UNIT V

Digital Electronics

Complements

Decimal number complements

9's complement of the decimal number N = $(10^n - 1) - N$ = n (9's) – N
i.e. {subtract each digit from 9}
Example → 9's complement of 134795 is 865204

Similarly 1's complement of the binary number N = $(2^n - 1) - N$
= n (1's) – N
Example →1's complement of 110100101 is 001011010
which can be obtained by replacing each one by a zero and each zero by one.

r's complement

10's complement of the decimal number N = 10^n – N = (r-1)'s complement + 1
Example→ 10's complement of 134795 is 865205
Example → find the 9's and 10's complements of 314700.
Answer → 9's complement = 685299 10's complement=685300

Rule: To find the 10's complement of a decimal number leave all leading zeros unchanged. Then subtract the first non-zero digit from 10 and all the remaining digits from 9's.

The 2's complement of a binary number is defined in a similar way.
Example: Find the 1's and 2's complements of the binary number 1101001101
Answer → 1's complement is 0010110010 2's complement is 0010110011
Example: Find the 1's and 2's complements of 100010100
Answer→ 1's complement is 011101011 2's complement is 011101100

Addition using 2's complement

When the decimal numbers to be added are expressed in 2's complement form, the addition of these numbers, following the basic laws of binary addition, gives correct results. Final carry obtained, if any, while adding MSBs should be disregarded. To illustrate this, we will consider the following four different cases:

1. Both the numbers are positive.
2. Larger of the two numbers is positive.
3. The larger of the two numbers is negative.
4. Both the numbers are negative.

Case 1

- Consider the decimal numbers +37 and +18.
- The 2's complement of +37 in eight-bit representation = 00100101.
- The 2's complement of +18 in eight-bit representation = 00010010.
- The addition of the two numbers, that is, +37 and +18, is performed as follows

$$\begin{array}{r} 00100101 \\ +\,00010010 \\ \hline 00110111 \end{array}$$

- The decimal equivalent of $(00110111)_2$ is (+55), which is the correct answer.

Case 2

- Consider the two decimal numbers +37 and -18.
- The 2's complement representation of +37 in eight-bit representation = 00100101.
- The 2's complement representation of −18 in eight-bit representation = 11101110.
- The addition of the two numbers, that is, +37 and −18, is performed as follows:

$$\begin{array}{r} 00100101 \\ +\,11101110 \\ \hline 00010011 \end{array}$$

- The final carry has been disregarded.
- The decimal equivalent of $(00010011)_2$ is +19, which is the correct answer.

Case 3

- Consider the two decimal numbers +18 and −37.
- −37 in 2's complement form in eight−bit representation = 11011011.
- +18 in 2's complement form in eight−bit representation = 00010010.
- The addition of the two numbers, that is, −37 and +18, is performed as follows:

$$\begin{array}{r} 11011011 \\ +\,00010010 \\ \hline 11101101 \end{array}$$

- The decimal equivalent of $(11101101)_2$, which is in 2's complement form, is −19, which is the correct answer. 2's complement representation was discussed in detail in Chapter 1 on number systems.

Case 4

- Consider the two decimal numbers −18 and −37.
- −18 in 2's complement form is 11101110.
- −37 in 2's complement form is 11011011.
- The addition of the two numbers, that is, −37 and −18, is performed as follows:

$$\begin{array}{r} 11011011 \\ +\,11101110 \\ \hline 11001001 \end{array}$$

- The final carry in the ninth bit position is disregarded.
- The decimal equivalent of $(11001001)_2$, which is in 2's complement form, is −55, which is the correct answer.

1. Represent the two numbers to be added in 2's complement form.
2. Do the addition using basic rules of binary addition.
3. Disregard the final carry, if any.
4. The result of addition is in 2's complement form.

Subtraction using r's complement

To find M-N in base r, we add M + r's complement of N
Result is M + (r^n – N)

1) If M > N then result is M – N + r^n (r^n is an end carry and can be neglected.
2) If M < N then result is r^n –(N-M) which is the r's complement of the result.

Example: Subtract (76425 – 28321) using 10's complements.
Answer → 10's complement of 28321 is 71679

```
  7 6 4 2 5
+ 7 1 6 7 9
1 4 8 1 0 4        Discard 1
```

Therefore the difference is 48104 after discarding the end carry.

