

Computer Organization and Design

Arithmetic for Computers Part II Multiplication & Division

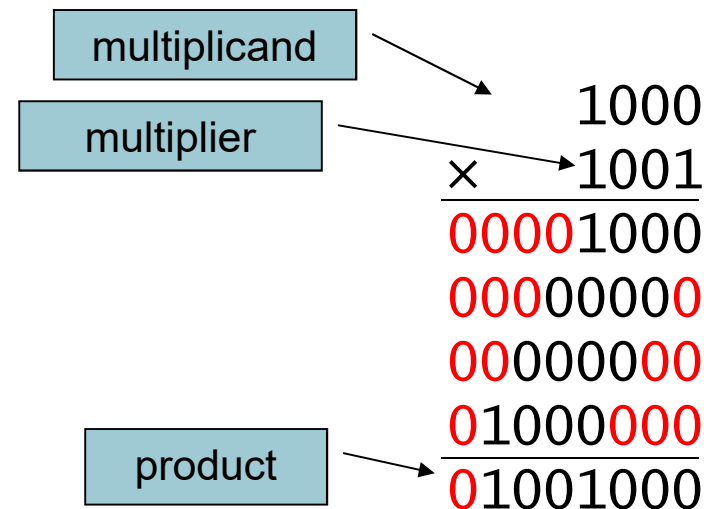
Jiang Zhong

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？

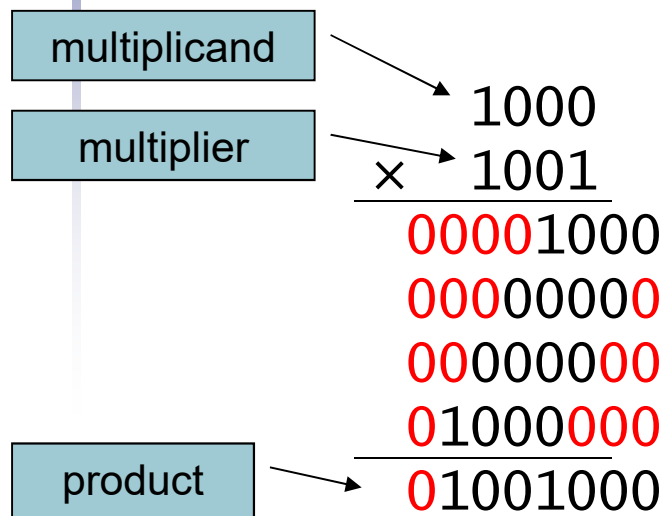
$$\begin{array}{r}
 1000 \\
 \times 1001 \\
 \hline
 1000 \\
 0000 \\
 0000 \\
 1000 \\
 \hline
 1001000
 \end{array}$$



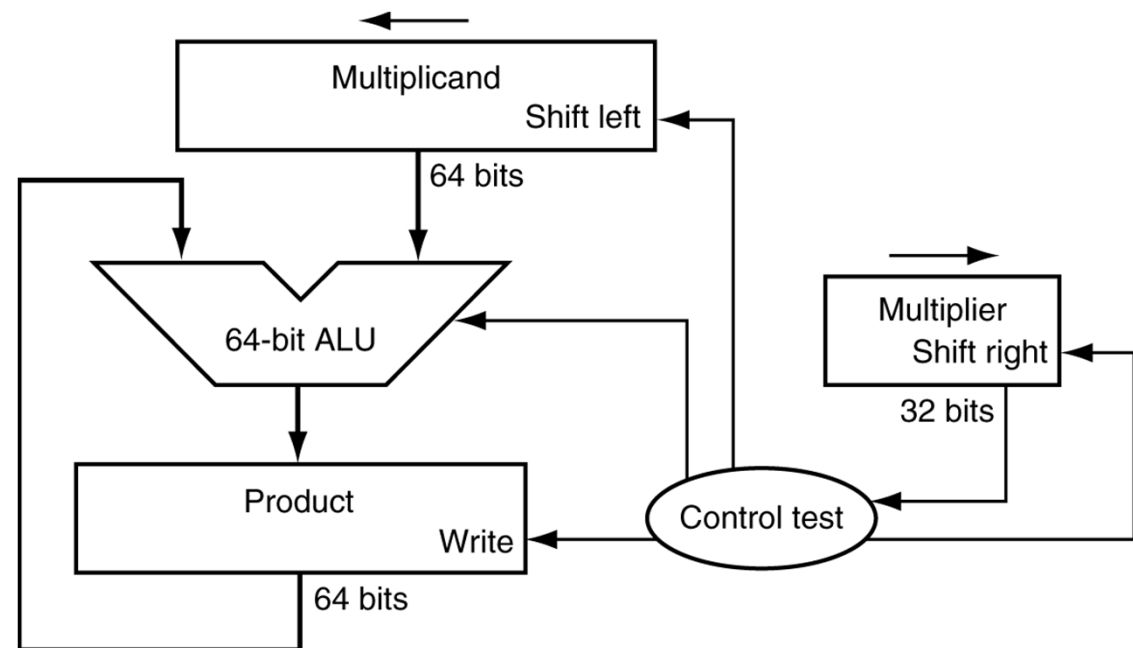
Length of product is the sum of operand lengths

Multiplication

- Start with long-multiplication approach



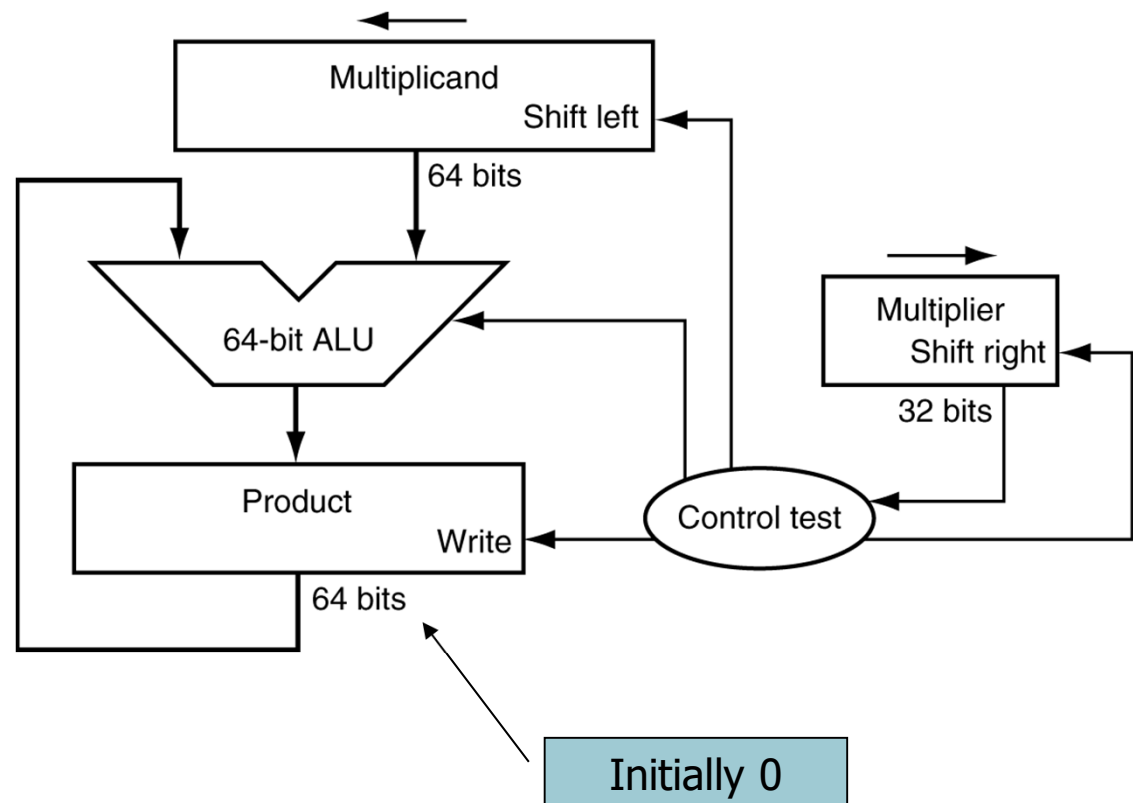
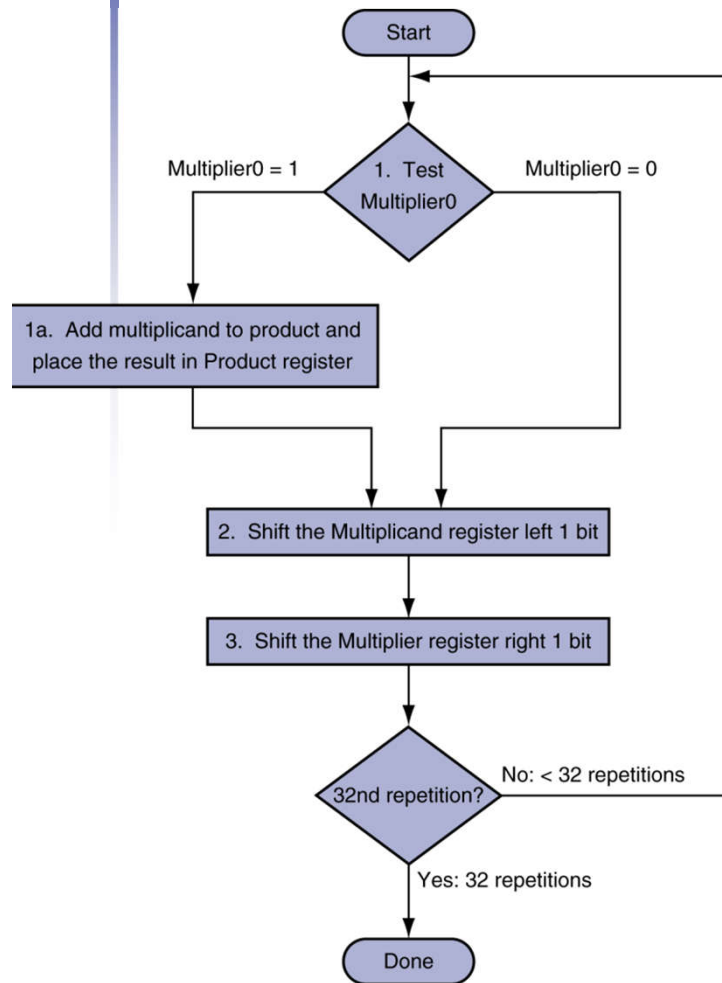
Length of product is the sum of operand lengths



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 than 1.

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

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

The product of x and y is

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

$$= x(y_12^{-1} + y_22^{-2} + \ldots + y_n2^{-n})$$

$$= 2^{-1}(y_1x + 2^{-1}(y_2x + 2^{-1}(\ldots + 2^{-1}(y_{n-1}x + 2^{-1}(y_nx + 0))\ldots)))$$

the recurrence formula for product

let z_i denotes the i th product, we could get the recurrence formula

$$z_0 = 0,$$

$$z_1 = 2^{-1}(y_n x + z_0)$$

$$z_2 = 2^{-1}(y_{n-1} x + z_1)$$

$$\vdots$$

$$z_i = 2^{-1}(y_{n-i+1} x + z_{i-1})$$

$$\vdots$$

$$z_n = x \cdot y = 2^{-1}(y_1 x + z_{n-1})$$

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。

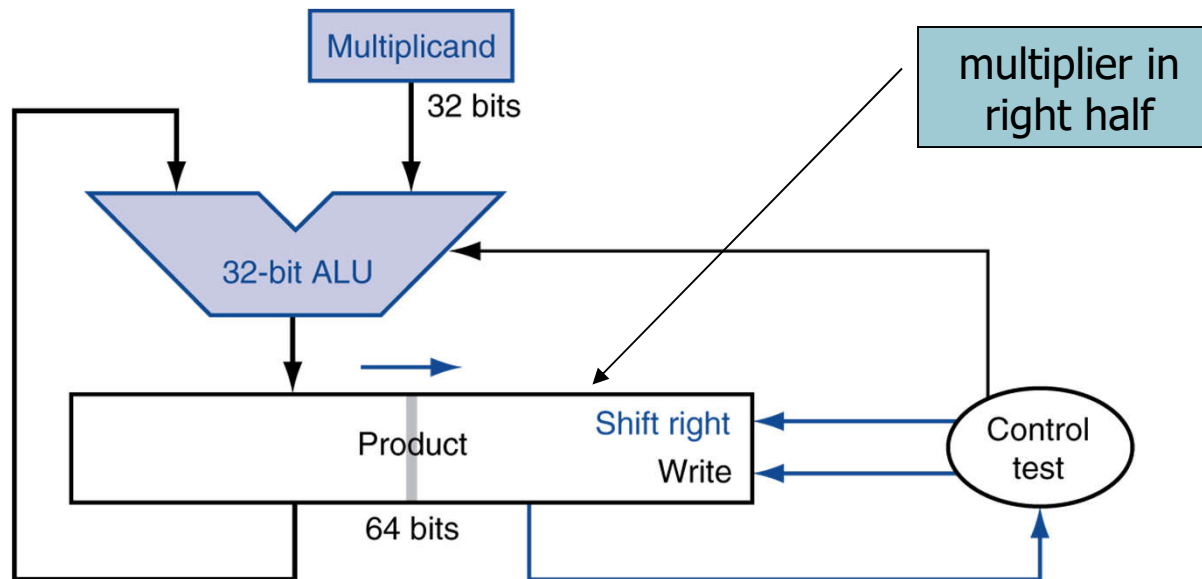
例： $x=0.1101$, $y=0.1011$, 求 $x \cdot y$ 。

部分积	乘数	说明
$ \begin{array}{r} 00.0000 \\ + 00.1101 \\ \hline 00.1101 \\ \rightarrow 00.0110 \\ + 00.1101 \\ \hline 01.0011 \\ \rightarrow 00.1001 \\ + 00.0000 \\ \hline 00.1001 \\ \rightarrow 00.0100 \\ + 00.1101 \\ \hline 01.0001 \\ \rightarrow 00.1000 \end{array} $	$ \begin{array}{l} y_f \quad 1011 \\ \begin{array}{l} \\ 1 \end{array} y_f \quad 101 \\ \begin{array}{l} \\ 11 \end{array} y_f \quad 10 \\ \begin{array}{l} \\ 111 \end{array} y_f \quad 1 \\ \begin{array}{l} \\ 1111 \end{array} y_f \end{array} $	$ \begin{array}{l} z_0=0 \\ y_4=1, +x \\ \\ \text{右移, 得 } z_1 \\ y_3=1, +x \\ \\ \text{右移, 得 } z_2 \\ y_2=0, +0 \\ \\ \text{右移, 得 } z_3 \\ y_1=1, +x \\ \\ \text{右移, 得 } z_4=x \cdot y \end{array} $
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

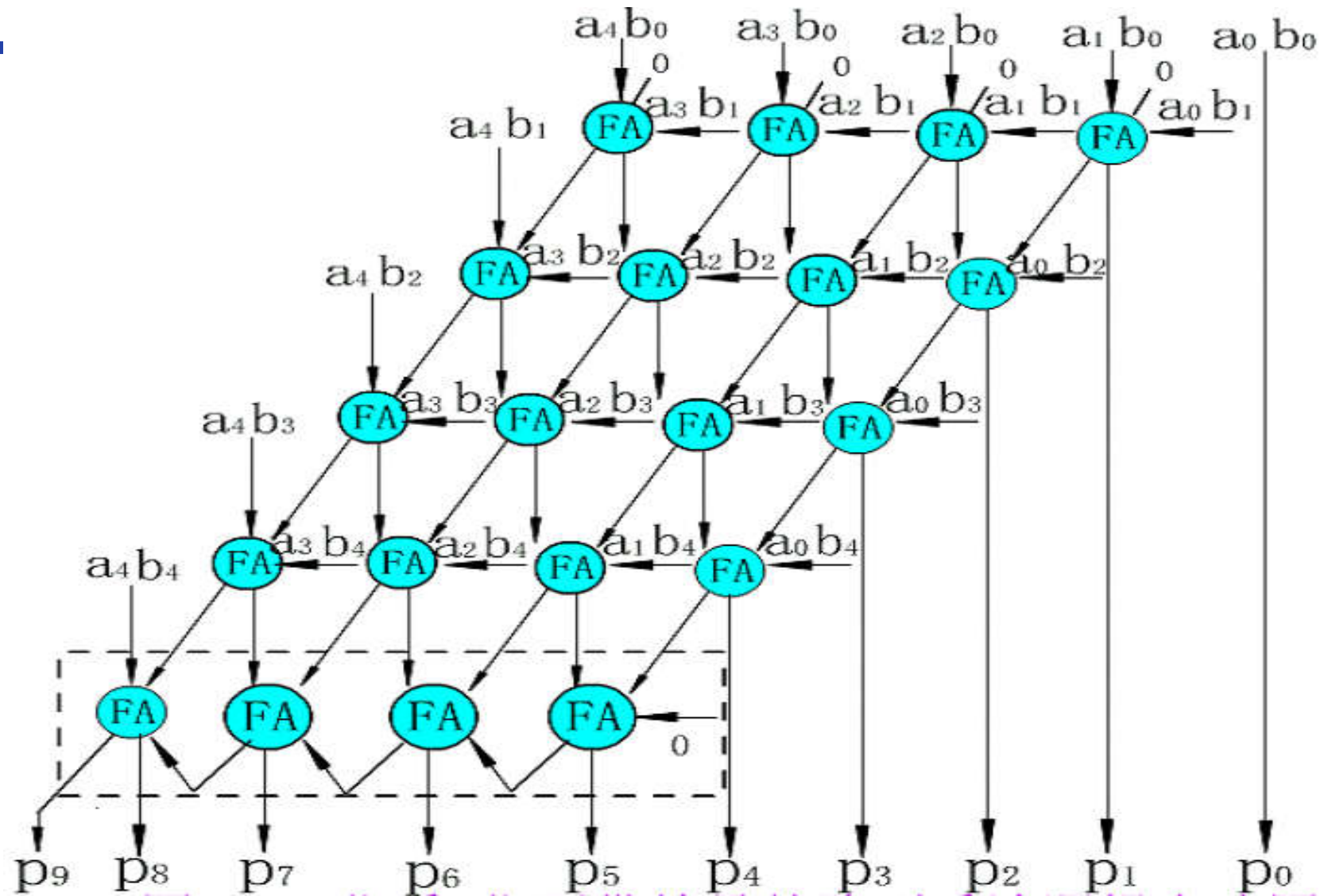


计算: $1.110_{10} * 10^{10} * 9.200_{10} * 10^{-5}$

正常使用主观题需2.0以上版本雨课堂

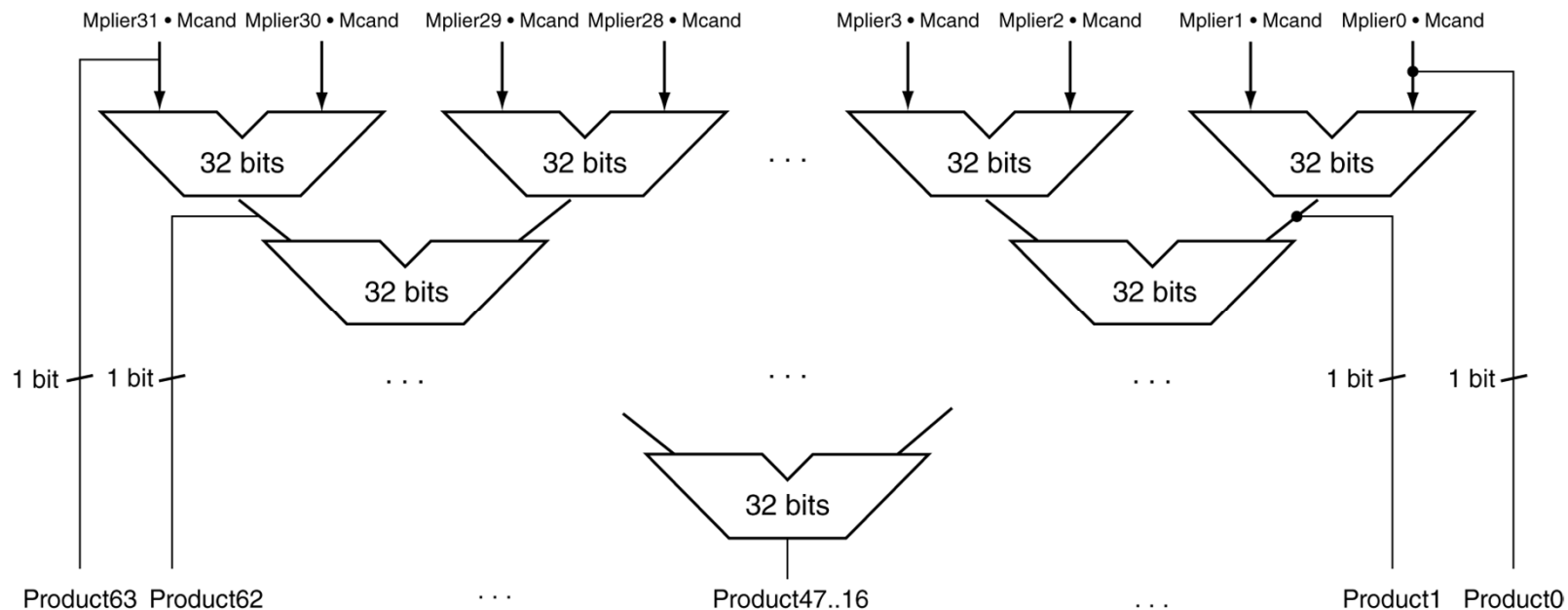
作答

Even Faster Multiplier



Even Faster Multiplier

- Uses multiple adders
 - Cost/performance tradeoff
 - 31 adders $\rightarrow \log_2(31) = \sim 5$ 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

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

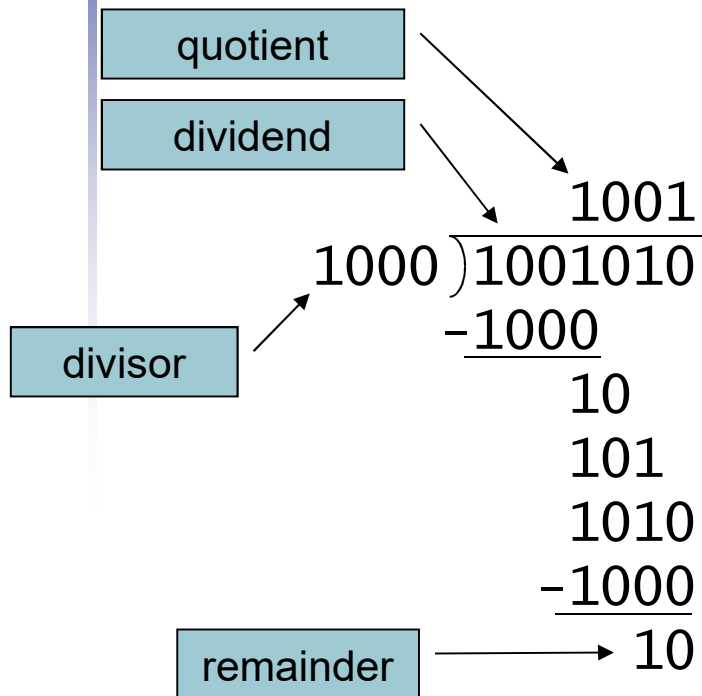
- **十进制整数除法回顾**
- 二进制整数除法
- 原码1位除法思想
- 恢复余数除法及其优化
- 加减交替除法
- 定点小数除法
- 快速除法概况

3.4 Division-- Decimal Division

$$\begin{array}{r} 021 \\ 6 \overline{)128} \\ \underline{0} \\ 1 \\ 12 \\ \underline{12} \\ 0 \\ 08 \\ \underline{6} \\ 2 \end{array}$$

$$\begin{array}{r} 02 \\ 62 \overline{)128} \\ \underline{00} \\ 12 \\ 128 \\ \underline{124} \\ 4 \end{array}$$

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 $[x]_{\text{原}} = x_f \cdot x_{n-1} \cdots x_1 x_0$

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

商 : $Q = X/Y$, 其原码为

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

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

设计二进制正整数求商和余数的算法

假设 $X = x_{n-1} \dots x_1 x_0$, $Y = y_{n-1} \dots y_1 y_0$,

X/Y 的商为 $Q = q_{n-1} \dots q_i \dots q_1 q_0$

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 $X = Y * Q + R = Y * (q_{n-1} \dots q_1 q_0) + R$, R 为余数

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

提示2: 如果 $X \geq Y * 2^{n-1}$, 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 $R \geq 0$

$q_t = 1$;

