FSE598 前沿计算技术

模块 1　计算思维
单元 1　计算机系统设计
第 3 讲　算术与逻辑单元

本讲座的英文版内容基于教材：
The English version of the lectures are partly based on the book:
Patterson and Hennessy, Computer Organization and Design: The Hardware Software Interface

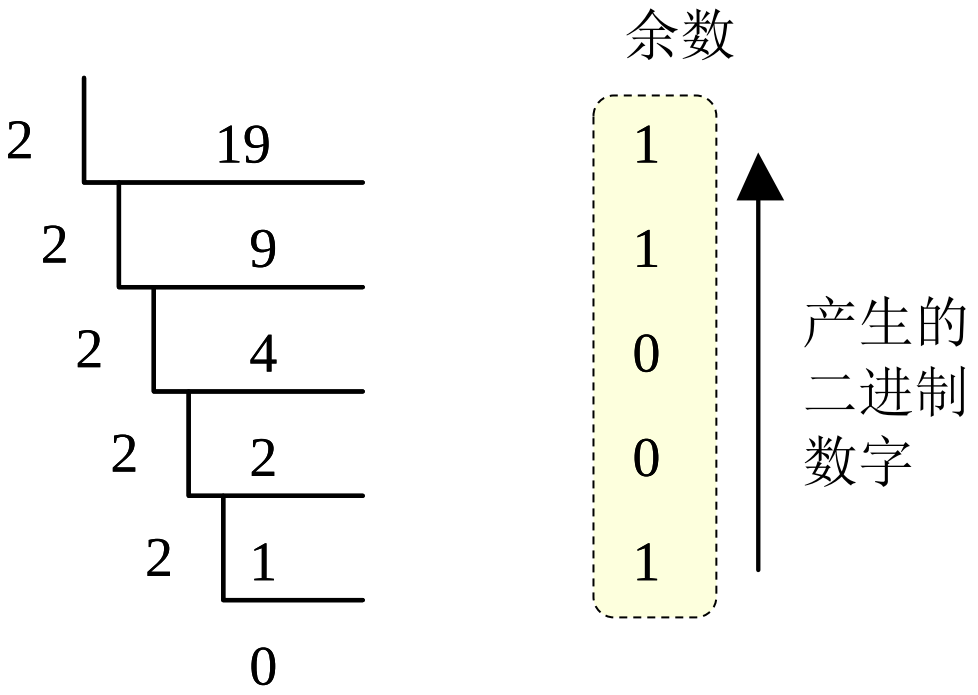
本讲提要

学习

- 二进制数运算与硬件设计
- 加减
- 其他指令
- ALU 设计
- 乘除法逻辑设计

二进制数和加法

❑ 十进制转二进制



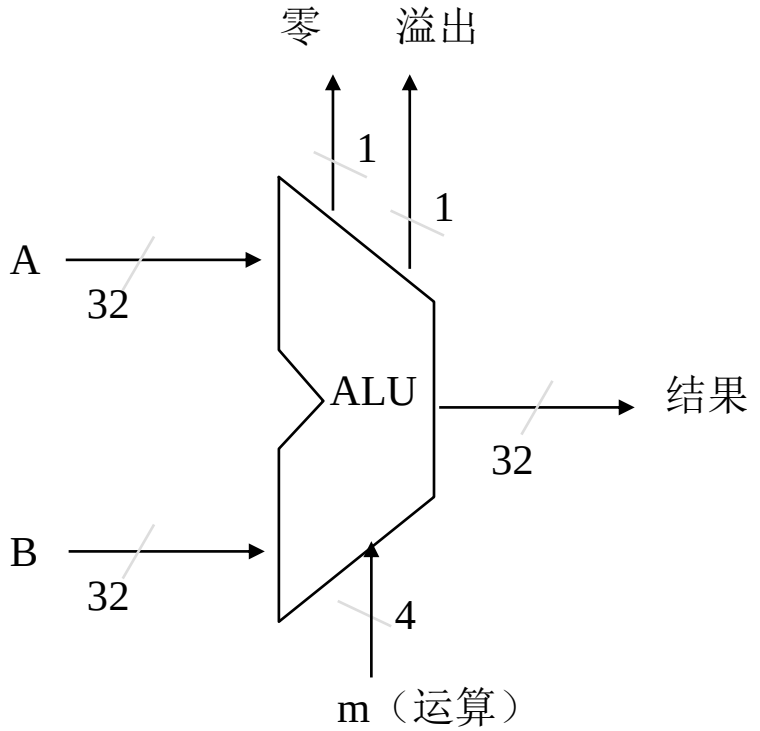
19 → 10011

❑ 二进制加法

$$
\begin{array}{r}
0000\,10110 \\
+\,0\,11110111 \\
\hline
100001101
\end{array}
$$

MIPS 算数逻辑单元(ALU)

