Multiplexers
Decoders
ROMs (LUTs)

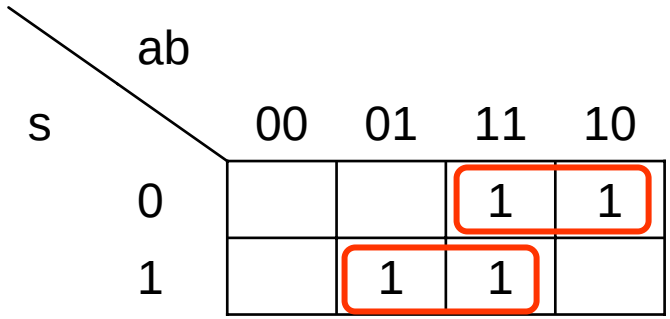
A Problem Statement

Design a circuit which will
select
between two inputs (A and B) and
pass
the selected one to the output (Q).

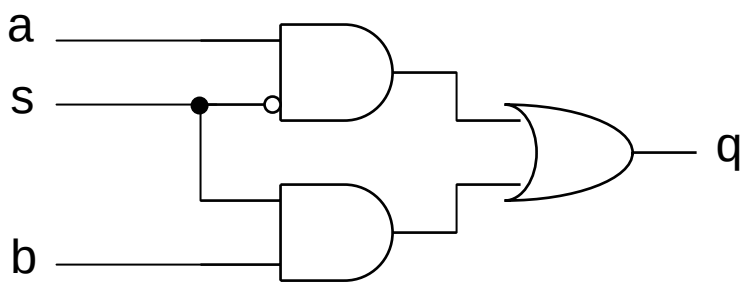
The desired circuit is called a multiplexer or MUX for short

Multiplexers

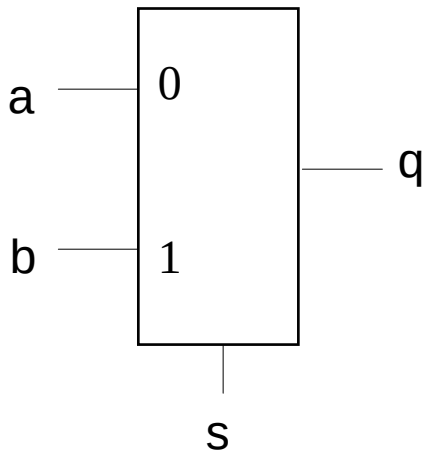
s	a	b	q
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

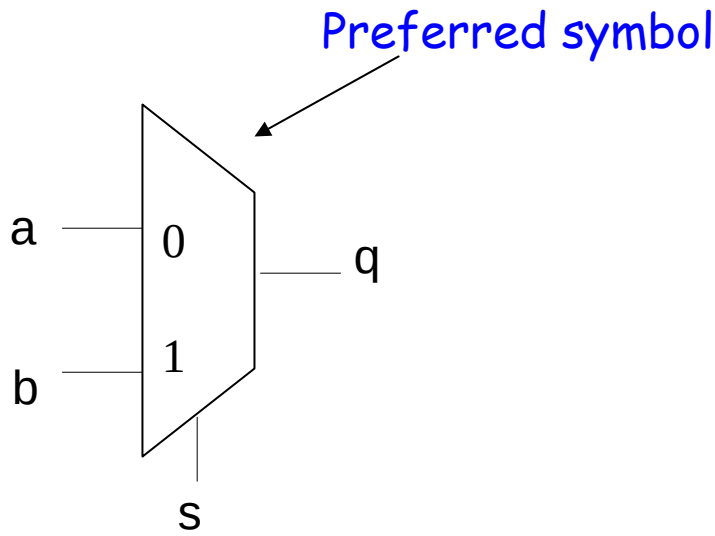


$q = s'a + sb$

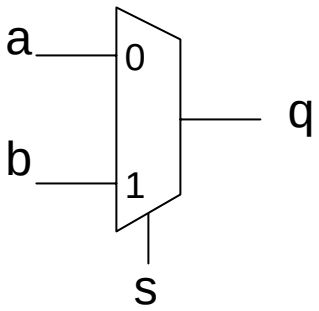


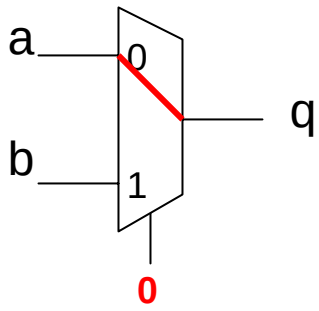
Multiplexer Symbols

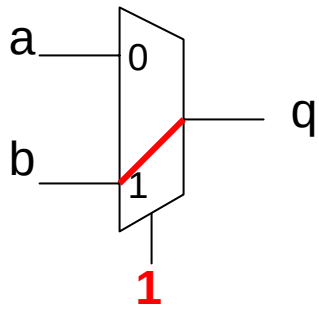




Data Steering

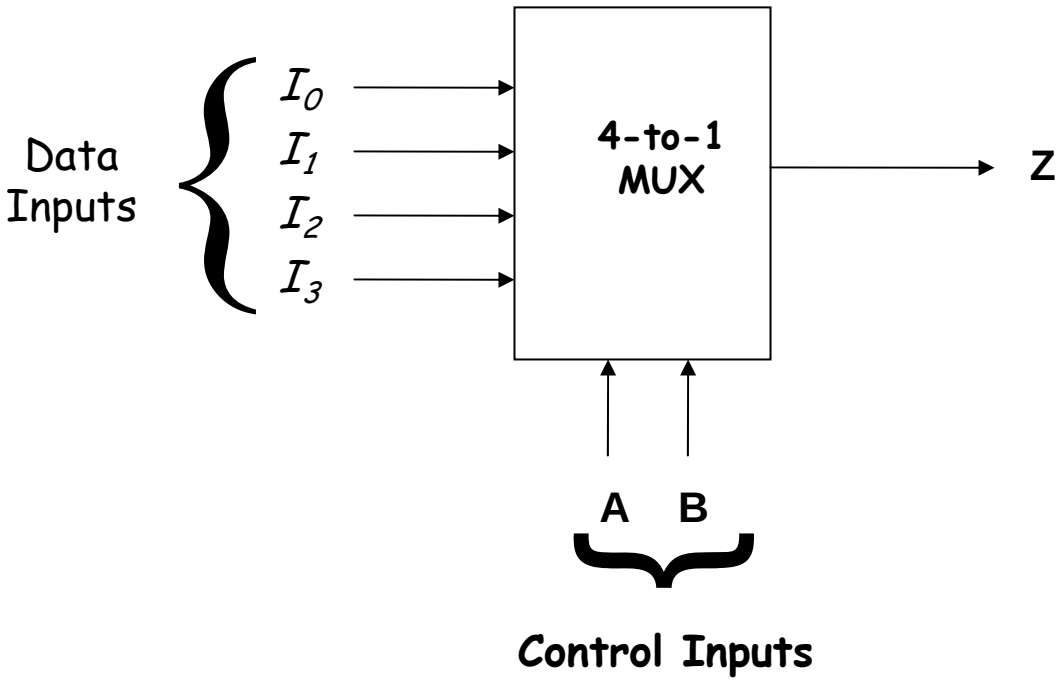






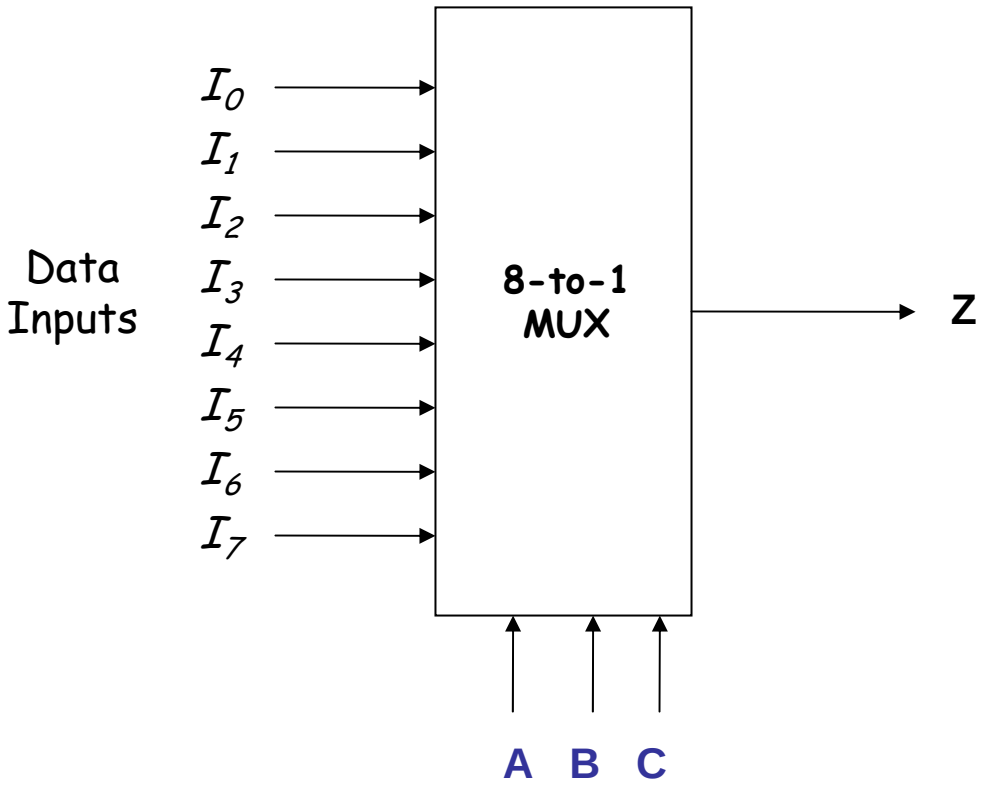
Key idea: the select input wire selects one of the inputs and passes it out to the output.

Multiplexers



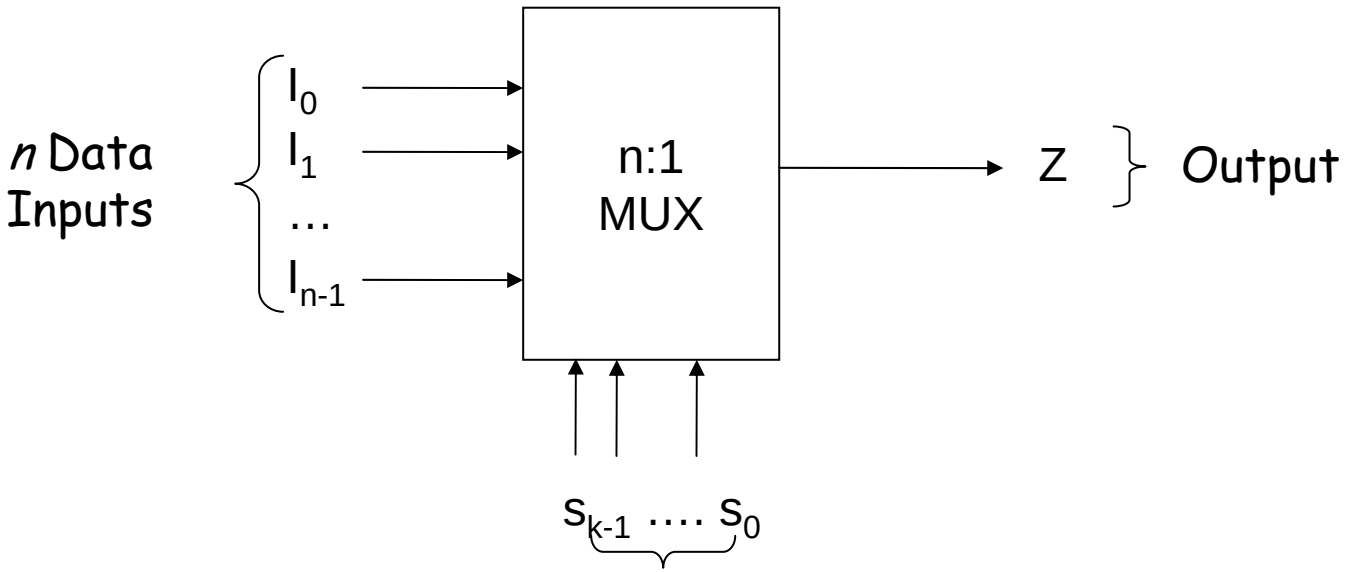
$$Z = A'B'I_0 + A'BI_1 + AB'I_2 + ABI_3$$

8 to 1 Multiplexer



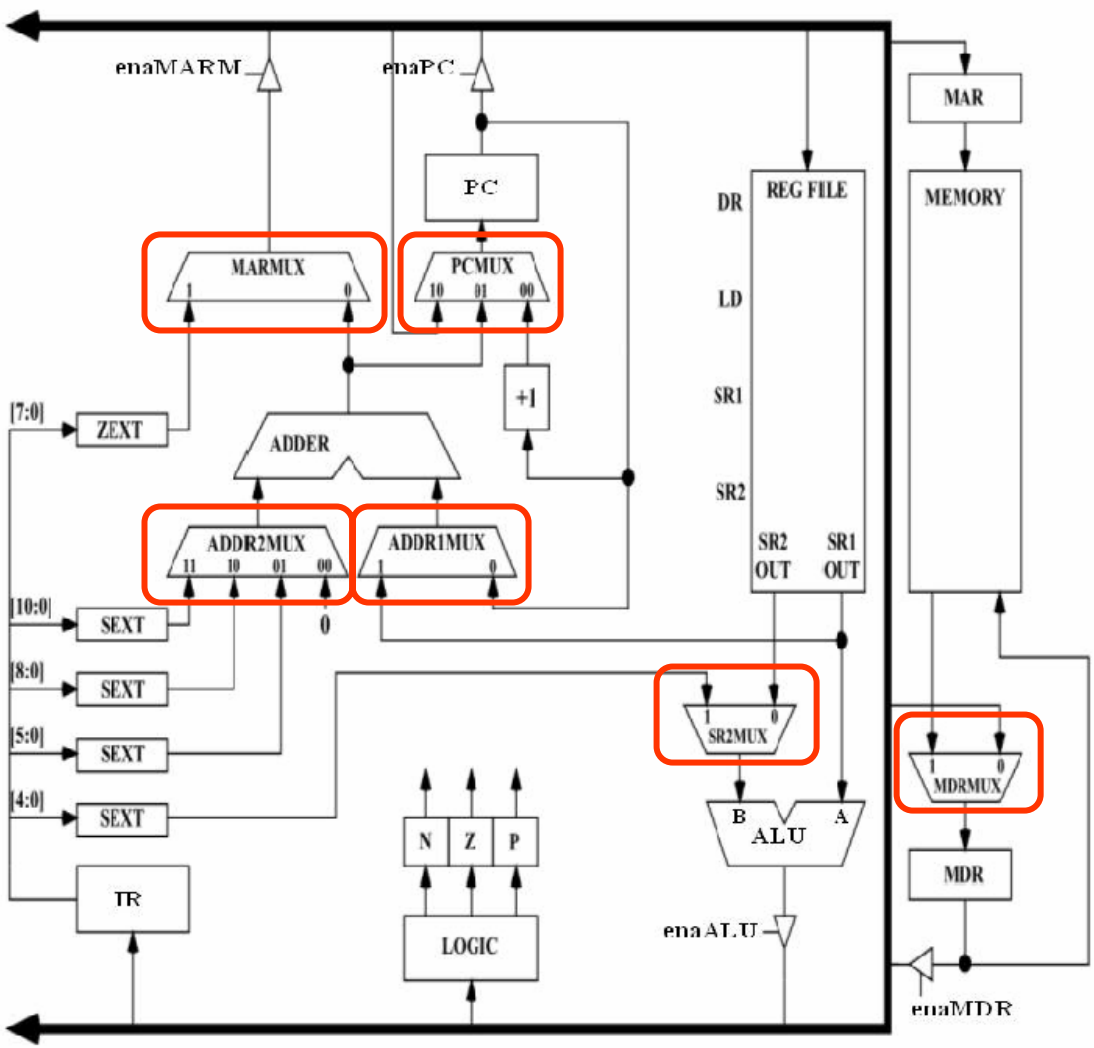
$$Z = A'B'C'I_0 + A'B'CI_1 + A'BC'I_2 + \ldots$$

A General MUX

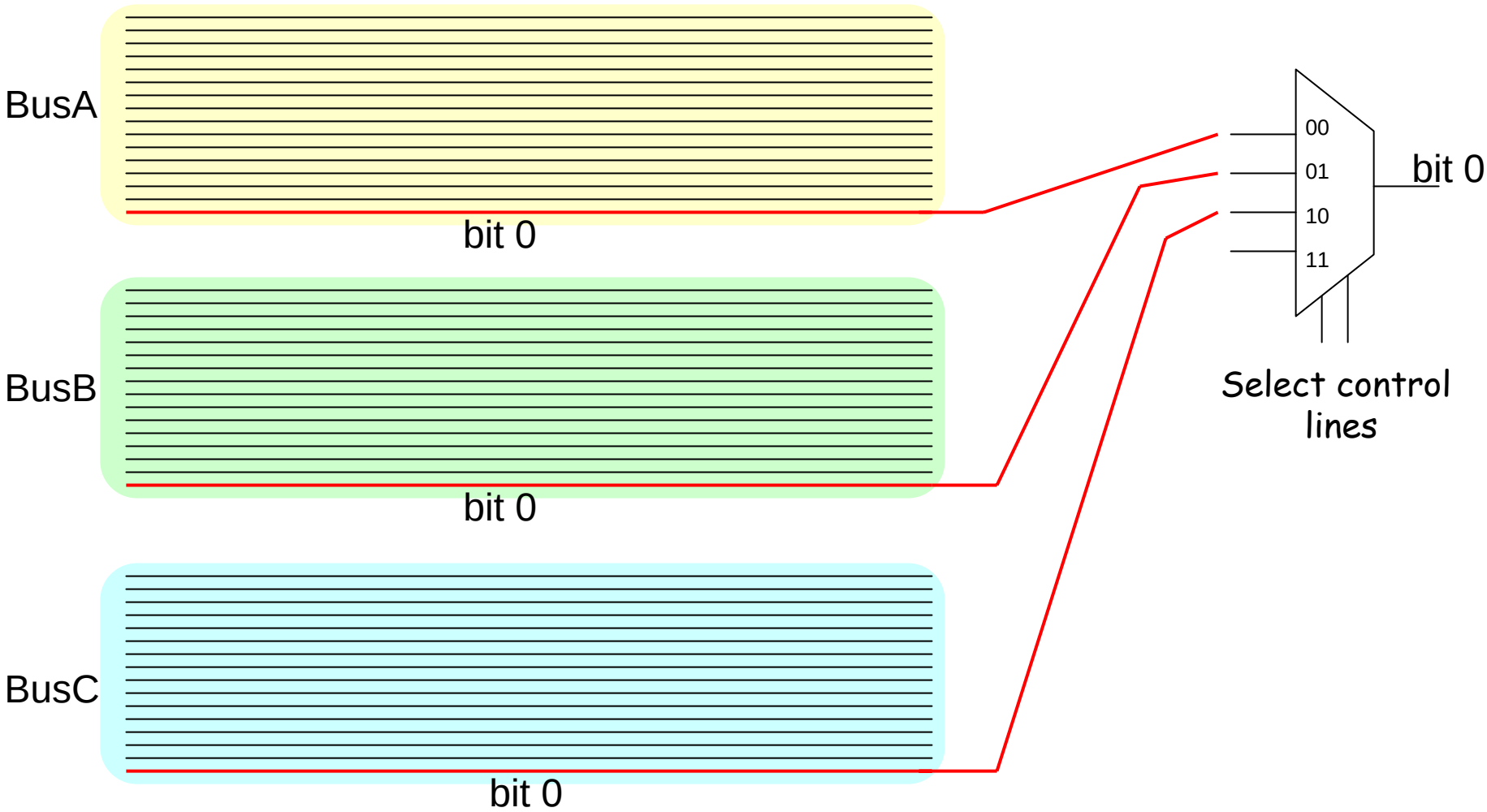


MUXes In A Microprocessor

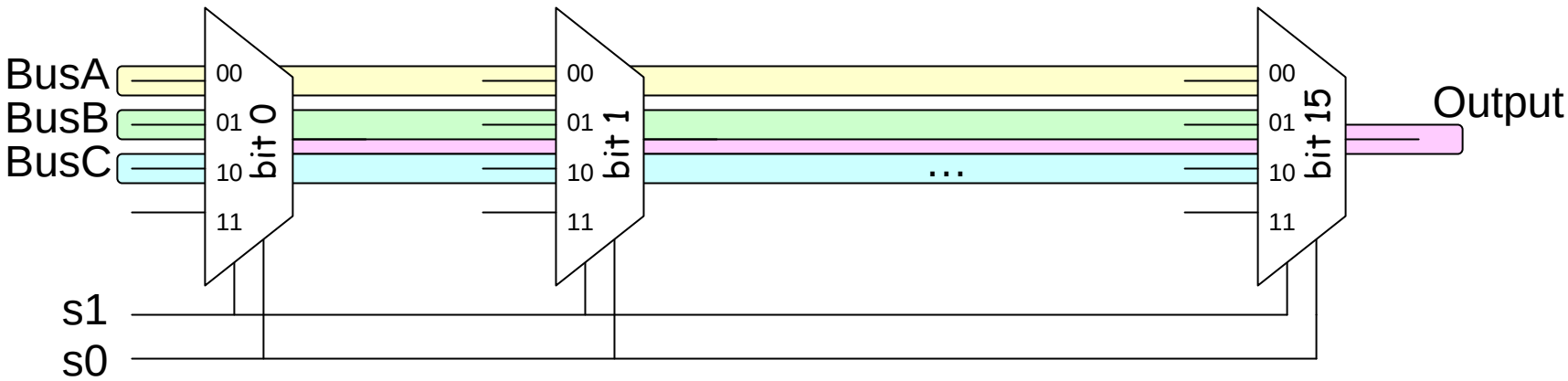
- They are each 16-bits wide
  - A 16-bit wide MUX is simply 16 1-bit wide MUXes all with common ***s*** inputs



16-bit 3:1 MUX Example



16-bit 3:1 MUX Example



They all use the same select control lines…
One bit from each bus goes to each MUX
The result is a 16-bit bus

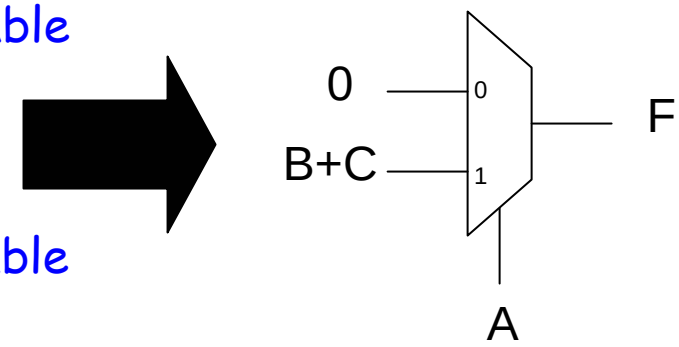
Implementing Logic with MUX Blocks

Example 1

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

A=0 part of the truth table
… when *A=0, F=0*

A=1 part of the truth table
… when A=1, F=B+C

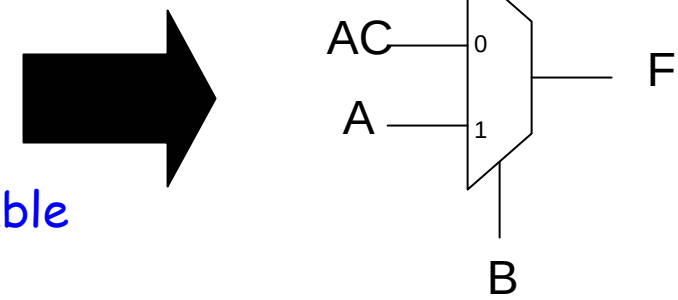


Example 2

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

B=0 part of the truth table
… when B=0, F=AC

B=1 part of the truth table
… when B=1, F=A

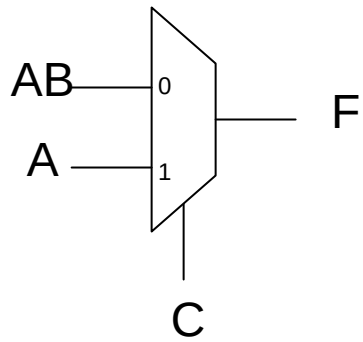


Example 3

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

C=0 part of the truth table
… when C=0, F=AB

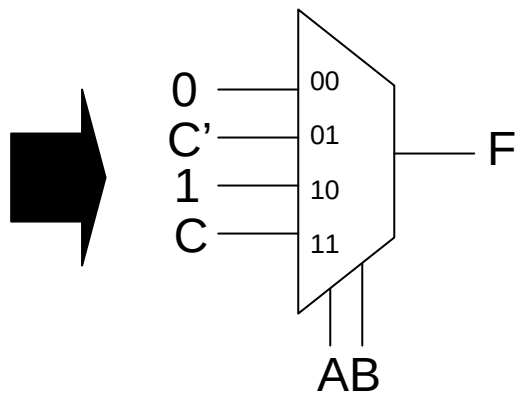
C=1 part of the truth table
… when C=1, F=A



All 3 of these examples are the same truth table…

Using a Bigger MUX

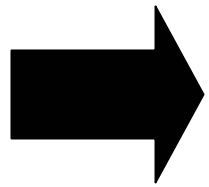
A	B	C	F	
0	0	0	0	AB=00 part of the truth table
0	0	1	0	… when AB=00, F=0
0	1	0	1	AB=01 part of the truth table
0	1	1	0	… when AB=01, F=C'
1	0	0	1	AB=10 part of the truth table
1	0	1	1	… when AB=10, F=1
1	1	0	0	AB=11 part of the truth table
1	1	1	1	… when AB=11, F=C

