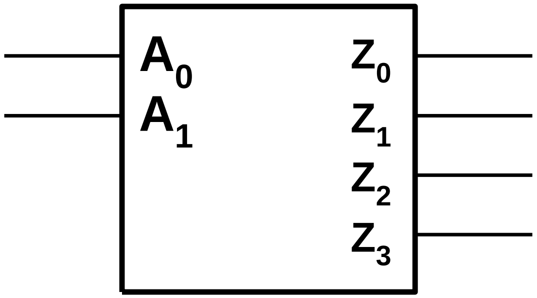
Combinational Logic II

Outline

- Decoders
- Encoders
- Multiplexers
- Demultiplexers

Decoders

- A n-to-2^n decoder takes an n-bit input and produces 2^n outputs. The n inputs represent a binary number that determines which of the 2^n outputs is *uniquely* true.
- The truth table for the 2 to 4 line decoder is given on the right
    - The 2-bit input is called A_1A_0, and the four outputs are Z_0-Z_3.
    - If the input is the binary number of *i*, then the corresponding output Z_i is uniquely true.
    - For instance, if the input A_1A_0 = 10 (decimal 2), then output Z_2 is true, and Z_0, Z_1, Z_3 are all false.



A_1	A_0	Z_0	Z_1	Z_2	Z_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

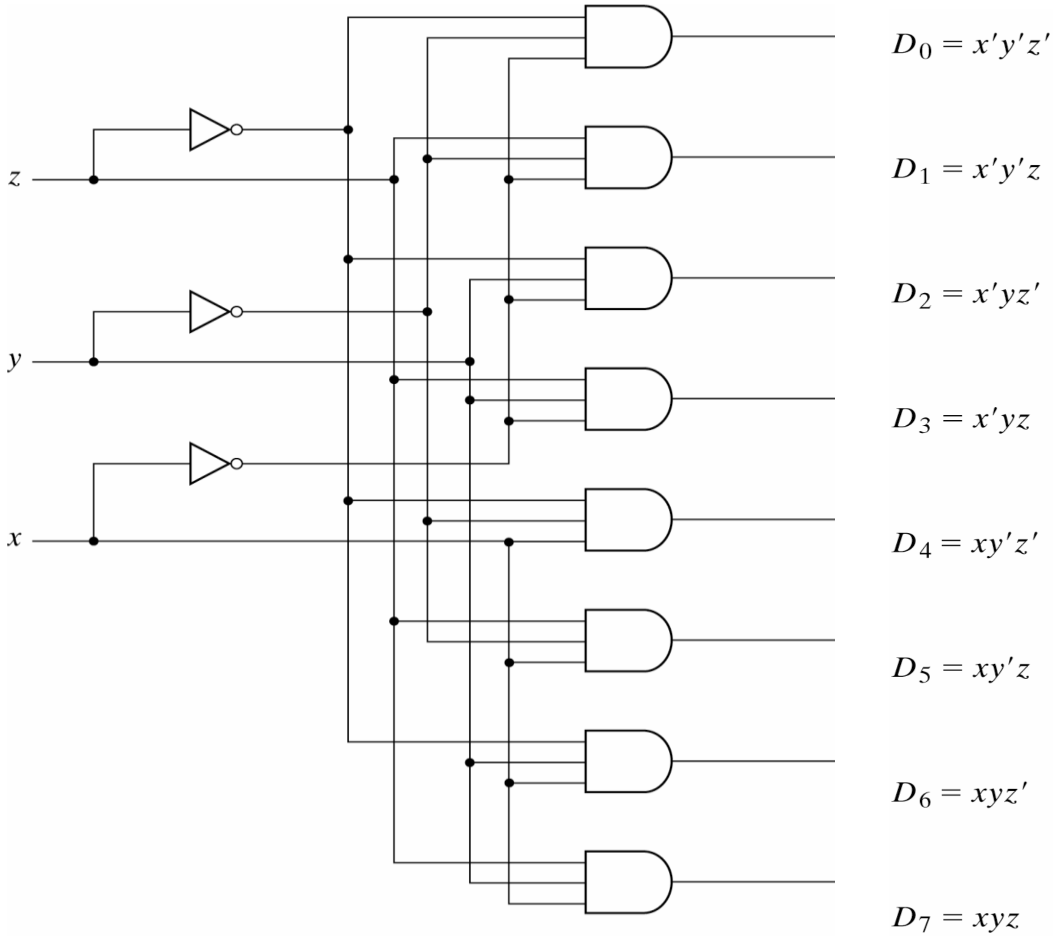
Decoders

- The truth table for the 3 to 8 line decoder is similar to that of the 2 to 4 line decoder.

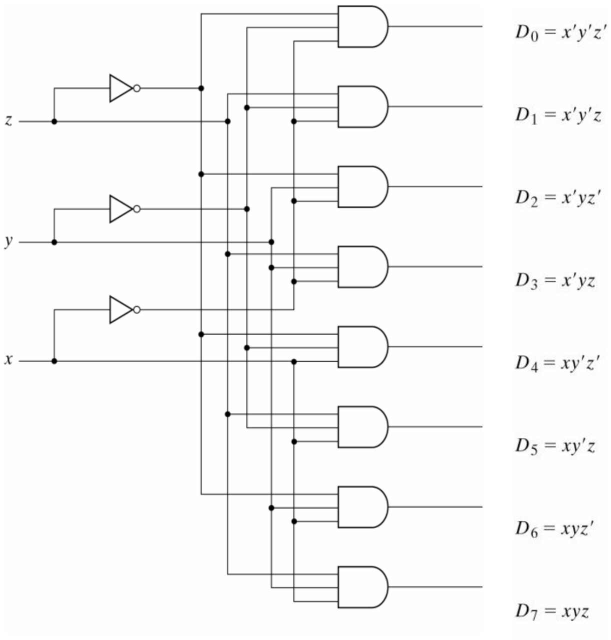
X	Y	Z	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

Decoders

- From the truth table for the 3 to 8 line decoder the implementation on the right can be drawn.
- The 3 to 8 line decoder is widely used, typical applications include *memory selection* and *keyboard scanning*.



Decoders



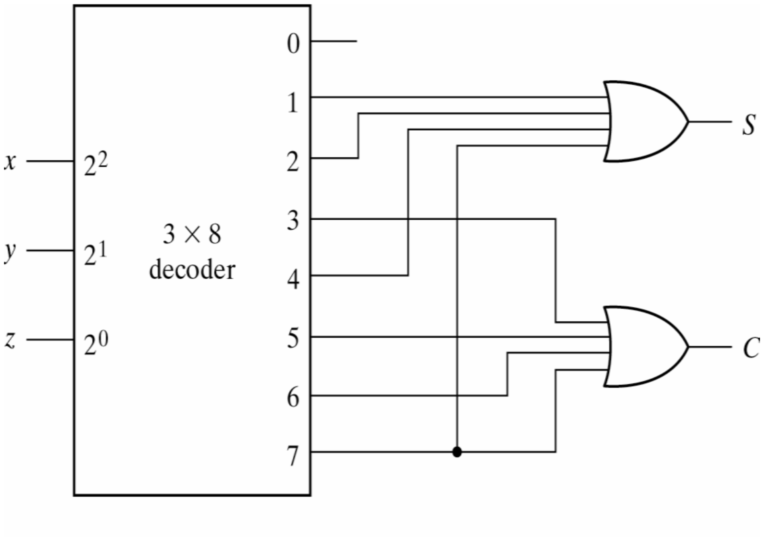
- If we look at each output term of the Decoder, it looks familiar? It is "Minterm"!
- Decoders are sometimes called Minterm Generators:
  - For each of the input combinations, exactly one output is true.
  - Each output equation contains all of the input variables.
- This means that if you have a sum of minterms equation for a logic function, you can easily use a decoder (a minterm generator) with an "OR" gate to implement that function.

Decoders

- For example, from the full adder truth table we have a sum and carry

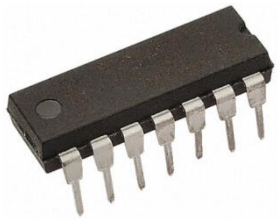
$$S = \overline{X}.\overline{Y}.Z + \overline{X}.Y.\overline{Z} + X.\overline{Y}.\overline{Z} + X.Y.Z$$
$$C = \overline{X}.Y.Z + X.\overline{Y}.Z + X.Y.\overline{Z} + X.Y.Z$$

- An off-the-shelf 3 to 8 line decoder can be used to implement the full adder.



