Introduction to Digital Logic

Lecture 11:

Cascading Decoders

Implementing Functions w/ Decoders

Encoders & Priority Encoders

Building Larger Functions/Circuits

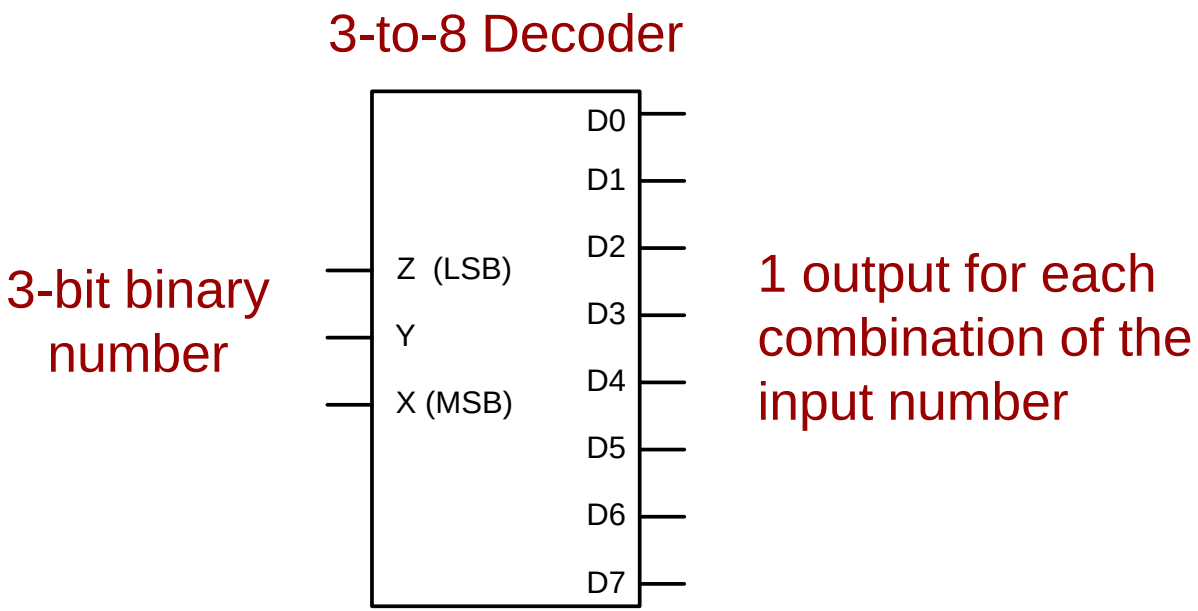
- Scalability issues
  - K-Maps: Up to 6 inputs
  - Decoders: 6-8 inputs
    - But larger decoders can be built from smaller ones
  - Multiplexers: Can be decomposed to multiple levels
    - Use Shannon's Theorem
  - Memories: 12-13 inputs (4K – 8K rows)
- Building Block Methodology
  - Decompose circuits into smaller units
  - Design the smaller units using any of the above methods
  - Use those smaller circuits as building blocks to construct arbitrarily large functions/circuits

Combinational Building Blocks

- Fundamental blocks that other combinational structures can be built from
  - Decoders
  - Encoders
  - Multiplexers
  - Demultiplexers
  - Adders (Multipliers)
  - Comparators
  - Shifters

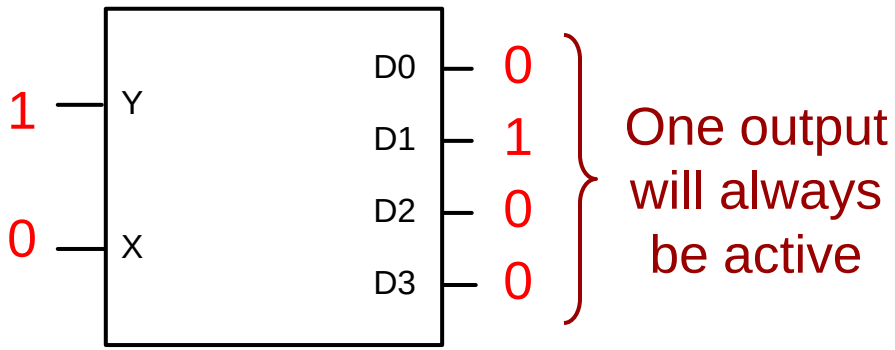
Decoders

- A decoder is a building block that:
  - Takes in an n-bit binary number as input
  - Decodes that binary number and activates the corresponding output
  - Individual outputs for EVERY input combination (i.e. 2^n outputs)

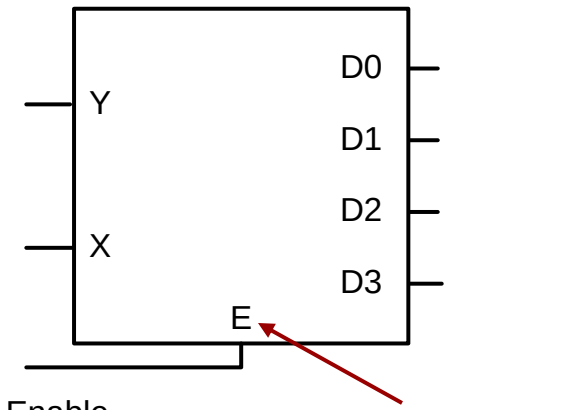


Enables

- Exactly one output is active at all times
- It may be undesirable to always have an active output
- Add an extra input (called an enable) that can independently force all the outputs to their inactive values

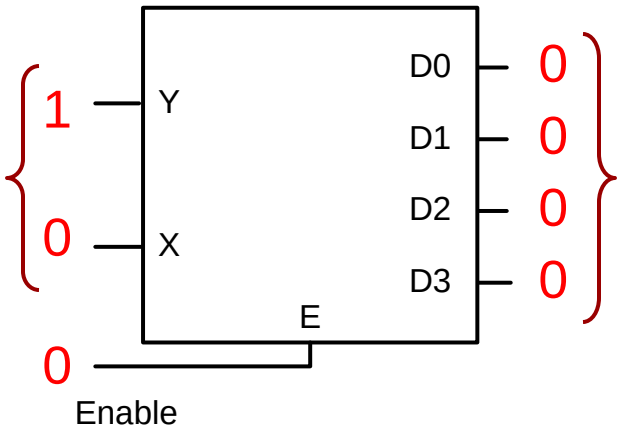


2-to-4 Decoder



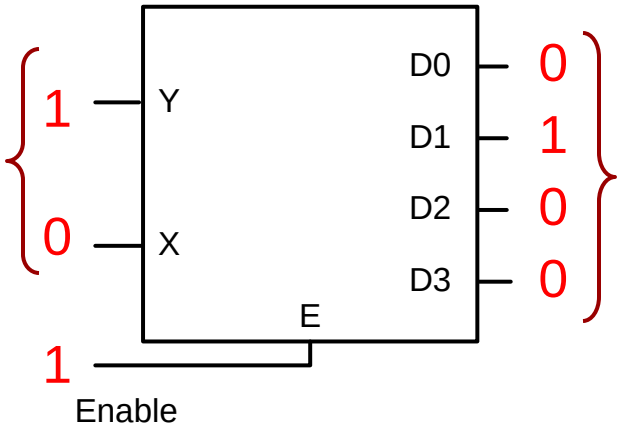
Enables

When E=0, inputs is ignored



Since E=0, all outputs = 0

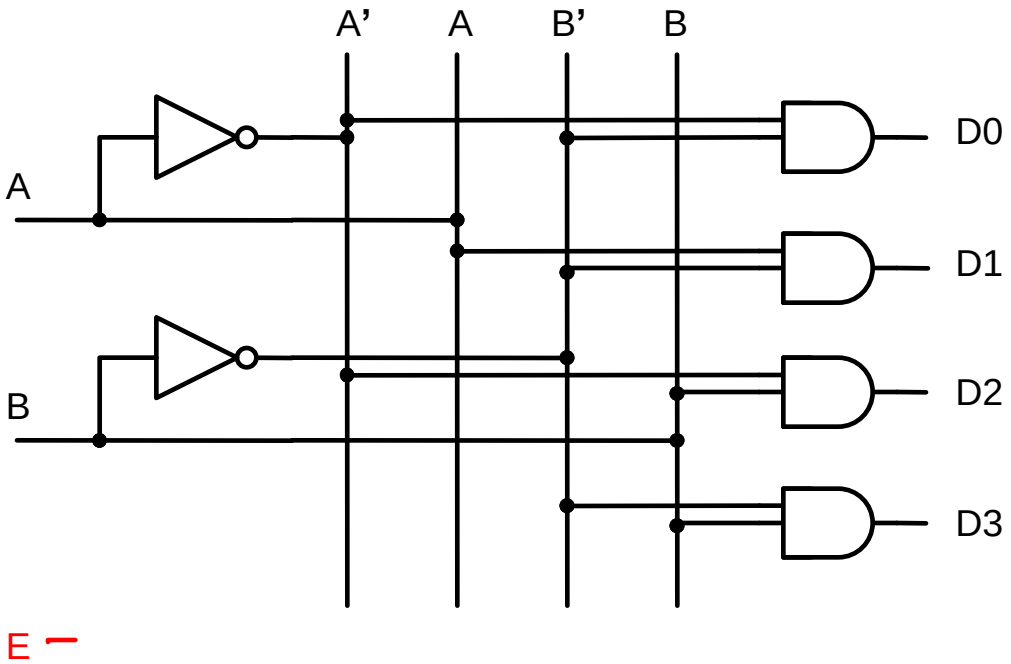
When E=1, inputs will cause the appropriate output to go active



Since E=1, outputs will function normally

Implementing Enables

- Original 2-to-4 decoder

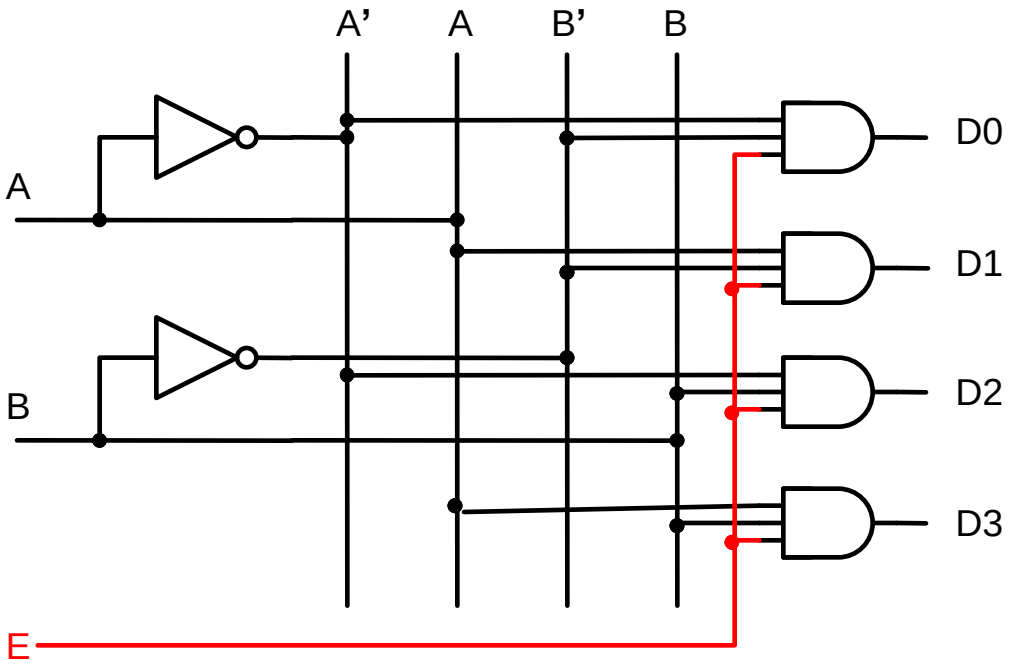


When E=0, force all outputs = 0

When E=1, outputs operate as they did originally

Enables

- Enables can be implemented by connecting it to each AND gate of the decoder



When E=0, 0 AND anything = 0

When E=1, 1 AND anything = that anything, which was the normal decoding logic

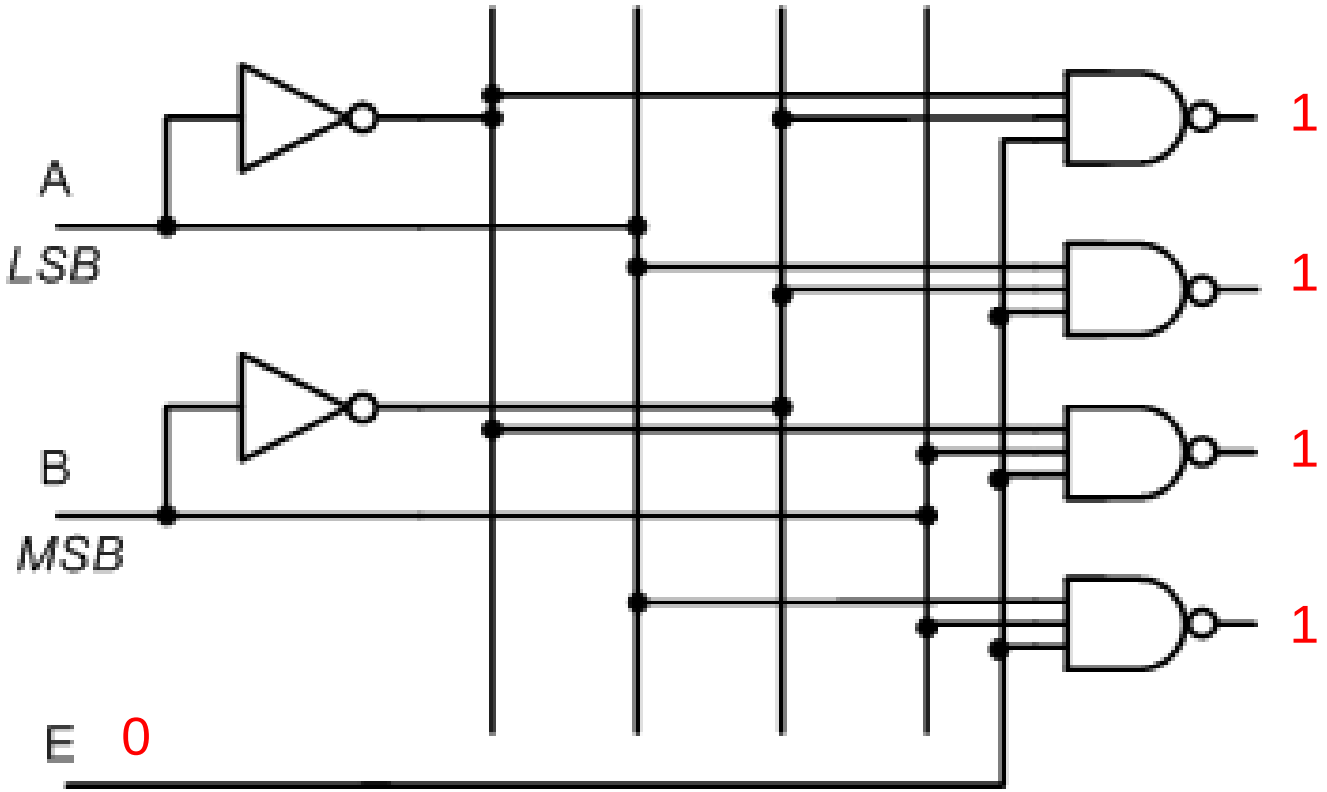
Active-hi vs. Active-low

- Active-hi convention
  - 1 = on/true/active
  - 0 = off/false/inactive
- Active-low convention
  - 0 = on/true/active
  - 1 = off/false/inactive
- To convert between conventions
  - INVERT!!!

