Arquitetura de Computadores II

Aritmética Computacional

Professor Matheus Alcântara Souza

Aritmética Computacional

- Sinais e Complemento
- Adição e Subtração
- Operações Lógicas
- A Unidade Lógica Aritmética
- Multiplicação e Divisão
- Ponto Flutuante

Complemento

Complemento a 1:
O complemento a 1 de um número binário é obtido trocando os 0s por 1 e os 1s por 0. Em outras palavras, devemos trocar cada bit do número por seu complemento. Exemplo: 101101 => 010010

Complemento a 2:
O complemento a 2 de um número binário é obtido tomando-se o complemento a 1 do número, e adicionando-se uma unidade ao seu bit menos significativo. Exemplo: 101101 => 010010 + 1 => 010011

Complemento - possíveis representações de sinal

Sinal magnitude	Complemento a 1	Complemento a 2
000 = +0	000 = +0	000 = +0
001 = +1	001 = +1	001 = +1
010 = +2	010 = +2	010 = +2
011 = +3	011 = +3	011 = +3
100 = -0	100 = -3	100 = -4
101 = -1	101 = -2	101 = -3
110 = -2	110 = -1	110 = -2
111 = -3	111 = -0	111 = -1

Números com sinal em complemento a 2

- Se o número é positivo, a magnitude é representada na forma binária normal do número, com um bit de sinal de valor 0, colocado à esquerda do bit mais significativo da magnitude.

- Se o número é negativo, a magnitude é representada na forma de complemento a 2 do número, e um bit de sinal de valor 1 é colocado à esquerda do bit mais significativo da magnitude.

Números com sinal em complemento a 2

```
32 bit signed numbers:
                                         base 2   base 10
0000 0000 0000 0000 0000 0000 0000 0000 = 0
0000 0000 0000 0000 0000 0000 0000 0001 = + 1
0000 0000 0000 0000 0000 0000 0000 0010 = + 2
                   ...
0111 1111 1111 1111 1111 1111 1111 1110 = + 2.147.483.646
0111 1111 1111 1111 1111 1111 1111 1111 = + 2.147.483.647   (MAX_INT)
1000 0000 0000 0000 0000 0000 0000 0000 – 2.147.483.648   (MIN_INT)
1000 0000 0000 0000 0000 0000 0000 0001 = – 2.147.483.647
1000 0000 0000 0000 0000 0000 0000 0010 = – 2.147.483.646
                   ...
1111 1111 1111 1111 1111 1111 1111 1101 = – 3ten
1111 1111 1111 1111 1111 1111 1111 1110 = – 2ten
1111 1111 1111 1111 1111 1111 1111 1111 = – 1ten
```

Aritmética Computacional

- Sinais e Complemento
- Adição e Subtração
- Operações Lógicas
- A Unidade Lógica Aritmética
- Multiplicação e Divisão
- Ponto Flutuante

Adição e Subtração (4 bits)

- Soma normal

```
  0111        0111        0110
+ 0110      - 0110      - 0101
  ----        ----        ----
  1101        0001        0001
```

Adição e Subtração (4 bits)

- Soma normal

```
  0111          0111          0110
+ 0110        - 0110        - 0101
  1101          0001          0001
```

- Operação usando complemento a 2

Subtração usando adição de um número negativo

```
  0111 (7)          0111 (7)
- 0110 (6)        + 1010 (-6)
                    0001 (1)
```

Adição e Subtração (4 bits)

- Soma normal

```
  0111        0111        0110
+ 0110      - 0110      - 0101
  1101        0001        0001
```

- Operação usando complemento a 2

Subtração usando adição de um número negativo

```
  0111 (7)          0111 (7)
- 0110 (6)        + 1010 (-6)
                    0001 (1)
```

- Overflow

```
  0111
+ 0001
  1000
```

O resultado da operação não pode ser representado com o hardware disponível!

Detectando Overflow

- Não há overflow quando somamos um número positivo com um número negativo.
- Não há overflow quando os sinais são os mesmos em uma subtração.

- Quando um overflow ocorre:
  . Soma de dois positivos gerar um negativo.
  . Soma de dois negativos gerar um positivo.
  . Subtrair um negativo de um positivo e gerar um negativo.
  . Subtrair um positivo de um negativo e gerar um positivo.

- Considere as operações A + B, e A – B
  . Pode ocorrer um overflow se B = 0?
  . Pode ocorrer um overflow se A = 0?

Efeitos de um Overflow

- Quando ocorre uma exceção (interrupção):
    . Controle passa para uma sub-rotina (jump para endereço predefinido)
    . Endereço de interrupção é salvo para possível retomada.

- Nem sempre queremos detectar overflow
    . Instruções específicas para não detectar

Aritmética Computacional

- Sinais e Complemento
- Adição e Subtração
- Operações Lógicas
- A Unidade Lógica Aritmética
- Multiplicação e Divisão
- Ponto Flutuante

Operações lógicas

- Além de se trabalhar (operações) com palavras, também há necessidade de se trabalhar com bits individuais (isolados) dentro de uma palavra

- **Shifts:** deslocamento à esquerda ou à direita

  Move todos os bits da palavra, por exemplo:

```
                   0000 0000 0000 1001 (9)
shift left by 4 => 0000 0000 1001 0000 (144)
```

Operações lógicas

- **AND:** atua bit-a-bit, deixando 1 como resultado, somente se ambos os bits correspondentes dos operandos forem 1, como no exemplo:

```
     0000 1101 1100 0000
AND  0011 1100 0000 0000
     0000 1100 0000 0000
```

Operações lógicas

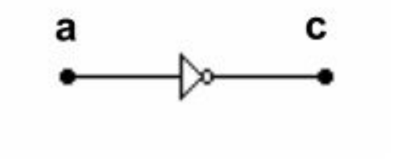
- **OR:** atua bit-a-bit, deixando 1 como resultado, se qualquer um dos bits correspondentes dos operandos for 1, como no exemplo:

```
     0000 1101 1100 0000
AND  0011 1100 0000 0000
     0011 1101 1100 0000
```

Blocos construtivos básicos

NOT (Inversora)

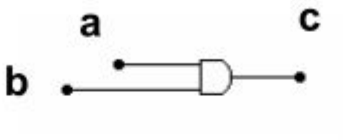
$$c = \overline{a}$$



Entrada	Saída
a	c
0	1
1	0

AND

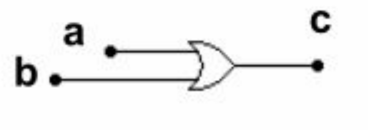
$$c = a.b$$



Entradas		Saída
a	b	c
0	0	0
0	1	0
1	0	0
1	1	1

OR

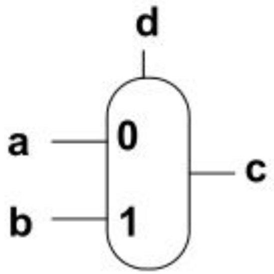
$$c = a + b$$



Entrada		Saída
a	b	c
0	0	0
0	1	1
1	0	1
1	1	1

Multiplexador

Se d==0, c=a senão c=b



Entrada	Saída
d	c
0	a
1	b

Aritmética Computacional

- Sinais e Complemento
- Adição e Subtração
- Operações Lógicas
- A Unidade Lógica Aritmética
- Multiplicação e Divisão
- Ponto Flutuante

Unidade Lógica Aritmética (ULA)

- A ULA é a responsável por executar operações lógicas e aritméticas

- Recebe dados de entrada (e.g., dos registradores)

- Realiza operação, e emite um resultado (e.g., para registradores também)

PROJETO DE ULA

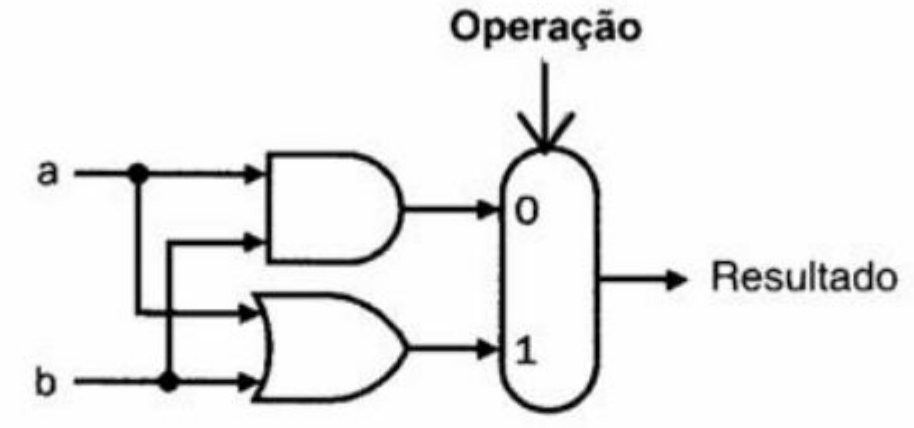
Uma ULA pode ser projetada para executar **qualquer operação**, no entanto estas operações podem ter certo grau de **complexidade**.

Quanto mais complexa a operação, mais cara é a ULA:

- ocupa mais espaço
- consome mais energia

ULA de 1 bit

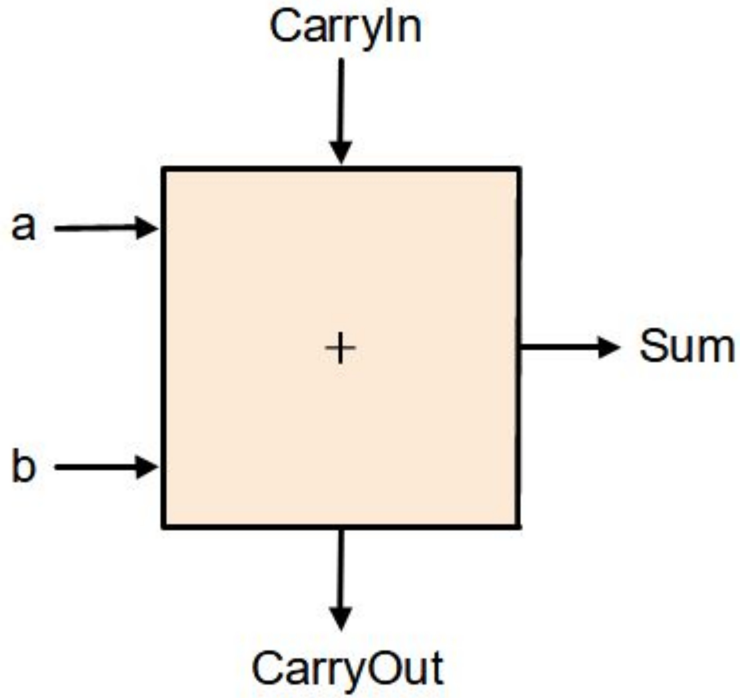
Se a operação for 0: AND
Se a operação for 1: OR



Somador de 1 bit

- Soma A + B + “vem 1”
- Gera resultado e “vai 1”

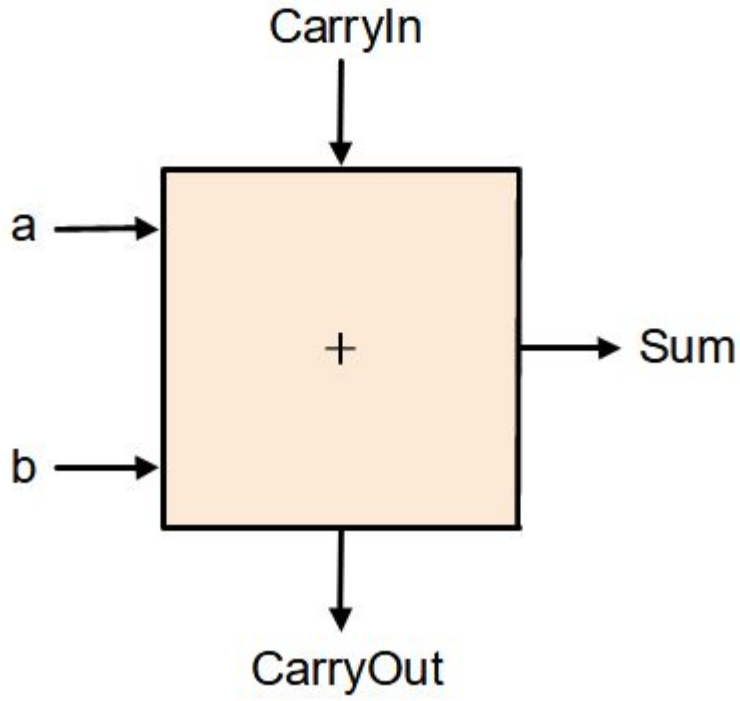
Entradas			Saídas		Comentários
A	B	Vem 1	Soma	Vai 1	
0	0	0	0	0	0+0+0 = 00
0	0	1	1	0	0+0+1 = 01
0	1	0	1	0	0+1+0 = 01
0	1	1	0	1	0+1+1 = 10
1	0	0	1	0	1+0+0 = 01
1	0	1	0	1	1+0+1 = 10
1	1	0	0	1	1+1+0 = 10
1	1	1	1	1	1+1+1 = 11



Unidade Lógica Aritmética (ULA)

Adição

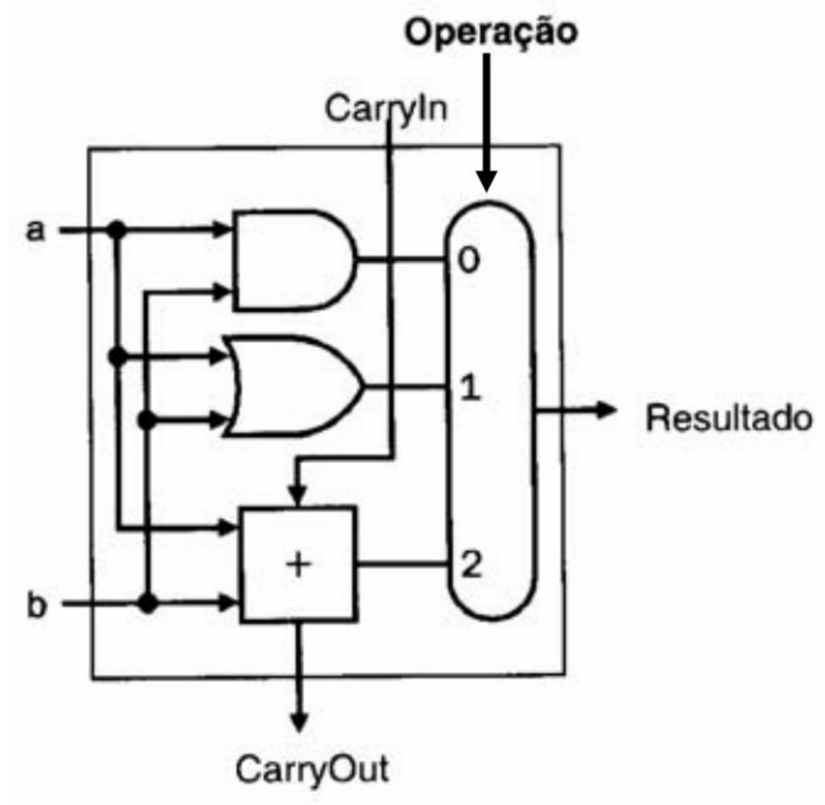
- Para incluir a adição na ULA, devemos incluir o circuto SOMADOR
- A e B: dois bits de operandos
- C: o CarryIn
  (“vai-um” da operação anterior)



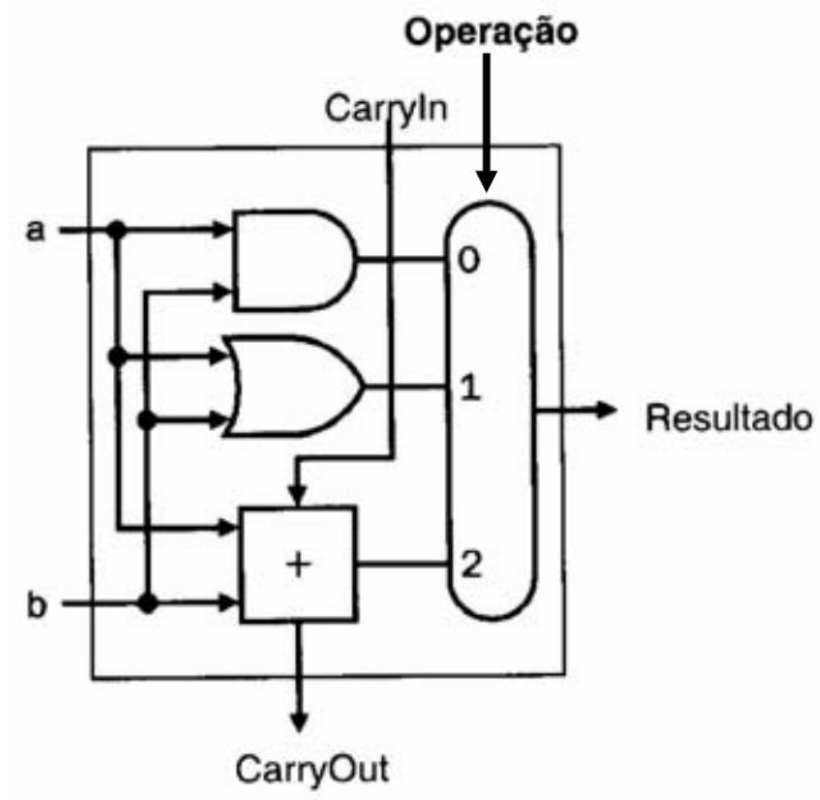
$$Soma = A.B.C + A.\overline{B}.\overline{C} + \overline{A}.B.\overline{C} + \overline{A}.\overline{B}.C$$

$$CarryOut = A.B + A.C + B.C$$

ULA de 1 bit com somador



ULA de 1 bit

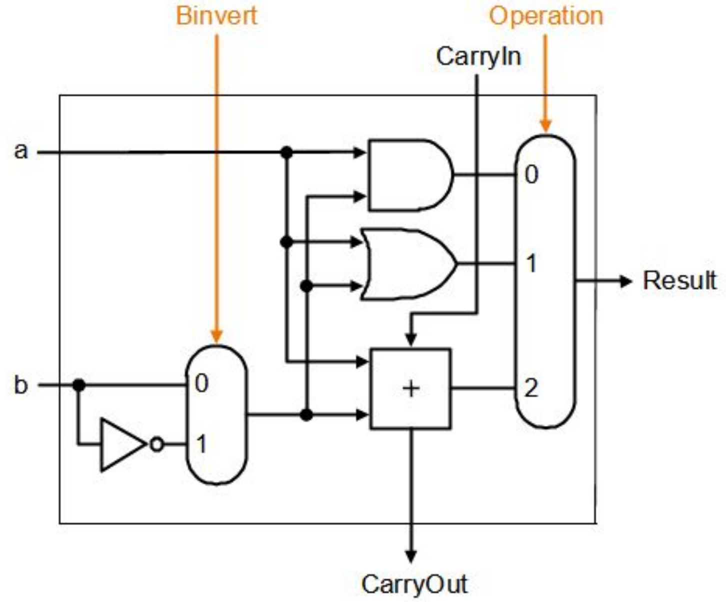


Como alterar a ULA para que ela gere o valor 0?

