

Digital Circuits: Part 3



M. B. Patil

mbpatil@ee.iitb.ac.in

www.ee.iitb.ac.in/~sequel

Department of Electrical Engineering
Indian Institute of Technology Bombay

Decimal (base 10) system

$$317 = 3 \times 10^2 + 1 \times 10^1 + 7 \times 10^0$$

10^2 10^1 10^0

Decimal (base 10) system

$$317 = 3 \times 10^2 + 1 \times 10^1 + 7 \times 10^0$$

10^2 10^1 10^0

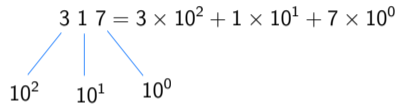
* Digits: 0,1,2,...,9

* example: 4153

most significant
digit

least significant
digit

Decimal (base 10) system

$$317 = 3 \times 10^2 + 1 \times 10^1 + 7 \times 10^0$$


* Digits: 0,1,2,...,9

* example: 4153

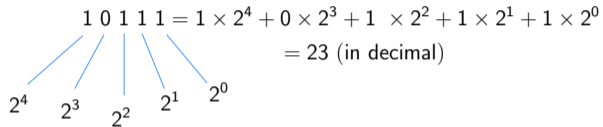
most significant
digit

least significant
digit

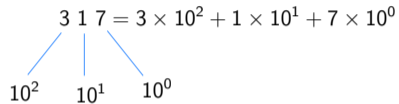
Binary (base 2) system

$$10111 = 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$$

= 23 (in decimal)



Decimal (base 10) system

$$317 = 3 \times 10^2 + 1 \times 10^1 + 7 \times 10^0$$


* Digits: 0,1,2,...,9

* example: 4153

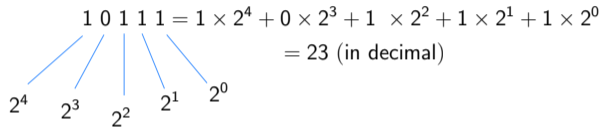
most significant
digit

least significant
digit

Binary (base 2) system

$$10111 = 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$$

= 23 (in decimal)



* Bits: 0,1

* example: 100110

most significant
bit (MSB)

least significant
bit (LSB)

Addition of binary numbers

Decimal (base 10) system

	10^4	10^3	10^2	10^1	10^0	weight
		3	1	7	9	first number
+		8	0	1	5	second number
	1			1		carry
<hr/>						
	1	1	1	9	4	sum

Addition of binary numbers

Decimal (base 10) system

	10^4	10^3	10^2	10^1	10^0	weight
		3	1	7	9	first number
+		8	0	1	5	second number
	1			1		carry
<hr/>						
	1	1	1	9	4	sum

Binary (base 2) system

	2^4	2^3	2^2	2^1	2^0	weight
		1	0	1	1	first number (dec. 11)
+		1	1	1	0	second number (dec. 14)
	1	1	1			carry
<hr/>						
	1	1	0	0	1	sum (dec. 25)

Addition of binary numbers

Decimal (base 10) system

	10^4	10^3	10^2	10^1	10^0	weight
		3	1	7	9	first number
+		8	0	1	5	second number
	1			1		carry
<hr/>						
	1	1	1	9	4	sum

Binary (base 2) system

	2^4	2^3	2^2	2^1	2^0	weight
		1	0	1	1	first number (dec. 11)
+		1	1	1	0	second number (dec. 14)
	1	1	1			carry
<hr/>						
	1	1	0	0	1	sum (dec. 25)

* $0 + 1 = 1 + 0 = 1 \rightarrow S = 1, C = 0$

Addition of binary numbers

Decimal (base 10) system

	10^4	10^3	10^2	10^1	10^0	weight
		3	1	7	9	first number
+		8	0	1	5	second number
	1			1		carry
<hr/>						
	1	1	1	9	4	sum

Binary (base 2) system

	2^4	2^3	2^2	2^1	2^0	weight
		1	0	1	1	first number (dec. 11)
+		1	1	1	0	second number (dec. 14)
	1	1	1			carry
<hr/>						
	1	1	0	0	1	sum (dec. 25)

* $0 + 1 = 1 + 0 = 1 \rightarrow S = 1, C = 0$

* $1 + 1 = 10 \text{ (dec. 2)} \rightarrow S = 0, C = 1$

Addition of binary numbers

Decimal (base 10) system

	10^4	10^3	10^2	10^1	10^0	weight
		3	1	7	9	first number
+		8	0	1	5	second number
	1			1		carry
<hr/>						
	1	1	1	9	4	sum

Binary (base 2) system

	2^4	2^3	2^2	2^1	2^0	weight
		1	0	1	1	first number (dec. 11)
+		1	1	1	0	second number (dec. 14)
	1	1	1			carry
<hr/>						
	1	1	0	0	1	sum (dec. 25)

* $0 + 1 = 1 + 0 = 1 \rightarrow S = 1, C = 0$

* $1 + 1 = 10$ (dec. 2) $\rightarrow S = 0, C = 1$

* $1 + 1 + 1 = 11$ (dec. 3) $\rightarrow S = 1, C = 1$

Addition of binary numbers

example

	2^4	2^3	2^2	2^1	2^0	weight
		1	0	1	1	first number
+		1	1	1	0	second number
	1	1	1	0	-	carry
<hr/>						
	1	1	0	0	1	sum

Addition of binary numbers

example

	2^4	2^3	2^2	2^1	2^0	weight
		1	0	1	1	first number
+		1	1	1	0	second number
	1	1	1	0	-	carry
<hr/>						
	1	1	0	0	1	sum

general procedure

	2^N		2^2	2^1	2^0	weight
	A_N	...	A_2	A_1	A_0	first number
	B_N	...	B_2	B_1	B_0	second number
C_N	C_{N-1}	...	C_1	C_0		carry
<hr/>						
	S_N		S_2	S_1	S_0	sum

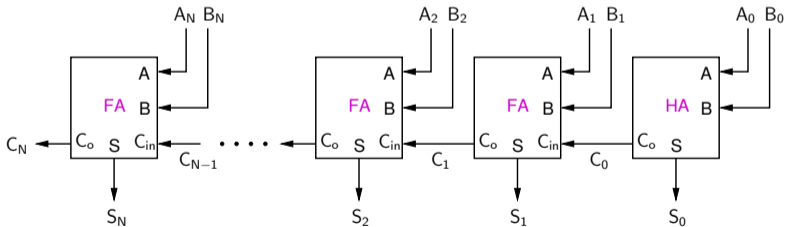
Addition of binary numbers

example

	2^4	2^3	2^2	2^1	2^0	weight
		1	0	1	1	first number
+		1	1	1	0	second number
	1	1	1	0	-	carry
<hr/>						
	1	1	0	0	1	sum

general procedure

	2^N	...	2^2	2^1	2^0	weight
	A_N	...	A_2	A_1	A_0	first number
+	B_N	...	B_2	B_1	B_0	second number
	C_N	C_{N-1}	...	C_1	C_0	carry
<hr/>						
	S_N		S_2	S_1	S_0	sum



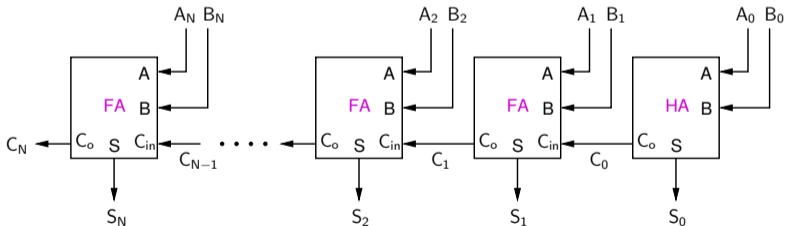
Addition of binary numbers

example

	2^4	2^3	2^2	2^1	2^0	weight
		1	0	1	1	first number
+		1	1	1	0	second number
	1	1	1	0	-	carry
<hr/>						
	1	1	0	0	1	sum

general procedure

	2^N	...	2^2	2^1	2^0	weight
	A_N	...	A_2	A_1	A_0	first number
+	B_N	...	B_2	B_1	B_0	second number
	C_N	C_{N-1}	...	C_1	C_0	carry
<hr/>						
	S_N		S_2	S_1	S_0	sum



- * The rightmost block (corresponding to the LSB) adds two bits A_0 and B_0 ; there is no input carry. This block is called a "half adder."

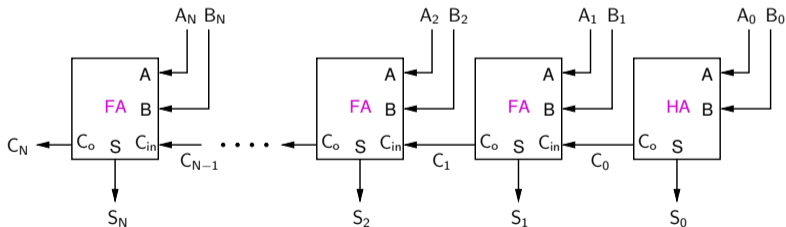
Addition of binary numbers

example

	2^4	2^3	2^2	2^1	2^0	weight
		1	0	1	1	first number
+		1	1	1	0	second number
	1	1	1	0	-	carry
<hr/>						
	1	1	0	0	1	sum

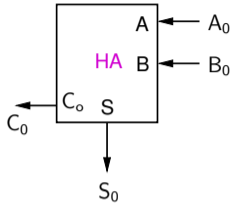
general procedure

	2^N	...	2^2	2^1	2^0	weight
	A_N	...	A_2	A_1	A_0	first number
+	B_N	...	B_2	B_1	B_0	second number
	C_N	C_{N-1}	...	C_1	C_0	carry
<hr/>						
	S_N		S_2	S_1	S_0	sum



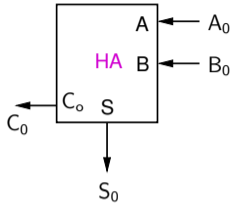
- * The rightmost block (corresponding to the LSB) adds two bits A_0 and B_0 ; there is no input carry. This block is called a "half adder."
- * Each of the subsequent blocks adds three bits (A_i, B_i, C_{i-1}) and is called a "full adder."

