Universidade de Brasília
Departamento de Ciência da Computação

Aula 9
Aritmética Computacional
Aritmética Inteira

Principais Arquiteturas Aritméticas

- Pilha:
  - As operações são sempre realizadas com os argumentos na pilha, e o resultado é também armazenado na pilha. (calculadoras HP, Linguagem Forth, x87(ST0-ST7, 80 bits)
- Acumulador:
  - As operações são feitas sobre registradores (incluindo A) e o resultado armazenado em um registrador especial chamado Acumulador (A).(Z80, 8051)
- Registrador-Registrador:
  - As operações são feitas sobre registradores e o resultado é armazenado em qualquer registrador. (MIPS, ARM)
- Registrador-Memória:
  - As operações aritméticas buscam um dos argumentos na memória e armazenam o resultado em um registrador. (x86)

Unidade Lógica e Aritmética

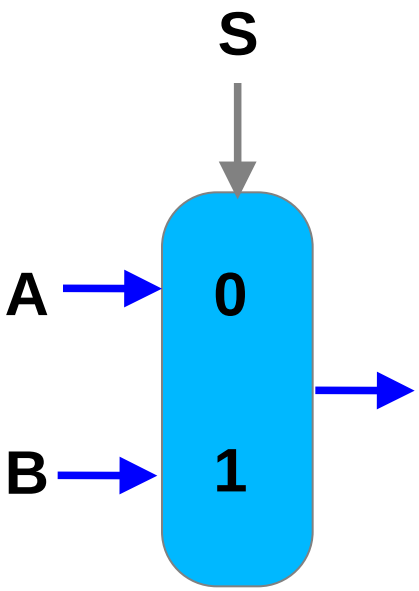
- O MIPS32 tem uma ULA de 32 bits

- Para ilustrar sua construção, construiremos uma ULA com as seguintes operações:

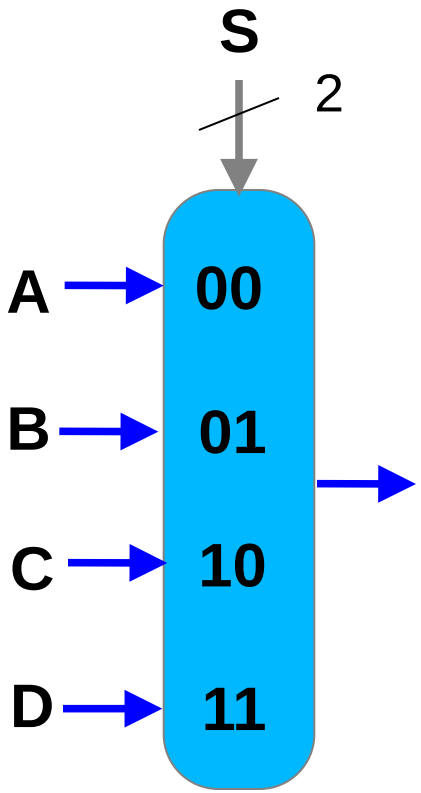
- Soma e subtração em complemento de 2
- Operações lógicas and, or e nor
- Instrução slt
- detecção de overflow
- detecção de igualdade

Elementos Básicos

- Multiplexador



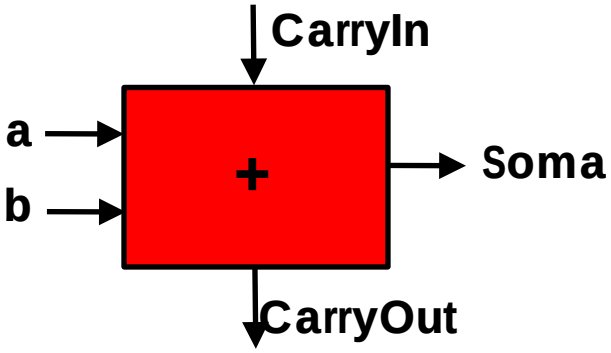
2x1



4x1

Elementos Básicos

- Somador

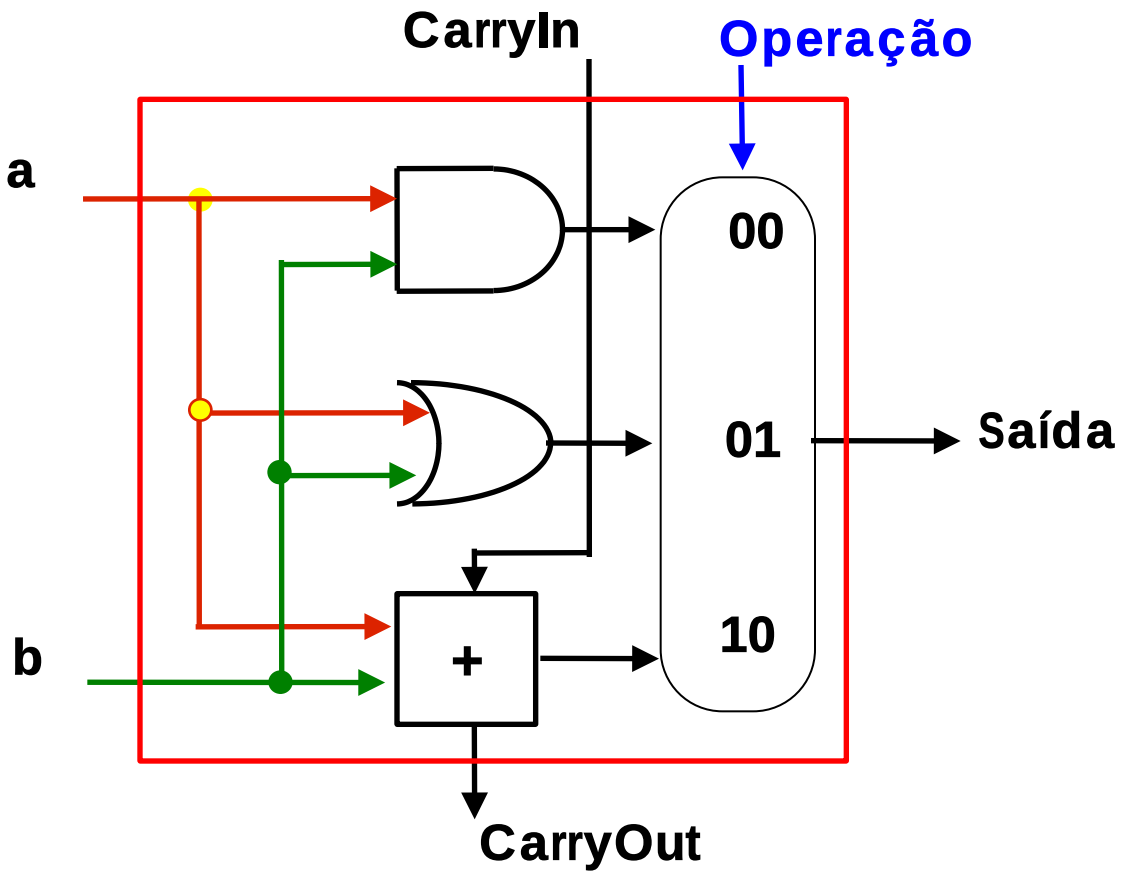


Entrada			Saídas	
a	b	CarryIn	CarryOut	Soma
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

