

Now take a look at signed numbers with fractions.

$$\begin{array}{r}
 (+1.25) \quad 01.01 \\
 \times (-1.5) \quad \times 10.10 \\
 \hline
 00000000 \\
 +0000101 \\
 \hline
 00001010 \\
 +000000 \\
 \hline
 00001010 \\
 +11011 \quad (\text{2's comp}) \\
 \hline
 \boxed{1110.0010} \\
 -1.875
 \end{array}$$

(2's comp)

2's comp is 0001.1110

$$1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} = 1.875$$

$$\begin{array}{r}
 01.01 \\
 \times 10.10 \\
 \hline
 (\underline{11} \underline{10}) \text{ BRC} \\
 00000000 \\
 00000000 \\
 +1111011 \\
 \hline
 11110110 \\
 +000101 \\
 \hline
 00001010 \\
 +11011 \\
 \hline
 \boxed{1110.0010} \\
 -1.875
 \end{array}$$

step "0"
 Bit 0 of BRC = shift
 Bit 1 of BRC = $\begin{matrix} \text{sub} \\ \text{shift} \end{matrix}$
 Bit 2 of BRC = $\begin{matrix} \text{add} \\ \text{shift} \end{matrix}$
 Bit 3 of BRC = $\begin{matrix} \text{sub} \\ \text{shift} \end{matrix}$

Note: With Booth's Algorithm we do not do anything different for the sign bit!

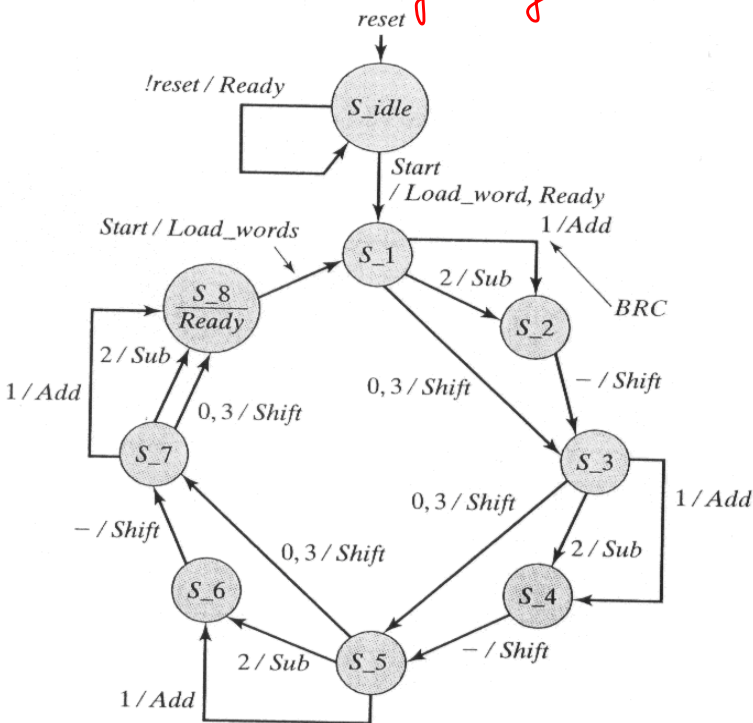


FIGURE 10-50 STG for a 4-bit Booth sequential multiplier.

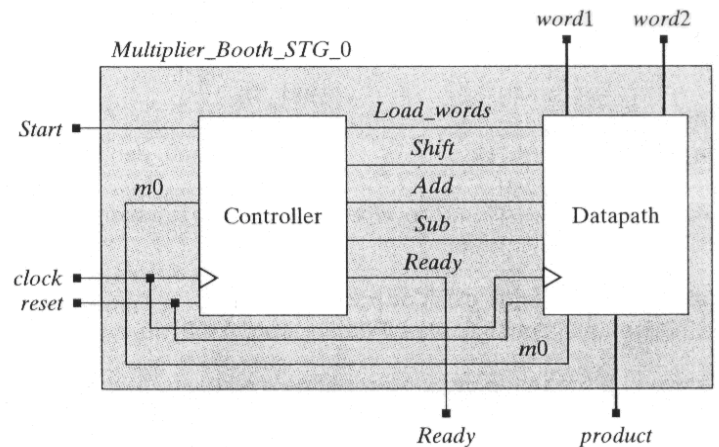


FIGURE 10-51 Structural units of a multiplier with Booth recoding.

The datapath above contains two registers and an adder/subtractor unit.

One register holds the multiplicand and the other holds the intermediate answer and multiplier.

Note however, that the state controller for this implementation of Booth's Algorithm shows that we don't actually save any more states (or compute cycles) as compared with our original signed multiplier.

This particular version of Booth's algorithm is called a Radix-2 Booth's Algorithm and it only looks at a bit and its neighbor (groups of two bits).

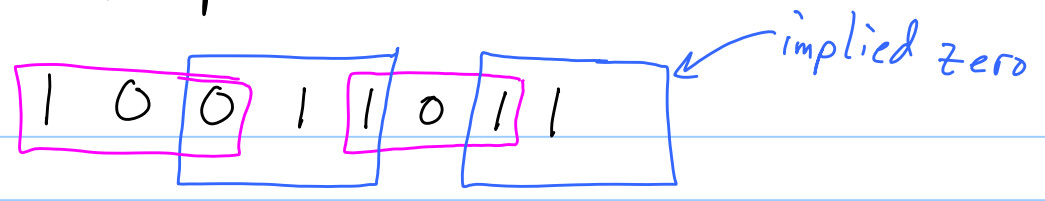
More elaborate versions of Booth's Algorithm however, will save us compute cycles.

Radix-4 Booth's Algorithm

To perform the Radix-4 Booth's Algorithm we will group three bits at a time instead of two.

The groups of 3 overlap each other by one bit with the first group using the least two significant digits (assuming an overlap with an implied zero to the right).

8-bit example:



m_i	m_{i-1}	m_{i-2}	Action (or Partial Product)
0	0	0	0
0	0	1	+ Multiplicand
0	1	0	+ Multiplicand
0	1	1	+ (2 * Multiplicand)
1	0	0	- (2 * Multiplicand)
1	0	1	- Multiplicand
1	1	0	- Multiplicand
1	1	1	0

Example:

$4^{3/16}$
 $-4^{7/16}$

0100.0011
 * 1011.1001

0000000010000011
 + 111111011110

 111111000101011
 + 000000000000

 111111000101011
 + 1110111101

 11101101.01101011

Annotations:
 - 010 + M (blue arrow)
 - 100 (pink arrow)
 - -2M (pink arrow)
 - 111 Nothing (orange arrow)
 - 101 -M (green arrow)

2's comp 006 | 06 | 0 . | 06 | 0 | 01
 is ↑ ↑ ↑ ↑ ↑ ↑
 16 + 2 + 1/2 + 1/16 + 1/64 + 1/256

$-18 \frac{149}{256}$