Chapter 1 - logic Gates

→ inverts or complements its input

NOT

$B = \bar{A}$

A	B
0	1
1	0

the Buffer gate have the ability to deliver large amounts of current

→ it's copy the input

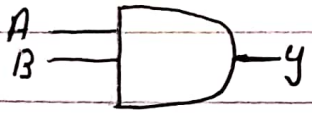
BUF

$B = A$

A	B
0	0
1	1

→ the logic gate that produces a high output only when all of the input are high

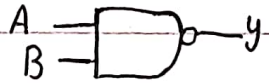
AND



A	B	y
0	0	0
0	1	0
1	0	0
1	1	1

produces a low output when all the input are high

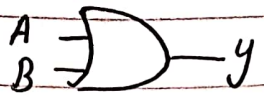
NAND



A	B	y
0	0	1
1	0	1
0	1	1
1	1	0

- A logic gate → output is low when or more of the input is high (low)

OR +



$y = A + B$

A	B	y
0	0	0
0	1	1
1	0	1
1	1	1

the opposite function of OR

NOR

$y = \overline{A+B}$

A	B	y
0	0	1
1	0	0
0	1	0
1	1	0

- A logic gate that produces a high output only when its two input are at opposite lev

XOR

*odd parity

$y = A \oplus B$

$y = \bar{A}B + A\bar{B}$

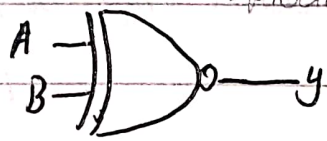
A	B	y
0	0	0
0	1	1 ✓
1	0	1 ✓
1	1	0

produce high output when inputs are the same level

XNOR

*Even parity

→ produce high output two input y is the same



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

A	B	y
0	0	1
0	1	0
1	0	0
1	1	1

Minterm → product

Maxterm → sumtion

- **Minterm**: product that include all input variables.
  $A\bar{B}\bar{C}$, $\bar{A}\bar{B}\bar{C}$, $AB\bar{C}$

- **Maxterm** sum that include all input variables
  $(A+B+\bar{C})$, $(\bar{A}+\bar{B}+\bar{C})$, $(\bar{A}+B+C)$

- **Sum of Product (Sop)** → 1
  ORing of ANDing
  Summation of minterms

- **Product of Sums (Pos)**
  ANDing of ORed terms
  Product of Maxterms

$$X=(A+B+C)(A+B+\bar{C})(A+\bar{B}+C)(\bar{A}+B+\bar{C})$$

(Pos) → 0

Find the (Sop) →

	A	B	C	X
output (1)	0	0	0	0 → $A+B+C$
	0	0	1	0 → $A+B+\bar{C}$
	0	1	0	0 → $A+\bar{B}+C$
	0	1	1	1 → $\bar{A}BC$
	1	0	0	1 → $A\bar{B}\bar{C}$
	1	0	1	0 → $\bar{A}+B+\bar{C}$
	1	1	0	1 → $AB\bar{C}$
	1	1	1	1 → ABC

$$X=(\bar{A}BC)+(A\bar{B}\bar{C})+(AB\bar{C})+(ABC)$$

(Sop) → 1

Basic Rule Boolean Algebra:

1. $A+0 = A$
2. $A+1 = 1$
3. $A \cdot 0 = 0$
4. $A \cdot 1 = A$
5. $A + A = A$
6. $A + \bar{A} = 1$
7. $A \cdot A = A$
8. $A \cdot \bar{A} = 0$
9. $\bar{\bar{A}} = A$
10. $A + AB = A$
11. $A + \bar{A}B = A + B$
12. $(A+B)(A+C) = A + BC$

Ex. $y = AB + \bar{A}B$

$= B(A + \bar{A})$

$= B(1)$

$= B$

Ex. $y = A(AB + ABC)$

$= A(AB(A + AC)$

$= A(B(A))$

$= (AA)B$

$= AB$

$A(AB(1+C))$

$A(AB(1))$

$AA.B$

AB

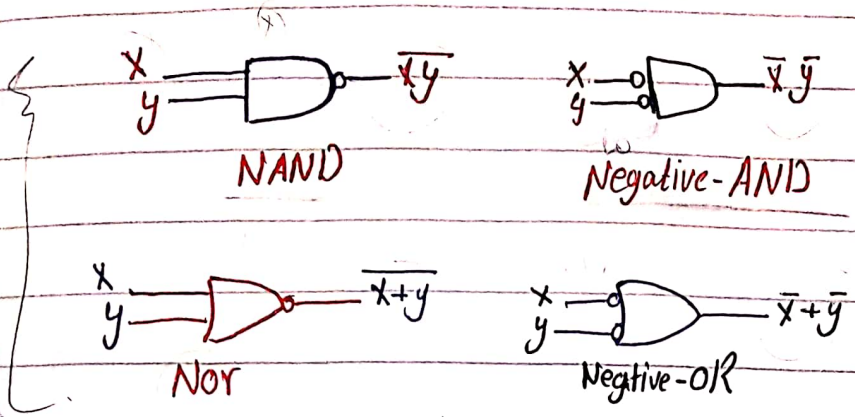
Find Some of product (Sop) → 1

A	B	C	y
0	0	0	1 → $\bar{A}\bar{B}\bar{C}$
0	0	1	0 → $\bar{A}B\bar{C}$
0	1	0	1 → $A\bar{B}\bar{C}$
0	1	1	1 → $A\bar{B}C$
1	0	0	1 → $AB\bar{C}$
1	0	1	0 → $\bar{A}\bar{B}C$
1	1	0	1 → ABC
1	1	1	

Sop = $(\bar{A}\bar{B}\bar{C}) + (\bar{A}B\bar{C}) + (A\bar{B}\bar{C}) + (A\bar{B}C) + (ABC)$ ✓

pos = $(A+B+\bar{C})(A\bar{B}C)(\bar{A}+\bar{B}+C)$ ✓

→ Boolean Algebra :



example 1

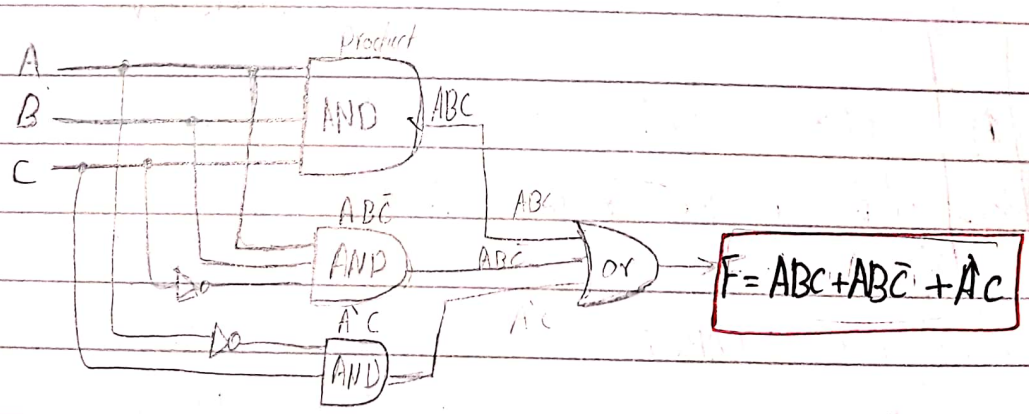
$$F = AB' + C'D + AB' + C'D$$

$$= \boxed{AB' + C'D}$$

$$F = \overline{(\bar{A}+B)} + \bar{C} = (\bar{A}+\bar{B})\cdot\bar{C} = (A+B)C = \boxed{AC + BC}$$

$$F = \overline{(\bar{A}+B)} + CD = \boxed{(A\cdot\bar{B}) * \overline{C+D}}$$

(1)

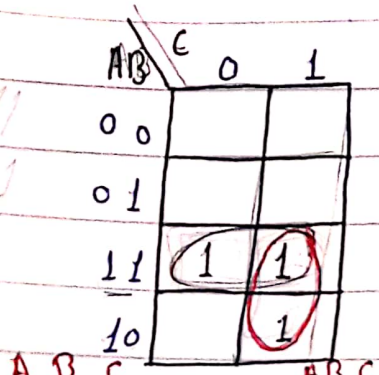


$AB + A\bar{B}C + ABC$

K-map :

→ Use K-map to minimize the following :

a) $AB + A\bar{B}C + ABC$

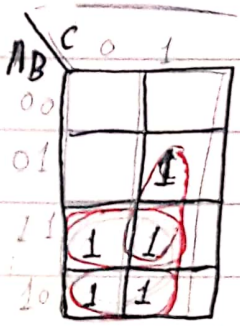


A B C
1 1 0
1 1 1

A B C
1 1 1
1 0 1

$AB + AC$

b) $A + BC$



1 1 0
1 1 1

0 1 1
1 1 1

1 0 0
1 0 1

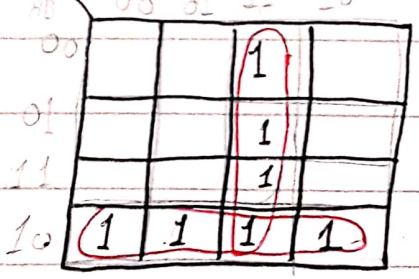
$AB + BC$ ✗ $A\bar{B}$ → $B(A + C + A\bar{B})$

