Department of
Computer Science and Engineering

Title: Introduction to assembly program structure and various arithmetic operations.

Microprocessors and microcontrollers

CSE 304

Green University of Bangladesh

1 Objective(s)

- To provide an introduction to syntax and structure of assembly language.
- To implement various basic arithmetical operations.

2 Problem analysis

We have already learned about 'instruction set' in the previous class. Today we will learn how we can integrate the instructions in a structured way to solve a specific problem. Each machine language programs consist of code, data and stack. Each part occupies a memory segment. The same organization is reflected in an assembly language program. In assembly program, the code, data and stack are structures as program segments. Each segment is translated into a memory segment by the assembler. In 8086 microprocessor Bus Interface Unit (BIU) there are four different segments present:

- Data Segment (DS)
- Stack Segment (SS)
- Code Segment (CS) and
- Extra Segment (ES)

2.1 Basic Structure of Assembly Programs

2.1.1 Memory Model

The size of the code and data a program can have is determined by specifying a memory model using .MODEL directive.

Syntax: **.MODEL memory_model**

The most frequently used memory models are SMALL, MEDIUM, COMPACT and LARGE. Unless there is a lot of code or data, the appropriate model is SMALL. The MODEL directive should come before any segment definition.

Table 1: General Purpose Registers

Model	**Description**
Small	Code in one segment; Data in one segment.
Medium	Code in more than one segment; data in one segment.
Compact	Code in one segment; data in more than one segment.
Large	Code in more than one segment; data in more than one segment. No array larger than 64K bytes.
Huge	Code in more than one segment; data in more than one segment. Arrays may be larger than 64K bytes.

[Note: Unless there is a lot of code or data, the appropriate model is SMALL. The MODEL directive should come before any segment definition.]

2.1.2 Stack Segment

The purpose of the stack segment declaration is to set aside a block of memory to store the stack. The stack area should be big enough to contain the stack at its maximum size.

Syntax: **.STACK size**

where size is an optional number that specifies the stack area size in bytes. Example: .STACK 100h. This sets aside 100h bytes for the stack area. If size is omitted, 1 KB is set aside for the stack area.

Table 2: Data Segment Example

.DATA		
WORD1	**DB**	**2**
WORD2	**DW**	**5**
MSG	**DB**	**'THIS IS A MESSAGE$'**
MASK	**EQU**	**10010010B**

2.1.3 Data Segment

A programs data segment contains all the variable definitions. Constant definitions are often made here as well. However, they may be placed elsewhere in the program since no memory allocation is involved.

To declare a data segment, we use the directive **.DATA**, followed by variables and constant declarations in 2.

[Note: Our compiler supports two types of variables: BYTE (DB) and WORD (DW). We'll learn about variable on next day.]

2.1.4 Code Segment

The code segment contains a programs instructions.

Syntax: **.CODE name**

where name is the optional name of the segment. Inside a code segment, instructions are organized as procedures.

Example:

.CODE
MAIN PROC
;main procedure instruction
MAIN ENDP
;other procedure go here

[Note: In assembly, comments are usually denoted by a semicolon (;).]

Finally, all putting together the basic structure of assembly language as follows,

```
MODEL SMALL
;This is a comment
.STACK 1OOH
.DATA
;data definitions go here
.CODE:
MAIN PROC
;instructions go here
MAIN ENDP
;other proced
END MAIN
```

2.2 Input-Output Instructions using Interrupt

2.2.1 INT Instruction

To invoke a DOS or BIOS routine, the INT(interrupt) instruction is used.

Syntax: **INT interrupt_number**

where interrupt_number is a number that specifies a routine. For example, **INT 16h** invokes a BIOS routine that performs keyboard input.

2.2.2 INT 21h Operations

INT 21h is used to invoke a large number of DOS functions such as input and output. The following Table 3 shows the different functionalities of the **INT 21h** interrupt.

Table 3: Functionalities of INT 21h Interrupt

Function Number	Task	Functionality
1	Single-key input	Input:AH = 1 Output: AL = ASCII code if character key is pressed AL = 0, if non-character key is pressed
2	Single-character output	Input: AH= 2 DL = ASCII code of the display character or control character Output:AL = ASCII code of the display character or control character
3	Character string output	Input: DX = offset address of string The string must end with a $ character

In the following example 2.2.3, a single key input is taken from user and displaying the single character as output.

2.2.3 Example

```
 1  .MODEL SMALL
 2   .STACK 100H
 3
 4   .CODE
 5     MAIN PROC
 6       MOV AH, 1                    ; read a character
 7       INT 21H
 8
 9       MOV BL, AL                   ; save input character into BL
10
11       MOV AH, 2
12       MOV DL, 0DH
13       INT 21H
14       MOV DL, 0AH                  ;line 11 to line 14 is used for printing new
            line.
15
16       MOV AH, 2                    ; display the character stored in BL
17       MOV DL, BL
18       INT 21H
19
20       MOV AH, 4CH                  ; return control to DOS
21       INT 21H
22
23     MAIN ENDP
24   END MAIN
```

2.2.4 Output

```
A
A
```

2.2.5 LEA (Load Effective Address)

INT 21h, function 09, expects the offset address of the character string to be in DX. To get it there we use a new instruction: **LEA destination, source** where destination is general register and source is memory location. It puts a copy of the source offset address into the destination. For Example:

LEA DX, MSG

Puts the offset address of the variable MSG into DX.

2.2.6 INT 21h functions

Character String Output: with DS initialized we may print the ‘HELLO!" message by placing its address in DX and executing INT 21h:

```
LEA   DX, MSG          ;get message
MOV   AH, 9            ;display string function
INT   21h              ;display string
```

2.2.7 Program Segment Prefix

When a program is loaded in memory , DOS prefaces it with 256 byte Program Segment Prefix (PSP). The PSP contains information information about the program. So that programs may access this area. DOS places its segment number in both DS and ES before executing the program. The result is that DS does not contain the segment number of the data segment. To correct this, a program containing a data segment begins with these two instructions:

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

@DATA is the name of the data segment defined by **.DATA**. The Assembler translates the name @DATA into a segment number. Two instruction are needed because a number (data segment number) may not be moved directly into a segment register.

3 Some Examples To Practice

3.1 Multiplication of two numbers(2 and 5)

```

.MODEL SMALL
.STACK 100H
.DATA
A DW 2
B DW 5
RESULT DW ?
.CODE
MAIN PROC
    MOV AX,@DATA
    MOV DS,AX
    ;MUL TWO NUMBERS

    MOV AX,A
    MUL B
    MOV RESULT,AX


    MOV AX,02H
    INT 21H


    MAIN ENDP
    END MAIN
```

3.1.1 Output

10

3.2 Printing Message

```

.MODEL SMALL
.STACK 100H
.DATA
MSG  DB  'HELLO!'

.CODE
MAIN PROC
;initialize  DS
    MOV AX, @DATA
    MOV DS, AX           ;initialize  DS

;display message
    LEA DX, MSG          ;get message
    MOV AH, 9            ;display string function
    INT 21h              ;display string

;return to DOS
    MOV AH, 4CH
    INT 21h             ;DOS exit

MAIN ENDP
     END MAIN
```

3.2.1 Output

HELLO!

4 Discussion & Conclusion

Based on the focused objective(s) to understand about the structure of assembly language,various arithmetic operations and the additional lab exercise made me more confident towards the fulfilment of the objectives(s).

5 Lab Task (Please implement yourself and show the output to the instructor)

1. Perform addition, subtraction, multiplication and division of 6 and 2 these two numbers in a single code.
2. Take two single digit integer inputs from and perform addition, subtraction, multiplication and division on two numbers.

6 Lab Exercise (Submit as a report)

- Take a double digit number input from the user.
- Convert 260°C to Fahrenheit using the following expression and store in a F variable: °F = °C×9/5 + 32 - 1
- Convert 1000 °F (Fahrenheit) to °C (Celsius) using the following expression and store in a C variable: °C = (°F - 32)× 5/9 + 1

7 Policy

Copying from internet, classmate, seniors, or from any other source is strongly prohibited. 100% marks will be *deducted* if any such copying is detected.