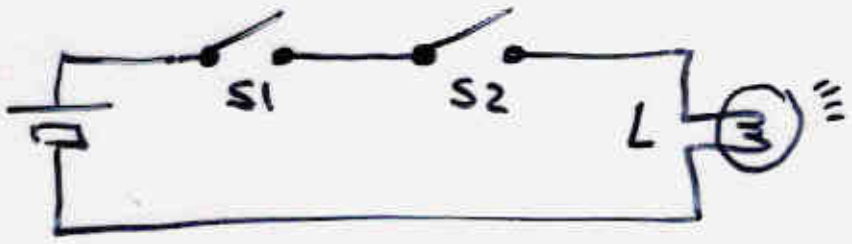
Logic gates

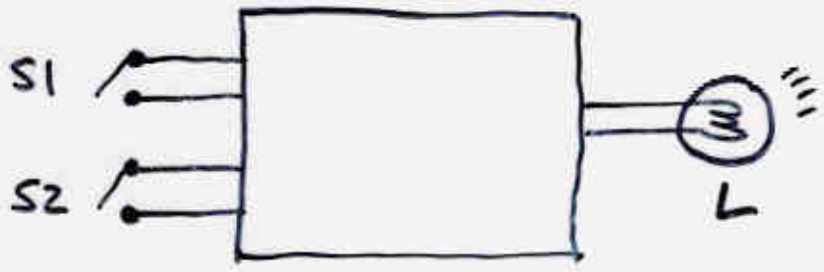
- circuits that perform operations on digital waveforms or number representations

- logical operation



S1 and S2 = L

- "black-box" version



one or more inputs, one or more outputs

- described by the "truth table"

S1	S2	L
0	0	0
0	1	0
1	0	0
1	1	1

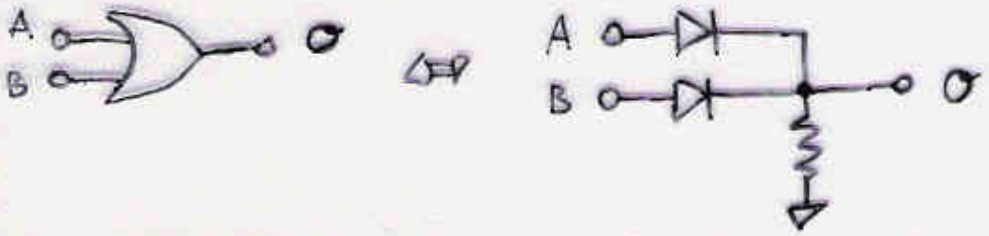
Ex binary addition truth table:

+	0	1
0	0	1
1	1	10

⇒

A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

- **OR gate** "if A is true, OR B is true, then O is tru



$A, B = \emptyset$ or $+V_o$
ideal diodes

if at least one of the diodes is forward biased ($A = V_o$ or $B = V_o$) then $O = V_o$

A	B	O
∅	∅	∅
1	∅	1
∅	1	1
1	1	1

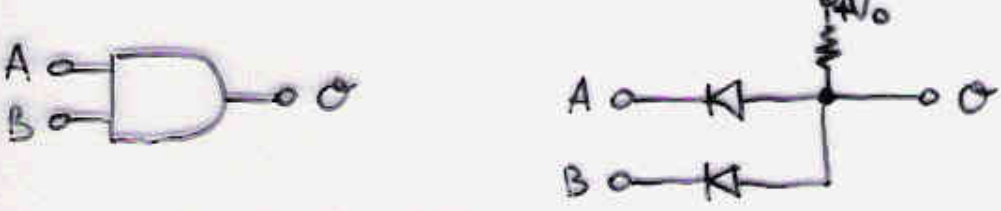
← in "units" of V_o

⇒ the truth table is almost like that of addition

⇒ $\boxed{O = A + B}$

possible: ≥ 2 inputs

- **AND gate** "if A is true AND B is true, then O is true"



(coincidence detecto

$A, B = \emptyset$ or $+V_o$

if either diode is forward biased ($A = \emptyset$ or $B = \emptyset$) then $O = \emptyset$, otherwise $O = +V_o$

A	B	O
∅	∅	∅
∅	1	∅
1	∅	∅
1	1	1

in units of V_o

⇒ the truth table is like that of multiplication

$\boxed{O = A \cdot B}$ ≥ 2 input

- diode logic gates limited by degradation of signal (~0.7V/gate) when several gates connected together. Also, they have limited # of inputs as each loads down V_o to a point where "1" is no longer recognized. Use transistors, op-amps ... instead!

Ex 1. Use truth table to prove De Morgan's theorem

$\overline{A+B} = \overline{A} * \overline{B}$ $\qquad$ $\overline{A*B} = \overline{A} + \overline{B}$

A	B	A+B	$\overline{A+B}$	$\overline{A}$	$\overline{B}$	$\overline{A}*\overline{B}$	A*B	$\overline{A*B}$	$\overline{A}+\overline{B}$
0	0	0	1	1	1	1	0	1	1
0	1	1	0	1	0	0	0	1	1
1	0	1	0	0	1	0	0	1	1
1	1	1	0	0	0	0	1	0	0

12

Ex 2. reduce the circuit via truth table

A	A	A+A	$\overline{A+A}$	$\overline{A}$
0	0	0	1	1
1	1	1	0	0

Ex 3

1)

A	B	$\overline{A}$	$\overline{B}$	$\overline{A}+\overline{B}$	$\overline{(\overline{A}+\overline{B})}$	A*B
0	0	1	1	1	0	0
0	1	1	0	1	0	0
1	0	0	1	1	0	0
1	1	0	0	0	1	1

2) Formally, using theorems

$\overline{(\overline{A}+\overline{B})} = A * \overline{\overline{B}} = A * B$

⇒ De Morgan's theorem: all logic operations with only NOR gates or only NAND gates

Aside : Boolean algebra

OR → just like addition (except $1+1=1$ ← should be $1\emptyset$)

AND → just like multiplication

⇒ Operations of an algebra of a two-state variable called Boolean algebra (George Boole)

One more operation : negation (complementation)
where $1 \to \emptyset$, $\emptyset \to 1$.

Notation : $A+B$, $A*B$, $\overline{A}$

A	B	A+B	A*B	$\overline{A}$
0	0	0	0	1
1	0	1	0	0
0	1	1	0	1
1	1	1	1	0

Gate types: 

Algebraic rules (theorems):

$A+A=A$	$A+\emptyset=A$	$\overline{(\overline{A})}=A$
$A*A=A$	$A*\emptyset=\emptyset$	$A+(A*B)=A$
$A+1=1$	$A+\overline{A}=1$	$A*(A+B)=A$
$A*1=A$	$A*\overline{A}=\emptyset$	$A*(\overline{A}+B)=A*B$

etc.

