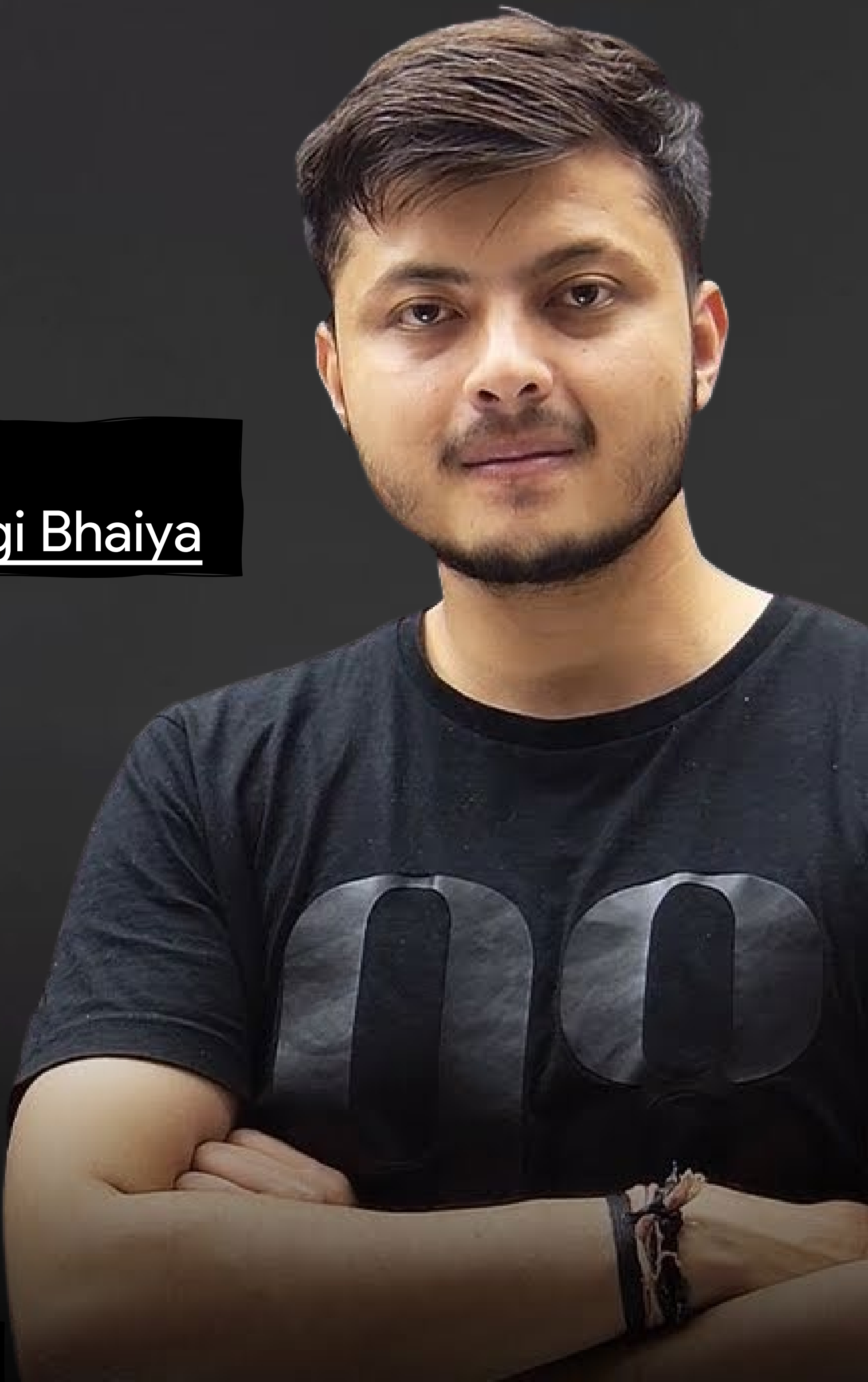
Coder Army



Lecture - 003

By - Rohit Negi Bhaiya

Typed By – Nishit Mehta

<u>Computer Memory Unit</u>

Transistors:

Transistors are the main component of the microchips used in computers. Computers operate on a binary system, which uses only two digits: 0 and 1. In a computer microchip, transistors act as switches, letting current through to represent the binary digit 1, or cutting it off to represent 0. All the data is stored in these transistors only.

Transistor stores **1 bit** i.e., either **0** or **1**. Therefore, the smallest unit is **1 bit**.

(2^0) 1 bit = binary
(2^3) 8 bits = 1 Bytes
(2^{10}) 1024 Bytes = 1 KB
(2^{10}) 1024 KB = 1 MB
(2^{10}) 1024 MB = 1 GB
(2^{10}) 1024 GB = 1 TB
(2^{10}) 1024 TB = 1 PB

For example, if we want to store **4** in our computer. We'll first of all convert **4** into its binary form i.e., **100** then this binary form will be sent to transistor to store in computer. Similarly, we can do for any number.

But let's say, if we want to store **A** in our computer then how can we store it?
To convert **A** directly to binary format, there's no direct method to convert it in binary format. Therefore, to resolve this, computer scientists thought to assign each character with a unique number. This unique number can be converted to binary format and sent to transistor and so we can store **A** in our computer.
Computer Scientists across the globe agreed to this mechanism which led to the formation of **ASCII (American Standard Code for Information Interchange) Table** which allocates very character with a unique number.

We can form our own language, for that we should have the knowledge of compiler that, how shall that language be created.

Dec	Hex	Oct	Binary	Char	Dec	Hex	Oct	Binary	Char	Dec	Hex	Oct	Binary	Char	Dec	Hex	Oct	Binary	Char
0	00	000	0000000	NUL (null character)	32	20	040	0100000	space	64	40	100	1000000	@	96	60	140	1100000	`
1	01	001	0000001	SOH (start of header)	33	21	041	0100001	!	65	41	101	1000001	A	97	61	141	1100001	a
2	02	002	0000010	STX (start of text)	34	22	042	0100010	"	66	42	102	1000010	B	98	62	142	1100010	b
3	03	003	0000011	ETX (end of text)	35	23	043	0100011	#	67	43	103	1000011	C	99	63	143	1100011	c
4	04	004	0000100	EOT (end of transmission)	36	24	044	0100100	$	68	44	104	1000100	D	100	64	144	1100100	d
5	05	005	0000101	ENQ (enquiry)	37	25	045	0100101	%	69	45	105	1000101	E	101	65	145	1100101	e
6	06	006	0000110	ACK (acknowledge)	38	26	046	0100110	&	70	46	106	1000110	F	102	66	146	1100110	f
7	07	007	0000111	BEL (bell (ring))	39	27	047	0100111	'	71	47	107	1000111	G	103	67	147	1100111	g
8	08	010	0001000	BS (backspace)	40	28	050	0101000	(	72	48	110	1001000	H	104	68	150	1101000	h
9	09	011	0001001	HT (horizontal tab)	41	29	051	0101001	)	73	49	111	1001001	I	105	69	151	1101001	i
10	0A	012	0001010	LF (line feed)	42	2A	052	0101010	*	74	4A	112	1001010	J	106	6A	152	1101010	j
11	0B	013	0001011	VT (vertical tab)	43	2B	053	0101011	+	75	4B	113	1001011	K	107	6B	153	1101011	k
12	0C	014	0001100	FF (form feed)	44	2C	054	0101100	,	76	4C	114	1001100	L	108	6C	154	1101100	l
13	0D	015	0001101	CR (carriage return)	45	2D	055	0101101	-	77	4D	115	1001101	M	109	6D	155	1101101	m
14	0E	016	0001110	SO (shift out)	46	2E	056	0101110	.	78	4E	116	1001110	N	110	6E	156	1101110	n
15	0F	017	0001111	SI (shift in)	47	2F	057	0101111	/	79	4F	117	1001111	O	111	6F	157	1101111	o
16	10	020	0010000	DLE (data link escape)	48	30	060	0110000	0	80	50	120	1010000	P	112	70	160	1110000	p
17	11	021	0010001	DC1 (device control 1)	49	31	061	0110001	1	81	51	121	1010001	Q	113	71	161	1110001	q
18	12	022	0010010	DC2 (device control 2)	50	32	062	0110010	2	82	52	122	1010010	R	114	72	162	1110010	r
19	13	023	0010011	DC3 (device control 3)	51	33	063	0110011	3	83	53	123	1010011	S	115	73	163	1110011	s
20	14	024	0010100	DC4 (device control 4)	52	34	064	0110100	4	84	54	124	1010100	T	116	74	164	1110100	t
21	15	025	0010101	NAK (negative acknowledge)	53	35	065	0110101	5	85	55	125	1010101	U	117	75	165	1110101	u
22	16	026	0010110	SYN (synchronize)	54	36	066	0110110	6	86	56	126	1010110	V	118	76	166	1110110	v
23	17	027	0010111	ETB (end transmission block)	55	37	067	0110111	7	87	57	127	1010111	W	119	77	167	1110111	w
24	18	030	0011000	CAN (cancel)	56	38	070	0111000	8	88	58	130	1011000	X	120	78	170	1111000	x
25	19	031	0011001	EM (end of medium)	57	39	071	0111001	9	89	59	131	1011001	Y	121	79	171	1111001	y
26	1A	032	0011010	SUB (substitute)	58	3A	072	0111010	:	90	5A	132	1011010	Z	122	7A	172	1111010	z
27	1B	033	0011011	ESC (escape)	59	3B	073	0111011	;	91	5B	133	1011011	[	123	7B	173	1111011	{
28	1C	034	0011100	FS (file separator)	60	3C	074	0111100	<	92	5C	134	1011100	\	124	7C	174	1111100	|
29	1D	035	0011101	GS (group separator)	61	3D	075	0111101	=	93	5D	135	1011101	]	125	7D	175	1111101	}
30	1E	036	0011110	RS (record separator)	62	3E	076	0111110	>	94	5E	136	1011110	^	126	7E	176	1111110	~
31	1F	037	0011111	US (unit separator)	63	3F	077	0111111	?	95	5F	137	1011111	_	127	7F	177	1111111	DEL

ASCII Table

Variables:

Variables are a name given to a memory location. It is the basic unit of storage in a program.

Rules For Declaring Variable:

- The name of the variable contains letters, digits, and underscores.
- The name of the variable is case sensitive (ex *Arr* and *arr* both are different variables).
- The name of the variable does not contain any whitespace and special characters (ex- #, $, %, *, *etc.*).
- All variable names must begin with a letter of the alphabet or an underscore (_).
- We cannot used C++ keyword (ex-*float*, *double*, *class*) as a variable name.

Datatypes:

While writing code we use **numbers**, **alphabets**, **words**, **decimal numbers** and **gestures**. For each category a specific **datatype** is assigned so that our computer can understand for which category we are talking about.

1. **int:** Numbers are recognized via **int** datatype.
    - Ex: **int name = 10;** Here 'int' is the **datatype**, 'name' is **variable**, '=' is the assignment operator and '10' is the **value**. In simple words, 'name' is a **variable** whose **datatype** is 'int' and which has **value** equal to '10'.
    - The value '10' is stored in binary format in the memory i.e., **1010**.
    - It allocates **4 Bytes** in the memory i.e., **32 bits** in the memory.
    - 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 **1 0 1 0** => 10.
    - To store a large number which **int** cannot fit we use **long int**, it allocates a memory of **8 Bytes** in the memory i.e., **64 bits** in the memory.

2. **char:** Alphabets are recognized via **char** datatype.
    - Ex: **char c = 'a'**; Here 'char' is the **datatype**, 'c' is **variable**, '=' is the assignment operator and "a" is the **value**. In simple words, 'c' is a **variable** whose **datatype** is 'char' and which has **value** equal to 'a'. The values of **char** are always stored inside the **single quotes** ('').
    - The value 'a' is stored in binary format in the memory i.e., **1100001 ->** (97).
    - It allocates **1 Byte** in the memory i.e., **8 bits** in the memory.
    - 0 **1 1 0 0 0 0 1** => **97** which is the ASCII value of 'a'.
    - **char c = 'da'; ->** this is not allowed as only single character is stored in **char**.

3. **float:** Decimal numbers with less precision are recognized via **float** datatype.
    - Ex: **float c = 1.28**; Here 'float' is the **datatype**, 'f' is **variable**, '=' is the assignment operator and '1.28' is the **value**. In simple words, 'f' is a **variable** whose **datatype** is 'float' and which has **value** equal to '1.28'.
    - It allocates **4 Bytes** in the memory i.e., **32 bits** in the memory.

4. **double:** Decimal numbers with more precision are recognized via **double** datatype.
    - Ex: **double d = '1.245'**; Here 'double' is the **datatype**, 'd' is **variable**, '=' is the assignment operator and '1.245' is the **value**. In simple words, 'd' is a **variable** whose **datatype** is 'double' and which has **value** equal to '1.245'.
    - It allocates **8 Bytes** in the memory i.e., **64 bits** in the memory.

Note: We should choose the datatype efficiently for storing the numbers/decimal numbers so that optimum amount of memory is used.

5. **bool:** To store only '0' or '1'/ 'true' or 'false', it is recognized via **bool** datatype.
    - Ex: **bool b = true**; Here 'bool' is the **datatype**, 'b' is **variable**, '=' is the assignment operator and 'true' is the **value**. In simple words, 'b' is a **variable** whose **datatype** is 'bool' and which has **value** equal to 'true'.
    - It allocates **1 Byte** in the memory i.e., **8 bits** in the memory.

Negative - Positive Integer Storage:

An integer takes 32 bits of storage, therefore maximum numbers which it can store is $\mathbf{2^{32}}$. Computer Scientists decided to give half of the bits to positive numbers and rest half to negative numbers. To distinguish that which number is positive and which number is negative, scientists decided that if the 1st bit is **0** then the number is positive and if the 1st bit is **1** then the number is negative. A conflict occurred for number **0** that where to keep this number whether in positive half or negative half. They decided to keep **0** in the positive half.

So, for example if we want to convert **-2** to binary, it does in two steps:

- Find 1's complement (interchanging **0** with **1** and **1** with **0**) of binary form of number **2** -> 0 **1 0** → 1st bit.

    1's complement for number 2 -> **1 0 1**
- Find 2's complement (add **1**) of converted number 1 0 1.

    2's complement for converted number -> **1 1 0** = **-2**

Therefore, the range of **int** to store numbers is $\boldsymbol{-2^{31} <-> 2^{31} - 1}$.