Why Active-low

- Some digital circuits are better at “sinking” (draining/sucking) electric current than “sourcing” (producing) current

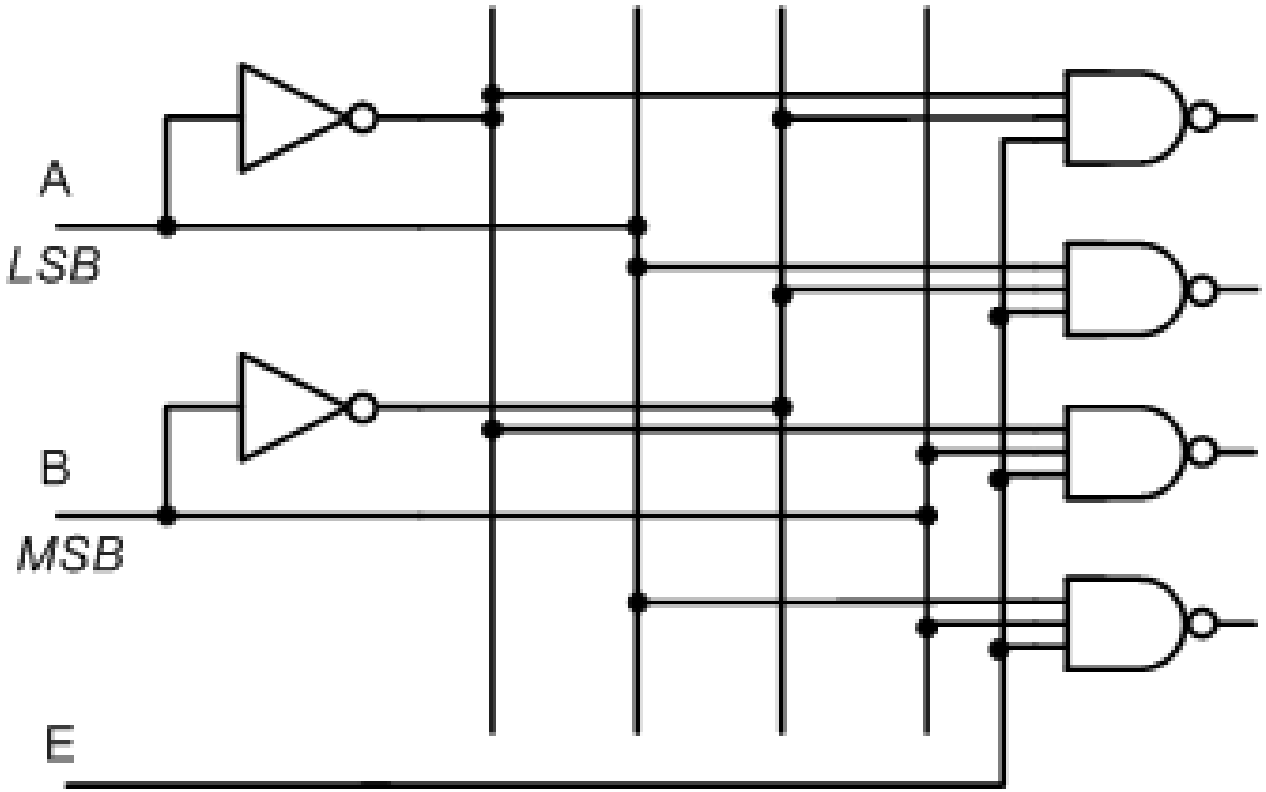
Active-Lo Outputs



When E=inactive (inactive means 0), Outputs turn off (off means 1)

When E=active (active means 1), Selected outputs turn on (on means 0)

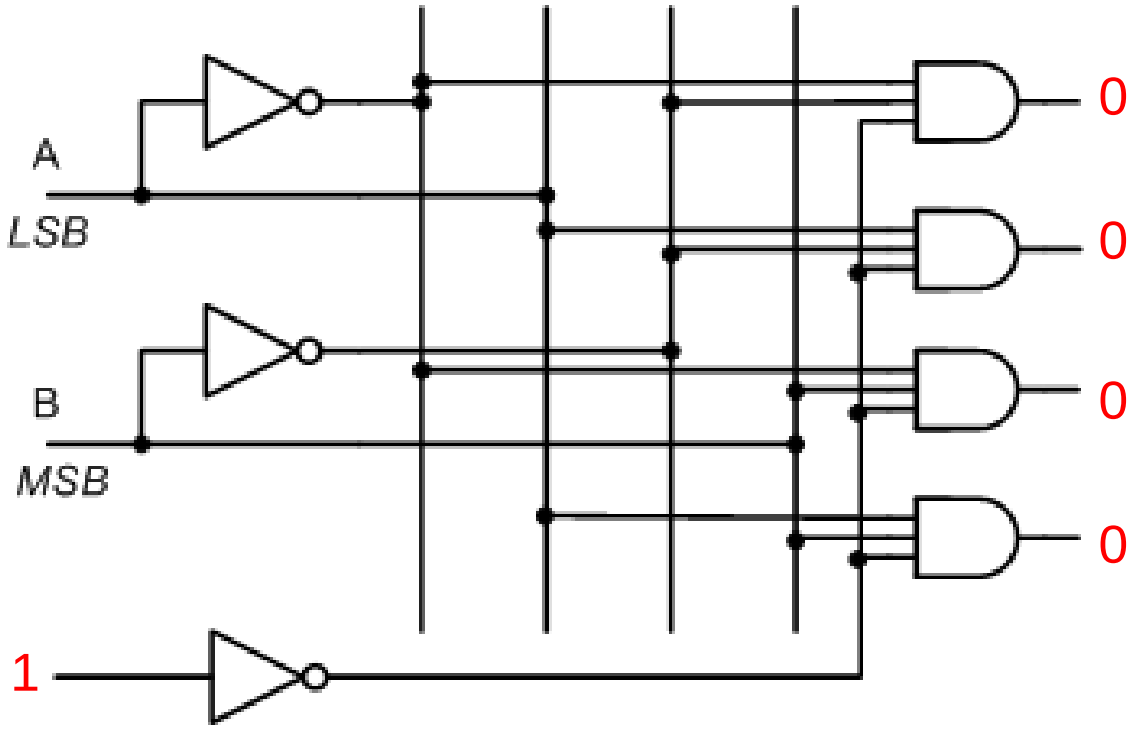
Active-Lo Outputs



When E=inactive (inactive means 0), Outputs turn off (off means 1)

When E=active (active means 1), Selected outputs turn on (on means 0)

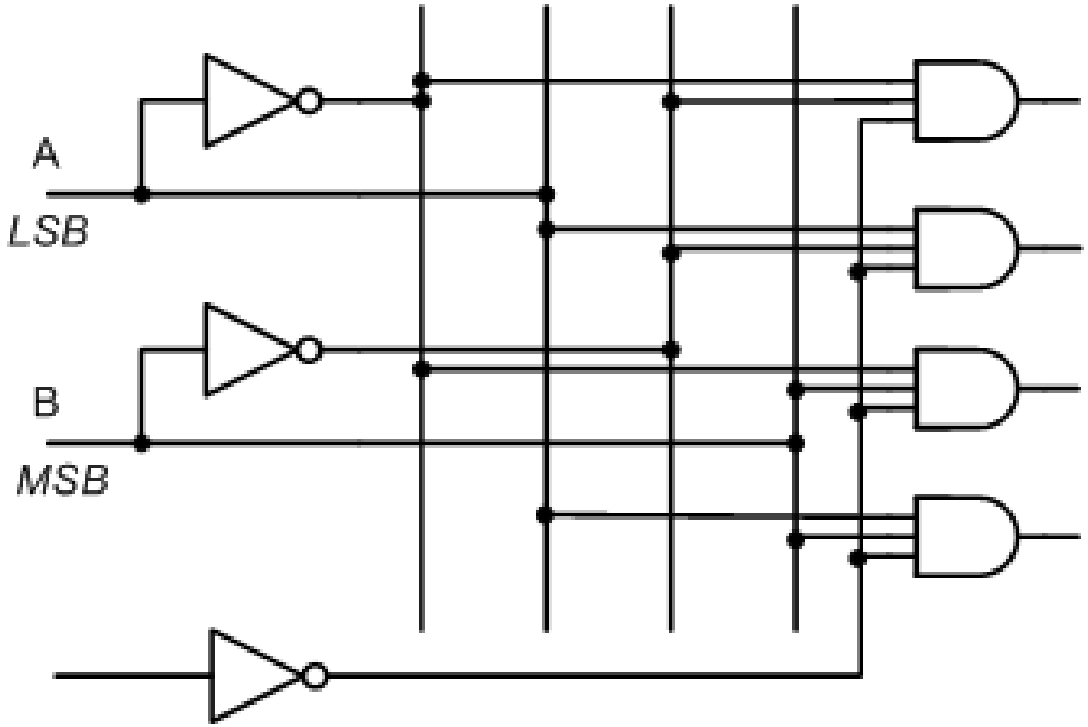
Active-Lo Enable



When E=inactive (inactive means 1), Outputs turn off (off means 0)

When E=active (active means 0), Selected outputs turn on (on means 1)

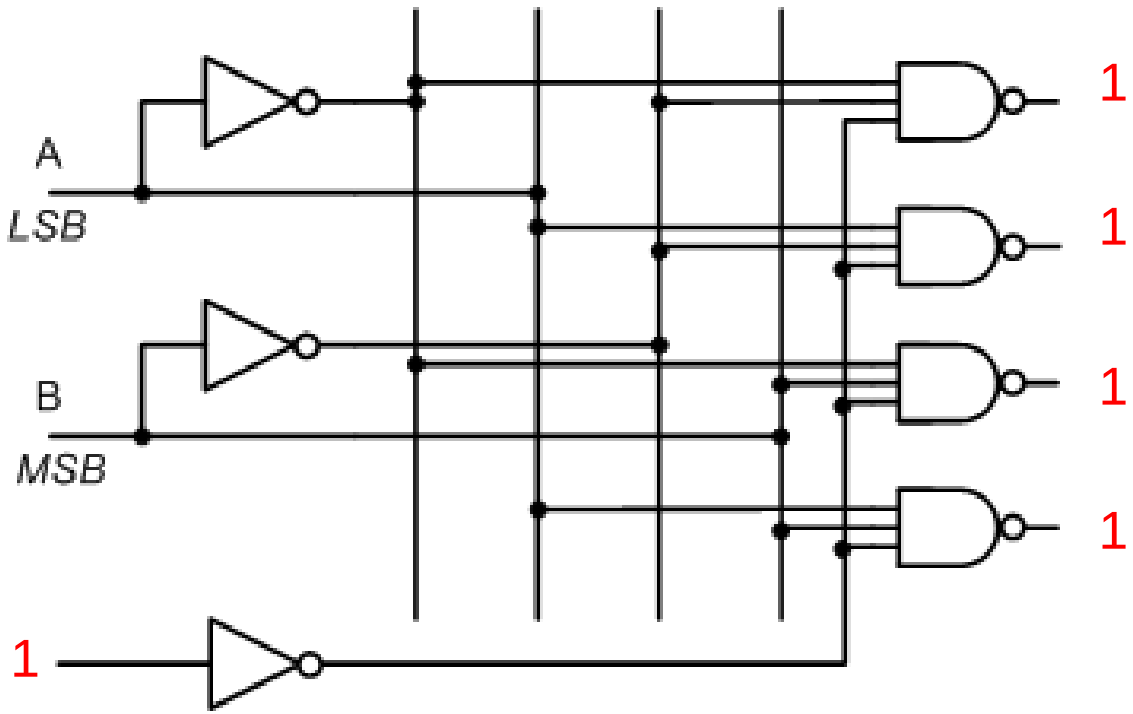
Active-Lo Enable



When E=inactive (inactive means 1), Outputs turn off (off means 0)

When E=active (active means 0), Selected outputs turn on (on means 1)

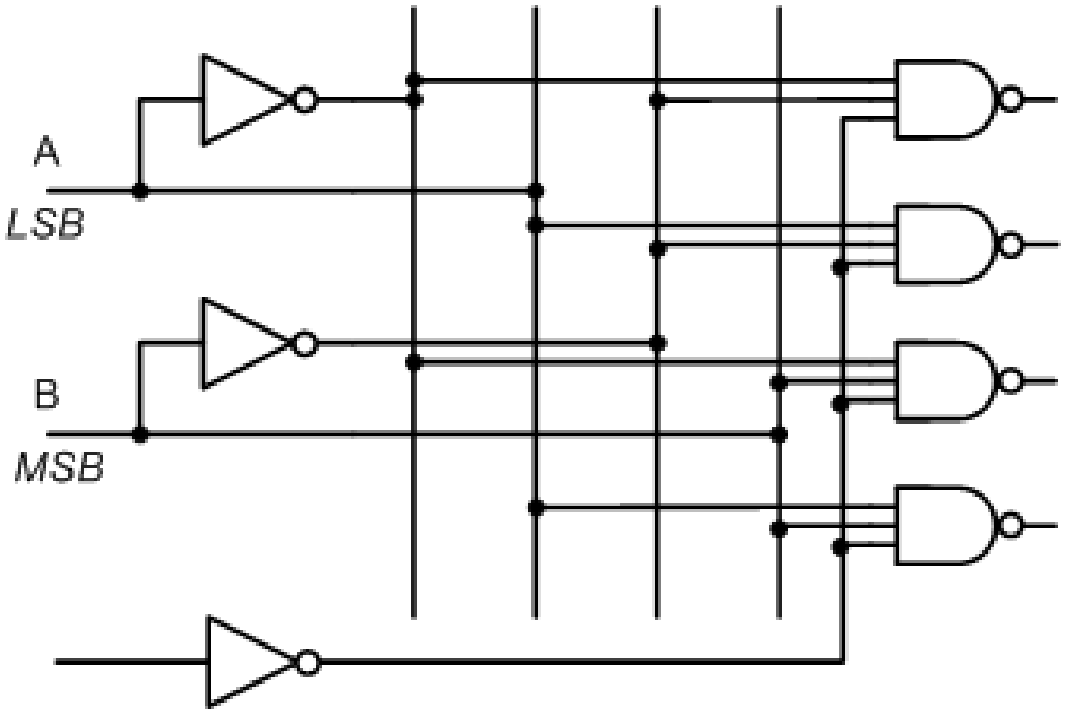
Active-Lo Enable



When E=inactive (inactive means 1), Outputs turn off (off means 1)

When E=active (active means 0), Selected outputs turn on (on means 0)

Active-Lo Enable



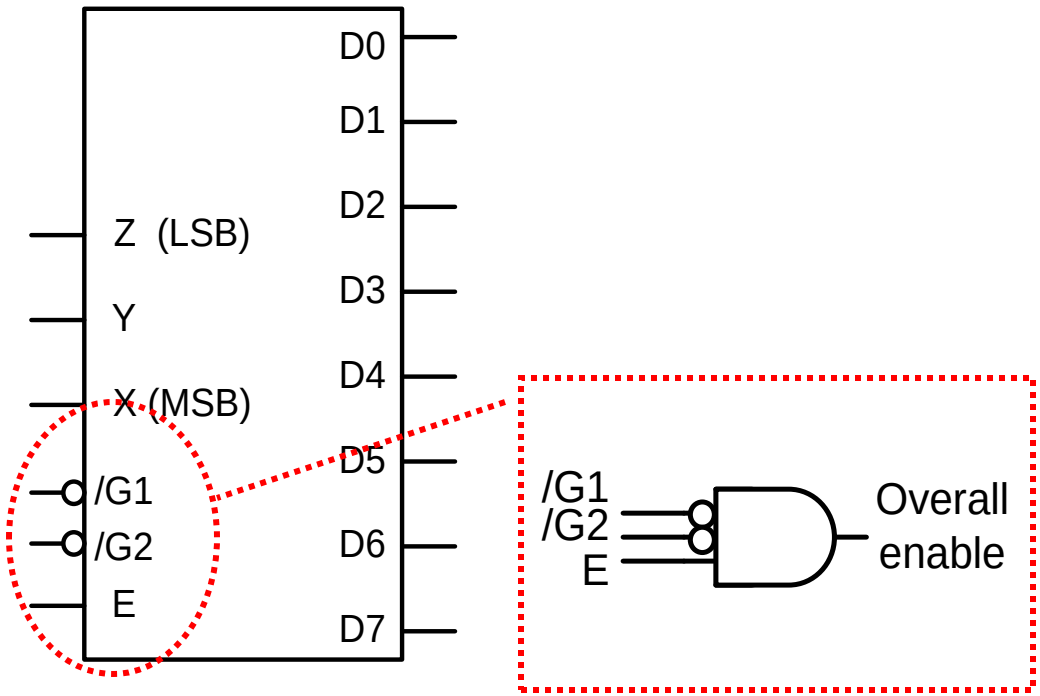
When E=inactive (inactive means 1), Outputs turn off (off means 1)

When E=active (active means 0), Selected outputs turn on (on means 0)

Decoder w/ Multiple Enables

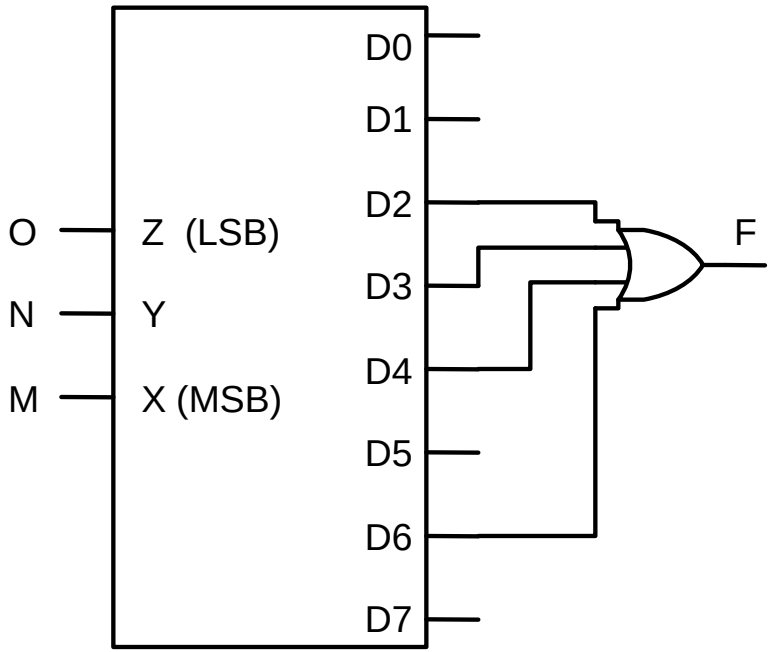
- When a decoder has multiple enables, all enables ***must be active*** for the decoder to be enabled

3 Enables
/G1 must equal 0
/G2 must equal 0
and E must equal 1



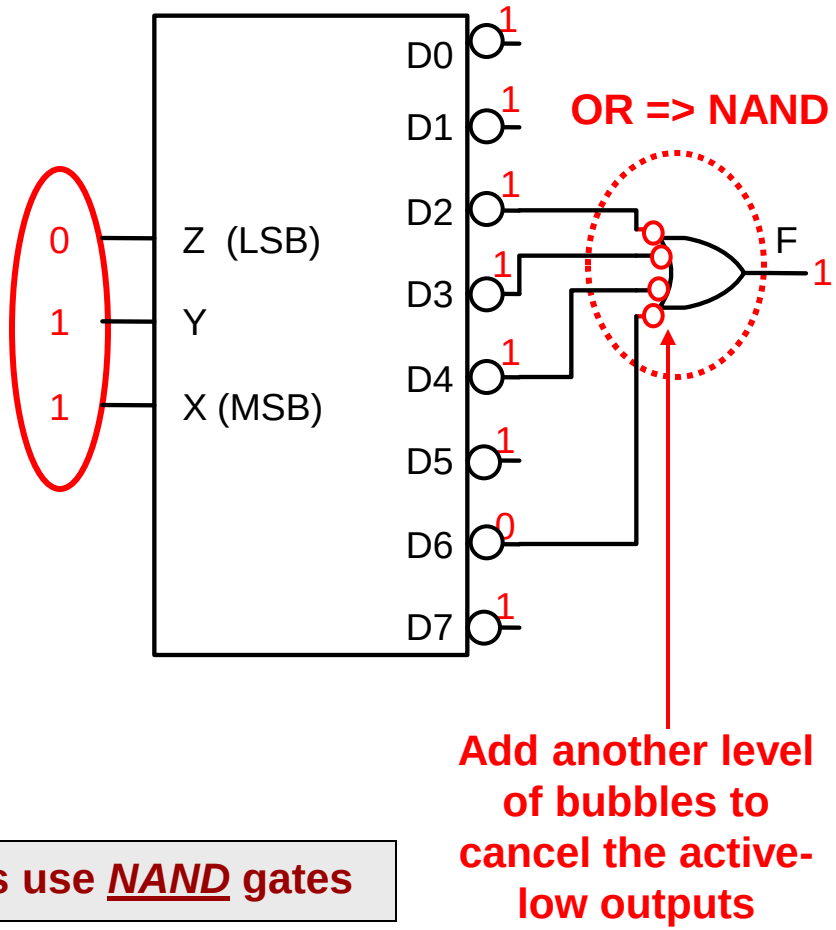
Implementing Logic Functions

- $F = \Sigma_{MNO}(2,3,4,6)$
  $= m_2 + m_3 + m_4 + m_6$
- Since decoders are just minterm generators, just OR together the appropriate minterms



Implementing Logic Functions

- $F = \Sigma_{MNO}(2,3,4,6)$
  $= m_2 + m_3 + m_4 + m_6$
- If we have active-low outputs just invert back to active-hi
- OR gate becomes NAND gate



Key: w/ active-lo outputs use *NAND* gates

Decoder/Logic Function Summary

- To produce F…

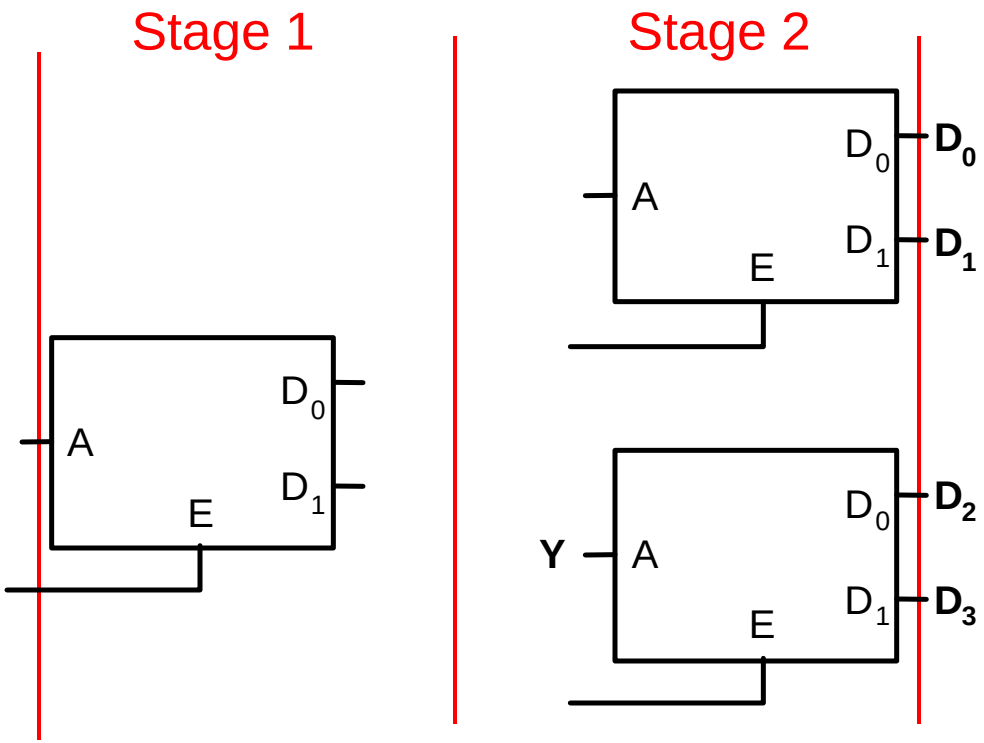
	Active-hi outputs	Active-lo outputs
Fewer minterms in F	Implement F with OR gate	Implement F with NAND gate
Fewer minterms in F'	Implement F' with NOR gate	Implement F' with AND gate

Building Larger Decoders

- Using the “building-block methodology”, cascade smaller decoders to build larger ones
- We’ll use stages of decoders
  - Start at the last stage (outputs) using as many small decoders as necessary to make the desired number (i.e. 2^n) of outputs
  - Connect enables of one stage to outputs of previous stage
  - All decoders in a stage should decode the same bit(s) of the input [usually MSB to first stage, LSB to last]

Cascading Decoders

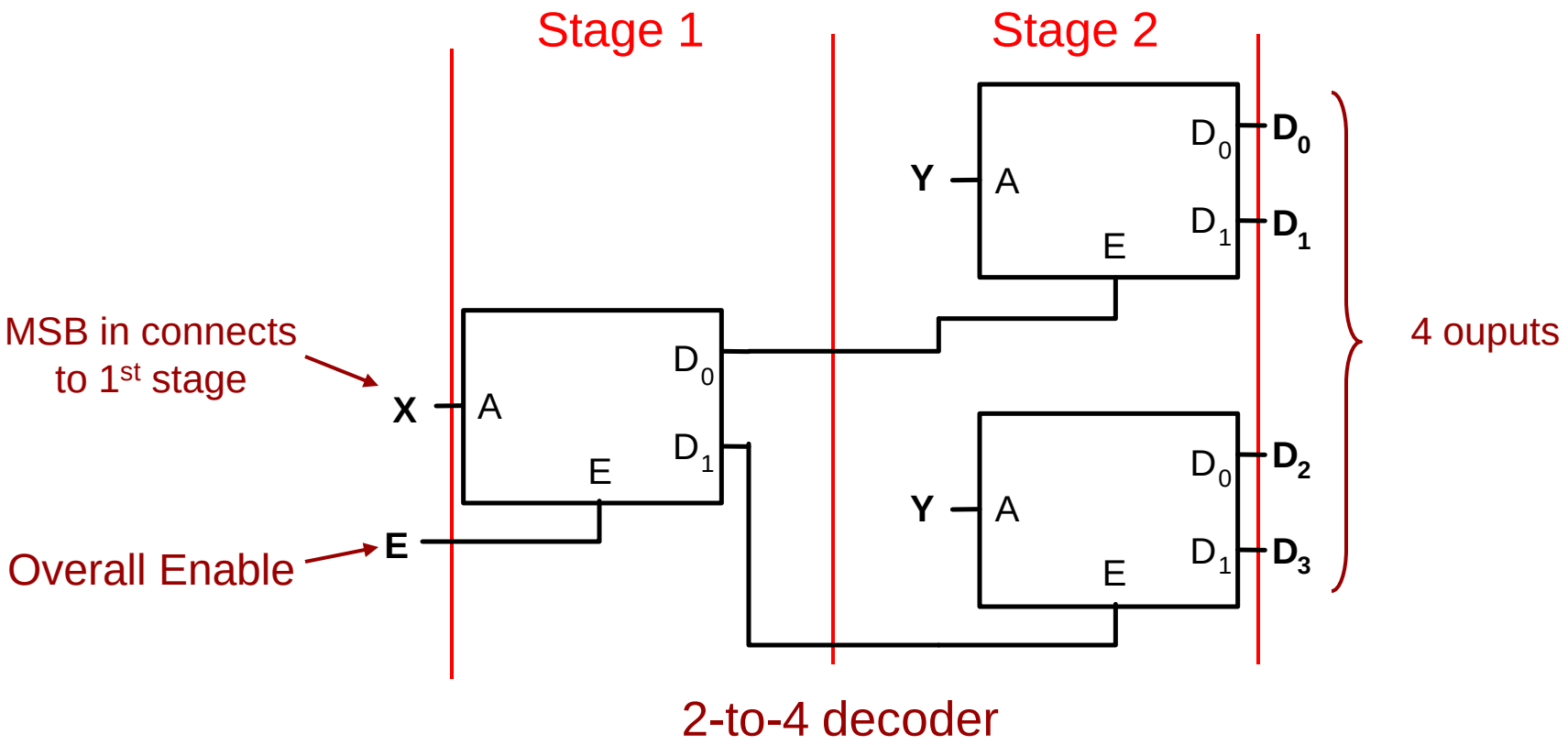
- Connect outputs of first stage to enables of next stage
- Usually, MSB's are connected to the first stage, LSB's to the following stages



2-to-4 decoder

Cascading Decoders

- Connect outputs of first stage to enables of next stage
- Usually, MSB's are connected to the first stage, LSB's to the following stages



Cascading Decoders

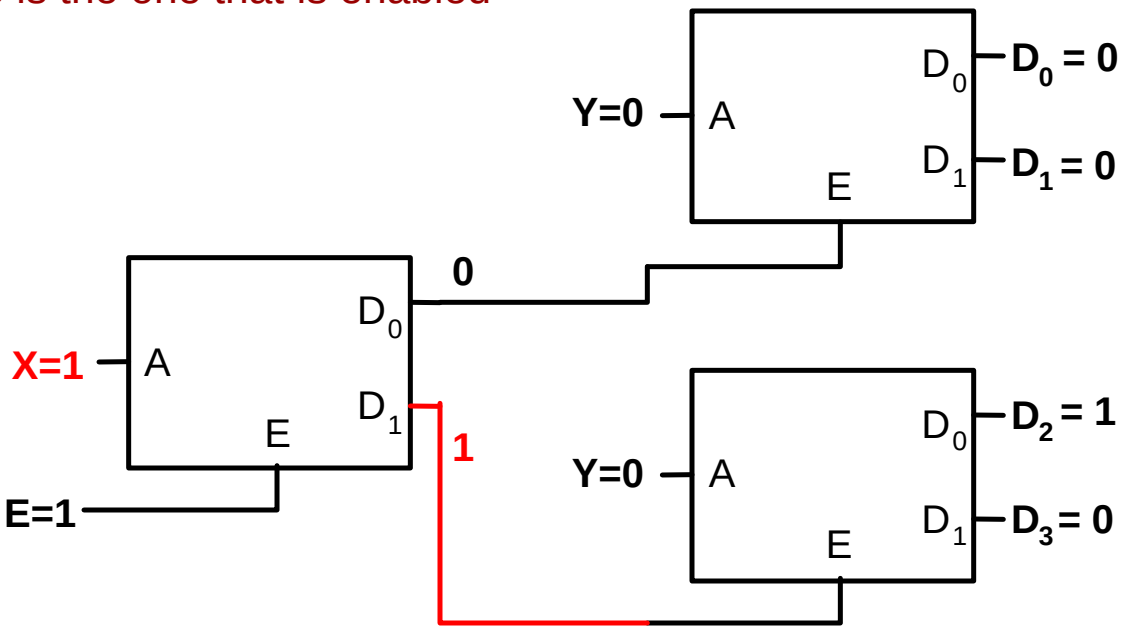
- To understand how this works think of the process of elimination
  - Given a 2-bit number X,Y (X = MSB)
  - If I tell you X=1, what are the possible numbers we can have…2 or 3
  - If I then tell you Y=0, then you know the number is 2
- By decoding one bit at a time we can eliminate half of the possibilities until we get down to the actual number

