Lab 2

Brief Introduction to Assembly Language

Assembly language is a low-level programming language that is closely tied to the architecture of a computer's central processing unit (CPU). It serves as a bridge between machine code (binary code that a CPU understands) and high-level programming languages (like C++, Java, or Python) that are more human-readable and user-friendly.

Here's a brief explanation of assembly language:

1. **Human-Readable Representation:** Assembly language uses mnemonics and symbols to represent the instructions that a CPU can execute. These mnemonics are easier for humans to understand than binary machine code.

2. **Close to Hardware:** Assembly language is specific to a particular computer architecture, which means that different CPUs have different assembly languages. It provides a direct interface to the computer's hardware, allowing programmers to manipulate registers, memory locations, and other hardware components at a very detailed level.

3. **Low-Level Operations:** Programmers write code in assembly language by specifying individual operations, such as moving data between registers, performing arithmetic operations, and controlling program flow (e.g., branching and looping). Each assembly instruction corresponds to a single operation that the CPU can execute.

4. **Efficiency and Control:** Assembly language programs can be highly efficient because they allow programmers to fine-tune code to make the most of a CPU's capabilities. It offers precise control over hardware resources, making it suitable for tasks where performance optimization is critical, such as device drivers, embedded systems, and real-time applications.

5. **Platform-Dependent:** Assembly language is not portable. Code written for one CPU architecture will not work on another without significant modifications. This lack of portability can be a drawback when compared to high-level languages that can be more easily adapted to different platforms.

6. **Learning Curve:** Assembly language programming can be challenging for beginners due to its close relationship with hardware and the need to understand the specific architecture of the target CPU. However, it can be a valuable skill for those who need to work on low-level system programming or who want to understand the inner workings of computers at a deeper level.

In summary, assembly language is a low-level programming language that allows programmers to interact directly with a computer's hardware by writing human-readable instructions specific to a particular CPU architecture. It offers control,

efficiency, and precision but requires a deep understanding of the hardware and is not easily portable between different computing platforms.

Brief Explanation of Fetch, Decode, Execute Cycle

The "Fetch, Decode, Execute" cycle, also known as the "instruction cycle," is a fundamental concept in computer architecture that describes how a computer's central processing unit (CPU) processes instructions from a program. It's a repetitive sequence of steps that the CPU goes through for each instruction in a program. Here's a brief explanation of each step:

Fetch:

- In this first step, the CPU fetches (loads) the next instruction from the computer's memory, typically from RAM (Random Access Memory).
- The CPU uses a special register called the Program Counter (PC) or Instruction Pointer (IP) to keep track of the memory address of the next instruction to be executed.
- The program counter is incremented to point to the next instruction after the current one has been fetched.

Decode:

- After fetching the instruction, the CPU decodes it. Decoding involves determining what operation the instruction is instructing the CPU to perform.
- The CPU identifies the instruction's opcode (operation code), which specifies the operation to be executed, and any associated operands or data.

Execute:

- In this step, the CPU performs the actual operation specified by the decoded instruction.
- Depending on the instruction, this operation could involve arithmetic calculations, data transfers between registers and memory, logical operations, or control flow changes (such as branching to another part of the program).
- The result of the execution may be stored in registers or memory, depending on the specific instruction.

Once the "Execute" step is completed, the cycle repeats. The program counter is incremented again to fetch the next instruction, and the process continues until the program's end is reached or a specific termination condition is met.

This cycle is at the heart of how a CPU processes instructions, making it a crucial concept in computer architecture. It allows the CPU to sequentially execute instructions from memory, enabling the computer to perform a wide range of tasks by following the instructions provided by a program.

Discussing Linker, Debugger, Assembler/Compiler In General

Linker:

- **Purpose:** A linker is a software tool that combines multiple object files or modules into a single executable program or library.
- **Function:** It resolves references between different parts of a program, ensuring that functions or variables declared in one part of the program can be correctly accessed and used by other parts.
- **Example:** In a program written in C or C++, when you have source code split into multiple files, the linker combines them into a single program, resolving references between functions and variables.

Debugger:

- **Purpose:** A debugger is a software tool used for identifying and resolving errors, or bugs, in a program.
- **Function:** It allows developers to execute a program step-by-step, set breakpoints to pause execution at specific points, inspect variable values, and analyze the program's state during runtime.
- **Example:** When a program crashes or exhibits unexpected behavior, a debugger helps pinpoint the exact location and cause of the problem by allowing developers to examine the program's execution flow and data.

Assembler:

- **Purpose:** An assembler is a tool used in low-level programming to convert assembly language code into machine code.
- **Function:** Assembly language is a human-readable representation of a computer's instructions. Assemblers translate this code into the binary machine code that the CPU can directly execute.
- **Example:** Programmers writing code in assembly language use assemblers to convert their code into executable machine code.

Compiler:

- **Purpose:** A compiler is a software tool used to translate high-level programming languages (e.g., C, C++, Java) into machine code or lower-level code.
- **Function:** Compilers analyze the high-level code, generate an intermediate representation, optimize it, and then produce efficient machine code that can run on a specific computer architecture.
- **Example:** When you write code in a high-level language, a compiler converts it into machine code, allowing the program to execute on a computer.

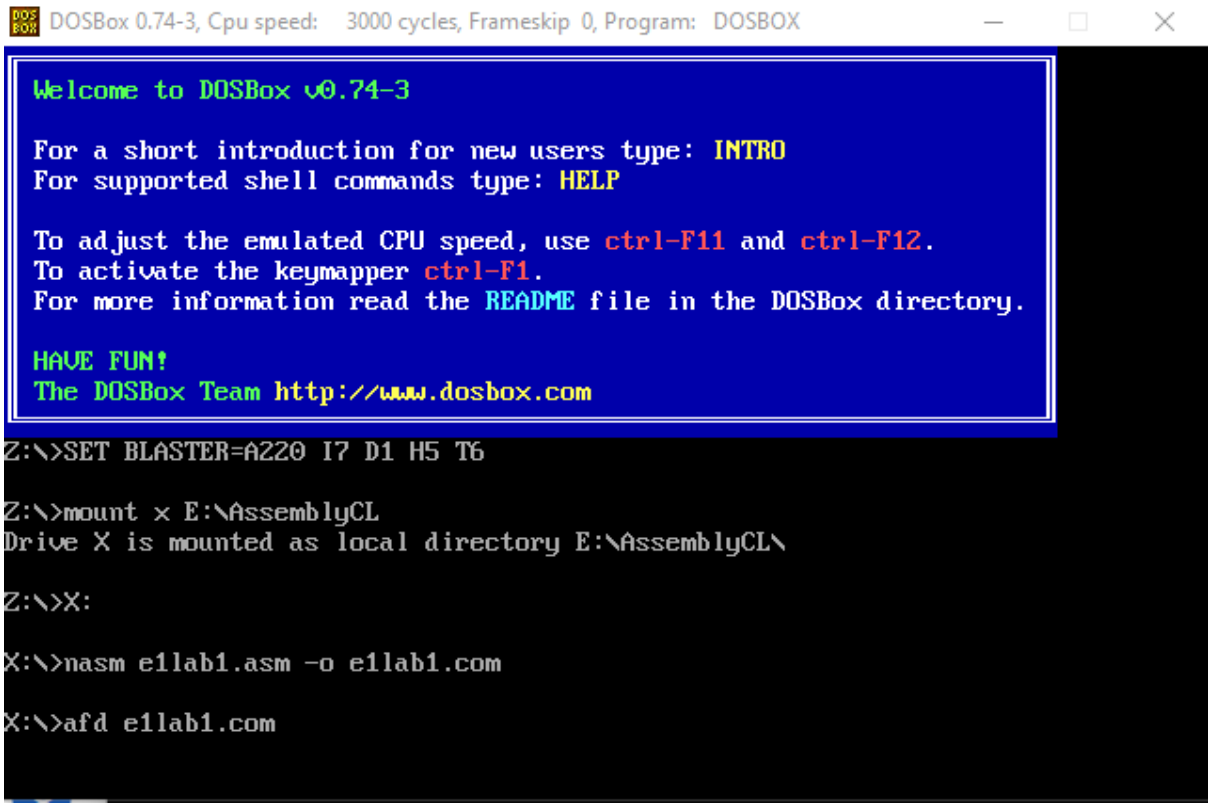
In summary, Linkers, Debuggers, Assemblers, and Compilers are essential tools in software development:

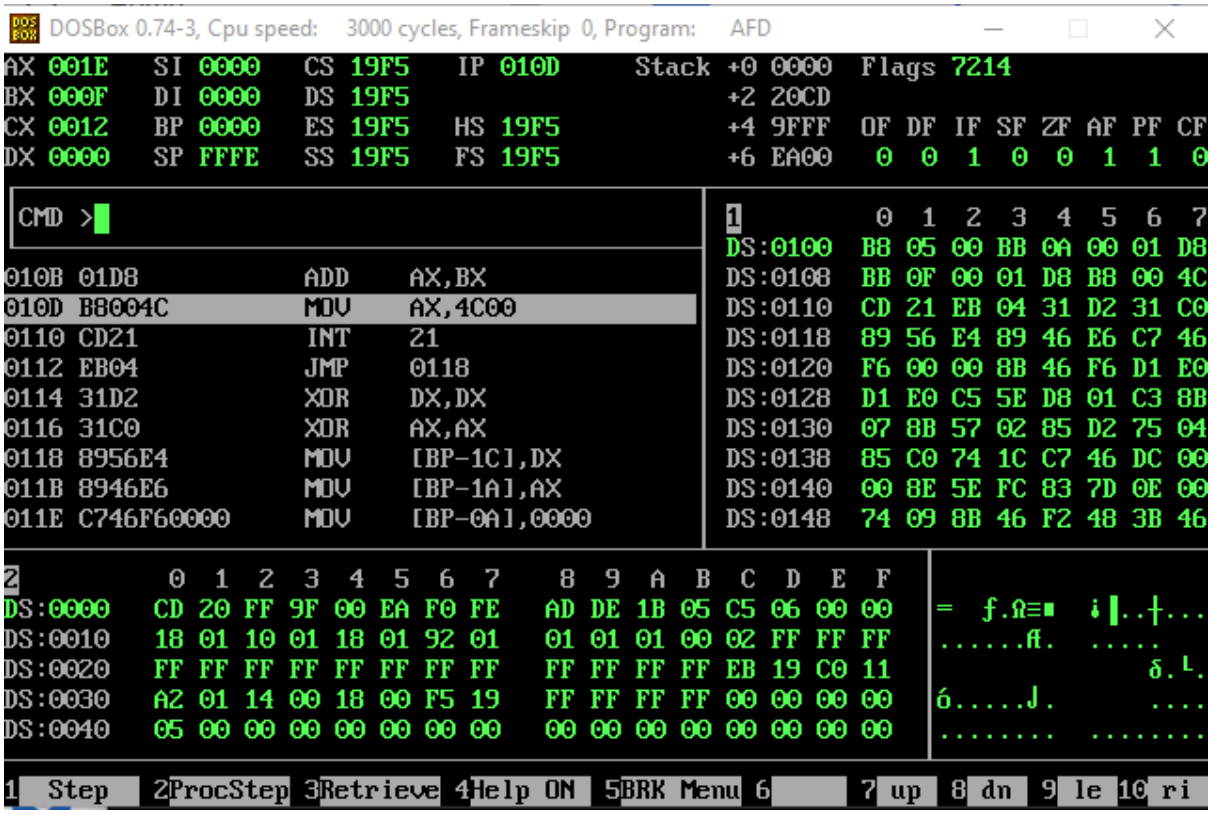
- **Linkers** help combine code modules and resolve references.
- **Debuggers** assist in identifying and fixing errors in code during development and testing.
- **Assemblers** translate human-readable assembly language into machine code.
- **Compilers** translate high-level code into machine code for execution.

These tools make it possible to create, maintain, and debug software across various programming languages and computer architectures.

Lab Work

```
1  [org 0x0100]
2
3      mov ax, 5          ;load first number in ax
4      mov bx, 10         ;load second number in bx
5      add ax, bx         ;accumulate sum in ax
6      mov bx, 15         ;load third number in bx
7      add ax, bx         ;accumulate sum in ax
8
9      mov ax, 0x4c00     ;terminate program
10     int 0x21
```


DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Program: DOSBOX
Welcome to DOSBox v0.74-3
For a short introduction for new users type: INTRO
For supported shell commands type: HELP
To adjust the emulated CPU speed, use ctrl-F11 and ctrl-F12.
To activate the keymapper ctrl-F1.
For more information read the README file in the DOSBox directory.
HAVE FUN!
The DOSBox Team http://www.dosbox.com
Z:\>SET BLASTER=A220 I7 D1 H5 T6
Z:\>mount x E:\AssemblyCL
Drive X is mounted as local directory E:\AssemblyCL\
Z:\>X:
X:\>nasm e1lab1.asm -o e1lab1.com
X:\>afd e1lab1.com


DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Program: AFD
AX 001E SI 0000 CS 19F5 IP 010D Stack +0 0000 Flags 7214
BX 000F DI 0000 DS 19F5 +2 20CD
CX 0012 BP 0000 ES 19F5 HS 19F5 +4 9FFF OF DF IF SF ZF AF PF CF
DX 0000 SP FFFE SS 19F5 FS 19F5 +6 EA00 0 0 1 0 0 1 1 0
CMD >
010B 01D8 ADD AX,BX
010D B8004C MOV AX,4C00
0110 CD21 INT 21
0112 EB04 JMP 0118
0114 31D2 XOR DX,DX
0116 31C0 XOR AX,AX
0118 8956E4 MOV [BP-1C],DX
011B 8946E6 MOV [BP-1A],AX
011E C746F60000 MOV [BP-0A],0000
DS:0100 B8 05 00 BB 0A 00 01 D8
DS:0108 BB 0F 00 01 D8 B8 00 4C
DS:0110 CD 21 EB 04 31 D2 31 C0
DS:0118 89 56 E4 89 46 E6 C7 46
DS:0120 F6 00 00 8B 46 F6 D1 E0
DS:0128 D1 E0 C5 5E D8 01 C3 8B
DS:0130 07 8B 57 02 85 D2 75 04
DS:0138 85 C0 74 1C C7 46 DC 00
DS:0140 00 8E 5E FC 83 7D 0E 00
DS:0148 74 09 8B 46 F2 48 3B 46
DS:0000 CD 20 FF 9F 00 EA F0 FE AD DE 1B 05 C5 06 00 00
DS:0010 18 01 10 01 18 01 92 01 01 01 01 00 02 FF FF FF
DS:0020 FF FF FF FF FF FF FF FF FF FF FF FF EB 19 C0 11
DS:0030 A2 01 14 00 18 00 F5 19 FF FF FF FF 00 00 00 00
DS:0040 05 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
1 Step 2ProcStep 3Retrieve 4Help ON 5BRK Menu 6 7 up 8 dn 9 le 10 ri