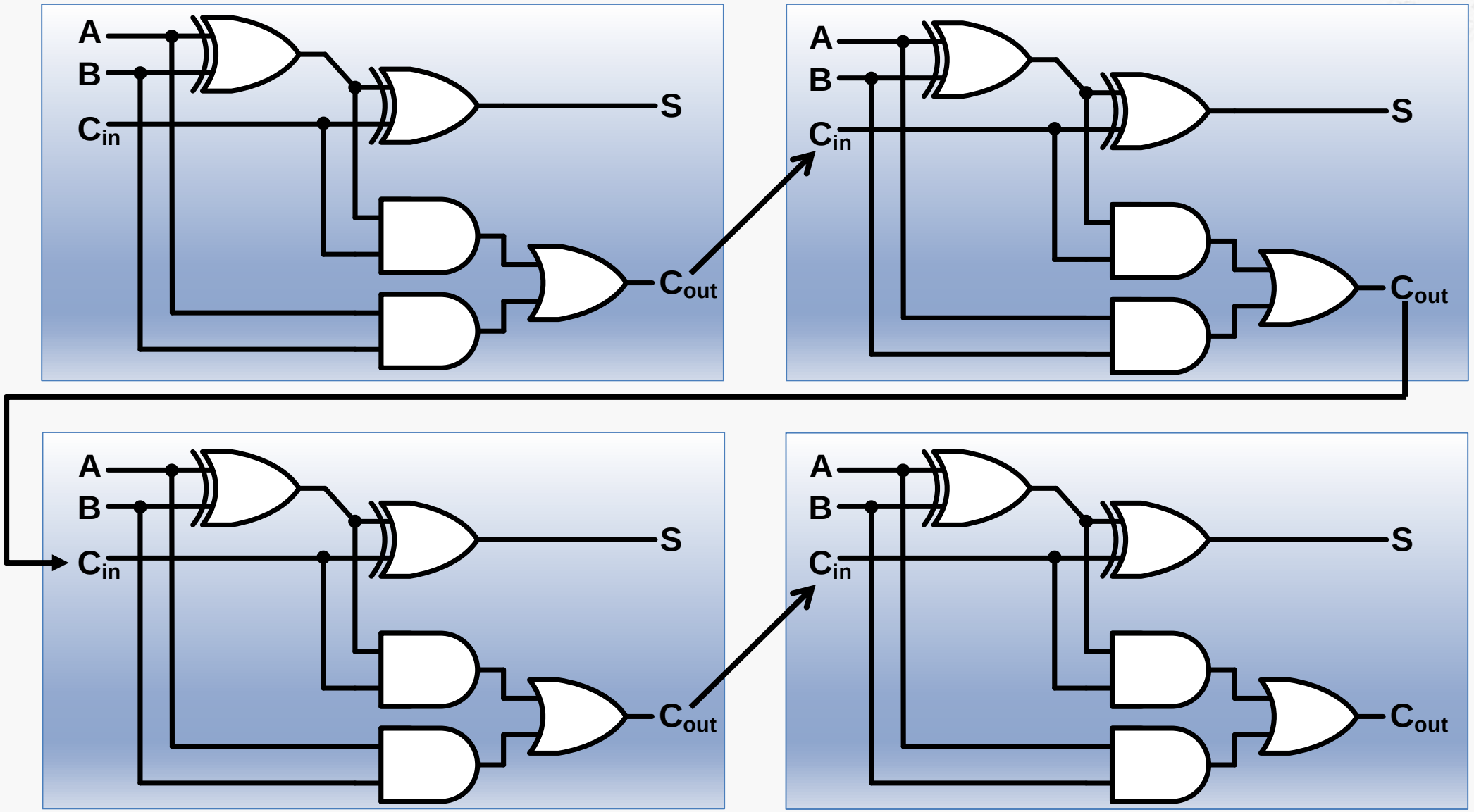
慕课回顾和扩展

运算器和控制器

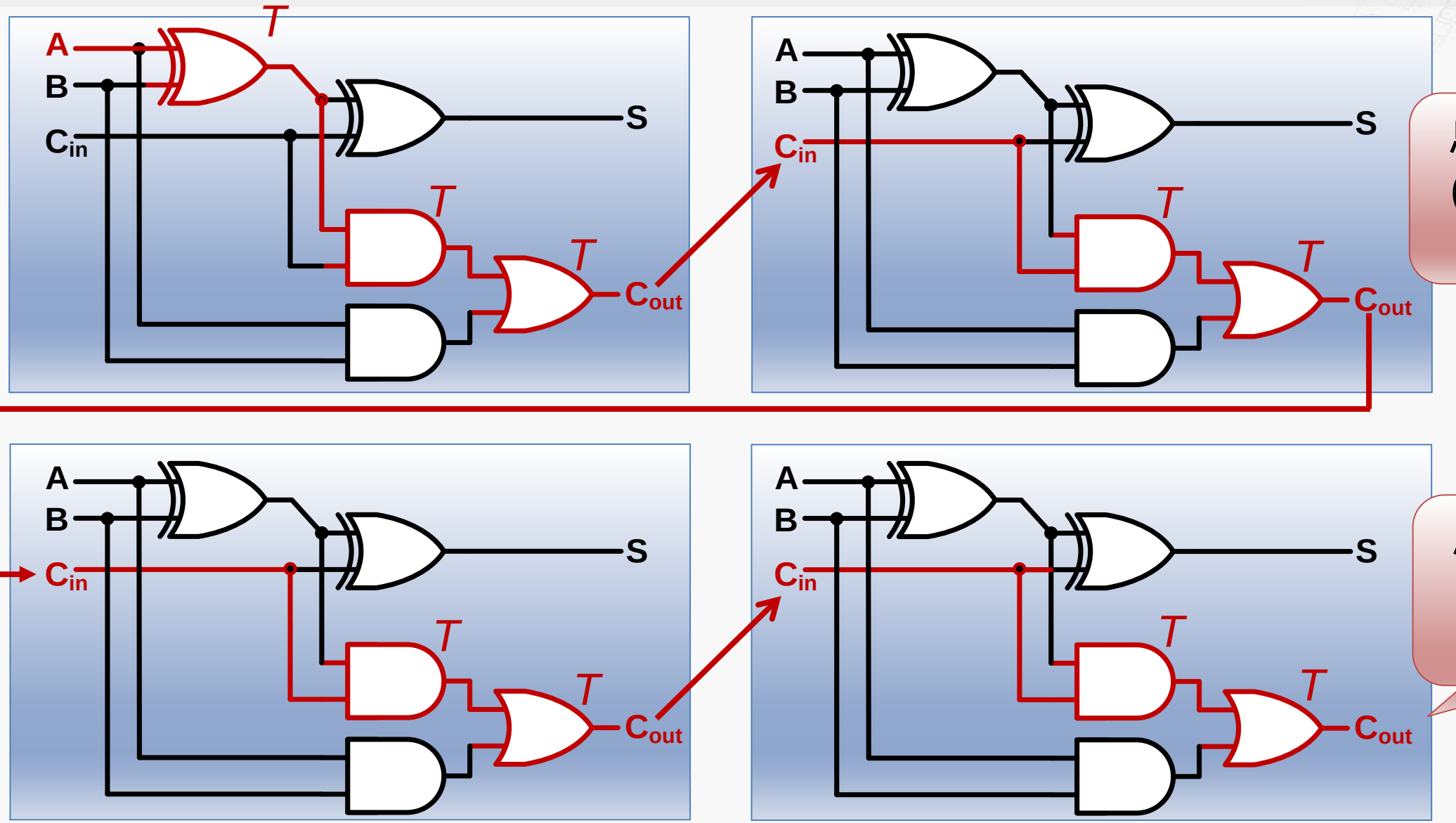
北京大学·慕课
计算机组成
制作人：陆俊林

（1）加法器

4-bit RCA的门电路实现



4-bit RCA的关键路径（延迟最长的路径）



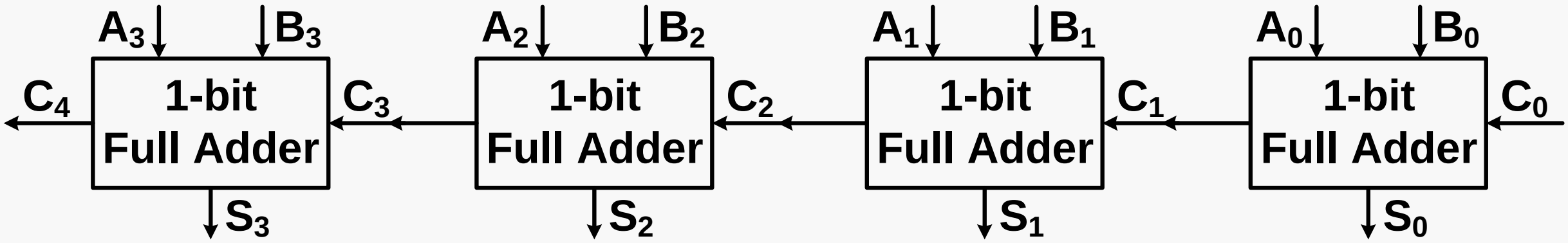
进位输出信号的分析

$$C_{i+1} = (A_i \cdot B_i) + (A_i \cdot C_i) + (B_i \cdot C_i)$$
$$= (A_i \cdot B_i) + (A_i + B_i) \cdot C_i$$

