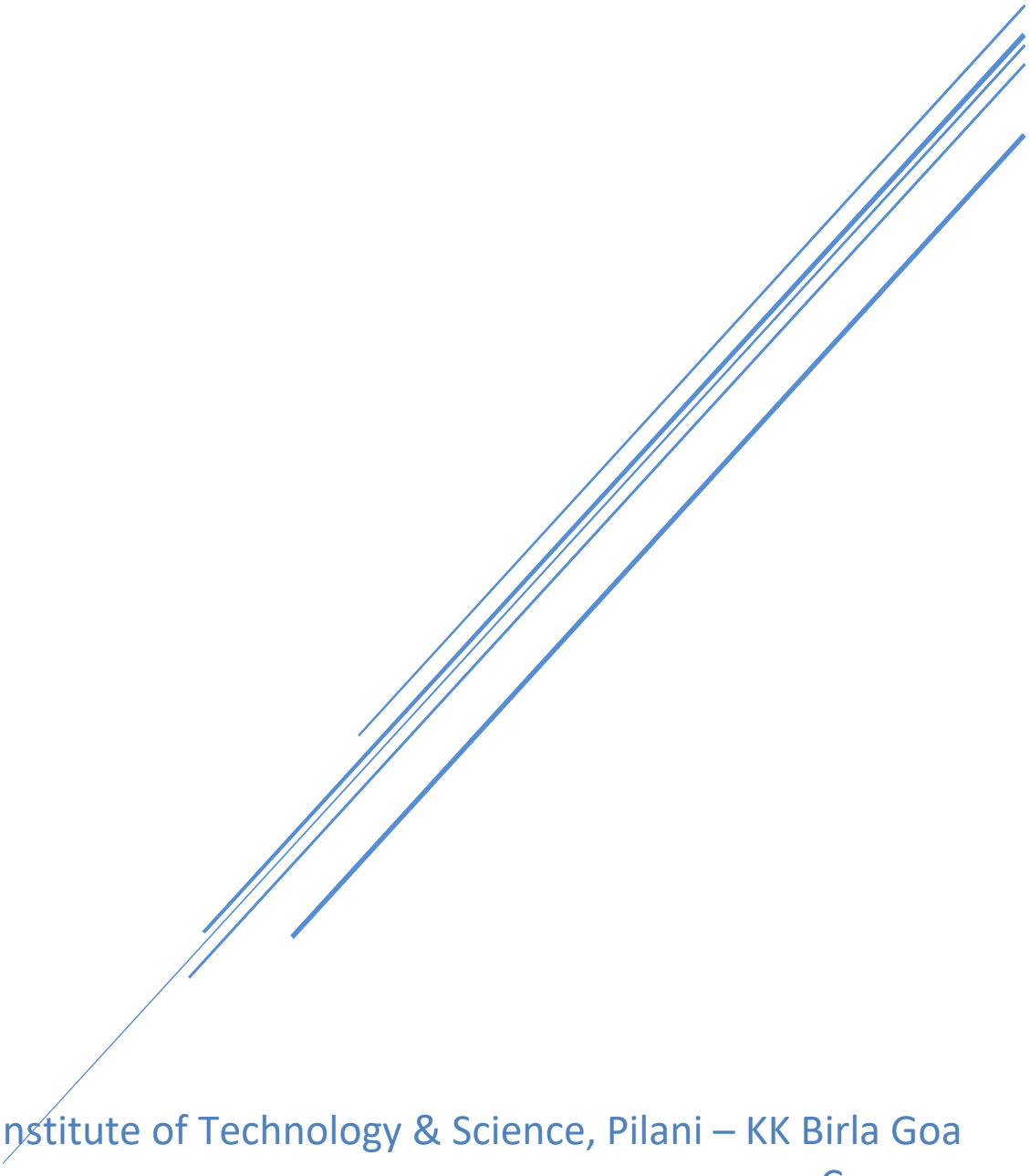
Laboratory Manual

Introduction to DEBUG & DEBUGX



Birla Institute of Technology & Science, Pilani – KK Birla Goa Campus

K.R.Anupama

Lab1- Introduction to DEBUG & DEBUGX

1.0 Introduction to DEBUGX

DEBUG, supplied by MS-DOS, is a program that traces the 8086 instructions. Using DEBUG, you can easily enter 8086 machine code program into memory and save it as an executable MS-DOS file (in .COM/.EXE format). DEBUG can also be used to test and debug 8086 and 8088 programs. The features include examining and changing memory and register contents including the CPU register. The programs may also be single-stepped or run at full speed to a break point.