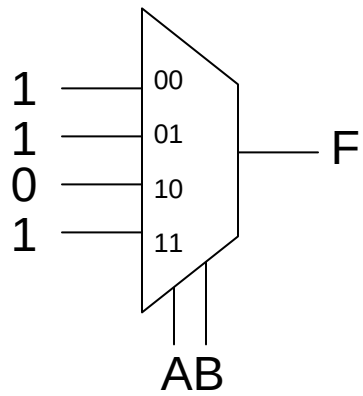


Can easily re-order truth table to use different MUX control inputs

Another 4:1 MUX Example

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





This shows that a large enough MUX can directly implement a truth table…

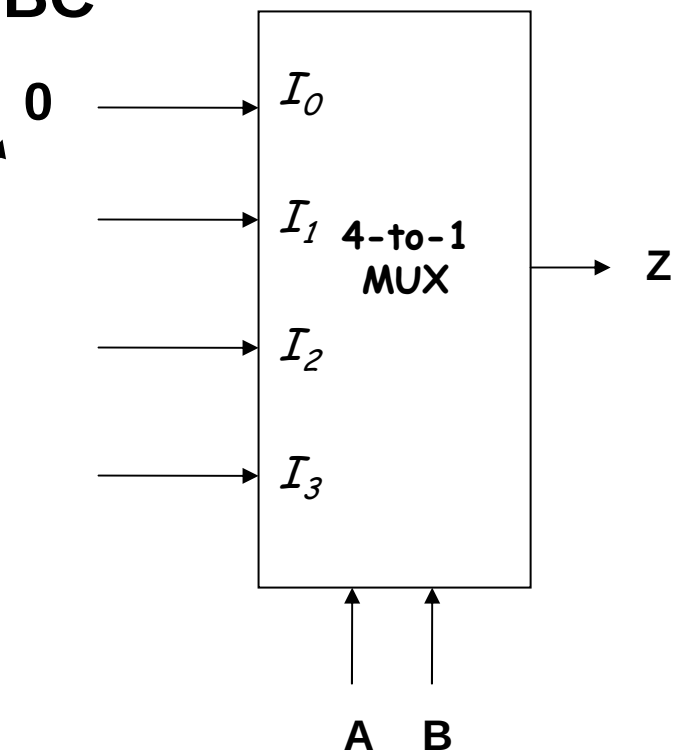
Implementing Logic Functions With Muxes

Implement:

$$Z = A'B + BC'$$

A B / C	00	01	11	10
0	0	1	1	0
1	0	1	0	0

for AB=00, Z=0



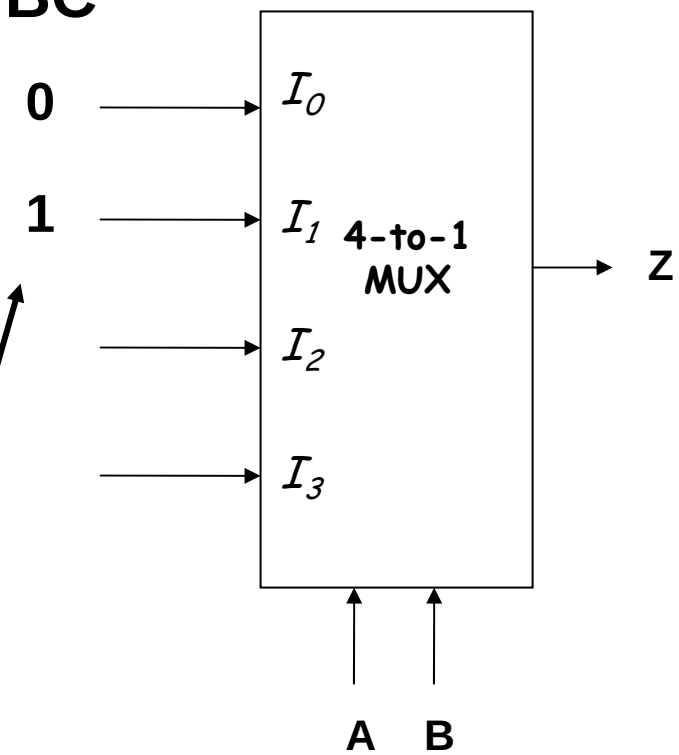
Implementing Logic Functions With Muxes

Implement:

$$Z = A'B + BC'$$

C \ A B	00	01	11	10
0	0	1	1	0
1	0	1	0	0

for AB=01, Z=1



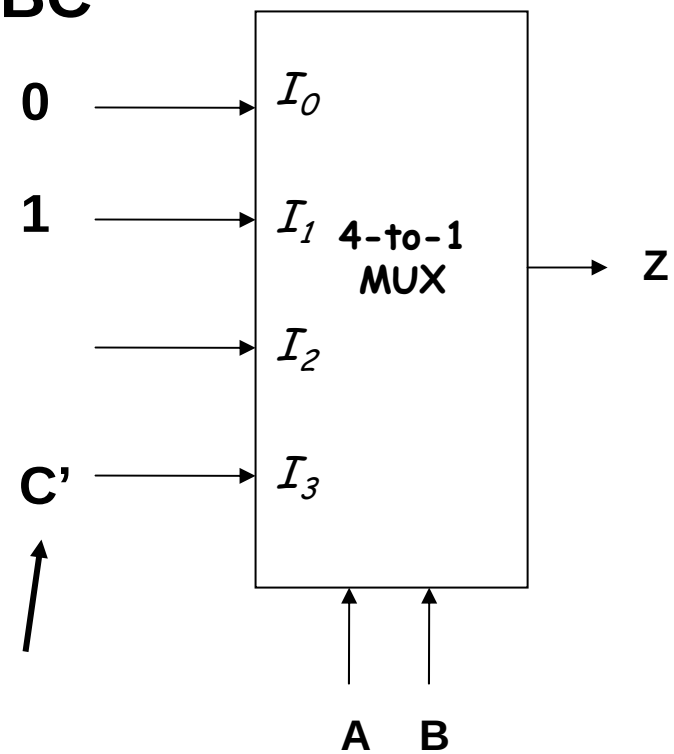
Implementing Logic Functions With Muxes

Implement:

$$Z = A'B + BC'$$

C \ A B	00	01	11	10
0	0	1	1	0
1	0	1	0	0

for AB=11, Z=C'

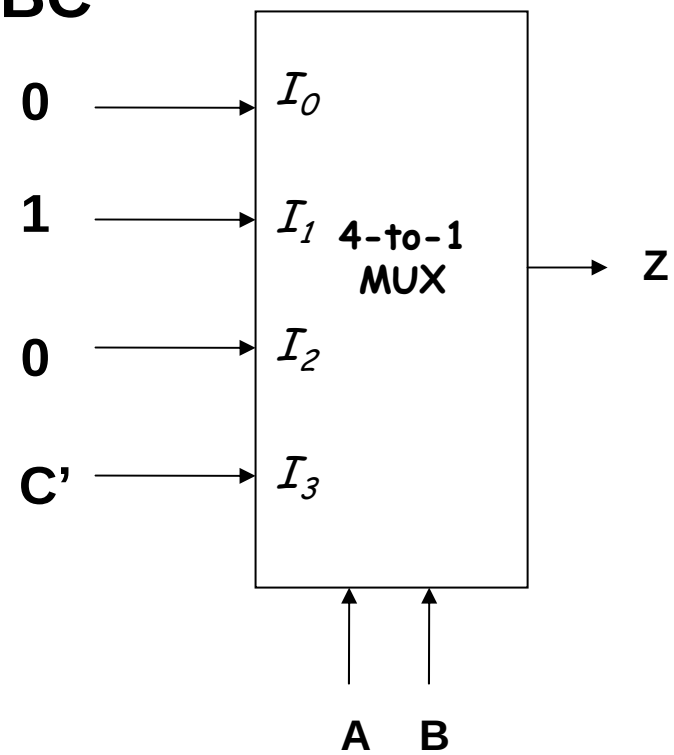


Implementing Logic Functions With Muxes

Implement:

$$Z = A'B + BC'$$

C \ A B	00	01	11	10
0	0	1	1	0
1	0	1	0	0



Implementing Logic Functions With Muxes
An alternate method

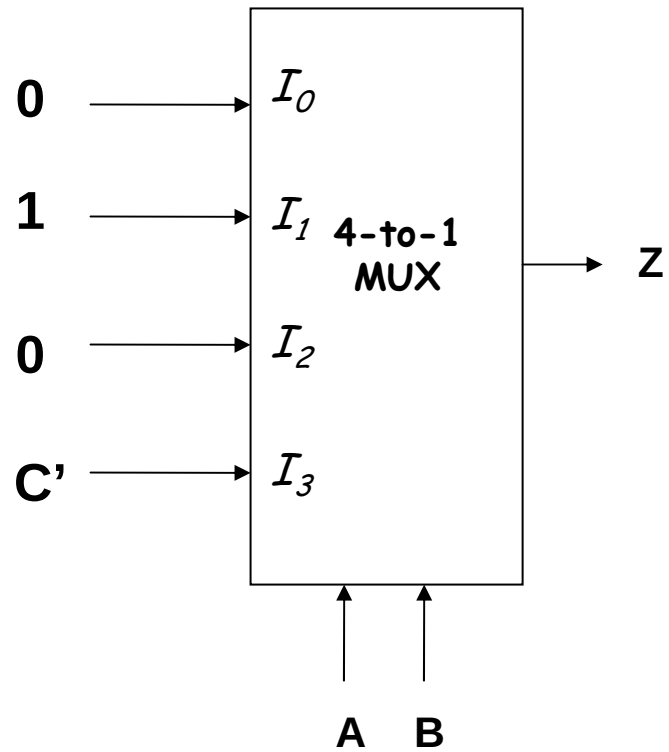
$$Z = A'B + BC'$$

A=0 B=0 $Z = 1 \bullet 0 + 0 \bullet C' = 0$

A=0 B=1 $Z = 1 \bullet 1 + 1 \bullet C' = 1$

A=1 B=0 $Z = 0 \bullet 0 + 0 \bullet C' = 0$

A=1 B=1 $Z = 0 \bullet 1 + 1 \bullet C' = C'$



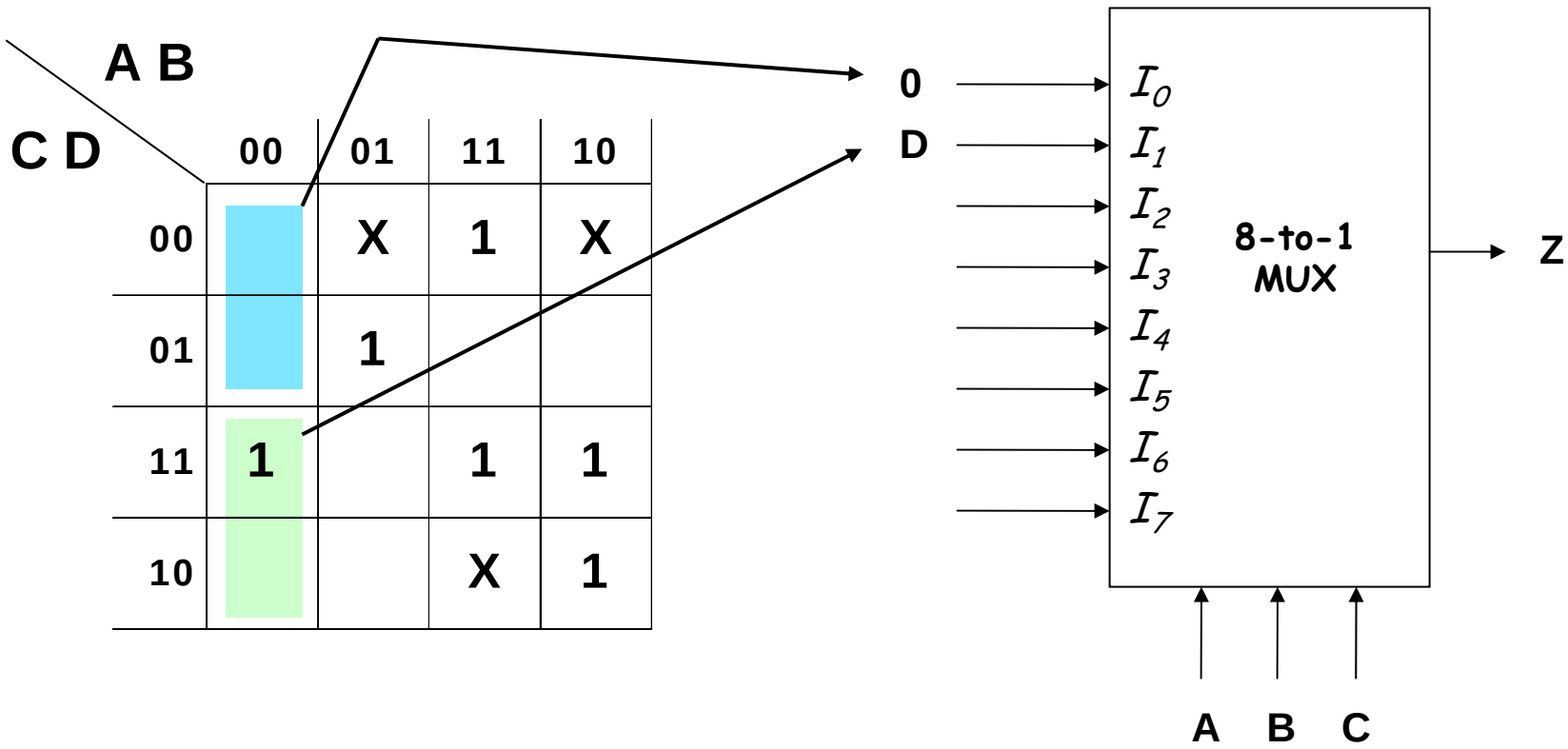
Implementing Logic Functions With Muxes

$$Z(A,B,C,D) = \sum m(3,5,10,11,12,15) + \sum d(4,8,14)$$

C D \ A B	00	01	11	10
00	0	X 4	1 12	X 8
01	1	1 5	13	9
11	1 3	7	1 15	1 11
10	2	6	X 14	1 10

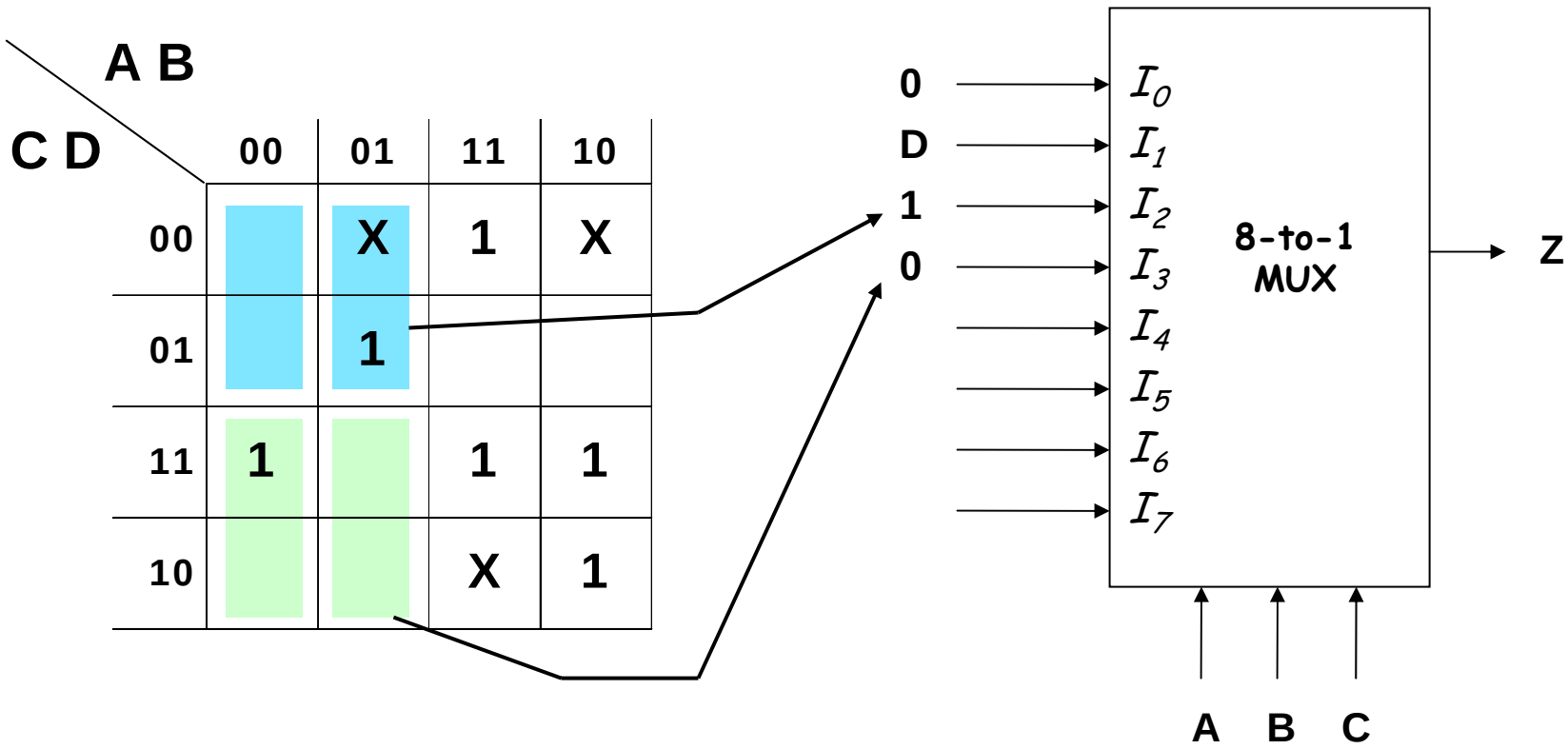
Implementing Logic Functions With Muxes

$$Z(A,B,C,D) = \sum m(3,5,10,11,12,15) + \sum d(4,8,14)$$



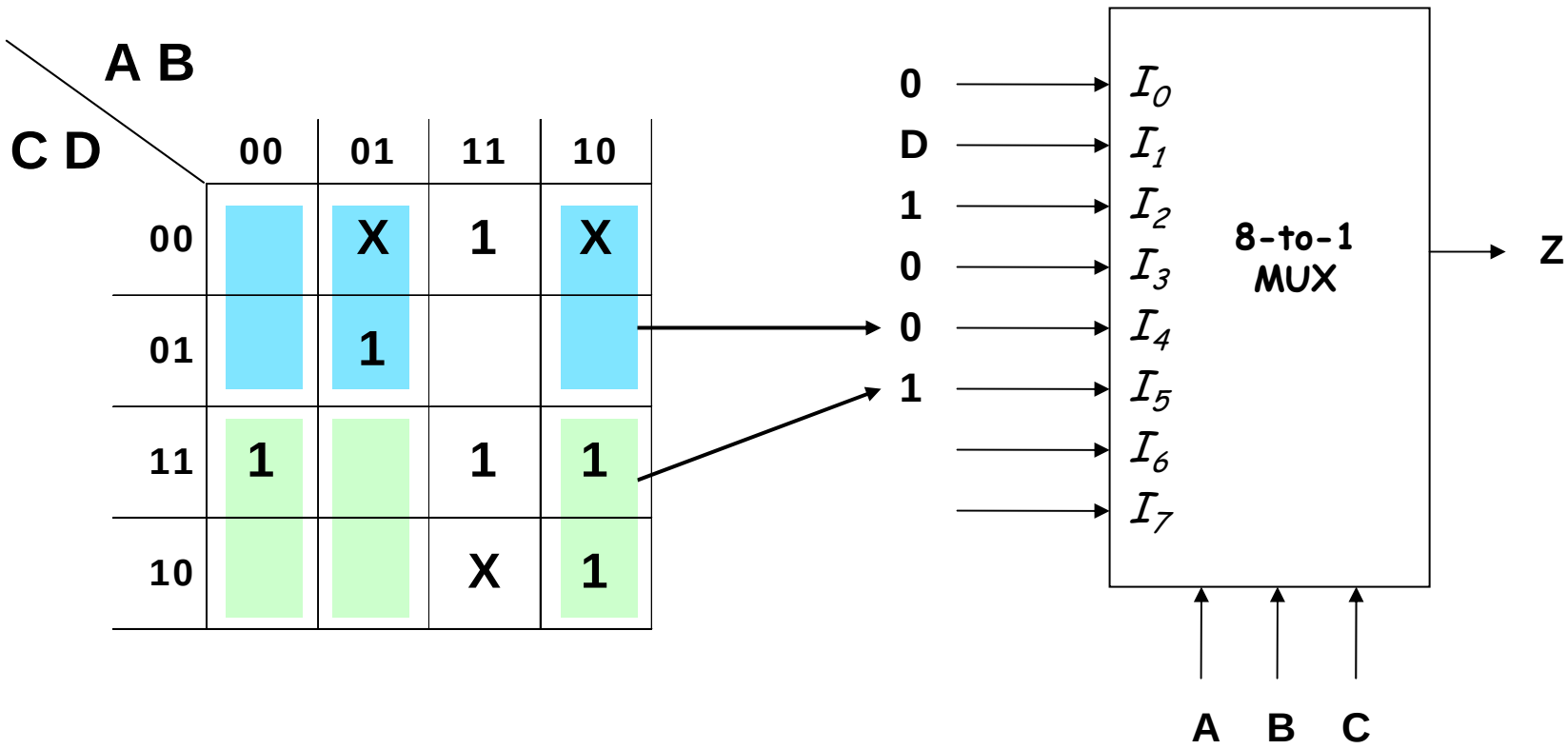
Implementing Logic Functions With Muxes

$$Z(A,B,C,D) = \sum m(3,5,10,11,12,15) + \sum d(4,8,14)$$



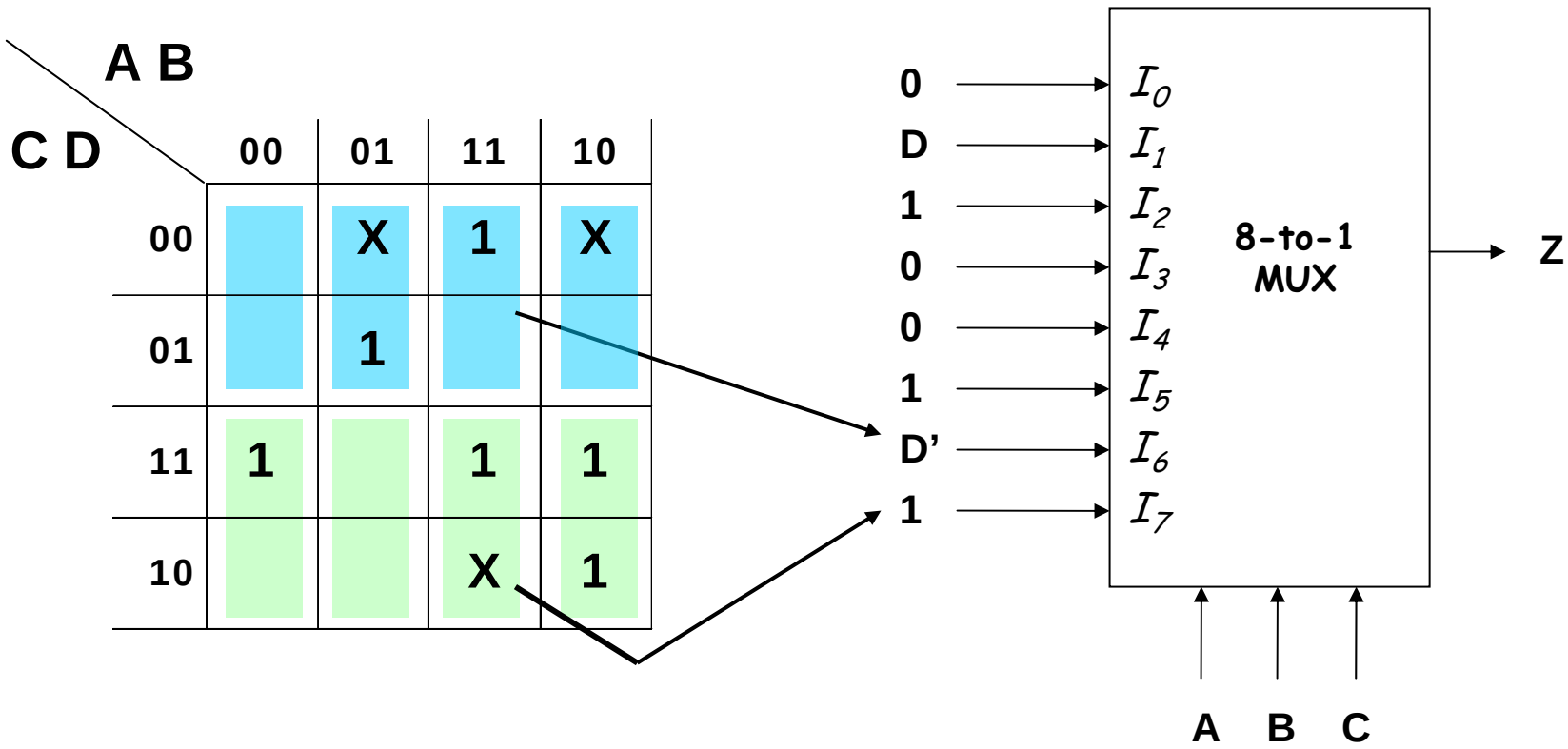
Implementing Logic Functions With Muxes

$$Z(A,B,C,D) = \sum m(3,5,10,11,12,15) + \sum d(4,8,14)$$



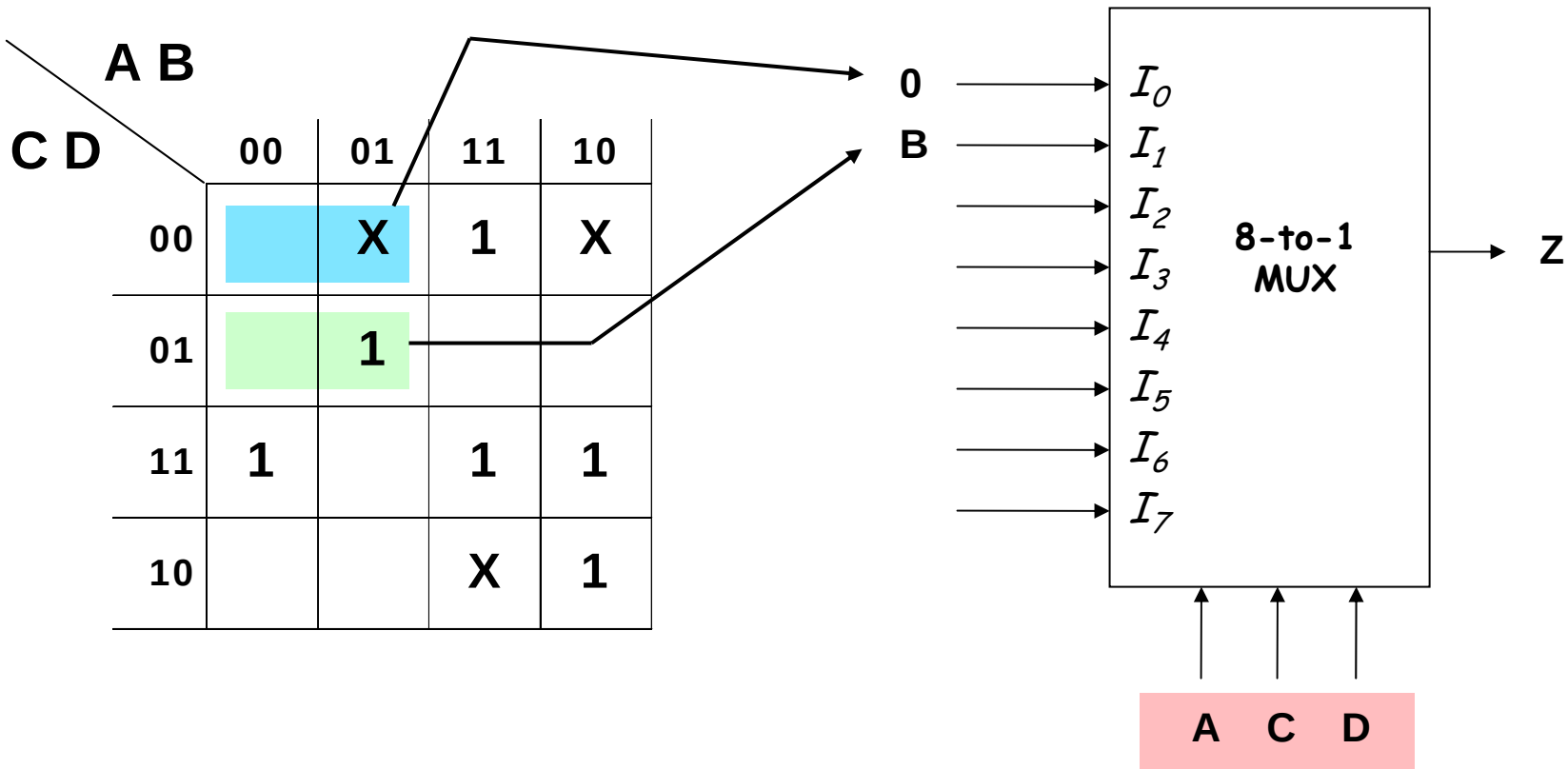
Implementing Logic Functions With Muxes

$$Z(A,B,C,D) = \sum m(3,5,10,11,12,15) + \sum d(4,8,14)$$



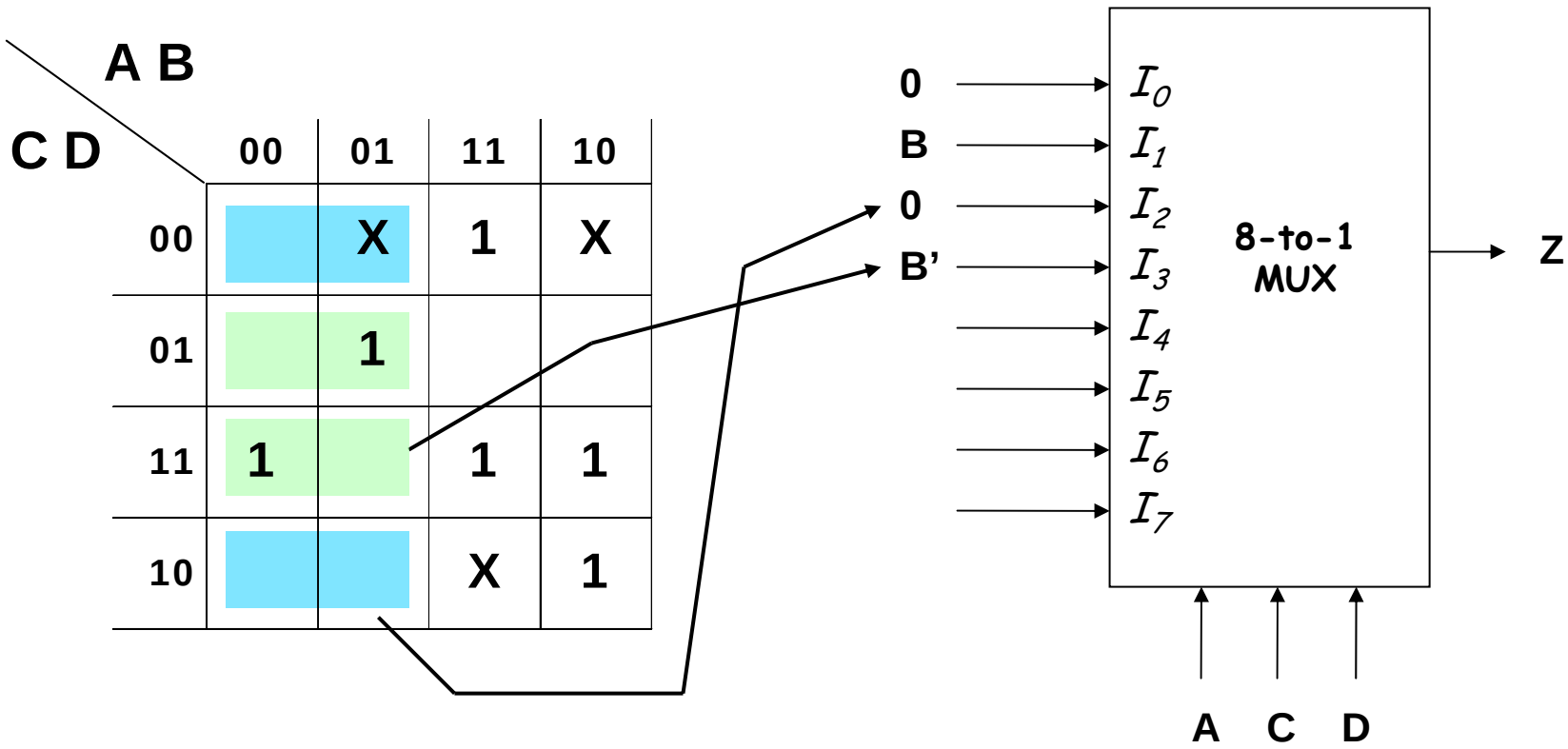
Implementing Logic Functions With Muxes

$$Z(A,B,C,D) = \sum m(3,5,10,11,12,15) + \sum d(4,8,14)$$