设：

- 生成（Generate）信号：$G_i = A_i \cdot B_i$
- 传播（Propagate）信号：$P_i = A_i + B_i$

则：$\mathbf{C_{i+1} = G_i + P_i \cdot C_i}$



如何提前计算“进位输出信号”

- $C_1 = G_0 + P_0 \cdot C_0$
- $\mathbf{C_2} = G_1 + P_1 \cdot C_1$

  $= G_1 + P_1 \cdot (G_0 + P_0 \cdot C_0)$

  $\mathbf{= G_1 + P_1 \cdot G_0 + P_1 \cdot P_0 \cdot C_0}$
- $\mathbf{C_3} = G_2 + P_2 \cdot C_2$

  $= G_2 + P_2 \cdot (G_1 + P_1 \cdot G_0 + P_1 \cdot P_0 \cdot C_0)$

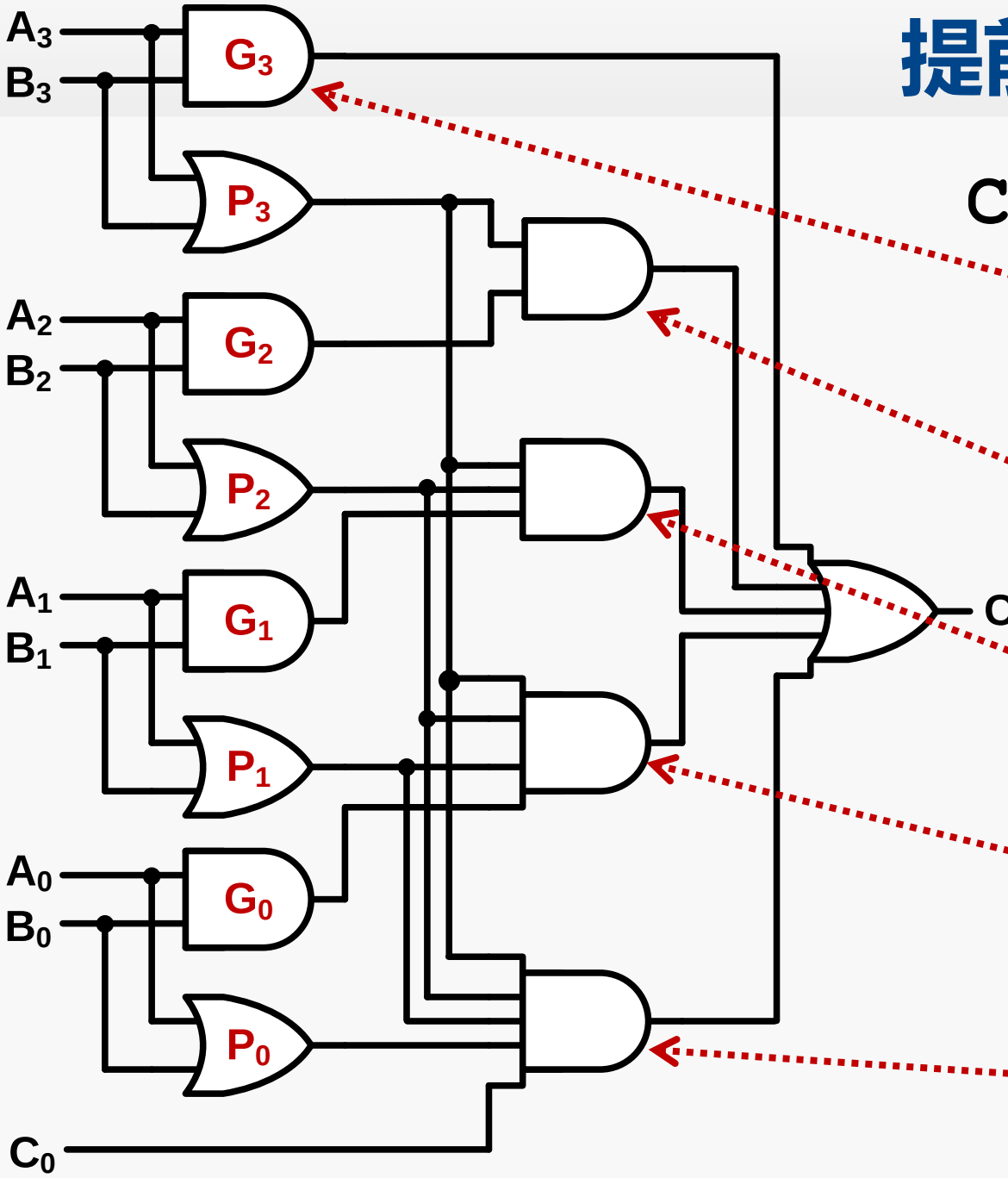
  $\mathbf{= G_2 + P_2 \cdot G_1 + P_2 \cdot P_1 \cdot G_0 + P_2 \cdot P_1 \cdot P_0 \cdot C_0}$
- $\mathbf{C_4} = G_3 + P_3 \cdot C_3$

  $= G_3 + P_3 \cdot (G_2 + P_2 \cdot G_1 + P_2 \cdot P_1 \cdot G_0 + P_2 \cdot P_1 \cdot P_0 \cdot C_0)$

  $\mathbf{= G_3 + P_3 \cdot G_2 + P_3 \cdot P_2 \cdot G_1 + P_3 \cdot P_2 \cdot P_1 \cdot G_0 + P_3 \cdot P_2 \cdot P_1 \cdot P_0 \cdot C_0}$