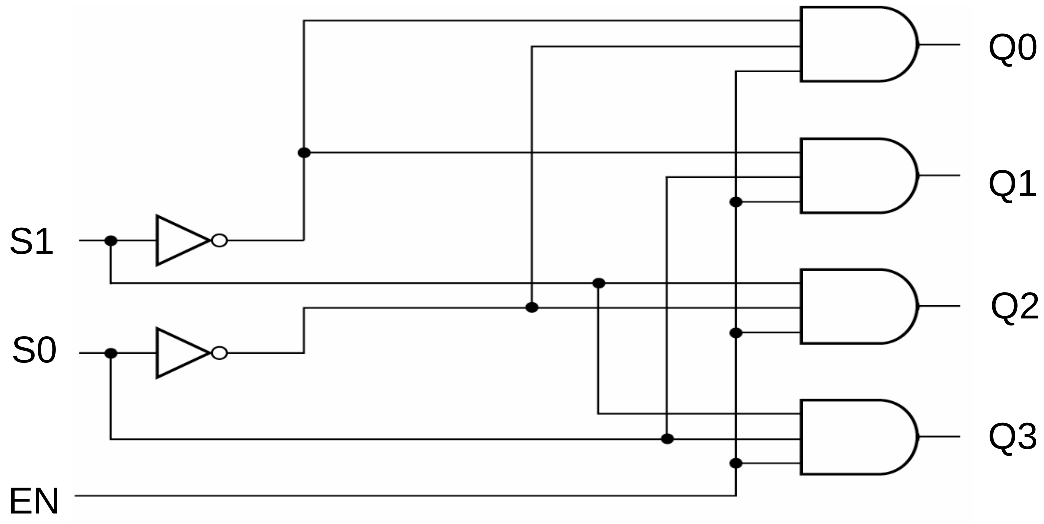
X	Y	Z	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

C(X,Y,Z) = Σm(3,5,6,7)
S(X,Y,Z) = Σm(1,2,4,7)



Enable Inputs

- Many devices have an additional enable input, which is used to “activate” or “deactivate” the device
  - EN=1 “activates” the decoder, so it behaves as specified earlier. Exactly one of the outputs will be 1.
  - EN=0 “deactivates” the decoder. *All* of the decoder’s outputs are 0.
- We can include this additional input in the decoder:



Enable Inputs

- The truth table of a decoder with EN input:

EN	S1	S0	Q0	Q1	Q2	Q3
0	0	0	0	0	0	0
0	0	1	0	0	0	0
0	1	0	0	0	0	0
0	1	1	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

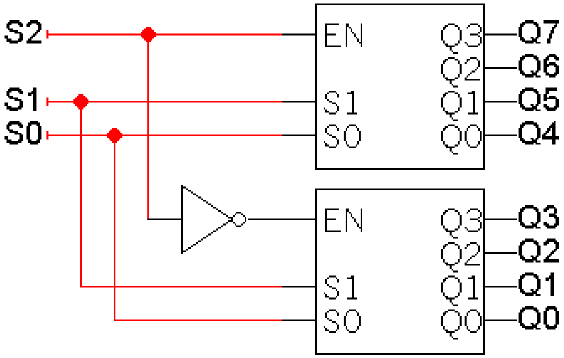
EN	S1	S0	Q0	Q1	Q2	Q3
0	x	x	0	0	0	0
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1

In this table, when EN=0, the outputs are always 0, regardless of inputs S1 and S0. We can abbreviate the table by writing x's in the input columns for S1 and S0.

Building a larger decoder

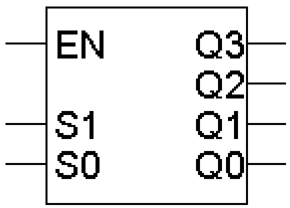
- You could build a larger decoder directly from the truth table and equations as shown earlier. Another way to design a larger decoder is to break it into smaller pieces.
- For example, we could build a 3-to-8 decoder using two 2-to-4 decoders.
  - When S2 = 0, outputs Q0-Q3 are generated as in a 2-to-4 decoder.
  - When S2 = 1, outputs Q4-Q7 are generated as in a 2-to-4 decoder.

S2	S1	S0	Q0	Q1	Q2	Q3	Q4	Q5	Q6	Q7
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



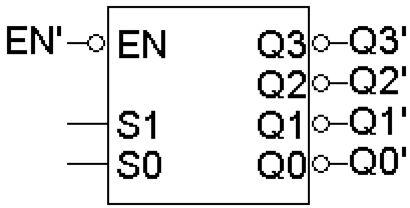
Variation of Standard Decoder

- The decoders we've seen so far are active-high decoders.