Cascading Decoders

Example:
X=1,Y=0

w/ X=1 we can narrow it down to D_2 or D_3…and you see that the lower decoder in the second stage is the one that is enabled

E	X	Y	D0	D1	D2	D3
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



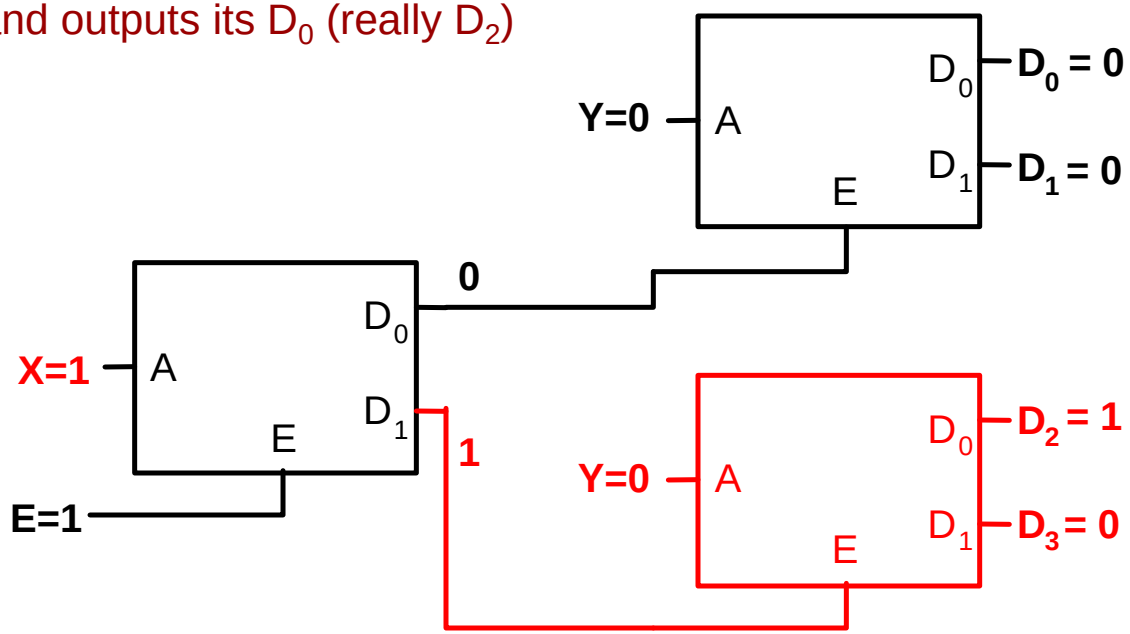
Cascading Decoders

Example:
X=1,Y=0

The top decoder is disabled so its outputs are forced to 0

The bottom decoder decodes the Y bit and outputs its D_0 (really D_2)

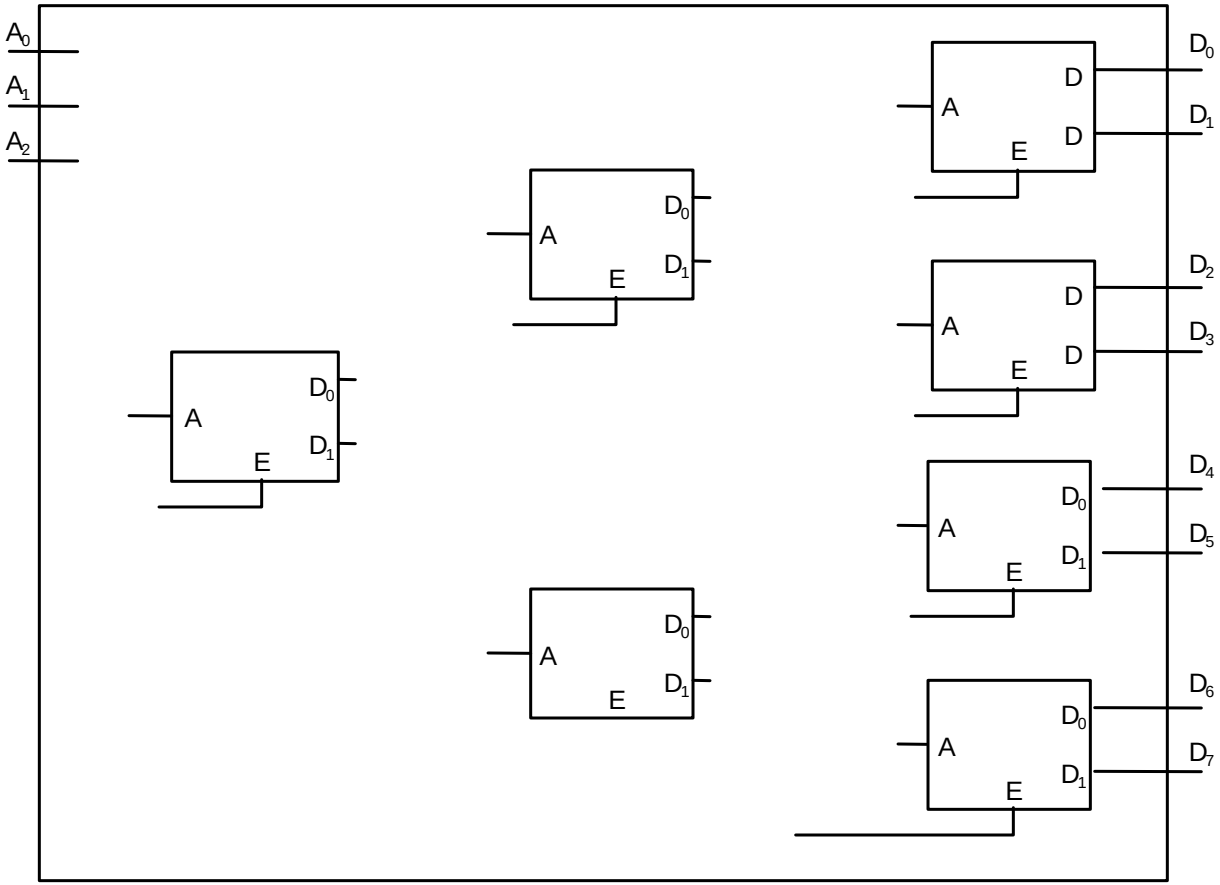
E	X	Y	D0	D1	D2	D3
0	x	x	0	0	0	0
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1



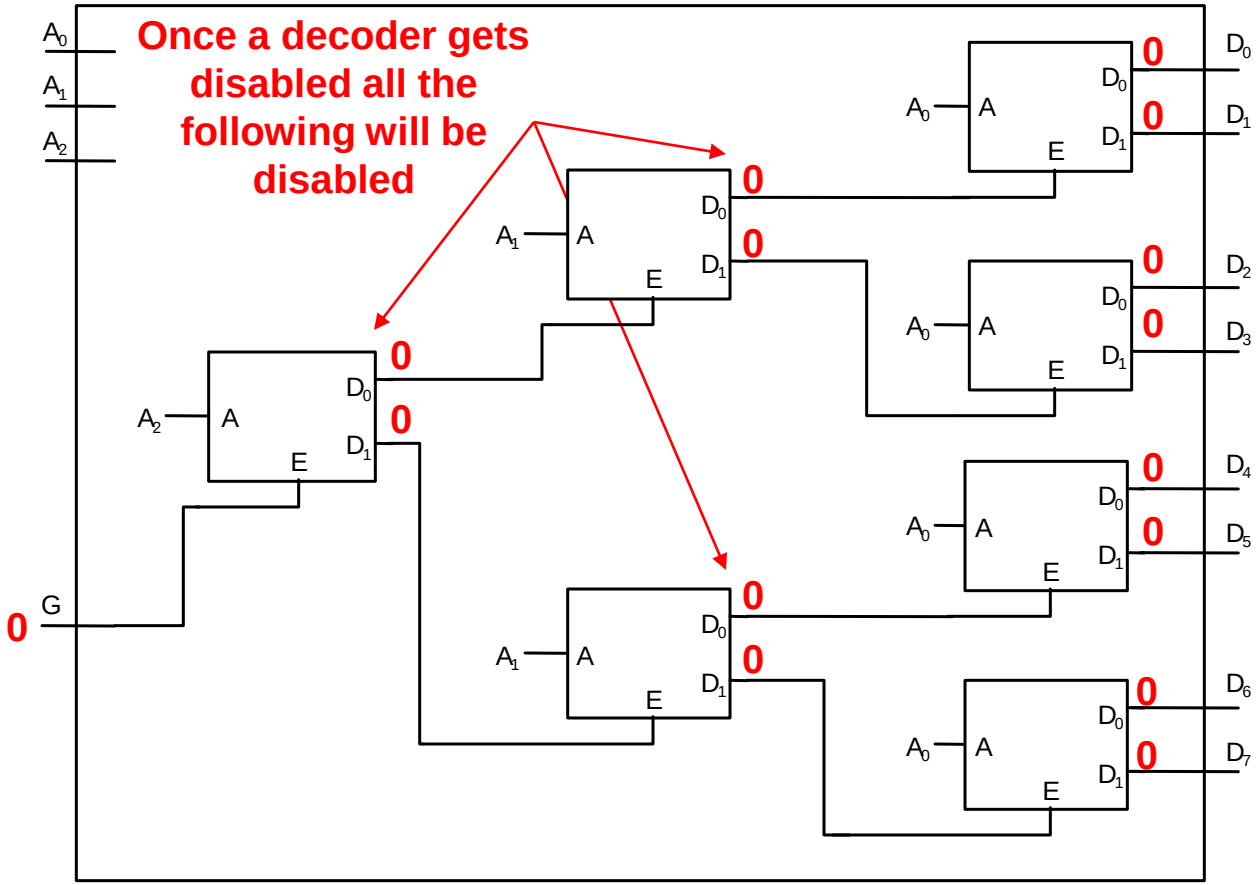
Rules for Making Larger Decoders

- Rule 1: Outputs of one stage should connect to the enables of the next stage
- Rule 2: All decoders in a stages should decode the same bit(s)
  - Usually, the MSB is connected to the first stage and LSB to the last stage

Build a 3-to-8 Decoder



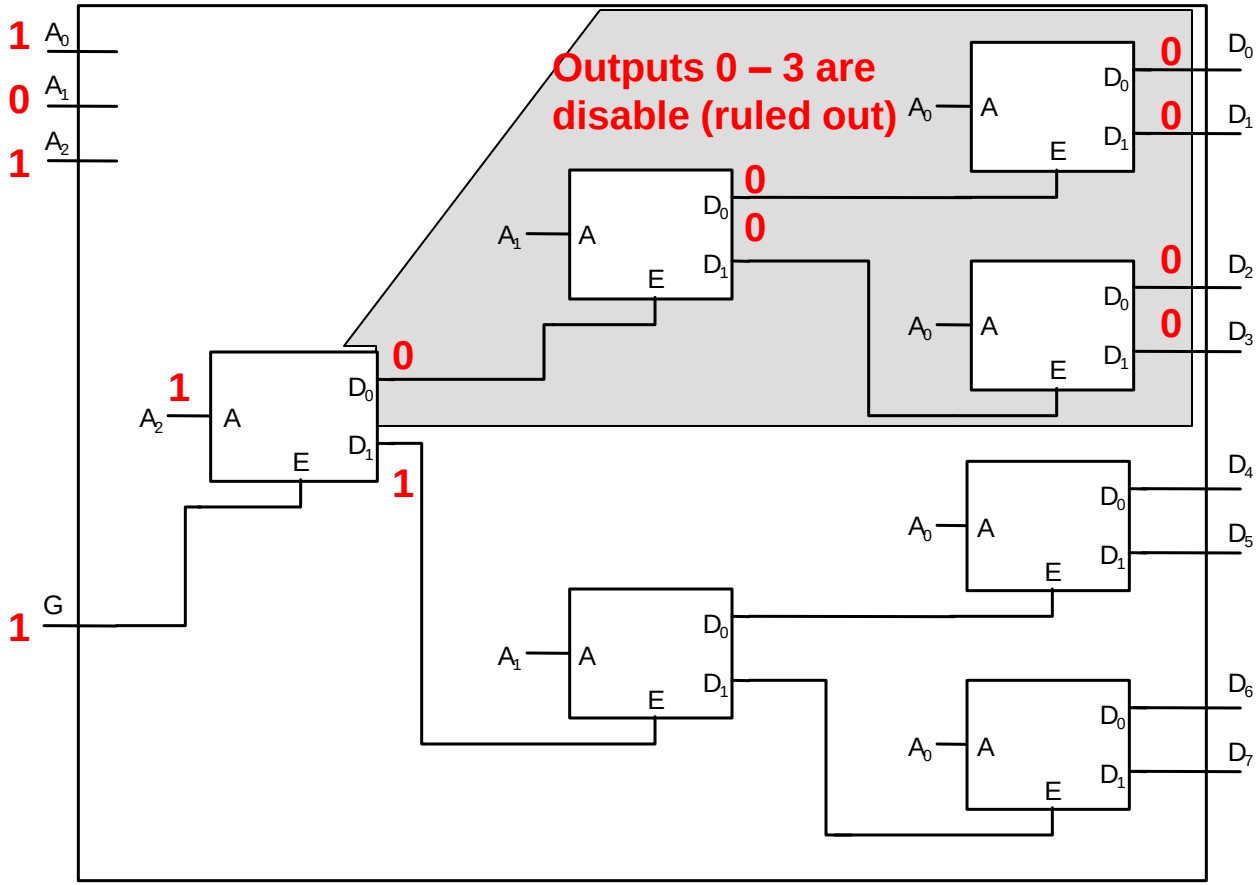
Cascading Decoders



G	A_2	A_1	A_0	Active Output
0	X	X	X	None
1	0	0	0	D_0
1	0	0	1	D_1
1	0	1	0	D_2
1	0	1	1	D_3
1	1	0	0	D_4
1	1	0	1	D_5
1	1	1	0	D_6
1	1	1	1	D_7

Cascading Decoders

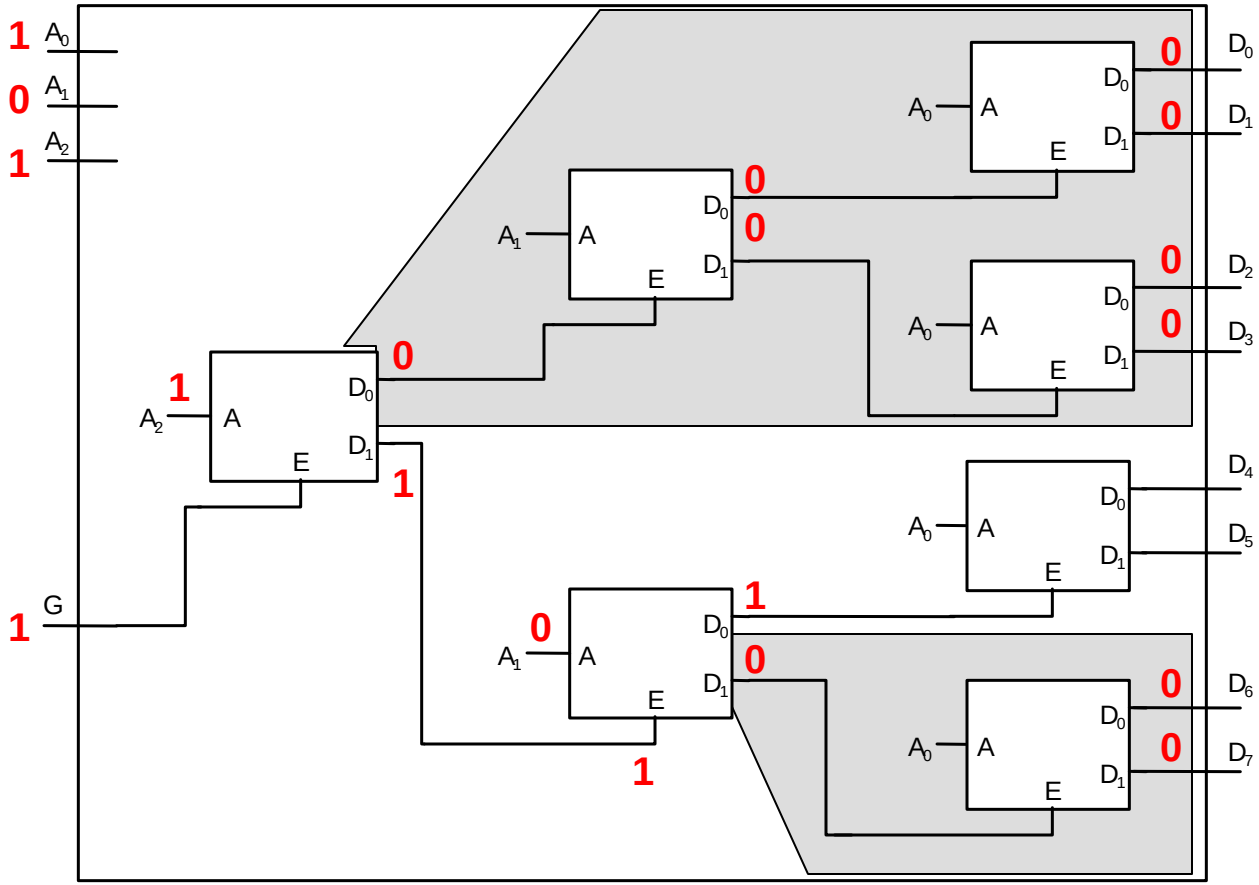
Decode the MSB…possible combos = 4-7



G	A_2	A_1	A_0	Active Output
0	X	X	X	None
1	0	0	0	D_0
1	0	0	1	D_1
1	0	1	0	D_2
1	0	1	1	D_3
1	1	0	0	D_4
1	1	0	1	D_5
1	1	1	0	D_6
1	1	1	1	D_7

Cascading Decoders

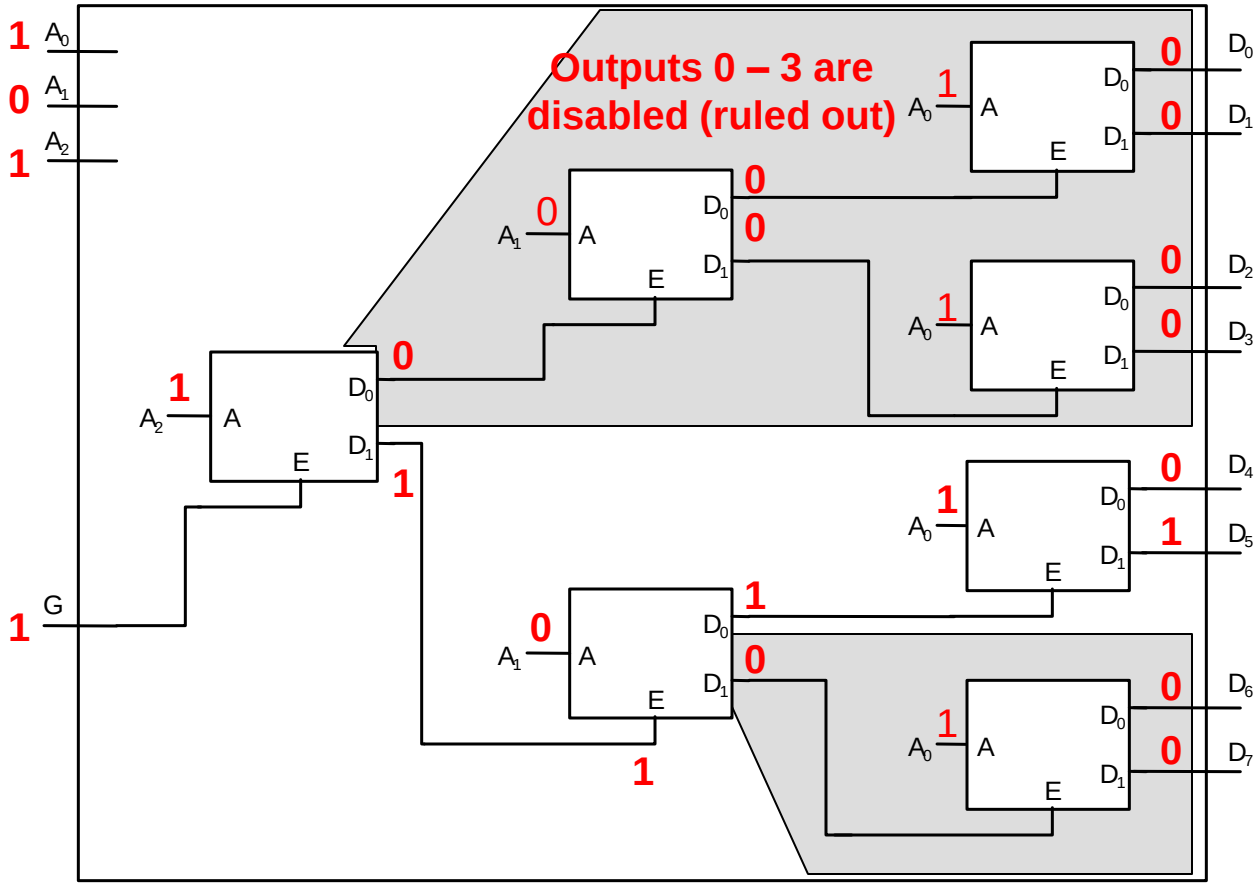
Decode the A_1 …possible combos = 5-6



G	A_2	A_1	A_0	Active Output
0	X	X	X	None
1	0	0	0	D_0
1	0	0	1	D_1
1	0	1	0	D_2
1	0	1	1	D_3
1	1	0	0	D_4
1	1	0	1	D_5
1	1	1	0	D_6
1	1	1	1	D_7

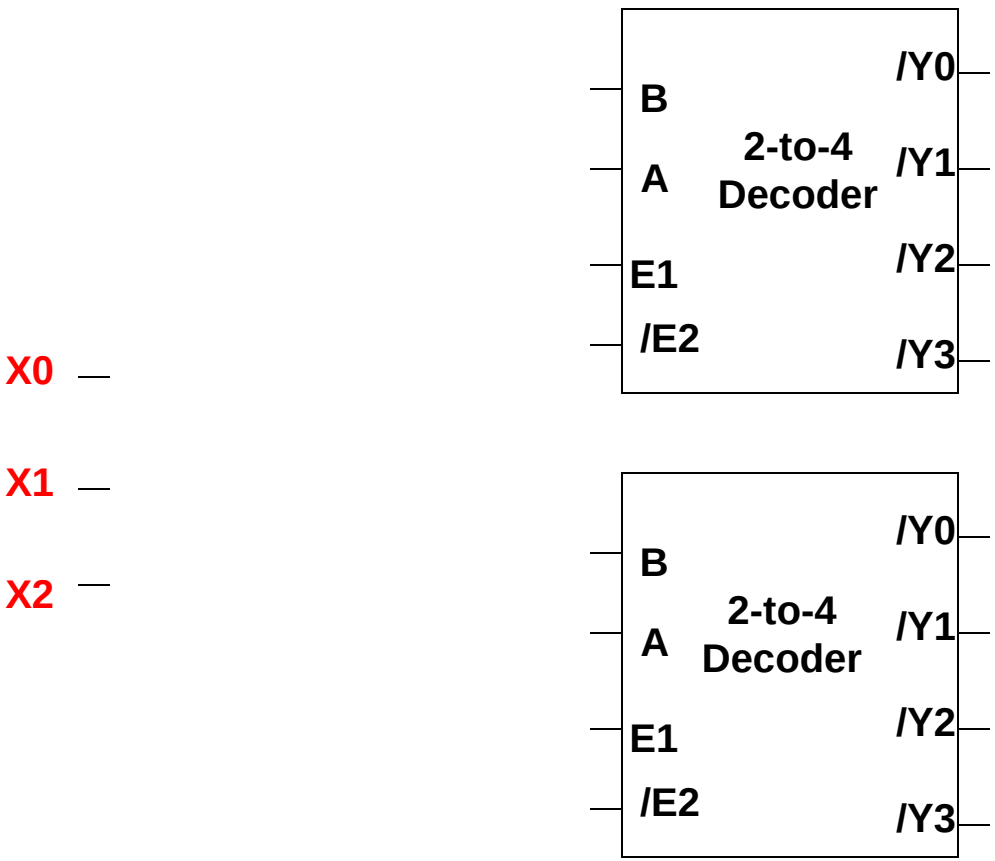
Cascading Decoders

Decode the LSB…combo = 5



G	A_2	A_1	A_0	Active Output
0	X	X	X	None
1	0	0	0	D_0
1	0	0	1	D_1
1	0	1	0	D_2
1	0	1	1	D_3
1	1	0	0	D_4
1	1	0	1	**D_5**
1	1	1	0	D_6
1	1	1	1	D_7

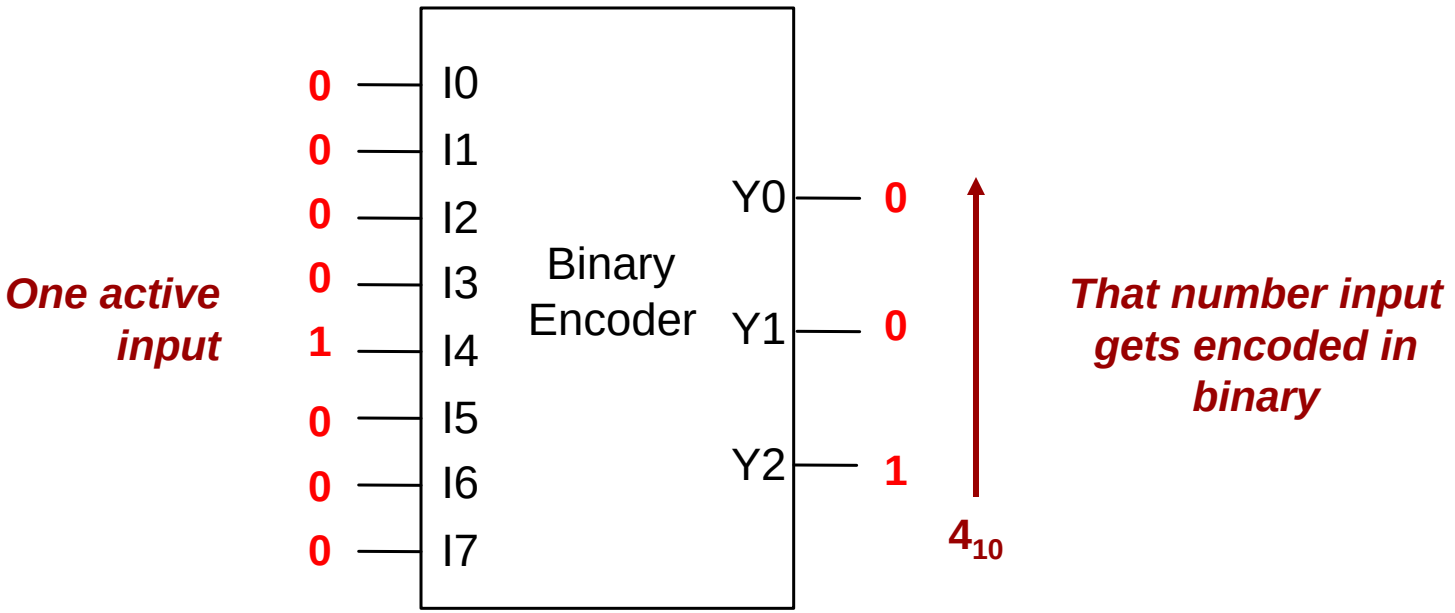
Build a 3-to-8 Decoder



SIMPLE & PRIORITY ENCODERS

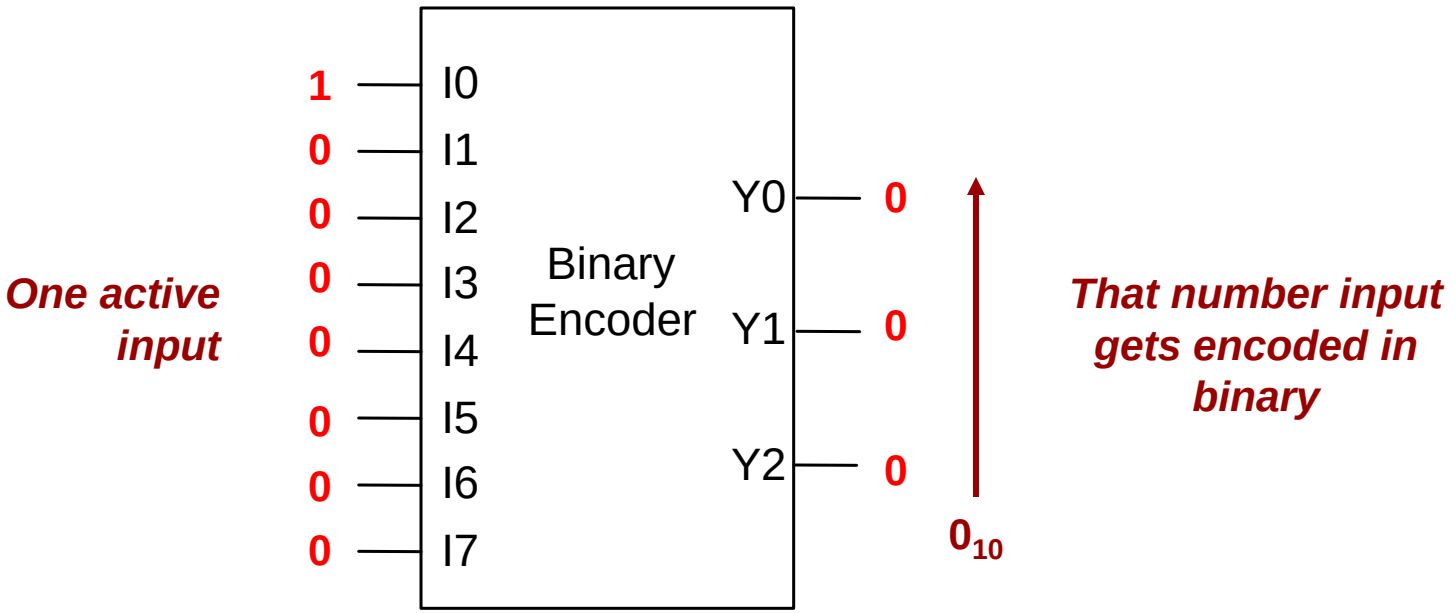
Encoders

- Opposite function of decoders
- Takes in 2^n inputs and produces an n-bit number



Encoders

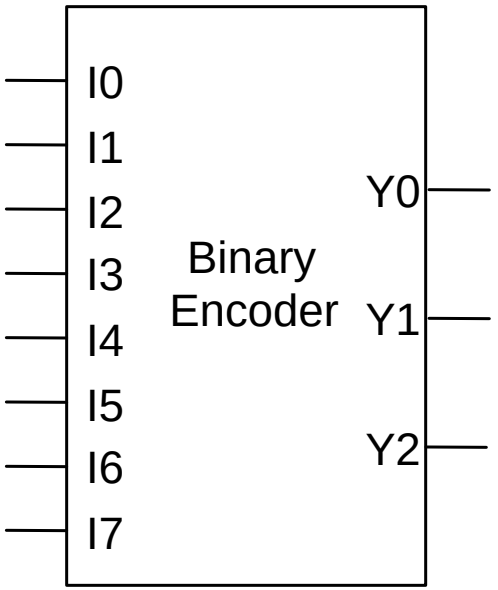
- Assumption: Only one input will be active at a time



Encoders

- What's inside an encoder?

I_0	I_1	I_2	I_3	I_4	I_5	I_6	I_7	Y_2	Y_1	Y_0
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

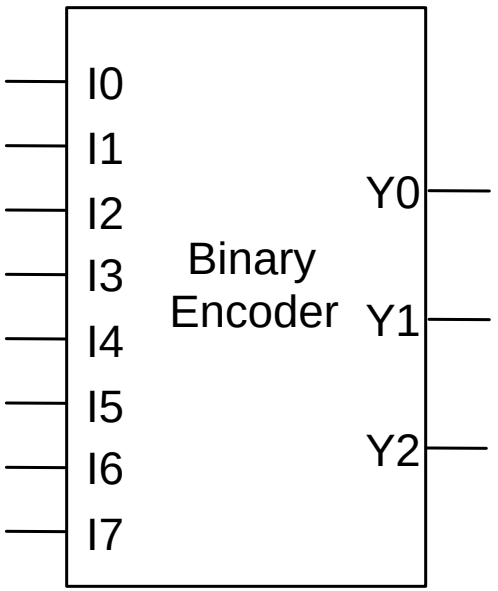


Deriving equations for Y_0, Y_1, Y_2 is made simpler because of the assumption that only 1 input can be active at a time. Rather than having 256 rows in our truth table we only have 8

Encoders

- What's inside an encoder?

I_0	I_1	I_2	I_3	I_4	I_5	I_6	I_7	Y_2	Y_1	Y_0
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



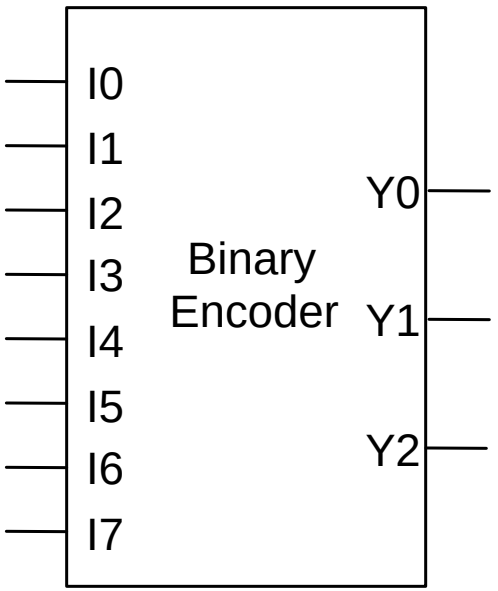
Y_2 = 1 when I_4 = 1 or I_5 = 1 or I_6 = 1 or I_7 = 1…

$Y_2 = I4 + I5 + I6 + I7$

Encoders

- What's inside an encoder?

I_0	I_1	I_2	I_3	I_4	I_5	I_6	I_7	Y_2	Y_1	Y_0
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



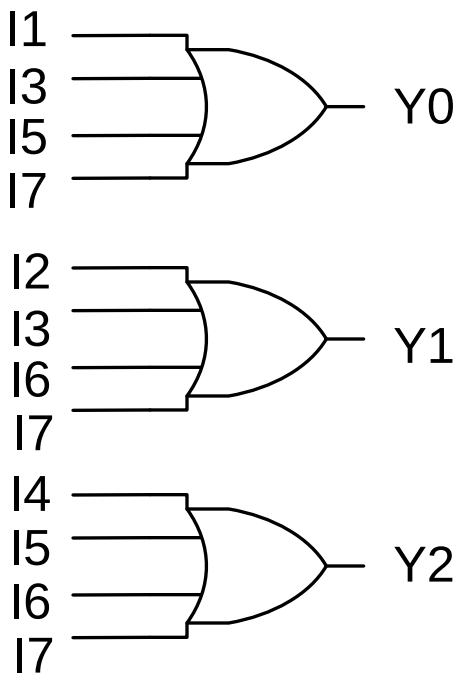
$Y_2 = I4 + I5 + I6 + I7$

$Y_1 = I2 + I3 + I6 + I7$

$Y_0 = I1 + I3 + I5 + I7$

Encoders

- A simple binary encoder can be made with just OR gates

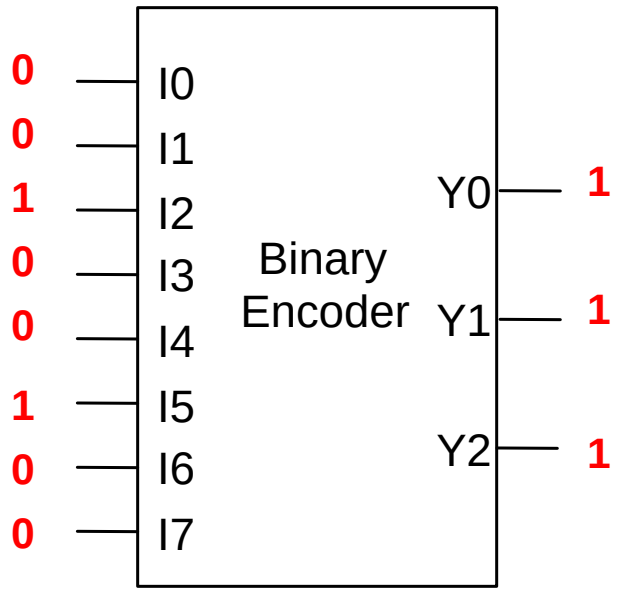


Problems

- There is a problem…
  - Our assumption is that only 1 input can be active at a time
  - What happens if 2 or more inputs are active or if 0 inputs are active

2 or More Active Inputs

- What if I5 and I2 are active at the same time?
  - Substitute values into equation
- Output will be '111' = 7
- Output is neither 2 nor 5, it's something different, 7



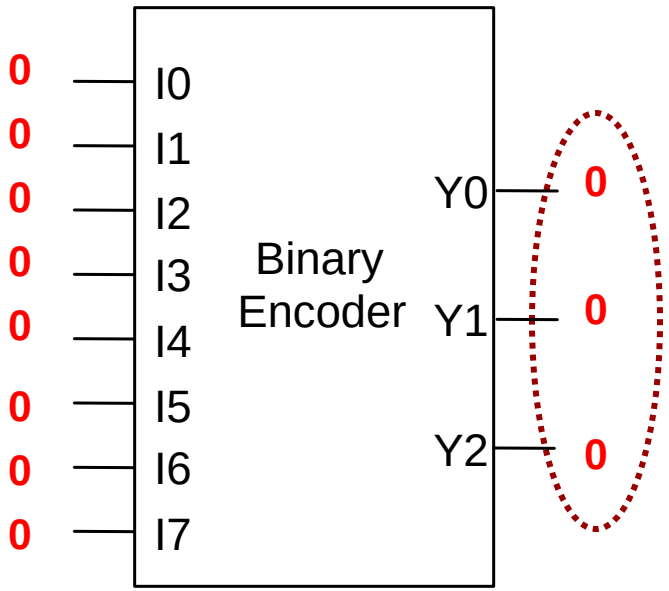
$Y_2 = I4 + I5 + I6 + I7$

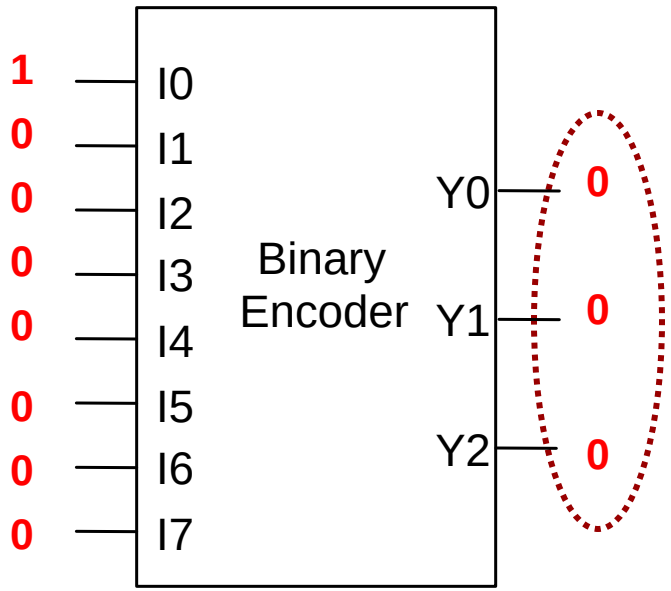
$Y_1 = I2 + I3 + I6 + I7$

$Y_0 = I1 + I3 + I5 + I7$

0 Active Inputs

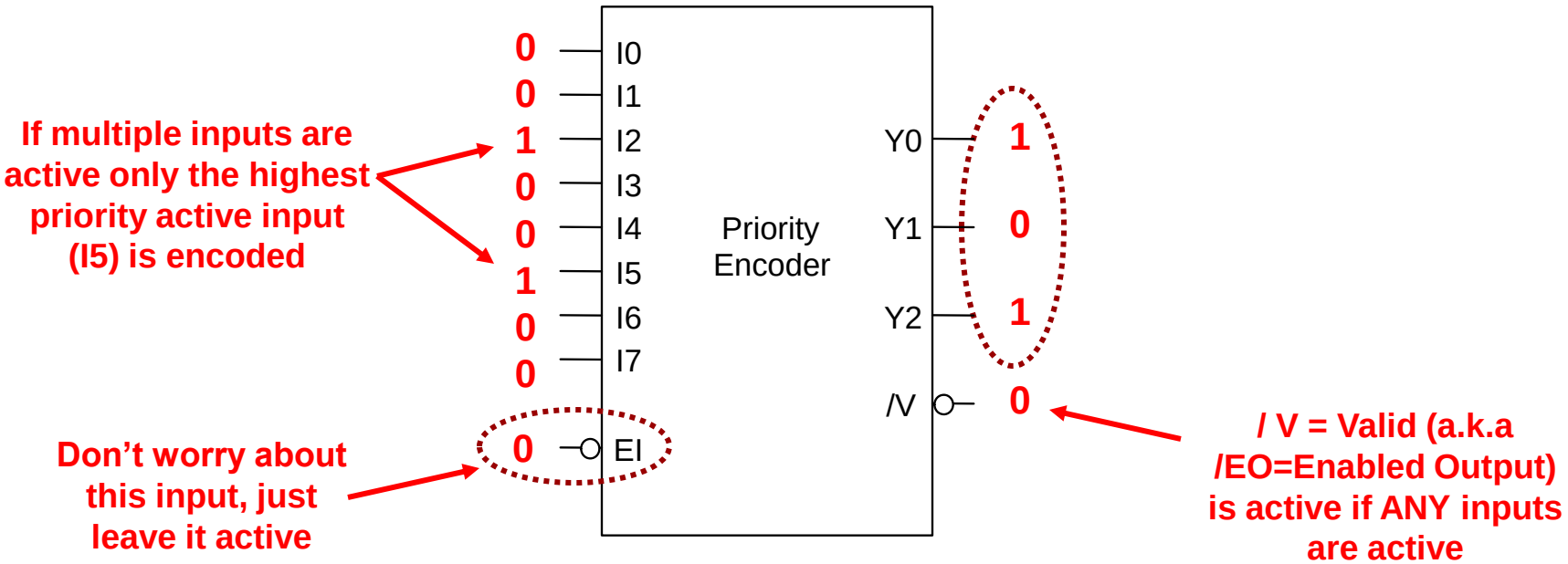
- What if no inputs are active?
  - Substitute values into equation
- Output will be '000' = 0
- Problem: '000' means that input 0 was active
  - Can't tell the difference between when '000' means input 0 was active or no inputs was active





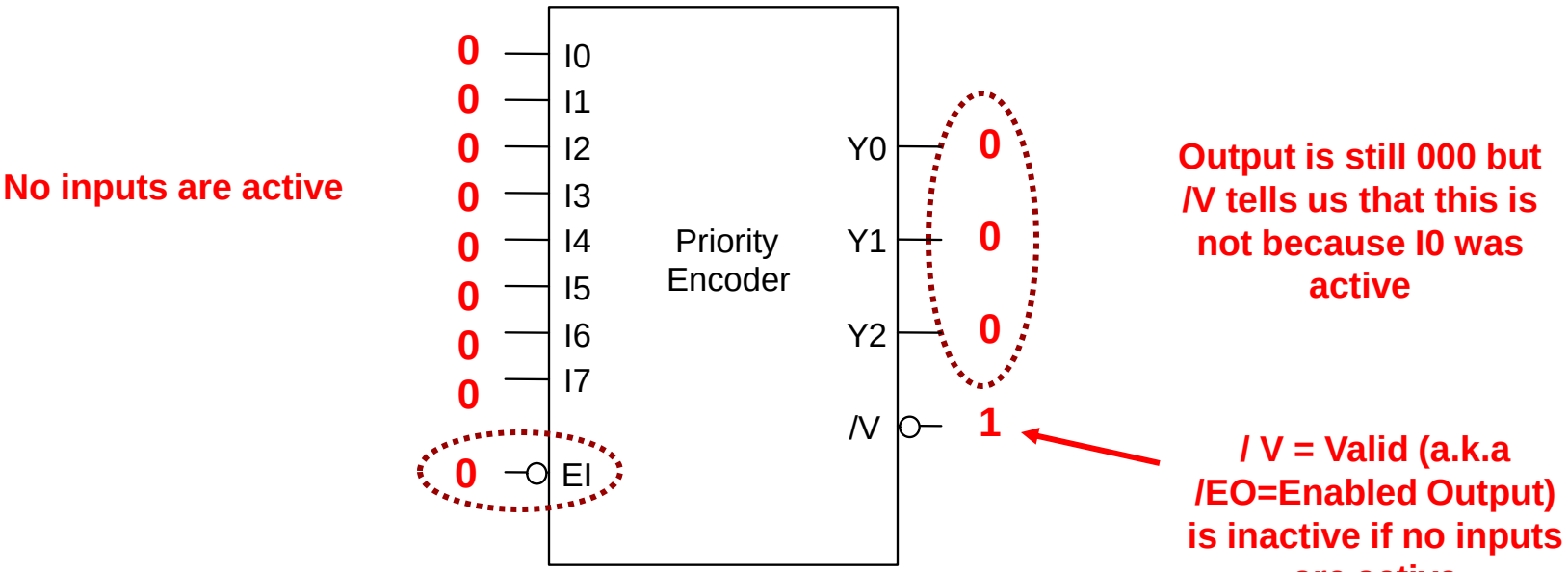
Priority Encoders

- Fix the 2 problems seen above
- Problem of more than 2 active inputs
  - Assign priority to inputs and only encode the highest priority active input
- Problem of zero active inputs
  - Create an extra output to indicate if any inputs are active
  - We will call this output the “Valid” output (/V)



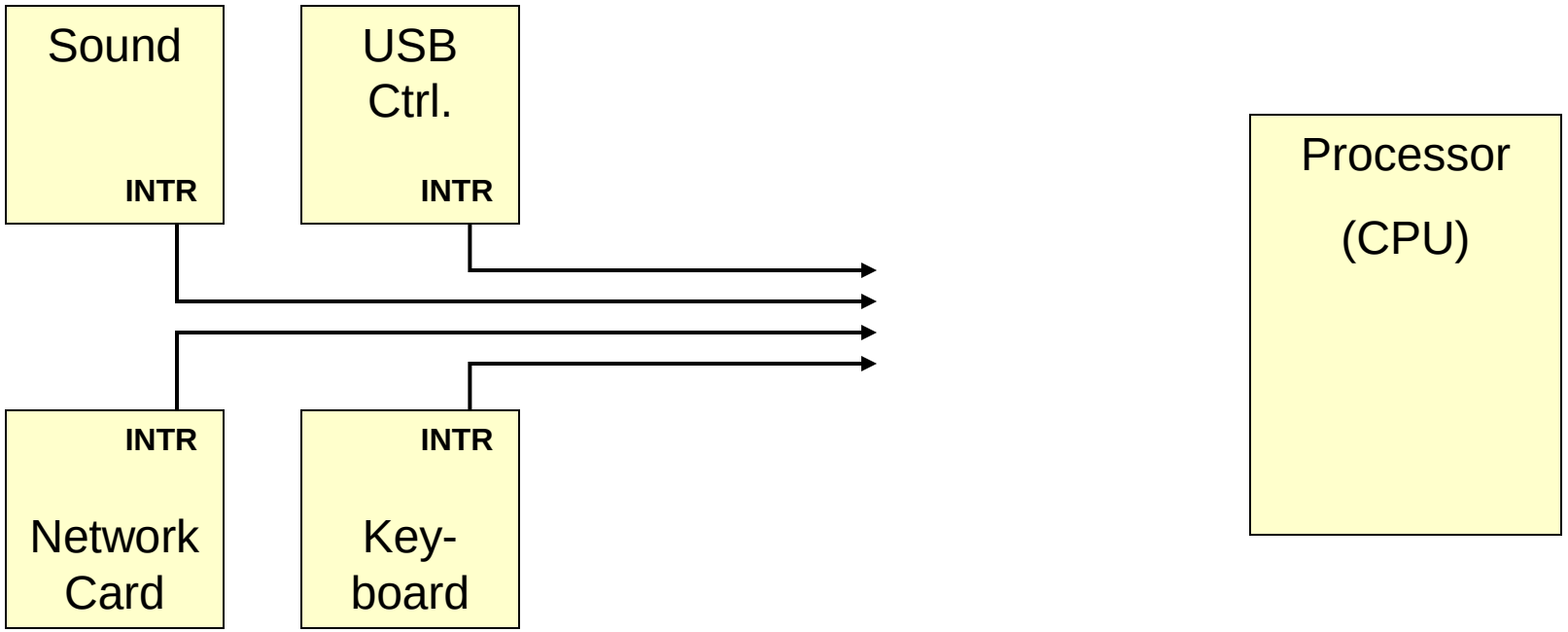
Priority Encoders

- Fix the 2 problems seen above
- Problem of more than 2 active inputs
  - Assign priority to inputs and only encode the highest priority active input
- Problem of zero active inputs
  - Create an extra output to indicate if any inputs are active



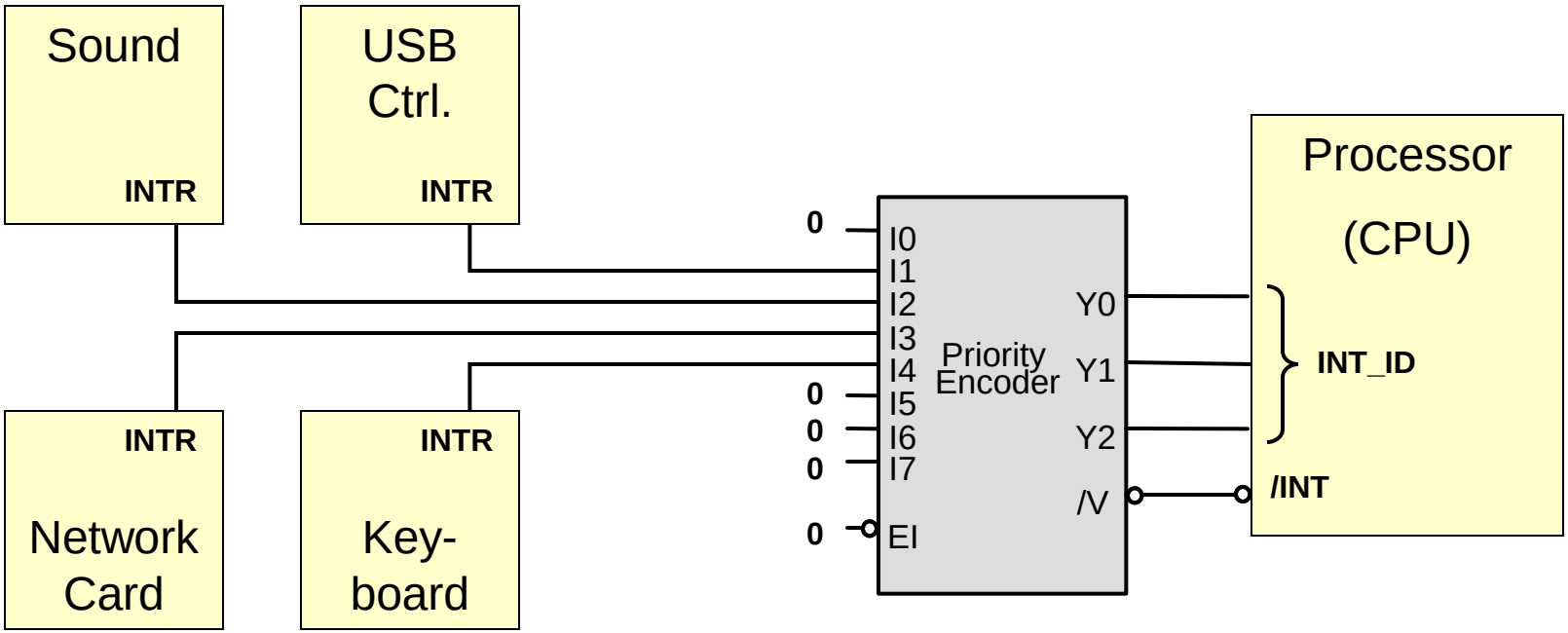
Encoder Application: Interrupts

- I/O Devices in a computer need to request attention from the CPU…they need to "interrupt" the processor
- CPU cannot have a dedicated line to each I/O device (too many inputs and outputs) plus it can only service one device at a time



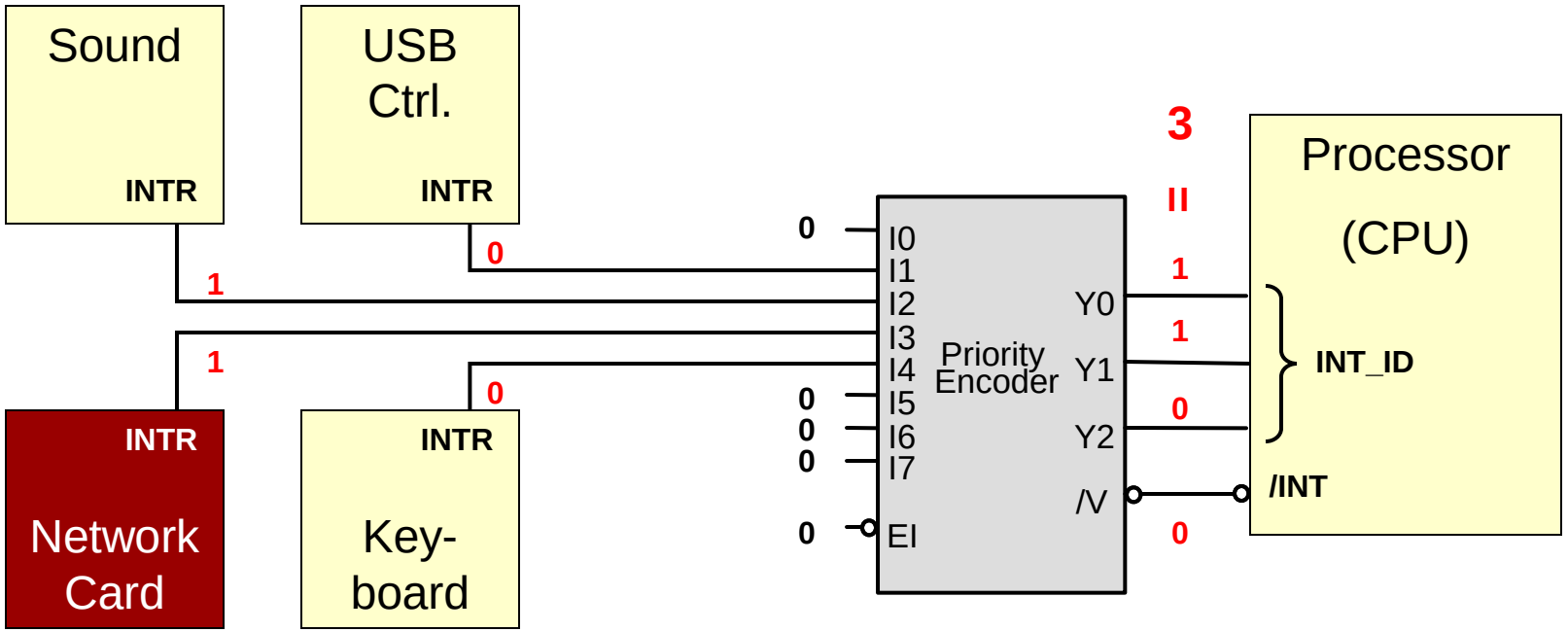
Encoder Application: Interrupts

- Solution: Priority Encoder
- /INT input of CPU indicates SOME device is requesting attention
- INT_ID inputs identify who is requesting attention



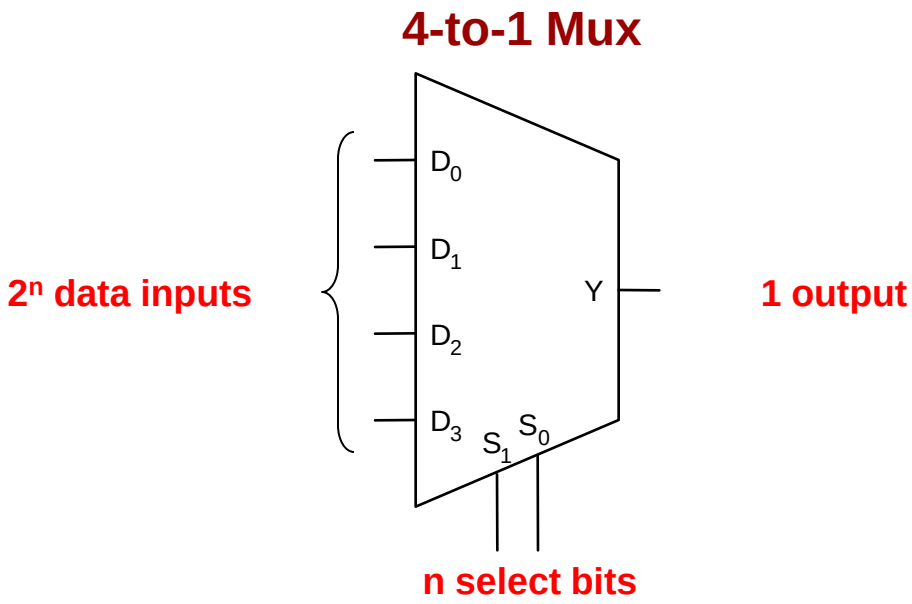
Encoder Application: Interrupts

- Example: Sound and Network request interrupt at the same time
- Network is highest priority and is encoded
- After network is handled, sound will cause interrupt

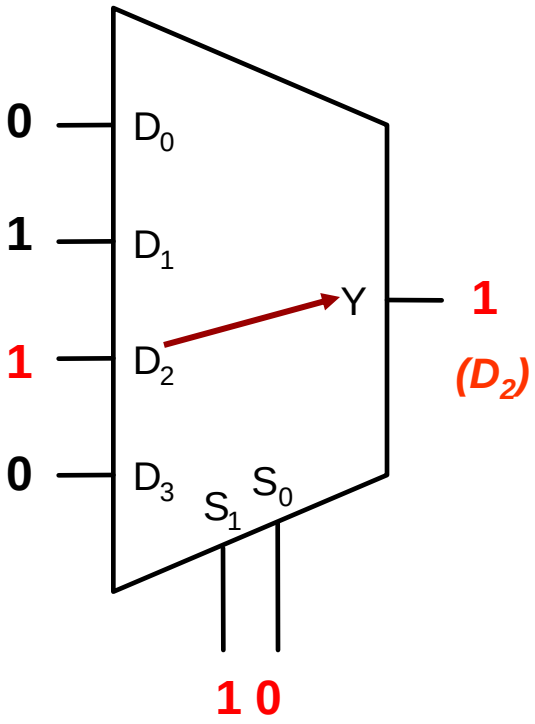


Multiplexers

- Along with adders, multiplexers are most used building block
- 2^n data inputs, n select bits, 1 output
- A multiplexer ("mux" for short) selects one data input and passes it to the output



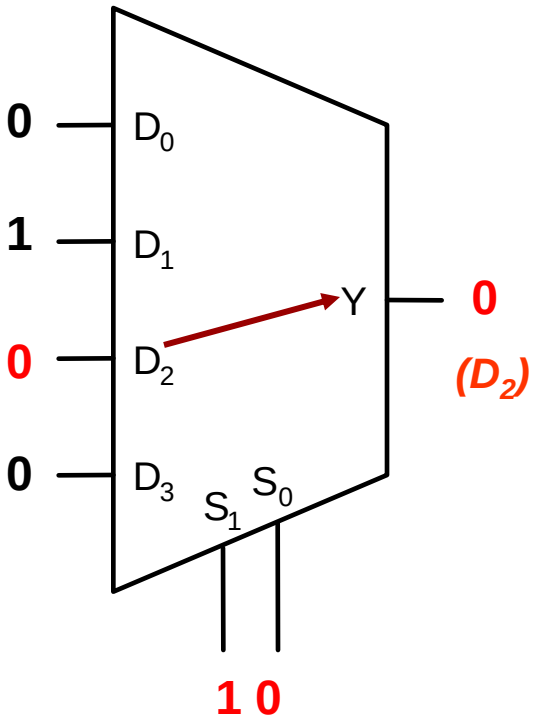
Multiplexers



② Thus, D2 is selected and passed to the output

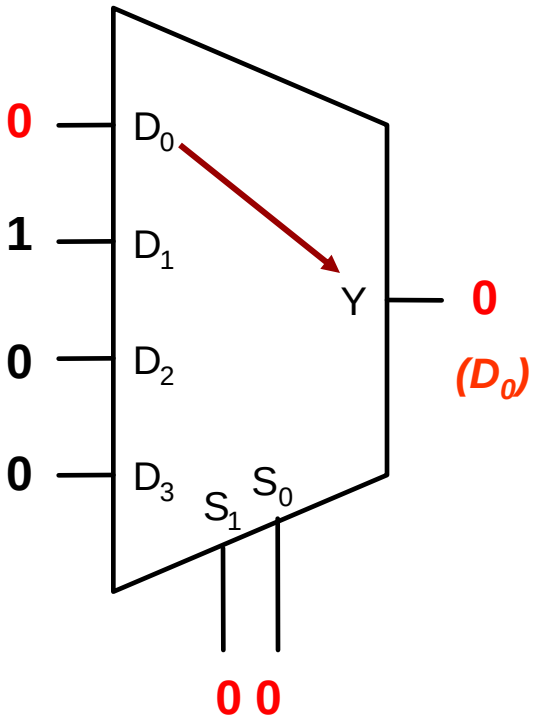
① Select bits = $10_2 = 2_{10}$.

