Islamic University of Technology

Department of Computer Science and Engineering (CSE)

CSE 4622: Microprocessor and Interfacing Lab

Summer, 2023-2024

Md Farhan Ishmam, Lecturer, CSE
Aashnan Rahman, Jr. Lecturer, CSE

Lab Manual

Contents

1 Assembly Language

Assembly language is a low-level programming language that provides a symbolic representation of a computer's machine code instructions. Unlike high-level languages like Python or C, assembly language is closely tied to a computer's architecture, offering direct control over hardware and memory. Each instruction in assembly corresponds almost one-to-one with an operation the CPU can perform, making it ideal for tasks requiring high performance or precise hardware manipulation.

In this lab, you will learn how to write, assemble, and execute basic assembly programs. You'll work with core concepts such as registers, memory addressing, control flow, and data manipulation. Understanding assembly language not only helps in systems programming and debugging but also deepens your knowledge of how software interacts with hardware.

2 Microprocessor

A microprocessor is the central unit of a computer system that performs arithmetic and logic operations, controls data flow, and executes program instructions. It acts as the "brain" of the system, processing input and producing output based on a set of instructions stored in memory.

Microprocessors consist of essential components such as the Arithmetic Logic Unit (ALU), control unit, and registers. They operate using the fetch-decode-execute cycle, carrying out instructions one step at a time.

Nowadays, microprocessors like Intel Core i9, AMD Ryzen, and Apple M-series chips are extremely powerful, capable of executing billions of instructions per second and handling complex tasks like machine learning, 3D rendering, and real-time data processing. Despite these advancements, understanding the foundational concepts is crucial.

In this lab, we will work with the Intel 8086 microprocessor. Although it is an older processor, it provides a simplified and clear architecture that is ideal for learning the basic principles of microprocessor operation and assembly language programming.

3 Microprocessor and Interfacing

The key components of microprocessor and interfacing are-

1. **Data Bus**
   The **data bus** is used to transfer data between the microprocessor, memory, and I/O devices.
   - It is **bidirectional**.
   - The width (e.g., 8, 16, 32, 64 bits) determines how much data can be transferred simultaneously.
2. **Address Bus**
   The **address bus** carries the address of the memory location that the CPU intends to access.
   - It is **unidirectional** (from CPU to memory/I/O).

- The width determines the addressable memory. For example, a 16-bit address bus can address $2^{16} = 65,536$ locations.

3. **Registers**

4. **Registers** are small, fast storage locations within the microprocessor.
   - **Accumulator (ACC)** – stores intermediate arithmetic/logic results.
   - **Program Counter (PC)** – holds the address of the next instruction.
   - **Instruction Register (IR)** – stores the current instruction.
   - **General-purpose Registers (R0, R1, . . .)** – used for temporary storage.

The communication between the microprocessor, I/O ports, and memory is illustrated in Figure 1

