Experiment N^o 3

Segmentation and Addressing Modes

Introduction:

In this experiment you will be introduced to physical segmentation of the memory, and the logical segmentation of programs. You will also deal with the different addressing modes, and learn how to calculate the physical and offset addresses.

Objectives:

1- Addressing modes in the 8086 processor
2- Segmentation: Physical Segments and Logical Segments.

References:

Textbook:

- Addressing modes: section 4.3,
- Segmentation: section 3.1,
- Lecture notes.

Addressing Modes:

The following table summarizes all addressing modes used by the 8086 processor.

Addressing Mode	Example	Source operand		
		Assuming: DS = 1000H, BX = 0200H, SI = 0300H		
		Type	Address Generation	Address
Register	MOV AX,BX	Register	-	-
Immediate	MOV AX, 0F7H	Immed.	-	-
Direct	MOV AX,[1234H]	Mem.	DS x 10H +1234H	11234H
Register-Indirect	MOV AX,[BX]	Mem.	DS x 10H +0200H	10200H
Based	MOVAX,[BX+06]	Mem.	DS x 10H +0200H + 0006H	10206H
Indexed	MOVAX,[SI+06]	Mem.	DS x 10H +0300H + 0006H	10306H
Based-Indexed	MOV AX,[BX+SI+06]	Mem.	DS x 10H +0200H +0300H + 0006H	10506H

Table 3.1: Addressing modes

Structure of an Assembly Language Program:

An assembly language program is written according the following structure and includes the following assembler directives:

TITLE "Optional: Write the Title of your program"
.MODEL SMALL
Assembler directive that defines the memory model to use in the program. The memory model determines the size of the code, stack and data segments of the program

.STACK
Assembler directive that reserves a memory space for program instructions in 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

Each of the segments is called a logical segment. Depending on the memory, the code and data segments may be in the same or in different physical segments according to table 3.3.

Memory Model	**Size of Code and Data**
TINY	Code and data no more than 64KB combined
SMALL	Code and data segments must be no more than 64KB each
MEDIUM	Code can be more than 64KB, data still limited to no more than 64KB
COMPACT	Code limited to no more than 64KB, data can be more than 64KB
LARGE	Code and data can each be more than 64K, no array can be larger than 64KB
HUGE	Code and data can each be more than 64KB, arrays can be larger than 64KB

Table 3.3: Memory Models

Stack Directive:

- Directive is `.stack` for stack segment
- Should be declared even if program itself doesn't use stack needed for subroutine calling (return address) and possibly passing parameters
- May be needed to temporarily save registers or variable content
- Will be needed for interrupt handling while program is running

Memory allocation:

- Directive is `.data` for data segment
- All variables must be declared, and memory space for each allocated.
- Data definition directive can be followed by a single value, or a list of values separated by commas
- Different data definition directives for different size types of memory
    1. DB - define byte (8 bits)
    2. DW - define word (16 bits)
    3. DD - define double word (32 bits)
    4. DQ - define quad word (64 bits)

Code Segment:

- Directive is `.code` for code segment
- The "program" resides here

End of Program:

- Directive is `End`
- Tells assembler that this is the end of the program

Note:

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

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

May be replaced by the following directive:

```
.STARTUP
```

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

Identically, the sequence used to terminate and exit to DOS can be replaced by the .EXIT directive, which has exactly the same effect.

Pre Lab Work:

1. Study the attached hand out, and review the material related to segmentation and addressing modes.
2. Write programs 3-1 and 3-2
3. Write the program given in assignment.
4. Fill in the tables associated with the different programs.
5. Bring your work to the lab.

Lab Work:

1- Assemble, Link and Run program 1.
2- Use CodeView Debugger to fill in the table associated with program 3.1.
3- Calculate both the effective and physical addresses of each instruction. Put the results on the given table.
4- Assemble, Link and Run program 2.
5- Fill in table 2, associated with program 2, in which you specify only the addressing mode, for both source and destination, for each instruction.

6- Show all tables to the instructor.
7- Submit all your work at the end of the lab session.

Lab Assignment:

Write a program that prompts the user to enter a string, in capital letters, of a maximum length of 20 characters. Read the string in capital letters and convert it to small letters. Then display the new string.

Note:
To convert a capital letter to a small one, use the following instruction:

```
;Read character
MOV AL, character_read
ADD AL, 20H
; Display character in AL register
```

Use the following to loop through the string you just entered.

```
        MOV CX, Number_of_bytes_read
Again:  Start loop here
        ; Convert to small letters.

        LOOP Again
```

; This program displays a string terminated by a $ sign using INT 21H function 09H.

```
TITLE “Program 3-1”
.MODEL SMALL

.STACK 200
.DATA
        MESSAGE    DB      'This is the message to be displayed: ', '$'
        MESSAGE2   DB      0dh, 0ah, 'The message you just entered : ' , '$'
        BUF        DB 10            ; Number of characters to be read
                   DB 11 DUP('$')         ; Reserve 10 bytes for string

.CODE
        MOV AX,@DATA
        MOV DS,AX

        LEA DX,MESSAGE
        MOV AH,09H
        INT 21H
        MOV AH, 0AH
        MOV DX, OFFSET BUF
        INT 21H

        LEA DX,MESSAGE2
        MOV AH,09H
        INT 21H

        LEA DX, BUF
        Add DX,02
        MOV AH,09H
        INT 21H

        MOV AX,4C00H
        INT 21H
END
```

```
TITLE “PROGRAM 2 EXPERIMENT 3”
; This program displays a  message and reads a new message from the keyboard

.MODEL SMALL
.STACK 200
.DATA
        CRLF        DB      0DH,0AH,'$'
        PROMPT      DB      'Enter a name of max. length 30 char.: ',0DH,0AH,'$'
        STRING1     DB      'Mr. ','$'
        STRING2     DB      ' studies 8086 programming. ','$'

        ; Allocate 32 bytes for BUFFER, and put the value 31 in the second byte.
        BUFFER      DB      31,32 DUP(?)

.CODE
.STARTUP                            ;This directive initializes the DS and CS segments.
        LEA DX,PROMPT               ;display prompt
        MOV AH,09H
        INT 21H

        MOV AH,0AH                  ;read into buffer
        LEA DX, BUFFER
        INT 21H

        LEA DX, CRLF                ;move cursor to next line
        MOV AH,09H
        INT 21H

        LEA DX,STRING1              ;display string1
        MOV AH,09H
        INT 21H

        ;now display the buffer i.e. what has been read.

        MOV AH,09H
        MOV BH,00H
        MOV BL,BUFFER[1]            ;move in BL buffer length
        MOV BUFFER[BX+2],'$'        ;put a $ sign at the end of buf
        LEA DX,BUFFER[2]            ;load actual length of buffer
        INT 21H

        LEA DX,STRING2              ;display string2
        MOV AH,09H
        INT 21H

        LEA DX, CRLF                ;move cursor to next line
        MOV AH,09H
        INT 21H

        MOV AH, 02H                 ; display number of characters read if less than 10
        MOV DL,BUFFER[1]            ; read second byte of buffer
        ADD DL, 30H                 ; convert to number
        INT 21H
        MOV AX,4C00H
        INT 21H
END
```