Digital Logic Design

Part 1: Numbering Systems

• Types of numbering systems

Type	From	To
Binary	0	1
Octal	0	7
Decimal	0	9
Hexadecimal	0	9 + A to F

• How to Switch Between Systems from Binary

1. Take the number to be switched (for example $(5)_{10}$)
2. Divide The Number By The system's size (2 for binary, 8 for Octal etc), writing down the quotient if there is any, or putting 0 if there isn't
3. take the quotient from step 2, and write it from bottom to top

Example: Switch the number 5 from Decimal to Binary

$(5)_{10}$ =

5	**2**	**1**
2	2	0
1	2	1
0	2	0

$(101)_2$

Exercise: Switch the numbers 1-9 from Decimal to Binary

Decimal	Binary
1	1
2	10

3	11
4	100
5	101
6	110
7	111
8	1000
9	1001

•Note: Numbers Can Be From more than 1 system

• How to Switch Between Systems to Binary

1. Take the number to be switched and count how many ranks there is (ones, tens, etc)
2. Calculate The Number in binary Using The Formula "A_r* B^r +......" where r is the rank A is the number and B is the System's size

Example: Switch the number 010110 from Binary to Decimal

$(010110)_2 = 0x2^0 + 1x2^1 + 1x2^2 + 0x2^3 + 1x2^4 + 0x2^5$

$= 0 + 2 + 04+ 0 + 16 + 0 = (22)_{10}$

Example: Switch the number 35 from Octal to Decimal

$(35)_8 = 5x8^0 + 3x8^1$

$5 + 24 = (29)_{10}$

• How to Switch Systems From Binary to Octal

1. Take the number to be switched and start taking 3 digits (ones to hundreds, thousands to 100 thousands etc)
2. Convert each 3 digits to Decimal
3. write down the new number, from right to left

Example: Switch the number 110110111011 from Binary to Octal

$(110110111011)_2$

110	**110**	**111**	**011**
6	6	7	3

$(6673)_8$

• How to Switch Systems From Octal to Binary

1. Take the number to be switched and start taking each digit separately
2. Convert each 3 digits as if it was decimal to Binary (3 degits)
3. write down the new number, from right to left

Example: Switch the number 115 from Octal to Binary

175_8

1	**7**	**5**
001	111	101

$(001111101)_2$

• How to Switch Systems From Binary to Hexadecimal

1. Take the number to be switched and start taking 4 digits (ones to thousand, 10 thousands to 10 millions etc)
2. Convert each 4 digits to binary
3. write down the new number, from right to left

Example: Switch the number 11011011011 from Binary to Hexadecimal

$(1011011110111)_2$

0001	**0110**	**1111**	**0111**
1	6	15 = F	7

$(16F7)_{16}$

• How to Switch Systems From Hexadecimal to Binary

1. Take the number to be switched and start taking each digit separately
2. Convert each digit as if it was decimal to Hexadecimal (4 digit numbers)
3. write down the new number, from right to left

Example: Switch the number 16F7 from Hexadecimal to Binary

$(16F7)_{16}$

1	6	F	7
0001	0110	1111	0111

$(1011011110111)_2$

• How to Switch Systems From Hexadecimal to Octal (and vice versa)

1. Take the number to be switched and switch it to Binary
2. Convert The Binary Number to The New System

Example: Switch the number F from Hexadecimal to Octal

$(F)_{16} = (1111)_2$

001	111
1	17

$= (17)_8$

Example: convert the number 17 from Octal to Hexadecimal

$(17)_8 = (1111)_2$

1111
F

$= (F)_{16}$

How to switch Decimal Numbers

Note: Decimal Numbers as in numbers with a decimal (,), also named fractions..... ie: 7.3

1. convert the number before the decimal as normal
2. to convert the number after the decimal, multiply that number by the system's base (2 for binary etc), until a loop forms
3. when you reach a loop, take each number before the decimal and write it

Note: we multiply the numbers after the decimal, for example if we have 1.2 during multiplication we only multiply the .2, not the whole number

Example, convert $(7.3)_{10}$ to binary

7 = 111

,3 * 2 = .6

.6 * 2 = 1.2

.2 * 2 = .4

.4 * 2 = .8

.8 * 2 = 1.6 (loop, stop before here)

= 0100

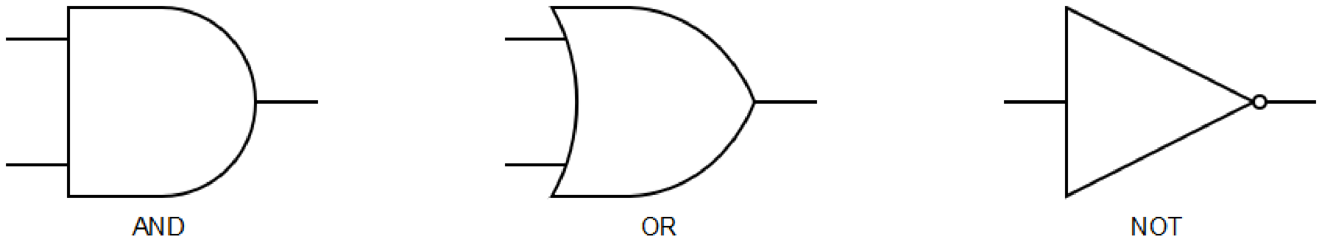
$(7.3)_{10} = (111.0100)_2$

Part 2: Boolean Algebra (logic Gates)

• Primary operator types

Type	Expression	Arabic Name	Explanation
or	add	أو	binary, needs one component to be on for it to be on

and	multiplication	و	binary, needs all components to be on for it to be on
not	X -> $\bar{X}$/ $\bar{X}$ -> X	غير	Uniary, needs the component to be off for it to be on



• Secondary operator types

1. Nand
2. Nor
3. Exor
4. Exnor

NOTE: we use the number 1 to represent on, and the number 0 to represent off

The Truth Table

General Rule: we can calculate how many options there is in a logic gate by the formula 2^n, where n is the number of components

Example: create a truth table for 2 components

A	B	OR	AND
0	0	0	0

A	B	OR	AND
0	1	1	0
1	0	1	0
1	1	1	1

Note: you can use the Octal system to memorize how a 3 component truth table

A	B	C	Equivalant
0	0	0	0
0	0	1	1
0	1	0	2
0	1	1	3
1	0	0	4
1	0	1	5
1	1	0	6
1	1	1	7

Note: you can use the Hexadecimal system to memorize how a 4 component truth table looks like

A	B	C	D	Equivalant
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	8
1	0	0	1	9
1	0	1	0	A
1	0	1	1	B

A	B	C	D	Equivalant
1	1	0	0	C
1	1	0	1	D
1	1	1	0	E
1	1	1	1	F

• Rules of Boolean Algebra

Rule Number 1:

if X = 1, then X̄ = 0 and vice versa.

Rule Number 2: Addition and Multipication Rules

A + 0 = A

A * 1 = A

A + A = A

A + Ā = 1

• Proof

A	Ā	=
1	0	1
0	1	1

A * 0 = 0

A * Ā = 0

• Proof

A	Ā	=
1	0	0
0	1	0

tldr

A	**$\bar{A}$**	**$A + \bar{A}$**	**$A * \bar{A}$**
1	0	1	0
0	1	1	0

Rule Number 3: Multiple Component Rules

$A + B = B + A$

$A B = B A$

• Example

$X + Y + Z = X +(Y + Z) =(X + Y)+ Z$

$X Y Z = X (Y Z) = (X Y) Z$

Rule Number 4: Distribution Rules

$A (B + C) = (A B) + (A * C)$

$A + (B C) =(A + B) (B + C)$

• Proof

A	**B**	**C**	**(B * C)**	**(A +B)**	**(B + C)**
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	1	1
0	1	1	1	1	1
1	0	0	0	1	0
1	0	1	0	1	1
1	1	0	0	1	1
1	1	1	1	1	1

Rule Number 5:

$X + \bar{X}Y = X + Y$

$X * (\bar{X}+Y) = XY$

• Proof

X	Y	X̄Y	X+Y	XY	X̄ + Y	X(X̄+Y)
0	0	0	0	0	1	0
0	1	1	1	0	1	0
1	0	0	1	0	0	0
1	1	0	1	1	1	1

• proof that X + X̄Y = X + Y

x *1* + *X̄* Y

x *(1 + y)*+ *X̄* Y -> 1+ y = 1

X (1 +y) + X̄ * Y = X + XY + X̄Y

(taking y as a common factor)

X + Y(X + X̄) -> (X+ X̄) = 0

X + Y

Rule Number 6: De Morgan's Rules

Rule 1: $\overline{\overline{x}}$ = x

Rule 2 = x̄ + ȳ = not (x * y)

Exercises: Simplify the following boolean expressions

1. f=A *B* + *A* $\overline{B}$ + $\overline{A}$ C +$\overline{A}$ $\overline{C}$

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

A + $\overline{A}$ = 1

2. F = x y +x y z̄ +x y z̄ + x y z

x y (1+ z̄) + xy (z̄ + z)

xy + xy = xy

3. F = x + xyz + x̄ y z + wx + x̄ y + $\overline{w}$ x

$x + yz(x + \bar{x}) + x(w + \bar{w}) + \bar{x}y$

$x + yz + x + \bar{x}y$ ---> $x + x = x$

$x + yz + \bar{x}y$ ----> $x + \bar{x}y = x + y$

$x + y + yz$

$x + y(z + 1)$

$x + y$

4. $F = x + \bar{x}\bar{y}z + \bar{x}\bar{y}\bar{z} + \bar{y}\bar{z} + \bar{y}$

$x + \bar{x}\bar{y}(z + \bar{z}) + \bar{y}\bar{z} + x + \bar{y}$

$x + \bar{x}\bar{y}(z + \bar{z}) + \bar{y}(\bar{z} + 1)$

$x + \bar{x}\bar{y} + \bar{y}$

$x + \bar{y}(\bar{x} + 1)$

$x + \bar{y}$

5. $!(!(x + \bar{y} + z)\,!(\bar{y} + \bar{z}))$

$!(!(x + \bar{y} + z)) + !(!(\bar{y} + \bar{z}))$

$x + \bar{y} + z + !(!(\bar{y} + \bar{z}))$

$x + \bar{y} + z + \bar{y} + \bar{z}$

$x + \bar{y} + \bar{y} + (z + \bar{z})$

$x + \bar{y} + 1 = 1$

6. $F = !(!(x + \bar{y})) + !(\bar{x} + y)) + !(\bar{x} + w) + !(\bar{y} + \bar{w}))$

$!(!(x + \bar{y}))\,!(!(\bar{x} + y))\,!(!(\bar{x} + w))\,!(!(\bar{y} + \bar{w}))$

$(x + \bar{y})(\bar{x} + y)(\bar{x} + w))(\bar{y} + \bar{w})$

$(x\bar{x} + \bar{x}\bar{y} + xy + y\bar{y}) == (0 + \bar{x}\bar{y} + xy + 0)$

$(\bar{x}\bar{y} + \bar{x}\bar{w} + \bar{y}w + w\bar{w}) == (\bar{x}\bar{y} + \bar{x}\bar{w} + \bar{y}w + 0)$

