Chapter II

Number System and Codes

Aim

The aim of this chapter is to:

- explain different types of number systems
- elucidate binary arithmetic
- explicate different types of codes

Objectives

The objectives of this chapter are to:

- explain conversion of digits from a number system to another
- elucidate binary system and binary arithmetic
- explicate inter-conversion of digits from one number system to another

Learning outcome

At the end of this chapter, you will be able to:

- understand number system into decimal, binary, octal and hex number systems
- describe methods of binary arithmetic
- identify BCD, ASCII and code gray

2.1 Introduction

Number system is simply the ways to count things. Aim of any number system is to deal with certain quantities which can be measured, monitored, recorded, manipulated arithmetically, observed and utilised. Each quantity has to be represented by its value as efficiently and accurately as is necessary for any application. The numerical value of a quantity can be basically expressed in either analog (continuous) or digital (step by step) method of representation. In analog method, a quantity is expressed by another quantity which is proportional to the first. For example, the voltage output of an amplifier is measured by a voltmeter.

The angular position of the needle of the voltmeter is proportional to the voltage output of the amplifier. Yet another example is of a thermometer. The height to which the mercury rises is proportional to the temperature. In both these examples, the value of voltage and temperature can be anywhere between zero and the maximum limit. In digital method, the value of a quantity is expressed by some symbols which are called digits, and not by a quantity which is proportional to the first. In a digital watch, the time, which changes continuously, is expressed by digits which do not change continuously. It is clear from the examples that the accuracy of the value of an analog quantity generally depends upon the judgement of the observer.

Digital technology is different from analog technology. Many number systems are being used in digital technology. Most common amongst them are decimal, binary, octal, and hexadecimal systems.

We are most familiar with the decimal number system, because we use it every day. It is the base-10 or radix-10 system. Note that there is no symbol for “10” or for the base of any system. We count 1 2 3 4 5 6 7 8 9, and then insert a 0 in the first column and add a new left column, starting at 1 again. Then we count 1-9 in the first column again. (People use the base-10 system because we have 10 fingers!). Each column in our system stands for a power of 10 starting at 100.

All computers use the binary system. Following section provides an overview of the binary system.

2.2 Binary System

- In the binary number system (base of 2), there are only two digits: 0 and 1 and the place values are 2^0, 2^1, 2^2, 2^3 etc. Binary digits are abbreviated as bits. For example, 1101 is a binary number of 4 bits (ie., it is a binary number containing four binary digits.)
- A binary number may have any number of bits. Consider the number 11001.01 1. Note the binary point (counterpart of decimal point in decimal number system) in this number.
- Each digit is known as a bit and can take only two values 0 and 1. The left most bit is the highest-order bit and represents the most significant bit (MSB) while the lowest-order bit is the least significant bit (LSB). Some useful definitions are:
  - Word is a binary number consisting of an arbitrary number of bits.
  - Nibble is a 4-bit word (one hexadecimal digit) 16 values.
  - Byte is an 8-bit word 256 values.

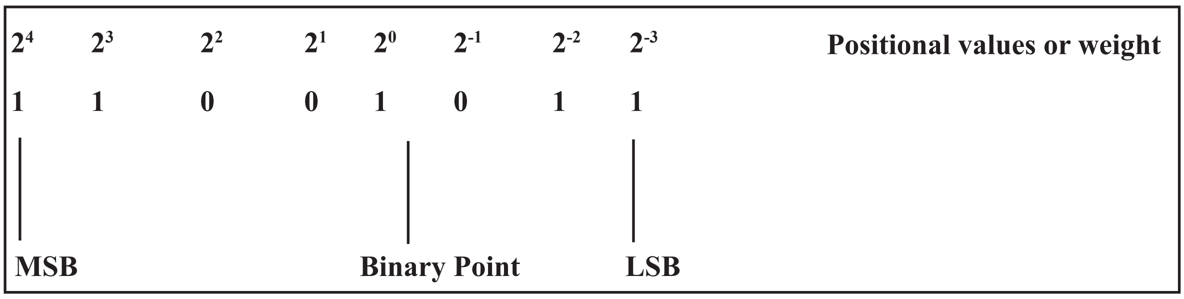


Fig. 2.1 Positional value (weight) of each bit
(Source: http://www.egyankosh.ac.in/bitstream/123456789/10991/1/Unit-10.pdf)

Any number can be expressed in binary form in the usual way. Table 2.1 shows expression of binary numbers.

2^3	2^2	2^1	2^0	**Binary Number**	**Decimal Number**
0	0	0	0	0000	0
0	0	0	1	0001	1
0	0	1	0	0010	2
0	0	1	1	0011	3
0	1	0	0	0100	4
0	1	0	1	0101	5
0	1	1	0	0110	6
0	1	1	1	0111	7
1	0	0	0	1000	8
1	0	0	1	1001	9
1	0	1	0	1010	10
1	0	1	1	1011	11
1	1	0	0	1100	12
1	1	0	1	1101	13
1	1	1	0	1110	14
1	1	1	1	1111	15

Table 2.1 Counting binary numbers
(Source: http://www.egyankosh.ac.in/bitstream/123456789/10991/1/Unit-10.pdf)

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

Fig. 2.2 1's complement
(Source: http://www.egyankosh.ac.in/bitstream/123456789/10991/1/Unit-10.pdf)

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.

Signed numbers
In a signed number, the left most bit is the so called sign bit: 0=positive number 1=negative number.

Sign-value notation
In this notation, the left-most bit is the sign bit and the others are used to represent the absolute value notation.

1’s complement
In this notation, the positive numbers have the same representation as the sign-value notation, and the negative numbers are obtained by taking the 1’s complement of the positive correspondents.

2’s complement
The positive numbers have the same representation as the sign-value notation, and the negative numbers are obtained by taking the 1’s complement of the positive correspondents.

Positive	All	00011001 (+25)
Negative	Sign-value	10011001
1’s complement		11100110
2’s complement		11100111

2.2.1 Binary to Decimal Conversion

Binary number can be converted into its decimal equivalent, by simply adding the weights of various positions in the binary number which have bit 1.

