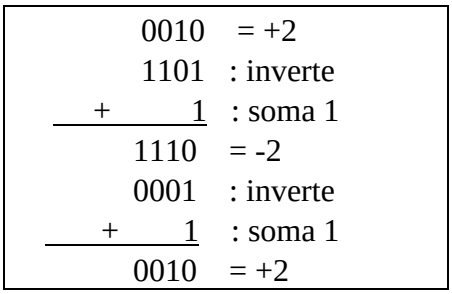
Universidade do Sul de Santa Catarina – Unisul
UC Sistemas Computacionais e Segurança
Professora Márcia Giraldi

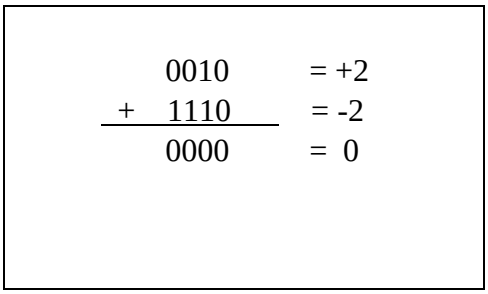
Parte Operativa – Unidade Lógica e Aritmética

1. Notação em Complemento de 2

A notação em complemento de 2 é a forma mais comumente utilizada para representar números com sinal em computadores. Nesta notação, se o bit mais significativo (o bit mais à esquerda quando o número é escrito) é igual a 0, o número é considerado positivo e o seu valor decimal pode ser lido diretamente pela maneira convencional de conversão de valores binários para valores decimais. No entanto, se o bit mais significativo é igual a um, o número é negativo e a sua magnitude pode ser encontrada invertendo-se bit-a-bit a representação binária, somando-se 1 ao valor invertido e fazendo-se a conversão normal de binário para decimal. Esta operação é ilustrada na fig. 1.



(a) Inversão de sinais

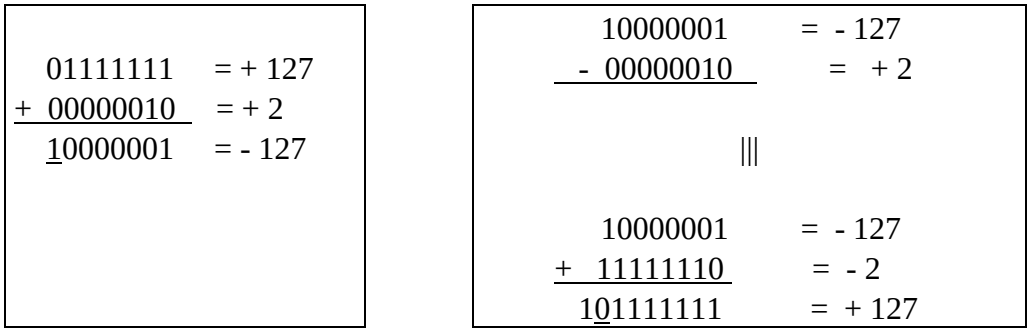


(b) Soma de números complementares.

Fig. 1. Operações em Complemento de 2.

2. Overflow

Overflow ocorre quando o número de bits do resultado é maior do que a capacidade de representação da arquitetura. A fig. 2 mostra situações de ocorrência de overflow em aritmética em complemento de dois em uma arquitetura de 8 bits. Overflow pode ocorrer tanto em operações de adição quanto de subtração. Uma situação de overflow pode ser detectada analisando-se os sinais dos operandos, o tipo de operação e o sinal do resultado. A tabela 1 apresenta as situações em que ocorre overflow.



(a) Adição **(b)** Subtração

Fig. 2. Ocorrência de overfow em operações aritméticas.

Operação	Sinal do Operando **A**	Sinal do Operando **B**	Sinal do Resultado
A+B	+	+	-
A+B	-	-	+
A-B	+	-	-
A-B	-	+	+
A+B	+	-	**não**
A+B	-	+	**não**
A-B	+	+	**não**
A-B	-	-	**não**

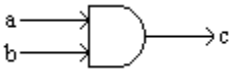
Tabela 1. Condições para ocorrência de overflow.

A detecção de overflow (transbordamento) e a decisão de o que fazer no caso de ocorrência de overflow é feita pelo projetista do processador. Pode-se gerar uma exceção forçando-se assim o software a tomar uma ação no caso de ocorrência de overflow, ou pode-se apenas setar um flag indicando que ocorreu overflow. Neste último caso é responsabilidade do programador verificar este flag e tomar alguma providência no caso de ocorrência de overflow.

3. Unidade Lógica e Aritmética para Computadores (ULA)

A Unidade Lógica e Aritmética é o componente principal da parte operativa de qualquer microprocessador. Ela realiza operações aritméticas como adição e subtração e operações lógicas como "E" e "OU". A seguir, demonstramos como uma ULA pode ser construída a partir das 4 funções lógicas elementares mostradas na fig. 3.

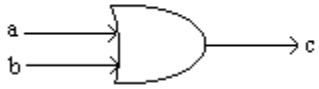
1. Porta E : (c = a . b)



a	b	c = a . b
0	0	0
0	1	0

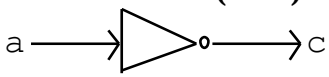
1	0	0
1	1	1

2. Porta OU : (c = a + b)



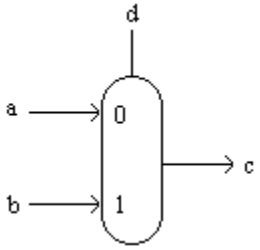
A	B	c = a + b
0	0	0
0	1	1
1	0	1
1	1	1

3. Inversor : (c = a)



a	C = ^a
0	1
1	0

4. Multiplexador: (Se d = 0 então c recebe a
senão c recebe b)



d	c
0	a
1	b

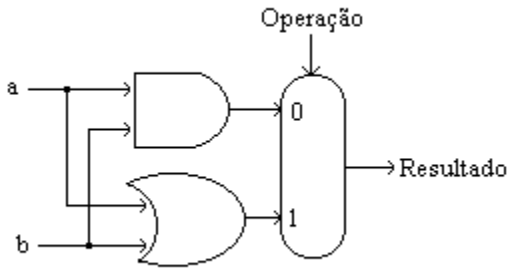
Fig. 3. Funções Lógicas Elementares.

Uma Unidade Lógica de um bit pode ser construída utilizando-se uma porta E, uma porta OU e um multiplexador como ilustrado na fig. 4.

A próxima função a incluir é a soma. Para construir um somador de 1 bit, devemos considerar como a soma é realizada. Conforme indicado na fig. 5, cada somador de 1 bit tem três entradas: os 2 operandos e o *vem-um* (carry-in) que é oriundo do transbordamento do bit imediatamente anterior, e produz duas saídas: 1 bit *soma* e 1 bit de *vai-um* (carry-out) que será usado no somador do bit posterior.

A “tabela verdade” do somador de 1 bit é apresentada na fig. 6. Observe que a saída "Vai-Um" será igual a 1 se e somente se pelo menos duas das entradas forem 1

simultaneamente. A saída *soma* será igual a um se e somente se um número ímpar de entradas for igual a 1.



OPERAÇÃO	RESULTADO
0	a . b
1	a + b

Fig. 4. Unidade Lógica de 1 bit para as operações "E" e "OU".

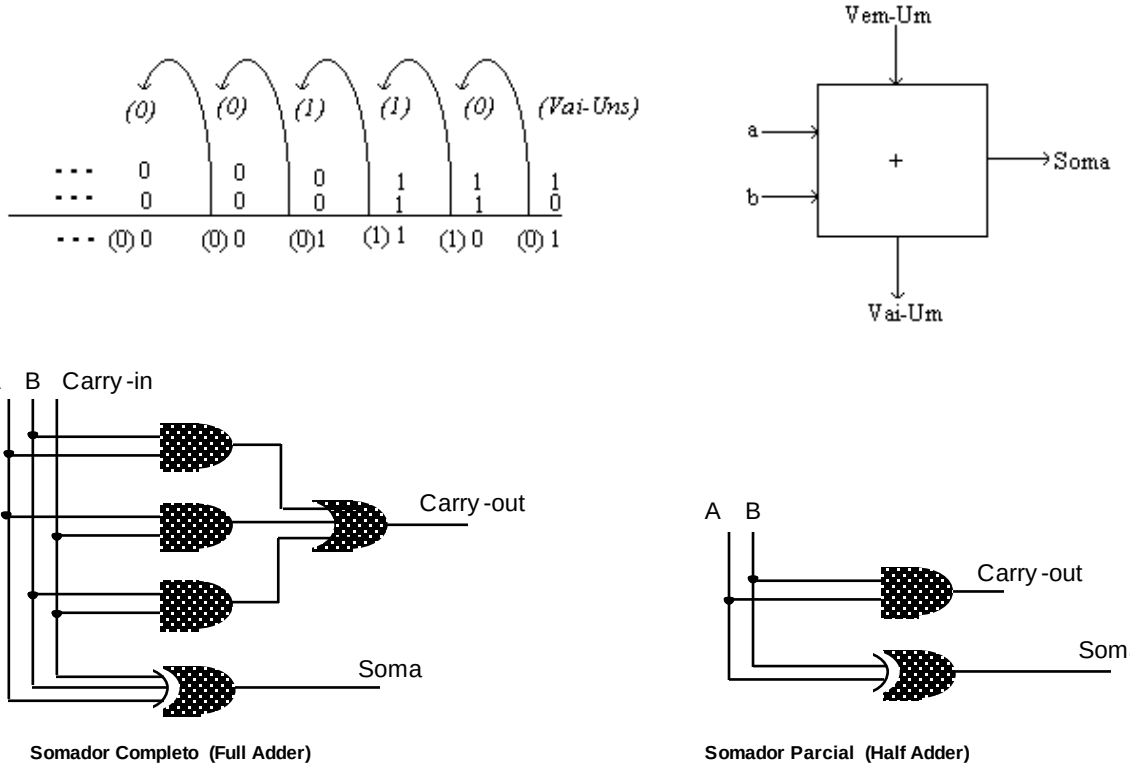


Fig. 5. Somador de 1 bit.

Entradas			Saídas		Comentários
a	b	Vem-um	Vai-um	Soma	
0	0	0	0	0	$0+0+0=00_{two}$
0	0	1	0	1	$0+0+1=01_{two}$
0	1	0	0	1	$0+1+0=01_{two}$
0	1	1	1	0	$0+1+1=10_{two}$
1	0	0	0	1	$1+0+0=01_{two}$

1	0	1	1	0	$1+0+1=10_{two}$
1	1	0	1	0	$1+1+0=10_{two}$
1	1	1	1	1	$1+1+1=11_{two}$

Fig. 6. Tabela verdade do somador de 1 bit.

Juntando a unidade lógica de 1 bit e o somador de 1 bit, podemos construir uma Unidade Lógica e Aritmética (ULA) de 1 bit conforme ilustrado na fig. 7. Observe que agora estamos utilizando um multiplexador com três entradas, portanto precisaremos de dois bits para especificar a operação a ser realizada na ULA.

A ULA mostrada na fig. 7 possui uma séria limitação: ela não subtrai. Lembrando que em notação de complemento de 2, pode-se obter o complemento de um número invertendo-se todos os bits e somando-se 1, podemos incluir a operação de subtração conforme ilustra a fig. 8, utilizando um segundo multiplexador e um inversor. Agora para realizar a operação de subtração devemos especificar a inversão do operando *b* e colocar 1 na entrada *vem-um* do somador.

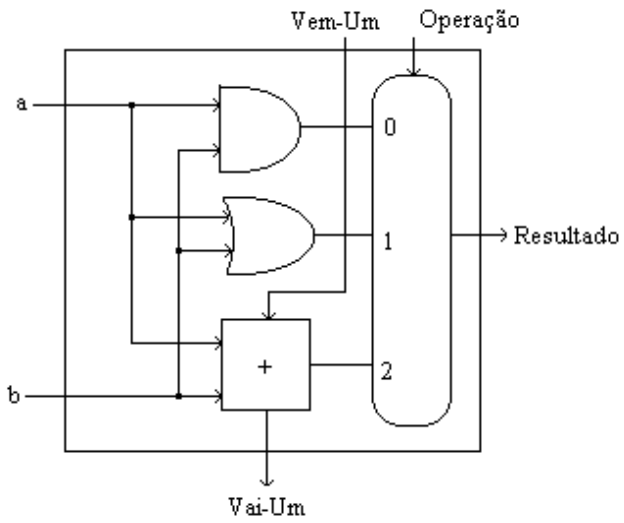


Fig. 7. Unidade Lógica e Aritmética de 1 bit.

Finalmente, 32 destas ULAs de 1 bit podem ser conectadas para formar uma ULA de 32 bits, conforme ilustrado na fig. 9.

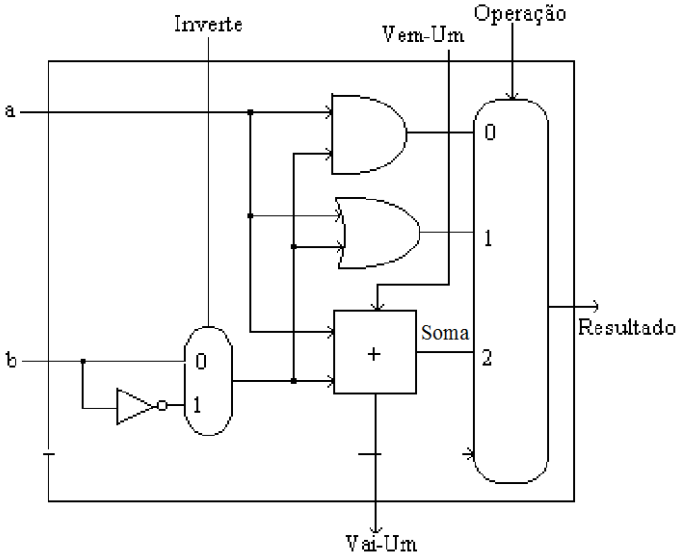


Fig. 8. ULA de 1 bit que soma, subtrai e realiza as operações lógicas "E" e "OU".

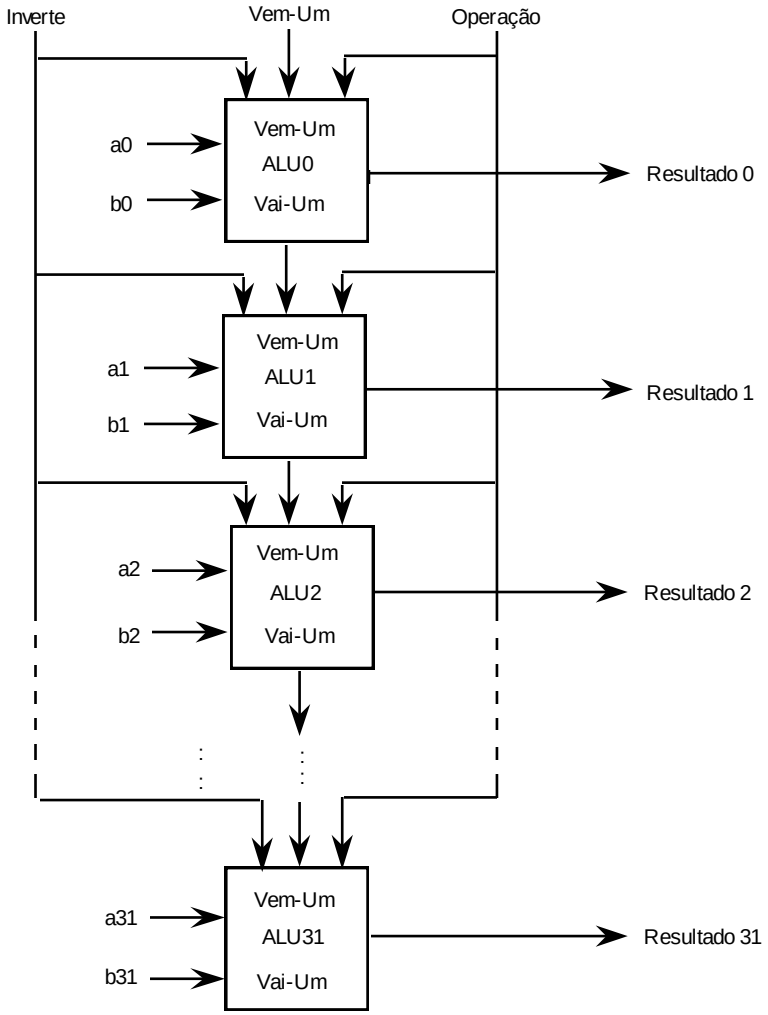
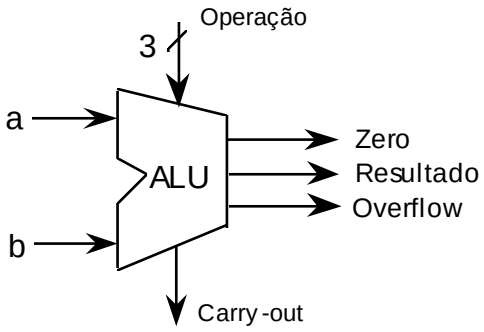


Fig. 9. ULA de 32 bits.

O símbolo utilizado com mais freqüência para representar uma ULA na representação em diagrama de blocos de o sistema é mostrado na fig. 10.



Linhas de Controle da ULA		Função
Inverte	Operação	
0	00	E
0	01	OU
0	10	soma
1	10	subtrai

Fig. 10. Símbolo para ULA (a linha "Operação" inclui as linhas "Inverte" e "Operação" da fig. 8).

3.1 Operação de Multiplicação

Nesta seção vamos estudar dois algoritmos de multiplicação. Começaremos com um algoritmo simples para compreender o fluxo de dados. Em seguida apresentaremos o Algoritmo de Booth, que é um dos algoritmos mais utilizados na prática. A escolha dos projetistas pelo Algoritmo de Booth se justifica pelo fato dele poder multiplicar números positivos e negativos, independentemente dos seus sinais.

Os dois operandos a serem multiplicados são chamados de multiplicando e multiplicador e o resultado é chamado de produto:

Multiplicando: B
Multiplicador: x A .
Produto: C

Para implementar um algoritmo de multiplicação em hardware necessitamos dois registradores: o registrador do Multiplicando e o do Produto. Se considerarmos uma multiplicação de números com *n* bits, o registrador do Produto deverá ter *2n* bits. O registrador do Produto é dividido em duas partes: produto(alto) e produto(baixo). A parte superior do registrador do Produto, produto(alto), com n bits, é inicializada com 0s, enquanto que o multiplicando é colocado no registrador do Multiplicando e o multiplicador é colocado na parte inferior do registrador do Produto: produto(baixo). A operação deste primeiro algoritmo de multiplicação pode ser descrita conforme visto na fig. 12.

Fig. 12. Algoritmo Multiplicação 1.

Exemplo: $5_{10} \times 2_{10} = 00101_2 \times 00010_2$, onde: multiplicador: 00101 (5_{10})
multiplicando: 00010 (2_{10})

MULTIPLICAÇÃO 1:

produto (alto) recebe zeros
produto (baixo) recebe multiplicador
para i = zero até 31
do se multiplicador bit zero = 1
então produto (alto) recebe produto (alto) + multiplicando
Fim se
produto recebe produto descolado para direita

Iteração	Operação	Multiplicando	Produto alto	Produto baixo	Multiplicador(0)
0	*Inicialização*	00010	00000	00101	1
1	Prod(alto) recebe Prod(alto) + Mult	00010	00010	00101	
	Prod recebe Prod deslocado para direita	00010	00001	00010	0
2	Sem operação	00010	00001	00010	
	Prod recebe Prod deslocado para direita	00010	00000	10001	1
3	Prod(alto) recebe Prod(alto) + Mult	00010	00010	10001	
	Prod recebe Prod deslocado para direita	00010	00001	01000	0
4	Sem operação	00010	00001	01000	
	Prod recebe Prod deslocado para direita	00010	00000	10100	0
5	Sem operação	00010	00000	10100	
	Prod recebe Prod deslocado para direita	00010	00000	01010	0

O hardware necessário para implementar este algoritmo é mostrado na fig. 13.

Este algoritmo estudado faz a multiplicação de números inteiros positivos. Para operar números inteiros negativos, poderíamos transformá-los em positivos, realizar a multiplicação e negar o resultado se os operandos fossem de sinais opostos.

Um algoritmo mais elegante para multiplicar números com sinal é o *Algoritmo de Booth*. Para entender este algoritmo, observamos que existem várias maneiras para calcular o produto de dois números. Ao encontrar uma seqüência de 1s no multiplicador, ao invés de realizar uma soma para cada um dos 1s da seqüência, o Algoritmo de Booth faz uma subtração ao encontrar o primeiro 1 e uma soma após o último.

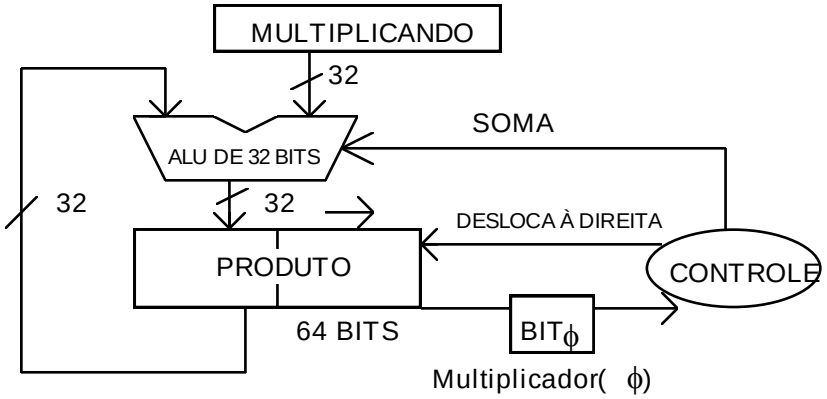


Fig. 13. Hardware para Multiplicação 1.

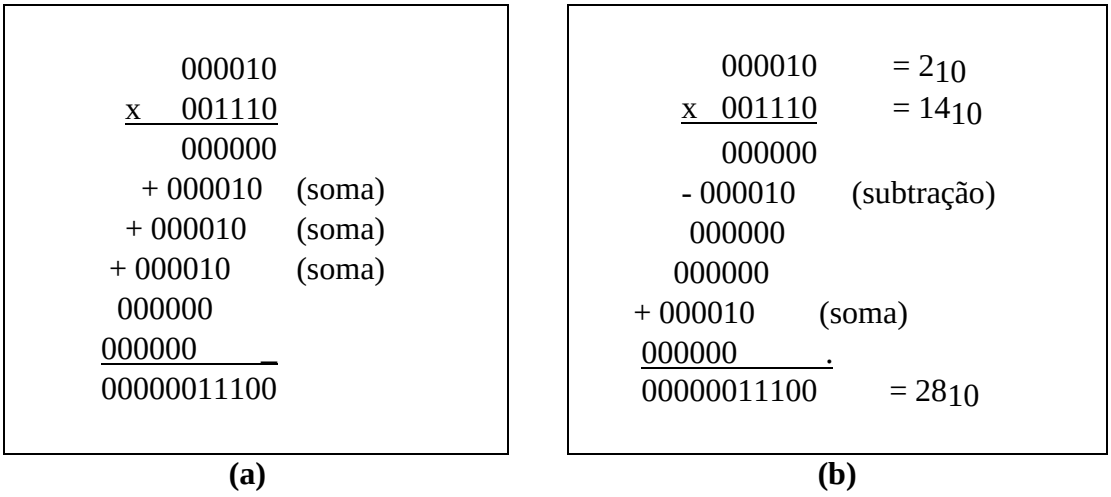


Fig. 14. Multiplicação de +2 por +14 representados em 6 bits: **(a)** Método tradicional; **(b)** Método de Booth.

Observe no exemplo apresentado na fig. 14 que enquanto o método tradicional necessitou de 3 operações aritméticas para efetuar a multiplicação, o método de Booth completou a multiplicação com apenas 2 duas. Podem-se construir exemplos em que o método de Booth necessita mais operações aritméticas. No entanto, como números negativos representados em complemento de dois tendem a possuir uma longa seqüência de uns em sua representação, a multiplicação destes números pelo método de Booth é em geral mais rápida. Na representação do algoritmo de multiplicação de Booth da fig. 15, produto(alto) representa a metade alta do registrador de produto, produto(baixo) representa a metade baixa do registrador de produto, produto(0) representa o bit menos significativo do registrador de produto e bit_à_direita representa um bit armazenado em hardware e que memoriza qual era o valor do bit à direita do bit do multiplicador que está sendo processado. O símbolo $>>_{aritmético}$ indica um deslocamento aritmético à direita. Quando um deslocamento aritmético é realizado, 0s são introduzidos à esquerda se o bit mais

significativo do número original era 0, e 1s são introduzidos à esquerda se o bit era 1. Em outras palavras, o deslocamento preserva o sinal do operando.

BOOTH:

1. produto (alto) recebe zero
2. produto (baixo) recebe multiplicador
3. bit_à_direita recebe zero
4. **for** i = zero **to** 31
5. **do if** bit_à_direita = zero **and** produto (zero) = 1
6. **then** produto (alto) recebe produto (alto) - multiplicando
7. **else if** bit_à_direita = 1 **and** produto (zero) = zero
8. **then** produto (alto) recebe produto (alto) + multiplicando
9. bit_à_direita recebe produto (zero)
10. produto recebe produto >>aritmético 1

Fig. 15. Algoritmo de Booth.

Lembrando que a operação de soma pode levar bastante tempo para ser realizada por causa do tempo necessário para propagar o "vai-um" até o bit mais significativo, e lembrando também que a subtração é feita pelo mesmo circuito do somador, o Algoritmo de Booth oferece a vantagem de redução do número de operações na ULA quando o multiplicador possui uma sequência longa de 1s.

Verifique no hardware para o algoritmo de Booth apresentado na fig. 16 a presença de um bit extra de armazenamento. Este bit é necessário para "lembrar" o bit à direita do bit que está sendo processado.

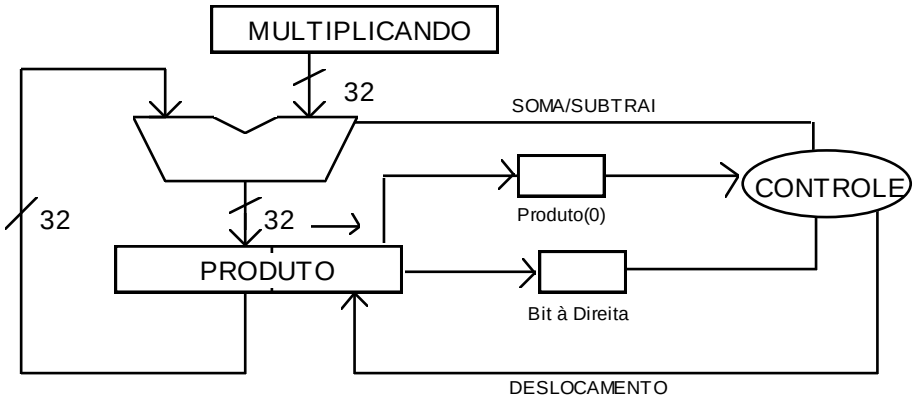


Fig. 16. Hardware para Algoritmo de Multiplicação de Booth.

No exemplo a seguir, o Algoritmo de Booth é usado para multiplicar números negativos expressos em notação complemento de 2.

Exemplo 1: $-3_{10} \times 2_{10} = 0010 \times 1101$, onde:

multiplicador: 1101 (-3_{10})
multiplicando: 0010 (2_{10})
Obs.: $-2_{10} = 1110$.

Iteração	Operação	Multiplicando	Produto alto	Produto baixo	Bit à direita/ Produto(0)
0	*Inicialização*	0010+ 1110-	0000	1101	0 / 1
1	Prod(alto)recebe Prod(alto) – Mult Prod recebe Prod deslocado aritmético 1	0010 0010	1110 1111 + 0010	1101 0110	1 / 0
2	Prod(alto) recebe Prod(alto) + Mult Prod recebe Prod deslocado aritmético 1	0010 0010	0001 0000	0110 1011	0 / 1
3	Prod(alto) recebe Prod(alto) – Mult Prod recebe Prod deslocado aritmético 1	0010 0010	1110 1111	1011 0101	1 / 1
4	Sem operação Prod recebe Prod descolado aritmético 1	0010 0010	1111 1111 00000101 +1 00000110	0101 1010	1 / 0

Exemplo 2: $-2_{10} \times -2_{10} = 1110 \times 1110$, onde:

multiplicador: 1110 (-2_{10})
multiplicando: 1110 (-2_{10})

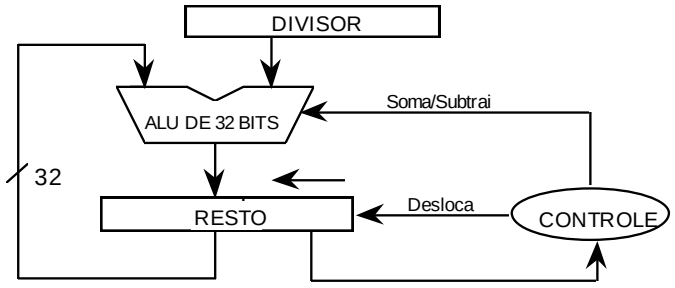
Iteração	Operação	Multiplicando	Produto alto	Produto baixo	Bit à direita/ Produto(0)
0	*Inicialização*	1110	0000	1110	0 / 0
1	Sem operação	1110	0000	1110	

	Prod recebe Prod descolado aritmético 1	1110	0000 0111	0 / 1
2	Prod recebe Prod - Mult Prod recebe Prod descolado aritmético 1	1110 1110	0010 0111 0001 0011	1 / 1
3	Sem operação Prod recebe Prod descolado aritmético 1	1110 1110	0001 0011 0000 1001	1 / 1
4	Sem operação Prod recebe Prod descolado aritmético 1	1110 1110	0000 1001 0000 0100	1 / 0

3.2 Operação de Divisão

Dividendo = A | B = Divisor
C = Quociente
R = Resto

A divisão pode ser computada com o mesmo hardware da multiplicação:



E o algoritmo para a divisão pode ser visto na fig. 17.

DIVISÃO:

resto (alto) recebe zero
resto (baixo) recebe dividendo
resto recebe resto deslocado para esquerda
Para i = zero até 31
faça resto (alto) recebe resto (alto) - divisor
Se resto(alto) menor que zero
então resto (alto) recebe resto (alto) + divisor
resto recebe resto deslocado para esquerda
senão resto recebe resto deslocado para esquerda aritmético 1

Fim se
Fim para
resto(alto) recebe resto(alto) deslocado para direita

Fig. 17. Algoritmo de Divisão.

Exemplo 1: $7_{10} \div 2_{10} = 0111 \div 0010$, onde: dividendo: 0111 (7_{10})
divisor: 0010 (2_{10})
Obs: $-2 = 1110_2$

Iteração	Operação	Divisor	Resto alto	Resto baixo
0	*Inicialização*	0010	0000	0111
	Resto recebe Resto deslocado para esquerda	0010	0000	1110
1	Resto(alto) recebe Resto(alto) - Divisor	0010	**1**110	1110
	Resto < 0 : Resto(alto) recebe Resto(alto) + Divisor	0010	0000	1110
	Resto recebe Resto deslocado para esquerda	0010	0001	1100
2	Resto recebe Resto – Divisor	0010	**1**111	1100
	Resto(alto) < 0 : Resto(alto) recebe Resto(alto) + Divisor	0010	0001	1100
	Resto recebe Resto deslocado para esquerda	0010	0011	1000
3	Resto recebe Resto – Divisor	0010	**0**001	1000
	Resto = 0 : Resto recebe Resto deslocado para esquerda aritmético 1	0010	0011	0001
4	Resto recebe Resto – Divisor	0010	**0**001	0001
	Resto > 0 : Resto recebe Resto deslocado para esquerda aritmético 1	0010	0010	0011
-	Resto (alto) recebe Resto (alto) deslocado para direita	0010	0001	0011

Resto / Quoc.

Exemplo 2: $8_{10} \div 3_{10} = 01000 \div 00011$, onde:
dividendo: 01000 (8_{10})
divisor: 00011 (3_{10})
Obs: $-3 = 11101_2$

Iteração	Operação	Divisor	Resto alto	Resto baixo
0	*Inicialização*	00011	00000	01000
	Resto recebe Resto deslocado para esquerda	00011	00000	10000

1	Resto recebe Resto - Divisor Resto < 0 : Resto + Divisor Resto recebe Resto deslocado para esquerda	00011 00011 00011	**1**1101 10000 00000 10000 00001 00000
2	Resto recebe Resto - Divisor Resto < 0 : Resto + Divisor Resto recebe Resto deslocado para esquerda	00011 00011 00011	**1**1110 00000 00001 00000 00010 00000
3	Resto recebe Resto - Divisor Resto < 0 : Resto recebe Resto + Divisor Resto recebe Resto deslocado para esquerda	00011 00011 00011	**1**1111 00000 00010 00000 00100 00000
4	Resto <-- Resto - Divisor Resto > 0 : Resto recebe Resto deslocado para esquerda aritmético 1	00011 00011	**0**0001 00000 00010 00001
5	Resto recebe Resto - Divisor Resto < 0 : Restorecebe Resto + Divisor Resto recebe Resto deslocado para esquerda	00011 00011 00011	**1**1111 00001 00010 00001 00100 00010
-	Resto (alto) recebe Resto (alto) deslocado para direita	00011	00010 00010

Resto / Quoc.

Até o momento números negativos foram ignorados na divisão. A maneira mais simples é lembrar dos sinais do divisor e do dividendo e então negar o quociente se os sinais são diferentes. Note que a seguinte equação deve ser verificada:

Dividendo = Quociente x Divisor + Resto

Assim, veja o **exemplo**:

+7 ÷ +2: quociente = +3 e resto = +1
-7 ÷ +2: quociente = -3 e resto = -1
+7 ÷ -2: quociente = -3 e resto = +1
-7 ÷ -2: quociente = +3 e resto = -1

Desta forma, não basta apenas inverter o sinal do quociente quando os sinais do dividendo e do divisor forem diferentes, também é preciso notar que o sinal do resto sempre é o mesmo do dividendo.

4. Notação em Ponto Flutuante

Permite a representação de números reais e de números com magnitudes muito diferentes em uma forma padrão. A representação utilizada é a notação científica em que o ponto decimal é colocado à direita do primeiro algarismo significativo e a magnitude do número é armazenada no expoente.

Bibliografia:

PATTERSON, David A. & Hennessy, John L. **Organização e Projeto de Computadores: a interface Hardware/Software.** Rio de Janeiro: Editora LTC, 2 ed. 2000