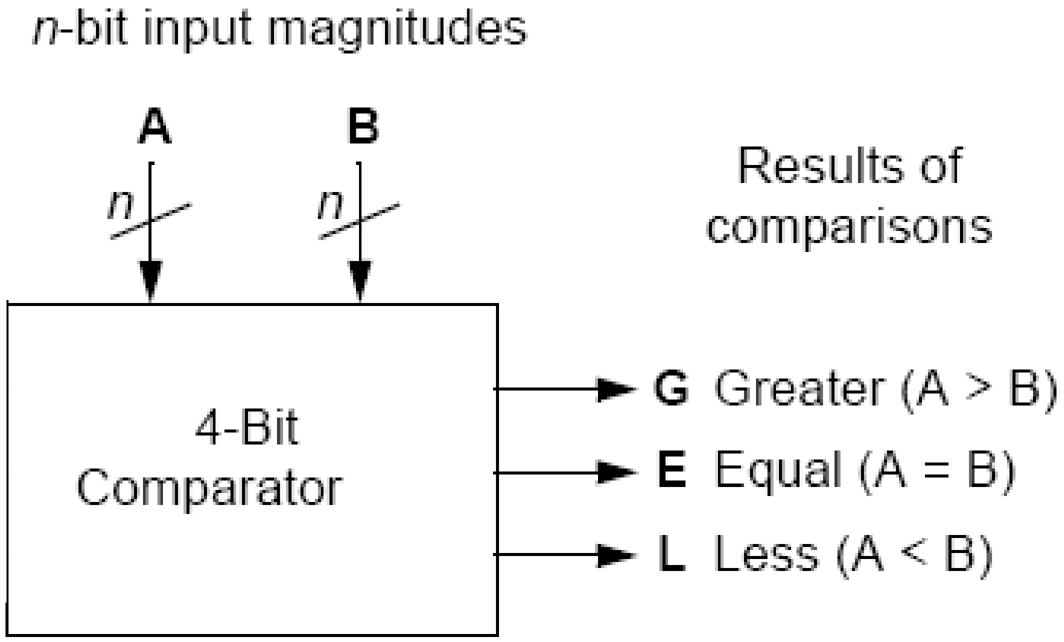
Back to Combinational Logic Ckt design

COMPARATORS
MAGNITUDE COMPARATOR

- Given two *n*-bit magnitudes, A and B, a comparator indicates whether
  - A = B, A > B, or A < B



- The approach is to use the XNOR function (equivalence) on each of the *n*-bits as follows

$$x_i = \mathbf{A}_i\mathbf{B}_i + \overline{\mathbf{A}}_i\overline{\mathbf{B}}_i = \overline{\mathbf{A}_i \oplus \mathbf{B}_i}$$

- The Boolean functions for a 4-bit magnitude comparator is as follows
  - $(\mathbf{A} = \mathbf{B}) = x_3x_2x_1x_0$
  - $(\mathbf{A} > \mathbf{B}) = \mathbf{A}_3\overline{\mathbf{B}_3} + x_3\mathbf{A}_2\overline{\mathbf{B}_2} + x_3x_2\mathbf{A}_1\overline{\mathbf{B}_1} + x_3x_2x_1\mathbf{A}_0\overline{\mathbf{B}_0}$
  - $(\mathbf{A} < \mathbf{B}) = \overline{\mathbf{A}_3}\mathbf{B}_3 + x_3\overline{\mathbf{A}_2}\mathbf{B}_2 + x_3x_2\overline{\mathbf{A}_1}\mathbf{B}_1 + x_3x_2x_1\overline{\mathbf{A}_0}\mathbf{B}_0$

Note: $\mathbf{A}_i\overline{\mathbf{B}}_i$ indicates whether $\mathbf{A}_i > \mathbf{B}_i$, $\overline{\mathbf{A}}_i\mathbf{B}_i$ indicates whether $\mathbf{A}_i < \mathbf{B}_i$, and x_i indicates whether $\mathbf{A}_i = \mathbf{B}_i$.

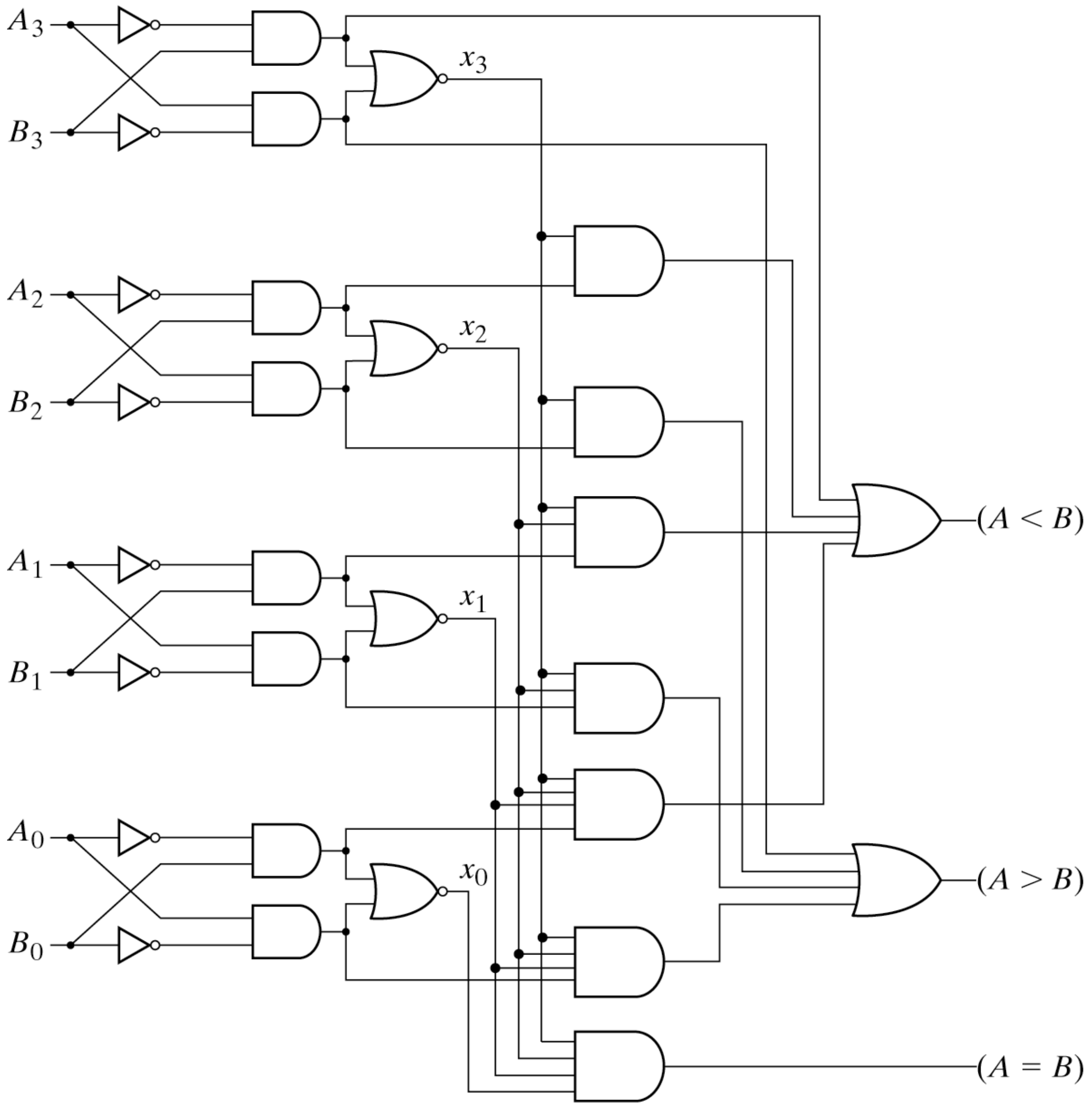
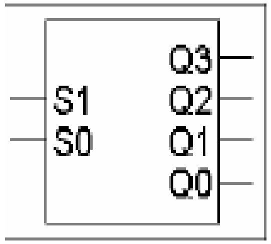


Fig. 4-17 4-Bit Magnitude Comparator

DECODERS

What a decoder does

- A *n*-to-2^n decoder uses its *n*-bit input to determine which of 2^n outputs will be uniquely activated.



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

- Here is a block diagram and truth table for a 2-to-4 decoder.
  - The two-bit input is called S1S0, and the four outputs are Q0-Q3.
  - If the input is the binary number i, then output Qi alone will be true.
- This circuit "decodes" a binary number into a "one-of-four" code.

Building a decoder

- We can use the truth table to derive minimal sum of products equations for each of the four outputs (Q0-Q3), based on the two inputs (S0-S1).

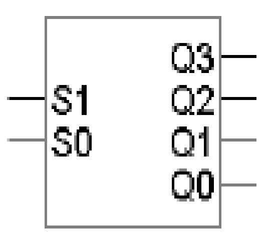
S1	S0	Q0	Q1	Q2	Q3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

- In this case there's not much to be simplified. Here are the equations:

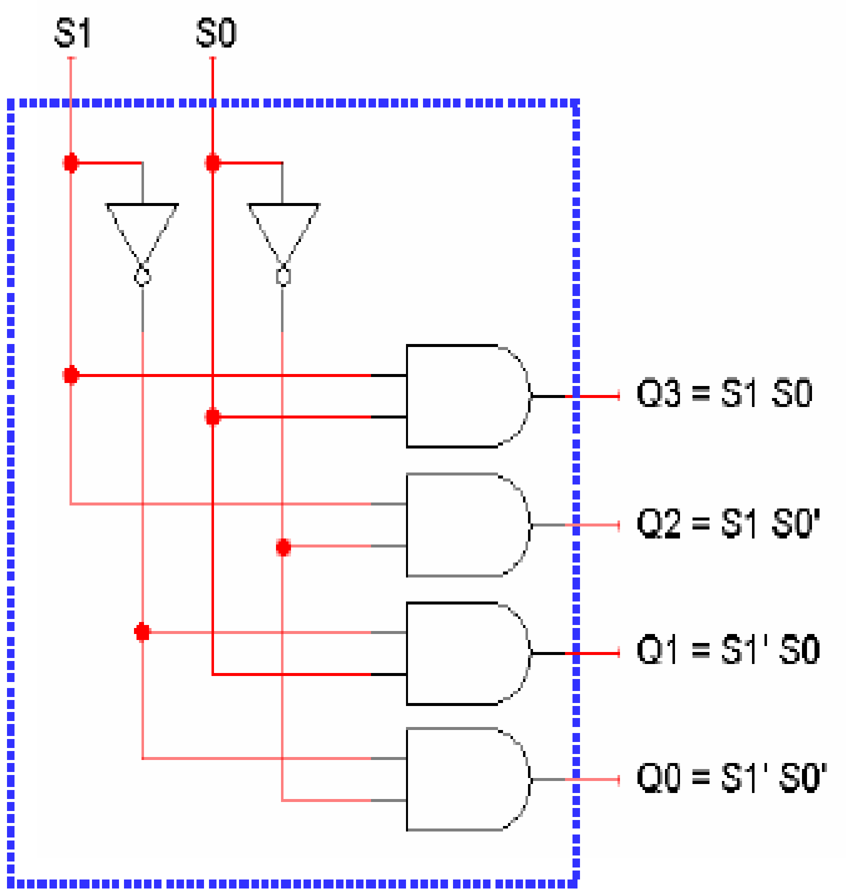
$$Q0 = S1'S0'$$
$$Q1 = S1'S0$$
$$Q2 = S1\ S0'$$
$$Q3 = S1\ S0$$

Decoder circuit diagram

- Here is an implementation of a 2-to-4 decoder.