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

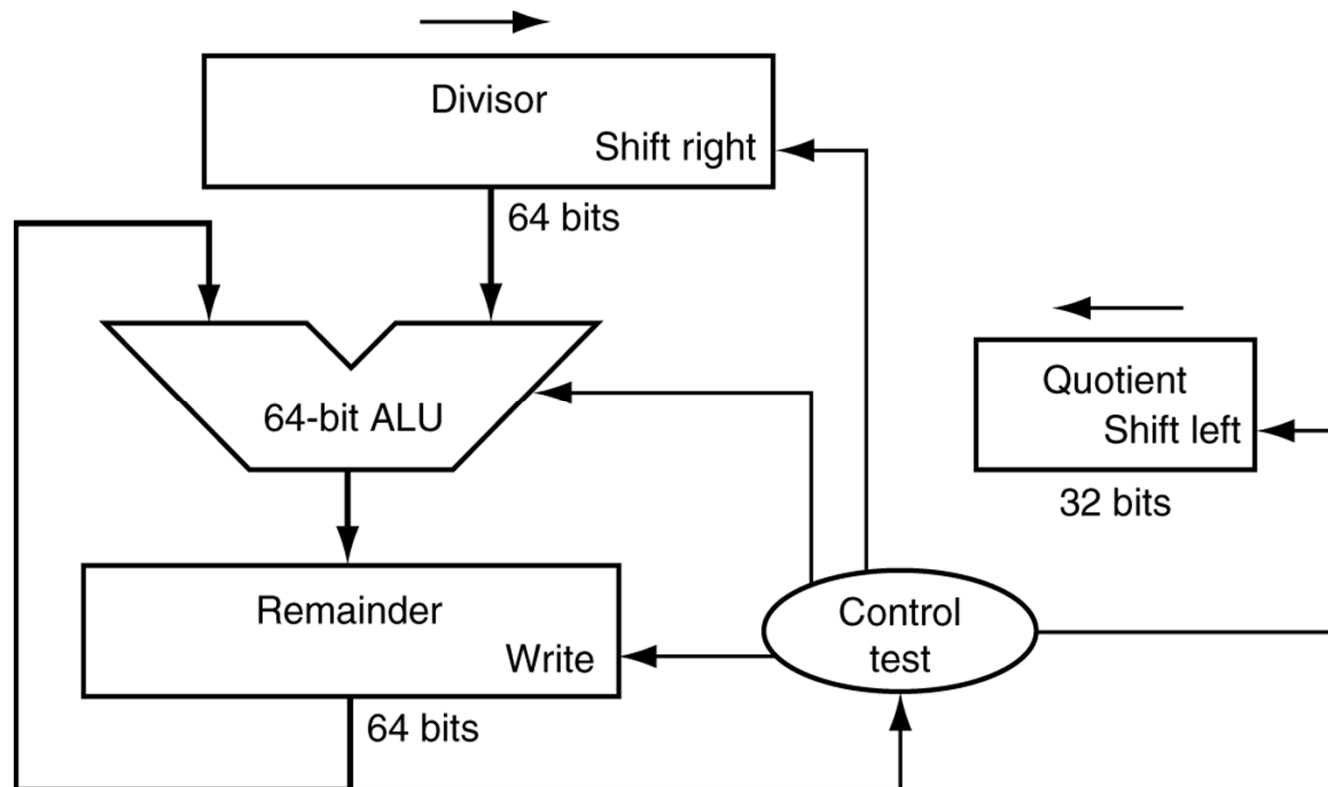
step 2. $t = t - 1$

step 3 if $t \geq 0$ goto step1

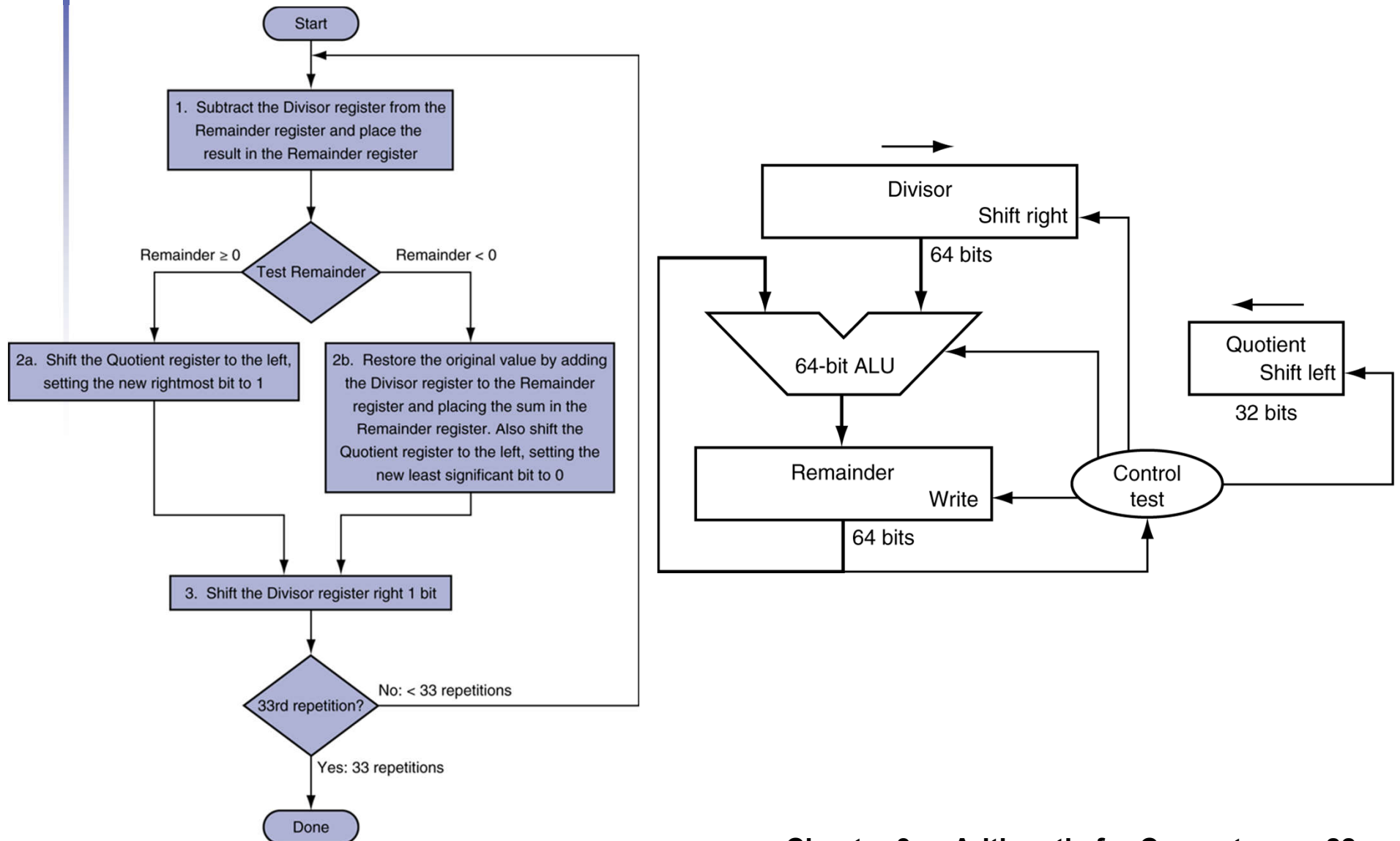
step 4 $R = X$

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

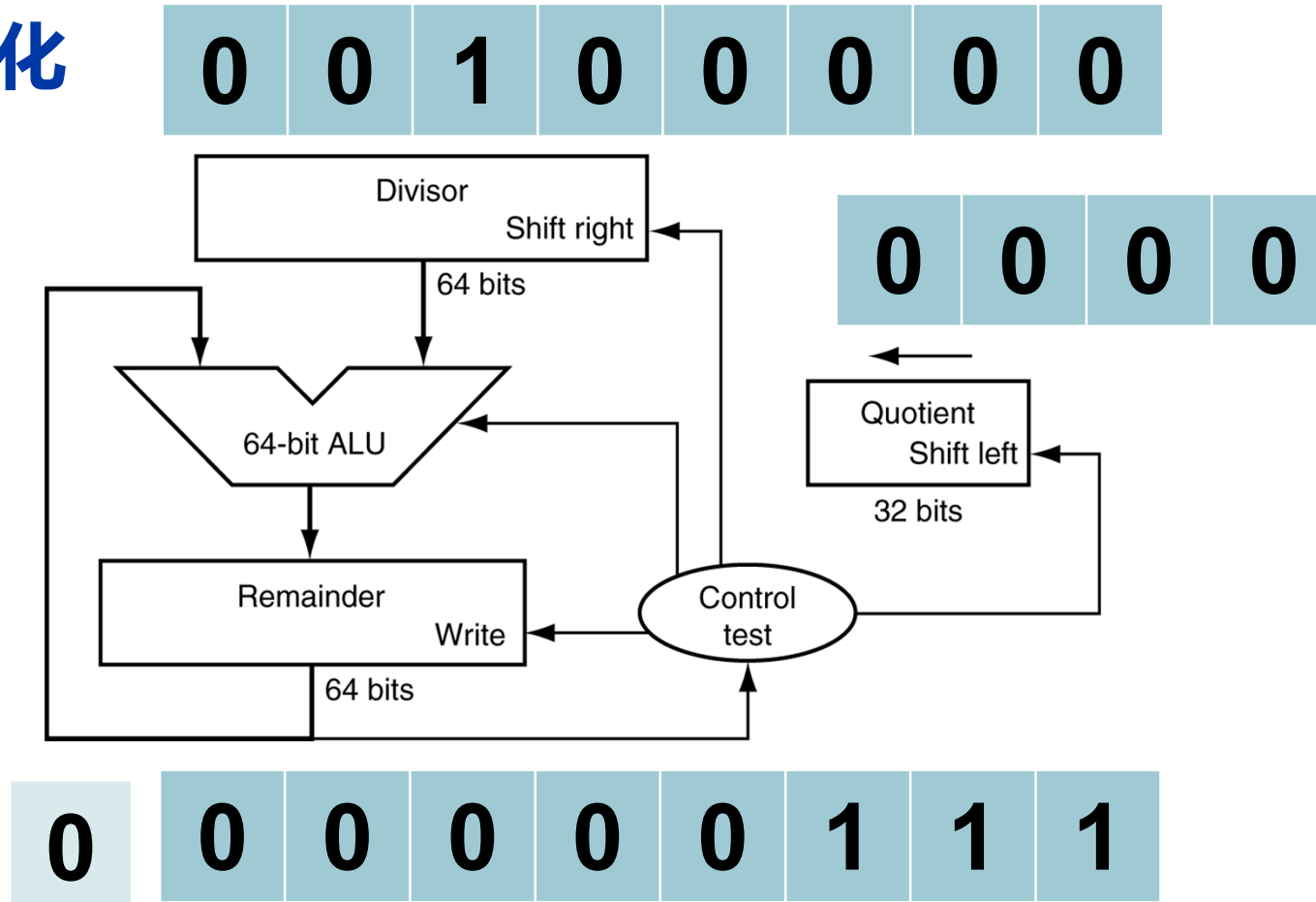
$$|x| = [0111]_2 \quad |y| = [0010]_2$$

求解 $|x/y|$

Integer Division Example

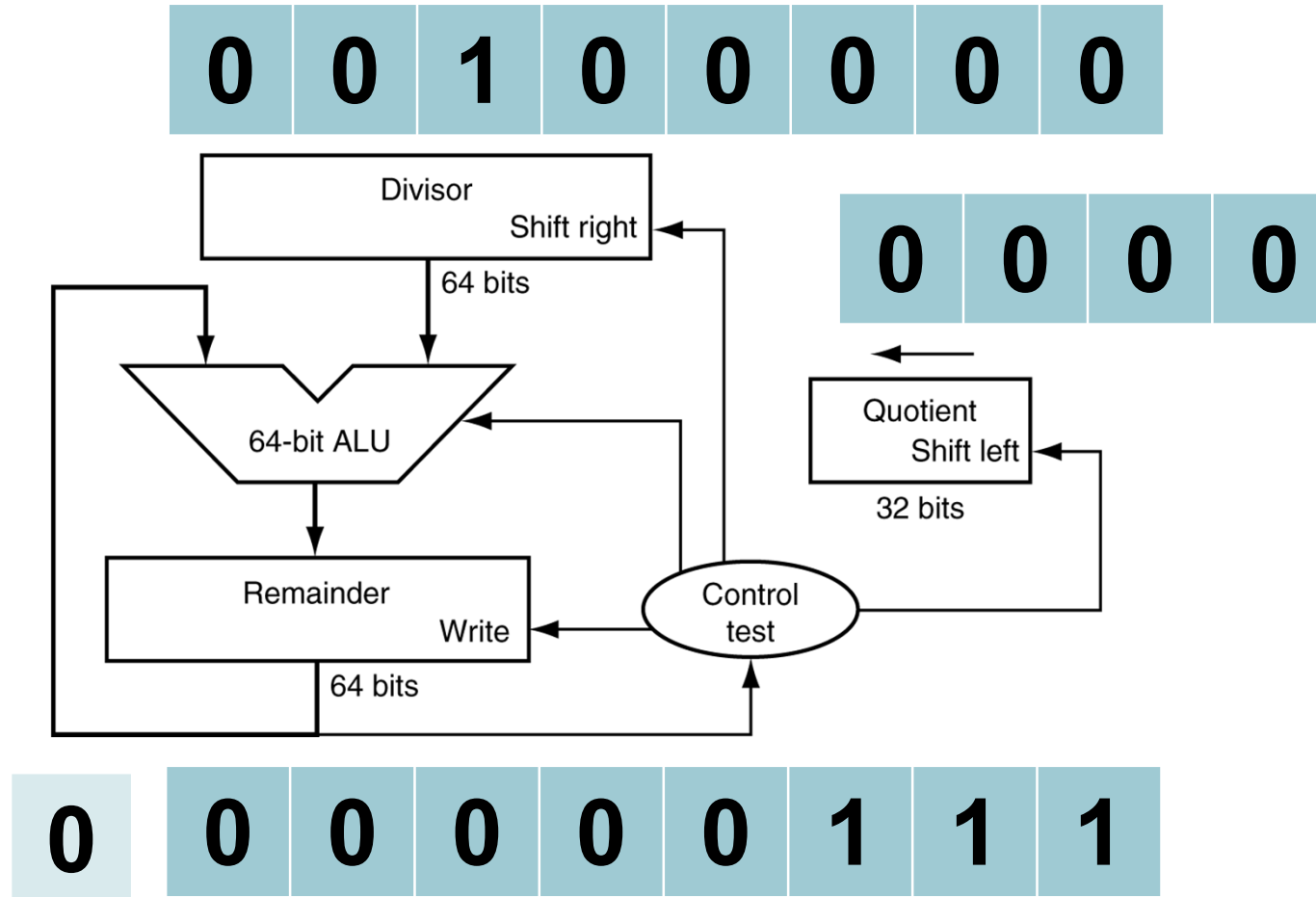
$$|x| = [0111]_2 \quad |y| = [0010]_2$$

初始化



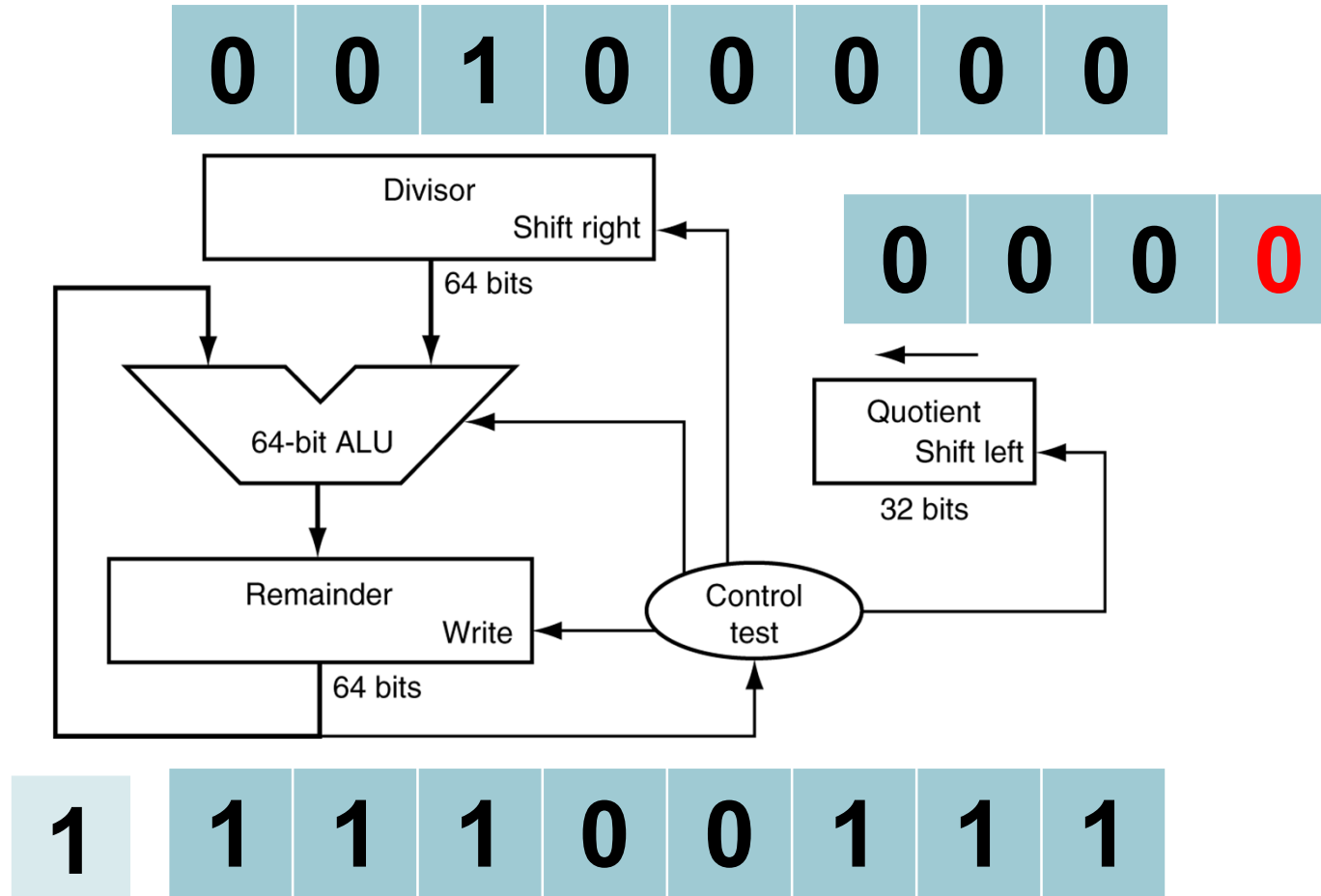
Integer Division Example

Step 1.1 余数-除数,试商



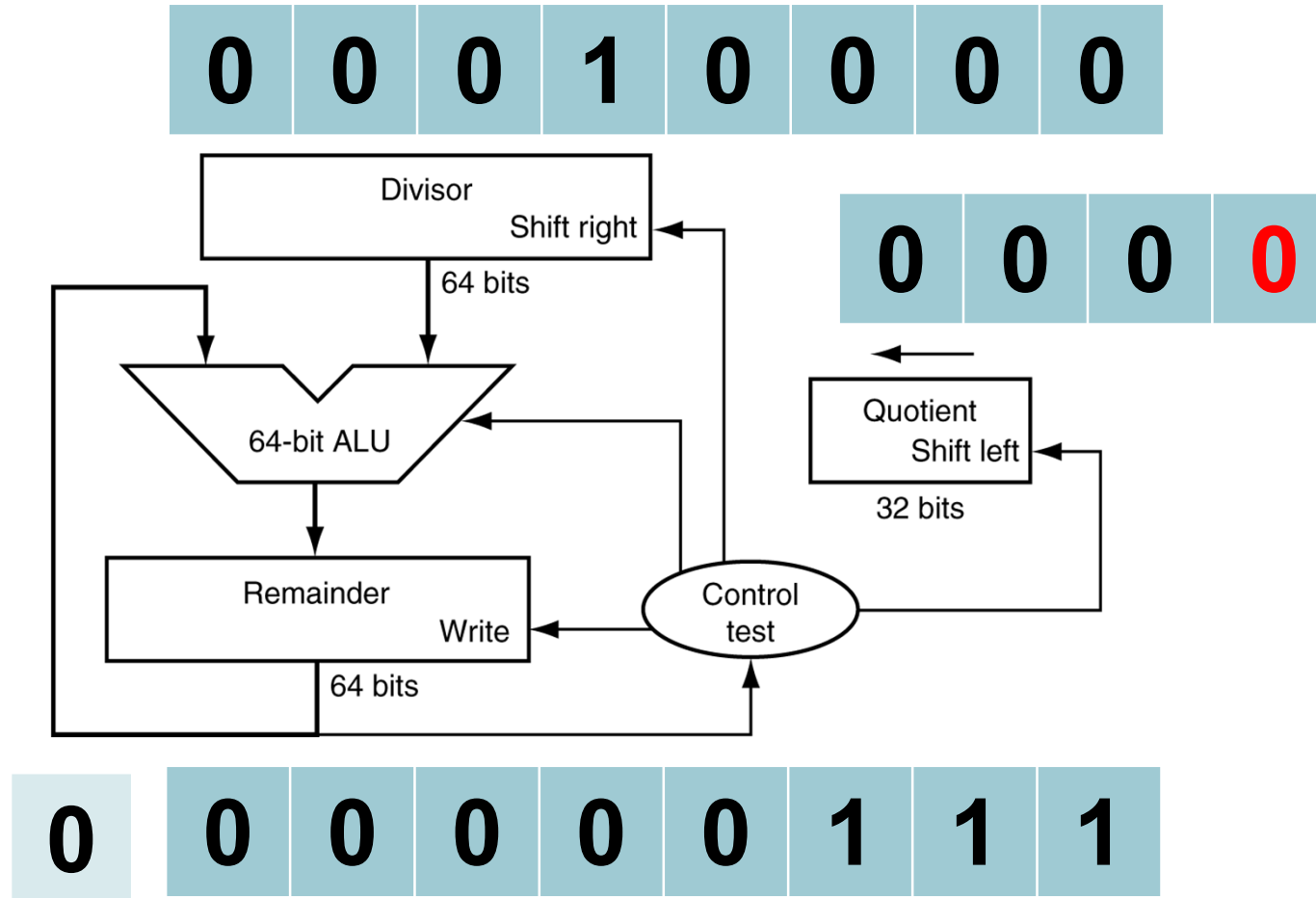
Integer Division Example

Step 1.2 余数<0,商左移1位, 置0



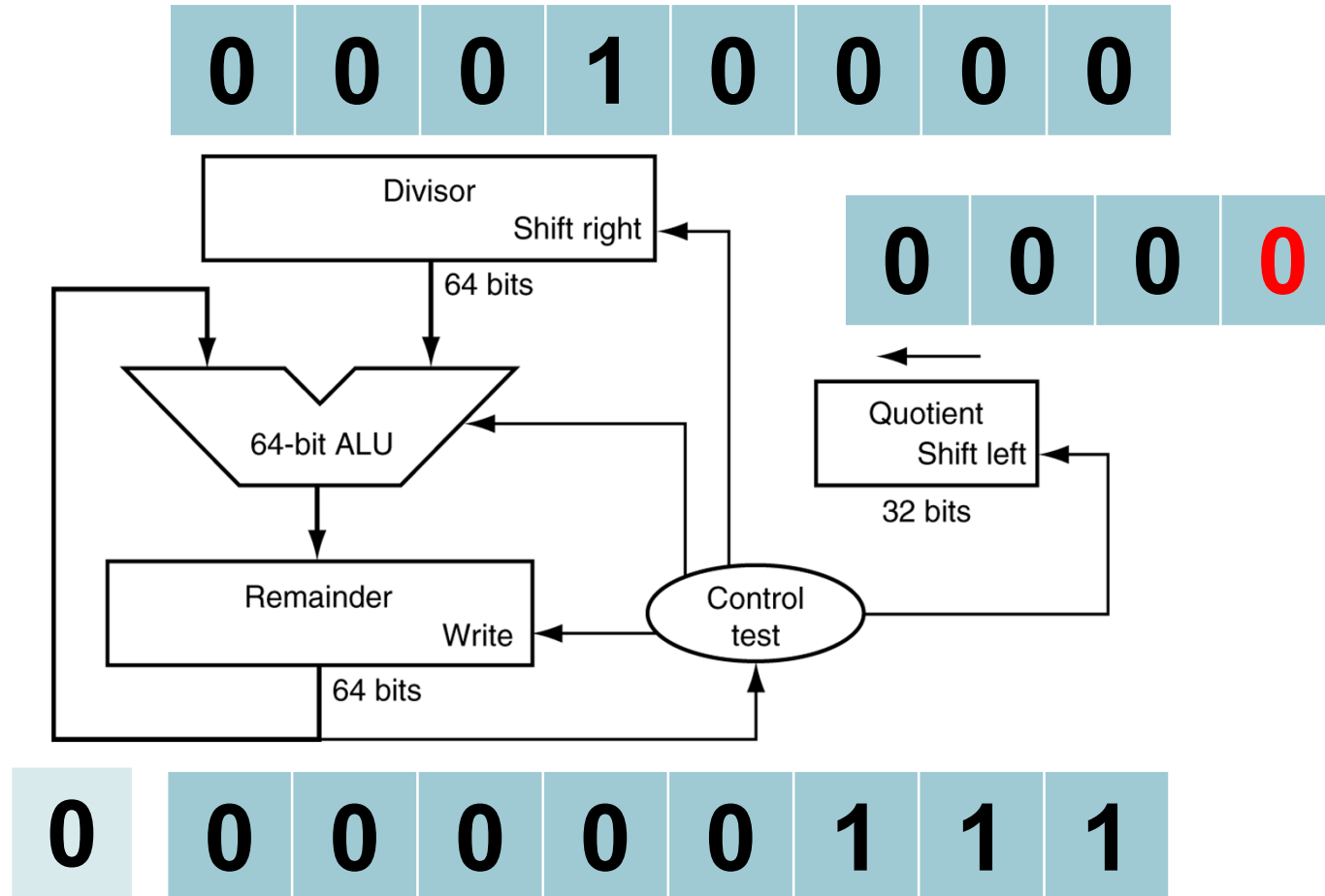
Integer Division Example

Step 1.3 恢复余数，除数右移1位



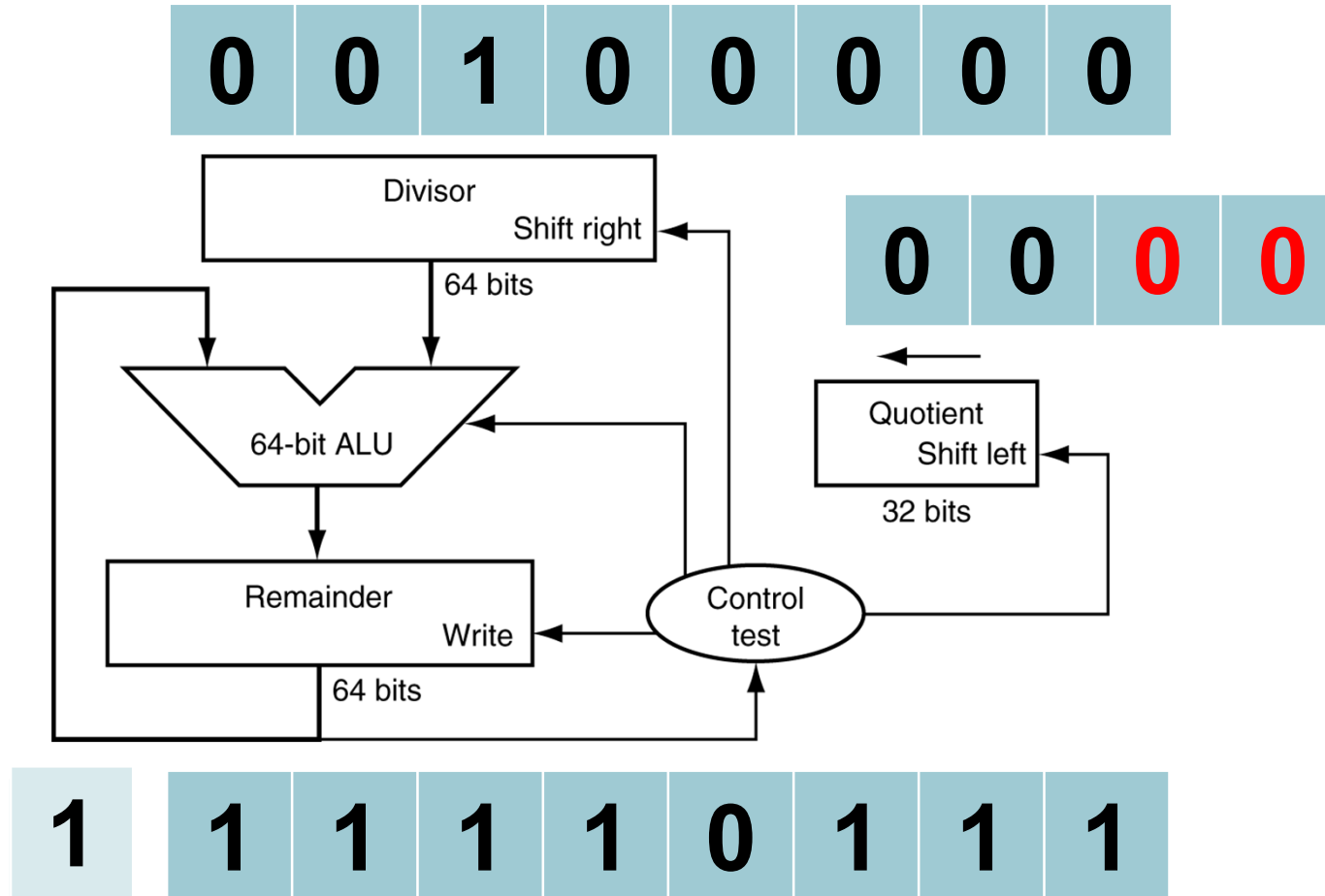
Integer Division Example

Step 2.1 余数-除数 试商



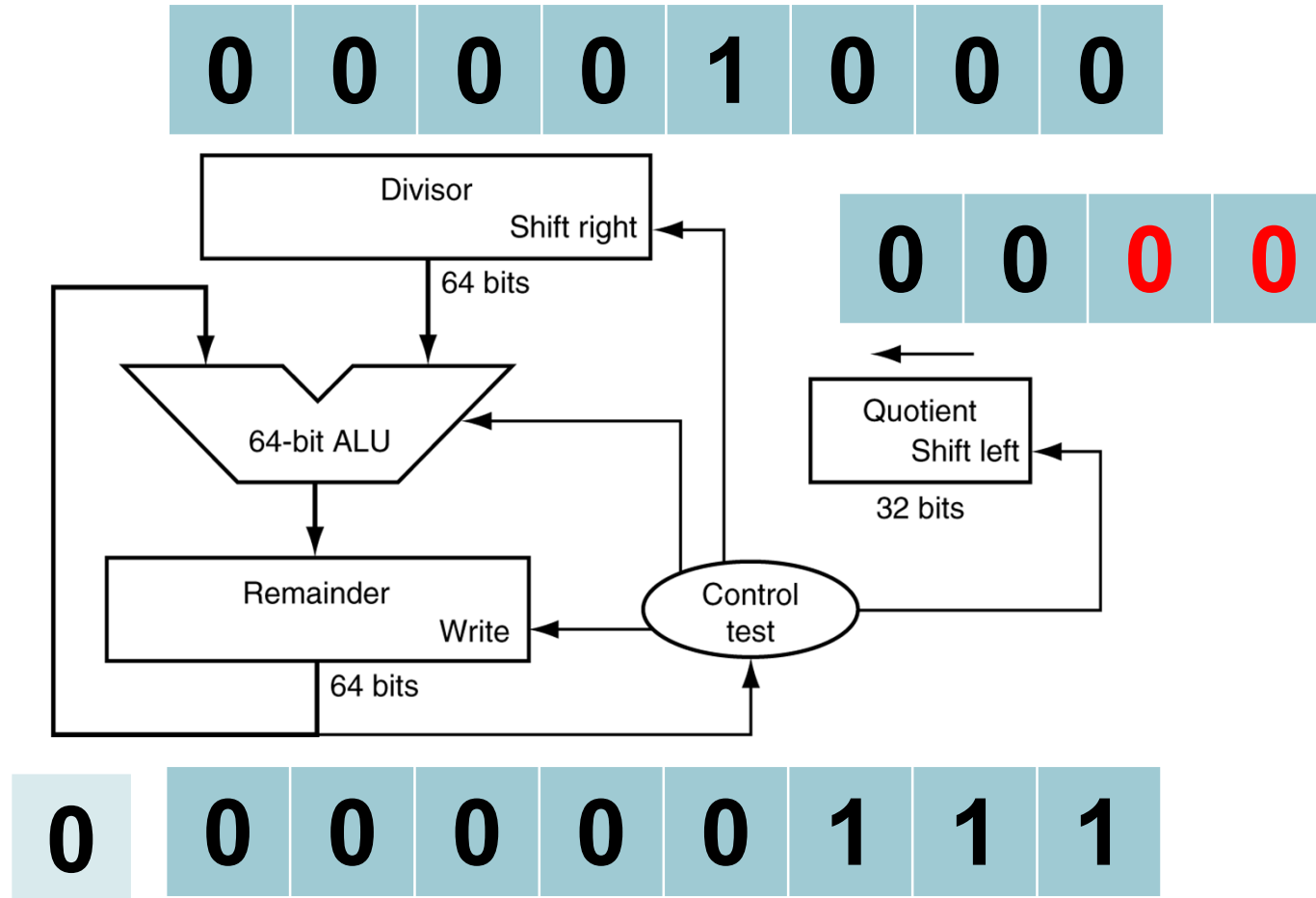
Integer Division Example

Step 2.2 余数<0,商左移1位, 置0



