Digital Logic Design

CS251 Part 1

CLICKER QUESTION FROM LAST DAY

What would be the correct instruction to Implement the following operation:

Go to Data memory at an address computed by the contents of $3 plus 100

Put that data into register $4

A) sw $4, 100($3)

B) lw $s4, 100($s3)

C) addi $s4 , $s3, 100

D) lw $s3, 100($s4)

E) none of the above

What would be the correct instruction to Implement the following operation:

Take a branch if the contents of $4 and $3 are not equal

A) beq $4, $3, 5

B) beq $3, $4, 5

C) bne $4, $4, $3, 5

D) bne $3, $4, 5

E) j 5

My Office Hours DC 2132

(Will vary in June and July)

- Wednesday 1-2pm
- Thursday: 11am-12

- **Check Piazza For Changes**

Truth Tables and Laws of Boolean Algebra

Course Notes Module 02
Examples on the Board

DeMorgan's Law:

$$\overline{X+Y} = \overline{X} \cdot \overline{Y} \qquad \overline{XY} = \overline{X} + \overline{Y}$$

First we will look at Logic Gates: How DeMorgan's is implemented via Gates

Distributive Law:

$X + YZ = (X + Y)(X + Z)$ why?

$=XX + XZ + XY + YZ$

Factor out X :

$=X\ (X + Z + Y) + YZ$

If X is false

Distributive :

$X + YZ = (X + Y)(X + Z)$ why?

$=XX + XZ + XY + YZ$

Factor out X :

$=X\ (X + Z + Y) + YZ$

Just X being True is enough
To make the first part of the expression true.
Therefore bracketed term drops

Distributive :

$X + YZ = (X + Y)(X + Z)$ why?

$=XX + XZ + XY + YZ$

Factor out X :

$=X+ YZ$

Just X being True is enough
To make the first part of the expression true.
Therefore bracketed term drops

Distributive :

$X + YZ = (X + Y)(X + Z)$ why?

$=XX + XZ + XY + YZ$

Factor out X :

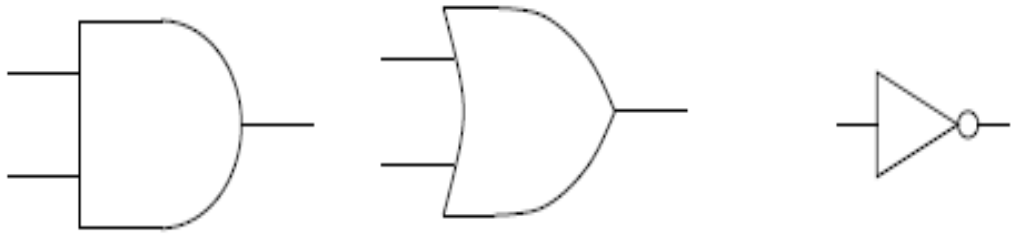
$=X+ YZ$

Just X being True is enough
To make the first part of the expression true.
Therefore bracketed term drops

Binary Numbers: 0010 ? Binary Number system: Review

Using Gates in Logic Design

- Here are symbols for AND, OR, NOT gates



- NOT often drawn as "bubble" on input or output
- AND, OR can be generalized to many inputs (useful)

Review Each of these Gates and Their Truth Tables: On Board

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

Formula Simplification Using Laws

- We can use algebraic manipulation (based on laws) to simplify formulas
- An example using the previous truth table

$$\begin{aligned} F &= \overline{X}\,\overline{Y}Z + X\overline{Y}Z + XY\overline{Z} + XYZ \\ &= \overline{Y}Z(\overline{X} + X) + XY(\overline{Z} + Z) \\ &= \overline{Y}Z + XY \end{aligned}$$

- Difficult even for humans, tricky to automate
- Seems inherently hard to get "simplest" formula
- Is simplest formula the best for implementation?

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

What was the formula for G?

G= Not (XYZ)

Formula Simplification Using Laws

- We can use algebraic manipulation (based on laws) to simplify formulas
- An example using the previous truth table

$$\begin{aligned} F &= \overline{X}\,\overline{Y}Z + X\overline{Y}Z + XY\overline{Z} + XYZ \\ &= \overline{Y}Z(\overline{X} + X) + XY(\overline{Z} + Z) \\ &= \overline{Y}Z + XY \end{aligned}$$

- Difficult even for humans, tricky to automate
- Seems inherently hard to get "simplest" formula
- Is simplest formula the best for implementation?

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

Give The Best Correct *reduced* Answer:

(note the bars above the inputs are separated)

A) $F = \overline{XYZ} + \overline{XY}Z + X\overline{YZ} + XY\overline{Z}$

B) $F = \overline{XYZ} + \overline{XYZ} + X\overline{YZ} + XY\overline{Z}$

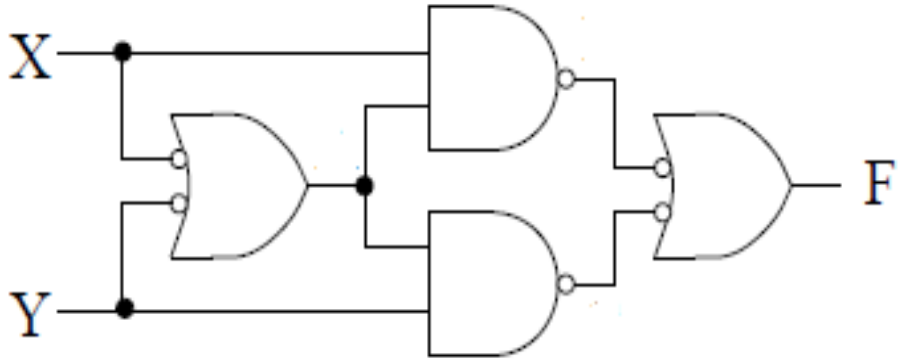
C) $F = \overline{XY} + X\overline{Z}$

D) $F = \overline{XYZ} + \overline{XY}Z + X\overline{YZ}$

E) None

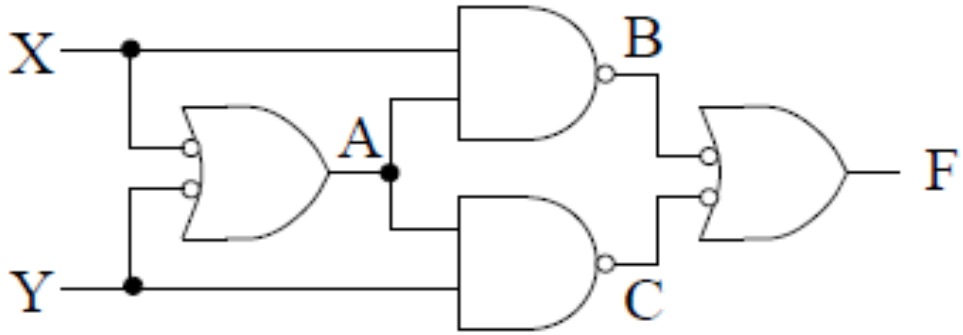
Deriving Truth Table from Circuit

- Label intermediate gate outputs
- Fill in truth table in appropriate order



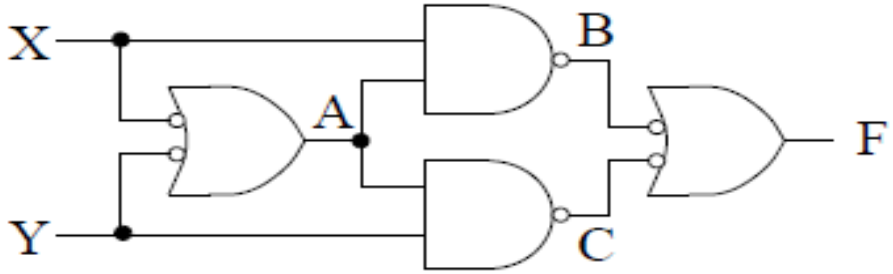
Deriving Truth Table from Circuit

- Label intermediate gate outputs
- Fill in truth table in appropriate order



Deriving Truth Table from Circuit

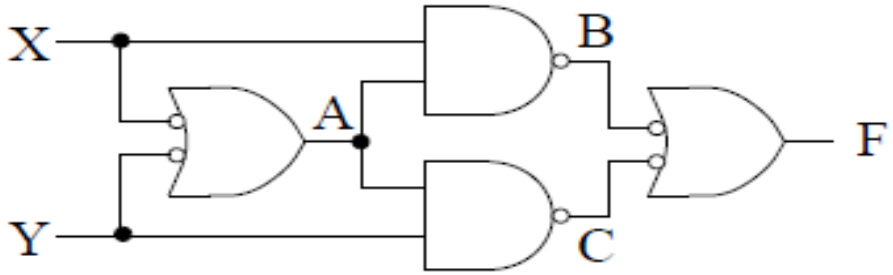
- Label intermediate gate outputs
- Fill in truth table in appropriate order



X	Y	A	B	C	F
0	0				
0	1				
1	0				
1	1				

Deriving Truth Table from Circuit

- Label intermediate gate outputs
- Fill in truth table in appropriate order

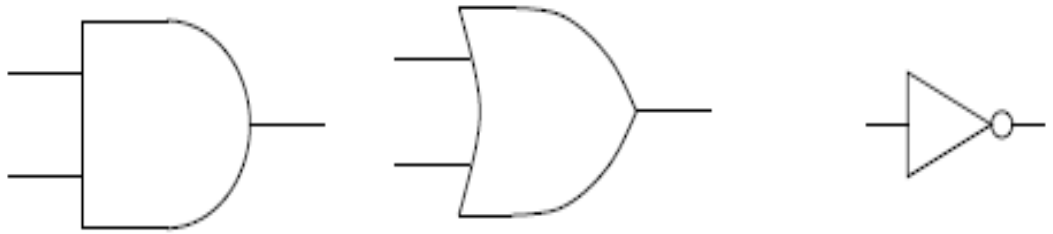


X	Y	A	B	C	F
0	0	0			
0	1	1			
1	0	1			
1	1	1			

IS THIS CORRECT?

