Ch.4 Combinational circuits

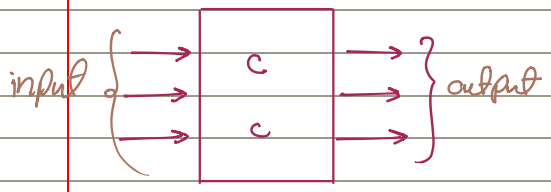
By: Rawan Alfares

Comparison between combinational and Sequential circut

output only depends on the present input.

ex:- adder

$$\begin{array}{r} 1 \\ +\ 0 \\ \hline 1 \end{array}$$



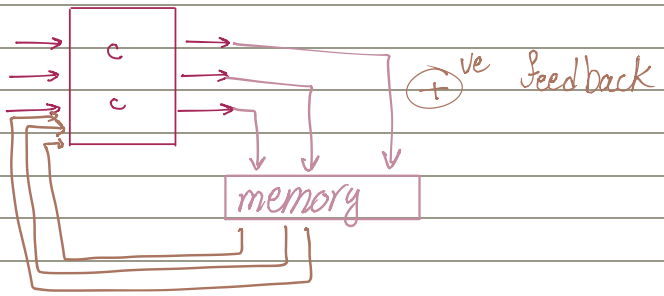
output depends on the present input as well as on previous output.

ex. Counter

⇓

it just increment one to the previous output.

* depends on flip flop memory.



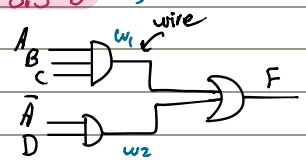
combinitional circuts (design)

* two types of digital circuts :-

1) Combinitional Circuts : output only depends on input.

2) Sequential Circuts :- outputs depends on inputs and previous output.

Analysis :- →



$w_1 = ABC$

$w_2 = \bar{A}D$

$F = w_1 + w_2$

* السيركت قدامك ، وبدك تكتب الآوتبوت بدلالة الإنبوت

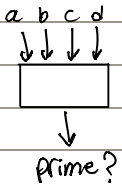
expression — truth table

Design procedure :-

1) أفهم الوصف المطلوب السيركت.
2) أحدد عدد الإنبوت والآوتبوت.
3) برسم K-map + truth table لكل output.
4) أصمم السيركت.
5) أتحقق من السيركت إذا بتشتغل صح.

Ex. Design a combinational Circuts that takes 4 bit input number, which check if the number is prime.

1)



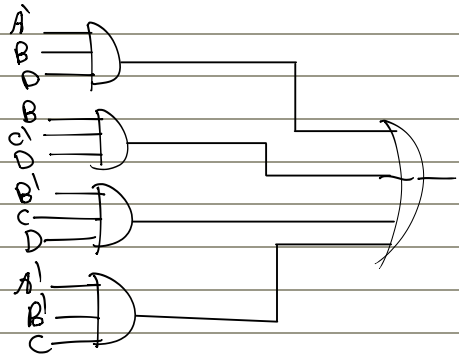
2)

a	b	c	d	P
0	0	0	0	0
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	0
1	1	1	1	0

3)

AB \ CD	00	01	11	10
00			1	1
01	0	1	1	0
11	0	1	0	0
10	0	0	1	0

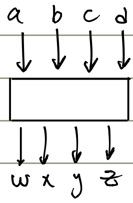
$F = \bar{A}BD + B\bar{C}D + B\bar{C}D + A\bar{B}C$



* بما إنه فقط (one output) فذلك بنعمل (one K-map)

Ex. design a BCD to exess-3 Code Conversion (always 4 bits)

①



②

a	b	c	d	w	x	y	z
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0
x	x	x	x	x	x	x	x
x	x	x	x	x	x	x	x
x	x	x	x	.	.	.	.
x	x	x	x	.	.		.
x	x	x	x			.	.
x	x	x	x	.	.	.	.

لكل Kmap لحال output

③ K-MAP

w

AB \ CD	00	01	11	10
00				
01		1	1	1
11	x	x	x	x
10	1	1	x	x

$w = BD + AC' + BC$

$= A + B(C+D)$

x

AB \ CD	00	01	11	10
00	0	1	1	1
01	1	0	0	0
11	x	x	x	x
10	0	1	x	x

$F = BC'D' + B'D + B'C$

$= BC'D' + B'(C+D)$

$= B.(C+D)' + B'(C+D)$

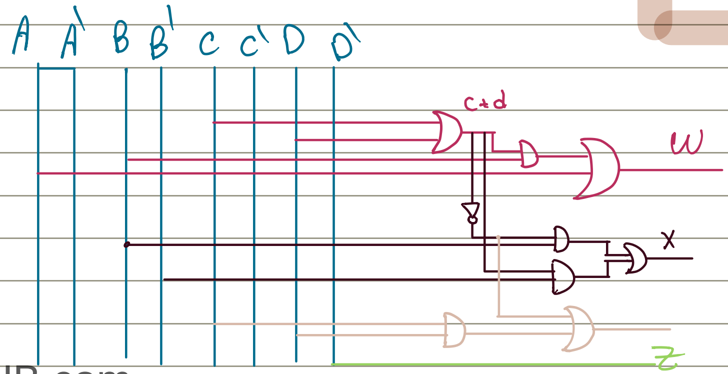
AB \ CD	00	01	11	10
00	1		1	
01	1		1	
11	x	x	x	x
10	1		x	x

$y = C'D' + CD$

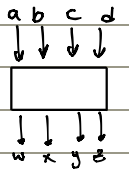
$= (C+D)' + CD$

AB \ CD	00	01	11	10
00	1			1
01	1			1
11	x	x	x	x
10	1		x	x

$Z = D'$



Ex. design a Combinational Circuit generate the 9's comp of a BCD digit?



②

a	b	c	d	w	x	y	z
0	0	0	0	1	0	0	1
0	0	0	1	1	0	0	0
0	0	1	0	0	1	1	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	0	1
0	1	0	1	0	1	0	0
0	1	1	0	0	0	1	1
0	1	1	1	0	0	1	0
1	0	0	0	0	0	0	1
1	0	0	1	0	0	0	0
x	x	x	x	x	x	x	x
x	x	x	x	x	x	x	x
x	x	x	x	x	x	x	x
x	x	x	x	x	x	x	x
x	x	x	x	x	x	x	x
x	x	x	x	x	x	x	x

Z

AB \ CD	00	01	11	10
00	1			1
01	1			1
11	x	x	x	x
10	1		x	x

$Z = D'$

y = C

AB \ CD	00	01	11	10
00			1	1
01			1	1
11	x	x	x	x
10			x	x

X

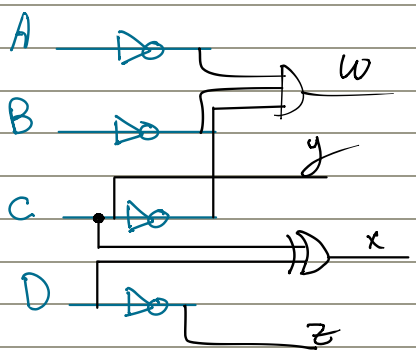
AB \ CD	00	01	11	10
00			1	1
01	1	1		
11	x	x	x	x
10			1	1

$X = D'C + DC'$

$= D \oplus C$

$W = AB'C'$

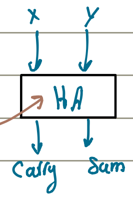
AB \ CD	00	01	11	10
00	1	1		
01				
11	x	x	x	x
10			x	x



binary adder & subtraction

1. Half adder

: Combinational Circuit that perform two bits.



$S = x \oplus y$
$C = x \cdot y$

السيركت الحالية فيهم بتكون موجودة داخل ال box.

* دايمًا بينتج عن الجمع carry and sum

x	y	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Sum

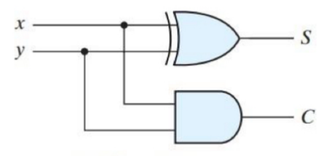
x \ y	0	1
0		1
1	1	

$= x \cdot y' + x'y$
$= x \oplus y$

Carry

x \ y	0	1
0		
1		1

$= x \cdot y$



(b) $S = x \oplus y$
$C = xy$

ميزة ⇐ HALF Adder (باختصار)

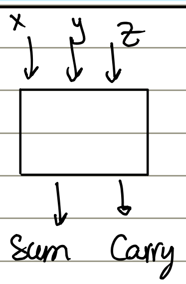
- XOR مكونة من AND
- 2 inputs
- 2 outputs (sum & carry).

* Single bit no's.
* it doesn't take carry from the previous sum.

---

2. FULL adder

: Combinitional Circuit perform addition of 3-bits.



(z = carry input)

X	y	z	Sum	Carry
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

x \ yz	00	01	11	10
0		1		1
1	1		1	

$Sum = xy'z' + x'y'z + xyz + x'yz'$
$= x'(y'z + yz') + x(y'z' + yz)$
$= x'(y \oplus z) + x(y \oplus z)'$
$= x'w + xw'$
$= x \oplus w = x \oplus y \oplus z$

x \ yz	00	01	11	10
0			1	
1		1	1	1

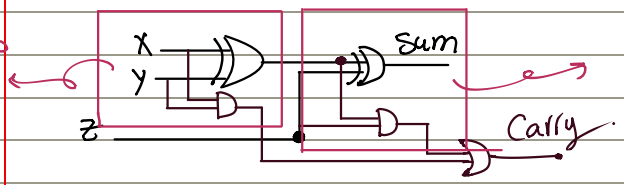
$Carry = xz + xy + yz$
$= xy'z + x'yz + xyz + xyz'$
$= z(xy' + x'y) + xy(z + z')$
$= z(x \oplus y) + xy$

or $= xy + xy'z + x'yz$
$= xy + z(x \oplus y)$

طريقة أسهل

Full adder using Half Adder

XOR داخلها AND و inputs نفس ال Half adder فاعتبرناك



* full Adder
  ↳ two Half adder and one OR gate

Ex. design the following circuits :- 2x+1

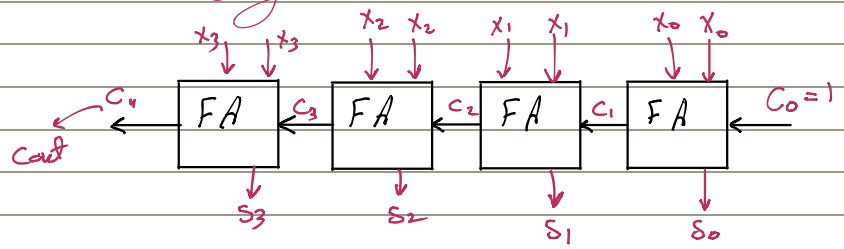
Ans:-

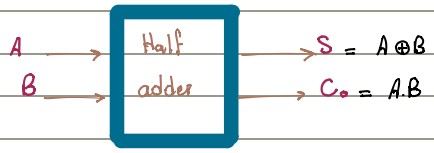
$X_3\ X_2\ X_1\ X_0$

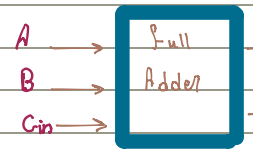
$X_3\ X_2\ X_1\ X_0$ +

1 +

as a Carry

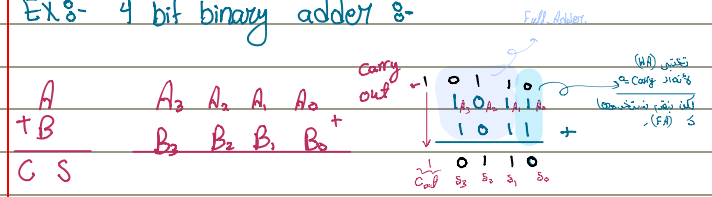




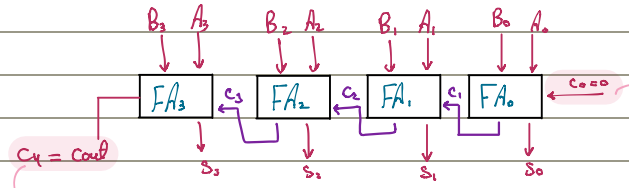


binary adder: Addition Any bits

Ex:- 4 bit binary adder :-



* جمعنا [4 bit] , فلذلك لزمني [4 Full Adders].



the least significant bit Carry is always fixed to be 0.

Carry out will be of Cell (i+1)

* this circuit Called Ripple Carry Adder because each Carry out in Cell i Will be Carry in at Cell i+1.

