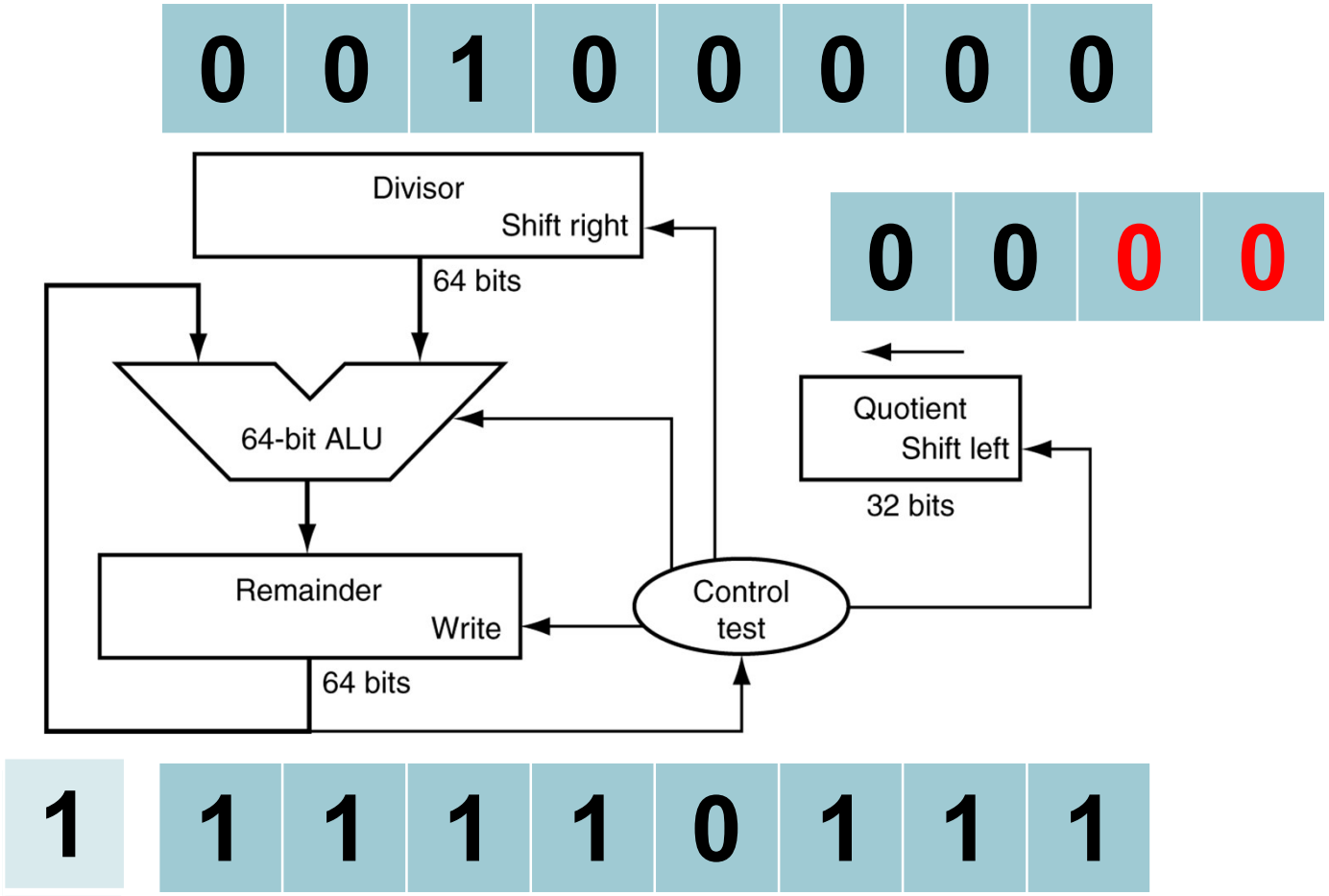
Integer Division Example

Step 2.1余数-除数 试商



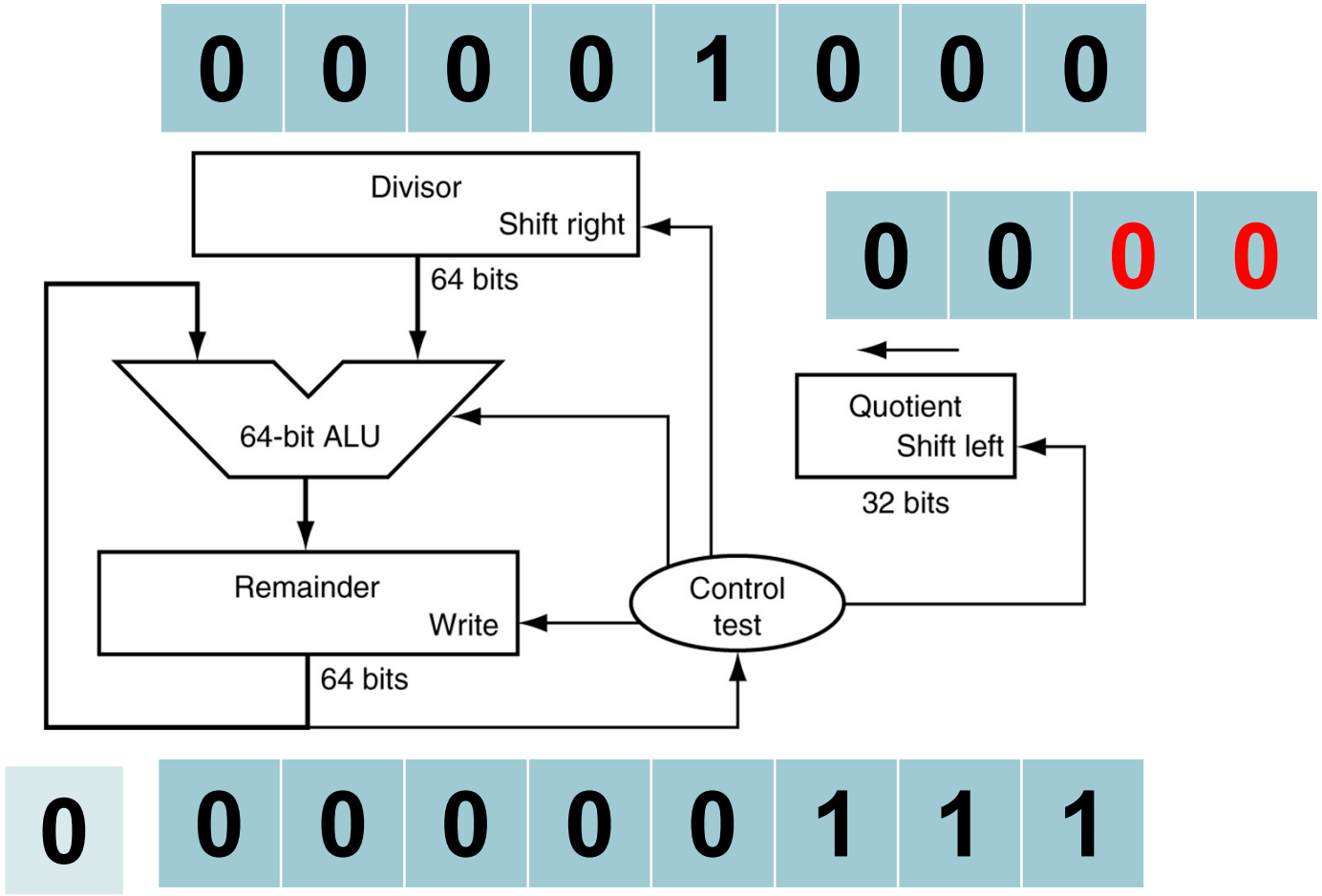
Integer Division Example

Step 2.2 余数<0,商左移1位，置0



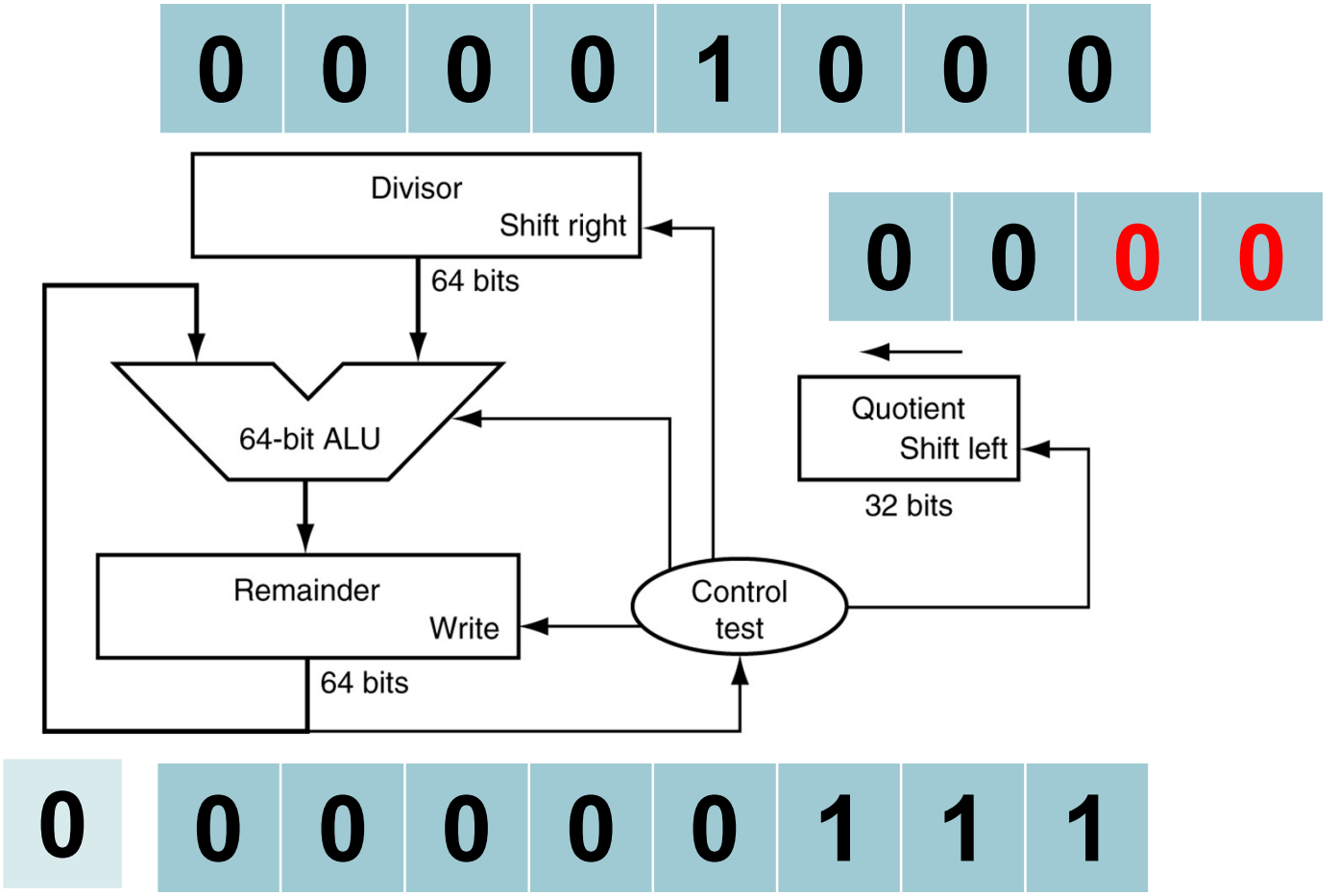
Integer Division Example

Step 2.3 恢复余数，除数右移1位



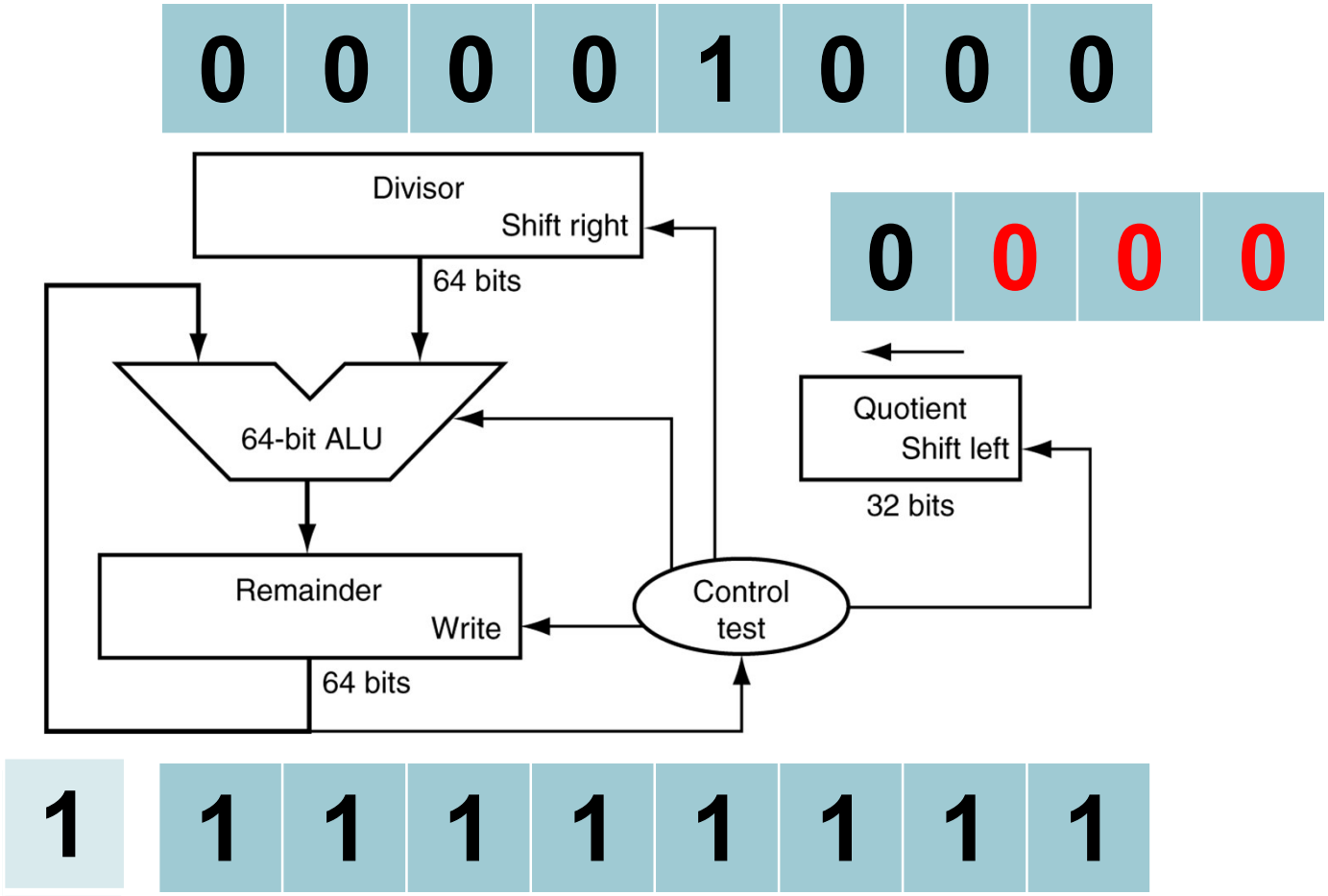
Integer Division Example

Step 3.1 余数-除数 试商



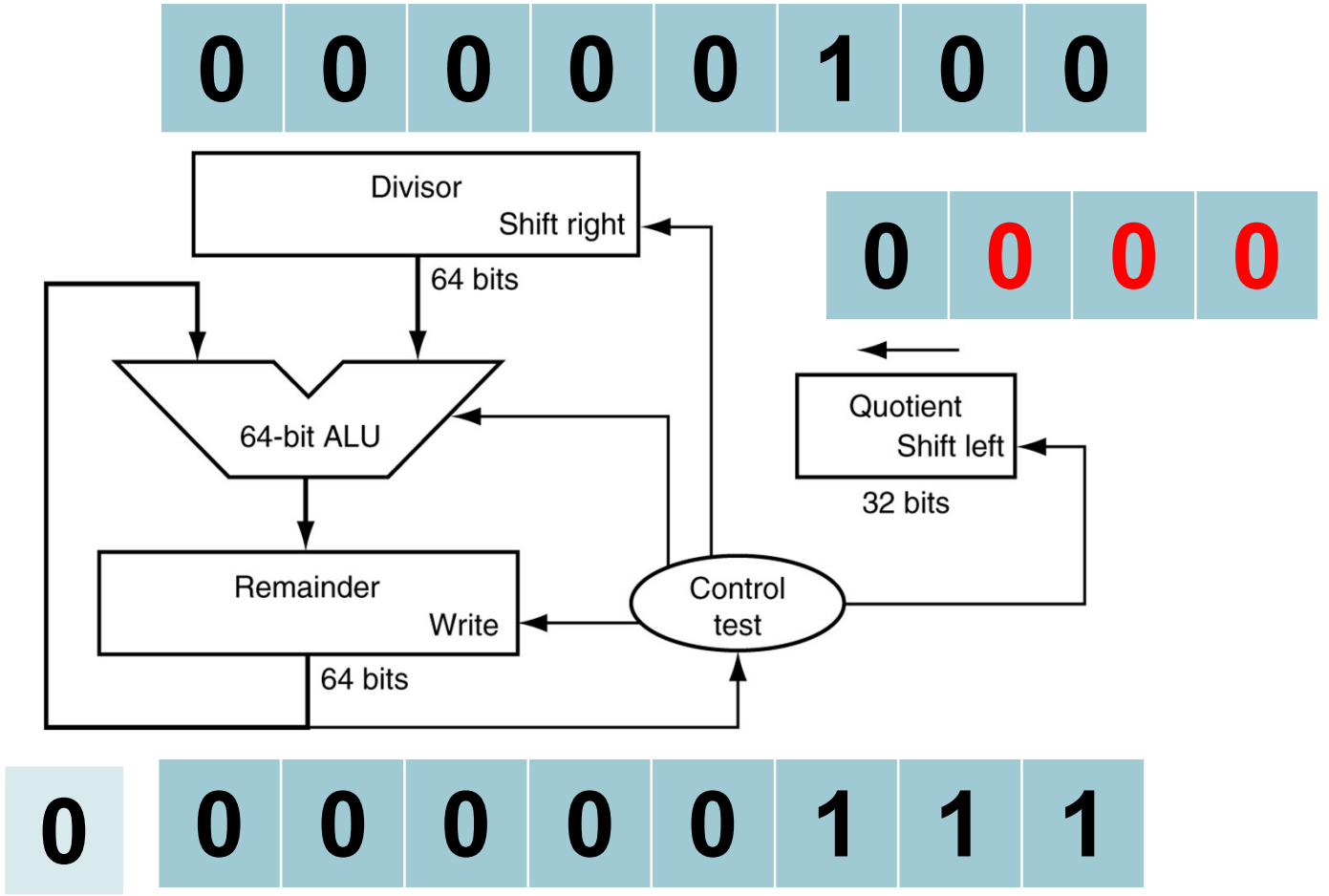
Integer Division Example

Step3.2 余数<0,商左移1位，置0



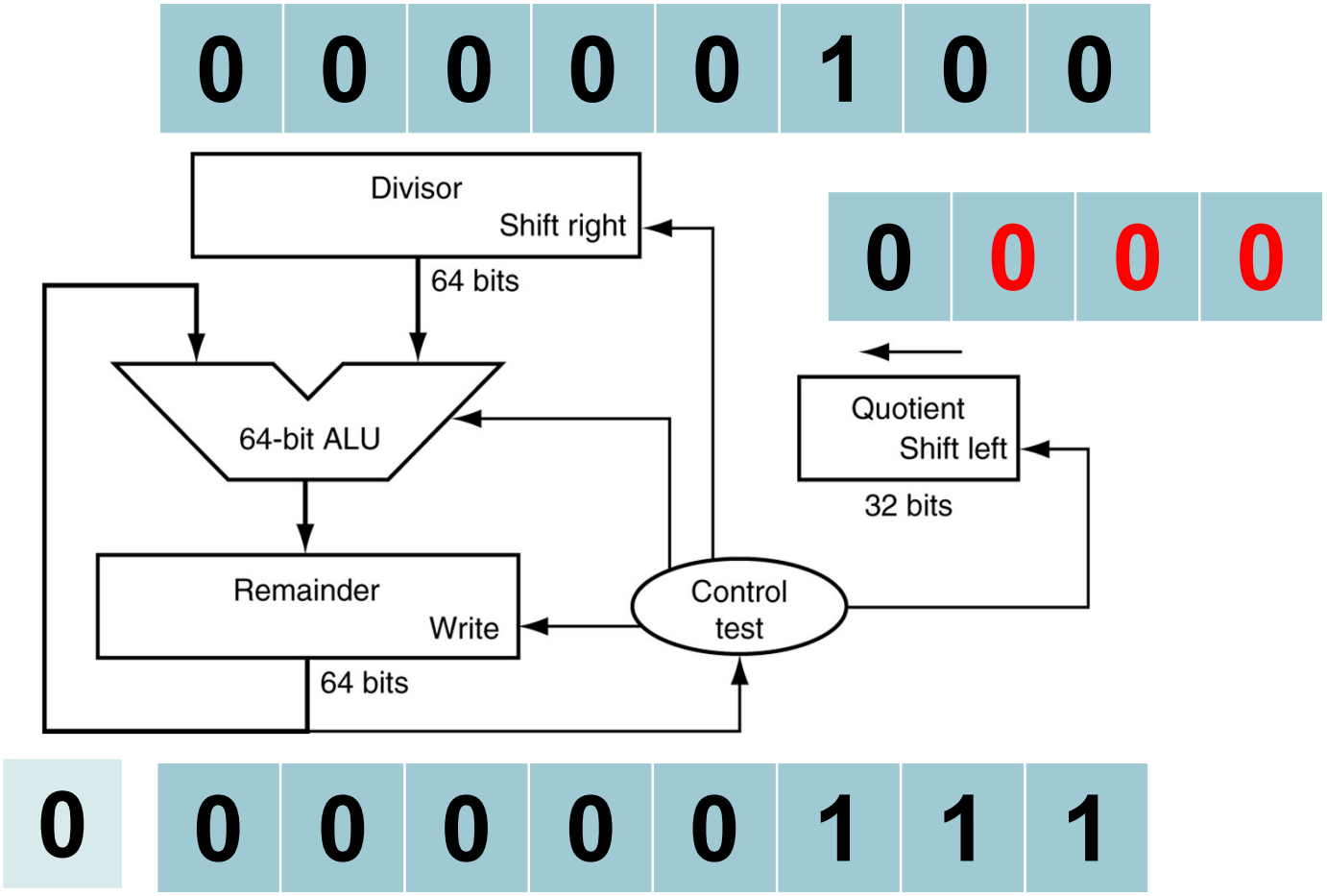
Integer Division Example

Step 3.3 恢复余数，除数右移1位



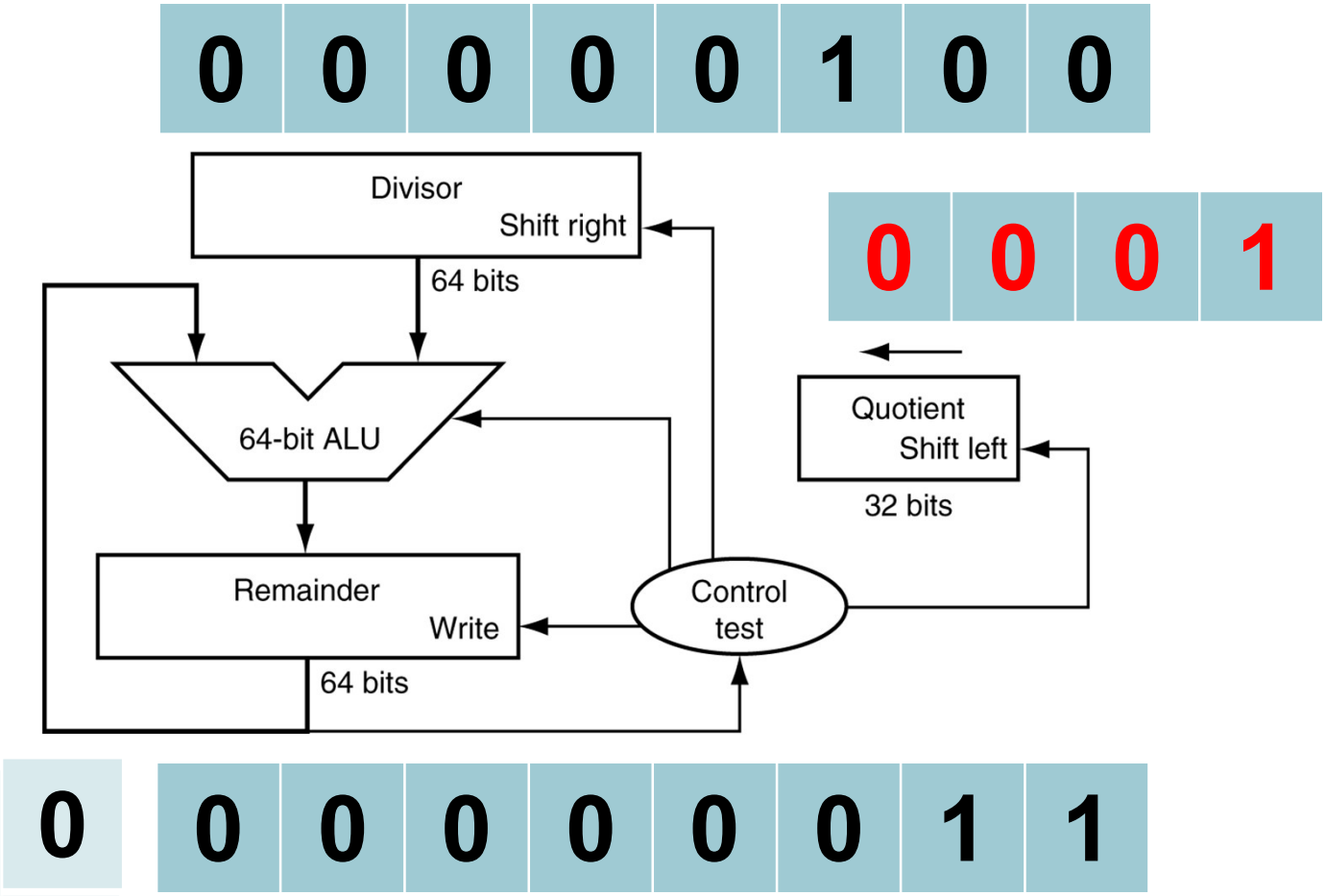
Integer Division Example

Step 4.1 余数-除数 试商



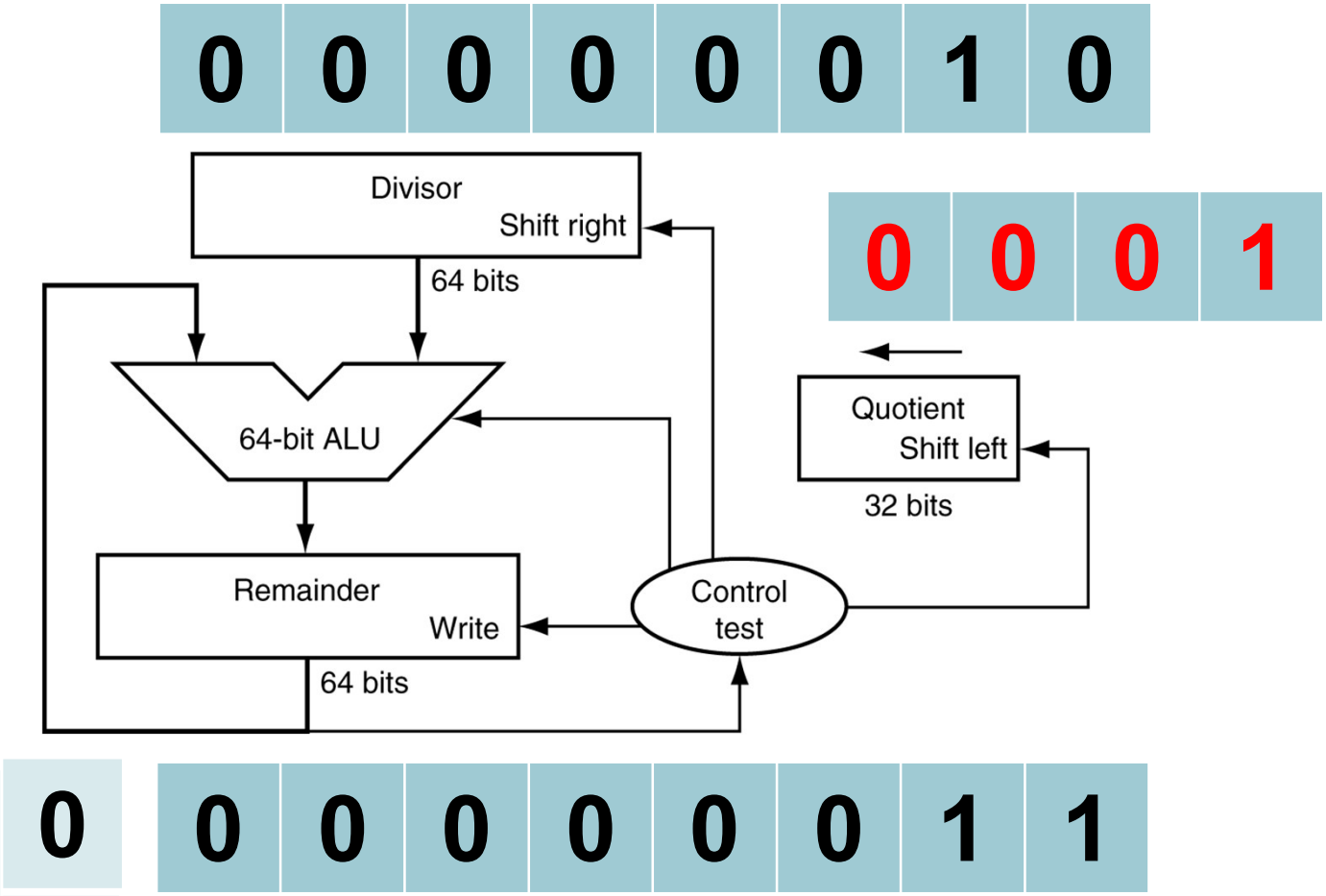
