Laboratório de Sistemas
Embarcados e Distribuídos

Aritmética Computacional

Parte II

Histórico de revisões

Revisão	Data	Responsável	Descrição
0.1	03/2016	Prof. Cesar Zeferino	Primeira versão
0.2	04/2017	Prof. Cesar Zeferino	Atualização do modelo

Observação: Este material foi produzido por pesquisadores do Laboratório de Sistemas Embarcados e Distribuídos (LEDS – Laboratory of Embedded and Distributed Systems) da Universidade do Vale do Itajaí e é destinado para uso em aulas ministradas por seus pesquisadores.

Introdução

- **Objetivo**
  - Conhecer o projeto de unidades de lógica e de aritmética utilizadas em processadores
- **Conteúdo**
  - Construção de uma Unidade Aritmética Lógica
  - Multiplicação
  - Divisão
  - Multiplicação e divisão no MIPS

Introdução

Bibliografia

- PATTERSON, David A.; HENNESSY, John L. Abstrações e tecnologias computacionais. *In*: _____. **Organização e projeto de computadores**: a interface hardware/software. 4. ed. Rio de Janeiro: Campus, 2014. cap. 3. Disponível em: <http://www.sciencedirect.com/science/article/pii/B9788535235852000032>. Acesso em: 25 abr. 2017.

- Edições anteriores
  - Patterson & Hennessy (2000, p. 134-160)
  - Patterson & Hennessy (2005, p. 132-142)

2 Construção de uma Unidade Aritmética Lógica

- **Operações aritméticas**
  - ADD: S = A + B
  - SUB: S = A – B = A + (B’ + 1) Complemento de 2

- **Operações lógicas**

AND

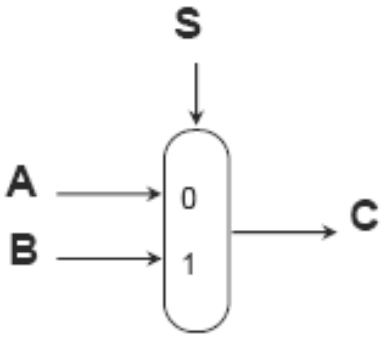
A	B	S
0	0	0
0	1	0
1	0	0
1	1	1

OR

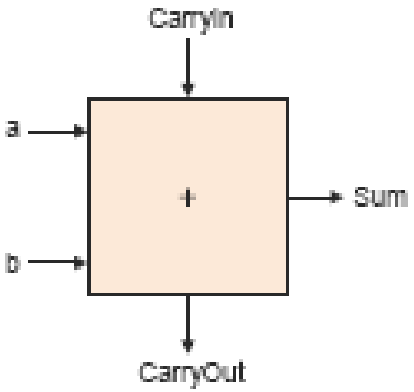
A	B	S
0	0	0
0	1	1
1	0	1
1	1	1

2 Construção de uma Unidade Aritmética Lógica

- **Blocos construtivos**
  - Portas lógicas AND, OR e NOT
  - Multiplexador



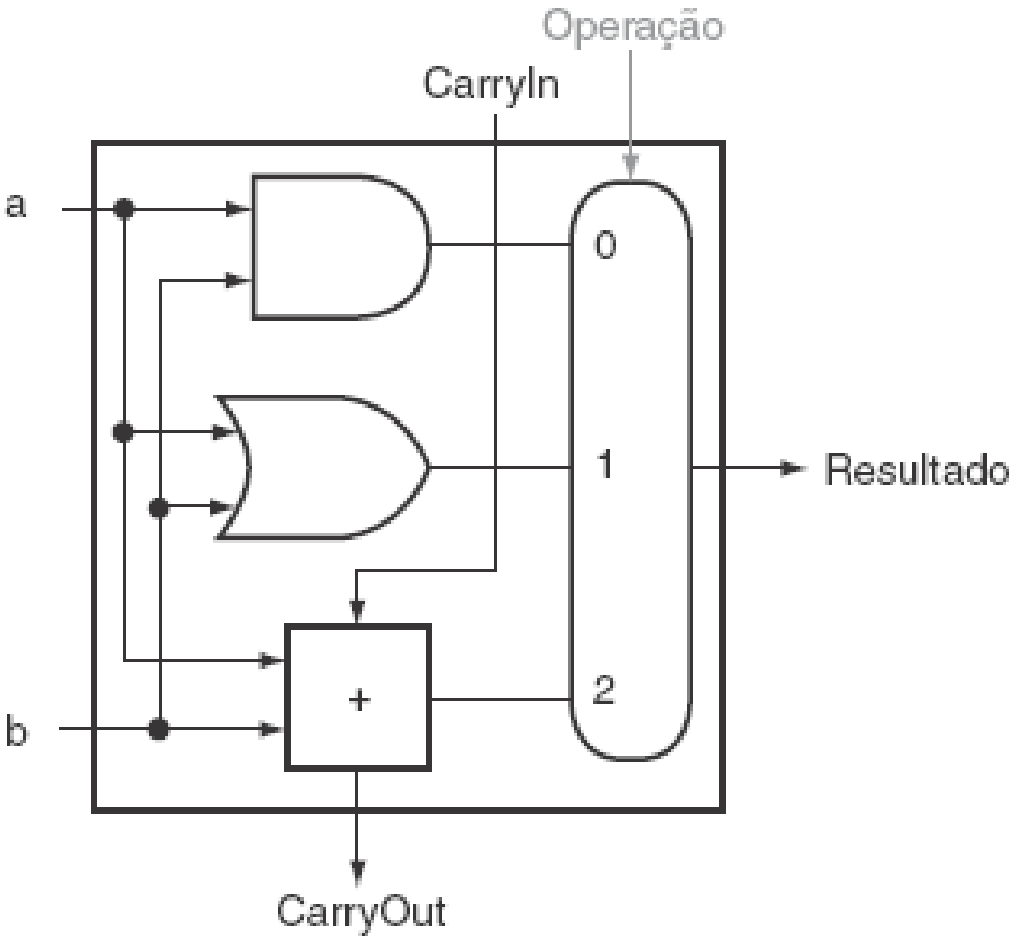
  - Somador completo de 1 bit



$$c_{out} = a\ b + a\ c_{in} + b\ c_{in}$$
$$sum = a\ \mathbf{xor}\ b\ \mathbf{xor}\ c_{in}$$

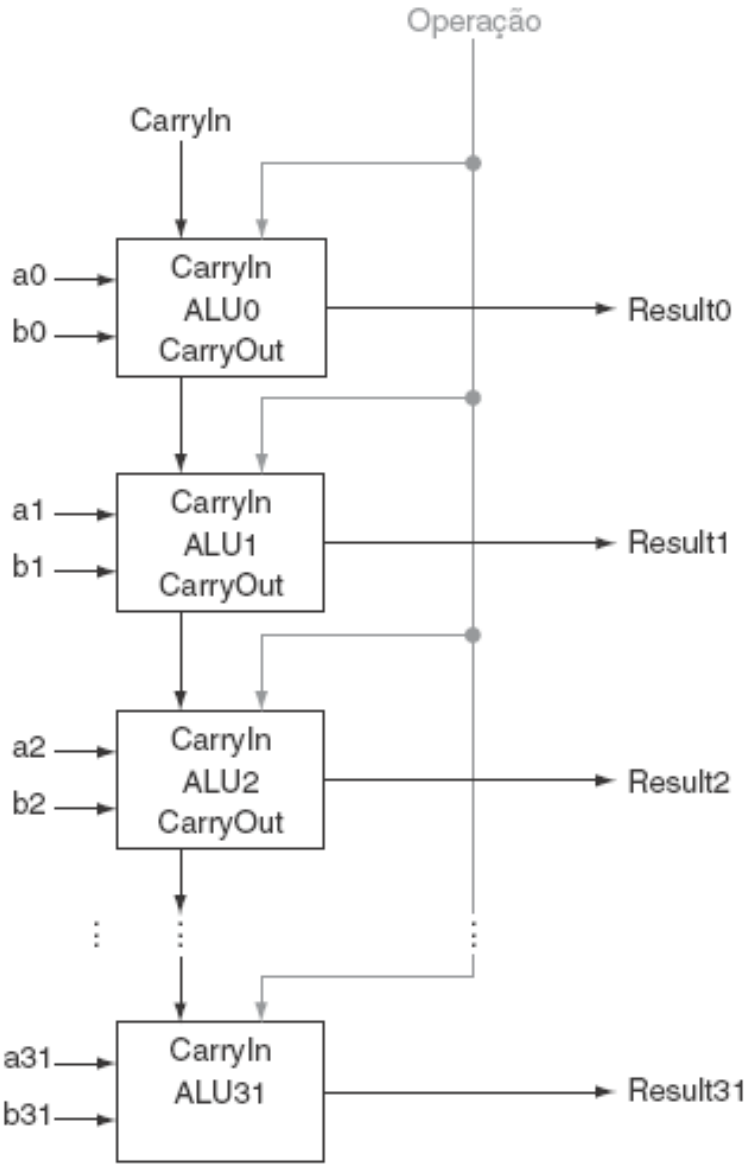
2 Construção de uma Unidade Aritmética Lógica