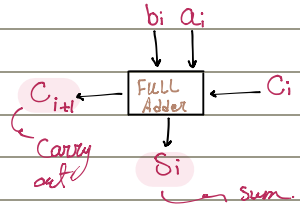
NOTE :-

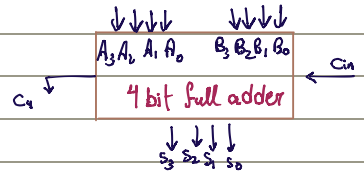
1. Carry out of Cell i will become Carry in at Cell (i+1).
2. Addition of n-bit numbers requires :-
   - A. ← A chain of n- full adders
   - B. ← A chain of one Half adder, and (n-1) full Adder.

* بالخطأ :-

(bits) إما بنستخدم عدد (FA) بنفس عدد الـ
(n-1) FA والباقي (HA) بدي أول واحد بنستخدم

3. The Carry propagates through the full

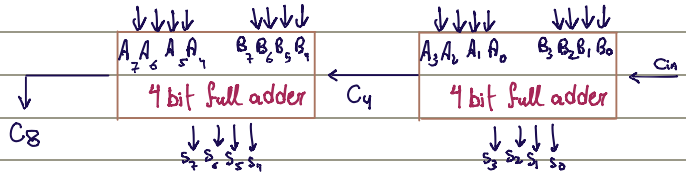


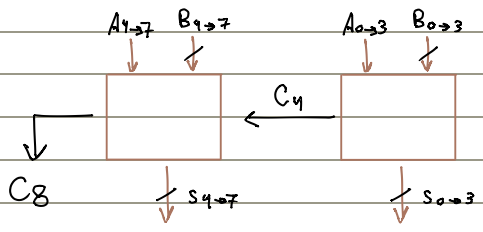


Adding 8-bit full adder

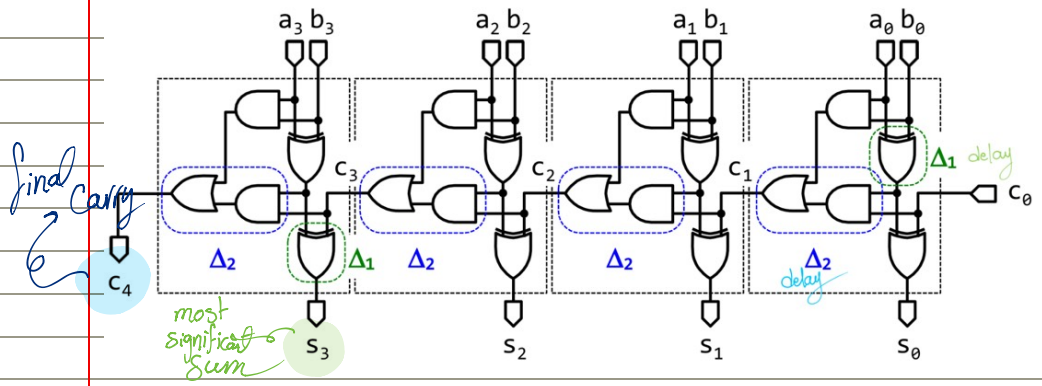
{8 full adder ← نحتاج 8 bit} *

(two of 4 bit full adder) * أو نستخدم





Longest Delay Analysis



Final Carry

* suppose that And-OR $\Rightarrow \Delta_2$
* Suppose that XOR $\Rightarrow \Delta_1$

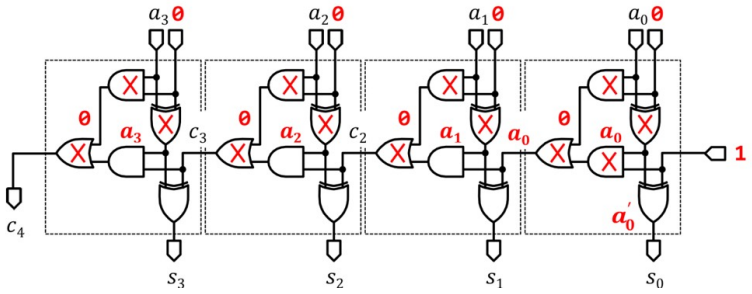
* for N-bit Ripple Carry adder, if all inputs are represents at once :-
  1. Most significant Sum bit delay = $3\Delta_2 + \Delta_1$ → ingeneral, $(n-1)\Delta_2 + \Delta_1$
  2. Final Carry out delay = $4\Delta_2 + \Delta_1$ → in general, $n\Delta_2 + \Delta_1$

Incrementor Circuit

$Sum = A + B$, $C_0 = 1$

$B = 0$

(A) بما أنه (B=0) أي نجمع A مع رقم ثابت ، فلذلك سنقدم نبسط الـ Circuit أكثر.

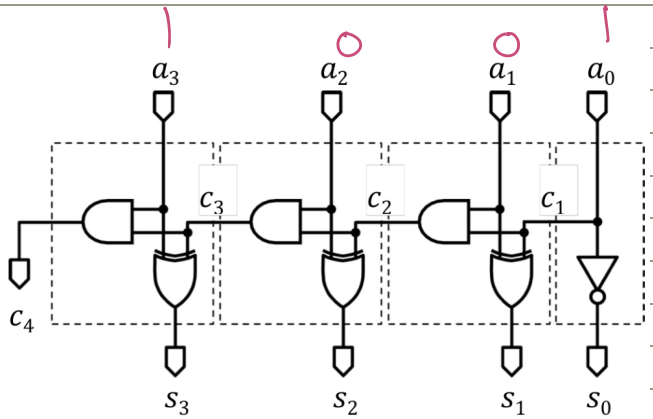


B=0

Sum = $A \oplus B \oplus C_{in} \Rightarrow A \oplus C_{in}$

Carray = $A.B + (A \oplus B).C_{in}$

$A.0 + (A \oplus 0).C_{in} \rightarrow A.C_{in}$



design Contraction :-

لما نجمع لقيمة ثابتة

ex :- 1. incrementor
2. Comparator

Carry look Ahead Adder

A	B	C_{in}	C_o	
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	✓

A⊕B & Cin

A.B

$C_o = A.B + (A \oplus B)C_{in}$

Carry generator (G) — ما بتعتمد على Carry

Carry propagation (P) — بس بتعتمد على الـ Carry

$C_o = G + P.C_{in}$

generally :- $C_{in} = G_i + P_i C_i$.

ex. 4 bit Carry look Ahead adder.

$C_1 = g_0 + P_0 C_0$

$C_2 = g_1 + P_1 C_1 = g_1 + P_1 g_0 + P_1 P_0 C_0$

$C_3 = g_2 + P_2 C_2 = g_2 + P_2 g_1 + P_2 P_1 g_0 + P_2 P_1 P_0 C_0$

$C_4 = g_3 + P_3 C_3 = g_3 + P_3 g_2 + P_3 P_2 g_1 + P_3 P_2 P_1 g_0 + P_3 P_2 P_1 P_0 C_0$

G.G group generation

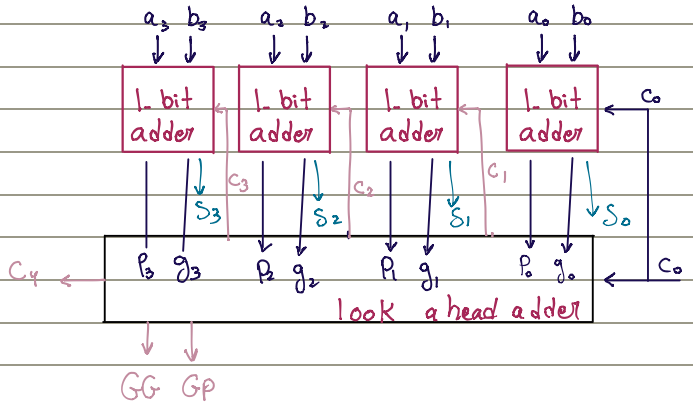
GP group propagation.

$C_4 = GG + GP.C_0$

* Carry doesn't ripple any more.
* Reduced delay.

⇒ يعني كل العدد منها إنها تختلف وقت وما تختلي الـ [Carry] يعتمد على اللي قبله.

* Calculate Carries in Parallel.



* every look Ahead Carry produced → GG, GP, C_{i+1}

* 16 bit Carry look ahead adder → $\frac{16}{4} = 4$, implement it by using 4 CLA 4bit

64 " " " " → $\frac{64}{16} = 4$, " " " " 4 CLA 16 bit

Binary Subtractor

$A - B = A + 2\text{'s Comp}$

$= A + 1\text{'s comp } B + 1$

① $= 0 \oplus X$

$= X$

② $= 1 \oplus X$

$= X'$

invert x by using XOR gate.

Ex. 4bit $A - B = A + 2\text{'s comp } B$

C_4 C_3 C_2 C_1

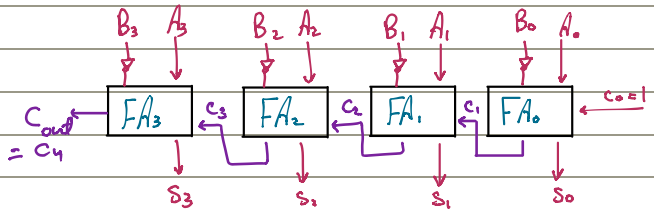
A_3 A_2 A_1 A_0

B_3' B_2' B_1' B_0'

① + ? → 2's comp → C_0

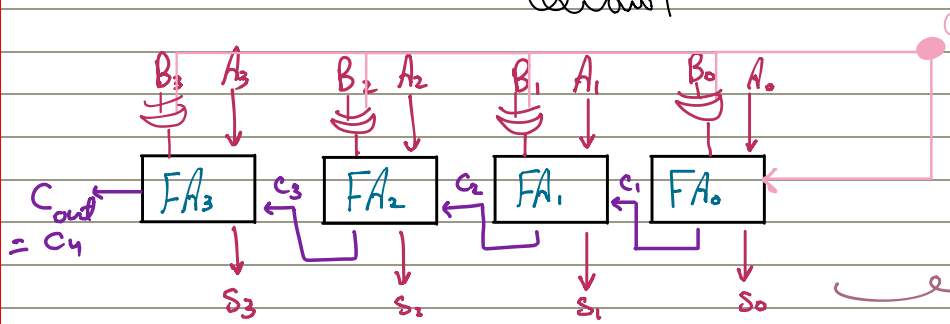
C_{out} S_3 S_2 S_1 S_0

Subtractor



لكن في الجمع ما بكون معك
وإنما بكون سيرتك وحدة
مرة بنعمل (add) مرة (sub)

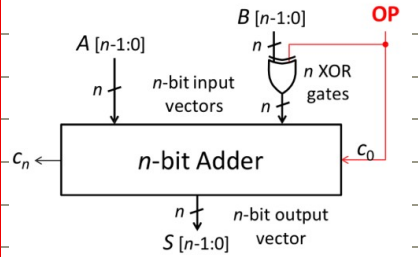
بسمتها!



1. op = 0 [Add] → Cin = 0
2. op = 1 [Sub] → Cin = 1

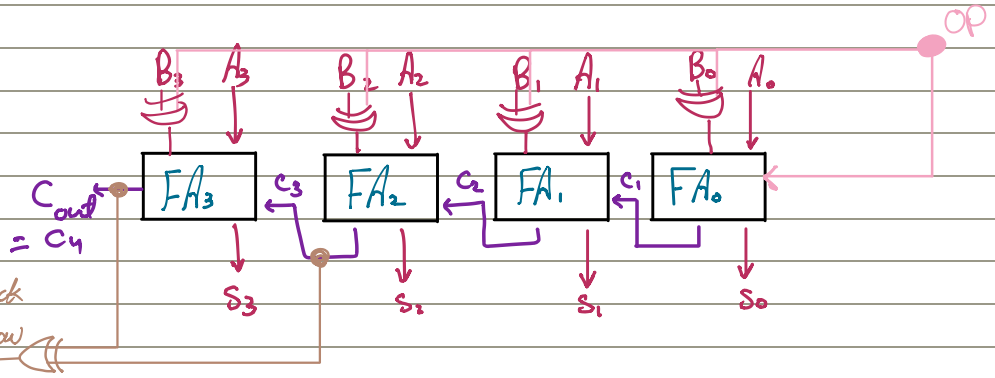
only for unsigned number.

In general :-



Do both addition And subtraction

[over flow] يعني من المحتمل حدوث (Signed) بما إنهم



إذا أخذ (two carry) منتابعات no overflow
مختلفات ← overflow

unsigned , Range, 0 → 15
Signed , Range -8 → +7

Ex: given that X is unsigned number.
and y is Signed number.

