Lab # 4

# OBJECTive

Define basic functionality of Assembly language.

# Theory

# list of processor architecture:

The processor in a personal computer or embedded in small devices is often called a [microprocessor](https://whatis.techtarget.com/definition/microprocessor-logic-chip). That term simply means that the processor's elements are contained on a single integrated circuitry ([IC](https://whatis.techtarget.com/definition/integrated-circuit-IC)) [chip](https://whatis.techtarget.com/definition/microchip).

Many competitors processor are followed in the market:

* Reduced instruction set computer (RISC)
* ARM architecture
* MIPS architecture
* Intel x86-32bit etc

The main competitor in the processor market is [Intel](https://whatis.techtarget.com/definition/Intel) and we discussed the architecture of Intel in further section.

**ASSEMBLY LANGUAGE**

Assembly language is a machine specific programming language with a one-to-one correspondence between its statements and the computer’s native machine language. There are many different types of assembly language, each specific to a processor or processor family. IBM-PC assembly language refers to instruction recognized by a number of different microprocessors in the Intel family: 8086, 8088, 80186, 80286, 80386, 80486, and Pentium.

**USES:**

* Assembly language is used most often when either communicating with the operating system or directly accessing computer hardware.
* Secondly, assembly language is used to optimize certain critical areas of application programs to speed up their runtime execution.

**ASSEMBLER**

An assembler is a program that converts source code programs from the assembly language into machine language. The assembler can optionally generate a source- listing file with line numbers, memory addresses, source code statements and a cross-reference listing of symbols and variables used in a program.

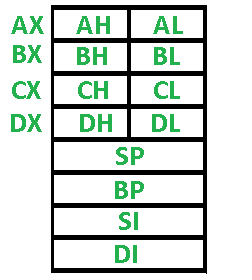
The most popular assemblers for the Intel family are MASM (Microsoft Assembler), TASM (Turbo Assembler).

**LINKER**

A companion program that combines individual files created by an assembler into a single executable file

GENERAL PURPOSE REGISTER IN 8086 MICROPROCESSOR

General purpose registers are used to store temporary data within the microprocessor. There are 8 general purpose registers in 8086 microprocessor.



****AX** –** This is the accumulator. It is of 16 bits and is divided into two 8-bit registers AH and AL to also perform 8-bit instructions. It is generally used for arithmetical and logical instructions but in 8086 microprocessor it is not mandatory to have accumulator as the destination operand.

****BX** –** This is the base register. It is of 16 bits and is divided into two 8-bit registers BH and BL to also perform 8-bit instructions. It is used to store the value of the offset.

****CX** –** This is the counter register. It is of 16 bits and is divided into two 8-bit registers CH and CL to also perform 8-bit instructions. It is used in looping and rotation.

****DX** –** This is the data register. It is of 16 bits and is divided into two 8-bit registers DH and DL to also perform 8-bit instructions. It is used in multiplication an input/output port addressing.

**ASSEMBLY PROGRAM SYNTAX**

* Assembly language program consists of statements.
* A statement is either an instruction to be executed when the program runs or a directive for the assembler.
* A program normally consists of three parts or segments.

**DIRECTIVE:**

Directives are instructions used by the assembler to help automate the assembly process and to improve program readability. Directives are used essentially in a pre-processing stage of the assembly process.

**.DATA SEGMENT**

* Variables are declared in the data segment.
* Each variable is assigned space in memory and may be initialized.

Exp:

* A DW 3501H

It sets memory for a variable called A, and initialize it to 3501H.

DW - Define word (16 bits = 2 memory locations)

* A DW (?) ; un- initialized variable

**.CODE SEGMENT**

* Program instructions are placed in the code segment. Instructions are actually organized into units called procedures. Every procedure starts with a line.

Exp:

* Main Proc;

Main is the name of procedure and PROC is the directive identify the start of the procedure

* Main Endp;

Main is again the name of the procedure and Endp is the directive identifies the end of the procedure

**.STACK SEGMENT**

* The stack segment is used for temporary storage of addresses and data. If no stack segment is declared, an error message is generated, so there must be a stack segment even if the program doesn’t utilize the stack.
* These segments begin with the directives .stack, .code, and .data

.MODEL

The .MODEL directive specifies the memory model for an assembler module that uses the simplified segment directives. The .MODEL directive must precede .CODE, .DATA, and .STACK. Note that near code is branched to (jumped to) by loading the IP register only, while far code is branched to by loading both CS and IP. Similarly, near data is accessed with just an offset, while far data must be accessed with a full segment: offset address. In short, far means that full 32-bit segment: offset addresses are used, while near means that 16-bit offsets can be used. The format of the .MODEL directive is:

.MODEL memory model[[,langtype]] [[,stackoption]]

**MEMORY MODEL:**

The memory model can be **TINY**, **SMALL**, **COMPACT**, **MEDIUM**, **LARGE**, **HUGE**, or **FLAT**. The langtype can be **C**, **BASIC**, **FORTRAN**, **PASCAL**, **SYSCALL**, or **STDCALL**. The stackoption can be **NEARSTACK** or **FARSTACK**.

|  |  |
| --- | --- |
| TINY | One segment. Thus both program code and data together must fit within the same 64 Kb segment. Both code and data are near. |
| SMALL | Program code must fit within a single 64 Kb segment, and data must fit within a *separate* 64 Kb segment. Both code and data are near. |
| MEDIUM | More than one code-segment. One data-segment. Thus code may be greater than 64K. |
| COMPACT | One code-segment. More than one data-segment. Thus data may be greater than 64K. |
| LARGE | More than one code-segment. More than one data-segment. No array larger than 64K. Thus both code and data may be greater than 64K. |
| HUGE | More than one code-segment. More than one data-segment. Arrays may be larger than 64K. Thus both code and data may be greater than 64K. |
| FLAT | No segmentation, all code and data can reach any location up to 4 Gb. |

All program models but TINY result in the creation of exe-format programs. The TINY model creates com-format programs.

PROGRAM SYNTAX

TITLE first program syntax

.Model Small ;specifies the memory model used

.Stack 100H ;allocate 100H memory locations for stack

.Data ;start of the data segment

; Data definitions here

A DB ?

……..

.Code ;start of the code segment

Main Proc ;start of the first procedure

; instructions here

……

Main Endp ;end of the first procedure

; Other procedures here

End Main ;end of the complete assembly program

**.STACK:** Defines the size of stack used in program

**.DATA:**  Defines the data segments for data used in the program. Mark the beginning of the data segment

**.CODE:**  Identifies the code segment which contains all the statements. Also .code marks the beginning of the code segment.

**PROC:** Beginning of the procedure

**ENDP:** End of the procedure

**END:** End of assembly language program