classmate
Date
Page

Fixed Point Arithmatic : Addition & Subtraction

In a Computer, the basic arithmatic operations are Addition & Subtraction.

Rule of Binary Addition is :

0+0 = 0
0+1 = 1
1+0 = 1
1+1 = 0 (carry 1)

```
0000 0
0001 1
0010 2
0011 3
0100 4
0101 5
0110 6
0111 7
1000 8
1001 9
1010 (10)
1011 (11)
1100 (12)
1101
1110
1111
```

```
  1
+ 1
----
 10
```

Examples of unsigned binary Addition are

```
  0000      1000      1108
+ 0101    + 0101    + 0101
------    ------    -------
  0101      1101     [1]0001
```

Adder : The hardware circuit which executes this addition is called Adder. There are 2 types of Adders namely Half Adder & full Adder

The Adder circuit characteristics are detailed by a ~~circuit~~ block symbol, a truth table, a circuit & formula.

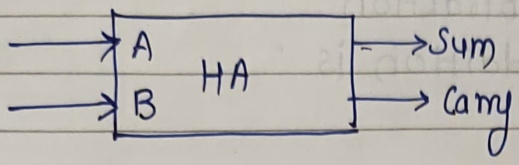
The Adder circuits are constructed from logic gates which satisfy the formula as per truth table. These are also called combinational logic.

The Half Adder (HA) has 2 inputs (A, B,) & two outputs (sum & carry).

The Sum is XOR of input while

The carry is AND of input

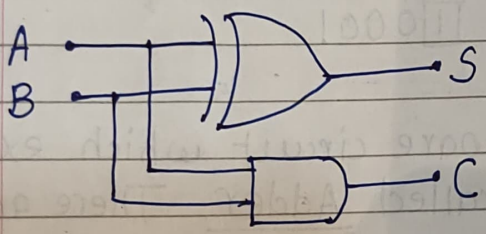
HA block symbol



HA Truth Table

Inputs		outputs.	
A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

HA Logic Circuit



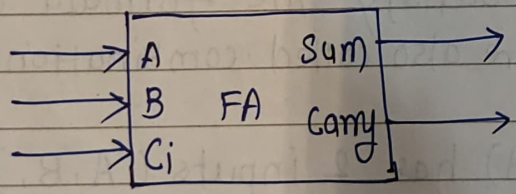
HA formula.

Sum $S = A \oplus B$

Carry $C = AB$

A full Adder (FA) Also performs 1-bit Addition but taking 3 inputs (A, B, & Ci) & produces two outputs (sum & carry)

FA block symbol

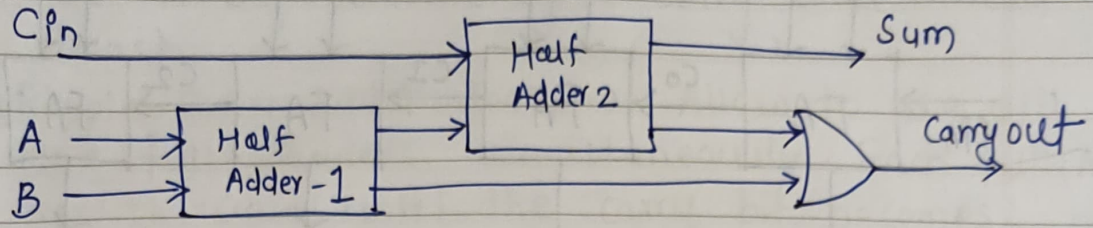


2^{inputs} $2^3 = 8$

FA Truth Table

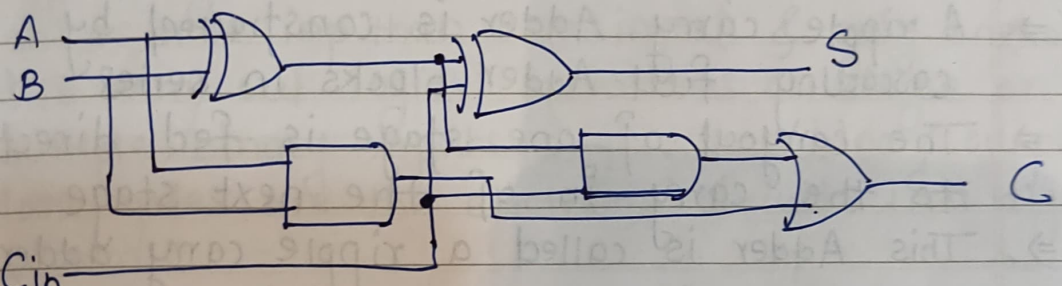
Inputs			outputs.	
A	B	Carry (Cin)	Sum	Carry out
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

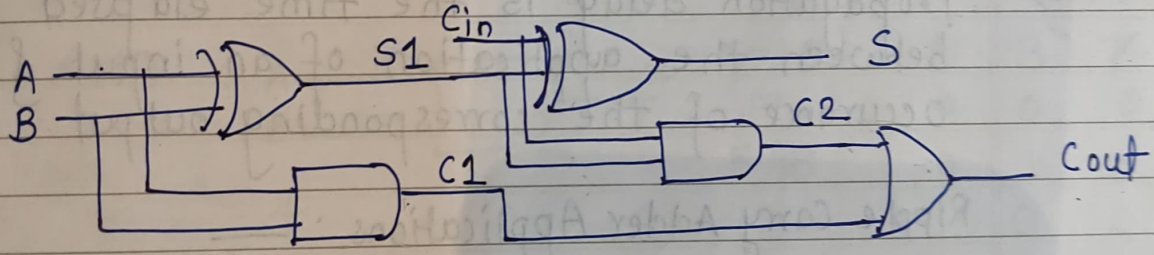
classmate
Date
Page



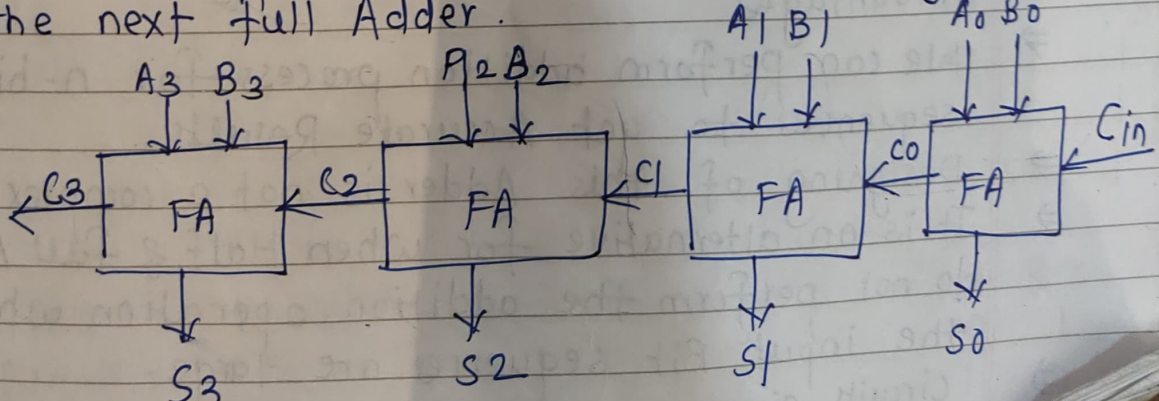
FA formula $S = A \oplus B \oplus C_i$

$C_{out} = AB + C_i(A+B)$





* A ripple carry Adder is the simple form of a parallel Adder, where the carry-out of each full Adder is connected to the carry-in of the next full Adder.

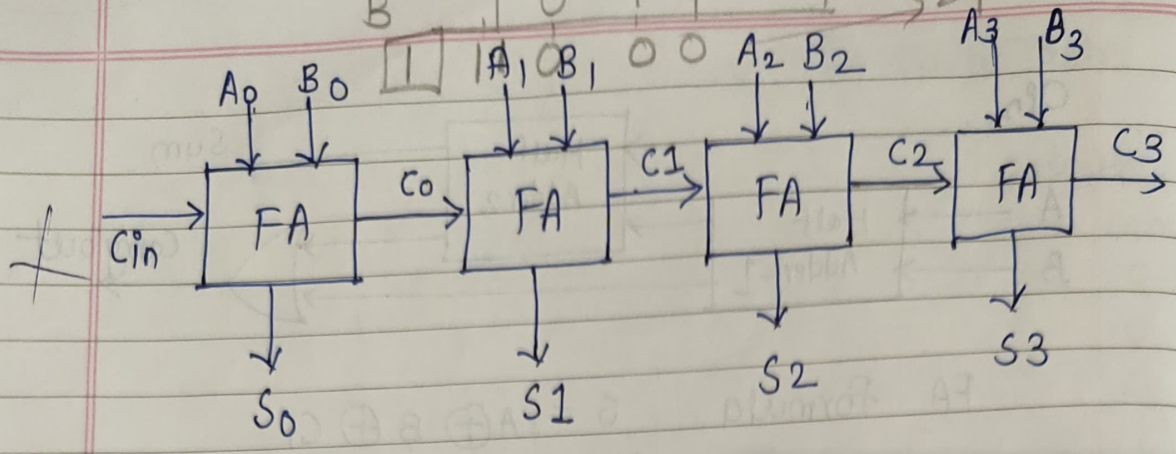


e.g. 4-bit Ripple carry Adder

A 1 1 0 1 → 13

B 1 0 1 1 → 11

1 1 0 0 0 → 24



⇒ An Adder is a digital circuit that performs addition of numbers.

⇒ A ripple carry Adder is constructed by cascading full Adder blocks in series.

⇒ The carryout of one stage is fed directly to the carry-in of the next stage.

⇒ This Adder is called a ripple carry adder because each carry bit gets rippled into the next stage.

⇒ Propagation delay is the time elapsed between the application of an input & occurance of the corresponding output.

Ripple Carry Adder Applications :—

⇒ Used mostly in addition to n-bit input sequences.

⇒ Applicable in the digital signal processing & microprocessors.

Advantages

⇒ We can perform addition process for n-bit sequences to get Accurate Results.

⇒ Designing of this Adder is not a complex process.

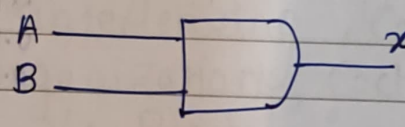
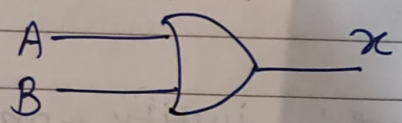
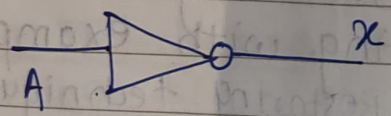
⇒ It is an alternative for when Half & full Adders do not perform the addition operation when the input Bit sequences are large.

⇒ Circuits gives o/p with Delay won't be preferable

This can be overcome by a carry look-Ahead Adder circuit.

