Simplification of Combinatorial Digital Logic

CS 64: Computer Organization and Design Logic

Lecture #13

Fall 2019

Ziad Matni, Ph.D.

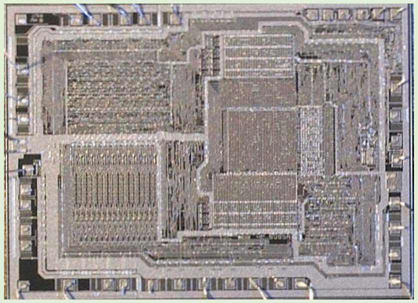
Dept. of Computer Science, UCSB

This Week on
"Didja Know Dat?!"

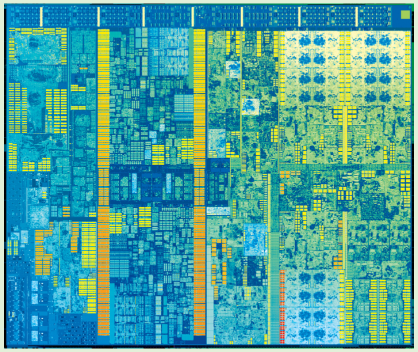
One of the first commercially available micro-processors (CPUs) was Intel's 8008 in the early 1970s and its follow-up the 8080 (1976).

The 8080 is STILL in production!

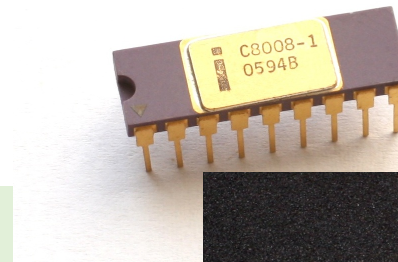
The size of the 8080 semiconductor (i.e. inside the ceramic package with all the pins) is 4.2 x 4.8 mm^2 (20.16 mm^2)

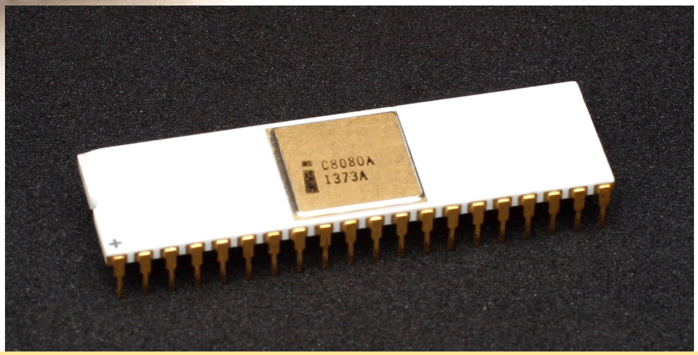


By comparison, the Intel i7 Dual Core is 9.2 x 13.5 mm^2 (124.2 mm^2)



BUT THAT'S NOT A FAIR COMPARISON!





8088: 16-bit CPU.
Cannot run Windows 1.0!

i7 Core: 64-bit CPUs.
Have a TON more features.

How do they do that?

Moore's Law

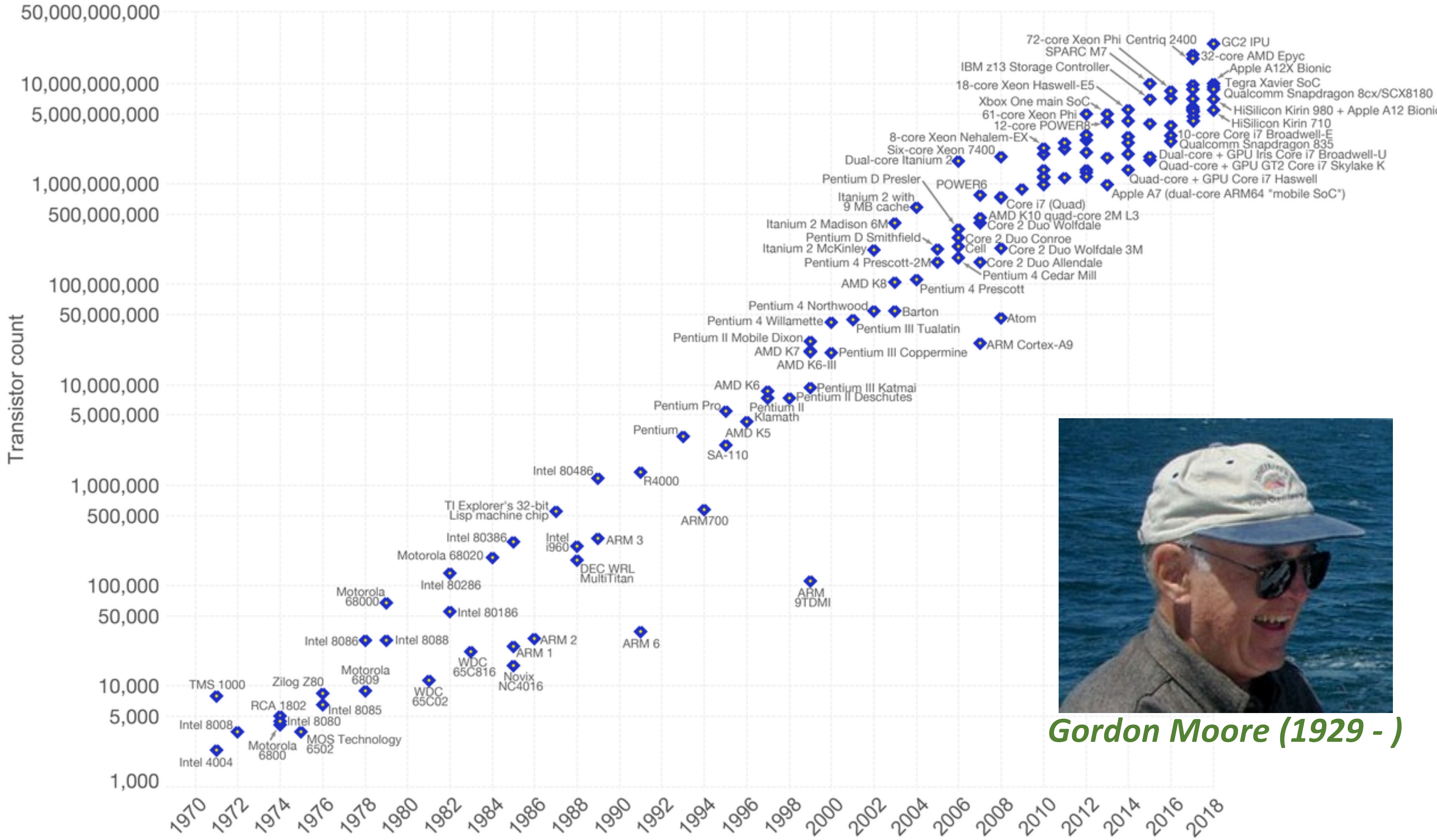
The number of *transistors* on a microchip doubles every two years, though the cost of computers is halved.

More transistors means higher capabilities. *Smaller transistors* mean higher speeds.

Moore's Law – The number of transistors on integrated circuit chips (1971-2018)

Our World in Data

Moore's law describes the empirical regularity that the number of transistors on integrated circuits doubles approximately every two years. This advancement is important as other aspects of technological progress – such as processing speed or the price of electronic products – are linked to Moore's law.



Gordon Moore (1929 -)

Administrative

- Lab 6 due Wednesday
- You have 3 more labs after this…
- Midterm Exam Grades are On GauchoSpace

Reviewing Your Midterm Exams

- You can review your midterm with a TA during office hours
  - *Last name*: **A thru M** **Kunlong Liu** **Tu 5 pm – 7 pm**
  - *Last name*: **N thru Z** **Charlie Uslu** **Tu 3 pm – 5 pm**
  - If you can't go to these o/hs, you can see me instead, but let me know ***many days ahead of time*** first so I can get your exam from the TA…

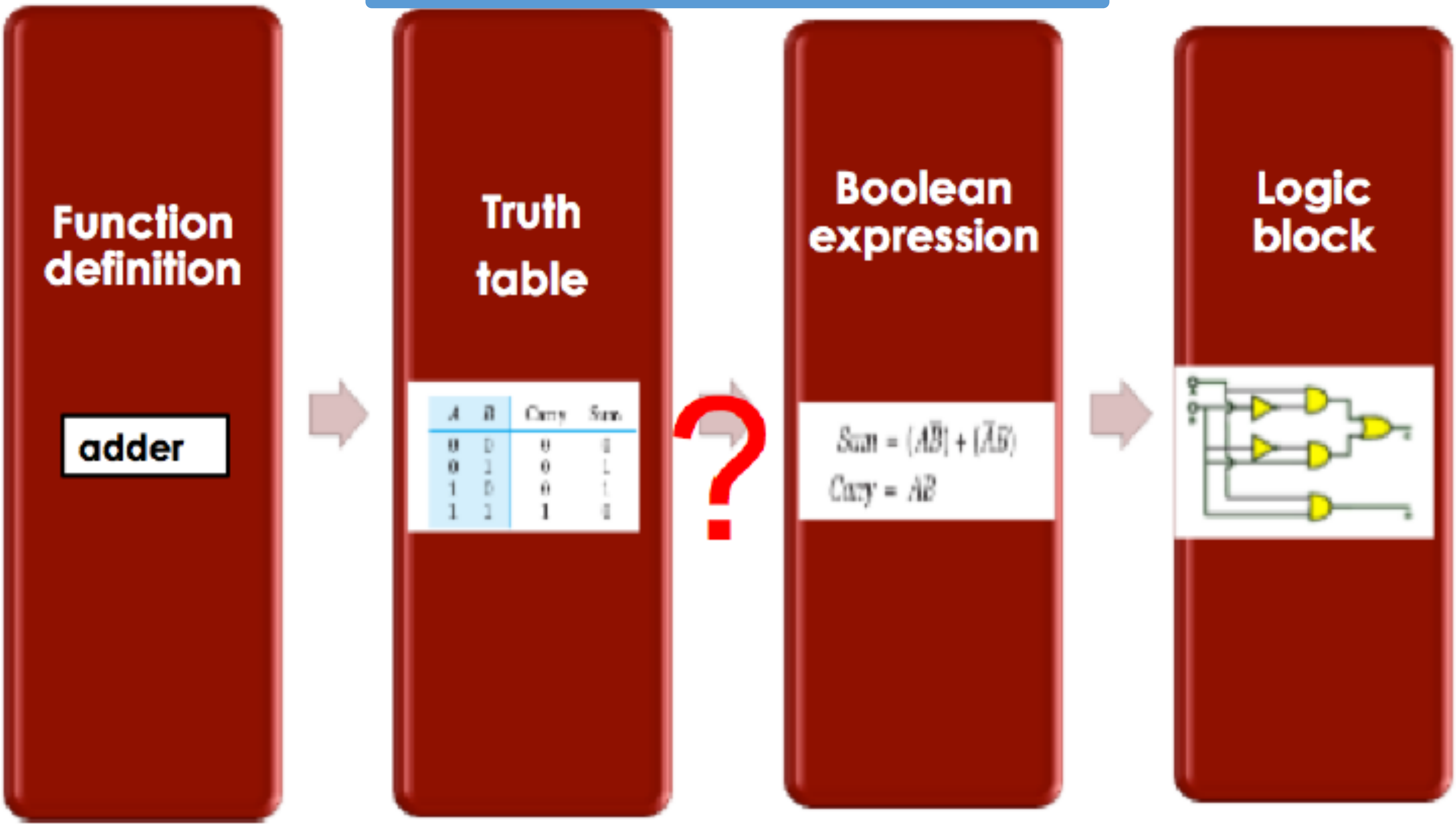
- When reviewing your exams:
  - Do **not** take pictures, do not copy the questions
  - TA cannot change your grade
    - If you have a legitimate case for grade change, the prof. will decide
    - Legitimate = When we graded, we added the total points wrong
    - Not legitimate = Why did you take off *N* points on this question????

Lecture Outline

- Simplifying Binary Functions using **Karnaugh Maps**
- **Multiplexers**

Digital Circuit Design Process

CAN THIS PROCESS BE REVERSED?



Important: Laws of Binary Logic

Circuit Equivalence - each law has 2 forms that are duals of each other.

Name	AND form	OR form
Identity law	$1A = A$	$0 + A = A$
Null law	$0A = 0$	$1 + A = 1$
Idempotent law	$AA = A$	$A + A = A$
Inverse law	$A\overline{A} = 0$	$A + \overline{A} = 1$
Commutative law	$AB = BA$	$A + B = B + A$
Associative law	$(AB)C = A(BC)$	$(A + B) + C = A + (B + C)$
Distributive law	$A + BC = (A + B)(A + C)$	$A(B + C) = AB + AC$
Absorption law	$A(A + B) = A$	$A + AB = A$
De Morgan's law	$\overline{AB} = \overline{A} + \overline{B}$	$\overline{A + B} = \overline{A}\,\overline{B}$

More Simplification Examples

Simplify the Boolean expression:

- **(A+B+C).!(D+E) + (A+B+C).(D+E)**

Simplify the Boolean expression and write it out on a truth table as proof

- **X.Z + Z.(!X+ X.Y)**

Use DeMorgan’s Theorm to re-write the expression below using at least one OR operation

- **NOT(X + Y.Z)**

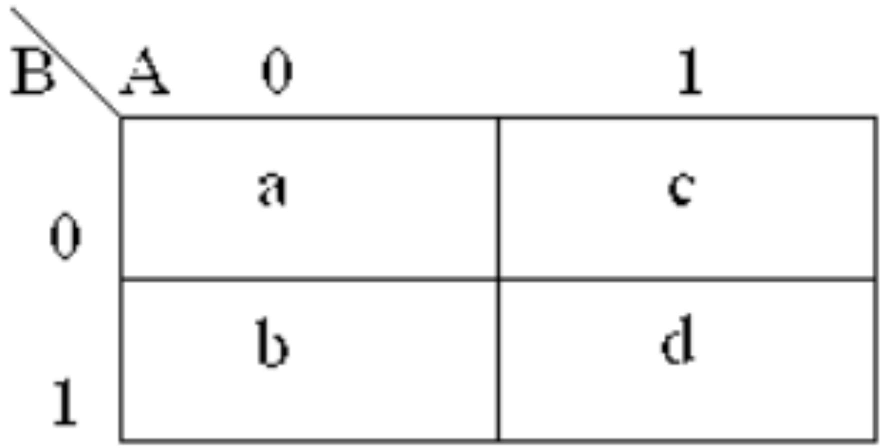
Scaling Up Simplification

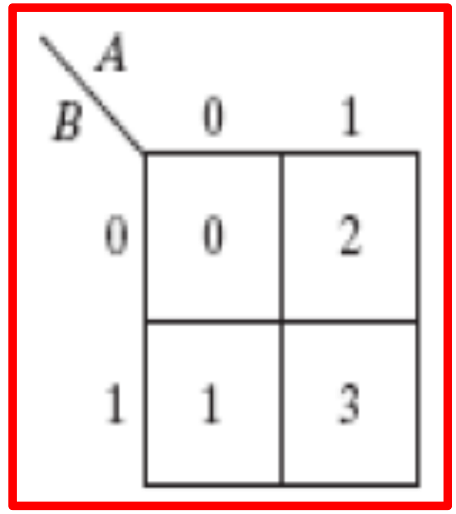
- When we get to *more* than 3 variables, it becomes challenging to use truth tables

- We can instead use ***Karnaugh Maps*** to make it immediately apparent as to what can be simplified

