INTRODUCTION:ASSEMBLY LANGUAGE X86

TOPICS

1. Basic Concepts: Applications of assembly language, basic concepts, machine language, and data representation.

2. x86 Processor Architecture: Basic microcomputer design, instruction execution cycle, x86 processor architecture, Intel64 architecture, x86 memory management, components of a microcomputer, and the input–output system.

3. Assembly Language Fundamentals: Introduction to assembly language, linking and debugging, and defining constants and variables.

4. Data Transfers, Addressing, and Arithmetic: Simple data transfer and arithmetic instructions, assemble-link-execute cycle, operators, directives, expressions, JMP and LOOP instructions, and indirect addressing.

5. Procedures: Linking to an external library, description of the book’s link library, stack operations, defining and using procedures, flowcharts, and top-down structured design.

6. Conditional Processing: Boolean and comparison instructions, conditional jumps and loops, high-level logic structures, and finite-state machines.

7. Integer Arithmetic: Shift and rotate instructions with useful applications, multiplication and division, extended addition and subtraction, and ASCII and packed decimal arithmetic.

8. Advanced Procedures: Stack parameters, local variables, advanced PROC and INVOKE directives, and recursion.

9. Strings and Arrays: String primitives, manipulating arrays of characters and integers, two-dimensional arrays, sorting, and searching.

10. Structures and Macros: Structures, macros, conditional assembly directives, and defining repeat blocks.

11. MS-Windows Programming: Protected mode memory management concepts, using the Microsoft-Windows API to display text and colors, and dynamic memory allocation.

12. Floating-Point Processing and Instruction Encoding: Floating-point binary representa- tion and floating-point arithmetic. Learning to program the IA-32 floating-point unit. Under- standing the encoding of IA-32 machine instructions.

13. High-Level Language Interface: Parameter passing conventions, inline assembly code, and linking assembly language modules to C and C++ programs.

14. 16-Bit MS-DOS Programming: Memory organization, interrupts, function calls, and stan- dard MS-DOS file I/O services.

15. Disk Fundamentals: Disk storage systems, sectors, clusters, directories, file allocation tables, handling MS-DOS error codes, and drive and directory manipulation.

16. BIOS-Level Programming: Keyboard input, video text, graphics, and mouse programming.

17. Expert MS-DOS Programming: Custom-designed segments, runtime program structure, and Interrupt handling. Hardware control using I/O ports.

KEYBOARD OPERATIONS

ASCII CONTROL CHARACTERS

This statement from the x86 Assembly book by Kip Irvine is referring to a list of ASCII control characters that are generated when a control key combination is pressed.

ASCII control characters are special characters that are used to control the formatting and communication of data on a screen or printer.

ASCII	HEX VALUE	MEANING	MNEMONIC	CTRL-
0	00h	Null	NUL	Ctrl-@
1	01h	Start of Heading	SOH	Ctrl-A
2	02h	Start of Text	STX	Ctrl-B
3	03h	End of Text	ETX	Ctrl-C
4	04h	End of Transmission	EOT	Ctrl-D
5	05h	Enquiry	ENQ	Ctrl-E
6	06h	Acknowledge	ACK	Ctrl-F
7	07h	Bell	BEL	Ctrl-G
8	08h	Backspace	BS	Ctrl-H
9	09h	Horizontal Tab	HT	Ctrl-I
10	0Ah	Line Feed	LF	Ctrl-J
11	0Bh	Vertical Tab	VT	Ctrl-K
12	0Ch	Form Feed	FF	Ctrl-L
13	0Dh	Carriage Return	CR	Ctrl-M

14	0Eh	Shift Out	SO	Ctrl-N
15	0Fh	Shift In	SI	Ctrl-O
16	10h	Data Link Escape	DLE	Ctrl-P
17	11h	Device Control 1	DC1 (XON)	Ctrl-Q
18	12h	Device Control 2	DC2	Ctrl-R
19	13h	Device Control 3	DC3 (XOFF)	Ctrl-S
20	14h	Device Control 4	DC4	Ctrl-T
21	15h	Negative Acknowledge	NAK	Ctrl-U
22	16h	Synchronous Idle	SYN	Ctrl-V
23	17h	End of Transmission Block	ETB	Ctrl-W
24	18h	Cancel	CAN	Ctrl-X
25	19h	End of Medium	EM	Ctrl-Y
26	1Ah	Substitute	SUB	Ctrl-Z
27	1Bh	Escape	ESC	Ctrl-[

28	1Ch	File Separator	FS	Ctrl-\
29	1Dh	Group Separator	GS	Ctrl-]
30	1Eh	Record Separator	RS	Ctrl-^
31	1Fh	Unit Separator	US	Ctrl-_
32	20h	Space	Space	Ctrl-@

33	21h	Exclamation Mark	!	Ctrl-!
34	22h	Quotation Mark	"	Ctrl-"
35	23h	Number Sign	#	Ctrl-#
36	24h	Dollar Sign	$	Ctrl-$
37	25h	Percent Sign	%	Ctrl-%
38	26h	Ampersand	&	Ctrl-&
39	27h	Apostrophe	'	Ctrl-'
40	28h	Left Parenthesis	(	Ctrl-(
41	29h	Right Parenthesis	)	Ctrl-)
42	2Ah	Asterisk	*	Ctrl-*
43	2Bh	Plus Sign	+	Ctrl-+
44	2Ch	Comma	,	Ctrl-,
45	2Dh	Hyphen	-	Ctrl--
46	2Eh	Period	.	Ctrl-.

47	2Fh	Slash	/	Ctrl-/
48	30h	Digit 0	0	Ctrl-0
49	31h	Digit 1	1	Ctrl-1
50	32h	Digit 2	2	Ctrl-2
51	33h	Digit 3	3	Ctrl-3
52	34h	Digit 4	4	Ctrl-4
53	35h	Digit 5	5	Ctrl-5
54	36h	Digit 6	6	Ctrl-6
55	37h	Digit 7	7	Ctrl-7
56	38h	Digit 8	8	Ctrl-8
57	39h	Digit 9	9	Ctrl-9
58	3Ah	Colon	:	Ctrl-:
59	3Bh	Semicolon	;	Ctrl-;
60	3Ch	Less Than Sign	<	Ctrl-<

61	3Dh	Equal Sign	=	Ctrl-=
62	3Eh	Greater Than Sign	>	Ctrl->
63	3Fh	Question Mark	?	Ctrl-?
64	40h	At Sign	@	Ctrl-@

The mnemonics and descriptions mentioned in the statement are simply labels and descriptions given to these control characters to make them easier to identify and use in programming.

These labels and descriptions allow programmers to quickly and easily understand the function of each control character, and use them to format their output or communicate data effectively.

Note that the ASCII code for **Ctrl-Hyphen (-)** mentioned in the question is actually 1Fh, and is included in the table above.

ALT-KEY COMBINATIONS

Key	Hex Value
Alt+A	1E
Alt+B	30
Alt+C	2E
Alt+D	20
Alt+E	12
Alt+F	21
Alt+G	22
Alt+H	23
Alt+I	17
Alt+J	24
Alt+K	25
Alt+L	26
Alt+M	32

Alt+N	31
Alt+O	18
Alt+P	19
Alt+Q	10
Alt+R	13
Alt+S	1F
Alt+T	14
Alt+U	16
Alt+V	2F
Alt+W	11
Alt+X	2D
Alt+Y	15
Alt+Z	2C

Note that these hexadecimal scan codes are generated by holding down the ALT key and then pressing the corresponding letter key.

These codes are often used in programming to create keyboard shortcuts or to enter special characters that are not available on the keyboard.

characters that are not available on the keyboard.

KEYBOARD SCAN CODES

Scan Code	Hex Value	Description
ESC	01	Escape key
1	02	1 key
2	03	2 key
3	04	3 key
4	05	4 key
5	06	5 key
6	07	6 key
7	08	7 key
8	09	8 key
9	0A	9 key
0	0B	0 key
-	0C	Minus key
=	0D	Equals key

Backspace	0E	Backspace key
Tab	0F	Tab key
Q	10	Q key
W	11	W key
E	12	E key
R	13	R key
T	14	T key
Y	15	Y key
U	16	U key
I	17	I key
O	18	O key
P	19	P key
[	1A	Left bracket
]	1B	Right bracket

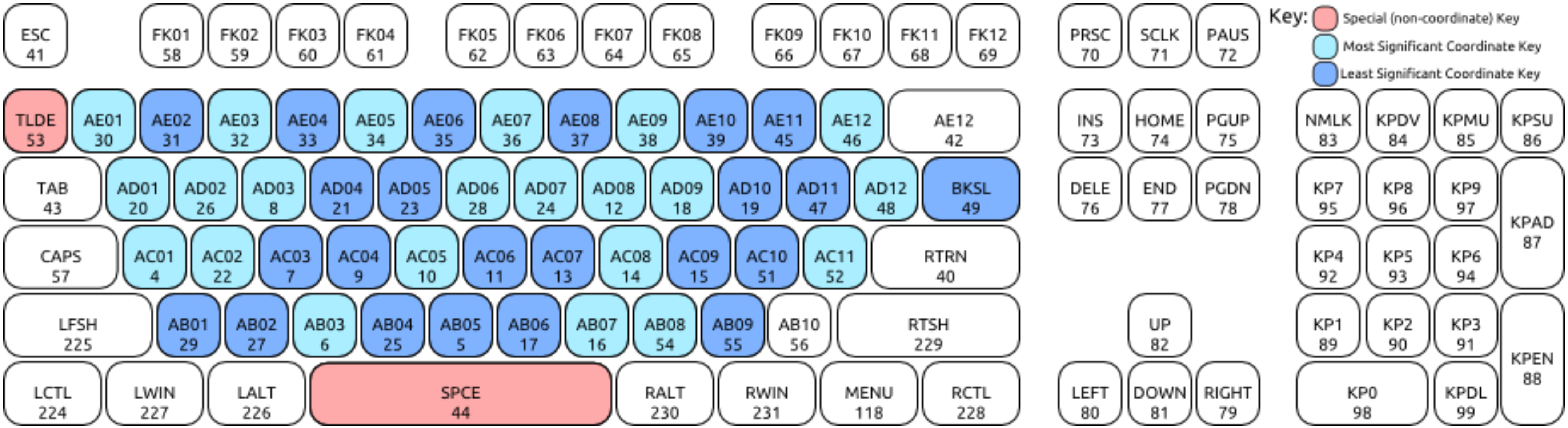
Enter	1C	Enter key
Ctrl	1D	Control key
A	1E	A key
S	1F	S key
D	20	D key
F	21	F key
G	22	G key
H	23	H key
J	24	J key
K	25	K key
L	26	L key
;	27	Semicolon key
'	28	Apostrophe key
`	29	Grave accent key

Shift	2A	Shift key
\	2B	Backslash key
Z	2C	Z key
X	2D	X key
C	2E	C key
V	2F	V key
B	30	B key
N	31	N key
M	32	M key
,	33	Comma key
.	34	Period key
/	35	Slash key
*	37	Asterisk key
Alt	38	Alt key

Space	39	Space bar
Caps Lock		

Keyboard scan codes, ASCII codes, and ALT-key combinations are all related to computer input and keyboard operations.

Keyboard scan codes are hexadecimal codes that represent the physical key pressed on a keyboard. When a key is pressed on a keyboard, it generates a keyboard scan code that is interpreted by the computer's hardware and translated into an ASCII code or other character code that is used by software applications.



ASCII codes are also hexadecimal codes that represent characters used in the ASCII character set.

This character set includes letters, numbers, punctuation marks, and other special characters. ASCII codes are used to represent text data in computers and are often used in software applications, file formats, and communication protocols.

ALT-key combinations are special key combinations that are activated by holding down the ALT key and pressing a specific key on the keyboard. These combinations are often used as shortcuts in software applications to perform specific tasks or commands.

In summary, keyboard scan codes, ASCII codes, and ALT-key combinations are all important aspects of computer input and keyboard operations.

Understanding these codes and combinations is important for developing software applications, working with data files, and communicating with other computer systems.

==

How do programming languages understand ASCII?

If you don’t know the basics of computers, bits, bytes etc, you may find this answer confusing. There must be a good indian somewhere teaching this on YouTube.

ASCII

The fundamental character encoding used in most computers.

American Standard Code for Information Interchange.

Character encoding is a method used to represent characters, symbols, and textual information in computers.

I mean, computers only understand binary’s (1 and 0), so we have to map our whole keyboard to numerical codes that the computer can understand. “We map the characters a, b , c, /, ;” to “binary”.

This mapping is what is called character encoding.

So this is the table, check it first then we continue the discussion:

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	[END 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]

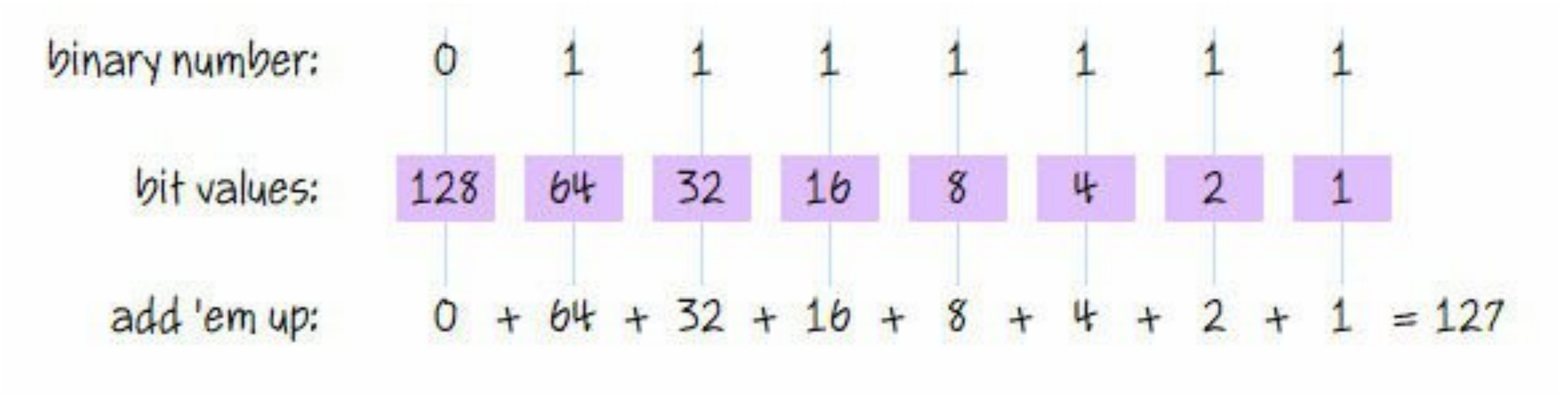
The character 'A' is represented by the numeric value 65 in ASCII, which is 01000001 in binary.

When you type the letter 'A' on your keyboard, the computer's hardware translates that keystroke into the binary representation 01000001, allowing the computer to understand and process the input.

Hex is another format to represent stuff on the computer, but its just a shorter version of binary. “Binary is too verbose”.

Let’s get a bit deeper according to that table, character encoding standard that was introduced in the early days of computing.

It uses 7 bits to represent characters, allowing for a total of 128 unique combinations (2^7 = 128).



The original ASCII table included control characters (0 to 31) and printable characters (32 to 127).

• Control characters (0 to 31): These are non-printable characters used for various control functions in computing, such as carriage return, line feed, tab, etc.

ASCII TABLE

Decimal	Hexadecimal	Binary	Octal	Char
0	0	0	0	[NULL]
1	1	1	1	[START OF HEADING]
2	2	10	2	[START OF TEXT]
3	3	11	3	[END OF TEXT]
4	4	100	4	[END OF TRANSMISSION]
5	5	101	5	[ENQUIRY]
6	6	110	6	[ACKNOWLEDGE]
7	7	111	7	[BELL]
8	8	1000	10	[BACKSPACE]
9	9	1001	11	[HORIZONTAL TAB]
10	A	1010	12	[LINE FEED]
11	B	1011	13	[VERTICAL TAB]
12	C	1100	14	[FORM FEED]
13	D	1101	15	[CARRIAGE RETURN]
14	E	1110	16	[SHIFT OUT]
15	F	1111	17	[SHIFT IN]
16	10	10000	20	[DATA LINK ESCAPE]
17	11	10001	21	[DEVICE CONTROL 1]
18	12	10010	22	[DEVICE CONTROL 2]
19	13	10011	23	[DEVICE CONTROL 3]
20	14	10100	24	[DEVICE CONTROL 4]
21	15	10101	25	[NEGATIVE ACKNOWLEDGE]
22	16	10110	26	[SYNCHRONOUS IDLE]
23	17	10111	27	[ENG OF TRANS. BLOCK]
24	18	11000	30	[CANCEL]
25	19	11001	31	[END OF MEDIUM]
26	1A	11010	32	[SUBSTITUTE]
27	1B	11011	33	[ESCAPE]
28	1C	11100	34	[FILE SEPARATOR]
29	1D	11101	35	[GROUP SEPARATOR]
30	1E	11110	36	[RECORD SEPARATOR]
31	1F	11111	37	[UNIT SEPARATOR]
32	20	100000	40	[SPACE]

• Printable characters (32 to 127): These represent the visible characters that you see on the screen, including letters (uppercase and lowercase), numbers, punctuation marks, and special symbols. In ASCII, the 128th combination (1111111 in binary or 127 in decimal) is used for the "DEL" (delete) control character.

ASCII has 128 characters represented by 7 bits, but modern character encodings like Unicode have significantly expanded the number of supported characters to meet the needs of global communication and computing.

So we’ve said ASCII has 33 control characters (0 to 31 including 127) and 95 printable characters (32 to 126).

UNICODE

When you see this one, you should see “emojis” + “many languages” + “many keyboards”. Its the modern encoding scheme, and uses variable-length encoding.

Unicode represents a very large character set, currently supporting over 144,000+ characters, including characters from multiple writing systems, symbols, emojis, and special characters.

ASCII allows a total of 128 characters.

ASCII is primarily for representing English characters, but Unicode can represent characters from all writing systems used in the world.

It includes characters from various languages, scripts, and symbols, so its multilingual.

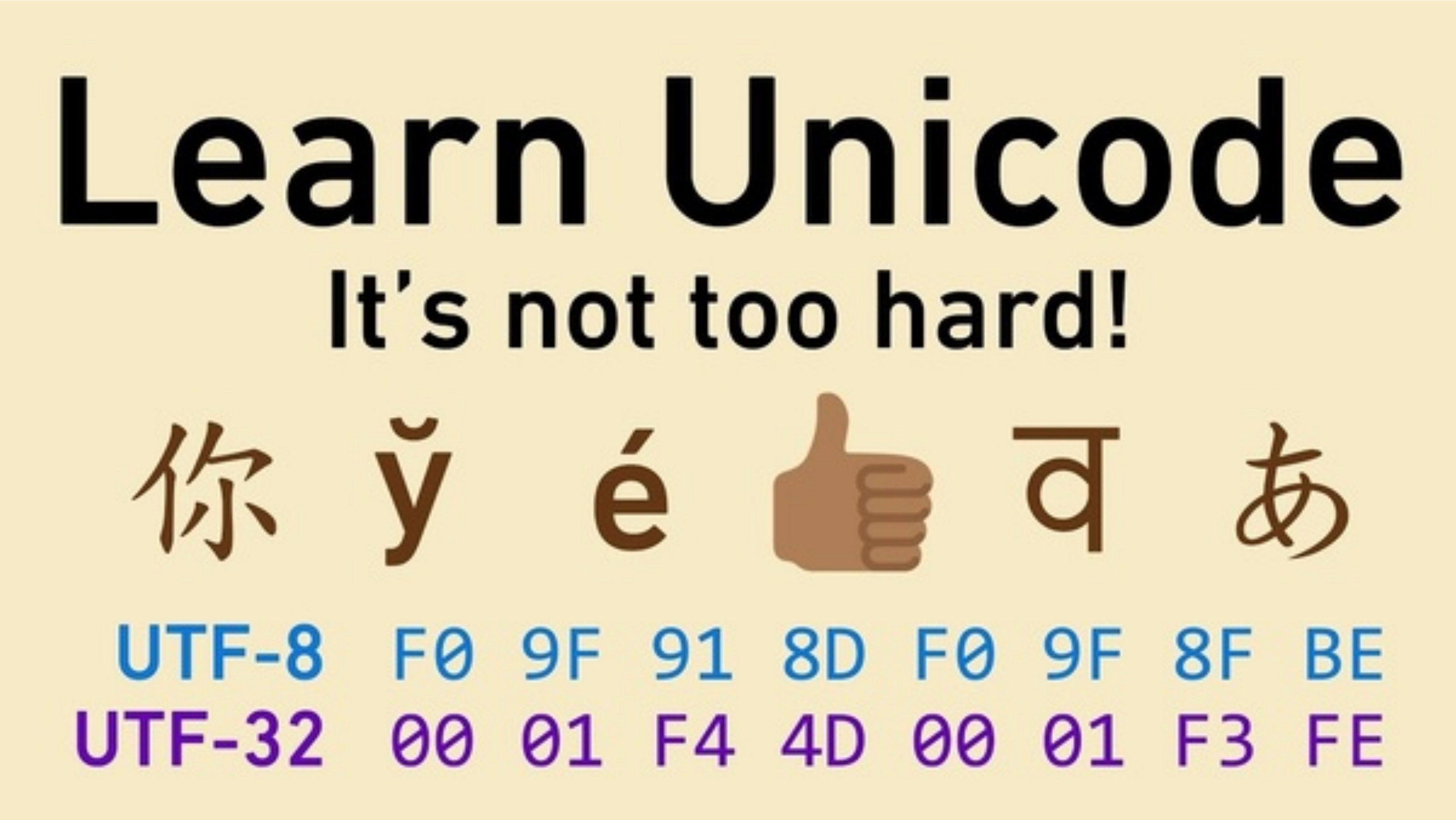
Nepal keyboard unicode:

ASCII uses fixed-length encoding, where each character is represented using 7 bits, Unicode uses variable-length encoding, eg the UTF-8, UTF-16, and UTF-32, to accommodate the larger character set.

These variable-length encodings allows the representation of characters using a variable number of bytes.

NB: UTF-8 is the most widely used encoding for Unicode coz it efficiently represents characters using a variable number of bytes, allowing for compact representation and multilingual support.

using a variable number of bytes, allowing for compact representation and multilingual support.



To finalise, ASCII is a simple and limited character encoding standard primarily used for representing characters in the English language.

Unicode is a more comprehensive and universal character encoding standard that supports characters from various languages and scripts, making it suitable for multilingual applications and global communication.

ANSWER:

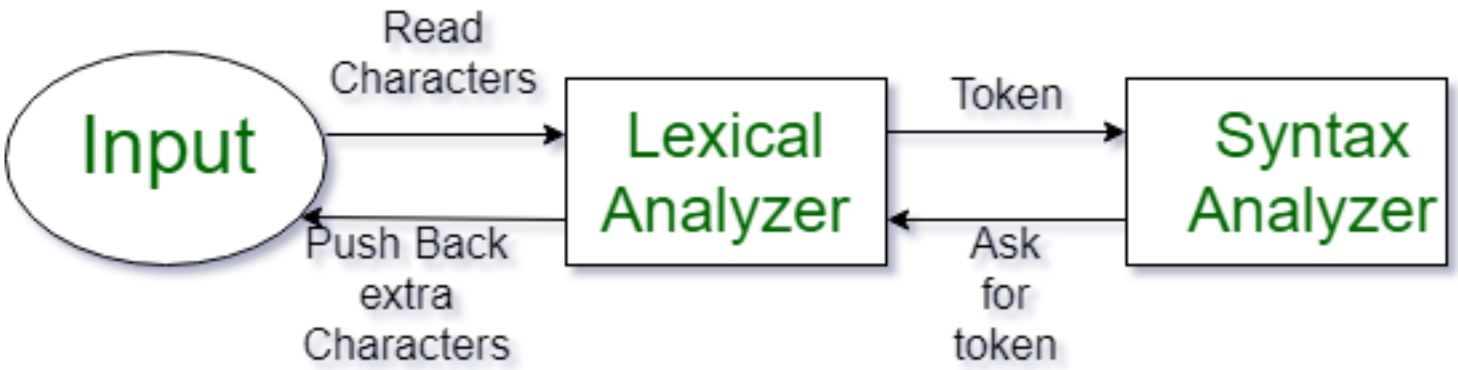
(YOU WILL GET MORE KNOWLEDGE AND UNDERSTANDING IF YOU DO COMPILER DESIGN)

So, you write your code using human-readable characters (letters, numbers, symbols) to represent the instructions and logic for your program.

Text in the source code file is encoded using a specific character encoding scheme.
Lexical analysis is like breaking down a sentence or paragraph into smaller pieces, or "tokens," to make it easier to understand.

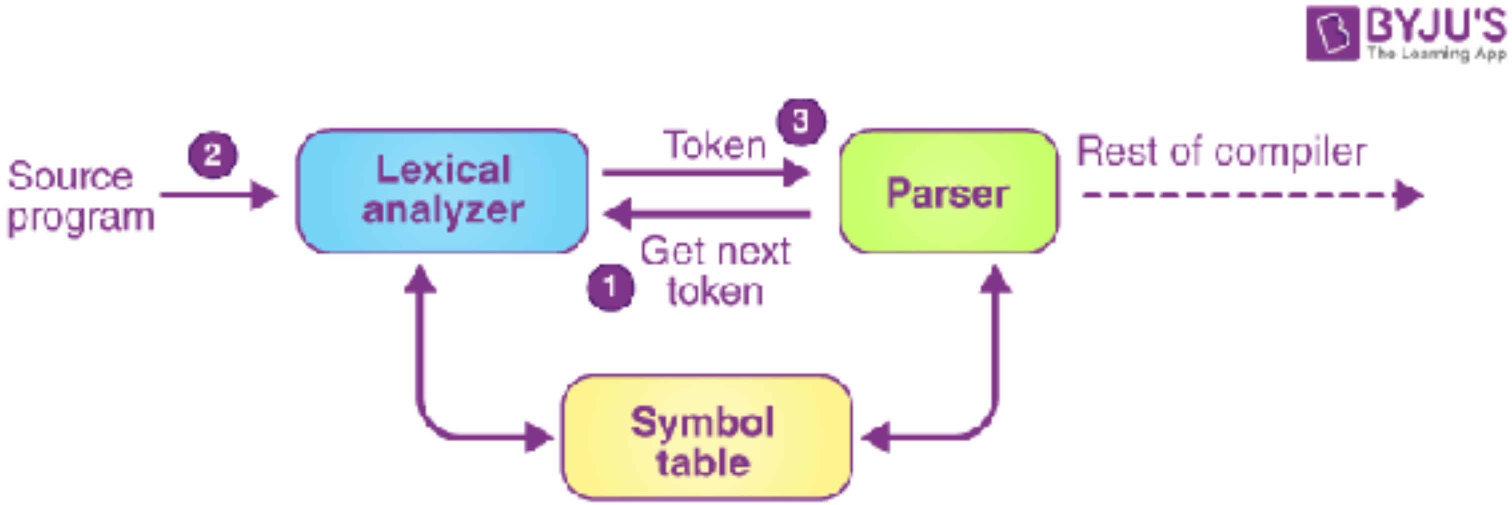
During this process, the compiler or interpreter converts the source code into tokens. Tokens are meaningful chunks of the code, like keywords, identifiers, operators, etc.

ASCII or UNICODE is used here to map the characters in the source code to their corresponding numerical values.



When you write code, the programming language's lexical analyzer breaks down your code into smaller pieces or "tokens." These tokens can be things like keywords (like "if" or "while"), variable names, numbers, operators.

By breaking down the code into tokens, the programming language can better understand the structure and meaning of your code, just like breaking down a sentence helps you understand its meaning. "The," "quick," "brown," "fox," "jumps," "over," "the," "lazy," and "dog".



Once the source code is converted into tokens, the compiler (or interpreter in the case of interpreted languages) translates those tokens into machine code or intermediate code, a low-level representation of the program.

Machine code is executed by the computer's CPU, , following the instructions represented by the binary values of the machine code. These binary values are ultimately derived from the ASCII representation of the characters in the source code.

Now you know something new. Not perfect, not full of every detail of the inner workings, but good enough. Bye!