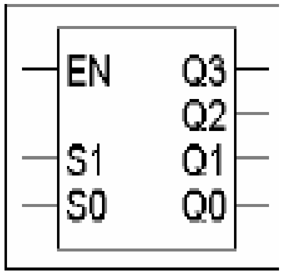
S1	S0	Q0	Q1	Q2	Q3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1



Enable inputs

decoders can include enable inputs.

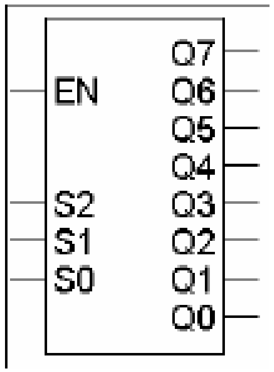
- EN=0 disables the decoder, which by convention means that all of the decoder's outputs are 0.
- EN=1 enables the decoder so that it behaves as specified earlier, with exactly one of the outputs being 1.



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

A 3-to-8 decoder

- Larger decoders are similar. Here is a 3-to-8 decoder.
  - There are three selection inputs S2S1S0, which activate one of eight outputs, Q0-Q7.
  - Again, only one output will be true for any input combination.
- A truth table and output equations for a 3-to-8 decoder (without EN) are given below.



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

Q0 = S2'S1'S0'
Q1 = S2'S1'S0
Q2 = S2'S1 S0'
Q3 = S2'S1 S0
Q4 = S2 S1'S0'
Q5 = S2 S1'S0
Q6 = S2 S1 S0'
Q7 = S2 S1 S0

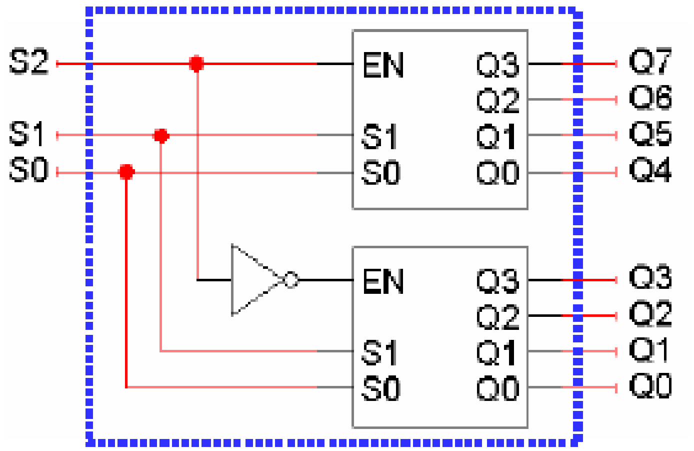
Building a 3-to-8 decoder

- You could build a 3-to-8 decoder from the truth table and MSP equations below, just like we built the 2-to-4 decoder earlier.
- Another way to design a decoder is to break it into smaller pieces.
- Notice some patterns in the table below:
  - 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

Decoder expansion

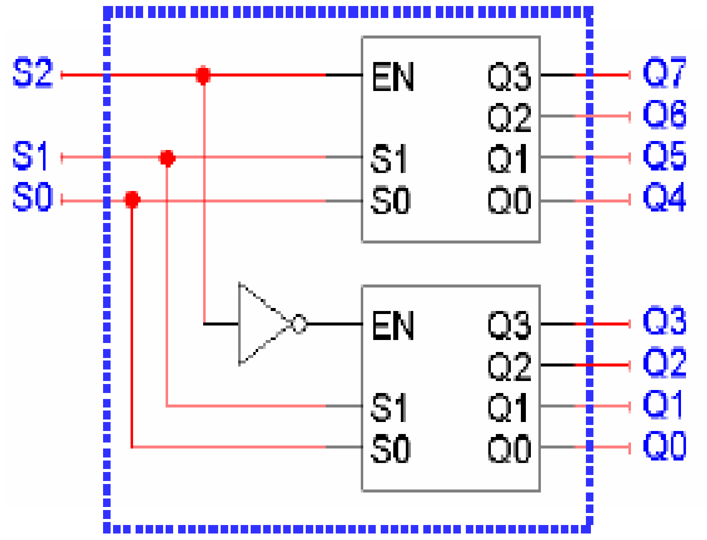
- Here's a 3-to-8 decoder built from two smaller 2-to-4 decoders.
- When S2=0, the bottom 2-to-4 decoder is enabled and generates a 1 for one of outputs Q0, Q1, Q2 or Q3.
- When S2=1, the top 2-to-4 decoder is enabled instead, and a 1 will be output for either Q4, Q5, Q6 or Q7.



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

Modularity

- You could verify that this circuit is a 3-to-8 decoder, by using equations for the 2-to-4 decoders to derive equations for the 3-to-8.



So what good is a decoder?

- Do the truth table and equations look familiar?

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

$Q0 = S1'S0'$
$Q1 = S1'S0$
$Q2 = S1\ S0'$
$Q3 = S1\ S0$

- Decoders are sometimes called minterm generators.
  - For each input combination, exactly one output is true.
  - Each output equation contains all of the input variables.
- This means that you can easily use a decoder, or a minterm generator, to implement any sum of minterms expression.

Example: addition

- A truth table and sum of minterm equations for C and S are shown below.

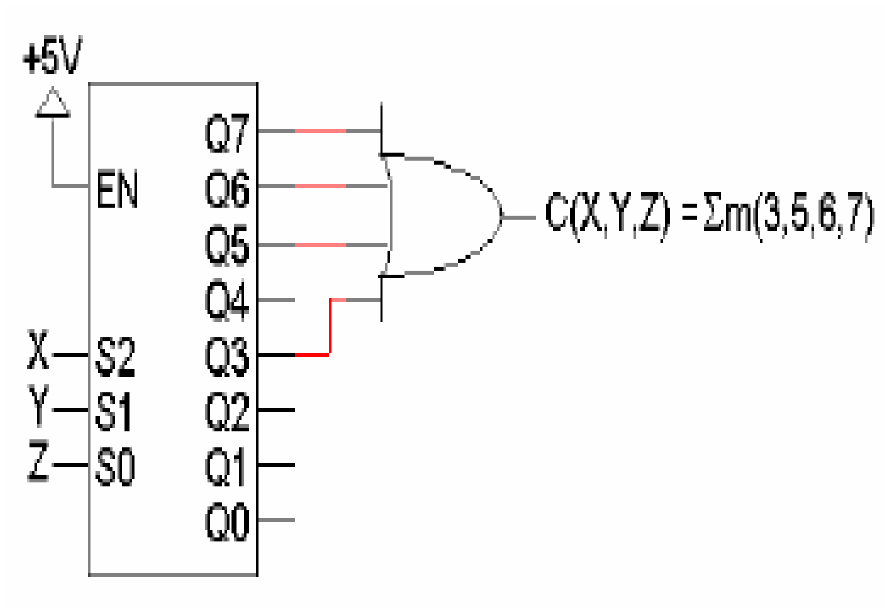
X	Y	Z	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

$C(X,Y,Z) = \Sigma m(3,5,6,7)$
$S(X,Y,Z) = \Sigma m(1,2,4,7)$

- Today we'll implement these two functions using 3-to-8 decoders.

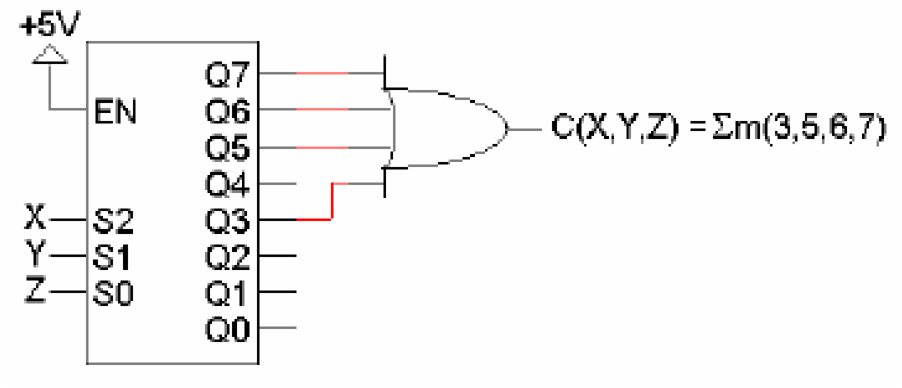
Implementing functions with decoders

- Here, a 3-to-8 decoder implements C as a sum of minterms.



- If XYZ is 011, 101, 110 or 111, then one of the decoder outputs Q3, Q5, Q6 or Q7 will be true, and the output C(X,Y,Z) will also be true.

