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
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.
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**

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>Z</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</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>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

**AND gate decoding**

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>Z</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<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>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Checkers / Decoders

• An OR gate only outputs ‘0’ for 1 combination
  – That combination can be changed by adding inverters to the inputs
  – We can think of the OR gate as “checking” or “decoding” a specific combination and outputting a ‘0’ when it matches.
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

<table>
<thead>
<tr>
<th>Instruction Type</th>
<th>6-bit OPCODE OP[5:0]</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD</td>
<td>001000</td>
</tr>
<tr>
<td>LOAD</td>
<td>100011</td>
</tr>
<tr>
<td>STORE</td>
<td>101011</td>
</tr>
<tr>
<td>BRANCH</td>
<td>000100</td>
</tr>
</tbody>
</table>

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 ALL $2^n$ input combinations

There are gates inside to implement each output

1 output for each combination of the input number
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')

Binary #6

Only that numbered output is activated
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’)

Binary #5

Only that numbered output is activated
Decoder Sizes

- A decoder with an \( n \)-bit input has \( 2^n \) outputs
  - 1 output for every combination of the \( n \)-bit input

\[
\begin{array}{c|c|c|c|c|c|c|c}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{array}
\]

- \( n \) inputs
- \( 2^n \) outputs

\[
\begin{array}{c|c|c|c|c|c|c|c}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{array}
\]

- \( n \) inputs
- \( 2^n \) outputs

- 2-to-4 Decoder
- 3-to-8 Decoder
Exercise

• Complete the design of a 2-to-4 decoder
Building Decoders

3-bit number [A2:A0]

- Checker for 000
  - O0
- Checker for 001
  - O1
- Checker for 010
  - O2
- Checker for 011
  - O3
- Checker for 100
  - O4
- Checker for 101
  - O5
- Checker for 110
  - O6
- Checker for 111
  - O7

A0
A1
A2

O0
O1
O2
O3
O4
O5
O6
O7
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.

2-to-4 Decoder

One output will always be active.

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

When \( E = 0 \), inputs is ignored

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

Since \( E = 0 \), all outputs = 0

Since \( E = 1 \), outputs will function normally
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.
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
Multiplexers

1. Select bits = 10₂ = 2₁₀.

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

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

Select bits = 00₂ = 0₁₀.

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

A
B
C
D

i0
i1
i2
i3

Output: y
Multiplexers

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

Select bits = 1₂ = 1₁₀.
Recall Using T1/T2

- 1\textsuperscript{st} 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)
- 2\textsuperscript{nd} 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 ICH\textsubscript{x} = ICH\textsubscript{x}
0 AND ICH\textsubscript{x} = 0

**OR:**
0 + ICH\textsubscript{1} + 0 + 0 = ICH\textsubscript{1}

**AND:**
1 AND ICH\textsubscript{1} = ICH\textsubscript{1}
0 AND ICH\textsubscript{1} = 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.

\[ S_1S_0 = 01_2 \]
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.

\[ S_1 S_0 = 11_2 \]
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.
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

\[
\begin{align*}
I_0 & \quad \rightarrow \quad Y \\
I_1 & \quad \rightarrow \quad Y \\
S & \quad \rightarrow \quad Y
\end{align*}
\]

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

• Use one mux per "lane" (bit)
  – To build a 4-bit wide 2-to-1 mux, use 4 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 $\textbf{m-bit wide (i.e. m-lane) n-to-1 mux}$, use $\textbf{m individual n-to-1 muxes}$
Thus, input 0 = A[31:0] is selected and passed to the output.

Select bits = 00₂ = 0₁₀.
Multiplexers

Thus, input 1 = B[31:0] is selected and passed to the output

Select bits = 1_2 = 1_{10}.
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 winners
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

Stage 1

D0

D1

D2

D3

Stage 2

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

Rule 2: LSB S0 connect to all muxes in first stage. MSB S1 connects to all muxes in second stage

4-to-1 mux built w/ 2-to-1 muxes
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:

$S_1S_0 = 01$
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:

$S_1S_0 = 01$

$S_0 = 1$ narrows our choices down to $D_1$ and $D_3$
Building a 4-to-1 Mux

<table>
<thead>
<tr>
<th>S₁</th>
<th>S₀</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>D₀</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>D₁</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>D₂</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>D₃</td>
</tr>
</tbody>
</table>

Walk through an example:

\[ S₁S₀ = 01 \]

S₁ = 0 selects our final choice, D₁
Exercise

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

<table>
<thead>
<tr>
<th>S1</th>
<th>S0</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>D0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>D1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>D2</td>
</tr>
</tbody>
</table>