EN	S1	S0	Q0	Q1	Q2	Q3
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

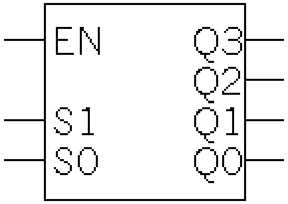
- An active-low decoder is the same thing, but with an inverted EN input and inverted outputs.



EN	S1	S0	Q0	Q1	Q2	Q3
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	0
1	x	x	1	1	1	1

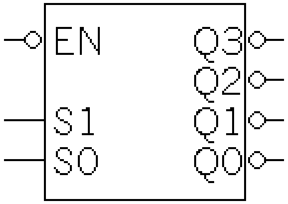
Active-low Decoders

- Active-high decoders generate *minterms*, as we've already seen.



$$
\begin{aligned}
Q3 &= S1\,S0 \\
Q2 &= S1\,S0' \\
Q1 &= S1'\,S0 \\
Q0 &= S1'\,S0'
\end{aligned}
$$

- The output equations for an active-low decoder are similar, yet somehow different, see below:

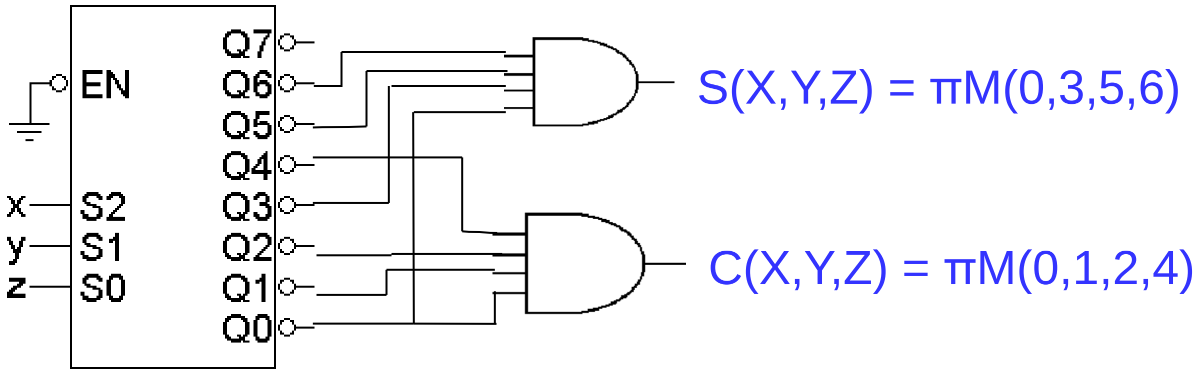


$$
\begin{aligned}
Q3' &= (S1\,S0)' &&= S1' + S0' \\
Q2' &= (S1\,S0')' &&= S1' + S0 \\
Q1' &= (S1'\,S0)' &&= S1 + S0' \\
Q0' &= (S1'\,S0')' &&= S1 + S0
\end{aligned}
$$

- It turns out that active-low decoders generate *maxterms*.

Active-low Decoders

- So we can use active-low decoders to implement arbitrary functions with a *product of maxterms*.
- For example, we can implement the full adder using an active-low decoder.



- The “ground” symbol connected to EN represents logical 0, so this decoder is always enabled.
- Remember that you need an AND gate for a product of sums.

Encoders

- An encoder is a combinational logic circuit which performs the inverse operation of a decoder, i.e. an encoder converts $m \leq 2^n$ input lines into n output lines.
- The truth table of the 8 to 3 line encoder is the inverse of the 3 to 8 line decoder.

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

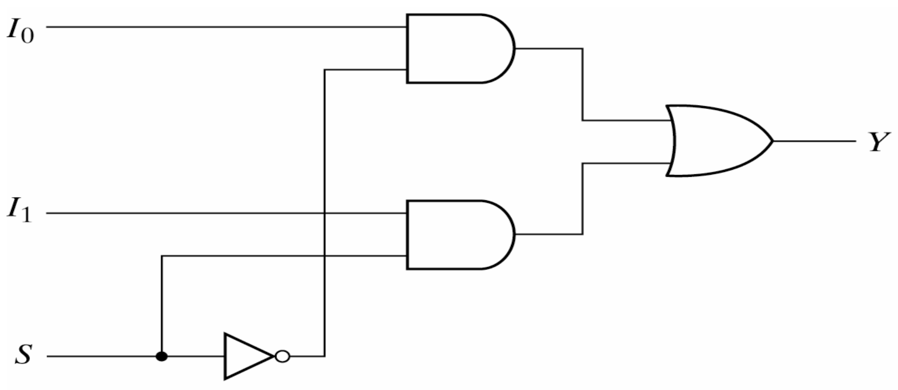
Priority Encoders

- In priority encoders, the input having the highest priority will take precedence.
- 8-to-3 Priority Encoder on the right

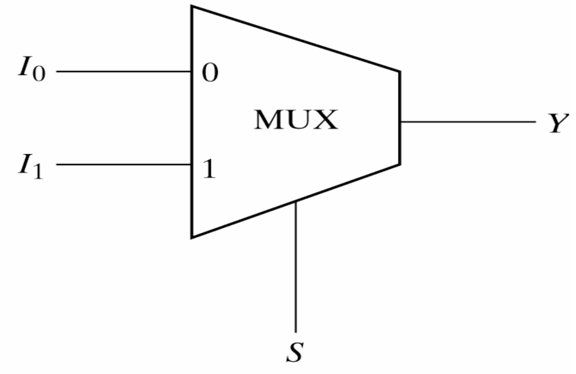
D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	A	B	C
1	0	0	0	0	0	0	0	0	0	0
x	1	0	0	0	0	0	0	0	0	1
x	x	1	0	0	0	0	0	0	1	0
x	x	x	1	0	0	0	0	0	1	1
x	x	x	x	1	0	0	0	1	0	0
x	x	x	x	x	1	0	0	1	0	1
x	x	x	x	x	x	1	0	1	1	0
x	x	x	x	x	x	x	1	1	1	1

Multiplexers

- A multiplexer operates in a similar way to an encoder. Rather than converting 2^n input lines into n output lines, a multiplexer converts 2^n input lines into a *single* output line. To do this a multiplexer needs an additional n selection lines.
- A 2 to 1 line multiplexer is shown below.