- **UAL de 1 bit**
  - Operações: AND, OR e ADD



2 Construção de uma Unidade Aritmética Lógica

- **UAL de 32 bits**
  - Operações: AND, OR e ADD



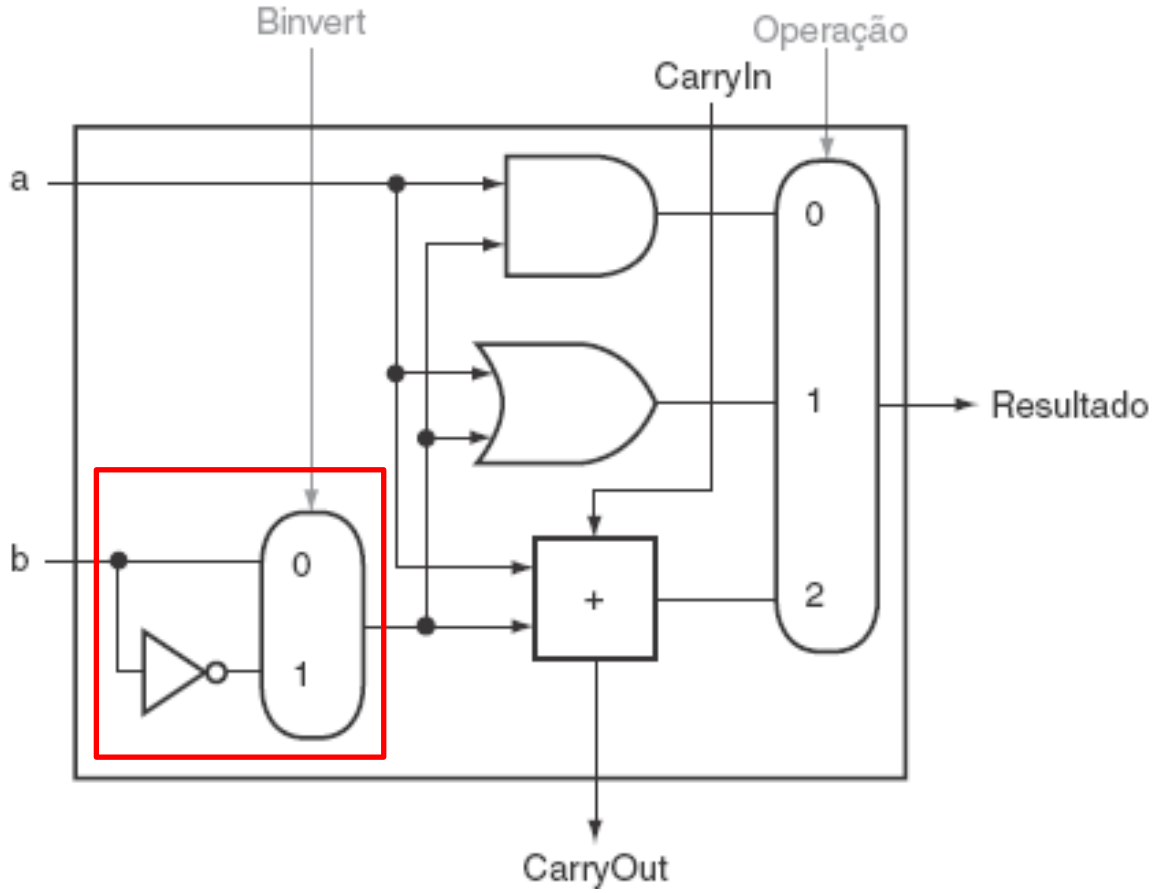
2 Construção de uma Unidade Aritmética Lógica

UAL de 1 bit com subtração

- Incluindo a inversão de B podemos suportar a operação SUB, pois

  A – B = A + B' + 1

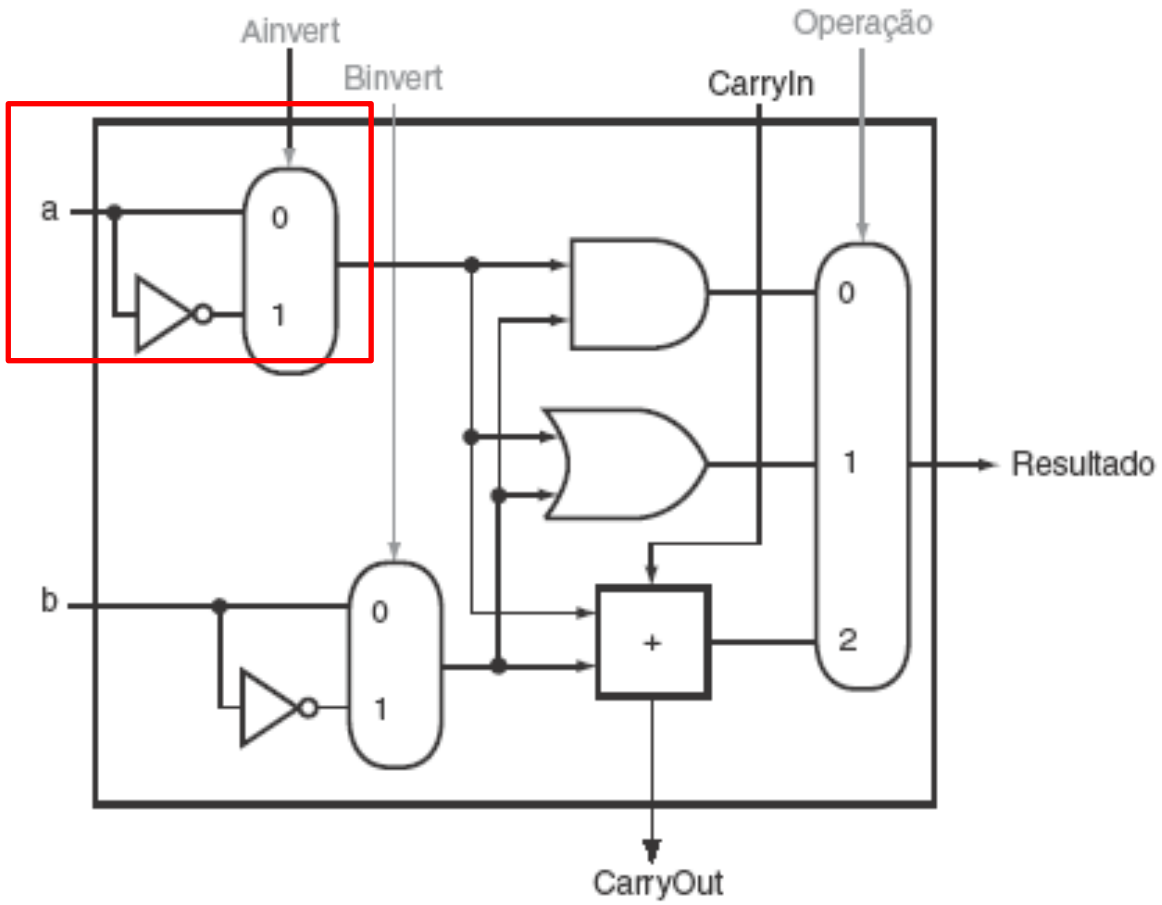
  - +1 é definido fazendo atribuindo 1 ao CarryIn do bit 0



2 Construção de uma Unidade Aritmética Lógica

UAL de 1 bit com suporte a operação NOR

- Incluindo a inversão de A podemos suportar a operação NOR porque **A' and B'= (A' and B')'' = (A or B)'**



2 Construção de uma Unidade Aritmética Lógica

- **Incluindo suporte à instrução SLT**
  - slt rd, rs, rt # rd ← 1, se rs < rt (teste feito com subtração)
    # rd ← 0, se rs >= rt

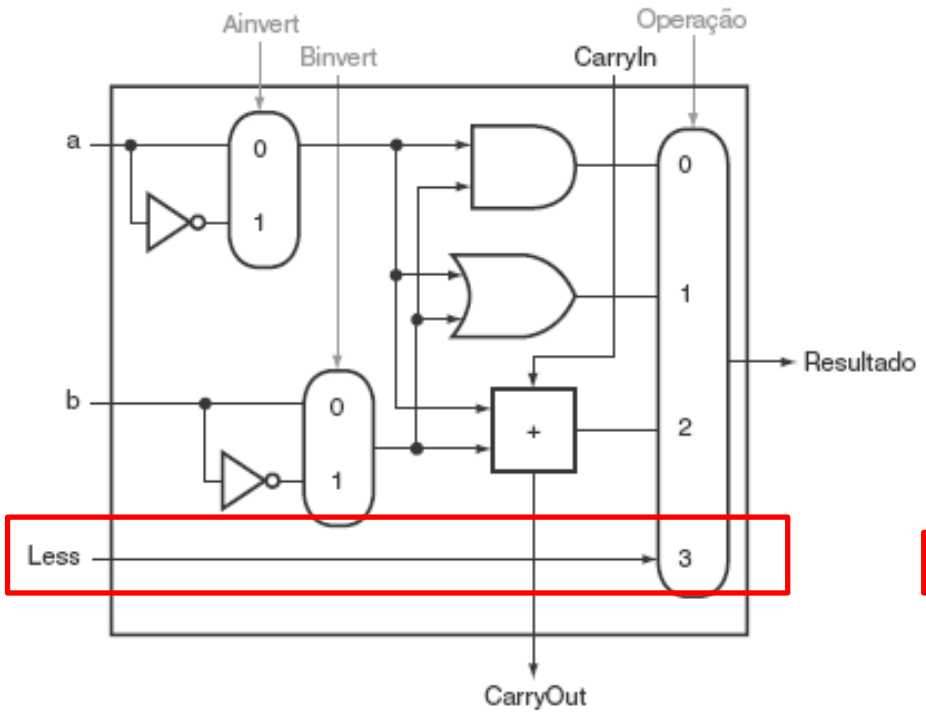
- **Como implementar**
  - Subtrair rs e rt (ou seja A – B)
    1. Se (rs) < (rt) o resultado da subtração será negativo e o bit de sinal será igual a 1, fazer **(rd) ← 1**
    2. Se (rs) >= (rt) o resultado da subtração será positivoe o bit de sinal será igual a 0, fazer **(rd) ← 0**
    3. Ou seja, fazer A – B e atribuir o bit de sinal da UAL do bit mais significativo ao resultado da UAL do bit menos significativo

