

Arithmetic for Computers

Chapter Three

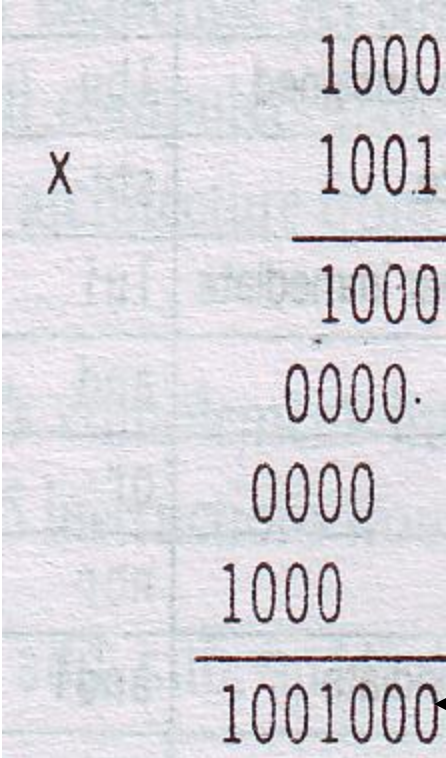
Book of David A. Patteron

P. Hayes

page-239-240

General Concept of Multiplication

$$1000_2 \times 1001_2$$

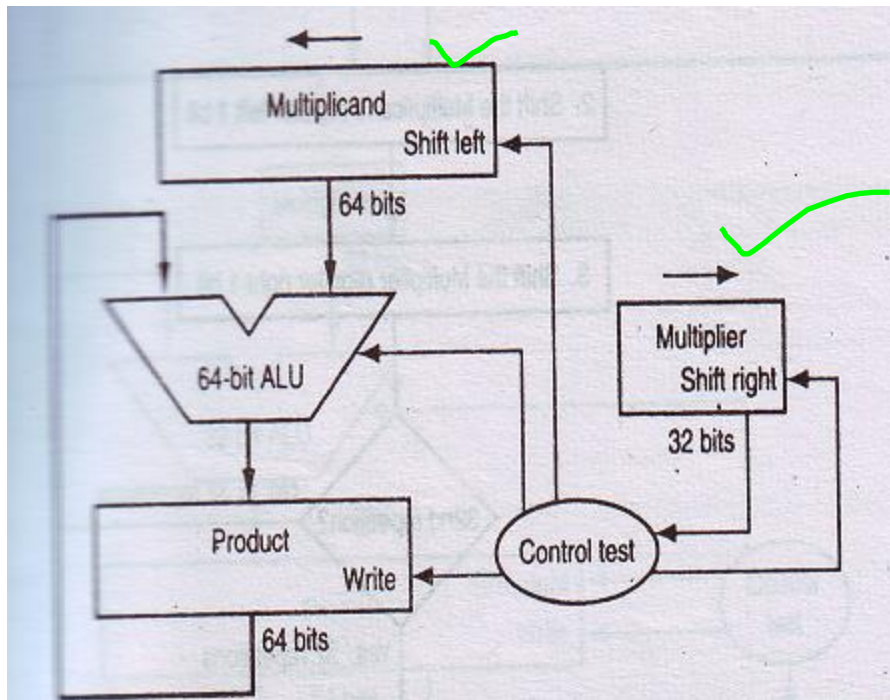


A handwritten binary multiplication on grid paper. The multiplicand 1000 is aligned to the right, and the multiplier 1001 is written below it. A horizontal line separates the multiplier from the partial products. The partial products are 1000 (shifted left by one position) and 0000 (shifted left by two positions). A second horizontal line separates the partial products from the final product 1001000, which is the sum of the partial products. Arrows point from the labels 'Multiplicand', 'Multiplier', and 'Product' to their respective parts in the calculation.

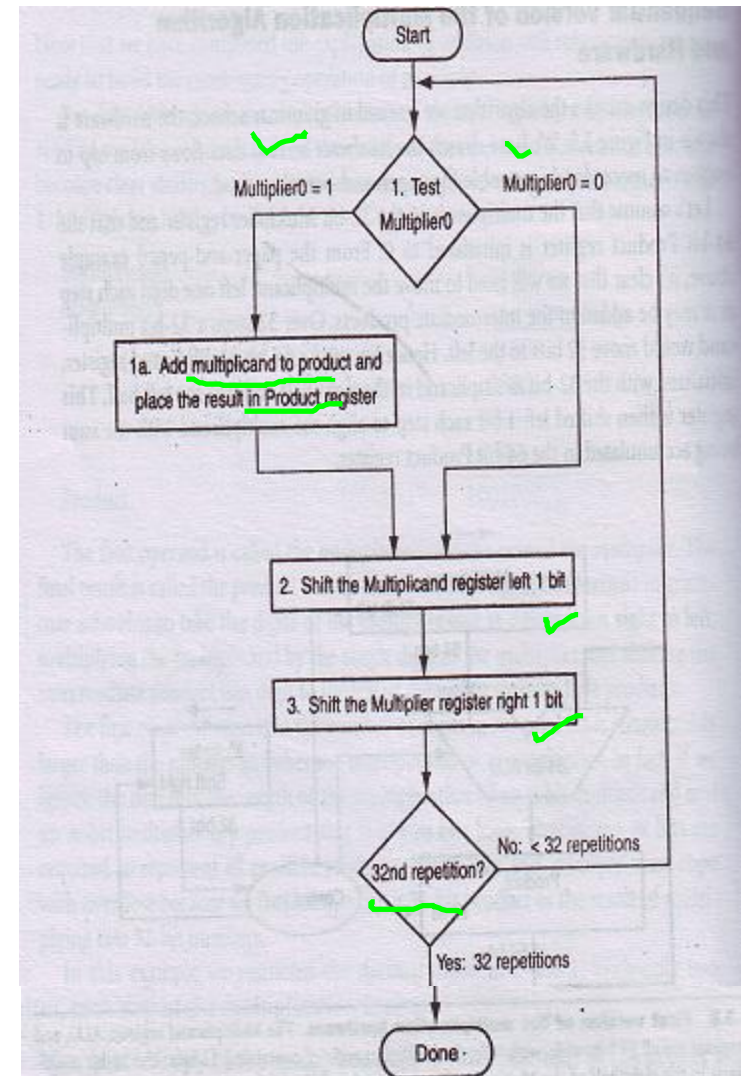
	1000	← Multiplicand
X	1001	← Multiplier
<hr/>		
	1000	
	0000	
	0000	
	1000	
<hr/>		
	1001000	← Product

- ✓ If multiplicand = n bits and multiplier = m bits then product = $n + m$ bits.
- ✓ Two rules:
 1. Place a copy of multiplicand in the proper place if multiplier bit=1.
 2. Place 0 in the proper place if multiplier bit = 0.
- ✓ Like addition, overflow can occur.

Sequential Version of the Multiplication Algorithm and Hardware (Unsigned Number)



✓ A moderate frequency for a slow operation can limit performance.



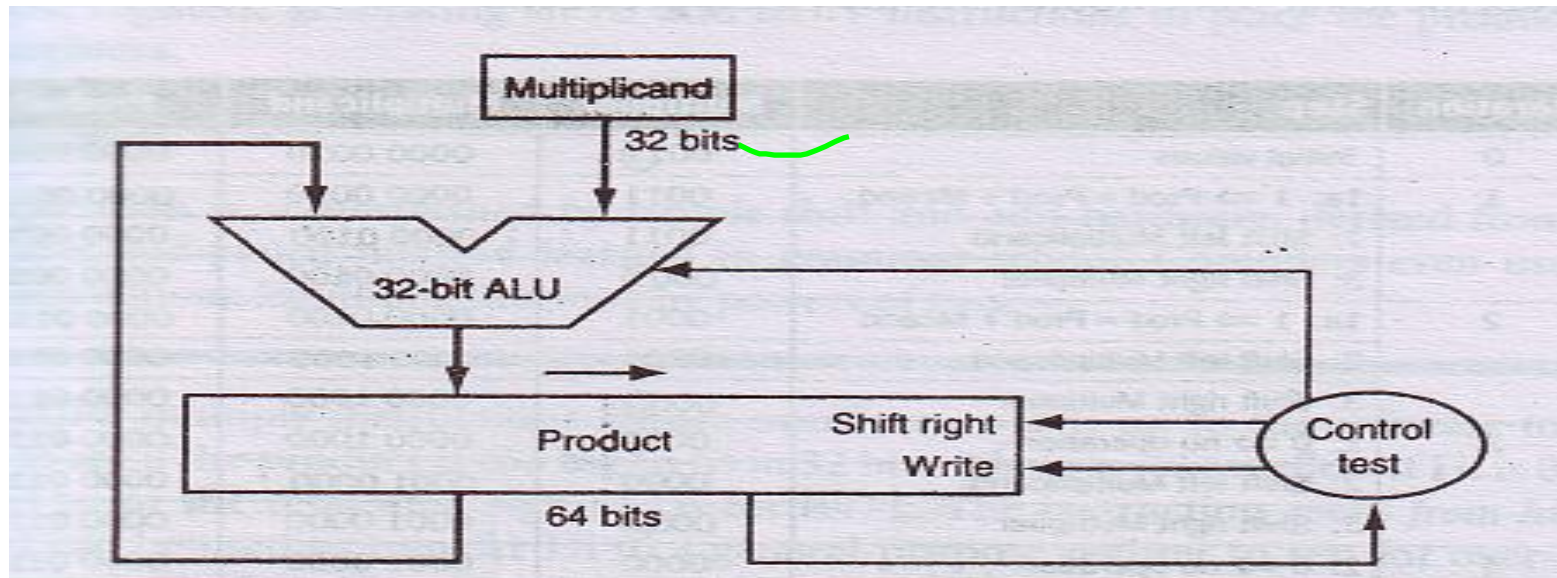
Example

Iteration	Step	Multiplier	Multiplicand	Product
0	Initial values	0011	0000 0010	0000 0000
1	1a: 1 \Rightarrow 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 \Rightarrow 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 \Rightarrow 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 \Rightarrow 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

Works for negative number. Consider sign extension during shift operation.

Refined Version of Multiplication Algorithm

- Execution time can be reduced through parallelism.
- Hardware cost can be reduced by using non-utilized portion of product register.



Example

Iteration	Multi- plicand	Original algorithm	
		Step	Product
0	<u>0010</u>	Initial values	0000 0110
1	0010	1: 0 \Rightarrow no operation	0000 0110
	0010	2: Shift right Product	0000 0011
2	0010	1a: 1 \Rightarrow Prod = Prod + Mcand	0010 0011
	0010	2: Shift right Product	0001 0001 ✓
3	0010	1a: 1 \Rightarrow Prod = Prod + Mcand	0011 0001
	0010	2: Shift right Product	0001 1000
4	0010	1: 0 \Rightarrow no operation	0001 1000
	0010	2: Shift right Product	0000 1100

Booth's Multiplication

- ✓ Used for both positive and negative number (2's complement number).
- ✓ It uses both add and sub to compute the product.
- ✓ Instead of 1 bit of a multiplier it considers 2 bits.
- ✓ It provides faster execution.

Concepts of Booth's Algorithm

$$2_{10} \times 6_{10} = 12_{10}$$

$$\begin{array}{r}
 \times \quad 0010_{\text{two}} \\
 \quad 0110_{\text{two}} \\
 \hline
 + \quad 0000 \quad \text{shift} \\
 + \quad 0010 \quad \text{add} \\
 + \quad 0010 \quad \text{add} \\
 + \quad 0000 \quad \text{shift} \\
 \hline
 00001100_{\text{two}}
 \end{array}$$

$$\begin{array}{r}
 \times \quad 0010_{\text{two}} \\
 \quad 0110_{\text{two}} \\
 \hline
 + \quad 0000 \quad \text{shift} \\
 - \quad 0010 \quad \text{add} \\
 + \quad 0000 \quad \text{shift} \\
 + \quad 0010 \quad \text{add} \\
 \hline
 00001100_{\text{two}}
 \end{array}$$

✓ $6_{10} = -2_{10} + 8_{10}$

» $6_{10} \times 2_{10} = -2_{10} \times 2_{10} + 8_{10} \times 2_{10}$

» $12_{10} = -4_{10} + 16_{10}$

✓ Condition:

1. subtract when first see a 1. (10).
2. add when see the bit after the last 1. (01)

Validity of Booth's Algorithm

$$X^* = x_i x_{i-1} x_{i-2} \dots x_{i-k+1} x_{i-k} x_{i-k-1}$$

$$= 0 \ 1 \ 1 \ \dots \ 1 \ 1 \ 0$$

- In normal multiplication the contribution of X^* to $P = X \times Y$ is $\sum_{j=i-k}^{i-1} 2^j Y$
- In booth's multiplication, $x_i x_{i-1} = 01$ which contributes $2^i Y$.
- When $x_{i-k} x_{i-k-1} = 10$ the contribution is $-2^{i-k} Y$.
- Net Contribution:

$$2^i Y - 2^{i-k} Y = 2^{i-k} Y (2^k - 1)$$

$$= 2^{i-k} \sum_{m=0}^{k-1} 2^m Y$$

$$= \sum_{m=0}^{k-1} 2^{m+i-k} Y$$

- If $j = m + i - k$ then we get $\sum_{j=i-k}^{i-1} 2^j Y$

Booth's Algorithm

✓ Step 1:

Depending on the current and previous bit, do one of the following:

00: No arithmetic operation.

01: Add multiplicand to the left half of the product.

10: Sub the multiplicand from the left half of the product.

11: No arithmetic operation.

✓ Step 2:

Shift the product register right 1 bit.

Example

Iteration	Multi-plicand	Original algorithm		Booth's algorithm	
		Step	Product	Step	Product
0	0010	Initial values	0000 0110	Initial values	0000 0110 0
1	0010	1: $0 \Rightarrow$ no operation	0000 0110	1a: 00 \Rightarrow no operation	0000 0110 0
	0010	2: Shift right Product	0000 0011	2: Shift right Product	0000 0011 0
2	0010	1a: $1 \Rightarrow \text{Prod} = \text{Prod} + \text{Mcand}$	0010 0011	1c: 10 $\Rightarrow \text{Prod} = \text{Prod} - \text{Mcand}$	<u>1110 0011 0</u>
	0010	2: Shift right Product	0001 0001 ✓	2: Shift right Product	1111 0001 1
3	0010	1a: $1 \Rightarrow \text{Prod} = \text{Prod} + \text{Mcand}$	0011 0001	1d: 11 \Rightarrow no operation	1111 0001 1
	0010	2: Shift right Product	0001 1000	2: Shift right Product	1111 1000 1
4	0010	1: $0 \Rightarrow$ no operation	0001 1000	1b: 01 $\Rightarrow \text{Prod} = \text{Prod} + \text{Mcand}$	0001 1000 1
	0010	2: Shift right Product	0000 1100	2: Shift right Product	0000 1100 0