Attention, please!

- Homework Assignment 1 is due on Feb. 19 in class.
- Midterm Exam1 is scheduled Feb. 24 in class.
- On Feb. 19, I will spend one lecture to help you prepare Midterm Exam1. Don't miss it!

Chapter 3 Part 2

- Functions and functional blocks
- Rudimentary logic functions
- Selecting
- Decoding
- Encoding
- Implementing Combinational Functions Using:
  - Decoders and OR gates
  - Multiplexers (and inverter)

Functions and Functional Blocks

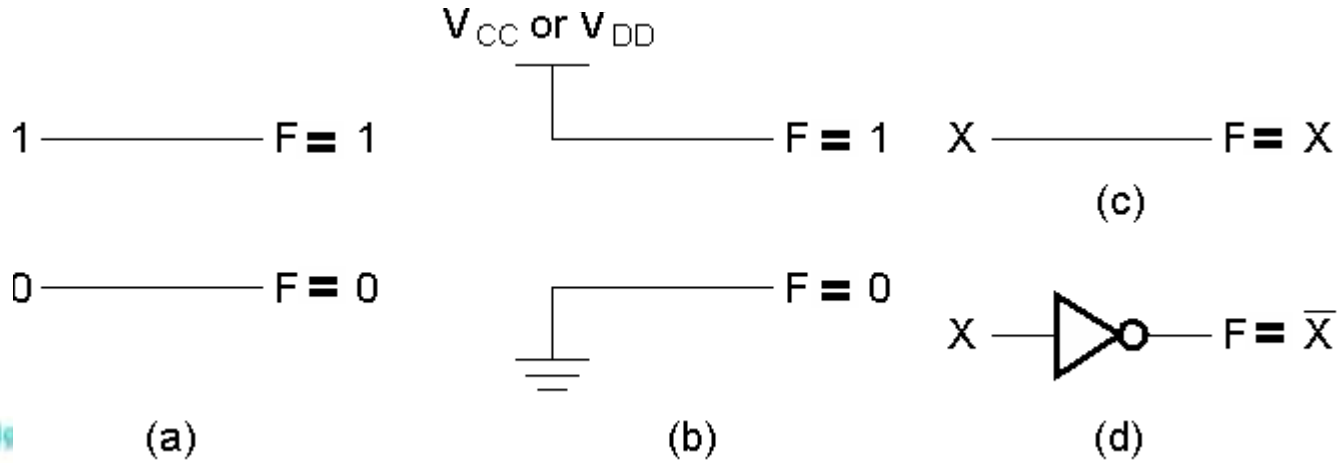
- The functions considered are those found to be very useful in design
- Corresponding to each of the functions is a combinational circuit implementation called a *functional block*.
- In the past, many functional blocks were implemented as SSI, MSI, and LSI circuits.
- Today, they are often simply parts within a VLSI circuits.

Rudimentary Logic Functions

- Functions of a single variable X
- Can be used on the inputs to functional blocks

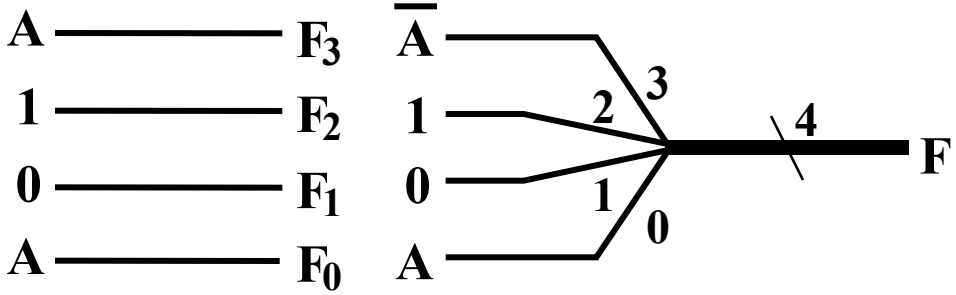
Functions of One Variable

X	F = 0	F = X	F = $\bar{X}$	F = 1
0	0	0	1	1
1	0	1	0	1



Multiple-bit Rudimentary Functions

- Multi-bit Examples:



- A wide line is used to represent a *bus* which is a vector signal
- In (b) of the example, F = (F_3, F_2, F_1, F_0) is a bus.
- The bus can be split into individual bits as shown in (b)
- Sets of bits can be split from the bus as shown in (c) for bits 2 and 1 of F.
- The sets of bits need not be continuous as shown in (d) for bits 3, 1, and 0 of F.

Selecting

- Selecting of data or information is a critical function in digital systems and computers
- Circuits that perform selecting have:
  - A set of information inputs from which the selection is made
  - A single output
  - A set of control lines for making the selection
- Logic circuits that perform selecting are called *multiplexers*
- Selecting can also be done by three-state logic or transmission gates

Multiplexers

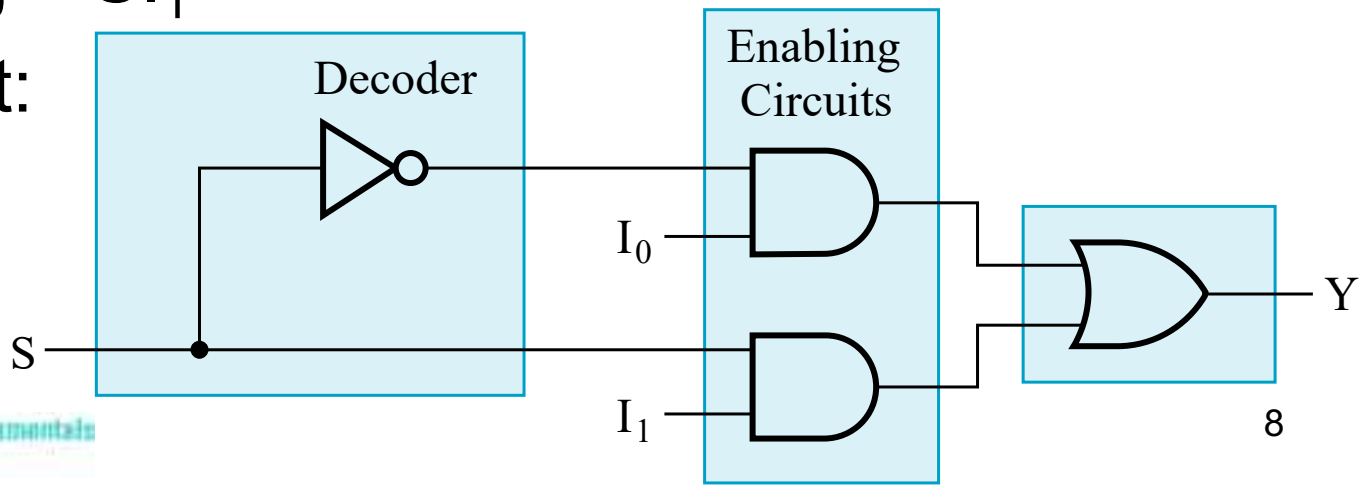
- A multiplexer selects information from an input line and directs the information to an output line
- A typical multiplexer has *n* control inputs (S_{n-1}, … S_0) called *selection inputs*, 2^n information inputs (I_{2^n-1}, … I_0), and one output Y
- A multiplexer can be designed to have *m* information inputs with $m < 2^n$ as well as *n* selection inputs

2-to-1-Line Multiplexer

- Since $2 = 2^1$, n = 1
- The single selection variable S has two values:
  - S = 0 selects input I_0
  - S = 1 selects input I_1
- The equation:

$$Y = \overline{S} I_0 + S I_1$$

- The circuit:

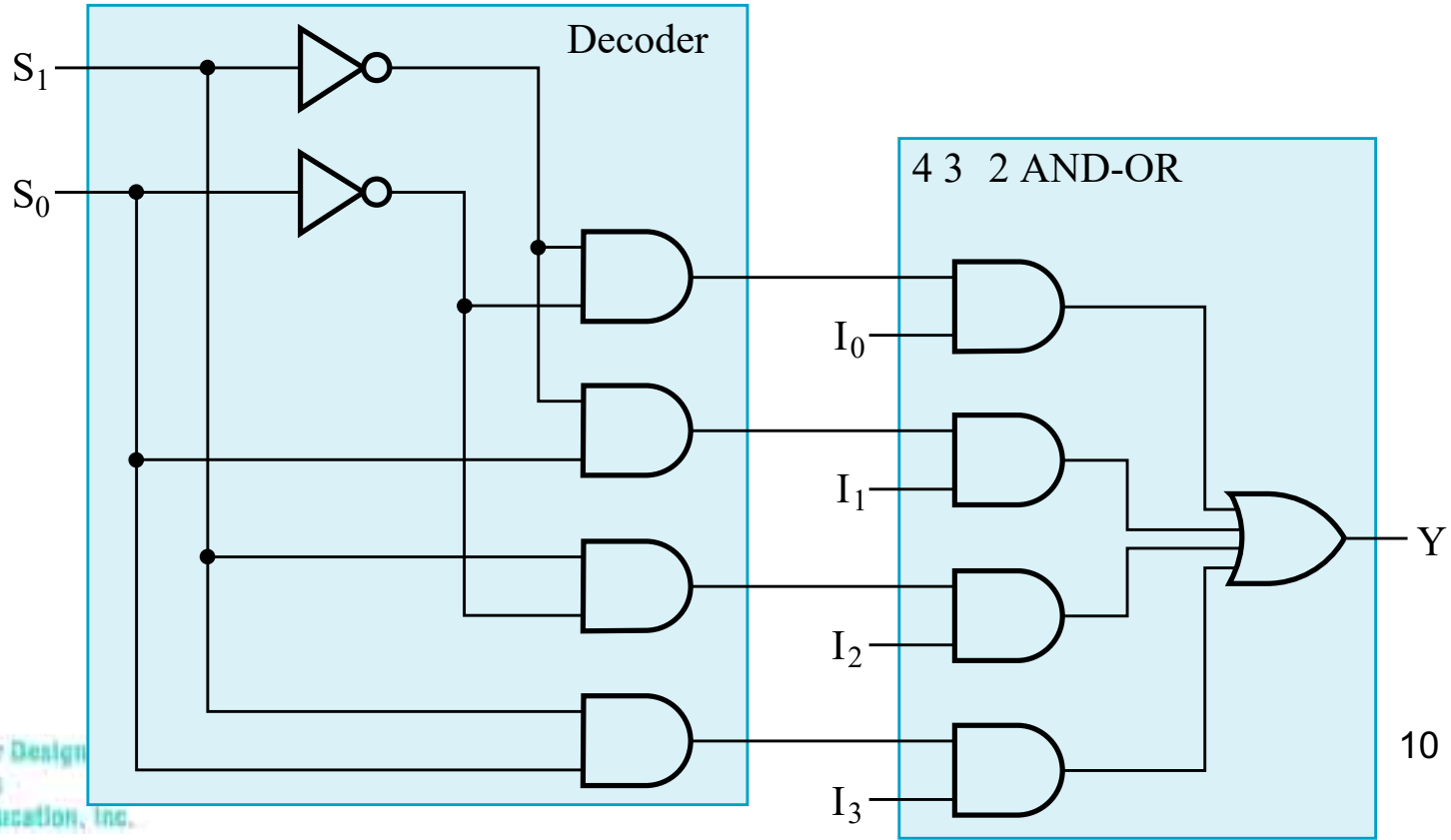


2-to-1-Line Multiplexer (**continued**)

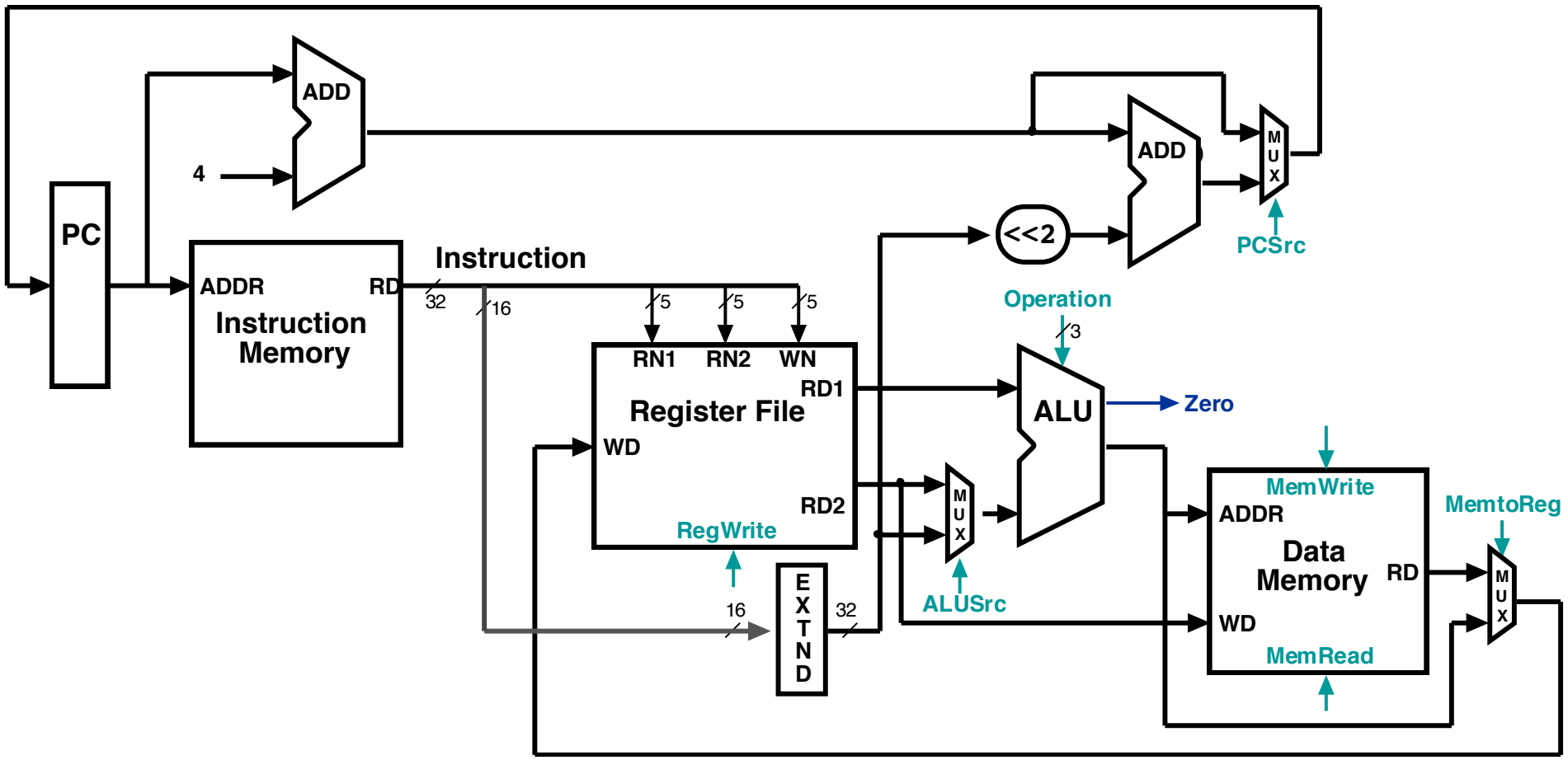
- Note the regions of the multiplexer circuit shown:
  - 1-to-2-line Decoder
  - 2 Enabling circuits
  - 2-input OR gate
- To obtain a basis for multiplexer expansion, we combine the Enabling circuits and OR gate into a 2×2 AND-OR circuit:
  - 1-to-2-line decoder
  - 2×2 AND-OR
- In general, for an 2^n-to-1-line multiplexer:
  - n-to-2^n-line decoder
  - $2^n \times 2$ AND-OR

Example: 4-to-1-line Multiplexer

- 2-to-2^2-line decoder
- $2^2 \times$ 2 AND-OR



Complete Single-Cycle Datapath



Demultiplexers

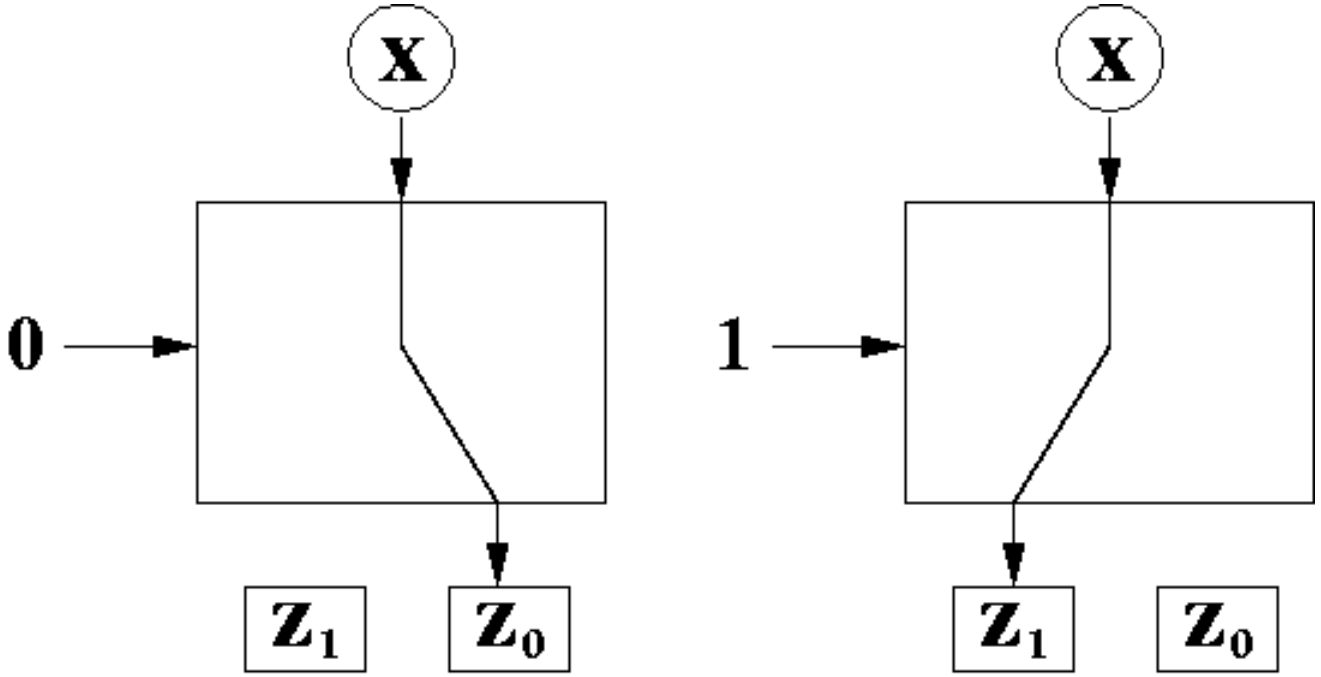
Demultiplexers (or DeMUX for short) are basically multiplexers where the inputs and outputs have been switched. An **1-n** DeMUX consists of the following:

- Data inputs: **1**
- Control inputs: **ceil($\log_2$ n)**
- Outputs: **n**

Nature of DeMUX

- Think of a DeMUX like a mailroom. You have many pieces of letters coming in, and you must distribute each letter to one of many different mail boxes.

Behavior of 1-2 DeMUX



Explanation

- When **c == 0**, the input, **x**, is directed to the output $\mathbf{z_0}$. When **c == 1**, the input, **x**, is directed to the output $\mathbf{z_1}$. Just as before, we think of **c** as a **1** bit number, which specifies the output we want to direct the input to.

Truth Table for 1-2 DeMUX

Row	c	z_0	z_1
0	0	**x**	**0**
1	**1**	**0**	**x**

Boolean Expression

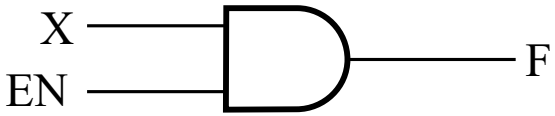
- $z_1 = c\mathbf{x}$
- $z_0 = c'\mathbf{x}$

Enabling Function

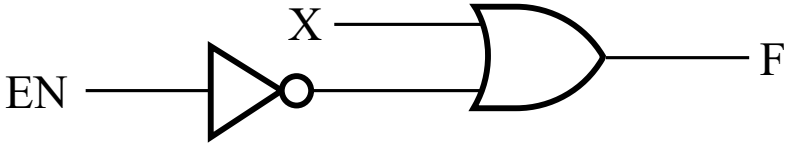
- *Enabling* permits an input signal to pass through to an output
- *Disabling* blocks an input signal from passing through to an output, replacing it with a fixed value
- The value on the output when it is disable can be Hi-Z (as for three-state buffers and transmission gates), 0 , or 1 (see next slide)

Enabling

- When disabled, 0 output
- When disabled, 1 output



(a)



(b)

Decoding

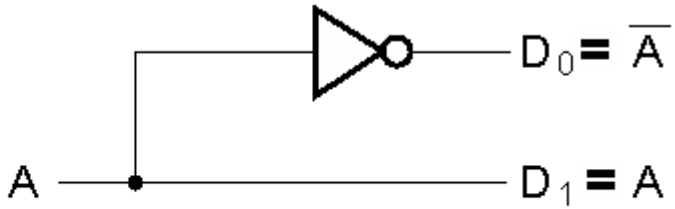
- Decoding - the conversion of an *n*-bit input code to an *m*-bit output code with $n \leq m \leq 2^n$ such that each valid code word produces a unique output code
- Circuits that perform decoding are called *decoders*
- Here, functional blocks for decoding are
  - called *n*-to-*m* line decoders, where $m \leq 2^n$, and
  - generate 2^n (or fewer) minterms for the *n* input variables

Decoder Examples

- 1-to-2-Line Decoder

A	D_0	D_1
0	1	0
1	0	1

(a)

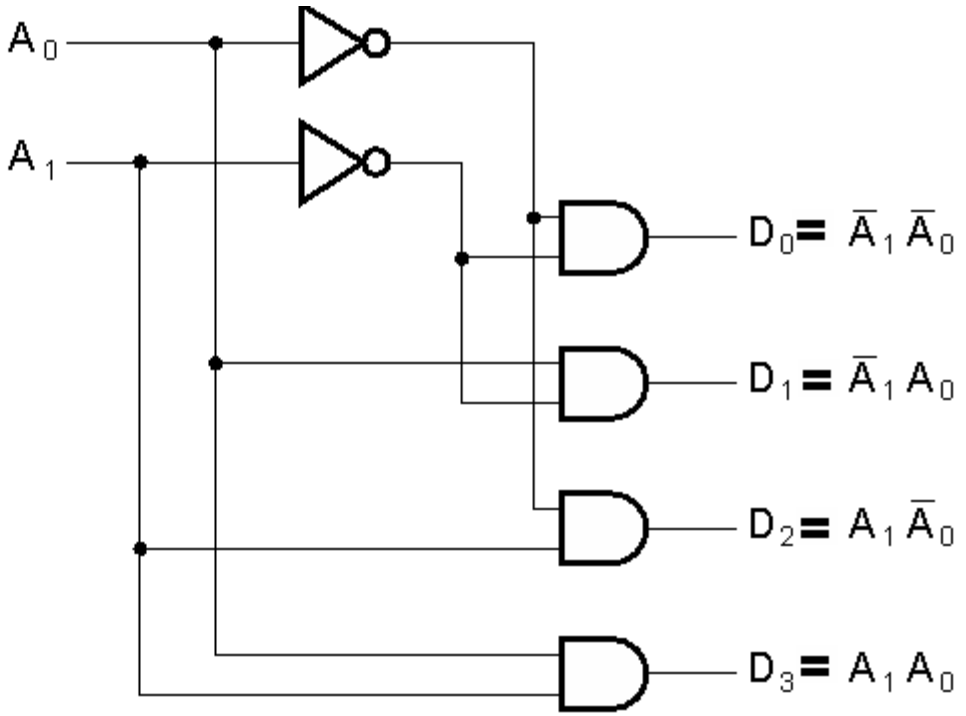


(b)

- 2-to-4-Line Decoder

A_1	A_0	D_0	D_1	D_2	D_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

(a)



(b)

Decoder Expansion

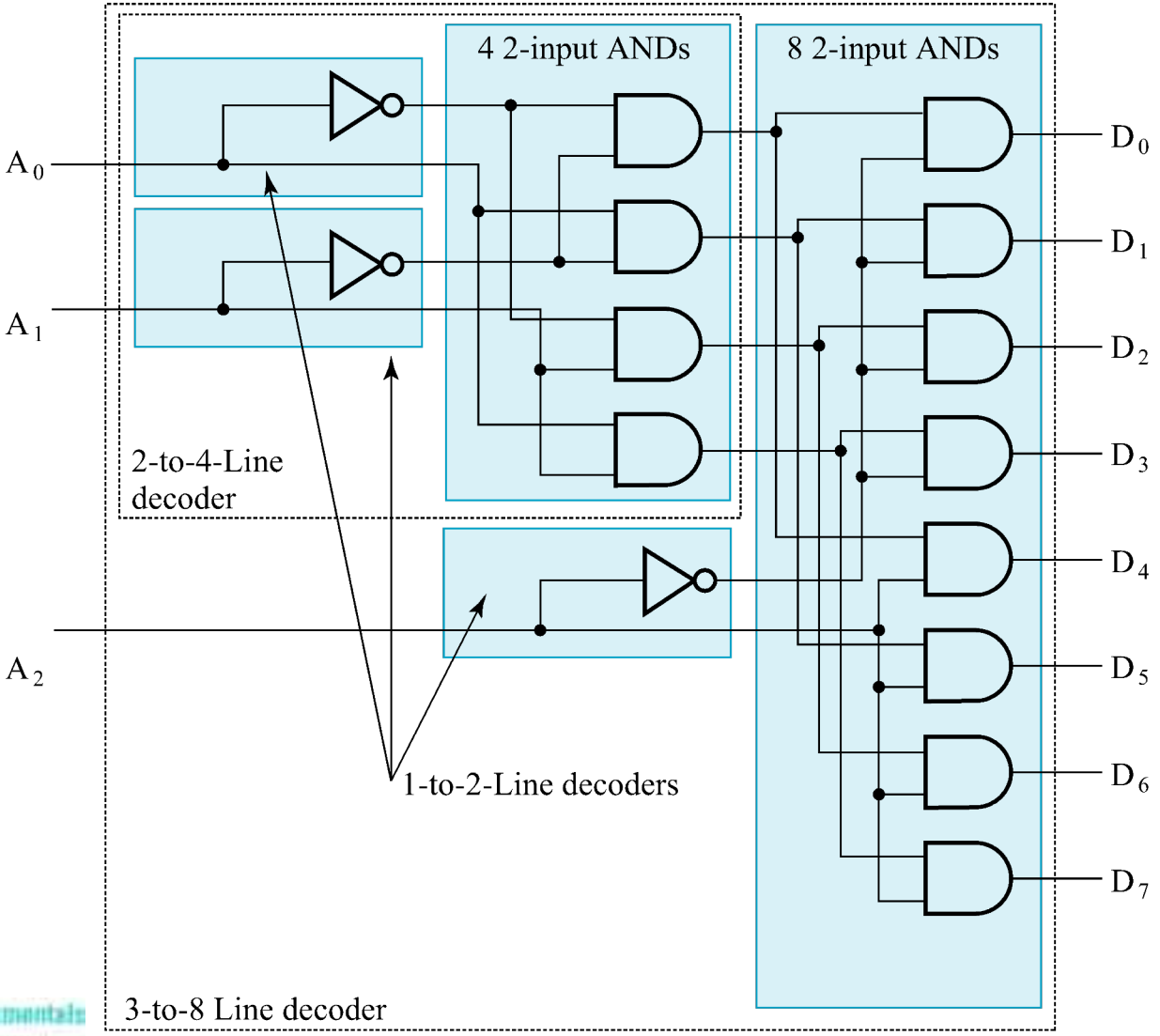
- General procedure given in book for any decoder with *n* inputs and 2^n outputs.
- This procedure builds a decoder backward from the outputs.
- The output AND gates are driven by two decoders with their numbers of inputs either equal or differing by 1.
- These decoders are then designed using the same procedure until 2-to-1-line decoders are reached.
- The procedure can be modified to apply to decoders with the number of outputs $\neq 2^n$

Decoder Expansion - Example 1

- 3-to-8-line decoder
  - Number of output ANDs = 8
  - Number of inputs to decoders driving output ANDs = 3
  - Closest possible split to equal
    - 2-to-4-line decoder
    - 1-to-2-line decoder
  - 2-to-4-line decoder
    - Number of output ANDs = 4
    - Number of inputs to decoders driving output ANDs = 2
    - Closest possible split to equal
      - Two 1-to-2-line decoders
- See next slide for result

Decoder Expansion - Example 1

- Result

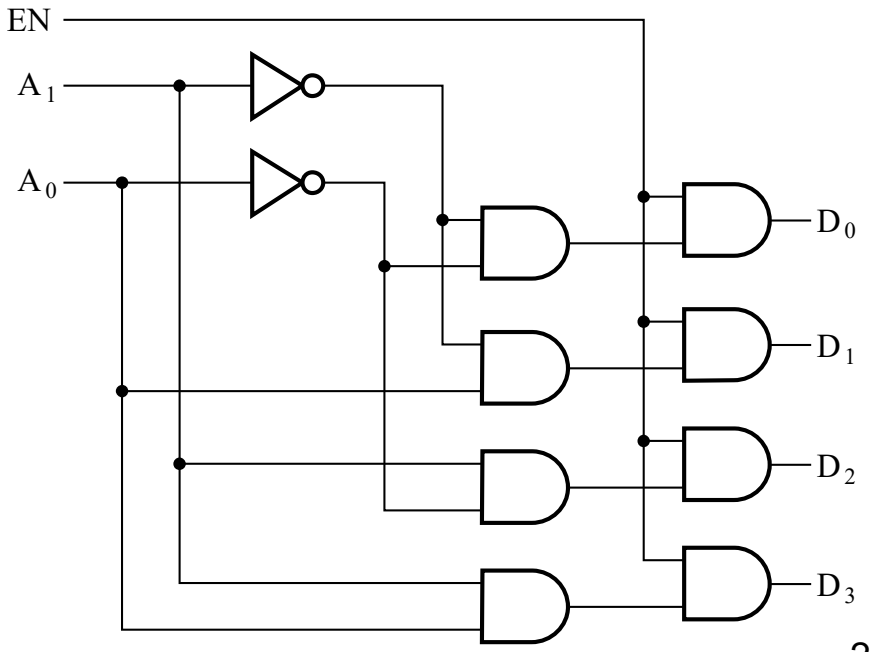


Decoder with Enable

- In general, attach *m*-enabling circuits to the outputs
- See truth table below for function
  - Note use of X's to denote both 0 and 1
  - Combination containing two X's represent four binary combinations
- Alternatively, can be viewed as distributing value of signal EN to 1 of 4 outputs
- In this case, called a *demultiplexer*

EN	A_1	A_0	D_0	D_1	D_2	D_3
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

(a)



(b)

Encoding

- Encoding - the opposite of decoding - the conversion of an *m*-bit input code to a *n*-bit output code with $n \leq m \leq 2^n$ such that each valid code word produces a unique output code
- Circuits that perform encoding are called *encoders*
- An encoder has 2^n (or fewer) input lines and *n* output lines which generate the binary code corresponding to the input values
- Typically, an encoder converts a code containing exactly one bit that is 1 to a binary code corresponding to the position in which the 1 appears.

Encoder Example

- A decimal-to-BCD encoder
  - Inputs: 10 bits corresponding to decimal digits 0 through 9, (D_0, …, D_9)
  - Outputs: 4 bits with BCD codes
  - Function: If input bit D_i is a 1, then the output (A_3, A_2, A_1, A_0) is the BCD code for i,
- The truth table could be formed, but alternatively, the equations for each of the four outputs can be obtained directly.

Truth table of the decimal-to-BCD encoder

D9	D8	D7	D6	D5	D4	D3	D2	D1	D0	A3	A2	A1	A0
0	0	0	0	0	0	0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	0	1	0	0	0	0	1
0	0	0	0	0	0	0	1	0	0	0	0	1	0
0	0	0	0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	0	1	0	0	0	0	0	1	0	0
0	0	0	0	1	0	0	0	0	0	0	1	0	1
0	0	0	1	0	0	0	0	0	0	0	1	1	0
0	0	1	0	0	0	0	0	0	0	0	1	1	1
0	1	0	0	0	0	0	0	0	0	1	0	0	0
1	0	0	0	0	0	0	0	0	0	1	0	0	1

From table:

A3 = D8 + D9
A2 = D4 + D5 + D6 + D7
A1 = D2 + D3 + D6 + D7
A0 = D1 + D3 + D5 + D7 + D9

We made use of the fact that only one input can be "1" at one time

Priority Encoder

- If more than one input value is 1, then the encoder just designed does not work.
- One encoder that can accept all possible combinations of input values and produce a meaningful result is a *priority encoder*.
- Among the 1s that appear, it selects the most significant input position (or the least significant input position) containing a 1 and responds with the corresponding binary code for that position.

Priority Encoder Example

- Priority encoder with 5 inputs (D_4, D_3, D_2, D_1, D_0) - highest priority to most significant 1 present - Code outputs A2, A1, A0 and V where V indicates at least one 1 present.

No. of Min-terms/Row	Inputs					Outputs			
	D4	D3	D2	D1	D0	A2	A1	A0	V
1	0	0	0	0	0	X	X	X	0
1	0	0	0	0	1	0	0	0	1
2	0	0	0	1	X	0	0	1	1
4	0	0	1	X	X	0	1	0	1
8	0	1	X	X	X	0	1	1	1
16	1	X	X	X	X	1	0	0	1

- Xs in input part of table represent 0 or 1; thus table entries correspond to product terms instead of minterms. The column on the left shows that all 32 minterms are present in the product terms in the table

Priority Encoder Example **(continued)**

- **Could use a K-map to get equations, but can be read directly from table and manually optimized if careful:**

$$A_2 = D_4$$

$$A_1 = \overline{D}_4 D_3 + \overline{D}_4 \overline{D}_3 D_2 = \overline{D}_4 F_1,\ F_1 = (D_3 + D_2)$$

$$A_0 = \overline{D}_4 D_3 + \overline{D}_4 \overline{D}_3 \overline{D}_2 D_1 = \overline{D}_4 (D_3 + \overline{D}_2 D1)$$

$$V = D_4 + F_1 + D_1 + D_0$$

Q 3-8

- Design a combinational circuit that accept a 3-bit number and generates a 6-bit binary number output equal to the square of the input number

Answer

A	B	C	S5	S4	S3	S2	S1	S0
0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	1
0	1	0	0	0	0	1	0	0
0	1	1	0	0	1	0	0	1
1	0	0	0	1	0	0	0	0
1	0	1	0	1	1	0	0	1
1	1	0	1	0	0	1	0	0
1	1	1	1	1	0	0	0	1

$S0 = C$

$S1 = 0$

$S2 = \overline{A}B\overline{C} + AB\overline{C}$

$S3 = \overline{A}BC + A\overline{B}C$

$S4 = A\overline{B} + AC$

$S5 = AB$

Weekly Exercises

- 3-2
- 3-8
- 3-13
- 3-16
- 3-30
- 3-35
- 3-36
- 3-39
- 3-42
- 3-47