Dica: a maneira mais fácil é expandir o multiplexador controlado por "Operação"

Unidade Lógica Aritmética (ULA)

ULA de 1 bit

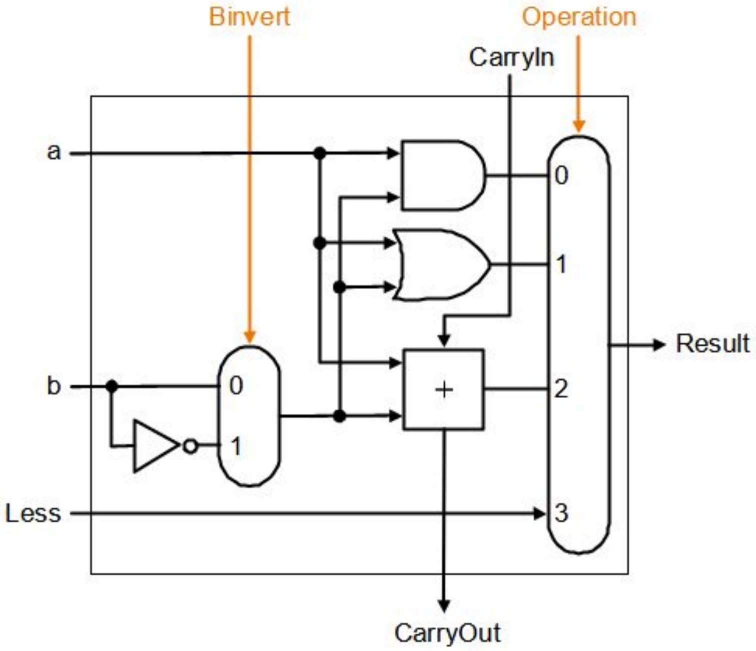


Como ocorre uma subtração?

$$a - b = a + (\bar{b} + 1)$$

Unidade Lógica Aritmética (ULA)

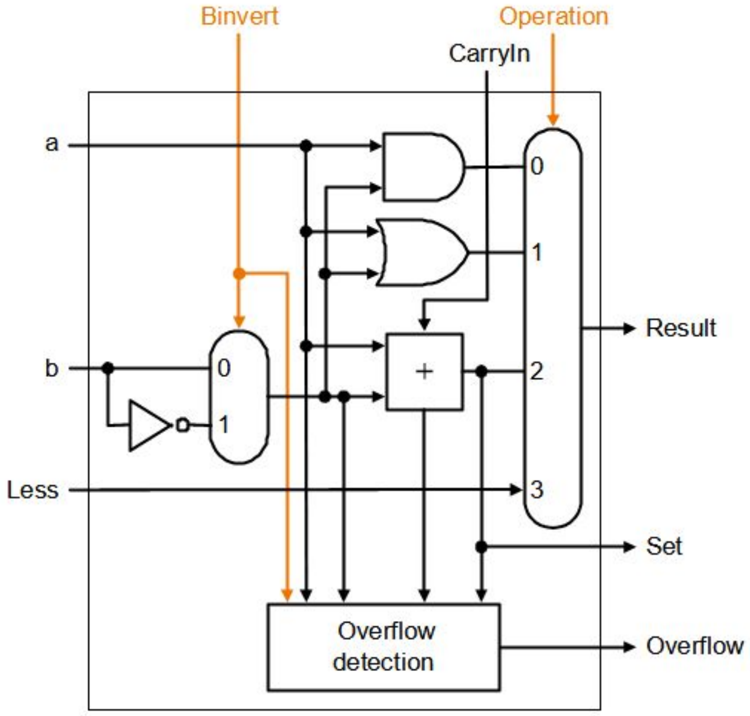
ULA de 1 bit



Set on less than

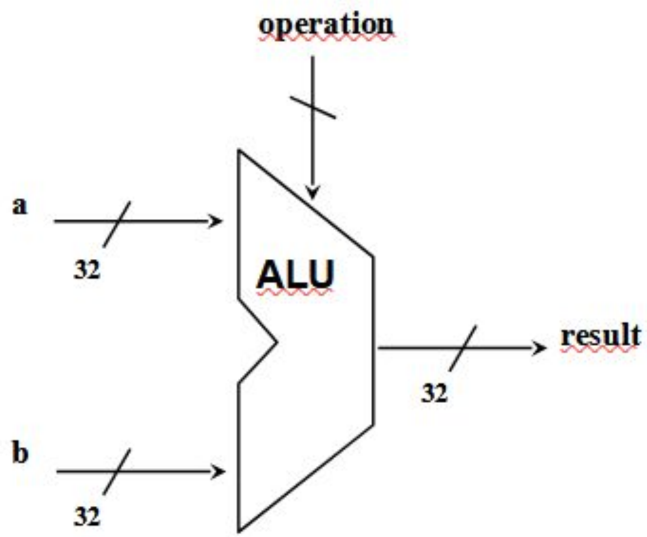
Unidade Lógica Aritmética (ULA)

ULA de 1 bit



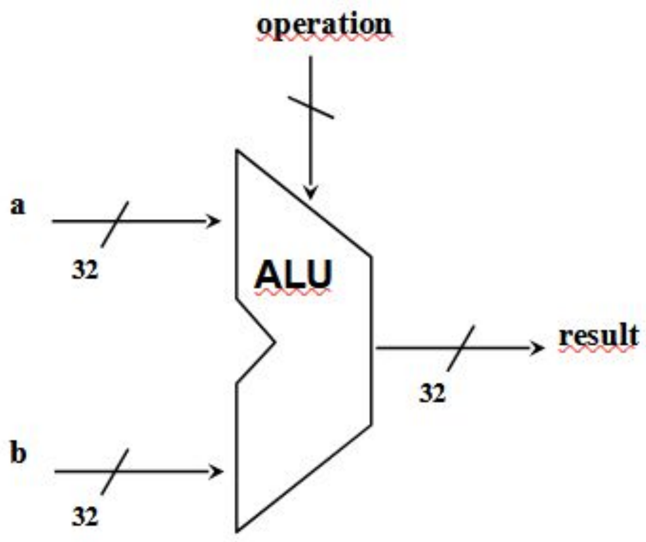
ULA com detecção de overflow

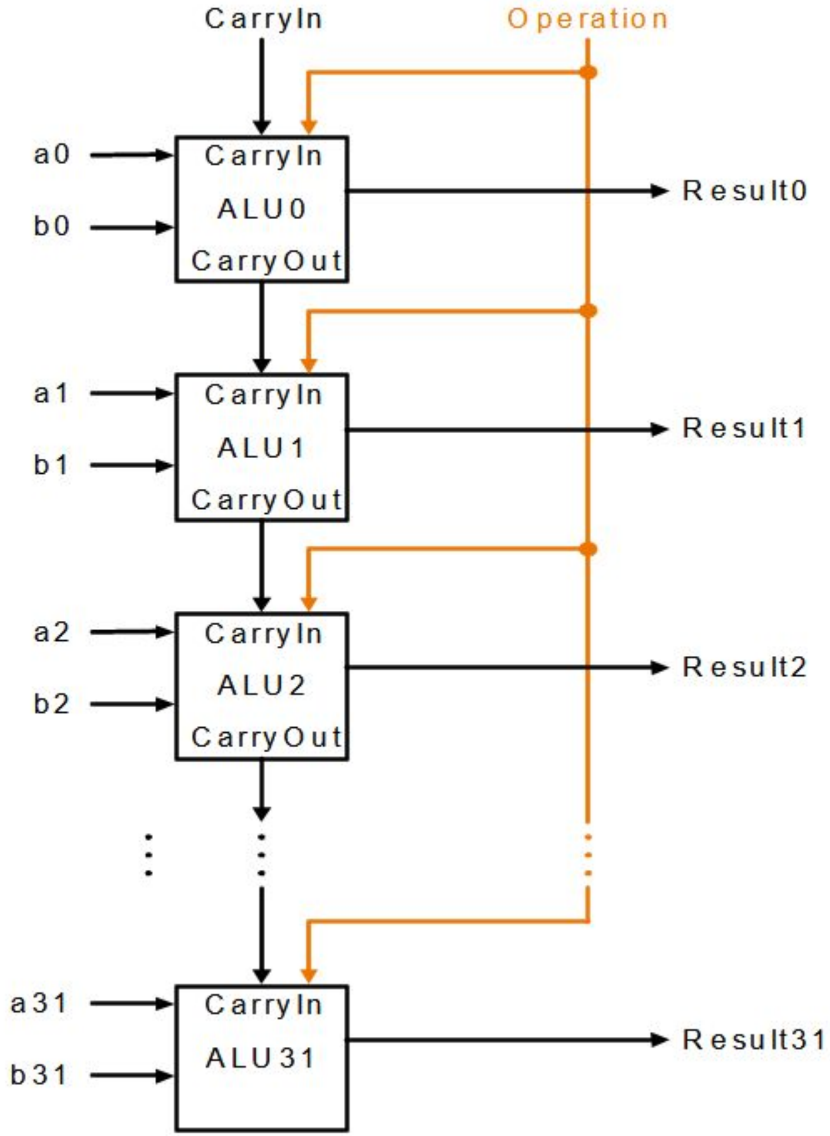
Unidade Lógica Aritmética (ULA)



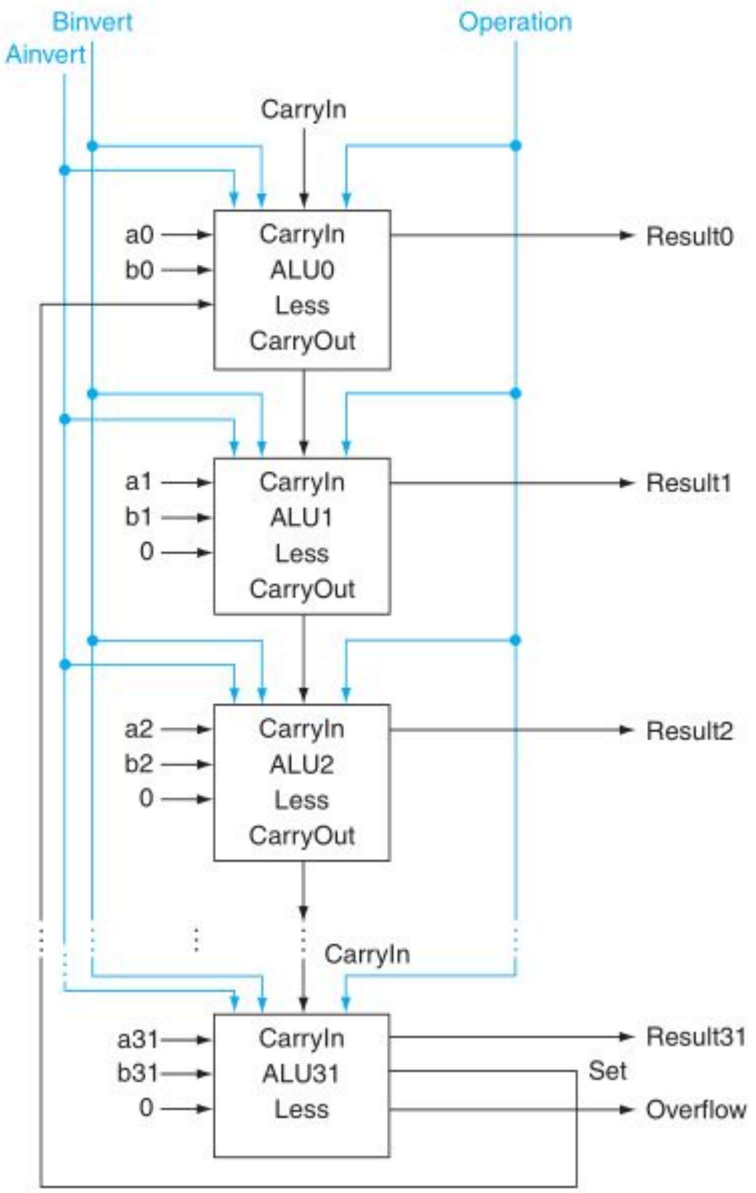
Como projetar uma ULA de 32 bits usando uma ULA de 1 bit?

ULA de 32 bits





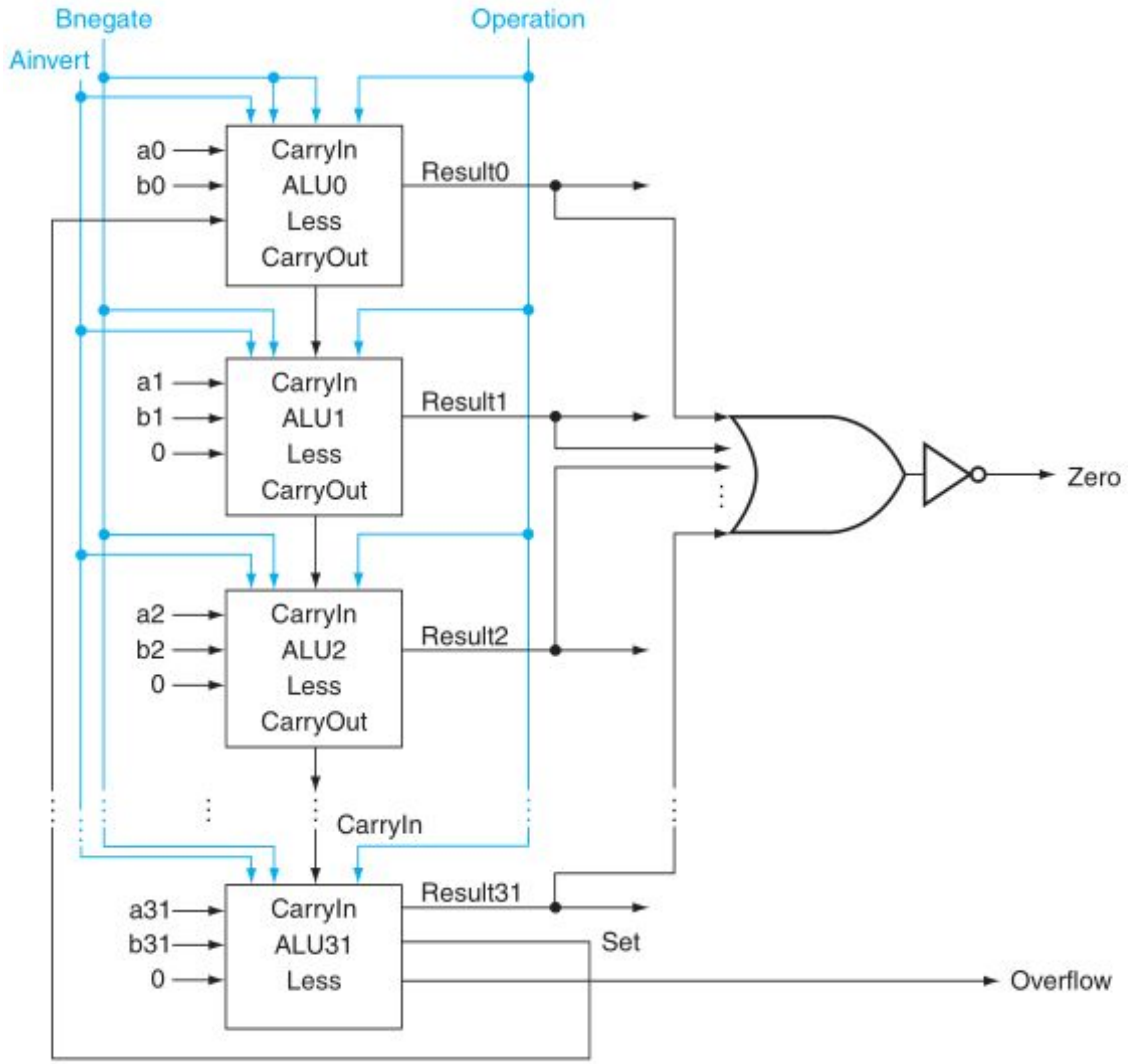
Unidade Lógica Aritmética (ULA)



<u>ULA de 32 bits</u>

É uma "cópia" de 31 ULAs de 1 bit do topo

Obs: *Ainvert* e *Binvert* permitem um ***NOR***



<u>ULA de 32 bits "final"</u>

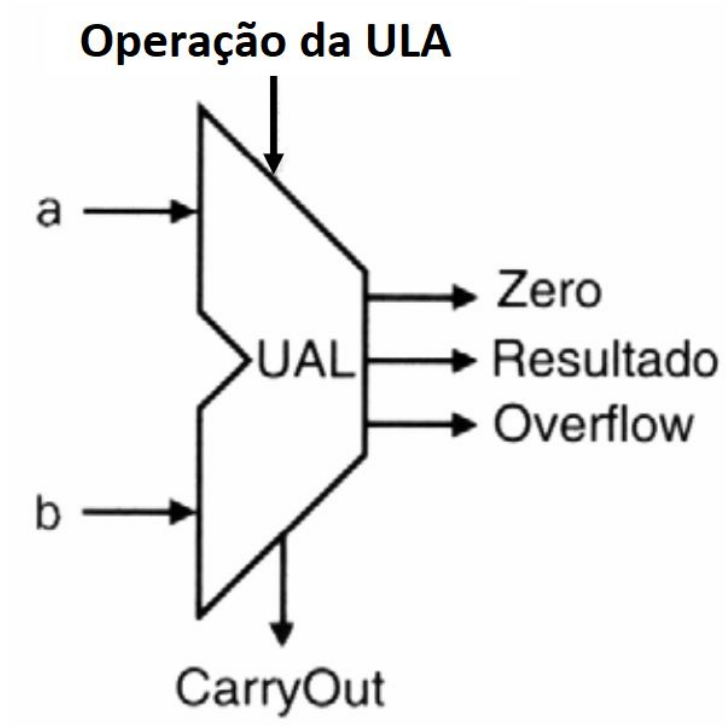
Adiciona um detector de zero, para operação de comparação.

a - b: se o resultado for zero, são iguais.

