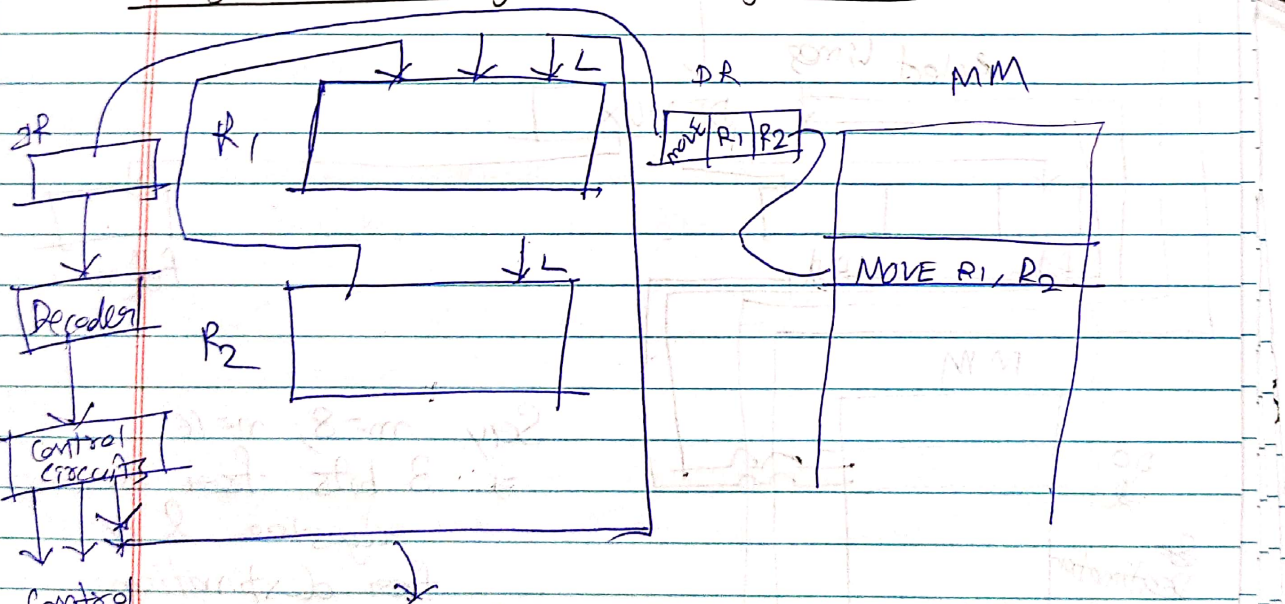
1/2/23
Wed Feb

Instruction Implementation

Data Movement

MOVE A_1, A_2
move from A_2 to A_1 → memory to memory
MOVE R_1, R_2
content of register R_2 transferred to R_1
LOAD X
AC ← M(X) memory to register
STORE X
M(X) ← AC register to memory

Register to register transfer



We have to realise which control signal realises register to register transfer
So activate R_1

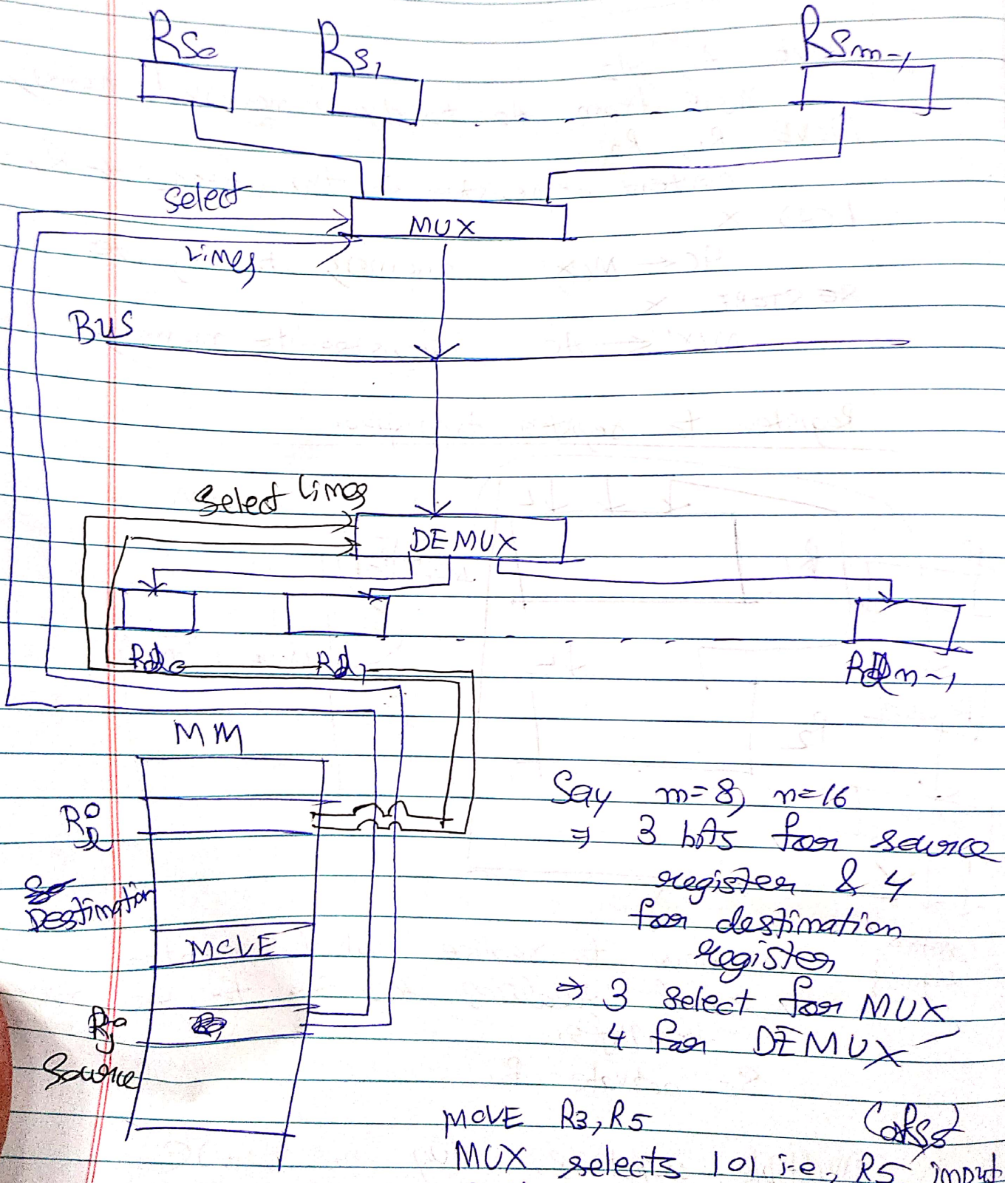
Multiple registers may be there
We can have MOVE R_2, R_1
Also more than 2 registers.

(2)

⇒ Many sources, many destinations

Say MOVE R_i R_j

Say m source registers, n destination registers



Say m=8, n=16

⇒ 3 bits for source register & 4 for destination register

⇒ 3 select for MUX
4 for DEMUX

MOVE R_3, R_5

MUX selects 101 i.e., R_5 input

DEMUX selects 011 i.e. R_3 output

Register to Memory

In Von Neumann x there

MOVE A_1, R_1

So 1st transfer R_1 to DR, DR to memory address of A_1

AR ← DR(operand), DR ← R_1, M(AR) A_1 ← DR

3 micro operations → All data movement

If memory to memory → ↑ No. of data movement operations

IN 07
↓
transfer content of input 07 to CPU i.e., Ac
≡ Register to register

Similarly AC to output is also data movement

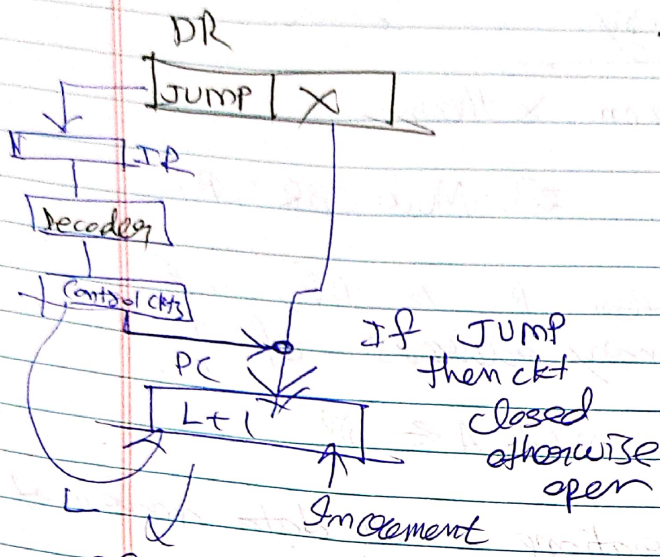
I/p, o/p data instruction transfer complete → Similar as above