Integer Division Example

Step4.2 余数>0,商左移1位，置1



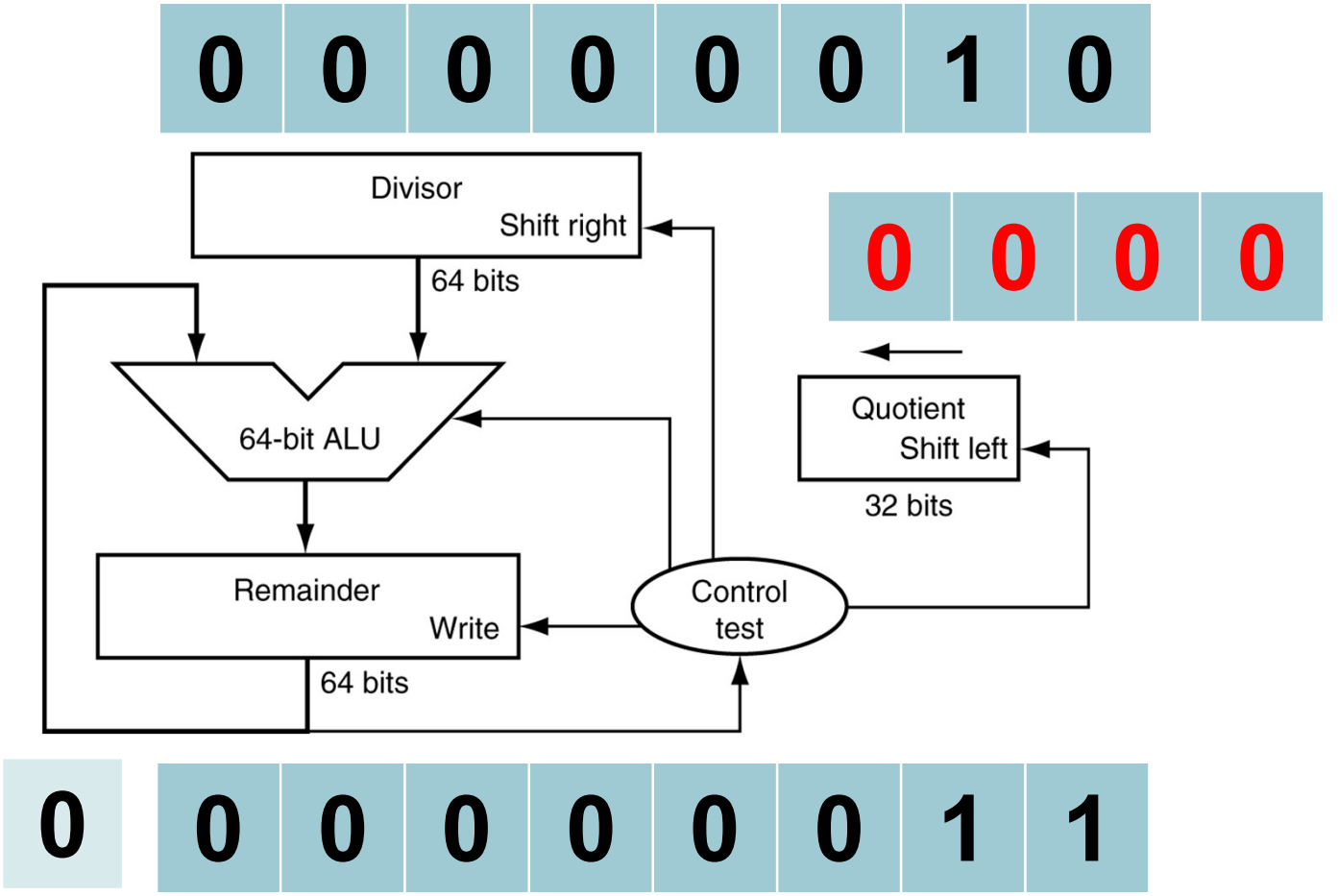
Integer Division Example

Step 4.4 除数右移1位



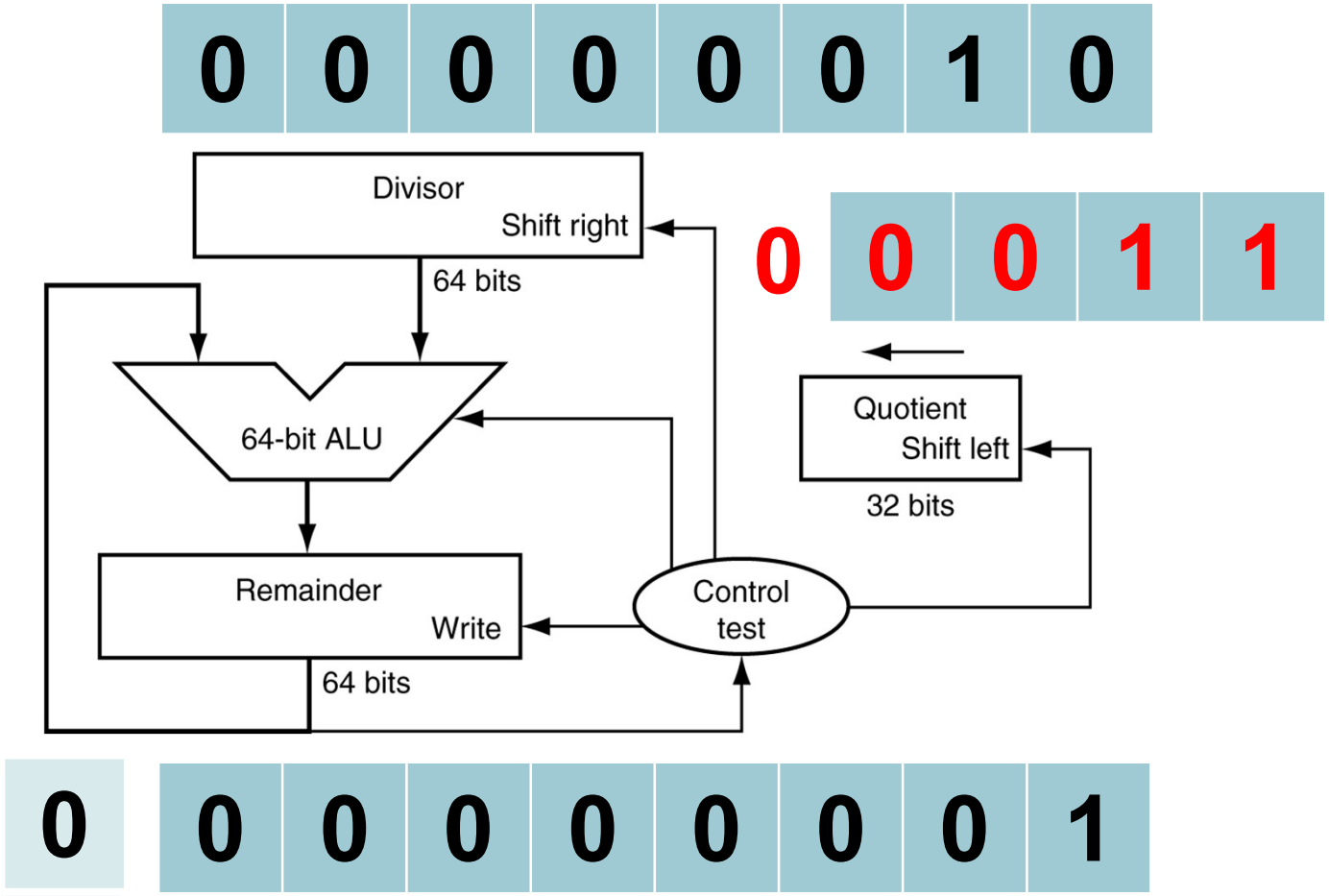
Integer Division Example

Step 5.1余数-除数 试商



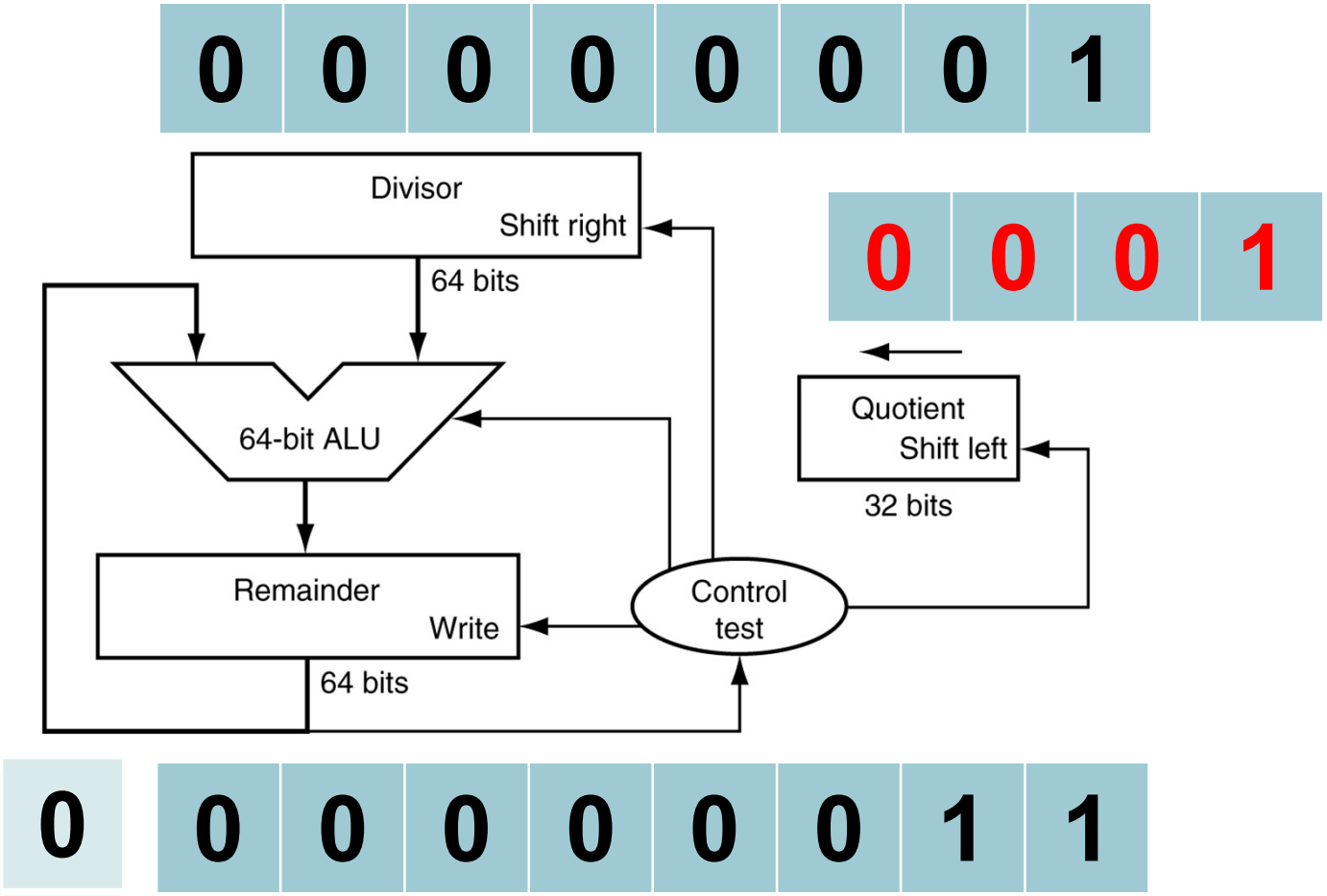
Integer Division Example

Step5.2 余数>0,商左移1位，置1



Integer Division Example

Step 5.3 除数右移1位(可选)



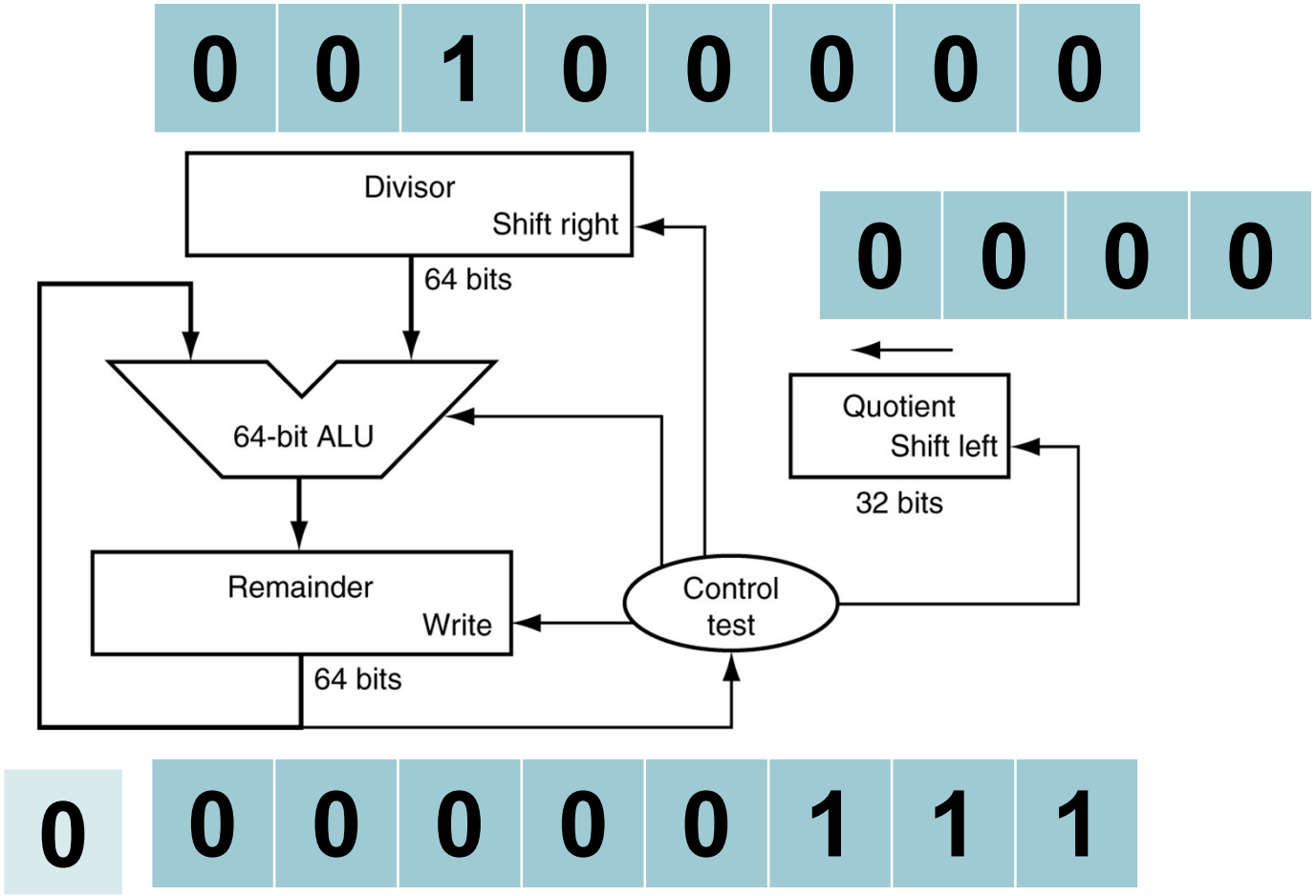
请同学们再观察一遍 |x|/|y|的过程

X= $[0111]_2$ Y=$[0010]_2$

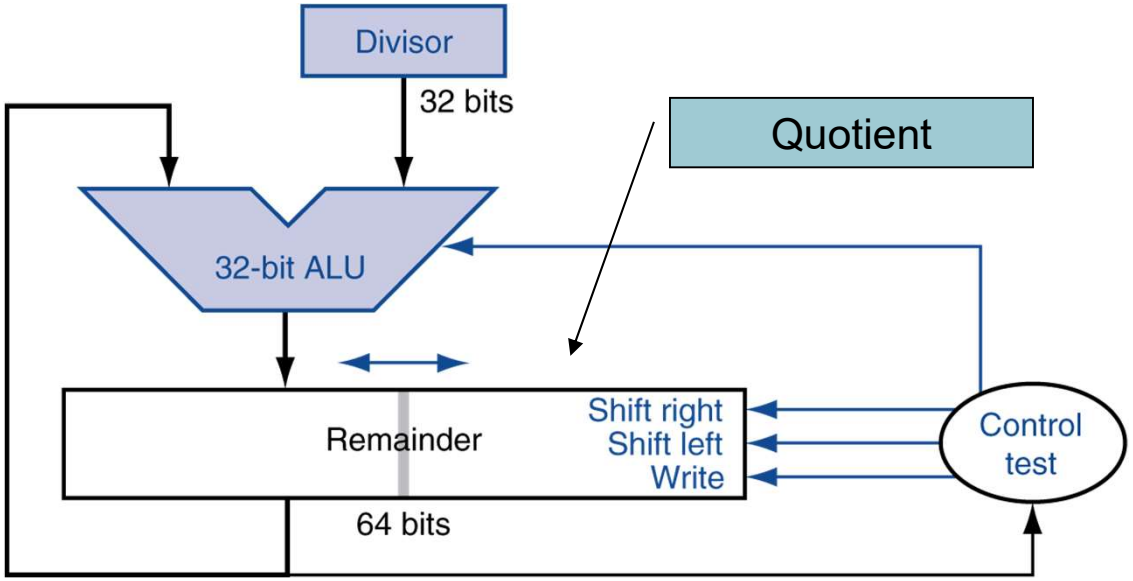
迭代次数	步骤	商	除数	余数
0	初始化	0000	**0010 0000**	**0** 0000 0111
1 **比较余数与 |Y| * 2^4**	余数=余数-除数 试商，余数<0 恢复余数 **商左移**1位，**最低位置**0 除数右移	0000 000**0** 000**0**	0010 0000 0010 0000 000**1** 0000	**1** 1110 0111 **0 0000 0111** 0 0000 0111
2	余数=余数-除数 试商，余数<0 恢复余数 **商左移**1位，**最低位置**0 除数右移	0000 00**0**0 00**0**0	**0001 0000** 0001 0000 0000 **1**000	**1** 1111 0111 **0 0000 0111** 0 0000 0111
3	余数=余数-除数 试商，余数<0 恢复余数 **商左移**1位，**最低位置**0 除数右移	0000 0**0**0**0** 0**0**0**0**	**0000 1000** 0000 1000 0000 0**1**00	**1** 1111 1111 **0 0000 0111** 0 0000 0111
4	余数=余数-除数 试商1，余数>=0 商左移1位，**最低置1** 除数右移	0000 **0001** **0001**	**0000 0100** 0000 0**1**00 0000 00**1**0	**0** 0000 0011 0 0000 0011 0 0000 0011
5 **比较余数与 |Y| * 2^0**	余数=余数-除数 试商1，余数>=0 商左移1位，**最低置1** 除数右移	0000 **0010** **0011**	**0000 0010** 0000 00**1**0 0000 0001	**0** 0000 0001 **0** 0000 0001 **0** 0000 0001

