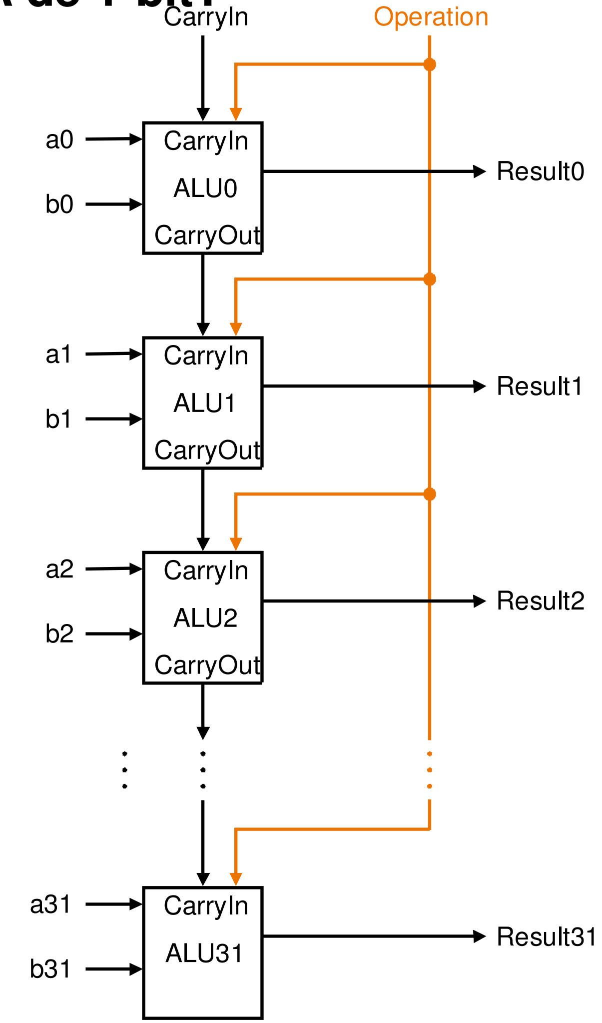
Aritmética Computacional
(Multiplicação e Divisão)

Capítulo 4

Problema: ripple carry adder é lento

- **Uma ULA de 32-bits é tão rápida quanto uma ULA de 1-bit?**
  atraso (ent ⇒ soma ou carry = 2G)
  n estágios ⇒ 2nG
- **Existe mais de uma forma de adição?**
  - **Dois extremos:**
    ripple carry (2nG)
    sum-of-products (2G)



$c_1 = b_0c_0 + a_0c_0 + a_0b_0$

$c_2 = b_1c_1 + a_1c_1 + a_1b_1 \qquad c_2 =$

$c_3 = b_2c_2 + a_2c_2 + a_2b_2 \qquad c_3 =$

$c_4 = b_3c_3 + a_3c_3 + a_3b_3 \qquad c_4 =$

Carry-lookahead adder

- **Uma abordagem entre os dois extremos**
- **Motivação:**
  - **Se não sabemos o valor de carry-in, o que poderíamos fazer?**
  - **Quando sempre geramos um carry?** $g_i = a_i b_i$
  - **Quando propagamos o carry?** $p_i = a_i + b_i$
-

$c_1 = g_0 + p_0 c_0$

$c_2 = g_1 + p_1 c_1 \qquad c_2 =$

$c_3 = g_2 + p_2 c_2 \qquad c_3 =$

$c_4 = g_3 + p_3 c_3 \qquad c_4 =$

Factível!

