CS 8351 – DIGITAL PRINCIPLES AND SYSTEM DESIGN

UNIT – I : DIGITAL FUNDAMENTALS

Number Systems

We all use numbers to communicate and perform several tasks in our daily lives. Our present day world is characterized by measurements and numbers associated with everything. In fact, many consider if we cannot express something in terms of numbers is not worth knowing. While this is an extreme view that is difficult to justify, there is no doubt that quantification and measurement, and consequently usage of numbers, are desirable whenever possible.

Manipulation of numbers is one of the early skills that the present day child is trained to acquire. The present day technology and the way of life require the usage of several number systems. Usage of decimal numbers starts very early in one's life. Therefore, when one is confronted with number systems other than decimal, some time during the high-school years, it calls for a fundamental change in one's framework of thinking. There have been two types of numbering systems in use through out the world.

One type is symbolic in nature. Most important example of this symbolic numbering system is the one based on Roman numerals

I = 1, V = 5, X = 10, L = 50, C = 100, D = 500 and M = 1000 IIMVII - 2007

While this system was in use for several centuries in Europe it is completely superseded by the weighted-position system based on Indian numerals. The

Roman number system is still used in some places like watches and release dates of movies.

The weighted-positional system based on the use of radix 10 is the most commonly used numbering system in most of the transactions and activities of today's world. However, the advent of computers and the convenience of using devices that have two well defined states brought the binary system, using the radix 2, into extensive use. The use of binary number system in the field of computers and electronics also lead to the use of octal (based on radix 8) and hex-decimal system (based on radix 16). The usage of binary numbers at various levels has become so essential that it is also necessary to have a good understanding of all the binary arithmetic operations. Here we explore the weighted-position number systems and conversion from one system to the other.

Weighted-Position Number System

In a weighted-position numbering system using Indian numerals the value associated with a digit is dependent on its position. The value of a number is weighted sum of its digits. Consider the decimal number 2357. It can be expressed as

$$2357 = 2 \times 103 + 3 \times 102 + 5 \times 101 + 7 \times 100$$

Each weight is a power of 10 corresponding to the digit's position. A decimal point allows negative as well as positive powers of 10 to be used;

$$526.47 = 5 \times 102 + 2 \times 101 + 6 \times 100 + 4 \times 10\text{-}1 + 7 \times 10\text{-}2$$

Here, 10 is called the base or radix of the number system.

In a general positional number system, the radix may be any integer r > 2, and a digit position i has weight r i . The general form of a number in such a system is

dp-1 dp-2, ... d1, d0 . d-1d-2 d-n

where there are p digits to the left of the point (called radix point) and n digits to the right of the point. The value of the number is the sum of each digit multiplied by the corresponding power of the radix.

Except for possible leading and trailing zeros, the representation of a number in positional system is unique (00256.230 is the same as 256.23). Obviously the values di's can take are limited by the radix value. For example a number like (356)5, where the suffix 5 represents the radix will be incorrect, as there can not be a digit like 5 or 6 in a weighted position number system with radix 5.

If the radix point is not shown in the number, then it is assumed to be located near the last right digit to its immediate right. The symbol used for the radix point is a point (.). However, a comma is used in some countries. For example 7,6 is used, instead of 7.6, to represent a number having seven as its integer component and six as its fractional. As much of the present day electronic hardware is dependent on devices that work reliably in two well defined states, a numbering system using 2 as its radix has become necessary and popular. With the radix value of 2, the binary number system will have only two numerals, namely 0 and 1. Consider the number

$(N)_2 = (11100110)_2$.

It is an eight digit binary number. The binary digits are also known as bits. Consequently the above number would be referred to as an 8-bit number. Its decimal value is given by

$(N)_2 = 1 \times 27 + 1 \times 26 + 1 \times 25 + 0 \times 24 + 0 \times 23 + 1 \times 22 + 1 \times 21 + 0 \times 20$

$= 128 + 64 + 32 + 0 + 0 + 4 + 2 + 0$

$= (230)_{10}$

Consider a binary fractional number $(N)_2 = 101.101$.

Its decimal value is given by

$(N)_2 = 1 \times 22 + 0 \times 21 + 1 \times 20 + 1 \times 2\text{-}1 + 0 \times 2\text{-}2 + 1 \times 2\text{-}3$

$= 4 + 0 + 1 + 1\ 2 + 0 + 1\ 8$

$= 5 + 0.5 + 0.125$

$= (5.625)_{10}$

From here on we consider any number without its radix specifically mentioned, as a decimal number. With the radix value of 2, the binary number system requires very long strings of 1s and 0s to represent a given number. Some of the problems associated with handling large strings of binary digits may be eased by grouping them into three digits or four digits.

We can use the following groupings. Octal (radix 8 to group three binary digits) Hexadecimal (radix 16 to group four binary digits) In the octal number

system the digits will have one of the following eight values 0, 1, 2, 3, 4, 5, 6 and 7. In the hexadecimal system we have one of the sixteen values 0 through 15. However, the decimal values from 10 to 15 will be represented by alphabet A (=10), B (=11), C (=12), D (=13), E (=14) and F (=15).

Conversion of a binary number to an octal number or a hexadecimal number is very simple, as it requires simple grouping of the binary digits into groups of three or four. Consider the binary number 11011011.

It may be converted into octal or hexadecimal numbers

as $(11011001)_2$ = (011) (011) (001)

= $(331)_8$

= (1101) (1001)

= $(D9)_{16}$

Note that adding a leading zero does not alter the value of the number. Similarly for grouping the digits in the fractional part of a binary number, trailing zeros may be added without changing the value of the number. Number System Conversions In general, conversion between numbers with different radices cannot be done by simple substitutions. Such conversions would involve arithmetic operations. Let us work out procedures for converting a number in any radix to radix 10, and viceversa.

Decimal value of the number is determined by converting each digit of the number to its radix-10 equivalent and expanding the formula using radix-10 arithmetic.

Some examples are:

$(331)_8$

= 3 x 82 + 3 x 81 + 1 x 80

= 192 + 24 + 1

= $(217)_{10}$ $(D9)_{16}$

= 13 x 161 + 9 x 160

= 208 + 9

= $(217)_{10}$

$(33.56)_8$

= 3 x 81 + 3 x 80 + 5 x 8-1 + 6 x 8-2

= $(27.69875)_{10}$ $(E5.A)_{16}$

= 14 x 161 + 5 x 160 + 10 x 16-1

= $(304.625)_{10}$

The conversion formula can be rewritten as D = ((... ((dn-1).r + dn-2) r + ...).r + d1).r + d0. This forms the basis for converting a decimal number D to a number with radix r. If we divide the right hand side of the above formula by r, the remainder will be d0, and the quotient will be Q = ((... ((dn-1).r + dn-2) r +....).r + d1 Thus, d0 can be computed as the remainder of the long division of D by the radix r. As the quotient Q has the same form as D, another long division by r

will give d1 as the remainder. This process can continue to produce all the digits of the number with radix r.

Consider the following examples.

	Quotient	Remainder
156 ÷ 2	78	0
78 ÷ 2	39	0
39 ÷ 2	19	1
19 ÷ 2	9	1
9 ÷ 2	4	1
4 ÷ 2	2	0
2 ÷ 2	1	0
1 ÷ 2	0	1

$(156)_{10} = (10011100)_2$

	Quotient	Remainder
678 ÷ 8	84	6
84 ÷ 8	10	4
10 ÷ 8	1	2

1 ÷ 8 0 1

$(678)_{10} = (1246)_8$

	Quotient	Remainder
678 ÷ 16	42	6
42 ÷ 16	2	A
2 ÷ 16	0	2

$(678)_{10} = (2A6)_{16}$

Representation of Negative Numbers In our traditional arithmetic we use the "+" sign before a number to indicate it as a positive number and a "-" sign to indicate it as a negative number. We usually omit the sign before the number if it is positive. This method of representation of numbers is called "sign-magnitude" representation. But using "+" and "-" signs on a computer is not convenient, and it becomes necessary to have some other convention to represent the signed numbers. We replace "+" sign with "0" and "-" with "1". These two symbols already exist in the binary system. Consider the following examples:

(+1100101)2 (01100101)2

(+101.001)2 (0101.001)2

(-10010)2 (110010)2

(-110.101)2 (1110.101)2

In the sign-magnitude representation of binary numbers the first digit is always treated separately. Therefore, in working with the signed binary numbers in sign-magnitude form the leading zeros should not be ignored. However, the leading zeros can be ignored after the sign bit is separated.

For example,

1000101.11 = -

101.11

While the sign-magnitude representation of signed numbers appears to be natural extension of the traditional arithmetic, the arithmetic operations with signed numbers in this form are not that very convenient, either for implementation on the computer or for hardware implementation. There are two other methods of representing signed numbers.

Diminished Radix Complement (DRC) or (r-1)-complement Radix Complement (RX) or r-complement When the numbers are in binary form Diminished Radix Complement will be known as “one’s-complement” Radix complement will be known as “two’s-complement”. If this representation is extended to the decimal numbers they will be known as 9’scomplement and 10’s- complement respectively.

One’s Complement Representation

Let A be an n-bit signed binary number in one’s complement form. The most significant bit represents the sign. If it is a “0” the number is positive and if it is a “1” the number is negative. The remaining (n-1) bits represent the magnitude, but not necessarily as a simple weighted number.

Consider the following one's complement numbers and their decimal equivalents:

0111111	+ 63
0000110	--> + 6
0000000	--> + 0
1111111	--> + 0
1111001	--> - 6
1000000	--> - 63

There are two representations of "0", namely 000.....0 and 111 .. 1.

From these illustrations we observe

- If the most significant bit (MSD) is zero the remaining (n-1) bits directly indicate the magnitude.
- If the MSD is 1, the magnitude of the number is obtained by taking the complement of all the remaining (n-1) bits.

For example consider one's complement representation of -6 as given above.

- Leaving the first bit '1' for the sign, the remaining bits 111001 do not directly represent the magnitude of the number -6.
- Take the complement of 111001, which becomes 000110 to determine the magnitude.

In the example shown above a 7-bit number can cover the range from +63 to -63. In general an n-bit number has a range from +(2n-1 - 1) to -(2n-1 - 1) with two representations for zero. The representation also suggests that if A is an integer in one's complement form, then one's complement of A = -A

One's complement of a number is obtained by merely complementing all the digits. This relationship can be extended to fractions as well.

For example

if A = 0.101 (+0.625)10,

then the one's complement of A is 1.010,

which is one's complement representation of (-0.625)10.

Similarly consider the case of a mixed number.

A = 010011.0101 (+19.3125)10

One's complement of A = 101100.1010 (- 19.3125)10

This relationship can be used to determine one's complement representation of negative decimal numbers. Example 1: What is one's complement binary representation of decimal number -75? Decimal number 75 requires 7 bits to represent its magnitude in the binary form. One additional bit is needed to represent the sign.

Therefore, one's complement representation of 75 = 01001011 one's complement representation of -75 = 10110100

Two's Complement Representation Let A be an n-bit signed binary number in two's complement form. The most significant bit represents the sign. If it is a "0", the number is positive, and if it is "1" the number is negative. The remaining (n-1) bits represent the magnitude, but not as a simple weighted number. Consider the following two's complement numbers and their decimal equivalents:

0111111 + 63

0000110 + 6

0000000 + 0

1111010 - 6

1000001 - 63

1000000 - 64

There is only one representation of "0", namely 000 . 0.

From these illustrations we observe If most significant bit (MSD) is zero the remaining (n-1) bits directly indicate the magnitude.

If the MSD is 1, the magnitude of the number is obtained by taking the complement of all the remaining (n-1) bits and adding a 1.

Consider the two's complement representation of -6.

We assume we are representing it as a 7-bit number.

Leave the sign bit.

The remaining bits are 111010.

These have to be complemented

(that is 000101) and a 1 has to be added (that is 000101 + 1 = 000110 = 6).

In the example shown above a 7-bit number can cover the range from +63 to -64.

In general an n-bit number has a range from + (2n-1 - 1) to - (2n-1) with one representation for zero. The representation also suggests that if A is an integer in two's complement form, then Two's complement of A = -A Two's complement of a number is obtained by complementing all the digits and adding '1' to the LSB. This relationship can be extended to fractions as well.

If A = 0.101 (+0.625)10,

then the two's complement of A is 1.011,

which is two's complement representation of (-0.625)10.

Similarly consider the case of a mixed number.

A = 010011.0101 (+19.3125)10

Two's complement of A = 101100.1011 (- 19.3125)10

This relationship can be used to determine two's complement representation of negative decimal numbers.

Example 2:

What is two's complement binary representation of decimal number -75?

Decimal number 75 requires 7 bits to represent its magnitude in the binary form. One additional bit is needed to represent the sign. Therefore, Two's complement representation of 75 = 01001011 Two's complement representation of -75 = 10110101

CODES:

When we wish to send information over long distances unambiguously it becomes necessary to modify (encoding) the information into some form before sending, and convert (decode) at the receiving end to get back the original information. This process of encoding and decoding is necessary because the channel through which the information is sent may distort the transmitted information. Much of the information is sent as numbers. While these numbers are created using simple weighted-positional numbering systems, they need to be encoded before transmission. The modifications to numbers were based on changing the weights, but predominantly on some form of binary encoding. There are several codes in use in the context of present day information technology, and more and more new codes are being generated to meet the new demands.

Coding is the process of altering the characteristics of information to make it more suitable for intended application By assigning each item of information a unique combination of 1s and 0s we transform some given information into binary coded form. The bit combinations are referred to as "words" or "code words". In the field of digital systems and computers different bit combinations have different designations.

Bit - a binary digit 0 or 1

Nibble - a group of four bits

Byte - a group of eight bits

Word - a group of sixteen bits;

a word has two bytes or four nibbles

Sometimes ‘word’ is used to designate a larger group of bits also, for example 32 bit or 64 bit words. We need and use coding of information for a variety of reasons

- to increase efficiency of transmission,
- to make it error free,
- to enable us to correct it if errors occurred,
- to inform the sender if an error occurred in the received information etc.
- for security reasons to limit the accessibility of information
- to standardise a universal code that can be used by all

Coding schemes have to be designed to suit the security requirements and the complexity of the medium over which information is transmitted.

Binary Coded Decimal Codes

The main motivation for binary number system is that there are only two elements in the binary set, namely 0 and 1. While it is advantageous to perform all computations on hardware in binary forms, human beings still prefer to work with decimal numbers. Any electronic system should then be able to accept

decimal numbers, and make its output available in the decimal form. One method, therefore, would be to

- convert decimal number inputs into binary form
- manipulate these binary numbers as per the required functions, and
- convert the resultant binary numbers into the decimal form

However, this kind of conversion requires more hardware, and in some cases considerably slows down the system. Faster systems can afford the additional circuitry, but the delays associated with the conversions would not be acceptable. In case of smaller systems, the speed may not be the main criterion, but the additional circuitry may make the system more expensive.

We can solve this problem by encoding decimal numbers as binary strings, and use them for subsequent manipulations.

There are ten different symbols in the decimal number system: 0, 1, 2, . . ., 9. As there are ten symbols we require at least four bits to represent them in the binary form. Such a representation of decimal numbers is called binary coding of decimal numbers.

As four bits are required to encode one decimal digit, there are sixteen four-bit groups to select ten groups. This would lead to nearly 30 x 1010 (16C10.10!) possible codes. However, most of them will not have any special properties that would be useful in hardware design. We wish to choose codes that have some desirable properties like

- ease of coding
- ease in arithmetic operations

- minimum use of hardware
- error detection property
- ability to prevent wrong output during transitions

In a weighted code the decimal value of a code is the algebraic sum of the weights of 1s appearing in the number. Let (A)10 be a decimal number encoded in the binary form as a3a2a1a0. Then

(A)10 = w3a3 + w2a2 + w1a1 +w0a0

where w3, w2, w1 and w0 are the weights selected for a given code, and a3,a2,a1and a0 are either 0s or 1s.

The more popularly used codes have the weights as

w3	w2	w1	w0
8	4	2	1
2	4	2	1
8	4	-2	-1

Table 1.1 - The decimal numbers in various codes

Decimal digit	Weights 8 4 2 1	Weights 2 4 2 1	Weights 8 4 -2 -1
0	0 0 0 0	0 0 0 0	0 0 0 0
1	0 0 0 1	0 0 0 1	0 1 1 1
2	0 0 1 0	0 0 1 0	0 1 1 0
3	0 0 1 1	0 0 1 1	0 1 0 1
4	0 1 0 0	0 1 0 0	0 1 0 0
5	0 1 0 1	1 0 1 1	1 0 1 1
6	0 1 1 0	1 1 0 0	1 0 1 0
7	0 1 1 1	1 1 0 1	1 0 0 1
8	1 0 0 0	1 1 1 0	1 0 0 0
9	1 0 0 1	1 1 1 1	1 1 1 1

In all the cases only ten combinations are utilized to represent the decimal digits. The remaining six combinations are illegal. However, they may be utilized for error detection purposes.

