Experiment 0 - Introduction to DEBUG and the Assembly Process

This experiment will introduce you to DEBUG and TASM, allowing you to become familiar with the process of assembling, debugging and executing an assembly language program with a PC. DEBUG is a program available with every version of WINDOWS and DOS. Here you will learn how to use DEBUG to assemble, disassemble, execute and debug assembly language programs with a PC. You will also be instructed on how to examine and modify the memory and CPU registers of your PC. TASM (Turbo Assembler) is the assembler you will use in the lab this semester. One may find many assemblers like: MASM, NASN, TASM, etc, on the web. I recommend you download one to your computer at home so that you may work on your project away from the lab. In this experiment you will copy, edit, assemble, link, debug and execute the program. During this process you will be using several programs, such as Turbo Assembler, Turbo Linker, Debug, and Notepad or Edit to accomplish your task. The knowledge acquired in this experiment will be extremely useful when working with the programs you will write during the semester. You may want to modify and save the program so that you can use it as the starting point when programming you next assignments. Do that by eliminating the red italicized statements.

The following convention will be used with all examples shown in this manual:

CAPITALIZED ITALICS REPRESENTS THE INFORMATION TYPED BY THE USER.

CAPITALIZED BOLD REPRESENTS THE COMPUTERS RESPONSE.

It is also suggested that the students use a floppy disk in drive A: or a pen drive to store their programs. This will avoid the possibility of having your programs copied in case you forget to erase them from drive C:.

Write the answers to the questions in this experiment on the space provided, and then hand the answer sheet to the Instructor before leaving the lab.

Loading DEBUG

1 – Open a DOS window by double clicking the DOS WINDOW icon on the desktop or in the Programs Menu.

2 – Issue the following command at the DOS prompt to load DEBUG:

```
A:\DEBUG <ENTER>
```

What prompt is displayed? _____________

Exiting DEBUG

3 – Issue the following command at the DEBUG prompt to exit DEBUG:

```
-Q<ENTER>
```

What prompt is displayed? ______________

Examining and modifying the contents of registers

4 – Load DEBUG and at the DEBUG prompt enter the following command

```
-R<ENTER>
AX=0000  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0B2C  ES=0B2C  SS=0B2C  CS=0B2C  IP=0100   NV UP EI PL NZ NA PO NC
0B2C:0100 7509          JNZ   010B
-R CX<ENTER>
CX 0000
:0008<ENTER>
-R<ENTER>
AX=0000  BX=0000  CX=0008  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0B2C  ES=0B2C  SS=0B2C  CS=0B2C  IP=0100   NV UP EI PL NZ NA PO NC
0B2C:0100 7509          JNZ   010B
```

Compare your display with the one shown above and discuss any possible discrepancies below.

```
-R CX<ENTER>
CX 0008
:321<ENTER>
-R<ENTER>
AX=0000  BX=0000  CX=0321  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0B2C  ES=0B2C  SS=0B2C  CS=0B2C  IP=0100   NV UP EI PL NZ NA PO NC
0B2C:0100 7509          JNZ   010B
-
```

Note that DEBUG displays all numeric values as hexadecimal numbers and that if you enter a value smaller than 4 digits, i.e. 321, DEBUG will pad it with zeros, 0321, when it writes them to the register.

5 – Modify the contents of register DX to 1F54.

6 – Now modify the contents of register DL from 54 to 68 without modifying the contents of register DH which you set to 1F in the previous step. Show the Instructor the results of steps 5 and 6.
7 – Enter the command below and tell me what happens:

```
-R AH<ENTER>
```

____

____

____

____

____

Assembling, disassembling, and executing programs
8 – Below you will find a series of commands to assemble and execute a program that adds the contents of registers AX and BX. Practice the procedure by entering the same information and verifying the results displayed. Note that the segment address may be different than 0B2C.

```
-A 100<ENTER>
0B2C:0100 MOV AX,1<ENTER>
0B2C:0103 MOV BX,2<ENTER>
0B2C:0106 ADD AX,BX<ENTER>
0B2C:0108 INT 3<ENTER>
0B2C:0109<ENTER>
-R<ENTER>
AX=0000  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0B2C  ES=0B2C  SS=0B2C  CS=0B2C  IP=0100   NV UP EI PL NZ NA PO NC
0B2C:0100 B80100        MOV   AX,0001
-G<ENTER>
AX=0003  BX=0002  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0B2C  ES=0B2C  SS=0B2C  CS=0B2C  IP=0108   NV UP EI PL NZ NA PE NC
0B2C:0108 CC            INT   3
-
```

You were exposed to two new commands:

A - used for the assembly of programs.
Format: A <starting address>.
The starting address may be given as an offset to the code segment address. This was done in the example above.

G - used for the execution of programs.
Format: G <=starting address> <breakpoint addresses>.

=starting address - Specifies the address that program execution will begin at. Note that the equal sign is used to differentiate the starting address from the breakpoint addresses. Execution will start at CS:IP if no start address is given. If the program ends with an INT 20 instruction, the IP register is reset back to offset 100, and if the program ends with an INT 3 instruction, the IP register will remain pointing to the next instruction after the last instruction executed.

breakpoint addresses - Specifies from 1 to 10 breakpoints that can be entered with the Go command. The program stops at the first breakpoint that it encounters and dumps the contents of all registers, the status of the FLAGS and displays the last instruction that was executed. Try the following command then write an explanation of what you observed during its execution:

```
-G =100 103 106<ENTER>
```

9 – Shown below are two formats for the command used to disassemble the program given in the example above. The first format uses a beginning and an ending address, and the second format uses a beginning address and a count of the number of bytes to be disassembled. Notice that the count is preceded by an ‘L’.

```
-U 100 108<ENTER>
0B2C:0100 B80100        MOV   AX,0001
0B2C:0103 BB0200        MOV   BX,0002
0B2C:0106 01D8          ADD   AX,BX
0B2C:0108 CC            INT   3
-U 100 L9<ENTER>
0B2C:0100 B80100        MOV   AX,0001
0B2C:0103 BB0200        MOV   BX,0002
0B2C:0106 01D8          ADD   AX,BX
0B2C:0108 CC            INT   3
-
```

10 – Write a program to subtract the content of register DX from the content of register AX, then add the result to the content of CX. Set the registers to 4, 0A and 1F respectively. Make sure to perform the following in the order given: assemble, and execute the program, then write the value of the registers shown below, and the instructions with the respective opcodes of your program:

AX = _____ BX=_____ CX=_____ CS=_____ IP=_____

Instruction Opcode

Tracing the execution of your program

11 – The trace command T is used to trace the execution of a program by displaying register information after the execution of the each instruction in the selected range.

Format: T <=starting address> <number of instructions>

Like the Go command if the starting address is not specified, it starts execution at CS:IP.

```
-T =100 4<ENTER>

AX=0001  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0B2C  ES=0B2C  SS=0B2C  CS=0B2C  IP=0103   NV UP EI PL NZ NA PO NC
0B2C:0103 BB0200        MOV   BX,0002

AX=0001  BX=0002  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0B2C  ES=0B2C  SS=0B2C  CS=0B2C  IP=0106   NV UP EI PL NZ NA PO NC
0B2C:0106 01D8          ADD   AX,BX

AX=0003  BX=0002  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0B2C  ES=0B2C  SS=0B2C  CS=0B2C  IP=0108   NV UP EI PL NZ NA PE NC
0B2C:0108 CC            INT   3

AX=0003  BX=0002  CX=0000  DX=0000  SP=FFE8  BP=0000  SI=0000  DI=0000
DS=0B2C  ES=0B2C  SS=0B2C  CS=0590  IP=13B1   NV UP DI PL NZ NA PE NC
0590:13B1 55            PUSH  BP
-
```

Check the example above then trace the execution of your program. Show the results to the instructor.

Accessing and modifying data in DEBUG

12 – In this section you will be exposed to the three commands: F – fill, D – dump, and E – enter. This command's address reference the data segment (DS). If you need to access information in another segment you need to include the segment in the address. The command's description and usage examples are given below:

F – used to fill blocks of memory with data.

Format: F <starting address> <ending address> <data>
D <starting address> <L number of bytes> <data>

D – used to display the memory content.

Format: D <starting address> <ending address>
D <starting address> <L number of bytes>

The starting and ending addresses may be given as offsets in the data segment. If access to another segment is required then the segment information should be included in the address, example: F CS:100 1FF 20

E – used to enter information in memory.

Format: E <address> <data list>
E <address>

If the E command is used without the data list, DEBUG assumes that you wish to examine that byte of memory and possibly modify it. The following options are given to you in that case:

a – You may enter a new data byte which DEBUG will write to memory.

b – You may press <ENTER> to signify you do not wish to modify the byte.

c – You may press the space bar which will leave the displayed byte unchanged and move to the next byte where you may possibly modify it.

d – You may enter the minus sign, which will leave the displayed byte unchanged and move you to the previous byte where you may possibly modify it.

See examples below:

```
-F 100 11F 20<ENTER>
-F 120 13F 30<ENTER>
-D 100 13F<ENTER>
0B2C:0100  20 20 20 20 20 20 20 20-20 20 20 20 20 20 20 20
0B2C:0110  20 20 20 20 20 20 20 20-20 20 20 20 20 20 20 20
0B2C:0120  30 30 30 30 30 30 30 30-30 30 30 30 30 30 30 30
0000000000000000
0B2C:0130  30 30 30 30 30 30 30 30-30 30 30 30 30 30 30 30
0000000000000000
-F 140 L20 31<ENTER>
-D 140 L20<ENTER>
0B2C:0140  31 31 31 31 31 31 31 31-31 31 31 31 31 31 31 31
1111111111111111
0B2C:0150  31 31 31 31 31 31 31 31-31 31 31 31 31 31 31 31
1111111111111111

-E 100 'Gabi'<ENTER>
-D 100 10F<ENTER>
0B2C:0100  47 61 62 69 20 20 20 20-20 20 20 20 20 20 20 20   Gabi
-E 100<ENTER>
0B2C:0100  47.67<ENTER>
```

```
-D 100 10F<ENTER>
0B2C:0100  67 61 62 69 20 20 20 20-20 20 20 20 20 20 20 20   gabi
-E 100<ENTER>
0B2C:0100  67.<SPACE BAR> 61.41<SPACE BAR>62.42<SPACE BAR>69.49<ENTER>
-D 100 10F<ENTER>
0B2C:0100  67 41 42 49 20 20 20 20-20 20 20 20 20 20 20 20   gABI
-
```

13 – Fill the memory locations 100 to 12F with the ASCII character which represents the number 5, then display the modified memory locations.
14 – Enter EE2730 in memory location 130, and then display those memory locations. Then using the E 130 command, modify the 2730 characters to 3751. Notice that the data list was excluded. Demonstrate to the instructor the procedures performed above.

Using Redirection with DEBUG
15 – Use Notepad to create a file called INPUT.TXT with the following contents:
F 150 L10 41<ENTER>
F 160 L10 61<ENTER>
D 150 L20<ENTER>
Q<ENTER>

Make sure this file is in the C:\WORK subdirectory or in the A: drive, whichever one you are using. Issue the following command at the DOS prompt:

```
\A:\DEBUG < INPUT.TXT<ENTER>
-F 150 L10 41
-F 160 L10 61
-D 150 L20
0B92:0150  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41
AAAAAAAAAAAAAAAA
0B92:0160  61 61 61 61 61 61 61 61-61 61 61 61 61 61 61 61
aaaaaaaaaaaaaaaa
-Q
A:\
```

Issue the following command next:

```
A:\DEBUG < INPUT.TXT > OUTPUT.TXT<ENTER>
A:\
```

Open the file OUTPUT.TXT with Notepad and check its contents.

Print file to be submitted to TA at end of lab.

Editing, assembling, linking and executing an Assembly Language program.

```
        .MODEL SMALL
        .386
        .STACK 64
        .DATA
```

```
MESS        DB     'Hello World!',13,10,'$'
            .CODE
BEGIN       PROC FAR
            MOV AX,@DATA
            MOV DS,AX

            MOV AH,9H
            MOV DX,OFFSET MESS
            INT 21H

            MOV AH,4CH
            INT 21H
BEGIN       ENDP
            END BEGIN
```

16 – Copy the program above using **Notepad**, and save it as **PROG0A.ASM** in the **A:\>** drive.

17 – Following you will find the commands that you should type to accomplish the task of assembling, linking and using DEBUG to execute the program given to you.

```
A:\>TASM PROG0A.ASM /L<ENTER>
Turbo Assembler  Version 3.1  Copyright (c) 1988, 1992 Borland
International
Assembling file:   PROG0A.ASM
Error messages:    None
Warning messages:  None
Passes:            1
Remaining memory:  435k
```

If there are any problems during assembly you will see several error messages displayed on a DOS window opened by the OS. At this point you may open the file **PROG0A.LST** to check where the errors occur and then edit them in the **PROG0A.ASM** file, before assembling it again, otherwise continue below.

```
A:\>TLINK PROG0A.OBJ<ENTER>
Turbo Link  Version 5.1  Copyright (c) 1992 Borland International
```

Now you are going to use DEBUG to execute the program.

```
A:\>DEBUG PROG0A.EXE<ENTER>
-G<ENTER>
Hello World!

Program terminated normally
-Q<ENTER>
```

```
A:\>
```

You can also execute the program directly by doing what is shown below:

```
A:\>PROG0A.EXE<ENTER>
Hello World!

A:\>
```

18 – Copy the program below using **Notepad**, and save it as **PROG0B.ASM** in the **A:\>** drive.

```
            .MODEL SMALL
            .386
            .STACK 64
            .DATA
N1          DB     33H
N2          DB     24H
SUM         DB     0H
            .CODE
BEGIN       PROC FAR
            MOV AX,@DATA
            MOV DS,AX

            MOV AL,N1
            ADD AL,N2
            MOV SUM,AL

            MOV AH,4CH
            INT 21H
BEGIN       ENDP
            END BEGIN
```

19 – Following you will find the commands that you should type to accomplish the task of assembling, linking and using DEBUG to execute the program given to you.

```
A:\>TASM PROG0B.ASM /L<ENTER>
Turbo Assembler  Version 3.1  Copyright (c) 1988, 1992 Borland
International
Assembling file:   PROG0B.ASM
Error messages:    None
Warning messages:  None
Passes:            1
Remaining memory:  435k
```

If there are any problems during assembly you will see several error messages displayed on a DOS window opened by the OS. At this point you may open the file **PROG0B.LST** to check where the errors occur and then edit them in the **PROG0B.ASM** file, before assembling it again, otherwise continue below.

```
A:\>TLINK PROG0B.OBJ<ENTER>
Turbo Link  Version 5.1  Copyright (c) 1992 Borland International
```

Now you are going to use DEBUG to execute the program and to verify memory to see if the program executed correctly. Follow the steps below exactly.

```
A:\>DEBUG PROG0B.EXE<ENTER>
```

The next two steps allow you to find out the initial address of the data segment, and its contents. You will do this by disassembling the first instruction (MOV AX,@DATA) of your program, then dumping the contents of several memory locations starting at the address you just found.

```
-U CS:0 1<ENTER>
10A2:0000 B8A410         MOV AX,10A4
-D 10A4:0 2<ENTER>
10A4:0000  33 24 00                                             3$.
```

To execute the program, and to display the memory, follow the steps below. These steps allow you to verify if the program executed correctly.

```
-G<ENTER>

Program terminated normally
-D 10A4:0 2<ENTER>
10A4:0000  33 24 57                                             3$W

-Q<ENTER>
A:\>
```

Procedure

20 - Using the information learned from the previous steps, create a new program called **PROG0C.ASM** to perform PRODUCT = A * B * C.

DATA segment array definition.

```
.DATA
A           DB    4H
B           DB    10H
C           DB    7FH
PRODUCT     DW    ?
```

Dump the memory where the data segment is located at and check the result of the multiplication you just performed. Show the instructor the results and hand him your observations made during the lab.