1 1 0
1 1 1
1 0 0
1 0 1

$A + BC$ ✓

$A + BC$

c) $A\bar{B} + A\bar{B}C\bar{D} + C\bar{D} + ABC\bar{D}$



0 0 1 1
0 1 1 1
1 1 1 1
1 0 1 1
$C\bar{D}$ +

1 0 0 0
1 0 0 1
1 0 1 1
$A\bar{B}$

$F = A\bar{B} + C\bar{D}$ ✓

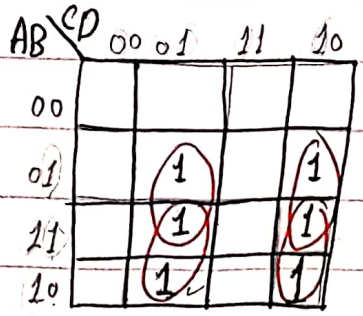
$B\bar{C}D + A\bar{C}D + AC\bar{D} + BC\bar{D}$ ✓

d) $A\bar{B}\bar{C}\bar{D} + AC\bar{D} + B\bar{C}D + \bar{A}BC\bar{D}$

1001 110 101 0110

A B C D
0101
1101
$B\bar{C}D$

1101
1001
$A\ \bar{C}D$ ✓



A B C D
0101
1101
$C\bar{D}$

A B C D
1101
1001
$A\ \bar{C}\ \bar{D}$

A B C D
0110
1110
$BC\bar{D}$

$A\bar{C}D + B\bar{C}D + BC\bar{D}$ ✓

1110
1010
$A\ C\bar{D}$

0110
1110
$BC\bar{D}$

Multiplexers & decoders:

Types of Logic Circuits

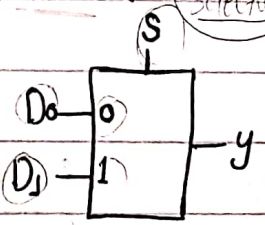
- Combinational Logic:
  - memoryless → decoders, Encoders, Multiplexer, Adder
  - ↳ without memory
- Sequential logic:
  - ↳ has memory

2:1 Multiplexer → select one data output from 2 data (selector)

many inputs and one output

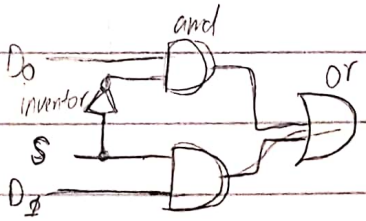
S	y
0	D_0
1	D_1

$\bar{S}D_0 + SD_1$



$y = \bar{S}D_0 + SD_1$

multiplexer equation

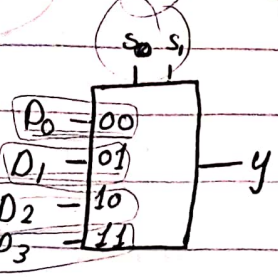


4:1 Multiplexer

- Selectes one data output from 4 data inputs

S_1

S_1	S_0	y
0	0	D_0
0	1	D_1
1	0	D_2
1	1	D_3



number of input

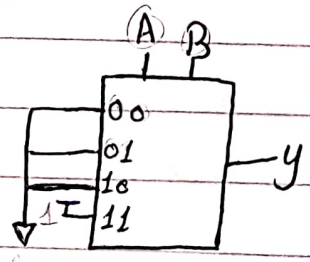
to find number of selector $\log_2 N$

$$\bar{S}_1 \bar{S}_0 D_0 + S_1 S_0 D_1 + S_1 \bar{S}_0 D_2 + S_1 S_2 D_3$$

- AND2 Using 4:1 Mux

↳ 2input AND function using 4:1 MUX

A	B	y
0	1	0
1	0	0
1	1	1
0	0	0

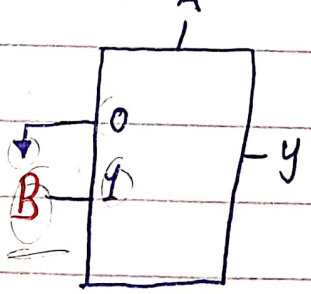


$y = AB$

- implement 2-iinput AND function using 2:1 Mux

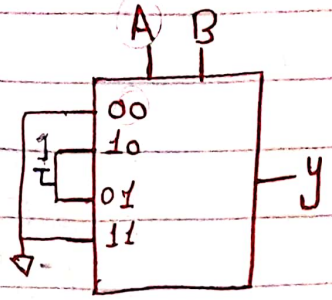
$A = 0 \rightarrow y = 0$

$A = 1 \rightarrow y = B$



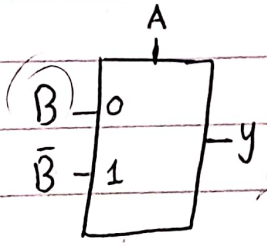
- implement 2 input XOR function using 4:1 MUX

A	B	y
0	0	0
0	1	1
1	0	1
1	1	0



$A \oplus B = \bar{A}B + A\bar{B}$

→ by using 2:1 MUX

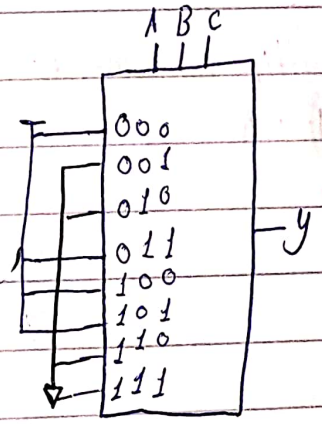


A = 0 y = B

A = 1 y = $\bar{B}$

- Implement the following function using 8:1 MUX

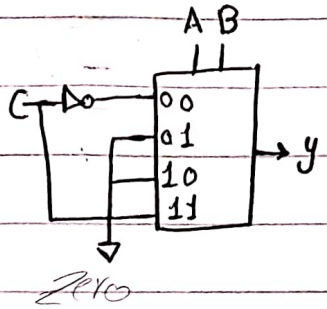
A	B	C	y
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0



a) 4:1 multiplexer and one inverter

$$y = \bar{A}\bar{B}\bar{C} + ABC$$

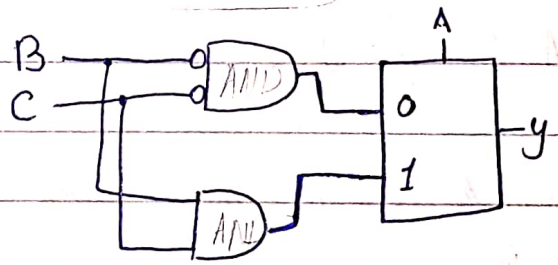
A	B	
0	0	$\bar{C}$
1	0	0
0	1	0
1	1	C



(c) a 2:1 Multiplexer and two other logic gates

$$y = \bar{A}\bar{B}\bar{C} + ABC$$

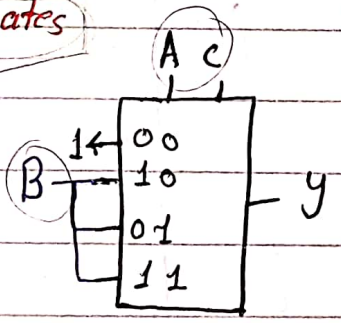
A	
0	$\bar{B}\bar{C}$
1	BC



(b) a 4:1 multipluxer and no other gates

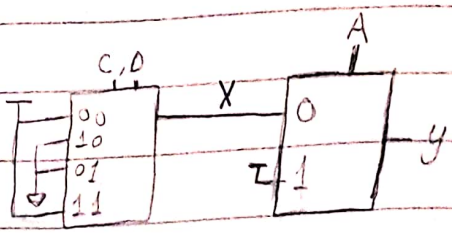
$$y = B + \bar{A}\bar{C}$$

A	B	y
0	0	1
1	0	B
0	1	B
1	1	B



Example : Write A minimized Boolean equation for the function.

C	D	X
0	0	1
1	0	0
0	1	0
1	1	1



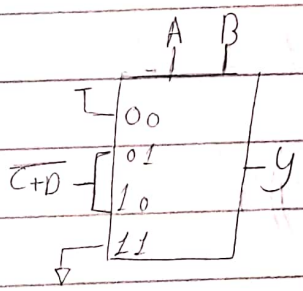
$X = \bar{C}\bar{D} + CD = C \odot D = \overline{C \oplus D}$ ~ XNOR ✓

A	y
0	$\bar{C}\bar{D} + CD$
1	1

$y = \bar{A}(\bar{C}\bar{D} + CD) + A = \boxed{\bar{C}\bar{D} + CD + A}$ ✓

Example

A	B	y
0	0	1
0	1	$\overline{C+D}$
1	0	$\overline{C+D}$
1	1	0

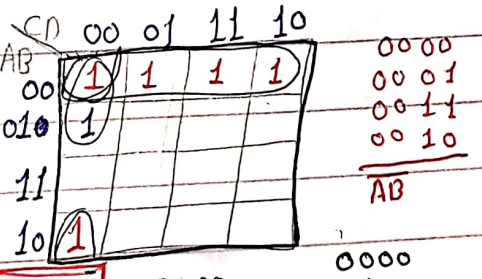


$\bar{A}\bar{B}\bar{C} + \bar{A}B\bar{D}$

$\bar{A}\bar{B} + \bar{A}B(\overline{C+D}) + A\bar{B}(\overline{C+D})$

$\bar{A}\bar{B} + \bar{A}B\bar{C}\bar{D} + A\bar{B}\bar{C}\bar{D}$

00 0100 1000

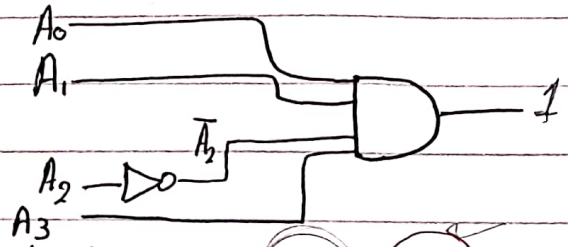


$\boxed{\bar{A}\bar{B} + \bar{A}\bar{C}\bar{D} + \bar{B}\bar{C}\bar{D}}$

0000 1000 $\bar{B}\bar{C}\bar{D}$

0000 0100 $\bar{A}\bar{C}\bar{D}$

- Decoder is a digital that detects the presence of a specified combination of bits (pattern) on its input → 1011 → 1



