To Do Multiply …

Multiply Example

```
          10111001   (185)
        x 11010111   (215)
        ----------
          10111001
         10111001
        10111001
       00000000
      10111001
     00000000
    10111001
   10111001
----------------
1001101101011111 (39775)
```

```
        10111001
      x 11010111
      ----------
       10111001
0000001000101011 <- running sum
```

```
        10111001
      x 11010111
      ----------
      10111001   <- third PP
0000001000101011
```

```
      10111001
    x 11010111
    ----------
    10111001
0000001100001111 <- running sum
```

```
      10111001
    x 11010111
    ----------
   00000000    <- fourth PP
0000001100001111
```

```
    10111001
  x 11010111
  ----------
 00000000
0000001100001111 <- running sum
```

Pollard's Attempt to Explain Booth's Multiply

First, the Equations

$$\text{Value} = A \times B$$

$$\text{Value} = A \times b_4 b_3 b_2 b_1 b_0$$

$$\text{Value} = A \times (-b_4 \times 2^4 + b_3 \times 2^3 + b_2 \times 2^2 + b_1 \times 2^1 + b_0 \times 2^0)$$

And the Trick:

$$2^k = 2^{k+1} - 2^k$$

$$2^3 = 2^4 - 2^3$$

$$2^2 = 2^3 - 2^2$$

… and so on …

$$
\begin{aligned}
\text{Value} = A \times (\quad & -b_4 \times 2^4 + b_3 \times 2^4 \\
& -b_3 \times 2^3 + b_2 \times 2^3 \\
& -b_2 \times 2^2 + b_1 \times 2^2 \\
& -b_1 \times 2^1 + b_0 \times 2^1 \\
& -b_0 \times 2^0 + 0 \times 2^0 \quad)
\end{aligned}
$$

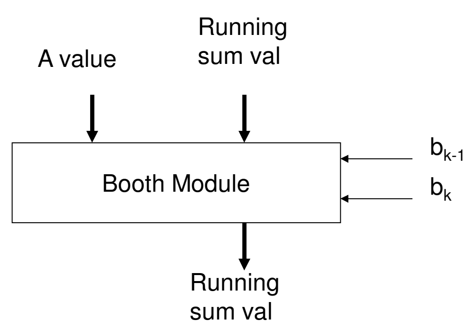
$$
\begin{aligned}
\text{Value} = A \times (\quad & (-b_4 + b_3) \times 2^4 \\
& (-b_3 + b_2) \times 2^3 \\
& (-b_2 + b_1) \times 2^2 \\
& (-b_1 + b_0) \times 2^1 \\
& (-b_0 + 0) \times 2^0 \quad)
\end{aligned}
$$

So, Two Bits Per Stage

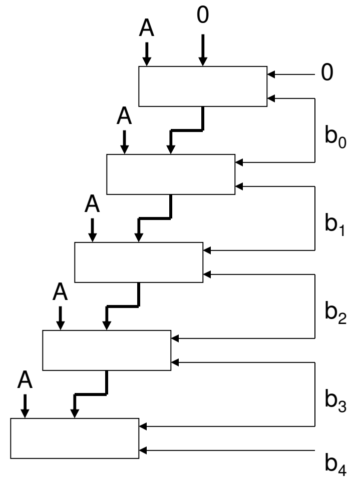
b_{k-1}	b_k	
0	0	0 - 0 = 0, pass value
0	1	0 - 1 = -1, subtract A
1	0	1 - 0 = 1, add A
1	1	1 - 1 = 0, pass value

Booth's Algorithm Module

So, system is made of modules that can
add (+1), subtract (-1), or pass (0) values of A….



Booth Conceptual Block Diagram



Try Algorithm: 9 x -12 in 8-bit 2's Complement

```
  9 = 00001001 = A
-12 = 11110100 = B
```

Step 1: use b_0 and 0, which are 0 and 0: Pass value

00000000 ← Running sum starts at zero

← Module passes value

000000000 ← Result of step 1

Pass these bits to step 2

00000000 ← Bits passed from step 1

$\boxed{}$ ← Step 2 uses $b_1 = b_0 = 0$ so pass value again

000000000 ← Result of step 2

Pass these bits to step 3

00000000 ← Bits passed from step 2

$\boxed{}$ ← Step 3 uses $b_2 = 1$, $b_1 = 0$ so subtract A from running sum

111110111 ← Result of step 3

Pass these bits to step 4

`11111011` ← Bits passed from step 3

$\boxed{}$ ← Step 4 uses $b_3 = 0$, $b_2 = 1$ so add A to running sum

`000000100` ← Result of step 4

Pass these bits to step 5

`00000010` ← Bits passed from step 4

$\boxed{}$ ← Step 5 uses $b_4 = 1$, $b_3 = 0$ so subtract A from running sum

`111111001` ← Result of step 5

Pass these bits to step 6

11111100 ← Bits passed from step 5

$\boxed{}$ ← Step 6 uses $b_5 = 1$, $b_4 = 1$ so pass running sum value

111111100 ← Result of step 6

Pass these bits to step 7

11111110 ← Bits passed from step 6

$\boxed{}$ ← Step 7 uses $b_5 = 1$, $b_4 = 1$ so pass running sum value

111111110 ← Result of step 7

Pass these bits to step 8

11111111 ← Bits passed from step 7

Step 8 uses $b_5 = 1$, $b_4 = 1$ so pass running sum value

111111111 ← Result of step 8

These bits are MSBs of result

Result is found by using one bit from steps 1 to 7, then remaining bits from step 8: 1111 1111 1 0 0 1 0 1 0 0