

ULA ou ALU

Introdução

- Representa os músculos do computador: realiza operações aritméticas como a adição e a subtração ou operações lógicas como AND e OR
- A partir de agora mostrar-se-á como construir uma ULA para realizar as quatro operações acima

Operações Lógicas

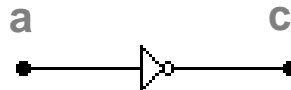
- Em muitas aplicações é necessário processar bits isolados dentro de uma palavra → operações lógicas
- Shifts: deslocamento à esquerda ou à direita

Operações AND e OR

- Operação AND: atua bit a bit, deixando 1 como resultado somente no caso de ambos os bits correspondentes dos operandos serem 1 (aplicação de máscara)
- Operação OR: também atua bit a bit colocando 1 no resultado se qualquer um dos bits correspondentes do operando for 1.

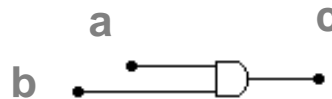
Blocos Construtivos Básicos

NOT (Inversora)
 $c = \overline{a}$



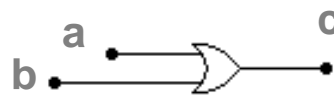
Entrada	Saída
a	c
0	1
1	0

AND
 $c = a.b$



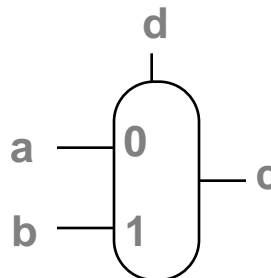
Entradas		Saída
a	b	c
0	0	0
0	1	0
1	0	0
1	1	1

OR
 $c = a + b$



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

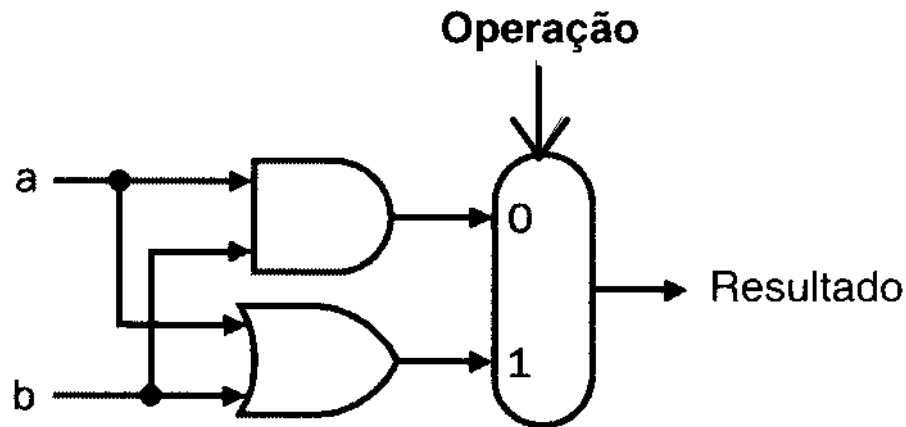
Multiplexador
Se $d=0$, $c=a$ senão $c=b$



Entrada	Saída
d	c
0	a
1	b

Uma ULA de 1 Bit

- Unidade lógica de 1 bit:



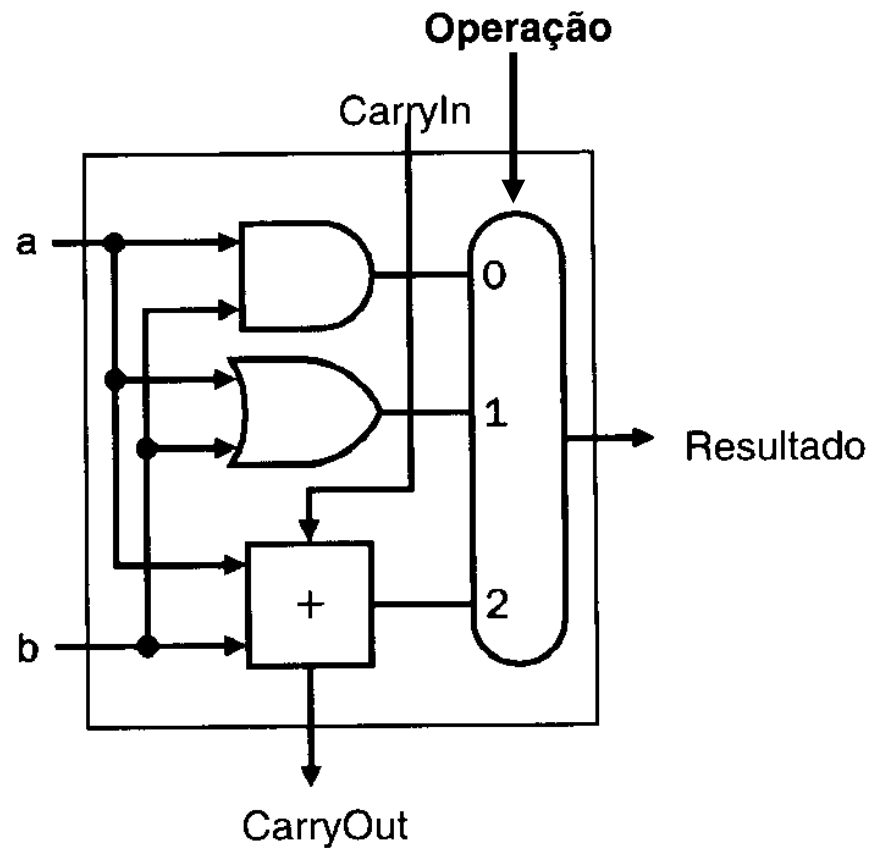
- Operação = 0 ou 1 → AND ou OR.

Somador de 1 bit

- Soma $A + B + \text{“vem 1”}$
- Gera Resultado e “vai um”
- Tabela Verdade:

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	$1+1+1 = 11$

ULA Simples de 1 bit



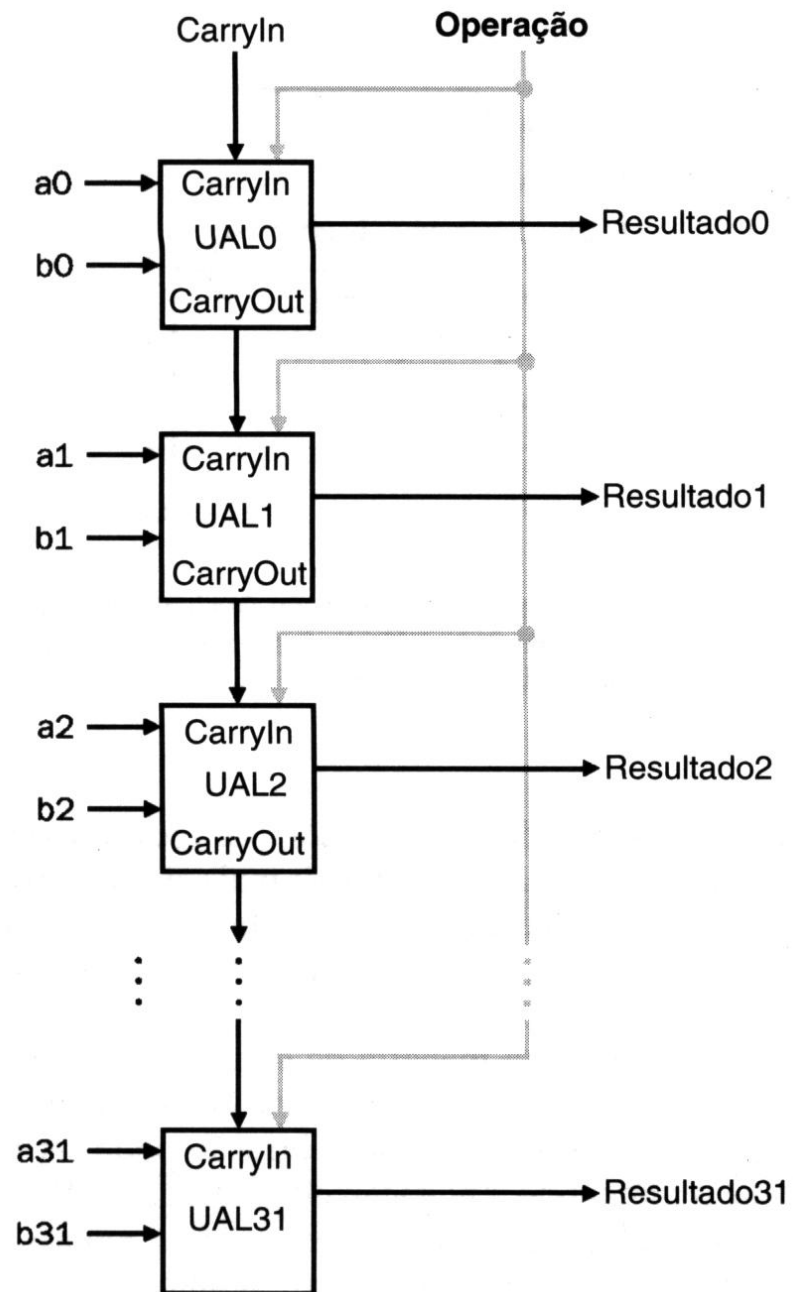
Exercícios

1. Alterar a UAL anterior para que ela gere o valor 0. Dica: a maneira mais fácil é expandir o multiplexador controlado pela linha Operação.
2. Como projetar uma UAL de 32 bits utilizando uma UAL de 1 bit?