$$\mathbf{C_{i+1} = G_i + P_i \cdot C_i}$$

提前计算C_4的电路实现

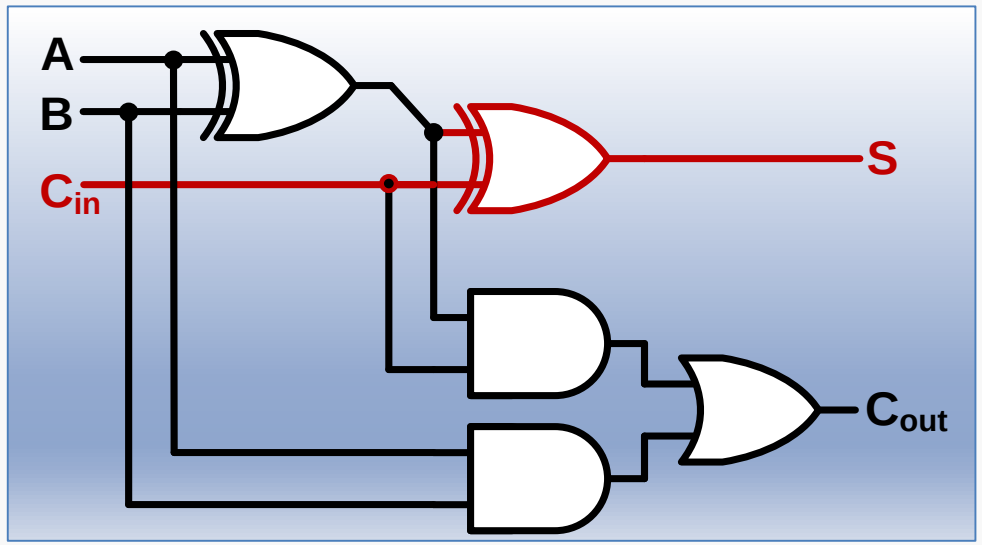


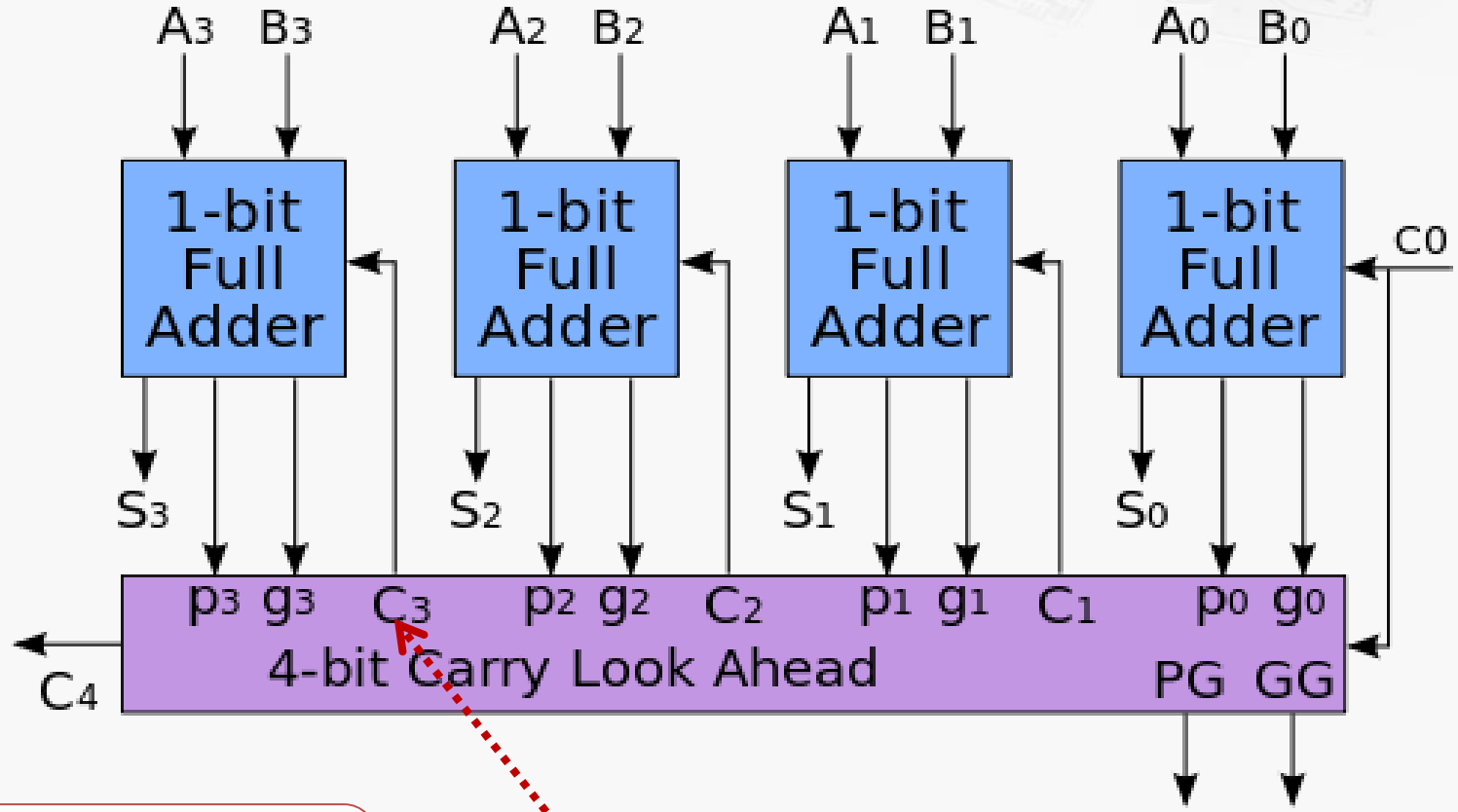
$$C_4 = G_3 + P_3 \cdot G_2 + P_3 \cdot P_2 \cdot G_1 + P_3 \cdot P_2 \cdot P_1 \cdot G_0 + P_3 \cdot P_2 \cdot P_1 \cdot P_0 \cdot C_0$$

优点：计算C_{i+1}的延迟时间固定为三级门延迟，与加法器的位数无关

缺点：如果进一步拓宽加法器的位数，则电路变得非常复杂

超前进位加法器（Carry-Lookahead Adder，CLA）





最后一级全加器还需要1级门延迟

总延迟时间为4级门延迟

计算C_3需要3级门延迟

参考值：4-bit行波进位加法器的总延迟时间为9级门延迟

32-bit加法器的实现

- 如果采用行波进位
  - 总延迟时间为65级门延迟

	延迟时间	时钟频率
32-bit RCA	1.3ns	769MHz
单个CLA	0.08ns	/
4级CLA	0.26ns	3.84GHz

注：参照28nm制造工艺，门延迟设为0.02ns

- 如果采用完全的超前进位
  - 理想的总延迟时间为4级门延迟
  - 实际上电路过于复杂，难以实现

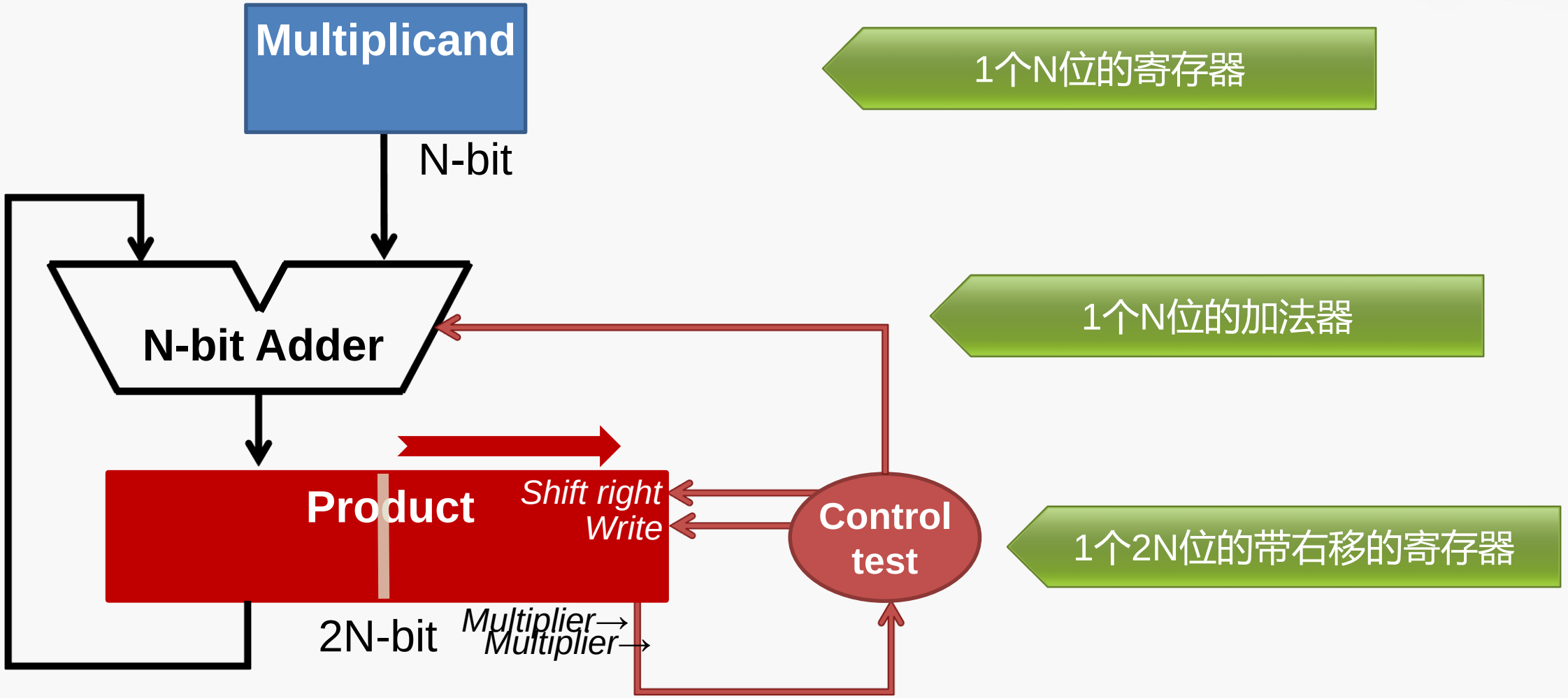
$$C_{31}=G_{30}+P_{30}\cdot G_{29}+P_{30}\cdot P_{29}\cdot G_{28}+\ldots$$
$$+P_{30}\cdot P_{29}\cdot P_{28}\cdot\ldots\cdot P_2\cdot P_1\cdot P_0\cdot C_0$$