Verifying our adder



- C is the sum of decoder outputs Q3, Q5, Q6 and Q7.

$$C = Q3 + Q5 + Q6 + Q7$$

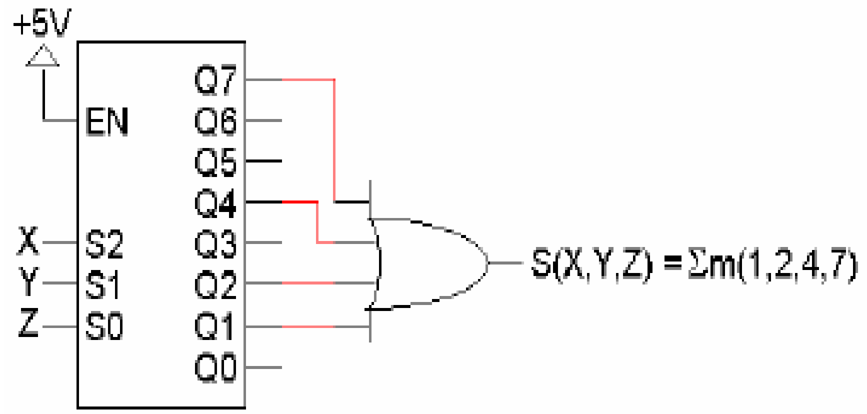
- We know how the decoder's outputs are generated from its inputs.

$$C = S2'S1S0 + S2S1'S0 + S2S1S0' + S2S1S0$$

- We can substitute XYZ for S2S1S0, resulting in a sum of minterms

$$C = X'YZ + XY'Z + XYZ' + XYZ$$

Decoder-based sum

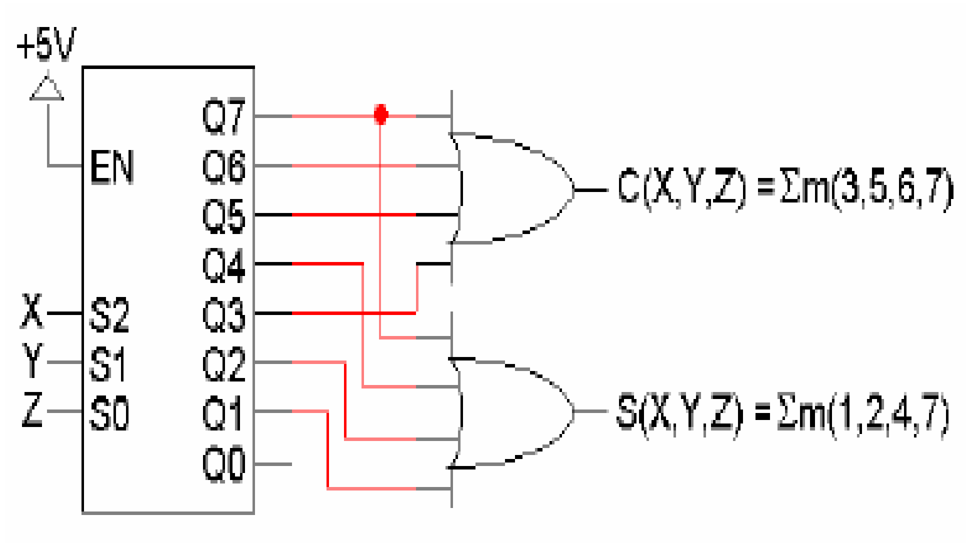


- If XYZ is 001, 010, 100 or 111, then one of decoder outputs Q1, Q2, Q4 or Q7 will be true, and S(X,Y,Z) will be true as well.
- We can verify this circuit algebraically, just like before.

$$\begin{aligned} S(X,Y,Z) &= Q1 + Q2 + Q4 + Q7 \\ &= S2'S1'S0 + S2'S1\ S0' + S2\ S1'S0' + S2\ S1\ S0 \\ &= X'Y'Z + X'YZ' + XY'Z' + XYZ \\ &= m_1 + m_2 + m_4 + m_7 \end{aligned}$$

Using just one decoder

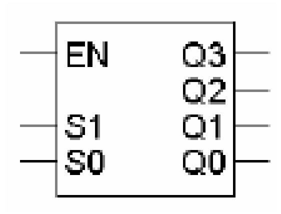
- Since the two functions C and S both have the same inputs, we could use just one decoder instead of two.



- Decoder output Q0 is unused, while Q7 is used multiple times. In general, you can always use circuit outputs as many or as few times as you need.

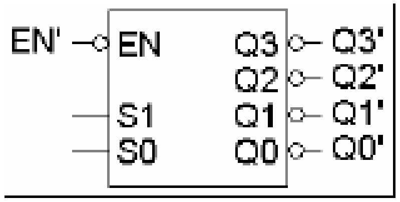
A variation of the standard decoder

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



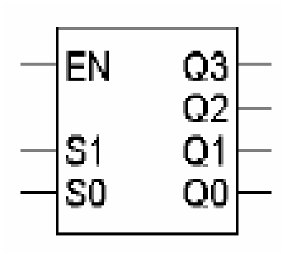
EN	S1	S0	Q0	Q1	Q2	Q3
0	x	x	0	0	0	0
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1

- An active-low decoder is similar, but with an inverted EN input *and* inverted outputs—exactly one of the outputs will be *false*.



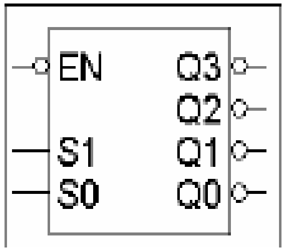
EN'	S1	S0	Q0'	Q1'	Q2'	Q3'
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	0
1	x	x	1	1	1	1

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



Q3 = S1 S0
Q2 = S1 S0'
Q1 = S1'S0
Q0 = S1'S0'

- The output equations for an active-low decoder are mysteriously similar, yet somehow different.

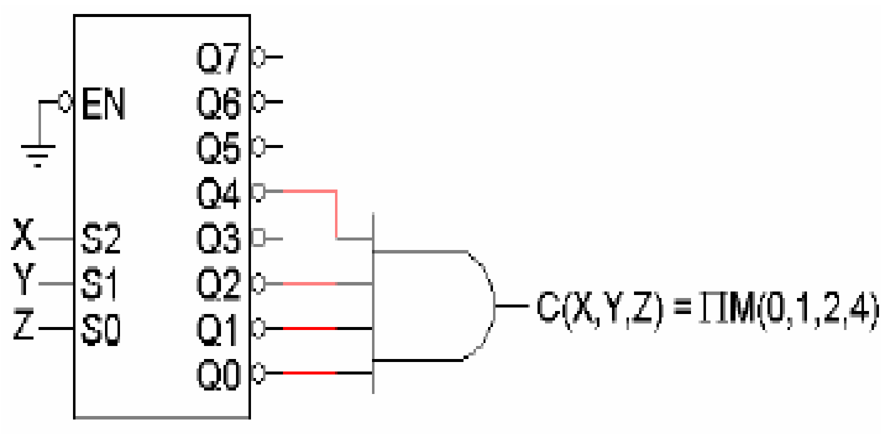


Q3' = (S1 S0)' = S1'+ S0'
Q2' = (S1 S0')' = S1'+ S0
Q1' = (S1'S0)' = S1 + S0'
Q0' = (S1'S0')' = S1 + S0

- It turns out that active-low decoders generate maxterms, so they can implement functions as sums of maxterms.

Active-low decoder example

- Here is the carry function C implemented with an active-low decoder.
  - Recall from our algebra lecture that $\Sigma m(3,5,6,7) = \Pi M(0,1,2,4)$.
  - We need an AND gate to generate a *product* of sums.

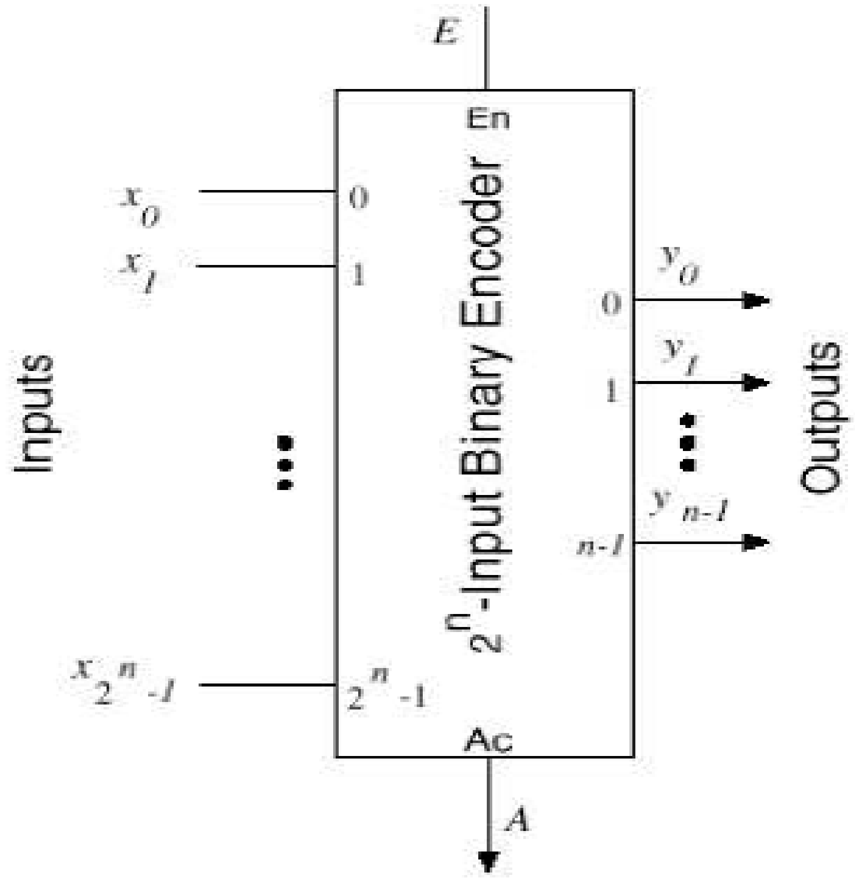


- The "ground" symbol connected to the EN input represents logical 0, so this decoder is always enabled.

Encoders

- An encoder is a digital circuit that performs the inverse operation of a decoder. An encoder has 2^n input lines and n output lines.
- The output lines generate the binary equivalent of the input line whose value is 1.

Encoders (cont.)



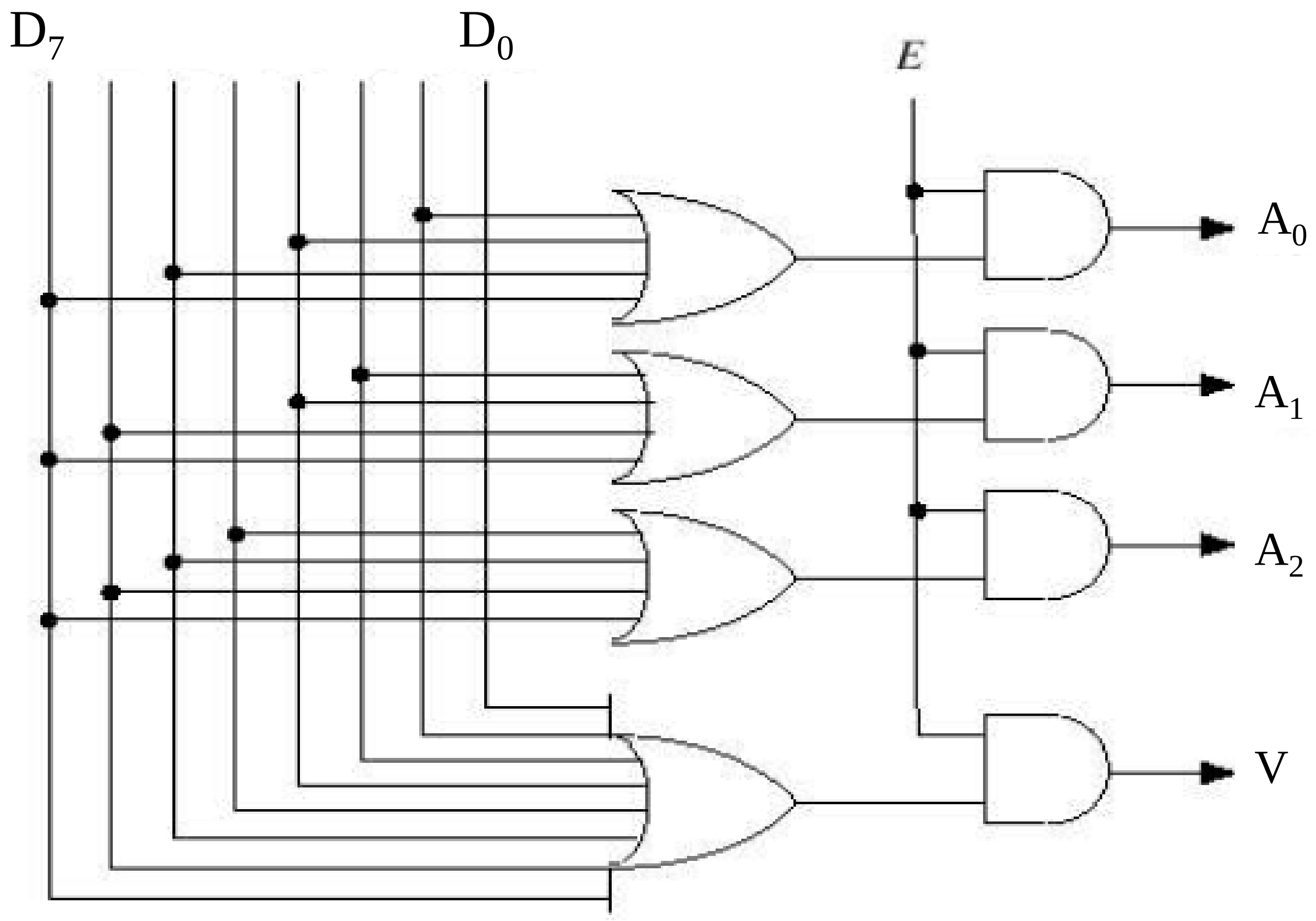
Encoder Example

- Example: 8-to-3 binary encoder (octal-to-binary)

Inputs								Outputs		
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	A_2	A_1	A_0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1

$$A_0 = D_1 + D_3 + D_5 + D_7$$
$$A_1 = D_2 + D_3 + D_6 + D_7$$
$$A_2 = D_4 + D_5 + D_6 + D_7$$

Encoder Example (cont.)



Encoder Design Issues

- There are two ambiguities associated with the design of a simple encoder:
  1. Only one input can be active at any given time. If two inputs are active simultaneously, the output produces an undefined combination (for example, if D_3 and D_6 are 1 simultaneously, the output of the encoder will be 111.
  2. An output with all 0's can be generated when all the inputs are 0's,or when D_0 is equal to 1.

Priority Encoders

- Solves the ambiguities mentioned above.
- Multiple asserted inputs are allowed; one has priority over all others.
- Separate indication of no asserted inputs.

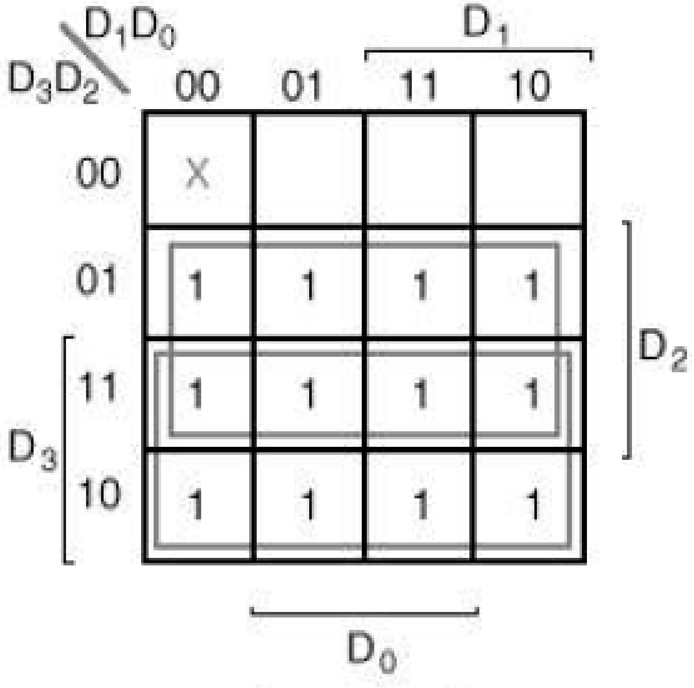
4-to-2 Priority Encoder (cont.)

- The operation of the priority encoder is such that:
- If two or more inputs are equal to 1 at the same time, the input in the highest-numbered position will take precedence.
- *A **valid output indicator***, designated by V, is set to 1 only when one or more inputs are equal to 1. $V = D_3 + D_2 + D_1 + D_0$ by inspection.

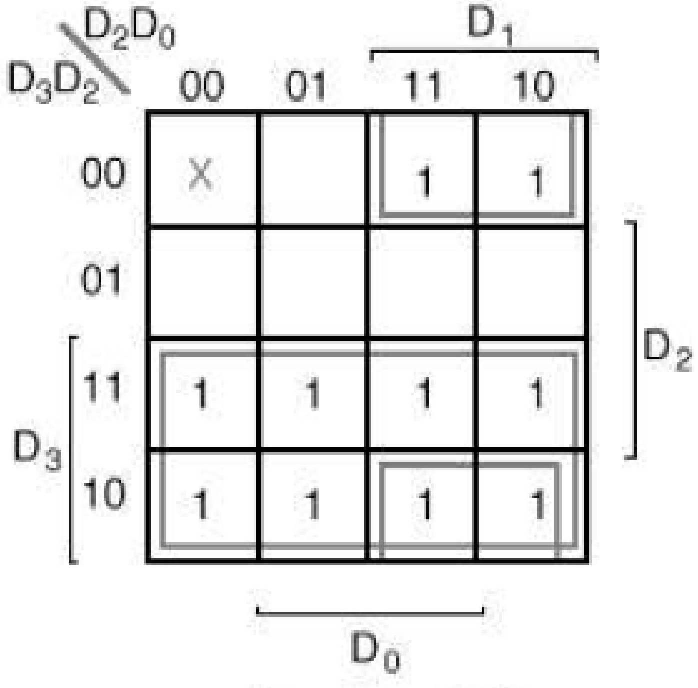
Example: 4-to-2 Priority Encoder Truth Table

Inputs				Outputs		
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

Example: 4-to-2 Priority Encoder K-Maps

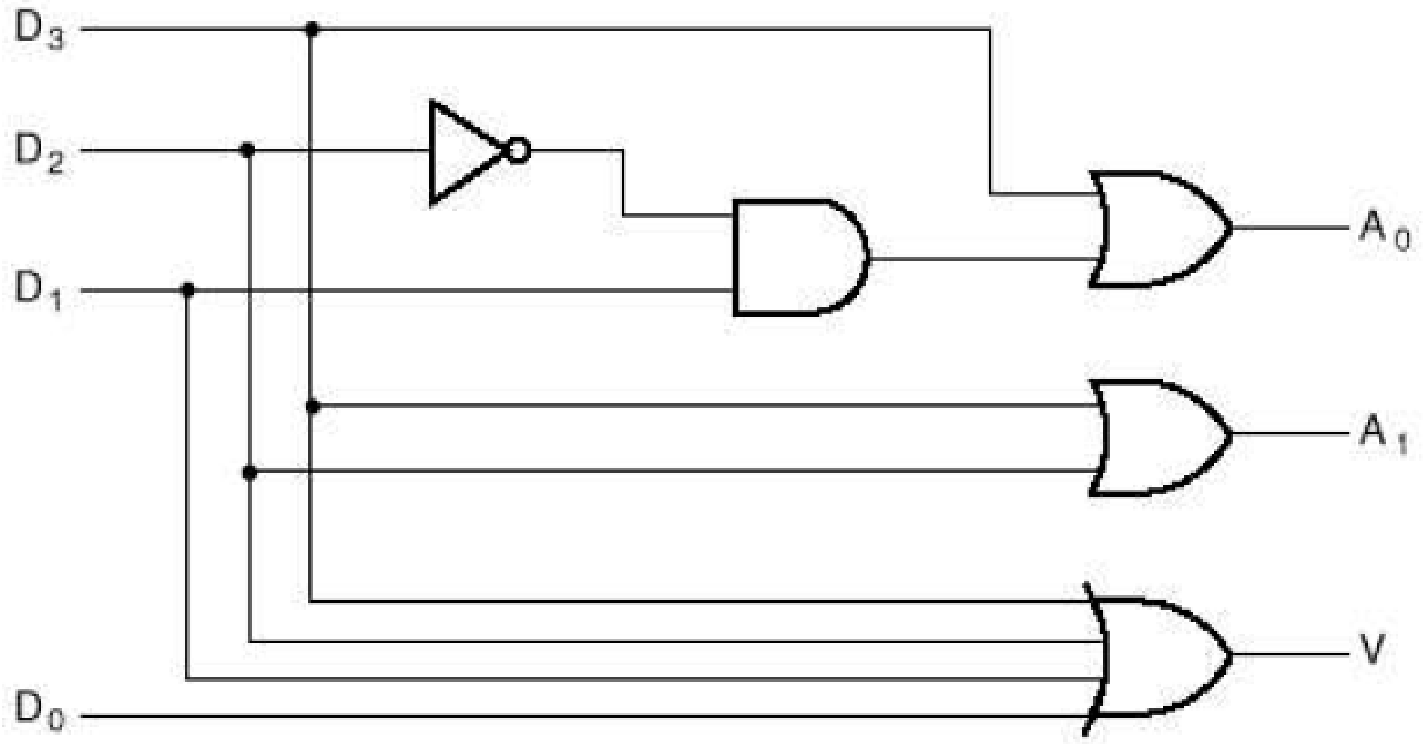


$A_1 = D_2 + D_3$



$A_0 = D_3 + D_1\overline{D}_2$

Example: 4-to-2 Priority Encoder Logic Diagram



8-to-3 Priority Encoder

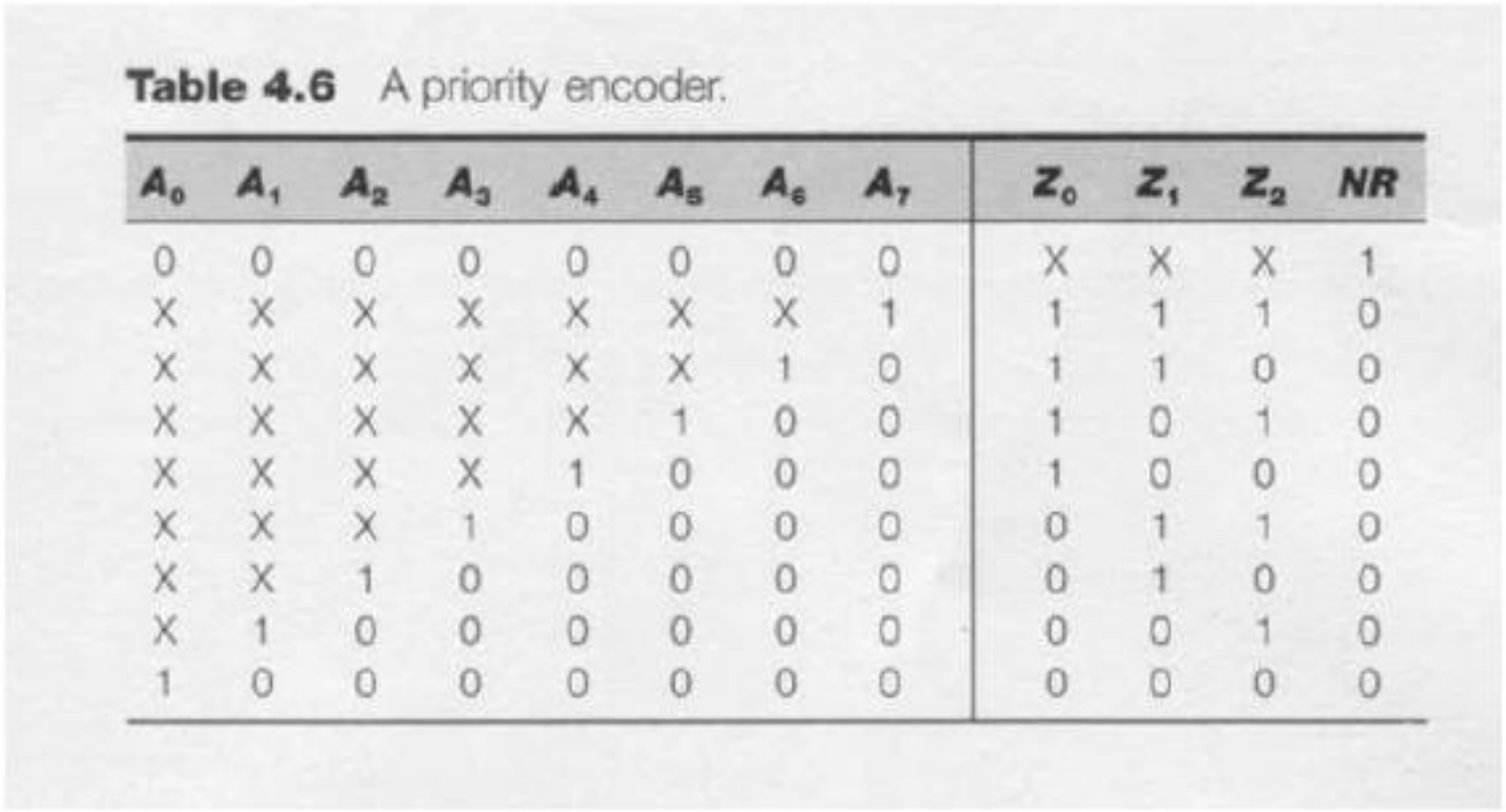
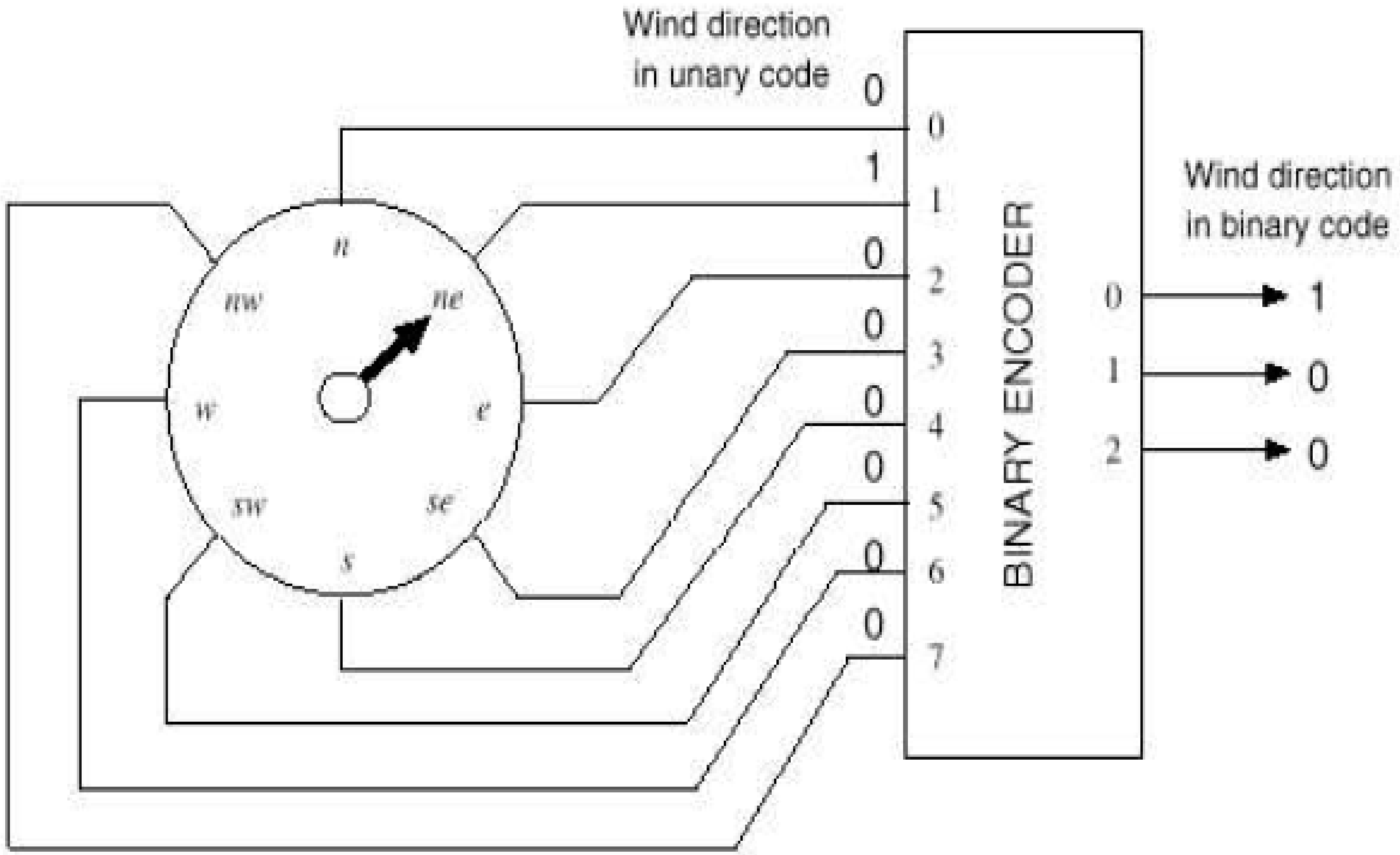


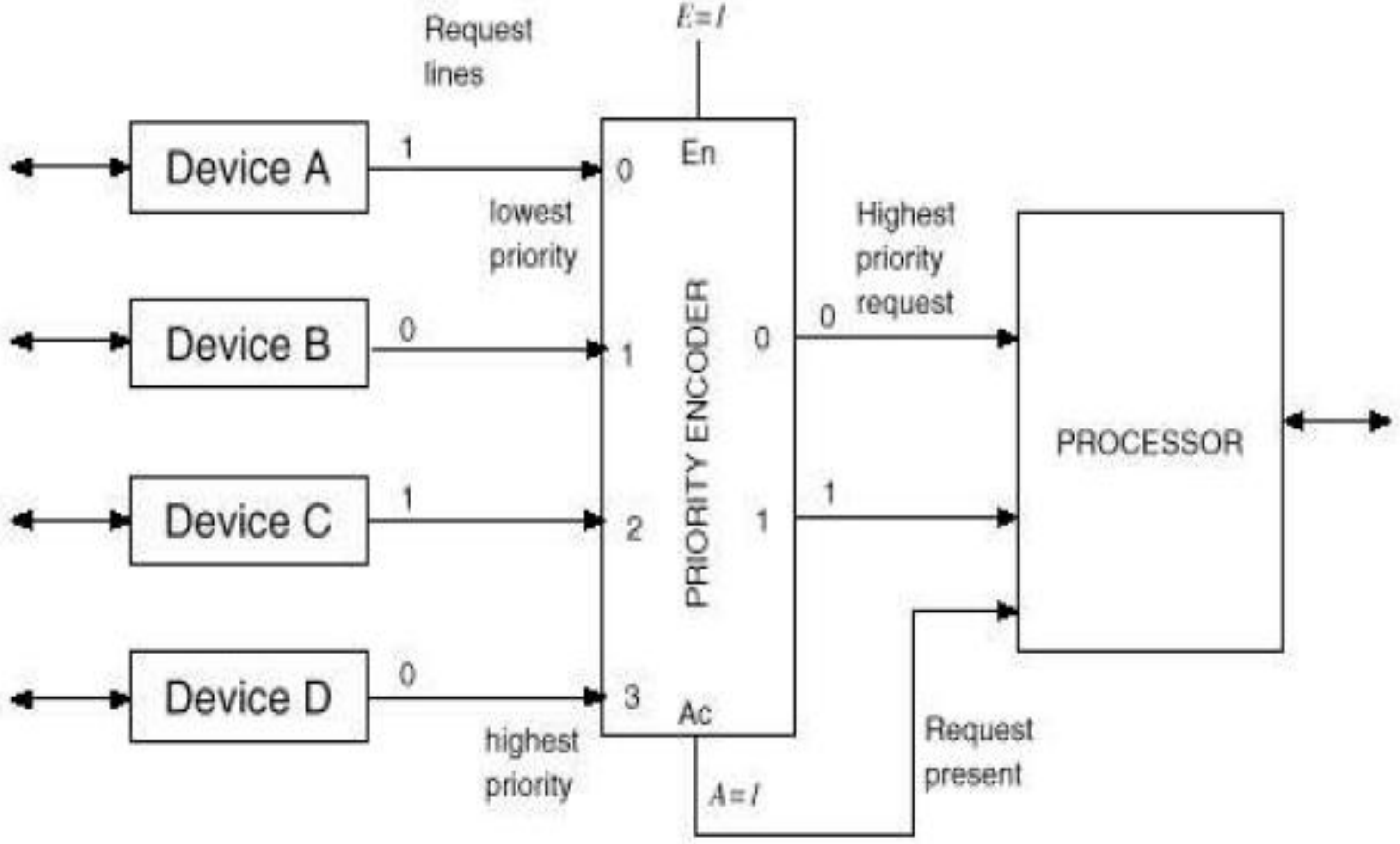
Table 4.6 A priority encoder.

A_0	A_1	A_2	A_3	A_4	A_5	A_6	A_7	Z_0	Z_1	Z_2	NR
0	0	0	0	0	0	0	0	X	X	X	1
X	X	X	X	X	X	X	1	1	1	1	0
X	X	X	X	X	X	1	0	1	1	0	0
X	X	X	X	X	1	0	0	1	0	1	0
X	X	X	X	1	0	0	0	1	0	0	0
X	X	X	1	0	0	0	0	0	1	1	0
X	X	1	0	0	0	0	0	0	1	0	0
X	1	0	0	0	0	0	0	0	0	1	0
1	0	0	0	0	0	0	0	0	0	0	0

Uses of binary Encoders

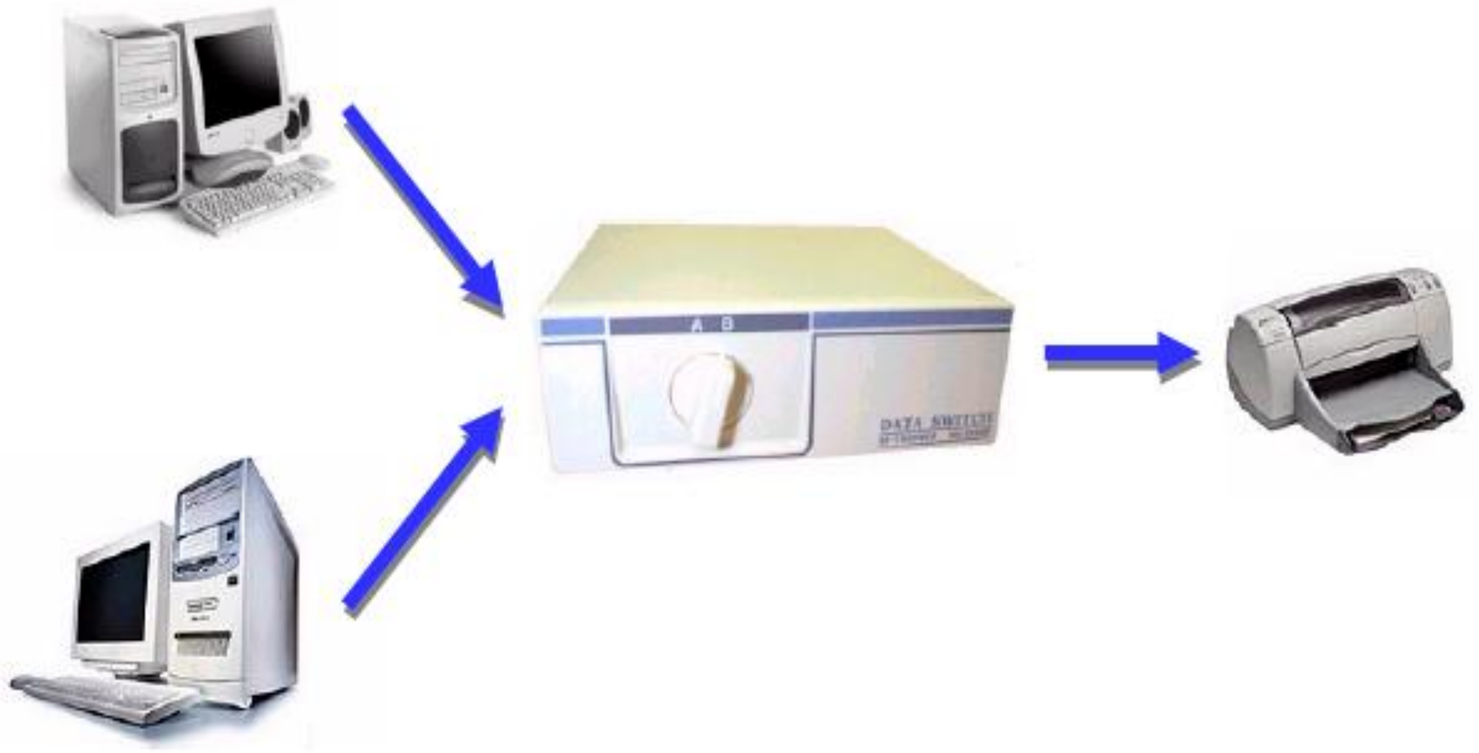


Uses of priority encoders (cont.)



Multiplexers

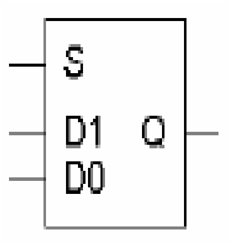
- Let's think about building another circuit, a multiplexer.
- In the old days, several machines could share an I/O device with a switch.



- The switch allows one computer's output to go to the printer's input.

A 2-to-1 multiplexer

- Here is the circuit analog of that printer switch.



- This is a 2-to-1 multiplexer, or mux.
  - There are two data inputs D0 and D1, and a select input called S.
  - There is one output named Q.
- The multiplexer routes one of its data inputs (D0 or D1) to the output Q, based on the value of S.
  - If S=0, the output will be D0.
  - If S=1, the output will be D1.

Building a multiplexer

- Here is a truth table for the multiplexer, based on our description from the previous page:

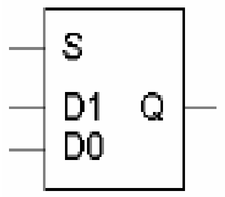
> The multiplexer routes one of its data inputs (D0 or D1) to the output Q, based on the value of S.
>
> – If S=0, the output will be D0.
>
> – If S=1, the output will be D1.

S	D1	D0	Q
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

- You can then find an MSP for the mux output Q.

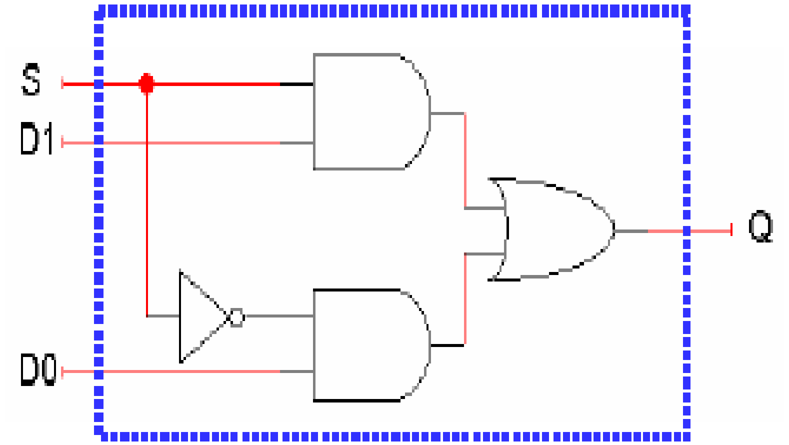
$$Q = S'D0 + S\ D1$$

- Note that this corresponds closely to our English specification above—sometimes you can derive an expression without first making a truth table.



Multiplexer circuit diagram

- Here is an implementation of a 2-to-1 multiplexer.

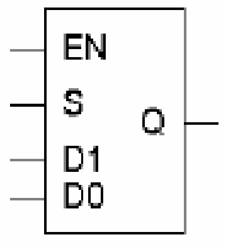


$$Q = S'D0 + S\ D1$$

- Remember that a minimal sum of products expression leads to a minimal two-level circuit.

Enable inputs

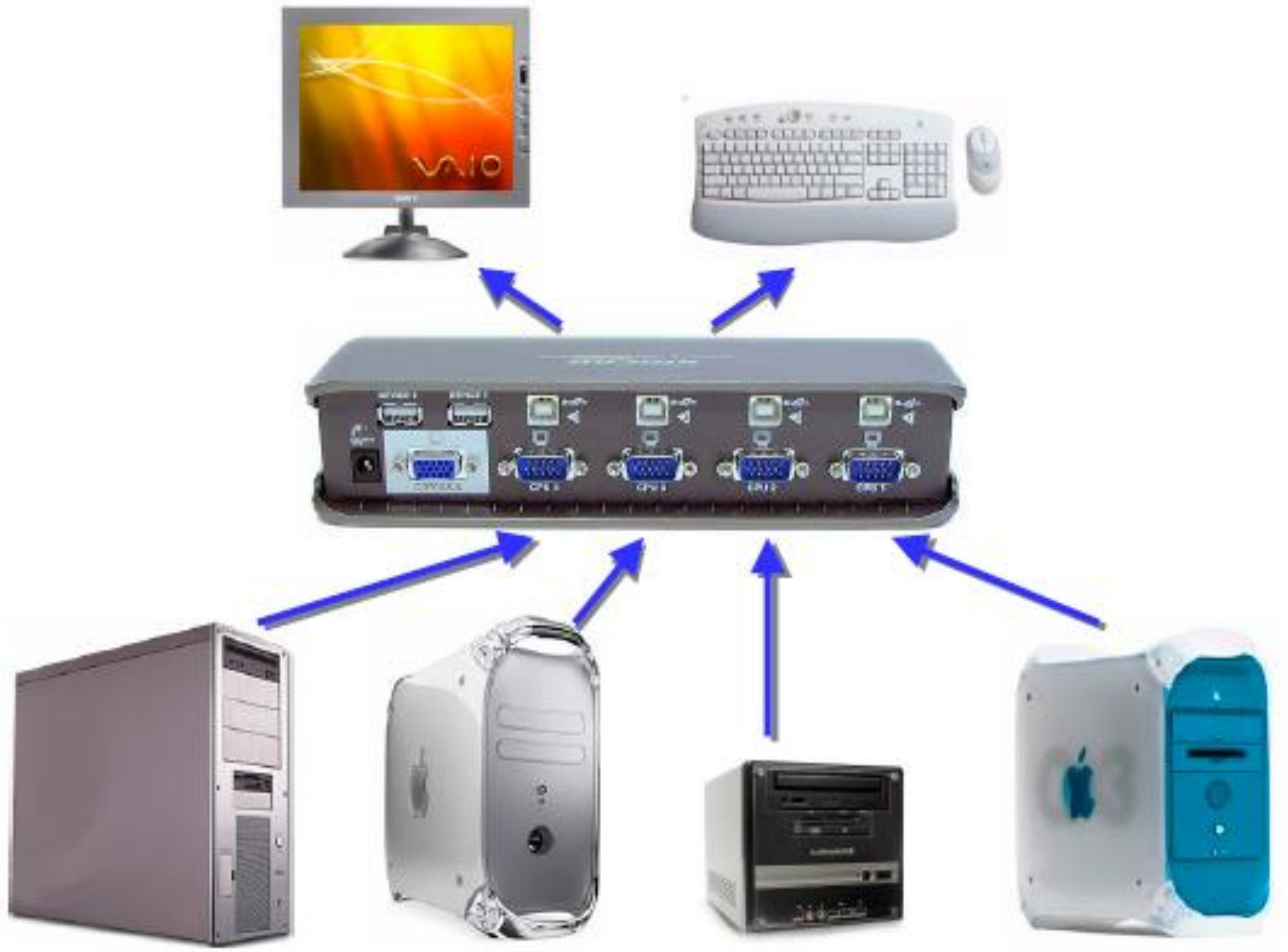
- Many devices have an additional enable input, which "activates" or "deactivates" the device.
- We could design a 2-to-1 multiplexer with an enable input that's used as follows.
  - EN=0 disables the multiplexer, which forces the output to be 0. (It does *not* turn off the multiplexer.)
  - EN=1 enables the multiplexer, and it works as specified earlier.
- Enable inputs are especially useful in combining smaller muxes together to make larger ones, as we'll see later today.



EN	S	D1	D0	Q
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	1
1	1	1	1	1

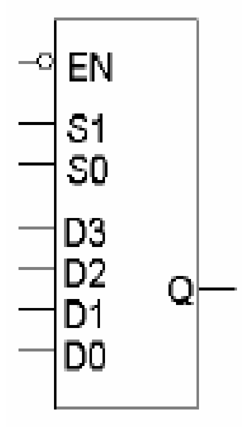
A KVM switch

- This KVM switch allows four computers to share a single keyboard, video monitor, and mouse.



A 4-to-1 multiplexer

- Here is a block diagram and abbreviated truth table for a 4-to-1 mux, which directs one of four different inputs to the single output line.
  - There are four data inputs, so we need *two* bits, S1 and S0, for the mux selection input.

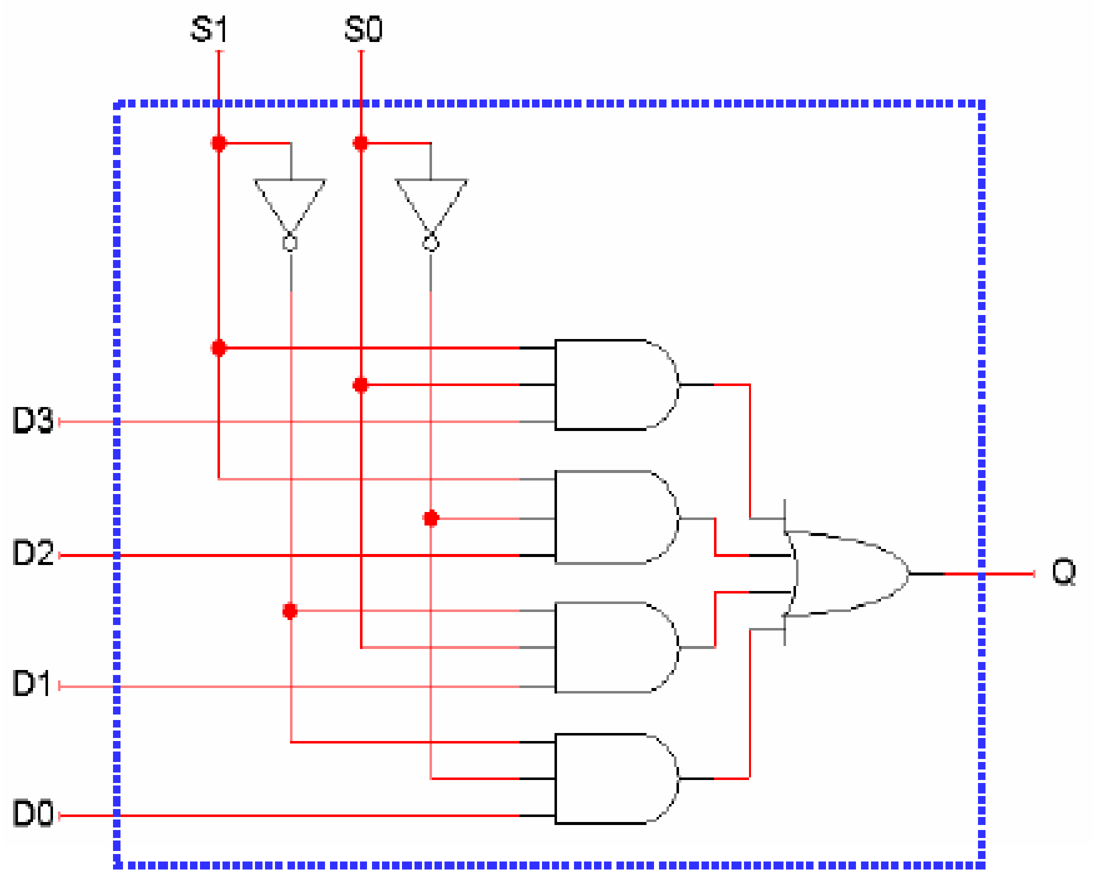


EN’	S1	S0	Q
0	0	0	D0
0	0	1	D1
0	1	0	D2
0	1	1	D3
1	x	x	1

Q = S1’S0’D0 + S1’S0 D1 + S1 S0’D2 + S1 S0 D3

A 4-to-1 multiplexer implementation

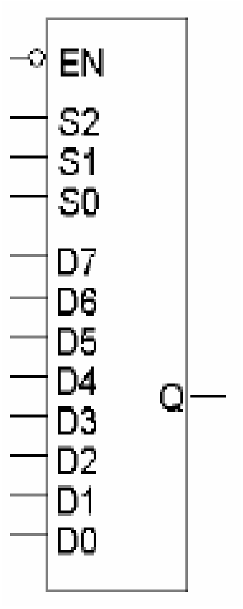
- Again we have a minimal sum of products expression, which leads to a minimal two-level circuit implementation.



Q = S1'S0'D0 + S1'S0 D1 + S1 S0'D2 + S1 S0 D3

2^n-to-1 multiplexers

- You can make even larger multiplexers, following the same pattern.
- A 2^n-to-1 multiplexer routes one of 2^n input lines to the output line.
  - There are 2^n data inputs, so there must also be *n* select inputs.
  - The output is a single bit.
- Here is an 8-to-1 multiplexer, probably the biggest we'll see in this class.



Example: addition

- Multiplexers can sometimes make circuit design easier.
- As an example, let’s make a circuit to add three 1-bit inputs X, Y and Z.
- We’ll need two bits to represent the total.
  - The bits will be called C and S, standing for “carry” and “sum.”
  - These are two separate functions of the inputs X, Y and Z.
- A truth table and sum of minterm equations for C and S are shown below.

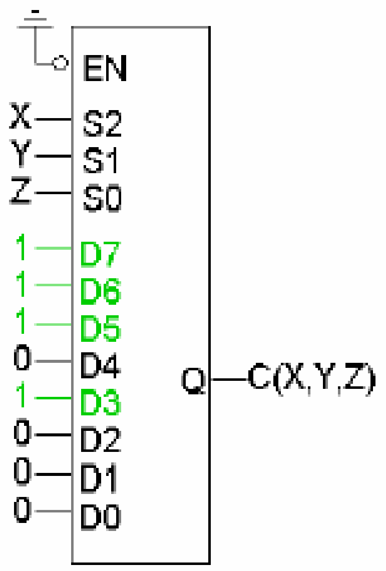
	X	Y	Z	C	S
	0	0	0	0	0
	0	0	1	0	1
	0	1	0	0	1
0 + 1 + 1 = 10 →	0	1	1	1	0
	1	0	0	0	1
	1	0	1	1	0
	1	1	0	1	0
1 + 1 + 1 = 11 →	1	1	1	1	1

$$C(X,Y,Z) = \Sigma m(3,5,6,7)$$
$$S(X,Y,Z) = \Sigma m(1,2,4,7)$$

Implementing functions with multiplexers

- We could implement a function of *n* variables with an *n*-to-1 multiplexer.
  - The mux select inputs correspond to the function's input variables, and are used to select one row of the truth table.
  - Each mux data input corresponds to one output from the truth table. We connect 1 to data input Di for each function minterm m_i, and we connect 0 to the other data inputs.
- For example, here is the carry function, $C(X,Y,Z) = \Sigma m(3,5,6,7)$.

X	Y	Z	C
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



Partitioning the truth table

- We can actually implement $C(X,Y,Z) = \Sigma m(3,5,6,7)$ with just a 4-to-1 mux, instead of an 8-to-1.
  - Instead of using three variables to select one row of the truth table, we'll use two variables to pick a *pair* of rows in the table.
  - The multiplexer data inputs will be functions of the remaining variable, which distinguish between the rows in each pair.
- First, we can divide the rows of our truth table into pairs, as shown on the right. X and Y are constant within each pair of rows, so C is a function of Z only.
  - When XY=00, C=0
  - When XY=01, C=Z
  - When XY=10, C=Z
  - When XY=11, C=1

X	Y	Z	C
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 more efficient adder

- All that's left is setting the multiplexer inputs.
  - The two input variables X and Y will be connected to select inputs S1 and S0 of our 4-to-1 multiplexer.
  - The expressions for C(Z) are then connected to the data inputs D0-D3 of the multiplexer.

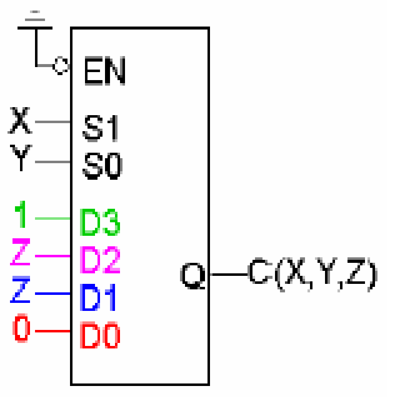
X	Y	Z	C
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

When XY=00, C=0

When XY=01, C=Z

When XY=10, C=Z

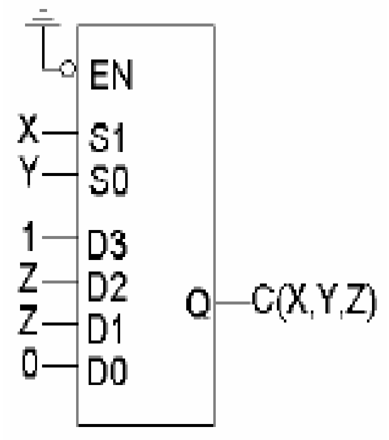
When XY=11, C=1



Verifying our adder

- Don't believe that this works? Start with the equation for a 4-to-1 multiplexer

$$Q = S1'S0'D0 + S1'S0\,D1 + S1\,S0'D2 + S1\,S0\,D3$$



- Then just plug in the actual inputs to our circuit, as shown again on the right: S1S0 = XY, D3 = 1, D2 = Z, D1 = Z, and D0 = 0.

$$\begin{aligned} C &= X'Y'\bullet 0 + X'YZ + XY'Z + XY\bullet 1 \\ &= X'YZ + XY'Z + XY \\ &= X'YZ + XY'Z + XY(Z' + Z) \\ &= X'YZ + XY'Z + XYZ' + XYZ \end{aligned}$$

- So the multiplexer output really is the carry function, $C(X,Y,Z) = \Sigma m(3,5,6,7)$.

Multiplexer-based sum

- Here's the same thing for the sum function, $S(X,Y,Z) = \Sigma m(1,2,4,7)$.

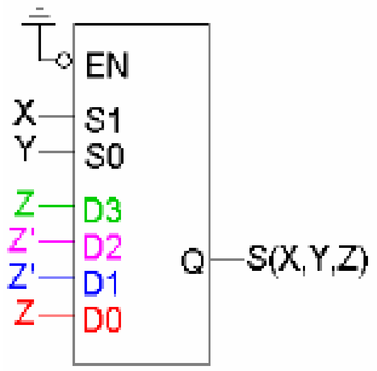
X	Y	Z	S
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

When XY=00, S=Z

When XY=01, S=Z'

When XY=10, S=Z'

When XY=11, S=Z

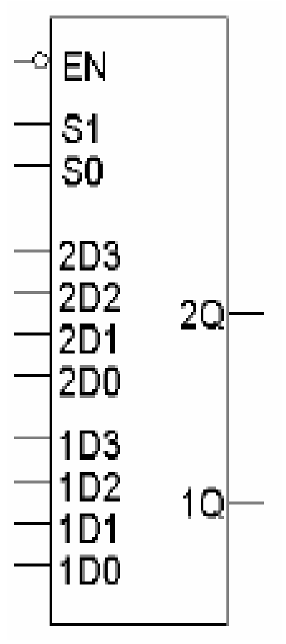


- Again, we can show that this is a correct implementation.

$$\begin{aligned} Q &= S1'S0'D0 + S1'S0\,D1 + S1\,S0'D2 + S1\,S0\,D3 \\ &= X'Y'Z + X'YZ' + XY'Z' + XYZ \\ &= \Sigma m(1,2,4,7) \end{aligned}$$

Dual multiplexers

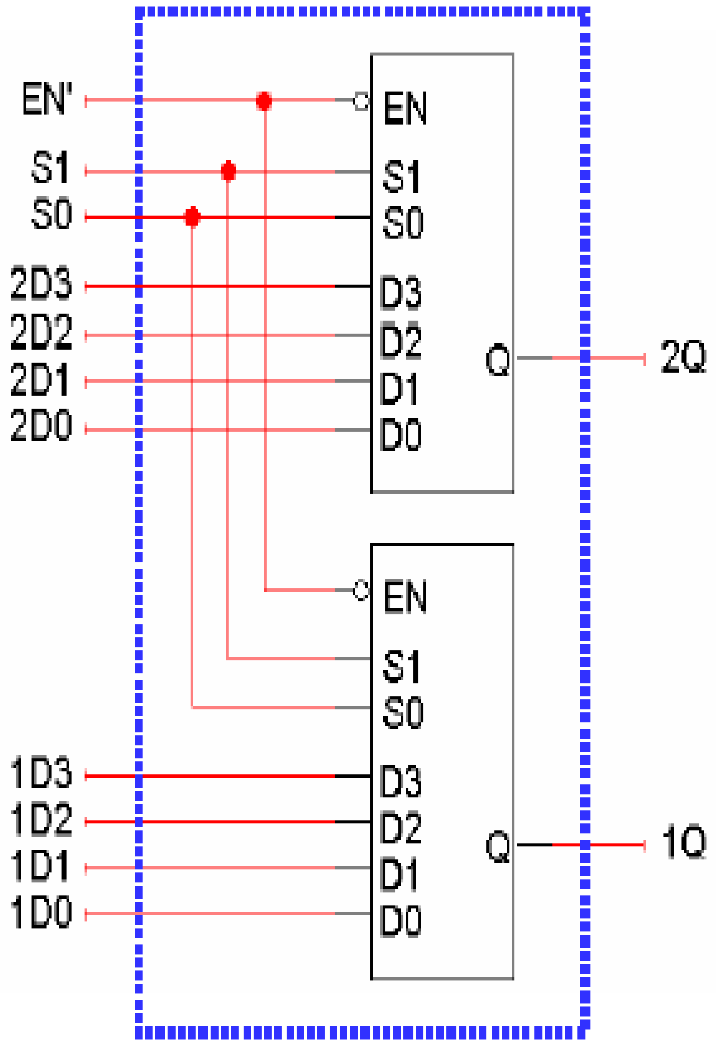
- A dual 4-to-1 mux allows you to select from one of four 2-bit data inputs.
  - The two output bits are 2Q 1Q, and S1-S0 select a *pair* of inputs.



EN'	S1	S0	2Q	1Q
0	0	0	2D0	1D0
0	0	1	2D1	1D1
0	1	0	2D2	1D2
0	1	1	2D3	1D3
1	x	x	1	1

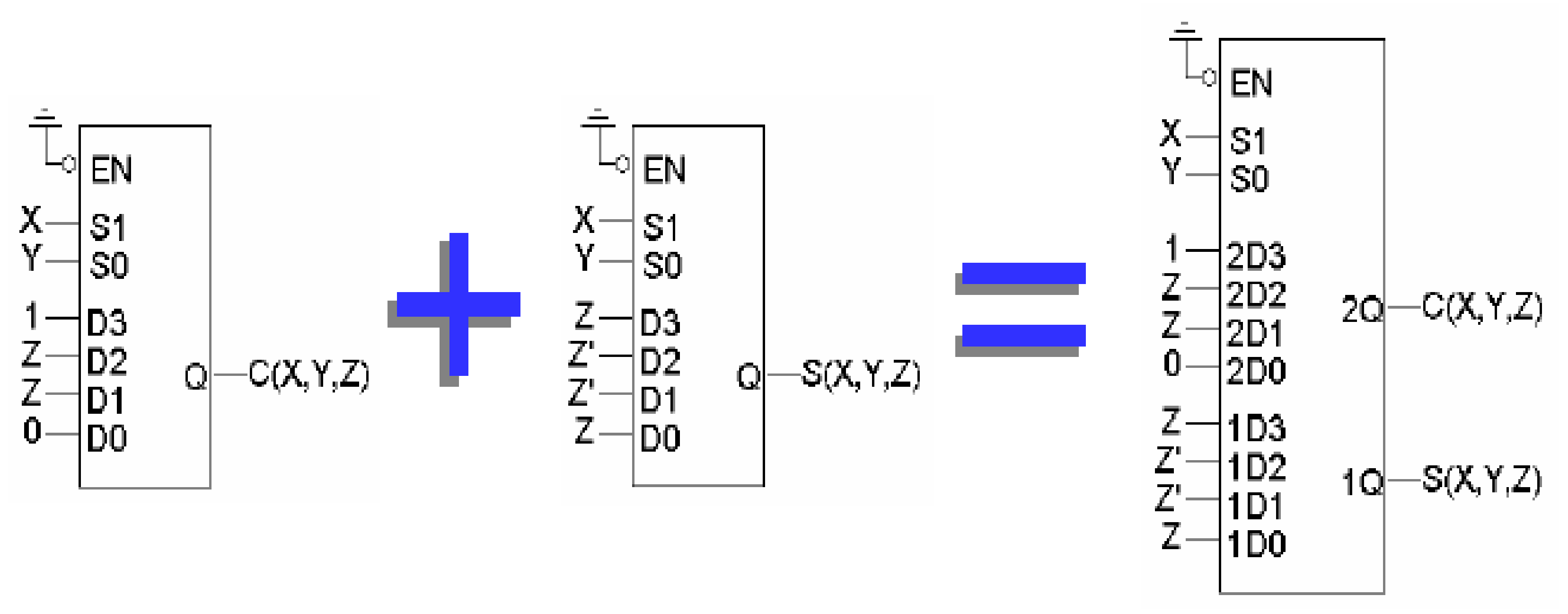
Dual muxes in more detail

- You could build a dual 4-to-1 mux from its truth table and our familiar circuit design techniques.
- It's also possible to combine smaller muxes together to form larger ones.
- You can build the dual 4-to-1 mux just by using two 4-to-1 muxes.
  - The two 4-to-1 multiplexers share the same EN', S1 and S0 signals.
  - Each smaller mux produces one bit of the two-bit output 2Q 1Q.
- This kind of hierarchical design is very common in computer architecture.



Dual multiplexer-based adder

- We can use this dual 4-to-1 multiplexer to implement our adder, which produces a two-bit output consisting of C and S.



- That KVM switch from earlier is really a "tri 4-to-1 multiplexer," since it selects from four sets of three signals (keyboard, video and mouse).

De Multiplexers

A De-multiplexer is a logic circuit that transmit Information on a single line on one of 2^n output lines

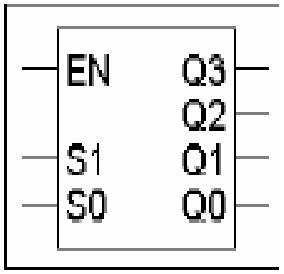
Selection of output line depends on the value of n select lines

A Decoder with enable input can function as a De-multiplexer

Enable inputs

decoders can include enable inputs.

- EN=0 disables the decoder, which by convention means that all of the decoder's outputs are 0.
- EN=1 enables the decoder so that it behaves as specified earlier, with exactly one of the outputs being 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