Computer Organization & Architecture
CS 204

Anil Kumar Sao, EECS, IIT Bhilai
anil@iitbhilai.ac.in

Number representation

Number Systems:

- Positive numbers:
- Negative Numbers
  - Sign and Magnitude Representation
  - 1’s Complement Representation
  - 2’s Complement Representation

Goal of number systems : arithmetic operations

Negative number systems

- Signed system: Simple. Just flip the sign bit
    - 0 = positive
    - 1 = negative

Definitions: Given a positive integer x, we represent -x

- 1's complement:
  - Formula: $2^n - 1 - x$
    - i.e. n=4, $2^4 - 1 - x = 15 - x$
    - In binary: $(1\ 1\ 1\ 1) - (b_3\ b_2\ b_1\ b_0)$
    - Just flip all the bits.
- 2's complement:
  - Formula: $2^n - x$
    - i.e. n=4, $2^4 - x = 16 - x$
    - Just flip all the bits and add 1.

Definitions: 4-Bit Example

id	b_3	b_2	b_1	b_0	Signed	One's	Two's
0	0	0	0	0	0	0	0
1	0	0	0	1	1	1	1
2	0	0	1	0	2	2	2
3	0	0	1	1	3	3	3
4	0	1	0	0	4	4	4
5	0	1	0	1	5	5	5
6	0	1	1	0	6	6	6
7	0	1	1	1	7	7	7
8	1	0	0	0	-0	-7	-8
9	1	0	0	1	-1	-6	-7
10	1	0	1	0	-2	-5	-6
11	1	0	1	1	-3	-4	-5
12	1	1	0	0	-4	-3	-4
13	1	1	0	1	-5	-2	-3
14	1	1	1	0	-6	-1	-2
15	1	1	1	1	-7	-0	-1

Given n-bits, what is the range of numbers in each system?

- 3 bits:
  - Signed: -3 , 3
  - 1's: -3 , 3
  - 2's: -4 , 3
- 6 bits
  - Signed: -31, 31
  - 1's: -31, 31
  - 2's: -32, 31

- 5 bits:
  - Signed: -15, 15
  - 1's: -15, 15
  - 2's: -16, 15
- Given 8 bits
  - Signed: -127, 127
  - 1's: -127, 127
  - 2's: -128, 127

Formula for calculating the range →

Signed & 1's: $-(2^{n-1} - 1)$, $(2^{n-1} - 1)$

2's: -2^{n-1} , $(2^{n-1} - 1)$

Arithmetic Operations: 2’s Complement

Input: two positive integers x & y,

1. We represent the operands in two’s complement.
2. We sum up the two operands and ignore bit n.
3. The result is the solution in two’s complement.

Arithmetic	2’s complement
$x + y$	$x + y$
$x - y$	$x + (2^n - y) = 2^n+(x-y)$
$-x + y$	$(2^n - x) + y =2^n+(-x+y)$

Arithmetic Operations: Example: 4 – 3 = 1

$4_{10} = 0100_2$

$3_{10} = 0011_2$ $\quad\quad$ $-3_{10} \rightarrow 1101_2$

$$\begin{array}{r} 0100 \\ +\ 1101 \\ \hline 10001 \end{array}$$

10001 → 1 (after discarding extra bit)

We discard the extra 1 at the left which is 2^n from 2's complement of -3. Note that bit b_{n-1} is 0. Thus, the result is positive.

Arithmetic Operations: Example: -4 +3 = -1

$4_{10} = 0100_2$ -4_{10} → Using two's comp. → $1011 + 1 = 1100_2$

$3_{10} = 0011_2$ (Invert bits)

1100
+ 0011
1111 → Using two's comp. → 0000 + 1 = 1,
so our answer is -1

If left-most bit is 1, it means that we have a negative number.

Arithmetic Operations: Example: 4 – 3 = 1

$4_{10} = 0100_2$

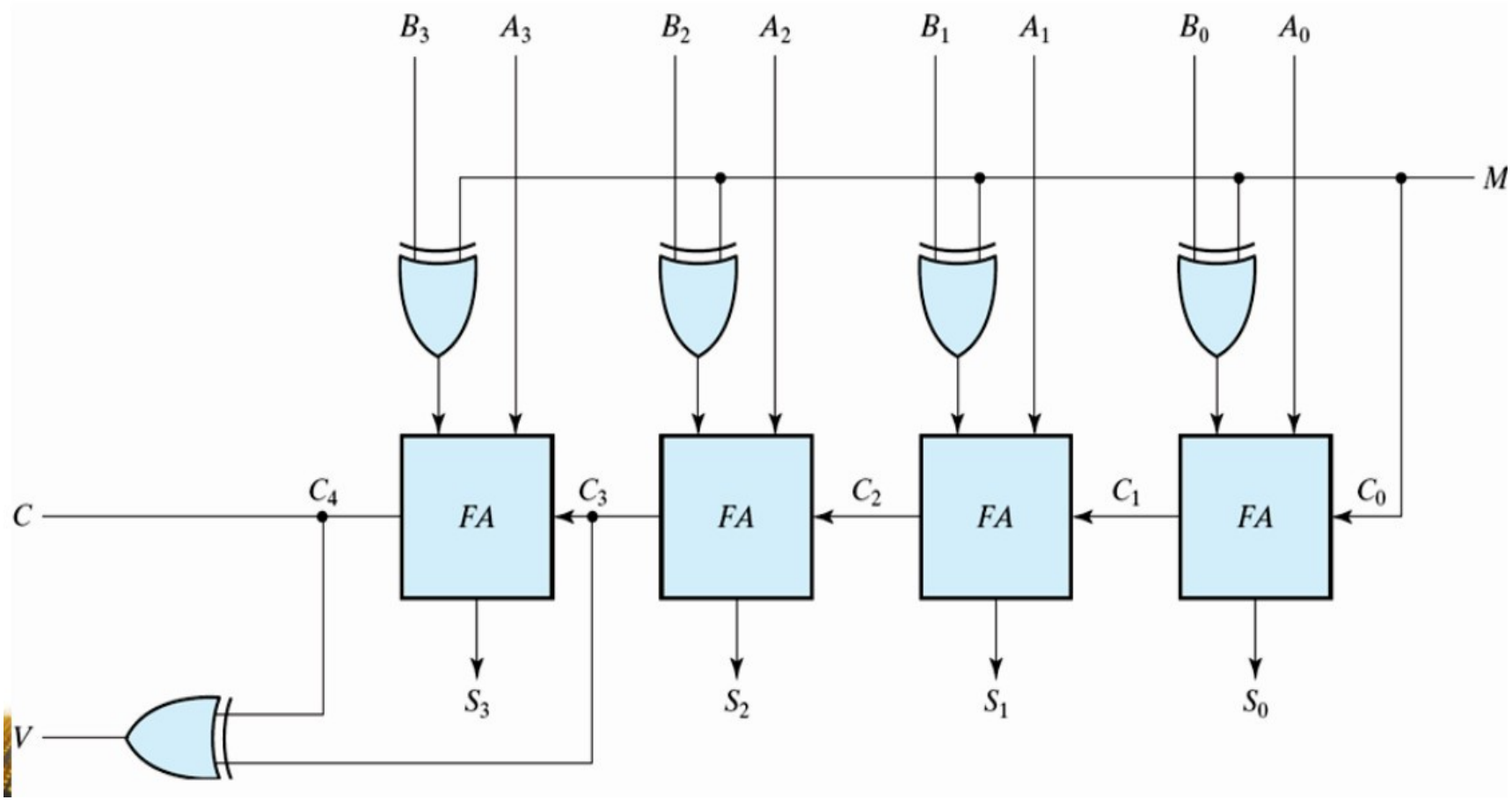
$3_{10} = 0011_2$ $\quad -3_{10} \rightarrow 1100_2$ in one's complement

$$\begin{array}{r} 0100 \\ +\ 1100 \\ \hline 1,0000 \end{array}$$

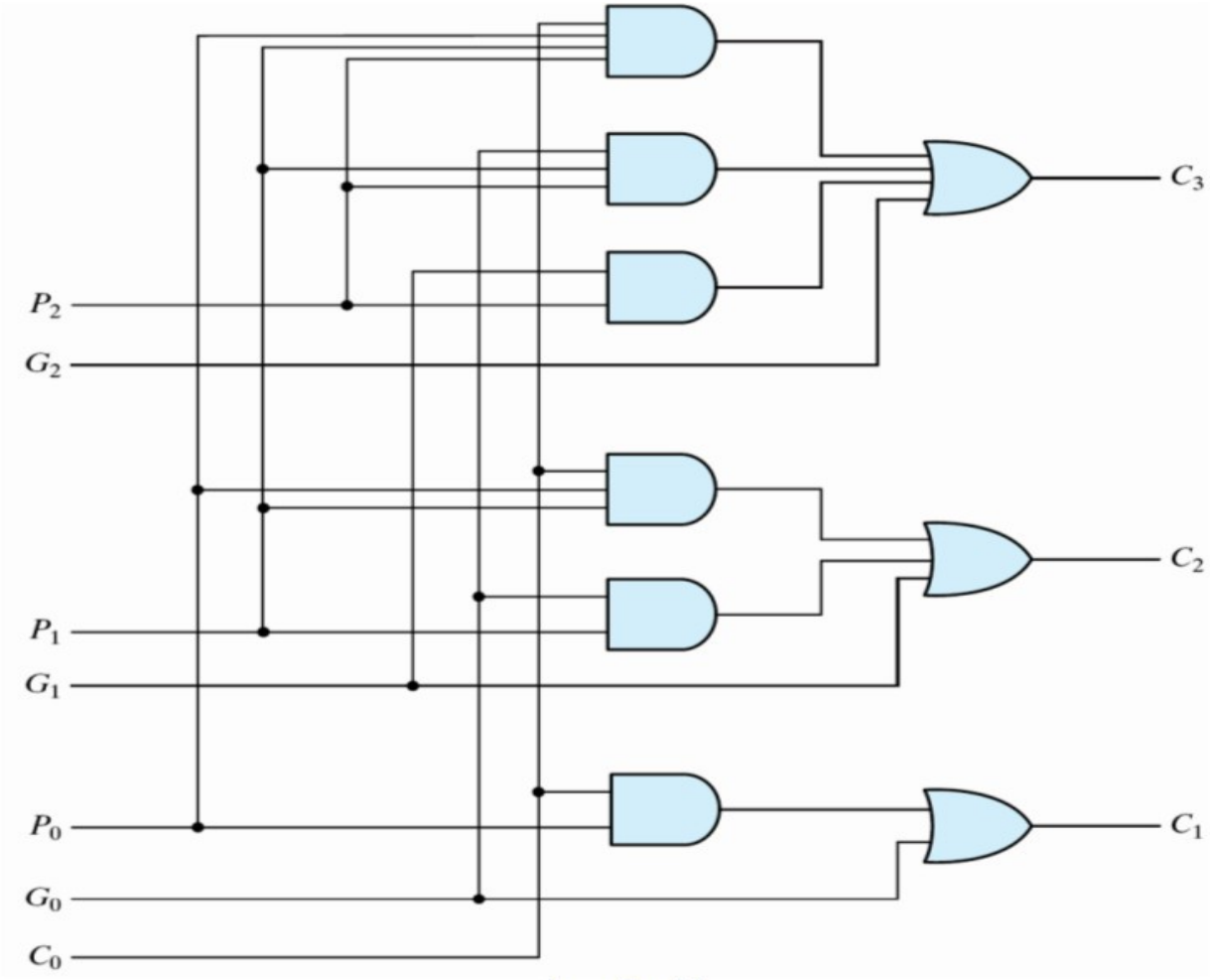
0001(after deleting 2^n-1)

If an end carry occurs, add 1 to least significant digit.

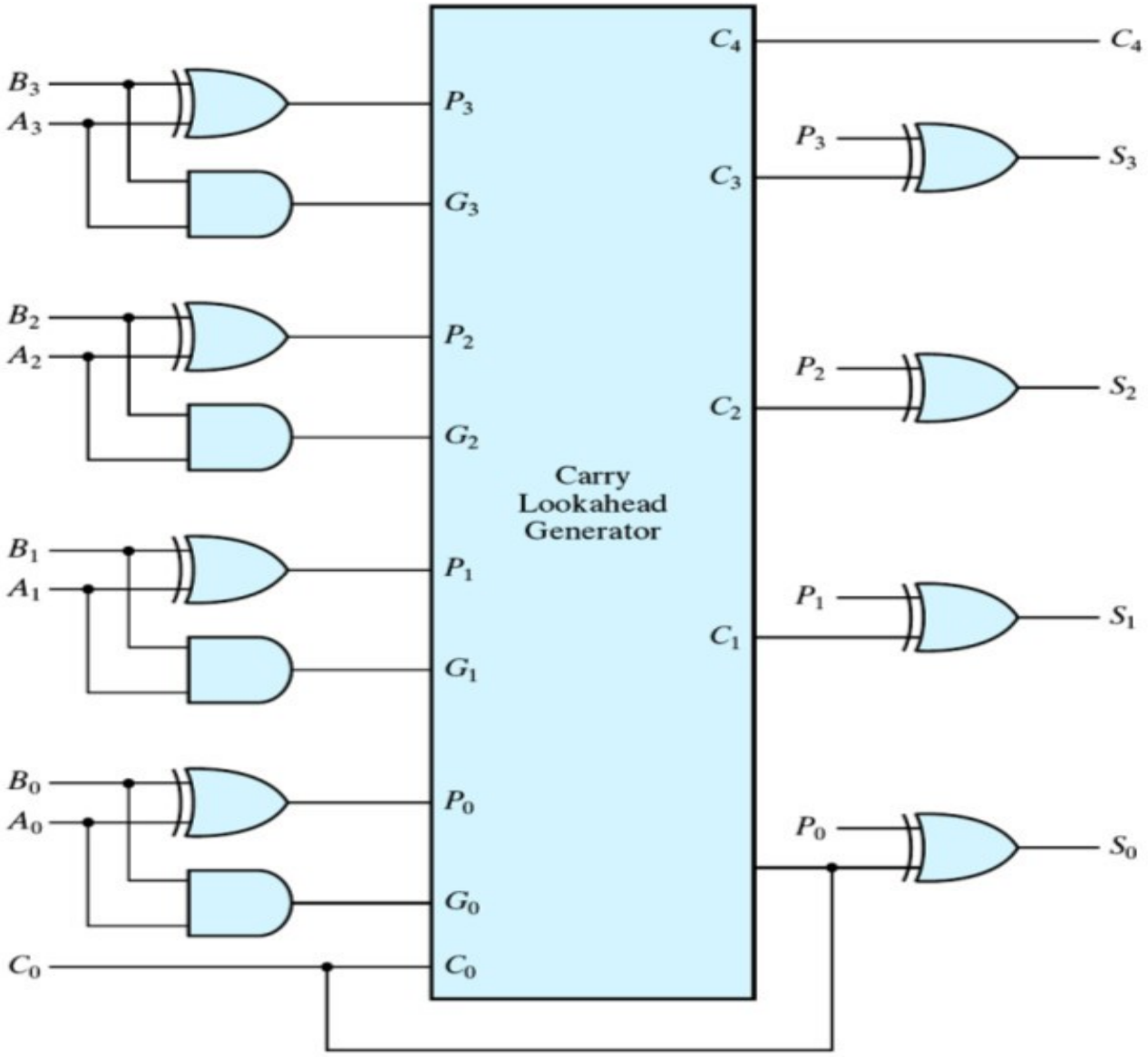
4-bit adder/subtractor



Carry look ahead adder

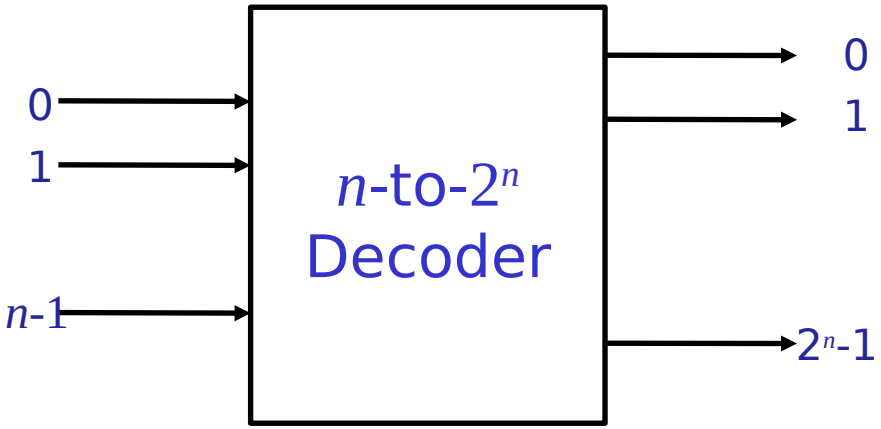


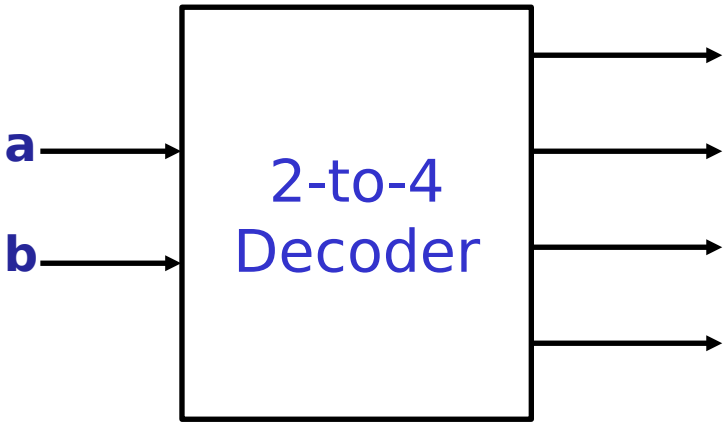
Carry look ahead adder



Decoder

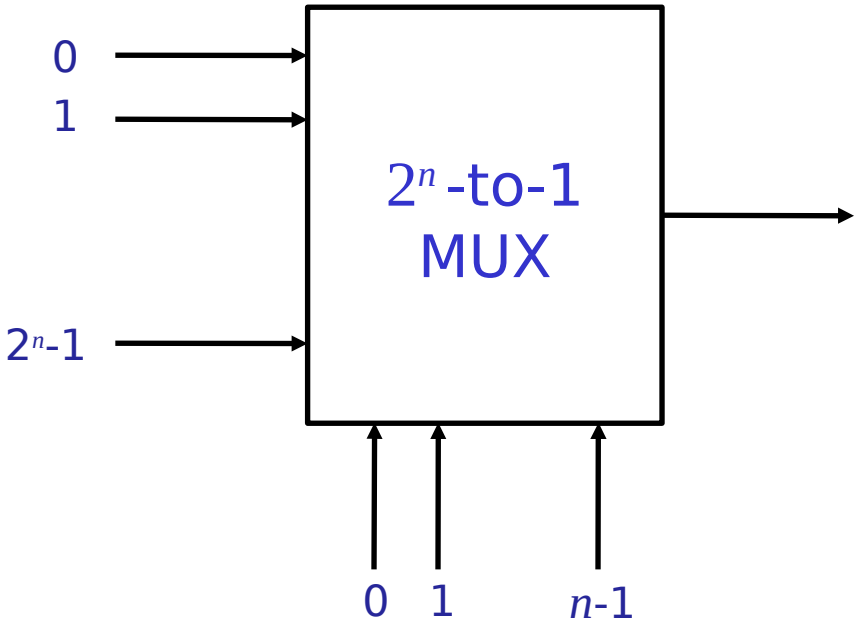
- Circuit that takes an n-bit number as input and uses it to select exactly one of the 2^n output lines

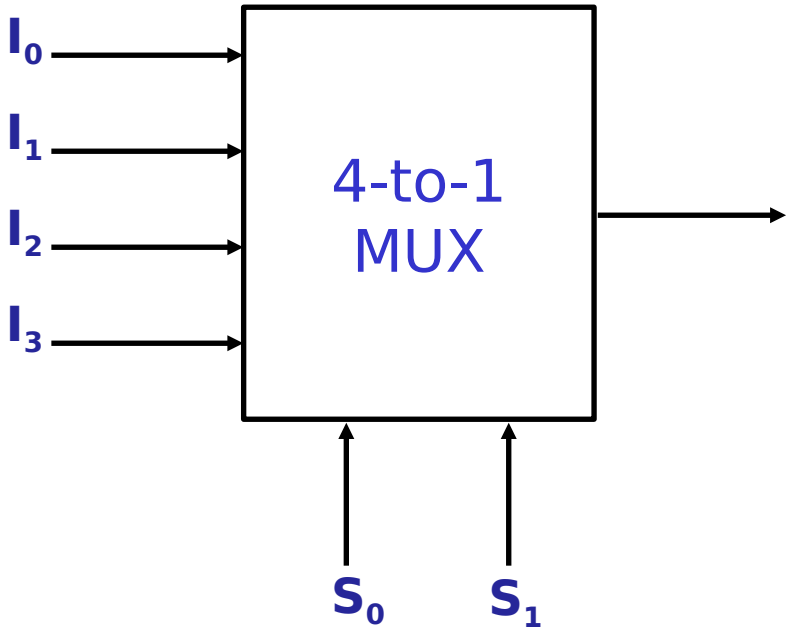




Multiplexers

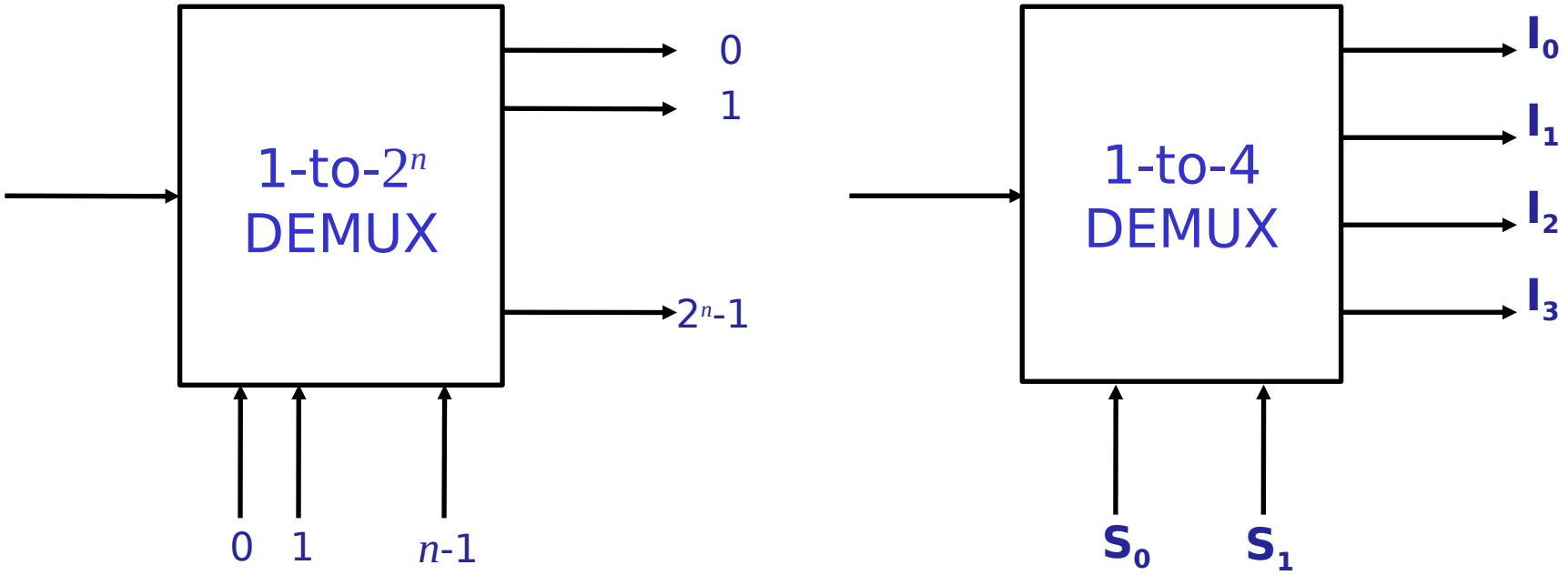
- Multiplexer is a circuit with 2^n data inputs and one data output and n control lines to select one of the data inputs
- The selected input is gated (i.e. routed) to the output



