

CHAPTER 2: Data Representations

DATA REPRESENTATIONS

- In the we examine the several formats a computer uses to store information it is to operate on.
- But because computers are binary in nature, all formats must be patterns of 1's and 0's.

DATA REPRESENTATIONS

Decimal System:

- In everyday life we use a system based on decimal digits (0, 1, 2, 3, 4, 5, 6, 7, 8, 9) to
- represent numbers.
- This system is referred to as the decimal system.
- In the decimal system, 10 different digits are used to represent numbers with a base of 10.

DATA REPRESENTATIONS

Binary System:

- In the binary system, we have only two digits, 1 and 0.
- Thus, numbers in the binary system are represented to the base 2.
- A computer uses binary digits (0's and 1's) to store data in different formats.
- These binary digits are called BITS.
- In Computer circuits, 0's and 1's are voltage levels:
 - 0 is low voltage (OFF)
 - 1 is high Voltage (ON)

DATA FORMATS

1. Numeric Formats

They store only numbers; There are three numeric formats:

- *Integer or Fixed point Formats.*
- *Floating Point Formats*
- *Binary Coded decimal (BCD)*

2. Alphanumeric Codes:

Store both numbers and characters including the alphabetic characters.

NUMERIC FORMATS

Integer Formats

Binary:

➤ To convert from decimal to binary we do successive divisions

➤ To convert from binary to decimal we expand
e.g. 11011

$$\begin{aligned} &= (1 * 2^4) + (1 * 2^3) + (0 * 2^2) + (1 * 2^1) + (1 * 2^0) \\ &= 16 + 8 + 0 + 2 + 1 = 27_{10}. \end{aligned}$$

or use Horner's Rule:

$$((((1 * 2) + 1) * 2 + 0) * 2 + 1) * 2 + 1 = 27_{10}$$

NUMERIC FORMATS

- All forms of data within computers are represented by various binary codes.
- The Binary system is convenient for computers but not for humans beings.
- Therefore, computer professionals who must spend time working with the actual raw data in the computer prefer a more compact notation.
- Hence the adoption of the **hexadecimal notation**.
- In hexadecimal notation, binary digits are grouped into sets of four.
- Each possible combination of four binary digits is given a symbol.

NUMERIC FORMATS

HEXADECIMAL

- It has 16 digits 0 – 15
- Each Hexadecimal digit can be represented by a unique combination of 4 binary bits; e.g.

$$\begin{array}{ccccccc} = > & 110011011 & = & 0001 & 1001 & 1011 & \\ & & & 1 & 9 & & B_{16} \end{array}$$

- To convert $1C2E_{16}$ to decimal you expand using powers of 16.

$$\begin{aligned} &= (1 * 16^3) + (12 * 16^2) + (2 * 16^1) + (14 * 16^0) \\ &= 4096 + 3072 + 32 + 14 = 7214 \end{aligned}$$

- To convert 15797_{10} to hexadecimal we perform successive divisions by 16.

NUMERIC FORMATS

OCTAL

- Each octal digit can be represented by a unique combination of three bits.

$$\text{e.g. } 110011011_2 = 110 \ 011 \ 011_2 = 633_8$$

- $101011000110_2 = 101 \ 011 \ 000 \ 110 = 5306_8$
 $= 1010 \ 1100 \ 0110 = AC6_{16}$

- $1573_8 = 001 \ 101 \ 111 \ 011_2$
 $= 0011 \ 0111 \ 1011 = 37B_{16}$

- $A748_{16} = 1010 \ 0111 \ 0100 \ 1000_2 =$
 $= 001 \ 010 \ 011 \ 101 \ 001 \ 000 = 123510_8$

Decimal	Hexadecimal	Binary	Octal
0	0	0000	0
1	1	0001	1
2	2	0010	2
3	3	0011	3
4	4	0100	4
5	5	0101	5
6	6	0110	6
7	7	0111	7
8	8	1000	10
9	9	1001	11
10	A	1010	12
11	B	1011	13
12	C	1100	14
13	D	1101	15
14	E	1110	16
15	F	1111	17
16	10	10000	20

NUMERIC FORMATS: (Practice Questions)

- Perform the following Conversions:
 - a) 110011011_2 to Decimal.
 - b) 27_{10} to Binary
 - c) 110011011_2 to Hexadecimal.
 - d) $1C2E_{16}$ to decimal.
 - e) 15797_{10} to hexadecimal
 - f) 101011000110_2 to octal.
 - g) 1573_8 to hexadecimal.
 - h) $A748_{16}$ to octal.

NUMERIC FORMATS

FRACTIONS

- The binary equivalent of the integer portion is obtained as usual but the fraction part is obtained by successively multiplying by 2.

e.g. To convert 13.6875 to binary;

$$13_{10} = 1101_2$$

$$.6875 * 2 = 1.375$$

$$.375 * 2 = 0.75$$

$$.75 * 2 = 1.5$$

$$.5 * 2 = 1.0$$

1 0 1 1



$$\Rightarrow 13.6875 = 1101.1011$$

NUMERIC FORMATS

FRACTIONS

- If a binary number contains digits to the right of the decimal point we convert them **by starting at the binary point** and move to the right.

$$\begin{aligned}\text{e.g. } 11.1010011011_2 &= 011 \ 101 \ 001 \ 101 \ 100 = 3.5154_8 \\ &= 0011 \ 1010 \ 0110 \ 1100 = 3.A6C_{16}\end{aligned}$$

$$5.145_8 = 101.001 \ 100 \ 101 = 0101.0011 \ 0010 \ 1000 = 5.328_{16}$$

- Just like in Base 10 the decimal point can be moved by multiplying by the appropriate power of the base.

$$\text{e.g. } 101.11 = 10111 * 2^{-2} = 0.10111 * 2^3$$

Binary Arithmetic

$$\begin{array}{r} 110101 \\ + 10010 \\ \hline 1000111 \end{array}$$

$$\begin{array}{r} 101101 \\ - 100110 \\ \hline 111 \end{array}$$

$$\begin{array}{r} 10110 \\ * 1011 \\ \hline \end{array}$$

$$11111101 / 1011$$

$$\begin{array}{r} BA41 \\ + 14AF \\ \hline CEF0 \end{array}$$

$$\begin{array}{r} BA41 \\ - 14AF \\ \hline A592 \end{array}$$

REPRESENTATION OF NUMBERS IN COMPUTERS

- The storage capacity of a computer's memory and control circuitry is finite
- If there are n bits in a group the number of possible combinations of 0's and 1's is 2^n .
- If the bits are used to represent non-negative integers, integers 0 through $2^n - 1$ can be represented. E.g With 8 bits integers 0 – 255 can be represented.
- Quite often the groups are large and n may be 24,32, 64 etc.

REPRESENTATION OF NUMBERS IN COMPUTERS

- Big numbers are estimated using powers of ten
 $2^{10} = 1024 \approx 10^3$

$$\text{e.g. } 2^{36} = 2^6 \cdot 2^{30} = 2^6 (2^{10})^3 = 2^6 (10^3)^3 = 64 * 10^9$$

Overflows

- If the result of any operation does not fit into the number of bits reserved for it an **overflow** is said to occur.
- All the 4 arithmetic operations can cause an overflow.

$$360 + 720 - 300 = 360 + (720 - 300) \text{ and} \\ (360 + 720) - 300$$

SIGNED INTEGERS

There are 2 ways of representing signed integers:
i.e Sign Magnitude Format and Complement format.

- In the **Sign Magnitude Format** the leftmost bit/MSB represents the sign.
- A negative number is represented by a 1 and a positive number by a 0.
- The range of integers that can be expressed in a group of 8 bits is from $-(2^7 - 1)$ to $(2^7 - 1)$
- In general a d bit binary sign magnitude representation in which the first bit represents the sign has a range of $-(2^{d-1} - 1)$ to $+(2^{d-1} - 1)$.

SIGNED INTEGERS

- To add two sign magnitude numbers, we follow the usual addition rule
- If the sign differs we subtract the smaller number from the larger number and give the result the sign of the larger number.
- If the signs are the same we add them and we give the result the same sign.

SIGNED INTEGERS

$$\begin{array}{r} +5 + -7 = 10000111 \\ - \quad \underline{00000101} \\ 10000010 \quad (-2) \end{array}$$

$$\begin{array}{r} -5 + -7 = 10000101 \\ + \quad \underline{10000111} \\ 10001100 \quad (-12) \end{array}$$

SIGNED INTEGERS

2's Complement

- The d digit 2's complement of a d bit binary integer N is equal to $2^d - N$ where the subtraction is done in binary.

- The eight bit 2's complement of an 8 bit binary number 00000101 is $100000000 - 00000101 = 11111011$

- The 2's complement may also be computed by applying the following rules.

Invert the bit values; the result is called **one's complement** then add 1

e.g. for $N = 00000101$ *Invert* $11111010 + 1$
 $= 11111011$

2's Complement

$$17_{10} = 00010001 \quad \text{Invert the bits and add 1}$$
$$11101110 + 1 = 11101111 = -17$$

$$119_{10} = 00010001 \quad \text{Invert the bits and add 1;}$$
$$10001000 + 1 = 10001001 = -119$$

- N.B. Note the difference between the sign magnitude representation and the 2's complement representation.

Rules to convert to decimal

- If a number is positive (beginning with a 0), convert it to base 10 directly as usual.
- If it is negative (begins with 1) get its 2's complement and convert it to base 10.
- e.g. to convert a 2's complement number 11111001 to decimal:
- Get its complement ; i.e. 11111001 *Invert* 00000110 + 1 = 00000111 = - 7

Addition of 2's Complement Binary Integers

The left most bits are not treated separately as in signed magnitude numbers.

$$\begin{array}{r} 7 \\ + 5 \\ \hline 12 \end{array}$$

$$\begin{array}{r} 00000111 \\ 00000101 \\ \hline 00001100 \end{array}$$

(b)

$$\begin{array}{r} -7 \\ + 5 \\ \hline -2 \end{array}$$

$$\begin{array}{r} 11111001 \\ 00000101 \\ \hline 11111110 \end{array}$$

(c)

$$\begin{array}{r} -7 \\ + - 5 \\ \hline -12 \end{array}$$

$$\begin{array}{r} 11111001 \\ 11111011 \\ \hline 1\ 11110100 \end{array}$$

(d)

$$\begin{array}{r} 7 \\ + - 5 \\ \hline 2 \end{array}$$

$$\begin{array}{r} 00000111 \\ 11111011 \\ \hline 1\ 00000010 \end{array}$$

The carry is discarded

Overflow in 2's complement

An overflow in 2's complement occurs when:

- Adding two positive numbers produces a negative result, or when adding two negative numbers produces a positive result. Adding operands of unlike signs never produces an overflow.
- Notice that discarding the carry out of the most significant bit during Two's Complement addition is a normal occurrence, and does not by itself indicate overflow.

e.g. 126	01111110	-126	10000010
+ 5	<u>00000101</u>	+ - 5	<u>11111011</u>
131	10000011	-131	01111101

FLOATING POINT FORMATS

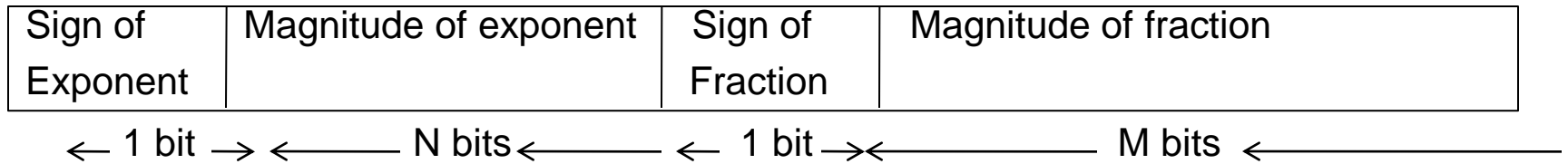
- It stores numbers given in scientific notation. It is often used to handle very large and very small numbers.
- It is written in the form: **Fraction * base^{exponent}**
e.g. $0.000000357 = 0.357 * 10^{-6}$,
 $625000000 = 0.625 * 10^9$
- The fraction part is called the **significand** and the exponent the **characteristic**.

Floating Point Numbers

A floating point format is designated by:

- The base
 - The number of bits reserved for the exponent
 - The number of bits reserved for the fraction
 - The method for storing the sign and magnitude of the exponent
 - The method for storing the sign and magnitude of the fraction.
 - The order in which the two signs and the two magnitudes are to occur.
- The combination of the above factors for a given computer depends upon the designer.

A Typical Floating Point Format



- The base chosen never appears in the format once chosen it is fixed.
- The total number of bits in the FPF is $N + M + 2$ where N = bits are reserved for the magnitude of the exponent and M = bits reserved for the magnitude of the fraction.

A Typical Floating Point Format

- If base 2 is assumed the largest number that can be stored using a floating point format is

approximately $2^{2^N - 1}$ where N = number of bits reserved for the exponent.

- If $N = 7$ the largest number is approximately $2^{127} = 2^7(2^{10})^{12} = 128 * 10^{36} = 10^{38}$ and the smallest is $2^{-126} = 10^{-38}$

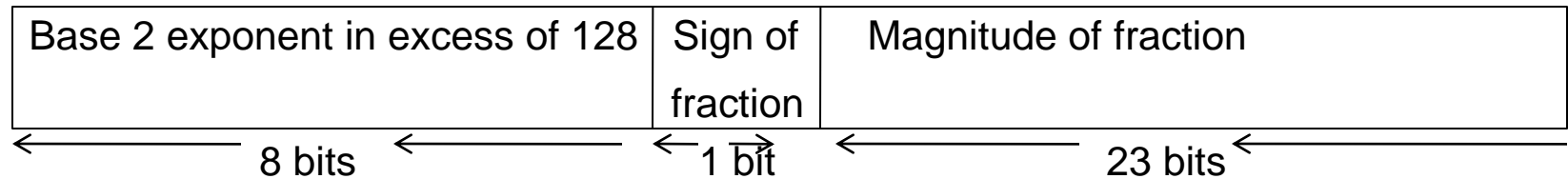
A Typical Floating Point Format

- **An exponent overflow** is said to occur if an operation results in a number that is so large that the maximum size of the exponent is exceeded.
- If the exponent is negative and the magnitude becomes too large, then an **exponent underflow** occurs.

Floating Point Numbers

- If $N + 1$ bits are reserved for the exponent and the sign, the offset or bias chosen = 2^N and the format is called the **excess 2^N format**
- If $N = 7$ the offset = $2^7 = 128 = 10000000_2$ and the format is called the **excess 128 format**.
- The real exponent is obtained from the quantity by subtracting the offset.
- In excess 128 format, the number $01111110 = 126$ implies that the exponent = -2.
- In Excess 2^N format a 1 in the MSB represents a positive exponent and a 0 in the MSB represents a negative exponent.

The Typical 32 bit floating Point Format



Example 1

$$-13.6875 = -1101.1011_2 = -0.11011011 \times 2^4$$

$$\text{Exponent in excess 128} = 128 + 4 = 132 = 10000100_2$$

Sign of the fraction is negative $\Rightarrow 1$

Exponent	sign	fraction
10000100	1	110110110000000000000000

= 84ED8000

The Typical 32 bit floating Point Format



Example 2

To convert the typical Floating point number 7E5C0000 to base 10

$$\text{Exponent} = 7E = 01111110 = 126$$

$$126 - 128 = -2$$

$$5C = 0.1011100$$

Sign of fraction  magnitude of fraction 

$$0.101110 * 2^{-2} = 0.0010111_2 = \mathbf{0.1796875_{10}}$$

Steps in FPF Arithmetic

1. Pre-normalization:

If the operands are not normalized, they should first be put in that form.

2. Alignment:

Two numbers in FPF must have the same exponent before they are added or subtracted, the decimal point associated with the fraction having the smaller exponent must be shifted to the left until the exponents are equal.

3. Post normalization of the result:

Finally if the arithmetic operation produces unnormalized result, then normalize the result.

Examples

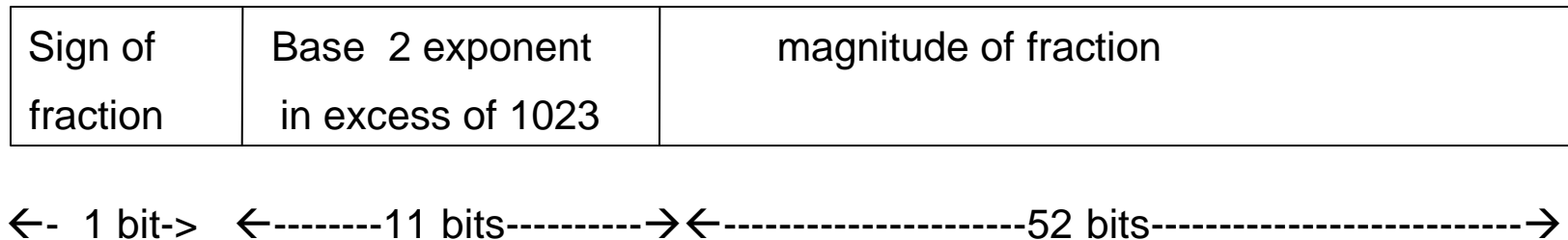
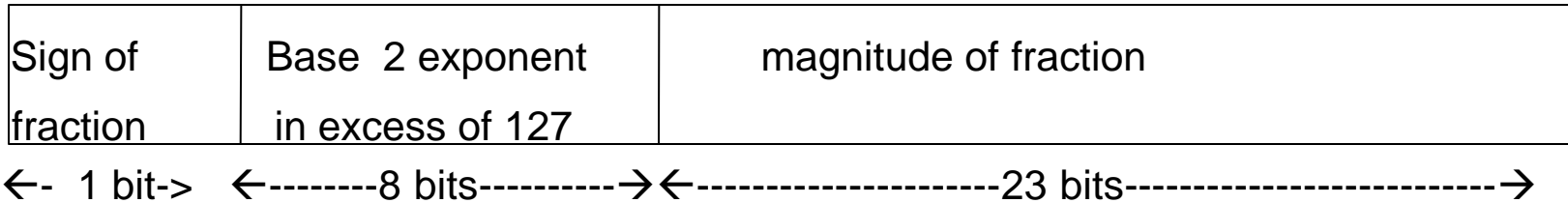
(1) 826013AC *Alignment* 826013AC
 +8040AB04 82102AC1
 82703E6D

(2) 83600000 *No Alignment*
 +83C00000
 83200000 *normalisation* 82400000

(3) 82600000 *No Alignment*
 x 85D00000
 87BC0000 *normalisation* 86F80000

The IEEE/INTEL Floating Point Format

It has 2 forms; the single precision and the double precision format where $N = 127$ and 1023 respectively.



IEEE Floating point Format

Examples

$$-5.375 = -101.011 = -1.01011 * 2^2$$

$$\text{Exponent} = 2 + 127 = 129 = 10000001$$

$$1 \ 10000001 \ 01011 \ \dots = \mathbf{C0AC0000}$$

- **Conversely $3E600000 = 0011 \ 1110 \ 0110 \ 0000 \ 0000$**

- Sign of fraction = 0 \Rightarrow +

- Exponent = 01111100 = 124 - 127 = -3 , i.e 2^{-3}

$$\text{Fraction} = 0.110000000\dots$$

$$\text{Significant} = 1.11 * 2^{-3} = 0.00111_2 = 0.21875_{10}$$

EXAMPLES

$$\begin{array}{r} (1) \ 40700000 \\ + \ \underline{C0580000} \end{array}$$

40180000 —————>

$$\begin{array}{r} (2) \ 40C00000 \ \textit{No Alignment} \\ + \ \underline{C0800000} \end{array}$$

3EC00000 40C00000 —————> 40000000

$$\begin{array}{r} (3) \ 41380000 \\ * \ \underline{3F400000} \\ \hline 410A0000 \end{array}$$

$$\begin{array}{r} (4) \ C0A80000 \\ / \ \underline{40E00000} \\ \hline BF400000 \end{array}$$

BINARY CODED DECIMAL (BCD)

- In **Binary-coded decimal (BCD)**, each decimal digit is represented by its 4 bit binary equivalent e.g.

8159 = 1000 0001 0101 1001

- There are two BCD formats.

➤ **Packed (Condensed) BCD:** In this format each figure occupies half a byte

e.g. 341 = 0011 0100 0001

➤ **Extended (unpacked) BCD:** Each decimal figure is represented by a byte. In this case the first 4 bits of a byte can be filled with zeros or ones depending on the manufacturer, e.g.

341 = 00000011 00000100 00000001

BCD

- Hardware designed for BCD is more complex than that for binary formats. E.g. 16 bits are used to write 8159 in BCD while only 13 bits (111111101111) would be required in binary.
- The advantage of BCD format is that it is closer to the alphanumeric codes used for I/Os.
- Numbers in text data formats must be converted from text form to binary form. Conversion is usually done by converting text input data to BCD, converting BCD to binary, do calculations then convert the result to BCD, then BCD to text output.

BCD

To convert a BCD number to binary, multiply consecutively by $10 = 1010_2$

e.g. $825_{10} = 1000 \ 0010 \ 0101$

$((1000 * 1010) + 0010)1010 + 0101 = 1100111001.$

The reverse conversion is done by successive divisions by 10 (1010_2) then using the 4 bit remainder as the BCD digit:

e.g. $1100111001/1010 = 1010010 \text{ rem } 0101;$

$1010010/1010 = 1000 \text{ rem } 0010$

⇒ $1000 \ 0010 \ 0101$

BCD

The signs often used are 1100 for positive and 1101 for negative.

e.g. $-34 = 0011\ 0100\ \mathbf{1101}$; $+159 = 0001\ 0101\ 1001\ \mathbf{1100}$

BCD addition can be done by successively adding the binary representation of the digits and adjusting the results.

The adjustment rule is: **If the sum of 2 digits is > 9, add 6 to the sum.**

1748	0001	0111	0100	1000
+ <u>2925</u>	<u>0010</u>	<u>1001</u>	<u>0010</u>	<u>0101</u>
4673	0100	0000	0111	1101
		<u>0110</u>		<u>0110</u>
	0100	0110	0111	0011

Non-numeric Data Representation

- Alphanumeric codes are used when the computing device has to handle letters and special symbols as well as decimal digits.
- There are three main coding methods in use:
 1. The American Standard Code for Information Interchange (ASCII)
 2. Extended Binary Coded Decimal Interchange Code (EBCDIC)
 3. Unicode also known as Universal code.

Alphanumeric Data

- Alphanumeric (character) data such as names and addresses are represented by assigning a unique binary code or sequence of bits to represent each character.
- As each character is entered from a keyboard (or other input device) it is converted into its binary code.

Alphanumeric Data

- Character code sets contain two classes of codes:
 - Printable (normal characters)
 - Non-printable ie. characters used as control codes. For example:
 - CTRL P
 - CTRL Z .
 - Most computers use ASCII codes to represent text, which makes it possible to transfer data from one computer to another.

ALPHANUMERIC CODES- ASCII

- ASCII represents each character with a 7 bit string.
- The total number of characters that can be represented is $2^7 = 128$.

e.g. J o h n = 4A 6F 68 6E

= 1001010 110111 1101000 1101110

- It has an extended 8-bit version i.e Extended ASCII which supports more characters.
- used on PC's and non-IBM mainframes
- Was widely used to transfer data from one computer to another – now being replaced by Unicode.

ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	\$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

ASCII control characters			ASCII printable characters			Extended ASCII characters				
00	NULL	(Null character)	32	space	64	@	96	`	128	Ç
01	SOH	(Start of Header)	33	!	65	A	97	a	129	ü
02	STX	(Start of Text)	34	"	66	B	98	b	130	é
03	ETX	(End of Text)	35	#	67	C	99	c	131	â
04	EOT	(End of Trans.)	36	\$	68	D	100	d	132	ä
05	ENQ	(Enquiry)	37	%	69	E	101	e	133	à
06	ACK	(Acknowledgement)	38	&	70	F	102	f	134	å
07	BEL	(Bell)	39	'	71	G	103	g	135	ç
08	BS	(Backspace)	40	(72	H	104	h	136	ê
09	HT	(Horizontal Tab)	41)	73	I	105	i	137	ë
10	LF	(Line feed)	42	*	74	J	106	j	138	è
11	VT	(Vertical Tab)	43	+	75	K	107	k	139	ï
12	FF	(Form feed)	44	,	76	L	108	l	140	î
13	CR	(Carriage return)	45	-	77	M	109	m	141	ì
14	SO	(Shift Out)	46	.	78	N	110	n	142	Ä
15	SI	(Shift In)	47	/	79	O	111	o	143	Å
16	DLE	(Data link escape)	48	0	80	P	112	p	144	É
17	DC1	(Device control 1)	49	1	81	Q	113	q	145	æ
18	DC2	(Device control 2)	50	2	82	R	114	r	146	Æ
19	DC3	(Device control 3)	51	3	83	S	115	s	147	ô
20	DC4	(Device control 4)	52	4	84	T	116	t	148	ö
21	NAK	(Negative acknowl.)	53	5	85	U	117	u	149	ò
22	SYN	(Synchronous idle)	54	6	86	V	118	v	150	û
23	ETB	(End of trans. block)	55	7	87	W	119	w	151	ù
24	CAN	(Cancel)	56	8	88	X	120	x	152	ÿ
25	EM	(End of medium)	57	9	89	Y	121	y	153	Ö
26	SUB	(Substitute)	58	:	90	Z	122	z	154	Ü
27	ESC	(Escape)	59	;	91	[123	{	155	ø
28	FS	(File separator)	60	<	92	\	124		156	£
29	GS	(Group separator)	61	=	93]	125	}	157	Ø
30	RS	(Record separator)	62	>	94	^	126	~	158	×
31	US	(Unit separator)	63	?	95	_			159	f
127	DEL	(Delete)							160	á
									161	í
									162	ó
									163	ú
									164	ñ
									165	Ñ
									166	ª
									167	º
									168	¿
									169	©
									170	¬
									171	½
									172	¼
									173	¡
									174	«
									175	»
									176	⋮
									177	⋮
									178	⋮
									179	⋮
									180	⋮
									181	À
									182	Â
									183	Ã
									184	©
									185	⋮
									186	⋮
									187	⋮
									188	⋮
									189	¢
									190	¥
									191	¬
									192	Ł
									193	ł
									194	Ť
									195	Ŧ
									196	—
									197	†
									198	‡
									199	Ä
									200	ℒ
									201	ℓ
									202	ℓ
									203	ℓ
									204	ℓ
									205	=
									206	≠
									207	≠
									208	δ
									209	Ð
									210	È
									211	Ê
									212	Ë
									213	Ì
									214	Í
									215	Î
									216	Ï
									217	Ĵ
									218	Œ
									219	█
									220	█
									221	█
									222	█
									223	█
									224	Ó
									225	ß
									226	Ô
									227	Õ
									228	ö
									229	Ö
									230	μ
									231	þ
									232	þ
									233	Ú
									234	Û
									235	Ü
									236	ý
									237	Ý
									238	—
									239	·
									240	≡
									241	±
									242	≡
									243	¾
									244	¶
									245	§
									246	÷
									247	°
									248	°
									249	°
									250	·
									251	·
									252	·
									253	²
									254	■
									255	nbsp

and Organisation

ALPHANUMERIC CODES- EBCDIC

- IBM invented this code to extend the Binary Coded Decimal which existed at that time.
- All the IBM computers and peripherals use this code.
- It is an 8 bit code and therefore can accommodate 256 characters.
- **NOTE:**
- Both ASCII and EBCDIC are inadequate for representing all international characters.
eg Chinese characters

ALPHANUMERIC CODES- Unicode

- The Unicode is a 16-bit code so it can represent 65,536 different characters.
- It changes as new character sets are introduced into it.
- It is the most complete character encoding scheme that allows text of all forms and languages to be encoded for use by computers.
- It also incorporates ASCII-7 as subset.

Reading Assignment III

1. what techniques does a computer use to represent images and sound (audio and video) in binary form.