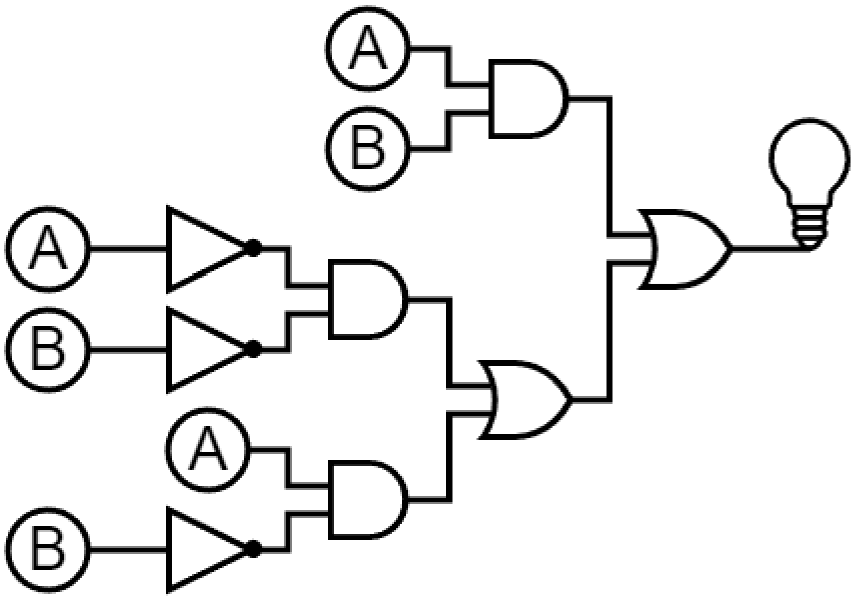
$(xy + \bar{x}\bar{y})(\bar{x}\bar{y} + \bar{x}w + \bar{y}w)$

$(x\bar{x}y\bar{y} + x\bar{x}y\overline{w} + xy\bar{y}w + \bar{x}\bar{y}\bar{x}\bar{y} + \bar{x}\bar{y}\bar{x}w + \bar{x}\bar{y}\bar{y}w)$

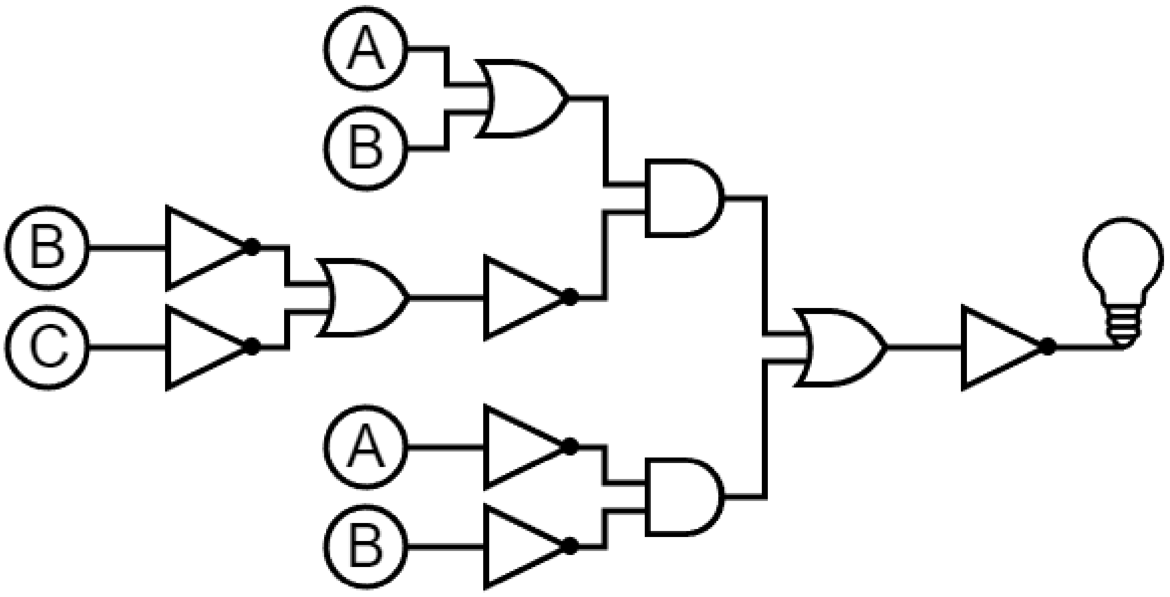
$\bar{x}\bar{y}(1+w+\overline{w}) = \bar{x}\bar{y}$

Building of Logic Circuits

$F = ab + \overline{a}\overline{b} + a\overline{b}$



$!((a+b)(\overline{b}+\overline{c})+\overline{a}\overline{b})$



Conanical Forms

Standard Forms

the form which all functions include all variables

f = $xyz + \bar{x}\bar{y}z + \bar{x}yz + xy\bar{z}$ + ...

Non Standard Forms

the form which not all functions include all variables

f = $xyz + \bar{x}\bar{y}z + \bar{x}z + xy$ + ...

Standard Forms take 1 of 2 Shapes

1. Sum of Product Form (صيغة جمع المضاريب): A number of AND functions tied by a OR function
   $F(ABC) = a\bar{b}c + \bar{a}\bar{b}\bar{c} + ab\bar{c}$.......

we can call it minterms (m)

{m ($m_0, m_1 ...... m_n$)

2. Product of Sum (صيغة ضرب المجاميع): A number of OR functions tied by an AND function
   $f(a,b,c) = (a + \bar{b} + c)(\bar{a}+\bar{b}+c)(a+\bar{b}+\bar{c})$.........
   maxterms

f(a,b,c) = p ($p_0, p_1 ...... p_n$)

A	B	C	Minterms	Maxterms
0	0	0	m = 0	p = 7
0	0	1	m = 1	p = 6
0	1	0	m = 2	p = 5
0	1	1	m = 3	p = 4
1	0	0	m = 4	p = 3
1	0	1	m = 5	p = 2
1	1	0	m = 6	p = 1
1	1	1	m = 7	p = 0

conclusion: the minterms and maxterms compliment each other, as when added up they should make a number equal to the number of $2^{functions}-1$

{m = !{p
m =!p

$F = \bar{a} + bc$

$f(a,b,c) = \bar{a} + bc$

to make it to standard form we need

for $(\bar{a}) = \bar{a}\,(\bar{b}\bar{c} + b\bar{c} + \bar{b}c + bc)$

for $(bc) = (bc)\,(a+\bar{a})$

the final product will be

$\bar{a}\bar{b}\bar{c} + \bar{a}b\bar{c} + \bar{a}\bar{b}c + \bar{a}bc + \bar{a}bc + abc$

{m(0,1,2,3,7)

Karnaugh Map

2 form karnaugh map

form	**$\bar{y}$**	**y**
$\bar{x}$	00	01
x	10	11

3 form karnaugh map = $x + \bar{x}y = x + y$

form	**$\bar{y}\bar{z}$**	**$\bar{y}z$**	**yz**	**$y\bar{z}$**
$\bar{x}$	000 = 0	001 = 1	011 = 3	010 = 2
x	100 = 4	101 = 5	111 = 7	110 = 6

4 form karnaugh map

form	**$\bar{z}\bar{w}$**	**$\bar{z}w$**	**zw**	**$z\bar{w}$**
$\bar{x}\bar{y}$	0000 = 0	0001 = 1	0011 = 3	0010 = 2
$\bar{x}y$	0100 = 4	0101 = 5	0111 = 7	0110 = 6
xy	1100 = C	1101 = D	1111 = F	1110 = E

form	**$\bar{z}\bar{w}$**	**$\bar{z}w$**	**zw**	**$z\bar{w}$**
$x\bar{y}$	1000 = 8	1001 = 9	1011 = B	1010 = A

How to Simplify equations using Karnaugh Maps

1. Make A truth table for all the variables in an equation, and calculate the result of the operation, adding t/1 or f/0, depending on the result
2. make a karnaugh map for the variables, adding 1 to where it says so in the truth table
3. draw a square over as many 1 is possible, as long as the number of 1s is a power of 2 (2,4,8 etc), and they are near each other (top and bottom/left and right of each other)
4. simplify each square seperatly, then add them together in or statments

example : simplify the following equations using a k-map

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

A	**B**	**C**	**result**
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

form	**$\bar{a}\bar{b}$**	**$\bar{a}b$**	**ab**	**$a\bar{b}$**
$\bar{c}$	0	1	1	0
c	0	0	1	1

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

$cab + ca\bar{b}$ --> $ca(b+\bar{b})$ --> ca

$\overline{c}b + ca$

$\overline{a}c + \overline{a}b + ab + ac$

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

form	$\overline{a}\overline{b}$	$\overline{a}b$	ab	$a\overline{b}$
$\overline{c}$	0	1	1	0
c	1	1	1	0

$\overline{a}bc + \overline{a}\overline{b}c \text{ --> } \overline{a}c$

$c\overline{a}b + cab + \overline{c}\overline{a}b + \overline{c}ab \text{ --> } b$

$\overline{a}c + b$

$\overline{ac} + b\overline{c} + a\overline{b}c$

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

form	$\overline{a}\overline{b}$	$\overline{a}b$	ab	$a\overline{b}$
$\overline{c}$	1	1	1	0

form	$\bar{a}\bar{b}$	$\bar{a}b$	ab	$a\bar{b}$
c	0	0	0	1

$\bar{a}\bar{b}\bar{c}$ + $\bar{a}b\bar{c}$ --> $\bar{a}\bar{c}$

$\bar{a}b\bar{c}$ + $ab\bar{c}$ -->$b\bar{c}$

$a\bar{b}c$

$\bar{a}\bar{c}$ + $b\bar{c}$ + $a\bar{b}c$

$\bar{c}$ ($\bar{a}\bar{b}$đ+d) + ađc + đ

A	B	C	D	Result
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	1
0	1	1	1	0
1	0	0	0	1
1	0	0	1	0
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	1
1	1	1	1	0

form	$\bar{c}$đ	$\bar{c}d$	cd	cđ
$\bar{a}\bar{b}$	1			1
$\bar{a}b$	1			1
ab	1			1
$a\bar{b}$	1			1

$\bar{a}\bar{b}\bar{c}$đ + $\bar{a}b\bar{c}$đ + $ab\bar{c}$đ +$a\bar{b}\bar{c}$đ + $\bar{a}\bar{b}c$đ + $\bar{a}bc$đ + abcđ + $a\bar{b}c$đ --> $\bar{c}$đ +cđ

đ

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

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

form	$\bar{a}\bar{b}$	$\bar{a}b$	ab	$a\bar{b}$
$\bar{c}$	1	1	1	0
c	1	0	1	0

$\bar{c}\bar{a}\bar{b}$+ $c\bar{a}\bar{b}$ --> $\bar{a}\bar{b}(\bar{c}+c)$ --> $\bar{a}\bar{b}$

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

$\bar{c}ab$ + $\bar{c}ab$ --> $ab(\bar{c}+c)$--> ab

$\bar{a}\bar{b}$ + $\bar{c}b$ + ab

$\bar{a}c+ \bar{a}b + a\bar{b}c+ bc$

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

A	B	C	result
1	1	1	1

form	a̅b̅	a̅b	ab	ab̅
c̅	0	1	0	0
c	1	1	1	1

ca̅b̅ + ca̅b + cab + cab̅ --> c(a̅b̅+a̅b+ab+ab̅) -->c(a̅+a) --> c

ca̅b + c̅a̅b --> a̅b (c̅+c) --> a̅b

a̅b + c

Don't Care

used in a k-map

looks like a d or an x

only used to make the square bigger, never taken alone, used as seen fit

example: solve the following k-map

F(a,b,c,d)

{(m2,m3,m5,m9,m12)

d(m0,m1,m11,m7)

form	c̅đ	c̅d	cd	cđ
a̅b̅	d	d	1	1
a̅b			d	
ab	1			
ab̅		1	d	

a̅b̅c̅đ + a̅b̅c̅d + a̅b̅cd + a̅b̅cđ = a̅b̅

a̅b̅c̅d + a̅b̅cd + ab̅c̅d + ab̅cd = a̅b̅d(c̅ + c) + ab̅d (c̅ + c) --> b̅d(a̅ +a) --> b̅d

a̅b̅ + b̅d + abc̅đ

f(a,b,c)
{(m0,m1,m5)

d(m3,m6)

form	$\bar{a}\bar{b}$	$\bar{a}b$	ab	$a\bar{b}$
$\bar{c}$	1	1	d	0
c	0	1	0	d

$\bar{a}\bar{b}\bar{c} + \bar{a}b\bar{c}$ --> $\bar{a}\bar{c}$ ($\bar{b}$+ b) --> $\bar{a}\bar{c}$