Integer Division

$|x| = [0111]_2 \quad |y| = [0010]_2$



Optimized Integer Divider

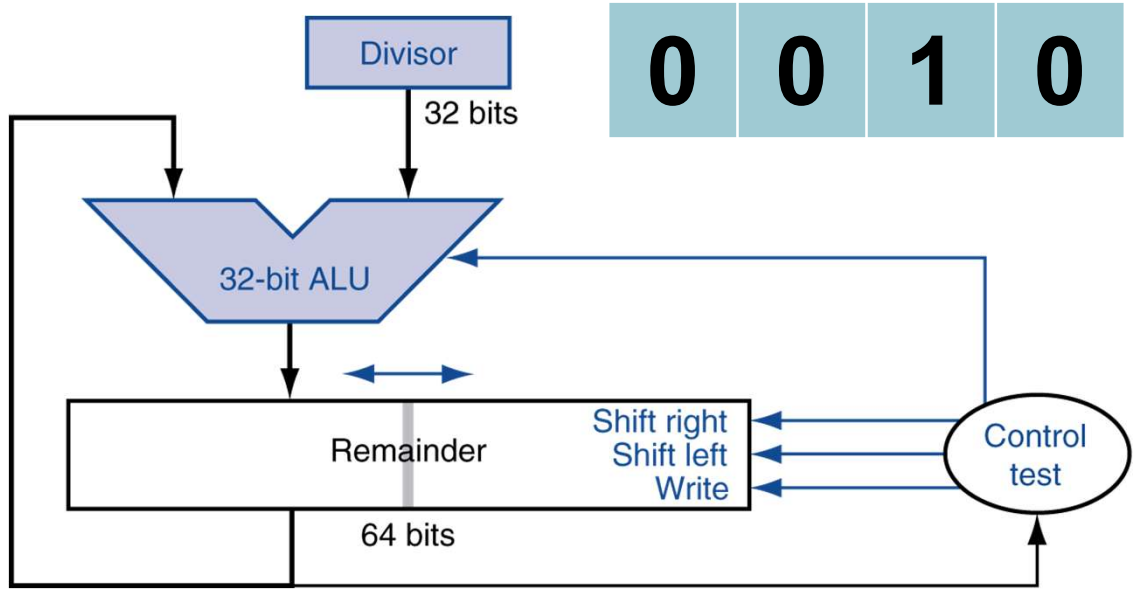


- One cycle per partial-remainder subtraction
- Looks a lot like a multiplier!
  - Same hardware can be used for both

