Lab # 6

# OBJECTive

Testing and Interfacing Memory Device with 8086 Microprocessor

# Theory

**DEBUG PROGRAM:**

Debug can act as an [assembler](https://en.wikipedia.org/wiki/Assembly_language#Assembler), [disassemble](https://en.wikipedia.org/wiki/Disassembler), or [hex dump](https://en.wikipedia.org/wiki/Hex_dump) program allowing users to interactively examine [memory](https://en.wikipedia.org/wiki/Computer_storage) contents (in [assembly language](https://en.wikipedia.org/wiki/Assembly_language), [hexadecimal](https://en.wikipedia.org/wiki/Hexadecimal) or [ASCII](https://en.wikipedia.org/wiki/ASCII)), make changes, and selectively execute [COM](https://en.wikipedia.org/wiki/COM_file). It also has several subcommands which are used to access specific [disk sectors](https://en.wikipedia.org/wiki/Disk_sector), [I/O ports](https://en.wikipedia.org/wiki/I/O_port) and [memory addresses](https://en.wikipedia.org/wiki/Memory_address).

To create a program in assembler two options exist, the first one is to use the assembler program such as TASM or Turbo Assembler from Borland, and the second one is to use the debugger - on this first section we will use this last one since it is found in any PC with the MS-DOS, which makes it available to any user who has access to a machine with these characteristics.

Debug can only create files with a .COM extension, and because of the characteristics of these kinds of programs they cannot be larger than 64 kb, and they also must start with displacement, offset, or 0100H memory direction inside the specific segment.

Debug provides a set of commands that lets you perform a number of useful operations:

**A** Assemble symbolic instruction into machine code

**D**  Display the contents of an area of memory

**E** Enter data into memory, beginning at a specific location

**G** Run the executable program in memory

**N** Name a program

**P** Proceed, or execute a set of related instructions

**Q** Quit the debug program

**R** Display the contents of one or more registers

**T** Trace the contents of one instruction

**U** Unassembled machine code into symbolic code

**W** Write a program onto disk

* It is possible to visualize the values of the internal registers of the CPU using the Debug program. To begin working with Debug, type the following prompt in your computer:

**C:/>Debug [Enter]**

* On the next line a dash will appear, this is the indicator of Debug, at this moment the instructions of Debug can be introduced using the following command:

**-r[Enter]**

AX=0000 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000

DS=0D62 ES=0D62 SS=0D62 CS=0D62 IP=0100 NV EI PL NZ NA PO NC

0D62:0100 2E CS:

0D62:0101 803ED3DF00 CMP BYTE PTR [DFD3],00 CS:DFD3=03

* All the contents of the internal registers of the CPU are displayed; an alternative of viewing them is to use the "r" command using as a parameter the name of the register whose value wants to be seen.

**For example:**

-rbx

BX 0000

:

**IMMEDIATE ADDRESSING:**

In assembly language code lines have two parts, the first one is the name of the instruction which is to be executed, and the second one are the parameters of the command. For example:

**Add ah, bh**

Here "add" is the command to be executed; in this case an addition, and "ah" as well as "bh" are the parameters. **For example:**

**Mov al, 25**

In the above example, we are using the instruction mov, it means move the value 25 to al register. The name of the instructions in this language is made of two, three or four letters. These instructions are also called mnemonic names or operation codes, since they represent a function the processor will perform.

**Creating basic assembler program:**

a 100[enter]

mov ax,0002[enter]

mov bx,0004[enter]

add ax,bx[enter]

nop[enter][enter]

C:\>debug

-a 100

0D62:0100 mov ax,0002

0D62:0103 mov bx,0004

0D62:0106 add ax,bx

0D62:0108 nop

0D62:0109

* Type the command "t" (trace), to execute each instruction of this program, example

-t

AX=0002 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000

DS=0D62 ES=0D62 SS=0D62 CS=0D62 IP=0103 NV EI PL NZ NA PO NC

0D62:0103 BB0400 MOV BX,0004

* You see that the value 2 moves to AX register. Type the command "t" (trace), again, and you see the second instruction is executed.

-t

AX=0002 BX=0004 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000

DS=0D62 ES=0D62 SS=0D62 CS=0D62 IP=0106 NV EI PL NZ NA PO NC

0D62:0106 01D8 ADD AX,BX

* Type the command "t" (trace) to see the instruction add is executed, you will see the follow lines:

-t

AX=0006 BX=0004 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000

DS=0D62 ES=0D62 SS=0D62 CS=0D62 IP=0108 NV EI PL NZ NA PE NC

0D62:0108 90 NOP

**DIRECT ADDRESSING**

Sometimes instructions are used as follows:

**Add al,[170]**

The brackets in the second parameter indicate to us that we are going to work with the content of the memory cell number 170 and not with the 170 value; this is known as direct addressing.

**EXERCISE:**

TASK#1

Edit the contents of the above memory locations 120,133,122 by 02, 04, and 03 respectively using **E** command.

TASK#2

Then again display the contents of the memory locations which we edit in the Task# 2.

TASK#3

Add the contents of the above defined memory location using **mov** instruction.

TASK#4

Subtract the content of 120 locations by 133 and then store the result in the120 location and add the new 120 location contents with the content of 122 location.