$\bar{a}b\bar{c} + \bar{a}bc$ --> $\bar{a}b$ (c+$\bar{c}$) --> $\bar{a}b$

$\bar{a}\bar{c} + \bar{a}b$

Half adder Circle

normally, 1+1 = 2, however in binary 1+1 = 10, where we take the 0 and

1. **Sum (S):** Output of a half adder representing the result of XOR operation on input bits A and B.
2. **Carry (C):** Output of a half adder representing the result of AND operation on input bits A and B.

normally, 1+1 = 2, however in binary 1+1 = 10, where we take the 0 and make it sum, and the 1 gets carried so the sum becomes 0, and the carry become 1, where if we add another 1 to the operation it becomes 0 +1, where the sum becomes 1 and the carry stays 1 as well

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

form	$\bar{a}$	a
$\bar{b}$	0	1
b	1	0

s = $\bar{a}b + a\bar{b}$

a xor b

$(5)_{10} + (3)_{10}$

1 0 1 + 0 1 1 = 1000

x	y	z	S	C
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

s =

form	$\bar{y}\bar{z}$	$\bar{y}z$	yz	$y\bar{z}$
$\bar{x}$	0	1	0	1
x	1	0	1	0

$s = \bar{x}\bar{y}\bar{z} + \bar{x}y\bar{z} + x\bar{y}\bar{z} + xyz$

c =

form	$\bar{y}\bar{z}$	$\bar{y}z$	yz	$y\bar{z}$
$\bar{x}$	0	0	1	0
x	0	1	1	1

$x\bar{y}\bar{z} + xy\bar{z} = x\bar{z}$

$xy\bar{z}+ xyz = xy$

$xyz + \bar{x}yz = yz$

$c = x\bar{z} + xy + yz$

Full Adder

basically a half adder but for more than 2 variables (3 or more)

x	y	z	S	C
0	0	0	0	0

x	y	z	S	C
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

s =

form	ȳz̄	ȳz	yz	yz̄
x̄	0	1	0	1
x	1	0	1	0

s = x̄ȳz̄ + x̄yz̄ + xȳz̄ + xyz

c =

form	ȳz̄	ȳz	yz	yz̄
x̄	0	0	1	0
x	0	1	1	1

xȳz̄ + xyz̄ = xz̄

xyz̄+ xyz = xy

xyz + x̄yz = yz

c = xz̄ + xy + yz

The Decoder

basically used to transfom numbers into letters using the ASCI code

The general property of a decoder is that it has 2^n output lines for n input lines. Each output line corresponds to a unique combination of input values.

example 2 = 2^2

out of all numbers produced, only one is active, where the rest are inactive until called

example 2 = 2^2

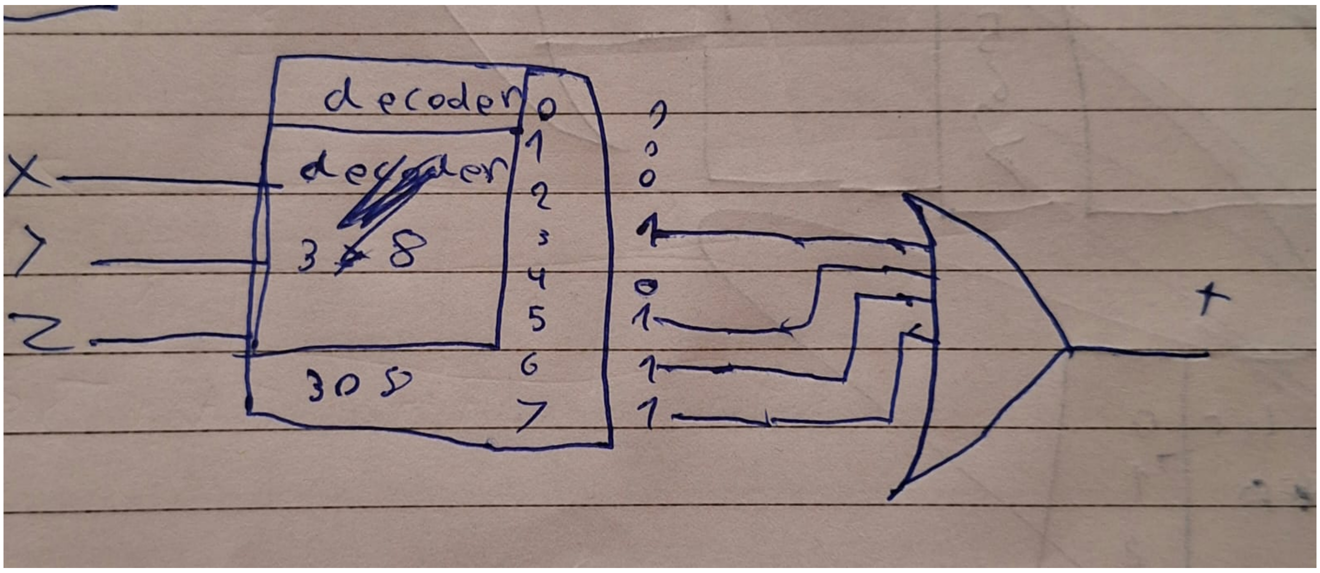
00 - active
01 - inactive
10 - inactive
11 - inactive

Q) create a decoder for a 3 sum of product where most inputs are 1

basically if 2 or more variables = 1, then f = 1

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

Drawing

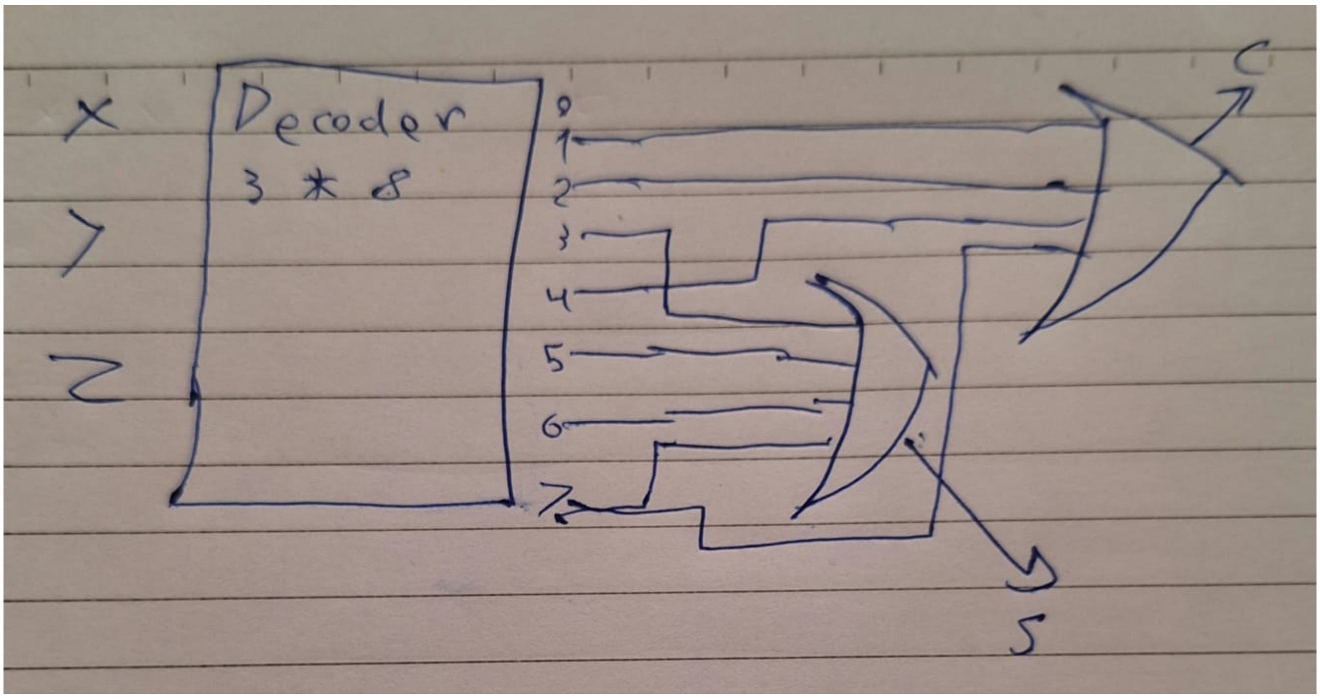


Q) create a decoder using a full adder

x	y	z	S	C

x	y	z	s	c
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Drawing



Enable Signal

gets into all gates and decides which one is active in a decoder

The Encoder

Sequential circuit contains an n input and n output

log $_2{}^n$ >= n

does the opposite of a decoder

2^n -> n

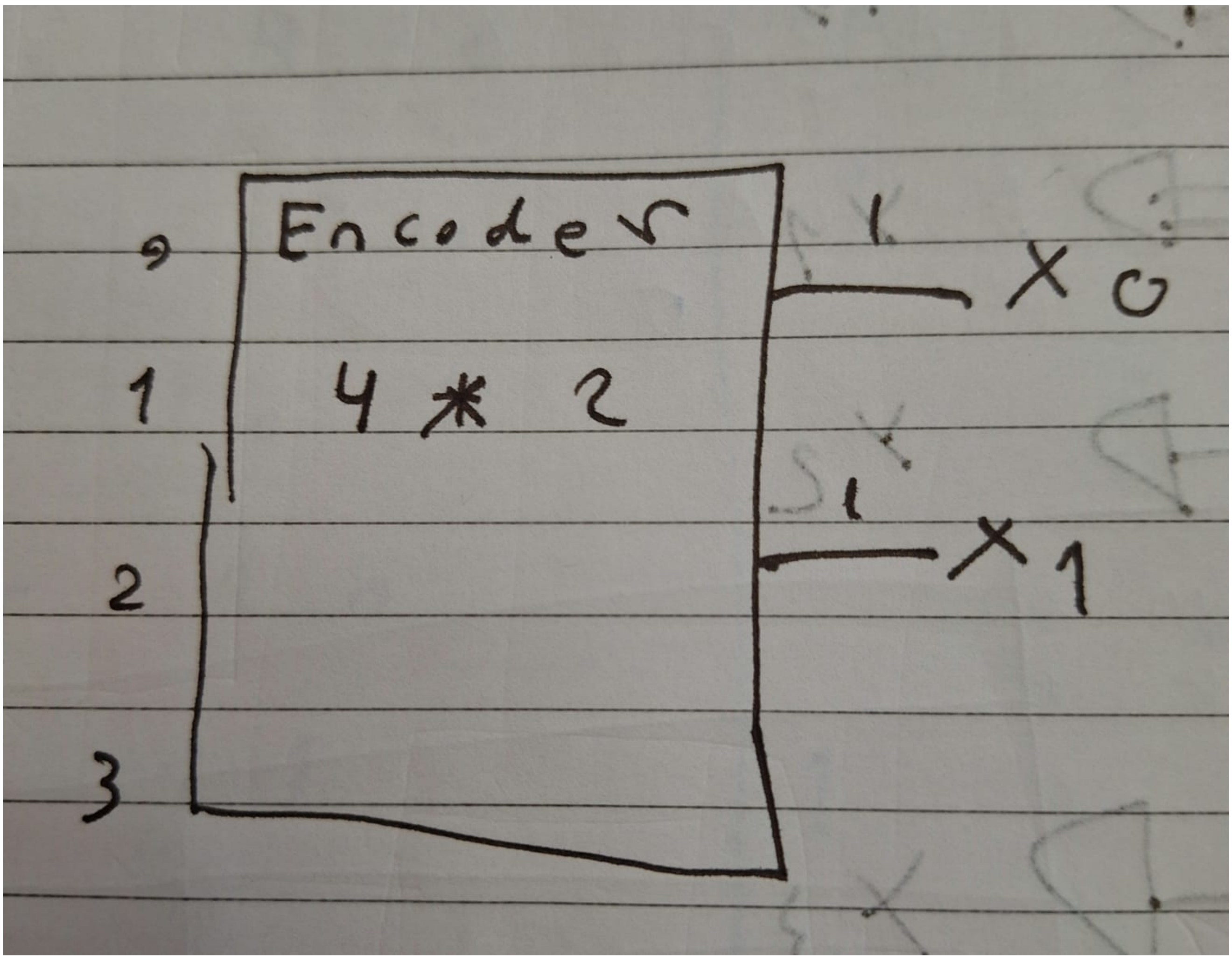
from decimal to binary

the outputs of an encoder are the inputs of a decoder and vice versa

example: using an incoder, solve the following

A	B
0	0
0	1
1	0
1	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



The Multiplexer (mux) / Data Selector

Multi input Sequential logic circuit, that has one output

2^n input

1 output

uses a select signal to pick which input to output

$\log_2{}^4 = 2$

$\log_2{}^8 = 3$

Question: design a circuit using a multiplexer

f = {m(0,1,2,5,9,10,13,15)