Optimized Integer Divider

Step 0 初始化

被除数= $[0111]_2$　除数=$[0010]_2$

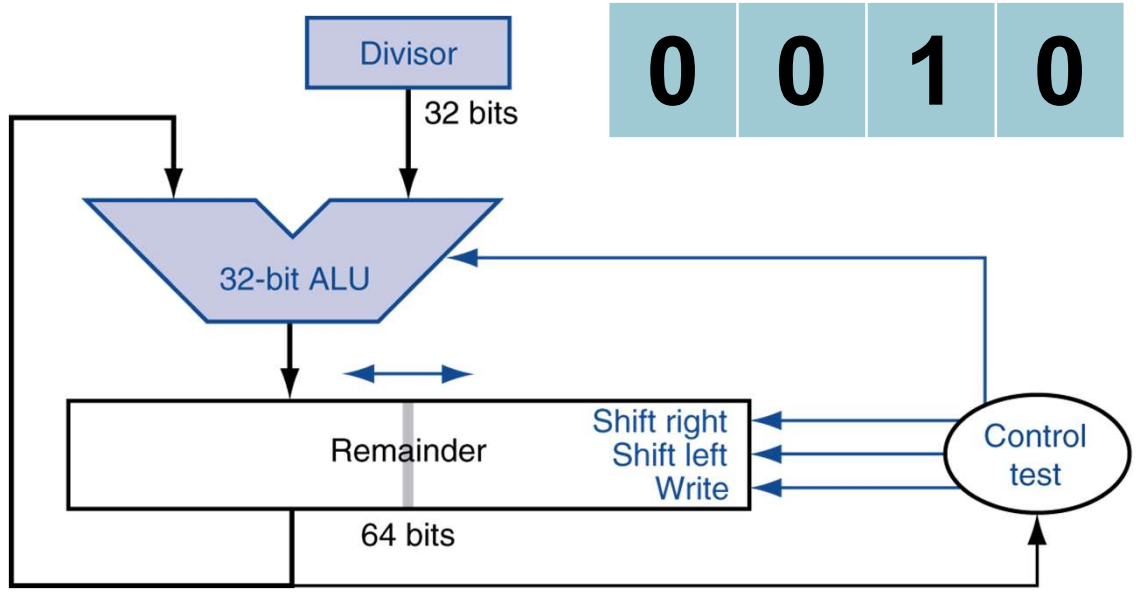


0	0	0	0	0	0	1	1	1

余数　商

Optimized Integer Divider

Step 1.1 余数=余数-除数 试商



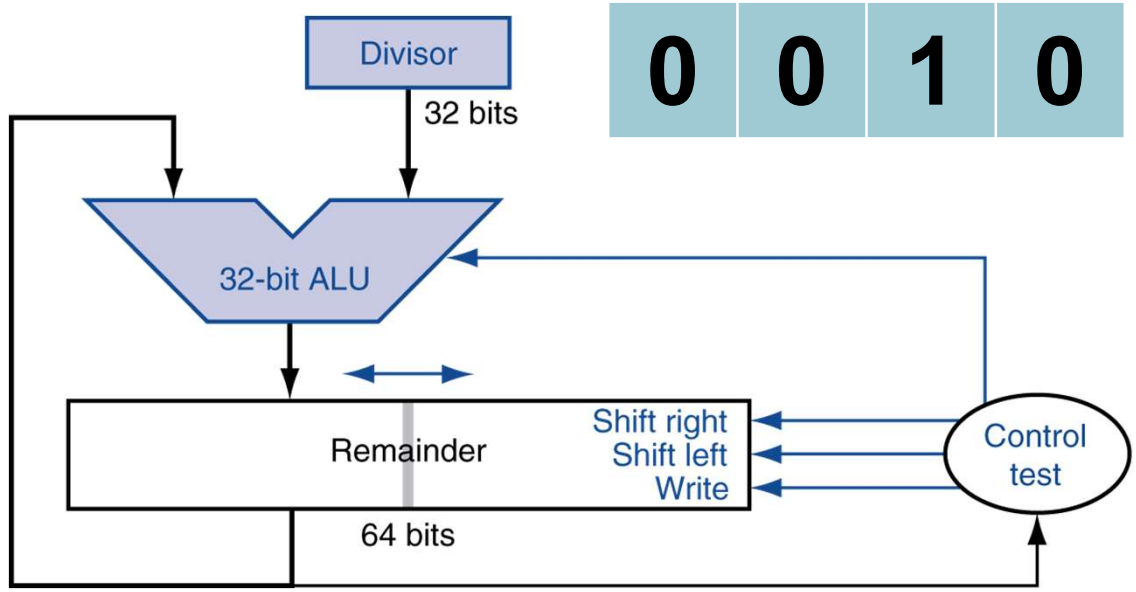
0	0	1	0

1	1	1	1	0	0	1	1	1

余数 | 商

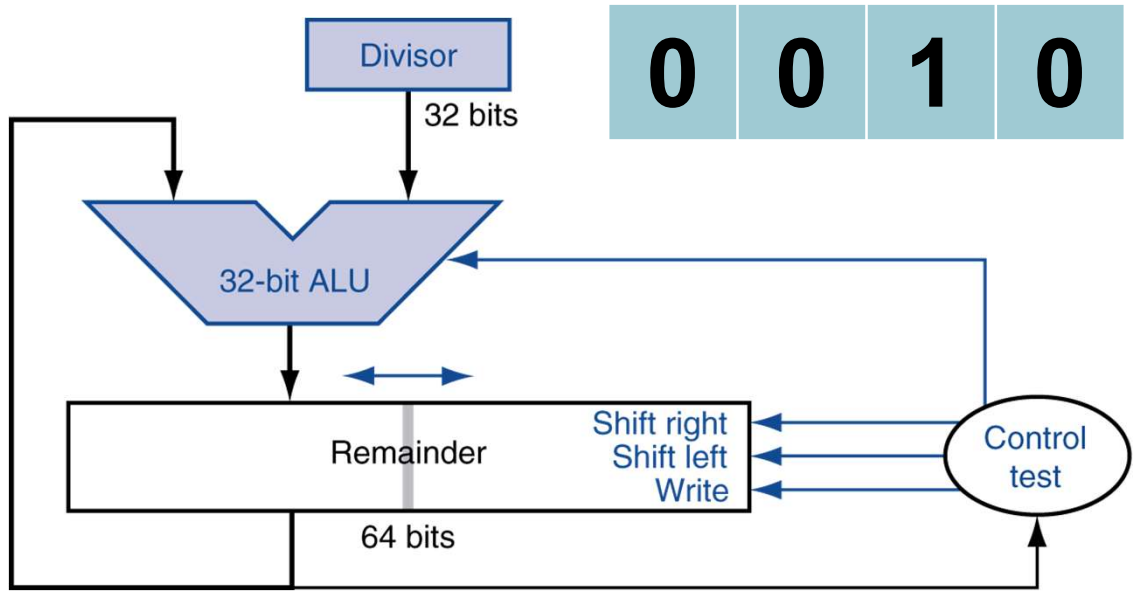
Optimized Integer Divider

Step 1.2 余数<0 恢复余数，余数和商 左移1位



Optimized Integer Divider

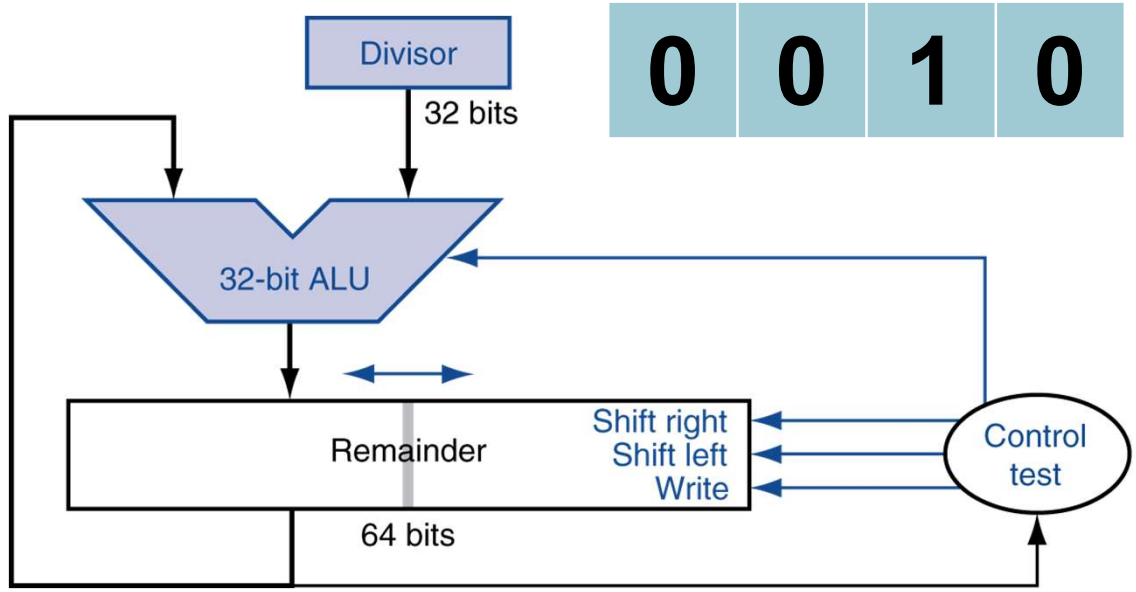
Step 1.3 商的 最低位置0



余数 商

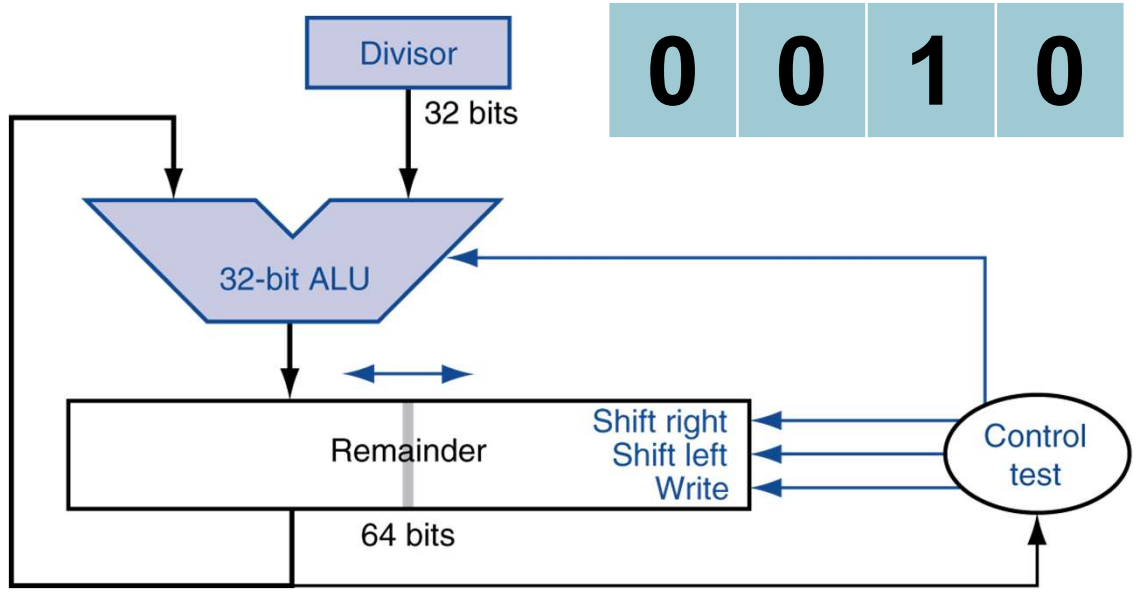
Optimized Integer Divider

Step 2.1 余数=余数-除数 试商



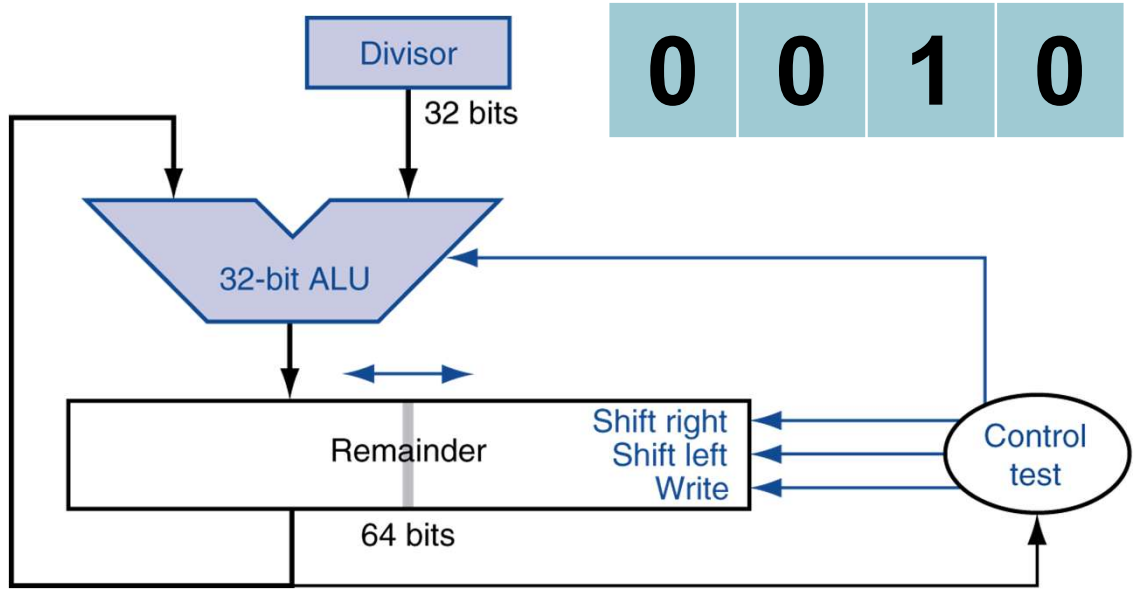
Optimized Integer Divider

Step 2.2 余数<0 恢复余数，余数和商 左移1位



Optimized Integer Divider

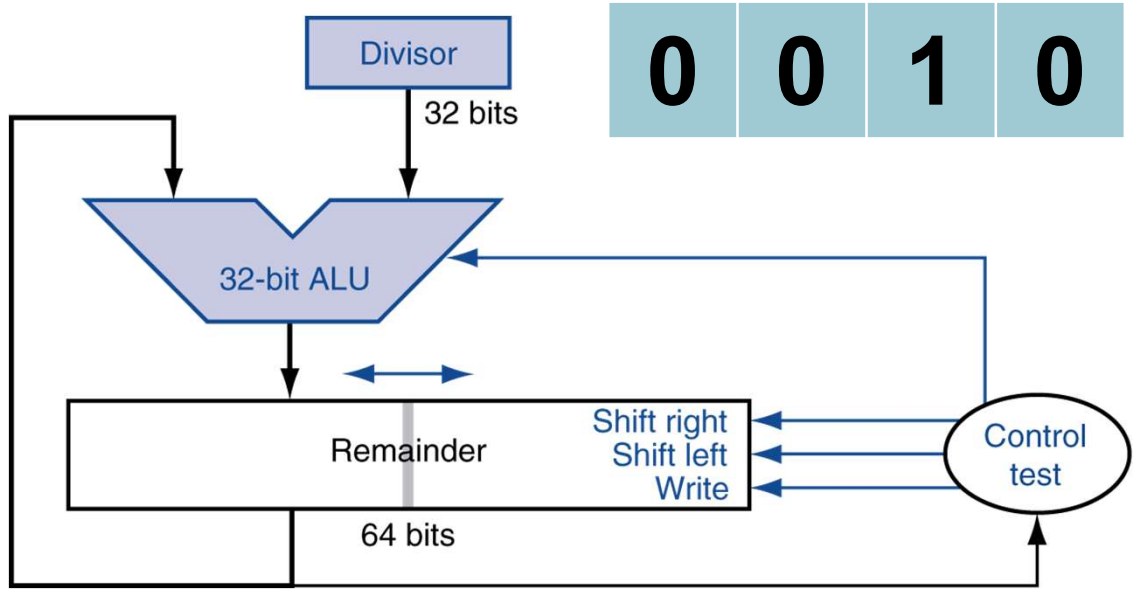
Step 2.3 商的 最低位置0



余数 商

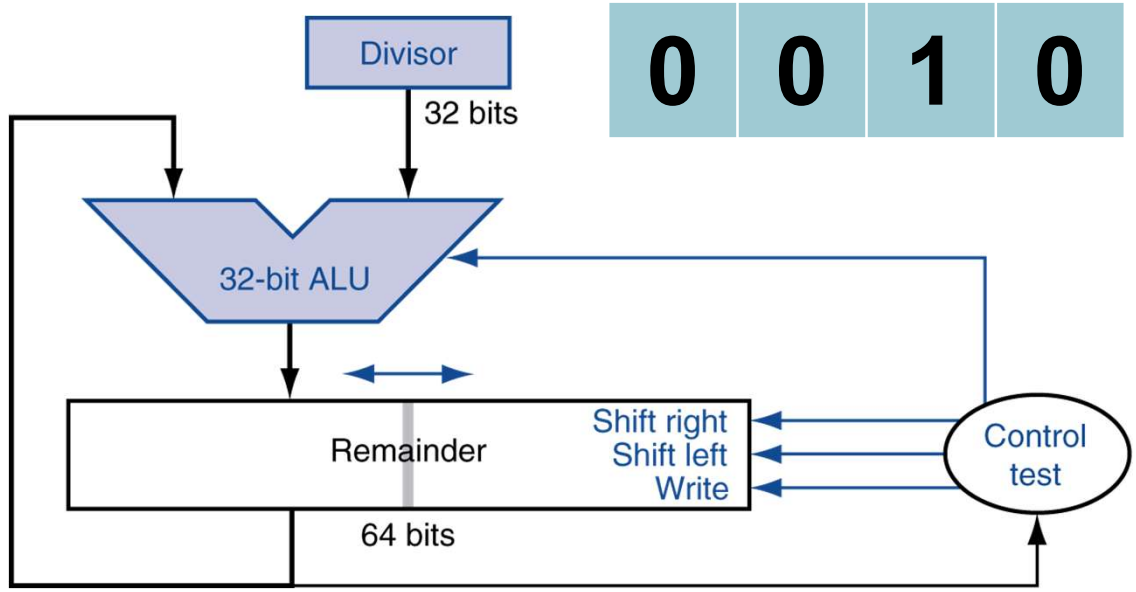
Optimized Integer Divider

Step 3.1 余数=余数-除数 试商



Optimized Integer Divider

Step 3.2余数<0 恢复余数，余数和商 左移1位

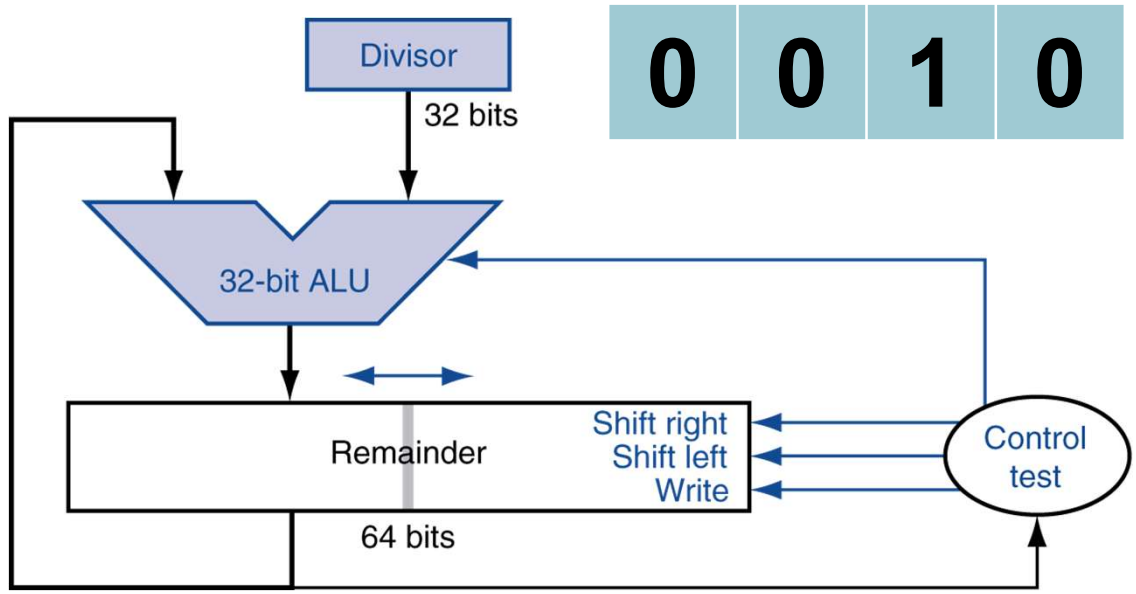


0	0	0	1	1	1	0	0	0

余数 | 商

Optimized Integer Divider

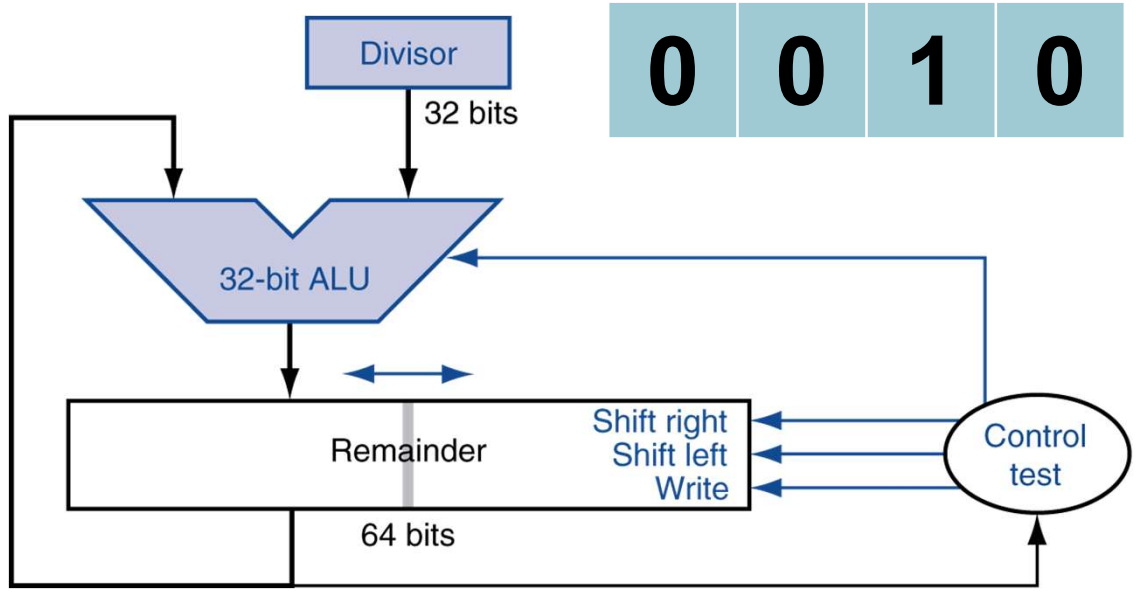
Step 3 余数最低位置0



余数 商

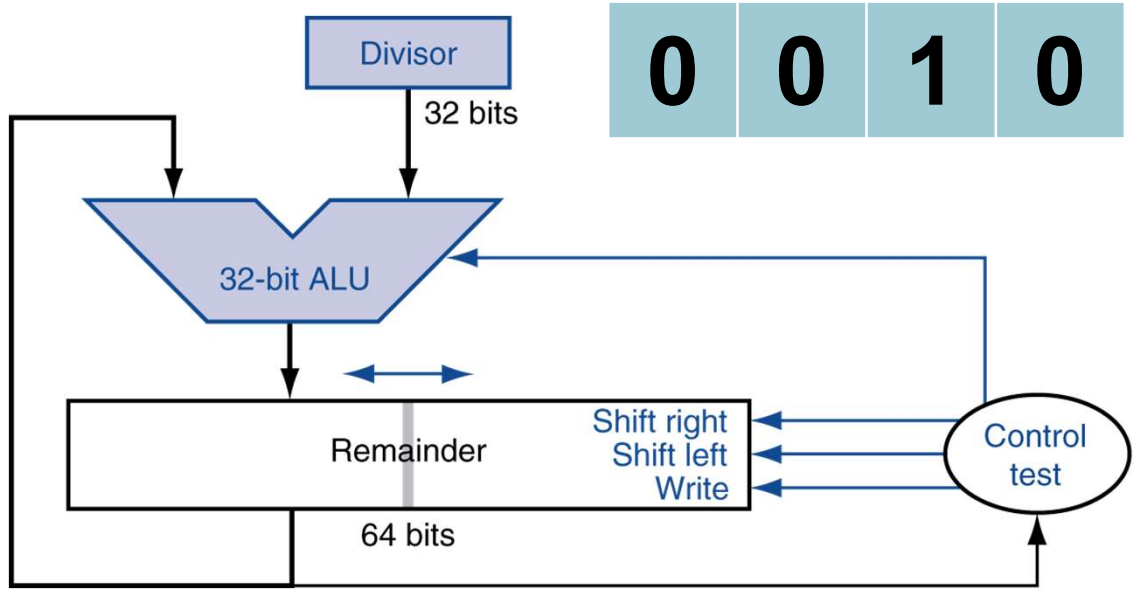
Optimized Integer Divider

Step 4.1 余数=余数-除数 试商



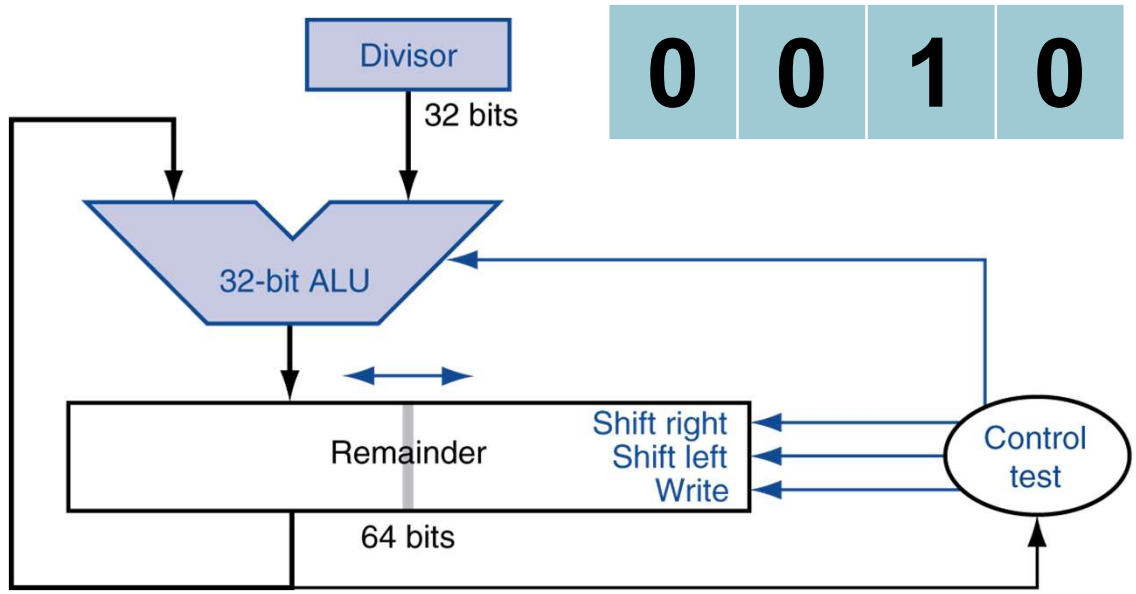