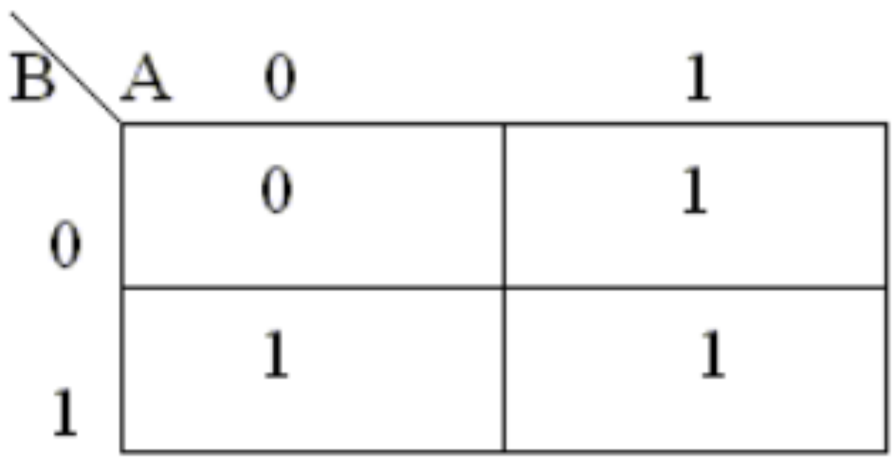
Example of a K-Map

	A	B	f(A,B)
0	0	0	a
1	0	1	b
2	1	0	c
3	1	1	d

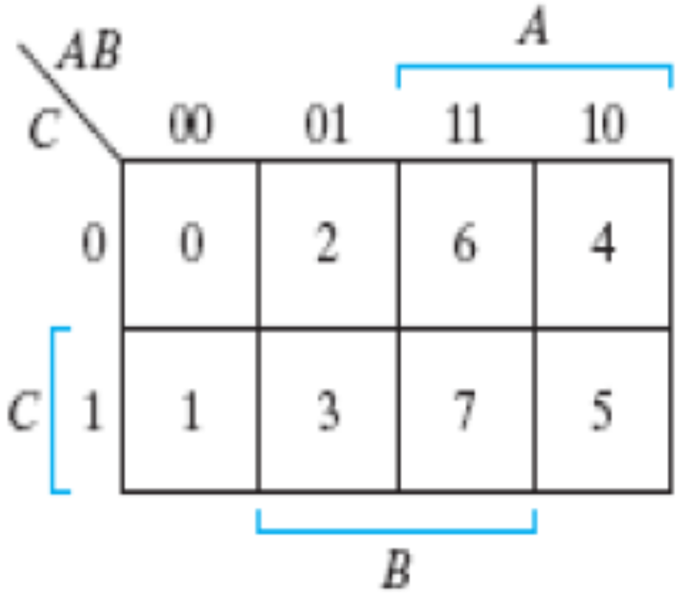


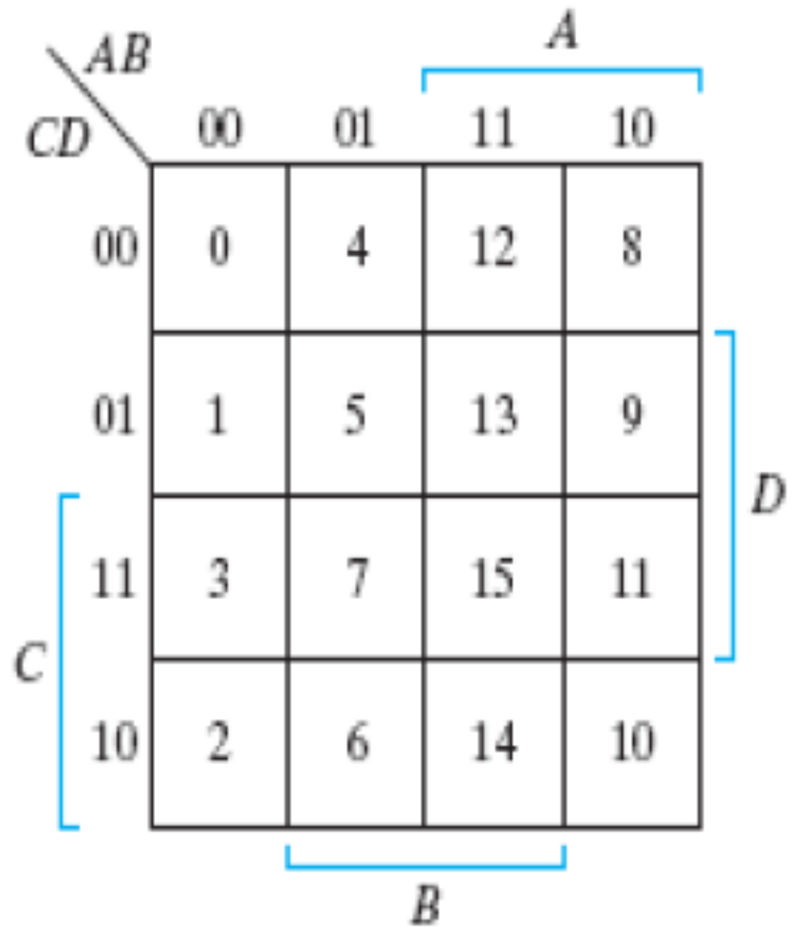


A	B	f(A,B)
0	0	0
0	1	1
1	0	1
1	1	1



K-Maps with 3 or 4 Variables





Note the adjacent placement of:

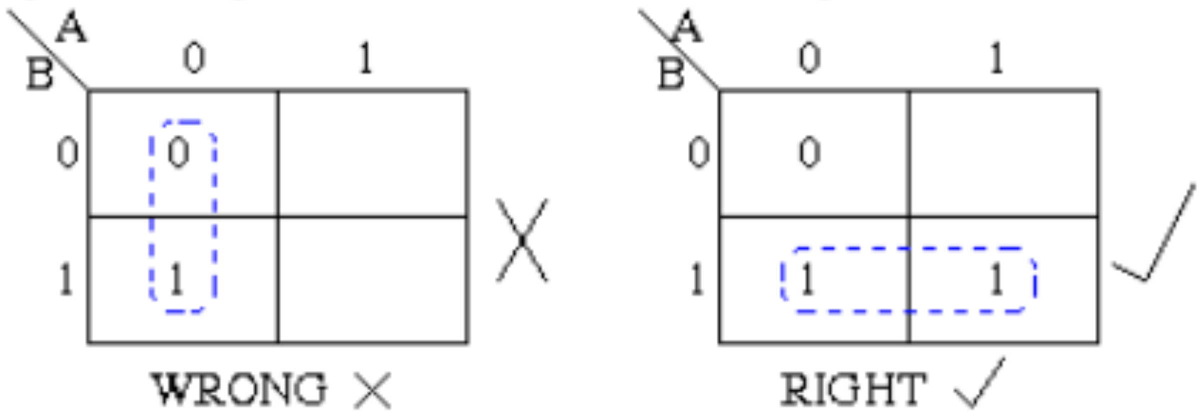
00 01 11 10

It's NOT:

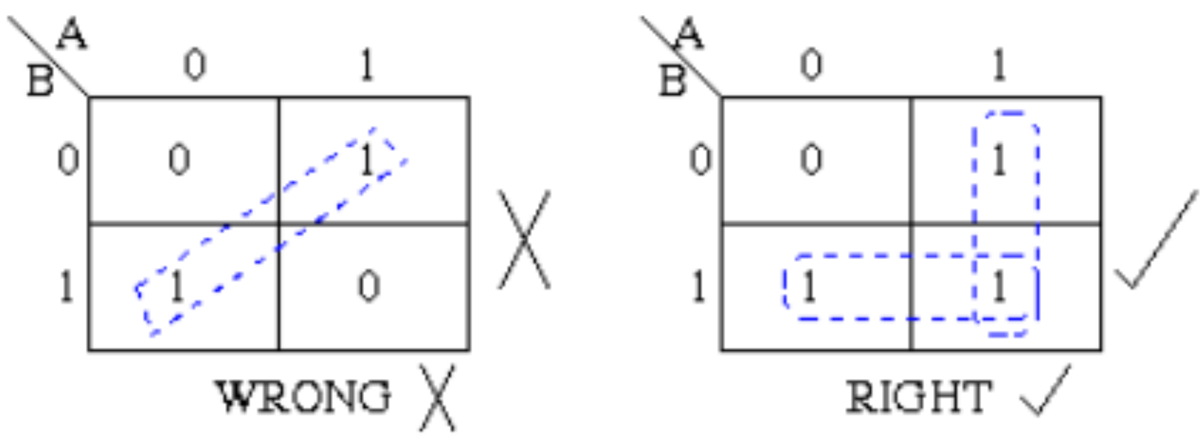
00 01 10 11

Rules for Using K-Maps for Simplification

1. Group together **adjacent cells** containing “1”
2. Groups should **not include** anything containing “0”

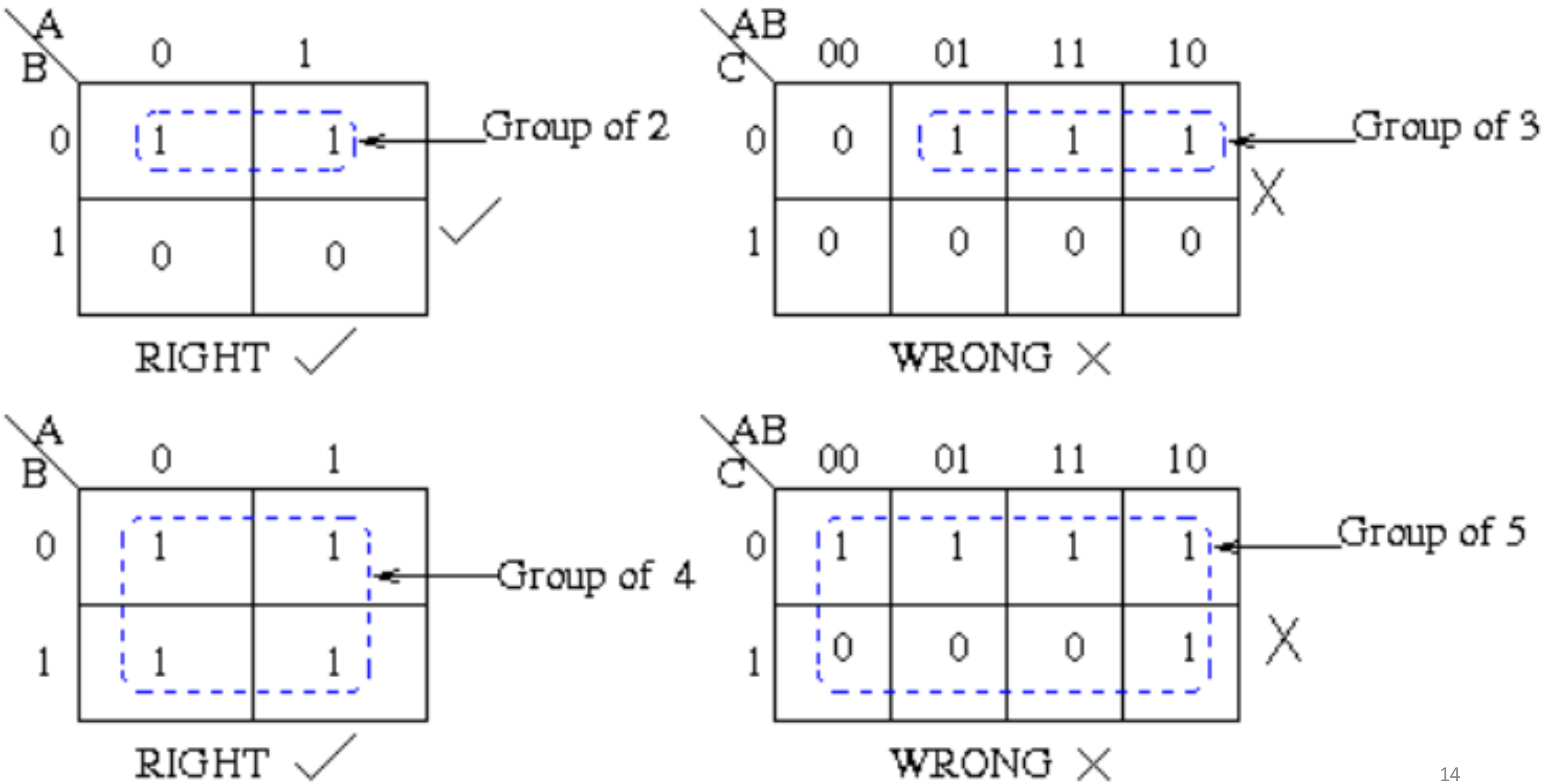


3. Groups may be horizontal or vertical, but **not diagonal**



Rules for Using
K-Maps for Simplification

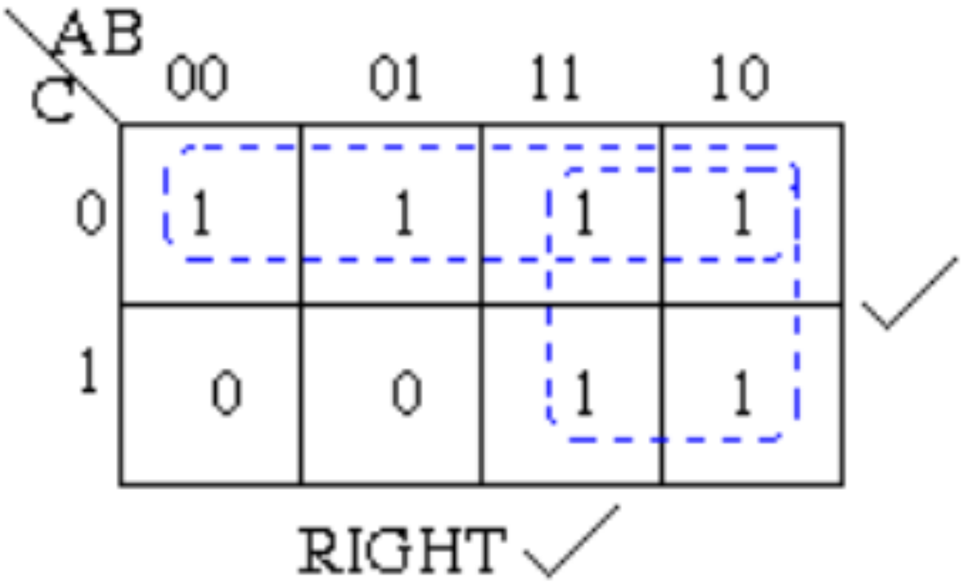
4. Groups must contain 1, 2, 4, 8, or in general 2^n cells.

