RECORD OF WORK DONE:

Sr no.		Experiment	Date of Experiment	Page No.	Signature
1	1.1	Study of DOS Debug Command.	09/07/2020	3	
2	2.1	Write an assembly language program for subtraction of two 16-bit numbers.	16/07/2020	23	
	2.2	Write an assembly language program to perform scalar multiplication of an array of five unsigned bytes.	16/07/2020	25	
3	3.1	Write an assembly language program for moving a string from one segment to another segment.	23/07/2020	29	
	3.2	Write an assembly language program to compare two strings of equal length.	23/07/2020	31	
	3.3	Write an assembly language program which accepts a character and a string from the user and prints the position of the character into the string if it is found, otherwise the message "NOT FOUND". For simplicity, enter the string with length in single digit, that is less than or equal to 9.	23/07/2020	35	
4	4.1	Write a multi module assembly program to divide 32-bit numbers by 16- bit numbers and return a 32-bit quotient.	06/08/2020	40	
	4.2	Write an assembly language program to develop a far procedure to find whether the given number is EVEN or ODD and prints the message appropriately. Write a main program to call this far procedure and pass the roll_no as a parameter to the far procedure.	06/08/2020	46	
5	5.1	Write an assembly language program of dividing four numbers. If the result of the division is too large to fit in the quotient register then the 8086 will do a type 0 interrupt immediately after the divide instruction finishes.	13/08/2020	49	
6	6.1	Write a C program to convert Celsius to Fahrenheit where the function "C2F" is assembly language function. Print the converted temperature in Fahrenheit from C program.	03/09/2020	57	
	6.2	Write a C program to convert Celsius to Fahrenheit where the functions "C2F" and "Show" are assembly language functions. (Note: Name, Roll and message you can print directly from the C program but to display converted temperature define show() function in assembly language.	03/09/2020	60	
7	7.1	Write a program to take one character from the keyboard and echo on screen.	10/09/2020	65	
	7.2	Write a program to take one character from keyboard and convert into lowercase.	10/09/2020	68	

RECORD OF WORK DONE:

	7.3	Write a program to get a string and convert this string from uppercase to lowercase.	10/09/2020	70	
	7.4	Here is a string “Hello Sunil Welcome to 8086 Microprocessor”. Write an assembly Language program to convert the above string to “Hollo Sunil Wolcomo to 8086 Microprocossor” (Replace ‘e’ with ‘o’).	10/09/2020	73	
8	8.1	Study of implementation of Recursion in assembly language.(Program to find Factorial using recursion)	15/10/2020	77	
9	9.1	Write an assembly language program to convert a 4-digit BCD number to binary. Use procedure and stack to pass parameters.	30/07/2020	80	
	9.2	Write an assembly language program to count the number of 1’s in the binary representation of 16-bit numbers using procedure and registers as parameter passing methods.	30/07/2020	84	
10	10.1	Active TSR using Hot key combination.	24/09/2020	87	
	10.2	Example of Active and Passive TSR(Screensaver).	24/09/2020	91	

IT081- Aanandi Pankhania - I1

Experiment -1

Aim: Study of DOS Debug Commands:

1 INTRODUCING DEBUG

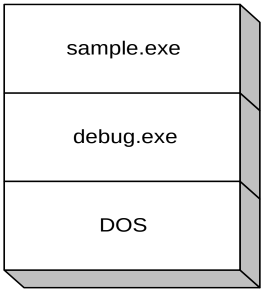
As you begin to learn assembly language programming, the importance of using a program called a *debugger* cannot be stressed too much. A debugger displays the contents of memory and lets you view registers and variables as they change. You can step through a program one line at a time (called *tracing*), making it easier to find logic errors. In this appendix, we offer a tutorial on using the debug.exe program that is supplied with both DOS and Windows (located in the \Windows\Command directory). From now on, we will just call this program *Debug*. Later, you will probably want to switch to a more sophisticated debugger such as Microsoft CodeView or Borland Turbo Debugger. But for now, Debug is the perfect tool for writing short programs and getting acquainted with the Intel microprocessor.

You can use Debug to test assembler instructions, try out new programming ideas, or to carefully step through your programs. It takes supreme overconfidence to write an assembly language program and run it directly from DOS the first time! If you forget to match pushes and pops, for example, a return from a subroutine will branch to an unexpected location. Any call or jump to a location outside your program will almost surely cause the program to crash. For this reason, you would be wise to run any new program you've written in Debug. Trace the program one line at a time, and watch the stack pointer (SP) very closely as you step through the program, and note any unusual changes to the CS and IP registers. Particularly when CS takes on a new value, you should be suspicious that your program has branched into the *Twilight Zone*®.

Debugging functions. Some of the most rudimentary functions that any debugger can perform are the following:

- Assemble short programs
- View a program's source code along with its machine code
- View the CPU registers and flags
- Trace or execute a program, watching variables for changes
- Enter new values into memory
- Search for binary or ASCII values in memory
- Move a block of memory from one location to another
- Fill a block of memory
- Load and write disk files and sectors

Many commercial debuggers are available for Intel microprocessors, ranging widely in sophistication and cost: CodeView, Periscope, Atron, Turbo Debugger, SYMDEB, Codesmith-86, and Advanced-Trace-86, to mention just a few. Of these, Debug is the simplest. The basic principles learned using Debug may then be applied to nearly any other debugger.



Debug is called an *assembly level* debugger because it displays only assembly mnemonics and machine instructions. Even if you use it to debug a compiled C++ program, for example, you will not see the program's source code. Instead, you will see a disassembly of the program's machine instructions.

To trace or execute a machine language program with Debug, type the name of the program as a command line parameter. For example, to debug the program sample.exe, you would type the following command line at the DOS prompt:

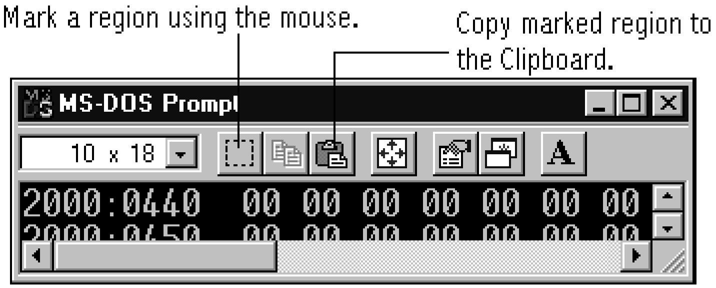
```
debug sample.exe
```

If we could picture DOS memory after typing this command, we would see DOS loaded in the lowest area, debug.exe loaded above DOS, and the program sample.exe loaded above Debug. In this way, several programs are resident in memory at the same time. DOS retains control over the execution of Debug, and Debug controls the execution of sample.exe.

Printing a Debugging Session (local printer only). If you have a printer attached directly to your computer, you can get a printed copy of everything you're doing during a debugging session by pressing the Ctrl-PrtScrn keys. This command is a toggle, so it can be typed a second time to turn the printer output off.

Printing a Debugging Session (network printer). If your computer is attached to a network and there is a printer on the network, printing a debugging session is a bit challenging. The best way we've found is to prepare a script file containing all the debug commands you plan to type. Run Debug, telling it to read its input from the script file, and have Debug send the output to another file. Then, print the output file in the same way you usually print on the network. In Windows, for example, the output file can be loaded into *Notepad* and printed from there. See the section later in this appendix entitled *Using Script Files with Debug.*

Using the Mark and Copy Operations in a DOS Window. Under Windows, when you run Debug in a window, a toolbar has commands that you can use to mark a section of the window, copy it to the clipboard, and paste it into some other application (such as Notepad or Word):


Mark a region using the mouse.
Copy marked region to the Clipboard.
MS-DOS Prompt
10 x 18
A

```
2000:0440   00 00 00 00 00 00 00
2000:0450   00 00 00 00 00 00 00
```

2 DEBUG COMMAND SUMMARY

Debug commands may be divided into four categories: program creation/debugging, memory manipulation, miscellaneous, and input-output:

Program Creation and Debugging

A	Assemble a program using instruction mnemonics
G	Execute the program currently in memory
R	Display the contents of registers and flags
P	Proceed past an instruction, procedure, or loop
T	Trace a single instruction
U	Disassemble memory into assembler mnemonics

Memory Manipulation

C	Compare one memory range with another
D	Dump (display) the contents of memory
E	Enter bytes into memory
F	Fill a memory range with a single value
M	Move bytes from one memory range to another
S	Search a memory range for specific value(s)

Miscellaneous

H	Perform hexadecimal addition and subtraction
Q	Quit Debug and return to DOS

Input-Output

I	Input a byte from a port
L	Load data from disk
O	Send a byte to a port
N	Create a filename for use by the L and W commands
W	Write data from memory to disk

Default Values. When Debug is first loaded, the following defaults are in effect:

1. All segment registers are set to the bottom of free memory, just above the debug.exe program.

2. IP is set to 0100h.

3. Debug reserves 256 bytes of stack space at the end of the current segment.

4. All of available memory is allocated (reserved).

5. BX:CX are set to the length of the current program or file.

6. The flags are set to the following values: NV (Overflow flag clear), UP (Direction flag = up), EI (interrupts enabled), PL (Sign flag = positive), NZ (Zero flag clear), NA (Auxiliary Carry flag clear), PO (odd parity), NC (Carry flag clear).

2.1 Command Parameters

Debug's command prompt is a hyphen (–). Commands may be typed in either uppercase or lowercase letters, in any column. A command may be followed by one or more parameters. A comma or space may be used to separate any two parameters. The standard command parameters are explained here.

Address. A complete segment-offset address may be given, or just an offset. The segment portion may be a hexadecimal number or register name. For example:

`F000:100`	`Segment, offset`
`DS:200`	`Segment register, offset`
`0AF5`	`Offset`

Filespec. A file specification, made up of a drive designation, filename, and extension. At a minimum, a filename must be supplied. Examples are:

```
b:prog1.com
c:\asm\progs\test.com
file1
```

List. One or more byte or string values:

```
10,20,30,40
'A','B',50
```

Range. A *range* refers to a span of memory, identified by addresses in one of two formats. In Format 1, if the second address is omitted, it defaults to a standard value. In Format 2, the value following the letter L is the number of bytes to be processed by the command. A range cannot be greater than 10000h (65,536):

Syntax	Examples
`Format 1: address [,address]`	`100,500 CS:200,300 200`

`Format 2: address L [value]  100 L 20` ***(Refers to the 20h bytes starting at location 100.)***

Sector. A sector consists of a starting sector number and the number of sectors to be loaded or written. You can access logical disk sectors Using the L (load) and W (write) commands.

String. A string is a sequence of characters enclosed in single or double quotes. For example:

```
'COMMAND'
"File cannot be opened."
```

Value. A value consists of a 1- to 4-character hexadecimal number. For example:

```
3A
3A6F
```

3 INDIVIDUAL COMMANDS

This section describes the most common Debug commands. A good way to learn them is to sit at the computer while reading this tutorial and experiment with each command.

3.1 ? (Help)

Press ? at the Debug prompt to see a list of all commands. For example:

Figure 1. Debug's List of Commands.

```
assemble         A [address]
compare          C range address
dump             D [range]
enter            E address [list]
fill             F range list
go               G [=address] [addresses]
hex              H value1 value2
input            I port
load             L [address] [drive] [firstsector] [number]
move             M range address
name             N [pathname] [arglist]
output                  O port byte
proceed          P [=address] [number]
quit             Q
register         R [register]
```

```
search                  S range list
trace             T [=address] [value]
unassemble        U [range]
write             W [address] [drive] [firstsector] [number]
```

3.2 A (Assemble)

Assemble a program into machine language. Command formats:

```
A
A address
```

If only the offset portion of *address* is supplied, it is assumed to be an offset from CS. Here are examples:

Example	Description
`A 100`	Assemble at CS:100h.
`A`	Assemble from the current location.
`A DS:2000`	Assemble at DS:2000h.

When you press Enter at the end of each line, Debug prompts you for the next line of input. Each input line starts with a segment-offset address. To terminate input, press the Enter key on a blank line. For example:

```
-a 100
5514:0100 mov ah,2
5514:0102 mov dl,41
5514:0104 int 21
5514:0106
```

(bold text is typed by the programmer)

3.3 C (Compare)

The C command compares bytes between a specified range with the same number of bytes at a target address. Command format:

```
C range address
```

For example, the bytes between DS:0100 and DS:0105 are compared to the bytes at DS:0200:

```
C 100 105 200
```

The following is displayed by Debug:

```
1F6E:0100  74  00  1F6E:0200
1F6E:0101  15  C3  1F6E:0201
1F6E:0102  F6  0E  1F6E:0202
1F6E:0103  C7  1F  1F6E:0203
1F6E:0104  20  E8  1F6E:0204
1F6E:0105  75  D2  1F6E:0205
```

3.4 D (Dump)

The D command displays memory on the screen as single bytes in both hexadecimal and ASCII. Command formats:

```
D
D address
D range
```

If no address or range is given, the location begins where the last D command left off, or at location DS:0 if the command is being typed for the first time. If *address* is specified, it consists of either a segment-offset address or just a 16-bit offset. *Range* consists of the beginning and ending addresses to dump.

Example	**Description**
`D F000:0`	Segment-offset
`D ES:100`	Segment register-offset
`D 100`	Sffset

The default segment is DS, so the segment value may be left out unless you want to dump an offset from another segment location. A range may be given, telling Debug to dump all bytes within the range:

```
D 150 15A          (Dump DS:0150 through 015A)
```

Other segment registers or absolute addresses may be used, as the following examples show:

Example	Description
`D`	Dump 128 bytes from the last referenced location.
`D SS:0 5`	Dump the bytes at offsets 0-5 from SS.
`D 915:0`	Dump 128 bytes at offset zero from segment 0915h.
`D 0 200`	Dump offsets 0-200 from DS.
`D 100 L 20`	Dump 20h bytes, starting at offset 100h from DS.

Memory Dump Example. The following figure shows an example of a memory dump. The numbers at the left are the segment and offset address of the first byte in each line. The next 16 pairs of digits are the hexadecimal contents of each byte. The characters to the right are the ASCII representation of each byte. This dump appears to be machine language instructions, rather than displayable characters.

Dump of offsets 0100h through 017Fh in COMMAND.COM:

```
-D 100
1CC0:0100  83 7E A4 01 72 64 C7 46-F8 01 00 8B 76 F8 80 7A  .~$.rdGFx...vx.z
1CC0:0110  A5 20 73 49 80 7A A5 0E-75 06 C6 42 A5 0A EB 3D  % sI.z%.u.FB%.k=
1CC0:0120  8B 76 F8 80 7A A5 08 74-0C 80 7A A5 07 74 06 80  .vx.z%.t..z%.t..
1CC0:0130  7A A5 0F 75 28 FF 46 FA-8B 76 FA 8B 84 06 F6 8B  z%.u(.Fz.vz...v.
1CC0:0140  7E F8 3A 43 A5 75 0C 03-36 A8 F4 8B 44 FF 88 43  ~x:C%u..6(t.D..C
1CC0:0150  A5 EB 0A A1 06 F6 32 E4-3B 46 FA 77 D8 8B 46 F8  %k.!.v2d;FzwX.Fx
1CC0:0160  40 89 46 F8 48 3B 46 A4-75 A1 A1 06 F6 32 E4 3B  @.FxH;F$u!!.v2d;
1CC0:0170  46 FC B9 00 00 75 01 41-A1 A8 F4 03 46 FC 8B 16  F|9..u.A!(t.F|..
```

The following dump shows a different part of COMMAND.COM. Because memory at this point contains a list of command names, the ASCII dump is more interesting:

```
-D 3AC0
1CD6:3AC0  05 45 58 49 53 54 EA 15-00 04 44 49 52 01 FA 09
.EXISTj...DIR.z.
1CD6:3AD0  07 52 45 4E 41 4D 45 01-B2 0C 04 52 45 4E 01 B2   .RENAME.2..REN.2
1CD6:3AE0  0C 06 45 52 41 53 45 01-3D 0C 04 44 45 4C 01 3D   ..ERASE.=..DEL.=
1CD6:3AF0  0C 05 54 59 50 45 01 EF-0C 04 52 45 4D 00 04 01   ..TYPE.o..REM...
1CD6:3B00  05 43 4F 50 59 01 CC 1A-06 50 41 55 53 45 00 1F   .COPY.L..PAUSE..
1CD6:3B10  13 05 44 41 54 45 00 38-18 05 54 49 4D 45 00 CE   ..DATE.8..TIME.N
1CD6:3B20  18 04 56 45 52 00 57 0E-04 56 4F 4C 01 C8 0D 03   ..VER.W..VOL.H..
1CD6:3B30  43 44 01 A6 12 06 43 48-44 49 52 01 A6 12 03 4D   CD.&..CHDIR.&..M
1CD6:3B40  44 01 D9 12 06 4D 4B 44-49 52 01 D9 12 03 52 44   D.Y..MKDIR.Y..RD
1CD6:3B50  01 0E 13 06 52 4D 44 49-52 01 0E 13 06 42 52 45   ....RMDIR....BRE
1CD6:3B60  41 4B 00 92 17 07 56 45-52 49 46 59 00 C7 17 04   AK....VERIFY.G..
1CD6:3B70  53 45 54 00 0F 10 07 50-52 4F 4D 50 54 00 FA 0F   SET....PROMPT.z.
1CD6:3B80  05 50 41 54 48 00 A0 0F-05 45 58 49 54 00 C9 11   .PATH. ..EXIT.I.
1CD6:3B90  05 43 54 54 59 01 F7 11-05 45 43 48 4F 00 59 17   .CTTY.w..ECHO.Y.
1CD6:3BA0  05 47 4F 54 4F 00 96 16-06 53 48 49 46 54 00 56   .GOTO....SHIFT.V
```

```
1CD6:3BB0  16 03 49 46 00 50 15 04-46 4F 52 00 68 14 04 43   ..IF.P..FOR.h..C
1CD6:3BC0  4C 53 00 53 12 00 00 00-00 00 00 00 00 00 00 00   LS.S............
```

3.5 E (Enter)

The E command places individual bytes in memory. You must supply a starting memory location where the values will be stored. If only an offset value is entered, the offset is assumed to be from DS. Otherwise, a 32-bit address may be entered or another segment register may be used. Command formats are:

`E` *`address`* — Enter new byte value at *address*.

`E` *`address list`* — Replace the contents of one or more bytes starting at the specified *address*, with the values contained in the *list*.

To begin entering hexadecimal or character data at DS:100, type:

```
E 100
```

Press the space bar to advance to the next byte, and press the Enter key to stop. To enter a string into memory starting at location CS:100, type:

```
E CS:100 "This is a string."
```

3.6 F (Fill)

The F command fills a range of memory with a single value or list of values. The range must be specified as two offset addresses or segment-offset addresses. Command format:

F ***range list***

Here are some examples. The commas are optional:

Example	Description
`F 100 500,' '`	Fill locations 100 through 500 with spaces.
`F CS:300 CS:1000,FF`	Fill locations CS:300 through 1000 with hex FFh.
`F 100 L 20 'A'`	Fill 20h bytes with the letter 'A', starting at location 100.

3.7 G (Go)

Execute the program in memory. You can also specify a breakpoint, causing the program to stop at a given address. Command formats:

G
G ***breakpoint***

```
G = startAddr breakpoint
G = startAddr breakpoint1 breakpoint2 ...
```

Breakpoint is a 16- or 32-bit address at which the processor should stop, and *startAddr* is an optional starting address for the processor. If no breakpoints are specified, the program runs until it stops by itself and returns to Debug. Up to 10 breakpoints may be specified on the same command line. Examples:

Example	**Description**
`G`	Execute from the current location to the end of the program.
`G 50`	Execute from the current location and stop before the instruction at offset CS:50.
`G=10 50`	Begin execution at CS:10 and stop before the instruction at offset CS:50.

3.8 H (Hexarithmetic)

The H command performs addition and subtraction on two hexadecimal numbers. The command format is:

```
H value1 value2
```

For example, the hexadecimal values 1A and 10 are added and subtracted:

```
H 1A 10
2A 0A                  (displayed by Debug)
```

3.9 I (Input)

The I command inputs a byte from a specified input/output port and displays the value in hexadecimal. The command format is:

```
I port
```

Where *port* is a port number between 0 and FFFF. For example, we input a byte from port 3F8 (one of the COM1 ports), and Debug returns a value of 00:

```
-I 3F8
00
```

3.10 L (Load)

The L command loads a file (or logical disk sectors) into memory at a given address. To read a file, you must first initialize its name with the N (Name) command. If *address* is

Table 1. Examples of the Load Instruction.

Example	Description
L	Load named file into memory at CS:0100
L DS:0200	Load named file into memory at DS:0200
L 100 2 A 5	Load five sectors from drive C, starting at logical sector number 0Ah.
L 100 0 0 2	Load two sectors into memory at CS:100, from the disk in drive A, starting at logical sector number 0.

omitted, the file is loaded at CS:100. Debug sets BX and CX to the number of bytes read. Command format:

```
L
L address
L address drive firstsector number
```

The first format, with no parameters, implies that you want to read from a file into memory at CS:0100. (Use the N command to name the file.) The second format also reads from a named file, but lets you specify the target address. The third format loads sectors from a disk drive, where you specify the drive number (0 = A, 1 = B, etc.), the first logical sector number, and the number of sectors to read. Examples are shown in Table 1.

Each sector is 512 bytes, so a sector loaded at offset 100 would fill memory through offset 2FF. Logical sectors are numbered from 0 to the highest sector number on the drive. These numbers are different from *physical* sector numbers, which are hardware-dependent. To calculate the number of logical sectors, take the drive size and divide by 512. For example, a 1.44 MB diskette has 2,880 sectors, calculated as 1,474,560 / 512.

Here is a disassembly of sector 0 read from a floppy disk, using Debug. This is commonly called the *boot record*. The boot record contains information about the disk, along with a short program that is responsible for loading the rest of the operating system when the computer starts up:

```
1F6E:0100 EB34          JMP     0136
...
1F6E:0136 FA            CLI
1F6E:0137 33C0          XOR     AX,AX
1F6E:0139 8ED0          MOV     SS,AX
1F6E:013B BC007C        MOV     SP,7C00
```

3.11 M (Move)

The M command copies a block of data from one memory location to another. The command format is:

M *range address*

Range consists of the starting and ending locations of the bytes to be copied. *Address* is the target location to which the data will be copied. All offsets are assumed to be from DS unless specified otherwise. Examples:

Example	Description
`M 100 105 110`	Move bytes in the range DS:100-105 to location DS:110.
`M CS:100 105 CS:110`	Same as above, except that all offsets are relative to the segment value in CS.

Sample String Move. The following example uses the M command to copy the string 'ABCDEF' from offset 100h to 106h. First, the string is stored at location 100h; then memory is dumped, showing the string. Next, we move (copy) the string to offset 106h and dump offsets 100h-10Bh:

```
-E 100 "ABCDEF"
-D 100 105
19EB:0100  41 42 43 44 45 46                      ABCDEF
-M 100 105 106
-D 100 10B
19EB:0100  41 42 43 44 45 46 41 42-43 44 45 46    ABCDEFABCDEF
```

3.12 N (Name)

The N command initializes a filename (and file control block) in memory before using the Load or Write commands. Comand format:

N [*d*:][*filename*][.*ext*]

Example:

```
N b:myfile.dta
```

3.13 P (Proceed)

The P command executes one or more instructions or subroutines. Whereas the T (trace) command traces into subroutine calls, the P command simply executes subroutines. Also, LOOP instruction and string primitive instructions (SCAS, LODS, etc.) are executed completely up to the instruction that follows them. Command format:

```
P
P =address
P =address number
```

Examples are:

Example	Description
P =200	Execute a single instruction at CS:0200.
P =150 6	Execute 6 instructions starting at CS:0150.
P 5	Execute the next 5 instructions.

Example: Debugging a Loop. Let's look at an example where the P command steps through MOV and ADD instructions one at a time. When the P command reaches the LOOP instruction, however, the complete loop is executed five times:

```
-A 100
4A66:0100 mov cx,5      ; loop counter = 5
4A66:0103 mov ax,0
4A66:0106 add ax,cx
4A66:0108 loop 106      ; loop to location 0106h

-R
AX=000F  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=4A66  ES=4A66  SS=4A66  CS=4A66  IP=0100   NV UP EI PL NZ NA PE NC
4A66:0100 B90500        MOV     CX,0005
-P
AX=000F  BX=0000  CX=0005  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=4A66  ES=4A66  SS=4A66  CS=4A66  IP=0103   NV UP EI PL NZ NA PE NC
4A66:0103 B80000        MOV     AX,0000
-P
AX=0000  BX=0000  CX=0005  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=4A66  ES=4A66  SS=4A66  CS=4A66  IP=0106   NV UP EI PL NZ NA PE NC
4A66:0106 01C8          ADD     AX,CX
-P
AX=0005  BX=0000  CX=0005  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=4A66  ES=4A66  SS=4A66  CS=4A66  IP=0108   NV UP EI PL NZ NA PE NC
4A66:0108 E2FC          LOOP    0106
-P
AX=000F  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=4A66  ES=4A66  SS=4A66  CS=4A66  IP=010A   NV UP EI PL NZ NA PE NC
```

3.14 Q (Quit)

The Q command quits Debug and returns to DOS.

3.15 R (Register)

The R command may be used to do any of the following: display the contents of one register, allowing it to be changed; display registers, flags, and the next instruction about to be executed; display all eight flag settings, allowing any or all of them to be changed. There are two command formats:

```
R
R register
```

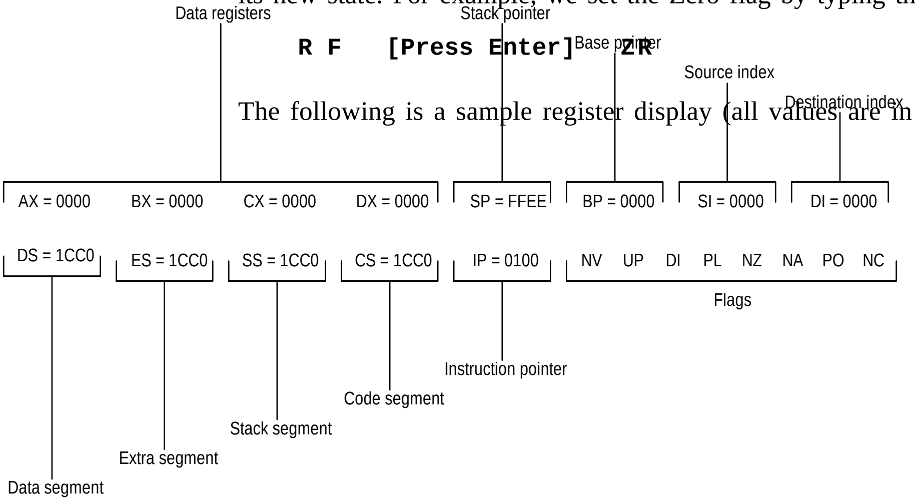
Here are some examples:

Example	**Description**
`R`	Display the contents of all registers.
`R IP`	Display the contents of IP and prompt for a new value.
`R CX`	Same (for the CX register).
`R F`	Display all flags and prompt for a new flag value.

Once the **R F** command has displayed the flags, you can change any single flag by typing its new state. For example, we set the Zero flag by typing the following two commands:

```
R F   [Press Enter]   ZR
```

The following is a sample register display (all values are in hexadecimal):

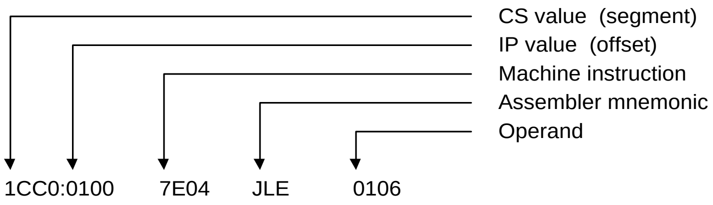


```
AX = 0000   BX = 0000   CX = 0000   DX = 0000   SP = FFEE   BP = 0000   SI = 0000   DI = 0000
DS = 1CC0   ES = 1CC0   SS = 1CC0   CS = 1CC0   IP = 0100   NV  UP  DI  PL  NZ  NA  PO  NC
```

The complete set of possible flag mnemonics in Debug (ordered from left to right) are as follows:

Set	***Clear***
OV = Overflow	NV = No Overflow
DN = Direction Down	UP = Direction Up
EI = Interrupts Enabled	DI = Interrupts Disabled
NG = Sign Flag negative	PL = Sign Flag positive
ZR = Zero	NZ = Not Zero
AC = Auxiliary Carry	NA = No Auxiliary Carry
PO = Odd Parity	PE = Even Parity
CY = Carry	NC = No Carry

The **R** command also displays the next instruction to be executed:



3.16 S (Search)

The S command searches a range of addresses for a sequence of one or more bytes. The command format is:

```
S range list
```

Here are some examples:

Example	**Comment**
`S 100 1000 0D`	Search DS:100 to DS:1000 for the value 0Dh.
`S 100 1000 CD,20`	Search for the sequence CD 20.
`S 100 9FFF "COPY"`	Search for the word "COPY".

3.17 T (Trace)

The T command executes one or more instructions starting at either the current CS:IP location or at an optional address. The contents of the registers are shown after each instruction is executed. The command formats are:

`T`
`T` ***`count`***
`T` ***`=address count`***

Where *count* is the number of instructions to trace, and *address* is the starting address for the trace. Examples:

Example	**Description**
`T`	Trace the next instruction.
`T 5`	Trace the next five instructions.
`T =105 10`	Trace 16 instructions starting at CS:105.

This command traces individual loop iterations, so you may want to use it to debug statements within a loop. The T command traces into procedure calls, whereas the P (*proceed*) command executes a called procedure in its entirety.

3.18 U (Unassemble)

The U command translates memory into assembly language mnemonics. This is also called *disassembling* memory. If you don't supply an address, Debug disassembles from the location where the last U command left off. If the command is used for the first time after loading Debug, memory is unassembled from location CS:100. Command formats are:

`U`
`U` ***`startaddr`***
`U` ***`startaddr endaddr`***

Where *startaddr* is the starting point and *endaddr* is the ending address. Examples are:

Example	**Description**
`U`	Disassemble the next 32 bytes.
`U 0`	Disassemble 32 bytes at CS:0.
`U 100 108`	Disassemble the bytes from CS:100 to CS:108.

3.19 W (Write)

The W command writes a block of memory to a file or to individual disk sectors. To write to a file, its name must first be initialized with the N command. (If the file was just loaded either on the DOS command line or with the Load command, you do not need to repeat the Name command.) The command format is identical to the L (load) command:

```
W
W address
W address drive firstsector number
```

Place the number of bytes to be written in BX:CX. If a file is 12345h bytes long, for example, BX and CX will contain the following values:

```
BX = 0001   CX = 2345
```

Here are a few examples:

Example	**Description**
N EXAMPLE.COM	Initialize the filename EXAMPLE.COM on the default drive.
R BX 0 R CX 20	Set the BX and CX registers to 00000020h, the length of the file.
W	Write 20h bytes to the file, starting at CS:100.
W 0	Write from location CS:0 to the file.
W	Write named file from location CS:0100.
W DS:0200	Write named file from location DS:0200.

The following commands are extremely dangerous to the data on your disk drive, because writing sectors can wipe out the disk's existing file system. Use them with extreme caution!

`W 100 2 A 5`	Write five sectors to drive C from location CS:100, starting at logical sector 0Ah.
`W 100 0 0 2`	Write two sectors to drive A from location CS:100, starting at logical sector number 0.

Table 2. Default Segments for Debug Commands.

Command	Description	Default Segment
A	Assemble	CS
D	Dump	DS
E	Enter	DS
F	Fill	DS
G	Go (execute)	CS
L	Load	CS
M	Move	DS
P	Procedure trace	CS
S	Search	DS
T	Trace	CS
U	Unassemble	CS
W	Write	CS

4 SEGMENT DEFAULTS

Debug recognizes the CS and DS registers as default segment registers when processing commands. Therefore, the value in CS or DS acts as a base location to which an offset value is added. Table 2 lists the default segment register for selected Debug commands.

The Assemble command, for example, assumes that CS is the default segment. If CS contains 1B00h, Debug begins assembling instructions at location 1B00:0100h when the following command is typed:

```
-A 100
```

The Dump command, on the other hand, assumes that DS is the default segment. If DS contains 1C20h, the following command dumps memory starting at 1C20:0300h:

```
-D 300
```

5 USING SCRIPT FILES WITH DEBUG

A major disadvantage of using Debug to assemble and run programs shows up when the programs must be edited. Inserting a new instruction often means retyping all subsequent instructions and recalculating the addresses of memory variables. There is an easier way: All of the commands and instructions may first be placed a text file, which we will call a script file. When Debug is run from DOS, you can use a redirection symbol (<) to tell it to read input from the *script file* instead of the console. For example, assume that a script file called input.txt contains the following lines:

```
a 100
mov ax,5
mov bx,10
add ax,bx
int 20
(blank line)
Q
```

(Always remember to put a Q on a line by itself at the end of your script file. The Q command returns to the DOS prompt.)

Debug can be executed, using the script file as input:

```
debug < input.txt
```

If you are running in a task-switching environment such as Windows, you can edit and save the script file with the Notepad editor or DOS Edit (in a separate window). Then switch back to a window in which you are running Debug. In this way, a program may be modified, saved, assembled, and traced within a few seconds. If you would like the output to be sent to a disk file or the printer, use redirection operators on the DOS command line:

```
debug < input.txt > prn              (printer)
debug < input.txt > output.txt       (disk file)
```

Name:	Pankhania Aanandi R.
Roll No:	IT081
Batch:	I1

Experiment 2

1. Write an assembly language program for subtraction of two 16-bit numbers.

Rules for Operands: 1st number=your roll no. of sem-5, 2nd number=reverse of your roll no.

e.g. roll no.=IT108 so, 16-bit 1st number = 0108h, 2nd number=8010h.