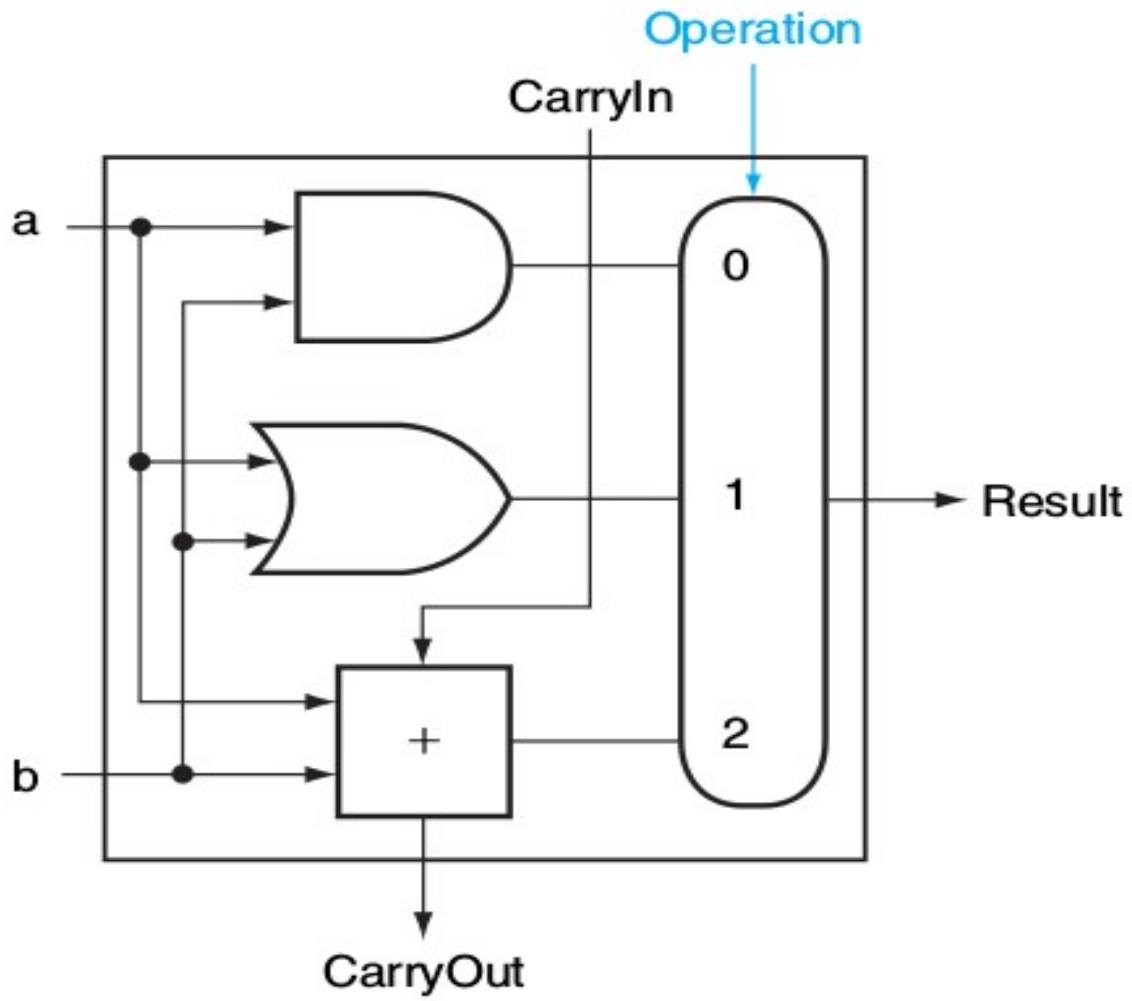


Demultiplexers

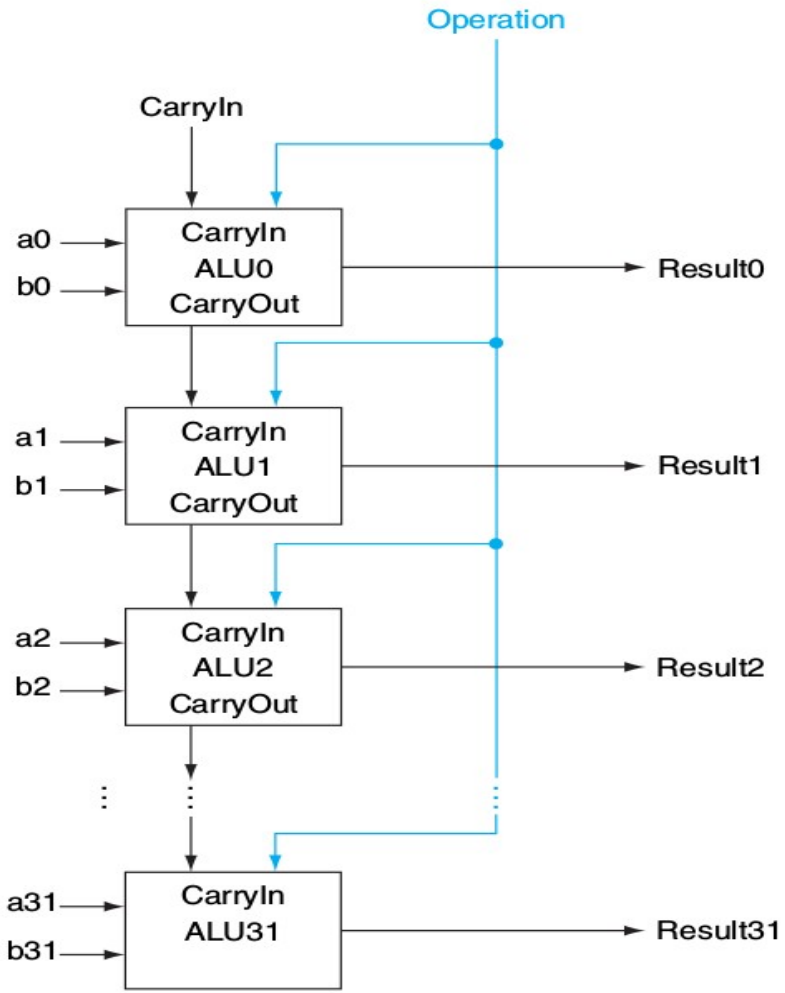
- Inverse of a multiplexer
- Routes its single input signal to one of 2^n outputs, depending on the values of n control lines

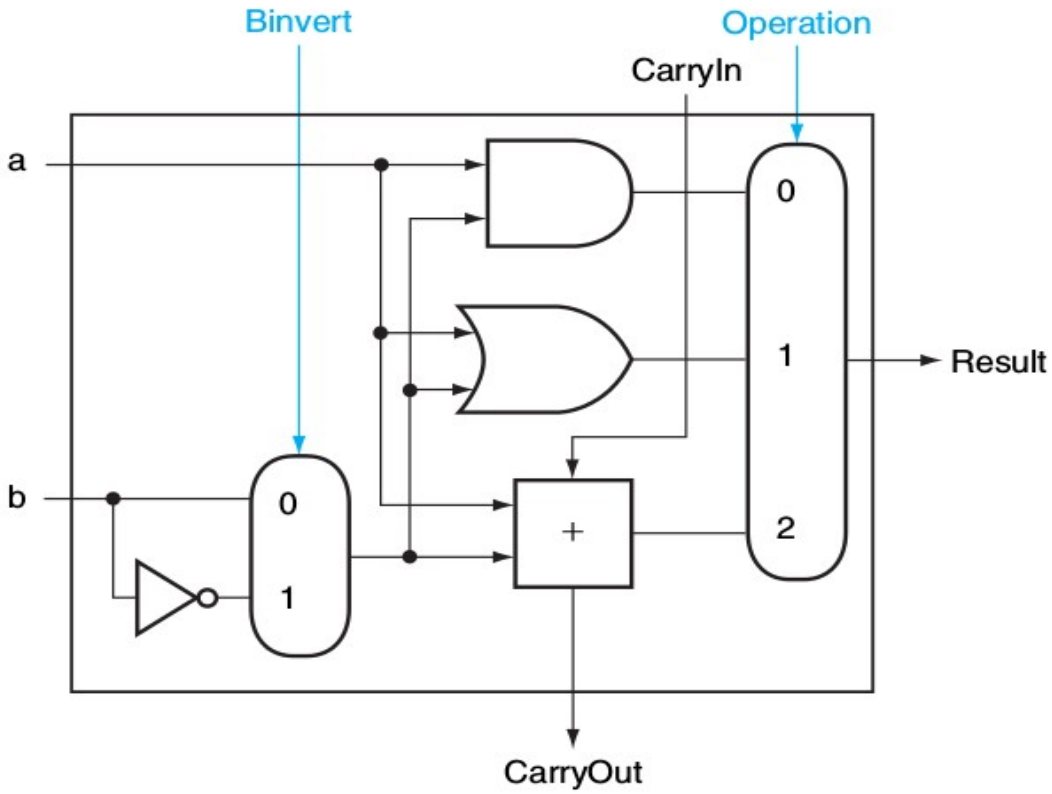


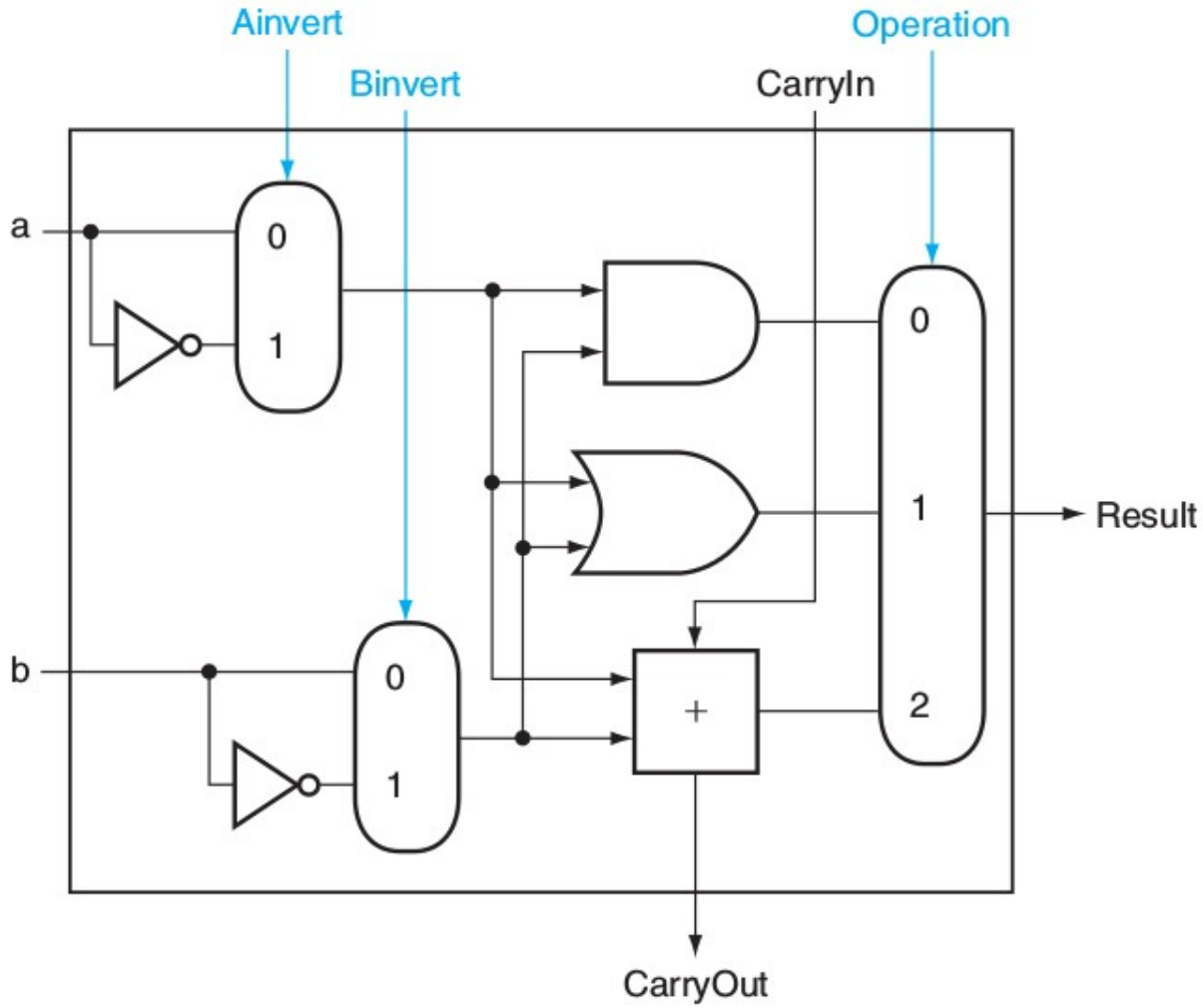
1-bit ALU



David A Patterson and John K. Hennessy, “Computer Organization and Design”, 3rd edition


Operation
CarryIn
a0
b0
CarryIn
ALU0
CarryOut
Result0
a1
b1
CarryIn
ALU1
CarryOut
Result1
a2
b2
CarryIn
ALU2
CarryOut
Result2
a31
b31
CarryIn
ALU31
Result31


Binvert
Operation
CarryIn
a
b
0
1
0
1
2
+
Result
CarryOut

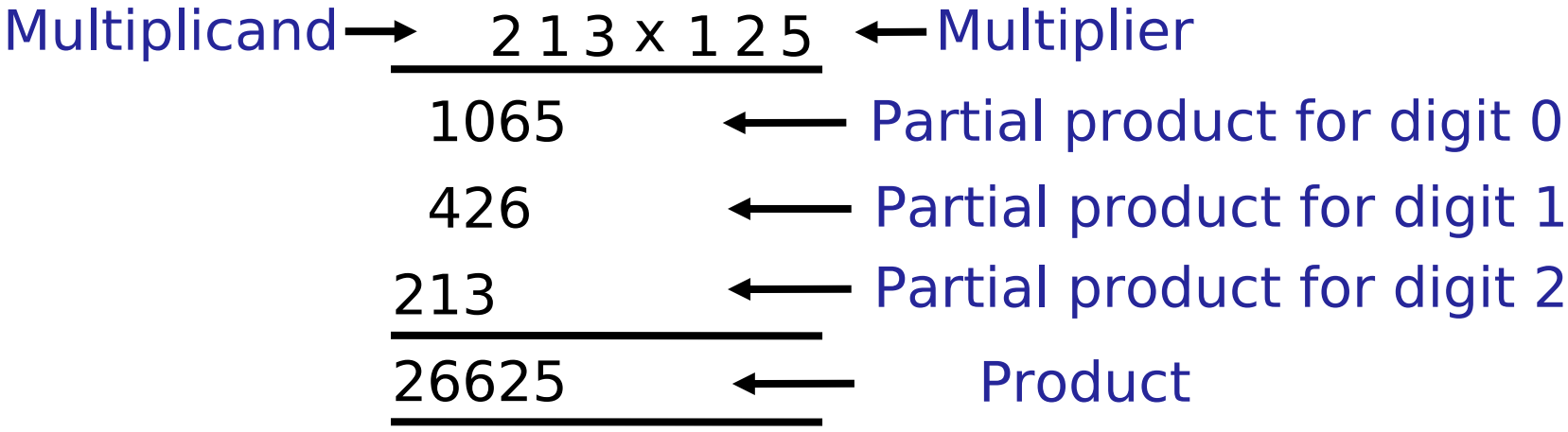


David A Patterson and John K. Hennessy, "Computer Organization and Design", 3rd edition

Digital Ckt for Multiplication

Multiplication of Unsigned Integers

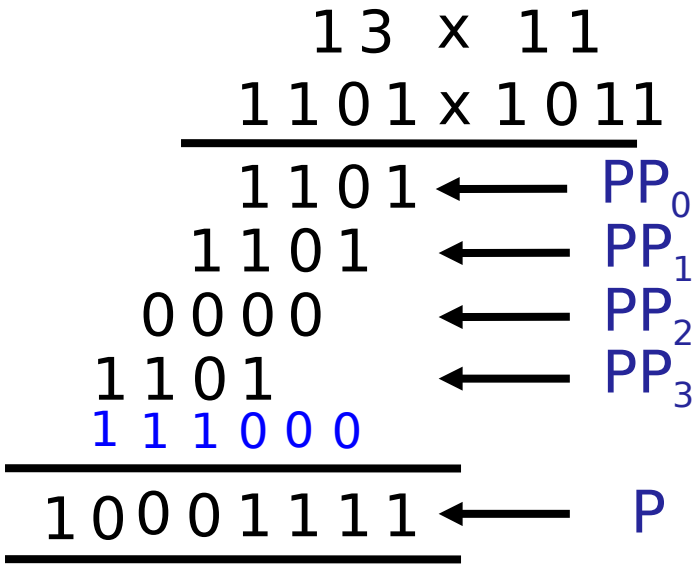
- Two operands: Multiplier and Multiplicand



- Each partial products are placed in such a way that each are shifted according to their weight in the multiplier
- Long-hand-multiplication (paper-and-pencil method)

n-bit Binary Multiplication

- The product of two n-bit integers leads to $2n$-bit product



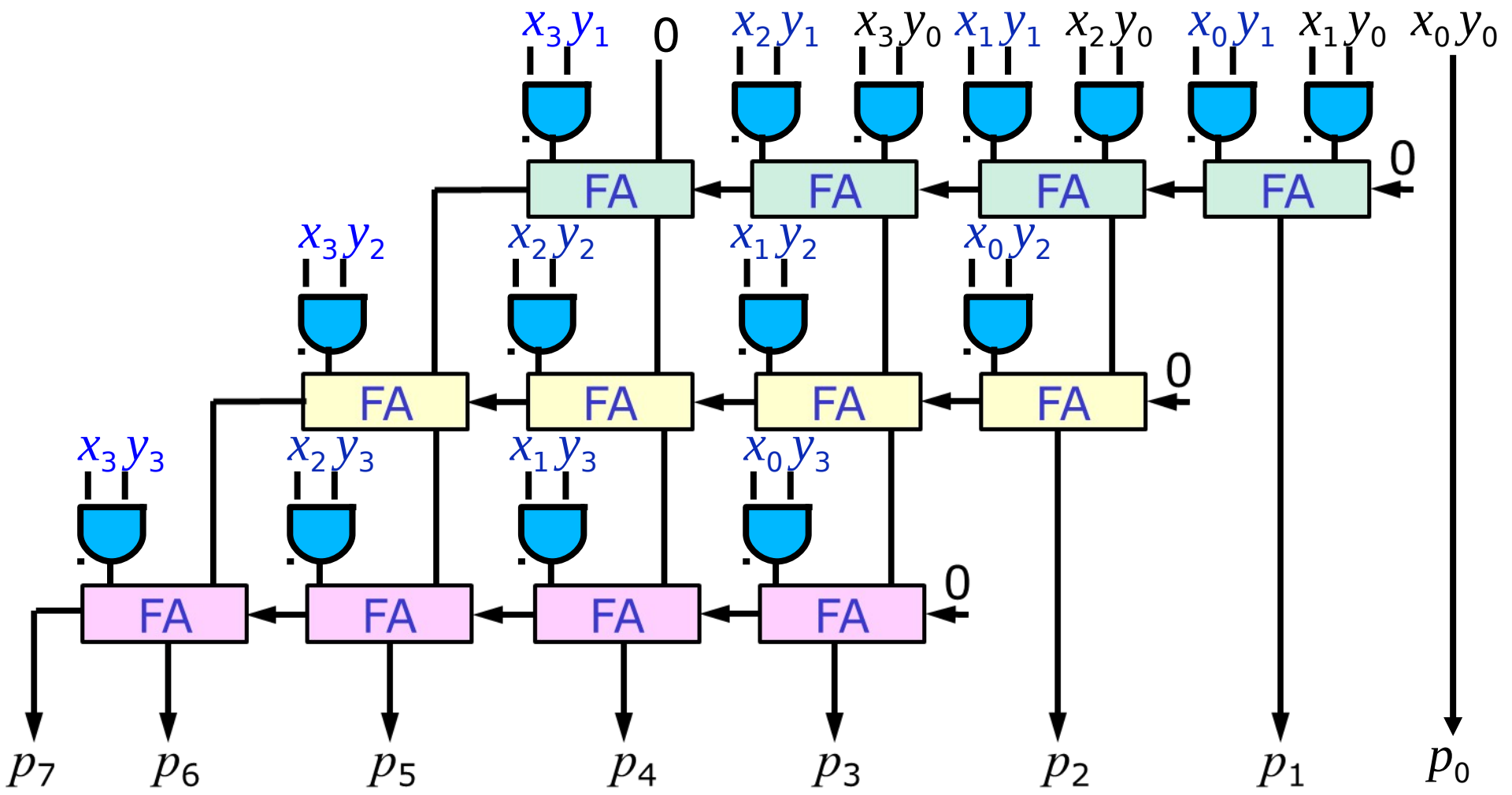
4-bit Multiplication

- $X_{3\text{-}0}$: $x_3 x_2 x_1 x_0$ Multiplicand
- $Y_{3\text{-}0}$: $y_3 y_2 y_1 y_0$ Multiplier

					$x_3 x_2 x_1 x_0$	x	$y_3 y_2 y_1 y_0$		
				$x_3 y_0$	$x_2 y_0$	$x_1 y_0$	$x_0 y_0$	←	PP_0
			$x_3 y_1$	$x_2 y_1$	$x_1 y_1$	$x_0 y_1$		←	PP_1
		$x_3 y_2$	$x_2 y_2$	$x_1 y_2$	$x_0 y_2$			←	PP_2
	$x_3 y_3$	$x_2 y_3$	$x_1 y_3$	$x_0 y_3$				←	PP_3
p_7	p_6	p_5	p_4	p_3	p_2	p_1	p_0	←	P

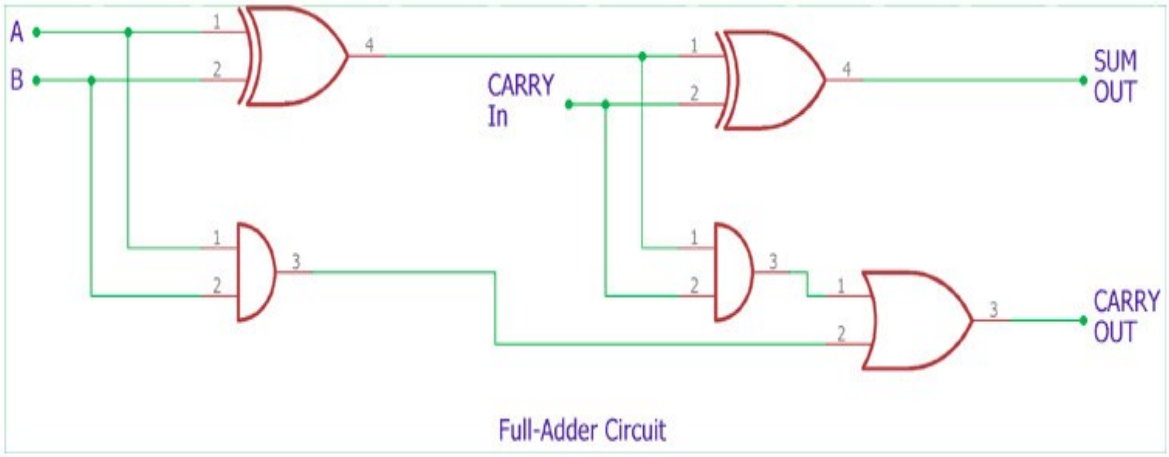
- Multiplier is designed using AND gates and 1-bit Full Adder circuits
- Combinational array multiplier circuit

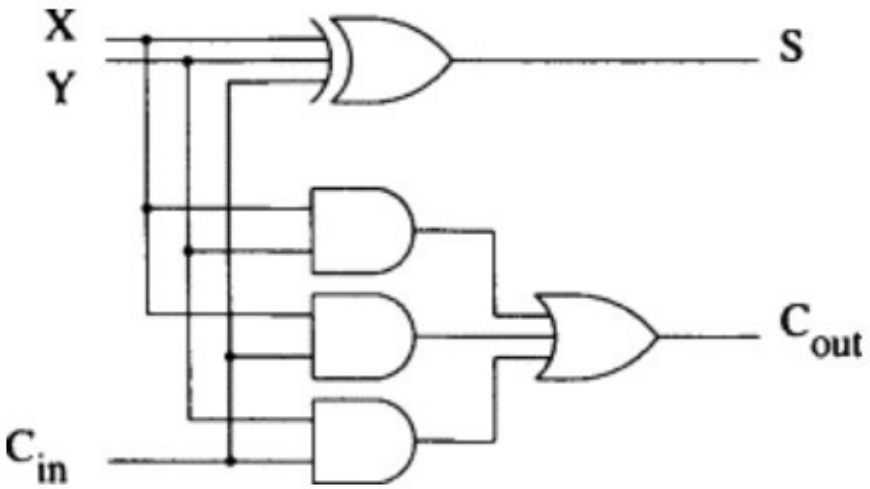
Combinational Array Multiplier Circuit



- Uses ripple carry adder

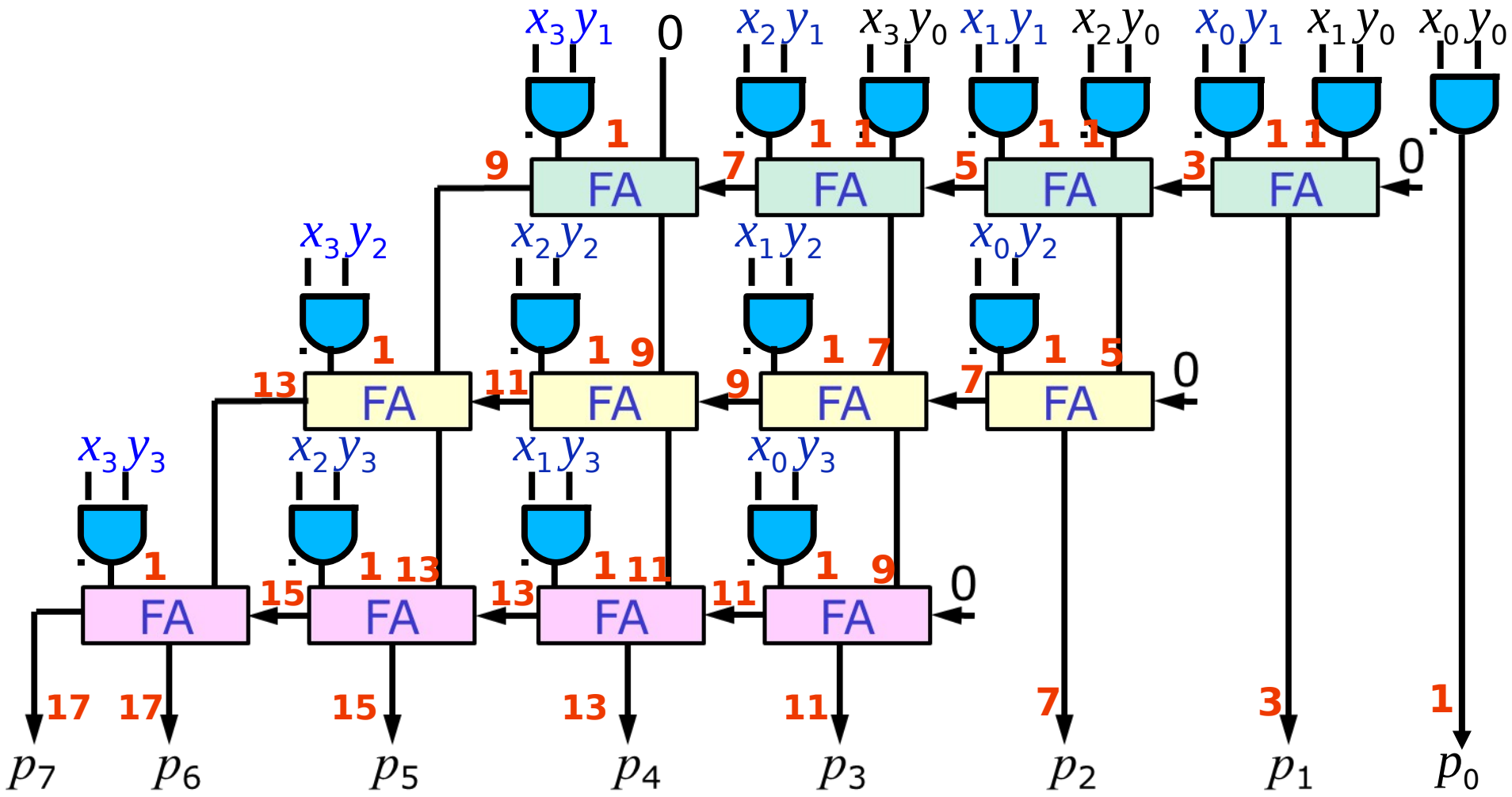
Full Adder





Combinational Array Multiplier

- Propagation delay through
  - AND gate : 1 t_{pd} 1-bit Full Adder: 2 t_{pd}



- Latency of 4-bit multiplication: 17 t_{pd}

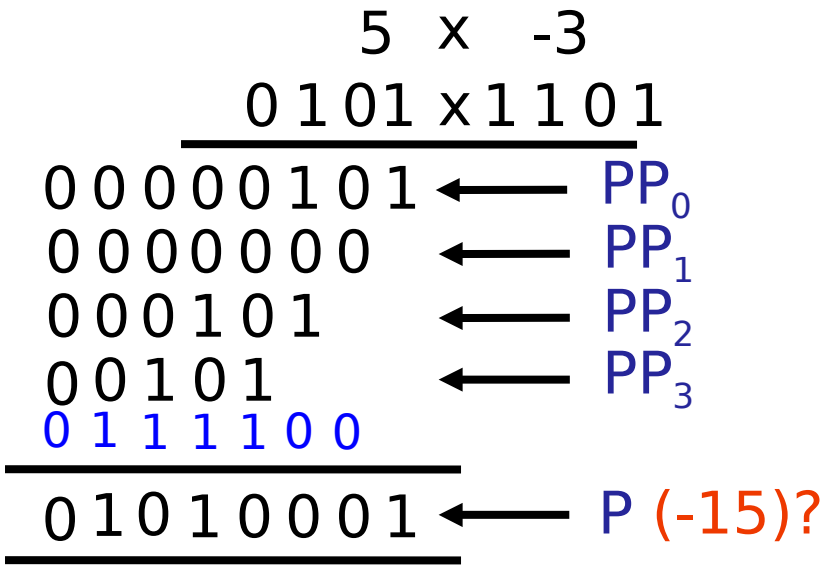
Latency of n-bit Multiplication

- Multiplication of two n-bit multiplication involve
  - (n-1) levels of addition
  - n Full adders at each level
- Input for first level of addition is obtained at: 1 t_{pd}
- First input for 2nd level of addition is obtained at: 5 t_{pd}
- First input for 3rd level of addition is obtained at: 9 t_{pd}

- Total latency: $4n-7 + 2n =$ **$(6n-7)$ t_{pd}**

Signed Integer Multiplication

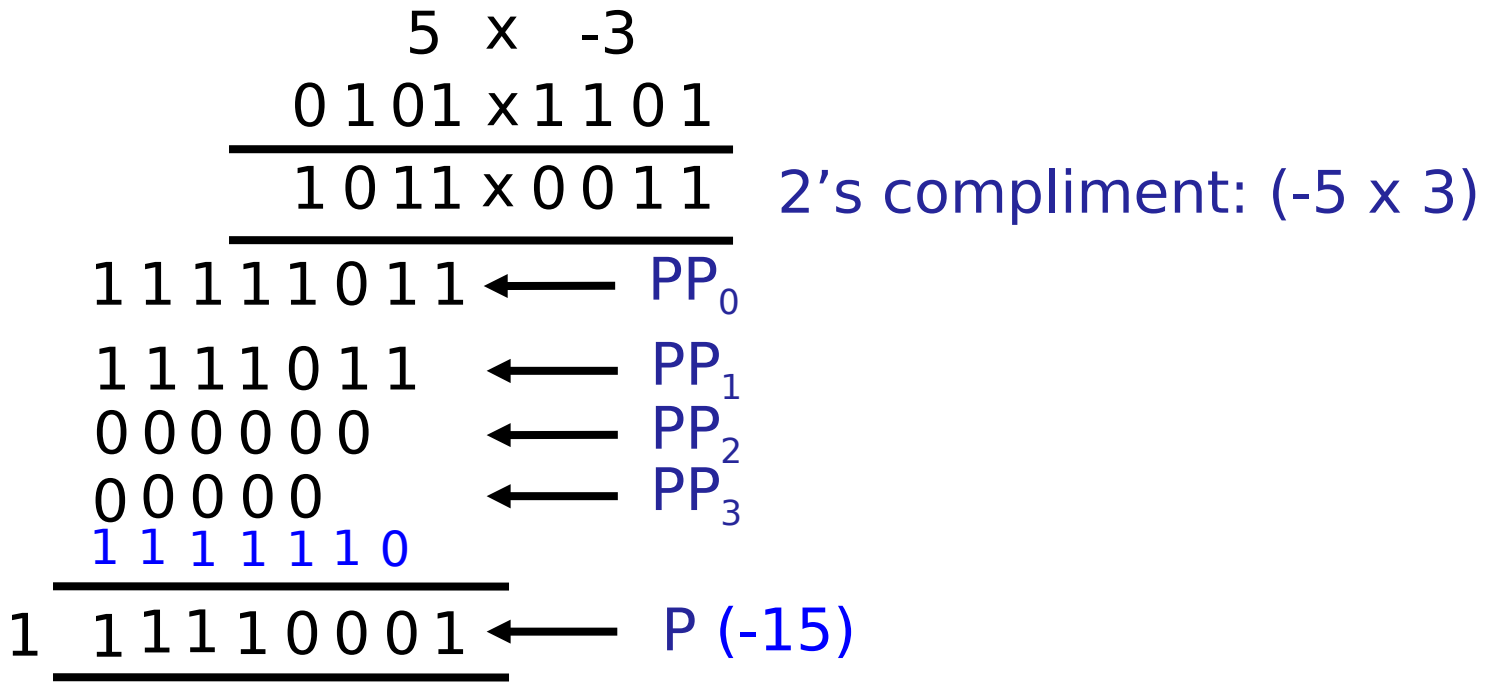
- Consider: -ve multiplier and +ve multiplicand



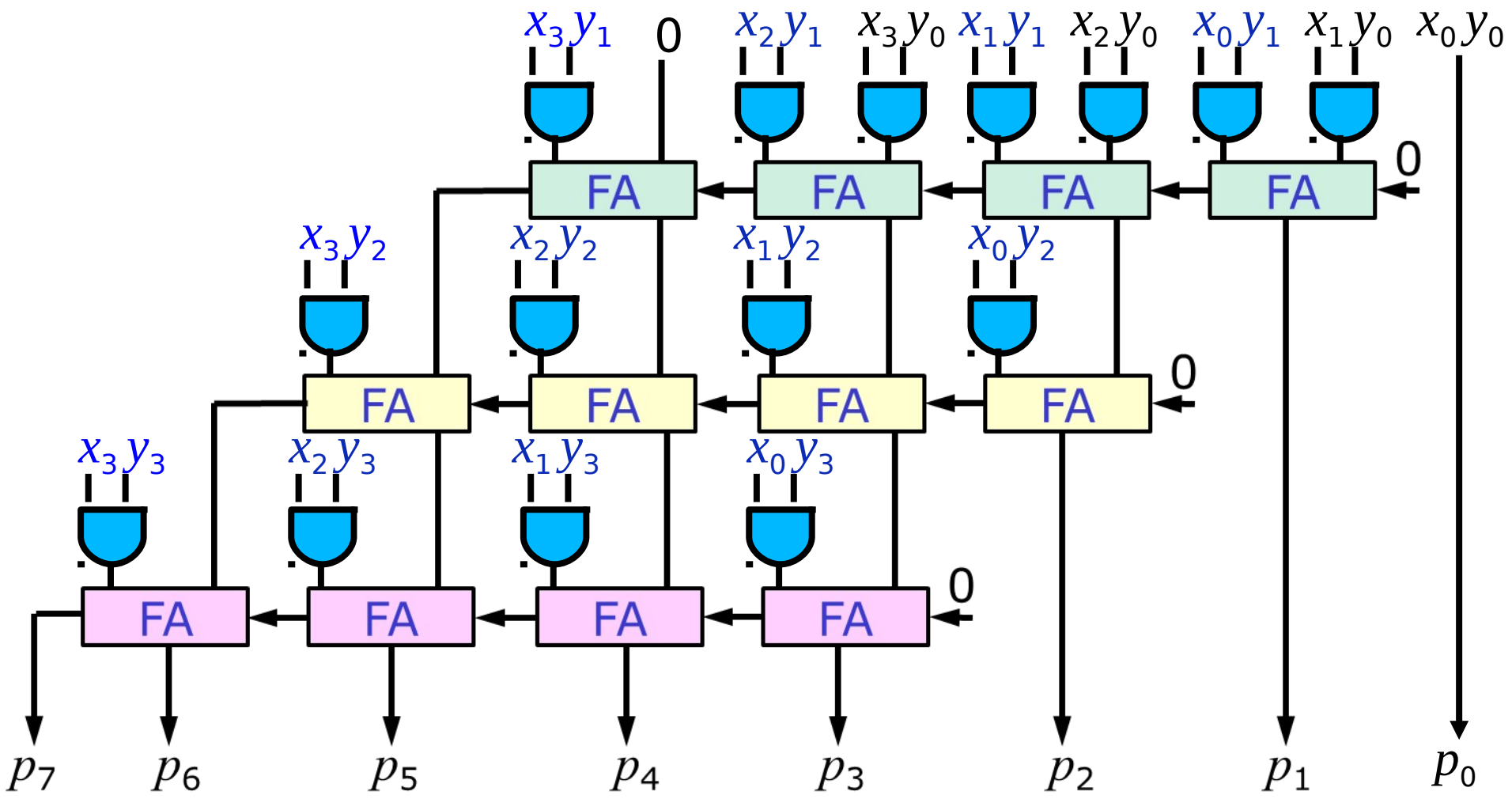
- Does not lead to correct solution
- For negative multiplier case, form 2's complement of both multiplier and multiplicand and proceed as in the case of positive multiplier

Signed Integer Multiplication

- Consider: -ve multiplier and +ve multiplicand

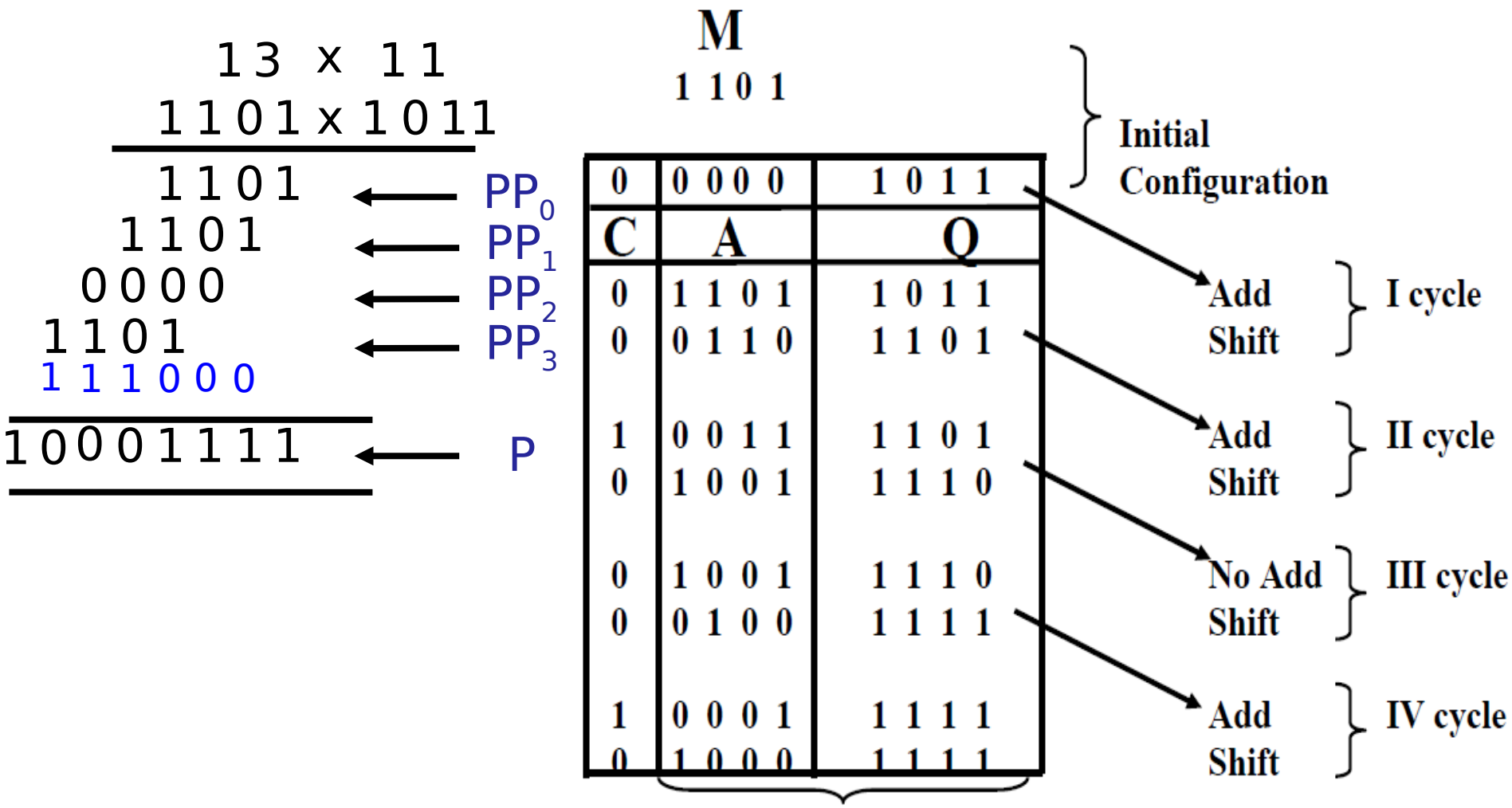


Combinational Array Multiplier Circuit

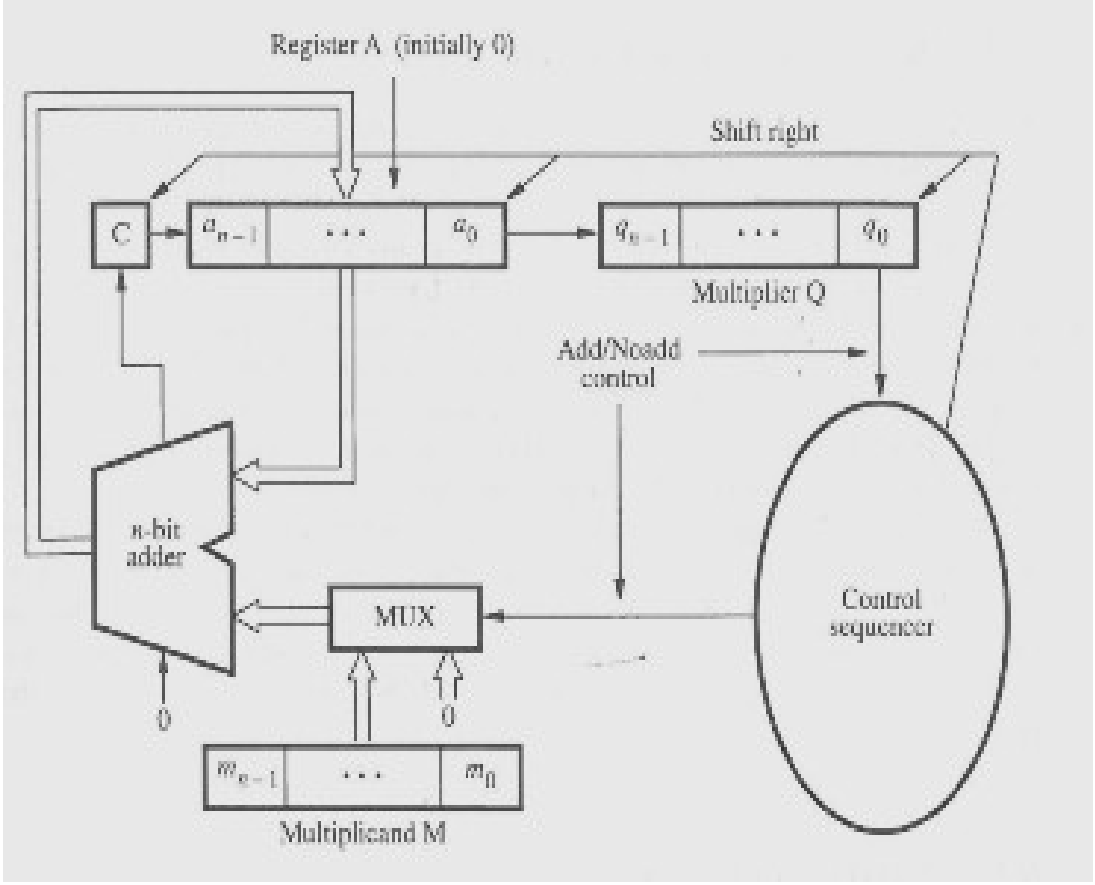


- Uses ripple carry adder

Sequential algoritm for *n*-bit Multiplication



Sequential ckt for n-bit Multiplication



M
1 1 0 1

C	A	Q	
0	0 0 0 0	1 0 1 1	Initial Configuration
0	1 1 0 1	1 0 1 1	Add
0	0 1 1 0	1 1 0 1	Shift — I cycle
1	0 0 1 1	1 1 0 1	Add
0	1 0 0 1	1 1 1 0	Shift — II cycle
0	1 0 0 1	1 1 1 0	No Add
0	0 1 0 0	1 1 1 1	Shift — III cycle
1	0 0 0 1	1 1 1 1	Add
0	1 0 0 0	1 1 1 1	Shift — IV cycle

Booth Recording of Multiplier Operand

- This approach treat both positive and negative integers (or operands) in 2's compliment form uniformly
- It records (encodes) the multiplier operand in Booth multiplier form
- Booth multiplier recording table:

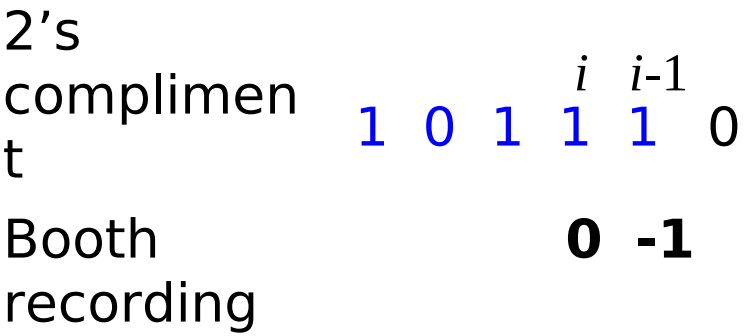
Multiplier		Version of multiplicand selected by bit *i*
bit *i*	bit *i-1*	
0	0	0x multiplicand
0	1	+1 x multiplicand
1	0	-1 x multiplicand
1	1	0x multiplicand

2's compliment	1	0	1	1	1	*i* 0	*i*-1
Booth recording						**-1**	

Booth Recording of Multiplier Operand

- This approach treat both positive and negative integers (or operands) in 2's compliment form uniformly
- It records (encodes) the multiplier operand in Booth multiplier form
- Booth multiplier recording table:

Multiplier		Version of multiplicand selected by bit *i*
bit *i*	bit *i-1*	
0	0	0x multiplicand
0	1	+1 x multiplicand
1	0	-1 x multiplicand
1	1	0x multiplicand



Booth Recording of Multiplier Operand

Multiplier		Version of multiplicand selected by bit *i*
bit *i*	bit *i-1*	
0	0	0x multiplicand
0	1	+1 x multiplicand
1	0	-1 x multiplicand
1	1	0x multiplicand

			i	*i*-1		
2's compliment	1	0	1	1	1	0
Booth recording			**0**	**0**	**-1**	

Booth Recording of Multiplier Operand

Multiplier		Version of multiplicand selected by bit *i*
bit *i*	bit *i-1*	
0	0	0x multiplicand
0	1	+1 x multiplicand
1	0	-1 x multiplicand
1	1	0x multiplicand

2's complement

i	*i*-1				
1	0	1	1	1	0

Booth recording

1	**0**	**0**	**-1**

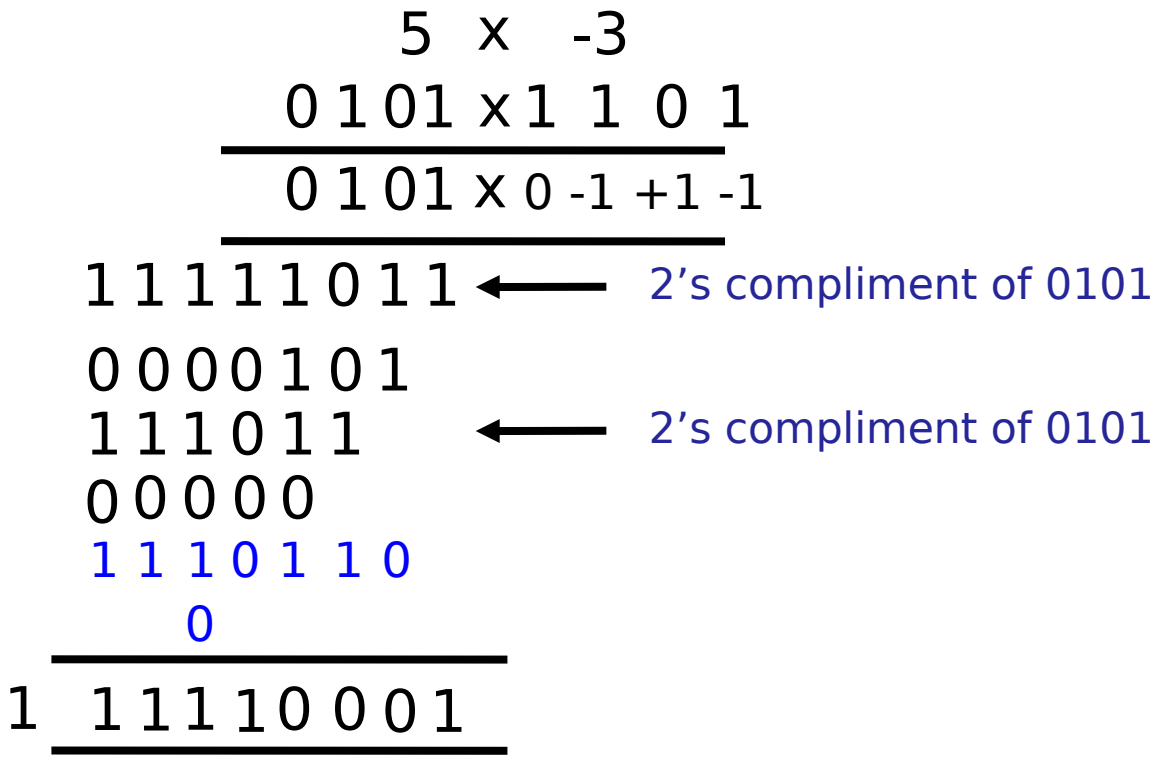
Booth Recording of Multiplier Operand

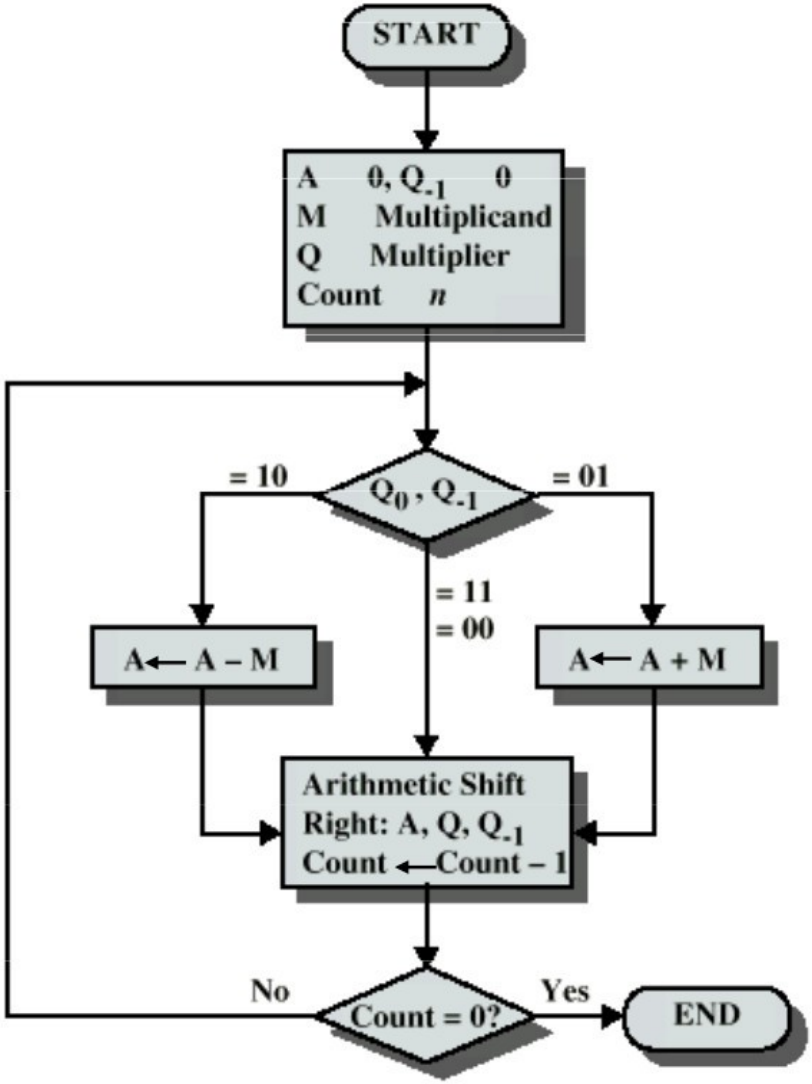
Multiplier		Version of multiplicand selected by bit *i*
bit *i*	bit *i-1*	
0	0	0x multiplicand
0	1	+1 x multiplicand
1	0	-1 x multiplicand
1	1	0x multiplicand

	i	i-1				
2's complement	1	0	1	1	1	0
Booth recording	**-1**	**1**	**0**	**0**	**-1**	

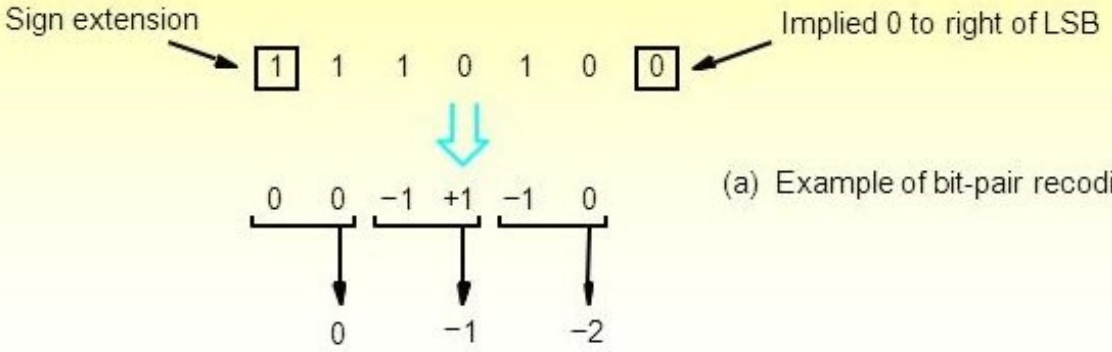
Signed Integer Multiplication

- Consider: -ve multiplier and +ve multiplicand
- Booth recording of multiplier




START
A 0, Q-1 0
M Multiplicand
Q Multiplier
Count n
Q0 , Q-1
= 10
= 01
= 11
= 00
A ← A – M
A ← A + M
Arithmetic Shift
Right: A, Q, Q-1
Count ← Count – 1
Count = 0?
No
Yes
END

Multiplier Bit-Pair Recoding



(a) Example of bit-pair recoding derived from Booth recoding

Multiplier bit-pair		Multiplier bit on the right	Multiplicand selected at position i
$i+1$	i	$i-1$	
0	0	0	$0 \times M$
0	0	1	$+1 \times M$
0	1	0	$+1 \times M$
0	1	1	$+2 \times M$
1	0	0	$-2 \times M$
1	0	1	$-1 \times M$
1	1	0	$-1 \times M$
1	1	1	$0 \times M$

Multiplication with Carry Save Addition of Summands

Carry Save Addition (CSA) of Summands

- In combinational array (CA) multiplication logic, the n partial products i.e. n summands are added sequentially
- Can we do the parallel addition of summands?
- Idea:
  - Avoid the carries ripple along rows
  - Carries can be saved and introduced into the next row
- Example:

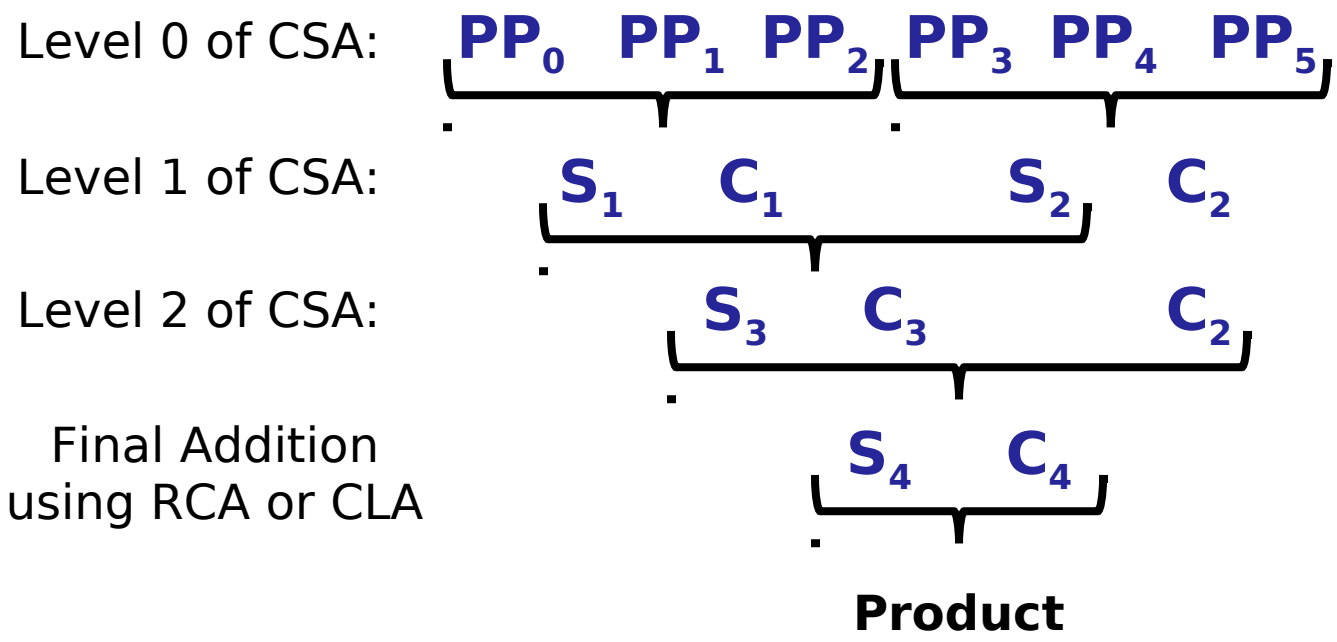
	1 1 0 1 1
	0 1 0 1 0
S	1 0 0 0 1
C	0 1 0 1 0 0

CSA of Summands

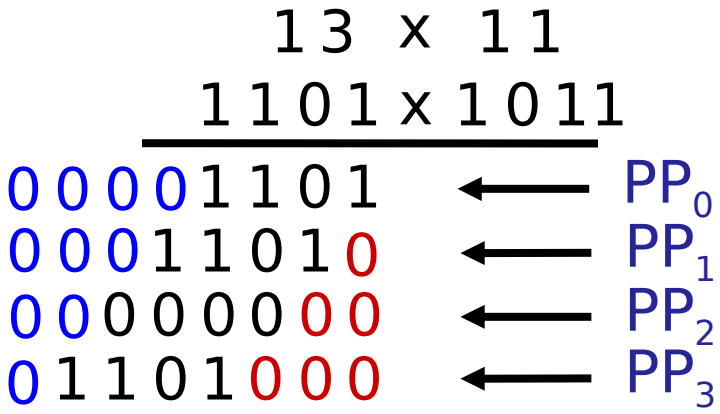
- Group the summands in threes and then perform CSA
- This generates the further set of sum (S) and carry (C) vectors
- Continue this process until there are only two vectors (summands) remaining
- The final summands are added in a RCA or CLA to produce desired product
- Note:
  - Each partial products are shifted versions of multiplicands
  - Partial products, S and C vectors are considered as summands
  - Each of these summands are shifted versions
- Multiplication using CSA can be applied to both unsigned and signed integer

Illustration of CSA

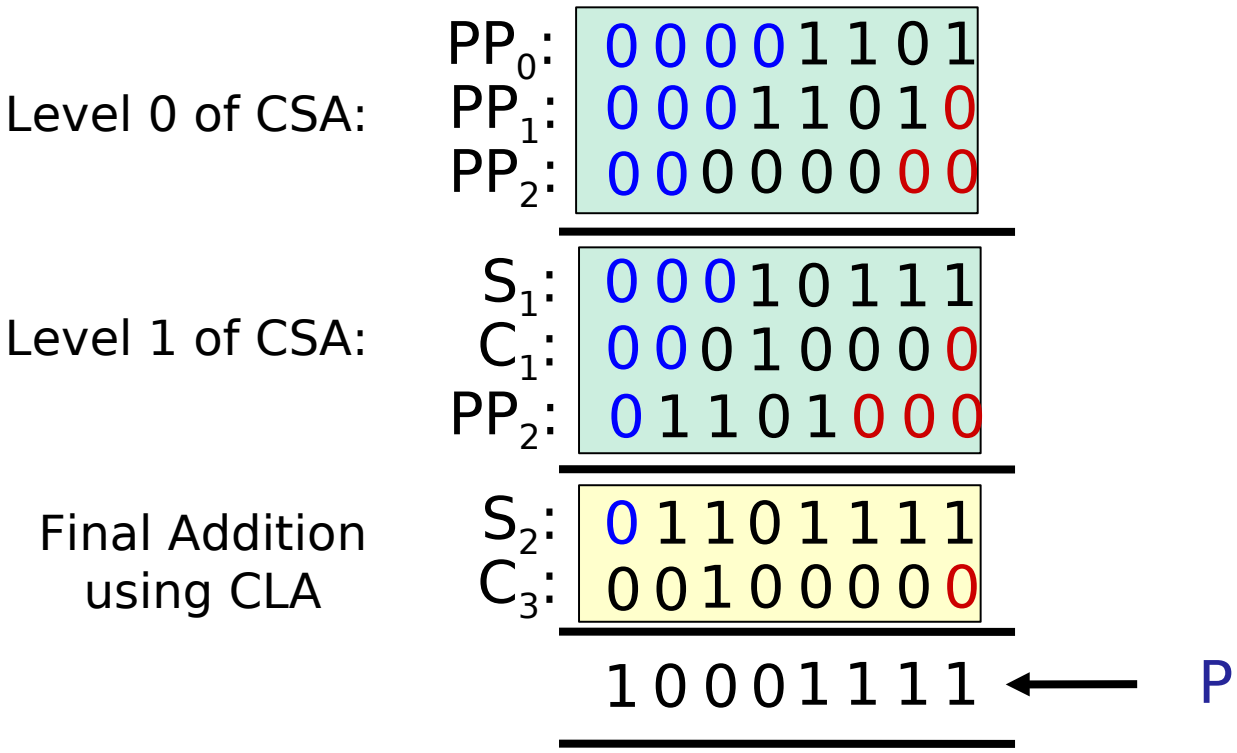
- Consider 6-bit multiplication that results in 6 partial products (summands)



4-bit Unsigned Multiplication using CSA



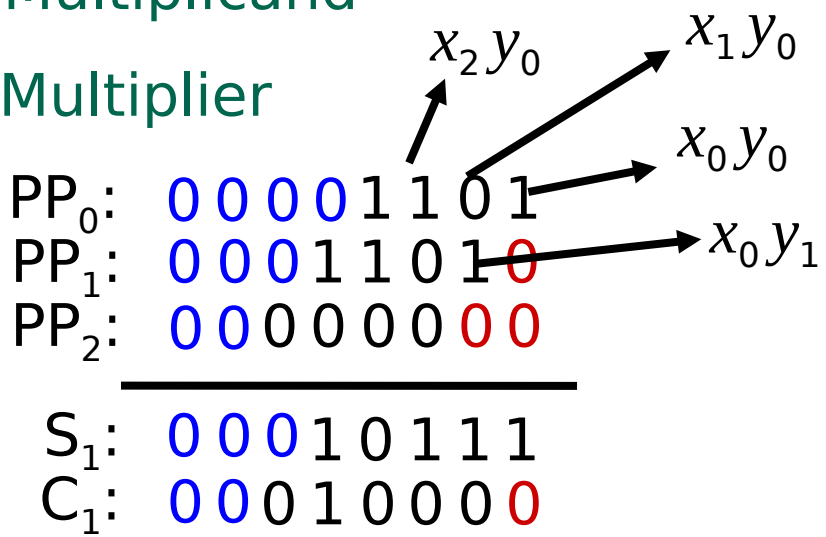
4-bit Unsigned Multiplication using CSA



4-bit Unsigned Multiplication using CSA

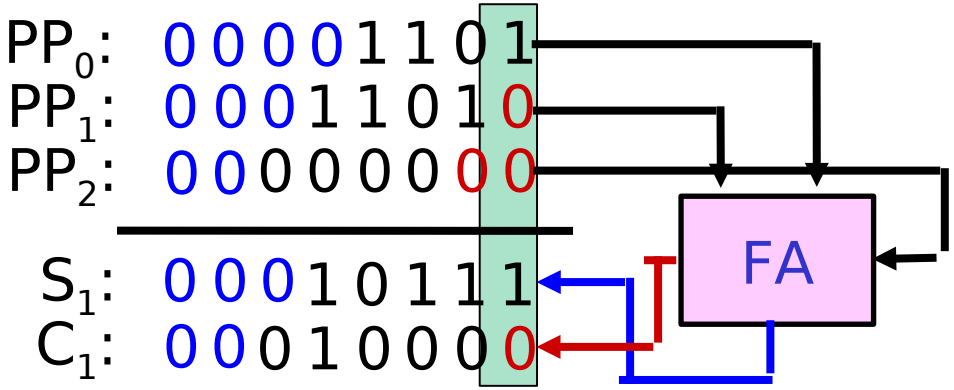
- $X_{3\text{-}0}$: $x_3 x_2 x_1 x_0$ Multiplicand
- $Y_{3\text{-}0}$: $y_3 y_2 y_1 y_0$ Multiplier

Level 0 of CSA:

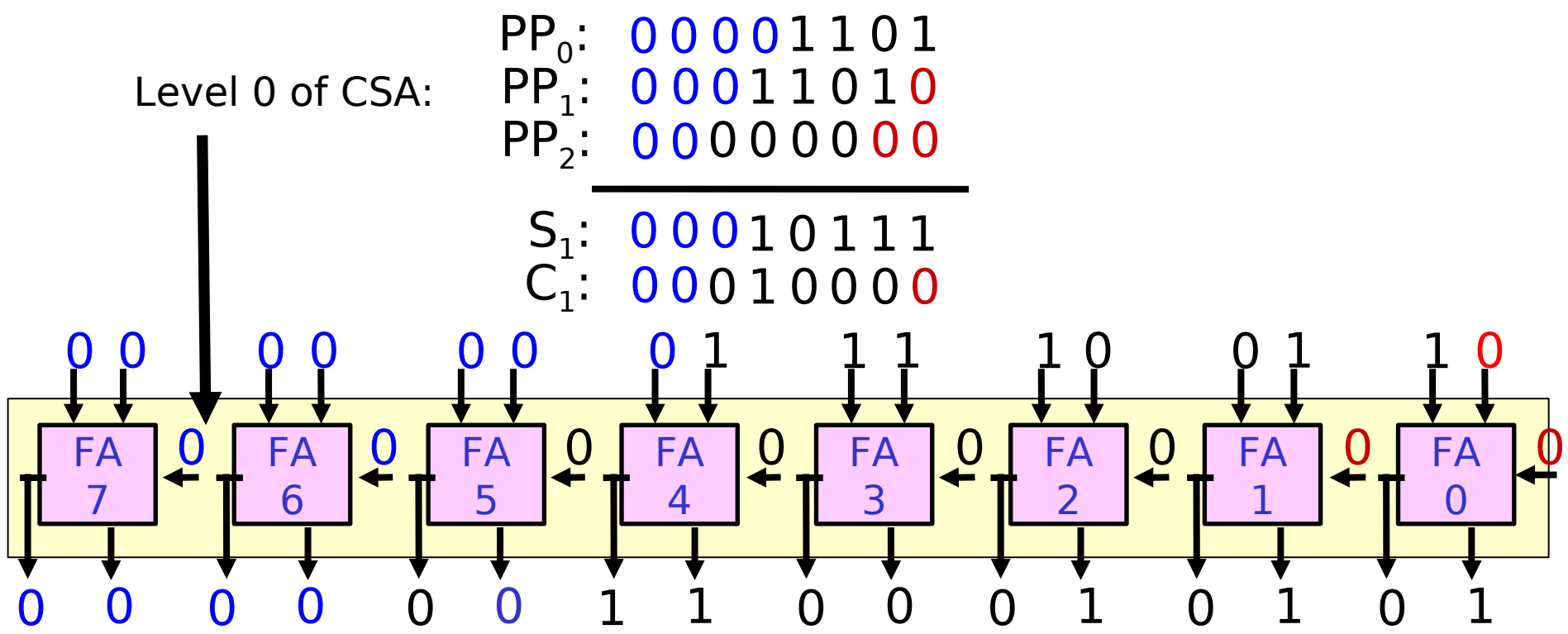


4-bit Unsigned Multiplication using CSA

Level 0 of CSA:



4-bit Unsigned Multiplication using CSA



- Propagation delay for CSA (given the summands): 2 t_{pd}
- S and C vectors are generated in 1 full adder delay

Latency of n-bit Multiplication using CSA

- We have seen that the latency of combinational array multiplication is of two n-bit multiplication: **$(6n-7)$** t_{pd}
- Partial products are generated at : 1 t_{pd}
- Delay through full adder (FA) : 2 t_{pd}
- Hence, delay for CSA at a level i is : 2 t_{pd}
- Let m be the number of CSA levels
- Delay through m CSA levels : $2m$ t_{pd}
- Last level uses $2m$-bit adder with l-bit CLA
- So, addition in the last stage has delay : $\left(2\left(\frac{2n}{l}\right)+2\right)$ t_{pd}
- Total latency: $1+2m+\left(\frac{4n}{l}\right)+2=\left(\mathbf{2m}+\left(\frac{\mathbf{4n}}{l}\right)+\mathbf{3}\right)$ t_{pd}
- When n=4, m=2 and let l=4 for CLA
  - Latency = 11 t_{pd}

Latency of n-bit Multiplication using CSA

- We have seen that the latency of combinational array multiplication is of two n-bit multiplication: **$(6n-7)$** t_{pd}
- Partial products are generated at : 1 t_{pd}
- Delay through full adder (FA) : 2 t_{pd}
- Hence, delay for CSA at a level i is : 2 t_{pd}
- Let m be the number of CSA levels
- Delay through m CSA levels : $2m$ t_{pd}
- Last level uses $2m$-bit adder with l-bit CLA
- So, addition in the last stage has delay : $\left(2\left(\frac{2n}{l}\right)+2\right)$ t_{pd}
- Total latency: $1+2m+\left(\frac{4n}{l}\right)+2=\left(\mathbf{2m}+\left(\frac{\mathbf{4n}}{l}\right)+\mathbf{3}\right)$ t_{pd}
- When n=4, m=2 and let l=4 for CLA
  - Latency = 11 t_{pd}

Latency of n-bit Multiplication using CSA

- We have seen that the latency of combinational array multiplication is of two n-bit multiplication: **$(6n-7)$** t_{pd}
- Partial products are generated at : 1 t_{pd}
- Delay through full adder (FA) : 2 t_{pd}
- Hence, delay for CSA at a level i is : 2 t_{pd}
- Let m be the number of CSA levels
- Delay through m CSA levels : $2m$ t_{pd}
- Last level uses $2m$-bit adder with l-bit CLA
- So, addition in the last stage has delay : $\left(2\left(\frac{2n}{l}\right)+2\right)$ t_{pd}
- Total latency: $1+2m+\left(\frac{4n}{l}\right)+2=\left(\mathbf{2m}+\left(\frac{\mathbf{4n}}{l}\right)+\mathbf{3}\right)$ t_{pd}
- When n=4, m=2 and let l=4 for CLA
  - Latency = 11 t_{pd}

Issue in CSA

- How to determine the m, the number of levels in CSA

- In general: $m \approx 1.7\ log_2 k - 1.7$ where k is number of initial summands