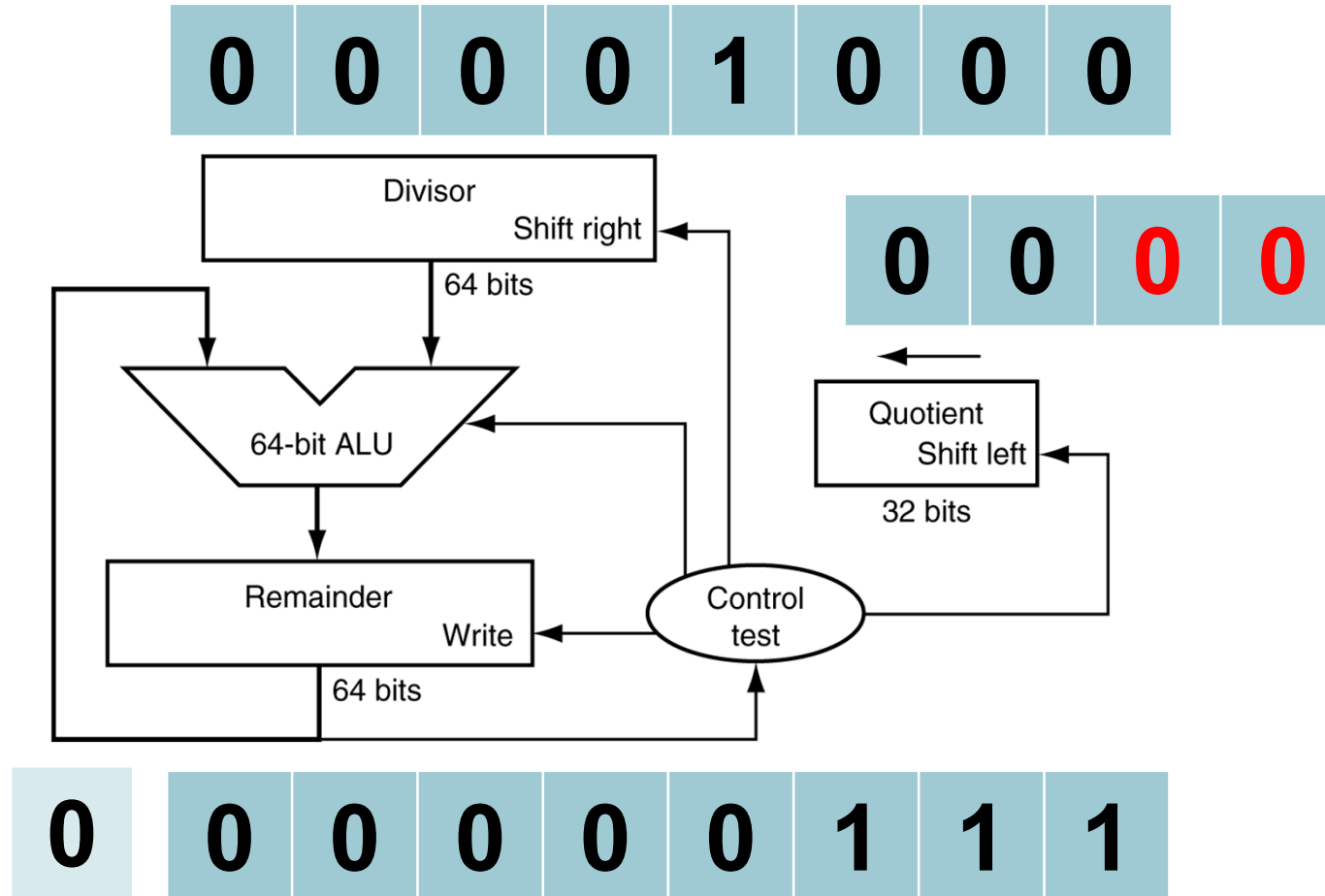
Integer Division Example

Step 2.3 恢复余数，除数右移1位



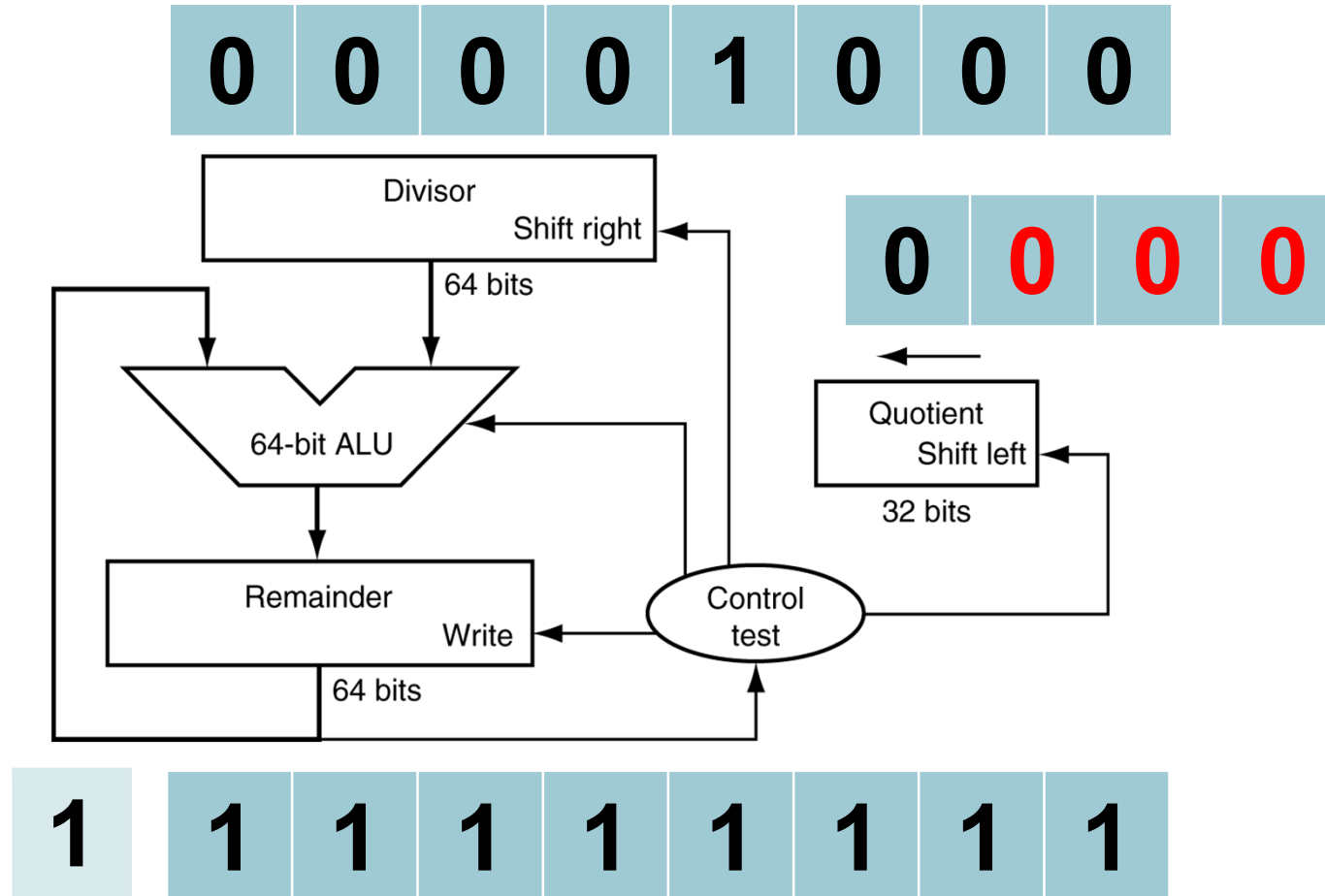
Integer Division Example

Step 3.1 余数-除数 试商



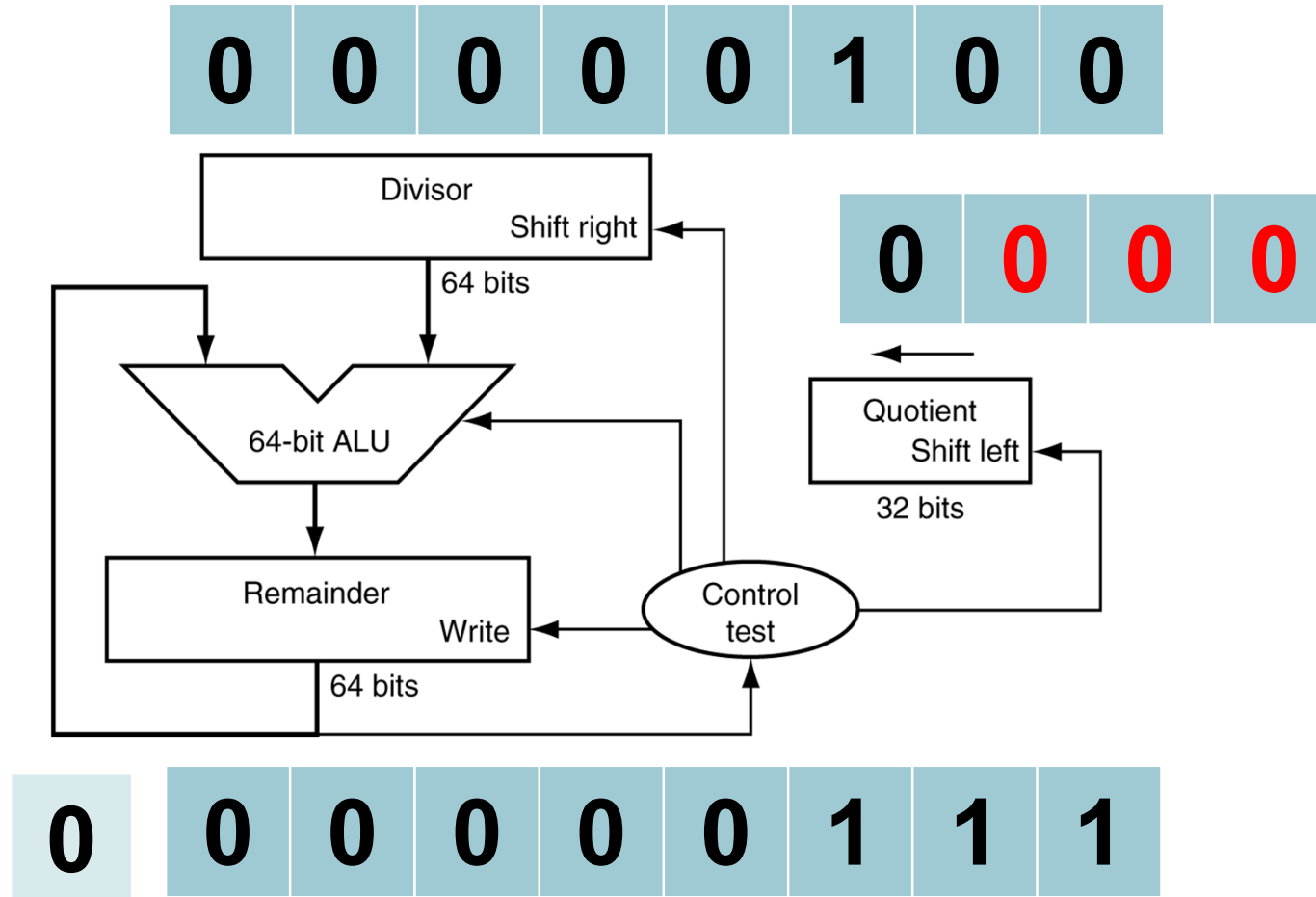
Integer Division Example

Step3.2 余数<0,商左移1位, 置0



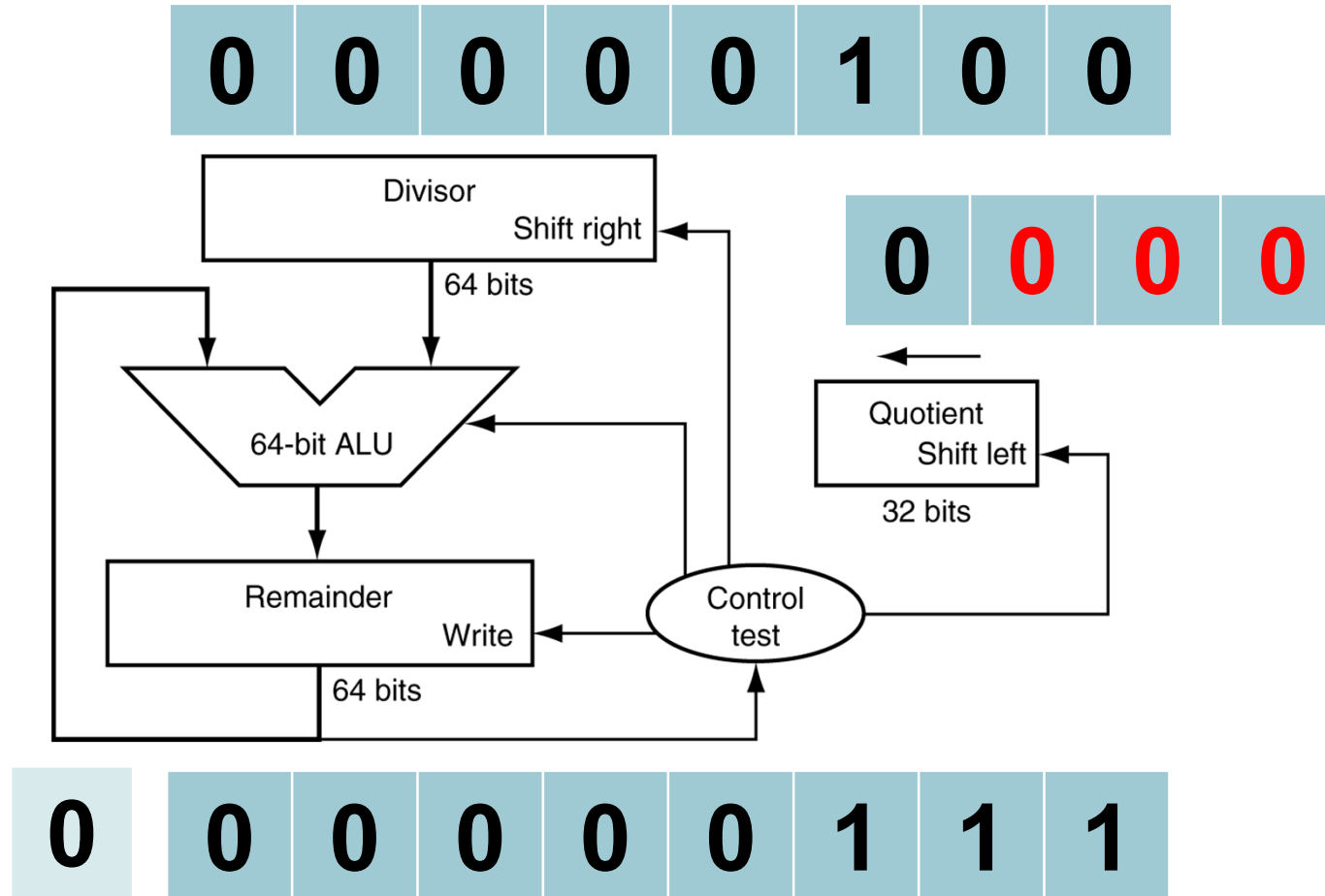
Integer Division Example

Step 3.3 恢复余数，除数右移1位



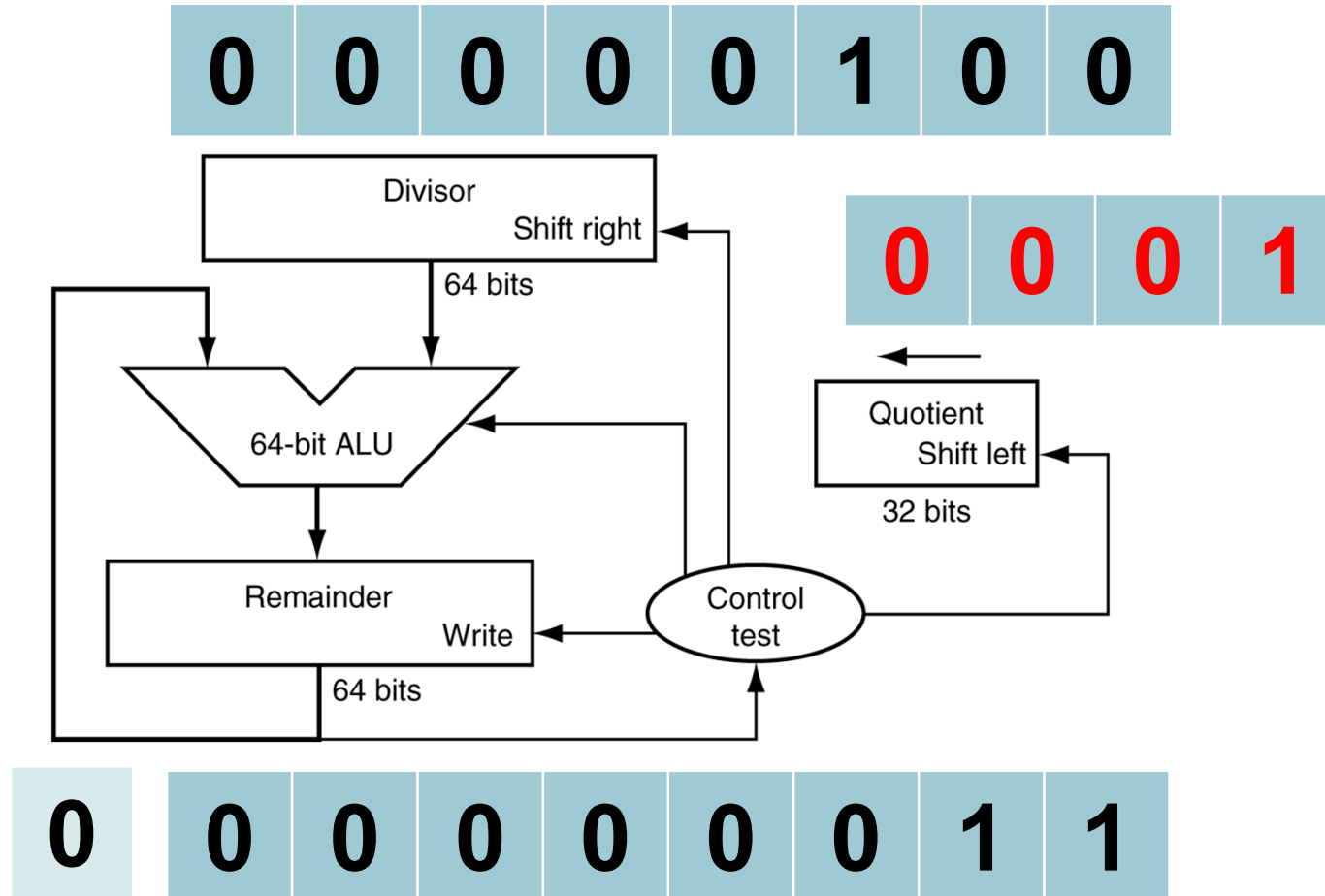
Integer Division Example

Step 4.1 余数-除数 试商



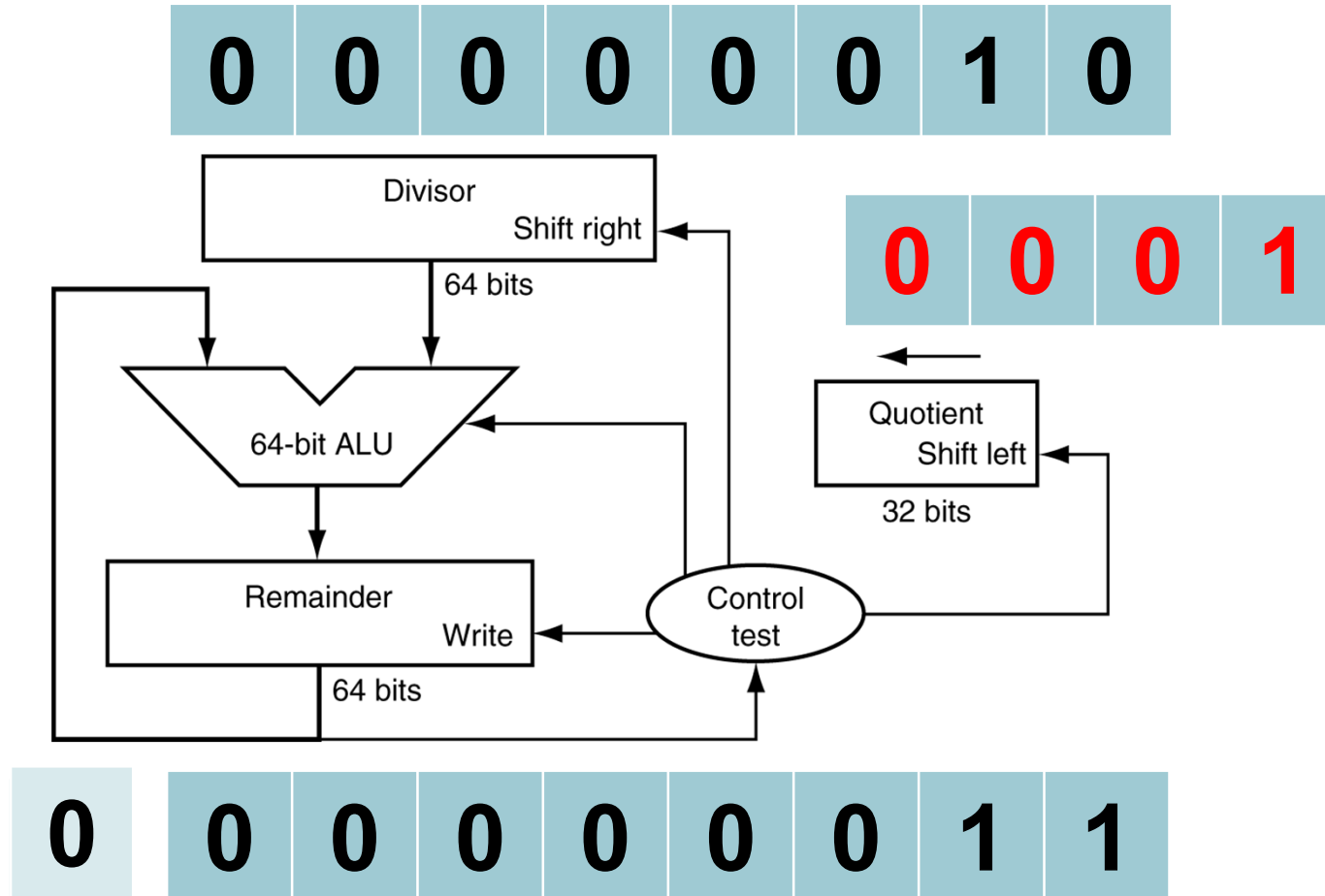
Integer Division Example

Step4.2 余数>0,商左移1位, 置1



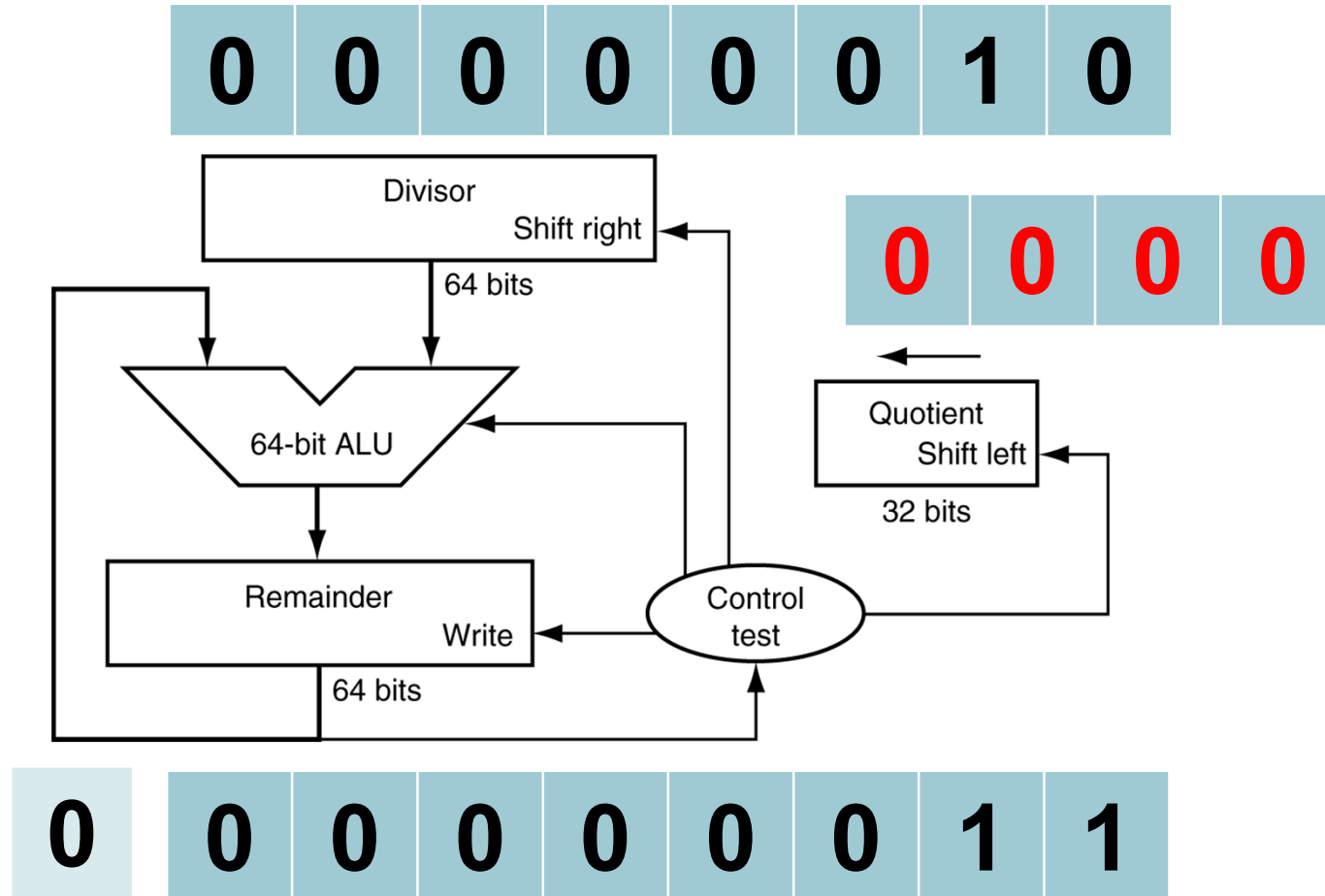
Integer Division Example

Step 4.4 除数右移1位



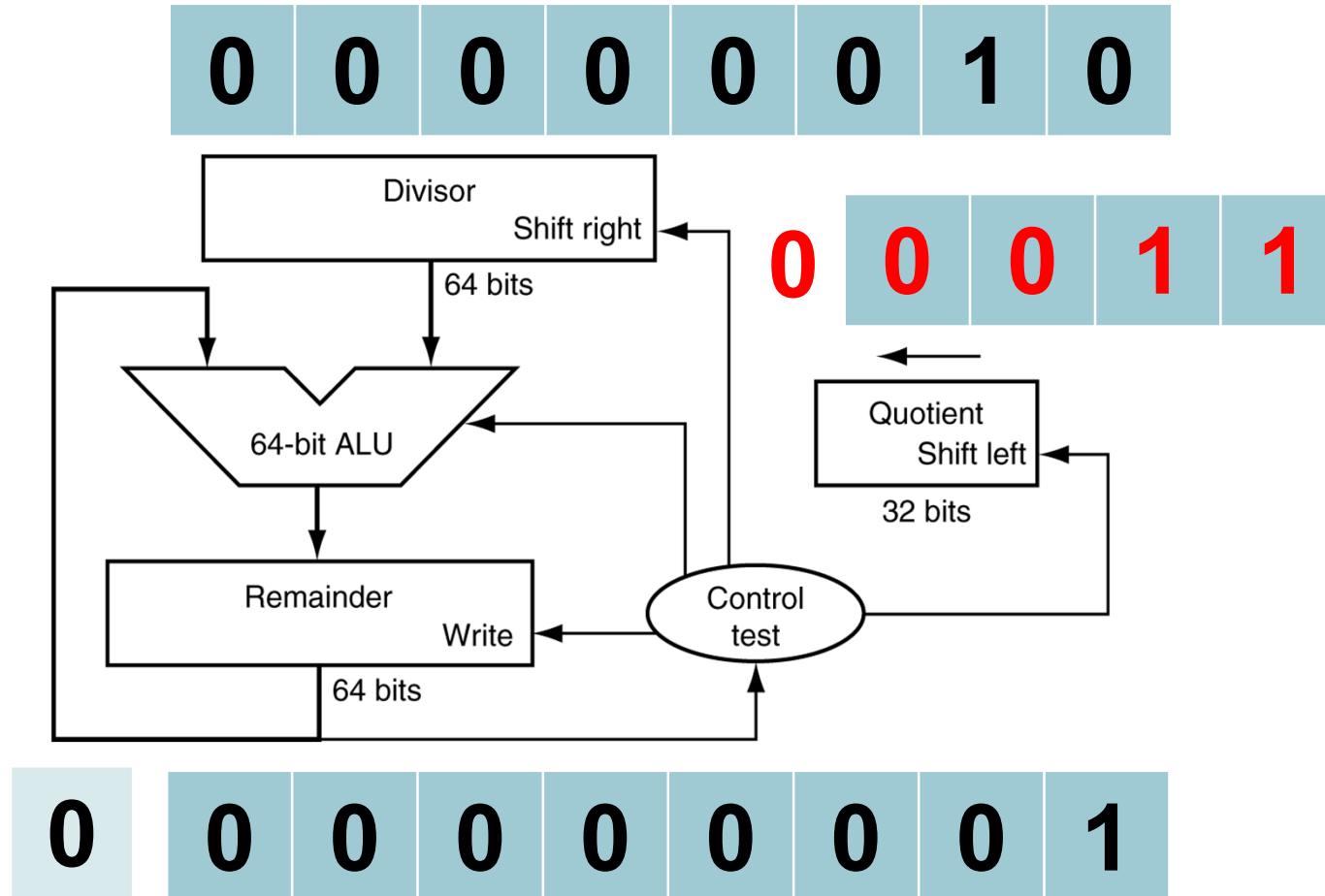
Integer Division Example

Step 5.1 余数-除数 试商



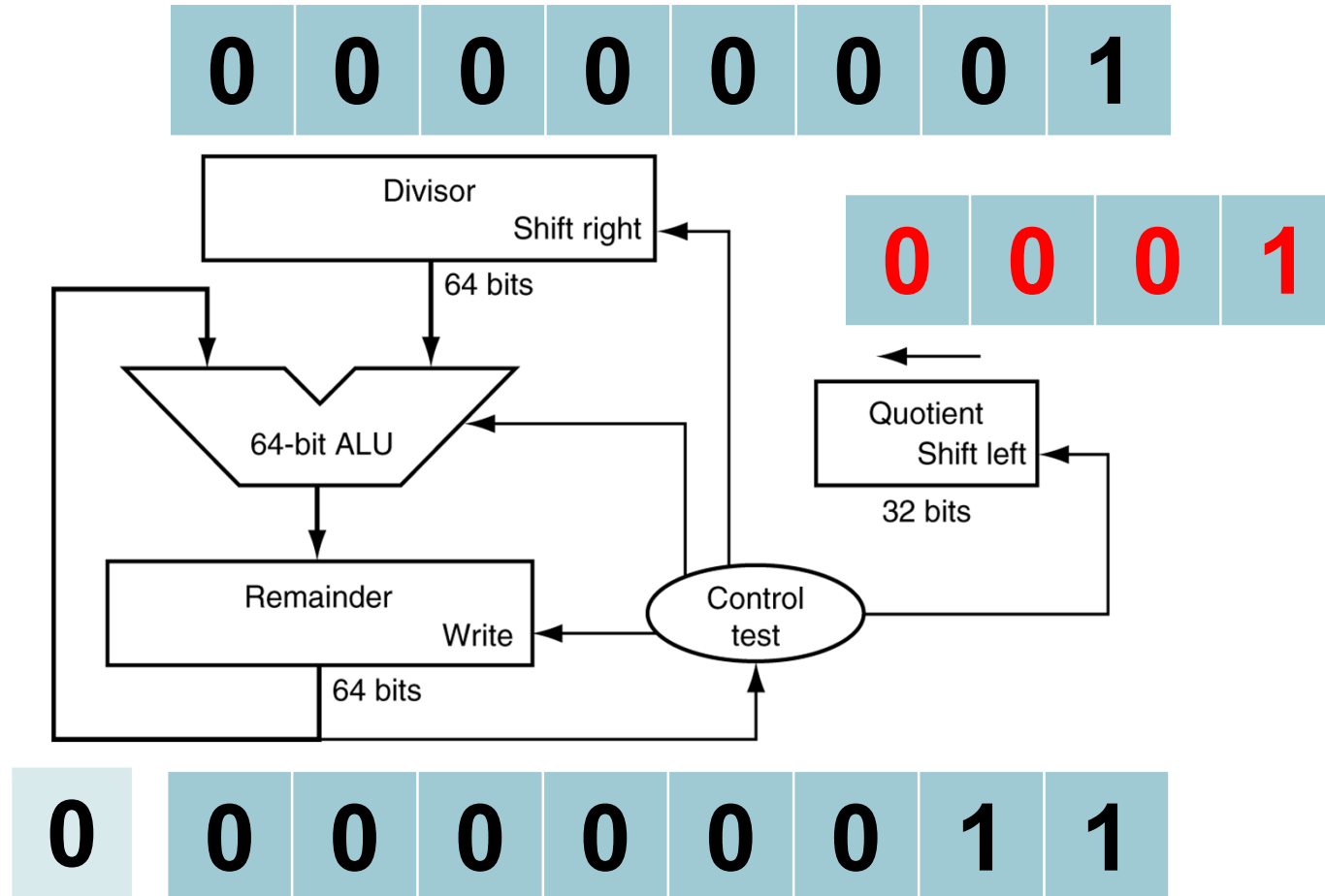
Integer Division Example

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)



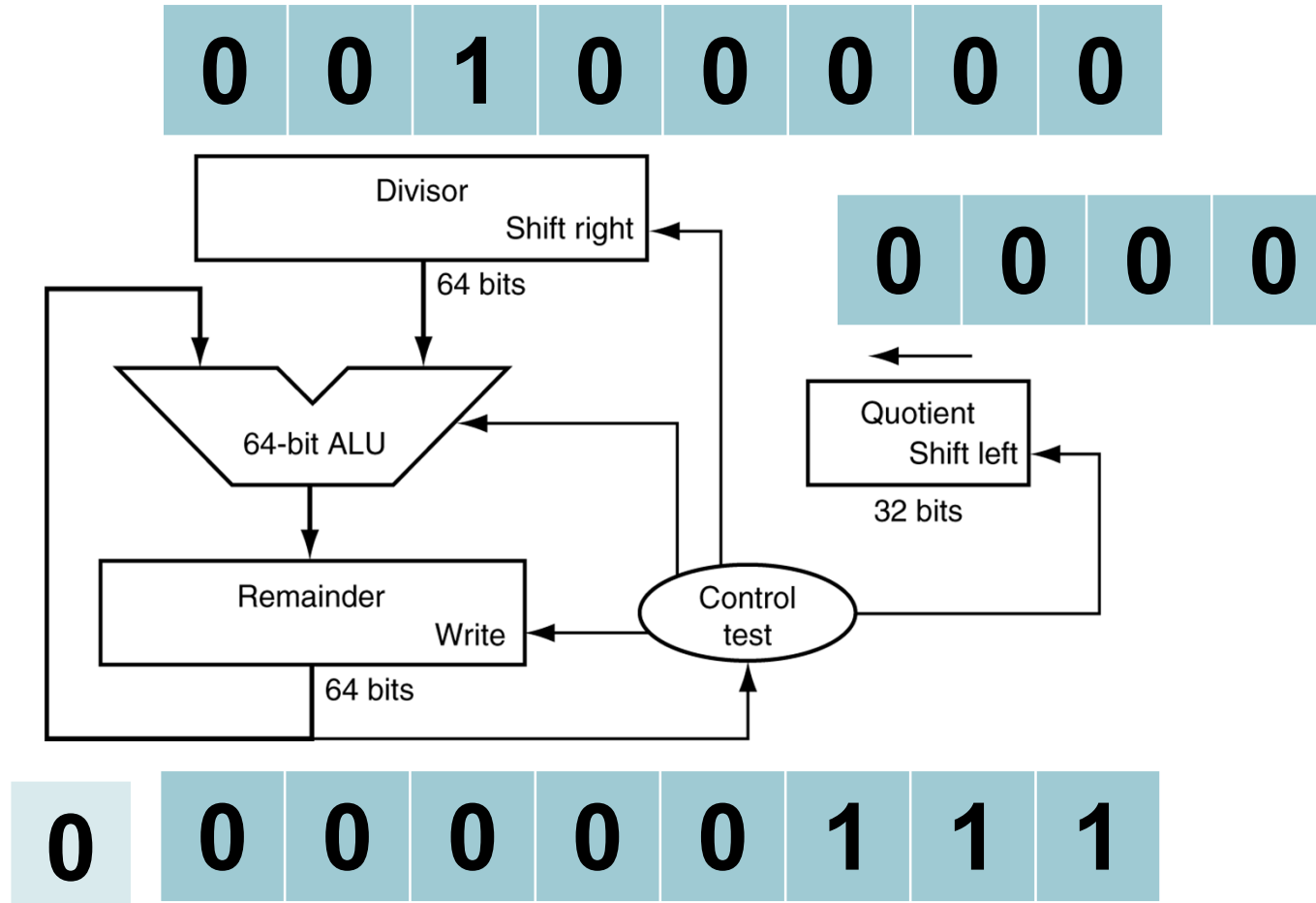
请同学们再观察一遍 $|x|/|y|$ 的过程

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

