UNIT 3
DIGITAL ARITHMETIC AND LOGIC GATES

Majority of arithmetic performed by computers is binary arithmetic, that is, arithmetic on base two numbers. Decimal and floating-point numbers, also used in computer arithmetic, depend on binary representations, and an understanding of binary arithmetic is necessary in order to understand either one.

Computers perform arithmetic on fixed-size numbers. The arithmetic of fixed size numbers is called finite-precision arithmetic. The rules for finite-precision arithmetic are different from the rules of ordinary arithmetic. The sizes of numbers which can be arithmetic operands are determined when the architecture of the computer is designed. Common sizes for integer arithmetic are eight, 16, 32, and recently 64 bits. It is possible for the programmer to perform arithmetic on larger numbers or on sizes which are not directly implemented in the architecture. However, this is usually so painful that the programmer picks the most appropriate size implemented by the architecture. This puts a burden on the computer architect to select appropriate sizes for integers, and on the programmer to be aware of the limitations of the size he has chosen and on finite-precision arithmetic in general.

Addition

Addition of binary numbers can be carried out in a similar way by the column method.

Adding binary numbers is a very simple task, and very similar to the longhand addition of decimal numbers. As with decimal numbers, you start by adding the bits (digits) one column, or place weight, at a time, from right to left.

0 + 0 = 0

1 + 0 = 1

0 + 1 = 1

1 + 1 = 10

1 + 1 + 1 = 11

Example 1: 10001 + 11101

Solution:

1 – carry bit

1 0 0 0 1

(+) 1 1 1 0 1

---

1 0 1 1 1 0

Example 2: 10111 + 110001

Solution:

1 1 1 – carry bit

1 0 1 1 1

(+) 1 1 0 0 0 1

---

1 0 0 1 0 0 0

Subtraction

The subtraction of the binary digit depends on the four basic operations

0 – 0 = 0

1 – 0 = 1

0-1=1 (borrow 1)

1 – 1 = 0

10 – 1 = 1

The above first three operations are easy to understand as they are identical to decimal subtraction. The fourth operation can be understood with the logic two minus one is one.

For a binary number with two or more digits, the subtraction is carried out column by column as in decimal subtraction. Also, sometimes one has to borrow from the next higher column. Consider the following example.

```
        0   10 <---- borrow
    1   1   0   0
(-) 1   0   1   0
-------------------
    0   0   1   0
-------------------
```

Example 1: 0011010 – 001100

Solution:

1 1 Borrow

0 0 1 1 0 1 0

(-) 0 0 1 1 0 0

---

0 0 0 1 1 1 0

Decimal Equivalent :

0 0 1 1 0 1 0 = 26

0 0 1 1 0 0 = 12

Therefore, 26 – 12 = 14

The binary resultant 0 0 0 1 1 1 0 is equivalent to the 14

Example 2: 0100010 – 0001010

Solution:

1 1 Borrow

$0\ 1\ 0\ 0\ 0\ 1\ 0 = 34_{10}$

$(-)\ 0\ 0\ 0\ 1\ 0\ 1\ 0 = 10_{10}$

———————

$0\ 0\ 1\ 1\ 0\ 0\ 0 = 24_{10}$

Multiplication

Binary multiplication is one of the four binary arithmetic. The other three fundamental operations are addition, subtraction and division. In the case of a binary operation, we deal with only two digits, i.e. 0 and 1. The operation performed while finding the binary product is similar to the conventional multiplication method. The four major steps in binary digit multiplication are:

$0 \times 0 = 0$

$0 \times 1 = 0$

$1 \times 0 = 0$

$1 \times 1 = 1$

Note: The binary product of the two binary numbers 1 and 1 is equal to 1 only. And no additional number is borrowed or carried forward in this operation.

Example 1: Multiply 110 by 100

110

X 100

———

000

000

110

———-

11000

———

Example 2: Solve 1010 × 101

Solution:

1010 × 101

1010

(×) 101

_____

1010

0000

1010

_____

110010

Division

The binary division is much easier than the decimal division when you remember the following division rules. The main rules of the binary division include:

- 1÷1 = 1
- 1÷0 = 0
- 0÷1 = not possible
- 0÷0 = not possible

Question: Solve 01111100 ÷ 0010

Solution:

Given

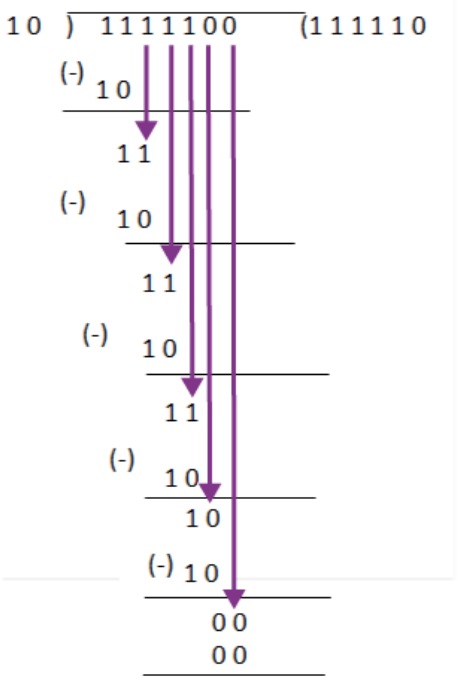
01111100 ÷ 0010

Here the dividend is 01111100, and the divisor is 0010

Remove the zero's in the **Most Significant Bit** in both the dividend and divisor, that doesn't change the value of the number.

So the dividend becomes 1111100, and the divisor becomes 10.

Now, use the long division method.