Using Gates in Logic Design

- Here are symbols for AND, OR, NOT gates

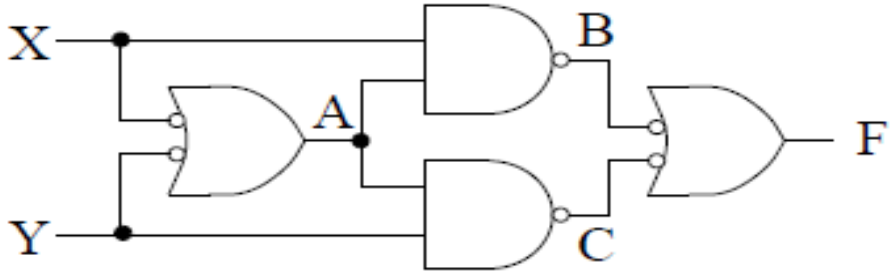


INVERTER!

- NOT often drawn as "bubble" on input or output
- AND, OR can be generalized to many inputs (useful)

Deriving Truth Table from Circuit

- Label intermediate gate outputs
- Fill in truth table in appropriate order



X	Y	A	B	C	F
0	0	0			
0	1	1			
1	0	1			
1	1	1			

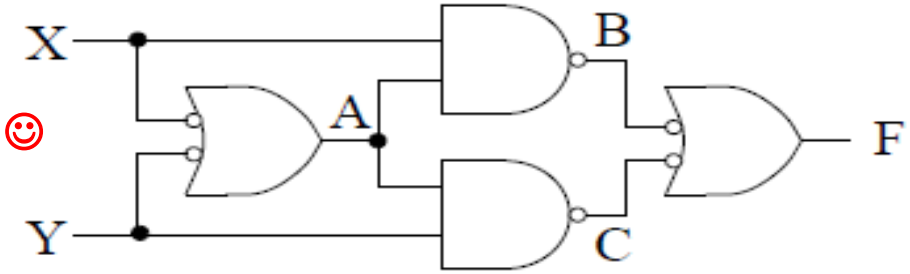
IS THIS CORRECT?

NO

Deriving Truth Table from Circuit

- Label intermediate gate outputs
- Fill in truth table in appropriate order

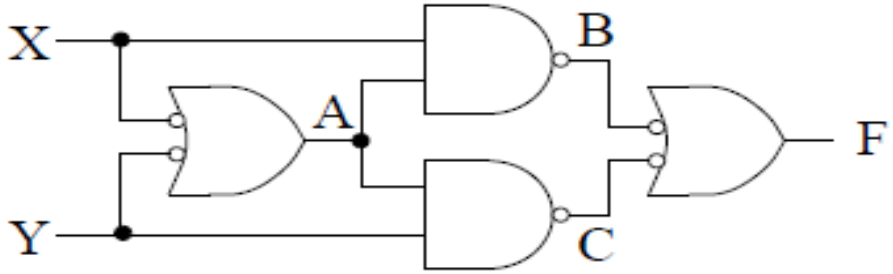
Inverted Inputs ☺



X	Y	A	B	C	F
0	0	1			
0	1	1			
1	0	1			
1	1	0			

Deriving Truth Table from Circuit

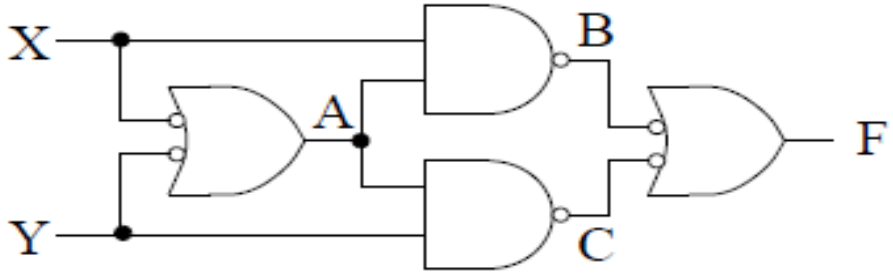
- Label intermediate gate outputs
- Fill in truth table in appropriate order



X	Y	A	B	C	F
0	0	1	1		
0	1	1	1		
1	0	1	0		
1	1	0	1		

Deriving Truth Table from Circuit

- Label intermediate gate outputs
- Fill in truth table in appropriate order

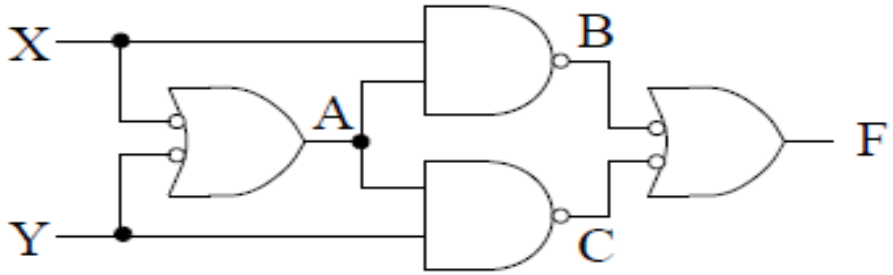


X	Y	A	B	C	F
0	0	1	1	1	0
0	1	1	1	0	1
1	0	1	0	1	1
1	1	0	1	1	0

1
5
-
1
-
5

Deriving Truth Table from Circuit

- Label intermediate gate outputs
- Fill in truth table in appropriate order

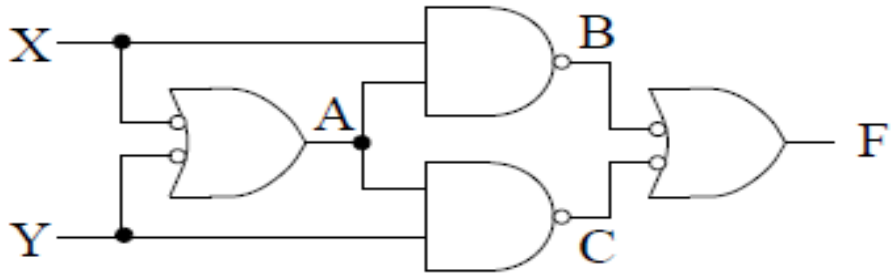


X	Y	A	B	C	F
0	0	1	1	1	0
0	1	1	1	0	1
1	0	1	0	1	1
1	1	0	1	1	0

What Gate is this

Deriving Truth Table from Circuit

- Label intermediate gate outputs
- Fill in truth table in appropriate order



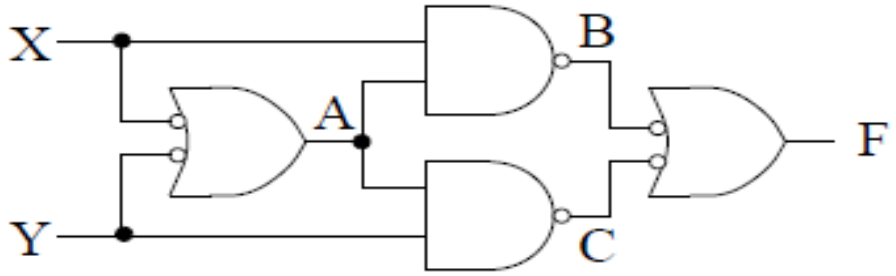
X	Y	A	B	C	F
0	0	1	1	1	0
0	1	1	1	0	1
1	0	1	0	1	1
1	1	0	1	1	0

What Gate is this

XOR : exclusively 1 input or the other Not Both

Deriving Truth Table from Circuit

- Label intermediate gate outputs
- Fill in truth table in appropriate order



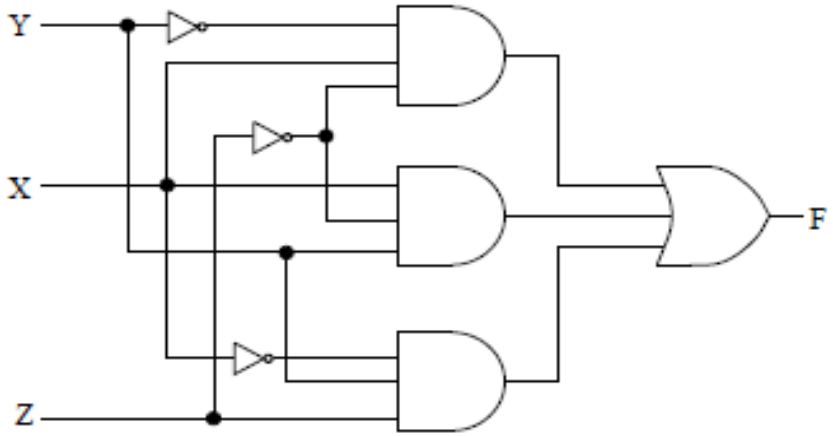
Note this is an alternative way of examining this circuit.
Both are correct.

This method you use
Intermediate gates
As AND gates
And then B,C values
Are double inverted
Which cancels out.