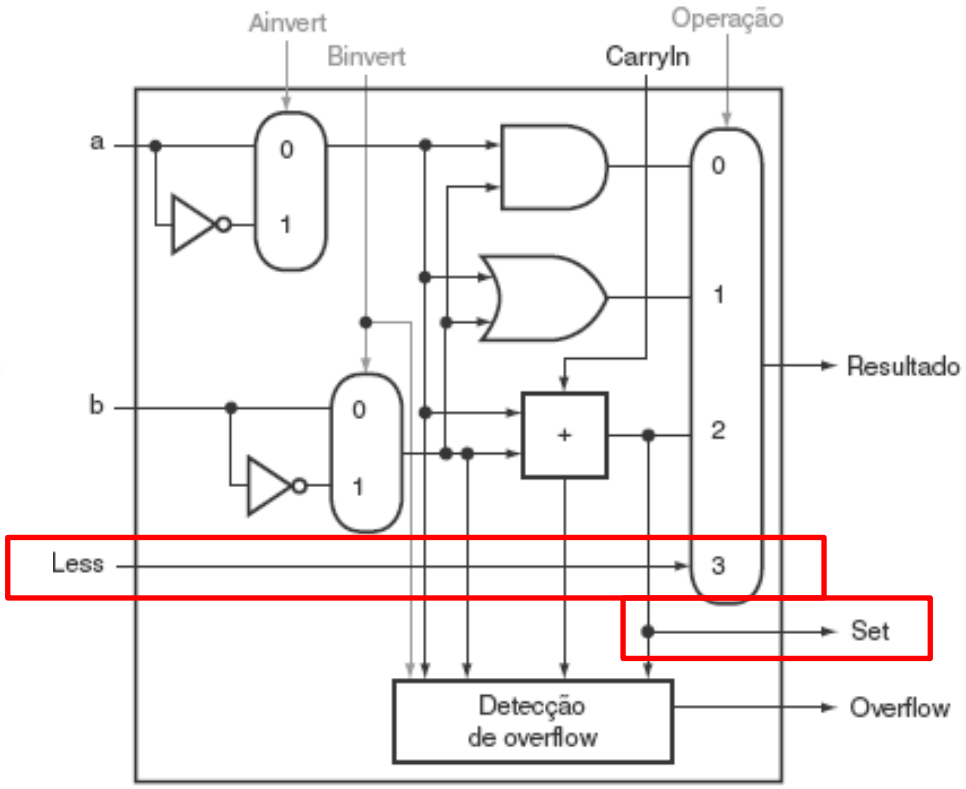
2 Construção de uma Unidade Aritmética Lógica

Incluindo suporte à instrução SLT

- slt rd, rs, rt # rd ← 1, se rs < rt (teste feito com subtração)
  # rd ← 0, se rs >= rt

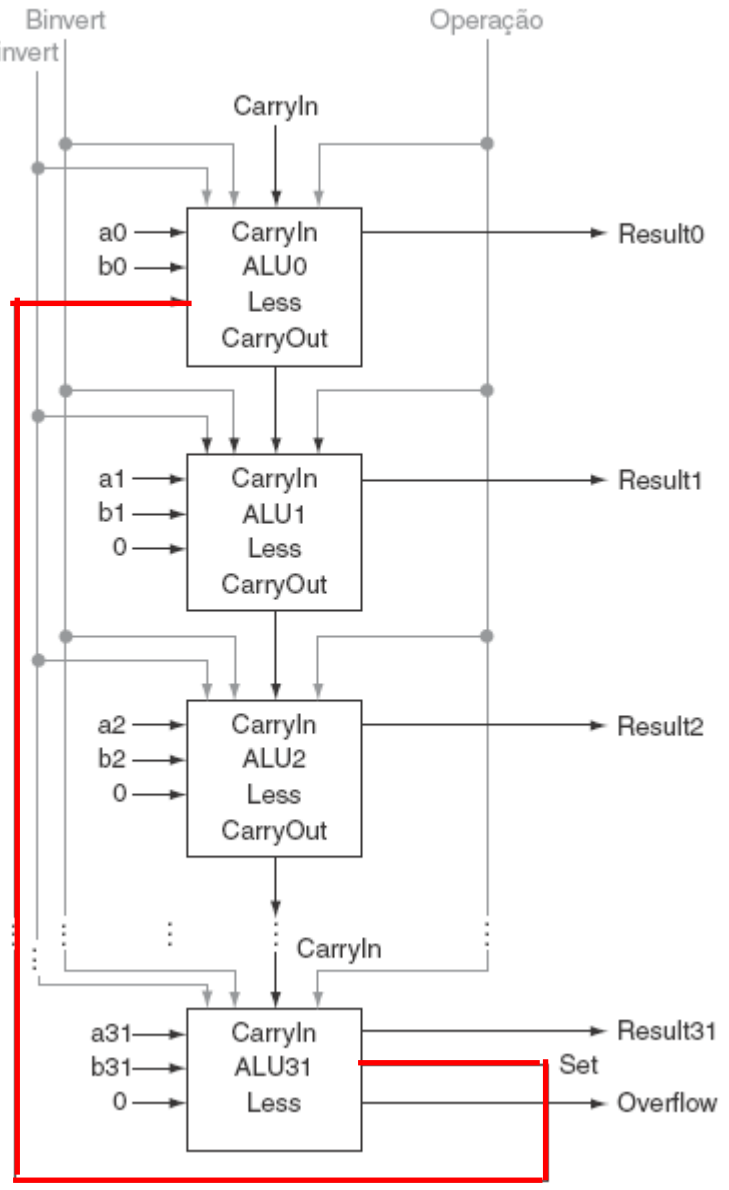
UAL do bit mais significativo





2 Construção de uma Unidade Aritmética Lógica

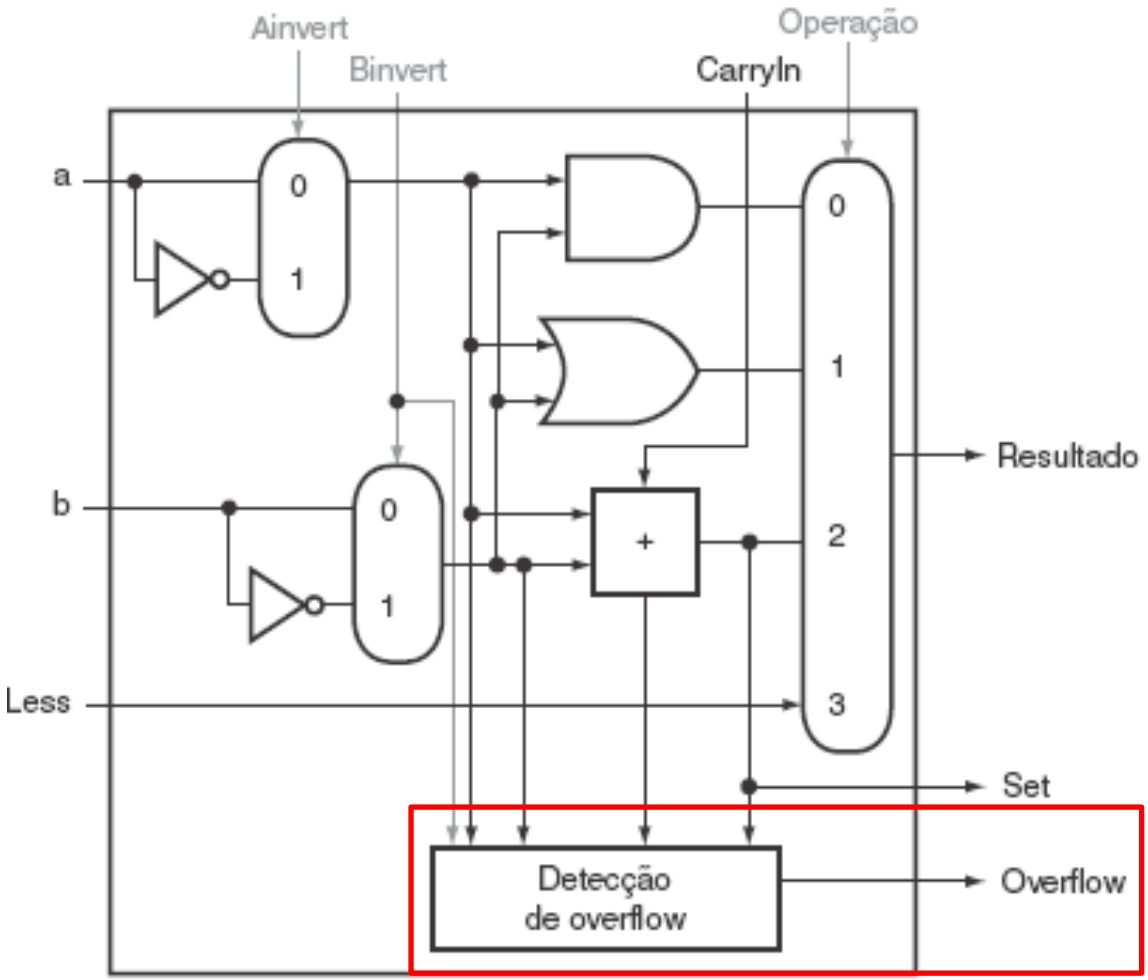
- **Incluindo suporte à instrução SLT**
  - A saída set do bit mais significativo (bit 31) é o ligada no bit de sinal do somador dessa UAL
  - Fazendo: Less(0) = Set
  - Se A – B < 0
    - Set = 1
    - Less(0) = Set = 1
    - Resultado = 000...0001
  - Se A – B >= 0
    - Set = 0
    - Less(0) = Set = 0
    - Resultado = 000...0000



