VIDYALANKAR INSTITUTE OF

TECHNOLOGY DEPARTMENT OF

COMPUTER ENGINEERING

Lab Manual

Subject: MICROPROCESSOR

SEM- IV

Suvarna Bhat

2021-2022

Subject	MP
Semester	IV
Academic Year	2021-22
Software Requirements	TASM, DosBox
Hardware Requirements	Desktops
Theory Faculty In-charge	Suvarna Bhat
Practical Faculty In-charge	Suvarna Bhat

Vidyalankar Institute of Technology
DEPARTMENT OF COMPUTER
ENGINEERING LAB CODE

1. Students should report to the concerned labs as per the time table schedule.
2. Students who turn up late to the labs will in no case be permitted to perform the experiment scheduled for the day.
3. After completion of the experiment, certification of the concerned staff in-charge in the observation book is necessary.
4. Students should bring a note book of about 100 pages and should enter the readings/observations into the note book while performing the experiment.
5. The record of observations along with the detailed experimental procedure of the experiment performed in the immediate last session should be submitted and certified by the faculty member.
6. The group-wise division made in the beginning should be adhered to, and no mix up of student among different groups will be permitted later.
7. The components required pertaining to the experiment should be collected from the concerned Lab Assistants.
8. When the experiment is completed, students should disconnect the setup made by them, and should return all the components/instruments taken for the purpose. Any damage of the equipment or burn-out of components will be viewed seriously either by putting penalty or by dismissing the total group of students from the lab for the semester/year.
9. Students should be present in the labs for the total scheduled duration. Students are required to prepare thoroughly to perform the experiment coming to Laboratory. Procedure sheets/data sheets provided to the students groups should be maintained neatly and to be returned after the experiment.

Course Outcome

Use appropriate instructions to program microprocessor to perform various task.
Develop the program in assembly/ mixed language for Intel 8086 processor.
Demonstrate the execution and debugging of assembly/ mixed language program.

List of Experiments

Sr. No.	Name of Experiment	CO
1	Study of tools used for programming of microprocessors (TASM)	CO2
2	Develop Assembly language program using 8086 microprocessor for addition, subtraction of 16/32 bit numbers	CO2
3	Develop Assembly language program using 8086 microprocessor for 16bit multiplication, 16 bit division	CO2
4	Develop Assembly language program using 8086 microprocessor for finding largest/smallest number from block of 16 bit numbers	CO2
5	Develop a 8086 Assembly Language Program for Conversions– (a) Packed to Unpacked BCD (b) Unpacked to Packed BCD (c) Packed BCD to ASCII	CO2
6	Develop Assembly language program using 8086 microprocessor for block copy, block exchange with and without string instructions	CO2
7	Develop assembly language program using 8086 microprocessor for I/O using INT N	CO2
8	8086 Assembly Language Program to Count Odd and Even Numbers from the given block of Numbers	CO2
9	8086 Assembly Language Program to Arrange the Numbers in Ascending Order	CO2
10	8086 Assembly Language Program to check if user entered string is palindrome or not	CO2

EXPERIMENT NO. 1	
Title	Study of tools used for programming of microprocessors (TASM)
Outcome	CO2
Theory	Study Experiment
Conclusion	

EXPERIMENT NO. 2	
Title	Develop Assembly language program using 8086 microprocessor for addition, subtraction of 16/32 bit numbers.
Outcome	**CO2**
Algorithm	Algorithm: **Addition of 2, 16-bit numbers** 1) Start 2) Allocate some space for the result and operands in data segment 3) In code segment, store accumulator with 1st operand 4) Store B register with 2nd operand 5) Initialise C register with 0 6) Add the content of register B with accumulator and store the result in accumulator 7) If operation doesn't result in carry, goto 9 8) Increment content of C register 9) The result is stored in the required memory location 10) Stop **Subtraction of 2, 16-bit numbers** 1) Start 2) Allocate some space for the result and operands in data segment 3) In code segment, store accumulator with 1st operand 4) Store B register with 2nd operand 5) Initialise C register with 0 6) Subtract the content of register B from accumulator and store the result in accumulator 7) If operation doesn't result in carry, goto 9 8) Increment content of C register 9) The result is stored in the required memory location 10) Stop
Code	**Addition of 2, 16-bit numbers** **Labels** / **Mnemonics** / **Operands** start / MOV / AX, @data down / MOV / SUM, AX Program: .model small

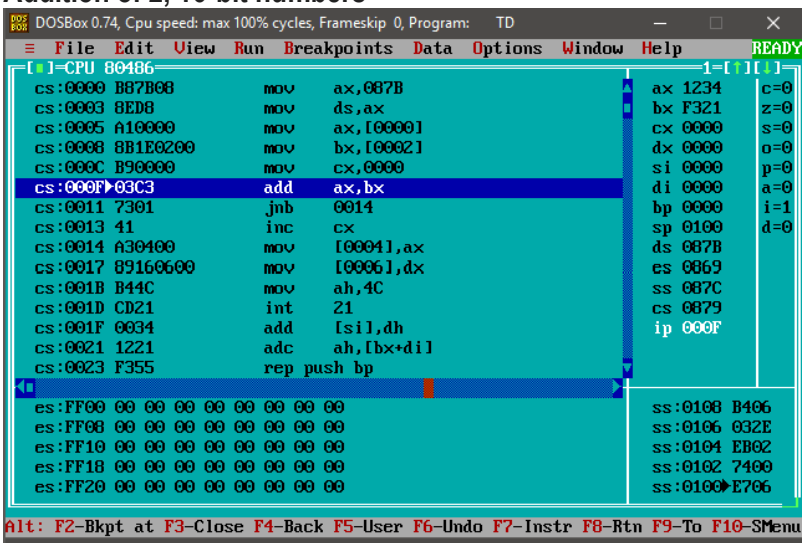
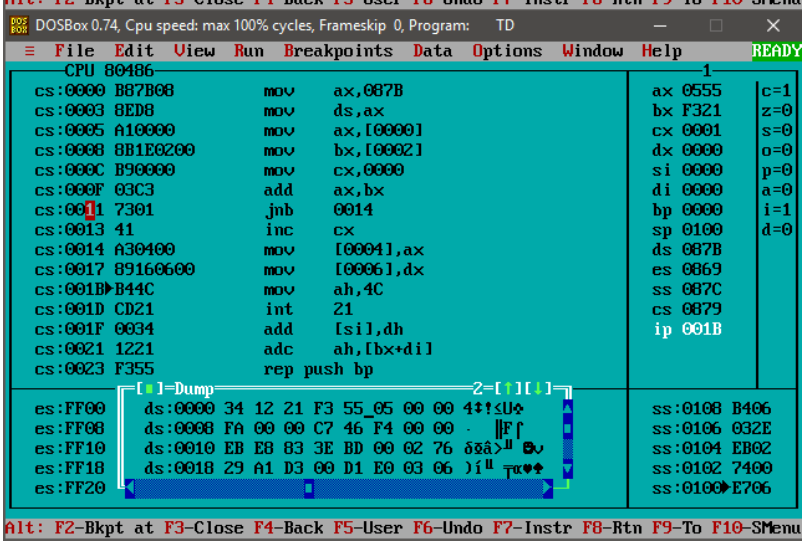
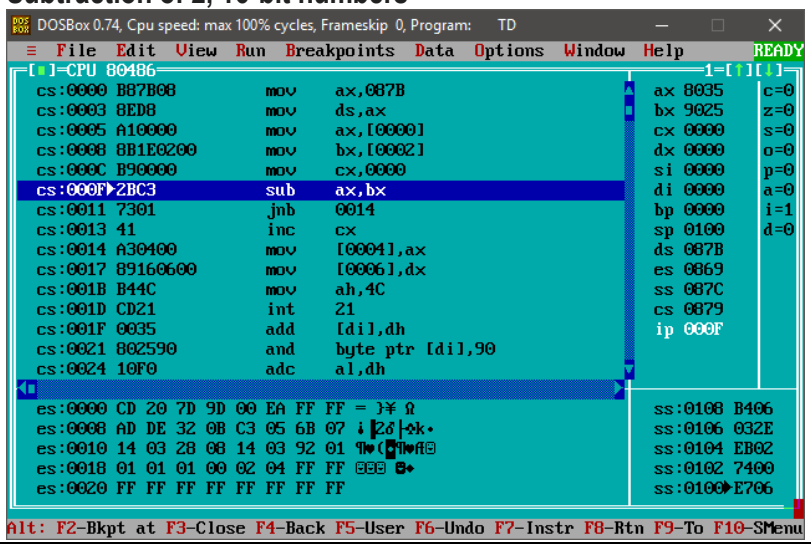
```
.stack 100H
.data
        n1 DW 1234H
        n2 DW 0F321H
        SUM DW ?
        ends
.code
        start:  MOV AX, @data
                MOV DS, AX
                MOV AX, n1
                MOV BX, n2
                MOV CX, 00H
                ADD AX, BX
                JNC DOWN
                INC CX
        DOWN:   MOV SUM, AX
                MOV SUM+2, DX
                MOV AH, 4CH
                INT 21H
        end     start
end
```

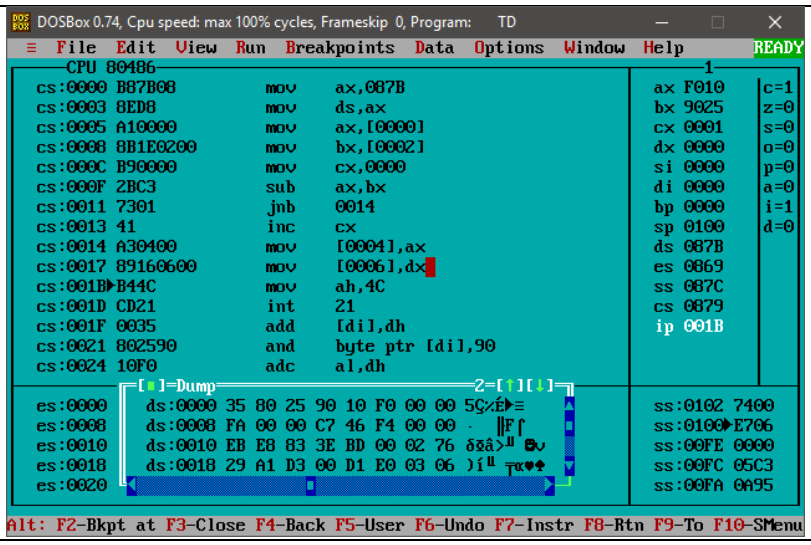
Subtraction of 2, 16-bit numbers

Labels	Mnemonics	Operands
start	MOV	AX, @data
down	MOV	DIFF, AX

Program:

```
.model small
.stack 100H
.data
        n1 DW 0F321H
        n2 DW 1234H
        DIFF DW ?
        ends
.code
        start:  MOV AX, @data
                MOV DS, AX
                MOV AX, n1
                MOV BX, n2
                MOV CX, 00H
                SUB AX, BX
                JNC DOWN
                INC CX
        DOWN:   MOV DIFF, AX
                MOV DIFF+2, DX
                MOV AH, 4CH
                INT 21H
```

	end start end
Output	**Addition of 2, 16-bit numbers**   **Subtraction of 2, 16-bit numbers** 



Conclusion

1) For storing 16-bit data in 8086, we use the data type word DW.
2) During both the operations, whenever the operation results in a carry, we increment the contents of C register to store the extra bit. Hence the whole result in case of addition is represented as CAX and during subtraction if C register is incremented, it implies the result is negative which can be confirmed observing the sign flag which would also be 1 (set).
3) Jump instruction can be used to create a 'if' condition
2) 4C is the service of int 21 that needs to be loaded in AH register for normal termination of program.

EXPERIMENT NO. 3	
Title	Develop Assembly language program using 8086 microprocessor for 16bit multiplication, 16 bit division.
Outcome	**CO2**
Algorithm	Algorithm: **Multiplication of 2, 16-bit numbers** 1) Start 2) Allocate some space for the result and operands in data segment 3) In code segment, store accumulator with 1st operand 4) Store B register with 2nd operand 5) Multiply the content of register B with accumulator. Higher order bits are stored in DX and lower order bits in AX. 6) The result is stored in the required memory location 7) Stop **Division of 2, 16-bit numbers** 1) Start 2) Allocate some space for the result and operands in data segment 3) In code segment, store accumulator with 1st operand 4) Store B register with 2nd operand 5) Subtract the content of register pair DXAX with register BX. Quotient is stored in AX and Remainder is stored in DX. 6) The result is stored in the required memory location Stop
Code	**Multiplication of 2, 16-bit numbers**

Labels	Mnemonics	Operands
Start	MOV	AX
Down	MUL	DS
	INT	BX
	ENDS	DX
	END	N1
	END START	N2
		PRO
		AH

Program:

```
.model small
.stack 100H
.data
        n1 dw 1234H
        n2 dw 000FH
        pro dw ?
        ends
.code
        START:          mov AX,@data
```

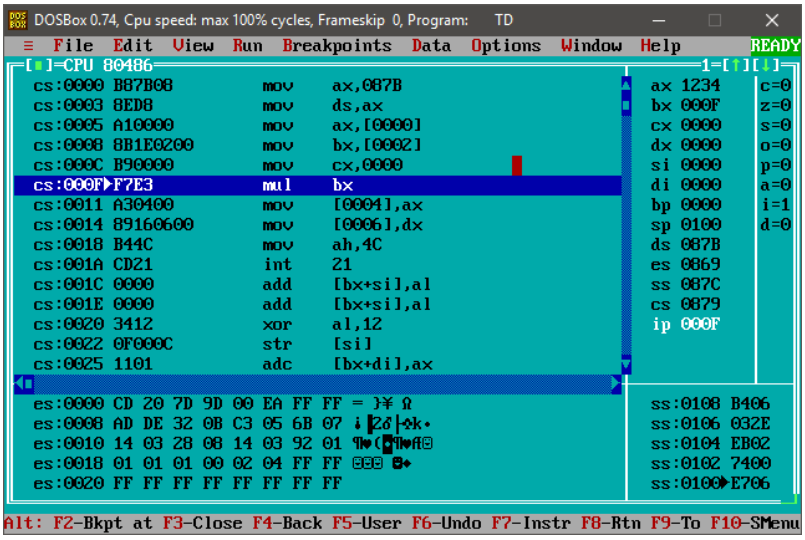
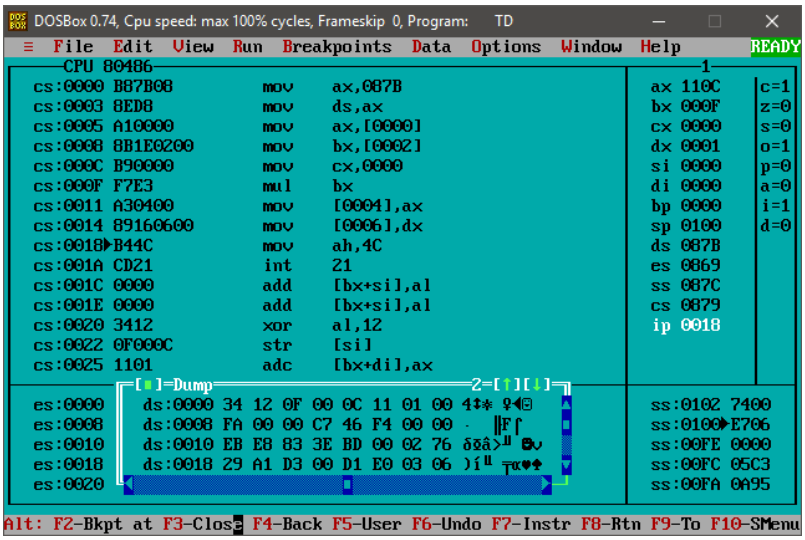
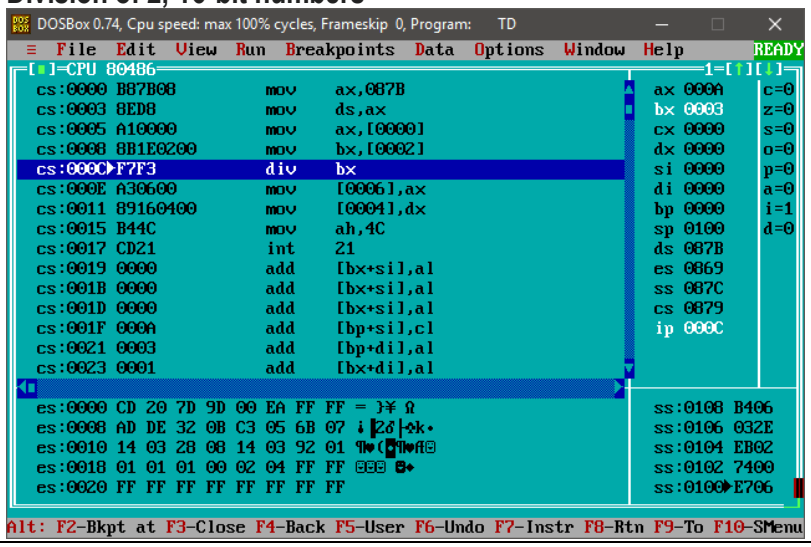
```
                    mov DS,AX
                    mov AX,n1
                    mov BX,n2
                    mov CX,00H
                    mul BX
        DOWN:       mov pro,AX
                    mov pro+2,DX
                    mov AH,4CH
                    int 21H
        end start
end
```

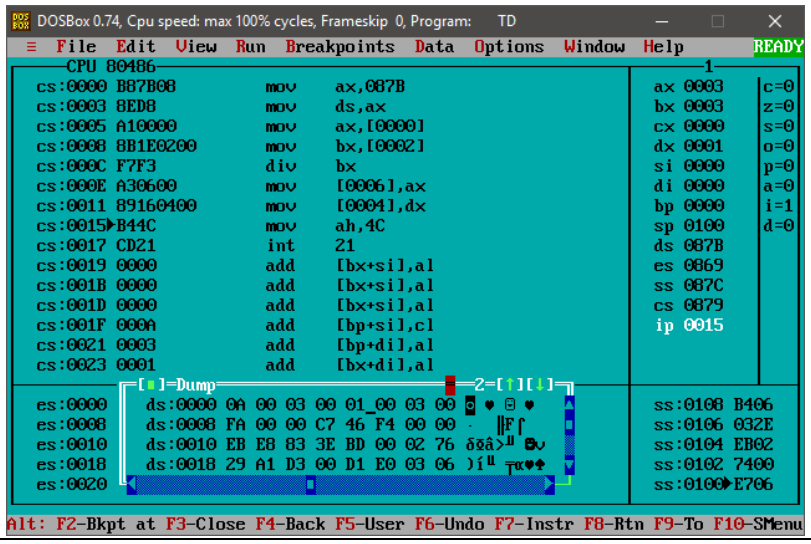
Division of 2, 16-bit numbers

Labels	Mnemonics	Operands
Start	MOV	AX
Down	DIV	DS
	INT	BX
	ENDS	CX
	END	N1
	END START	N2
		Q
		R
		AH

Program:

```
.model small
.stack 100H
.data
        n1 dw 000AH
        n2 dw 0003H
        r dw ?
        q dw ?
        ends
.code
        START:mov AX,@data
                    mov DS,AX
                    mov  AX,n1
                    mov  BX,n2
                    div bx
                    mov q,AX
                    mov r,DX
                    mov AH,4CH
                    int 21H
        end start
```

	end
Output	**Multiplication of 2, 16-bit numbers**   **Division of 2, 16-bit numbers** 



Conclusion

1) For storing 16-bit data in 8086, we use the data type word DW.
2) During multiplication, two 16-bit numbers when multiplied may produce a result larger than 16-bit. This is resolved internally by the microprocessor by storing the higher 16 bits in DX register and lower 16 bits in AXregister.
3) During division, there are 2 results generated i.e. quotient and remainder. The quotient is stored in AX while the remainder is stored in DX.
2) 4C is the service of int 21 that needs to be loaded in AH register for normal termination of program.

EXPERIMENT NO. 4	
Title	Develop Assembly language program using 8086 microprocessor for finding largest/smallest number from block of 16 bit numbers.
Outcome	**CO2**
Algorithm	Algorithm: **Finding the largest number from a block of 16-bit numbers** 1) Start 2) Allocate some space for the result, length of array and array in data segment 3) In code segment load the effective address of the start of array in SI register 4) Initialize CX with length of the array(number of elements) 5) Assume the first element as largest and move it to AL register and a memory location 'large' 6) Compare the contents of memory location small with AL register. 7) If large > AL, goto step 9 8) Move contents of AL in memory location 'large' 9) Increment the effective address in SI register such that it points to the next 16-bit number in array 10) Decrement CX register by 1 11) Repeat steps 6-10 until CX=0 12) Stop **Finding the smallest number from a block of 16-bit numbers** 1) Start 2) Allocate some space for the result, length of array and array in data segment 3) In code segment load the effective address of the start of array in SI register 4) Initialize CX with length of the array(number of elements) 5) Assume the first element as largest and move it to AL register and a memory location 'small' 6) Compare the contents of memory location small with AL register. 7) If small < AL, goto step 9 8) Move contents of AL in memory location 'small' 9) Increment the effective address in SI register such that it points to the next 16-bit number in array 10) Decrement CX register by 1 11) Repeat steps 6-10 until CX=0 12) Stop
Code	**Largest number from a block of 16-bit numbers**

Labels	Mnemonics	Operands
Start	MOV	AX
Y	LEA	@data
X	CMP	CX

	ENDS	larg
	END	arr
	END START	len
	JNG	si
	INC	DS
	LOOP	
	INT	

```
model small
.stack 100H
.data
        arr db 2h,4h,5h,11h,10h
        len dw $-arr
        larg db ?
        ends
.code
        START:          mov ax,@data
                        mov ds,AX
                        mov cx,len
                        lea si,arr
                        mov al,[si]
                        mov larg,al
        Y:              mov al,[si]
                        cmp al,larg
                        jng X
                        mov larg,al
        X:              inc si
                        loop Y
                        mov AH,4CH
                        int 21H
        end     start
end
```

Smallest number from a block of 16-bit numbers

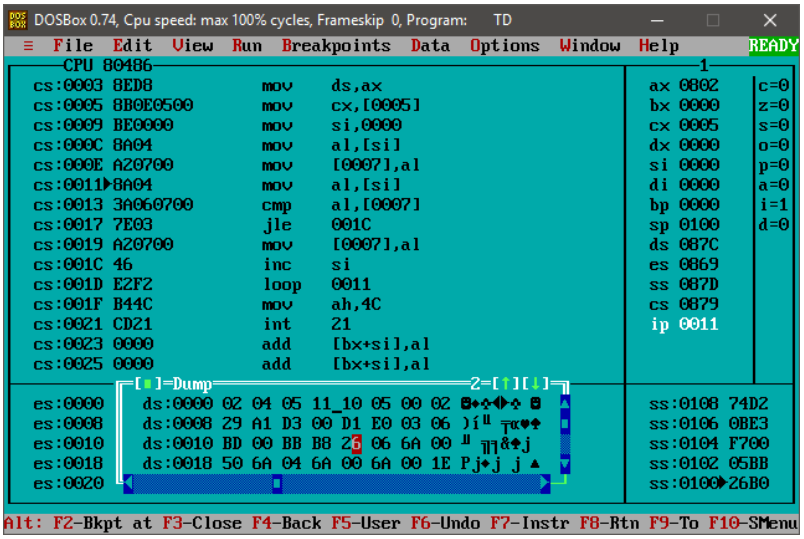
Labels	Mnemonics	Operands
Start	MOV	AX
Y	LEA	@data
X	CMP	CX
	ENDS	smal
	END	arr
	END START	len
	JG	si
	INC	DS
	LOOP	
	INT	

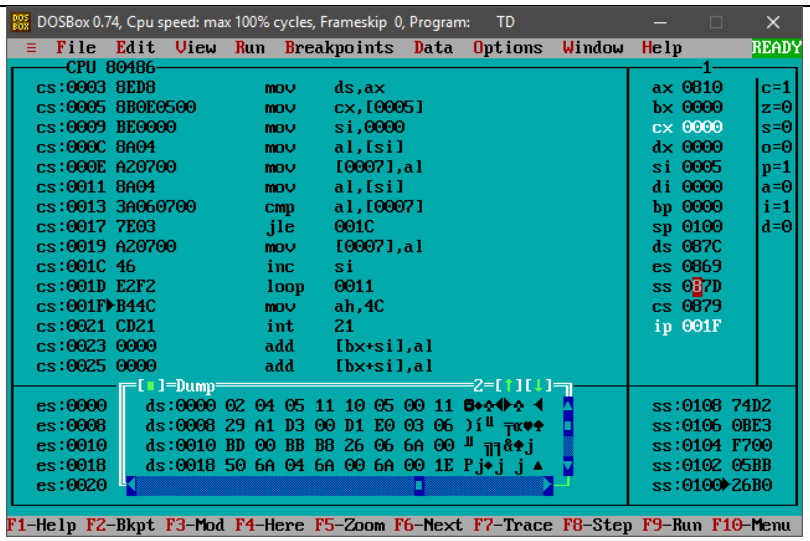
```
model small
.stack 100H
.data
        arr db 2h,4h,5h,11h,10h
        len dw $-arr
```

```
        smal db ?
        ends
.code
        START:      mov ax,@data
                    mov ds,AX
                    mov cx,len
                    lea si,arr
                    mov al,[si]
                    mov smal,al
        Y:          mov al,[si]
                    cmp al,smal
                    jg X
                    mov smal,al
        X:          inc si
                    loop Y
                    mov AH,4CH
                    int 21H
        end     start
end
```

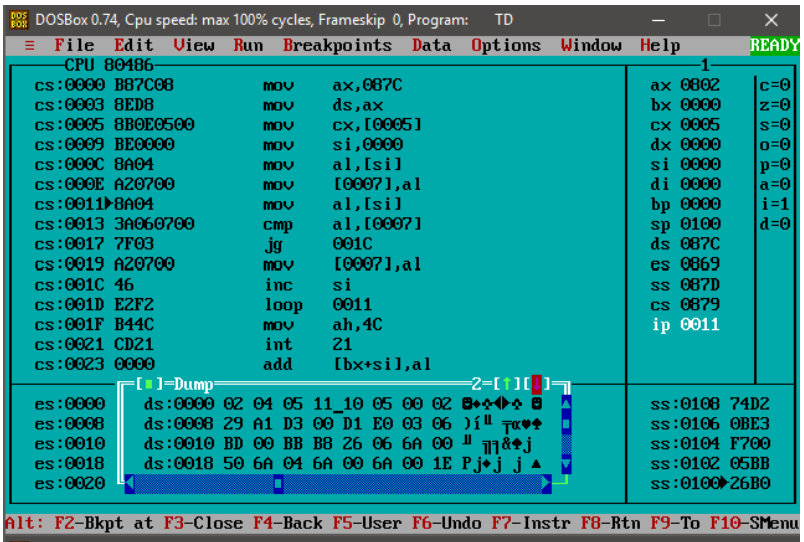
Output

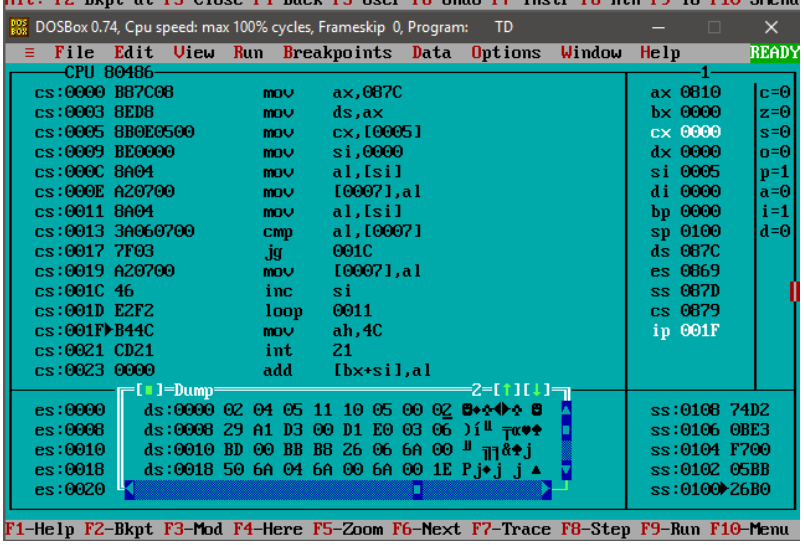
Largest number from a block of 16-bit numbers





Smallest number from a block of 16-bit numbers





Conclusion	1) For storing 16-bit data in 8086, we use the data type word DW. 2) LEA instruction is used to retrieve the base address of the array.

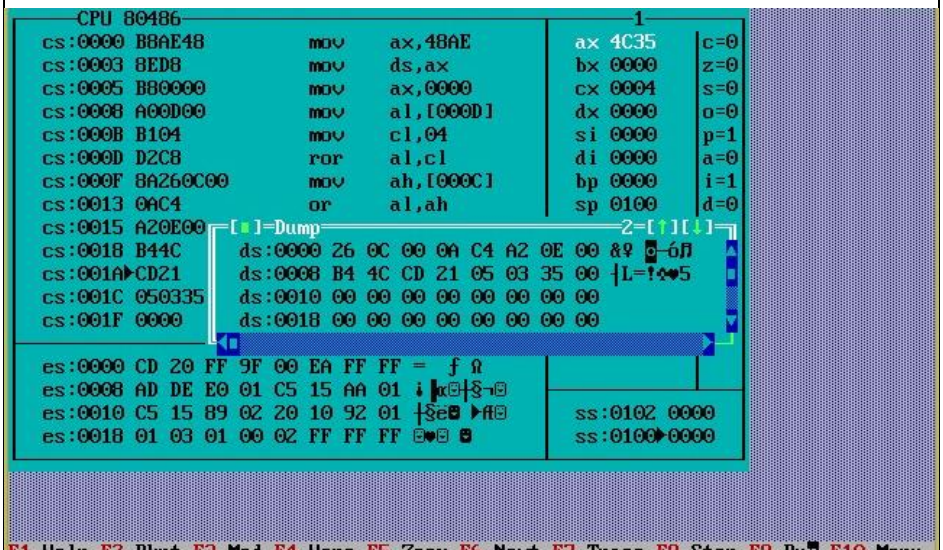
	3) CMP instruction performs comparison of 2 numbers by performing subtraction and interpreting its result. The result is not stored anywhere, only the flag register is affected accordingly. 4) LOOP instruction is used to simulate a loop. It does 2 operations: Decrement CX and check if CX=0,if not it Loops to a given label. 5) Jump instruction can be used to create a 'if' condition 6) 4C is the service of int 21 that needs to be loaded in AH register for normal termination of program.

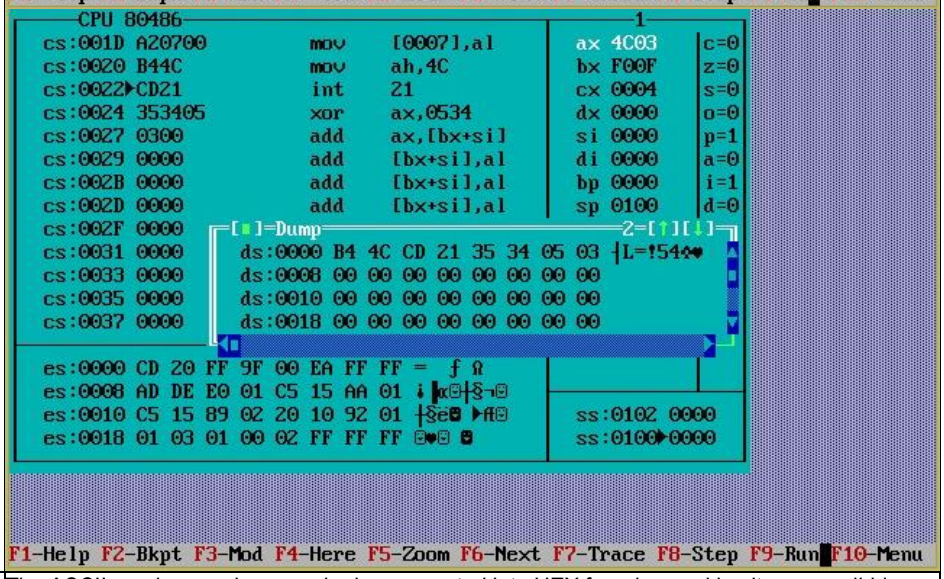
EXPERIMENT NO. 5	
Title	Develop a 8086 Assembly Language Program for Conversions– (a) Packed to Unpacked BCD (b) Unpacked to Packed BCD (c) Packed BCD to ASCII
Name	**Sanika Chavan**
Roll No	**19101A0053**
Outcome	**CO2**
Algorithm	

Code	(a)Packed to Unpacked BCD
	
	.model small
	.stack 100H
	.data cinf
	a db 35h ; store 1st no at location a
	ab1 ? ; define ub1 location location to save unpacked version of a
	ab2 ? ; define ub1 location location to save unpacked version of a
	ends
	.code
	start: mov ax,@data
	mov ds,ax
	mov ax,000h ;clear content of accumulator
	mov bl, 00fh ;move 0fh into reg bl
	mov bh, 0f0h ;move f0h in reg bh
	mov al, a ; move the first ASCII in al
	AND al,bl ; AND the content of al and bl
	mov ub1,al ; move the result of ANDing at location ub1
	mov al,a ;move the first ASCII no in al
	AND al,bh ; AND the content of al and bh
	mov cl,04h ; move 04h in reg cl
	ror al,cl ; shift the content of ch reg to left by 4 bits
	mov ub2,al ; move the result at location ub1
	mov ah, 4ch
	int 21h
	end start
	end
	
	(b)Unpacked to Packed BCD
	.model small
	.stack 100H
	.data
	a db 05h ; store 1st no at location a
	b db 03h ; store 2nd no at location b
	pb1 db ? ; define pb1 locatio to save packed version of a and b
	
	ends
	.code
	start: mov ax,@data
	mov ds,ax
	mov ax,000h ;clear content of accumulator
	mov al,b ; move the second unpacked in al
	mov cl,04h ; move 04h in reg cl
	ror al,cl ; shift the content of ch reg to left by 4 bits
	mov ah,a ; move the first number into register ah
	OR al,ah ; or the content of al with ah
	mov pb1,al
	mov ah, 4ch

```
        int 21h
        end start
end
```

Output





Conclusion

The ASCII number can be unpacked or converted into HEX form by masking its upper nibble. This masking can be done by two methods. One method is we can AND the ASCII number by 0FH or we can perform SAL/SHL operation 4 times. This SHL instruction shifts the content of the source to left and appends 0s in LSB.

The unpacked number can be packed by using OR operation.

For e.g., 01H and 02H are two numbers to be packed, we can shift one of the number 4 times towards left by using SAL/SHL instruction and OR shifted number with the another number i.e. if 01H is shifted left four times, it becomes 10H. Then OR it with 02H. The result will be 12H which is packed version of those two numbers.

EXPERIMENT NO. 6	
Title	Develop Assembly language program using 8086 microprocessor for block copy, block exchange with and without string instructions.
Outcome	**CO2**
Algorithm	**Copying Byte array from one location to another without using string instructions** 1) Start. 2) Allocate some space for the input array, output array and length of array in data segment. 3) Initialize CX with the length of the array. 4) Load the effective address of input string in SI and that of output string in DI. 5) Move the contents from the location whose offset is currently pointed by SI in Data segment to the location whose offset is currently pointed by DI in Data segment. 6) Increment SI,DI. 7) Decrement CX. 8) Repeat steps 5-7 until CX = 0. 9) Stop. **Copying Byte array from one location to another using string instructions** 1) Start. 2) Allocate some space for the input array, output array and length of array in data segment. 3) Initialize CX with the length of the array. 4) Load the effective address of input string in SI and that of output string in DI. 5) Overlap the Data Segment and Extra Segment in memory. 6) Using the REPNZ prefix for MOVSB instruction, the array is copied into destination location. 7) Stop. **Copying Word array from one location to another without using string instructions** 1) Start. 2) Allocate some space for the input array, output array and length of array in data segment. 3) Initialize CX with the length of the array. 4) Load the effective address of input string in SI and that of output string in DI. 5) Move the contents from the location whose offset is currently pointed by SI in Data segment to the location whose offset is currently pointed by DI in Data segment. 6) Increment SI and DI by 2. 7) Decrement CX by 2. 8) Repeat steps 5 & 6 until CX = 0. 9) Stop. **Copying Word array from one location to another using string instructions**

	1) Start. 2) Allocate some space for the input array, output array and length of array in data segment. 3) Initialize CX with the length of the array. 4) Load the effective address of input string in SI and that of output string in DI. 5) Overlap the Data Segment and Extra Segment in memory. 6) Using the REPNZ prefix for MOVSW instruction, the array is copied into destination location. 7) Stop.

Code	1. Copying Byte array from one location to another without using string instructions
	Labels Mnemonics Operands
	START MOV AX,@DATA
	REPEAT LEA DS
	INC CX
	LOOP LEN
	INT ARR1
	ARR2
	SI
	DI
	21H,4CH
	BL
	
	.model small
	.stack 100h
	.data
	arr1 db 02h,09h,06h,10h,07h
	arr2 db ?
	len dw $-arr1
	ends
	.code
	start: mov ax,@data
	mov ds,ax
	mov cx,len
	lea si,arr1
	lea di,arr2
	repeat: mov bl,[si]
	mov ds:[di],bl
	inc si
	inc di
	loop repeat
	mov ah,4ch
	int 21h
	ends
	end start
	2. Copying Byte array from one location to another using string instructions
	Labels Mnemonics Operands
	START MOV AX,@DATA
	LEA DS

```
        MOVSBCX
        REPNZ LEN
        INT     ARR1
                ARR2
                SI
                DI
                21H,4CH
                ES

.model small
.stack 100h
.data
        arr1 db 02h,09h,06h,10h,07h
        arr2 db ?
        len dw $-arr1
        ends
.code
        start:  mov ax,@data
                        mov ds,ax
                        mov es,ax
                        mov cx,len
                        lea si,arr1
                        lea di,arr2
                        repnz movsb
                        mov ah,4ch
                        int 21h
        ends
end start
```

3. Copying Word array from one location to another without using string instructions

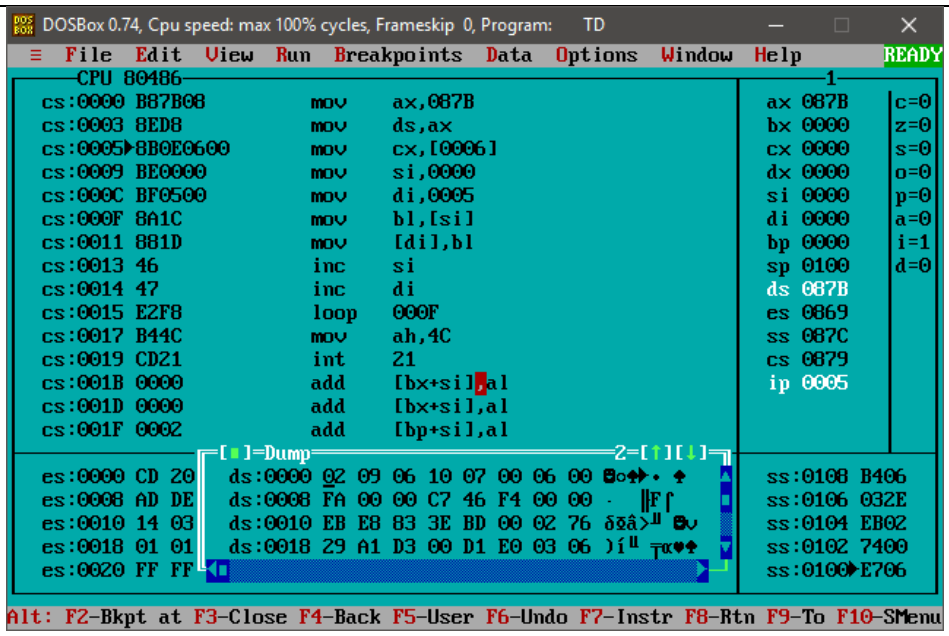
```
Labels  Mnemonics       Operands
START MOV       AX,@DATA
REPEAT          LEA     DS
        ADD     CX
        SUB     LEN
        JNZ     ARR1
        INT     ARR2
                SI
                DI
                21H,4CH,2H
                BX

.model small
.stack 100h
.data
        arr1 dw 0212h,0933h,0621h,1023h,0798h
        arr2 dw ?
        len dw $-arr1
        ends
```

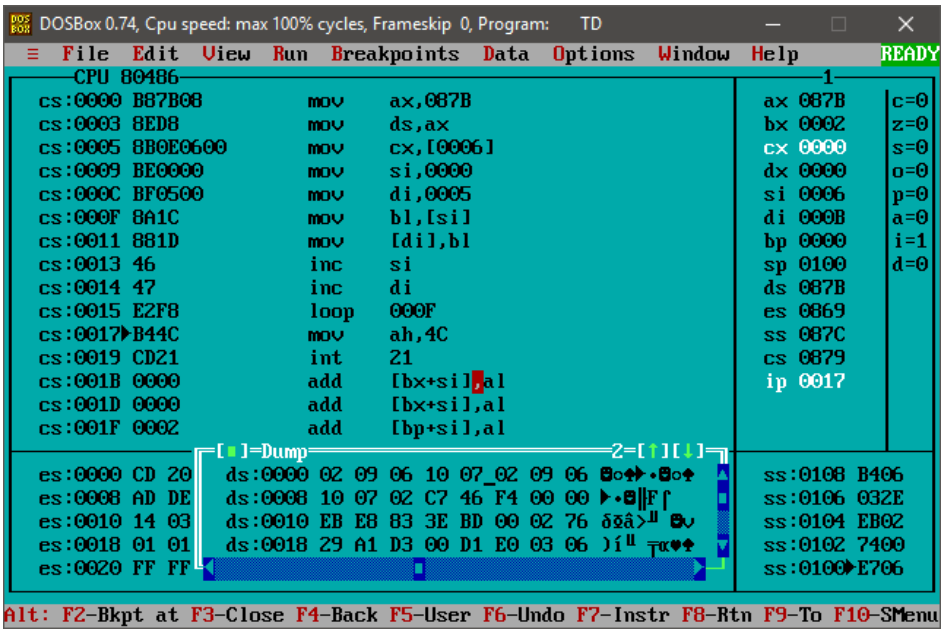
```
.code
        start:  mov ax,@data
                        mov ds,ax
                        mov cx,len
                        lea si,arr1
                        lea di,arr2
        repeat: mov bx,[si]
                        mov ds:[di],bx
                        add si,2h
                        add di,2h
                        sub cx,2h
                        jnz repeat
                        mov ah,4ch
                        int 21h
        ends
end start
```

4. Copying Word array from one location to another using string instructions

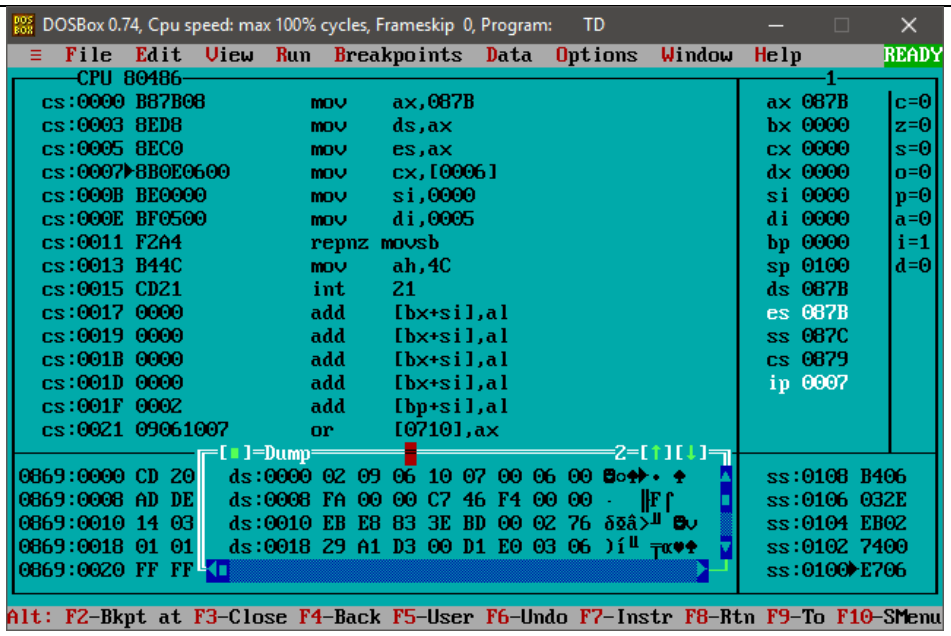
```
Labels  Mnemonics       Operands
START   MOV     AX,@DATA
        LEA     DS
        MOVSW           CX
        REPNZ LEN
        INT     ARR1
                ARR2
                SI
                DI
                21H,4CH
                ES

.model small
.stack 100h
.data
        arr1 dw 0212h,0933h,0621h,1023h,0798h
        arr2 dw ?
        len dw $-arr1
        ends
.code
        start:  mov ax,@data
                        mov ds,ax
                        mov es,ax
                        mov cx,len
                        lea si,arr1
                        lea di,arr2
                        repnz movsw
                        mov ah,4ch
                        int 21h
        ends
end start
```

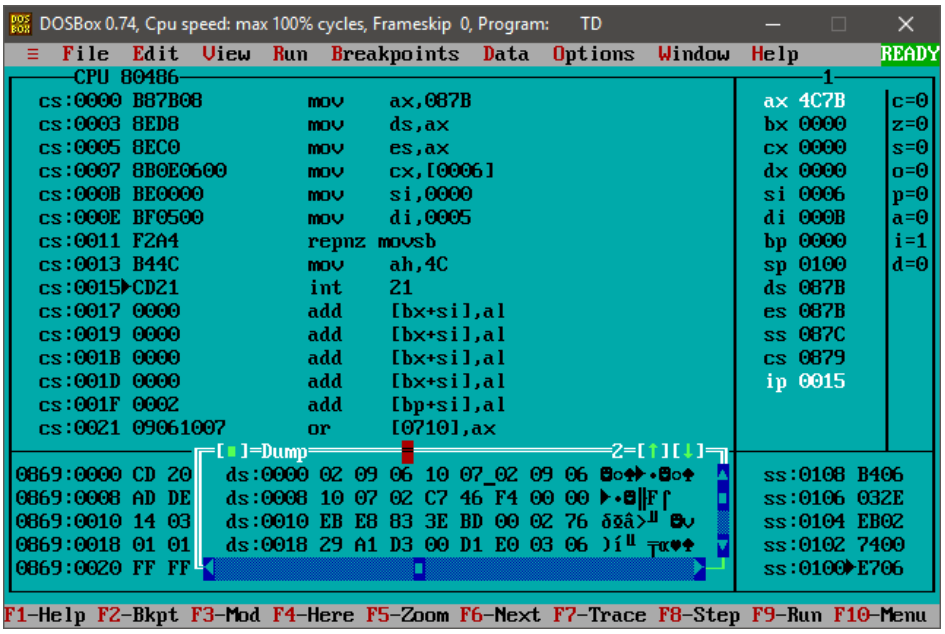
Output	**Copying Byte array from one location to another without using string**



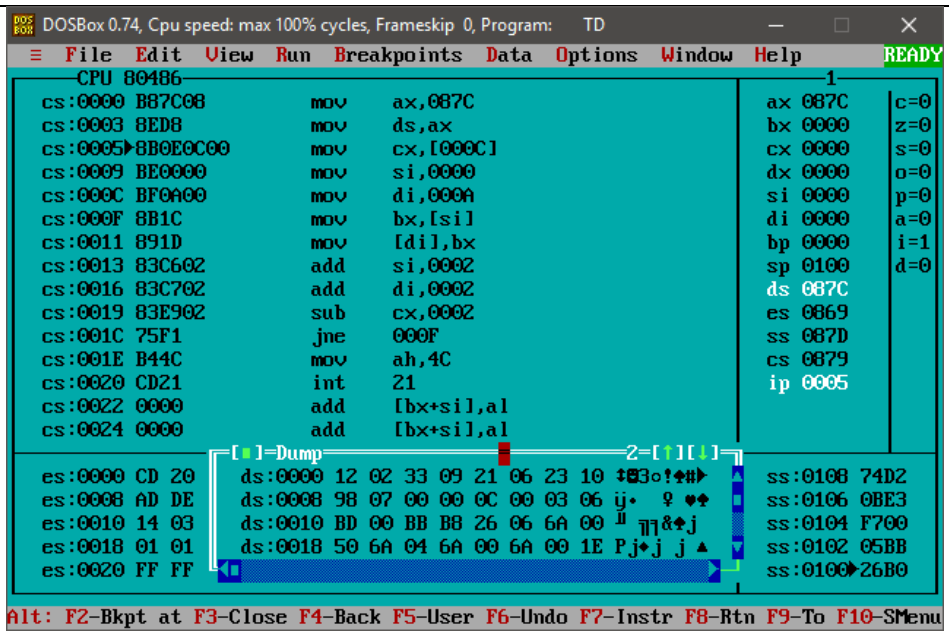


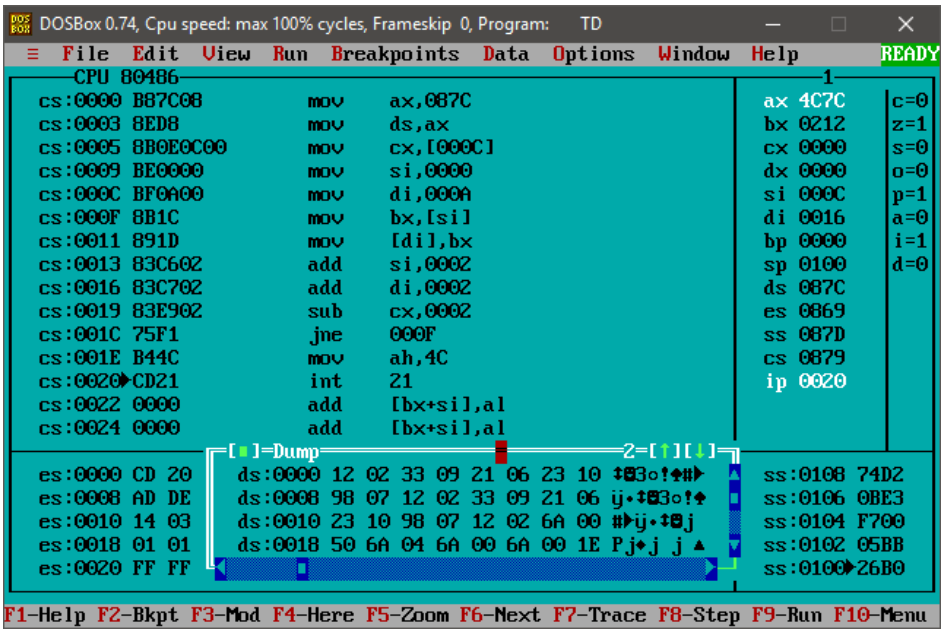
Copying Byte array from one location to another using string



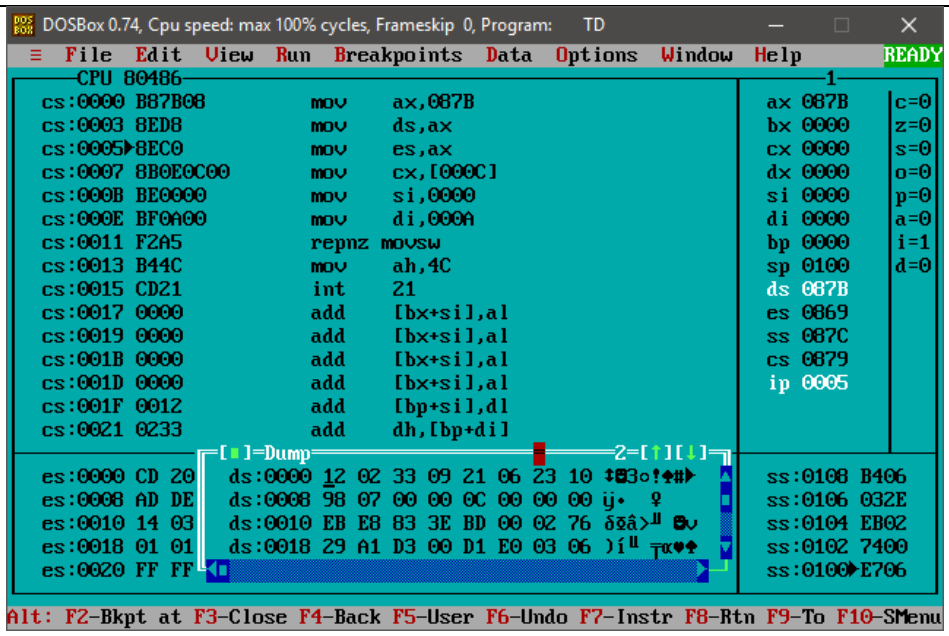


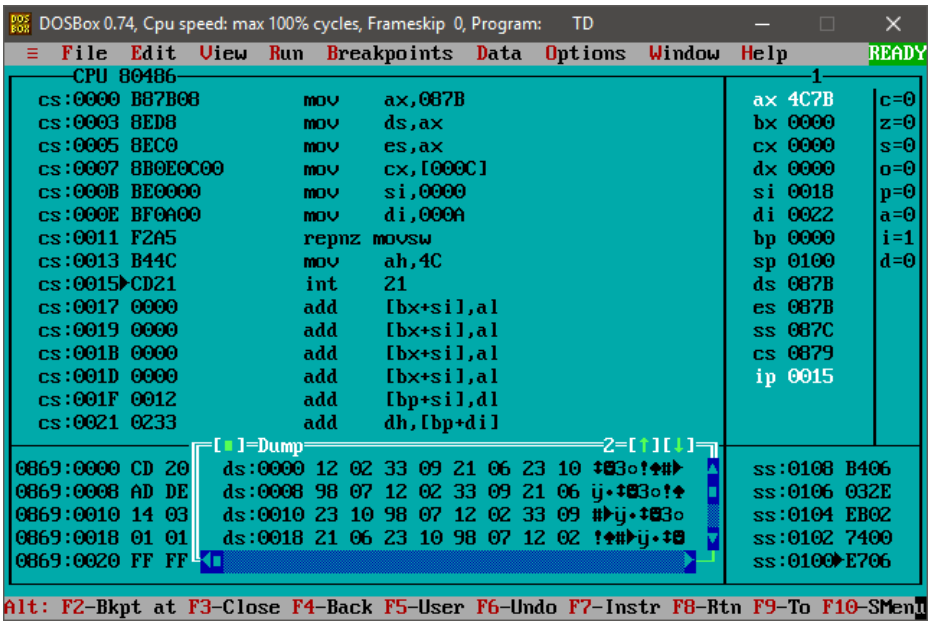
Copying Word array from one location to another without using string





Copying Word array from one location to another using string





Conclusion	1) Segment override prefix is used to change the default segment when physical address is calculated internally by the microprocessor. 2) String instructions by default use the Data segment for input and Extra segment for output. 3) The instruction prefix REPNZ can only be used with string instructions and makes the processor execute the following string instruction till the whole input string is processed.

EXPERIMENT NO. 7	
Title	Develop assembly language program using 8086 microprocessor for I/O using INT N.
Outcome	**CO2**
Algorithm	Algorithm **Perform addition of 2 numbers using I/O operations** 1) Start 2) Allocate some space for the messages to be displayed in the data segment. 3) Display message to accept 1st input. 4) Accept the input from user and store it in some register. 5) Display message to accept 2nd input. 6) Accept the 2nd input from user and store in in some register. 7) Perform addition and store the result in DL register. 8) Stop.

Code

Perform addition of 2 numbers using I/O operations

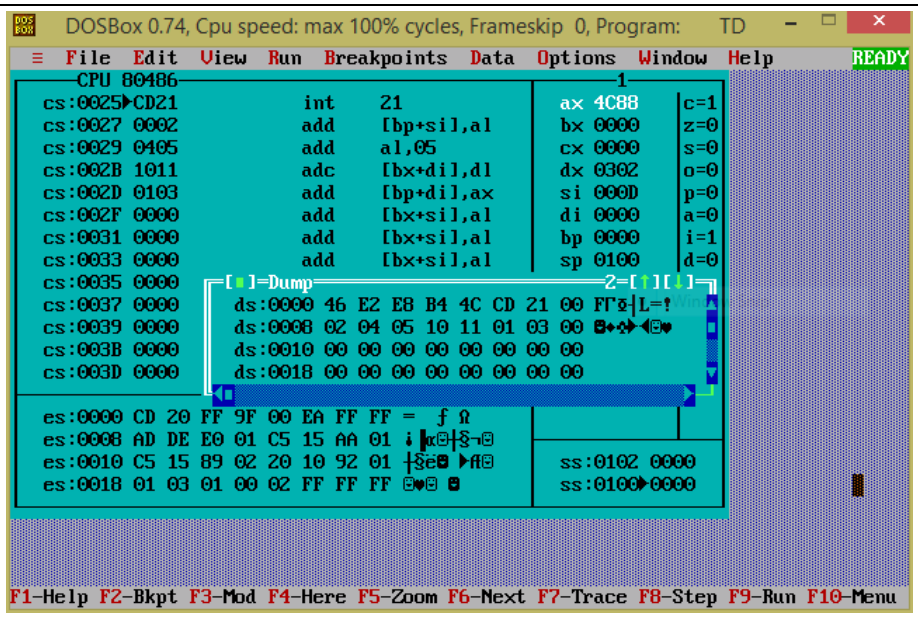
Labels	Mnemonics	Operands
START	MOV	AX,@DATA
	LEA	DS
	INC	AH
	INT	msg1,msg2,msg3
	SUB	CL
	ADD	BL
	AAA	DL
		DX
		AL
		21H,4CH
		09H,01H,30H,13H,10,02H

```
.model small
.stack 100h
.data
        msg1 db 'Enter the 1st number: $'
        msg2 db 'Enter the 2nd number: $'
        msg3 db 'Result: $'
        ends
.code
        start:          mov ax,@data
                        mov ds,ax
                        mov ah,09h
                        lea dx,msg1
                        int 21h
                        mov ah,01h
                        int 21h
                        sub al,30h
                        mov bl,al

                        mov DL, 10      ;printing new line
                        mov AH, 02h
```

```
                    int 21h mov
                    DL, 13
                    mov AH, 02h
                    int 21h

                    mov  ah,09h
                    lea  dx,msg2
                    int 21h
                    mov  ah,01h
                    int 21h
                    sub al,30h
                    mov cl,al

                    mov DL, 10      ;printing new line
                    mov AH, 02h
                    int 21h
                    mov DL, 13
                    mov AH, 02h
                    int 21h

                    mov  ah,09h
                    lea  dx,msg3
                    int 21h

                    add  cl,bl
                    mov al,cl
                    aaa
                    add al,30h
                    mov dl,al

                    mov ah,02h
                    int 21h

                    mov ah,4ch
                    int 21h
          end       start
end
```

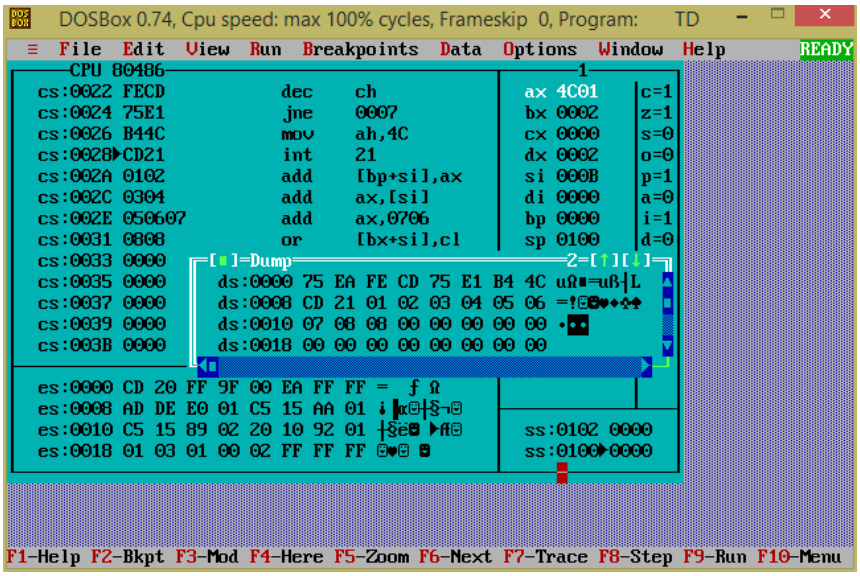
Output	DOSBox 0.74-2, Cpu speed: 3000 cycles, Frameskip 0, Program: DOSBOX C:\TASM\BIN>tasm ipadd.asm Turbo Assembler Version 4.1 Copyright (c) 1988, 1996 Borland International Assembling file: ipadd.asm Error messages: None Warning messages: None Passes: 1 Remaining memory: 467k C:\TASM\BIN>tlink ipadd Turbo Link Version 7.1.30.1. Copyright (c) 1987, 1996 Borland International C:\TASM\BIN>ipadd Enter the 1st number: 3 Enter the 2nd number: 5 Result: 8 C:\TASM\BIN>
Conclusion	1) DX is used to store the offset of the message which is to be displayed on the output screen. 2) By default the input taken from the user is stored in AL. 3) Combination of various data stored in AH and interrupts allow us to make use of different services like display message, display data in register, display new line etc.

EXPERIMENT NO. 8	
Title	8086 Assembly Language Program to CountOddandEvenNumbersfromthegivenblockof Numbers
Outcome	**CO2**
code	.model small .stack 100H .data array db 02h,04h,05h,10h,11h oddcnt db ? evencnt db ? ends .code start: mov ax,@data mov ds,ax mov cx,05h lea si,array next: mov al,[si] roral,01h ;rotaterightarrayelementpresentinalsoits lsb goes intoCF jnceven ; if CF =0 goto lable even inc dl ;elsedue toCF=1 incrementoddcount jmp ahead ; goto nextelement even:incdh ;due toCF=0 increment evencountinregdh movoddcnt,dl;copy countindlreg intovariable oddcount mov evencnt,dh ; copy count in dh reg into variable evencount ahead:incsi ;increment array pointer to addressnext element loop next;dec byte counter cx andifcx not=0 gotolable next mov ah,4ch int 21h end start end

Output	
Conclusion	To find the number of even and odd numbers in given string, we can make use of RCR instruction which rotates the all the bits in a specified word or byte some number of bit positions to the right. The operation is circular because the LSB of the operand is rotated into the carry flag and the bit in the carry flag is rotated around into the MSB of the operand. Thus by checking carry flag we can increment the content of location where number of even numbers is to be saved if carry flag is reset. Then we can also find the number of odd numbers in string by subtracting number of even numbers from the string length.

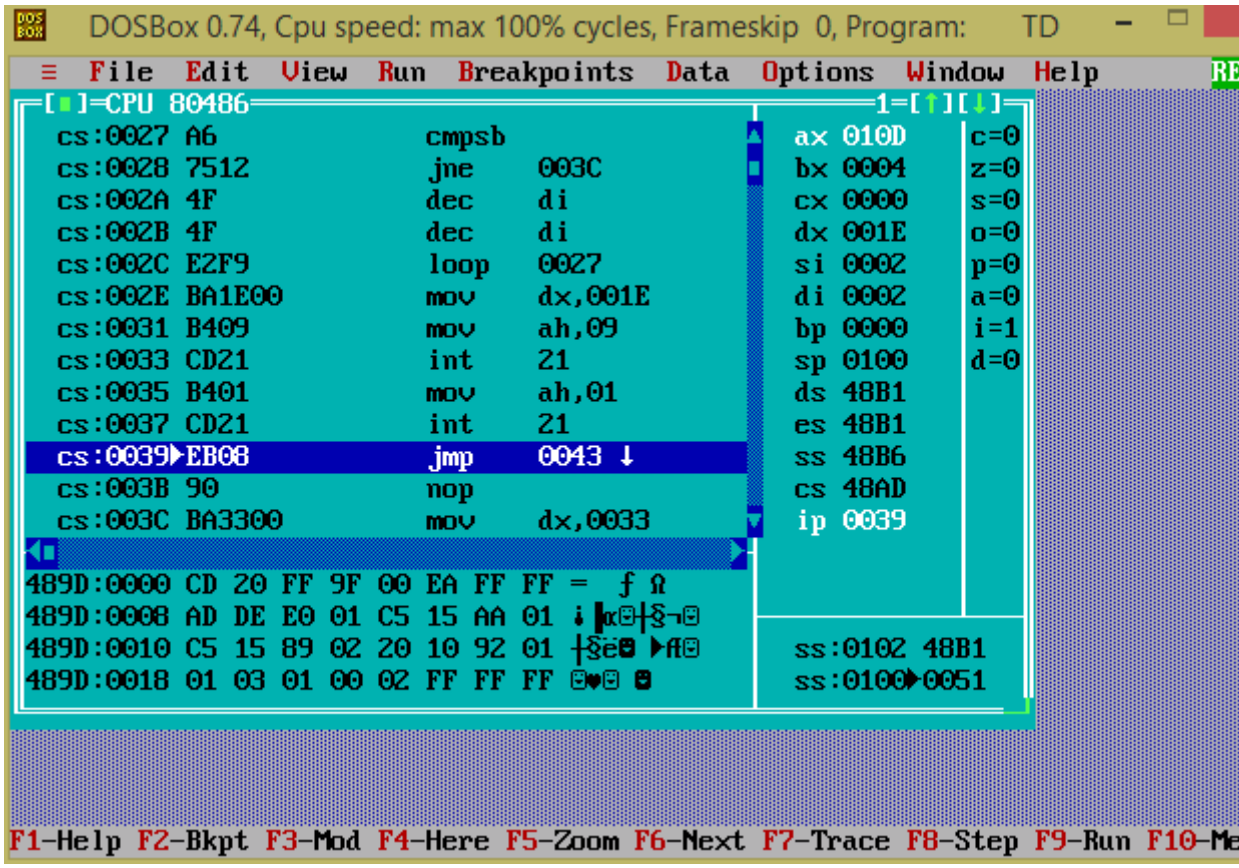
EXPERIMENT NO. 9	
Title	8086 Assembly Language Program to Arrange the Numbers in Ascending Order .
Outcome	**CO2**
Code	

```
.model small
.stack 100H
.data
arr db 4H,3H,7H,2H,5H,1H,8H,6H
len db $-arr
.code
        START:      MOV AX, @DATA
                    MOV DS,AXMOVCH,
                    07H

        NEXTPASS:   MOV CL, CH
                    LEA SI, arr

        NEXTCMP:    MOV AL, [SI]
                    MOV BL, [SI+1]
                    CMP AL, BL JC
                    NOSWAP MOV
                    DL, [SI+1] XCHG
                    [SI], DL MOV
                    [SI+1], DL

        NOSWAP: INC SI
                    DEC CL
                    JNZ NEXTCMP DEC
                    CH
                    JNZNEXTPASS MOV
                    AH,4CH INT 21H
                      END start
END
```

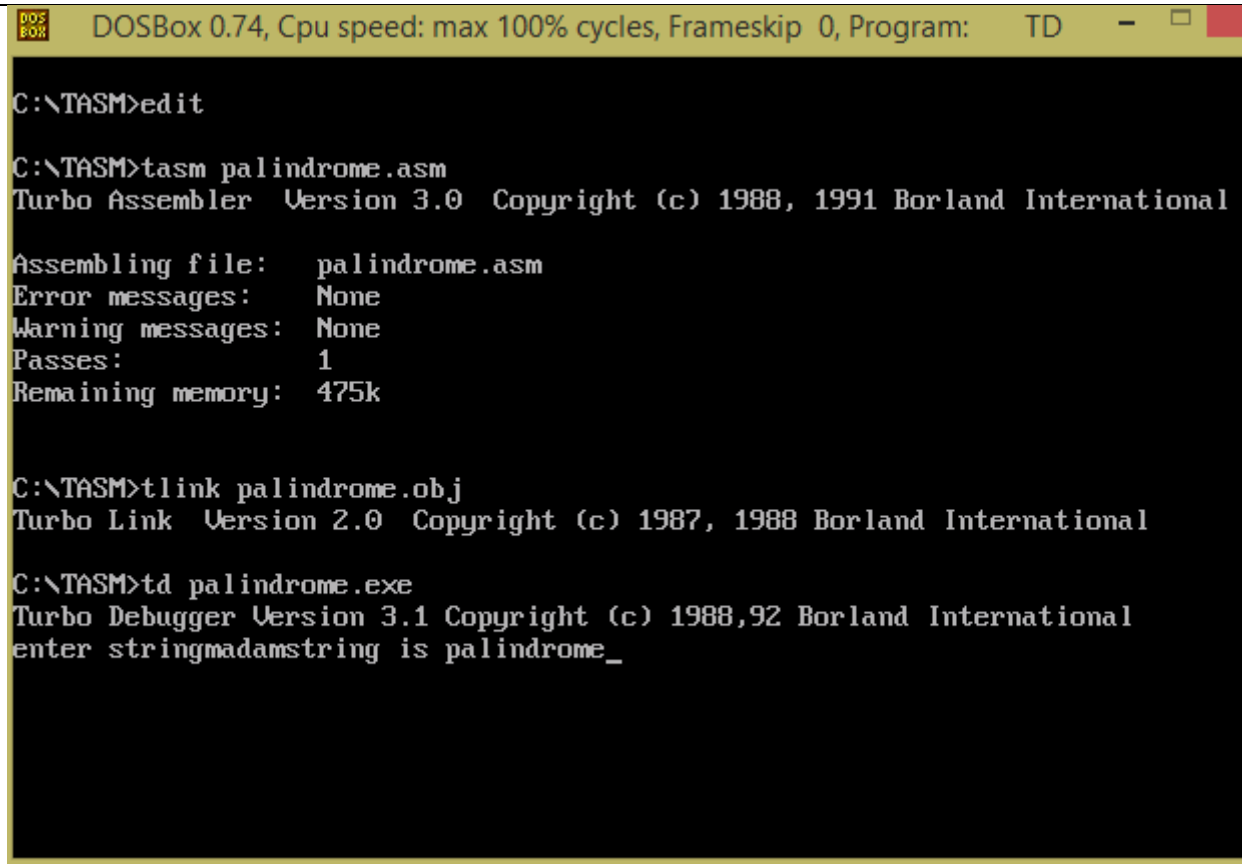
Output	
Conclusion	To find the smallest number in given string, we can make use of CMP instruction which will compare the number of one memory location copied in AX with the number of other memory location. If the result of comparison does not generate carry, then the number stored in second memory location is smaller than the number in first memory location which is copied in AX. But, as our area of interest is the smallest number stored in second memory location; we will copy it to AX and then further cycle of comparison process will be continued till counter becomes zero. Then at the end, the number in AX is the smallest number among all the string elements

EXPERIMENT NO. 10	
Title	8086 Assembly Language Program to check if user entered string is palindrome or not
Outcome	**CO2**
code	

```
.model small
.stack 100h
.data
        Arr db 9 dup(00h)
        msg db 'enter string$'
        msg1 db 'string is palindrome$'
        msg2 db 'string is not palindrome$'
        ends
.code
start:  mov ax,@data
        mov ds,ax
        mov es,ax
        mov cx,0005h
        mov bx,0000h
to_read_str:lea dx,msg
            mov ah,09h
            int 21h
nxt_chr:mov ah,01h
        int 21h
        mov [bx],al
        inc bx
        loop nxt_chr
        dec bx
        mov si,0000h
        mov di,bx
        mov cx,0002h
        CLD
nxt_cmp:cmpsb
        jne ahead
        dec di
        dec di
        loop nxt_cmp
        lea dx,msg1
        mov ah,09h
        int 21h
        mov ah,01h
        int 21h
        jmp stop
ahead:  lea dx,msg2
        mov ah,09h
        int 21h
stop:   mov ah,4ch
        int 21h
        ends
end start
```

Output	

DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TD
File Edit View Run Breakpoints Data Options Window Help
CPU 80486
cs:0027 A6 cmpsb
cs:0028 7512 jne 003C
cs:002A 4F dec di
cs:002B 4F dec di
cs:002C E2F9 loop 0027
cs:002E BA1E00 mov dx,001E
cs:0031 B409 mov ah,09
cs:0033 CD21 int 21
cs:0035 B401 mov ah,01
cs:0037 CD21 int 21
cs:0039 EB08 jmp 0043
cs:003B 90 nop
cs:003C BA3300 mov dx,0033
489D:0000 CD 20 FF 9F 00 EA FF FF
489D:0008 AD DE E0 01 C5 15 AA 01
489D:0010 C5 15 89 02 20 10 92 01
489D:0018 01 03 01 00 02 FF FF FF
ax 010D c=0
bx 0004 z=0
cx 0000 s=0
dx 001E o=0
si 0002 p=0
di 0002 a=0
bp 0000 i=1
sp 0100 d=0
ds 48B1
es 48B1
ss 48B6
cs 48AD
ip 0039
ss:0102 48B1
ss:0100 0051
F1-Help F2-Bkpt F3-Mod F4-Here F5-Zoom F6-Next F7-Trace F8-Step F9-Run



Conclusion	Palindrome check for a given string in 8086 Assembly language. The program prompts the user for a string and checks whether the given string is a palindrome displays the appropriate message.