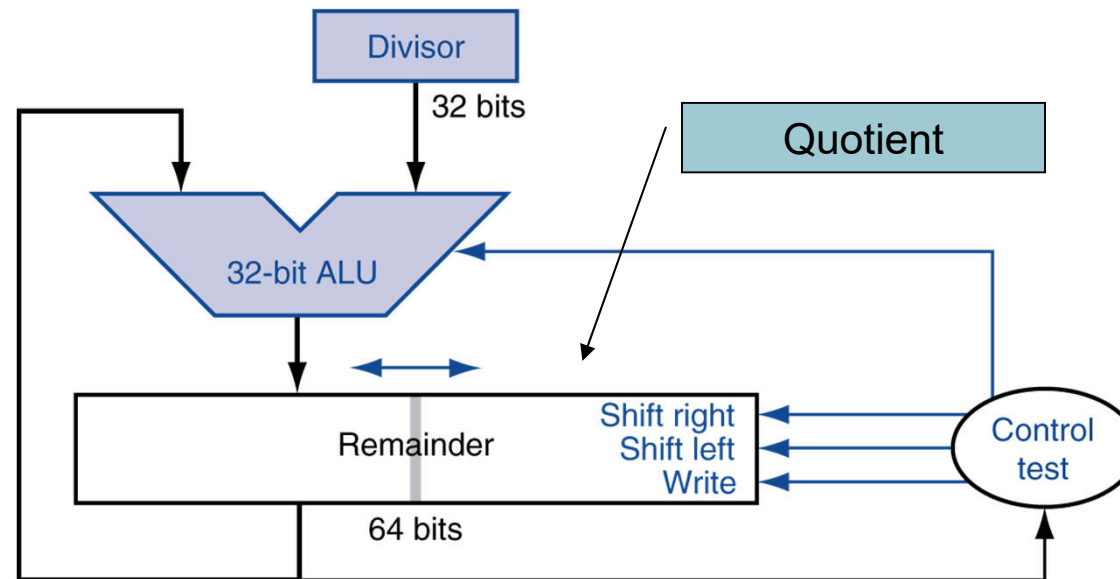
迭代次数	步骤	商	除数	余数
0	初始化	0000	<u>0010 0000</u>	0 0000 0111
1	余数=余数-除数 试商, 余数<0 恢复余数 商左移1位, 最低位置0 除数右移	0000 0000 0000	0010 0000 0010 0000 0001 0000	1 1110 0111 0 0000 0111 0 0000 0111
2	余数=余数-除数 试商, 余数<0 恢复余数 商左移1位, 最低位置0 除数右移	0000 0000 0000	0001 0000 0001 0000 0000 1000	1 1111 0111 0 0000 0111 0 0000 0111
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 0000 0000	0000 1000 0000 1000 0000 0100	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 0100 0000 0010	0 0000 0011 0 0000 0011 0 0000 0011
5	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	<u>0000 0010</u> 0000 0010 0000 0001	0 0000 0001 0 0000 0001 0 0000 0001

Integer Division

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



Optimized Integer Divider

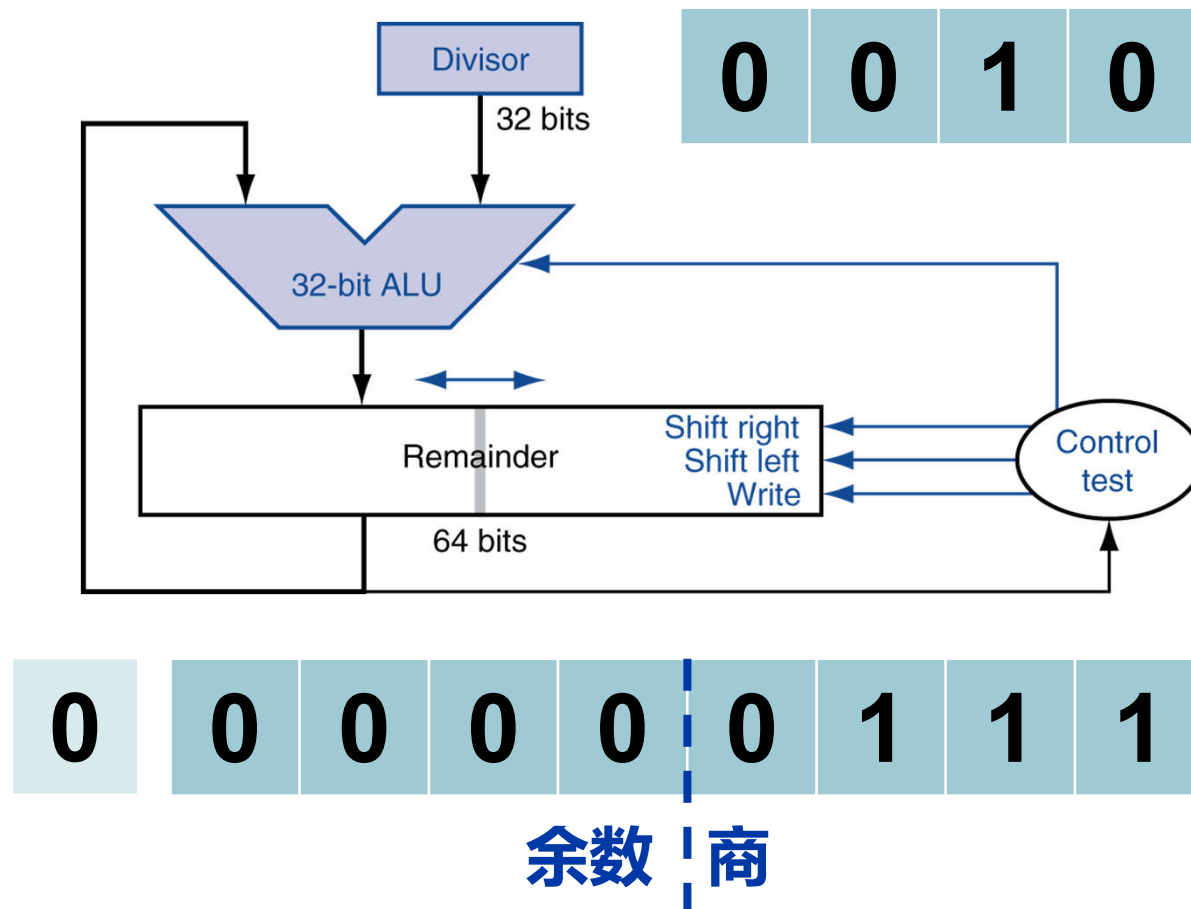


- One cycle per partial-remainder subtraction
- Looks a lot like a multiplier!
 - Same hardware can be used for both

