Arithmetic Operation

Addition of Binary Numbers:

The addition of two binary numbers is performed in exactly the same manner as the addition of decimal numbers. Only four cases can occur in adding the two binary digits (bits) in any position. They are:

0+0=0
1+0=1
0+1=1
1+1=0 with carry 1
1+1+1=1 with carry 1

Examples:

011 (3)	1001 (9)	11.011 (3.375)	1010 (10)
+ 110 (6)	+ 1111 (15)	+ 10.110 (2.750)	+ 1101 (13)
1001 (9)	11000 (24)	110.001 (6.125)	10111 (23)

Subtraction of Binary Numbers (Using Direct Method):

The four basic rules for subtracting binary digits are:

0-0=0
1-1=0
1-0=1
0-1=1 with borrow 1

Examples:

→ when 1 is borrowed, a 0 is left .
→ when 1 is borrowed, making 10 instead of 0.

11 (3)	11 (3)	101 (5)
- 01 (1)	- 10 (2)	- 011 (3)
10 (2)	01 (1)	010 (2)

Subtraction of Binary Numbers (Using Complement Method):

The 1's complement and the 2's complement of a number are important because they permit the representation of negative numbers. The method of 2's complement arithmetic is used in computer to handle negative numbers

- 1's complement: the 1's complement form of any binary number is obtained simply by changing each 0 in the number to a 1 and each 1 to a 0. In other word, change each bit to its complement. For example:

1 0 1 1 0 1	Binary No.	0 1 1 0 1 0	Binary No.
↓ ↓ ↓ ↓ ↓ ↓		↓ ↓ ↓ ↓ ↓ ↓	
0 1 0 0 1 0	1's complement	1 0 0 1 0 1	1's complement

- 2's complement: the 2's complement form of a binary number is formed simply by taking the 1's complement of the number and adding 1 to the least significant bit position.

2's complement = (1's complement) +1

Example: find 2's complement of 10110010.

1 0 1 1 0 0 1 0	binary number.
↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓	
0 1 0 0 1 1 0 1	1's complement
+ 1	adding 1
0 1 0 0 1 1 1 0	2's complement

Example: find $11010_{(2)} - 10000_{(2)}$ using 1's complement method (Case 1).

As long as the carry appear, the number is positive and a carry must be added to the result. ←

```
  11010
+ 01111    1's complement of 10000
 101001
+     1
  01010
```

$11010_{(2)} - 10000_{(2)} = 01010_{(2)}$

Example: find $10000_{(2)} - 11010_{(2)}$ using 1's complement method (Case 2).

As long as no carry appear, the number is negative, then 1's complementing of the final result in needed. ←

```
  10000
+ 00101    1's complement of 11010
  10101
  ↓ ↓ ↓ ↓ ↓   1's complement
  01010
```

$10000_{(2)} - 11010_{(2)} = -01010_{(2)}$

Example: find $11010_{(2)} - 10000_{(2)}$ using 2's complement method (Case 3).

As long as the carry appear, the number is positive and a carry must be discarded ←

```
  11010
+ 10000     2's complement of 10000
 101010
```

$$11010_{(2)} - 10000_{(2)} = 01010_{(2)}$$

Example: find $10000_{(2)} - 11010_{(2)}$ using 2's complement method (Case 4).

As long as no carry appear, the number is negative, then 2's complementing of the final result in needed. ←

```
  10000
+ 00110     2's complement of 11010
  10110
  ↓↓↓↓↓ 2's complement
  01010
```

$$10000_{(2)} - 11010_{(2)} = -01010_{(2)}$$

Multiplication of Binary Numbers:

The numbers in a multiplication are the ***multiplican***d, the ***multiplier***, and the ***product***. These are illustrated in the following decimal multiplication:-

the multiplication rules for binary numbers are:

$0 \times 0 = 0$

$0 \times 1 = 0$

$1 \times 0 = 0$

$1 \times 1 = 1$

```
  8   ⟶ multiplicand
× 3   ⟶ multiplier
 24   ⟶ product
```

Example: find the product of $100_{(2)}$ and $010_{(2)}$.

$$100_{(2)} \times 010_{(2)} = 01000_{(2)}$$

```
  100      (4)
× 010      (2)
  000
 1000  +
00000  +
01000      (8)
```

Division of Binary Numbers:

The numbers in a division are the ***dividend***, the ***divisor***, and the ***quotient***. These are illustrated in the following standard.