- **atraso: ent** $\Rightarrow g_i\ p_i$ **(1G)**

  $g_i\ p_i \Rightarrow$ **carry (2G)**

  carry $\Rightarrow$ **saídas (2G)**

$\boxed{\textbf{total: 5G independente de n}}$

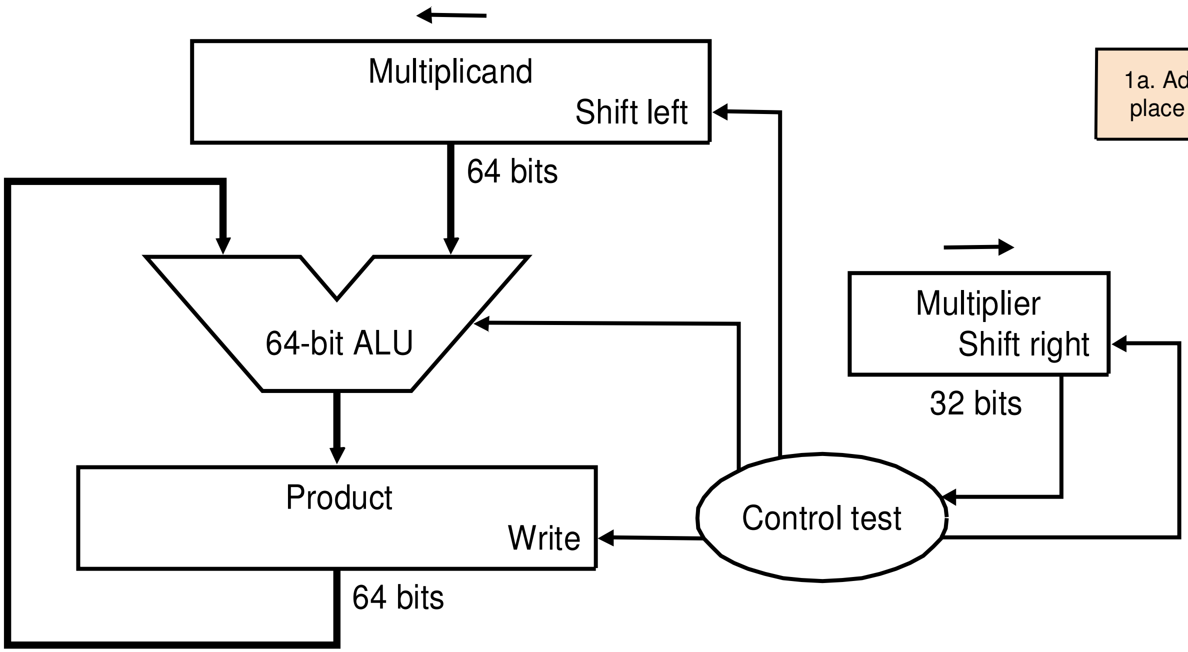
Multiplicação

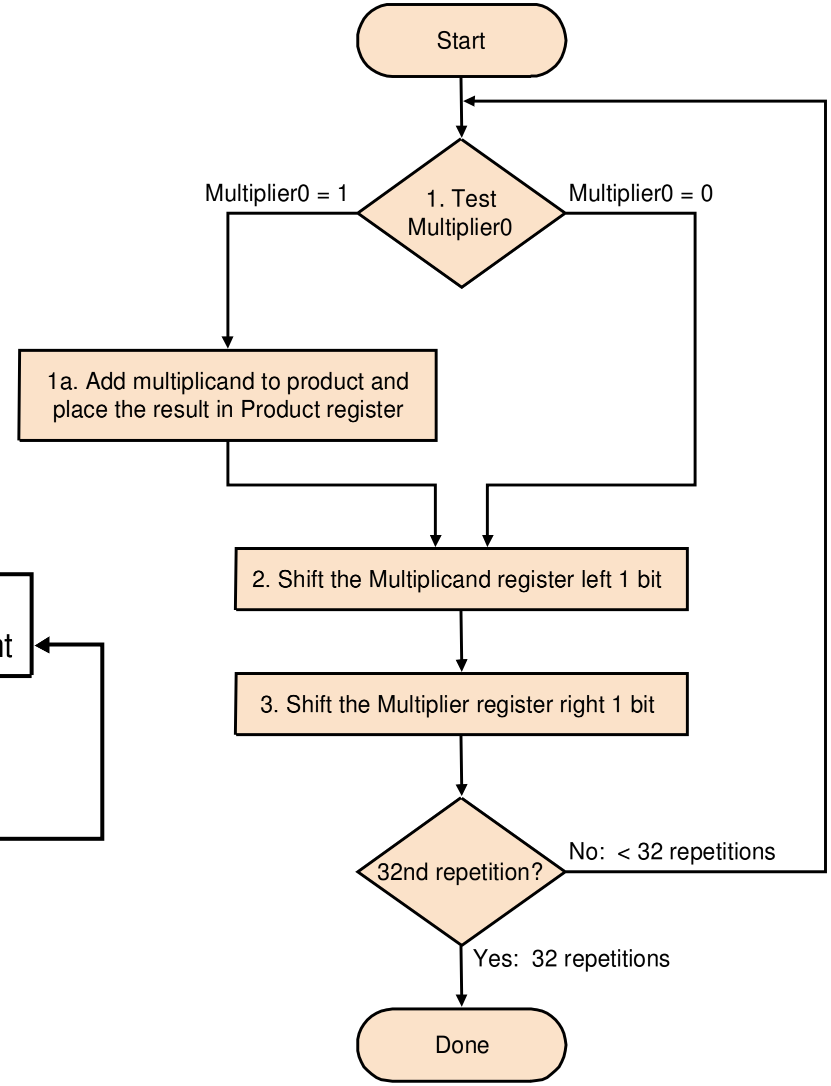
- Mais complexa que a adição
  - Conseguida via deslocamentos e adições
- Mais tempo de processamento e mais área
-