Subtraction using r's complement

Example: subtract (28531 – 345920)
Answer →It is obvious that the difference is negative. We also have to work with the same number of digits, when dealing with complements.
10's complement of 345920 is 654080
Then add→

```
  0 2 8 5 3 1
+ 6 5 4 0 8 0
  6 8 2 6 1 1
```

Therefore the difference is negative and is equal to the 10's complement of the answer.
Difference is→ - 317389

The same rules apply to binary.

Example: subtract (11010011 – 10001100)
Answer→ 2's complement of 10001100 is 01110100
Then add →

```
  1 1 0 1 0 0 1 1
+ 0 1 1 1 0 1 0 0
1 0 1 0 0 0 1 1 1
```

The difference is positive and is equal to 01000111

Subtraction using (r-1)'s complement

The same rules apply to subtraction using the (r-1)'s complements.
The only difference is that when an end carry is generated, it is not discarded but added to the least significant digit of the result.
Also, if no end carry is generated, then the answer is negative and in the (r-1)'s complement form.

Example: Subtract (76425 – 28321) using 9's complements.
Answer→ 9's complement of 28321 is 71678
Then add→

```
  7 6 4 2 5
+ 7 1 6 7 8
1 4 8 1 0 3
        1
  4 8 1 0 4
```

Example: subtract (11010011 – 10001100) using 1's complement.
Answer→ 1's complement of 10001100 is 01110011
Then add →

```
  1 1 0 1 0 0 1 1
+ 0 1 1 1 0 0 1 1
1 0 1 0 0 0 1 1 0
              1
  0 1 0 0 0 1 1 1
```

- In the latter part of the nineteenth century, George Boole incensed philosophers and mathematicians alike when he suggested that logical thought could be represented through mathematical equations.
  - *How dare anyone suggest that human thought could be encapsulated and manipulated like an algebraic formula?*
- Computers, as we know them today, are implementations of Boole's *Laws of Thought*.
  - John Atanasoff and Claude Shannon were among the first to see this connection.

14

- In the middle of the twentieth century, computers were commonly known as “thinking machines” and “electronic brains.”
  - Many people were fearful of them.
- Nowadays, we rarely ponder the relationship between electronic digital computers and human logic. Computers are accepted as part of our lives.
  - Many people, however, are still fearful of them.
- In this chapter, you will learn the simplicity that constitutes the essence of the machine.

15

- Boolean algebra is a mathematical system for the manipulation of variables that can have one of two values.
  - In formal logic, these values are “true” and “false.”
  - In digital systems, these values are “on” and “off,” 1 and 0, or “high” and “low.”
- Boolean expressions are created by performing operations on Boolean variables.
  - Common Boolean operators include AND, OR, and NOT.

16

- A Boolean operator can be completely described using a truth table.
- The truth table for the Boolean operators AND and OR are shown at the right.
- The AND operator is also known as a Boolean product. The OR operator is the Boolean sum.

X AND Y

X	Y	XY
0	0	0
0	1	0
1	0	0
1	1	1

X OR Y

X	Y	X+Y
0	0	0
0	1	1
1	0	1
1	1	1

17

- The truth table for the Boolean NOT operator is shown at the right.
- The NOT operation is most often designated by an overbar. It is sometimes indicated by a prime mark (‘) or an “elbow” (¬).

NOT X

X	$\overline{X}$
0	1
1	0

18

- A Boolean function has:
  - At least one Boolean variable,
  - At least one Boolean operator, and
  - At least one input from the set {0,1}.
- It produces an output that is also a member of the set {0,1}.

Now you know why the binary numbering system is so handy in digital systems.

19

- The truth table for the Boolean function:

  $F(x,y,z) = x\bar{z}+y$

  is shown at the right.
- To make evaluation of the Boolean function easier, the truth table contains extra (shaded) columns to hold evaluations of subparts of the function.