Half adder implementation



A	B	C_0	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Half adder implementation



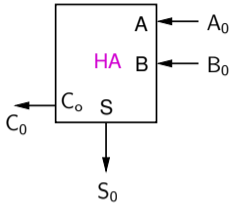
A	B	C _o	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0



$$S = \bar{A}B + A\bar{B} = A \oplus B$$

$$C_o = AB$$

Half adder implementation



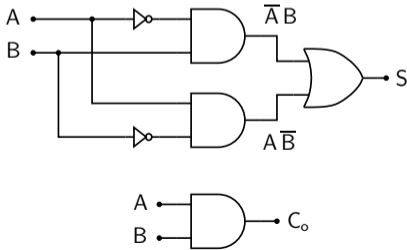
A	B	C _o	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0



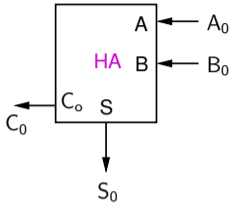
$$S = \bar{A}B + A\bar{B} = A \oplus B$$

$$C_o = AB$$

Implementation 1



Half adder implementation



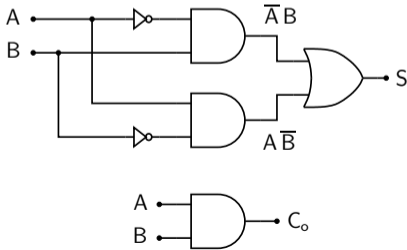
A	B	C _o	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0



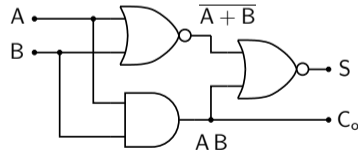
$$S = \bar{A}B + A\bar{B} = A \oplus B$$

$$C_o = AB$$

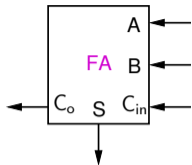
Implementation 1



Implementation 2

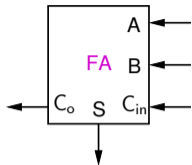


Full adder implementation



A	B	C _{in}	C _o	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Full adder implementation



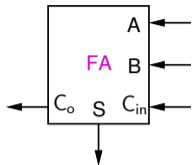
A	B	C _{in}	C _o	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

S:

	AB			
C _{in}	00	01	11	10
0	0	1	0	1
1	1	0	1	0

$$S = \bar{A}B\bar{C}_{in} + A\bar{B}\bar{C}_{in} + \bar{A}\bar{B}C_{in} + ABC_{in}$$

Full adder implementation



A	B	C _{in}	C _o	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

S:

	AB			
C _{in}	00	01	11	10
0	0	1	0	1
1	1	0	1	0

$$S = \bar{A}\bar{B}\bar{C}_{in} + A\bar{B}\bar{C}_{in} + \bar{A}B\bar{C}_{in} + ABC_{in}$$

C_o:

	AB			
C _{in}	00	01	11	10
0	0	0	1	0
1	0	1	1	1

$$C_o = AB + BC_{in} + AC_{in}$$

Implementation of functions with only NAND gates

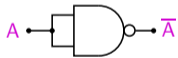
The NOT, AND, OR operations can be realised by using only NAND gates:

Implementation of functions with only NAND gates

The NOT, AND, OR operations can be realised by using only NAND gates:

NOT

$$\bar{A} = \overline{A \cdot A}$$

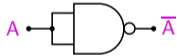


Implementation of functions with only NAND gates

The NOT, AND, OR operations can be realised by using only NAND gates:

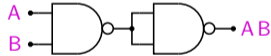
NOT

$$\bar{A} = \overline{A \cdot A}$$



AND

$$A \cdot B = \overline{\overline{A \cdot B}}$$

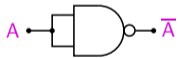


Implementation of functions with only NAND gates

The NOT, AND, OR operations can be realised by using only NAND gates:

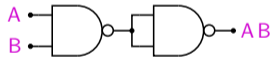
NOT

$$\bar{A} = \overline{A \cdot A}$$



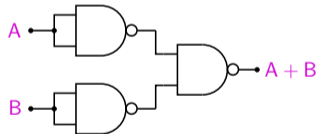
AND

$$A \cdot B = \overline{\overline{A \cdot B}}$$



OR

$$A + B = \overline{\overline{A \cdot B}}$$



Implementation of functions with only NAND gates

Implement $Y = AB + BC\bar{D} + \bar{A}D$ using only NAND gates.

Implementation of functions with only NAND gates

Implement $Y = AB + BC\bar{D} + \bar{A}D$ using only NAND gates.

$$\bar{A} = \overline{A \cdot A}$$

$$A \cdot B = \overline{\overline{A \cdot B}}$$

$$A + B = \overline{\overline{A \cdot B}}$$

Implementation of functions with only NAND gates

Implement $Y = AB + BC\bar{D} + \bar{A}D$ using only NAND gates.

$$Y = \overline{\overline{AB} \cdot \overline{BC\bar{D}} \cdot \overline{\bar{A}D}}$$

$$\bar{A} = \overline{A \cdot A}$$

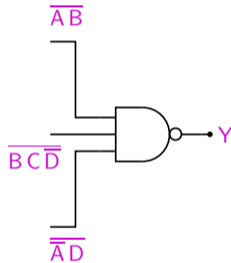
$$A \cdot B = \overline{\overline{A \cdot B}}$$

$$A + B = \overline{\overline{A} \cdot \overline{B}}$$

Implementation of functions with only NAND gates

Implement $Y = AB + BC\bar{D} + \bar{A}D$ using only NAND gates.

$$Y = \overline{\overline{AB} \cdot \overline{BC\bar{D}} \cdot \overline{\bar{A}D}}$$



$$\bar{A} = \overline{A \cdot A}$$

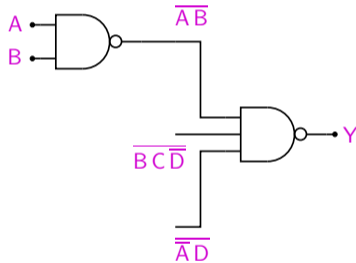
$$A \cdot B = \overline{\overline{A \cdot B}}$$

$$A + B = \overline{\overline{A \cdot B}}$$

Implementation of functions with only NAND gates

Implement $Y = AB + BC\bar{D} + \bar{A}D$ using only NAND gates.

$$Y = \overline{\overline{AB} \cdot \overline{BC\bar{D}} \cdot \overline{\bar{A}D}}$$



$$\bar{A} = \overline{A \cdot A}$$

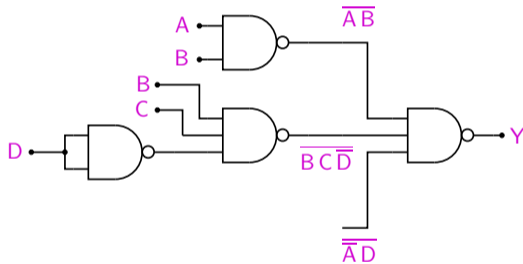
$$A \cdot B = \overline{\overline{A \cdot B}}$$

$$A + B = \overline{\overline{A \cdot B}}$$

Implementation of functions with only NAND gates

Implement $Y = AB + BC\bar{D} + \bar{A}D$ using only NAND gates.

$$Y = \overline{\overline{AB} \cdot \overline{BC\bar{D}} \cdot \overline{\bar{A}D}}$$



$$\bar{A} = \overline{A \cdot A}$$

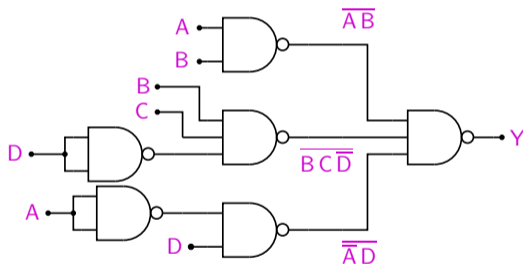
$$A \cdot B = \overline{\overline{A \cdot B}}$$

$$A + B = \overline{\overline{A} \cdot \overline{B}}$$

Implementation of functions with only NAND gates

Implement $Y = AB + BC\bar{D} + \bar{A}D$ using only NAND gates.

$$Y = \overline{\overline{AB} \cdot \overline{BC\bar{D}} \cdot \overline{\bar{A}D}}$$



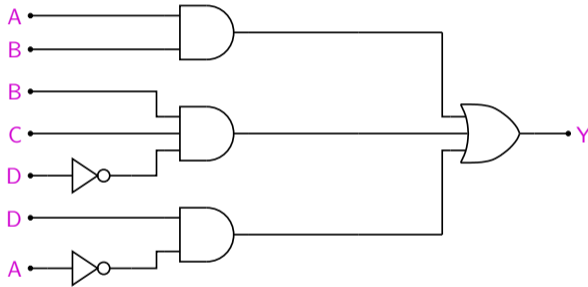
$$\bar{A} = \overline{A \cdot A}$$

$$A \cdot B = \overline{\overline{A \cdot B}}$$

$$A + B = \overline{\overline{A \cdot B}}$$

Implementation of functions with only NAND gates

Implement $Y = AB + BC\bar{D} + \bar{A}D$ using only NAND gates.



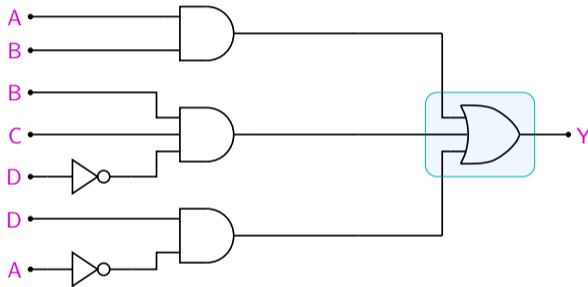
$$\bar{A} = \overline{A \cdot A}$$

$$A \cdot B = \overline{\overline{A \cdot B}}$$

$$A + B = \overline{\overline{A} \cdot \overline{B}}$$

Implementation of functions with only NAND gates

Implement $Y = AB + BC\bar{D} + \bar{A}D$ using only NAND gates.



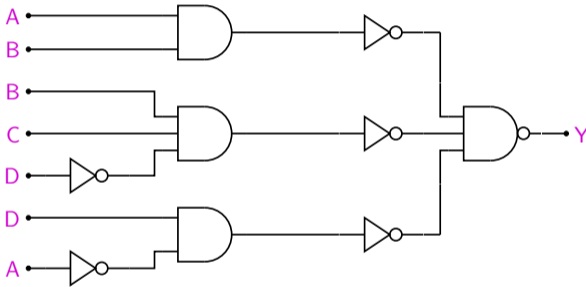
$$\bar{A} = \overline{A \cdot A}$$

$$A \cdot B = \overline{\overline{A \cdot B}}$$

$$A + B = \overline{\overline{A} \cdot \overline{B}}$$

Implementation of functions with only NAND gates

Implement $Y = AB + BC\bar{D} + \bar{A}D$ using only NAND gates.



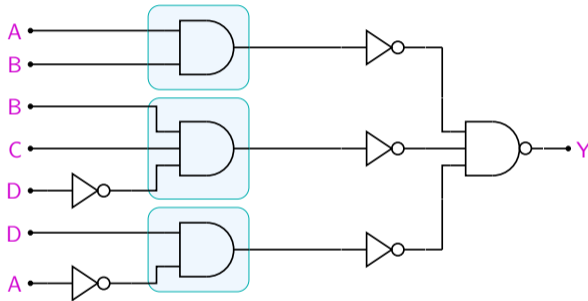
$$\bar{A} = \overline{A \cdot A}$$

$$A \cdot B = \overline{\overline{A \cdot B}}$$

$$A + B = \overline{\overline{A} \cdot \overline{B}}$$

Implementation of functions with only NAND gates

Implement $Y = AB + BC\bar{D} + \bar{A}D$ using only NAND gates.



