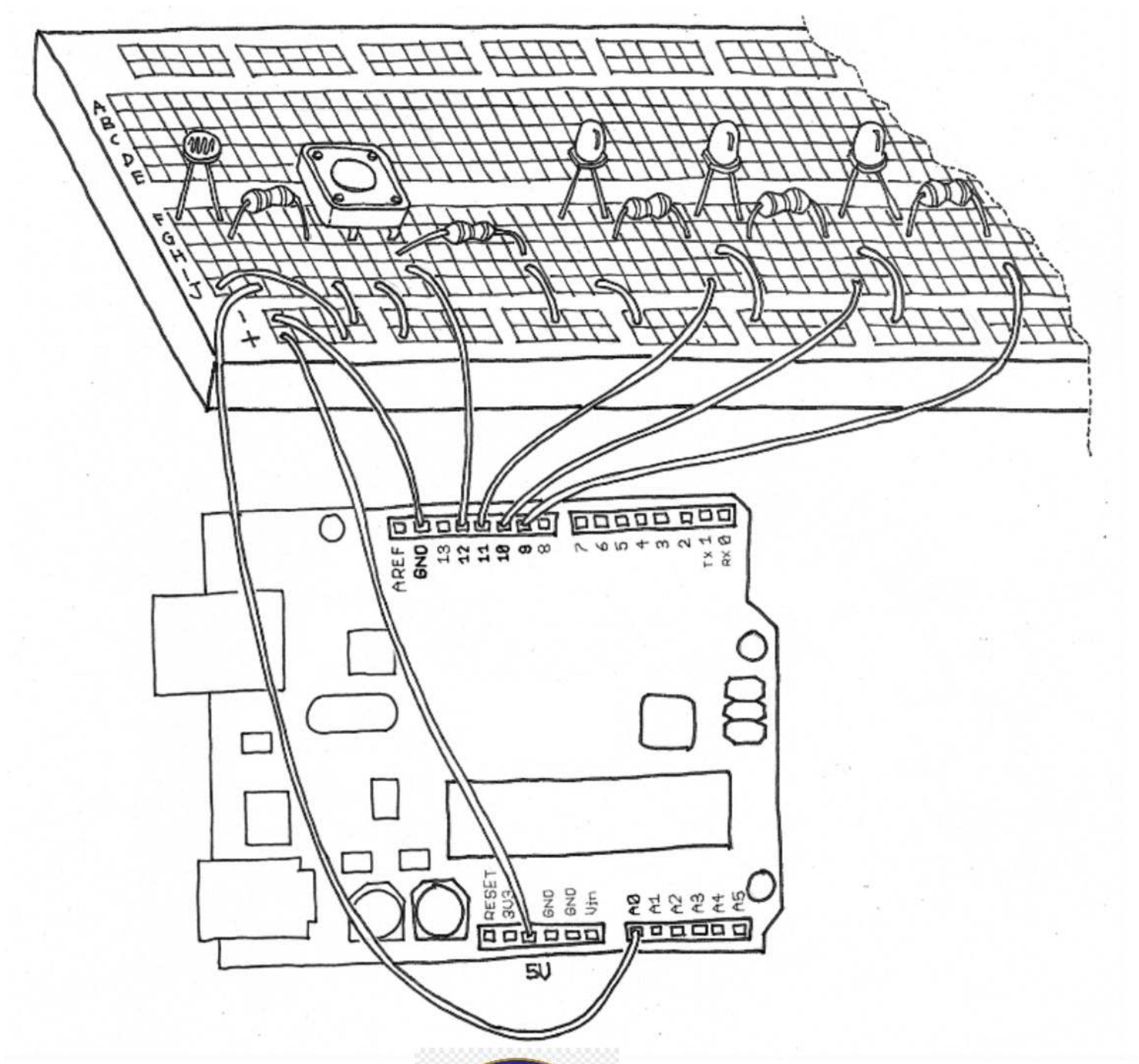
CS/EEE/INSTR F241
Microprocessor Programming and Interfacing

Lab 2- Debugx Commands in Detail



Dr. Vinay Chamola and Anubhav Elhence

Lab 2 - DEBUGX Commands in Detail

After completing the previous lab session, we hope you are more confident with using DEBUGX. In this lab session, we start with a recap of using DebugX and an overview of the commands. This is followed by a more detailed exploration of the commands. Finally, you will be exposed to a few other commands in the Tasks for this session.

