

**Department of CSE**

**Microprocessors & Microcontrollers Lab (CSE 360)**

**LAB REPORT 1**

**Submitted By :**

**Fahima Abida Chowdhury**

**ID: 0432220005101135**

**Semester: Spring-2025**

**Batch: 52(6C1)**

**Submitted To :**

**Md. Ismail**

**Lecturer**

**Department of CSE**

**University of Information Technology and Sciences**

**Types of Registers in a Microprocessor & Their Applications**

Registers in a microprocessor are small storage locations used for fast data access and processing. They can be classified based on their functions. In 8086 microprocessor, the registers are categorized under four types:

1. General Purpose Registers
2. Segment Registers
3. Special Purpose Registers
4. Flag Registers

**General Purpose Registers**

General-purpose registers are versatile and store data or memory addresses. They include AH, AL, BH, BL, CH, CL, DH, DL (8-bit each). For 16-bit operations, they form pairs: AX, BX, CX, DX, doubling storage capacity. These registers aid in arithmetic, memory, and control tasks.

The roles of each general-purpose register are described below:

*Accumulator Register (AX):* Stores operands for arithmetic and logical operations.

Example: Used in multiplication, division, and input/output operations.

*Base Register (BX):* Holds base memory address for data access.

Example: Used in indexed addressing mode to access arrays.

*Counter Register (CX):* Stores loop count and shift/rotate values.

Example: Used in LOOP instruction for iterative tasks.

*Data Register (DX):* Holds MSB of 32-bit results in multiplication/ division.

Example: Used in I/O port addressing and multiplication operations.

These registers enable efficient execution of arithmetic, memory, and control operations in 8086 microprocessor.

**Segment Registers**

The Bus Interface Unit (BIU) manages memory access using four 16-bit segment registers: CS, DS, SS, and ES. Each holds the starting address of a memory segment, enabling efficient data access during execution.

Functions of segment registers:

*Code Segment (CS):* Stores the address of the program code.

Example: Fetching instructions for execution.

*Data Segment (DS):* Holds frequently used program data.

Example: Accessing variables in memory.

*Stack Segment (SS):* Manages stack operations during execution.

Example: Storing return addresses in function calls.

*Extra Segment (ES):* Provides additional memory for data storage.

Example: Handling string operations.

**Special Purpose Registers**

The index and pointer registers serve as memory pointers, generating 20-bit physical addresses. The five 16-bit registers are SP, BP, SI, DI, IP.

Functions of special purpose registers:

*Stack Pointer (SP):* Holds the top address of the stack.

Example: Managing function calls.

*Base Pointer (BP*): Stores the base memory address.

Example: Accessing local variables in stack.

*Source Index (SI):* Holds the source offset address.

Example: String operations using MOVS instruction.

*Destination Index (DI):* Holds the destination offset address.

Example: Used in string manipulations.

*Instruction Pointer (IP):* Stores the address of the next instruction.

Example: Controls program flow execution.

**Flag Registers**

The flag register is a 16-bit register containing 9 flags, categorized as:

*1.* *Conditional Flags (ALU result indicators)*

* Carry Flag (CF) – Set if an arithmetic operation generates a carry.
* Auxiliary Carry Flag (AF) – Set if a carry occurs from the 4th bit.
* Parity Flag (PF) – Set if the result has even parity.
* Zero Flag (ZF) – Set if the result is zero.
* Sign Flag (SF) – Set if the result is negative.
* Overflow Flag (OF) – Set if the result exceeds register capacity.

*2*. *Control Flags (Processor operation control)*

* Trap Flag (TF) – Enables single-step execution for debugging.
* Interrupt Flag (IF) – Controls maskable interrupts.
* Direction Flag (DF) – Determines string operation direction.

These flags help in arithmetic operations, decision-making, and processor control.

**Fundamental operation code for assembly language**

An assembly program can be divided into three sections −

* The data section,
* The bss section, and
* The text section.

**The data Section** -

The data section is used for declaring initialized data or constants. This data does not change at runtime. We can declare various constant values, file names, or buffer size, etc., in this section.

The syntax for declaring data section is – section.data

**The bss Section** - The bss section is used for declaring variables.

The syntax for declaring bss section is − section.bss

**The text section** -

The text section is used for keeping the actual code. This section must begin with the declaration global \_start, which tells the kernel where the program execution begins.The syntax for declaring text section is –

section.text

global \_start

\_start:

**Comments**

Comments in assembly language can be marked by putting ; before their beginning.

The snippet below demonstrates how to add comments to assembly code:

mov value,8 ;

**Assembly Language Statements**

Assembly programs include three types of statements:

1. Instructions – Perform operations (e.g., MOV, ADD).
2. Directives (Pseudo-ops) – Guide the assembler (e.g., DB, SECTION).
3. Macros – Define reusable code blocks.

**Assembly Language Instructions**

An assembly instruction consists of:

1. Instruction Name – Specifies the operation (e.g., MOV, ADD).
2. Instruction Parameters – Data or addresses involved in the operation.

Example:

MOV var, 8 ; Moves the value 8 into 'var'

Here, MOV is the instruction name, and var, 8 are operands (perameters). This instruction updates var to 8.

**Assembly Directives**

Assembly directives help control the assembly process, such as allocating memory, including libraries, and debugging.

Example:

section .data

message db 'Welcome to Educative!', 0xA ; Defines a string with a newline

Here, db (define byte) allocates memory for message.

**Macros**

Macros are basically a text substitution mechanism.