Optimized Integer Divider

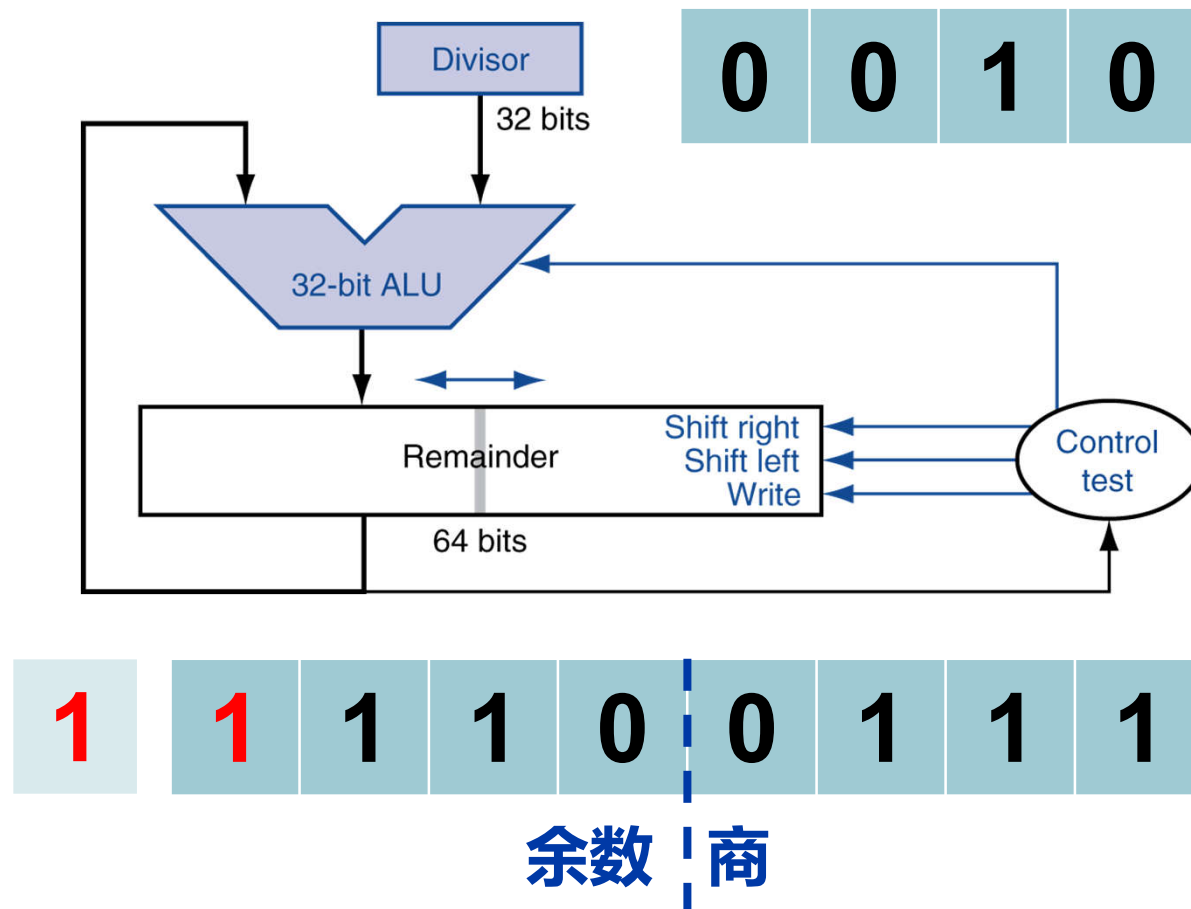
Step 0 初始化

被除数 = $[0111]_2$ 除数 = $[0010]_2$



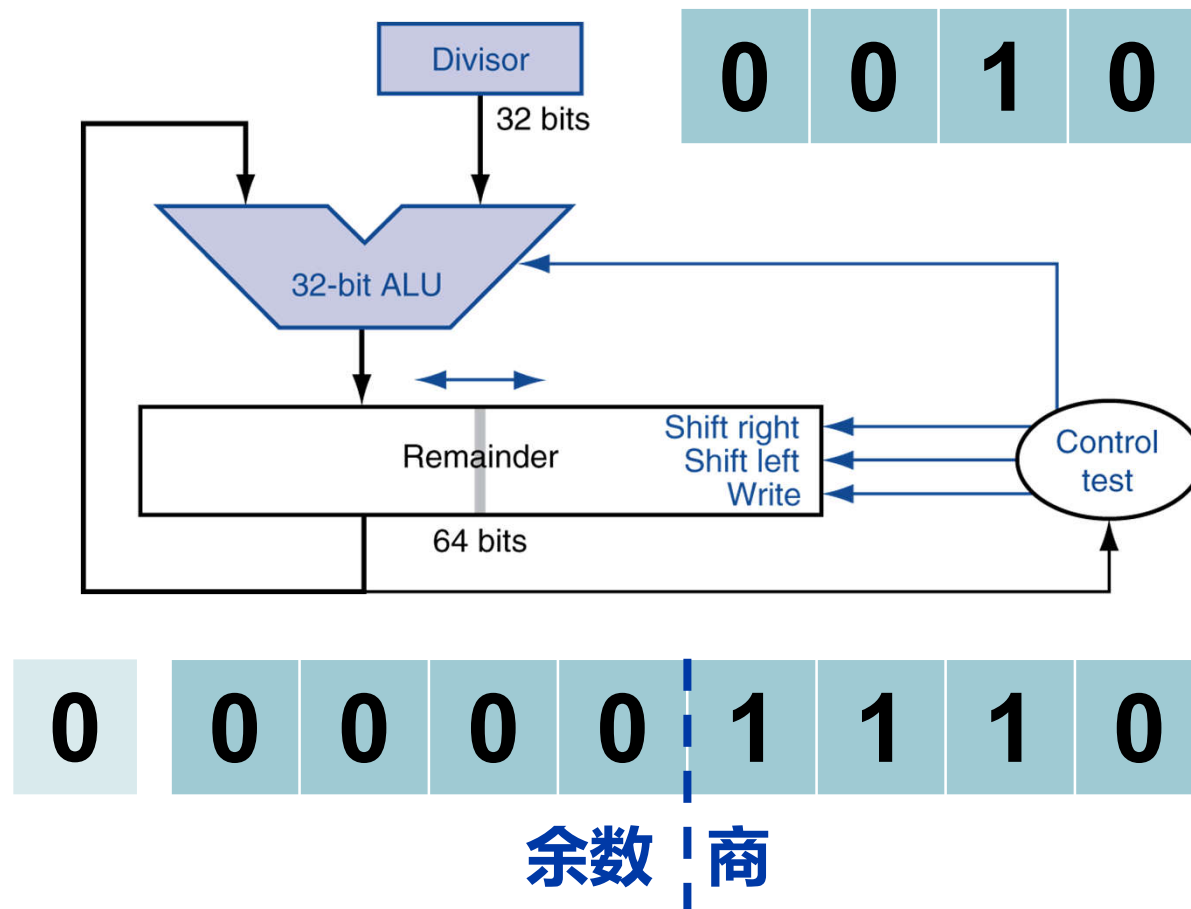
Optimized Integer Divider

Step 1.1 余数=余数-除数 试商



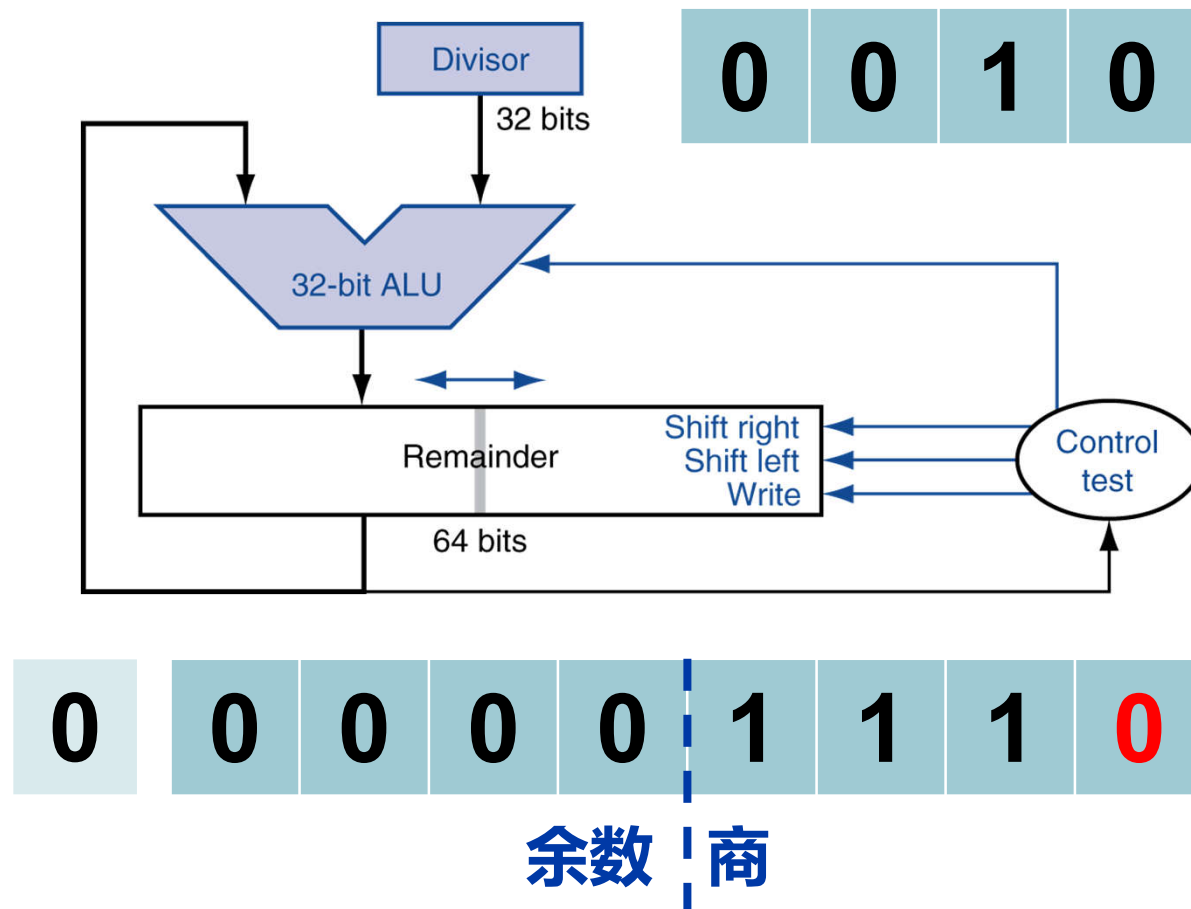
Optimized Integer Divider

Step 1.2 余数<0 恢复余数，余数和商 左移1位



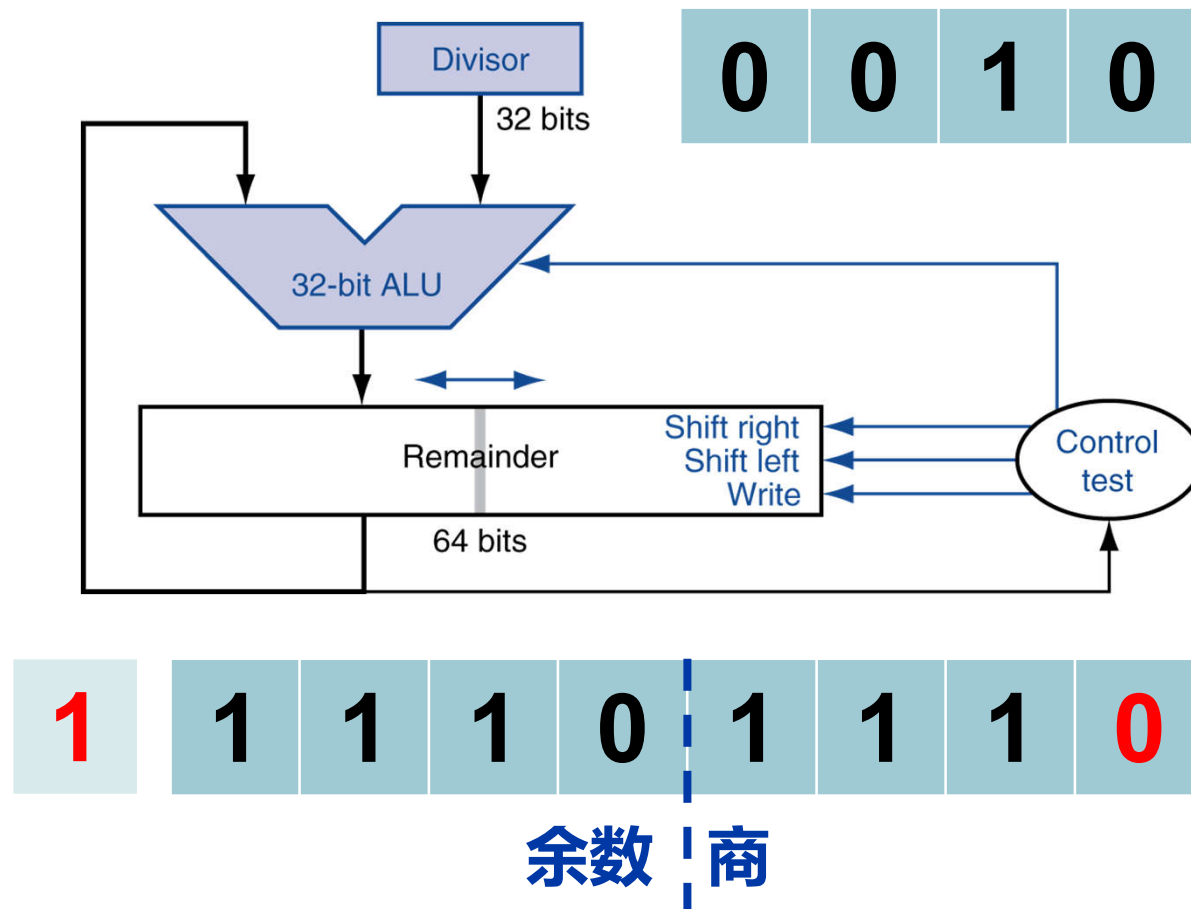
Optimized Integer Divider

Step 1.3 商的 最低位置0



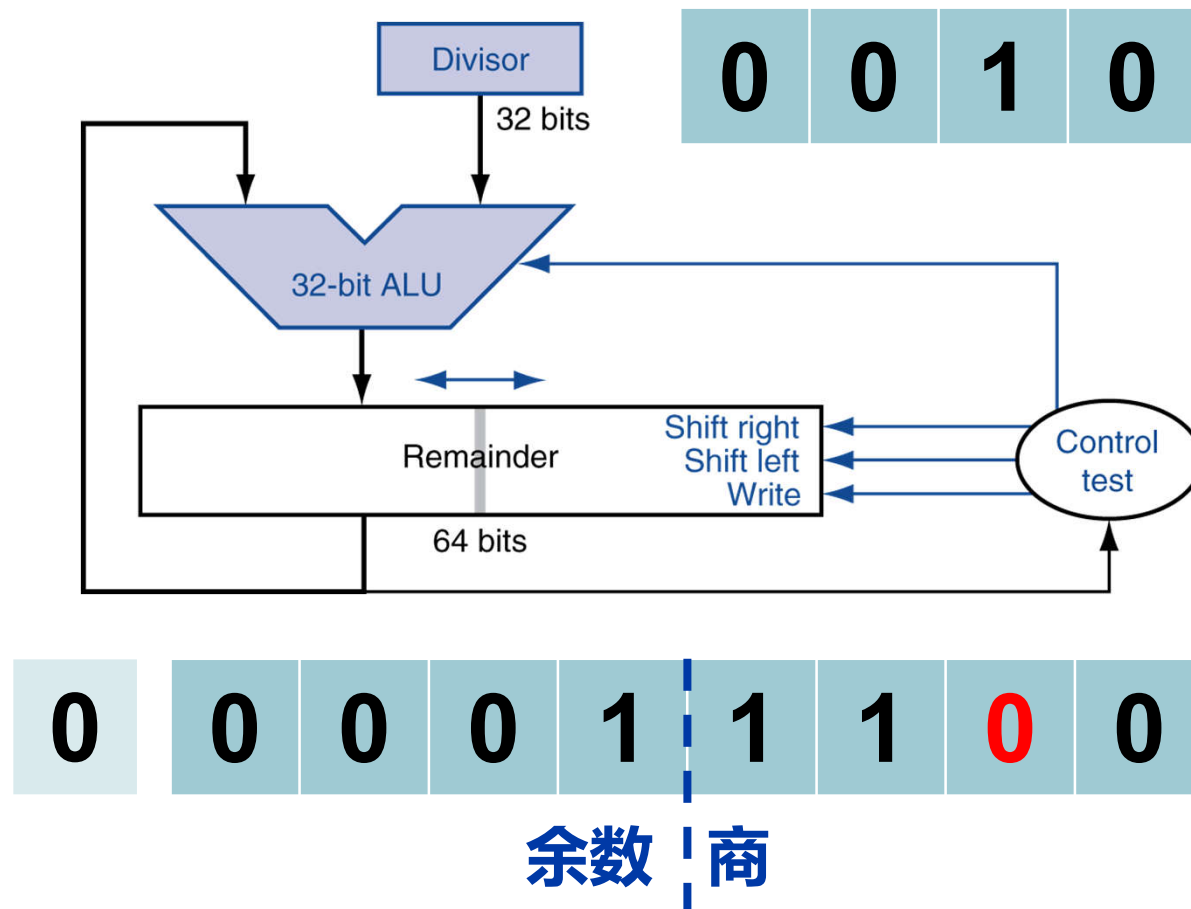
Optimized Integer Divider

Step 2.1 余数=余数-除数 试商



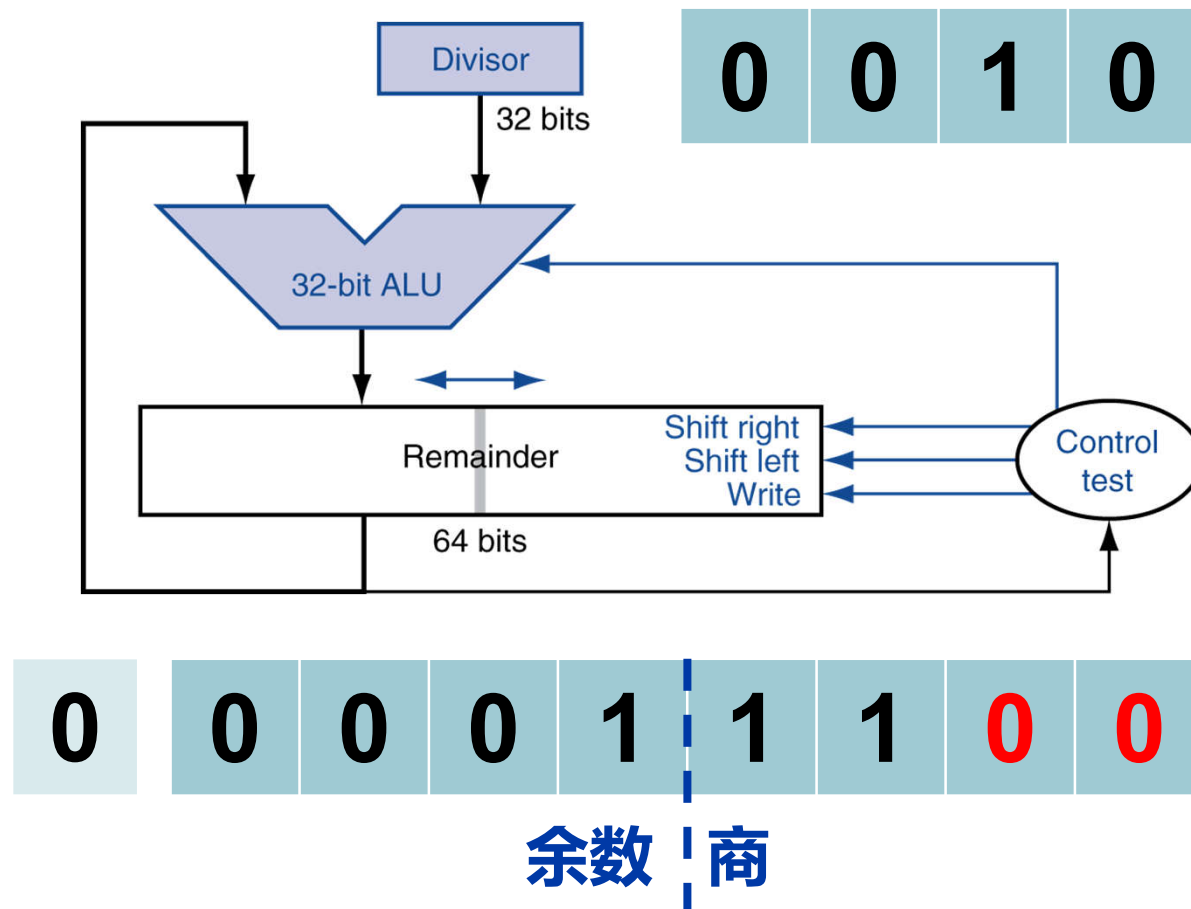
Optimized Integer Divider

Step 2.2 余数<0 恢复余数，余数和商 左移1位



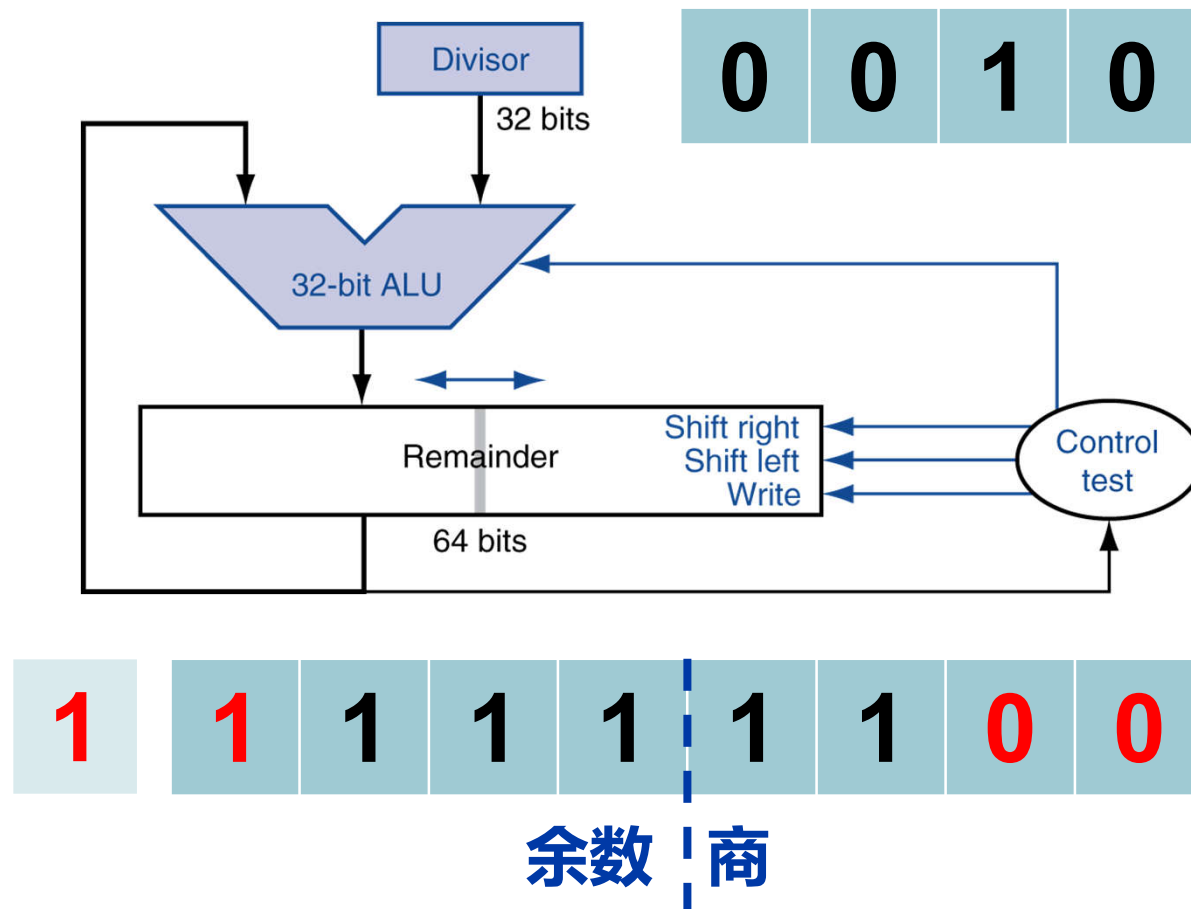
Optimized Integer Divider

Step 2.3 商的最低位置0



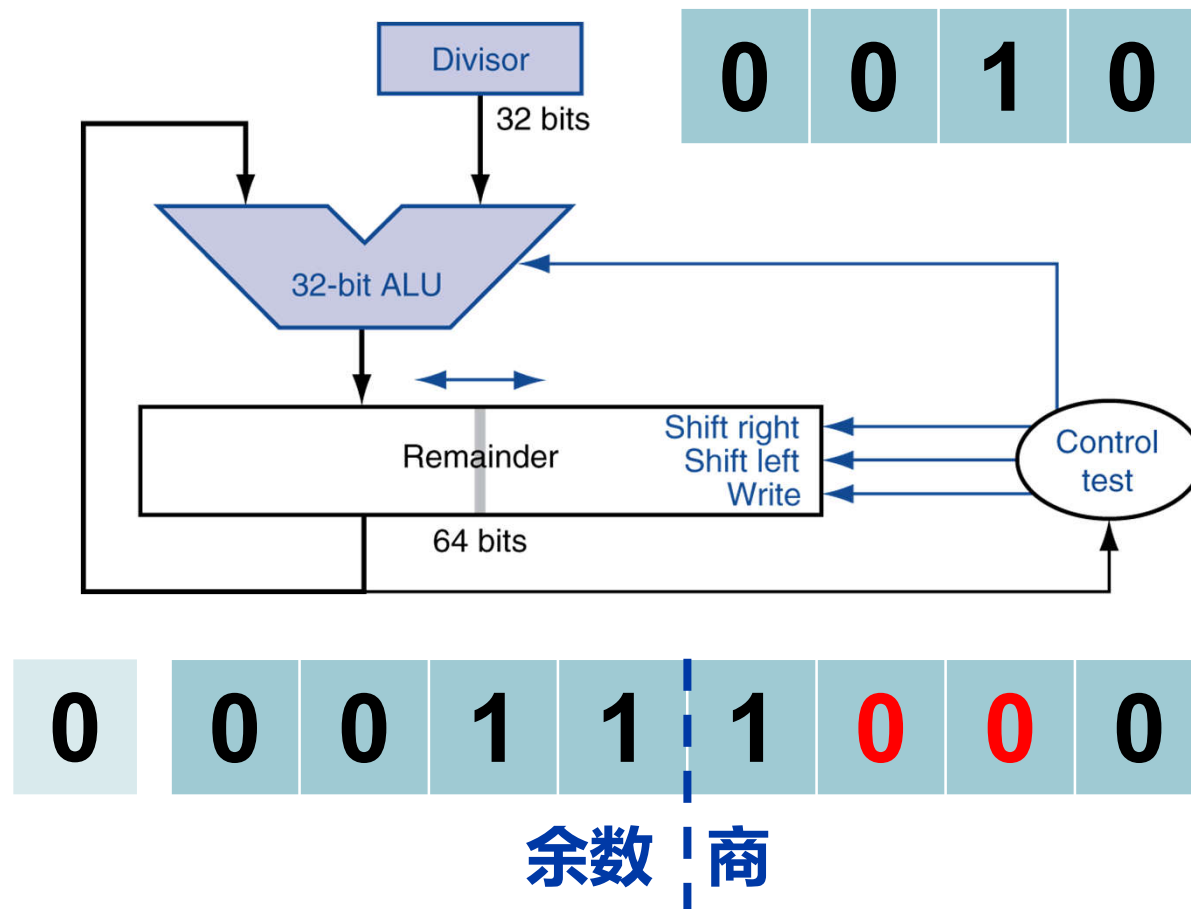
Optimized Integer Divider

Step 3.1 余数=余数-除数 试商



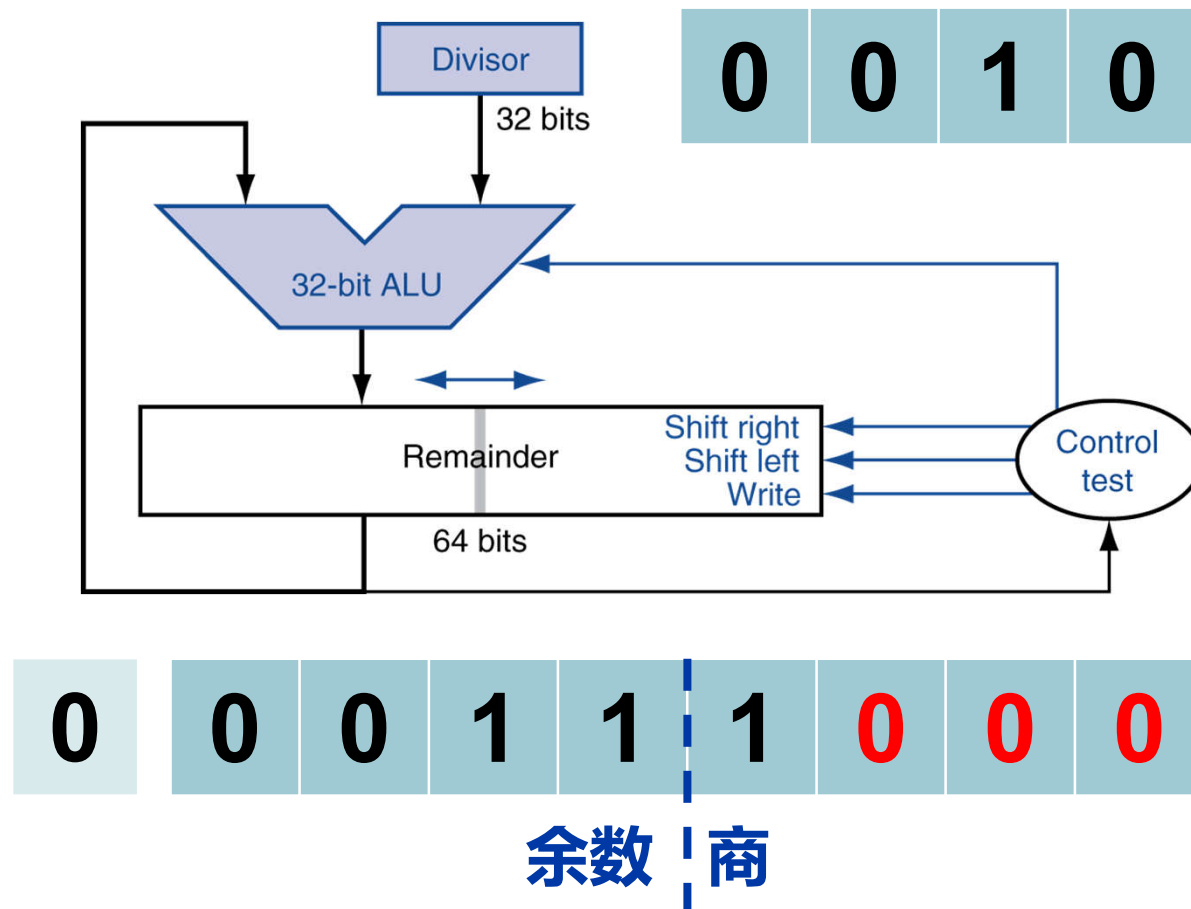
Optimized Integer Divider

Step 3.2 余数 < 0 恢复余数，余数和商 左移1位



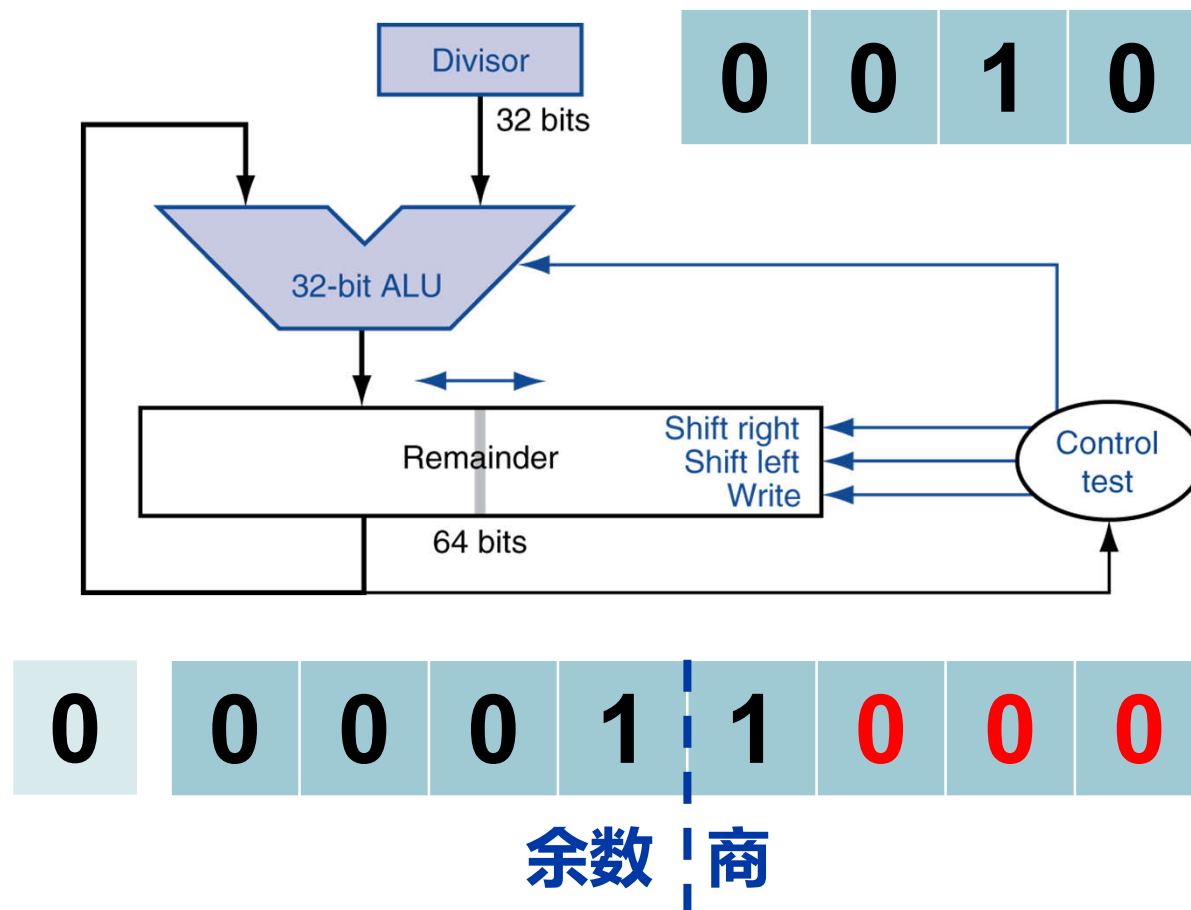
Optimized Integer Divider

Step 3 余数最低位置0



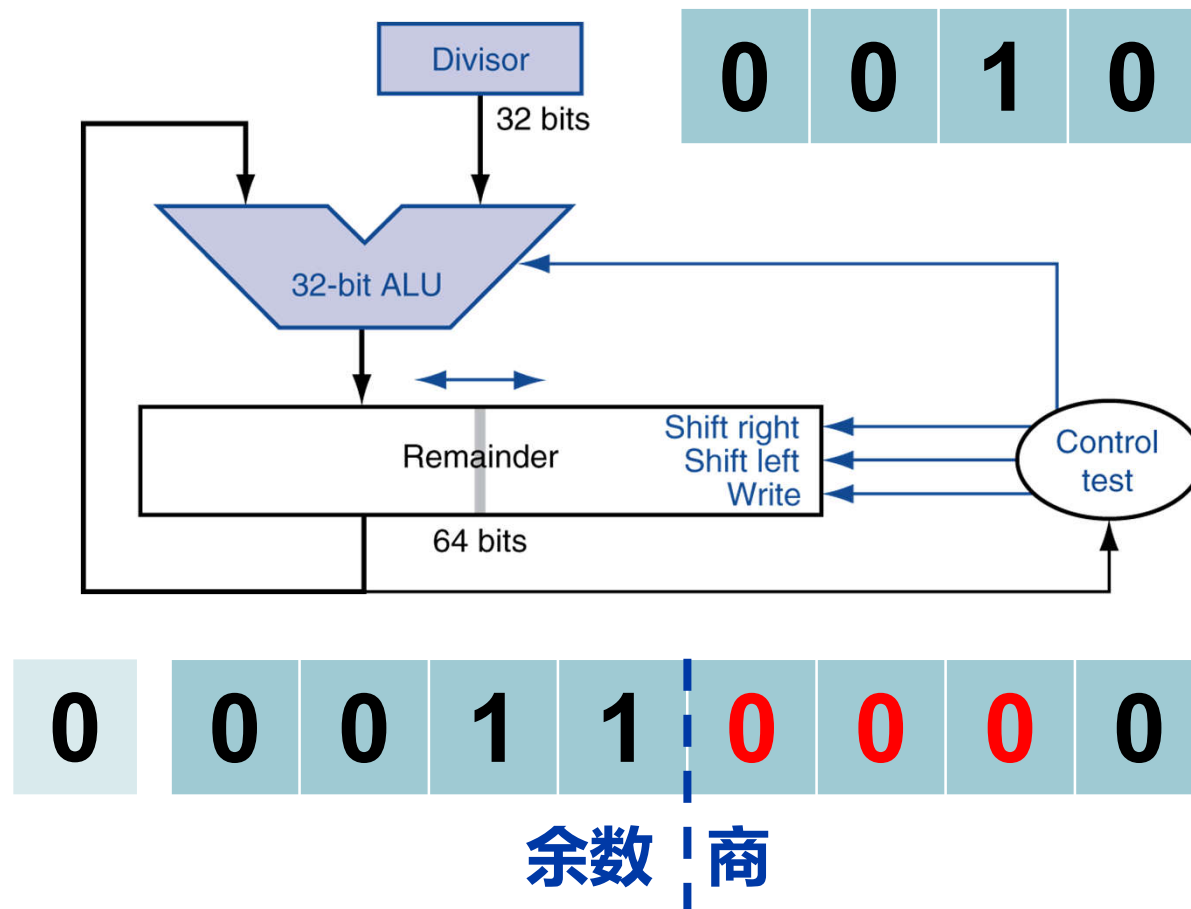
Optimized Integer Divider

Step 4.1 余数=余数-除数 试商



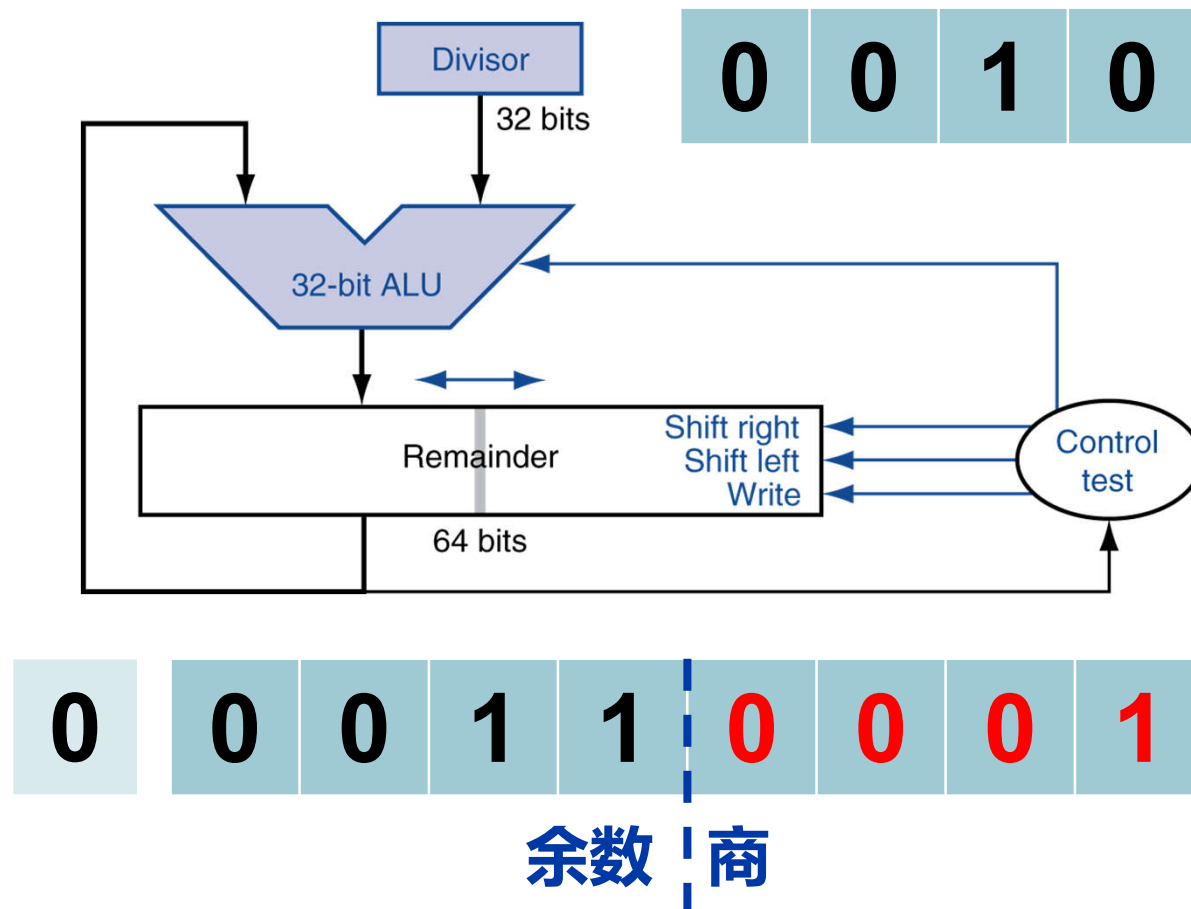
Optimized Integer Divider

Step 4.2 余数>0, 余数-商左移1位



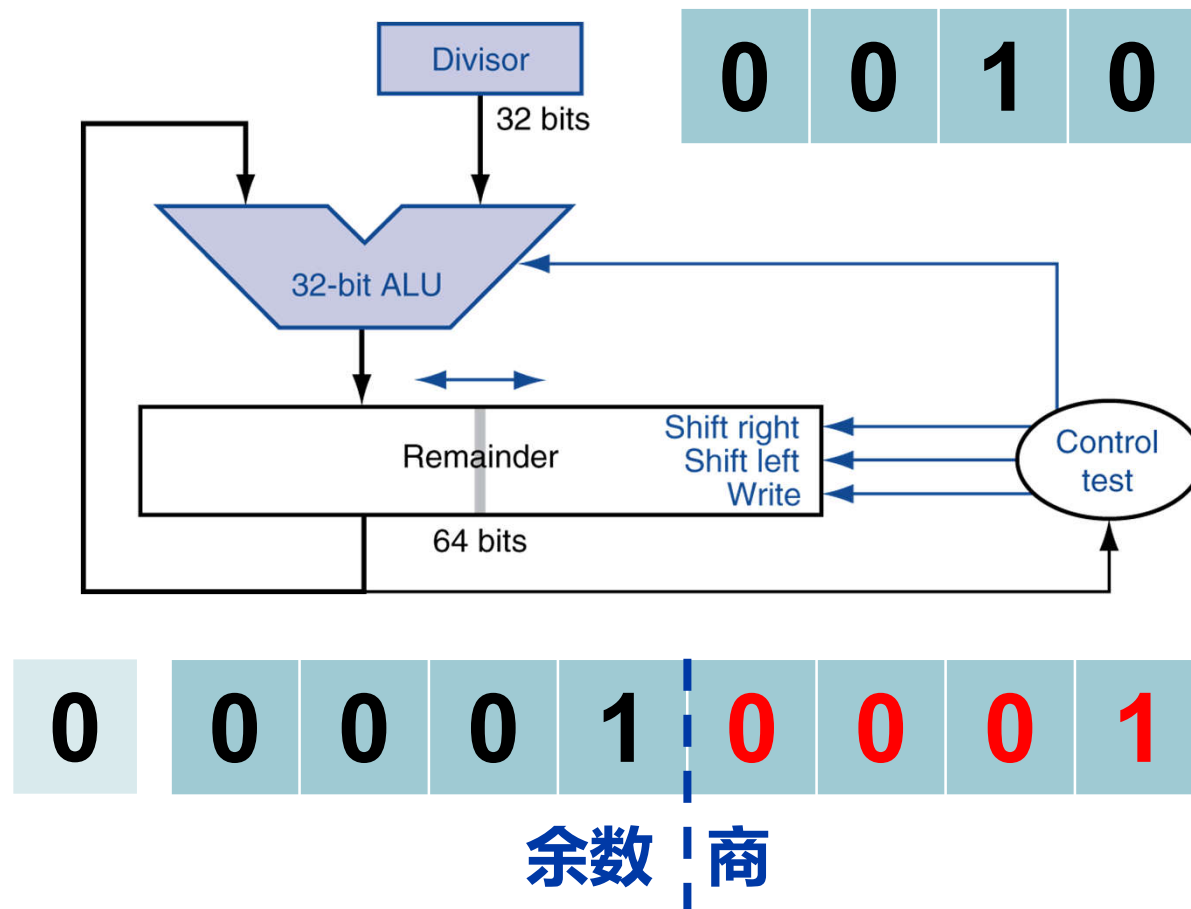
Optimized Integer Divider

Step 4.3 余数最低位置1



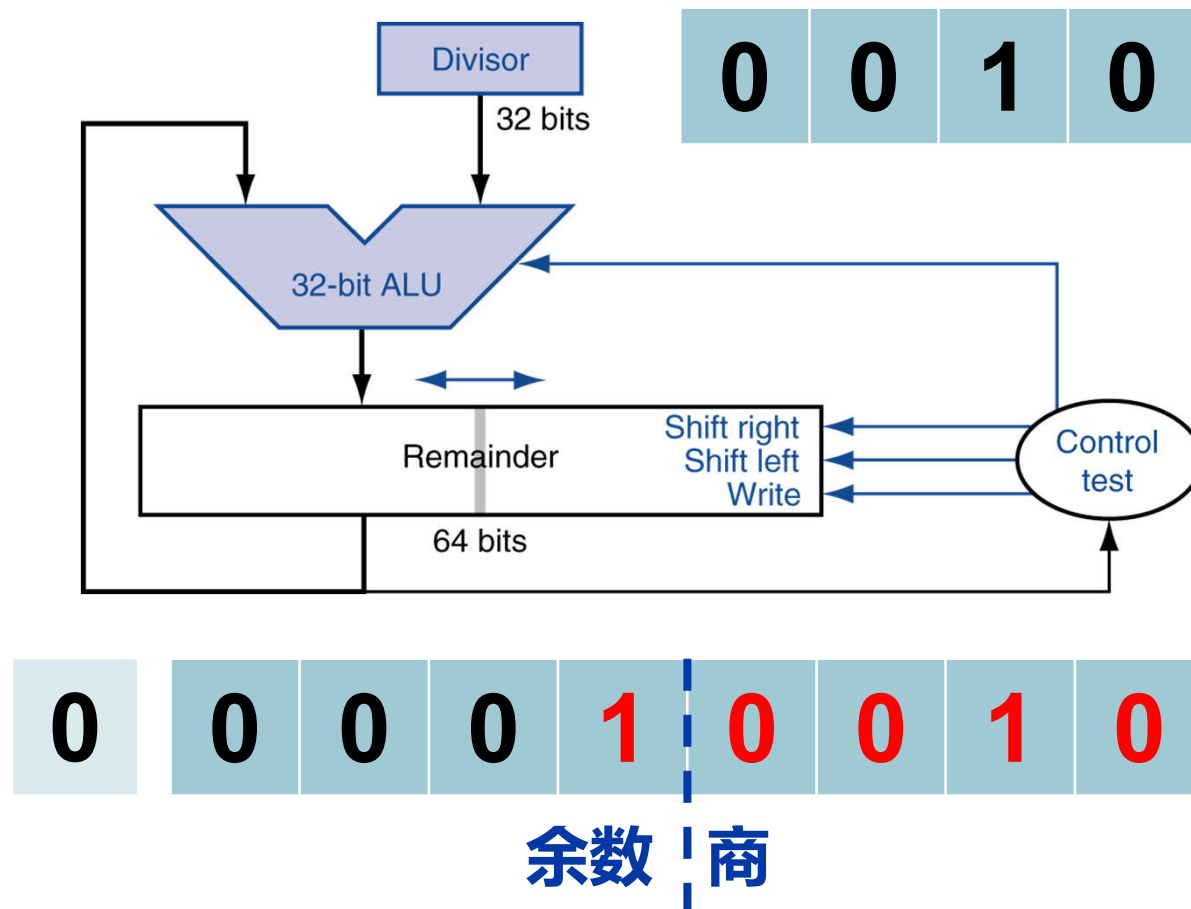
Optimized Integer Divider

Step 5.1 余数=余数-除数 试商



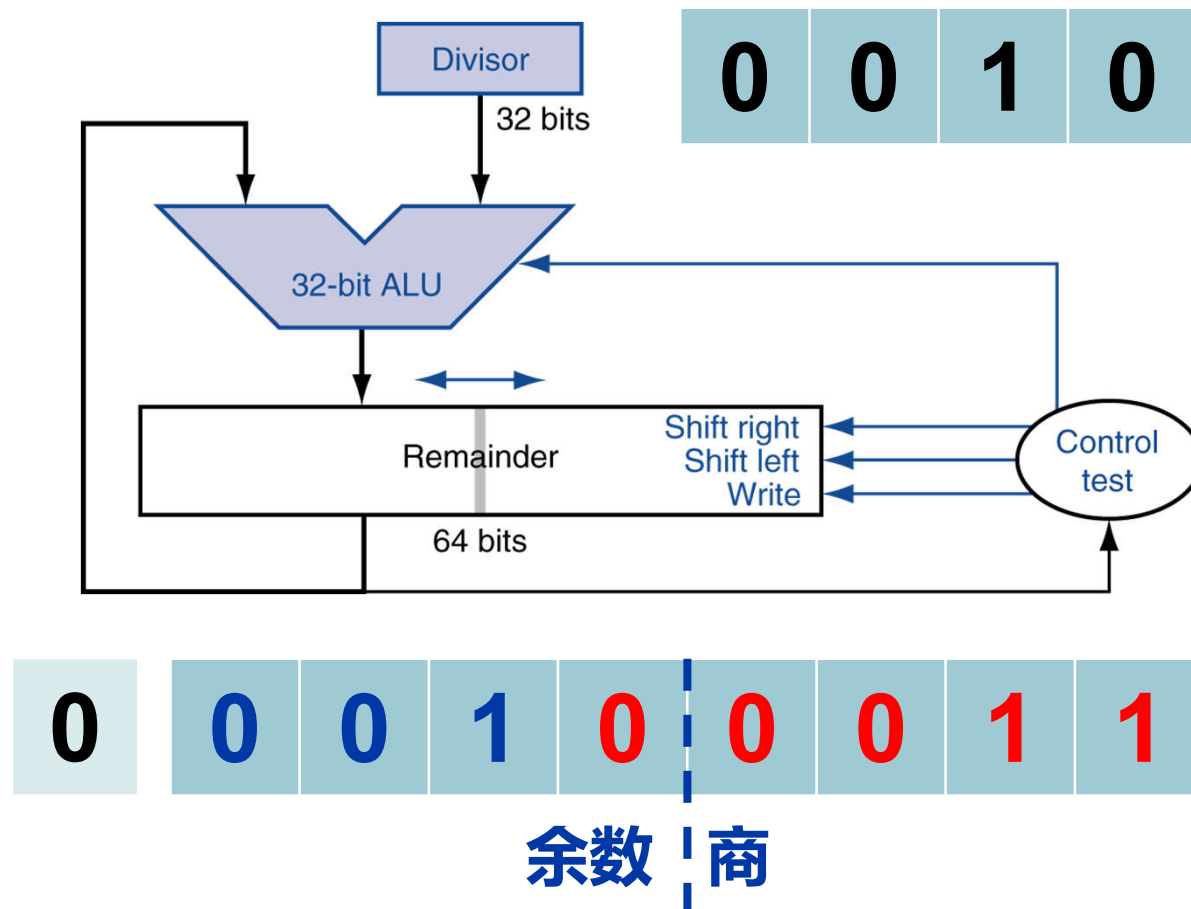
Optimized Integer Divider

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 1 110 0111 0 0000 1110 1 0000 1110
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 1 110 1110 0 0001 1100 0 0001 1100
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 1 111 1110 0 0011 1000 0 0011 1000
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 1000 0 0011 0000 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 0010 0 0010 0011

商为 $[0011]_2$ 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 R_i 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 $|y|$