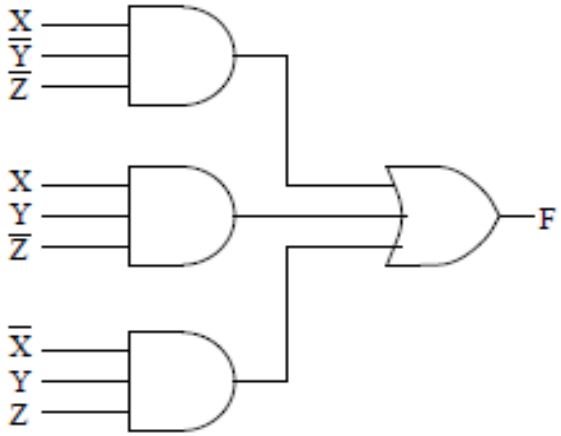
X	Y	A	B	C	F
0	0	1	0	0	0
0	1	1	0	1	1
1	0	1	1	0	1
1	1	0	0	0	0

Good Style in Circuit Drawing

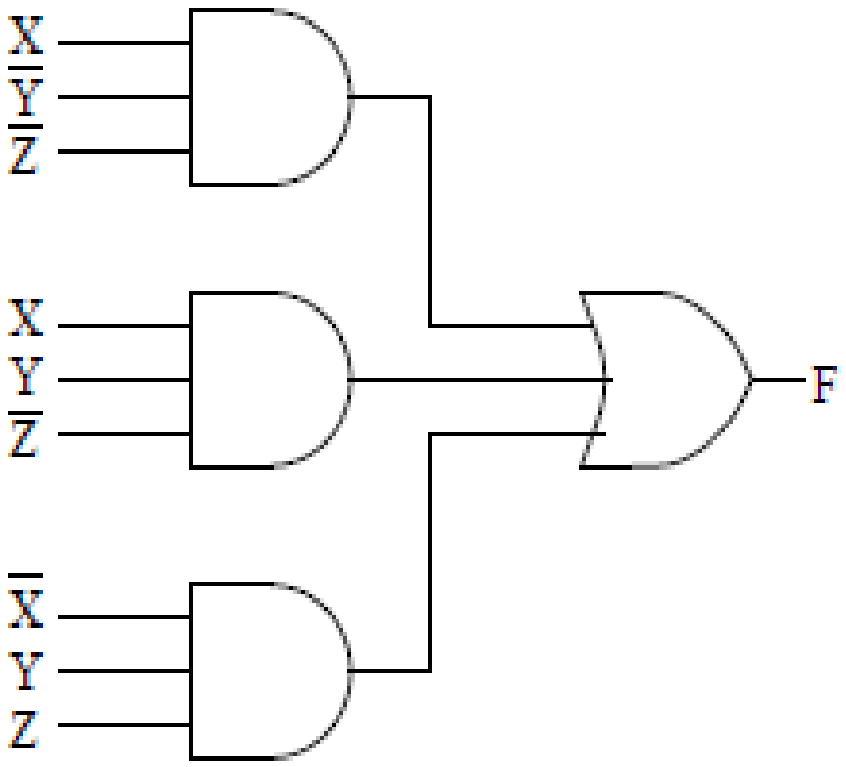
- Assume all literals (variables and their negations) are available
- Rectilinear wires, dots when wires split
- Do not draw spaghetti wires for inputs; instead, write each literal as needed



Bad

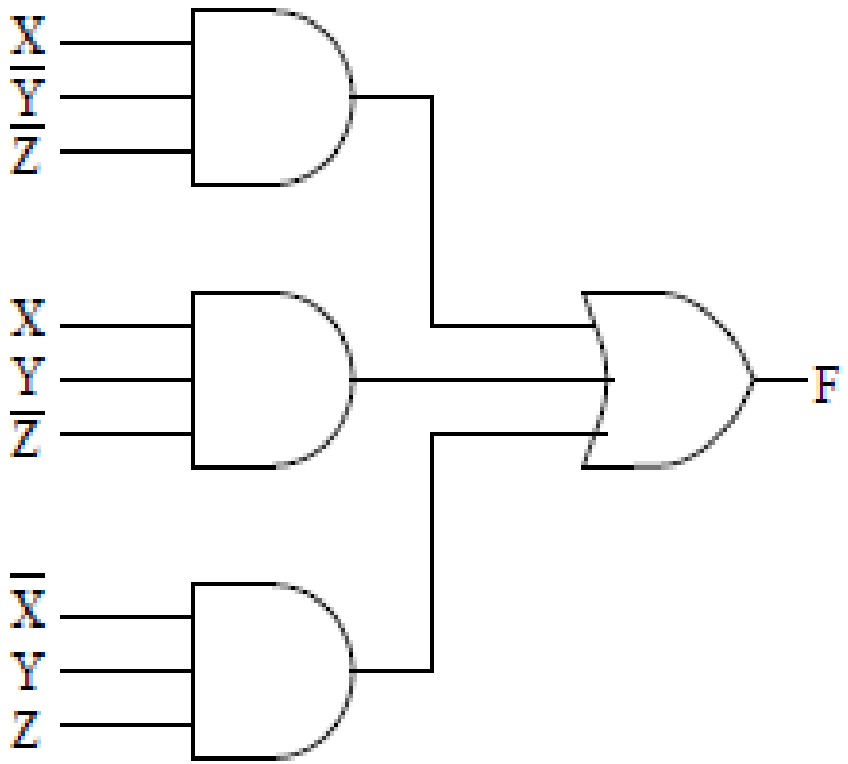


Good



Good

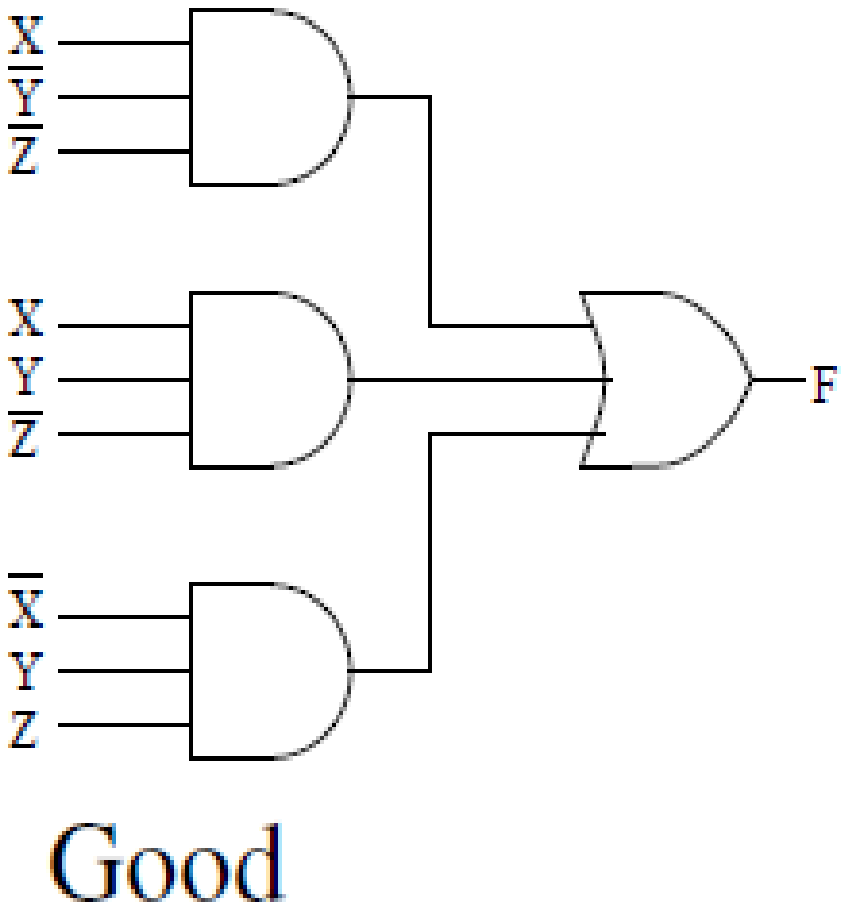
We can see clearly that
F = _____



Good

$$F = X\overline{Y}\overline{Z} + XY\overline{Z} + \overline{X}YZ$$

$$F = X\overline{Y}\overline{Z} + XY\overline{Z} + \overline{X}YZ$$



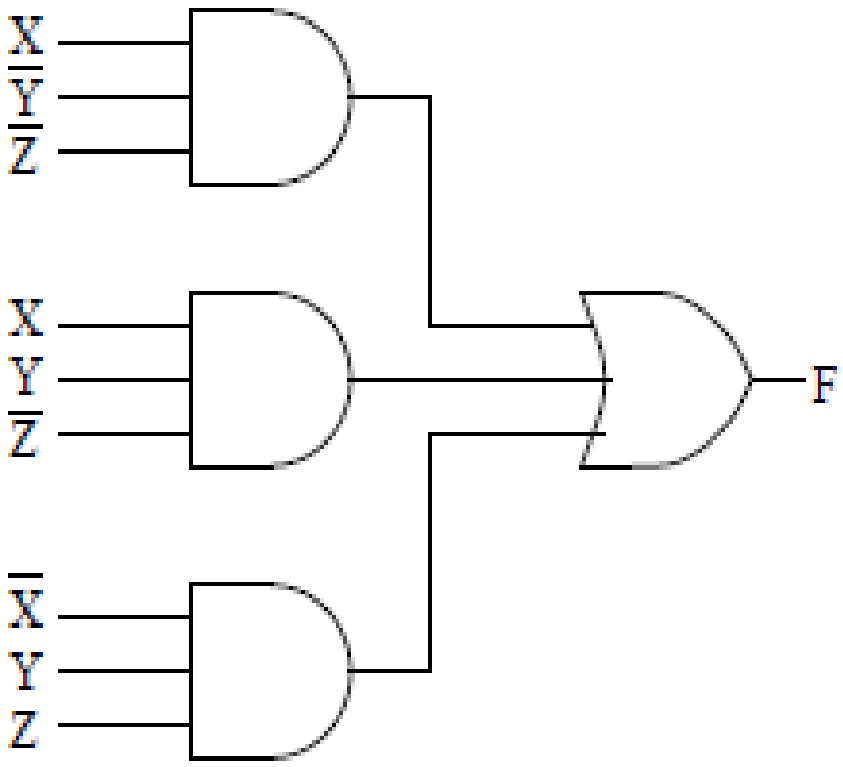
Good

Let us say:
Lowest binary value is Z. Highest binary value is X

Numerical Value of Input:

Therefore F is true when the input is either:

A) 2 , 4 or 6
B) 1 , 2 or 3
C) 3 , 6 or 4
D) NONE
E) 1,3 or 6



Good

Lowest Digit is X. Highest is Z

Numerical Value of Input:

Therefore F is true when the input is either:

The Output on F is essentially
Allowing only the values 3 ,4 or 6
To give True
Otherwise, any other value will
Be false.

1
5
-
1
-
5

Useful Components: Decoders

- n inputs, 2^n outputs (converts binary to "unary")
- Example: 3-to-8 (or 3-bit) decoder

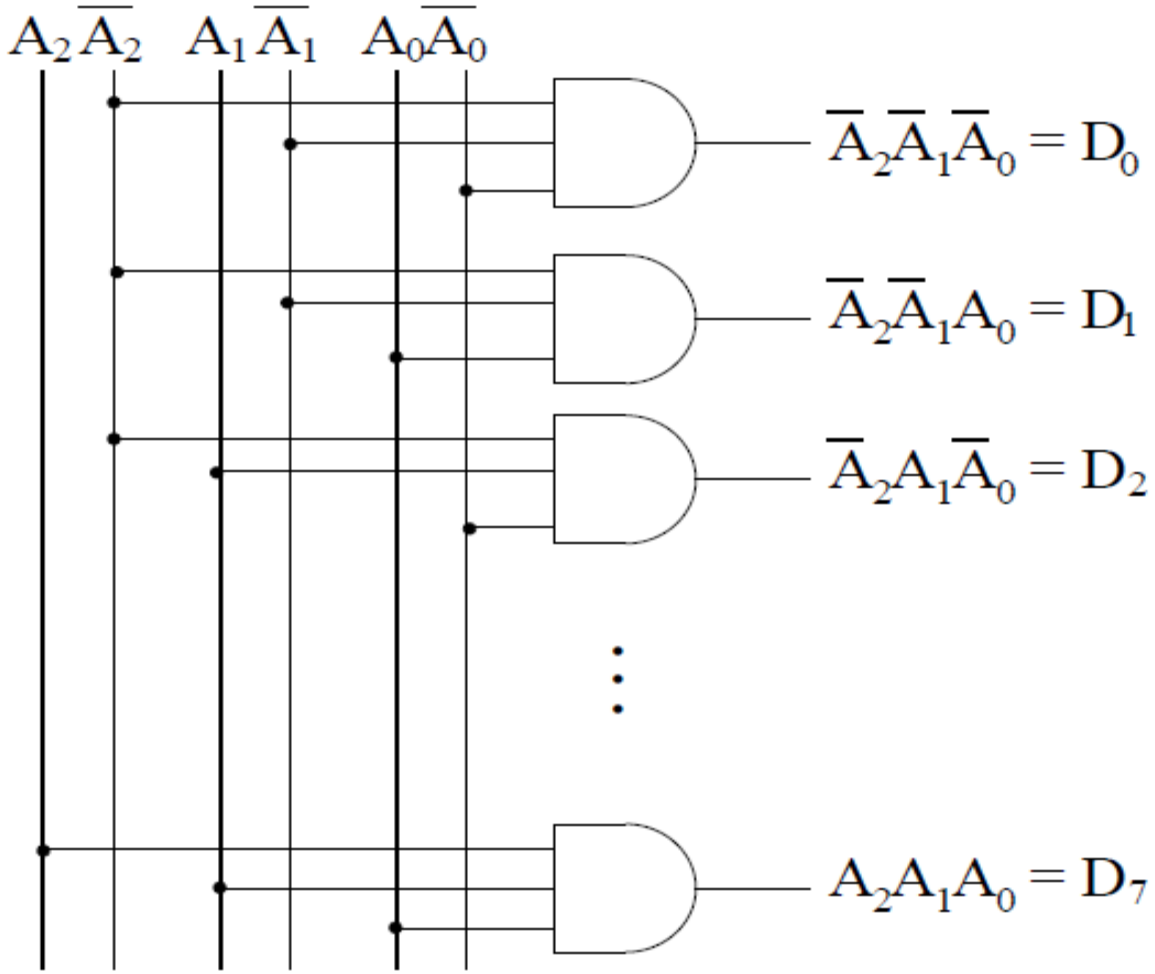
A_2	A_1	A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	1	0	0
0	1	1	0	0	0	0	1	0	0	0
1	0	0	0	0	0	1	0	0	0	0
1	0	1	0	0	1	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0

- Circuit has regular structure

Only One Output asserted for each Input combination

Decoder: Translates the n-bit input into Signal that corresponds to the Binary Value of the n-bit input

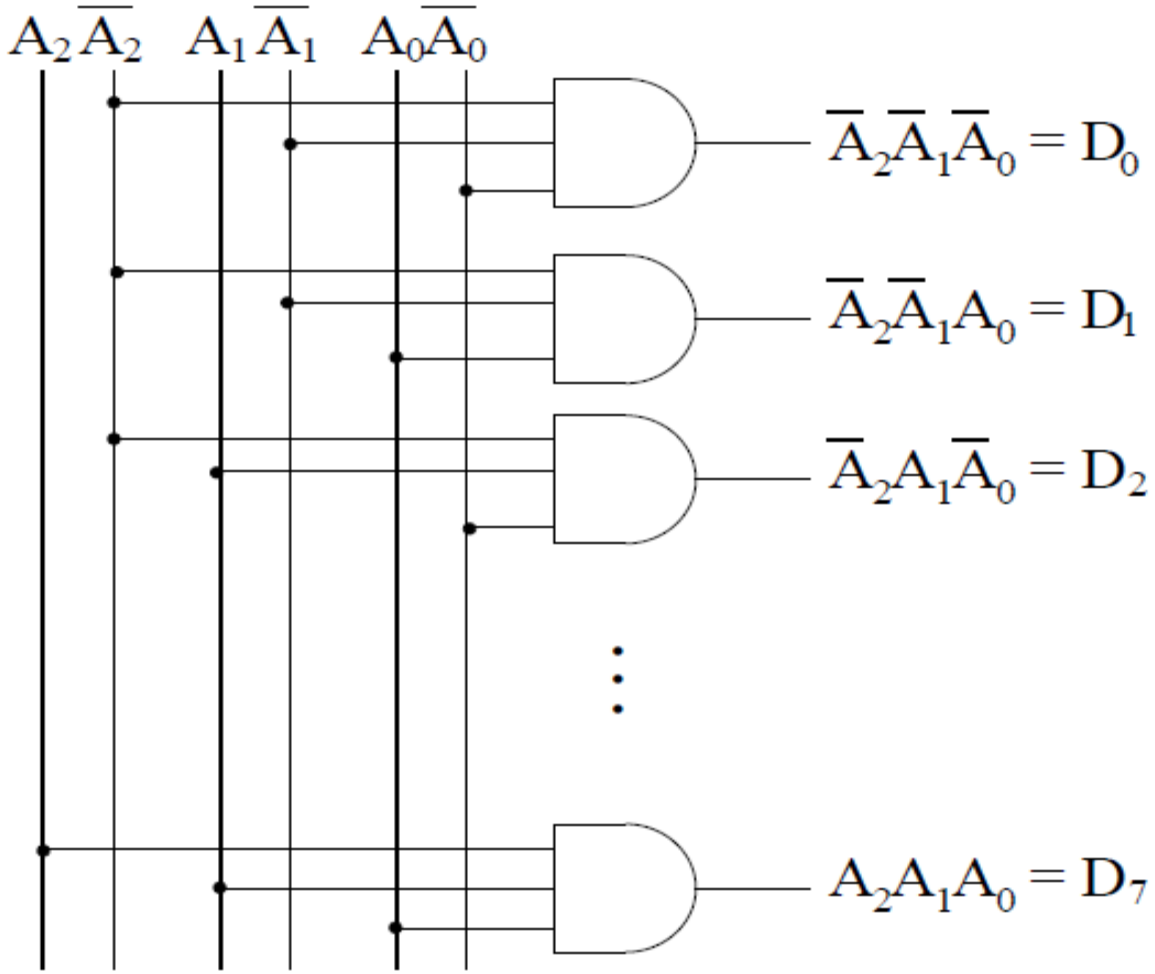
Decoder useful in building larger components



3-to-8 (or 3-bit) Decoder

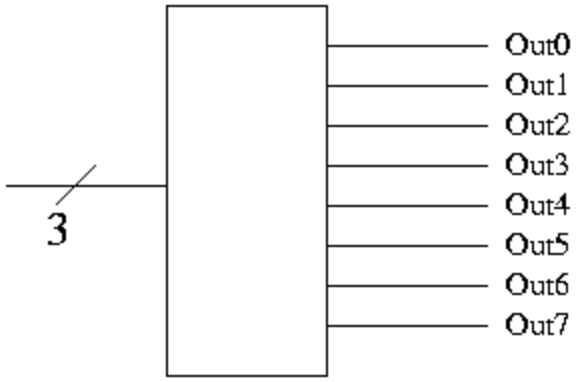
All Outputs are expressed in Terms of Minterms from the Truth Table.

Easy to see how AND gates implement Each of minterms

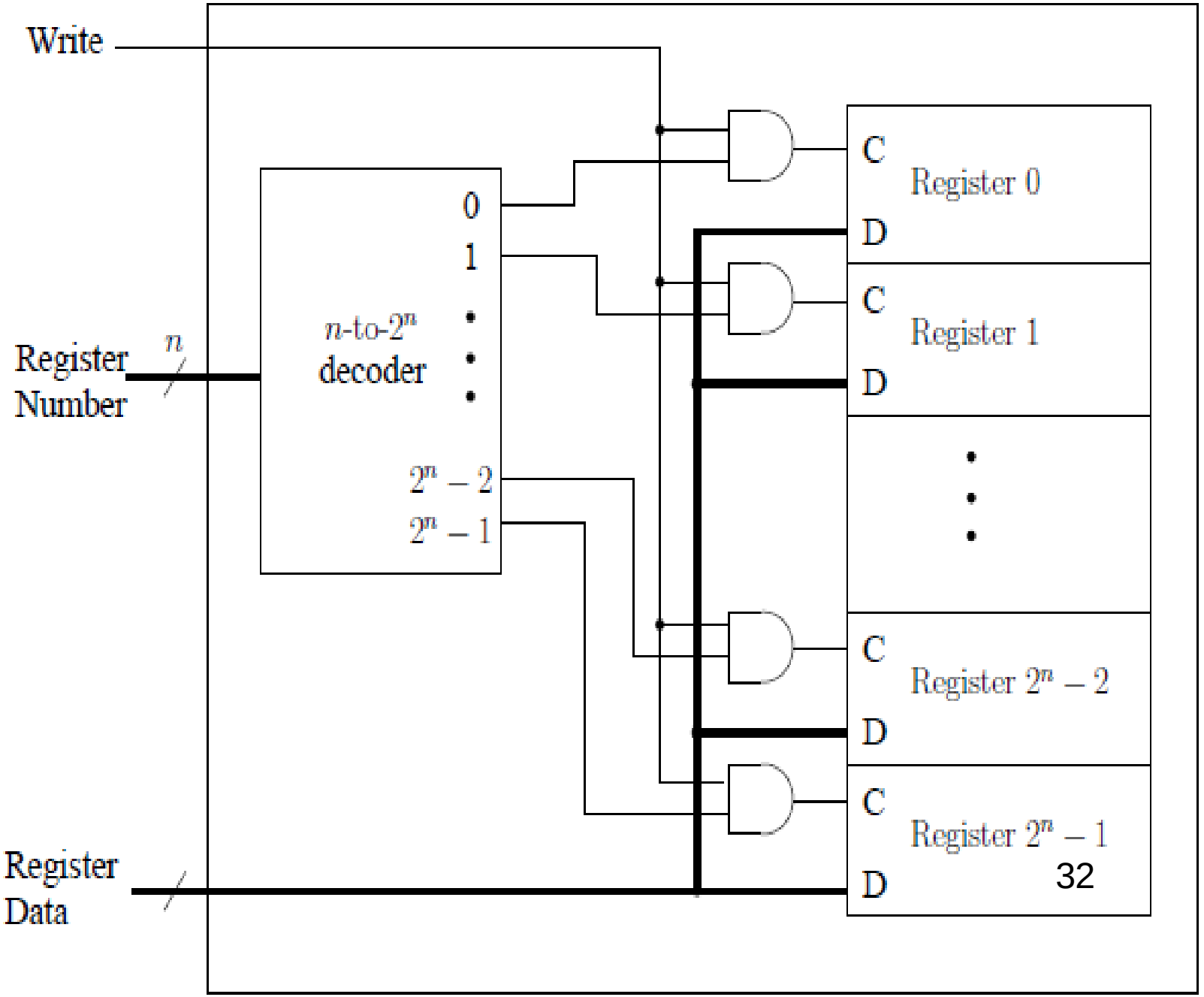


3-to-8 (or 3-bit) Decoder

Decoder



One output asserted at One time



Writing Data to a Register

Using a Decoder

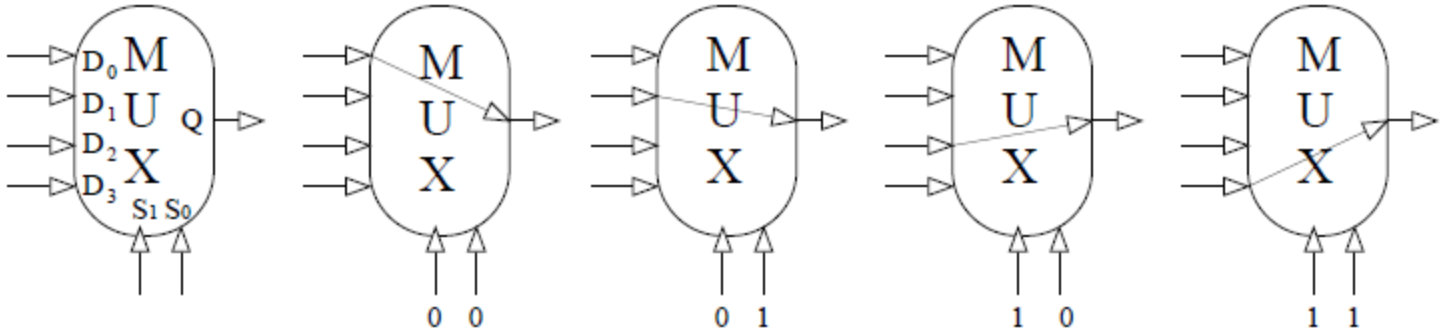
Multiplexors

- Inputs: 2^n lines $(D_0, \ldots, D_{2^n-1})$
  n select lines $(S_{n-1}, \ldots, S_0)$
- Output: The value of the D_S line
- Example: 4-1 Multiplexer

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

Also called a selector:

The Output is only ONE of the inputs
Which input gets through
Is decided by Control lines
Or Select lines.



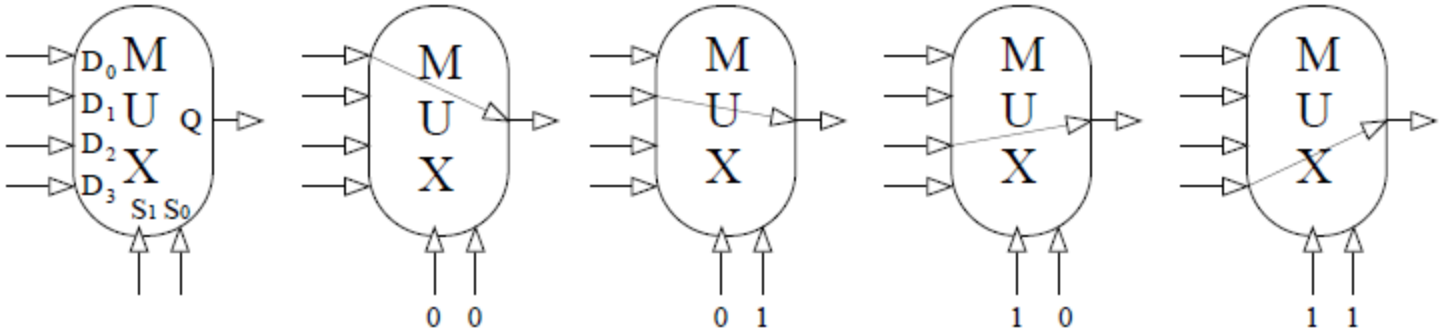
2 bit input line example

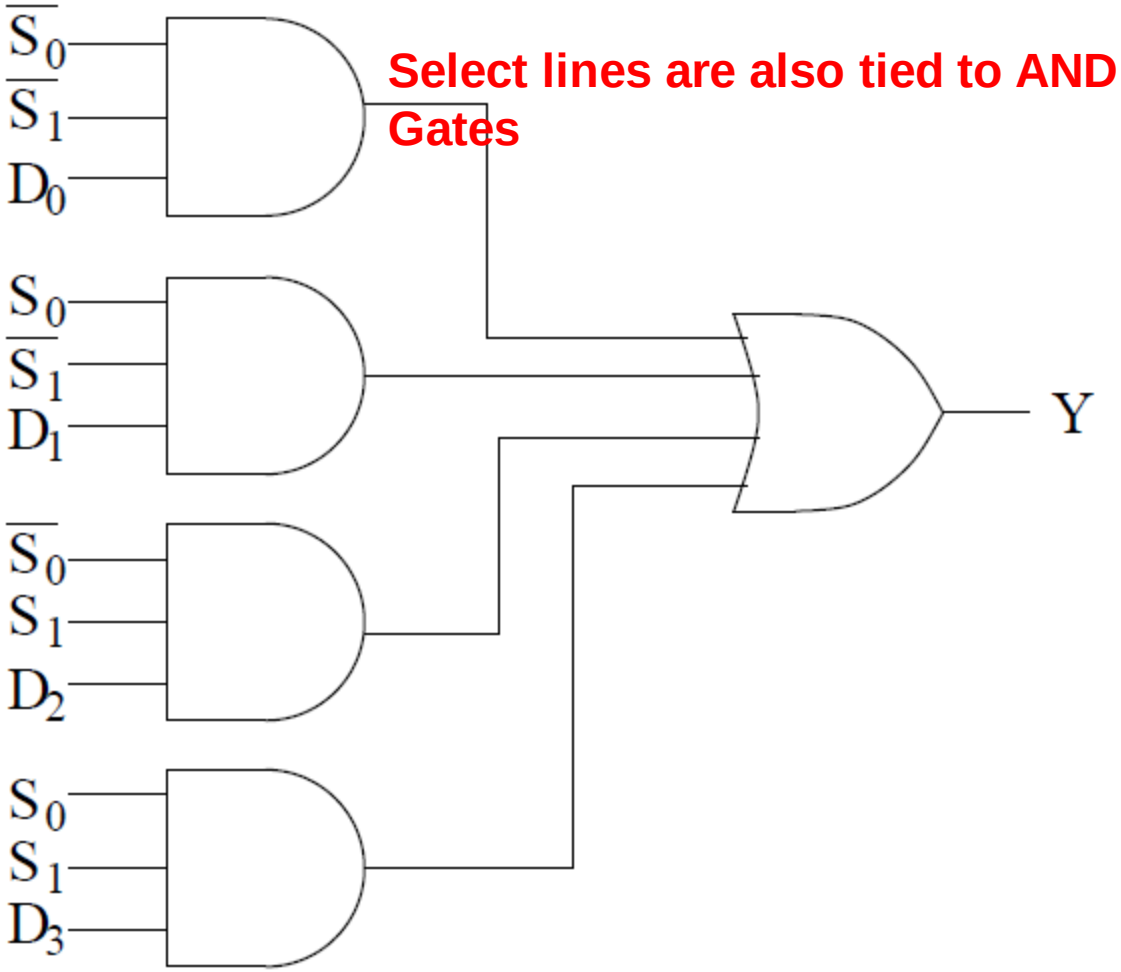
Multiplexors

- Inputs: 2^n lines $(D_0, \ldots, D_{2^n-1})$
  n select lines $(S_{n-1}, \ldots, S_0)$
- Output: The value of the D_S line
- Example: 4-1 Multiplexer

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

Internally:
Select Lines combine with
Input lines to give
Desired output





Select lines are also tied to AND Gates

4-1 Multiplexor

The selector lines

Enable one of D0 to D3 to be taken as output Y.

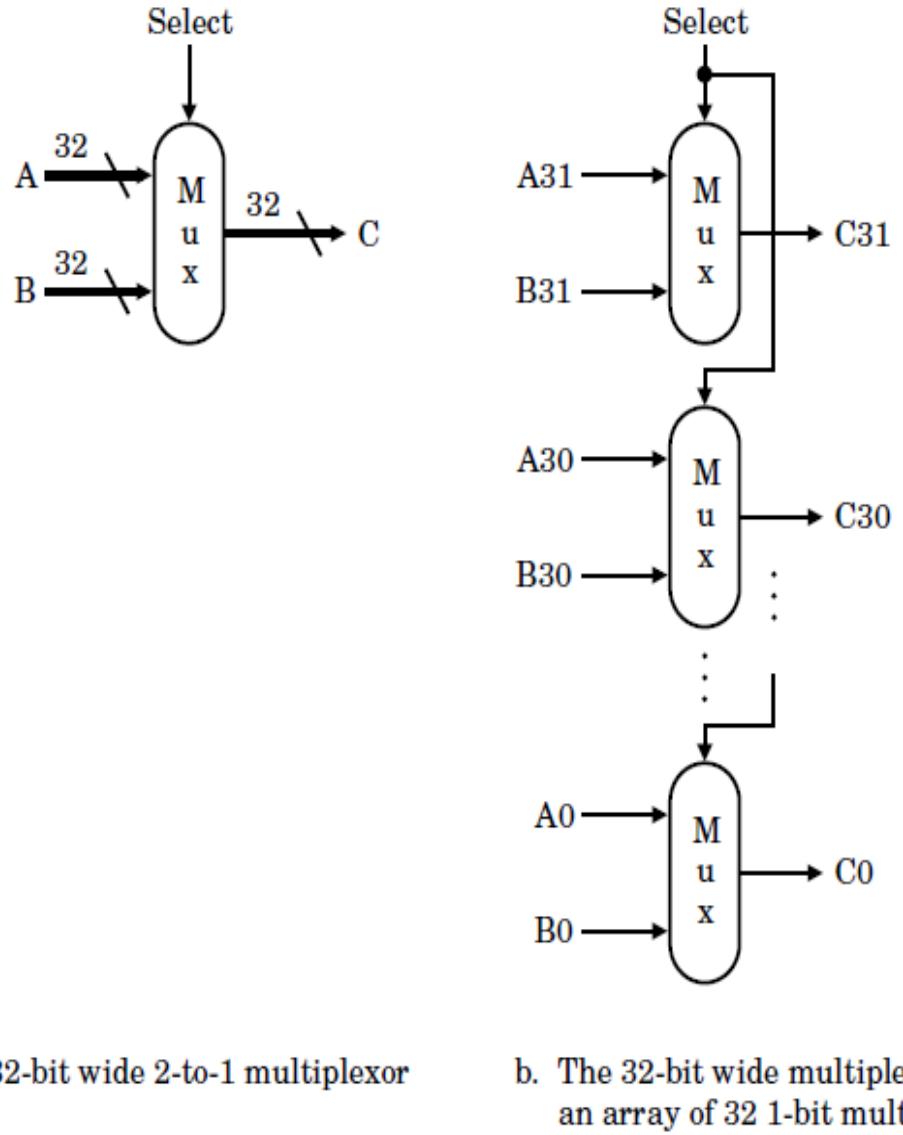
Application of a Multiplexor

2 Registers.
Selector lines decides which Register gets used

Slash indicates a 32 bit line Bus

Bus: Collection of data lines that are treated together as a single logical signal.

Multiplexor: Allows us to select one bus over another



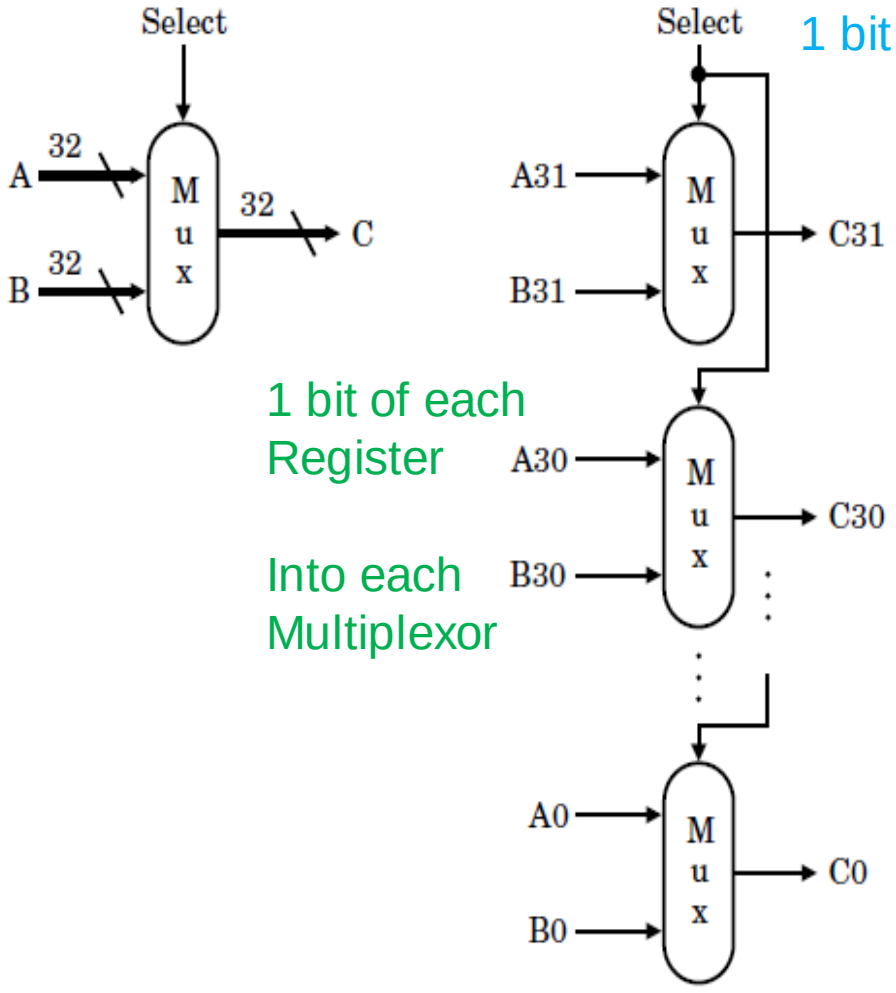
a. A 32-bit wide 2-to-1 multiplexor

b. The 32-bit wide multiplex is actually an array of 32 1-bit multiplexors

Slash indicates a 32 bit line Bus

Bus: Collection of data lines that are treated together as a single logical signal.

Multiplexor: Allows us to select one bus over another



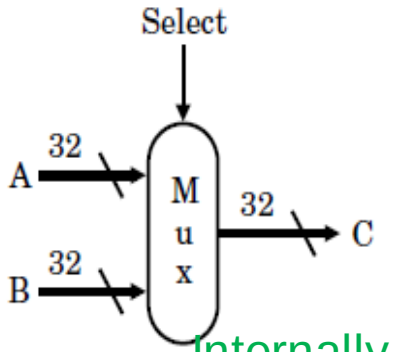
a. A 32-bit wide 2-to-1 multiplexor

b. The 32-bit wide multiplex is actually an array of 32 1-bit multiplexors

Slash indicates a 32 bit line Bus

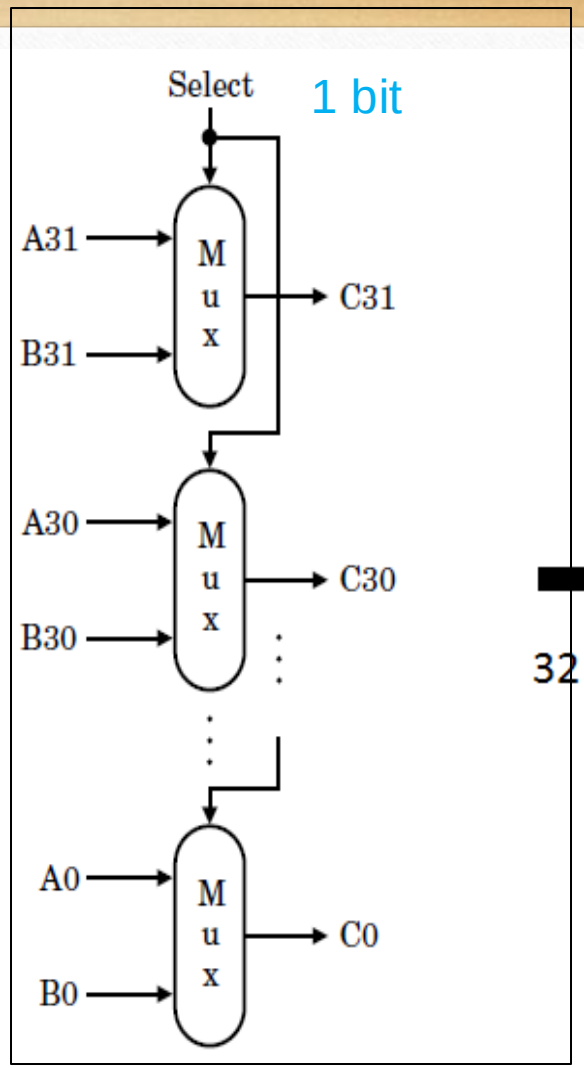
Bus: Collection of data lines that are treated together as a single logical signal.