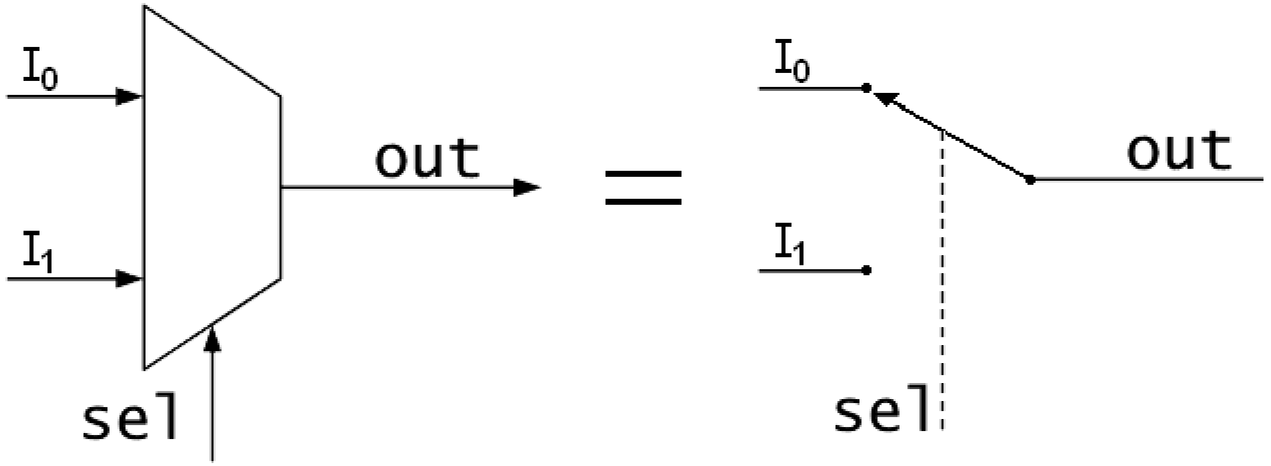
(a) Logic diagram



(b) Block diagram

Multiplexers

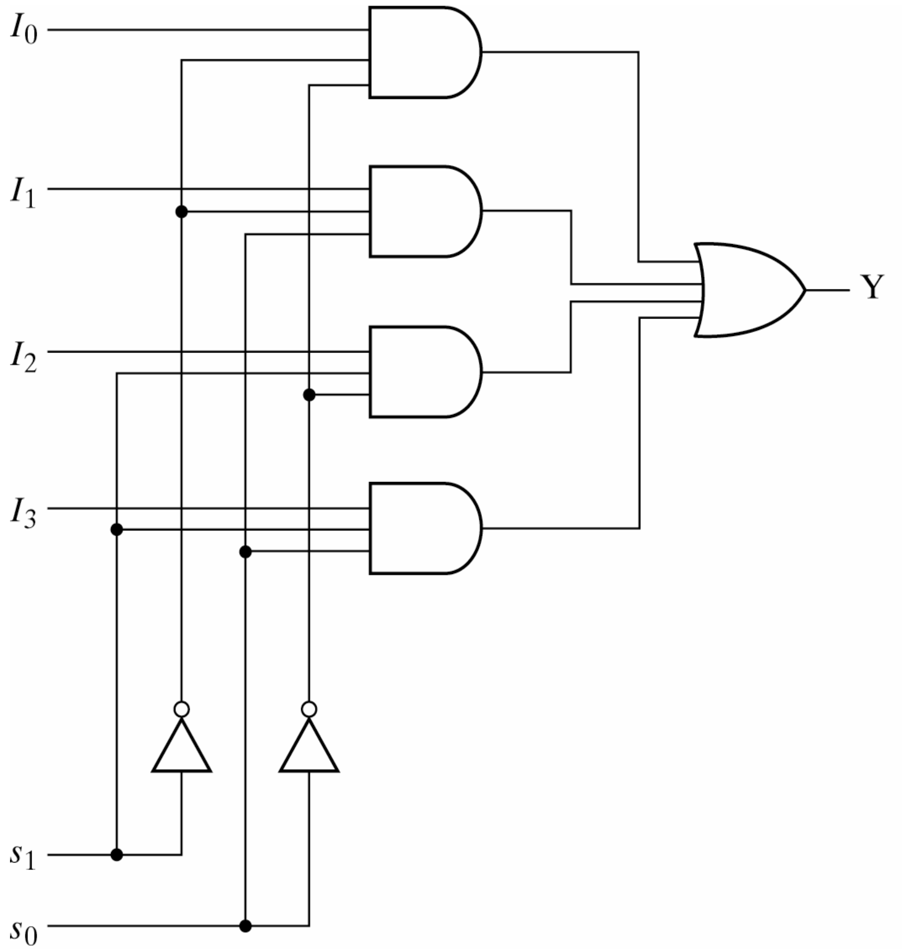
- When the selection line *S* (Sel) is set to '1' the data input I_0 is passed to the output.
- When the selection line *S* (Sel) is set to '0' the data input I_1 is passed to the output.
- The multiplexer is also known as a data selector.



Multiplexers

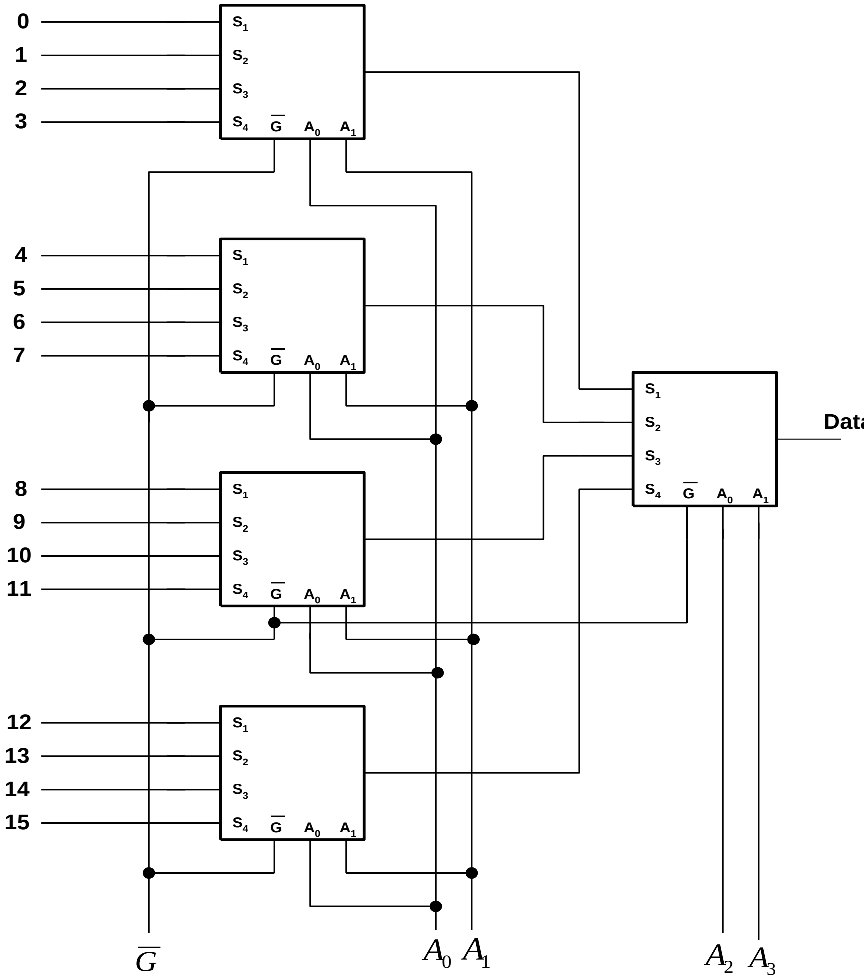
- As the number of input lines increases more selection lines are needed.
- For a 4 to 1 line multiplexer we have a truth table as below and an implementation as on the right.

s_1	s_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3



Multiplexers

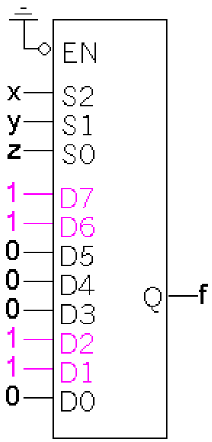
- It can be convenient to use lower order multiplexers to create higher order multiplexers.
- On the right an implementation of a 16 line multiplexer uses five 4 line multiplexers.



Implementing Boolean Function Using Multiplexer

- Multiplexer can be used to implement arbitrary functions. One way to implement a function of *n* variables is to use an *n-to-1* multiplexer:
  - Connect the function's input variables to the selection inputs. These are used to indicate a particular input combination.
  - For each minterm m_i of the function, connect 1 to the data input Di. Each data input corresponds to one row of the truth table.
- For example, let's look at $f(x,y,z) = \Sigma m(1,2,6,7)$.

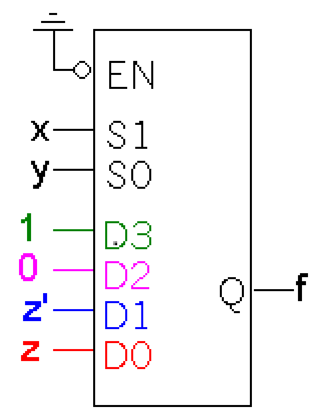
x	y	z	f
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1



- We can actually implement $f(x,y,z) = \Sigma m(1,2,6,7)$ with just a 4-to-1 multiplexer, instead of an 8-to-1.
- <u>Step 1:</u> Find the truth table for the function, and group the rows into pairs. Within each pair of rows, x and y are the same, so f is a function of z only.
  - When xy=00, f=z
  - When xy=01, f=z'
  - When xy=10, f=0
  - When xy=11, f=1

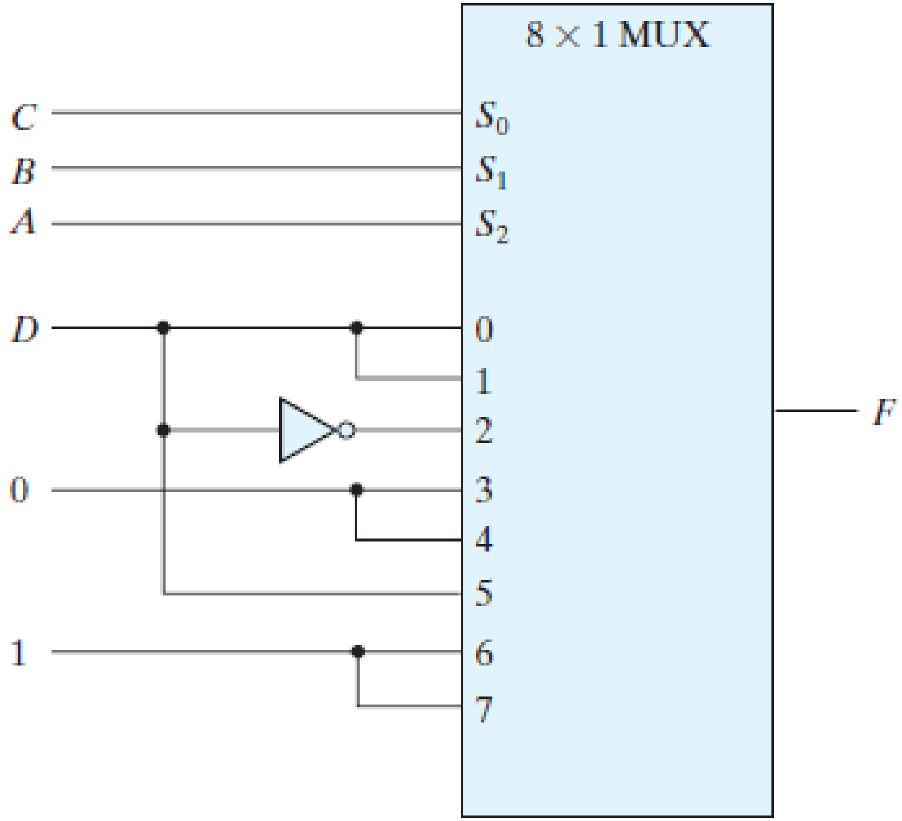
x	y	z	f
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