to illustrate, consider the following division examples:

$$\frac{dividend}{divisor} = quotient$$

```
   110          0010.1        010.1
100|11000    100|1010     10|101
   100          100          10
   0100         00100        0010
    100          100          10
    0000         000          00
```

Addition of Hexadecimal Numbers:

Hex numbers are used extensively in machine-language computer programming and in conjunction with computer memories. When working in these areas, there will be situations where hex numbers have to be added or subtracted. The addition can be done in the same manner as decimal addition. Let's add the hex numbers 58 and 24, 58 and 4B.

```
  58        3AF        58
+ 24      + 23C      + 4B
  7C        5EB        93
```

Examples: add the following hexadecimal numbers.

(a) $23_{(16)}+16_{(16)}$ (b) $58_{(16)}+22_{(16)}$ (c) $DF_{(16)}+AC_{(16)}$

```
  23
+ 16
  39
```

right column: $3_{(16)}+6_{(16)} = 3_{(10)}+6_{(10)} = 9_{(10)} = 9_{(16)}$

left column: $2_{(16)}+1_{(16)} = 2_{(10)}+1_{(10)} = 3_{(10)} = 3_{(16)}$

```
  58
+ 22
  7A
```

right column: $8_{(16)}+2_{(16)} = 8_{(10)}+2_{(10)} = 10_{(10)} = A_{(16)}$

left column: $5_{(16)}+2_{(16)} = 5_{(10)}+2_{(10)} = 7_{(10)} = 7_{(16)}$

DF
+ AC
18B

right column: $F_{(16)}+C_{(16)} = 15_{(10)}+12_{(10)} = 27_{(10)}$
$= 27_{(10)}-16_{(10)} = 11_{(10)} = B_{(16)}$ with a carry of 1

left column: $D_{(16)}+A_{(16)}+1_{(16)}= 13_{(10)}+10_{(10)}+1_{(10)}= 24_{(10)}$
$= 24_{(10)}-16_{(10)} = 8_{(10)} = 8_{(16)}$ with a carry of 1

Subtraction of Hexadecimal Numbers (Using Direct Method):

Reverse operation of addition may be used as a direct way to subtract hexadecimal numbers as shown in the following examples:

D3A
- F4
C46

right column: $A_{(16)}- 4_{(16)} = 10_{(10)} - 4_{(10)} = 6_{(10)} = 6_{(16)}$
middle column: $3_{(16)} - F_{(16)}= 3_{(10)} - 15_{(10)}$ (need borrow)
$= 19_{(10)} - 15_{(10)}= 4_{(10)} =4_{(16)}$
left column: $D_{(16)} - 1_{(16)} = C_{(16)}$

84
- 2A
5A

right column : $4_{(16)}- A_{(16)} = 4_{(10)} - 10_{(10)}$ (need borrow)
$= 20_{(10)} - 10_{(10)}= 10_{(10)} =A_{(16)}$
left column : $8_{(16)} - 2_{(16)} - 1_{(16)}= 5_{(16)}$

Subtraction of Hexadecimal Numbers (Using Complement Method):

Remember that hex numbers are just an efficient way to represent binary numbers. Thus we can subtract hex numbers using the same method we used for binary numbers. In order to find the complement of hex numbers, two ways are found

- first way:

7	3	A	→ hex number
0111	0011	1010	→ convert to binary
1000	1100	0101	→ 1's complement representation
1000	1100	0110	→ 2's complement representation
8	C	6	→ conversion back to hex

- Second way: this procedure is quicker, subtract each hex digit from F, and then add 1. let's try this for the same hex number from the example above:

F	F	F	
-7	-3	-A	→ Subtract each digit from F
8	C	5	
		+1	→ adding 1
8	C	6	→ hex. Equivalent of 2's comp.

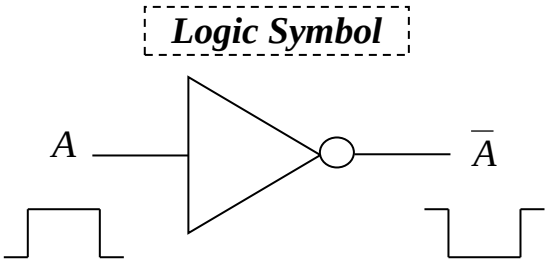
1- Inverter (NOT Gate):

The inverter performs the operation called inversion or complementation. The purpose of the inverter is to change the one logic level to the opposite level. In terms of bits, it changes a 1 to 0 and a 0 to a 1.

