5th October, 2023

UNIT-3 Adders, Multiplication, Division

Architecture

ALU Memory

base component: adder

* we cannot use 1bit FA to make 32 bit/64 bit FA, delay would increase.

⇓

we use Prefix adder

Prefix adder: purpose: to reduce time taken by FA

4 bit ⇒ i → [0 - 3]

8 bit ⇒ i → [0 - 7]

a_i, b_i, sum_i ⇒ i → [0 - (n-1)]

$c_3\ c_2\ c_1\ c_0$

$a_3\ a_2\ a_1\ a_0$

$b_3\ b_2\ b_1\ b_0$

$s_3\ s_2\ s_1\ s_0$

Carry - Lookahead Adder

N-bit

a_i	b_i	c_i	s_i	c_{i+1}	
0	0	0	0	0	① if $a_i = b_i = 0$, $c_{i+1} = 0$ / grad — kill (k_i)
0	0	1	1	0	
0	1	0	1	0	③ if $a_i \neq b_i$, $c_{i+1} = c_i$ — propagate (p_i)
0	1	1	0	1	
1	0	0	1	0	
1	0	1	0	1	
1	1	0	0	1	② if $a_i = b_i = 1$, $c_{i+1} = 1$ / Vdd — generate (g_i)
1	1	1	1	1	

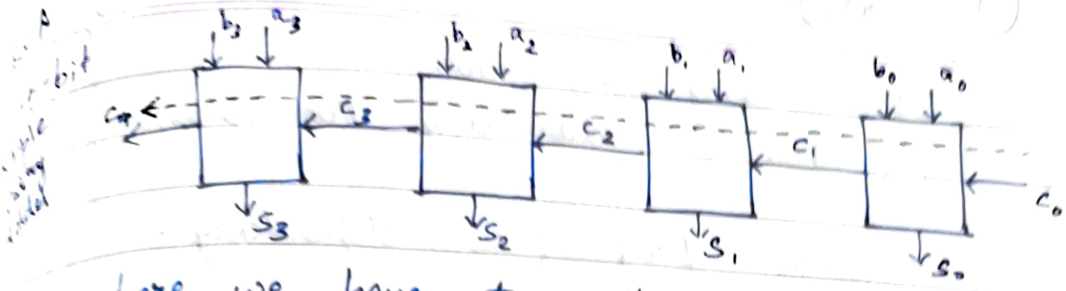
* critical path = longest path

PG adder

kill signal $k_i = \overline{a_i}\ \overline{b_i}$

propagate signal $p_i = a_i \text{ xor } b_i$

generate signal $g_i = a_i . b_i$



* here we have to wait for carry from previous stage

critical path : <---

$p_i = a_i \oplus b_i$

$g_i = a_i . b_i$

$k_i = \bar{a}_i \bar{b}_i$

$S_i = a_i \oplus b_i \oplus c_i$ (from FA)

$\boxed{S_i = p_i \oplus c_i}$ — (1)

$c_{i+1} = \bar{a}bc + a\bar{b}c + ab\bar{c} + abc$ (from TT)

$c_{i+1} = a_i b_i + b_i c_i + c_i a_i$ (from k-map)

$= c(\bar{a}b + a\bar{b}) + ab(\bar{c} + c)$

$= c(a \oplus b) + ab$

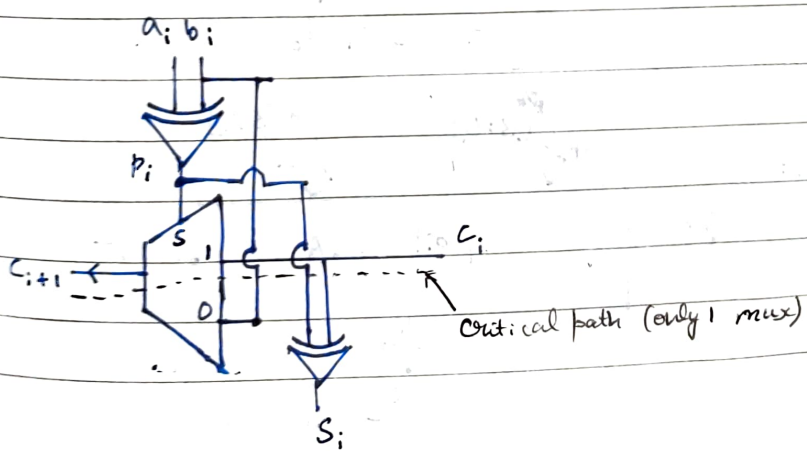
$\boxed{c_{i+1} = c_i p_i + g_i}$ — (2)

Ripple ⇒ propagation, carry is propagate in every state.

if $p_i = 1$; $c_{i+1} = c_i$

if $p_i = 0$; $c_{i+1} = a_i$ (or) $c_{i+1} = b_i$

CLA



* CLA: Carry Look Ahead
here no dependency from previous state

→ Mathematical proof of no dependency:

$$c_{i+1} = g_i + p_i c_i$$

i = 0 $\quad c_1 = g_0 + p_0 c_0$

i = 1 $\quad c_2 = g_1 + p_1 c_1$

$c_2 = g_1 + p_1(g_0 + p_0 c_0)$

$c_2 = g_1 + p_1 g_0 + p_1 p_0 c_0$ } only initial carry is enough, c_1 is eliminated

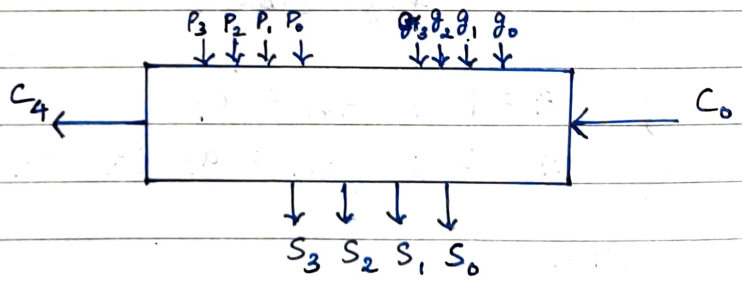
i = 2 $\quad c_3 = g_2 + p_2 c_2$

$= g_2 + p_2 (g_1 + p_1 g_0 + p_1 p_0 c_0)$

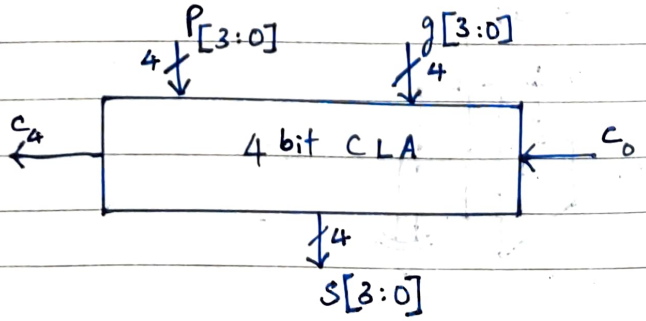
$= g_2 + p_2 g_1 + p_1 p_2 g_0 + p_0 p_1 p_2 c_0$

∴ we only need a_i, b_i and c_0
only p and g (no k) ∴ pg adder.

* PG adder = CLA adder

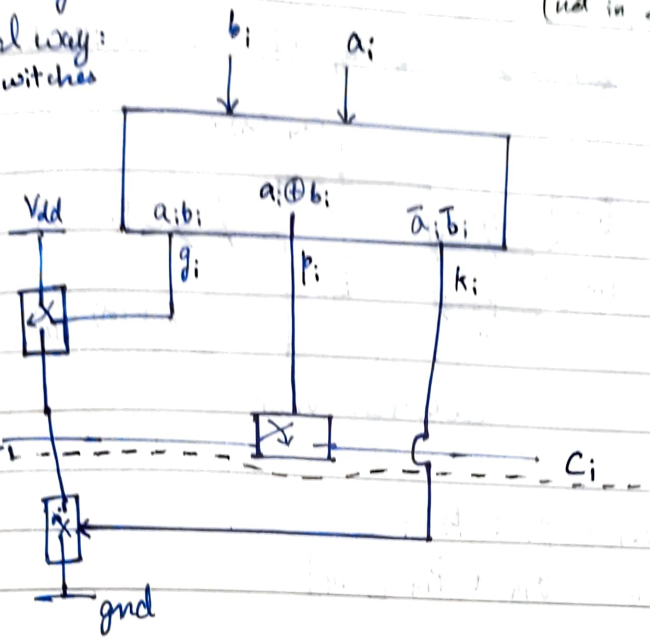


TxBk notation:



another way:
Practical way: using switches

(not in syllabus)



9th October, 2023

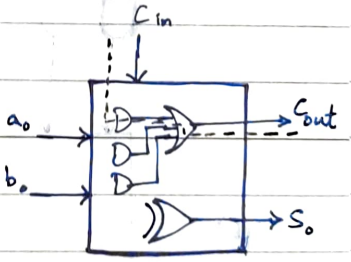
→ Types of CLAs

1. Ripple - carry adders (slow) $N < 8$
2. Carry - lookahead adders (fast) $N > 8$
3. Prefix adders (faster)

RCA $N = 8$ - bit

$$t_p = N t_{AND-OR}$$

(max delay)



1-bit FA

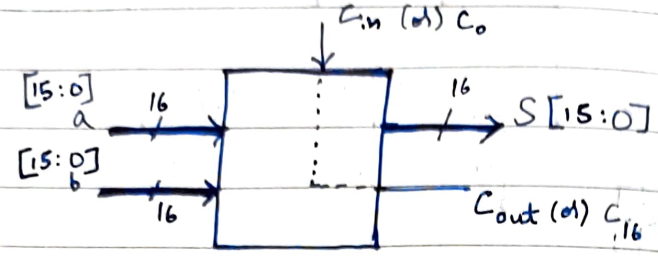
CLA

$$p_i = a_i \oplus b_i$$
$$g_i = a_i b_i$$

$$s_i = p_i \oplus c_i$$
$$c_{out} = g_i + p_i c_i$$
$$c_{i+1}$$

N = 16

for any value of N C_{in} will be 1-bit



external view

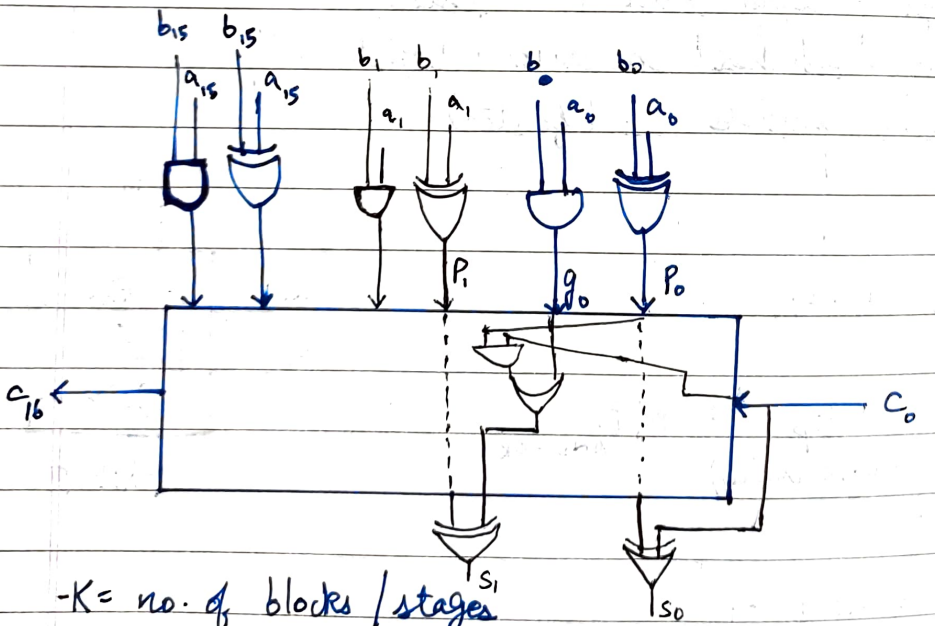
$$C_{i+1} = g_i + p_i c_i$$

$$c_{15} = g_{14} + p_{14}[c_{14}]$$

$$C_{15} = G_{[14:0]} + P_{[14:0]} C_0$$

group generate signal — group propogate signal

$$C_{16} = G_{[15:0]} + P_{[15:0]} C_0$$

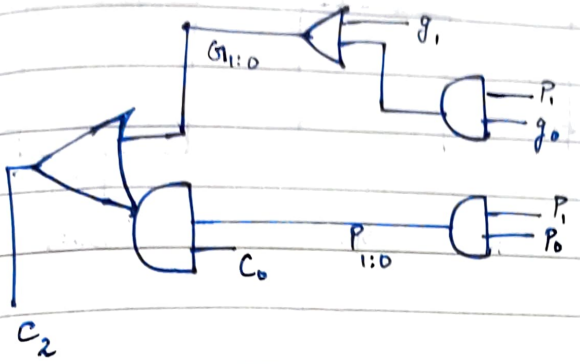


-K = no. of blocks / stages

(not + x bk) convention

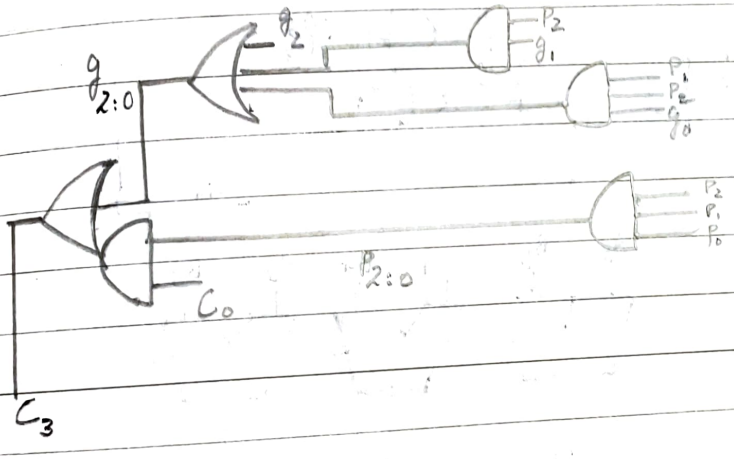
architecture for C_2

$$C_2 = \underbrace{g_1 + p_1 g_0}_{g_{1:0}} + \underbrace{p_1 p_0}_{P_{1:0}} c_0$$



for C_3

$$C_3 = \underbrace{g_2 + p_2 g_1 + p_1 p_2 g_0}_{g_{2:0}} + \underbrace{p_0 p_1 p_2}_{P_{2:0}} c_0$$

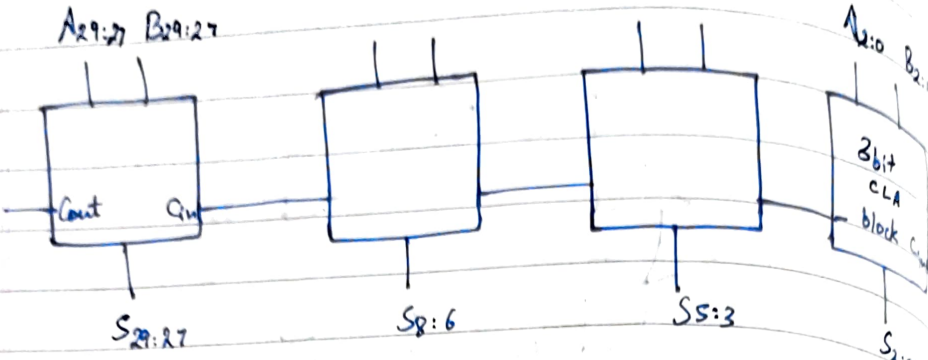


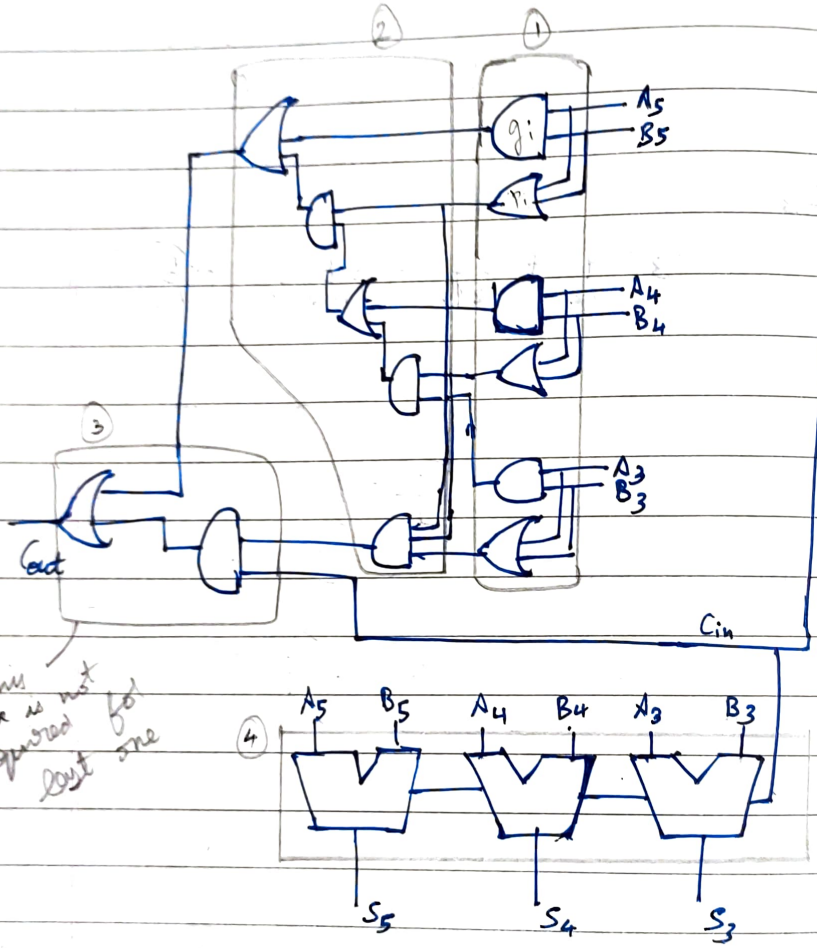
CLA delay: $t_{CLA} = t_{pg} + t_{pg\_block} + \left(\frac{N}{k} - 1\right) t_{AND\_OR} + k t_{FA}$

txbk notation k = width of each stage

$\left(\frac{N}{k}\right)$ = no. of stages

→ Analyzing CLA $N = 30$, $k = 3$





$$t_{p_{CLA}} = \underbrace{t_{pg}}_{\max(t_{AND}, t_{OR})} + \underbrace{t_{pg\_block}}_{(k-1)t_{AND} + (k-1)t_{OR}} + \left(\frac{N}{k} - 1\right) t_{AND\_OR} + \underbrace{k\,t_{p_{sum}}}_{FA}$$

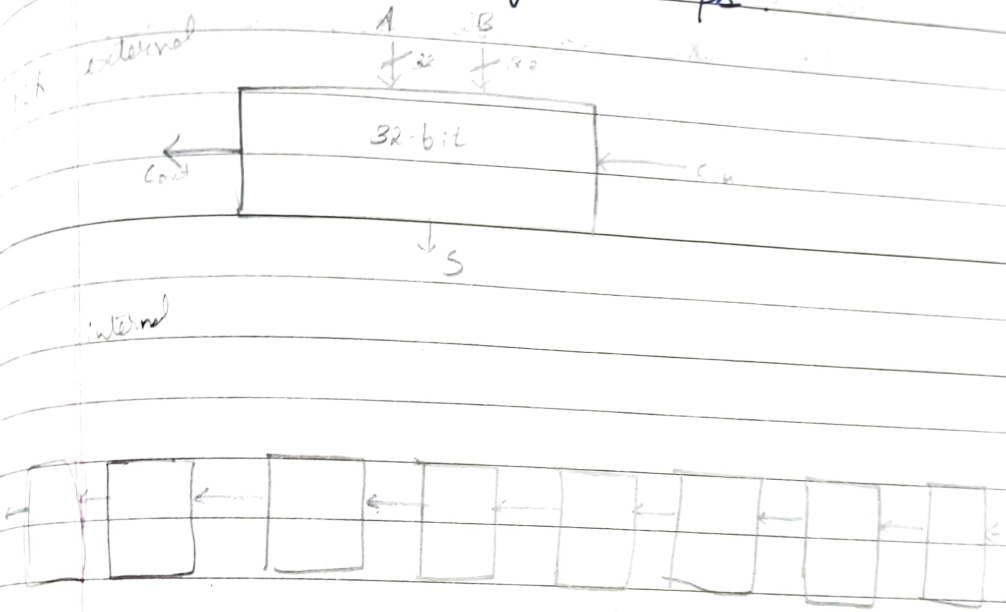
① Bitwise P/g

② 4 group P/g

③

④ sum logic

→ Compare the delay of 32-bit ripple-carry, carry-lookahead adders. The carry-lookahead adder has 4-bit blocks. Assume that each two-input gate delay is 100 ps and the full adder's delay is 300 ps.



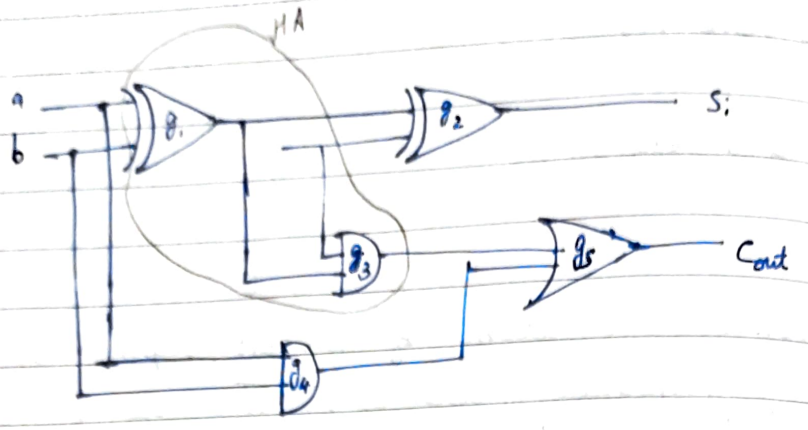
$$t_{ripple} = N t_{FA} = 32(300ps) = 9.6\ ns$$

$$t_{CLA} = t_{pg} + t_{pg\text{-}block} + (H/k - 1)t_{AND\_OR} + k t_{FA}$$

$$= 100 + 600 + 7(200) + 4(300)$$

$$= 3.3\ ns$$

F.A



q. Find out gate delay for 4-bit RCA, CLA. Take each gate delay is 10ns.

t_{RCA} =

? >= 40 ns

$$t_{CLA} = t_{pg} + t_{pg\_block} + \left(\frac{N}{k} - 1\right) t_{AND\_OR} + k t_{FA}$$

Prefix Adder Delay

delay of an N-bit prefix adder

$$t_{PA} = \underset{\text{first block}}{t_{pg}} + \log_2 N \left(t_{pg\_prefix}\right) + t_{XOR}$$

10th October, 2023

N = 8 * depends on prefix component

* replacing c with g with proper subscript

* 3 steps ⟶

i=3	i=2	i=1	i=0
			c_{in}
a_3	a_2	a_1	a_0
b_3	b_2	b_1	b_0

$$c_{i+1} = g_i + p_i c_i$$
$$c_i = g_{i-1} + p_{i-1} c_{i-1}$$
$$c_{i-1} = g_{i-2} + p_{i-2} c_{i-2}$$

PA generate and propagate signals :

$$G_{i:j} = G_{i:k} + P_{i:k} G_{k-1:j}$$
$$P_{i:j} = P_{i:j} P_{k-1:j}$$

spanning bit | spanning bit

i (max) | k+1 k | k-1 | j (min)

L | R

$$\textcircled{*}\ G_{i:j} = G_L + P_L G_R$$
$$= G_{i:k} + P_{i:k} G_{k-1:j}$$

eg: given

A 1 0 1 0

B 0 1 1 1

C_{in} 1

N = 4

(a) col.

$G_i = a_i b_i$

$P_i = a_i$ xor b_i

R1	col no.	3	2	1	0	-1 (C_{in})
R2	G_i	0	0	1	0	1
R3	P_i	1	1	0	1	X

Algorithm :

Step 1: Fill C_{in} → -1^{th} col.

Step 2: Compute G_i & P_i ⇒ Bit wise
(R2) (R3)

Step 3: Group G & P (grouping of 2 cols from R to L)
(" 4 " ")
⋮

Step 4: S_i expression

grouping of 2 col.

(a) $G_{0:-1} = G_L + P_L G_R$

$= G_0 + P_0 G_{-1} \Rightarrow 0 + 1(1) = 1$

$C_1 \rightarrow \boxed{G_{0:-1} = 1}$

(b) $G_{2:1} = G_L + P_L G_R$

$= G_2 + P_2 G_1$

$= 0 + 1(1)$

$\boxed{G_{2:1} = 1}$

$P_{0:-1} = P_0 P_{-1}$ ← we don't have this

$= X$

(c) $P_{2:1} = P_2 P_1$

$= 1 \cdot 0 = 0$

(d) $P_{0:-1}$ = Not used

$G_{2:-1} = G_L + P_L G_R$

$= G_{2:1} + P_{2:1} G_{0:-1}$

$= 1 + 0(1)$

$\boxed{G_{2:-1} = 1}$

RCA procedure

	C_3	C_2	C_1		
	(1)	(1)	(1)	(1)	← C_{in}
A	1	0	1	0	
B	0	1	1	1	
	1	0	0	1	0

white cell - implementing bit-cell
black cell - implementing $G_{i:i}$ & $P_{i:i}$

$G_{1:-1} = G_L + P_L G_R$
$= G_1 + P_1 G_{0:-1}$
$= 1\phi + 0(1) = 1$

needed for S2 (from big) — prev carry (C_1)

* replace c with g values, keeps data ready without c

* Prefix Adder Schematic

$G_{14:-1}$

L: $G_{14:7}$ R: $G_{6:-1}$

eg: A → 0 1 1 0 0 1
B → 1 0 1 0 1 0
C_{in} → 0

~~$G_{8:-1}$~~

$G_{2:1} = G_2 + P_2 G_1$
$= 0 + 0(0) = 0$

~~$G_{7:-1} = G_L + P_L G_R$~~
~~$= G_{8:7} + P_{8:7} G_{6:-1}$~~

$P_{2:1} = P_2 P_1 = 0$

	5	4	3	2	1	0	-1
A	0	1	1	0	0	1	0
B	1	0	1	0	1	0	
G_i	0	0	1	0	0	0	0
P_i	1	1	0	0	1	1	X

$G_{4:3} = 1$ $G_{2:1} = 0$ $G_{0:-1} = 0 + 1(0) = 0$

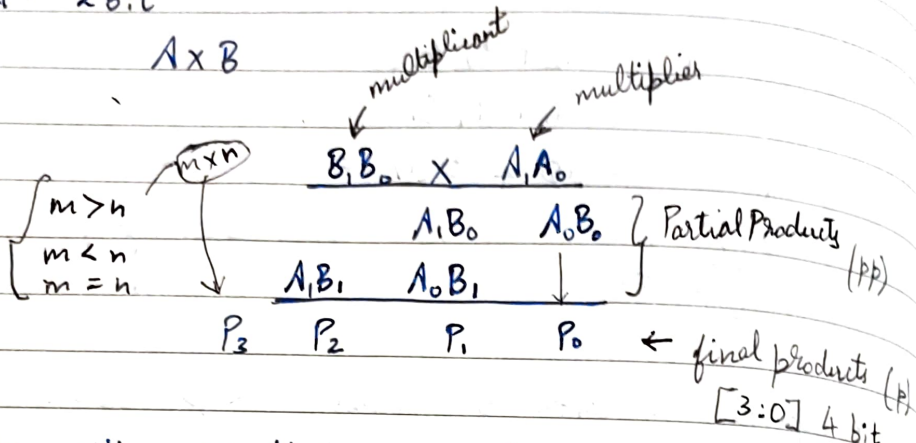
$G_{2:-1} = G_{2:1} + P_{2:1} G_{0:-1} = 0 + 0(0) = 0$

12th October, 2023 Binary Multiplication

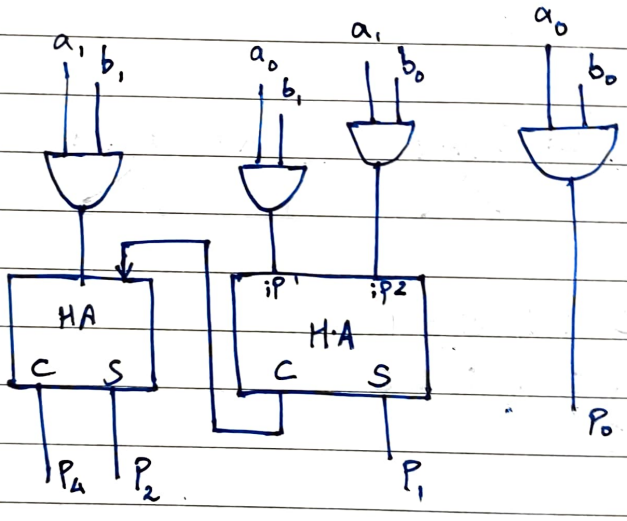
Array Multiplier ①

→ N = 2 bit

A x B



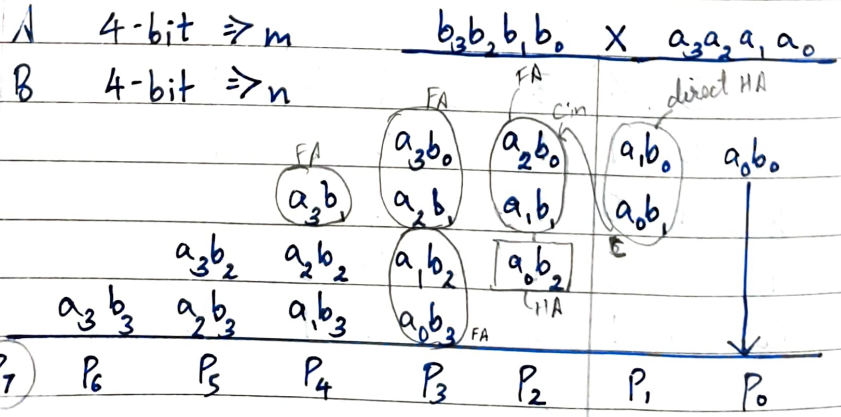
* We will use H.A as there is no carry-in (we take direct o/p of AND)

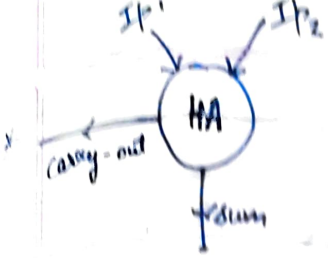


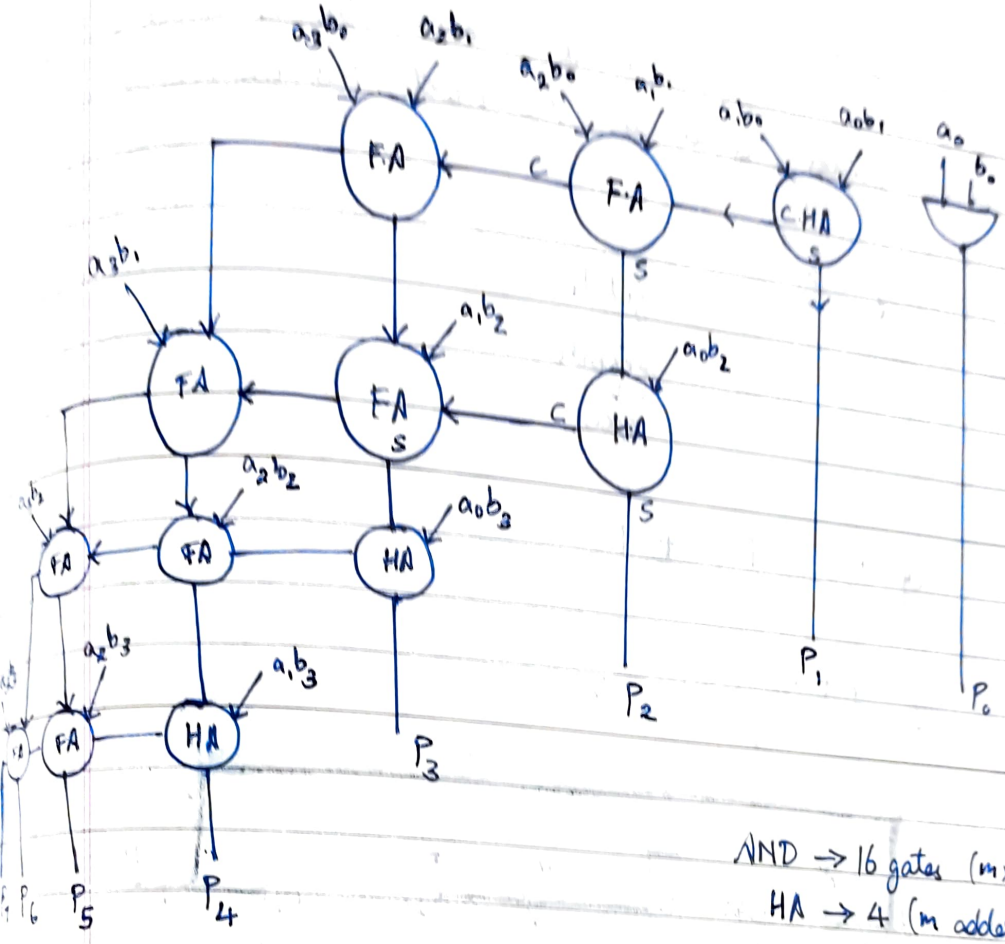
→ N = 4 bit

A 4-bit ⇒ m

B 4-bit ⇒ n



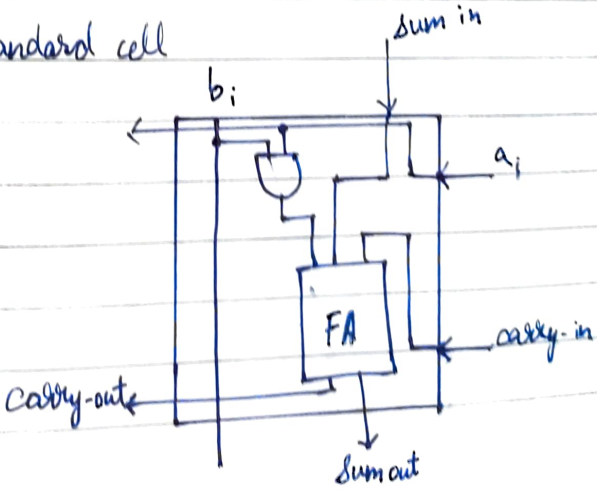




AND → 16 gates (m x n)
HA → 4 (m adders)
FA → 2m

* N+1 = FA
  1 = HA

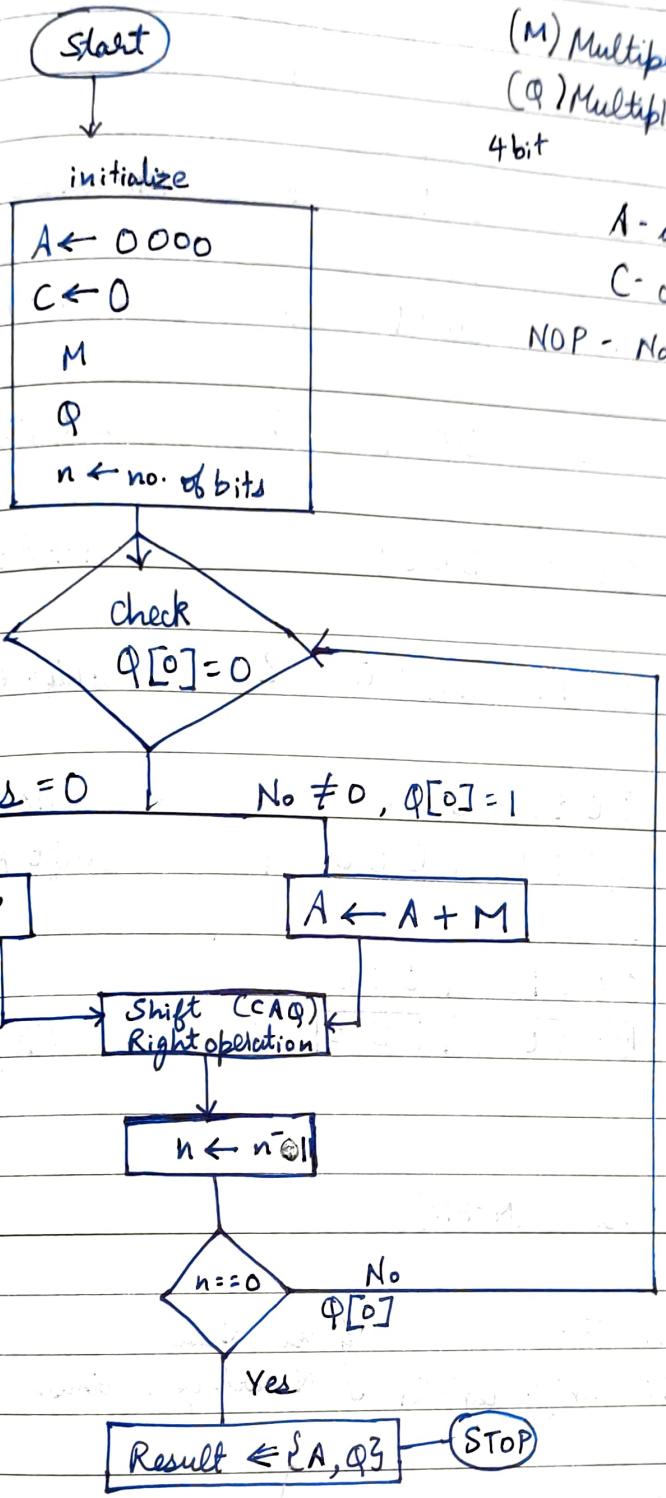
* In FA if we make $C_{in} = 0$ it becomes HA



5th October, 2023

Add - Shift method (2)

Algorithm:



(M) Multiplicand

(Q) Multiplier

4 bit $Q_3 Q_2 Q_1 Q_0$

A - accumulator

C - carry

NOP - No Operation

add A, M carry generated is C

eg: $M = 0100\ (4)_{10}$ $Q = 0011\ (3)_{10}$

	C	A	result Q ($Q_3Q_2Q_1Q_0$)	n	
					⇒ add M
i=0	0	0000	0011	4	initialize
		+0100			
1st iteration	0	0100	0011	4	A ← A+M
	0	0010	0001	3 ($n \leftarrow n-1$)	shift Right (CAQ)
		+0100			
2nd iteration	0	0110	0001		A ← A+M
	0	0011	0000	2	shift Right
3rd iteration	0	0011	0000		NOP
		0001	1000	1	shift Right
4th iteration	0	0001	1000		
		0000	1100	0	Stop
			$(12)_{10}$		

Resources (N-bit)

1. N-bit FA
2. N-bit shift registes

} use these N clock cycle times. (N iterations)

advantage: no use of multiple resources

8 4 2 1
1 1 0 1
1 0 1 1

q. M = 13 (1101) Q = 11 (1011)

C	A	Q	n	
0	0000	1011	4	
	+1101			
0	1100	1011	4	A ← A+M
	0110	1101	3	shift
	+1101		(non-)	
1	0011	1101	3	
	1001	1110	2	
0	1001	1110		
	0100	1111	1	
	+1101			
1	0001	1111		
	1000	1111	0	

q. M = 5 (101) Q = 6 (110)

C	A	Q	n	
0	000	110	3	
0	000	110		NOP
	000	011	2	shift
	+101			
0	101	011	2	
0	010	101	1	
	+101			
0	111	101	1	
	011	110	0	

11010011

101 0 0

Array Multiplier method (doesn't work with signed nos.)

6 x 2

6 → 0 1 1 0

2 → 0 0 1 0

0 0 0 0

0 1 1 0

0 0 0 0

0 0 0 0

(0 0 0 1 1 0 0)$_{12}$

array multiplier

-6 x -2

1 0 1 0

1 1 1 0

0 0 0 0

1 0 1 0 x

1 0 1 0 x x

1 0 1 0 x x x

1 0 0 1 1 0 0

replicating MSB

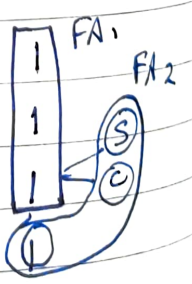
1 1 1 0 0 0 0

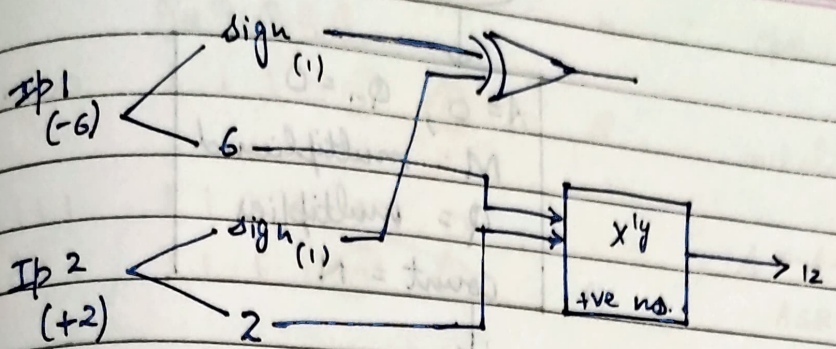
1 1 1 0 1 0

1 1 0 1 0

1 0 1 0

1 0 1 1 0 0





XOR

0 0	0	+ve	+M
0 1	1	-ve	-M
1 0	1		-M
1 1	0	+ve	+M

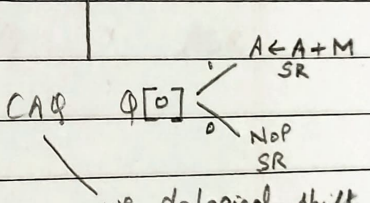
17th October, 2023

Booth's Multiplier / Algorithm : ③

① array
② add-shift (C A Q)
③ Booth's

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

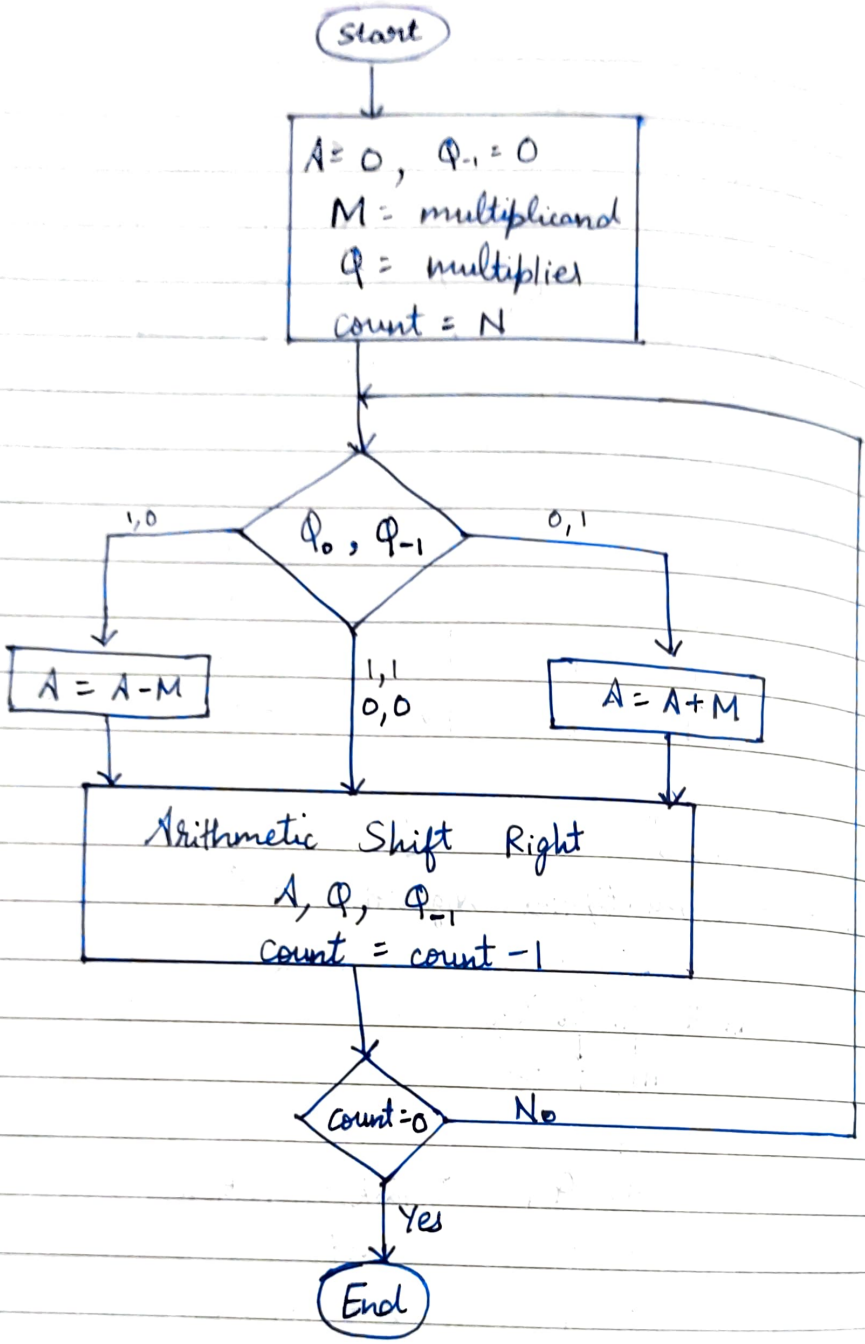
i+1 i i-1



Q_0	Q_{-1}	Task/Operation
0	0	NoP
0	1	A ← A+M
1	0	A ← A-M
1	1	NoP

* Arithmetic Shift Right (retains sign)

Algorithm



eg. 9 x (-3) using n = 5 bit

M Q

M = 01001

-M = 10111

Q = 11101

if 10 : A ← A-M

A: 00000

(-M): 10111

\+ 10111

if 01: A ← A+M

11011

\+ 01001

00100

ignore

A	$Q_4 Q_3 Q_2 Q_1 Q_0$	Q_{-1}	n	Operation
00000	11101	0	5	Initialize
10111	11101	0		A ← A − M
11011	11110	1	4	ASR
01100	11110			A ← A + M
00110	01111	0	3	ASR
11001	01111			A ← A − M
11100	10111	1	2	ASR
11110	10111			NOP
11111	01011	1	1	ASR
11110	01011			NOP
11111	00101	1	0	ASR

sign bit — 1 1 1 1 1 | 0 0 1 0 1

7 6 5 4 3 2 1 0

$-(1\times2^0)+(1\times2^3)+(1\times2^4)+(1\times2^5)+(1\times2^6)+(1\times2^7)+(1\times2^8)$

$= -27$

2's comp.

00000 11010

+1

00000 11011 $(27)_{10}$

eg2. -5×4 $n = 4$

1,0 ⇒ A ← A − M 0,1 ⇒ A ← A + M

A 0000 − M 0101 = 0101

0010 + 1011 = 1101

M = 1011 (−5)

Q = 0100 (4)

−M = 0101 (+5)

A	Q	Q_{-1}	n	operation
0000	0100	0	4	initialize
0000	0100	0		NOP
0000	0010	0	3	ASR
0000	0010	0		NOP
0000	0001	0	2	ASR
0101	0001	0		A ← A − M
0010	1000	1	1	ASR
1101	1000	1		A ← A + M
1110	1100	0	0	ASR

eg3. -6×-5 $n = 5$ bit

(M) (Q)

M: 11011 (−5)

M: 11010 (−6)

−M: 00110 (6)

−Q: 00101 (5)

11001 + 1 = 11010 (−6)

11010 + 1 = 11011 (−5)

	16	8	4	2	1
(6)	0	0	1	1	0
(5)	0	0	1	0	1

```
  1 1 1 0 1          0 0 0 0 1
+ 0 0 1 1 0        + 1 1 0 1 0
-----------        -----------
  0 0 0 1 1          1 1 0 1 1
```

Division

A	Q	Q_{-1}	n	Operation
00000	11010	0	5	initialize
00110	11011	0		A ← A-M
00011	01101	1	4	ASR
00011	01101	1		NOP
00001	10110	1	3	ASR
11011	10110	1		A ← A+M
11101	11011	0	2	ASR
00011	11011	0		A ← A-M
00001	11101	1	1	ASR
00001	11101	1		NOP
00000	11110	1	0	ASR

16 8 4 2 1

Division

(13) Divisor M = 1101

(274) Dividend Q = 100010010

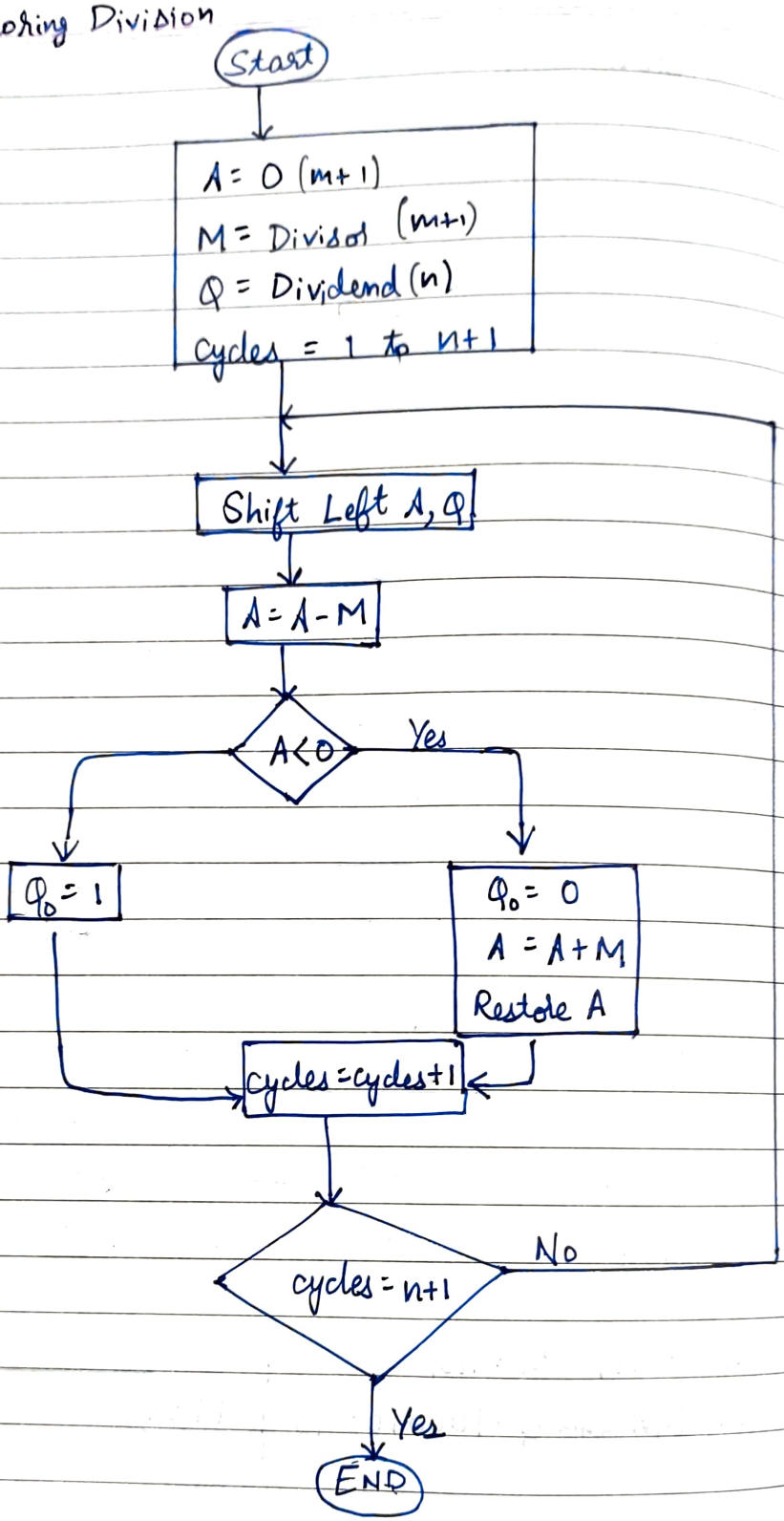
```
1101)
```

19th October, 2023

Method 1 Restoring Division

Algorithm

Txbk :

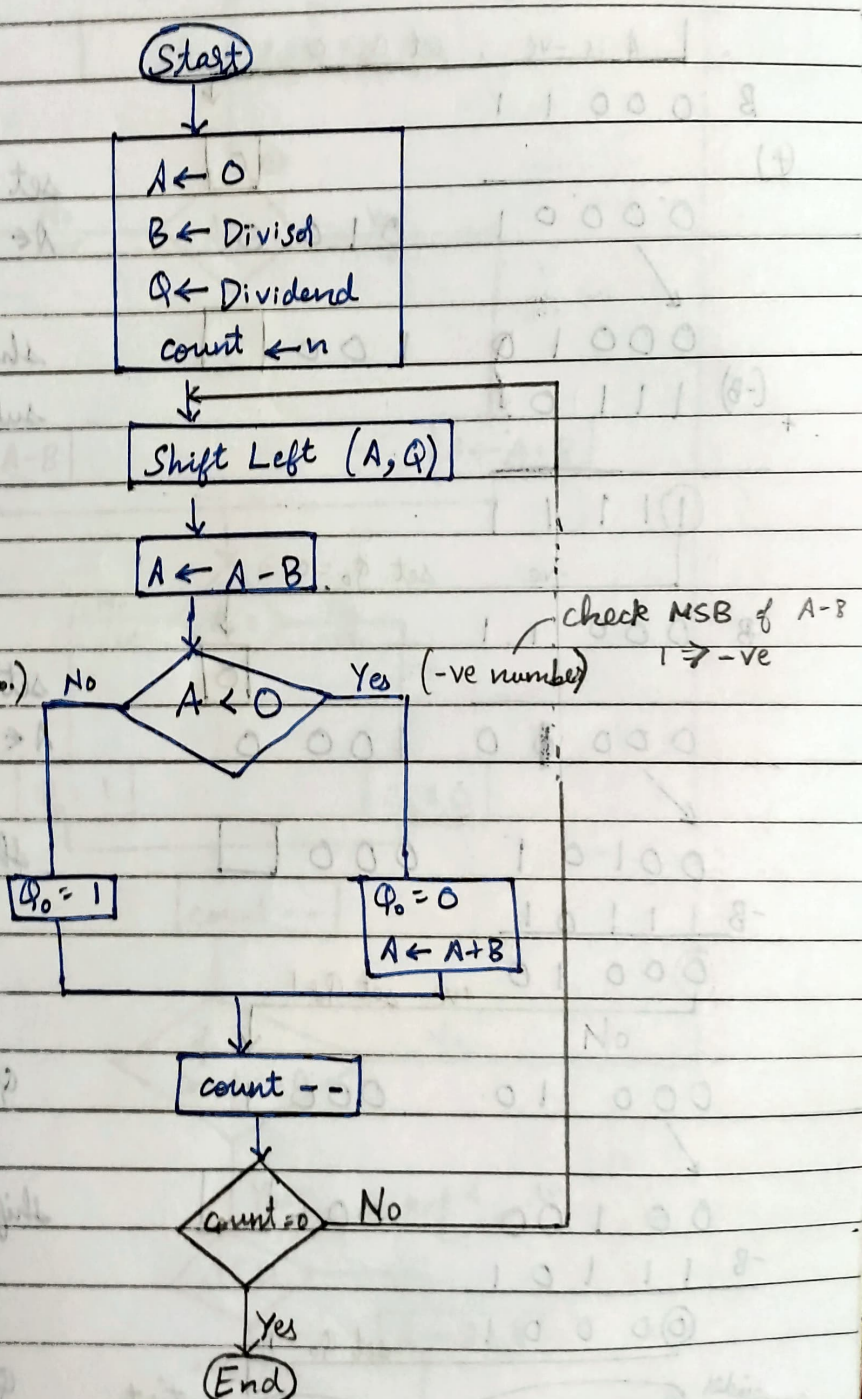


eg: Dividend = 1010 → $(10)_d$ ← Q
Divisor = 0011 → $(3)_d$ ← B
n = 4 bit

Change representation of B from n to n+1
Consider -B

B => 00011
-B => 11101

use this:



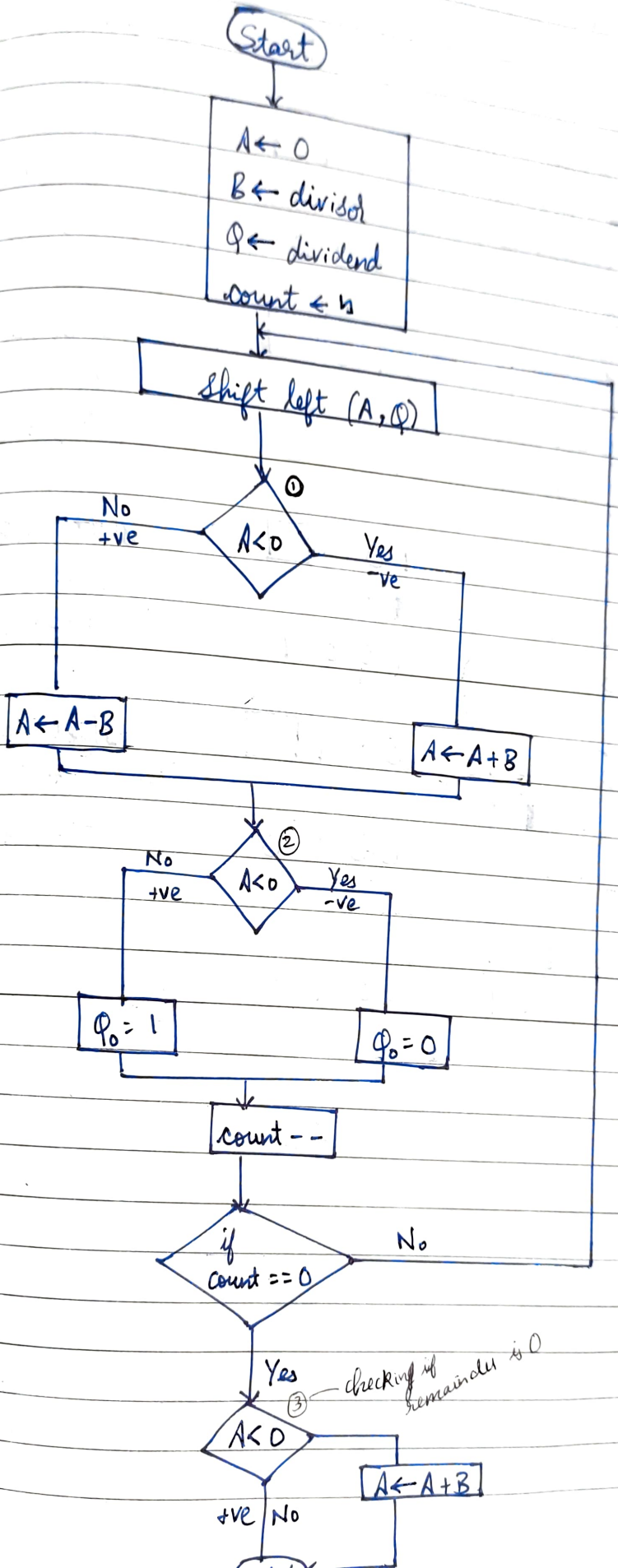
10 ÷ 3

Dividend

A	Q	operation
0 0 0 0 0	$Q_3 Q_2 Q_1 Q_0$ (8 4 2 1)	
0 0 0 0 0	1 0 1 0	initialize
A 0 0 0 0 1	0 1 0 □	shift left
(-B) 1 1 1 0 1		
+		
A 1 1 1 1 0		I cycle
A is -ve ; set $Q_0 = 0$		
B 0 0 0 1 1	0	
(+)		set $Q_0 = 0$
0 0 0 0 1	0 1 0 0	$A \leftarrow A + B$
0 0 0 1 0	1 0 0 □	shift left
(-B) 1 1 1 0 1		sub (A - B)
+		
1 1 1 1 1		II cycle
-ve set $Q_0 = 0$		
B 0 0 0 1 1	0	
		set $Q_0 = 0$
0 0 0 1 0	1 0 0 0	$A \leftarrow A + B$
0 0 1 0 1	0 0 0 □	shift left
-B 1 1 1 0 1		A - B
0 0 0 1 0		III cycle
+ve set $Q_0 = 1$		
0 0 0 1 0	0 0 0 1	$Q_0 = 1$
0 0 1 0 0	0 0 1 □	shift left
-B 1 1 1 0 1		A - B
0 0 0 0 1		IV cycle
set $Q_0 = 1$		
remainder 0 0 0 0 1	0 0 1 1 quotient	$Q_0 = 1$

Practical 2: Non - Restoring Division

Algorithm:



eg: 11/5

Q = 1011
B = 0101
-B = 1011
Dividend

A	Q $Q_3 Q_2 Q_1 Q_0$	operation	
00000	1011	initialize	
00001	011 □	shift left	1st
+ (-B) 11011		A ← A-B	
11100	0110	-ve ; Q_0 = 0	
11000	110 □	shift left	2nd
+ (+B) 00101		A ← A+B	
11101	1100	-ve ; Q_0 = 0	
11011	100 □	shift left	3rd
(+B) 00101		A ← A+B	
00000	1001	+ve ; Q_0 = 1	
00001	001 □	shift left	4th
(-B) 11011		A ← A-B	
11100	0010	-ve ; Q_0 = 0	
(+B) 00101			
00001	0010	checking remainder	

remainder = 1 quotient = 2

Q = 1100
B = 00100
-B = 11100

11011 8421
11100 1100

q: 12/4
Non-restoring

A	Q	$q_3 q_2 q_1 q_0$	cp	
00000	1100			initialize
00001	1000			Shift left
(-B) 11100				A ← A-B
11101	1000			-ve ; q_0 = 0
11011	0000			Shift left
(+B) 00100				A ← A+B
11111	0000			-ve ; q_0 = 0
11110	0000			SL
(+B) 00100				A ← A+B
00010	0001			+ve ; q_0 = 1
00100	0011			SL
(-B) 11100				A ← A-B
00000	0011			+ve ; q_0 = 1
R=0	Q=2			
00000	0110			SL
(-B) 11100				A ← A-B
11100	0110			-ve ; q_0 = 0
(+B) 00100				Remainder check
X 00000	0110			

3

q. M = 00011 = (B) Divisor -B = 11101

S	A	Q
0	0000	1000

8 | 3

8

Restoring Division

A Q op.

00000 $Q_3 Q_2 Q_1 Q_0$

00000 1000 initialize

A 00001 000 □ shift left

-B 11101 A ← A-B

(1)11110 1

11110 set Q_0 = 0

+B 00011 A ← A+B

00001 0000

00010 000 □ SL

-B 11101 A ← A-B

11111 2

(1)11111 set Q_0 = 0

+B 00011 A ← A+B

00010 0000

00100 000 □ SL

-B 11101 A ← A-B 3

(0)00001 0001 set Q_0 = 1

00010 0010 □ SL

-B 11101 A ← A-B 4

(1)11111 set Q_0 = 0

A ← A+B

```
    1 1 1 1 1
+B  0 0 0 1 1
    0 0 0 1 0
```

R = 2

0 0 1 0

Q = 2

} 4

q: 12/4 restoring Q = 1100 B = 00100 -B = 11100

A	Q $Q_3 Q_2 Q_1 Q_0$	op
00000	1100	initialize
00001	100 □	SL
A		A ← A-B
-B 11100		
11101		
11101		set Q_0 = 0
+B 00100		A ← A+B
00001	1000	} 1
00011	000 □	SL
-B 11100		A ← A-B
11111		
11111		set Q_0 = 0
+B 00100		A ← A+B
00011	0000	} 2
00110	000 □	SL
-B 11100		A ← A-B
00010	0001	set Q_0 = 1 } 3
00100	001 □	SL
-B 11100		A ← A-B
00000	0011	set Q_0 = 1 } 4

R = 00000

Q = 0011

20th October, 2023

LAB

1. Design 6:64 decoder using 2:4 decoder and 3 ip and gates

2. $y = a\bar{b} + \bar{b}\bar{c} + \bar{a}bc$
   a) using 8:1 Mux
   b) using 4:1 Mux
   c) using 2:1 Mux

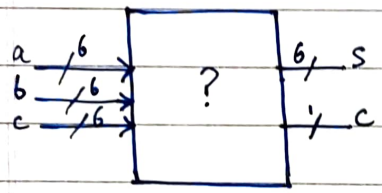
25th October, 2023

Carry Save Adder

↳ prev carry doesn't matter

eg: $y = a + b + c$

c is not carry-in

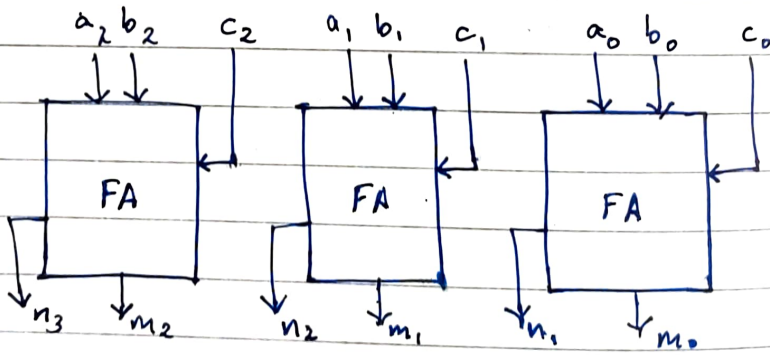


$A = 45 \rightarrow 1\ 0\ 1\ 1\ 0\ 1$ (a_0)

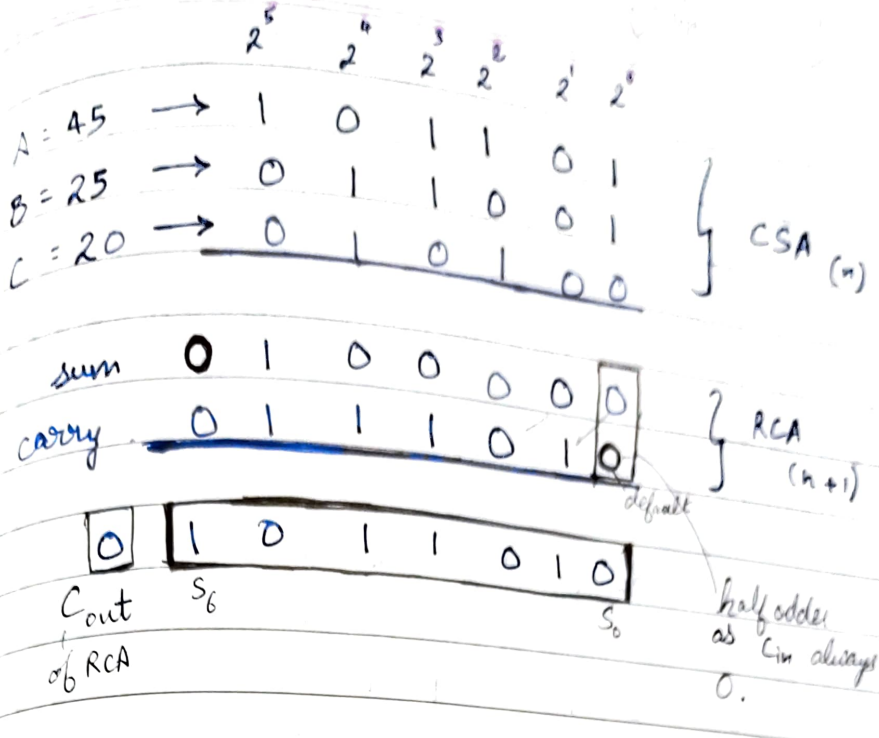
$B = 25 \rightarrow 0\ 1\ 1\ 0\ 0\ 1$ (b_0)

$C = 20 \rightarrow 0\ 1\ 0\ 1\ 0\ 0$ (c_0)

CSA



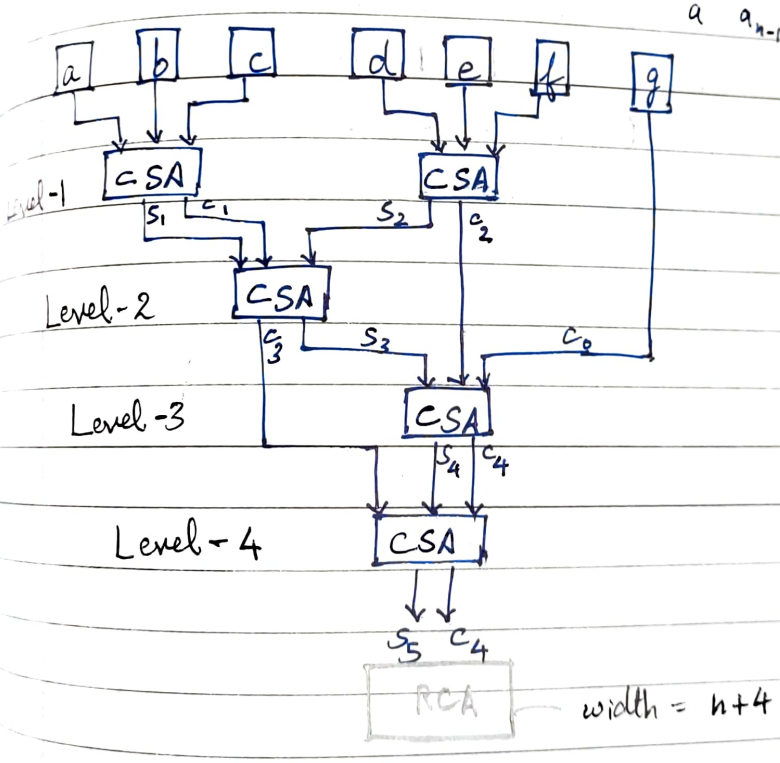
* carry won't propagate



→ 7 n-bit binary nos.

n-bit

$a\ a_{n-1}\ a_{n-2} \cdots a_2\ a_1\ a_0$



CSA increases width by 1

CPA = RCA

carry Propagate Adder

eg: a = 17
b = 22
c = 20
d = 32
e = 44
f = 62
g = 09

7 6-bit nos.
N = 6

	32	16	8	4	2	1
a	0	1	0	0	0	1
b	0	1	0	1	1	0
c	0	1	0	1	0	0
d	1	0	0	0	0	0
e	1	0	1	1	0	0
f	1	1	1	1	1	0
g	0	0	1	0	0	1

a	0	1	0	0	0	1	
b	0	1	0	1	1	0	
c	0	1	0	1	0	0	
sum	0	0	1	0	0	1	S_1 ✓
carry	0	1	0	1	0	0	C_1 ✓

d	1	0	0	0	0	0	
e	1	0	1	1	0	0	
f	1	1	1	1	1	0	
sum	0	1	1	0	0	1	S_2 ✓
carry	1	0	1	1	0	0	C_2 ✓

S_5 | 46
C_5 | 160

S_1 0 0 1 0 0 1 1
C_1 0 1 0 1 0 0 0
S_2 0 1 1 0 0 1 0

sum 0 0 0 0 1 0 0 1 S_3
carry 0 1 0 0 1 0 0 C_3

S_3 0 0 0 0 1 0 0 1
C_2 0 1 0 1 1 0 0 0
g 0 0 0 0 1 0 0 1

sum 0 0 1 0 1 1 0 0 S_4
carry 0 0 0 0 1 0 0 1 0 C_4

S_4 0 0 1 0 1 1 0 0
C_4 0 0 0 0 1 0 0 1 0
C_3 0 0 1 1 0 0 1 0 0

sum 0 0 0 0 1 0 1 1 0 S_5 ✓
carry 0 0 1 0 1 0 0 0 0 C_5 ✓

128 64 32 16 8 4 2 1

* If CSA is adopted in generating products it's called Wallace Tree Multipliers.

a = 8
b = 15

q. Can CSA handle two's complement numbers ?

26th October, 2023

Fixed Point Numbers

eg $\overleftarrow{10}.\overrightarrow{24}_{(10)}$

integers (pre binary point) (m) — fractional (post binary point bits) (f)

$(1 \times 10^1) + (0 \times 10^0) + (2 \times 10^{-1}) + (2 \times 10^{-2})$

eg $\overset{2^2}{1}\ \overset{2^1}{0}\ \overset{2^0}{1}.\ \overset{2^{-1}}{0}\ \overset{2^{-2}}{1}_{(2)}$

m = 3 n = 2

m-1 → 0 -f

[3 : -2]

[m-1 : -f]

$(1 \times 2^2) + (0 \times 2^1) + (1 \times 2^0) + (0 \times 2^{-1}) + (1 \times 2^{-2})$

$(5.25)_{10}$

eg. 5.625

101 . 101000

0.5 $\frac{1}{2}$ 0 $\frac{0}{2^2}$ 0 $\frac{0}{2^3}$

0.125

0.625

m = 5

f = 8

00101.10100000

Floating Point

increase range by using exponents

101.101 fixed

5.625 → 0.5625 × 10¹

10.1101 × 2¹ floating

2^{-1} 2^{-2} 2^{-3} 2^{-4}

M

$1.\overline{01101} \times 2^{(2)}$ E

0.101101×2^3

101101.0×2^{-3}

[3 : -4] } default (8 bit)
m=4 n=4

0000.0000 to 1111.1111
0.0 to 15.9375

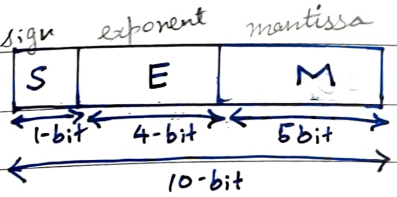
→ Normalization

1. Explicit
2. Implicit

} 2 techniques to represent no.

27th October, 2023

case (i)



(1)01.101 ⇒ $(5.625)_{10}$

Explicit LHS (E) (I) implicit RHS point

shift . to MSB 1 L or R

0.101101×2^3 1.01101×2^2

S = 0
E = 3
M = 10110(1)

in terms of 10 bit

10 bits { S = 0
E = 0011
M = 10110 } ignored

→ Biasing Chart - to find out if the exponent is -ve/+ve

2's comp case of 4-bits

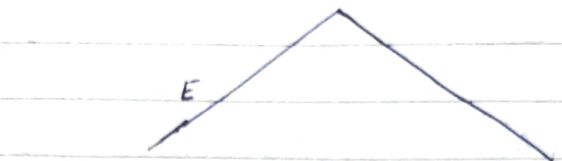
-8 -7 -6 -5 -4 -3 -2 -1 0 1 2 3 4 5 6 7

|max value| = 8 0000

+8 +8 +8 +8 .....

0 1 2 3 4 5 15

101.101



0.101101×2^{3} $\qquad$ $101101. \times 2^{-3}$

eg: $(0.0101)_2$

$\downarrow$

0.101×2^{-1}

S	E	M
0	1111	101 00

2s comp. of 1 $\qquad$ for decimals 0s Right sides no significa

* Biasing $\Rightarrow$ adding the max value

$(101.101)_2$

(E) $\boxed{0}.101101 \times 2^{3}$

$S = 0$

$E = 3 + 8 \Rightarrow (11)_d$ (Bias)

$M = 10110$

0	1011	10110

(I) 1.01101×2^{2}

$S = 0$

$E = 2 + 8 = (10)_d \Rightarrow 1010$

$M = 01101$

0	1010	01101

(E) $(-1)^S \times 0.M \times 2^{E-8}$

to reconstruct
& constructing back to source

(I) $(-1)^S \times 1.M \times 2^{E-8}$