$2^n = 16$

n = 4

x_0, x_1, x_2, x_3

we take the biggest value which is x3

	i_0	i_1	i_2	i_3	i_4	i_5	i_6	i_7
$\bar{x}_3$	0	1	2	3	4	5	6	7
x_3	8	9	10	11	12	13	14	15

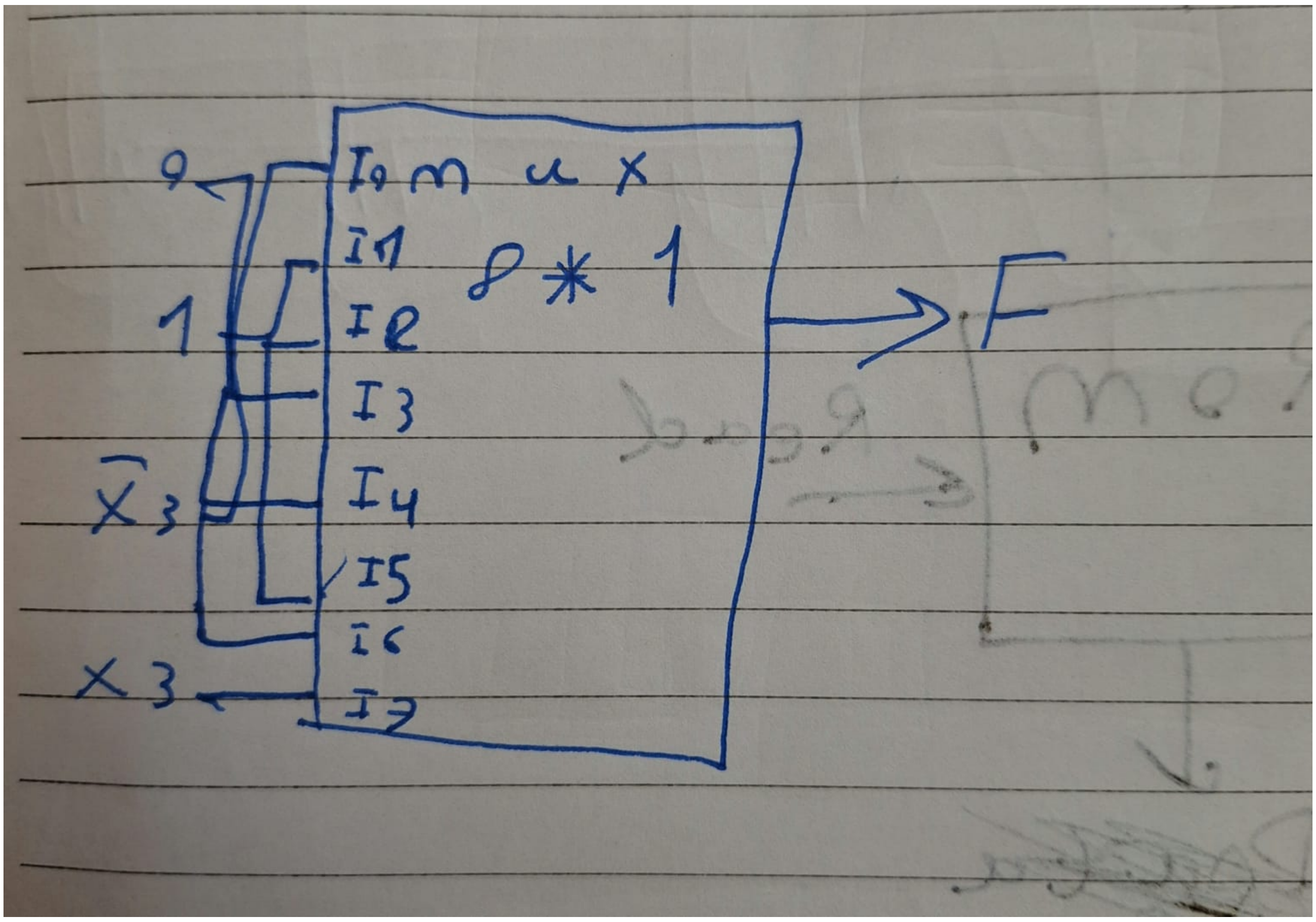
circle the min terms values

	i_0	i_1	i_2	i_3	i_4	i_5	i_6	i_7
$\bar{x}_3$	0 #	1 #	2 #	3	4	5 #	6	7
x_3	8	9#	10 #	11	12	13 #	4	15#
	$\bar{x}_3$	1	1	0	0	1	0	x_3

if there are no circles in a row we put 0
if there is a circle on top but not on bottom we put $\bar{x}_3$ (note: letter and number are interchangeable)
if there are circles on top and bottom we put 1

if there is a circles on bottom and not top we put x_3 (note: letter and number are interchangeable)

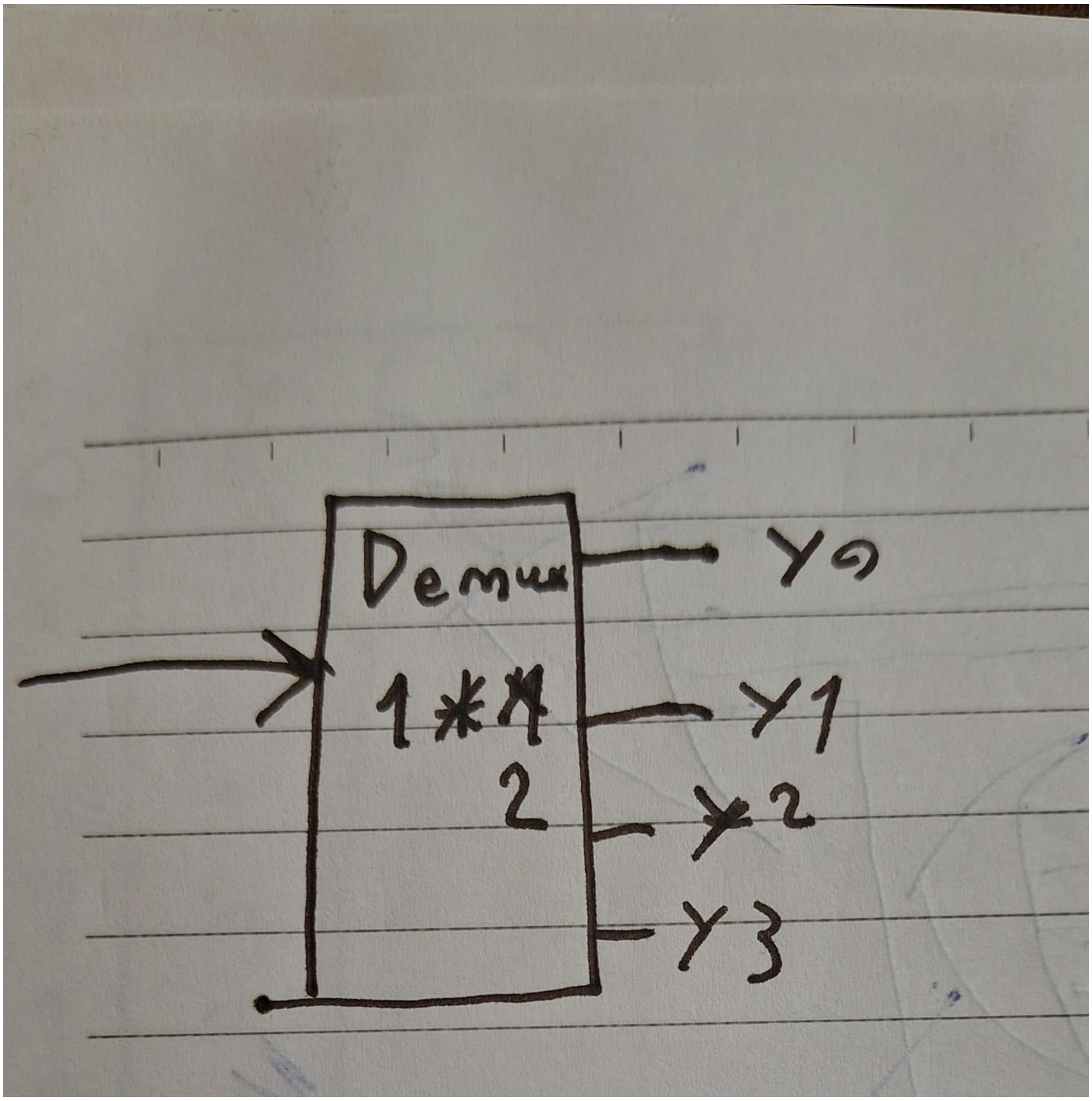
drawing tldr: using the table we made above, connect each i with the appropriate value (0,1,x̄,x̄)



De-multiplexer (de-mux)

a Combinational Circuit that does the opposite operation of a multiplexer 1 input

many outputs



x0	x1	y0	y1	y2	y3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

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

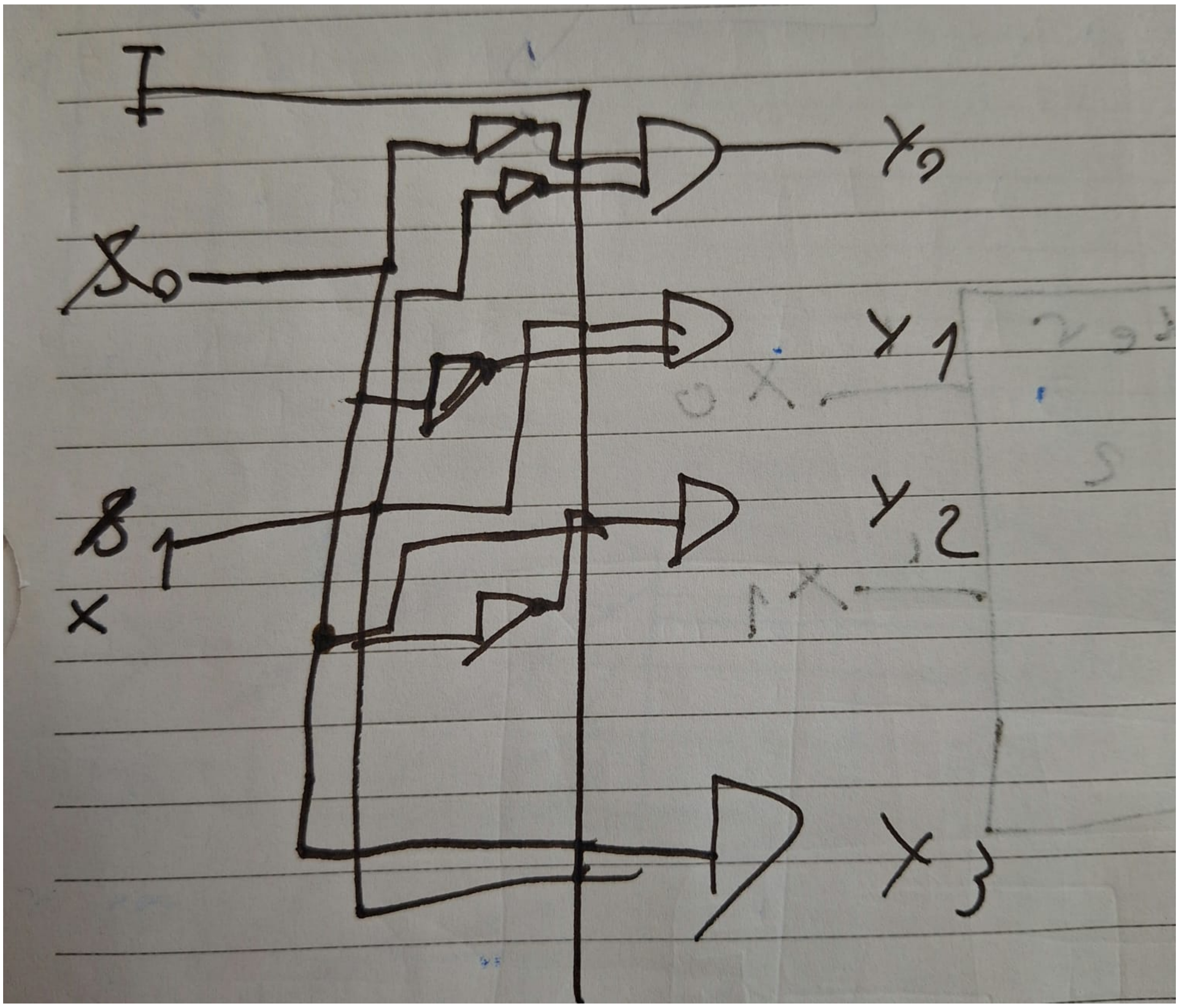
$y_1 = \bar{x}_0 x_1 y_1$

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

$y_3 = x_0 x_1 y_3$

Drawing

tldr: follow the truth table results to draw, add an I that links to every output