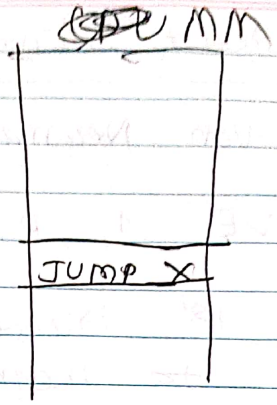
Branch Control Instruction

- Conditional
  - ↓
  - JUMPZ x
- Unconditional
  - ↓
  - JUMP x
  - PC ← x

4

JUMP





Every PC has signal line called increment

PC → Counter Register

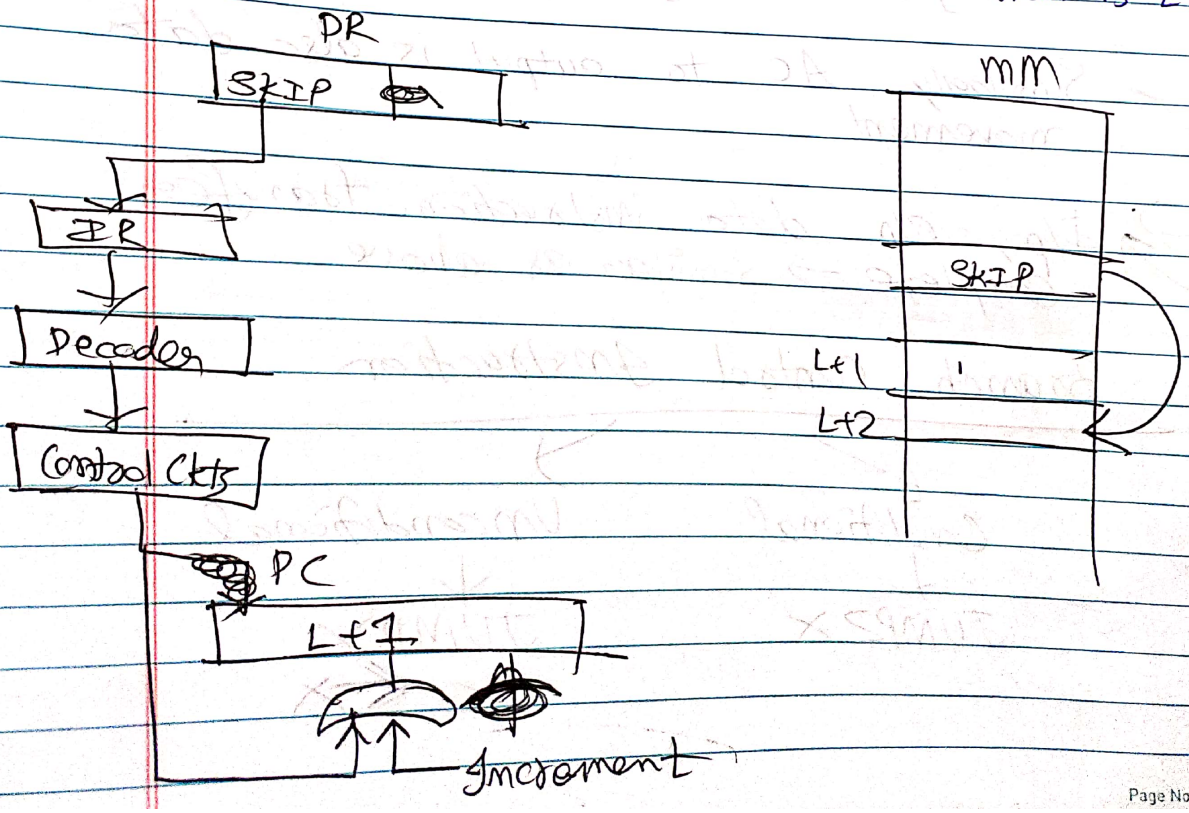
If A active transfer to PC

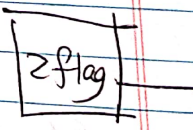
If L location instruction execute then L+1 in PC

SKIP → No explicit operand

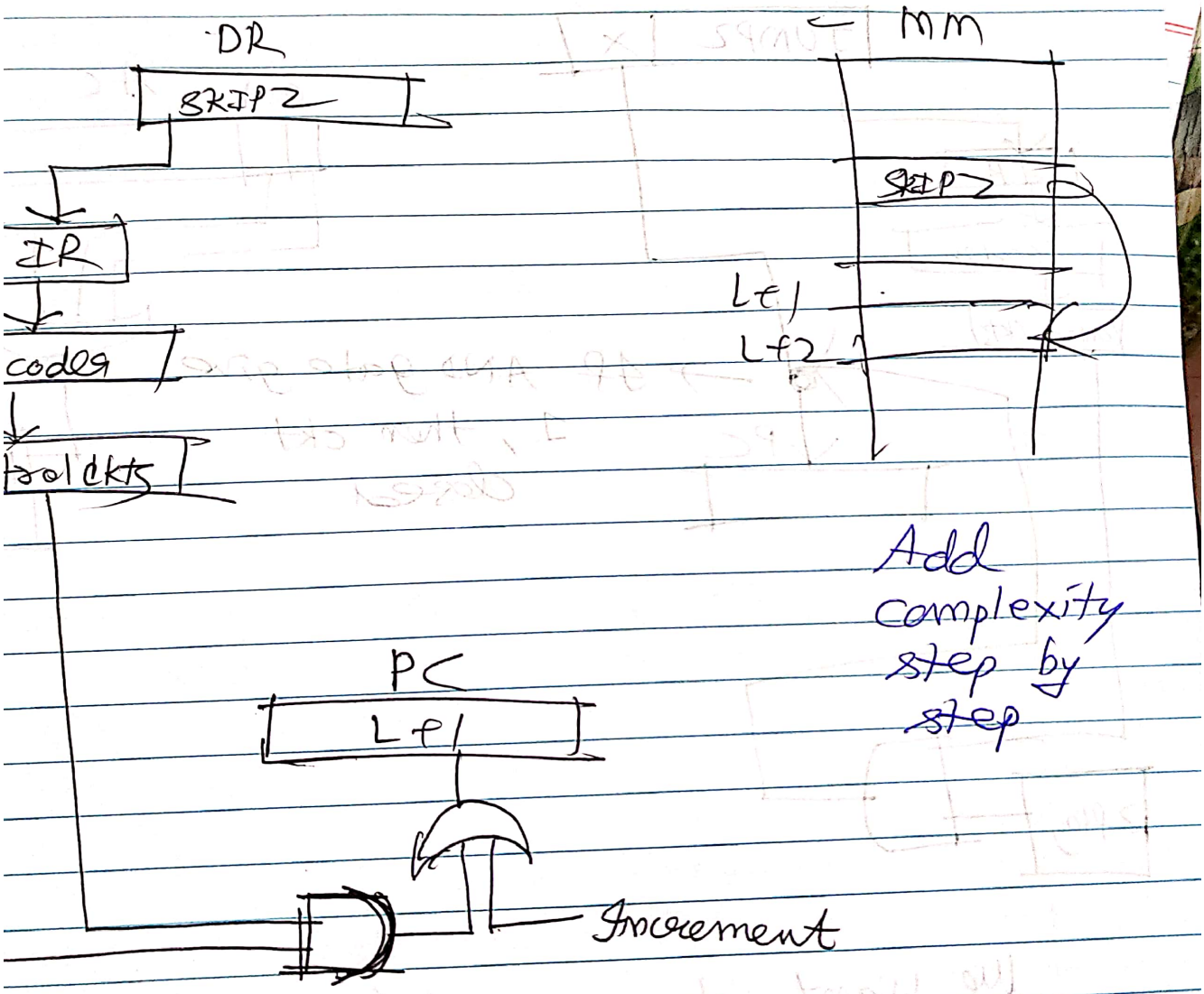
Skip instruction at L+1

Next executable instruction is L+2





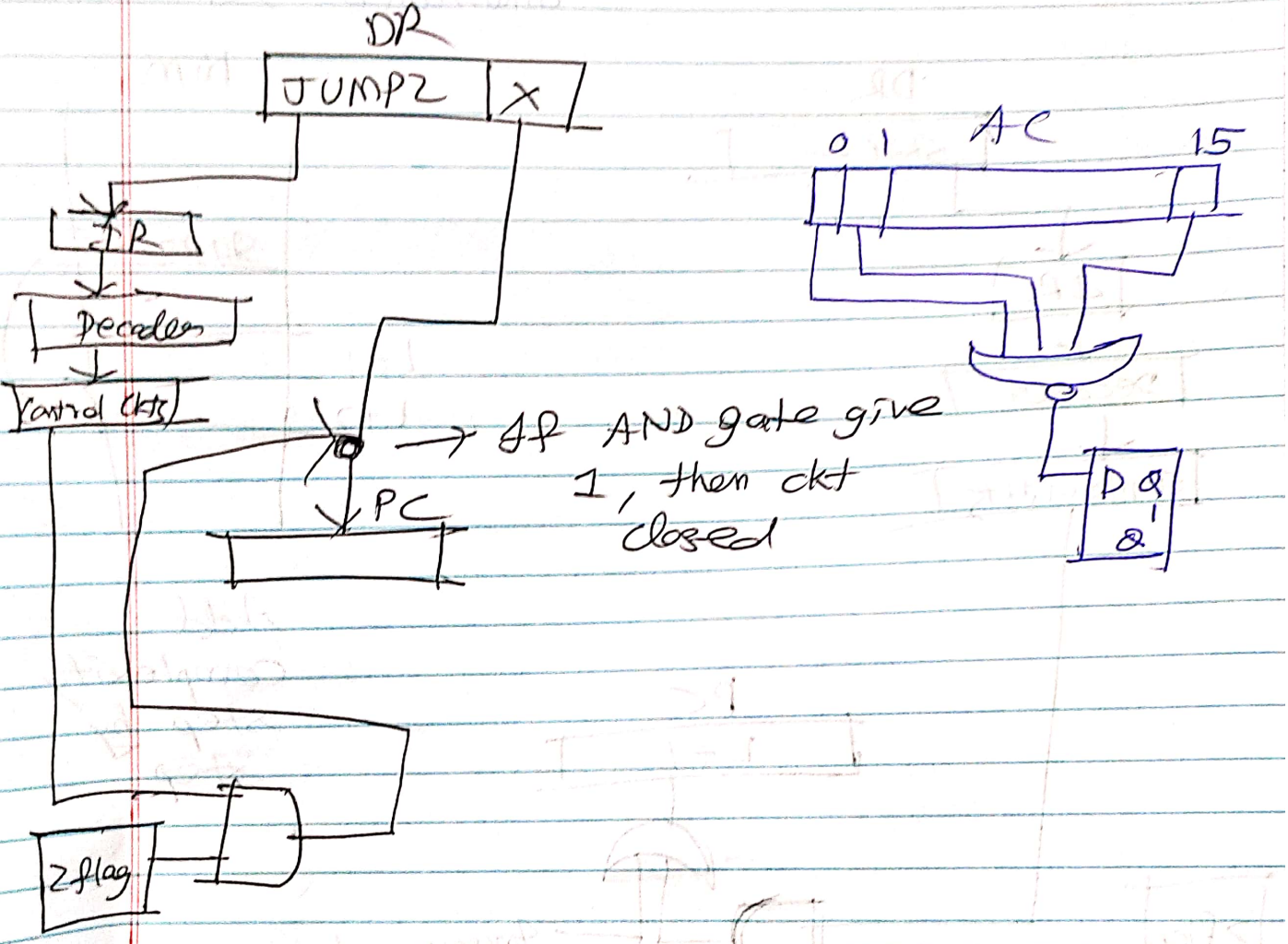
SKIP Z → Skip if Z at reset
If Z flag true
then skip
otherwise X



Add complexity step by step

All branch instructions are register to register instructions

JUMPZ X → X transferred to PC if Z flag true



We want to accurately give accumulator content

If AC = 0 → Set Z flag,

For this use D flip flop

Say AC has 16 bits

Check all 16 bits 0 or not for AC=0

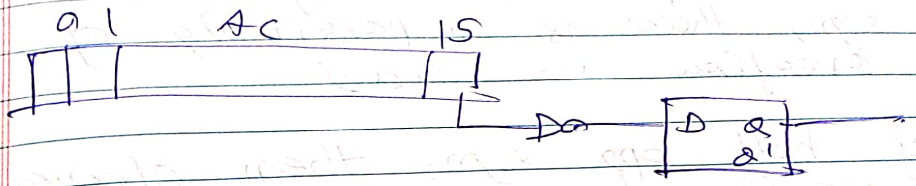
If all 0, NOR gate output 1

7

apsara

Date: / /

Say have flag set when ~~such~~ ~~that~~ jump only if AC has ~~content~~ even content
Check LSB only → If 0, then even



Say have flag set with parity
↳ Check all bits

Say jump if +ve content in AC
Check MSB ⇒ If 0 +ve

Say jump on overflow
Overflow
Store/Add ↑ no. of instances } → At a stage contains
or exceed certain val. } X store

Here val. → Least / Max. value
container → register

For signed magnitude with n bits

$-2^{n-1}+1$ to $2^{n-1}-1$ → X exploit all 2^n combinations
as 0000, 1000
both 0
Similar in 1's complement

∴ Use 2's complement

Page No.: / /

In addition, subtraction overflow can occur when

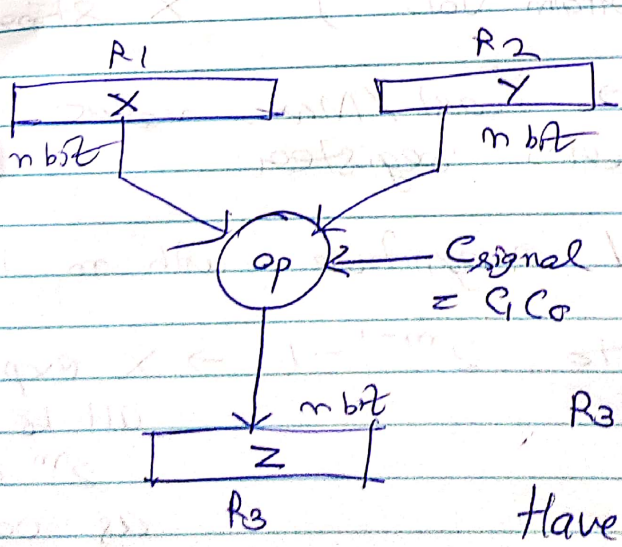
if both operands are of same sign, there is a possibility of overflow in add

if both opp. sign, then chance of overflow in subtract.

How to set 2's complement overflow condition?

2/12/23 Thu Feb

Logical Instructions



Same logic ckt. support diff. operations

$R_3 \leftarrow R_1$ op R_2

Have to decide at time t which operation executed → Decide by control

Say have AND, NAND, OR, NOR, NOT

apsara

Date: __/__/____

If SoP → Consider for which Z true
If PoS → Consider for which Z false

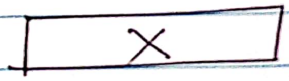
Say SoP

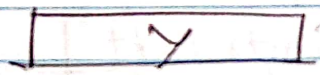
Say at $C_1 C_0$

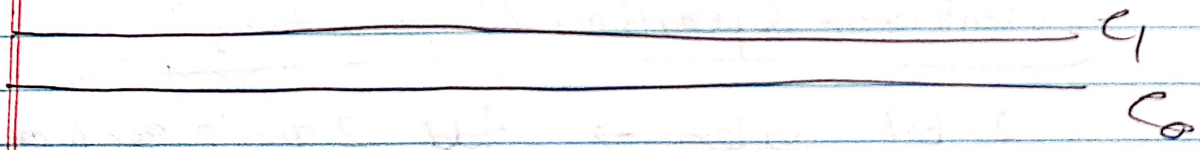
00	NOT
01	AND
10	NAND
11	OR

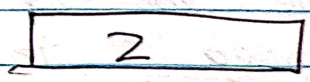
Expression of Z:

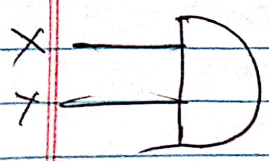
$$Z = \bar{C_1}\bar{C_0}\bar{X} + \bar{C_1}C_0 X.Y + C_1\bar{C_0}(\overline{X.Y}) + C_1C_0(X+Y)$$









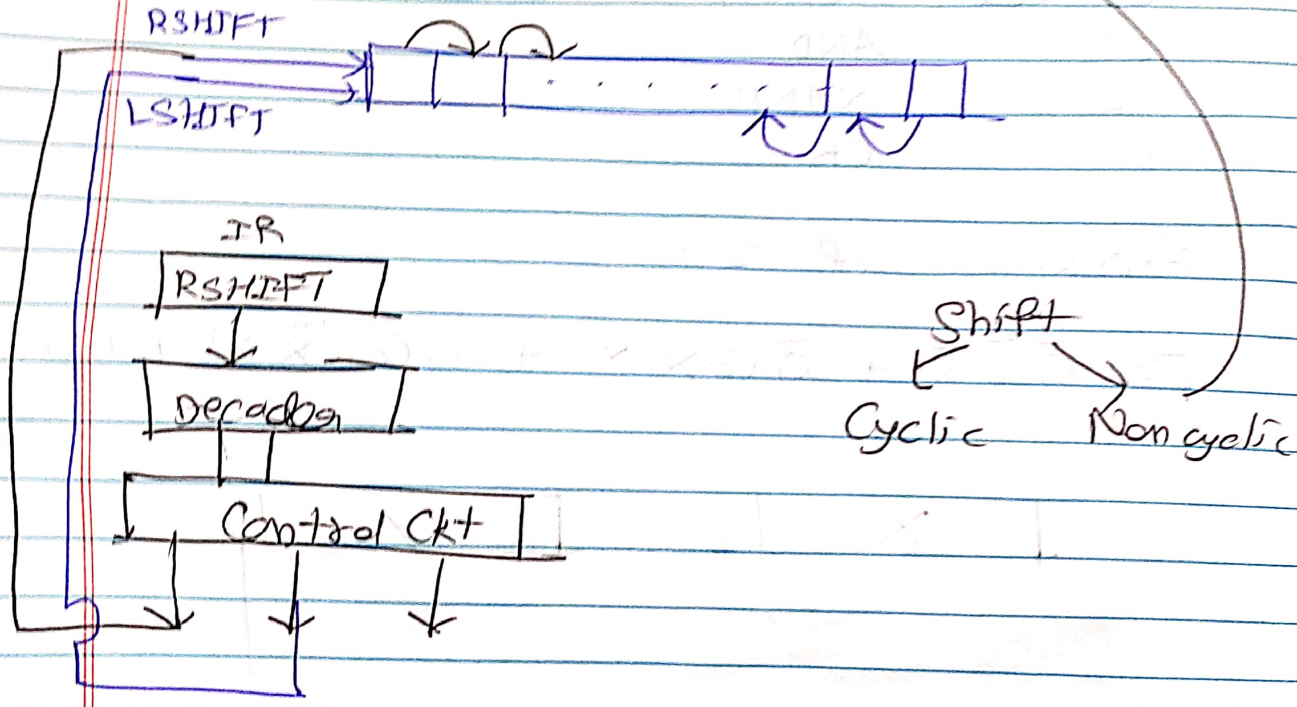


Word AND gate → Has n 2 input AND gate

Shift Operations / Instructions

RSHIFT — Content of AC right shifted

Design AC such that shift register

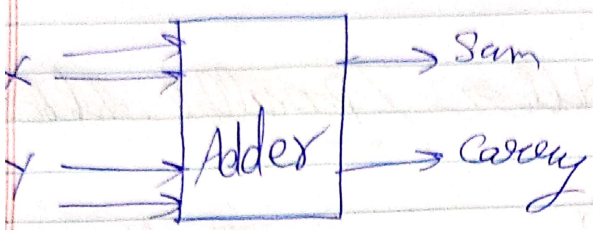


Arithmetic Operations / Instructions

2 bit adder → Add 2 no.'s each of 2 bit

Addition is done lot of times. Even if x in prog. to execute instructions CPU may do addition

Binary Adder



Delay independent of no. of var.

X Y Sum Carry (final)

x_i x_0 y_i y_0 s_i s_0

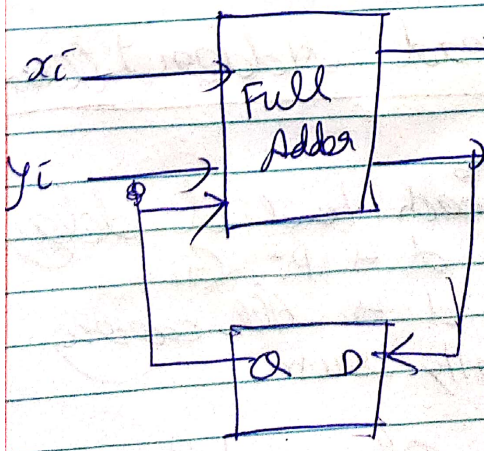
If say SOP form → 2 levels ~~2 lt~~
X Have lesser delay than this
∴ Fastest adder.

But X use binary adder in computer as ↑ gates required.
Hardware cost is exponentially high

for 4 variable to get product terms need 4 input gate
Fan in ↑ as no. of var.'s ↑

Serial Adder

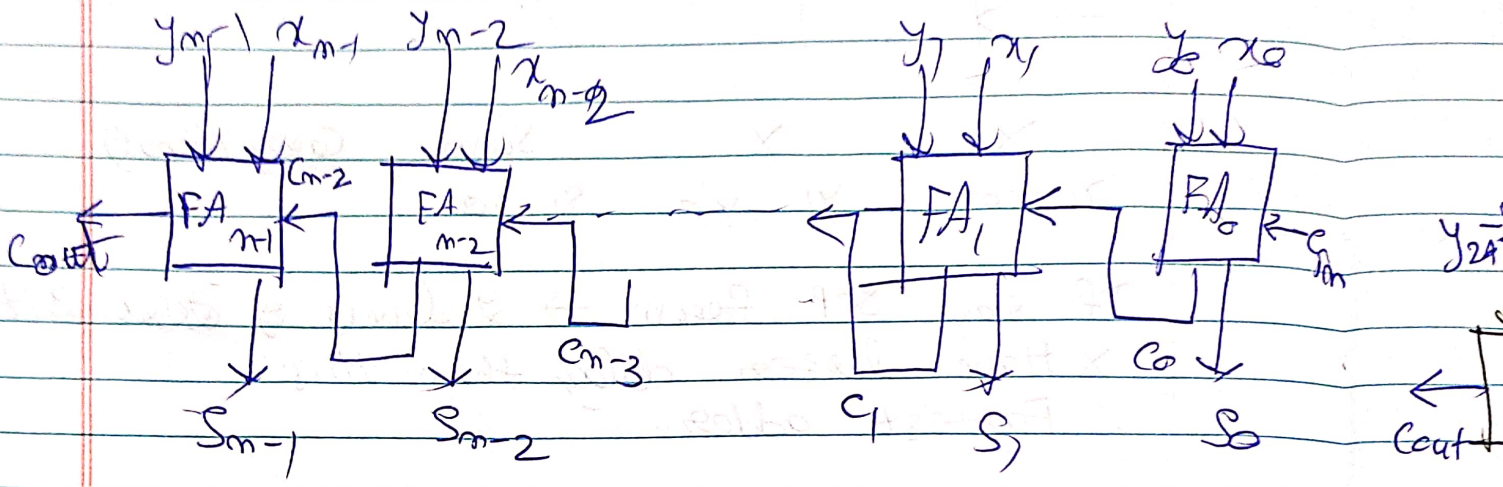
Least cost adder (hardware cost)



Delay
$n(t_1 + t_2) \propto n$
t_1 → delay of full adder
t_2 → delay of flip flop

Want adder with delay close to binary adder & cost close to serial adder

(3) Ripple Carry Adder / Parallel Adder



Let each full adder have d delay

Delay $= nd$ α n

$nd < n(t_1 + t_2)$ → Faster than serial adder

Cost αn → Less than binary adder

SM want delay close to binary adder

(4) Carry Look Ahead Adder (CLA)

Delay → $3d$ → Each level d delay

1st d → P_i, G_i

2nd d → All carry

Finally sum

Delay independent of n bits

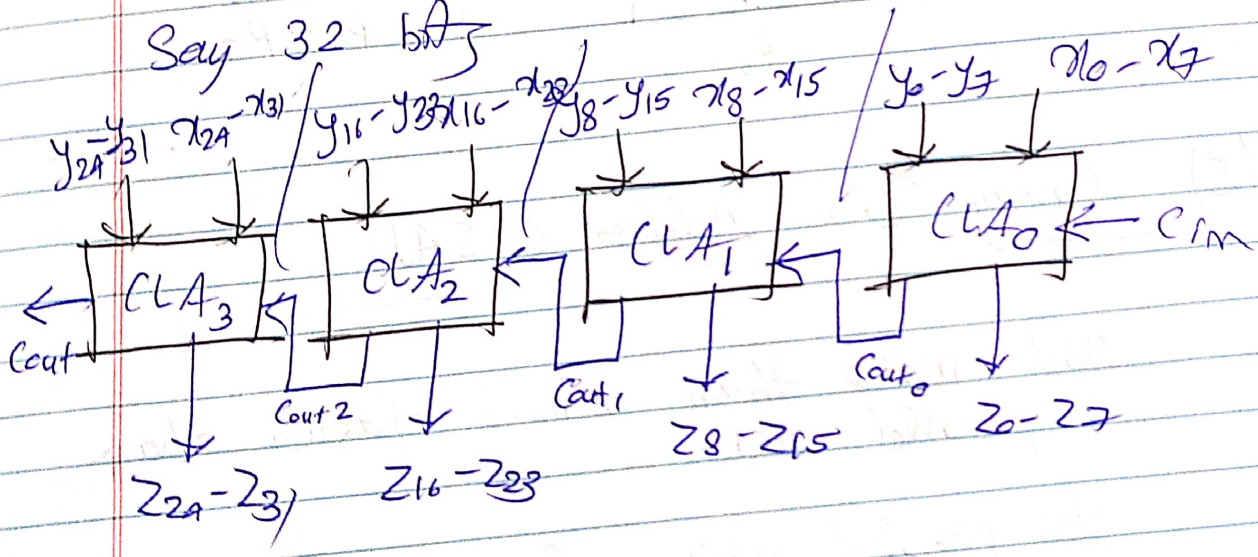
Page No.

Fan in ↑
Expression for carry complex
In product term, lot of variables

Max 8 bit, 16 bit carry look ahead adder

For higher bits use cascading

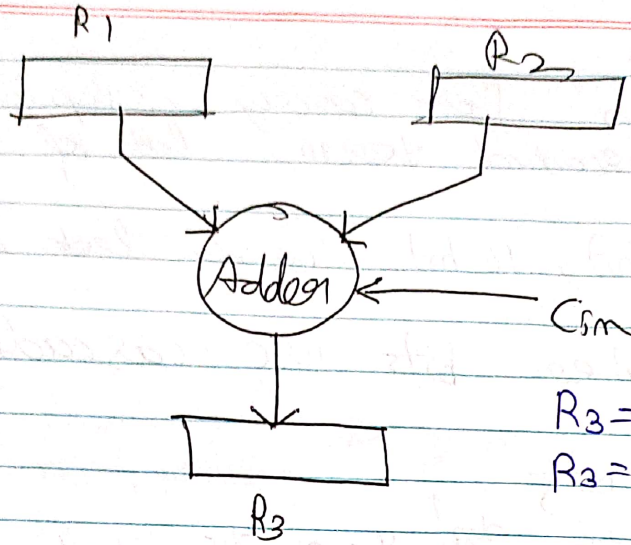
Say 32 bits



Delay → Each CLA 3d delay
1st Carry comes in 2d time
After that each carry d time

$2d + d + d + d + d = 6d$ for 4 CLA's

2d → 1st carry
d + d + d → Remaining Carry
d → Sum



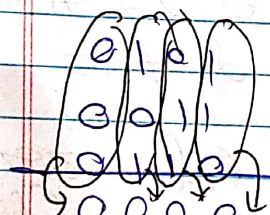
Adder & Subtractor

$R_3 = R_1 + R_2$
$R_3 = R_1 - R_2$

(5) Carry Save Adder (CSA):

Add more than 2 no.'s faster than parallel adder

In parallel adder for adding 3 n bit no.'s → 2nd ∴ 2 additions



Sum } 4 full adders → d ∴ only full adder delay irrespective of n
Carry }

Sum
Carry } Parallel Adder } → nd
(Flip prev. carry)
(Shift carry)

Get correct sum
Carry

∴ d + nd < 2nd

apsara
Date:

Say multiply 2 six bit nos

⇒ Product size < 12 bits (2n)

32 → Multiplicant
× 23 → Multiplier
96 → Partial product
64 x
↑ Wt. based on position of multiplier

2 six bit nos multiply ⇒ 6 partial product & Add 6 no.'s

Parallel → 6(2n)d
∵ Each partial product 2n bit
Lot of time/delay

∴ CSA adders help us

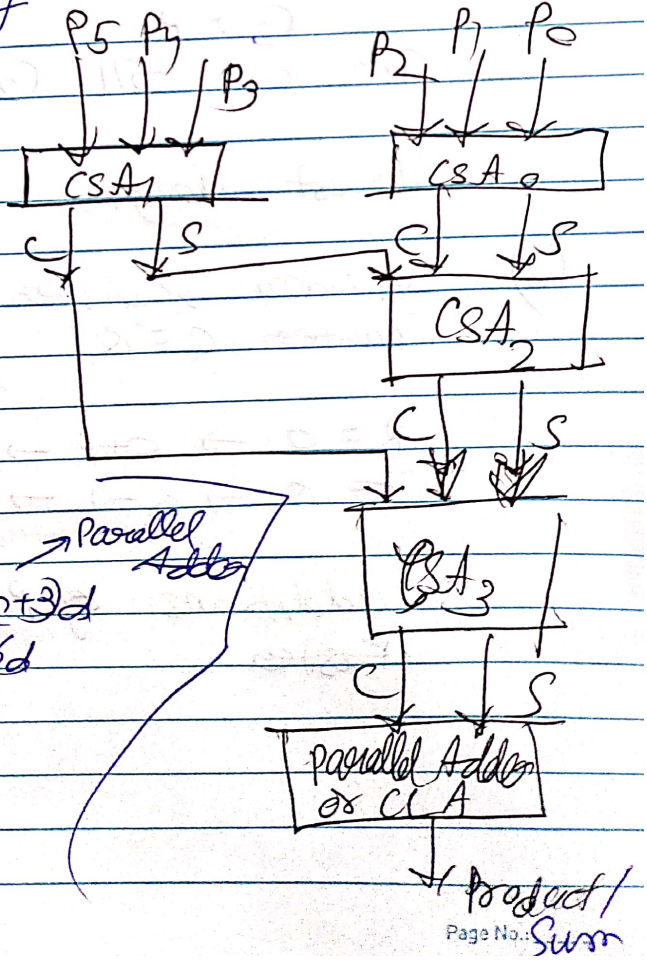
0 1 0 1
0 0 1 1
0 1 0 0
0 0 0 0 } Say CSA
0 1 1 1

Split each of no.'s in (3+3) bits & supply to each CSA

Total: d+d+d+nd = (n+3)d → Parallel Adder

For CSA → 3d+3d=6d

∴ Very fast



apsara
Date:

Multiplication of 2 numbers in binary

(1) Use counter

Say $P \times Q$
multiplicant → multiplier

Let $C_1 = 0$, $C_2 = Q$, $C_3 = P$

Increment $C_1 = 0 \to 1 \to 2 \to \ldots \to P$
$C_2 = Q$
Decrement $C_3 = P \to P-1 \to P-2 \to \ldots \to 0$
After this
$C_2 = Q \to Q-1$
Now again $C_3 = P$
Increment $C_1 = P \to P+1 \to P+2 \to \ldots \to 2P$
$C_2 = Q-1$
Decrement $C_3 = P \to P-1 \to P-2 \to \ldots \to 0$
$C_2 = Q-1 \to Q-2$
$C_3 = P$
Go on till $C_2 = 0$

Slowest design

(2) Consider register variable R
Counter $C = Q$

$R = 0 \to 0+P \to P+P \to 2P+P \to \ldots PQ$
$C = Q \to Q-1 \to Q-2 \to Q-3 \to \ldots 0$

Q additions of P
Faster

apsara

Date:

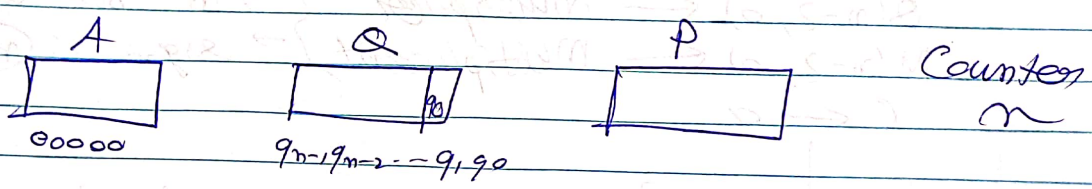
Let

P = $P_{n-1} P_{n-2} \cdots P_1 P_0$

Q = $Q_{n-1} Q_{n-2} \cdots Q_1 Q_0$

n partial products & n sums of those

Let 3 registers A, Q, P



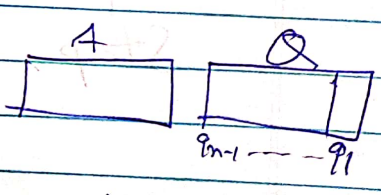
~~If LSB of Q = 0,~~

Next partial product & prev. partial product should be properly adjusted

```
 1 0 1
 1 0 1 0
 0 0 0 0
 1 1 0 1      → Partial Product shifted left
```

OR can shift 0000 to right

So shift both A, Q right



If LSB of Q = 0 ⇒ X add anything

Q = 1 ⇒ A ← P

Now shift A, Q right

Process goes on n times

A, Q store final product.

8/2/23
Wed Feb

Consider multiplication of 2 no.'s in signed magnitude form

Start

A(n-2, 0) ← 0
Q(n-2, 0) ← multiplier
M(n-2, 0) ← Multiplicand
} → n^{th} bit will be → sign i.e. (n-2)

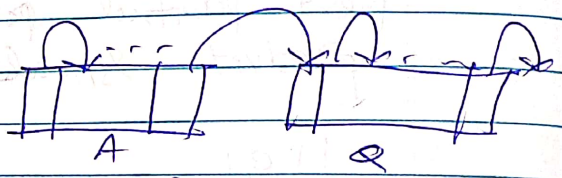
C ← 0

Q[0] = 1 ?
↓ yes
A ← A + m
↓ No

RS A, Q
C = n-1 ?
No ↓
C = C+1
Yes ↓

Output A, Q
↓
Stop

A stores MSB of product



Right most bit get out

MSB of AQ i.e. of A have to be modified
↓
We only get magnitude from above

Multiply signed bit of M, Q using XOR gate

Say 0 1010 ≡ + (10)$_{10}$
X 1 1101 ≡ − (13)$_{10}$

apsara

Date: / /

Algorithm N = 5

C

A	Q
0000	1101

0

+ 1010

1010

New RS

A	Q
0101	0110

0 ≠ N-1 i.e, 5-1=4

∴ C=1

Only RS

A	Q
0010	1011

1 ≠ 4

∴ C=2

+ 1010

1100

RS

A	Q
0110	0101

2 ≠ 4

C=3

+ 1010

10000

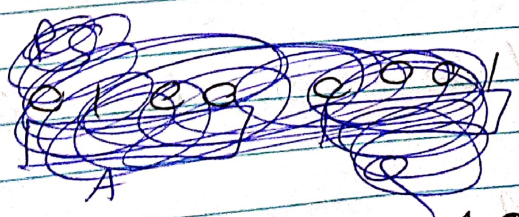
RS

A	Q
1000	0010

3 ≠ 4

C=4

4 = 4



∴ Product is AQ i.e, = 1000 0010

= (130)10

Need to change algo. to multiply 2's complement numbers

$Y = y_{n-1} y_{n-2} \ldots y_0$ — Multiplicand
$X = x_{n-1} x_{n-2} \ldots x_0$ — Multiplier

Multiply in 2's complement

Case 1: $y_{n-1} = x_{n-1} = 0 \Rightarrow$ +ve nos
Our algo. sufficient
Get pdt.

Case 2: $y_{n-1} = 1$, $x_{n-1} = 0$
Multiplicant −ve
Multiplier +ve
Q→ Show magnitude
M = Y → −ve
If follow our algo. if correct
M.Q = −ve ∴ −ve × +ve = −ve.
Change of algo.
check for 1 in MSB of A to see if −ve or not

Case 3: $y_{n-1} = 0$, $x_{n-1} = 0$

Say ~~toto~~ $0101 \times 1010 \equiv (5)_{10} \times (10)_{10}$
Considering magnitude
111
$(50)_{10}$
✗ correct
as 1010 = 2's comp.
$\equiv (-6)_{10}$
So ans. should be $(-30)_{10}$ but we get $(50)_{10}$

Assumed magnitude 10 but actually it is 10 − 16

apsara

Date: __ . __ . ____

∴ Need Correction
Actual product } ≠ Product by Algo. — (2^n • Multiplier)

Here 4 bits ⇒ $n = 4$ ∴ Subtract $16 \times 5 = 80$

2's complement in decimal for n bit
nc. = -2^n + Expansion of nc.

Case: IV: $y_{n-1} = x_{n-1} = 1$
Follow corrections of both cases 2, 3.

∴ Need Correction

Actual product } ≠ Product by Algo. — (2^n · Multiplier)

Here 4 bits ⇒ $n = 4$ ∴ Subtract $16 \times 5 = 80$

2's complement in decimal for n bit

nc. = -2^n + Expansion of nc.

Case: IV: $y_{n-1} = x_{n-1} = 1$

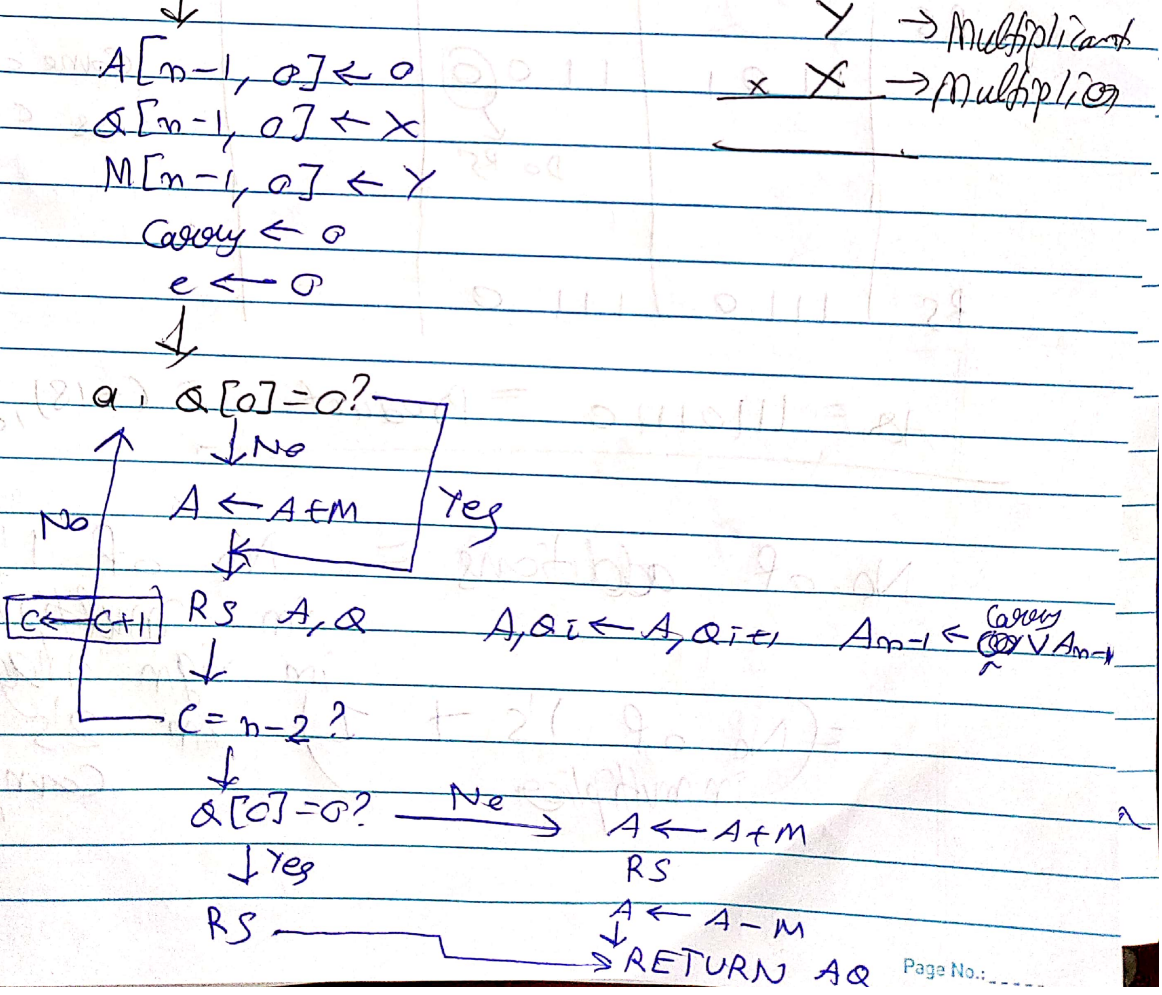
Follow corrections of both cases 2, 3.

