Chapter 3: Number System

Definitions:

Number System:

Number System is a numeral system is a framework where a set of number are represented by numerals in a consistent manner. It can be seen as the context that allows the numeral 11 to be printed as binary numeral for three, the decimal numeral for eleven or other number is different system.

Numeral system is sometimes called **number system.**

Number system is an every computer stores number, letters, and other special character in coded form. Before going into the details of theses codes, it is essential to have a basic under standing of **number system.** Two types of Number System are:

Non – Positional Number Systems:

In early days, human beings counted on fingers. When counting beyond ten fingers, they used stones, pebbles, or stick to indicate values. This method of counting uses an additive approach or **non – positional number system.**

Positional Number Systems:

In a positional number system, there are only a few symbols called digits. These symbols represent different values, depending on the position they occupy in a number. The value of each digit in such a number is determinate by three considerations:

1. The digit itself,
2. The position of the digit in the number, and
3. The base of the number system.

Computer Memory:

Computer memory is basically divided into two parts. Main memory (RAM) and auxiliary memory (HDD, FDD, CD). They are also called primary and secondary storage respectively. Primary storage can store less amount of data compared to secondary storage.

Nibble: Four bits constitute a Nibble.

Bit: The binary digit which is either 0 or 1.
One binary number (0 or 1) is called bit. Bit is the smallest unit of information.

Byte: A group of 8 bits used to represent characters.

1 byte = 8 bits
1 kilo byte (KB) = 1024 bytes =10^3 bytes
1 Mega byte (MB) = 1024 KB =10^3 KB = 10^6 bytes
1 Giga byte (GB) = 1024 MB =10^3 MB = 10^9 bytes
1 Tera byte (TB) = 1024 GB =10^3 GB = 10^{12} bytes

(BCD) Binary coded Decimal:

A representation of decimal which uses unlike string of 4 (0000) bites to representation each digit.

EBCDIC: Extended binary – coded decimal interchange code.

ASCII: American Standard Code for Information Interchange.

ASCII-7: ASCII-7 is a 7 bit code that can represent 2^7 =128.
Computer using 8-bit ASCII either set that 8^{th} (left most bits) of it each bit as zero (0) us it as parity bit.

ASCII-8:

Is an 8 bit code that can represent 2^8 =256 different characters

The additional bit is added to the left of the 7^{th} bit (left most bit) of ASCII-7 codes.

Decimal to Binary:

We use decimal number system. In this system, base is equal to 10 because there are altogether ten symbols or digits (0, 1, 2, 3, 4, 5, 6, 7, 8, and 9). You know that in decimal number system.

Binary Addition

Consider the addition of decimal numbers:

```
 23
+48
___
```

We begin by adding 3+8=11. Since 11 is greater than 10, a one is put into the 10's column (carried), and a 1 is recorded in the one's column of the sum. Next, add {(2+4) +1} (the one is from the carry)=7, which is put in the 10's column of the sum. Thus, the answer is 71.

Binary addition works on the same principle, but the numerals are different. Begin with one-bit binary addition:

```
  0    0    1
 +0   +1   +0
___  ___  ___
  0    1    1
```

1+1 carries us into the next column. In decimal form, 1+1=2. In binary, any digit higher than 1 puts us a column to the left (as would 10 in decimal notation). The decimal number "2" is written in binary notation as "10" (1*2^1)+(0*2^0). Record the 0 in the ones column, and carry the 1 to the twos column to get an answer of "10." In our vertical notation,

```
  1
 +1
___
```

10

The process is the same for multiple-bit binary numbers:

```
 1010
+1111
_____
```

- Step one:
  Column 2^0: 0+1=1.
  Record the 1.
  Temporary Result: 1; Carry: 0
- Step two:
  Column 2^1: 1+1=10.
  Record the 0, carry the 1.
  Temporary Result: 01; Carry: 1
- Step three:
  Column 2^2: 1+0=1 Add 1 from carry: 1+1=10.
  Record the 0, carry the 1.
  Temporary Result: 001; Carry: 1
- Step four:
  Column 2^3: 1+1=10. Add 1 from carry: 10+1=11.
  Record the 11.
  Final result: 11001

Alternately:

```
 11   (carry)
 1010
+1111
_____
11001
```

Always remember

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

Binary Subtraction

Here are some examples of binary subtraction. These are computed without regard to the word size, hence there can be no sense of "overflow" or "underflow". Work the columns right to left subtracting in each column. If you must subtract a one from a zero, you need to “borrow” from the left, just as in decimal subtraction.

• 1011011 − 10010 = 1001001:

	1	0	1	1	0	1	1
−			1	0	0	1	0
	1	0	0	1	0	0	1

• 1010110 − 101010 = 101100:

	0		0				
	×1	10	×1	10	1	1	0
−		1	0	1	0	1	0
		1	0	1	1	0	0

• 1000101 − 101100 = 11001:

	0	1	1				
	×1	×10	×10	10	1	0	1
−		1	0	1	1	0	0
			1	1	0	0	1

• 100010110 − 1111010 = 10011100:

	0	1	1	1	10				
	×1	×10	×10	×10	×1	10	1	1	0
−			1	1	1	1	0	1	0
		1	0	0	1	1	1	0	0

• 101101 − 100111 = 110:

			0	10		
	1	0	×1	×1	10	1
−	1	0	0	1	1	1
				1	1	0

• 1110110 − 1010111 = 11111:

		0	10	1	10	10	
	1	×1	×1	×10	×1	×1	10
−	1	0	1	0	1	1	1
			1	1	1	1	1

Transforming Binary, Hex and Octal

Binary	**Hex**	**Binary**	**Octal**
0000	0	000	0
0001	1	001	1
0010	2	010	2
0011	3	011	3

0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	A
1011	B
1100	C
1101	D
1110	E
1111	F

100	4
101	5
110	6
111	7

1. $153_{10} = ?_2$

2	153	1
2	76	0
2	38	0
2	19	1
2	9	1
2	4	0
2	2	0
	1	1

Ans.
$153_{10}=10011001_2$

2. $233_{10} = ?_2$

2	233	1
2	116	0
2	58	0
2	29	1
2	14	0
2	7	1
2	3	1
	1	1

$47_{10}=11101001_2$

3. $3856_{10} = ?_2$

2	3856	0
2	1928	0
2	964	0
	482	0
2	241	1
2	120	0
2	60	0
2	30	0
2	15	1
2	7	1
2	3	1
2	1	1

$3856_{10} = 111100010000_2$

4. $47_{10} = ?_2$

2	47	
2	23	1
2	11	1
2	5	1
2	2	1
	1	0
	1	1

$47_{10} = 101111_2$

5. 2764.35_{10}

$= (2 * 10^3) + (7 * 10^2) + (6 * 10^1) + (4 * 10^0) + (3 * 10^{-1}) + (5 * 10^{-2})$

$= (2 * 1000) + (7 * 100) + (6 * 10) + (4 * 1) + (3 / 10) + (5/100)$

$= 2000 + 700 + 60 + 4 + 0.3 + 0.05$

$= 2764.35$

6. $(107.625)_{10} = (?)_2$

2	107	1
2	53	1
2	26	0
2	13	1
2	6	0
2	3	1
	1	1

0.625	0.250	0.500
* 2	* 2	* 2
1.250	0.500	1.000

$(107.625)10 = (1101011.101)_2$

7. $(76.83)_{10} = (?)_2$

2	76	0
2	38	0

0.83	0.66	0.32	0.64	0.28	0.56
* 2	* 2	* 2	* 2	* 2	* 2
1.66	1.32	0.64	1.28	0.56	1.23

2	19	1
2	9	1
2	4	0
2	2	0
	1	1

$76.38)_{10}= (1001100.110101)_2$

Binary to Decimal:

Binary number system is like decimal number system, except that the base is 2, instead of 10. We can use only two symbols or digits (0 and 1) in this system. Note that the largest digit is 1 (One less the base).

1. $(101000)_2=?\ _{10}$

   $= 1*(2)^5 + (0*2)^4 + (1*2)^3+ (0*2)^2+ (0*2)^1+ (0*2)^0$

   = 32+0+8+0+0+0

   = 32+8

   $= (40)_{10}$

2. $(101111)_2=?\ _{10}$

   $= 1*(2)^5 + 0*(2)^4 + 1*(2)^3 + 1*(2)^2 + 1*(2)^1 + 1*(2)^0$

   = 32+0+8+4+2+1

   = 32+8+4+2+1

   $= (47)_{10}$

3. $(10011101)_2=?\ _{10}$

   $= (1*2)^7 + (0*2)^6+ (0*2)^5 + (1*2)^4+ (1*2)^3+ (1*2)^2+ (0*2)^1+ (1*2)^0$

   =128+0+0+16+8+4+0+1

   =128+16+8+4+1

   $= (157)_{10}$

4. $(10010110)_2=?$ 10

$= (1*2)^7 + (0*2)^6 + (0*2)^5 + (1*2)^4 + (0*2)^3 + (1*2)^2 + (1*2)^1 + (0*2)^0$

=128+0+0+16+0+4+2+0

=128+16+4+2

$= (150)_{10}$

Real Binary to Decimal

1. $(11001.01110)_2 = ?_{10}$

   $= (1*2^4) + (1*2^3) + (0*2^2) + (0*2^1) + (1*2^0) + (1*2^4) + (1*2^3) + (0*2^2) + (0*2^{-1}) + (1*2^{-2})$

   = 16 + 8 + 0 + 0 + 1 + 1/4 + 1/8 +1/16

   = 25 + 0.25 + 0.125 + 0.0625

   = 25.4375

   Ans. $(11001.01110)_2 = (25.4375)_{10}$

2. $(1111001.01111)_2 = ?_{10}$

   $= (1*2^6) + (1*2^5) + (1*2^4) + (1*2^3) + (0*2^2) + (0*2^1) + (1*2^0) +$

   $+ (0*2^{-1}) + (1*2^{-2}) + (1*2^{-3}) + (1*2^{-4}) + (1*2^{-5})$

   = 64 + 32 +16 + 8 + 0 + 0 + 1 + 0 + 1/4 + 1/8 + 1/16 + 1/32

   = 121 + 0.25 + 0.125 + 0.0625 + 0.021125

   = 121.458

   Ans. $(111101.01111)_2 = (121.458)_{10}$

BCD Number: Binary Coded Decimal

Binary Coded Decimal Code is one of the early computer codes. The idea of this coding scheme is to converts each digit of a decimal number into its binary equivalent instead of converting the entire decimal value into a binary number.

BCD-Binary Coded Decimal->Binary equivalent of each decimal expressed using 4 Bits-> For single digit decimal BCD is same as its binary. In BCD only first 10 binary Numbers are valid. The remaining 5 are invalid.

Gray code is an unweighed code.

"In BCD, a digit is usually represented by four bits which, in general, represent the values/digits/characters 0–9. Other bit combinations are sometimes used for a sign or other indications."

Basically, BCD is just a representation of a single number using 4 Bits Binary Example:

9 in BCD = 1001

8 in BCD = 1000

ex:

G3=B3

E.g. 4210

4=0100

2=0010

OR

01000010 in BCD

E.g. write the binary digits used to record the world BASE in BCD.

Solution:

In BCD binary notation

B = 110010

A = 110001

S = 010010

E = 110101

Hence, the binary digits

Will record the BASE in BCD.

E.g. using octal notation, write the BCD coding for the word DIGIT.

Solution:

In BCD octal notation

D = 64
I = 71
G = 67
I = 71
T = 23

Hence, the BCD coding for the word DIGIT in octal notation will be

D = 64, I = 71, G = 67, I = 71, T = 23

EBCDIC: Extended Binary Coded Decimal Interchange Code

The major problem with BCD cdoe is that it can represent only 64(2^6) different character. This is not sufficient for providing decimal numbers (10), lowercase letters (26) uppercase (26), and a large number of other special character (28+).
EBCDIC (Extended Binary Coded Decimal Information Code) is an eight-bit character set that was developed by International Business Machines (IBM). It was the character set used on most computers manufactured by IBM prior to 1981.
EBCDIC is not used on the IBM PC and all subsequent "PC clones". These computer systems use ASCII as the primary character and symbol coding system. (Computer makers other than IBM used the ASCII system since its inception in the 1960s.)
EBCDIC is widely considered to be an obsolete coding system, but is still used in some equipment, mainly in order to allow for continued use of software written many years ago that expects an EBCDIC communication environment.
(Information on printing color tables on color printers can be found here.)

Notes:
The following EBCDIC characters have no equivalents in the ASCII or ISO-8859

character sets used on the Internet, and cannot be displayed in this table.

(1) Code 79 is a solid vertical bar, similar to the broken vertical bar (character 106).
(2) Code 204 is the mathematics integration symbol.
(3) Code 206 is a "Y" drawn with only right angles.
(4) Code 236 is a horizontally-flipped "h".

In this table, the code or symbol name is shown on the first line, followed by the decimal value for that code or symbol, followed by the hexadecimal value. The binary value can be computed based on the row and column where the code or symbol resides, or directly from the hexadecimal value. For example, the character "+" has the binary value "0100 1110", with "0100" taken from the row and "1110" taken from the column. Similarly, the lowercase letter 'p' has the binary value "1001 0111".

The background color for each code or symbol indicates the category that the code resides in. Red indicates control (non-printable) codes. Orange indicates basic punctuation and symbols. Yellow indicates numeric digits. Green indicates the uppercase letters. Blue indicates lowercase letters. Purple indicates codes that have no assigned function or symbol in IBMs U.S. assignments. (Some of these values do have character assignments when using the T-11 or TN print chains or fonts. T-11 and TN are generally supersets of the standard EBCDIC shown above.)

E.g. write the EBCDIC coding for the word "BIT" in both binary and hexadecimal notations. How many bytes are required to store this word using this coding?

Solution:

In EBCDIC

B = 1100 0010 in binary and C2 in hexadecimal

I = 1100 1001 in binary and C9 in hexadecimal

T = 1100 0011 in binary and E3 in hexadecimal

Hence, the EBCDIC coding for the word "BIT" will be

B = 11000010

I = 11001001

T = 11000011 in Binary and

C2=B, C9=I, E3=T in Hexadecimal

ASCII: American Standard Code for Information Interchange

Another widely used computer code is ASCII. Several American computer manufactures have adopted ASCII as their computers' internal code. This code is popular in data communication, is used almost exclusively to represent data in microcomputers, and is frequently found in larger computers produced by some vendors.

ASCII is the American Standard Code for Information Interchange, also known as ANSI X3.4. There are many variants of this standard, typically to allow different code pages for language encoding, but they all basically follow the same format. ASCII is quite elegant in the way it represents characters, and it is very easy to write code to manipulate upper/lowercase and check for valid data ranges.

ASCII is essentially a 7-bit code which allows the 8th most significant bit (MSB) to be used for error checking, however most modern computer systems tend to use ASCII values of 128 and above for extended character sets.

The basics of ASCII

The name ASCII, originally an abbreviation for "American Standard Code for Information Interchange", denotes an old character repertoire, code, and encoding.

Most character codes currently in use contain ASCII as their subset in some sense. ASCII is the safest character repertoire to be used in data transfer. However, not even all ASCII characters are "safe"!

ASCII has been used and is used so widely that often the word ASCII refers to "text" or "plain text" in general, even if the character code is something else! The words "ASCII file" quite often mean any text file as opposite to a binary file.

The definition of ASCII also specifies a set of control codes ("control characters") such as linefeed (LF) and escape (ESC). But the character repertoire proper, consisting of the printable characters of ASCII, is the following (where the first item is the blank, or space, character):

! " # $ % & ' ( ) * + , - . /

0 1 2 3 4 5 6 7 8 9 : ; < = > ?

@ A B C D E F G H I J K L M N O

P Q R S T U V W X Y Z [\ ] ^ _

` a b c d e f g h i j k l m n o

p q r s t u v w x y z { | } ~

The appearance of characters varies, of course, especially for some special characters. Some of the variation and other details are explained in The ISO Latin 1 character repertoire - a description with usage notes.

National variants of ASCII

There are several national variants of ASCII. In such variants, some special characters have been replaced by national letters (and other symbols). There is great variation here, and even within one country and for one language there might be different variants. The original ASCII is therefore often referred to as US-ASCII; the formal standard (by ANSI) is ANSI X3.4-1986.

The phrase "original ASCII" is perhaps not quite adequate, since the creation of ASCII started in late 1950s, and several additions and modifications were made in the 1960s. The 1963 version had several unassigned code positions. The ANSI standard, where those positions were assigned, mainly to accommodate lower case letters, was approved in 1967/1968, later modified slightly. For the early history, including pre-ASCII character codes, see Steven J. Searle's A Brief History of Character Codes in North America, Europe, and East Asia and Tom Jennings' ASCII: American Standard Code for Information Infiltration. See also Jim Price's ASCII Chart, Mary Brandel's 1963: ASCII Debuts, and the computer history documents, including the background and creation of ASCII, written by Bob Bemer, "father of ASCII".

The international standard ISO 646 defines a character set similar to US-ASCII but with code positions corresponding to US-ASCII characters @[\]{|} as "national use positions". It also gives some liberties with characters #$^`~. The standard also defines "international reference version (IRV)", which is (in the 1991 edition of ISO 646) identical to US-ASCII. Ecma International has issued the ECMA-6 standard, which is equivalent to ISO 646 and is freely available on the Web.

Within the framework of ISO 646, and partly otherwise too, several "national variants of ASCII" have been defined, assigning different letters and symbols to the "national use" positions. Thus, the characters that appear in those positions - including those in US-ASCII - are somewhat "unsafe" in international data transfer, although this problem is losing significance. The trend is towards using the corresponding codes strictly for US-ASCII meanings; national characters are handled otherwise, giving them their own, unique and universal code positions in character codes larger than ASCII. But old software and devices may still reflect various "national variants of ASCII".

The following table lists ASCII characters which might be replaced by other characters in national variants of ASCII. (That is, the code positions of these US-ASCII characters might be occupied by other characters needed for national use.) The lists of characters appearing in national variants are not intended to be exhaustive, just typical examples.

dec	oct	hex	glyph	official Unicode name	National variants
35	43	23	#	number sign	£ Ù
36	44	24	$	dollar sign	¤
64	100	40	@	commercial at	É § Ä à ³
91	133	5B	[	left square bracket	Ä Æ ° â ¡ ÿ é
92	134	5C	\	reverse solidus	Ö Ø ç Ñ ½ ¥
93	135	5D	]	right square bracket	Å Ü § ê é ¿ |
94	136	5E	^	circumflex accent	Ü î
95	137	5F	_	low line	è
96	140	60	`	grave accent	é ä µ ô ù
123	173	7B	{	left curly bracket	ä æ é à ° ¨
124	174	7C	|	vertical line	ö ø ù ò ñ f
125	175	7D	}	right curly bracket	å ü è ç ¼
126	176	7E	~	tilde	ü ¯ ß ¨ û ì ´ _

Almost all of the characters used in the national variants have been incorporated into ISO Latin 1. Systems that support ISO Latin 1 in principle may still reflect the use of national variants of ASCII in some details; for example, an ASCII character might get printed or displayed according to some national variant. Thus, even "plain ASCII text" is thereby not always portable from one system or application to another.

Example:

01000 0010

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

11. $45=?_2$

2	45	
2	22	1
2	11	0
2	5	1
2	2	1
2	1	0

$45_{10}= (101101)_2$

12). $(101101)_2=?_{10}$

$= (1*2)^5 + (0*2)^4 + (1*2)^3+ (1*2)^2 + (0*2)^1 + (1*2)^0$

$=32+0+8+4+0+1$

$=32+8+4+1$

$= (45)_{10}$

13). $(905)_{10}$ =? $_{16}$

	905	
16	56	9
	3	8

$(905)_{10} = (389)_{16}$

HEXA Decimal number:

The hexadecimal number system is a positional weighted system uses 16 symbols 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F.

The decimal equivalent of A, B, C, D, and F is assumed to be 10, 11, 12, 13, 14 and respectively.

The base or radix in this system is 16.

0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
A10	1010
B11	1011
C12	1100

D13	1101
E14	1110
F15	1111

1. $(2050)_{10}$=? $_{16}$

16	2050	
16	128	2
	8	0

$(2050)_{10}= (802)_{16}$

2. $(15050)_{10}$=? $_{16HEXA}$

16	15050	
16	940	10A
16	5	12C
	3	10A

$(15050)_{10}$=(3ACA) $_{16HEXA}$

3. (ABCD) $_{16}$=? $_{10}$ = ?$_2$

= $(10*16)^3 + (11*16)^2 + (12*16)^1 + (13*16)^0$

=0960+2816+192+13

= $(43981)_{10}$=? $_2$

2	43981	
2	21990	1
2	1995	0
2	5497	1
2	2748	1
2	1374	0
2	687	0
2	343	1
2	171	1
2	85	1
2	42	1

2	21	0
2	10	1
2	5	0
2	2	1
	1	0

$(43981)_{10}$= $(1010101111001101)_2$

Octal Decimal number:

This positional weighted system uses the 8 symbols 0, 1, 2, 3, 4, 5, 6, and 7.

It is the base – 8 number system as there all eight in the system.

Octal numbers can be made from binary numbers by grouping consecutive digits into groups of three.

Transform 3724 $_8$ into binary number

37,24 $_8$ = 011 111 010 100 $_2$ = **111110101** $_2$

1. $(112)_8 = (?)_{10}$

$= 1 * 8^2 + 1 * 8^1 + 2 * 8^0$

$= 64 + 8 + 2$

$= (74)_{10}$

2. $(2057)_8 = (?)_{10}$

$= 2 * 8^3 + 0 * 8^2 + 5 * 8^1 + 5 * 8^0$

$= 1024 + 0 + 40 + 5$

$= (1071)_{10}$

3. $(952)_{10} = (?)_8$

8	952	0
8	119	7
8	14	6
8	1	1
	0	

1. $(3015)_{10} = (?)_8$

8	3015	7
8	376	0
8	47	7
8	5	5
	0	

Hence, $(952)_{10}$ = $(1670)_8$	**Hence, $(3015)_{10}$ = (5707)**

Example

Transform 11,000011001 $_2$ into hex number

11,000011001 $_2$ = 11 0000 1100 1 $_2$

= 0011 0000 1100 1000 $_2$

= 3 0 C 8 $_{16}$

= **30C8** 16

Example

Transform 111101110011101 $_2$ into octal number

1111011,10011101 $_2$ = 1 111 011 100 111 01 $_2$

= 001 111 011 100 111 010 $_2$

= 1 7 3 4 7 2 $_{16}$

= **173 472** $_{16}$

Exercise

7).97_{10}=? $_2$

2	97	
2	48	1
2	24	0
2	12	0
2	6	0
2	3	0
	1	1

97_{10}=1100001_2

= $(1*2)^6+ (1*2)^5+ (0*2)^4+ (0*2)^3+ (0*2)^2+ (0*2)^1+ (1*2)^0$

=64+32+0+0+0+0+1

=64+32+1

= $(97)_{10}$

8). $(1000)_{10}=?_{2}$

2	1000	
2	500	0
2	250	0
2	125	0
2	62	1
2	31	0
2	15	1
2	7	1
2	3	1
	1	1

$1000_{10}=1111101000_{2}$

9). $(1111101000)_{2}=?\ 1_{0}$

$=(1*2)^9+(1*2)^8+(1*2)^7+(1*2)^6 \quad +(1*2)^5+(0*2)^4+(1*2)^3+(0*2)^2+(0*2)^1+(0*2)^0$

$=512+256+128+64+32+0+8+0+0+0$

$=512+256+128+64+32+8$

$= (1000)_{10}$

10). $(1011)_{10}=?_{2}$

2	1011	
2	505	1
2	252	1
2	126	0
2	63	0
2	31	1
2	15	1
2	7	1

2	3	1
	1	1

$1011_{10}=1111110011_2$

$=(1*2)^9+(1*2)^8+(1*2)^7+(1*2)^6+(1*2)^5+(1*2)^4+(0*2)^3+(0*2)^2+(1*2)^1+(1*2)^0$

$=512+256+128+64+32+16+0+0+2+1$

$=(1011)_{10}$

Decimal to Binary:

2)156 0
2)78 0
2)39 1
2)19 1
2)9 1
2)4 0
2)2 0
2)1 1
0