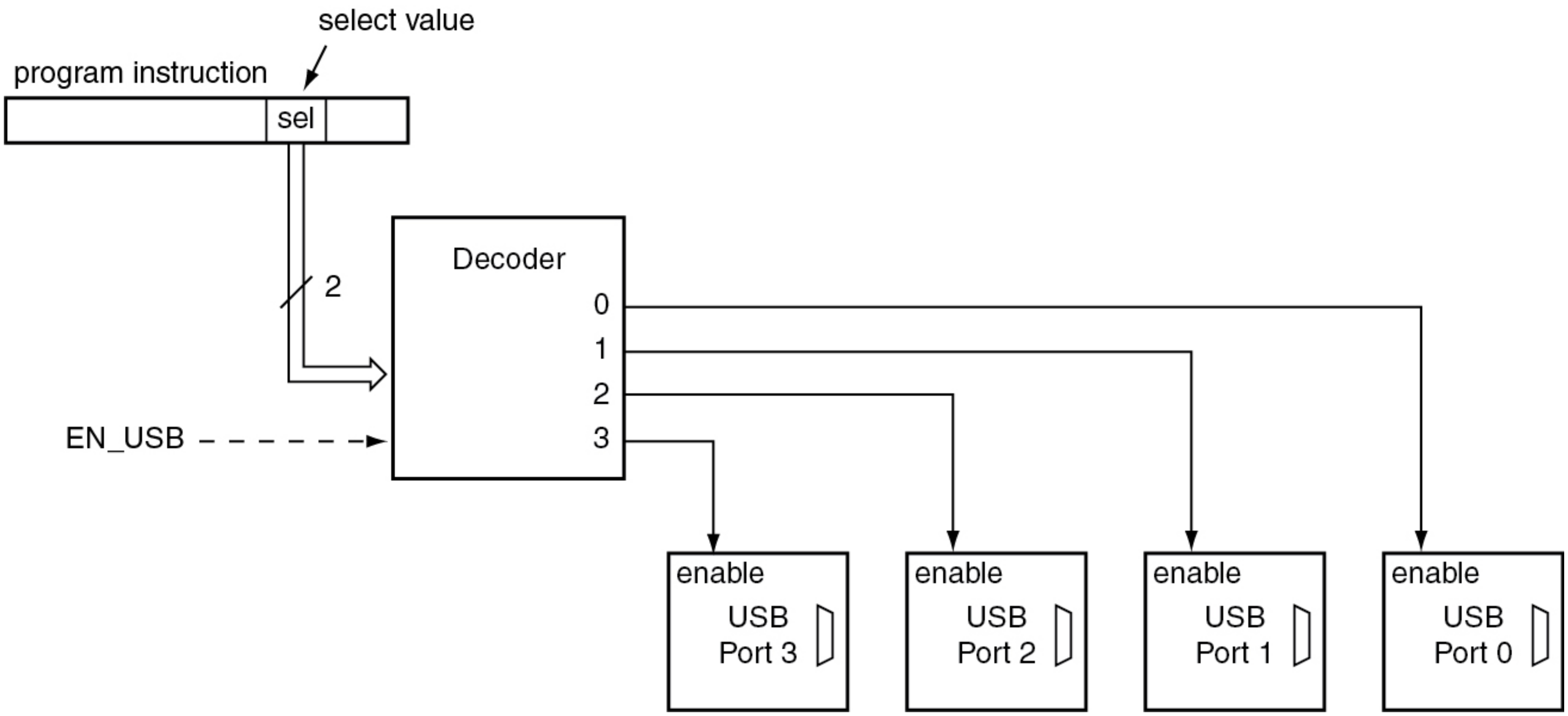
EECE 2322: Fundamentals of Digital Design and Computer Organization

Lecture 4_1: Decoder, Encoder

Xiaolin Xu
Department of ECE
Northeastern University

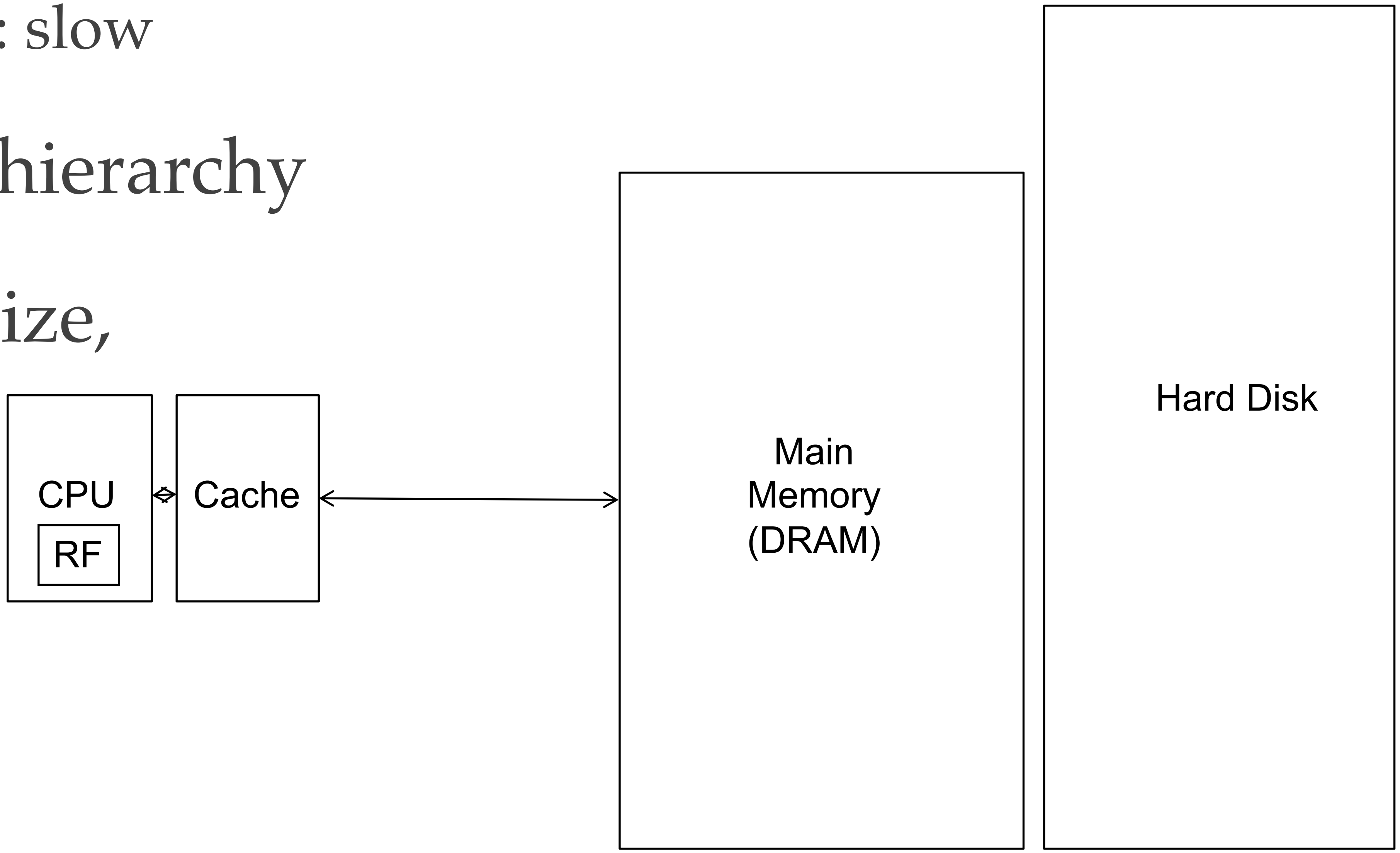
Typical Decoder Application (1) in a Computer

- Many applications might require a combinational circuit to activate one or more other circuits, elements.



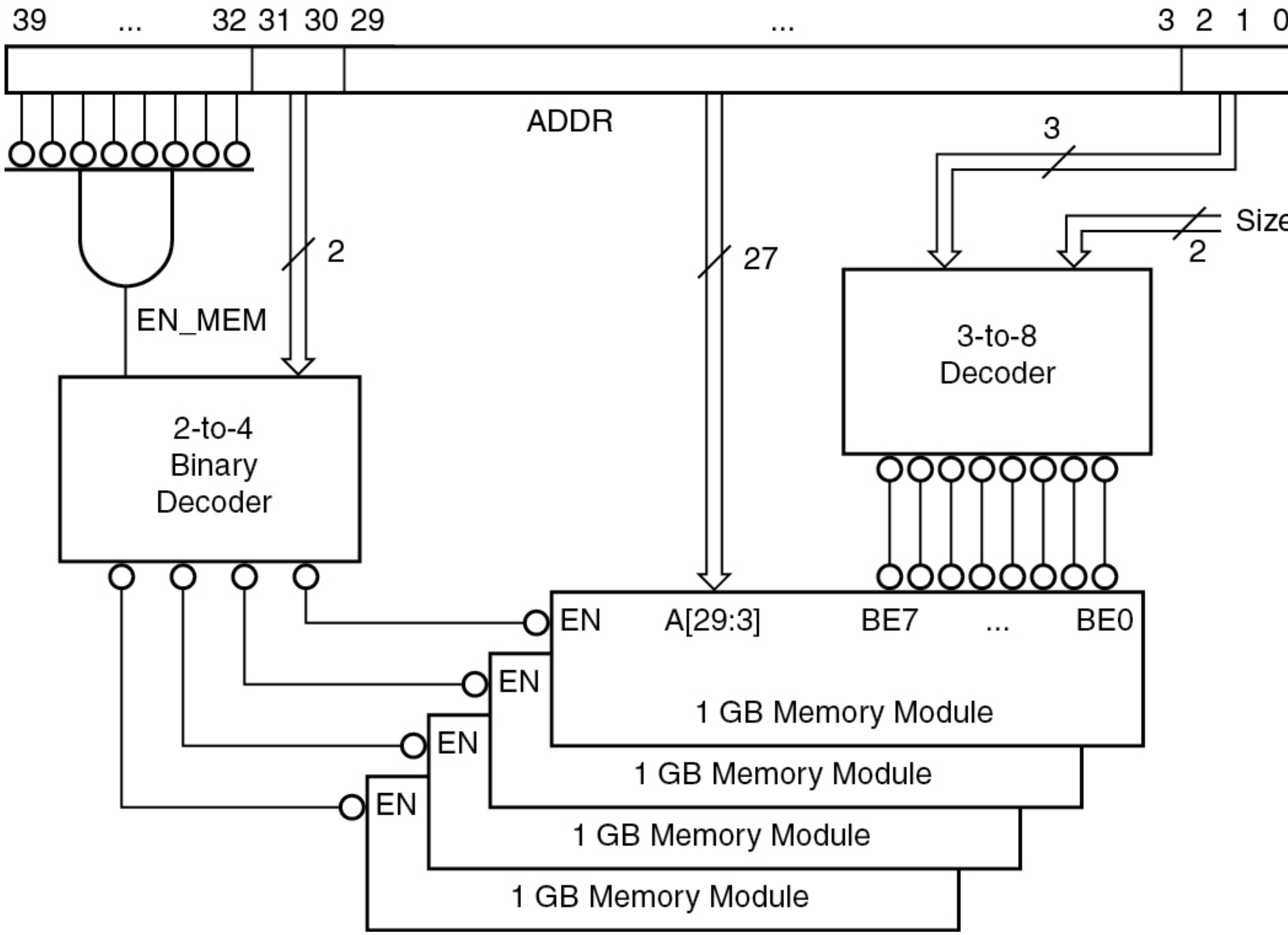
Typical Decoder Application (2) in a Computer Memory Hierarchy

- Fundamental tradeoff
  - Fast memory: small
  - Large memory: slow
- Idea: Memory hierarchy
- Latency, cost, size,
- bandwidth



Memory-Module Decoding in a Computer System

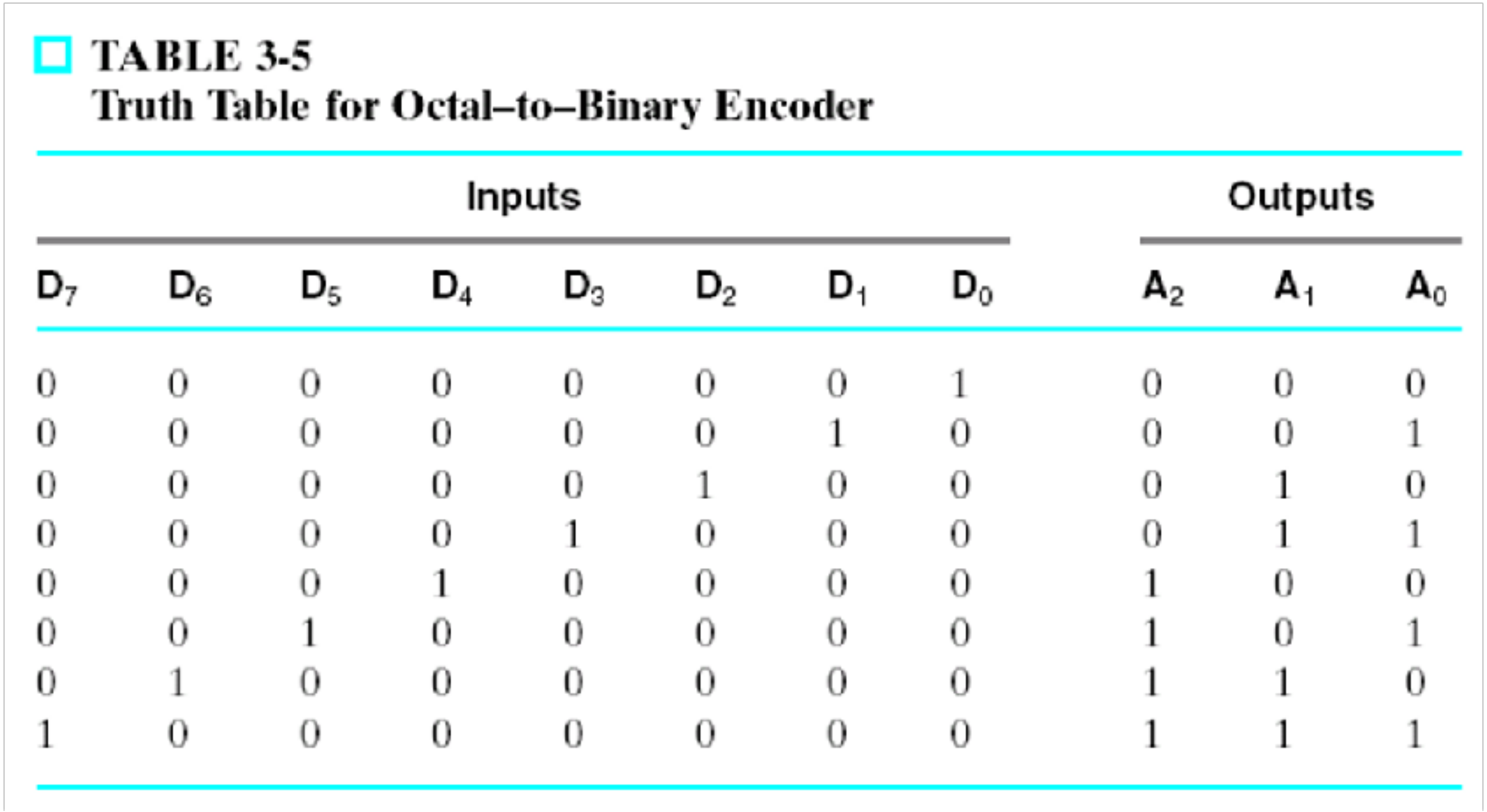
- 40-bit physical address
- 8 high-order bits=0
- 2-4 decoder
- 2^27 -> longword(8B)
- 3-8 -> 1 Byte