You will be using DEBUGX which is a program similar to DEBUG but offers full support for 32-bit instructions and addressing. DEBUGX includes the 80x86 instructions through the Pentium instructions.

1.1 How to start DEBUGX

At a MS-DOS prompt (Available in accessories of windows start-up if you have a Windows 7 or lesser system/else use DosBox), type DEBUGX

Type a question mark (?) to get a list of available commands.

You may return to the DOS prompt by entering Q and carriage return (<CR>).

1.2 Rules of DEBUGX

- It does not distinguish between lower case and upper case letters.
- It assumes that all numbers are in Hexadecimal unlike Microsoft assembler, which assumes numbers to be decimal.
- Segment and offset are specified as segment: offset.
- Spaces in commands are used only to separate parameters

1.3 Some DEBUGX Commands

RX – toggles between 32 bit and 16 bit register mode

1.3.1 Commands common to DEBUG & DEBUGX

A	Assembles symbolic instructions into machine code
D	Display the contents of an area of memory in hex format
E	Enter data into memory beginning at specific location
G	Run the executable program in the memory (Go)
P	Proceed, Execute a set of related instructions
Q	Quit the debug session
R	Display the contents of one or more registers in hex format
T	Trace the execution of on instruction
U	Unassemble machine code into symbolic code
?	Debug Help

1.4 R, the Register command: examining and altering the content of registers

In DEBUGX, the register command "R" allows you to examine and/or alter the contents of the internal CPU registers.

R <register name>

The "R" command will display the contents of all registers unless the optional <register name> field is entered. In which case, only the content of the selected register will be displayed. The R command also displays the instruction pointed to by the IP. The "R" command without the register name field, responds with three lines of information. The first two lines show you the programmer's model of the 80x86 microprocessor. (The first line displays the contents of the general-purpose, pointer, and index registers. The second line displays the current values of the segment registers, the instruction pointer, and the flag register bits. The third line shows the location, machine code, and Assembly code of the next instruction to be executed.)

If the optional <register name> field is specified in the "R" command, DEBUGX will display the contents of the selected register and give you an opportunity to change its value. Just type a <return> if no change is needed, e.g.

- r ECX

ECX 00000000

: FFFF

- r ECX

ECX 0000FFFF

:

Note that DEBUGX pads input numbers on the left with zeros if fewer than four digits are typed in for 16-bit register mode and if fewer than eight digits in 32 bit addressing mode

1.5 A, the Assemble command

The assemble command is used to enter Assembly language instructions into memory.

A <starting address>

The starting address may be given as an offset number, in which case it is assumed to be an offset into the code segment. Otherwise, CS can also be specified explicitly. (eg. "A 100" and "A CS:100" will achieve the same result). When this command is entered, DEBUG/DEBUGX will begin prompting you to enter Assembly language instructions. After an instruction is typed in and followed by <return>, DEBUG/DEBUGX will prompt for the next instruction. This process is repeated until you type a <return> at the address prompt, at which time DEBUG/DEBUGX will return you to the debug command prompt.

Use the “A” command to enter the following instructions starting from offset 0100H.

32-bit format	**16-bit format**
A 100	**- A 100**
xxxx:0100 mov eax,1	**xxxx:0100 mov ax,1**
xxxx:0106 mov ebx,2	**xxxx:0103 mov bx,2**
xxxx:010C mov ecx,3	**xxxx:0106 mov cx,3**
xxxx:0112 add eax, ecx	**xxxx:0109 add ax, cx**
xxxx:0115 add eax, ebx	**xxxx:010B add ax, bx**
xxxx:0118 jmp 100	**xxxx:010D jmp 100**
xxxx:011A <enter>	**xxxx:010F <enter>**

Where xxxx specifies the base value of instruction address in the code segment. Please note that the second instruction starts at xxxx:0106 in 32 bit format and xxxx:0103 in 16-bit format. This implies that first instruction is six bytes long in 32-bit format and three byte long in 16-bit format.

Similarly note the size of all instructions.

When DEBUGX is first invoked, what are the values in the general-purpose registers?

What is the reason for setting [IP] = 0100 ?