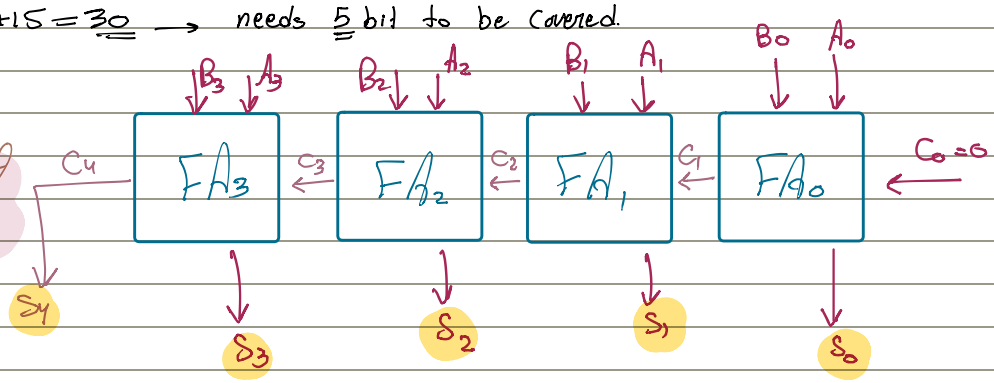
X = 1101 (binary) then X = 13 decimal.
y = 1101 (binary) then y = -3 decimal.
0010

إذا بدنا نكبّر الرقم
unsigned ✗ → zero-extended
Signed ✗ → Sign-extended

* if X is zero extended from 4 to 6 bit → X = 001101 = 13
if y is sign extended from 4 to 6 bit. → y = 111101 = -3
(sign extention)

unsigned addition x+y

Design a Circuit that Computes S = X+y
if X and y are 4 bit, Range 0 → 15
* min = 0+0 = 0
* max = 15+15 = 30 → needs 5 bit to be covered.



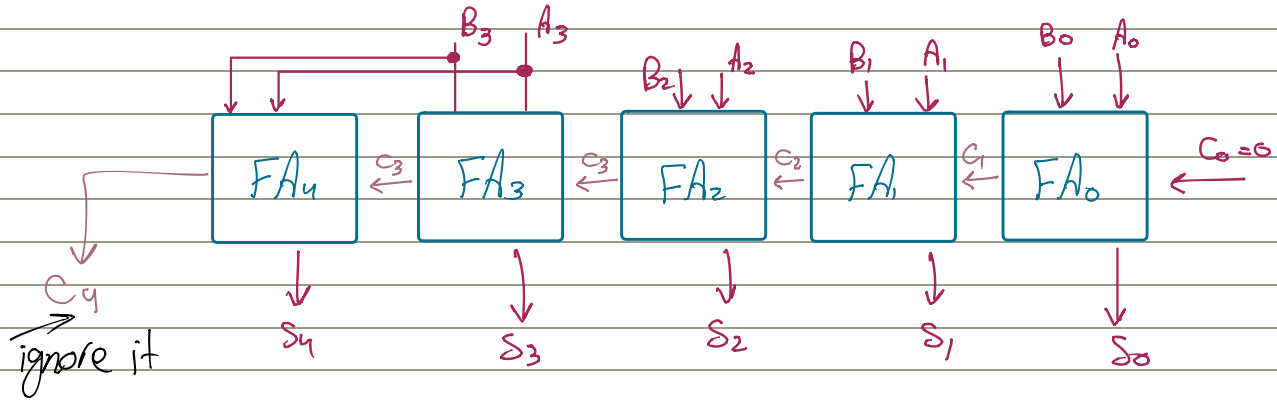
* Most Significant Sum bit S_4 is the Carry bit C_4

5bit.

signed addition x+y

Design a Circuit that Computes S = X+y (Signed X and y).
if X[3:0] and Y[3:0] are bit signed integers → Range -8 → 7.

min = -8 + -8 = -16
max = 7+7 = 14
$-2^{n-1} \rightarrow +2^{n-1}-1$
n = 5
$-2^{4} \rightarrow +2^{4}-1$
-16 → 15



* دائماً قبل ما أعمل (design) للسيركت لازم أعرف شو أكبر عدد ممكن الـ input وبشوف كم (bit) بيلزم أعمل.

unsigned subtraction s=x-y

Design a circuit that computes S=X-y (unsigned X and y).

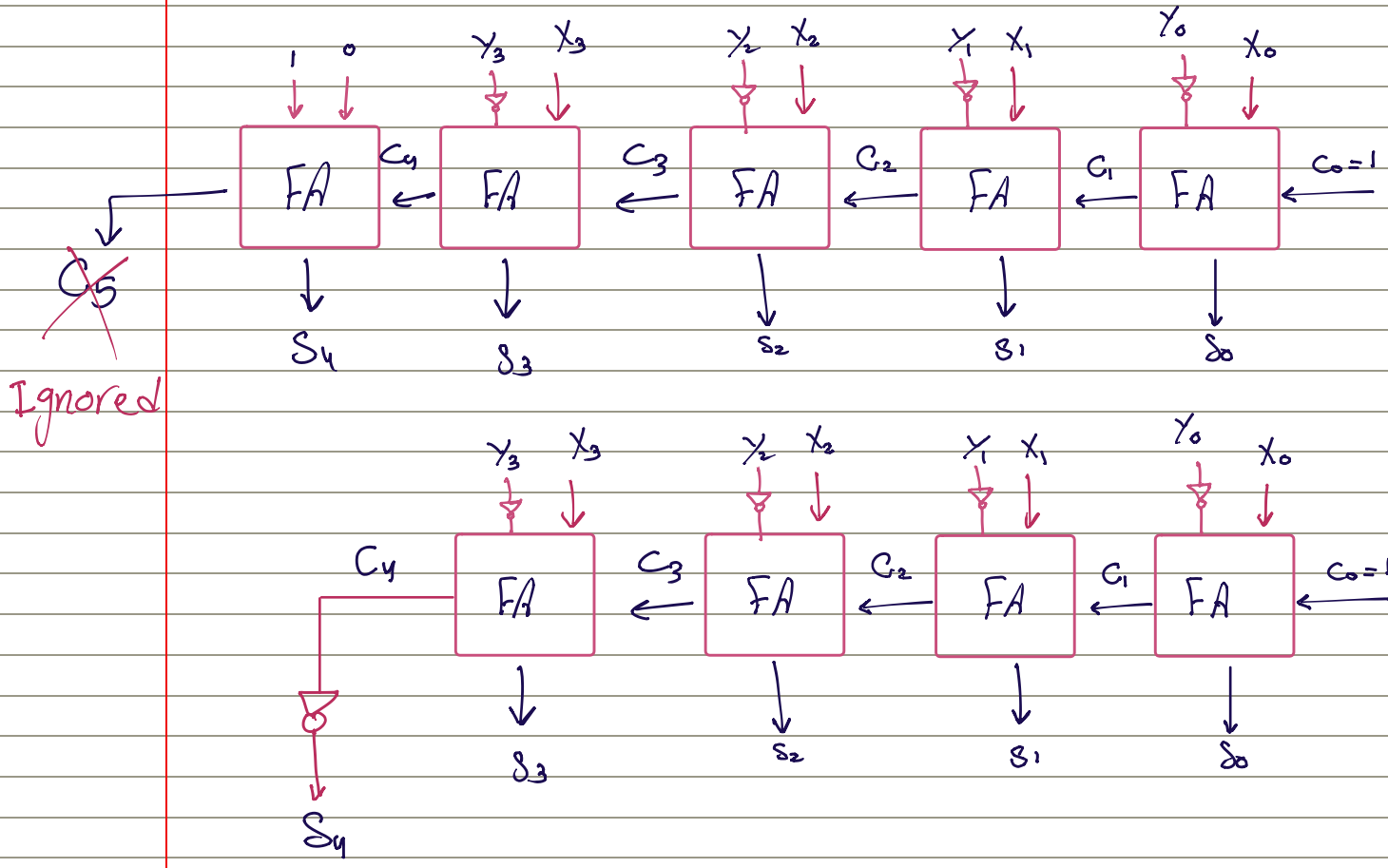
X : [0:3] , and y [0:3]

X: 0 → 15 , Y: 0 → 15 → S : 0 → 30

↳ 5 bit

Sol :- X - y = x + 2's comp y

= x + 1's comp y + 1

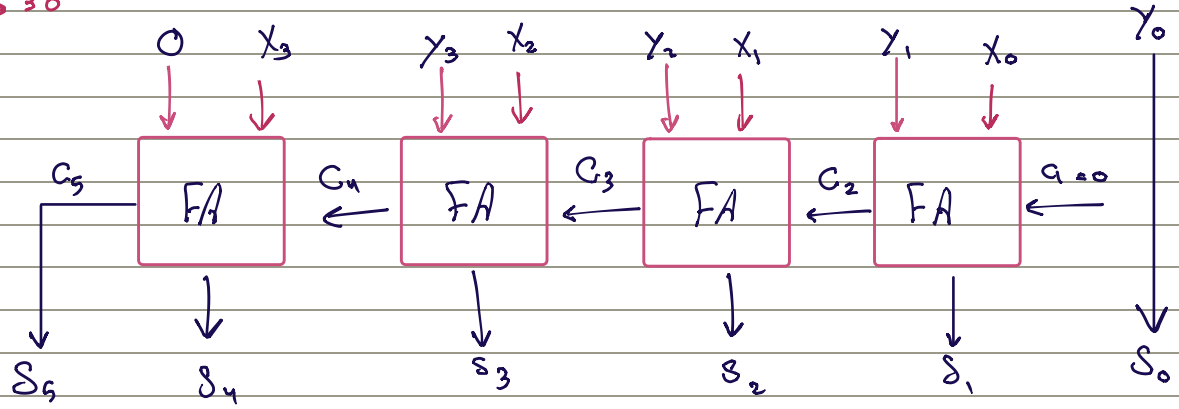


S=2*X+Y unsigned x and y

X[3:0] and y[3:0] → S[5:0]

2 x (0 → 15) + 0 → 15 → 0 → 45 → Covered with 6 bits.

0 → 30



Ex : X = 101 → 2X = 1010 ← shifting by 1

S=2*X+Y signed x and y

X[3:0] , y[3:0]

2X[6:0]

x & Range
-8 → +7
-16 → +14

Range y &
-8 → +7

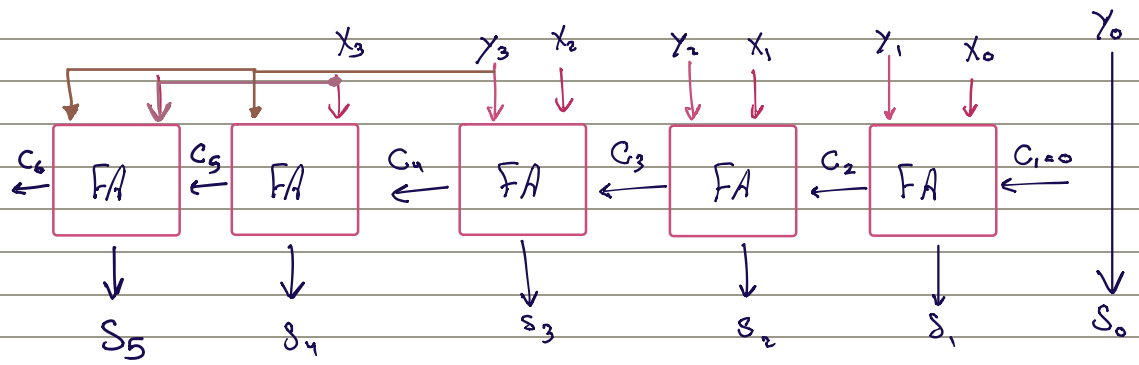
S[5:0]

$-2^{n-1} \rightarrow +2^{n-1}-1$

$-24 \rightarrow 21$

$n = 6$

Sign extends.



more examples-

1. Design a circuit for unsigned $S = X+Y+Z$.
2. Design a circuit for signed $S = W+X-Y-Y-Z$.
3. Design a circuit for absolute difference. $|X-Y|$, of signed X and Y.

BCD Addition

باختصار :- بجمع عادي ولكن إذا طلع الرقم (الرقم > 9) بجمع 6.

output Sum 8 9+9+1 (Carr from previous digit.)

we use a 4 bit binary adder to add the BCD digits.

* the adder will produce a result that ranges from 0 through 19
* if the result is more than 9, it must be corrected. to use 2 digits
* to Correct the digit, add 6 to the digit Sum (a-4bit binary adder).

لما يكون C=0 يعني بدي أجمع صفر للأرقام.

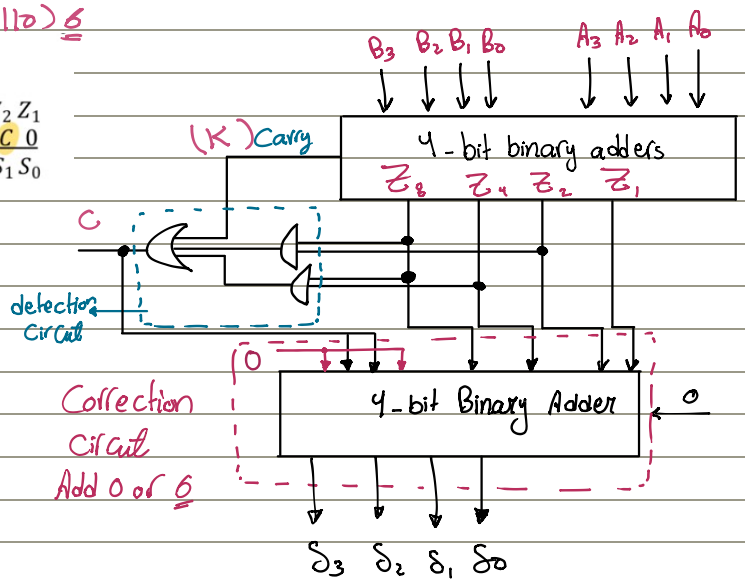
لما يكون C=1 بجمع 6 (0110)

$C = 0$

$$\begin{array}{r} Z_8\ Z_4\ Z_2\ Z_1 \\ +\ 0\ 0\ 0\ 0 \\ \hline S_3\ S_2\ S_1\ S_0 \end{array}$$

$C = 1$

$$\begin{array}{r} Z_8\ Z_4\ Z_2\ Z_1 \\ +\ 0\ 1\ 1\ 0 \\ \hline S_3\ S_2\ S_1\ S_0 \end{array} \Rightarrow \begin{array}{r} Z_8\ Z_4\ Z_2\ Z_1 \\ +\ 0\ C\ C\ 0 \\ \hline S_3\ S_2\ S_1\ S_0 \end{array}$$



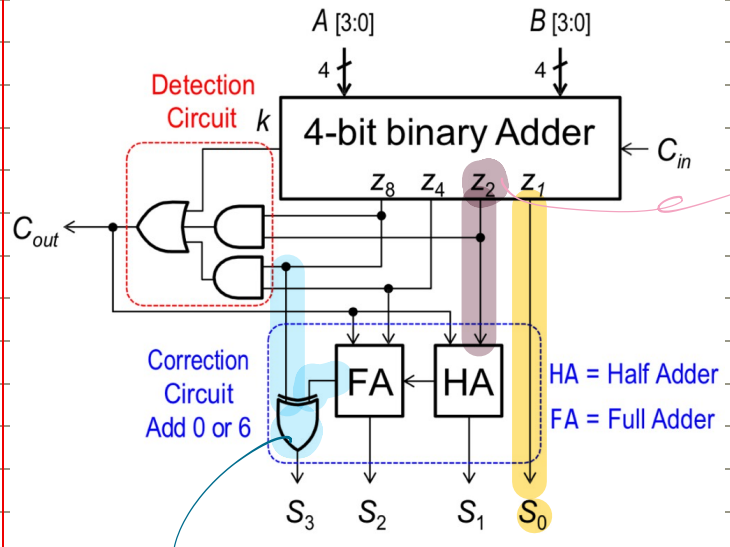
اللي بيحدد إذا أجمع أو لا.

* Correction required if :-
  1. $Z > 9$
  2. $K = 1$

$C = K + Z_8 Z_2 + Z_8 Z_4$

Z_8Z_4 \ Z_2Z_1	00	01	11	10
00				
01				
11	1	1	1	1
10			1	1

$= Z_8 Z_2 + Z_8 Z_4$



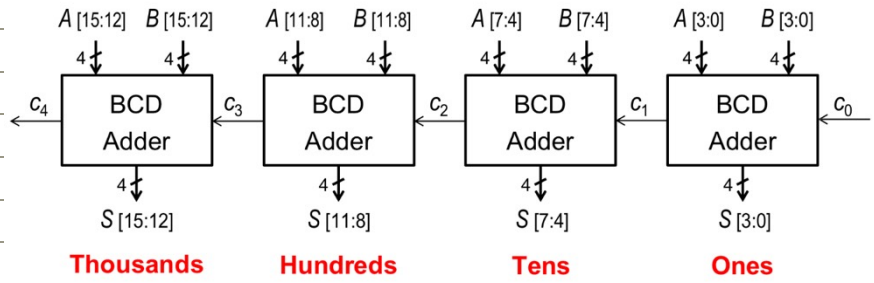
Won't be exist any Carry, so we can use Half Adder.

$$\begin{array}{r} Z_8\ Z_4\ Z_2\ Z_1 \\ 0\ 1\ 1\ 0 \\ + \\ \hline S_4\ S_3\ S_2\ S_1 \end{array}$$

same

نوع البسيط XOR :- (HA)

Ripple Carry BCD adder. → 2905 + 1897 — إذا جمعنا عددين



Binary multiplication

0x1=0 , 0x0=0 , 1x0=0 , 1x1=1 } as AND gate.

multiplicand 1100
multiplier 1101 x
0000
0000 +
1100 +
1100 +
product 10011000

* n-bit multiplicand x m-multiplier = (n+m)-bit product.

2-bit X 2-bit binary multiplier

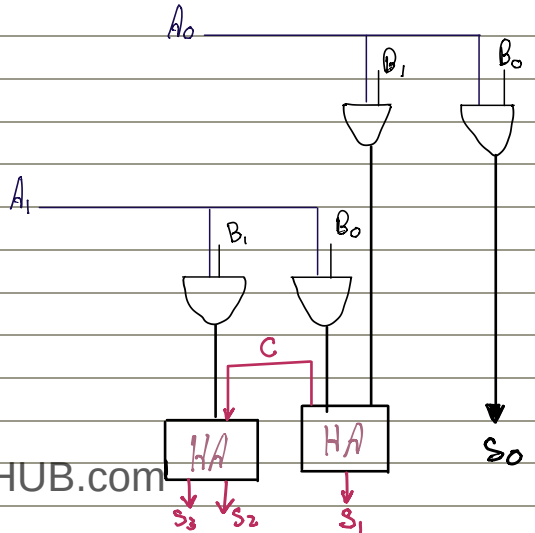
* AND gate Requires (n x m)

$B = B_1B_0$, $A = A_1A_0$

# of AND gates $n \times m = 2 \times 2 = 4$ AND gates

B_1 B_0
A_1 A_0 x
B_1A_0 B_0A_0
0 B_0A_1 + } = 4 AND gates

$m \times n = 2 \times 2 = 4$ (AND gate) عدد



4-bit * 3-bit binary multiplier

$B = B_3B_2B_1B_0$, $A = A_2A_1A_0$

Ans: 4+3 = 7 bit

AND gate 8-3x4 = 12 AND gate

$B_3\ B_2\ B_1\ B_0$

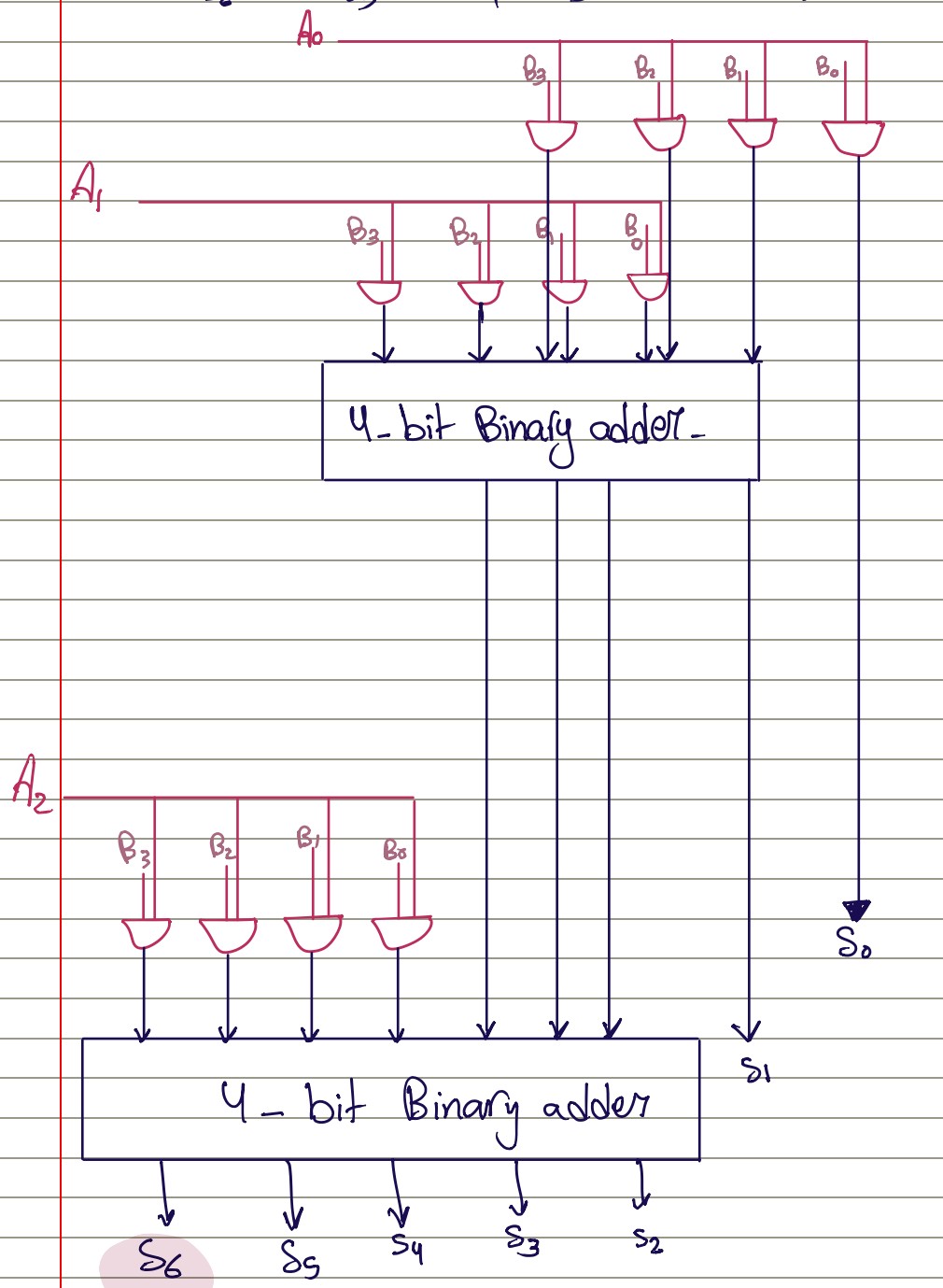
$A_2\ A_1\ A_0$ X

$A_0B_3\ \ A_0B_2\ \ A_0B_1\ \ A_0B_0$

$A_1B_3\ \ A_1B_2\ \ A_1B_1\ \ A_1B_0$ +

$A_2B_3\ \ A_2B_2\ \ A_2B_1\ \ A_2B_0$ +

$S_6\ \ S_5\ \ S_4\ \ S_3\ \ S_2\ \ S_1\ \ S_0$



magnitude comparator

* input :- 2
* output :- 3 → A=B, A>B, A<B

* 4-bit magnitude Comparators

$A = A_3 A_2 A_1 A_0$
$B = B_3 B_2 B_1 B_0$

لا يراح نحلها على K-map ما بتخلص، فلذلك نستخدم المنطق

① $A = B$

↳ $A_3 = B_3$ and $A_2 = B_2$ and $A_1 = B_1$ and $A_0 = B_0$

$A = B \rightarrow E_3 E_2 E_1 E_0 = 1$

$EQ = E_3 E_2 E_1 E_0 = 1$

② $A > B$

↳ $A_3 > B_3$ or $[A_3 = B_3$ and $A_2 > B_2]$ or $[A_3 = B_3$ and $A_2 = B_2$ and $A_1 > B_1]$ or $[A_3 = B_3$ and $A_2 = B_2$ and $A_1 = B_1$ and $A_0 > B_0]$

$GT = G_3 + E_3 G_2 + E_3 E_2 G_1 + E_3 E_2 E_1 G_0$

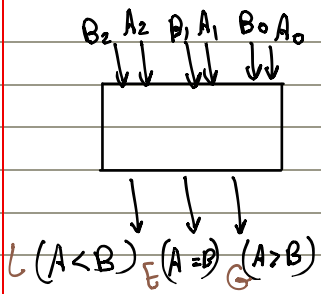
③ $A < B$

$LT = L_3 + E_3 L_2 + E_3 E_2 L_1 + E_3 E_2 E_1 L_0$

↳ بنقدر نحكي إنها الكومبلمنت لـ EQ + GT

$LT = (EQ + GT)'$

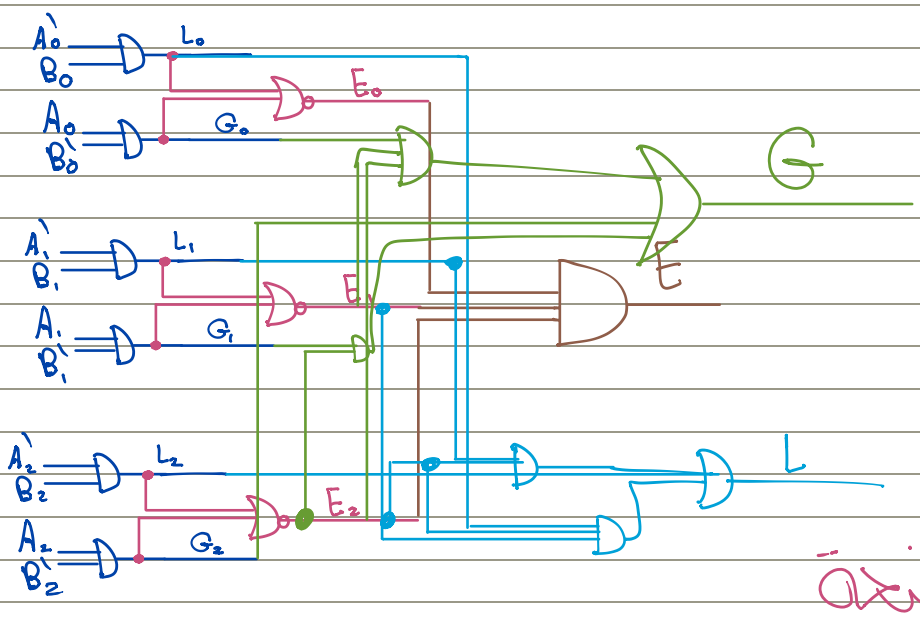
Draw 3bit magnitude Comparator :-



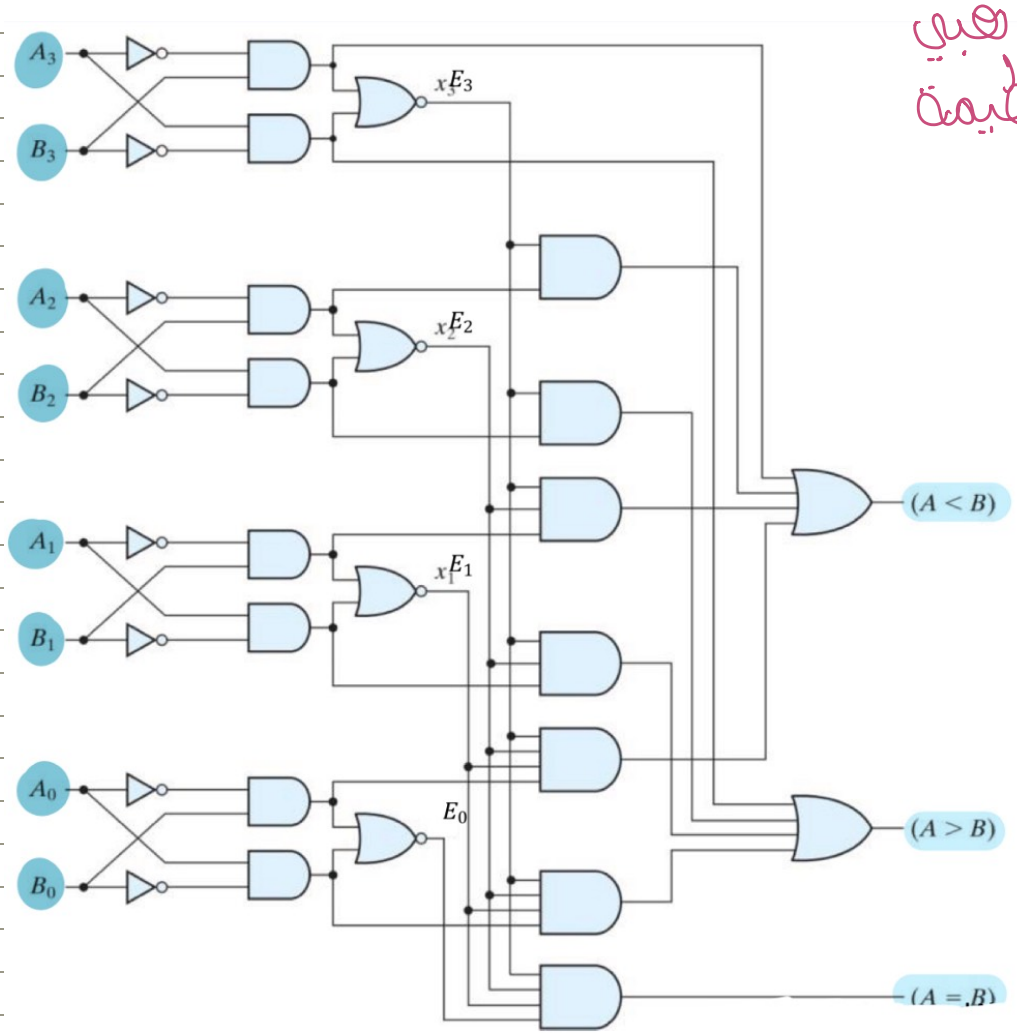
$LT = L_2 + E_2L_1 + E_2E_1L_0$

$GT = G_2 + E_2G_1 + E_2E_1G_0$

$EQ = E_2 \cdot E_1 \cdot E_0$



رسمة حلوة
عشان معاهبي
بالرسم غبية



Design a 1 bit magnitude Comparator ?

A_0	B_0	E	L	G
0	0	1	0	0
0	1	0	1	0
1	0	0	0	1
1	1	1	0	0

E

A_0 \ B_0	0	1
0	1	
1		1

$A'B' + AB$

Greater

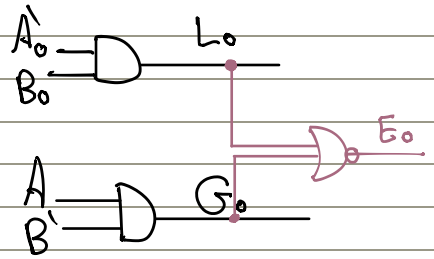
A_0 \ B_0	0	1
0		
1	1	

AB'

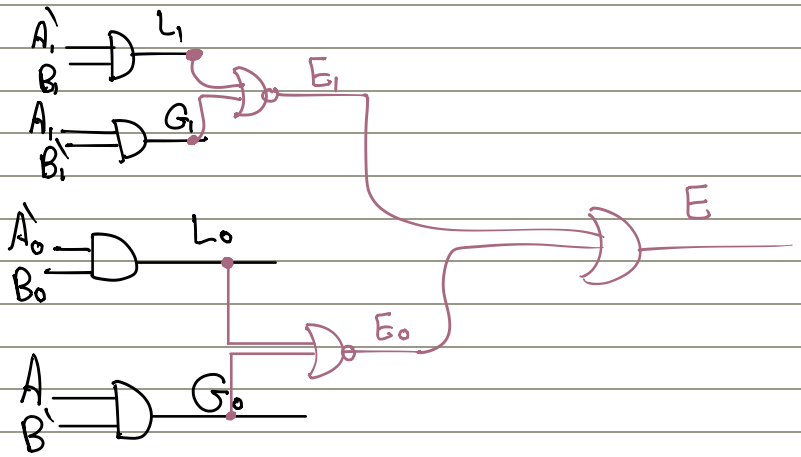
Less

A_0 \ B_0	0	1
0	0	1
1	0	0

$A'B$

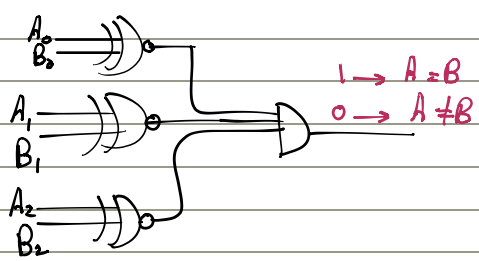


* Design 2 bit magnitude Comparator ?



* Design a 3bit magnitude equality Comparator ?

$A_0 = B_0$ && $A_1 = B_1$ && $A_2 = B_2$ ⟶ (E = 1)



إذا كان B = A
الجواب = 1
عن طريق
XNOR

2.

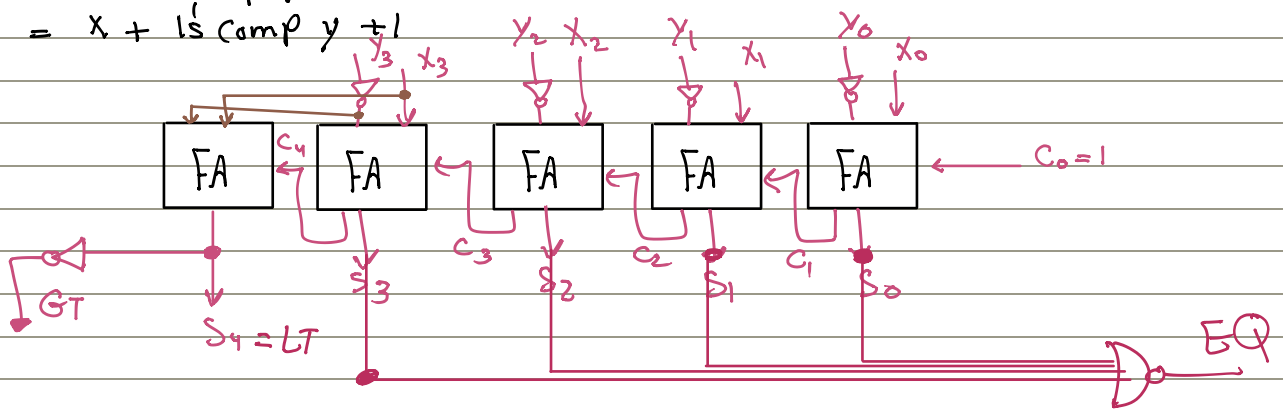
signed Less Than LT = X<Y

$X < y \rightarrow$ signed $\rightarrow$ 4 bits.

if $X < y$, then $x - y < 0$
if $x = y$, then $x = y$

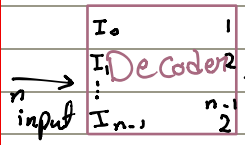
$= X +$ 2's comp y
$=$ x + 1's comp y +1

$-8 \rightarrow +7$
$-8 \rightarrow +7$
$-16 \rightarrow +14$
→ required 5 bit



Decoder

a combinational circuit that has n inputs and 2^n output.

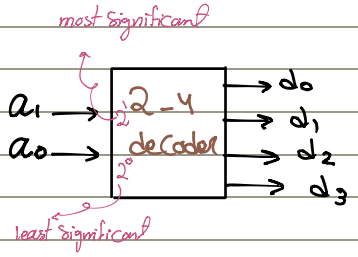


2^n outputs (minterms or maxterms).

2	4
3	8
4	16
5	32
⋮	⋮

EX: Design a 2-4 Decoder

* input : 2 , X , Y
* output : 4 d_0, d_1, d_2, d_3



most sign. ← least sign.

a_1	a_0	d_0	d_1	d_2	d_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

* كإنه بدخل عليه binary و فقط بيطلعلي عن قيمة الـ decimal تاعته.

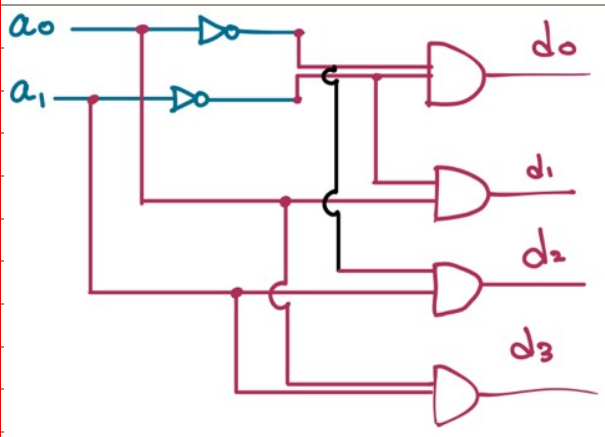
لو بنلاحظ هون إنه هدول نفسهم الـ minterms.

$d_0 = m_0$, $d_1 = m_1$, $d_2 = m_2$

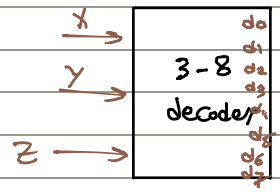
$d_3 = m_3$.

$d_0 = \bar{X}\bar{Y}$, $d_1 = \bar{X}Y$, $d_2 = X\bar{Y}$

$d_3 = XY$



EX. Design 3X8 decoder.



* input 3, x,y,z
* output 8, $d_0, d_1, \dots d_7$

Binary

Decimal	x	y	z	d_0	d_1	d_2	d_3	d_4	d_5	d_6	d_7
0	0	0	0	1	0	0	0	0	0	0	0
1	0	0	1	0	1	0	0	0	0	0	0
2	0	1	0	0	0	1	0	0	0	0	0
3	0	1	1	0	0	0	1	0	0	0	0
4	1	0	0	0	0	0	0	1	0	0	0
5	1	0	1	0	0	0	0	0	1	0	0
6	1	1	0	0	0	0	0	0	0	1	0
7	1	1	1	0	0	0	0	0	0	0	1

$d_0 = \bar{x}\bar{y}\bar{z}$

$d_1 = \bar{x}\bar{y}z$

$d_2 = \bar{x}y\bar{z}$

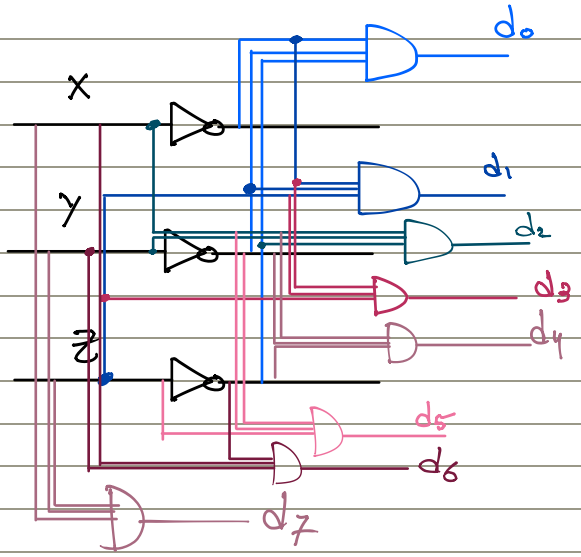
$d_3 = \bar{x}yz$

$d_4 = x\bar{y}\bar{z}$

$d_5 = x\bar{y}z$

$d_6 = xy\bar{z}$

$d_7 = xyz$



EX:- Implement the following function using decoder :-

$F_1(x,y) = \Sigma(0,3)$, $F_2(x,y) = (0,1,2)$, $F_3(x,y) = \Pi(0,3)$.

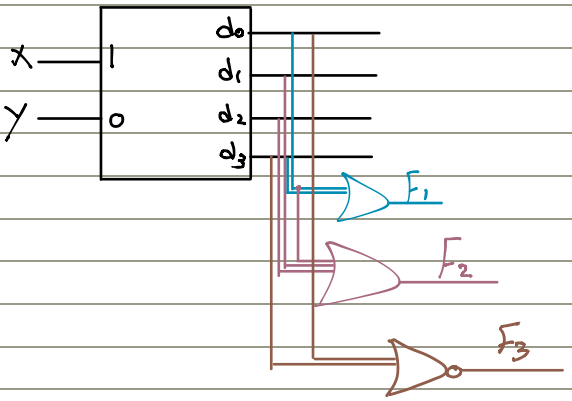
$F_1 = m_0 + m_3$
$= \bar{x}\bar{y} + xy$
Sum of minterms
OR gate

$F_2 = m_0 + m_1 + m_2$
$= \bar{x}\bar{y} + \bar{x}y + x\bar{y}$
SOM
OR gate

$\bar{F}_3 = \Sigma(0,3)$
$= m_0 + m_3$
$F_3 = \overline{(m_0 + m_3)}$ → complement of sum of minterms
using NOR gate.

* input :- 2 , x, y
* output :- 4 d_0, d_1, d_2, d_3

x	y	d_0	d_1	d_2	d_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	3



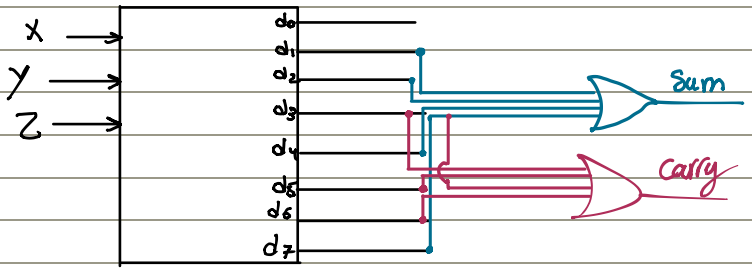
EX:- design Full Adder Sum $= \Sigma(1,2,4,7)$, Cout $= \Sigma(3,5,6,7)$. using decoder.

* input : 3 , x, y, z
* output, 8 , $d_0 \rightarrow d_7$

Sum $= m_0 + m_2 + m_4 + m_7$

Cout $= m_3 + m_5 + m_6 + m_7$

x	y	z	Sum	Cout
0	0	0	1	0
0	0	1	0	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



decoder with Enable

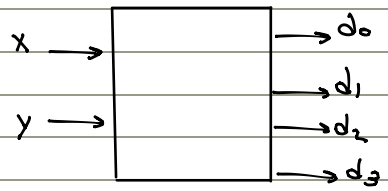
E → high level :- if Enable is Zero then all outputs are Zero regarless of the inputs (x,y).

- if Enable is one then the outputs are the minterms

→ low level :- if Enable is one all the outputs are zero regarless of the inputs.

Active high decoder :-

outputs are minterms.

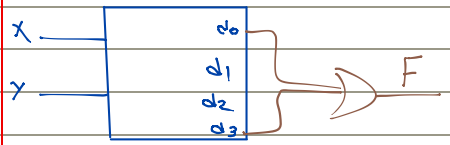


x y	d_0	d_1	d_2	d_3
0 0	1	0	0	0
0 1	0	1	0	0
1 0	0	0	1	0
1 1	0	0	0	1

$d_0 = \bar{x}\bar{y}$ $\quad d_2 = x\bar{y}$

$d_1 = \bar{x}y$ $\quad d_3 = xy$

EX:- $F(x,y) = \Sigma(0,3)$

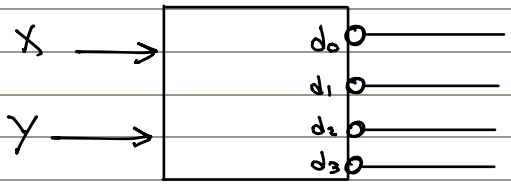


* we use for Implementation OR gate

2* To Implement by using maxterms we use AND "fewer Zeros".

Active low decoder :-

output are max terms.



x y	d_0	d_1	d_2	d_3
0 0	0	1	1	1
0 1	1	0	1	1
1 0	1	1	0	1
1 1	1	1	1	0

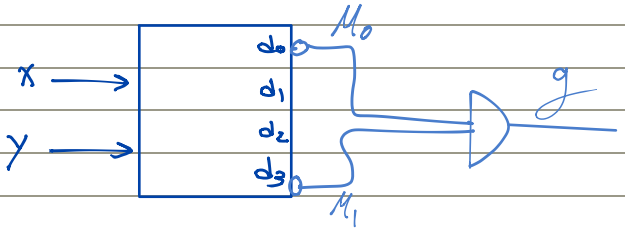
$d_0 = x+y = (\bar{x}.\bar{y})' = \overline{m_0}$

$d_1 = x+\bar{y} = (\bar{x}.y)' = \overline{m_1}$

$d_2 = \bar{x}+y = (x.\bar{y})' = \overline{m_2}$

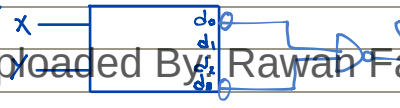
$d_3 = \bar{x}+\bar{y} = (xy)' = \overline{m_3}$

EX:- $g(x,y) = \Pi(0,3) = M_0 . M_3$



1* To Implement a Sum of minterms function Using active low, we just use NAND. "fewer ones"

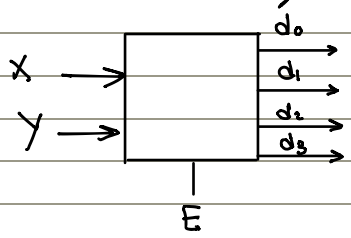
EX:- $F = \Sigma(0,3)$ = nor m_3



EX. Design 2-to-4 Decoder with Enable Input & Active high

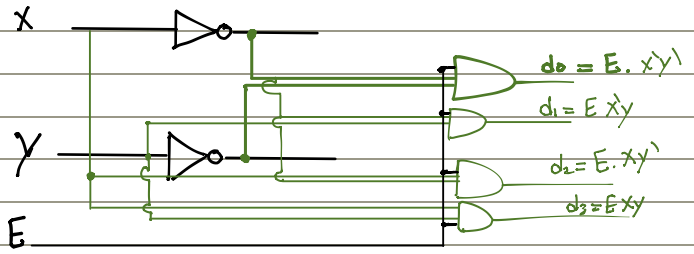
E	X	Y	d_0	d_1	d_2	d_3
0	X	X	0	0	0	0
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1

input = 3 , output = 4



* إذا كان (E=0) بتكون النظر عن قيمة x,y فبتعطي أصفار .

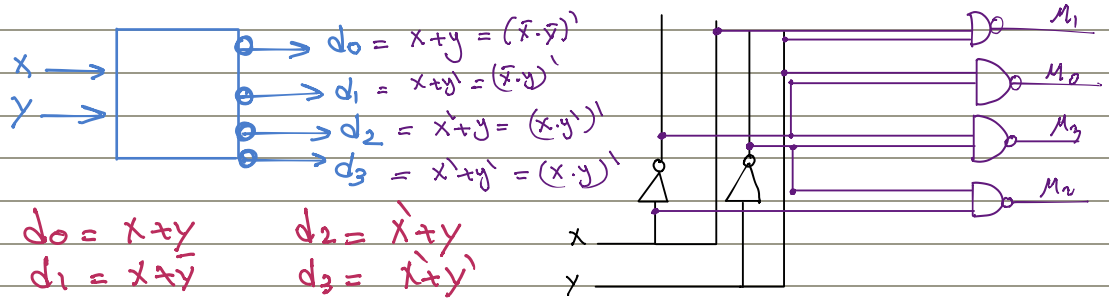
* ولما يكون (E=1) بتعطيني قبل



EX. Design 2x4 decoder "active low" & with Enable

maxterms ≡ ⊜

E	X	Y	d_0	d_1	d_2	d_3
0	X	X	1	1	1	1
1	0	0	0	1	1	1
1	0	1	1	0	1	1
1	1	0	1	1	0	1
1	1	1	1	1	1	0



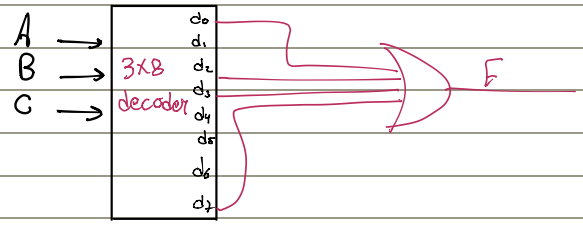
$d_0 = x+y$ $\quad d_2 = x'+y$

$d_1 = x+y'$ $\quad d_3 = x'+y'$

EX8- F(A,B,C) = Σ(0,2,3,7) Decoder.

* input = 3 , A,B,C
* output = 8 , $d_0 \rightarrow d_7$

$F = m_0 + m_2 + m_3 + m_7$



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

Implement 3-8 Decoder by using 2-4 Decoders.

8/4 = 2 → so we need 2 from (2-4 decoder)

* حسب الجدول , فهو بينقسم قسمين , إذا الأول اشتغل الثاني بيطفي , فلذلك بستخدم a_2 على إنه (enable).

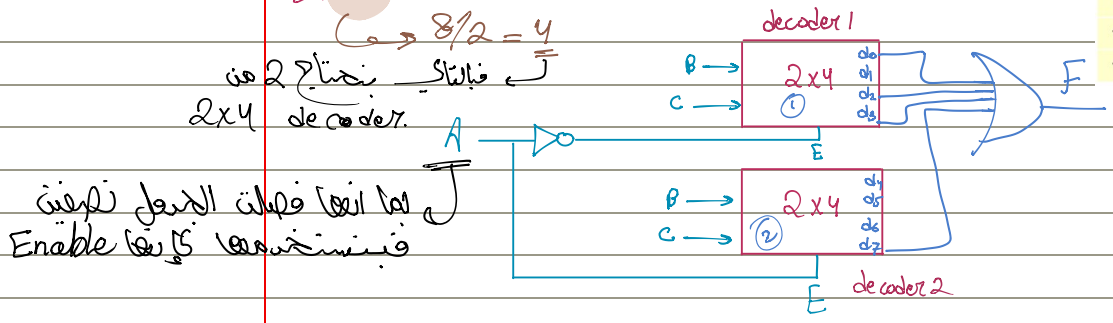
Inputs			Outputs							
a_2	a_1	a_0	d_0	d_1	d_2	d_3	d_4	d_5	d_6	d_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

3x8

→ 8/2 = 4

فبالتالي بحتاج 2 من 2x4 decoder.