2 Construção de uma Unidade Aritmética Lógica

Detecção de overflow

- Circuito adicional na UAL do bit mais significativo



2 Construção de uma Unidade Aritmética Lógica

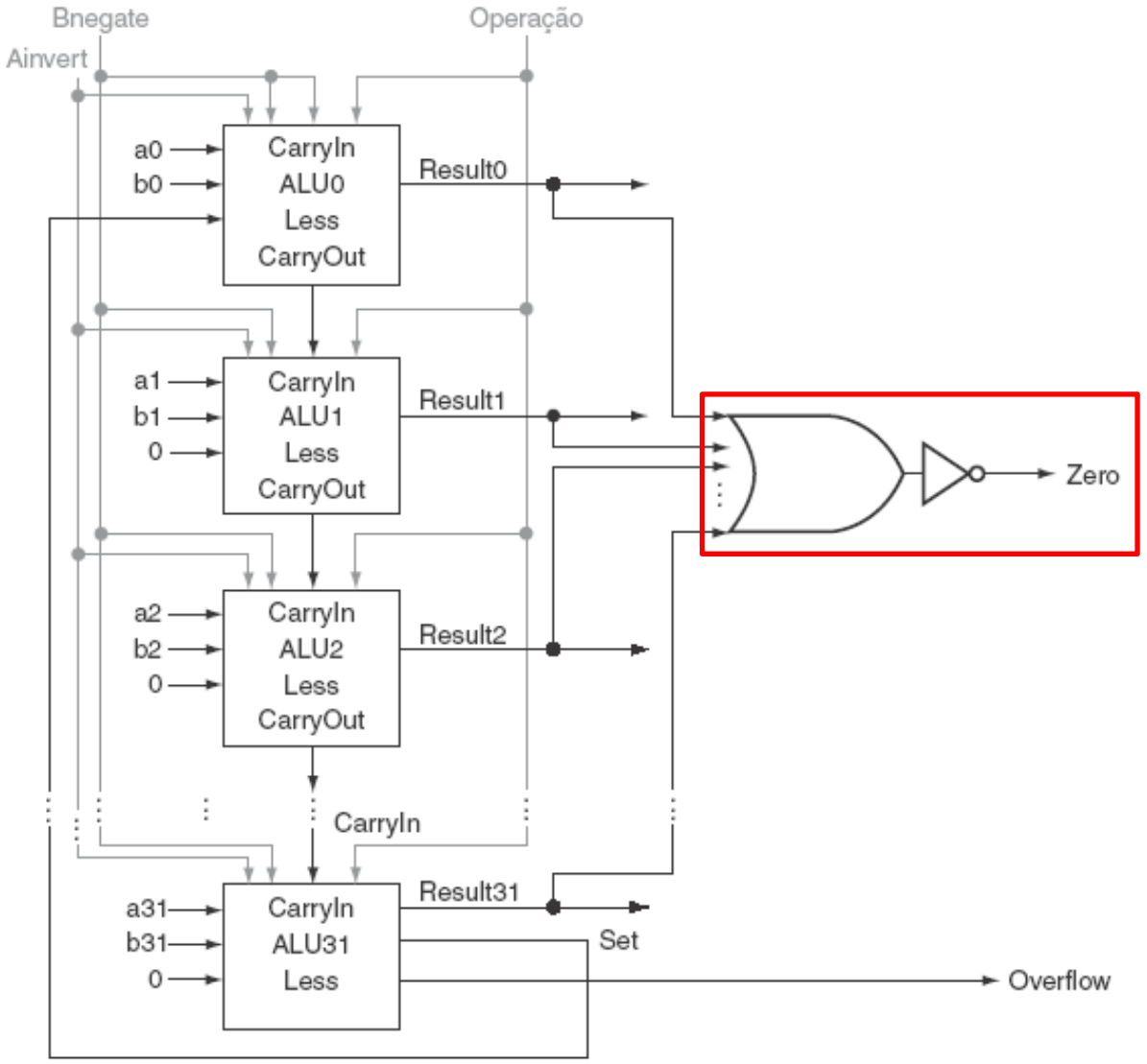
- **Teste de igualdade (para instruções BEQ e BNE)**
  - Como detectar se dois números A e B são iguais?
    1. Fazer A – B
    2. Se A = B, Resultado = 0
  - Como detectar se Resultado = 0?
    1. Usar uma operação lógica que resulte 1 apenas quando todos as entradas forem iguais a 0
    2. Solução:

NOR

A	B	S
0	0	1
0	1	0
1	0	0
1	1	0

2 Construção de uma Unidade Aritmética Lógica

- Teste de igualdade (para instruções BEQ e BNE)



3 Multiplicação

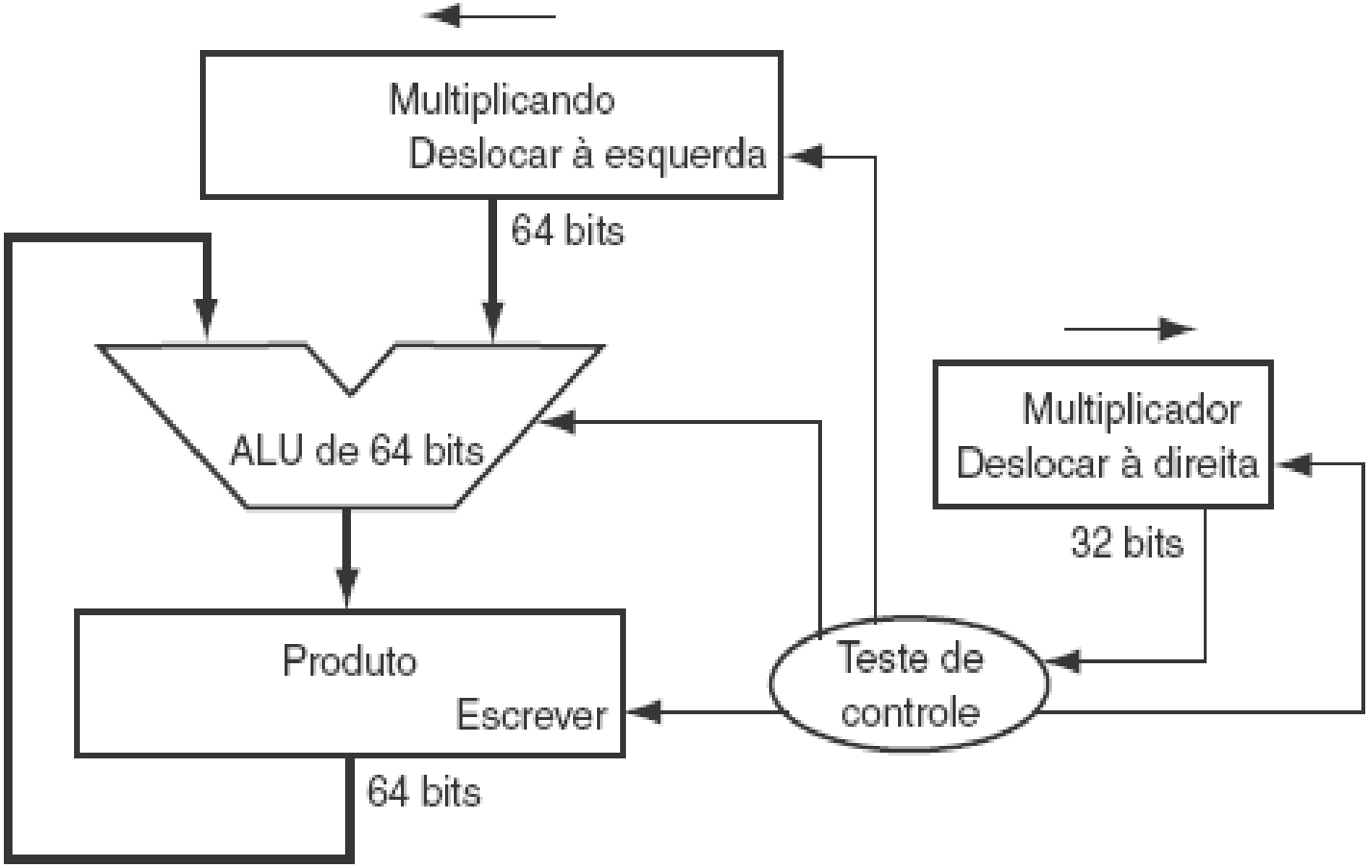
- **Princípio da multiplicação**

Multiplicando	$1000_{(10)}$
Multiplicador	x $1001_{(10)}$
	1000
	0000
	0000
	1000
Produto	$1001000_{(10)}$

- **O número de dígitos do produto é maior que os do multiplicando e do multiplicador**
- **Como implementar em hardware?**