8_{10}	1000	multiplicando
9_{10}	1001	multiplicador
	1000	
	0000	produtos parciais
	0000	
	1000	
72_{10}	1001000	max= $(2^4-1)*(2^4-1) = 225$
		$225 > 128 \Longrightarrow 8$ bits
		$32 * 32$ bits $\Longrightarrow 64$ bits

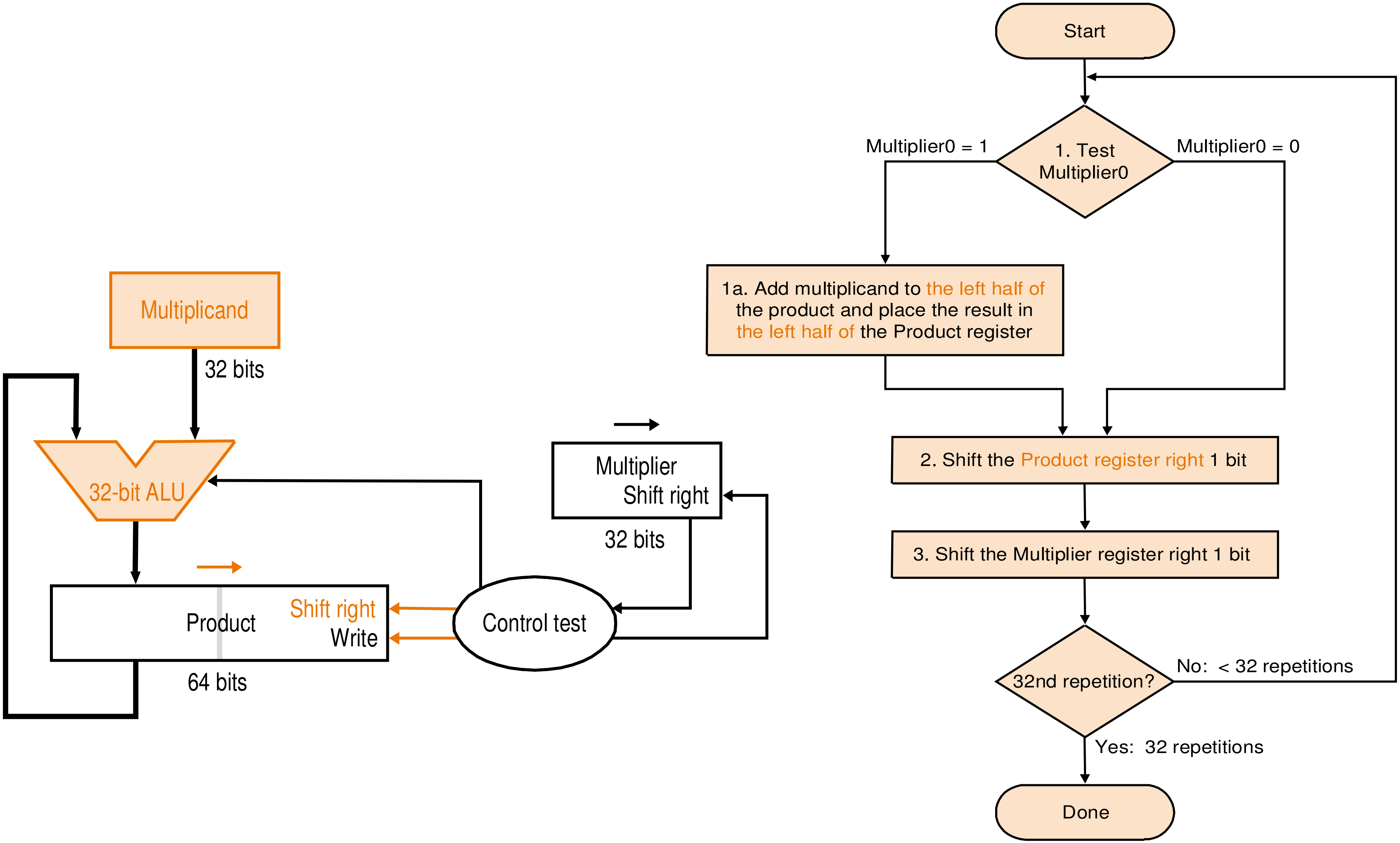
- Números negativos: converter e multiplicar

