Experiment 3

Introduction:
In this experiment the students are exposed to the structure of an assembly language program and the definition of data variables and constants.

Objectives:
• Assembly language program structure
• Instructions and Directives
• Data representation
• Variable & constant declaration,
• ADD & SUB instructions

Assembly Language Program Structure

• An assembly language program is a sequence of instructions and directives.
• A program consists of one statement per line.
• The general structure of an assembly language program follows the guidelines shown in the following table:

| TITLE “Optional: Write here the Title of your program” |
| .MODEL SMALL |
| This directive defines the memory model used in the program. |
| .STACK |
| This directive specifies the memory space reserved for the stack |
| .DATA |
| Assembler directive that reserves a memory space for constants and variables |
| .CODE |
| Assembler directive that defines the program instructions |
| END |
| Assembler directive that finishes the assembler program |

Table 3.1: Assembly Language Program Structure

Instructions and Directives:

Instruction:
• The format of an assembly instruction closely mirrors the structure of a machine instruction
• An instruction is meant for the processor.
• The assembler translates this instruction into machine code

Statement syntax:

<table>
<thead>
<tr>
<th>Name</th>
<th>operation</th>
<th>operand(s)</th>
<th>;comment</th>
</tr>
</thead>
</table>
Examples:

```plaintext
MOV AX, BX  ; Load AX to prepare for multiplication
ADD AX, MEM16 ; AX = AX + MEM16
```

**Directive:**

*Pseudo-instructions* or *assembler directives* are instructions that are directed to the assembler. They will affect the machine code generated by and will not be translated directly into machine code. Directives are used to declare variables, constants, segments, macros, and procedures as well as supporting conditional assembly.

**Model Directive:**

The model determines the size of the code stack and data segments of the program. Each, of the segments is called a logical segment. Depending on the model used, the code and data segments may be in the same or in different physical segments as shown in Table 3.2.

In most of our programs, the model small is sufficient. The tiny model is usually used to generate *command* files (files with extension `.com`). This type of files is smaller in size than the executable files with extension `.exe`.

<table>
<thead>
<tr>
<th>Memory Model</th>
<th>Code</th>
<th>Data</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>TINY</td>
<td>≤ 64KB</td>
<td>≤ 64KB</td>
<td>Code + Data ≤ 64KB</td>
</tr>
<tr>
<td>SMALL</td>
<td>≤ 64KB</td>
<td>≤ 64KB</td>
<td></td>
</tr>
<tr>
<td>MEDIUM</td>
<td>may be ≥ 64KB</td>
<td>≤ 64KB</td>
<td></td>
</tr>
<tr>
<td>COMPACT</td>
<td>≤ 64KB</td>
<td>may be ≥ 64KB</td>
<td></td>
</tr>
<tr>
<td>LARGE</td>
<td>may be ≥ 64KB</td>
<td>may be ≥ 64KB</td>
<td>no array ≥ 64KB</td>
</tr>
<tr>
<td>HUGE</td>
<td>may be ≥ 64KB</td>
<td>may be ≥ 64KB</td>
<td>arrays can be ≥ 64KB</td>
</tr>
</tbody>
</table>

**Table 3.2:** Memory Models

**Stack Directive:**

- Directive is `.stack` for stack segment
- Should be declared even if program itself does not use stack needed for subroutine calling (return address) and possibly passing parameters
- May be needed to temporarily save registers or variable content
Memory Segment:
- Directive = .Data
- All variables must be declared at this level
- All constants must be defined at this level
- A variable is declared by: DB, DW ,…..
- A constant is defined using: the directive equ.

Code Segment:
- The directive .code is used for code segment
- The program code resides here

End of Program:
- The Directive End is used to tell the assembler that this is the end of the program source file.

Note:

The following sequence of instructions is always used at the beginning of a program to assign the data segment:

```
MOV AX, @DATA
MOV DS, AX
```

This sequence may be replaced by the following directive:

```
.STARTUP
```

which assigns both DATA and CODE segments, and hence the assembler will issue no warning. However, it should be noted that the program would start at address CS:0017H. The Startup directive occupies the bytes CS:0000 to CS:0017H.

```
.EXIT
```

ically, the sequence used to terminate and exit to DOS

```
MOV AH, 4CH
INT 21H
```

can be replaced by the .EXIT directive, which has exactly the same effect.
Data Representation:

Numbers:

- 11011 decimal
- 11011B binary
- 64223 decimal
- -21843D decimal
- 1,234 illegal, contains a non-digit character
- 1B4DH hexadecimal number
- 1B4D illegal hex number, does not end with “H”
- FFFFH illegal hex number, does not begin with a digit
- 0FFFFH hexadecimal number

- Signed numbers are represented using 2’s complement notation

Characters

- A character must be enclosed in single or double quotes: e.g. “Hello”, ‘Hello’, “A”, ‘B’
- The ASCII code is used to encode characters
- Examples:
  - ‘A’ has ASCII code 41H
  - ‘a’ has ASCII code 61H
  - ‘0’ has ASCII code 30H
  - Line feed has ASCII code 0AH
  - Carriage Return has ASCII code 0DH
  - Back Space has ASCII code 08H
  - Horizontal tab has ASCII code 09H

Note:

- The value of a variable, the content of registers or memory is based on the programmer interpretation:
  - AL = FFH
    - represents the unsigned number 255
    - represents the signed number -1 (in 2’s complement)
  - AH = 30H
    - represents the decimal number 48
    - represents the character ‘0’
  - BL = 80H
    - represents the unsigned number +128
    - represents the signed number -128
Variable Declaration

- Each variable has a type
- Based on its definition, a variable is assigned a memory location
- The location is defined by its address and number of bytes.
- Different data definition directives for different size types of memory
  
  - DB: define byte
  - DW: define word
  - DD: define double word (two consecutive words)
  - DQ: define quad word (four consecutive words)
  - DT: define ten bytes (five consecutive words)

- Each pseudo-op can be used to define one or more data items of given type.

Byte Variables

- The following directive defines a variable of size byte:

  - Var_name DB initial value
  - a question mark (?) place in initial value leaves variable non-initialized

Examples:

- I DB 4 define variable I with initial value 4
- J DB ? Define variable J with no initial value
- Name DB “Course” allocate 6 bytes for the variable Name
- K DB 5, 3, -1 allocates 3 bytes

Word Variables:

- The following directive defines a variable of size word:

  - Var_name DW initial value
  - a question mark (?) place in initial value leaves variable non-initialized
Examples:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>04 00</td>
</tr>
<tr>
<td>J</td>
<td>FE FF</td>
</tr>
<tr>
<td>K</td>
<td>BC 1A</td>
</tr>
<tr>
<td>L</td>
<td>31 30</td>
</tr>
</tbody>
</table>

Double Word Variables

- The following directive defines a variable of size double word:
  - Var_name DD initial value
**Constant Definition:**

- The EQU pseudo-op is used to assign a name to a constant
- Syntax:
  
  \[
  \text{Cst\_name} \text{ EQU } \text{Cst\_Value}
  \]

- No memory allocated for EQU names.
- Makes assembly language easier to understand

**Examples:**

Example 1:

\[
\text{MOV DL, 0AH}
\]

Can be replaced by:

\[
\text{LF \hspace{1cm} EQU \hspace{1cm} 0AH} \\
\text{MOV DL, LF}
\]

Example 2:

\[
\text{MSG DB “Type your name”}
\]

Can be replaced by:

\[
\text{PROMPT \hspace{1cm} EQU \hspace{1cm} “Type your name”} \\
\text{MSG DB PROMPT}
\]
# ASCII Table

<table>
<thead>
<tr>
<th>Hex</th>
<th>Binary</th>
<th>LSN</th>
<th>0000</th>
<th>0001</th>
<th>0010</th>
<th>0011</th>
<th>0100</th>
<th>0101</th>
<th>0110</th>
<th>0111</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>NUL</td>
<td>00</td>
<td>0000</td>
<td>0001</td>
<td>0010</td>
<td>0011</td>
<td>0100</td>
<td>0101</td>
<td>0110</td>
<td>0111</td>
</tr>
<tr>
<td>1</td>
<td>SOH</td>
<td>01</td>
<td>0000</td>
<td>0001</td>
<td>0010</td>
<td>0011</td>
<td>0100</td>
<td>0101</td>
<td>0110</td>
<td>0111</td>
</tr>
<tr>
<td>2</td>
<td>STX</td>
<td>02</td>
<td>0000</td>
<td>0001</td>
<td>0010</td>
<td>0011</td>
<td>0100</td>
<td>0101</td>
<td>0110</td>
<td>0111</td>
</tr>
<tr>
<td>3</td>
<td>ETX</td>
<td>03</td>
<td>0000</td>
<td>0001</td>
<td>0010</td>
<td>0011</td>
<td>0100</td>
<td>0101</td>
<td>0110</td>
<td>0111</td>
</tr>
<tr>
<td>4</td>
<td>EOT</td>
<td>04</td>
<td>0000</td>
<td>0001</td>
<td>0010</td>
<td>0011</td>
<td>0100</td>
<td>0101</td>
<td>0110</td>
<td>0111</td>
</tr>
<tr>
<td>5</td>
<td>ENQ</td>
<td>05</td>
<td>0000</td>
<td>0001</td>
<td>0010</td>
<td>0011</td>
<td>0100</td>
<td>0101</td>
<td>0110</td>
<td>0111</td>
</tr>
<tr>
<td>6</td>
<td>ACK</td>
<td>06</td>
<td>0000</td>
<td>0001</td>
<td>0010</td>
<td>0011</td>
<td>0100</td>
<td>0101</td>
<td>0110</td>
<td>0111</td>
</tr>
<tr>
<td>7</td>
<td>BEL</td>
<td>07</td>
<td>0000</td>
<td>0001</td>
<td>0010</td>
<td>0011</td>
<td>0100</td>
<td>0101</td>
<td>0110</td>
<td>0111</td>
</tr>
<tr>
<td>8</td>
<td>BS</td>
<td>08</td>
<td>0000</td>
<td>0001</td>
<td>0010</td>
<td>0011</td>
<td>0100</td>
<td>0101</td>
<td>0110</td>
<td>0111</td>
</tr>
<tr>
<td>9</td>
<td>HT</td>
<td>09</td>
<td>0000</td>
<td>0001</td>
<td>0010</td>
<td>0011</td>
<td>0100</td>
<td>0101</td>
<td>0110</td>
<td>0111</td>
</tr>
<tr>
<td>A</td>
<td>LF</td>
<td>0A</td>
<td>0000</td>
<td>0001</td>
<td>0010</td>
<td>0011</td>
<td>0100</td>
<td>0101</td>
<td>0110</td>
<td>0111</td>
</tr>
<tr>
<td>B</td>
<td>VT</td>
<td>0B</td>
<td>0000</td>
<td>0001</td>
<td>0010</td>
<td>0011</td>
<td>0100</td>
<td>0101</td>
<td>0110</td>
<td>0111</td>
</tr>
<tr>
<td>C</td>
<td>FF</td>
<td>0C</td>
<td>0000</td>
<td>0001</td>
<td>0010</td>
<td>0011</td>
<td>0100</td>
<td>0101</td>
<td>0110</td>
<td>0111</td>
</tr>
<tr>
<td>D</td>
<td>CR</td>
<td>0D</td>
<td>0000</td>
<td>0001</td>
<td>0010</td>
<td>0011</td>
<td>0100</td>
<td>0101</td>
<td>0110</td>
<td>0111</td>
</tr>
<tr>
<td>E</td>
<td>SO</td>
<td>0E</td>
<td>0000</td>
<td>0001</td>
<td>0010</td>
<td>0011</td>
<td>0100</td>
<td>0101</td>
<td>0110</td>
<td>0111</td>
</tr>
<tr>
<td>F</td>
<td>SI</td>
<td>0F</td>
<td>0000</td>
<td>0001</td>
<td>0010</td>
<td>0011</td>
<td>0100</td>
<td>0101</td>
<td>0110</td>
<td>0111</td>
</tr>
</tbody>
</table>

Example on the use of the ASCII table:

<table>
<thead>
<tr>
<th>Character</th>
<th>Column #</th>
<th>Row #</th>
<th>Code (H)</th>
<th>Code (binary)</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>6</td>
<td>1</td>
<td>61H</td>
<td>140</td>
</tr>
<tr>
<td>A</td>
<td>4</td>
<td>1</td>
<td>41H</td>
<td>10001</td>
</tr>
<tr>
<td>β</td>
<td>E</td>
<td>1</td>
<td>E1H</td>
<td>11001</td>
</tr>
<tr>
<td>%</td>
<td>2</td>
<td>5</td>
<td>25H</td>
<td>11011</td>
</tr>
</tbody>
</table>

Table 3.3: Using the ASCII table:
### ADD & SUB instructions:

<table>
<thead>
<tr>
<th>Type</th>
<th>Inst.</th>
<th>Example</th>
<th>Meaning</th>
<th>Flags Affected</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>O  S  Z  A  P  C</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>F  F  F  F  F  F</td>
</tr>
<tr>
<td><strong>Addition</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADD</td>
<td>ADD AX, 7BH</td>
<td>AX ← AX + 7B</td>
<td>*  *  *  *  *  *</td>
<td></td>
</tr>
<tr>
<td>ADC</td>
<td>ADC AX, 7BH</td>
<td>AX ← AX + 7B +CF</td>
<td>*  *  *  *  *  *</td>
<td></td>
</tr>
<tr>
<td>INC</td>
<td>INC [BX]</td>
<td>[BX]←[BX]+1</td>
<td>*  *  *  *  *  -</td>
<td></td>
</tr>
<tr>
<td>DAA</td>
<td>DAA</td>
<td></td>
<td>?  *  *  *  *  *</td>
<td></td>
</tr>
<tr>
<td><strong>Subtraction</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SUB</td>
<td>SUB CL,AH</td>
<td>CL ← CL – AH</td>
<td>*  *  *  *  *  *</td>
<td></td>
</tr>
<tr>
<td>SBB</td>
<td>SBB CL,AH</td>
<td>CL ← CL – AH – CF</td>
<td>*  *  *  *  *  *</td>
<td></td>
</tr>
<tr>
<td>DEC</td>
<td>DEC DAT</td>
<td>[DAT] ← [DAT] – 1</td>
<td>*  *  *  *  *  -</td>
<td></td>
</tr>
<tr>
<td>DAS</td>
<td>DAS</td>
<td></td>
<td>?  *  *  *  *  *</td>
<td></td>
</tr>
<tr>
<td>NEG</td>
<td>NEG CX</td>
<td>CX ← 0 – CX</td>
<td>*  *  *  *  *  *</td>
<td></td>
</tr>
</tbody>
</table>

*Table 3.4*: Summary of add and sub instructions
Exercises

Program 1: A Case Conversion Program

Write a program that prompts the user to enter a lowercase letter, and on next line displays another message with letter in uppercase.

- Enter a lowercase letter: a
- In upper case it is: A

Title “Program Small to Upper Case Conversion”

.Model Small
.Stack 100
.DATA
   CR EQU 0DH
   LF EQU 0AH
   MSG1 DB ‘Enter a lower case letter: $’
   MSG2 DB CR, LF, ‘In upper case it is: ‘
   Char  DB ?, ‘$’
.CODE
.STARTUP ; initialize data segment
   LEA DX, MSG1 ; display first message
   MOV AH, 9
   INT 21H
   MOV AH, 1 ; read character
   INT 21H
   SUB AL, 20H ; convert it to upper case
   MOV CHAR, AL ; and store it
   LEA DX, MSG2 ; display second message and
   MOV AH, 9 ; uppercase letter
   INT 21H
.EXIT ; return to DOS
END

Program 2: A Case Conversion Program 2

Write a program that prompts the user to enter an uppercase letter, and on the next line displays another message with letter in lowercase.

Program 3:

Write a program that reads small characters from the keyboard and converts them online to uppercase ones. Use the following to make your program loop. Also use function 08 to read a character without echo.

        next: ................. ; read character
             ................. ; convert
             ................. ; display
        Loop next