- 必须支持的指令集架构中的算数/逻辑运算



指令示例

add：加 `add $10, $11, $12 # $10 ← $11 + $12`

sub：减 `sub $10, $11, $12 # $10 ← $11 - $12`

mult：乘 `mult $10, $11, $12 # $10 ← $11 * $12`

div：除 `div $10, $11, $12 # $10 ← $11 / $12`

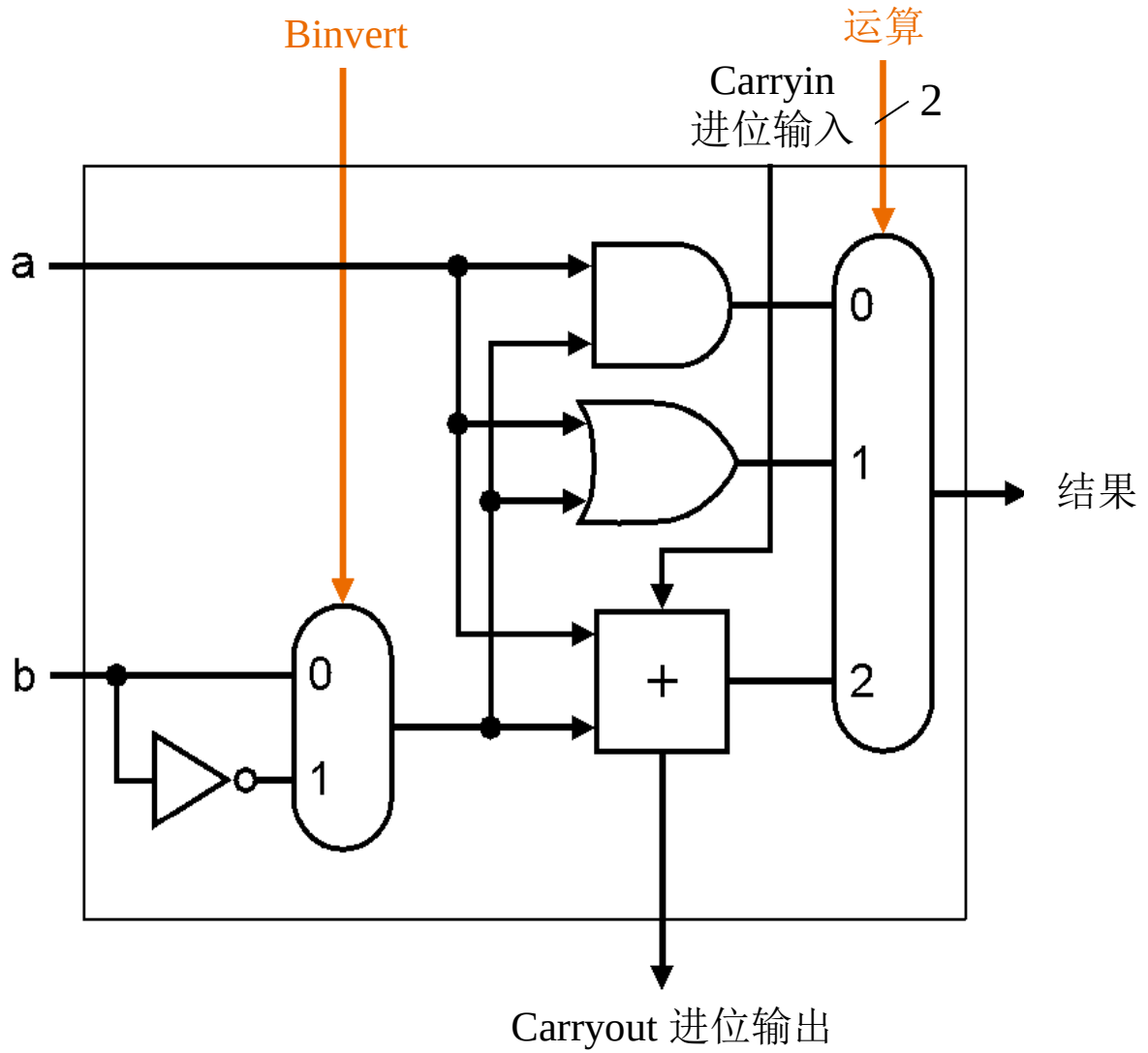
and, or, not, xor: 与、或、非、异或

beq：相等时分支 `beq $14, $15, L # if ($14 == $15) go to L`

bne：不相等时分支 `bne $14, $15, L # if ($14 != $15) go to L`

slt：小于则置位 `slt $14, $15, $16 # $14 = 1 if ($15 < $16) else $14 = 0`

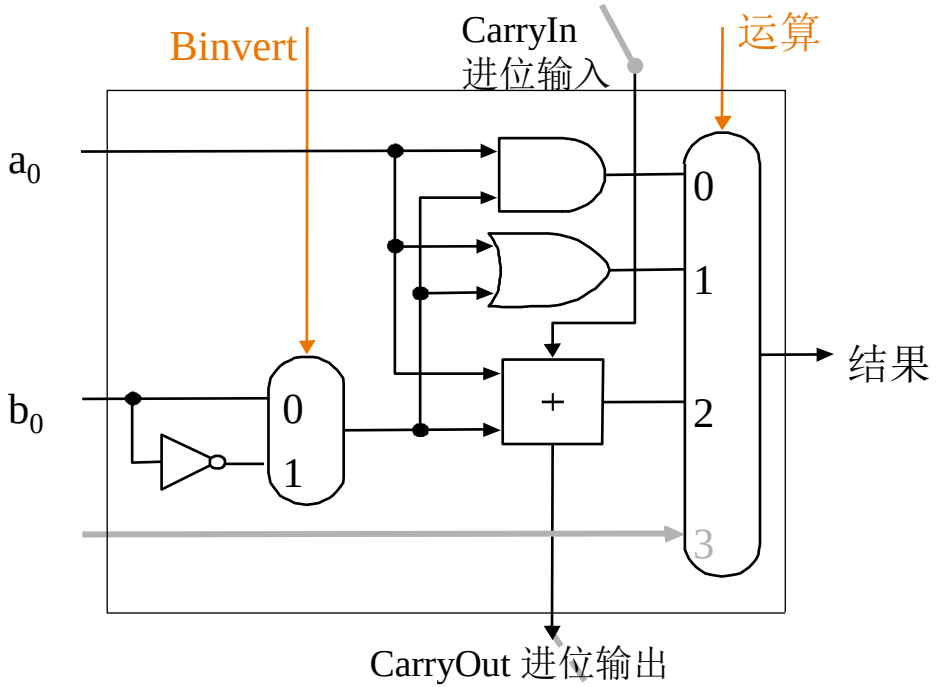
包含三个运算的一位 ALU



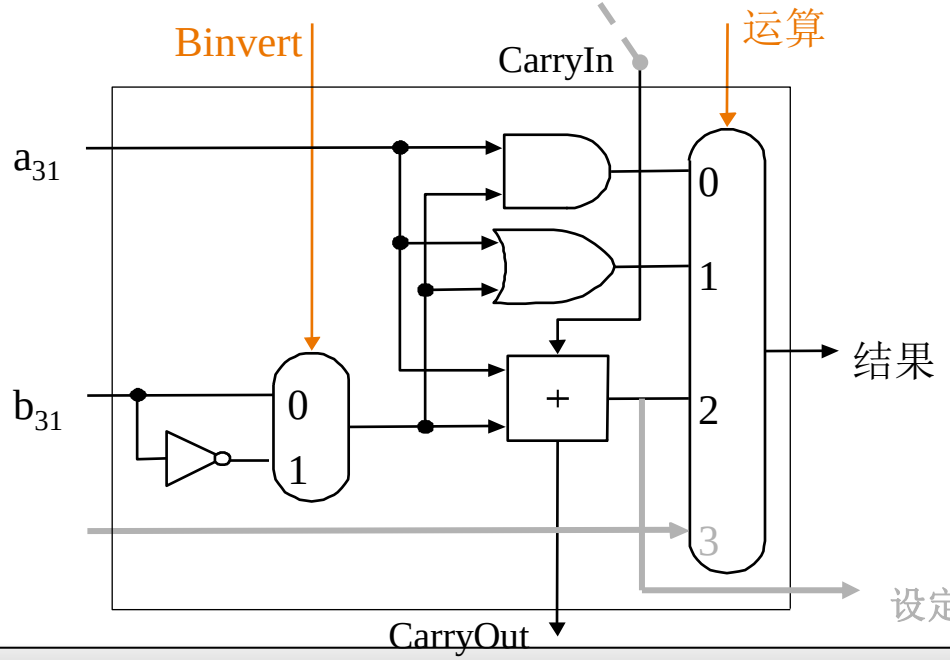
资料来源：Patterson 和 Hennessy，《计算机组成与设计：硬件/软件接口》(Computer Organization and Design: The Hardware Software Interface)