DEBUGX Recap

DEBUGX:-

- **Numbers in Hexadecimal by default**
- **NOT Case Sensitive**
- **Segment:Offset**
- **16-bit registers by default (RX – toggles between 32-bit and 16-bit register mode)**

Starting DEBUGX:-

- **Click on the DOSBox icon**
- **At the command prompt, type '*mount d: d:\masm611\bin*' [Replace 'd' with the drive containing the folder MASM611]**
- **Again type '*d:*'**
- **Finally, type '*debugx*'**

Commands Overview:-

- **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**

Commands in Detail

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

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

A 100

```
xxxx:0100 mov eax,1

xxxx:0106 mov ebx,2

xxxx:010C mov ecx,3

xxxx:0112 add eax, ecx

xxxx:0115 add eax, ebx

xxxx:0118 jmp 100

xxxx:011A <enter>
```

<u>**16-bit format**</u>

```
- A 100

xxxx:0100 mov ax,1

xxxx:0103 mov bx,2

xxxx:0106 mov cx,3

xxxx:0109 add ax, cx

xxxx:010B add ax, bx

xxxx:010D jmp 100

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 ?

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

```
U 100 118

xxxx:0100 66B801000000 mov eax,01

xxxx:0106 66BB02000000 mov ebx,02

xxxx:010C 66B903000000 mov ecx,03
```

xxxx:0112 6603C3 add eax, ebx

xxxx:0115 6603C1 add eax, ecx

xxx:0118 EBE6 jmp 0100

16-bit format

U 100 10D

xxxx:0100 B80100 mov ax,1

xxxx:0103 BB0200 mov bx,2

xxxx:0106 B90300 mov cx,3

xxxx:0109 03C3 add ax,bx

xxxx:010B 03C1 add ax,cx

xxxx:010D EBF1 jmp 100

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

E, the Enter Command

The Enter command (E) is used to enter data into memory. The required address parameter specifies the starting location at which to enter the data. If the address parameter does not specify a segment, DEBUG uses DS. If the optional list parameter is not included, DEBUG displays the byte value at the specified address and prompts for a new value. The spacebar or minus key can be used to move to the next or previous byte. Pressing the Enter key without entering a value terminates the command.

```
E <memory_address>
```

Examples:

```
-E 100 01 02 03
```

```
-E 100 'ABC'
```

```
-E 100 'ABC' 0
```

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

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 breakpoint 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 starting address.

Please refer to the file MASM611\BIN\DEBUG.txt for further details about the commands. It is very well documented.

N, the Name command

The Name command (N) is used to input a filename (actually, a file specification that can include a drive and a path) for a subsequent Load or Write command. Note that this command *does not create a new file*. It is meant for files already present in the BIN folder (files outside of this may result in "Access Denied" errors).

```
N filename.txt
```

L, the Load command

The Load command (L) is used to load a file into the specified memory address. *The file to load is specified with a preceding Name command (N)*. The optional address parameter specifies the load address. *The default load address is CS:100* for all files other than EXE files, for which information in the file header determines the load address. After DEBUGX loads the file it *sets BX:CX to the file size in bytes.*

```
L <memory_address>
```

Tasks to be Completed

Create a .txt file that contains your First Name in small letters followed by "MUP," separated by a '*.' For example, if your name is 'Apple,' then the content of the file should be

Apple*MUP

Now place this string in the Extra Segment at 0200h, with the "*" replaced by "[". Place an additional "]" at the end of the string too.

ES:0200 *Apple[MUP]*

(*Hint:* ASCII Characters

- **[= 5B**
- **] = 5D**
- *** = 2A**

1. Create a .txt file containing the above string in the BIN folder.

2. load the value 5B into AH and 5D into AL

3. Load the file you created in step 1 using the L command into the code segment (assume/use offset to code segment as 0200h)

4. Copy the values AH and AL into the appropriate locations using the MOV instruction with the appropriate offsets.)

BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI

ज्ञानं परमं बलम्