Optimized Integer Divider

Step 4.2 余数>0，余数-商左移1位



Optimized Integer Divider

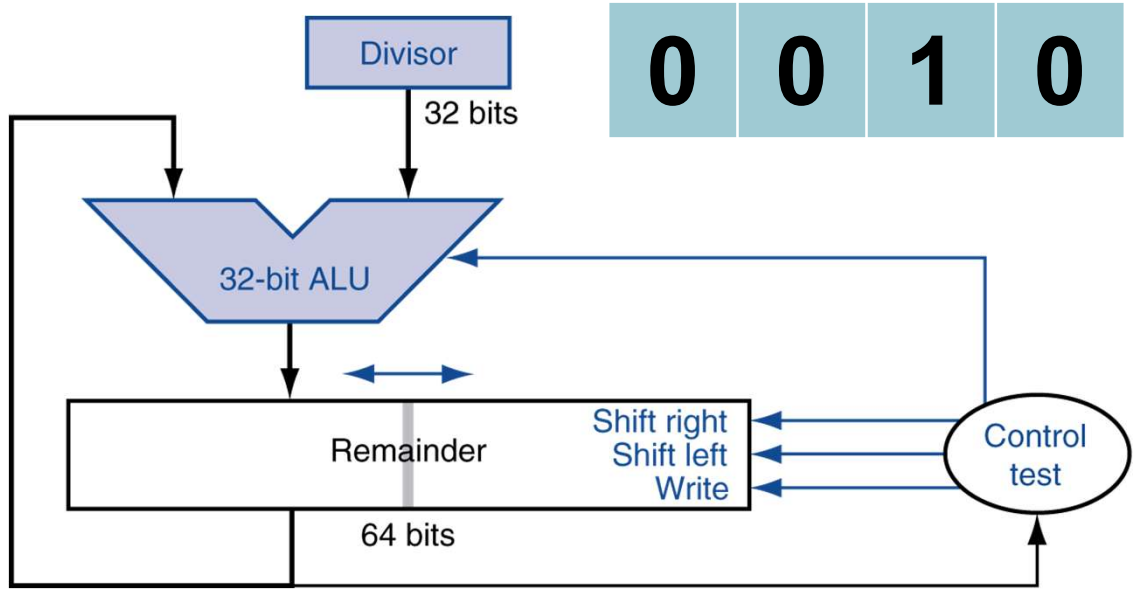
Step 4.3 余数最低位置1



余数 商

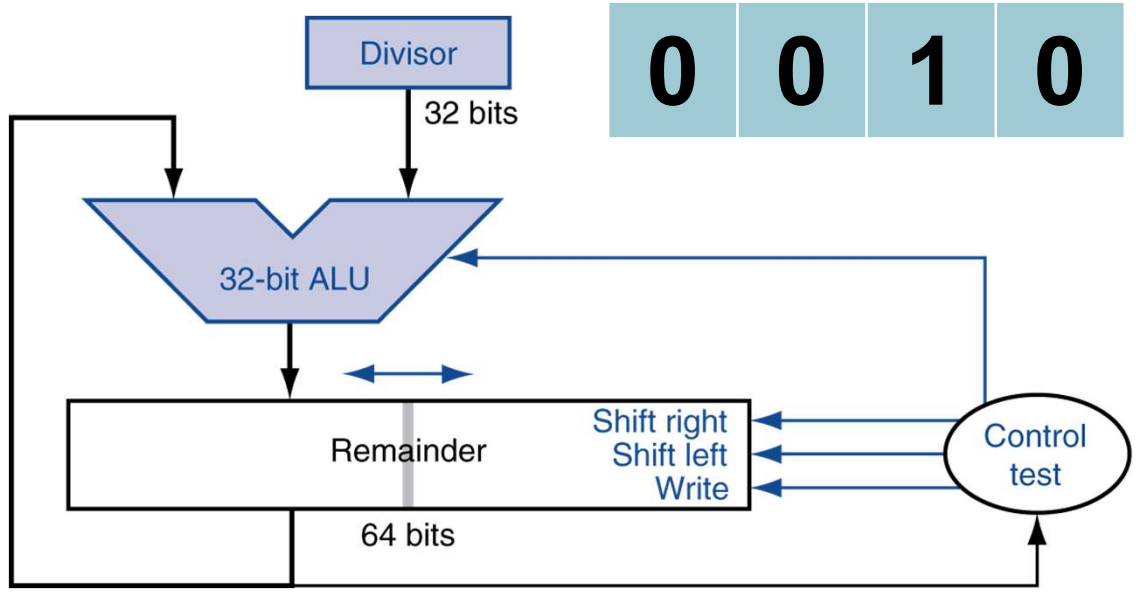
Optimized Integer Divider

Step 5.1 余数=余数-除数 试商



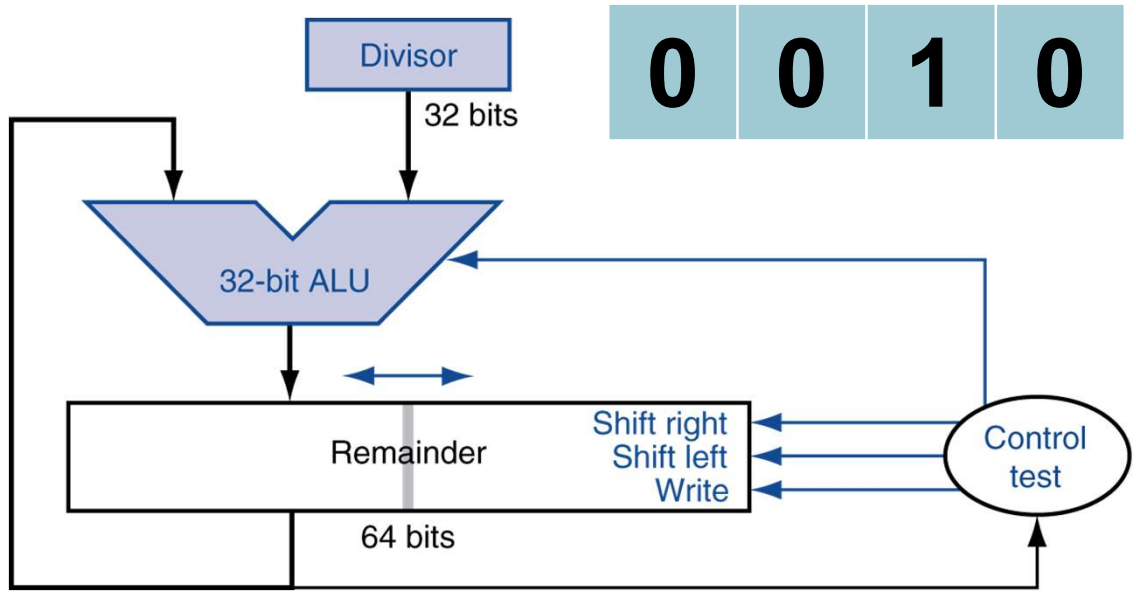
Optimized Integer Divider

Step 5.2 余数>0，余数-商左移1位



Optimized Integer Divider

Step 5.3 余数最低位置1



令X= $[0111]_2$ Y=$[0010]_2$ 优化的除法汇总

迭代次数	步骤	除数	余数 | 商
0	初始化	0010	0 0000 0111
1	余数=余数-除数 试商 余数<0 **恢复**余数并**左移**1位 **余数最低位置**0	0010	**1** 1110 **0111** 0 0000 1110 1 0000 111**0**
2	余数=余数-除数 试商 余数<0 **恢复**余数并**左移**1位 **余数最低位置**0	0010	**1** 1110 111**0** 0 0001 11**0**0 0 0001 11**00**
3	余数=余数-除数 试商 余数<0 **恢复**余数并**左移**1位 **余数最低位置**0	0010	**1** 1111 111**0** 0 0011 1**00**0 0 0011 1**000**
4	余数=余数-除数 试商 余数>0，余数-商**左移**1位 **余数最低位置**1	0010	0 0001 1**000** 0 0011 **000**0 0 0011 **0001**
5	余数=余数-除数 试商 余数>0，余数-商左移1位 **余数最低位置**1	0010	0 0001 **0001** 0 0001 **001**0 0 001**0 0011**

商为= [0011] 余数为=$[0000\ 0001]_2$

原码1位除法-加减交替除法

实际上常用的是加减交替法，其运算规则：

- **商的符号**单独处理 $q_f = x_f \oplus y_f$
- 试商运算并左移
  - 当余数为正时，商1，余数左移一位，减除数；
  - 当余数为负时，商0，余数左移一位，加除数；
- 上述步骤重复n+1次（n位尾数，1位为符号位）得到商的绝对值，最后一步余数不左移
- 最后一步余数为负值时，需要加上|y|得到正确的余数