1) Adicionar mais uma entrada no MUX ligado em terra para o op code 11 no caso

2) Conectando com os carries e com a chave seletora de operação

ULA de 32 bits

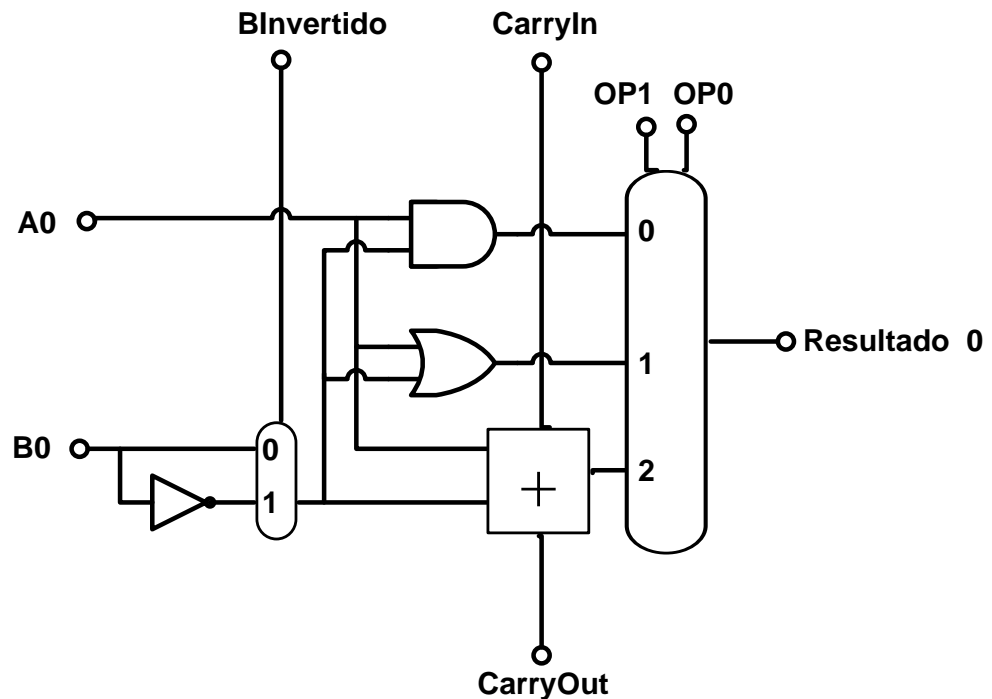


Subtração

- A subtração é obtida somando-se o minuendo ao complemento a 2 do subtraendo, ou seja,

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

- O circuito ao lado inverte o valor de b. Faltava ainda somar 1 ao valor de b invertido.
- Como fazê-lo?



Subtração

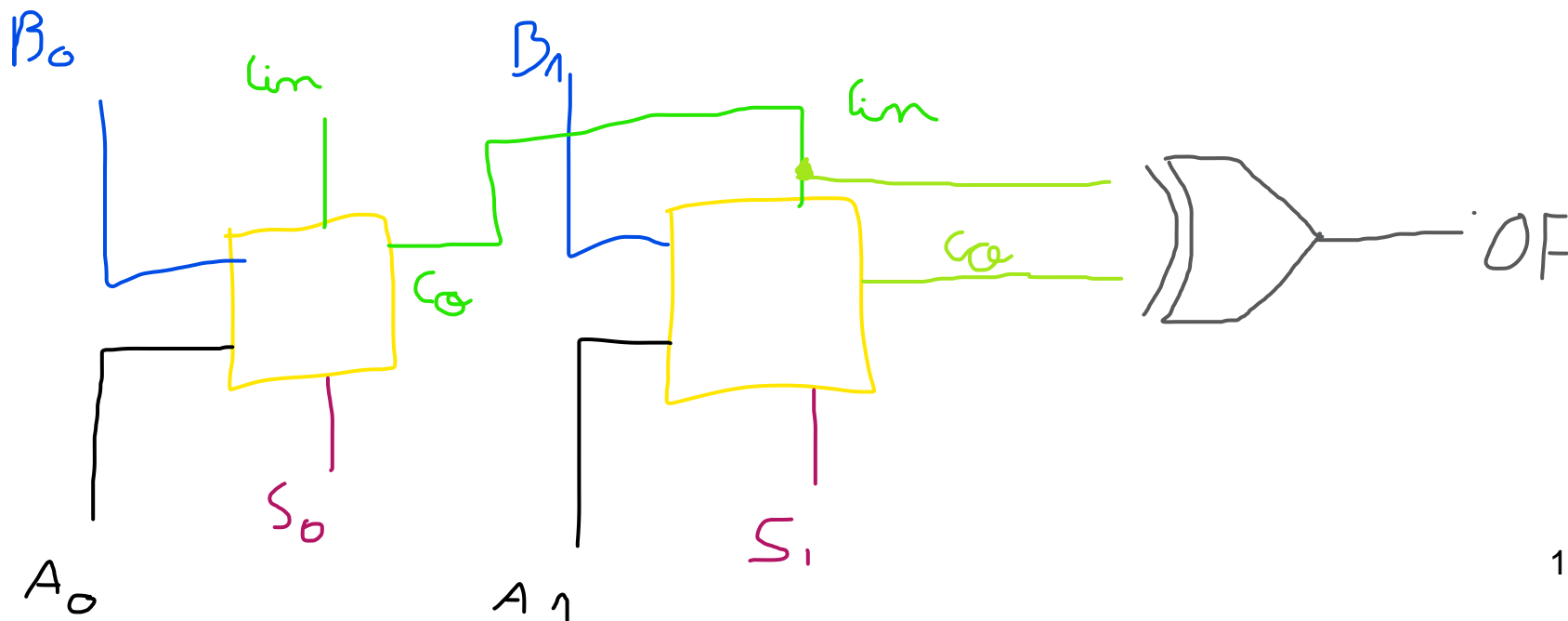
- Na soma o primeiro CarryIn (vem 1) é 0.
- O que acontece se fizermos o primeiro CarryIn = 1?

O valor será somado em um caso de b invert para se obter o complemento de dois a fim de realizar a subtração

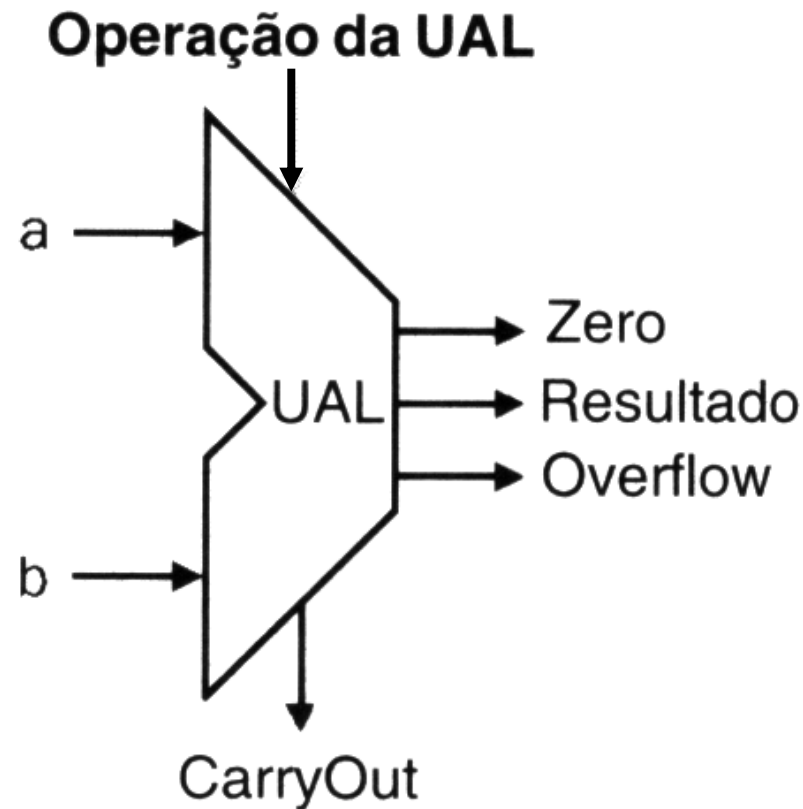
Overflow

- Como fazer a detecção de overflow?

Faça um XOR entre o CarryIn do último somador, que é o CarryOut do penúltimo, e com o seu CarryOut



Símbolo Geral da UAL



Problema

- Qual é o problema de uma UAL projetada como a anterior?