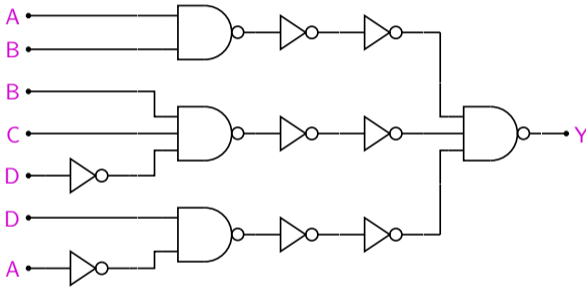
$$\bar{A} = \overline{A \cdot A}$$

$$A \cdot B = \overline{\overline{A \cdot B}}$$

$$A + B = \overline{\overline{A} \cdot \overline{B}}$$

Implementation of functions with only NAND gates

Implement $Y = AB + BC\bar{D} + \bar{A}D$ using only NAND gates.



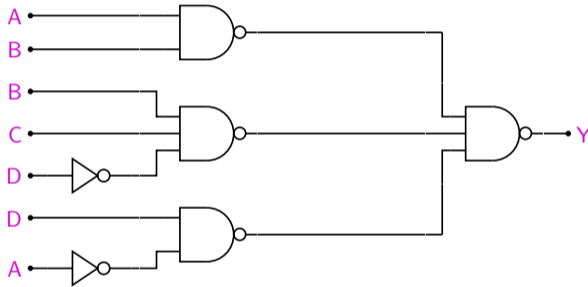
$$\bar{A} = \overline{A \cdot A}$$

$$A \cdot B = \overline{\overline{A \cdot B}}$$

$$A + B = \overline{\overline{A} \cdot \overline{B}}$$

Implementation of functions with only NAND gates

Implement $Y = AB + BC\bar{D} + \bar{A}D$ using only NAND gates.



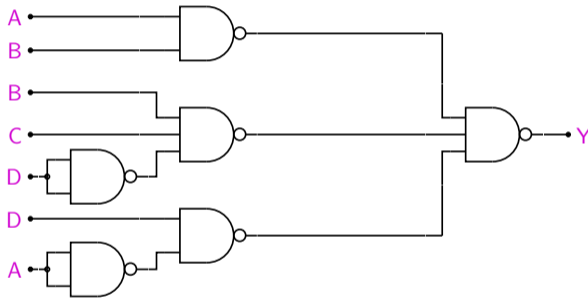
$$\bar{A} = \overline{A \cdot A}$$

$$A \cdot B = \overline{\overline{A \cdot B}}$$

$$A + B = \overline{\overline{A} \cdot \overline{B}}$$

Implementation of functions with only NAND gates

Implement $Y = AB + BC\bar{D} + \bar{A}D$ using only NAND gates.



$$\bar{A} = \overline{A \cdot A}$$

$$A \cdot B = \overline{\overline{A \cdot B}}$$

$$A + B = \overline{\overline{A} \cdot \overline{B}}$$

Implementation of functions with only NAND gates

Implement $Y = A + B + C$ using only 2-input NAND gates.

Implementation of functions with only NAND gates

Implement $Y = A + B + C$ using only 2-input NAND gates.

$$\bar{A} = \overline{A \cdot A}$$

$$A \cdot B = \overline{\overline{A \cdot B}}$$

$$A + B = \overline{\overline{A \cdot B}}$$

Implementation of functions with only NAND gates

Implement $Y = A + B + C$ using only 2-input NAND gates.

$$\begin{aligned} Y &= (A + B) + C \\ &= \overline{\overline{(A + B)} \cdot \overline{C}} \end{aligned}$$

$$\overline{A} = \overline{A \cdot A}$$

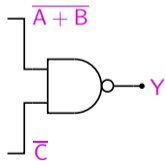
$$A \cdot B = \overline{\overline{A \cdot B}}$$

$$A + B = \overline{\overline{A} \cdot \overline{B}}$$

Implementation of functions with only NAND gates

Implement $Y = A + B + C$ using only 2-input NAND gates.

$$Y = (A + B) + C$$
$$= \overline{\overline{(A + B)} \cdot \overline{C}}$$



$$\overline{A} = \overline{A \cdot A}$$

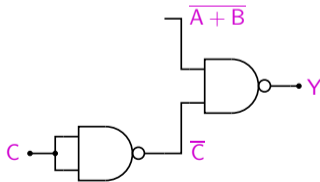
$$A \cdot B = \overline{\overline{A} \cdot \overline{B}}$$

$$A + B = \overline{\overline{A} \cdot \overline{B}}$$

Implementation of functions with only NAND gates

Implement $Y = A + B + C$ using only 2-input NAND gates.

$$\begin{aligned} Y &= (A + B) + C \\ &= \overline{\overline{(A + B)} \cdot \overline{C}} \end{aligned}$$



$$\overline{A} = \overline{A \cdot A}$$

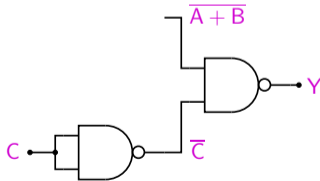
$$A \cdot B = \overline{\overline{A} \cdot \overline{B}}$$

$$A + B = \overline{\overline{A} \cdot \overline{B}}$$

Implementation of functions with only NAND gates

Implement $Y = A + B + C$ using only 2-input NAND gates.

$$\begin{aligned} Y &= (A + B) + C \\ &= \overline{\overline{(A + B)} \cdot \overline{C}} \\ &= \overline{\overline{\overline{A} \cdot \overline{B}} \cdot \overline{C}} \end{aligned}$$



$$\overline{A} = \overline{A \cdot A}$$

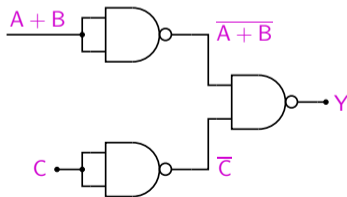
$$A \cdot B = \overline{\overline{A} \cdot \overline{B}}$$

$$A + B = \overline{\overline{A} \cdot \overline{B}}$$

Implementation of functions with only NAND gates

Implement $Y = A + B + C$ using only 2-input NAND gates.

$$\begin{aligned} Y &= (A + B) + C \\ &= \overline{\overline{(A + B)} \cdot \overline{C}} \\ &= \overline{\overline{\overline{A} \cdot \overline{B}} \cdot \overline{C}} \end{aligned}$$



$$\overline{A} = \overline{A \cdot A}$$

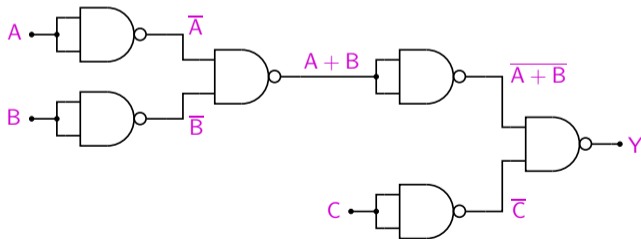
$$A \cdot B = \overline{\overline{A} \cdot \overline{B}}$$

$$A + B = \overline{\overline{A} \cdot \overline{B}}$$

Implementation of functions with only NAND gates

Implement $Y = A + B + C$ using only 2-input NAND gates.

$$\begin{aligned} Y &= (A + B) + C \\ &= \overline{\overline{(A + B)} \cdot \overline{C}} \\ &= \overline{\overline{\overline{A} \cdot \overline{B}} \cdot \overline{C}} \end{aligned}$$



$$\overline{A} = \overline{A \cdot A}$$

$$A \cdot B = \overline{\overline{A} \cdot \overline{B}}$$

$$A + B = \overline{\overline{A} \cdot \overline{B}}$$

Implementation of functions with only NOR gates

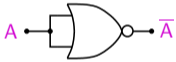
The NOT, AND, OR operations can be realised by using only NOR gates:

Implementation of functions with only NOR gates

The NOT, AND, OR operations can be realised by using only NOR gates:

NOT

$$\bar{A} = \overline{A + A}$$

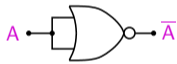


Implementation of functions with only NOR gates

The NOT, AND, OR operations can be realised by using only NOR gates:

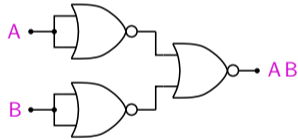
NOT

$$\bar{A} = \overline{A + A}$$



AND

$$A \cdot B = \overline{\overline{A} + \overline{B}}$$

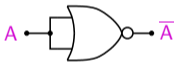


Implementation of functions with only NOR gates

The NOT, AND, OR operations can be realised by using only NOR gates:

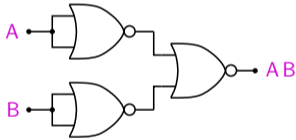
NOT

$$\bar{A} = \overline{A + A}$$



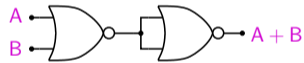
AND

$$A \cdot B = \overline{\overline{A + B}}$$



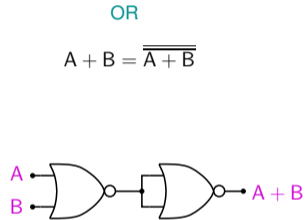
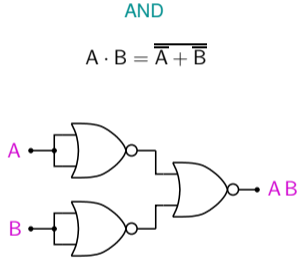
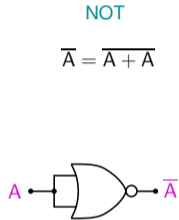
OR

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



Implementation of functions with only NOR gates

The NOT, AND, OR operations can be realised by using only NOR gates:



Implementation of functions with only NOR (or only NAND) gates is more than a theoretical curiosity. There are chips which provide a “sea of gates” (say, NOR gates) which can be configured by the user (through programming) to implement functions.

Implement $Y = AB + BC\bar{D} + \bar{A}D$ using only NOR gates.

