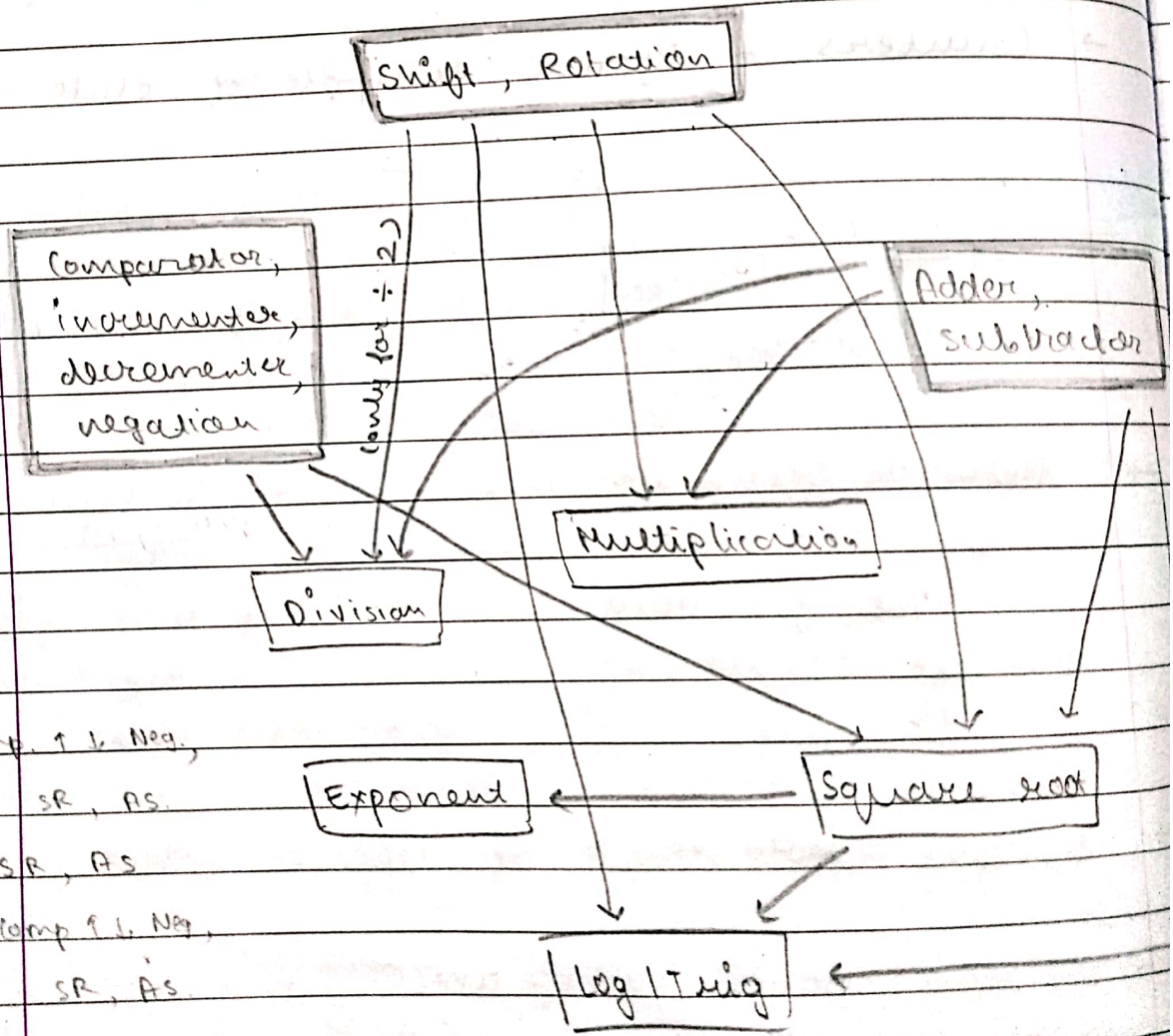
PHASE 2

Page No.:
Date:
YOUVA

Arithmetic circuits

→ Does any kind of arithmetic operations.
→ They are heart of every digital circuits.
→ Used in optimisers to improve performance.
→ Most Microcontrollers have ALU — multiplication, addition, ALU, shifter, incrementer, decrementer etc.

Types of arithmetic functions — shift, rotation, increment, decrement, compare, negation, add, subtract, *, ÷, √, Exp, log, trig max complexity. min comp



Note

Div - Comp, ↑ L, Neg, SR, AS

Mul - SR, AS

Sqrt - Comp, ↑ L, Neg, SR, AS

Exp - Sqrt

log/Trig - Sqrt, AS

→ Most operators are using the basic adder circuit, then optimising them is a great idea.

1 1 0 1 1 1 0 1 1 → 7 ones.

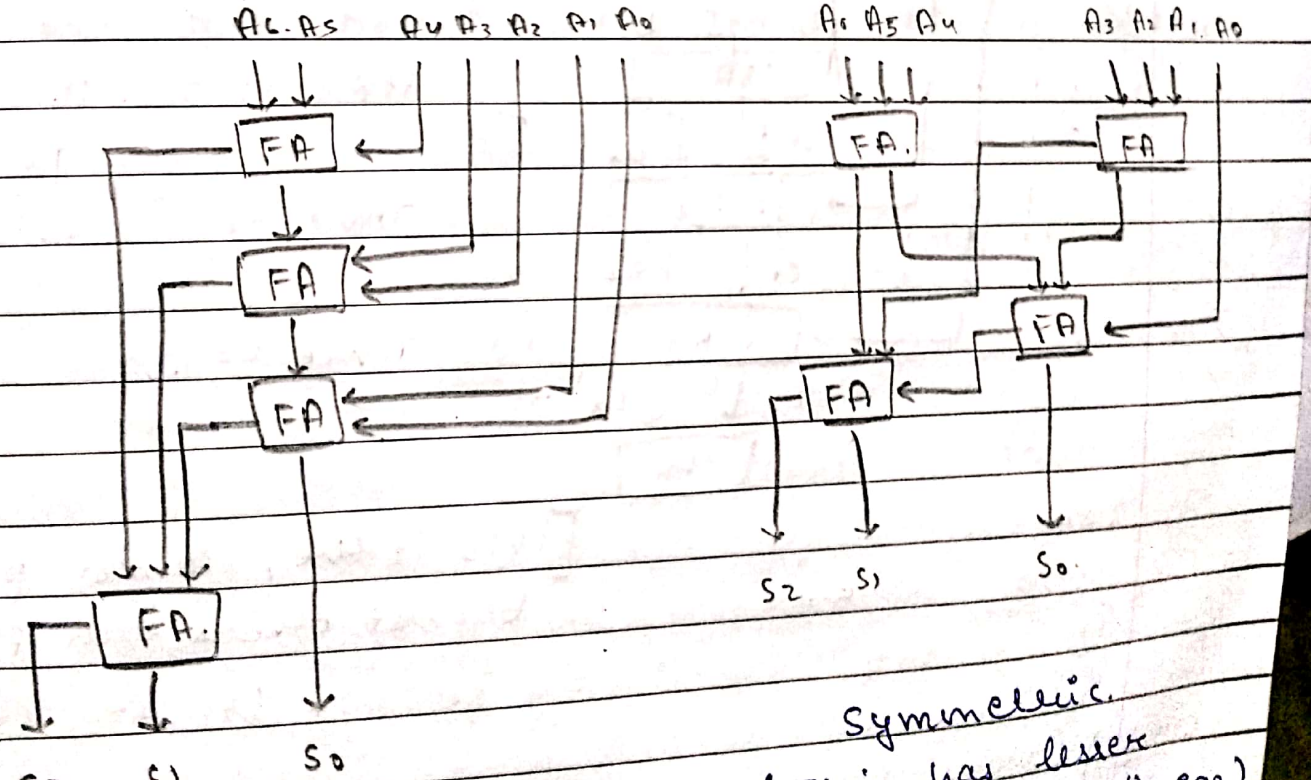
→ We found this 7 by adding, which is essentially counting.

→ Therefore, adders are counters.

→ Half adder is a (2, 2) counter. It can add (2) one bit numbers and result in a 2 bit number.

→ Full adder is a (3, 2) counter. It is also called compressor.

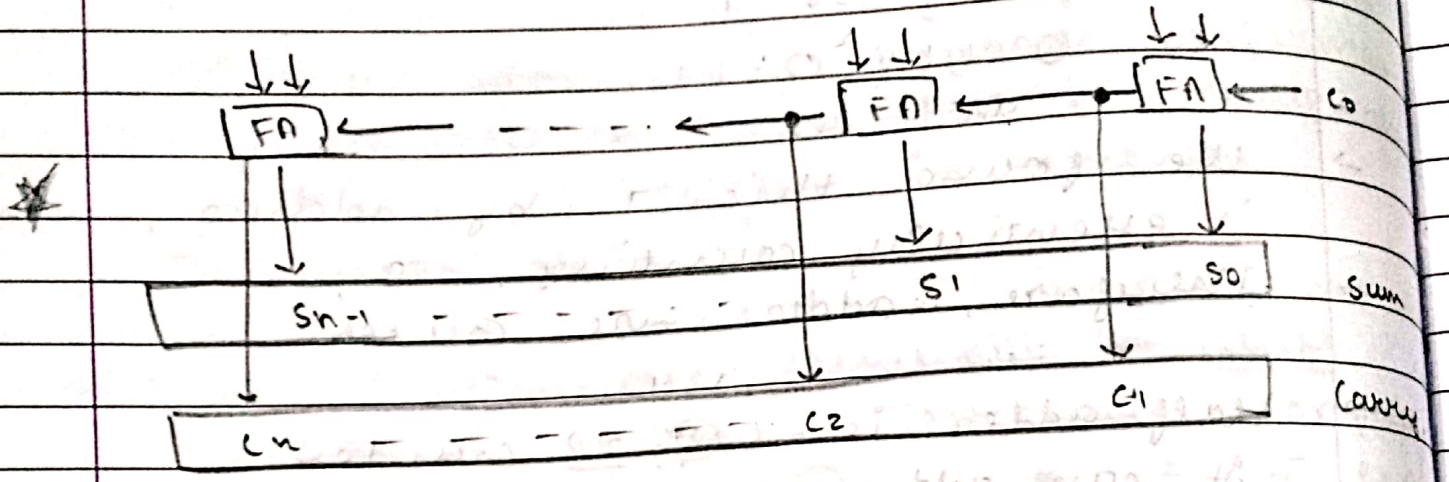
→ Adding multiple one bit numbers.



Linear

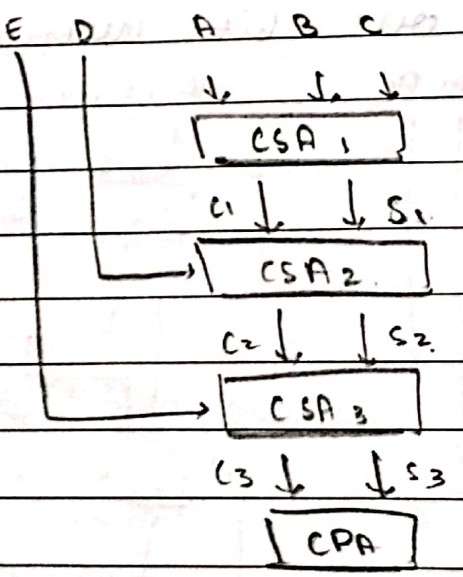
Symmetric
(This has lesser delay than linear)

→ Instead of carry to be propogated, we can use carry save adders to reduce multiple inputs to 2 and then single CPA is used



Here we compressed sum and carry stage.

$A + B + C = \text{sum} + \text{carry}$.



How delay reduced?

[This is the slowest path, because others sum parallely in max (t1, t2, t3)]

Read! [Carry Save adder
Carry skip. adder.

Page No.:
Date:
YOUVA

Carry save Adder

Eg.

x	1357
y	3564
z	2934
Ans	7855

sum without carry ie pseudo sum (sum):

	1357
	3564
	2934
	6745

carry:

	1357
	3564
	2934
	01110

Ans = x + y + z = sum + carry

So now, here carry and sum can be computed independently and at last stage both are added using carry ripple adder.

