

计系3 期末速通教程

3. 运算

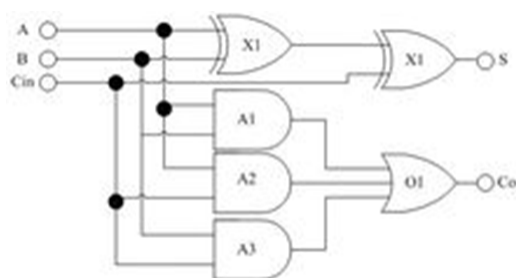
3.1 整数加减法

3.1.1 整数加法

[溢出] 若结果超出整数的表达范围, 则发生溢出.

- (1) 正负操作数相加: 不溢出.
- (2) 两正操作数相加: 符号位为 1 时溢出.
- (3) 两负操作数相加: 符号位为 0 时溢出.

[硬件实现] 一位全加器的逻辑表达式:
$$\begin{cases} S = A \oplus B \oplus Cin \\ Co = A \cdot Cin + B \cdot Cin + A \cdot B \end{cases}$$
 其中 A, B 为加数, Cin 为进位输入, S 为和, Co 为进位输出.



3.1.2 整数减法

[溢出] 若结果超出整数的表达范围, 则发生溢出.

- (1) 两同号操作数相减: 不溢出.
- (2) 负操作数减正操作数: 符号位为 0 时溢出.
- (3) 正操作数减负操作数: 符号位为 1 时溢出.
- (4) MIPS中, 指令 `add` 检测溢出, `addu` 忽略溢出.

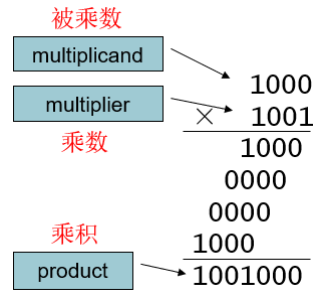
[硬件实现] 用取反 +1 (在最低位的进位上提供)求补码后用加法器即可.

