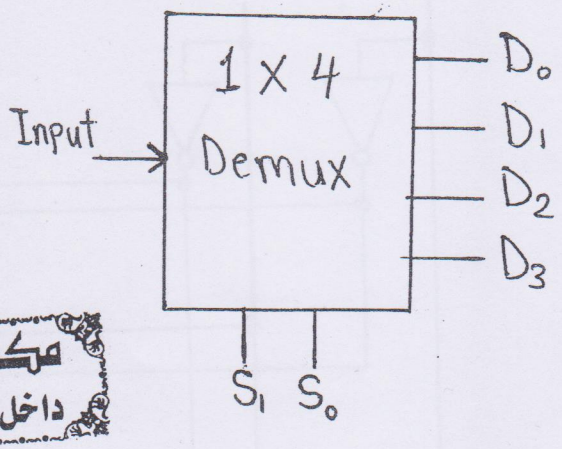
Demultiplexer (One to Many)

The Demultiplexer is the reverse of the Multiplexer where it recieves the input information on a single line and transmits this information on one of the 2^n possible output lines ($1\text{-to-}2^n$).



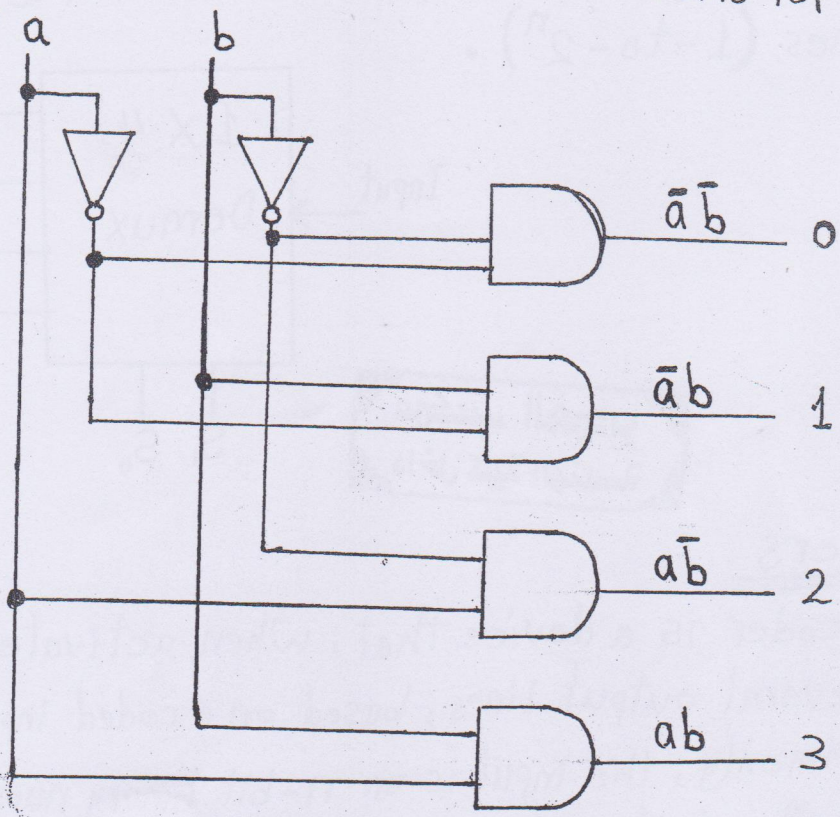
Decoders

A decoder is a device that, when activated, selects one of several output lines, based on a coded input signal. Most commonly, the input is an n-bit binary number, and there are 2^n output lines.

The truth table for a two-input (four-output) decoder is shown below. The inputs are treated as a binary number and the output selected is made active. This decoder just consists of an AND gate for each output, plus NOT gates to invert the inputs.

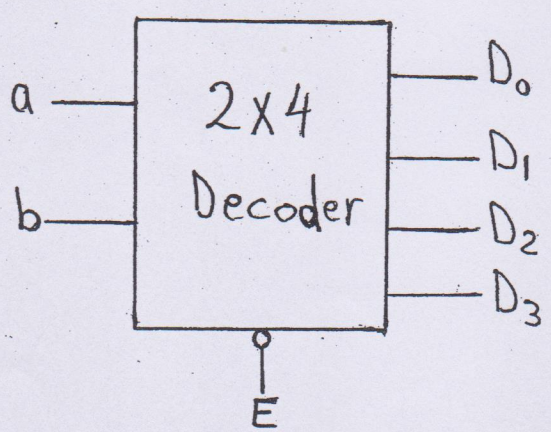
a	b	0	1	2	3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

The block diagram is given below. Output 0 is just $\bar{a}\bar{b}$; output 1 is $\bar{a}b$; output 2 is $a\bar{b}$; and output 3 is ab. Each output corresponds to one of the minterms for a two-variable function.



Most decoders also have one or more enable inputs. When such an input is active, the decoder behaves as described. When it is inactive, all of the outputs of the decoder are inactive. In most systems with a single enable input (not just decoders), that input is active low.

E	a	b	D_0	D_1	D_2	D_3
1	X	X	0	0	0	0
0	0	0	1	0	0	0
0	0	1	0	1	0	0
0	1	0	0	0	1	0
0	1	1	0	0	0	1

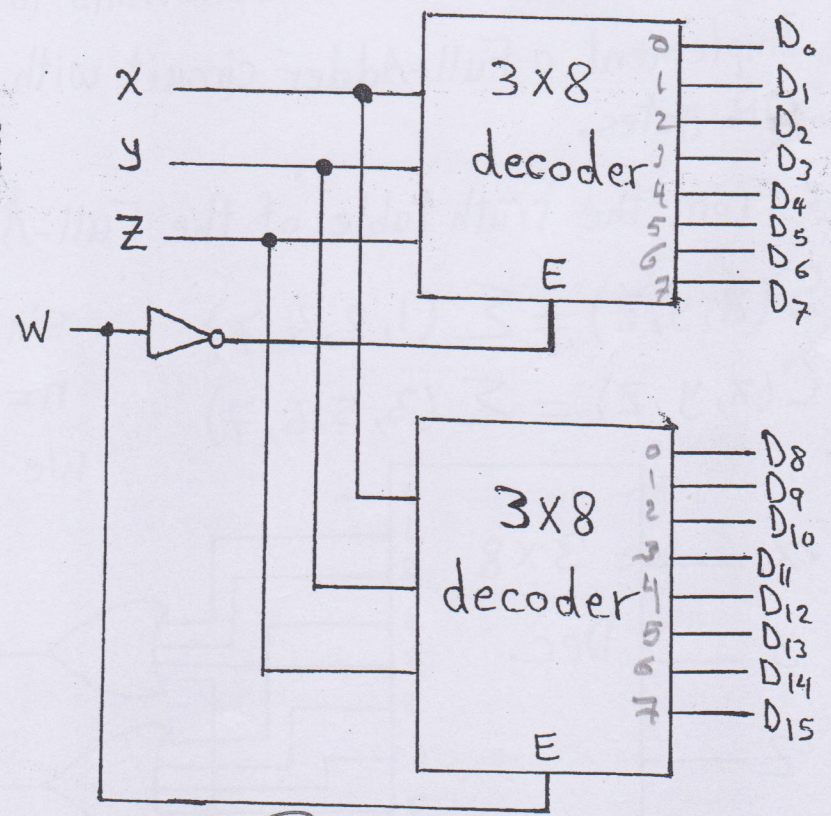


Larger decoders can be built; three-input, eight-output as well as four-input, 16-output decoders are commercially available. The limitation on size is based on the number of connections to the integrated circuit chip that is required. A three-input decoder uses 11 logic connections (three inputs and eight outputs) in addition to two power connections and one or more enable inputs.

One application of decoders is to select one of many devices, each of which has a unique address. The address is the input to the decoder; one output is active, to select the one device that was addressed. Sometimes, there are more devices than can be selected with a single decoder.

Ex: Use two 3-to-8 decoders with enable inputs to form a 4-to-16 line decoders.

Soln



Boolean Function Implementation

A decoder provides the 2^n minterms of n input variables. Since any Boolean function can be expressed in sum of minterms, one can use a decoder to generate the minterms and an external OR gate to form the logical sum. In this way, any combinational circuit with n inputs and m outputs can be implemented with an n-to-2^n-line decoder and m OR gates.

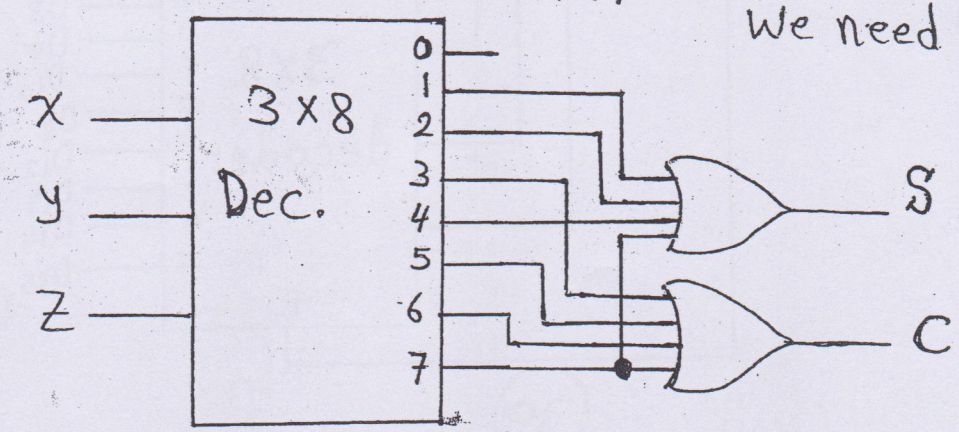
The procedure for implementing a combinational circuit by means of a decoder and OR gates requires that the Boolean function for the circuit is expressed in sum of minterms. This form can be easily obtained from the truth table or by expanding the functions to their Sum of Minterms.

Ex: Implement a Full-Adder circuit with a decoder and two OR gates.

Soln: From the truth table of the Full-Adder we can find

$$S(x,y,z) = \sum(1,2,4,7)$$
$$C(x,y,z) = \sum(3,5,6,7)$$

Since there are 3 inputs, $n=3$, $m=2^n=8$.
We need 3-to-8 decoder



Seven-Segment Decoders

Figure A, shows a seven-segment indicator; i.e., seven LEDs labeled a through g. By forward-biasing different LEDs, we can display the digits 0 through 9 (see Figure B). For instance to display a 0, we need to light up segments a, b, c, d, e, and f. To light up a 5, we need segments a, c, d, f, and g.

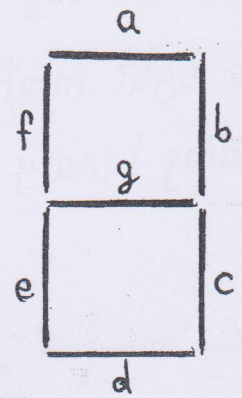


Figure A

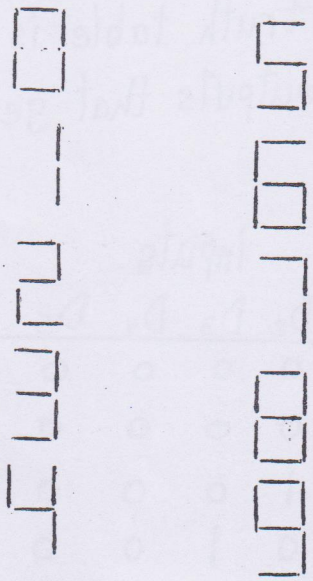
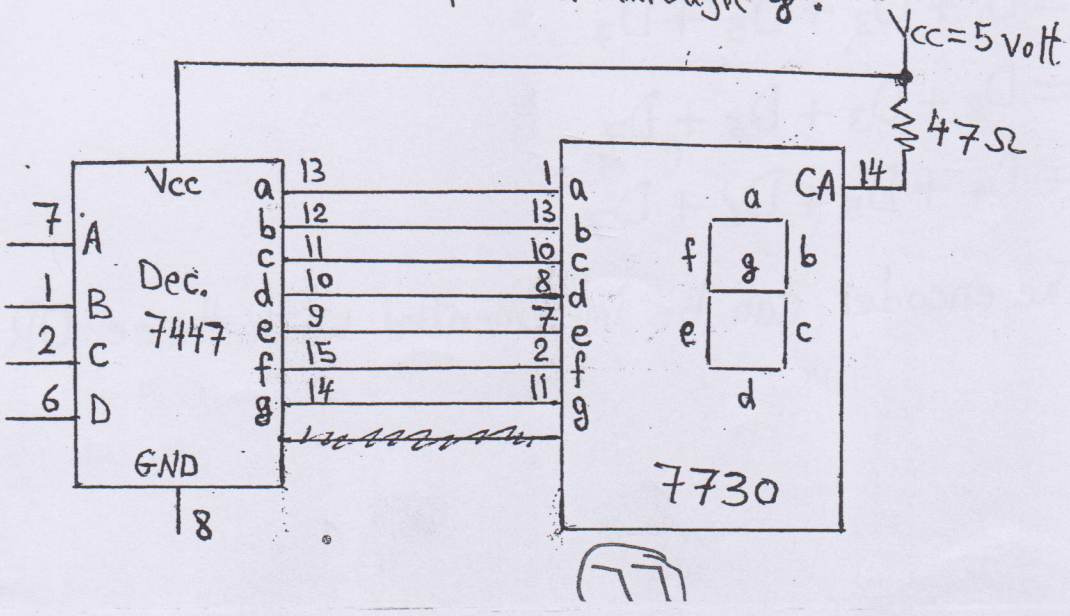


Figure B

BCD-to-seven-segment decoder/driver. It has four inputs for the BCD digit. Input D is the Most significant and input A the least significant. The 4-bit BCD digit is converted to a seven-segment code with outputs a through g.



Encoders

An Encoder is a digital circuit that performs the inverse operation of a decoder. An encoder has 2^n (or fewer) input lines and n output lines. The output lines generate the binary code corresponding to the input value.

An example of an encoder is the octal-to-binary encoder whose truth table is given below. It has eight inputs and three outputs that generate the corresponding binary number.

Inputs								Outputs		
D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	x	y	z
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

$$z = D_1 + D_3 + D_5 + D_7$$
$$y = D_2 + D_3 + D_6 + D_7$$
$$x = D_4 + D_5 + D_6 + D_7$$

The encoder can be implemented with three OR gates.

Read-Only Memory (ROM)

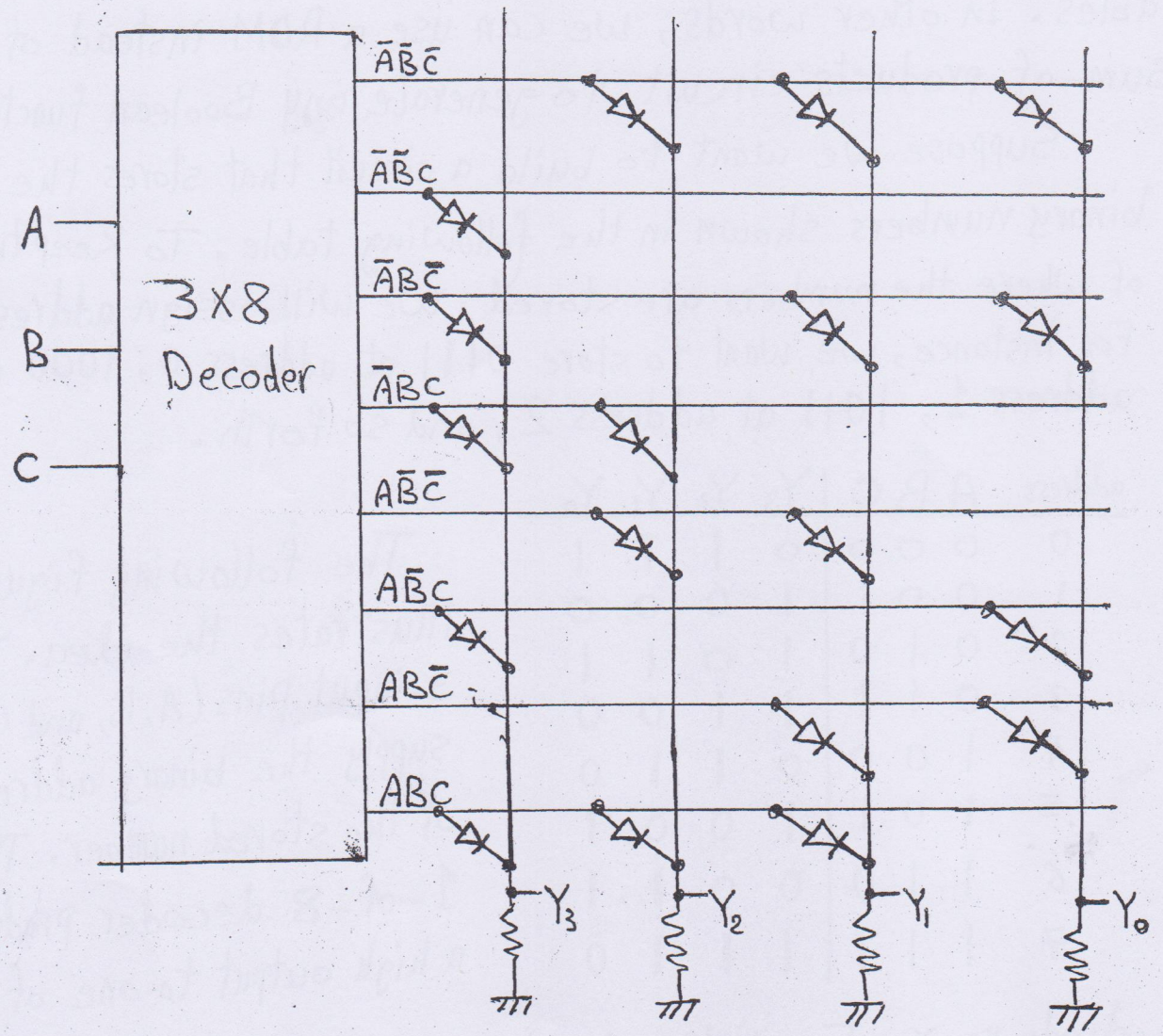
It is an IC that can store thousands of binary numbers representing computer instructions and data. Some of the smaller ROMs are also used to implement truth tables. In other words, we can use a ROM instead of sum-of-products circuit to generate any Boolean function.

Suppose we want to build a circuit that stores the binary numbers shown in the following table. To keep track of where the numbers are stored, we will assign addresses. For instance, we want to store 0111 at address 0, 1000 at address 1, 1011 at address 2, and so forth.

address	A	B	C	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	1	1	1
1	0	0	1	1	0	0	0
2	0	1	0	1	0	1	1
3	0	1	1	1	1	0	0
4	1	0	0	0	1	1	0
5	1	0	1	1	0	0	1
6	1	1	0	0	0	1	1
7	1	1	1	1	1	1	0

The following figure illustrates the idea. The 3-input pins (A, B, and C) supply the binary address of the stored number. Then 1-of-8 decoder produces a high output to one of the diode rows. For instance, if $ABC = 100$, the 1-of-8 decoder applies a high voltage to the $A\bar{B}\bar{C}$ line, and the ROM output is $Y_3Y_2Y_1Y_0 = 0110$, if you change the binary address to $ABC = 110$, the ROM output change to $Y_3Y_2Y_1Y_0 = 0011$. n inputs can select 2^n memory locations (stored numbers).

For instance, we need 3 address lines to access 8 memory locations, 4 address lines for 16 memory locations, 8 address lines for 256 memory locations, and so on.



A binary number is somtimes called a **word**. In a compute binary numbers or words, represent instructions, alphabet letters, decimal numbers, etc. The circuit given in the above Figure is a 32-bit ROM organized as 8 words with 4 bits at each address (an 8x4 ROM). A commercially available ROM (TTL ROMs) are:

7488: 256 bits organized as 32x8
74187: 1024 bits organized as 256x4
74S370: 2048 bits organized as 512x4

If you want to store bytes (words with 8 bits), then you can parallel the 4-bit ROMs. For example, two parallel 74187s can store 256 words of 8 bits each.

Boolean Functions Implementation

If you start with a truth table like the previous one. There are four outputs: Y_3, Y_2, Y_1, and Y_0. A sum-of-products solution would lead to four AND-OR circuits.

$$Y_3 = \bar{A}\bar{B}C + \bar{A}B\bar{C} + \bar{A}BC + A\bar{B}C + ABC$$
$$Y_2 = \bar{A}\bar{B}\bar{C} + \bar{A}BC + A\bar{B}\bar{C} + ABC$$
$$Y_1 = \bar{A}\bar{B}\bar{C} + \bar{A}B\bar{C} + A\bar{B}\bar{C} + AB\bar{C} + ABC$$
$$Y_0 = \bar{A}\bar{B}\bar{C} + \bar{A}B\bar{C} + A\bar{B}C + AB\bar{C}$$

The ROM solution is different. With a ROM you have to store the binary numbers of the previous table at the indicated addresses. When this is done, the ROM given in the above figure is equivalent to a sum-of-products circuit.