# Lab 4: Syntax and Semantics

### Objective

- To learn about syntax and semantics in assembly language
- To learn about code, data, and other segments in assembly language

### Introduction

The assembly code is very different from other high level programming languages. We have to do a lot of tasks manually, that high level language manages automatically. There are some directives in Assembly Language, to be defined at the start of program.

### Activity Time boxing

| Task No. | Activity Name | Activity time | Total Time |
|---|---|---|---|
| 1 | Lab Manual Lecture | 60 mins | |
| 2. | Example | 10 mins | |
| 3. | Walkthrough Tasks. | 20 mins | |
| 4. | Lab Tasks | 60 mins | |
| 5. | Evaluation | 30 mins | 180 mins |

## Concept Map

- Syntax and Structure
- Model Directive
- Stack Segment Directive
- Data Segment Directive
- Code Segment Directive
- DOS Segment
- Procedure
- Syntax Rules

### Syntax and Structure

In start there are some directives that are used in assembly language. In the start of program, we have to define that directives. These are like preprocessors in C++. A directive is a code line (statement) that tells the compiler something. It is not code that ever gets executed; it is simply to help the compiler compile your code. Directives are instructions used by the assembler to help automate the assembly process and to improve program readability. The most common directive in C++. For example

*#include <iostream>*

## 1. Model Directive:

.model *key_word*

It defines memory for our program. In Assembly program we have to define space for our program. Any code we write it will save in reserved part of RAM. We need two types of space of RAM. For data as well as for code. We have to define space for data and code separate or combined. Following are keywords used for different types of space required.

| | |
|---|---|
| Tiny | Code and Data <= 64 KB |
| Small | Code <= 64KB, Data <= 64 KB |
| Medium | Code = any size, Data <= 64 KB |
| Compact | Code <= 64 KB, Data = any size |
| Large | Code = any size, Data = any size |

For example, if we want to reserve space less than or equal to 64KB combined for code and data then we write directive as

.model small

## 2. Stack Segment Directive:

In case of stack, we have to manage memory for stack separately. We can write it even if we are not using stack. Although operating system can manage itself stack memory but in practice we have to write this directive in our program.

.stack 100h

Here 100h means reserve 256 byte of data for stack.

## 3. Data Segment Directive:

It is written as .data. All the variables are defined here after .data directive. The .data section is used to declare the memory region where data elements are stored for the program. This section cannot be expanded after the data elements are declared, and it remains static throughout the program.

## 4. Code Segment Directive:

We write our code in this section. Our executable instructions are written in this part. It is written as follows.

.code

It is best practice that we write end statement here after **.code**. And between these code and end main end we write our all instructions. The syntax of end statement is as follows.

end main

This defines an area in memory that stores the instruction codes. This is also a fixed area.

## 5. DOS Segment:

Manges the arrangement of segment in a program. If we forgot the exact sequences of directives, we use dosseg. It manages the sequence itself according to Operating System requirement.

***dosseg***

**Procedure:**

We make function in code segment. At least one procedure is required in our case. We can name the procedure, for example in this case we make our procedure name as proc. It is very important to end main procedure with **main endp** statement

**main proc**

//instructions . . . . .

**main endp**

Inside procedure we write code.

**Syntax Rules:**

There should be space after opcode

One operand must be general purpose register

Operands must be of same size and type.

Comma between operands

So, there might be a question that why we have 14 types of register in CPU why not one? The answer is very simple, we cannot overload one register with different tasks, and therefore, we have different types of registers to perform different tasks.

**Sample Program:**

**; program that print 'is'**

```
.model small        ; reserve 64 KB space for data and code together.
.stack 100h         ; reserve space for stack
.data               ; data directive, all variables are defined after this instruction
.code               ; out instructions are written after this segment.
main proc           ; start of procedure, we have to define at least one procedure,
mov dl,'i'          ; store 'i' inside data register's part of dl
mov ah,2            ; display output of data that is stored in dl, in this case i is stored in dl
int 21h                     ; generate interrupt every time we access hardware, we are
                    accessing dl register
mov dl,'s'          ; store s inside data register's part of dl
mov ah,2            ; display output of data that is stored in dl, in this case s is stored in dl
int 21h             ; again, generate interrupt
mov ah,4ch          ; code to exit from data registers
int 21h             ; generate interrupt
main endp           ; end of procedure proc
end main            ; end of program
```