Implementação da multiplicação

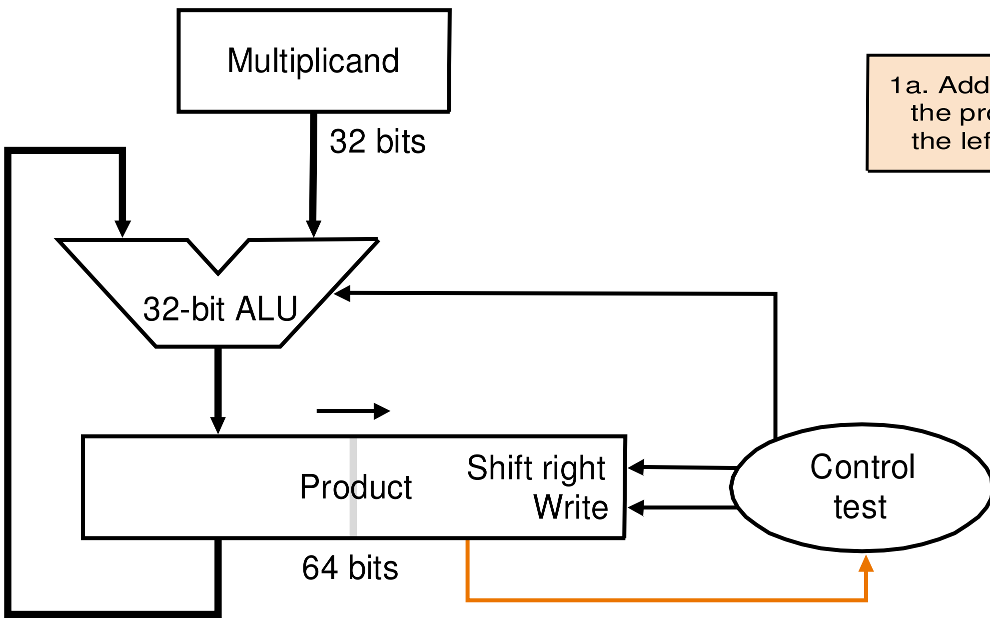


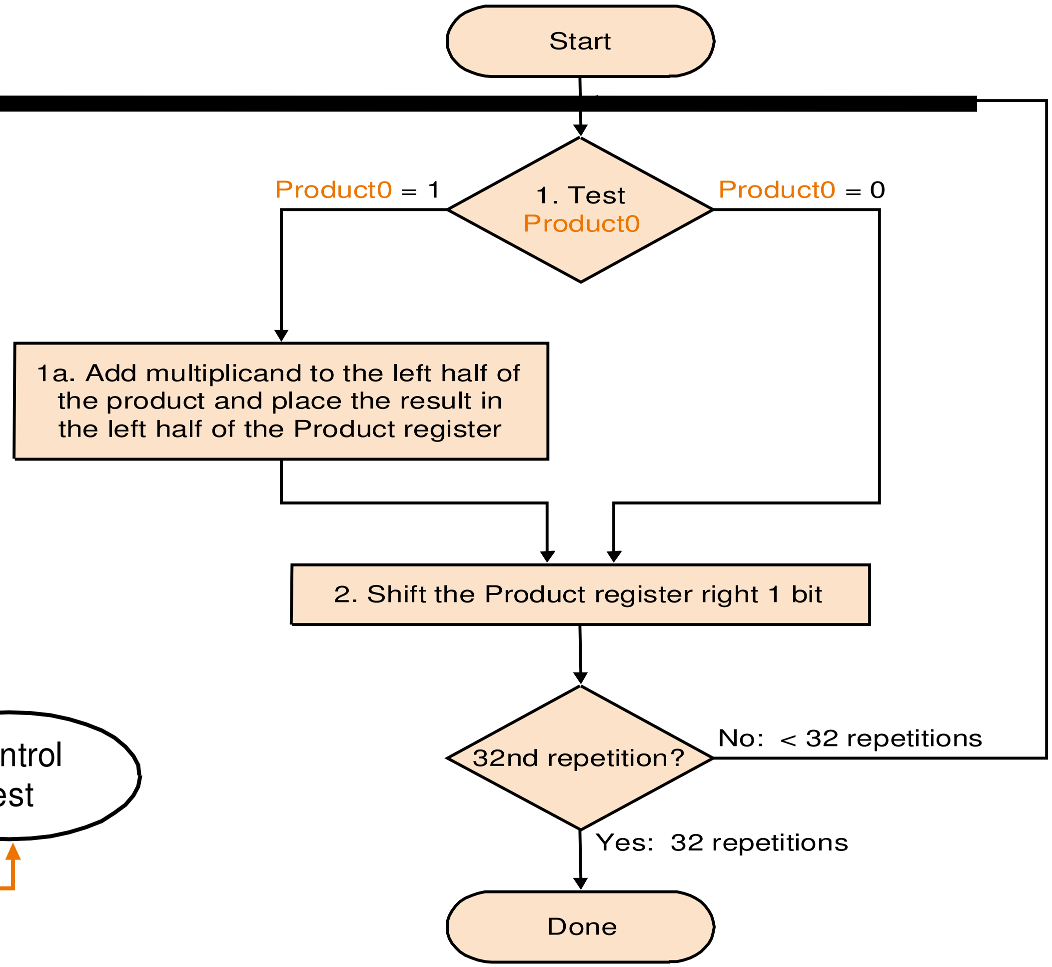


2a. versão: redução do tamanho do reg. multiplicando



Versão final





- **No MIPS:**
  - **dois novos registradores de uso dedicado para multiplicação: Hi e Lo (32 bits cada)**
  - **mult \$t1, \$t2 # Hi Lo ⇐ \$t1 * \$t2**
  - **mfhi \$t1 # \$t1 ⇐ Hi**
  - **mflo \$t1 # \$t1 ⇐ Lo**

Algoritmo de Booth (visão geral)

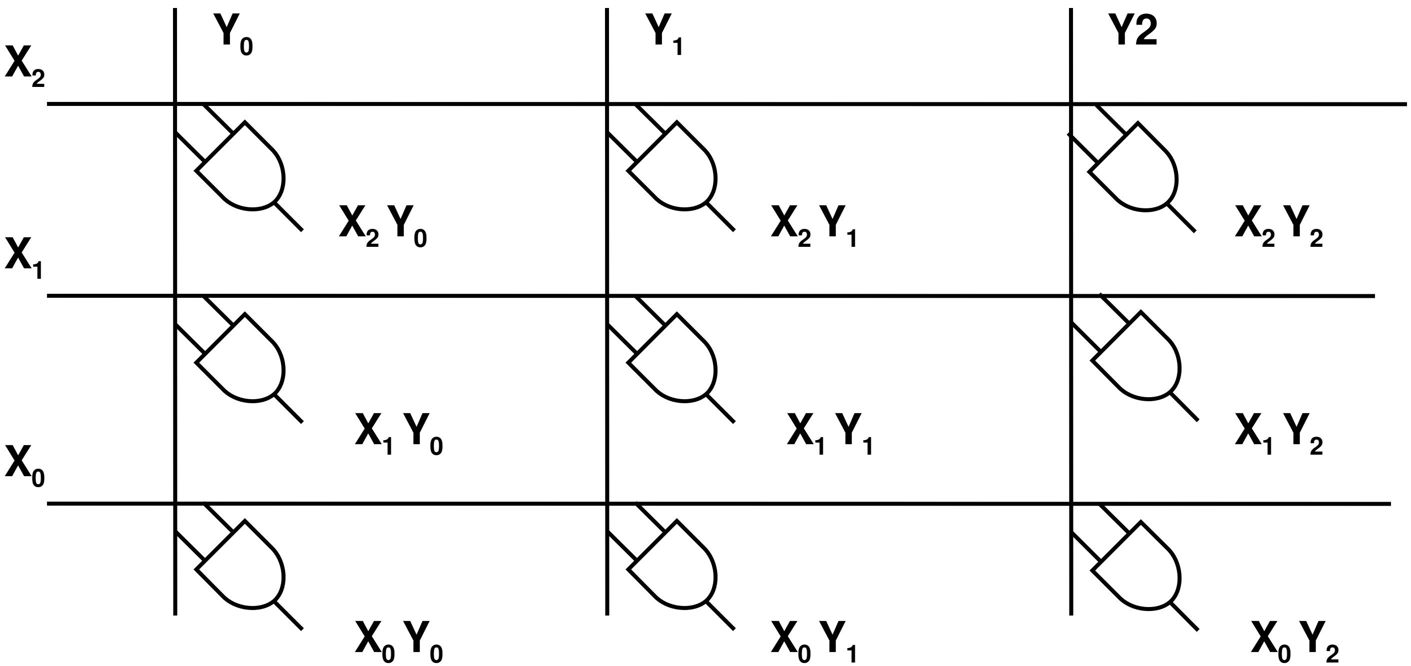
- Idéia: “acelerar” multiplicação no caso de cadeia de “1´s” no multiplicador:

      0 1 1 1 0 * (multiplicando) =
  +   1 0 0 0 0 * (multiplicando)
  -   0 0 0 1 0 * (multiplicando)

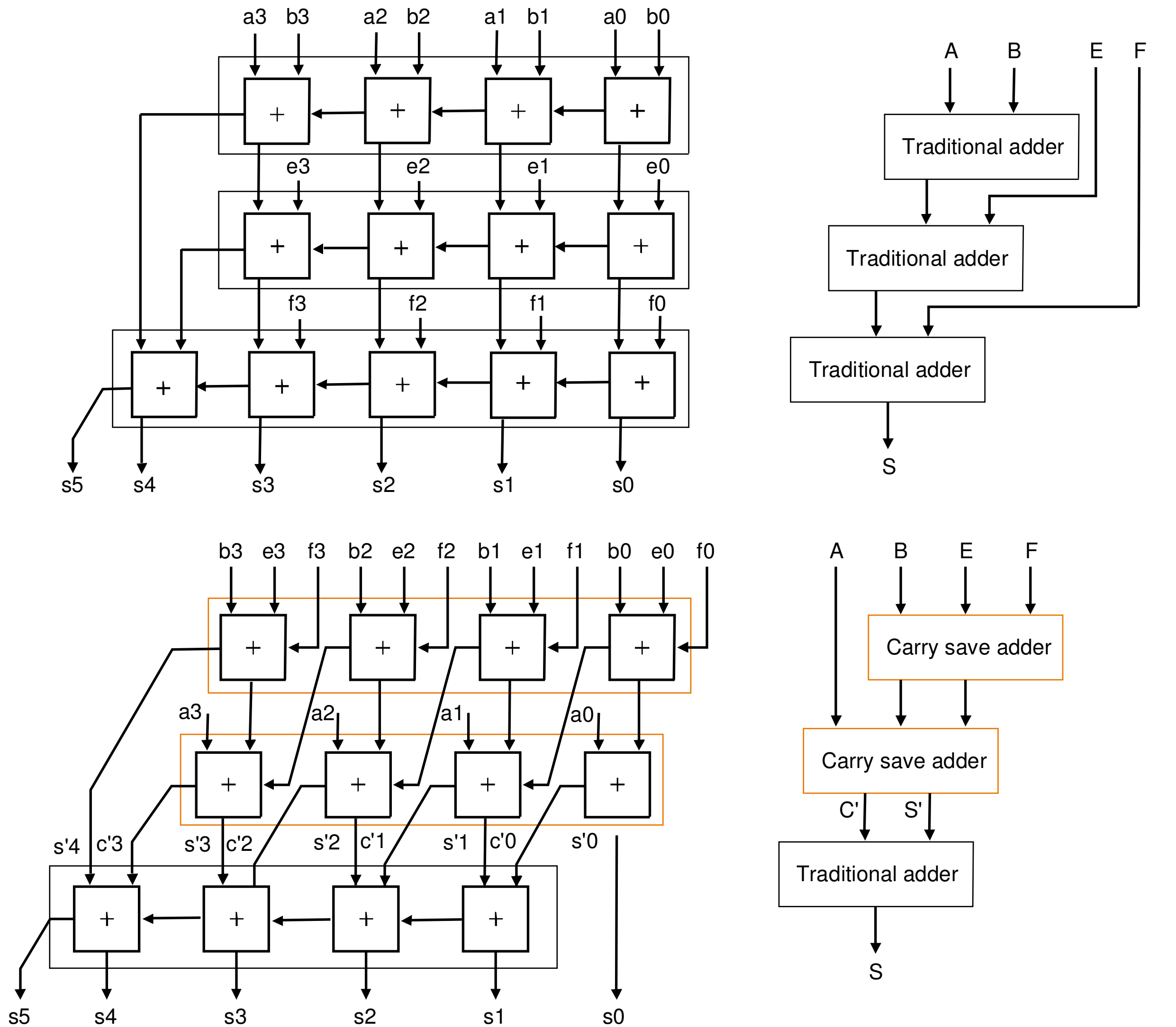
- Olhando bits do multiplicador 2 a 2
  – 00   nada
  – 01   soma (final)
  – 10 subtrai (começo)
  – 11 nada (meio da cadeia de uns)
- Funciona também para números negativos
- Para o curso: só os conceitos básicos
- Algoritmo de Booth estendido
  – varre os bits do multiplicador de 2 em 2
- Vantagens:
  – (pensava-se: shift é mais rápido do que soma)
  – gera metade dos produtos parciais: metade dos ciclos

Geração rápida dos produtos parciais

		Y_0	Y_1	Y_2
		X_0	X_1	X_2
		$X_2\ Y_0$	$X_2\ Y_1$	$X_2\ Y_2$
	$X_1\ Y_0$	$X_1\ Y_1$	$X_1\ Y_2$	
$X_0\ Y_0$	$X_0\ Y_1$	$X_0 Y_2$		



Carry Save Adders (soma de produtos parciais)



Divisão

$29 \div 3 \Rightarrow \quad \underbrace{29}_{\text{dividendo}} = \underbrace{3}_{\text{divisor}} * \underbrace{Q}_{\text{quociente}} + \underbrace{R}_{\text{resto}} = 3 * 9 + 2$

$29_{10} = 011101 \qquad 3_{10} = 11$

$$\begin{array}{r|l} 0\,1\,1\,1\,0\,1 & 1\,1 \\ \cline{2-2} \underline{1\,1} & 0\,1\,0\,0\,1 \\ 0\,0\,1\,0\,1 & \\ \underline{1\,1} & \\ 1\,0 & \end{array}$$

Q = 9 R = 2

Como implementar em hardware?

Alternativa 1: divisão com restauração

- hardware não sabe se "vai caber ou não"
- registrador para guardar resto parcial
- verificação do sinal do resto parcial
- caso negativo ⇒ restauração

$29 - 3 * 2^4 = -19$	$q_4 = 1$	
$-19 + 3 * 2^4 = 29$	$q_4 = 0$	Restauração
$29 - 3 * 2^3 = 5$	$q_3 = 1$	
$5 - 3 * 2^2 = -7$	$q_2 = 1$	
$-7 + 3 * 2^2 = 5$	$q_2 = 0$	Restauração
$5 - 3 * 2^1 = -1$	$q_1 = 1$	
$-1 + 3 * 2^1 = 5$	$q_1 = 0$	Restauração
$5 - 3 * 2^0 = 2$	$q_0 = 1$	

$R = 10 = 2$ $\quad q_4q_3q_2q_1q_0 = 01001 = 9$

Alternativa 2: divisão sem restauração

Regras

se resto parcial	> 0	próxima operação	subtração	
se resto parcial	< 0	próxima operação	soma	

objetivo
$R \to 0$

se operação corrente	+	$q_i = \bar{1}$
se operação corrente	-	$q_i = 1$

$29 - 3 * 2^4 = -19 < 0$	próx = SOMA	$q_4 = 1$
$-19 + 3 * 2^3 = 5 > 0$	próx = SUB	$q_3 = \bar{1}$
$5 - 3 * 2^2 = -7 < 0$	próx = SOMA	$q_2 = 1$
$-7 + 3 * 2^1 = -1 < 0$	próx = SOMA	$q_1 = \bar{1}$
$-1 + 3 * 2^0 = 2$		$q_0 = \bar{1}$

Resto = 2 Quociente = $1\bar{1}1\bar{1}\bar{1}$??

Alternativa 2: conversão do resultado

$$1\,\bar{1}\,1\,\bar{1}\,\bar{1} = (2^4 - 2^3 + 2^2 - 2^1 - 2^0)$$

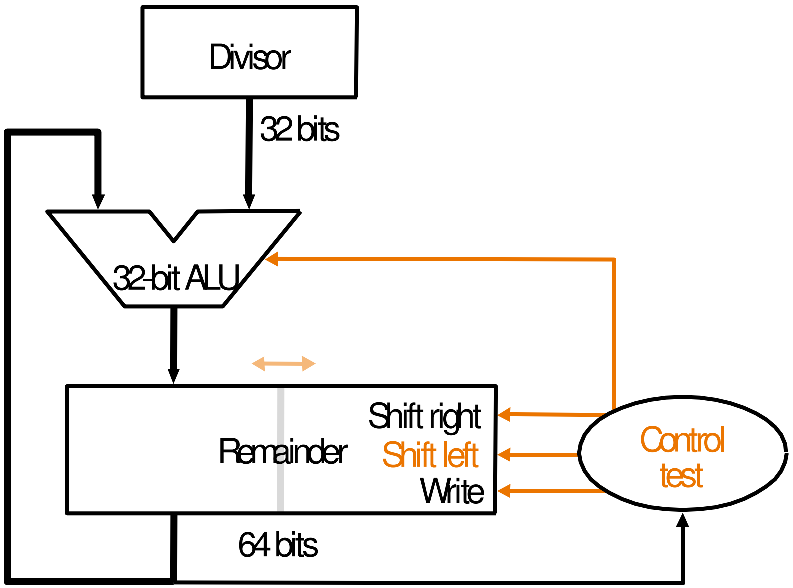
16 - 8 + 4 - 2 - 1

$$...1\,\bar{1}... = 2^n - 2^{(n-1)} = 2^{(n-1)}(2-1) = 2^{(n-1)}$$

$$\begin{array}{r} 1\,\bar{1}\,1\,\bar{1}\,\bar{1} \\ 0\,1\,0\,1\,\bar{1} \\ 0\,1 \end{array}$$

- **Nº de somas: 3**
- **Nº de subtrações:2**
- **Total: 5**
- **OBS: se resto < 0 deve haver correção de um divisor para que resto > 0**

Hardware para divisão: terceira alternativa



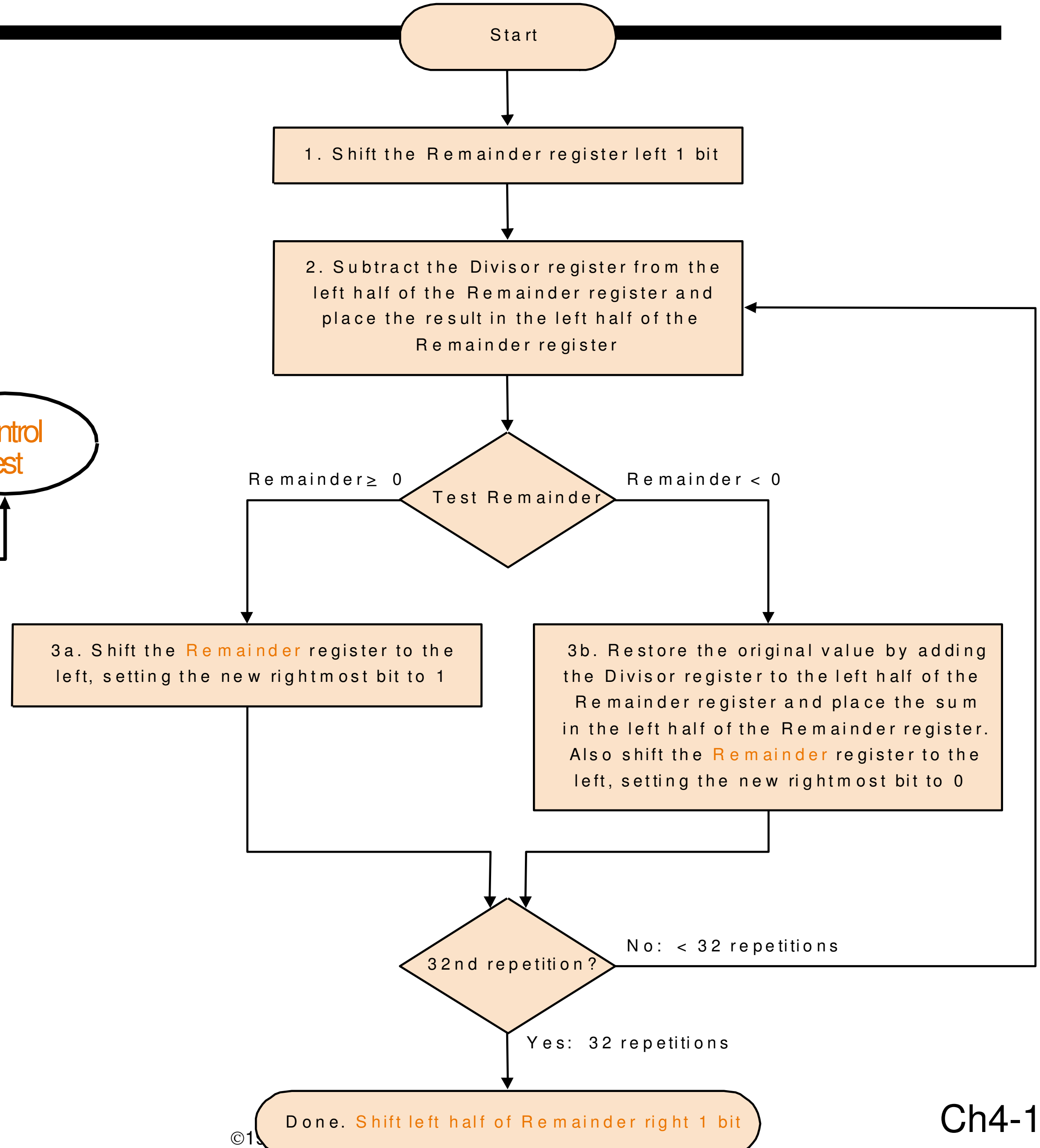


Ilustração da divisão: hardware

$$29 \div 3 \Rightarrow \quad 29 = 3 * Q + R = 3 * 9 + 2$$

dividendo, divisor, quociente, resto

$29_{10} = 0001\ 1101 \qquad 3_{10} = 0011$

```
0 0 0 1 1 1 0 1  | 0 0 1 1
  0 0 1 1        |---------
---------------    1 0 0 1
      0 0 1 0 1
        0 0 1 1
---------------
        0 0 1 0
```

Q = 9 R = 2

Instruções

- No MIPS:
  - dois novos registradores de uso dedicado para multiplicação: Hi e Lo (32 bits cada)
  - mult $t1, $t2 # Hi Lo ⇐ $t1 * $t2
  - mfhi $t1 # $t1 ⇐ Hi
  - mflo $t1 # $t1 ⇐ Lo

- Para divisão:
  - div $s2, $s3 # Lo ⇐ $s2 / $s3
    Hi ⇐ $s2 mod $s3

  - divu $s2, $s3 # idem para “unsigned”