Unit 1 Number Systems

Structure:

1.1 Introduction

The number system is a collection of number to represent the quantifiable information. Most of the computations in the number systems are addition, subtraction, multiplication, division, etc. we usually perform all calculation using the decimal number system.

In decimal number system we have ten different digits or symbols i.e. 0, 1, 2, 3, 4, 5, 6, 7, 8 and 9. The combination of this number gives quantifiable values. To represent these values in the electronics world, we don't have the hardware, so there was need for the hardware which can represent the any one number system. The invention of transistor, have triggered the engineers to make use of the binary number system. The binary numbers system have only two values i.e., 0, 1. With the help of the transistor we can represent the two states (0, 1) of the number.

The invention of transistor made a remarkable change in the electronic world especially in computer science. The transistor acted as a switch, able to represent the binary number. In computer, the computations are not

carried similar to the normal day to day computation. Enormous numbers of algorithms are developed to carry out the computation operations.

There is a need of human computer interpretation requirement because human requires the value must be represented in the decimal systems whereas the computer computes internal value in binary systems. So there was a need of binary to decimal interpretation representation system and it is called Binary Coded Decimal (BCD). All decimal numbersare converted into the binary and then computation is performed and later is converted back into the decimal system.

The encoding of data in digital system is common. Since we know that the computer can only understand binary, there is a need for the conversion of alphabets and other special characters also into the binary. The different encoding and decoding has designed to suite the requirements. In this unit, we will discuss the different numbering system and their conversion between across the different numbering system.

Objectives:

By the end of Unit 1 the learners are able to:

- list and explain the different numbering systems.
- explain the negative number representation
- explain conversion techniques in number systems
- perform complements arithmetic

1.2 Decimal Number Systems

From the kindergarten to nowadays, we are exposed only to the decimal number systems.

A decimal number system has ten different digits or symbols. They are 0, 1, 2, 3, 4, 5, 6, 7, 8 and 9. The **base** (or radix) of a number system is the number of different symbols available to represent any digit within that system. For example, the base or radix of decimal number system is 10 as there are ten different digits.We represent the value with base i.e. 25_{10} here the number 25 represent the quantifiable value and the number 10 represent the base of the decimal numbering system.

Example 1.1: How do you represent decimal number 4567_{10}?

This number is represented as
$4 \times 10^3 + 5 \times 10^2 + 6 \times 10^1 + 7 \times 10^0$

The number position always starts from Zero. Here if you observe
7 is in 0^{th} position it give value 7
6 is in 1^{st} position it gives value 60
5 is in 2^{nd} position it gives value 500
4 is in 3^{rd} position it gives value 4000
Summing all together we get the value is 4567_{10}.

All decimal number is split into two parts. The value before dot and the value after dot. The value before dot is an absolute vale and the value after dot is a fractional value.

Example 1.2: How do you represent decimal number 4567.8901_{10}?

This number can represented as
$4 \times 10^3 + 5 \times 10^2 + 6 \times 10^1 + 7 \times 10^0 . 8 \times 10^{-1} + 9 \times 10^{-2} + 0 \times 10^{-3} + 1 \times 10^{-4}$

The number is position is given as

The value before dot
- 7 is in 0^{th} position it give value 7
- 6 is in 1^{st} position it gives value 60
- 5 is in 2^{nd} position it gives value 500
- 4 is in 3^{rd} position it gives value 4000

The value after dot
- 8 is in -1^{st} position it give value 0.8
- 9 is in -2^{nd} position it gives value 0.09
- 0 is in -3^{rd} position it gives value 0.000
- 1 is in -4^{th} position it gives value 0.0001

Summing all together we get the value 4567.8901_{10}

1.3 Binary Number Systems

The word bi means two.Binary number system has only two digits or symbols. They are 0 and 1. Hence the base of this number system is 2. Numbers represented in this system are commonly called binary numbers. The digit 0 or 1 is called binary digit (Bit).The use of binary number system came to existence because of the computer. The transistor can represent the value ‘0’ by *off* state and the value “1” by *on* state.

Example 1.3: How to represent the binary value “1010” in decimal format?

$$1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0$$

The decimal representation is

$$8 + 0 + 2 + 0 = 10_{10}$$

Example 1.4: How do you represent the binary value 1010.101 in decimal format?

$$1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0.\ \ 1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3}$$

The binary representation is

$$8 + 0 + 2 + 0\ .\ 0.5 + 0.0 + 0.125 = 10.625_{10}$$

1.4 Octal Number Systems

In octal number system, there are eight different digits or symbols. They are 0, 1, 2, 3, 4, 5, 6 and 7. Since there are eight digits, the base of the octal number system is 8.

Example 1.5: How do to represent the octal value “1760” in decimal format?

$$1 \times 8^3 + 7 \times 8^2 + 6 \times 8^1 + 0 \times 8^0$$

The decimal representation is

$$512 + 448 + 48 + 0 = 1008_{10}$$

Example 1.6: How do you represent the octal value 1760.154 in decimal format?

$$1 \times 8^3 + 7 \times 8^2 + 6 \times 8^1 + 0 \times 8^0.\ \ 1 \times 8^{-1} + 5 \times 8^{-2} + 4 \times 8^{-3}$$

The decimal representation is

$512 + 448 + 48 +0+0.125 + 0.078125 + 0.0078125 = 1008.2109375_{10}$

1.5 Hexadecimal Number Systems

The base of Hexadecimal number system is 16, means there are sixteen different symbols or digits. They are 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F.

Where A = $(10)_{10}$, B = $(11)_{10}$, C = $(12)_{10}$, D = $(13)_{10}$, E=$(14)_{10}$ and F = $(15)_{10}$

Example 1.7: How to represent the Hexadecimal value “1A3F” in decimal format?

$$1 \times 16^3 + A \times 16^2 + 3 \times 16^1 + F \times 16^0$$

$$1 \times 16^3 + 10 \times 16^2 + 3 \times 16^1 + 15 \times 16^0$$

The decimal representation is

$$4096 + 2560 + 48 + 15 = 6719_{10}$$

Example 1.8: How do you represent the hexadecimal value “1A3F.C0E” in decimal format?

$$1 \times 16^3 + A \times 16^2 + 3 \times 16^1 + F \times 16^0 . C \times 16^{-1} + 0 \times 16^{-2} + E \times 16^{-3}$$

The decimal representation is

$$4096 + 2560 + 48 + 15 + 0.75 + 0 + 0.0034179 = 6719.7534179_{10}$$

The table 1.1 says about the collection of the binary number as

Table 1.1: Acronyms

4 bits	Nibble
8 bits	Byte
16 bits	Half Word
32 bit	Word

The binary presented in two ways little endian and big endian method are as shown in the tables 1.2 and 1.3 respectively. Most of computer architecture has been designed based on either of this format to transfer the data across the system. The Least Significant Bit (LSB) where the value has least place holder value. The Most Significant Value (MSB), where the value has highest place holder value.

Table 1.2: Big Endian Method

1	0	1	0	1
MSB				LSB

Table 1.3: Little Endian Method

1	0	1	0	1
LSB				MSB

1.6 Negative Number Representation

The negative number has represented in many ways, they are

- Signed magnitude
- 1's complement
- 2's complement

1.6.1 Signed Magnitude Representation

The signed magnitude representation is one of the methods to represent the negative and positive number in binary system. It is a regular binary representation with addition of one more bit in the MSB side represent the number is negative or positive. The MSB "0" represent the positive value and MSB "1" represent the negative value.

The range for this representation is: $-2^{n-1}-1$ to $+2^{n-1}-1$

The drawbacks of this representation are: It has both positive and negative zero, and complex architecture required for computation.

1.6.2 1's Complement

The 1's complement is slightly different from the signed magnitude number. All binary bits are complemented and represented along with the sign bit. As shown in table 1.4

The range function for this representation is: $-2^{n-1}-1$ to $+2^{n-1}-1$

The drawbacks of this representation are: It has both positive and negative zero and complex architecture required for computation.

1.6.3 2's Complement

The 2's Complement is asymmetric system with additional bit is required to represent any numbers. The binary bits are complemented and added 1 with it.The range function for this representation is: -2^{n-1} to $+2^{n-1}-1$.The addition and subtraction can be performed using same hardware. The table 1.4 shows the comparison of negative number systems.