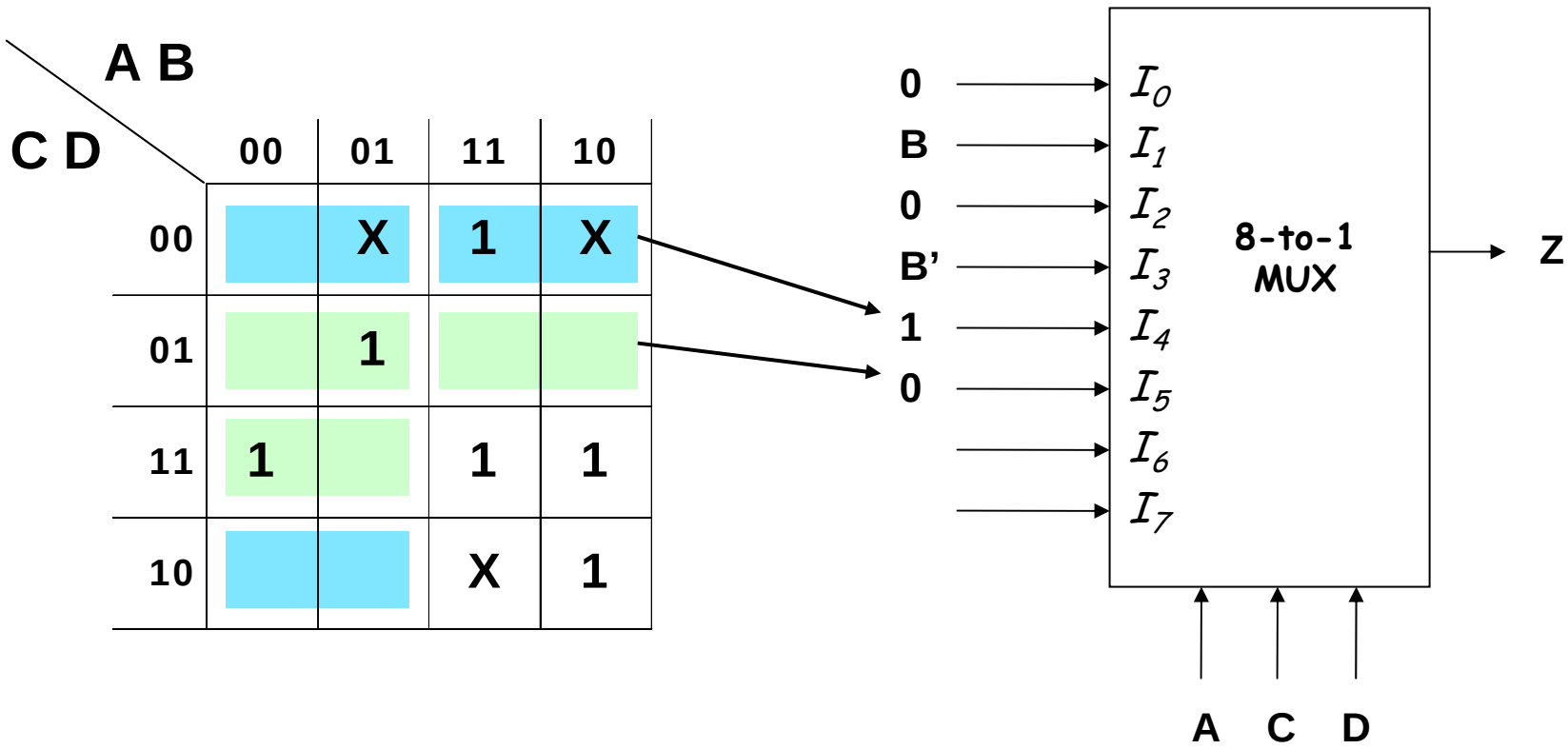
Implementing Logic Functions With Muxes

$$Z(A,B,C,D) = \sum m(3,5,10,11,12,15) + \sum d(4,8,14)$$



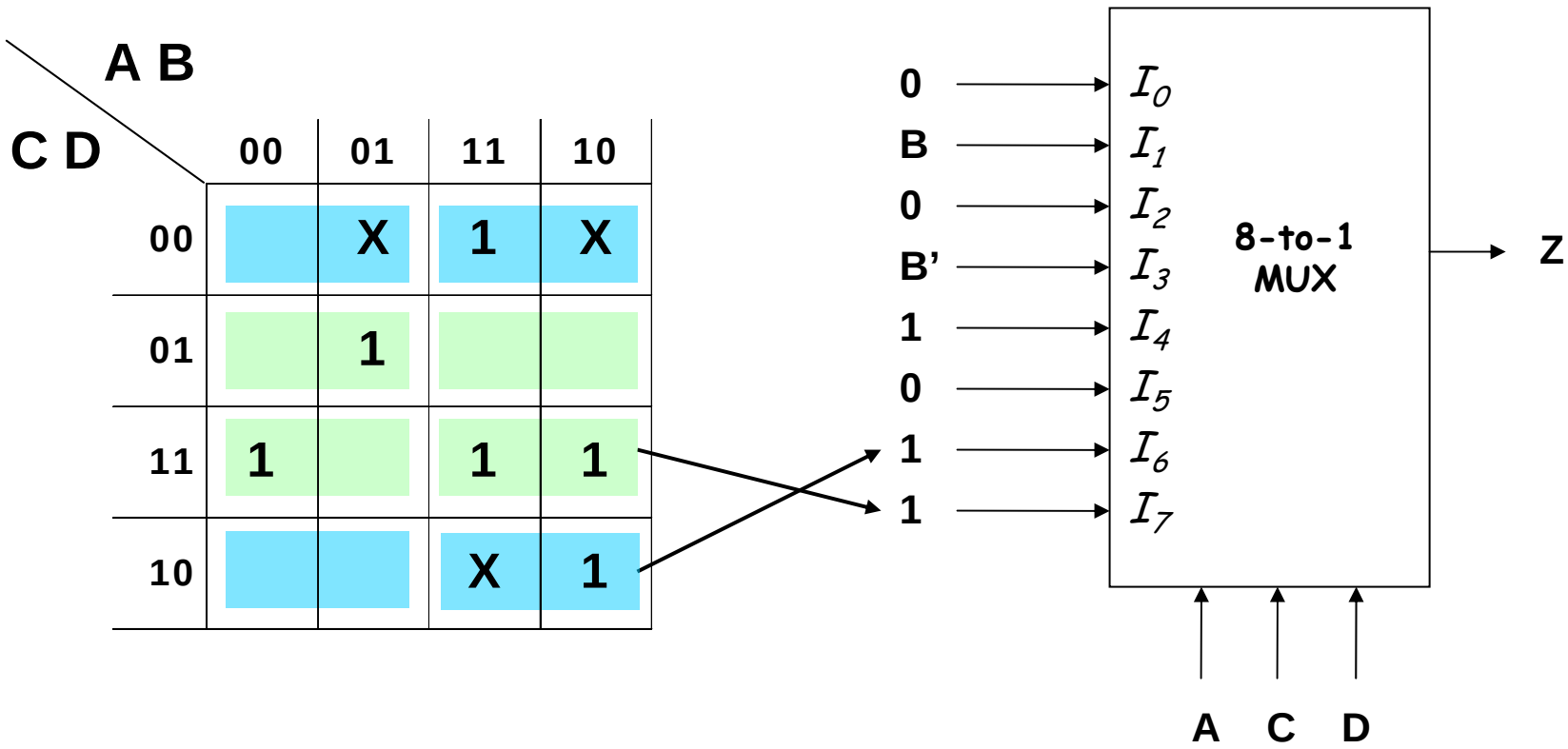
Implementing Logic Functions With Muxes

$$Z(A,B,C,D) = \sum m(3,5,10,11,12,15) + \sum d(4,8,14)$$



Implementing Logic Functions With Muxes

$$Z(A,B,C,D) = \sum m(3,5,10,11,12,15) + \sum d(4,8,14)$$

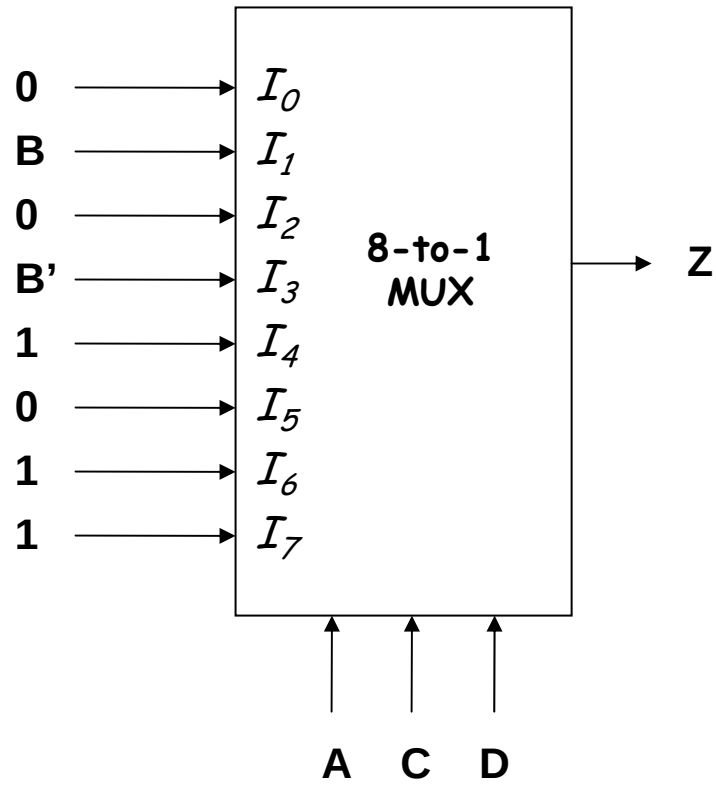


Implementing Logic Functions With Muxes

An alternate method

C D \ A B	00	01	11	10
00		X	1	X
01		1		
11	1		1	1
10			X	1

$Z = AC + AD' + A'BC' + B'CD$



Implementing Logic Functions With Muxes

An alternate method

A	C	D	Z =	AC	+	AD'	+	A'BC'	+	B'CD	
0	0	0	Z =	(0)(0)	+	(0)(1)	+	(1)B(1)	+	B'(0)(0)	= B
0	0	1	Z =	(0)(0)	+	(0)(0)	+	(1)B(1)	+	B'(0)(1)	= B
0	1	0	Z =	(0)(1)	+	(0)(1)	+	(1)B(0)	+	B'(1)(0)	= 0
0	1	1	Z =	(0)(1)	+	(0)(0)	+	(1)B(0)	+	B'(1)(1)	= B'
1	0	0	Z =	(1)(0)	+	(1)(1)	+	(0)B(1)	+	B'(0)(0)	= 1
1	0	1	Z =	(1)(0)	+	(1)(0)	+	(0)B(1)	+	B'(0)(1)	= 0
1	1	0	Z =	(1)(1)	+	(1)(1)	+	(0)B(0)	+	B'(1)(0)	= 1
1	1	1	Z =	(1)(1)	+	(1)(0)	+	(0)B(1)	+	B'(1)(1)	= 1

Implementing Logic Functions With Muxes

An alternate method

A	C	D	New Method	Original Method
0	0	0	Z = B	Z = 0
0	0	1	Z = B	Z = B
0	1	0	Z = 0	Z = 0
0	1	1	Z = B'	Z = B'
1	0	0	Z = 1	Z = 1
1	0	1	Z = 0	Z = 0
1	1	0	Z = 1	Z = 1
1	1	1	Z = 1	Z = 1

Why are they different?

Implementing Logic Functions With Muxes

An alternate method

C D \ A B	00	01	11	10
00		X	1	X
01		1		
11	1		1	1
10			X	1

The original method used this grouping. It was determined to be "0".

$$Z = AC + AD' + A'BC' + B'CD$$

Implementing Logic Functions With Muxes

An alternate method

C D \ A B	00	01	11	10
00		X	1	X
01		1		
11	1		1	1
10			X	1

The new method used this grouping. It was determined to be "1".

$$Z = AC + AD' + A'BC' + B'CD$$

Implementing Logic Functions With Muxes

An alternate method

C D \ A B	00	01	11	10
00		X	1	X
01		1		
11	1		1	1
10			X	1

The new method used this grouping.
It was determined to be "1".

Which is right??

$$Z = AC + AD' + A'BC' + B'CD$$

Implementing Logic Functions With Muxes

An alternate method

C D \ A B	00	01	11	10
00		X	1	X
01		1		
11	1		1	1
10			X	1

The new method used this grouping.
It was determined to be "1".

Which is right??
They both are!!!!

$$Z = AC + AD' + A'BC' + B'CD$$

Implementing Logic Functions With Muxes

An alternate method

C D \ A B	00	01	11	10
00		0 / 1	1	X
01		1		
11	1		1	1
10			X	1

For A=0 C=0 D=0

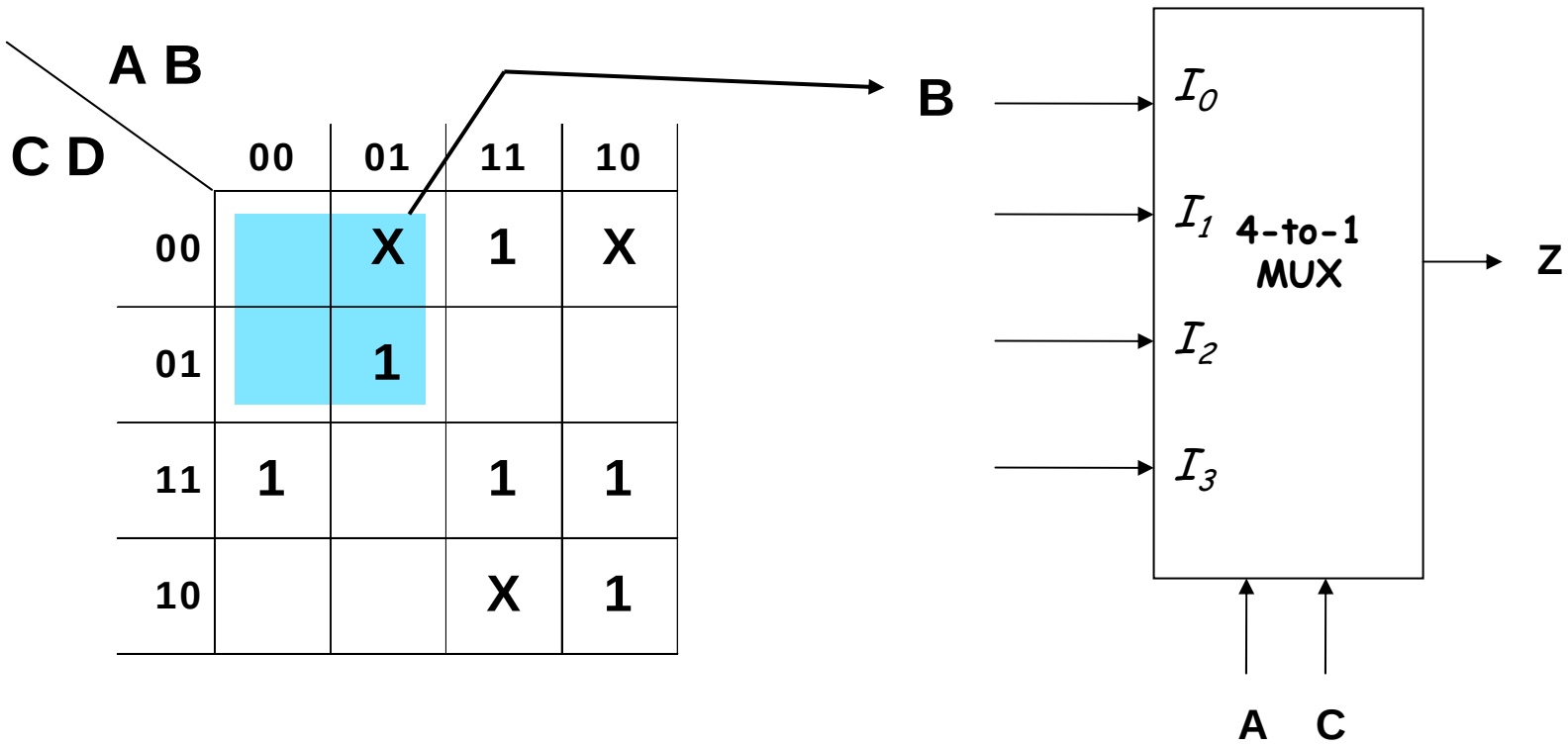
If A'BC'D' = 0 then input = 0

If A'BC'D' = 1 then input = B

They are both right!!!

