2.5. ASSEMBLY LANGUAGE PROGRAMMING USING MASM SOFTWARE

Machine language: Instructions the CPU actually runs — just 0s and 1s.

(e.g., 0110000 01100001).

Assembly language: A symbolic version of those instructions, easier for humans to read and write (e.g., MOV AL, 61h).

Assembler: A program that converts assembly language into machine code.

Assembly language uses mnemonic codes and symbols to represent the instructions and data elements that need to be processed by the computer. Assembly language programs are written using a text editor and saved with a file extension like .asm.

This software used to write a program (8086, Pentium processors etc.) The programs are written using assembly language in editor then compile it. The complier converts assembly language statements into machine language statements/checks for errors. Then execute the compiled program.

There are different soft wares developed by different companies for assembly language programming.

They are

- MASM - Microsoft Company
- TASM - Bore Land Company

1. MASM (Microsoft Macro Assembler): MASM is an x86 assembler developed by Microsoft.

- It is commonly used for developing software on Windows platforms.
- MASM supports a rich set of features, including macros, conditional assembly, and advanced debugging capabilities.
- It is often used in conjunction with the Microsoft Visual Studio development environment.

2. TASM (Turbo Assembler):

TASM is an x86 assembler that was originally developed by Borland and later acquired by Embarcadero Technologies.

- It is known for its compatibility with the Turbo Pascal programming language.
- TASM provides a comprehensive set of features and supports various modes, including real mode, protected mode, and long mode.
- It is widely used for DOS-based applications and legacy systems.

MERITS OF MASM:

1. produces binary code
2. Referring data items by their names rather than by their address.

Data Types

Assembly language programming does not have explicit data types.Data types are implicitly determined based on how the data is used or interpreted by the instructions in the program.

Integer: Integers are the most commonly used data type in assembly language programming. They can be represented using various sizes, such as 8-bit (byte), 16-bit (word), 32-bit (double word), or 64-bit (quad word). Integer values can be manipulated using arithmetic and logical instructions.

Floating-point: Floating-point numbers represent real numbers with fractional parts. Floating-point operations involve calculations with numbers in scientific notation, consisting of a sign, mantissa, and exponent. Assembly language provides instructions for performing floating-point arithmetic and conversions.

Character: Characters are typically represented using ASCII (American Standard Code for Information Interchange) encoding, where each character is assigned a unique 8-bit value. Assembly language instructions can manipulate individual characters or process strings of characters.

Boolean: Boolean data represents true or false values. In assembly language, logical operations such as AND, OR, and XOR can be used to manipulate Boolean values stored as binary digits (0 or 1).

Pointers: Pointers are memory addresses that refer to other data in the program. They are commonly used for accessing and manipulating data structures, arrays, and

dynamically allocated memory. Pointers typically have the same size as the system's address bus.

Packed and unpacked decimal: Packed and unpacked decimal formats are used for representing decimal numbers in assembly language. Packed decimal compresses multiple decimal digits into fewer bytes, while unpacked decimal stores each digit in a separate byte.

Binary: Binary data represents values in base 2 using only two digits, 0 and 1. Assembly language provides instructions for performing bitwise operations and manipulating binary data.

Execution commands

For the 8086 microprocessor, the execution of an assembly language program typically involves the following commands:

1.MOV (Move): The MOV instruction is used to move data between registers, memory locations, and immediate values. It copies the value from the source operand to the destination operand.

2. ADD (Addition): The ADD instruction performs addition between two operands, storing the result in the destination operand. It can operate on registers, memory locations, and immediate values**.**

3. SUB (Subtraction): The SUB instruction subtracts the second operand from the first operand, storing the result in the destination operand. Like ADD, it can operate on registers, memory locations, and immediate values.

4. CMP (Compare): The CMP instruction compares two operands by subtracting the second operand from the first operand but does not store the result. It sets the flags based on the comparison result, which can be used by subsequent conditional jump instructions.

5. JMP (Jump): The JMP instruction transfers program control to a specified location. It allows unconditional branching to a new location in the program, specified by a label or a memory address.

6. Jcc (Conditional Jump): Jcc instructions provide conditional branching based on the state of the flags register. They allow program flow to be controlled based on specific conditions, such as equal, not equal, greater than, less than, etc.

7. CALL (Call Procedure): The CALL instruction is used to call a subroutine or procedure. It saves the return address on the stack and transfers control to the specified procedure. The RET (Return) instruction is used to return from the called procedure.

8. LOOP (Loop): The LOOP instruction performs a loop operation. It decrements the loop counter (usually stored in a register) and jumps to a specified label if the counter is not zero. It provides a convenient way to repeat a block of code.

9. INT (Interrupt): The INT instruction generates a software interrupt, triggering a specific interrupt handler routine. It is used for invoking operating system services or handling exceptional conditions**.**

ALP Tools

1.Editor: An editor is used to write assembly language source code. Popular text editors like Notepad++, Sublime Text, Vim, or Emacs can be used to write and edit assembly language programs.

2. Assembler: An assembler is a software tool that converts assembly language source code into machine code.

For the 8086 microprocessor, an assembler like NASM (Netwide Assembler), MASM (Microsoft Macro Assembler), or TASM (Turbo Assembler) can be used to assemble the assembly code into object files or executables.

3. Linker: A linker is used to combine object files generated by the assembler into a single executable file. It resolves symbol references, assigns memory addresses, and creates the final executable. Linkers such as the Microsoft Linker (LINK), the GNU linker (ld), or the Borland Linker (TLINK) can be used.

4. Debugger: A debugger is a crucial tool for testing and debugging assembly language programs. It allows step-by-step execution of the program, setting breakpoints, examining and modifying registers and memory, and analyzing program flow. Debuggers like WinDbg, OllyDbg, GDB (GNU Debugger), or Turbo Debugger (TD) can be used for debugging assembly language programs.

5. Simulator/Emulator: Simulators or emulators provide a virtual environment for running and testing assembly language programs without the need for physical hardware.

These tools simulate the behaviour of the 80386 microprocessor, allowing you to test your code on different platforms or operating systems. QEMU, Bochs, and DOSBox are some examples of simulators/emulators that can be used for assembly language development.

6. Performance Profilers: Performance profilers help analyze the performance of assembly language programs by identifying bottlenecks, measuring execution times, and providing insights into code optimization. Tools like Perf, Valgrind, or Intel VTune can be used to profile assembly language programs.

Structure of ALP

The structure of an assembly language program typically consists of several components. Here is a basic structure that is commonly followed:

1. Program Header: The program header includes comments that provide information about the program, such as its purpose, author, creation date, and any other relevant details. It serves as a documentation section for the program.

2. Include Directives: Include directives are used to include external files or libraries that contain pre-defined macros, constants, or other utility functions that will be used in the program. This allows reusability and modular programming.

3. Data Segment (optional): The data segment is used to declare and initialize variables, constants, and data structures that will be used in the program. This section typically includes instructions for defining memory locations and assigning initial values to variables.

4. Code Segment: The code segment contains the main body of the program, where the actual assembly language instructions are written. It includes labels, instructions, and control flow structures that define the program's logic and operations.

5. Procedures and Functions (optional): Assembly language programs may include procedures or functions, which are reusable blocks of code that perform specific tasks. These procedures can be called from the main program or other parts of the code.

6. End of Program

Sample Program

```
; Program Header
 ; Author: John Doe
 ; Date: May 17, 2023
 ; Description: This program demonstrates a simple addition operation.
 ; Include Directives
 include macros.inc
 include io.inc
; Data Segment
 .data
message db 'Hello, World!', 0
    num1 dd 10
    num2 dd 20
     result dd ?
 ; Code Segment
.code
main:
; Display a message
 mov ah, 09h
  lea dx, message
  int 21h
; Add two numbers
mov eax, num1
 add eax, num2
 mov result, eax
```

```
; Display the result
    display_int result
    ; Exit the program
    exit_program
; End of Program
```

some common functions often used in ALP:

- Input and Output: Displaying output to the console: Functions like printf or puts can be used to print text or formatted data to the console.
- Reading input from the user: Functions like scanf or gets can be used to read input from the user.

Arithmetic Operations:

- Addition: The add instruction performs addition between two operands and stores the result.
- Subtraction: The sub instruction subtracts the second operand from the first operand and stores the result.
- Multiplication: The mul instruction multiplies two operands and stores the result.
- Division: The div instruction divides the first operand by the second operand and stores the quotient and remainder.

Control Flow:

- Branching: Instructions like jmp (unconditional jump) and conditional jumps (je, jne, jl, jg, etc.) are used for branching and changing the flow of execution based on certain conditions.
- Loops: Instructions like loop or conditional jumps can be used to create loops and repeat a block of code.

Memory Operations:

- Loading and storing values: Instructions like mov are used to move data between registers and memory locations.

- Accessing arrays and structures: You can use indexing and addressing modes to access elements of arrays and structures in memory.

String Manipulation:

- Copying strings: Functions like strcpy or instructions like movsb can be used to copy strings from one memory location to another.
- Concatenating strings: Instructions like strcat or a combination of movsb

System Calls:

- Interacting with the operating system: System calls provide access to operating system services like file I/O, process management, memory allocation, etc. The specific system call numbers and calling conventions depend on the operating system and the processor architecture being used.