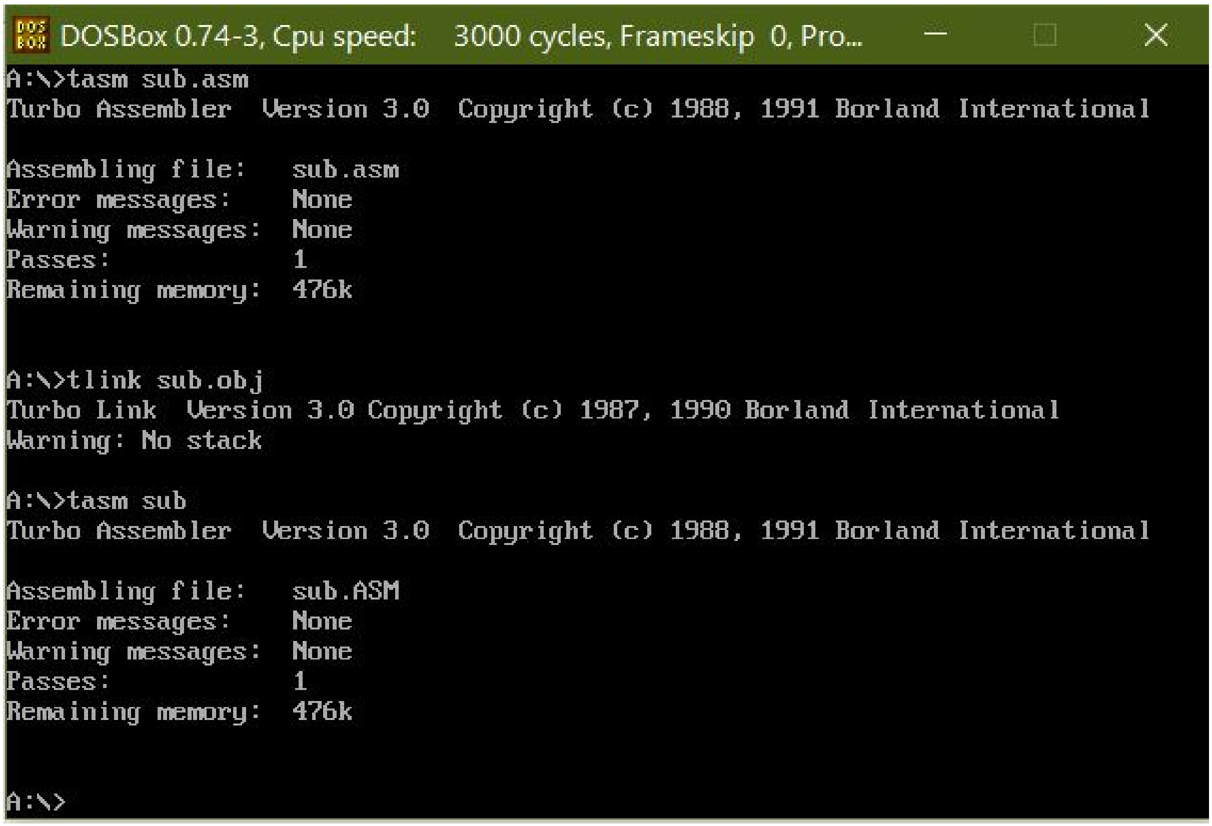
e.g. for repeater student ID=18ITUOS103, 1st number=1803h, 2nd number=3081h.

Write your code here:

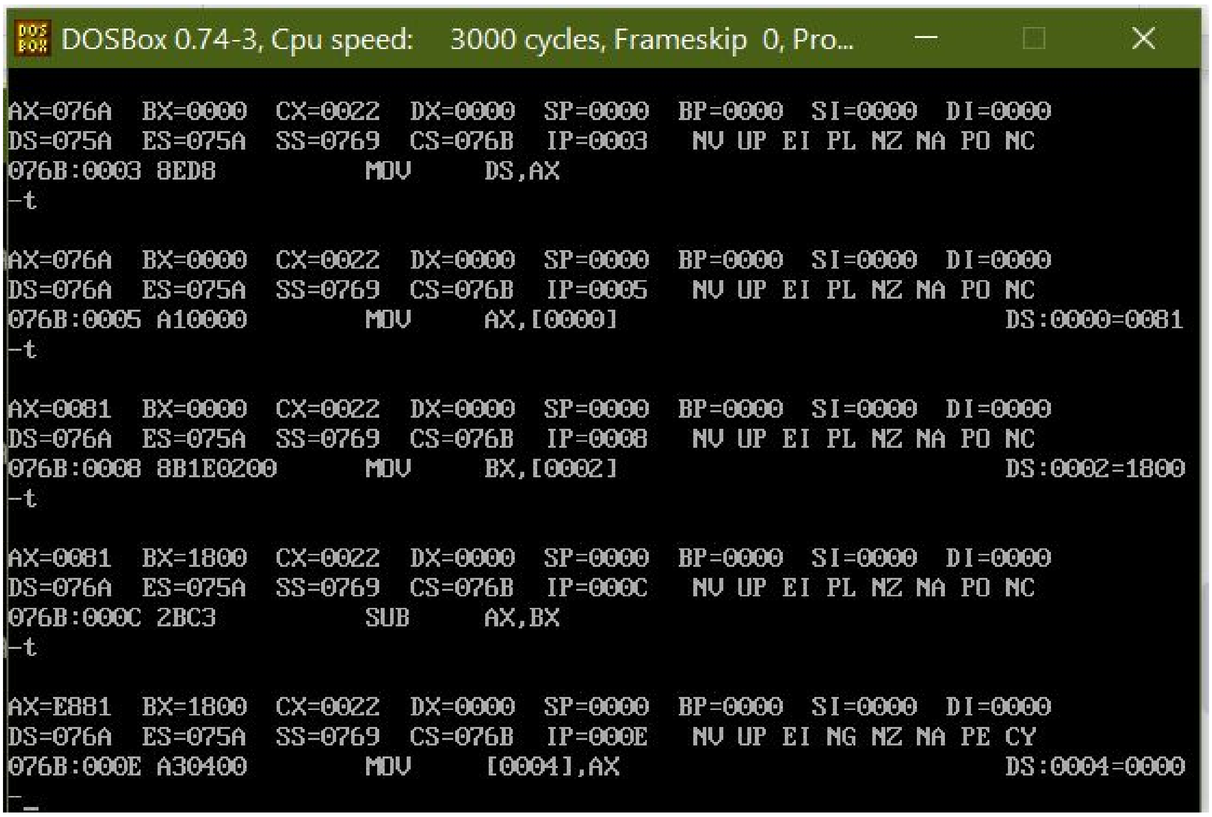
```
data_here segment
                a dw 0081h
                b dw 1800h
                c dw ?
data_here ends

code_here segment
                assume cs:code_here,ds:data_here
                start:
                        mov ax,data_here
                        mov ds,ax
                        mov ax,a   ;ax=0081h   ; 129d
                        mov bx,b   ;bx=1800h   ;6144d
                        sub  ax,bx      ;ax=ax-bx      ;129d-6144d   (0081h-1800h)=   -6015d(2's
complement of E881h)
                        mov c,ax    ;c=ax
                        int 3h

code_here ends
end start
```

Compilation /Running and Debugging steps:

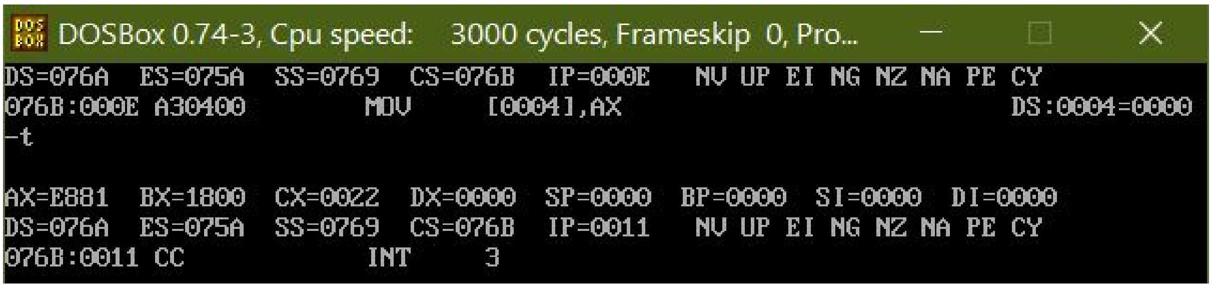
(As given in lab manual as an example of multiplication program on page no:5 of lab manual)


DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Pro...

```
A:\>tasm sub.asm
Turbo Assembler  Version 3.0  Copyright (c) 1988, 1991 Borland International

Assembling file:   sub.asm
Error messages:    None
Warning messages:  None
Passes:            1
Remaining memory:  476k


A:\>tlink sub.obj
Turbo Link  Version 3.0 Copyright (c) 1987, 1990 Borland International
Warning: No stack

A:\>tasm sub
Turbo Assembler  Version 3.0  Copyright (c) 1988, 1991 Borland International

Assembling file:   sub.ASM
Error messages:    None
Warning messages:  None
Passes:            1
Remaining memory:  476k


A:\>
```


DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Pro...

```
AX=076A  BX=0000  CX=0022  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=075A  ES=075A  SS=0769  CS=076B  IP=0003   NV UP EI PL NZ NA PO NC
076B:0003 8ED8          MOV     DS,AX
-t

AX=076A  BX=0000  CX=0022  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=076A  ES=075A  SS=0769  CS=076B  IP=0005   NV UP EI PL NZ NA PO NC
076B:0005 A10000        MOV     AX,[0000]                          DS:0000=0081
-t

AX=0081  BX=0000  CX=0022  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=076A  ES=075A  SS=0769  CS=076B  IP=0008   NV UP EI PL NZ NA PO NC
076B:0008 8B1E0200      MOV     BX,[0002]                          DS:0002=1800
-t

AX=0081  BX=1800  CX=0022  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=076A  ES=075A  SS=0769  CS=076B  IP=000C   NV UP EI PL NZ NA PO NC
076B:000C 2BC3          SUB     AX,BX
-t

AX=E881  BX=1800  CX=0022  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=076A  ES=075A  SS=0769  CS=076B  IP=000E   NV UP EI NG NZ NA PE CY
076B:000E A30400        MOV     [0004],AX                          DS:0004=0000
-_
```

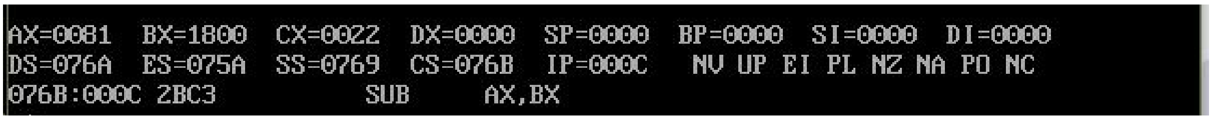

DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Pro...

```
DS=076A  ES=075A  SS=0769  CS=076B  IP=000E   NV UP EI NG NZ NA PE CY
076B:000E A30400        MOV     [0004],AX                        DS:0004=0000
-t

AX=E881  BX=1800  CX=0022  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=076A  ES=075A  SS=0769  CS=076B  IP=0011   NV UP EI NG NZ NA PE CY
076B:0011 CC            INT     3
```

Output:

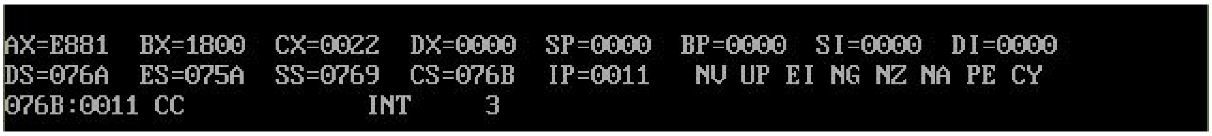
Screenshots of internal register contents before execution and after execution.

Before execution:



```
AX=0081  BX=1800  CX=0022  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=076A  ES=075A  SS=0769  CS=076B  IP=000C   NV UP EI PL NZ NA PO NC
076B:000C 2BC3          SUB     AX,BX
```

After execution:



```
AX=E881  BX=1800  CX=0022  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=076A  ES=075A  SS=0769  CS=076B  IP=0011   NV UP EI NG NZ NA PE CY
076B:0011 CC            INT     3
```

2. **Write an assembly language program to perform scalar multiplication of an array of five unsigned bytes.**
   Rules for Operands: Array elements are 10h,15h,20h,25h,30h (you can take any values). Multiply each element by the last digit of your roll no./ (repeater student – student id) i.e. IT067 so, multiply array elements by 7 and store result in another array.

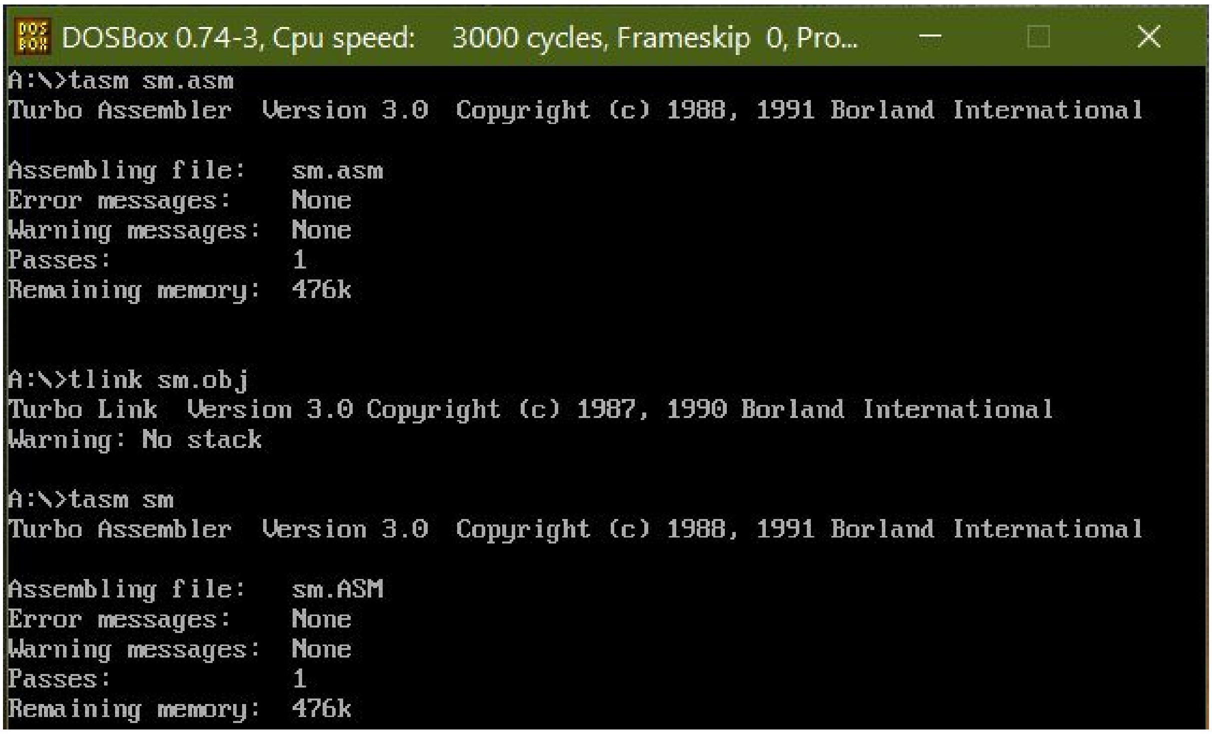
Write your code here:

```
data_here segment
            arr db 1h,2h,3h,14h,25h  ;created an array
            ld db 1h  ;roll no.'s(IT081) last digit---1
            ar dw 5 dup(?)  ;another array
data_here ends
code_here segment
            assume cs:code_here,ds:data_here
            start:   mov ax,data_here
                     mov ds,ax
```

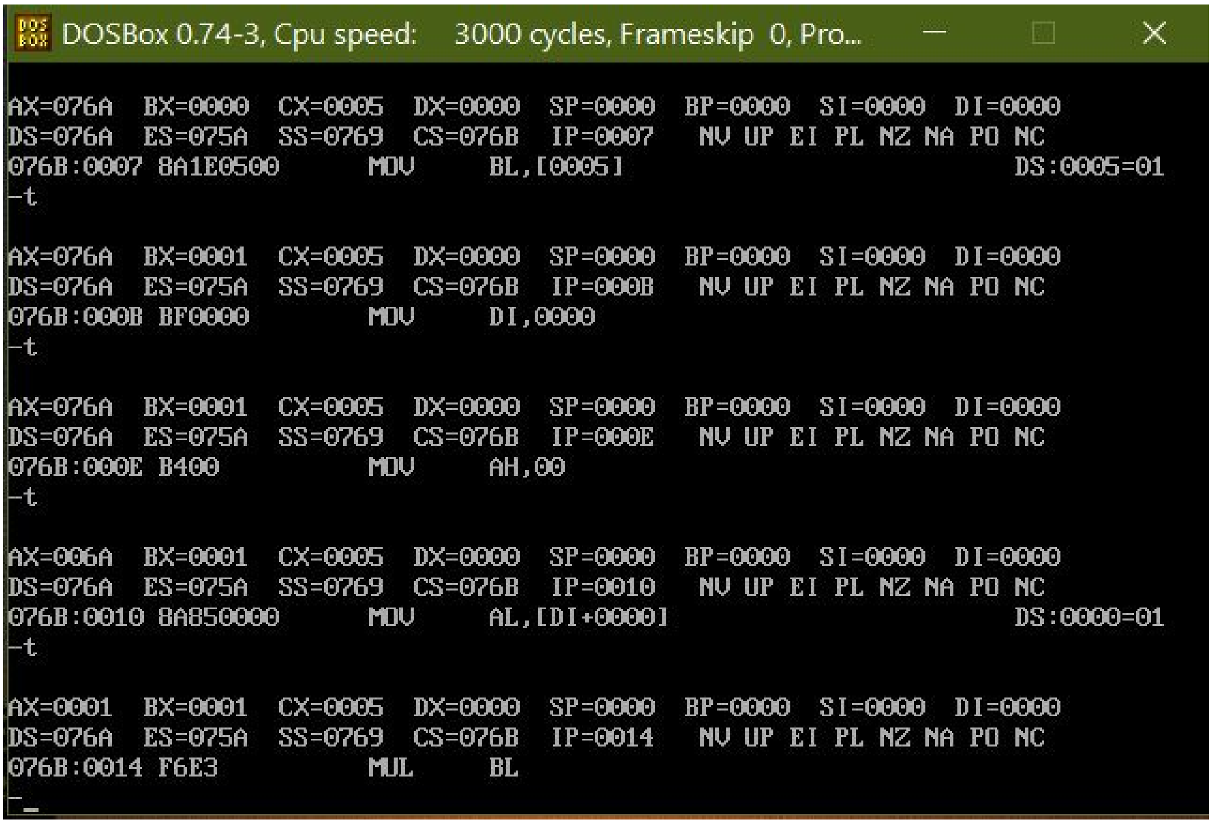
```
                mov cl, 5   ;count value 5
                mov bl, ld
                mov DI, 0
                mov ah,0
        l1: mov al, arr[DI] ;loop l1 start
                mul bl
                mov ar[DI], al
                inc DI
                dec cl    ;count--
                jnz l1    ;when count 0 loop will end...
                int 21h
                int 3h
code_here ends
end start
```

Compilation /Running and Debugging steps:

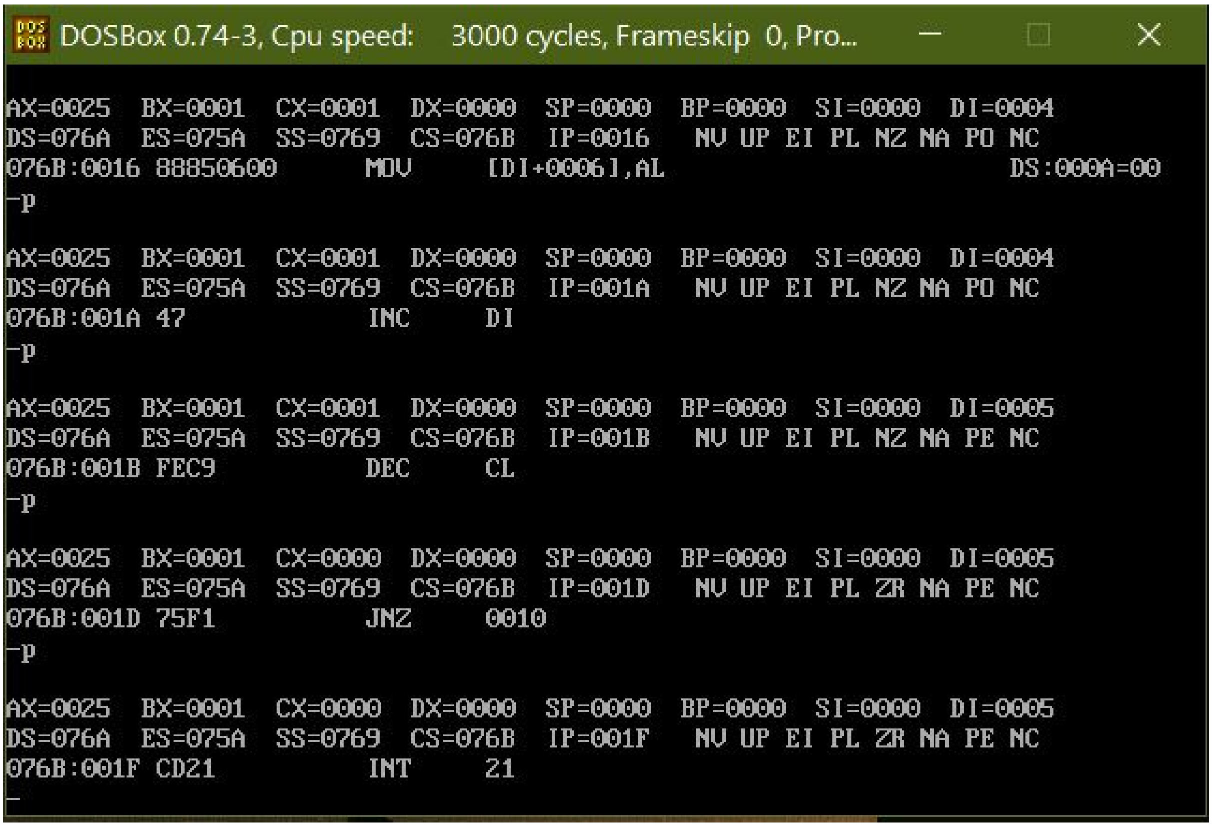
(As given in the lab manual as an example of multiplication program on page no:5 of lab manual)


DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Pro...

```
A:\>tasm sm.asm
Turbo Assembler  Version 3.0  Copyright (c) 1988, 1991 Borland International

Assembling file:   sm.asm
Error messages:    None
Warning messages:  None
Passes:            1
Remaining memory:  476k


A:\>tlink sm.obj
Turbo Link  Version 3.0 Copyright (c) 1987, 1990 Borland International
Warning: No stack

A:\>tasm sm
Turbo Assembler  Version 3.0  Copyright (c) 1988, 1991 Borland International

Assembling file:   sm.ASM
Error messages:    None
Warning messages:  None
Passes:            1
Remaining memory:  476k
```


DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Pro...

```
AX=076A  BX=0000  CX=0005  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=076A  ES=075A  SS=0769  CS=076B  IP=0007   NV UP EI PL NZ NA PO NC
076B:0007 8A1E0500      MOV     BL,[0005]                          DS:0005=01
-t

AX=076A  BX=0001  CX=0005  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=076A  ES=075A  SS=0769  CS=076B  IP=000B   NV UP EI PL NZ NA PO NC
076B:000B BF0000        MOV     DI,0000
-t

AX=076A  BX=0001  CX=0005  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=076A  ES=075A  SS=0769  CS=076B  IP=000E   NV UP EI PL NZ NA PO NC
076B:000E B400          MOV     AH,00
-t

AX=006A  BX=0001  CX=0005  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=076A  ES=075A  SS=0769  CS=076B  IP=0010   NV UP EI PL NZ NA PO NC
076B:0010 8A850000      MOV     AL,[DI+0000]                       DS:0000=01
-t

AX=0001  BX=0001  CX=0005  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=076A  ES=075A  SS=0769  CS=076B  IP=0014   NV UP EI PL NZ NA PO NC
076B:0014 F6E3          MUL     BL
-_
```

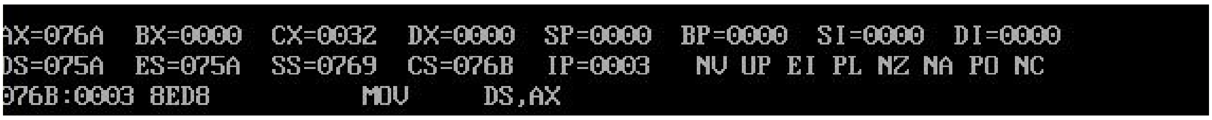

DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Pro...

```
AX=0025  BX=0001  CX=0001  DX=0000  SP=0000  BP=0000  SI=0000  DI=0004
DS=076A  ES=075A  SS=0769  CS=076B  IP=0016   NV UP EI PL NZ NA PO NC
076B:0016 88850600      MOV     [DI+0006],AL                       DS:000A=00
-p

AX=0025  BX=0001  CX=0001  DX=0000  SP=0000  BP=0000  SI=0000  DI=0004
DS=076A  ES=075A  SS=0769  CS=076B  IP=001A   NV UP EI PL NZ NA PO NC
076B:001A 47            INC     DI
-p

AX=0025  BX=0001  CX=0001  DX=0000  SP=0000  BP=0000  SI=0000  DI=0005
DS=076A  ES=075A  SS=0769  CS=076B  IP=001B   NV UP EI PL NZ NA PE NC
076B:001B FEC9          DEC     CL
-p

AX=0025  BX=0001  CX=0000  DX=0000  SP=0000  BP=0000  SI=0000  DI=0005
DS=076A  ES=075A  SS=0769  CS=076B  IP=001D   NV UP EI PL ZR NA PE NC
076B:001D 75F1          JNZ     0010
-p

AX=0025  BX=0001  CX=0000  DX=0000  SP=0000  BP=0000  SI=0000  DI=0005
DS=076A  ES=075A  SS=0769  CS=076B  IP=001F   NV UP EI PL ZR NA PE NC
076B:001F CD21          INT     21
-
```

Output:

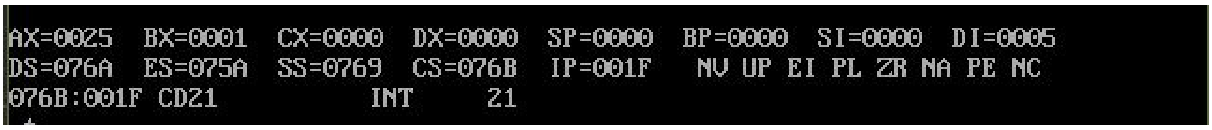
Screenshots of internal register contents before execution and after execution.

Before Execution:



```
AX=076A  BX=0000  CX=0032  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=075A  ES=075A  SS=0769  CS=076B  IP=0003   NV UP EI PL NZ NA PO NC
076B:0003 8ED8          MOV     DS,AX
```

After Execution:



```
AX=0025  BX=0001  CX=0000  DX=0000  SP=0000  BP=0000  SI=0000  DI=0005
DS=076A  ES=075A  SS=0769  CS=076B  IP=001F   NV UP EI PL ZR NA PE NC
076B:001F CD21          INT     21
```

Name:	Pankhania Aanandi R.
Roll No:	IT081
Batch	I1

Experiment 3

AIM: Study of string related instructions

1. **Write an assembly language program for moving a string from one segment to another segment.**

Rules for Operands: You have to use your name as a string name.

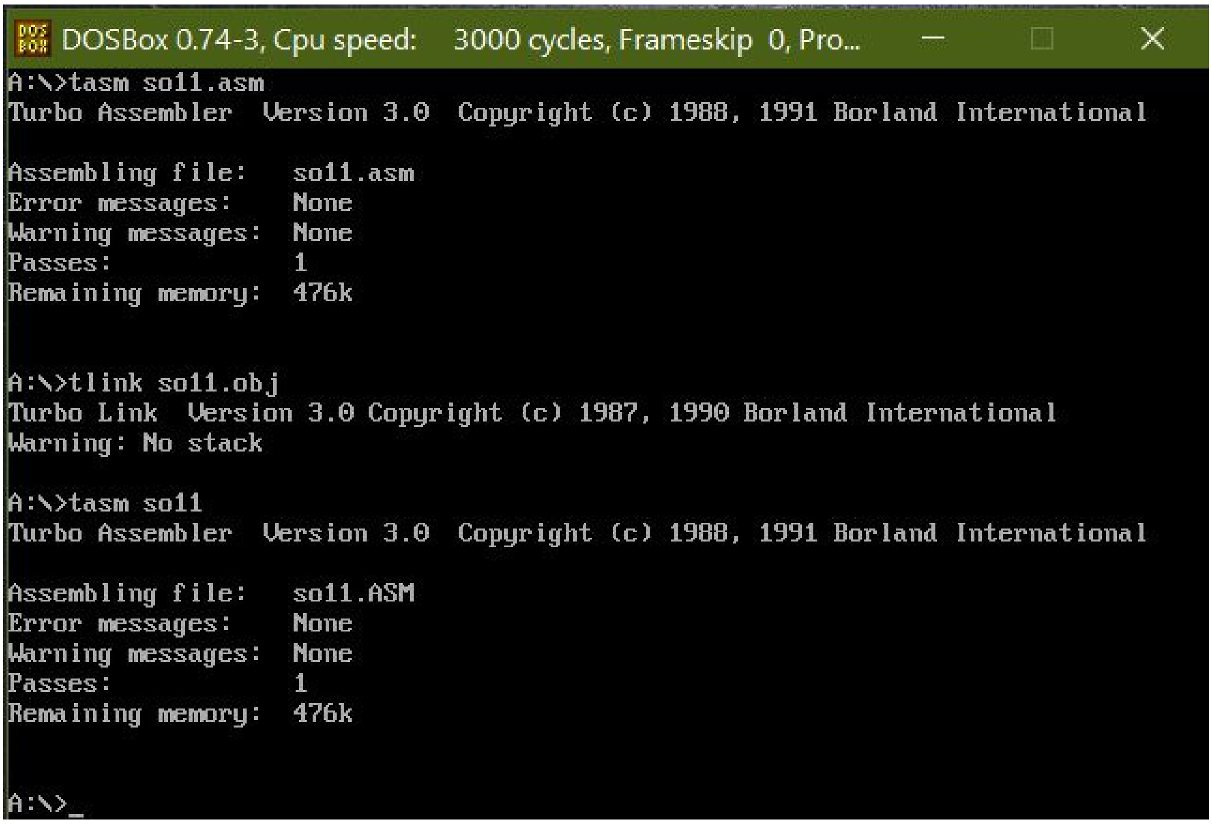
e.g. myname DB “Sunil K. Vithlani$”

Write your code here:

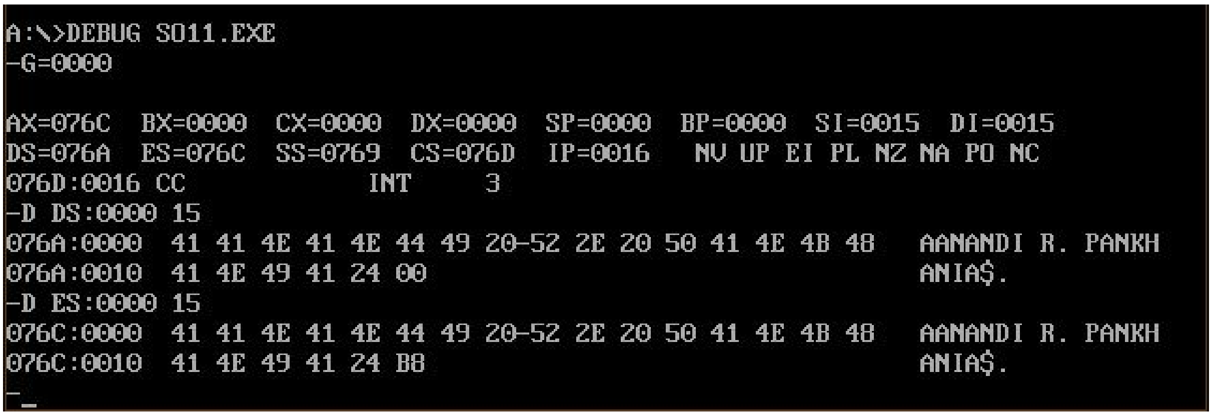
```
DATA SEGMENT
        myname db 'AANANDI R. PANKHANIA$'
        len equ $-myname
DATA ENDS
DATA1 SEGMENT
        str1 db 15 DUP(0)
DATA1 ENDS

CODE SEGMENT
assume CS:CODE ,DS:DATA , ES:DATA1
   Start : mov AX,DATA
        mov DS,AX
        mov AX,DATA1
        mov ES, AX
        LEA SI,myname
        LEA DI,str1
        mov CX,len
        CLD
        REP movsb
        INT 3h
CODE ENDS
END Start
```

Compilation /Running and Debugging steps:

(As given in the lab manual as an example of multiplication program on page no:5 of lab manual)


DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Pro...

