Unit 7

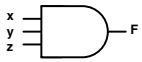
Fundamental Digital Building Blocks:

Decoders & Multiplexers

CHECKERS / DECODERS

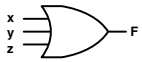
Gates

- Gates can have more than 2 inputs but the functions stay the same
  - AND = output = 1 if ALL inputs are 1
    - Outputs 1 for only 1 input combination
  - OR = output = 1 if ANY input is 1
    - Outputs 0 for only 1 input combination



X	Y	Z	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

3-input AND

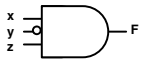


X	Y	Z	F
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

3-input OR

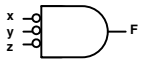
Checkers / Decoders

- An AND gate only outputs '1' for 1 combination
  - That combination can be changed by adding inverters to the inputs
  - We can think of the AND gate as "checking" or "decoding" a specific combination and outputting a '1' when it matches.



X	Y	Z	F
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

AND gate decoding (checking for) combination ____

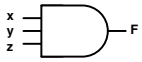


X	Y	Z	F
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

AND gate decoding (checking for) combination ____

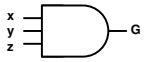
Checkers / Decoders

- Place inverters at the input of the AND gates such that
  - F produces '1' only for input combination {x,y,z} = {010}
  - G produces '1' only for input combination {x,y,z} = {110}



AND gate decoding (checking for) combination 010

X	Y	Z	F
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

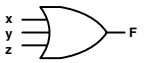


AND gate decoding (checking for) combination 110

X	Y	Z	G
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

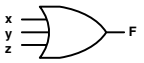
Checkers / Decoders

- An OR gate only outputs '0' for 1 combination
  - That combination can be changed by adding inverters to the inputs



Add inverters to create an OR gate decoding (checking for) combination 010

X	Y	Z	F
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1



Add inverters to create an OR gate decoding (checking for) combination 110

X	Y	Z	F
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	1

Decoder Exercise

- Compilers translate software to instructions that tell the processor to ADD, LOAD from Memory, Store to Memory, etc.
- These instructions are binary codes
- The processor must decode the instruction
- Create an AND gate decoder for each instruction type in the table that will produce '1' when that instruction is about to be executed

Instruction Type	6-bit OPCODE OP[5:0]
ADD	001000
LOAD	100011
STORE	101011
BRANCH	000100

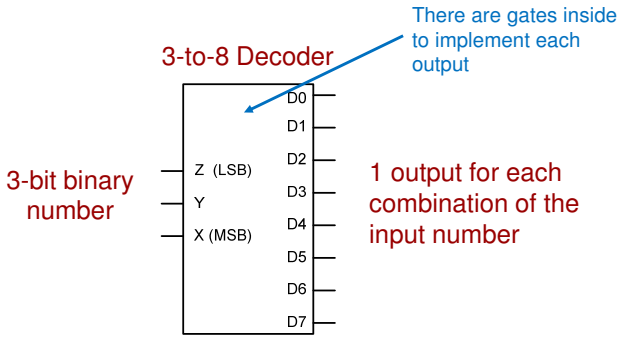
ADD _____

LOAD _____

STORE _____

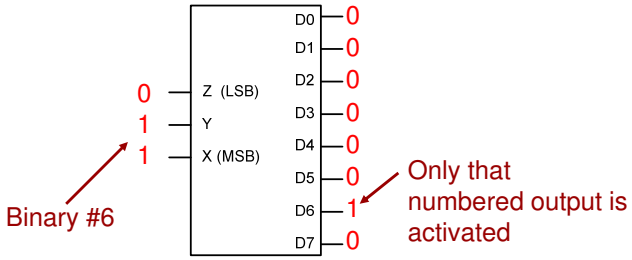
Full Decoders

- A full 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 _____ input combinations



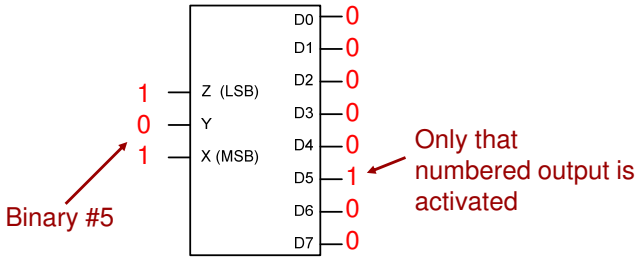
Decoders

- A decoder is a building block that:
  - Takes a binary number as input
  - Decodes that binary number and activates the corresponding output
  - Put in 6=110, Output 6 activates ('1')
  - Put in 5=101, Output 5 activates ('1')



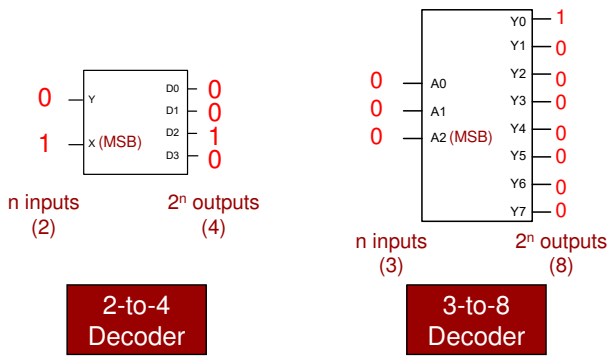
Decoders

- A decoder is a building block that:
  - Takes a binary number as input
  - Decodes that binary number and activates the corresponding output
  - Put in 6=110, Output 6 activates ('1')
  - Put in 5=101, Output 5 activates ('1')



Decoder Sizes

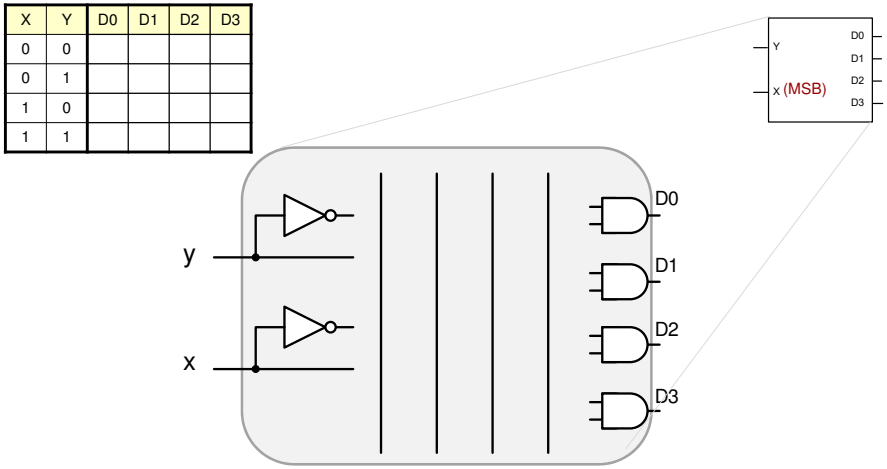
- A decoder w/ an **n-bit input** has **2^n outputs**
  - 1 output for every combination of the n-bit input



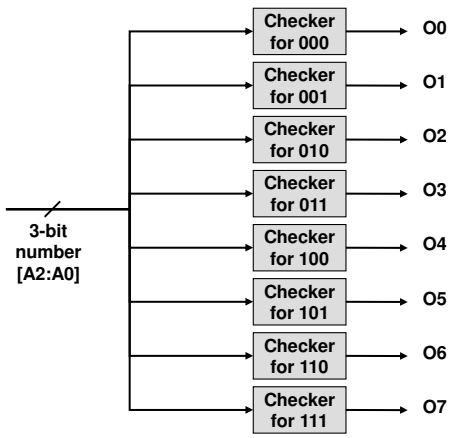
Exercise

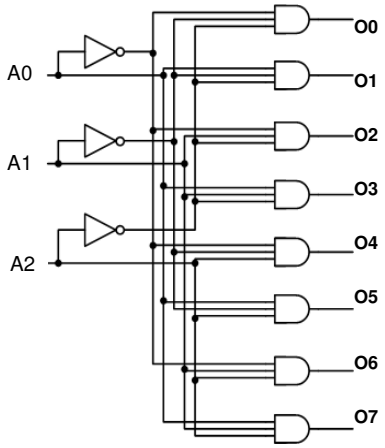
- Complete the design of a 2-to-4 decoder

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



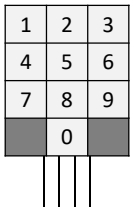
Building Decoders

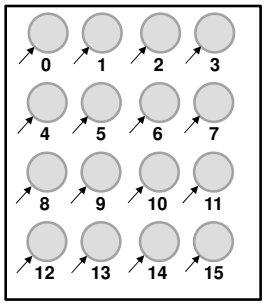




Vending Machine Example

Assuming the keypad produces a 4-bit numeric output, add logic to produce the release signals for each of the 16 vending items.

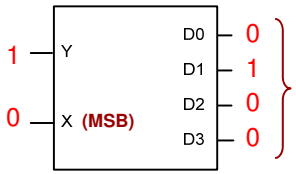




Consider any problems with this design.

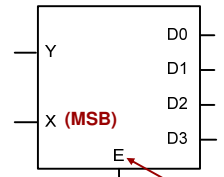
Enables

- In a normal decoder exactly one output is active at all times
- It may be undesirable to always have an active output
- We can add an extra input (called an enable) that can independently force all the outputs to their inactive values



One output will always be active

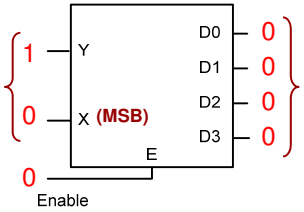
2-to-4 Decoder



Will force all outputs to 0 when E = 0 (i.e. not enabled)

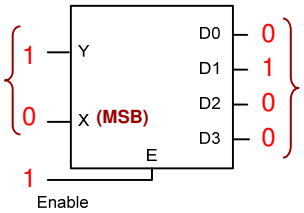
Enables

When E=0, inputs is ignored



Since E=0, all outputs = 0

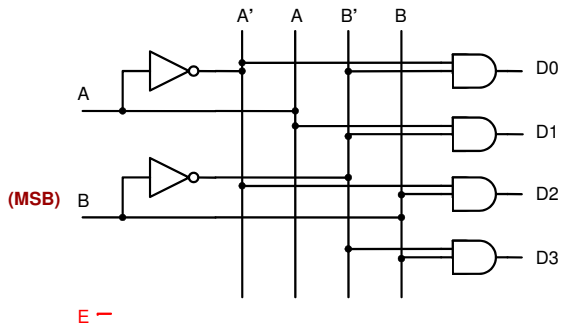
When E=1, inputs will cause the appropriate output to go active



Since E=1, outputs will function normally

Implementing Enables

- Original 2-to-4 decoder

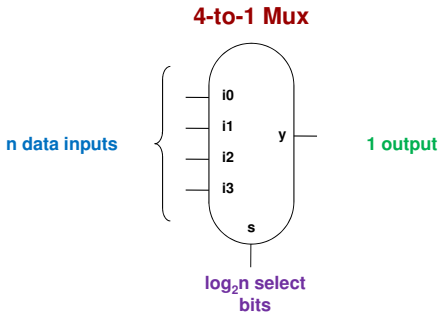


When E=0, force all outputs = 0

When E=1, outputs operate as they did originally

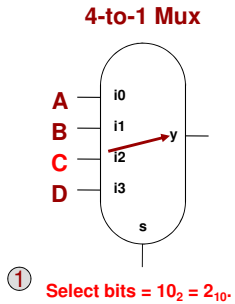
Multiplexers

- Multiplexers are one of the most common digital circuits
- Anatomy: n data inputs, $\log_2 n$ select bits, 1 output
- A multiplexer ("mux" for short) selects one data input and passes it to the output



S_1	S_0	Y
0	0	i0
0	1	i1
1	0	i2
1	1	I3

Multiplexers



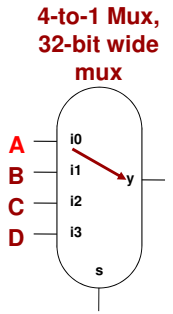
② Thus, input 2 = C is selected and passed to the output

① Select bits = $10_2 = 2_{10}$.

C

S_1	S_0	Y
0	0	i0
0	1	i1
1	0	i2
1	1	I3

Multiplexers



② Thus, input 0 = A is selected and passed to the output

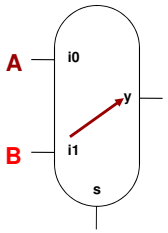
① Select bits = $00_2 = 0_{10}$.

A

S_1	S_0	Y
0	0	i0
0	1	i1
1	0	i2
1	1	I3

Multiplexers

2-to-1 Mux,
32-bit wide mux



B

S	Y
0	i0
1	I1

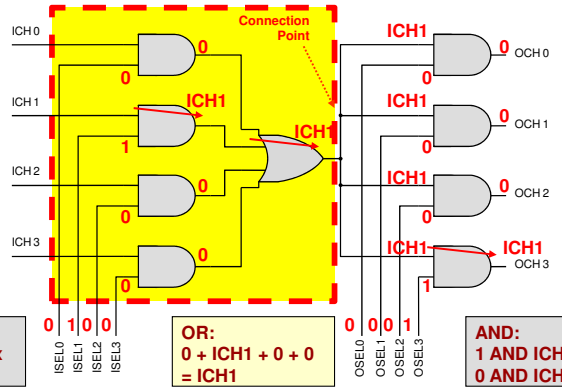
② Thus, input 1 = B is selected and passed to the output

① Select bits = $1_2 = 1_{10}$.

Recall Using T1/T2

- 1st Level of AND gates act as barriers only passing 1 channel
- OR gates combines 3 streams of 0's with the 1 channel that got passed (i.e. ICH1)
- 2nd Level of AND gates passes the channel to only the selected output

Essentially this logic forms a 4-to-1 mux where one level of gates blocks all but 1 and then the OR gate combines all signals



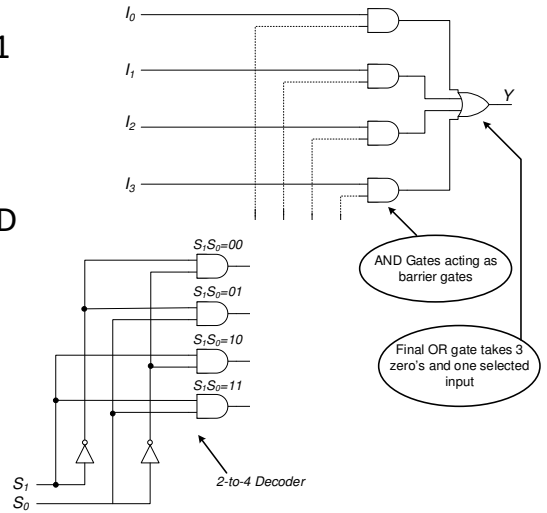
AND:
1 AND ICHx = ICHx
0 AND ICHx = 0

OR:
0 + ICH1 + 0 + 0
= ICH1

AND:
1 AND ICH1 = ICH1
0 AND ICH1 = 0

Exercise: Build a 4-to-1 mux

- Complete the 4-to-1 mux to the right by drawing wires between the 2-to-4 decode and the AND gates

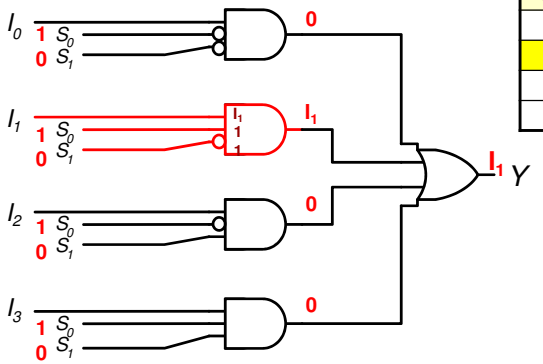


Building a Mux

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

S_1	S_0	Y
0	0	i0
0	1	i1
1	0	i2
1	1	i3

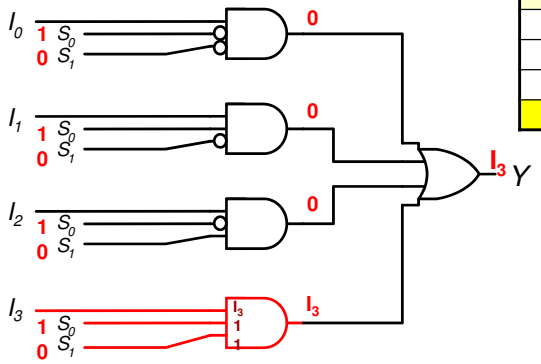
$S_1S_0 = 01_2$



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.

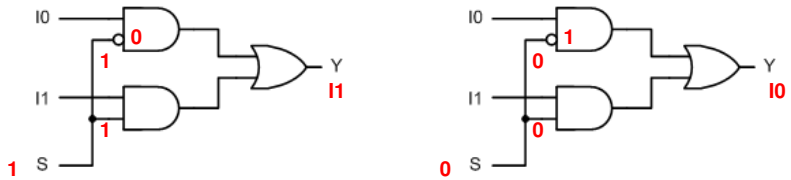
$S_1S_0 = 11_2$



S_1	S_0	Y
0	0	i0
0	1	i1
1	0	i2
1	1	i3

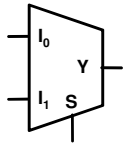
Building a Mux

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

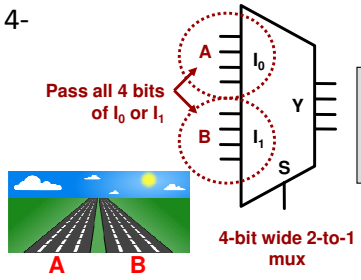


Building Wide Muxes

- 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



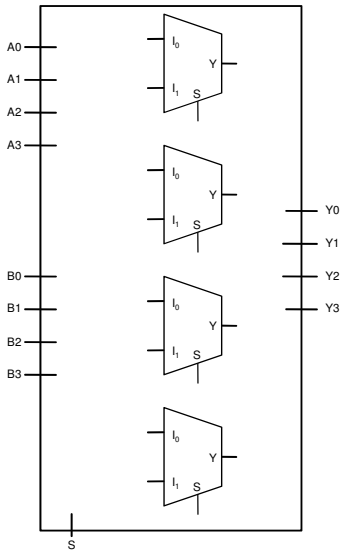
1-bit wide 2-to-1 mux



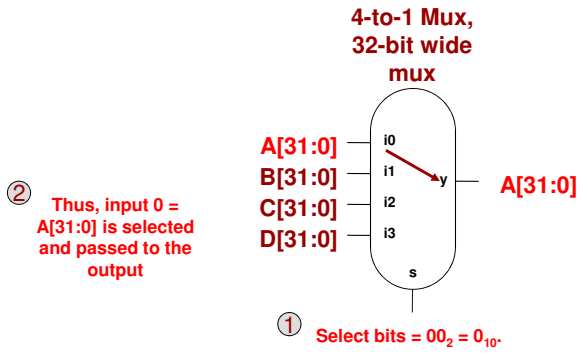
4-bit wide 2-to-1 mux

Building Wide Muxes

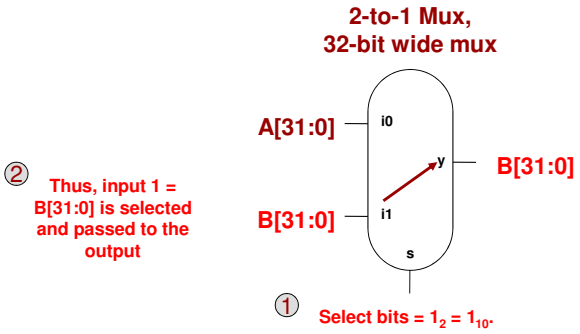
- Use one mux per _____
  - To build a 4-bit wide 2-to-1 mux, use ___ separate 2-to-1 muxes
- Operation:
  - When S=0, all muxes pass their I_0 inputs which means all the A bits get through
  - When S=1, all muxes pass their I_1 inputs which means all the B bits get through
- In general, to build an **m-bit wide (i.e. m-lane) n-to-1 mux**, use __ **individual** _____ **muxes**



Multiplexers



Multiplexers

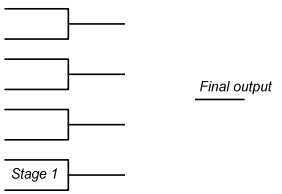


Exercise

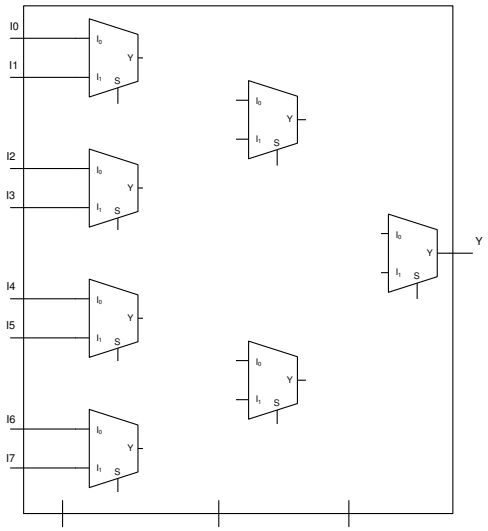
- How many 1-bit wide muxes and of what size would you need to build a **4-to-1, 8-bit** wide mux (i.e. there are 4 numbers: W[7:0], X[7:0], Y[7:0] and Z[7:0] and you must select one)
- How many 1-bit wide muxes and of what size would you need to build a **8-to-1, 2-bit** wide mux?

Building Large Muxes

- Similar to a tournament of sports teams
  - Many teams enter and then are narrowed down to 1 winner
  - In each round winners play _____



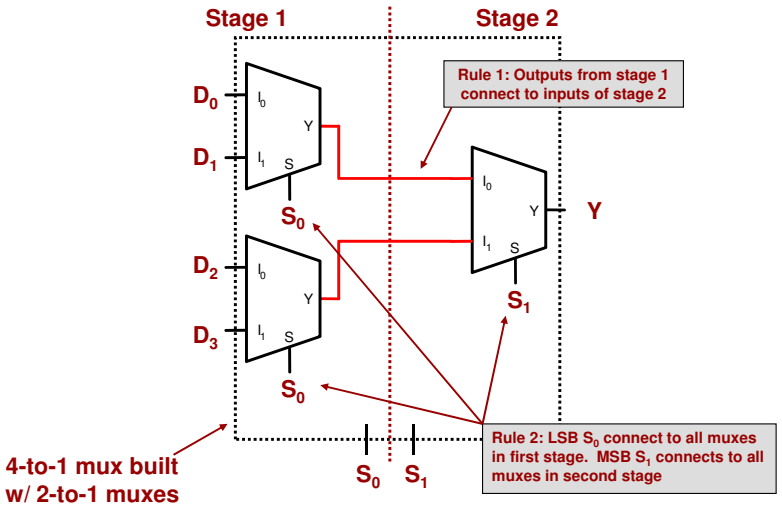
Design an 8-to-1 mux with 2-to-1 Muxes



Cascading Muxes

- Use several small muxes to build large ones
- Rules
  1. Arrange the muxes in stages (based on necessary number of inputs in 1st stage)
  2. Outputs of one stage feed to inputs of the next until only 1 final output
  3. All muxes in a stage connect to the same group of select bits
     - Usually, LSB connects to first stage
     - MSB connect to last stage

Building a 4-to-1 Mux

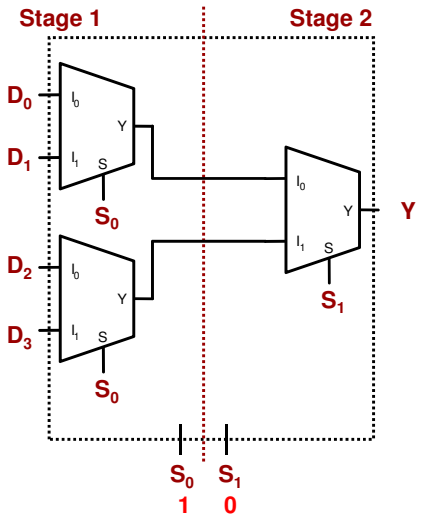


Building a 4-to-1 Mux

S_1	S_0	Y
0	0	D_0
0	1	D_1
1	0	D_2
1	1	D_3

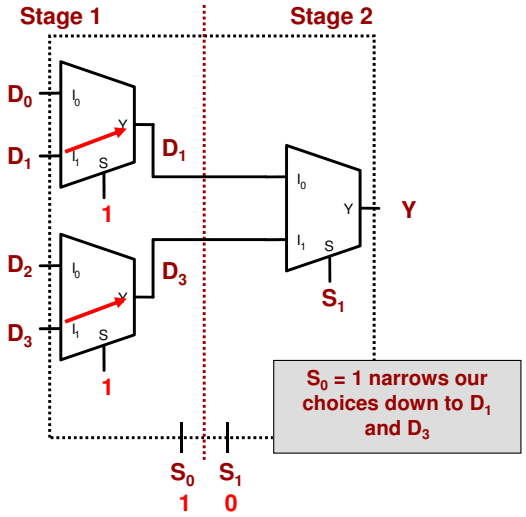
Walk through an example:

S_1S_0 = 01



Building a 4-to-1 Mux

S_1	S_0	Y
0	0	D_0
0	1	D_1
1	0	D_2
1	1	D_3



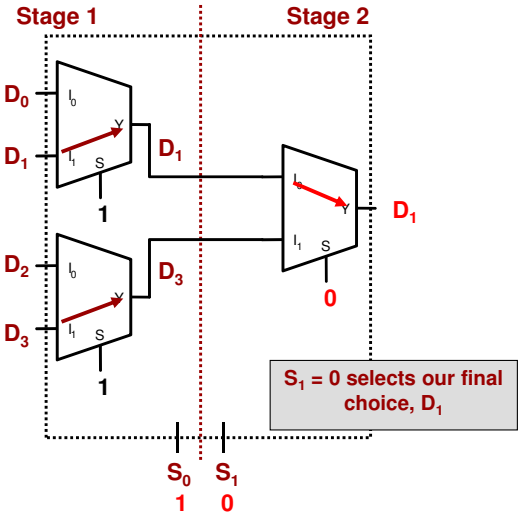
Walk through an example:

$S_1S_0 = 01$

$S_0 = 1$ narrows our choices down to D_1 and D_3

Building a 4-to-1 Mux

S_1	S_0	Y
0	0	D_0
0	1	D_1
1	0	D_2
1	1	D_3



Walk through an example:

$S_1S_0 = 01$

$S_1 = 0$ selects our final choice, D_1

Device vs. System Labels

- When using hierarchy (i.e. building blocks) to design a circuit be sure to show both device and system labels
  - Device Labels: Signal names used _____ the block
    - _____ to indicate which input/output is which to the outside user
  - System labels: Signal names used _____ the block
    - _____ signals from the circuit being built
    - Can have the same name as the device label if such a signal name exists at the outside level

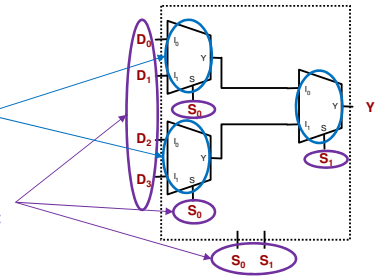
Analogy: Formal and Actual parameters in software function calls

1. a and b are like device labels and indicate the names used inside a block.
2. x and y are like system labels and represent the actual values to be used.

```
int div(int i0, int i1)
{ int t = i0/i1;
  return t;  }
int main()
{
  int d0=10, d1=2;
  int s =div(d0,d1);
}
```

Device Labels: Indicate which input/output is which inside the bock.

System Labels: Actual signals from the circuit being built



Exercise

- Sketch how you could build a 16-to-1 mux with 4-to-1 muxes? 8-to-1 and 2-to1 muxes?

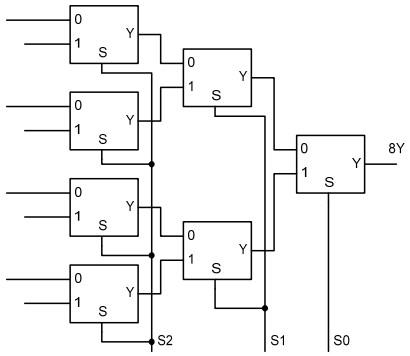
Exercise

- Create a 3-to-1 mux using 2-to-1 muxes
  - Inputs: I0, I1, I2 and select bits S1,S0
  - Output: Y

S_1	S_0	Y
0	0	D_0
0	1	D_1
1	0	D_2

Select-bit Ordering

- If we connect the select bits as shown to build an 8-to-1 mux, show how to label the inputs (i0-i7) so that the correct input is passed based on the binary value of S2:S0



Selects S_2	S_1	S_0	OUT Y
0	0	0	
1			
0	1		
1			
0	0	1	
1			
0	1		
1			