Lab Outline

In this lab you cover following

1) Declare data in Assembly language using labels
2) Little endian and Big endian
3) Data in memory
4) Moving data to and from memory using labels+ offsets

Declaring data:

There are three ways to declare data in Assembly Language details are given in following table

Method	Syntax	Examples
1 Declare single value with label	<label_name>: <db/dw/dd> <value>	[org 0x0100] ; code goes here mov ax, 0x4c00 ; termination statements int 21h ; data declaration num1: dw 5 num2: db 10 num3: dd 15 num4: dw 9
2 Declare Multiple values with single labels, values can be of different size	<label_name>: <db/dw/dd> <value 1> <db/dw/dd> <value 2> <db/dw/dd> <value 3> . . . <db/dw/dd> <value n>	[org 0x0100] ; code goes here mov ax, 0x4c00 ; termination statements int 21h ; data declaration num1: dw 5 dw 10 db 15 dd 9
3 Declare Mutiple values with single label, all values will be of same size	<label_name>: <db/dw/dd> <value1>, <value2>, …. <value n>	[org 0x0100] ; code goes here mov ax, 0x4c00 ; termination statements int 21h ; data declaration num1: dw 5, 10,15, 90

- db stands for define byte,
- dw is stands for define word (word is of 2 bytes)
- dd stands for define double (double is of 2 words that is 4 bytes)
- Labels are like pointer. They only have the address of the lowest byte of value. In case of multiple values declared with single label (method 3 and 4) label is the address of lowest byte of first value.
- For now you will declare data after the termination statements of code.

Little Endian vs Big Endian:

Big endian (BE), number is written from higher BYTE to lower BYTE

Little Endian (LE), number is written from Lower BYTE to higher BYTE

Example:

Number (in hex)	Big Endian format	Little Endian Format
0A0B	0A0B	0B0A
01020356	01020356	56030201

1A	1A	1A
0A0B0C0D0E0F1268	0A0B0C0D0E0F1268	68120F0E0D0C0B0A

- Remember 2 digits of hex is one byte.

It important to know where data is written in LE form and BE form.

- When you write code you write numbers in BE form
- When you look at registers in AFD you see numbers in BE form
- When you see machine code in AFD or listing file you see number on LE form
- When you look at memory in AFD you will see data in LE form

Why we need to know these two formats?
Answer: because for the processor we are studying data is stored in memory in Little endian format. Whereas in code you write data in Big Endian Format.
Why is it like that?
Well, because Intel's processors are designed like this. There are other processors that store data in Big Endian form and some even support bi-endian (can switch between big-endian and little endian)
Read more on https://en.wikipedia.org/wiki/Endianness#History

Data in Memory:

It's important to know how data is stored in memory and at which byte of data labels points at

Labels are like pointer. They only have the address of lowest byte of value (because data is saved in LE form). In case of multiple values declared with single label (method 3 and 4) label is address of lowest byte of first value. To access byte pointed by label we use [label_name] to access other bytes of data associated with label we have to add offset [label_name+offset].

Task 1a: Write, assemble (with listing file) and open the code in AFD

```
; lab2Task1code

[org 0x0100]

       ; no code because we just want to look

       ; at how data is stored in memory

mov ax, 0x4c00 ; termination statements int
21h
```

LISTING FILE

```
 1                                      [org 0x0100]
 2
 3                                      ; no code because we just want to look
 4                                      ; at how data is stored in moemory
 5
 6 00000000 B8004C                      mov ax, 0x4c00 ; termination statements
 7 00000003 CD21                        int 21h
 8                                      ; data declaration
 9 00000005 0D0C0B0A                    Num1: dd 0A0B0C0Dh
10 00000009 0201                        Num2: dw 0102h
11 0000000B 33                          Num3  db 33h
12 0000000C 5634                        Nums: dw 3456h
13 0000000E 99                                db  99h
14 0000000F 0F0E060508071009            my_array: dw 0E0Fh, 0506h, 0708h, 0910h
15
```

offset address on num1

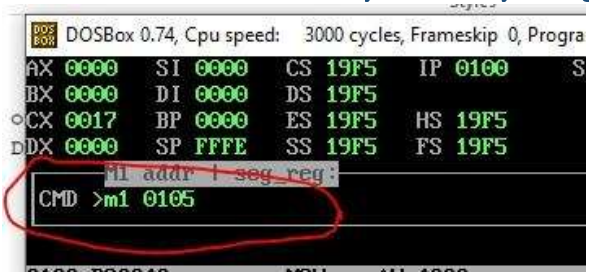
in machine code all the numbers are in LE form

In code, data is writen in big endian form

offset address of my_array

Figure 1 listing file of lab2Task1 code

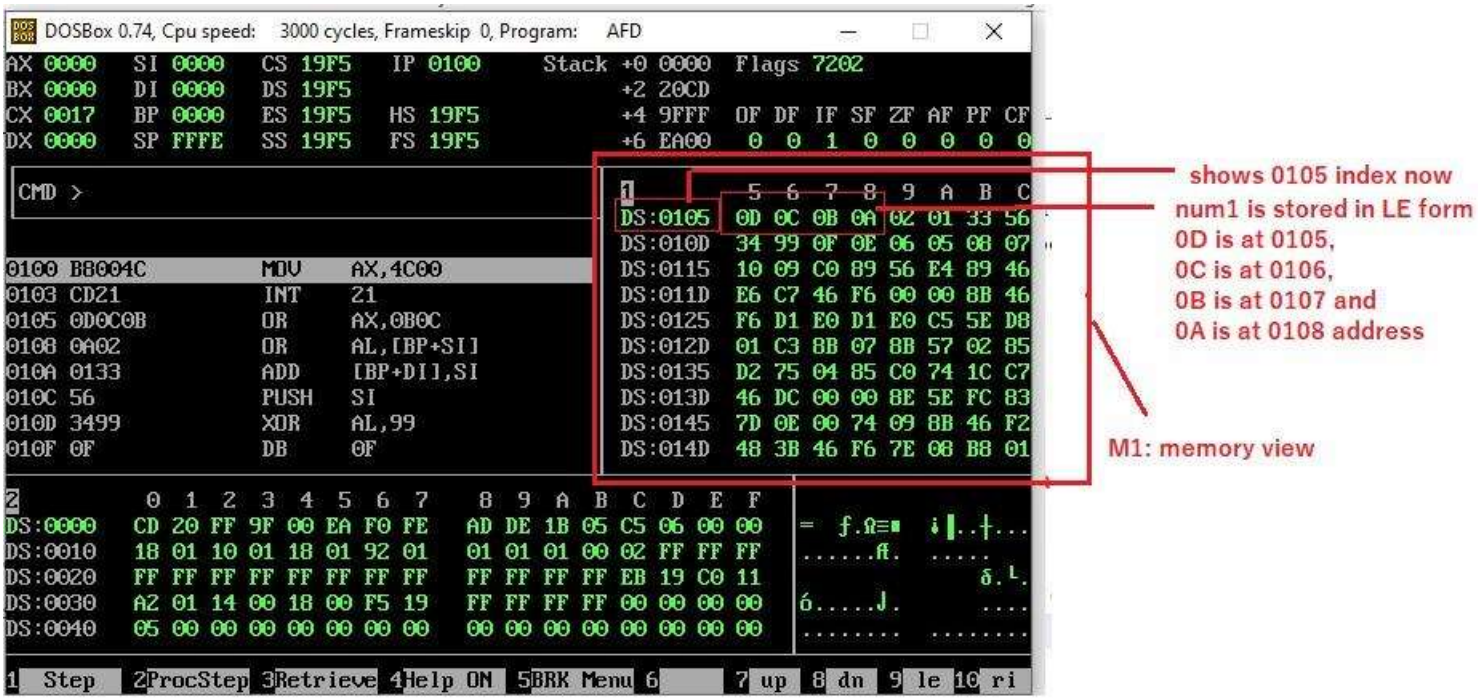
Task1b: See the data in memory view M1 by writing the following statement



Where 0105 is address at which num1 is pointing.

You can see the address of num1 from listing file, see fig1, just add 0100h to it (base address from where code starts to load)

See how data is stored in little endian form in memory and write the address that points to each byte



Task 1C: see how num2, num3, nums and my_array is stored in memory, fill the following table (first row is filled as an example)

Note: you can see the address of labels from listing file, Just add 0100h to the address that you see in listing file

label	Data in code	Data in memory	Address of each byte [label+offset]
Num1	0A0B0C0D	0D0C0B0A	[num1] for 0D [num1+1] for 0C [num1+2] for 0B [num1+3] for 0A
Num2			
Num3			
Nums			
my_array			

Moving data to and from memory:

You can only move one byte or word (two bytes) to and from memory at one time

- To access any byte you give address of that byte to as source or destination
- To access any word (2 byte) you give address of lowest byte of that word as source/destination
- Addresses are ***label_name+/-offset*** Where offset is some number >=0

Memory to register:

Mov register, [label_name+/-offset]

Register to Memory:

Mov [label_name+/-offset] register

Immediate operand to Memory

You have to specify the size of value you are moving

Mov byte [label_name+/-offset], value
Mov word [label_name+/-offset], value

Memory to Memory:
Not possible, first move from source memory to registers and then from register to destination memory

Task2: Run the following code and see the changes in registers write the values of ax, al and ah after each line.

```
;lab2task2code
[org 0x0100]
;code
      Mov ax, [num1]   ;ax=?
      Mov ax, [num2]   ;ax=?
      Mov ax, [num2+2] ;ax=?
      Mov ax, [num2+1] ;ax=?
      Mov al, [num2+3] ;ax=?
      Mov ah, [num1] ;ax=? Mov
      ax, [array1] ;ax=?
      Mov ax, [array1+2]      ;ax=?
      Mov al, [array2] ;ax=?
      Mov al, [array2+1]      ;ax=?
      Mov ax, [array2] ;ax=?
mov ax, 0x4c00 ; termination statements
int 21h ; data
      Num1: dw 0A0Bh
      Num2: dd 0C0D0E0Dh
      Array1: dw 0102h , 0304h
      Array2: db 05h , 06h, 07h
```

Task 3: Run the following code and see the changes in memory (in labels you declared)

```
;lab2task2code
[org 0x0100]
;code

      Mov ax, 9876h
      Mov bx, 5432h
      Mov  [num1], ax
      Mov [num2], bx
      Mov [num2+2], bx
      Mov [array1], ax
      Mov [array2], bl
      Mov [array2], ax
      Mov word [num1], 0000h
      Mov byte [num1], 01h
      Mov  byte [num2+1], 11h
      Mov word [array1+2], 3870h

mov ax, 0x4c00 ; termination statements
int 21h ; data

      Num1: dw 0A0Bh
      Num2: dd 0C0D0E0Dh
      Array1: dw 0102h , 0304h
      Array2: db 05h , 06h, 07h
```

(All the data given below is in is in BE form)

Exercise Prelab

Try to solve these questions before coming to lab

1. Write a code the add 10 in num1 and add 20 in num2. Num1 and num2 should be defined as follow

   Num1: dw 0102h
   Num2: db 09h

2. Write a code the swaps the values of lower byte and higher byte in num1. Num1 should be defined as follow

   Num1: dw AABBh

   At the end of code Num1 should be BBAA