ALU control lines	Function
0000	AND
0001	OR
0010	add
0110	subtract
0111	set on less than
1100	NOR

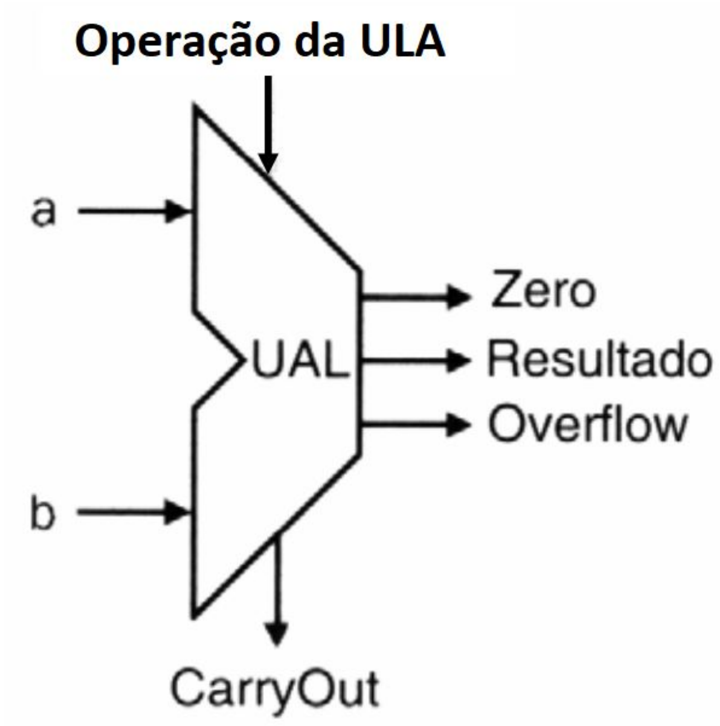
Unidade Lógica Aritmética (ULA)

Símbolo geral da ULA



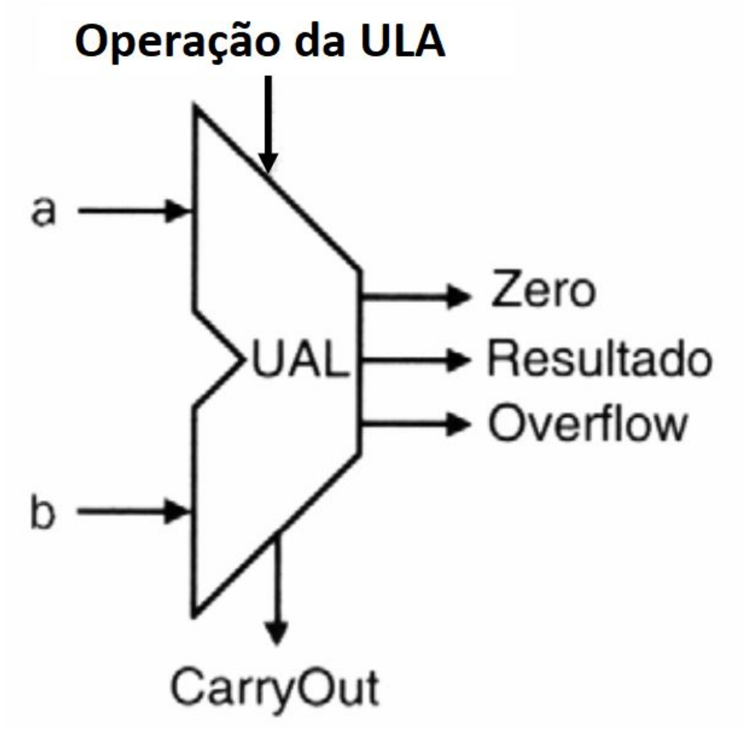
Unidade Lógica Aritmética (ULA)

Símbolo geral da ULA



Qual é o problema de uma ULA projetada desta forma?

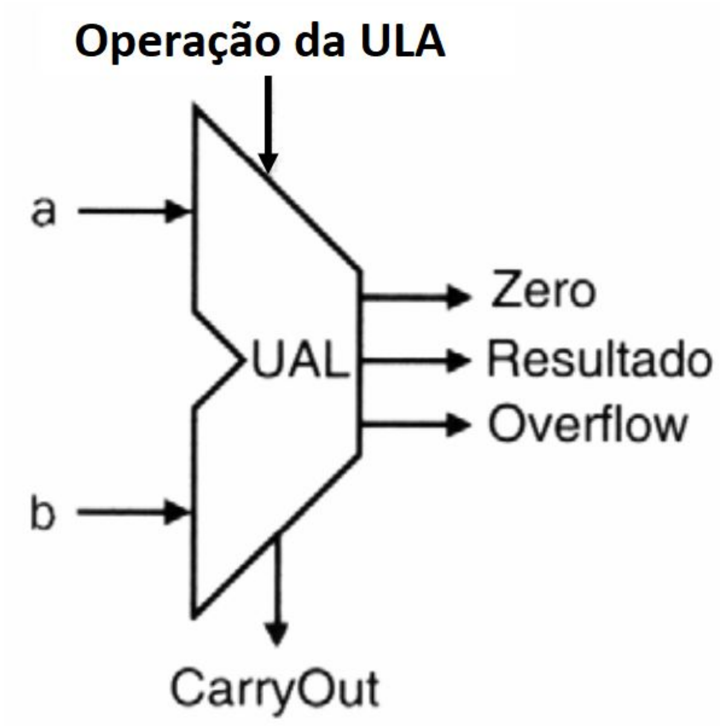
Símbolo geral da ULA



Qual é o problema de uma ULA projetada desta forma?

Com que "velocidade" é possível pode somar dois operandos de 32 bits?

Símbolo geral da ULA



Qual é o problema de uma ULA projetada desta forma?

Com que “velocidade” é possível pode somar dois operandos de 32 bits?

Observe que as entradas “a” e “b” podem ser perfeitamente determinadas a qualquer tempo. Porém, a entrada “**CarryIn**” - não demonstrada no desenho - de um determinado somador de 1 bit **depende do resultado** da operação realizada no **somador vizinho**.

Carry Lookahead

- A solução é usar o mecanismo de **Carry Lookahead**

- Em um primeiro nível de abstração, o somador usa os conceitos de **propagador** e **gerador**

Sabemos que o CarryIn de um *somador2* é o CarryOut do *somador1*:

$$c2 = (b1 \cdot c1) + (a1 \cdot c1) + (a1 \cdot b1)$$
$$c1 = (b0 \cdot c0) + (a0 \cdot c0) + (a0 \cdot b0)$$

Carry Lookahead - Propagador e Gerador

Se fatorarmos a equação, temos:

$$\begin{aligned} ci + 1 &= (bi \cdot ci) + (ai \cdot ci) + (ai \cdot bi) \\ &= (ai \cdot bi) + (ai + bi) \cdot ci \end{aligned}$$

Ao aplicar a equação para o somador completo, percebe-se um padrão de repetição de (a*i*.b*i*) e (a*i*+b*i*)

Esses dois fatores são chamados de gerador (g*i*) e propagador (p*i*)

Carry Lookahead - Propagador e Gerador

"No fim do dia", teremos uma equação baseada na definição do propagador e gerador:

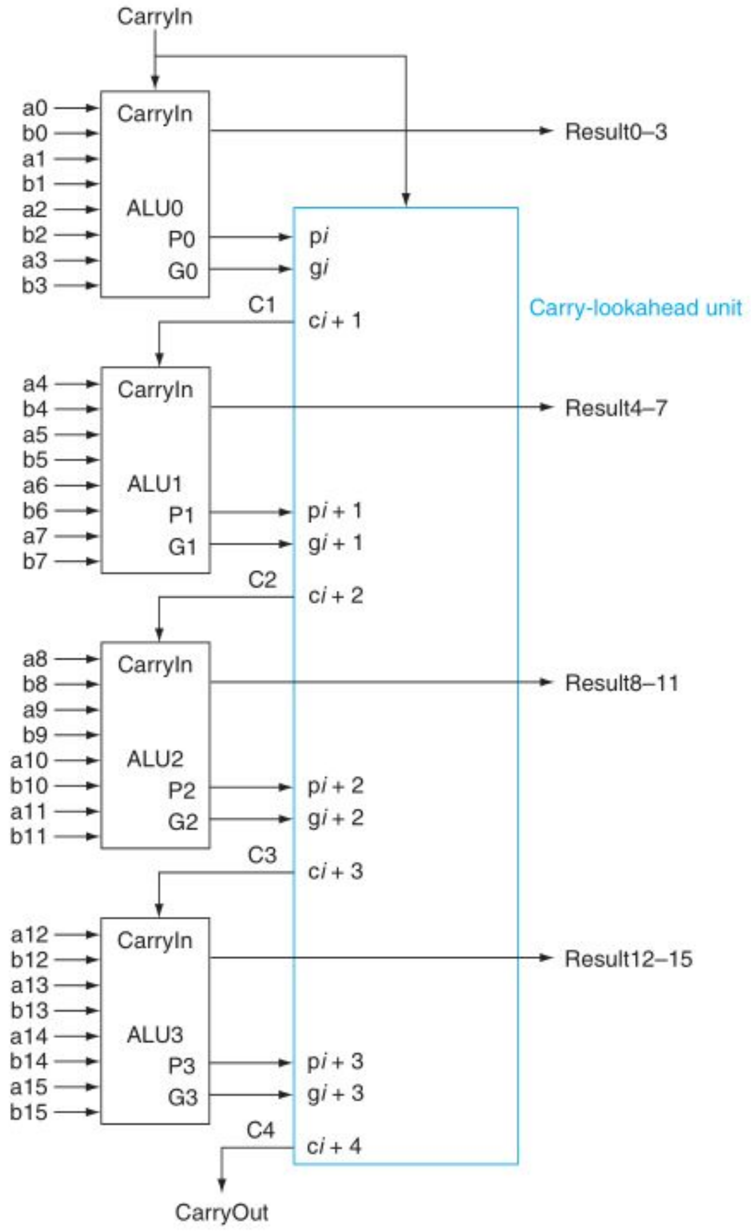
$$
\begin{aligned}
c1 &= g0 + (p0 \cdot c0) \\
c2 &= g1 + (p1 \cdot g0) + (p1 \cdot p0 \cdot c0) \\
c3 &= g2 + (p2 \cdot g1) + (p2 \cdot p1 \cdot g0) + (p2 \cdot p1 \cdot p0 \cdot c0) \\
c4 &= g3 + (p3 \cdot g2) + (p3 \cdot p2 \cdot g1) + (p3 \cdot p2 \cdot p1 \cdot g0) \\
&\quad + (p3 \cdot p2 \cdot p1 \cdot p0 \cdot c0)
\end{aligned}
$$

OBS: Para entender no detalhe, leia o apêndice **B-6** do livro
"PATTERSON, D.A; HENESSY, J.L. Organização e Projeto de Computadores - A interface hardware/software"

Carry Lookahead

Quatro ALUs de 4 bits usando Carry Lookahead para formar um somador de 16 bits

Observe que os bits de "carry" são produzidos pela unidade de carry-lookahead.



Aritmética Computacional

- Sinais e Complemento
- Adição e Subtração
- Operações Lógicas
- A Unidade Lógica Aritmética
- Multiplicação e Divisão
- Ponto Flutuante

Multiplicação

- Exemplo 1

```
multiplicando        1000
multiplicador      x 1001
                     1000
                    0000
                   0000
                  1000
      produto     1001000
```

Número de dígitos do resultado:
multiplicando + multiplicador

32 bits + 32 bits = 64 bits

Multiplicação

- Exemplo 2

```
multiplicando      0010
multiplicador    x 0011
                   0010
                  0010
                 0000
                0000
      produto   0000110
```

Número de dígitos do resultado:
multiplicando + multiplicador

32 bits + 32 bits = 64 bits

Multiplicação

Algoritmo seguindo os exemplos práticos:

1) Coloque uma cópia do multiplicando (1 x multiplicando) no lugar apropriado, se o dígito do multiplicando for igual a 1, ou

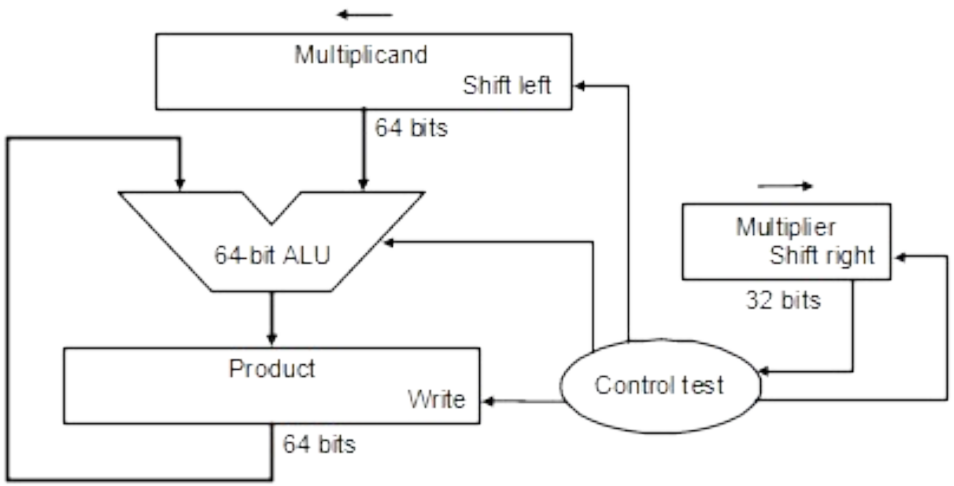
2) Coloque 0 (0 x multiplicando) no lugar apropriado, se o dígito do multiplicando for igual a 0.

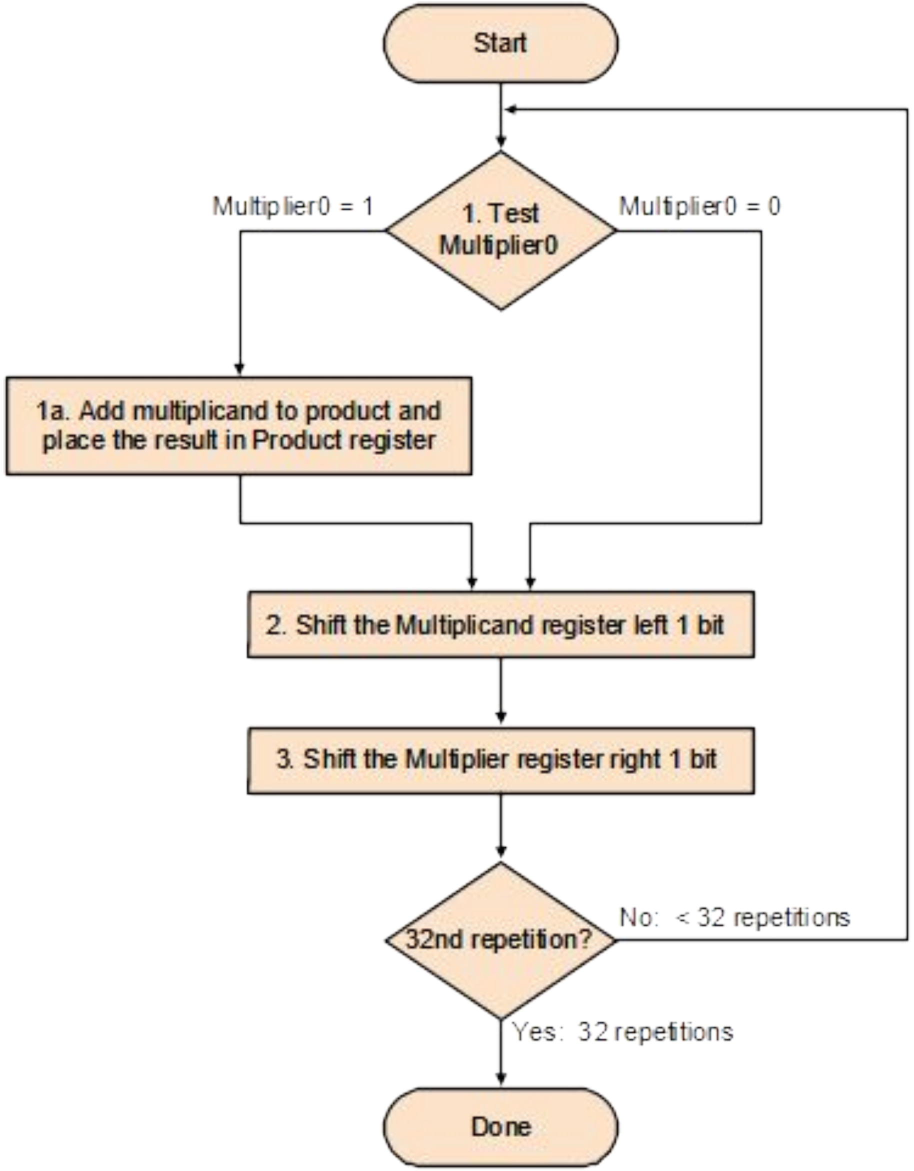
> Veremos a seguir 3 versões do algoritmo de multiplicação para 32 bits na entrada (32 x 32 bits)

Multiplicação - 1ª versão

```
multiplicando       1000
multiplicador     x 1001
produto             1000
```

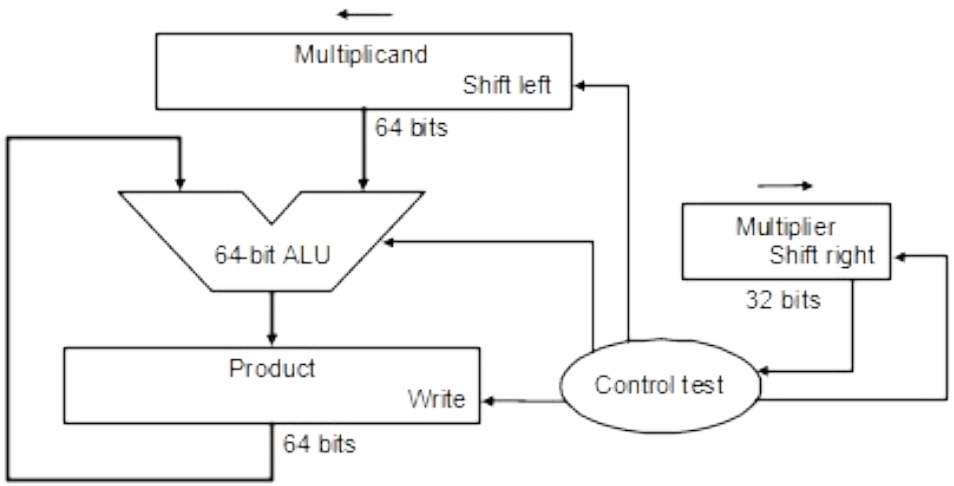
1000

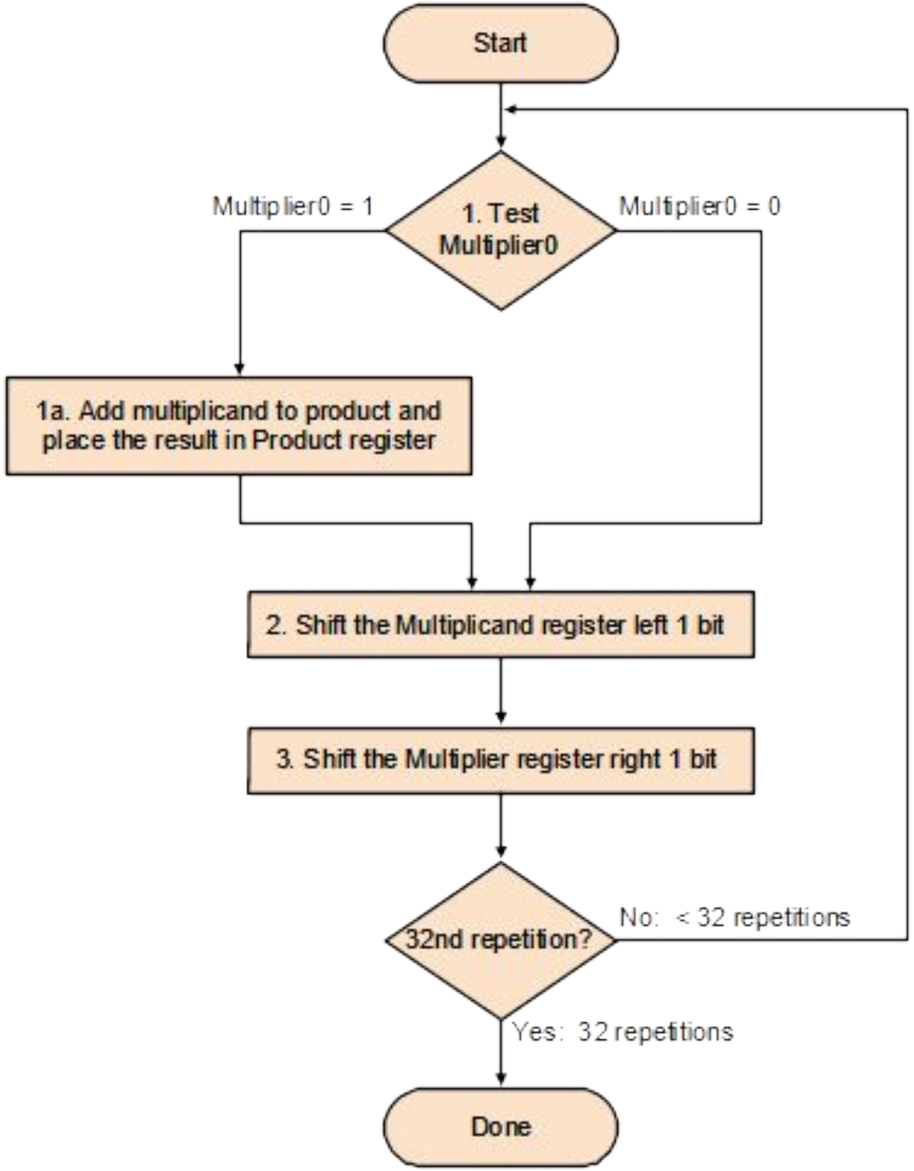




Multiplicação - 1ª versão

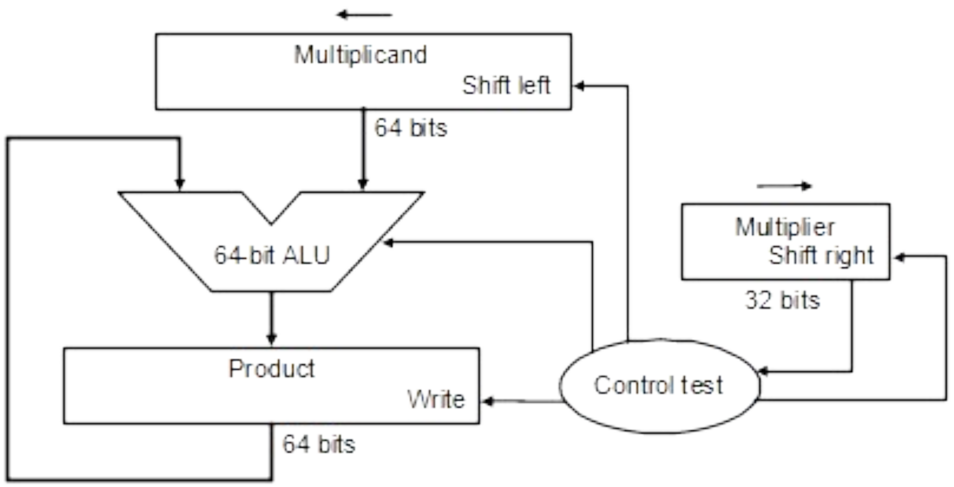
```
multiplicando        1000
multiplicador      x 1001
produto              1000

   10000
 x   100
   00000

1000
```

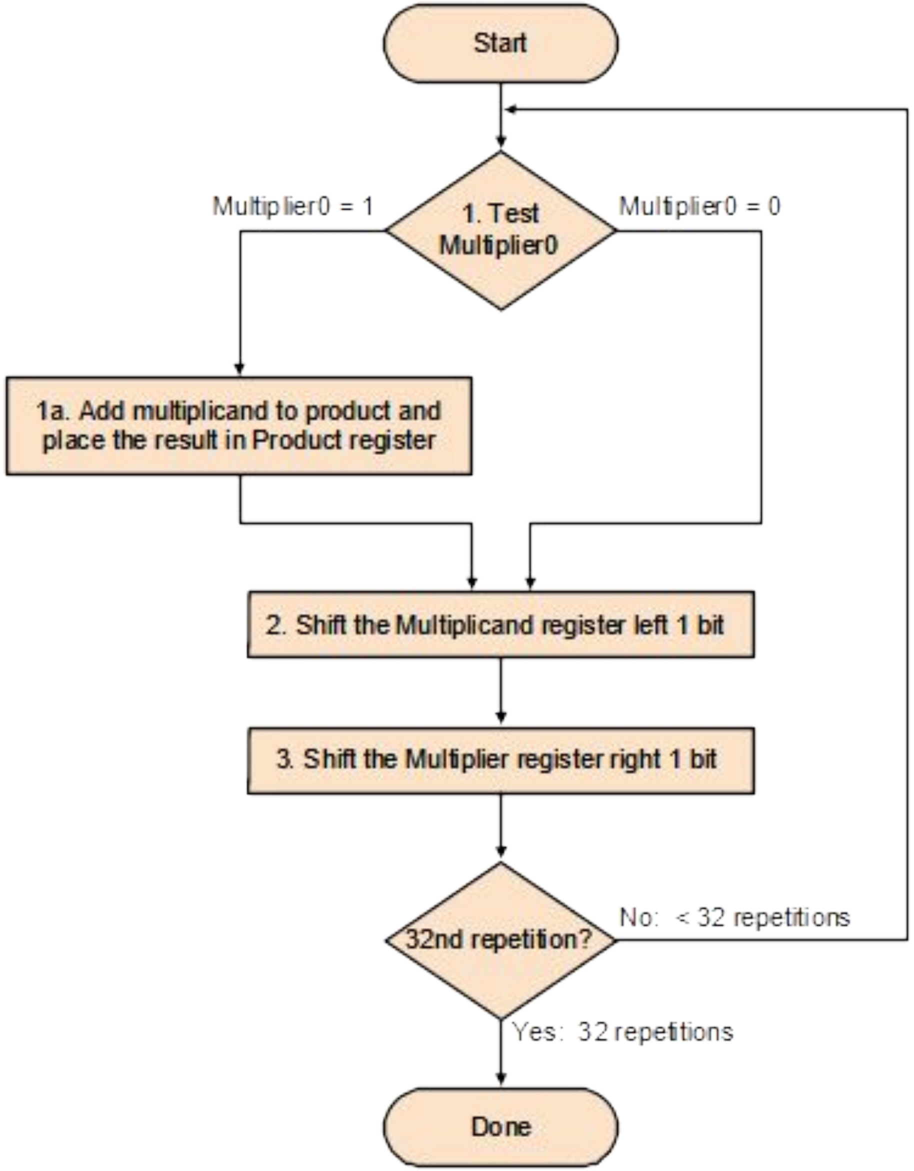




Multiplicação - 1ª versão

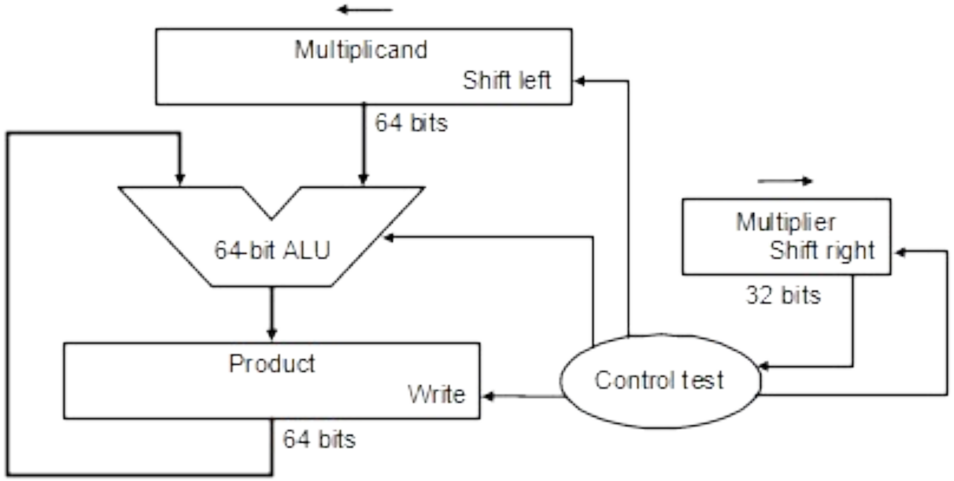
```
multiplicando        1000
multiplicador      x 1001
produto              1000

   10000    100000
x    100  x     10
   00000     00000

1000
```

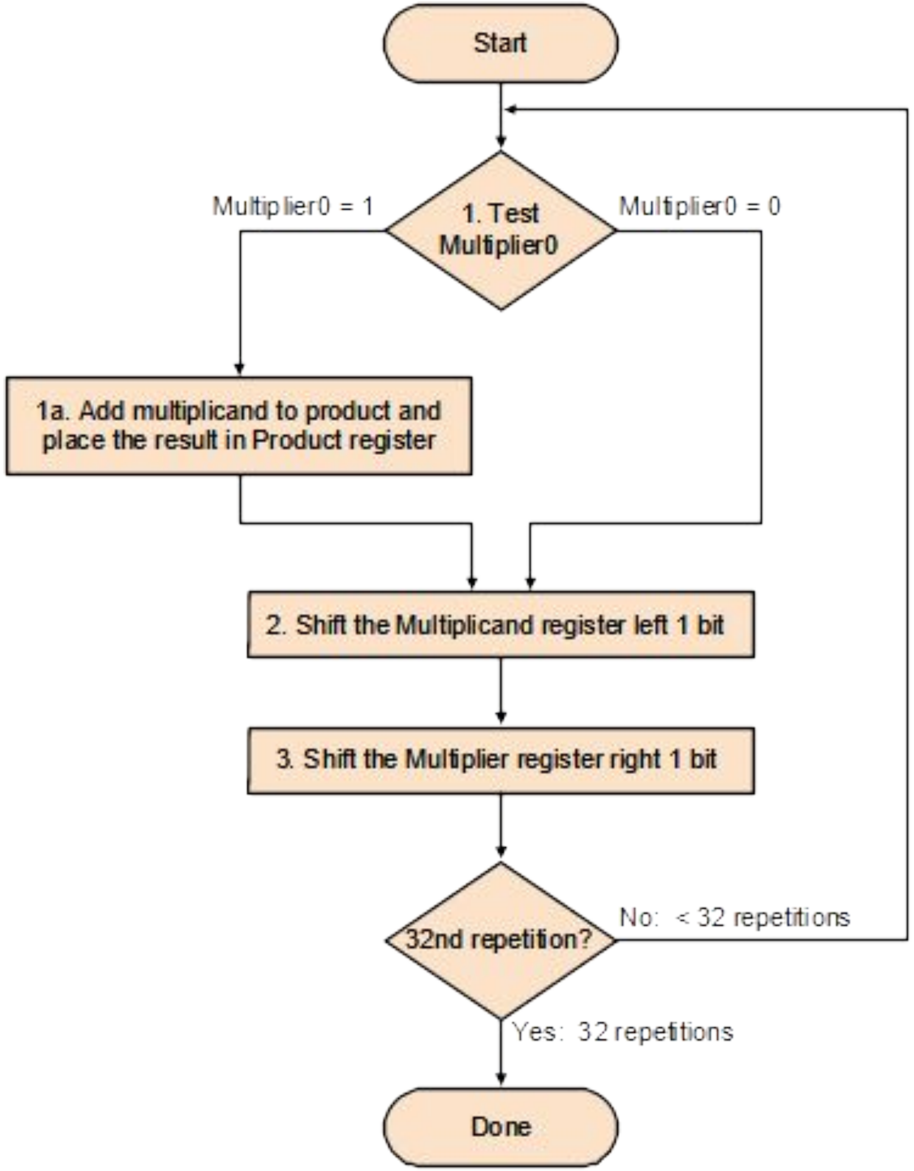




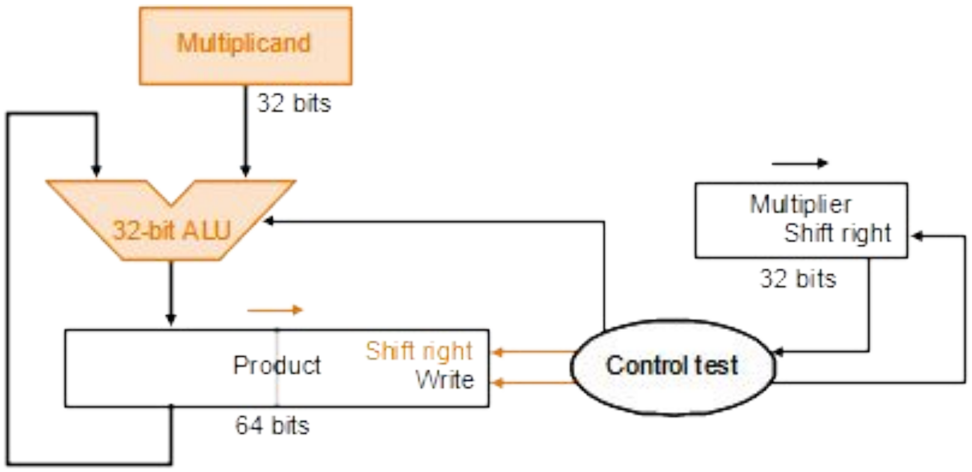
Multiplicação - 1ª versão

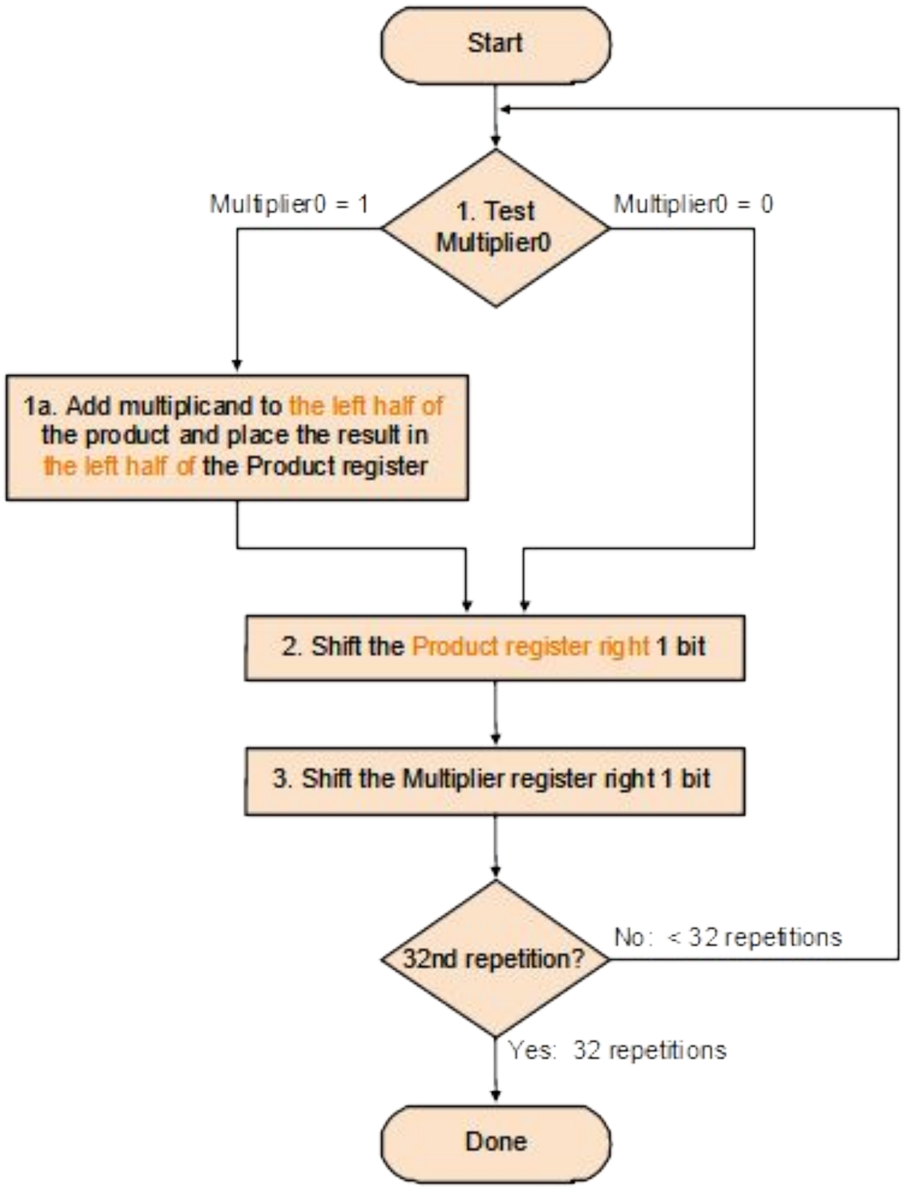
```
multiplicando        1000
multiplicador      x 1001
produto              1000

   10000    100000    1000000
x    100  x     10  x       1
   00000     00000    1000000

1000 + 1000000 = 1001000
```