Implement $Y = AB + BC\bar{D} + \bar{A}D$ using only NOR gates.

$$\bar{A} = \overline{A + A}$$

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

$$A \cdot B = \overline{\overline{A + B}}$$

Implement $Y = AB + BC\bar{D} + \bar{A}D$ using only NOR gates.

$$Y = \overline{\overline{AB + BC\bar{D} + \bar{A}D}}$$

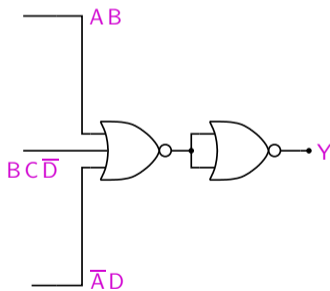
$$\bar{A} = \overline{A + A}$$

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

$$A \cdot B = \overline{\overline{A + B}}$$

Implement $Y = AB + BC\bar{D} + \bar{A}D$ using only NOR gates.

$$Y = \overline{\overline{AB + BC\bar{D} + \bar{A}D}}$$



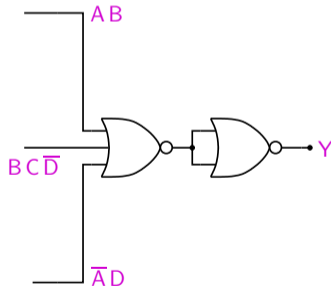
$$\bar{A} = \overline{A + A}$$

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

$$A \cdot B = \overline{\overline{A \cdot B}}$$

Implement $Y = AB + BC\bar{D} + \bar{A}D$ using only NOR gates.

$$Y = \overline{\overline{AB + BC\bar{D} + \bar{A}D}}$$
$$= \overline{(\overline{A + B}) + (\overline{B + C + D}) + (\overline{A + D})}$$



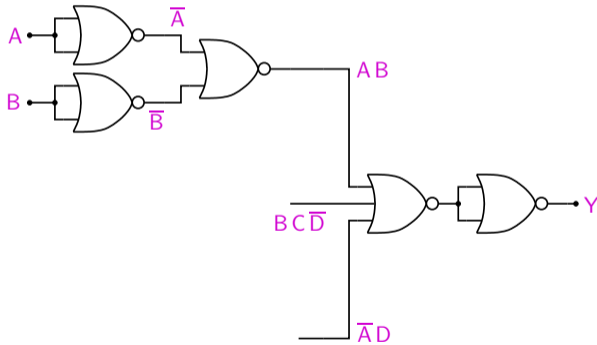
$$\bar{A} = \overline{A + A}$$

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

$$A \cdot B = \overline{\overline{A} + \overline{B}}$$

Implement $Y = AB + BC\bar{D} + \bar{A}D$ using only NOR gates.

$$Y = \overline{\overline{AB + BC\bar{D} + \bar{A}D}}$$
$$= \overline{(\overline{A + B}) + (\overline{B + C + D}) + (\overline{A + D})}$$



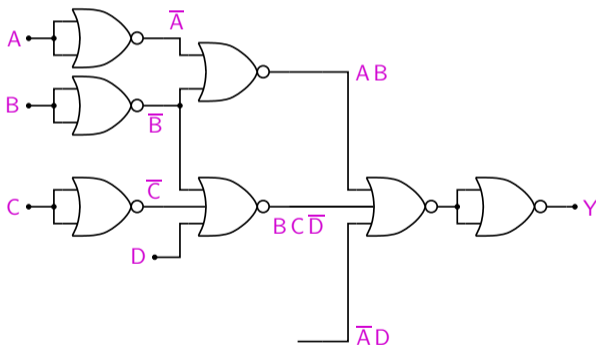
$$\bar{A} = \overline{A + A}$$

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

$$A \cdot B = \overline{\overline{A \cdot B}}$$

Implement $Y = AB + BC\bar{D} + \bar{A}D$ using only NOR gates.

$$Y = \overline{\overline{AB + BC\bar{D} + \bar{A}D}}$$
$$= \overline{\overline{(A + B)} + \overline{(B + C + D)} + \overline{(A + D)}}$$



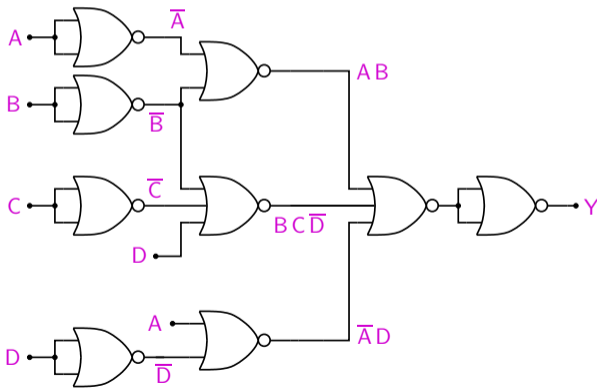
$$\bar{A} = \overline{A + A}$$

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

$$A \cdot B = \overline{\overline{A \cdot B}}$$

Implement $Y = AB + BC\bar{D} + \bar{A}D$ using only NOR gates.

$$Y = \overline{\overline{AB + BC\bar{D} + \bar{A}D}}$$
$$= \overline{\overline{(A + B)} + \overline{(B + C + D)} + \overline{(A + D)}}$$

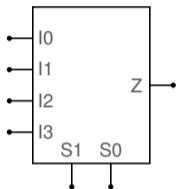


$$\bar{A} = \overline{A + A}$$

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

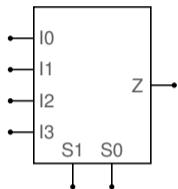
$$A \cdot B = \overline{\overline{A} + \overline{B}}$$

Multiplexers



S1	S0	Z
0	0	I0
0	1	I1
1	0	I2
1	1	I3

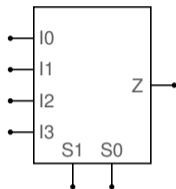
Multiplexers



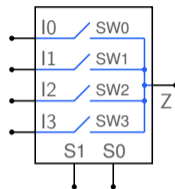
S1	S0	Z
0	0	I0
0	1	I1
1	0	I2
1	1	I3

- * A multiplexer or data selector (MUX in short) has N Select lines, 2^N input lines, and it *routes* one of the input lines to the output.

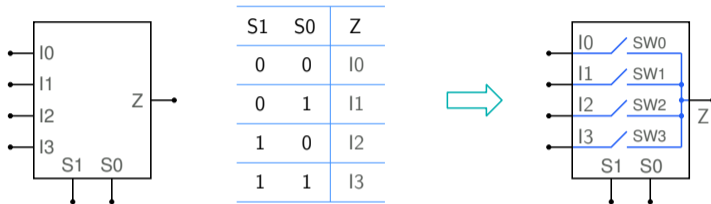
Multiplexers



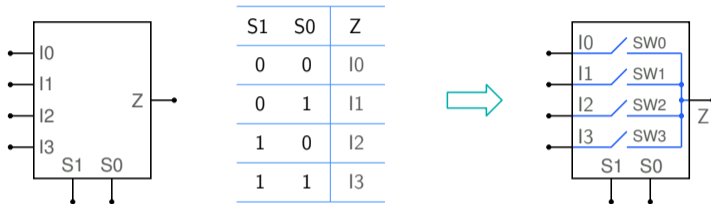
S1	S0	Z
0	0	I0
0	1	I1
1	0	I2
1	1	I3



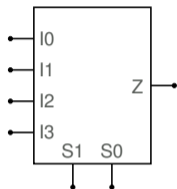
- * A multiplexer or data selector (MUX in short) has N Select lines, 2^N input lines, and it *routes* one of the input lines to the output.



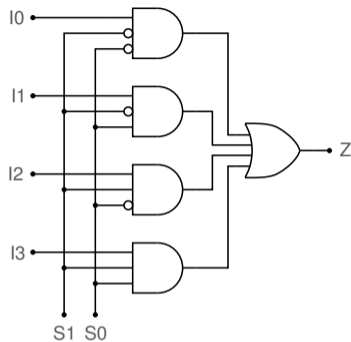
- * A multiplexer or data selector (MUX in short) has N Select lines, 2^N input lines, and it routes one of the input lines to the output.
- * Conceptually, a MUX may be thought of as 2^N switches. For a given combination of the select inputs, only one of the switches closes (makes contact), and the others are open.

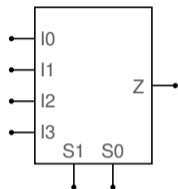


- * A multiplexer or data selector (MUX in short) has N Select lines, 2^N input lines, and it routes one of the input lines to the output.
- * Conceptually, a MUX may be thought of as 2^N switches. For a given combination of the select inputs, only one of the switches closes (makes contact), and the others are open.
- * SEQUEL file: `mux_test_1.sqproj`

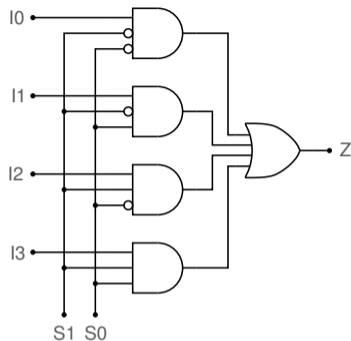


S1	S0	Z
0	0	I0
0	1	I1
1	0	I2
1	1	I3





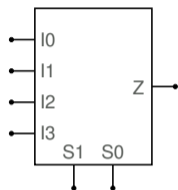
S1	S0	Z
0	0	I0
0	1	I1
1	0	I2
1	1	I3



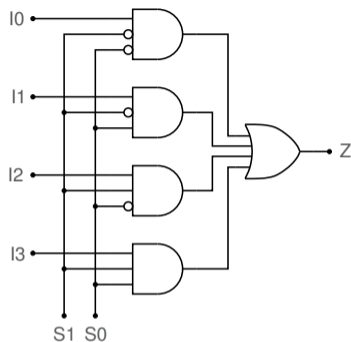
- * A 4-to-1 MUX can be implemented as,

$$Z = I_0 \bar{S}_1 \bar{S}_0 + I_1 \bar{S}_1 S_0 + I_2 S_1 \bar{S}_0 + I_3 S_1 S_0.$$

For a given combination of S_1 and S_0 , only one of the terms survives (the others being 0). For example, with $S_1 = 0$, $S_0 = 1$, we have $Z = I_1$.



S1	S0	Z
0	0	I0
0	1	I1
1	0	I2
1	1	I3

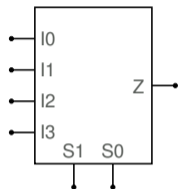


- * A 4-to-1 MUX can be implemented as,

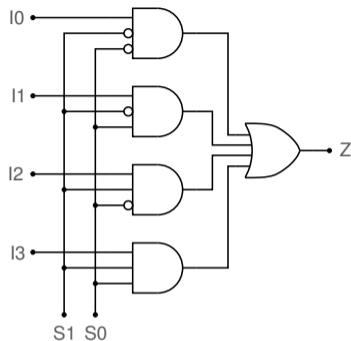
$$Z = I_0 \overline{S_1} \overline{S_0} + I_1 \overline{S_1} S_0 + I_2 S_1 \overline{S_0} + I_3 S_1 S_0.$$

For a given combination of S_1 and S_0 , only one of the terms survives (the others being 0). For example, with $S_1 = 0$, $S_0 = 1$, we have $Z = I_1$.

- * Multiplexers are available as ICs, e.g., 74151 is an 8-to-1 MUX.



S1	S0	Z
0	0	I0
0	1	I1
1	0	I2
1	1	I3



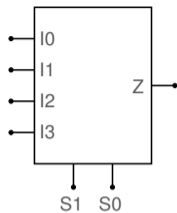
- * A 4-to-1 MUX can be implemented as,

$$Z = I_0 \bar{S}_1 \bar{S}_0 + I_1 \bar{S}_1 S_0 + I_2 S_1 \bar{S}_0 + I_3 S_1 S_0.$$

For a given combination of S_1 and S_0 , only one of the terms survives (the others being 0). For example, with $S_1 = 0$, $S_0 = 1$, we have $Z = I_1$.

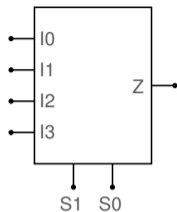
- * Multiplexers are available as ICs, e.g., 74151 is an 8-to-1 MUX.
- * ICs with *arrays* of multiplexers (and other digital blocks) are also available. These blocks can be configured (“wired”) by the user in a programmable manner to realise the functionality of interest.