Table 1.4: Comparisons of Negative Number Systems

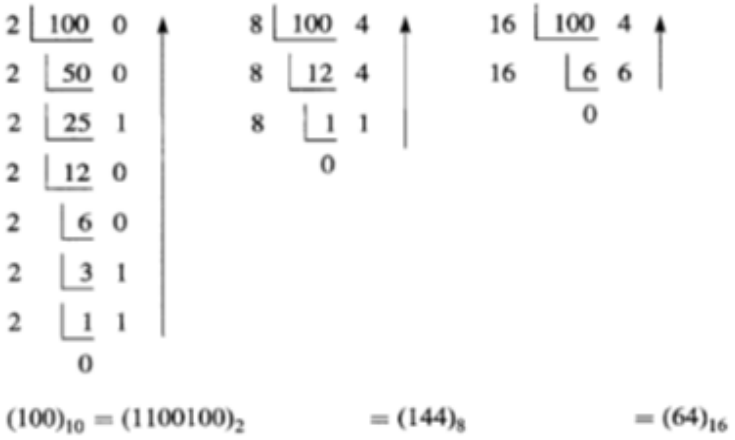
Decimal	Signed Magnitude	1's Complement	2's Complement
-8	–	–	1000
-7	1111	1000	1001
-6	1110	1001	1010
-5	1101	1010	1011
-4	1100	1011	1100
-3	1011	1100	1101
-2	1010	1101	1110
-1	1001	1110	1111
-0	1000	1111	–
+0	0000	0000	0000
+1	0001	0001	0001
+2	0010	0010	0010
+3	0011	0011	0011
+4	0100	0100	0100
+5	0101	0101	0101
+6	0110	0110	0110
+7	0111	0111	0111

1.7 Conversion from one number system to other number system

From the earlier example from 1.1 to 1.8 was showing how to convert from all number system to decimal. From here onwards, we will see how to convert the number from decimal to all other number systems.

The decimal number must be divided by the base of the other number systems to convert decimal number (i.e. Integer part) to other numbering system. For example, if we want to convert the decimal number to binary number system, we must divide the decimal number by 2 repeatedly till our quotient becomes zero as shown in example 1.9. The reminder moving from the bottom to top will be our binary number. It is similar for all other number system.

Example 1.9: How to convert 100_{10} into binary, octal and hexadecimal?

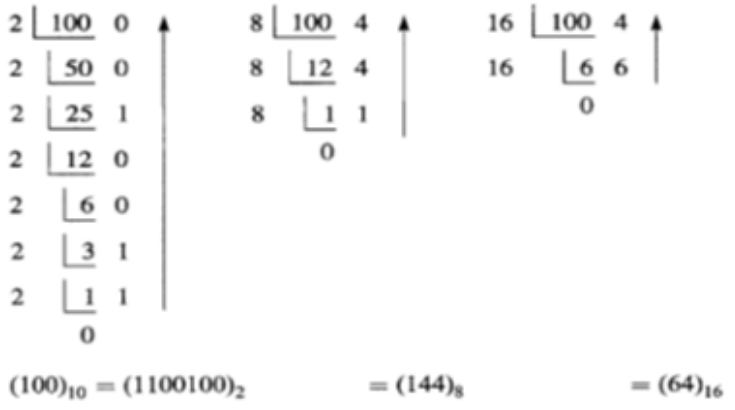


So far we have studied the decimal number to other number system without fraction. Now we will concentrate on the fractional conversion.

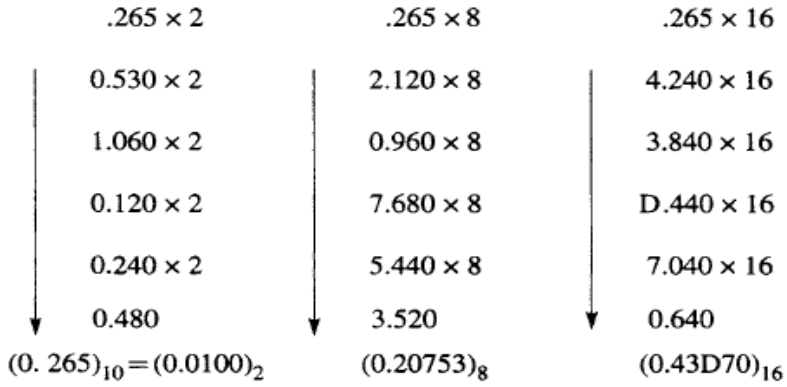
The fractional decimal number must be multiplied by the base of the other number system to convert it into other number systems as shown in example 1.10. For example decimal number is to be converted into the binary. Fractional part has to be multiplied by 2 repeatedly till we will make the fractional part zero. If the fractional part goes on and on, we can terminate in between.

Example 1.10: How to convert 100.265_{10} into binary, octal and hexadecimal?

Let us take the integer part first and convert it to other number systems.



Now for the fractional part, we follow the multiplication process as shown below:



Therefore,

$100.265_{10} = (1100100.0100)_2$; $100.265_{10} = (144.20753)_8$;

$100.265_{10} = (64.43D70)_{16}$

Now before we convert number from octal to binary and vice versa, one has to know the equivalent numbers for both the systems. The table 1.5 shows the octal numbers with its equivalent the binary numbers.

Table 1.5: Octal numbers with equivalent binary number formats

Octal	Binary
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

From the table 1.5, it is clear that any octal digit can be represented by a group of three binary digits (or Bits)

To covert any hexadecimal number to binary and vice versa, refer the table 1.6 which shows the binary equivalent for the hexadecimal numbers. The table 1.6 also gives the comparison of all the four number systems

Table 1.6: Comparison of decimal values with other number formats

Decimal	Binary	Octal	Hexadecimal
0	0000	0	0
1	0001	1	1
2	0010	2	2
3	0011	3	3
4	0100	4	4
5	0101	5	5
6	0110	6	6
7	0111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

From the table 1.6, it is clear that any hexadecimal digit can be represented by a group of four binary digits (or Bits)

Example 1.11: Convert the 6135_8 into binary value

In this example, we can replace all independent value into equivalent 3-bit binary value as shown below to get our answer.

6 1 3 5_8

110 001 011 101_2

Example 1.12: Convert $(6135.2478)_8$ into binary value

Similar with the example 1.11, here we will replace equivalent 3 bit binary value to represent the octal value.

6 1 3 5 . 2 4 7_8

110 001 011 101 . 010 100 111_2

Example 1.13: Convert $1A2C_{16}$ into Binary Value.
In this example, we can replace all independent value into 4-bit binary equivalent as shown below to get our answer.

1 A 2 C_{16}

0001 1010 0010 1100_2

Example 1.14: Convert $1A2C.3B4D_{16}$ into Binary Value.

In this example, we can replace all independent value into 4-bit binary equivalent as shown below to get our answer irrespective of its position.

1	A	2	C	.	3	B	4	D_{16}
0001	1010	0010	1100	.	0011	1011	0100	1101_2

1.8 Complements Arithmetic

This is a powerful yet simple technique which minimizes the hardware implementation of signed arithmetic operations in a digital machine. In practice, when using complement arithmetic, the process of subtraction becomes one of addition.

In any number system, complements are available. In the binary system they are

a) *2's complement* or *radix complement*
b) *1's complement* or *diminished radix complement.*

For the decimal number system, they are:

a) *10's complement* or *radix complement*
b) *9's complement* or *diminished radix complement.*

The tables 1.7 and 1.8 show the rules for binary addition and subtraction respectively.

Table 1.7: Addition

Value 1	Value 2	Carry	Sum
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Table 1.8: Subtraction

Value 1	Value 2	Borrow	Difference
0	0	0	0
0	1	1	1
1	0	0	1
1	1	0	0

It is worth noting that the use of the 1's complement in the binary system raises certain hardware implementation difficulties so that signed arithmetic processes are invariably performedusing 2's complement notation.

The major advantage of 2's complement over 1's complement is that 2's complement has only one value for zero. One's complement has a "positive" zero and a "negative" zero which is the disadvantage of 1's complement. Also note that there is no addition of end around carry in 2's complement method.

Example 1.15: Compute binary arithmetic operation for 9 – 5 using 2's Complement.

Note: Refer table 1.4 for the equivalent values.

Here keep the binary equivalent of 9 (1001) as it is and keep the 2's complement equivalent of 5 (1011) and add both vales to get the result 4, ignore carry generated MSB bit.

```
  1001
  1011
 10100
```

Discarding the carry (i.e. 1), we get the answer 0100 which is equal to 4 in decimal.

Example 1.16: Compute Binary arithmetic Operation for 9 – 5 using 1's Complement.

Note: Refer table 1.4 for the equivalent values.

Here keep the binary equivalent of 9 (1001) as it is and keep the 1's complement equivalent of 5(1010) and add both vales to get the partial result and add the carry value in the LSB to get the result 4.

```
  1001
  1010
 10011
   ↘ +1
 ------
  0100
```

Excess Notation

The excess notation is a means of representing both negative and positive numbers in a manner in which the order of the bit patterns is maintained.

The algorithm for computing the excess notation bit pattern is as follows:

1. Add the excess value 2^{N-1}, (where N is the number of bits used to represent the number) to the number.
2. Convert the resulting number into binary format.

The 2^{N-1} is often referred to as the *Magic Number* for computing the excess representation of the number (except that there is no magic in it). Table 1.9 presents all the numbers that can be represented using the excess-8 notation.

Table 1.9: Numbers using the Excess-8 representation

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

The number of bits used to represent a code in excess-8 is 4 bits. Also, the bit patterns are in sequence (the largest number that can be represented has the bit pattern 1111).