Example 1:

Find the decimal equivalent of the binary number $(11111)_2$
The equivalent decimal number is
$=1X2^4+1X2^3+1X2^2+1X2^1+1X2^0$
=16+8+4+2+1
$= (31)_{10}$

To differentiate between numbers represented in different number systems, either the corresponding number system may be specified along with the number or a small subscript at the end of the number may be added signifying the number system. Example $(1000)_2$ represents a binary number and is not one thousand.

Example 2:

Consider the conversion of $(100011.101)_2$
1 0 0 0 1 1.1 0 1
= 25+0+0+0+2^1+2^0+2-1+0+2-3
=32+2+1+0.5+0.125
$=(35.625)_{10}$
Consider the following examples.
1111.00 = 15
11110.0 = 30
111100.0 = 60
From these examples, it is clear that if the binary point is shifted towards right side, then the value of the number is doubled.
Now consider the following examples.
111.100 = 7.5
11.1 100 = 3.75
1.1 1100 = 1.875
From these examples it is clear that if the binary point is shifted towards the left side, then the value of the number is halved.

2.3 Decimal to Binary Conversion

A decimal number is converted into its binary equivalent by its repeated divisions by 2. The division is continued till we get a quotient of 0. Then all the remainders are arranged sequentially with first remainder taking the position of LSB and the last one taking the position of MSB. Consider the conversion of 27 into its binary equivalent as follows.

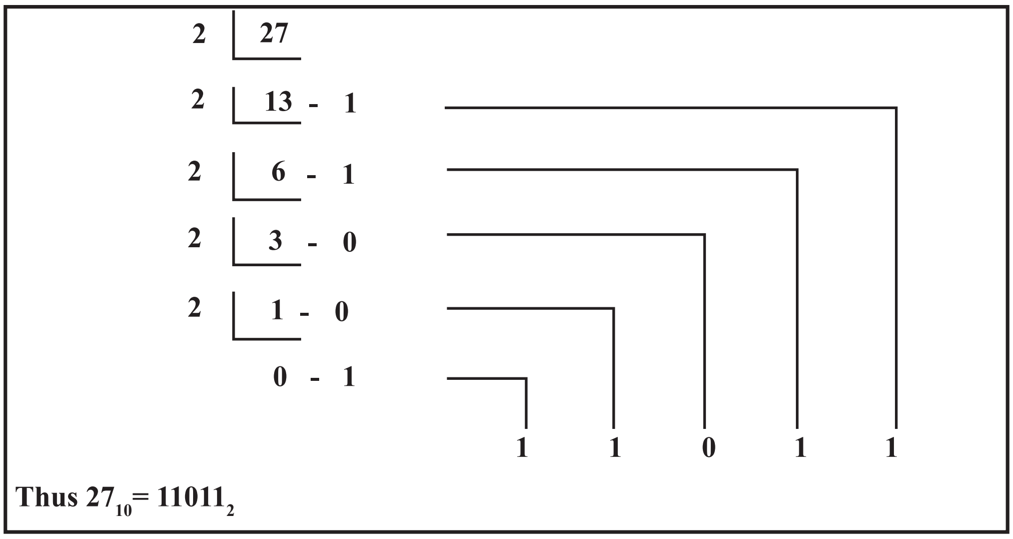


Fig. 2.3 (a) Decimal to binary conversion

(Source: http://www.egyankosh.ac.in/bitstream/123456789/10991/1/Unit-10.pdf)

If the number also has some figures on the right of the decimal point, then this part of the number is to be treated separately. Multiply this part repeatedly by two. After first multiplication by 2, either 1 or 0 will appear on the left of the decimal point. Keep this 1 or 0 separately and do not multiply it by 2 subsequently. This should be followed for every multiplication. Continue multiplication by 2 till you get all 0s after the decimal point or up to the level of the accuracy desired. This will be clear from the following example. Consider the conversion of 27.62510 into its binary equivalent. We have already converted 27 into its binary equivalent which is $(11011)_2$. Now for the conversion of 0.625, multiply it by 2 repeatedly as follows:

Thus, 27.62510 = 11011.101

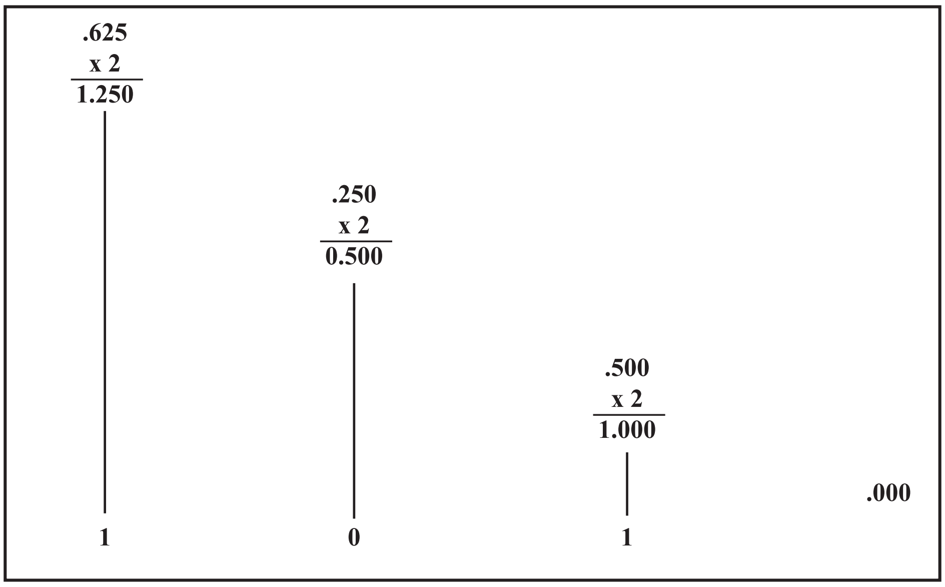


Fig. 2.3 (b) Decimal to binary conversion

(Source: http://www.egyankosh.ac.in/bitstream/123456789/10991/1/Unit-10.pdf)

Let us try another example, conversion of $(58.0725)_{10}$ into binary. Split this number in two parts, i.e., 58 and .0725 and convert them into binary separately as described above.

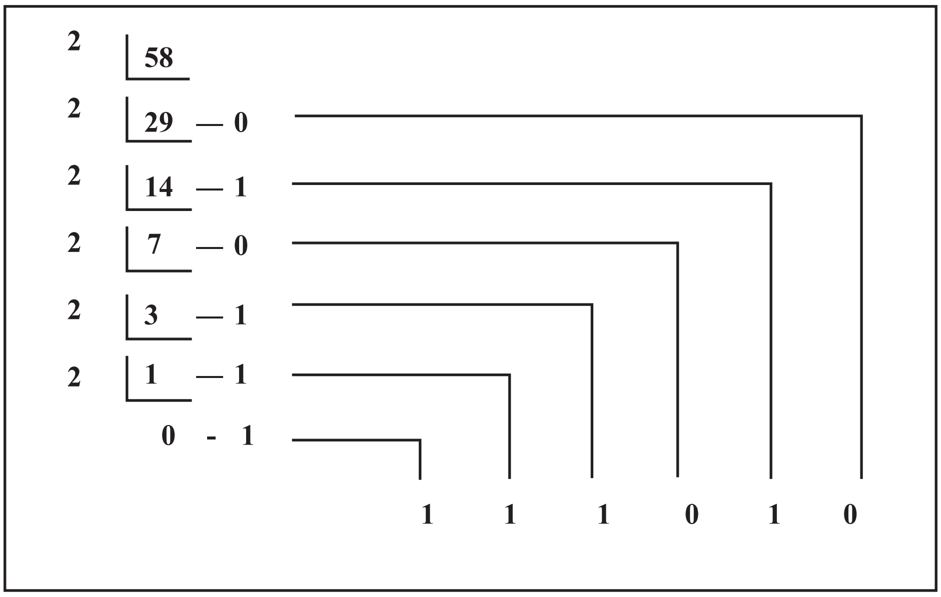


Fig. 2.4 (a) Decimal to binary conversion
(Source: http://www.egyankosh.ac.in/bitstream/123456789/10991/1/Unit-10.pdf)

Now let's look at the conversion of 0.725

Thus, (58.0725)10 = 111010.00010

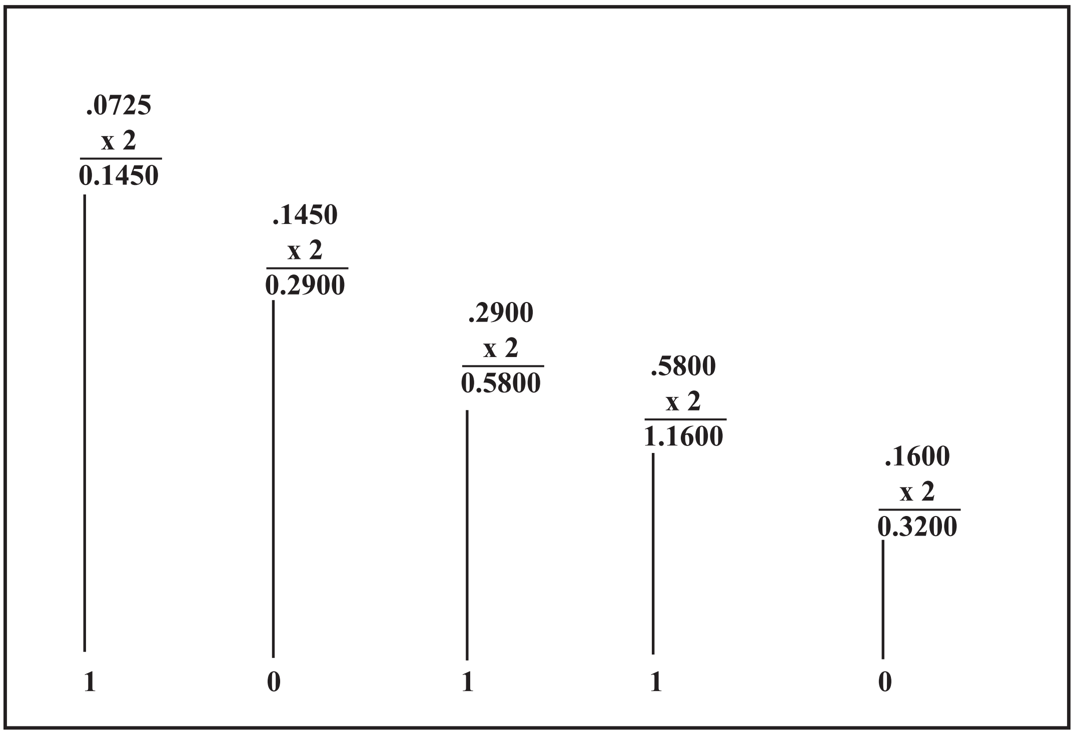


Fig. 2.4 (b) Decimal to binary conversion
(Source: http://www.egyankosh.ac.in/bitstream/123456789/10991/1/Unit-10.pdf)

Representing numbers in binary is very tedious, since binary numbers often consist of a large chain of 0's and 1's. Convenient shorthand forms for representing the binary numbers are developed such as octal system and hexadecimal system. With these number systems long strings of 0's and 1's can be reduced to a manageable form. The section below gives an overview of these systems.