Active high and active low inputs/outputs



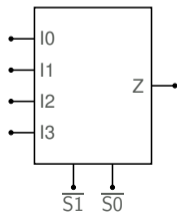
S1	S0	Z
0	0	I0
0	1	I1
1	0	I2
1	1	I3

Select inputs are active high.



S1	S0	Z
0	0	I0
0	1	I1
1	0	I2
1	1	I3

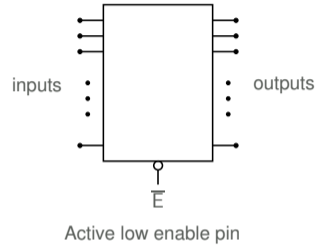
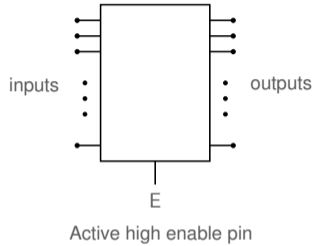
Select inputs are active high.



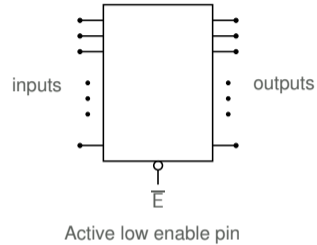
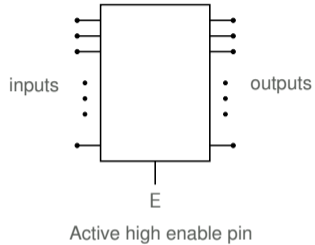
$\overline{S1}$	$\overline{S0}$	Z
1	1	I0
1	0	I1
0	1	I2
0	0	I3

Select inputs are active low.

Enable (E) pin

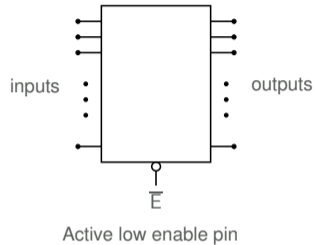
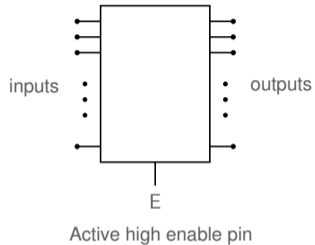


Enable (E) pin



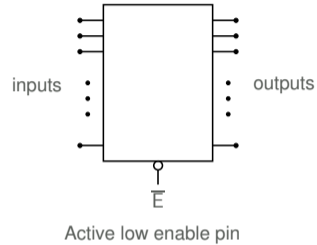
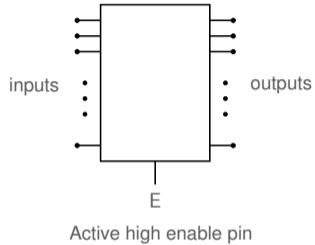
- * Many digital ICs have an “Enable” (E) pin. If the Enable pin is active, the IC functions as desired; else, it is “disabled,” i.e., the outputs are set to some default values.

Enable (E) pin



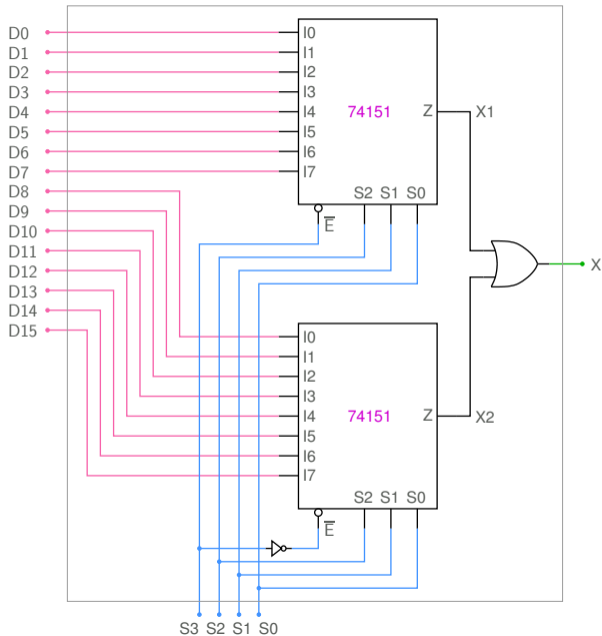
- * Many digital ICs have an “Enable” (E) pin. If the Enable pin is active, the IC functions as desired; else, it is “disabled,” i.e., the outputs are set to some default values.
- * The Enable pin can be active high or active low.

Enable (E) pin



- * Many digital ICs have an “Enable” (E) pin. If the Enable pin is active, the IC functions as desired; else, it is “disabled,” i.e., the outputs are set to some default values.
- * The Enable pin can be active high or active low.
- * If the Enable pin is active low, it is denoted by $\overline{\text{Enable}}$ or \bar{E} . When $\bar{E} = 0$, the IC functions normally; else, it is disabled.

Using two 8-to-1 MUXs to make a 16-to-1 MUX

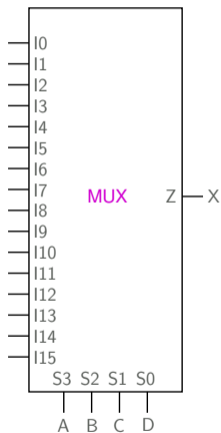


S3	S2	S1	S0	X
0	0	0	0	D0
0	0	0	1	D1
0	0	1	0	D2
0	0	1	1	D3
0	1	0	0	D4
0	1	0	1	D5
0	1	1	0	D6
0	1	1	1	D7
1	0	0	0	D8
1	0	0	1	D9
1	0	1	0	D10
1	0	1	1	D11
1	1	0	0	D12
1	1	0	1	D13
1	1	1	0	D14
1	1	1	1	D15

Implement $X = A\bar{B}\bar{C}D + \bar{A}B\bar{C}\bar{D}$ using a 16-to-1 MUX.

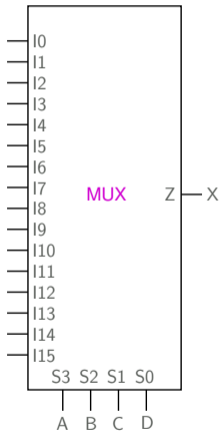
Implement $X = A\bar{B}\bar{C}D + \bar{A}B\bar{C}\bar{D}$ using a 16-to-1 MUX.

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



Implement $X = A\bar{B}\bar{C}D + \bar{A}B\bar{C}\bar{D}$ using a 16-to-1 MUX.

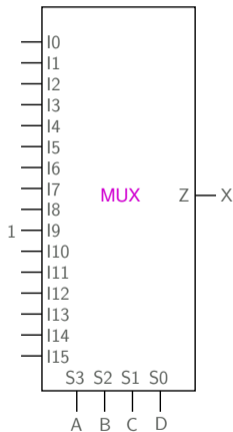
A	B	C	D	X
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0



- * When $A\bar{B}\bar{C}D = 1$, we want $X = 1$.
 $A\bar{B}\bar{C}D = 1 \rightarrow A = 1, B = 0, C = 0,$
 $D = 1$, i.e., the input line corresponding to
 1001 (I9) gets selected.
 \rightarrow Make $I9 = 1$.

Implement $X = A\bar{B}\bar{C}D + \bar{A}B\bar{C}\bar{D}$ using a 16-to-1 MUX.

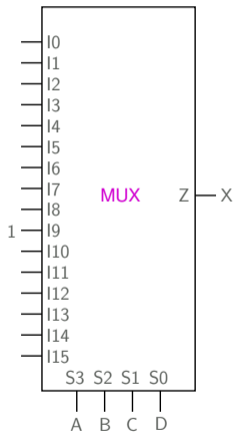
A	B	C	D	X
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0



- * When $A\bar{B}\bar{C}D = 1$, we want $X = 1$.
 $A\bar{B}\bar{C}D = 1 \rightarrow A = 1, B = 0, C = 0, D = 1$, i.e., the input line corresponding to 1001 (I9) gets selected.
 \rightarrow Make $I9 = 1$.

Implement $X = A\bar{B}\bar{C}D + \bar{A}B\bar{C}\bar{D}$ using a 16-to-1 MUX.

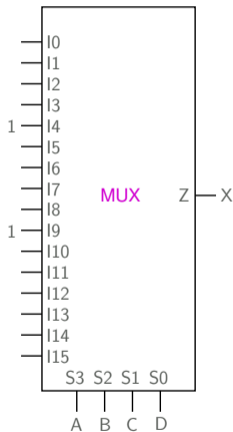
A	B	C	D	X
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0



- * When $A\bar{B}\bar{C}D = 1$, we want $X = 1$.
 $A\bar{B}\bar{C}D = 1 \rightarrow A = 1, B = 0, C = 0, D = 1$, i.e., the input line corresponding to 1001 (I9) gets selected.
 \rightarrow Make I9 = 1.
- * Similarly, when $\bar{A}B\bar{C}\bar{D} = 1$, we want $X = 1$.
 \rightarrow Make I4 = 1.

Implement $X = A\bar{B}\bar{C}D + \bar{A}B\bar{C}\bar{D}$ using a 16-to-1 MUX.

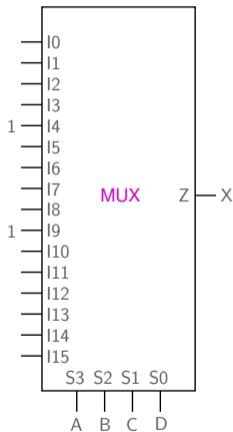
A	B	C	D	X
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0



- * When $A\bar{B}\bar{C}D = 1$, we want $X = 1$.
 $A\bar{B}\bar{C}D = 1 \rightarrow A = 1, B = 0, C = 0, D = 1$, i.e., the input line corresponding to 1001 (I9) gets selected.
 \rightarrow Make I9 = 1.
- * Similarly, when $\bar{A}B\bar{C}\bar{D} = 1$, we want $X = 1$.
 \rightarrow Make I4 = 1.

Implement $X = A\bar{B}\bar{C}D + \bar{A}B\bar{C}\bar{D}$ using a 16-to-1 MUX.

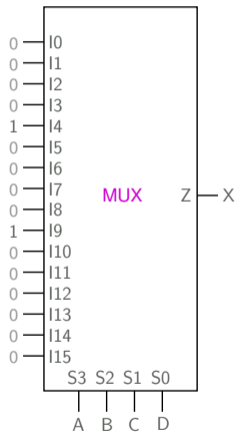
A	B	C	D	X
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0



- * When $A\bar{B}\bar{C}D = 1$, we want $X = 1$.
 $A\bar{B}\bar{C}D = 1 \rightarrow A = 1, B = 0, C = 0, D = 1$, i.e., the input line corresponding to 1001 (I9) gets selected.
 \rightarrow Make I9 = 1.
- * Similarly, when $\bar{A}B\bar{C}\bar{D} = 1$, we want $X = 1$.
 \rightarrow Make I4 = 1.
- * In all other cases, X should be 0.
 \rightarrow connect all other pins to 0.

Implement $X = A\bar{B}\bar{C}D + \bar{A}B\bar{C}\bar{D}$ using a 16-to-1 MUX.

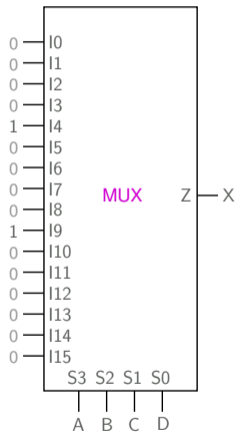
A	B	C	D	X
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0



- * When $A\bar{B}\bar{C}D = 1$, we want $X = 1$.
 $A\bar{B}\bar{C}D = 1 \rightarrow A = 1, B = 0, C = 0, D = 1$, i.e., the input line corresponding to 1001 (I9) gets selected.
 \rightarrow Make I9 = 1.
- * Similarly, when $\bar{A}B\bar{C}\bar{D} = 1$, we want $X = 1$.
 \rightarrow Make I4 = 1.
- * In all other cases, X should be 0.
 \rightarrow connect all other pins to 0.

Implement $X = A\bar{B}\bar{C}D + \bar{A}B\bar{C}\bar{D}$ using a 16-to-1 MUX.

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

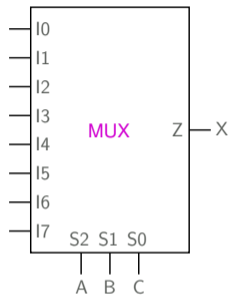


- * When $A\bar{B}\bar{C}D = 1$, we want $X = 1$.
 $A\bar{B}\bar{C}D = 1 \rightarrow A = 1, B = 0, C = 0, D = 1$, i.e., the input line corresponding to 1001 (I9) gets selected.
 \rightarrow Make I9 = 1.
- * Similarly, when $\bar{A}B\bar{C}\bar{D} = 1$, we want $X = 1$.
 \rightarrow Make I4 = 1.
- * In all other cases, X should be 0.
 \rightarrow connect all other pins to 0.
- * In this example, since the truth table is organized in terms of $ABCD$, with A as the MSB and D as the LSB (the same order in which A, B, C, D are connected to the select pins), the design is simple: connect
 I0 to X(0000),
 I1 to X(0001),
 I2 to X(0010), etc.

Implement $X = A\bar{B}\bar{C}D + \bar{A}B\bar{C}\bar{D}$ using an 8-to-1 MUX.

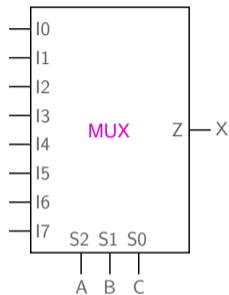
Implement $X = A\bar{B}\bar{C}D + \bar{A}B\bar{C}\bar{D}$ using an 8-to-1 MUX.

A	B	C	X
0	0	0	0
0	0	1	0
0	1	0	\bar{D}
0	1	1	0
1	0	0	D
1	0	1	0
1	1	0	0
1	1	1	0



Implement $X = A\bar{B}\bar{C}D + \bar{A}B\bar{C}\bar{D}$ using an 8-to-1 MUX.

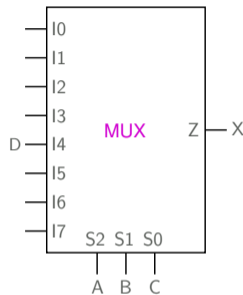
A	B	C	X
0	0	0	0
0	0	1	0
0	1	0	\bar{D}
0	1	1	0
1	0	0	D
1	0	1	0
1	1	0	0
1	1	1	0



- * When $A\bar{B}\bar{C}=1$, i.e., $A=1, B=0, C=0$, we have $X=D$.
→ connect the input line corresponding to 100 (I4) to D.

Implement $X = A\bar{B}\bar{C}D + \bar{A}B\bar{C}\bar{D}$ using an 8-to-1 MUX.

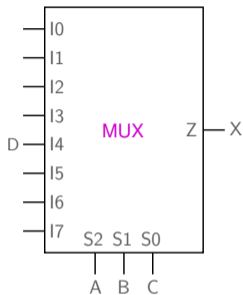
A	B	C	X
0	0	0	0
0	0	1	0
0	1	0	\bar{D}
0	1	1	0
1	0	0	D
1	0	1	0
1	1	0	0
1	1	1	0



- * When $A\bar{B}\bar{C}=1$, i.e., $A=1, B=0, C=0$, we have $X=D$.
→ connect the input line corresponding to 100 (I4) to D.

Implement $X = A\bar{B}\bar{C}D + \bar{A}B\bar{C}\bar{D}$ using an 8-to-1 MUX.

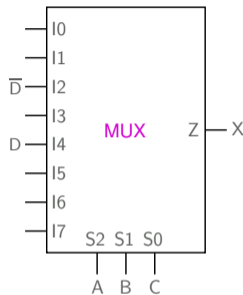
A	B	C	X
0	0	0	0
0	0	1	0
0	1	0	\bar{D}
0	1	1	0
1	0	0	D
1	0	1	0
1	1	0	0
1	1	1	0



- * When $A\bar{B}\bar{C}=1$, i.e., $A=1, B=0, C=0$, we have $X=D$.
→ connect the input line corresponding to 100 (I4) to D.
- * When $\bar{A}B\bar{C}=1$, i.e., $A=0, B=1, C=0$, we have $X=\bar{D}$.
→ connect the input line corresponding to 010 (I2) to \bar{D} .

Implement $X = A\bar{B}\bar{C}D + \bar{A}B\bar{C}\bar{D}$ using an 8-to-1 MUX.

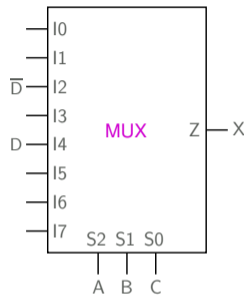
A	B	C	X
0	0	0	0
0	0	1	0
0	1	0	\bar{D}
0	1	1	0
1	0	0	D
1	0	1	0
1	1	0	0
1	1	1	0



- * When $A\bar{B}\bar{C}=1$, i.e., $A=1, B=0, C=0$, we have $X=D$.
→ connect the input line corresponding to 100 (I4) to D.
- * When $\bar{A}B\bar{C}=1$, i.e., $A=0, B=1, C=0$, we have $X=\bar{D}$.
→ connect the input line corresponding to 010 (I2) to \bar{D} .

Implement $X = A\bar{B}\bar{C}D + \bar{A}B\bar{C}\bar{D}$ using an 8-to-1 MUX.

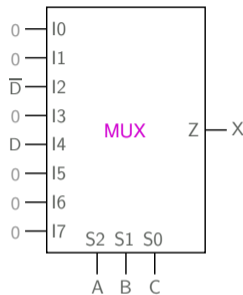
A	B	C	X
0	0	0	0
0	0	1	0
0	1	0	\bar{D}
0	1	1	0
1	0	0	D
1	0	1	0
1	1	0	0
1	1	1	0



- * When $A\bar{B}\bar{C}=1$, i.e., $A=1, B=0, C=0$, we have $X=D$.
→ connect the input line corresponding to 100 (I4) to D.
- * When $\bar{A}B\bar{C}=1$, i.e., $A=0, B=1, C=0$, we have $X=\bar{D}$.
→ connect the input line corresponding to 010 (I2) to \bar{D} .
- * In all other cases, X should be 0.
→ connect all other pins to 0.

Implement $X = A\bar{B}\bar{C}D + \bar{A}B\bar{C}\bar{D}$ using an 8-to-1 MUX.

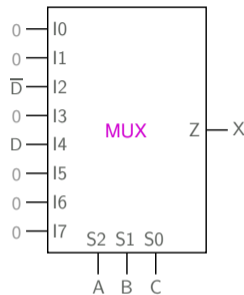
A	B	C	X
0	0	0	0
0	0	1	0
0	1	0	\bar{D}
0	1	1	0
1	0	0	D
1	0	1	0
1	1	0	0
1	1	1	0



- * When $A\bar{B}\bar{C}=1$, i.e., $A=1, B=0, C=0$, we have $X=D$.
→ connect the input line corresponding to 100 (I4) to D.
- * When $\bar{A}B\bar{C}=1$, i.e., $A=0, B=1, C=0$, we have $X=\bar{D}$.
→ connect the input line corresponding to 010 (I2) to \bar{D} .
- * In all other cases, X should be 0.
→ connect all other pins to 0.

Implement $X = A\bar{B}\bar{C}D + \bar{A}B\bar{C}\bar{D}$ using an 8-to-1 MUX.

A	B	C	X
0	0	0	0
0	0	1	0
0	1	0	\bar{D}
0	1	1	0
1	0	0	D
1	0	1	0
1	1	0	0
1	1	1	0

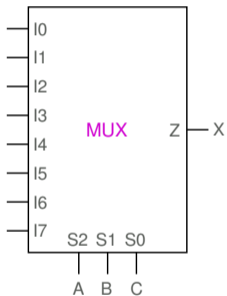


- * When $A\bar{B}\bar{C}=1$, i.e., $A=1, B=0, C=0$, we have $X=D$.
→ connect the input line corresponding to 100 (I4) to D.
- * When $\bar{A}B\bar{C}=1$, i.e., $A=0, B=1, C=0$, we have $X=\bar{D}$.
→ connect the input line corresponding to 010 (I2) to \bar{D} .
- * In all other cases, X should be 0.
→ connect all other pins to 0.
- * Home work: Implement the same function (X) with $S2=B, S1=C, S0=D$.

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

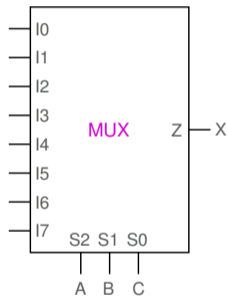
Implement the function X using an 8-to-1 MUX.

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



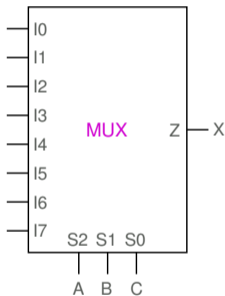
Implement the function X using an 8-to-1 MUX.

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



Implement the function X using an 8-to-1 MUX.

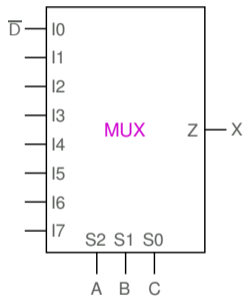
A	B	C	D	X
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0



Implement the function X using an 8-to-1 MUX.

* When $ABC = 000$, $X = \bar{D} \rightarrow I_0 = \bar{D}$.

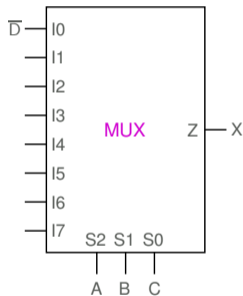
A	B	C	D	X
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0



Implement the function X using an 8-to-1 MUX.

* When $ABC = 000$, $X = \bar{D} \rightarrow I_0 = \bar{D}$.

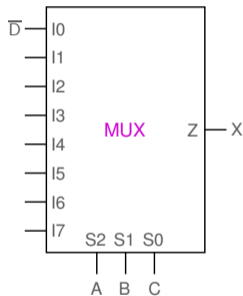
A	B	C	D	X
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0



Implement the function X using an 8-to-1 MUX.

* When $ABC = 000$, $X = \bar{D} \rightarrow I_0 = \bar{D}$.

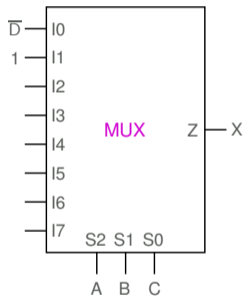
A	B	C	D	X
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0



Implement the function X using an 8-to-1 MUX.

- * When $ABC = 000$, $X = \bar{D} \rightarrow I0 = \bar{D}$.
- * When $ABC = 001$, $X = 1 \rightarrow I1 = 1$, and so on.

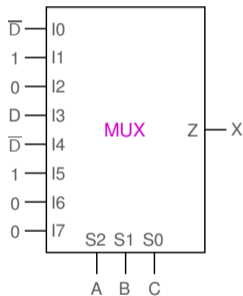
A	B	C	D	X
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0



Implement the function X using an 8-to-1 MUX.

- * When $ABC = 000$, $X = \bar{D} \rightarrow I0 = \bar{D}$.
- * When $ABC = 001$, $X = 1 \rightarrow I1 = 1$, and so on.

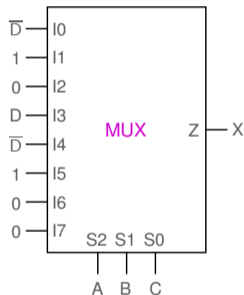
A	B	C	D	X
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0



Implement the function X using an 8-to-1 MUX.

- * When $ABC = 000$, $X = \bar{D} \rightarrow I0 = \bar{D}$.
- * When $ABC = 001$, $X = 1 \rightarrow I1 = 1$, and so on.

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

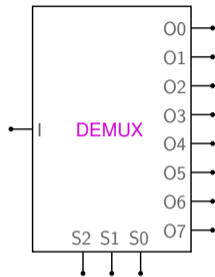


Implement the function X using an 8-to-1 MUX.

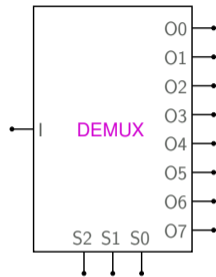
- * When $ABC = 000$, $X = \bar{D} \rightarrow I0 = \bar{D}$.
- * When $ABC = 001$, $X = 1 \rightarrow I1 = 1$, and so on.
- * Home work: repeat with $S2 = B$, $S1 = C$, $S0 = D$.

Demultiplexers

S2	S1	S0	O0	O1	O2	O3	O4	O5	O6	O7
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

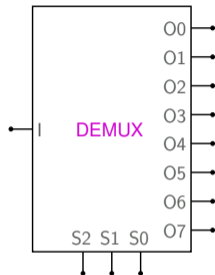


S2	S1	S0	O0	O1	O2	O3	O4	O5	O6	O7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1



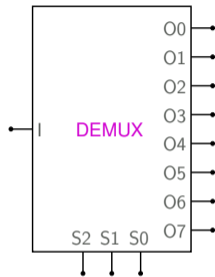
* A demultiplexer takes a *single* input (I) and *routes* it to one of the output lines ($O0, O1, \dots$).

S2	S1	S0	O0	O1	O2	O3	O4	O5	O6	O7
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



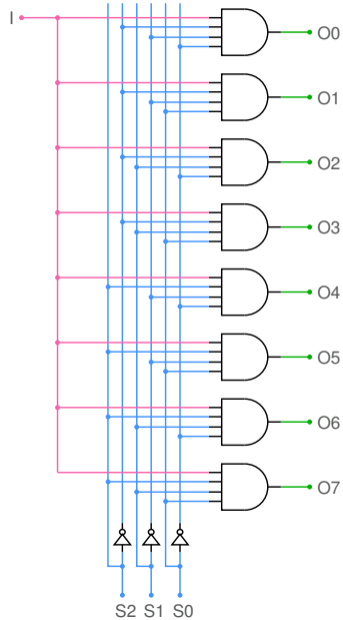
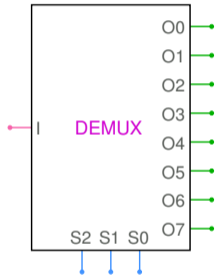
- * A demultiplexer takes a *single* input (I) and *routes* it to one of the output lines ($O0, O1, \dots$).
- * For N Select inputs ($S0, S1, \dots$), the number of output lines is 2^N .

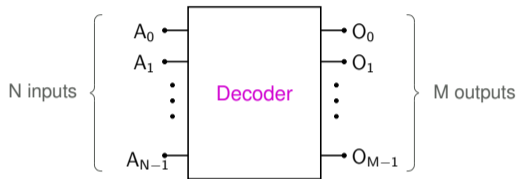
S2	S1	S0	O0	O1	O2	O3	O4	O5	O6	O7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

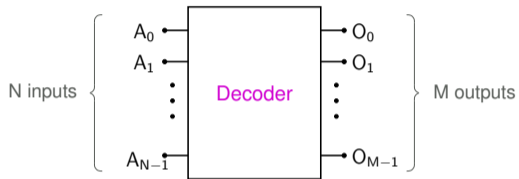


- * A demultiplexer takes a *single* input (I) and *routes* it to one of the output lines ($O0, O1, \dots$).
- * For N Select inputs ($S0, S1, \dots$), the number of output lines is 2^N .
- * SEQUEL file: `demux_test_1.sqproj`

Demultiplexer: gate-level diagram

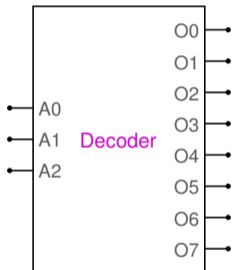






- * For each input combination, an associated bit pattern appears at the output.

3-to-8 decoder (1-of-8 decoder)



A2	A1	A0	O0	O1	O2	O3	O4	O5	O6	O7
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

* Example:

Decimal 75

* Example:

Decimal 75

Binary 1001011

* Example:

Decimal 75

Binary 1001011

BCD 0111 0101

- * Example:

Decimal 75

Binary 1001011

BCD 0111 0101

- * BCD coding is commonly used to display numbers in electronic systems.

Binary-Coded-Decimal (BCD) encoding

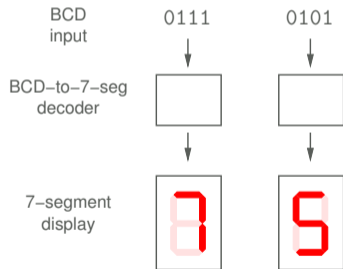
- * Example:

Decimal 75

Binary 1001011

BCD 0111 0101

- * BCD coding is commonly used to display numbers in electronic systems.



Binary-Coded-Decimal (BCD) encoding

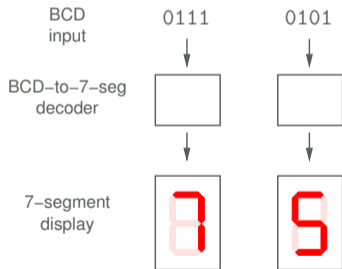
- * Example:

Decimal 75

Binary 1001011

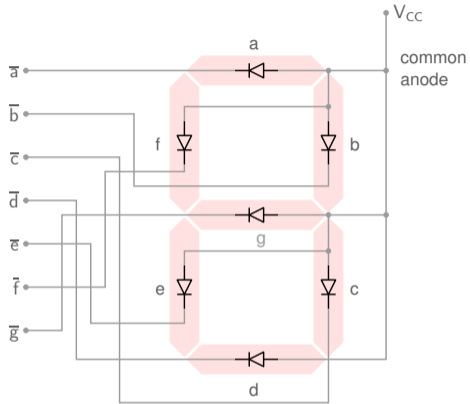
BCD 0111 0101

- * BCD coding is commonly used to display numbers in electronic systems.

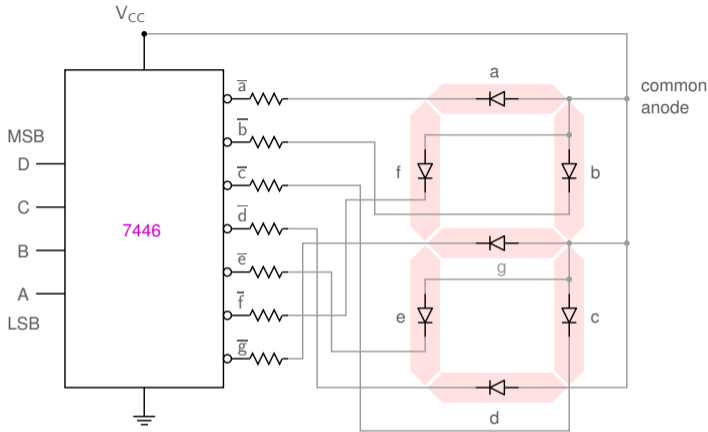


- * In some electronic systems (e.g., calculators), all computations are performed in BCD.

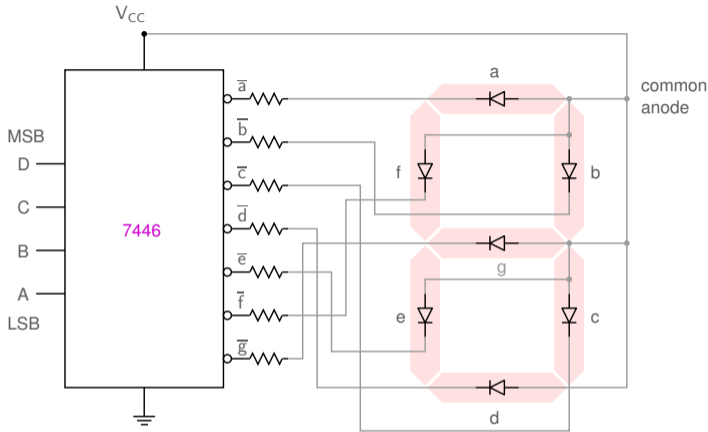
7-segment display



BCD-to-7 segment decoder



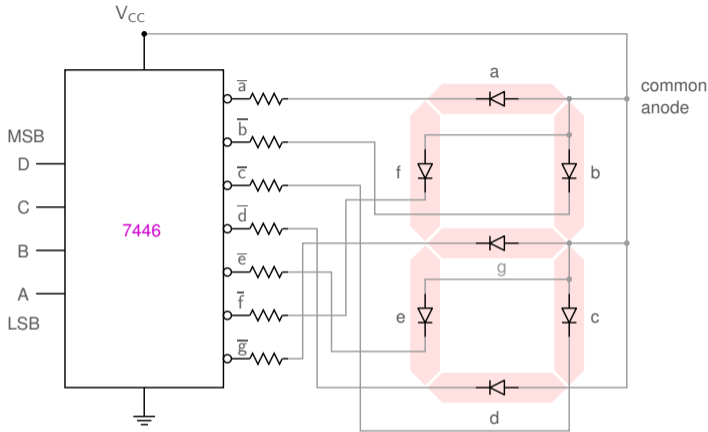
BCD-to-7 segment decoder



* The resistors serve to limit the diode current. For $V_{CC} = 5\text{ V}$, $V_D = 2\text{ V}$, and $I_D = 10\text{ mA}$, $R = 300\ \Omega$.

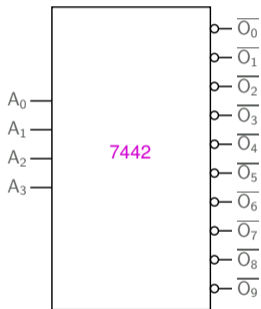


BCD-to-7 segment decoder

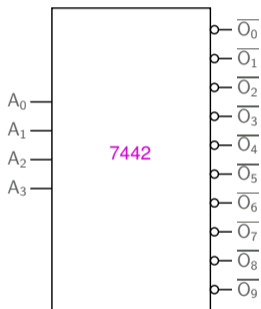


- * The resistors serve to limit the diode current. For $V_{CC} = 5\text{ V}$, $V_D = 2\text{ V}$, and $I_D = 10\text{ mA}$, $R = 300\ \Omega$.
- * Home work: Write the truth table for \bar{c} (in terms of D, C, B, A). Obtain a minimized expression for \bar{c} using a K map.

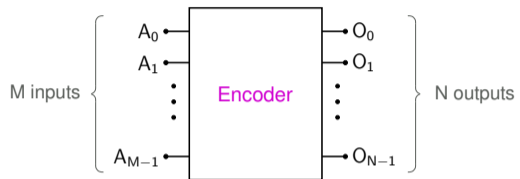


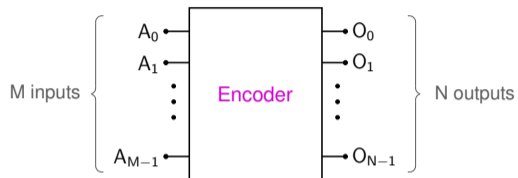


A_3	A_2	A_1	A_0	Active output
0	0	0	0	$\overline{O_0}$
0	0	0	1	$\overline{O_1}$
0	0	1	0	$\overline{O_2}$
0	0	1	1	$\overline{O_3}$
0	1	0	0	$\overline{O_4}$
0	1	0	1	$\overline{O_5}$
0	1	1	0	$\overline{O_6}$
0	1	1	1	$\overline{O_7}$
1	0	0	0	$\overline{O_8}$
1	0	0	1	$\overline{O_9}$
1	0	1	0	none
1	0	1	1	none
1	1	0	0	none
1	1	0	1	none
1	1	1	0	none
1	1	1	1	none

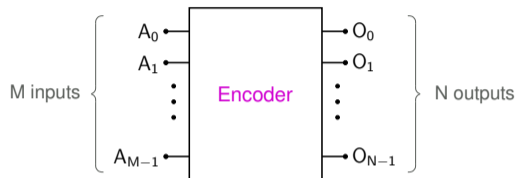


A_3	A_2	A_1	A_0	Active output
0	0	0	0	$\overline{O_0}$
0	0	0	1	$\overline{O_1}$
0	0	1	0	$\overline{O_2}$
0	0	1	1	$\overline{O_3}$
0	1	0	0	$\overline{O_4}$
0	1	0	1	$\overline{O_5}$
0	1	1	0	$\overline{O_6}$
0	1	1	1	$\overline{O_7}$
1	0	0	0	$\overline{O_8}$
1	0	0	1	$\overline{O_9}$
1	0	1	0	none
1	0	1	1	none
1	1	0	0	none
1	1	0	1	none
1	1	1	0	none
1	1	1	1	none

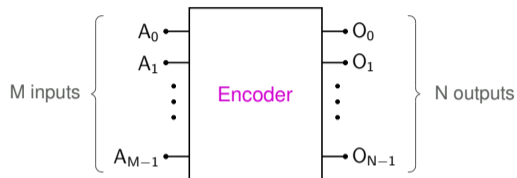




- * Only one input line is assumed to be active. The binary number corresponding to the active input line appears at the output pins.

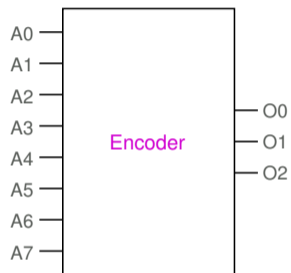


- * Only one input line is assumed to be active. The binary number corresponding to the active input line appears at the output pins.
- * The N output lines can represent 2^N binary numbers, each corresponding to one of the M input lines, i.e., we can have $M = 2^N$. Some encoders have $M < 2^N$.



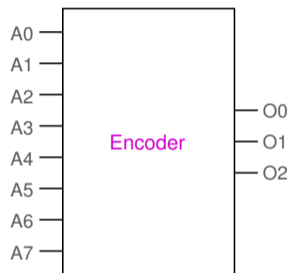
- * Only one input line is assumed to be active. The binary number corresponding to the active input line appears at the output pins.
- * The N output lines can represent 2^N binary numbers, each corresponding to one of the M input lines, i.e., we can have $M = 2^N$. Some encoders have $M < 2^N$.
- * As an example, for $N = 3$, we can have a maximum of $2^3 = 8$ input lines.

8-to-3 encoder example



A0	A1	A2	A3	A4	A5	A6	A7	O2	O1	O0
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

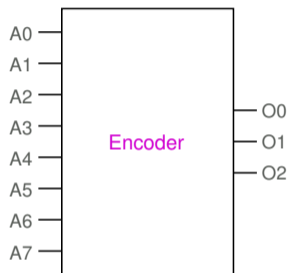
8-to-3 encoder example



A0	A1	A2	A3	A4	A5	A6	A7	O2	O1	O0
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

* Note that only one of the input lines is assumed to be active.

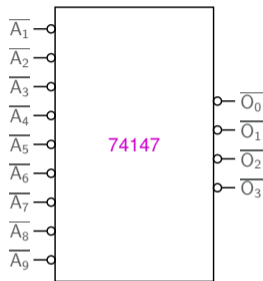
8-to-3 encoder example



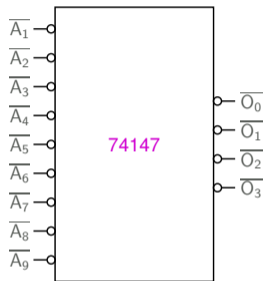
A0	A1	A2	A3	A4	A5	A6	A7	O2	O1	O0
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

- * Note that only one of the input lines is assumed to be active.
- * What if two input lines become simultaneously active?
→ There are “priority encoders” which assign a *priority* to each of the input lines.

74147 decimal-to-BCD priority encoder



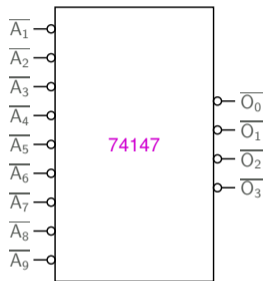
$\overline{A_1}$	$\overline{A_2}$	$\overline{A_3}$	$\overline{A_4}$	$\overline{A_5}$	$\overline{A_6}$	$\overline{A_7}$	$\overline{A_8}$	$\overline{A_9}$	$\overline{O_3}$	$\overline{O_2}$	$\overline{O_1}$	$\overline{O_0}$
1	1	1	1	1	1	1	1	1	1	1	1	1
X	X	X	X	X	X	X	X	0	0	1	1	0
X	X	X	X	X	X	X	0	1	0	1	1	1
X	X	X	X	X	X	0	1	1	1	0	0	0
X	X	X	X	X	0	1	1	1	1	0	0	1
X	X	X	X	0	1	1	1	1	1	0	1	0
X	X	X	0	1	1	1	1	1	1	0	1	1
X	X	0	1	1	1	1	1	1	1	1	0	0
X	0	1	1	1	1	1	1	1	1	1	0	1
0	1	1	1	1	1	1	1	1	1	1	1	0



$\overline{A_1}$	$\overline{A_2}$	$\overline{A_3}$	$\overline{A_4}$	$\overline{A_5}$	$\overline{A_6}$	$\overline{A_7}$	$\overline{A_8}$	$\overline{A_9}$	$\overline{O_3}$	$\overline{O_2}$	$\overline{O_1}$	$\overline{O_0}$
1	1	1	1	1	1	1	1	1	1	1	1	1
X	X	X	X	X	X	X	X	0	0	1	1	0
X	X	X	X	X	X	X	0	1	0	1	1	1
X	X	X	X	X	X	0	1	1	1	0	0	0
X	X	X	X	X	0	1	1	1	1	0	0	1
X	X	X	X	0	1	1	1	1	1	0	1	0
X	X	X	0	1	1	1	1	1	1	0	1	1
X	X	0	1	1	1	1	1	1	1	1	0	0
X	0	1	1	1	1	1	1	1	1	1	0	1
0	1	1	1	1	1	1	1	1	1	1	1	0

* Note that the higher input lines get priority over the lower ones.

For example, $\overline{A_7}$ gets priority over $\overline{A_1}$, $\overline{A_2}$, $\overline{A_3}$, $\overline{A_4}$, $\overline{A_5}$, $\overline{A_6}$. If $\overline{A_7}$ is active (low), the binary output is 1000 (i.e., 0111 inverted bit-by-bit) which corresponds to decimal 7, *irrespective of* $\overline{A_1}$, $\overline{A_2}$, $\overline{A_3}$, $\overline{A_4}$, $\overline{A_5}$, $\overline{A_6}$.



$\overline{A_1}$	$\overline{A_2}$	$\overline{A_3}$	$\overline{A_4}$	$\overline{A_5}$	$\overline{A_6}$	$\overline{A_7}$	$\overline{A_8}$	$\overline{A_9}$	$\overline{O_3}$	$\overline{O_2}$	$\overline{O_1}$	$\overline{O_0}$
1	1	1	1	1	1	1	1	1	1	1	1	1
X	X	X	X	X	X	X	X	0	0	1	1	0
X	X	X	X	X	X	X	0	1	0	1	1	1
X	X	X	X	X	X	0	1	1	1	0	0	0
X	X	X	X	X	0	1	1	1	1	0	0	1
X	X	X	X	0	1	1	1	1	1	0	1	0
X	X	X	0	1	1	1	1	1	1	0	1	1
X	X	0	1	1	1	1	1	1	1	1	0	0
X	0	1	1	1	1	1	1	1	1	1	0	1
0	1	1	1	1	1	1	1	1	1	1	1	0

* Note that the higher input lines get priority over the lower ones.

For example, $\overline{A_7}$ gets priority over $\overline{A_1}$, $\overline{A_2}$, $\overline{A_3}$, $\overline{A_4}$, $\overline{A_5}$, $\overline{A_6}$. If $\overline{A_7}$ is active (low), the binary output is 1000 (i.e., 0111 inverted bit-by-bit) which corresponds to decimal 7, *irrespective of* $\overline{A_1}$, $\overline{A_2}$, $\overline{A_3}$, $\overline{A_4}$, $\overline{A_5}$, $\overline{A_6}$.

* The lower input lines are therefore shown as “don't care” (X) conditions.