Inverter Truth Table

Input (A)	Output ($\overline{A}$)
0	1
1	0

Logic Symbol



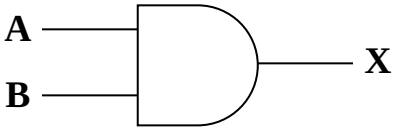
2- AND Gate:

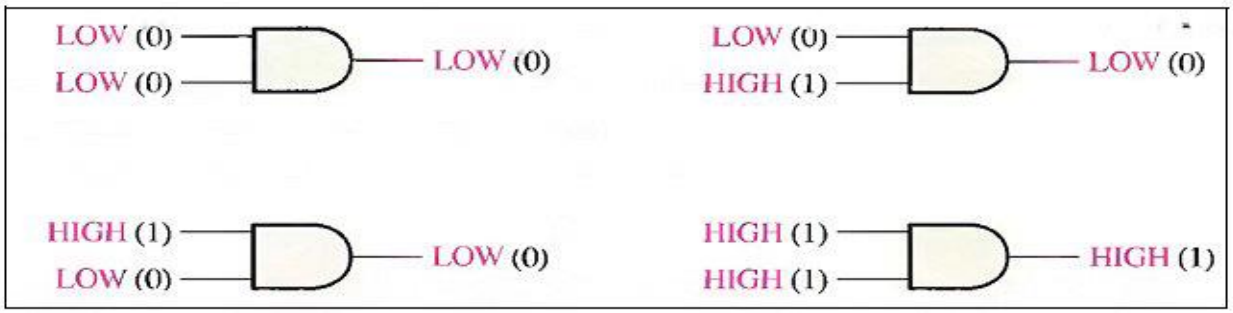
The AND Gate is one of the basic gates from which all logic functions are constructed. An AND gate can have two or more inputs and performs what is known as logical multiplication. Figure below, all possible logic levels for a 2-input AND gate.

AND Gate Truth Table

Inputs		Output
A	**B**	**X**
0	0	0
0	1	0
1	0	0
1	1	1

Logic Symbol





The total number of possible combinations of binary inputs to a gate is determined by the following formula:

$$N = 2^n$$

Where N is the total possible combinations and n is the number of input variables. To illustrate,

For two input variables: $N=2^2=4$

For three input variables: $N=2^3=8$

For four input variables: $N=2^4=16$

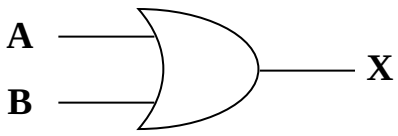
3- OR Gate:

The OR gate is one of the basic gates from which all logic functions are constructed. An OR gate can have two or more inputs and performs what is know as logical addition. Figure below, all possible logic levels for a 2-input OR gate.

OR Gate Truth Table

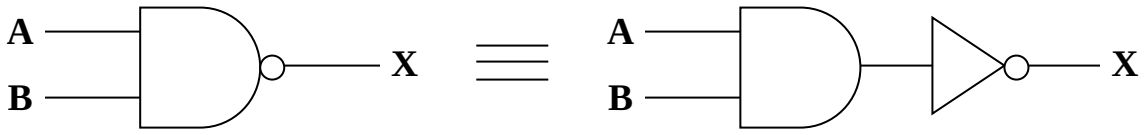
Inputs		Output
A	**B**	**X**
0	0	0
0	1	1
1	0	1
1	1	1

Logic Symbol



4- NAND Gate:

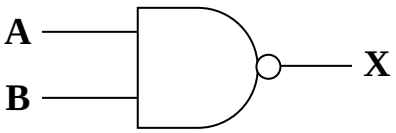
The NAND gate is a popular logic element because it can be used as a universal gate; that is; NAND gate can be used to perform the AND, OR, and Inverter operations, or any combination of these operations. The term NAND is a contraction of NOT-**AND** and implies an AND function with a complemented (Inverted) output. Figure below Operation of a 2-input NAND gate.



NAND Gate Truth Table

Inputs		Output
A	**B**	**X**
0	0	1
0	1	1
1	0	1
1	1	0

Logic Symbol



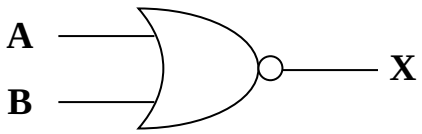
5- NOR Gate:

The NOR gate, like the NAND gate, is a very useful logic element because it can also be used as a universal gate; that is; NOR gate can be used to perform the AND, OR, and Inverter operations, or any combination of these operations.