9/2/23 Thu Feb

General Algorithm

Y → Multiplicand

X → Multiplier



Page No.:

$(-6)_{10}$ $(3)_{10}$

Let $Y = 1010$, $X = 0011$

$n = 4$

Carry	A	Q	M	C	
0	0000	0011	1010	0	
	0000				
	1010				
0	1010				
RS					
	1101	0001		1	$\because 0 \neq 2$
	1101				
	1010				
1	0111				
RS	1011	1000		2	$\because 1 \neq 2$
RS	1101	1100			Come out of loop as $c = 2$
		Do RS			
RS	1110	1110			

$AQ = 11101110 = \text{Product} = (-18)_{10}$

No. of additions = No. of 1's in multiplier in sign magnitude

= (No. of 1's in multiplier + 1) in 2's complement

apsara

Date: __ __ __

0101 × 0101

1011 × 0100 / 0101

Let $Y = 1011 = -(5)_{10}$, $X = 1010 = (-6)_{10}$

$n = 4$

Carry	A	Q	M	C
0	0000	1010	1011	0
RS				
0	0000	0101		1 ∴ 0≠2
	0000			
	1011			
0	1011			
RS	1101	1010		2 ∴ 1≠2
RS	1110	1101		Come out of loop as C=2
	1110			
	1011			
1	1001			
RS	1100	1110		
	1100			
	0101			
1	0001			

$0001\ 1110 = (30)_{10}$

2 one's in multiplier (2+1) = 3 additions i.e. 2 addition / +1 subtraction

~~2+1+3+16~~ / 6

main memory

time

output

all proce... with mm... bcess li...

addre... also ha... e → ... → ...

→ outp...

addre...

→ Jmp ... Out...

l

-ve

PU ... ad ... whic... IR

apsara

Date:

We want to reduce no. of 1's so that can reduce no. of additions ∴ have to change algo. as addition is costly.

For this Booth proposed an algo. for multiplication of 2's complement numbers.

Booth Algorithm

$1011 = M$

$0110 = Q = (6)_{10}$

Value of pdt. $= 6M = 8M - 2M$

$8M$ → Equivalent to 3 RS

$2M$ → 1 RS

But if use $Q \rightarrow$ 2 additions

1 Subtraction

Say

$Q = 0111110 = (62)_{10}$ → Need 5 additions

$62M = 64M - 2M$ → 1 subtraction

There are 2 flips:

$0 \rightarrow 1$ → Here subtract $2^{\text{index of } 1} M$

6 5 4 3 2 1 0

0 1 1 1 1 1 0 → $-2^1 M$

$1 \rightarrow 0$ → Here add $2^{\text{index of } 6} M$

6 5 4 3 2 1 0

0 1 1 1 1 1 0 → $+2^6 M$.

Page No.:

→ 0109

apsara

Date:

$$\text{Let } Q = \overset{7\,6\,5\,4\,3\,2\,1\,0}{0\,1\,1\,1\,0\,1\,1\,0}$$ → 5 additions based on no. of 1's

$$\text{Product} = -2^1M + 2^3M - 2^4M + 2^7M$$

↓

3 additions based on no. of 1's.

Booth's algo → ~~4~~ runs of 1's & 0's just RS.

Say have

$$Q = \overset{4\,3\,2\,1\,0}{0\,1\,1\,1\,1} = 2^4M$$ but this is 15 not 16

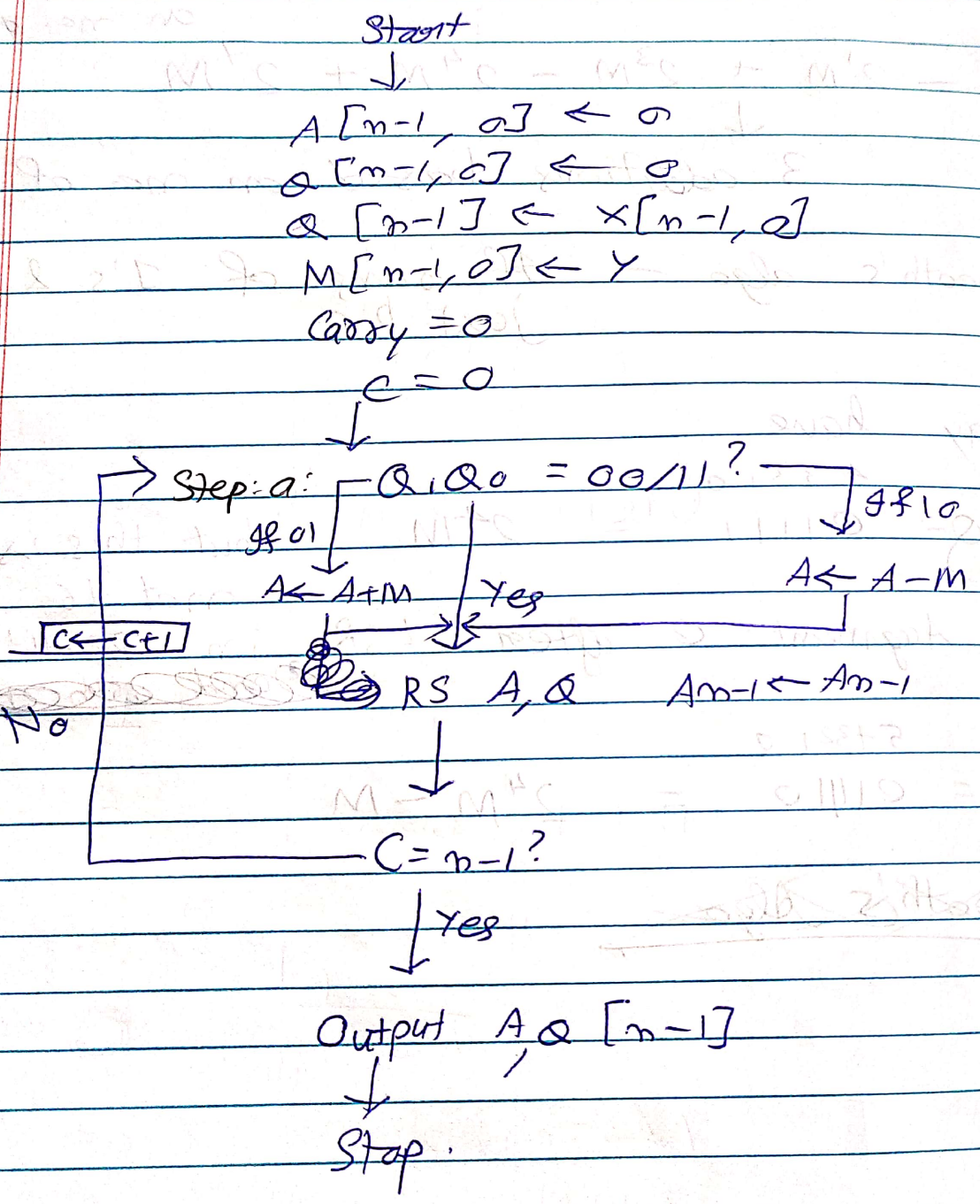
∴ Augment 0 after LSB in Booth's algo

$$Q = \overset{5\,4\,3\,2\,1\,0}{0\,1\,1\,1\,1\,0} = 2^4M - M$$

~~Booth's Algo~~

apsara
Date:

Booth's Algorithm :



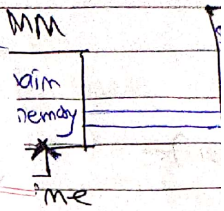
1011 → 0100

0101 = 5

27

apsara

Date:



Cargy	A	Q	M	C
	0000	1011	0110	0
0	0000 − 0110 = 1010	0101		0≠3 ∴ 1
RS	1101	0101		1≠3 ∴ 2
RS	1110	1010		2≠3
1	1110 + 0110 = 0100	1010		3=3
RS	0010 − 0110 = 1100	0101		
RS	1110	0010		

Product

Product = 11100010 = $(-30)_{10}$

output

ll proce
ith mm
ress l

addre
lso h
→
→

Outp

addre

Impo
Outp

ich
ns

Page No.:

1010
0101
1
0110

1101
0010
1
0011

28

SUCCESSFUL
Date
Page No.

M = 1101 = $(-3)_{10}$
Q = 1010 = $(-6)_{10}$

Carry	A	Q	M	C
0	0000	10100	1101	0
RS	0000	01010		0≠3 1
	0000 0011 0011			
0				
RS	0001	10101		1≠3 2
	0001 1101 1110			
0				
RS	1111	01010		2≠3 3
	1111 1111 0011 0010			
1				
RS	0001	00101		

111

(18)$_{10}$

Realisation in Hardware

3 registers A, Q, M
Registers should

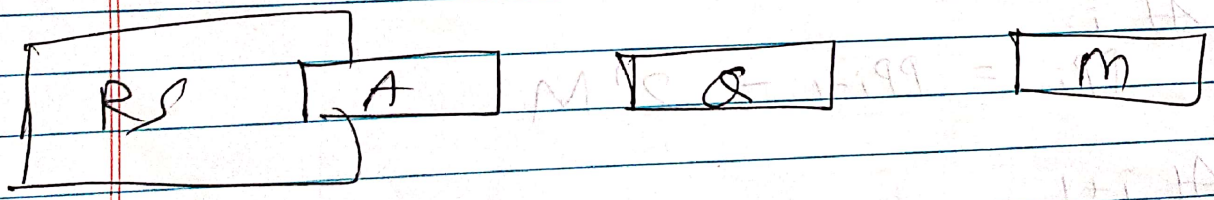
2 bits of Q → input to multiplier control
↓
Check Q_1, Q_0
↓
Either generate
Add & RS or
RS or
Sub & RS

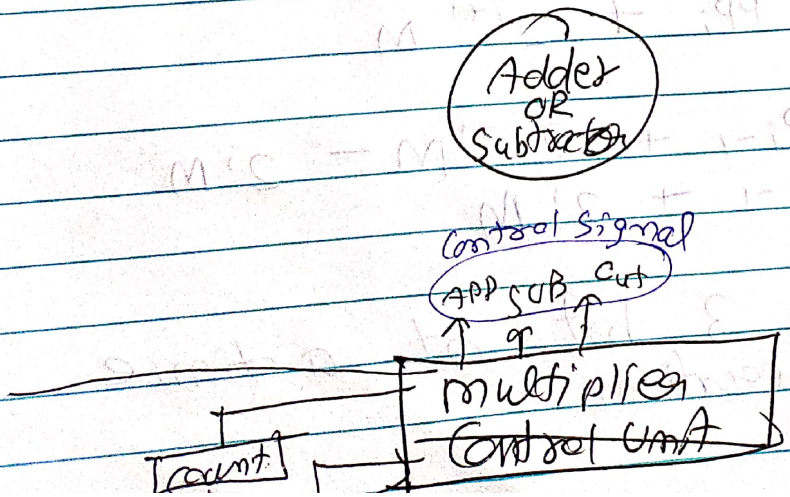
Also increament counter, change count value also.

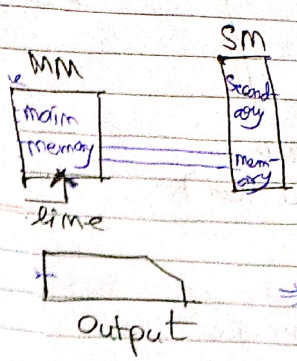
After addition/subtraction, result stored

Output of add, sub go to Ac

Finally A, Q content go to bus







all processing with mm
bus lines,

address
lso have
→ Output
→ Inpu
Output of

address li

Input to
Output a

data i

have 1
& 1
ch ope
ist suc

Runs of a very fast than operation

Drawbacks:

If alternate 0's & 1's in multiplier, it will be costly.

In case of

0 1 0 1 0 1 0 1 0 → 4 per prev. algo.
7 additions per Booth's algo.

Solution:

Bit Pair Multiplication Algo:

	i+1	i	i-1	
Q:	0	1	0	→ 2 operations as per Booth

Let PP_{i-1} be partial product till i-1

At i:

$$PP_i = PP_{i-1} + 2^i M$$

At i+1

$$PP_{i+1} = PP_i + 2^{i+1} M$$

$$\therefore PP_i = PP_{i-1} + 2^{i+1} M - 2^i M$$
$$= PP_{i-1} + 2^i M$$

If consider 3 bit at a time
⇒ 1 operation.

apsara

Date:

For 01:

$PP_{i+1} = PP_{i-1} - 2^i M$

Multiplication gets reduced to shift and add/subtract.

These should be fastest

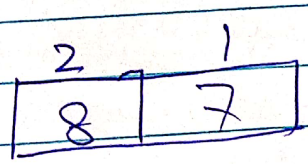
Fastest shift realised by Booth's algo.

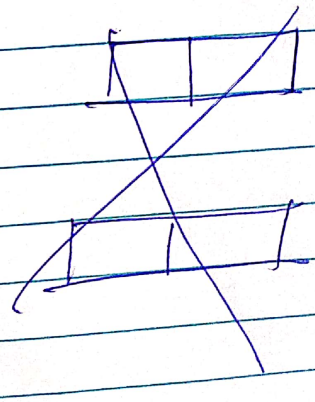
Fastest Addition, Subtraction → CLA, CSA

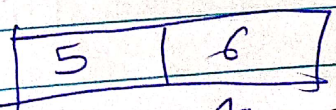
But in Booth → Alternate 0's & 1's
× Fast shift

CLA → × ↑ capacity ∴ × Fast add

If have n bit multiplier, multiplicant then divide each into k parts of m bits



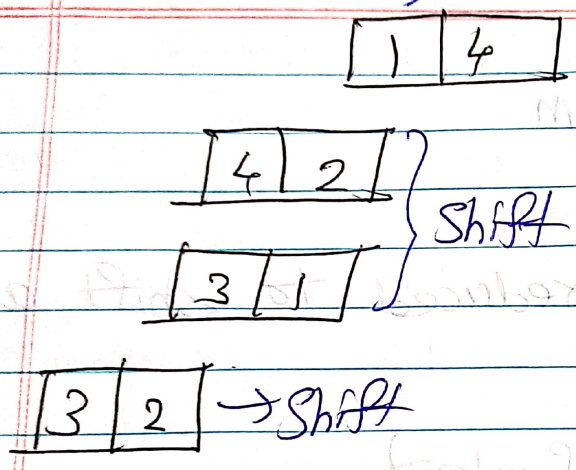




5 2 2
4 3 5 ×

} 2 partial products

Partial pdt. of blocks 1, 4



But generating prev. 2 partial pdts take time

Instead generate 4 partial pdts which is cheaper ∵ ↓ bits

~~Division~~