Al Imam Mohammad Ibn Saud Islamic University

College of Engineering

Department of Electrical Engineering

Digital System Laboratory Manual EE332

Index

Program No: 1.MULTI BYTE ADDITION

Aim: Write an 8086 Assembly Language program to Add two multi byte numbers and store the result as the third number.

```
DATA SEGMENT
N1DB 55H, 66H, 77H
N2 DB 11H, 22H, 33H
RESULTDB3H DUP (00)
DATA ENDS
CODE SEGMENT
ASSUME CS: CODE, DS: DATA

        START:      MOV AX, DATA
                     MOV DS, AX
                     MOV SI, OFFSET N1
                     MOV DI, OFFSET N2
                     MOV BX, OFFSET RESULT
                     CLC
                     MOV CX, 0003H
                     MOV AX, 0000H
        BACK:MOV AL, [SI]
                     MOV DL, [DI]
                     ADC AL, DL
                     MOV [BX], AL
                     INC SI
                     INC DI
                     INC BX
                     DEC CX
                     JNZ BACK
                     MOV AH, 4CH
                     INT 21H
                     INT 3H
        CODE ENDS
        END START
```

RESULT:

Writew the content of Registers and Memory of the microprocessor as it executes the program.

AX		BX		CX		DX		IP		SP	

Register L	Register H

Instruction Pointer
Code Segment
Data Segment

Stack Segment
Extra Segment

Viva Questions:

1. What is the purpose of BX register?

2. What is the Function of CLC?

3. What is the other instruction which can be used instead of MOV SI offset N1?

4. What is the function of MOV AH, 4CH & INT 21H?

5. What is the purpose of INT 3H?

Program No. 2: MULTI BYTE SUBTRACTION

Aim: Write an 8086 Assembly Language program to subtract two multi byte numbers and store the result as the third number

```
DATA SEGMENT
N1 DB 55H, 66H, 77H, 88H
N2 DB 11H, 22H, 33H, 44H
RESULT DB 4H DUP(00)
DATA ENDS
CODE SEGMENT
ASSUME CS: CODE, DS: DATA

        START:      MOV AX, DATA
                    MOV DS, AX
                    MOV SI, OFFSET N1
                    MOV DI, OFFSET N2
                    MOV BX, OFFSET RESULT
                    CLC
                    MOV CX, 0004H
                    MOV AX, 0000H
                    BACK: MOV AL, [SI]
                    MOV DL, [DI]
                    SBB AL, DL
                    MOV [BX], AL
                    INC SI
                    INC DI
                    INC BX
                    LOOP BACK
                    MOV AH, 4CH
                    INT 21H
                    INT 3H
            CODE ENDS
            END START

        RESULT:
```

Writew the content of Registers and Memory of the microprocessor as it executes the program.

AX		BX		CX		DX		IP		SP	

Register L	Register H

Instruction Pointer
Code Segment
Data Segment

Stack Segment
Extra Segment

Viva Questions:

1. Why subtract with carry instruction is used in the loop?

2. What is the purpose served by BX register?

3. Why subtraction is done with AL register why not with AX ?

4. What is the other instruction which can be used instead of MOV DI, offset N2 ?

Program No.3: MULTI BYTE MULTIPLICATION

Aim: Write an 8086 Assembly Language program to multiply two multi byte numbers and store the result as the third number

```
DATA SEGMENT
N1 DB 05H, 04H, 02H
N2 DB 01H, 02H, 03H
RESULT DB 4H DUP (00)
DATA ENDS
CODE SEGMENT
ASSUME CS: CODE, DS: DATA

    START: MOV AX, DATA
                MOV DS, AX
                MOV SI, OFFSET N1
                MOV DI, OFFSET N2
                MOV BX, OFFSET RESULT
                MOV CL, 03H
                MOV AX, 0000H
                MOV DX, 0000H
                BACK: MOV AL, [SI]
                MOV CH, [DI]
                MUL DH
                MOV [BX], AL
                INC SI
                INC DI
                INC BX
                LOOP BACK
                MOV AH, 4CH
                INT 21H
                INT 3H
        CODE ENDS
        END START

RESULT:
```

Writew the content of Registers and Memory of the microprocessor as it executes the program.

AX		BX		CX		DX		IP		SP	

Register L	Register H

Instruction Pointer
Code Segment
Data Segment

Stack Segment
Extra Segment

Viva Questions:

1. What is the use of stack pointer?
2. What is a directive?
3. What is a pseudo operation?
4. ORG 2000H implies what?
5. Al register is used why not AX?

Program No. 4: MULTI BYTE DIVISION

Aim: Write an 8086 Assembly Language program to divide two multi byte numbers and store the result as the third number

```
DATA SEGMENT
N1 DB 55H, 66H, 99H
N2DB11H, 22H, 33H
RESULT DB 3H DUP(00)
DATA ENDS
CODE SEGMENT
ASSUME CS:CODE, DS:DATA
START:          MOV AX, DATA
                 MOV DS, AX
                 MOV SI, OFFSET N1
                 MOV DI, OFFSET N2
                 MOV BX, OFFSET RESULT
                 MOV CL, 03H
                 MOV AX, 0000H
                 MOV DX, 0000H
                 BACK: MOV AL, [SI]
                 MOV CH, [DI]
                 DIV CH
                 MOV [BX], AL
                 INC SI
                 INC DI
                 INC BX
                 LOOP BACK
                 MOV AH, 4CH
                 INT 21H
                 INT 3H
               CODE ENDS
               END START
       RESULT:
```

Writew the content of Registers and Memory of the microprocessor as it executes the program.

AX		BX		CX		DX		IP		SP	

Register L	Register H

Instruction Pointer
Code Segment
Data Segment

Stack Segment
Extra Segment

Viva Questions:

1. Why AL has been used and not AX?

2. What happens if num1 contains 0AAH and num2 contains 0FFH. ?

3. How do you account for the difference obtained in previous question?

4. Why should AX be used not AL. ?

5. What happens if num1 and num2 values are interchanged?

6. If carry is set to 1 before subtraction what is the instruction to be used?

7. What is an extended accumulator?

8. AL and BL are used for multiplying why not AX & BX?

9. Instead of using MOV BL is it not possible to MUL num2?

10. What is the instruction used for signed multiplication?

Program No. 5: CONVERTION OF BCD TO ASCII NUMBER

Aim: Write an 8086 Assembly Language program to convert BCD to ASCII number

```
        DATA SEGMENT
        N1 DB 56H
        N2 DB 02H DUP (00)
        DATA ENDS
        CODE SEGMENT
        ASSUME CS:CODE, DS:DATA

START:MOV AX, DATA
                MOV DS, AX
                XOR AX, AX
                MOV AL, N1
                MOV SI, OFFSET N2
                MOV BL, AL
                AND AL, 0F0H
                ADD AL, 30H
                MOV CL, 4H
                ROR BL, CL
                AND BL, 0FH
                ADD BL, 30H
                MOV [SI] , BL
                MOV AH, 4CH
                INT 21H
                INT 3H
            CODE ENDS
            END START
```

RESULT:

Writew the content of Registers and Memory of the microprocessor as it executes the program.

AX		BX		CX		DX		IP		SP	

Register L	Register H

Instruction Pointer
Code Segment
Data Segment

Stack Segment
Extra Segment

Viva Questions:

1. What is the difference between adding 30h and OR 30H to a BCD number to conversion toASCII?

2. Why unpacking is necessary during the conversion?

3. What is the ASCII character for symbol A?

4. What is the ASCII character for symbol zero ó0ô?

5. What is ROR instruction will do?

Program No. 6: BLOCK TRANSFER

Aim:
To write an ALP using 8088 to transfer a block of memory content from one location to other location.

```
DATA SEGMENT
N1 DB 01H,02H,03H
DATA ENDS
EXTRA SEGMENT
N2 DB 03H DUP (00)
EXTRA ENDS
CODE SEGMENT
ASSUME CS:CODE, DS:DATA, ES:EXTRA
START: MOV AX, DATA
          MOV DS, AX
          MOV AX, EXTRA
          MOV ES, AX
          MOV SI, OFFSET N1
          MOV DI, OFFSET N2
          CLD
          MOV CX, 0003H
          REP MOVSB
          MOV AH, 4CH
          INT 21H
          INT 3H
          CODE ENDS
     END START
```

RESULT:

Writew the content of Registers and Memory of the microprocessor as it executes the program.

AX		*BX*		*CX*		*DX*		*IP*		*SP*	

Register L	Register H

Instruction Pointer
Code Segment
Data Segment

Stack Segment
Extra Segment

Viva Questions:

1. If the DF=1, will the SI and DI register decremented?
2. The destination memory is pointed by which register combination?
3. The source is pointed to by which register combination?
4. What is the purpose of instruction pointer?
5. What is the purpose of stack pointer?

Program No. 7: FACTORIAL OF A NUMBER

Aim: To write an ALP to find the factorial of a given 8 bit number using indirect addressing mode.

```
DATA SEGMENT
FIRST DB ?
SUM DW 1 DUP(0)
DATA ENDS
CODE SEGMENT
ASSUME CS:CODE,DS:DATA
START:MOV CL,FIRST
          MOV AL,01H
L:      MUL CX
          DEC CL
          JNZ L
          MOV SUM,AX
CODE ENDS
END START
```

Result:

Writew the content of Registers and Memory of the microprocessor as it executes the program.

AX		BX		CX		DX		IP		SP	

Register L	Register H

Instruction Pointer
Code Segment
Data Segment

Stack Segment
Extra Segment

Program No. 8: GENERATION OF FIBONACCI SERIES

Aim: To write an ALP for finding Fibonacci series using direct addressing mode.

```
START:      CODE SEGMENT
            ASSUMECS:CODE
            START:
            MOV SI,0000H
            MOV CX,0005H
            MOV AX,0001H
            MOV BX,0000H
      L:     ADDAX,BX
            MOV [SI],BX
            MOVBX,AX
            MOV AX,[SI]
            INCSI
            DEC CX
            JNZ L
            CODE ENDS
            END START
```

RESULT:

Writew the content of Registers and Memory of the microprocessor as it executes the program.

AX		BX		CX		DX		IP		SP	

Register L	Register H

Instruction Pointer
Code Segment
Data Segment

Stack Segment
Extra Segment

Program No. 9: CONVERTION OF PACKED BCD NUMBERS TO UNPACKED BCD NUMBERS

Aim: To write an ALP using 8086/8088 program to convert Packed BCD numbers to unpacked BCD numbers.

```
DATA SEGMENT
NUMBERDB ?
RESULTDB 1 DUP(0)
DATA ENDS
CODE SEGMENT
ASSUMECS:CODE,DS:DATA
START:MOVAL,NUMBER
        ROL AL,04H
        AND AL,0FH
        MOVRESULT,AL
        MOVAL,NUMBER
        AND AL,0FH
        MOV RESULT+1,AL
        CODE ENDS
END START
```

Result:

Writew the content of Registers and Memory of the microprocessor as it executes the program.

AX		*BX*		*CX*		*DX*		*IP*		*SP*	

Register L	Register H

Instruction Pointer
Code Segment
Data Segment

Stack Segment
Extra Segment

Program No.10: CONVERTION OF BCD NUMBER TO HEXADECIMAL USING INDIRECT ADDRESSING MODE

Aim: To write an ALP to convert BCD number to HEXADECIMAL using indirect addressing mode.

```
DATA SEGMENT
NUMBERDB ?
RESDW 1 DUP(0)
DATA ENDS
CODE SEGMENT
ASSUMECS:CODE,DS:DATA
        START:MOVAL,NUMBER
        AND AL,0FH
        MOVAH,NUMBER
        ROL AH,04H
        AND AH,0FH
        AAD
        MOVRES,AX
        CODE ENDS
        END START
```

Result:

Writew the content of Registers and Memory of the microprocessor as it executes the program.

AX		*BX*		*CX*		*DX*		*IP*		*SP*	

Register L	Register H

Instruction Pointer
Code Segment
Data Segment

Stack Segment
Extra Segment

Program No: 11. SORTING IN ASCENDING ORDER

Aim: To write an ALP to arrange an array of numbers in Ascending order using 8086l8088 microprocessor.

```
     DATA SEGMENT
     N1 DB 56H, 49H, 33H,05H,12H,17H,08H
     DATA ENDS
     CODE SEGMENT
     ASSUME CS:CODE, DS:DATA

     START: MOV AX, DATA
     MOV DS, AX
     XOR AX, AX
     MOV BX, 0006H
   Z: MOV SI, OFFSET N1
     MOV CX, 0006H
     BACK: MOV AL, [SI]
     INC SI
     CMP AL,[SI]
     JBE Y
     XCHG AL,[SI]
     DEC SI
     MOV [SI], AL
     INC SI
  Y: DEC CX
     JNZ BACK
     DEC BX
     JNZ Z
     MOV AH, 4CH
     INT 21H
     INT 3H
     CODE ENDS
     END START
```

RESULT:

Writew the content of Registers and Memory of the microprocessor as it executes the program.

AX		BX		CX		DX		IP		SP	

Register L	Register H

Instruction Pointer
Code Segment
Data Segment

Stack Segment
Extra Segment

Viva Questions:

1. What is the function of JBE ?

2. What is the need of CMP instructions?

3. What is the difference between conditional and unconditional jump instructions?

4. What is the function of XCHG in the program?

5. What is significance of accumulator?

Program No: 12. SORTING IN ORDER DESCENDING ORDER

Aim: To write an ALP to arrange an array of numbers in descending order using 8086l8088 microprocessor.

```
DATA SEGMENT
N1 DB 56H, 49H, 33H,05H,12H,17H,08H
DATA ENDS
CODE SEGMENT
ASSUME CS:CODE, DS:DATA
START: MOV AX, DATA
       MOV DS, AX
       XOR AX, AX
       MOV BX, 0006H
     Z: MOV SI, OFFSET N1
       MOV CX, 0006H
       BACK: MOV AL, [SI]
      INC SI
       CMP AL,[SI]
       JAE Y
       XCHG AL,[SI]
       DEC SI
       MOV [SI],AL
       INC SI
    Y: DEC CX
      JNZ BACK
      DEC BX
      JNZ Z
      MOV AH, 4CH
      INT 21H
      INT 3H
CODE ENDS
END START
```

RESULT:

Writew the content of Registers and Memory of the microprocessor as it executes the program.

AX		BX		CX		DX		IP		SP	

Register L	Register H

Instruction Pointer
Code Segment
Data Segment

Stack Segment
Extra Segment

Viva Questions:

1. What is the function of JAE?

2. What IS the need of CMP instructions?

3. What is the difference between conditional and unconditional jump instructions?

4. What is the function of XCHG in the program?

5. What is significance of accumulator?

Program No: 13. 8279 KEYBOARD DISPLAY

Aim: To demonstrate 8279 Keyboard Display using 8088.

Interface keyboard and display controller 8279 with 8086 at addresses 0080H. Write an ALP to set up 8279 in scanned keyboard mode with encoded scan, N-key rollover mode. Use a 16-character display in right entry display format. Then clear the display RAM with zeroes. Read the FIFO for key closure. If any key is closed, store its code to register CL. Then write the byte 55 to all the displays and return to DOS. The lock input to 8279 is 2MHz, operate it at 100 kHz.
Tools Required: TASM, 8086 Kit, 8279 interfacing card.

Theory:

The 8279 is interfaced with lower byte of the data bus, i.e. D0-D7. Hence the A0 input of 8279 is connected with address line A1. The data register of 8279 is to be addressed as 0080H, i.e. A0=0. For addressing the command or status word A0 input of 8279 should be 1 (the address line A1 of 8086 should be 1), i.e. the address of the command word should be 0082H.

Procedure:

Step1: Set 8279 command words according to program i.e. Keyboard/Display Mode Set CW, Program clock selection, Clear Display RAM, Read FIFO, Write Display RAM commands.

Step2: Read FIFO command for checking display RAM.

Step3: Wait for clearing of Display RAM by reading FIFO Du bit of the status word i.e. if Du bit is not set wait, else proceed.

Step4: Read FIFO command for checking key closure, also read FIFO status.

Step5: Mask all bits except the number of characters bits. If any key is pressed, take required action; otherwise proceed to write display RAM by using write display command.

Step 6: Write the byte 55H to all display RAM locations.

Step 7: Call routine to read the key code of the pressed key is assumed available.

This Program displays the code of the key, which is pressed on the keyboard pad. The code is displayed in the data field and remains unchanged till the next key is pressed.

Description of the Program:

The port of 8255 i.e. P1 is initialized to make port A as input port and port C as output port. The three Rows of the key are scanned one by one and process is repeated till the key is pressed, in the routine code and F code (final code). The information of code is then displayed and the monitor jumps back again to see if any other key is pressed.

Addresses	Opcodes	Label	Mnemonics	Comments
0400	BA FF FF	KBD	MOV DX,FFFF	Initialize the port – B and C as an output
0403	B0 90		MOV AL,90	
0405	EE		OUT DX,AL	
0406	B7 00	INIT	MOV BH,00	Initialize the final key code in Reg. BH
0408	B3 01		MOB BL,01	Put the walking one pattern in register C with one LSB position
040A	88 D8	SCAN	MOV AL, BL	Move the pattern in AL on port C
040C	BA FD FF		MOV DX, FFFD	
040F	EE		OUT DX, AL	
0410	BA F9 FF		MOV DX, FFF9	
0413	EC		IN AL,DX	Input Port – A
0414	E8 27 00		CALL CODE	Classify the 8 word into 8 bits
0417	3C 08		CMP AL,08	Any Ke closure
0419	78 10		JS DISP	Yeas – go to display it
041B	80 C7 08		ADD BH,08	Increment the PC code in the partial result.
041E	80 FF 18		CMP BH,18	Has PC code become 18
0421	79 E3		JNS INIT	Yes – go start scanying from Row 0
0423	88 D8		MOV AL,BL	No
0425	D0 D0		RCL AL,01	Move the walking one to scan the next line
0427	88 C3		MOV BL,AL	
0429	EB DF		JMP SCAN	Continue scanning
042B	08 F8	DISP	OR AL,BH	OR the PA code with PC code
042D	B4 00		MOV AH, 00	Display the code in data field
042F	50		PUSH AX	
0430	B0 00		MOV AL, 00	
0432	50		PUSH AX	
0433	B0 01		MOV AL, 01	
0435	50		PUSH AX	
0436	50		PUSH AX	
0437	9A E0 0B 00 FF		CALL DB "OUTWARDS"	
043C	EB C8		JMP INIT	Go to scan the keyboard again
043E	08 C0	CODE	OR AL, AL	Checking for valid key press
0440	75 03		JNZ CODE2	If yes go to code2 else
0442	B0 08		MOV AL, 08	
0444	*C3*		RET	

```
0445    B5 00    CODE 2    MOV  CH, 00
0447    D0 C8    CODE 5    ROR  Al,01      Let LSB in AL go to carry
0449    72 04              JC   CODE 10    Go to return if this bit was one
044B    FE C5              INC  CH         Increment counter
044D    EB F8              JMP  CODE5      Check the next bit
044F    88 E8    CODE10    MOV  AL, CH
0451    C3                 RET
```

Writew the content of Registers and Memory of the microprocessor as it executes the program.

AX		BX		CX		DX		IP		SP	

Register L	Register H

Instruction Pointer
Code Segment
Data Segment

Stack Segment
Extra Segment

Program No: 14. GENERATION OF SINUSOIDAL WAVE USING 8255

Aim: To write an ALP to generate Sinusoidal Wave Using 8255.

```
ASSUME CS:CODE,DS:DATA
SINE DB 0,11,22,33,43,54,63,72,81,90,97,104,109,115,119,122
DB 125,,126,127,126,122,119,115,109,104,97,90,81,72,63,54,43,33,22,11
PA EQU 44A0H
CR EQU 44A3H
DATA ENDS
CODE SEGMENT
        START: MOV AX, DATA
               MOV DS, AX
               MOV DX, CR
               MOV AL, 80H
               OUT DX, AL
          REPEAT: MOV DX, PA

               LEA SI, SINE
               MOV CX, 36
          NEXT: MOV AL,[SI]
               ADD AL, 128
               OUT DX, AL
               INC SI
               LOOP NEXT
               MOV CX, 36
               LEA SI, SINE
        NEXT1: MOV AL, 128
               MOV AH,[SI]
               SUB AL, AH
               OUT DX, AL
               INC SI
               LOOP NEXT1
               JMP REPEAT
               MOV AH, 4CH
               INT 21H
               CODE ENDS
               END START
```

Result:

Writew the content of Registers and Memory of the microprocessor as it executes the program.

AX		BX		CX		DX		IP		SP	

Register L	Register H

Instruction Pointer
Code Segment
Data Segment

Stack Segment
Extra Segment

Appendix –A

Microprocessor Intel 8088/8086 Kit Commands

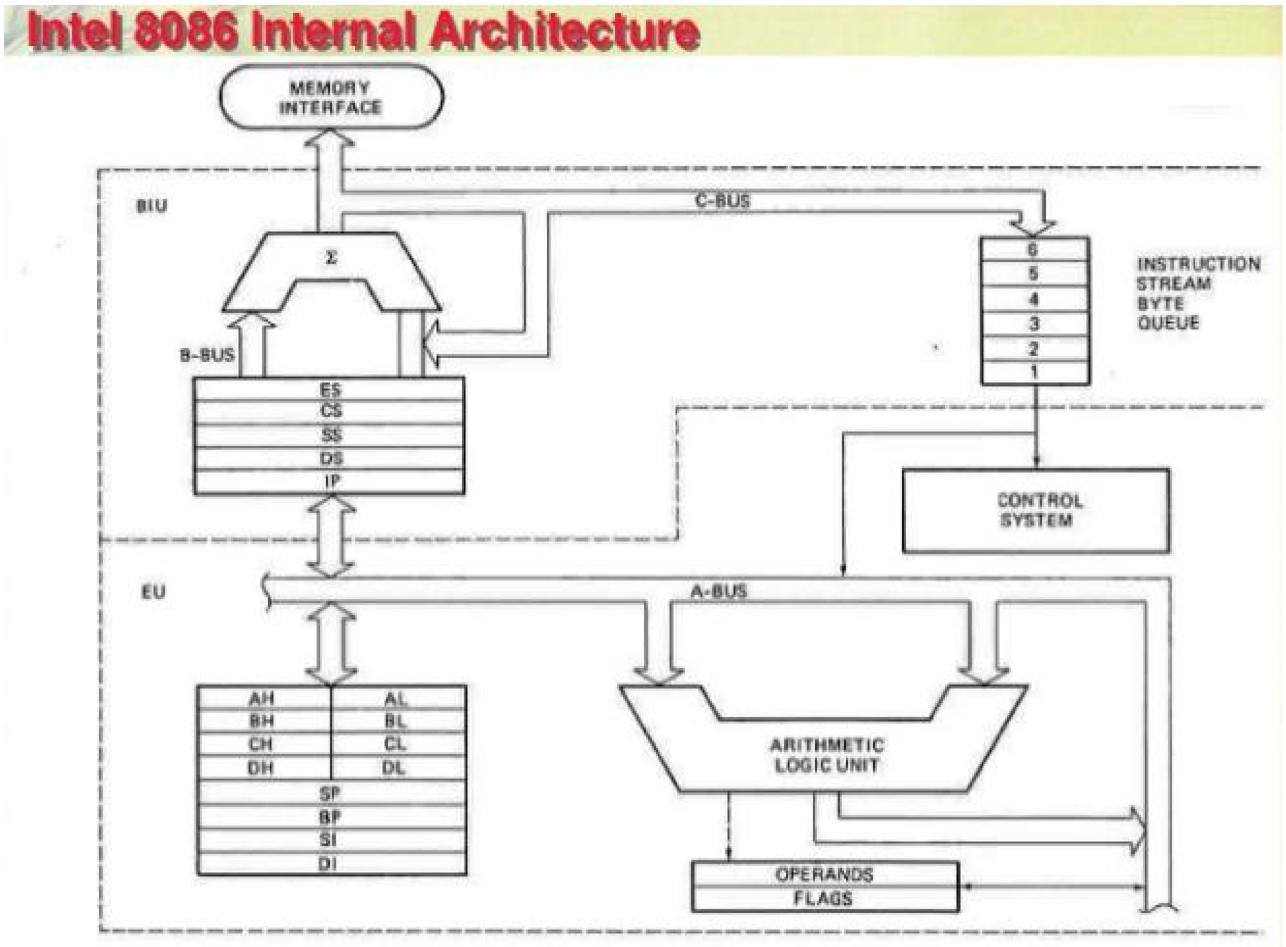

FIGURE 8086 internal block diagram. (*Intel Corp.*)

1. Line Assembler and Disassembler Commands:

Command	Syntax	Description
A **(Assembler)**	1)A 2)A<add>	To let the user to type 8088 ALP into memory and assemble them into machine code line by line.
L **(Disassembler)**	1) L 2) L <addr1> 3) L <addr1>/<n> 4) L <addr1><addr2>	To translate (disassemble) a block of machine codes in memory into 8088 assembly instructions.

2. Program Execution Commands:

Command	Syntax	Description
G **(Go)**	1) G 2) G <addr>	To execute a program in memory
S **(Step)**	1) S 2) S n	To single-step a program or execute a specified number of instructions and then stop with a display of register contents on the screen; execution starts from the address pointed to by the code segment CS register and the IP instruction pointer.

3. Debugging Commands:

Command	Syntax	Description
B (Breakpoint)	1) B 2) B<n> 3) B<n><addr>	To set up to three breakpoints or display their current settings. When a program is on execution and runs into a breakpoint address, the program execution will be halted.
C (Cancel)	1) C 2) C <n>	To cancel one or all of the break points set previously.
X (Register)	1) X 2) X <register name>	To display or change the contents of any of the registers
M (Memory)	1) M 2) M <addr1> 3) M <addr1><addr2> 4) M <addr1><addr2> / <data1>/ 5) M <addr1> / <data1> [data2]é ../ 6) M <addr1> / <addr2> / <data1> [data2]é ../	To display or change the contents of a memory location or a range of memory location.
I (Insert)	1) I 2) I /<data1> [data2]é ../ 3) I <addr1> 4) I <addr1> /<data1> [data2]é . / 5) I <addr1><addr2> /<data1> [data2]é / 6) I <addr1><addr2> /<data1> [data2]é /	To insert data into a memory location or a portion of memory locations

D **(Delete)**	1) D 2) D /<n> 3) D <addr1> 4) D <addr1> /<n> 5) D <addr1><addr2> 6) D <addr1><addr2< /<n>	To delete a byte of data or a segment of data in memory
F **(Find)**	1) F /<datastring> 2) F<addr1> /<datastring> 3) F <add1><addr2> /<datastring>	To search for a specified value or set of values in memory
J **(Jump)**	1) J <addr>	To directly jump to the particular address from which program execution must start.
T **(Transfer)**	1) T <addr1><addr2><addr3 > 2) T <addr4><addr5> /<n>	To copy a range of memory contents to another area in memory.
P **(Pause)**	1) P<n>	To adjust the speed of displaying on the screen.
N **(Input)**	1) N <port_address>	To input and display in hexadecimal One byte of data from the specified port
W **(Write)**	1) W <addr1><addr2> /<file_name>	To record the contents of a range of memory on tape
R **(Read)**	1) R /<file_name> 2) R <addr> /<file_name 3) R 4) R <addr>	Read data from tape

4. Control Characters:

Control characters are entered by pressing a predefined key while holding down the CTRL or ALT key. Each control character performs a specific function

- <CTRL-S> :Suspends output to the display. Pressing any key resumes output to the display.
- <CTRL-P> : This control character is a toggle switch. Entering this control character once turns on the printer, causing the screen output to be sent to the line printer. Entering this control character a second time turns off the printer.
- <CTRL-X> : Cancels the current input line to the command buffer.
- <ALT-Y> : Returns the system control to the monitor program.

Control Characters for moving Cursor in the Command Line Buffer:

- <ALT-E> : Moves the cursor up one line.
- <ALT-X> : Moves the cursor down one line.
- <ALT-S> : Moves the cursor to the left one position.
- <ALT-D> : Moves the cursor to the right one position.
- <ALT-F> : Returns the cursor to where it was located before it was moved.

Control Characters for Scrolling the screen:

- <ALT-A> : Scrolls down the screen
- <ALT-Z> : Scrolls up the screen
- <ALT-Q> : Returns the cursor to where it was located before it was moved.

Appendix-B

DOS Debugger Program

A DOS debugger is a utility program that comes with the MS-DOS operating system as a programming tool for debugging (trouble shooting) and testing executable files. The debugger is sometimes called a "mini-assembler," because it allows the user to create and run short or small assembly language programs. With the debugger, the user can access the system's main memory and save the program directly on the disk without the intervention of the DOS. In addition, the debugger can be used to disassemble object codes (machine instructions) into readable codes.

There are two ways to run the program under the debugger: **single-step** (trace) or **Go** with full speed. Single-stepping or tracing the program is to execute one instruction at a time. After each tracing, the program stops so that the user can examine the effects that are caused by every instruction in the program. The debugger also allows us to set a few **breakpoints**, and run program at the full speed until the particular instruction is reached.

Note that unless you are real sure about the operational function of the program that you are debugging, don't use **G** command to run the program, because it might cause some unexpected results. Since the DOS debugger allows the user to write on the absolute sector address on a disk, for the safety reason, it is very important that the user should make a backup copy of the program and then run the program.

DOS Debugger Environment

Before invoking or loading the DOS Debugger, you may want to locate where the DEBUG.COM program is stored in the system: it may be stored under C:\DOS\DEBUG.COM, if you are using a PC with a hard disk; or, it may be on a DOS supplemental disk or operating disk. After invoking the DEBUG.COM, the DEBUG itself then is loaded into the following memory area in the PC's main memory.

Refer to Figure 1, via the DEBUG you have complete access to the memory and the CPU registers of the 8088/8086 used in the PC.

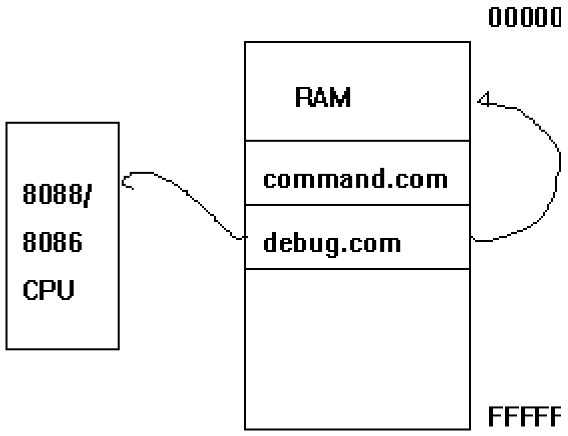

Figure 1. The Debugger environment

Figure 2 shows the 16-bit registers that are shown on the debug screen. Note that the 8-bit registers such as AH, AL, and so on, are not directly displayed on the debug screen.

16-bit 8-bit Special Registers/general purpose

AX AH AL Accumulator

BX BH BL Base register

CX CH CL Counter register

DX DH DL Data register

16-bit Instruction Fetch

CS Code Segment Register Base address

IP Instruction Pointer Offset address

16-bit Stack Segment Operation

SP Stack Pointer Offset address

BP Base register Offset address

SS Stack segment register Base address

16-bit Source of String /Array

DS Data Segment Register Base address

SI Source Index Register Offset address

16-bit Destination of String /Array

ES Extra Data Seg. Reg. Base address

DI Destination Index Reg. Offset address

Flags: Overflow (OF), Directional Flag (DF), Interrupt Flag (IF), Trace Flag (TF), Sign bit flag (SF),

Zero Flag (ZF), Auxiliary Flag (AF), Parity Flag (PF), and Carry Flag (CF)

Figure 2. 8088/8086/80286 CPU registers

Debugger Commands

All DEBUG commands are single letter commands, and are usually followed by one or more parameters. The commands can be uppercase or lower case letters, or a combination of both. Delimiters are only required, however, between two consecutive hex numbers. The commands become effective only after you press the [Enter] key. To end the command, enter [Ctrl] and [Break] keys.

To invoke the debugger, enter one of two commands (depending on the system setup); the debug prompt, a dash '-', is displayed; debugger is then ready to take commands from you.

C>DOS\DEBUG ....If using a hard disk

A>DEBUG ....*If two floppy drive PC is used*

- ....*Debug prompt*

DEBUG COMMANDS

Commonly used debug commands such as ASSEMBLE, DUMP, ENTER, FILL, MOVE, COMPARE, and SEARCH are available for use in entering programs, examining or modifying storage locations in the memory.

A[address] (Assemble command)

We can write an assembly language program with the "A" or "a" command. The Assemble command "A" allows us to enter assembly language instructions using mnemonic symbols directly into memory. When we enter just "A" or "a" after the debugger prompt:

```
-A

0910:100- ..... you enter the instruction right after this prompt
```

CS:0100 is the default address. It uses the offset address 100H as the beginning address of the program. The other alternative is to specify the beginning address explicitly as follows:

```
-A100

0910:100- ..... you enter the instruction right after this prompt
```

C [addr1][range][addr2] (Compare command)

- [addr1] is the beginning address of 1st block of memory

- [range] determine the length of comparison

- [addr2] is the beginning address of the second block of memory

The "C" command compares the contents of two blocks of memory. For example, the following commands compare a memory block starting at DS:200 through DS:300 to an equal-size data block starting at address DS: 500. It will display each unequal element found in the address and contents of that byte in both blocks. No information is displayed if both of these blocks contain the same data.

```
-C 200 300 500
```

D[address] (Display or dump memory command)

The "D" command allows us to examine the contents of memory locations by dumping the data stored in the memory onto the screen.

If [address] is omitted,

```
-D
```

then the address used by previous D command is assumed and the 128 consecutive bytes offset from the current value in DS are displayed. When no previous D is used, the offset address 100 Hex is used.

Other examples of using the "D" command are

```
-D 100 .. dump memory locations starting at 100H

-D DS:100 .. dump memory locations starting at DS:100

-D CS:100 .. dump memory location starting at CS:100

-D DS:200 210 .. dump memory location in the range of 200-210
```

E[address][list] (Enter command)

Enter command will replace the contents of memory locations starting at the address specified by the DS register. Using this command, you can enter the machine code directly into the memory. The following examples show how to use the "E" command:

```
-E DS:100 FF FF FF FF .. enter four FF at 100H

-E DS:200 "The E Command:" .. enter ASCII characters
```

F [range][list] (Fill memory command)

The format of the "F" command is filling the memory locations in the [range] with the values in the [list]. The "F" command can be used to fill a block of consecutive memory locations with the same data. The "F" command examples are:

```
-F 100 110 FF .. Fill memory 100-110 with FF

-F 200 220 00 .. Fill memory 200-220 with 00
```

G[=address][address[address..]] (Go, execute command)

The "G" command allows us to execute the instructions and programs under the debugger. Examples of the GO commands are

```
-G =CS:200 210 .. run the program from 200 through 210

-G =CS:100 .. run the program at 100

-G .. run the program from current CS:IP location
```

H[value][value] (Hex arithmetic: add and sub)

Add and subtract two hex numbers; display sum and display. For example, we enter

```
-H AE BF

01B0 FF04 .. first number is sum .. second no is difference

-H 96 C2

0158 FF04
```

I[port address] (Input command)

Input and display one byte from the specified port in hex

L[address[drive [sector sector]] (Load a file or disk sectors)

Load a file or absolute diskette sectors into memory; maximum number of sectors that can be loaded with a single Load command is Hex 80.

M[range][address] (Move a block of data)

Move or copy from one block of memory to another. Example of M command is

```
-M 100 200 300 .. move a data block start from the location

through 200 to the destination block starting

at 300
```

N[d:][path] filename (Name)

Name the file for the write disk command or Load command to call it.

O[port address][byte] (Output)

Send output bytes to the output port

P[=address][value] (Proceed)

Execute a loop, a repeat string instructions, a software interrupt, or a procedure

Q exit DEBUG program and return to the DOS

R[register name] (Register)

Display and/or modify the contents of registers and flags

S[range][list] (Search)

Search for characters through a specified range of address. The example is

```
-S 200 210 FF .. search the data byte FF in the range 200

through 210
```

T[=address][value] (Trace)

Execute an instruction and display the CPU registers. The examples of "T" commands are

```
-T .. trace the program from current CS:IP

-T =CS:100 .. trace the program from CS:100

-T =CS:100 4 .. trace the program from CS:100 for 4

instructions
```

U[address] (Unassemble command)

Translate the contents of memory into assembly instructions. The examples of "U" commands are

```
-U .. unassemble the program from current CS:IP

-U CS:100 105 .. unassemble the program from 100 to 105
```

W[address[drive sector sector]] (Write to disk command)

Write file on absolute diskette sectors. The CX should hold the number of bytes to be written on the disk, and the name of the program should be given by using the "N" command.

DEBUG COMMANDS (Continue)

Following commands are found in DOS 5.0 only

XA[count] (Allocate expanded memory)

Allocate a specific page of expanded memory. A page is 16-kbytes of memory.

For example: allocate 10 pages of expanded memory:

```
-XA A ..... you enter
Handle created=0003
```

XD[handle] (Deallocate expanded memory)

Deallocate a handle to expanded memory

For example: deallocate the handle 0003

```
-XD 0003
Handle 0003 deallocated
```

XM[lpage][ppage][handle] (Map expanded memory pages)

Map a logical page of expanded memory, belonging to the specific handle, to a physical page of expanded memory.

For example: to map logical page 6 of handle 0003 to physical page 2

```
-XM 5 2 0003
```

XS (Display expanded memory status)

Display information about the status of expanded memory

For example:

```
-XS
```

Using DEBUG

Example 1:

Using DEBUG.COM to examine 8088/8086/80286 CPU registers, modify the contents of AX register, verify the change, and then exit the debugger.

```
-r ....Examine registers command

AX=0000 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000

DS=111E ES=111E SS=111E CS=111E IP=0100 NV UP EI PL NZ NA PO NC

111E:0100 2D444F SUB AX, 4F44

-RAX ....Examine and modify AX register

AX=0000

:10 ....You enter 10

-R ....Examine the resiters to see AX

AX=0010 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000

DS=111E ES=111E SS=111E CS=111E IP=0100 NV UP EI PL NZ NA PO NC

111E:0100 2D444F SUB AX, 4F44

-Q ....Exit debugger and return the control

to DOS

A>
```

Figure 3. Output display of example 1

After the **r** command is entered, the DEBUG sets the CPU registers with default values: registers AX, BX, CX, DX, BP, SI, and DI are set to 0000.

All segment registers are set to DS=111E, ES=111E, SS=111E, CS=111E. This number is the base address of the memory that the DEBUG is working on and it may be different from one machine to another machine, and may be different from one time to another. The instruction pointer, IP, is set to its default offset address (0100) by the DOS. Starting from actual memory location 112E0 Hex, that we calculated by shifting the IP one digit to the right and added to the CS, there is a garbage instruction

SUB AX, 4F44.

CS 111E

+ IP 0100

Effective Address 112E0 Hex

You then enter: **RAX** (or **R AX** or **rax**)

for examining/or modifying AX register. If you want to examine other registers, the correct register name must be referred. The AX=0010 is the correct entered value.

Finally, you enter **Q** or **q** to exit the DEBUG.

Example 2. Using the DEBUG to perform the following tasks:

1) Enter a short assembly language program as shown in Figure 4.

2) Save this program as "a:ex2-2.com" then exit DEBUG

3) Reinvoke the DEBUG to test the **ex2-2.com** program.

```
A>DEBUG .... invoke or load debugger

-A100 .... assemble the instructions at CS:100

111E:0100 MOV AX,0123 ... hit Enter key

111E:0103 ADD AX,25 ... hit Enter key

111E:0106 MOV BX,AX

111E:0108 ADD BX,AX

111E:010A MOV CX,BX

111E:010C SUB CX,AX

111E:010E SUB AX,AX
```

```
111E:0110 ... hit Enter key to end

-R ... examine registers

AX=0010 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000

DS=111E ES=111E SS=111E CS=111E IP=0100 NV UP EI PL NZ NA PO NC

111E:0100 B82301 MOV AX, 1023

-U ... disassemble the instructions

111E:0100 B82301 MOV AX,0123

111E:0103 052500 ADD AX,25

111E:0106 89C3 MOV BX,AX

111E:0108 01C3 ADD BX,AX

111E:010A 89D9 MOV CX,BX

111E:010C 29C1 SUB CX,AX

111E:010E 29C0 SUB AX,AX

111E:0110 0800 OR [BX+SI],AL .... Junks
```

Figure 4. Output display of example 2

The assembly language instructions that we use in this example are:

```
MOV AX, 1023 ; Move 1023 Hex into AX register

ADD AX, 25 ; AX = AX + 25 Hex = 1048 Hex

MOV BX, AX ; Move the content of AX register to BX register and

; the AX is unchanged BX = AX = 1048

ADD BX, AX ; BX = BX + AX = 1048 + 1048 = 2090 Hex

MOV CX, BX ; a copy of BX is move to CX register

SUB CX, AX ; CX = CX - AX

SUB AX, AX ; AX =AX - AX or clear AX register
```

Example 2 (continue)

2) Save the entered program as **a:ex2-2.com.**

Before you save the program, you may want to count how many bytes of memory space will be needed by this program. We count it starting from the instruction MOV AX, 1023 through SUB AX,AX.

The number on the first column is the base address and the numbers on second column are the offset addresses of the memory.

The numbers on the third column are translated machine codes. One hexadecimal digit on this column is 4-bit (two digits give 1-byte).

We counted the number on the third column: 16-bytes. The DEBUG uses the CX register as a counter that indicates the number of bytes to be saves. So we entered the following commands to save the program:

-RCX ..... *modify the CX register*

CX 0000

:10 ..... *hex (decimal 16)*

-n a:ex2-2.com ..... *name the program*

-w ..... *write to the disk*

Writing 0010 bytes

-q ..... *Exit DEBUG*

A>DIR ex2-2.com ..... *see if the program is saved*

Figure 2-12 Output display of example 2-2

3) Reinvoke the DEBUG for testing a:ex2-2.com

```
A>DEBUG ex2-2.com
```

-u .... *unassembly the program*

```
1133:0100 B82301 MOV AX,0123

1133:0103 052500 ADD AX,25

1133:0106 89C3 MOV BX,AX

1133:0108 01C3 ADD BX,AX

1133:010A 89D9 MOV CX,BX

1133:010C 29C1 SUB CX,AX

1133:010E 29C0 SUB AX,AX

1133:0110 50 PUSH AX .... Junks -
```

Figure 5. Output display of example 2

Example 2 (continue)

4) Test the ex2-2.com program by tracing the instructions

```
-R ..... examine CPU regiter before single-step

AX=0010 BX=0000 CX=0010 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000

DS=1133 ES=1133 SS=1133 CS=1133 IP=0100 NV UP EI PL NZ NA PO NC

1133:0100 B82301 MOV AX, 1023 ....(Move 1023Hex to AX)

-T ...... execute the instruction appears

AX=1023 BX=0000 CX=0010 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000

DS=1133 ES=1133 SS=1133 CS=1133 IP=0103 NV UP EI PL NZ NA PO NC

1133:0103 052500 ADD AX, 0025 ...... (AX = AX + 25 Hex)

-T

AX=1048 BX=0000 CX=0010 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000

DS=1133 ES=1133 SS=1133 CS=1133 IP=0106 NV UP EI PL NZ NA PE NC

1133:0106 89C3 MOV BX, AX ...... (BX = AX)

-T
```

```
AX=1048 BX=1048 CX=0010 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000

DS=1133 ES=1133 SS=1133 CS=1133 IP=0108 NV UP EI PL NZ AC PE NC

1133:0108 01C3 ADD BX, AX ......(BX = BX + AX)

-T

AX=1048 BX=0290 CX=0010 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000

DS=1133 ES=1133 SS=1133 CS=1133 IP=010A NV UP EI PL NZ AC PE NC

1133:010A 89D9 MOV CX, BX ......(CX = BX)

-T

AX=1048 BX=0290 CX=0290 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000

DS=1133 ES=1133 SS=1133 CS=1133 IP=010C NV UP EI PL NZ AC PE NC

1133:010C 29C1 SUB CX, AX ......(CX = CX - AX)

-T

AX=1048 BX=0290 CX=1048 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000

DS=1133 ES=1133 SS=1133 CS=1133 IP=010E NV UP EI PL NZ AC PE NC

1133:010E 29C0 SUB AX, AX ......(AX = AX - AX)

-T

AX=0000 BX=0290 CX=1048 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000

DS=1133 ES=1133 SS=1133 CS=1133 IP=0110 NV UP EI PL NZ NA PE NC

1133:0110 50 PUSH AX ...... Junks, stop!
```

Figure 6. Output display of example 2

Notice that in the debugger environment, all the numeric data are represented in hex numbers. The debugger shows the instruction to be executed, before you enter the trace command (**T**). After the execution of each instruction, the instruction pointer IP is incremented and the register that appears on the left side of the instruction is modified. In addition, some flags are changed to reflect the status after the operations. All these changes are showed in the high-lighted areas. If you want to run trace the program one more time, the IP must be set to the right offset address.

Example 3. Using the debugger to perform the following tasks:

1) Create a short assembly language program that does the following:

- Move 09 Hex to AX register:

- Double this number in AX and then move a copy of this number to both

BX and CX registers

- Set the source index register (SI) to 180 Hex: [MOV SI, 180]

- Store the contents of AX, BX, and CX to the data area pointed to by DS:SI, where

the data segment register (DS) holds the default base address:

```
MOV [SI], AX

INC SI

MOV [SI], BX

INC SI

MOV [SI], CX
```

2) Save this program as **a:ex2-3.com**

3) Exit the Debug

4) Invoke DEBUG to load **a:ex2-3.com**

5) Trace the program and examine the data area that is pointed to by DS:180

- Fill the data area pointed to by the DS:200 through DS:220 with the hex number FF

and examine this result.

Solution:

1) Moving the hex number 09 into AX register requires the instruction [MOV, AX, 09].

This is the instruction of the **immediate addressing mode**, since the data 09 is included in the instruction. Notice that AX is the destination of the data movement.

2) There is more than one instruction than can be used to double the contents of the AX register. We use addition instruction: [ADD AX,AX]. This is the **register addressing** mode instruction since the source and destination of the data are all within the CPU register. This instruction does the following operation: AX = AX + AX

3) The [MOV BX,AX; MOV CX, AX] instructions move data from AX to both BX and CX. The addressing mode of this type of instruction is register addressing.

4) The source index register is initialized with the offset address 180 hex through the instruction MOV SI, 180

5) To store the data in the data area whose base address is held by the data segment register DS and offset address is held by the SI register. We use the **indirect addressing** instruction:

MOV [SI], AX

Where the bracket around the SI register means that the destination of this data movement is the memory pointed by the DS:SI register.

We move three 16-bit words' data to memory. After the first movement, we then increment the SI by one so it can point to next available location:

INC SI

We then repeat the SI updating and data storing until all the contents of the AX, BX and CX are stored. Figure 2-15 shows the programming model for the solving this problem.

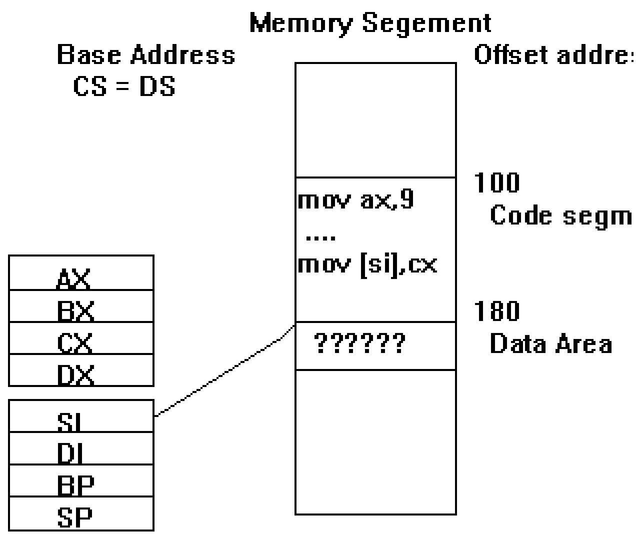

Figure 4 Programming model of the example 3

Invoke the debug , enter the instructions, name the program, count file size, and write to

the disk.

```
A>DEBUG

-a100 .... assemble the instructions at CS:100

111E:0100 mov ax,09 .... hit enter

111E:0103 add ax,ax

111E:0105 mov bx, ax

111E:0107 mov cx, ax

111E:0109 mov si, 180

111E:010C mov [si],ax

111E:010E inc si

111E:011F mov [si],bx

111E:0111 inc si

111E:0112 mov [si],cx

111E:0114
```

```
-n a:ex2-3.com .....name the program

-r cx ....No of bytes to write

CX 0000

:14

-w

Writing 0014 bytes
```

Figure 5Output screen of example 3.

There are two ways to load the program into the debug. The first approach is to invoke the DEBUG and pass the program name as the parameter: **A>DEBUG A:EX2-3.COM**

The second approach is to invoke the debug, name the program, and then load the program. We take this approach to bring the program back to DEBUG

```
A>DEBUG

-n a:ex2-3.com

-l ...... load the program

-r

AX=0000 BX=0000 CX=0014 DX=0000 SP=FFFE BP=0000 SI=0000 DI=0000

DS=1133 ES=1133 SS=1133 CS=1133 IP=0100 NV UP EI PL NZ NA PO NC

1133:0100 B80900 MOV AX,0009

-t

AX=0009 BX=0000 CX=0014 DX=0000 SP=FFFE BP=0000 SI=0000 DI=0000

DS=1133 ES=1133 SS=1133 CS=1133 IP=0103 NV UP EI PL NZ NA PO NC

1133:0103 01C0 ADD AX,AX

-t

AX=0012 BX=0000 CX=0014 DX=0000 SP=FFFE BP=0000 SI=0000 DI=0000
```

```
DS=1133 ES=1133 SS=1133 CS=1133 IP=0105 NV UP EI PL NZ AC PE NC
1133:0105 89C3 MOV BX,AX
```

Figure 6 Loading and tracing the example ex2-3.com program

```
 -t
AX=0012 BX=0012 CX=0014 DX=0000 SP=FFFE BP=0000 SI=0000 DI=0000
DS=1133 ES=1133 SS=1133 CS=1133 IP=0107 NV UP EI PL NZ AC PE NC
1133:0107 89C1 MOV CX,AX
-t
AX=0012 BX=0012 CX=0012 DX=0000 SP=FFFE BP=0000 SI=0000 DI=0000
DS=1133 ES=1133 SS=1133 CS=1133 IP=0109 NV UP EI PL NZ AC PE NC
1133:0109 BE8001 MOV SI,0180
-t
AX=0012 BX=0012 CX=0012 DX=0000 SP=FFFE BP=0000 SI=0180 DI=0000
DS=1133 ES=1133 SS=1133 CS=1133 IP=010C NV UP EI PL NZ AC PE NC
1133:010C 8904 MOV [SI],AX DS:0180=1212
-t
AX=0012 BX=0012 CX=0012 DX=0000 SP=FFFE BP=0000 SI=0180 DI=0000
DS=1133 ES=1133 SS=1133 CS=1133 IP=010E NV UP EI PL NZ AC PE NC
1133:010E 46 INC SI
-t
AX=0012 BX=0012 CX=0012 DX=0000 SP=FFFE BP=0000 SI=0181 DI=0000
DS=1133 ES=1133 SS=1133 CS=1133 IP=010F NV UP EI PL NZ NA PE NC
1133:010F 891C MOV [SI],BX DS:0181=1200
-t
```

```
AX=0012 BX=0012 CX=0012 DX=0000 SP=FFFE BP=0000 SI=0181 DI=0000

DS=1133 ES=1133 SS=1133 CS=1133 IP=0111 NV UP EI PL NZ NA PE NC

1133:0111 46 INC SI

-t

AX=0012 BX=0012 CX=0012 DX=0000 SP=FFFE BP=0000 SI=0182 DI=0000

DS=1133 ES=1133 SS=1133 CS=1133 IP=0112 NV UP EI PL NZ NA PE NC

1133:0112 890C MOV [SI],CX DS:0182=0000

-t

AX=0012 BX=0012 CX=0012 DX=0000 SP=FFFE BP=0000 SI=0182 DI=0000

DS=1133 ES=1133 SS=1133 CS=1133 IP=0114 NV UP EI PL NZ NA PE NC

1133:0114 0D0A00 OR AX,000A
```

Figure 7. Loading and tracing the example ex2-3.com program (continue)

To verify that the data area pointed by the register pair DS:SI, we dump or display the data block using **D** command. We see that the data are stored in the right place.

```
 -d ds:180

1133:0180 12 12 12 00 72 65 61 64-73 20 66 72 6F 6D 20 61 ....reads from a

1133:0190 20 64 65 76 69 63 65 0D-0A 00 B3 39 42 52 45 41 device....9BREA

1133:01A0 4B 20 69 73 20 00 DC 39-56 45 52 49 46 59 20 69 K is ..9VERIFY i

1133:01B0 73 20 00 E8 39 45 43 48-4F 20 69 73 20 00 F5 39 s ..9ECHO is ..9

1133:01C0 6F 66 66 0D 0A 00 00 3A-6F 6E 0D 0A 00 08 3A 49 off....:on....:I

1133:01D0 6E 76 61 6C 69 64 20 70-61 74 68 20 6F 72 20 66 nvalid path or f

1133:01E0 69 6C 65 20 6E 61 6D 65-0D 0A 00 0F 3A 49 6E 76 ile name....:Inv

1133:01F0 61 6C 69 64 20 6E 75 6D-62 65 72 20 6F 66 20 70 alid number of p
```