بما انها قبلت الجدول تكون Enable فبستخدمها كذا

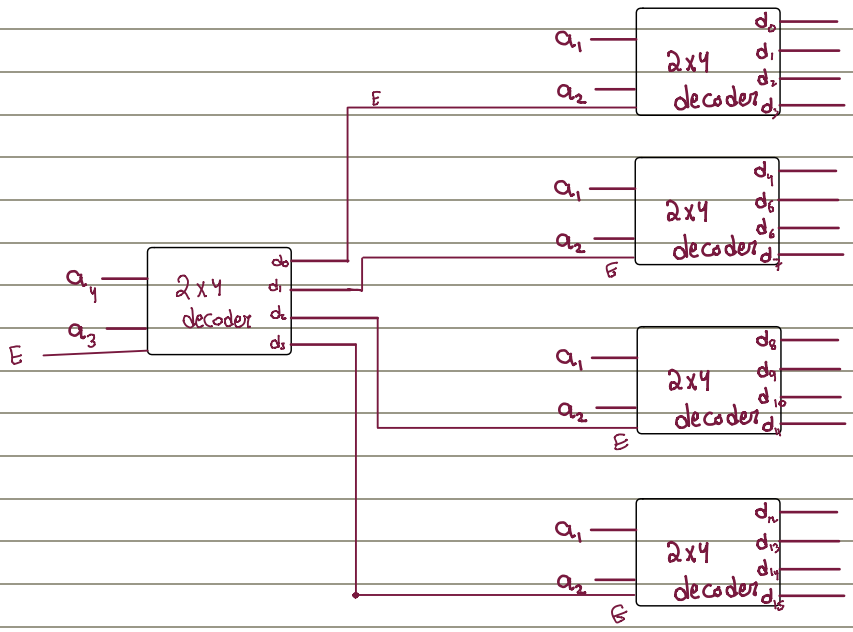


if A = 0, then decoder 2 will work, and decoder 1 will be disabled. and vise versa.

Example 8 Implement 4-16 decoder by using (2-4) decoder.

16/4 = 4 decoders.

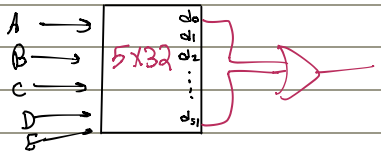
4 input, 16 output } 16 → 2 input → 4 decoders.



EX8- $F(A,B,C,D,E) = \Sigma(0,31)$

* input = 5, (A → E)
* output = $2^5 = 32$, ($d_0 \rightarrow d_{31}$)

Ans8- $F = m_0 + m_{31}$

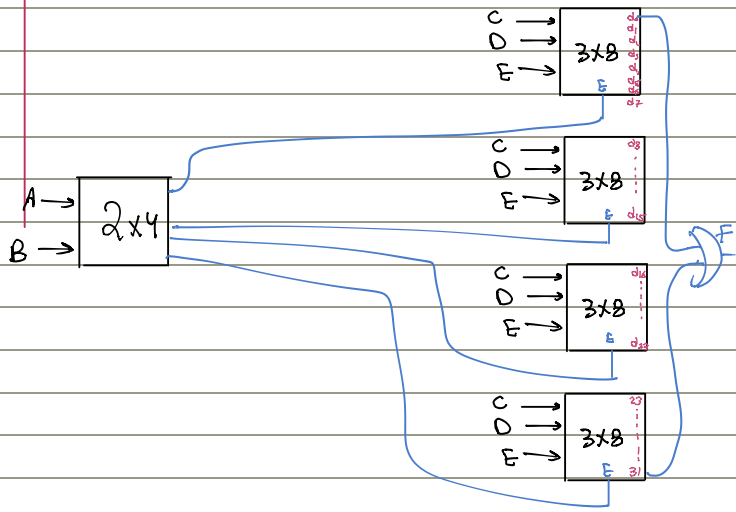


[3x8] Decoder باستخدام [5x32] لبناء

$5 \times 32 \rightarrow \frac{32}{8} = 4$

we need 4 decoders 3x8

5 4x2



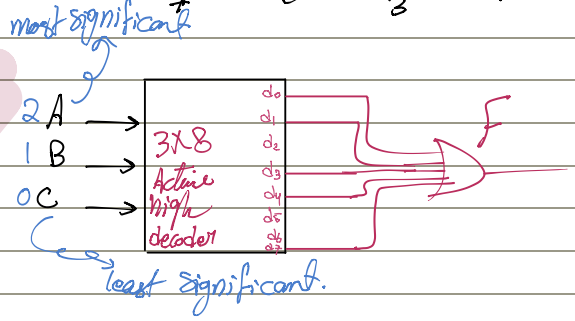
Examples- Implement the Boolean function $F(A,B,C) = AB + A'C + A'B'$

1. using a single 3x8 decoder and OR gate.
2. using single NOR gate and the minimum number of 2x4 decoders with enable

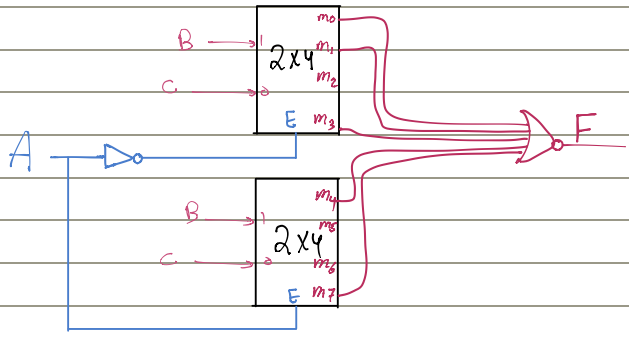
$F(ABC) = ABC + ABC' + A'BC + A'B'C + A'BC + A'B'C'$

$= m_7 + m_6 + m_3 + m_1 + m_0$

(A)

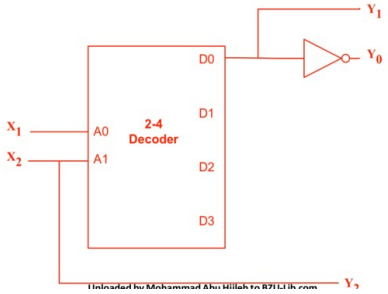


(B) $\frac{8}{4} = 2$.



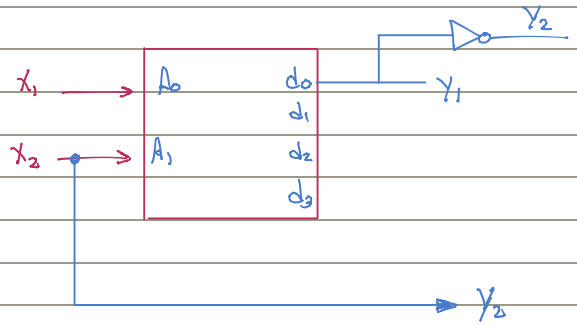
❖ Consider the following truth table, in which X_2, X_1, and X_0 are the inputs and Y_2, Y_1, and Y_0 are the outputs. Using a **minimum-size decoder** and a **minimum number** of additional gates, show how to implement Y_2, Y_1, and Y_0. Your additional logic gates must use the smallest possible number of inputs.

X_2	X_1	X_0	Y_2	Y_1	Y_0
0	0	0	0	1	0
0	0	1	0	1	0
0	1	0	0	0	1
0	1	1	0	0	1
1	0	0	1	0	1
1	0	1	1	0	1
1	1	0	1	0	1
1	1	1	1	0	1



Note that: $X_2 = Y_2$

$Y_1 = Y_0$



* A larger decoder can be built hierarchically in a similar way.

* يقدر أعمل (Implementation) لأي فنكشن باستخدام decoder + أي external gate

* Active high decoder :-
  - → OR gate (with minterms).
  - → NOR gate (with maxterms).

* Active low decoder
  - → NAND gate (with minterms).
  - → AND gate (with maxterms).

مهم جدًا :-
لازم أختار اللي عدده أقل بين الـ (minterms و maxterms).

* Enable :-
  - En = 0 → output = 0 0 0
  - En = 1 → it works.

* the most significant → are for the Enables.
* the least significant → are the inputs for the smaller decoder.

Encoders

① ↳ the opposite operation of decoder

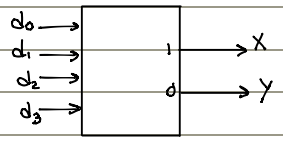
② # input 2^n
output n

③ Convert from decimal to Binary.

④ only one input should be 1 and all others must be 0's.

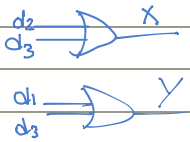
Ex:- 4 x 2 EnCoder
(4 → input, 2 → output)

Implement 4x2 Encoder



d_3	d_2	d_1	d_0	X	Y
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

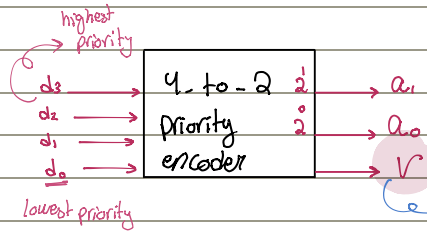
$X = d_2 + d_3$
$Y = d_1 + d_3$



فين لاقي الاتنين عندها
= قيمة البايني 10

This type of EnCoder has two problems (Limitations) :-
① if all input are Zero's.
② if more than one input are 1.

↳ "priority EnCoder" :-



d_3	d_2	d_1	d_0	a_1	a_0	V
0	0	0	0	X	X	0
0	0	0	1	0	0	1
0	0	1	X	0	1	1
0	1	X	X	1	0	1
1	X	X	X	1	1	1

(3 ← 2, 11 ← 10, 7 ← 4, 111 ← 100, 15 ← 8, 1111 ← 1000)

"تتحقق إذا كانوا كل الإتنين صفر"

a_1, a_0 وعندها
undefined! بكونو

100 → 4
111 → 7

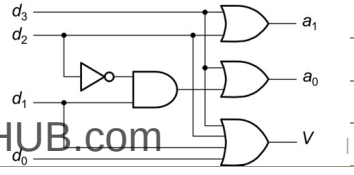
$V = d_0 + d_1 + d_2 + d_3$
$a_1 = d_3 + d_2$
$a_0 = \bar{d_3} d_0 + d_2$

a_1

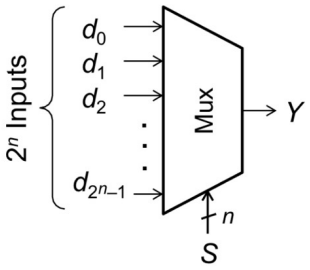
d_3d_2 \ d_1d_0	00	01	11	10
00				
01	1	1	1	1
11	1	1	1	1
10	1	1	1	1

a_0

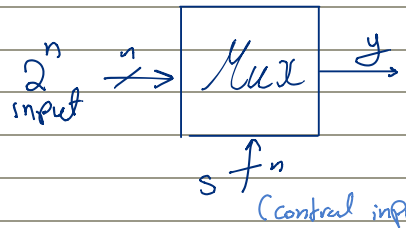
d_3d_2 \ d_1d_0	00	01	11	10
00			1	1
01				
11	1	1	1	1
10	1	1	1	1



multiplexer



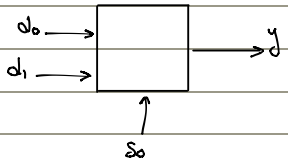
* $d_0 \rightarrow d_{n-1}$ input $\equiv 2^n$ input.
* only one output.
* select only one input of the data inputs to the output.



* Just one of the data inputs directed to the output based on the value of S

Ex: Design a 2x1 mux.

* input = 2
* output = 1
* S = 1



if (S==0) y = d_0;
else y = d_1;

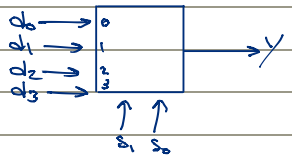
S	d_0	d_1	Y
0	0	X	$d_0 = 0$
0	1	X	$d_0 = 1$
1	X	0	$d_1 = 0$
1	X	1	$d_1 = 1$

logic expressions:

$Y = d_0 S' + d_1 S$.