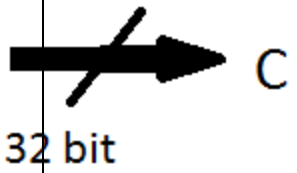
Multiplexor: Allows us to select one bus over another



Internally
1 bit of each
bus

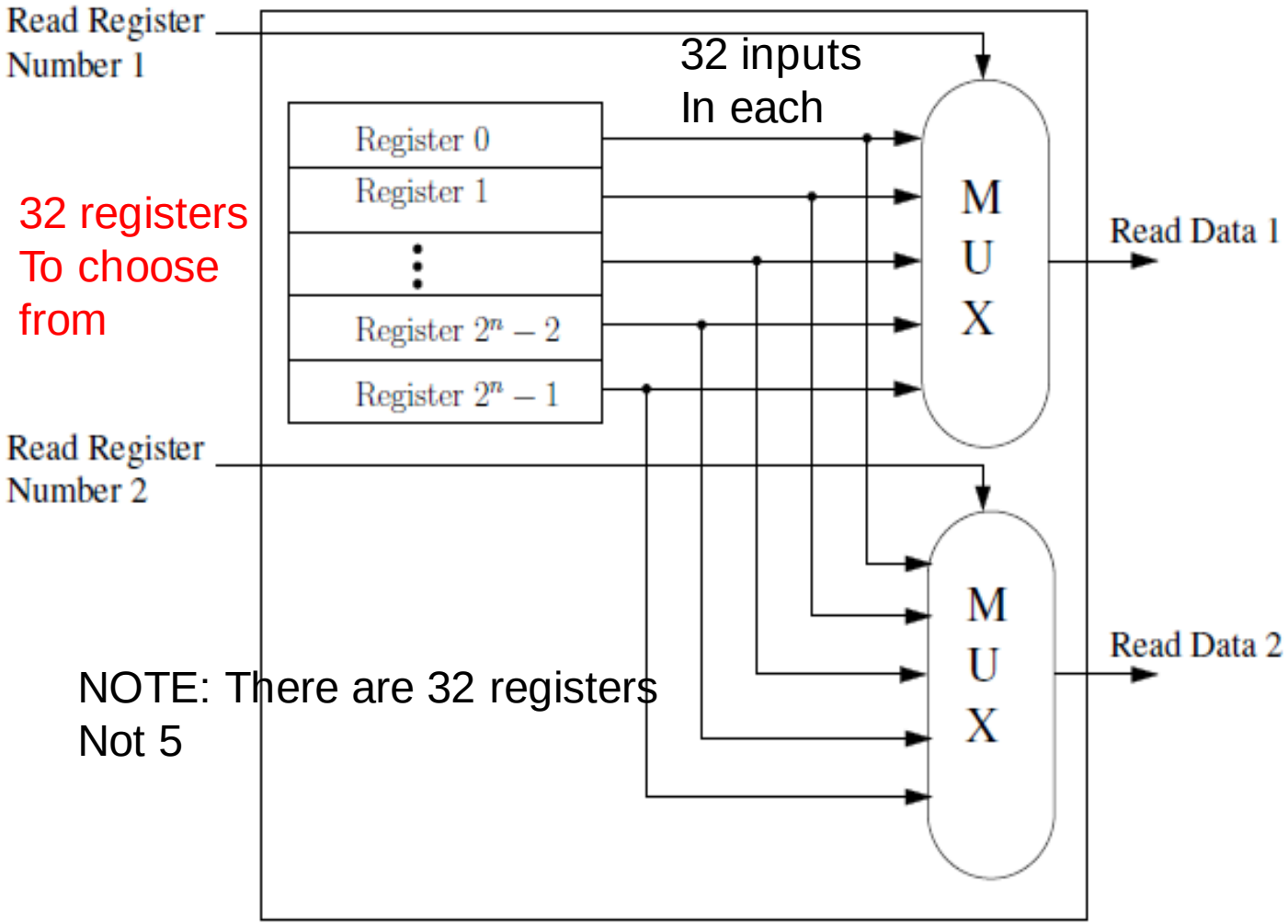
Into each
Multiplexor





a. A 32-bit wide 2-to-1 multiplexor

b. The 32-bit wide multiplex is actually an array of 32 1-bit multiplexors



32 registers
To choose from

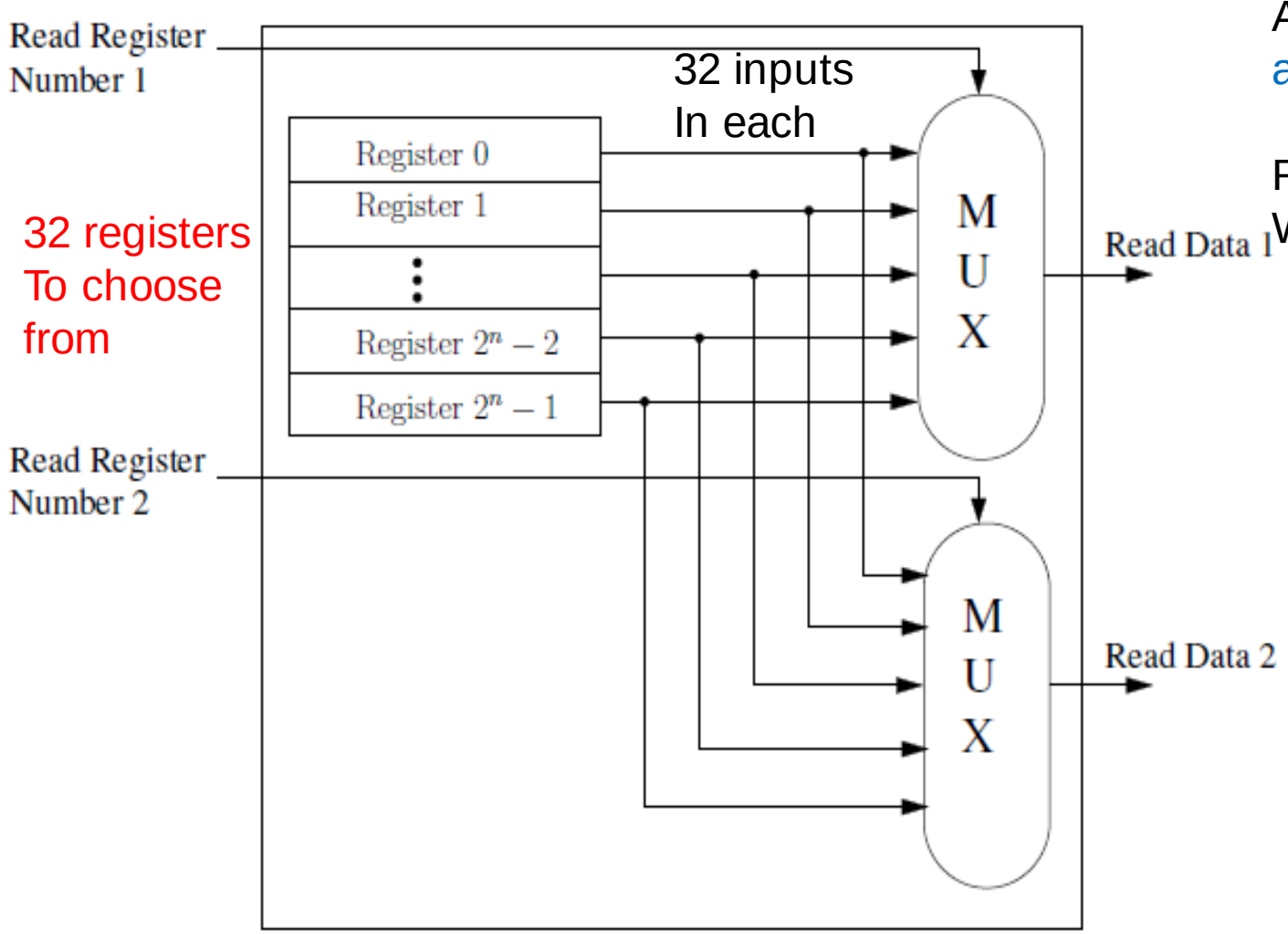
NOTE: There are 32 registers
Not 5

Add instruction:
add $1, $2 , $3

Read from two registers
Write to One Register

Which register do we Read from: Depends on Selector lines

How many bits on Selector lines?: **A) 2** **B) 3** **C) 5** **D) 4** **E) 32**

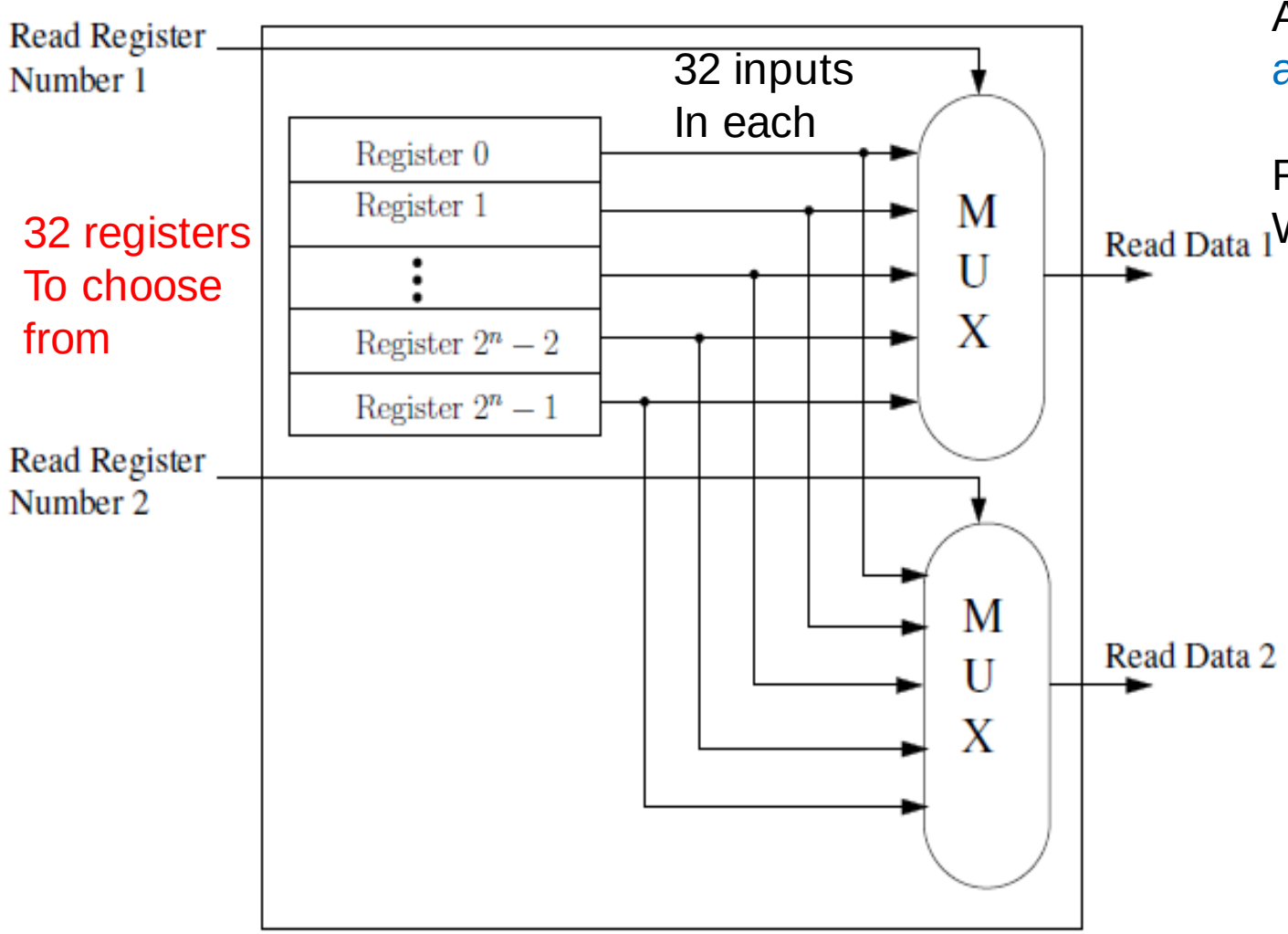


Add instruction:
add $1, $2 , $3

Read from two registers
Write to One Register

Which register do we Read from: Depends on Selector lines

How many bits on Selector lines?: **A) 2** **B) 3** **C) 5*** **D) 4** **E) 32**



Add instruction:
add $1, $2 , $3

Read from two registers
Write to One Register

Which register do we Read from: Depends on Selector lines

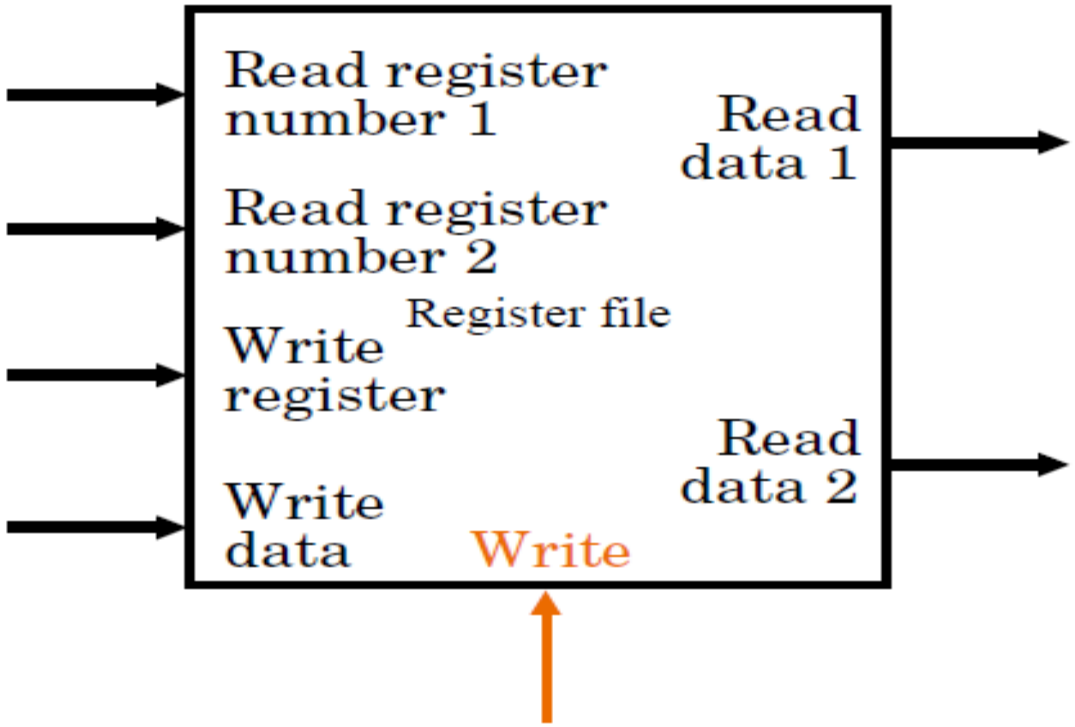
Using a Multiplexor to Do More:

Use A Multiplexor to implement a Boolean Function

Example Done in Class

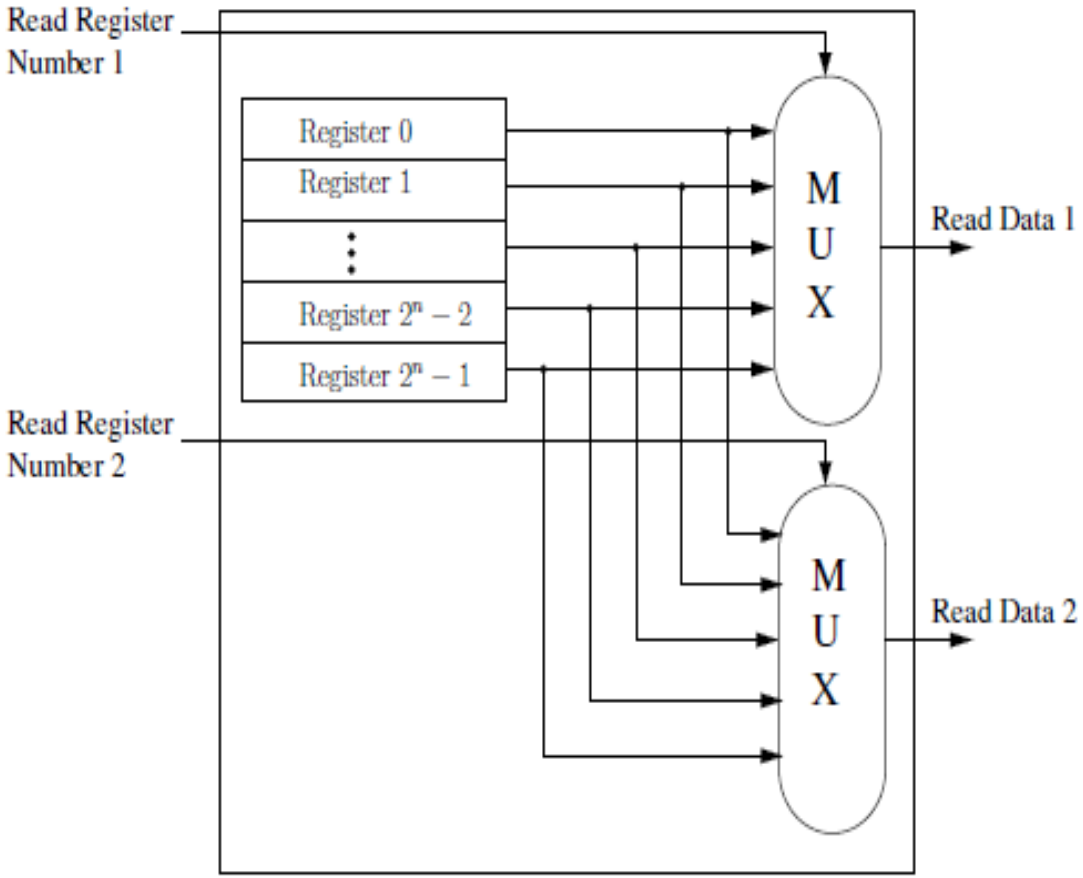
Registers and Register Files

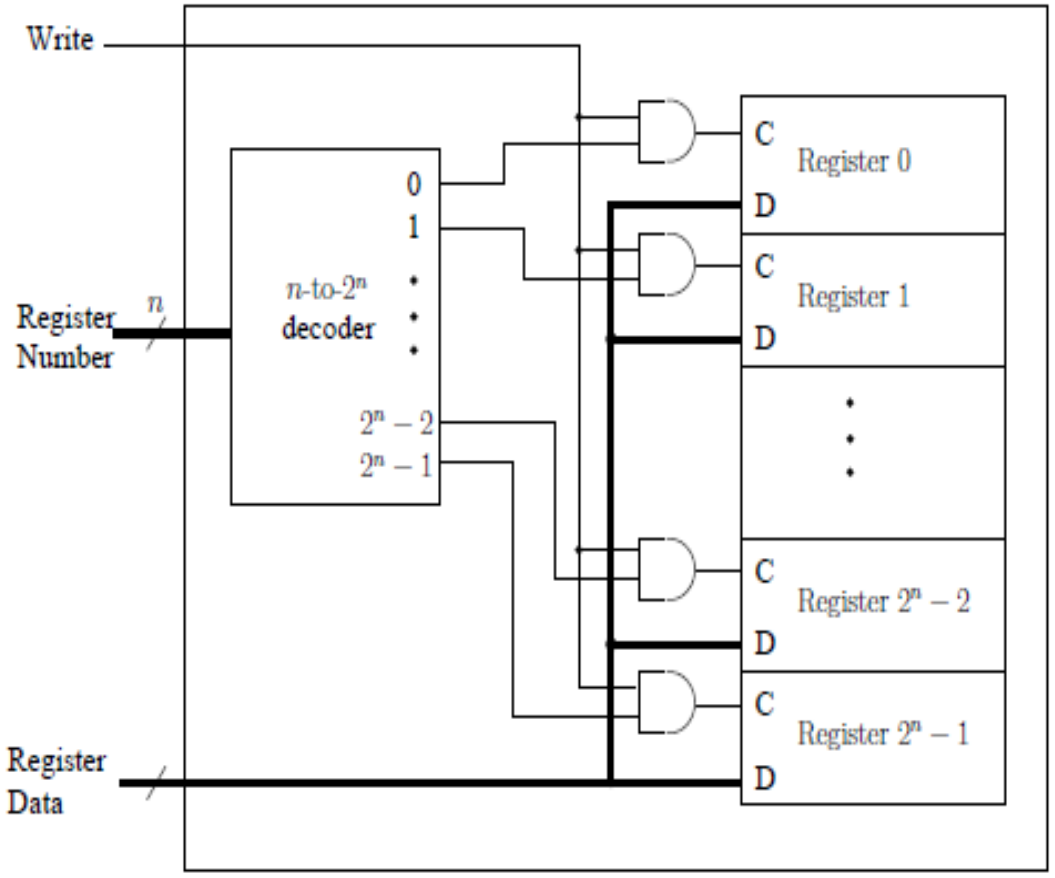
- Register: an array of flip-flops (e.g. 32 for a word register)
- Register file: a way of organizing registers



Register File is the means by which we access all 32 Registers

Read/Write Logic for Register File





How many bits on Read Register Num1 and Num2?

How many bits, Write Register Data?