Multiplexers



D2 is being selected and passed.
So if it changes the output changes as well.

Multiplexers

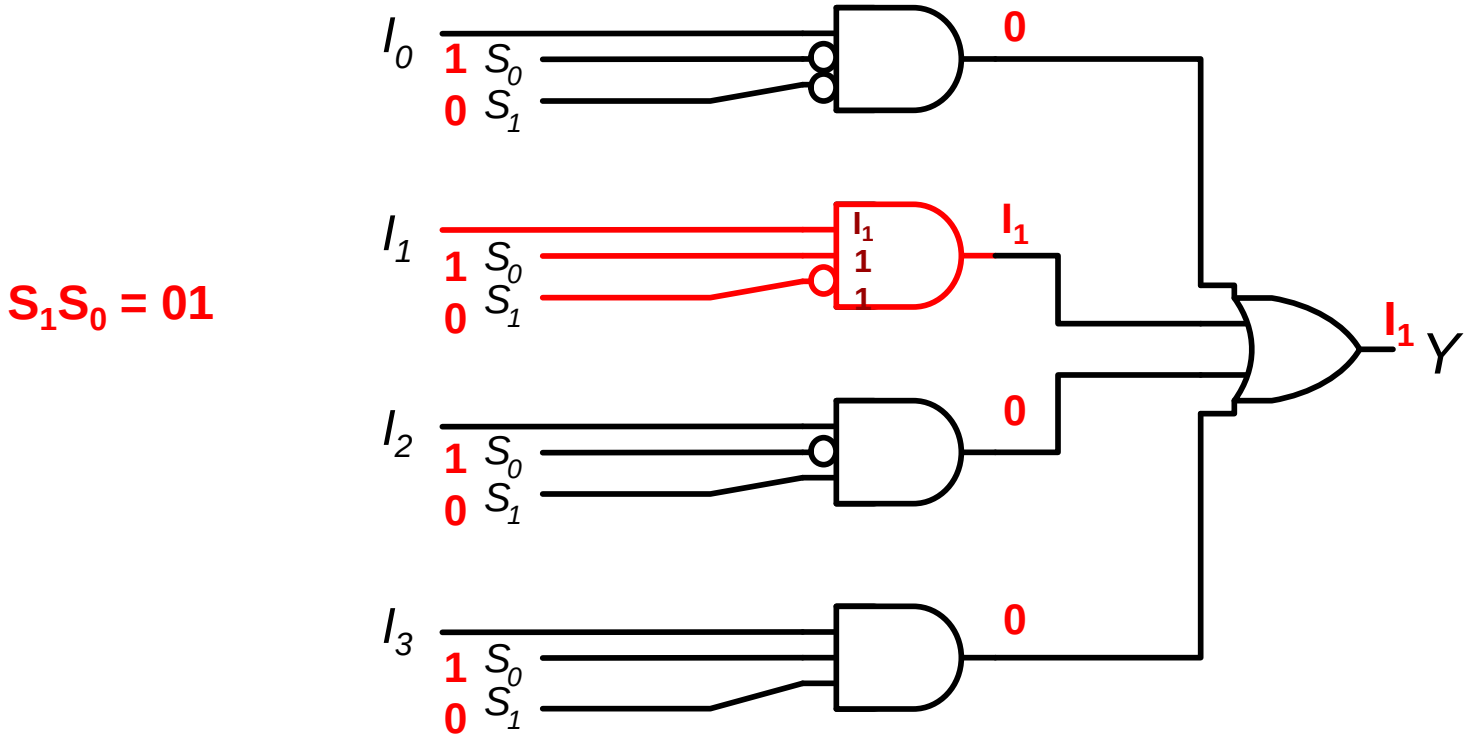


② Thus, D0 is selected and passed to the output

① Select bits = $00_2 = 0_{10}$.

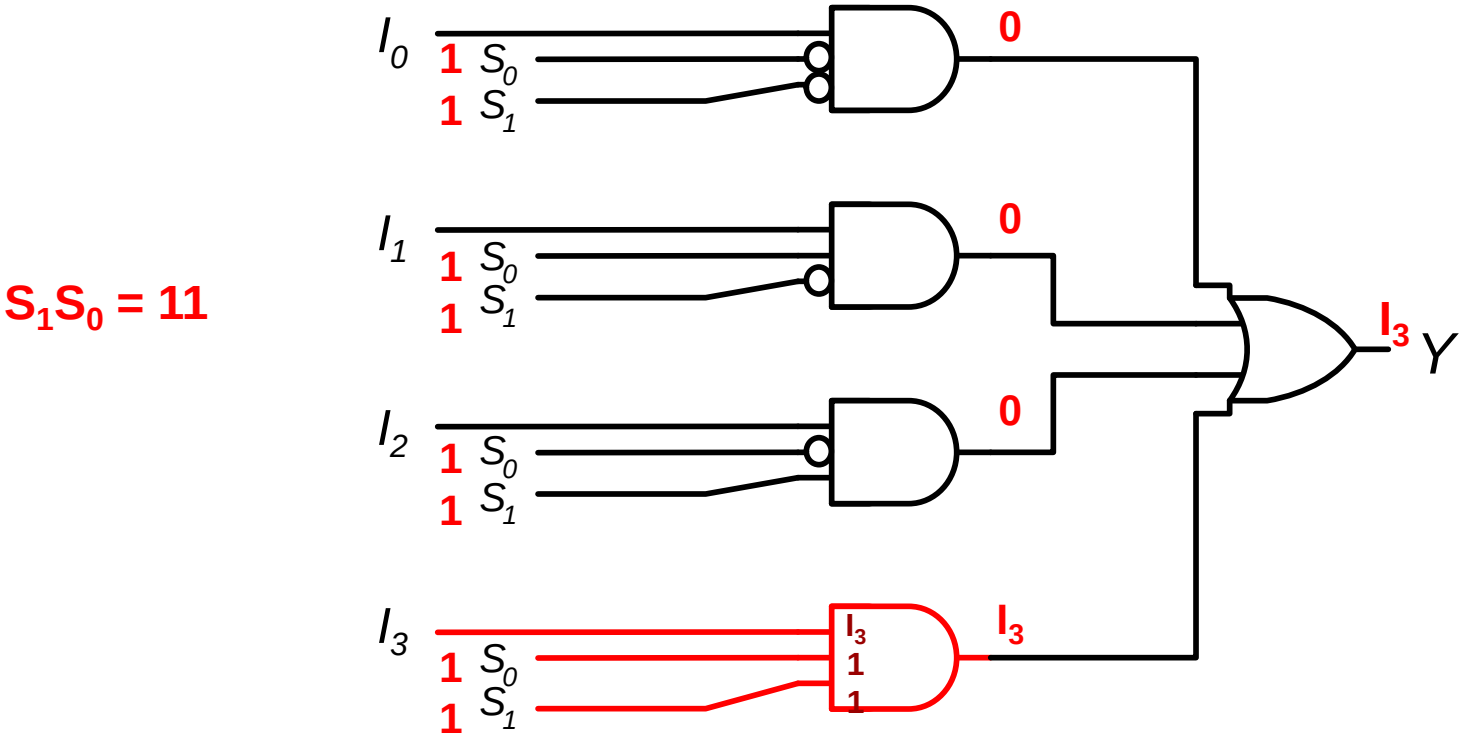
Building a Mux

- To build a mux
  - Decode the select bits and include the corresponding data input.
  - Finally OR all the first level outputs together.



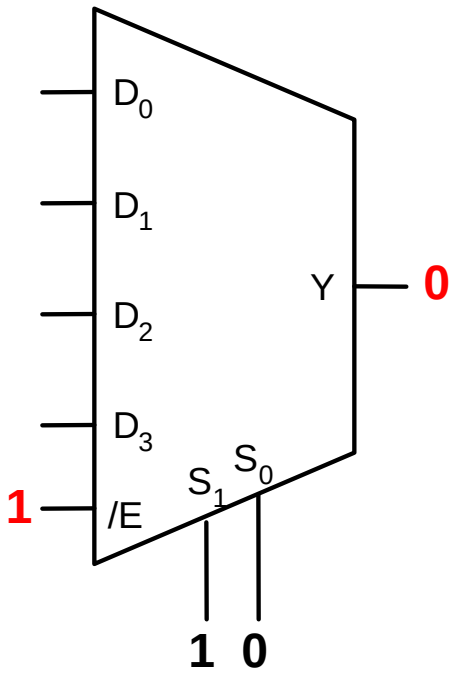
Building a Mux

- To build a mux
  - Decode the select bits and include the corresponding data input.
  - Finally OR all the first level outputs together.

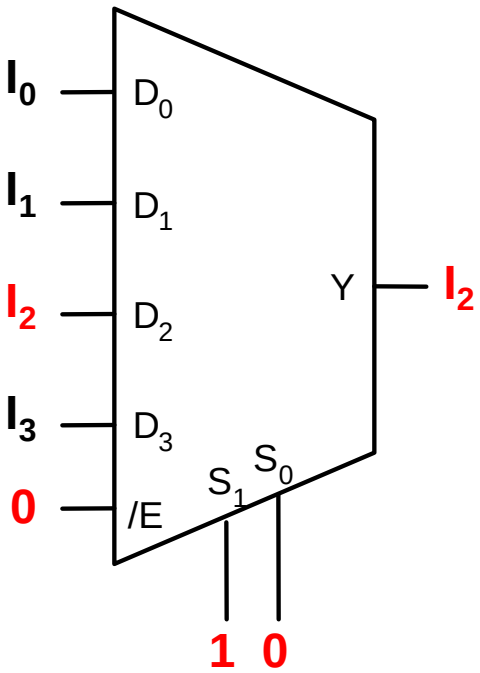


Adding Enables to Muxes

- When Enable = inactive, Y = inactive
- When Enable = active, normal mux

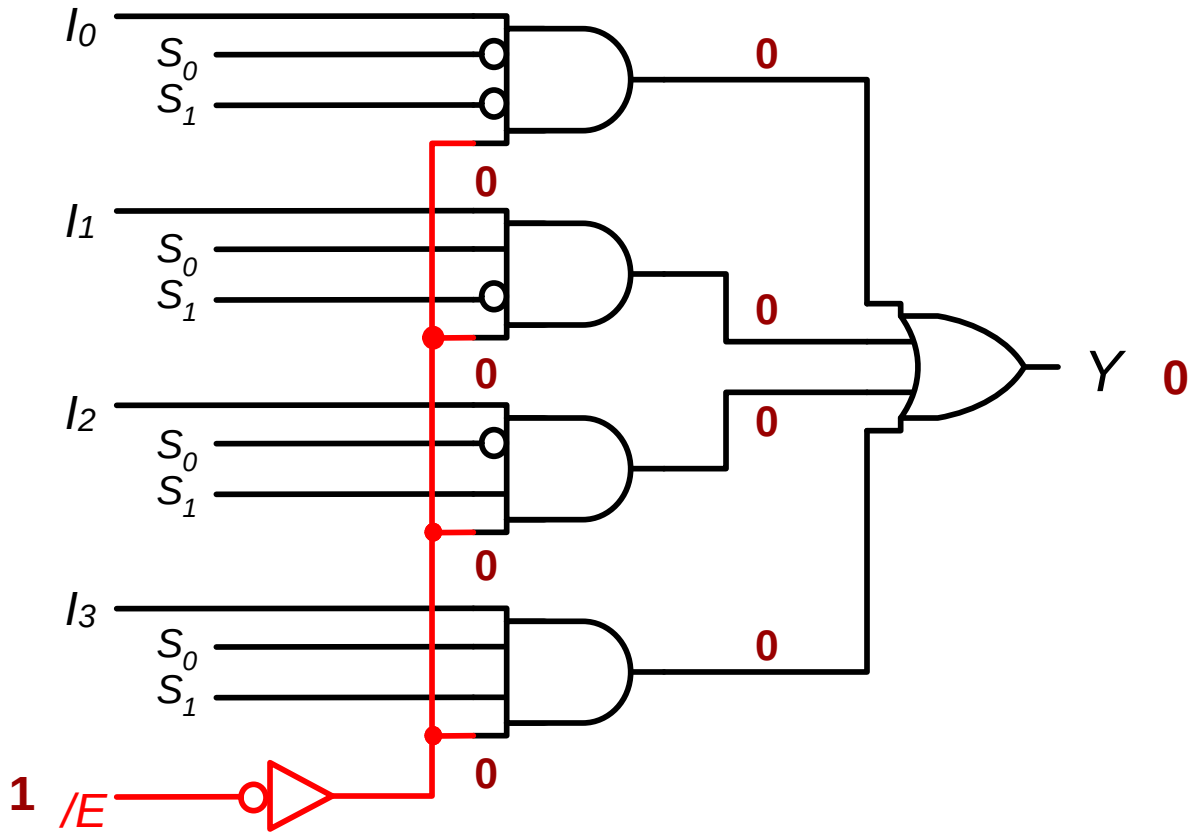


/E = inactive forces output to 0



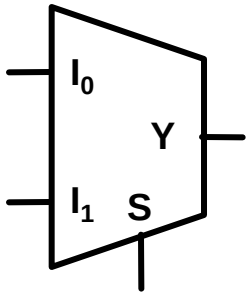
/E = active allows normal mux function

Adding Enables to Muxes

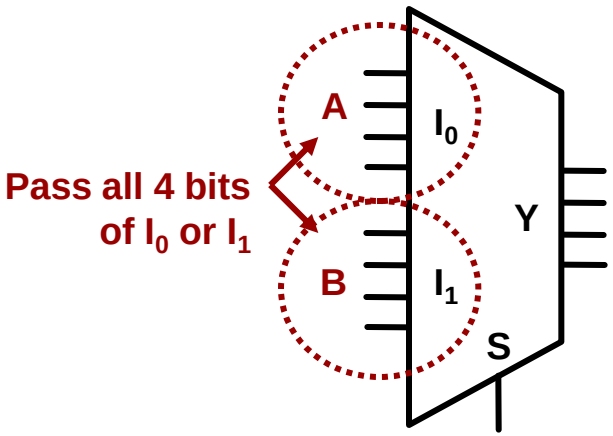


Building Wide Muxes

- So far muxes only have single bit inputs…
  - I_0 is only 1-bit
  - I_1 is only 1-bit
- What if we still want to select between 2 inputs but now each input is a 4-bit number
- Use a 4-bit wide 2-to-1 mux



1-bit wide 2-to-1 mux



When we select I_0 or I_1 we want all 4-bits of that input to be passed

4-bit wide 2-to-1 mux

Building Wide Muxes

- To build a 4-bit wide 2-to-1 mux, use 4 separate 2-to-1 muxes
- When S=0, all muxes pass their I_0 inputs which means all the A bits get through
- When S=1, all muxes pass their I_1 inputs which means all the B bits get through
- In general, to build an **m-bit wide n-to-1 mux**, use **m individual (separate) n-to-1 muxes**