32位 ALU

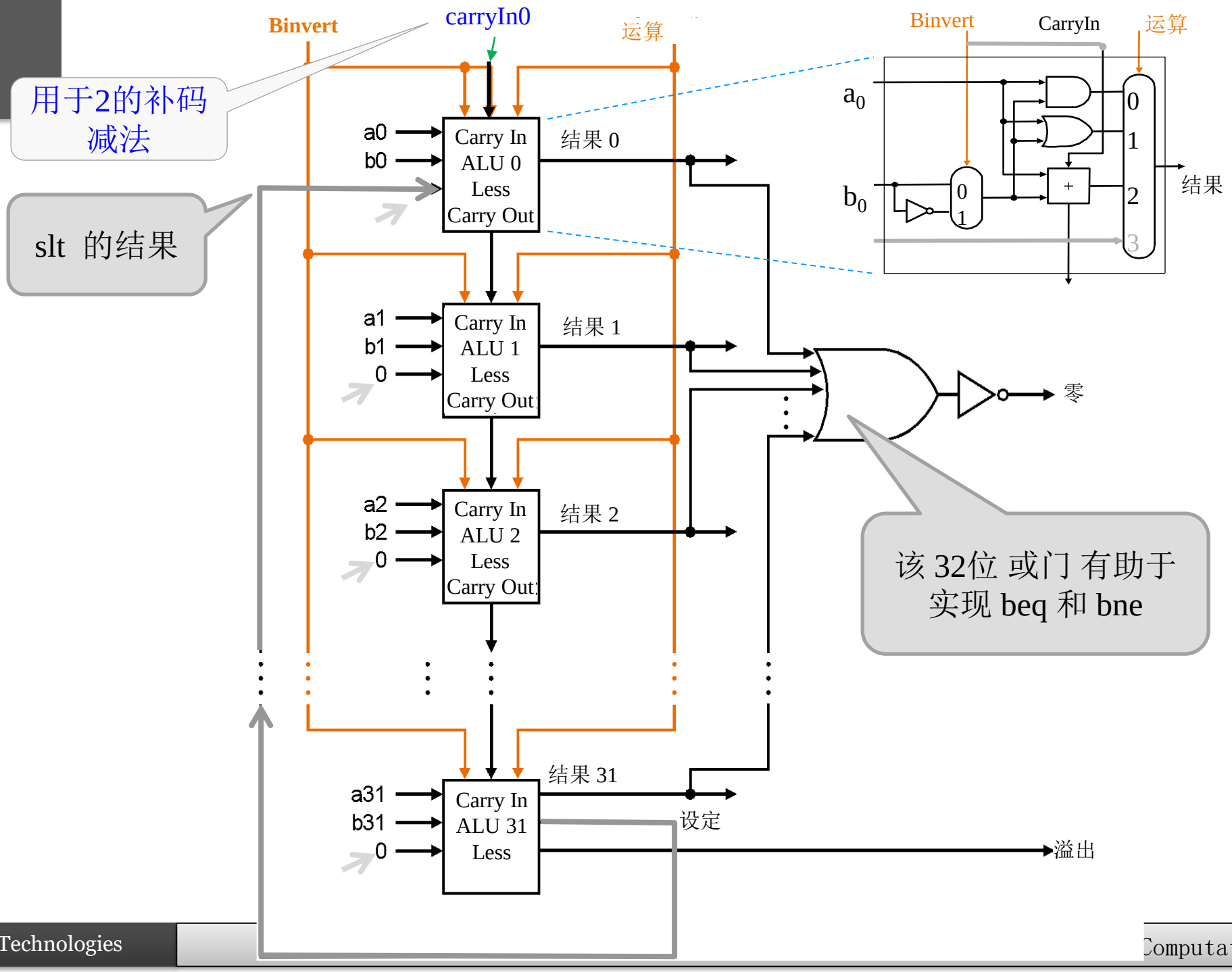
最低 1位 ALU (bit 0)



最高 1位 ALU (bit 31)

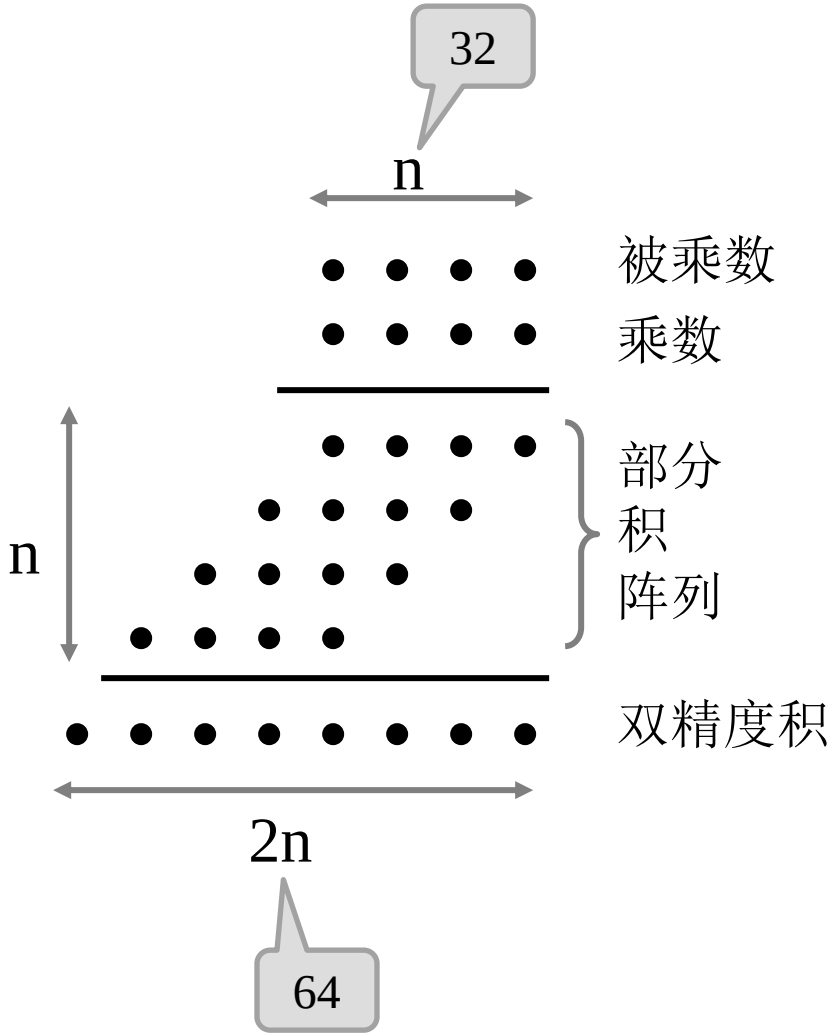


包含 slt、beq、bne 的 32-Bit ALU



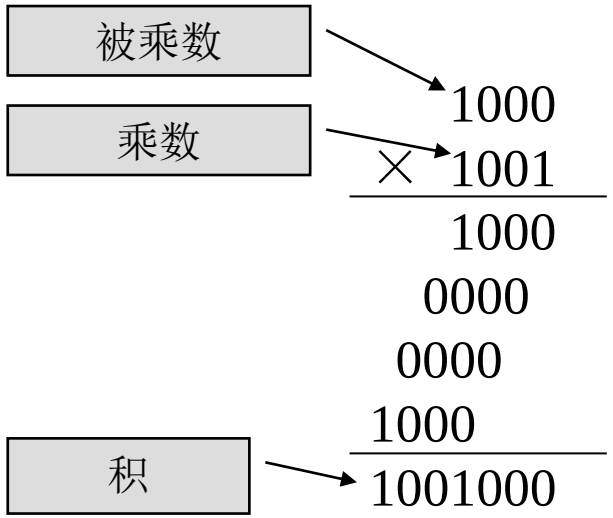
乘法

- 二进制乘法只是一*堆*右移和加法

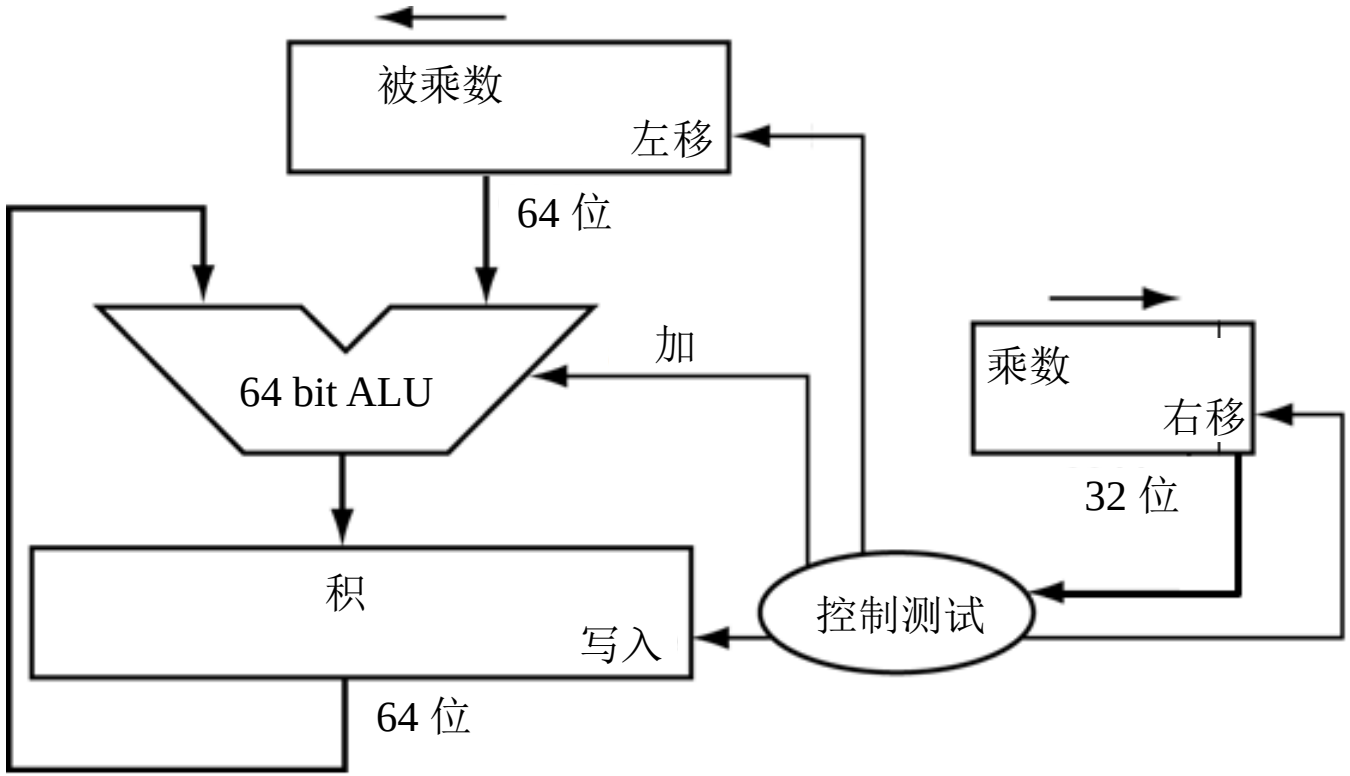


乘法器硬件（基本设计）

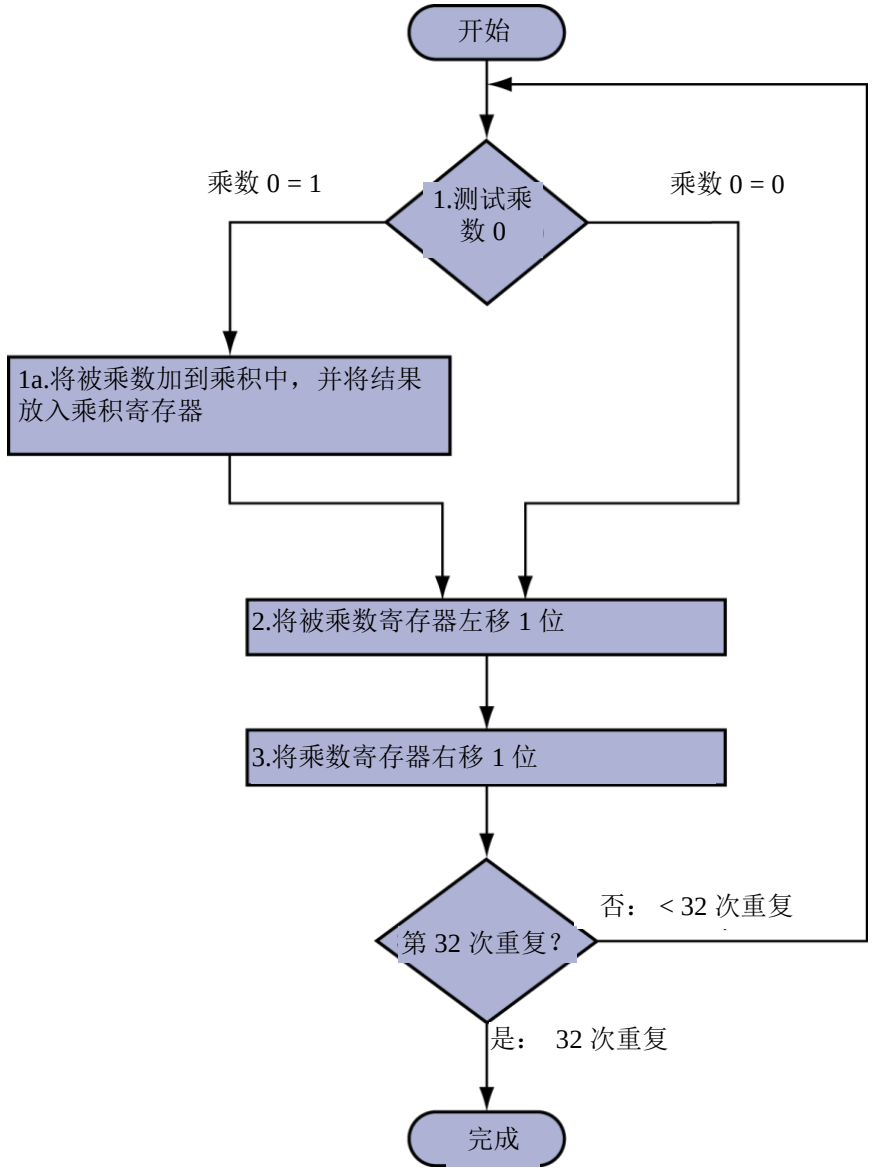