Ex: Design 4-to-1 Multiplexer

* input = $4 = 2^2$
* output = 1
* S = 2

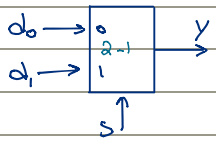


expression:

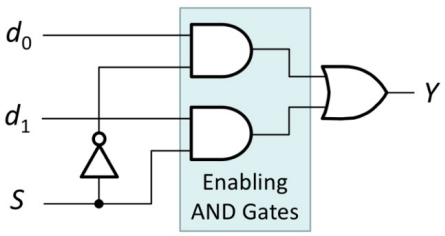
$Y = S_1' S_2' d_0 + S_1' S_0 d_1 + S_1 S_0' d_2 + S_1 S_0 d_3$

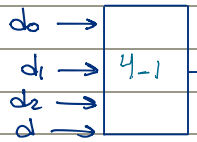
Inputs						Output
S_1	S_0	d_0	d_1	d_2	d_3	Y
0	0	0	X	X	X	$0 = d_0$
0	0	1	X	X	X	$1 = d_0$
0	1	X	0	X	X	$0 = d_1$
0	1	X	1	X	X	$1 = d_1$
1	0	X	X	0	X	$0 = d_2$
1	0	X	X	1	X	$1 = d_2$
1	1	X	X	X	0	$0 = d_3$
1	1	X	X	X	1	$1 = d_3$

Implementation multiplexer 8o

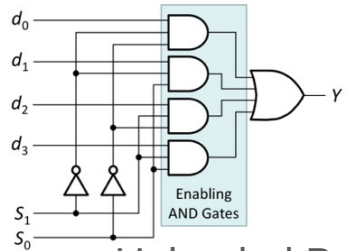


$Y = d_0 S' + d_1 S$

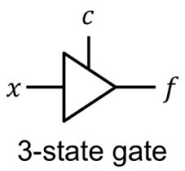




$Y = d_0 S_1' S_0' + d_1 S_1' S_0 + d_2 S_1 S_0' + d_3 S_1 S_0$



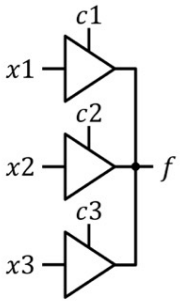
3-state gate



c	x	f
0	0	Z
0	1	Z
1	0	0
1	1	1

→ high Impedance, disconnected. "open switch".

if c=0 then f=Z
if c=1 the f=x

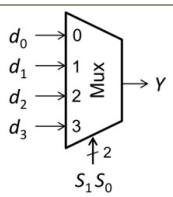


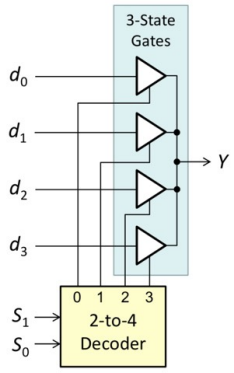
c1	c2	c3	f
0	0	0	Z
1	0	0	x1
0	1	0	x2
0	0	1	x3
0	1	1	Burn
1	0	1	Burn
1	1	0	Burn
1	1	1	Burn

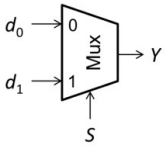
→ إذا كان فيه أكثر من 1 input = فضلها بتحرق الدارة

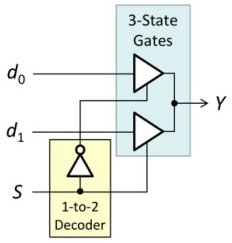
A multiplexer Can also be implemented using 2-

1. A decoder.
2. Three-state-gates → input بكون بعد الـ





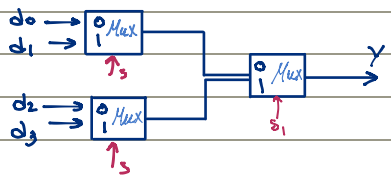




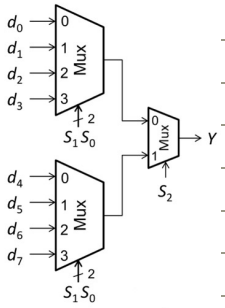
* Building larger Multiplexers 8-

EX.1 :- build 4-1 Mux using 2-1 mux.

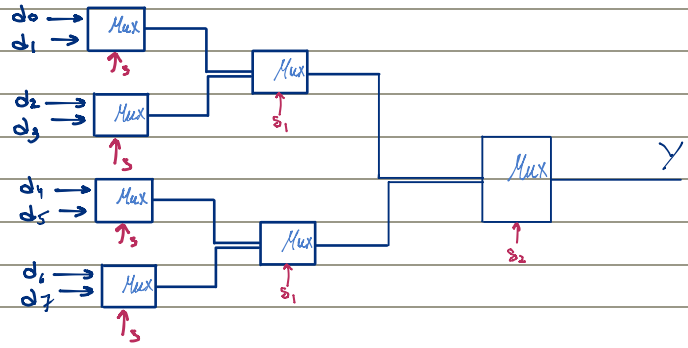
$\frac{4}{2} = 2$,



EX. build 8-1 Mux using 4-1 Muxes.



EX. build 8-1 mux using 2-1 muxes.



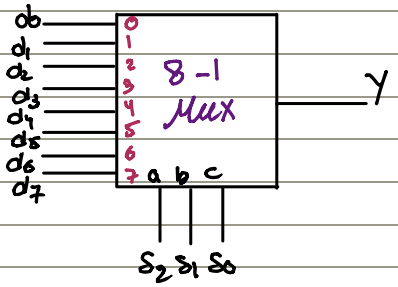
Implementing a function with a Multiplexer 80.

- A multiplexer can be used to implement any logic function.
- the function must be expressed by using minterms.

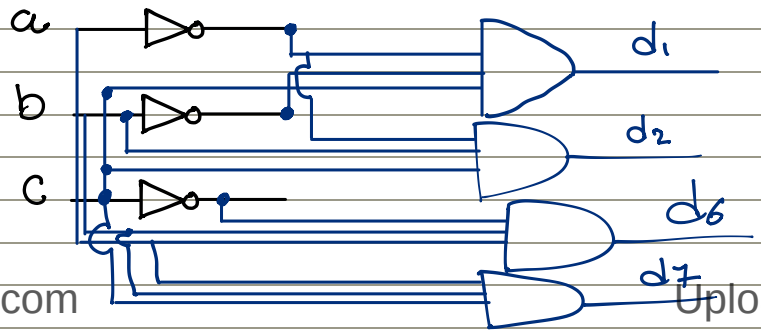
EX:- Implement $F(a,b,c) = \Sigma(1,2,6,7)$ using a Mux ← بعاد السؤال ماحددلك إ يش حجم الـ mux اللي تستخدموا فبستخدم اللي بدنا إياه.

a	b	c	F
0	0	0	0
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

* input = 8
* output = 1
* selectors = 3

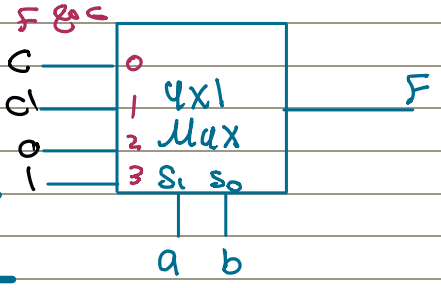


$F = d\,b'c + a'bd + abc' + abc.$



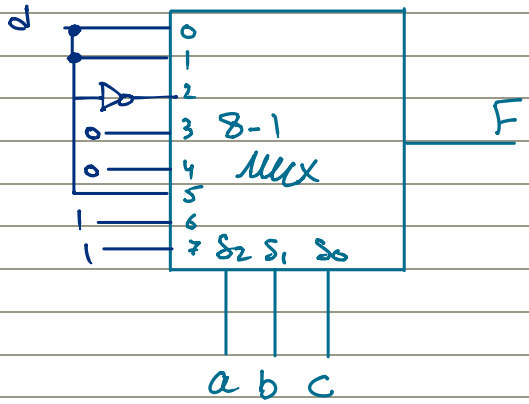
EX8 Re Implement $F(a,b,c) = \Sigma(1,2,6,7)$ using a 4-to-1

a	b	c	F	Comment → F & C
0	0	0	0	F = C
0	0	1	1	
0	1	0	1	F = C'
0	1	1	0	
1	0	0	0	F = 0
1	0	1	0	
1	1	0	1	F = 1
1	1	1	1	



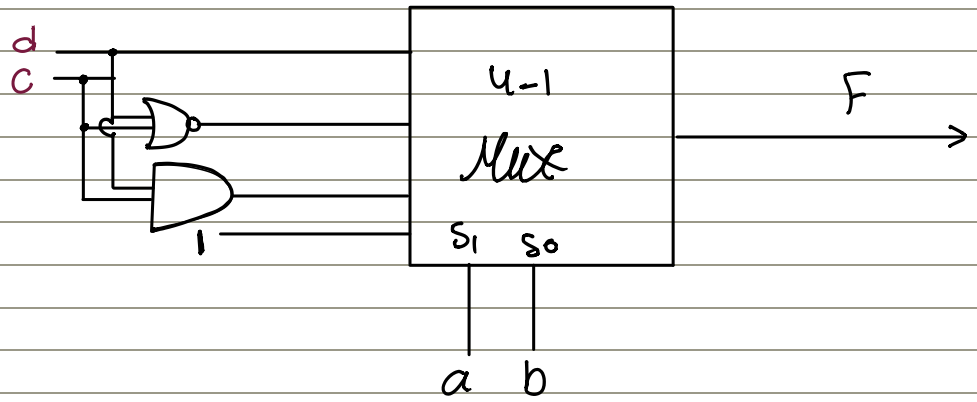
Example 8- Implement $F(a,b,c,d) = \Sigma(1,3,4,11,12,13,14,15)$ using 8-to-1 Mux

a	b	c	d	F	Comment
0	0	0	0	0	F = d
0	0	0	1	1	
0	0	1	0	0	F = d
0	0	1	1	1	
0	1	0	0	1	F = d'
0	1	0	1	0	
0	1	1	0	0	F = 0
0	1	1	1	0	
1	0	0	0	0	F = 0
1	0	0	1	0	
1	0	1	0	0	F = d
1	0	1	1	1	
1	1	0	0	1	F = 1
1	1	0	1	1	
1	1	1	0	1	F = 1
1	1	1	1	1	



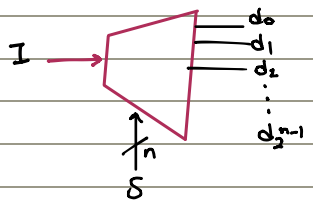
Ex8- Re Implement $F(a,b,c,d) = \Sigma(1,3,4,11,12,13,14,15)$ by using 4-1 mux

a	b	c	d	F	Comment
0	0	0	0	0	
0	0	0	1	1	F = d
0	0	1	0	0	
0	0	1	1	1	
0	1	0	0	1	
0	1	0	1	0	$F = (c+d)'$
0	1	1	0	0	
0	1	1	1	0	
1	0	0	0	0	
1	0	0	1	0	F = cd
1	0	1	0	0	
1	0	1	1	1	
1	1	0	0	1	
1	1	0	1	1	F = 1
1	1	1	0	1	
1	1	1	1	1	



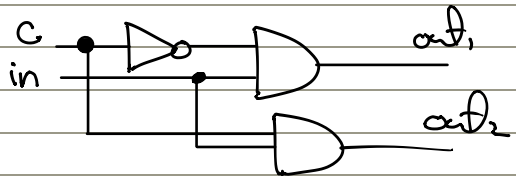
Demultiplexer ← عكس الـ Mux

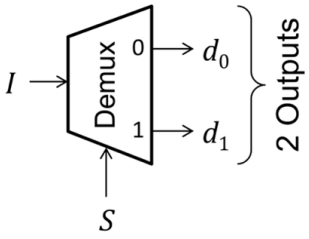
- only one input.
- an n bit selector.



1-2 Demultiplexer :-

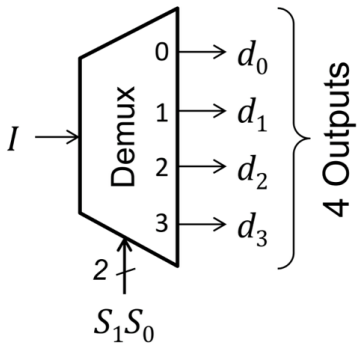
in	C	out1	out2
0	0	0	0
1	0	1	0
0	1	0	0
1	1	0	1





if $S=0 \rightarrow I=d_0$

$S=1 \rightarrow I=d_1$



$S_1S_0 = 00 \rightarrow I = d_0$

$S_1S_0 = 01 \rightarrow I = d_1$

$S_1S_0 = 10 \rightarrow I = d_2$

$S_1S_0 = 11 \rightarrow I = d_3$