Eg.

x	10011
y	11001
z	01011
Ans	110111

sum

	10011
	11001
	01011
	00001

carry

	10011
	11001
	01011
	110110

Ans

	00001
	110110
	110111

→ We need 3 operands for the hardware

A	0 1 1 1	Three operands for first carry save stage
B	1 0 1 0	
C	1 1 1 1	

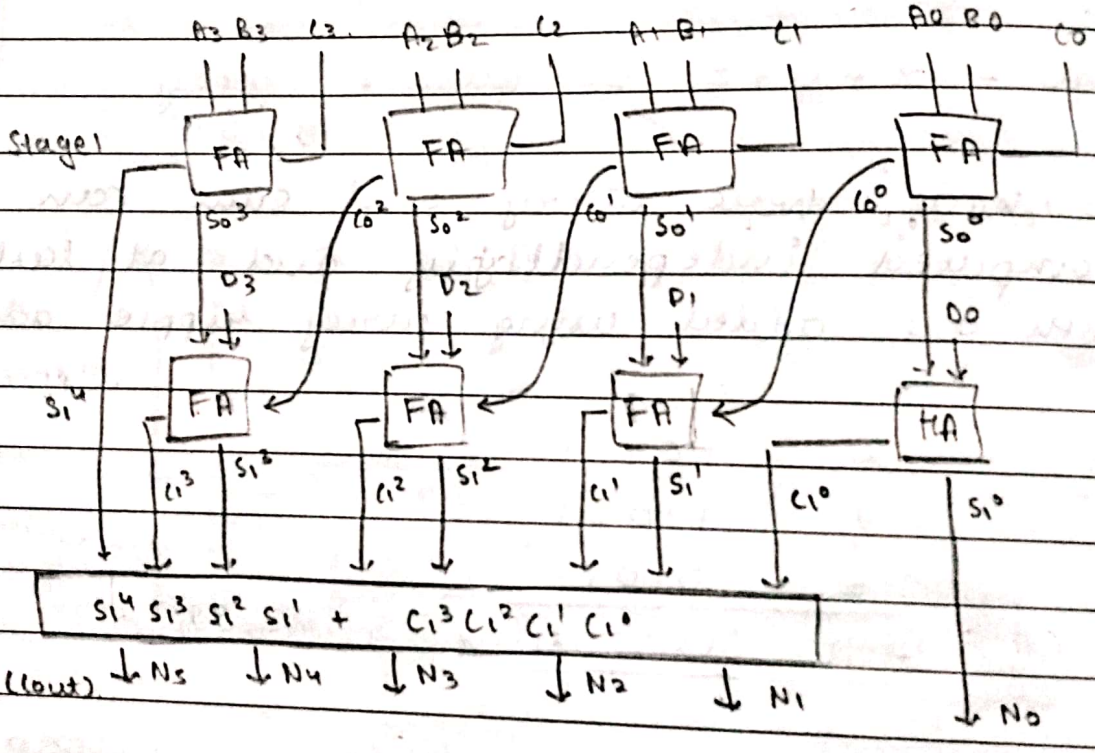
Sum	0 0 1 0	S_0
Carry	1 1 1 1 0	C_0
D	1 0 1 1	D

Three operands for second carry save stage

Sum 1 0 1 1 1 Pseudo sum S_1

Carry 0 1 0 1 0 0 Final carry out C_1

Ans. 1 0 1 0 1 1 N



→ For m integer operands - involves (m-2) CSA additions.

① In RCA: (m-1) t_{RCA} delay

② In CSA: (m-2) t_{CSA} + t_{RCA} delay. (m>3)

→ For multiplication of $A_3 A_2 A_1 A_0$ and $B_3 B_2 B_1 B_0$ we addition of partial products.

Page No.:
Date:

A_3 A_2 A_1 A_0

B_3 B_2 B_1 B_0

2FA FA HA

FA P_{03} P_{02} P_{01} P_{00}

HA P_{13} P_{12} P_{11} P_{10}

HA P_{23} P_{22} P_{21} P_{20}

P_{33} P_{32} P_{31} P_{30}

Partial products

P_7 P_6 P_5 P_4 P_3 P_2 P_1 P_0

Thus multiplication of 2 4 bit numbers gives at max 8 bit number.

$P_0 = P_{00}$

$P_1 = P_{01} \oplus P_{10}$

$P_2 = P_{02} \oplus P_{11} \oplus P_{20} \oplus C_{12}$

$P_3 = P_{03} \oplus P_{12} \oplus P_{21} \oplus P_{30} \oplus C_{23} \oplus C_{23}'$

$P_4 = P_{13} \oplus P_{22} \oplus P_{31} \oplus C_{34} \oplus C_{34}' \oplus C_{34}''$

$P_5 = P_{23} \oplus P_{32} \oplus C_{45} \oplus C_{45}' \oplus C_{45}''$

$P_6 = P_{33} \oplus C_{56} \oplus C_{56}'$

$P_7 = C_{67}$

$P_{00} = A_0 B_0$

$P_{10} = A_0 B_1$

$P_{01} = A_1 B_0$

$P_{20} = A_0 B_2$

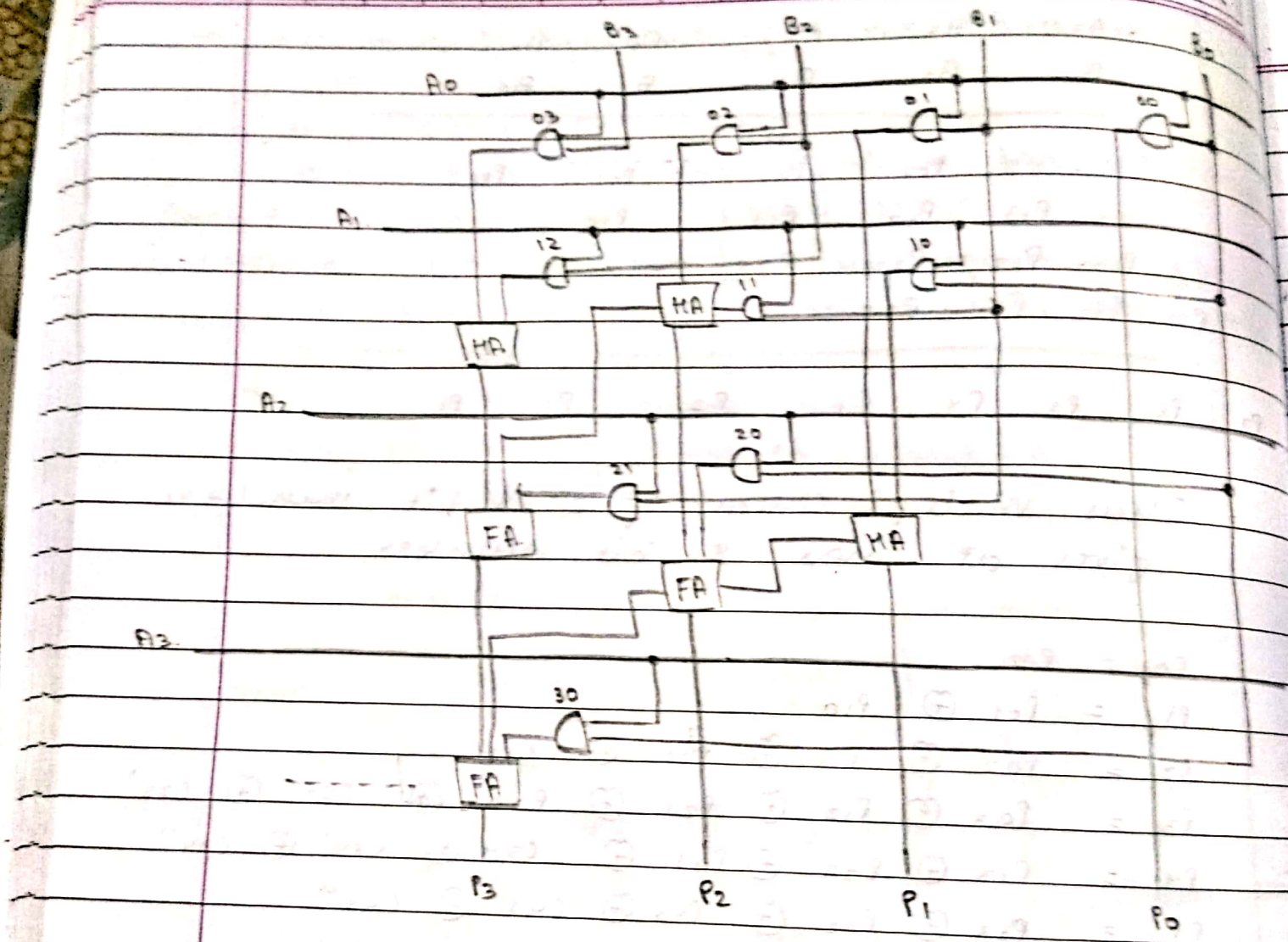
$P_{11} = A_1 B_1$

!

$P_{33} = A_3 B_3$

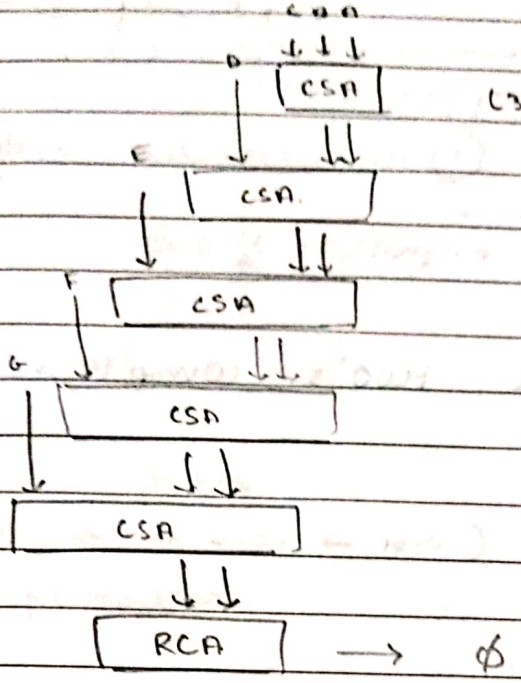
Initial stages are realised used AND gates. Then subsequently adders are also used.

