Digital Systems

Week 2: Number Systems and Boolean Algebra
Part II

Fenerbahce University

Course 2 Content

- Bits and Data Types
  - The smallest unit of information is bits
  - Data types
- Integer Data Types
  - Unsigned integers
  - Signed integers
- Complements (2's complement)
  - How are processors made with digital logic structures?
  - How do operations execute on processors?
- Binary to Decimal Conversions
  - From binary to decimal conversion
  - From decimal to binary conversion

Course 2 Content

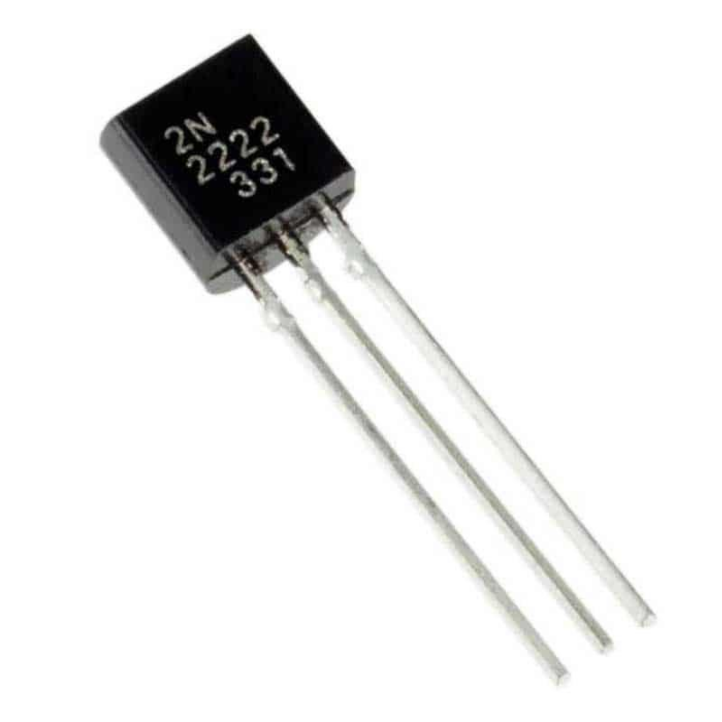
- Operations on bits, Arithmetic Operations
  - Addition and subtraction
  - Sign extension
  - Overflow
- Operations on bits, Logic Operations
  - And/ Or
  - Not
  - Exclusive Or (XOR)
- Other Impressions
  - Bit vector
  - Floating Point data type
  - ASCII codes
  - Hexadecimal notation

How is Data Stored on a Computer?

- The computer is an electronic circuit.
  - It basically works by controlling the flow of electrons.

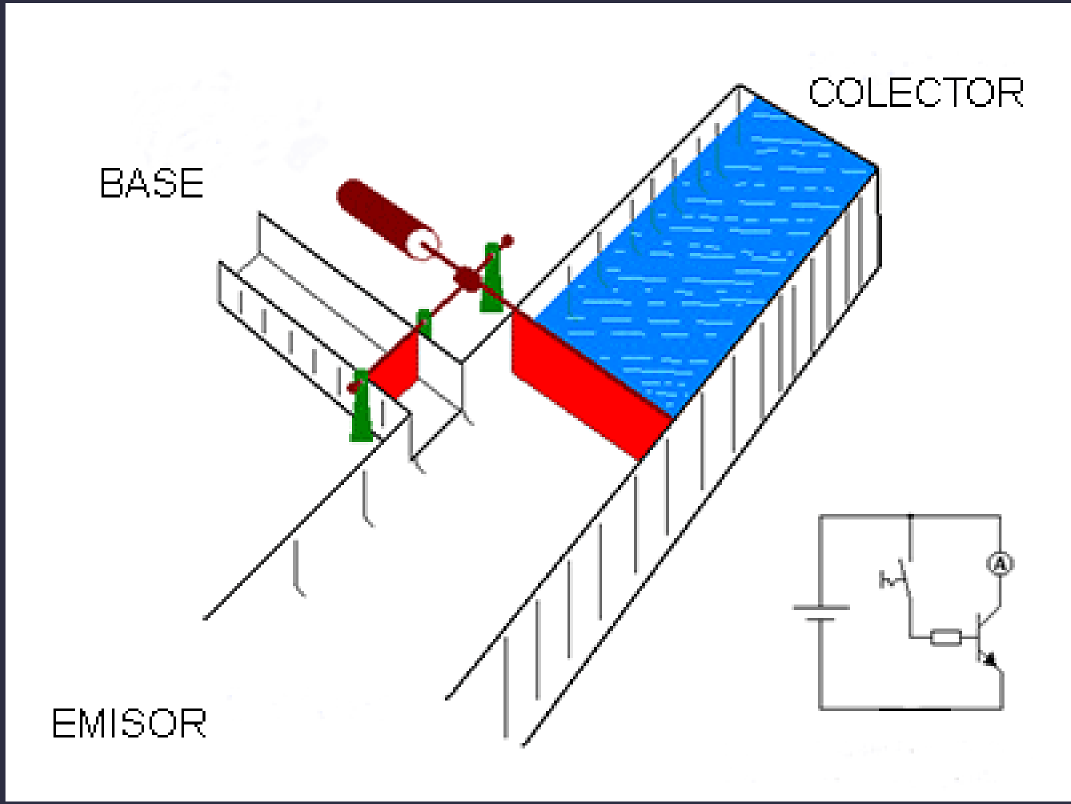
How is Data Stored on a Computer?

- Electrons are controlled by " Transistors " .
  - It basically works by controlling the flow of electrons.



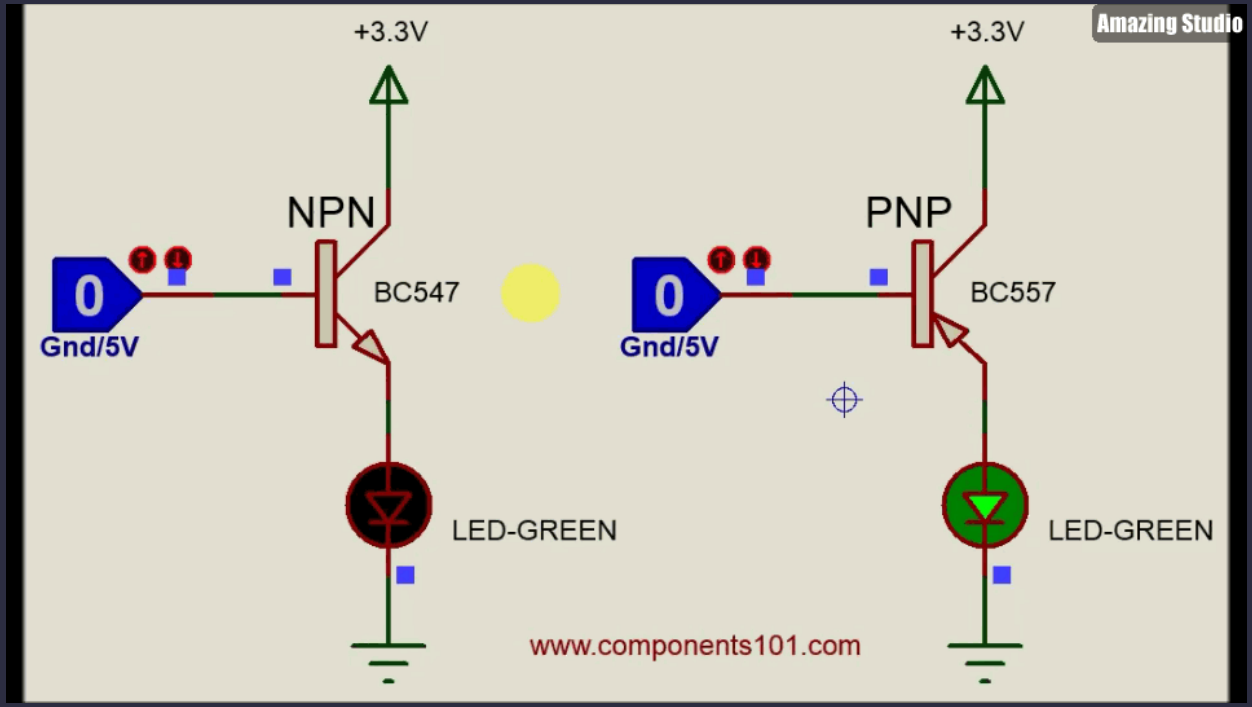
How is Data Stored on a Computer?

- Electrons are controlled by " Transistors " .
  - It basically works by controlling the flow of electrons.



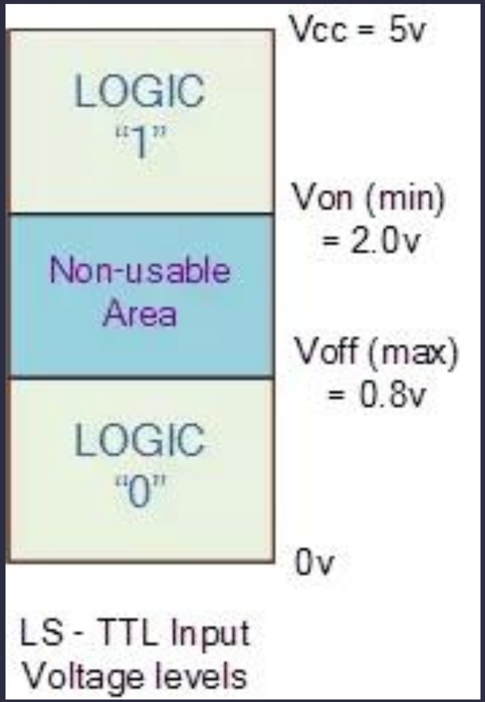
How is Data Stored on a Computer?

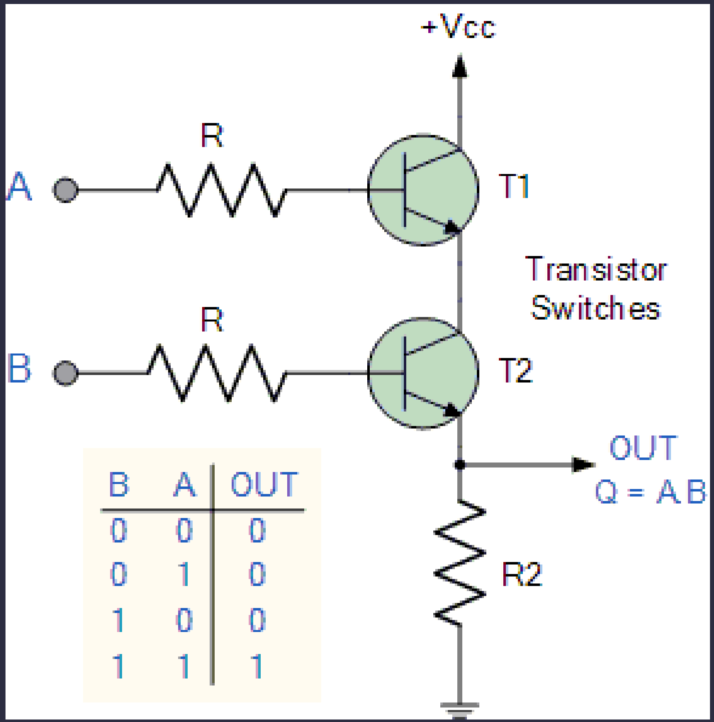
- Electrons are controlled by " Transistors " .
  - It basically works by controlling the flow of electrons.



How is Data Stored on a Computer?

- Data has two states :
  1. the voltage (Voltage) exists – This state is called "1".
  2. The state where the voltage disappears - This state is called "0".





How is Data Stored on a Computer?

- It is also possible to make a computer that works according to more than two voltage states.
  - But the control circuit of this computer will be much more complex.
  - For this reason, today's modern computers work with the concept of bit, which is the smallest unit, while expressing information.
  - It is the smallest data storage unit that can hold 0 or 1 on a bit.

The computers works with binary system.

Binary system :

- It has two states : 0 and 1

- Larger storage areas are obtained by combining multiple bits.
  - two bits , 4 different numbers can be expressed.

The computers works with binary system.

- Two bits , 4 different numbers can be expressed.

- 00 = 0 (in decimal)
- 01 = 1
- 10 = 2
- 11 = 3

The computers works with binary system.

- 3 bits 8 numbers can be expressed by combining them :

- 000 = 0
- 001 = 1
- 010 = 2
- 011 = 3
- 100 = 4
- 101 = 5
- 110 = 6
- 111 = 7

The computers works with binary system.

- In summary;

- 2^n with n bits different numbers can be expressed.

- ^{2} = 4 different numbers for 2 bits
- 2^3 = 8 different numbers for 3 bits
- 2^4 = 16 different numbers for 4 bits

…

can be expressed.

What types of data are expressed / stored in the computer?

- Numbers – signed (unsigned) , integers , decimal numbers (floating _ _ _ _ point), complex numbers (complex) , rational , irrational , ...
- Texts – Characters (characters) , texts (string) , ...
- Images – pixels , images , ...
- Sound
- Logic (logic) – true (true) , false (false)
- Operations (Instructions)
- ...

- Let's start with the numbers ...

Unsigned (Unsigned) Integers (Integers)

- Unsigned integers
  - They always store positive values

  - Ex :

329 (in base 10)

10^2 10^1 10^0

3x100 + 2x10 + 9x1 = 329

101 (in base 2)

2^2 2^1 2^0

1x4 + 0x2 + 1x1 = 5

Unsigned Integers

- An n - bit unsigned integer 2^n has a value : from 0 to 2^n -1 .

2^2	2^1	2^0	
0	0	0	0
0	0	1	1
0	1	0	2
0	1	1	3
1	0	0	4
1	0	1	5
1	1	0	6
1	1	1	7

Unsigned Binary Base Arithmetic

- Binary base addition (like base 10)
  - It is collected starting from the rightmost, and if it is available, it is transferred to the next total.

10010	10010	1111
+ 1001	+1011	+ 1
11011	11101	10000

Signed Integers (Integrs)

- With n bits , we can store 2^n different values .
  - 2^n different value;
  - Signed integers are obtained by assigning half to positive numbers and half to negative numbers.
  - Positive numbers 1 to 2^{n-1}
    Negative numbers $-(2^{n-1})$ to -1
  - For example , if we have a 3-bit storage;
  - Positive numbers are from 1 to 4 and negative numbers are from -4 to -1.

Signed Integers (Integrs)

- For example , if we have a 3-bit storage;
- Positive numbers are from 1 to 4 and negative numbers are from -4 to -1.

- If the number 0 is also used, a number from either positive or negative part is expressed as 0.

Signed Integers (Integrs)

- Positive integers
  - They are like unsigned integers.
    00101 = 5

- Negative integers
  - Sign Bit Representation – Always sign bit is first bit,
    Other bits are written as in unsigned representation.
    10101 = -5

  - 1 's complement – Each bit is inverted
    . 11010 = -5

  - In both representations, the largest bit represents the sign of the number :
    0= positive , 1= negative

Two’s complement

- Sign bit notation and 1 's complement problems
  - The number 0 has two representations (+0 and –0)

  - Sign Bit
    0 0 0 0 0 = +0
    1 0 0 0 0 = - 0

  - 1 's complement

  00000 = +0
  11111 = -0

Two’s complement

- Sign bit notation and 1 's complement problems
  - The necessary hardware circuits of arithmetic operations are very complex.
    - Problem with sign bit denoted addition

```
  1 0 1 1  (-3)
+ 0 0 1 0  (2)
  1 1 0 1  (-5) → Wrong
```

    - For the solution, before adding, it is necessary to check which one is larger, subtract the smaller from the larger, and place the sign of the larger number.
    - Therefore, it is necessary to have a circuit, subtractor and sign bit setter in the necessary hardware to perform the necessary addition process. That is, the hardware becomes complex and large.

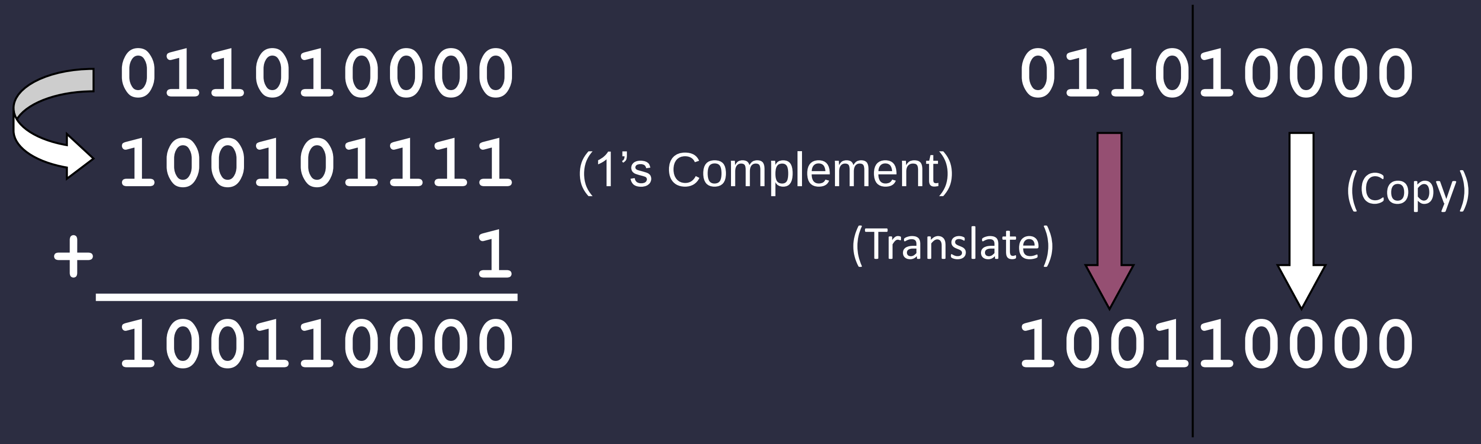
Two’s Complement

- If the value to be expressed is 0 or positive ,
  - They are written as unsigned integers, with the largest bits filled with 0.
- If the number is negative ,
  - written as a positive number
  - Each bit is inverted (1's complement)
  - 1 is added to the result.

```
  00101  (5)                     01001  (9)
  11010  (1 's complement )      1 0110  ( 1's complement )
+     1                        +      1
  11011  (-5)                    10111  (-9)
```

Two’s Complement

- Shortcut to find Two’s complement :
  - Copy bits of the number from right to left until you see the first "1"
  - Reverse remaining bits



Two’s complement

- Biggest bit sign bit and weight -2^{n-1} is .
- -2^{n-1} with n bits It can be expressed from 2^{n-1} to 1 .
  - The smallest negative number (-2^{n-1}) has no positive counterpart .

-2^3	2^2	2^1	2^0	
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7

-2^3	2^2	2^1	2^0	
1	0	0	0	-8
1	0	0	1	-7
1	0	1	0	-6
1	0	1	1	-5
1	1	0	0	-4
1	1	0	1	-3
1	1	1	0	-2
1	1	1	1	-1

Convert binary complement to base 10

1. If the largest bit (leftmost) is 1, take the twos complement of the number and find its positive value.
2. Add the values by multiplying by powers of 2, starting with the rightmost bit.
3. If the number is negative when starting the process (i.e. its leftmost bit is 1), put a - sign on the base 10 number that appears.

n	2^n
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128
8	256
9	512
10	1024

$X = 01101000_{\text{binary}}$

$= 2^6 + 2^5 + 2^3 = 64+32+8$

$= 104_{\text{tens}}$

Convert binary complement to base 10

$X = 00100111_{binary}$
$= 2^5 + 2^2 + 2^1 + 2^0 = 32+4+2+1$
$= 39_{tens}$

$X = 11100110_{binary}$
$-X = 00011010$
$= 2^4 + 2^3 + 2^1 = 16+8+2$
$= 26_{tens}$
$X = -26_{tens}$

n	**2^n**
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128
8	256
9	512
10	1024

Decimal to Binary Complement Conversion

- Method 1 : *Division*

1. Get the absolute value of the decimal number . (It should always be positive .)
2. Divide by two – remainder is the smallest bit .
3. Keep dividing until you find 0, and write the remainder of the divisions from right to left.
4. Add zeros to the most right for completing width of the number. (in the example below the number is assumed to be 8 bits)
   If the decimal number is negative, take the binary complement of the resulting number.

$X = 104_{tens}$

104/2	=	52 k0	*bit 0*
52/2	=	26 k0	*bit 1*
26/2	=	13 k0	*bit 2*
13/2	=	6 k1	*bit 3*
6/2	=	3 k0	*bit 4*
3/2	=	1 k1	*bit 5*
1/2	=	0 k1	*bit 6*

$X = 01101000_{binary}$

Decimal to Binary Complement Conversion

- Second Method : *Subtracting powers of 2*

1. Get the absolute value of the decimal number.
2. Subtract the number less than or equal to the number from the powers of 2.
3. Place 1 in the relevant place .
4. Keep going until you get 0 .
5. Add zeros to the most right for completing width of the number. If the decimal number is negative, take the binary complement of the resulting binary number

n	2^n
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128
8	256
9	512
10	1024

Decimal to Binary Complement Conversion

$X = 104_{tens}$

104 - 64	=	40		*bit 6*
40 - 32	=	8		*bit 5*
8 - 8	=	0		*bit 3*

$X = 01101000_{binary}$

n	2^n
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128
8	256
9	512
10	1024

Addition

- Binary complement numbers is similar to the addition of unsigned numbers. No control mechanism is required.

  - The hand bit to be obtained from the largest bit is discarded.

```
   01101000  (104)
+  11110000  (-16)
   --------
   01011000  (88)
```

Subtraction

- Find the negative form of the second number and add .
  - Binary complement of the second number and add it with the first number

```
  01101000  (104)
- 00010000  (16)
  --------
  01101000  (104)
+ 11110000  (-16)
  --------
  01011000  (88)
```

Sign Extension

- When adding two numbers, both numbers must have the same bit width.
- If we just add 0 to the left of the two numbers to make them the same bit width;

4-bit		8-bit	
0100	(4)	00000100	(currently 4)
1100	(-4)	00001100	(12, not -4)

- For correct calculation, the sign bit of the number is placed where it will be expanded.

4-bit		8-bit	
0100	(4)	00000100	(currently 4)
1100	(-4)	11111100	(currently -4)

Overflow

- When numbers are very large , the sum may turn out to be too large to be expressed in n-bit numbers .

01000	(8)	11000	(-8)
+ 01001	(9)	+ 10111	(-9)
10001	(-15)	01111	(+15)

- Overflow status :
  - It can happen in addition operations where both numbers have the same sign .

Logic Operations

- Are calculated as
  - There are two cases, True =1, False =0

A	B	A and B
0	0	0
0	1	0
1	0	0
1	1	1

A	B	A or B
0	0	0
0	1	1
1	0	1
1	1	1

A	Not A
0	1
1	0

Logic Operation Examples

- And
    - With 0 = result is 0
    - With 1 = result no change

```
        11000101
and     00001111
        --------
        00000101
```

- Or
    - With 0 or operation = no change
    - With 1 or operation = 1

```
        11000101
or      00001111
        --------
        11001111
```

- Not
    - It changes every bit.

```
not     11000101
        --------
        00111010
```

Hexadecimal Notation

- It is a 16 bit format that is frequently used on computers.
  - Each 4 bits of a binary number represents a hexadecimal representation.
  - It provides fewer mistakes than using long 0's and 1's.

Bin	Hex	Dec
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7

Bin	Hex	Dec
1000	8	8
1001	9	9
1010	A	10
1011	B	11
1100	C	12
1101	D	13
1110	E	14
1111	F	15

Converting from Binary to Hexadecimal

- Each 4 bits equals 1
  - They are grouped starting from the right.

```
 011 1010 1000 1111 0100 1101 0111
  ⇓    ⇓    ⇓    ⇓    ⇓    ⇓    ⇓
  3    A    8    F    4    D    7
```

Decimal Numbers : Fixed -Point Representation

- Decimals expression
  - A point is chosing for seperating integer and fraction parts
  - Addition and subtration operations are calculating as twos complement operations

$2^{-1} = 0.5$

$2^{-2} = 0.25$

$2^{-3} = 0.125$

```
  00101000.101 (40.625)
+ 11111110.110 (-1.25)
  ------------
  00100111.011 (39.375)
```

Very Large and Very Small Numbers : Floating Point

- Very large numbers : 6.023 x 10 ^{23} -- Requires
- Very small numbers : 6.626 x 10 $^{-34}$ -- Requires

- Clled “scientific notation” : fraction 2 $^{\text{Force}}$
- fraction (*fraction*), force (*exp1nt*), and It is expressed as sign (I) .
- IEEE 754 Floating Point Representation (32-bits):

1b	*8b*	*23b*
i	Power	Fraction

$$Number = (-1)^{i} \times 1.\text{fraction} \times 2^{\text{power}-127}, 1 \leq \text{power} \leq 254$$
$$Number = (-1)^{i} \times 0.\text{fraction} \times 2^{-126}, \text{power} = 0$$

Floating Point Example

- IEEE floating point representation
- 10111111010000000000000000000000

Sign *Power* *Fraction*

  - Sign bit is 1, so the number is negative
  - Force: 01111110 = 126.
  - Fraction: 0.100000000000… = 0.5.

- Value = $-1.5 \times 2^{(126-127)} = -1.5 \times 2^{-1} = -0.75$.

Texts : ASCII Characters

- The ASCII table is an 8-bit table. Each number between 0-255 has a corresponding character or control signal.

00	nul	10	dle	20	sp	30	0	40	@	50	P	60	`	70	p
01	soh	11	dc1	21	!	31	1	41	A	51	Q	61	a	71	q
02	stx	12	dc2	22	"	32	2	42	B	52	R	62	b	72	r
03	etx	13	dc3	23	#	33	3	43	C	53	S	63	c	73	s
04	eot	14	dc4	24	$	34	4	44	D	54	T	64	d	74	t
05	enq	15	nak	25	%	35	5	45	E	55	U	65	e	75	u
06	ack	16	syn	26	&	36	6	46	F	56	V	66	f	76	v
07	bel	17	etb	27	'	37	7	47	G	57	W	67	g	77	w
08	bs	18	can	28	(	38	8	48	H	58	X	68	h	78	x
09	ht	19	em	29	)	39	9	49	I	59	Y	69	i	79	y
0a	nl	1a	sub	2a	*	3a	:	4a	J	5a	Z	6a	j	7a	z
0b	vt	1b	esc	2b	+	3b	;	4b	K	5b	[	6b	k	7b	{
0c	np	1c	fs	2c	,	3c	<	4c	L	5c	\	6c	l	7c	|
0d	cr	1d	gs	2d	-	3d	=	4d	M	5d	]	6d	m	7d	}
0e	so	1e	rs	2e	.	3e	>	4e	N	5e	^	6e	n	7e	~
0f	si	1f	us	2f	/	3f	?	4f	O	5f	_	6f	o	7f	del

Other Data Types

- Texts
  - Formed by sequential writing of characters
- Image
  - They are formed by the combination of pixels.
    - Black and White : 1 bit (1/0 = black / white)
    - Color : Red, Blue, Green (RGB) comp1nts are available. Each is stored as 8-bit numbers.
- Sound
  - It is usually represented as a sequential recording of fixed-point notation numbers.