Experiment No:1

FAMILIARISATION OF x86 ASSEMBLY LANGUAGE PROGRAMMING USING TASM/MASM/NASM

Aim:

To familiarise with assembler directives, addressing modes and memory modes.

Assembly Language

Assembly language is a low-level programming language closely associated with a computer's architecture. x86 Assembly is written specifically for Intel's 8086 and later processors, and it provides direct control over hardware using simple mnemonics and machine-level instructions.

x86 Architecture Overview

The Intel x86 architecture is a CISC (Complex Instruction Set Computer) architecture introduced with the 8086 processor. It uses a segmented memory model, supports 16-bit registers, and is backward compatible with earlier Intel processors. Key features include:

- 16-bit data and address bus (8086).
- 20-bit physical address space allowing access to 1MB of memory.

- Use of memory segmentation for logical addressing.
- Includes General Purpose Registers, Segment Registers, and Flags.

Assemblers

An assembler converts assembly language source code (.ASM) into machine code (.OBJ, .EXE). Popular x86 assemblers include:

- TASM (Turbo Assembler) - Borland's assembler, often used with DOSBox
- MASM (Microsoft Macro Assembler) - Official Microsoft assembler.
- EMU8086 - Educational assembler with an emulator for running 8086 code.
- NASM (Netwide Assembler) - used in Windows as well as Linux.

Assembler Workflow

Source Code (.ASM) → Assembler → Object File (.OBJ) → Linker → Executable (.EXE)

Registers in x86

Registers are small, fast storage locations in the CPU used for operations.

General Purpose Registers (16-bit):

Register	Description
AX	Accumulator
BX	Base
CX	Counter
DX	Data

Each of these can be spilt into 8-bit halves

. AX → AH (high) and AL (low), etc

Segment Registers

Register	Description
CS	Code Segment
DS	Data Segment
SS	Stack Segment
ES	Extra Segment

Pointer and Index Registers:

- SP (Stack Pointer), BP (Base Pointer)
- SI (Source Index), DI (Destination Index).

Memory Segmentation

x86 architecture uses segmented memory

A physical address = Segment × 16 + Offset

Segments:

- Code Segment (CS): Contains the instructions to be executed.
- Data Segment (DS):- Stores variables and constants
- Stack Segment (SS): Manages function calls and local variables.
- Extra Segment (ES): Used in string and memory operations

Assembler Directives

Assembler directives are commands to the assembler and do not generate machine code.

Common directives

Directive	Meaning
.model	Defines memory model (eg: small, tiny)
.data	Starts the data segment
.code	Starts the code segment

.stack	Reserves stack space
end	Ends the program

Addressing Modes

Addressing modes define how operands are accessed.

Mode	Example	Description
Immediate	mov al, 25	Operand is not a constant value
Register	mov ax, bx	Operand is a register
Direct	mov al, [1234h]	Operand is at memory address
Register Indirect	mov al, [bx]	Memory address in a register
Indexed	mov al, [si]	Uses index registers
Based Indexed	mov al, [bx+si]	Combines base and index

Memory models in x86

Memory models define how code and data are organised

Model	Code	Data	Notes
Tiny	1	1	Code and data in one segment
Small	1	1	Code and data in separate segments
Medium	Many	1	Multiple code segments
Compact	1	Many	Multiple data segments
Large	Many	Many	Multiple code and data segments

For simpler programs, .model small is most commonly used.

RESULT:

~~Successfully~~ familiarised with x86 programming.

10:25

ChatGPT

📌 Algorithm (8 Lines)

1. Start the program and initialize data segment.
2. Display the message: "Enter two characters".
3. Read the first character from the keyboard.
4. Store the first character in memory.
5. Read the second character from the keyboard.
6. Store the second character in memory.
7. Display the message: "You entered: " followed by the two characters.
8. Terminate the program.

Let me know if you also want a **flowchart** or **block diagram**.

Ask anything

✅ Algorithm (10 Lines)

1. Start the program and initialize the data segment.
2. Declare an 8-bit value `0x7A` using `DB`.
3. Declare a 16-bit value `0x3456` using `DW`.
4. Declare a 32-bit value `0x11223344` using `DD`.
5. Load the 8-bit value into register `AL`.
6. Increment `AL` and store it back in memory.
7. Load the 16-bit value into register `AX`.
8. Increment `AX` and store it back in memory.
9. Load the 32-bit value into register `EAX`, increment it, and store it back.
10. Terminate the program using DOS interrupt `INT 21h`.

10:26

ChatGPT

Here is the **8-line algorithm** to **add two 8-bit numbers** in Assembly (MASM):

📌 Algorithm to Add Two 8-bit Numbers (8 Lines)

1. Start the program and initialize the data segment.
2. Load the first 8-bit number into register AL.
3. Load the second 8-bit number into register BL.
4. Add the contents of BL to AL (AL = AL + BL).
5. Check for carry if needed (optional step).
6. Store or display the result from AL.
7. Terminate the program.
8. End.

Ask anything