Worst case delay - 7 gate delays

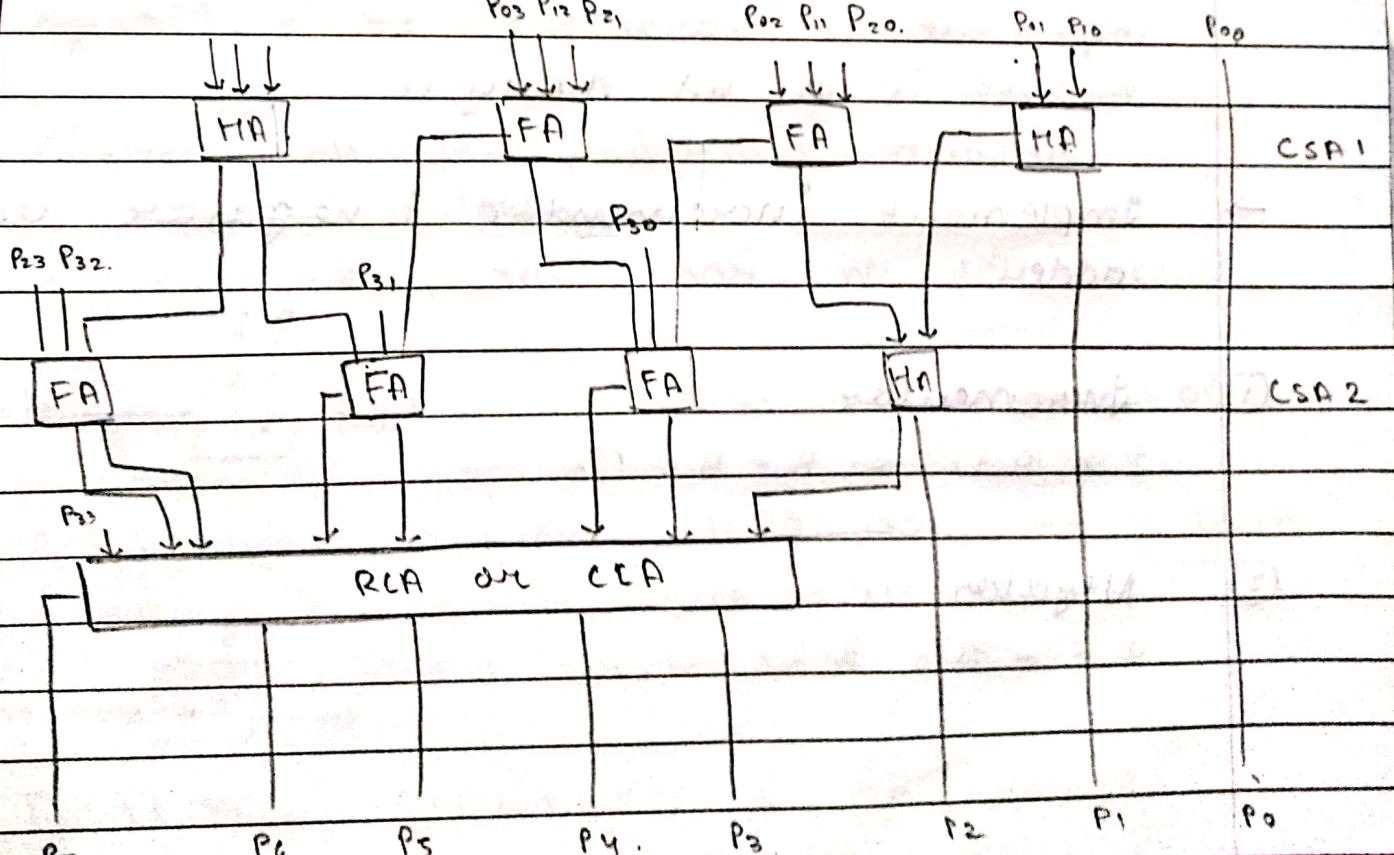


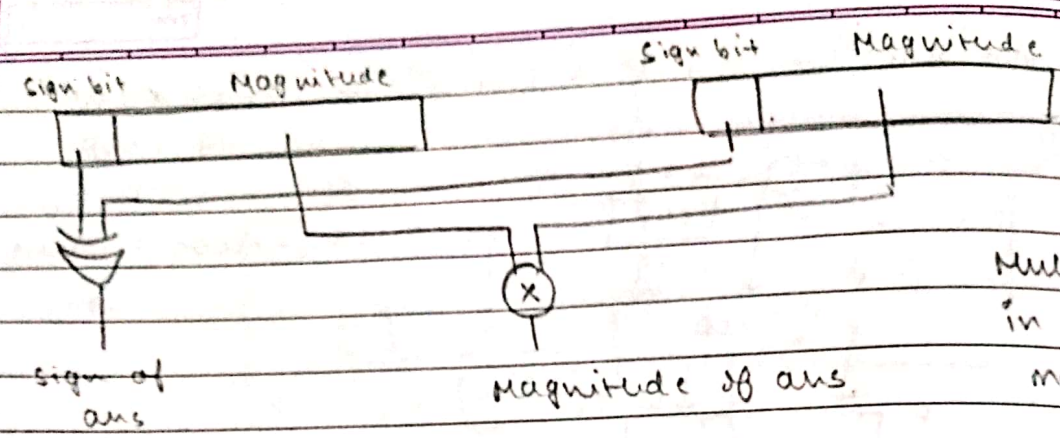
Parallel Multiplier

A + B + C + D + E + F + G → Implement with CSA



Since I can replace a FA or RCA stage to CSA stage, I can replace parallel multiplier with CSA stage.





Multiplication in signed magnitude

→ Multiplication in two's complement ...

3 0011
-5 1011 (0101 → 1010 +1 → 1011)

↓

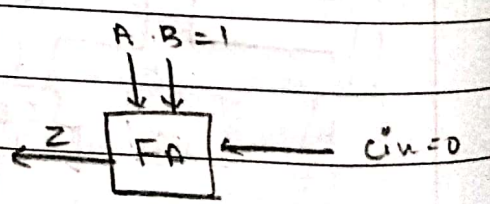
This is basically 16 - 5 = 11. Had it be a 3 bit number, we can take 8 - 5 = 3 (011)
— shortcut for 2's complement.

Difficult to multiply manually. Costly to implement in hardware. So 2's complement method is a bit messy

→ Implement incrementor / negator using adder.

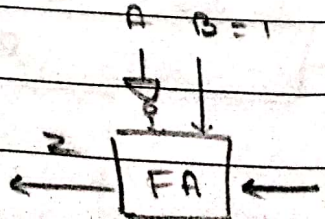
① Incrementor

$Z = A + 1 = A + B + C_{in}$



② Negator

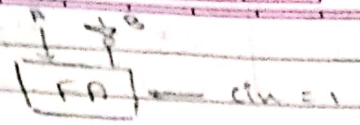
$Z = -A = \bar{A} + 1$.



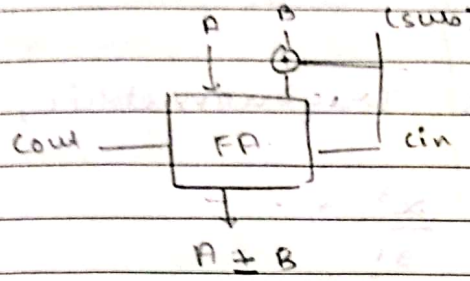
④ Subtractor

$Z = A - B$

$= A + \overline{B} + 1$



⑪ Adder subtractor.



When cin = 1, it works as subtractor

⑤ Comparator.

Do A − B :

① If cout = 1 ; A > B

② If cout = 0 ; A < B.

We check A = B using XOR separately

A = B	EQ
A != B	NEQ
A > B	GE
A < B	LT
A ≥ B.	GE . EQ
A ≤ B	$\overline{GE}$ + EQ

→ Circuits realised without adders.
- Shifters / Rotators.
- and, or, xor, nor, nand, not.

⑥ Logical shifter. 11001 >> 2 → 00110

⑦ Arithmetic shifter 11001 >> → 11110

same as logical, but fills front places with old MSB.

⑧ Rotator 11001 ROR 2 → 01110

(Rotate right)

(Bits shifted off one end are shifted into other end)

→ Shifters can be used for * or ÷

$<< 1 = \times 2^1$

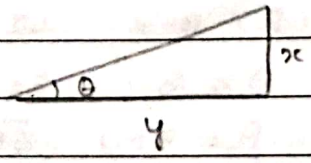
$>> 1 = \div 2^1$

⑨ Dividers, logarithms, Trigonometric, Exponential

$$e^{x} = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots$$

↓

Use adder, divider, multiplier and look up table to implement this by storing some values.



$$\tan\theta = \frac{x}{y}$$

If θ is very small, its almost like linear transformation / translation.

→ tan θ found by shifting (equivalent to 2^{-1})

Different stages...