$\overline{A*B}=\overline{A}+\overline{B}$ $\qquad$ $\overline{A+B}=\overline{A}*\overline{B}$ ← De Morgan's theorems

⇒ can generate all logical operations out of OR, AND, and NOT

- other logic gates

NOR  $O = \overline{A+B}$

A	B	O
0	0	1
0	1	0
1	0	0
1	1	0

NAND  $O = \overline{A*B}$

A	B	O
0	0	1
0	1	1
1	0	1
1	1	0

XOR  $O = (A+B)*\overline{(A*B)}$
$= A \oplus B$

A	B	O
0	0	0
0	1	1
1	0	1
1	1	0

exclusive OR : A or B, but not both

XNOR  $O = (A*B) + (\bar{A}*\bar{B})$

A	B	O
0	0	1
0	1	0
1	0	0
1	1	1

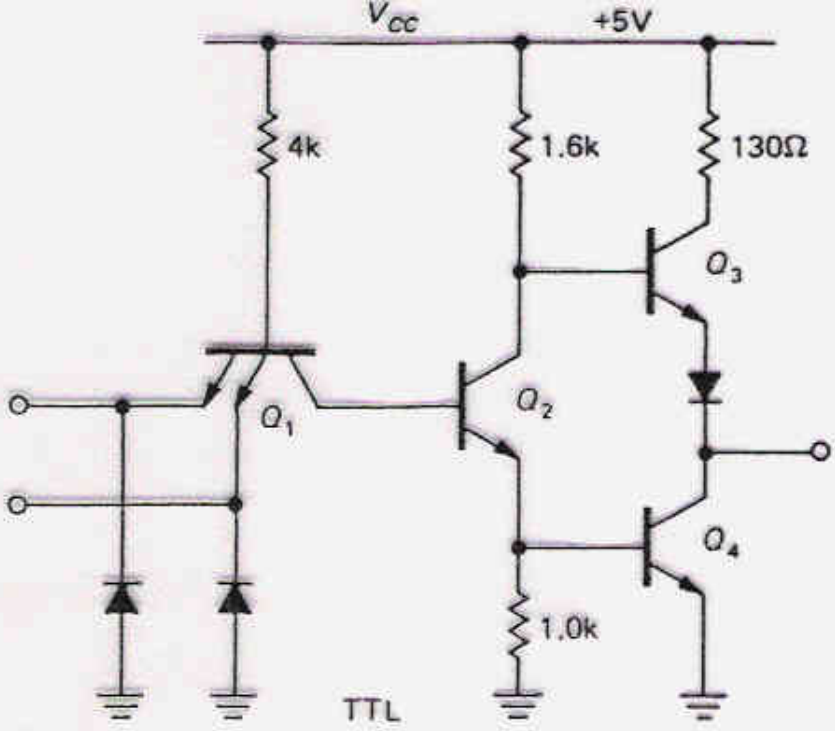
exclusive NOR: only if A and B same
⇒ equality detector

- IC logic gates

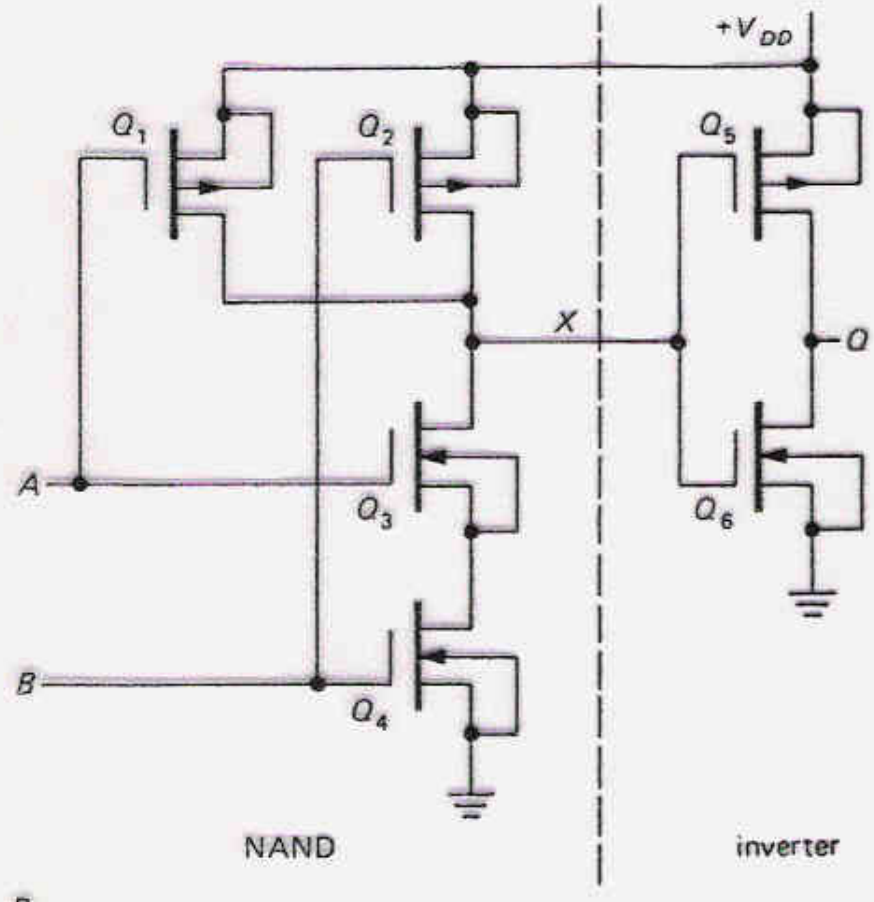
- improve reliability
- reduce power dissipation
- increase operating speed

} trade-off, usually

IC types		Speed	V (0/1)	Power	Fanout
rare	RTL	50 ns	0.2/0.9	10 mW	4
rare	DTL	25 ns	0.2/0.4	15 mW	8
	TTL	10 ns	0.3 / 3	20 mW	10
	ECL	2 ns	-1.5/-0.75	50 mW	24
	MOS	200 ns	0/3-15	0.3 mW	50



A



B

Figure 8.17

A: TTL NAND gate. B: CMOS AND gate.

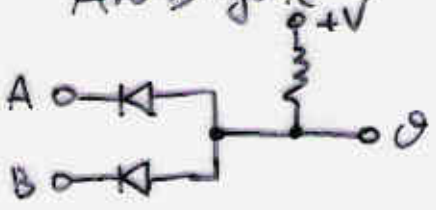
- negative logic

so far, "0" ⇒ 0 (or low) Voltage
"1" ⇒ +5V (or +Ve) Voltage
} positive logic

interchange the designation of logic levels
⇒ negative logic "0" ⇒ +5V , "1" ⇒ 0V

A	B	ϑ
0V	0V	0V
0V	+5V	0V
+5V	0V	0V
+5V	+5V	+5V

AND gate



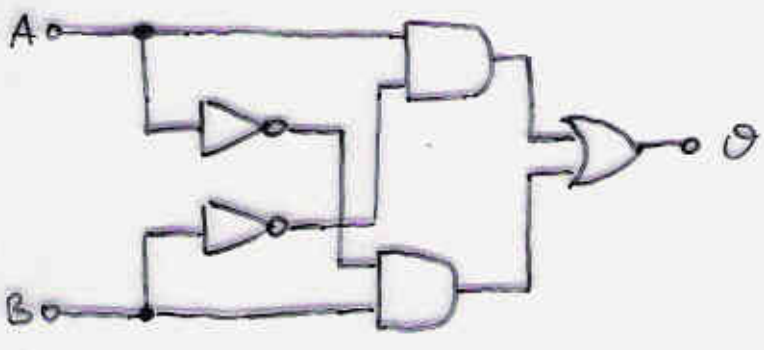
A	B	ϑ
1	1	1
1	0	1
0	1	1
0	0	0

+ve logic : $\vartheta = A * B$
−ve logic : $\vartheta = A + B$!

OR gate ⇔ AND gate

inverter : logic level exchanger

Ex.

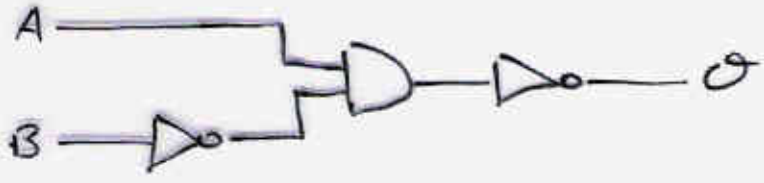


A	B	$\bar{A}$	$\bar{B}$	$A * \bar{B}$	$B * \bar{A}$	$(A * \bar{B}) + (B * \bar{A})$	$A \oplus$
0	0	1	1	0	0	0	0
0	1	1	0	0	1	1	1
1	0	0	1	1	0	1	1
1	1	0	0	0	0	0	0

⇒ This is an XOR gate (A or B, but not both)

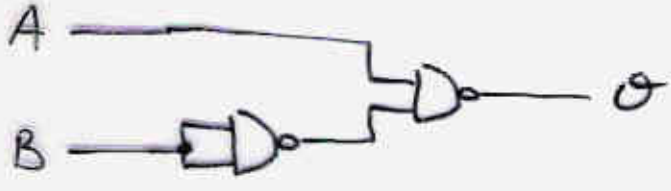
Ex negative logic on the input

i.e.



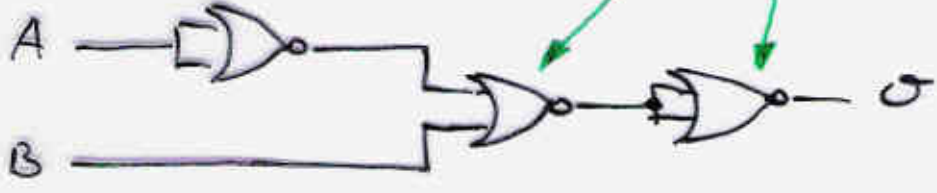
A	B	$\bar{B}$	$A * \bar{B}$	$\overline{A * \bar{B}} = O$
0	0	1	0	1
0	1	0	0	1
1	0	1	1	0
1	1	0	0	1

- Using only NAND gates :



Using only NOR gates :

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



because

Aside

The most general set of functions of two variables:

A	B	F_0	F_1	F_2	F_3	....	F_7	F_8	.....	F_{14}	F_{15}
0	0	0	0	0	0		0	1		1	1
0	1	0	0	0	0		1	0		1	1
1	0	0	0	1	1		1	0		1	1
1	1	0	1	0	1		1	0		0	1
		↑	↑				↑	↑		↑	↑
		$\emptyset$	A*B				A+B	$\overline{A+B}$		$\overline{A*B}$	1
			AND				OR	NOR		NAND	

$F_0 = \emptyset$

$F_1 = A \cdot B$

$F_2 = A \cdot \bar{B}$

$F_3 =$

$F_4 =$

$F_5 =$

$F_6 =$

$F_7 = A+B$, OR

$F_8 = \overline{A+B}$, NOT OR = NOR

$F_9 =$

$F_{10} =$

$F_{11} =$

$F_{12} =$

$F_{13} =$

$F_{14} = \overline{A \cdot B} = \bar{F}_1$, NOT AND = NAN

$F_{15} = 1$

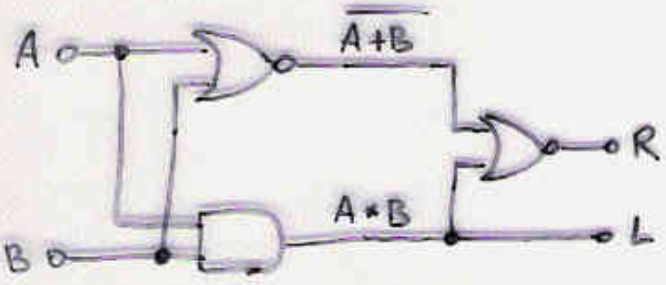
EFTS: complete the above table.

Ex: algebraic simplification

$$f = B\bar{C}\bar{D} + \bar{A}BD + ABD + B\bar{C}\bar{D} + \bar{B}CD + \bar{A}\bar{B}\bar{C}D + A\bar{B}\bar{C}D$$

$$= B\bar{D}(C+\bar{C}) + BD(\bar{A}+A) + \bar{B}CD + \bar{B}\bar{C}D(\bar{A}+A)$$

Logic-gate applications : adders



A	B	L	R
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

reads:
0+0 = 0
0+1 = 1
1+0 = 1
1+1 = 2 $= (LR)_{10}$

XOR

$L = A*B$

$R = \overline{(\overline{A+B})+(A*B)} = \overline{(\overline{A+B})}*\overline{(A*B)}$

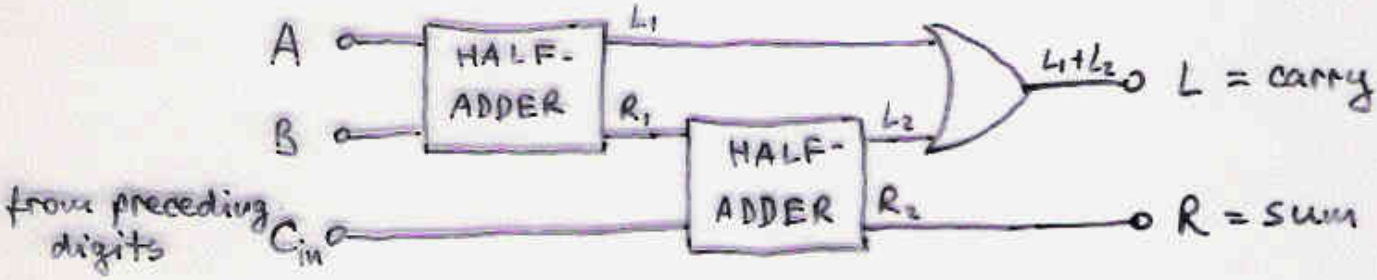
$= (A+B)*(\bar{A}+\bar{B}) = A\bar{A}+A\bar{B}+B\bar{A}+B\bar{B}$

$= A \oplus B$ ← $A+B$ but not $A*B \equiv$ XOR

⇒ adder of 2 bits with a carry, called

HALF-ADDER – because need to include the possibility that a carry bit (from another digit) affects the sum

To get a FULL ADDER, combine two HA.'s and an OR gate (needed for the case of a carry bit added to the sum generating a new carry):

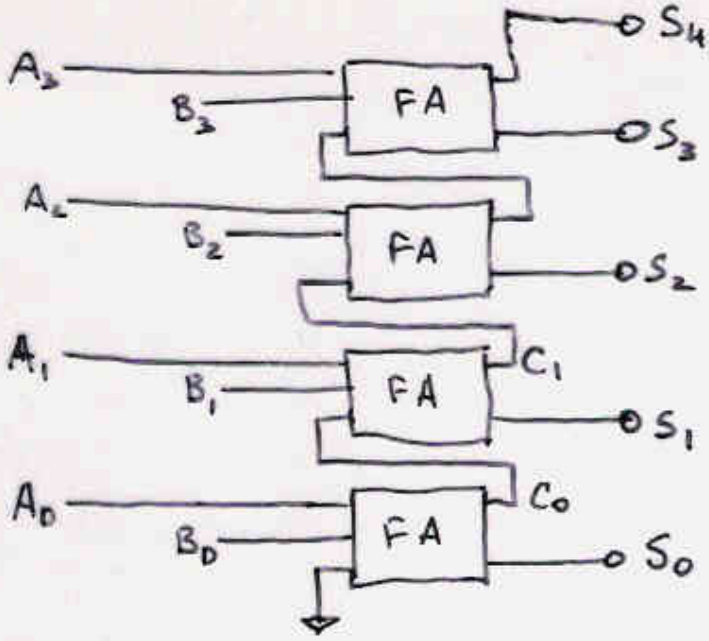


EFTS: use the truth table of a full adder to verify its operation.

⇒ Now we can do non-trivial operations!

E.g. add two 4-bit numbers.

Stack F.A.s together:



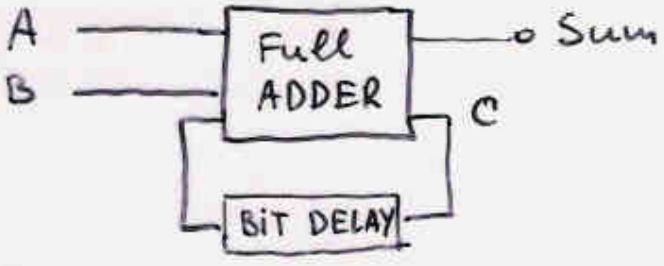
A	B	S
9	13	22

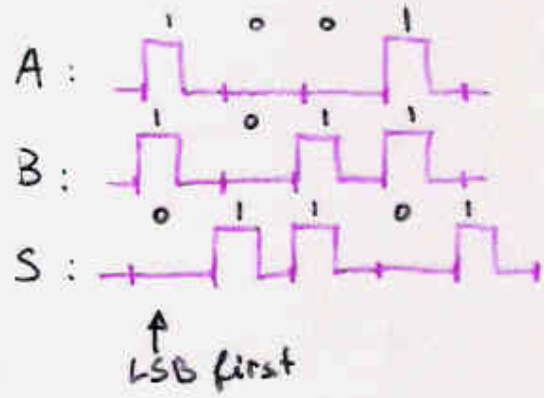
1001 + 1101 = 10110
A_3 A_0 B_3 B_0 S_4 S_0

N.B. 4-bit adder
⇒ 5-bit result!
(max sum = 15+15 = 30

The above is an example of a parallel adder, i.e. simultaneously operating on all bits.

Also possible: serial adder, operating on each bit in turn:





BIT DELAY: so that we allow the carry bit generated in the previous bit addition to modify the subsequent bit addition.

SERIAL	vs	PARALLEL
– one bit at a time, slow		– many bits at once, fast
– unlimited size of integers		– max size of Nº fixed
– simple circuit		– more complex circuit

Aside : parallel vs. serial data transmission

serial :

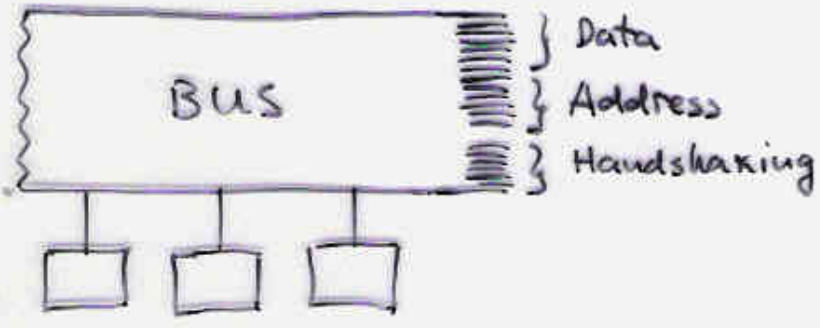
- long-distance on 2-conductor line
- rate is bits per second = baud rate
  Max baud rate proportional to the bandwidth of the circuit : $\lesssim$ 10 kHz $\Rightarrow$ 10 kbaud
  for audio circuits (telephone)
- Modem : modulator - demodulator

Tx :	2225 Hz = 1	2025 Hz = Ø	} 300 baud
Rx :	1270 Hz = 1	1070 Hz = Ø	}

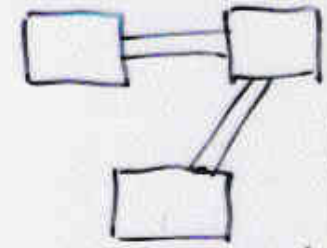
- error correction : parity bits

parallel :

- short distances, multi-conductor lines
- often: bus configuration



cf.



All devices have access to all of the information, but only respond when their address is "called"

Ex. Computer's internal buses; General-Purpose Interconnect Bus (GPIB); scsi, etc.
Note: serial buses also possible : cars, USB

So far: circuit ⟹ equation describing its operation or the truth table

⟸ ?

What about the reverse?

Designing circuits for a given task

Ex.1. two-way light switch.

A	B	Light	
1	1	ON	1
1	0	OFF	2
0	1	OFF	3
0	0	ON	4

all 4 change the state of the light

Recall:

AND : coincidence circuit, only one combination of inputs gives 1 on the output

OR : gathers together the 1's from different inputs.

⇒

2 minitermes →

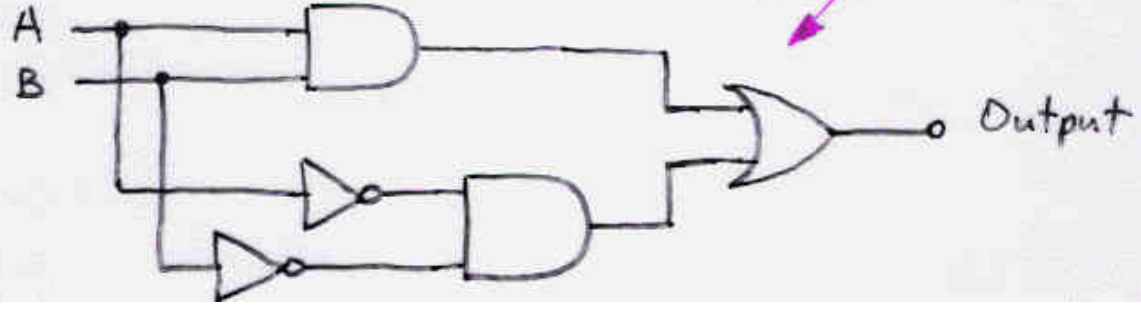
A=1 AND B=1 — line 1

A=0 AND B=0 — line 4

OR

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

⇒



Ex 2 Adder of two 4-bit numbers

Solution: four stacked full adders, as before

Ex 3: Multiplier of two 4-bit numbers

Solution:

$$\begin{array}{r} A = 12_{10} \\ \times\ B = 13_{10} \\ \hline 36 \\ +\ 12 \\ \hline 156_{10} \end{array}$$

$$\begin{array}{r} 1100_2 \\ \times\ 1101_2 \\ \hline 1100 \\ +\ 0000 \\ +\ 1100 \\ +\ 1100 \\ \hline 10011100_2 \end{array}$$

← first "partial product"

← second, shifted

etc

i.e. multiplication → series of additions

Also possible: Brute force approach; add A to itself B times, i.e. A+A while counting B down to Ø.

Ex 4. 2-bit multiplication

$$\begin{array}{r} A_1 A_0 \\ \times\ B_1 B_0 \\ \hline (A_1 B_0)(A_0 B_0) \\ (A_1 B_1)(A_0 B_1) \\ \hline \cdots \end{array}$$

where

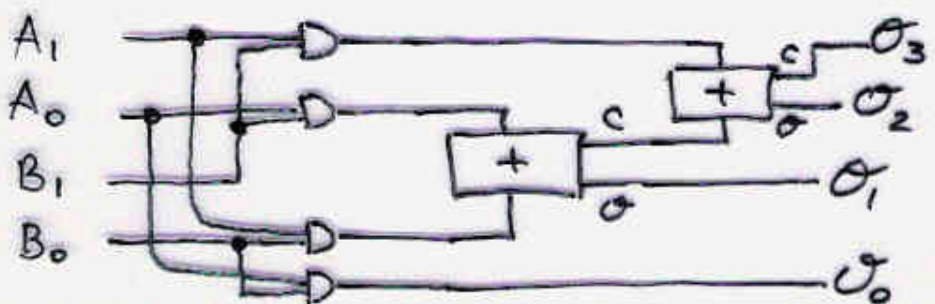
A/B	0	1
0	0	0
1	0	1

$A_1 A_0 \quad B_1 B_0 \quad O_3 O_2 O_1 O_0$

4 inputs = 16 lines

. . . . .

EFTS



Ex 5

A	B	C	X	Y	Z
1	1	1	ϕ	ϕ	ϕ
1	1	ϕ	ϕ	1	1
1	ϕ	1	ϕ	1	1
1	ϕ	ϕ	1	ϕ	ϕ
ϕ	1	1	ϕ	ϕ	1
ϕ	1	ϕ	1	ϕ	ϕ
ϕ	ϕ	1	1	1	ϕ
ϕ	ϕ	ϕ	ϕ	1	1

For example, for X:

1) 3 miniterms

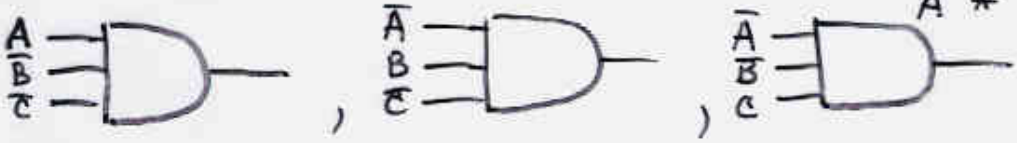
⇒ 

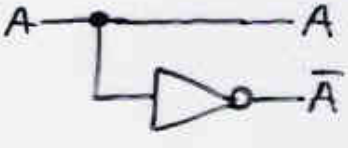
2) A=1 and B=ϕ and C=ϕ

⇒ $A * \bar{B} * \bar{C}$

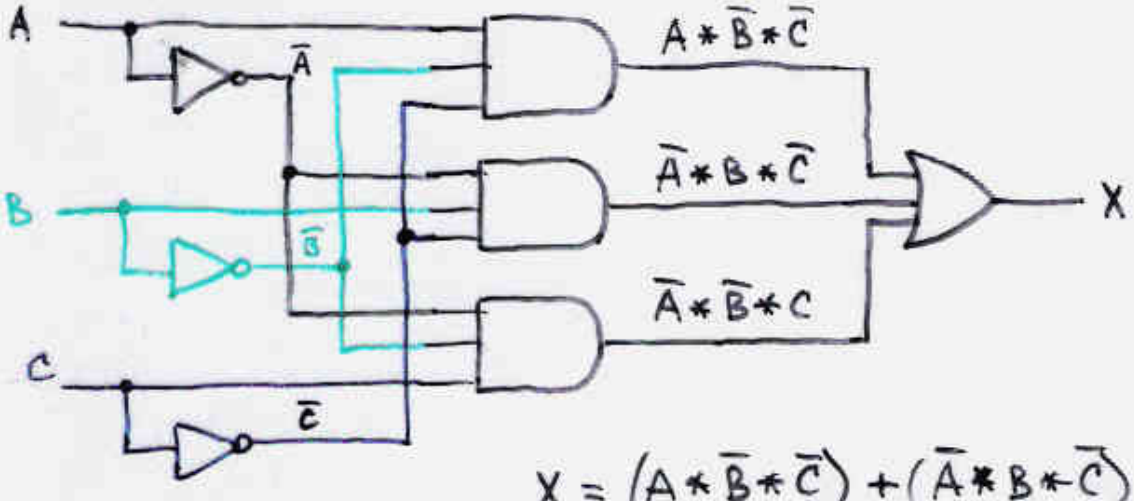
3) Also: $\bar{A} * B * \bar{C}$

$\bar{A} * \bar{B} * C$

⇒ 

4) use inverters to get $\bar{A}, \bar{B}, \bar{C}$ ⇒ 

Finally:



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

A	B	C	X	Y	Z
1	1	1	ϕ	ϕ	ϕ
1	1	ϕ	ϕ	1	1
1	ϕ	1	ϕ	1	1
1	ϕ	ϕ	1	ϕ	ϕ
ϕ	1	1	ϕ	ϕ	1
ϕ	1	ϕ	1	ϕ	ϕ
ϕ	ϕ	1	1	1	ϕ
ϕ	ϕ	ϕ	ϕ	1	1

For Y:

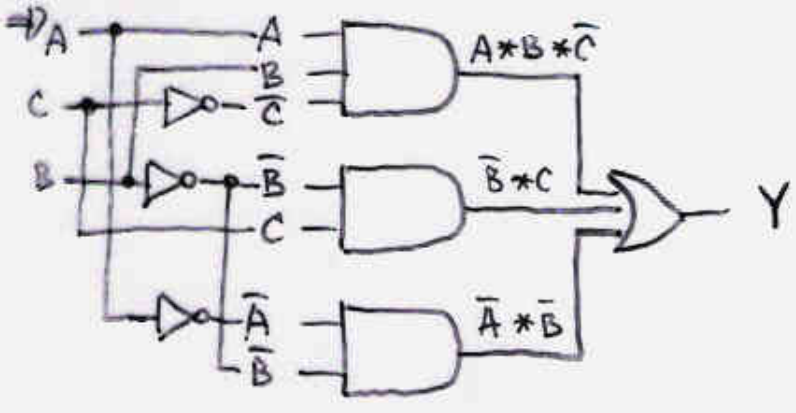
4 **miniterms**:

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

$$(A+\bar{A})*\bar{B}*C$$
$$\bar{A}*\bar{B}*(C+\bar{C})$$

$$A+\bar{A} = C+\bar{C} = 1$$

$$\Rightarrow\quad Y = (A*B*\bar{C}) + (\bar{B}*C) + (\bar{A}*\bar{B})$$



Aside: we used here one of the theorems:

$$A+A = A$$

since we needed two "copies" of $\bar{A}*\bar{B}*C$

The basic principle: interpret the truth table as a sum of products.

Programmable Logic Arrays (PLA's, or PAL's in old notation) allow implementation of arbitrary truth tables. Typically, one enters an equation, and the PLA gets burned-in so as to become a circuit implementing this equation. Often called "Glue logic". Ex: Early IBM PC had ≈ 100-200 IC's. Modern equivalent: 5-IC chipset: CPU, ICU... + logic PLA's

Digital Logic Devices

Parity generator

parity = the simplest error-detection scheme

Convention: even or odd parity

Add an extra bit to every group of bits (7,8) so that the total number of "1"s is even (odd)

			odd parity / even parity	
E.g.	00100100	+	1	0
	01101101	+	0	1

At the receiving end need a parity checker.

E.g. for even parity: error when the total number of bits in the bit group is odd, i.e. need an odd-parity detector.

For 4 bits: $A_3 A_2 A_1 A_0$

1) divide into 2 groups, A_3A_2 and A_1A_0

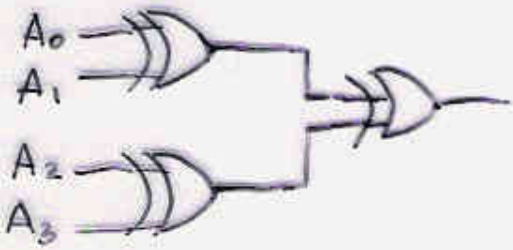
2) possibilities:

# bits in:	A_3A_2	A_1A_0	total
	odd	odd	even
	odd	even	odd
	even	odd	odd
	even	even	even

i.e. an XOR of detected odd parities from subgroups

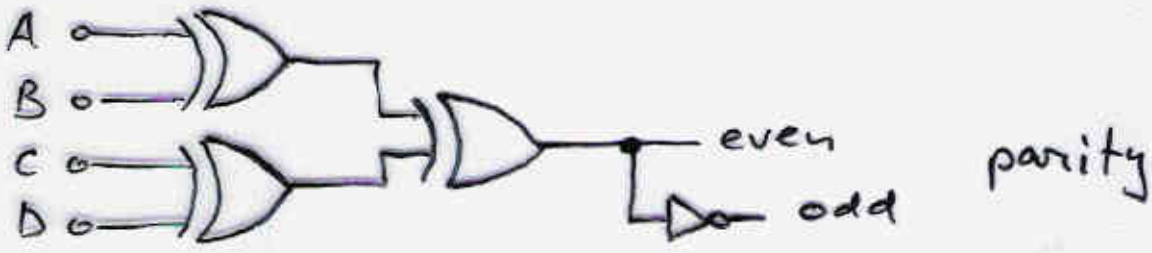
3) continue with procedure ...

⇒

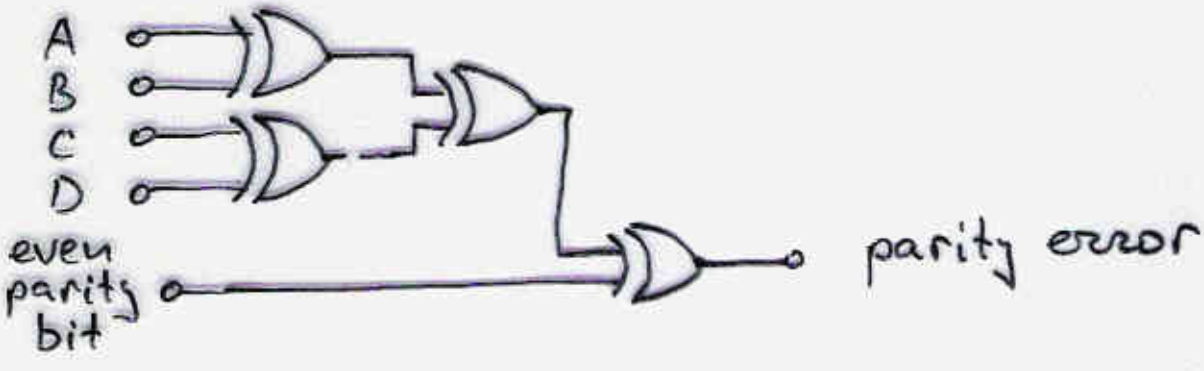


N.B. can generalize to more bits

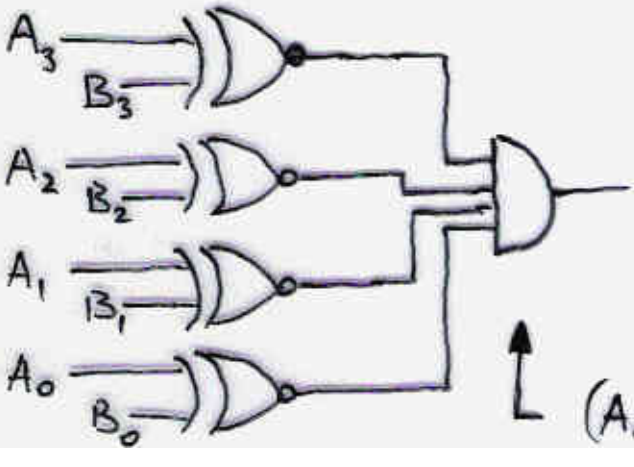
Ex a parity generator ...



vs. parity checker



Ex a digital comparator



A_i	B_i	$(A_i + B_i) * \overline{(A_i * B_i)}$
0	0	1
0	1	0
1	0	0
1	1	1

$A_i = B_i$

$(A_0 = B_0) * (A_1 = B_1) * (A_2 = B_2) * (A_3 = B_3$

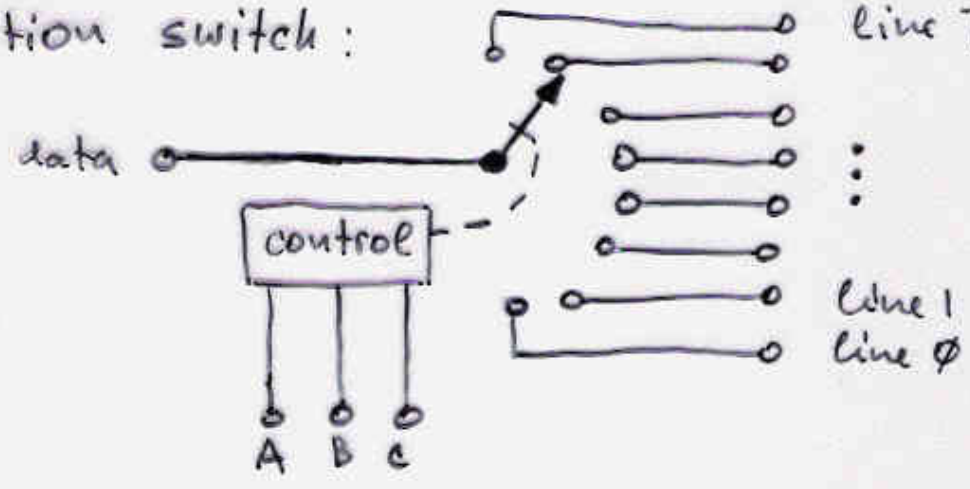
Note: parity error does not tell which of the bits is incorrect, only that there is an error

⇒ need re-transmission to correct.

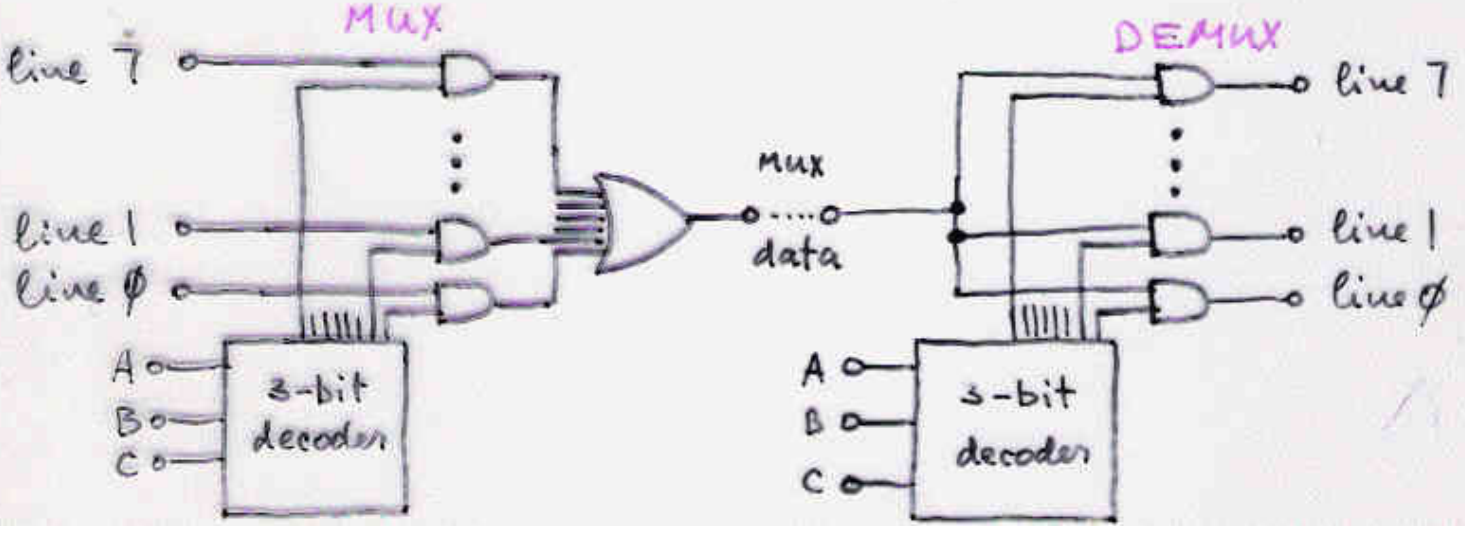
Other algorithms are possible which may provide for self-correction capability.

Multiplexing / Demultiplexing

A multi-position switch:

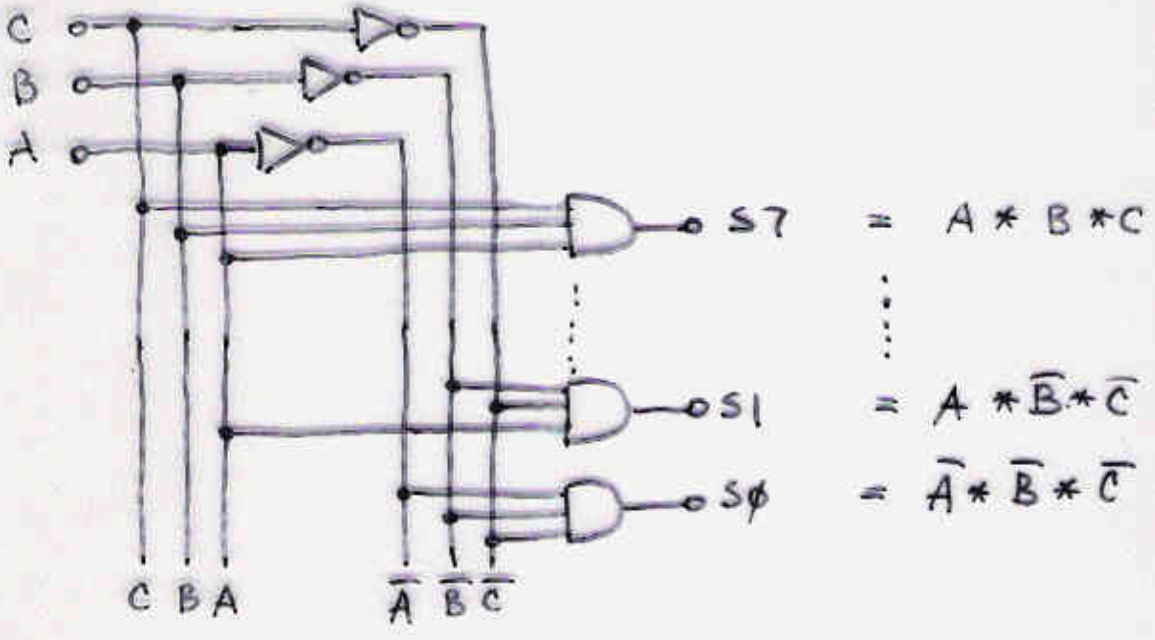


- Send data to 1-of-8 lines: demux (data distributor)
- Receive data from 1-of-8 lines: mux (data selector)

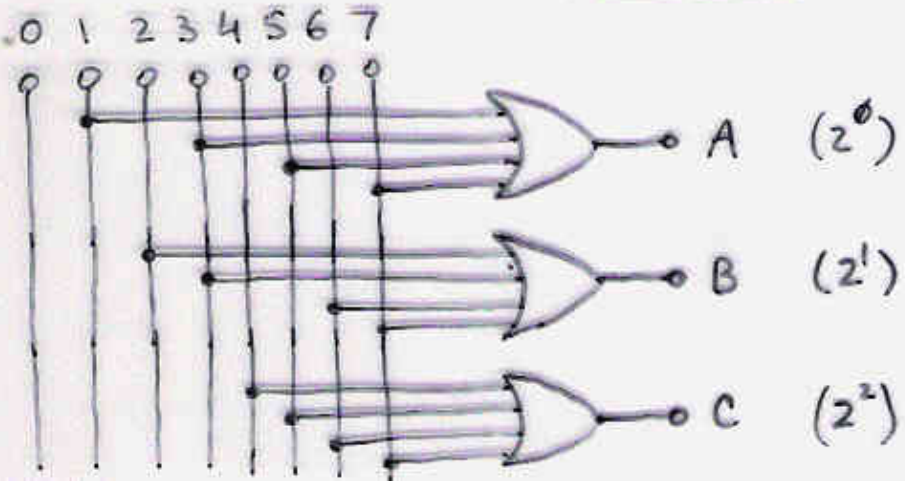


The crucial device: 3-bit to 1-of-8-line decoder

A	B	C	$(CBA)_{10}$	S0	S1	S2	S3	S4	S5	S6	S7
0	0	0	0	1	0	0	0	0	0	0	0
1	0	0	1	0	1	0	0	0	0	0	0
0	1	0	2	0	0	1	0	0	0	0	0
1	1	0	3	0	0	0	1	0	0	0	0
0	0	1	4	0	0	0	0	1	0	0	0
1	0	1	5	0	0	0	0	0	1	0	0
0	1	1	6	0	0	0	0	0	0	1	0
1	1	1	7	0	0	0	0	0	0	0	1

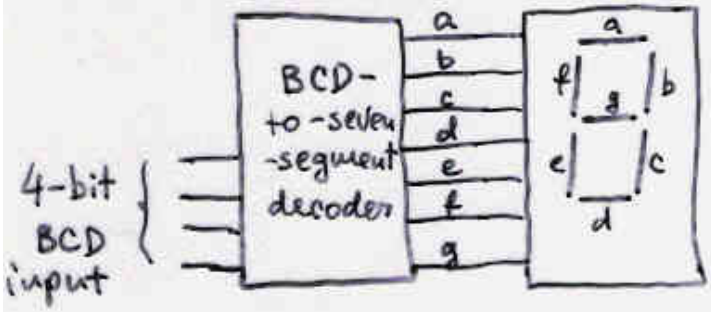


Of course, an octal-to-binary encoder is possible:



BCD and 7-segment displays

BCD: binary-coded-decimal, i.e. supply a decimal digit encoded in binary. Need 4 bits, and some upper combinations will go unused ($2^4-1 = 15 = max$)

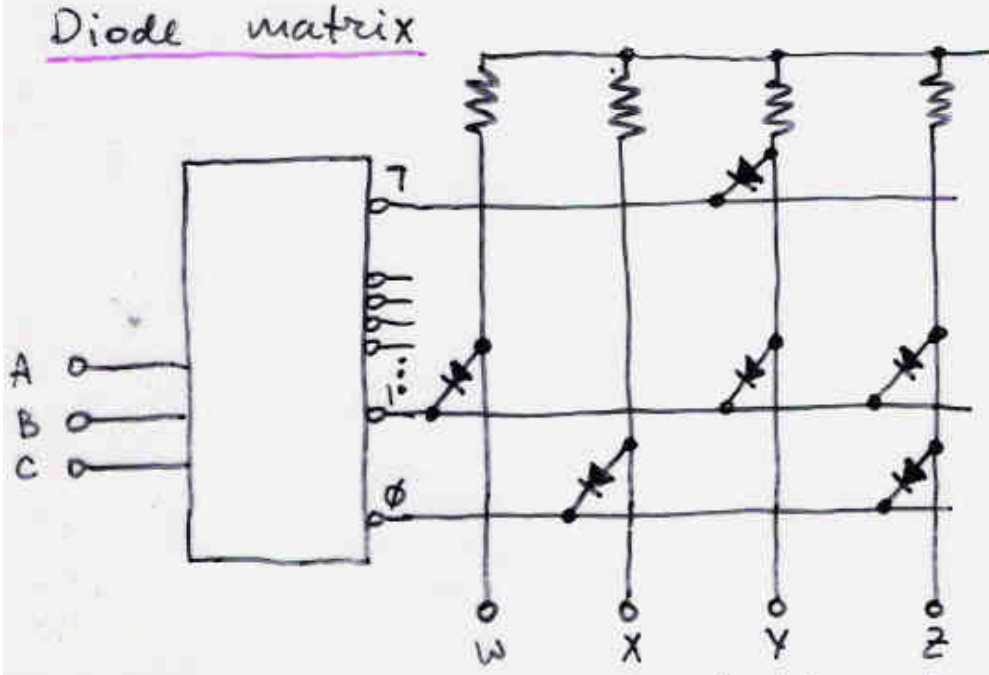


E.g. "1" = b & c
"2" = a & b & g & e & d

Truth table:

A_3	A_2	A_1	A_0	a	b	c	d	e	f	g
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
...										

Diode matrix



$(CBA)_{10}$	WXYZ	$(WXYZ)_{10}$
0	1010	= 10
1	0100	= 4
⋮		
7	1101	= 13

(WXYZ) = "contents" at "address" (CBA)

⇒ a look-up table, permanent. = Read-Only Memory (ROM)