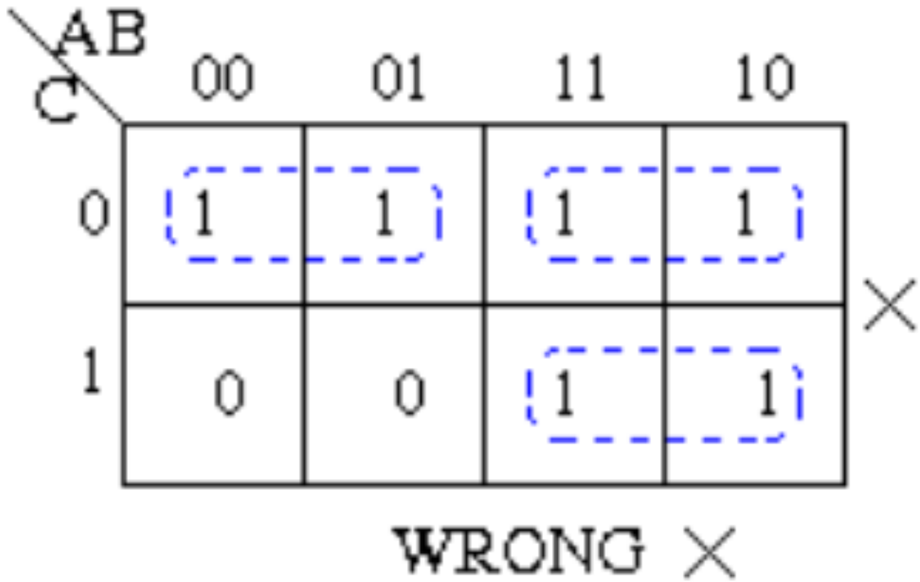


Rules for Using K-Maps for Simplification

5. Each group must be as large as possible

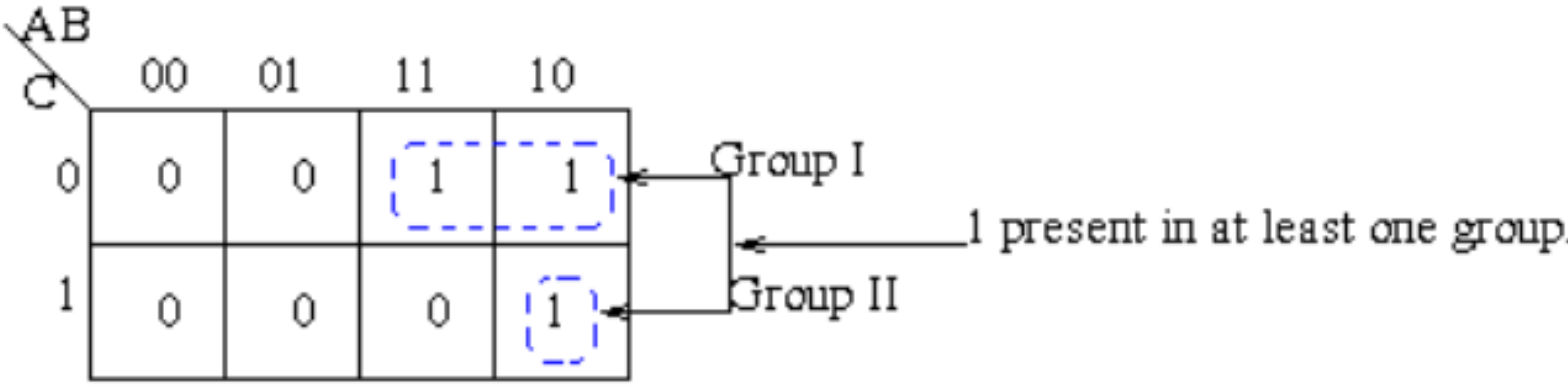
(Otherwise we're not being as minimal as we can be, even though we're not breaking any Boolean rules)





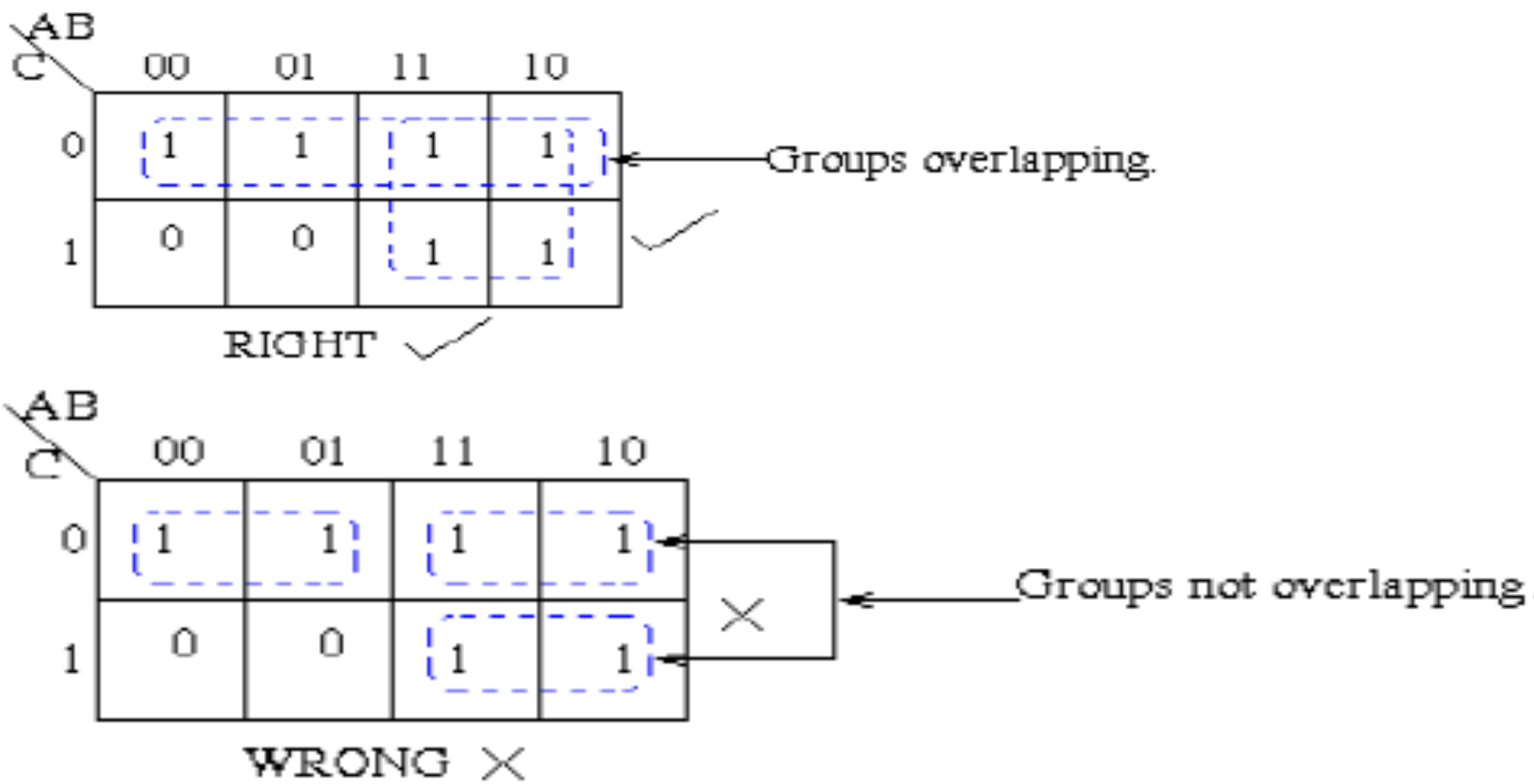
Rules for Using K-Maps for Simplification

6. **Each cell containing a "1" must be at least in one group**



Rules for Using K-Maps for Simplification

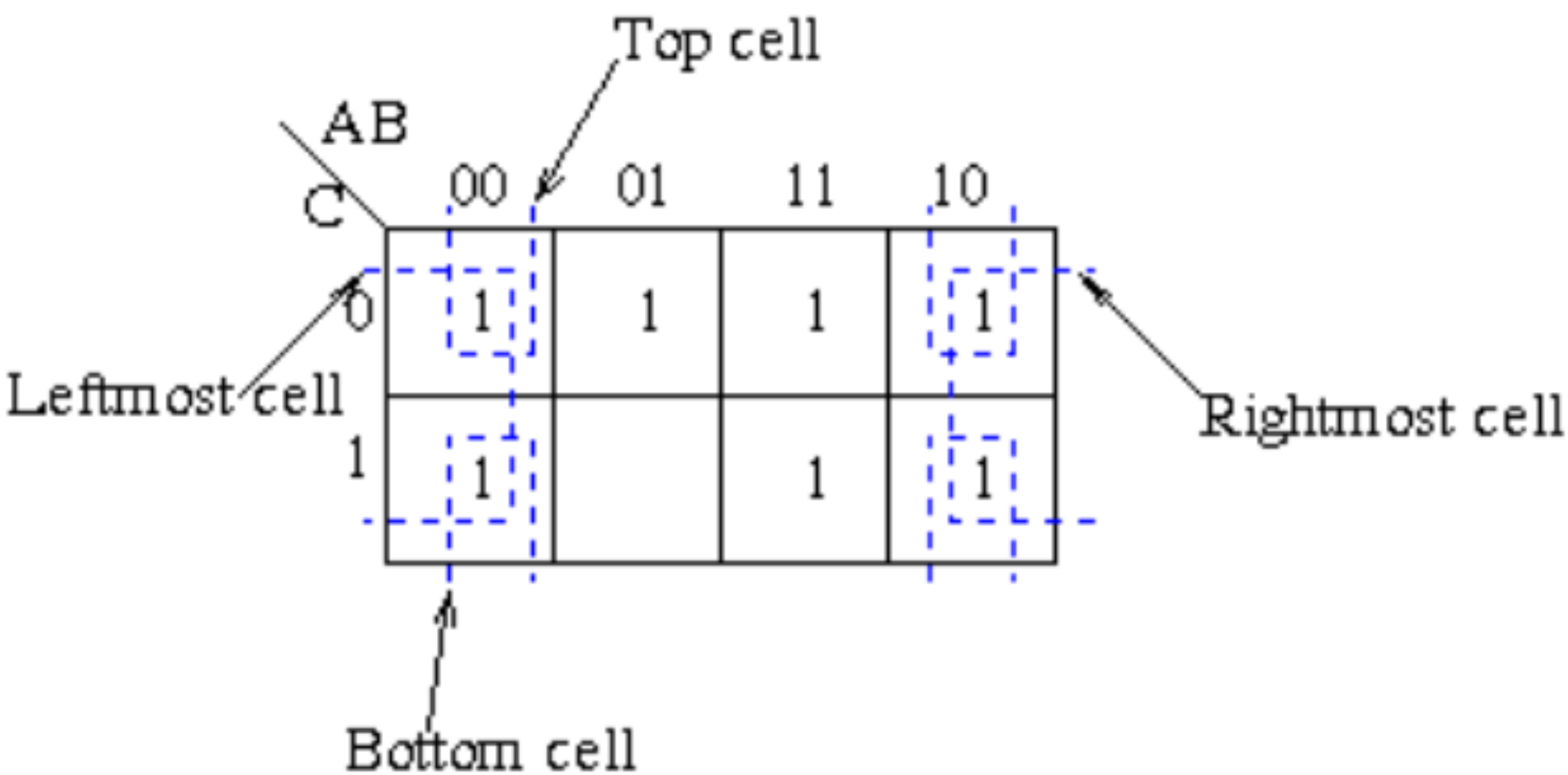
7. Groups may overlap esp. to maximize group size



Rules for Using
K-Maps for Simplification

8. Groups may wrap around the table.

The leftmost cell in a row may be grouped with the rightmost cell **<u>and</u>** the top cell in a column may be grouped with the bottom cell.



Example 1: *2 variables*

F(X,Y)

= XY + Y

= Y (X + 1)

= Y

Y = 1 column

X \ Y	0	1
0		1
1		1

Verifying results!

F(X,Y) = Y

Example 2: *3 variables*

F(X,Y,Z)

$= XZ + Z(X' + XY)$

$= XZ + ZX' + ZXY$

$= Z\,(X + X' + XY)$

$= Z\,(1 + XY)$

$= Z$

Y = 1 *X = 1*

Z \ XY	00	01	11	10
0				
1	1	1	1	1

Verifying results!

F(X,Y,Z) = Z

Example 3: *3 variables*

Class Ex.

!A!B!C + !A!BC + !ABC + !AB!C + A!B!C + AB!C

C \ AB	00	01	11	10
0	1	1	1	1
1	1	1		

F(X,Y,Z) = !C + !A

Example 4: *4 variables*

F(A,X,Y,Z)

= AX + Z(X+A’+Y)

= AX + ZX+ ZA’+ ZY

Y = 1 *X = 1*

AZ \ XY	00	01	11	10
00				
01	1	1	1	1
11		1	1	1
10			1	1

Z = 1

A = 1

F(A,X,Y,Z) = ZA’ + AX + ZY

Example 4: *4 variables*

Class Ex.

F(A,B,C,D)

= ABCD’ + ABC’D + CD + A’B’ + C’D

B = 1   *A = 1*

CD \ AB	00	01	11	10
00	1			
01	1	1	1	1
11	1	1	1	1
10	1		1	

D = 1

C = 1

F(A,B,C,D) = A’B’ + D + ABC

K-Map Rules Summary

1. Groups can contain only 1s
2. Only 1s in adjacent groups are allowed
3. Groups may ONLY be horizontal or vertical (no diagonals)
4. The number of 1s in a group must be a power of two (1, 2, 4, 8...)
5. Groups must be as large AND as few in no.s as “legally” possible
6. All 1s must belong to a group, even if it’s a group of one element
7. Overlapping groups are permitted
8. Wrapping around the map is permitted

Exploiting "Don't Cares"

- An *output variable* that's designated "don't care" (symbol = **X**) means that it could be a **0** or a **1** (i.e. we "don't care" which)

- That is, it is **unspecified**,
  usually because of invalid inputs

- **In K-Maps, "Don't Cares" Can Be Advantageous!!**

Example of a Don't Care Situation

- Consider coding all decimal digits (say, for a digital clock app):
  - 0 thru 9 --- requires how many bits?
    - 4 bits
  - But! 4 bits convey more numbers than that!
    - Don't forget A thru F!

- Not all binary values map to decimal

Example Continued…

Binary	Decimal
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7

Binary	Decimal
1000	8
1001	9
1010	X
1011	X
1100	X
1101	X
1110	X
1111	X

Don’t Care: So What?

- Recall that in a K-map, we can only group 1s

- Because the value of a don’t care is irrelevant, we can treat it as a 1 ***if it is convenient to do so*** (or a 0 if that would be more convenient)

Example

- A circuit that calculates if the 4-bit binary coded *single digit* decimal **input % 2 == 0**

- So, although 4-bits will give me numbers from 0 to 15, I ***don't care*** about the ones that yield 10 to 15.

I3	I2	I1	I0	R
0	0	0	0	**1**
0	0	0	1	**0**
0	0	1	0	**1**
0	0	1	1	**0**
0	1	0	0	**1**
0	1	0	1	**0**
0	1	1	0	**1**
0	1	1	1	**0**
1	0	0	0	**1**
1	0	0	1	**0**
1	0	1	0	**X**
1	0	1	1	**X**
1	1	0	0	**X**
1	1	0	1	**X**
1	1	1	0	**X**
1	1	1	1	**X**

Example as a K-Map

I_3I_2 \ I_1I_0	00	01	11	10
00	1	0	0	1
01	1	0	0	1
11	X	X	X	X
10	1	0	X	X

If We **Don't Exploit** "Don't Cares"

$$R = \overline{I_1}\,\overline{I_0}\,\overline{I_3} + I_1\overline{I_0}\,\overline{I_3} + \overline{I_0}\,\overline{I_1}\,\overline{I_2}\,I_3$$

I_3I_2 \ I_1I_0	00	01	11	10
00	1	0	0	1
01	1	0	0	1
11	X	X	X	X
10	1	0	X	X

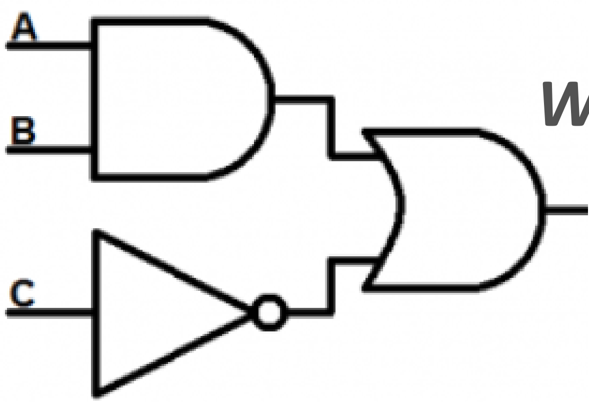
If We DO **Exploit** “Don’t Cares”

$$\mathbf{R = \overline{I}_0}$$

I_3I_2 \ I_1I_0	00	01	11	10
00	1	0	0	1
01	1	0	0	1
11	X	X	X	X
10	1	0	X	X

Combinatorial Logic Designs

- When you *combine* multiple logic blocks together to form a more complex logic function/circuit



What is the output?

$A.B + \overline{C}$

What is its truth table?

A	B	C	F
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

What is its K-Map?

C \ AB	00	01	11	10
0	1	1	1	1
1			1	

Exercise 1

- Given the following truth table, draw the resulting logic circuit
  - **STEP 1**: Draw the K-Map and simplify the function
  - **STEP 2**: Construct the circuit from the now simplified function

A	B	C	D	F
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	0
1	1	1	0	1
1	1	1	1	1

A	B	C	D	F
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	0
1	1	1	0	1
1	1	1	1	1

Exercise 1 – Step 1

Get the simplified function

B = 1 *A = 1*

CD \ AB	00	01	11	10
00		1	1	
01				
11			1	1
10			1	1

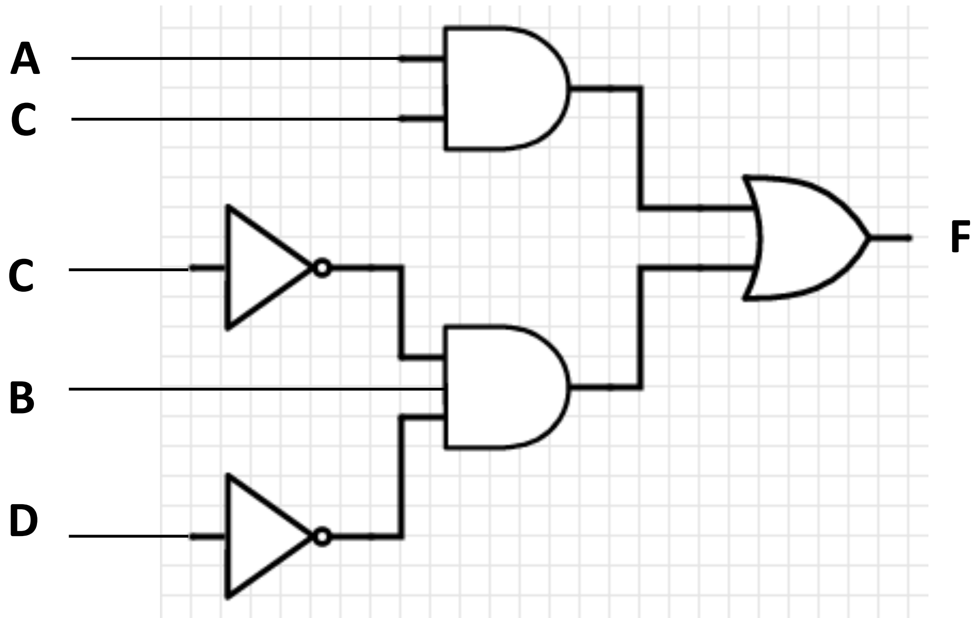
D = 1

C = 1

F(A,B,C) = B.C’.D’ + A.C

Exercise 1 – Step 2
Draw the logic circuit diagram

F(A,B,C) = B.C'.D' + A.C



Exercise 2

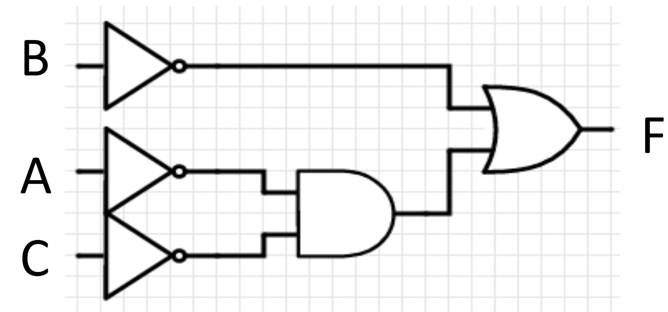
Class Ex.

- Given the following truth table, draw the resulting logic circuit

A	B	C	F
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

C \ AB	00	01	11	10
0	1	1		1
1	1			1

$F(A,B,C) = B' + A'.C'$

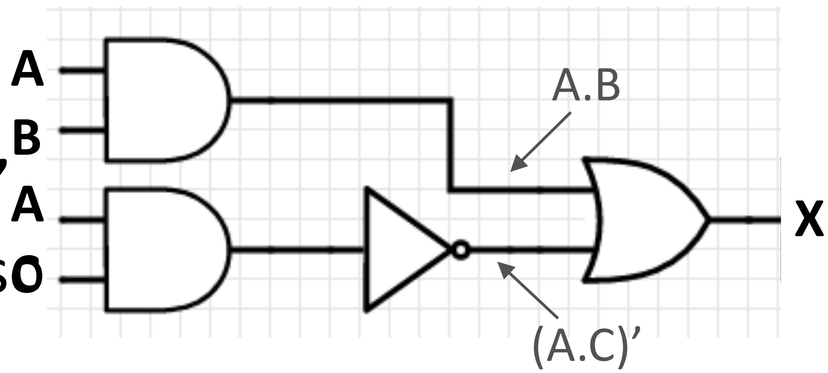


Exercise 3

- Given the following schematic of a circuit, (a) write the function and (b) fill out the truth table:

X = A.B + (A.C)'

(note that also



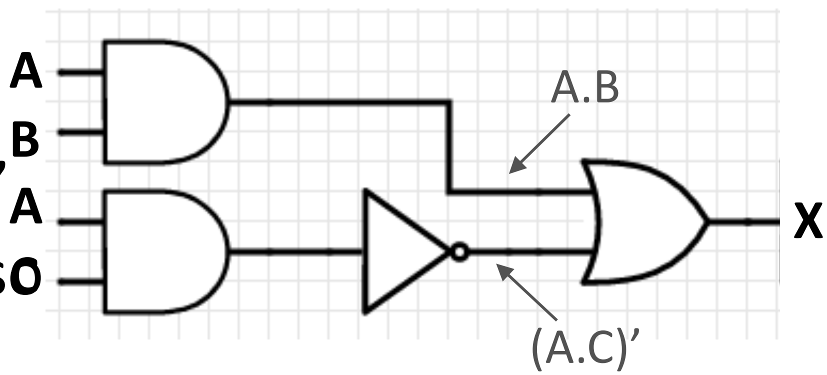
A	B	C	X
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

Exercise 3

- Given the following schematic of a circuit, (a) write the function and (b) fill out the truth table:

X = A.B + (A.C)'

(note that also



A	B	C	X
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

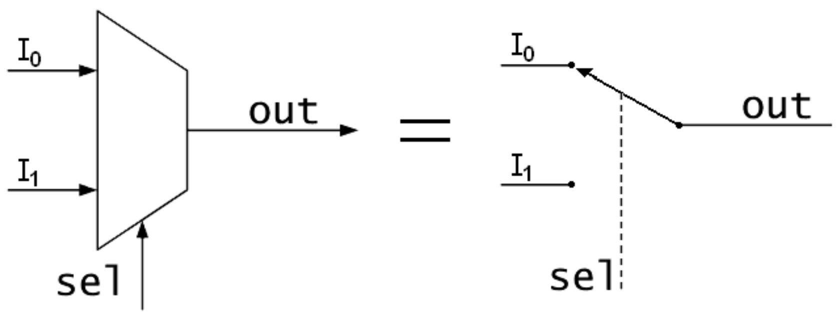
Multiplexer

- A logical selector:
  - Select either input A or input B to be the output

```
// if s = 0, output is a
// if s = 1, output is b
int mux(int a, int b, int s)
{
    if (!s) return a;
    else return b;
}
```

Multiplexer
(Mux for short)

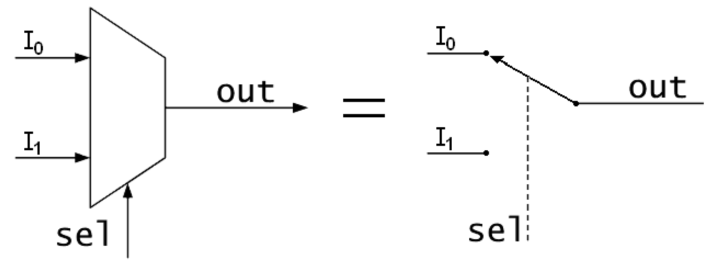
- Typically has 3 *groups of* inputs and 1 output
  - IN: 2 data , 1 select
  - OUT: 1 data



- 1 of the input data lines gets selected to become the output, based on the 3rd (select) input
  - If "Sel" = 0, then I_0 gets to be the output
  - If "Sel" = 1, then I_1 gets to be the output

- The opposite of a Mux is called a **Demulitplexer** (or **Demux**)

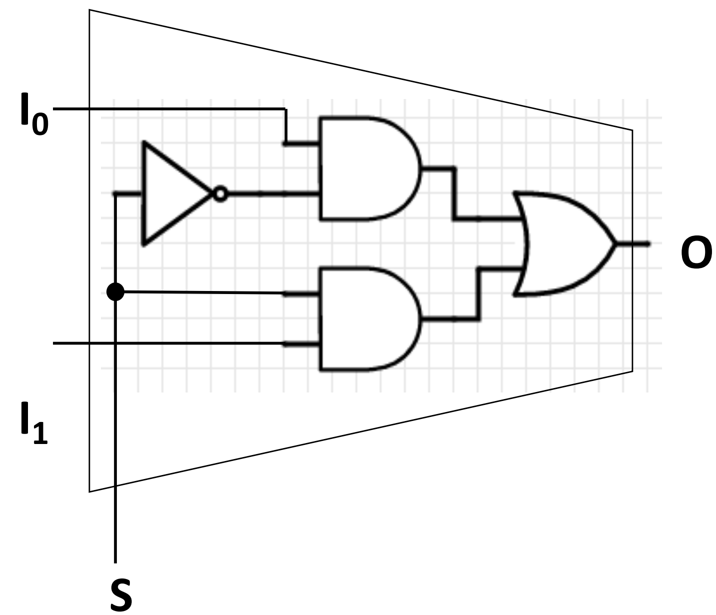
Mux Truth Table and Logic Circuit
1-bit Mux



I_0	I_1	S	O
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

S \ I_0 I_1	00	01	11	10
0			1	1
1		1	1	

$$O = S.I_1 + S'.I_0$$



• = lines are physically connected

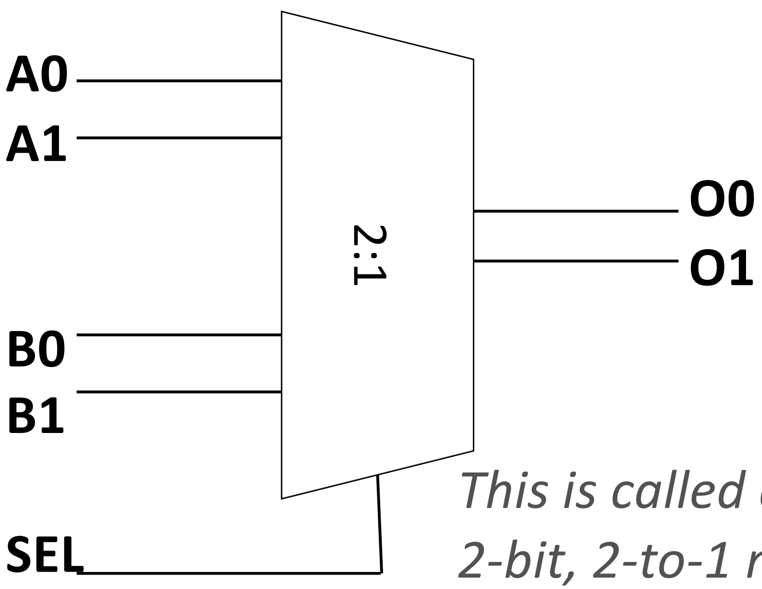
YOUR TO-DOs

- Lab 6!

</LECTURE>

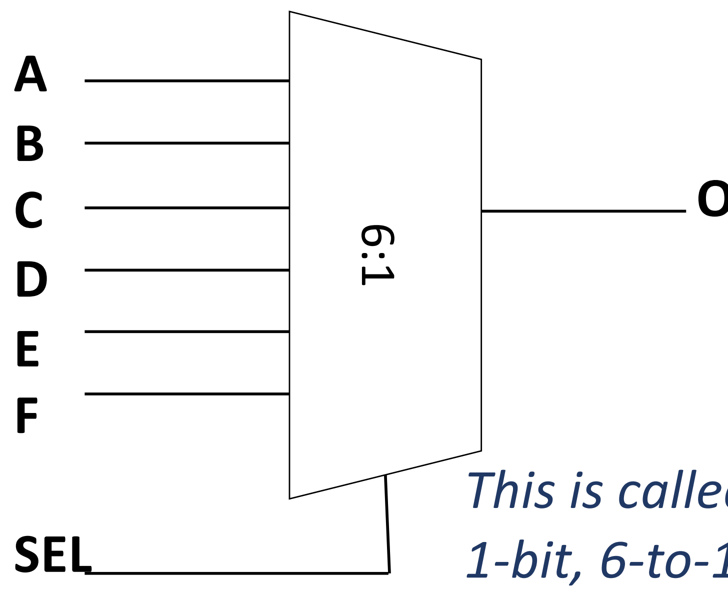
Mux Configurations

Muxes can have I/O that are multiple bits



This is called a 2-bit, 2-to-1 mux

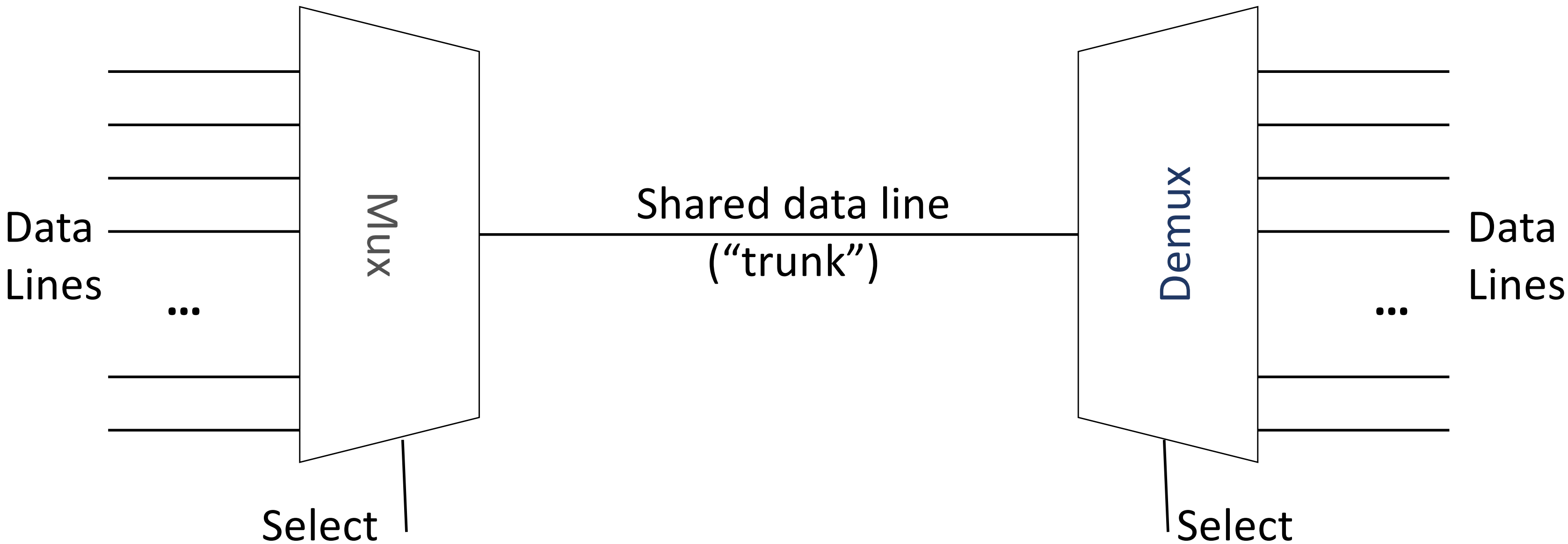
Or they can have more than two data inputs



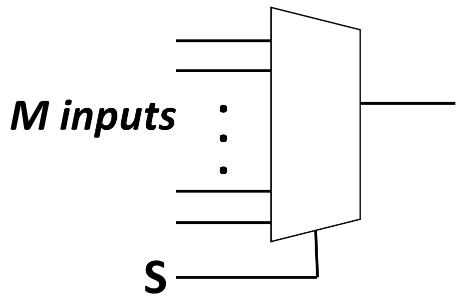
This is called a 1-bit, 6-to-1 mux

The Use of Multiplexers

- Makes it possible for several signals (variables) to share one resource
  - Very commonly used in data communication lines



Selection Lines in Muxes



- General mux description: **N-bit, M-to-1**
- Where: N = how "wide" the input is (# of input bits, min. 1)
  M = how many inputs to the mux (min. 2)

- The "select" input (S) has to be able to select **1 out of M inputs**
  - So, if M = 2, S should be at least 1 bit *(S = 0 for one line, S = 1 for the other)*
  - But if M = 3, S should be at least **2 bits** *(why?)*
  - If M = 4, S should be ??? *(**ANS**: at least 2 bits)*
  - If M = 5, S should be ??? *(**ANS**: at least 3 bits)*

Combining Muxes Together

Can I do a **4:1** mux from 2:1 muxes?

Generally, you can do **2^n:1** muxes from 2:1 muxes.