Limitations Ripple Carry Adder does not Allow to use all the full Adders simultaneously. Each full Adder has to wait until the carry bit becomes available from its adjacent full adder. This increases the propagation time.

Logic gates are the basic building blocks of any digital system. It is an electronic circuit having one or more inp than one input & only one output Used To. carry out logical operations on single or multiple binary inputs & give one binary output. logic gates are the electronic circuits in a digital system.

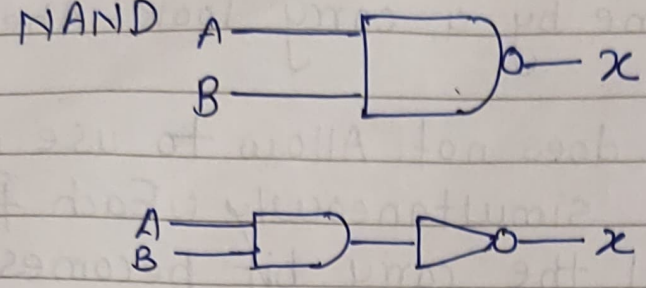
Name	Graphic Symbol	Algebric fn	Mode of op^n Truth Table (Rule)
AND		$x = A \cdot B$ or $x = AB$	A B x: 0 0 0; 0 1 0; 1 0 0; 1 1 1
OR		$x = A + B$	A B x: 0 0 0; 0 1 1; 1 0 1; 1 1 1
NOT (Inverter)			A x: 0 1; 1 0

2 inputs $= 2^2 = 4$

The NAND gate combines the logic of an AND gate & a NOT gate, it produces the inverse of the output of an AND gate

NOT AND

NAND

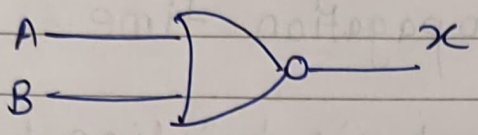


$x = (AB)'$

A	B	x
0	0	1
0	1	1
1	0	1
1	1	0

NOT OR

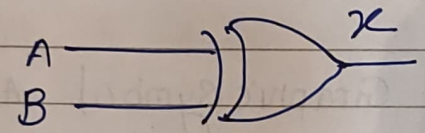
NOR



$x = (A+B)'$

A	B	x
0	0	1
0	1	0
1	0	0
1	1	0

Exclusive-OR (XOR)



$x = A \oplus B$

or

$x = A'B + AB'$

$1 \times 1 + 0 \cdot 0 = 1$

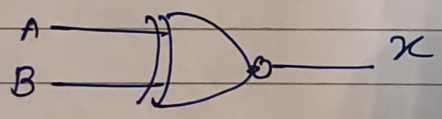
$0 \cdot 1 + 1 \cdot 1 = 1$

$0 \cdot 1 + 1 \cdot 0 = 0$

A	B	x
0	0	0
0	1	1
1	0	1
1	1	0

UNIT

NOT XOR XNOR

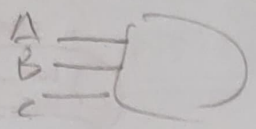


$x = \overline{A \oplus B}$

A	B	x
0	0	1
0	1	0
1	0	0
1	1	1

Analyze: Compare: justify = prof.

1) Write a short Note on Ripple Carry Adder
2) Explain Carry Look Ahead Adder.
3) Explain H/w implementation of unsigned Binary Multiplication.
4) Solve 7 x -3 using Booth Algorithm
5) Explain Division Restoring with example
6) Explain non-Restoring technique for Division with example.



$2^3 = 8$

A	B	C
0	0	0
0	0	1
0	1	0

Classmate

UNIT II

→ Full Adder, Ripple Carry Adder, Carry Look Ahead Adder
→ Multiplication - shift-Add, Booth Multiplier
→ Carry save Multiplier, H/W Implementation of Integer Multip
→ Division Restoring & Non Restoring Techniques, Floating Point Arit

Unit I: Functional Blocks of a Computer & Data Representation.

⇒ Computer Architecture & Organization, Computer components CPU, Memory, Input-Output subsystems, Control unit, Interconnection structures & Bus Interconnection Signed Number Representation, Fixed & floating point Representation, IEEE 754 format, Character Representation, Number Conversion.
Self-study : Top level view of computer system.