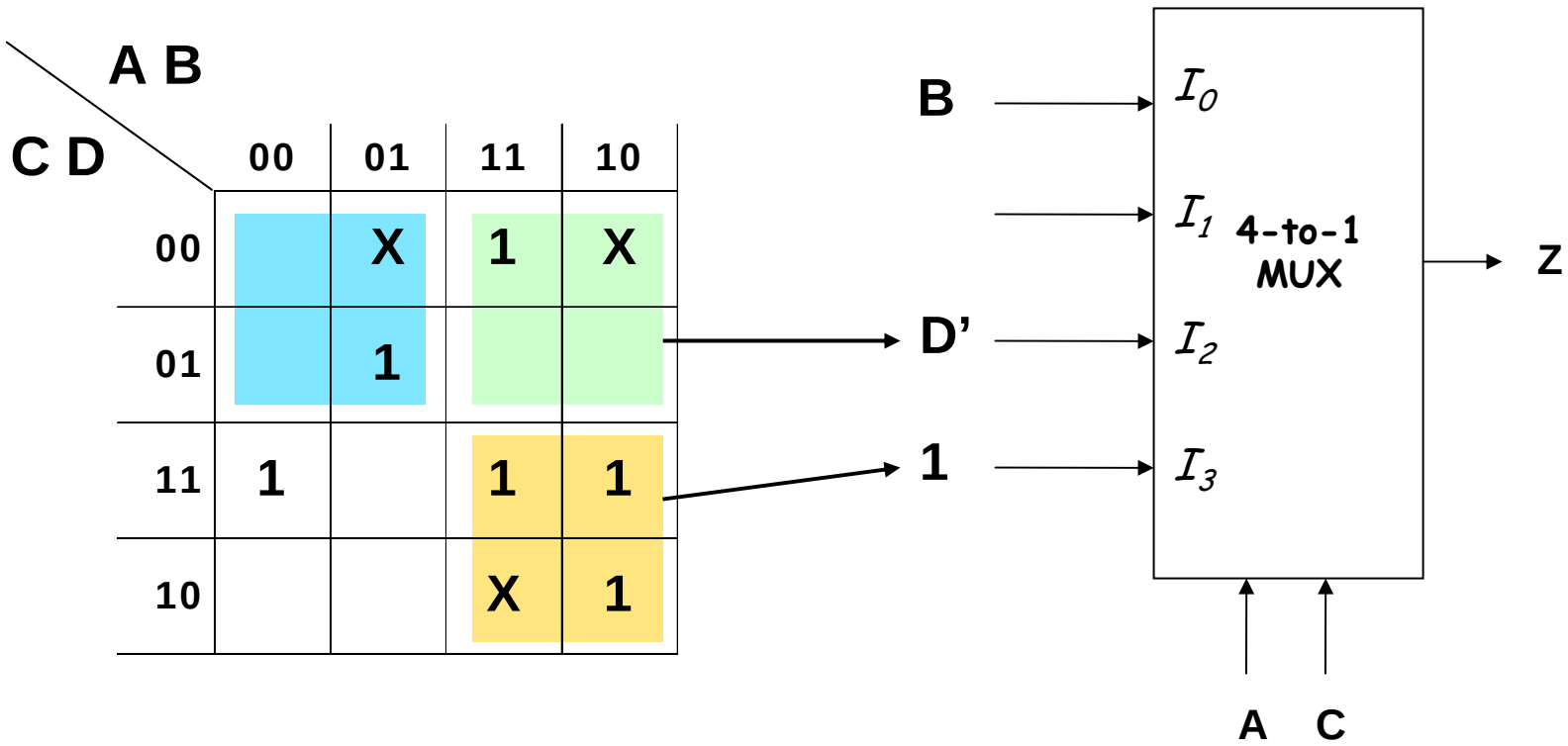
Implementing Logic Functions With Muxes

$$Z(A,B,C,D) = \sum m(3,5,10,11,12,15) + \sum d(4,8,14)$$



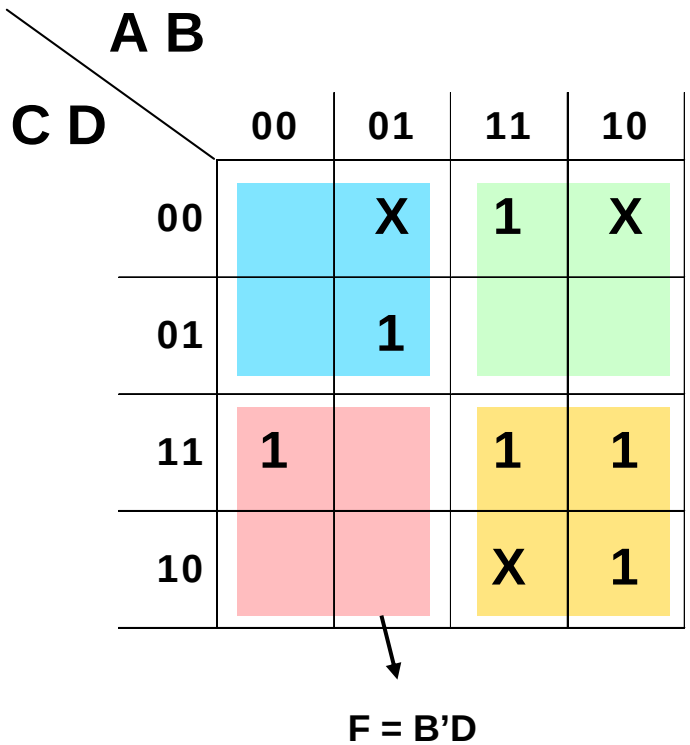
Implementing Logic Functions With Muxes

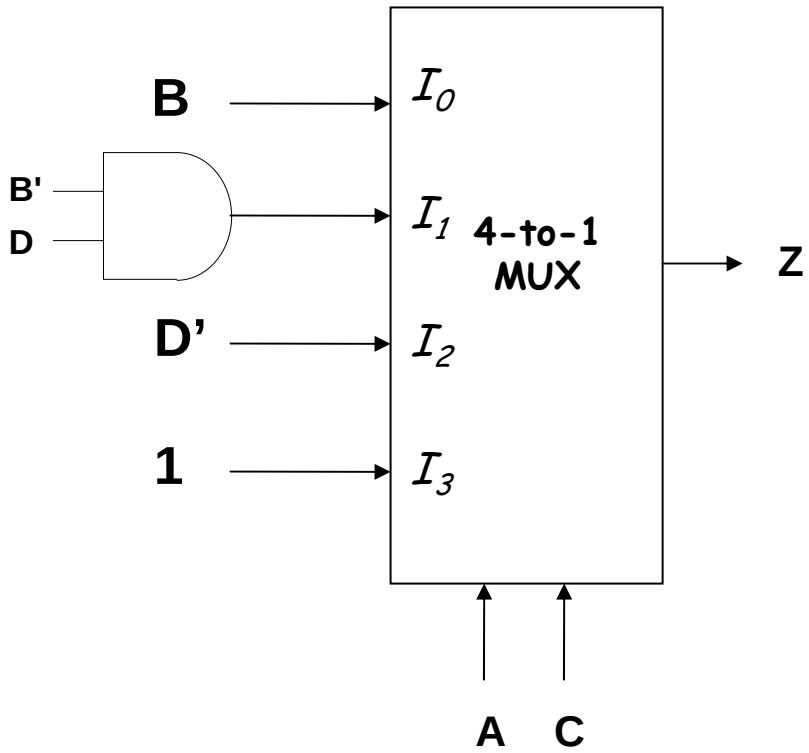
$$Z(A,B,C,D) = \sum m(3,5,10,11,12,15) + \sum d(4,8,14)$$



Implementing Logic Functions With Muxes

$$Z(A,B,C,D) = \sum m(3,5,10,11,12,15) + \sum d(4,8,14)$$





Implementing Logic Functions With Muxes

An alternate method

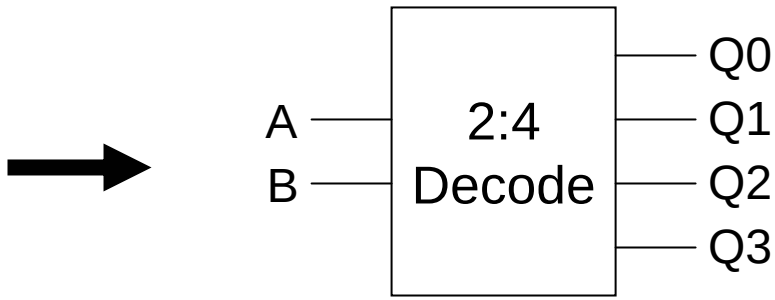
A C	Z = AC + AD' + A'BC' + B'CD	
0 0	Z = (0)(0) + (0)D' + (1)B(1) + B'(0)D	= B
0 1	Z = (0)(1) + (0)D' + (1)B(0) + B'(1)D	= B'D
1 0	Z = (1)(0) + (1)D' + (0)B(1) + B'(0)D	= D'
1 1	Z = (1)(1) + (1)D' + (0)B(0) + B'(1)D	= 1

Same as before!

Decoders

2:4 Decoder

A	B	Q0	Q1	Q2	Q3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

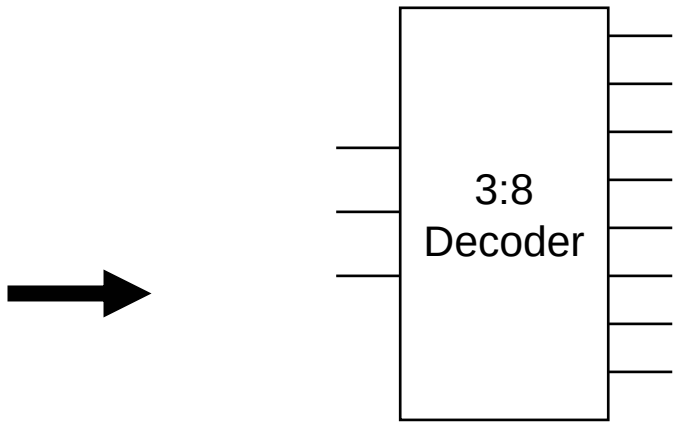


Q0 = A'B' m0
Q1 = A'B m1
Q2 = AB' m2
Q3 = AB m3

A decoder is a minterm generator…

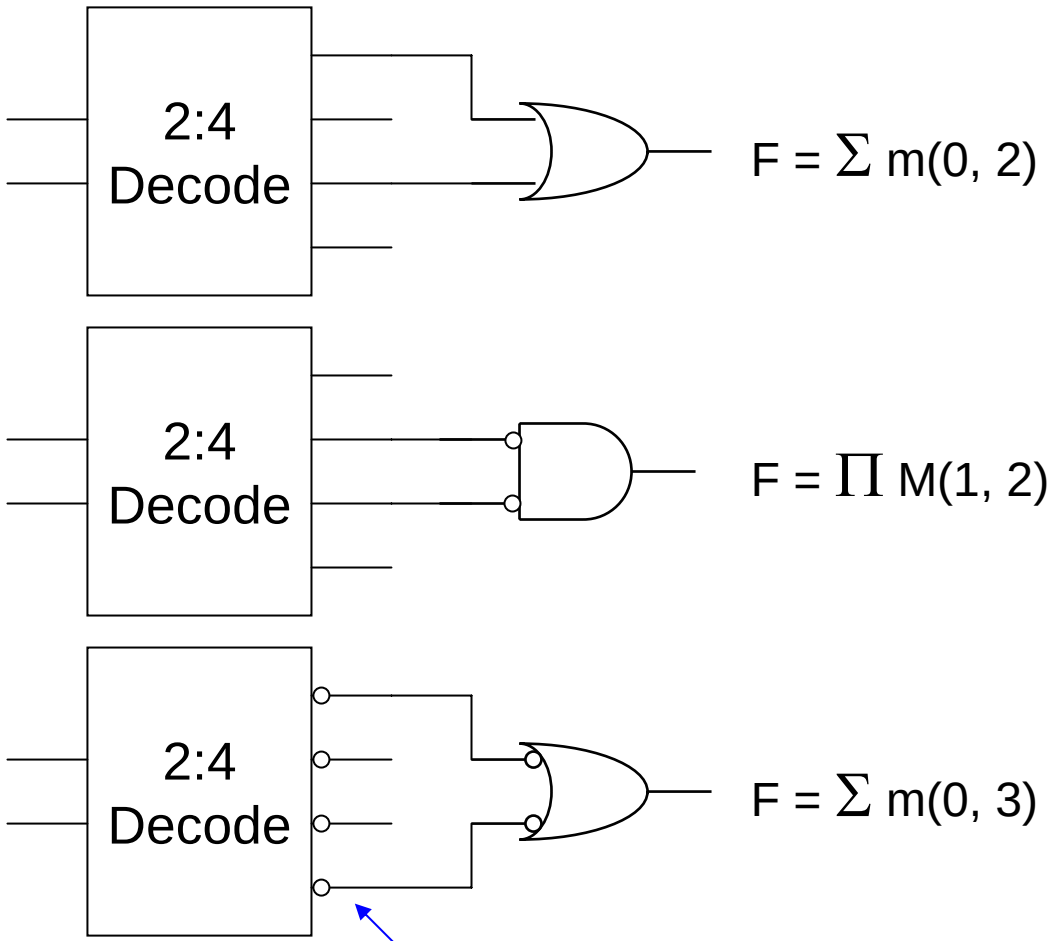
3:8 Decoder

A	B	C	Q0	Q1	Q2	Q3	Q4	Q5	Q6	Q7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1



Q0 = A'B'C' m0
Q1 = A'B'C m1
… …
Q7 = ABC m7

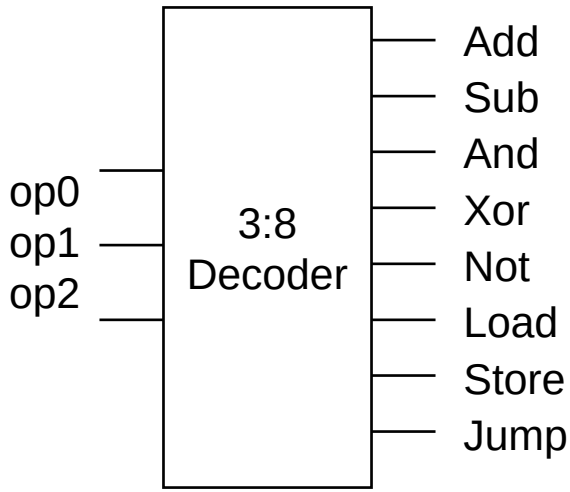
Implementing Logic With Decoders



Historically, some decoders have come with inverted outputs…

Uses of Decoders

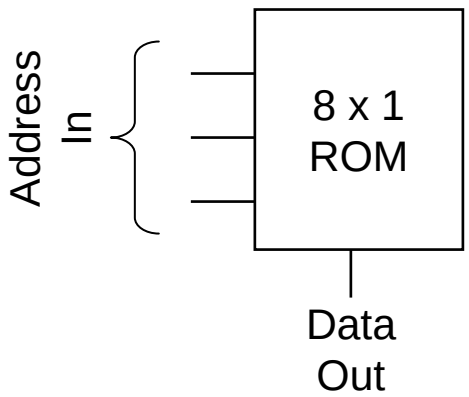
- Decode 3-bit op-code into a set of 8 signals



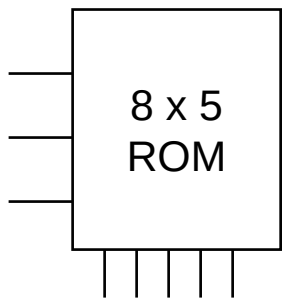
ROM: Read-Only Memory

- Can read values from it
- Cannot write to it

Address	Data
0	1
1	0
2	1
3	0
4	0
5	1
6	1
7	0

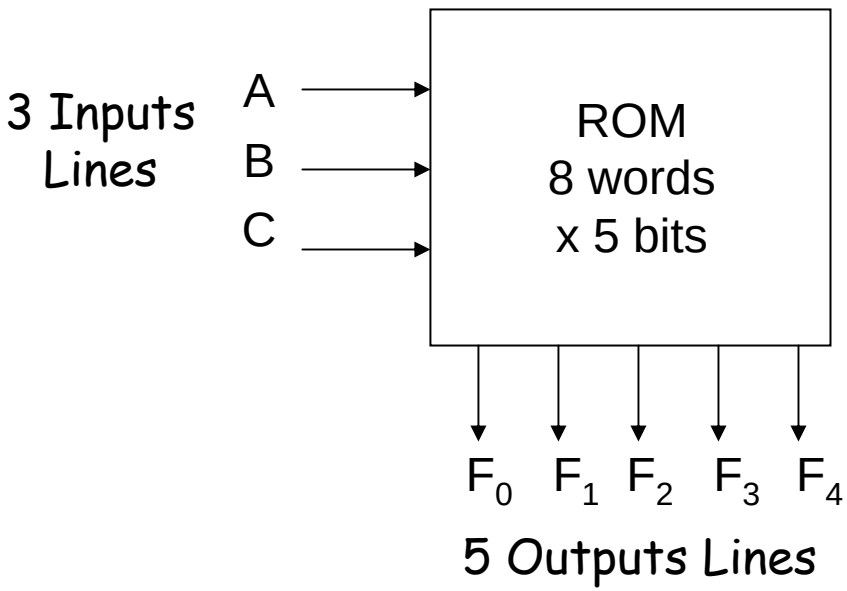


Address	Data
0	0
1	7
2	2
3	14
4	26
5	0
6	18
7	22



Read Only Memory (ROM)

Each minterm of each function can be specified

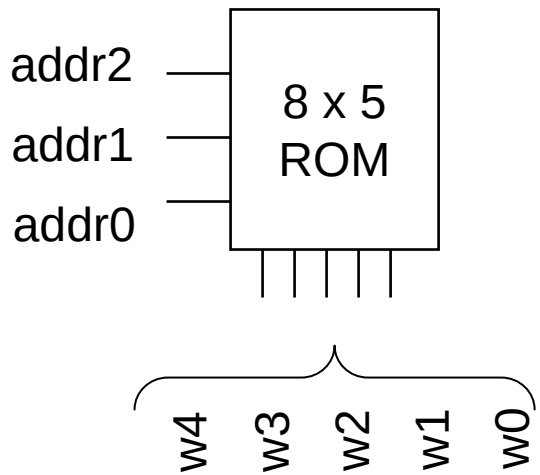


A	B	C	F_0	F_1	F_2	F_3	F_4
0	0	0	0	0	0	0	1
0	0	1	0	0	1	1	0
0	1	0	0	0	1	1	0
0	1	1	1	0	1	0	1
1	0	0	0	0	1	1	0
1	0	1	0	0	1	0	1
1	1	0	1	0	1	0	1
1	1	1	1	1	1	1	0

When you program a ROM, you are specifying these

ROM – View #1

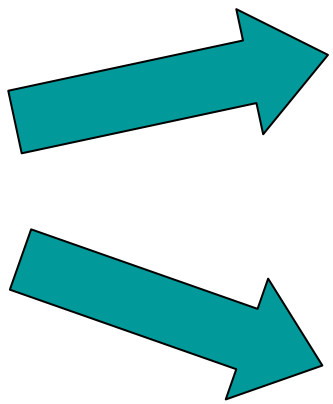
- An addressable memory
  - Send in address ($addr_n$)
  - Receive data stored at that location (w_n)
  - Can have multi-bit data

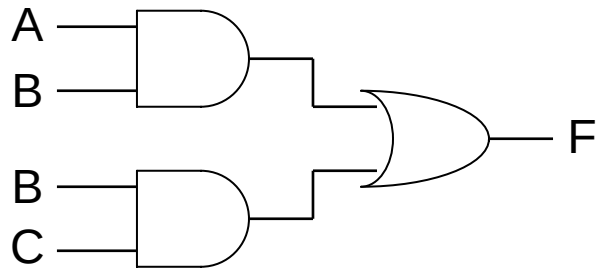


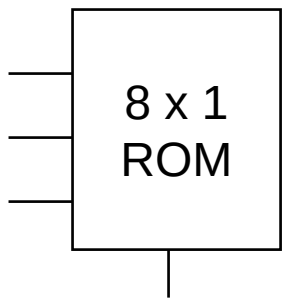
ROM – View #2

- A hardware implementation of a truth table

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



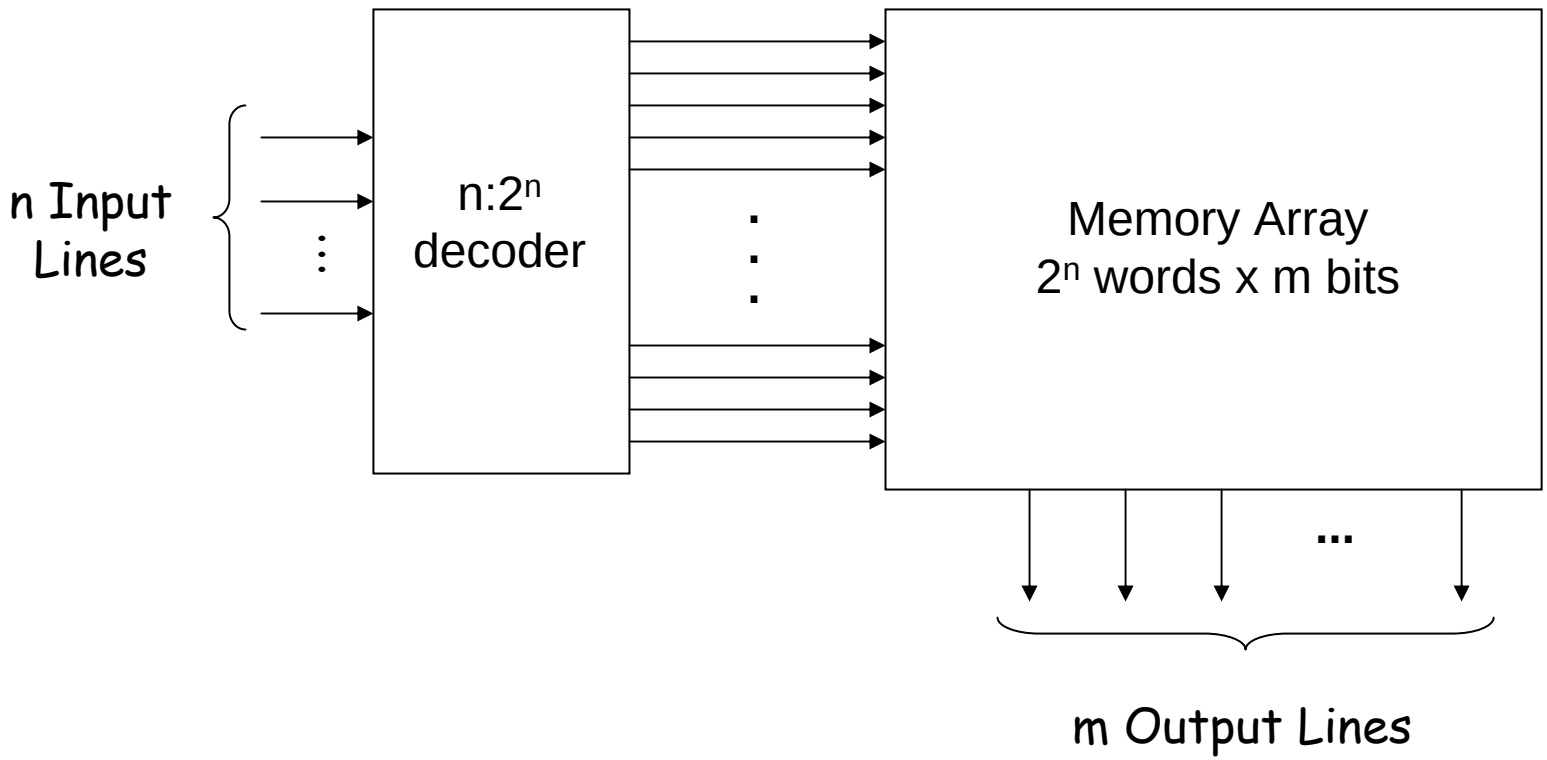




ROM – Two Different Views

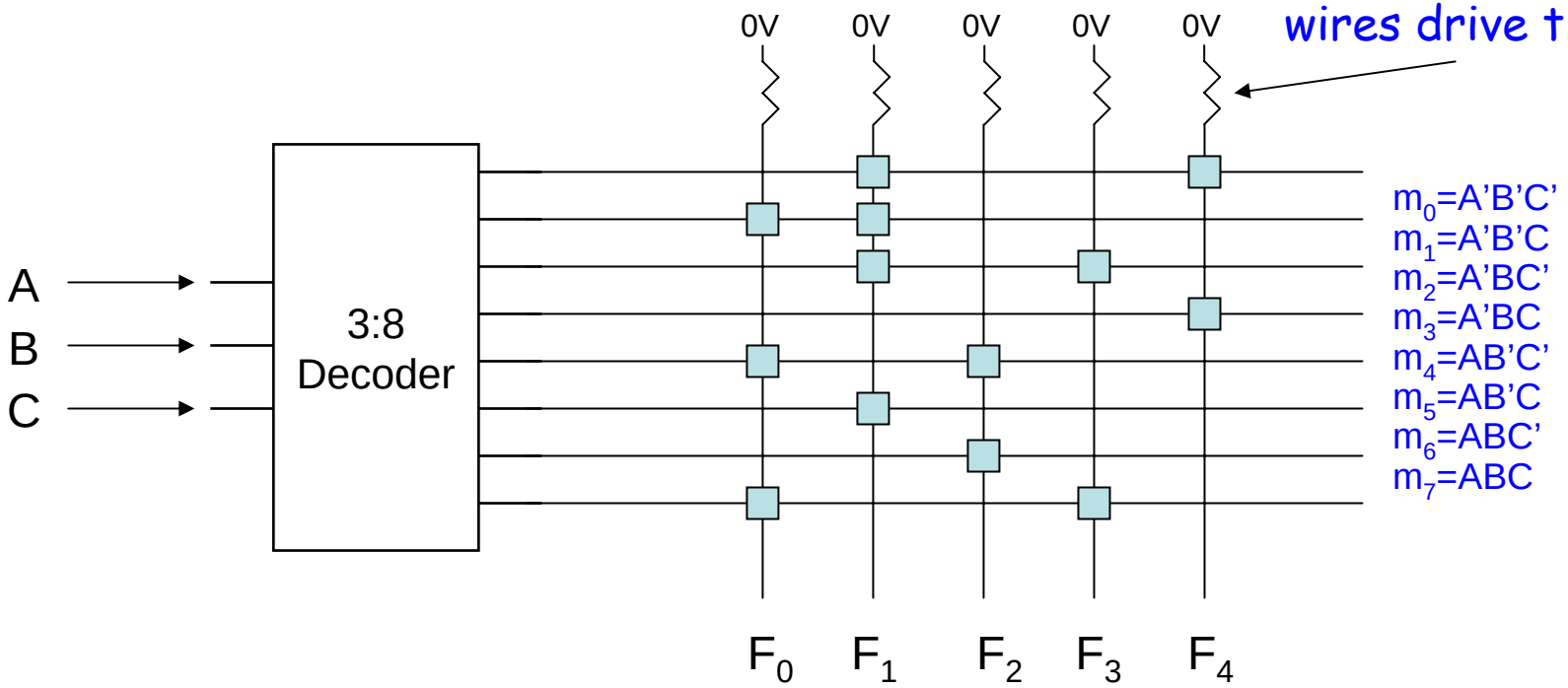
- Both views are accurate

ROM Internal Structure

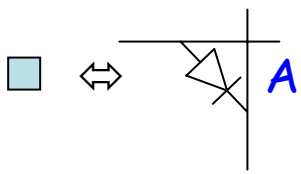


ROM Memory Array

Resistors – pull outputs down to GND if no row wires drive them high



F_0 = m1 + m4 + m7
F_1 = m0 + m1 + m2 + m5
…



A diode – a one-way 'resistor'

ROM Technologies: Not Always Diode-Based

- Program Only Once
  - Mask programmable
  - Fusible Link
- Re-Programmable
  - EPROM (ultraviolet erase)
  - EE-PROM (electrically erase)
  - Flash memory

Beyond the scope of this class

Using a ROM For Logic

F = AB + A'BC'

G = A'B'C + C'

H = AB'C' + ABC' + A'B'C

A	B	C	F	G	H
0	0	0			
0	0	1			
0	1	0			
0	1	1			
1	0	0			
1	0	1			
1	1	0			
1	1	1			

Using a ROM For Logic

F = AB + A'BC'

G = A'B'C + C'

H = AB'C' + ABC' + A'B'C

A	B	C	F	G	H
0	0	0	0		
0	0	1	0		
0	1	0	1		
0	1	1	0		
1	0	0	0		
1	0	1	0		
1	1	0	1		
1	1	1	1		

Using a ROM For Logic

F = AB + A'BC'

G = A'B'C + C'

H = AB'C' + ABC' + A'B'C

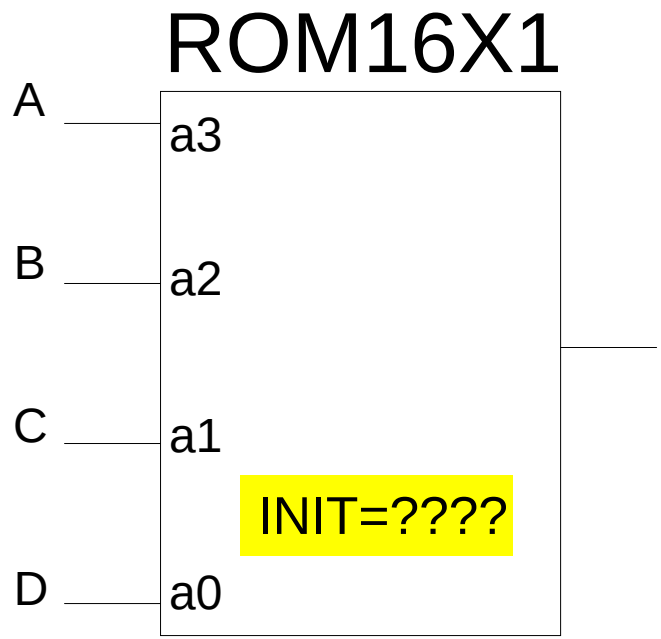
A	B	C	F	G	H
0	0	0	0	1	0
0	0	1	0	1	1
0	1	0	1	1	0
0	1	1	0	0	0
1	0	0	0	1	1
1	0	1	0	0	0
1	1	0	1	1	1
1	1	1	1	0	0

Just fill out the truth table

CAD tools usually do the rest…

Using ROM for Combinational Logic

A	B	C	D	Q
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
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	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1



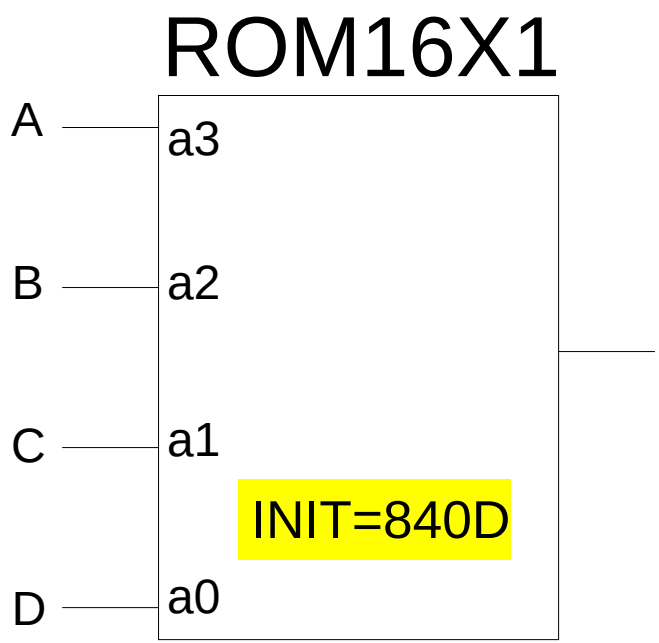
- ROM16X1 is a Xilinx cell
- Insert it into schematics like any other primitive
- Set its contents by double-clicking on inserted cell in schematic.

Using ROM for Combinational Logic

A	B	C	D	Q
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
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	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

1000 0100 0000 1101

8 4 0 D



ROM Types in Xilinx

- ROM16X1 = 4-input LUT
  - Specify INIT contents with 4-digit HEX value
- ROM32X1 = 5-input LUT
  - Specify INIT contents with 8-digit HEX value