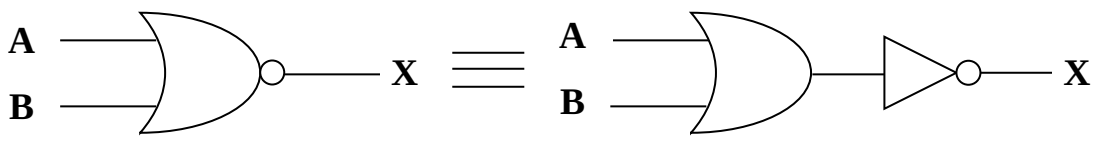
NOR Gate Truth Table

Inputs		Output
A	**B**	**X**
0	0	1
0	1	0
1	0	0
1	1	0

Logic Symbol



The term NOR is contraction of **NOT-OR** and implies an OR function with an inverted output. Figure below Operation of a 2-input NOR gate.



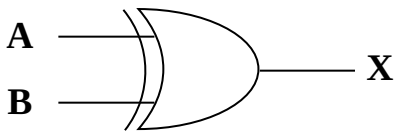
6- Exclusive-OR Gate (XOR):

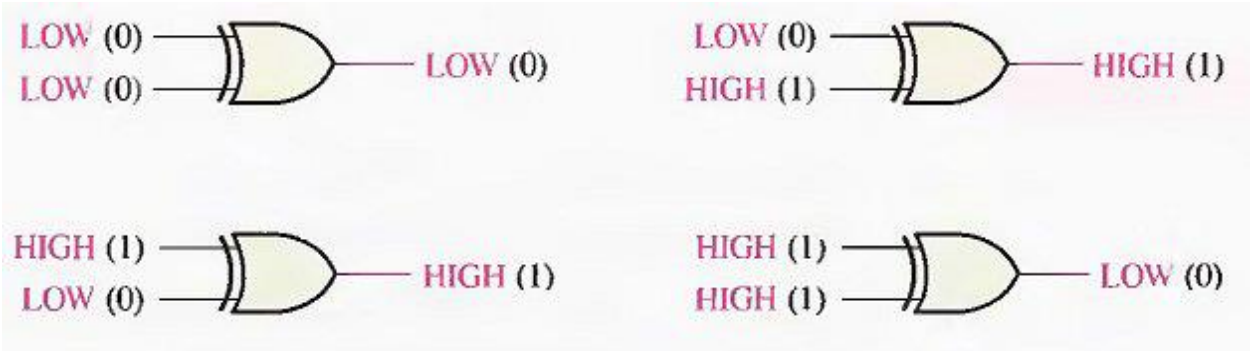
The Exclusive-OR is actually formed by a combination of other gates. Figure below, all possible logic levels for an exclusive-OR gate.

XOR Gate Truth Table

Inputs		Output
A	**B**	**X**
0	0	0
0	1	1
1	0	1
1	1	0

Logic Symbol





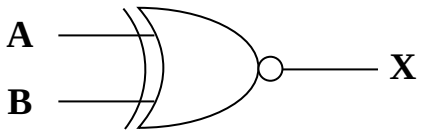
7- Exclusive-NOR Gate (XNOR):

The Exclusive-NOR is actually formed by a combination of other gates. Figure below, all possible logic levels for an exclusive-NOR gate.

XNOR Gate Truth Table

Inputs		Output
A	**B**	**X**
0	0	1
0	1	0
1	0	0
1	1	1

Logic Symbol



Example: (a) Develop the truth table for a 3-input AND gate.

(b) Determine the total number of possible input combinations for a 5-input AND gate.

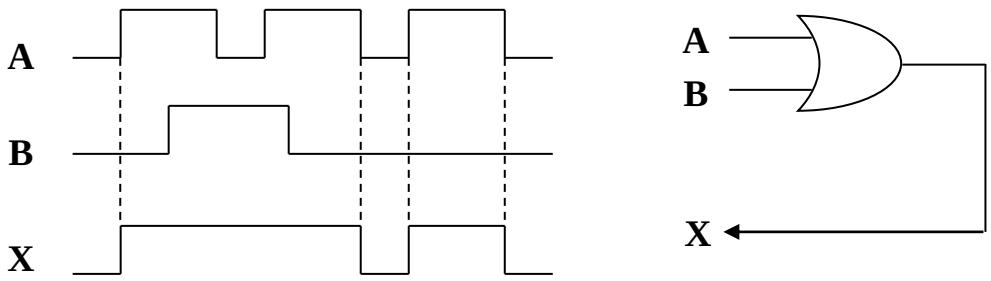
For branch (a) there are eight possible input combinations for a 3-input AND gate.