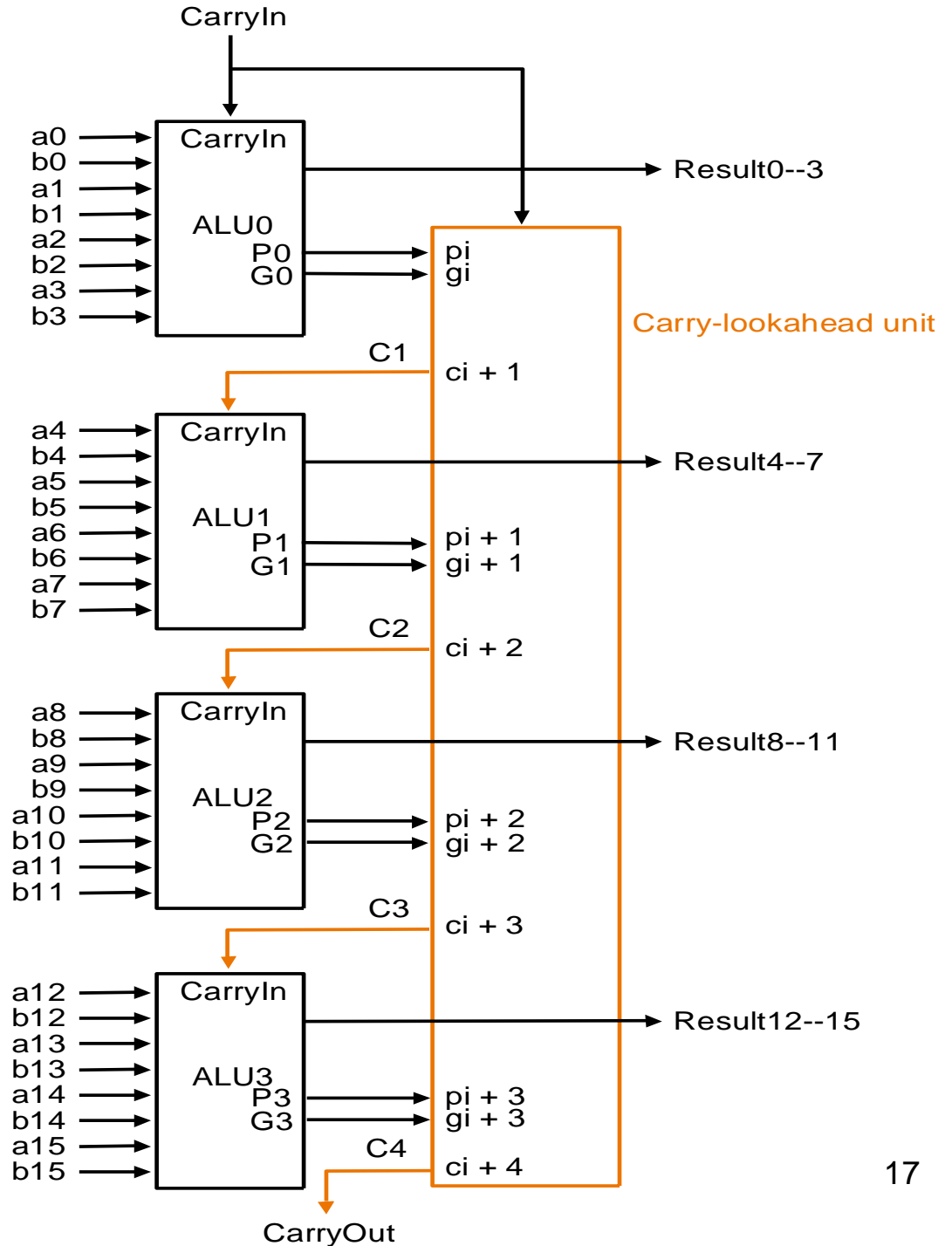
O tempo de espera para o carry se propagar pode gerar um atraso indesejado para obter o resultado de uma operação

Carry Lookahead

- Com que velocidade podemos somar dois operandos de 32 bits?
 - Observe que as entradas **a** e **b** podem ser perfeitamente determinadas a qualquer tempo, mas a entrada CarryIn de um determinado somador de 1 bit depende do resultado da operação realizada no somador de 1 bit vizinho.
 - Solução:
 - Carry Lookahead
 - Propagador e Gerador (conforme visto anteriormente)

Soma:

Conforme visto,
através de CLA.



Multiplicação

Multiplicação: como na prática

$$\begin{array}{r} \text{multiplicando} \quad 0010 \\ \text{multiplicador} \quad \times \quad \underline{0011} \\ 0010 \\ 0010 \\ 0000 \\ 0000 \\ \hline \text{produto} \quad 0000110 \end{array}$$

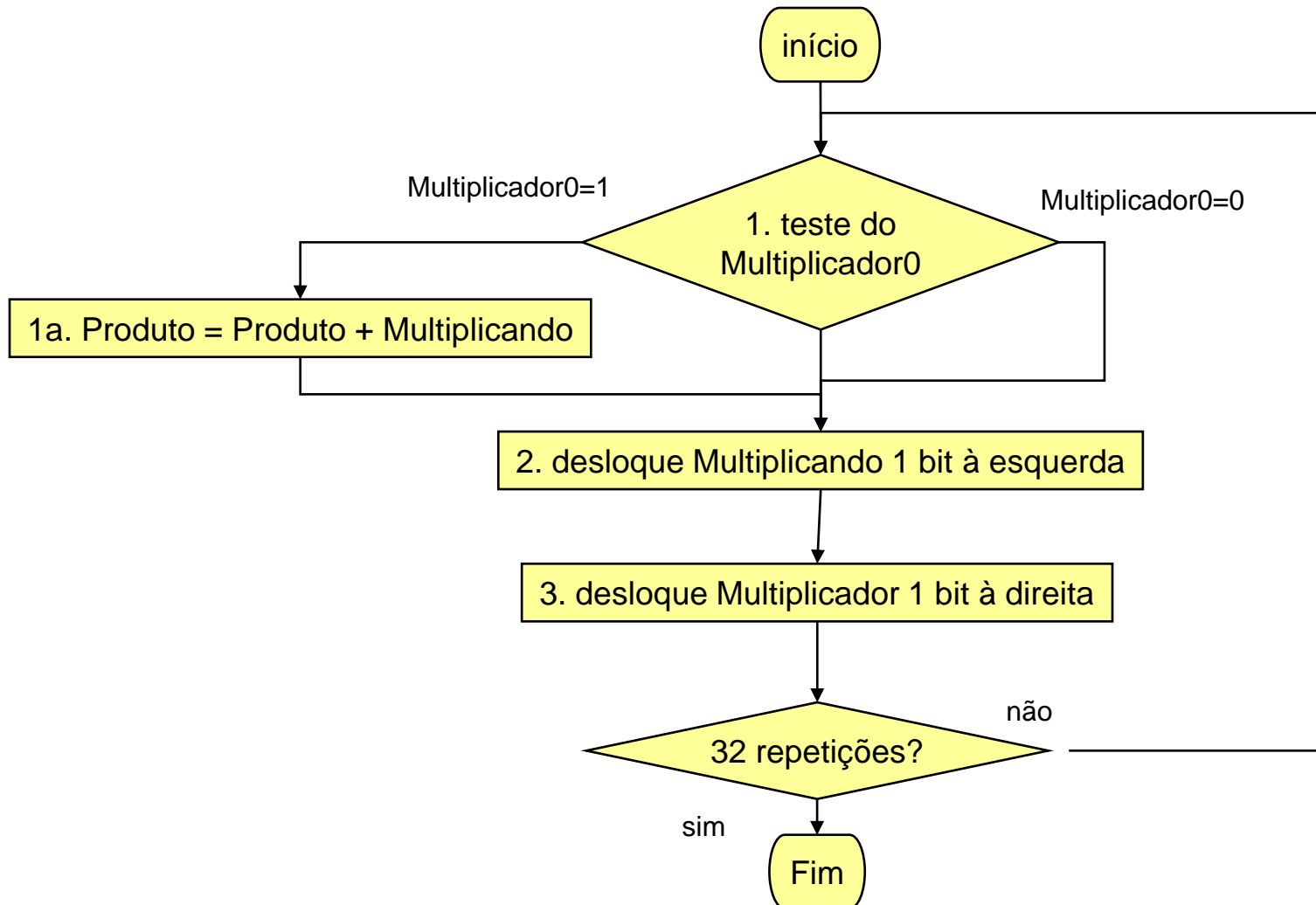
Número de dígitos: multiplicando +
multiplicador.

32 bits x 32 bits = 64 bits.

Algoritmo

- Como na prática
- Simplesmente coloque um cópia do multiplicando ($1 \times$ multiplicando) no lugar apropriado, se o dígito do multiplicando for igual a 1, ou
- Coloque 0 ($0 \times$ 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 (32×32 bits)

Algoritmo: 1ª Versão



Algoritmo: 1ª Versão

```
int palavra = 4;  
int mdor [palavra];  
int mando [2*palavra];  
int produto [2*palavra];
```

Multiplicando = 0000 0010

Multiplicador = 0011

```
int controle = palavra;
```

```
While (controle > 0)  
{  
    if (mdor[0] == 1)  
        produto = produto + mando;  
    mando << 1;  
    mdor >> 1;  
    controle = controle -1;  
}
```

Produto = 0000 0000

Hardware para multiplicação – Versão 1

	Passo	Multiplicador	Multiplicando	Produto
0	Valores iniciais	0011	0000 0010	0000 0000
1	<u> </u> => Prod=Prod+Mcand			
1	Desloca Mcando esq			
1	Desloca Mcador dir			
2				
2				
2				
3				
3				
3				
4				
4				
4				

Hardware para multiplicação – Versão 1

	Passo	Multiplicador	Multiplicando	Produto
0	Valores iniciais	0011	0000 0010	0000 0000
1	1 => Prod=Prod+Mcand	0011	0000 0010	0000 0010
1	Desloca Mcando esq	0011	0000 0100	0000 0010
1	Desloca Mcador dir	0001	0000 0100	0000 0010
2				
2				
2				
3				
3				
3				
4				
4				
4				