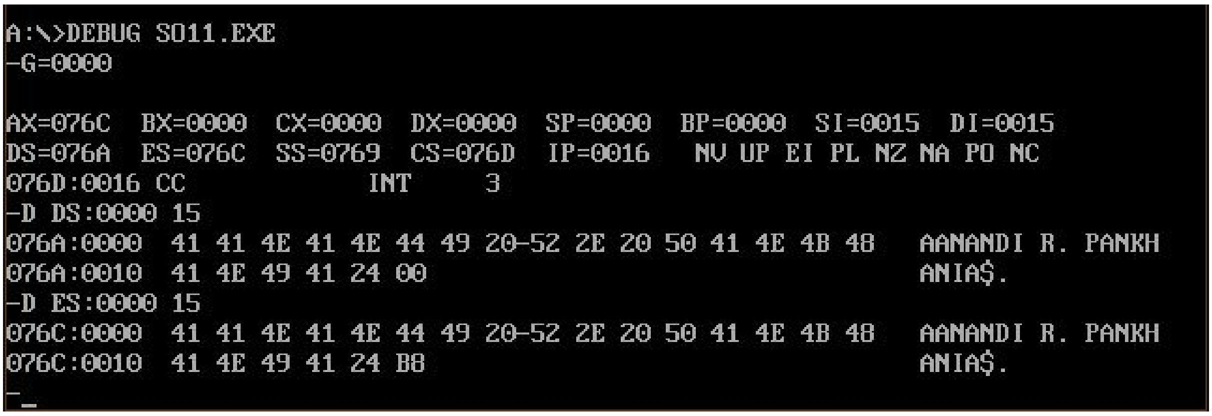
```
A:\>tasm so11.asm
Turbo Assembler  Version 3.0  Copyright (c) 1988, 1991 Borland International

Assembling file:   so11.asm
Error messages:    None
Warning messages:  None
Passes:            1
Remaining memory:  476k


A:\>tlink so11.obj
Turbo Link  Version 3.0 Copyright (c) 1987, 1990 Borland International
Warning: No stack

A:\>tasm so11
Turbo Assembler  Version 3.0  Copyright (c) 1988, 1991 Borland International

Assembling file:   so11.ASM
Error messages:    None
Warning messages:  None
Passes:            1
Remaining memory:  476k


A:\>_
```



```
A:\>DEBUG SO11.EXE
-G=0000

AX=076C  BX=0000  CX=0000  DX=0000  SP=0000  BP=0000  SI=0015  DI=0015
DS=076A  ES=076C  SS=0769  CS=076D  IP=0016   NV UP EI PL NZ NA PO NC
076D:0016 CC            INT     3
-D DS:0000 15
076A:0000  41 41 4E 41 4E 44 49 20-52 2E 20 50 41 4E 4B 48   AANANDI R. PANKH
076A:0010  41 4E 49 41 24 00                                 ANIA$.
-D ES:0000 15
076C:0000  41 41 4E 41 4E 44 49 20-52 2E 20 50 41 4E 4B 48   AANANDI R. PANKH
076C:0010  41 4E 49 41 24 B8                                 ANIA$.
-_
```

Output:

Screenshots of memory location before moving and after moving a string. (output of -d ds:offset_addres command.)



```
A:\>DEBUG SO11.EXE
-G=0000

AX=076C  BX=0000  CX=0000  DX=0000  SP=0000  BP=0000  SI=0015  DI=0015
DS=076A  ES=076C  SS=0769  CS=076D  IP=0016   NV UP EI PL NZ NA PO NC
076D:0016 CC            INT     3
-D DS:0000 15
076A:0000  41 41 4E 41 4E 44 49 20-52 2E 20 50 41 4E 4B 48   AANANDI R. PANKH
076A:0010  41 4E 49 41 24 00                                 ANIA$.
-D ES:0000 15
076C:0000  41 41 4E 41 4E 44 49 20-52 2E 20 50 41 4E 4B 48   AANANDI R. PANKH
076C:0010  41 4E 49 41 24 B8                                 ANIA$.
-_
```

2. **Write an assembly language program to compare two strings of equal length.**
   Rules for Operands:
   Case1: Take your name as both string and show results.
   E.g   str1 DB “sunil”   and   str2 DB “sunil”
   Case2: Take your name as a upper case in 1st string and as a lower case in 2nd string.
   E.g. str1 DB “SUNIL” and str2 DB “sunil”

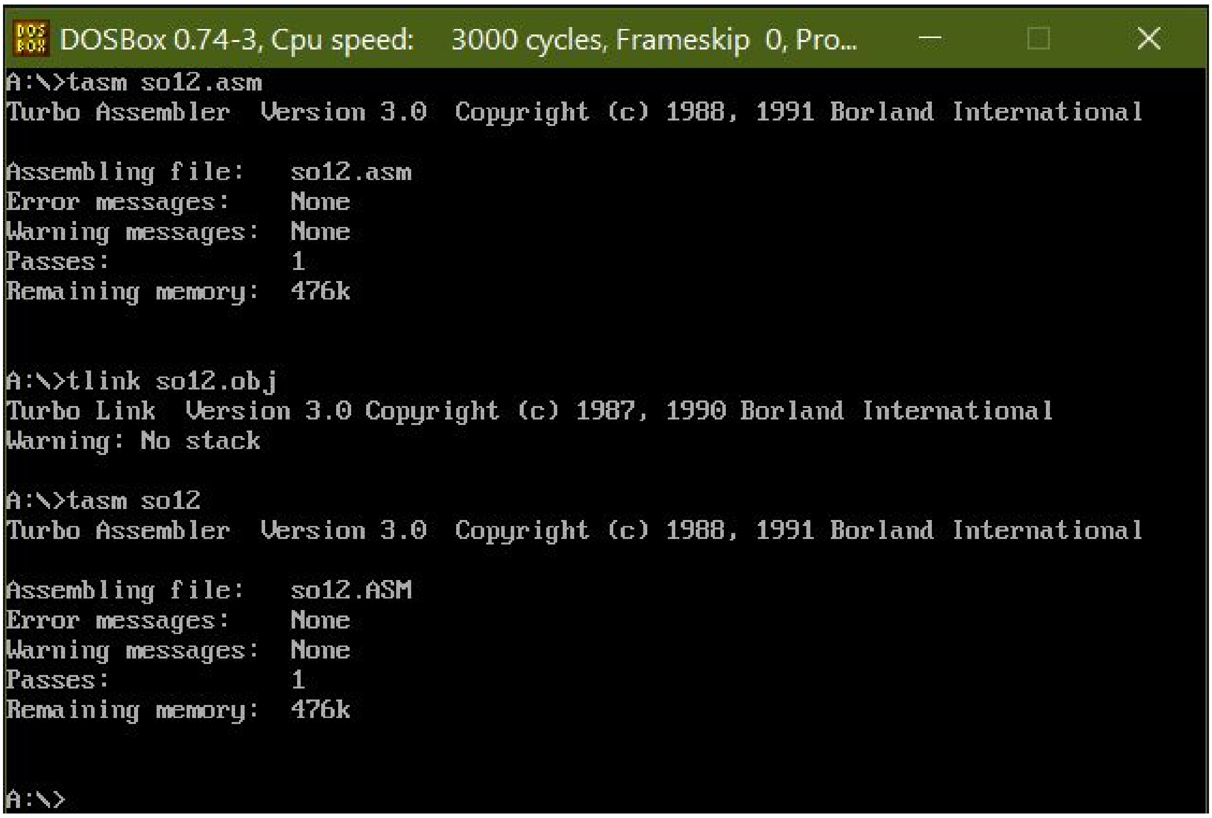
Write your code here:

```
DATA SEGMENT

  DEMO DB 'aanandi$

  STRNG DB 'aanandi$'

  msg1 DB 'strings are same$'

  msg2 DB 'strings are not same$'

DATA ENDS


CODE SEGMENT

assume CS:CODE,DS:DATA,ES:DATA


start:mov AX,DATA

    mov DS,AX
```

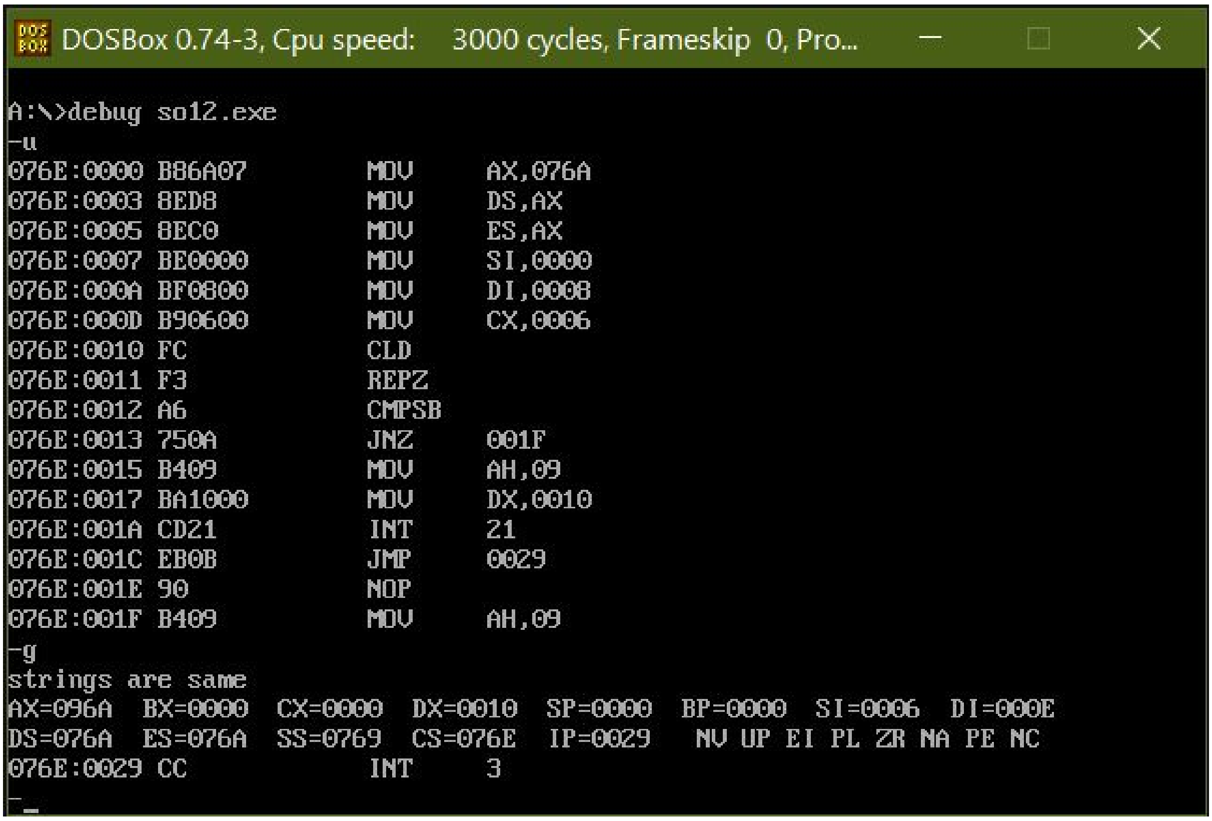
```
   mov ES,AX
   LEA SI,DEMO
   LEA DI,STRNG
   MOV CX,6
   CLD
   REPE CMPSB
   jnz msg22

  msg11:
  mov AH,09H
  mov DX,OFFSET msg1
  int 21h
  jmp exit

  msg22:
  mov AH,09H
  mov DX,OFFSET msg2
  int 21h
  jmp exit

  exit:int 3

CODE ENDS
END START
```

Compilation /Running and Debugging steps:

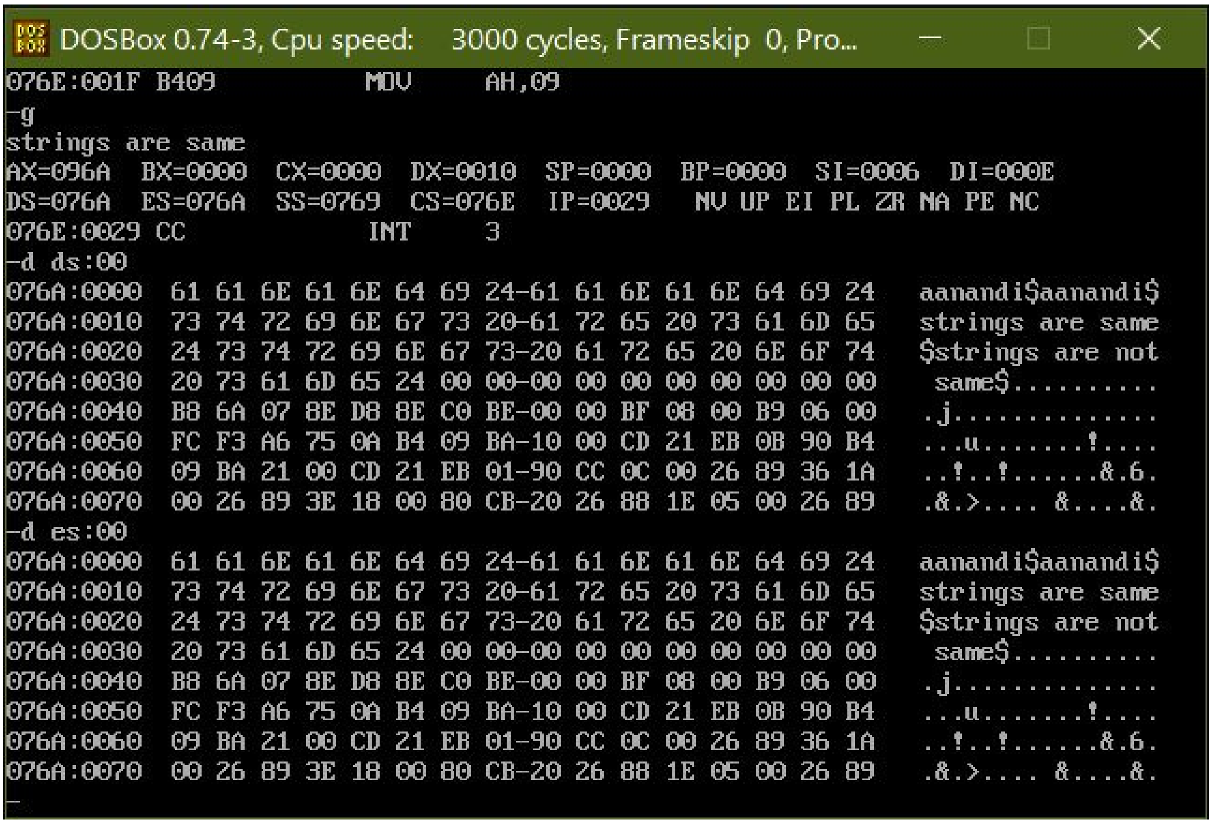
(As given in lab manual as an example of multiplication program on page no:5 of lab manual)



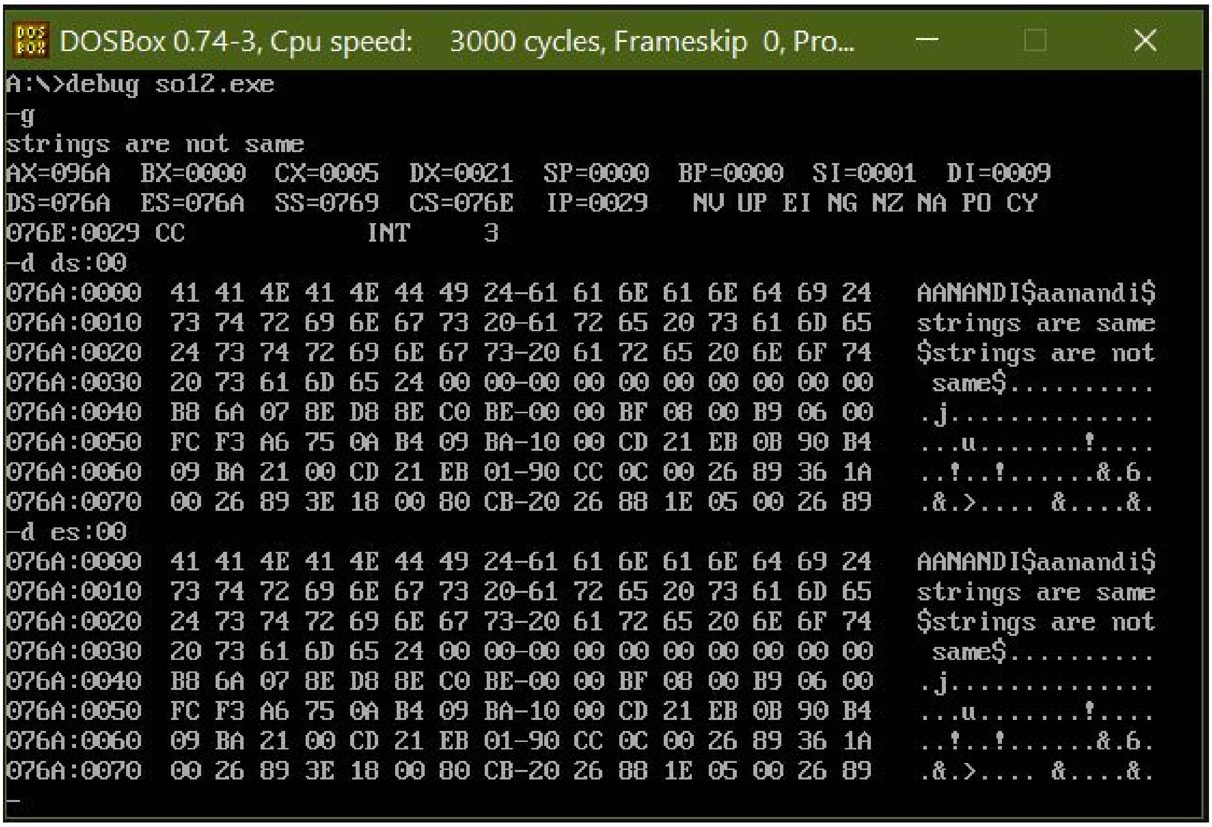
```
A:\>tasm so12.asm
Turbo Assembler  Version 3.0  Copyright (c) 1988, 1991 Borland International

Assembling file:   so12.asm
Error messages:    None
Warning messages:  None
Passes:            1
Remaining memory:  476k


A:\>tlink so12.obj
Turbo Link  Version 3.0 Copyright (c) 1987, 1990 Borland International
Warning: No stack

A:\>tasm so12
Turbo Assembler  Version 3.0  Copyright (c) 1988, 1991 Borland International

Assembling file:   so12.ASM
Error messages:    None
Warning messages:  None
Passes:            1
Remaining memory:  476k


A:\>
```



```

A:\>debug so12.exe
-u
076E:0000 B86A07        MOV     AX,076A
076E:0003 8ED8          MOV     DS,AX
076E:0005 8EC0          MOV     ES,AX
076E:0007 BE0000        MOV     SI,0000
076E:000A BF0800        MOV     DI,0008
076E:000D B90600        MOV     CX,0006
076E:0010 FC            CLD
076E:0011 F3            REPZ
076E:0012 A6            CMPSB
076E:0013 750A          JNZ     001F
076E:0015 B409          MOV     AH,09
076E:0017 BA1000        MOV     DX,0010
076E:001A CD21          INT     21
076E:001C EB0B          JMP     0029
076E:001E 90            NOP
076E:001F B409          MOV     AH,09
-g
strings are same
AX=096A  BX=0000  CX=0000  DX=0010  SP=0000  BP=0000  SI=0006  DI=000E
DS=076A  ES=076A  SS=0769  CS=076E  IP=0029   NV UP EI PL ZR NA PE NC
076E:0029 CC            INT     3
-_
```



```
076E:001F B409          MOV     AH,09
-g
strings are same
AX=096A  BX=0000  CX=0000  DX=0010  SP=0000  BP=0000  SI=0006  DI=000E
DS=076A  ES=076A  SS=0769  CS=076E  IP=0029   NV UP EI PL ZR NA PE NC
076E:0029 CC            INT     3
-d ds:00
076A:0000  61 61 6E 61 6E 64 69 24-61 61 6E 61 6E 64 69 24   aanandi$aanandi$
076A:0010  73 74 72 69 6E 67 73 20-61 72 65 20 73 61 6D 65   strings are same
076A:0020  24 73 74 72 69 6E 67 73-20 61 72 65 20 6E 6F 74   $strings are not
076A:0030  20 73 61 6D 65 24 00 00-00 00 00 00 00 00 00 00    same$..........
076A:0040  B8 6A 07 8E D8 8E C0 BE-00 00 BF 08 00 B9 06 00   .j..............
076A:0050  FC F3 A6 75 0A B4 09 BA-10 00 CD 21 EB 0B 90 B4   ...u.......!....
076A:0060  09 BA 21 00 CD 21 EB 01-90 CC 0C 00 26 89 36 1A   ..!..!......&.6.
076A:0070  00 26 89 3E 18 00 80 CB-20 26 88 1E 05 00 26 89   .&.>.... &....&.
-d es:00
076A:0000  61 61 6E 61 6E 64 69 24-61 61 6E 61 6E 64 69 24   aanandi$aanandi$
076A:0010  73 74 72 69 6E 67 73 20-61 72 65 20 73 61 6D 65   strings are same
076A:0020  24 73 74 72 69 6E 67 73-20 61 72 65 20 6E 6F 74   $strings are not
076A:0030  20 73 61 6D 65 24 00 00-00 00 00 00 00 00 00 00    same$..........
076A:0040  B8 6A 07 8E D8 8E C0 BE-00 00 BF 08 00 B9 06 00   .j..............
076A:0050  FC F3 A6 75 0A B4 09 BA-10 00 CD 21 EB 0B 90 B4   ...u.......!....
076A:0060  09 BA 21 00 CD 21 EB 01-90 CC 0C 00 26 89 36 1A   ..!..!......&.6.
076A:0070  00 26 89 3E 18 00 80 CB-20 26 88 1E 05 00 26 89   .&.>.... &....&.
-
```

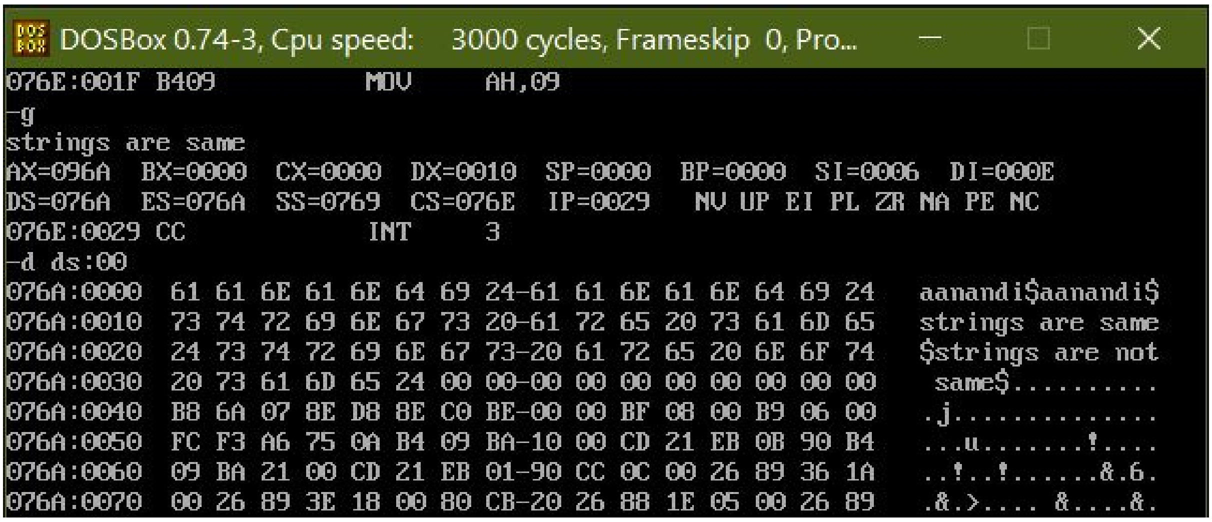


```
A:\>debug so12.exe
-g
strings are not same
AX=096A  BX=0000  CX=0005  DX=0021  SP=0000  BP=0000  SI=0001  DI=0009
DS=076A  ES=076A  SS=0769  CS=076E  IP=0029   NV UP EI NG NZ NA PO CY
076E:0029 CC            INT     3
-d ds:00
076A:0000  41 41 4E 41 4E 44 49 24-61 61 6E 61 6E 64 69 24   AANANDI$aanandi$
076A:0010  73 74 72 69 6E 67 73 20-61 72 65 20 73 61 6D 65   strings are same
076A:0020  24 73 74 72 69 6E 67 73-20 61 72 65 20 6E 6F 74   $strings are not
076A:0030  20 73 61 6D 65 24 00 00-00 00 00 00 00 00 00 00    same$..........
076A:0040  B8 6A 07 8E D8 8E C0 BE-00 00 BF 08 00 B9 06 00   .j..............
076A:0050  FC F3 A6 75 0A B4 09 BA-10 00 CD 21 EB 0B 90 B4   ...u.......!....
076A:0060  09 BA 21 00 CD 21 EB 01-90 CC 0C 00 26 89 36 1A   ..!..!......&.6.
076A:0070  00 26 89 3E 18 00 80 CB-20 26 88 1E 05 00 26 89   .&.>.... &....&.
-d es:00
076A:0000  41 41 4E 41 4E 44 49 24-61 61 6E 61 6E 64 69 24   AANANDI$aanandi$
076A:0010  73 74 72 69 6E 67 73 20-61 72 65 20 73 61 6D 65   strings are same
076A:0020  24 73 74 72 69 6E 67 73-20 61 72 65 20 6E 6F 74   $strings are not
076A:0030  20 73 61 6D 65 24 00 00-00 00 00 00 00 00 00 00    same$..........
076A:0040  B8 6A 07 8E D8 8E C0 BE-00 00 BF 08 00 B9 06 00   .j..............
076A:0050  FC F3 A6 75 0A B4 09 BA-10 00 CD 21 EB 0B 90 B4   ...u.......!....
076A:0060  09 BA 21 00 CD 21 EB 01-90 CC 0C 00 26 89 36 1A   ..!..!......&.6.
076A:0070  00 26 89 3E 18 00 80 CB-20 26 88 1E 05 00 26 89   .&.>.... &....&.
-
```

Output:

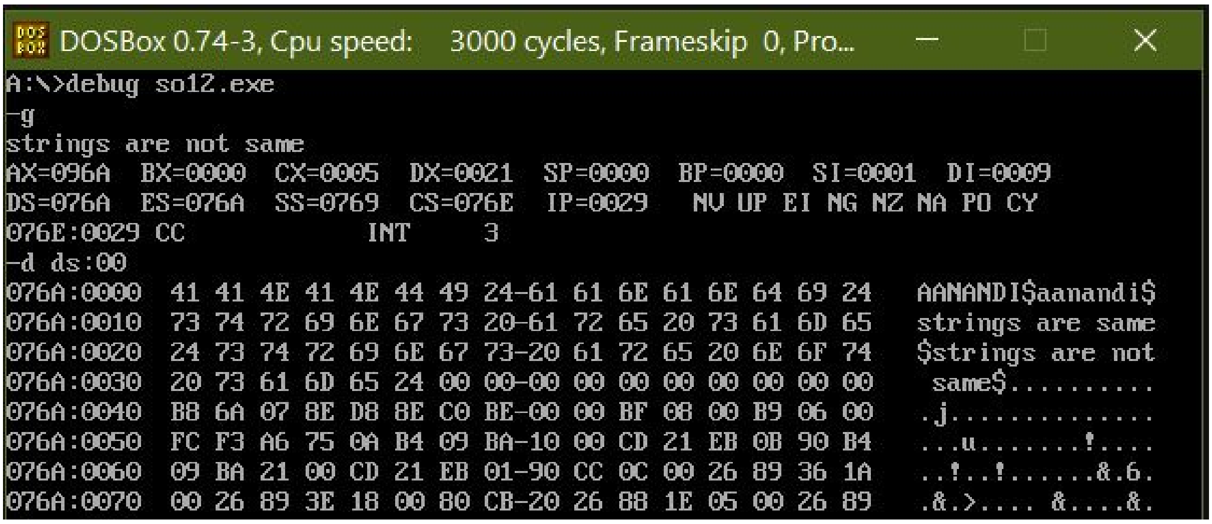
Screenshots of the output in both cases.

Case 1:Strings are same-



```
076E:001F B409          MOV     AH,09
-g
strings are same
AX=096A  BX=0000  CX=0000  DX=0010  SP=0000  BP=0000  SI=0006  DI=000E
DS=076A  ES=076A  SS=0769  CS=076E  IP=0029   NV UP EI PL ZR NA PE NC
076E:0029 CC            INT     3
-d ds:00
076A:0000  61 61 6E 61 6E 64 69 24-61 61 6E 61 6E 64 69 24   aanandi$aanandi$
076A:0010  73 74 72 69 6E 67 73 20-61 72 65 20 73 61 6D 65   strings are same
076A:0020  24 73 74 72 69 6E 67 73-20 61 72 65 20 6E 6F 74   $strings are not
076A:0030  20 73 61 6D 65 24 00 00-00 00 00 00 00 00 00 00    same$..........
076A:0040  B8 6A 07 8E D8 8E C0 BE-00 00 BF 08 00 B9 06 00   .j..............
076A:0050  FC F3 A6 75 0A B4 09 BA-10 00 CD 21 EB 0B 90 B4   ...u.......!....
076A:0060  09 BA 21 00 CD 21 EB 01-90 CC 0C 00 26 89 36 1A   ..!..!......&.6.
076A:0070  00 26 89 3E 18 00 80 CB-20 26 88 1E 05 00 26 89   .&.>.... &....&.
```

Case 2 :Strings are not same-



```
A:\>debug so12.exe
-g
strings are not same
AX=096A  BX=0000  CX=0005  DX=0021  SP=0000  BP=0000  SI=0001  DI=0009
DS=076A  ES=076A  SS=0769  CS=076E  IP=0029   NV UP EI NG NZ NA PO CY
076E:0029 CC            INT     3
-d ds:00
076A:0000  41 41 4E 41 4E 44 49 24-61 61 6E 61 6E 64 69 24   AANANDI$aanandi$
076A:0010  73 74 72 69 6E 67 73 20-61 72 65 20 73 61 6D 65   strings are same
076A:0020  24 73 74 72 69 6E 67 73-20 61 72 65 20 6E 6F 74   $strings are not
076A:0030  20 73 61 6D 65 24 00 00-00 00 00 00 00 00 00 00    same$..........
076A:0040  B8 6A 07 8E D8 8E C0 BE-00 00 BF 08 00 B9 06 00   .j..............
076A:0050  FC F3 A6 75 0A B4 09 BA-10 00 CD 21 EB 0B 90 B4   ...u.......!....
076A:0060  09 BA 21 00 CD 21 EB 01-90 CC 0C 00 26 89 36 1A   ..!..!......&.6.
076A:0070  00 26 89 3E 18 00 80 CB-20 26 88 1E 05 00 26 89   .&.>.... &....&.
```

3. **Write an assembly language program which accepts a character and a string from the user and prints the position of the character in to the string if it is found, otherwise the message "NOT FOUND". For simplicity, enter the sting with length in single digit, that is less than or equal to 9.**

   Rules for Operands: Take your name as an input string and search one of the character from it.

   Write your code here:

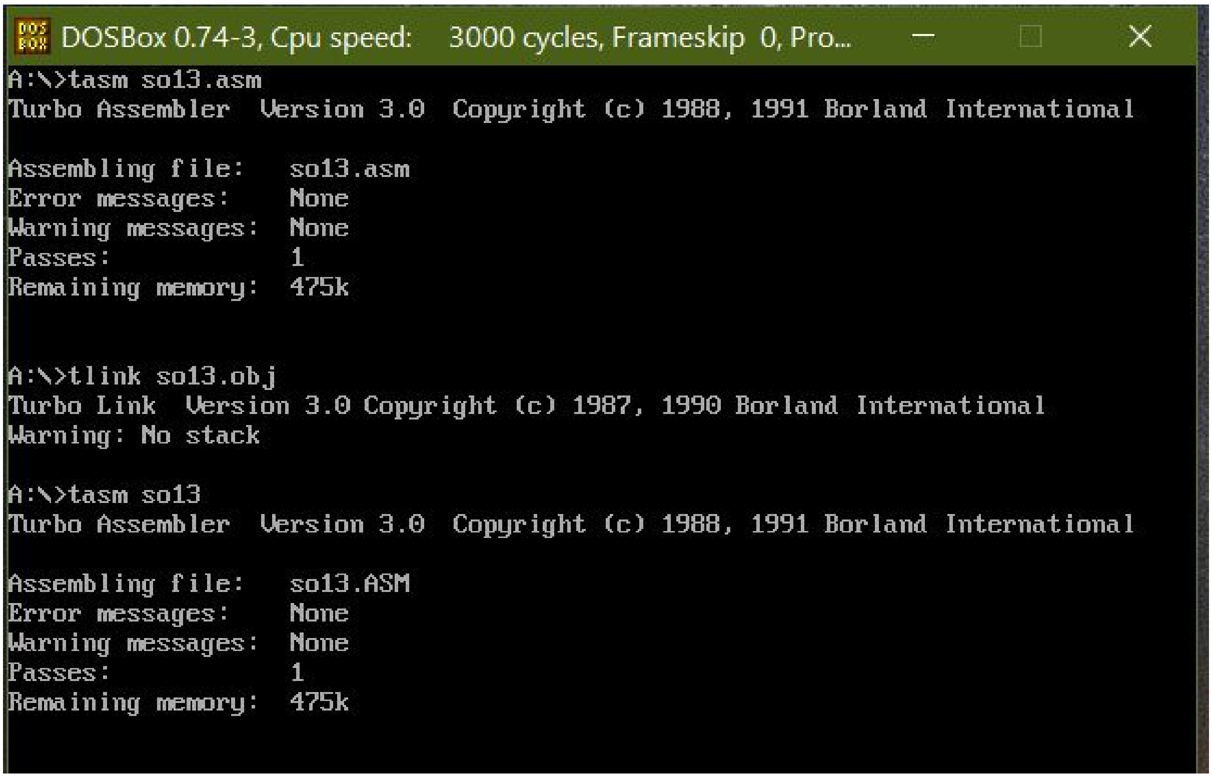
```
DATA SEGMENT

     OP1 DB "ENTER A STRING :$"
```

```
        STR_BUFF DB 15,16 DUP(0)
        OP2 DB 0Dh,0Ah,"ENTER A CHARACTER :$"
        MESS1 DB 0Dh,0Ah,"CHARACTER FOUND AT THE POSITION :$"
        MESS2 DB 0Dh,0Ah,"CHARACTER NOT FOUND$"
DATA ENDS

CODE SEGMENT
ASSUME CS:CODE , DS:DATA ,ES:DATA
START : MOV AX,DATA
                MOV DS,AX
                MOV ES,AX

          MOV AH,09h
          LEA DX,OP1              ;DISPLAY OP1 MESSAGE
          INT 21h

          MOV AH,0Ah
          LEA DX,STR_BUFF             ;GET A STRING IN STR_BUFF
          INT 21h

          MOV AH,09h
          LEA DX,OP2          ;DISPLAY OP2 MESSAGE
          INT 21h

          MOV AH,01h
          INT 21h                   ;GET A CHARACTER

          MOV DI,OFFSET STR_BUFF+1
          MOV CX,00
          MOV CL,BYTE PTR[DI]
```

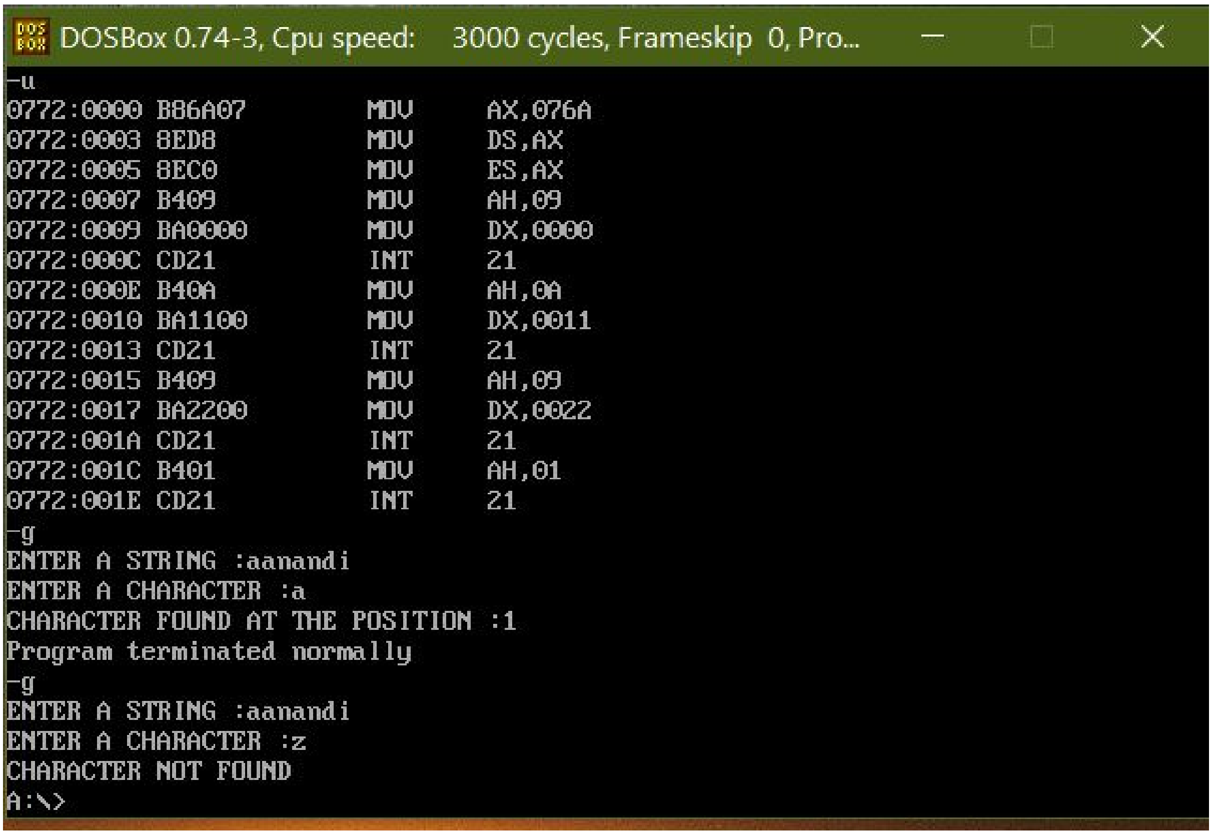
```
            INC DI

            MOV BX,DI

            CLD

REPNE  SCASB

            JNZ NOTFOUND


            MOV AH,9

            LEA DX,MESS1              ;CHARACTER FOUND

            INT 21h


            SUB DI,BX

            MOV DX,DI

            ADD DL,'0'

            MOV AH,2

            INT 21h

            JMP EXIT


NOTFOUND: MOV AH,09h

                 LEA DX,MESS2          ;CHARACTER NOT FOUND

                 INT 21h


EXIT :  MOV AX,4C00h

               INT 21h

CODE ENDS


END START
```

Compilation /Running and Debugging steps:

(As given in lab manual as an example of multiplication program on page no:5 of lab manual)



```
A:\>tasm so13.asm
Turbo Assembler  Version 3.0  Copyright (c) 1988, 1991 Borland International

Assembling file:   so13.asm
Error messages:    None
Warning messages:  None
Passes:            1
Remaining memory:  475k


A:\>tlink so13.obj
Turbo Link  Version 3.0 Copyright (c) 1987, 1990 Borland International
Warning: No stack

A:\>tasm so13
Turbo Assembler  Version 3.0  Copyright (c) 1988, 1991 Borland International

Assembling file:   so13.ASM
Error messages:    None
Warning messages:  None
Passes:            1
Remaining memory:  475k
```

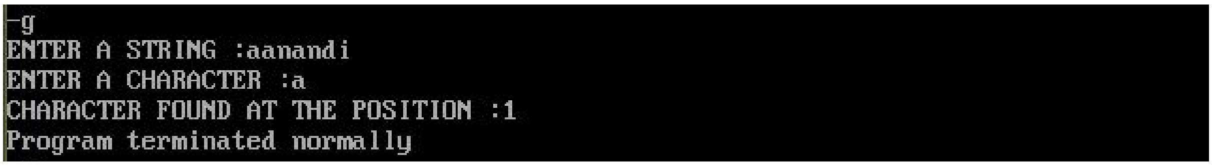


```
-u
0772:0000 B86A07        MOV     AX,076A
0772:0003 8ED8          MOV     DS,AX
0772:0005 8EC0          MOV     ES,AX
0772:0007 B409          MOV     AH,09
0772:0009 BA0000        MOV     DX,0000
0772:000C CD21          INT     21
0772:000E B40A          MOV     AH,0A
0772:0010 BA1100        MOV     DX,0011
0772:0013 CD21          INT     21
0772:0015 B409          MOV     AH,09
0772:0017 BA2200        MOV     DX,0022
0772:001A CD21          INT     21
0772:001C B401          MOV     AH,01
0772:001E CD21          INT     21
-g
ENTER A STRING :aanandi
ENTER A CHARACTER :a
CHARACTER FOUND AT THE POSITION :1
Program terminated normally
-g
ENTER A STRING :aanandi
ENTER A CHARACTER :z
CHARACTER NOT FOUND
A:\>
```

Output:

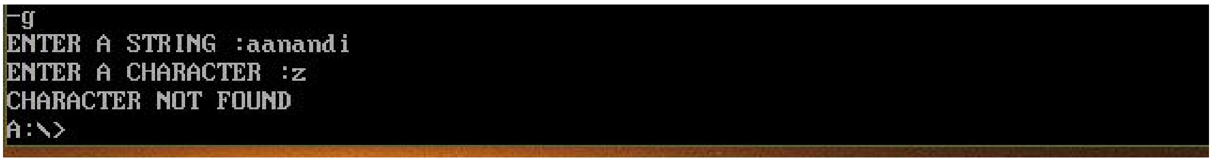
Screenshots of the output in both cases.

1. Character FOUND



```
-g
ENTER A STRING :aanandi
ENTER A CHARACTER :a
CHARACTER FOUND AT THE POSITION :1
Program terminated normally
```

2. Character NOT FOUND



```
-g
ENTER A STRING :aanandi
ENTER A CHARACTER :z
CHARACTER NOT FOUND
A:\>
```

Name:	Pankhania Aanandi R.
Roll No:	IT081
Batch:	I1

Experiment 4

AIM: To study multi module program

1. **Write a multi module assembly program to divide 32-bit number by 16-bit number and return a 32-bit quotient.**

Rules for Operands:

1. You have to use ascii values of the first 4-letters of your name as a **DIVIDEND**. E.g. According to my name (sunil), my DIVIDEND is: 73756E69h

LETTER (use lowercase letters)	ASCII Value in Hex
s	73h
u	75h
n	6Eh
i	69h

Note: If your name is having only 2/3 letters then consider the remaining letters as 00h e.g. "jay" so the dividend will be 6A617900h.

2. You have to use the ascii value of the first 2-letters of your name as a **DIVISOR**. E.g. According to my surname (VITHLANI), my DIVISOR is: 5649h

LETTER (use UPPERCASE letters)	ASCII Value in Hex
V	56h
I	49h

3. Clearly mention ascii values of your name and surname and then write your program.

Write your code here:

According to my name (aanandi), my DIVIDEND is: 61616E61h

LETTER (use lowercase letters)	ASCII Value in Hex
a	61h
a	61h
n	6Eh
a	61h

According to my surname (PANKHANIA), my DIVISOR is: 5041h

LETTER (use UPPERCASE letters)	ASCII Value in Hex
P	50h
A	41h

CODE:

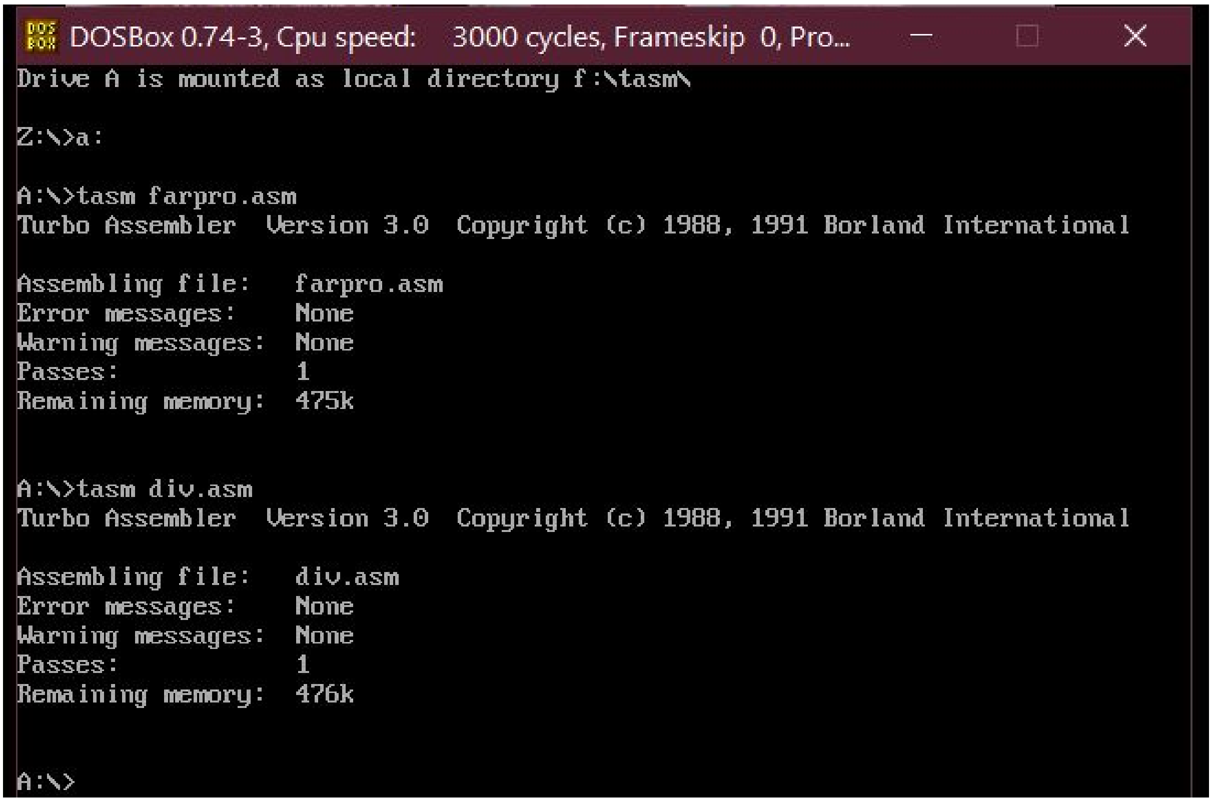
```
;MAIN PROGRAM : FARPRO

data_here segment word public

        dvd dw 6E61h,6161h

        dvs dw 5041h

data_here ends


data1_here segment word

        quotient dw 2 dup(0)

        reminder dw 0

data1_here ends


stack_here segment stack

        dw 30 dup(0)

        t1 label word

stack_here ends


public dvs


procedure_here segment public

        extrn division:Far

procedure_here ends


code_here segment word public
```

```
        assume cs:code_here,ds:data_here, ss:stack_here

start:  mov ax,data_here
                mov ds,ax
                mov ax,stack_here
                mov ss,ax
                mov sp,offset t1
                mov ax,dvd
                mov dx,dvd+2
                mov cx,dvs

                call division
                jnc X
                jmp q

                assume ds:data1_here

  X:    push ds
                mov bx,data1_here
                mov ds,bx

                mov quotient,ax
                mov quotient+2,dx
                mov reminder,cx

                assume ds:data_here
                pop ds

q:      int 3h
code_here ends
end start
```

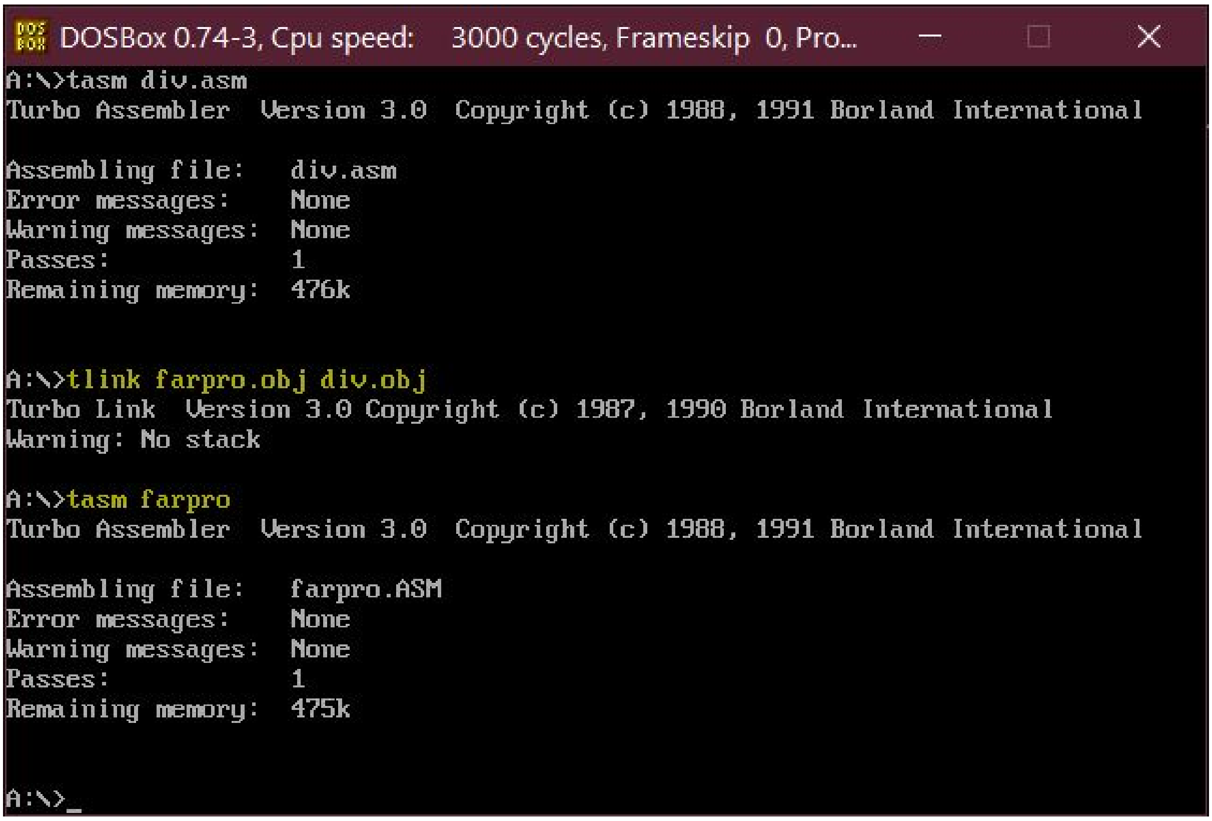
```
;MODULE
data_here segment public
        extrn dvs:word
data_here ends
public division
procedure_here segment public

        division proc far
                assume cs:procedure_here,ds:data_here
                cmp dvs,0
                je carry

                mov bx,ax
                mov ax,dx
                mov dx,0000h
                div cx
                mov bp,ax
                mov ax,bx
                div cx
                mov cx,dx
                mov dx,bp
                clc
                jmp q

carry:  stc
q:              ret
division endp

procedure_here ends
end
```

Compilation /Running and Debugging steps:

- Clearly mention each step

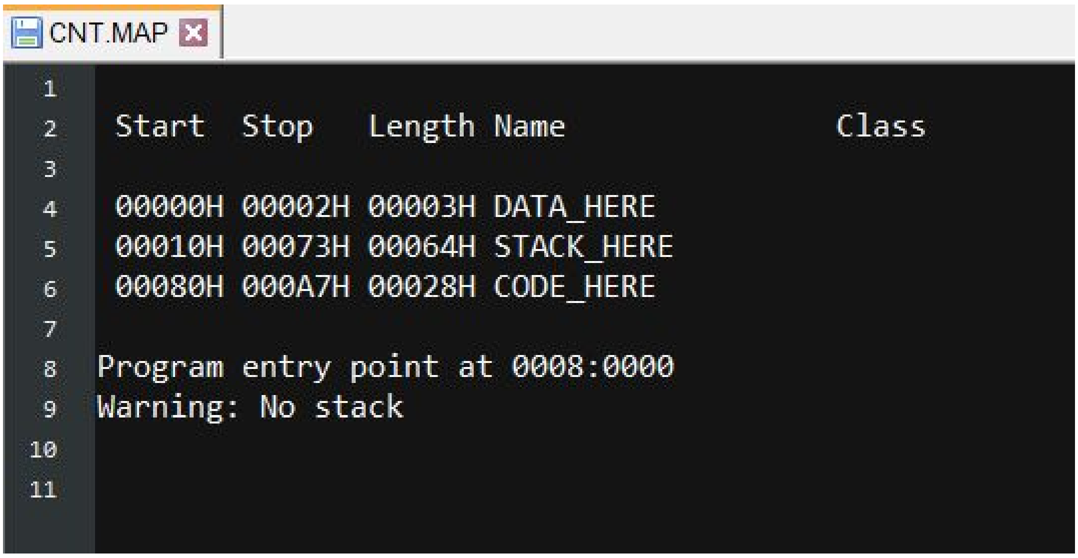


```
Drive A is mounted as local directory f:\tasm\

Z:\>a:

A:\>tasm farpro.asm
Turbo Assembler  Version 3.0  Copyright (c) 1988, 1991 Borland International

Assembling file:   farpro.asm
Error messages:    None
Warning messages:  None
Passes:            1
Remaining memory:  475k


A:\>tasm div.asm
Turbo Assembler  Version 3.0  Copyright (c) 1988, 1991 Borland International

Assembling file:   div.asm
Error messages:    None
Warning messages:  None
Passes:            1
Remaining memory:  476k


A:\>
```



```
A:\>tasm div.asm
Turbo Assembler  Version 3.0  Copyright (c) 1988, 1991 Borland International

Assembling file:   div.asm
Error messages:    None
Warning messages:  None
Passes:            1
Remaining memory:  476k


A:\>tlink farpro.obj div.obj
Turbo Link  Version 3.0 Copyright (c) 1987, 1990 Borland International
Warning: No stack

A:\>tasm farpro
Turbo Assembler  Version 3.0  Copyright (c) 1988, 1991 Borland International

Assembling file:   farpro.ASM
Error messages:    None
Warning messages:  None
Passes:            1
Remaining memory:  475k


A:\>_
```

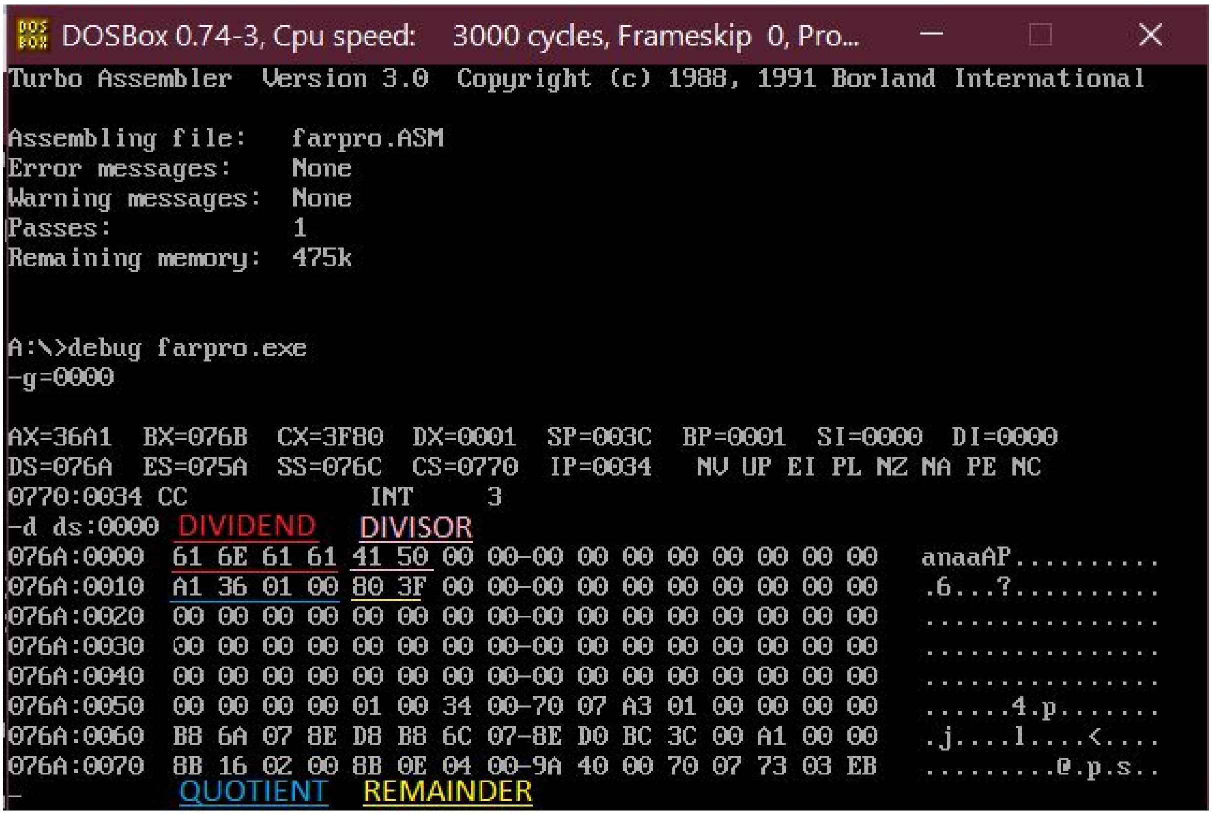
- Put a screenshot of the mapping file. (Generated after linkage of object files)



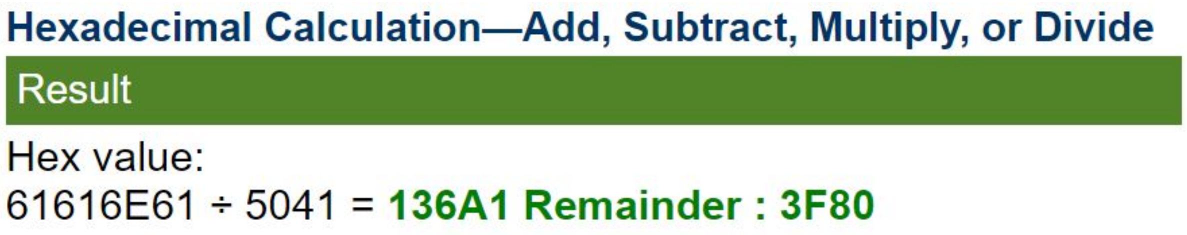
```

 Start  Stop   Length Name               Class

 00000H 00002H 00003H DATA_HERE
 00010H 00073H 00064H STACK_HERE
 00080H 000A7H 00028H CODE_HERE

Program entry point at 0008:0000
Warning: No stack

```

Output:

Screenshots of memory that contains values of DIVIDENT, DIVISOR, QUOTIENT and REMINDER (output of -d ds:offset_addres command.)


DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Pro...
DIVIDEND
DIVISOR
QUOTIENT
REMAINDER

```
Turbo Assembler  Version 3.0  Copyright (c) 1988, 1991 Borland International

Assembling file:   farpro.ASM
Error messages:    None
Warning messages:  None
Passes:            1
Remaining memory:  475k


A:\>debug farpro.exe
-g=0000

AX=36A1  BX=076B  CX=3F80  DX=0001  SP=003C  BP=0001  SI=0000  DI=0000
DS=076A  ES=075A  SS=076C  CS=0770  IP=0034   NV UP EI PL NZ NA PE NC
0770:0034 CC            INT     3
-d ds:0000
076A:0000  61 6E 61 61 41 50 00 00-00 00 00 00 00 00 00 00   anaaAP..........
076A:0010  A1 36 01 00 80 3F 00 00-00 00 00 00 00 00 00 00   .6...?..........
076A:0020  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0030  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0040  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0050  00 00 00 00 01 00 34 00-70 07 A3 01 00 00 00 00   ......4.p.......
076A:0060  B8 6A 07 8E D8 B8 6C 07-8E D0 BC 3C 00 A1 00 00   .j....l....<....
076A:0070  8B 16 02 00 8B 0E 04 00-9A 40 00 70 07 73 03 EB   .........@.p.s..
-
```



Hexadecimal Calculation—Add, Subtract, Multiply, or Divide

Result

Hex value:
61616E61 ÷ 5041 = 136A1 Remainder : 3F80

2. **Write an assembly language program to develop a far procedure to find whether the given number is EVEN or ODD and print message appropriately. Write a main program to call this far procedure and pass the roll_no as a parameter to the far procedure.**

   Rules for Operands:

   1. You have to pass your Roll_NO/ID_no for repeater students as a parameter to the procedure.
   2. You can use multi module program or single module program **(If you are using single module then first define procedure segment and then code segment in your program)**

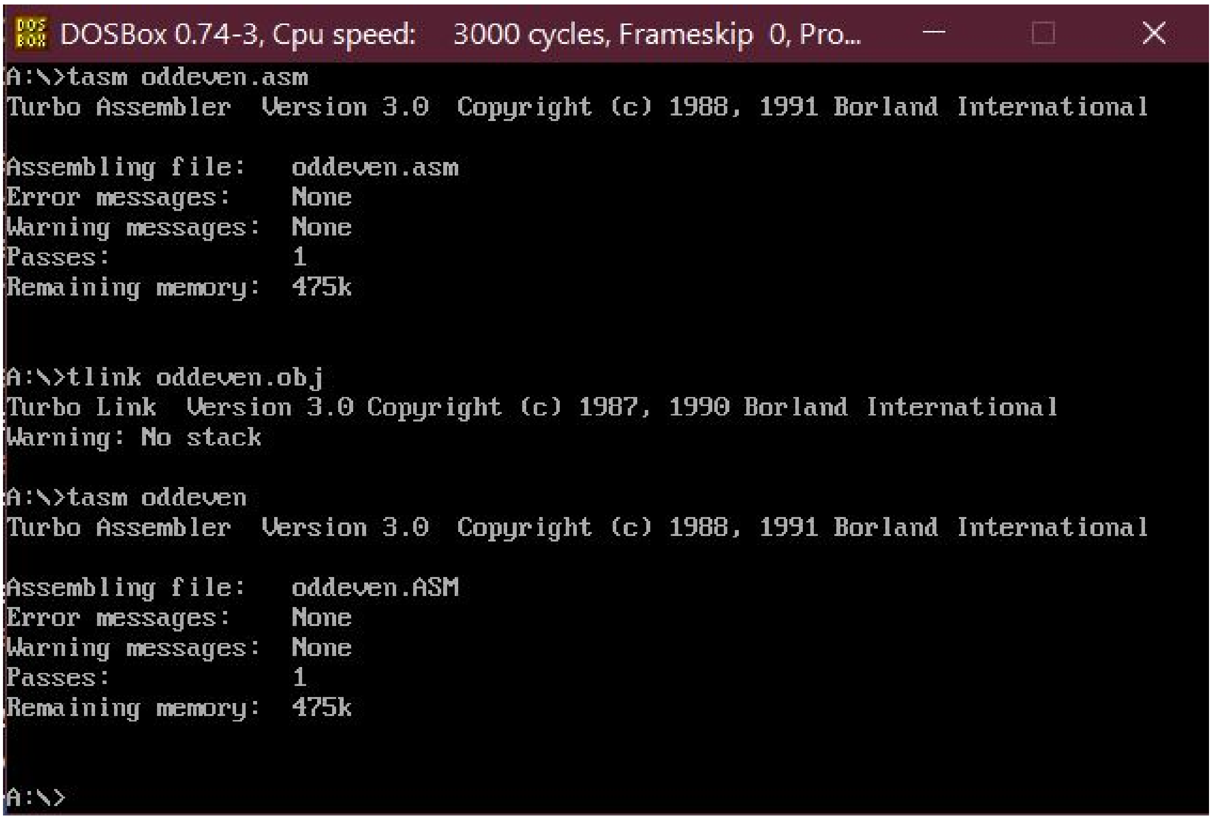
Write your code here:

```
Data_here segment
        num DW 0081H ; REQ. INPUT : IT081(MY ROLL NUM)
        msg1 db 'Given number is ODD$'
        msg2 db'Given number is EVEN$'
Data_here ends


Stack_here segment stack
        dw 50 dup(0)
        stk1 label word
Stack_here ends


msgproc segment
Check proc far
Assume  cs:msgproc
                PUSHF
```

```
                    PUSH DX
                    shr ax,01H
                    jnc evn

odd :   MOV AH,09h
                              MOV DX,offset msg1
                              INT 21h
                              JMP q

evn :  MOV AH,09h
                              MOV DX,offset msg2
                              INT 21h
                              JMP q

q          :        POP DX
                    POPF
                    RET
                    Check endp
msgproc ends
Code_here segment
Assume cs:Code_here ,ds:Data_here ,ss:Stack_here
Start : mov ax,Data_here
                    mov ds,ax
                    mov ax,Stack_here
                    mov ss,ax
                    LEA SP,stk1
                    mov ax,num
                    CALL Check
                    INT 3h
Code_here ends
End Start
```

Compilation /Running and Debugging steps:

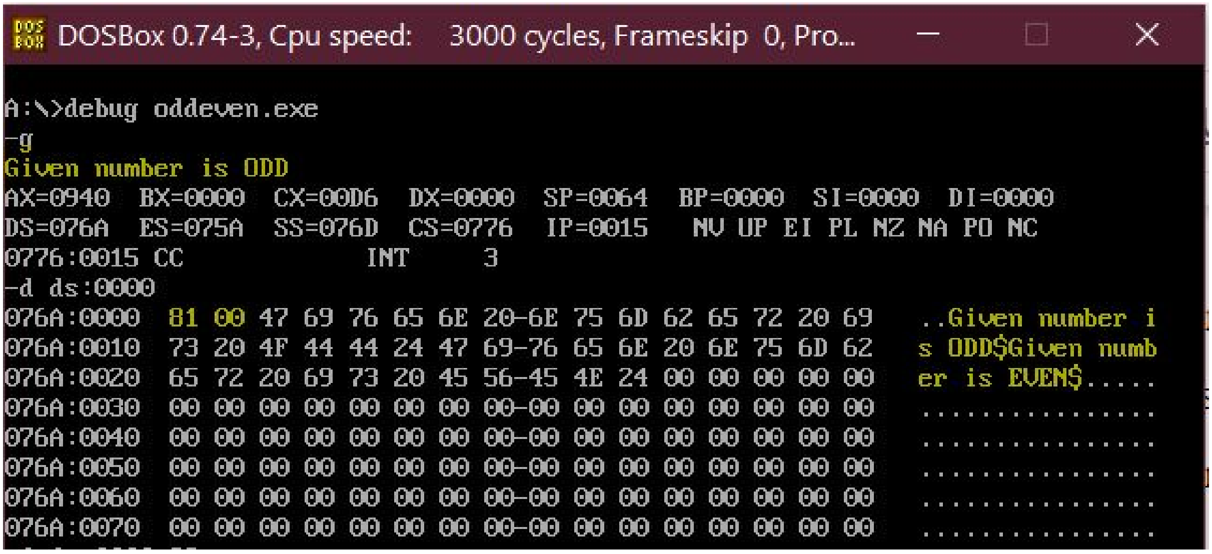
(As given in lab manual as an example of multiplication program on page no:5 of lab manual)



```
A:\>tasm oddeven.asm
Turbo Assembler  Version 3.0  Copyright (c) 1988, 1991 Borland International

Assembling file:   oddeven.asm
Error messages:    None
Warning messages:  None
Passes:            1
Remaining memory:  475k


A:\>tlink oddeven.obj
Turbo Link  Version 3.0 Copyright (c) 1987, 1990 Borland International
Warning: No stack

A:\>tasm oddeven
Turbo Assembler  Version 3.0  Copyright (c) 1988, 1991 Borland International

Assembling file:   oddeven.ASM
Error messages:    None
Warning messages:  None
Passes:            1
Remaining memory:  475k


A:\>
```

Output:

1. Screenshot of the memory where you have stored your number.
2. Screenshot of the output message. (e.g. “Your roll_no is EVEN”)



```
A:\>debug oddeven.exe
-g
Given number is ODD
AX=0940  BX=0000  CX=00D6  DX=0000  SP=0064  BP=0000  SI=0000  DI=0000
DS=076A  ES=075A  SS=076D  CS=0776  IP=0015   NV UP EI PL NZ NA PO NC
0776:0015 CC            INT     3
-d ds:0000
076A:0000  81 00 47 69 76 65 6E 20-6E 75 6D 62 65 72 20 69   ..Given number i
076A:0010  73 20 4F 44 44 24 47 69-76 65 6E 20 6E 75 6D 62   s ODD$Given numb
076A:0020  65 72 20 69 73 20 45 56-45 4E 24 00 00 00 00 00   er is EVEN$.....
076A:0030  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0040  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0050  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0060  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0070  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
```

Name:	Pankhania Aanandi R.
Roll No:	IT081
Batch:	I1

Experiment 5

AIM: To Study the response of Type-0 interrupt.

1. Write an assembly language program of dividing four numbers. If the result of the division is too large to fit in the quotient register then the 8086 will do a type 0 interrupt immediately after the divide instruction finishes.
   - Write two programs one is main line program which contains div instruction and second program is interrupt service routine which handles the type 0 interrupt.

   Rules for Operands:

   1. You have to use following values as dividend
      DIVIDEND DW 00ABh,0CDEh,7FFFh,0FFFFh

   2. You have to use the ASCII value (in hex) of the first 1-letters of your name as a **DIVISOR**.
      E.g. According to my surname (VITHLANI), my DIVISOR is: 56h

LETTER (use UPPERCASE letters)	ASCII Value in Hex
V	56h

   3. Clearly mention ASCII values of surname and then write your program.

Write your code here:

According to my surname (PANKHANIA), my DIVISOR is: 56h

LETTER (use UPPERCASE letters)	ASCII Value in Hex
P	50h

----

1. **isrexp.asm**

```
DATA_HERE SEGMENT WORD PUBLIC
        INPUT DW 00ABH,0CDEH,7FFFH,0FFFFH
        QUOTIENTS DB 4 DUP(0)
        DIVISOR DB 50H ; DIVISOR P(50H)
        FLAGS DB 4 DUP (0)
        EFLAG DB 0 ; ERROR FLAG
DATA_HERE ENDS
```

```
STACK_HERE SEGMENT STACK
        DW 100 DUP(0)
        STACK1 LABEL WORD
STACK_HERE ENDS

PUBLIC EFLAG

        PROC_HERE SEGMENT WORD PUBLIC
        EXTRN DIV_PROC : FAR
        PROC_HERE ENDS

CODE_HERE SEGMENT WORD PUBLIC

                ASSUME CS:CODE_HERE , DS:DATA_HERE , SS:STACK_HERE
START:          MOV AX , STACK_HERE
                MOV SS , AX
                MOV SP , OFFSET STACK1
                MOV AX , DATA_HERE
                MOV DS , AX
                MOV AX,0000
                MOV ES, AX

                ;CHANGE INTERRUPT TYPE0

                MOV WORD PTR ES:0002,SEG DIV_PROC
                MOV WORD PTR ES:0000,OFFSET DIV_PROC
                MOV SI,OFFSET INPUT
                MOV BX,OFFSET QUOTIENTS
                MOV DI,OFFSET FLAGS
                MOV CX,0004
NEXT:   MOV AX,[SI]
                DIV DIVISOR
                CMP EFLAG,01
                JNE NXT
                MOV BYTE PTR[BX],00
                MOV BYTE PTR[DI],01
                JMP NXT1
NXT:    MOV [BX],AL
                MOV BYTE PTR[DI],00
NXT1:   MOV EFLAG,00
                ADD SI,02H
                INC BX
                INC DI
                LOOP NEXT
STOP: NOP
CODE_HERE ENDS
END START
```

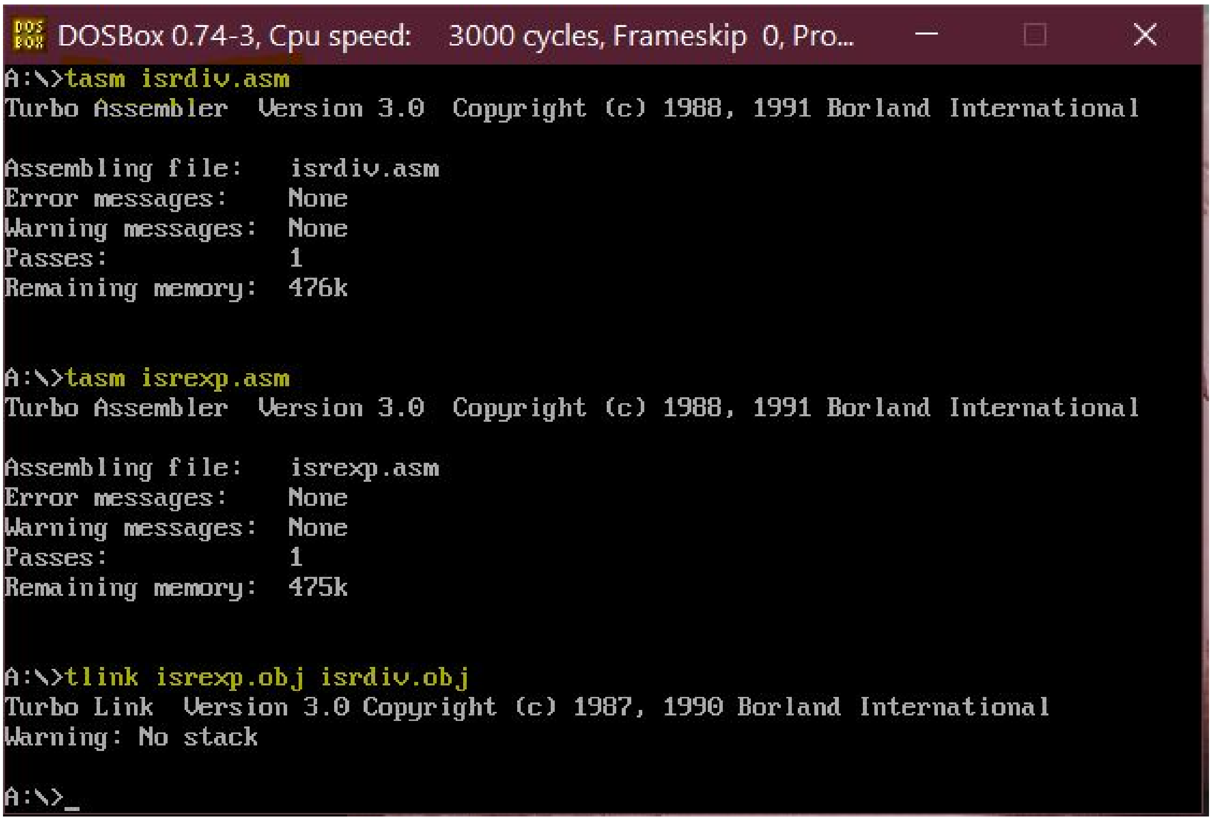
2. **isrdiv.asm**

```
        DATA_HERE SEGMENT WORD PUBLIC
        EXTRN EFLAG: BYTE
DATA_HERE ENDS

PUBLIC DIV_PROC

PROC_HERE SEGMENT WORD PUBLIC
        DIV_PROC PROC FAR
                        ASSUME CS:PROC_HERE, DS:DATA_HERE
                        PUSH AX
                        PUSH DS
                        PUSH BX

                        MOV AX, DATA_HERE
                        MOV DS,AX

                        MOV BP, SP ; INCREMENT IP BY 4
                        MOV BX, WORD PTR [BP+6]
                        ADD BX, 04H
                        MOV [BP+6],BX

                        MOV EFLAG,01 ; SET EFLAG(ERROR FLAG) 1

                        POP BX
                        POP DS
                        POP AX
                        IRET
                DIV_PROC ENDP
        PROC_HERE ENDS
```

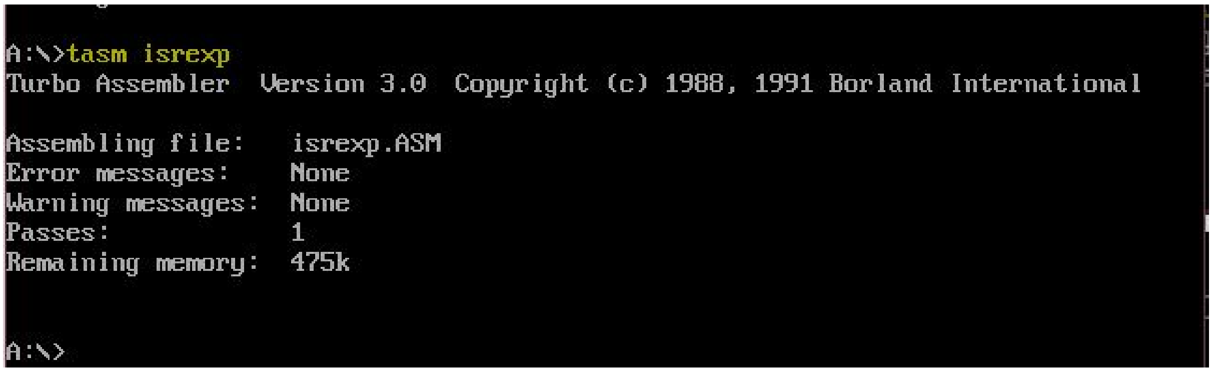
END

Compilation /Running and Debugging steps:

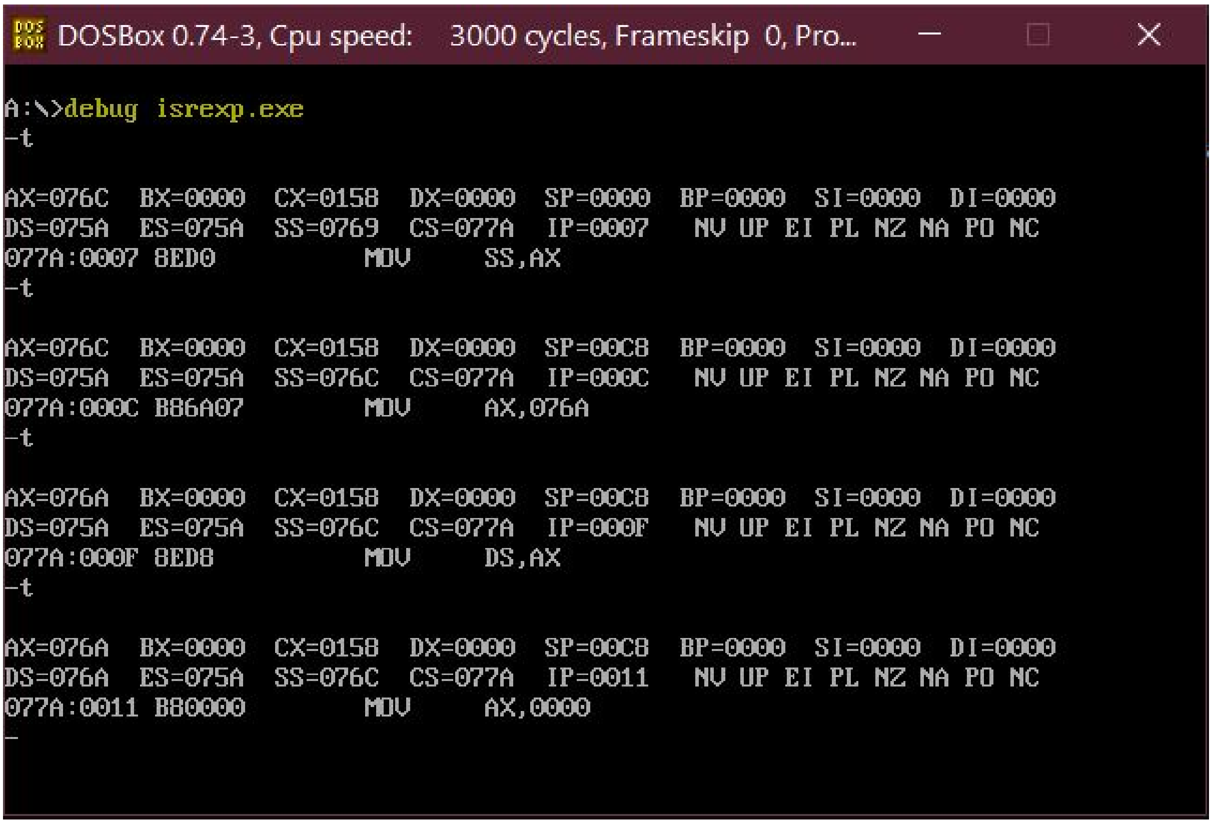
- Clearly mention each step


DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Pro...

```
A:\>tasm isrdiv.asm
Turbo Assembler  Version 3.0  Copyright (c) 1988, 1991 Borland International

Assembling file:   isrdiv.asm
Error messages:    None
Warning messages:  None
Passes:            1
Remaining memory:  476k


A:\>tasm isrexp.asm
Turbo Assembler  Version 3.0  Copyright (c) 1988, 1991 Borland International

Assembling file:   isrexp.asm
Error messages:    None
Warning messages:  None
Passes:            1
Remaining memory:  475k


A:\>tlink isrexp.obj isrdiv.obj
Turbo Link  Version 3.0 Copyright (c) 1987, 1990 Borland International
Warning: No stack

A:\>_
```

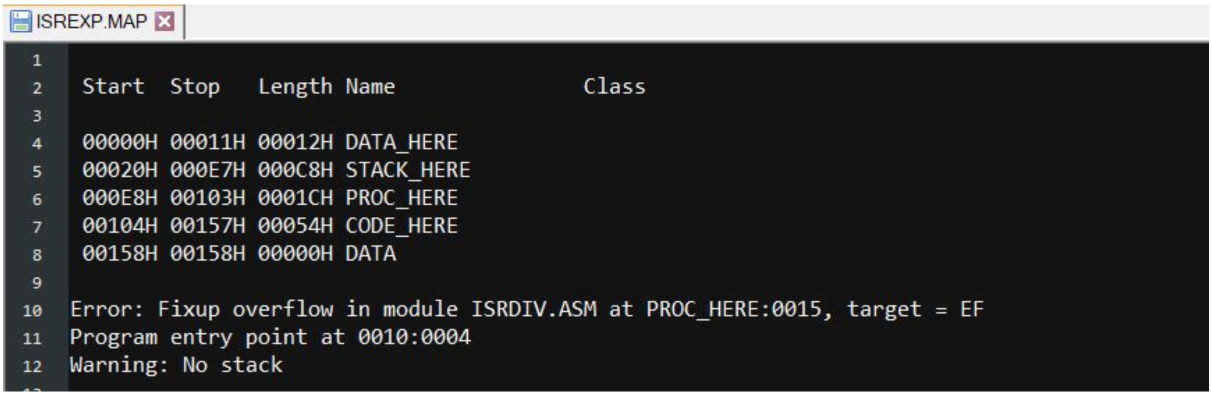


```
A:\>tasm isrexp
Turbo Assembler  Version 3.0  Copyright (c) 1988, 1991 Borland International

Assembling file:   isrexp.ASM
Error messages:    None
Warning messages:  None
Passes:            1
Remaining memory:  475k


A:\>
```

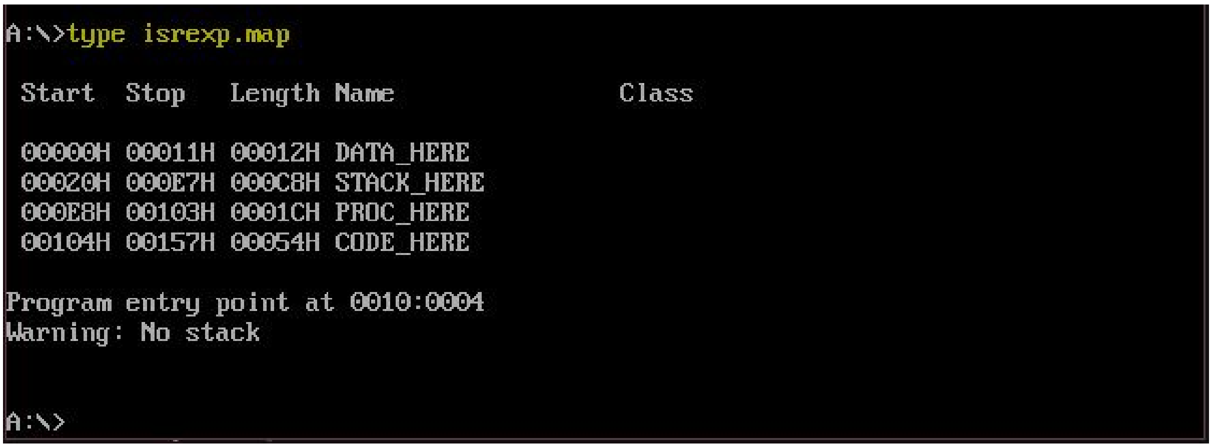


```
A:\>debug isrexp.exe
-t

AX=076C  BX=0000  CX=0158  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=075A  ES=075A  SS=0769  CS=077A  IP=0007   NV UP EI PL NZ NA PO NC
077A:0007 8ED0          MOV     SS,AX
-t

AX=076C  BX=0000  CX=0158  DX=0000  SP=00C8  BP=0000  SI=0000  DI=0000
DS=075A  ES=075A  SS=076C  CS=077A  IP=000C   NV UP EI PL NZ NA PO NC
077A:000C B86A07        MOV     AX,076A
-t

AX=076A  BX=0000  CX=0158  DX=0000  SP=00C8  BP=0000  SI=0000  DI=0000
DS=075A  ES=075A  SS=076C  CS=077A  IP=000F   NV UP EI PL NZ NA PO NC
077A:000F 8ED8          MOV     DS,AX
-t

AX=076A  BX=0000  CX=0158  DX=0000  SP=00C8  BP=0000  SI=0000  DI=0000
DS=076A  ES=075A  SS=076C  CS=077A  IP=0011   NV UP EI PL NZ NA PO NC
077A:0011 B80000        MOV     AX,0000
-
```

- Put a screenshot of the mapping file. (Generated after linkage of object files)



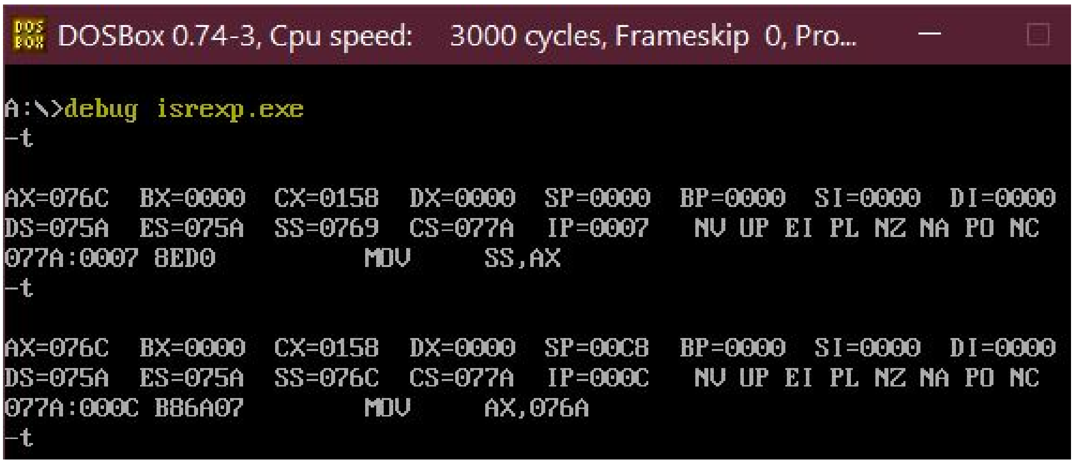
```

 Start  Stop   Length Name               Class

 00000H 00011H 00012H DATA_HERE
 00020H 000E7H 000C8H STACK_HERE
 000E8H 00103H 0001CH PROC_HERE
 00104H 00157H 00054H CODE_HERE
 00158H 00158H 00000H DATA

Error: Fixup overflow in module ISRDIV.ASM at PROC_HERE:0015, target = EF
Program entry point at 0010:0004
Warning: No stack
```



```
A:\>type isrexp.map

 Start  Stop   Length Name               Class

 00000H 00011H 00012H DATA_HERE
 00020H 000E7H 000C8H STACK_HERE
 000E8H 00103H 0001CH PROC_HERE
 00104H 00157H 00054H CODE_HERE

Program entry point at 0010:0004
Warning: No stack


A:\>
```

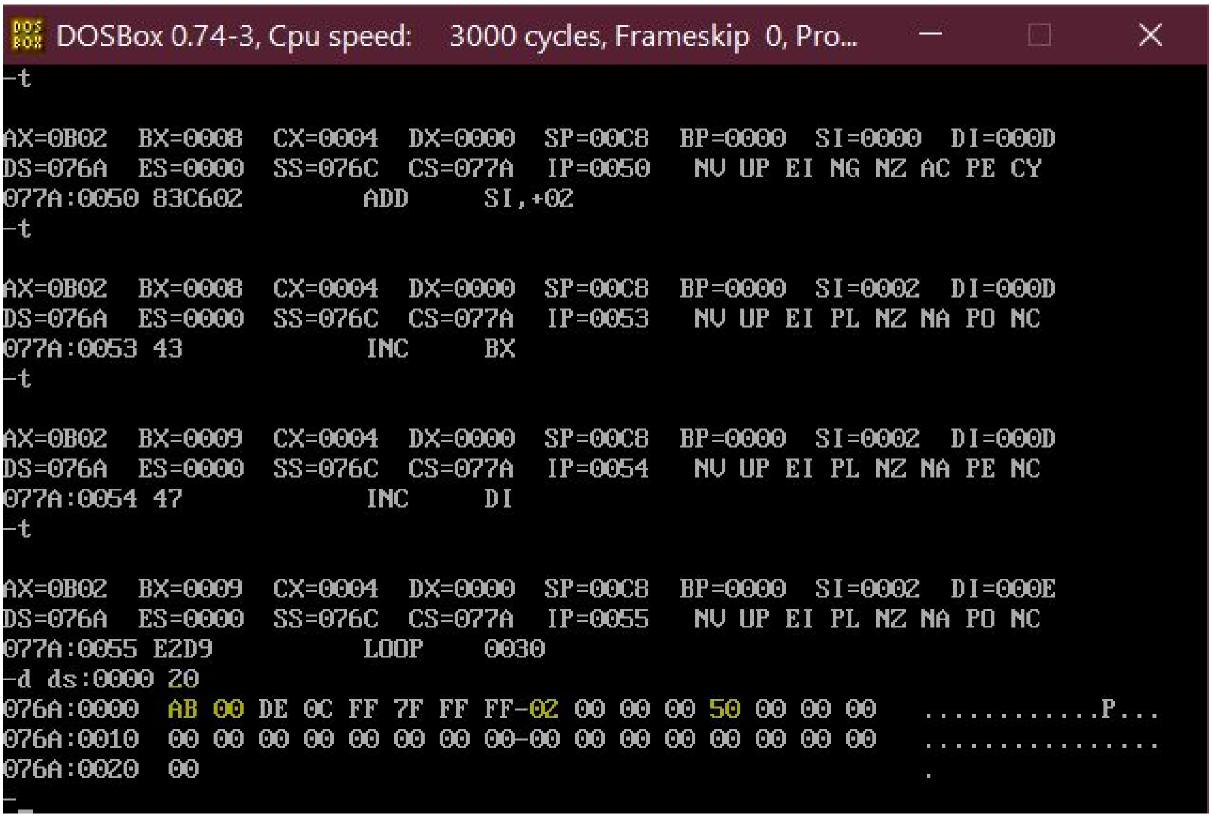
Output:

1. Screenshot of memory after each iteration of loop. (See below screenshot for more clarification)



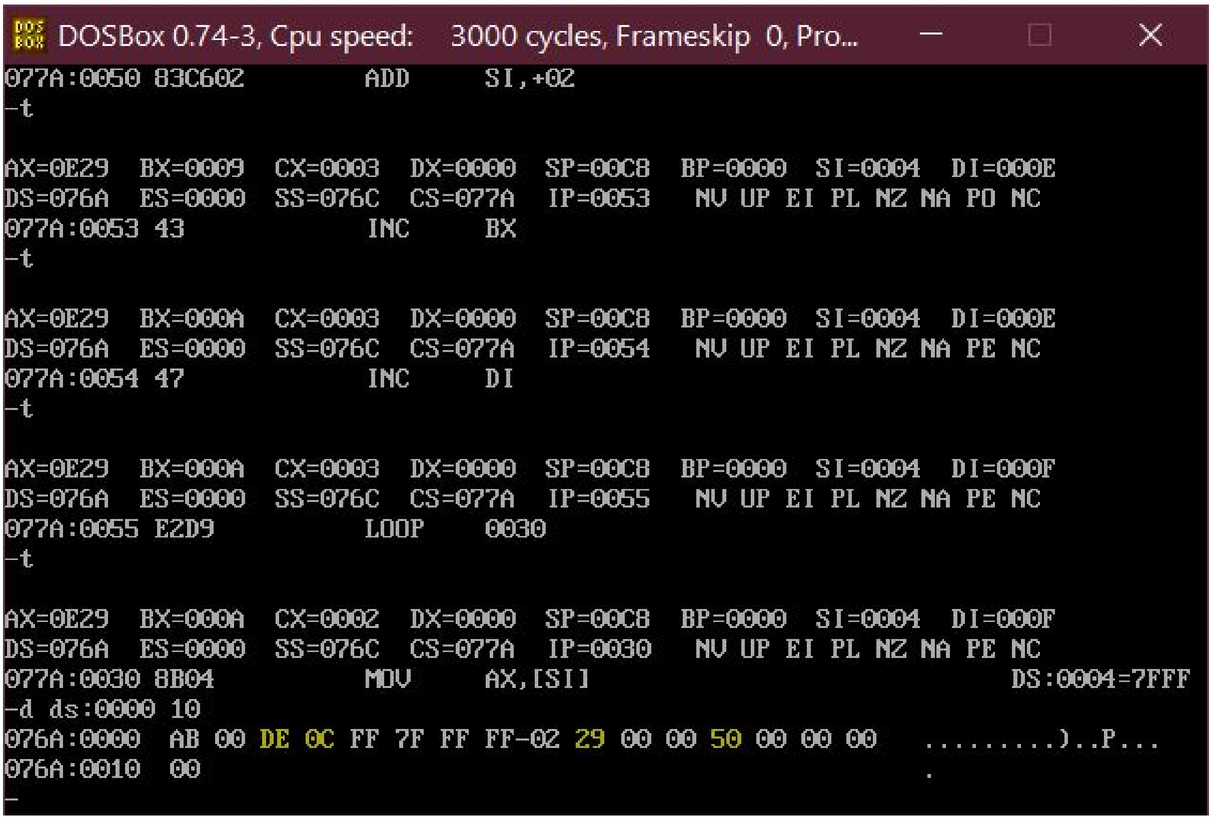
```
A:\>debug isrexp.exe
-t

AX=076C  BX=0000  CX=0158  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=075A  ES=075A  SS=0769  CS=077A  IP=0007   NV UP EI PL NZ NA PO NC
077A:0007 8ED0          MOV     SS,AX
-t

AX=076C  BX=0000  CX=0158  DX=0000  SP=00C8  BP=0000  SI=0000  DI=0000
DS=075A  ES=075A  SS=076C  CS=077A  IP=000C   NV UP EI PL NZ NA PO NC
077A:000C B86A07        MOV     AX,076A
-t
```

2. Our 1st number is 00ABh, so in the 1st screenshot highlight this number from memory and its quotient & division flag stored in memory. As below screenshot.



```
-t

AX=0B02  BX=0008  CX=0004  DX=0000  SP=00C8  BP=0000  SI=0000  DI=000D
DS=076A  ES=0000  SS=076C  CS=077A  IP=0050   NV UP EI NG NZ AC PE CY
077A:0050 83C602        ADD     SI,+02
-t

AX=0B02  BX=0008  CX=0004  DX=0000  SP=00C8  BP=0000  SI=0002  DI=000D
DS=076A  ES=0000  SS=076C  CS=077A  IP=0053   NV UP EI PL NZ NA PO NC
077A:0053 43            INC     BX
-t

AX=0B02  BX=0009  CX=0004  DX=0000  SP=00C8  BP=0000  SI=0002  DI=000D
DS=076A  ES=0000  SS=076C  CS=077A  IP=0054   NV UP EI PL NZ NA PE NC
077A:0054 47            INC     DI
-t

AX=0B02  BX=0009  CX=0004  DX=0000  SP=00C8  BP=0000  SI=0002  DI=000E
DS=076A  ES=0000  SS=076C  CS=077A  IP=0055   NV UP EI PL NZ NA PO NC
077A:0055 E2D9          LOOP    0030
-d ds:0000 20
076A:0000  AB 00 DE 0C FF 7F FF FF-02 00 00 00 50 00 00 00   ............P...
076A:0010  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0020  00                                                .
-_
```

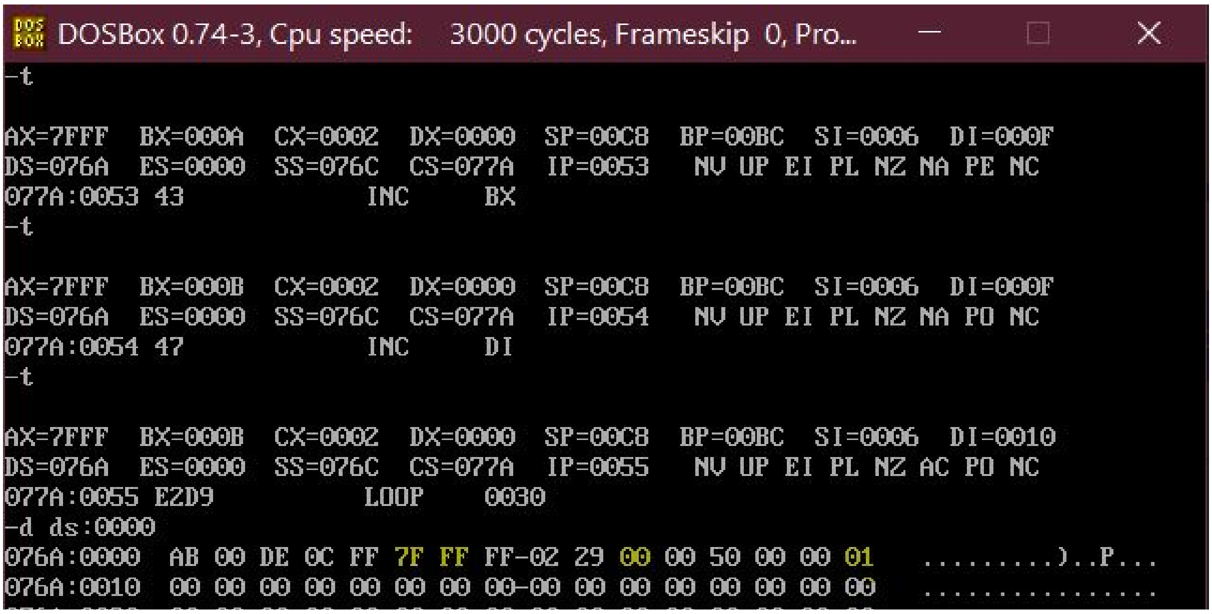
3. Our 2nd number is 0CDEh, so in the 2nd screen shot highlight this number from memory and its quotient & division flag stored in memory. As below screenshot.



DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Pro...

```
077A:0050 83C602        ADD     SI,+02
-t

AX=0E29  BX=0009  CX=0003  DX=0000  SP=00C8  BP=0000  SI=0004  DI=000E
DS=076A  ES=0000  SS=076C  CS=077A  IP=0053   NV UP EI PL NZ NA PO NC
077A:0053 43            INC     BX
-t

AX=0E29  BX=000A  CX=0003  DX=0000  SP=00C8  BP=0000  SI=0004  DI=000E
DS=076A  ES=0000  SS=076C  CS=077A  IP=0054   NV UP EI PL NZ NA PE NC
077A:0054 47            INC     DI
-t

AX=0E29  BX=000A  CX=0003  DX=0000  SP=00C8  BP=0000  SI=0004  DI=000F
DS=076A  ES=0000  SS=076C  CS=077A  IP=0055   NV UP EI PL NZ NA PE NC
077A:0055 E2D9          LOOP    0030
-t

AX=0E29  BX=000A  CX=0002  DX=0000  SP=00C8  BP=0000  SI=0004  DI=000F
DS=076A  ES=0000  SS=076C  CS=077A  IP=0030   NV UP EI PL NZ NA PE NC
077A:0030 8B04          MOV     AX,[SI]                            DS:0004=7FFF
-d ds:0000 10
076A:0000  AB 00 DE 0C FF 7F FF FF-02 29 00 00 50 00 00 00   .........)..P...
076A:0010  00                                                .
-
```

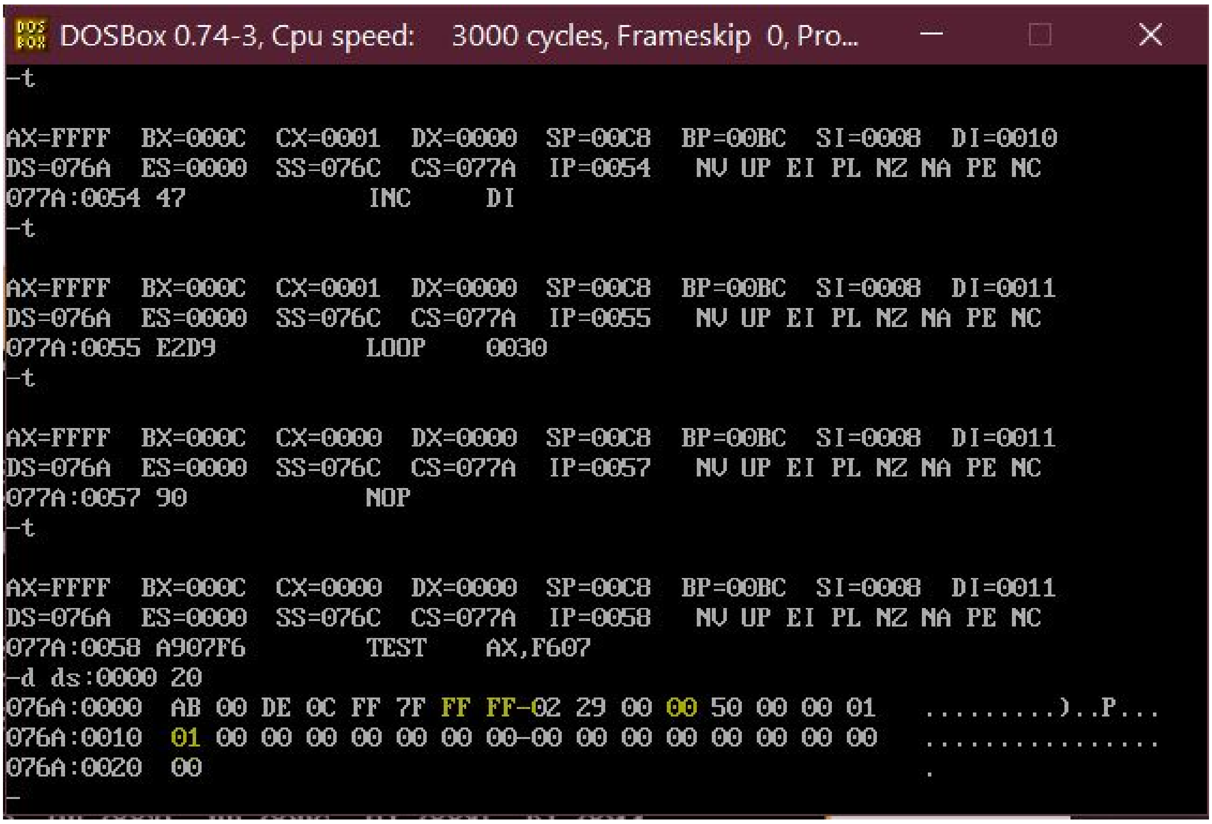
4. Our 3rd number is 7FFFh, so in 3rd screen shot highlight this number from memory and its quotient & division flag stored in memory. As below screen shot.



DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Pro...

```
-t

AX=7FFF  BX=000A  CX=0002  DX=0000  SP=00C8  BP=00BC  SI=0006  DI=000F
DS=076A  ES=0000  SS=076C  CS=077A  IP=0053   NV UP EI PL NZ NA PE NC
077A:0053 43            INC     BX
-t

AX=7FFF  BX=000B  CX=0002  DX=0000  SP=00C8  BP=00BC  SI=0006  DI=000F
DS=076A  ES=0000  SS=076C  CS=077A  IP=0054   NV UP EI PL NZ NA PO NC
077A:0054 47            INC     DI
-t

AX=7FFF  BX=000B  CX=0002  DX=0000  SP=00C8  BP=00BC  SI=0006  DI=0010
DS=076A  ES=0000  SS=076C  CS=077A  IP=0055   NV UP EI PL NZ AC PO NC
077A:0055 E2D9          LOOP    0030
-d ds:0000
076A:0000  AB 00 DE 0C FF 7F FF FF-02 29 00 00 50 00 00 01   .........)..P...
076A:0010  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
```

5. Our 4th number is 0FFFFh, so in 4th screen shot highlight this number from memory and its quotient & division flag stored in memory. As below screen shot.



```
-t

AX=FFFF  BX=000C  CX=0001  DX=0000  SP=00C8  BP=00BC  SI=0008  DI=0010
DS=076A  ES=0000  SS=076C  CS=077A  IP=0054   NV UP EI PL NZ NA PE NC
077A:0054 47            INC     DI
-t

AX=FFFF  BX=000C  CX=0001  DX=0000  SP=00C8  BP=00BC  SI=0008  DI=0011
DS=076A  ES=0000  SS=076C  CS=077A  IP=0055   NV UP EI PL NZ NA PE NC
077A:0055 E2D9          LOOP    0030
-t

AX=FFFF  BX=000C  CX=0000  DX=0000  SP=00C8  BP=00BC  SI=0008  DI=0011
DS=076A  ES=0000  SS=076C  CS=077A  IP=0057   NV UP EI PL NZ NA PE NC
077A:0057 90            NOP
-t

AX=FFFF  BX=000C  CX=0000  DX=0000  SP=00C8  BP=00BC  SI=0008  DI=0011
DS=076A  ES=0000  SS=076C  CS=077A  IP=0058   NV UP EI PL NZ NA PE NC
077A:0058 A907F6        TEST    AX,F607
-d ds:0000 20
076A:0000  AB 00 DE 0C FF 7F FF FF-02 29 00 00 50 00 00 01   .........)..P...
076A:0010  01 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0020  00                                                .
-
```

Name:	Pankhania Aanandi R.
Roll No:	IT081
Batch:	I1

Experiment 6

AIM: To study interfacing between C program and assembly language program.

1. Write a C program to convert Celsius to Fahrenheit where the functions "C2F" is assembly language function. Print the converted temperature in Fahrenheit from the C program.

Rules for Operands:

1. You have to initialize the Celsius_temperature variable with your roll no.
   E.g. IT020 so, tempc=20 (decimal number).
2. Your output screenshot should contain. (Look at the output screenshot)
   "Name: ……………"
   "Roll_no:……………"
   "C2f is defined in Assembly Program"
   "Temperature in Celsius………and temperature in Fahrenheit………."

Write your code here:

1. **C-program File (c2f.c)**

```
int tempc=81,tempf;
extern int c2f(int c);
void main()
{
clrscr();
printf("Name       : Aanandi Pankhania\n");
printf("Roll_no     : IT 081\n");
printf("C2f is defined in Assembly Program.\n");
tempf=c2f(tempc);
printf("Celsius     : %d\nFahrenheit  : %d \n",tempc,tempf);
getch();
}
```

2. **Assembly program File (c2f.asm)**

```
 _TEXT SEGMENT BYTE PUBLIC 'CODE'

DGROUP group _DATA, _BSS

          assume cs:_TEXT, ds:DGROUP, SS: DGROUP

_TEXT ends
```

```
_DATA segment word public 'DATA'

_DATA ends


_TEXT segment byte public 'CODE'

PUBLIC _c2f

_c2f PROC NEAR

        PUSH BP

        MOV BP,SP

        PUSH SI

        MOV AX, WORD PTR [BP + 4]

        MOV DX,9

        MUL DX

        MOV BX,5

        CWD

        IDIV BX

        MOV SI,AX

        ADD SI,32

        MOV AX,SI

        POP SI

        POP BP

        RET

_c2f ENDP

_TEXT ENDS


_BSS segment word public 'BSS'

EXTRN _tempf:WORD
```

```
_BSS ends


_DATA segment word public 'DATA'

EXTRN _tempc:WORD

_DATA ends

END
```

Compilation /Running and Debugging steps:

- Clearly mention each step. (For reference use my ppt or you can refer Experiment-6 from the lab manual)

  1. Create c2f.c file using TurboC++ and importing c2f1() from asm file.

  Compiling file until no error in file.

  2. Create c2f1.asm file using Notepad++ and write code for c2f1() which

  is used in c file.

  3. After creating asm module run following command in DOSBox after mounting

  the drive where tasm folder is stored and create obj file

  >tasm c2f1.asm

  4. Move the c2f1.obj file from tasm folder to TurboC3\bin.

  5. Launch TurboC++ and Go to the project menu and select open project.

  6. When dialog box appears, type c1.prj.

  7. Use the add item option in project menu and add c2f.c and c2f1.obj file.

  After this press done option of dialog box.

  8. Go to the option menu and select linker. In this menu go to the case sensitive

  link and press enter key to turn it off to avoid Upper/lower case disagreements

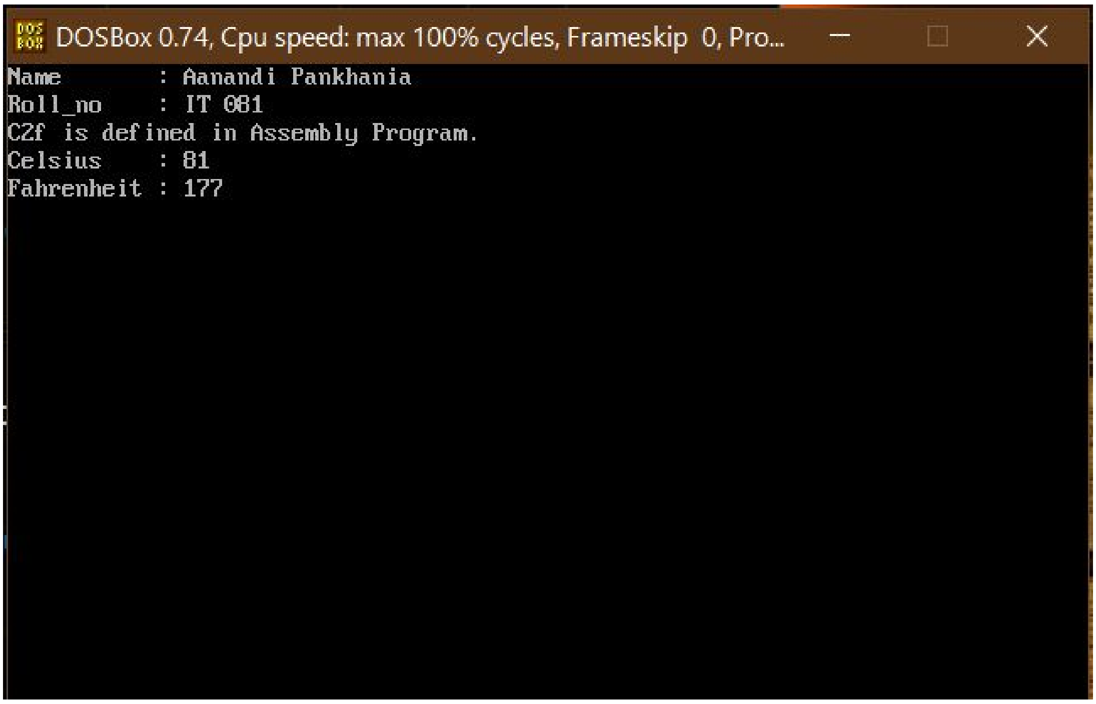
  between asm and c file.

  9. Go to compile menu, select build all and press enter key to combine c and asm

file and converted into obj file.

10.Go to run menu and select run to run the project.

Output: Screenshot of output. (Fonts should be clearly visible for your output.)


DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Pro...

```
Name        : Aanandi Pankhania
Roll_no     : IT 081
C2f is defined in Assembly Program.
Celsius     : 81
Fahrenheit  : 177
```

2. Write a C program to convert Celsius to Fahrenheit where the functions "C2F" and "Show" are assembly language functions. (Note: Name, Roll, and message you can print directly from C program but to display converted temperature define show() function in assembly language.)

Rules for Operands:

1. You have to initialize the Celsius_temperature variable with your roll no. E.g. IT020 so, tempc=20 (decimal number).
2. Your output screenshot should contain. (Look at the output screenshot)
   (You can directly write printf statements in C.)
   "Name:…………………"
   "Roll_no:…………………."
   "Both functions c2f and show are defined in Assembly Program"
   (Below msg should be printed from Assembly program Show() method.)
   "Temperature in Celsius………and temperature in Fahrenheit………."

Write your code here:

1. **C-program File (c2fshow.c)**

```
int tempc=81,tempf;
extern int c2f(int c);
extern int show(void);

void main()
```

```
{
    printf("Name        : Aanandi Pankhania\n");
    printf("Roll_no     : IT 081\n");

    printf("Both function C2F and Show are defined in Assembly program \n");
    tempf=c2f(tempc);
    show();
}
```

2. Assembly program File (c2fshow.asm)

```
 _TEXT segment byte public 'CODE'

    DGROUP group _DATA, _BSS

    assume cs:_TEXT, ds:DGROUP, SS: DGROUP

_TEXT ends



_DATA segment word public 'DATA'

    s@ db 'Celsius: %d Fahrenheit=%d' ; PRINTF STRING

_DATA ends

_TEXT segment byte public 'CODE'

    PUBLIC _c2f

    PUBLIC _show

    EXTRN _PRINTF:NEAR

    _c2f PROC NEAR

            PUSH BP

            MOV BP,SP

            PUSH SI

            MOV AX, WORD PTR [BP + 4]

            MOV DX,9

            MUL DX

            MOV BX,5

            CWD

            IDIV BX
            MOV SI,AX
            ADD SI,32
            MOV AX,SI
            POP SI
            POP BP
            RET
    _c2f ENDP
    _show PROC NEAR
            push word ptr DGROUP:_tempf
            push word ptr DGROUP:_tempc
            mov ax, offset DGROUP:s@
            push ax
            call near ptr _printf
            add sp, 6
            ret
    _show ENDP
_TEXT ENDS

_BSS segment word public 'BSS'
EXTRN _tempf:WORD



_BSS ends
_DATA segment word public 'DATA'
EXTRN _tempc:WORD

_DATA ends
```

END

Compilation /Running and Debugging steps:

- Clearly mention each step. (For reference use my ppt or you can refer Experiment-6 from the lab manual)

1. Create c2fshow.c file using TurboC++ and importing c2f1show() and show_data() from asm file. Compiling file until no error in file.

2. Create c2fshow.asm file using Notepad++ and write code for c2fshow() and show_data() which is used in c file. Also extern the printf fuction from the c library file for show_data function.

3. After creating asm module run following command in DOSBox after mounting the drive where tasm folder is stored and create obj file

```
>tasm c2fshow.asm
```

4. Move the c2fshow.obj file from tasm folder to TurboC3\bin.

5. Launch TurboC++ and Go to the project menu and select open project.

6. When dialog box appears, type c2.prj.

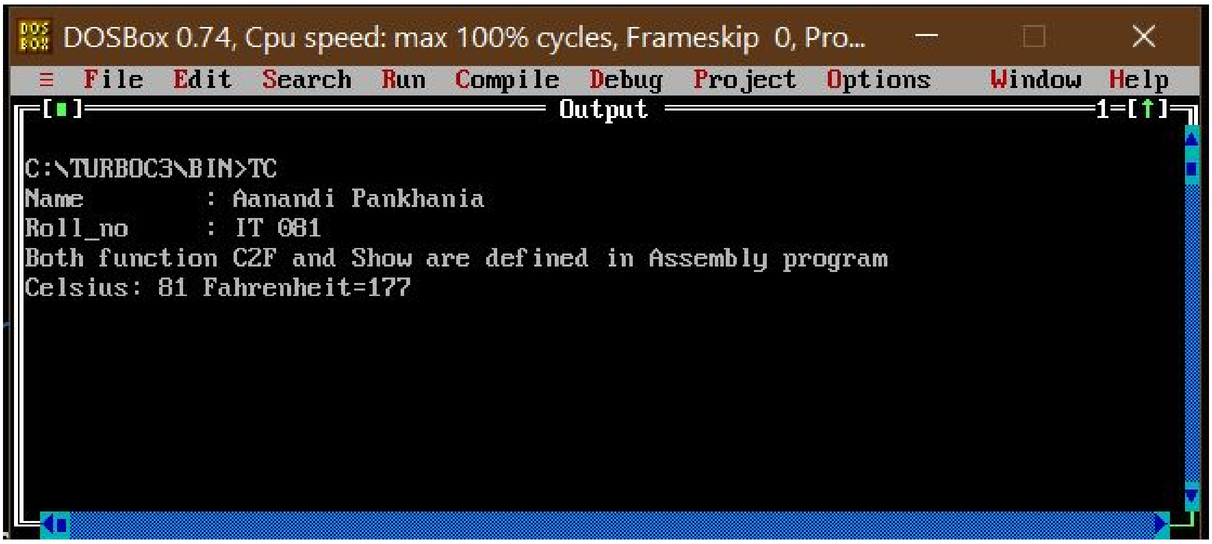
7. Use the add item option in project menu and add c2fshow.c and c2fshow.obj file. After this press done option of dialog box.

8. Go to the option menu and select linker. In this menu go to the case sensitive link and press enter key to turn it off to avoid Upper/lower case disagreements between asm and c file.

9. Go to compile menu, select build all and press enter key to combine c and asm file and converted into obj file.

10.Go to run menu and select run to run the project.

Output: Screenshot of output. (Fonts should be clearly visible for your output.)


DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Pro...
≡ File Edit Search Run Compile Debug Project Options Window Help
Output
1

```
C:\TURBOC3\BIN>TC
Name         : Aanandi Pankhania
Roll_no      : IT 081
Both function C2F and Show are defined in Assembly program
Celsius: 81 Fahrenheit=177
```

Name:	Pankhania Aanandi R.
Roll No:	IT081
Batch:	I1

Experiment - 7

AIM: Study of DOS and BIOS function calls

Using the following DOS function call to write programs.

1. AH = 01h / INT 21h - read character from standard input, with echo.
   Return: AL = character read.

2. AH = 02h / INT 21h - write character to standard output.
   Input: DL = character to write
   Return: AL = last character output

3. AH = 09h/INT 21h -write string to standard output
   Input: DS: DX -> offset address of the string and the string is terminated with '$'.
   Return: AL = 24h

4. AH = 0Ah /INT 21h -buffered input
   Entry: DS:DX -> buffer (reads from standard input)
   Return: buffer filled with user input.

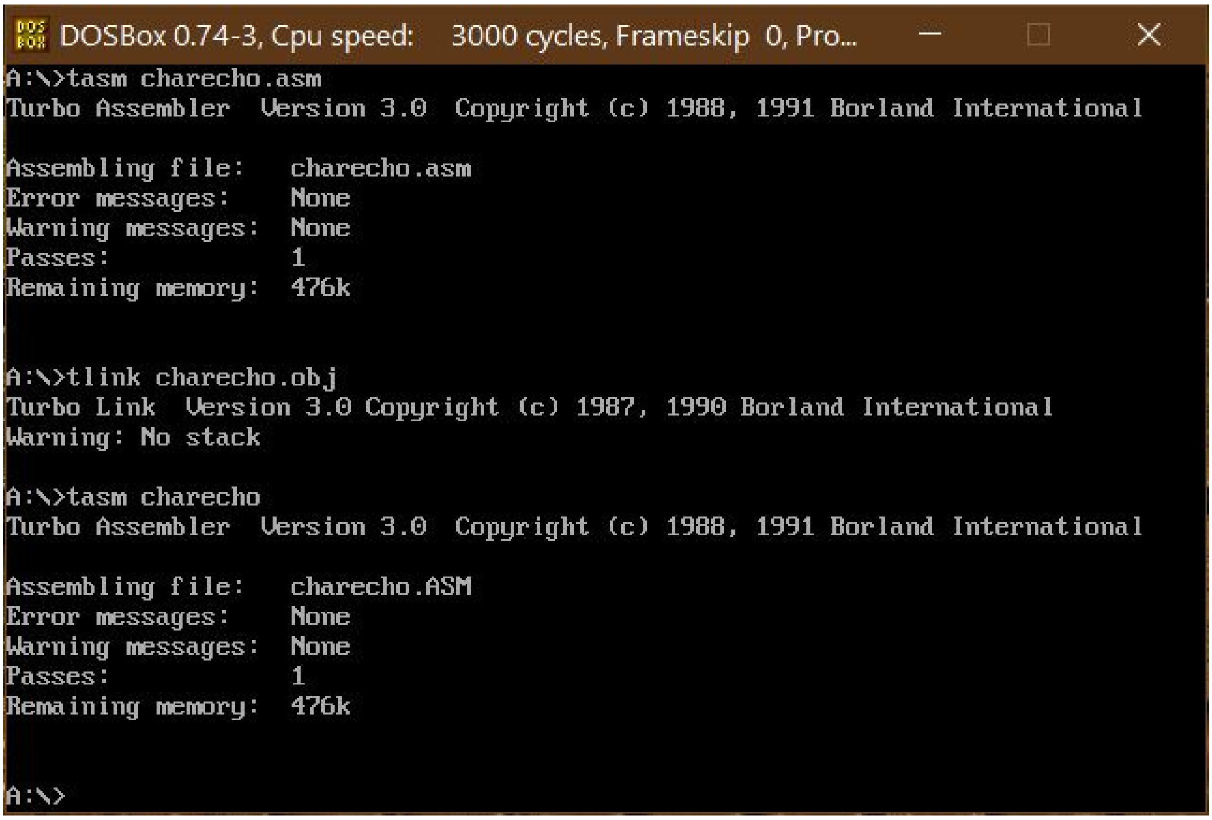
1. Write a program to take one character from the keyboard and echo on-screen.

Write your code here:

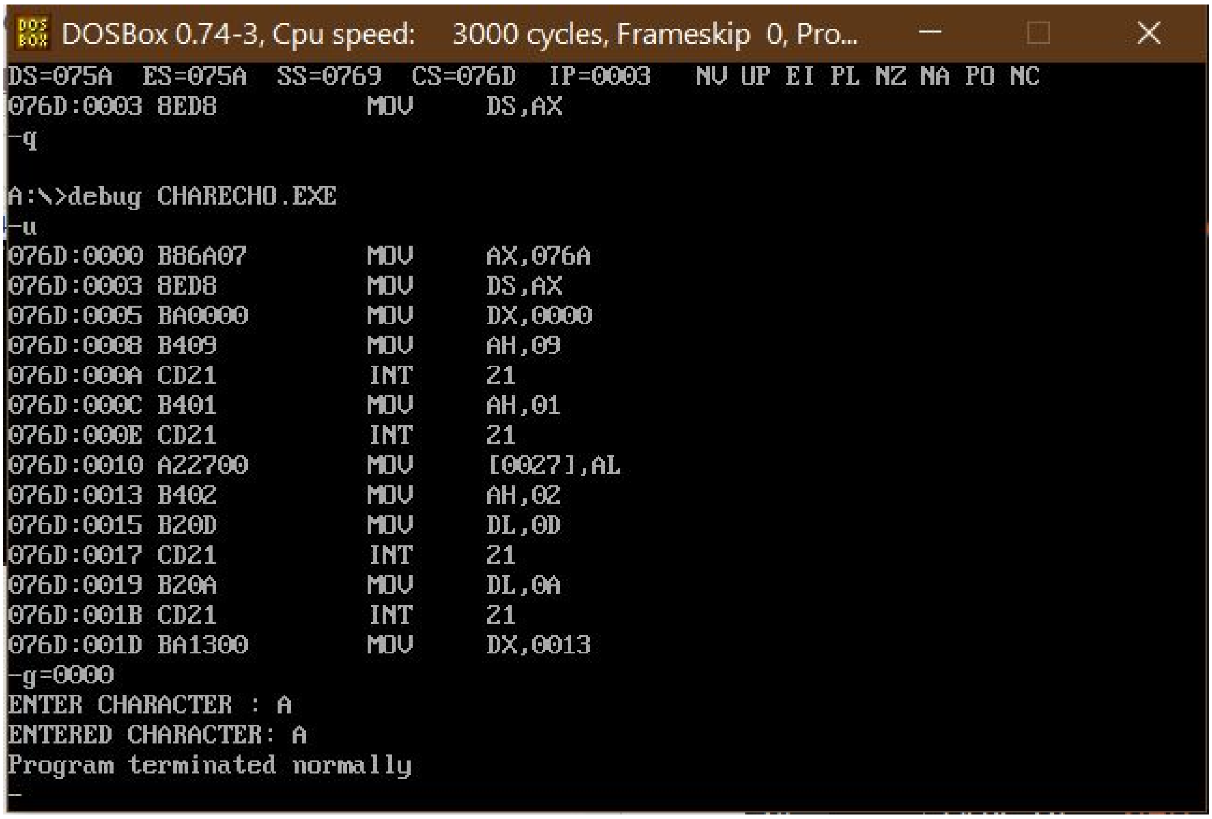
```
        DATA SEGMENT
MESSAGE DB "ENTER CHARACTER : $"
MESSAGE1 DB "ENTERED CHARACTER: $"
X DB ?
DATA ENDS


CODE SEGMENT
ASSUME DS: DATA, CS: CODE
START:
        MOV AX,DATA
        MOV DS,AX
        LEA DX,MESSAGE
        MOV AH, 9               ; Print message
        INT 21H
        MOV AH, 1            ; read a character
        INT 21H

    MOV X, AL            ; save input character into X

    MOV AH, 2            ; carriage return
    MOV DL, 0DH
    INT 21H

    MOV DL, 0AH          ; line feed
    INT 21H

        LEA DX,MESSAGE1
        MOV AH, 9            ; Print message1
        INT 21H

    MOV AH, 2            ; display the character stored in X
    MOV DL, X
    INT 21H

    MOV AH, 4CH          ; return control to DOS
    INT 21H
CODE ENDS
END START
```

Compilation /Running and Debugging steps:

(As given in the lab manual as an example of multiplication program on page no:5 of lab manual)



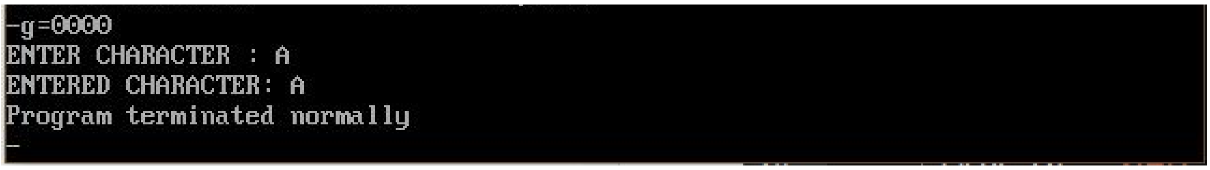
```
A:\>tasm charecho.asm
Turbo Assembler  Version 3.0  Copyright (c) 1988, 1991 Borland International

Assembling file:   charecho.asm
Error messages:    None
Warning messages:  None
Passes:            1
Remaining memory:  476k


A:\>tlink charecho.obj
Turbo Link  Version 3.0 Copyright (c) 1987, 1990 Borland International
Warning: No stack

A:\>tasm charecho
Turbo Assembler  Version 3.0  Copyright (c) 1988, 1991 Borland International

Assembling file:   charecho.ASM
Error messages:    None
Warning messages:  None
Passes:            1
Remaining memory:  476k


A:\>
```



```
DS=075A  ES=075A  SS=0769  CS=076D  IP=0003   NV UP EI PL NZ NA PO NC
076D:0003 8ED8          MOV     DS,AX
-q

A:\>debug CHARECHO.EXE
-u
076D:0000 B86A07        MOV     AX,076A
076D:0003 8ED8          MOV     DS,AX
076D:0005 BA0000        MOV     DX,0000
076D:0008 B409          MOV     AH,09
076D:000A CD21          INT     21
076D:000C B401          MOV     AH,01
076D:000E CD21          INT     21
076D:0010 A22700        MOV     [0027],AL
076D:0013 B402          MOV     AH,02
076D:0015 B20D          MOV     DL,0D
076D:0017 CD21          INT     21
076D:0019 B20A          MOV     DL,0A
076D:001B CD21          INT     21
076D:001D BA1300        MOV     DX,0013
-g=0000
ENTER CHARACTER : A
ENTERED CHARACTER: A
Program terminated normally
-
```

Output:

Screenshots of the output.



```
-g=0000
ENTER CHARACTER : A
ENTERED CHARACTER: A
Program terminated normally
-
```

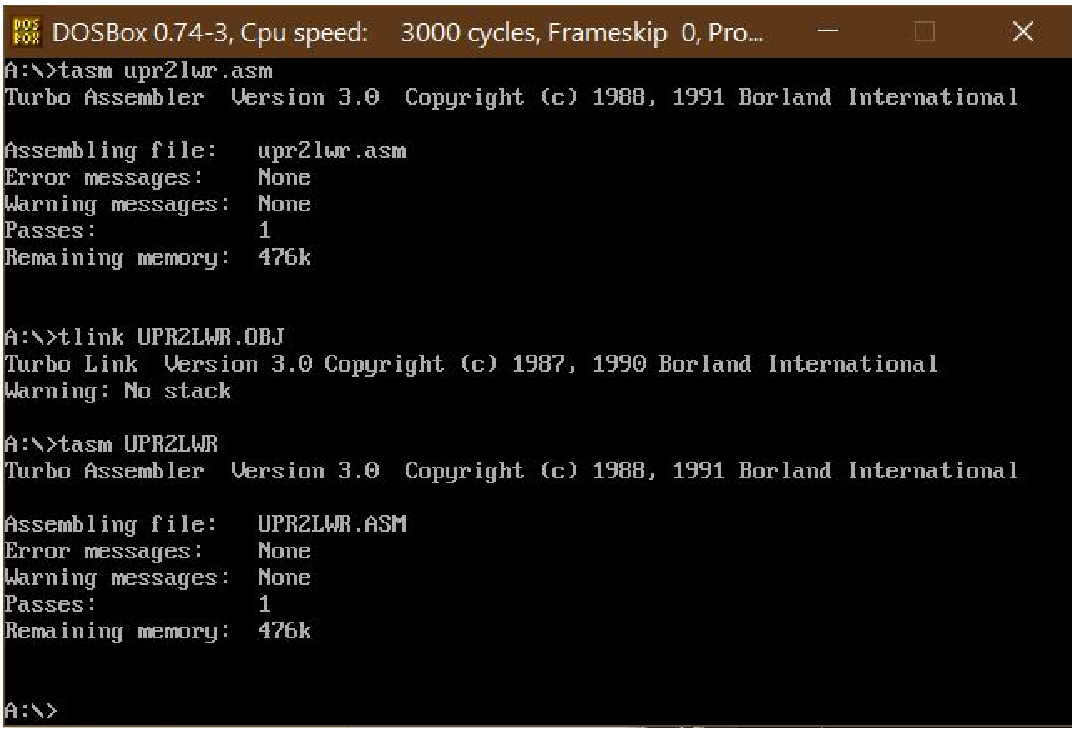
2. Write a program to take one character from key board and convert into lowercase.

Write your code here:

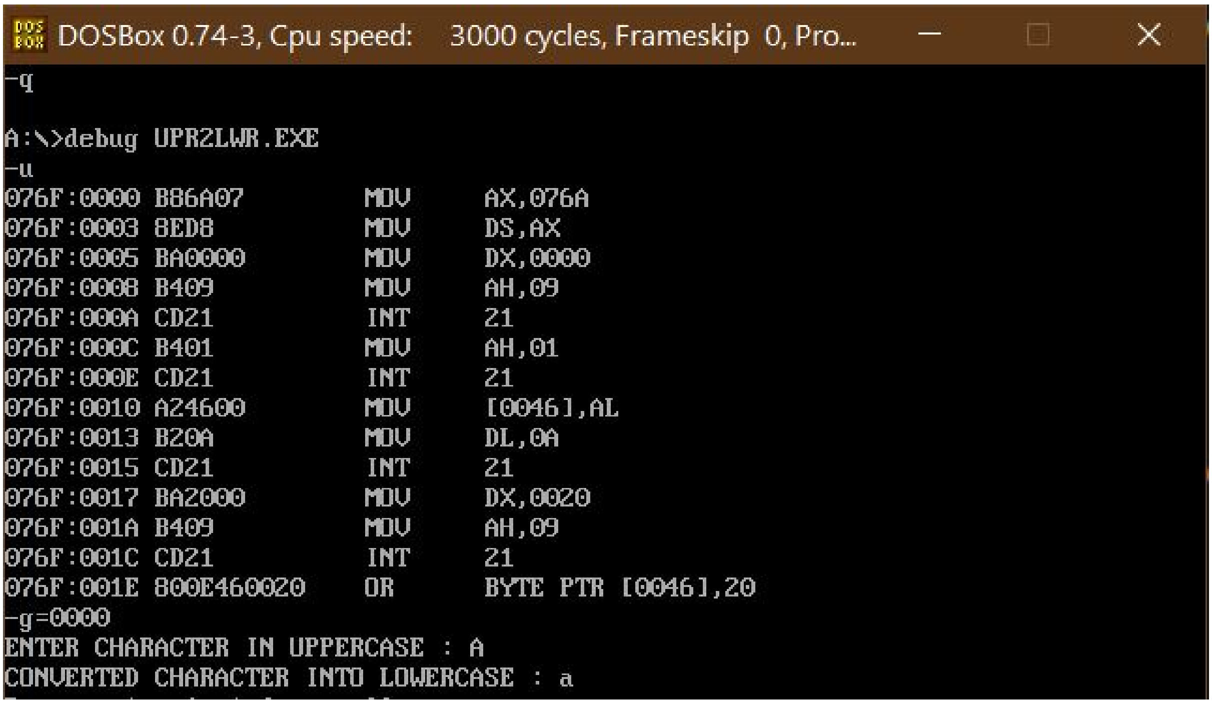
```
DATA SEGMENT
MESSAGE DB "ENTER CHARACTER IN UPPERCASE : $"
MESSAGE1 DB "CONVERTED CHARACTER INTO LOWERCASE : $"
X DB ?
DATA ENDS

CODE SEGMENT
ASSUME DS: DATA, CS: CODE
START:
        MOV AX,DATA
        MOV DS,AX
        LEA DX,MESSAGE
        MOV AH, 9               ; Print message
        INT 21H
        MOV AH, 1               ; read a character
        INT 21H

  MOV X, AL               ; save input character into X

        MOV DL, 0AH              ; line feed
  INT 21H

        LEA DX,MESSAGE1
        MOV AH, 9               ; Print message1
        INT 21H

        OR X, 20H

    MOV AH, 2               ; display the character stored in X
    MOV DL, X
    INT 21H

    MOV AH, 4CH              ; return control to DOS
    INT 21H
CODE ENDS
END START
```

Compilation /Running and Debugging steps:

(As given in lab manual as an example of multiplication program on page no:5 of lab manual)



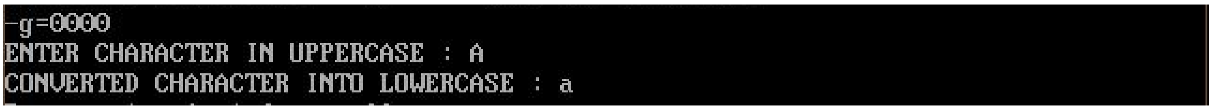
```
A:\>tasm upr2lwr.asm
Turbo Assembler  Version 3.0  Copyright (c) 1988, 1991 Borland International

Assembling file:   upr2lwr.asm
Error messages:    None
Warning messages:  None
Passes:            1
Remaining memory:  476k


A:\>tlink UPR2LWR.OBJ
Turbo Link  Version 3.0 Copyright (c) 1987, 1990 Borland International
Warning: No stack

A:\>tasm UPR2LWR
Turbo Assembler  Version 3.0  Copyright (c) 1988, 1991 Borland International

Assembling file:   UPR2LWR.ASM
Error messages:    None
Warning messages:  None
Passes:            1
Remaining memory:  476k


A:\>
```


DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Pro...

```
-q

A:\>debug UPR2LWR.EXE
-u
076F:0000 B86A07        MOV     AX,076A
076F:0003 8ED8          MOV     DS,AX
076F:0005 BA0000        MOV     DX,0000
076F:0008 B409          MOV     AH,09
076F:000A CD21          INT     21
076F:000C B401          MOV     AH,01
076F:000E CD21          INT     21
076F:0010 A24600        MOV     [0046],AL
076F:0013 B20A          MOV     DL,0A
076F:0015 CD21          INT     21
076F:0017 BA2000        MOV     DX,0020
076F:001A B409          MOV     AH,09
076F:001C CD21          INT     21
076F:001E 800E460020    OR      BYTE PTR [0046],20
-g=0000
ENTER CHARACTER IN UPPERCASE : A
CONVERTED CHARACTER INTO LOWERCASE : a
```

Output:

Screenshots of the output.



```
-g=0000
ENTER CHARACTER IN UPPERCASE : A
CONVERTED CHARACTER INTO LOWERCASE : a
```

3. **Write a program to get a string and convert this string from uppercase to lowercase.** **Rules for Operands:** Take your name as an input string and convert it.

Write your code here:

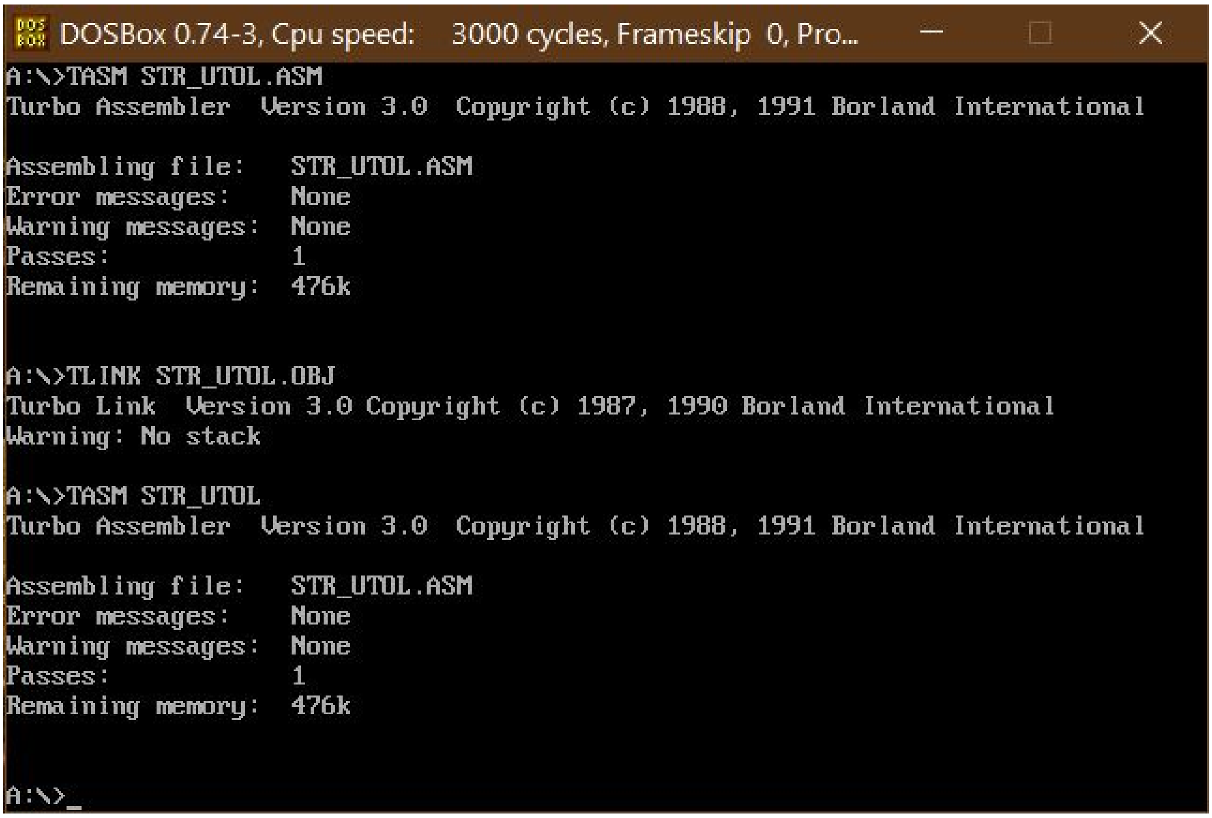
```
DATA SEGMENT

MESSAGE DB "ENTER STRING : $"

STR1 DB  255 DUP(?)

MESSAGE1 DB "STRING AFTER CONVERSION : $"

DATA ENDS


CODE SEGMENT

ASSUME DS: DATA, CS: CODE, ES: DATA

START:

        MOV AX,DATA

        MOV DS,AX
```

```
          MOV ES,AX

          MOV AH,09H
          LEA DX,MESSAGE                  ; Print message
          INT 21H

          LEA SI,STR1
           MOV AH,01H

;;Logic for conversion

READ:

   INT 21H
   MOV BL,AL
   CMP AL,0DH
   JE  PRNT
   XOR AL,20H
   MOV BYTE PTR [SI],AL
   INC SI
   JMP READ

PRNT:
   MOV AL,'$'
   MOV BYTE PTR [SI],AL
;;end logic : string is converted
          MOV AH,09H
          LEA DX,MESSAGE1               ; Print message1
          INT 21H

   LEA DX,STR1                          ; Print converted string
   MOV AH,09H
```

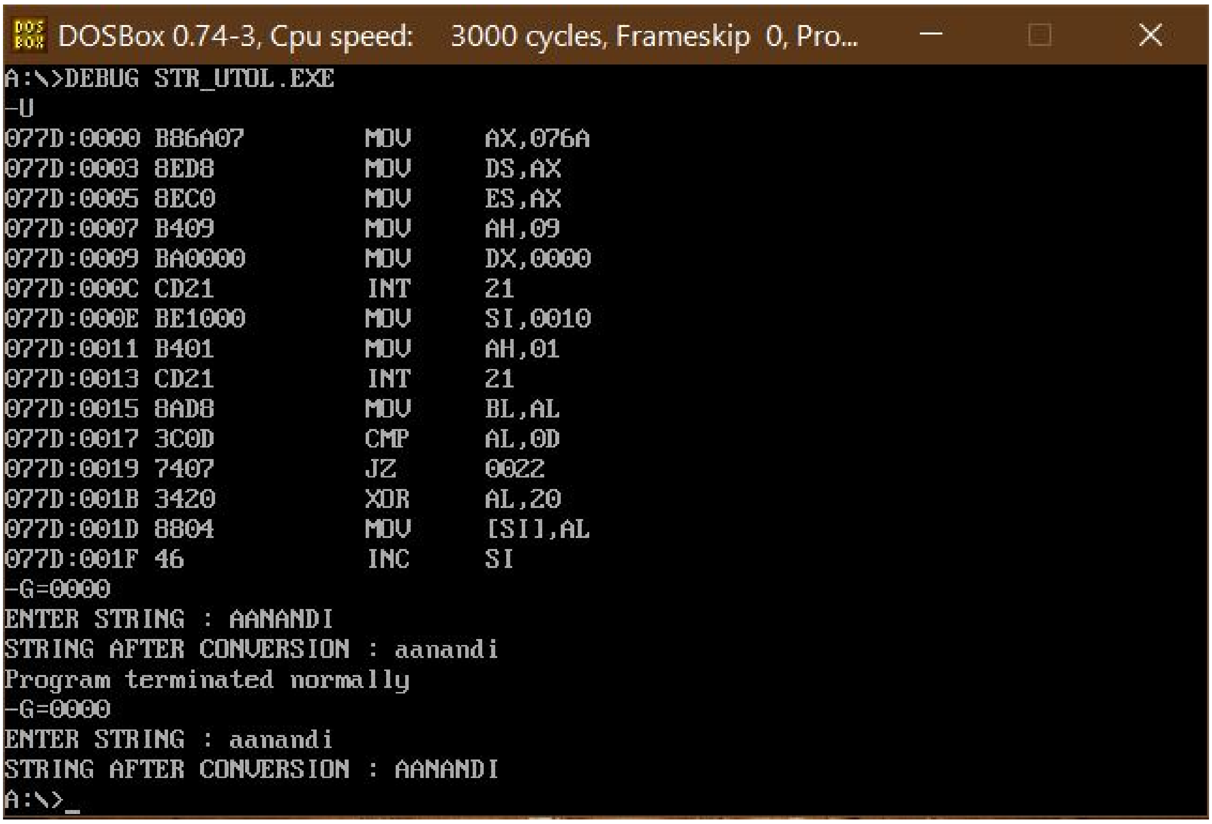
```
  INT 21H

        MOV AH, 4CH             ; return control to DOS
  INT 21H

CODE ENDS
END START
```

Compilation /Running and Debugging steps:

(As given in lab manual as an example of multiplication program on page no:5 of lab manual)


DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Pro...

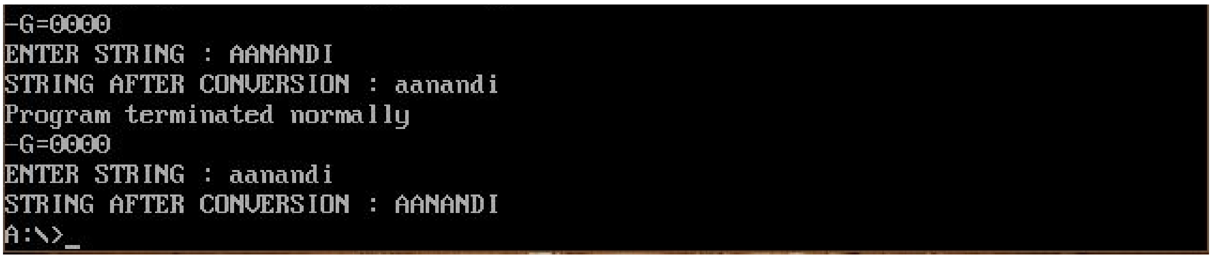
```
A:\>TASM STR_UTOL.ASM
Turbo Assembler  Version 3.0  Copyright (c) 1988, 1991 Borland International

Assembling file:   STR_UTOL.ASM
Error messages:    None
Warning messages:  None
Passes:            1
Remaining memory:  476k


A:\>TLINK STR_UTOL.OBJ
Turbo Link  Version 3.0 Copyright (c) 1987, 1990 Borland International
Warning: No stack

A:\>TASM STR_UTOL
Turbo Assembler  Version 3.0  Copyright (c) 1988, 1991 Borland International

Assembling file:   STR_UTOL.ASM
Error messages:    None
Warning messages:  None
Passes:            1
Remaining memory:  476k


A:\>_
```



```
A:\>DEBUG STR_UTOL.EXE
-U
077D:0000 B86A07        MOV     AX,076A
077D:0003 8ED8          MOV     DS,AX
077D:0005 8EC0          MOV     ES,AX
077D:0007 B409          MOV     AH,09
077D:0009 BA0000        MOV     DX,0000
077D:000C CD21          INT     21
077D:000E BE1000        MOV     SI,0010
077D:0011 B401          MOV     AH,01
077D:0013 CD21          INT     21
077D:0015 8AD8          MOV     BL,AL
077D:0017 3C0D          CMP     AL,0D
077D:0019 7407          JZ      0022
077D:001B 3420          XOR     AL,20
077D:001D 8804          MOV     [SI],AL
077D:001F 46            INC     SI
-G=0000
ENTER STRING : AANANDI
STRING AFTER CONVERSION : aanandi
Program terminated normally
-G=0000
ENTER STRING : aanandi
STRING AFTER CONVERSION : AANANDI
A:\>_
```

Output:

Screenshots of the output. (Both strings should be present in your screenshot)



```
-G=0000
ENTER STRING : AANANDI
STRING AFTER CONVERSION : aanandi
Program terminated normally
-G=0000
ENTER STRING : aanandi
STRING AFTER CONVERSION : AANANDI
A:\>_
```

4. Here is a string “Hello Sunil Welcome to 8086 Microprocessor”. Write an assembly Language program to convert the above string to “Hollo Sunil Wolcomo to 8086 Microprocossor” (Replace ‘e’ with ‘o’).

Rules for Operands: Take your name in place of “Sunil” and write the program.

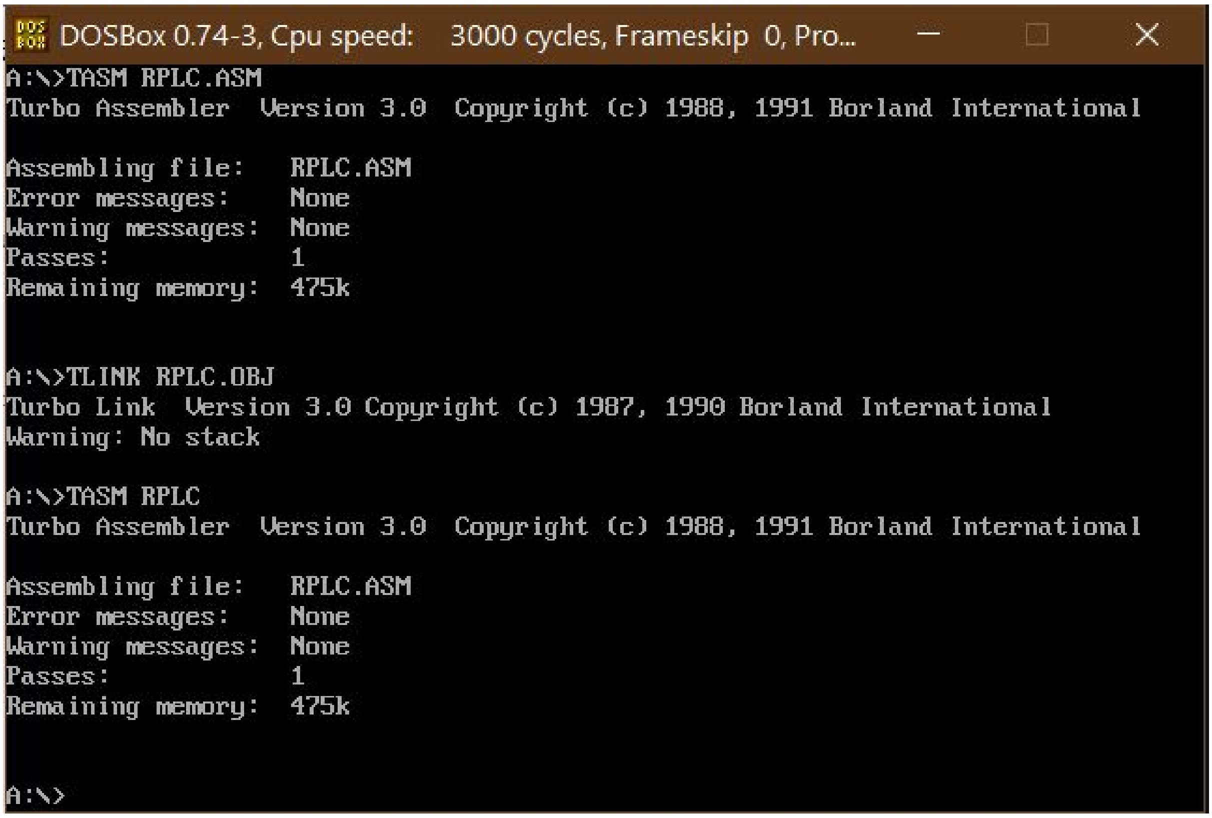
Write your code here:

```
DATA SEGMENT
    MESSAGE DB "HELLO AANANDI WELCOME TO 8086 MICROPROCESSOR $"
```

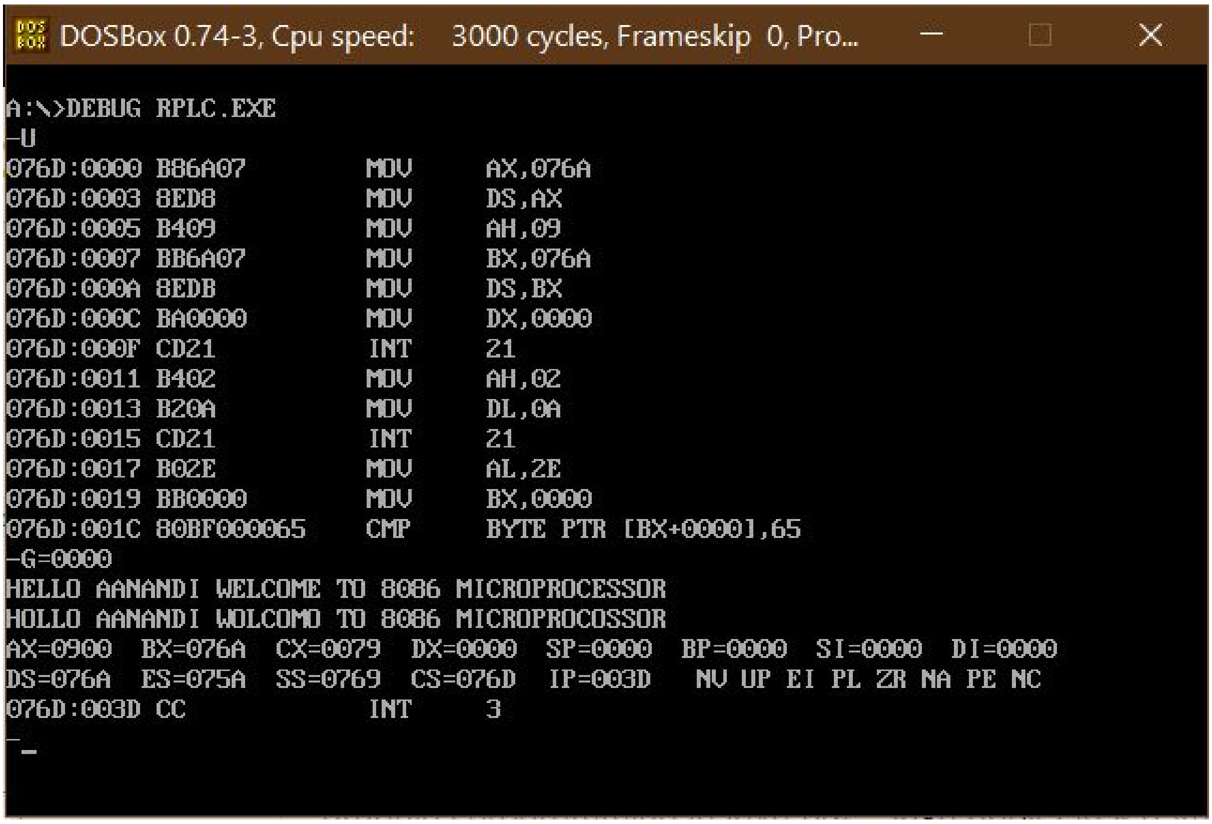
```
        LEN EQU $-MESSAGE
DATA ENDS

CODE SEGMENT
        ASSUME DS: DATA, CS: CODE

START:
        MOV AX,DATA
        MOV DS,AX

        MOV AH,09H
        MOV BX,SEG MESSAGE
        MOV DS,BX
        MOV DX,OFFSET MESSAGE
        INT 21H

        MOV AH,02H
        MOV DL,0AH
        INT 21H
        MOV AL,LEN
        LEA BX,MESSAGE
LOOP1:
        CMP MESSAGE[BX],65H
        JE RPLC
        CMP MESSAGE[BX],45H
        JE RPLC
        INC BX
        DEC AL
        CMP AL,00H
        JNE LOOP1
SHOW:
        MOV AH,09H
        MOV BX,SEG MESSAGE
        MOV DS,BX
        MOV DX,OFFSET MESSAGE
        INT 21H
EXIT:
        INT 3H
RPLC:
        ADD MESSAGE[BX],0AH
        DEC AL
        CMP AL,00H
        JNE LOOP1
CODE ENDS
END START
```

Compilation /Running and Debugging steps:

(As given in the lab manual as an example of multiplication program on page no:5 of lab manual)


DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Pro...

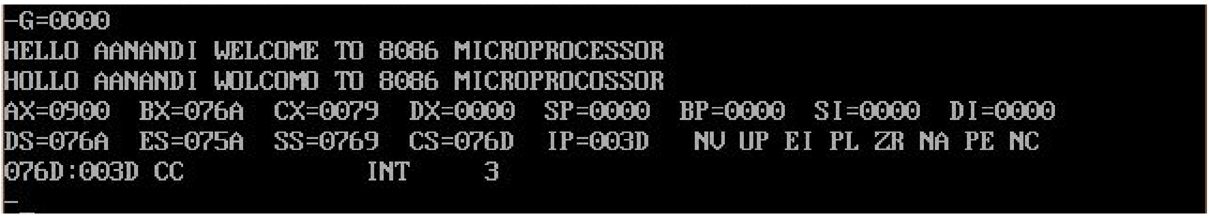
```
A:\>TASM RPLC.ASM
Turbo Assembler  Version 3.0  Copyright (c) 1988, 1991 Borland International

Assembling file:   RPLC.ASM
Error messages:    None
Warning messages:  None
Passes:            1
Remaining memory:  475k


A:\>TLINK RPLC.OBJ
Turbo Link  Version 3.0 Copyright (c) 1987, 1990 Borland International
Warning: No stack

A:\>TASM RPLC
Turbo Assembler  Version 3.0  Copyright (c) 1988, 1991 Borland International

Assembling file:   RPLC.ASM
Error messages:    None
Warning messages:  None
Passes:            1
Remaining memory:  475k


A:\>
```



```
A:\>DEBUG RPLC.EXE
-U
076D:0000 B86A07        MOV     AX,076A
076D:0003 8ED8          MOV     DS,AX
076D:0005 B409          MOV     AH,09
076D:0007 BB6A07        MOV     BX,076A
076D:000A 8EDB          MOV     DS,BX
076D:000C BA0000        MOV     DX,0000
076D:000F CD21          INT     21
076D:0011 B402          MOV     AH,02
076D:0013 B20A          MOV     DL,0A
076D:0015 CD21          INT     21
076D:0017 B02E          MOV     AL,2E
076D:0019 BB0000        MOV     BX,0000
076D:001C 80BF000065    CMP     BYTE PTR [BX+0000],65
-G=0000
HELLO AANANDI WELCOME TO 8086 MICROPROCESSOR
HOLLO AANANDI WOLCOMO TO 8086 MICROPROCOSSOR
AX=0900  BX=076A  CX=0079  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=076A  ES=075A  SS=0769  CS=076D  IP=003D   NV UP EI PL ZR NA PE NC
076D:003D CC            INT     3
-_
```

Output:

Screenshots of the output. (Both strings should be present in your screenshot)



```
-G=0000
HELLO AANANDI WELCOME TO 8086 MICROPROCESSOR
HOLLO AANANDI WOLCOMO TO 8086 MICROPROCOSSOR
AX=0900  BX=076A  CX=0079  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=076A  ES=075A  SS=0769  CS=076D  IP=003D   NV UP EI PL ZR NA PE NC
076D:003D CC            INT     3
-_
```

Name:	Pankhania Aanandi R.
Roll No:	IT081
Batch:	I1

Experiment - 8

AIM: Study of implementation of Recursion in assembly language.

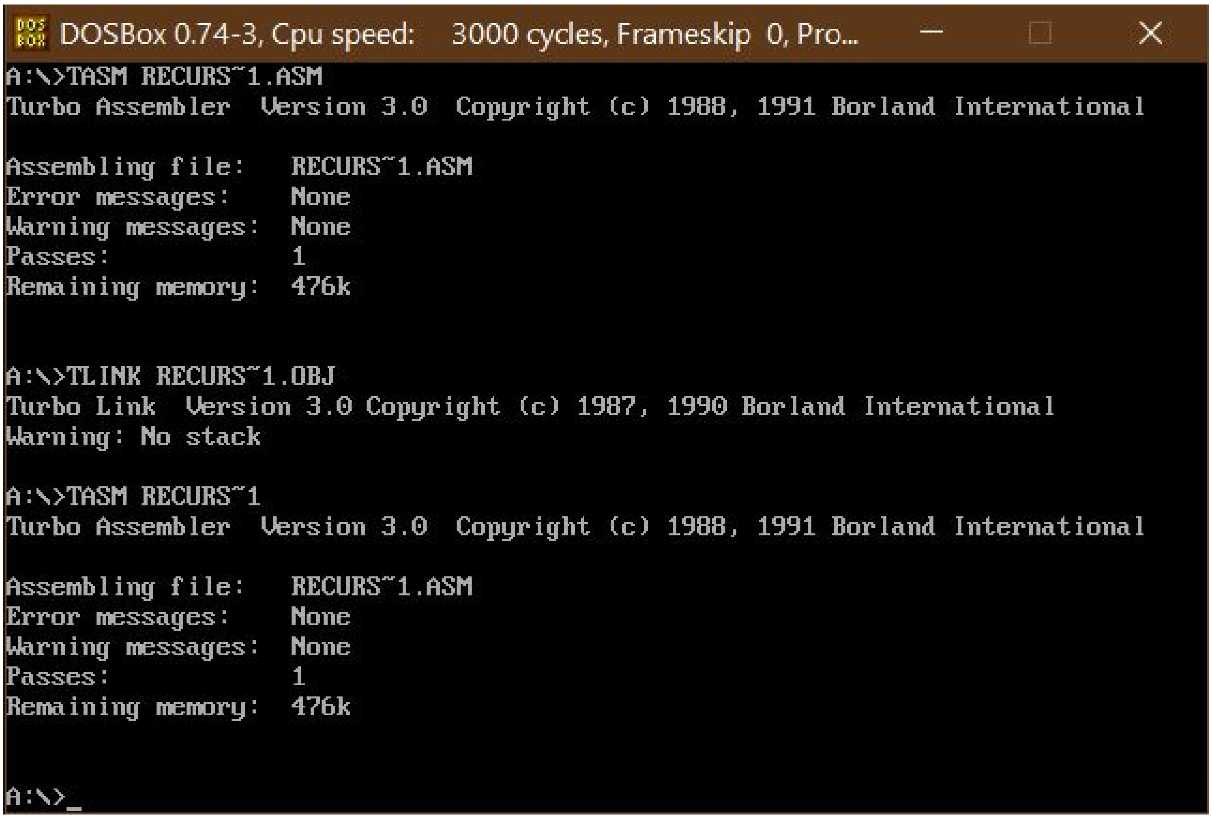
Program to find Factorial:

Write your code here:

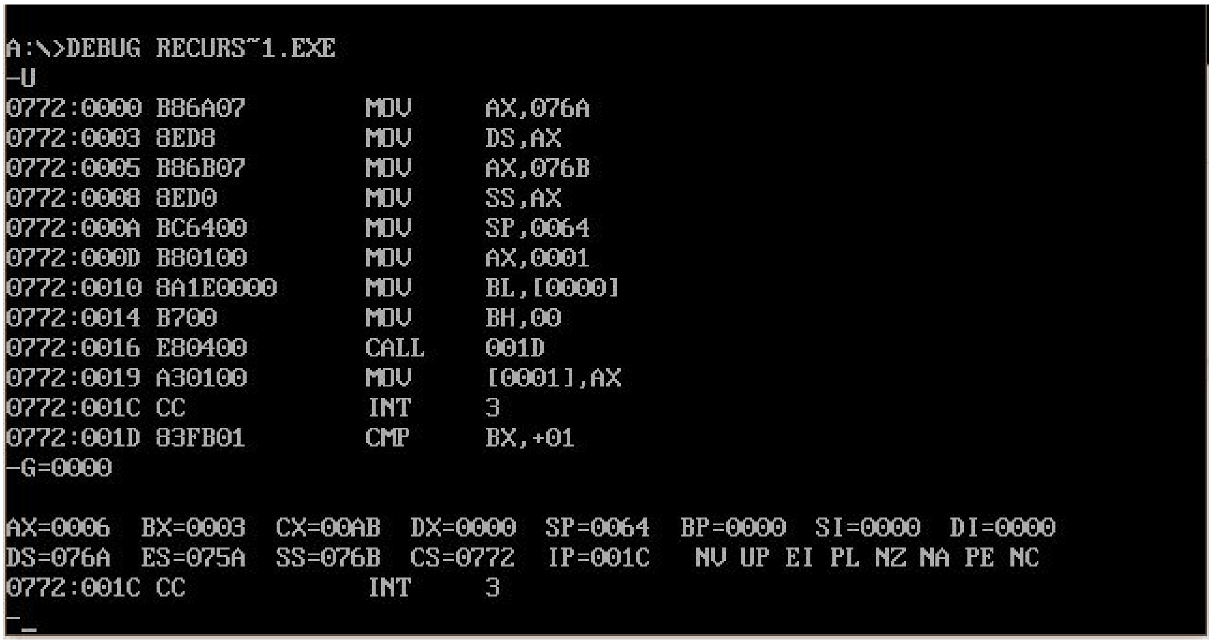
```
DATA_HERE SEGMENT
        N DB 03H        ;; FACTORIAL OF 3=3*2=6 //SHOULD BE DISPLAYED IN O/P
        FACT DW ?
DATA_HERE ENDS

STACK_HERE SEGMENT
        DW 50 DUP(0)
        STACK_TOP LABEL WORD
STACK_HERE ENDS

CODE_HERE SEGMENT
        ASSUME CS:CODE_HERE, DS:DATA_HERE, SS:STACK_HERE
START:
        MOV AX, DATA_HERE
        MOV DS, AX
        MOV AX, STACK_HERE
        MOV SS, AX
        MOV SP, OFFSET STACK_TOP
        MOV AX, 1
        MOV BL, N
        MOV BH, 0

        CALL FACTORIAL
        MOV FACT, AX
        INT 3H
        FACTORIAL PROC
        CMP BX, 1
        JE L1
        PUSH BX
        DEC BX
        CALL FACTORIAL
        POP BX
        MUL BX
L1:     RET
FACTORIAL ENDP
CODE_HERE ENDS
END START
```

Compilation /Running and Debugging steps:

(As given in lab manual as an example of multiplication program on page no:5 of lab manual)


DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Pro...

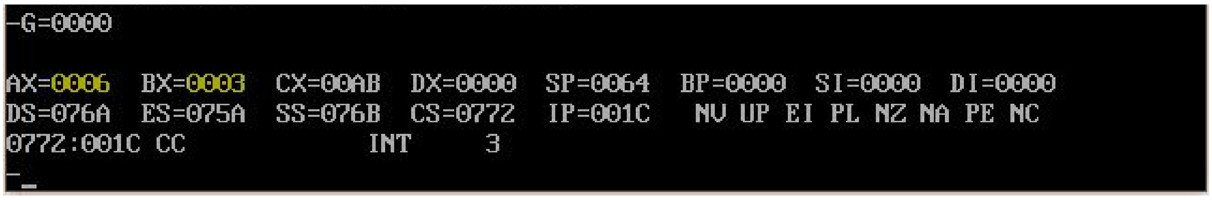
```
A:\>TASM RECURS~1.ASM
Turbo Assembler  Version 3.0  Copyright (c) 1988, 1991 Borland International

Assembling file:   RECURS~1.ASM
Error messages:    None
Warning messages:  None
Passes:            1
Remaining memory:  476k


A:\>TLINK RECURS~1.OBJ
Turbo Link  Version 3.0 Copyright (c) 1987, 1990 Borland International
Warning: No stack

A:\>TASM RECURS~1
Turbo Assembler  Version 3.0  Copyright (c) 1988, 1991 Borland International

Assembling file:   RECURS~1.ASM
Error messages:    None
Warning messages:  None
Passes:            1
Remaining memory:  476k


A:\>_
```



```
A:\>DEBUG RECURS~1.EXE
-U
0772:0000 B86A07        MOV     AX,076A
0772:0003 8ED8          MOV     DS,AX
0772:0005 B86B07        MOV     AX,076B
0772:0008 8ED0          MOV     SS,AX
0772:000A BC6400        MOV     SP,0064
0772:000D B80100        MOV     AX,0001
0772:0010 8A1E0000      MOV     BL,[0000]
0772:0014 B700          MOV     BH,00
0772:0016 E80400        CALL    001D
0772:0019 A30100        MOV     [0001],AX
0772:001C CC            INT     3
0772:001D 83FB01        CMP     BX,+01
-G=0000

AX=0006  BX=0003  CX=00AB  DX=0000  SP=0064  BP=0000  SI=0000  DI=0000
DS=076A  ES=075A  SS=076B  CS=0772  IP=001C   NV UP EI PL NZ NA PE NC
0772:001C CC            INT     3
-_
```

Output:

Screenshots of the output.



```
-G=0000

AX=0006  BX=0003  CX=00AB  DX=0000  SP=0064  BP=0000  SI=0000  DI=0000
DS=076A  ES=075A  SS=076B  CS=0772  IP=001C   NV UP EI PL NZ NA PE NC
0772:001C CC            INT     3
-_
```

Name:	Pankhania Aanandi R.
Roll No:	IT081
Batch	I1

Experiment 9

AIM: Study of various methods of passing parameters to a procedure

1. **Write an assembly language program to convert a 4-digit BCD number to binary. Use procedure and stack to pass parameters.**

Rules for Operands: You have to use your roll-no/registration no. as 4-digit BCD number.

E.g. IT025, so BCD input should be 0025H.

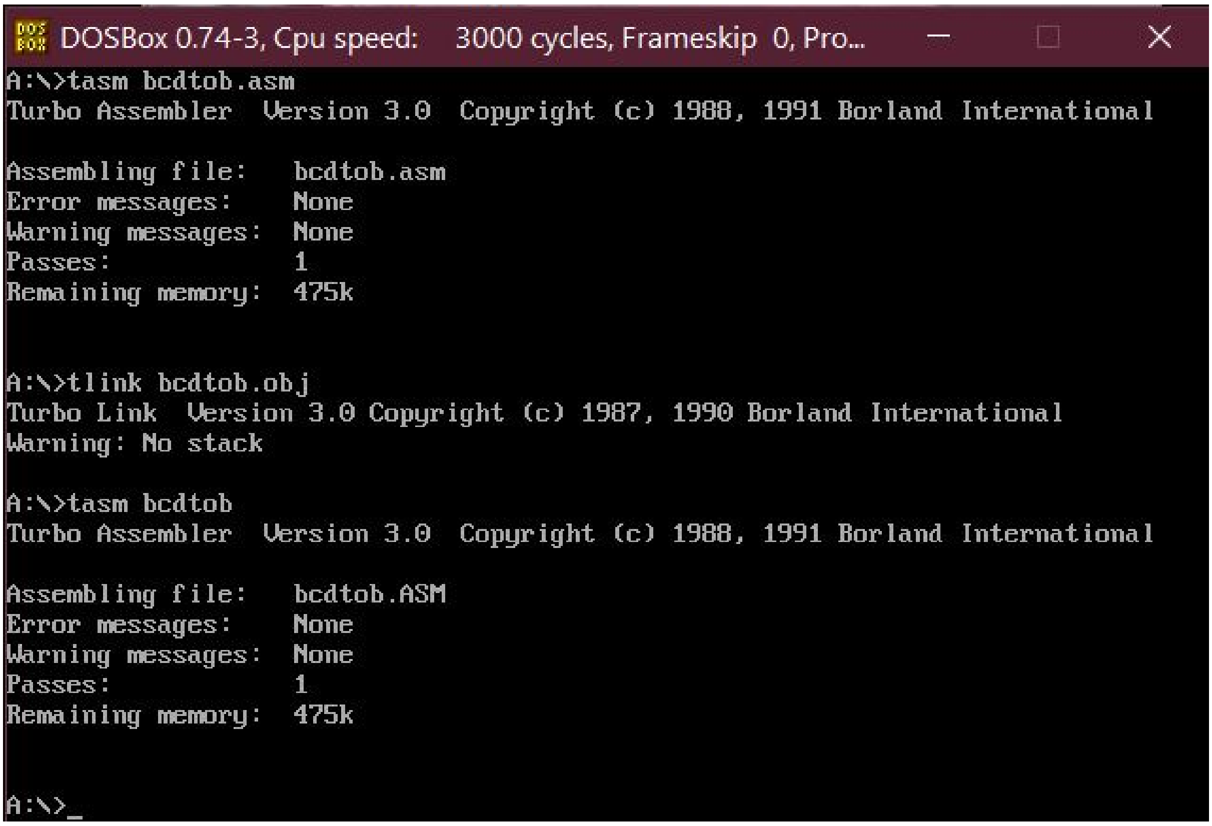
e.g. for repeater student ID=18ITUOS103, BCD input should be 0103h

Write your code here:

```
data_here SEGMENT
        bcd_n DW 0081H
        bin_n DW ?
data_here ENDS

stack_here SEGMENT STACK
        DW 50 DUP(0)
        Stack1 LABEL WORD
stack_here ENDS

code_here SEGMENT

        ASSUME CS:code_here, ES:data_here, DS:data_here, SS:stack_here
START : MOV AX,data_here
                MOV DS,AX
                MOV ES,AX
                MOV AX,stack_here
                MOV SS,AX
                LEA SP,Stack1

                MOV AX, bcd_n
                PUSH AX             ;store value in stack
                CALL CONVERT1                ;call procedure
                POP AX              ;store the result of procedure will be popped from stack
                MOV bin_n,AX          ;copy result in bin_n

                INT 3h

        CONVERT1 PROC NEAR

                PUSHF
                PUSH BX
                PUSH CX
                PUSH BP

                MOV BP,SP
                MOV AX,[BP+10]

                MOV BX,AX
                AND AX,000FH      ;by this and operation last digit will be stored at last position
                MOV BP,AX

                MOV AX,BX
                AND AX,00F0H          ;to store at second last position
                MOV CL,04H
                SHR AX,CL                 ;shift by 4 right position
                MOV SI,000AH
                MUL SI                               ;digit will be multiplied by 10
                MOV SI,AX
                MOV AX,BX
                AND AX,0F00H            ; to store at third from last position
                MOV CL,08H
                SHR AX,CL        ;shift by 8 right position
                MOV DI,0064H
                MUL DI           ;digit will be multiplied by 100
                MOV DI,AX

                MOV AX,BX
                AND AX,0F000H     ;to store at fourth from last position
                MOV CL,0CH
                SHR AX,CL         ;shift by 12 right position
                MOV CX,03E8H
                MUL CX            ;digit will be multiplied by 1000
                ADD AX,SI
                ADD AX,DI
                ADD AX,BP         ; add all digits
                MOV BP,SP
                MOV [BP+10], AX    ;storing result in stack

                POP BP
                POP CX
                POP BX
                POPF
                RET
        CONVERT1 ENDP

    code_here ENDS

    END START
```

Compilation /Running and Debugging steps:

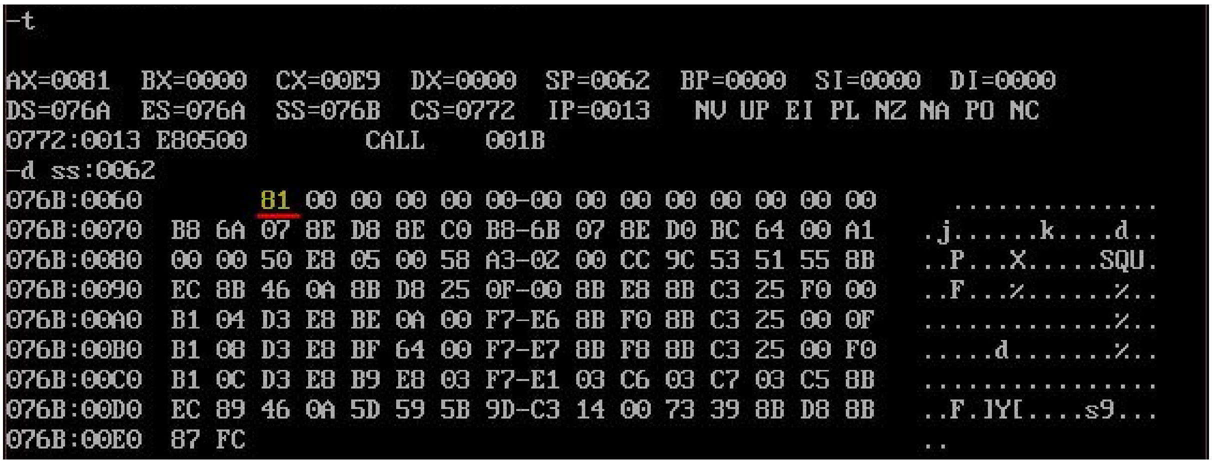
(As given in lab manual as an example of multiplication program on page no:5 of lab manual)



```
A:\>tasm bcdtob.asm
Turbo Assembler  Version 3.0  Copyright (c) 1988, 1991 Borland International

Assembling file:   bcdtob.asm
Error messages:    None
Warning messages:  None
Passes:            1
Remaining memory:  475k


A:\>tlink bcdtob.obj
Turbo Link  Version 3.0 Copyright (c) 1987, 1990 Borland International
Warning: No stack

A:\>tasm bcdtob
Turbo Assembler  Version 3.0  Copyright (c) 1988, 1991 Borland International

Assembling file:   bcdtob.ASM
Error messages:    None
Warning messages:  None
Passes:            1
Remaining memory:  475k


A:\>_
```

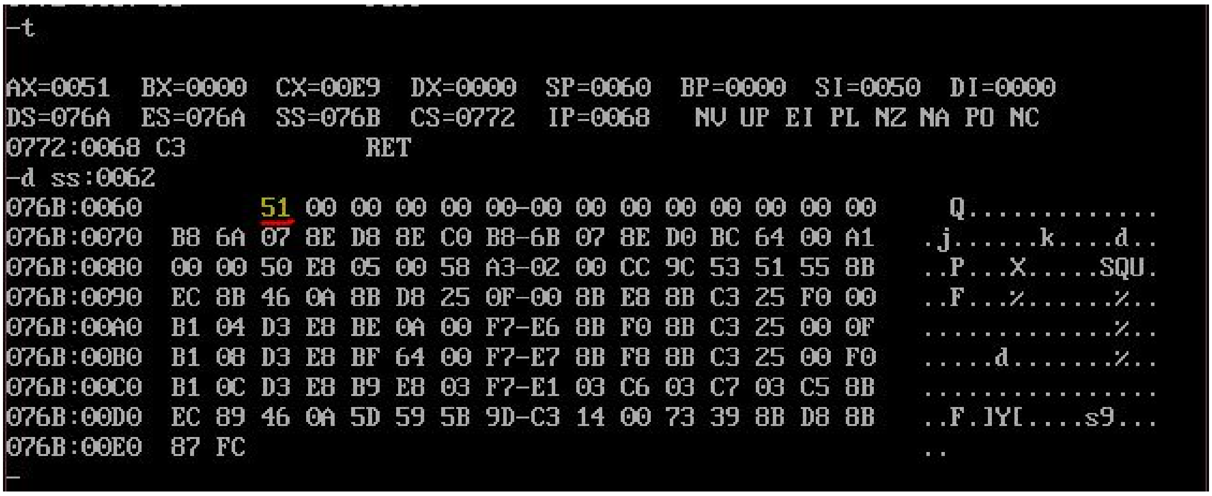
Output:

1. Put a screenshot of stack memory content (immediately after CALL instruction). Mark/highlight the parameter which you have passed from main program to procedure.



```
-t

AX=0081  BX=0000  CX=00E9  DX=0000  SP=0062  BP=0000  SI=0000  DI=0000
DS=076A  ES=076A  SS=076B  CS=0772  IP=0013   NV UP EI PL NZ NA PO NC
0772:0013 E80500        CALL    001B
-d ss:0062
076B:0060        81 00 00 00 00 00-00 00 00 00 00 00 00 00     ..............
076B:0070  B8 6A 07 8E D8 8E C0 B8-6B 07 8E D0 BC 64 00 A1   .j......k....d..
076B:0080  00 00 50 E8 05 00 58 A3-02 00 CC 9C 53 51 55 8B   ..P...X.....SQU.
076B:0090  EC 8B 46 0A 8B D8 25 0F-00 8B E8 8B C3 25 F0 00   ..F...%......%..
076B:00A0  B1 04 D3 E8 BE 0A 00 F7-E6 8B F0 8B C3 25 00 0F   .............%..
076B:00B0  B1 08 D3 E8 BF 64 00 F7-E7 8B F8 8B C3 25 00 F0   .....d.......%..
076B:00C0  B1 0C D3 E8 B9 E8 03 F7-E1 03 C6 03 C7 03 C5 8B   ................
076B:00D0  EC 89 46 0A 5D 59 5B 9D-C3 14 00 73 39 8B D8 8B   ..F.]Y[....s9...
076B:00E0  87 FC                                              ..
```

2. Put a screenshot of stack memory content (immediately after RET instruction). Mark/highlight the parameter which you have passed from procedure to program.



```
-t

AX=0051  BX=0000  CX=00E9  DX=0000  SP=0060  BP=0000  SI=0050  DI=0000
DS=076A  ES=076A  SS=076B  CS=0772  IP=0068   NV UP EI PL NZ NA PO NC
0772:0068 C3            RET
-d ss:0062
076B:0060        51 00 00 00 00 00-00 00 00 00 00 00 00    Q.............
076B:0070  B8 6A 07 8E D8 8E C0 B8-6B 07 8E D0 BC 64 00 A1  .j......k....d..
076B:0080  00 00 50 E8 05 00 58 A3-02 00 CC 9C 53 51 55 8B  ..P...X.....SQU.
076B:0090  EC 8B 46 0A 8B D8 25 0F-00 8B E8 8B C3 25 F0 00  ..F...%......%..
076B:00A0  B1 04 D3 E8 BE 0A 00 F7-E6 8B F0 8B C3 25 00 0F  .............%..
076B:00B0  B1 08 D3 E8 BF 64 00 F7-E7 8B F8 8B C3 25 00 F0  .....d.......%..
076B:00C0  B1 0C D3 E8 B9 E8 03 F7-E1 03 C6 03 C7 03 C5 8B  ................
076B:00D0  EC 89 46 0A 5D 59 5B 9D-C3 14 00 73 39 8B D8 8B  ..F.]Y[....s9...
076B:00E0  87 FC                                            ..
-
```

2. Write an assembly language program to count the number of 1's in the binary representation of 16-bit number using procedure and registers as parameter passing method.

Rules for Operands: You have to use binary representation of your roll-no/registration no. as 16-bit binary input.

E.g. IT025, so Binary input should be 0025H. i.e 0000 0000 0010 0101.

For repeater studentID=18ITUOS103, binary input should be 0103H. i.e. 0000 0001 0000 0011

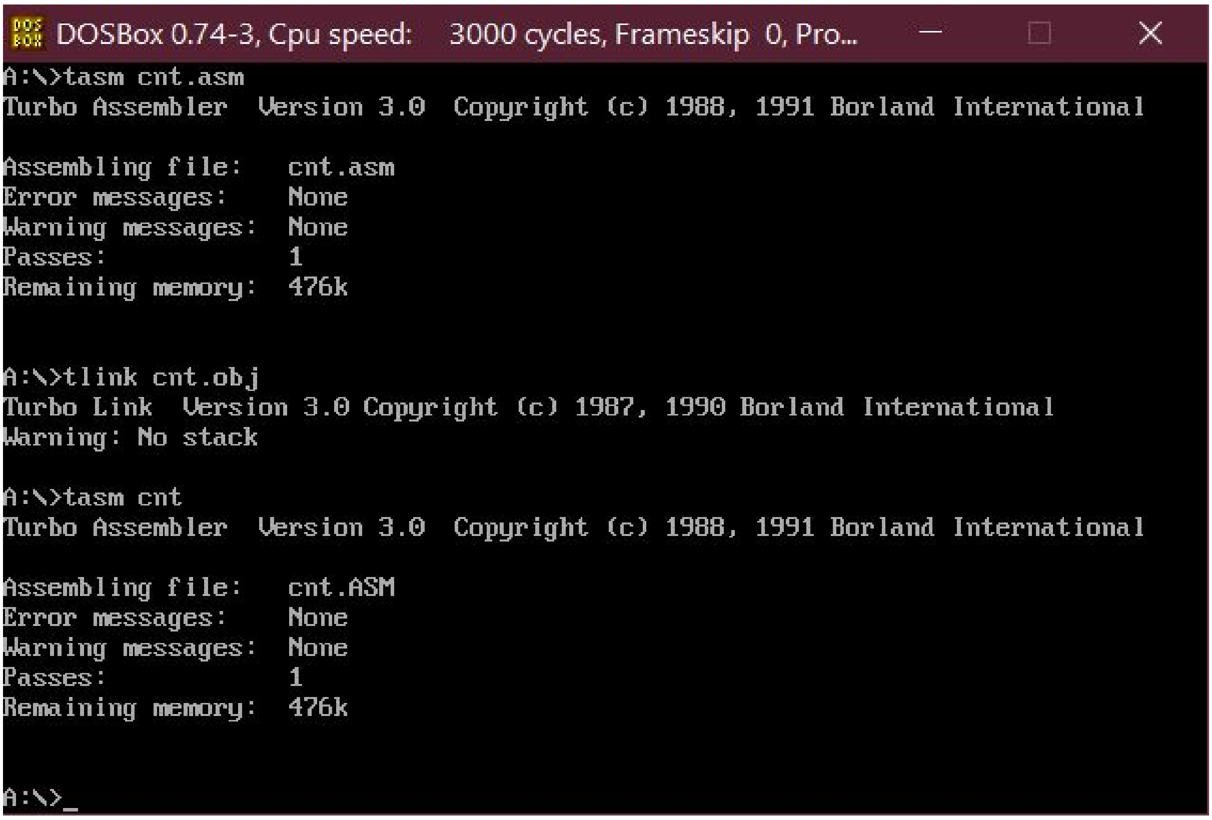
Write your code here:

```
data_here segment
        input1 DW 0081H ;0081 == 0000 0000 1000 0001
        ans DB ?
data_here ENDS

stack_here segment STACK
```

```
        DW 50 DUP(0)
        stack1 LABEL WORD
stack_here ENDS

code_here segment
ASSUME CS:code_here ,SS:stack_here ,DS:data_here
START : MOV AX,data_here
                MOV DS,AX
                MOV AX,stack_here
                MOV SS,AX
                LEA SP ,stack1

                MOV AX,input1
                CALL cnt1
                MOV ans,AL
                INT 3H

cnt1 PROC NEAR
                MOV BL,00H
                MOV CL ,10H
NEXT:   SHR AX,1        ;at a time shift reg AX
                JNC POS         ;if carry not generated jump to POS
                INC BL          ;if generated BL++

POS:    DEC CL
                JNZ NEXT
                MOV AL,BL
                RET
cnt1  ENDP
code_here ENDS
END START
```

Compilation /Running and Debugging steps:

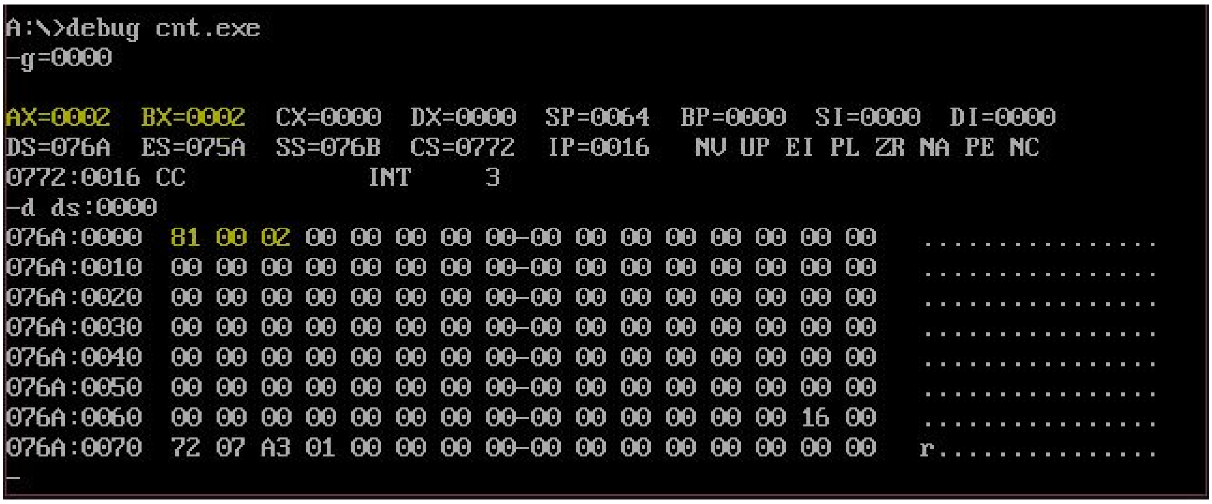
(As given in lab manual as an example of multiplication program on page no:5 of lab manual)



```
A:\>tasm cnt.asm
Turbo Assembler  Version 3.0  Copyright (c) 1988, 1991 Borland International

Assembling file:   cnt.asm
Error messages:    None
Warning messages:  None
Passes:            1
Remaining memory:  476k


A:\>tlink cnt.obj
Turbo Link  Version 3.0 Copyright (c) 1987, 1990 Borland International
Warning: No stack

A:\>tasm cnt
Turbo Assembler  Version 3.0  Copyright (c) 1988, 1991 Borland International

Assembling file:   cnt.ASM
Error messages:    None
Warning messages:  None
Passes:            1
Remaining memory:  476k


A:\>_
```

Output:

Screenshots of the memory/registers, which you are using to store your answer.



```
A:\>debug cnt.exe
-g=0000

AX=0002  BX=0002  CX=0000  DX=0000  SP=0064  BP=0000  SI=0000  DI=0000
DS=076A  ES=075A  SS=076B  CS=0772  IP=0016   NV UP EI PL ZR NA PE NC
0772:0016 CC            INT     3
-d ds:0000
076A:0000  81 00 02 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0010  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0020  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0030  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0040  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0050  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0060  00 00 00 00 00 00 00 00-00 00 00 00 00 00 16 00   ................
076A:0070  72 07 A3 01 00 00 00 00-00 00 00 00 00 00 00 00   r...............
-
```

Name:	Pankhania Aanandi R.
Roll No:	IT081
Batch:	I1

EXPERIMENT-10

Aim: Study of implementation of TSR:

Program 1: Active TSR using hot key combination.

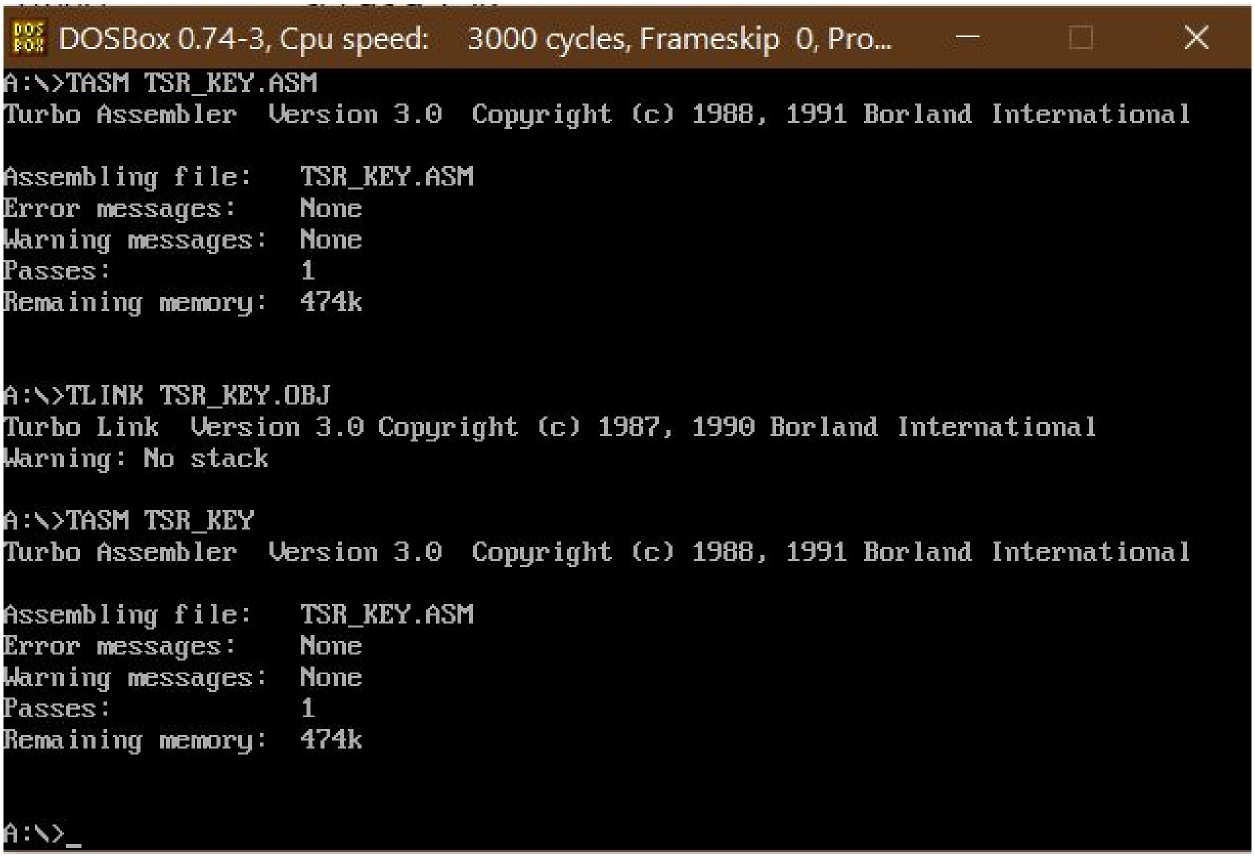
Code:

```
;a TSR program
;the ALT-K key combination is activated.
;A hot key is composed of a key scan code and a code found in memory location 0000:0417.
;The keyboard generates type9 interrupt whenever a key is typed. When intercepted with the
TSR handler, it reads the keyboard code directly from I/O port 60H, which returns the keyboard
scan code.


        .MODEL TINY
        .386
        .CODE
        .STARTUP
        JMP     INSTALL                 ;install  VEC9

HFLAG           DB      0               ;Hot-key detected
ADD9    DD      ?                       ;old vector 9 address

KEY     DB      25H                     ;scan code for K
HMASK           DB      8               ;alternate key mask
MKEY    DB      8                       ;alternate key
SCRN    DB      300 DUP (?)             ;screen buffer
MES1    DB      'TSR IS ACTIVE'


VEC9    PROC FAR                        ;keyboard intercept
```

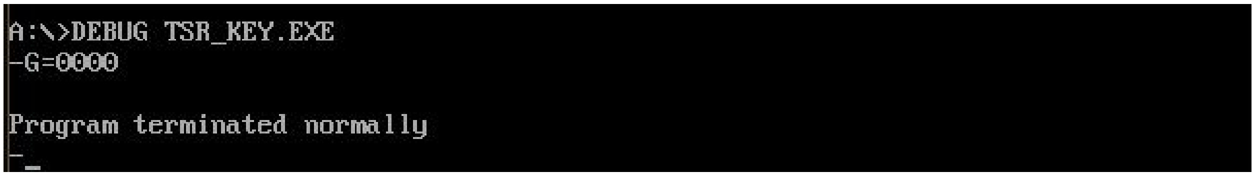
```
     STI                      ;enable interrupts
     PUSH AX                  ;save AX
     IN   AL,60H              ;get scan code
     CMP  AL,CS:KEY           ;test for K
     JNE  VEC91               ;no hot-key
     MOV  AX,0                ;address segment 0000
     PUSH DS                  ;save DS
     MOV  DS,AX
     MOV  AL,DS:[417H]        ;get shift/alternate data
     POP  DS
     AND  AL,CS:HMASK         ;isolate alternate key
     CMP  AL,CS:MKEY          ;test for alternate key
     JE   VEC93               ;if hot-key found
VEC91:

     POP  AX
     JMP  CS:ADD9             ;do normal interrupt
VEC93:                        ;if hot-key pressed
     CLI                      ;interrupts off
     IN   AL,61H              ;clear keyboard and
     OR   AL,80H              ;throw away hot key
     OUT  61H,AL
     AND  AL,7FH
     OUT  61H,AL
     MOV  AL,20H              ;reset keyboard interrupt
     OUT  20H,AL
     STI                      ;enable interrupts
     MOV  CS:HFLAG,1          ;indicate hot-key pressed
   push cx

   push di
   push si
   push ds
   push es
   cld
   mov ax,cs
   mov es,ax
   mov ax,0b800h
   mov ds,ax
   mov cx,160
   mov di,offset scrn
   mov si,0
   rep movsb
   push ds
```

```
      push es
      pop ds
      pop es
      mov di,80
      mov si,offset mes1
      mov ah,0fh
      mov cx,13
vec95: lodsb
     stosw
     loop vec95
     pop es
     pop ds
     pop si
     pop di

     pop cx
     POP     AX
        IRET

VEC9    ENDP


INSTALL:                                ;install VEC9

        MOV     AX,CS                   ;load DS
        MOV     DS,AX


        MOV     AX,3509H                ;get current vector 9
        INT     21H                     ;and save it
        MOV     WORD PTR ADD9,BX
        MOV     WORD PTR ADD9+2,ES

        MOV     AX,2509H
        MOV     DX,OFFSET VEC9          ;address interrupt procedure
        INT     21H                     ;install vector 9

        MOV     DX,OFFSET INSTALL               ;find paragraphs
        SHR     DX,4
        INC     DX

        MOV     AX,3100H                ;set as a TSR
        INT     21H
        END
```

Compilation:

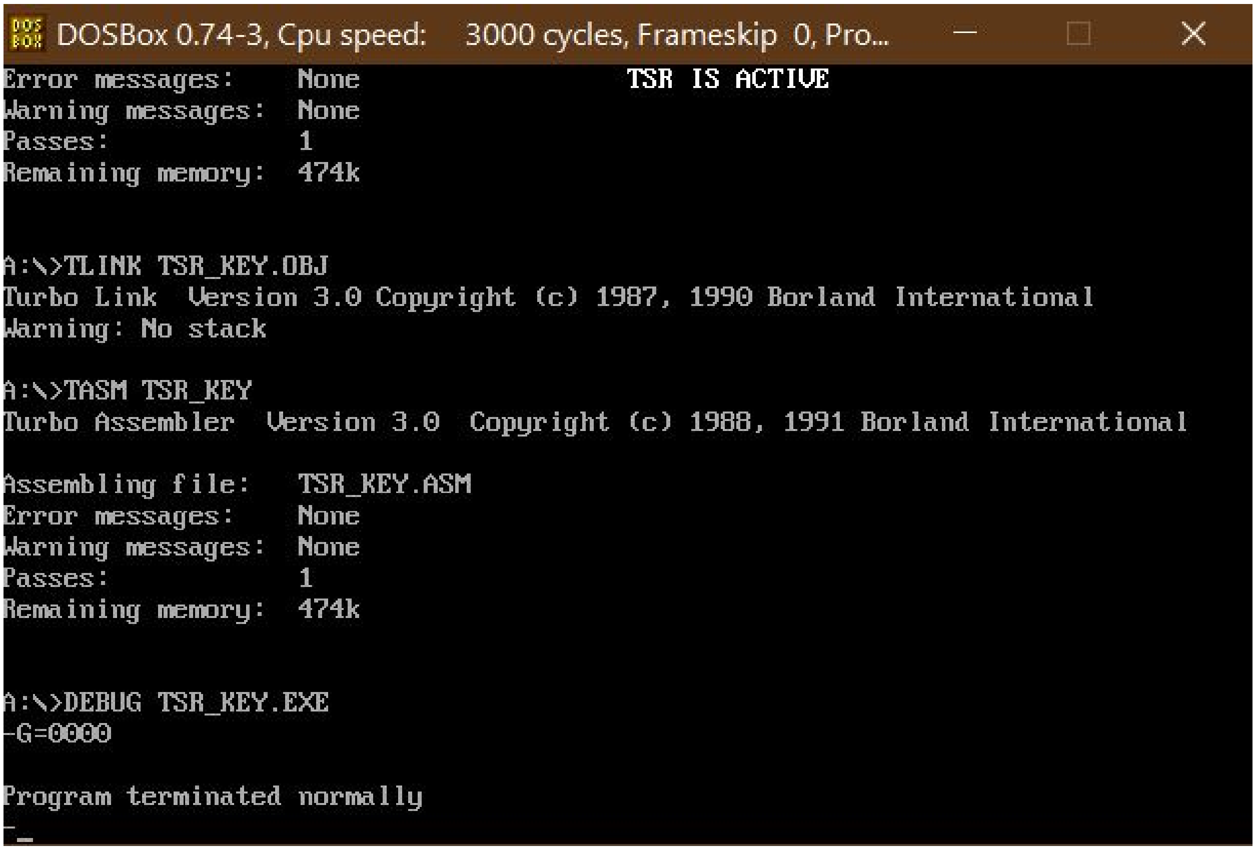

DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Pro...

```
A:\>TASM TSR_KEY.ASM
Turbo Assembler  Version 3.0  Copyright (c) 1988, 1991 Borland International

Assembling file:   TSR_KEY.ASM
Error messages:    None
Warning messages:  None
Passes:            1
Remaining memory:  474k


A:\>TLINK TSR_KEY.OBJ
Turbo Link  Version 3.0 Copyright (c) 1987, 1990 Borland International
Warning: No stack

A:\>TASM TSR_KEY
Turbo Assembler  Version 3.0  Copyright (c) 1988, 1991 Borland International

Assembling file:   TSR_KEY.ASM
Error messages:    None
Warning messages:  None
Passes:            1
Remaining memory:  474k


A:\>_
```



```
A:\>DEBUG TSR_KEY.EXE
-G=0000

Program terminated normally
-_
```

Output: //After Pressing Ctrl+Alt



```
Error messages:    None                    TSR IS ACTIVE
Warning messages:  None
Passes:            1
Remaining memory:  474k


A:\>TLINK TSR_KEY.OBJ
Turbo Link  Version 3.0 Copyright (c) 1987, 1990 Borland International
Warning: No stack

A:\>TASM TSR_KEY
Turbo Assembler  Version 3.0  Copyright (c) 1988, 1991 Borland International

Assembling file:   TSR_KEY.ASM
Error messages:    None
Warning messages:  None
Passes:            1
Remaining memory:  474k


A:\>DEBUG TSR_KEY.EXE
-G=0000

Program terminated normally
-_
```

Program 2: Example of Active and Passive TSR - Screensaver.

Code:

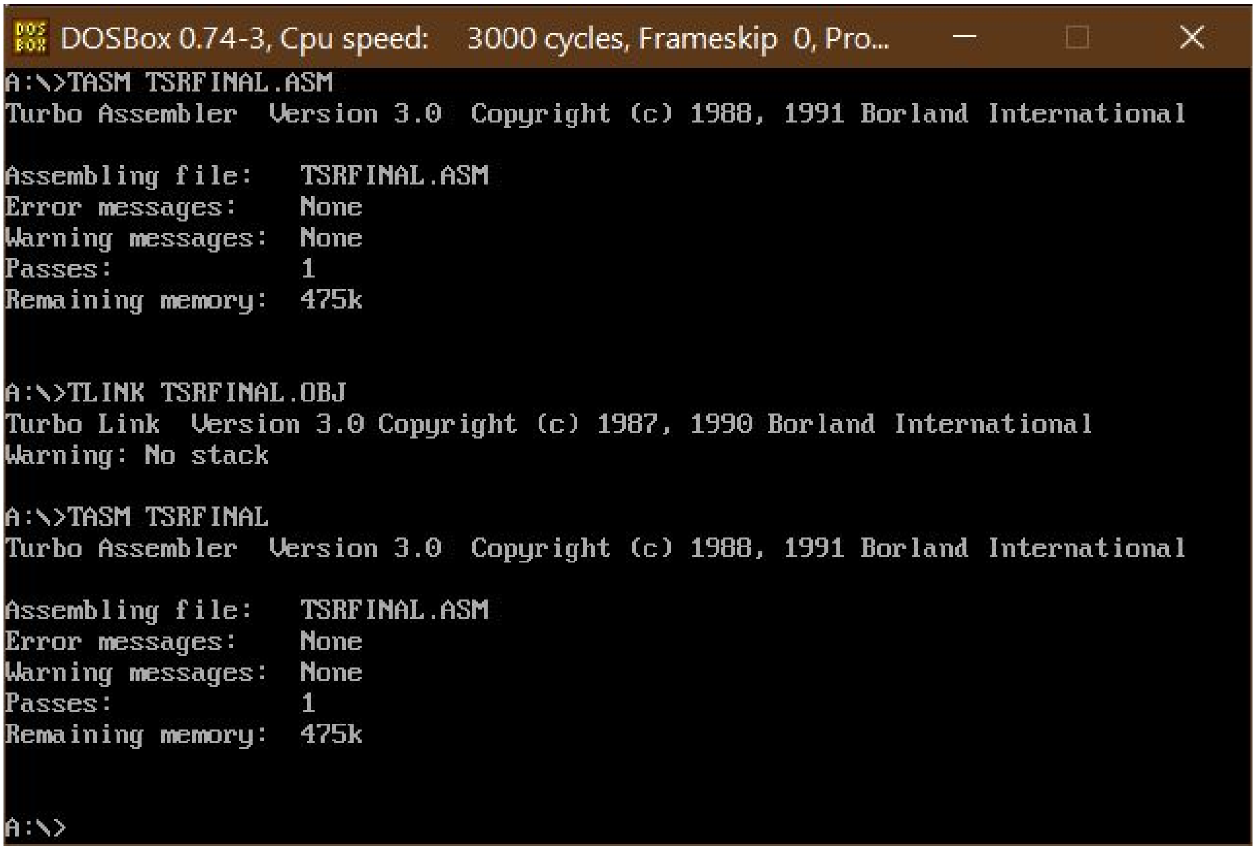
```
;Write a TSR program in 8086 ALP to implement Screen Saver. Screen Saver should get
;activated if the keyboard is idle for 7 seconds. Access the video RAM directly in your routine.
;http://books.google.co.in/books?id=zWrZY1OgTPsC&pg=PA283&lpg=PA283&dq=tsr+program;
+in+8086+with+hot+key+combination&source=bl&ots=9A_74oJXRL&sig=iqn5tQUedewU44M8
YPnDPxMP6bk&hl=en&sa=X&ei=F-b6U_zHII2jugTR84HABw&ved=0CBwQ6AEwAA#v=onepag
e&q=tsr%20program%20in%208086%20with%20hot%20key%20combination&f=false
```

```
CODE SEGMENT
        ASSUME CS:CODE,DS:CODE,ES:CODE
        ORG 100H
START : JMP BEGIN
        TIMER_IP DW ?
        TIMER_CS DW ?
        KB_IP DW ?
        KB_CS DW ?
        FLAG DB 0
        CNT DB 180
        BUFFER DW 2000 DUP(0)
TIMER:
        PUSH AX
        PUSH BX
        PUSH CX
        PUSH DX
        PUSH SI
        PUSH DI
        PUSH DS
        PUSH ES

        MOV AX,CS
        MOV DS,AX
        MOV ES,AX

        CMP FLAG,00H
        JNE TIMER_END
        DEC CNT
        JNE TIMER_END

        CLD
        MOV AX,0B800H
        MOV DS,AX
        MOV SI,0000H
        MOV DI,OFFSET BUFFER
        MOV CX,2000
        REP MOVSW

        MOV AX,0B800H
        MOV ES,AX
        MOV DI,0000H
        MOV AL,48
        MOV AH,89
```

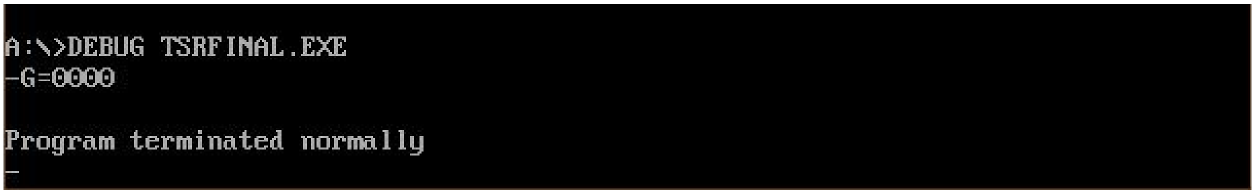
```
    MOV CX,2000
    REP STOSW

    MOV CS:FLAG,01H
TIMER_END:
    POP ES
    POP DS
    POP DI
    POP SI
    POP DX
    POP CX
    POP BX
    POP AX
JMP DWORD PTR CS:TIMER_IP
KB:
    PUSH AX
    PUSH BX
    PUSH CX
    PUSH DX
    PUSH SI
    PUSH DI
    PUSH DS
    PUSH ES

    MOV AX,CS
    MOV DS,AX
    MOV ES,AX

    MOV CNT,180
    CMP FLAG,01
    JNE KB_END

    CLD
    MOV AX,0B800H
    MOV ES,AX
    MOV SI,OFFSET BUFFER
    MOV DI,0000H
    MOV CX,2000
    REP MOVSW

    MOV FLAG,00H
KB_END :
    POP ES
    POP DS
```

```
        POP DI
        POP SI
        POP DX
        POP CX
        POP BX
        POP AX
JMP DWORD PTR CS:KB_IP

BEGIN:
        MOV AX,CS
        MOV DS,AX
        MOV ES,AX

        MOV AH,35H
        MOV AL,08H
        INT 21H

        MOV TIMER_IP,BX
        MOV TIMER_CS,ES

        MOV AH,35H
        MOV AL,09H
        INT 21H

        MOV KB_IP,BX
        MOV KB_CS,ES

        MOV AH,25H
        MOV AL,08H
        MOV DX,OFFSET TIMER
        INT 21H
        MOV AH,25H
        MOV AL,09H
        MOV DX,OFFSET KB
        INT 21H
        MOV AH,31H
        MOV DX,OFFSET BEGIN
        MOV CL,04H
        SHR DX,CL
        INC DX
        INT 21H

CODE ENDS
END START
```

Compilation and Debugging:

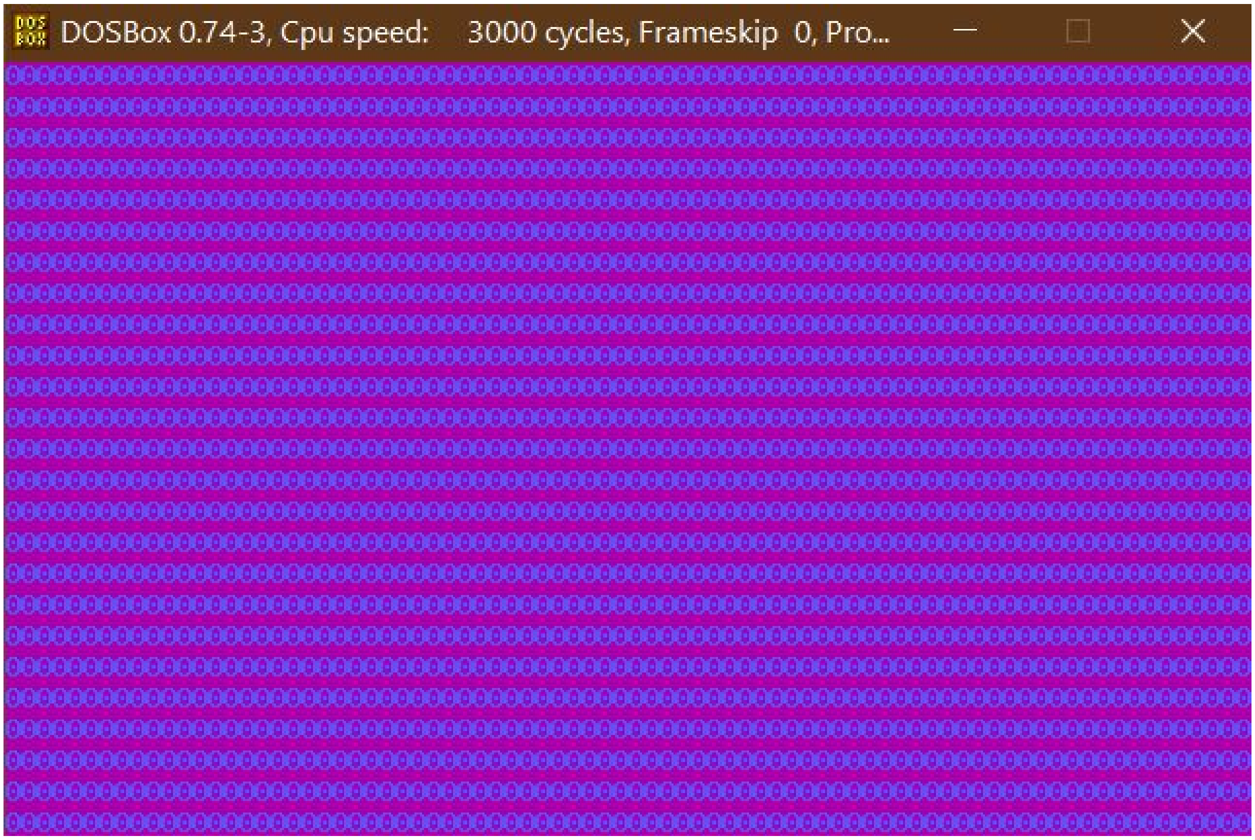

DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Pro...

```
A:\>TASM TSRFINAL.ASM
Turbo Assembler  Version 3.0  Copyright (c) 1988, 1991 Borland International

Assembling file:   TSRFINAL.ASM
Error messages:    None
Warning messages:  None
Passes:            1
Remaining memory:  475k


A:\>TLINK TSRFINAL.OBJ
Turbo Link  Version 3.0 Copyright (c) 1987, 1990 Borland International
Warning: No stack

A:\>TASM TSRFINAL
Turbo Assembler  Version 3.0  Copyright (c) 1988, 1991 Borland International

Assembling file:   TSRFINAL.ASM
Error messages:    None
Warning messages:  None
Passes:            1
Remaining memory:  475k


A:\>
```



```
A:\>DEBUG TSRFINAL.EXE
-G=0000

Program terminated normally
-
```

Output://after 7/8 seconds:


DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Pro...