Input			Output
A	**B**	**C**	**X**
0	**0**	**0**	**0**
0	**0**	**1**	**0**
0	**1**	**0**	**0**
0	**1**	**1**	**0**
1	**0**	**0**	**0**

1	**0**	**1**	**0**
1	**1**	**0**	**0**
1	**1**	**1**	**1**

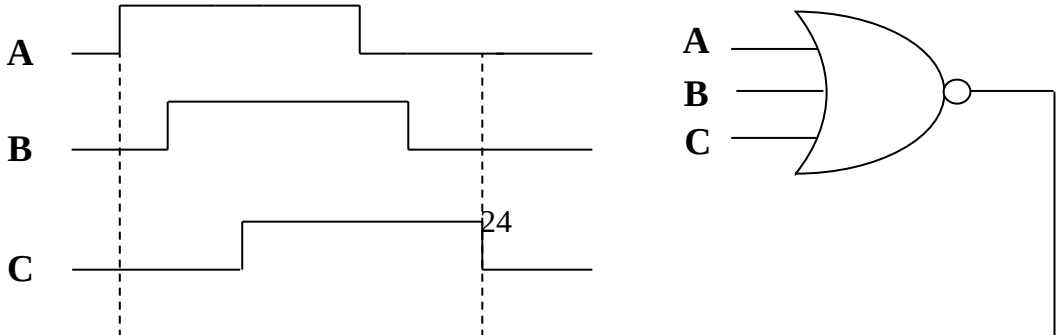
For branch (b), $N=2^5=32$. There are 32 possible combinations of input bits for a 5-input AND gate.

Example: for the two input waveforms, A and B, sketch the output waveform, showing its proper relation to the inputs.



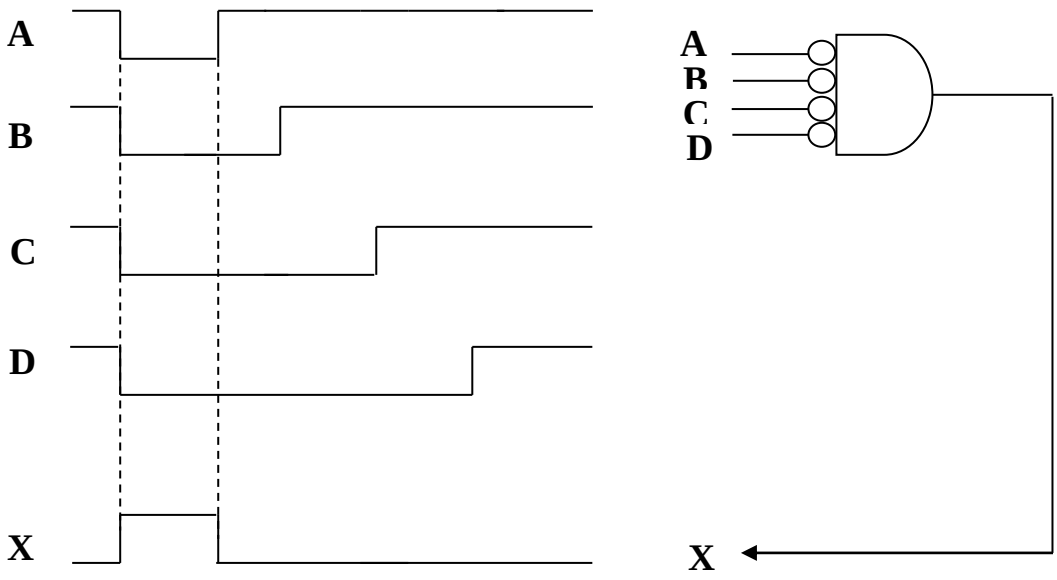
When either or both inputs are **HIGH**, the output is **HIGH** as shown by the output waveform X in the timing diagram.

Example: Sketch the output waveform for the 3-input NOR gate, showing the proper relation to the input.