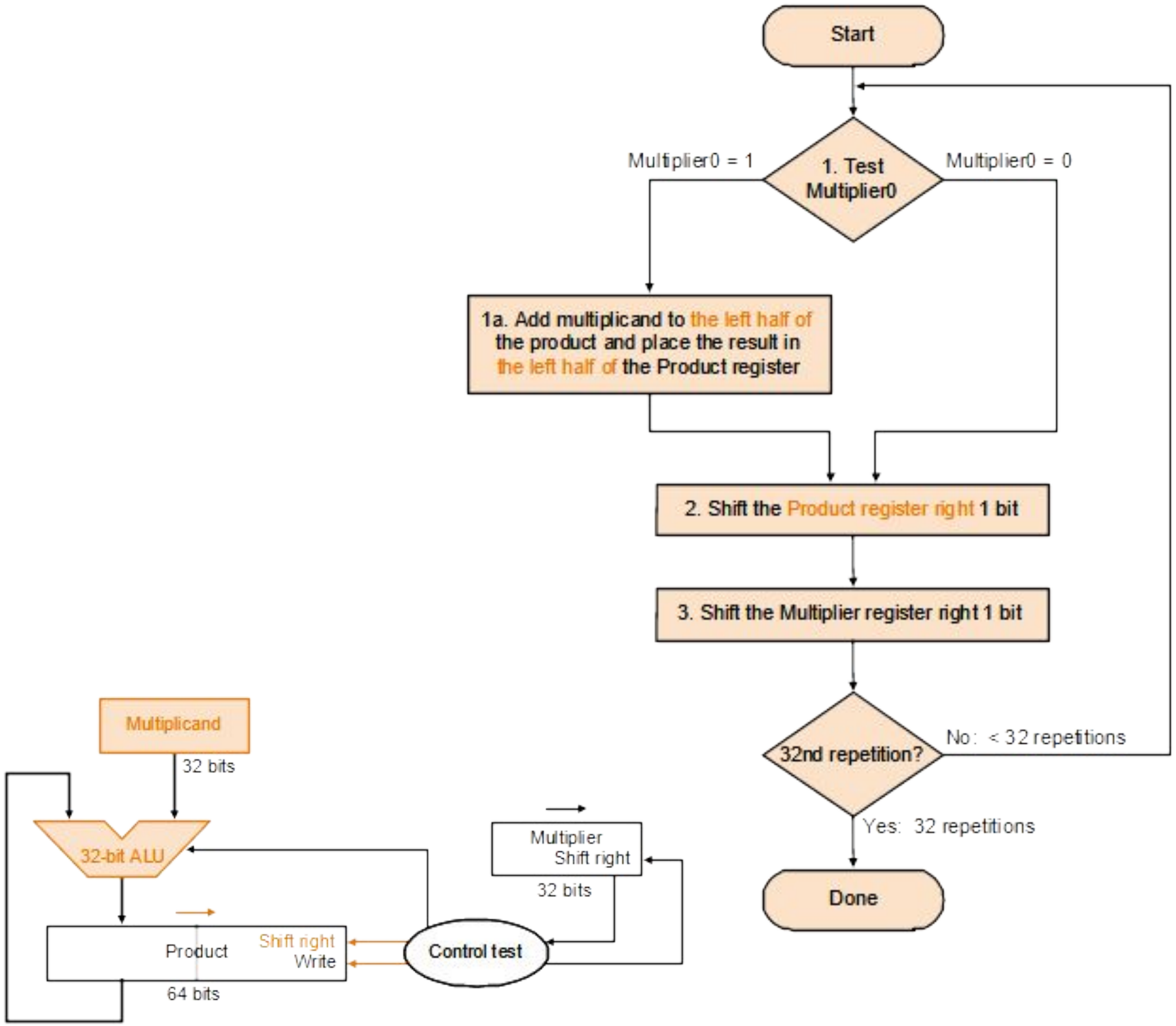
Multiplicação - 2ª versão





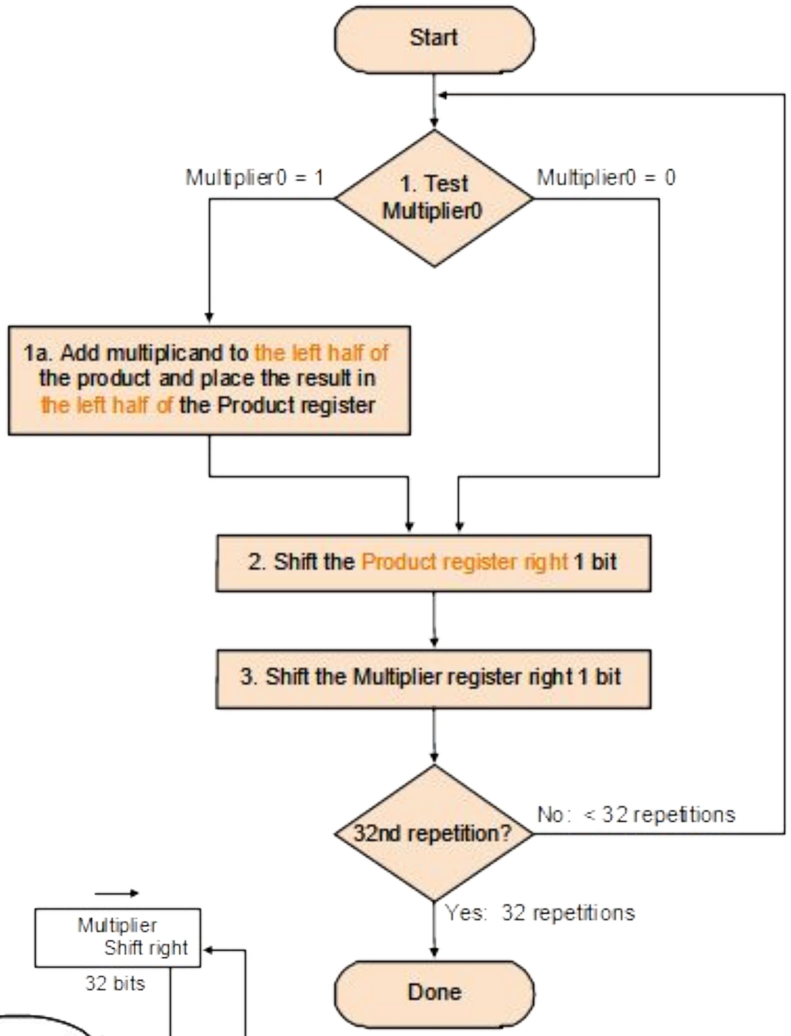
Multiplicação - 2ª versão

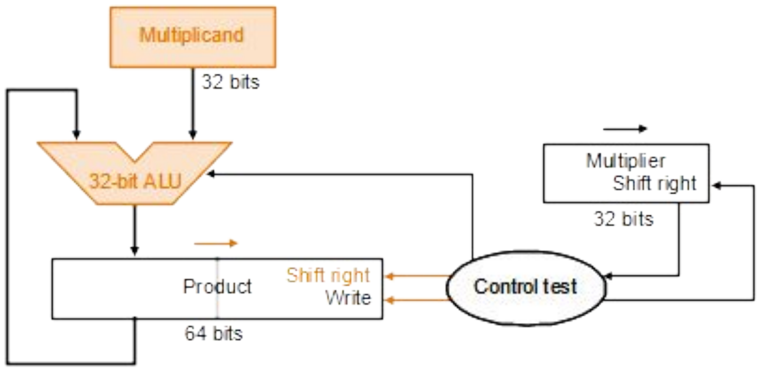
#	Passo	Multiplicador (MR)	Multiplicando (MD)	Produto (P)
0	Valores iniciais	0011	0010	0000 0000
1				
1				
1				
2				
2				
2				
3				
3				
3				
4				
4				
4				



Multiplicação - 2ª versão

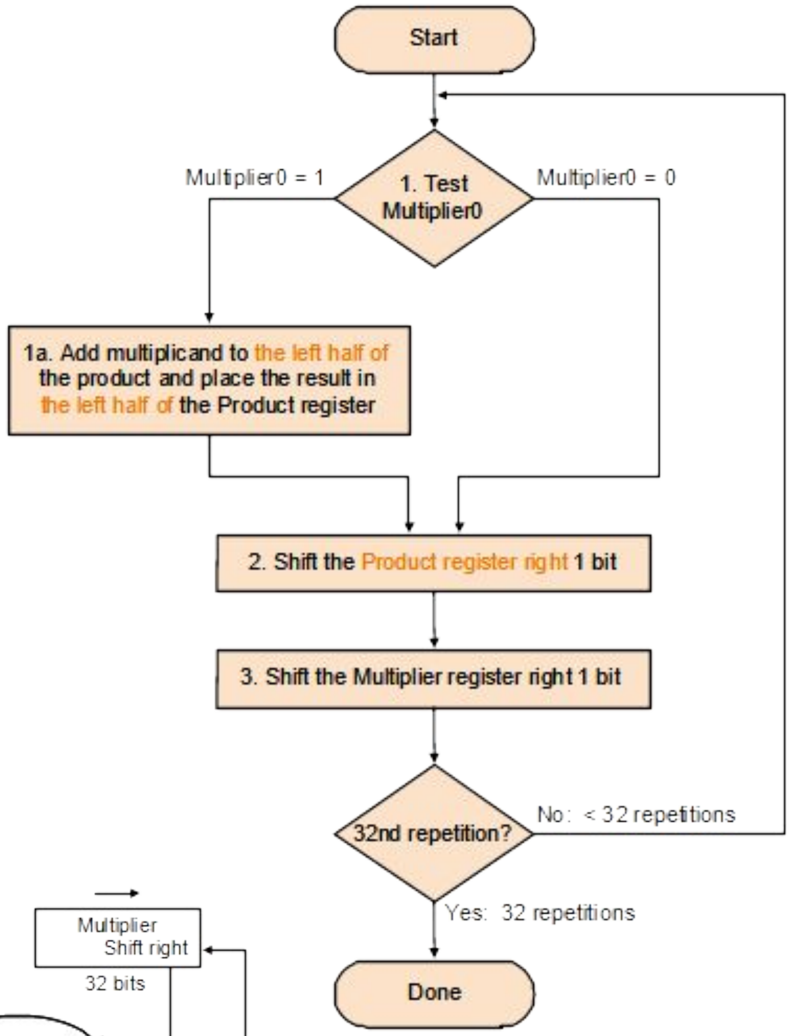
#	Passo	Multiplicador (MR)	Multiplicando (MD)	Produto (P)
0	Valores iniciais	0011	0010	0000 0000
1	1 => P = P+MD	0011	0010	0010 0000
1	Desloca P à direita	0011	0010	0001 0000
1	Desloca MR à direita	0001	0010	0001 0000
2				
2				
2				
3				
3				
3				
4				
4				
4				

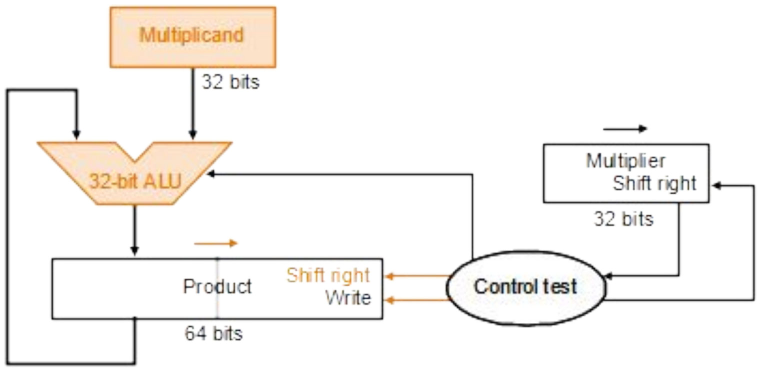




Multiplicação - 2ª versão

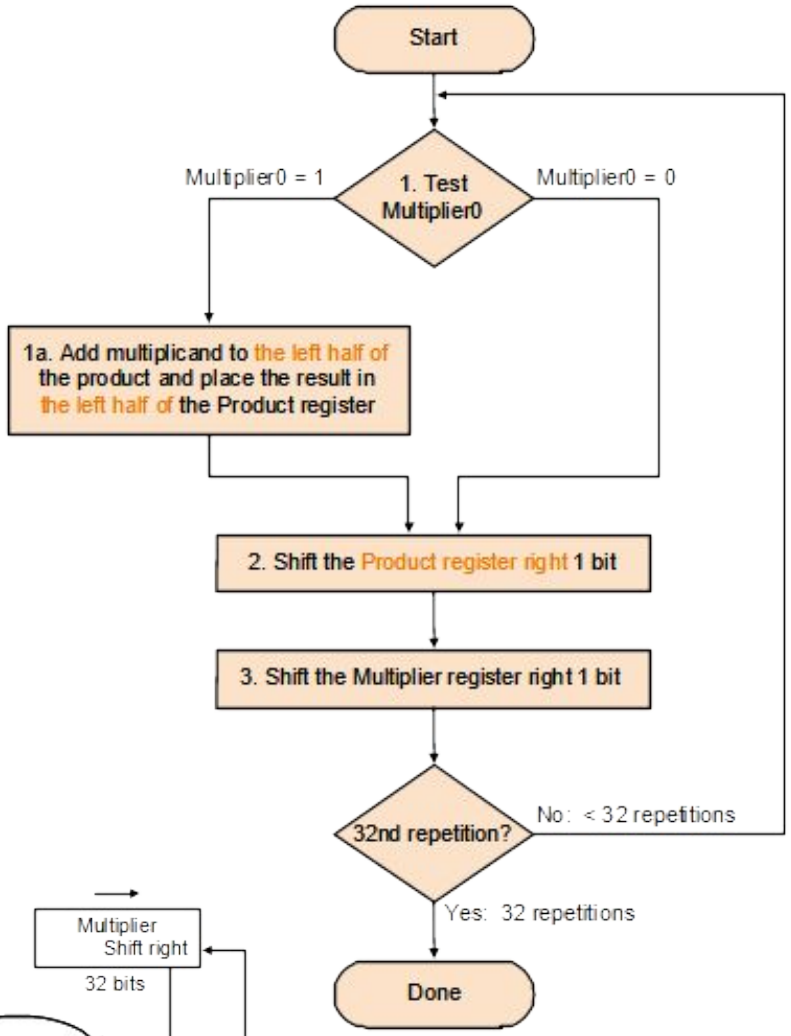
#	Passo	Multiplicador (MR)	Multiplicando (MD)	Produto (P)
0	Valores iniciais	0011	0010	0000 0000
1	1 => P = P+MD	0011	0010	0010 0000
1	Desloca P à direita	0011	0010	0001 0000
1	Desloca MR à direita	0001	0010	0001 0000
2	1 => P = P+MD	0001	0010	0011 0000
2	Desloca P à direita	0001	0010	0001 1000
2	Desloca MR à direita	0000	0010	0001 1000
3				
3				
3				
4				
4				
4				

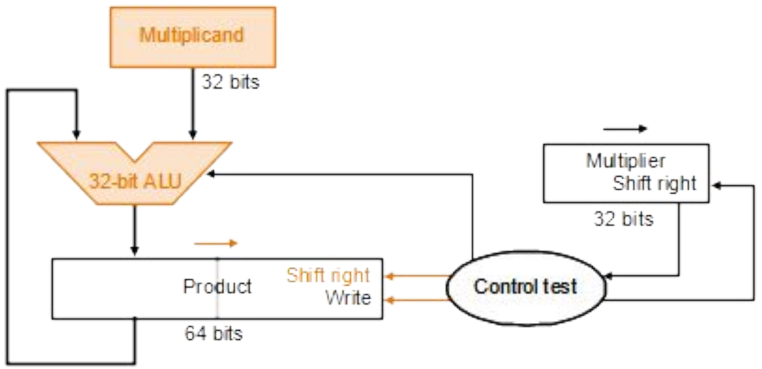




Multiplicação - 2ª versão

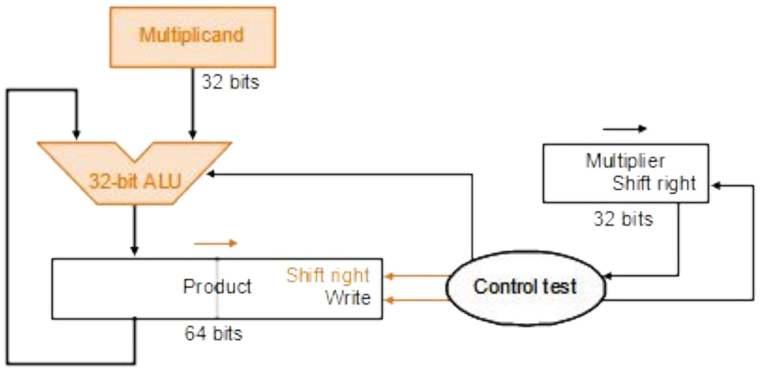
#	Passo	Multiplicador (MR)	Multiplicando (MD)	Produto (P)
0	Valores iniciais	0011	0010	0000 0000
1	1 => P = P+MD	0011	0010	0010 0000
1	Desloca P à direita	0011	0010	0001 0000
1	Desloca MR à direita	0001	0010	0001 0000
2	1 => P = P+MD	0001	0010	0011 0000
2	Desloca P à direita	0001	0010	0001 1000
2	Desloca MR à direita	0000	0010	0001 1000
3	0 => Não faça nada	0000	0010	0001 1000
3	Desloca P à direita	0000	0010	0000 1100
3	Desloca MR à direita	0000	0010	0000 1100
4				
4				
4				

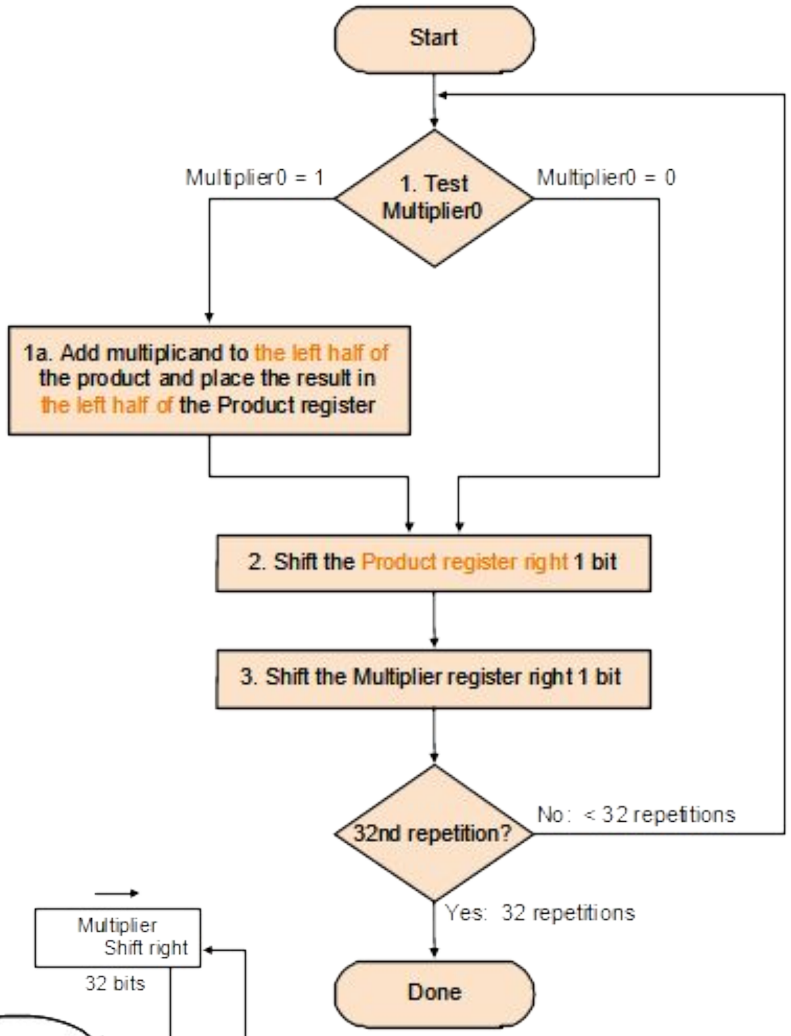




Multiplicação - 2ª versão

#	Passo	Multiplicador (MR)	Multiplicando (MD)	Produto (P)
0	Valores iniciais	0011	0010	0000 0000
1	1 => P = P+MD	0011	0010	0010 0000
1	Desloca P à direita	0011	0010	0001 0000
1	Desloca MR à direita	0001	0010	0001 0000
2	1 => P = P+MD	0001	0010	0011 0000
2	Desloca P à direita	0001	0010	0001 1000
2	Desloca MR à direita	0000	0010	0001 1000
3	0 => Não faça nada	0000	0010	0001 1000
3	Desloca P à direita	0000	0010	0000 1100
3	Desloca MR à direita	0000	0010	0000 1100
4	0 => Não faça nada	0000	0010	0000 1100
4	Desloca P à direita	0000	0010	0000 0110
4	Desloca MR à direita	0000	0010	**0000 0110**

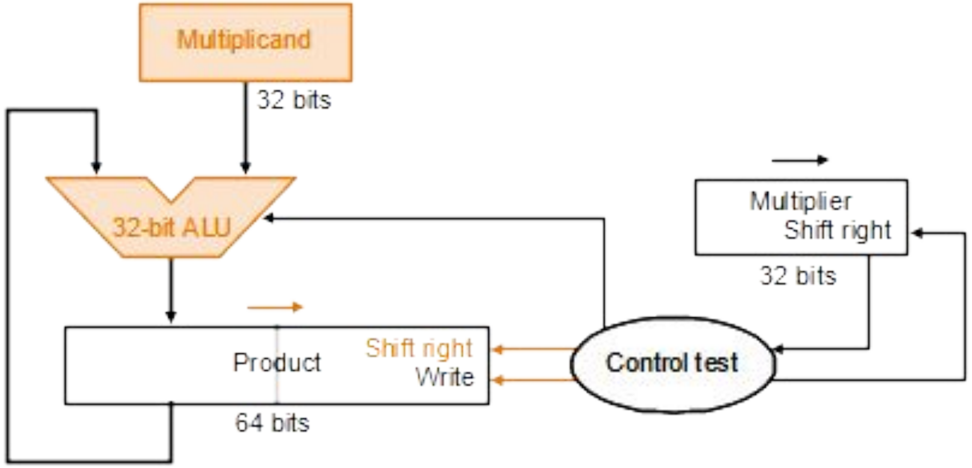


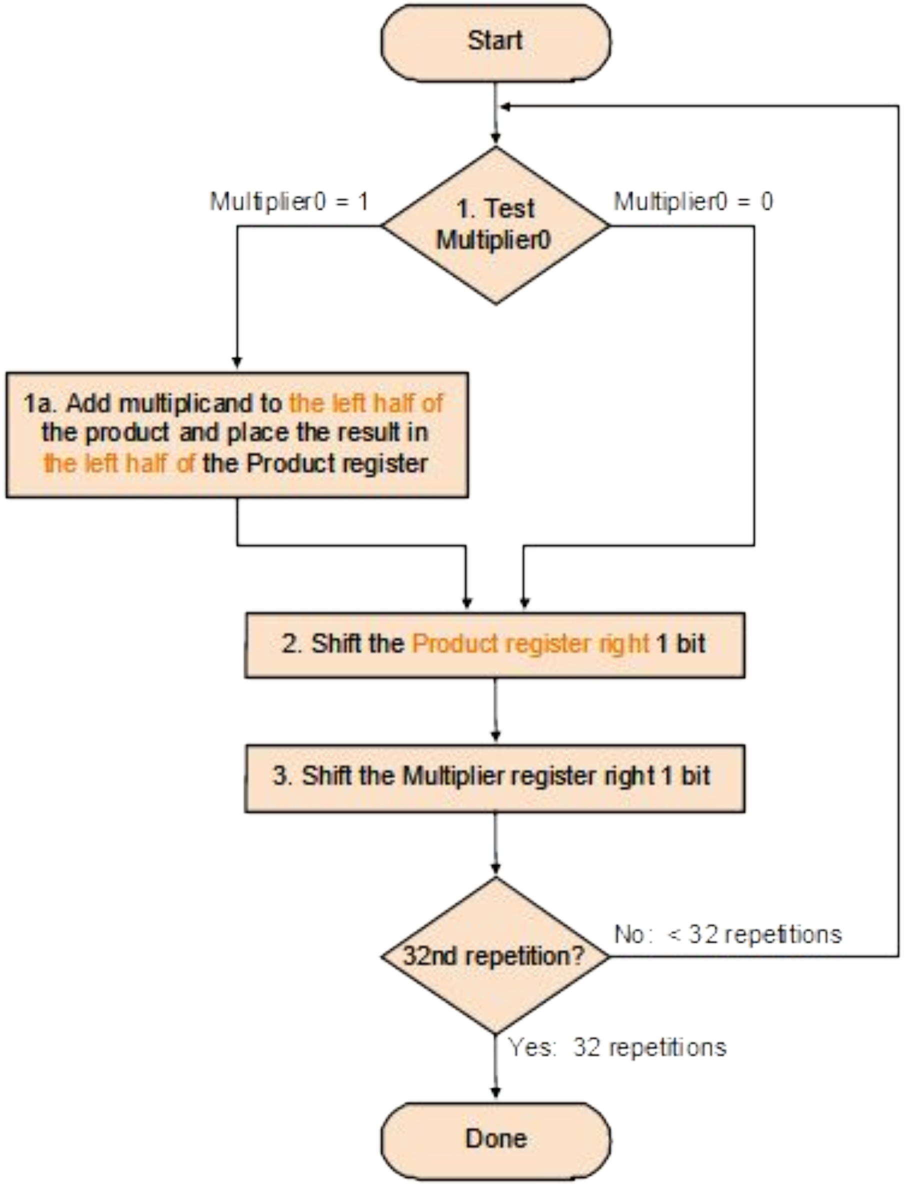


Multiplicação - 2ª versão

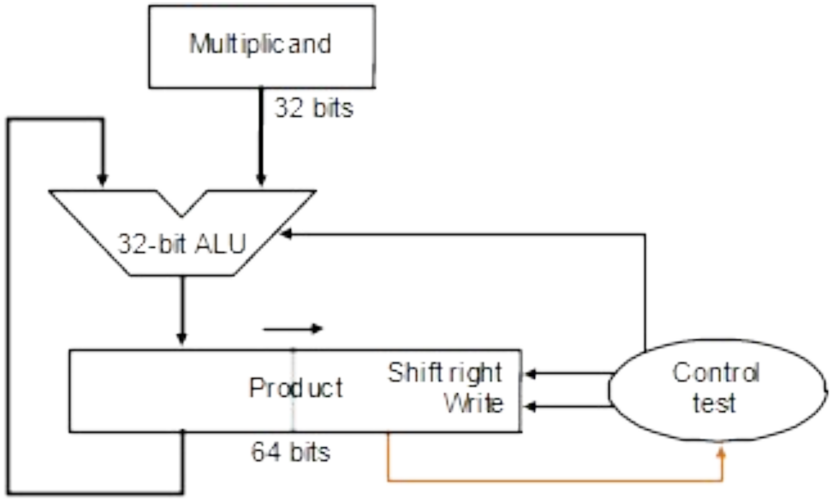
O registrador reservado ao produto desperdiça tanto espaço quanto o do multiplicador.

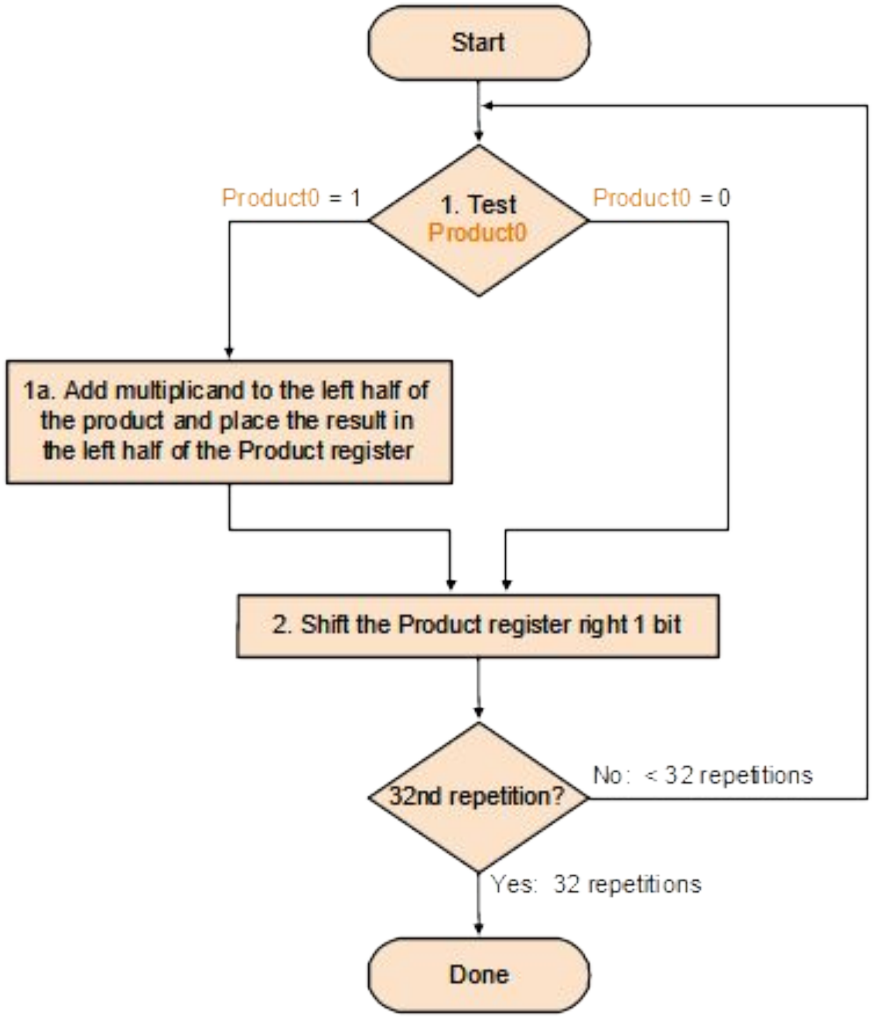
À medida que o desperdício do espaço do produto se reduzia, a mesma coisa acontecia com o multiplicador





Multiplicação - 3ª versão

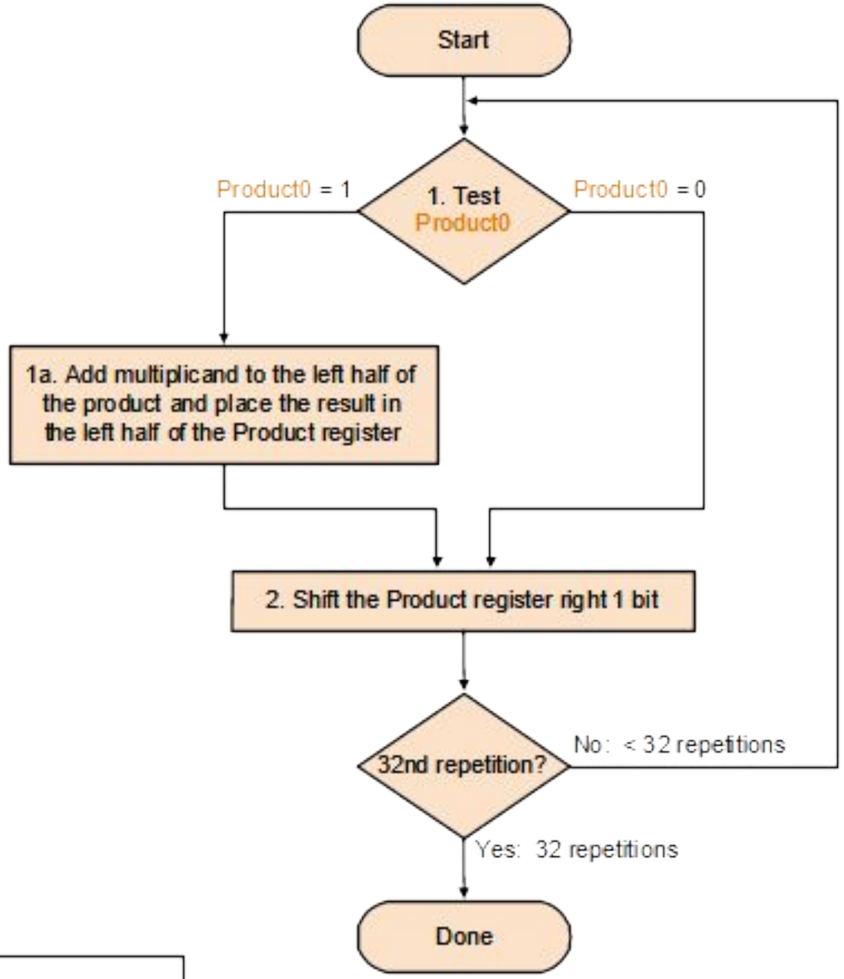


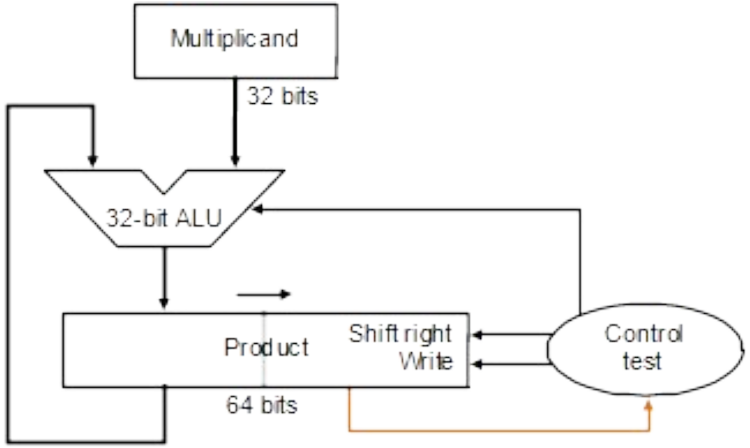


Multiplicação - 3ª versão

Multiplicador = 0011

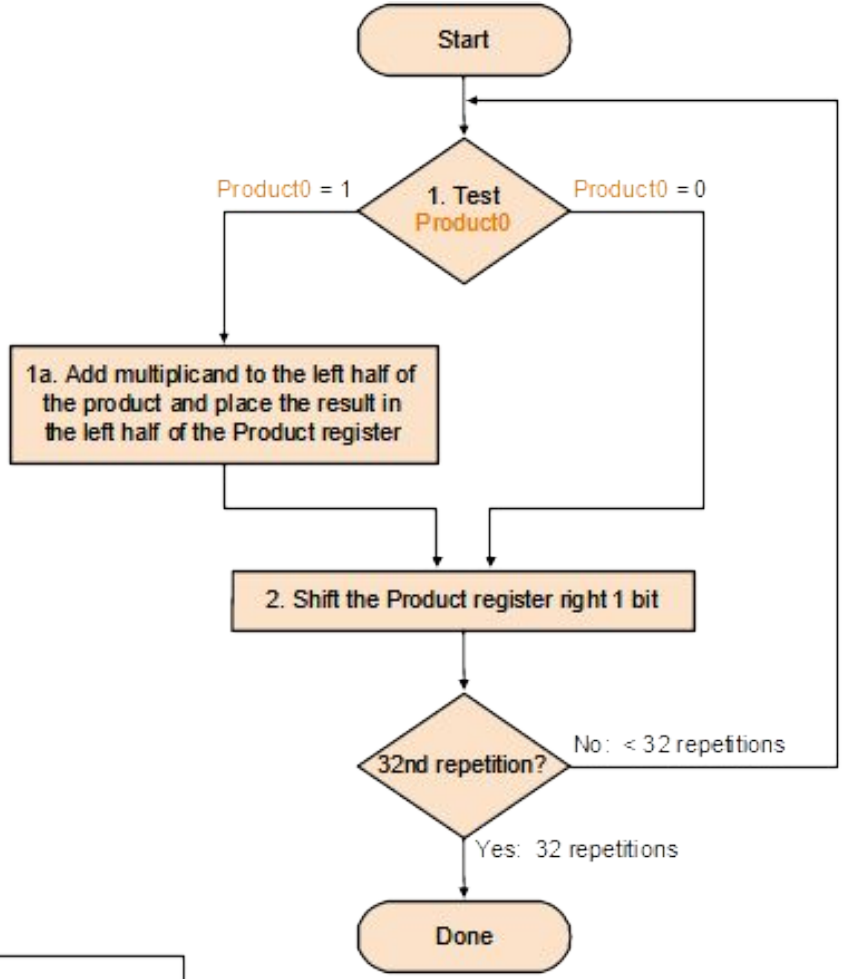
#	Passo	Multiplicando (MD)	Produto (P)
0	Valores iniciais	0010	0000 0011
1			
1			
2			
2			
3			
3			
4			
4			