1.6 U, the Unassemble command: looking at machine code

The unassemble command displays the machine code in memory along with their equivalent Assembly language instructions. The command can be given in either format shown below:

U <starting address> <ending address>

32-bit format	**16-bit format**
U 100 118	**U 100 10D**
xxxx:0100 66B801000000 mov eax,01	**xxxx:0100 B80100 mov ax,1**
xxxx:0106 66BB02000000 mov ebx,02	**xxxx:0103 BB0200 mov bx,2**
xxxx:010C 66B903000000 mov ecx,03	**xxxx:0106 B90300 mov cx,3**
xxxx:0112 6603C3 add eax, ebx	**xxxx:0109 03C3 add ax,bx**
xxxx:0115 6603C1 add eax, ecx	**xxxx:010B 03C1 add ax,cx**
xxx:0118 EBE6 jmp 0100	**xxxx:010D EBF1 jmp 100**

All the data transfer and arithmetic instructions in 32-bit format have 66 as a prefix why?

1.7 E, the Enter Command

E address

You can modify the content of any memory location in the data segment using the "E" command

1.8 T, the Trace command: single-step execution

The trace command allows you to trace through the execution of your programs one or more instructions at a time to verify the effect of the programs on registers and/or data.

T < =starting address>

T =100

If you do not specify the starting address then you have to set the IP using the R command before using the T command

Usage

Using r command to set IP to 100 and then execute T 5

Using r command to set IP to 100 and then execute T

Trace through the above sequence of instructions that you have entered and examine the registers and Flag registers

1.9 G, the Go command

The go command instructs DEBUG to execute the instructions found between the starting and stop addresses.

G < = starting address > < stop address >

Execute the following command and explain the result (this is with respect to code you have written in 32-bit format earlier)

G = 100 118

Caution: the command G= 100 119 will cause the system to hang. Explain why?

Often, it is convenient to pause the program after executing a few instructions, thus effectively creating a break point in the program. For example, the command "g 10c" tells the DEBUG to start executing the next instruction(s) and pause at offset 010CH.

The main difference between the GO and TRACE commands is that the GO command lists the register values after the execution of the last instruction while the TRACE command does so after each instruction execution

Try executing the sequence of instructions you entered using several options of the "G" command. Remember to reload 0100 into IP every time you use G without the staring address.

Tasks to be completed

Task1: ALP assembling and run.

1. Write an ALP that will exchange 8-bit data between location DS:0200 and DS:0210 using **A**.
    a. Do this using direct addressing, register relative, base- indexed
2. Store data in memory location DS:0200 and DS:0210 using **E** command
3. Run the program using **T**
4. Repeat the same ALP for exchanging 16-bit and 32-bit data.
5. Set IP to 100
6. Run the program using **G Last Address**
7. Find the Last Address using **U**
8. Verify the result by using **D**

Task2: To search for a string in a memory location.

1. Load a txt file having a string 'virus' anywhere in the file using ***N*** and ***L*** commands of DEBUGX.

'***N***' Names a program or a file you intend to read or write onto disk.

Format of command ***N*** path:\nameoffile.ext

For e.g. if your file is new.txt in folder D:\mpi – command will be ***N D:\mpi\new.txt***

2. Load File into memory using *L* command.

'***L***' Loads a File or Disk Sectors into memory.

L 100 will load the file specified by ***N*** into CS:100.

3. Search for string 'virus' using ***S*** command.

'***S***' Searches memory for a string.

Format ***S*** start-addr L val "data"

L – number of bytes in memory you want to search so for e.g. if you want to search for 50 bytes in memory starting from memory location DS: 500 for the string "mpi"; command format will be S 500 L 32 "mpi"

Format ***S*** start-addr end-addr "data"

If you want to search memory starting from location DS: 500 to DS:600 for the string "mpi"; command format will be S 500 600 "mpi"

Task3: To use DEBUGX to create a very small machine language program that you save on disk as a file.

1. Store data id DS:0200 using E command.

2. Using ***A*** command write a small ALP that will transfer the contents of this memory location into AX, BX, CX, DX, SI, DI registers.

3. Use ***N*** command to create a file. (Note: The executable file that you will create will have .com extension)

4. Enter value '0' in BX and size of program in 'CX' using R command

5. Using ***W*** command now write executable program.

   Format ***W*** address e.g. W 100 (100 is the offset at which your ALP is)

6. Exit and again open DEBUGX with 'name of file'.com and trace through the program and examine contents of registers e.g. DEBUGX file.com

Task4: Move a string from location DS:500 to DS:600

Explore ***M*** command.

Format ***M src_start address L no.of bytes dst_start address***

Task5: What happens if you type ***H*** 14F 22?

Task6: Fill a set of memory location with ASCII character 'H'

Explore ***F*** command.

Format ***F start-addr L val "data"***