3.2 整数乘法

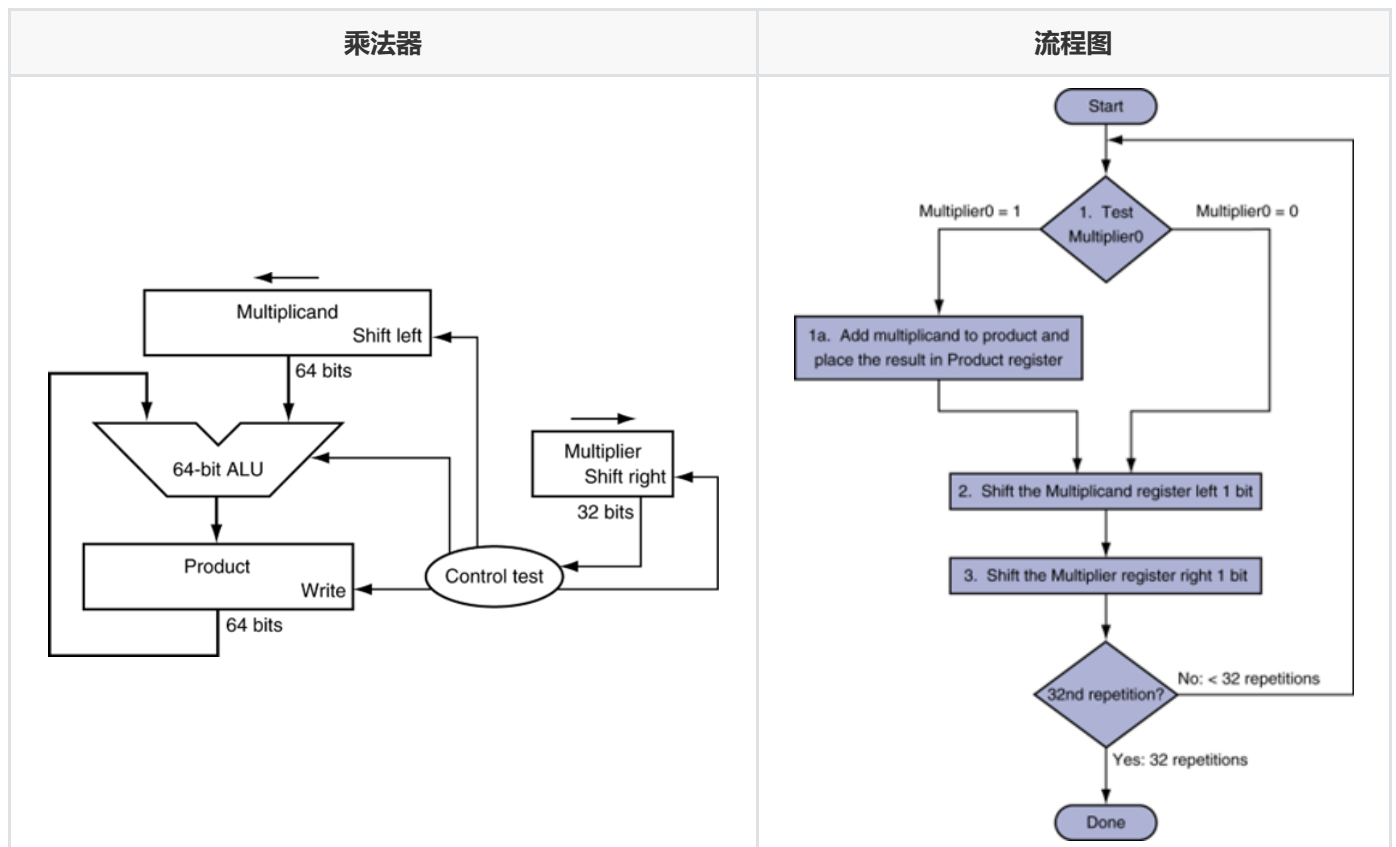
3.2.1 整数乘法

[乘法器]

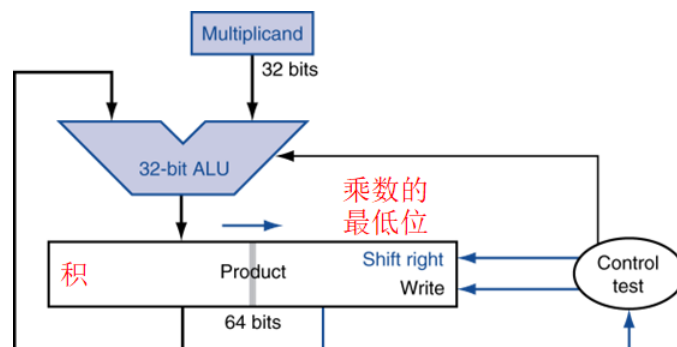
(1) 二进制数乘法过程:



(2) 无符号数乘法器与流程图:



(3) 优化后的无符号数乘法器: 将乘数和积放在一个 64 位寄存器中.



(4) 有符号数乘法器: 先做无符号数乘法, 再单独计算符号位.

3.2.2 整数除法

[除法器]

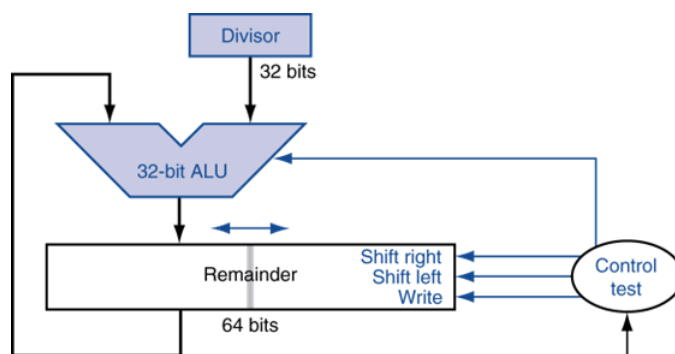
(1) 二进制数除法过程:

二进制数除法过程	过程
<div><div>商</div><div>quotient</div><div>dividend</div><div>被除数</div><div>1000 1001010</div><div>divisor</div><div>除数</div><div>1000</div><div>-1000</div><div>10</div><div>101</div><div>1010</div><div>-1000</div><div>10</div><div>remainder</div><div>余数</div></div>	<div>(1) 除数判 0 .</div> <div>(2) 除法步骤:</div> <div>① 若除数 \leq 被除数, 则商 +1, 做减法.</div> <div>② 否则, 商 +0, 从被除数中提取下一 bit .</div> <div>(3) 恢复余数: 做减法后, 若余数 < 0 , 则加回除数.</div> <div>(4) 有符号数除法:</div> <div>① 先对无符号数做除法.</div> <div>② 根据符号位调整商和余数的符号.</div>

(2) 无符号数除法器与流程图:

除法器	流程图
<div><div>Divisor</div><div>Shift right</div><div>64 bits</div><div>64-bit ALU</div><div>Remainder</div><div>Write</div><div>64 bits</div><div>Control test</div><div>Quotient</div><div>Shift left</div><div>32 bits</div></div>	<div><div>Start</div><div>1. Subtract the Divisor register from the Remainder register and place the result in the Remainder register</div><div>Test Remainder</div><div>Remainder ≥ 0</div><div>2a. Shift the Quotient register to the left, setting the new rightmost bit to 1</div><div>Remainder < 0</div><div>2b. Restore the original value by adding the Divisor register to the Remainder register and placing the sum in the Remainder register. Also shift the Quotient register to the left, setting the new least significant bit to 0</div><div>3. Shift the Divisor register right 1 bit</div><div>33rd repetition?</div><div>No: < 33 repetitions</div><div>Yes: 33 repetitions</div><div>Done</div></div>

(3) 优化后的无符号数除法器: 将被除数和余数放在一个 64 位寄存器中.



除法器可与乘法器使用同一硬件.

(4) 有符号数除法器: 先做无符号数乘法, 再考虑符号位.

① 符号相同时商为正数, 相异时商为负数.

② 余数与被除数同号.

3.3 浮点数加减法

[例3.3.1] 计算 $9.999 \times 10^1 + 1.610 \times 10^{-1}$.

[解]

(1) 对阶: 将小阶的值调整到与大阶一致.

$$\text{原式} = 9.999 \times 10^1 + 0.016 \times 10^1.$$

(2) 尾数相加.

$$9.999 \times 10^1 + 0.016 \times 10^1 = 10.015 \times 10^1.$$

(3) 结果规格化, 检查是否溢出.

$$10.015 \times 10^1 = 1.0015 \times 10^2.$$

(4) 舍入.

$$1.0015 \times 10^2 \text{ 向偶数舍入为 } 1.002 \times 10^2.$$

(5) 检查是否需重新规格化.

[例3.3.2] 计算 $1.000_2 \times 2^{-1} + (-1.110_2 \times 2^{-2})$.

[解]

(1) 对阶: 原式 $= 1.000_2 \times 2^{-1} + (-0.111_2 \times 2^{-1})$.

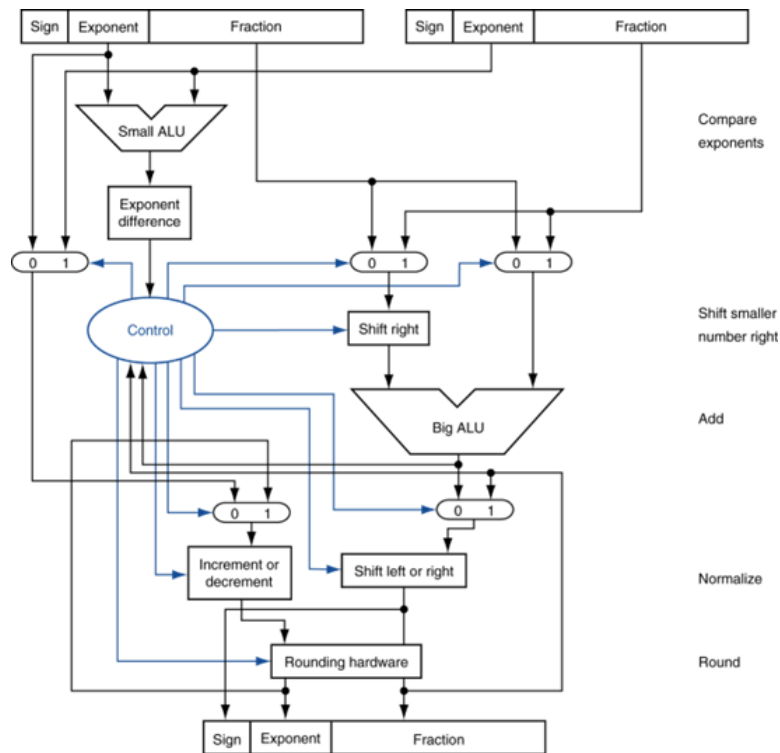
(2) 尾数相加: $1.000_2 \times 2^{-1} + (-0.111_2 \times 2^{-1}) = 0.001_2 \times 2^{-1}$.

(3) 结果规格化, 检查是否溢出: $0.001_2 \times 2^{-1} = 1.000_2 \times 2^{-4}$.

(4) 舍入. $1.000_2 \times 2^{-4}$ 无需舍入.

(5) 检查是否需重新规格化.

[浮点加法器]



3.4 浮点数乘法

[例3.4.1] 计算 $(1.110 \times 10^{10}) \times (9.200 \times 10^{-5})$.

[解]

- (1) 阶码相加: $10 + (-5) = 5$.
- (2) 尾数相乘: $1.110 \times 9.200 = 10.212$.
- (3) 结果规格化, 检查是否溢出: $10.212 \times 10^5 = 1.0212 \times 10^6$.
- (4) 舍入: 1.0212×10^6 舍入为 1.021×10^6 .
- (5) 检查是否需重新规格化.
- (6) 确定积的符号: 为 + .

[例3.4.2] 计算 $(1.000_2 \times 2^{-1}) \times (-1.110_2 \times 2^{-2})$.

[解]

- (1) 阶码相加: $(-1) + (-2) = -3$.
- (2) 尾数相乘: $1.000_2 \times 1.110_2 = 1.110_2$.
- (3) 结果规格化, 检查是否溢出: $1.110_2 \times 2^{-3}$ 无需规格化.
- (4) 舍入: $1.110_2 \times 2^{-3}$ 无需舍入.
- (5) 检查是否需重新规格化.
- (6) 确定积的符号: 为 - .