需要32输入的与门和或门？！

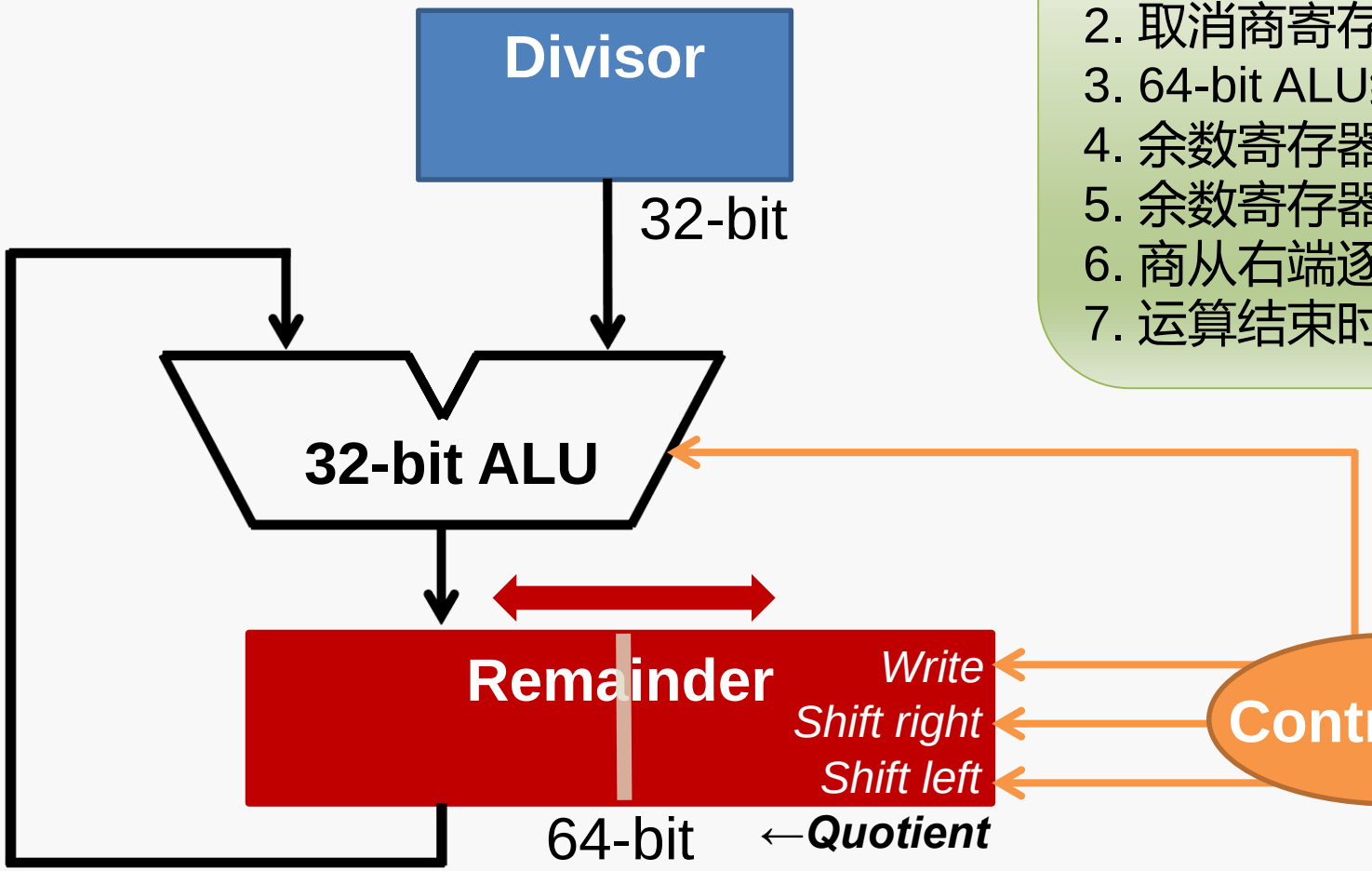
- 通常的实现方法
  - 采用多个小规模的超前进位加法器拼接而成
  - 例如，用4个8-bit的超前进位加法器连接成32-bit加法器

（2）乘法器和除法器

N位乘法器的实现结构


Multiplicand
N-bit
N-bit Adder
Product
Shift right
Write
2N-bit
Multiplier→
Multiplier→
Control test
1个N位的寄存器
1个N位的加法器
1个2N位的带右移的寄存器

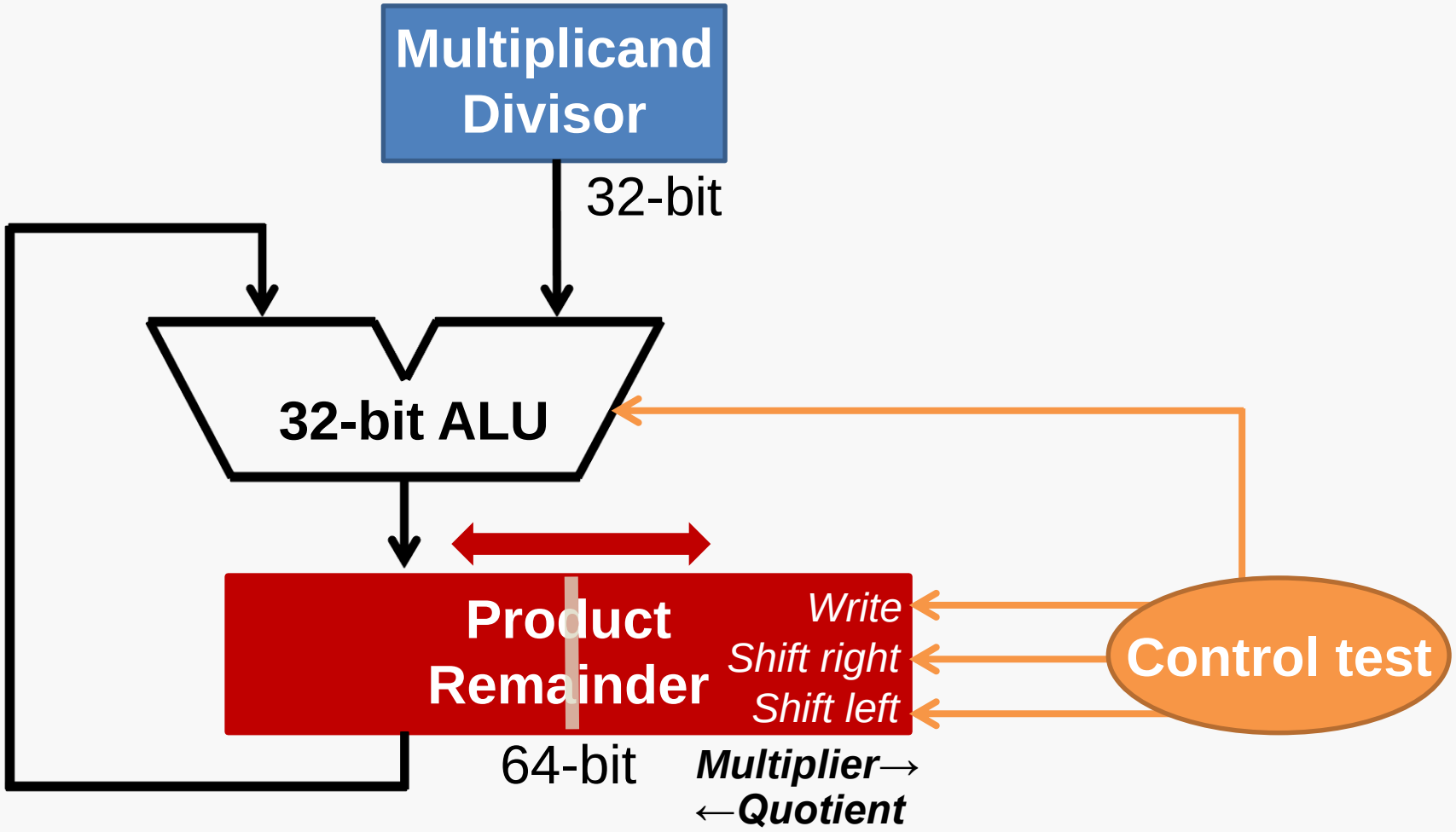
除法器的实现（第二版）



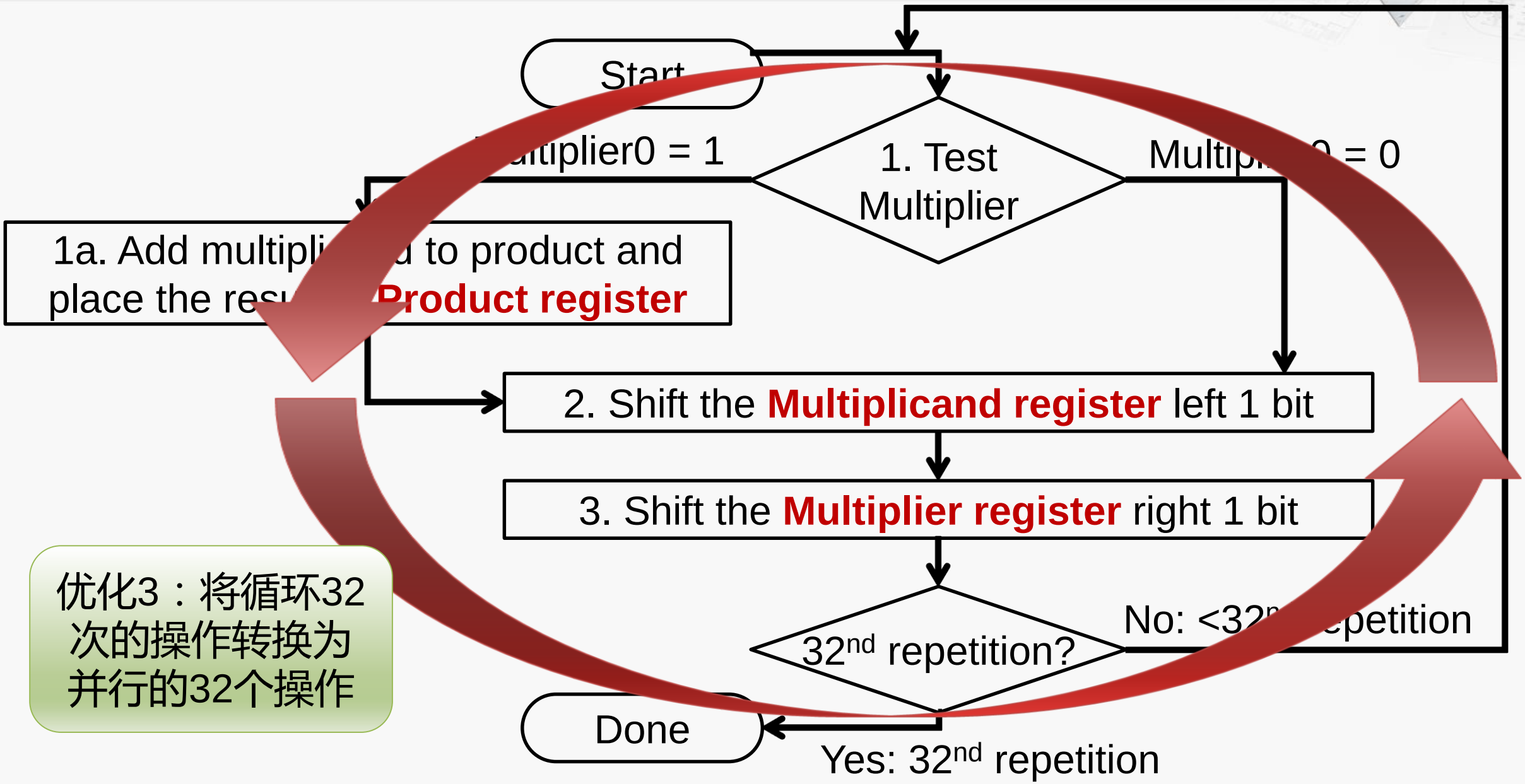
优化方案
1. 除数寄存器缩小为32-bit，无需支持移位
2. 取消商寄存器
3. 64-bit ALU缩小为32-bit
4. 余数寄存器只有高32-bit参与加减法运算
5. 余数寄存器需支持左移和右移
6. 商从右端逐位移入余数寄存器
7. 运算结束时，商占据余数寄存器的低32-bit

原先的结构：
一个64-bit的“余数”寄存器
一个64-bit的“除数”寄存器，带右移功能
一个32-bit的“商”寄存器，带左移功能
一个64-bit的ALU，支持加法和减法运算

乘法器和除法器的合并


Multiplicand
Divisor
32-bit
32-bit ALU
Product
Remainder
Write
Shift right
Shift left
Control test
64-bit
Multiplier→
←Quotient

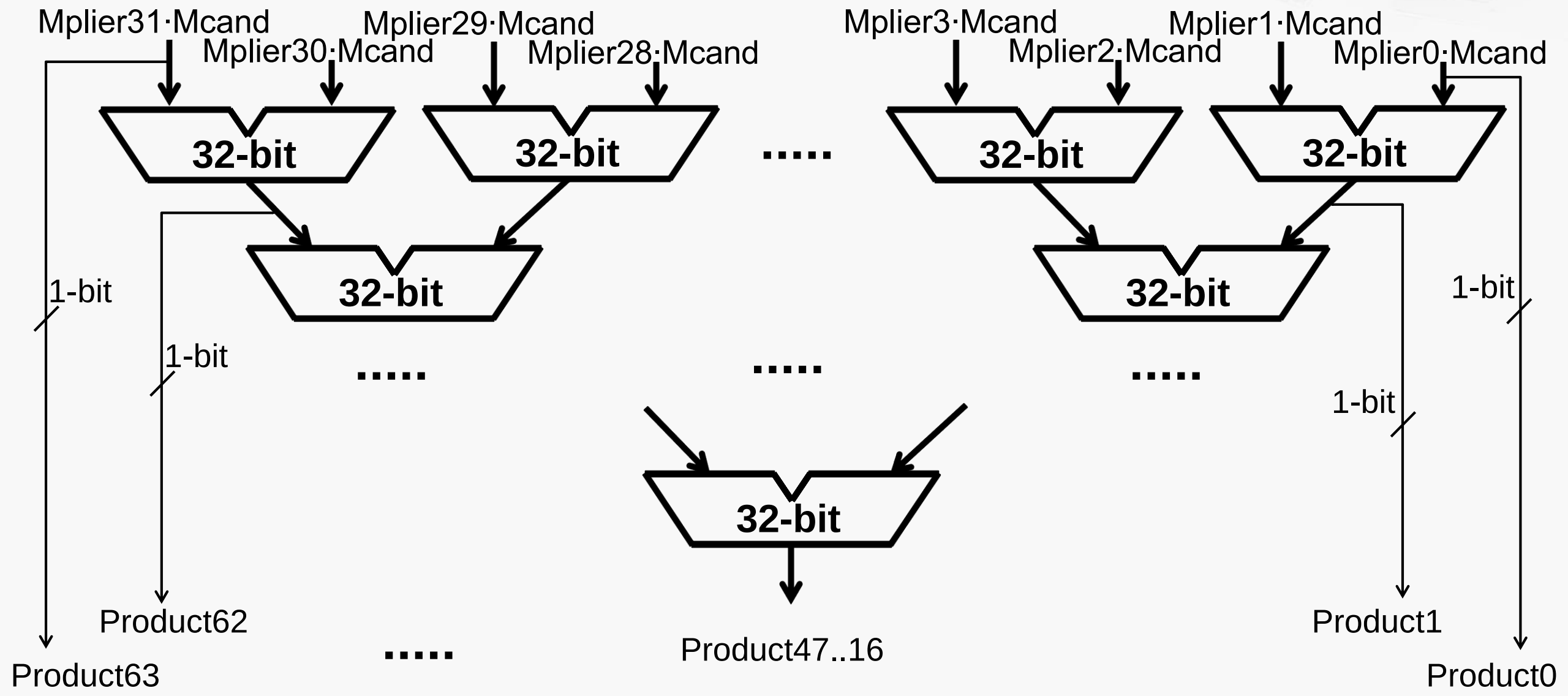
32-bit 乘法器的优化3：硬件的“循环展开”



优化3：将循环32次的操作转换为并行的32个操作

32-bit 乘法器的实现（第三版）

教材上的图有问题吗???



乘法的进一步优化

- 乘数：0 1 1 1 1 1 1 0
  - 会产生6个部分积

- 乘数：1 0 0 0 0 0 -1 0
  - 会产生2个部分积

（3） 乘法和除法指令

带符号数的乘法和除法

简单有效的方法

- 先将负数转换成正数进行运算
- 如果两个源操作数的符号不一致，则将运算结果取相反数

除法的余数

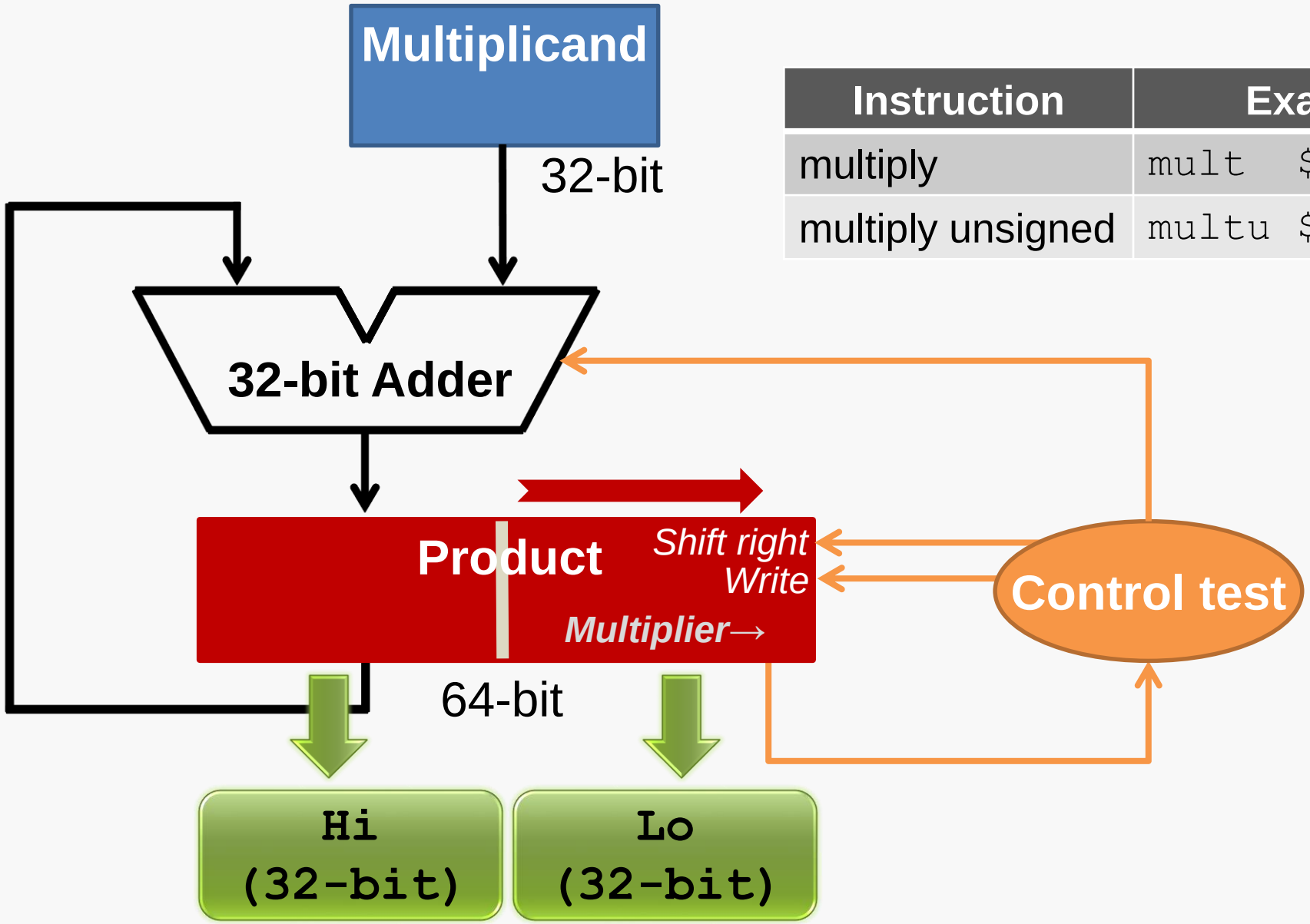
- 规则：余数的符号与被除数保持一致
- 示例：

  7÷2=3，余数为+1；　　7÷(-2)=-3，余数为+1

  -7÷2=-3，余数为-1；　　-7÷2=-4，余数为+1？

- 注意，应保持该等式成立：-(x÷y)=(-x)÷y

MIPS的乘法指令



Instruction	Example	Meaning
multiply	`mult $s2, $s3`	`Hi,Lo=$s2×$s3`
multiply unsigned	`multu $s2, $s3`	`Hi,Lo=$s2×$s3`

MIPS乘除法的专用寄存器：Hi，Lo

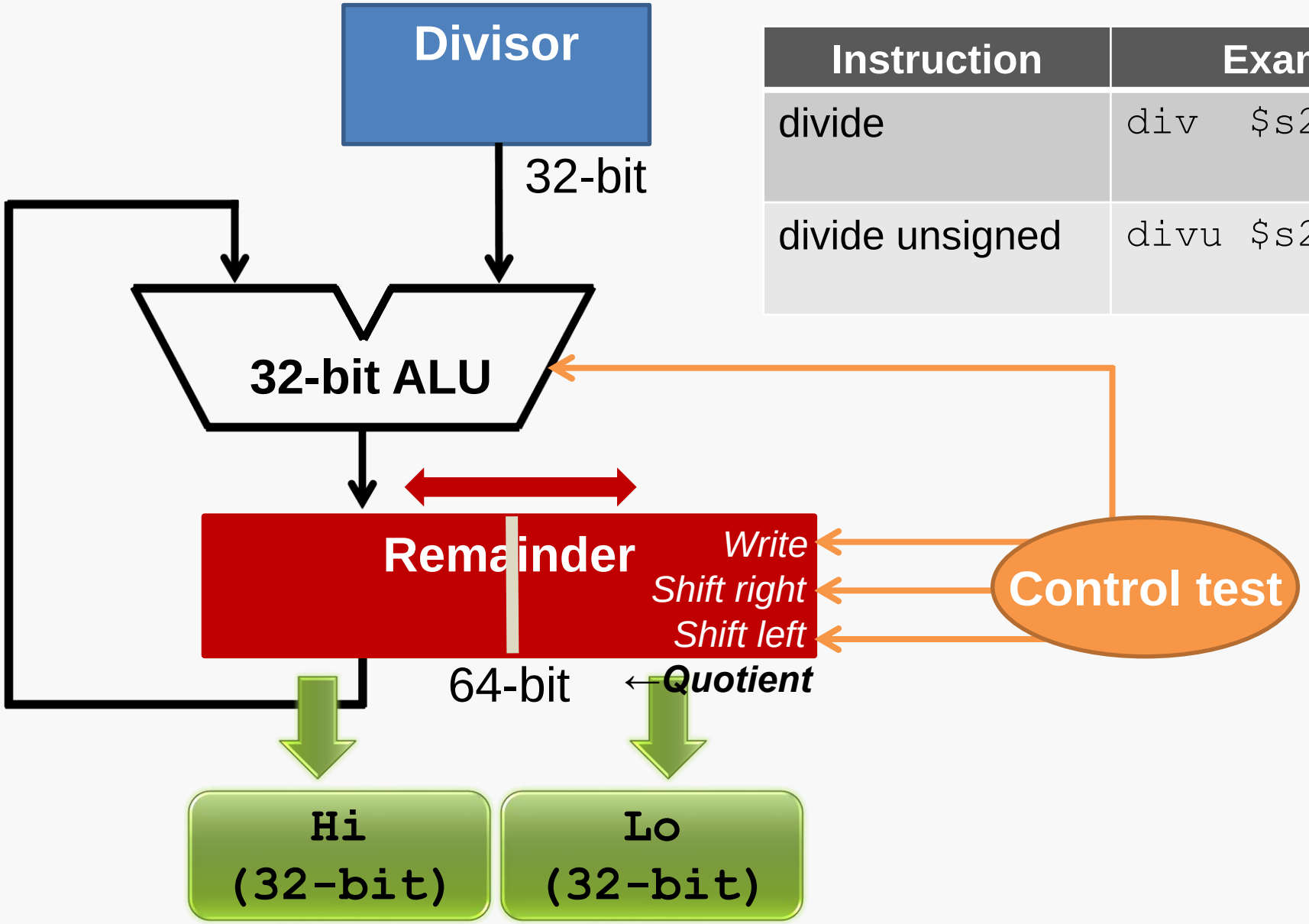
- 乘法
  - Hi/Lo联合用于保存64-bit乘积
- 除法
  - Lo保存商（32-bit）
  - Hi保存余数（32-bit）
- 如何读取Hi/Lo寄存器

Instruction	Example	Meaning
move from Hi	`mfhi $s1`	`$s1=Hi`
move from Lo	`mflo $s1`	`$s1=Lo`

MIPS的32个通用寄存器

编号	名称	编号	名称
0	`$zero`	24-25	`$t8-$t9`
1	`$at`	26-27	`$k0-$k1`
2-3	`$v0-$v1`	28	`$gp`
4-7	`$a0-a3`	29	`$sp`
8-15	`$t0-$t7`	30	`$fp`
16-23	`$s0-$s7`	31	`$ra`

MIPS的除法指令



Instruction	Example	Meaning
divide	`div $s2, $s3`	`Lo=$s2/$s3` `Hi=$s2 mod $s3`
divide unsigned	`divu $s2, $s3`	`Lo=$s2/$s3` `Hi=$s2 mod $s3`

x86的乘法指令

MUL指令（无符号乘法）

- 格式：MUL　SRC
- 操作：
  - 8位　：AX←AL×SRC
  - 16位：DX:AX ← AX×SRC
  - 32位：EDX:EAX ← EAX×SRC
  - 32位：RDX:RAX ← RAX×SRC

IMUL指令（带符号乘法）

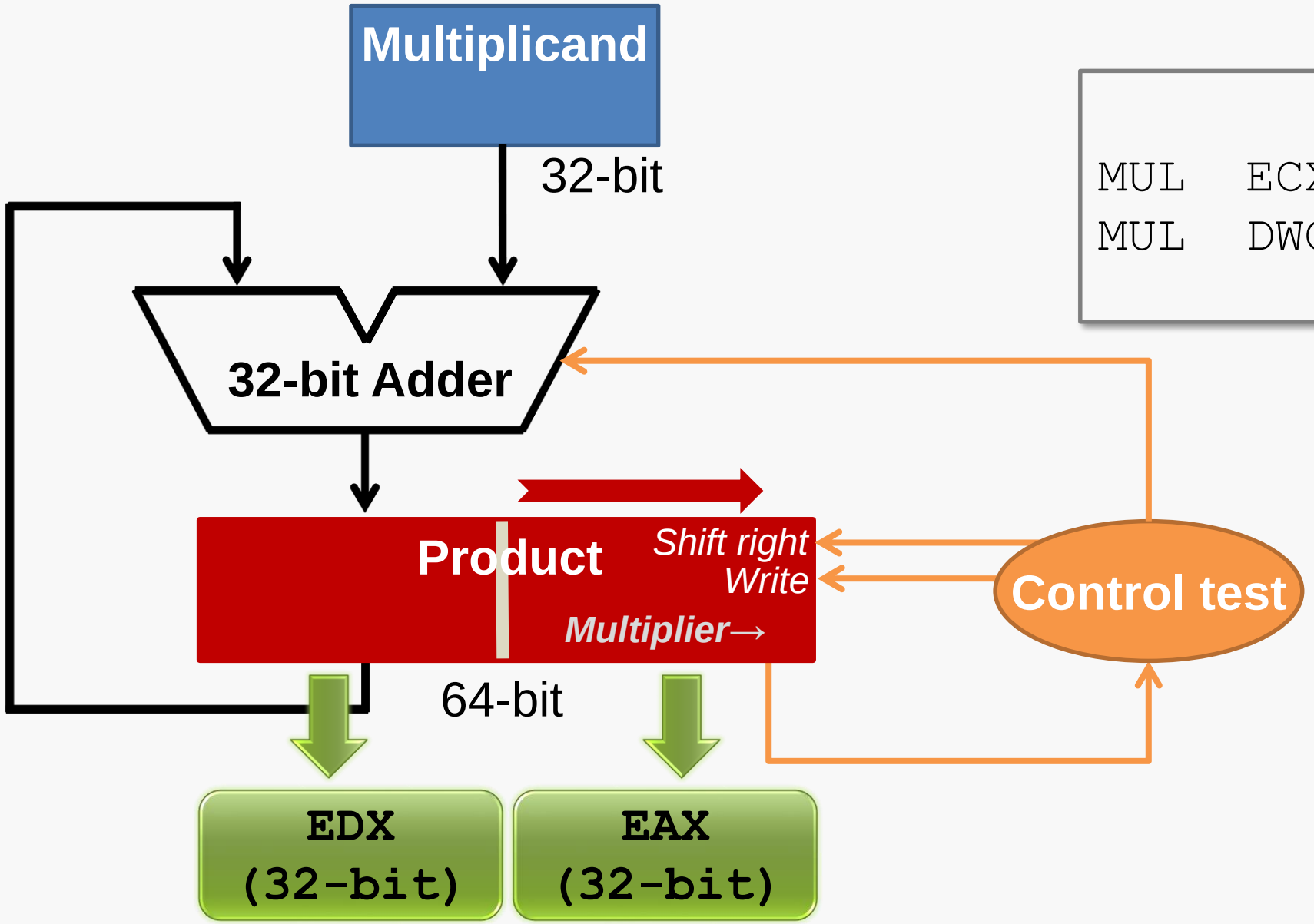
- 格式和操作：同MUL指令
- 说明：操作数和乘积均为带符号数

示例

```
MOV    AL, 6
MOV    BL, 8
MUL    BL
;执行后，AL=AL×BL

MOV    AX, 1000H
MOV    BX, 3000H
MUL    WORD PTR[BX]
;执行后，DX:AX=AX×[BX]
```

x86的乘法指令（32位）



示例

```
MUL   ECX
MUL   DWORD PTR [ESI]
```

x86的除法指令

DIV指令（无符号除法）

- 格式：DIV　SRC
- 操作：
  - 8位　：AL←AL/SRC的商；AH←余数
  - 16位：AX←(DX:AX)/SRC的商；DX←余数
  - 32位：EAX←(EDX:EAX)/SRC的商；EDX←余数
  - 64位：RAX←(RDX:RAX)/SRC的商；RDX←余数

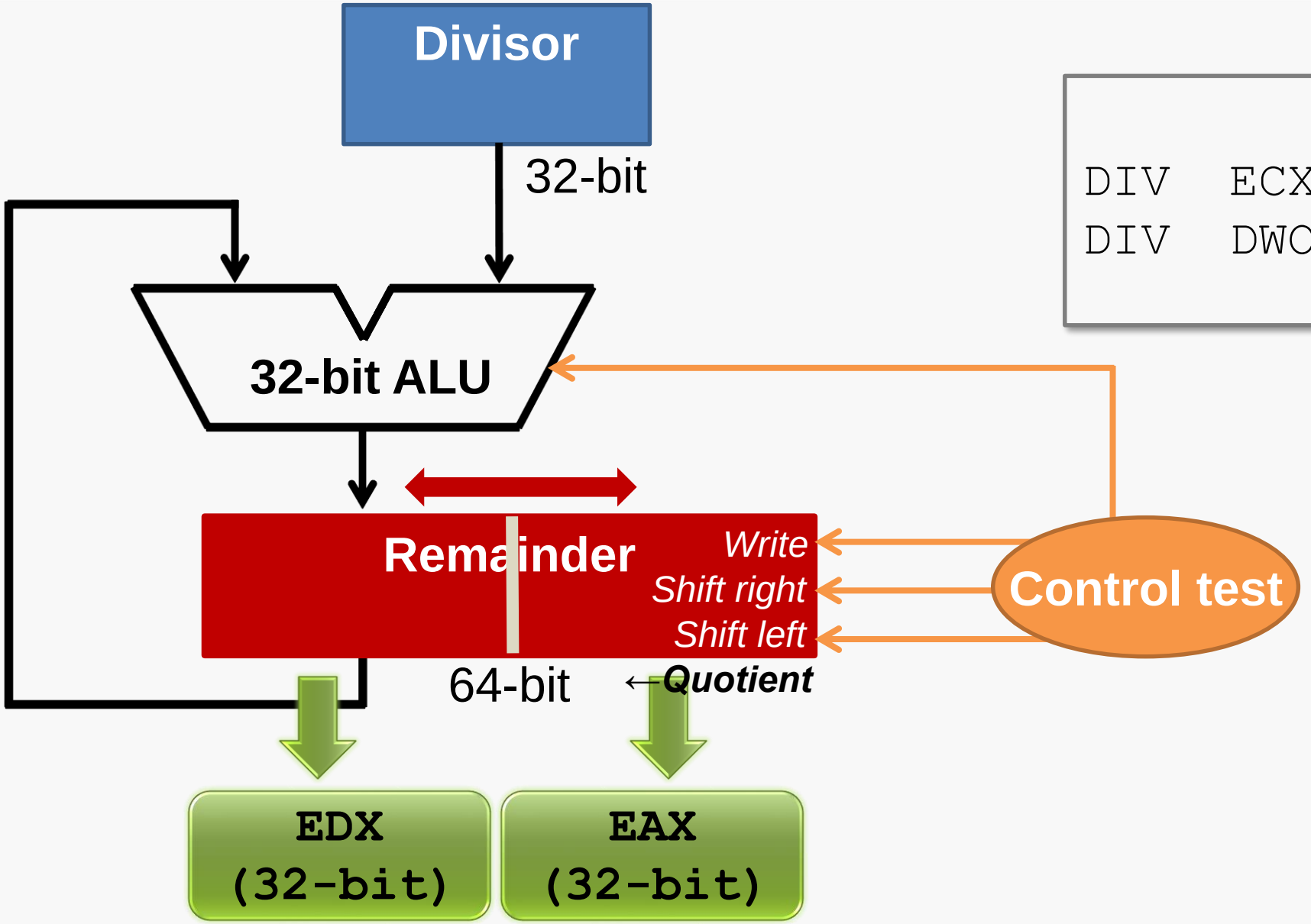
```
MOV   DX, 1234H
MOV   AX, 5678H
MOV   BX, 0108H
DIV   BX
```

示例

IDIV指令（带符号除法）

- 格式和操作：同DIV指令
- 说明：被除数、除数、商、余数均为带符号数；余数符号与被除数相同

x86的除法指令（32位）



示例

```
DIV   ECX
DIV   DWORD PTR [ESI]
```

符号扩展类指令说明

CBW指令（字节扩展为字）

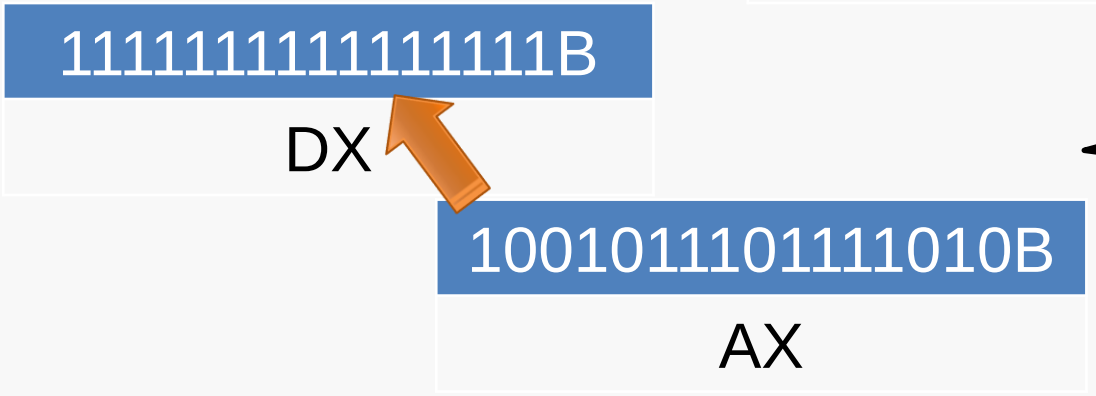
- 格式：CBW
- 操作：将AL的符号位扩展到AH
  - 即：把AL的最高位送到AH的所有位

CWD指令（字扩展为双字）

- 格式：CWD
- 操作：将AX的符号位扩展到DX
  - 即：把AX的最高位送到DX的所有位

符号扩展类指令示例

```
MOV   AL,35H
CBW
ADD   AX,1234H
```




```
MOV   AX, 977AH
CWD
IDIV 5678H
```

注：IA-32新增的相关指令
CWDE指令：将AX符号扩展到EAX
CDQ指令：将EAX符号扩展到EDX:EAX

（4） 控制器

处理器的设计步骤

① 分析指令，得出对数据通路的需求

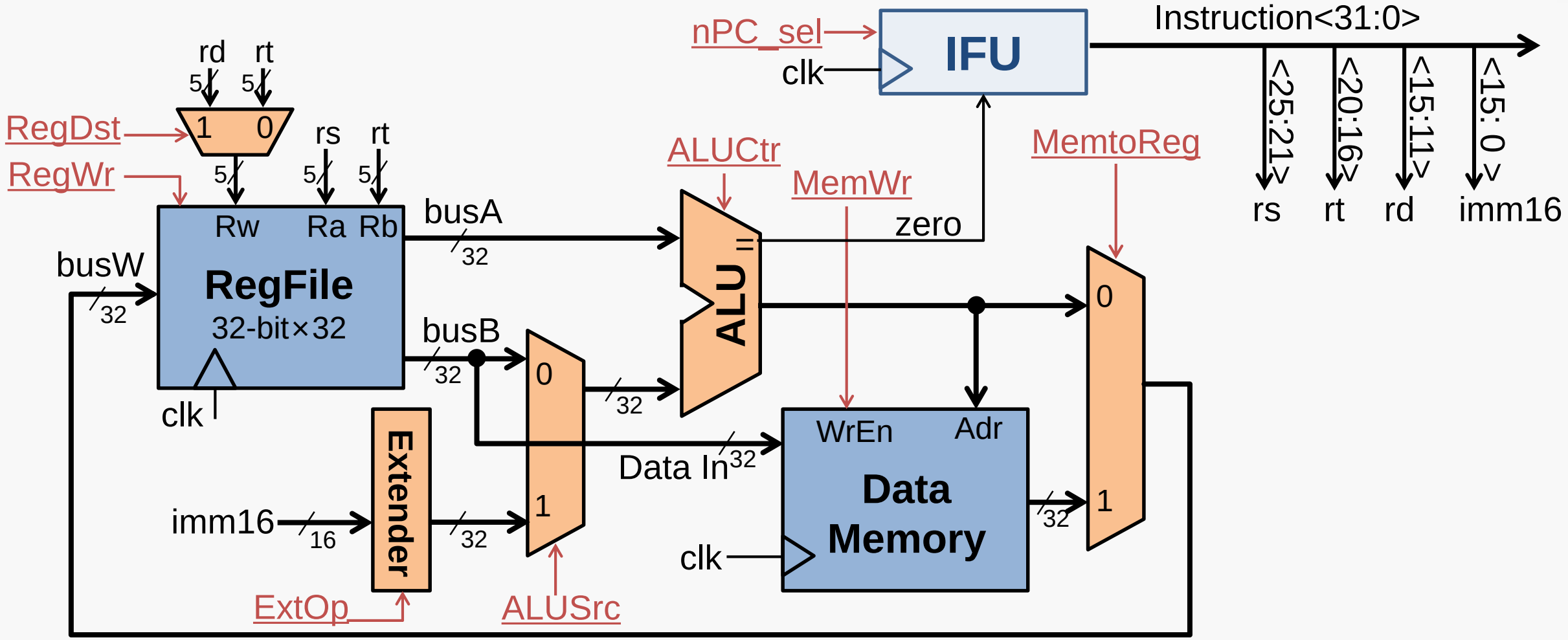
② 为数据通路选择合适的组件

③ 连接组件建立数据通路

④ 分析每条指令的实现，以确定控制信号

⑤ 集成控制信号，形成完整的控制逻辑

现有指令所需的控制信号



控制信号的逻辑表达式

func	100000	100010	/			
opcode(op)	000000	000000	001101	100011	101011	000100
	add	sub	ori	lw	sw	beq
RegDst	1	1	0	0	x	x
ALUSrc	0	0	1	1	1	0
MemtoReg	0	0	0	1	x	x
RegWr	1	1	1	1	0	0
MemWr	0	0	0	0	1	0
nPC_sel	0	0	0	0	0	1
ExtOp	x	x	0	1	1	x
ALUctr<1:0>	00(ADD)	01(SUB)	10(OR)	00(ADD)	00(ADD)	01(SUB)

慕课回顾

运算器和控制器

北京大学·慕课
计算机组成
制作人：陆俊林