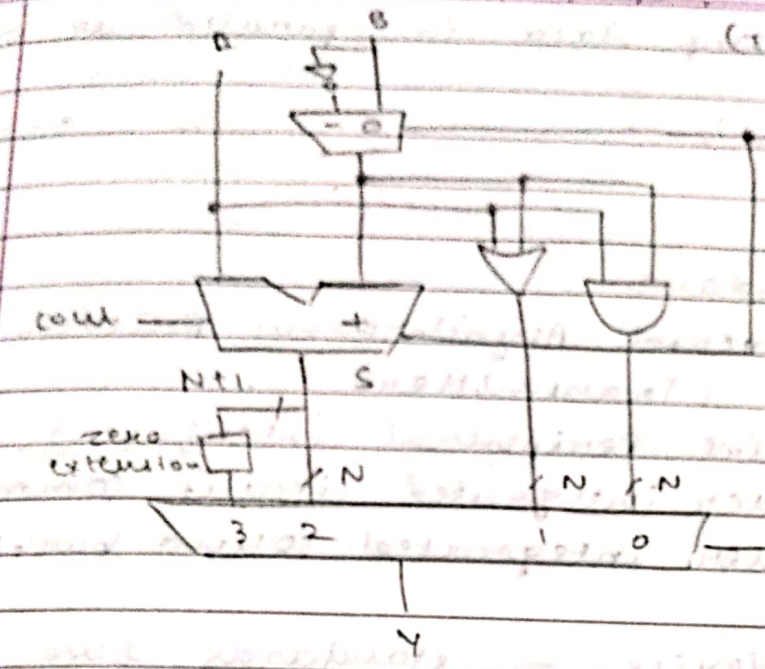
Fetch — Decode — Execute.

↓ ↓ ↓

Get the instruc^n from instruc^n memory

Extract from the instruc^n what the ALU has to do

ALU actually works.



[Actual implementation of ALU]

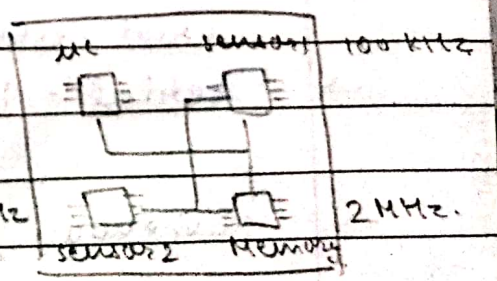
F 2:0	Function
000	A & B
001	A | B
010	A + B
011	Not Used
100	A & -B
101	A | -B
110	A - B
111	SLT (Shift left)

A - B = -7
2's complement representation has MSB = 1
∴ S_{31} or Y_{31} = 1

Y = 000 ... 001
(zero extension to make 32 bits)

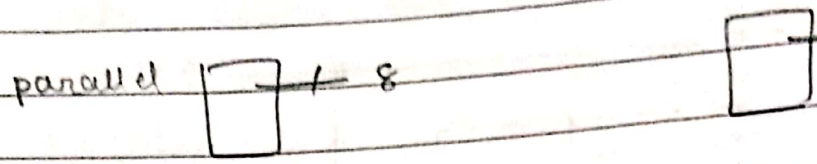
Communication Protocols / Interfaces (few)

① → On chip (chip to chip).
② → Device to device



① Chip to chip: communicate ← using I^2C.

→ We can dump data in parallel or serial

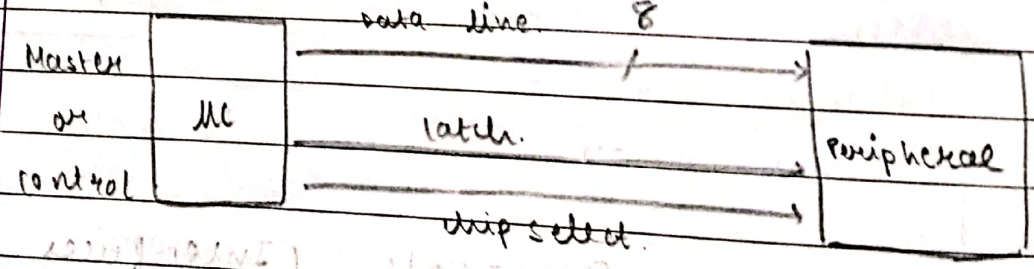


→ serial standards —
UART (Universal Asynchronous Reciever Transmitter
SPI (Serial Peripheral interfaces)
I^2C (Inter integrated circuit communication
I^2S. (Inter integrated sound burst)

(2) Device to device – standards for communication : USB, Ethernet, VGA, SCSI, HDMI, PS2, sonet etc

Communication

* Serial — Oneway, Twoway
Parallel — Oneway, Twoway



→ Parallel is fast, but expensive (pin (real estate) is expensive)
→ serial is slow, but now expensive.
→ somehow, always parallel is not preferred

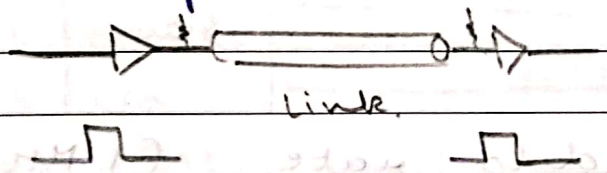
→ Serial connection – due to lesser no. of pins, the complexity of communication is high.

★ Can be both types:
① synchronous (with clock)
② Asynchronous (without a clock – but speed i.e. datarate has to be agreed beforehand)

Asynchronous serial bus standards

① RS 232 : (No clock – synchronisation apriori).

→ Used for one one communication

→ 

→ One Tx, one Rx and data-rate decided earlier. (using Hyperterminal)

→ Transmission process – 9600 baud rate.
speed of data transmission

$\frac{1}{9600}$ s = 0.104 ms

receive new info every 0.104 ms.

→ Transmitter idles high

→ It goes low for 1 bit.

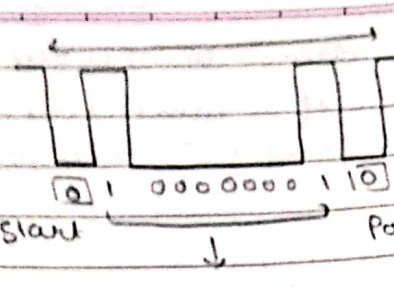
→ Send LSB first and MSB last.

→ Even parity / odd parity

→ start and stop bits for communication.

1 bit (0)

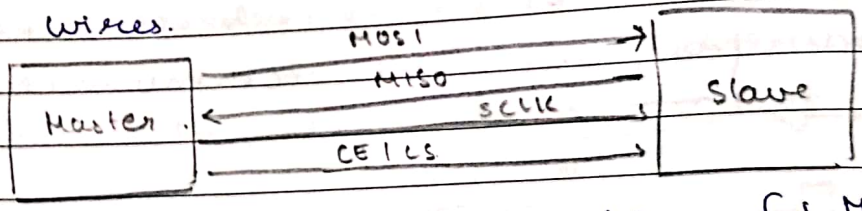
2 bits (11)



→ Key points of serial communications:
- Started with < 115.2 Kbps
- Now typically 10 - 30 Gbps.

② serial peripheral interface:

→ 4 wires.



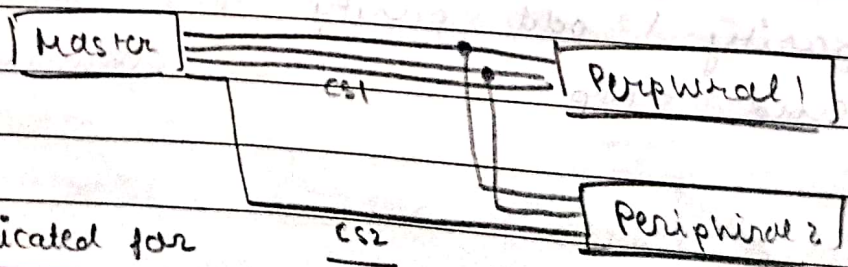
→ Supports high data rate (1 MHz ~ 70 MHz clk).

→ MOSI — Master Out Slave In
MISO — Master in Slave out
CE / CS — Chip enable / Chip select
SCLK — Synchronous clock.

↓

UART does not have this. There speed decided apriori. Here clock decides datarate.

→ MOSI, MISO, SCLK can be shared across peripherals but CE / CS is dedicated.



wires are unidirectional

dedicated for each slave.

→ Radio channels have SPI interfaces. – Need order of MHz. (can't use USB
→ All sensors have SPI interfaces.

$\overline{CS}$ = 0 enable (-ve logic)
CS = 1 enable

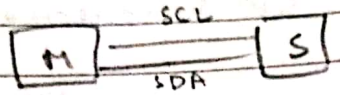
→ CS used to make a dedicated connection between one slave and master (i.e. selects one particular slave)
→ MISO – master accepts input from slave
→ MOSI – master sends data to slave

→ SPI Pins – There are typically 6 pins.
1) CE / CS
2) SCK
3) MOSI
4) MISO
5) RES (Reset).
6) D/C – Data command (seen in devices to write lots of data eg. display).
(2 optional pins – VDD and ground)

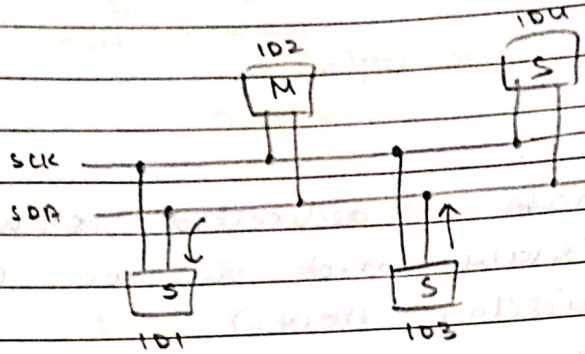
→ For only one slave device, we can drop chip select pin.
→ We can also drop RES pin and simply use soft reset
→ Drop D/C from simpler application.

Thus SCK, MOSI, MISO are only imp

③ I^2C - Inter Integrated Circuit.



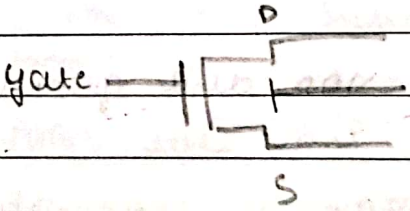
uses 7 bit addressing so can connect 27 slaves



Two blocks cannot write on bus at same time. Suppose M want to write to S, it takes control of data bus. At that point, no other block can write. After done, the bus again collected in high impedence state and other block can now use it

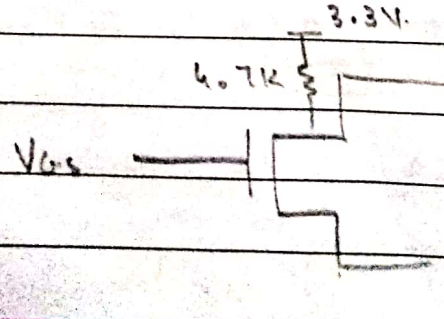
→ Select IO. by pulling external pins Hz or low

→ I^2C uses open drain (both master and slave are either low or Hz)



high impedance.

When no input at gate, D is floating. Hence is Hz state. So solve this add a pull up resistor.



→ Master is in charge of SCK frequency (typically 100 KHz or 400 KHz)
→ Master generates the address of the slave
→ It recieves an ack or nack (if it gets a nack it means the slave address is invalid, so regenerate)
→ It sends a RD / WR depending on whether it has to read or write.
→ It will send / recieve data.

① Sequence is as following when master is to write to a slave.

Start → write (WR = 1.) → deviceaddress → register you want to write to → send data → stop

② Sequence is as following when master is to read from slave

Start → read. (write = 1) → send device address → send the message you want to read from → restart communication → read = 1 → send device address → read bits → after every byte ack to the slave → continue till nack

→ Difference in SPI and I^2C: Clock speed - SPI has 100 MHz while I^2C has 100 KHz. (But problem with SPI — it is dedicated. Once it is connected to sensor (slave), it cannot be shared)
→ Advantage of SPI — bidirectional. We can configure master into slave at same time.

* SPI vs I²C video !

→ SPI – single source single destination.
→ I²C – single source multiple destinations

↓

fanout – 127 devices (limitation)
(Now 1023 slaves are supported)

→ Data Collision – when 2 devices try to use the same bus, data collides. Here, one device will have to back up
→ For repetitive transfer (same source and destination), we can remove source, destination, checksum bytes, but we always have to send preamble.

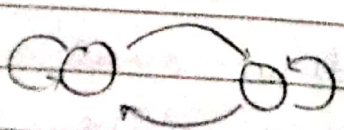
→ Optical Tx Rx standards – beam angle = 30° (half angle = 15°)

→ Isochronous data transfer – happening at same time.

State Machines

Controls a digital system that carries a step by step procedure (algorithm).

State Diagrams: Transitions are called arcs



(state graph)

Typical flow chart used that are called Algorithmic State Machines (ASM)

→ Mainly used to design control units.

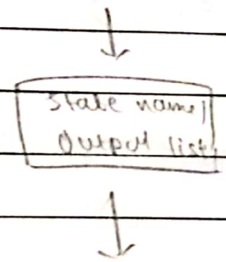
→ A given state machine chart can be converted to several form. It gives rise to several different digital (network) implementation

→ SM (state machine) — certain rules are to be followed.

→ When these rules are followed, SM chart is equal to a state graph. (leads directly to network implementation.)

→ 3 parts (the 3 SM block) that make a SM chart:

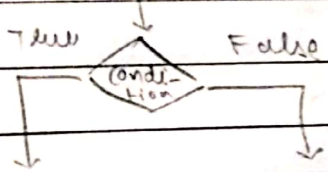
① state box.



