Author: **Muhammad Ammar** - The Group Leader

BSCS-F19-M-63

Q #2). Why should we learn **Assembly Language**?

How does assembly language relate to high-level languages?

Note:- The Answer of Both Questions is Compiled in one single detailed answer

To understand "WHY" we should learn the Assembly Language, first, we have to know, "WHAT" was the purpose of the creation of The Assembly Language, when there was Machine Language available.

- **Purpose** of the **Assembly Language**
  - The Main Purpose of the creation of Assembly Language was to convert the Machine Code into "Human Readable Language".
  - Machine Code is used to Communicate with the Computer, in the Computer's Language. But it is inconvenient for Humans to read a couple of 0's and 1's and decipher their meaning. So, Assembly Language came into existence with the purpose of Human Readable Machine Instructions.
  - Assembly Language is an exact Mapping of Machine Code, and the Assembler converts the "English" words into the corresponding sequence of 0's and 1's, as according to the Machine Code.
  - The Assembler of the Assembly Language directly "Translates" the Assembly instructions into Machine Code with no Bound Checkings or additional overheads. Hence, it is the direct Translation of the Machine Code with no Performance Loss or hidden changes in the code that the Assembler will make, etc.
  - Simply, Assembly Language is just a Short-Hand for Machine Language.

"Assembly is not just a language, it's a Family of different Hardware Languages"

Since The "WHAT" is answered, now let's move towards the actual Question, "WHY Assembly Language?"
To answer this question, we have three (3)perspectives to perceive the importance and requirement of learning/using assembly language. Let's discuss them one by one.

Importance of Assembly Language in,

- **Software Engineering**

  - **Platform Specific** Softwares

    In Software Development, we have different scenarios in which we have to develop any Software. If we are developing Commercial Software, it has to run on Specific Hardware and utilize the Hardware Resources efficiently. We have an option of various Compilers to compile the code from Higher Level Languages like C/C+, but in many scenarios, we have to perform a specific task, which should be instructed in Assembly Language Only, related to that specific Hardware. More will be discussed in further points.

  - Dealing with **Embedded Systems**

    Every Hardware has its own Machine Code, and hence, unique Assembly Language. While dealing with the Embedded Systems, we directly deal with the hardware and its Machine Code. Assembly Languages are widely used for the Programming of Embedded Systems to Program them efficiently and to make their Source Code in Human Readable format. Modern Systems have their Compilers for higher-level languages like Arduino and Raspberry Pi, but still, Programming them in Assembly gives better control over hardware and its manipulation to cope with the changing requirements and modifications.

  - Control over **Hardware**

    Designing a Hardware Manipulating and Dealing Softwares like HDD/SSD Defragmenter, Performance Benchmarkers, Multi-Threading Environments, and GPU Overclockers and Management Suites, etc, all require strong Hardware knowledge and understanding of Target Machine, and its Architecture. To cope with everything and to design the most efficient Machine Code, we need to have a Strong Grip over the Assembly Language of particular Target Systems.

  - **Multithreading** Environment

    Dealing with Multi-Threading environments and their production requires a strong understanding of CPU Architecture. Having an understanding and knowing the tricks of the CPU, how it manages and executes the operations using multiple cores and ALUs heavily affects the structure of the

instructions provided to the CPU. For that purpose, proper programming of Assembly Language instructions and their flow is necessary to utilize the maximum resources and to use them efficiently. More on CPU Architecture is discussed below.

- **CPU Architecture** Management

  During the Journey of Software Development, we face a condition that the Target Machine has a Specific CPU Architecture and that is specified during the Development Process of Software. Hence, having proper knowledge and understanding of Assembly Language comes in handy to deal with any type of CPU Architecture and not having any trouble while the Development Process of our Software.

  Another case is, during the development of any Application Software, it has to run almost all Platforms having any CPU Architecture. For that purpose, Binaries are built, using almost exactly similar Source Code, using the Tricks and Manipulation of the Assembly Code, generated by the Compiler.

- **OS Libraries** Development

  Operating Systems have a strong connection with organizing and managing the available resources and utilizing them accordingly. The methods of Data Storage, Ease of Access, Hierarchy of Dependant files, and Libraries have to be systematic and fast in terms of performance. All the methods of their organization and execution are stored in the Libraries of Operating System and they are programmed in Assembly Language of the Respective Hardware.