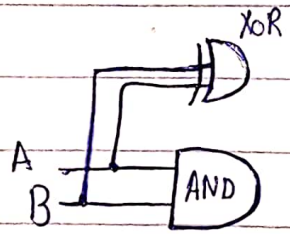
producing a high output when 1011 is on the inputs

→ Half Adder:

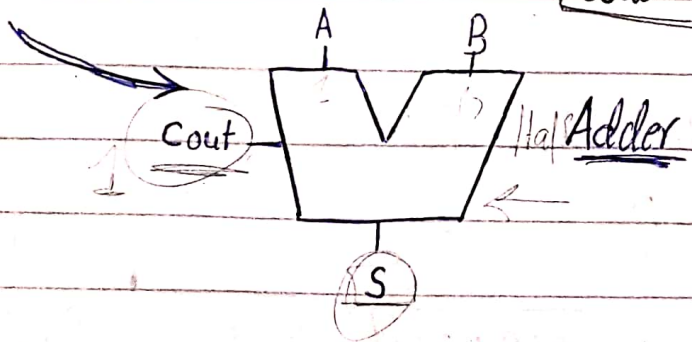
↳ accepts two binary digits on its input and produce two binary digits on its output

A B
0 + 0 = 0
0 + 1 = 1
1 + 0 = 1
1 + 1 = 10

A	B	Cout	S
0	0	0	0
1	0	0	1
0	1	0	1
1	1	1	0



XOR
$S = A \oplus B$
$Cout = AB$



Full Adder:

↳ accepts two input bits and an input carry and generates a sum output and an output carry.

Cin	A	B	Cout	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

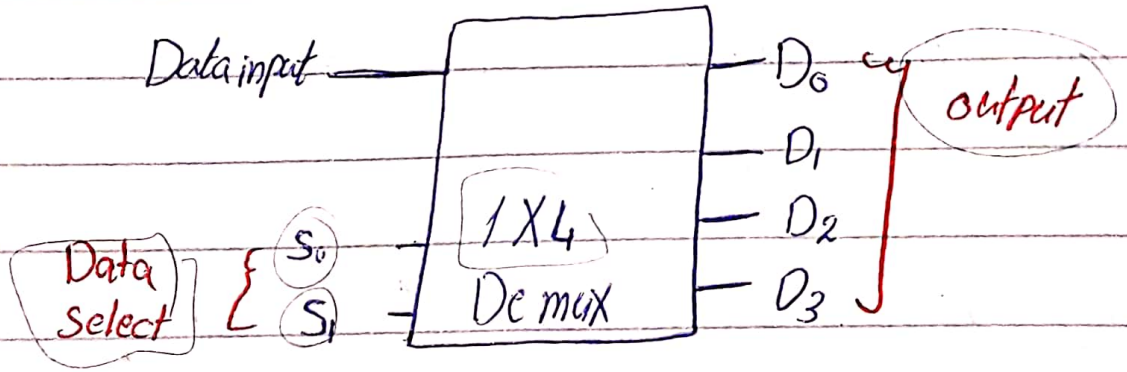
$S = A \oplus B \oplus C$

Cin B	Cin A	A B
1 1 1	1 1 0	0 1 1
1 0 1	1 1 1	1 1 1

Cin \ A B	00	01	11	10
0			1	
1		1	1	1

$AB + ACin + Cin B = AB + Cin(A+B)$

→ Demultiplexers:



(1 X 4) Demultiplexer

Data input	S1	S0	D0	D1	D2	D3
Din	0	0	1	0	0	0
Din	0	1	0	1	0	0
Din	1	0	0	0	1	0
Din	1	1	0	0	0	1

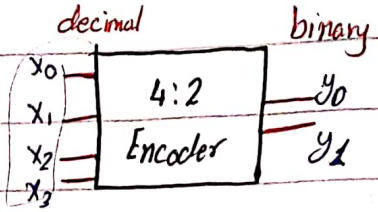
$D_0 = Din\ \bar{S_0}\ \bar{S_1}$

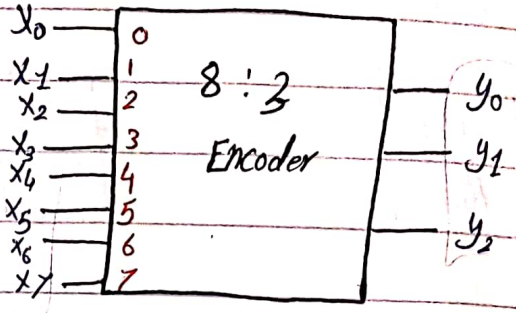
$D_1 = Din\ \bar{S_1}\ S_0$

$D_2 = Din\ S_1\ \bar{S_0}$

$D_3 = Din\ S_1\ S_0$

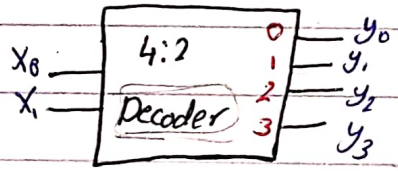
→ Encoder (decimal → binary) *يشتغل*





x_0	x_1	x_2	x_3	y_1	y_0
1	0	0	0	0	0
0	1	0	0	0	1
0	0	1	0	1	0
0	0	0	1	1	1

Decoder (binary → decimal) *يفك الشفرات*



x_0	x_1
0	0
1	1
2	0
3	1

y_0	y_1	y_2	y_3
1	0	0	0
0	1	0	0
0	0	1	0
0	0	0	1

Logic expression using SOP

$y_0 = m_0 = \bar{x}_0 \bar{x}_1$

$y_1 = m_1 = \bar{x}_0 x_1$

$y_2 = m_2 = x_0 \bar{x}_1$

$y_3 = m_3 = x_0 x_1$

En = ستين

de = فك

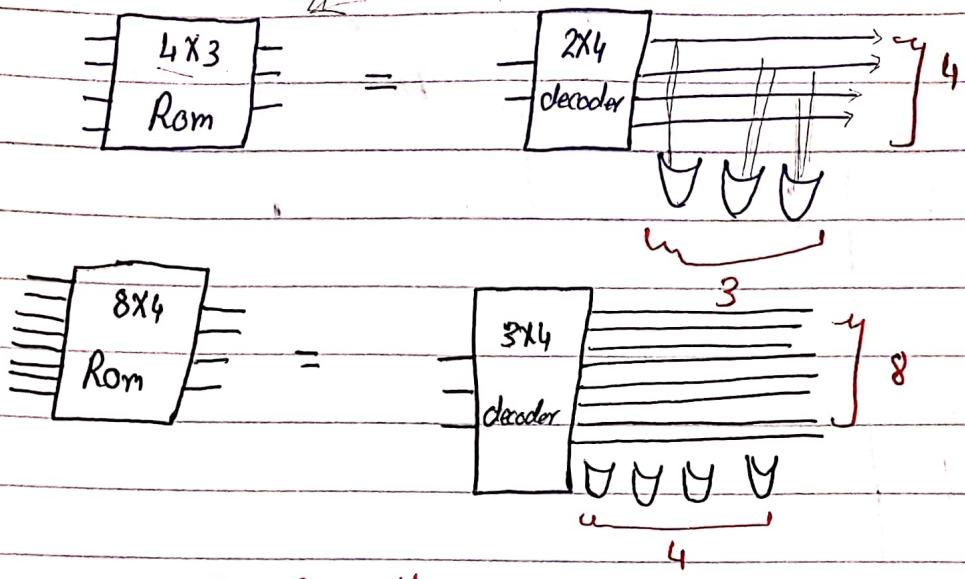
decoder + OR

Read only memory [Rom]

↳ Store binary information permently

* its a decoder Connected to ORs (OR)

b → d

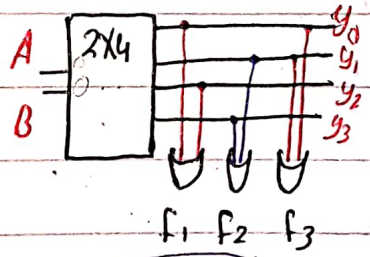


★ implement the Rom that store this data

Address	Data
0	1011
1	011
2	100
3	010

4 Addreses = 4 decoders

Data of 3 bit so use 3OR



4X3 : Rom

decoder truth tabel					ORs		
AB	y_3	y_2	y_1	y_0	f_1	f_2	f_3
0 0	0	0	0	1	1	0	1
0 1	0	0	1	0	0	1	1
1 0	0	1	0	0	1	0	0
1 1	1	0	0	0	0	1	0

types of Rom:

- Rom: Store information (function) during production
  - Mask is used in the production process
  - unalterable غير قابل للتغيير
  - low cost for large quantity production → used in final products

PRom (programmable Rom):

Store info electrically using PROM programmer at the user's site
- unalterable
- higher cost than ROM → used in the system development phase
  → can be used in small quantity system

EPROM (Erasable ROM) ← تخزين المعلومات كهربائيا

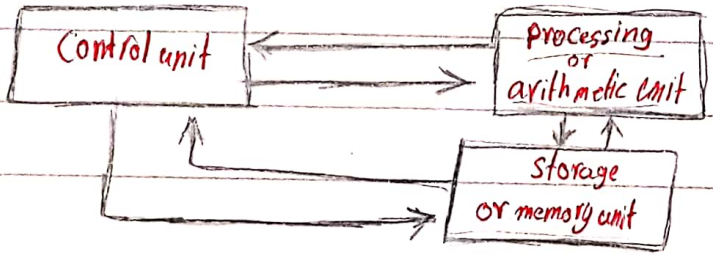
- Store info electrically using PROM programmer at the user's site
- Stored info is erasable (alterable) using UV light (electrically in some devices) and rewritable. قابل للمسح
- higher cost than PROM but reusable → used in the system development
  Not used in the system production due to erasability قابلية المسح

→ INtegrated CiRCuts :

- Classification by circuts density :
  - SSI ⇝ several (less than 10) independent gates
  - MSI ⇝ 10 to 200 gats : Decoder, adder, register, parity checker
  - LSI ⇝ 200 to few thousand gates : Digital subsystem
    processor, memory, etc
  - VLSI ⇝ Thousands of gates : Digita system, Microprocessor, memory module
- Classification by technology :
  - Transistor-transistor logic (TTL) ⇝ Bipolar transistors, NAND
  - Emitter-coupled logic (ECL) ⇝ Bipolar transistor, Nor
  - Metal-oxide semiconductor (MOS) ⇝ unipolar transistor, high density
  - (CMOS) Complementary Mos → Low power consumption

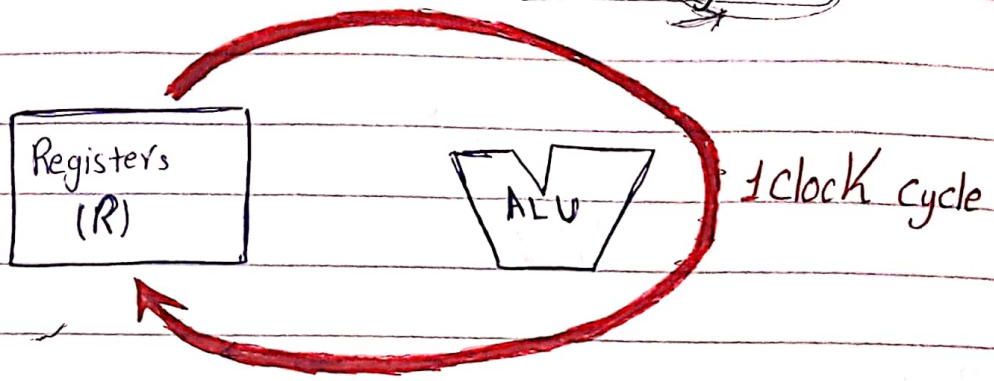
Register Transfer Language :

- A digital system is an interconnection of digital hard ware moduls that accmplish a specific information processing tasIk.
- the various modules are interconnected with common data and control paths to form a digital computer system



* Digital Computer = Registers + Microoperations Hardware + Control Functions

- Micro operation : An elementary operation performed during one clock, the information stored in one or more registers



$R \leftarrow f(R, R)$

↳ f: shift, count, clear, load, add, ...

→ Register tranfer language (RTL) :

RTL: is the symbolic notation used to describe the microoperation transfer among registers.

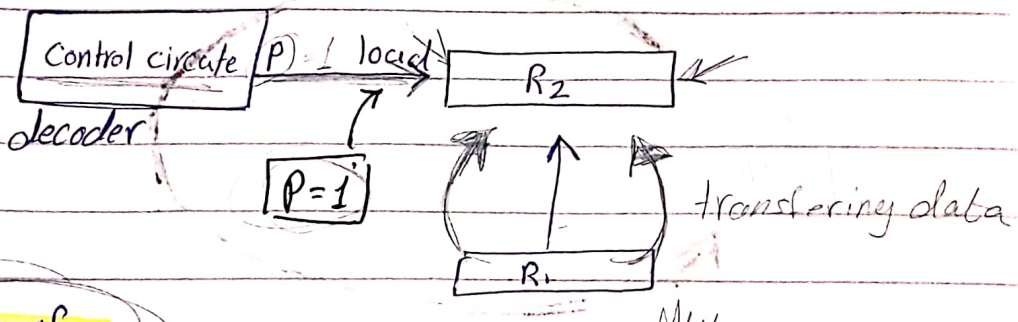
Symbol	Description	Examples
letters and numerals	denote a register	MAR, R2
Parenthese ()	a part of a register	R3(0,7), R1(L)
Arrow ←	transfer of data	$R_3 \leftarrow R_2$
Comma ,	Separates two microoperation	$R_3 \leftarrow R_2$, $R_4 \leftarrow R_1$
Colon :	Terminates control condition	$p: R_3 \leftarrow R_2$

Control Condition

-the control condition means that the transfer operation if P=1

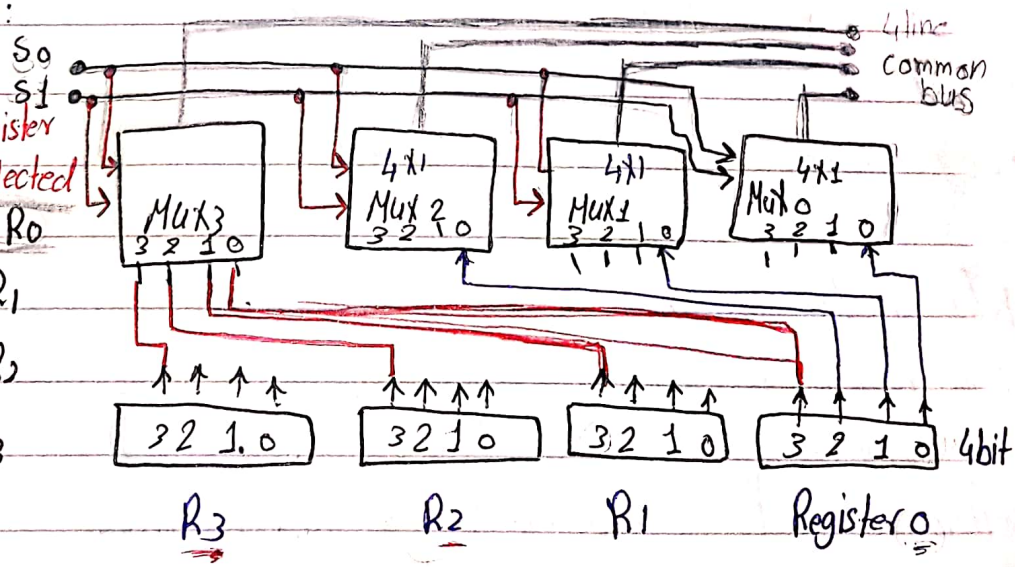
P: $R_3 \leftarrow R_1$

Transfer of n-bit data from R1 to R2 when P=1



Bus transfer

↳ Bus transfer for register R0 to R3 in 4 bit bus system is shown below:



S_1	S_0	register selected
0	0	R0
0	1	R1
1	0	R2
1	1	R3

- K registers of n bits can be multiplexed to produce a n-line common bus
  - n MUX are needed (= bit numbers)
  - the size of each MUX is Kx1 (= number of register x1)
  - w selection line are needed where, $2^w \geq K$

K = 64

Example → for register R0 to RB3 in a 16 bit bus system what is the MUX size we use? how many select bits are needed?

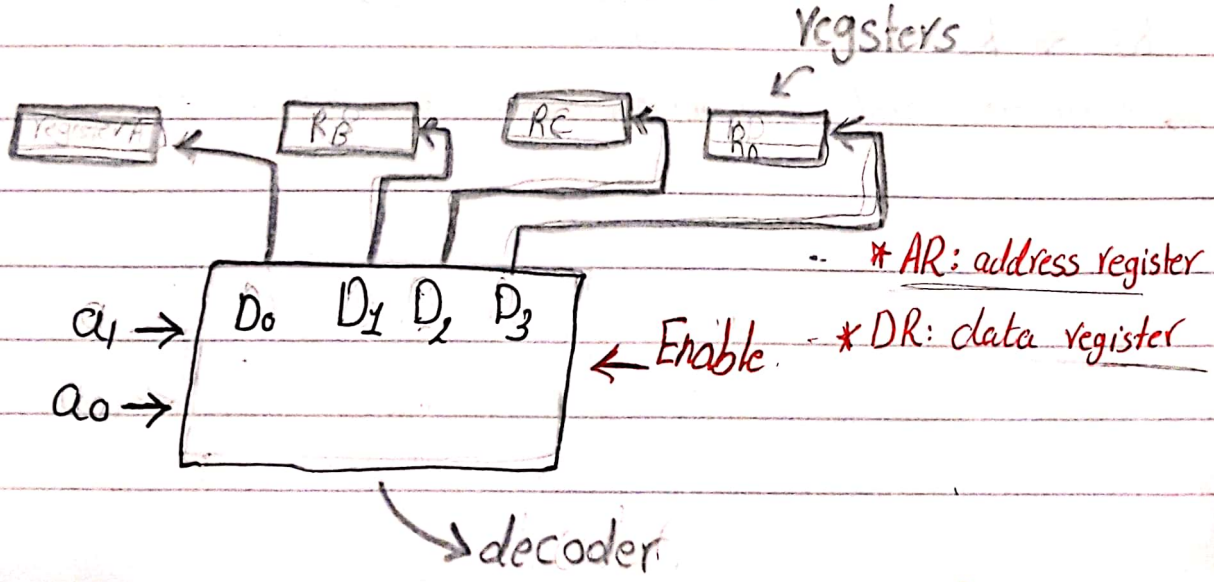
MUX = bit numbers = 16 1 MUX size = 64x1

→ Memory transfer

{transfer from memory}

- The read operation is the transfer of information from a memory word to outside enviroment: DR ← M[AR]
- The write operation is the transfe of new information to be stored into a memory word M[AR] ← R1

{transfer to memory}



* AR: address register
* DR: data register

→ Micro operations in computers can be one of three types:

1. Arithmetic
2. logic
3. Shift

$R_3 \leftarrow R_1 + R_2$ — R_1 plus R_2

$R_3 \leftarrow R_1 - R_2$ — R_1 minus R_2

$R_2 \leftarrow \bar{R}_2$ — 1's Complement of R_2

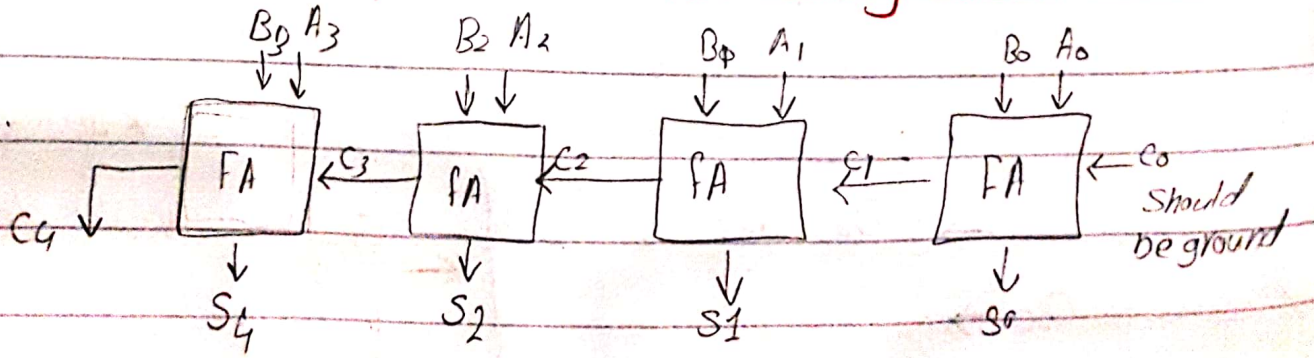
$R_2 \leftarrow \bar{R}_2 + 1$ — 2's Complement

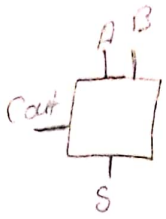
$R_3 \leftarrow R_1 + \bar{R}_2 + 1$ — 2's complement plus R_1 (Subtract) $R_1 - R_2$

$R_1 \leftarrow R_1 + 1$ — increment the contents of R_1 by 1

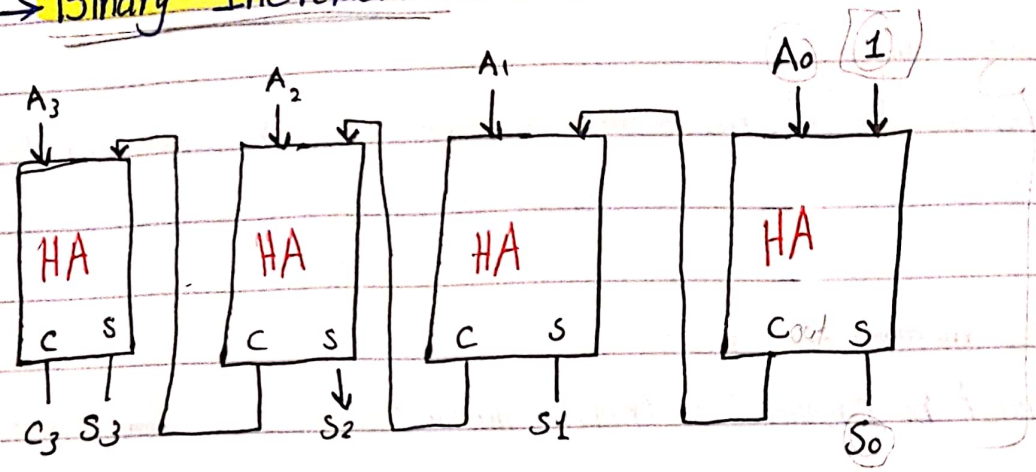
$R_1 \leftarrow R_1 - 1$ — decrement " " "

→ Arithmetic hard ware implementation "4 bit binary adder"





→ Binary Incrementor :



hardware Organization of Digital Computer :

- The Set of register it contains of function (تحتوي) (تجري على)
- The sequence of micro-operations performed on the binary information stored in the registers (يبدأ في تسلسل)
- The Control initiates the sequence of micro-operation

→ Micro - operation :

- Micro-operation are the basis of Micro-processors
- the operation executed on data stored in register are called Micro oper... (تنفذ)
- the Micro-processor performs the micro-operation in order to realize the instruction (التعليمات)
- the Micro-operations only specify which data transfers may occur (يحدث)

four types of micro-operation

- Register transfir Microoperation
- Arithmetic microoperation
- Logic microoperation
- Shift microoperation

→ Register Transfer Language :

- A sombolic language
- A convenient tool for describing internal organization of digital computers
- Can also be used to facilitate the design process of digita system

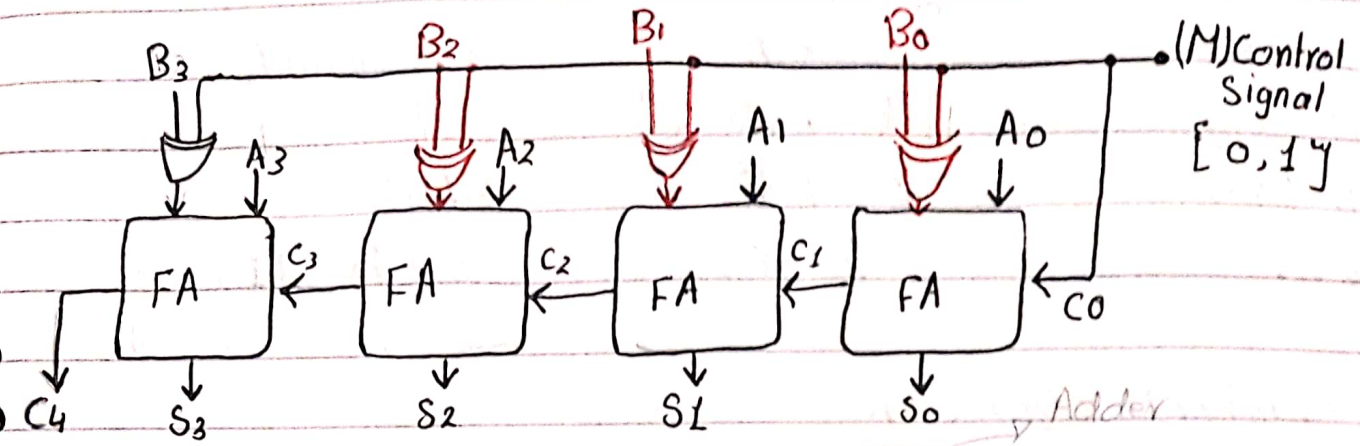
Bus and memory transfers :

- paths must be provided to transfer information from one register to another
- A Common Bus System is a scheme for transfering information between registers in a multiple-register
- A bus: set of common lines, one for each bit of a register the binary information transfared through it
- Control signals determine which register is selected by bus

→ Bus selection : two selection lines S_1 and S_0 are connected to the selection inpute

S_1	S_0	R
0	0	A
0	1	B
1	0	C
1	1	D

adder - subtractor :

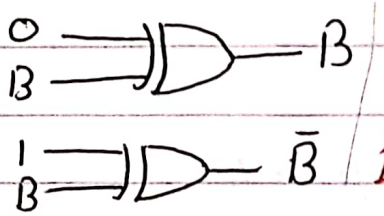


Adder

$m = 0 \Rightarrow A + B$

$m = 1 \Rightarrow A_0 + \bar{B}_0 + 1 = \boxed{A_0 - B_0}$

subtractor



$\cdot\ 0\bar{B} + \bar{0}B = B$

$1\bar{B} + \bar{1}B = \bar{B}$

* A + 1 → Incrementer
  A - 1 → Decrementer

⇒ Logic Micro Operations :