- 从长乘法开始

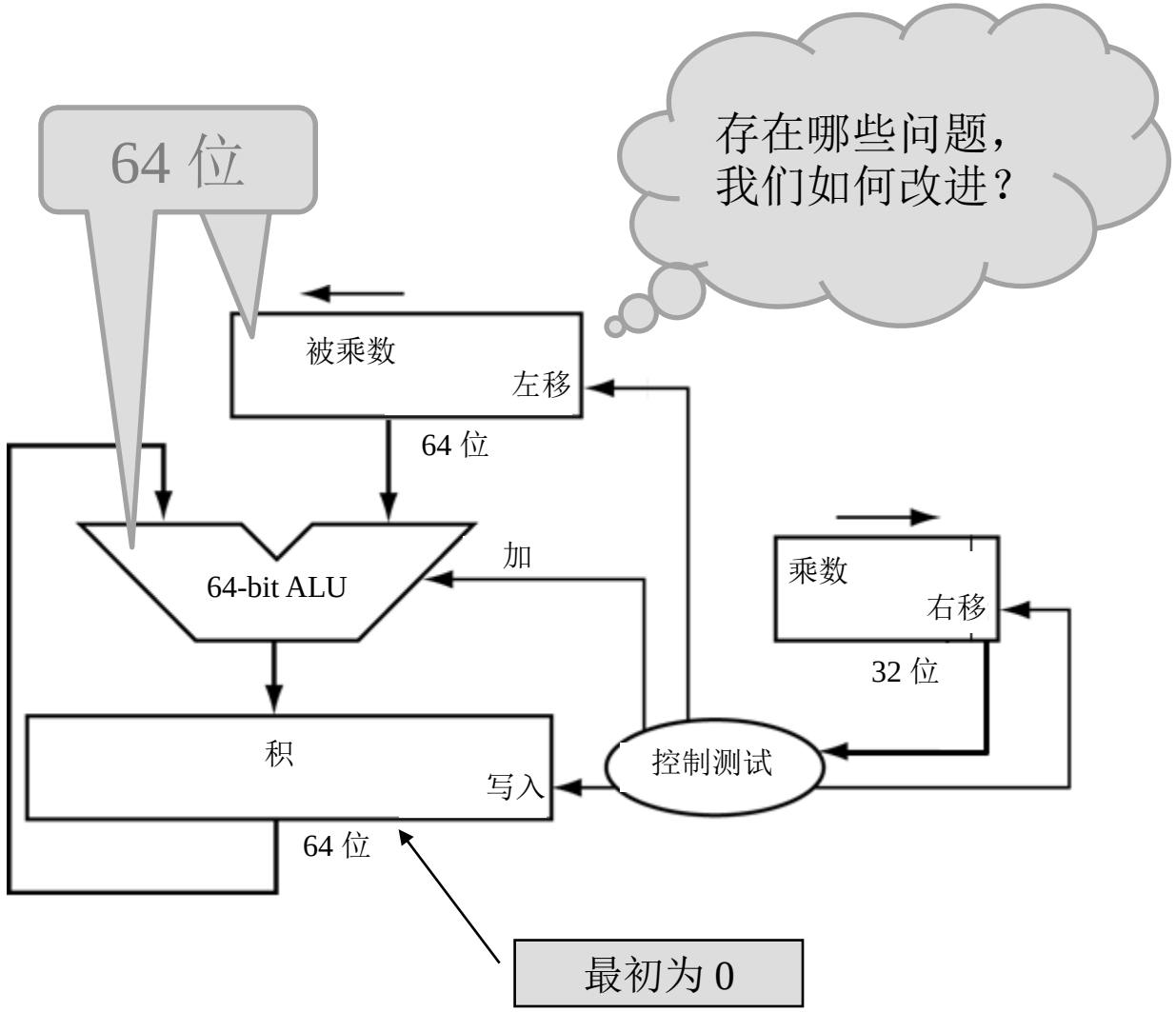


乘积长度是运算数长之和



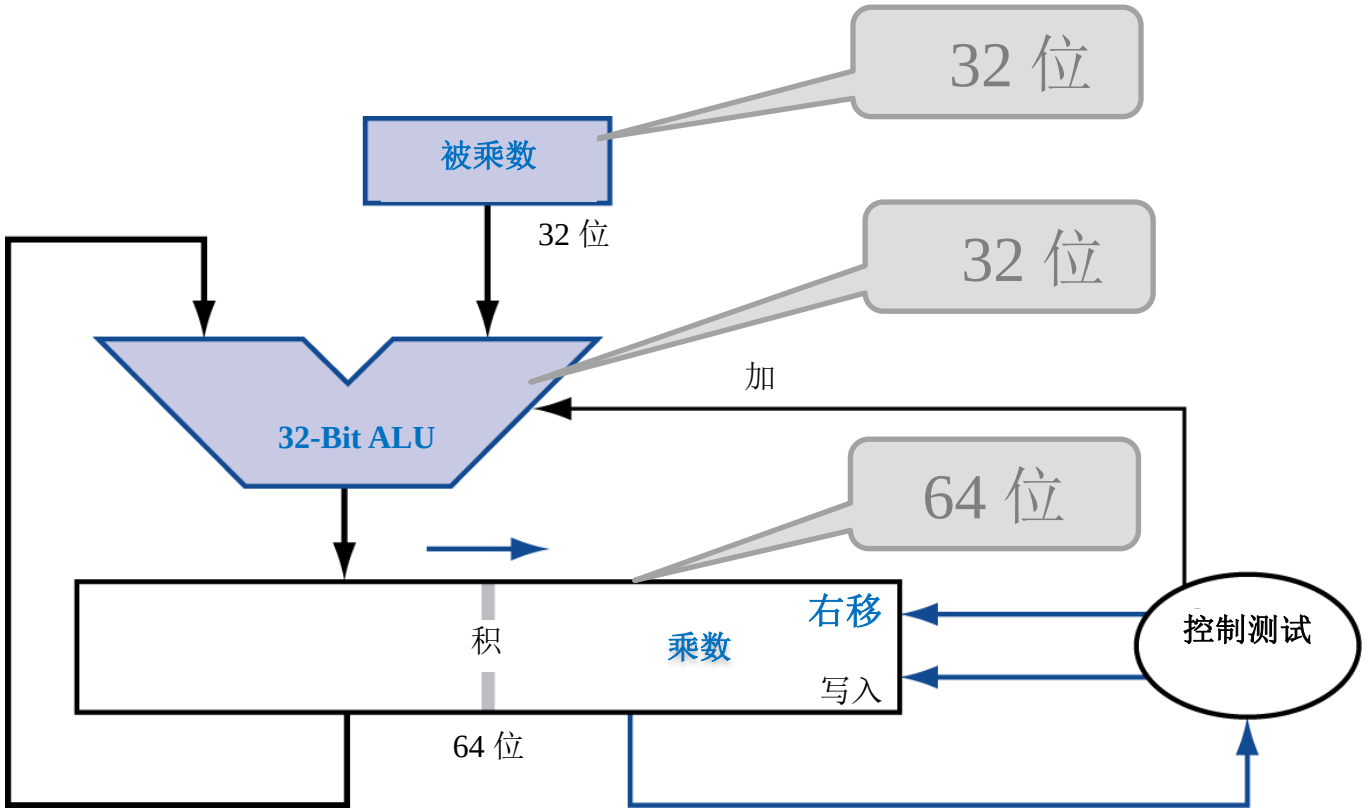
乘法器硬件


开始
1.测试乘数 0
乘数 0 = 1
乘数 0 = 0
1a.将被乘数加到乘积中，并将结果放入乘积寄存器
2.将被乘数寄存器左移 1 位
3.将乘数寄存器右移 1 位
第 32 次重复？
否： < 32 次重复
是： 32 次重复
完成

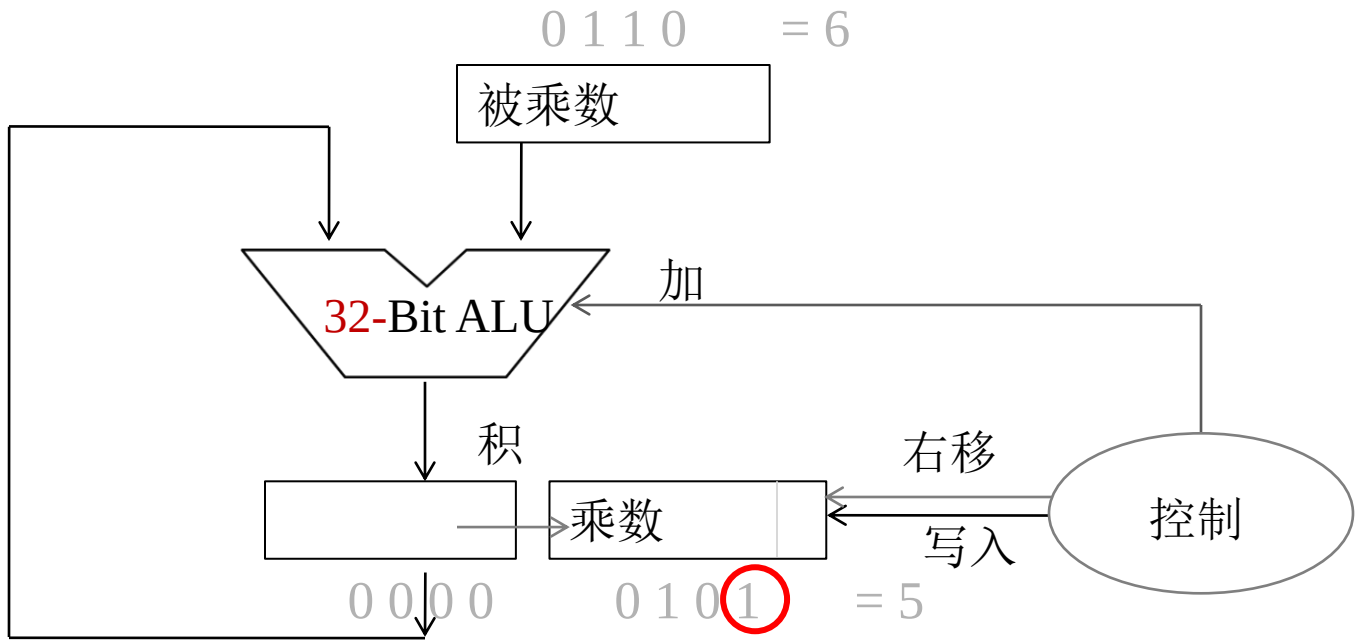

64 位
存在哪些问题，
我们如何改进？
被乘数
左移
64 位
64-bit ALU
加
乘数
右移
32 位
积
写入
控制测试
64 位
最初为 0

优化的乘法器

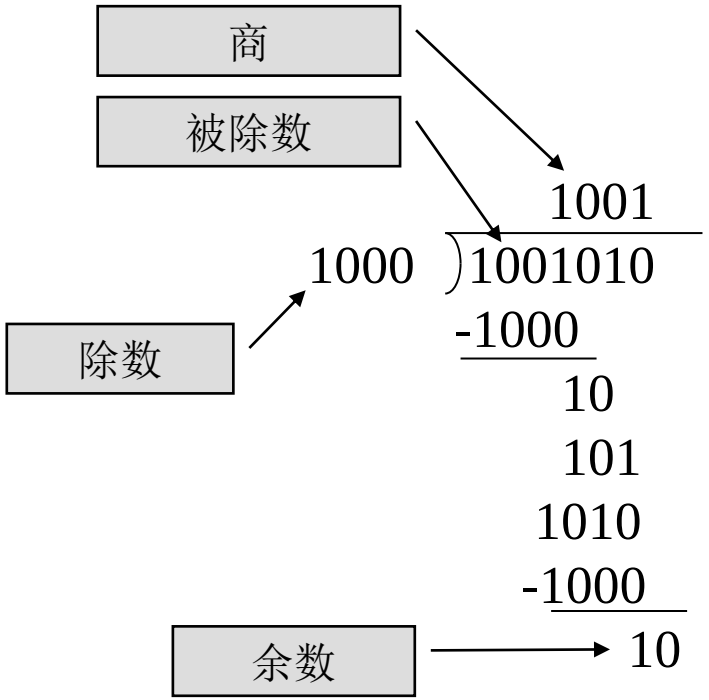
- 并行执行步骤：加法/移动



带加法和右移的乘数器硬件



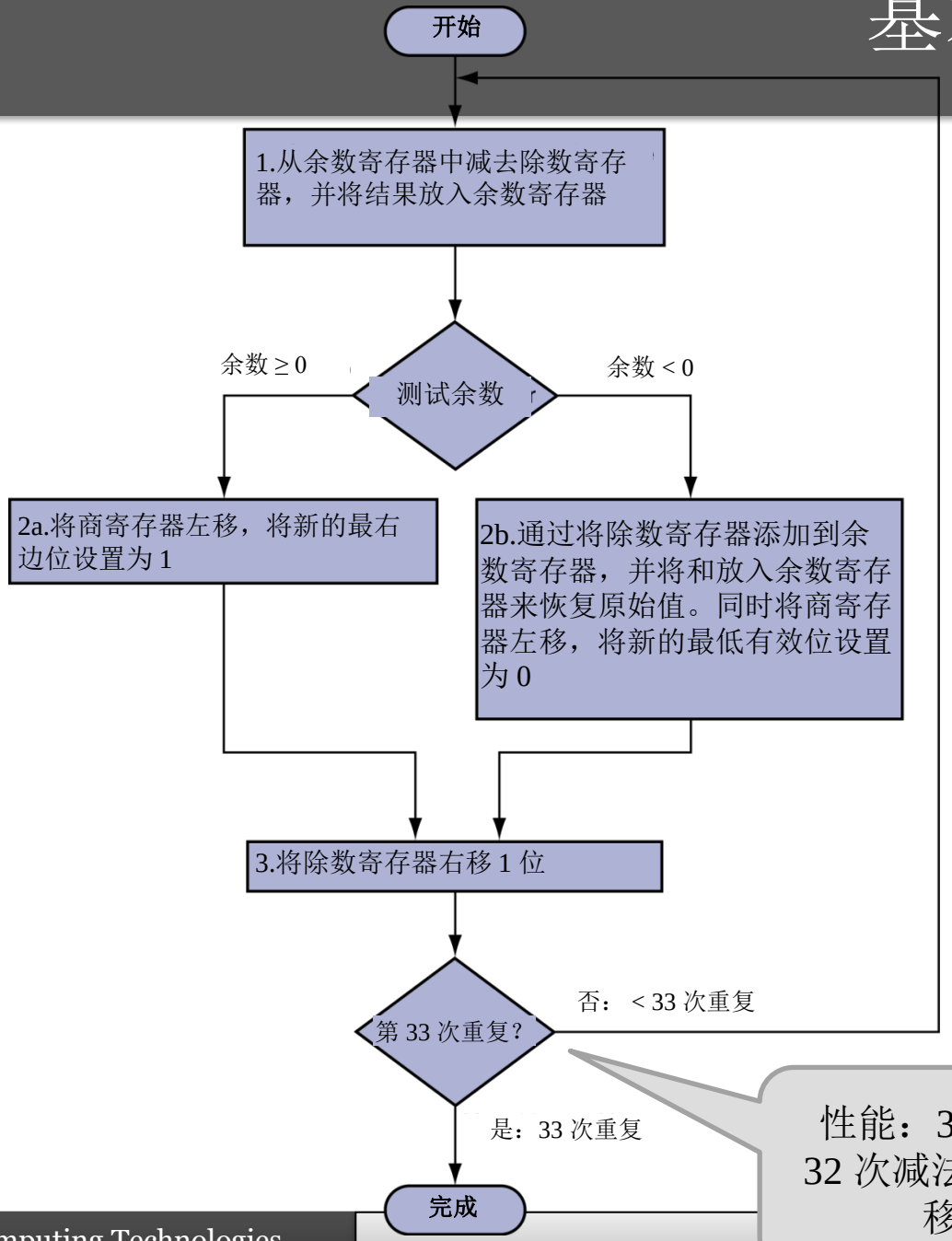
除法

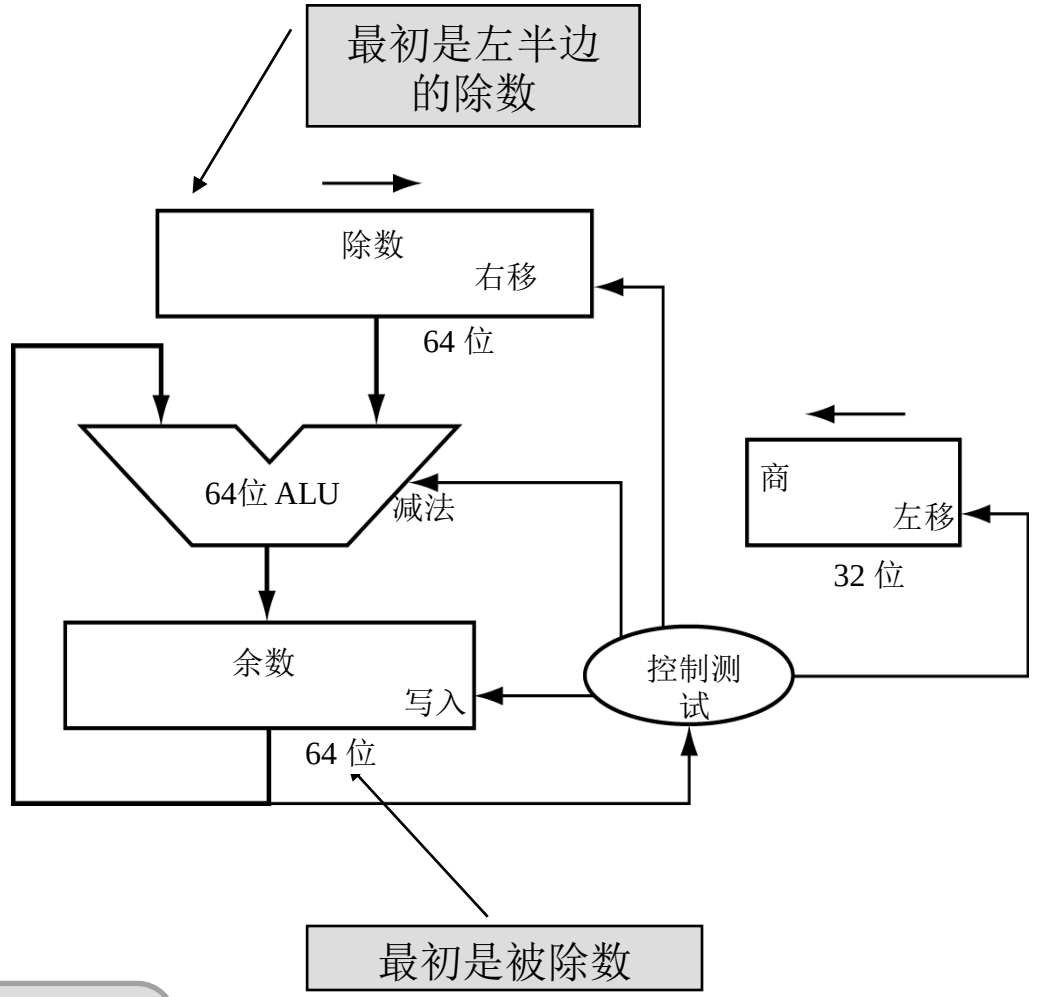


- 检查 0 除数
- 长除法
  - 如果除数 ≤ 被除数位
    - 商数位1，减
  - 否则
    - 商数位0，把下一个被除数位挪下来
- 恢复除法
  - 做减法，如果余数< 0，则将除数加回
- 有符号除法
  - 使用绝对值除
  - 根据需要调整商和余数的符号

n 位运算数产生 *n* 位商和 *n* 位余数

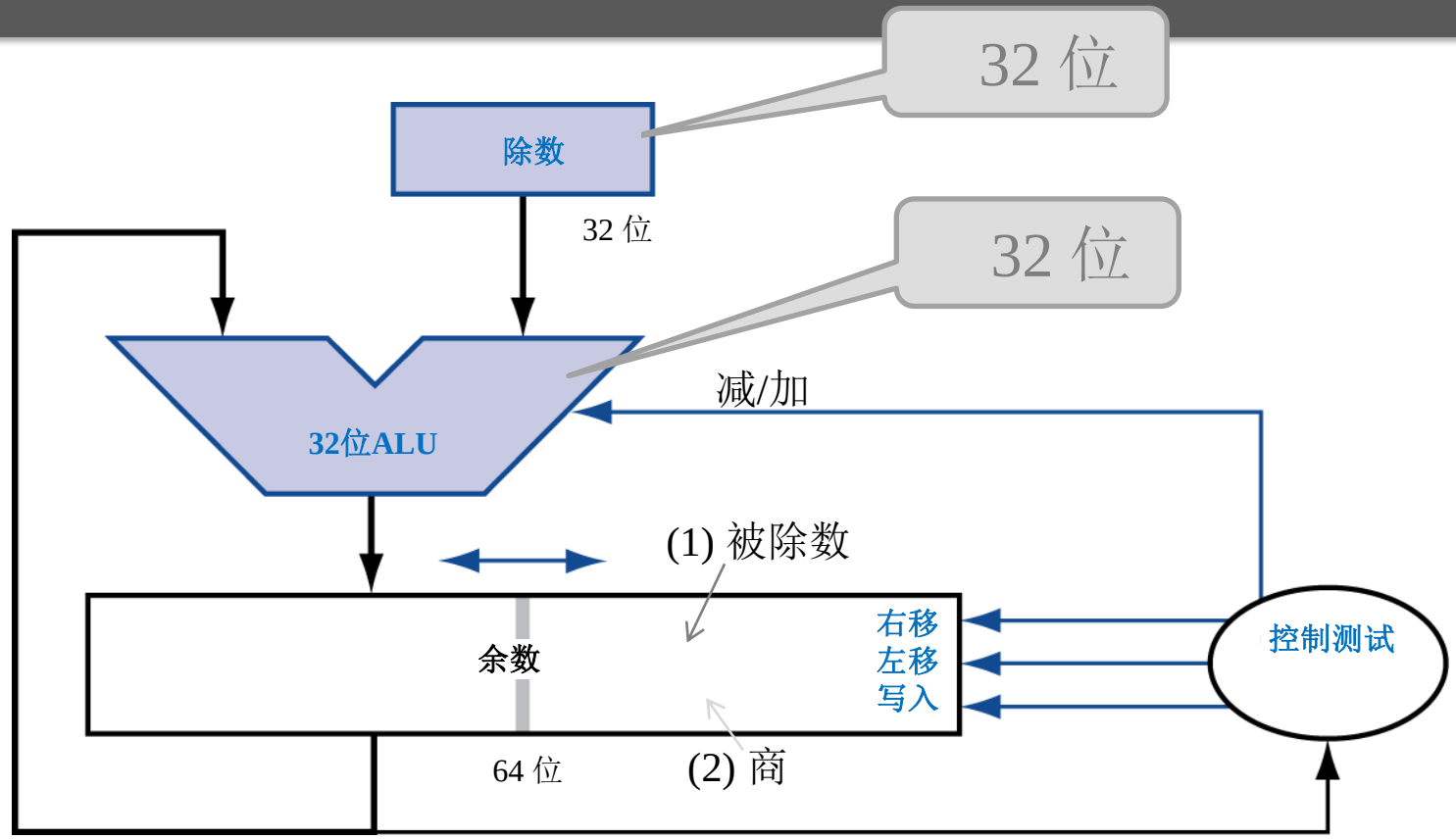
基本除法器硬件





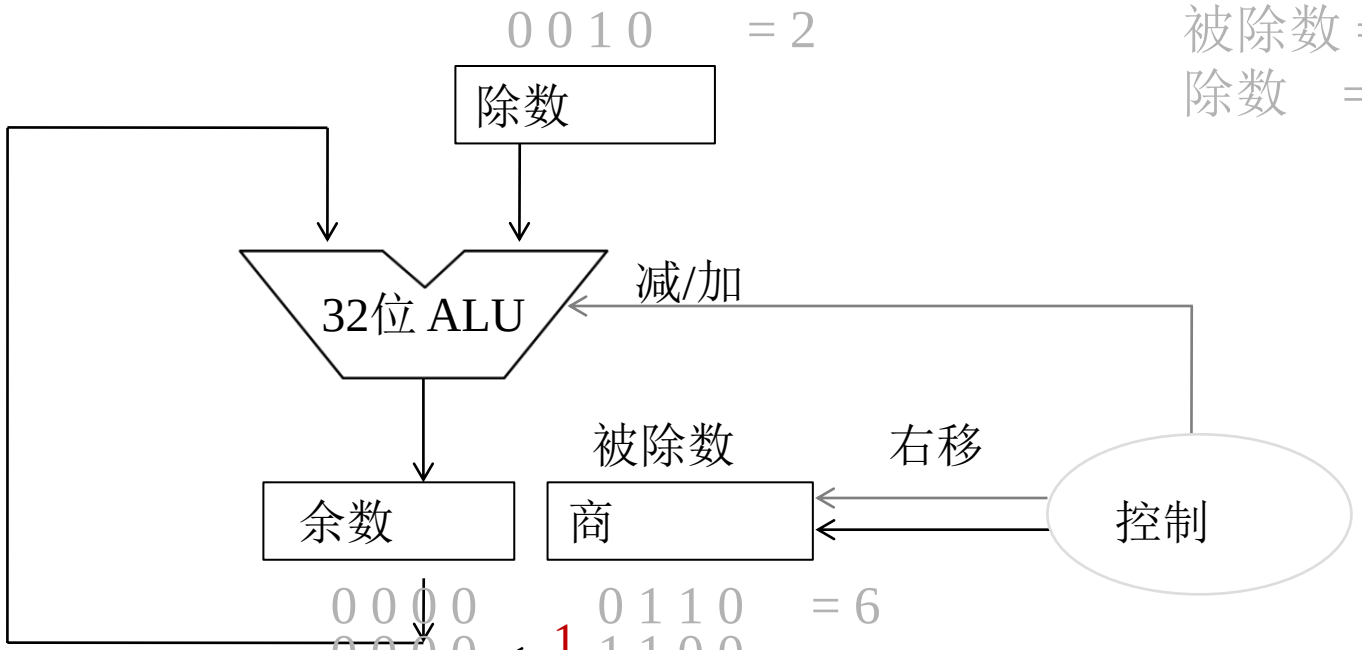
性能：32 次迭代：
32 次减法 32 次加法/移位！

优化的除法器



- 每个部分余数减法形成一个周期
- 看起来很像乘法器！
  - 相同的硬件可用于乘法和除法

左移和减除硬件



被除数 = 6
除数 = 2

0 0 0 0 0 1 1 0 = 6

0 0 0 0	1 1 0 0		
sub 1 1 1 0	1 1 0 0	rem neg, so quotient bit = 0	余数为负数，因此商位= 0 → 恢复余数
add 0 0 0 0	1 1 0 0	restore remainder	
0 0 0 1	1 0 0 0		
sub 1 1 1 1	1 1 0 0	rem neg, so quotient bit = 0	余数为负数，因此商位= 0 → 恢复余数
add 0 0 0 1	1 0 0 0	restore remainder	
0 0 1 1	0 0 0 0		
sub 0 0 0 1	0 0 0 1	rem pos, so quotient bit = 1	余数为正数，因此商位= 1 → 无需恢复
0 0 1 0	0 0 1 0		
sub 0 0 0 0	0 0 1 1	rem pos, so quotient bit = 1	余数为正数，因此商位= 1 → 无需恢复

结果 = 3，余数 0