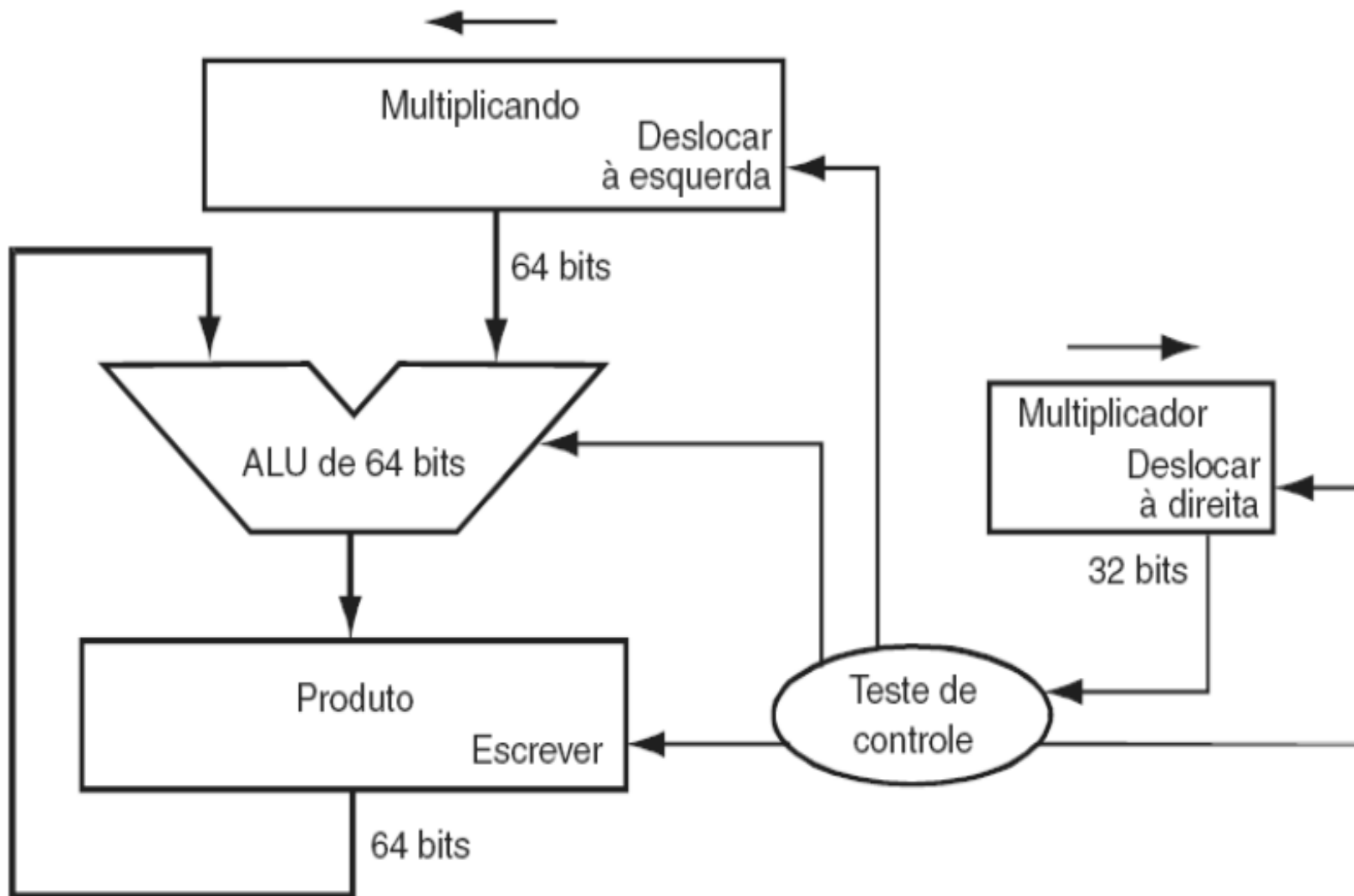
Hardware para multiplicação – Versão 1

	Passo	Multiplicador	Multiplicando	Produto
0	Valores iniciais	0011	0000 0010	0000 0000
1	1 => Prod=Prod+Mcand	0011	0000 0010	0000 0010
1	Desloca Mcando esq	0011	0000 0100	0000 0010
1	Desloca Mcador dir	0001	0000 0100	0000 0010
2	__ => ?			
2	Desloca Mcando esq			
2	Desloca Mcador dir			
3	__ => ?			
3	Desloca Mcando esq			
3	Desloca Mcador dir			
4	__ => ?			
4	Desloca Mcando esq			
4	Desloca Mcador dir			

Hardware para multiplicação – Versão 1

	Passo	Multiplicador	Multiplicando	Produto
0	Valores iniciais	0011	0000 0010	0000 0000
1	1 => Prod=Prod+Mcand	0011	0000 0010	0000 0010
1	Desloca Mcando esq	0011	0000 0100	0000 0010
1	Desloca Mcador dir	0001	0000 0100	0000 0010
2	1 => Prod=Prod+Mcand	0001	0000 0100	0000 0110
2	Desloca Mcando esq	0001	0000 1000	0000 0110
2	Desloca Mcador dir	0000	0000 1000	0000 0110
3	0 => Não Faz Nada	0000	0000 1000	0000 0110
3	Desloca Mcando esq	0000	0001 0000	0000 0110
3	Desloca Mcador dir	0000	0001 0000	0000 0110
4	0 => Não Faz Nada	0000	0001 0000	0000 0110
4	Desloca Mcando esq	0000	0010 0000	0000 0110
4	Desloca Mcador dir	0000	0010 0000	0000 0110

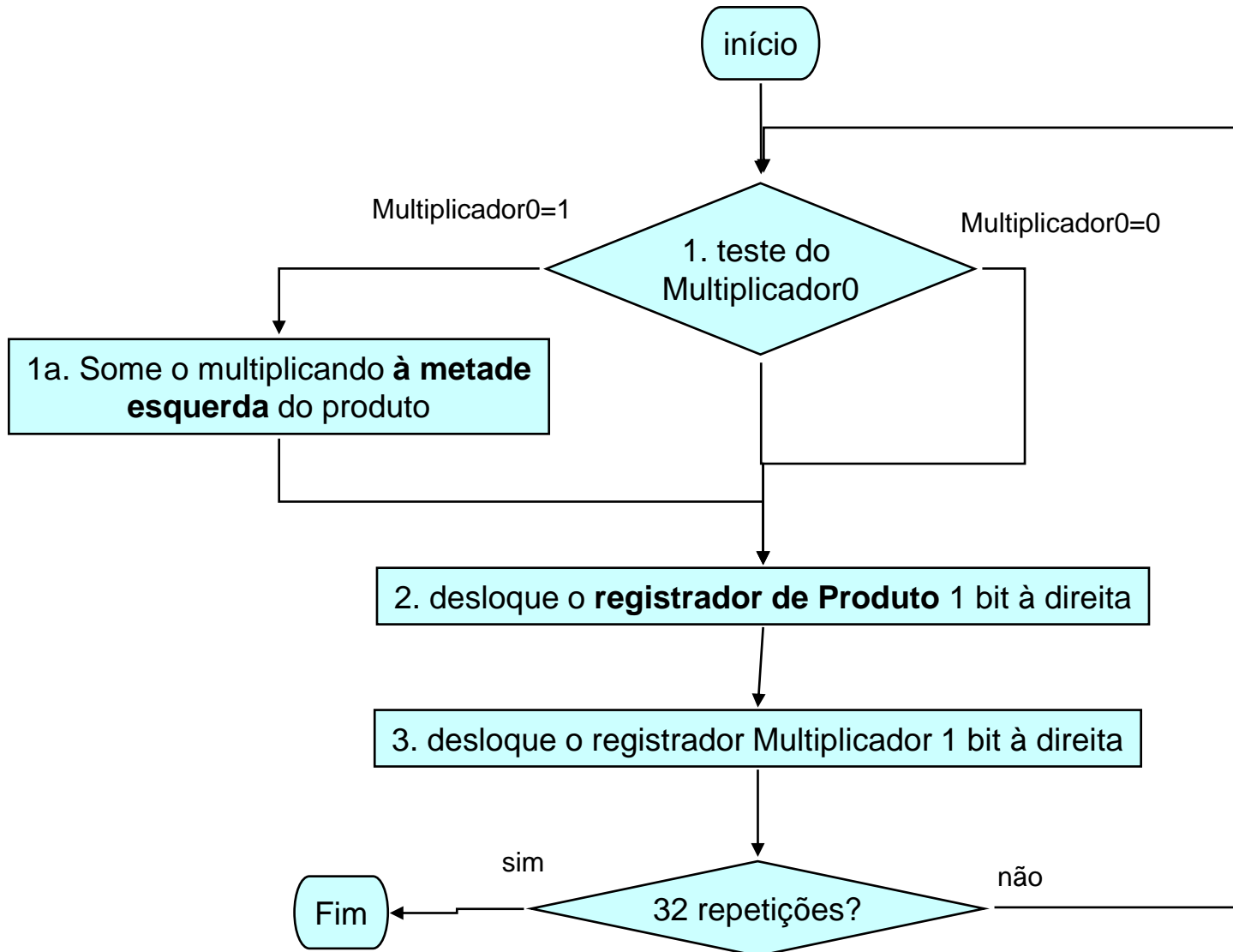
Hardware para multiplicação – Versão 1



Desvantagens

- UAL de 64 bits.
- 2 registradores de 64 bits
- Próxima versão:
 - Metade dos bits do multiplicando da primeira versão são sempre zero, de modo que somente metade deles poderia conter informações úteis. A segunda versão utiliza-se desta informação para melhorar a performance da multiplicação.