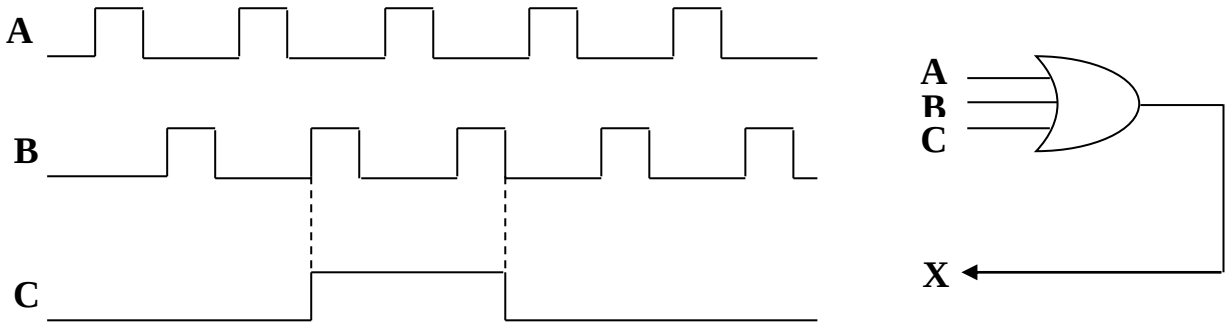
The output X is **LOW** when any input is **HIGH** as shown by the output waveform X in the timing diagram.

Example: For the 4-input NOR gate operating as a negative-AND. determine the output relative to the inputs.

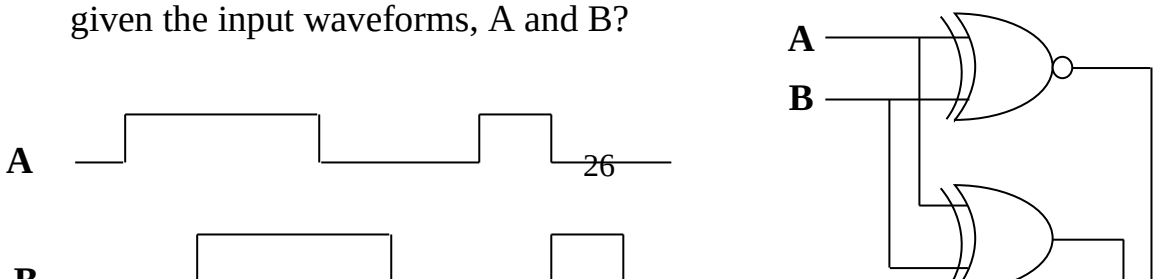


Review Questions:

1. When is the output of an AND gate HIGH?
2. Describe the truth table for a five-input AND gate?
3. How can you use an XOR gate to detect when two bits are different?
4. For the 3-input OR gate, determine the output waveform in proper relation to the inputs?



5. Determine the output waveform for the XOR gate and for the XNOR gate, given the input waveforms, A and B?



Choose the right answer:

1. An inverter performs an operation known as:

   (a) Complementation (b) Assertion

   (c) Inversion (d) both a and c

2. The output of an OR gate with inputs A,B, and C is a 1 when:

   (a) A=1, B=1, C=1 (b) A=0, B=0, C=1

   (c) A=0, B=0, C=0 (d) All (e) only a and b

3. A pulse is applied to each input of a 2-input NAND gate. One pulse goes **HIGH** at t=0 and goes **LOW** at t=1*ms*, the other pulse goes **HIGH** at t=0.8*ms* and goes back **LOW** at t=3*ms*. the output pulse can be described as follows:

   (a) It goes **LOW** at t=0 and back **HIGH** at t=3*ms*.

   (b) It goes **LOW** at t=0.8*ms* and back **HIGH** at t=3*ms*.

   (c) It goes **LOW** at t=0.8*ms* and back **HIGH** at t=1*ms*.

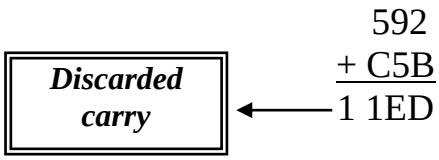
   (d) It goes **HIGH** at t=0.8*ms* and back **LOW** at t=1*ms*.

4. when the input to an inverter is **HIGH**, the output is :

   (a) **HIGH** (b) **LOW**

Example: subtract $3A5_{(16)}$ from $592_{(16)}$.

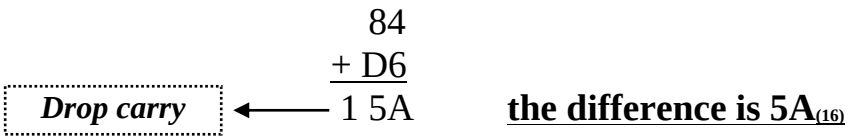
First, covert 3A5 to its 2's complement form by using either method presented above. The result is C5B. Then add this to 592.



Example: subtract the following hexadecimal numbers:

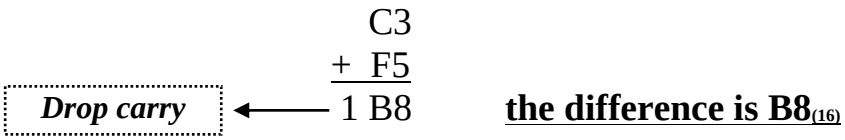
(a) 84 – 2A (b) C3 – 0B

For branch (a): the 2's complement of 2A = D6



the difference is $5A_{(16)}$

For branch (b): the 2's complement of 0B = F5



the difference is $B8_{(16)}$

Multiplication of Hexadecimal Numbers:

The multiplication of hex numbers is well illustrated in the following example:

$$\begin{array}{r} 3A \\ \times\ \ F \\ \hline 366 \end{array}$$

right column : $A_{(16)} \times F_{(16)} = 10_{(10)} \times 15_{(10)} = 150_{(10)} = 96_{(16)}$
$= 6$ with carry 9

left column : $3_{(16)} \times F_{(16)} + 9_{(16)} = 3_{(10)} \times 15_{(10)} + 9_{(10)} = 45_{(10)} + 9_{(10)} = 54_{(10)}$
$= 36_{(16)}$

Review Questions:

1. perform the following binary additions:
   (a) 1101+1010 (b) 10111+01101
2. perform the following binary subtractions:
   (a) 11101- 0100 (b) 1001- 0111
3. perform the indicated binary operation:
   (a) 110 ×111 (b) 1100 ÷ 011
4. determine the 1's complement of each binary number:
   (a) 11010 (b) 001101
5. determine the 2's complement of each binary number:
   (a) 10111 (b) 010001
6. subtract the hexadecimal numbers:
   (a) $75_{(16)} - 21_{(16)}$ (b) $94_{(16)} - 5C_{(16)}$
7. add the hexadecimal numbers directly:
   (a) $18_{(16)} + 34_{(16)}$ (b) $3F_{(16)} + 2A_{(16)}$
8. multiply the following pairs of binary numbers:
   (a) 101.101 × 110.010 ***Ans.:100011.00101***
   (b) 0.1101 × 0.1011 ***Ans.:0.10001111***
9. perform the following divisions:
   (a) 10110.1101 ÷ 1.1 ***Ans.:1111.0011***
   (b) 111111 ÷ 1001 ***Ans.:111***

Digital Codes

1- Binary Coded Decimal (BCD):

When numbers, letters, or words are represented by a special group of symbols, this is called encoding, and the group of symbols is called a code. Probably one of the familiar codes is the Morse code, where series of dots and dashes represent letters of the alphabet. We have seen that decimal numbers can be represented by an equivalent binary number. The group of 0s and 1s in the binary number can be thought of as a code representing the decimal number. When a decimal number is represented by its equivalent binary number, we call it (straight binary coding). We have seen that conversion between decimal and binary can become long and complicated for large numbers. For this reason, a means of encoding decimal numbers that combines some features of both the decimal and binary systems is used in certain situations.

The 8421 code is a type of binary coded decimal (BCD) code. Binary coded decimal means that each decimal digit, 0 though 9, is represented by a binary code of 4 bits. The designation 8421 indicates the binary weights of the four bits (2^3,2^2,2^1,2^0). The ease of conversion between 8421 code numbers and the familiar decimal numbers is the main advantage of this code. All you have to remember are the ten binary combinations that represent the ten decimal digits as shown in Table (1). The 8421 code is the predominant BCD code, and when referring to BCD, it always means the 8421 code unless otherwise stated.

Table (1)

Decimal digit	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
BCD	**0000**	**0001**	**0010**	**0011**	**0100**	**0101**	**0110**	**0111**	**1000**	**1001**

To illustrate the BCD code, take a decimal number such as 874. Each digit is changed to its binary equivalent as follows: