Princess Sumaya University for Technology
Computer Engineering Department
22440: Microprocessor Lab

Experiment 1: Introduction to Microsoft Assembler (MASM)

Introduction

The Microsoft Assembler (MASM) translates an assembly language source file into a machine language object file. The assembler program requires that a symbolic program be first written, using a text editor provided with the assembler package. The editor provided by this version is **edit.exe**. When you use any word processor, the source file that you generate must use the extension "**.asm**" that is required for the assembler to properly identify your source program.

Once your source file is prepared, it must be assembled. This is accomplished by using a DOS command. Once a program is assembled, it must be linked before it can be executed. The linker converts the object file into an executable file "**.exe**". You can use the DOS command (**ml name.asm**) to assemble and link your program.

Code view is also available with MASM. (**cv name.exe**) must be typed at the DOS command line to access it.

Program Structure

The machine language program consists of code, data, and stack. Each part occupies a memory segment. Each program segment is translated into a memory segment by the assembler.

Memory Models

The size of code and data a program can have is determined by specifying a memory model using `.MODEL` directive, which has the following syntax:

```
.MODEL    memory-model
```

The most frequently used memory models are: **small**, **medium**, and **compact**. The `.MODEL` directive should come before any segment definition. The following table describes these models.

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

Data Segment

A program's data segment contains all the variable definitions. To declare a data segment, we use the directive `.DATA` followed by variable and constant declarations. For example:

```
.DATA
```

```
Word1     db   2
Msg       db   'This is a message$'
```

Stack Segment

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

```
     .STACK    size
```

Where size is an optional number that specifies the stack area size in bytes, for example:

```
     .STACK    size
```

If `size` is omitted, 1KB is set aside for the stack area.

Code segment

The code segment contains a program's instructions. The declaration syntax is:

```
     .CODE     name
```

Where `name` is the optional name for the segment (there is no need for the name in a **small** program, because the assembler will generate an error). Inside a code segment, instructions are organized as procedures. The simplest procedure definition is:

```
name PROC
     ; body of the procedure
name ENDP
```

Where `name` is the name of the procedure.

Putting It All Together

Now you have seen all the program segments, we can construct the general form of a **small** program. With minor variables, this form may be used in most applications.

```
.MODEL small
.STACK 100H
.DATA
     ;Data definitions go here.
.CODE
MAIN PROC
     ;Initializing data and extra sections
     MOV AX, @DATA
     MOV DS, AX
     MOV ES, AX
     ;Instructions go here.
     MOV AH, 4CH
     INT 21H                  ;Exit to DOS
     MAIN ENDP
;Other procedures go here
END MAIN
```

Exercise

1) Enter the following program using MS-DOS editor (**edit exp1.asm**).

```
.MODEL small
.STACK 100H
.DATA
     ;Data definitions go here.
.CODE
MAIN PROC
     MOV AX, @DATA
     MOV DS, AX
     MOV ES, AX

     ;Instructions go here.
     MOV CX, 5
     MOV BX, 50H
     MOV AL, BL
     MOV DX, CX

     ;Exit to DOS
     MOV AH, 4CH
     INT 21H
     MAIN ENDP
END MAIN
```

2) Save your program as **exp1.asm**
3) Use the DOS command (**ml exp1.asm**) to assemble and link your program. What is the result of this operation?

____

4) Type the command (**cv exp1**) to access the code view.
5) Fill the **AL** register with 45h. Declare in the data segment `label1 db ?`. Move the contents of the **AL** register to the memory location `label1`. Check that the memory location does contain 45h. How do you check the contents of memory locations and the registers?

____
____
____
____

6) View the registers and memory view ports and execute your program step by step using F8.

7) Add the following instructions to your program and write down the register value after applying each instruction, write your own comments for each instruction below.

```
    Mov al, 100     ; AL =
    Mov ah, 32H     ; AH =
    Mov bl, 'a'     ; BL =
    Mov bh, '1'     ; BH =
```

8) What is the difference between the size of "DE" in ASCII code and 0DEH in Hex?

____

9) Write below the appropriate instruction to fill the hexadecimal word 0ABCDH in to Register **AX**. Test the instruction and check that the register does really contains the number?

____

10) Try the wrong instruction `Mov ax, bl`. Link the program. What does the output state for the line in which there is this error? Write below the output.

____

Remember

```
Label1 db  10 dup(?) ;Reserves 10 locations starting at Label1
Label1 db  10 dup(5) ;Reserves 10 locations starting from Label1, all have 5
       Mov cx, [Bx]  ;Moves the memory word specified by Bx + 10*Ds to cx.
       Mov [Bp], dl  ;Moves dl contents to memory address specified by Bp + 10*ss
       Mov [Di], Bh  ;Moves Bh contents to memory address specified by Di + 10*Ds
```