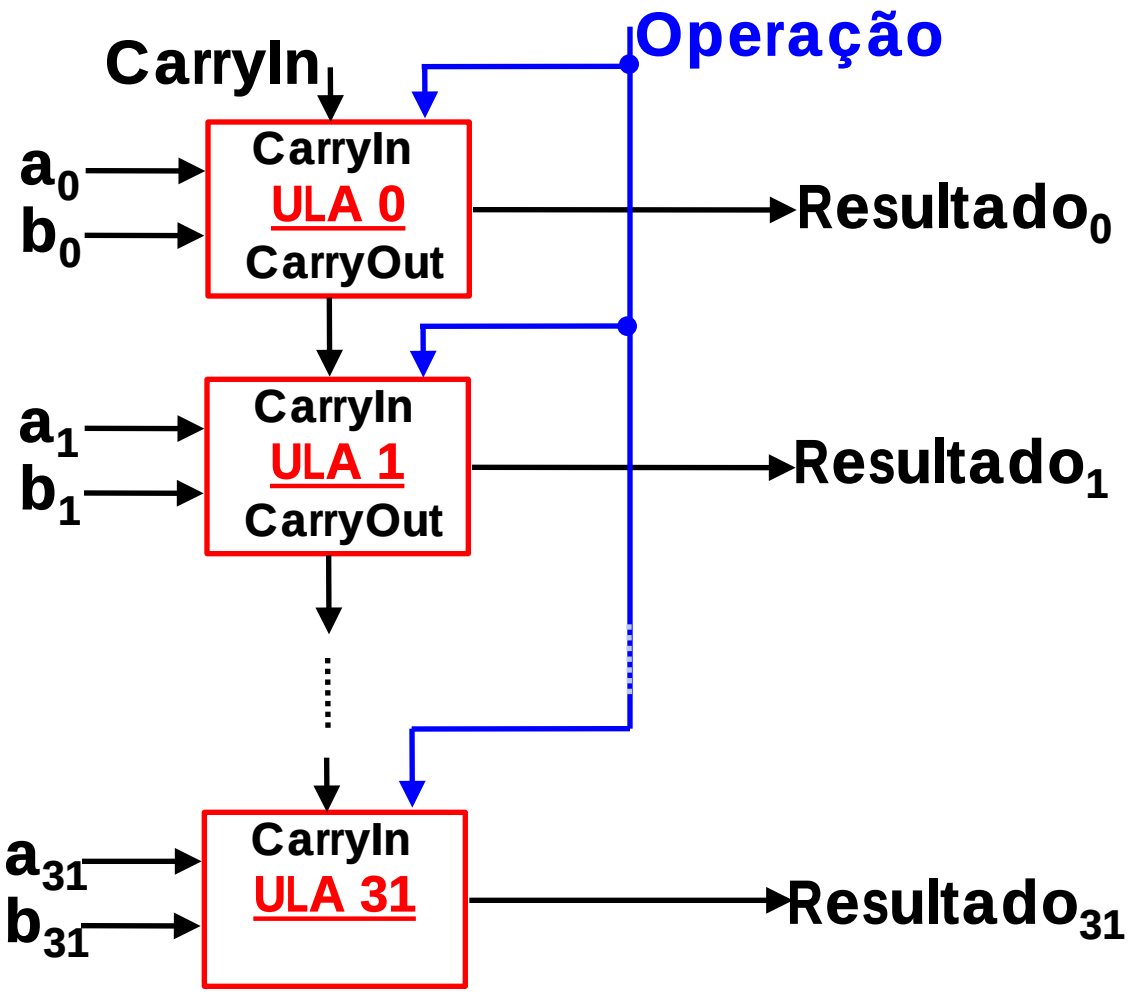
Ex.: Rever a implementação com portas lógicas

ULA de 1-bit

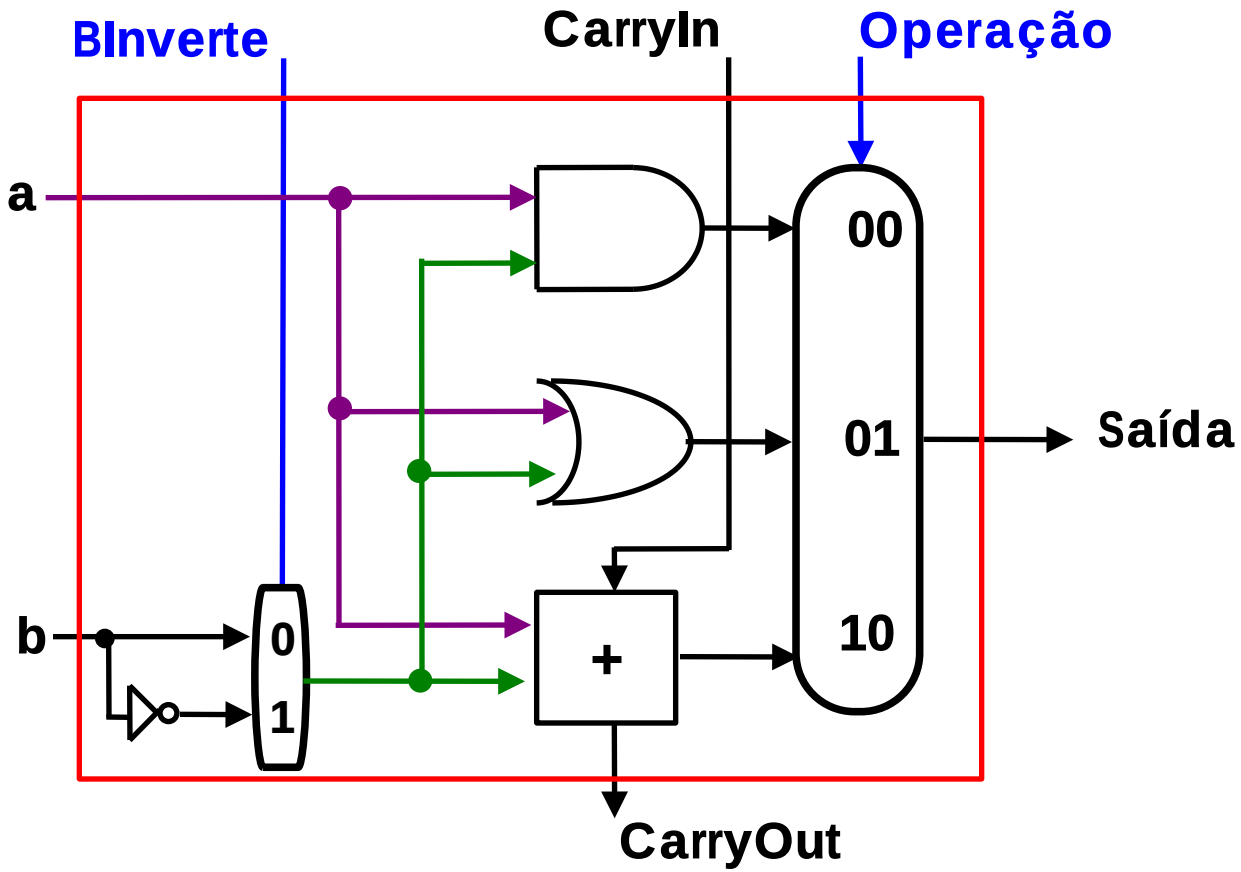
Operações de soma, and, or :



ULA de 32 bits



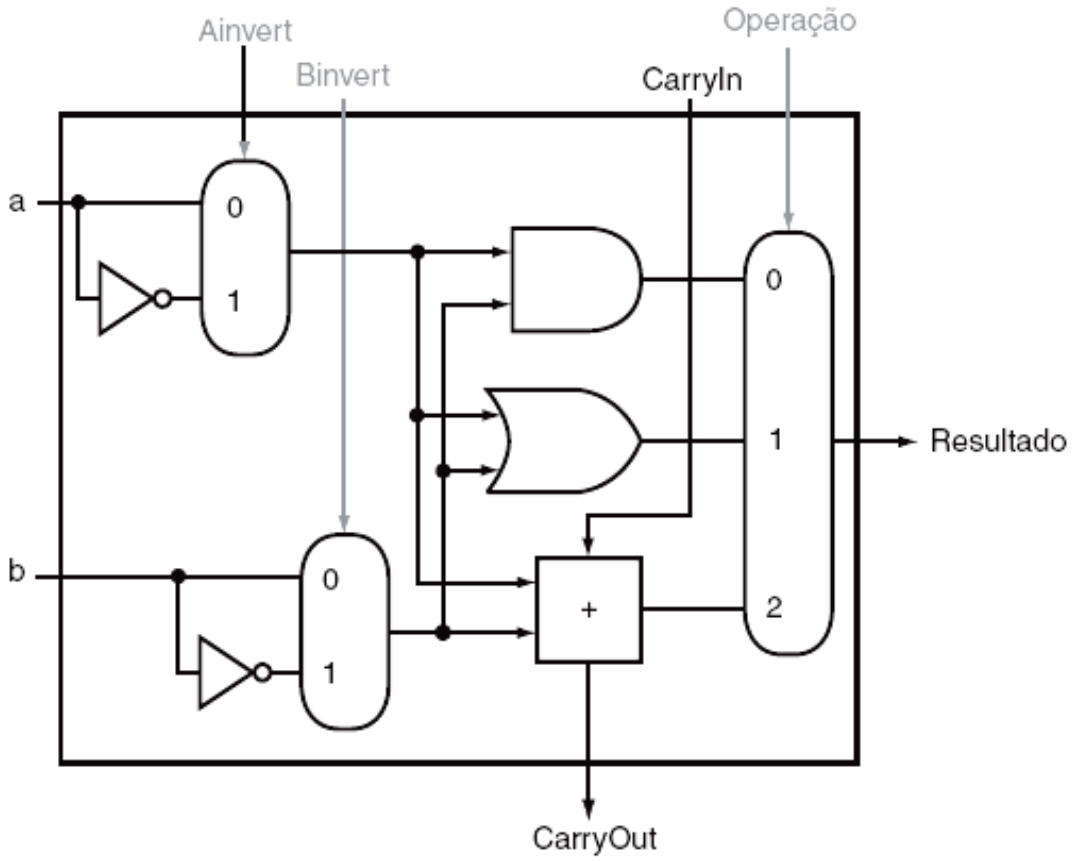
Incluindo Subtração:



Obs.: Em Complemento de 2: $a - b = a + (\overline{b} + 1)$

Operação nor:

Pelo Teorema de DeMorgan: $\overline{a+b} = \overline{a}.\overline{b}$



Instrução slt

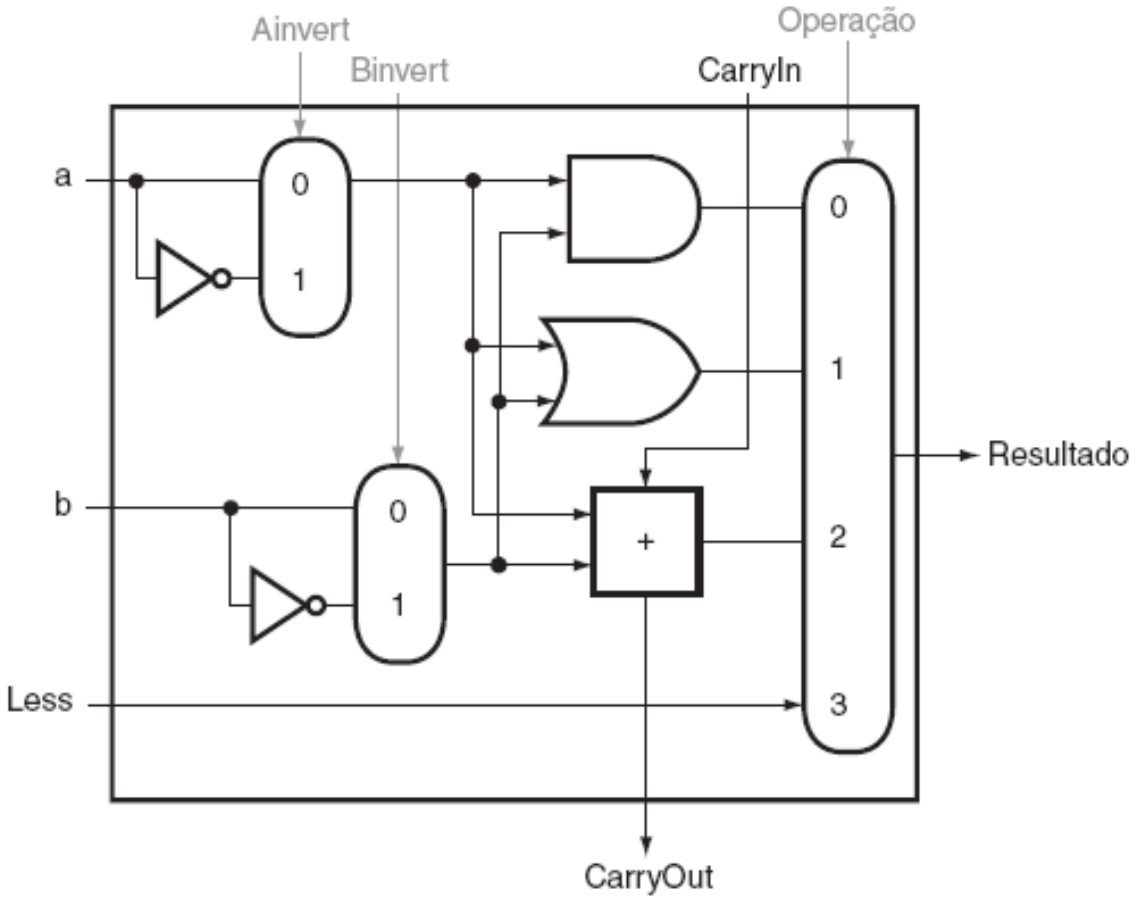
- A instrução slt gera saída 1 se rs < rt, e 0 caso contrário.

