Microprocessors – Laboratory 01 Debug/EMU86

#	Student ID	Student Name	Grade (10)
-			

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).

Description of main commands

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.

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`	offset

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.

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."
```

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.

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
```

Q (Quit)

The Q command quits Debug and returns to DOS

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.

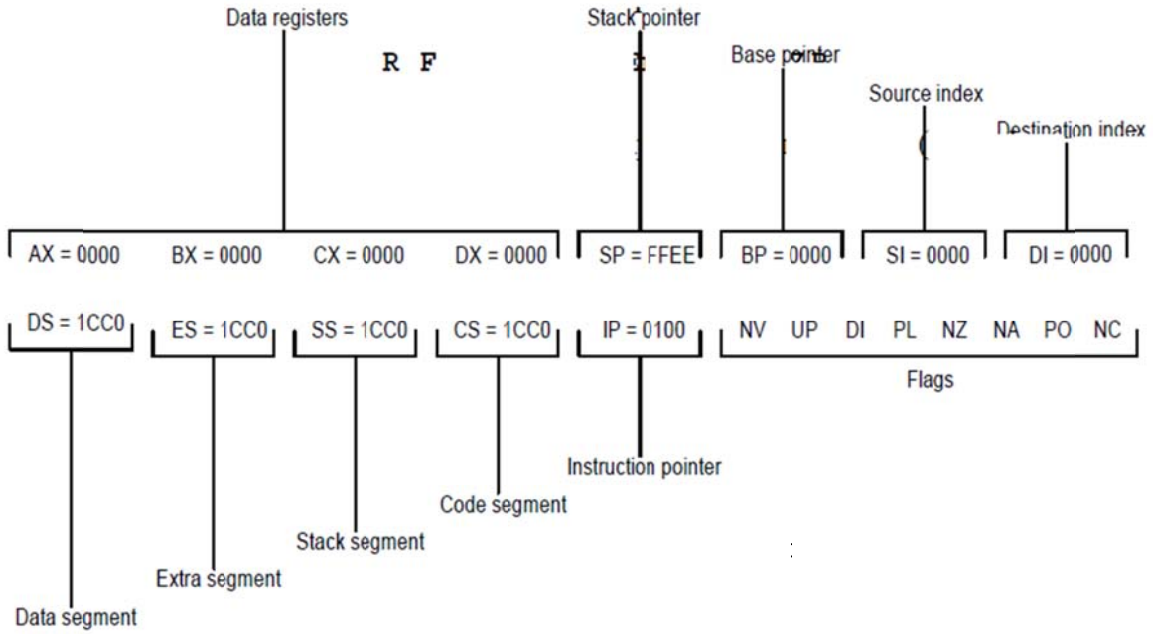
There are two command formats:

```
R
R register
```

Here are some examples:

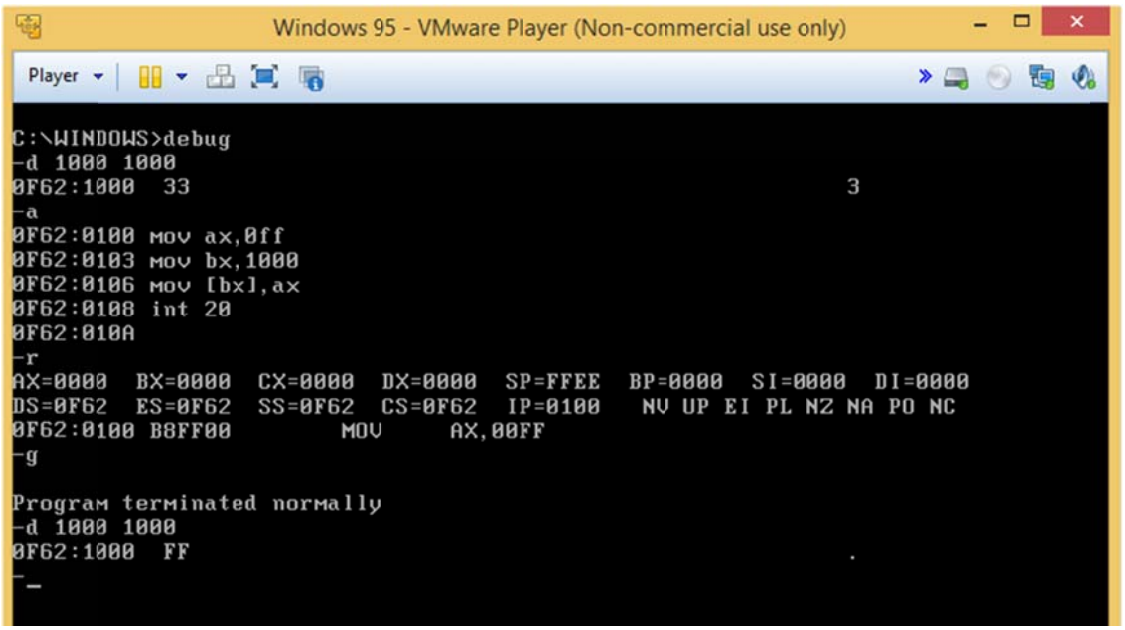
Example	Description
`R`	Display the contents of all registers.
`R IP`	Display the contents of IP and prompt for a new value.
`R CX`	Same (for the CX register).
`R F`	Display all flags and prompt for a new flag value.

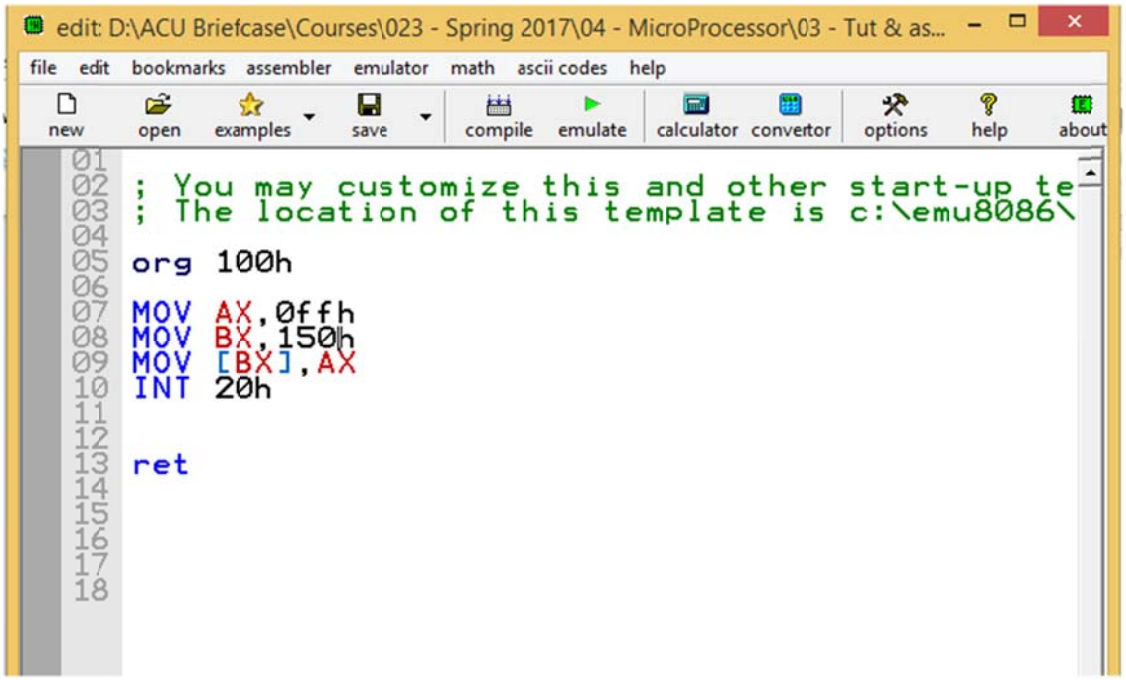
Once the **R F** command has displayed the flags, you can change any single flag by typing its new state. For example, we set the Zero flag by typing the following two commands:

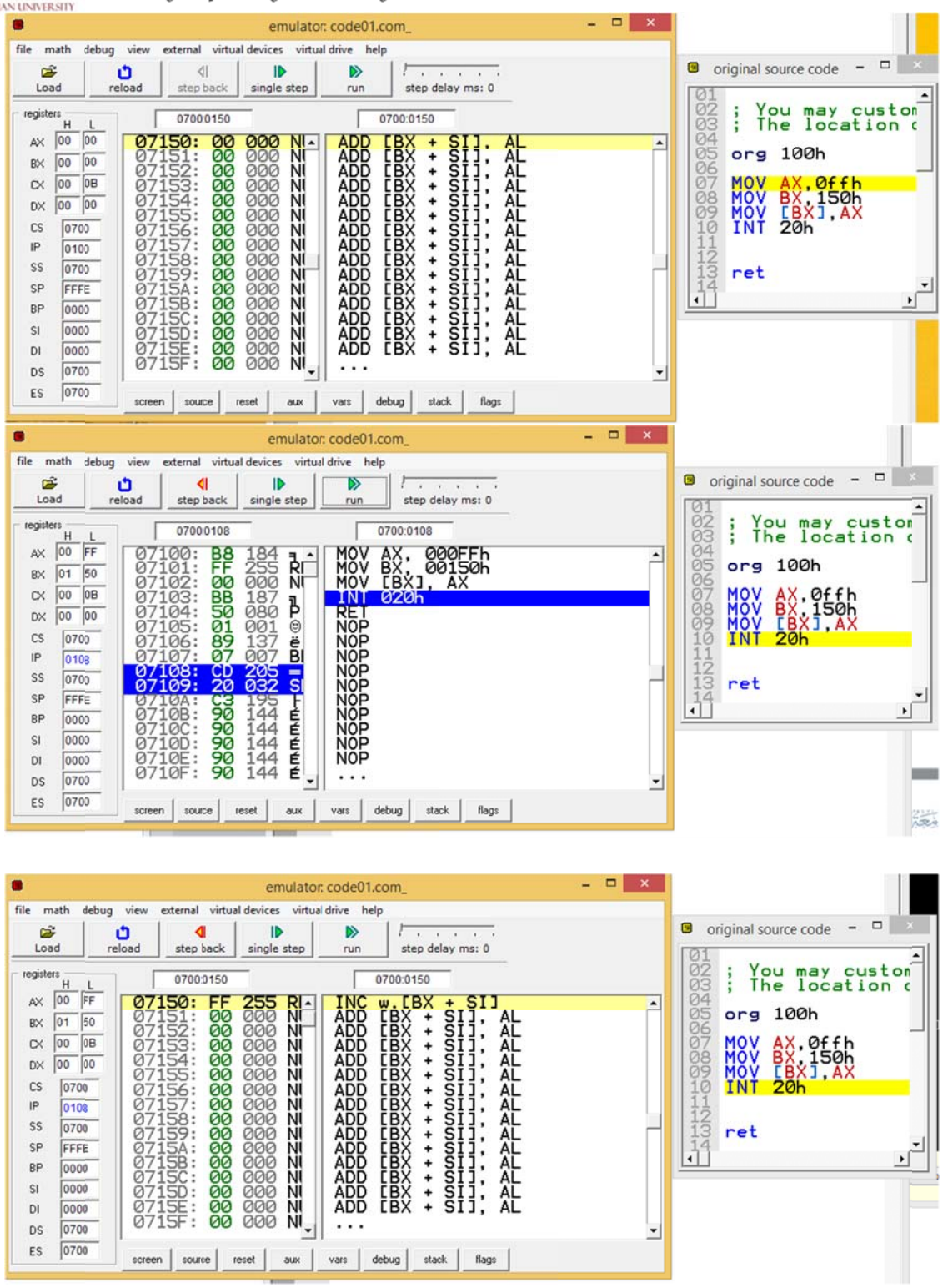


Code 01:

```
MOV AX,0FFh
MOV BX,150h
MOV [BX],AX
INT 20
```






emulator: code01.com_
file math debug view external virtual devices virtual drive help
Load reload step back single step run step delay ms: 0
registers
AX BX CX DX CS IP SS SP BP SI DI DS ES
07000150
0700:0150
07150: 00 000 NI ADD [BX + SI], AL
07151: 00 000 NI ADD [BX + SI], AL
07152: 00 000 NI ADD [BX + SI], AL
07153: 00 000 NI ADD [BX + SI], AL
07154: 00 000 NI ADD [BX + SI], AL
07155: 00 000 NI ADD [BX + SI], AL
07156: 00 000 NI ADD [BX + SI], AL
07157: 00 000 NI ADD [BX + SI], AL
07158: 00 000 NI ADD [BX + SI], AL
07159: 00 000 NI ADD [BX + SI], AL
0715A: 00 000 NI ADD [BX + SI], AL
0715B: 00 000 NI ADD [BX + SI], AL
0715C: 00 000 NI ADD [BX + SI], AL
0715D: 00 000 NI ADD [BX + SI], AL
0715E: 00 000 NI ADD [BX + SI], AL
0715F: 00 000 NI ...
screen source reset aux vars debug stack flags
original source code
; You may custon
; The location
org 100h
MOV AX,0ffh
MOV BX,150h
MOV [BX],AX
INT 20h
ret
emulator: code01.com_
file math debug view external virtual devices virtual drive help
Load reload step back single step run step delay ms: 0
registers
AX BX CX DX CS IP SS SP BP SI DI DS ES
07000108
0700:0108
07100: B8 184 MOV AX, 000FFh
07101: FF 255 MOV BX, 00150h
07102: 00 000 MOV [BX], AX
07103: BB 187 INT 020h
07104: 50 080 RET
07105: 01 001 NOP
07106: 89 137 NOP
07107: 07 007 NOP
07108: CD 205 NOP
07109: 20 032 NOP
0710A: C3 195 NOP
0710B: 90 144 NOP
0710C: 90 144 NOP
0710D: 90 144 NOP
0710E: 90 144 NOP
0710F: 90 144 ...
screen source reset aux vars debug stack flags
original source code
; You may custon
; The location
org 100h
MOV AX,0ffh
MOV BX,150h
MOV [BX],AX
INT 20h
ret
emulator: code01.com_
file math debug view external virtual devices virtual drive help
Load reload step back single step run step delay ms: 0
registers
AX BX CX DX CS IP SS SP BP SI DI DS ES
0700:0150
0700:0150
07150: FF 255 INC w.[BX + SI]
07151: 00 000 ADD [BX + SI], AL
07152: 00 000 ADD [BX + SI], AL
07153: 00 000 ADD [BX + SI], AL
07154: 00 000 ADD [BX + SI], AL
07155: 00 000 ADD [BX + SI], AL
07156: 00 000 ADD [BX + SI], AL
07157: 00 000 ADD [BX + SI], AL
07158: 00 000 ADD [BX + SI], AL
07159: 00 000 ADD [BX + SI], AL
0715A: 00 000 ADD [BX + SI], AL
0715B: 00 000 ADD [BX + SI], AL