$F(x,y,z) = x\bar{z}+y$

x	y	z	$\bar{z}$	$x\bar{z}$	$x\bar{z}+y$
0	0	0	1	0	0
0	0	1	0	0	0
0	1	0	1	0	1
0	1	1	0	0	1
1	0	0	1	1	1
1	0	1	0	0	0
1	1	0	1	1	1
1	1	1	0	0	1

20

- As with common arithmetic, Boolean operations have rules of precedence.
- The NOT operator has highest priority, followed by AND and then OR.
- This is how we chose the (shaded) function subparts in our table.

$F(x, y, z) = x\bar{z} + y$

x	y	z	$\bar{z}$	$x\bar{z}$	$x\bar{z}+y$
0	0	0	1	0	0
0	0	1	0	0	0
0	1	0	1	0	1
0	1	1	0	0	1
1	0	0	1	1	1
1	0	1	0	0	0
1	1	0	1	1	1
1	1	1	0	0	1

21

- Digital computers contain circuits that implement Boolean functions.
- The simpler that we can make a Boolean function, the smaller the circuit that will result.
  - Simpler circuits are cheaper to build, consume less power, and run faster than complex circuits.
- With this in mind, we always want to reduce our Boolean functions to their simplest form.
- There are a number of Boolean identities that help us to do this.

22

- Most Boolean identities have an AND (product) form as well as an OR (sum) form. We give our identities using both forms. Our first group is rather intuitive:

Identity Name	AND Form	OR Form
Identity Law	$1x = x$	$0 + x = x$
Null Law	$0x = 0$	$1 + x = 1$
Idempotent Law	$xx = x$	$x + x = x$
Inverse Law	$x\bar{x} = 0$	$x + \bar{x} = 1$

23

- Our second group of Boolean identities should be familiar to you from your study of algebra:

Identity Name	AND Form	OR Form
Commutative Law	$xy = yx$	$x+y = y+x$
Associative Law	$(xy)z = x(yz)$	$(x+y)+z = x + (y+z)$
Distributive Law	$x+yz = (x+y)(x+z)$	$x(y+z) = xy+xz$

24

- Our last group of Boolean identities are perhaps the most useful.
- If you have studied set theory or formal logic, these laws are also familiar to you.

Identity Name	AND Form	OR Form
Absorption Law	$x(x+y) = x$	$x + xy = x$
DeMorgan's Law	$\overline{(xy)} = \bar{x} + \bar{y}$	$\overline{(x+y)} = \bar{x}\bar{y}$
Double Complement Law	$\overline{(\bar{x})} = x$	

25

- We can use Boolean identities to simplify the function: $F(X,Y,Z) = (X + Y)(X + \bar{Y})\,\overline{(X\bar{Z})}$ as follows:

$(X + Y)(X + \bar{Y})\,\overline{(X\bar{Z})}$	Idempotent Law (Rewriting)
$(X + Y)(X + \bar{Y})(\bar{X} + Z)$	DeMorgan's Law
$(XX + X\bar{Y} + XY + Y\bar{Y})(\bar{X} + Z)$	Distributive Law
$((X + Y\bar{Y}) + X(Y + \bar{Y}))(\bar{X} + Z)$	Commutative & Distributive Laws
$((X + 0) + X(1))(\bar{X} + Z)$	Inverse Law
$X(\bar{X} + Z)$	Idempotent Law
$X\bar{X} + XZ$	Distributive Law
$0 + XZ$	Inverse Law
XZ	Idempotent Law

26

- Sometimes it is more economical to build a circuit using the complement of a function (and complementing its result) than it is to implement the function directly.
- DeMorgan's law provides an easy way of finding the complement of a Boolean function.
- Recall DeMorgan's law states:

$$\overline{(xy)} = \bar{x} + \bar{y} \quad \text{and} \quad \overline{(x+y)} = \bar{x}\bar{y}$$

27

- DeMorgan's law can be extended to any number of variables.
- Replace each variable by its complement and change all ANDs to ORs and all ORs to ANDs.
- Thus, we find the the complement of:

$$F(X,Y,Z) = (XY) + (\bar{X}Z) + (Y\bar{Z})$$

is:

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

28

- Through our exercises in simplifying Boolean expressions, we see that there are numerous ways of stating the same Boolean expression.
  - These "synonymous" forms are *logically equivalent.*
  - Logically equivalent expressions have identical truth tables.
- In order to eliminate as much confusion as possible, designers express Boolean functions in *standardized* or *canonical* form.

29

- There are two canonical forms for Boolean expressions: sum-of-products and product-of-sums.
  - Recall the Boolean product is the AND operation and the Boolean sum is the OR operation.
- In the sum-of-products form, ANDed variables are ORed together.
  - For example: $F(x,y,z) = xy + xz + yz$
- In the product-of-sums form, ORed variables are ANDed together:
  - For example: $F(x,y,z) = (x+y)(x+z)(y+z)$

30

- It is easy to convert a function to sum-of-products form using its truth table.
- We are interested in the values of the variables that make the function true (=1).
- Using the truth table, we list the values of the variables that result in a true function value.
- Each group of variables is then ORed together.

$F(x,y,z) = x\bar{z}+y$

x	y	z	$x\bar{z}+y$
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

31

- The sum-of-products form for our function is:

$$F(x,y,z) = \bar{x}y\bar{z}+\bar{x}yz+x\bar{y}\bar{z}+xy\bar{z}+xyz$$

We note that this function is not in simplest terms. Our aim is only to rewrite our function in canonical sum-of-products form.

$F(x,y,z) = x\bar{z}+y$

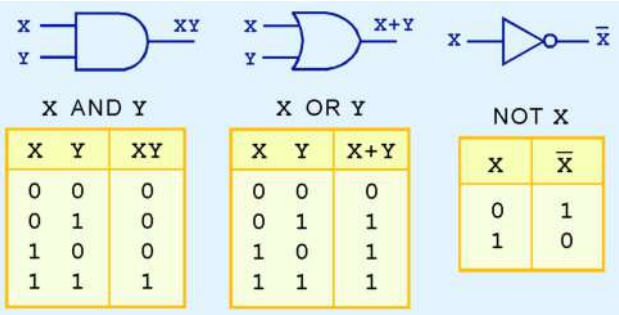
x	y	z	$x\bar{z}+y$
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

32

- We have looked at Boolean functions in abstract terms.
- In this section, we see that Boolean functions are implemented in digital computer circuits called gates.
- A gate is an electronic device that produces a result based on two or more input values.
  - In reality, gates consist of one to six transistors, but digital designers think of them as a single unit.
  - Integrated circuits contain collections of gates suited to a particular purpose.

33

- The three simplest gates are the AND, OR, and NOT gates.



X AND Y

X	Y	XY
0	0	0
0	1	0
1	0	0
1	1	1

X OR Y

X	Y	X+Y
0	0	0
0	1	1
1	0	1
1	1	1

NOT X

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

- They correspond directly to their respective Boolean operations, as you can see by their truth tables.

34

- Another very useful gate is the exclusive OR (XOR) gate.
- The output of the XOR operation is true only when the values of the inputs differ.

X XOR Y

X	Y	X⊕Y
0	0	0
0	1	1
1	0	1
1	1	0

Note the special symbol ⊕ for the XOR operation.

35

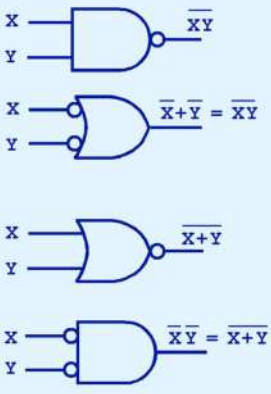
- NAND and NOR are two very important gates. Their symbols and truth tables are shown at the right.

X NAND Y

X	Y	X NAND Y
0	0	1
0	1	1
1	0	1
1	1	0

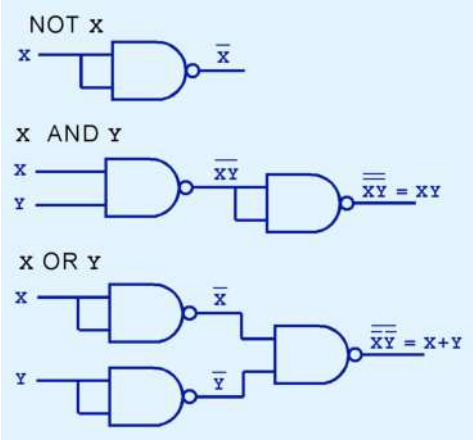
X NOR Y

X	Y	X NOR Y
0	0	1
0	1	0
1	0	0
1	1	0



36

- NAND and NOR are known as *universal gates* because they are inexpensive to manufacture and any Boolean function can be constructed using only NAND or only NOR gates.

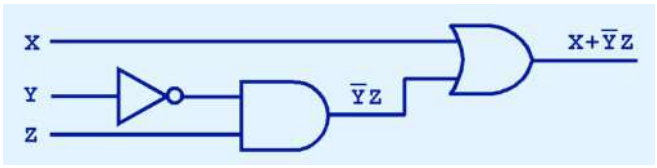


37

- Gates can have multiple inputs and more than one output.
  - A second output can be provided for the complement of the operation.
  - We'll see more of this later.

38

- The main thing to remember is that combinations of gates implement Boolean functions.
- The circuit below implements the Boolean function: $F(X,Y,Z) = X+\overline{Y}Z$



We simplify our Boolean expressions so that we can create simpler circuits.

39

- We have designed a circuit that implements the Boolean function:

$$F(X,Y,Z) = X+\overline{Y}Z$$

- This circuit is an example of a *combinational logic* circuit.
- Combinational logic circuits produce a specified output (almost) at the instant when input values are applied.
  - In a later section, we will explore circuits where this is not the case.

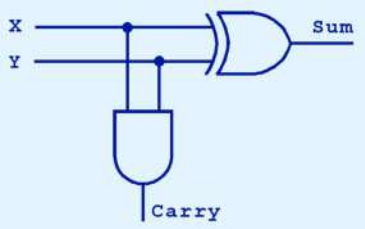
40

- Combinational logic circuits give us many useful devices.
- One of the simplest is the *half adder*, which finds the sum of two bits.
- We can gain some insight as to the construction of a half adder by looking at its truth table, shown at the right.

Inputs		Outputs	
X	Y	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

41

- As we see, the sum can be found using the XOR operation and the carry using the AND operation.



Inputs		Outputs	
X	Y	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

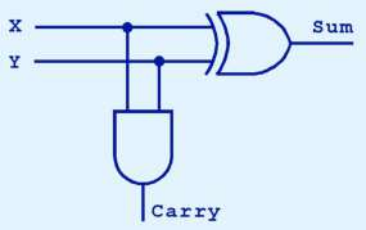
42

- We can change our half adder into to a full adder by including gates for processing the carry bit.
- The truth table for a full adder is shown at the right.

Inputs			Outputs	
X	Y	Carry In	Sum	Carry Out
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

43

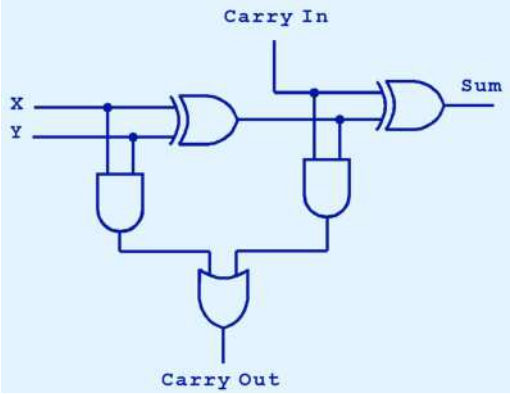
- How can we change the half adder shown below to make it a full adder?



Inputs			Outputs	
X	Y	Carry In	Sum	Carry Out
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

44

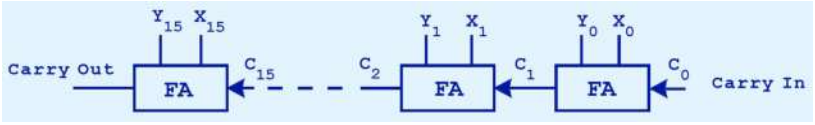
- Here's our completed full adder.



Inputs			Outputs	
X	Y	Carry In	Sum	Carry Out
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

45

- Just as we combined half adders to make a full adder, full adders can connected in series.
- The carry bit "ripples" from one adder to the next; hence, this configuration is called a *ripple-carry adder*.



Today's systems employ more efficient adders.