Algoritmo: 2ª Versão



Hardware para multiplicação – Versão 2

```
int palavra = 4;  
int mdor [palavra];  
int mando [palavra];  
int produto [2*palavra];
```

Multiplicando = 0010

Multiplicador = 0011

```
int controle = palavra;
```

Produto = 0000 0000

```
While (controle > 0)  
{  
    if (mdor[0] == 1)  
        produto [esq]= produto[esq] + mando;  
    produto >> 1;  
    mdor >> 1;  
    controle = controle -1;  
}
```

Hardware para multiplicação – Versão 2

	Passo	Multiplicador	Multiplicando	Produto
0	Valores iniciais	0011	0010	0000 0000
1				
1				
1				
2				
2				
2				
3				
3				
3				
4				
4				
4				

Hardware para multiplicação – Versão 2

	Passo	Multiplicador	Multiplicando	Produto
0	Valores iniciais	001 1	0010	0000 0000
1	1 => Prod=Prod+Mcand			
1	Desloca Produto dir			
1	Desloca Mcador dir			
2				
2				
2				
3				
3				
3				
4				
4				
4				

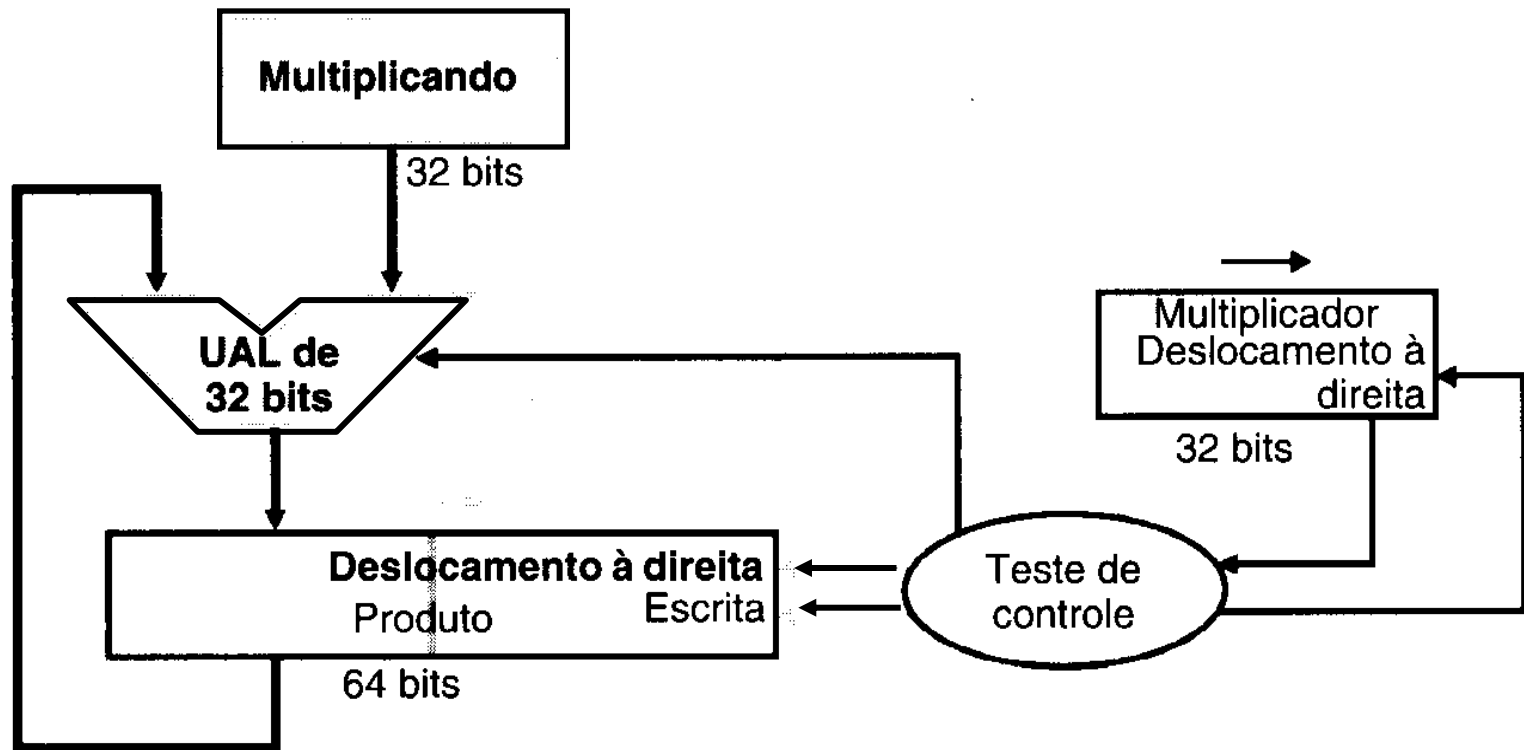
Hardware para multiplicação – Versão 2

	Passo	Multiplicador	Multiplicando	Produto
0	Valores iniciais	001 1	0010	0000 0000
1	1 => Prod=Prod+Mcand	0011	0010	0010 0000
1	Desloca Produto dir	0011	0010	0001 0000
1	Desloca Mcador dir	0001	0010	0001 0000
2	_ => Prod=Prod+Mcand			
2	Desloca Produto dir			
2	Desloca Mcador dir			
3	_ => Não Faz Nada			
3	Desloca Produto dir			
3	Desloca Mcador dir			
4	_ => Não Faz Nada			
4	Desloca Produto dir			
4	Desloca Mcador dir			

Hardware para multiplicação – Versão 2

	Passo	Multiplicador	Multiplicando	Produto
0	Valores iniciais	001 1	0010	0000 0000
1	1 => Prod=Prod+Mcand	0011	0010	0010 0000
1	Desloca Produto dir	0011	0010	0001 0000
1	Desloca Mcador dir	000 1	0010	0001 0000
2	1 => Prod=Prod+Mcand	0001	0010	0011 0000
2	Desloca Produto dir	0001	0010	0001 1000
2	Desloca Mcador dir	0000	0010	0001 1000
3	0 => Não Faz Nada	0000	0010	0001 1000
3	Desloca Produto dir	0000	0010	0000 1100
3	Desloca Mcador dir	0000	0010	0000 1100
4	0 => Não Faz Nada	0000	0010	0000 1100
4	Desloca Produto dir	0000	0010	0000 0110
4	Desloca Mcador dir	0000	0010	0000 0110

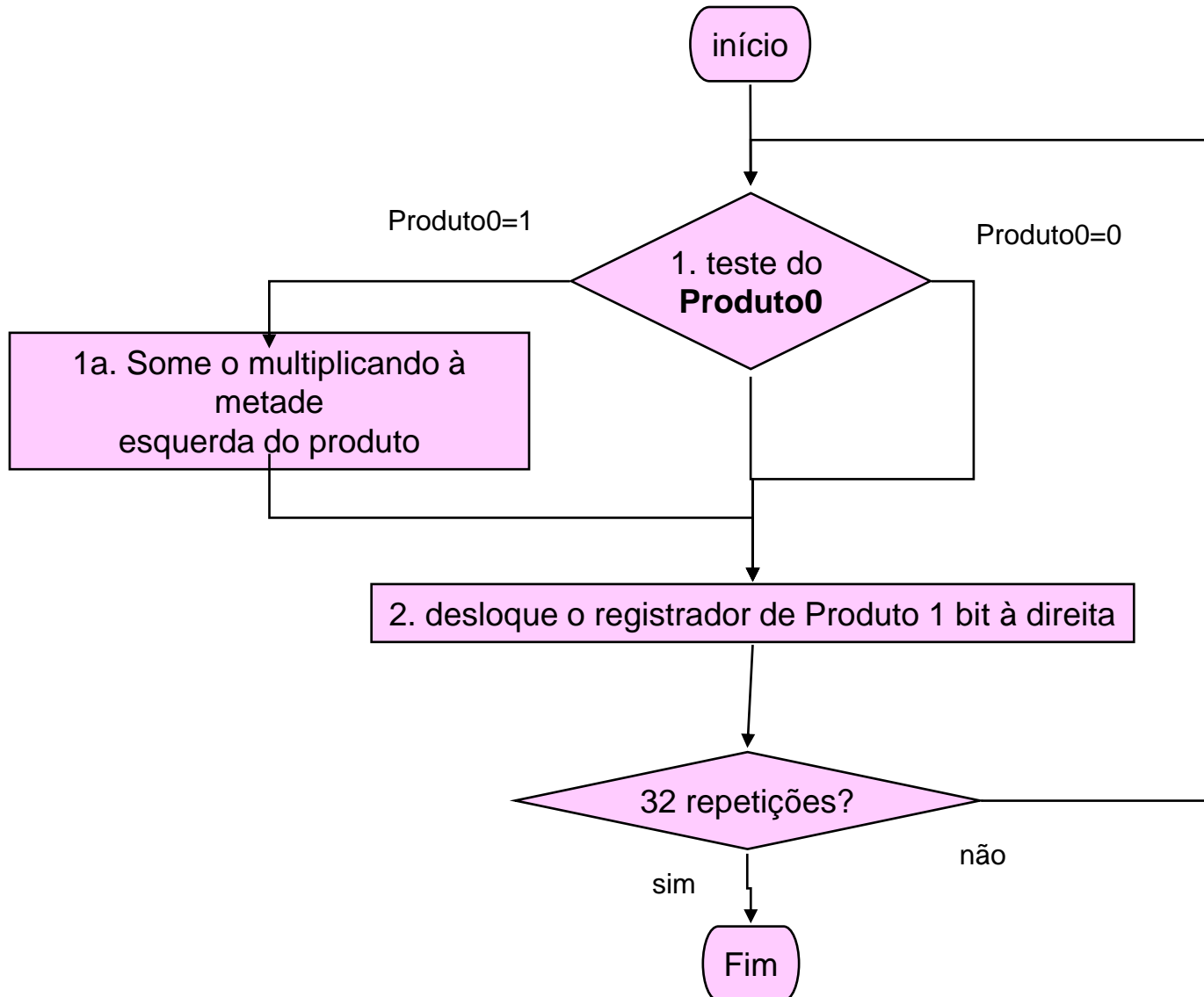
Hardware: 2ª Versão



Versão Final do Algoritmo de Multiplicação

- O registrador reservado ao produto desperdiça tanto espaço quanto o do multiplicador: à medida que o desperdício de espaço do produto se reduzia, a mesma coisa acontecia com o multiplicador.

