Data Representation and Numbering Systems

Number Systems

- A number system is a set of symbols used for counting
- There are various number systems
  - Decimal
  - Binary
  - Octal
  - Hexadecimal

The Decimal Number System

- The Decimal number system is based on the ten different digits (or symbols) 0,1,2,3,4,5,6,7,8,9.
- It is a *base ten number* system
- Though it is widely used, it is inconvenient for computer to represent data.

The Binary number system

- Binary number system is based on the two different digits; 0 and 1.
- We see that the nature of the electronic devices has similarity with the binary number system in that both represent only two elementary states;
- It is therefore convenient to use binary number system to represent data in a computer;
- An “ON” corresponds to a 1;
- An “OFF” corresponds to a 0;
- In the computer “ON” is represented by the existence of a current and “OFF” is represented by non existence of current
- On a magnetic disk, the same information is stored by changing the polarity of magnetized particles on the disk’s surface.

Octal number System (base 8) (Oct)

- It uses 8 symbols 0-7 to represent numbers;
- Like binary number system it is complete number system.
- Example 77 in octal equals 63 in decimal and 111111 in binary.
- When we compare the octal with the decimal, 0-7 in octal is the same as 0-7 in decimal but 10 in octal is not the same as 10 in decimal because 10 in octal holds the position of 8 in decimal.

Hexadecimal number system (16) (hex)

- It uses 16 symbols (0,1,…9,A,B,C,D,E,F) to represent numbers.
- Numbers greater than 15 are represented in terms of the 16 symbols.
- For example the decimal number 16 is represented as 10, 20 as 14, 30 as 1E and so on.
- When we compare hexadecimal with decimal, 0-9 in hexadecimal is the same as 0-9 in decimal but 10 in hexadecimal is not the same as 10 in decimal. 10 in hexadecimal is rather equal to 16 in decimal.

Example

DECIMAL	OCTAL	BINARY	HEXADECIMAL
0	0	0	0
3	3	11	3
8	10	1000	8
10	12	1010	A
16	20	10000	A10

Conversion from base M to decimal

- A number $X_1\ X_2\ X_3\ ...X_n$ in base M can be expanded as:

  $(X_1\ X_2\ X_3\ .....X_n)_M = X_1 * m^{n-1} + X_2 * m^{n-2} X_3 * m^{n-3} + \ ... + X_n m^0$ in base 10

Example

$(1101)_2$ $= 1*2^3+1*2^2+0*2^1+1*2^0 =$

$= (1*8)+(1*4)+(0*2)+(1*1)$

$= 8+4+0+1$

$= 13_{10}$

Conversion from decimal (base 10) to other base (base M)

- To convert a decimal number X to a number in base m, divide X by m, store the remainder, again divide the quotient by M, store the remainder, and continue until the quotient is 0. And concatenate (collect) the remainders starting from the last up to the first.
- **Example**
  - Convert 30_{10} to base sixteen (hexadecimal)

    $30_{10}=1E_{16}$

  - Convert 78_{10} to base eight (Octal)

    $78_{10}=116_{8}$

Binary to Decimal Conversion (the simple way)

- The columns in binary represent:

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
128s	64s	32s	16s	8s	4s	2s	units

- e.g. the binary number

0	0	0	1	0	1	0	1

  is equal to 16 + 4 + 1 = 21 in decimal.

- The number 1110 = 8+4+2 = 14 in decimal

Convert the following binary numbers into decimal based on the following example.

Ex. The number 1110 = 8+4+2 = 14 in decimal

1. 110101
2. 1001
3. 0101
4. 1111
5. 11010
6. 10001

Decimal to binary conversion (the simple way)

- We want to represent the number 83 in binary

128	64	32	16	8	4	2	1
0 (coz 128 don't fit!)							
	1 (64 fits, leaves 19)						
		0					
			1 (leaves 3)				
				0	0		
Gives us						1 (1 left)	
							1
0	1	0	1	0	0	1	1

Conversion from binary (base2) to Octal (base 8)

- To convert a number in binary to octal group three binary digits together starting from the last digit (right) and if there are no enough digits add zeros to the front end (left) and find the corresponding Octal of each group.
- **Example**

Convert 1001001 to octal

1001001=001,001,001

$= 111_8$

Convert 101101001 to octal

101101001 =101,101,001

$=551_8$

Conversion from Octal (base 8) to binary (base2)

- To convert from Octal to binary, convert each octal digit to its equivalent 3 bit binary starting from right.
- **Example**

Convert $(675)_{eight}$ to binary

675_{eight} =110 111 101

$=110111101_{two}$

Convert 231_{eight} to binary

231_{eight} = 010 011 001

$=10011001_{two}$

Conversion from binary (base 2) to hexadecimal (base 16)

- To convert binary to hexadecimal group four binary digits together starting from right and if there are no enough digits add zeros at the left. Then convert each nibble into its corresponding hex value.
- **Example**

Convert 111100100 to hexadecimal

111100100	=0001	1110	0100
	=1	14	4
	=1	E	4

$= 1E4_{16}$

Convert 111001111 to hexadecimal

111001111	=0001	1100	1111
	=1	12	15
	=1	C	F

$=(1CF)_{16}$

Conversion from hexadecimal (base 16) to binary (base 2)

- To convert from Hexadecimal to binary convert each hex. Digit to its equivalent 4-bit binary starting from right.
- **Example**

Convert 2AC to binary

$2AC_{16} = 0010\ \ 1010\ \ 1100$

$= 1010101100_2$

Convert 234_{16} to binary

$234_{16} = 0010\ \ 0011\ 0100$

$= 1000110100_2$

Conversion from Octal to hexadecimal and Vise versa

- To convert from Octal to hexadecimal, the octal number has to be first converted into binary and then to hexadecimal.
- A similar procedure is followed to convert from octal to hex.
- **Example**

Convert 1A to Octal

1A=0001 1010

=000 011 010

=0 3 2

=32_8

Convert 235_8 to hexadecimal

235_8=010 011 101

=0000 1001 1101

= 0 9 13

=$9D_{16}$

Converting decimal number with fractions to binary

- First change the integer part to its equivalent binary.
- Multiply the fractional part by 2 and take out the integer value, and again multiply the fractional part of the result by 2 and take out the integer part, continue this until the product is 0.
- Collect the integer values from top to bottom & concatenate them with the integer part.

Example

$12.25_{10} \Leftrightarrow 1100.01_2$

$3.1875_{10} \Leftrightarrow 11.0011_2$

Converting Binary with fraction to decimal

- To convert a binary number $Y_1Y_2...Y_n.d_1d_2...d_m$ to decimal, first convert the integer part to decimal using the formal conversion method.
- Convert the fractional part to decimal using:

$$d_1d_2d...d_m=d_1*2^{-1}+d_2*2^{-2}+d_3*2^{-3}+..+d_m*2^{-m}$$

- Then decimal equivalence of $y_1\ y_2\ ...y_n.d_1d_2...d_m$ will be Q+R where Q is the binary integer part and R is the binary fractional part.

Example

Convert 11001.0101 to decimal

$11001 = 1x2^4 + 1x2^3 + 0x2^2+0x2^1+1x2^0 = 16+8+1= 25$

$0101 \quad =0x2^{-1}+1x2^{-2}+0x2^{-3}+1x2^{-4}$

= 0+¼+0+1/16 = 0.3125

=>11001.0101 = 25.3125.

Convert 1000.1 to decimal

$1000 = 1+2_3 +0+0+0=8$

$1= 1x2^{-1}$=½ = 0.5

$1000.1 = 8.5_{10}$

Conversion from Binary with fraction to Octal/hex

- Group three/four digits together starting from the last digit of the integer part, and if there is less number of digits add some zeros in the beginning.
- Group three/ four digits together starting from the first digit of the fractional part, and if there is less number of digits add some zeros to the end.
- Covert each group of the integer and the fractional part to their equivalent Octal/hexadecimal.
- **Example**

$1010.0111_2 \Leftrightarrow 12.34_8$

$1110101.10111_2 \Leftrightarrow 75.B8_{16}$

Conversion from Octal or hex with fraction to binary

- Convert each Octal/hexadecimal digit to its equivalent 3/4-bit binary digit.
- Collect the binary sequences by separating the integer part binaries from the fractional part binaries with point (.)
- **Example**

$A3.15_{16} \Leftrightarrow 10100011.00010101_2$

$34.27_8 \Leftrightarrow 011100.010111_2$

- **Conversion from Octal with fraction to hexadecimal**
  - To convert from Octal to hexadecimal, first convert the Octal to binary and then the binary to hexadecimal
- **Conversion from Hexadecimal with fraction to octal**
  - To convert from hexadecimal to octal, first convert the hexadecimal to binary and then the binary to octal.
- **Conversion from octal/hexadecimal with fraction to decimal.**
  - To convert from octal/hexadecimal to decimal, first convert to binary and –then the binary to decimal.
  - You can also convert directly from octal/hexadecimal to decimal just as we did for the conversion from binary to decimal.

Binary addition

- Binary addition operates by the same rule as decimal addition.
- A carry to the next higher order (or more significant) position occurs when the sum is decimal 2, that is, binary 10.
- The binary addition rules in general may be written as follows:

0+0=0

0+1=1

1+0=1

1+1=0 plus a carry of 1

Example

110+111=1101

10011 +11111+1010=111100

Binary Subtraction

- Binary subtraction operates by the same rule as decimal subtraction.
- The rules for subtraction are as follows:

0-0=0

1-0=1

1-1=0

10-1=1

Example

11100	101101	11001.011
- 11010	- 111	- 111.110
00010	100110	10001.101

Binary Multiplication

- It is a very simple process that operates by the following intuitive rules:
  - Multiplying any number by 1 makes the multiplicand unchanged

    0x1=0

    1x1=1
  - Multiplying any number by 0 produces 0

    0x0=0

    1x0=0
- Example

101101 x 1011 = 111101111

Binary division

- It is similar to decimal division
- It simply is the process for dividing one binary number (the dividend) by another (the divisor) and is based on the rules for binary subtraction and multiplication.
- **Example**

  1111101÷ 11001

  11001   101

  11001

  11001

  00000
- Thus, 1111101 ÷ 11001 = 101

Representation of Negative numbers

- There are different ways of representing negative numbers in a computer:
  - **Sign- Magnitude Representation**
  - **One’s Complement Representation**
  - **Two’s Complement Representation**

Sign- Magnitude representation

- In signed binary representation, the left-most bit is used to indicate the sign of the number.
- 0 is used to denote a positive number and
- 1 is used to denote a negative number.
- But the magnitude part will be the same for the negative and positive values
- For example 11111111 represents -127 while, 01111111 represents +127
- In a 5-bit representation, we use the first bit for sign and the remaining 4-bits for the magnitude

Sign- magnitude representation cont'd..

- Using 5 bit representation, the range of numbers that can be represented is from -15 (11111) to 15(01111)
- **Example**

Represent -12 using 5-bit sign magnitude representation

first we convert 12 to binary i. e 1100

Now -12 = 11100

Represent –24 using 8-bits

24 = 00011000

-24 = 10011000

Sign- magnitude representation cont'd..

- In general, for n-bit sign–magnitude representation, the range of values that can be represented is $-(2^{n-1}-1)$ to $(2^{n-1}-1)$.
- In sign magnitude representation zero can be represented as 0 or -0.
- Sign magnitude representation has two problems:
  - It reduces the maximum size of magnitude, and
  - It lacks speed efficiency to perform arithmetic and other operations when implemented in computer hardware. This is because, for sign magnitude representation, addition and subtraction are relatively complex, involving the comparison of signs and relative magnitude of the two numbers

One’s complement

- In one’s complement representation, all positive integers are represented in their correct binary format.
  - Eg. +3 is represented as 00000011 in 8-bit 1’s complement
- Negative numbers are represented by complementing (changing each 0 into 1 and each 1 into 0) their positive equivalent.
  - Eg. -3 is represented as 11111100 in 8-bit
- As in sign magnitude, when the MSB(most significant bit) is 1, it indicates that we have a negative number
- More formally, the 1’s complement of a negative number -N is defined as:

  $N^* = (2^n - 1) - N$

  where: n is the number of bits per word

  N is a positive integer

  N^* is $-N$ in 1's complement notation

One's complement cont'd..

- For example with an 8-bit word and N = 6, we have:

$N*$ = $(2^8 - 1) - 6 = 255 - 6 = 249 = 11111001_2$. That is

11111111

-<u>00000110</u>

11111001. Thus, 11111001 is -6 in 1's complement.

Example: +2 is 00000010

-2 is 11111101

- Note that in this representation, positive numbers start with a 0 on the left, and negative numbers start with a 1 on the left most bit.

One’s complement cont’d..

- Ex1. add –3 and 3 with word size 4

  3= 0011

  -3=1100

  sum =1111 (=0)

- Ex2. Add -4 and +6 with 8-bits

  \- 4 is 11111011

  \+ 6 is 00000110

  the sum is (1) 00000001 the one in the parenthesis is the external carry.

  The correct result should be 2 or 00000010.

- In one’s complement addition and subtraction, if there is an external carry it should be added to get the correct result. This indicates it requires additional circuitry for implementing this operation.

One's complement cont'd..

- The largest number that can be represented in 8-bit 1's complement is 01111111_2 = 127. The smallest is 10000000_2 = -127. Note that the values 00000000_2 and 11111111_2 both represent zero.
- What is the largest and smallest number representations for a 4-bit word? Ans: 7 and -7.
- Use the formula 2^{n-1}- 1 for the maximum and $1-2^{n-1}$ for the minimum(smallest).
- There is no **overflow** as long as the magnitude of the result is not greater than $2^{n-1}-1$.
- One disadvantage of one's complement representation is that there are two representations of zero (i.e. 0000 and 1111, say in 4-bits).
- The other disadvantage is that the end-around carry complicates the addition operation.

Two's Complement Representation

- In two's complement representation, positive numbers are represented just like in one's complement.
- Negative numbers, however, are represented by first computing the one's complement and then adding 1.
- Negating a number (whether negative or positive) is done by inverting all the bits and then adding 1 to that result.
- As in sign magnitude and 1s complement, when the MSB is 1, it indicates that we have a negative number.

Two's Complement Representation cont'd..

- The reason 2's complement was introduced is so as to ignore the external carry that results during the addition process of one's complement.
- Signed integer values are usually stored on the computer in 2s complement form.
  - Ex: +3 is represented in signed binary as 00000011
  - Its one's complement representation is 11111100.
  - The two's complement is obtained by adding one. It is 11111101.
- Formally, the 2's complement of a negative number N is defined as:

  $N^* = 2^n - N$

  where: *n* is the number of bits per word

  N is a positive integer

  *N** is -*N* in 2's complement notation

Two's Complement Representation cont'd..

- For example with an 8-bit word and $N = 6$, we have:

  $N^* = 2^8 - 6 = 256 - 6 = 250 = 11111010$, that is

$$\begin{array}{r} 100000000 \\ -\underline{\quad\quad 110} \\ 11111010 \end{array}$$

- An alternate way to find the 2's complement is to start at the right and complement each bit to the left of the first "1".
  - For example: $N = +6 = 00000110_2$, $N^* = -6 = 11111010_2$
- Conversely, given the 2's complement we can find the magnitude of the number by taking it's 2's complement.

Two's Complement Representation cont'd..

- The largest number that can be represented in 8-bit 2's complement is 01111111_2 = 127. The smallest is 10000000_2 = -128.
- For an n digit in 2's complement, maximum number is 2^{n-1}-1 and the minimum is -2^{n-1}.
- Ex let's try addition.

  (3) 00000011

  \+ (5) 00000101

  (8) 0001000

- Ex2. Let's try subtraction

  (3) 00000011

  (-5) 111111011

  11111110

You can convert the result to 2's complement to get the magnitude of the number. It becomes 00000010.

Two's Complement Representation cont'd..

- Ex2: add +4 and -3(the subtraction is performed by adding the two's complement).

  +4 is 00000100

  -3 is 11111101

  The result is [1-the carry] 00000001

- If we ignore the external carry the result is 00000001 (i. e 1 in decimal). This is the correct result.

Two's Complement Representation cont'd..

- In two's complement, it is possible to add or subtract signed numbers, regardless of the sign.
- Using the usual rules of binary addition, the result comes out correct, including the sign.
- The carry is ignored. One's complement may be used, but if one's complement is used, special circuitry is required to "correct the result".
- When the addition of two values results in a carry, the carry bit is ignored. There is no **overflow** as long as the result is not greater than $2^{n-1}-1$ nor less than -2^{n-1}.

Two's Complement Representation cont'd..

- Find 12-10 = 12 + -10

```
  00001100
+11110110
  00000010
```

- E.g. using 8-bit add 10 and 12.

```
  00001010
+00001100
  00010110
```

- Find -12 + 10.

```
  11110100
+00001010
  11111110
```

- 11111100 is a negative number. Determine it's magnitude by finding it's 2's complement. We have 00000010 which is 2.

Two's Complement Representation cont'd..

- **Overflow**: this is an example of an overflow condition in 2's complement.
- Example

(100) 01100100

+(30) 00011110

10000010

- However, the result represents not 130 but -126. That means, because 130 is represented by 8 bits, it requires at least 9 bits to hold the sign of the result.

Two’s Complement Representation cont’d..

- The two’s complement representation has one anomaly not found with sign magnitude or one’s complement. The bit pattern 1 followed by N-1 zeros is its own 2’s complement.
- For example, for 8-bit word,

  -128 = 10000000

  its 1’s complement =01111111

  +1

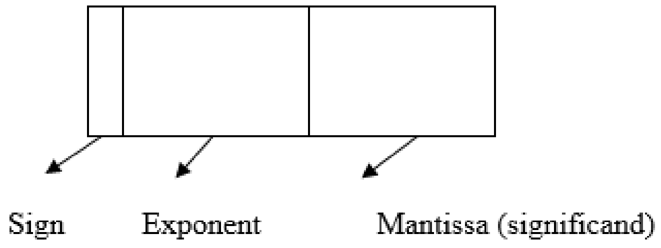
  =100000000
- To maintain sign bit consistency, this bit pattern is assigned the value $–2^{N-1}$.
- Thus, in 2’s complement 8-bit, 10000000 represents -128.

Floating-Point Representation

- In this representation, decimal numbers are represented with a fixed length format
- To avoid wastage of bits, this representation normalizes all the numbers.
- For example, 0.000123 wastes three zeroes on the left before non -zero digits. Normalizing this number result in $.123 \times 10^{-3}$,
    - .123 is the normalized mantissa;
    - -3 is the exponent.
- The general form of floating point representation is $\pm M \times 10^{\pm E}$ where M is the mantissa, and E is the exponent.
-

Floating-Point Representation cont'd..

- To represent floating numbers in the computer system it should be normalized after converting to binary number representation system.
- Ex2 111.01 is normalized as $.11101 \times 10^3$.
  - The mantissa is 11101. The exponent is 3.
- The general structure of floating point is:



- In representing a number in floating point we use 1 bit for sign, some bits for exponent and the remaining bit for mantissa.

Floating-Point Representation cont'd..

- In floating point representation, the exponent is represented by a biased exponent.
- (Biased exponent)= (true exponent) + (2^{n-1}), where n is the number of bits reserved for the exponent. The biasing exponent representation is called excess 2^{n-1}.
- <u>Example</u>. Represent –226.375 in floating point using 7 bit for exponent and 16 bit for mantissa.
  - First we have to change to normalized binary (i.e 226 = 11100010 and 0.375= 0.011)
  - 226.375 = 11100010.011 = 0.11100010011×2^8
  - true exponent = 8
  - excess $2^{n-1} = 2^{7-1} = 2^6 = 64$
  - Biased exponent = 8+64 = 72 = $100\,1000_2$
  - Therefore –234.375 is represented as:

1	1001000	1110001001100000

Floating-Point Representation cont'd..

- Example. Given the number represented in a floating point representation

0	1000110	100010010000000000000000

- what is the number in decimal form?
- The exponent 1000110_2 is 70. Since it is 7 bits, the biasing is 64; hence the true exponent is 70-64=6. The mantissa is 0.10001001_2. Thus, $0.10001001_2 \times 2^6 = 100010.01_2$. Converting the number into decimal, we get 34.25.

Floating-point Arithmetic

- To perform floating-point arithmetic:
  - First correct the numbers to binary with the same exponent
  - Apply the operator on the mantissa and
  - Normalize the result
- Example.
  - Find 123456.375+ 101.25 using 7-bit for exponent and 24 bits for mantissa.
  - 123456.375 =11,110,001,001,000,000.011=0.11110001001000000011x2^{17}.
  - 101.25 =1,100,101.01 = 0.110010101 x2^{7} = 0.0000000000110010101 x2^{17}.
- Adding the mantissas gives,
  - 0.11110001001000000011 + 0.0000000000110010101= 0.11110001010100101101.
- The final number becomes, 0.11110001010100101101 x 2^{17}.

Coding Methods in computer

- It is possible to represent any of the characters in our language as a series of electrical switches (transistors);
- These switch arrangements can therefore be coded as a series of an equivalent arrangements of bits
- There are different coding systems, that convert one or more character sets into computer codes. Some are: EBCDIC, ASCII-7, ASCII-8 & Unicode.
- In all cases, binary coding schemes separate the characters, known as character set, in to zones.
- A zone groups characters together so as to make the data easier to process by computers.
- With in each zone the individual characters are identified by digit code.

EBCDIC

- Pronounced as "Eb-see-dick" and stands for Extended Binary Coded Decimal Interchange Code.
- Proprietary specification developed by IBM
- It is an 8-bit coding scheme; (00000000 to 11111111)
- It accommodates to code 2^8 or 256 different characters
- It is a standard coding scheme for mainframe computers of IBM.
- Coding Examples

Character	**Zone(4BIT)**	**Digit(4 BIT)**
0-9	15	0-9
a-I	8	1-9
j-r	9	1-9
s-z	10	2-9
A-I	12	1-9

EBCDIC cont'd..

- Coding Examples

Character	**Zone**	**Digit**
a	1000	0001
b	1000	0010
A	1100	0001
B	1100	0010
0	1111	0000
9	1111	1001

EBCDIC

Coding of Alphabetic and Numeric Characters in EBCDIC

Char	EBCDIC Code		Hex
	Digit	Zone	
A	1100	0001	C1
B	1100	0010	C2
C	1100	0011	C3
D	1100	0100	C4
E	1100	0101	C5
F	1100	0110	C6
G	1100	0111	C7
H	1100	1000	C8
I	1100	1001	C9
J	1101	0001	D1
K	1101	0010	D2
L	1101	0011	D3
M	1101	0100	D4

Char	EBCDIC Code		Hex
	Digit	Zone	
N	1101	0101	D5
O	1101	0110	D6
P	1101	0111	D7
Q	1101	1000	D8
R	1101	1001	D9
S	1110	0010	E2
T	1110	0011	E3
U	1110	0100	E4
V	1110	0101	E5
W	1110	0110	E6
X	1110	0111	E7
Y	1110	1000	E8
Z	1110	1001	E9

(Continued on next sli

Character	EBCDIC Code		Hexadecima l Equivalent
	Digit	Zone	
0	1111	0000	F0
1	1111	0001	F1
2	1111	0010	F2
3	1111	0011	F3
4	1111	0100	F4
5	1111	0101	F5
6	1111	0110	F6
7	1111	0111	F7
8	1111	1000	F8
9	1111	1001	F9

EBCDIC

EBCDIC Coding Scheme

Example

Using binary notation, write EBCDIC coding for the word BIT. How many bytes are required for this representation?

Solution:

B = 1100 0010 in EBCDIC binary notation
I = 1100 1001 in EBCDIC binary notation
T = 1110 0011 in EBCDIC binary notation

Hence, EBCDIC coding for the word BIT in binary notation will be

<u>11000010</u>	<u>11001001</u>	<u>11100011</u>
B	I	T

3 bytes will be required for this representation because each letter requires 1 byte (or 8 bits)

EBCDIC

00	NUL	20	DS	40	SP	60	-	80		A0		C0	{	E0	\
01	SOH	21	SOS	41		61	/	81	a	A1	~	C1	A	E1	
02	STX	22	FS	42		62		82	b	A2	s	C2	B	E2	S
03	ETX	23		43		63		83	c	A3	t	C3	C	E3	T
04	PF	24	BYP	44		64		84	d	A4	u	C4	D	E4	U
05	HT	25	LF	45		65		85	e	A5	v	C5	E	E5	V
06	LC	26	ETB	46		66		86	f	A6	w	C6	F	E6	W
07	DEL	27	ESC	47		67		87	g	A7	x	C7	G	E7	X
08		28		48		68		88	h	A8	y	C8	H	E8	Y
09		29		49		69		89	i	A9	z	C9	I	E9	Z
0A	SMM	2A	SM	4A	¢	6A	¦	8A		AA		CA		EA	
0B	VT	2B	CU2	4B	.	6B	,	8B		AB		CB		EB	
0C	FF	2C		4C	<	6C	%	8C		AC		CC		EC	
0D	CR	2D	ENQ	4D	(	6D	_	8D		AD		CD		ED	
0E	SO	2E	ACK	4E	+	6E	>	8E		AE		CE		EE	
0F	SI	2F	BEL	4F	|	6F	?	8F		AF		CF		EF	
10	DLE	30		50	&	70		90		B0		D0	}	F0	0
11	DC1	31		51		71		91	j	B1		D1	J	F1	1
12	DC2	32	SYN	52		72		92	k	B2		D2	K	F2	2
13	TM	33		53		73		93	l	B3		D3	L	F3	3
14	RES	34	PN	54		74		94	m	B4		D4	M	F4	4
15	NL	35	RS	55		75		95	n	B5		D5	N	F5	5
16	BS	36	UC	56		76		96	o	B6		D6	O	F6	6
17	IL	37	EOT	57		77		97	p	B7		D7	P	F7	7
18	CAN	38		58		78		98	q	B8		D8	Q	F8	8
19	EM	39		59		79		99	r	B9		D9	R	F9	9
1A	CC	3A		5A	!	7A	:	9A		BA		DA		FA	|
1B	CU1	3B	CU3	5B	$	7B	#	9B		BB		DB		FB	
1C	IFS	3C	DC4	5C	*	7C	@	9C		BC		DC		FC	
1D	IGS	3D	NAK	5D	)	7D	'	9D		BD		DD		FD	
1E	IRS	3E		5E	;	7E	=	9E		BE		DE		FE	
1F	IUS	3F	SUB	5F	¬	7F	"	9F		BF		DF		FF	

STX	Start of text	RS	Reader Stop	DC1	Device Control 1	BEL	Bell
DLE	Data Link Escape	PF	Punch Off	DC2	Device Control 2	SP	Space
BS	Backspace	DS	Digit Select	DC4	Device Control 4	IL	Idle
ACK	Acknowledge	PN	Punch On	CU1	Customer Use 1	NUL	Null
SOH	Start of Heading	SM	Set Mode	CU2	Customer Use 2		
ENQ	Enquiry	LC	Lower Case	CU3	Customer Use 3		
ESC	Escape	CC	Cursor Control	SYN	Synchronous Idle		
BYP	Bypass	CR	Carriage Return	IFS	Interchange File Separator		
CAN	Cancel	EM	End of Medium	EOT	End of Transmission		
RES	Restore	FF	Form Feed	ETB	End of Transmission Block		
SI	Shift In	TM	Tape Mark	NAK	Negative Acknowledge		
SO	Shift Out	UC	Upper Case	SMM	Start of Manual Message		
DEL	Delete	FS	Field Separator	SOS	Start of Significance		
SUB	Substitute	HT	Horizontal Tab	IGS	Interchange Group Separator		
NL	New Line	VT	Vertical Tab	IRS	Interchange Record Separator		
LF	Line Feed	UC	Upper Case	IUS	Interchange Unit Separator		

ASCII

- ASCII stands for American Standard Code for Information Interchange
- ASCII is of two types: ASCII-7 and ASCII-8
- ASCII-7 Used widely before the introduction of ASCII-8 (the Extended ASCII)
- ASCII-7 Uses 7 bits to represent a character
- With the seven bits, 2^7(or 128) different characters can be coded (0000000-1111111)
- It has a zone and digit bits positions
- Coding examples:

Character	**zone (3 BIT)**	**digit(4 BIT)**
0-9	3	0-9
A-O	4	1-15
P-Z	5	0-10

ASCII

Coding of Numeric and Alphabetic Characters in ASCII

Coding of Numeric and Alphabetic Characters in ASCII

ntinued from previous slide..)

Character	ASCII-7 / ASCII-8		Hexadecimal Equivalent
	Zone	Digit	
0	0011	0000	30
1	0011	0001	31
2	0011	0010	32
3	0011	0011	33
4	0011	0100	34
5	0011	0101	35
6	0011	0110	36
7	0011	0111	37
8	0011	1000	38
9	0011	1001	39

Character	ASCII-7 / ASCII-8		Hexadecimal Equivalent
	Zone	Digit	
A	0100	0001	41
B	0100	0010	42
C	0100	0011	43
D	0100	0100	44
E	0100	0101	45
F	0100	0110	46
G	0100	0111	47
H	0100	1000	48
I	0100	1001	49
J	0100	1010	4A
K	0100	1011	4B
L	0100	1100	4C
M	0100	1101	4D

Character	ASCII-7 / ASCII-8		Hexadecimal Equivalent
	Zone	Digit	
N	0100	1110	4E
O	0100	1111	4F
P	0101	0000	50
Q	0101	0001	51
R	0101	0010	52
S	0101	0011	53
T	0101	0100	54
U	0101	0101	55
V	0101	0110	56
W	0101	0111	57
X	0101	1000	58
Y	0101	1001	59
Z	0101	1010	5A

ASCII-7 *cont'd..*

- Coding examples:

Character	**Zone**	**Digit**
$	010	0100
%	010	0101
A	100	0001
a	110	0001
b	110	0010

ASCII-7 *cont'd..*

ASCII-7 Coding Scheme

Example

Write binary coding for the word BOY in ASCII-7. How many bytes are required for this representation?

Solution:

B = 1000010 in ASCII-7 binary notation
O = 1001111 in ASCII-7 binary notation
Y = 1011001 in ASCII-7 binary notation

Hence, binary coding for the word BOY in ASCII-7 will be

1000010	1001111	1011001
B	O	Y

Since each character in ASCII-7 requires one byte for its representation and there are 3 characters in the word BOY, 3 bytes will be required for this representation

The ASCII System

- Also referred as ASCII-8 or Extended ASCII
- It is the most widely used type of coding scheme for microcomputer systems
- ASCII uses 8-bits to represent alphanumeric characters(letters, digits and special symbols).
- With the 8-bits, ASCII can represent 2^8 or 256 different characters(00000000-11111111).
- Coding Example

Character	Binary representation in ASCII
a	01100001
b	01100010
A	01000001
B	01000010
?	00111111
+	00101011
1	00110001
2	00110010
3	00110011

ASCII

Decimal	Hex	Char.	Comment	Decimal	Hex	Char.	Decimal	Hex	Char.	Decimal	Hex	Char.
0	00	NUL	Null	32	20	Space	64	40	@	96	60	`
1	01	SOH	Start of Heading	33	21	!	65	41	A	97	61	a
2	02	STX	Start of Text	34	22	"	66	42	B	98	62	b
3	03	ETX	End of Text	35	23	#	67	43	C	99	663	c
4	04	EOT	End of Transmission	36	24	$	68	44	D	100	64	d
5	05	ENQ	Enquiry	37	25	%	69	45	E	101	65	e
6	06	ACK	Acknowledge	38	26	&	70	46	F	102	66	f
7	07	BEL	Bell (Ding!)	39	27	'	71	47	G	103	67	g
8	08	BS	Backspace	40	28	(	72	48	H	104	68	h
9	09	HT	Horizontal Tab	41	29	)	73	49	I	105	69	i
10	0A	LF	Line Feed	42	2A	*	74	4A	J	106	6A	j
11	0B	VT	Vertical Tab	43	2B	+	75	4B	K	107	6B	k
12	0C	FF	Form Feed (new page)	44	2C	,	76	4C	L	108	6C	l
13	0D	CR	Carraige Return	45	2D	-	77	4D	M	109	6D	m
14	0E	SO	Shift Out	46	2E	.	78	4E	N	110	6E	n
15	0F	SI	Shift In	47	2F	/	79	4F	O	111	6F	o
16	10	DLE	Data Link Escape	48	30	0	80	50	P	112	70	p
17	11	DC1	Device Control 1	49	31	1	81	51	Q	113	71	q
18	12	DC2	Device Control 2	50	32	2	82	52	R	114	72	r
19	13	DC3	Device Control 3	51	33	3	83	53	S	115	73	s
20	14	DC4	Device Control 4	52	34	4	84	54	T	116	74	t
21	15	NAK	Negative Acknowledge	53	35	5	85	55	U	117	75	u
22	16	SYN	Synchronous Idle	54	36	6	86	56	V	118	76	v
23	17	ETB	End of Transmission Block	55	37	7	87	57	W	119	77	w
24	18	CAN	Cancel	56	38	8	88	58	X	120	78	x
25	19	EM	End of Medium	57	39	9	89	59	Y	121	79	y
26	1A	SUB	Substitute	58	3A	:	90	5A	Z	122	7A	z
27	1B	ESC	Escape	59	3B	;	91	5B	[	123	7B	{
28	1C	FS	File Separator	60	3C	<	92	5C	\	124	7C	|
29	1D	GS	Group Separator	61	3D	=	93	5D	]	125	7D	}
30	1E	RS	Record Separator	62	3E	>	94	5E	^	126	7E	~
31	1F	US	Unit Separator	63	3F	?	95	5F	_	127	7F	DEL (Delete)

The ASCII System

ASCII-8 Coding Scheme

Example

Write binary coding for the word SKY in ASCII-8. How many bytes are required for this representation?

Solution:

S = 01010011 in ASCII-8 binary notation
K = 01001011 in ASCII-8 binary notation
Y = 01011001 in ASCII-8 binary notation

Hence, binary coding for the word SKY in ASCII-8 will be

01010011	01001011	01011001
S	K	Y

Since each character in ASCII-8 requires one byte for its representation and there are 3 characters in the word SKY, 3 bytes will be required for this representation

Unicode

- Unicode is a computing industry standard allowing computers to consistently represent and manipulate text expressed in most of the world's writing systems.
- Unicode consists of a collection of more than 100,000 characters.
- Unicode can be implemented by different character encodings.
- The most commonly used encodings are UTF-8 which uses 1 byte for all ASCII characters, which have the same code values as in the standard ASCII encoding, and up to 4 bytes for other characters

Unicode

Unicode

- **Why Unicode:**
  - No single encoding system supports all languages
  - Different encoding systems conflict
- **Unicode features:**
  - Provides a consistent way of encoding multilingual plain text
  - Defines codes for characters used in all major languages of the world
  - Defines codes for special characters, mathematical symbols, technical symbols, and diacritics

Unicode cont'd..

- Let's consider how Ethiopia's character sets are represented
- The character set is called Ethiopic
- Range: 1200-1378 (in hexadecimal)
- Example character sets

Syllables

1200	ሀ	ETHIOPIC SYLLABLE HA
1201	ሁ	ETHIOPIC SYLLABLE HU
1202	ሂ	ETHIOPIC SYLLABLE HI
1203	ሃ	ETHIOPIC SYLLABLE HAA
1204	ሄ	ETHIOPIC SYLLABLE HEE
1205	ህ	ETHIOPIC SYLLABLE HE
1206	ሆ	ETHIOPIC SYLLABLE HO
1207	ሇ	ETHIOPIC SYLLABLE HOA
1208	ለ	ETHIOPIC SYLLABLE LA
1209	ሉ	ETHIOPIC SYLLABLE LU
120A	ሊ	ETHIOPIC SYLLABLE LI
120B	ላ	ETHIOPIC SYLLABLE LAA
120C	ሌ	ETHIOPIC SYLLABLE LEE
120D	ል	ETHIOPIC SYLLABLE LE
120E	ሎ	ETHIOPIC SYLLABLE LO
120F	ሏ	ETHIOPIC SYLLABLE LWA
1210	ሐ	ETHIOPIC SYLLABLE HHA
1211	ሑ	ETHIOPIC SYLLABLE HHU
1212	ሒ	ETHIOPIC SYLLABLE HHI
1213	ሓ	ETHIOPIC SYLLABLE HHAA
1214	ሔ	ETHIOPIC SYLLABLE HHEE
1215	ሕ	ETHIOPIC SYLLABLE HHE
1216	ሖ	ETHIOPIC SYLLABLE HHO
1217	ሗ	ETHIOPIC SYLLABLE HHWA
1218	መ	ETHIOPIC SYLLABLE MA
1219	ሙ	ETHIOPIC SYLLABLE MU

1242	ቂ	ETHIOPIC SYLLABLE QI
1243	ቃ	ETHIOPIC SYLLABLE QAA
1244	ቄ	ETHIOPIC SYLLABLE QEE
1245	ቅ	ETHIOPIC SYLLABLE QE
1246	ቆ	ETHIOPIC SYLLABLE QO
1247	ቇ	ETHIOPIC SYLLABLE QOA
1248	ቈ	ETHIOPIC SYLLABLE QWA
1249		<reserved>
124A	ቊ	ETHIOPIC SYLLABLE QWI
124B	ቋ	ETHIOPIC SYLLABLE QWAA
124C	ቌ	ETHIOPIC SYLLABLE QWEE
124D	ቍ	ETHIOPIC SYLLABLE QWE
124E		<reserved>
124F		<reserved>
1250	ቐ	ETHIOPIC SYLLABLE QHA
1251	ቑ	ETHIOPIC SYLLABLE QHU
1252	ቒ	ETHIOPIC SYLLABLE QHI
1253	ቓ	ETHIOPIC SYLLABLE QHAA
1254	ቔ	ETHIOPIC SYLLABLE QHEE
1255	ቕ	ETHIOPIC SYLLABLE QHE
1256	ቖ	ETHIOPIC SYLLABLE QHO
1257		<reserved>
1258	ቘ	ETHIOPIC SYLLABLE QHWA
1259		<reserved>
125A	ቚ	ETHIOPIC SYLLABLE QHWI
125B	ቛ	ETHIOPIC SYLLABLE QHWAA
125C	ቜ	ETHIOPIC SYLLABLE QHWEE
125D	ቝ	ETHIOPIC SYLLABLE QHWE