  Assim, todos os bits, com exceção do bit menos significativo, são fixados em zero.

- O bit menos significativo depende do resultado da comparação.

- Inclui-se a entrada Less

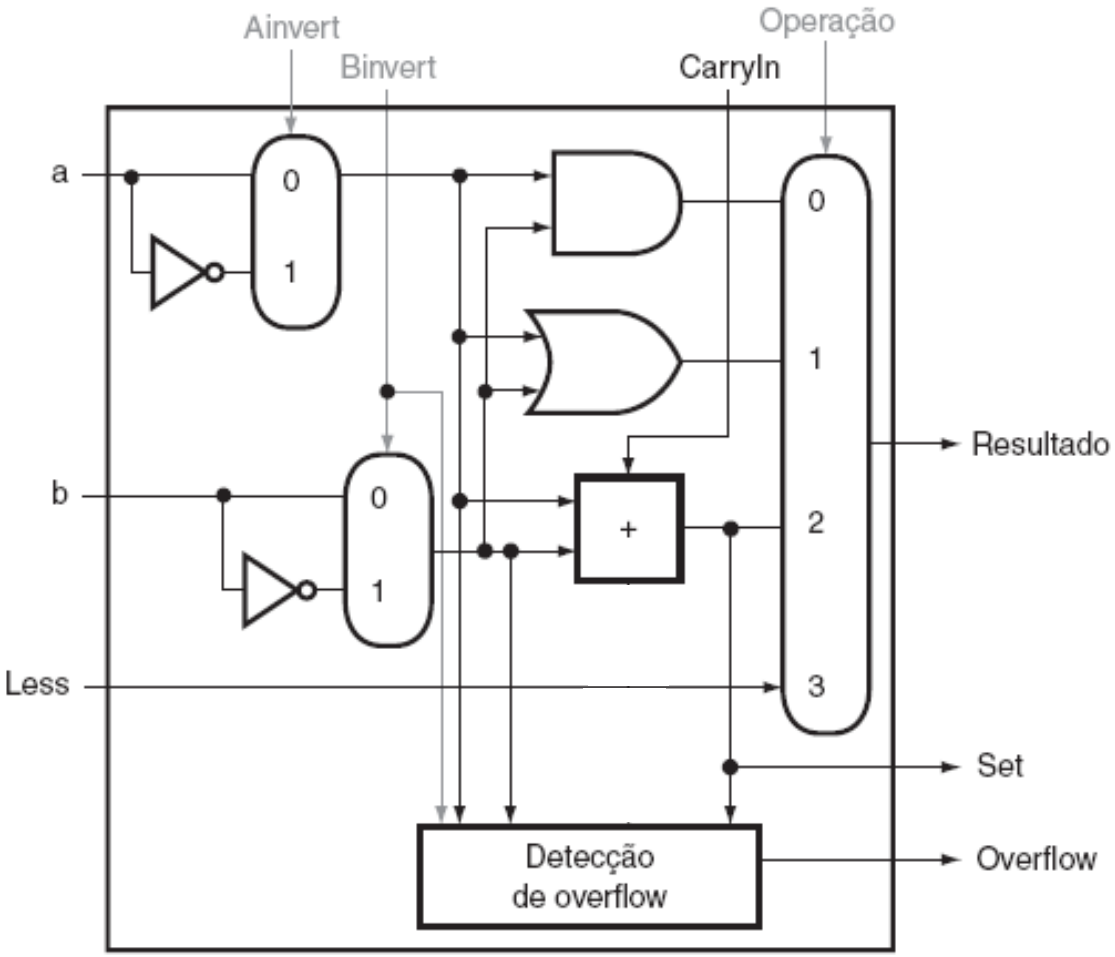
ULA de1-bit com slt (entrada Less)