Example 1.17: Consider the following operation 7 – 2. Substituting the bit patterns from the table:

$$\begin{array}{r} 1111 \\ +\ 0110 \\ \hline 1\,0101 \end{array}$$

The result of the addition operation is the bit-pattern used for 5 in binary.

The excess notation representation however takes longer to compute than the 2's complement notation. The excess notation will however play an important part in computing floating-point representations.

Bias Notation

The excess notation is a special case of the biased notation. For instance, excess-8 is biased around 8 (i.e.0 has the bit pattern associated with decimal 8). Instead of using the magic number, any number (bias) can be used.

Note: This concept becomes important when we address the IEEE Single Precision Floating-Point standard.

Floating-Point Notation

The floating-point notation is used:

a. To represent integers that are larger than the maximum value that can be held by a bit-pattern (the maximum value that can be held by 8 bits is 255).
b. To represent real numbers.

Large Integers

Consider a really large number 1,234,567. The number requires seven places to represent the value. If the number of places available to represent the number is limited to say four places, certain digits have to be dropped. The selection of digits to be dropped is based on the value associated with the digit. In this case, we will drop the last three digits '567'.

The resulting number is:

1,234,000

The loss of '567' is a loss of ***precision*** but if the most significant digits were to be eliminated, says '123', and then the resulting number is **4,567,** which presents an even greater loss of precision.

Rules for determining significance (integers):

1. A nonzero digit is always significant
2. The digit '0' is significant if it lies between other significant digits
3. The digit '0' is ***never*** significant if it precedes all the nonzero digits

Self Assessment Questions

1. Decimal means base _____.
2. The decimal value of the bit pattern 11111111 is _____.

3. The range of values represented by an 8-bit binary number is _____.
4. The binary equivalent of 228 is _____.
5. The result of 7 – 2 using 1's complement notation is _____.
6. The bit pattern 1011 in 1's complement notation is _____.
7. The result of 7 – 2 using 2's complement notation is _____.
8. The bit pattern 1110 in 2's complement notation is _____.
9. The 2^{N-1} is often referred to as the _____ for computing the excess representation of the number.
10. The significant digits of 0012340 are _____.
11. The bit pattern 1101 Excess Number converted value is _____.

1.9 Summary

Let us recapitulate the important concepts discussed in this unit:

- In decimal number system, we have ten different digits or symbols i.e. 0, 1, 2, 3, 4, 5, 6, 7, 8 and 9.
- The base of octal number system is 8, means the number system has eight different digits 0, 1, 2, 3, 4, 5, 6 and 7.
- Any octal digit can be represented by a group of three binary digits (or Bits)
- In Hexadecimal number system, we have sixteen different digits or symbols i.e. 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 and A, B, C, D, E and F.
- The negative number can be represented in many ways. They are Signed magnitude, 1's complement and 2's complement.
- 2's complement is obtained by complementing the binary digits i.e. bits and then 1 is added to it.
- The major advantage of 2's complement over 1's complement is that 2's complement has only one value for zero.

1.10 Terminal Questions

1. Convert the following binary numbers to base 10:
   a. 10101101
   b. 110110.1
2. Convert the following octal numbers to base 10:
   a. 273
   b. 1021

3. Convert the following hexadecimal numbers to base 10:
   a. 145
   b. A2C1
4. Convert the following decimal numbers to base 2:
   a. 122
   b. 98
5. Convert the following decimal numbers to hexadecimal:
   a. 1145
   b. 2421
6. Perform the following binary arithmetic operations
   a. 101011 + 10111
   b. 1101 + 1110 + 1001
   c. 11101 –10110
   d. 1100.010 – 1000.111
7. Write the 8-bit signed magnitude, 2's complement and 1's complement form of the following decimal numbers:
   a. + 119
   b. – 77
   c. – 3
8. Perform the following arithmetic operations using 2's complement
   a. 9 – 8
   b. 10 - 6

1.11 Answers

Self Assessment Questions

1. 10
2. 255
3. 0 to 255.
4. 11100100
5. (0101)
6. $(0100)_2$
7. (0101)
8. 0010
9. Magic Number

10. 0012340
11. 13

Terminal Questions

1. Refer to Section 1.3 for conversion method
2. Refer to Section 1.4 forconversion method
3. Refer to Section 1.5 for conversion method
4. Refer to Section 1.7 for conversion method
5. Refer to Section 1.7 for conversion method
6. Refer to Section 1.7 for conversion method
7. Refer to Section 1.6 for method
8. Refer to Section 1.8 for 2’s complement subtraction method