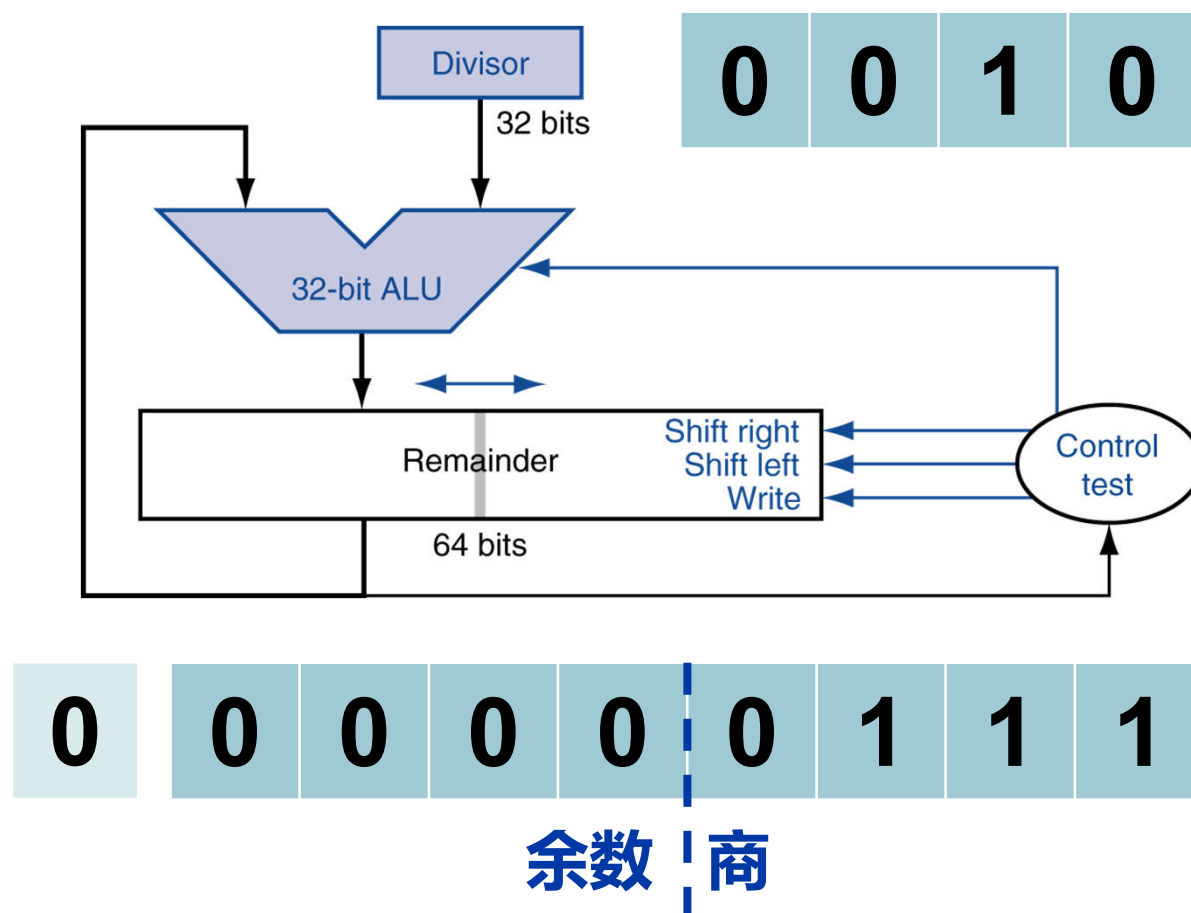
$$R_{i+1} = 2R_i + |y|$$

因此这两种算法计算结果是一样的！

加减交替法

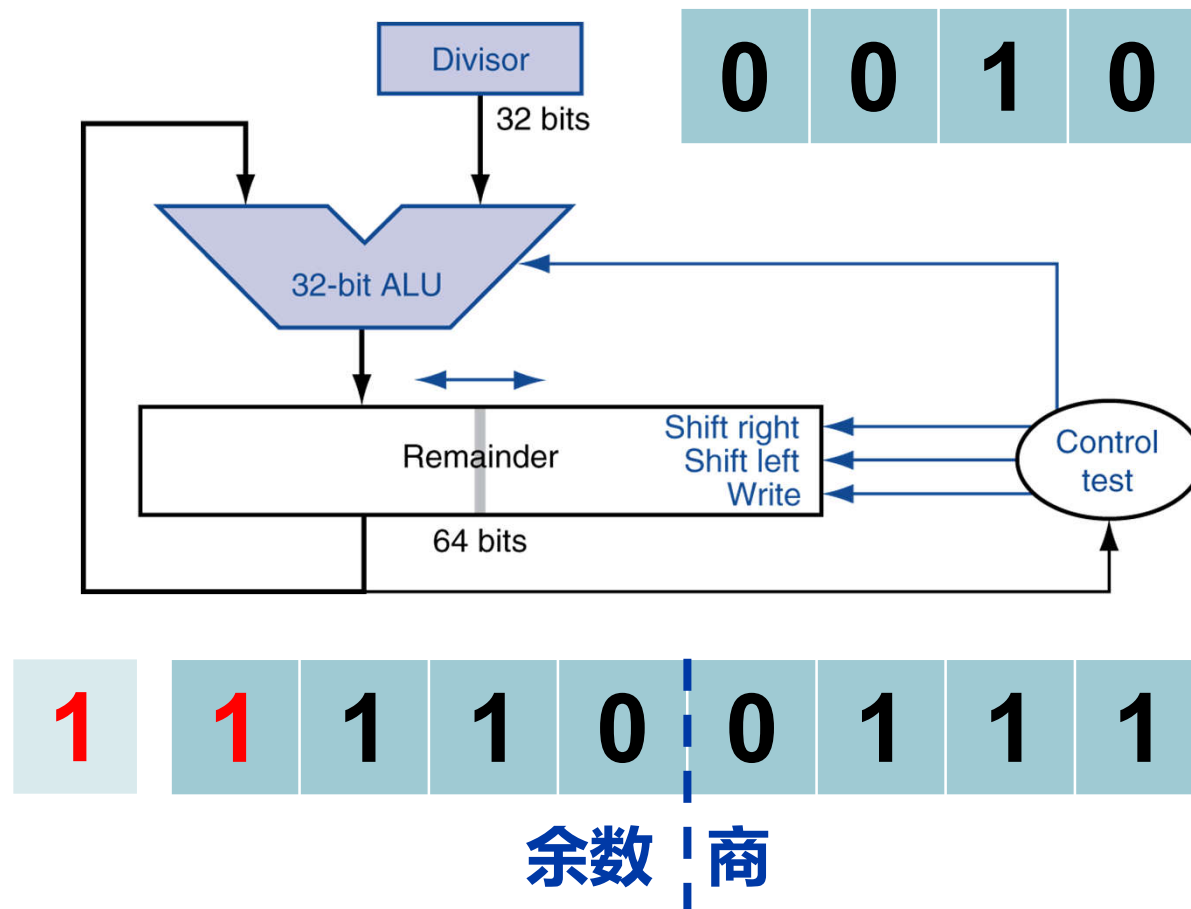
Step 0 初始化

被除数 = $[0111]_2$ 除数 = $[0010]_2$



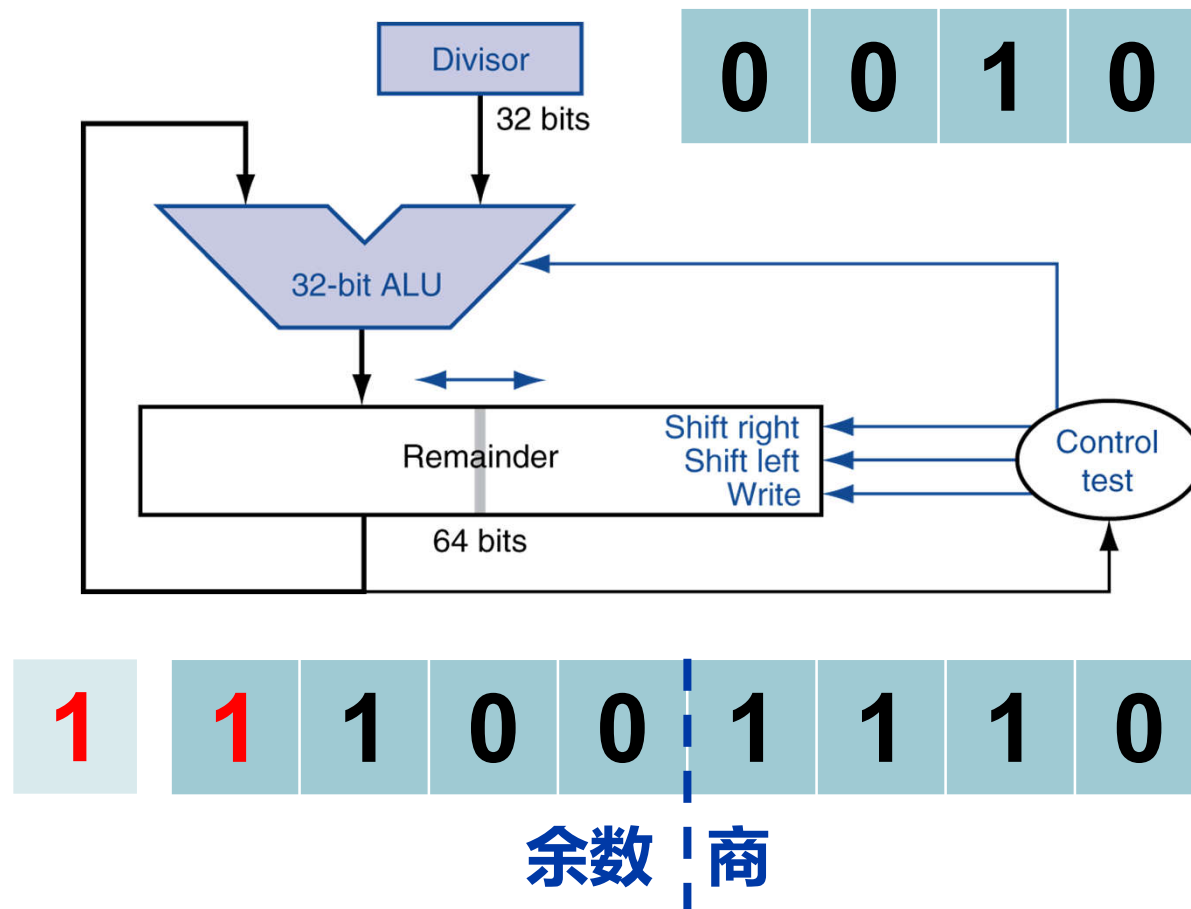
加减交替法

Step 1.1 余数>0, 余数-除数 试商



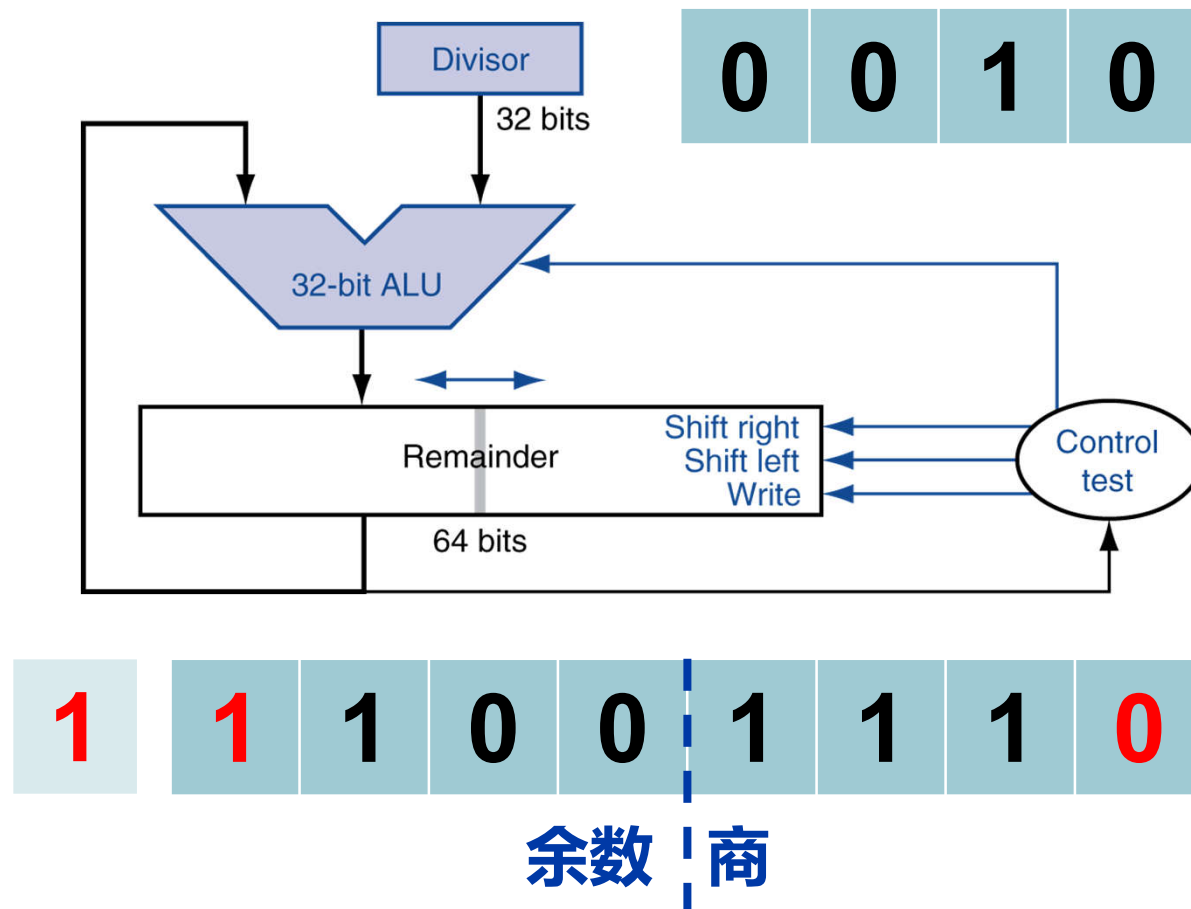
加减交替法

Step 1.2 余数和商 左移1位



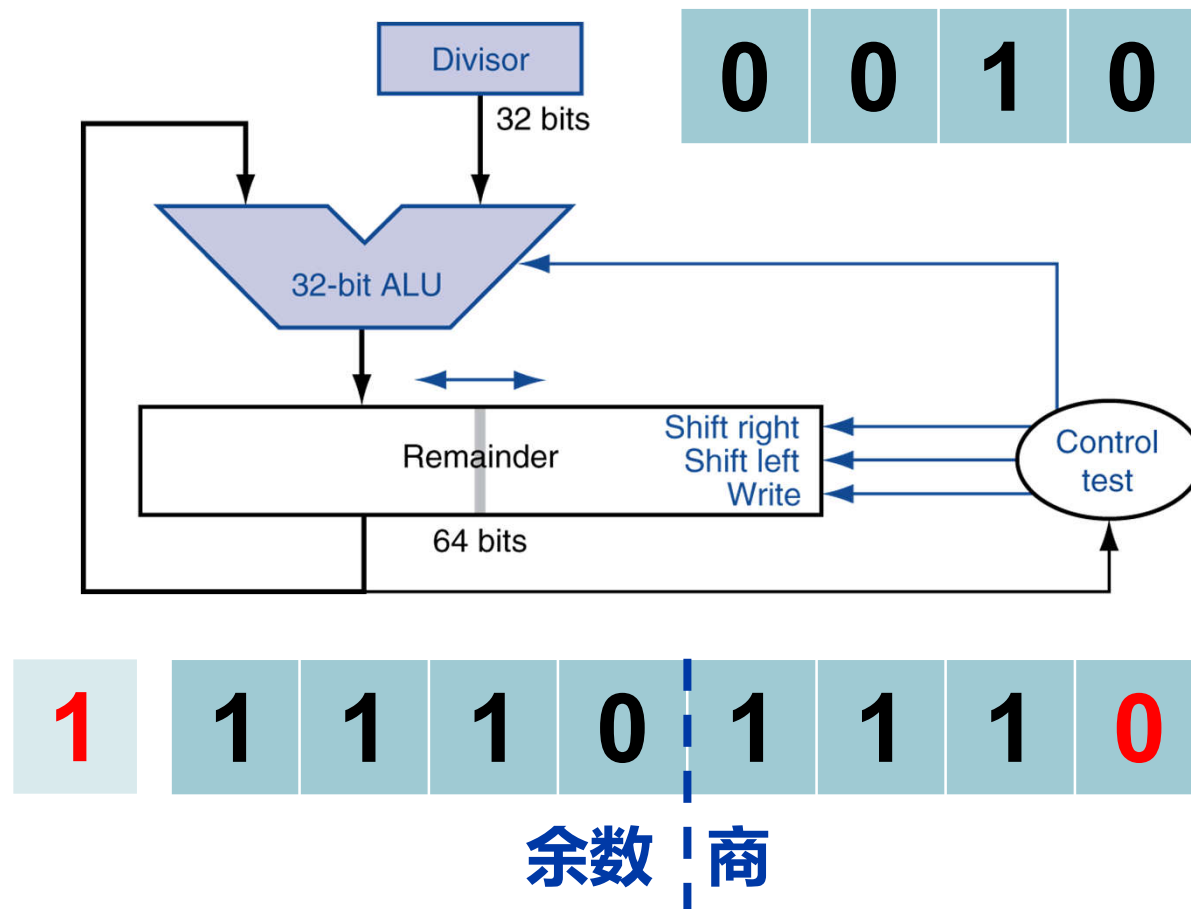
加减交替法

Step 1.3 余数 <0 , 商的最低位置0



加减交替法

Step 2.1 余数<0, 余数+除数 试商

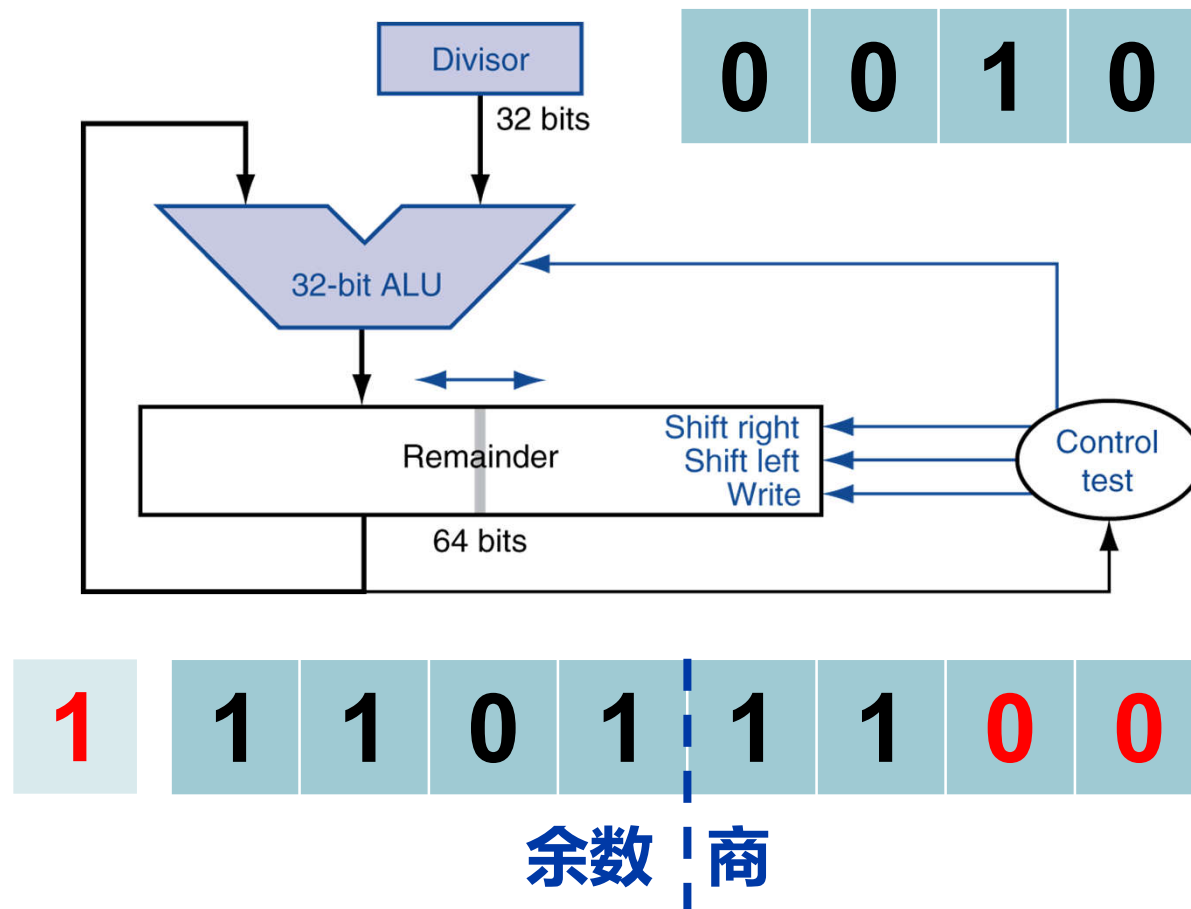


Step 2.2 余数和商 左移1位



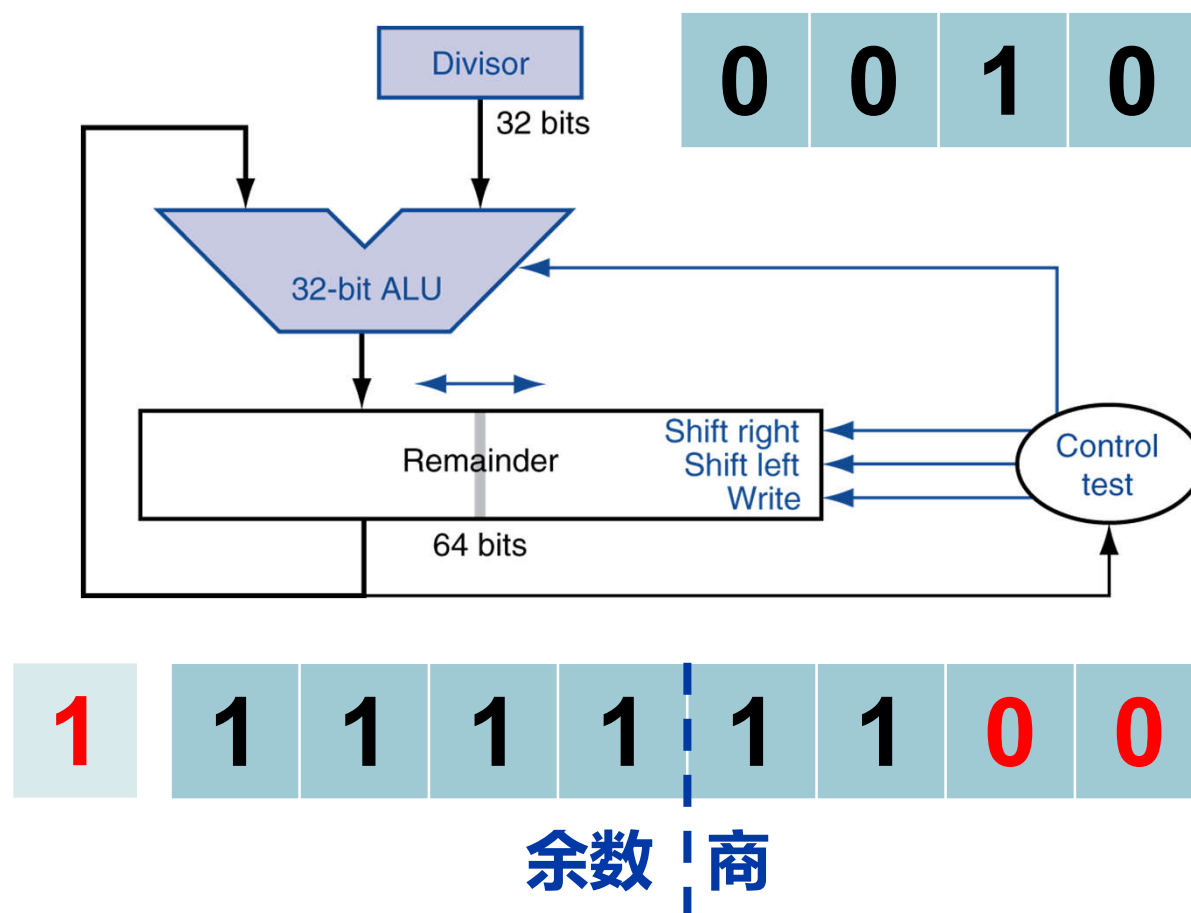
加减交替法

Step 2.3 余数 <0 , 商的最低位置0



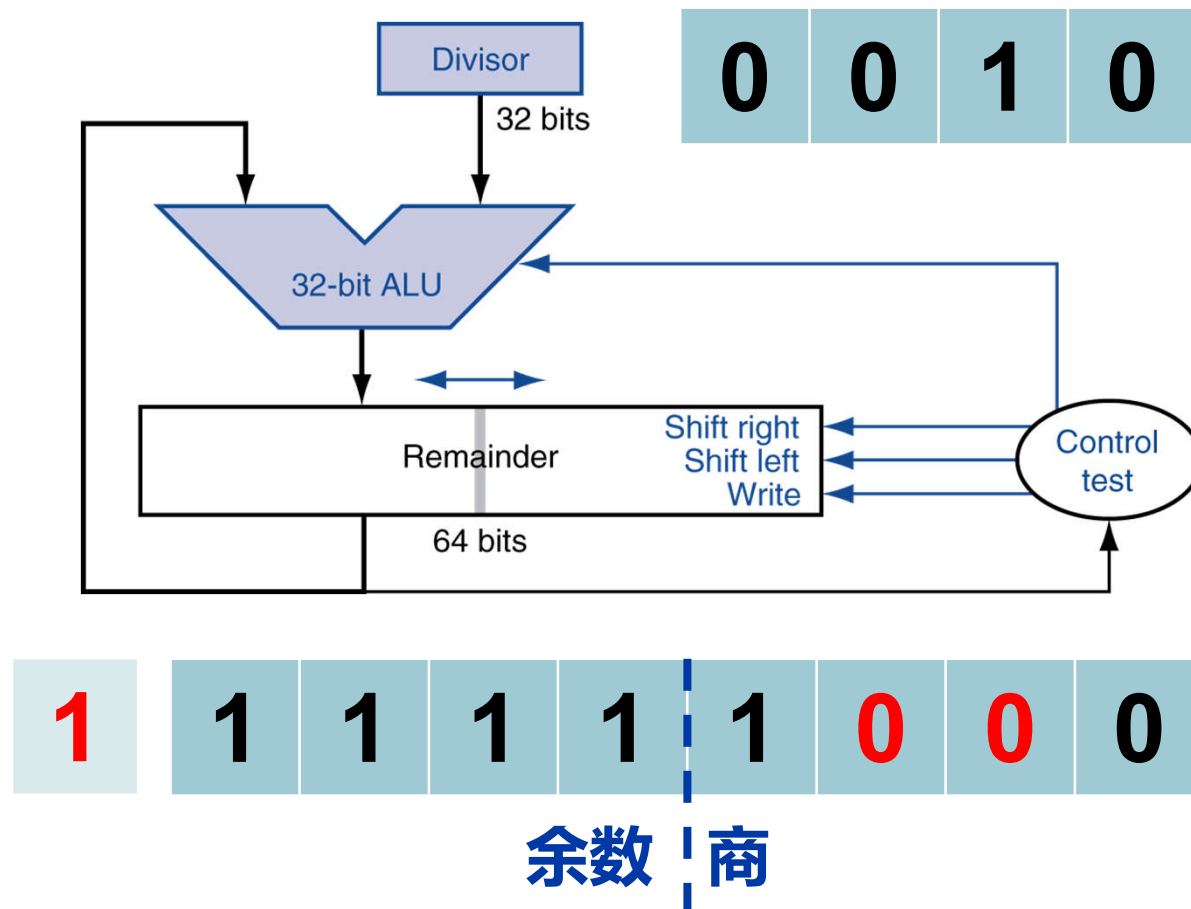
加减交替法

Step 3.1 余数<0, 余数+除数 试商



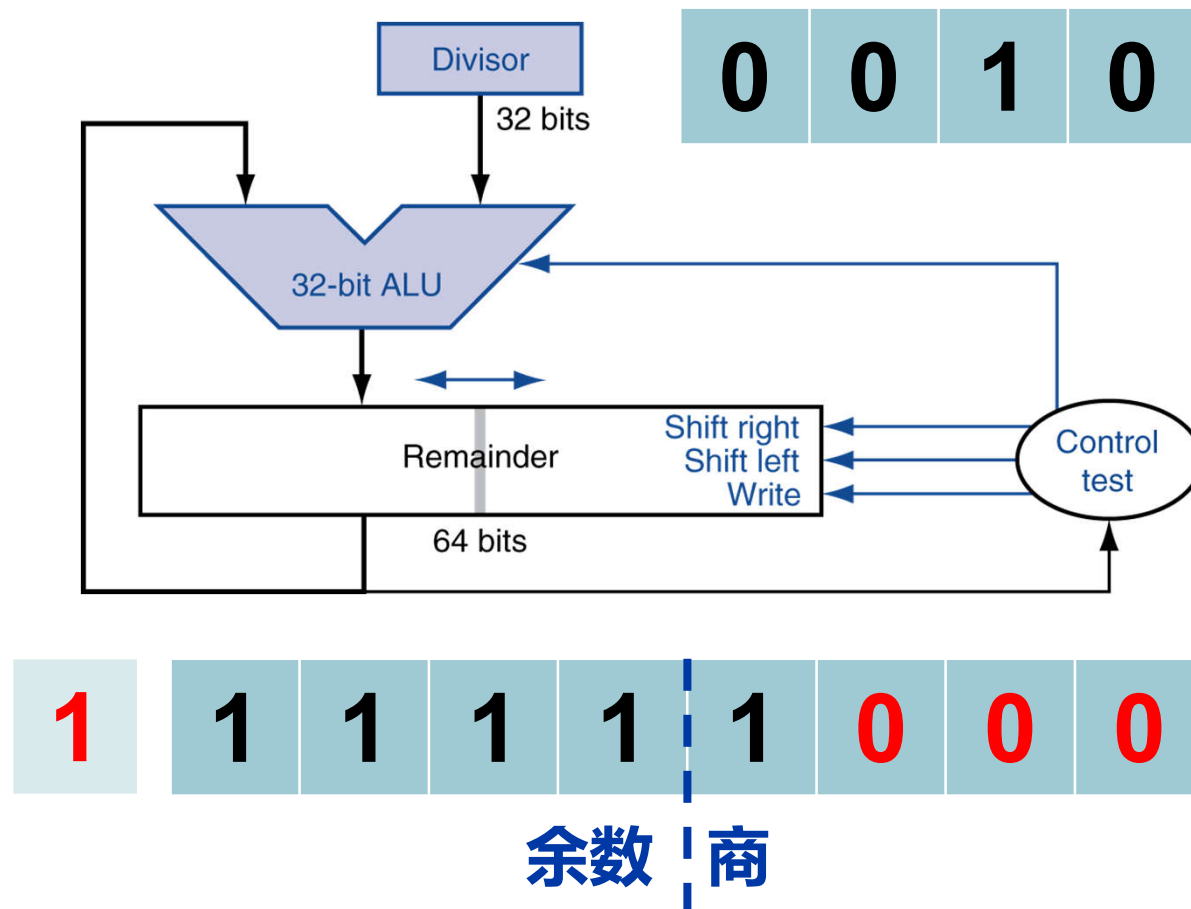
加减交替法

Step 3.2 余数和商 左移1位



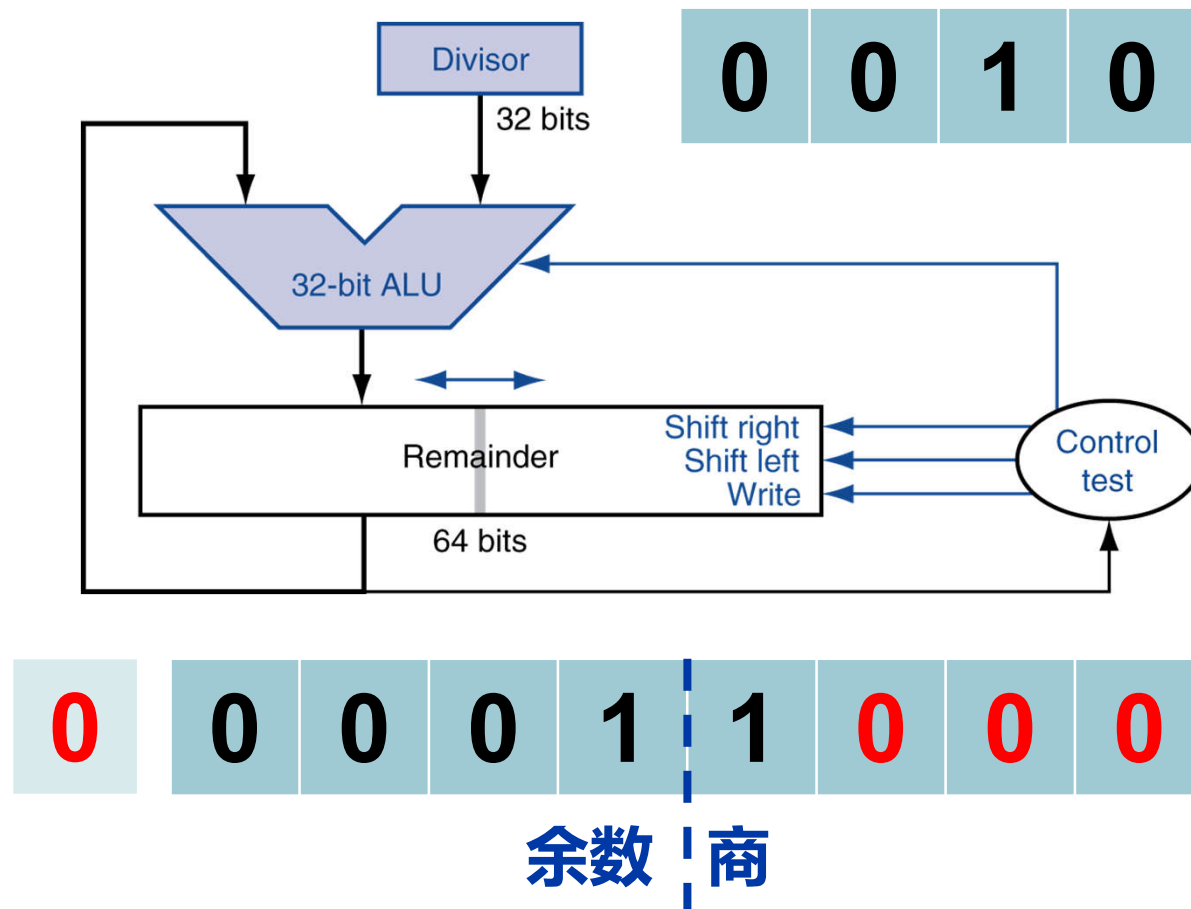
加减交替法

Step 3.3 余数 <0 , 商的最低位置0



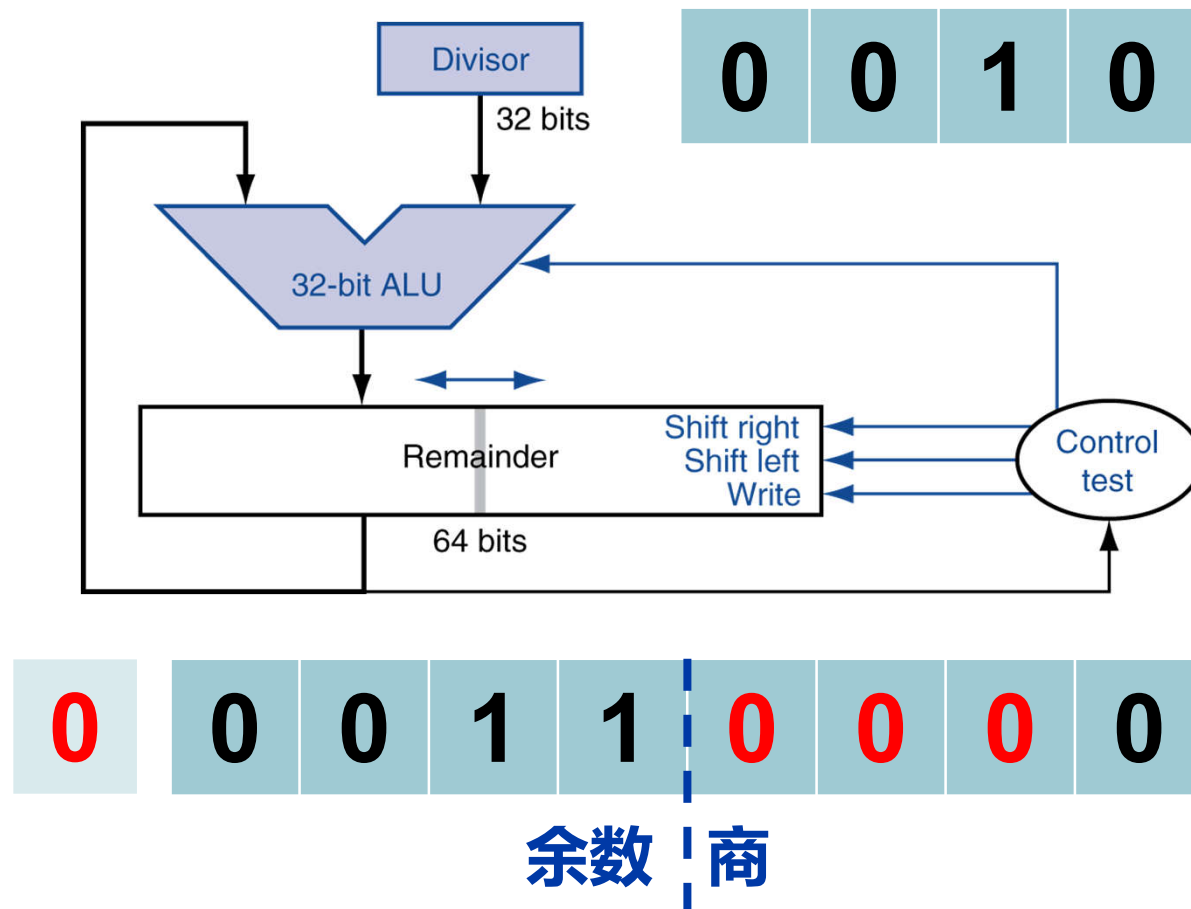
加减交替法

Step 4.1 余数<0, 余数+除数 试商



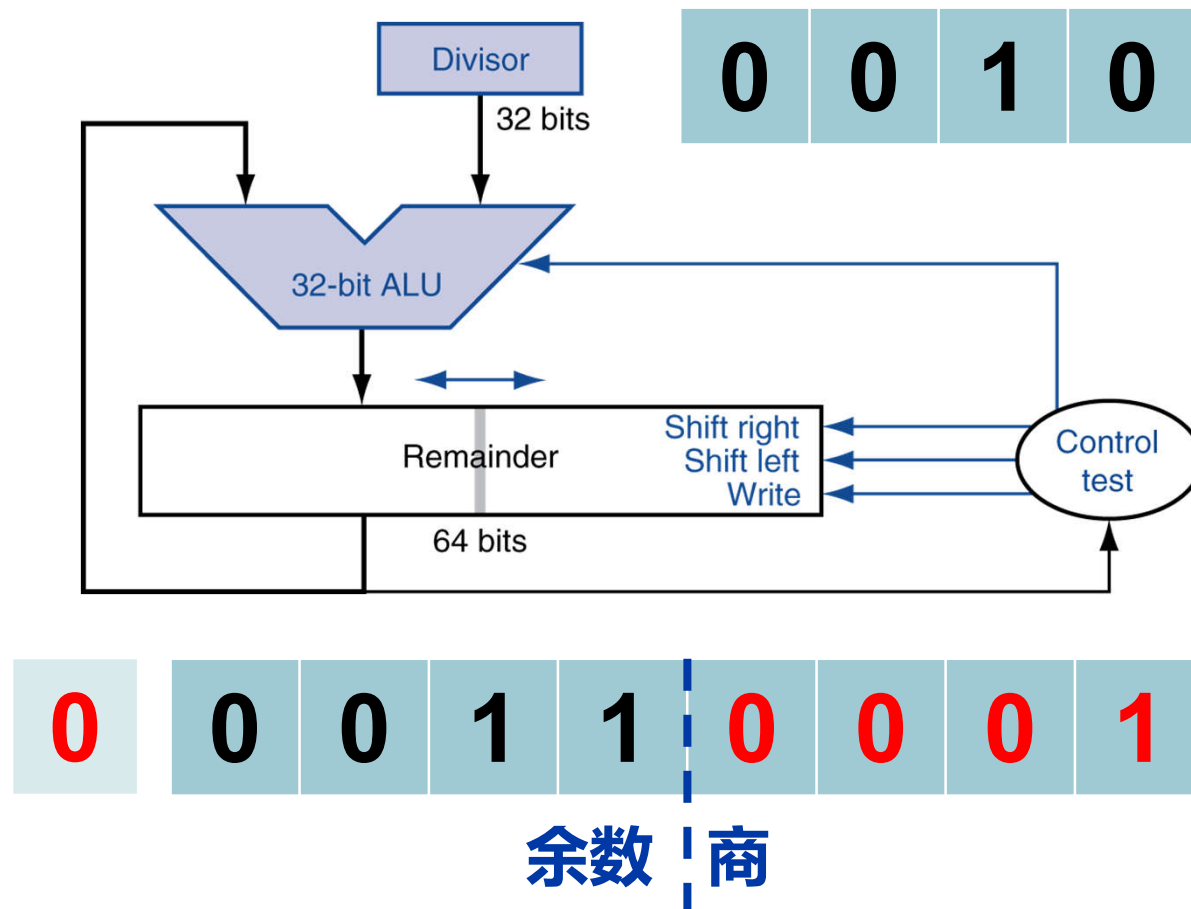
加减交替法

Step 4.2 余数和商 左移1位



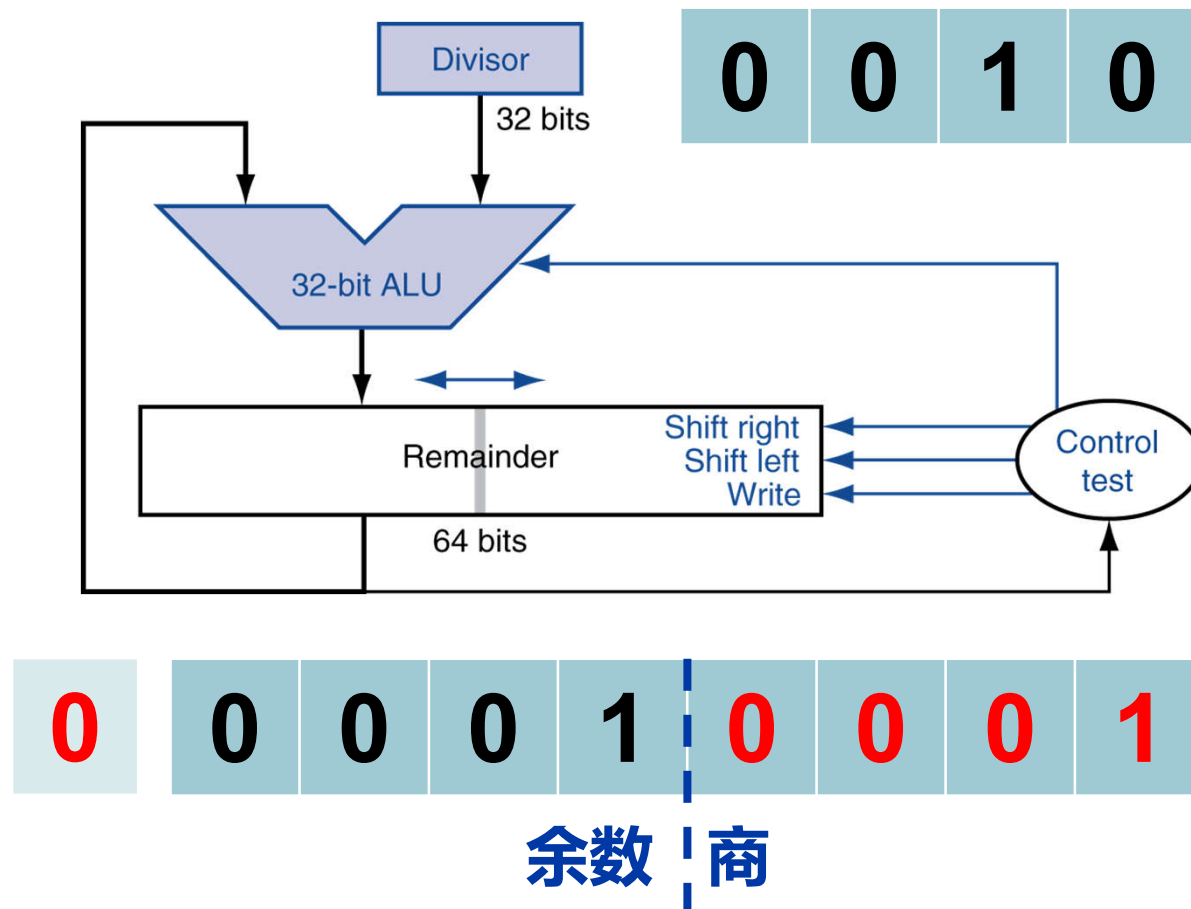
加减交替法

Step 4.3 余数 ≥ 0 , 商的最低位置1



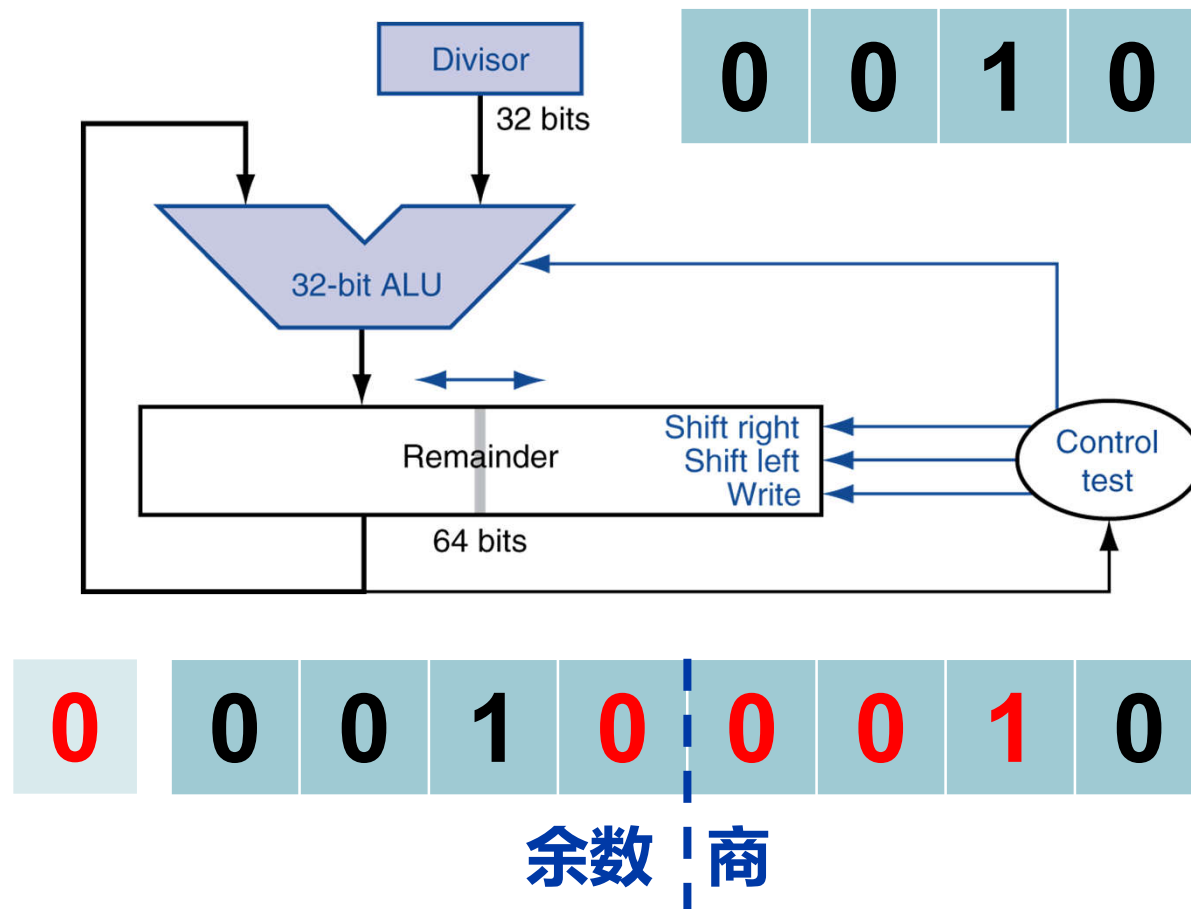
加减交替法

Step 5.1 余数>0, 余数-除数 试商



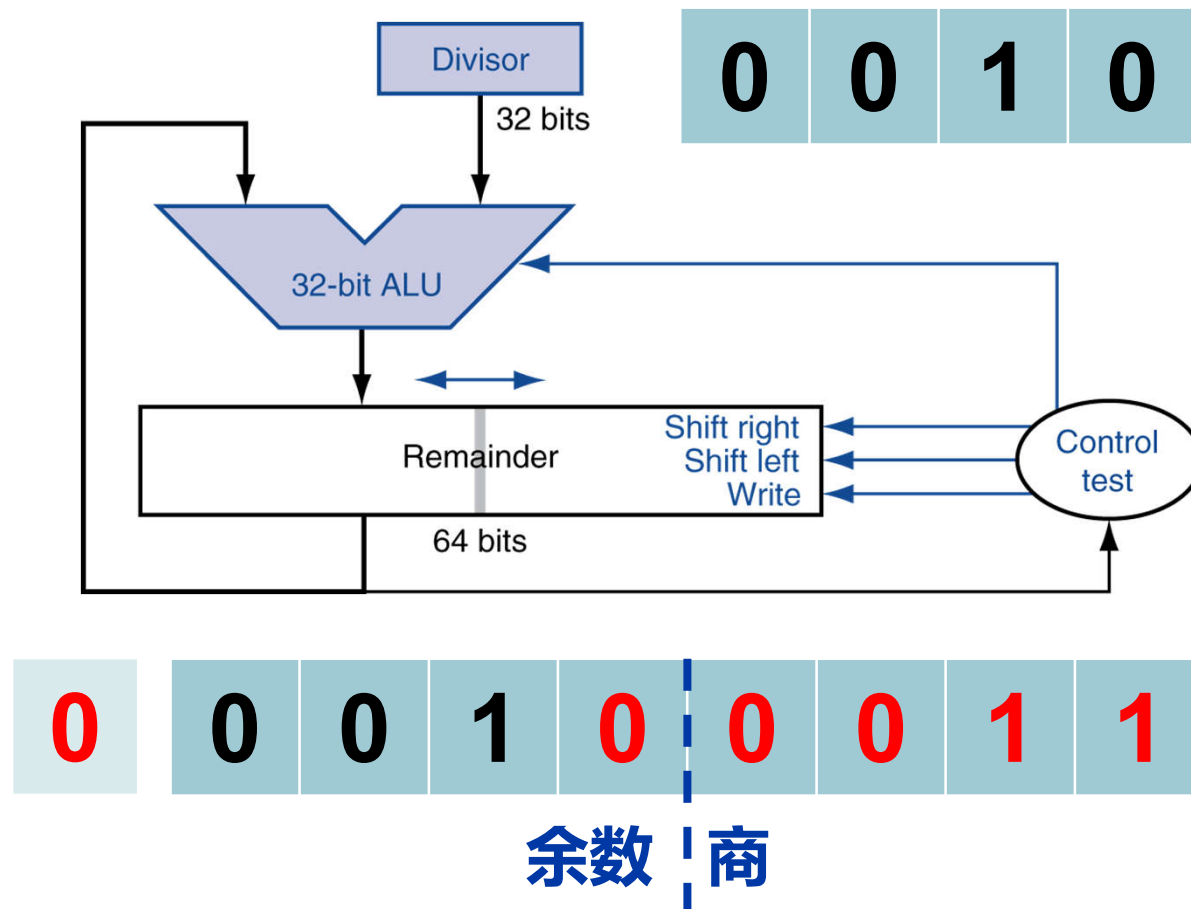
加减交替法

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 1110 0111 1 1100 1110 1 1100 1110
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 1110 1110 1 1101 1100 1 1101 1100
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 1111 1100 1 1111 1000 1 1111 1000
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 1000 0 0011 0000 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 0010 0010 0 0010 0011

商为 $[0011]_2$ 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 $R \geq 0$ $Q(n)=1$ 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

被除数x / 余数 r	商q	说明
$\begin{array}{r} 00.1001 \\ +[-y]_{\text{补}} 11.0101 \\ \hline 11.1110 \end{array}$	0.0000	$[x]_{\text{原}} = 00.1001,$ $[y]_{\text{补}} = 00.1011,$ $[-y]_{\text{补}} = 11.0101$ x减y 余数 $r_0 < 0$, r和q左移一位, 商0
$\begin{array}{r} \leftarrow 11.1100 \\ +[y]_{\text{补}} 00.1011 \\ \hline 00.0111 \end{array}$	0.0000	加y 余数 $r_1 > 0$ r和q左移一位, 商1
$\begin{array}{r} \leftarrow 00.1110 \\ +[-y]_{\text{补}} 11.0101 \\ \hline 00.0011 \end{array}$	0.0001	减y 余数 $r_2 > 0$ r和q左移一位, 商1
$\begin{array}{r} \leftarrow 00.0110 \\ +[-y]_{\text{补}} 11.0101 \\ \hline 11.1011 \end{array}$	0.0011	减y 余数 $r_3 < 0$ 商0, r和q左移一位
$\begin{array}{r} \leftarrow 11.0110 \\ +[y]_{\text{补}} 00.1011 \\ \hline 00.0001 \end{array}$	0.0110	加y 余数 $r_4 > 0$ 商1, 仅q左移一位
00.0001	0.1101	

得: $q = x/y = 0.1101$

余数 $r = 2^{-4} r_4 = 0.00000001$

Faster Division

- 除法运算无法像乘法一样并行化
 - 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, 第十二届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

$$[X]_{\text{原}} = 0.1001$$

$$[Y]_{\text{原}} = 0.1101$$

请采用加减交替法计算 $[X/Y]_{\text{原}}$ 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

被除数x / 余数 r	商q	说明
$\begin{array}{r} 00.1001 \\ +[-y]_{\text{补}} 11.0011 \\ \hline 11.1100 \end{array}$	0.0000	$[x]_{\text{原}} = 00.1001,$ $[y]_{\text{补}} = 00.1101,$ $[-y]_{\text{补}} = 11.0011$ x减y 余数 $r_0 < 0$, r和q左移一位, 商0
$\begin{array}{r} \leftarrow 11.1000 \\ +[y]_{\text{补}} 00.1101 \\ \hline 00.0101 \end{array}$	0.0000	加y 余数 $r_1 > 0$ r和q左移一位, 商1
$\begin{array}{r} \leftarrow 00.1010 \\ +[-y]_{\text{补}} 11.0011 \\ \hline 11.1101 \end{array}$	0.0001	减y 余数 $r_2 < 0$ r和q左移一位, 商0
$\begin{array}{r} \leftarrow 11.1010 \\ +[y]_{\text{补}} 00.1101 \\ \hline 00.0111 \end{array}$	0.0010	减y 余数 $r_3 > 0$ 商1, r和q左移一位
$\begin{array}{r} \leftarrow 00.1110 \\ +[-y]_{\text{补}} 11.0011 \\ \hline 00.0001 \end{array}$	0.0101	加y 余数 $r_4 > 0$ 商1, 仅q左移一位
$\begin{array}{r} 00.0001 \end{array}$	0.1011	

得: $q = x/y = 0.1011$

余数 $r = 2^{-4} r_4 = 0.00000001$