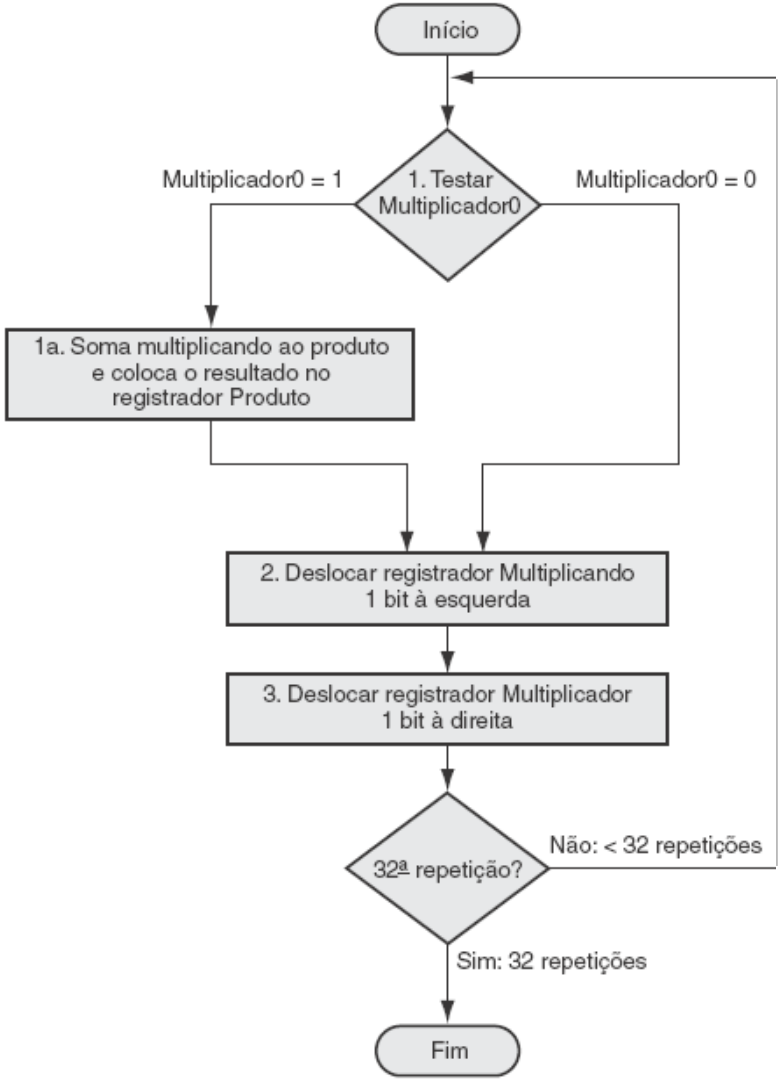
3 Multiplicação

- **Hardware de multiplicação (primeira versão)**
  - Caminho de dados



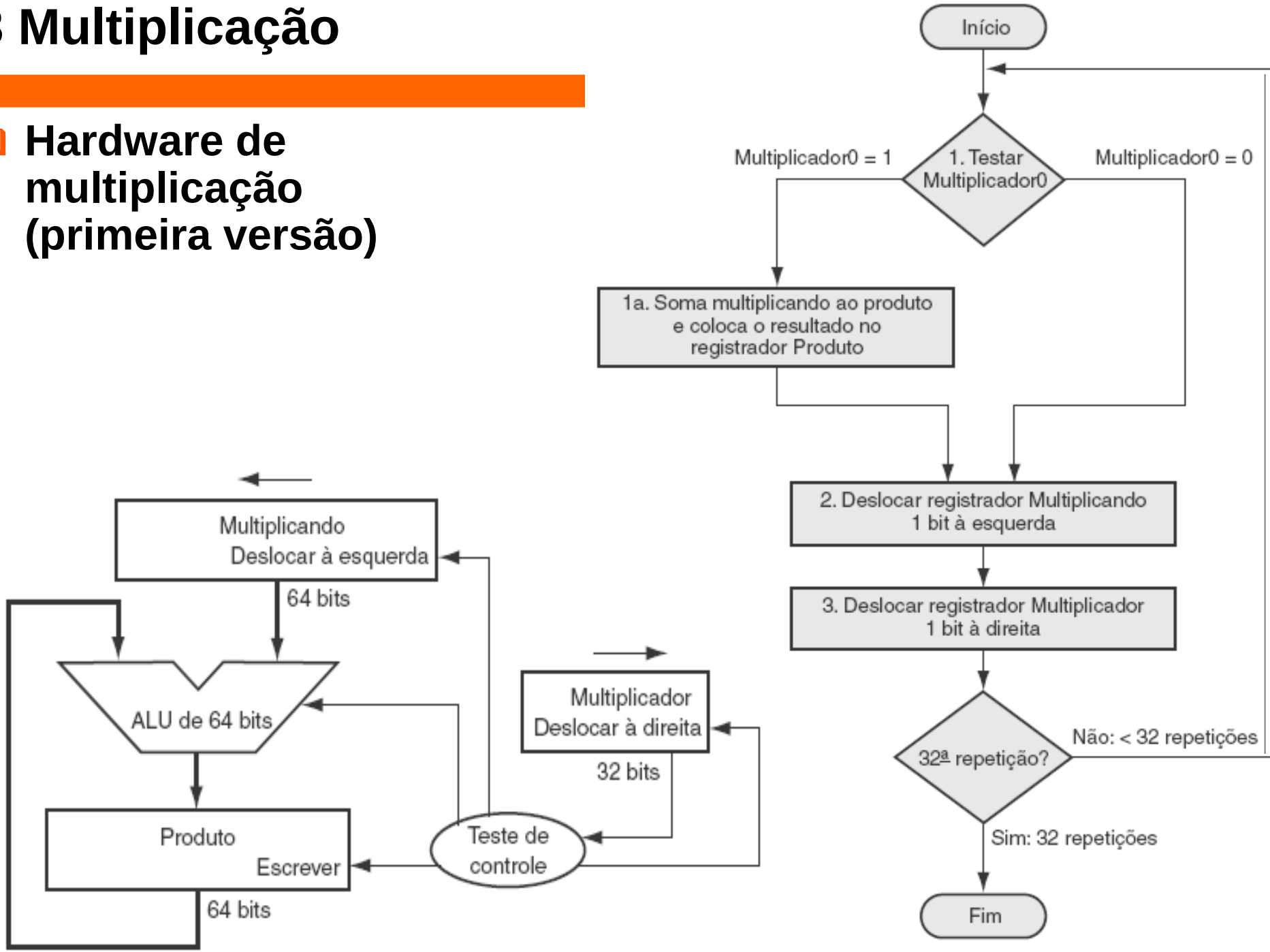
3 Multiplicação

- **Hardware de multiplicação (primeira versão)**
  - Controle



3 Multiplicação

- **Hardware de multiplicação (primeira versão)**



3 Multiplicação

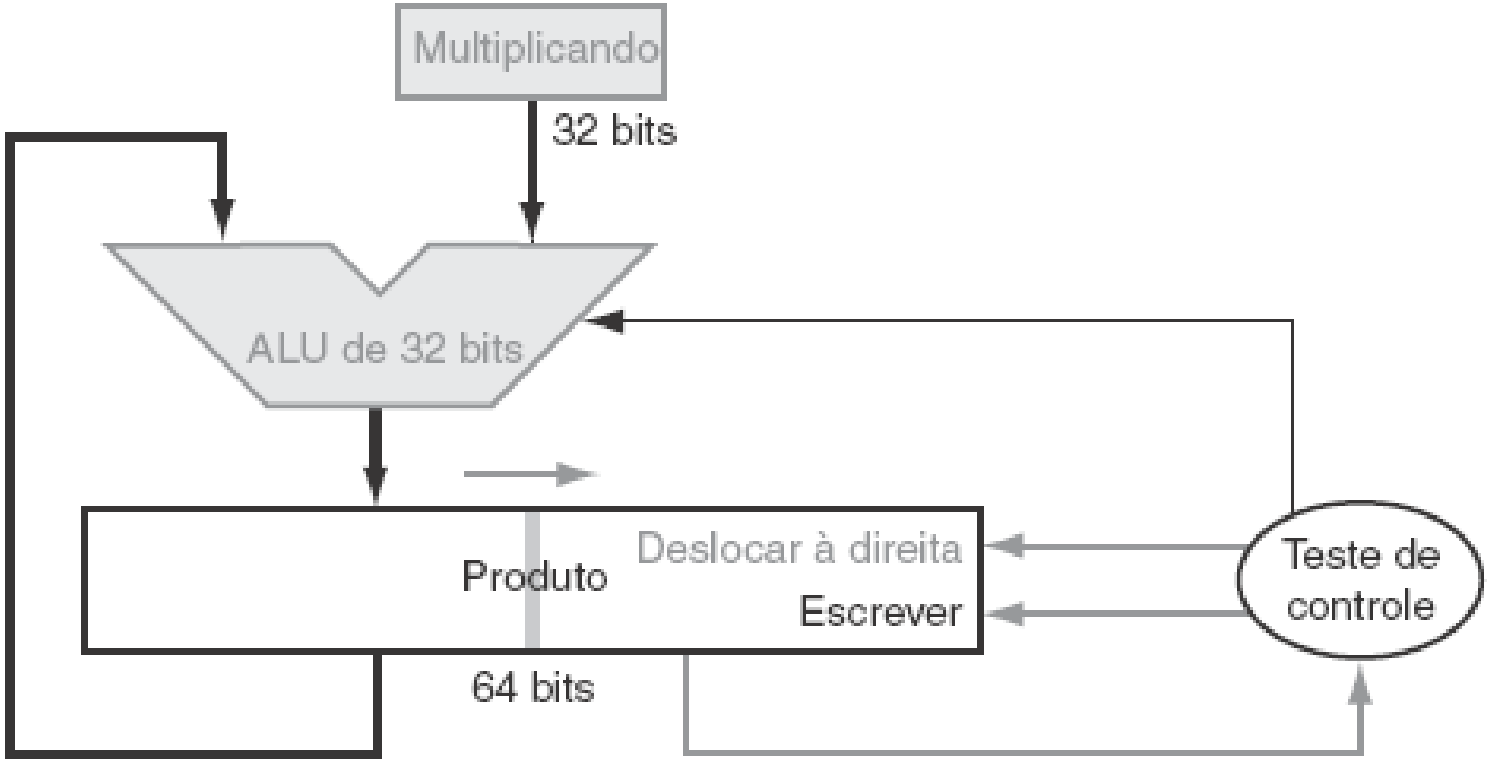
Hardware de multiplicação (primeira versão)

- Exemplo

Iteration	Step	Multiplier	Multiplicand	Product
0	Initial values	0011	0000 0010	0000 0000
1	1a: 1 ⟹ Prod = Prod + Mcand	0011	0000 0010	0000 0010
	2: Shift left Multiplicand	0011	0000 0100	0000 0010
	3: Shift right Multiplier	0001	0000 0100	0000 0010
2	1a: 1 ⟹ Prod = Prod + Mcand	0001	0000 0100	0000 0110
	2: Shift left Multiplicand	0001	0000 1000	0000 0110
	3: Shift right Multiplier	0000	0000 1000	0000 0110
3	1: 0 ⟹ No operation	0000	0000 1000	0000 0110
	2: Shift left Multiplicand	0000	0001 0000	0000 0110
	3: Shift right Multiplier	0000	0001 0000	0000 0110
4	1: 0 ⟹ No operation	0000	0001 0000	0000 0110
	2: Shift left Multiplicand	0000	0010 0000	0000 0110
	3: Shift right Multiplier	0000	0010 0000	0000 0110

3 Multiplicação

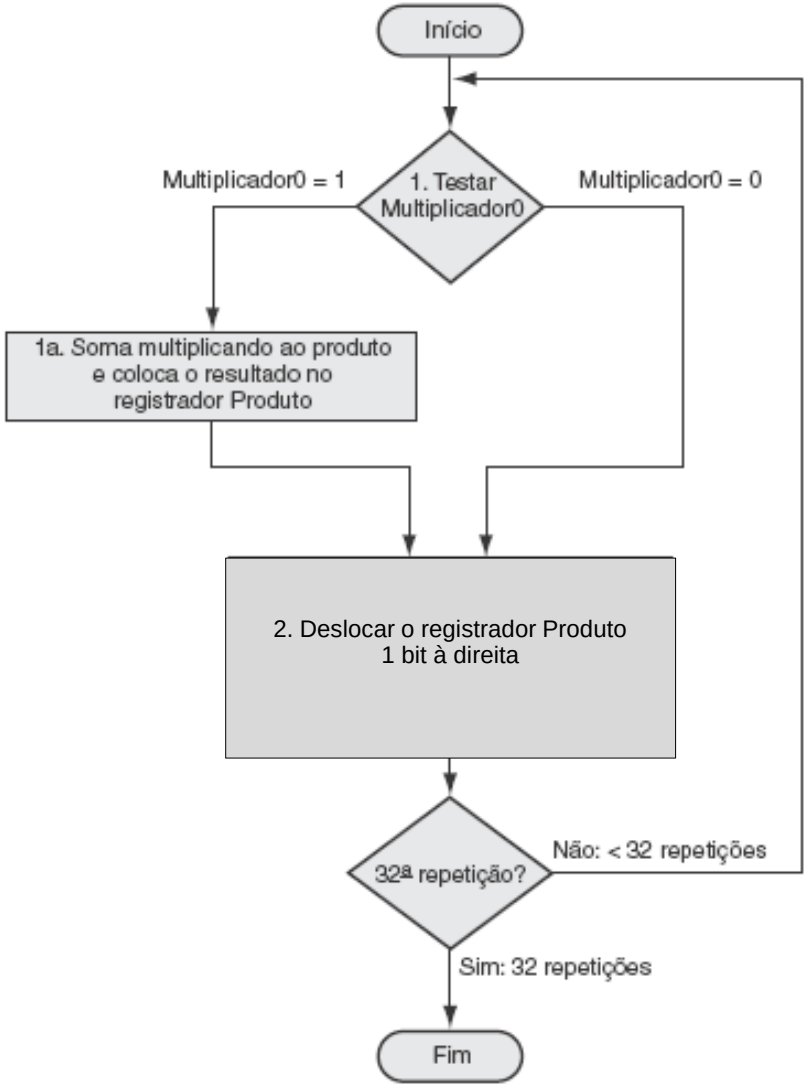
- **Hardware de multiplicação (versão refinada*)**
  - Caminho de dados



* 3ª versão da 2ª edição do livro texto

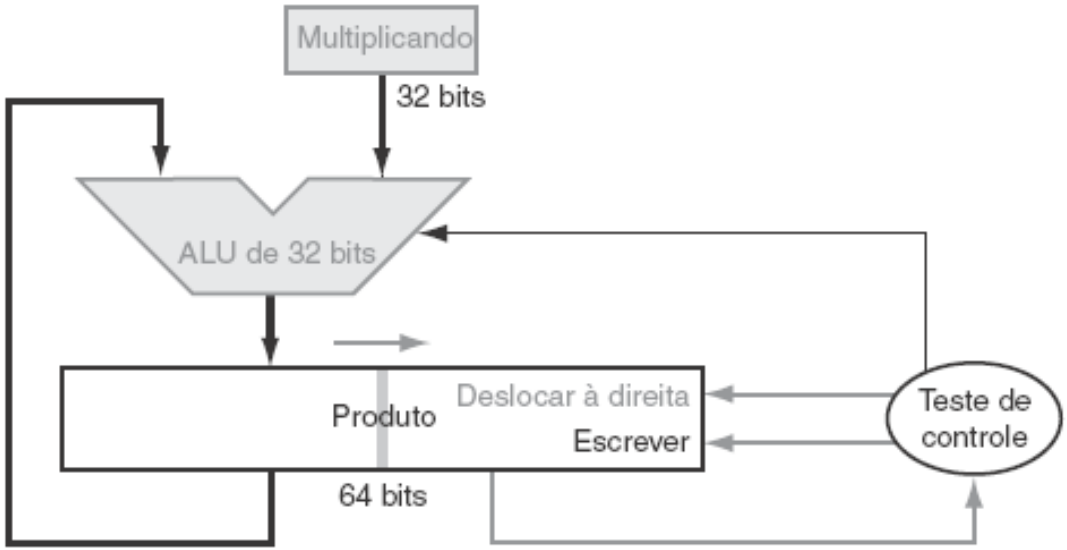
3 Multiplicação

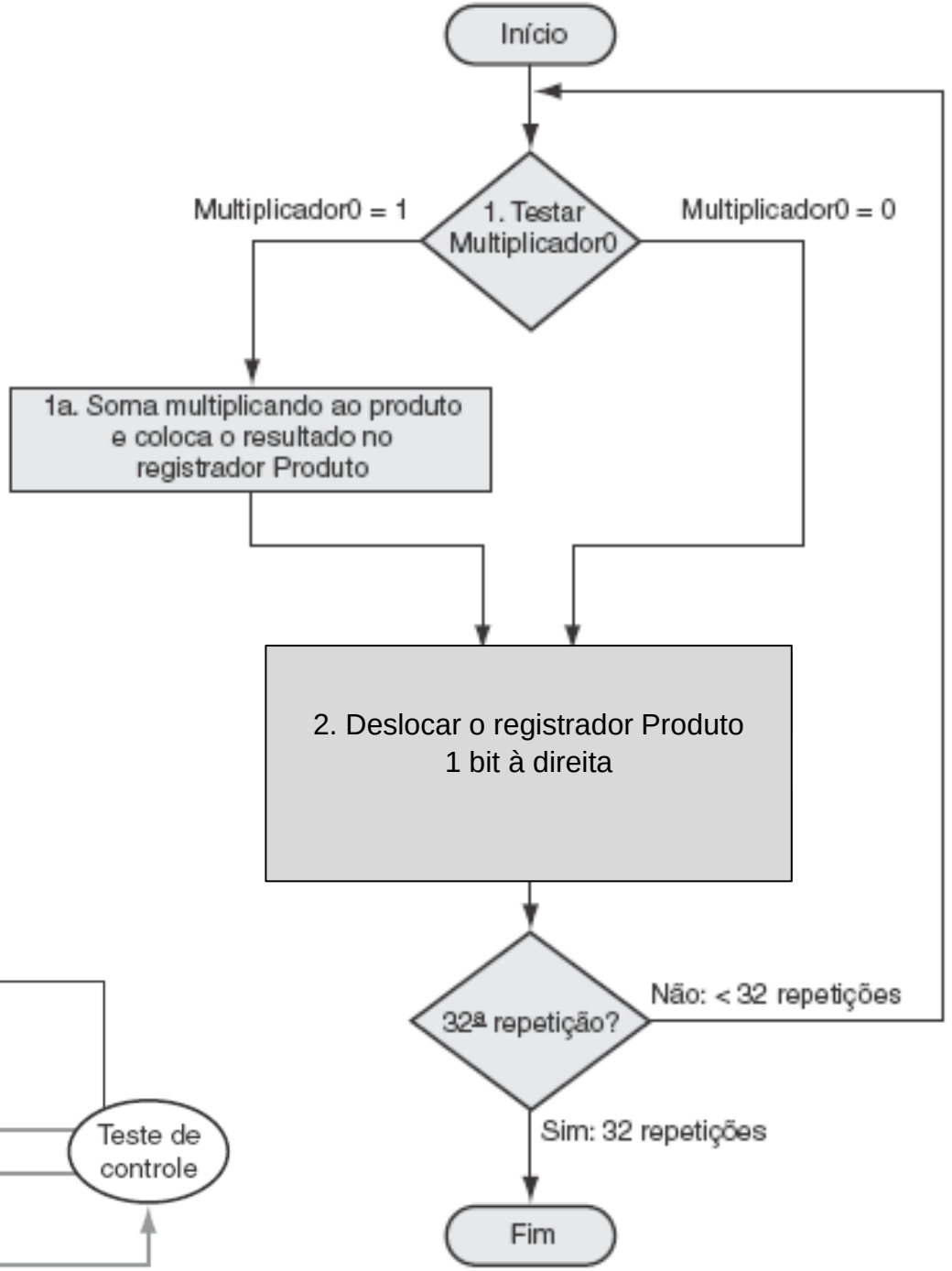
- **Hardware de multiplicação (versão refinada)**
  - Controle



3 Multiplicação

- **Hardware de multiplicação**
  (versão refinada)





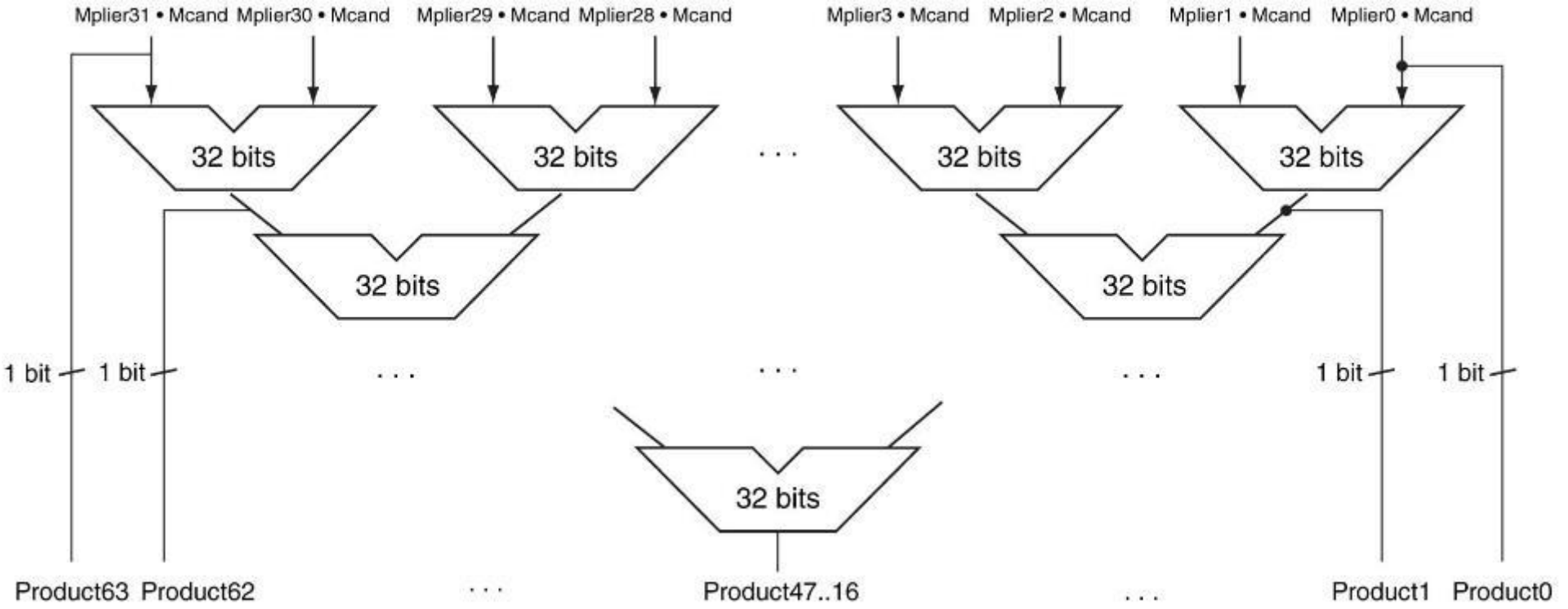
3 Multiplicação

- **Hardware de multiplicação (versão refinada)**
  - Exemplo

Iter.	Passo	Multiplicando	Produto
0	Valores iniciais	0010	0000 0011
1	1a: **1** => Produto = Produto + Multiplicando	0010	**0010** 0011
	2 : Deslocamento à direita do Produto	0010	**0001 0001**
2	1a: **1** => Produto = Produto + Multiplicando	0010	0011 0001
	2 : Deslocamento à direita do Produto	0010	**0001 1000**
3	1a: **0** => Nenhuma operação	0010	0001 1000
	2 : Deslocamento à direita do Produto	0010	**0000 1100**
4	1a: **0** => Nenhuma operação	0010	0000 1100
	2 : Deslocamento à direita do Produto	0010	**0000 0110**

3 Multiplicação

- **Multiplicador rápido (multiplicação paralela)**
  - Ao invés de usar um único somador de 32 bits 31 vezes, esse hardware desenrola o loop para usar 31 somadores e depois os organiza para minimizar o atraso



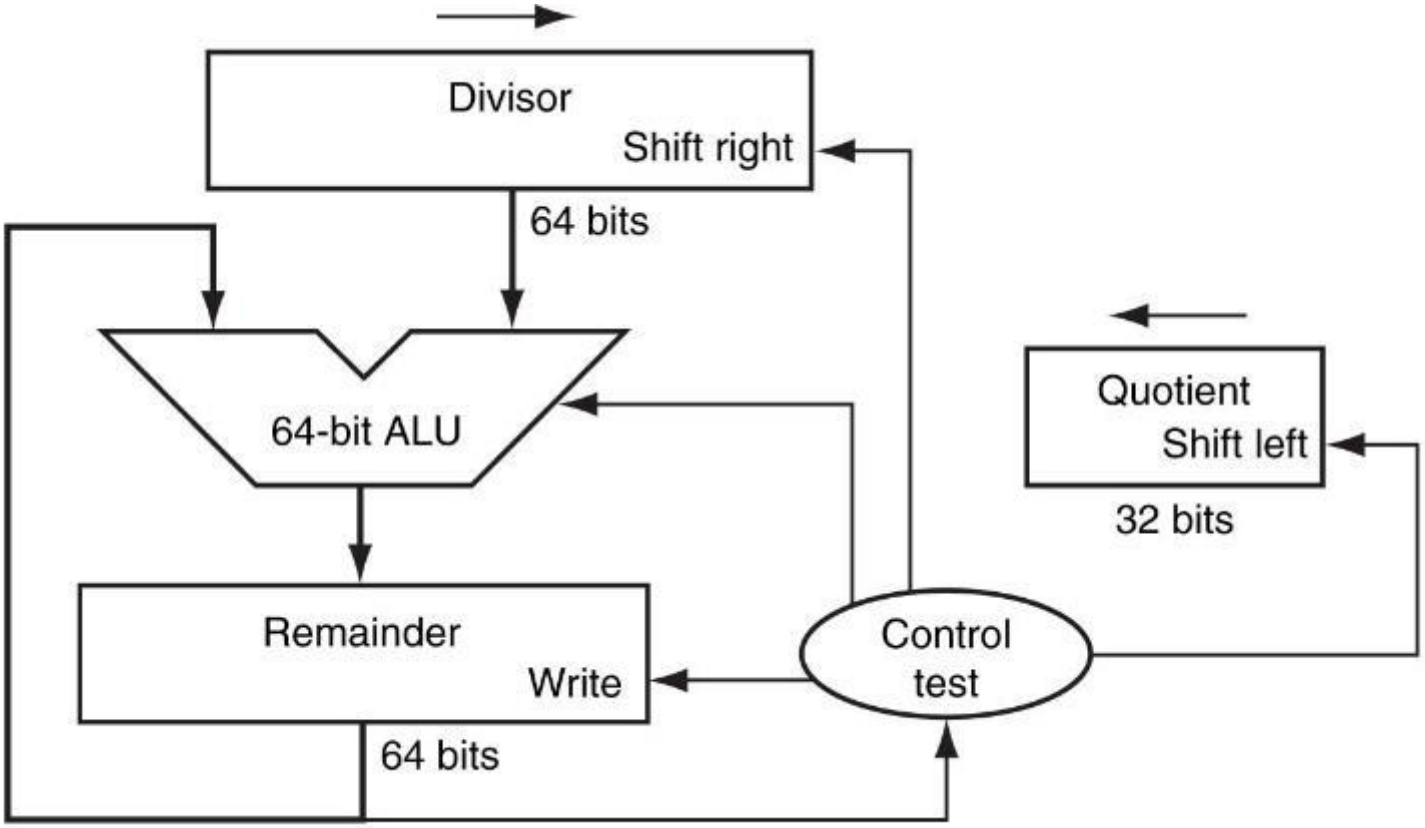
4 Divisão

Princípio da divisão

Dividendo	Divisor	
$1001010_{(10)}$	$1000_{(10)}$	
- 1000	1001	Quociente
1010		
- 1000		
10		Resto