2.4 Octal Number System

The octal number system has base-8 that is, there arc 8 digits in this system. These digits are 0, 1, 2, 3, 4, 5, 6, and 7. The weight of each octal digit is some power of 8 depending upon the position of the digit. Octal numbers showing positional values (weights) of each digit are as follows:

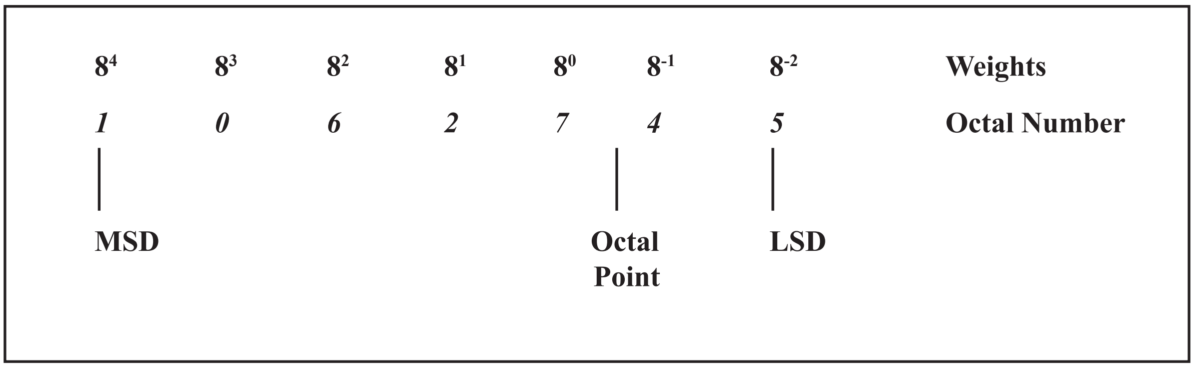

8^4 8^3 8^2 8^1 8^0 8^{-1} 8^{-2} Weights
1 0 6 2 7 4 5 Octal Number
MSD
Octal Point
LSD

Fig. 2.5 Octal number system

(Source: http://www.egyankosh.ac.in/bitstream/123456789/10991/1/Unit-10.pdf)

Octal number does not include the decimal digits 8 and 9. If any number includes decimal digits 8 and 9, then the number can not be an octal number.

2.4.1 Octal to Decimal Conversion

As has been done in case of binary numbers, an octal number can be converted into its decimal equivalent by multiplying the octal digit by its positional value. For example,
let us convert 36.48 into decimal number.
$36.48 = 3 \times 8^1 + 6 \times 8^0 + 4 \times 8^{-1}$
$= 24 + 6 + 0.5$
$= (30.5)_{10}$

2.4.2 Decimal to Octal Conversion

A decimal number can be converted by repeated division by 8 into equivalent octal number. This method is similar to that adopted in decimal to binary conversion. If the decimal number has some digits on the right of the decimal point, then this part of the number is converted into its octal equivalent by repeatedly multiplying it by 8. The process is same as has been followed in binary number system. Consider the conversion of 126.3810 into its decimal equivalent. Split it into two parts, that is 126 and .38

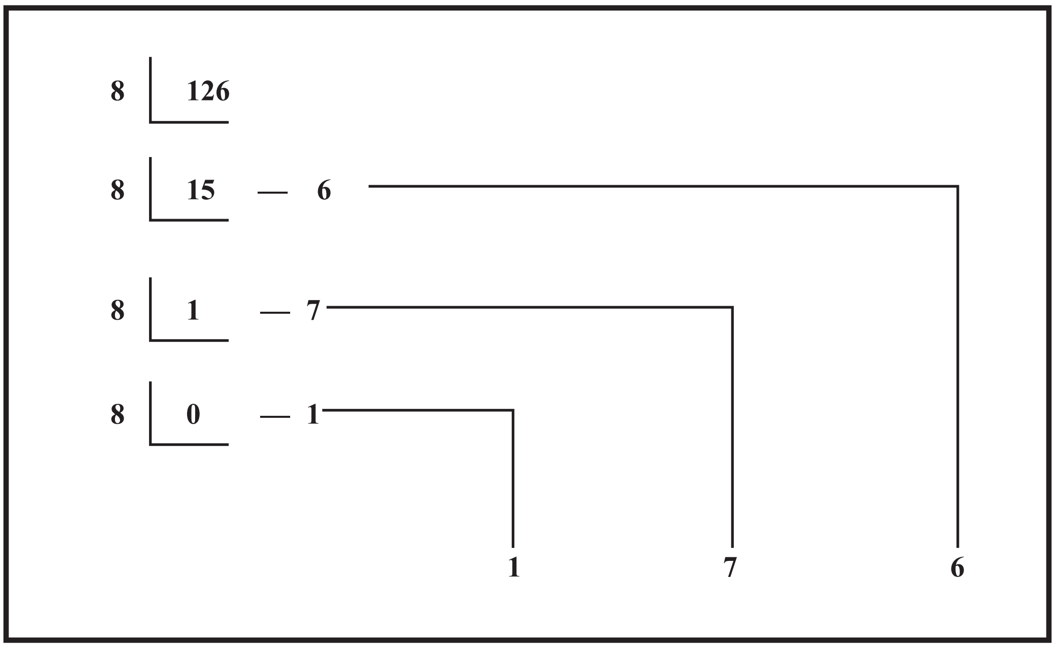


Fig. 2.6 (a) Decimal to octal conversion
(Source: http://www.egyankosh.ac.in/bitstream/123456789/10991/1/Unit-10.pdf)

The conversion of .38 is as follows

Thus, $(126.38)_{10}$ = 176.3024

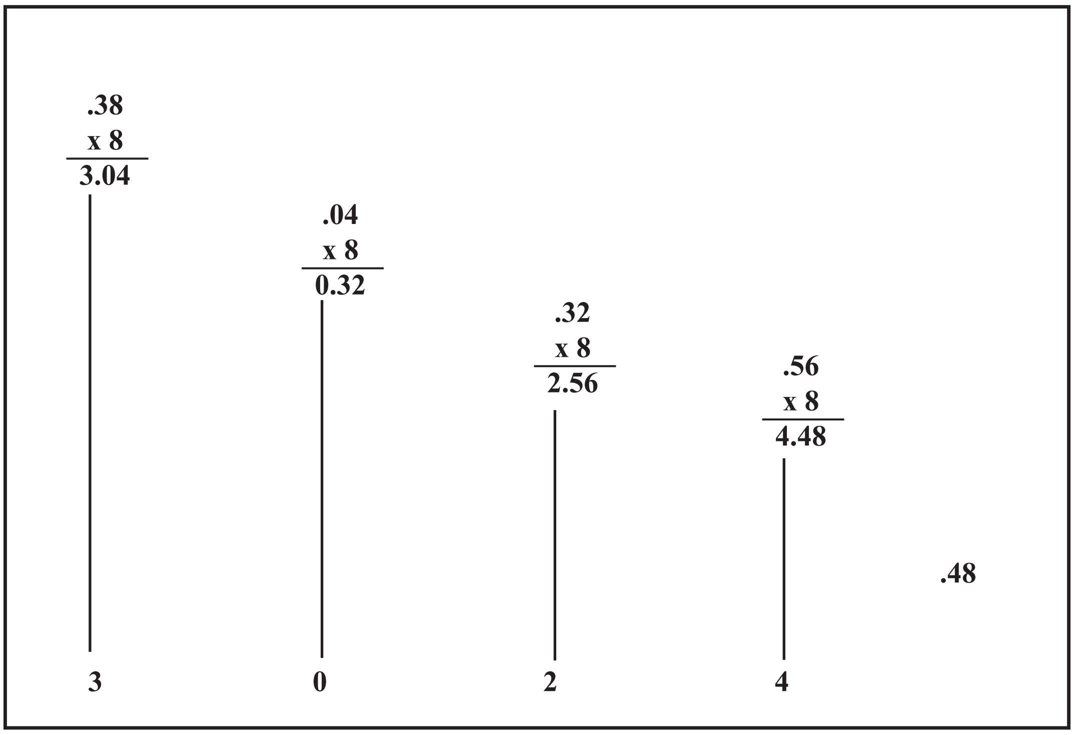


Fig. 2.6 (b) Decimal to octal conversion
(Source: http://www.egyankosh.ac.in/bitstream/123456789/10991/1/Unit-10.pdf)

2.4.3 Octal to Binary Conversion

In the octal number system the highest octal digit, i.e., 7 can be expressed as a 3-bit binary number. Therefore, all the octal digits have to be represented by n 3-bit binary number. The binary equivalent of each octal digit is shown in Table 1.2. The main advantage of the octal number system is the easiness with which any octal number can be converted into its binary equivalent.

Octal digit	3-bit binary equivalent
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

Table 2.2 Binary equivalent of octal digit
(Source: http://www.egyankosh.ac.in/bitstream/123456789/10991/1/Unit-10.pdf)

Using this conversion of octal digit into 3-bit binary number, any octal number can be converted into its binary equivalent by simply replacing each octal digit by a 3-bit binary number. For example, conversion of 567, into its binary equivalent is:
567 = 101110111

2.4.4 Binary to Octal Conversion

A binary number can be converted into its octal equivalent by first making groups of 3-bits starting from the LSB side. If the MSB side does not have 3 bits, then add 0s to make the last group of 3 bits. Then by replacing each group of 3 bits by its octal equivalent, a binary number can be converted into its binary equivalent. For example, consider the conversion of 1100011001, into its octal equivalent as follows:
= 001 100 01 1 001 [As the MSB side does not have 3 bits, we have added two 0's to make the last group of 3 bits]
= 1 4 3 1

Thus, $(1100011001)_2 = (1431)_8$

2.5 Hexadecimal Number System

The hexadecimal number system has base 16 that is it has 16 digits (Hexadecimal means'16'). These digits are 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F. The digits A, B, C, D, E and F have equivalent decimal values 10, 11, 12, 13, 14, and 15 respectively. Each Hex (Hexadecimal is popularly known as hex) digit in a hex number has a positional value that is some power of 16 depending upon its position in the number.

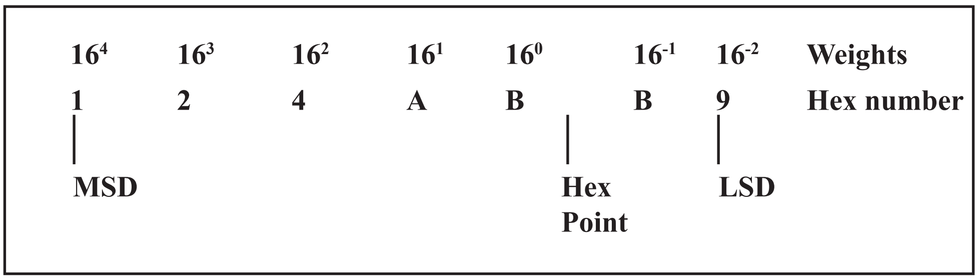

16^4 16^3 16^2 16^1 16^0 16^{-1} 16^{-2} Weights
1 2 4 A B B 9 Hex number
MSD
Hex Point
LSD

Fig. 2.7 Hexadecimal number showing positional values (weight) of digits
(Source: http://www.egyankosh.ac.in/bitstream/123456789/10991/1/Unit-10.pdf)

Relationship of hex digits with decimal and binary numbers is given in Table 1.3. Note that to represent the largest hex digit we require four binary bits. Therefore, the binary equivalent of all the hex digits has to be written in four bit numbers.

Hex digit	Decimal equivalent	4-bit Binary equivalent
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
A	10	1010
B	11	1011
C	12	1100
D	13	1101
E	14	1110
F	15	1111

Table 2.3 Binary and decimal equivalent of each hex digit
(Source: http://www.egyankosh.ac.in/bitstream/123456789/10991/1/Unit-10.pdf)

2.5.1 Hex to Decimal Conversion

Hex to decimal conversion is done in the same way as in the cases of binary and octal to decimal conversions. A hex number is converted into its equivalent decimal number by summing the products of the weights of each digit and their values. This is clear from the example of conversion of 514.AF16 into its decimal equivalent.

$514.\,AF16 = 5 \text{ X } 16^2 + 1 \text{ X } 16^1 + 4 \text{ X } 16^0 + 10 \text{ X } 16^{-1} + 15 \text{ X } 16^{-2}$
$= 1280+16+4+0.625+0.0586$
$= (1300.6836)_{10}$

2.5.2 Decimal to Hex Conversion

A decimal number is converted into hex number in the same way as a decimal number is converted into its equivalent binary and octal numbers. The part of the number on the left of the decimal point is to be divided repeatedly by 16 and the part an the right of the decimal point is to be repeatedly multiplied by 16. This will be clear from the examples of conversion of $(579.26)_{10}$ into hex equivalent. Split the number into two parts, 579 and .26.

Thus, $(579)_{10} = (2443)_{16}$

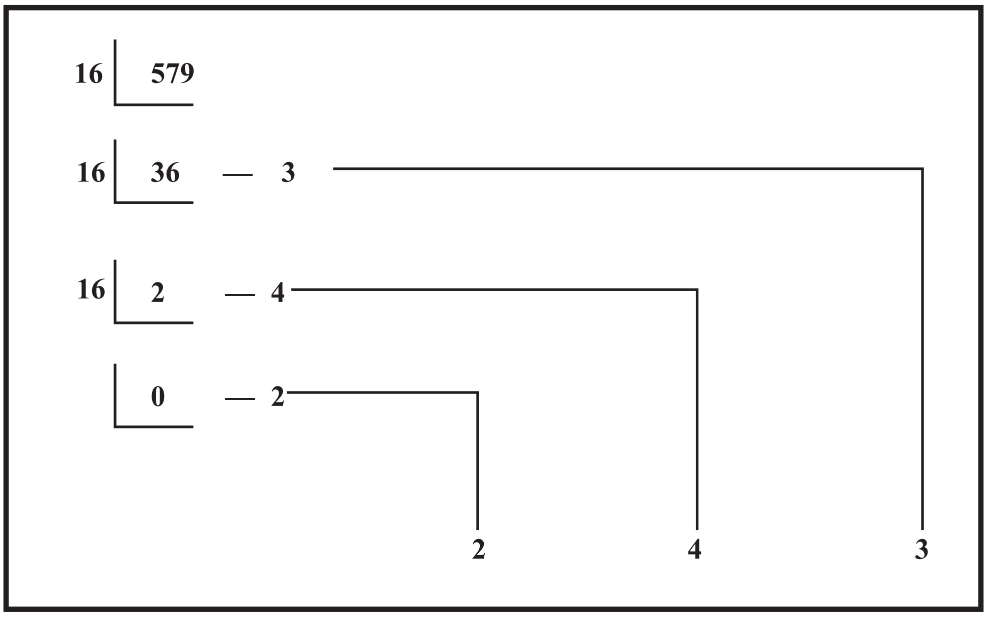


Fig. 2.8 (a) Decimal to hex conversion
(Source: http://www.egyankosh.ac.in/bitstream/123456789/10991/1/Unit-10.pdf)

Now .26 is converted into hex number as follows:

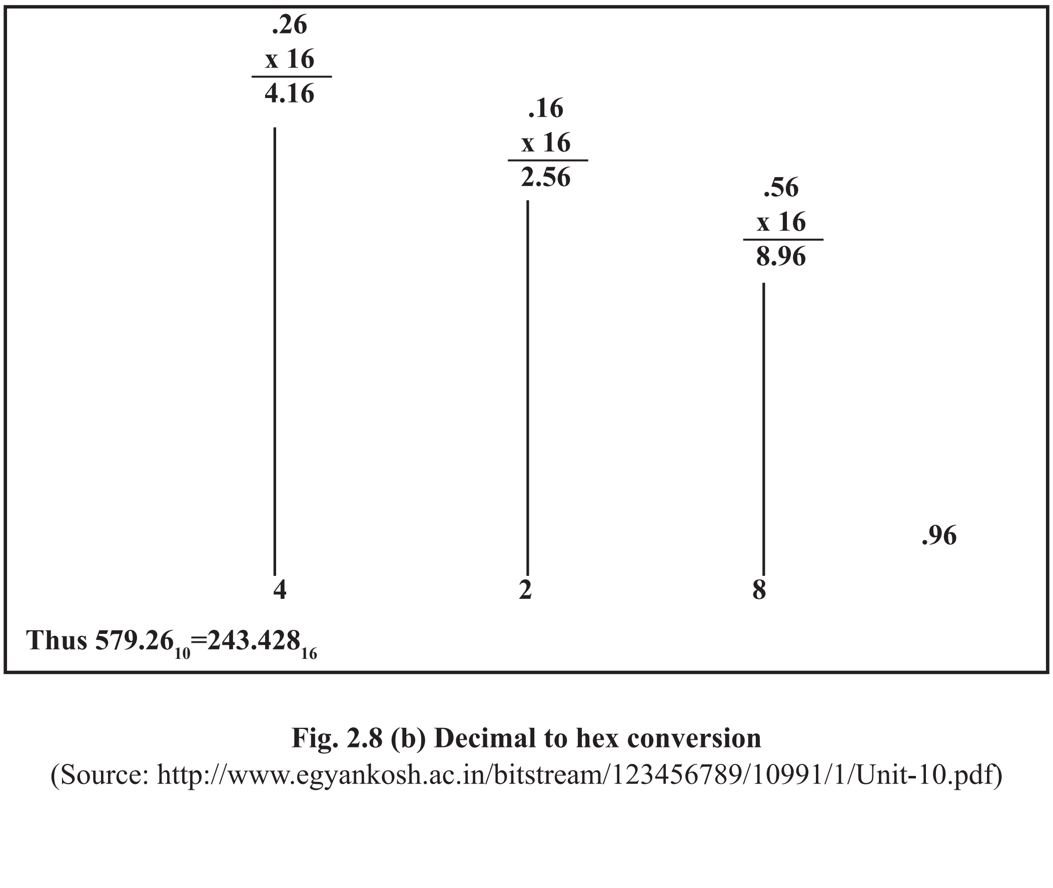


Fig. 2.8 (b) Decimal to hex conversion
(Source: http://www.egyankosh.ac.in/bitstream/123456789/10991/1/Unit-10.pdf)

2.5.3 Hex to Binary Conversion

As in octal number system, a hex number is converted into its binary equivalent by replacing each hex digit by its equivalent 4-bit binary number. This is clear from the following example:

```
(BA6)16 = B          A          6
        = 1011       1010       010
        = (101110100110)2
```

2.5.4 Binary to Hex Conversion

By a process that is reverse of the process described in the above section, a binary number can be converted in to its hex equivalent. Starting from the LSB side, group the binary number bits into groups of lour bits. If towards the MSB side, the numbers of bits is less than four, then add zeros on the left of the MSB so that the group of four is complete. Replace each group by its equivalent hex digit. This is clear from the following example:

```
(1001101110)2 = 0010   0110   1110
              = 2      6      E
              = (26E)16
```

2.5.5 Hex to Octal Conversion

Each digit of the hex number is first converted into its equivalent four bit binary number. Then the bits of the equivalent binary number are grouped into groups of three bits. Then each group is replaced by its equivalent octal digit to get the octal number.
For example:

```
(5AF)16 = 0101          1010          1111
        = 010110101111
        =010    110    101    111
        = 2     6      5      7
        =(2567)8
```

2.5.6 Octal to Hex Conversion

For octal to hex conversion, just reverse the process described in the section above.
This is clear from the following example:

```
(5457)8 = 101    100    101    111
        = 1011   0010   1111
        = B      2      F
        = (B2F)16
```

This method can also be applied to hex to decimal and decimal to hex conversions. For example, consider the conversion of 3C16, into its decimal equivalent:

```
3C16   = 0011  1100
       = 1111002
```

Check the conversion.
3C16 = 3 X16^1 + C X16^0
= 3 x 16^1+ 12 x 16^0
= 48 + 12
= $(60)_{10}$
1111002 = $2^5+2^4+2^3+2^2$
= 32+16+8+4
= $(60)_{10}$
Thus, 3C16 = $(111100)_2$ = 6010

2.6 Codes

We had an overview of binary, octal and hexadecimal number system. For any number system with n base B and digits N_0 (LSB), N_1 N_2...... N_{10} (M SB), the decimal equivalent N_{10} is given by

$$N_{10} = N_m \times B^m + ... N_3 \times B^3 + N_2 \times B^2 + N_1 \times B^1 + N_0 B^0$$

When numbers, letters or words are represented by a specific group of symbols, it is said that the number, letter or word is being encoded. The group of symbols is called as the code.
Few codes will be discussed in the following sections.

2.6.1 BCD Code

In BCD (BCD stands Binary coded decimal) code, each digit of a decimal number is converted in to its binary equivalent. The largest decimal digit is 9; therefore the largest binary equivalent is 1001. This is illustrated as follows

951 10 = 1001 0101 0001
= $(100101010001)_{BCD}$

2.6.2 ASCII Code

The word ASCII is run acronym of American Standard Code for Information Interchange. This is the alphanumeric code most widely used in computers. The alphanumeric code is one that represents alphabets, numerical numbers, punctuation marks and other special characters recognised by a computer. The ASCII code is a 7-bit code representing 26 English alphabets, 0 through 9 digits, punctuation marks, etc. A 7-bit code has 27 = 128 possible code groups which arc quite sufficient.

2.6.3 Code Gray

Gray Code is a form of binary that uses a different method of incrementing from one number to the next. With Gray Code, only one bit changes state from one position to another. This feature allows a system designer to perform some error checking (i.e., if more than one bit changes, the data must be incorrect).

Decimal	Binary	Gray	Decimal	Binary	Gray
0	0000	0000	8	1000	1100
1	0001	0001	9	1001	1101
2	0010	0011	10	1010	1111
3	0011	0010	11	1011	1110
4	0100	0110	12	1100	1010
5	0101	0111	13	1101	1011
6	0110	0101	14	1110	1001
7	0111	0100	15	1111	1000

Table 2.4 Gray code
(Source: http://dpnc.unige.ch/tp/elect/doc/07-Digital.pdf)

2.6.4 Excess 3

- Excess-3 binary-coded decimal (XS-3), also called biased representation or Excess-N, is a numeral system used on some older computers that uses a pre-specified number N as a biasing value. It is a way to represent values with a balanced number of positive and negative numbers. In XS-3, numbers are represented as decimal digits, and each digit is represented by four bits as the BCD value plus 3 (the "excess" amount)

- The smallest binary number represents the smallest value. (i.e. 0 − Excess Value)
- The greatest binary number represents the largest value. (i.e. 2_{N+1} − Excess Value − 1)
- The primary advantage of XS-3 coding over BCD coding is that a decimal number can be 9’s complemented (for subtraction) as easily as a binary number can be 1's complemented; just invert all bits. In addition, when the sum of two XS-3 digits is greater than 9, the carry bit of a four bit adder will be set high. This works because, when adding two numbers that are greater or equal to zero, an "excess" value of six results in the sum. Since a four bit integer can only hold values 0 to 15, an excess of six means that any sum over nine will overflow.

2.7 Binary Arithmetic

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.

We are considering binary arithmetic in the context of building digital logic circuits to perform arithmetic. Not only do we have to deal with the fact of finite precision arithmetic, we must consider the complexity of the digital logic. When there is more than one way of performing an operation we choose the method which results in the simplest circuit.

2.7.1 Addition

Addition of binary numbers can be carried out in a similar way by the column method. But before this, view four simple cases. In the decimal number system, 3 + 6 = 9 symbolizes the combination of 3 with 6 to get a total of 9.

View the four simple cases.

- Case 1: When nothing is combined with nothing, we get nothing. The binary representation of this is 0 + 0 = 0.
- Case 2: When nothing is combined with1, we get1. Using binary numbers to denote this gives 0 + 1 = 1.
- Case 3: Combining.1 with nothing gives 1. The binary equivalent of this is 1 + 0 = 1.
- Case 4: When we combine 1 with 1, the result is 2. Using binary numbers, we symbolize 1 + 1 = 10.

The last result is sometimes confusing because of our long time association with decimal numbers. But it is correct and makes sense because we are using binary numbers. Binary number 10 stands for'1','0' and not for 10 (ten). To summarize our results for binary addition,

  - 0+0 = 0
  - 0+1 = 1
  - 1+0 = 1
  - 1+1=10

- One can also express the rules of binary addition with a truth table. This is important because there are techniques for designing electronic circuits that compute functions expressed by truth tables. The fact that we can express the rules of binary addition as a truth table implies that we can design a circuit which will perform addition on binary numbers, and that turns out to be the case.
- We only need to write the rules for one column of bits; we start at the right and apply the rules to each column in succession until the final sum is formed. Call the bits of the addend and augend A and B, and the carry in from the previous column Ci. Call the sum S and the carry out Co.
- The truth table for one-bit binary addition looks like this:

A	B	Ci	S	Co
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

Table 2.5 One-bit binary addition

This says if all three input bits are zero, both S and Co will be zero. If any one of the bits is one and the other two are zero, S will be one and Co will be zero. If two bits are 1's, S will be zero and Ci will be one. Only if all three bits are 1's, both S and Co will be 1's.

2.7.1.1 Addition of Signed Numbers

Binary addition of 2's complement signed numbers can be performed using the same rules given above for unsigned addition. If there is a carry out of the sign bit, it is ignored. It is possible for the result of an addition to be too large to fit in the available space. The answer will be truncated, and will be incorrect. This is the overflow condition discussed above. There are two rules for determining whether overflow has occurred:

- If two numbers of opposite signs are added, overflow cannot occur.
- If two numbers of the same sign are added, overflow has occurred if and only if the result is of the opposite sign.

2.7.2 Subtraction

Addition has the property of being commutative, that is, a+b = b+a. This is not true of subtraction. 5 – 3 is not the same as 3 – 5. For this reason, we must be careful of the order of the operands when subtracting. We call the first operand, the number which is being diminished, the minuend; the second operand, the amount to be subtracted from the minuend, is the subtrahend. The result is called the difference.

$$\begin{array}{l} \ \ 51 \text{ minuend} \\ \underline{-22 \text{ subtrahend}} \\ \ \ 29 \text{ difference} \end{array}$$

- It is possible to perform binary subtraction using the same process we use for decimal subtraction, namely subtracting individual digits and borrowing from the left.
- This process quickly becomes cumbersome as you borrow across successive zeroes in the minuend. Further, it doesn't lend itself well to automation.
- Jacobowitz describes the "carry" method of subtraction which some of you may have learned in elementary school, where a one borrowed in the minuend is "paid back" by adding to the subtrahend digit to the left. This means that one need look no more than one column to the left when subtracting.
- Subtraction can thus be performed a column at a time with a carry to the left, analogous to addition. This is a process which can be automated, but we are left with difficulties when the subtrahend is larger than the minuend or when either operand is, signed.
- Since we can form the complement of a binary number easily and can add signed numbers easily, the obvious answer to the problem of subtraction is to take the 2's complement of the subtrahend, then add it to the minuend. That is 51–22 = 51+ (–22).
- Not only does this approach remove many of the complications of subtraction by the usual method, but it also means special circuits to perform subtraction need not be built All that is needed is a circuit which can form the bitwise complement of a number and an adder.

2.7.3 Multiplication

A simplistic way to perform multiplication is by repeated addition. In the example below, we could add 42 to the product register 27 times. In fact, some early computers performed multiplication this way. However, one of our goals is speed, and we can do much better using the familiar methods we have learned for multiplying decimal numbers. Recall that the multiplicand is multiplied by each digit of the multiplier to form a partial product, and then the partial products are added to form the total product. Each partial product is shifted left to align on the right with its multiplier digit.

```
  42  multiplicand
x 27  multiplier
 294  first partial product     (42 X 7)
  84  second partial product (42 X 2)
1134 total product
```

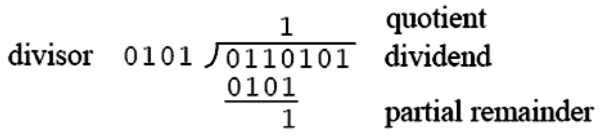
Binary multiplication of unsigned (or positive 2's complement) numbers works exactly the same way, but is even easier because the digits of the multiplier are all either zero or one. That means the partial products are either zero or a copy of the multiplicand, shifted left appropriately. Consider the following binary multiplication:

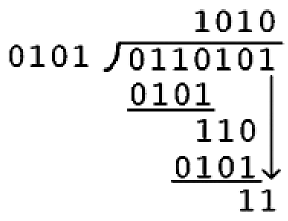
```
   0111        multiplicand
x 0101        multiplier
   0111        first partial product      (0111 X 1)
   0000        second partial product   (0111 X 0)
   0111        third partial product      (0111 X 1)
   0000        fourth partial products   (0111 X 0)
0100011       total product
```

Notice that no true multiplication is necessary in forming the partial products. The fundamental operations required are shifting and addition. This means we can multiply unsigned or positive integers using only shifters and adders.

2.7.4 Division

As with the other arithmetic operations, division is based on the paper-and-pencil approach we learned for decimal arithmetic. We will show an algorithm for unsigned long division that is essentially similar to the decimal algorithm we learned in grade school. Let us divide 0110101 (5310) by 0101 (510). Beginning at the left of the dividend, we move to the right one digit at a time until we have identified a portion of the dividend which is greater than or equal to the divisor. At this point, a one is placed in the quotient; all digits of the quotient to the left are assumed to be zero. The divisor is copied below the partial dividend and subtracted to produce a partial remainder as shown below.





Now digits from the dividend are "brought down" into the partial remainder until the partial remainder is again greater than or equal to the divisor. Zeroes are placed in the quotient until the partial remainder is greater than or equal to the divisor, and then a one is placed in the quotient, as shown below.

The divisor is copied below the partial remainder and subtracted from it to form a new partial remainder. The process is repeated until all bits of the dividend have been used. The quotient is complete and the result of the last subtraction is the remainder.

This completes the division. The quotient is $(1010)_2$ (1010) and the remainder is $(11)_2$ (310), which is the expected result. This algorithm works only for unsigned numbers, but it is possible to extend it to 2’s complement numbers. As with the other algorithms, it can be implemented using only shifting, complementation, and addition. Digital computers can perform arithmetic operations using only binary numbers. And hence the above section of binary arithmetic is the basic step of digital electronics.