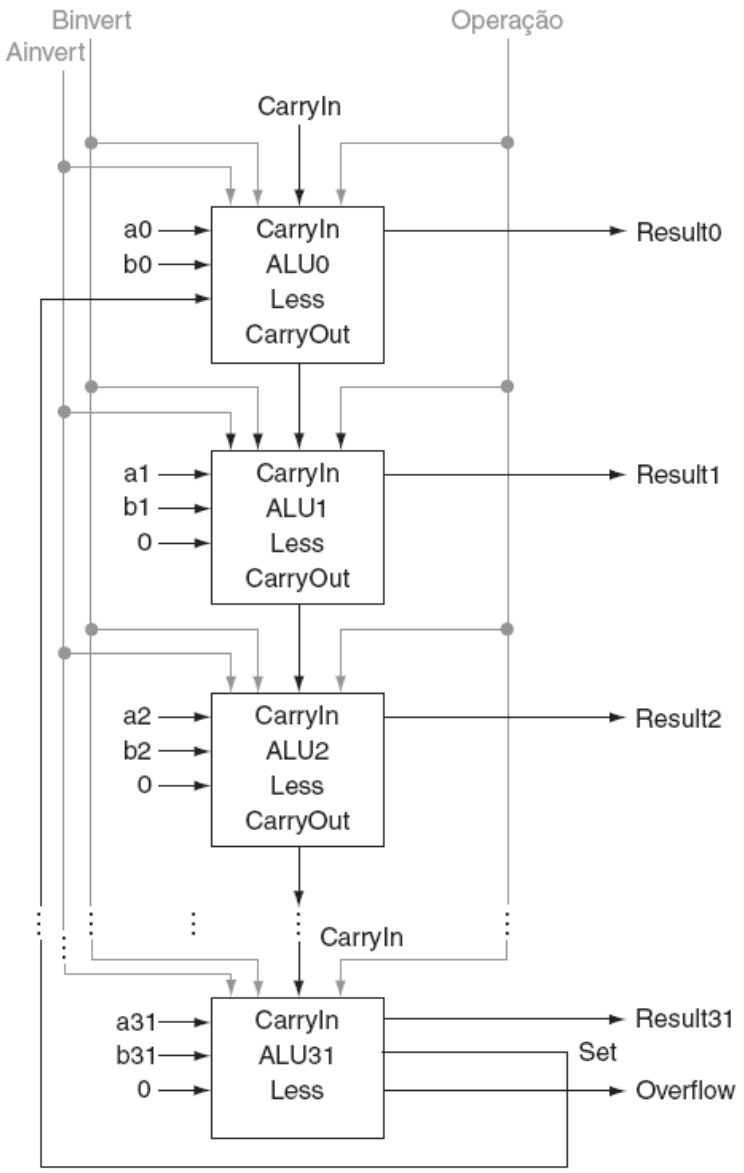
Calculando slt

- O bit menos significativo da ULA deve ser 1 se rs < rt
- Calcula-se este valor subtraindo rs de rt e tomando-se o bit mais significativo (“Sinal”):
  - se rs – rt < 0, rs <rt
- Utiliza-se o próprio subtrator da ULA para obter este valor, modificando-se o último estágio

Último estágio da ULA, com Set e detector Overfow



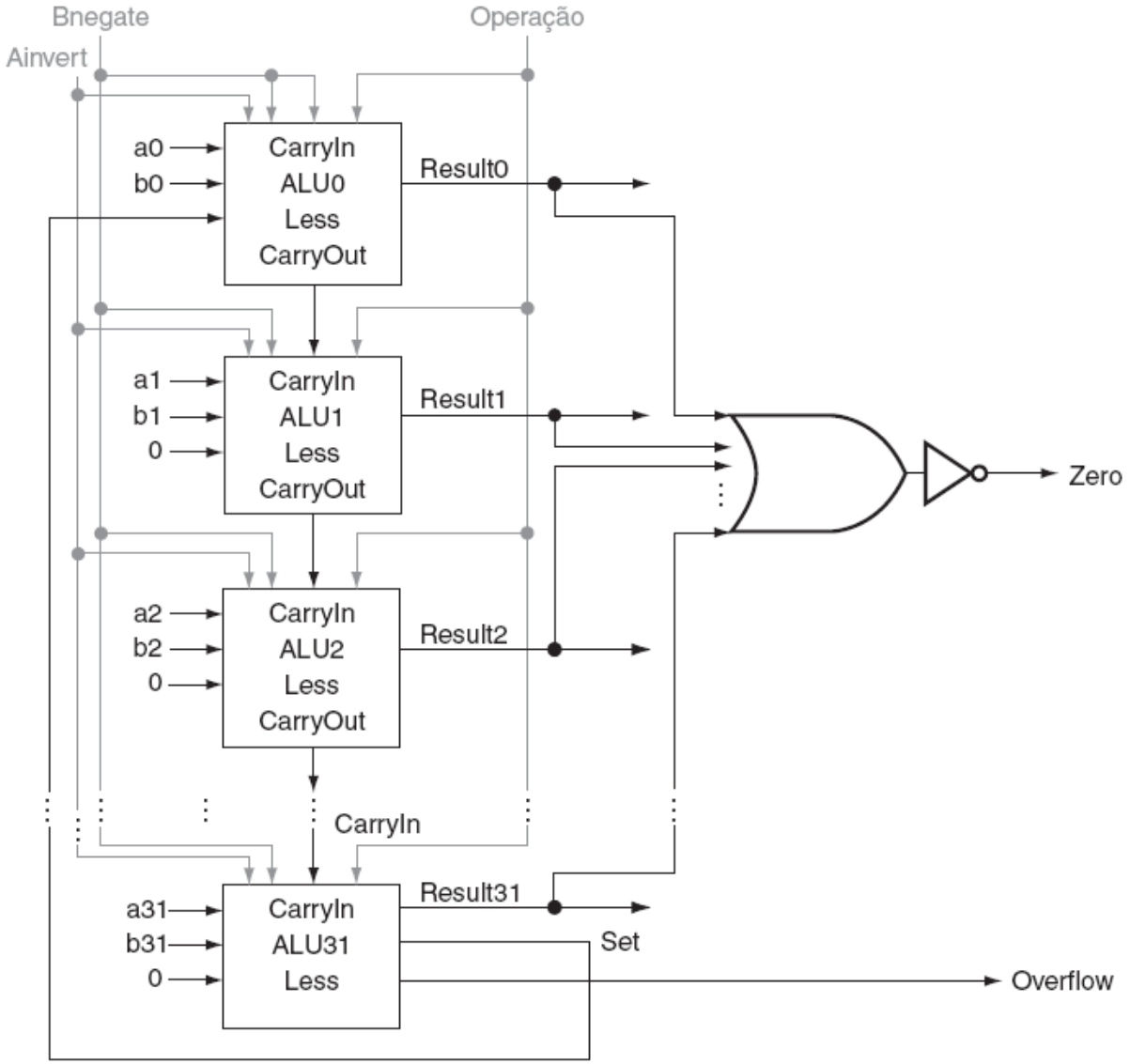
ULA de 32 bits



Testando igualdade

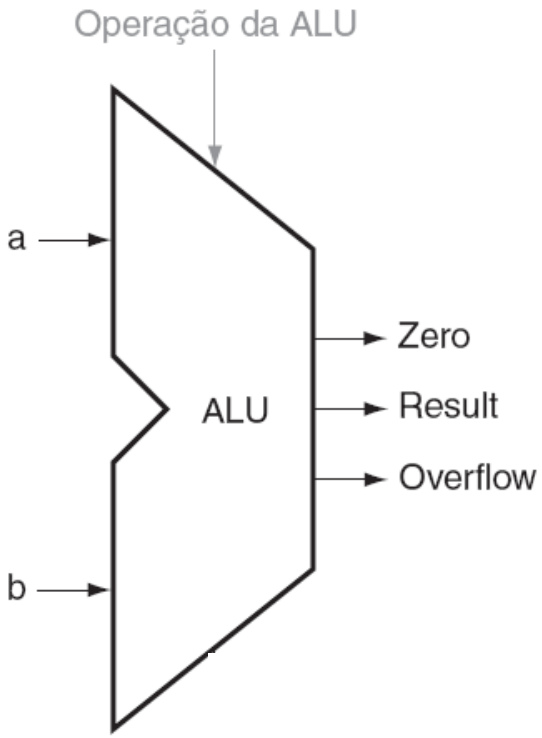
- As instruções bne e beq testam se dois valores são iguais ou não
- Para testar igualdade subtrai-se os dois operandos e testa-se se o resultado é zero:
  - A = B se A – B = 0 = R
  - Teste: operação NOR entre os bits do resultado
  - Igual = not(R_0 or R_1 or ... or R_{31})

ULA 32 bits com Comparação



Linhas de Controle e Simbologia

Linhas de controle da ALU	Função
0000	AND
0001	OR
0010	add
0110	subtract
0111	set on less than
1100	NOR



Obs.: Quais os tamanhos das linhas?

Implementação em Linguagem de Descrição de Hardware - Verilog

```
module MIPSALU (ALUctl, A, B, ALUOut, Zero);
   input [3:0] ALUctl;
   input [31:0] A,B;
   output reg [31:0] ALUOut;
   output Zero;
   assign Zero = (ALUOut==0); // Zero é verdadeiro se ALUOut é 0; vai para algum lugar
   always @(ALUctl, A, B) // reavalia se estes mudarem
      case (ALUctl)
         0: ALUOut <= A & B;
         1: ALUOut <= A | B;
         2: ALUOut <= A + B;
         6: ALUOut <= A – B;
         7: ALUOut <= A < B ? 1:0;
         12: ALUOut <= ~(A | B); // resultado é nor
         default: ALUOut <= 0; // default é 0, não deverá acontecer;
      endcase
endmodule
```

FIGURA B.4.3 Uma definição comportamental em Verilog de uma ALU MIPS. Isso poderia ser sintetizado por meio de uma biblioteca de módulos contendo operações aritméticas e lógicas básicas.

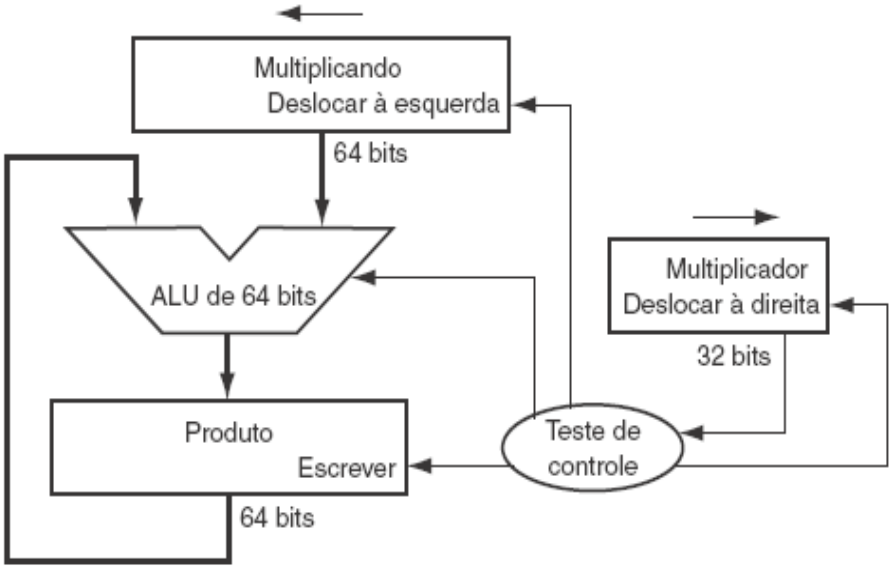
Multiplicação

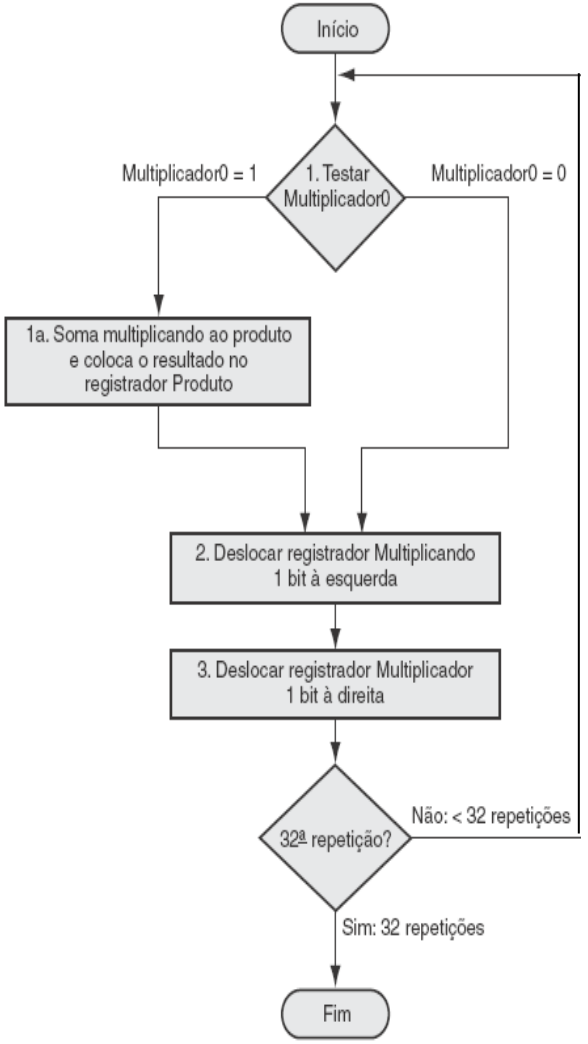
- Mais complexa do que a adição
  - realizada através de deslocamento e adição
- Requer mais tempo e mais área de chip
- Veremos apenas três versões
- Números negativos: converta para sem sinal, multiplique, defina sinal do resultado.

- existem técnicas melhores, que não serão tratadas aqui (Algoritmo de Booth)

Multiplicação: Versão Sequencial

0010 (multiplicando)
x 1011 (multiplicador)

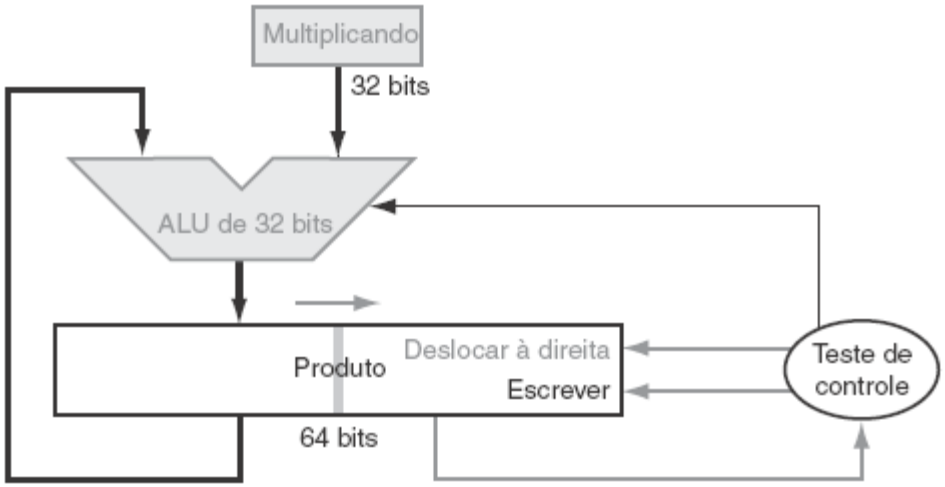


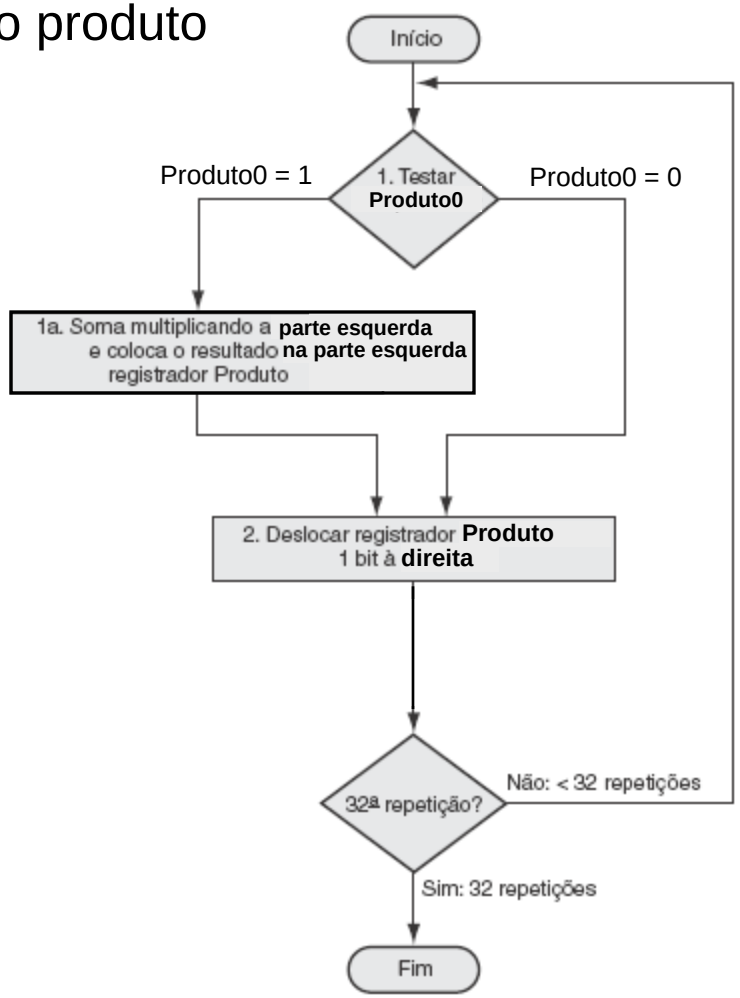


Versão Refinada

O multiplicador inicia na metade direita do produto

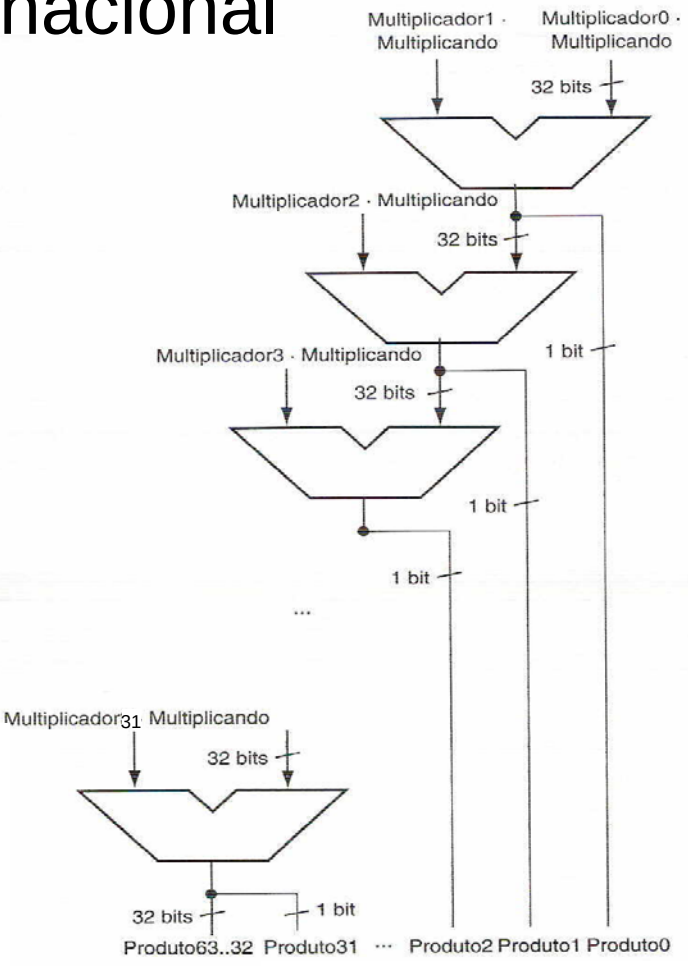
0010 (multiplicando)
x 1011 (multiplicador)





Multiplicação: Versão Combinacional

-Maior área em chip



-Versão com menor atraso

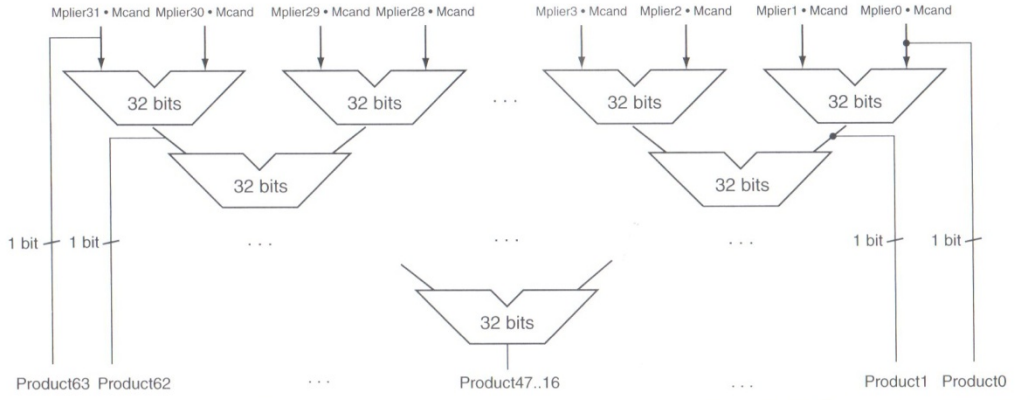


FIGURE 3.8 Fast multiplication hardware. Rather than use a single 32-bit adder 31 times, this hardware "unrolls the loop" to use 31 adders and then organizes them to minimize delay.

Divisão

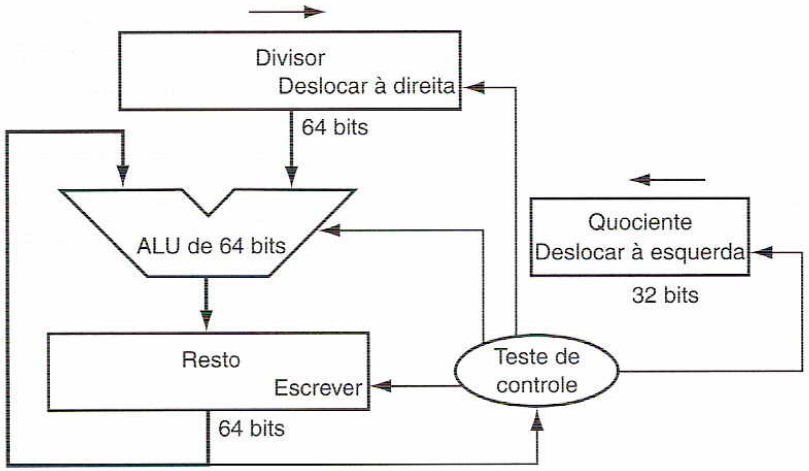
- Operação recíproca da multiplicação
- Ainda menos frequente e mais peculiar

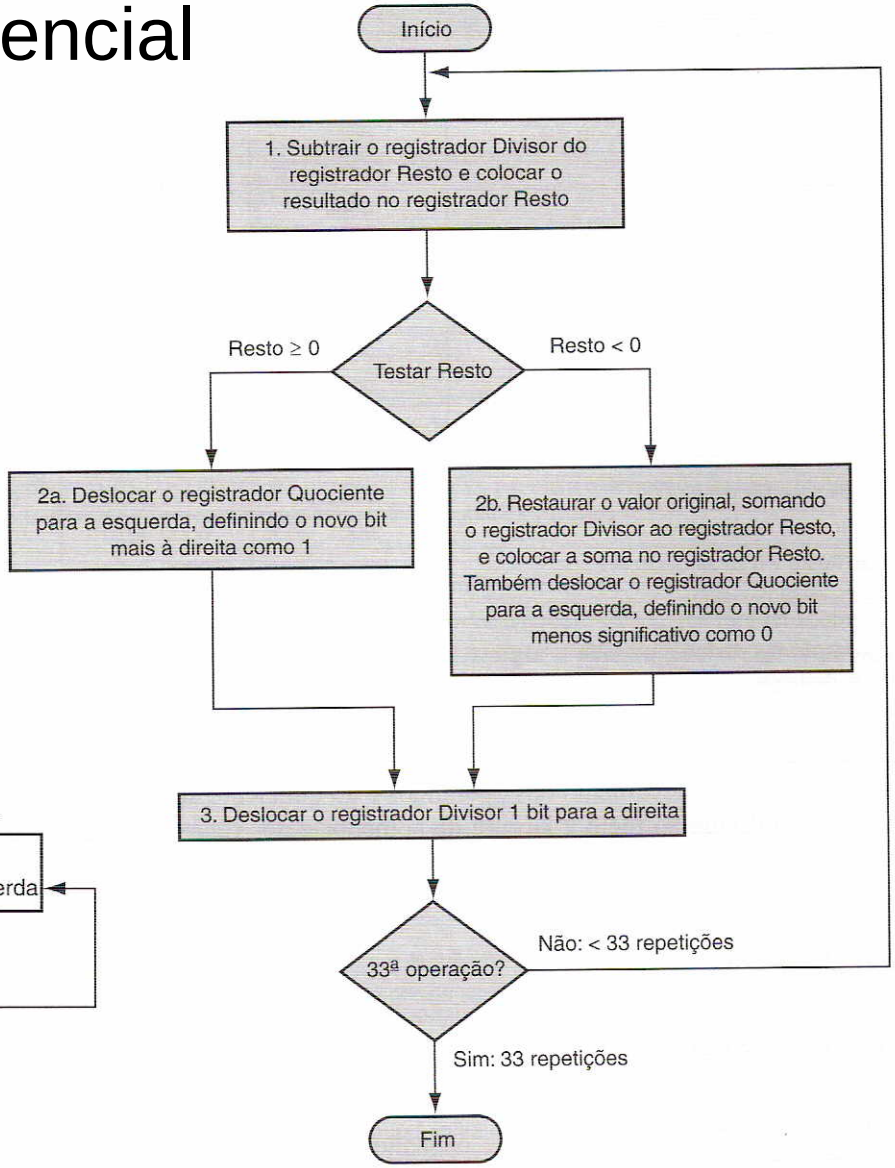
13270	13

110100	101

- Dividendo = Divisor x Quociente + Resto

Divisão: Versão Sequencial





Obs.: Existem versões otimizadas

Multiplicação e Divisão inteira em complemento de 2

Os algoritmos clássicos apenas funcionam com números sinal e magnitude.
Devemos operar com os módulos dos operandos e ajustar o sinal do(s) resultado(s).

Ajuste do sinal do resultado da multiplicação:
Basta verificar se os sinais do multiplicando e do multiplicador são iguais ou diferentes, definindo assim o sinal do produto.

Ajuste dos resultados da divisão:
Precisamos definir o sinal do quociente e do resto.

Dividendo = Quociente x Divisor + Resto

- $7 \div 2 \rightarrow 7 = 3 \times 2 + 1$
- $(-7) \div 2 \rightarrow -7 = (-3) \times 2 + (-1)$
- $7 \div (-2) \rightarrow 7 = (-3) \times (-2) + 1$
- $(-7) \div (-2) \rightarrow -7 = 3 \times (-2) + (-1)$

Regra: Quociente: Mesma regra da multiplicação .
Resto: Mesmo sinal do Dividendo.

Aritmética inteira no MIPS

- Adição:
  - add, addi, addu, addiu
    - Ex.: addu $t0,$t1,$t2 # $t0=$t1+$t2 sem sinal/overflow
- Subtração:
  - sub, subu
    - Ex.: sub $t0,$t1,$t2 # $t0=$t1-$t2 com sinal/overflow
- Multiplicação:
  - mult, multu
    - Ex.: mult $t0,$t1 # {Hi,Lo}=$t0×$t1 Registradores Hi e Lo
- Divisão:
  - div, divu
    - Ex.: div $t0,$t1 # Lo=floor($t0/$t1) Hi=$t0 % $t1 (Resto)
- Movimentação: Move From/To High/Low
  - mfhi, mflo, mthi, mtlo
    - Ex.: mfhi $t0 # $t0=Hi
      mtlo $t1 # Lo=$t1