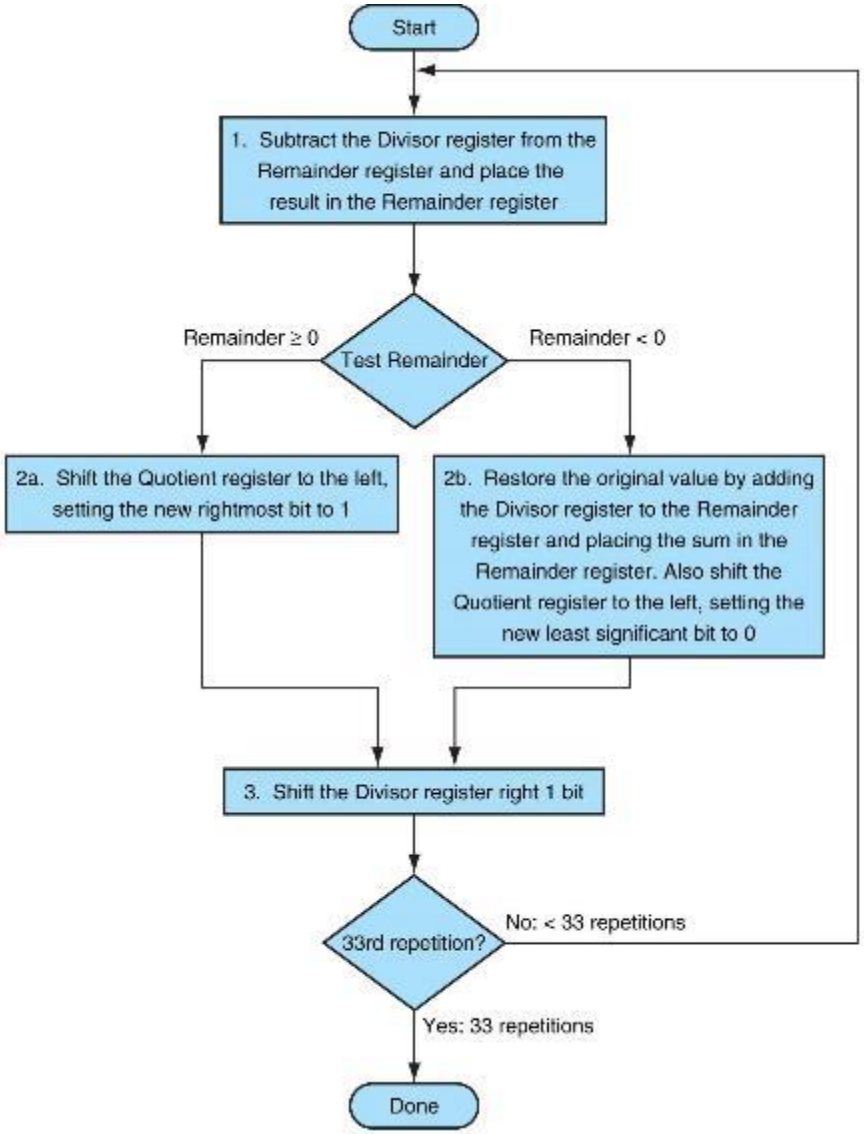
4 Divisão

- **Hardware de divisão (primeira versão)**
  - Caminho de dados



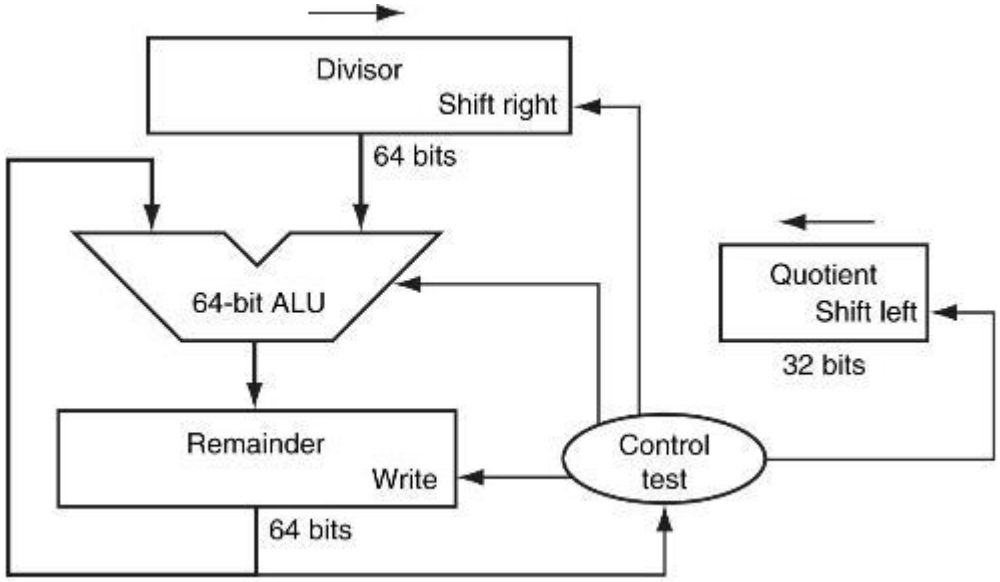
4 Divisão

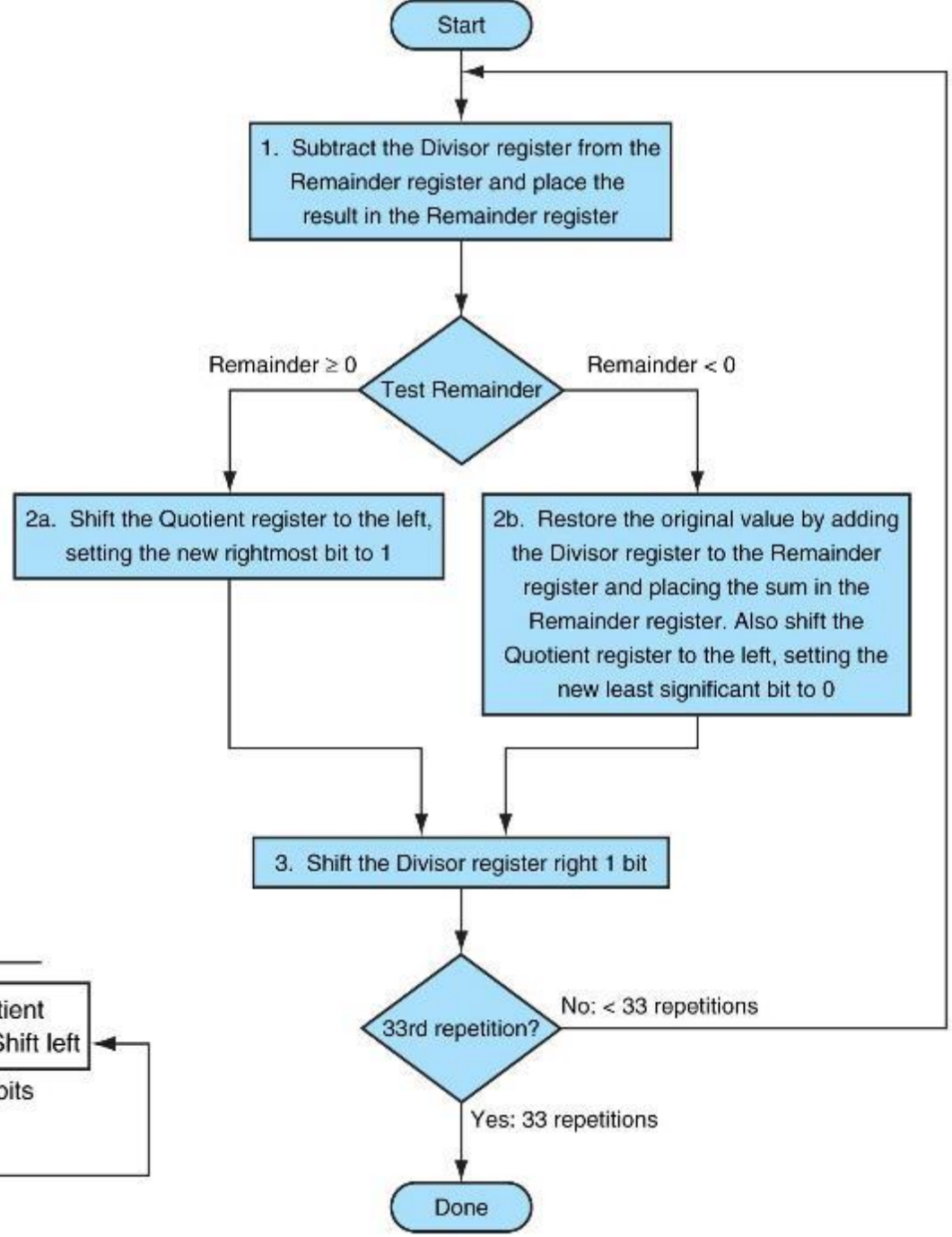
- **Hardware de divisão (primeira versão)**
  - Controle



4 Divisão

- **Hardware de divisão**





4 Divisão

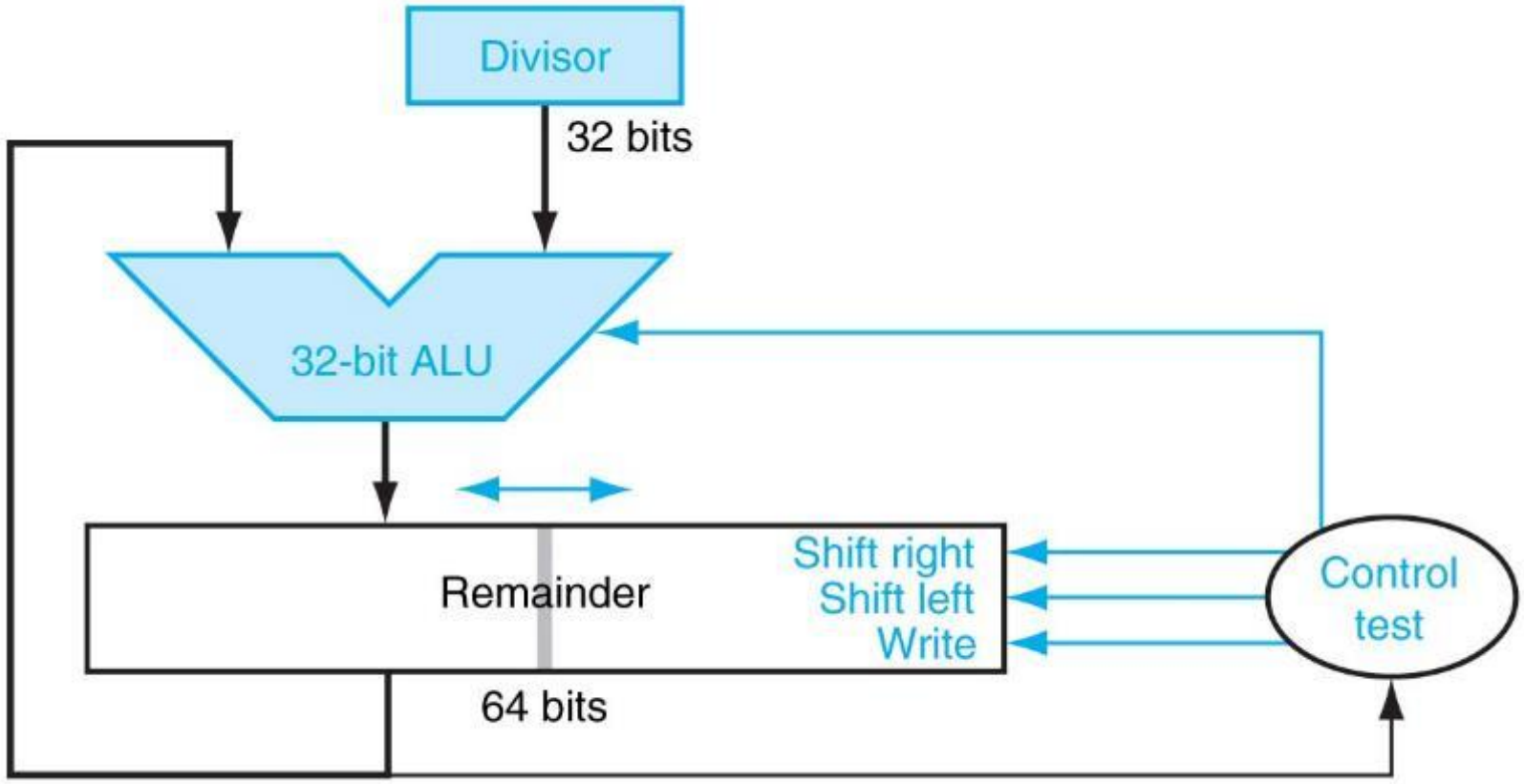
Hardware de divisão (primeira versão)

- Exemplo

Iteration	Step	Quotient	Divisor	Remainder
0	Initial values	0000	0010 0000	0000 0111
1	1: Rem = Rem – Div	0000	0010 0000	1110 0111
	2b: Rem < 0 ⟹ +Div, sll Q, Q0 = 0	0000	0010 0000	0000 0111
	3: Shift Div right	0000	0001 0000	0000 0111
2	1: Rem = Rem – Div	0000	0001 0000	1111 0111
	2b: Rem < 0 ⟹ +Div, sll Q, Q0 = 0	0000	0001 0000	0000 0111
	3: Shift Div right	0000	0000 1000	0000 0111
3	1: Rem = Rem – Div	0000	0000 1000	1111 1111
	2b: Rem < 0 ⟹ +Div, sll Q, Q0 = 0	0000	0000 1000	0000 0111
	3: Shift Div right	0000	0000 0100	0000 0111
4	1: Rem = Rem – Div	0000	0000 0100	0000 0011
	2a: Rem ≥ 0 ⟹ sll Q, Q0 = 1	0001	0000 0100	0000 0011
	3: Shift Div right	0001	0000 0010	0000 0011
5	1: Rem = Rem – Div	0001	0000 0010	0000 0001
	2a: Rem ≥ 0 ⟹ sll Q, Q0 = 1	0011	0000 0010	0000 0001
	3: Shift Div right	0011	0000 0001	0000 0001

4 Divisão

- **Hardware de divisão (versão refinada)**
  - Caminho de dados



5 Instruções de multiplicação e de divisão no MIPS

- **Multiplicação no MIPS**
  - 2 instruções com e sem sinal (`mult` e `multu`)
  - 2 registradores de 32 bits (`Hi` e `Lo`) para armazenar os 64 bits do produto – o resultado é mantido no registrador `Lo`
  - 2 instruções para copiar o conteúdo dos registradores (`mflo` e `mfhi`)

- **Divisão no MIPS**
  - 2 instruções com e sem sinal (`div` e `divu`)
  - Registrador `Lo` recebe o quociente
  - Registrador `Hi` recebe o resto

5 Instruções de multiplicação e de divisão no MIPS

- **Overflow**
  - O MIPS não detecta o overflow
  - É de responsabilidade do software determinar se ocorreu overflow
  - Multiplicação sem sinal: `Hi = 0`
  - Multiplicação com sinal: `Hi = sinal do Lo`

- **Divisão por zero**
  - O software tem que analisar o divisor (=0)

5 Instruções de multiplicação e de divisão

Categoria	Instrução	Exemplo	Significado
Aritmética	Multiplicação	`mult $s2, $s3`	`Hi, Lo = $s2 x $s3`
	Multiplicação sem sinal	`multu $s2, $s3`	`Hi, Lo = $s2 x $s3`
	Divisão	`div $s2, $s3`	`Lo = $s2 / $s3 # quociente` `Hi = $s2 mod $s3 # resto`
	Divisão sem sinal	`divu $s2, $s3`	`Lo = $s2 / $s3 # quociente s/s` `Hi = $s2 mod $s3 # resto`
	Mover do Hi	`mfhi $s1`	`$s1 = Hi`
	Mover do Lo	`mflo $s1`	`$s1 = Lo`