Encoder

- 2^n (or fewer) input lines and n output lines.
- The output lines generate the binary code corresponding to the input value, assuming only one input is high
- An encoder is the reverse function of a decoder

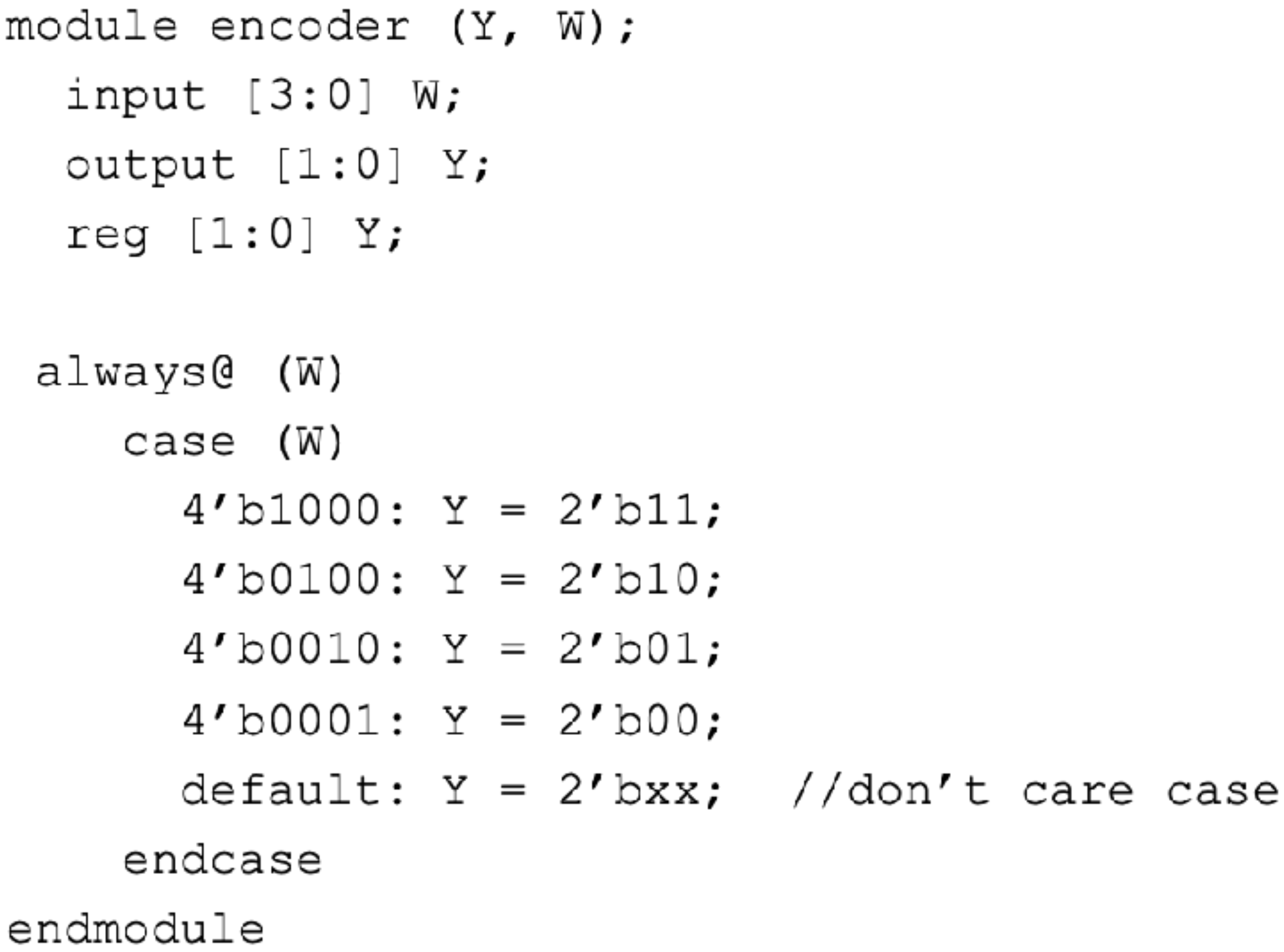
Encoder Example



□ **TABLE 3-5**
Truth Table for Octal–to–Binary Encoder

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

4-to-2 Binary Encoder in Verilog



```
module encoder (Y, W);
  input [3:0] W;
  output [1:0] Y;
  reg [1:0] Y;

 always@ (W)
    case (W)
      4’b1000: Y = 2’b11;
      4’b0100: Y = 2’b10;
      4’b0010: Y = 2’b01;
      4’b0001: Y = 2’b00;
      default: Y = 2’bxx;  //don’t care case
    endcase
endmodule
```

Encoder Output Ambiguity

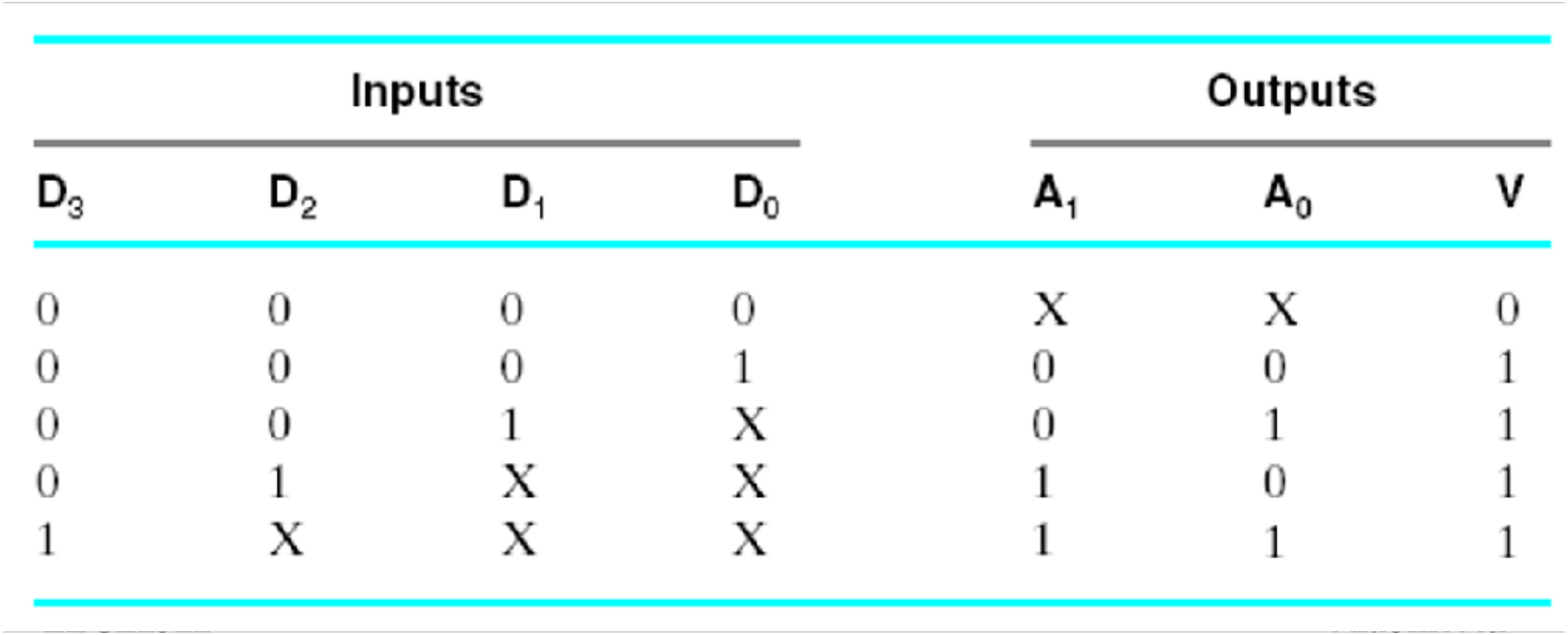
- If two inputs are active simultaneously, the output produces an incorrect combination
    - To resolve, some encoders use an input priority to ensure that only one input is encoded.

Priority Encoder

- If two or more inputs are equal to 1 at the same time, the input with highest priority takes precedence.
- Truth Table of 4-Input Priority Encoder
-

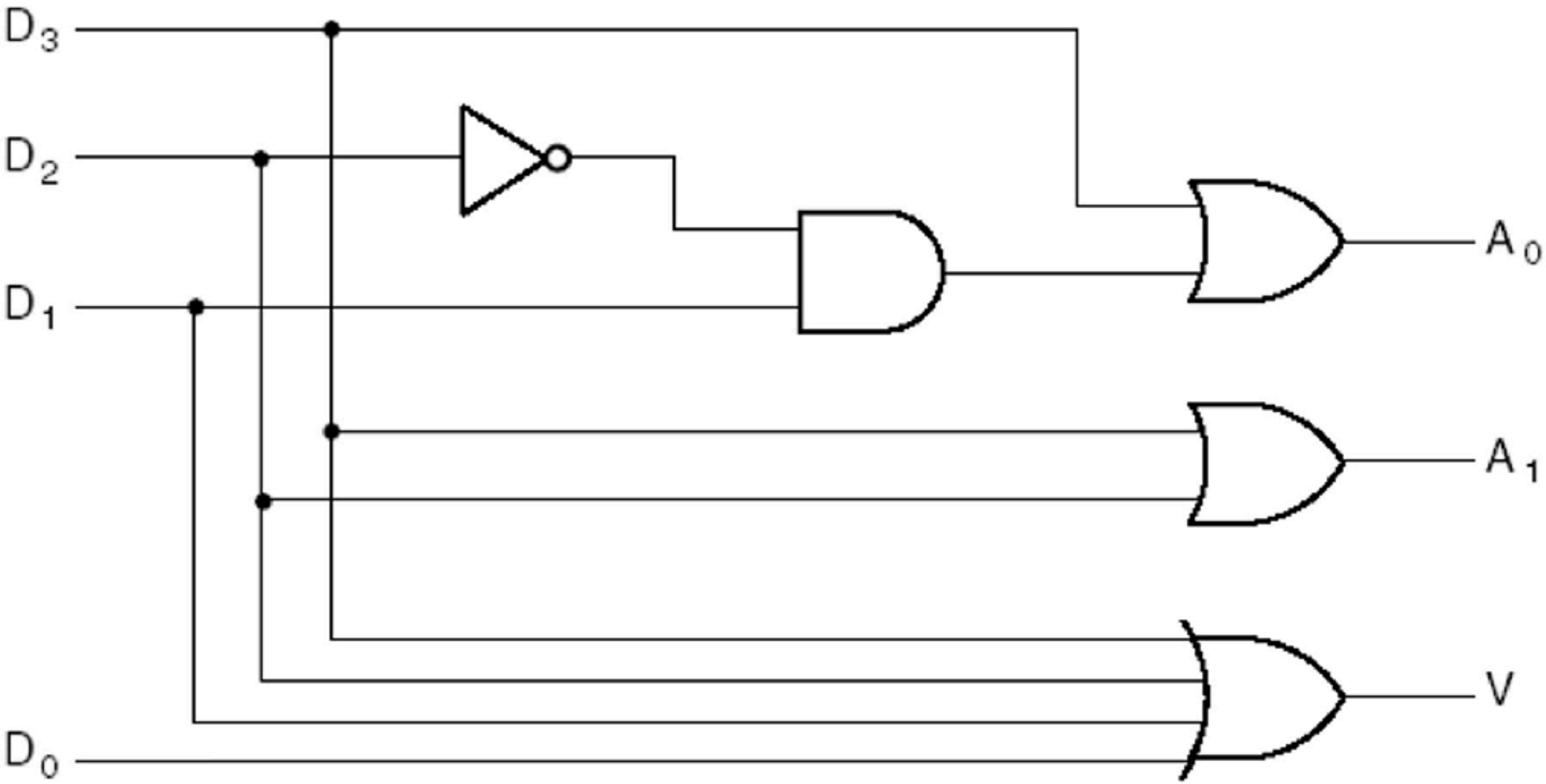
Priority Encoder

- If two or more inputs are equal to 1 at the same time, the input with highest priority takes precedence.
- Truth Table of 4-Input Priority Encoder

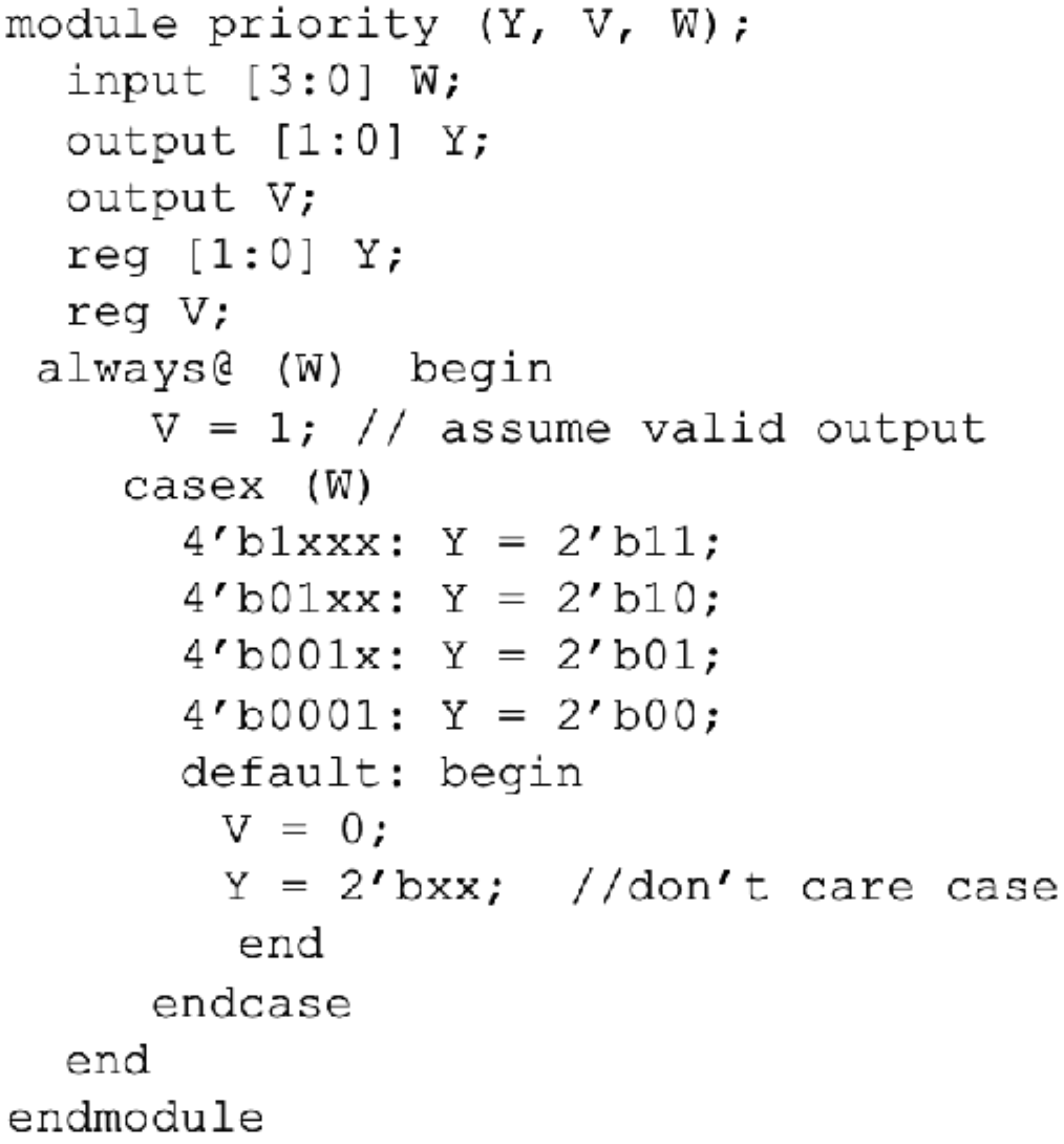


Inputs				Outputs		
D_3	D_2	D_1	D_0	A_1	A_0	V
0	0	0	0	X	X	0
0	0	0	1	0	0	1
0	0	1	X	0	1	1
0	1	X	X	1	0	1
1	X	X	X	1	1	1