特点：计算步数固定，控制简单。

加减交替除法正确性说明

对于**恢复余数**法，假设第i次余数为R_i

- 如果R_i为正数，那么R_{i+1}的值为：

  $R_{i+1}=2R_i-|y|$

- 如果Ri为负数,那么R_{i+1}的值为：

  恢复余数法：先恢复余数然后，左移1位，再减去|y|

  $R_{i+1}=2(R_i+|y|)-|y|=2R_i+|y|$

  加减交替法：余数左移加上|y|

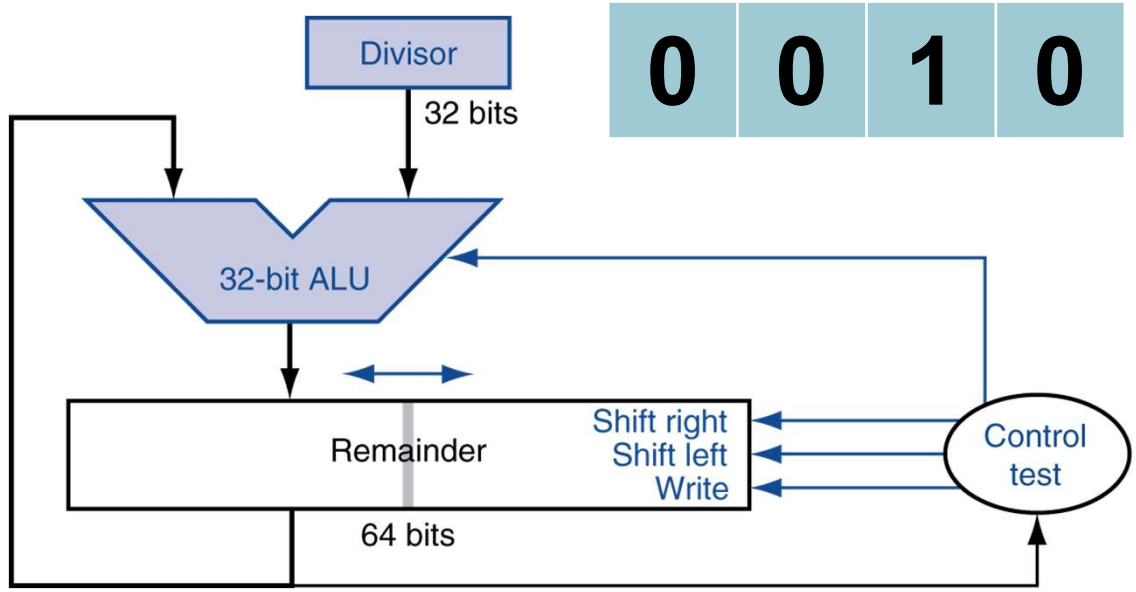
  $R_{i+1}=2R_i+|y|$

因此这两种算法计算结果是一样的！

加减交替法

Step 0 初始化

被除数= $[0111]_2$　除数=$[0010]_2$

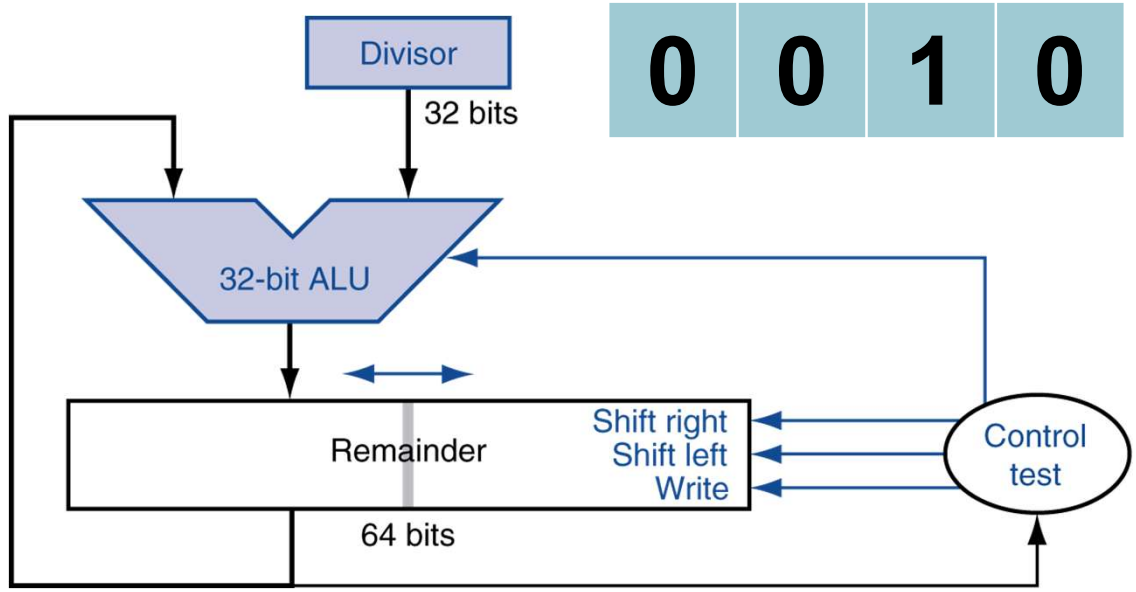


0	0	0	0	0	0	1	1	1

余数　商

加减交替法

Step 1.1 余数>0, 余数-除数 试商

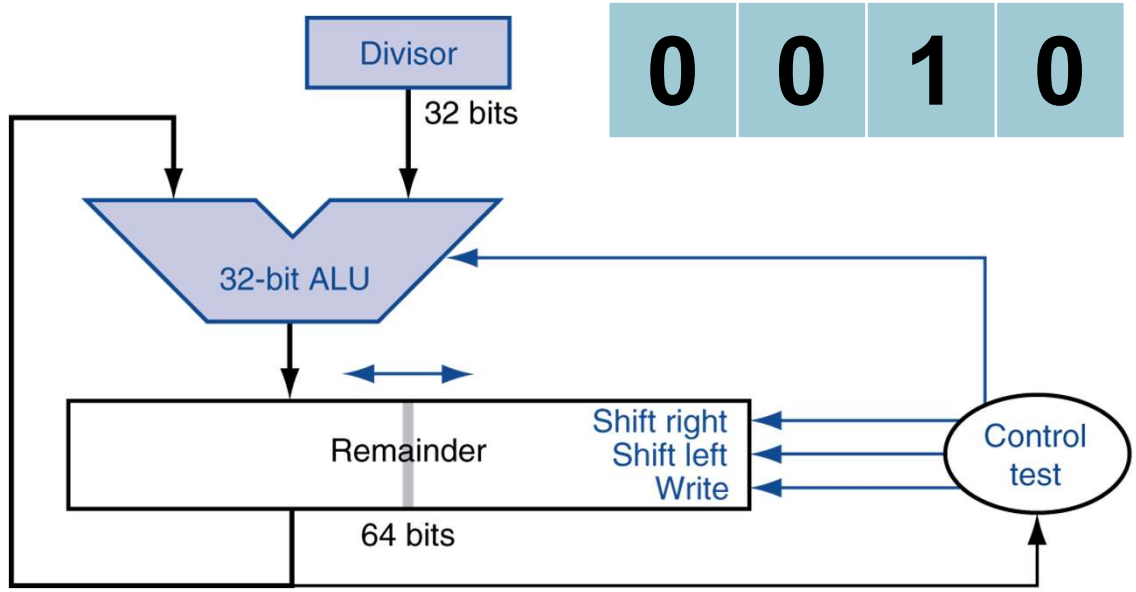


1	1	1	1	0	0	1	1	1

余数 | 商

加减交替法

Step 1.2 余数和商 左移1位



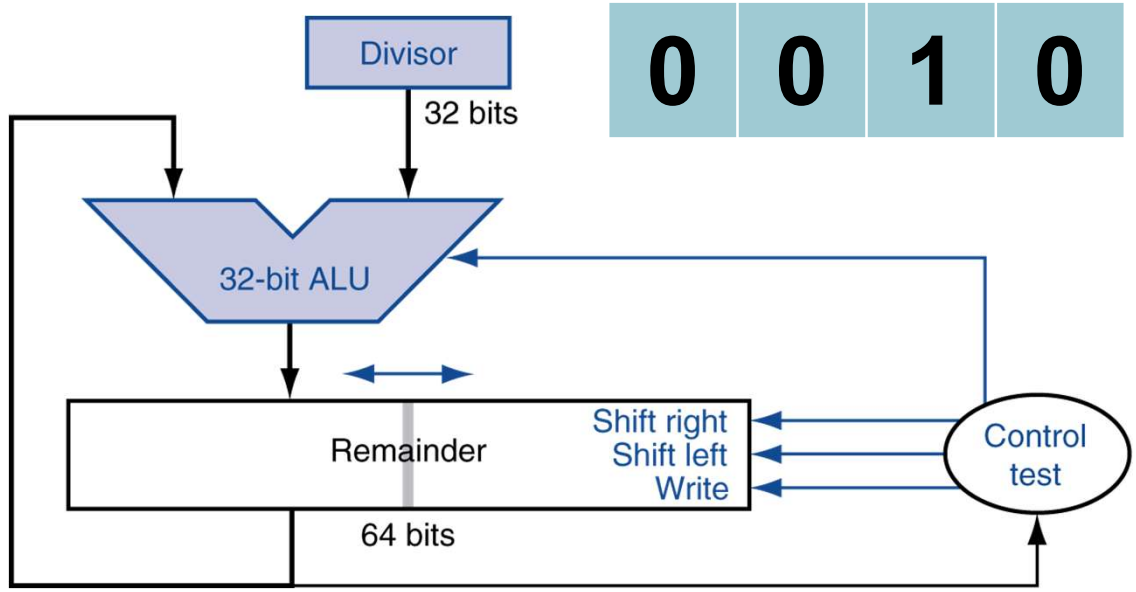
0	0	1	0

1	1	1	0	0	1	1	1	0

余数 | 商

加减交替法

Step 1.3 余数<0，商的 最低位置0



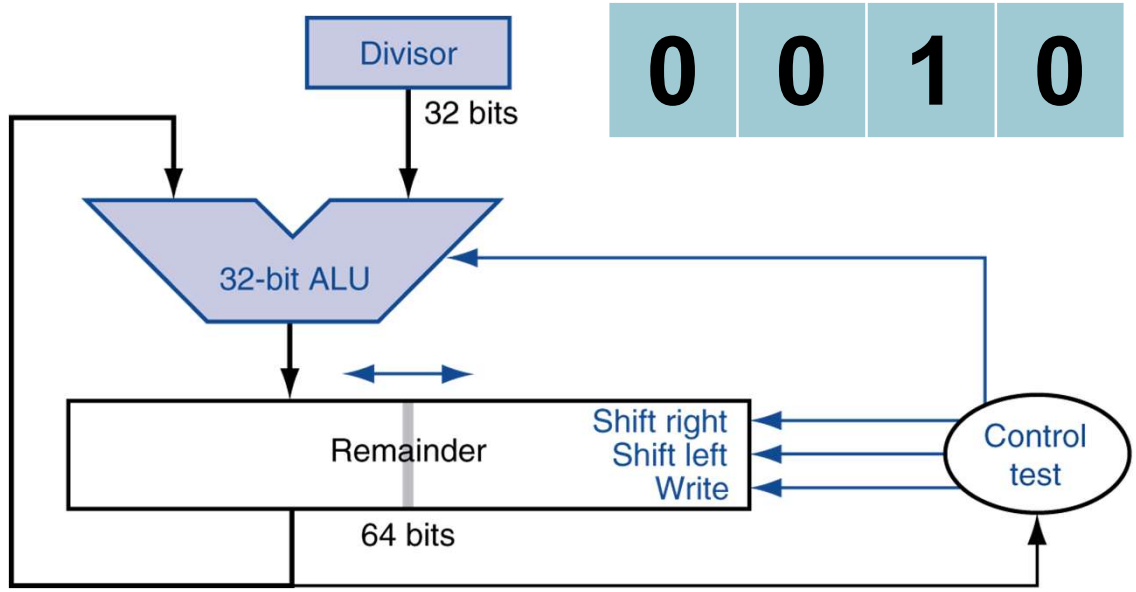
0	0	1	0

1	1	1	0	0	1	1	1	0

余数 | 商

加减交替法

Step 2.1 余数<0, 余数+除数 试商



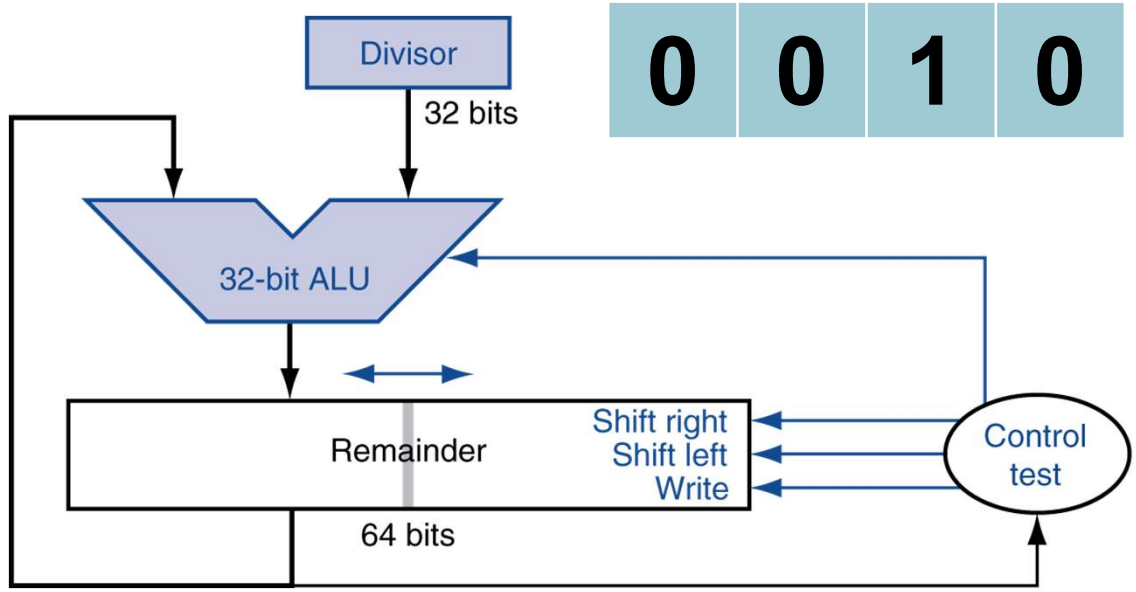
0	0	1	0

1	1	1	1	0	1	1	1	0

余数 | 商

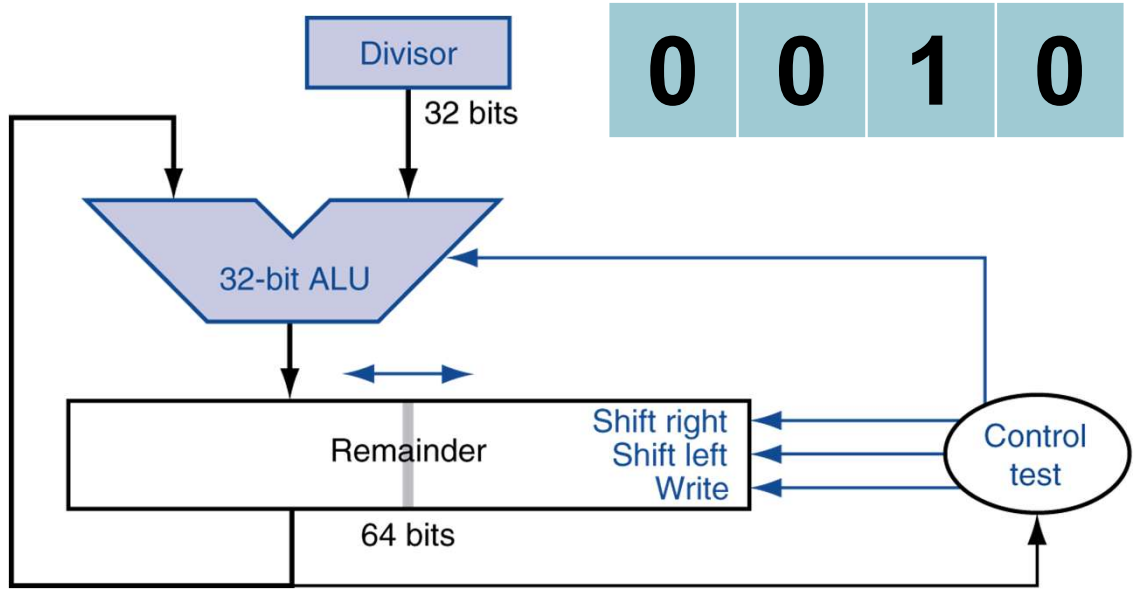
加减交替法

Step 2.2 余数和商 左移1位



加减交替法

Step 2.3 余数<0，商的 最低位置0

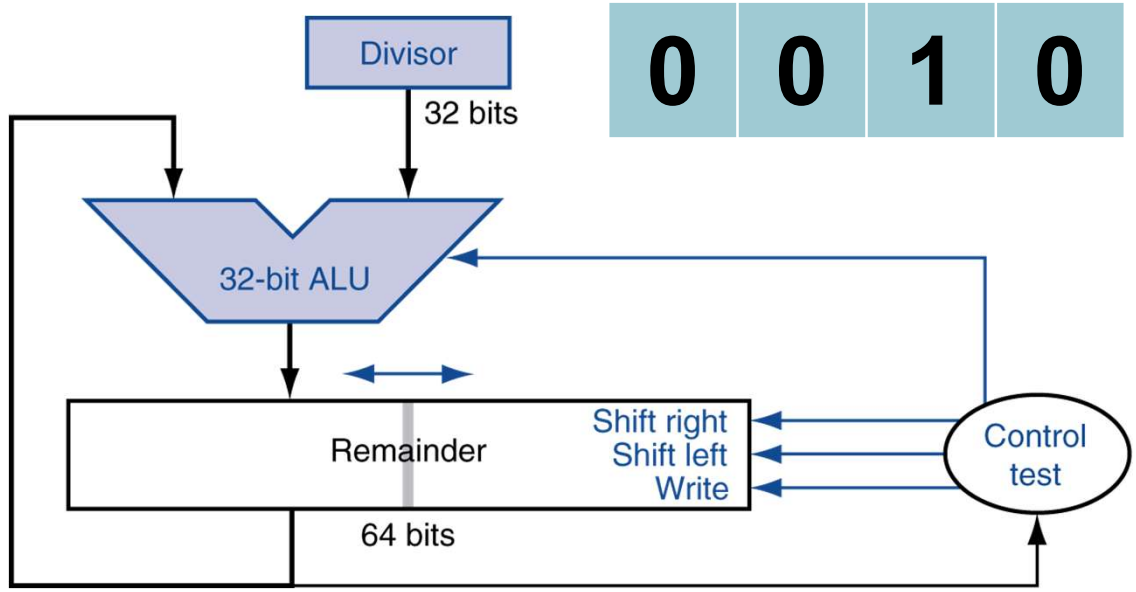


1	1	1	0	1	1	1	0	0

余数 | 商

加减交替法

Step 3.1 余数<0, 余数+除数 试商



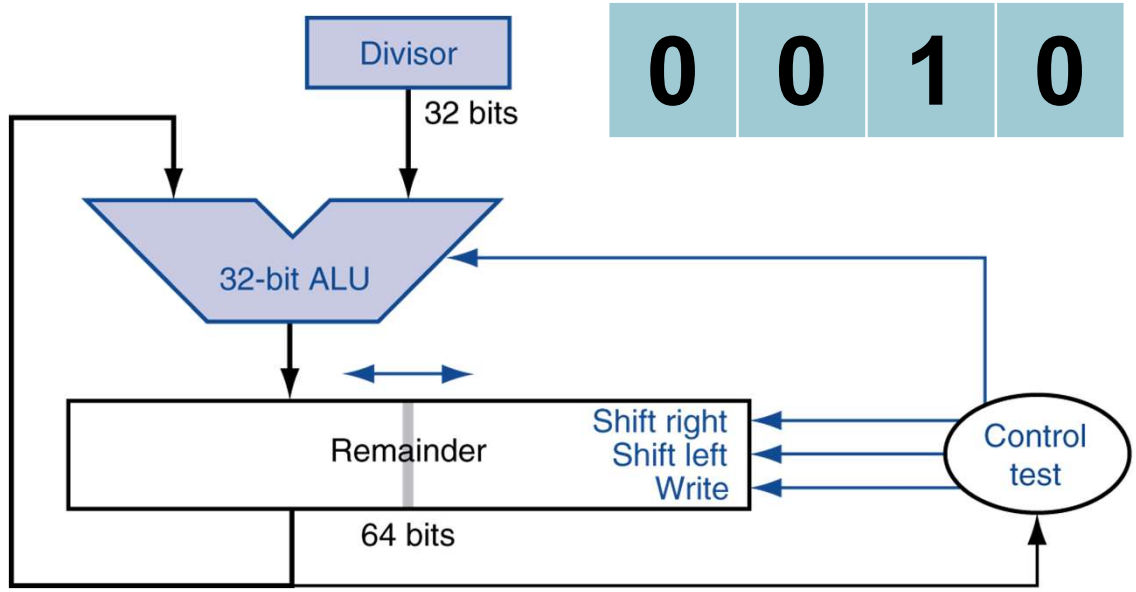
0	0	1	0

1	1	1	1	1	1	1	0	0

余数 | 商

加减交替法

Step 3.2 余数和商 左移1位



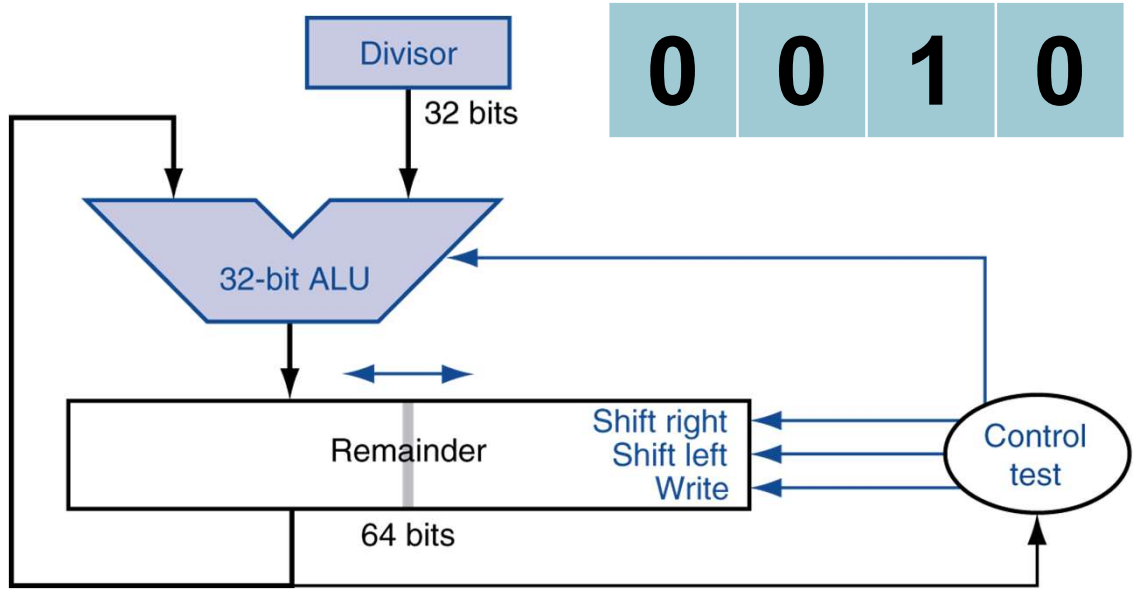
0 0 1 0

1 1 1 1 1 1 0 0 0

余数 商

加减交替法

Step 3.3 余数<0，商的 最低位置0

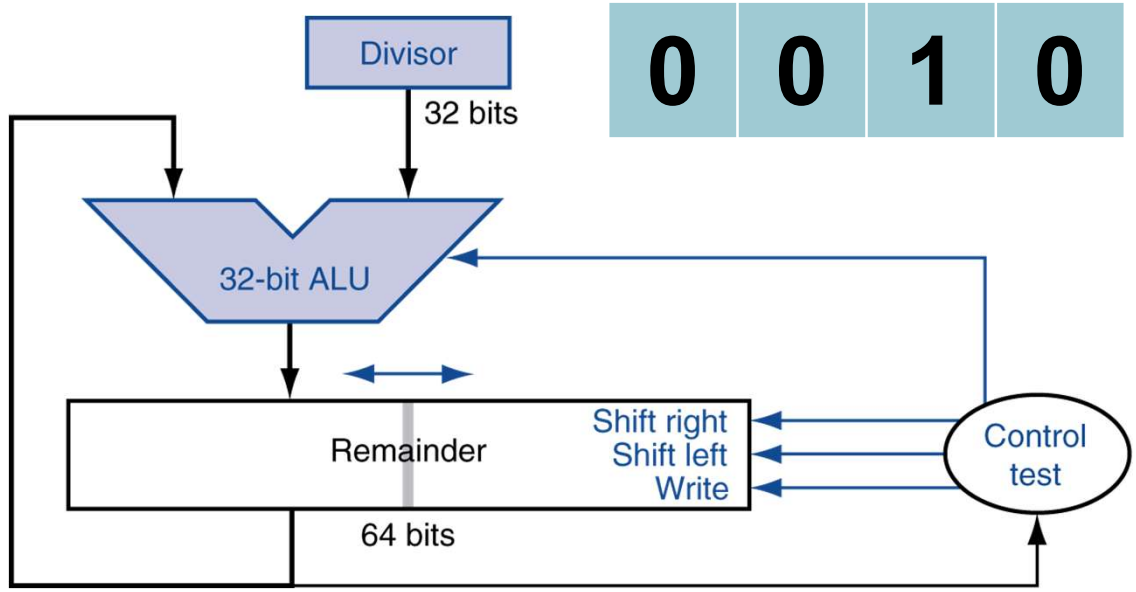


1	1	1	1	1	1	0	0	0

余数 | 商

加减交替法

Step 4.1 余数<0, 余数+除数 试商

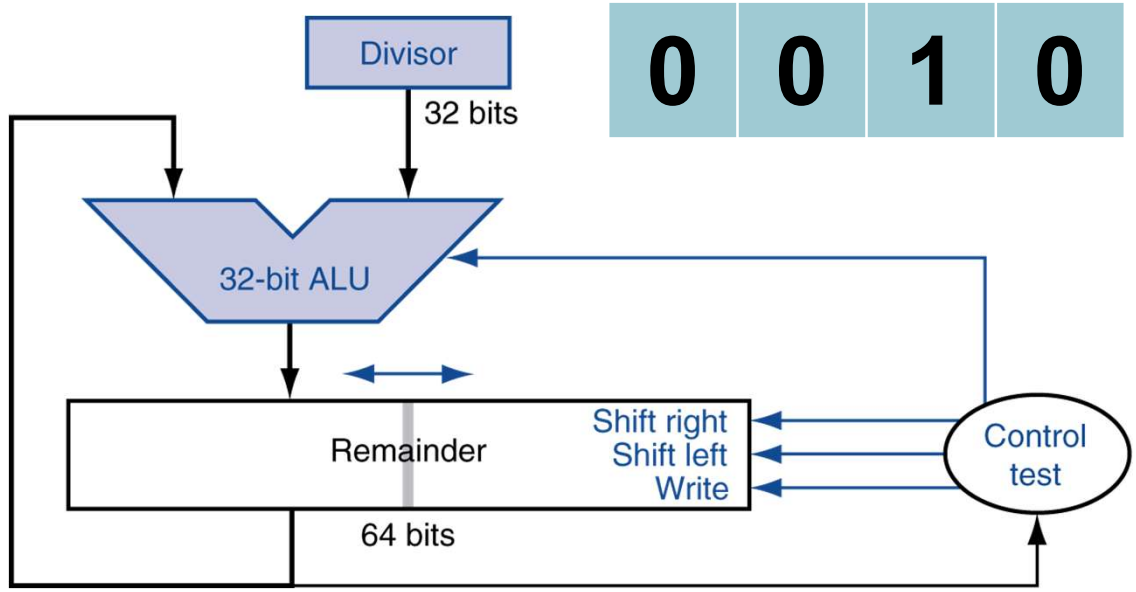


0	0	0	0	1	1	0	0	0

余数 | 商

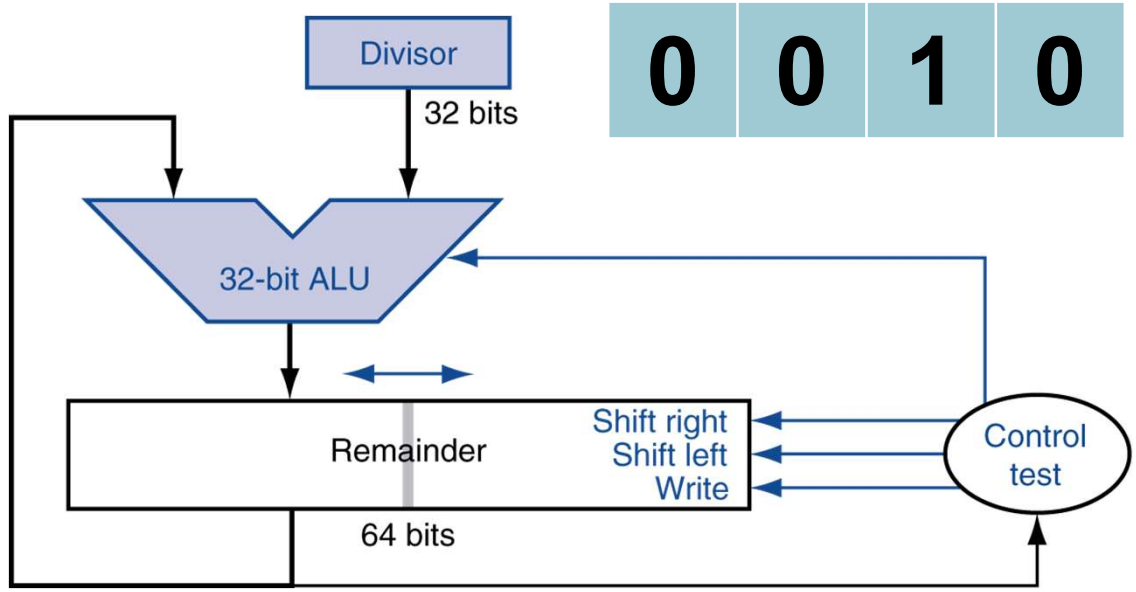
加减交替法

Step 4.2 余数和商 左移1位



加减交替法

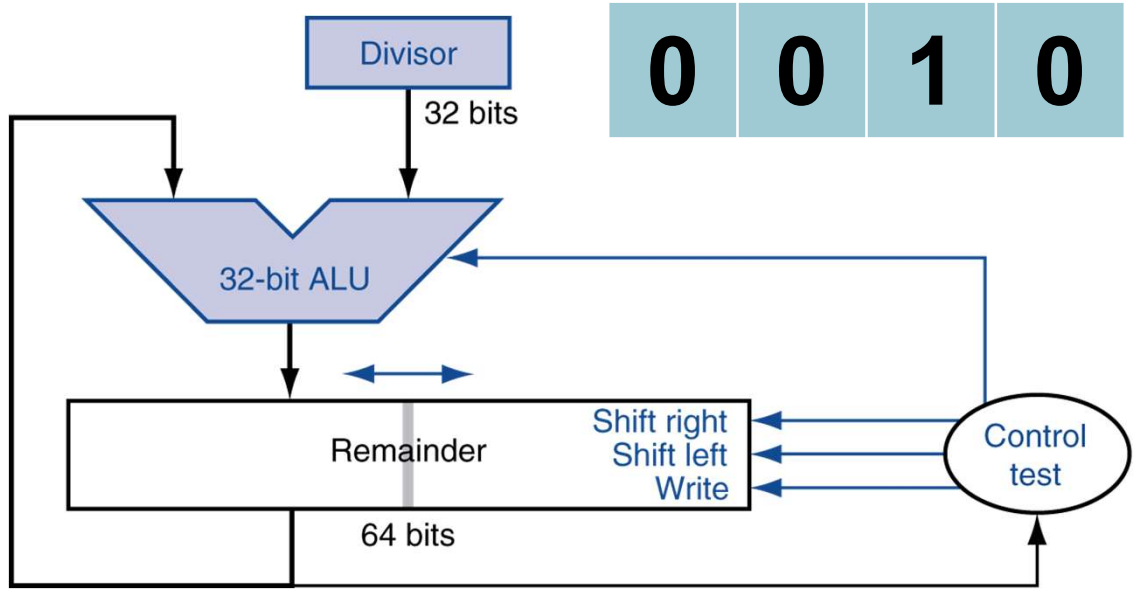
Step 4.3 余数>=0，商的 最低位置1



余数 商

加减交替法

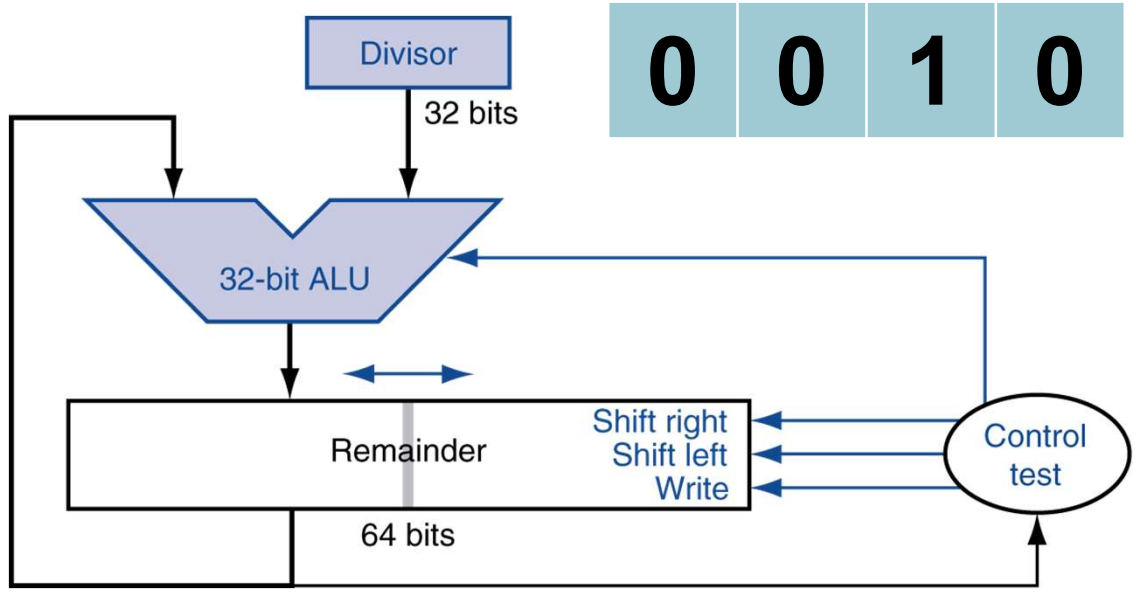
Step 5.1 余数>0, 余数-除数 试商



余数 | 商

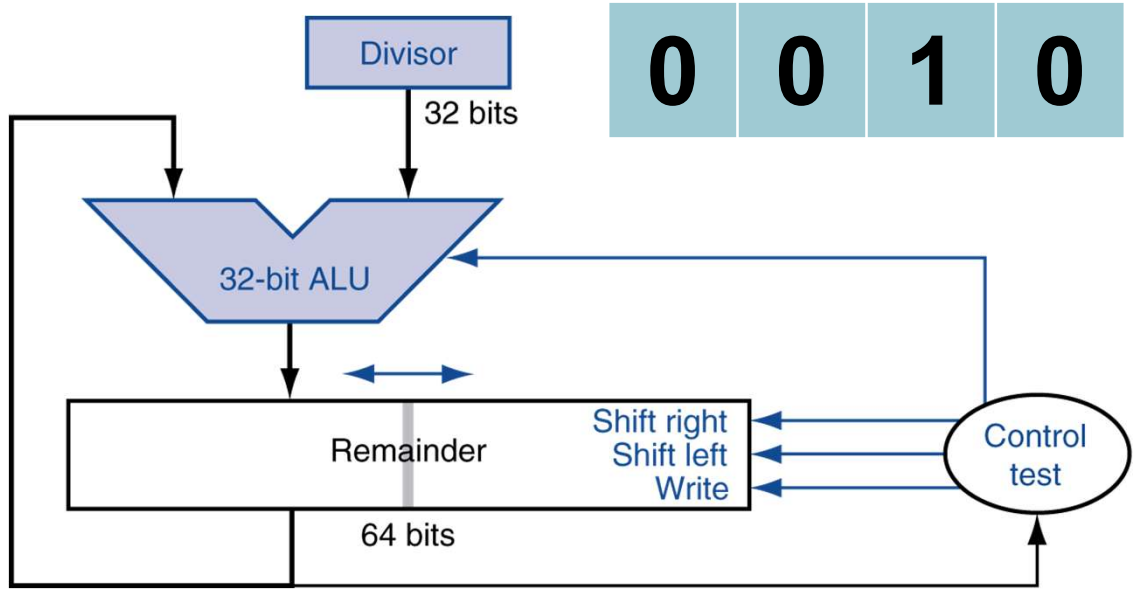
加减交替法

Step 5.2 余数和商 左移1位



加减交替法

Step 5.3 余数>=0，商的 最低位置1



令x= $[0111]_2$ y=$[0010]_2$　加减交替计算过程

迭代次数	步骤	除数	余数 | 商
0	初始化	0010	0 0000 0111
1	当前余数为正，余数=余数-除数 余数<0 ，左移1位 **余数最低位置**0	0010	**1** **1**110 **0111** 1 1100 1110 1 1100 111**0**
2	当前余数<0,余数=余数+除数 余数并左移1位 **余数最低位置**0	0010	**1** **1**110 111**0** 1 1101 11**0**0 1 1101 11**00**
3	当前余数<0，余数=余数+除数 余数<0 **恢复**余数并左移1位 **余数最低位置**0	0010	**1** **1**111 11**00** **1** **1**111 1**00**0 **1** **1**111 1**000**
4	当前余数<0余数=余数+除数 余数>0，余数左移1位 **余数最低位置**1	0010	0 0001 1**000** 0 0011 **000**0 0 0011 **0001**
5	当前余数>0,余数=余数-除数 余数>0，余数左移1位 **余数最低位置**1	0010	0 0001 **0001** 0 001**0** **001**0 0 001**0** **0011**

商为= [0011] 余数为=$[0000\ 0001]_2$

3.4 Division

- 十进制整数除法回顾
- 二进制整数除法
- 原码1位除法思想
- 恢复余数除法及其优化
- 加减交替除法
- 定点小数除法
- 快速除法概况

定点小数原码1位除法(加减交替)

算法：小数求商和余数算法

输入：n位的二进制小数被除数X和除数Y

输出：商Q和余数R

step 0:　初始化，R=X, t=0

step 1. if R>0　R=R-X　else　R=R+X

step 2. left_shift(R), left_shift(Q), t=t+1

step 3　if R>=0 Q(n)=1　else Q(n)=0

step 4　　if t <n goto　step 1;

step 5　　输出 R, Q

定点小数原码1位除法

算法推广到定点小数时需要注意：

- 被除数后面补零扩展n位

  与整数除法一项，被除数的有效值保持不变。

- 参与运算的小数必须是规格化小数

  绝对值>=0.5，原码表示的小数点后第一位为1

- 加法器运算时采用双符号位

  这是运算结果可以是[-2,2) 的值。

定点小数除法采用双符号位补码（模4补码）

定点小数的补码采用2个符号位来表示数字的符号

例如：00.1101 或者11.1010

如果其表示范围为[-1,1）最终的结果为 10 或则 01，则表示其结果存在溢出。

如果采用双符号位，计算结果都是正确的无需特别处理。

加减交替法推广到定点小数的除法运算

$[x]_{原}$= 0 0 .1001,
$[y]_{补}$ = 00.1011,
$[-y]_{补}$=11.0101

被除数x / 余数 r	商q	说明
0 0.1 0 0 1		
+$[-y]_{补}$ 1 1. 0 1 0 1	0.0000	x减y
1 1.1 1 1 0		余数 r_0<0，
← 1 1.1 1 0 0	0.0000	r和q左移一位， 商0
+$[y]_{补}$ 0 0. 1 0 1 1		加y
0 0. 0 1 1 1		余数r_1>0
← 0 0. 1 1 1 0	0.0001	r和q左移一位，商1
+$[-y]_{补}$ 1 1. 0 1 0 1		减y
0 0. 0 0 1 1		余数r_2>0
← 0 0. 0 1 1 0	0.0011	r和q左移一位,商1
+$[-y]_{补}$ 1 1. 0 1 0 1		减y
1 1. 1 0 1 1		余数r_3<0
← 1 1. 0 1 1 0	0.0110	商0，r和q左移一位
+$[y]_{补}$ 0 0. 1 0 1 1		加y
0 0. 0 0 0 1		余数r_4>0
	0.1101	商1，仅q左移一位

得： q = x/y =0.1101　　余数 r = 2^{-4} r_4 = 0.00000001

Faster Division

- 除法运算无法项乘法一样并行化
  - 因为减法操作需要根据余数的符号位条件执行
- 快速除法器 SRT division
  - 通过查表的方式来获取多位（2/3/4）的商
    - 在后续的步骤中矫正错误取值
  - 需要多次迭代
- 其它快速除法器
  - 函数迭代算法(Newton-Raphson等 近似计算)

Atkins, Daniel. (1968). Higher-Radix Division Using Estimates of the Divisor and Partial Remainders. Computers, IEEE Transactions on. C-17. 925- 934. 10.1109/TC.1968.226439.
王县,倪晓强,邢座程，浮点除法算法的分析与研究，第十二属计算机工程与工艺学术年会，2008

MIPS Division

- Use HI/LO registers for result
  - HI: 32-bit remainder
  - LO: 32-bit quotient
- Instructions
  - `div rs, rt` / `divu rs, rt`
  - Use `mfhi`, `mflo` to access result
  - No overflow or divide-by-0 checking
    - Software must perform checks if required

主观题 10分

设置

$[X]_{原}=0.1001$

$[Y]_{原}=0.1101$

请采用加减交替法计算$[X/Y]_{原}$的结果

15分钟完成，选择适当的算法需要给出具体的步骤

得： q = x/y =0.1011　　余数 r = 0.00000001

正常使用主观题需2.0以上版本雨课堂

作答

参考答案 x=[0.1001] y= [0.1101],计算x/y的值

$[x]_{原}$= 00.1001,
$[y]_{补}$ = 00.1101,
$[-y]_{补}$=11.0011

被除数x / 余数 r	商q	说明
0 0.1 0 0 1		
$+[-y]_{补}$ 1 1. 0 0 1 1	0.0000	x减y
1 1.1 1 0 0		余数 r_0<0,
← 1 1.1 0 0 0	0.0000	r和q左移一位， 商0
$+[y]_{补}$ 0 0. 1 1 0 1		加y
0 0. 0 1 0 1		余数r_1>0
← 0 0. 1 0 1 0	0.0001	r和q左移一位，商1
$+[-y]_{补}$ 1 1. 0 0 1 1		减y
1 1. 1 1 0 1		余数r_2<0
← 1 1. 1 0 1 0	0.0010	r和q左移一位,商0
$+[y]_{补}$ 0 0. 1 1 0 1		减y
0 0. 0 1 1 1		余数r_3>0
← 0 0. 1 1 1 0	0.0101	商1，r和q左移一位
$+[-y]_{补}$ 1 1. 0 0 1 1		加y
0 0. 0 0 0 1		余数r_4>0
	0.1011	商1，仅q左移一位

得： q = x/y =0.1011　　余数 r = 2^{-4} r_4 = 0.00000001