$f_0 = 0$	$F \leftarrow 0$	Clear
$f_1 = xy$	$F \leftarrow A \wedge B$	AND
$f_2 = xy'$	$F \leftarrow A \wedge \bar{B}$	AND
$f_3 = x$	$F \leftarrow A$	Transfer A
$f_4 = x'y$	$F \leftarrow \bar{A} \wedge B$	
$f_5 = y$	$F \leftarrow B$	Transfer B
$f_6 = x \oplus y$	$F \leftarrow A \oplus B$	Exclusive - OR
$f_7 = x + y$	$F \leftarrow A \vee B$	OR
$f_8 = \overline{(x + y)}$	$F \leftarrow \overline{A \vee B}$	NOR
$f_9 = \overline{(x \oplus y)}$	$F \leftarrow \overline{A \oplus B}$	XNOR

$F_{10} = y'$	$F \leftarrow B'$	complement B
$F_{11} = x + y'$	$F \leftarrow A \vee B'$	
$F_{12} = x'$	$F \leftarrow A'$	complement A
$F_{13} = x' + y$	$F = A' \vee B$	
$F_{14} = (\overline{xy})$	$F = \overline{A \wedge B}$	NAND
$F_{15} = 1$	$F \leftarrow$ all 1's	Set to all 1's

→ Shift Microoperations

- There are three types of Shift operation:
  logic Shift, circular Shift (Rotate), Arithmetic shift

R ← shl R	shift left register R
R ← shr R	Shift-right register R
R ← cil R	Circular shift-left
R ← cir R	circular shift-right
R ← ashl R	Arithmetic shif-left
R ← ashr R	Arithmetic Shift-right

Arithmitic operation ($S_3S_2 = 00$)

logic operation ($S_3S_2 = 01$)

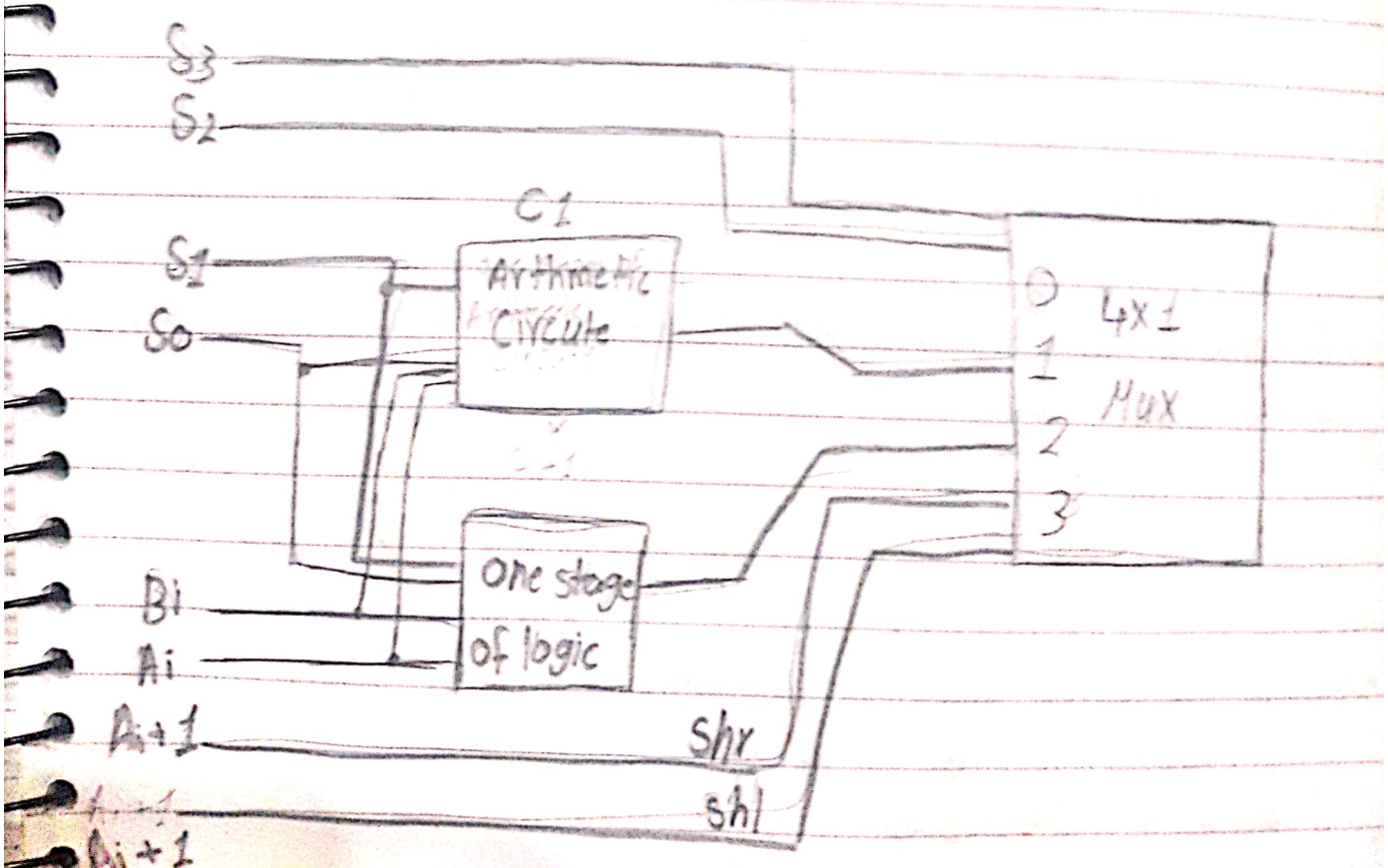
Shift right operation ($S_3\ S_2 = 10$)

Shift left operation ($S_3S_2 = 11$)

→ Shift micro-operations explained

B_{n-1}	B_{n-2}	B_{n-3}	...	B_2	B_1	B_0

$R \leftarrow shr\ R$	0	B_{n-1}	B_{n-2}	...	B_3	B_2	B_1	
$R \leftarrow shl\ R$	B_{n-2}	B_{n-3}	B_{n-4}	...	B_1	B_0	0	
$R \leftarrow cir\ R$	B_0	B_{n-1}	B_{n-2}	...	B_3	B_2	B_1	
$R \leftarrow cil$	B_{n-2}	B_{n-3}	B_{n-4}	...	B_1	B_0	B_{n-1}	
$R \leftarrow ashr\ R$	B_{n-1}	B_{n-1}	B_{n-2}	...	B_3	B_2	B_1	
$R \leftarrow ashl\ R$	B_{n-2}	B_{n-3}	B_{n-4}	...	B_1	B_0	0	



instructure Codes :

- The User of a Computer Can Control the process by mean of a Program

- A program is a set of instructions that specify the operation operands, and the Sequence by which the processing has to occur

- A computer instruction is a binary code that specifies a Sequence of micro operation for the computer to perform a Specific operation.

- The instruction Code is divided into operation part and address part

instruction code	
operation	Address

Stored program Organization :

- the simplest way to organize a computer is to have one processor and an instruction code with two parts

- instruction are Stored in one Section of memory and data in another.

- 12 bit ~~to~~ are needed to specify an address

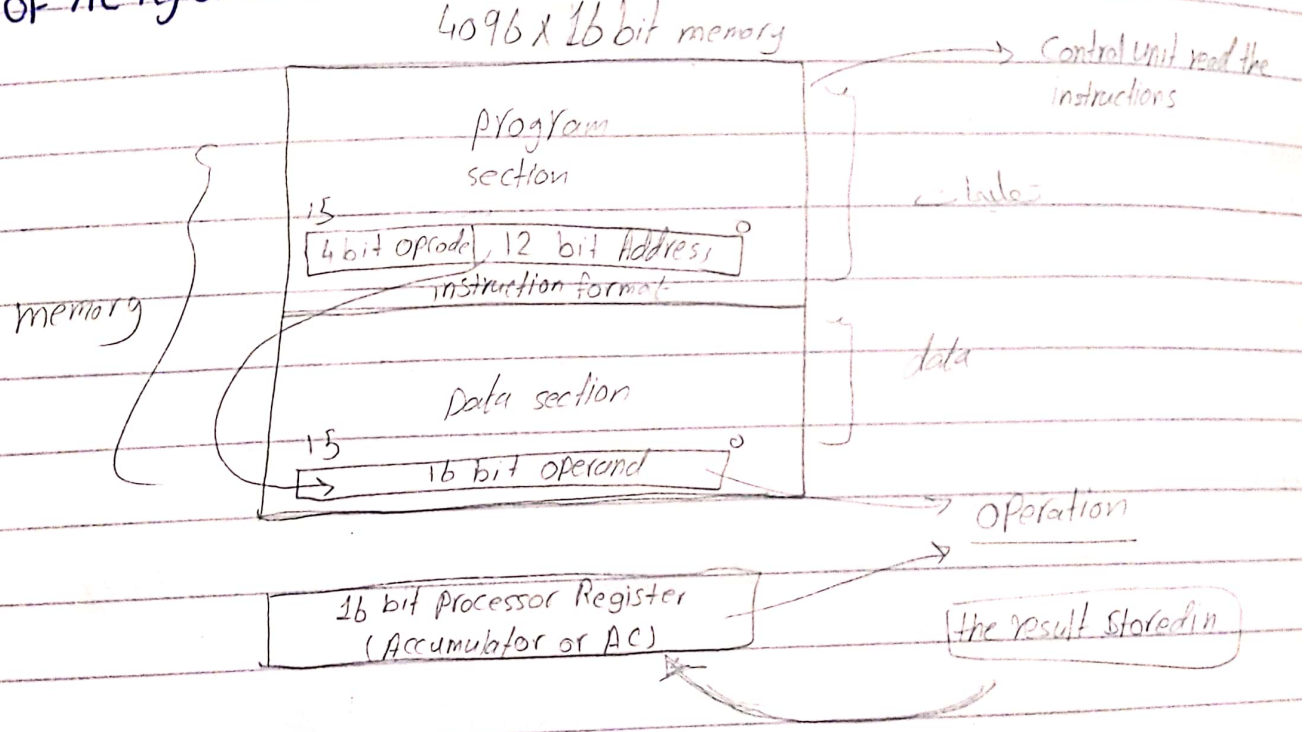
address parst length = 12 bits

Number of addresses = 2^{12} = 4096 = 4KB.