The ROM

Stands for "Read Only Memory"

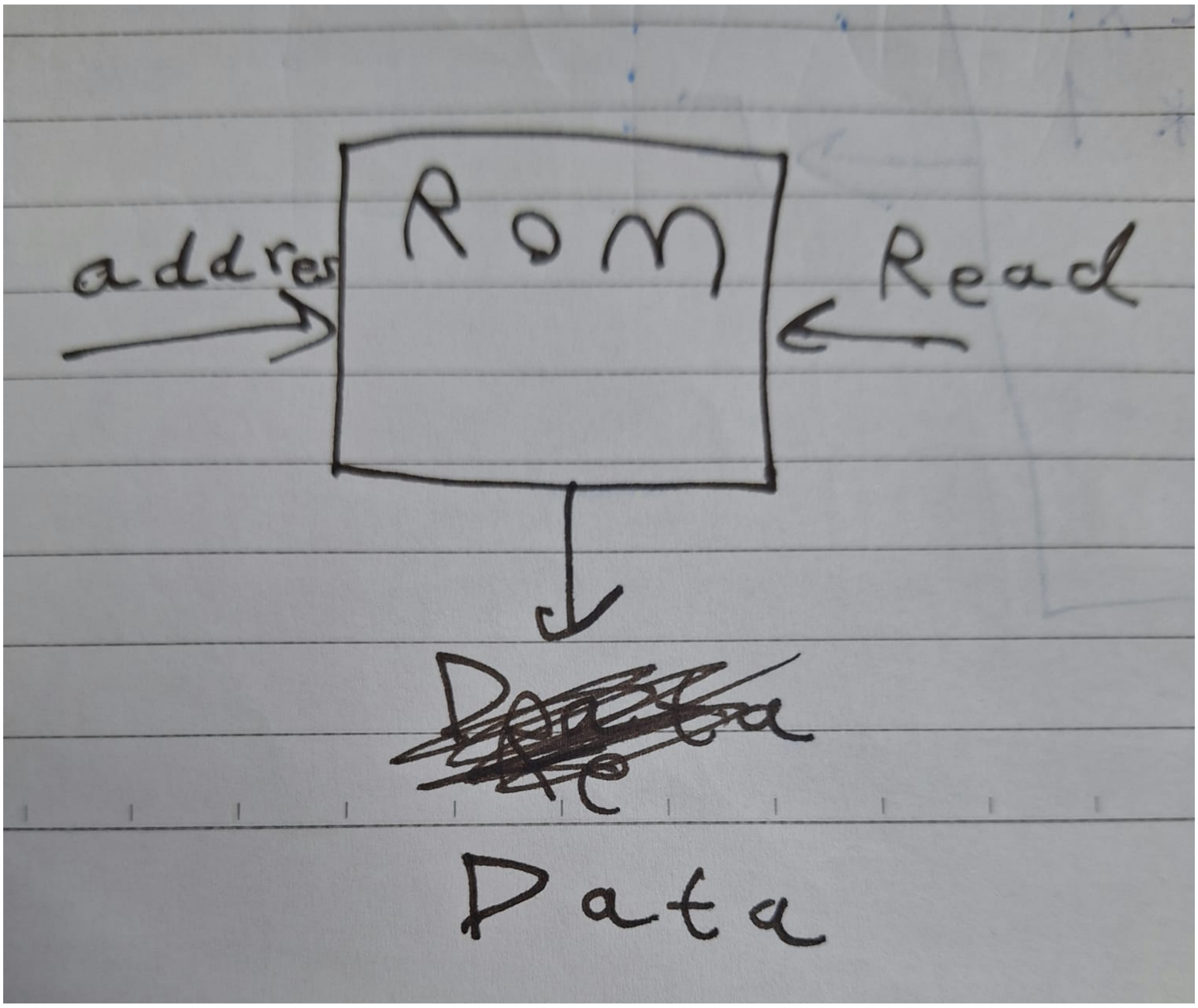
The Memory which we can only read but cannot write on it

the OS and other consts are typically stored on it

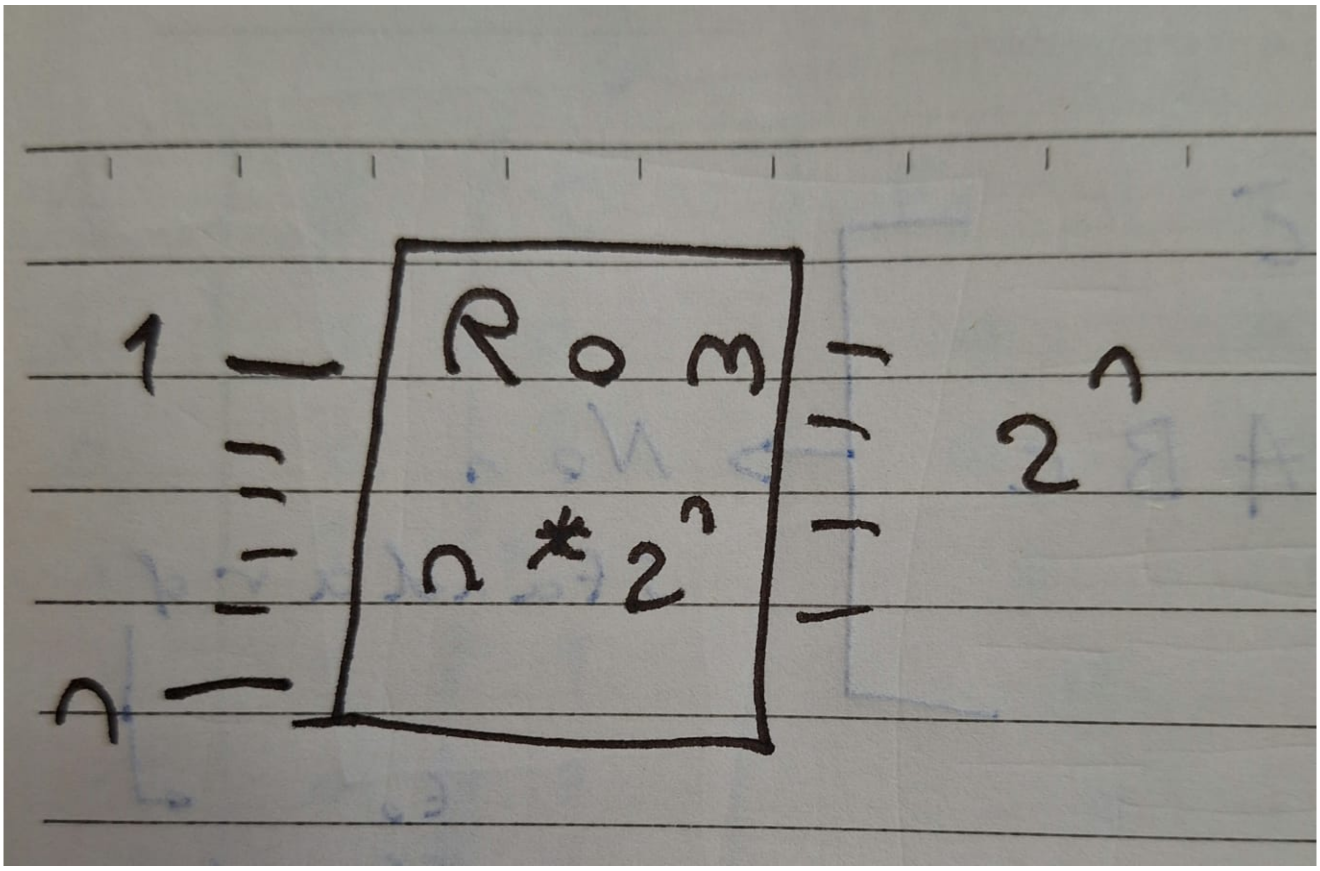
it uses addresses to get data

made from cells, that store the necessary data

all memory types are made using different locations, that are found from addresses , with every location having a different address



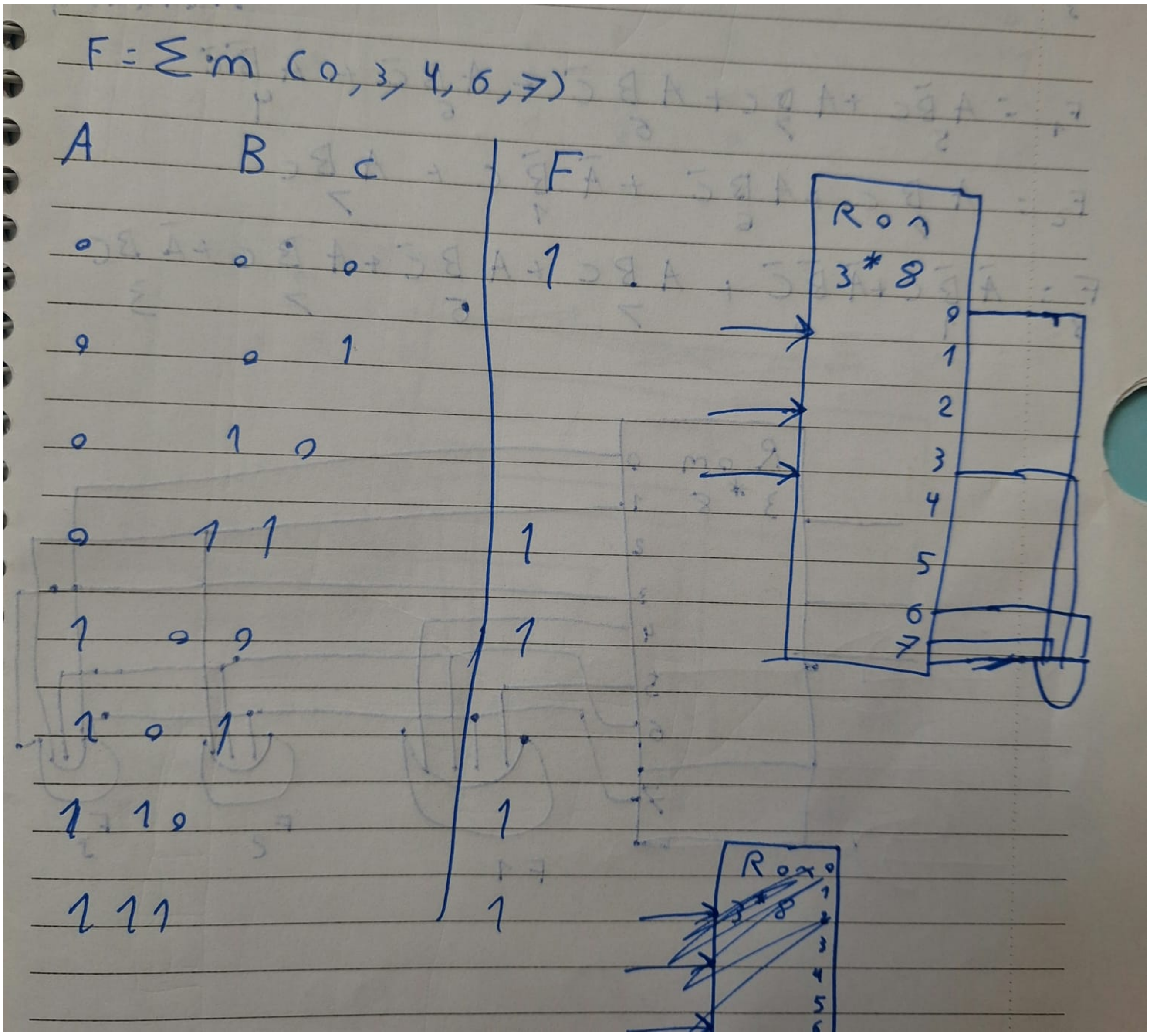
the shape is similar to a decoder



Examples

example 1.create a rom that takes n inputs

f = {m(0,3,4,6,7)



$F = \sum m(0,3,4,6,7)$

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

Example 2. create a rom for a function

$F_1 = A\bar{B}C + AB + A\bar{C}$

$F_2 = AB + \bar{A}\bar{B}C + ABC$

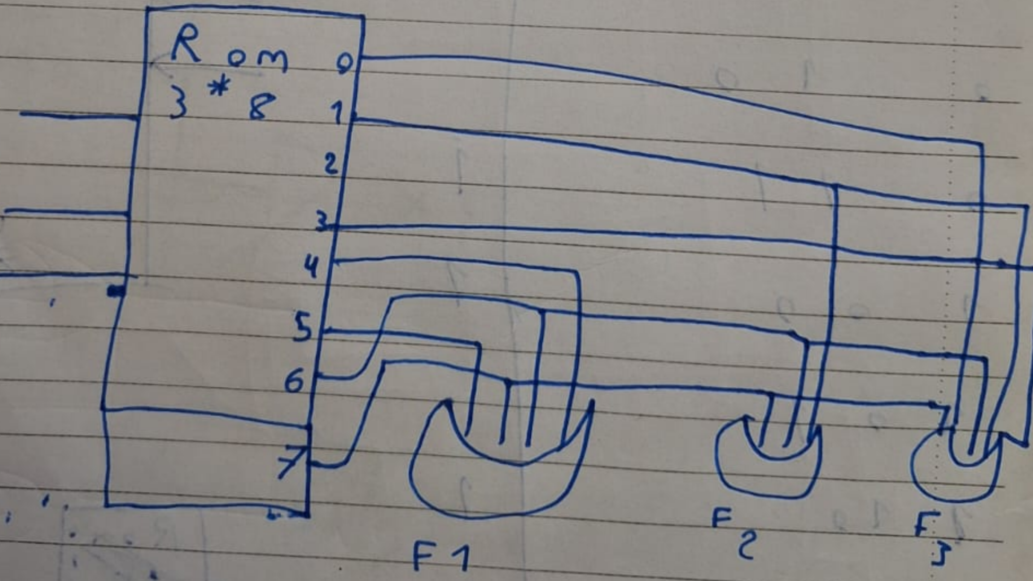
$F_3 = \bar{A}\bar{B} + AB + BC$

$\rightarrow$ Non standard $\rightarrow$ to standard

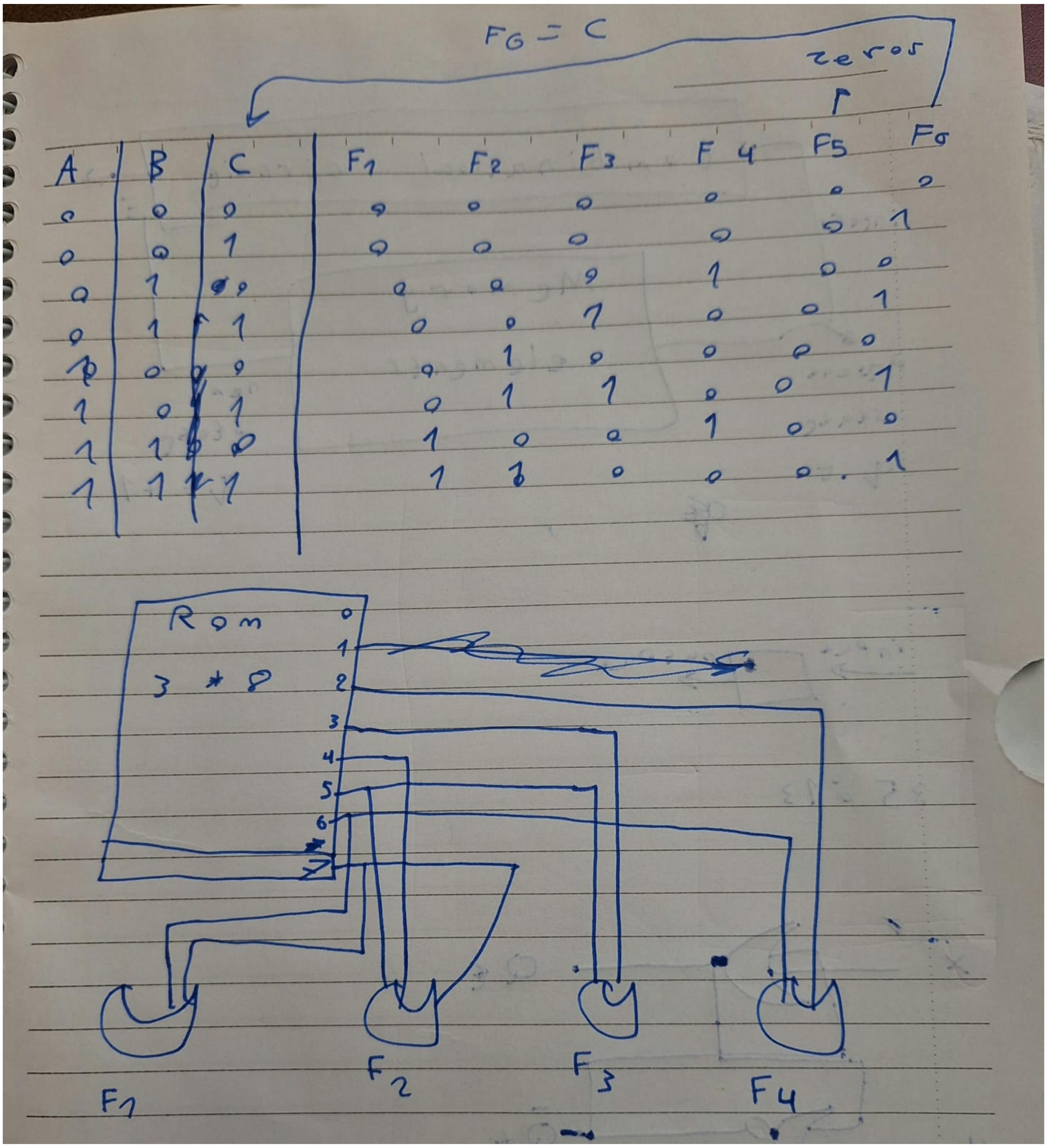
$F_1 = \underset{5}{A\bar{B}C} + \underset{7}{ABC} + \underset{6}{AB\bar{C}} + \underset{6}{AB\bar{C}} + \underset{4}{A\bar{B}\bar{C}}$

$F_2 = \underset{7}{ABC} + \underset{6}{AB\bar{C}} + \underset{1}{\bar{A}\bar{B}C} + \underset{7}{ABC}$

$F_3 = \underset{1}{\bar{A}\bar{B}C} + \underset{0}{\bar{A}\bar{B}\bar{C}} + \underset{7}{ABC} + \underset{6}{AB\bar{C}} + \underset{7}{ABC} + \underset{3}{\bar{A}BC}$



example 3.create a logic circuit that has a 3 bit input and outputs the square of an input



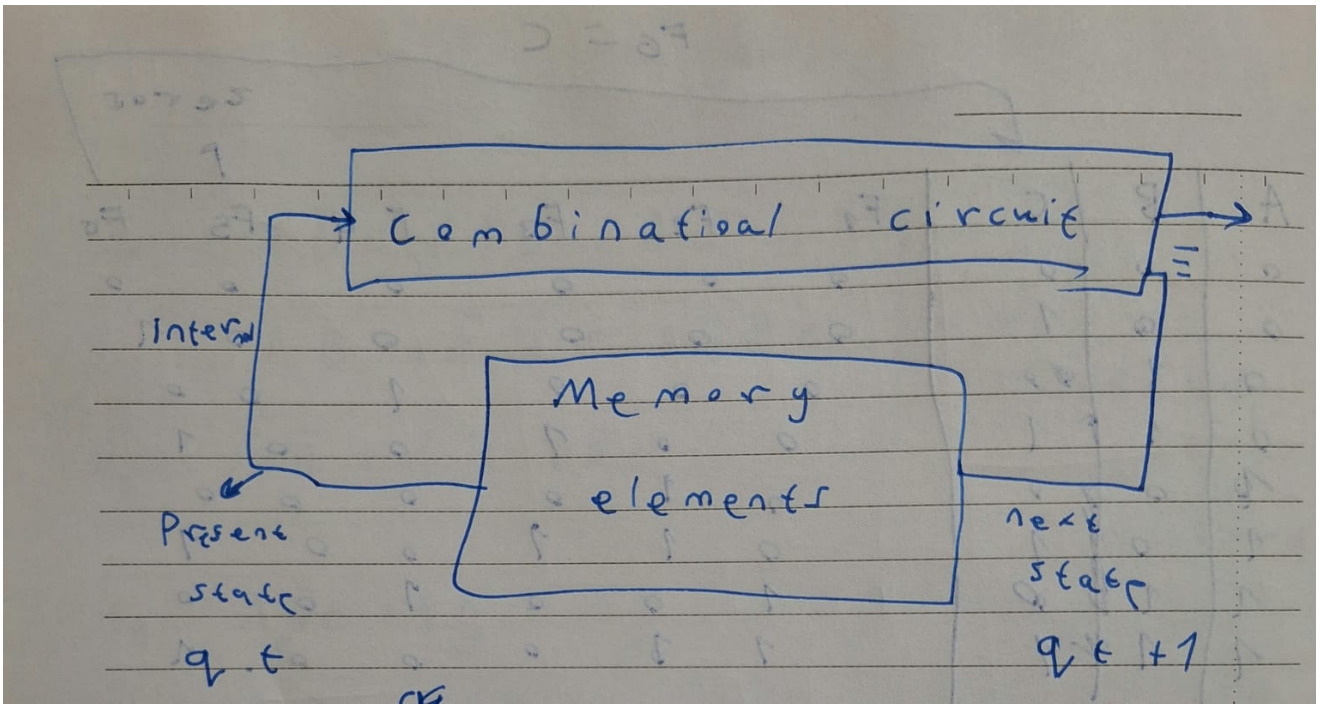
A	B	C	F1	F2	F3	F4	F5	F6
0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	1
0	1	0	0	0	0	1	0	0
0	1	1	0	0	1	0	0	1
1	0	0	0	1	0	0	0	0
1	0	1	0	1	1	0	0	1
1	1	0	1	0	0	1	0	0
1	1	1	1	1	0	0	0	1

Sequential Logic Circuit

Asynchronic غير متزامنة

Synchronic متزامن

Sequential Logic Circuit have what's called memory elements, and feedback from the input and output



flip-flops

المرجاح

قلاب بالسوري

نطاط بالعراقي

electronic circuits with two stable states, that are 1 and 0

if found in 0, it's always stable in 0 until the data is changed, and vice versa

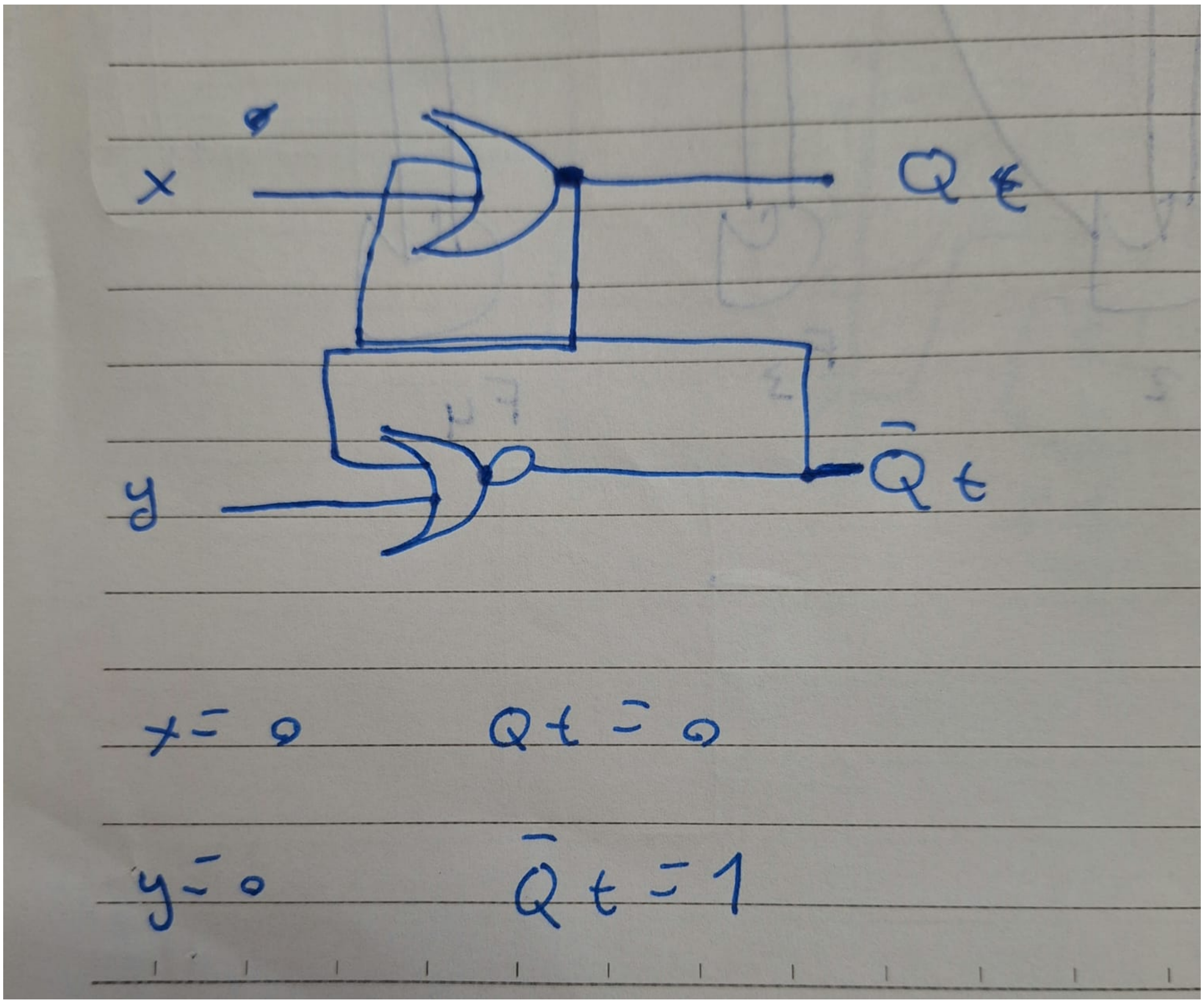
example: counters

it can store either 0 or 1, which is why it's an important component in the memory

has atleast one input, and on it the state of the flip flop depends

generally and most of the time, the flip flop circuits have more than 1 inputs and 2 outputs, with each output storing 1 bit (either 1 or 0)

the most important (and simplest) example is Not/or (latch circuit)



Adder/Substractor

in computers there is no such thing as subtractors, instead there is complements (basically the number that when added to another, equals the number of digits in a system (2 for binary 8 for octal etc))

two's complement = one's complement + 1

example --> 011+ 1 = 100

how to add and substract binary numbers

1. **Initialize:**
   - Draw a series of full adders equal to the number of bits in the system.
2. **Determine Operation:**
   - Establish whether you are performing addition or subtraction.
3. **Setup for Operation:**
   - Place a line above the full adders.

- Position XOR gates above each full adder on this line.
- Set the XOR gate values to 0 if for addition, or 1 if for subtraction.

4. **Initial Carry Input:**
    - Draw a line before the first full adder.
    - Set the value on this line to 0 for addition, or 1 for subtraction.
5. **Process First Bit:**
    - For the first bit (a_0) of the first value:
        - Add it to the value from step 4, keeping track of the sum and carry.
6. **XOR Gate Operation:**
    - For the first bit (b_0) of the second value:
        - Compare it with the corresponding XOR gate:
            - If the bits are the same, set the XOR gate output to 0.
            - If the bits are different, set the XOR gate output to 1.
7. **Full Adder Operation:**
    - Combine the XOR gate output and the sum from step 5 using the full adder.
    - Draw a line to represent the carry, pointing to the next full adder in the sequence.
    - Repeat steps 5-7 for each subsequent bit, moving from the least significant bit (LSB) to the most significant bit (MSB).

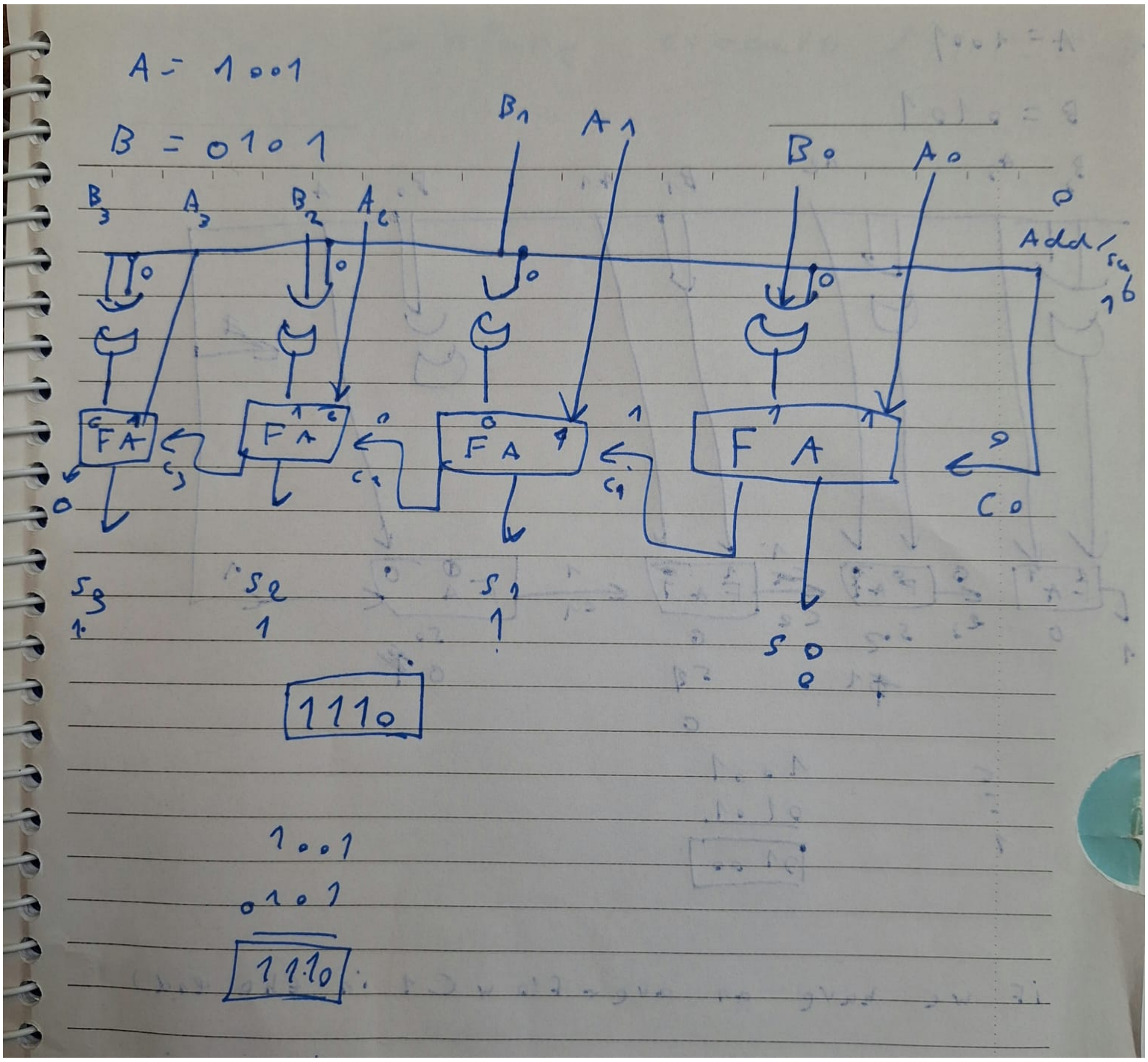
there might be carry of 1 after the final operation, this is called overflow

examples

1.add the following digits

a = 1001

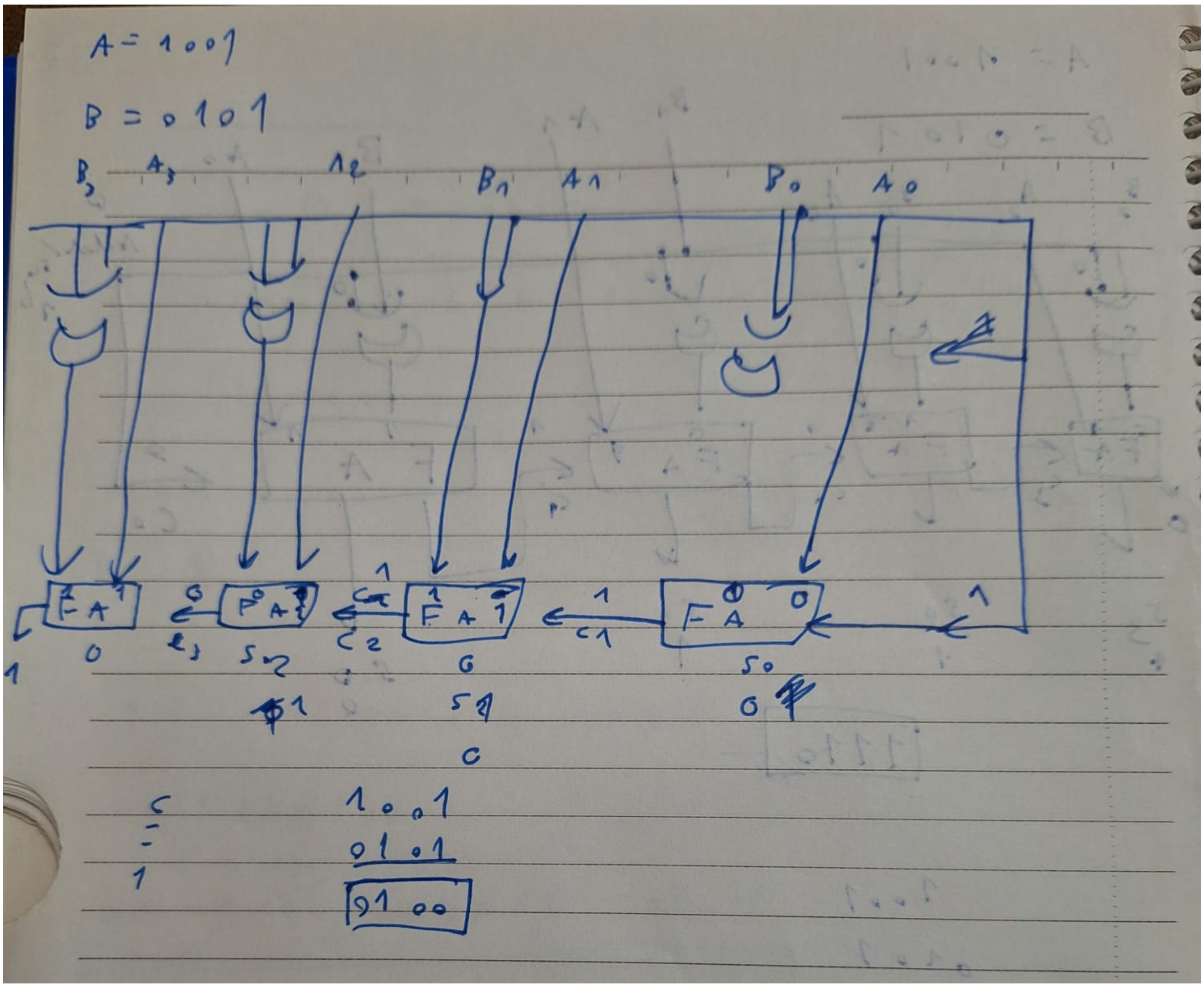
b = 0101



2.add the following digits

a = 1001

b = 0101



Compare Circuit / magenitic comparator

a combinational circuit that compares between 2 binary numbers inputs , and generates 3 output signals to indicate the result of the comparison

Types of comparisons

1. Greater than
2. Less than
3. Equal

example : design a 4 bit magenitic comparator to compare between 2 2 bit numbers

$a = a_0a_1$

$b = b_0b_1$

a_0	a_1	b_0	b_1	**G**	**L**	**E**
0	0	0	0	0	0	1
0	0	0	1	0	1	0
0	0	1	0	1	0	0
0	0	1	1	0	1	0
0	1	0	0	1	0	0
0	1	0	1	0	0	1
0	1	1	0	0	1	0
0	1	1	1	0	1	0
1	0	0	0	1	0	0
1	0	0	1	1	0	0
1	0	1	0	0	0	1
1	0	1	1	0	1	0
1	1	0	0	1	0	0
1	1	0	1	1	0	0
1	1	1	1	0	0	1

we can also make k-maps for every output

G k-map

G	$\bar{b}_0\bar{b}_1$	$\bar{b}_0b_1$	b_0b_1	$b_0\bar{b}_1$
$\bar{a}_0\bar{a}_1$	0	0	0	0
$\bar{a}_0a_1$	1	0	0	0
a_0a_1	1	1	0	1
$a_0\bar{a}_1$	1	1	0	0

L k-map

L	$\bar{b}_0\bar{b}_1$	$\bar{b}_0b_1$	b_0b_1	$b_0\bar{b}_1$
$\bar{a}_0\bar{a}_1$	0	1	1	1
$\bar{a}_0a_1$	0	0	1	1

L	$\bar{b}_0\bar{b}_1$	$\bar{b}_0b_1$	b_0b_1	$b_0\bar{b}_1$
a_0a_1	0	0	0	0
$a_0\bar{a}_1$	0	0	1	0

E k-map

E	$\bar{b}_0\bar{b}_1$	$\bar{b}_0b_1$	b_0b_1	$b_0\bar{b}_1$
$\bar{a}_0\bar{a}_1$	1	0	0	0
$\bar{a}_0a_1$	0	1	0	0
a_0a_1	0	0	1	0
$a_0\bar{a}_1$	0	0	0	1

We can then simplify the equations and create circuits

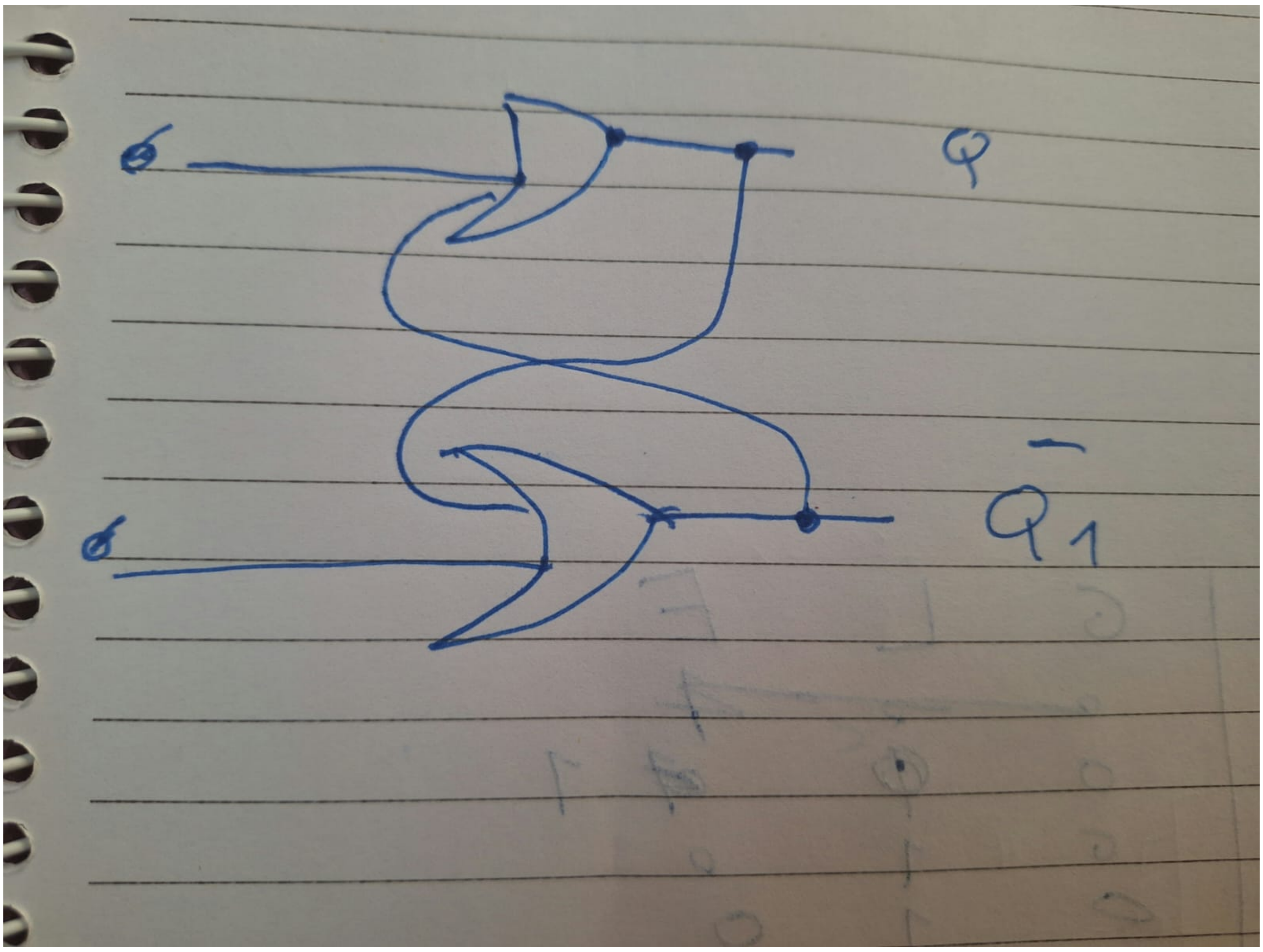
drawing of the compare circuit

“Schematic-of-2-bit-comparator-using-logic-gates.png” could not be found.

The RS flip flop (reset/set)

uses the NOR GATE

basic drawing

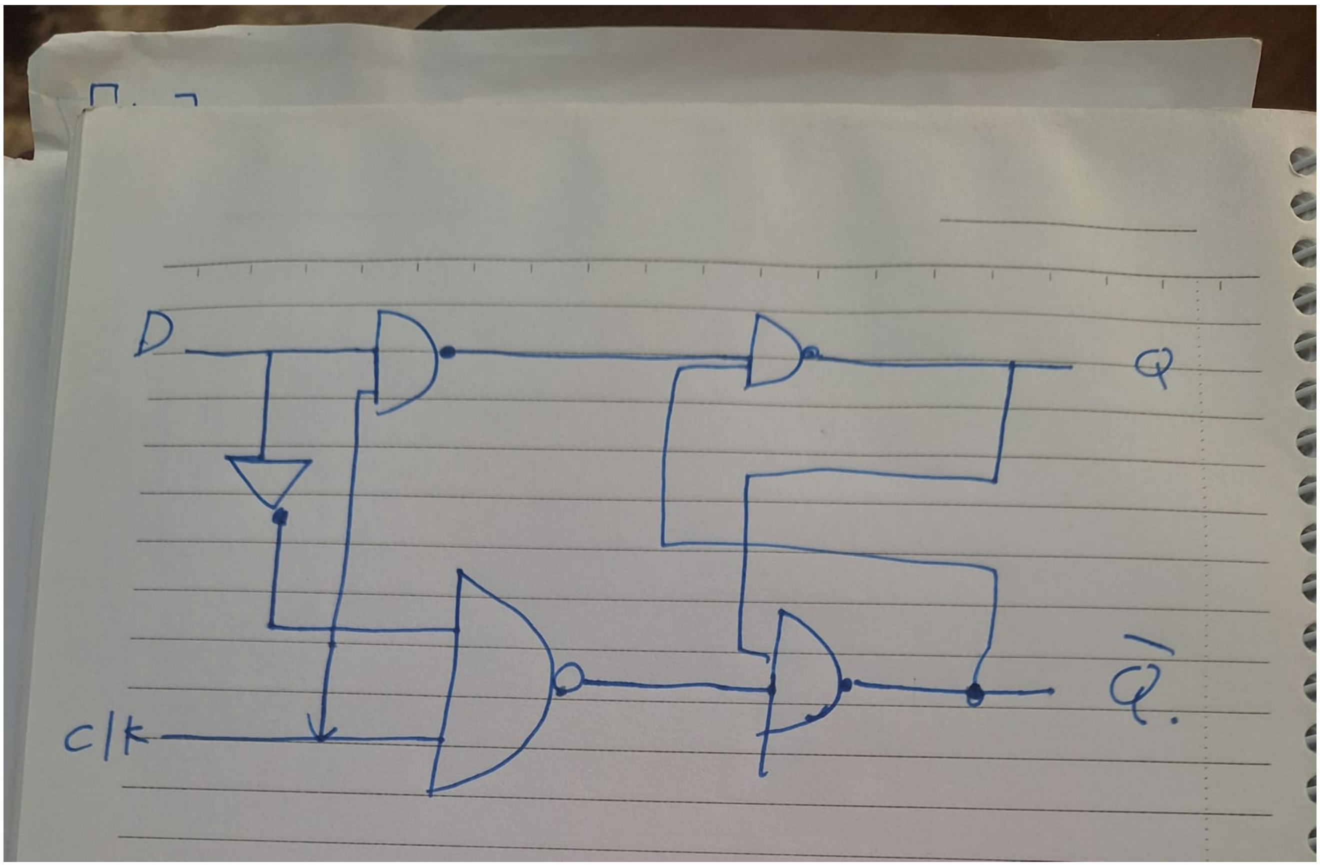


how it works

S	R	QT	Not QT
0	0	0	1
0	1	0	1
1	0	1	0
1	1	infinity	infinity

note: qt depends on the value of S, and there is no way both S and R are 1

Delay Circuit



D	QT	Not QT
0	0	1
1	1	0

it basically gives the value of D, however (hence the name), it delays the input