- **Media Codecs**

  Media Codecs are Hardware Accelerated and use the Hardware to Encode and Decode Media files. These media files include images, audio, and videos, including vector graphics and 3D Object files. As it's clear, Codecs use hardware all the time, and so, Programmed using a Hardware Language/Assembly.

- **Kernel** Development

  Kernel Systems are installed on Bare Machine and are deeply coded into Respective Machine Language. Their Development also requires a strong understanding of the Assembly Language of respective Hardware

➢ **Math** Functions

In Higher Level Languages, many Mathematically critical operations are mostly programmed in Assembly, as The Processor is basically designed to do Mathematical Operations and the ALU is mainly involved here.

Apart from Math functions, there are some other functions too, which are coded in Assembly, like malloc()

❖ Game Development

➢ Performance

In The Game Development, besides Story Building, Artistic Visions, Concept Arts, and Mechanics, the main focus of the Development Team is on the Performance of the Game on all the Target Platforms and Structured and Efficient usage of Resources of the Target Machine. All of these factors strongly affect the Gameplay Experience and so the Reputation of the Company. So, to deeply integrate the Game Elements and to make them fully compatible with the available resources, The Assembly Languages are widely used. More on this topic is discussed below.

➢ **Engine** Development

Here comes the most important aspect of the focus of Gaming Industries. Almost Every Gaming industry has its Engine to develop games, like Unreal, Unity, IW, CryEngine, REDEngine, Frostbite, Dunia Engine, etc. To use the hardware most efficiently and effectively, for the smoothest gameplay, these Engines are Coded in Lower Level Languages like C/C++. But for more Performance gain and Lower down the CPU/GPU Loads, many of the Modules of the Engine are coded in Target's Assembly Language to utilize the hardware to its maximum.

➢ **Platform** Independency

To make the Game Platform-independent and to make it available on multiple platforms, the Source Code should be the same and the Engine should compile the Code in required Platforms. To achieve this goal, Engines know various Assembly Languages and Compile the Source Code of Higher Level languages into different Assembly Languages of Target Platforms.

➢ **More** from Hardware

Compilers are designed to convert Higher Languages Codes into Lower Languages ones, but most of the time, The Compiler Generated Code doesn't do the task as efficiently as it is possible and as it should be done. So, Assembly Language Experts are hired and Specific Code chunks, Functions, and Modules of the Engines are specifically coded directly into machine language.

❖ **Computer Science** Student

For Computer Science / IT students, the perspectives for the learning of Assembly get wider. Some of more importance of learning Assembly for CS Students are as follows,

➢ Going Through various **Paradigms**

CS Students go through various Environments and Paradigms in their Course and they have to know every aspect of Computer Science and that's what is taught. To cope with all the Subjects like DLD, Electronics, Automata, and Discrete Mathematics. A Computer Programming Language which builds the connections between all the subjects becomes necessary to have the Best Understanding Possible. So, Assembly is the most Basic and Closest to Hardware Language and hence becomes an important aspect of the course of CS.

➢ Working of **Computer Components**

As CS students, we should know every aspect of The Computer, as we discussed earlier. Not only the structure but the working and the Architecture which the CPU follows and its Organization. To make our imagination a reality, Assembly Language plays a role to let us communicate with our PC and understand how everything works and how the CPU performs its tasks

➢ **Under the Hood** Operations

Most of the time in CS, we deal with the higher-level programming languages and we usually don't know how everything is being done, how the operations are being performed, how the Binaries are getting executed, and how the Compiler is generating Assembly instructions. To understand everything, we have to look at "Under The Hood" and take a deeper look at how the CPU handles our instructions, and which practices are better to follow than the other.

➢ **Understanding**

By learning assembly languages, students come to know the Basics of Computer Architecture and Organization. Some of them are,

- Data Handling
- Memory Hierarchy and Management
- Execution of Instructions
- Arithmetic Operation Execution

➢ Getting to know about **Abstraction**

After stepping into Assembly Language and getting used to its components and management techniques and methods, Students come to know, how much Highly Abstracted the Higher Level Languages are, as compared to Assembly Language and why knowing Assembly is important to understand the Basics of Computer Science, building the connections between the context of different subjects, and to cope with every platform.

➢ Understanding of the **Importance over Higher Level Languages**

So far, we have discussed maximum aspects of the importance of Assembly Language over Higher Level Languages, but let's shed some more light on its significance over Higher Level Languages.

- As Higher Level Languages are **Highly Abstracted**, most of the details are kept hidden and out of the range of The Programmers. Most of the time, it becomes increasingly difficult to perform a Hardware specific Task or to do a Particular Task, or perform a specific Operation, or solve a specific Mathematical Problem. Having the Grip on Assembly Instructions helps programmers to do whatever they want because as soon as they get into the assembly, Compiler lets us do anything we want.
- As discussed earlier, not always the **Code Generated by The Compiler** is as expected. Sometimes an Operation could be performed more efficiently or effectively as compared to the Compiler Generated Sequence of Instructions. Hence, using Assembly instructions manually where possible is a good practice to follow, if a Programmer knows what he's doing.

These were some of the Important Aspects of Learning the Assembly Language.

Q #10) Compare between the **Logic Gate** and **Memory Cell**.

What do you know about AND, OR, NOR, XOR, bits

and bytes?

Part #1 - **Logic Gates** and **Memory Cells**

❖ Introduction

- ➢ Logic Gates and Memory Cells are two, of the most Basic Building Blocks of Computer Circuits and Memory Techniques. Logic Gates are used in all types of Processing and CPU thinking and operations. Memory Cells are built on the Logic Gates, which physically store Digital Signals temporarily in Random Access Memory, or simply, The RAM.
- ➢ Both will be discussed briefly, with their connections with each other.

❖ Memory Cell

- ➢ Memory Cell is a combination of Electronic Circuits and Logic Gates, which stores Binary Signals/Digits (0, 1) in a particular way.
- ➢ It consists of a Unique Combination of Transistors, Logic Gates, and Terminals.
- ➢ Read and Write operations are generally performed on Memory Cells.
- ➢ A Multi-Dimensional Array of Millions of Memory Cells makes a complete RAM Unit.
- ➢ There exist two types of Memory Cells
    - Cells of SRAM
        - Abbreviation for “Static Random Access Memory”.
        - Digital Bits are stored in Semiconductor Material, such that CMOS
        - Have Complex Mechanism
        - 4 to 6 Transistors are used in a Single Memory Cell
        - No Refresh Needed to Retain Data for a long duration of time
        - Needs continuous power to retain data

- Cells of DRAM
    - Abbreviation for “Dynamic Random Access Memory”.
    - Digital Bits are stored in Capacitors
    - Have Simpler Mechanism than SRAM Cells
    - Charging and Discharging of Capacitors represent the Binary Data
    - Charge Leakage is a common disadvantage, so Refresh is needed to retain the data
    - Also needs continuous power to retain data

❖ **Logic Gate**

- ➢ Logic Gate is an “idealized” model for a Bool Function, which receives certain Digital Binary Input, and Based on the input, produces a specialized Output.
- ➢ Logic Gates are widely used in Electronics such as, in Integrated Circuits, CPU Chips, and Electronics Switches
- ➢ CPU takes Decisions and performs execution based on various Logic Gates connected in a particular fashion.
- ➢ The ways of representing a Logical Gate are:
    - Specialized Symbols are used to represent the Logic Gate, (details in Part #2)
    - All the possible input and output combinations are represented in a Truth Table
    - Logic Gates can also be expressed in Boolean Functions
    - Simpler Logic Gates have a certain Circuit Diagram to represent the input and output

❖ Difference between **Memory Cell** and **Logic Gate**

- ➢ As it’s clear from the above explanation that both are interrelated and have a certain significance in Electronics.
- ➢ Memory Cells are Storage Units, and Logic Gates are Operational Units
- ➢ Both have distinct parts and mechanism, and different operation to perform

Part #2 - Gates/Operators

"Computers are composed of nothing more than logic gates stretched out to the horizon in a vast numerical irrigation system."

— Stan Augarten

❖ AND

➢ **Definition**

AND Gate implements the Logic Operation of "Conjunction".

It gives the Output '1' when all the inputs are ON, Otherwise '0'.

➢ **Mathematical** Representation: (.)

If A and B are two inputs, then their Conjunction will be, "A . B".

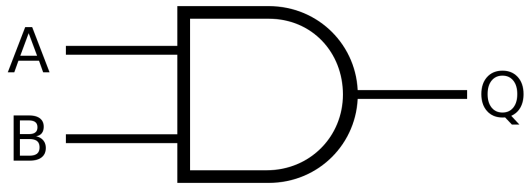
Pronounced as, "A Dot B", or "A conjunction B".

Bitwise Operator: &&

➢ **Truth Table**

A	B	Q = A . B
0	0	**0**
0	1	**0**
1	0	**0**
1	1	**1**

➢ **Symbolic** Representation



❖ OR

➢ Definition

OR Gate implements the Logic Operation of “Disjunction”.

It gives the Output ‘1’ when all or any of the inputs is ON, Otherwise ‘0’.

➢ **Mathematical** Representation: (+)

If A and B are two inputs, then their Disjunction will be, “A + B”.

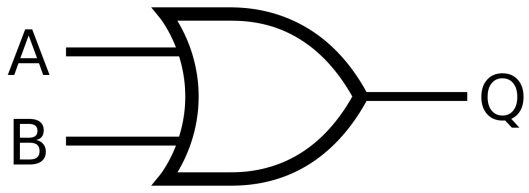
Pronounced as, “A OR B”, or “A disjunction B”.

Bitwise Operator: ||

➢ Truth Table

A	B	Q = A + B
0	0	**0**
0	1	**1**
1	0	**1**
1	1	**1**

➢ **Symbolic** Representation



❖ NOR

➢ Definition

OR Gate implements the Logic Operation of “Negation of Disjunction”.

NOR is the Logical inversion of OR Gate.

Gives the Output ‘1’ when all the Inputs are OFF, Otherwise ‘0’.

➢ **Mathematical** Representation: (+)

If A and B are two inputs, then their NOR will be, **~(A + B)**.

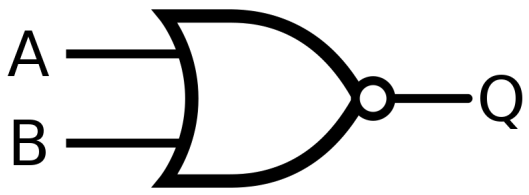
Pronounced as, “Inversion of A OR B”, or “Inversion of A disjunction B”.

Bitwise Operator: using ‘**!**’ before **||**

➢ Truth Table

A	**B**	**Q = ~(A + B)**
0	0	**1**
0	1	**0**
1	0	**0**
1	1	**0**

➢ **Symbolic** Representation



❖ XOR

➢ Definition

XOR Gate gives Output ‘1’ when the number of Inputs having ‘1’ is ODD, Otherwise ‘0’.

XOR is Pronounced as Exclusive-OR Gate.

➢ **Mathematical** Representation: (⊕)

If A and B are two inputs, then their XOR will be, “A ⊕ B”.

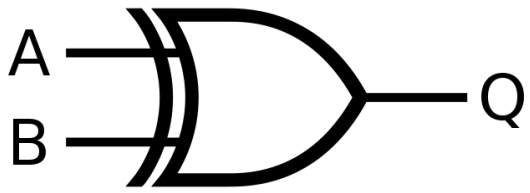
Pronounced as, “A XOR B”.

Bitwise Operator: **^**

➢ Truth Table

A	**B**	**Q = A ⊕ B**
0	0	**0**
0	1	**1**
1	0	**1**
1	1	**0**

➢ **Symbolic** Representation



❖ Bits

➢ Definition

The bit is the smallest quantifying unit of the Data in Computers

"Bit" is a Short Form of "Binary Digit"

A Combination of 8 Bits create another unit of Data, called a Byte

A Collection of multiple bits make meaningful information, called Byte Stream, or Binary Data

➢ Representation

The Computer only understands the Binary Language, which means 0 and 1.

So, a Bit only contains two possible units, '0' or '1'.

These two possibilities can be represented as 'ON' and 'OFF', 'True' and 'False', and '0' and '1'.

➢ Usage in Computers

The computer is made to understand only Binary Language, which consists of particular and unique combinations of Bits, or, '0' and '1's. So, it is used everywhere in Computers, from Storage to Networking, from Media files to the data collected by **NASA's Perseverance**.