- if each instruction code is stored in one 16-bit memory word then 12 bits → in the address part will specify the address of an operand and 4 bits → will be available for the opcode part to specify one of 16 possible operations → $2^4 = 16$

4-bit	12 bit

- the operation is performed with the memory operand and the content of AC register

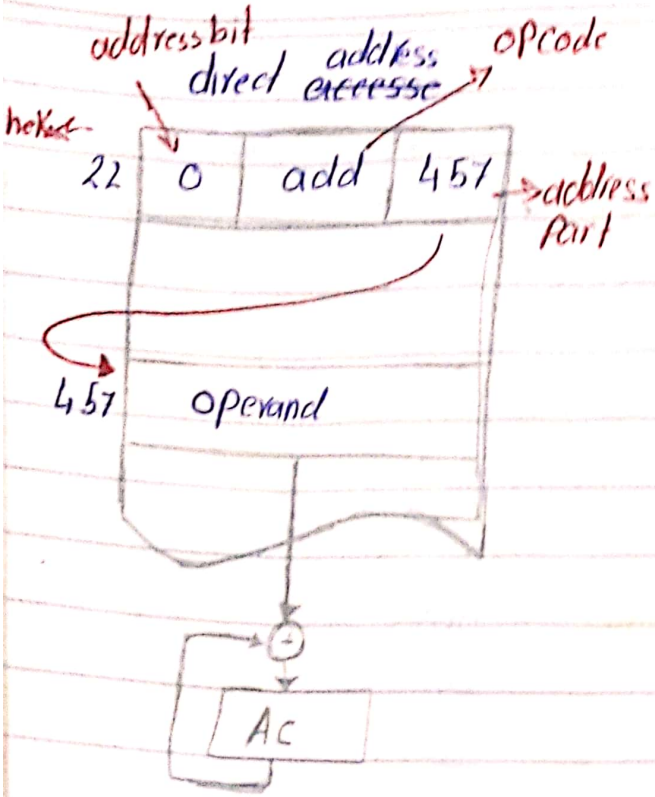


indirect Address : → Select the operation

1bit	3-bits	2-bits
I	op Code	Address → the address in the memory

to Know the difference between direct and indirect access

= 0 → Direct
= 1 → indirect

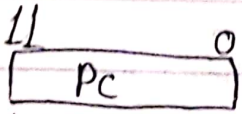


indirect address

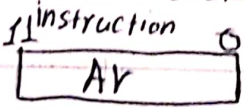
#Computer register :

↳ Registers are needed for storing Instructions and address for data manipulate.

Program counter



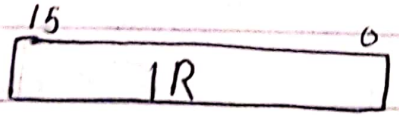
hold address of instruction

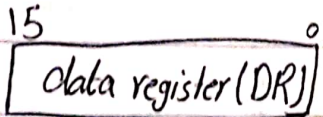


Memory 4096 X 16

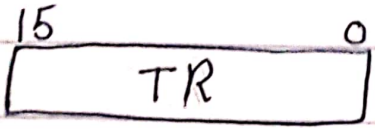
Address register (hold address of memory

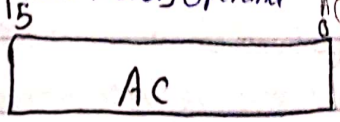
instruction register (hold instruction code)



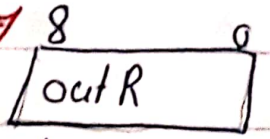


hold memory operand

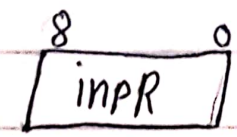




temporary register (hold temporary data)



hold output Character

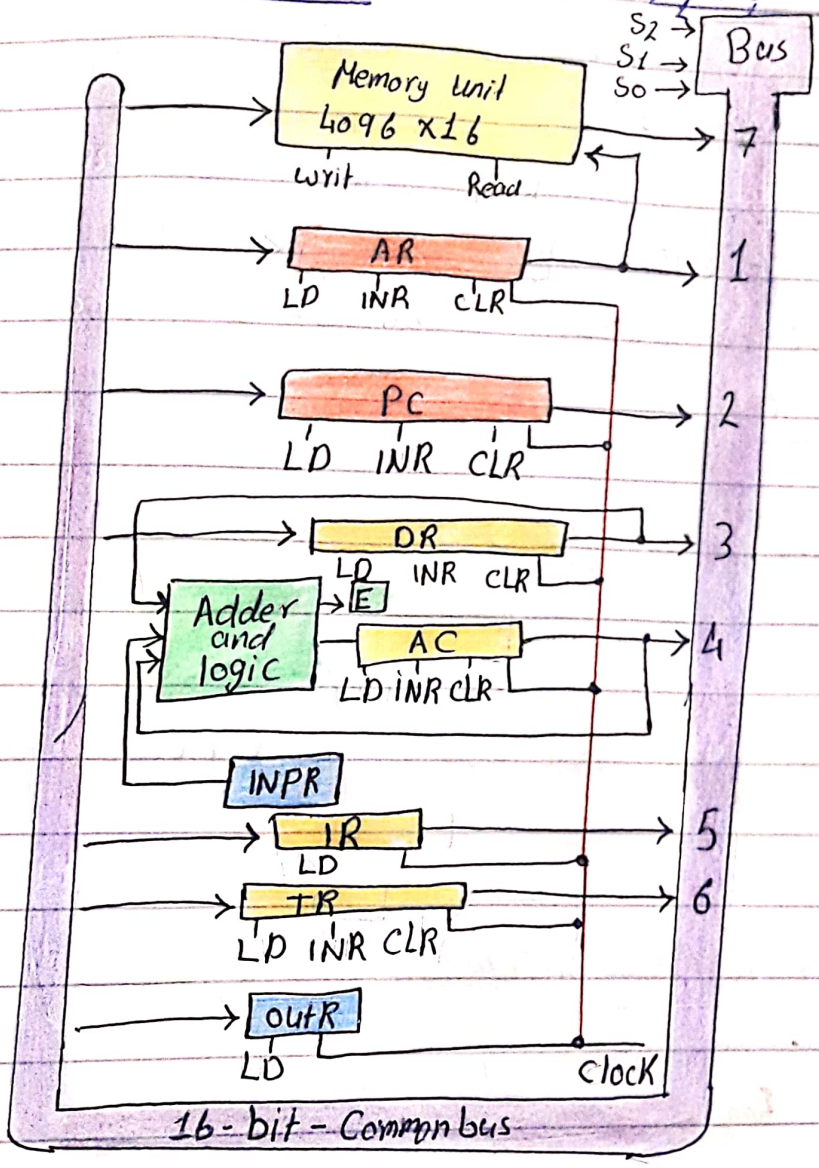


output caracter

Processor registry

Comon Bus System:

S_2 S_1 S_0	Regsiter
0 0 1	x
0 0 0	AR
0 1 1	PC
0 1 0	DR
1 0 1	AC
1 0 0	IR
1 1 1	TR
1 1 0	memory



001 → AR (1)

110 → TR

111 → memory (7)

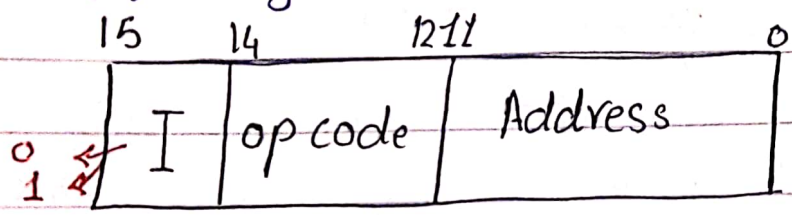
- Instruction Types:
  - Functional Instructions:
    Arithetic, logic, shift instructions
    - Add, CMA, INC, CIR, CIL, AND, CLA
  - transfer Instructions:
    - Data transfers between the main memory and processor register
    - LDA, STA

- Control Instruction:
  - program sequencing and control
  - BUN, BSA, ISZ

- INPut/output Instraction:
  - Input and out put
  - INP, OUT

Computer Instruction Types :

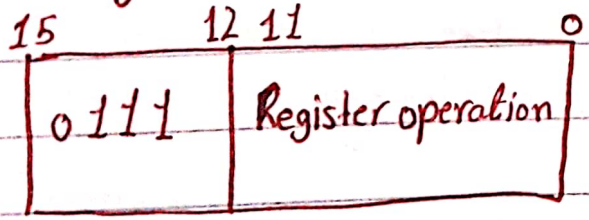
(a) Memory-reference instruction



0 ~ 6 (21)

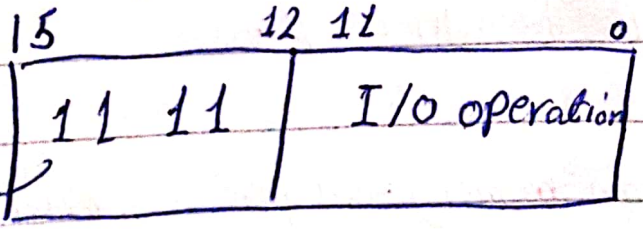
(opcode = 000 to 110

(b) Register-reference instruction



(opCod = 111 and I=0)

(C) input-out put instruction



(opCode = 111 amel I = 1)

Hex Code		Description
I = 0	I = 1	
0XXX	8XXX	AND memory word to AC
1XXX	9XXX	ADD memory word to AC
2XXX	AXXX	LDA, Load AC from memory
3XXX	BXXX	(STA), store content of AC into memory
4XXX	CXXX	(BUN), Branch unconditionally
5XXX	DXXX	(BSA), Branch and save return address
6XXX	EXXX	(ISZ), Increment and skip if Zero

7800	CLA → Clear Ac
7400	CLE → Clear E
7200	CMA → Complement AC
7100	CME → Complement E
7080	CIR → Circulate right AC and E
7040	CIL → Circulate left AC and E
7020	INC → Increment AC
7010	SPA → Skip next instr. if Ac is positive
7008	SNA → Skip next instr. if Ac is negative
7004	SZA → Skip next instr. if Ac is zero
7002	SZE → Skip next instr. if E is Zero
7001	HLT → halt computer

F800	INP	, Input caracter to AC
F400	OUT	, output character from AC
F200	SKI	, skip on input flag
F100	SKO	,skip on output flag
F080	ION	Interrupt on
F040	IOF	Interrupt off

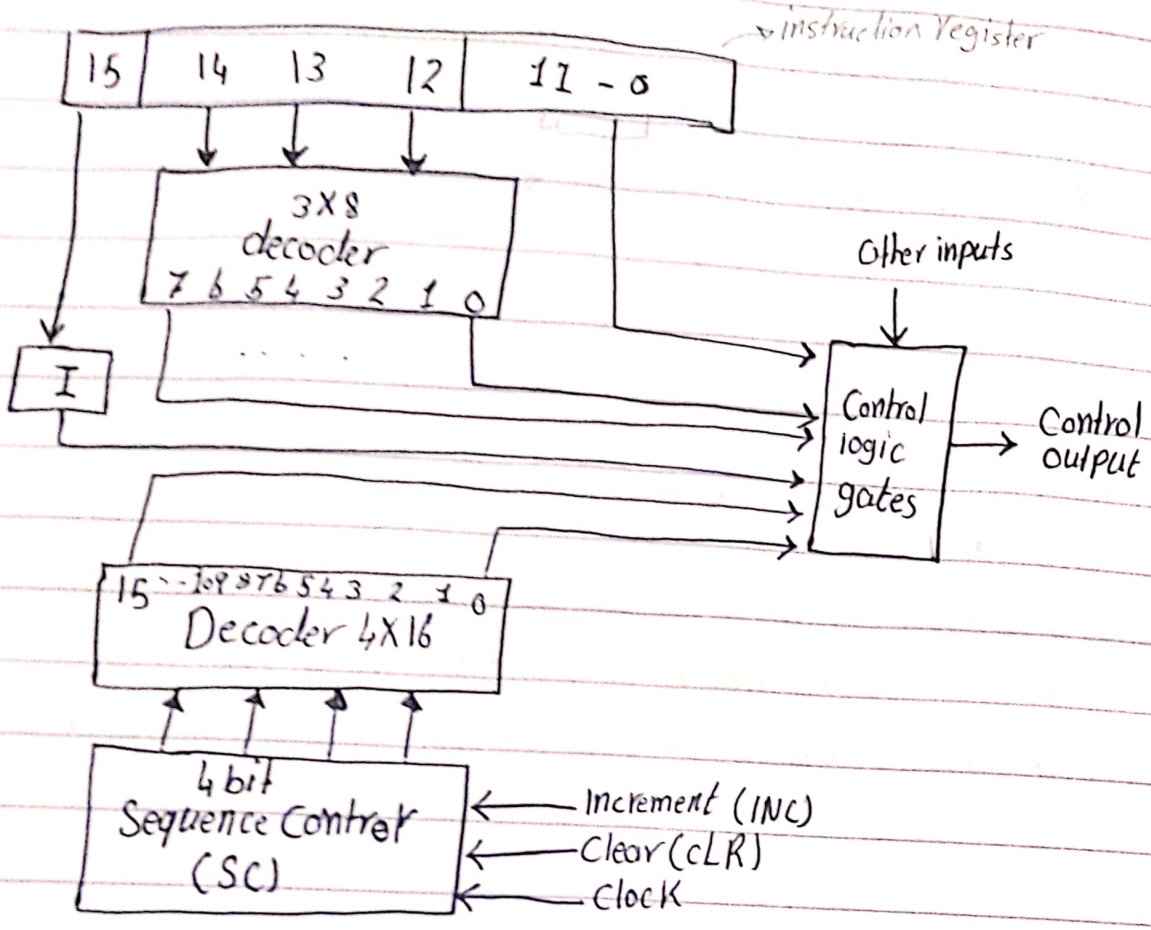
liming and Control :

- The timing for all registers in the basic computer is controlled by a master clock generator.
- The state of a register do not change unless the register is enable by a control signal which is generated in the control unit and provide control inputs for all circuits in the computer

types of Control organization :

1. hardwired control :
   - implemented with digital circuits
   - Require changes in the wiring to modify the desgin
2. Micro programmed control :
   - Control information stored in a control memory
   - Modification are done by updating the micro program

#Control Unit of Basic Computer:



Instruction cycle:

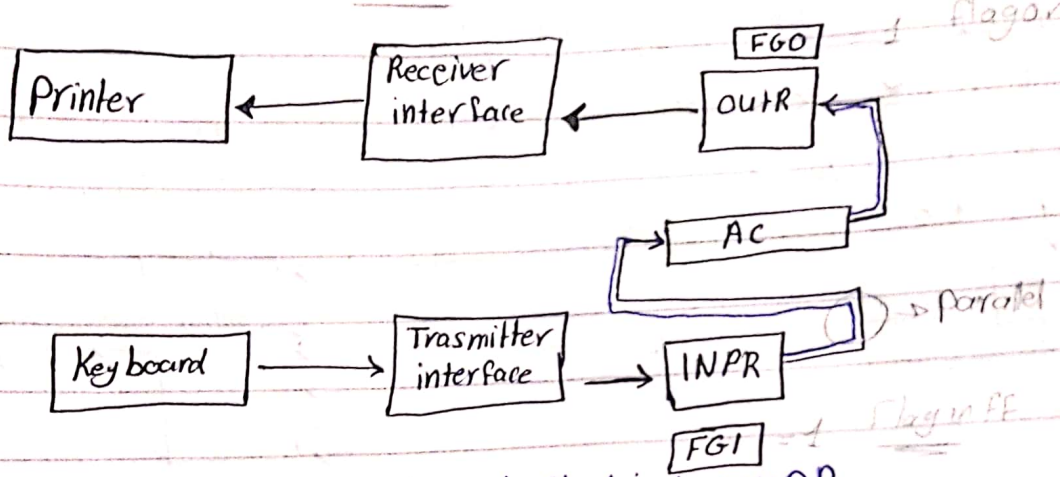
1. Fetch an instruction from memory بحث وايجاد التعليمات
2. Decode the instruction فك الشفرة وفهم العمليات
3. Read the effective address from memory . قراءة العنوان
4. Execute the Instruction تنفيذ التعليمات

- Every different processor has its own (different) cycle

INPR → 8 bits
OUTR → 8 bits
FGI → input Flag - 1bit
FGO → output Flag - 1bit
I EN → Interrupt enable - 1bit

input - output terminals :

↳ Send and receive serial information



- the serial info. from Keyboard is shifted into INPR
- " " " for the printer is stord in the OUTR
- INPR and OUTR Communicate with terminal "serially" and with Ac "parallel"
- the flags are needed to synchronize the timing difference between I/O device and the computer

#

Questions on Interrupt :

- How can the CPU recognize the device requesting an interrupt ?
  if the CPU detects a signal on the interrupt request line, it passes control to the interrput handller routine
- how can the CPU obtain the starting address of the appropriate routine in each cause ? we are aware that the fetch, decode, execute and read/write operations make up the instruction cycle
  The CPU will check for interruptions to be handled after each instruction cycle

- Should any device be allowed to interrupt the CPU while another interrupt is being serviced? → No

- how can the situation be handled when two or more interrupt requests occur simultaneously?
if there is only "single interrupt" and when multiple interrupt occurs then it will have certain external controller to handle it
وحدة تحكم خارجية

Control Unit (CU)
↳ a processor translates from machine instructions to the control signals for the microoperations that implement them ينفذ

- the ways that control unit implemented in:
  - hardwired control: متسلسلة تواقيه تولد وتنتج
  - CU is made up of sequential and combinational circuits to generate the control signals

  - Microprogrammed control:
  - Activate the necessary control signals.