- <u>Step 2:</u> Connect the first two input variables of the truth table (here, x and y) to the selection bits S1 S0 of the 4-to-1 multiplexer.
- <u>Step 3:</u> Connect the equations above for f(z) to the data inputs D0-D3.



More Examples

A	B	C	D	F	
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	



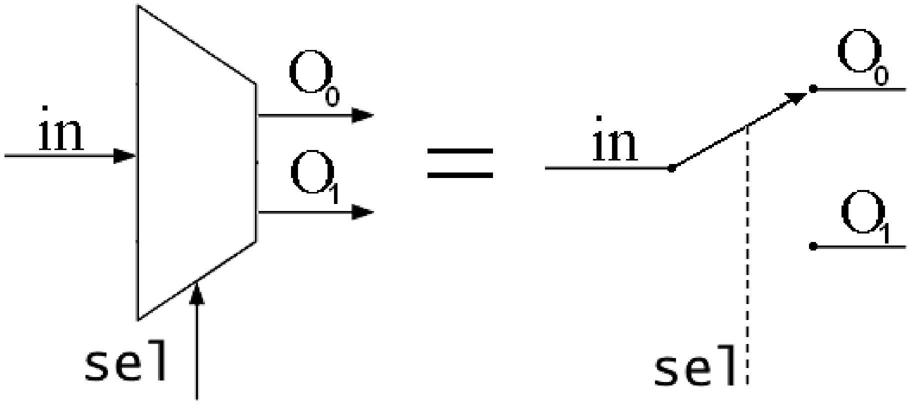
$$F(A,B,C,D) = \sum(1,3,4,11,12,13,14,15)$$

Summary of Implementing Boolean Function Using Multiplexer

- A Boolean function of n variables can be implemented with a multiplexer with n-1 selection inputs and 2^{n-1} inputs.
- The first n-1 variables are connected to the selection inputs of the multiplexer.
- The remaining single variable is used for the data inputs. Each data input can be 0,1, the variable, or the complement of the variable.

Demultiplexers

- Perform the inverse operation of a multiplexer, i.e. converts a single input line into 2^n output lines.
- Again to do this a de-multiplexer needs to have access to the *n* selection line signals used by the multiplexer.
- If the multiplexer and de-multiplexer are only connected by a single data line a scheme is required so that both the multiplexer and de-multiplexer are using the same selection line signals.



Multiplexing and Demultiplexing

- Example: Multiplexing in Communications

- Line between Mux and Demux above represents communications channel e.g. telephone cable, or “free space” in a wireless system...

- This type of system is used in mobile phones. It allows many users to communicate at the same time along the same channel.

Demultiplexers

Truth Table

S_1	S_0	I_0	F_3	F_2	F_1	F_0
0	0	0	0	0	0	0
		1	0	0	0	1

S_1	S_0	I_0	F_3	F_2	F_1	F_0
0	1	0	0	0	0	0
		1	0	0	1	0

S_1	S_0	I_0	F_3	F_2	F_1	F_0
1	0	0	0	0	0	0
		1	0	1	0	0

S_1	S_0	I_0	F_3	F_2	F_1	F_0
1	1	0	0	0	0	0
		1	1	0	0	0