UNIT III Memory System Design

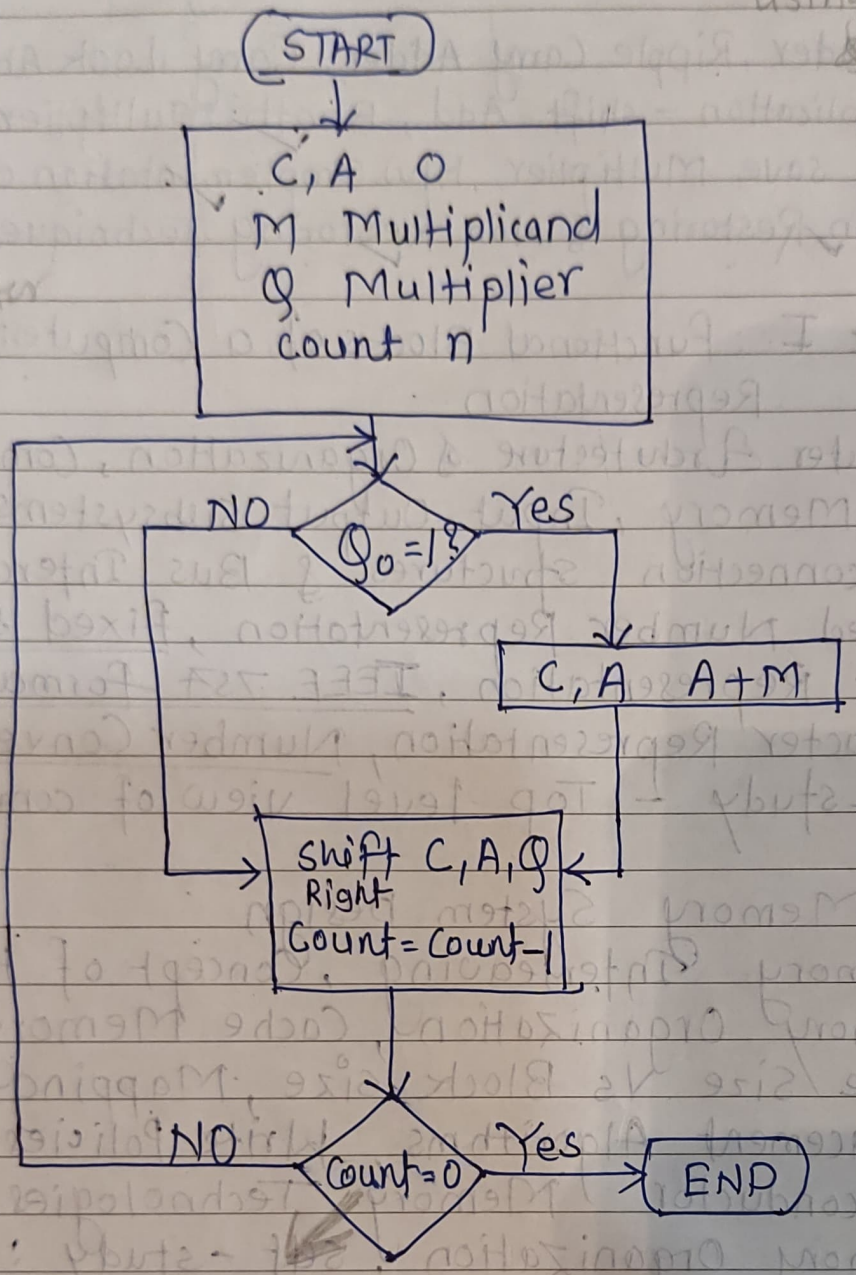
Memory Interleaving, Concept of Hierarchical Memory Organization, Cache Memory, Cache Size Vs Block size, Mapping functions, Replacement Algorithms, Write Policies, Semiconductor Memory Technologies, Memory Organization. self-study : RAID.

I

1) Ripple carry Adder, Carry look Ahead,
2) shift-Add & Booth.
3) Division Restoring & Non-Restoring
4) H/W Imple of unsigned Binary Multiplication.

III] Memory Interleaving, Memory Hierarchy.
3 Mapping fns, 3 Replacement.
Memory Types.
cache write policies

Flowchart For Unsigned Binary Multiplication



No. of Additions - required 1's in multiplier

1011 · Multiplicand (11 dec)
× 1101 Multiplier (13 dec)

```
  1011
 0000+
1011++
1011+++
```

1 0001111 product (143)

$2^7+2^3+2^2+2^1+2^0$
$= 128+8+4+2+1 = 143$

$n \times n = 2n$

n = no. of bits Q

Right shift

	C	A	Q	M	
	0	0000	1101	1011	Initial values
0000 + 1011 = 1011	0	1011	1101	1011	Add A→A+M } First cycle
	0	0101	1110	1011	shift C,A,Q }
0010 + 1011 = 1101	0	0010	1111	1011	shif C,A,Q } second cycle
	0	1101	1111	1011	Add A→A+M } third cycle
	0	0110	1111	1011	shift C,A,Q }
0110 + 1011 = 10001	1	0001	1111	1011	Add A→A+M } Fourth cycle
	0	1000	1111	1011	shift C,A,Q }

product (Result)

69
12
57

$2^6+2^5+2^4+2^3+2^2+2^1+2^0$

128 × 2 + 1 = 256

$256+16+8+4+2+1$

$128+16+8+4+2+1$

$= 128+31$

1010 (10 dec) M.
X 0010 (2 dec)

0000
1010+
0000++
0000+++
00101000
2^4 2^2 2^1 16 + 4 = 20

	C	A	Q	M	
	0	0000	0010	1010	Initial.
0000	0	0000	0001	1010	shift 1st cycle
1010					
1010	0	1010	0001	1010	Add
	0	0101	0000	1010	shift } 2nd cycle
	0	0010	1000	1010	shift } 3rd cycle
	0	0001	0100	1010	shift } 4th cycle

Carry look Ahead Adder

⇒ Disadvantage of Ripple carry Adder is overcomed by carry look Ahead Adder.

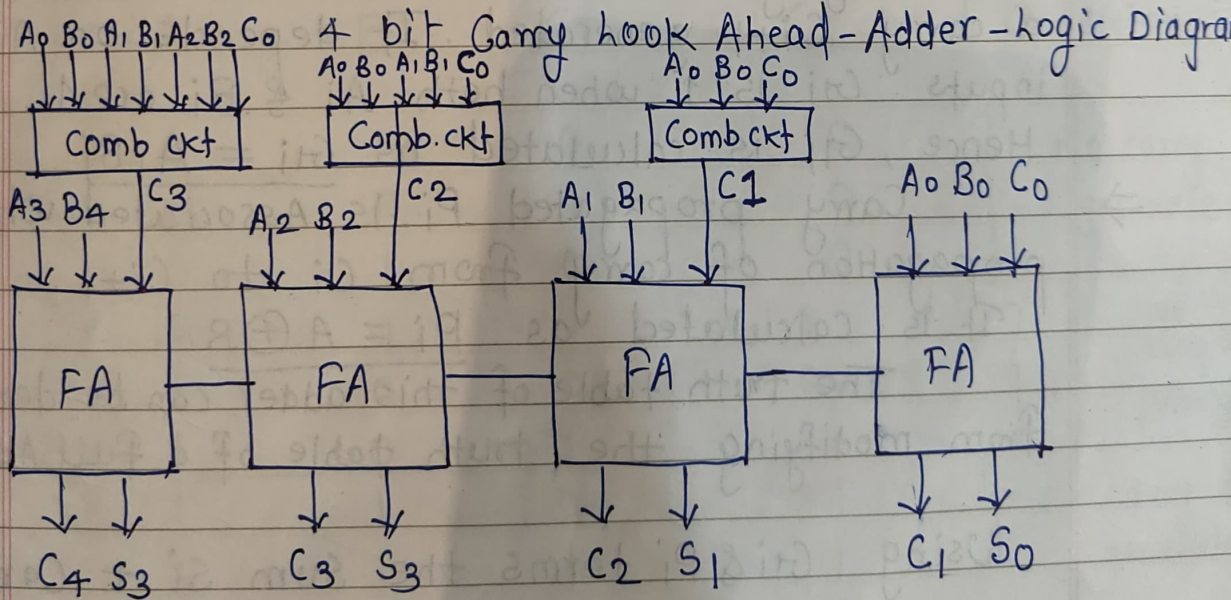
⇒ Carry propogation Delay is in R.C.A. wait time for access the carry & make the sum.

⇒ The carry look Ahead Adder calculates one or more carry bits before the sum which Reduces the wait time

⇒ CLA is the faster circuit in performing binary addition by using the concepts of carry generate & Carry Propagate

⇒ A CLA is termed as the successor of a ripple carry Adder.

⇒ A CLA circuit minimizes the propagation delay time



⇒ In this Adder, the carry input at any stage of the adder is independent, of the carry bits generated at the independent stages.

⇒ Here the output of any stage is dependant only on the bits which are added in the previous stages & the carry input provided at the beginning stage.

⇒ Hence the circuit at any stage does not have to wait for the generation of carry-bit from the previous stage & carry bit can be evaluated at any instant of time.

Truth Table of CLA Adder

⇒ For deriving the truth table of this Adder. 2 new terms are introduced. — carry generate & carry propagate

⇒ Carry generate $G_i = 1$ whenever there is a carry C_{i+1} generated. It depends on A_i & B_i inputs. G_i is 1 when both A_i & B_i are 1. Hence, G_i is calculated as $G_i = A_i B_i$

⇒ Carry propagated P_i is Associated with the propagation of carry from C_i to C_{i+1}. It is calculated as $P_i = A \oplus B$

The truth table of this adder can be derived from modifying the truth table of a full Adder.

Using G_i & P_i terms the sum S_i + Carry C_i are given as below —

$$S_i = P_i \oplus G_i$$
$$C_{i+1} = C_i P_i + G_i$$

Therefore the carry bits C1, C2, C3 & C4 can be calculated as

when i=0

~~C1 = C0P0 + G1~~ $\quad C1 = C0 \cdot P0 + G0$

$$C2 = C1P1 + G1 = (C0 \cdot P0 + G0) \cdot P1 + G1$$

$$C3 = C2P2 + G2 = (C1P1 + G1) \cdot P2 + G2$$

$$C4 = C3P3 + G3 \quad ((C1P1 + G1) \cdot P2 + G2)P3 + G3$$

$$= C0 \cdot P0 \cdot P1 \cdot P2 \cdot P3 + P3 \cdot P2 \cdot P1 \cdot G0 + P3 \cdot P2 \cdot G1 + G2 \cdot P3 + G3$$

It can be observed from the equations that carry C_{i+1} only depends on carry C0, not on the intermediate carry bits.

A	B	C_i	C_{i+1}	Condition
0	0	0	0	No carry generate
0	0	1	0	
0	1	0	0	
0	1	1	1	No carry propagate
1	0	0	0	
1	0	1	1	
1	1	0	1	carry generate
1	1	1	1	

G P

$C_0 = A \cdot B + A \oplus B \cdot C_n$

carry gen — carry propagate

$C_0 = G + PC_{in}$

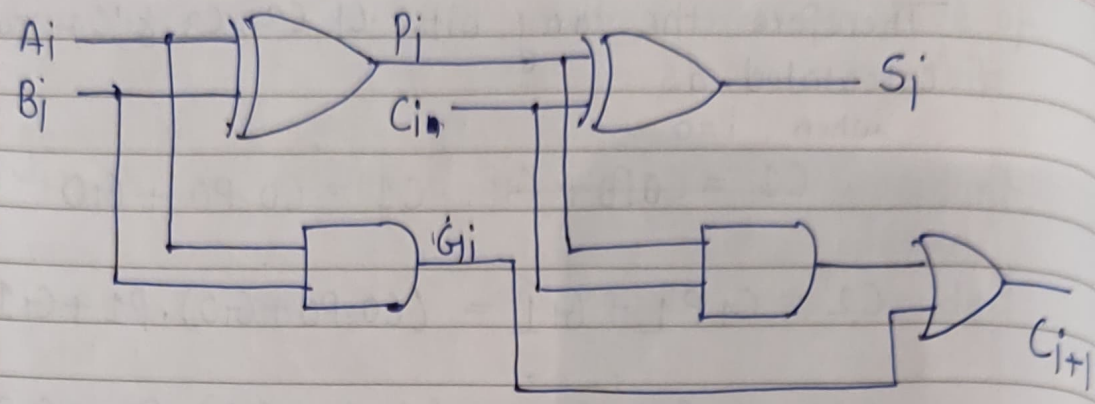
$C0P0 + G10(P1) + G1 \cdot P2 + G2 \; P3 +$

$C0P0P1 \cdot \quad (C1P1 \cdot P2 + G1 \cdot P2 + G2)P3$

$C1P1 \cdot P2P3 + P2 \cdot G1 \cdot P3 + G2 \cdot P3$

$C1P1P2P3 + G1P2P3 + G2P3$

$(C0P0 + G0)$

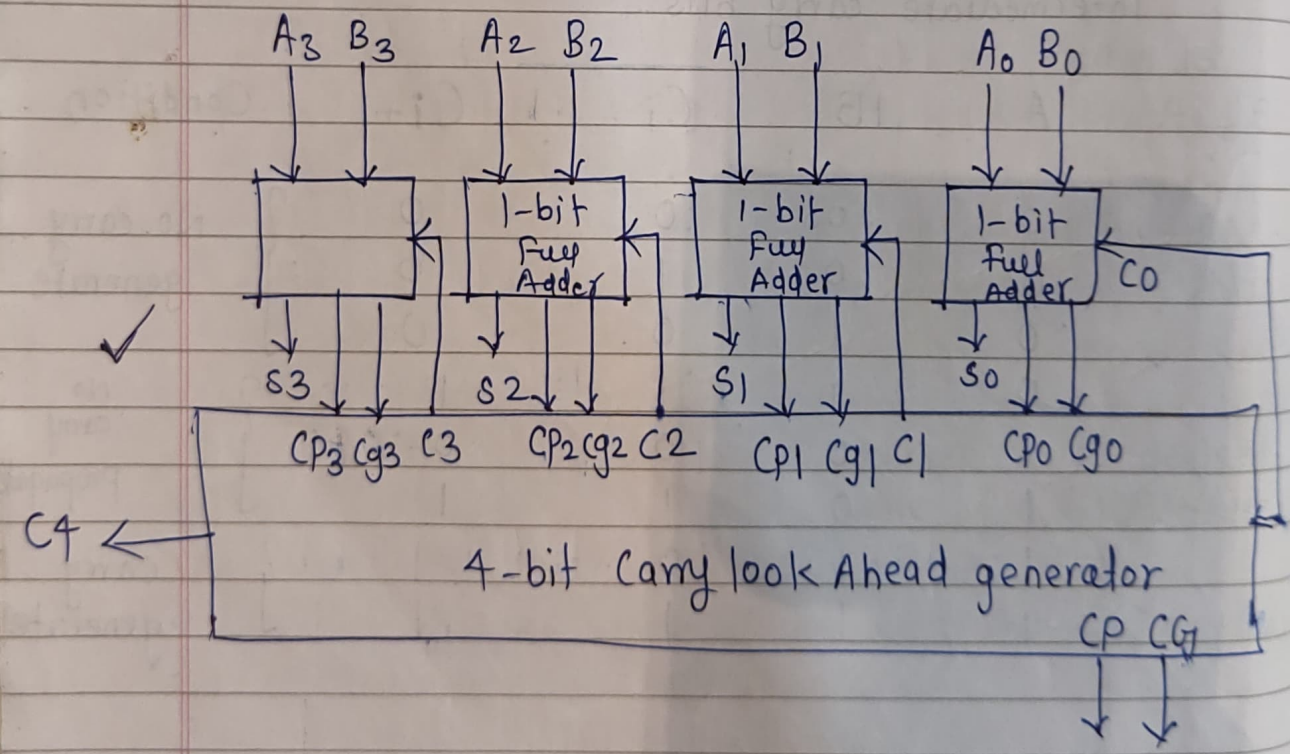


$G_i = A_i \cdot B_i$

$P_i = A_i \oplus B_i$

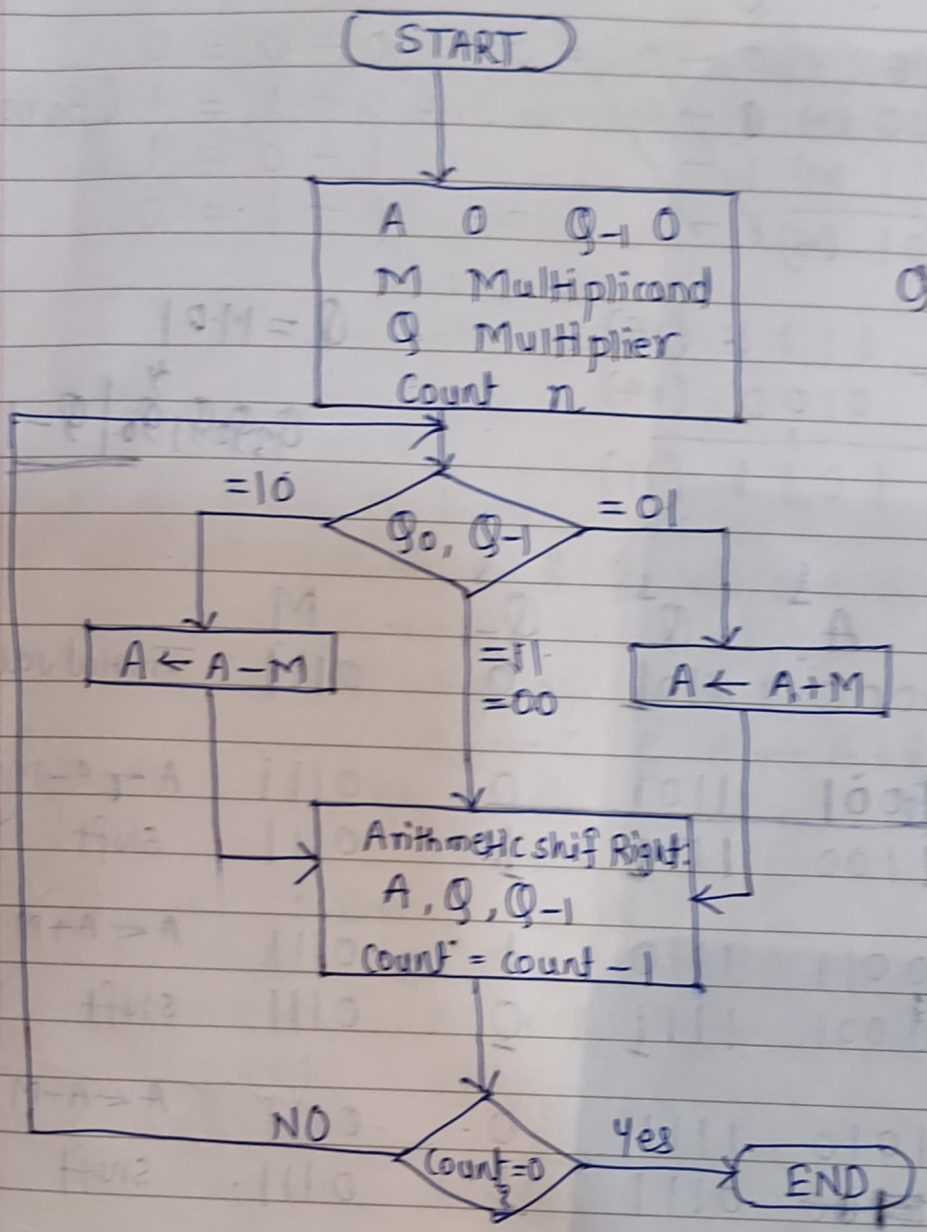
$S_i = P_i \oplus C_{in}$

$C_{i+1} = G_i + P_i \cdot C_i$



(Hlw 2 Implement)

* Booth's Algorithm for 2's Complement Multiplication



Booth's Multiplication Algorithm is a multiplication algorithm that multiplies two signed binary numbers in 2's complement notation.

What will be the value obtained after multiplication of $-2 * -3$ using Booths Algorithm? Justify your answer.

-7 / -3   7 / ×3 / -21.

Remember Arithmatic shift right

Example of Booth's Algorithm

Binary Subtraction

e.g. ~~1010~~

```
  1100 (12)
- 0111 (07)
  ----
  0101 (05)
```

0 − 0 ≠ 0 ... 0 − 0 = 0
0 − 1 = 1 (Borrow 1)
1 − 0 = 1
1 − 1 = 0

```
- 1111 (15)
  0100 (04)
  ----
  1011 (11)
```

3 = 0011, 1100 + 1 = 1101

Q = 1101

$Q_3 Q_2 Q_1 Q_0 \mid Q_{-1}$

7 = 0111, ×−3 = −21

3 = 0011, 1100, + 1, −3 = 1101

A	Q	Q_{-1}	M	
0000	1101	0	0111	Initial value
1001	1101	0	0111	A ← A − M (0000 − 0111)
1100	1110	1	0111	Shift (1001)
0011	1110	1	0111	A ← A + M (1100 + 0111)
0001	1111	0	0111	Shift (0011)
1010	1111	0	0111	A ← A − M (0001 − 0111 = 1010)
1101	0111	1	0111	Shift
1110	1011	1	0111	Shift

0010 + 1 = 0011

7 × −3 = −21

1 0010100 (1's), + 1

1 0010101

Sign $2^6\ 2^5\ 2^4\ 2^3\ 2^2\ 2^1\ 2^0$

16 + 4 + 1 = −21

16 8 4 2 1

* Restoring Division Algorithm (Diagram)

Input:
- M - positive divisor (n-bit) (3)
- Q - positive dividend (n-bit) (7)

Output
- Q - Quotient
- A - Remainder.

Begin
- A ← 0
- Do n times
  - shift A & Q left one binary position
  - A ← A − M
  - If sign of A is 1
    - q_0 ← 0 & A ← A + M (Restore A)
  - Else
    - q_0 ← 1
  - End

A	Q	
0000	0111 (7)	M = 0011 (3) Initial value
0000	1110	shift
1101		subtract ✓
0000	1110	Restore

} 1st cycle

0000 − 0011 = 1101 (7/3)

0001	1100	shift
1110		subtract
0001	1100	Restore

} 2nd

+1101 + 0011 = 10000

0011	1000	shift
0000		subtract
0000	1001	set q_0 = 1

} 3rd

$$3\overline{)7}\ 2,\ 6,\ 1$$

$2^1\ 2^0$
$0\ 0\ 1\ 0$

A	Q		
0001	0010	shift	4th cycle
1110		subtract	
Remainder 0001	0010	Restore	
	Quotient		

Non-Restoring:

$10|3$ $\quad 3\overline{)10}$ — Q, -9, M, 1

M = 3 = 0011
Q = 10 = 1010

A	Q		
0000	1010	Initial value	
0001	010_	shift	
1010		A ← A − M	0001 − 0011
1010	0100	$q_0 = 0$	1 1010
1110	101_	shift	
0001		A ← A + M	1110 + 0001
0001	1011	$q_0 = 0$	1 0001
0011	011_	shift	
		A ← A − M	0011 − 0011
			0000

23/3/22 6, 7, 18, 19, 21, ~~26~~, 27, 28, 33, 34, 36, 39, 40, ~~41~~, 43, 44, ~~48~~ Pe.
49, 51, 52, 58, 59, 60, 62, 63, 64, 69.

* Binary Non-Restoring Division Algo. for Unsigned Integer.

Input
M - positive divisor (n-bit)
Q - positive divident (n-bit)

Output
Q = Quotient
A - Remainder

Begin
- A ← 0
- Do n times
  - If the sign of A is 0
    shift A & Q left one bit position & A ← A-M
    else
    shift A & Q left one bit position & A ← A+M
  - If sign of A is 0
    - $q_0 \leftarrow 1$
    Else
    - $q_0 \leftarrow 0$
- If sign of A is 1
  - A ← A+M

End

10/3

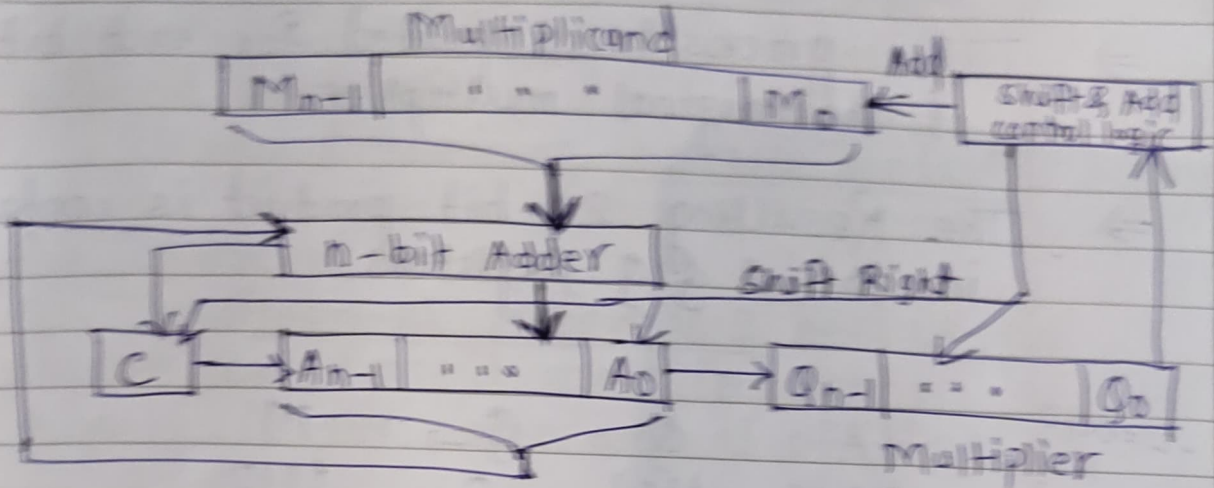
$$\begin{array}{r|l} 3 & \\ \hline 3 & 10 \\ -9 & \\ \hline 1 & \end{array}$$

M = 3 = 0011
Q = 10 = 1010

A	Q		
0000	1010	– Initial	
0001	010_	shift	
1110		A ← A − M	0001 − 0011
1110	0100	$q_0 = 0$	1 1110
1100	100_	shift	
0001		A ← A + M	1100 + 0011
1111	1001	$q_0 = 1$	1 0001
0111	001_	shift	
		A ← A − M	0011 − 001

A	Q		
0000	1010	– initial	
0001	010_	shift	
1110		A ← A − M	0001 − 0011
1110	0100	$q_0 = 0$	1 1110
1100	100_	shift	
1111		A ← A + M	1100 + 0011
1111	1000	$q_0 = 0$	1111
1111	000_	shift	
0010		A ← A + M	1111 + 0011
0010	0001	$q_0 = 1$	1 0010
0100	001_	shift	
0001		A ← A − M	0100 − 0011
0001	0011	$q_0 = 1$	0001

Hardware Implementation of Unsigned Binary Multiplication



Block Diagram

⇒ The Multiplier & multiplicand are loaded into 2 registers (Q & M)

⇒ A third Register, the A Register, is also needed & is initially set to 0

⇒ There is also a 1 bit C register, initialized to 0 which holds a potential carry bit resulting from addition

The operation of the Multiplier is as follows

⇒ Control logic Reads the bits of the multiplier one at a time.

⇒ If Q_0 is 1, then Multiplicand is Added to the A register & the Result is Stored in the A register, with the C bit used for overflow.

⇒ Then all of the bits of the C, A, & Q registers are shifted to the right one bit

⇒ If Q_0 is 0, then no addition is performed just the shift.

⇒ This process is Repeated for each bit of the original multiplier.

⇒ The Resulting 2n-bit product is contained in the A & Q Registers.

A	Q	Q_{-1}	M	
0000	1101	0	1110	
0010	1101	0	1110	A ← A − M
0001	0110	1	1110	shift
1111	0110	1	1110	A ← A + M
1111	1011	0	1110	shift
0001	1011	0	1110	A ← A − M
0000	1101	1	1110	shift
0000	0110	1	1110	shift

-2 = 0010 → 1101 + 1 = 1110

-3 = 0011 → 1100 + 1 = 1101

-2 × -3

1110 = −2 M = Multiplicand

1101 = −3 Q = Multiplier

0000 0110 → 6