Assign optional state code

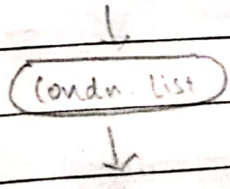
Optional output list

② Decision box



Boolean Exp.

③ conditional list



contains conditional output list.

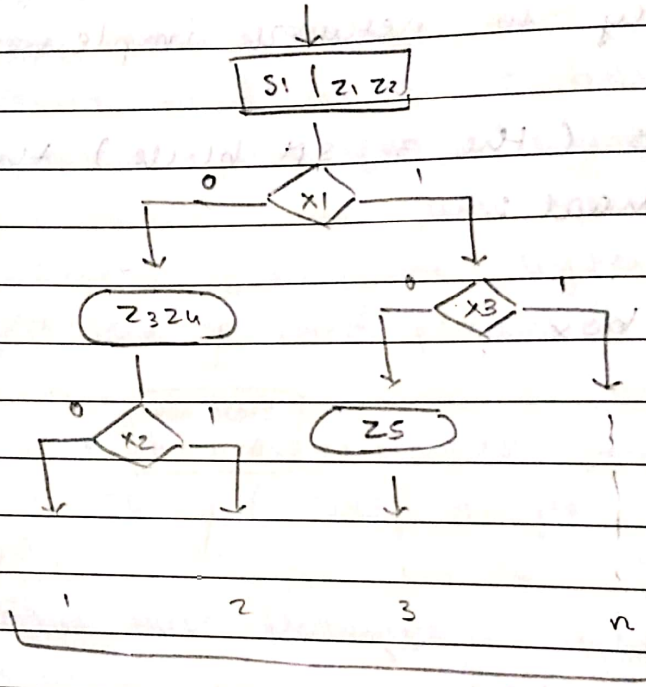
Conditional output depends both on state of system and inputs.

→ Thus, SM chart has 3 SM blocks (an input and an output — it describes that the machine is in one state)

① One state box
② Few decision blocks
③ Conditional output boxes (associated state)

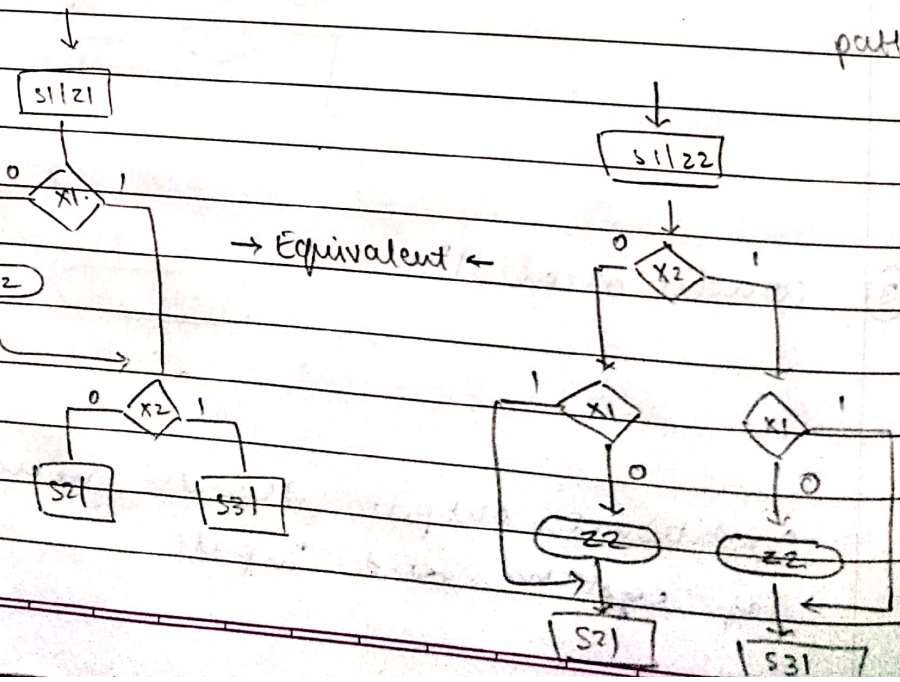
→ A path through an SM block from entrance to exit is known as link path

Eg:

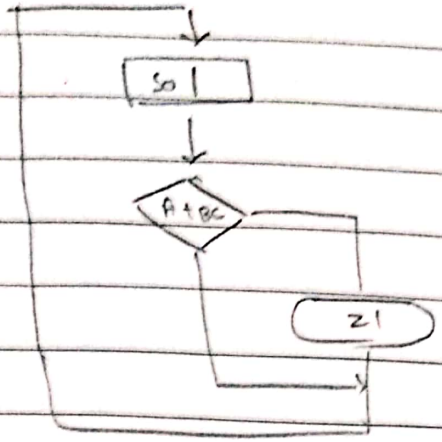


There can be n exit paths.

Eg2



Eg 3

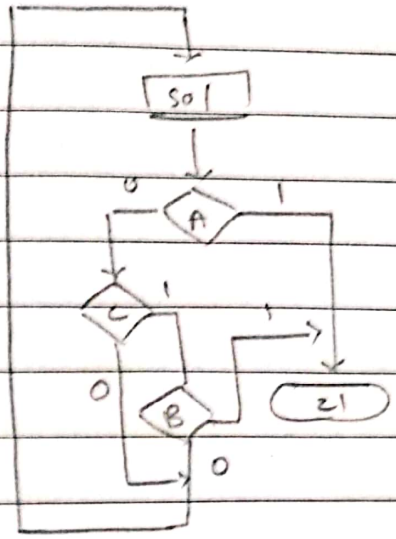


Z1 = 1 if A + BC = 1

Z1 = 0 if A + BC = 0

Z1 = A + BC

Eg 4



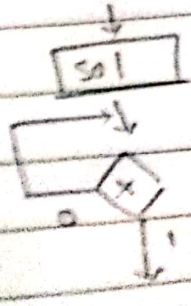
Z1 = 1 if A = 1

if A = 0, C = 1, B = 1

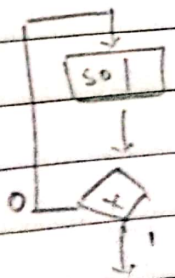
So Z1 = A + A'BC

Rules

1) For valid combinations of input variables, there must be exactly one path. This is because a single input can lead to a single next state
2) Internal feedback is not allowed.



Not allowed



Allowed