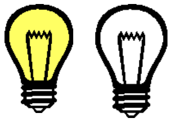
BIRZEIT UNIVERSITY

Data Representation

Computer Science Department

Data Representation

❖**Computer understand two things: on and off .**



❖**Data represented in binary form .**

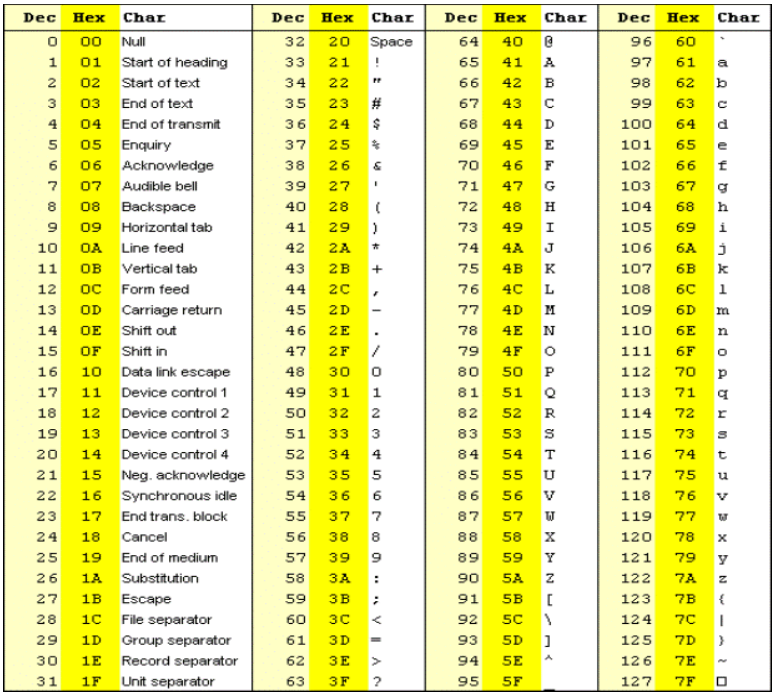
❖**Bit is the basic unit for storing data 0→off ,1→on .**

❖**Byte is a group of 8 bits. That is, each byte has 256(2^8) possible values.**

❖**Two bytes form a word**

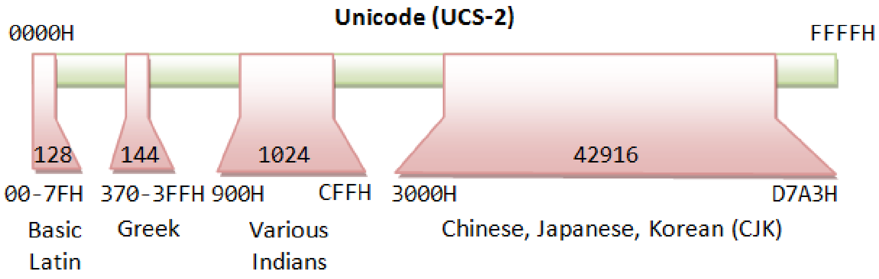
Text: ASCII Characters

- ASCII: Maps 128 characters to 7-bit code



Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	00	Null	32	20	Space	64	40	@	96	60	`
1	01	Start of heading	33	21	!	65	41	A	97	61	a
2	02	Start of text	34	22	"	66	42	B	98	62	b
3	03	End of text	35	23	#	67	43	C	99	63	c
4	04	End of transmit	36	24	$	68	44	D	100	64	d
5	05	Enquiry	37	25	%	69	45	E	101	65	e
6	06	Acknowledge	38	26	&	70	46	F	102	66	f
7	07	Audible bell	39	27	'	71	47	G	103	67	g
8	08	Backspace	40	28	(	72	48	H	104	68	h
9	09	Horizontal tab	41	29	)	73	49	I	105	69	i
10	0A	Line feed	42	2A	*	74	4A	J	106	6A	j
11	0B	Vertical tab	43	2B	+	75	4B	K	107	6B	k
12	0C	Form feed	44	2C	,	76	4C	L	108	6C	l
13	0D	Carriage return	45	2D	-	77	4D	M	109	6D	m
14	0E	Shift out	46	2E	.	78	4E	N	110	6E	n
15	0F	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	Neg. 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	End 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	Substitution	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	□

UCS-2 (Universal Character Set - 2 Byte)



Interesting Properties of ASCII Code

- What is relationship between a decimal digit ('0', '1', …) and its ASCII code?
- What is the difference between an upper-case letter ('A', 'B', …) and its lower-case equivalent ('a', 'b', …)?
- Given two ASCII characters, how do we tell which comes first in alphabetical order?
- Are 128 characters enough?
  (http://www.unicode.org/)

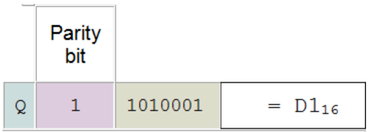
Parity bit

- Used for error detection
- Two types: 1. Odd parity (number of 1's are odd)
  2. Even parity (number of 1's are even)

Characters Representation

Using the **even parity** bit to represent the character **Q** (**Q = 81 in ASCII**) in memory (Hexadecimal) ?

$(81)_{10}=(01010001)_2$



	Parity bit		
Q	1	1010001	= $D1_{16}$

Memory

D1

Note: ASCII for A=65 and a=97

American Standard Code for Information Interchange

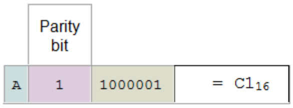
A=65	a=97
B=66	b=98
.	.
.	.

Characters Representation

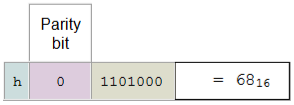
Using the **odd parity** bit to represent **your name** in memory ?

Ex. Ahmad

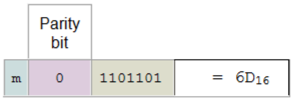
A 01000001
h 01101000
m 01101101
..



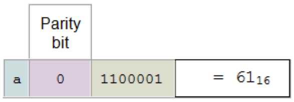
	Parity bit		
A	1	1000001	= $C1_{16}$



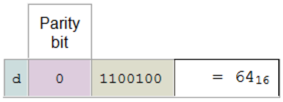
	Parity bit		
h	0	1101000	= 68_{16}



	Parity bit		
m	0	1101101	= $6D_{16}$



	Parity bit		
a	0	1100001	= 61_{16}



	Parity bit		
d	0	1100100	= 64_{16}

Memory

C1
68
6D
61
64

Integers Representation

Represent the following integer in memory using 2 byte?

92 ~ '\'
92 = 1011100

Answer

0000 0000 01011100
0 0 5 C

Memory

5C
00

Integers Representation

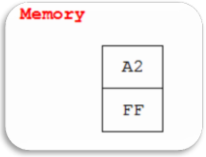
Represent the following integer in memory using 2 byte?

-94

94 = 0000000001011110

1's → 1111111110100001
2's →+ 1

1111111110100010
F F A 2



Memory

A2
FF

Byte Order - Big and Little Endian

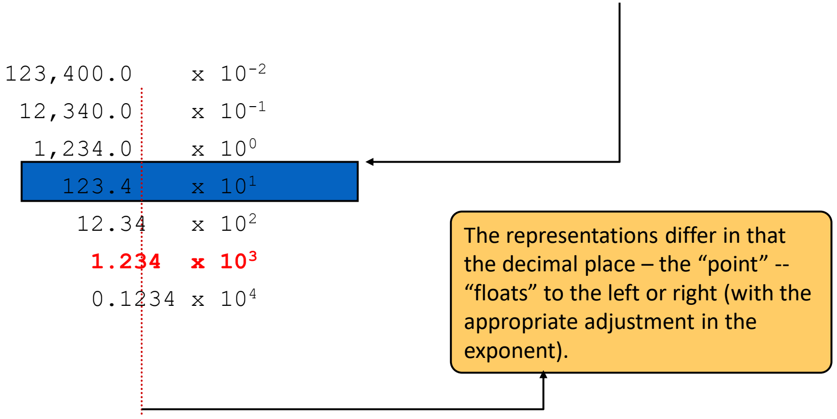
- Endian refers to the order in which bytes are stored.
- **Little Endian:** If the hardware is built so that the lowest, least significant byte of a multi-byte scalar is stored "first", at the lowest memory address.
- **Big Endian:** If the hardware is built so that the highest, most significant byte of a multi-byte scalar is stored "first", at the lowest memory address.
- **Example**: four-byte integer 0x44332211.

Memory Address	**Big-Endian byte value**	**Little-Endian byte value**
104	11	44
103	22	33
102	33	22
101	44	11

Floating Point Numbers

Exponential Notation

- The following are equivalent representations of 1,234



123,400.0 x 10^{-2}

12,340.0 x 10^{-1}

1,234.0 x 10^{0}

123.4 x 10^{1}

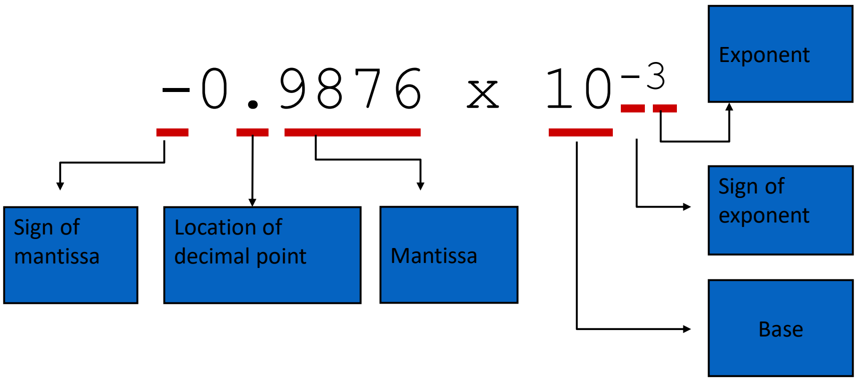
12.34 x 10^{2}

1.234 x 10^{3}

0.1234 x 10^{4}

The representations differ in that the decimal place – the “point” -- “floats” to the left or right (with the appropriate adjustment in the exponent).

Parts of a Floating Point Number



-0.9876×10^{-3}

Exponent

Sign of exponent

Base

Sign of mantissa

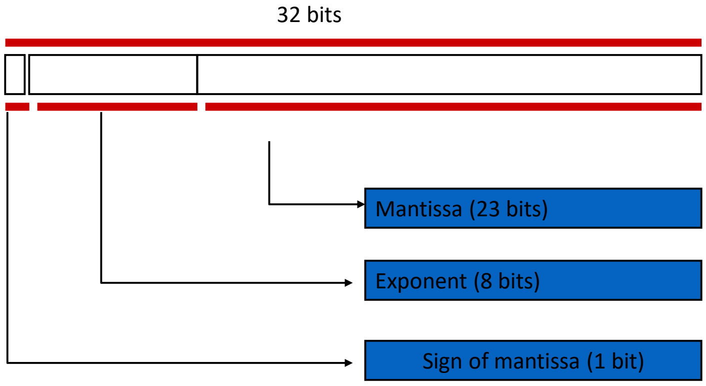
Location of decimal point

Mantissa

IEEE 754 Standard

- Most common standard for representing floating point numbers
- Single precision: 32 bits, consisting of...
  - Sign bit (1 bit)
  - Exponent (8 bits)
  - Mantissa (23 bits)
- Double precision: 64 bits, consisting of...
  - Sign bit (1 bit)
  - Exponent (11 bits)
  - Mantissa (52 bits)

Single Precision Format



Normalization

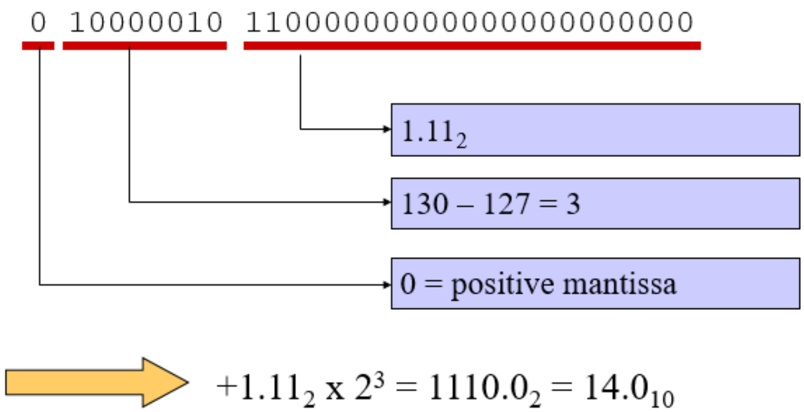
- The mantissa is *normalized*
- Has an implied decimal place on left
- Has an implied “1” on left of the decimal place
- E.g.,
  - Mantissa → `10100000000000000000000`
  - Represents… $1.101_2 = 1.625_{10}$

Excess Notation

- To include +ve and –ve exponents, “excess” notation is used
- Single precision: excess 127
- Double precision: excess 1023
- The value of the exponent stored is larger than the actual exponent
- E.g., excess 127,
  - Exponent → `10000111`
  - Represents… `135 – 127 = 8`

Example 1:

- Single precision



0 10000010 11000000000000000000000

1.11_2

130 – 127 = 3

0 = positive mantissa

$+1.11_2 \text{ x } 2^3 = 1110.0_2 = 14.0_{10}$

Hexadecimal

- It is convenient and common to represent the original floating point number in hexadecimal
- The preceding example…

0 100	0001	0 110	0000	0000	0000	0000	0000
4	1	6	0	0	0	0	0

Example2: Converting from Floating Point

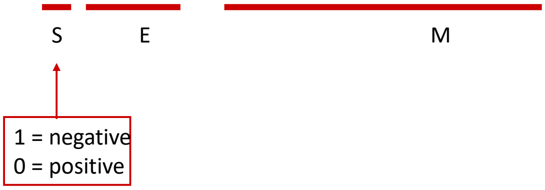
- E.g., What decimal value is represented by the following 32-bit floating point number?

$$\mathtt{C17B0000}_{16}$$

- Step 1
  - Express in binary and find S, E, and M

$\mathtt{C17B0000}_{16}$ =

$\mathtt{1\ 10000010\ 11110110000000000000000}_2$



1 = negative
0 = positive

- Step 2
  - Find “real” exponent, *n*
  - *n* = E – 127
    = 10000010_2 – 127
    = 130 – 127
    = 3

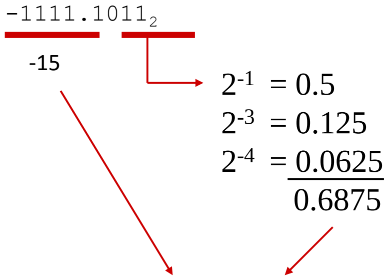
- Step 3
  - Put S, M, and *n* together to form binary result
  - (Don’t forget the implied “1.” on the left of the mantissa.)

$$-1.1111011_2 \text{ x } 2^n =$$

$$-1.1111011_2 \text{ x } 2^3 =$$

$$-1111.1011_2$$

- Step 4
  - Express result in decimal



-1111.1011_2

-15

$2^{-1} = 0.5$

$2^{-3} = 0.125$

$2^{-4} = 0.0625$

0.6875

Answer: -15.6875

Example 3: Converting to Floating Point

- E.g., Express 36.5625_{10} as a 32-bit floating point number (in hexadecimal)

- Step 1
  - Express original value in binary

$36.5625_{10} =$

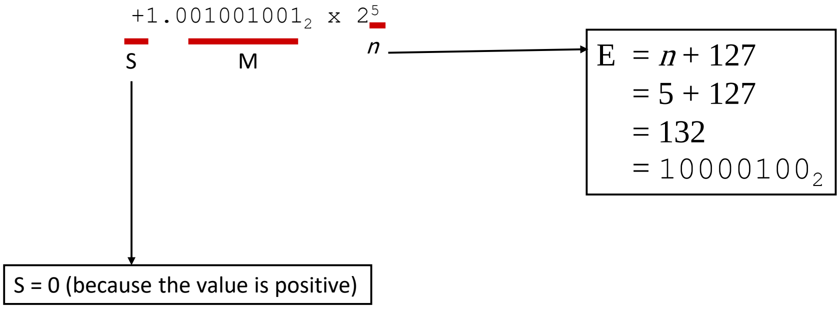
100100.1001_2

- Step 2
  - Normalize

$100100.1001_2 =$

$1.001001001_2 \text{ x } 2^5$

- Step 3
  - Determine S, E, and M



$+1.001001001_2 \text{ x } 2^5$

S

M

n

$$
\begin{aligned}
E &= n + 127 \\
&= 5 + 127 \\
&= 132 \\
&= 10000100_2
\end{aligned}
$$

S = 0 (because the value is positive)

- Step 4
  - Put S, E, and M together to form 32-bit binary result

$$\underbrace{0}_{S}\ \underbrace{10000100}_{E}\ \underbrace{00100100100000000000000}_{M}{}_2$$

- Step 5
  - Express in hexadecimal

$0\ 10000100\ 00100100100000000000000_2$ =

$0100\ 0010\ 0001\ 0010\ 0100\ 0000\ 0000\ 0000_2$ =

4 2 1 2 4 0 0 0_{16}

Answer: 42124000_{16}

Example4: Floating point in Memory

Use the 32-bit floating representation to represent the following the binary number and show how it will represented in the memory?

$(26.75)_{10}$

Answer:
Convert the number from decimal to binary

Floating Point Representation

$(26.75)_{10} = (11010.11)_2$

$(11010.11)_2 = (1.101011 * 2^4)_2$ Scientific notation

Exponent = 127+4=131

$(131)_{10} = (10000011)_2$

0	10000011	10101100000000000000000

4	1	D	6	0	0	0	0

Memory
00
00
D6
41

H.W

Lab 1 . P8,9

Q.5,6,7,9,11