0715C: 00 000 ADD [BX + SI], AL
0715D: 00 000 ADD [BX + SI], AL
0715E: 00 000 ADD [BX + SI], AL
0715F: 00 000 ...
screen source reset aux vars debug stack flags
original source code
; You may custon
; The location
org 100h
MOV AX,0ffh
MOV BX,150h
MOV [BX],AX
INT 20h
ret

Exercise 01:

Write following codes and perform indicated operations. Take help from previously stated operations for loading and executing the program.

a)

```
MOV AX,3012h
MOV BX,AX
MOV [1000],BX
```

Observe content of BX register.
what operation happened here

.....
.....
.....
.....
.....
.....

b)

```
MOV AX,30h
MOV [2010],AX
MOV BX,[2010]
```

Observe content of BX register.
what operation happened here

.....
.....
.....
.....
.....
.....

c)

```
MOV SI,1256h
MOV [SI],3251h
MOV AX,[SI]
```

Observe content of BX register.
what operation happened here

.....
.....
.....
.....

d)

```
MOV AL,87h
NEG AL
```

Observe content of AL register.
what operation happened here
*NEG get 2's complement of a number

.....
.....
.....
.....
.....
.....

e)

```
MOV AL,87h
NEG AL
NEG AL
```

Observe content of AL register.
what operation happened here

.....
.....
.....
.....
.....
.....

Code 02:

ADD, SUB, DIV, MUL are all arithmetic commands. ADD is used to add two numbers.

```
MOV AX, 1236H
MOV BX, 1438H
ADD AX,BX
Mov [150H],AX
```

```
MOV AX, 1236H
MOV BX, 1438H
SUB AX,BX
Mov [150H],AX
```

```
MOV AX,1456H
MOV BX,5898H
MUL BX
Mov [150H],BX
```

```
MOV AX,5327H
MOV BX,15F2H
DIV BX
Mov [150H],BX
```

Exercise 02:

Write following codes and perform indicated operations. Take help from previously stated operations for loading and executing the program.

a)

```
MOV AX,1782H
MOV BX,1278H
ADD AX,BX
Mov [150H],AX
```

Observe content of ?? register.
what operation happened here

.....
.....
.....
.....
.....
.....

b)

```
MOV AX,1782H
MOV BX,1278H
SUB AX,BX
Mov [150H],AX
```

Observe content of ?? register.
what operation happened here

.....
.....
.....
.....
.....
.....

c)

```
MOV AX,2782H
MOV BX,1278H
MUL BX
Mov [150H],BX
```

Observe content of ?? register.
what operation happened here

.....
.....
.....
.....
.....
.....

d)

```
MOV AX,57F2H
MOV BX,1375H
DIV BX
Mov [150H],BX
```

Observe content of ?? register.
what operation happened here

.....
.....
.....
.....
.....
.....