➢ Storage Medium

Since it is the "Data", and Data is made to transfer, to share, or to store somewhere. Hence, Bits need a Medium to be stored somewhere.

Memory Cells are used to store individual bits for quick access times.

NAND or XNOR Flash Drives are used to store a vast collection of Data (Bits), to be accessed anytime.

➢ Capable Operations

Many Operations can be applied to the Digital Stream of Data. Some of them are;

- AND → OR
- NOT → NOR
- NAND → XNOR

❖ Byte

➢ Composition

The byte is another Quantifying Unit of Data in Computers, consisting of 8 bits.

8 Bits collectively create a Byte, having some meaningful information i.e 0110011

One Latin Character can be encoded into 1 Byte

➢ Purpose

The purpose of proposing a unit having a combination of 8 bits is to encoding the Meaningful Data into Binary form.

Like, to represent a Latin Character/Letter, we use a particular encoding to convert it into Byte to store that into the memory of Bits.

➢ Representation of **Data Types**

To Program Softwares in Computers, we use various Data and deal with their types. Those types also get encoded into the stream of Bytes. Some of them are (in x86 System);

Data Type	Storage in Bytes
Character	1
Integer	4
Floating Point	4
Double	8

➢ **ASCII** in Bytes

ASCII Characters are encoded in Bytes in following fashion;

ASCII	Decimal	Binary
A	65	01000001
B	66	01000010
C	67	01000011
D	68	01000100
E	69	01000101
F	70	01000110
G	71	01000111
H	72	01001000
I	73	01001001
J	74	01001010
K	75	01001011
L	76	01001100
M	77	01001101
N	78	01001110
O	79	01001111
P	80	01010000
Q	81	01010001
R	82	01010010
S	83	01010011

Q #5). What is an **Embedded system**?

What are the **Real-world applications** of embedded systems?

❖ Embedded System

➢ System

A System is a Combination of different devices which work together to perform a particular task.

➢ **Definition** of Embedded System

Embedded System is simply a Hardware System consisting of a Microprocessor with a Programmed Software, Storage Medium/Device, and Input-Output Peripheral Devices, which is designed to perform a specific task, or a number of particular tasks independently, or as a part of a larger framework.

Embedded Systems are not like Ordinary PCs, in fact, they are dedicated to perform an Operation just like an Ordinary Computer.

They carry out Operations and Computations in Real-Time, like Military Weapon Target System etc.

➢ **Structure**

Depending on the purpose, an Embedded System may consist of;

- Microprocessor with a Programmed Software
- Embedded Storage Medium
- Input/Output Peripheral devices
- Mechanical Parts
- USB Ports
- HDMI Ports

❖ Applications

Embedded Systems are used in many Fields of Technology. Some of them are;

➢ House Hold

We use Embedded System containing devices in our Daily Routine, some of them are,

- Refrigerator
- Microwave
- Air Conditioners
- Robotic Systems
- A.I. Devices

➢ Transport

Embedded Systems are used everywhere in the Transport Systems, like in

- Vehicle Control System
- Vehicle Navigation System
- Self-Driving Vehicles
- Illegal Practices controllers
- Aerial Navigation Systems
- Auto-Pilot Managing Systems

➢ Medical

Embedded Systems are also widely used in Medical systems such as,

- Database Management
- Records Management
- Real-Time Health Diagnostics
- Artificial Organs Maintenance Systems

➢ **Military**

Military systems are almost entirely based on Embedded Systems, like

- GPS Navigation
- Radio Communication Devices
- Digital Mapping and Coordinates Finders
- Remote Controllers
- UAVs - Unmanned Aerial Vehicles
- UGVs - Unmanned Ground Vehicles
- VTOL Balancing Systems
- Detonators
- Infrared Sensors
- Motion Detectors
- Laser Equipments
- Spotting Device
- Stinger Missiles
- Missile Targeting Systems
- Range Finding
- Shot Precision Measurers
- Auto Targeting Systems
- RADAR Systems
- SONAR Systems
- Tracking Devices

❖ **References**

➢ 99% By Myself :), Alhamdulillah

➢ Pictures included from wikipedia