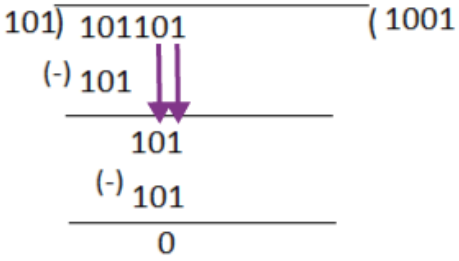
- **Step 1:** First, look at the first two numbers in the dividend and compare with the divisor. Add the number 1 in the quotient place. Then subtract the value, you get 1 as remainder.
- **Step 2:** Then bring down the next number from the dividend portion and do the step 1 process again
- **Step 3:** Repeat the process until the remainder becomes zero by comparing the dividend and the divisor value.
- **Step 4:** Now, in this case, after you get the remainder value as 0, you have zero left in the dividend portion, so bring that zero to the quotient portion.

Therefore, the resultant value is quotient value which is equal to 111110

So, 01111100 ÷ 0010 = 111110

Example 2: Solve using the long division method: 101101 ÷ 101

Solution:



9's and 10's Complement

9's complement and **10's complement** are used to make the arithmetic operations in digital system easier. These are used to make computational operations easier using complement implementation and usually trade hardware usage to the program.

To obtain the 9's complement of any number we have to subtract the number with $(10^n - 1)$ where n = number of digits in the number, or in a simpler manner we have to subtract each digit of the given decimal number from 9.

10's complement, it is relatively easy to find out the 10's complement after finding out the 9's complement of that number. We have to add 1 with the **9's complement** of any number to obtain the desired 10's complement of that number. Or if we want to find out the 10's complement directly, we can do it by following the following formula, $(10^n$ – number), where n = number of digits in the number.

Let us take a decimal number 456, 9's complement of this number will be

```
 999
-456
_____
543
```

10's complement of this no

```
543
(+)1
_____
544
```

Consider some decimal numbers 7, 34, 566, 3456, now let's find 9's complement of each of these.

9-7 = 2, 99-34 = 65, 999-566 = 433, 9999-3456 = 6543

Now, in order to find 10's complement of these numbers let's add 1 to each of these numbers.

2+1 = 3, 65+1 = 66, 433+1 = 434, 6543+1 = 6544

Thus, 3, 66, 434 & 6544 are the 10's complement of the numbers 7, 34, 566, 3456 respectively.

Subtraction of decimal number using 9`s complement

Here are the steps are given below:

1. Make the both numbers having the same number of digits.
2. Determine the 9`s complement of the number from which we subtracted (subtrahend).
3. Add the 9`s complement to the given number from which we subtract (minuend).
4. If there exists` any additional digit (carry) in the result after addition, remove it and add it to the complement of the result and prefix by a negative sign to get the final result.

E.g. Subtract $(123)_{10}$ From $(345)_{10}$

9`s complement of 123= (999 -123) =876

Adding the 9`s complement with 345, i.e 345 + 876 = 1221

In the result, most significant digit 1 is the carryover. So add this carry over to remaining digits 221

i.e, 221 + 1 = 222

Hence, $(222)_{10}$ is the required result after subtracting $(123)_{10}$ from $(345)_{10}$.

Subtraction using 10`s complement:

Here are the steps are given below:

1. Make the both numbers having same numbers of digits.
2. Determine the 10`s complement of the number to be subtracted (subtrahend).
3. Add the 10`s complement to the given number from which we subtract (minuend).
4. If there exists`s any additional digit (carry) in the result after addition, remove it from the result and the remaining digits form the final result.
5. If there exists` no any carry then determine the 10`s complement of the result and prefix by the negative sign to get the final result.

Example: Subtract $(123)_{10}$ from $(345)_{10}$

10`s Complement of 123 = (999 - 123) = 876 + 1 = 877

Adding the 10`s complement with 345, i.e. 345 + 877 = 1222

In this result, most significant digit 1 is the carry over.So remove it to find the result. Therefore, $(222)_{10}$ is the required result.

When smaller number is to be subtracted from larger one

Regular subtraction	Subtraction using 9's complement
678 - 234 444	678 +765 ← (9's complement of 234) 1443 + 1 444

When larger number is to be subtracted from smaller one

Regular subtraction	Subtraction using 9's complement
228 - 485 - 257	228 +514 ← (9's complement of 485) 742 (No carry indicates –ve value) ↓ - 257 (9's complement of result)

When smaller number is to be subtracted from larger one

Regular subtraction	Subtraction using 10's complement
678 - 234 444	678 +766 ← (10's complement of 234) ①444 ← (Ignore the carry) ↓ 444

When larger number is to be subtracted from smaller one

Regular subtraction	Subtraction using 10's complement
228 - 485 - 257	228 +515 ← (10's complement of 485) 743 (No carry indicates –ve value) ↓ - 257 (10's complement of result)

1's and 2's Complement

1's complement

The 1's complement of a binary number is obtained just by changing each 0 to 1 and each 1 to 0.

Binary number	1	0	1	1	1	0	1	0
	↓	↓	↓	↓	↓	↓	↓	↓
1-complement	0	1	0	0	0	1	0	1

Fig. 1's complement

2's complement

The 2's complement of a binary number is obtained adding 1 to the 1's complement of this number:

2's complement = 1's complement+1

Binary number 1 0 1 1 1 0 1 0
1's complement 0 1 0 0 0 1 0 1
\+ **1**
2's complement 0 1 0 0 0 1 1 0

There is a simple method to obtain the 2's complement:

- Beginning with the LSB, just write down bits as they are moving to left till the first 1, including it. Substitute the rest of bits by their 1's complement.

ADDITION AND SUBTRACTION USING 1'S COMPLEMENT

Addition using 1's complement

There are three different cases possible when we add two binary numbers which are as follows:

Case 1: Addition of the positive number with a negative number when the positive number has a greater magnitude.

Initially, calculate the 1's complement of the given negative number. Sum up with the given positive number. If we get the end-around carry 1, it gets added to the LSB.

Example: 1101 and -1001

First, find the 1's complement of the negative number 1001. So, for finding 1's complement, change all 0 to 1 and all 1 to 0. The 1's complement of the number 1001 is 0110.

Now, add both the numbers, i.e., 1101 and 0110;

1101+0110=1 0011

By adding both numbers, we get the end-around carry 1. We add this end around carry to the LSB of 0011.

0011+1=0100

Case 2: Adding a positive value with a negative value in case the negative number has a higher magnitude.

Initially, calculate the 1's complement of the negative value. Sum it with a positive number. In this case, we did not get the end-around carry. So, take the 1's complement of the result to get the final result.

Example: 1101 and -1110

First find the 1's complement of the negative number 1110. So, for finding 1's complement, we change all 0 to 1, and all 1 to 0. 1's complement of the number 1110 is 0001.

Now, add both the numbers, i.e., 1101 and 0001;

1101+0001= 1110

Now, find the 1's complement of the result 1110 that is the final result. So, the 1's complement of the result 1110 is 0001, and we add a negative sign before the number so that we can identify that it is a negative number.

Case 3: Addition of two negative numbers

In this case, first find the 1's complement of both the negative numbers, and then we add both these complement numbers. In this case, we always get the end-around carry, which get added to the LSB, and for getting the final result, we take the 1's complement of the result.

Example: -1101 and -1110 in five-bit register

Firstly find the 1's complement of the negative numbers 01101 and 01110. So, for finding 1's complement, we change all 0 to 1, and all 1 to 0. 1's complement of the number 01110 is 10001, and 01101 is 10010.

Now, we add both the complement numbers, i.e., 10001 and 10010;

10001+10010= 1 00011

By adding both numbers, we get the end-around carry 1. We add this end-around carry to the LSB of 00011.

00011+1=00100

Now, find the 1's complement of the result 00100 that is the final answer. So, the 1's complement of the result 00100 is 110111, and add a negative sign before the number so that we can identify that it is a negative number.

Subtraction using 1's complement

These are the following steps to subtract two binary numbers using 1's complement

In the first step, find the 1's complement of the subtrahend.

Next, add the complement number with the minuend.

If got a carry, add the carry to its LSB. Else take 1's complement of the result which will be negative

Example 1: 10101 - 00111

We take 1's complement of subtrahend 00111, which comes out 11000. Now, sum them. So,

10101+11000 =1 01101.

In the above result, we get the carry bit 1, so add this to the LSB of a given result, i.e., 01101+1=01110, which is the answer.

Example 2: 10101 - 10111

We take 1's complement of subtrahend 10111, which comes out 01000. Now, add both of the numbers. So,

10101+01000 =11101.

In the above result, we didn't get the carry bit. So calculate the 1's complement of the result, i.e., 00010, which is the negative number and the final answer.

Addition and Subtraction using 2's complement

Addition using 2's complement

There are three different cases possible when we add two binary numbers using 2's complement, which is as follows:

Case 1: Addition of the positive number with a negative number when the positive number has a greater magnitude.

Initially find the 2's complement of the given negative number. Sum up with the given positive number. If we get the end-around carry 1 then the number will be a positive number and the carry bit will be discarded and remaining bits are the final result.

Example: 1101 and -1001

First, find the 2's complement of the negative number 1001. So, for finding 2's complement, change all 0 to 1 and all 1 to 0 or find the 1's complement of the number 1001. The 1's complement of the number 1001 is 0110, and add 1 to the LSB of the result 0110. So the 2's complement of number 1001 is 0110+1=0111

Add both the numbers, i.e., 1101 and 0111;

1101+0111=1 0100

By adding both numbers, we get the end-around carry 1. We discard the end-around carry. So, the addition of both numbers is 0100.

Case 2: Adding of the positive value with a negative value when the negative number has a higher magnitude.

Initially, add a positive value with the 2's complement value of the negative number. Here, no end-around carry is found. So, we take the 2's complement of the result to get the final result.

Example: 1101 and -1110

First, find the 2's complement of the negative number 1110. So, for finding 2's complement, add 1 to the LSB of its 1's complement value 0001.

0001+1=0010

Add both the numbers, i.e., 1101 and 0010;

1101+0010= 1111

Find the 2's complement of the result 1110 that is the final result. So, the 2's complement of the result 1110 is 0001, and add a negative sign before the number so that we can identify that it is a negative number.

Case 3: Addition of two negative numbers

In this case, first, find the 2's complement of both the negative numbers, and then we will add both these complement numbers. In this case, we will always get the end-around carry, which will be added to the LSB, and forgetting the final result, we will take the2's complement of the result.

Example: -1101 and -1110 in five-bit register

Firstly find the 2's complement of the negative numbers 01101 and 01110. So, for finding 2's complement, we add 1 to the LSB of the 1's complement of these numbers. 2's complement of the number 01110 is 10010, and 01101 is 10011.

We add both the complement numbers, i.e., 10001 and 10010;

10010+10011= 1 00101

By adding both numbers, we get the end-around carry 1. This carry is discarded and the final result is the 2.s complement of the result 00101. So, the 2's complement of the result 00101 is 11011, and we add a negative sign before the number so that we can identify that it is a negative number.

Subtraction using 2's complement

These are the following steps to subtract two binary numbers using 2's complement

In the first step, find the 2's complement of the subtrahend.

Add the complement number with the minuend.

If we get the carry by adding both the numbers, then we discard this carry and the result is positive else take 2's complement of the result which will be negative.

Example 1: 10101 - 00111

We take 2's complement of subtrahend 00111, which is 11001. Now, sum them. So,

10101+11001 =1 01110.

In the above result, we get the carry bit 1. So we discard this carry bit and remaining is the final result and a positive number.

Example 2: 10101 - 10111

We take 2's complement of subtrahend 10111, which comes out 01001. Now, we add both of the numbers. So,

10101+01001 =11110.

In the above result, we didn't get the carry bit. So calculate the 2's complement of the result, i.e., 00010. It is the negative number and the final answer.

LOGIC GATES

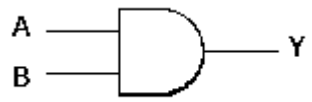
Logic gates are the basic building blocks of any digital system. It is an electronic circuit having one or more than one input and only one output. The relationship between the input and the output is based on a **certain logic**. Based on this, logic gates are named as AND gate, OR gate, NOT gate etc.

AND Gate

A circuit which performs an AND operation is shown in figure. It has n input (n >= 2) and one output.

Y=A . B or AB

Logic diagram



Truth Table

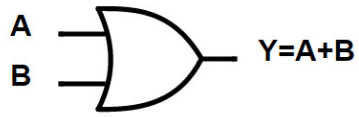
2 Input AND gate		
A	B	A.B
0	0	0
0	1	0
1	0	0
1	1	1

OR Gate

A circuit which performs an OR operation is shown in figure. It has n input (n >= 2) and one output.

Y=A + B

Logic diagram



Truth Table

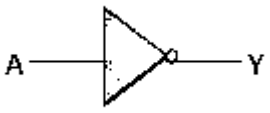
2 Input OR gate		
A	B	A+B
0	0	0
0	1	1
1	0	1
1	1	1

NOT Gate

NOT gate is also known as **Inverter**. It has one input A and one output Y.

Y= A' or $\bar{A}$

Logic diagram



Truth Table

NOT gate	
A	$\bar{A}$
0	1
1	0

NAND Gate

A NOT-AND operation is known as NAND operation. It has n input (n >= 2) and one output.

Y=(A.B)' or $\overline{AB}$ =A'+B'

Logic diagram

Truth Table

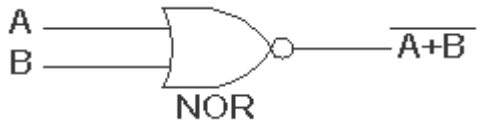
2 Input NAND gate		
A	B	$\overline{A.B}$
0	0	1
0	1	1
1	0	1
1	1	0

NOR Gate

A NOT-OR operation is known as NOR operation. It has n input (n >= 2) and one output.

Y=(A+B)' or $\overline{A+B}$ =A'.B'

Logic diagram



Truth Table

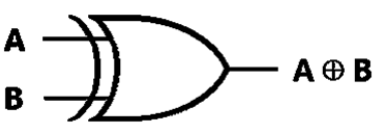
2 Input NOR gate		
A	B	$\overline{A+B}$
0	0	1
0	1	0
1	0	0
1	1	0

XOR Gate

XOR or Ex-OR gate is a special type of gate. It can be used in the half adder, full adder and subtractor. The exclusive-OR gate is abbreviated as EX-OR gate or sometime as X-OR gate. It has n input (n >= 2) and one output.

Y= A ⊖ B

Logic diagram



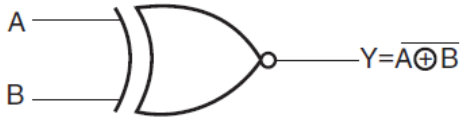
Truth Table

2 Input EXOR gate		
A	B	A⊕B
0	0	0
0	1	1
1	0	1
1	1	0

XNOR Gate

XNOR gate is a special type of gate. It can be used in the half adder, full adder and subtractor. The exclusive-NOR gate is abbreviated as EX-NOR gate or sometime as X-NOR gate. It has n input (n >= 2) and one output.

Logic diagram



$$Y = (\overline{A \oplus B}) = (A.B + \overline{A}.\overline{B})$$

Truth Table

2 Input EXNOR gate		
A	B	$\overline{A \oplus B}$
0	0	1
0	1	0
1	0	0
1	1	1

UNIVERSAL GATES:

A universal gate is a gate which can implement any Boolean function without need to use any other gate type. The NAND and NOR gates are universal gates. In practice, this is advantageous since NAND and NOR gates are economical and easier to fabricate and are the basic gates used in all IC digital logic families. In fact, an AND gate is typically implemented as a NAND gate followed by an inverter not the other way around!! Likewise, an OR gate is typically implemented as a NOR gate followed by an inverter not the other way around!!

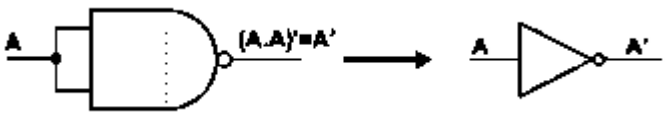
NAND Gate is a Universal Gate:

To prove that any Boolean function can be implemented using only NAND gates, we will show that the AND, OR, and NOT operations can be performed using only these gates.

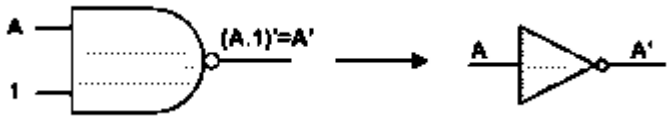
Implementing an Inverter Using only NAND Gate

The figure shows two ways in which a NAND gate can be used as an inverter (NOT gate).

1. All NAND input pins connect to the input signal A gives an output A'.



2. One NAND input pin is connected to the input signal A while all other input pins are connected to logic 1.

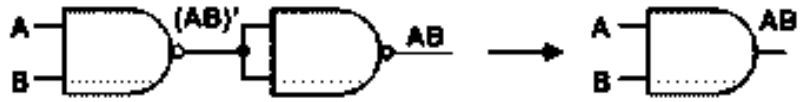


The output will be A'.

Implementing AND Using only NAND Gates

An AND gate can be replaced by NAND gates as shown in the figure.

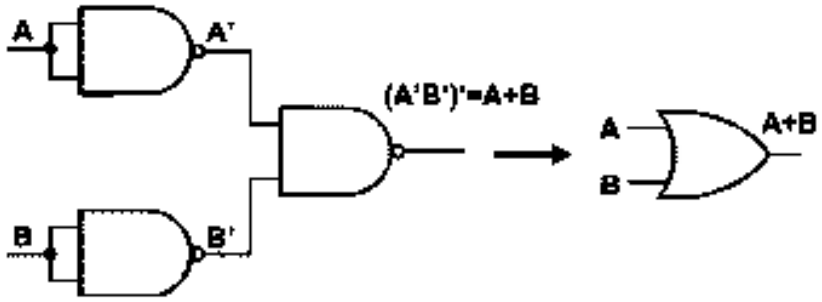
The AND is replaced by a NAND gate with its output complemented by a NAND gate inverter.



Implementing OR Using only NAND Gates

An OR gate can be replaced by NAND gates as shown in the figure

(The OR gate is replaced by a NAND gate with all its inputs complemented by NAND gate inverters).



Thus, the NAND gate is a universal gate since it can implement the AND, OR and NOT functions.

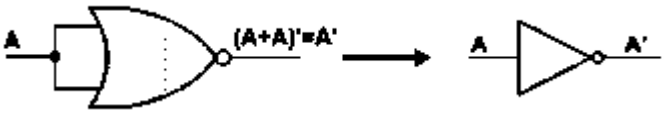
NOR Gate is a Universal Gate:

To prove that any Boolean function can be implemented using only NOR gates, we will show that the AND, OR, and NOT operations can be performed using only these gates.

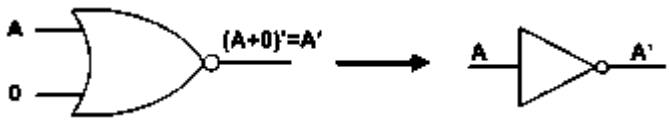
Implementing an Inverter Using only NOR Gate

The figure shows two ways in which a NOR gate can be used as an inverter (NOT gate).

1. All NOR input pins connect to the input signal A gives an output A'.



2. One NOR input pin is connected to the input signal A while all other input pins are connected to logic 0.

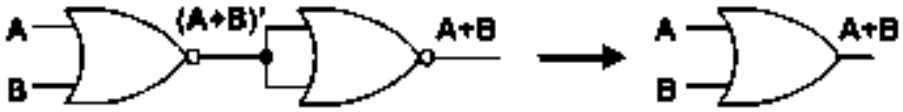


The output will be A'.

Implementing OR Using only NOR Gates

An OR gate can be replaced by NOR gates as shown in the figure.

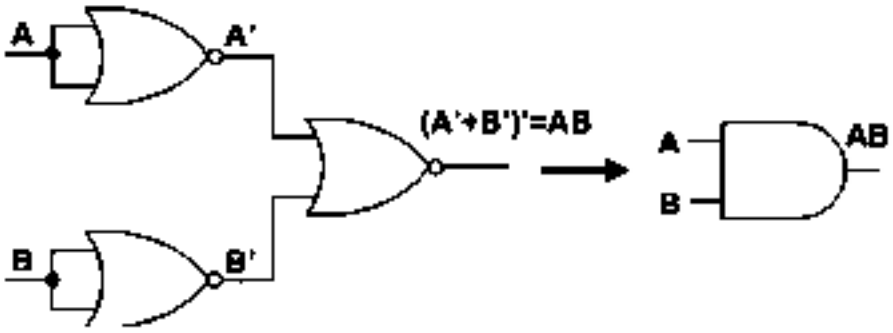
The OR is replaced by a NOR gate with its output complemented by a NOR gate inverter.



Implementing AND Using only NOR Gates

An AND gate can be replaced by NOR gates as shown in the figure.

The AND gate is replaced by a NOR gate with all its inputs complemented by NOR gate inverters.



Thus, the NOR gate is a universal gate since it can implement the AND, OR and NOT functions.

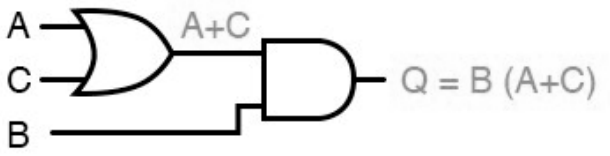
IMPLEMENTATION OF BOOLEAN FUNCTION USING LOGIC GATES

Any Boolean function can be represented by using a number of logic gates by interconnecting them. Logic gates implementation or logic representation of Boolean functions is very simple and easy form.

The implementation of Boolean functions by using logic gates involves in connecting one logic gate's output to another gate's input and involves in using AND, OR, NAND and NOR gates.

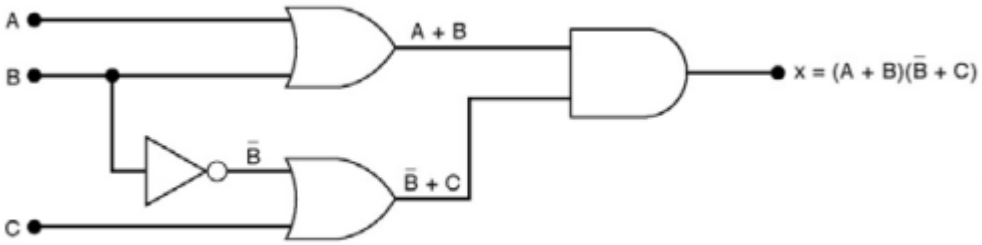
The sum of product or SOP form is represented by using basic logic gates like AND gate and OR gate. The SOP form implementation will have the AND gate at its input side and as the output of the function is the sum of all product terms, it has an OR gate at its output side. The product of sums or POS form can be represented by using basic logic gates like AND gate and OR gates. The POS form implementation will have the OR gate at its input side and as the output of the function is product of all sum terms, it has AND gate at its output side.

Example 1: Implement the Boolean function by using basic logic gates. F= B(A + C)

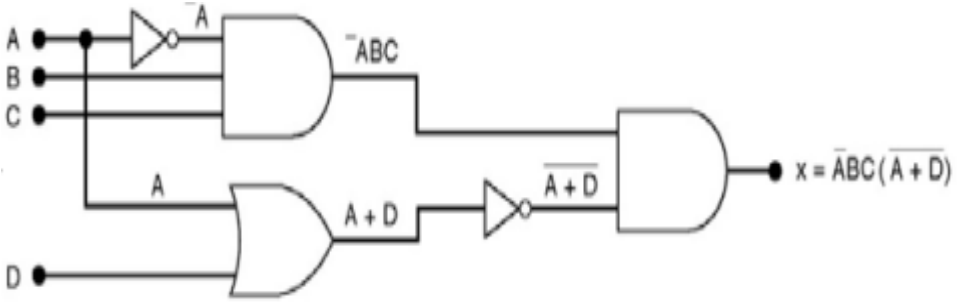


Example 2: Implement the Boolean function by using basic logic gates.

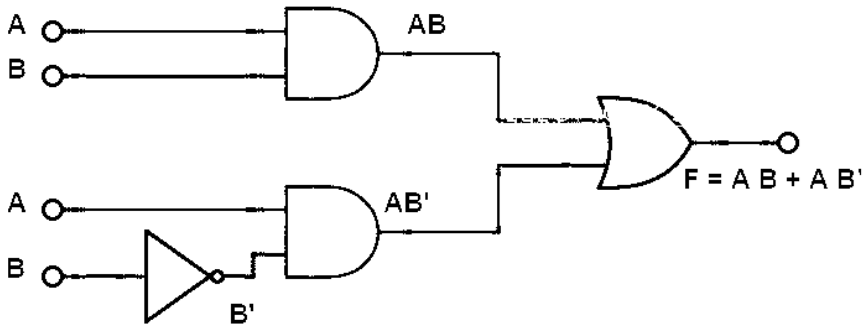
x = (A + B)(B' + C)



Example 3: Implement the Boolean function by using basic logic gates x = A'BC(A+ D)'

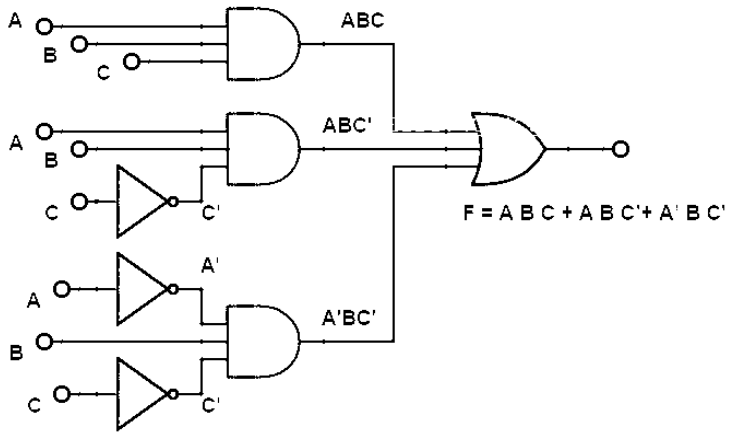


Example 4: Implement the Boolean function by using basic logic gates. F = A B + A B'



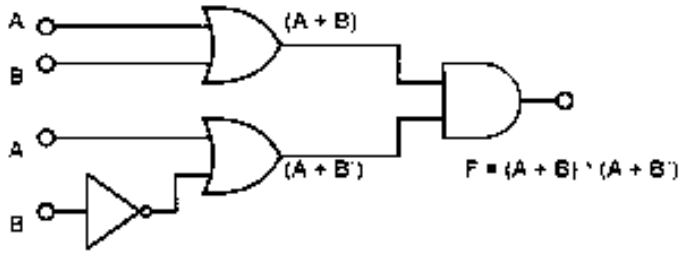
Example 5: Implement the Boolean function by using basic logic gates.

F = A B C + A B C'+ A' B C'



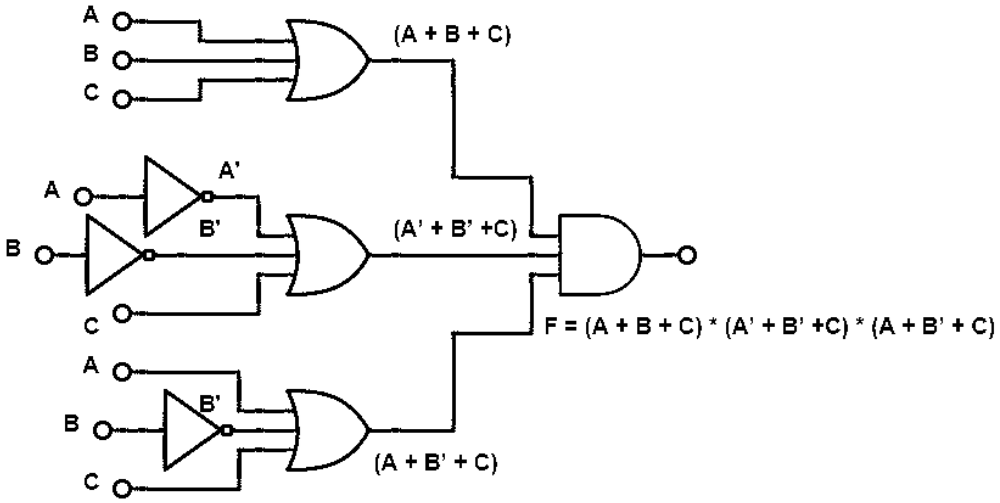
Example 6: Implement the Boolean function by using basic logic gates.

F = (A + B) * (A + B')



Example 7: Implement the Boolean function by using basic logic gates.

F = (A + B + C) * (A' + B' +C) * (A + B' + C)



Implementation of Boolean functions using Universal logic gates

'Universal logic gates' are NAND gate and NOR gates. The reason behind this is, NAND gate and NOR gate can perform (or can function like) all the 3 basic gates, such as AND gate, OR gate and NOT gate. We can design any basic logic gate by using NAND gate or NOR gate. This is why they are called as "Universal gates".

Let's see the implementation of the Boolean functions using universal logic gates.

Implementation of Boolean functions using NAND gates

NAND gate is a logical combination of AND gate and NOT gate and this can function like AND gate, OR gate and NOT gate. So we use NAND gates to implement the Boolean function. The important thing to remember about NAND gate is this is the inverse of basic AND gate. This means the output of the NAND gate is equal to the complement of the output of the AND gate.

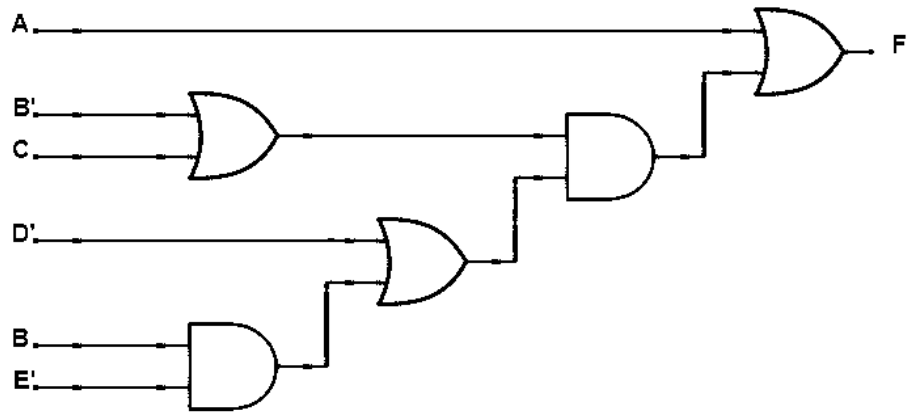
Let's see an example to understand the implementation.

Implement the Boolean function by using a NAND logic gate.

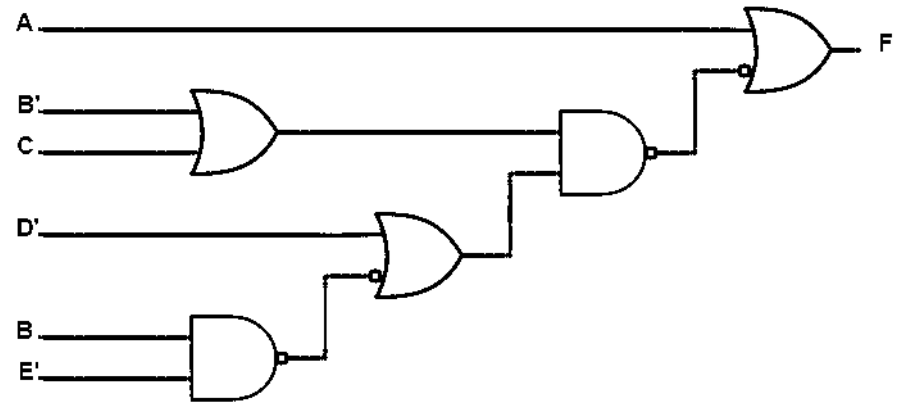
F (A, B, C, D, E) = A + (B' + C) (D' + BE')

In NAND gate implementation, we use NAND gates at both input and output side. Observe the designed logic diagram below. The step by step procedure to implement the given Boolean function using NAND gates is shown below.

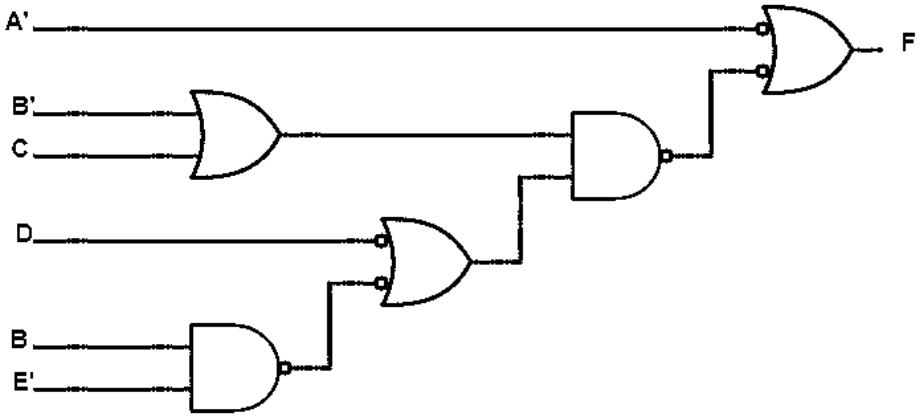
First, the given Boolean function or equation should be represented using AND-OR gates. The AND-OR implementation is shown below.



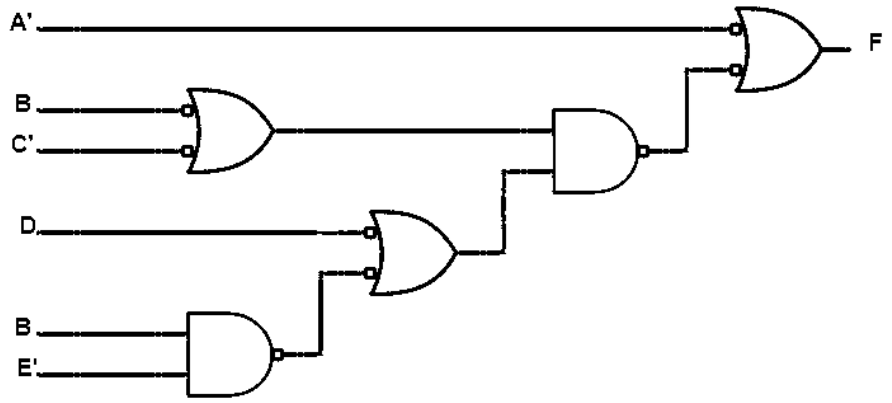
In order to convert the AND gates into NAND gates, a bubble (complement) is introduced at the output of the AND gate. To compensate the bubble, the input of the next gate is also introduced with a bubble. The implementation is shown below.



To impose uniformity at the input, if a gate has one input with a bubble, the other input is also introduced with a bubble. Again, in order to compensate the bubble, the output of the preceding gate is introduced with bubble or complement the literal. The same is shown in the following figure.

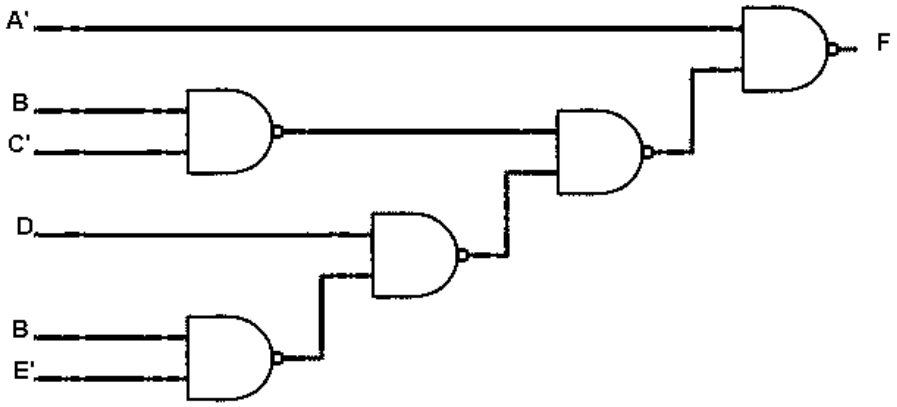


If an OR gate is not having any bubble at either of the inputs, bubbles are introduced and are appropriately compensated as shown in the figure below.



An OR gate with two complemented inputs is equivalent to a NAND gate (according to DeMorgan's Law A'+B' = (AB)'). Hence, replacing the OR gate, which is having two

complemented inputs, with NAND gate, we get the final structure of the implementation of the Boolean function using NAND gates. The final implementation is shown below.



Implementation of Boolean functions using NOR gates

NOR gate is the combination of OR gate and NOT gate and this can function like AND gate, OR gate and NOT gate. So we use NOR gate to implement the Boolean functions. The important thing to remember about NOR gate is this is the inverse of basic OR gate. This means the output of the NOR gate is equal to the output of the OR gate. Let's see an example to understand the implementation.

Implement the Boolean function by using NOR logic gate.

g (A, B, C, D, E, F) = (A E) + (B D E) + (B C E F)

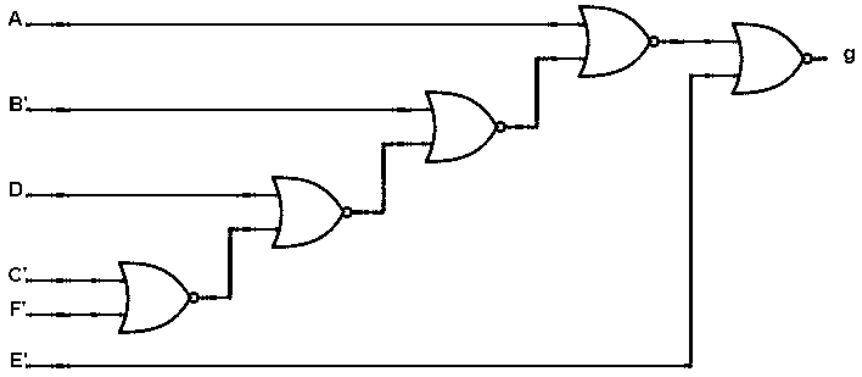
We can solve the given equation as

g (A, B, C, D, E, F) = AE + BDE + BCEF

= (A + BD + BCF) E

= (A + B (D + CF)) E

In NOR gate implementation, we use NOR gates at both input and output side. Observe the designed logic diagram below.



g = (A.E) + (B.D.E) + (B.C.E.F)