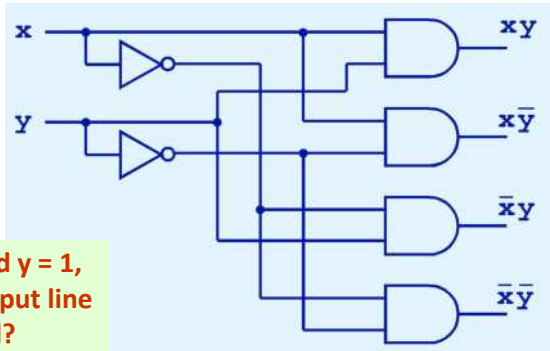
46

- Decoders are another important type of combinational circuit.
- Among other things, they are useful in selecting a memory location according a binary value placed on the address lines of a memory bus.
- Address decoders with *n* inputs can select any of 2^n locations.

This is a block diagram for a decoder.

47

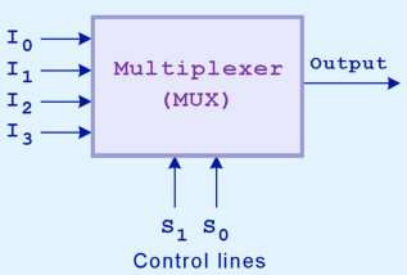
- This is what a 2-to-4 decoder looks like on the inside.



If x = 0 and y = 1, which output line is enabled?

48

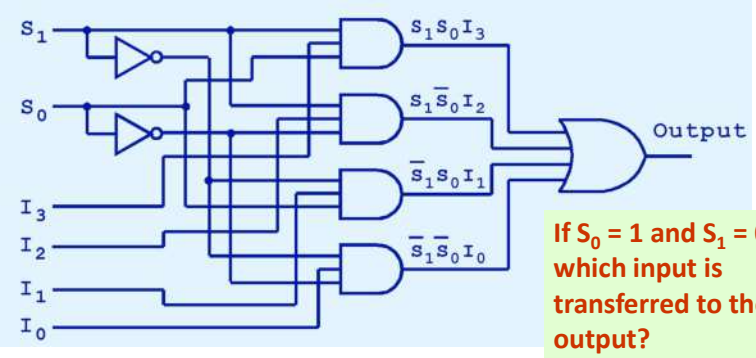
- A multiplexer does just the opposite of a decoder.
- It selects a single output from several inputs.
- The particular input chosen for output is determined by the value of the multiplexer's control lines.
- To be able to select among *n* inputs, $\log_2 n$ control lines are needed.



This is a block diagram for a multiplexer.

49

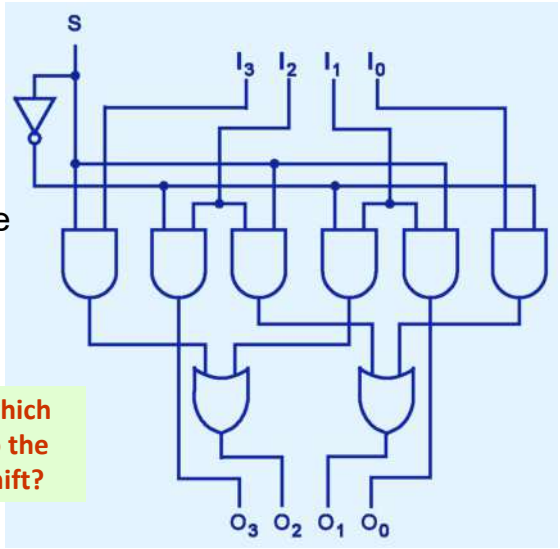
- This is what a 4-to-1 multiplexer looks like on the inside.



If $S_0 = 1$ and $S_1 = 0$, which input is transferred to the output?

50

- This shifter moves the bits of a nibble one position to the left or right.



If S = 0, in which direction do the input bits shift?

51

i) $(X+\overline{Y}+XY)\,(X+\overline{Y})\,\overline{X}Y = 0$

ii) $(AB+C+D)\,(\overline{C}+D)\,(\overline{C}+D+E) = AB\overline{C}+D$

a) $\overline{(A+\overline{B})(\overline{A}+B)}$ b) $\overline{\overline{A}\,B\,\overline{C}\,D}$.

(i) $f(x, y, z) = \sum m(1,3,5)$

(ii) $f(w, x, y, z) = \pi M(0,2,5,6,7,8,9,11,12)$.