Consider, for example, the representation of the decimal number 16.85 in Natural Binary Coded Decimal code (NBCD)

$(16.85)10 = (0001\ 0110\ .\ 1000\ 0101)_{NBCD}$

1 6 8 5

There are many possible weights to write a number in BCD code. Some codes have desirable properties, which make them suitable for specific applications. Two such desirable properties are: 1. Self-complementing codes 2. Reflective codes When we perform arithmetic operations, it is often required to take the “complement” of a given number. If the logical complement of a coded number is also its arithmetic complement, it will be convenient from hardware

point of view. In a self-complementing coded decimal number, (A)10, if the individual bits of a number are complemented it will result in (9 - A)10.

Table 1.2 : Three self-complementing codes

Decimal Digit	Excess-3 Code	631-1 Code	2421 Code
0	0011	0011	0000
1	0100	0010	0001
2	0101	0101	0010
3	0110	0111	0011
4	0111	0110	0100
5	1000	1001	1011
6	1001	1000	1100
7	1010	1010	1101
8	1011	1101	1110
9	1100	1100	1111

Unit Distance Codes

There are many applications in which it is desirable to have a code in which the adjacent codes differ only in one bit. Such codes are called Unit distance Codes. “Gray code” is the most popular example of unit distance code. The 3-bit and 4-bit Gray codes are

Table 1.3 : Unit distance Codes

Decimal	3 bit Gray	4 bit Gray
0	000	0000
1	001	0001
2	011	0011
3	010	0010
4	110	0110
5	111	0111
6	101	0101
7	100	0100
8	-	1100
9	-	1101
10	-	1111
11	-	1110
12	-	1010
13	-	1011
14	-	1001
15	-	1000

Alphanumeric Codes

When information to be encoded includes entities other than numerical values, an expanded code is required. For example, alphabetic characters (A, B,

....Z) and special operation symbols like +, -, /, *, (,) and other special symbols are used in digital systems. Codes that include alphabetic characters are commonly referred to as Alphanumeric Codes. However, we require adequate number of bits to encode all the characters. As there was a need for alphanumeric codes in a wide variety of applications in the early era of computers, like teletype, punched tape and punched cards, there has always been a need for evolving a standard for these codes. Alphanumeric keyboard has become ubiquitous with the popularization of personal computers and

notebook computers. These keyboards use ASCII (American Standard Code for Information Interchange) code

Table 1.4 : Alpha Numeric Codes

b4	b3	b2	b1	b7 b6 b5							
				000	001	010	011	100	101	110	111
0	0	0	0	NUL	DLE	SP	0	@	P	`	p
0	0	0	1	SOH	DC1	!	1	A	Q	a	q
0	0	1	0	STX	DC2	"	2	B	R	b	r
0	0	1	1	ETX	DC3	#	3	C	S	c	s
0	1	0	0	EOT	DC4	$	4	D	T	d	t
0	1	0	1	ENQ	NAK	%	5	E	U	e	u
0	1	1	0	ACK	SYN	&	6	F	V	f	v
0	1	1	1	BEL	ETB	'	7	G	W	g	w
1	0	0	0	BS	CAN	(	8	H	X	h	x
1	0	0	1	HT	EM	)	9	I	Y	i	y
1	0	1	0	LF	SUB	*	:	J	Z	j	z
1	0	1	1	VT	ESC	+	;	K	[	k	{
1	1	0	0	FF	FS	,	<	L	\	l	|
1	1	0	1	CR	GS	-	=	M	]	m	}
1	1	1	0	SO	RS	.	>	N		n	~
1	1	1	1	SI	US	/	?	O	-	o	DEL

Alphanumeric codes like EBCDIC (Extended Binary Coded Decimal Interchange Code) and 12-bit Hollerith code are in use for some applications. However, ASCII code is now the standard code for most data communication networks. Therefore, the reader is urged to become familiar with the ASCII code.