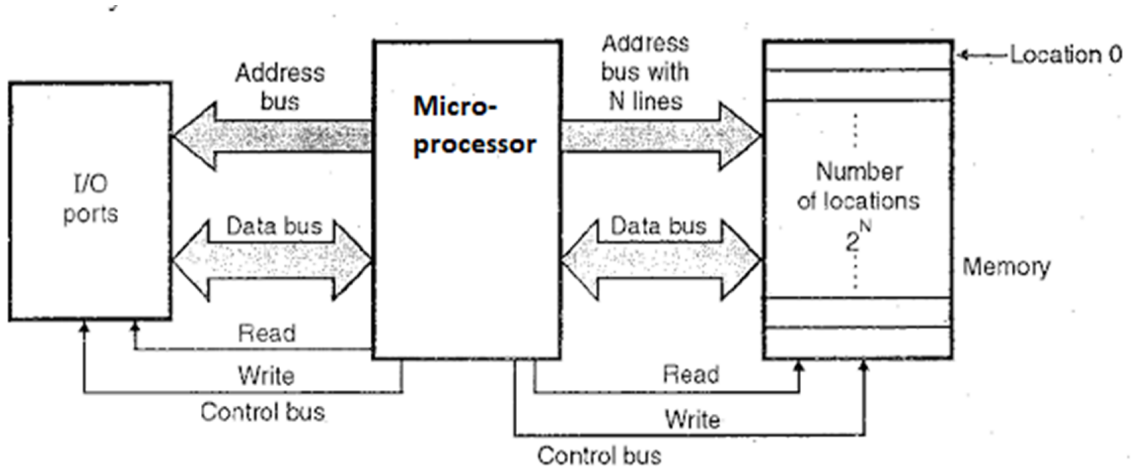

Figure 1: Microprocessor and Interfacing

4 Intel 8086 Microprocessor

The Intel 8086 is a 16-bit microprocessor introduced by Intel in 1978. It laid the foundation for the x86 architecture, which is still used in modern processors today. The 8086 was widely adopted in early personal computers and is known for its simple yet powerful architecture, making it an ideal platform for learning microprocessor fundamentals.

It follows a CISC (Complex Instruction Set Computer) architecture, meaning it supports a wide range of instructions, including complex ones that operate directly on memory or multiple data types.

4.1 Features

- **16-bit Processor:** It has a 16-bit ALU, registers, and data bus.
- **20-bit Address Bus:** Can address up to 1 MB of memory (2^{20} = 1,048,576 bytes).

- **Segmented Memory Architecture:** Uses segment registers (CS, DS, SS, ES) to divide memory into segments, simplifying memory management.
- **16-bit Registers:** Includes general-purpose registers (AX, BX, CX, DX), segment registers, pointer and index registers.
- **Instruction Queue:** Features a 6-byte instruction queue (pipelining), allowing prefetching of instructions and improving execution speed.
- **Clock Speed:** Initially ran at 5 MHz to 10 MHz.
- **40-pin DIP Package:** The chip comes in a 40-pin Dual In-line Package.
- **Supports Multiprocessing:** Can be configured as a master or slave in multiprocessor systems.

5 8086 Block Diagram

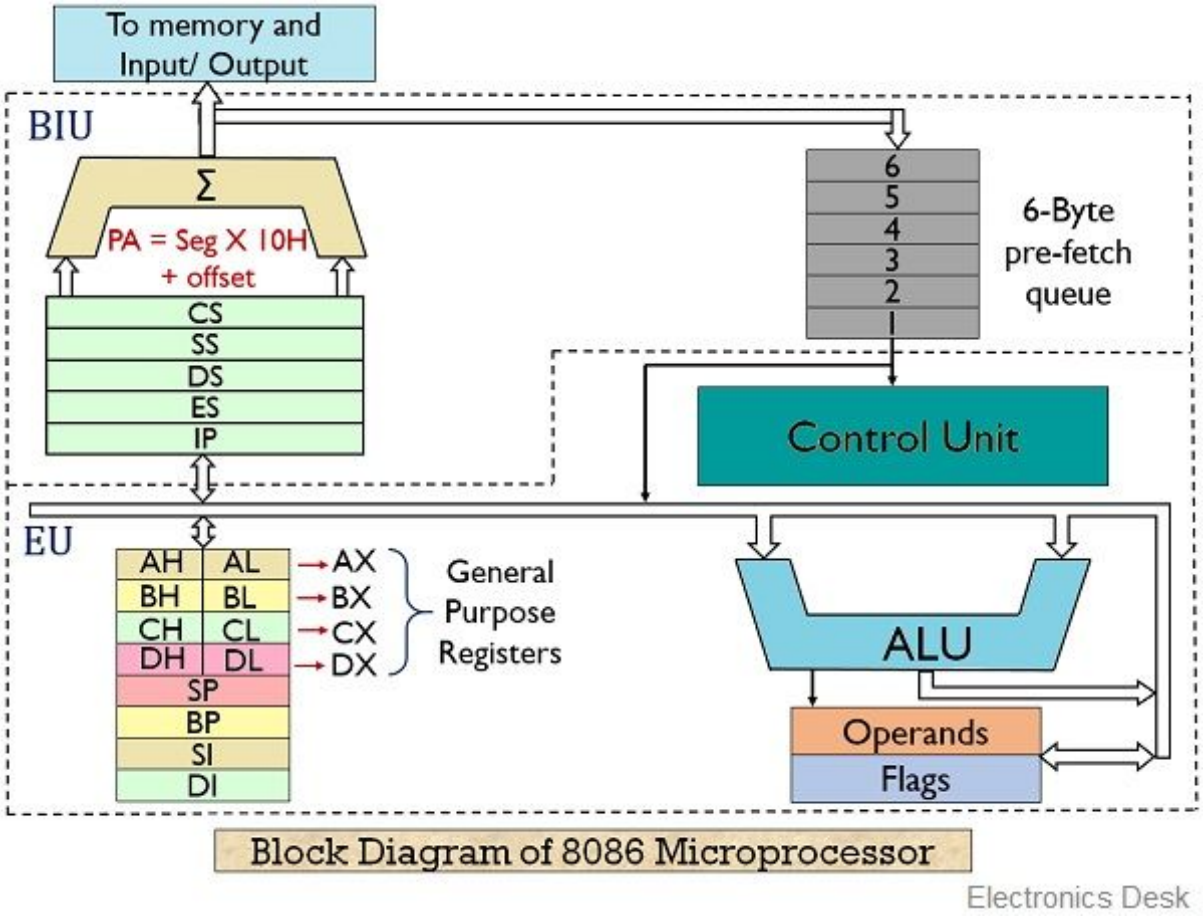

Figure 2: 8086 Block Diagram

5.1 Registers

Category	Bits	Register Names	Count
General	16 8	AX, BX, CX, DX AH, AL, BH, BL, CH, CL, DH, DL	4
Pointer	16	SP (Stack Pointer) BP (Base Pointer)	2
Index	16	SI (Source Index) DI (Destination Index)	2
Segment	16	CS (Code Segment) DS (Data Segment) SS (Stack Segment) ES (Extra Segment)	4
Instruction	16	IP (Instruction Pointer)	1
Flag	16	FR (Flag Register)	1

Table 1: 8086 Microprocessor Register Organization

Segment Register	**Offset**	**What It Stores**
CS (Code Segment)	IP (Instruction Pointer)	Base address of the code segment. The offset (IP) points to the next instruction to execute.
DS (Data Segment)	SI(Source Index) or any offset within DS (e.g., variable)	Base address of the data segment. The offset is used to reference specific data in memory (e.g., variables, constants).
SS (Stack Segment)	SP (Stack Pointer) or BP (Base Pointer)	Base address of the stack segment. SP points to the top of the stack, and BP is used to reference function parameters and local variables.
ES (Extra Segment)	Any offset within ES (e.g., string data)	Additional data segment used for string operations, memory buffers, etc. The offset refers to data in the extra segment.

Table 2: Segment, Offset, and Their Roles in the 8086 Microprocessor

5.2 Flag Registers

The flag register in the 8086 microprocessor is a 16-bit register. It reflects the status of arithmetic/logical operations and controls certain processor behaviors. Only 9 out of 16 bits are used; the rest are reserved. Flags are categorized into:

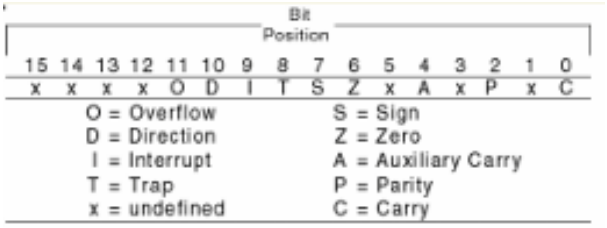

Figure 3: Flag Register

1. **Status Flags**
   Indicate the result of operations.
   - **CF (Carry Flag)** – Set if there is a carry/borrow out of the most significant bit.
   - **PF (Parity Flag)** – Set if the result has even parity (even number of 1s).
   - **AF (Auxiliary Carry Flag)** – Set if there is a carry/borrow from bit 3 to bit 4.
   - **ZF (Zero Flag)** – Set if the result of an operation is zero.
   - **SF (Sign Flag)** – Set if the result is negative (MSB is 1).
   - **OF (Overflow Flag)** – Set if signed overflow occurs.
2. **Control Flags**
   Affect the operation of the CPU.
   - **TF (Trap Flag)** – Enables single-step debugging when set.
   - **IF (Interrupt Flag)** – Enables or disables maskable interrupts.
   - **DF (Direction Flag)** – Controls direction for string operations (increment/decrement).

Summary Table
Below is a table summarizing each flag:

Flag	Bit	Purpose
CF (Carry Flag)	0	Set if there is an unsigned carry or borrow.
PF (Parity Flag)	2	Set if the number of 1s in the result is even.
AF (Auxiliary Carry)	4	Used in BCD operations, set if there is carry between nibbles.
ZF (Zero Flag)	6	Set if the result is zero.
SF (Sign Flag)	7	Set if the result is negative.
TF (Trap Flag)	8	Enables single-step execution for debugging.
IF (Interrupt Flag)	9	Enables/disables maskable interrupts.
DF (Direction Flag)	10	Determines direction of string operations.
OF (Overflow Flag)	11	Set on signed overflow.

Table 3: 8086 Flags and Their Functions

6 Instructions

6.1 Data Transfer Instruction

MOV *Destination*, *Source*

Direct (Registers):

Move contents of one register to another register

```
MOV AL, BL
MOV AX, BX
```

Direct (Variables):

Move contents of the variable named COUNT to a register

```
MOV DL, COUNT ; here COUNT is a 8-bit variable
MOV DX, COUNT ; here COUNT is a 16-bit variable
```

Immediate:

Load a register with an immediate value or equivalent binary/hexa-decimal

```
MOV CL, 240
MOV CL, 11110000B
MOV CL, 0F0H
MOV CX, 256
MOV CX, 0000000100000000B
MOV CX, 0100H
```

6.2 Arithmetic / Logic Instructions:

Arithmetic and logic instructions can be performed on 8-bit (byte) and 16-bit values.

1. **Increment**
   - Increment the contents of a register by a value (decimal/binary/hexa-decimal)

     ```
     ADD AX, 4
     ```

   - Add the contents of a register with the contents of another register

     ```
     ADD AX, BX
     ```

2. **Subtract**
   - Subtract a value (decimal/binary/hexa-decimal) from the contents of a register

     ```
     SUB DL, 4
     ```

   - Subtract the contents of a register from the contents of another register

     ```
     SUB DX, CX
     ```

3. **Multiply**
   Multiply AX by BL, the result will be in AX
   ```
   MUL BL
   ```

4. **Divide**
   Divide the contents of AX register with the value of CL and store the result in AX
   ```
   DIV CL
   ```

5. **Increment/Decrement** Increase or Decrease the contents of BX register by 1
   ```
   INC BX ; Increase
   DEC BX ; Decrease
   ```

6. **Clear**
   Clear the contents of AX register
   ```
   XOR AX, AX
   ```

7. **Negation**
   Clear the contents of AX register
   ```
   XOR AX, AX
   ```

7 Lab Task

7.1 Lab 1(A): Introduction to Assembly Language

For today's lab do the following tasks-

7.1.1 Getting Started

Either download the 8086 Emulator or use Online 8086 Compiler

7.1.2 Run Your First Assembly Code

```
ORG 0100h           ; Offset of the program in memory

.DATA
A DB 11             ; Variable A got a BYTE value 11
B DW 500            ; Variable B got a WORD value 500
SUM DW ?            ; Variable SUM is defined as a WORD variable without any value
DIFFERENCE DB ?     ; Variable DIFFERENCE is defined as a BYTE variable without any value
MULTIPLICATION DW ?
DIVISION DB ?

.CODE               ; Code Segment Starts

MAIN PROC           ; Initialize Data Segment Register
    MOV AL, 30      ; Move decimal 30 to AL register
    ADD AL, 15      ; Add decimal 15 to the content of AL and store the result in AL
MAIN ENDP           ; End Procedure

END MAIN            ; End MAIN
RET                 ; Return to DOS
```

7.1.3 Write Your First Assembly Code

Your task is to do the following using minimum number of registers and store your output.

- $(30 + 15) \times (575–225) + 210$
- $0FFFh \times 10h + 1111b$
- Convert 260 C(Celsius) to F(Fahrenheit) using the following expression and store in a variable F:

$$^\circ F = \left(^\circ C \times \frac{9}{5}\right) + 32$$