Introduction to Digital Logic

Lecture 12:
Multiplexers (“Muxes”)
Demultiplexers
Adders
Combinational Building Blocks

• Fundamental blocks that other combinational structures can be built from
  – Decoders
  – Encoders
  – Multiplexers
  – Demultiplexers
  – Adders (Multipliers)
  – Comparators
  – Shifters
SIMPLE & PRIORITY ENCODERS
Encoders

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

![Binary Encoder Diagram]

One active input

That number input gets encoded in binary

$4_{10}$
Encoders

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

• What’s inside an encoder?

<table>
<thead>
<tr>
<th>$I_0$</th>
<th>$I_1$</th>
<th>$I_2$</th>
<th>$I_3$</th>
<th>$I_4$</th>
<th>$I_5$</th>
<th>$I_6$</th>
<th>$I_7$</th>
<th>$Y_2$</th>
<th>$Y_1$</th>
<th>$Y_0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

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?

<table>
<thead>
<tr>
<th>I_0</th>
<th>I_1</th>
<th>I_2</th>
<th>I_3</th>
<th>I_4</th>
<th>I_5</th>
<th>I_6</th>
<th>I_7</th>
<th>Y_2</th>
<th>Y_1</th>
<th>Y_0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

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

Y_2 = I_4 + I_5 + I_6 + I_7
Encoders

- What’s inside an encoder?

<table>
<thead>
<tr>
<th>I_0</th>
<th>I_1</th>
<th>I_2</th>
<th>I_3</th>
<th>I_4</th>
<th>I_5</th>
<th>I_6</th>
<th>I_7</th>
<th>Y_2</th>
<th>Y_1</th>
<th>Y_0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

\[ Y_2 = I_4 + I_5 + I_6 + I_7 \]
\[ Y_1 = I_2 + I_3 + I_6 + I_7 \]
\[ Y_0 = I_1 + I_3 + I_5 + I_7 \]
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

\[
\begin{align*}
Y_2 &= I_4 + I_5 + I_6 + I_7 \\
Y_1 &= I_2 + I_3 + I_6 + I_7 \\
Y_0 &= I_1 + I_3 + I_5 + I_7
\end{align*}
\]
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)

If multiple inputs are active only the highest priority active input (I5) is encoded

Don’t worry about this input, just leave it active

/ V = Valid (a.k.a /EO=Enabled Output) is active if ANY inputs are 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

No inputs are active

Output is still 000 but /V tells us that this is not because I0 was active

/ V = Valid (a.k.a /EO=Enabled Output) is inactive if no 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
MUXES
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
Thus, $D_2$ is selected and passed to the output.

Select bits $= 10_2 = 2_{10}$. 

Thus, $D_2$ is selected

$= 2_{10}$. 

Select bits $= 10_2 = 2_{10}$. 

Thus, $D_2$ is selected and passed to the output.
Multiplexers

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

Thus, $D_0$ is selected and passed to the output.

Select bits $= 00_2 = 0_{10}$. 

© Mark Redekopp, All rights reserved
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.

\[ S_1S_0 = 01 \]
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.

\[ S_1S_0 = 11 \]
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

\[ I_0 \quad S_0 \quad S_1 \]
\[ I_1 \quad S_0 \quad S_1 \]
\[ I_2 \quad S_0 \quad S_1 \]
\[ I_3 \quad S_0 \quad S_1 \]

\[ \begin{array}{c}
   1 \\
   \text{/E} \\
\end{array} \]

\[ Y \quad 0 \]
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

![Diagram of 1-bit wide 2-to-1 mux](image)

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

![Diagram of 4-bit wide 2-to-1 mux](image)

Pass all 4 bits of \( I_0 \) or \( I_1 \)
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
BUILDING LARGE DECODERS AND MUXES
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

Stage 1

Stage 2

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

Stage 1

MSB in connects to 1\textsuperscript{st} stage

Overall Enable

Stage 2

2-to-4 decoder

4 outputs
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
D₂ or D₃...and you see that the
lower decoder in the second
stage is the one that is enabled

<table>
<thead>
<tr>
<th>E</th>
<th>X</th>
<th>Y</th>
<th>D0</th>
<th>D1</th>
<th>D2</th>
<th>D3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>x</td>
<td>x</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
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 D₀ (really D₂).

<table>
<thead>
<tr>
<th>E</th>
<th>X</th>
<th>Y</th>
<th>D₀</th>
<th>D₁</th>
<th>D₂</th>
<th>D₃</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>x</td>
<td>x</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
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
Once a decoder gets disabled all the following will be disabled.

<table>
<thead>
<tr>
<th></th>
<th>A&lt;sub&gt;2&lt;/sub&gt;</th>
<th>A&lt;sub&gt;1&lt;/sub&gt;</th>
<th>A&lt;sub&gt;0&lt;/sub&gt;</th>
<th>Active Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>None</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>D&lt;sub&gt;0&lt;/sub&gt;</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>D&lt;sub&gt;1&lt;/sub&gt;</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>D&lt;sub&gt;2&lt;/sub&gt;</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>D&lt;sub&gt;3&lt;/sub&gt;</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>D&lt;sub&gt;4&lt;/sub&gt;</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>D&lt;sub&gt;5&lt;/sub&gt;</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>D&lt;sub&gt;6&lt;/sub&gt;</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>D&lt;sub&gt;7&lt;/sub&gt;</td>
</tr>
</tbody>
</table>
Cascading Decoders

Decode the MSB...possible combos = 4-7

Outputs 0 – 3 are disable (ruled out)

<table>
<thead>
<tr>
<th>G</th>
<th>A_2</th>
<th>A_1</th>
<th>A_0</th>
<th>Active Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>None</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>D_0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>D_1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>D_2</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>D_3</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>D_4</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>D_5</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>D_6</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>D_7</td>
</tr>
</tbody>
</table>
Cascading Decoders

Decode the $A_1$ ... possible combos = 5-6

<table>
<thead>
<tr>
<th>G</th>
<th>$A_2$</th>
<th>$A_1$</th>
<th>$A_0$</th>
<th>Active Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>None</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>$D_0$</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>$D_1$</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>$D_2$</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>$D_3$</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>$D_4$</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>$D_5$</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>$D_6$</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>$D_7$</td>
</tr>
</tbody>
</table>
Cascading Decoders

Decode the LSB…combo = 5

Outputs 0 – 3 are disabled (ruled out)

<table>
<thead>
<tr>
<th>G</th>
<th>A_2</th>
<th>A_1</th>
<th>A_0</th>
<th>Active Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>None</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>D_0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>D_1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>D_2</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>D_3</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>D_4</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>D_5</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>D_6</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>D_7</td>
</tr>
</tbody>
</table>
Build a 3-to-8 Decoder

X0 —

X1 —

X2 —
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 winners
Design an 8-to-1 mux
Cascading Muxes

- Use several small muxes to build large ones
- Rules
  1. Arrange the muxes in stages (based on necessary number of inputs in 1\textsuperscript{st} stage)
  2. Outputs of 1 stage feed to inputs of the next
  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

Stage 1

Stage 2

Rule 1: Outputs from stage 1 connect to inputs of stage 2

Rule 2: LSB $S_0$ connect to all muxes in first stage. MSB $S_1$ connects to all muxes in second stage

4-to-1 mux built w/ 2-to-1 muxes
Building a 4-to-1 Mux

Walk through an example:

\[ S_1S_0 = 01 \]
Building a 4-to-1 Mux

Walk through an example:

$$S_1S_0 = 01$$
Building a 4-to-1 Mux

<table>
<thead>
<tr>
<th>$S_1$</th>
<th>$S_0$</th>
<th>$Y$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>$D_0$</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>$D_1$</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>$D_2$</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>$D_3$</td>
</tr>
</tbody>
</table>

Walk through an example:

$S_1S_0 = 01$
Select-bit Ordering

Selects OUT

<table>
<thead>
<tr>
<th>S2</th>
<th>S1</th>
<th>S0</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Mapping Algorithms to HW

• Wherever an if..then..else statement is used usually requires a mux
    • Z = A+2
  – else
    • Z = B+5
Mapping Algorithms to HW

- Wherever an if..then..else statement is used usually requires a mux
    - Z = A+2
  - else
    - Z = B+5
Mux Application: Network Router

• Network Routers direct Internet traffic
• Assume a router connects communication links from LA, Seattle, Chicago, NY
• Router must "select" a source to output to a destination
0110101100  
Seattle -> NY

1101110111  
NY -> Chicago