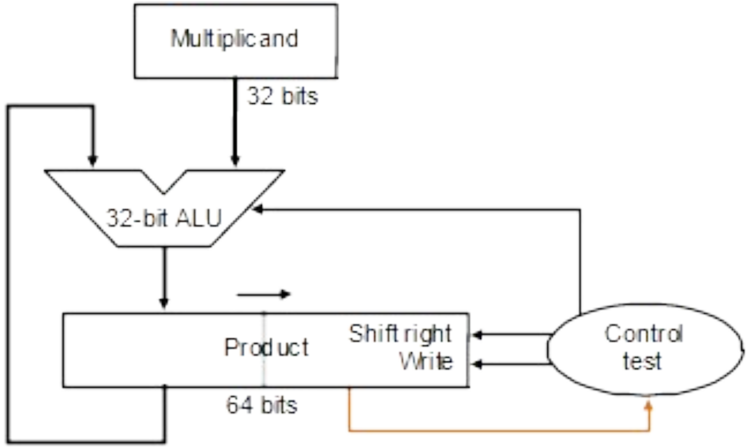




Multiplicação - 3ª versão

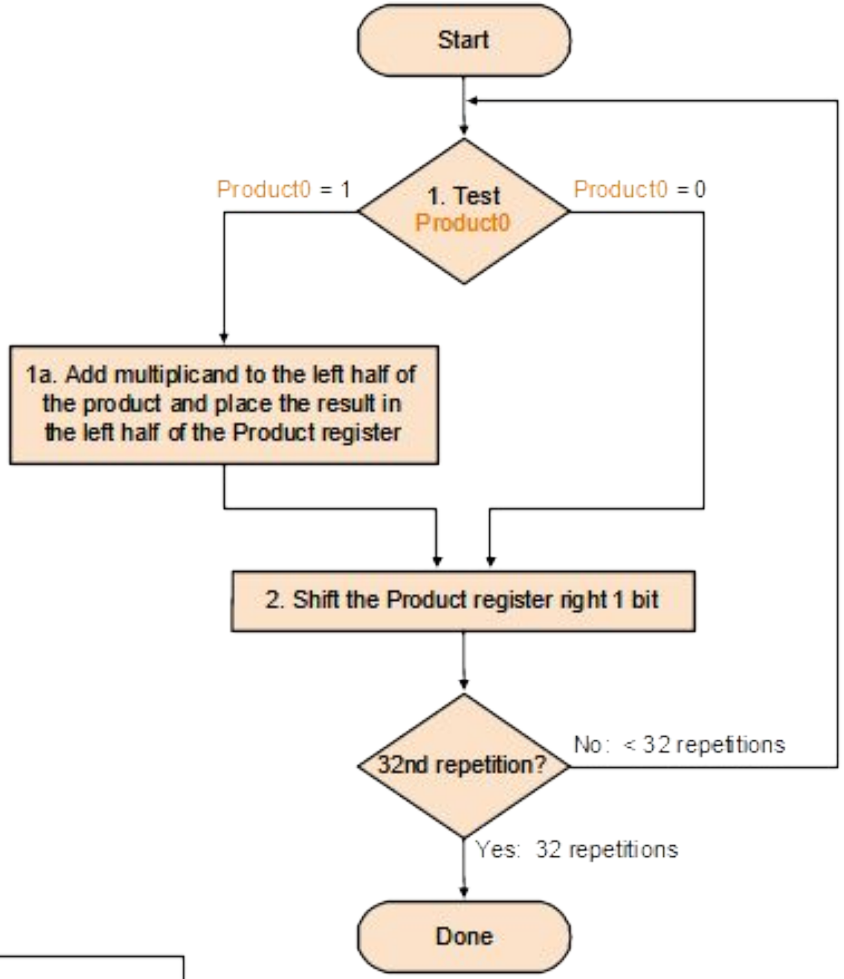
#	Passo	Multiplicando (MD)	Produto (P)
0	Valores iniciais	0010	0000 0011
1	1 => P = P+MD	0010	0010 0011
1	Desloca P à direita	0010	0001 0001
2			
2			
3			
3			
4			
4			

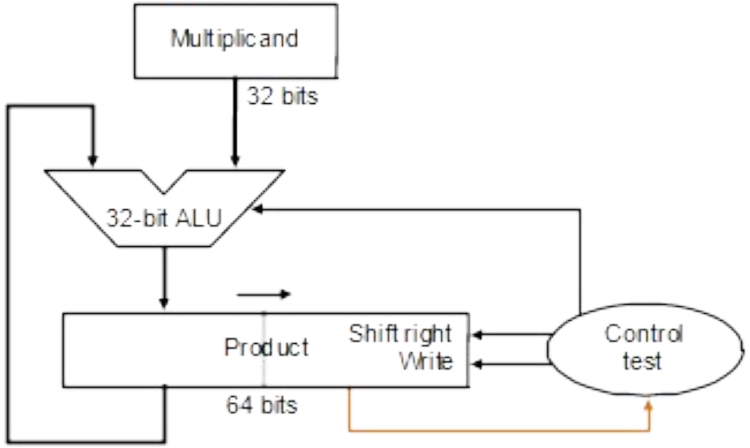




Multiplicação - 3ª versão

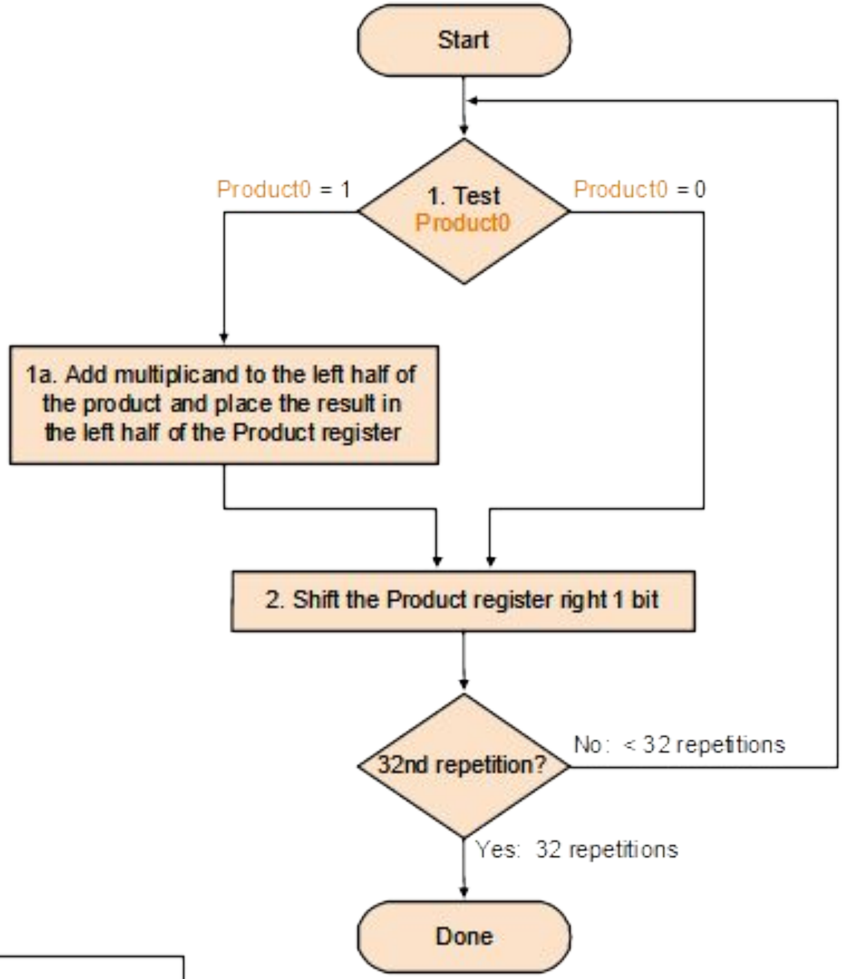
#	Passo	Multiplicando (MD)	Produto (P)
0	Valores iniciais	0010	0000 0011
1	1 => P = P+MD	0010	0010 0011
1	Desloca P à direita	0010	0001 0001
2	1 => P = P+MD	0010	0011 0001
2	Desloca P à direita	0010	0001 1000
3			
3			
4			
4			

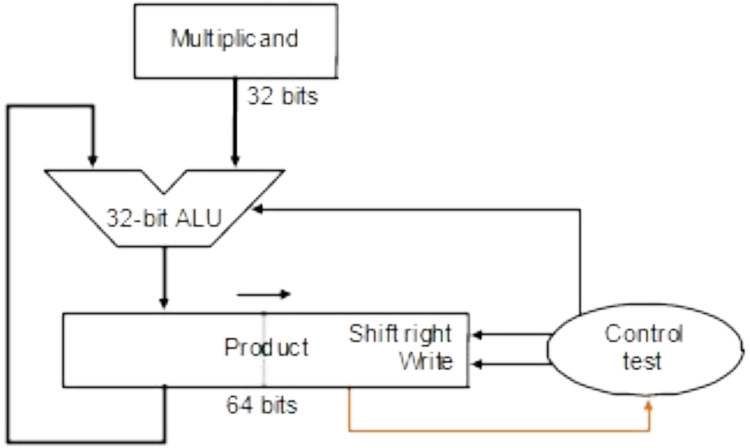




Multiplicação - 3ª versão

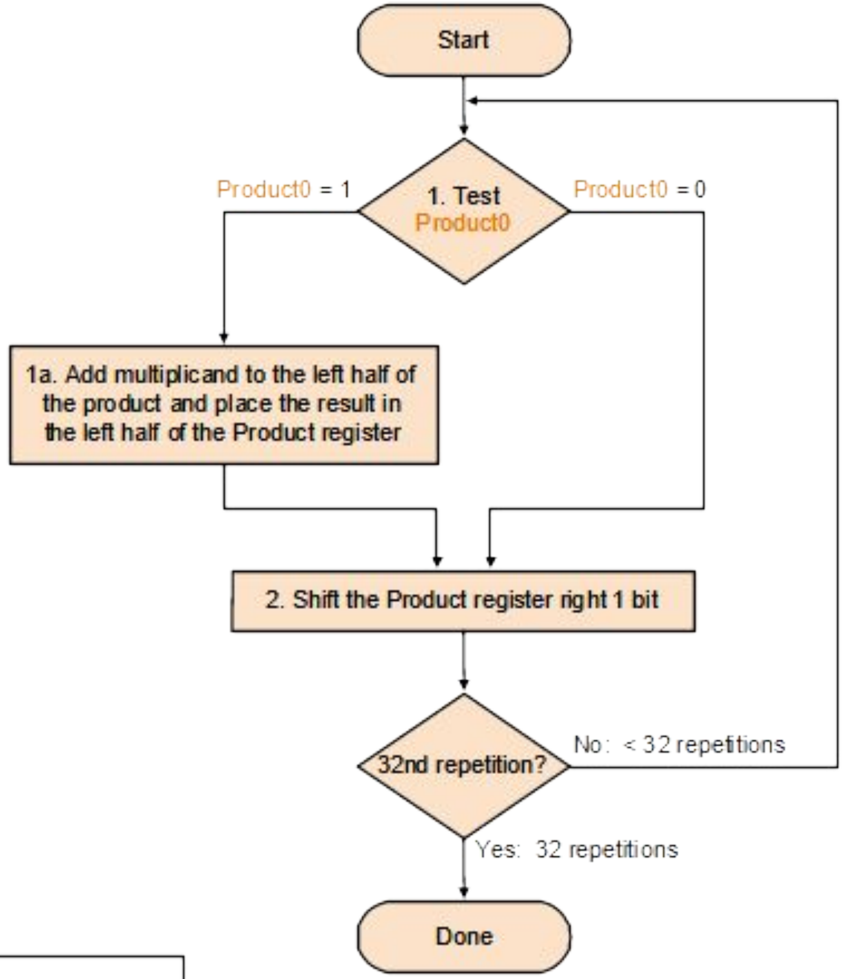
#	Passo	Multiplicando (MD)	Produto (P)
0	Valores iniciais	0010	0000 0011
1	1 => P = P+MD	0010	0010 0011
1	Desloca P à direita	0010	0001 0001
2	1 => P = P+MD	0010	0011 0001
2	Desloca P à direita	0010	0001 1000
3	0 => Não faça nada	0010	0001 1000
3	Desloca P à direita	0010	0000 1100
4			
4			

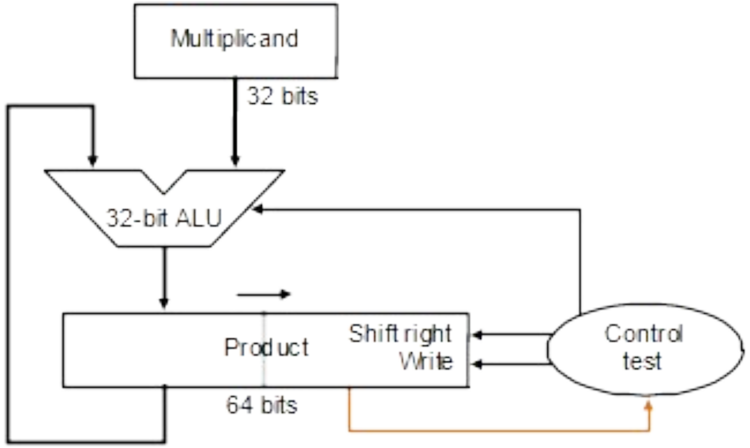




Multiplicação - 3ª versão

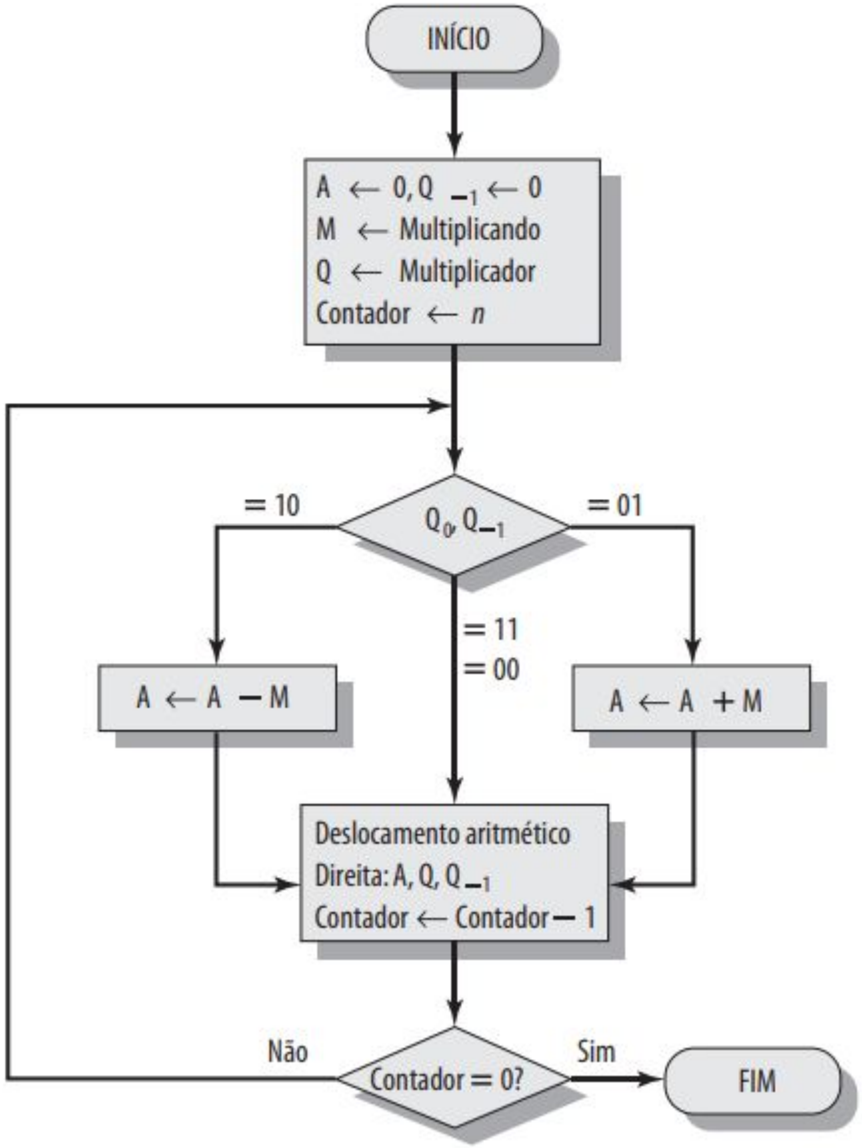
#	Passo	Multiplicando (MD)	Produto (P)
0	Valores iniciais	0010	0000 0011
1	1 => P = P+MD	0010	0010 0011
1	Desloca P à direita	0010	0001 0001
2	1 => P = P+MD	0010	0011 0001
2	Desloca P à direita	0010	0001 1000
3	0 => Não faça nada	0010	0001 1000
3	Desloca P à direita	0010	0000 1100
4	0 => Não faça nada	0010	0000 1100
4	Desloca P à direita	0010	0000 0110





Multiplicação - Algoritmo de Booth

Multiplicação por complemento a 2



Multiplicação - Algoritmo de Booth

Exemplo: 0111 x 0011 (7 x 3)

A	Q	Q_{-1}	M		
0000	0011	0	0111	Valores iniciais	
1001	0011	0	0111	A ← A − M	Primeiro ciclo
1100	1001	1	0111	Deslocamento	
1110	0100	1	0111	Deslocamento	Segundo ciclo
0101	0100	1	0111	A ← A + M	Terceiro ciclo
0010	1010	0	0111	Deslocamento	
0001	0101	0	0111	Deslocamento	Quarto ciclo

Multiplicação - Algoritmo de Booth

Outros exemplos

```
     0111
   × 0011      (0)
 --------
 11111001      1-0
 0000000       1-1
 000111        0-1
 --------
 00010101      (21)
```

(a) (7) × (3) = (21)

```
     0111
   × 1101      (0)
 --------
 11111001      1-0
 0000111       0-1
 111001        1-0
 --------
 11101011      (-21)
```

(b) (7) × (−3) = (−21)

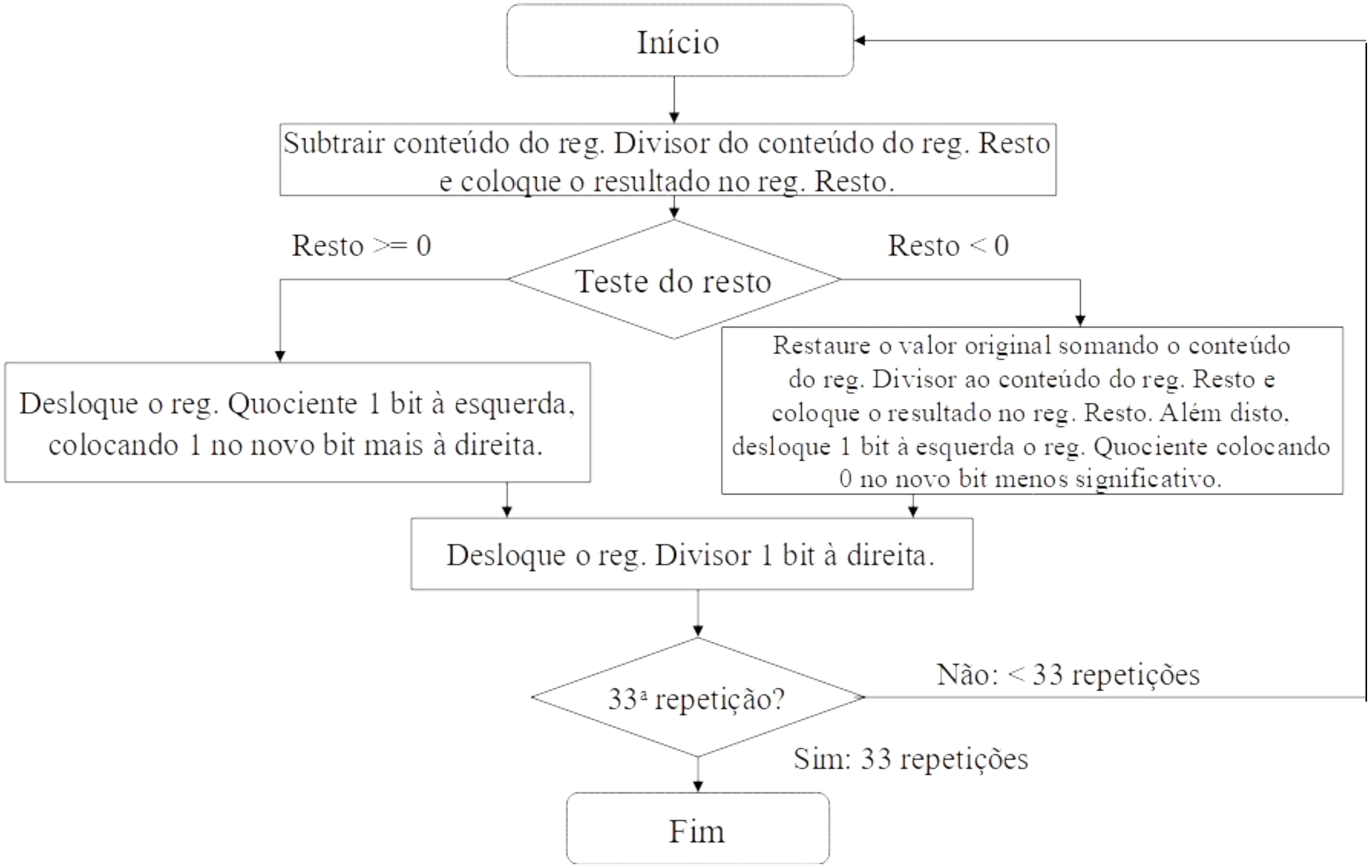
```
     1001
   × 0011      (0)
 --------
 00000111      1-0
 0000000       1-1
 111001        0-1
 --------
 11101011      (-21)
```

(c) (−7) × (3) = (−21)

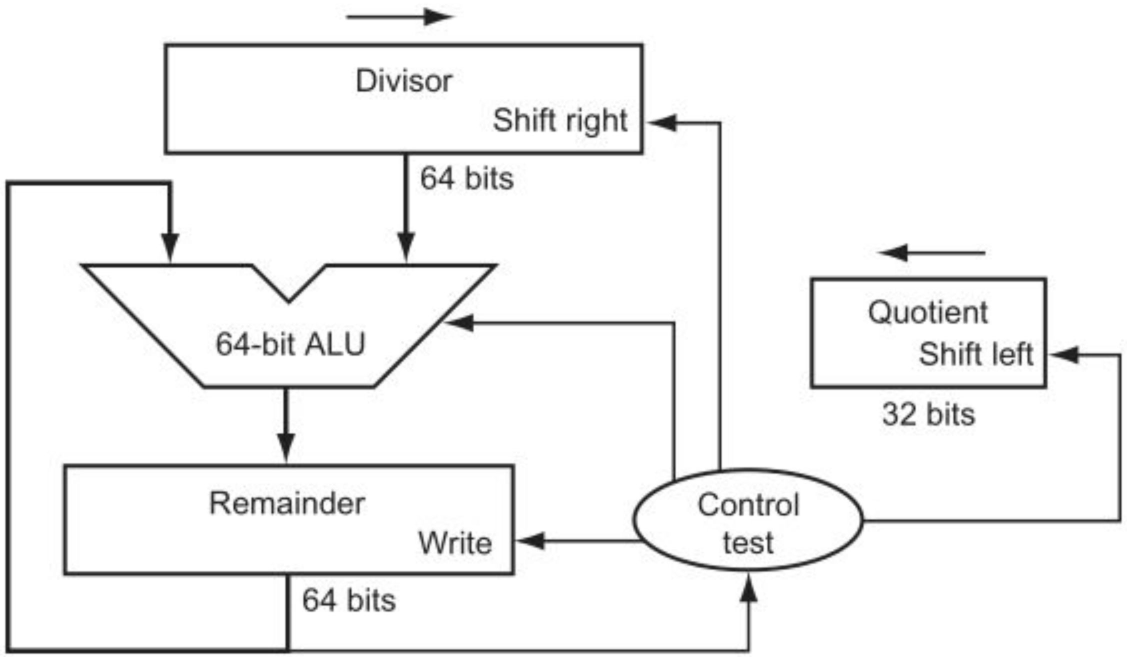
```
     1001
   × 1101      (0)
 --------
 00000111      1-0
 1111001       0-1
 000111        1-0
 --------
 00010101      (21)
```

(d) (−7) × (−3) = (21)

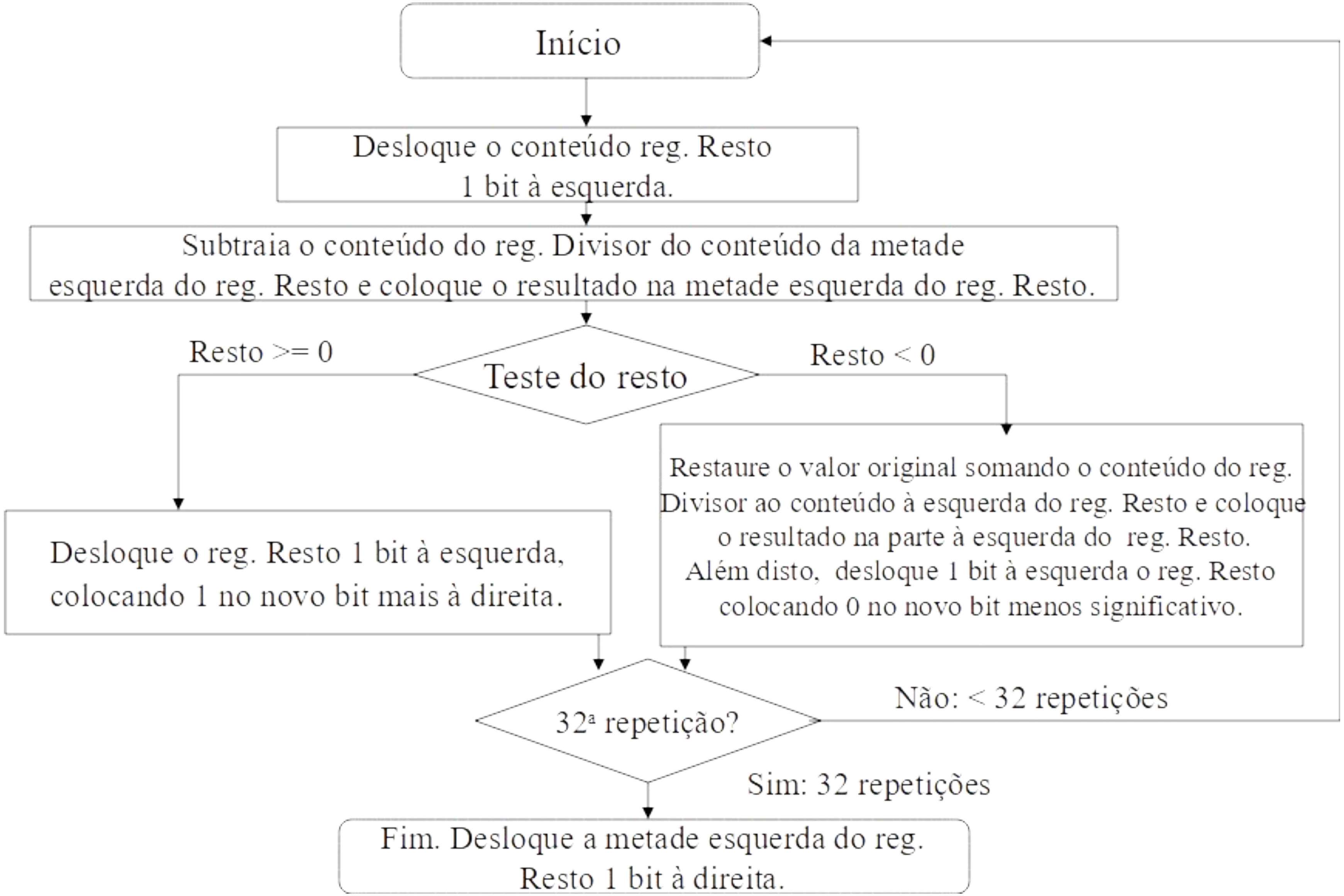
Divisão - 1ª versão



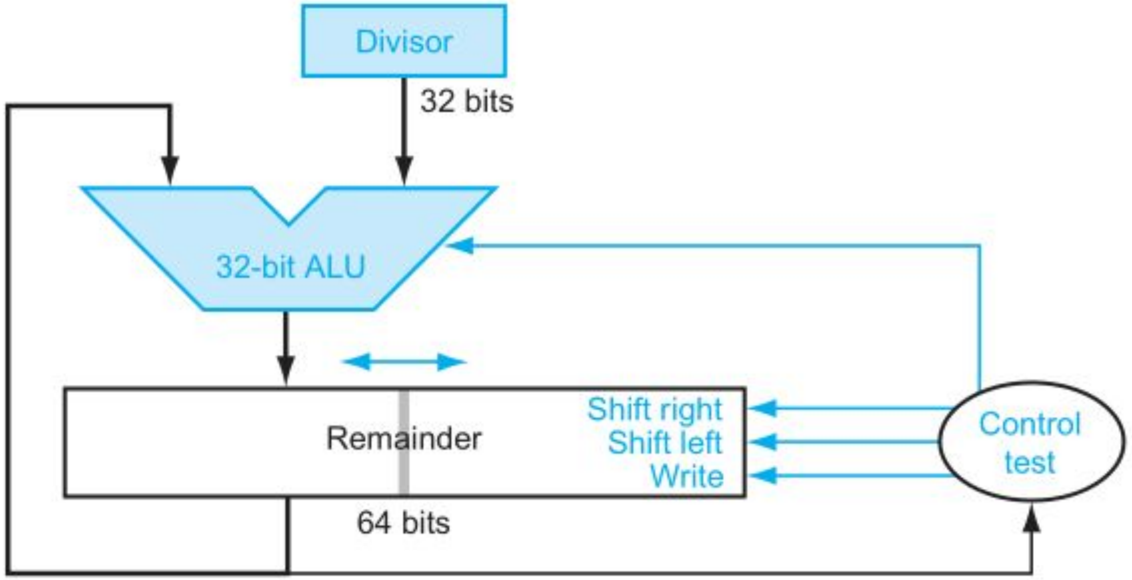
Divisão - 1ª versão



Divisão - 2ª versão



Divisão - 2ª versão



Aritmética Computacional

- Sinais e Complemento
- Adição e Subtração
- Operações Lógicas
- A Unidade Lógica Aritmética
- Multiplicação e Divisão
- Ponto Flutuante

Operações em Ponto Flutuante

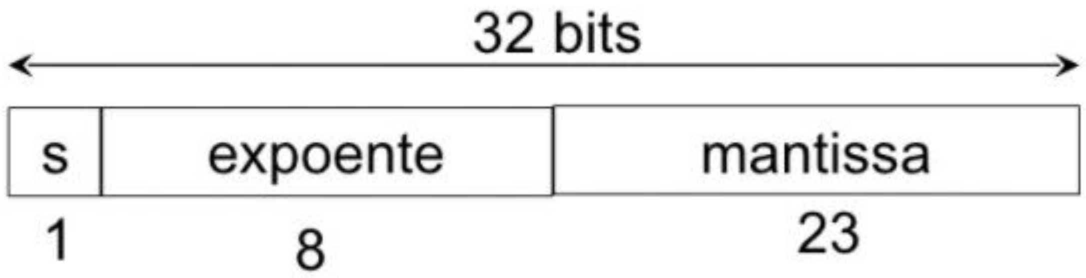
- Suporte a números inteiros com e sem sinal
- Suporte a números fracionários: 3.1414, 0.00001, etc.
- Notação científica: 1.34×10^3
- Números normalizados: 1 dígito antes do ponto decimal
- Números binários também podem ser normalizados
- Ponto decimal / ponto binário
- Aritmética com números normalizados: aritmética de ponto flutuante

- Em C: “float”

- Formato: $1.x \times 2^y$

Representação em Ponto Flutuante

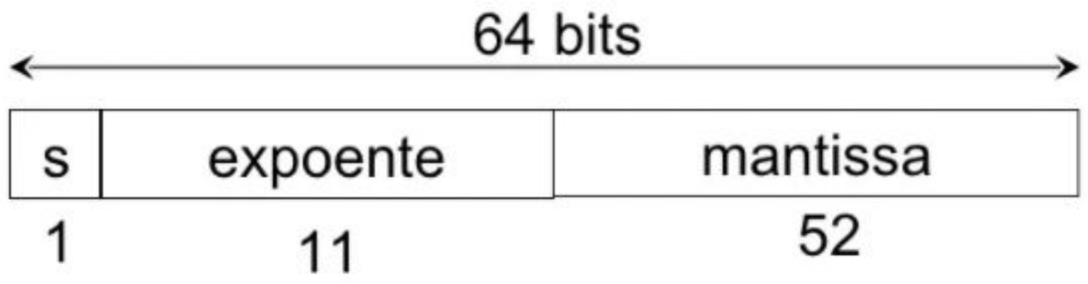
$$N = (-1)^S \times F \times 2^E$$



- S: sinal do número (0 é positivo, 1 é negativo)
- F: mantissa, normalizada
- E: expoente
- Tamanho: -2.0×10^{38} a 2.0×10^{38}

Overflow em Ponto Flutuante

- O expoente é muito grande para ser representado pelos 8 bits!
- Underflow: o módulo do expoente negativo é muito grande
- Necessário números maiores: precisão dupla. (Em C: “double”)



- Tamanho: -2.0×10^{-308} a 2.0×10^{308}

O padrão IEEE 754

- Torna implícito o “1” à esquerda do ponto binário

- Quando o expoente for zero
  - O hardware não considera o primeiro bit “1” implícito
  - Permite-se a representação do número “0” em ponto flutuante

$$N = (-1)^S \times (1+Mantissa) \times 2^E$$

O padrão IEEE 754

- Deve permitir comparações rápidas
- Seria melhor:
  - o menor coeficiente possível valer 00000000_{bin}
  - o maior coeficiente possível valer 11111111_{bin}

- Modficação:
  - Subtrair 127 (peso) do expoente
  - Representação de -1: -1 + 127 = $-1 + 011111111_{bin}$ = 01111110_{bin}
  - +1 = +1 +1257 = 10000000_{bin}

$$N = (-1)^{S} \times (1+Mantissa) \times 2^{(E\ -\ Peso)}$$

- Peso para precisão dupla: 1.023

Exemplo 1

- O número 25.5 em ponto flutuante e <u>precisão simples</u>

1) Converter para algo parecido com $1.x \times 2^y$:

   a) 25.5 / 2 = 12.75
   b) 12.75 / 2 = 6.375
   c) 6.375 / 2 = 3.1875
   d) 3.1875 / 2 = **1.59375**

Foram 4 divisões, logo, 25.5 = **1.59375×2^4**

Exemplo 1

- O número 25.5 em ponto flutuante e <u>precisão simples</u>

2) Mantissa baseada na parte fracionária de **1.59375×2^4**, que é **0.59375**

a) 0.59375 x 2 = **1**.1875
b) 0.1875 x 2 = **0**.375
c) 0.375 x 2 = **0**.75
d) 0.75 x 2 = **1**.5
e) 0.5 x 2 = **1**.0
f) 0.0 x 2 = **0**.0   (Acabou!)

Mantissa calculada: **100110**

Exemplo 1

- O número 25.5 em ponto flutuante e <u>precisão simples</u>

3) Representar em binário (coletar informações)

a) Sinal:

   é positivo, logo, **0** (negativo seria 1)

b) Expoente:

   4 (calculado na primeira etapa)

   Segundo o padrão IEEE, deve ser acrescido de 127

   Logo, **4** + 127 = 131 = 10000011

c) Mantissa (completa-se com 0 até 23 posições):

   10011000000000000000000 (calculada na segunda etapa)

Exemplo 1

- O número 25.5 em ponto flutuante e <u>precisão simples</u>

4) Representar em binário (montagem das informações)

25.5 = **0** **10000011** **100110** **00000000000000000**

Exemplo 2

- Converter a palavra 1 10000001 01000000000000000000 . . . de ponto flutuante para número

$$N = (-1)^S \times (1+Mantissa) \times 2^{(E\ -\ Peso)}$$

$$\Rightarrow \quad (-1)^1 \times (1+0.25) \times 2^{129-127}$$

$$= -1 \times 1.25 \times 2^2$$

$$= -1.25 \times 4$$

$$= -5.0$$