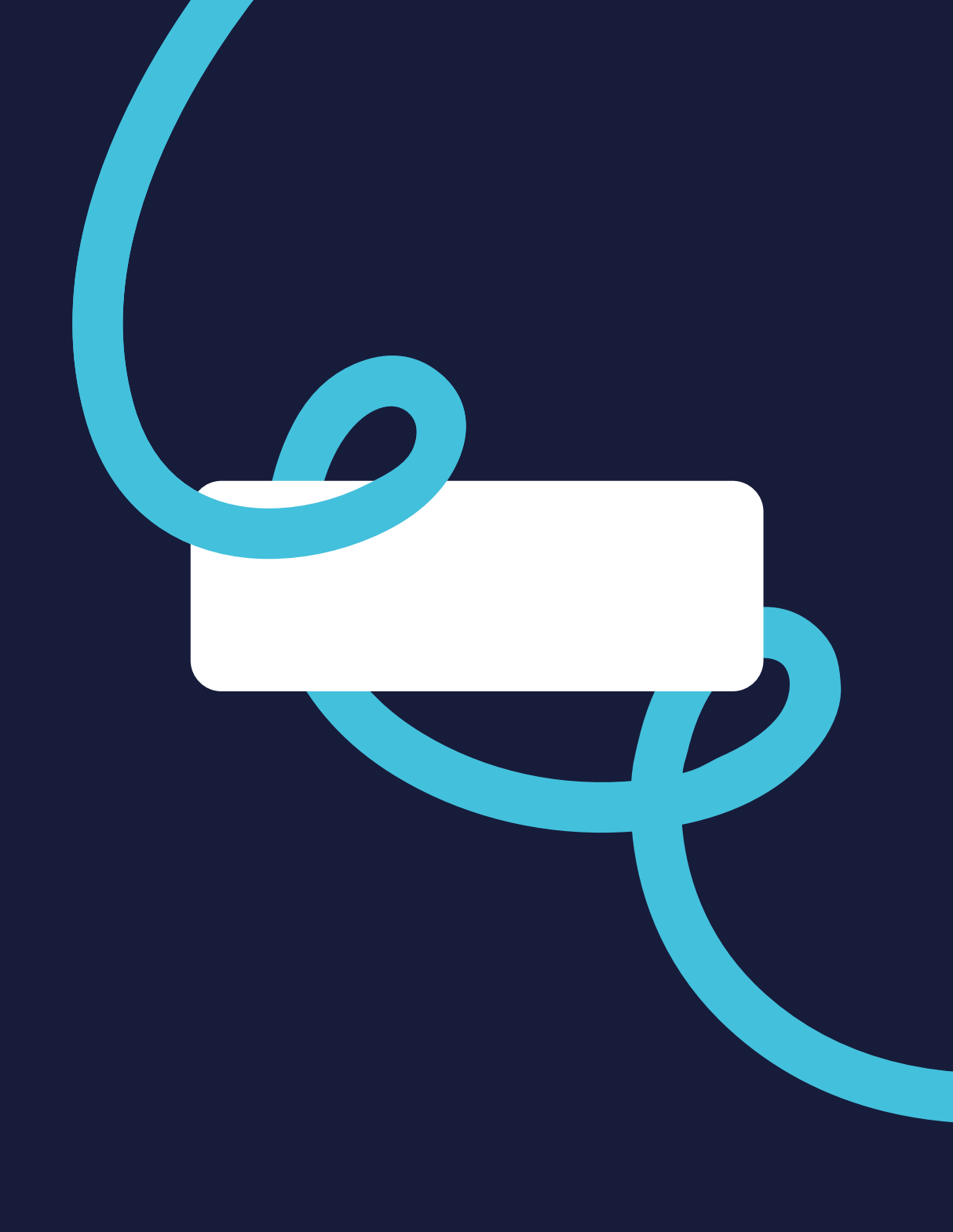
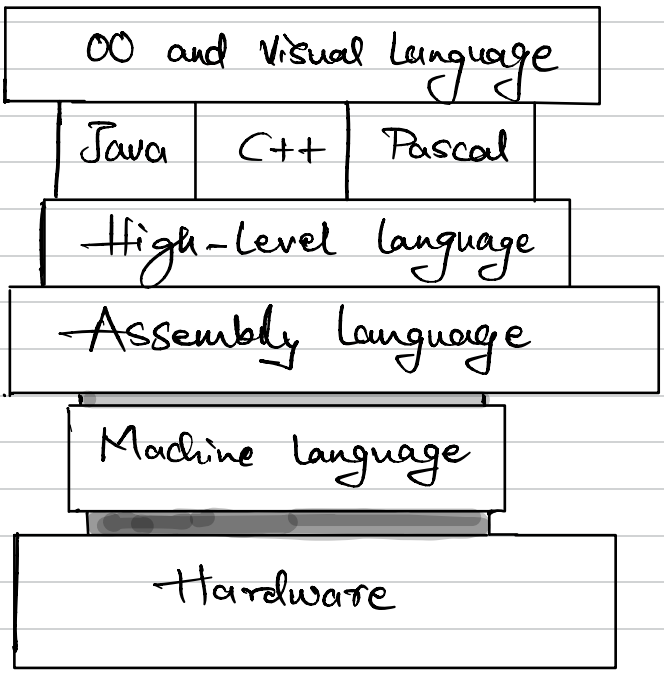


(FreeCode Camp)

↔ Assembly language with ARM:



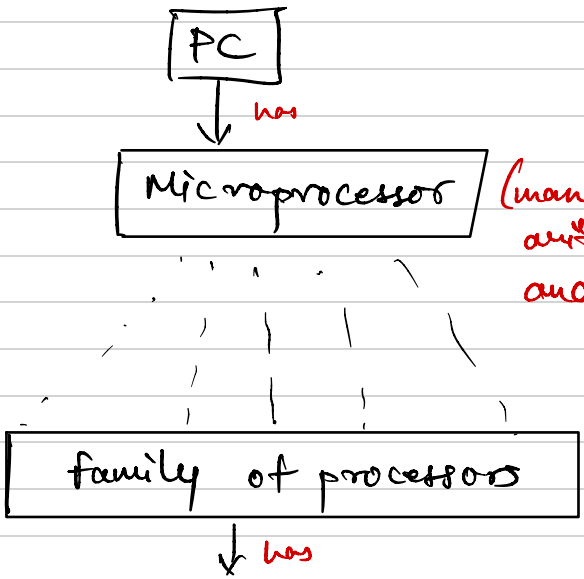
↔ Using ARMv7-DE1SOC Emulator (cpulator.01xz.net)

↔ Some basics before we dive into coding

↳ Assembly language: It's a low level programming language for a computer or other programmable device specific to a particular computer architecture in contrast to most high-level programming languages, which are generally portable across multiple systems.

↳ AL is converted into executable machine code by a utility program referred to as an

assembler like NASM, MASM, etc.



own set of instructions for handling various operations such as getting input from keyboard, displaying information on screen, etc.

↓

These set of instructions are called Machine Language Instructions

- A processor only understands machine language instructions, which are strings of 0's and 1's. However, Machine language → too obscure and complex for using in S/W development. So, low-level AL is designed for a specific family of processors

that represents various instructions in symbolic code and a more understandable code.

Advantages of AL:

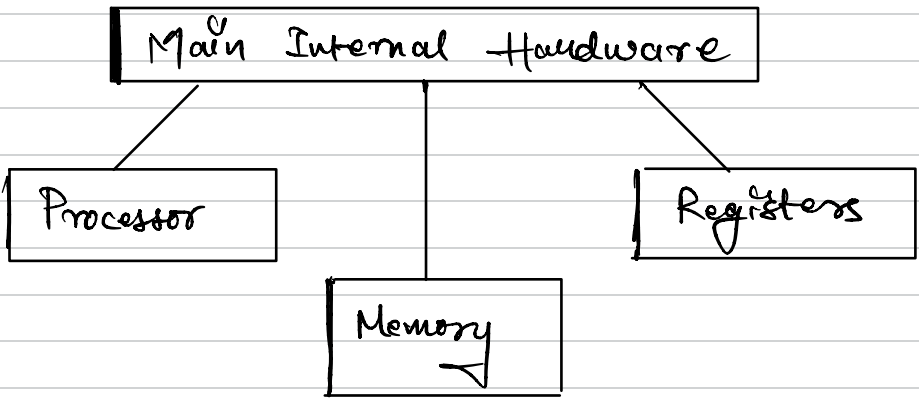
Having an understanding of AL makes one aware of:

- How programs interface with OS, processor, and BIOS;
- How data is represented in memory and other external devices;
- How the processor accesses and execute instruction;
- How instructions access and process data;
- How a program access external device.

Other advantages of using AL:

- AL requires less memory and execution time
- allows hardware-specific complex jobs in an easier way;
- suitable for time-critical jobs;
- most suitable for writing interrupt service routines and other memory resident programs.

Basic features of PC Hardware :



Registers: processor components that hold data and addresses.

To execute a program, the system copies it from the external device into the internal memory.

Processor → executes the program instructions.

Fundamental unit of computer storage : a bit

↓

it could be ON (1) or OFF (0).

Group of 8 related bits makes a byte

Parity bit: used to make the number of bits in a byte odd.

If parity is even, the system assumes that there had been a parity error (though rare), which might have been caused due to hardware fault or electrical disturbance.

Processor supports following data sizes:

- Word: a 2 byte data item
- Doubleword: a 4 byte (32 bit) data item.
- Quadword: an 8 byte (64 bit) " "
- Paragraph: a 16 byte (128 bit) area.
- Kilobyte: 1024 bytes
- Megabyte: 1,048,576 bytes

** Each position is a power of the base, which is 2 for binary number system, and these powers begin at 0 and increased by 1.

Value of a binary number is based on the presence of 1 bits and their positional value.

** Hexadecimal number system uses base 16. Digits in this system range from 0 to 15.

- By convention, the letters A through F is used to represent the hexadecimal digits corresponding to decimal values 10 through 15.

** Hexadecimal numbers in computing is used for abbreviating length binary representations.

Hexadecimal number system

↓

represents

↓

a binary data by dividing each byte in half and expressing the value of each half-byte.

Decimal number	Binary representation	Hexadecimal representation
0	0	0
1	1	1
2	10	2
3	11	3
4	100	4
5	101	5
6	110	6
7	111	7
8	1000	8
9	1001	9

Decimal no.	Binary rep.	Hexadecimal rep.
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

How to count in Binary ?

- **Rules for Counting** (same for Decimal no.)
  1. Increase right most column by one
     & Remaining columns simply carry down
  2. When you run out of digits:
     - Reset column that run out to 0. Increase next column by one.

Decimal	Binary rep
0	0
1	1
2	10
3	11
4	100
5	101
6	110
7	111
8	1000
9	1001
10	1010
11	1011
12	1100
13	1101
14	1110

increase column by 1 → 1 → 10 (no. runs out)

carry down → 11

15	1111

** Binary Conversion

506207		101011		
↓				in Dec
500000		100000	1 x 100000	(32) = 32
00000		00000	0 x 10000	(16) = 0
6000		1000	1 x 1000	(8) = 8
200		000	0 x 100	(4) = 0
00		10	1 x 10	(2) = 2
7		1	1 x 1	(1) = 1
				43

∴ 101011 = 43

1 ⇒ 1
10 ⇒ 2
100 ⇒ 4
1000 ⇒ 8
10000 ⇒ 16
100000 ⇒ 32
1000000 ⇒ 64
10000000 ⇒ 128

Ex: 0 0 0 1 0 0 0 1

for this

Shortcut method

128	64	32	16	8	4	2	1
0	0	0	1	0	0	0	1

Place this here

↳ 16 + 1 ⇒ 17 (Your Answer)

Or minus the place value of 0s from 255 (total of 1, 2, 4, 8, 16, 32, 64, 128)

0 and 255 are the smallest and largest 8-bit no.

Every IPv4 address is 32 bits long.

↓

Broken up in to four 'octets' that are each 8 bits

∴ Smallest 8 bit binary number : 0000 0000 (0)

Largest : 1111 1111 (255)

OR

Hexa to Binary

Hexa ($16 = 2^4$)	Binary (2) 2^3 (8)	2^2 (4)	2^1 (2)	2^0 (1)	
0	0	0	0	0	
1	0	0	0	1	(2^0)
2	0	0	1	0	(2^1)
3	0	0	1	1	($2+1$)
4	0	1	0	0	(2^2)
5	0	1	0	1	($4+1$)
.	.				
.	.				
.	.				

111 only it goes down

Ex: $64.25_{(H)} \rightarrow X_{(2)}$

01100100 . 100101 ⇒ Your Binary conversion

Binary to Hexadecimal

Ex: 10010.1101

⇒ (Group numbers in 4, ← . →)

Grp in this way — Grp in this way

Padded

∴ 0001 0010 . 1101

000 → 1 2 . 13

8+4+1 = 13

∴ 12.13 ⇒ C.D

② 1100 . 1011

⇒ 8421 8421

↳ 12.11

⇒ C.B

Binary to Decimal

1100.1011

1st way

8	4	2	1		1/2	1/4	1/8	1/16
1	1	0	0	.	1	0	1	1
8+4				.	$\frac{1}{2}$ +		$\frac{1}{8}+\frac{1}{16}$	

12 . 0.5 + 0.125 + 0.0625

⇒ 12.6875

2nd way

just diff representation (in writing) [same as above]

(1101.1011)

3	2	1	0		-1	-2	-3	-4
1	1	0	0	.	1	0	1	1

$1\times2^3 + 1\times2^2 + 0\times2^1 + 0\times2^0 \;.\; 1\times2^{-1} + 0\times2^{-2} + 1\times2^{-3} + 1\times2^{-4}$

⇒ $1\times2^3 + 1\times2^2 \;.\; 1\times\frac{1}{2} + 1\times\frac{1}{8} + 1\times\frac{1}{16}$

⇒ 8+4 . 0.5 + 0.125 + 0.0625

⇒ 12.6875

** Binary Arithmetic

0	1	1	1
+0	+0	+1	1
0	1	10	+1
			11

** To convert a binary number to its negative value.

↳ Reverse its bits value and add 1.

Ex: 53 to −53 (53 → Decimal)

↳ This process is called Two's complement

⇒ 53	00111101
↳ reverse	11000010
Add 1	1
−53	11000011

To Add

Ex: 60 + 42 (60 → Dec, 42 → Dec)

	128	64	32	16	8	4	2	1
60 ⇒	0	0	1	1	1	1	0	0

i.step ⇒ 60

−32

28 (we can't do minus 16, ∵ its smaller)

−16

12 (111 in 8)

−8

4 (111 in 4)

−4

0

42 → 128 64 32 16 8 4 2 1

0 0 1 0 1 0 1 0

42 − 32 = 10

10 − 8 = 2

2 − 2 = 0

60 + 42 = 102

```
   (1) (1) (1)
    0   0   1   1   1   1   0   0
    0   0   1   0   1   0   1   0
+ ------------------------------------
    0   1   1   0   0   1   1   0
```

102 → 0 1 1 0 0 1 1 0

↳ To Subtract,

53 − 42

↳ Convert no. being subtracted to two's complement and add the numbers (Here 42)

→ 53 → 128 64 32 16 8 4 2 1

0 0 1 1 0 1 0 1

53 − 32 = 21

21 − 16 = 5

5 − 4 = 1

1 − 1 = 0

Two complement of 42:

42:	128	64	32	16	8	4	2	1
	0	0	1	0	1	0	1	0

42
−(32)
10
−(8)
2
−(2)
0

Reverse: 1 1 0 1 0 1 0 1
Add 1: 1
1 1 0 1 0 1 1 0

53
+(-42)
11

0 0 1 1 0 1 0 1
1 1 0 1 0 1 1 0
0 0 0 0 1 0 1 1

Overflow of the last bit (1 in above ex's answer) is lost

Addressing Data in Memory:

Process through which the processor controls the execution of instructions is referred to as the fetch-decode-execute cycle or the execution cycle.

↳ Contains 3 continuous steps:-

1. Fetching the instruction from memory.
2. Decoding or identifying the instruction.
3. Executing the instruction.

The processor may access one or more bytes at a time.

Ex:- Hexadecimal number: 0725H.

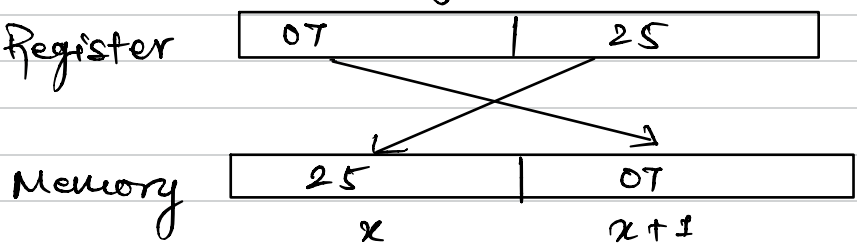
This no. requires two bytes of memory.

The high order byte or most significant byte is 07

Low-order byte is 25.

Processor stores data in reverse-byte sequence, i.e., a low-order byte is stored in a low memory and high-order byte in high memory address.

So, if the processor brings the value 0725H from registers to memory, it will transfer 25 first to the lower memory and 07 to the next memory address.



x: memory address

When the processor gets the numeric data from memory to registers, it again reverses the bytes.

Two kinds of memory addresses:-

- Absolute address — a direct reference of specific location.
- Segment address (or offset): starting address of a memory segment with the offset value.

Good Assembler programs:

- MASM : Microsoft Assembler
- TASM : Borland Turbo Assembler
- GAS : The GNU Assembler

Assembly Basic Syntax: