Multiplication

Multiplication = series of additions:

1. align first operand *(multiplicand)* with i^{th} digit of second operand *(multiplier)*
2. add shifted multiplicand n times to result (n = value of the multiplier digit)
3. repeat 1. and 2. for all multiplier digits i
4. n-digit $\times$ n-digit = $2n$-digit result

- binary system $\Rightarrow$ either 0 or 1
- problem is reduced to *add/noadd*

Example (decimal): $\boxed{6 \times 13 = 78}$

	0 6	Multiplicand
$\times$	1 3	Multiplier
	1 8	03×06
+ 0 6		10×06
0 0 7 8		Product

Example (binary): $\boxed{6 \times 13 = 78}$

	0 1 1 0	Multiplicand
$\times$	1 1 0 1	Multiplier
	0 1 1 0	0001×0110
	0 0 0 0	0000×0110
	0 1 1 0	0100×0110
+ 0 1 1 0		1000×0110
0 1 1 0 0 0 0		(carry)
0 1 0 0 1 1 1 0		Product

Multiplication Circuitry

1. Complete logical network:

- direct and fast implementation
- large number $(n \times n)$ of full adders
- potentially complex circuitry

Ripple-Carry: most intuitive realization
$\Rightarrow$ rippling per *row*
$\Rightarrow$ feed last carry to MSB of next row

Carry-Save: three rows at once
$\Rightarrow$ create separate Carry & Sum vectors
$\Rightarrow$ add up later
$\Rightarrow$ tree-like structure results

Figures 6.16–6.19 in the book

Multiplication Circuitry

2. Sequential shift/add multiplication:

- involves less circuitry
- uses single adder unit only
- result is developed in a shift register
- requires sequencing control $\Rightarrow$ *slow!*

3. Software multiplication:

- no extra circuitry
- subroutine solving multiplication
- loop with shift/rotate/add instructions
- requires several instructions $\Rightarrow$ *slowest!*

Multiplication Circuitry

Signed multiplicands?

- how to handle in partial products?
  $\Rightarrow$ sign extension

Optimization?

- reduce carry-over impacts
- reduce number of partial products
  $\Rightarrow$ Booth recoding
  $\Rightarrow$ bit pairing

next time...

Multiplication: $6 \times 13 = 78$

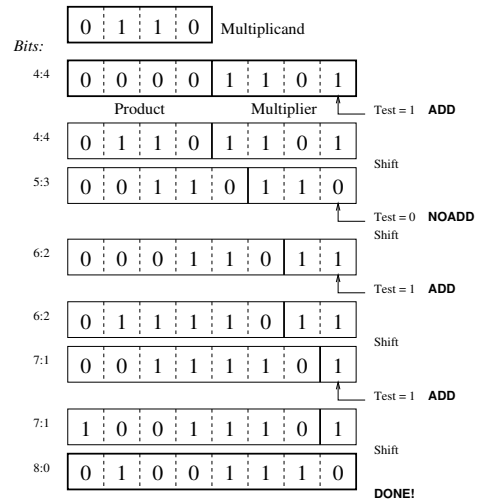
Sequential addition from row to row:

		Sum:
	0 1 1 0	
	× 1 1 0 1	
	0 1 1 0	00000110
	0 0 0 0	00000110
	0 1 1 0	00011110
+	0 1 1 0	01001110
0	1 0 0 1 1 1 1 0	Product

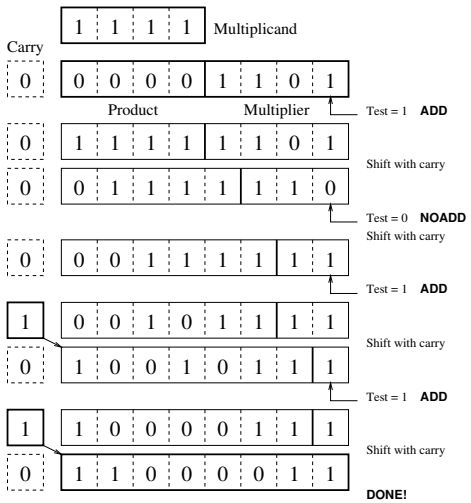
Sequential Shift/Add-Method

- Method to avoid adder arrays
- shift register for partial product and multiplier
- with each cycle,
  1. partial product increases by one digit
  2. multiplier is reduced by one digit
- MSBs of partial product and multiplicand are aligned in each cycle
- not the multiplicand is shifted
  $\Rightarrow$ partial product and multiplier are shifted together

Example 1: $6 \times 13 = 78$



Example 2: $15 \times 13 = 195$



Sequential Shift/Add-Method

1. Load multiplier into *lower* half of shift register (the *upper* half is to be zeroed)
2. test LSB of the shift register
3. if LSB is set
   - then add multiplicand to the *upper* half of the shift register
   - else add nothing (make sure carry-bit is cleared!)
4. perform right shift including carry on *full* shift register
5. repeat from 2. as long as multiplier part of shift register is not empty
6. after termination, the shift register (both halves!) contains the product

- Easy to implement in software