Algoritmo: 3ª Versão



Hardware para multiplicação – Versão 3

```
int palavra = 4;  
int mando [palavra];  
int produto [2*palavra];
```

```
int controle = palavra;
```

```
produto [dir] = mdor;
```

```
While (controle > 0)
```

```
{  
    if (produto[0] == 1)  
        produto [esq]= produto[esq] + mando;  
    produto >> 1;  
    controle = controle -1;  
}
```

Multiplicando = 0010

Multiplicador = 0011

Passo inicial

Copie o Multiplicador
na metade dir. do
produto

Produto = 0000 0011

	Passo	Multiplicando	Produto
0	Valores iniciais	0010	0000 0011
1	1 => Prod=Prod+Mcand		
1	Desloca Produto dir		
2			
2			
3			
3			
4			
4			

Hardware para multiplicação – Versão 3

	Passo	Multiplicando	Produto
0	Valores iniciais	0010	0000 001 1
1	1 => Prod=Prod+Mcand	0010	0010 0011
1	Desloca Produto dir	0010	0001 0001
2			
2			
3			
3			
4			
4			

Hardware para multiplicação – Versão 3

	Passo	Multiplicando	Produto
0	Valores iniciais	0010	0000 001 ¹
1	¹ => Prod=Prod+Mcand	0010	0010 0011
1	Desloca Produto dir	0010	0001 000 ¹
2	<u> </u> => Prod=Prod+Mcand		
2	Desloca Produto dir		
3	<u> </u> => Não Faz Nada		
3	Desloca Produto dir		
4	<u> </u> => Não Faz Nada		
4	Desloca Produto dir		

Hardware para multiplicação – Versão 3

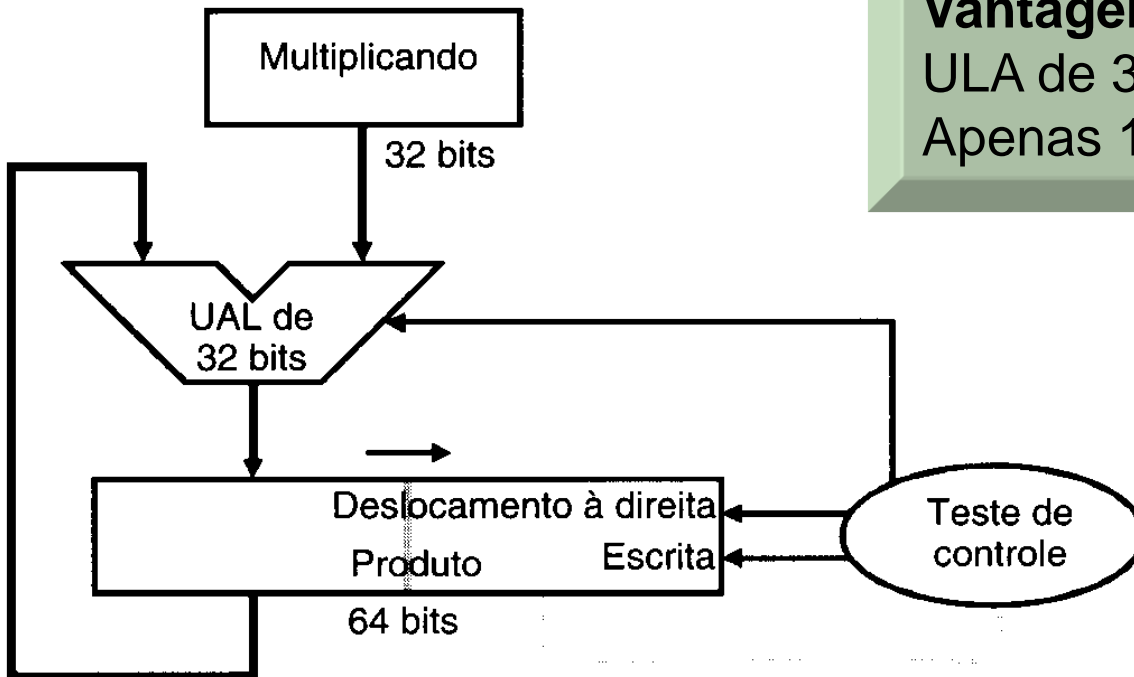
	Passo	Multiplicando	Produto
0	Valores iniciais	0010	0000 001 1
1	1 => Prod=Prod+Mcand	0010	0010 0011
1	Desloca Produto dir	0010	0001 000 1
2	1 => Prod=Prod+Mcand	0010	0011 0001
2	Desloca Produto dir	0010	0001 1000
3	0 => Não Faz Nada	0010	0001 1000
3	Desloca Produto dir	0010	0000 1100
4	0 => Não Faz Nada	0010	0000 1100
4	Desloca Produto dir	0010	0000 0110

Hardware: 3ª Versão

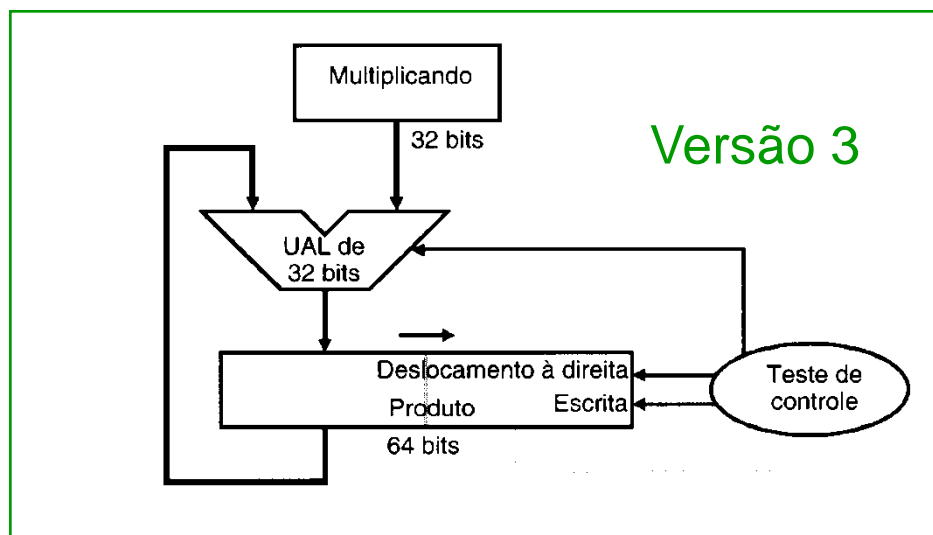
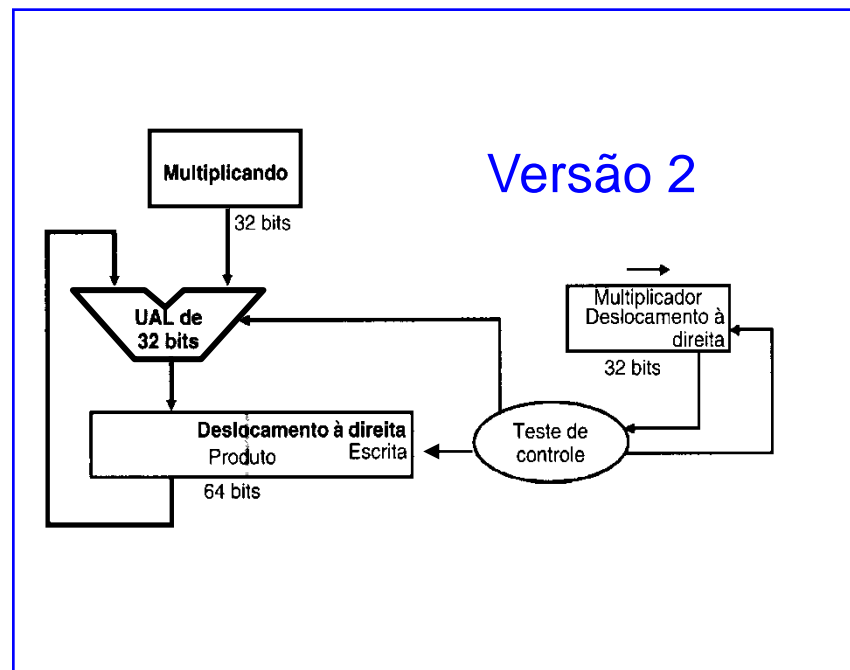
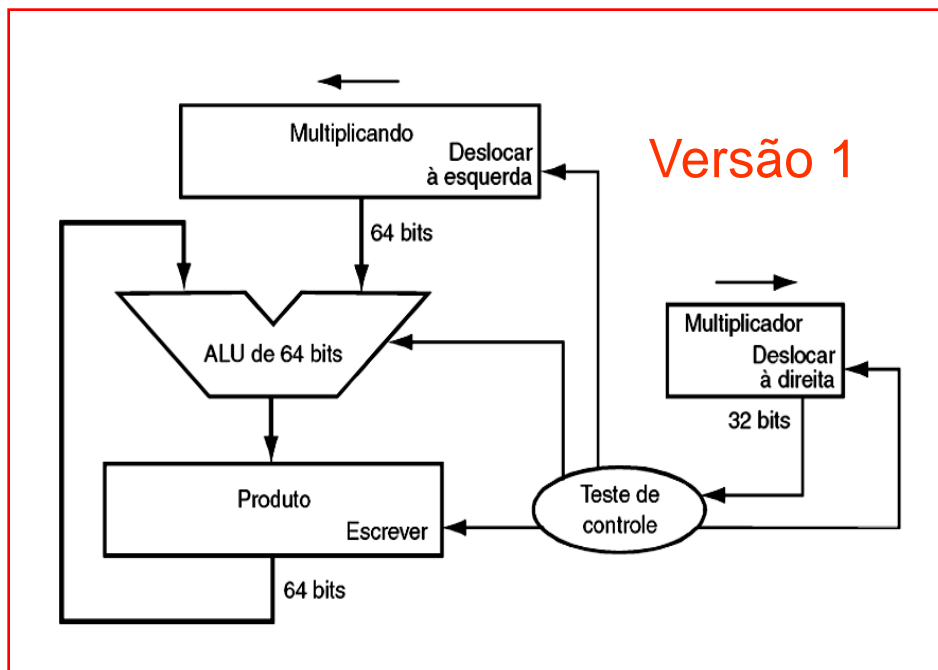
Vantagens:

ULA de 32 bits.

Apenas 1 registrador de 64 bits.



Comparativo do Hardware para multiplicação



Algoritmo de Booth

Pesquisar

Multiplicação Paralela

$$\begin{array}{cccccccl} & a_5 & a_4 & a_3 & a_2 & a_1 & a_0 & = A \\ x & b_5 & b_4 & b_3 & b_2 & b_1 & b_0 & = B \end{array}$$

$$\begin{array}{cccccccc}
 & & & & a5b0 & a4b0 & a3b0 & a2b0 & a1b0 & a0b0 & = W1 \\
 & & & & & & & & & & \\
 & & & & & a5b1 & a4b1 & a3b1 & a2b1 & a1b1 & a0b1 & = W2 \\
 & & & & & & & & & & \\
 & & & & & & a5b2 & a4b2 & a3b2 & a2b2 & a1b2 & a0b2 & = W3 \\
 & & & & & & & & & & \\
 & & & & & a5b3 & a4b3 & a3b3 & a2b3 & a1b3 & a0b3 & = W4 \\
 & & & & & & & & & & \\
 & & & & & a5b4 & a4b4 & a3b4 & a2b4 & a1b4 & a0b4 & = W5 \\
 & & & & & & & & & & \\
 a5b5 & a4b5 & a3b5 & a2b5 & a1b5 & a0b5 & & & & & = W6
 \end{array}$$

P11 P10 P9 P8 P7 P6 P5 P4 P3 P2 P1 P0 = $A \times B = P$

Multiplicação Paralela

Exercício:

Construir um multiplicador paralelo no logisim capaz de multiplicar números de 3 bits.

$$\begin{array}{rclcl} a_2 & a_1 & a_0 & = & A \\ b_2 & b_1 & b_0 & = & B \end{array}$$

$$P_5 \quad P_4 \quad P_3 \quad P_2 \quad P_1 \quad P_0 = A \times B = P$$

Representação ponto flutuante

- Formato



- s é o bit de sinal
 - exp é usado para obter E
 - $frac$ é usado para obter M
- $s = 0$ positivo $s = 1$ negativo

- Valor representado

$$(-1)^s M 2^E$$

- Significando M é um valor fracionário no intervalo $[1.0, 2.0)$, para números normalizados e $[0 e 1)$ para números denormalizados
- Exponente E fornece o peso em potência de dois

Valores numéricos Normalizados

- Condição **$\text{exp} \neq 000\dots 0$ e $\text{exp} \neq 111\dots 1$**
- Expoente codificado como valor polarizado (*biased*)
 $E = \text{exp} - \text{bias}$
 - exp : valor não sinalizado
 - bias : valor da polarização
 - Precisão Simples: 127 (exp : 1...254, E : -126...127)
 - Precisão dupla: 1023 (exp : 1...2046, E : -1022...1023)
 - Em geral: $\text{bias} = 2^{e-1} - 1$, onde e é o número de bits do expoente
- Significando codificado com bit 1 mais significativo (leading bit) implícito
 $M = 1.\text{xxx}\dots\text{x}_2$
 - **$\text{xxx}\dots\text{x}$** : bits da *frac*
 - Mínimo quando 000...0 ($M = 1.0$)
 - Máximo quando 111...1 ($M = 2.0 - \varepsilon$)
 - O bit extra (leading bit 1) é obtido “implicitamente”

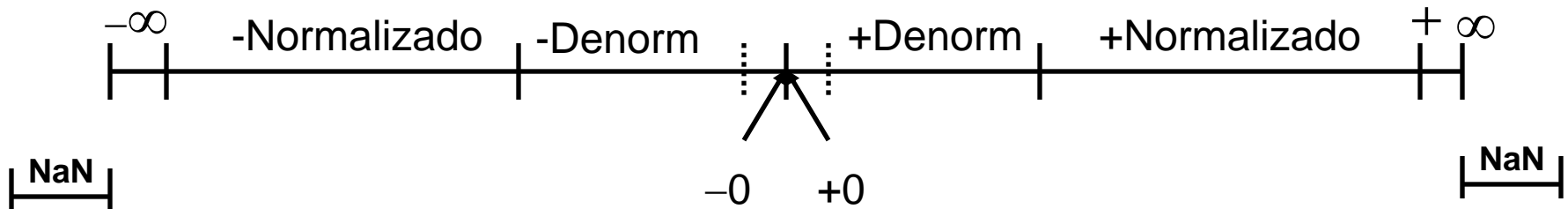
Valores denormalizados

- Condição **$\text{exp} = 000\dots 0$**
- Valor
 - Valor do Expoente $E = -\text{Bias} + 1$
 - Valor do Significando $M = 0.\text{xxx}\dots\text{x}_2$
 - $\text{xxx}\dots\text{x}$: bits de frac
- Casos
 - **$\text{exp} = 000\dots 0, \text{frac} = 000\dots 0$**
 - Representa valor 0
 - Nota-se que existem valores distintos $+0$ e -0
 - **$\text{exp} = 000\dots 0, \text{frac} \neq 000\dots 0$**
 - Numeros muito próximos de 0.0
 - Perde precisão à medida que vai diminuindo
 - “ underflow gradual”

Valores especiais

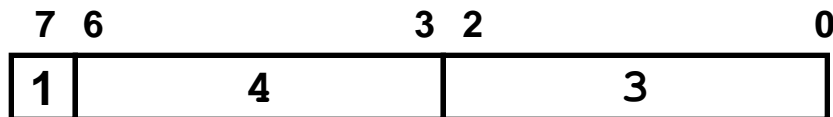
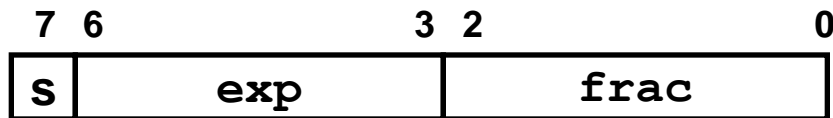
- Condição **exp = 111...1**
- Casos
 - **exp = 111...1, frac = 000...0**
 - Representa valor ∞ (infinito)
 - Operação que transborda (overflow)
 - Ambos positivo e negativo
 - P. ex., $1.0/0.0 = -1.0/-0.0 = +\infty$, $1.0/-0.0 = -\infty$
 - **exp = 111...1, frac \neq 000...0**
 - Not-a-Number (NaN)
 - Nenhum valor numérico pode ser determinado
 - P. ex., $\text{sqrt}(-1)$, $\infty - \infty$

Resumo da codificação de números reais em ponto flutuante

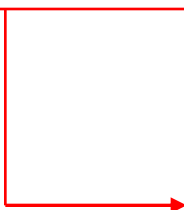


Representação ilustrativa de 8 bits

- Representação ponto flutuante de 8 bits
 - O bit de sinal é o bit mais significativo.
 - Os seguintes quatro bits são expoente, com bias de 7.
 - Os últimos três bits são *frac*
- Semelhante a forma geral no formato IEEE
 - normalizado, denormalizado
 - Representação de 0, NaN, infinito



Valores Relativos ao Expoente

$$\text{Bias} = 2^{(4-1)} - 1$$


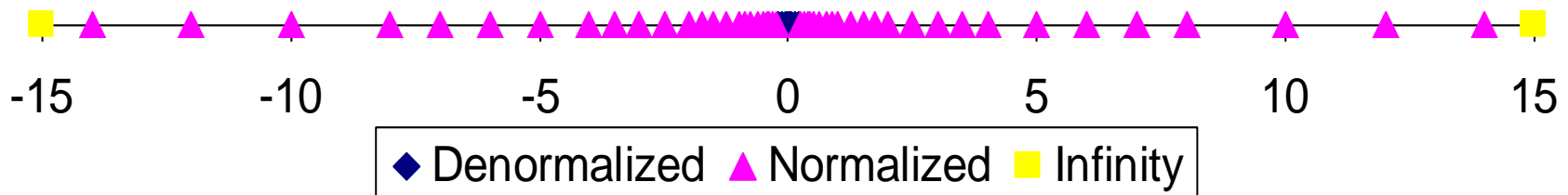
	exp	E	2^E	
0	0000	-6	1/64	(denorms)
1	0001	-6	1/64	
2	0010	-5	1/32	
3	0011	-4	1/16	
4	0100	-3	1/8	
5	0101	-2	1/4	
6	0110	-1	1/2	
7	0111	0	1	
8	1000	+1	2	
9	1001	+2	4	
10	1010	+3	8	
11	1011	+4	16	
12	1100	+5	32	
13	1101	+6	64	
14	1110	+7	128	
15	1111	n/a		(inf, NaN)

Intervalo

	s	exp	frac	E	Valor	
números denormalizados...	0	0000	000	-6	0	
	0	0000	001	-6	$1/8 * 1/64 = 1/512$	← Mais perto de zero
	0	0000	010	-6	$2/8 * 1/64 = 2/512$	
	0	0000	110	-6	$6/8 * 1/64 = 6/512$	
	0	0000	111	-6	$7/8 * 1/64 = 7/512$	← maior denorm
	0	0001	000	-6	$8/8 * 1/64 = 8/512$	← menor norm
	0	0001	001	-6	$9/8 * 1/64 = 9/512$	
	...					
	0	0110	110	-1	$14/8 * 1/2 = 14/16$	
	0	0110	111	-1	$15/8 * 1/2 = 15/16$	← perto de 1 abaixo
números Normalizados	0	0111	000	0	$8/8 * 1 = 1$	
	0	0111	001	0	$9/8 * 1 = 9/8$	← perto de 1 acima
	0	0111	010	0	$10/8 * 1 = 10/8$	
	...					
	0	1110	110	7	$14/8 * 128 = 224$	
	0	1110	111	7	$15/8 * 128 = 240$	← maior norm
	0	1111	000	n/a	inf	

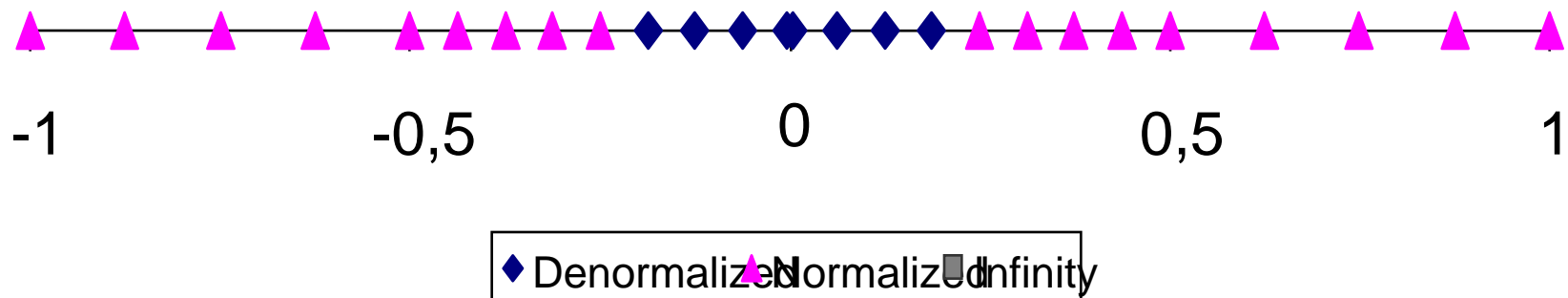
Distribuição de valores

- Formato de 6-bits tipo IEEE
 - $e = 3$ bits de expoente
 - $f = 2$ bits de Mantissa
 - bias $e' = 3$
- Notar como a distribuição fica mais densa perto de zero.



Distribuição de Valores perto de zero

- Formato de 6-bits, tipo IEEE
 - $e = 3$ bits de expoente
 - $f = 2$ bits de fração
 - Bias igual a 3



Multiplicação em FP

Operandos

$$(-1)^{s1} M1 2^{E1} * (-1)^{s2} M2 2^{E2}$$

Resultado exato

$$(-1)^s M 2^E$$

Sinal s : $s1 \text{ xor } s2$

Significando M : $M1 * M2$

Expoente E : $E1 + E2$

Representação final

se $M \geq 2$, deslocar à direita M , incrementar E

se E fora do intervalo, overflow

Arredonda M para caber em `frac`

Multiplicação em FP

Operandos

$$(-1)^{s1} M1 2^{E1} * (-1)^{s2} M2 2^{E2}$$

Resultado exato

$$(-1)^s M 2^E$$

Sinal s : $s1 \text{ xor } s2$

Significando M : $M1 * M2$

Expoente E : $E1 + E2$

Resolver:

$$-0,375 * 104$$

Usar IEEE754

$E = 4$ bits e $M = 3$ bits

Representação final

se $M \geq 2$, deslocar à direita M , incrementar E

se E fora do intervalo, overflow

Arredonda M para caber em `frac`

Solução

$$0,375 * 104$$

$$E=4 \Rightarrow \text{Bias} = 7$$

1) Passar para norma IEEE 754

$$0.375_{(10)} = 0.011_{(2)} \Rightarrow 1.1 \times 2^{-2} \Rightarrow 00101100$$

$$104_{(10)} = 1101000_{(2)} \Rightarrow 1.101 \times 2^6 \Rightarrow 01101101$$

2) Sinal =

3) M =

4) E =

5) M ≥ 2 ?

6) Arredonda M

Resultado Final \Rightarrow

Solução

$$0,375 * 104$$

$$E=4 \Rightarrow \text{Bias} = 7$$

1) Passar para norma IEEE 754

$$0.375_{(10)} = 0.011_{(2)} \Rightarrow 1.1 \times 2^{-2} \Rightarrow 00101100$$

$$104_{(10)} = 1101000_{(2)} \Rightarrow 1.101 \times 2^6 \Rightarrow 01101101$$

2) Sinal = XOR(0,0) = 0 (positivo)

3) $M = 1.100 * 1.1101 = 10.011100$

4) $E = -2 + 6 = 4$

5) $M \geq 2$ desloca para direita e incrementa Expoente
 $M=1.00111$ e $E=4+1=5$

6) Arredonda M para tamanho correto : $M=1.001$

Resultado Final $\Rightarrow 01100001$

Adição FP

Operandos

$$(-1)^{s1} M1 2^{E1}$$

$$(-1)^{s2} M2 2^{E2}$$

Assumir $E1 > E2$

Resultado exato

$$(-1)^s M 2^E$$

Sinal s , significando M :

Resultado de alinhamento e adição

Expoente E : $E1$

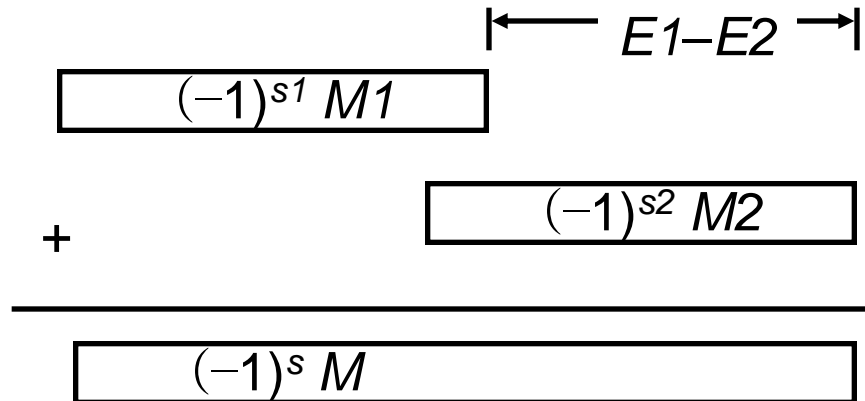
Representação final

Se $M \geq 2$, deslocar à direita M , incrementa E

Se $M < 1$, deslocar à esquerda M de k posições, decrementar E de k

Overflow se E fora do intervalo

arredonda M para número correto de bits



Exemplo da soma na base 10

$$1.234 \times 10^5 + 4.32 \times 10^{-1}$$

$$E1 = 5 \text{ e } E2 = -1 \rightarrow E1 - E2 = 5 - (-1) = 6$$

1.	2	3	4	0	0	0	0	0	0
----	---	---	---	---	---	---	---	---	---

+

0.	0	0	0	0	0	4	3	2	0
----	---	---	---	---	---	---	---	---	---

123400

0.432 +

123400.432

1.	2	3	4	0	0	4	3	2	0
----	---	---	---	---	---	---	---	---	---

 $\times 10^5$