Priority Encoder: Circuit Implementation



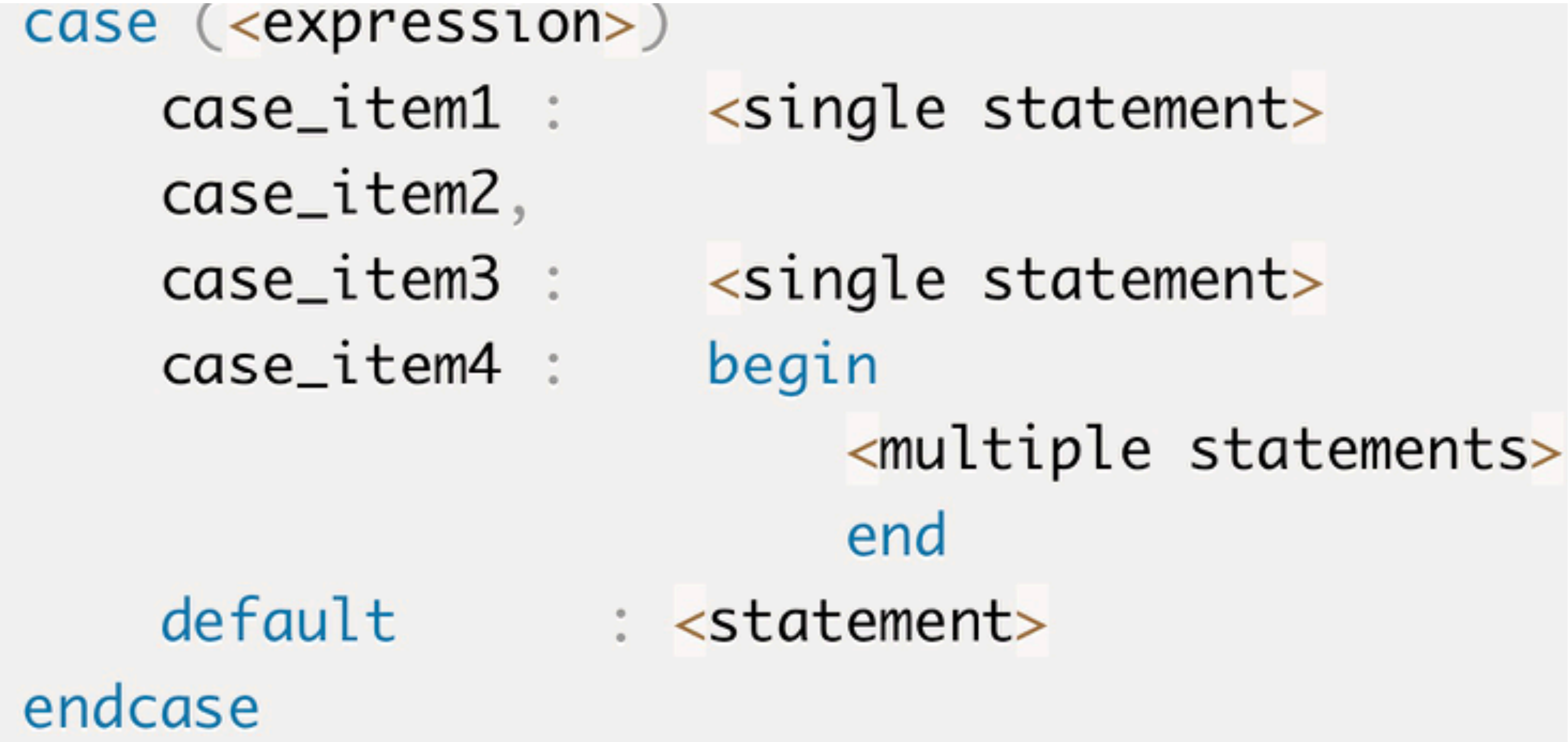
4-to-2 Priority Encoder in Verilog



```
module priority (Y, V, W);
  input [3:0] W;
  output [1:0] Y;
  output V;
  reg [1:0] Y;
  reg V;
 always@ (W)  begin
     V = 1; // assume valid output
    casex (W)
      4’b1xxx: Y = 2’b11;
      4’b01xx: Y = 2’b10;
      4’b001x: Y = 2’b01;
      4’b0001: Y = 2’b00;
      default: begin
        V = 0;
        Y = 2’bxx;  //don’t care case
        end
     endcase
  end
endmodule
```

case in Verilog

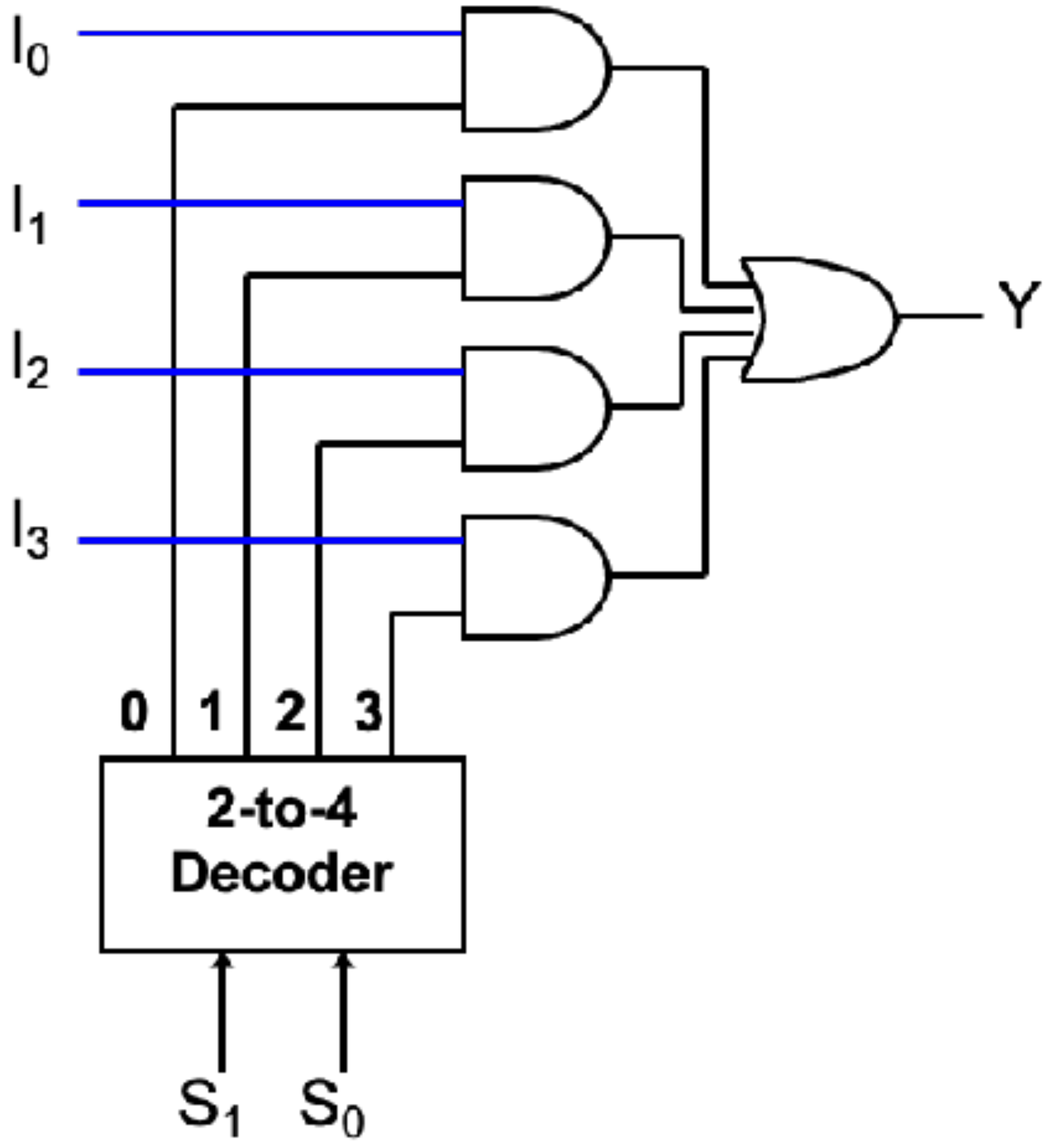
- Checks if the given expression matches one of the other expressions in the list and branches accordingly
- **Mostly used in building MUXes (see why later)**



```
case (<expression>)
    case_item1 :    <single statement>
    case_item2,
    case_item3 :    <single statement>
    case_item4 :    begin
                        <multiple statements>
                    end
    default       : <statement>
endcase
```

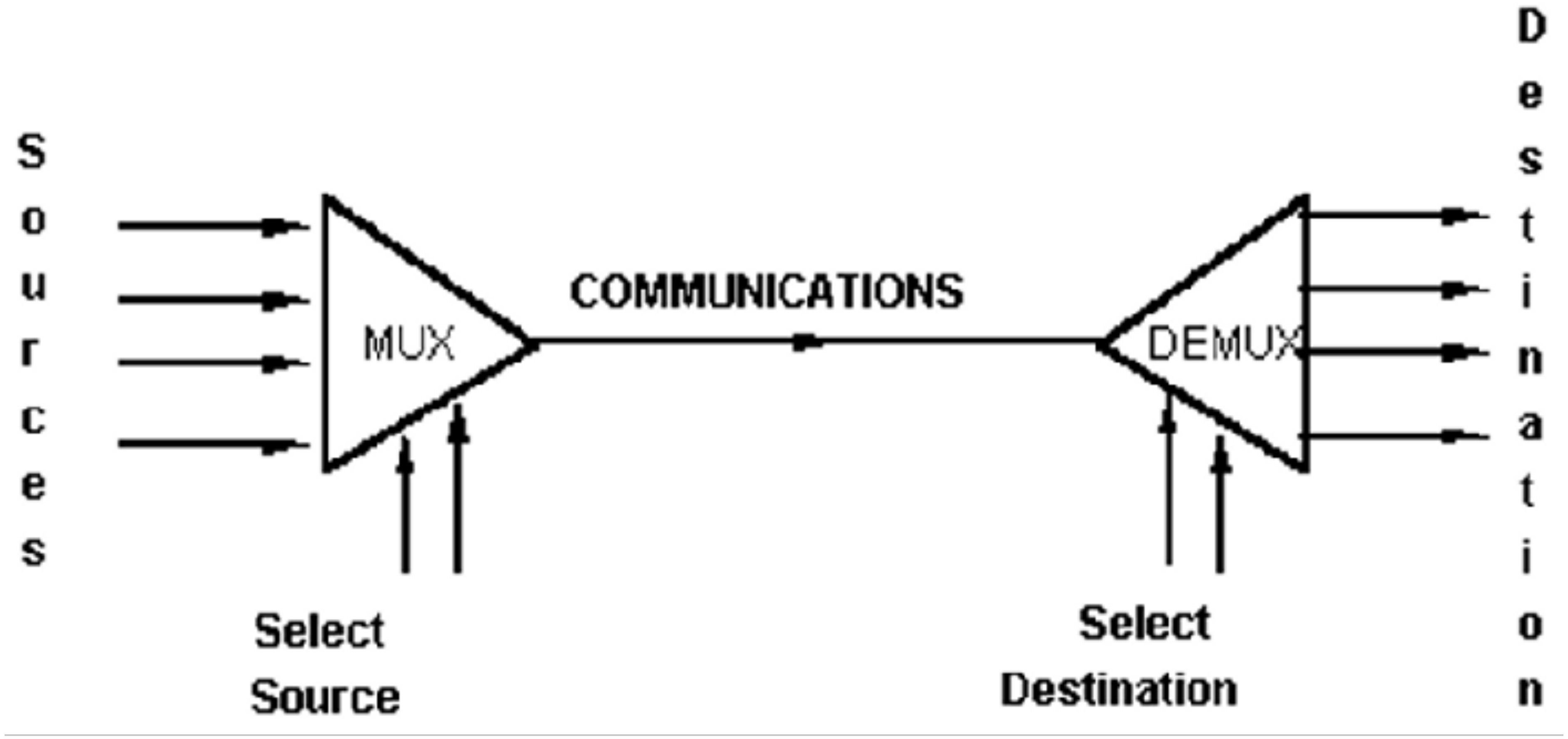
Circuit of a MUX

- 4-1 MUX has a decoder inside



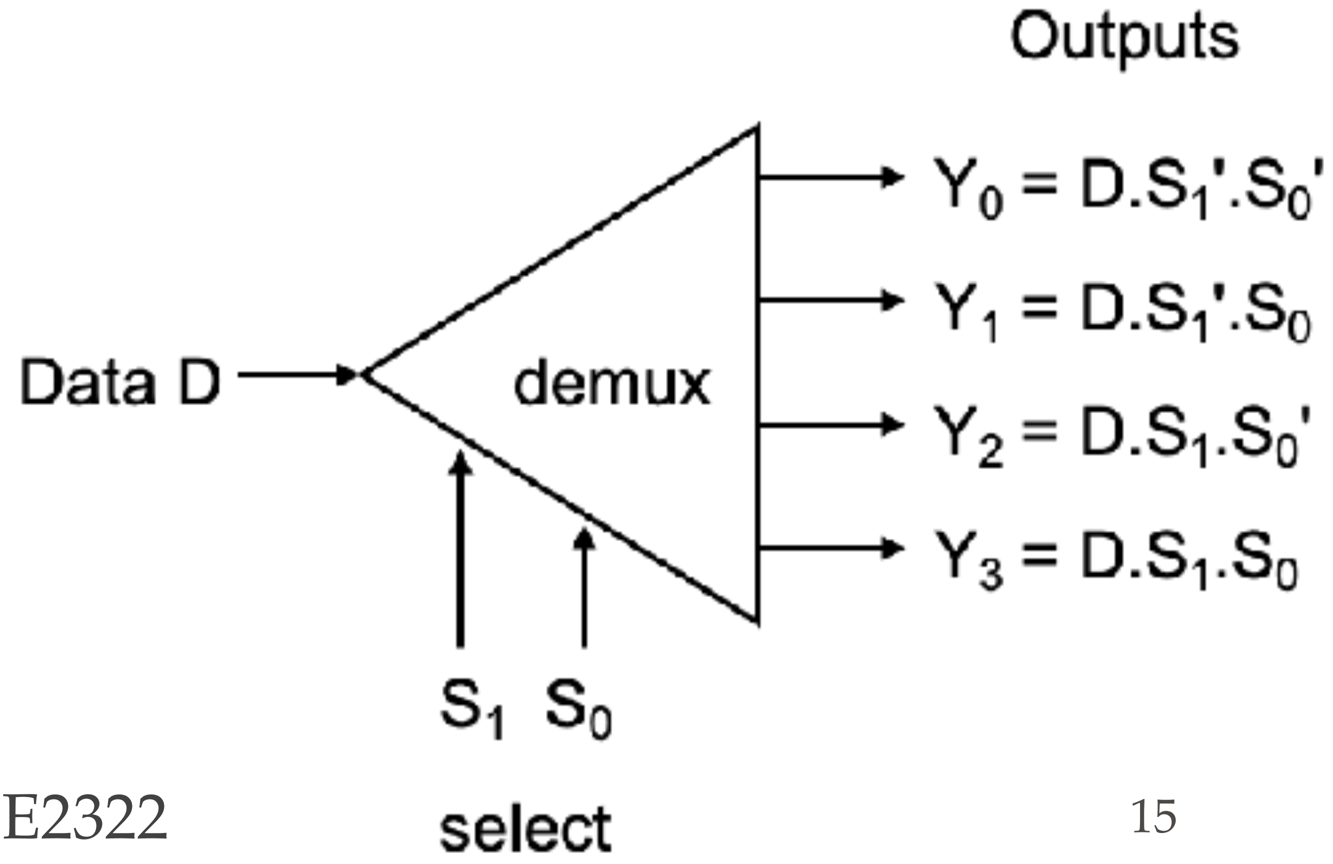
Real-world Applications of MUX

- Sharing a single communication line among a number of devices. ***At any time, only one source and one destination can use the communication line***



Demultiplexer

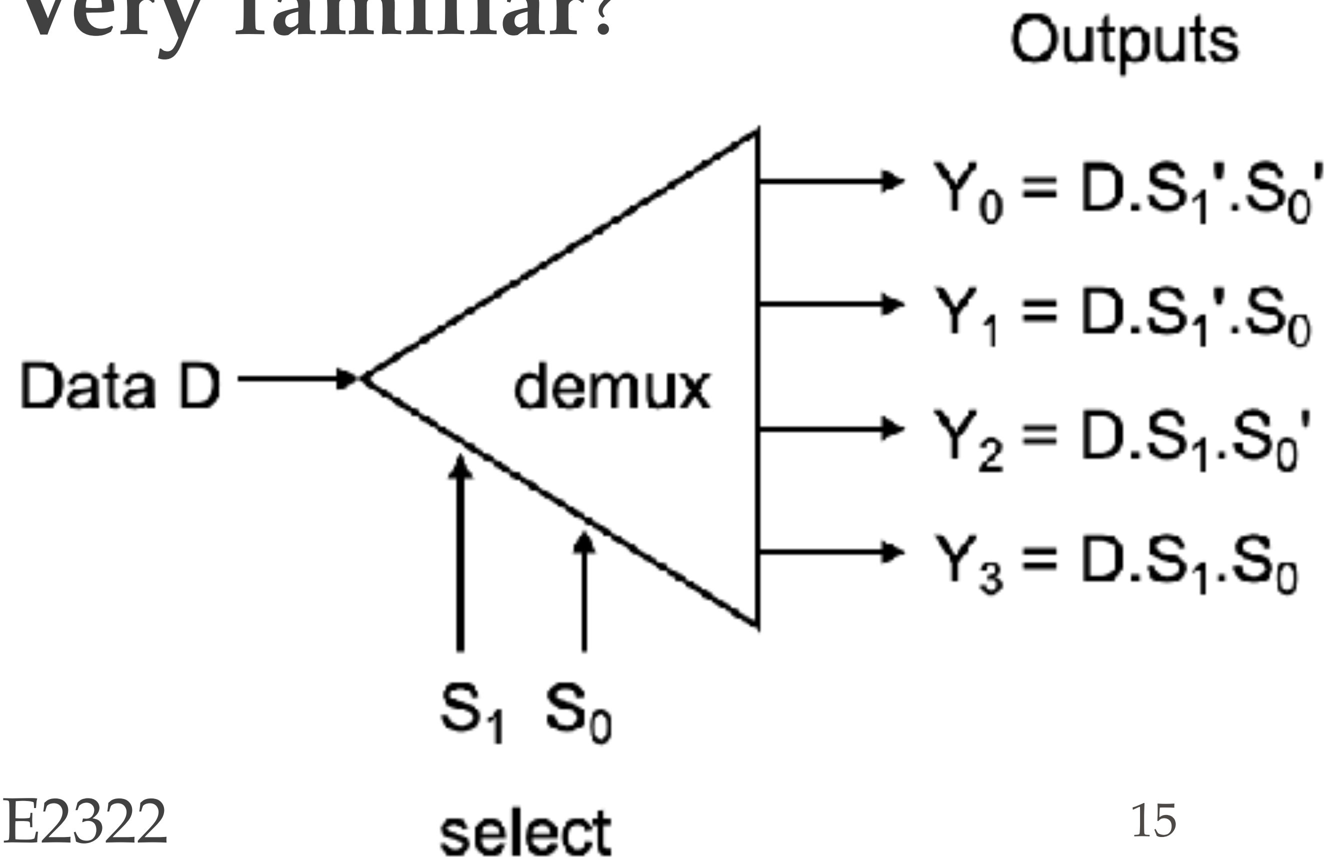
- Given an input line & a set of selection lines, the demultiplexer will direct data from input to a selected output line.
- An example of a 1-to-4 demultiplexer:



S_1	S_0	Y_0	Y_1	Y_2	Y_3
0	0	D	0	0	0
0	1	0	D	0	0
1	0	0	0	D	0
1	1	0	0	0	D

Demultiplexer

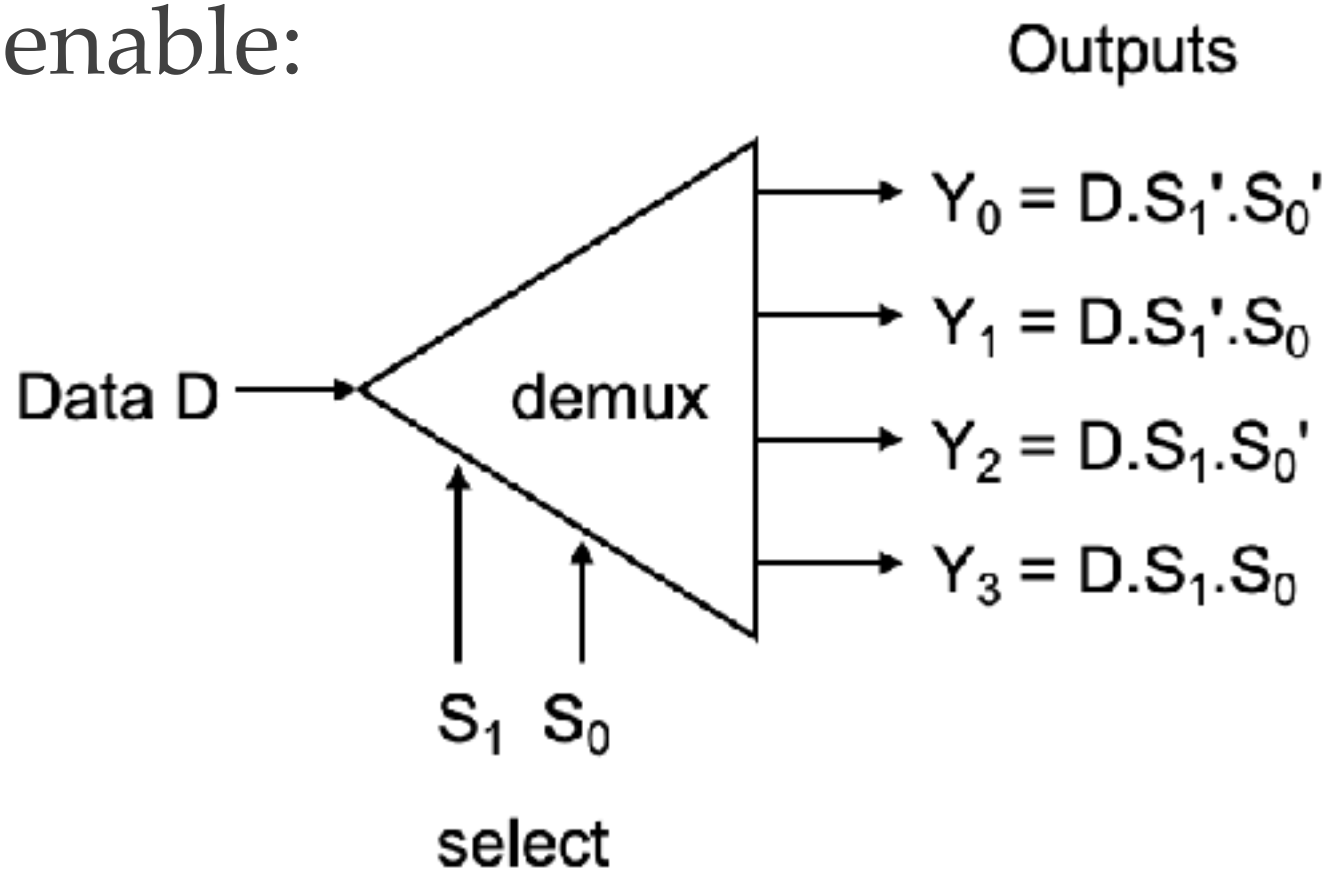
- Given an input line & a set of selection lines, the demultiplexer will direct data from input to a selected output line.
- An example of a 1-to-4 demultiplexer:
- **Very familiar**?



S_1	S_0	Y_0	Y_1	Y_2	Y_3
0	0	D	0	0	0
0	1	0	D	0	0
1	0	0	0	D	0
1	1	0	0	0	D

DeMUX

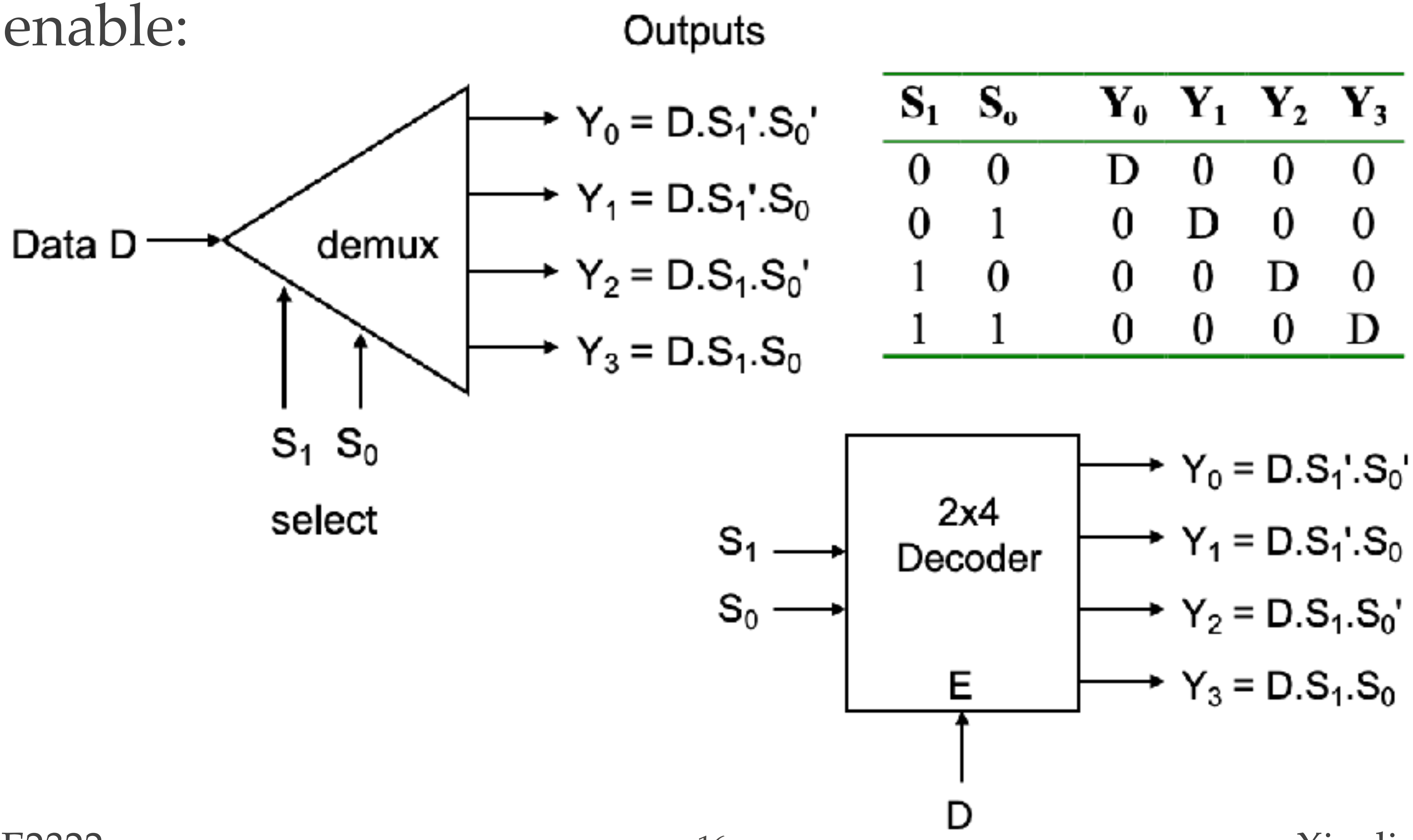
- The demultiplexer is actually identical to a decoder with enable:
-



S_1	S_0	Y_0	Y_1	Y_2	Y_3
0	0	D	0	0	0
0	1	0	D	0	0
1	0	0	0	D	0
1	1	0	0	0	D

DeMUX

- The demultiplexer is actually identical to a decoder with enable:
-

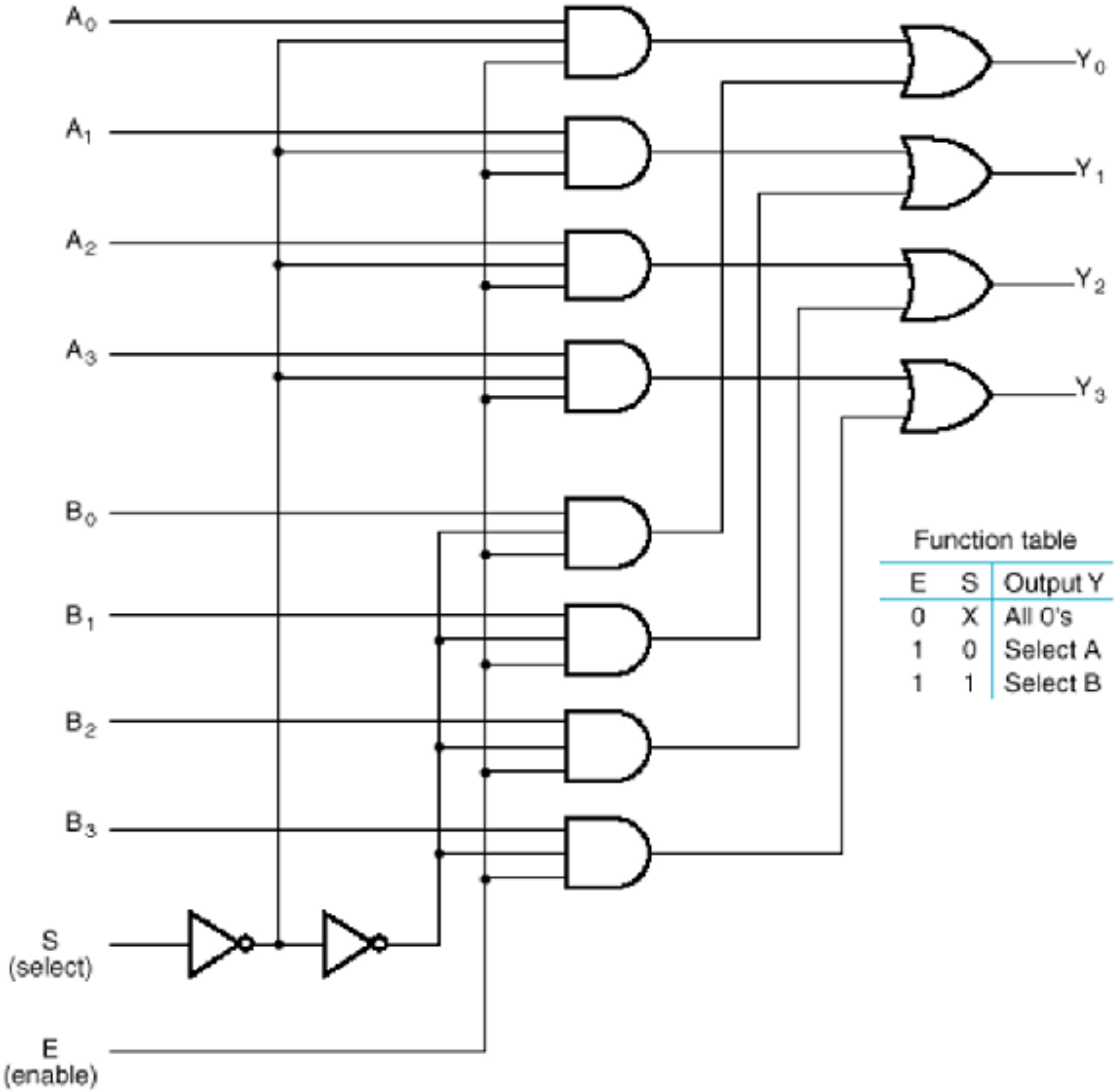


S_1	S_0	Y_0	Y_1	Y_2	Y_3
0	0	D	0	0	0
0	1	0	D	0	0
1	0	0	0	D	0
1	1	0	0	0	D

Quadruple 2–to–1-Line Multiplexer

- Notice enable bit
- Notice select bit
- 4 bit inputs

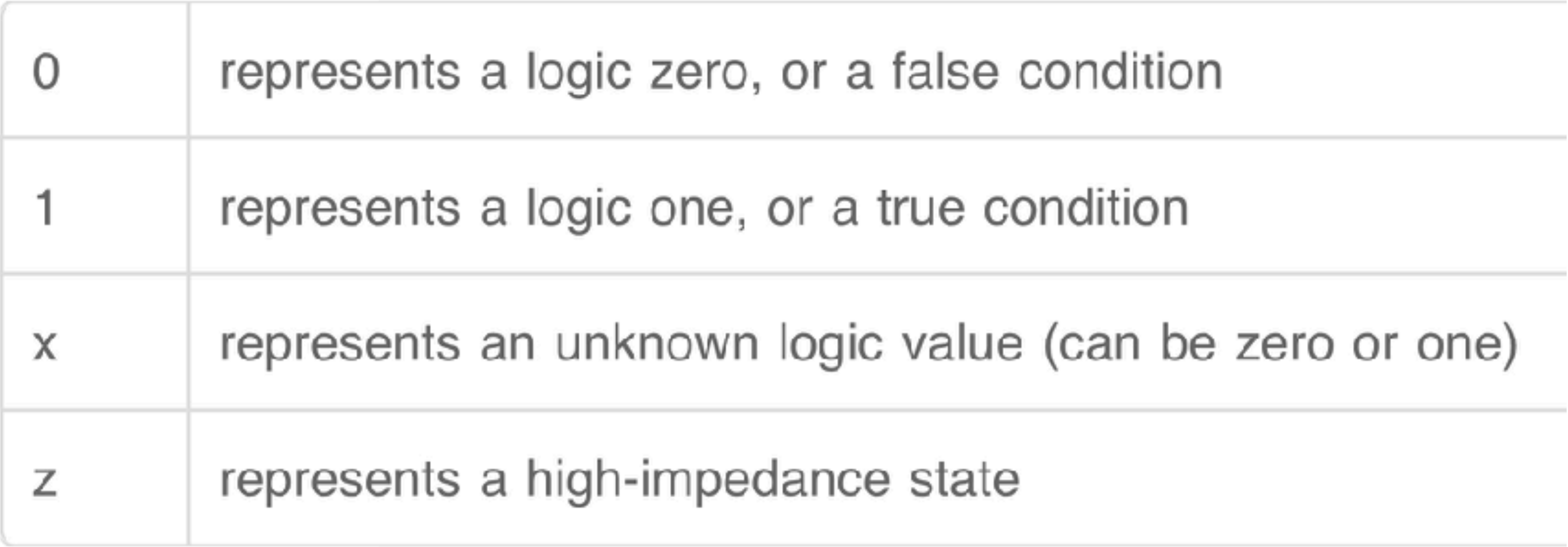
- **Data type X**



Function table

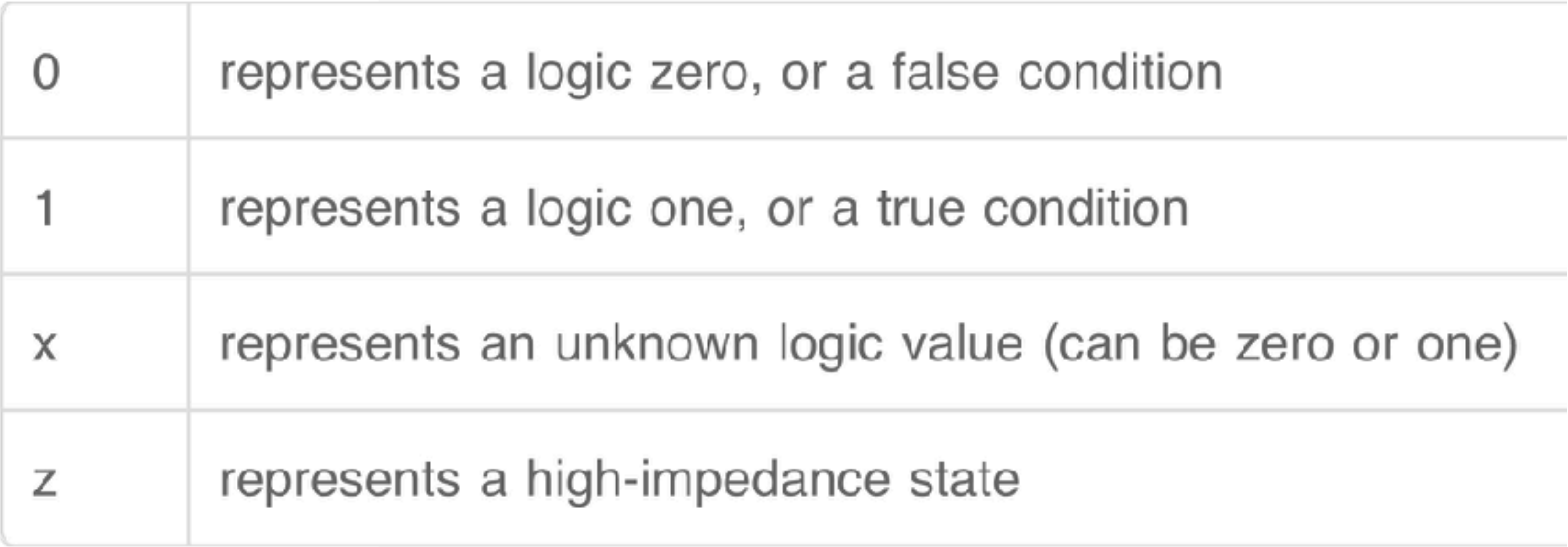
E	S	Output Y
0	X	All 0's
1	0	Select A
1	1	Select B

Verilog Data Types

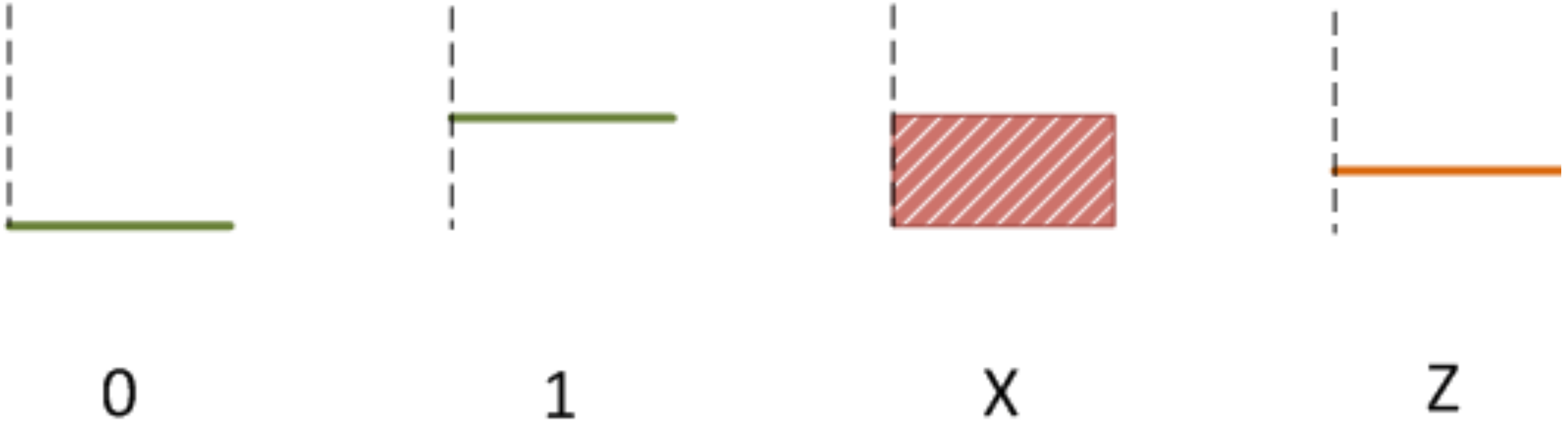


0	represents a logic zero, or a false condition
1	represents a logic one, or a true condition
x	represents an unknown logic value (can be zero or one)
z	represents a high-impedance state

Verilog Data Types



0	represents a logic zero, or a false condition
1	represents a logic one, or a true condition
x	represents an unknown logic value (can be zero or one)
z	represents a high-impedance state

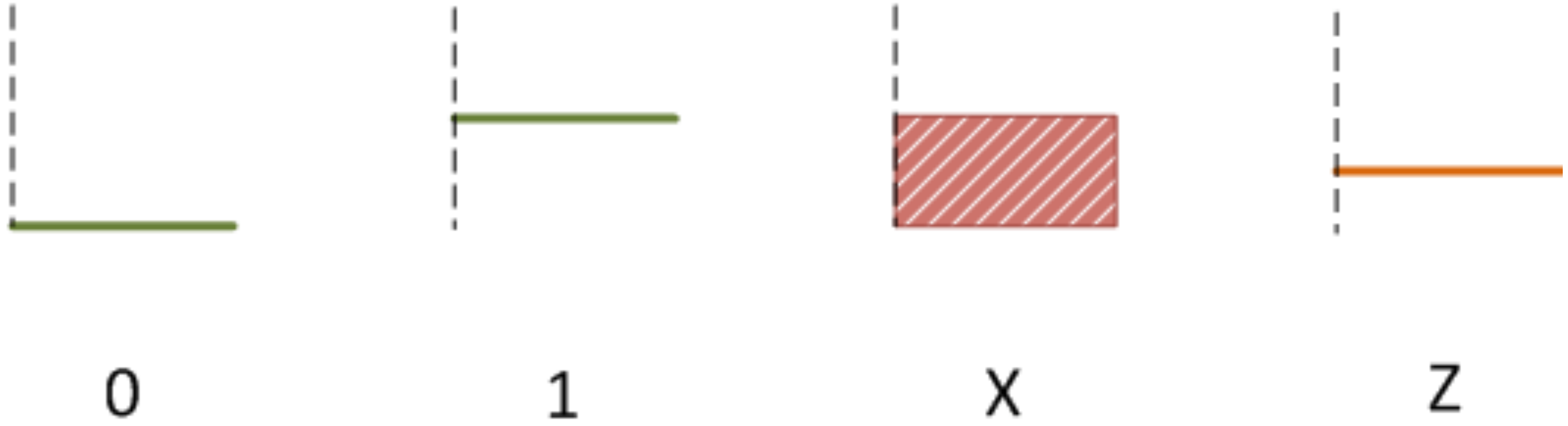


Verilog Data Types

- X means that the value is simply unknown at the time, and could be either 0 or 1
  - *Different from the way X is treated in boolean logic, where it means "don't care"*

Verilog Data Types

- X means that the value is simply unknown at the time, and could be either 0 or 1
    - *Different from the way X is treated in boolean logic, where it means "don't care"*

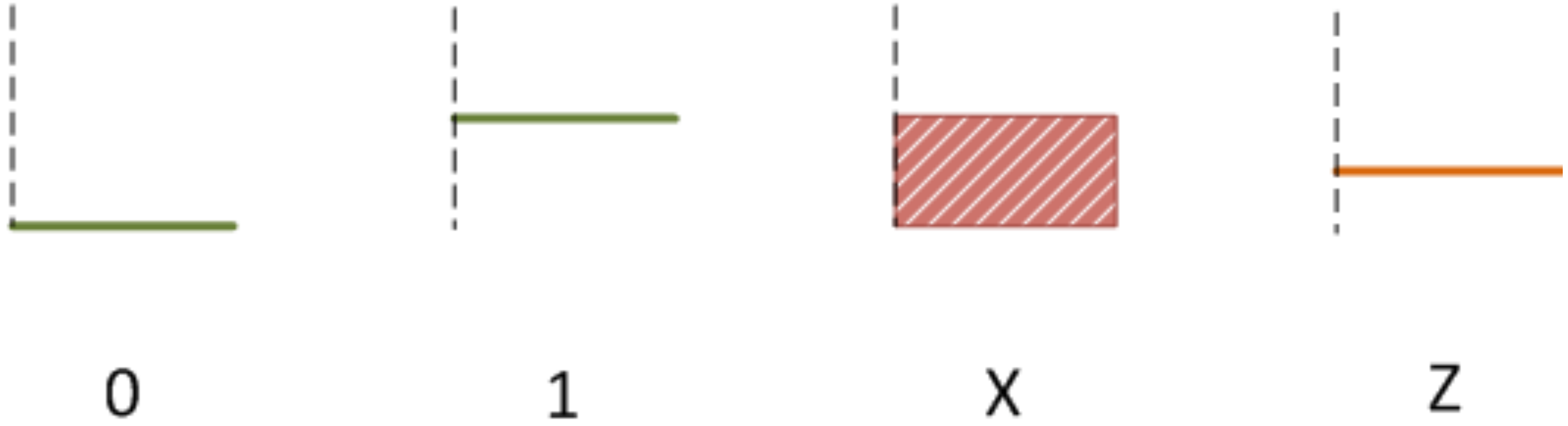


Verilog Data Types

- In any incomplete electric circuit, the wire not connected to anything will have a high-impedance at that node and is represented by Z or z.
  - *Even in verilog, any unconnected wire will result in a high impedance.*

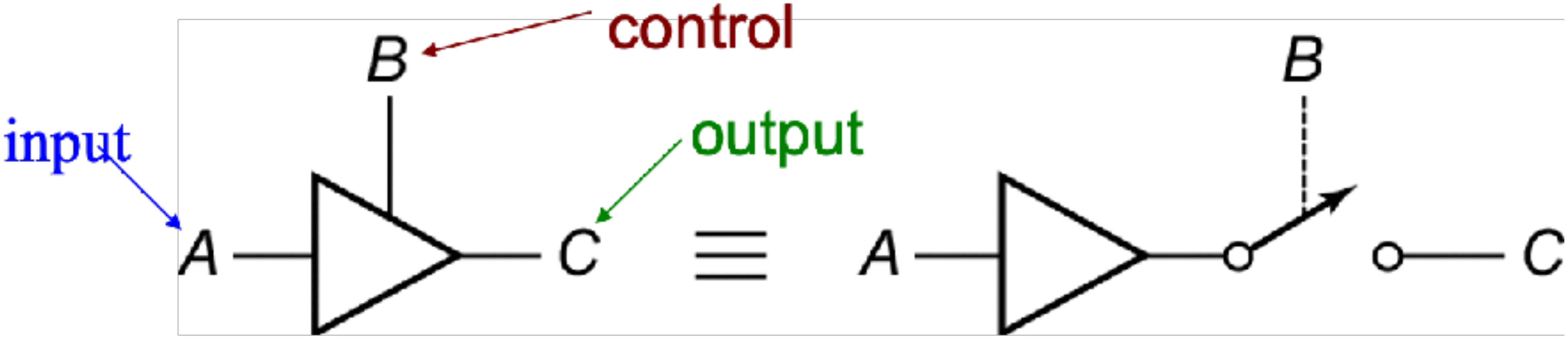
Verilog Data Types

- In any incomplete electric circuit, the wire not connected to anything will have a high-impedance at that node and is represented by Z or z.
  - *Even in verilog, any unconnected wire will result in a high impedance.*

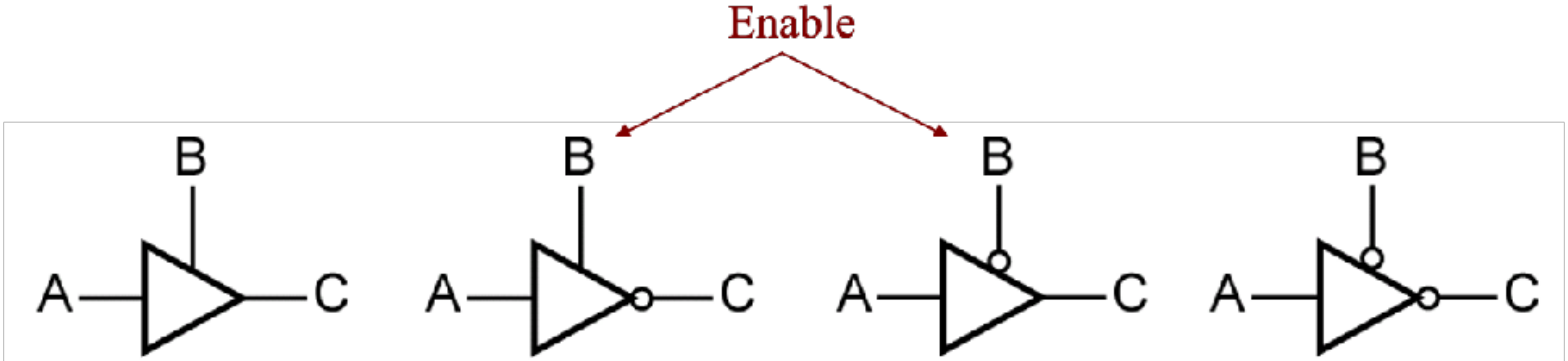


These Data Types in Practical Circuits

- **Tristate Buffer**
  - A tristate buffer can output 3 different values:
  - Logic 1 (high)
  - Logic 0 (low)
  - High-Impedance (Z)



Tristate Buffer



B	A	C
0	0	Z
0	1	Z
1	0	0
1	1	1

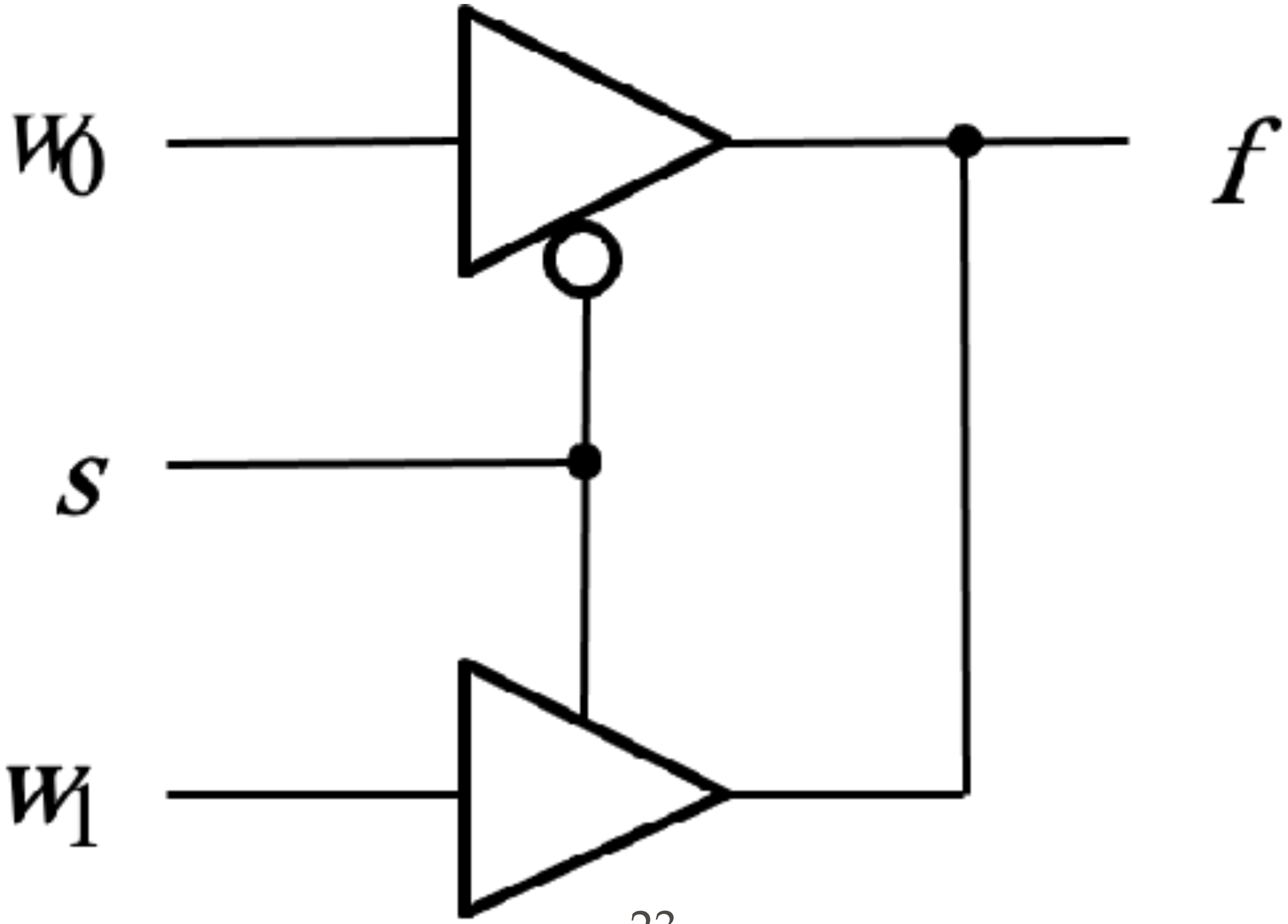
B	A	C
0	0	Z
0	1	Z
1	0	1
1	1	0

B	A	C
0	0	0
0	1	1
1	0	Z
1	1	Z

B	A	C
0	0	1
0	1	0
1	0	Z
1	1	Z

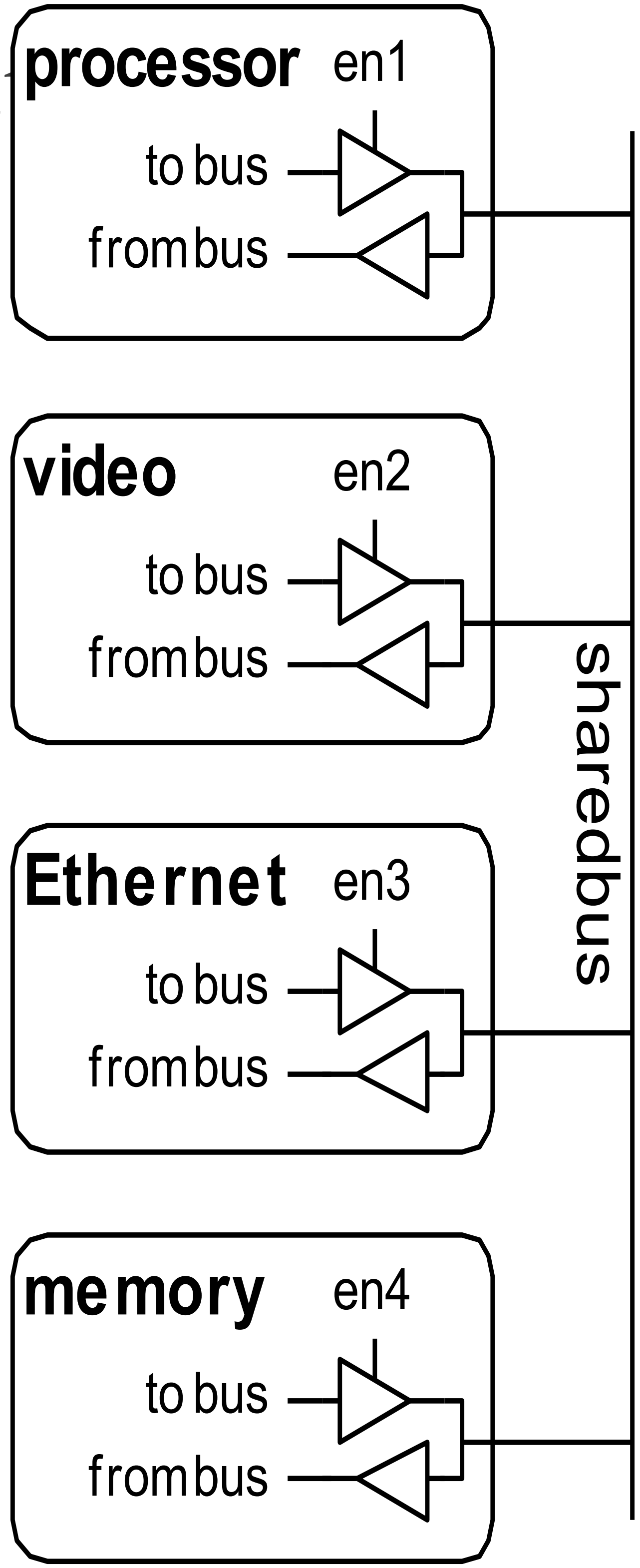
Application Scenario (1) of Tristate Buffers: 2-1 MUX

- **2x1 MUX Using Tristate Buffers**
  - Used in place of a MUX – this is the one case where the output lines can just be "tied together"
  - The control line s selects which buffer will pass its input



Application Scenario (2) of Tristate Buffers: Tristate Buses

- Floating nodes are used in trista
  - Many different drivers
  - Exactly one is active at once



EECE 2322: Fundamentals of Digital Design and Computer Organization

Lecture 4_1: Sequential and Combinational Circuits

Xiaolin Xu
Department of ECE
Northeastern University

Difference Between Combinational and Sequential Circuits

- Combinational Circuit
  - Time independent circuits
  - Do not depend on previous inputs to generate any output
-

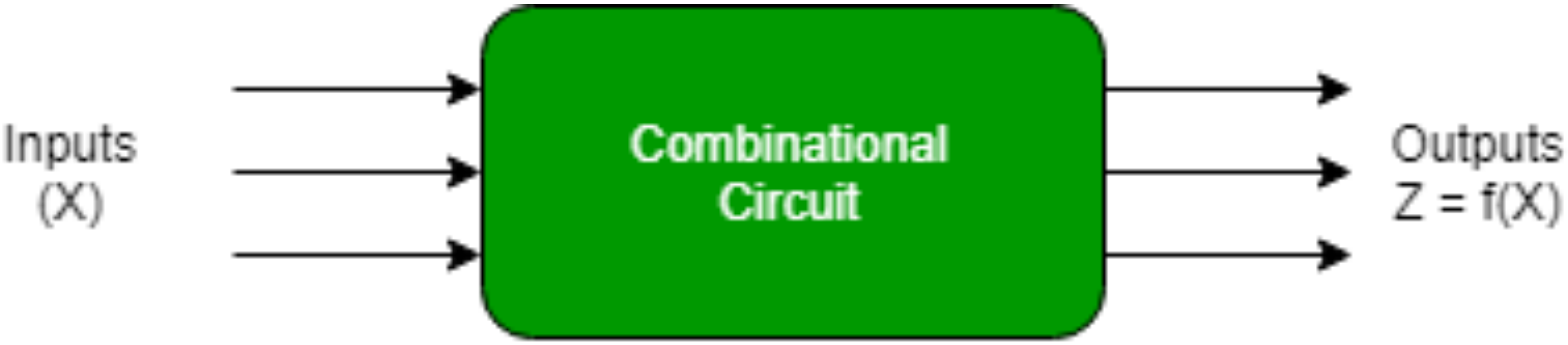


Figure: Combinational Circuits

Difference Between Combinational and Sequential Circuits

- Sequential Circuit
  - Dependent on
    - Clock cycles
    - Present inputs
    - Past inputs
  - to generate any

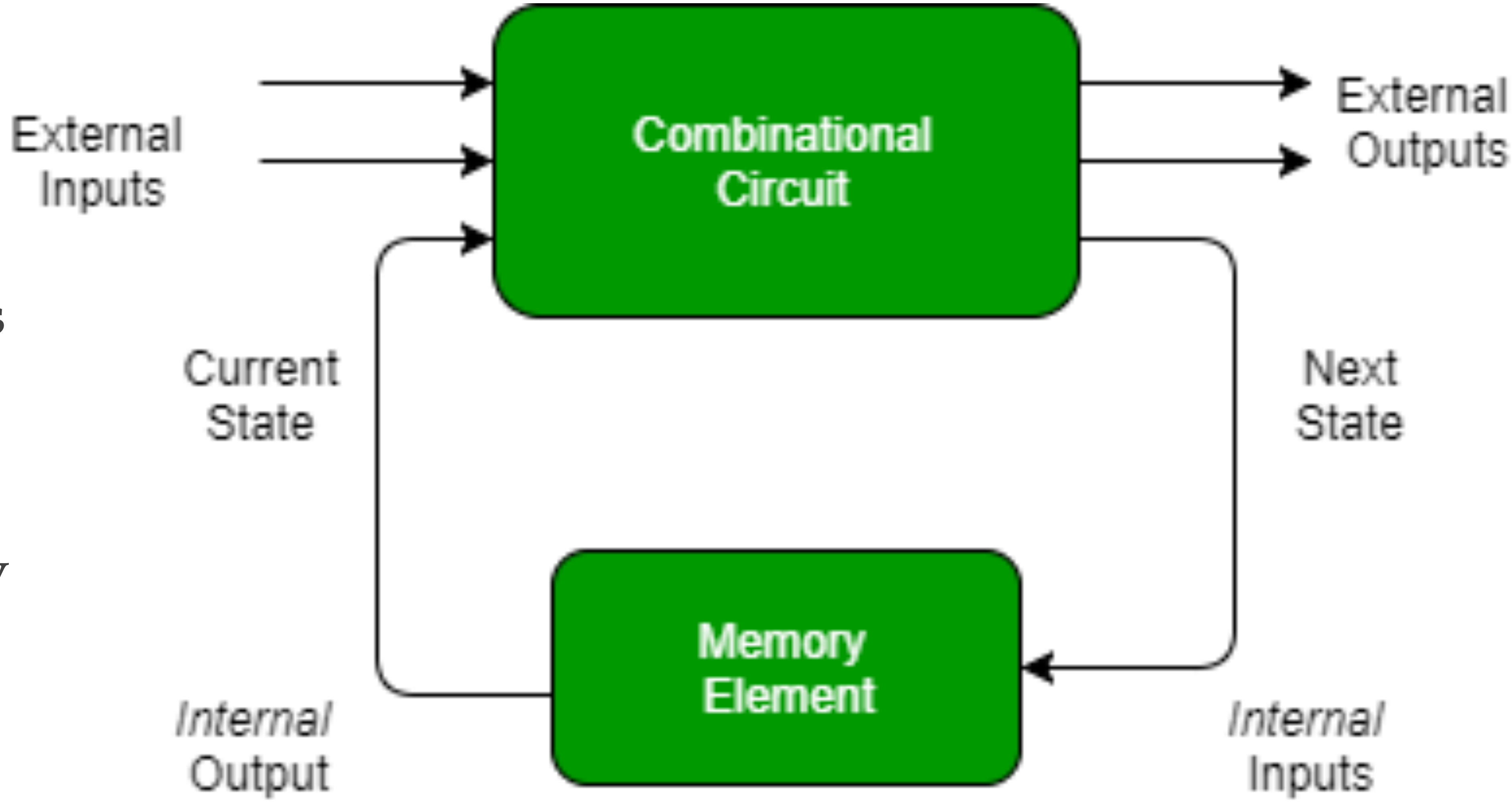


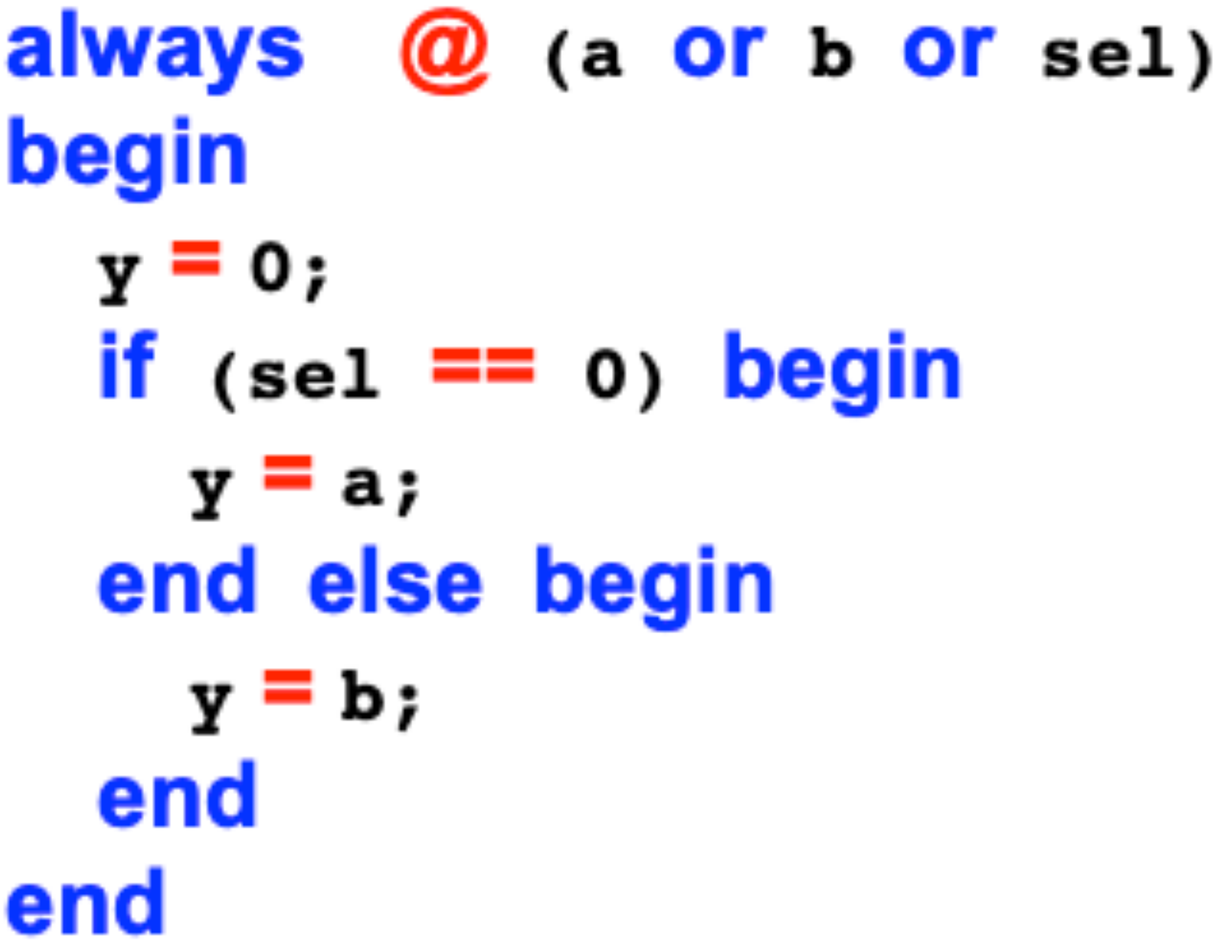
Figure: Sequential Circuit

Combinational and Sequential Circuits

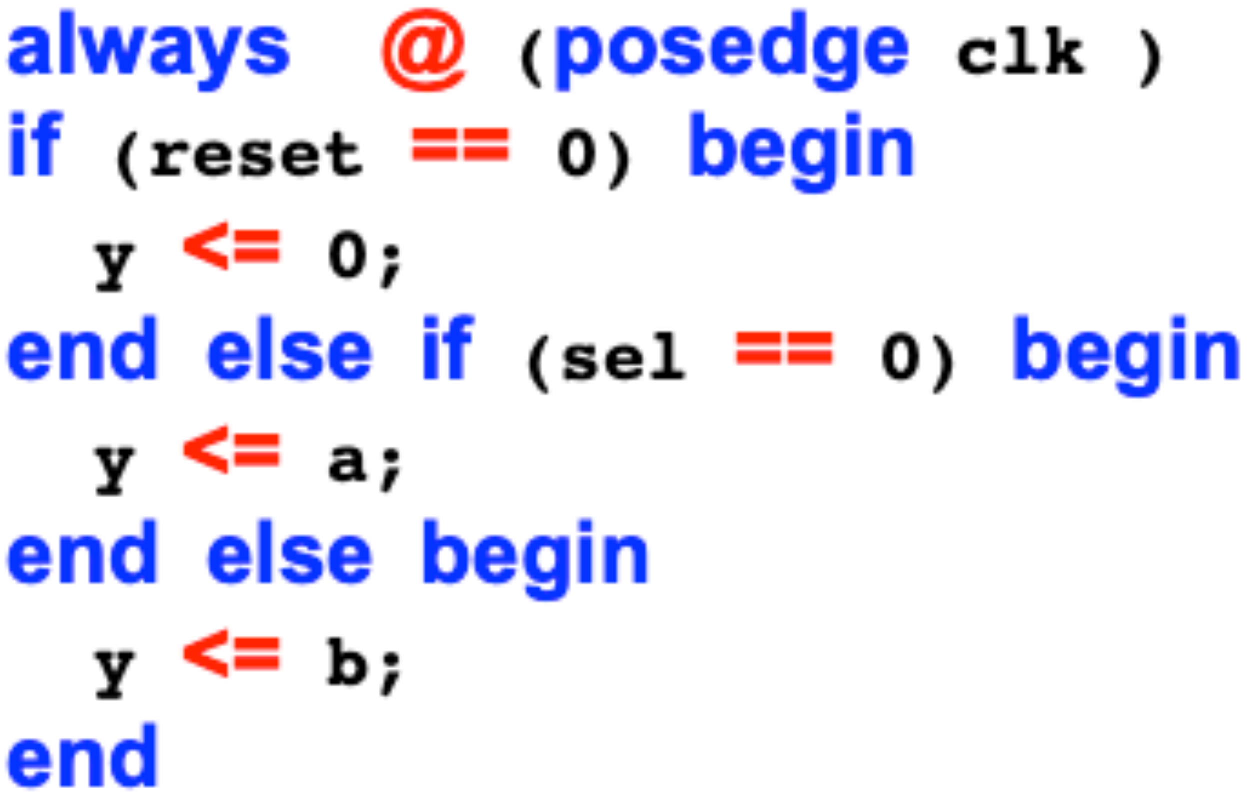
- **Summary**
  - Combinational Circuits
    - Output depends on current values of inputs only
    - No feedback
    - No memory
  - Sequential Hardware
    - Feedback
    - Output depends on current inputs and current state
    - Circuit has memory elements
    - Sequential Hardware = Combinational Hardware + memory elements (latches, flipflops, memories) ...

Level-Sensitive and Edge-Sensitive

- Left: will block until there is a change in the value of a or b
- Right: will block until clk transitions from 0 to 1.



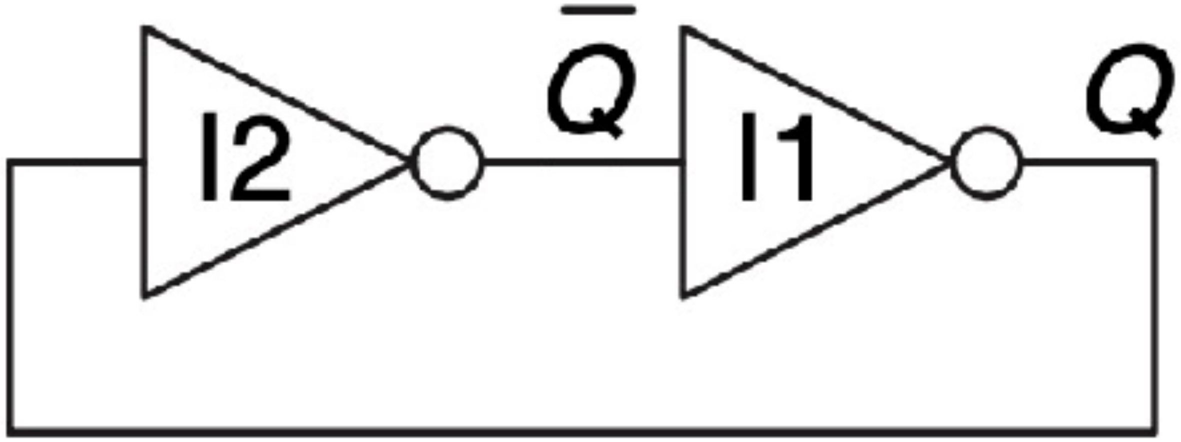
```
always  @ (a or b or sel)
begin
   y = 0;
   if (sel == 0) begin
      y = a;
   end else begin
      y = b;
   end
end
```



```
always  @ (posedge clk )
if (reset == 0) begin
   y <= 0;
end else if (sel == 0) begin
   y <= a;
end else begin
   y <= b;
end
```

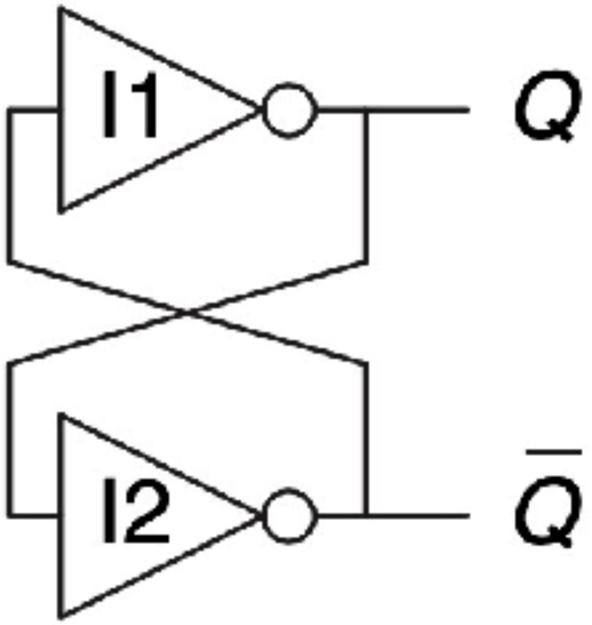
Sequential Logic (1): Memory

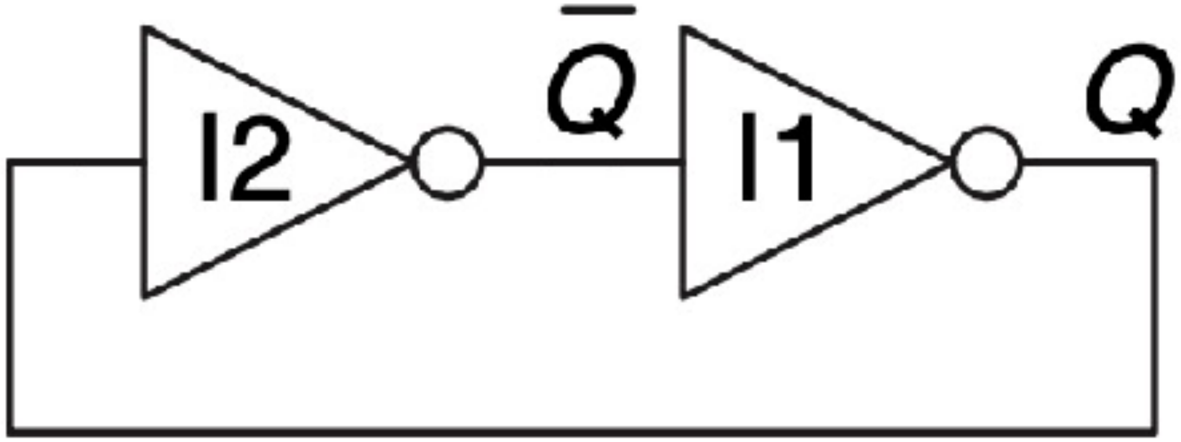
- Fundamental building block of memory
  - Bi-stable element
  - Two stable and complementary states



Sequential Logic (1): Memory

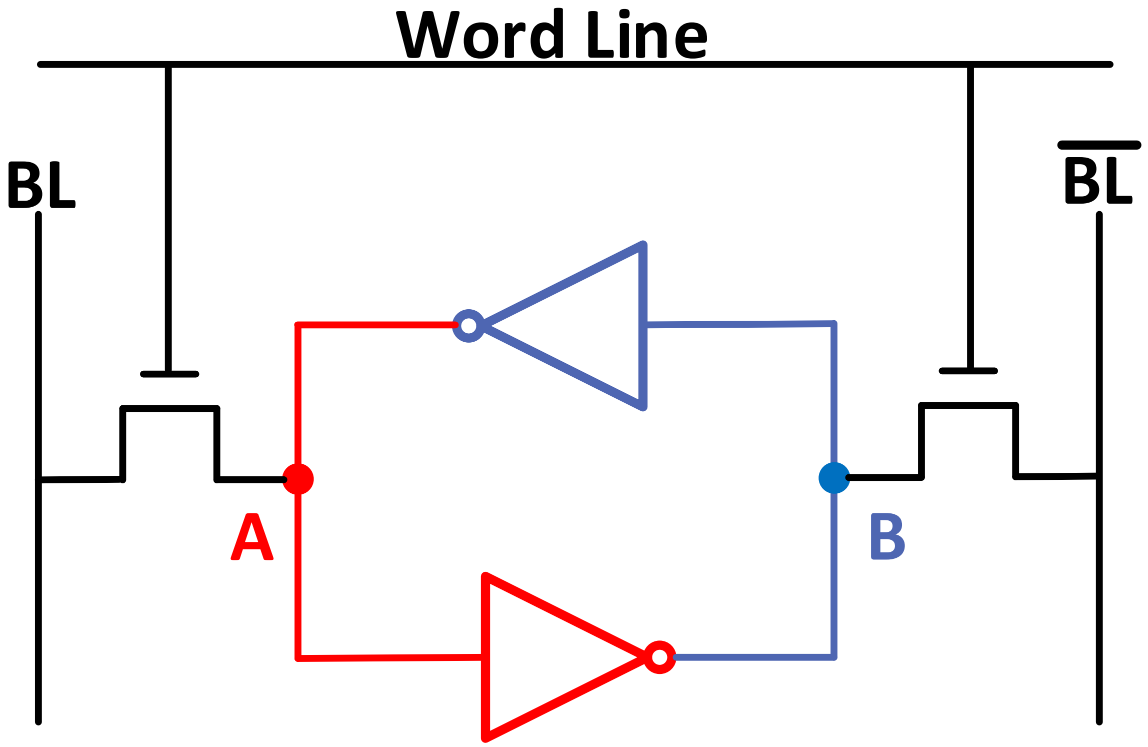
- Fundamental building block of memory
  - Bi-stable element
  - Two stable and complementary states



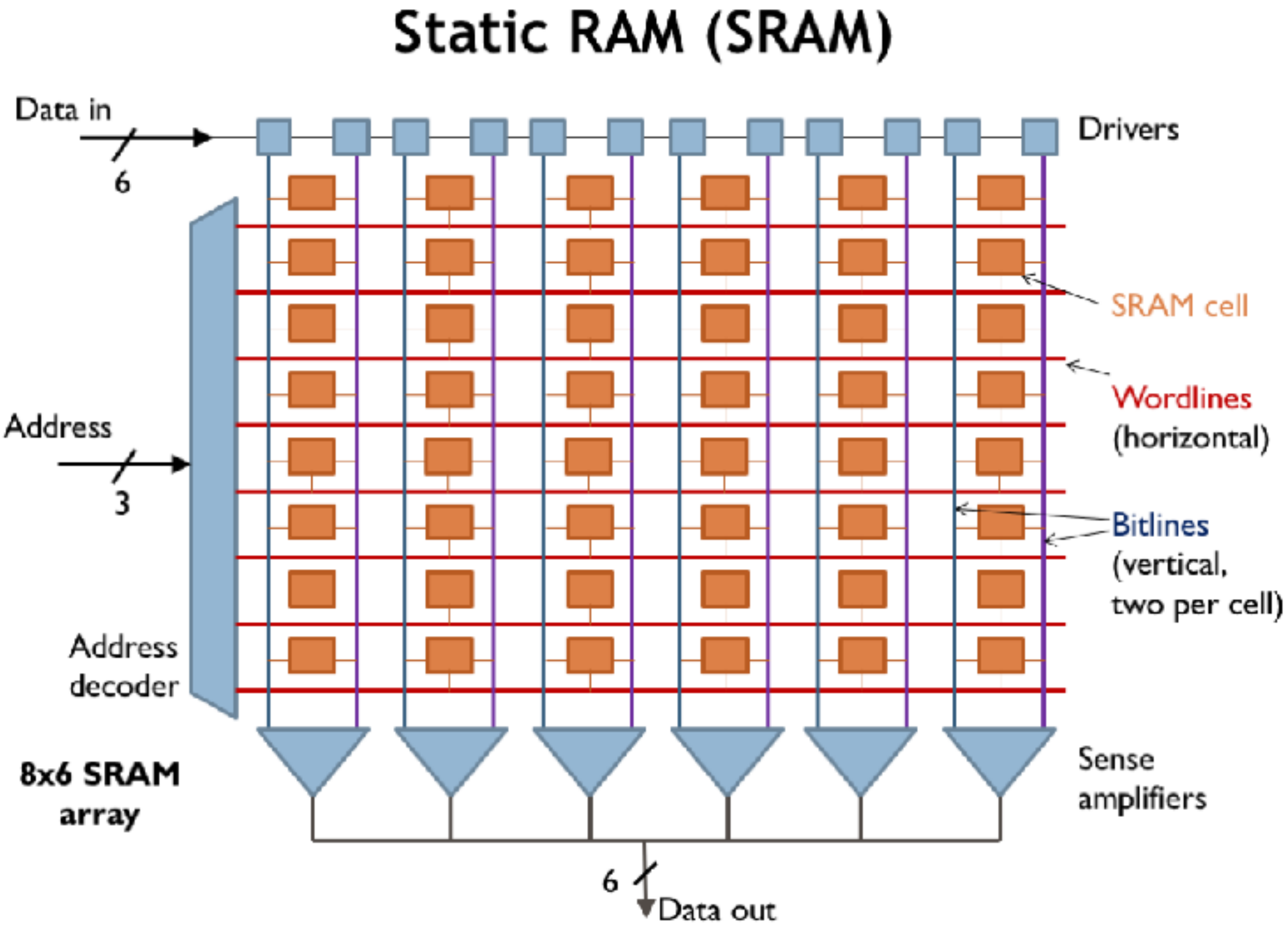


Static Random Access Memory

- Two possible states: “0” (AB=01) or “1” (AB=10)

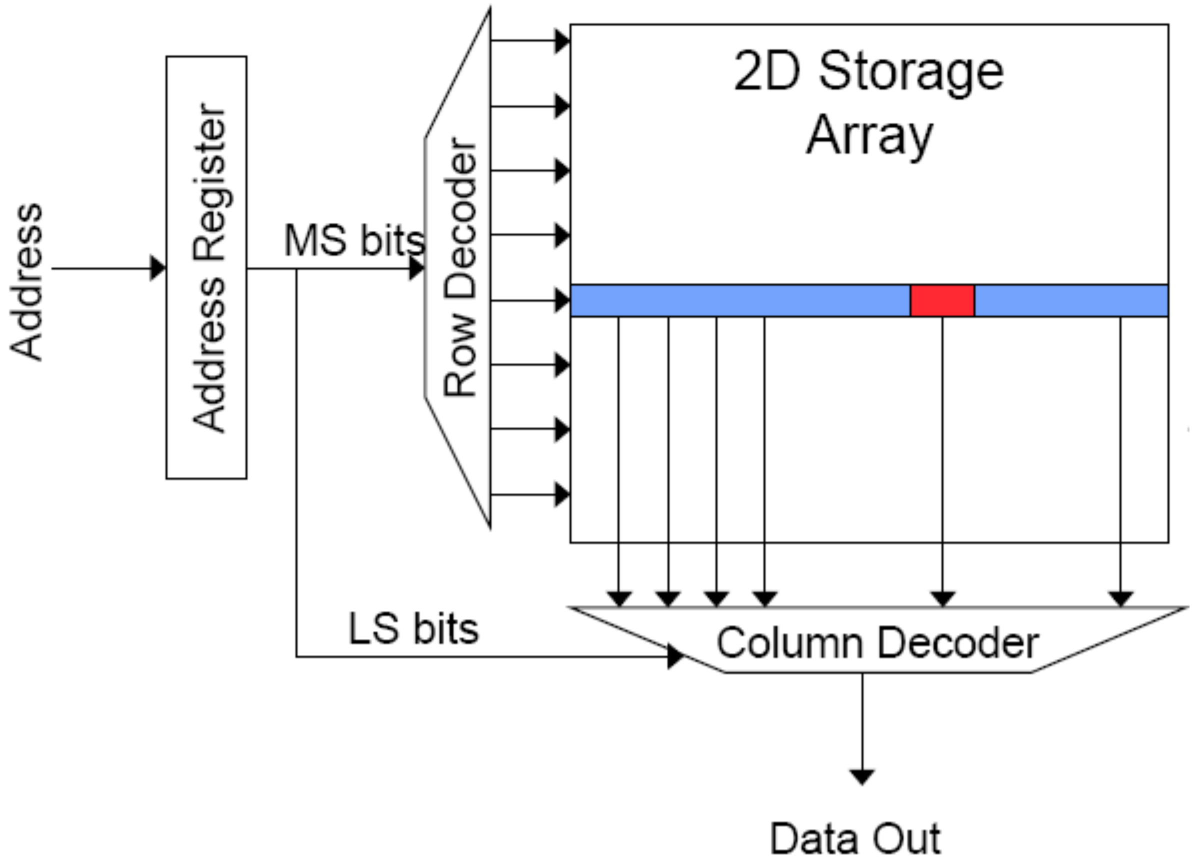


Static-RAM Array



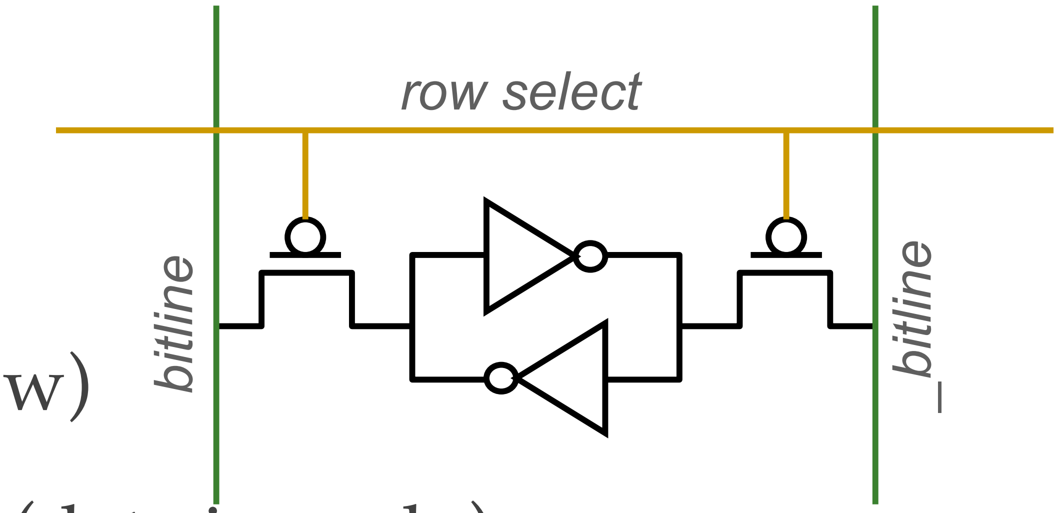
Memory Bank Organization and Operation

- Read access sequence:
- 1. Decode row address & drive
- 2. Selected bits drive bit-lines
  - Entire row read
- 3. Amplify row data
- 4. Decode column address & se
  - Send to output
- 5. Precharge bit-lines
  - For next access

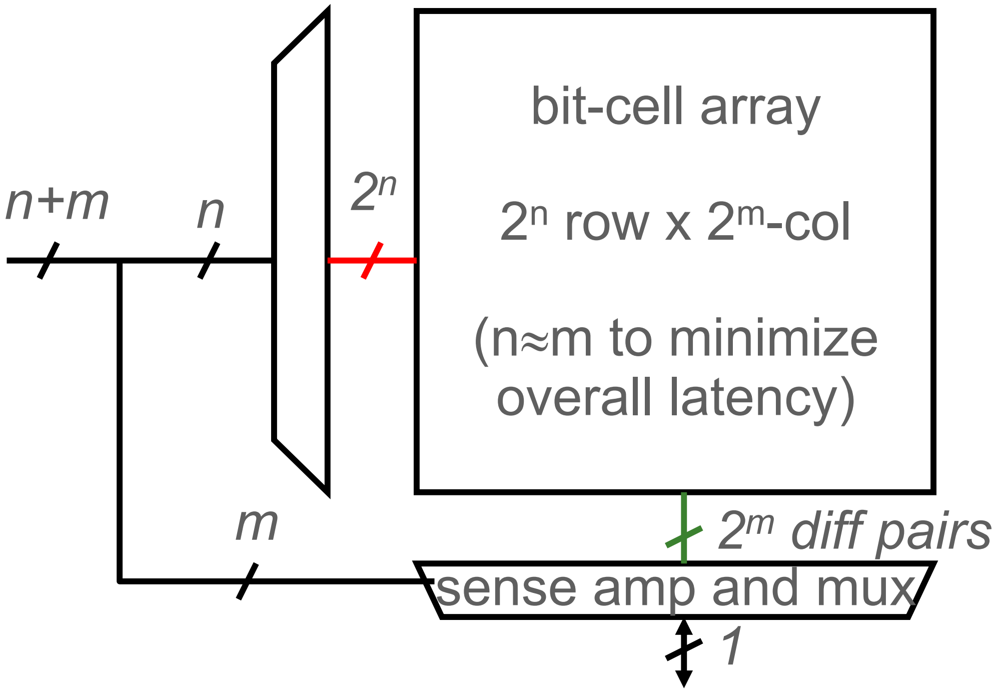


SRAM (Static Random Access Memory)

- Read Sequence
- 1. address decode
- 2. drive row select
- 3. selected bit-cells drive bitlines, (entire row)



- 4. differential sensing and column select (data is ready)
- 5. precharge all bitlines
- Access latency dominated by steps 2 and 3
- Cycling time dominated by steps 2, 3 and 5
  - step 2 proportional to 2^m
  - step 3 and 5 proportional to 2^n


n+m
n
2^n
bit-cell array
2^n row x 2^m-col
(n≈m to minimize overall latency)
m
2^m diff pairs
sense amp and mux
1