

**FR. Conceicao Rodrigues College of Engineering**  
**Department of Computer Engineering**

## 1. Addition of Two 8/16/32 bit numbers

### 1. Course, Subject & Experiment Details

Academic Year	2023-24	Estimated Time	Experiment No. 1– 02 Hours
Course & Semester	S.E. (Comps) – Sem. IV	Subject Name	Microprocessor
Chapter No.	2	Chapter Title	Instruction Set and Programming
Experiment Type	Software	Subject Code	CSC405

### Rubrics

Timeline (2)	Practical Skill & Applied Knowledge (2)	Output (3)	Postlab (3)	Total (10)	Sign

### 2. Aim & Objective of Experiment

#### TO ADD TWO 8/16/32 BIT NUMBERS

**Objective :** Program involves storing the two 8-bit no in memory locations and adding them taking into consideration the carry generated. The objective of this program is to give an overview of arithmetic instructions of 8086 for 8-bit operands

### 3. Software Required

TASM Assembler

## 4 . Brief Theoretical Description

- Pre-Requisites:**
1. Instructions of microprocessor 8086
  2. Addressing mode of microprocessor 8086.
  3. Knowledge of TASM directories.

**Theory:** The addressing modes used in program are:

- 1) Direct addressing mode: in this mode address of operand is directly specified in the instruction. This address is offset address of the segment being indicated by an instruction.

E.g. `MOV AL,[2000h]`

$$EA = DS \times 10H + 2000H$$

- 2) Register Addressing Mode: In this mode operand are specified using registers. Instructions are shorter but operations cannot be identified looking at instruction.

E.g. `MOV CL, DL`

- 3) Based Indexed Addressing Mode: The operand address is calculated using base register and index register.

E.g. `MOV DX, [BX + SI]` moves word from address pointed by BX + SI in data segment to DX.

$$EA = DS \times 10H + BX + SI$$

- 4) Base indexed plus displacement: In this mode address of operand is calculated using base register , index register and displacement.

E.g. `MOV CX, [BX+DI+10h]`

This moves a word from address pointed by BX + DI +10h of segment to CX.

## 5. Algorithm:

1. Initialize the data segment.
2. Store two 8/16 -bit numbers in memory locations.
3. Move the 1<sup>st</sup> number in any one of the general purpose register.
4. Move the 2<sup>nd</sup> number in any other general purpose register.
5. Add the 2 numbers.
6. Store the result in memory location.
7. Check for carry flag. If carry flag is set then store '1' as MSB of result.
8. Stop

## 6. Conclusion:

### 8 bit addition:-

The screenshot shows the DOSBox Emulator interface. The assembly code window displays the following program:

```
1 .8086
2 model small
3 .data
4 A db 8FH
5 B db 9DH
6 sum db ?
7 carry db ?
8 .code
9 start:
0     MOV AX,@data
1     MOV DS,AX
2     MOV AL,A
3     MOV BL,B
4     ADD AL,BL
5     JNC l1
6     INC carry
7     l1: MOV sum,AL
8     MOV AH,4CH
9     INT 21H
0 end start
```

The CPU register window shows the following values:

Register	Value	Description
ax	070B	c=0
bx	0001	z=0
cx	0314	s=0
dx	0006	o=0
si	000C	p=0
di	0000	a=0
bp	0314	i=0
sp	0318	d=0
ds	F000	
es	3002	
ss	070B	
cs	0000	
ip	0000	

The memory dump window shows the following memory dump:

Address	Value	Content
006C:0000	CD 20 7D 9D 00 EA FF FF	INT 21H
006C:0008	AD DE 32 0B C3 05 6B 07	push ds
006C:0010	14 03 28 0E 14 03 92 01	push cx
006C:0018	FF FF FF FF FF FF FF FF	int 21H
007B:0002	0000	
007B:0000	0000	

## 16 bit addition:-

The screenshot shows the DOSBox environment running TASM/BIN. The assembly code in the editor window is:

```
.8086
.model small
.data
4 A dw 8F0EH
5 B dw 9D8CH
6 sum dw ?
7 carry dw ?
.code
9 start:
10    MOV AX,@data
11    MOV DS,AX
12    MOV AX,A
13    MOV BX,B
14    ADD AX,BX
15    JNC l1
16    INC carry
17 l1: MOV sum,AX
18    MOV AH,4CH
19    INT 21H
20 end start
```

The CPU window shows the assembly code and its corresponding opcodes. The registers window shows the state of the CPU registers:

ax	070B	c=0
bx	0001	z=0
cx	0314	s=0
dx	0006	a=0
si	0000	p=0
di	0000	a=0
bp	0314	i=0
sp	0310	d=0
ds	F000	
es	3002	
ss	070B	
cs	0000	
ip	0000	

The stack window shows the current stack state.

## Postlab:

1. Write a program for addition of two 32 bit numbers ,execute and take the screen shots of the results.

.8086

.model small

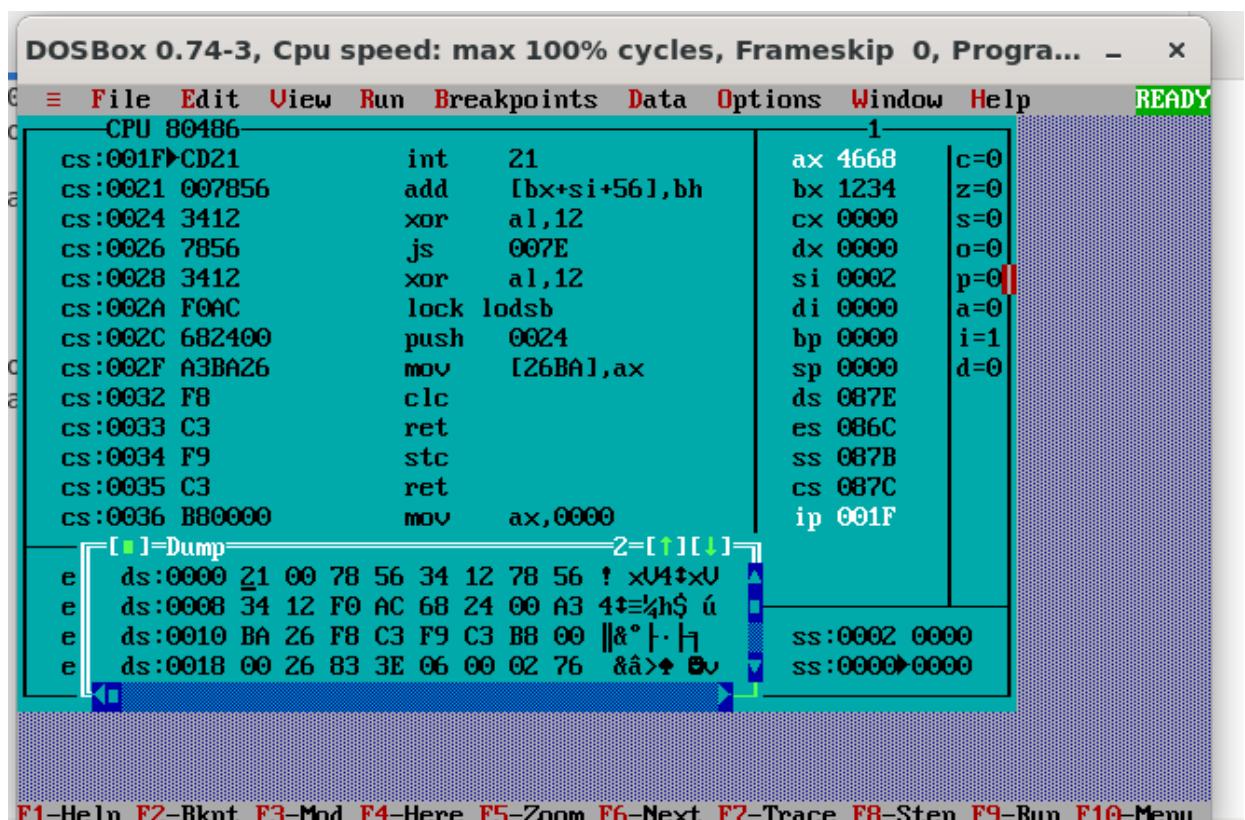
.data

num1 dd 12345678H

num2 dd 12345678H

suml dw ?

```
sumh dw ?  
.code  
start:  
    MOV AX, @data  
    MOV DS, AX  
    LEA SI, num1  
    MOV AX, [SI]  
    MOV BX, [SI+04H]  
    ADC AX, BX  
    MOV suml, AX  
  
    MOV AX, [SI+02H]  
    MOV BX, [SI+06H]  
    ADC AX, BX  
    MOV sumh, AX  
    MOV AH, 46H  
    INT 21H  
end start
```



2. Write a program to Subtract two 16 bit numbers.

.8086

.model small

.data

A dw 2456H

B dw 3280H

subt dw ?

burrow dw ?

.code

start:

MOV AX, @data

MOV DS, AX

MOV AX, A

MOV BX, B

SBB BX, AX

JNC skip

INC burrow

skip: MOV subt, BX

MOV AH, 4CH

INT 21H

end start

DOSBox 0.74-3, Cpu speed: max 100% cycles, Frameskip 0, Program - x

CPU 80486

Address	OpCode	Assembly	Registers
cs:0000 8B7D08	mov	ax,0E7D	ax 2456
cs:0003 8ED8	mov	ds,ax	bx 0E2A
cs:0005 A10C00	mov	ax,[0000C]	cx 0000
cs:0008 BB1E0F00	mov	bx,[000E1]	dx 0000
cs:000C 1BD8	sbb	bx,ax	si 0000
cs:000E 7304	jnb	0014	di 0000
cs:0010 FF061200	inc	word ptr [0012]	bp 0000
cs:0014 891E1000	mov	[0010],bx	sp 0000
cs:0018 B44C	mov	ah,4C	ds 0E7D
cs:001A CD21	int	21	es 0E6C
cs:001C 56	push	si	ss 0E7B
cs:001D 2480	and	al,80	cs 0E7C
cs:001F 322A	xor	ch,[bp+si]	ip 0018

Stack dump:

Address	Value
ds:0000 FF 06 12 00 89 1E 10 00	♦ t è►
ds:0008 B4 4C CD 21 56 24 80 32	L=U\$G2
ds:0010 ZA OE OE EB C3 9B 07 3D	*Jñ8 C=
ds:0018 00 00 75 08 A1 BD 00 A3	uñiú

F1-Help F2-Bkpt F3-Mod F4-Here F5-Zoom F6-Next F7-Trace F8-Step F9-Run F10-Menu

**FR. Conceicao Rodrigues College of Engineering**  
**Department of Computer Engineering**

## **2. Multiplication of Two 8/16/32 bit numbers**

### **1. Course, Subject & Experiment Details**

Academic Year	2023-24	Estimated Time	Experiment No. 2– 02 Hours
Course & Semester	S.E. (Comps) – Sem. IV	Subject Name	Microprocessor
Chapter No.	2	Chapter Title	Instruction Set and Programming
Experiment Type	Software	Subject Code	CSC405

### **Rubrics**

Timeline (2)	Practical Skill & Applied Knowledge (2)	Output (3)	Postlab (3)	Total (10)	Sign

### **2. Aim & Objective of Experiment**

**TO MULTIPLY TWO 8/16/32 BIT NUMBERS**

**Objective :**Program involves storing the two 8/16/32 bit numbers in memory locations and multiplying them the objective of this program is to give an overview of arithmetic instructions of 8086 for 32 bit operation

### **3. Software Required**

TASM Assembler

**Prepared by: Prof. Heenakausar Pendhari**

## **4 . Brief Theoretical Description**

- Pre-Requisites:**
1. Instructions of microprocessor 8086
  2. Addressing mode of microprocessor 8086.
  3. Flag register of microprocessor 8086
  4. Knowledge of TASM directories.

**Theory:** MUL instruction This instruction multiplies byte/word present in source with AL/AX.

a) When two 8 bit numbers are multiplied a 16 bit product is made available in AX register where AH stores higher byte and AL stores lower byte of the product.

e.g. MUL BL

b) When two 16 bit numbers are multiplied a 32 bit product is made available in DX and AX register pair. DX has higher word and AX has lower word of the product.

e.g. MUL BX

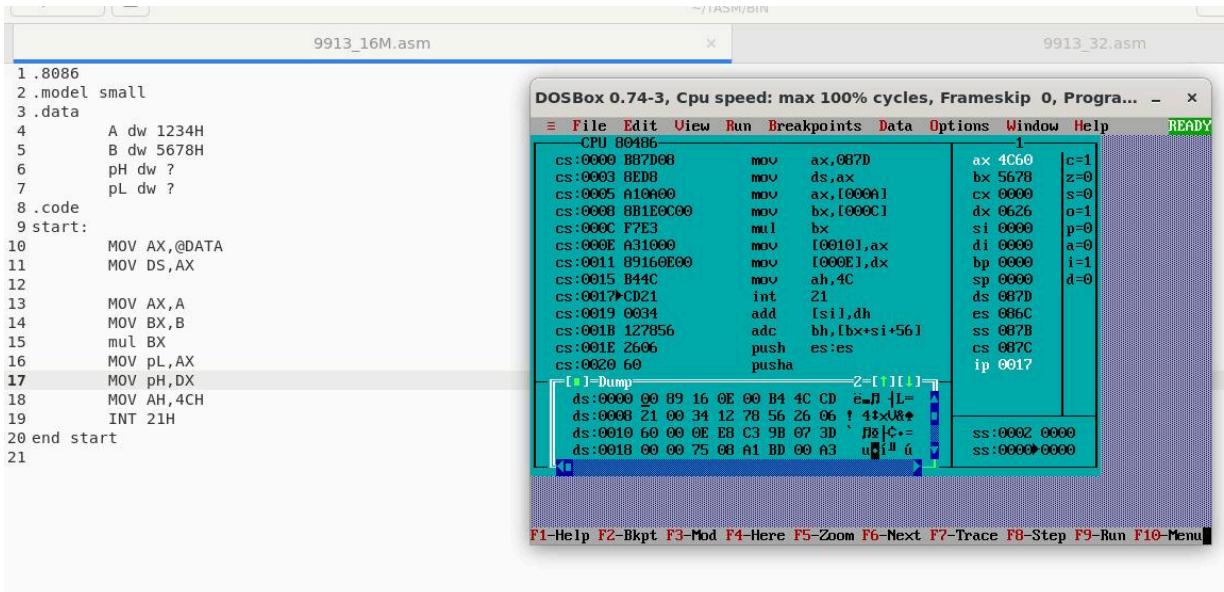
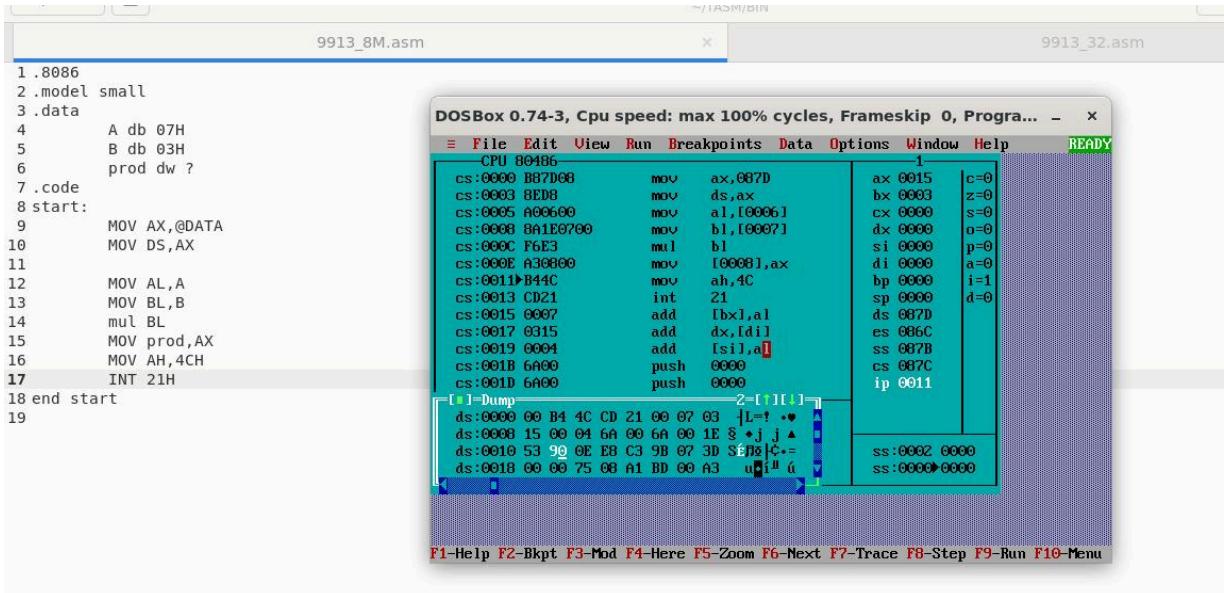
## **5. Algorithm:**

1. Initialize the data segment.
2. Store two 8/16/32 bit numbers in memory locations.
3. Move the 1st number in any TWO general purpose register.
4. Move the 2nd number in any other TWO general purpose register
5. Multiply the 2 numbers.
6. Store the result in memory location.
7. Stop.

## **6. Conclusion:**

```
1 .8086
2 .model small
3 .data
4     A db 07H
5     B db 03H
6     prod dw ?
7 .code
8 start:
9     MOV AX,@DATA
10    MOV DS,AX
11
12    MOV AL,A
13    MOV BL,B
14    mul BL
15    MOV prod,AX
16    MOV AH,4CH
17    INT 21H
18 end start
19
```

```
1 .8086
2 .model small
3 .data
4     A dw 1234H
5     B dw 5678H
6     DH dw ?
7     pL dw ?
8 .code
9 start:
10    MOV AX,@DATA
11    MOV DS,AX
12
13    MOV AX,A
14    MOV BX,B
15    mul BX
16    MOV pL,AX
17    MOV pH,DX
18    MOV AH,4CH
19    INT 21H
20 end start
21
```



## 7. Postlab:

1. Write a program to Multiply two 32 bit number.

```

.8086
.model small
.data
    num1h dw 9FFFH
    num1l dw 9FFFH
    num2h dw 9FFFH
    num2l dw 9FFFH
    resultll dw 0000H
    resultlh dw 0000H
    resulthl dw 0000H
    resulthh dw 0000H
.code
.start:
    MOV AX, @data
    MOV DS, AX

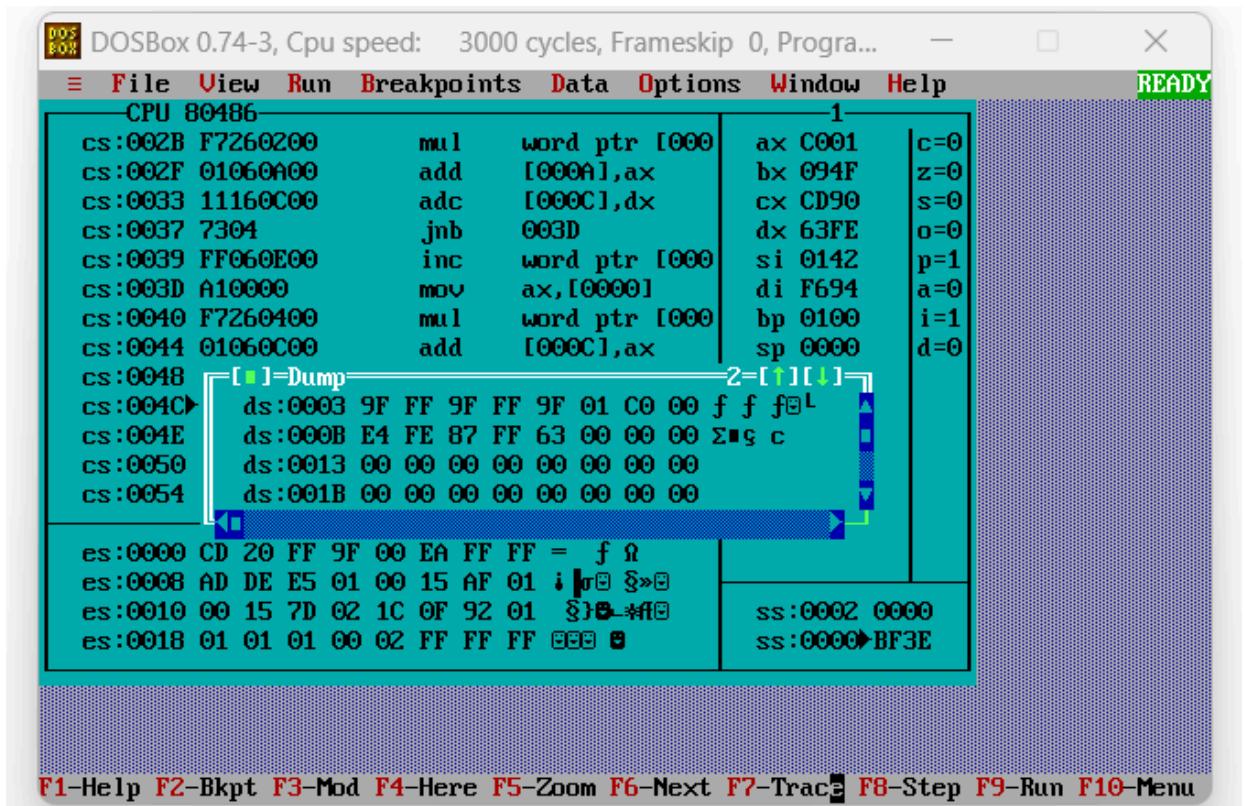
    MOV AX, num1l
    MUL num2l
    MOV resultll, AX; 16bit llsb.1
    MOV resultlh, DX; 16bit lhsb.1

    MOV AX, num1h;
    MUL num2l;
    ADD resultlh, AX; 16bit lhsb + lsb.2 can gen carry
    ADC resulthl, DX; 16bit hlsb, if there is a carry for hl add it alongside. 1
    JNC skip1; ignore the carry addition if there is not carry generated
    INC resulthh; if there was a carry generated in the above addition the add it in the highest 16bit
    skip1:

    MOV AX, num2h;
    MUL num1l;
    ADD resultlh, AX; 16bit lhsb + lsb.3 can gen carry
    ADC resulthl, DX; 16bit hlsb + hsb, if there is a carry for hl add it alongside. 2 can gen carry
    JNC skip2; ignore the carry addition if there is not carry generated
    INC resulthh; if there was a carry generated in the above addition the add it in the highest 16bit
    skip2:
    MOV AX, num1h;
    MUL num2h;
    ADD resulthl, AX; 16bit hlsb.3 can gen carry
    ADC resulthh, DX; move the 16 bit hsb in BX. 1

    MOV AH, 4CH
    INT 21H
end start

```



## 2. What are the different types of addressing modes of 8086

The 8086 microprocessor supports a wide range of addressing modes

### i) Immediate:

- Operand is embedded directly within the instruction itself.
- Useful for small, constant values.
- Example: MOV AX, 10 (Move the immediate value 10 into the AX register)

### ii) Register:

- Operand resides in one of the 8086's internal registers (AX, BX, CX, DX, etc.).
- Fastest and most efficient for accessing register data.
- Example: ADD AX, BX (Add the value in BX to the value in AX)

### iii) Direct:

- Operand's memory address is specified directly within the instruction.
- Requires 16 bits for addressing within the same segment.

- Example: `MOV BX, [1000h]` (Move the value from memory location 1000h to the BX register)

iv) Register Indirect:

Address of the operand is stored in one of the register (BX, SI, DI).

Useful for accessing data stored in memory locations whose addresses are dynamic or calculated during program execution.

Example: `MOV AX, [BX]` (Move the value from the memory location whose address is stored in BX to the AX register)

3. Briefly Explain The Pointers And Index Group of Registers.?

In the 8086 microprocessor, pointers and index registers are two special groups of registers used primarily for memory addressing and manipulation.

i) Pointers

Purpose: Store memory addresses that "point" to specific locations within a segment.

Key Registers:

IP (Instruction Pointer): Holds the offset address of the next instruction to be fetched by the Execution Unit (EU) from the Code Segment (CS).

SP (Stack Pointer): Points to the top of the stack within the Stack Segment (SS). The stack is used for storing temporary data, function parameters, and return addresses.

BP (Base Pointer): Often used to point to the base of a data structure or array within the Data Segment (DS).

ii) Index

Purpose: Primarily used for holding offsets in various addressing modes, making it easier to access elements within an array or data structure.

Key Registers:

SI (Source Index): Often used to hold the offset of the source operand in string operations or memory blocks.

DI (Destination Index): Frequently used to hold the offset of the destination address in string operations or memory blocks.



**FR. Conceicao Rodrigues College of Engineering**  
**Department of Computer Engineering**

### **3. TO IMPLEMENT BLOCK TRANSFER**

#### **1. Course, Subject & Experiment Details**

Academic Year	2023-24	Estimate d Time	Experiment No. 3– 02 Hours
Course & Semester	S.E. (Comps) – Sem. IV	Subject Name	Microprocessor
Chapter No.	2	Chapter Title	Instruction Set and Programming
Experiment Type	Software	Subject Code	CSC405

#### **Rubrics**

Timeline (2)	Practical Skill & Applied Knowledge (2)	Output (3)	Postlab (3)	Total (10)	Sign

#### **2. Aim & Objective of Experiment**

**Aim:** Write a program to transfer a block of data from one location to another.

**Objective :** Program involves transferring source string from a particular location in source segment (Data Segment) to the desired location in destination segment (Extra Segment). The objective of this program is to give an overview of the String instructions of 8086.

#### **3. Software Required**

TASM Simulator

- 4. Pre-Requisites:**
1. Knowledge of TASM directives.
  2. Knowledge of String Instructions of 8086.

- 5. Algorithm:**
1. Initialize the data segment.
  2. Store the source string in consecutive memory location
  3. Initialize the extra segment.
  4. Allocate consecutive memory locations for transfer.
  5. Load the effective address of source string in SI register.
  6. Load the effective address of destination string in DI register.
  7. Initialize the Direction flag for Auto increment or Auto Decrement.
  8. Store number of bytes to be transferred in any of the general Purpose registers.
  9. Transfer the source string using appropriate string instructions (MOVSB / MOVSW)
  10. Decrement count
  11. Check if count = 0. If yes then stop else repeat steps 9 - 11.
  12. Stop

## 6. Conclusion:

The screenshot shows the TASM/BIN interface on the left and DOSBox 0.74-3 on the right. The assembly code in TASM is:

```

1 .8086
2 .model small
3 .data
4     loc1 db 1AH,1BH,2CH,3DH,52H,3CH,21H,7EH,6AH,5EH
5     loc2 db ?
6 .code
7 start:
8     MOV AX,@DATA
9     MOV DS,AX
10    MOV ES,AX
11    LEA SI, loc1
12    LEA DI, loc2
13    CLD
14    MOV CX, 000AH
15    REP MOVSB
16    INT 03H
17
18 end start

```

The DOSBox window shows the CPU dump and memory dump. The CPU dump shows the assembly code and its corresponding opcodes. The memory dump shows the source string (loc1) at address 0000:0014 and the destination string (loc2) at address 0000:0018. The memory dump also shows the direction flag (DF) being set to 1, indicating a move from source to destination.

## **Postlab:**

- 1.** What is the advantage of segmentation?

In the context of microprocessors, segmentation refers to a technique used to **divide the program's memory space into smaller, more manageable sections**. This technique offers several advantages:

**i) Improved Memory Management:** By dividing memory into segments, the microprocessor can track and protect different parts of the program more effectively. This prevents one program section from accidentally overwriting another, enhancing program stability and security.

**ii) Increased Efficiency:** Segmentation allows the microprocessor to translate logical addresses (used by the program) to physical addresses (used by memory) more efficiently. This is because the segment table helps map logical segment addresses to physical memory locations, reducing the number of memory accesses needed.

- 2.** Explain the significance of REP Prefix

The REP (Repeat) prefix is an x86 instruction prefix that is used to repeat certain string and memory operations. It plays a crucial role in repetitive operations where the same operation needs to be performed multiple times on a sequence of data. The significance of the REP prefix lies in its ability to simplify and optimize repetitive string operations in x86 assembly language.

**FR. Conceicao Rodrigues College of Engineering**  
**Department of Computer Engineering**

**4. ARRANGING NUMBER IN ASCENDING / DESCENDING ORDER.**

**1. Course, Subject & Experiment Details**

Academic Year	2023-24	Estimated Time	Experiment No. 4– 02 Hours
Course & Semester	S.E. (Comps) – Sem. IV	Subject Name	Microprocessor
Chapter No.	2	Chapter Title	Instruction Set and Programming
Experiment Type	Software	Subject Code	CSC405

**Rubrics**

Timeline (2)	Practical Skill & Applied Knowledge (2)	Output (3)	Postlab (3)	Total (10)	Sign

**2. Aim & Objective of Experiment**

**Arrange the given numbers in Ascending/ Descending order.**

**Objective :** Program involves sorting an array in ascending order using Bubble sort algorithm. The objective of this program is to give an overview of the Compare and Jump instructions. Use of Indirect Addressing mode for array addressing is expected

**3. Software Required**

TASM Assembler

## **4 . Brief Theoretical Description**

- Pre-Requisites:**
1. Knowledge of TASM directories.
  2. Knowledge of CMP and Jump Instructions of 8086.

## **5. Algorithm:**

1. Initialize the data segment.
2. Initialize the array to be sorted.
3. Store the count of numbers in a register.
4. Store count-1 in another register.
5. Load the effective address of array in any general purpose register.
6. Load the first element of the array in a register.
7. Compare with the next element of the array.
8. Check for carry flag.
9. If carry=0 first number > second number. Swap the 2 numbers.
10. Increment to the contents of the SI register so that it points to the next element of the array.
11. Decrement (count-1) by 1.
12. Check if (count-1) =0. If no then repeat steps 7 to 11.
13. Decrement count by 1.
14. Check if count = 0.If no then repeat steps 6 through Stop.

## 6. Conclusion:

The screenshot shows a DOSBox window running TASM/BIN. On the left, the assembly code for file \*9913ascen.asm is displayed:

```
1 .8086
2 .model small
3 .data
4 array db 21H,34H,98H,1EH,5CH
5
6 .code
7 start:
8 MOV AX, @DATA
9 MOV DS, AX
10 MOV CH, 04H
11
12 UP2: MOV CL, 05H
13 LEA SI, array
14
15 UP1: MOV AL, [SI]
16 MOV BL, [SI+1]
17 CMP AL, BL
18 JC DOWN
19 MOV DL, [SI+1]
20 XCHG [SI], DL
21 MOV [SI+1], DL
22
23 DOWN: INC SI
24 JNZ UP1
25 DEC CH
26 JNZ UP2
27 INT 3H
28 end start
```

On the right, the DOSBox CPU window shows the assembly code and registers:

CPU 80486	1	ax 0098	c=1
cs:0007 B105	mov cl,05	bx 008D	z=0
cs:0009 BE0000	mov si,0000	cx 0005	s=0
cs:000C 8A04	mov al,[si]	dx 0098	o=0
cs:000E 8A5C01	mov bl,[si+01]	si 0000	p=1
cs:0011 3AC3	cmp al,bl	di 0000	a=0
cs:0013 720B	jb 001D	bp 0000	i=1
cs:0015 8A5401	mov dl,[si+01]	sp 0000	d=0
cs:0018 8614	xchg [si],dl		
cs:001A 885401	[l=Dump]		
cs:001D 46	ds:0000 75 EA FE CD 75 E1 CC 00	ah=0B	
cs:001E FEC9	ds:0008 21 34 1E 50 03 BD 00 A3 14	al=00	
cs:0020 75EA	ds:0010 B6 26 F8 C3 F9 C3 BB 00	bx=00	
cs:0022 FEC0	ds:0018 00 26 83 3E 06 00 02 76	cx=00	
es:0000 CD 20 7D 9D 00 EA FF FF	=	dx=00	
es:0000 AD DE 32 0B C3 05 6B 07	↓	si=00	
es:0010 14 03 28 00 14 03 92 01	↑	di=00	
es:0018 01 01 01 00 02 04 FF FF	↑	bp=00	
		ss:0002 0000	
		ss:0000 0000	

The DOSBox status bar at the bottom shows various keyboard shortcuts.

## Postlab:

### 1. Compare JMP and CMP instruction

JMP and CMP are both assembly language instructions used in various processor architectures, but they serve different purposes:

JMP (Jump):

- Function: Performs an unconditional jump to a specific memory address.`expand_more`
- Effect: Transfers program execution control to the instruction located at the specified address.`expand_more`
- No comparison involved: JMP doesn't perform any comparison; it simply jumps regardless of the processor's state.`exclamation`

- Example: `JMP 0x1000` would jump to the instruction located at memory address `0x1000`.

### CMP (Compare):

- Function: Compares two operands (values).
- Effect: Does not modify the operands themselves.`expand_more` Instead, it sets the flags register in the processor based on the comparison result (equal, greater than, less than, etc.).`expand_more`
- Used for conditional execution: The CMP instruction is usually followed by a conditional jump instruction (e.g., JE - jump if equal, JNE - jump if not equal, etc.) to alter program flow based on the comparison outcome.`expand_more`
- Example: `CMP register1, register2` would compare the values in `register1` and `register2`, setting the flags register accordingly.`expand_more` You might then follow this with `JE label` to jump to a specific label (`label`) if the values were equal.

**FR. Conceicao Rodrigues College of  
Engineering Department of Computer  
Engineering**

**5.TO COUNT EVEN AND ODD NUMBERS FROM AN ARRAY OF 10 NUMBERS.**

**1. Course, Subject & Experiment Details**

Academic Year	2023-24	Estimated Time	Experiment No. 5– 02 Hours
Course & Semester	S.E. (Comps) – Sem. IV	Subject Name	Microprocessor
Chapter No.	2	Chapter Title	Instruction Set and Programming
Experiment Type	Software	Subject Code	CSC405

**Rubrics**

Timeline (2)	Practical Skill & Applied Knowledge (2)	Output (3)	Postlab (3)	Total (10)	Sign

**2. Aim & Objective of Experiment**

**Arrange the given numbers in Ascending/ Descending order.**

**Objective :** Program involves counting even and odd numbers from a given array. The objective of this program is to give an overview of the string instructions of 8086

**3. Software Required**

TASM Assembler

## **4 . Brief Theoretical Description**

- Pre-Requisites:**
1. Knowledge of TASM directories.
  2. Knowledge of CMP and Jump Instructions of 8086.

Theory: A string is a series of bytes stored sequentially in the memory. string

Instruction operates on such ‘strings’. The SRC element is taken from the data segment using and SI register. The destination element is in extra segment pointed by DI register. These registers are incremented or decremented after each operation depending upon the direction flag in flag register.

Some of the instructions useful for program are,

- 1) CLC - the instruction clears the carry flag.
- 2) RCR destination, count- Right shifts the bits of destination. LSB is shifted into CF. CF goes to MSB. Bits Are shifted counts no of times.
- 3) JC: jump to specified location.
- 4) INC/DEC destination: add/subtract 1 from the specified destination.
- 5) JMP label: The control is shifted to an instruction to which label is attached.
- 6) JNZ label: The control is shifted to an instruction to which label is attached if ZF = 0

## **5. Algorithm:**

1. Initialize the data segment.
2. Initialize the array.
3. Load the effective address of an array in any index register.
4. Load total number of elements of the array in any register.
5. Initialize any two registers as counter for even and odd numbers to zero.
6. Load first element of an array in any general purpose register.
7. Shift/rotate the contents of loaded register to right.
8. If CF=1 increment counter for odd numbers otherwise increment counter of even numbers.
9. Store the value of even and odd counter register to two memory locations.
10. Stop.

## **6. Conclusion:**

```
evodd.asm
1 .model small
2
3 .data
4     array db 01H, 02H, 03H, 04H, 05H, 06H, 07H, 08H, 09H, 0AH
5     odd_no db 0
6     even_no db 0
7
8 .code
9 start:
0     MOV AX, @data
1     MOV DS, AX
2
3     LEA SI, array
4     MOV CX, 000AH
5
6 check:
7     MOV AL, [SI]
8
9     TEST AL, 1
0     JNZ odd
1
2     INC even_no
3     JMP next
4
5 odd:
6     INC odd_no
7
8 next:
9     INC SI
0     LOOP check
1
2     MOV AH, 02H
3     MOV DL, [even_no]
4     ADD DL, 30H
5     INT 21H
6
7     MOV DL, '-'
8     INT 21H
9
0     MOV DL, [odd_no]
1     ADD DL, 30H
2     INT 21H
3
4     INT 20H
5 end start
```

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

CPU 80486

			1
cs:001D	E2EC	loop	000B
cs:001F	B402	mov	ah,02
cs:0021	8A161500	mov	dl,[0015]
cs:0025	80C230	add	dl,30
cs:0028	CD21	int	21
cs:002A	B22D	mov	dl,2D
cs:002C	CD21	int	21
cs:002E	8A161400	mov	dl,[0014]
cs:0032	[ █ ]=Dump		2=[↑][↓]
cs:0035	ds:0000 14 00 80 C2 30 CD 21 CD ¶ G T 0!=!=		
cs:0037	ds:0008 20 00 01 02 03 04 05 06 E C v + + +		
cs:0039	ds:0010 07 08 09 0A 05_05 00 00 . o o o o		
cs:003B	ds:0018 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00		
es:0000	CD 20 FF 9F 00 EA FF FF = f ¶		
es:0008	AD DE E5 01 00 15 AF 01 i r g >>		
es:0010	00 15 ?D 02 1C 0F 92 01 S o - # A		
es:0018	01 01 01 00 02 FF FF FF EEE E		
		ss:0002 6568	
		ss:0000 6474	

F1-Help F2-Bkpt F3-Mod F4-Here F5-Zoom F6-Next F7-Trace F8-Step F9-Run F10-Menu

## 7. Postlab:

### 1. Explain CMPSB/CMPSW , LODS/STOS instructions

#### **CMPSB (Compare String Byte)**

- Purpose: Compare the byte at the current position in one string with the byte at the same position in another string.
- Example Use: Compare characters in two strings.

#### **CMPSW (Compare String Word)**

- Purpose: Similar to CMPSB, but compares two words (16 bits) instead of bytes.
- Example Use: Compare 16-bit values in two strings.

## **LODS (Load String)**

- Purpose: Load a byte or a word from the memory into a register (AL or AX) and advance the source index (SI).
- Example Use: Retrieve a character or value from a string.

## **STOS (Store String)**

- Purpose: Store a byte or a word from a register (AL or AX) into the memory at the destination index (DI) and advance DI.
- Example Use: Put a character or value into a string.

**FR. Conceicao Rodrigues College of Engineering**  
**Department of Computer Engineering**

## **6. Matrix Addition/ Multiplication.**

### **1. Course, Subject & Experiment Details**

Academic Year	2023-24	Estimated Time	Experiment No. 6– 02 Hours
Course & Semester	S.E. (Comps) – Sem. IV	Subject Name	Microprocessor
Chapter No.	2	Chapter Title	Instruction Set and Programming
Experiment Type	Software	Subject Code	CSC405

### **Rubrics**

Timeline (2)	Practical Skill & Applied Knowledge (2)	Output (3)	Postlab (3)	Total (10)	Sign

### **2. Aim & Objective of Experiment**

**Perform Addition & Multiplication of 3 X 3 Matrix**

**Objective :** The objective is to Add & multiply 3 X 3 matrix

### **3. Software Required**

TASM Assembler

## **4 . Brief Theoretical Description**

- Pre-Requisites:**
1. Knowledge of TASM directives.
  2. Knowledge of DOS interrupts.
  3. Knowledge of string instruction and MACRO

## **5. Algorithm:**

1. Initialize the data segment.
2. Initialize counter = 9
3. Initialize pointer DI to matrix 1.
4. Initialize pointer BX to matrix 2.
5. Initialize pointer SI to result matrix 3.
6. Get the number from matrix 1.
7. Add number from matrix 1 with matrix 2 number.
8. Save the carry if any.
9. Save the result in result matrix 3.
10. Increment DI, BX, and SI to point to next element.
11. Decrement count.
12. Check if count = 0, if not go to step VI else go to step XIII
13. Display the result.
14. Stop.

Matrix addition:-

```
ASM matadd.asm
1    .model small
2    .stack 100h
3
4    .data
5    matrix1 db 1, 1, 1, 1, 1, 1, 1, 1, 1
6    matrix2 db 2, 2, 2, 2, 2, 2, 2, 2, 2
7    result_add db 0, 0, 0, 0, 0, 0, 0, 0, 0
8
9    .code
10   start:
11    mov ax, @data
12    mov ds, ax
13
14    mov cx, 9 ; Number of elements in the matrices (3x3)
15
16    lea si, matrix1
17    lea di, matrix2
18    lea bx, result_add
19
20    addition_loop:
21    mov al, [si]
22    add al, [di]
23    mov [bx], al
24
25    inc si
26    inc di
27    inc bx
28
29    loop addition_loop
30
31    mov ah, 09h
32    lea dx, result_add
33    int 21h
34
35    mov ah, 4ch
36    int 21h
37
38    end start
39
```

DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Program... — X

File View Run Breakpoints Data Options Window Help READY

CPU 80486

				1
CS:001A	E2F5	loop	0011	ax 4C03
CS:001C	B409	mov	ah,09	bx 001B
CS:001E	BA1200	mov	dx,0012	cx 0000
CS:0021	CD21	int	21	dx 0012
CS:0023	B44C	mov	ah,4C	si 0009
CS:0025	CD21	int	21	di 0012
CS:0027	0000	add	[bx+sil],al	bp 0000
CS:0029	0000	add	[bx+sil],al	sp 0100
CS:002B	I=Dump=		Z=[↑] [↓]	
CS:002D	ds:0000	01 01 01 01 01 01 01 01	BBBBBBBB	
CS:002F	ds:0008	01 02 02 02 02 02 02 02	CCCCCCCC	
CS:0031	ds:0010	02 02 03 03 03 03 03 03	DDDDDDDD	
CS:0033	ds:0018	03 03 03 00 00 00 00 00	EEEEEE	
ES:0000	CD 20 FF 9F 00 EA FF FF	= f		
ES:0008	AD DE E5 01 00 15 AF 01	i		
ES:0010	00 15 7D 02 1C 0F 92 01	S>		
ES:0018	01 01 01 00 02 FF FF FF	BBB		
			SS:0102 0000	
			SS:0100 0000	

F1-Help F2-Bkpt F3-Mod F4-Here F5-Zoom F6-Next F7-Trace F8-Step F9-Run F10-Menu

### Postlab:

1. Write a program to Multiply 3 X 3 Matrix.

```
SEVENUSASM          matmul.asm          F:\RAJ\ASSEMBLY\matmul.asm
ASM matmul.asm
1   .model small
2   .data
3   matrix1 db 1, 2, 3, 4, 5, 6, 7, 8, 9
4   matrix2 db 9, 8, 7, 6, 5, 4, 3, 2, 1
5   result_mult db 9 dup(?)
6
7   .code
8   start:
9       mov ax, @data
10      mov ds, ax
11
12      mov di, offset matrix1
13      mov cx, 9
14      mov bx, offset matrix2
15      mov si, offset result_mult
16
17      multiplication_loop:
18          mov al, [di]
19          mov ah, 0
20          mov bl, [bx]
21          mul bl
22          add al, [si]
23          mov [si], al
24          adc ah, 0
25          mov [si + 1], ah
26
27          inc di
28          inc bx
29          add si, 2
30
31          loop multiplication_loop
32
33          mov dx, offset result_mult
34          mov ah, 09h
35          int 21h
36
37          mov ah, 4Ch
38          int 21h
39
40      end start
41  |
```

DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip: 0, Program: READY

CPU 80486

cs:001D	80D400	adc	ah,00	ax 0900	c=0
cs:0020	886401	mov	[si+01],ah	bx 0001	z=0
cs:0023	47	inc	di	cx 0000	s=0
cs:0024	43	inc	bx	dx 0018	o=0
cs:0025	83C602	add	si,0002	si 002A	p=0
cs:0028	E2E7	loop	0011	di 000F	a=0
cs:002A	BA1800	mov	dx,0018	bp 0000	i=1
cs:002D	B409	mov	ah,09	sp 0000	d=0
cs:002F	[ ]=Dump				
cs:0031	ds:0000 21 B4 4C CD 21 00 01 02				
cs:0033	ds:0008 03 04 05 06 07 08 09 09				
cs:0035	ds:0010 08 07 06 05 04 03 02 01				
cs:0037	ds:0018 09 00 0A 00 03 00 30 01				
es:0000	CD 20 FF 9F 00 EA FF FF	= f	Ω		
es:0008	AD DE E5 01 00 15 AF 01	i	▀	▀	▀
es:0010	00 15 7D 02 1C 0F 92 01	§)	▀	▀	▀
es:0018	01 01 01 00 02 FF FF FF BBB	▀	▀	▀	▀
				ss:0002 6568	
				ss:0000 6474	

F1-Help F2-Bkpt F3-Mod F4-Here F5-Zoom F6-Next F7-Trace F8-Step F9-Run F10-Menu

**FR. Conceicao Rodrigues College of Engineering**  
**Department of Computer Engineering**

**7. DISPLAY A TO Z ON SCREEN.**

**1. Course, Subject & Experiment Details**

Academic Year	2023-24	Estimated Time	Experiment No. 7– 02 Hours
Course & Semester	S.E. (Comps) – Sem. IV	Subject Name	Microprocessor
Chapter No.	2	Chapter Title	Instruction Set and Programming
Experiment Type	Software	Subject Code	CSC405

**Rubrics**

Timeline (2)	Practical Skill & Applied Knowledge (2)	Output (3)	Postlab (3)	Total (10)	Sign

**2. Aim & Objective of Experiment**

**7(A) DISPLAY A TO Z ON SCREEN**

**Objective :** To store A to Z Alphabets on an array and display them on user screen.

**3. Software Required**

TASM Assembler

## **4 . Brief Theoretical Description**

- Pre-Requisites:**
1. Knowledge of TASM directives.
  2. Knowledge of DOS interrupts.
  3. Knowledge of string instruction and MACRO

## **5. Algorithm:**

1. Initialize the data segment.
2. Store all Alphabets in array.
3. Initialize counter to 1AH.
4. Load starting Address of array in to SI.
5. Get each character in DL.
6. Display Character on user screen.
7. Increment SI.
8. Decrement counter.
9. Repeat step 5 to 8 until count becomes Zero.
10. Stop

ASM 32mul.asm

ASM matadd.asm

ASM MATADD.OBJ

ASM matmul.asm

ASM A-Z

```
ASM A-Z.asm
1 .model small
2 .data
3
4 .code
5 start:
6     MOV AX, @data
7     MOV DS, AX
8
9     MOV CX, 26    ; Number of characters from 'A' to 'Z'
10    MOV DL, 'A'   ; Start with 'A'
11
12    display_loop:
13        MOV AH, 02H  ; Function to display character
14        INT 21H
15
16        INC DL      ; Move to the next character
17        LOOP display_loop
18
19        INT 20H      ; Exit the program
20
21    end start
22 |
```

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

For a short introduction for new users type: INTRO  
For supported shell commands type: HELP

To adjust the emulated CPU speed, use **ctrl-F11** and **ctrl-F12**.  
To activate the keymapper **ctrl-F1**.  
For more information read the **README** file in the DOSBox directory.

**HAVE FUN!**  
The DOSBox Team <http://www.dosbox.com>

```
Z:\>SET BLASTER=A220 I7 D1 H5 T6
Z:\>mount c c://tasm
Drive C is mounted as local directory c://tasm\
Z:\>c://
C:\>tlink A-Z
Turbo Link Version 4.0 Copyright (c) 1991 Borland International
Warning: No stack
C:\>td A-Z
Turbo Debugger Version 2.51 Copyright (c) 1988,91 Borland International
ABCDEFIGHJKLMNOPQRSTUVWXYZ
```

7(B)

### DISPLAY CHARACTER FROM KEYBOARD UNTIL 0 IS ENTERED.

Objective: To Read Character from Keyboard and display on screen until 0 is pressed.

Theory: Instructions used in program are:

**MOV AH,08H**

**INT 21H**

Read Input From Keyboard without echo and store at AL.

**MOV AH,02H**

**INT 21H**

Display Character on screen. Character should be in DL register.

**Algorithm:**

1. Initialize the data segment.
2. Read input from keyboard.
3. Compare input with ASCII value of ZERO.
4. If result is 0, go to step 7.
5. Move content of AL to DL, to display it on screen.
6. Display character on screen.
7. Stop

The screenshot shows a Microsoft Visual Studio IDE interface with multiple tabs open. The active tab is 'check0.asm'. The code in the editor is as follows:

```
ASM 32mul.asm | ASM matadd.asm | MATADD.OBJ | ASM matmul.asm | ASM A-Z.asm | ASM  
ASM check0.asm  
1 .8086  
2 .model small  
3 .data  
4  
5 .code  
6 start:  
7     MOV AX, @data  
8     MOV DS, AX  
9  
10    read_loop:  
11        MOV AH, 01H ; Function to read a character from the keyboard  
12        INT 21H  
13  
14        CMP AL, '0' ; Check if the entered character is '0'  
15        JE end_program ; If '0', jump to end_program  
16  
17        MOV DL, AL  
18        MOV AH, 02H ; Function to display character  
19        INT 21H  
20  
21        JMP read_loop ; Repeat the loop to read more characters  
22  
23    end_program:  
24        MOV AH, 4CH ; Terminate program  
25        INT 21H  
26  
27    end start  
28
```

DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Program: matmul

```
Z:\>SET BLASTER=A220 I7 D1 H5 T6
Z:\>mount c c://tasm
Drive C is mounted as local directory c://tasm\

Z:\>c://
C:\>tasm check0.asm
Turbo Assembler Version 2.51 Copyright (c) 1988, 1991 Borland International
Assembling file: check0.asm
Error messages: None
Warning messages: None
Passes: 1
Remaining memory: 491k

C:\>tlink check0
Turbo Link Version 4.0 Copyright (c) 1991 Borland International
Warning: No stack

C:\>td check0
Turbo Debugger Version 2.51 Copyright (c) 1988,91 Borland International
mmmmyy0_
```

**FR. Conceicao Rodrigues College of Engineering**  
**Department of Computer Engineering**

**8. PASSWORD VERIFICATION.**

**1. Course, Subject & Experiment Details**

Academic Year	2023-24	Estimated Time	Experiment No. 8– 02 Hours
Course & Semester	S.E. (Comps) – Sem. IV	Subject Name	Microprocessor
Chapter No.	2	Chapter Title	Instruction Set and Programming
Experiment Type	Software	Subject Code	CSC405

**Rubrics**

Timeline (2)	Practical Skill & Applied Knowledge (2)	Output (3)	Postlab (3)	Total (10)	Sign

**2. Aim & Objective of Experiment**

**PASSWORD VERIFICATION**

**Objective :** The objective is to make use of string instruction and MACRO, to check whether the entered password by the user is correct or not..

**3. Software Required**

TASM Assembler

## **4 . Brief Theoretical Description**

- Pre-Requisites:**
1. Knowledge of TASM directives.
  2. Knowledge of DOS interrupts.
  3. Knowledge of string instruction and MACRO

## **5. Algorithm:**

1. Store Initial password into Array.
2. Write Macro for printing output message.
3. Write Macro to display '\*'.
4. Initialize the data segment.
5. Set the counter value=no. of character present in password.
6. Load Effective address of stored password in BX.
7. Take input from the keyboard.
8. Compare input with the password string.
9. If zero=0, both value are equal. Go to step 10.  
.If zero is not equal to 0.Go to step 15.
10. display '\*' Macro.
11. Increment BX.
12. Decrement counter by 1.
13. Check if counter=0.If not, Repeat step 7 to 12.
14. Display Macro message for correct password, Go to step 16.
15. Display '\*' macro and Macro message for wrong password.
16. End

```
pass.asm
1 .8086
2 .model small
3
4 .data
5     password db 'pass123$'
6     user     db 8 dup ('$')
7     msg_right db 'Correct password entered.$'
8     msg_wrong db 'Wrong password entered.$'
9     input_buffer db 8 dup ('$')
10
11 .code
12 print_star MACRO
13     MOV DL, '*'
14     MOV AH, 02H
15     INT 21H
16 ENDM
17
18 print_msg MACRO msg
19     MOV AH, 09H
20     LEA DX, msg
21     INT 21H
22 ENDM
23
24 start:
25     MOV AX, @data
26     MOV DS, AX
27     MOV ES, AX
28
29     MOV CX, 8H
30     LEA BX, password
31     LEA DI, user
32
33 input_loop:
34     MOV AH, 01H
35     INT 21H
36     MOV [DI], AL
37     INC DI
38     MOV DL, AL
39     MOV AH, 02H
40     INT 21H
41     DEC CX
42     JNZ input_loop
43
44 check_password:
45     LEA SI, user
46     LEA DI, password
47     MOV CX, 8H
48     cld
49     repe cmpsb
50     jz correct_password
51     jmp wrong_password
52
53 correct_password:
54     print_star
55     print_msg msg_right
56     jmp end_program
57
58 wrong_password:
59     print_star
60     print_msg msg_wrong
61     jmp end_program
62
63 end_program:
64     int 20H
65
66 end start
```

pass123\$\*Correct password entered.

**FR. Conceicao Rodrigues College of Engineering**  
**Department of Computer Engineering**

## **9. TO CHECK IF THE ENTERED STRING IS PALINDROME OR NOT**

### **1. Course, Subject & Experiment Details**

Academic Year	2023-24	Estimated Time	Experiment No. 9– 02 Hours
Course & Semester	S.E. (Comps) – Sem. IV	Subject Name	Microprocessor
Chapter No.	2	Chapter Title	Instruction Set and Programming
Experiment Type	Software	Subject Code	CSC405

### **Rubrics**

Timeline (2)	Practical Skill & Applied Knowledge (2)	Output (3)	Postlab (3)	Total (10)	Sign

### **2. Aim & Objective of Experiment**

#### **CHECK IF THE ENTERED STRING IS PALINDROME OR NOT**

**Objective :** Read a string from user and check whether it is palindrome or not and display appropriate message.

### **3. Software Required**

TASM Assembler

## **4 . Brief Theoretical Description**

- Pre-Requisites:**
1. Knowledge of TASM directives.
  2. Knowledge of DOS interrupts.
  3. Knowledge of string instruction and MACRO

String instruction/ data transfer

- a) LEA register, Source - Loads effective address (offset Address) of source in given register.  
e.g. LEA BX, Total;
- b) STOSB/STOSW (Store string) – It is used to store AL (or AX) into a memory location pointed by ES:DI. DI is incremented or decremented after transfer depending upon DF.
- c) LODSB/LODSW: Used to copy the contents of memory location pointed by DS: SI and store it in AL registers. SI is incremented /decremented after transfer depending upon DF.

## **5. Algorithm:**

1. Start.
2. Initialize Data segment (DS).
3. Initialize Extra segment (ES).
4. Ask user to enter a string.
5. Read each character of a string using function value 01h or 08h.
6. Store individual character read in a string S.
7. Make SI register point to first element of a string and DI register to point to last element of a string.
8. Initialize count register to number of comparisons required.
9. Move contents pointed by SI to AL.
10. Compare character in AL with character pointed by DI.
11. If there is a mismatch (ZF=0) display a message “Not a palindrome” and stop.
12. If two characters are matching then increment SI, decrement DI, decrement count register.
13. If count register becomes zero display a message “String is palindrome” and stop.

```

100 palin.asm
1      .8086
2      .model small
3
4      print macro msg
5          push ax
6          mov ah, 09h
7          lea dx, msg
8          int 21h
9          pop ax
10     endm
11
12     .data
13     get db 10,13, "Enter a string: $"
14     yes db 10,13, "String is a palindrome.$"
15     no db 10,13, "String is not a palindrome.$"
16
17     inp db 20 dup(' ')
18     rev db 20 dup(' ')
19
20     .code
21     start:
22         mov ax, @data
23         mov ds, ax
24         mov es, ax
25
26         lea si, inp
27         lea di, rev
28         mov cx, 0000h
29
30         print get
31
32     back:
33         mov ah, 01h
34         int 21h
35
36         cmp al, 13
37         jz next
38
39         mov [si], al
40         inc si
41         inc cx
42         jmp back
43
44     next:
45         mov bl, cl
46
47     back1:
48         dec si
49         mov al, [si]
50         mov [di], al
51         inc di
52         loop back1
53
54         mov cl, bl
55
56         lea si, inp
57         lea di, rev
58
59         repe cmpsb
60
61         jz next1
62
63         print no
64         jmp exit
65
66     next1:
67         print yes
68
69     exit:
70         mov ah, 4ch
71         int 21h
72
73     end start
74

```

```
C:\>tasm palin.asm
Turbo Assembler Version 2.51 Copyright (c) 1988, 1991 Borland International

Assembling file: palin.asm
Error messages: None
Warning messages: None
Passes: 1
Remaining memory: 490k

C:\>tlink palin.asm
Turbo Link Version 4.0 Copyright (c) 1991 Borland International
Fatal: Bad object file palin.asm

C:\>tlink palin
Turbo Link Version 4.0 Copyright (c) 1991 Borland International
Warning: No stack

C:\>td palin
Turbo Debugger Version 2.51 Copyright (c) 1988,91 Borland International

Enter a string: civic

String is a palindrome.
```

### Postlab:

Q1. Give the difference between Macro and procedure.

#### Macros:

- Execution: Occurs at assembly time, providing a form of text replacement.
- Parameterization: Highly flexible, allows passing parameters for reuse.
- Scope: Local to the source file where they are defined.
- Usage: Primarily for code expansion, especially for repetitive or boilerplate code.

#### Procedures:

- Execution: Occurs at runtime with a function call and return.
- Parameterization: Also supports parameter passing, but in a more structured manner.
- Scope: Can be global, allowing use across multiple files.
- Usage: Primarily for organizing code into reusable and modular functions or subroutines.

**Q2.** How many bytes are pushed onto the stack by a far CALL instruction? What do they represent?

A far CALL instruction in x86 assembly language pushes either 2 or 4 bytes onto the stack, depending on whether it uses a near or far addressing mode. These bytes represent the return address for control to resume after the called subroutine or function completes.

**Near Addressing Mode:**

- If a near address is used in the far 'CALL' instruction, the return address pushed onto the stack is the offset of the instruction following the 'CALL'.
- In this case, 2 bytes are pushed onto the stack.

**Far Addressing Mode:**

- If a far address is used in the far 'CALL' instruction, the return address pushed onto the stack consists of two parts:
  - The offset of the instruction following the 'CALL' (2 bytes).
  - The segment of the instruction following the 'CALL' (2 bytes).
- In this case, 4 bytes are pushed onto the stack.

**FR. Conceicao Rodrigues College of Engineering**  
**Department of Computer Engineering**

## lab 10

### 1. Course, Subject & Experiment Details

Academic Year	2023-24	Estimated Time	Experiment No. 1– 02 Hours
Course & Semester	S.E. (Comps) – Sem. IV	Subject Name	Microprocessor
Chapter No.	-	Chapter Title	-
Experiment Type	Software& Hardware	Subject Code	CSC405

#### Rubrics

Timeline (2)	Practical Skill & Applied Knowledge (2)	Output (3)	Postlab (3)	Total (10)	Sign

Dynalog India Ltd.

Dyna-86>R

AX=0000 BX=0000 CX=0000 DX=0000 SP=06FF BP=0000 SI=0000

DI=0000 CS=0000 DS=0000 SS=0000 ES=0000 IP=0700 FL=0000

Dyna-86>AX. . .

0000:1011

Dyna-86>A 1000

0000:1000

Dyna-86>

Dyna-86>A 1000

0000:1000 MOV AX, 55

0000:1003 MOV BX,55

0000:1006 ADD AX, BX  
0000:1008 INT 3  
0000:1009  
Dyna-86>G 1000

Break at FFFF:1009  
Dyna-86>R  
AX=00AA BX=0055 CX=0000 DX=0000 SP=06FD BP=0000 SI=0000  
DI=0000 CS=FFFF DS=0000 SS=0000 ES=0000 IP=1009 FL=F006 Dyna-  
86>A 1000  
0000:1000 MOV AX,F0

0000:1003  
Dyna-86>A 1000  
0000:1000 MOV AX,F0\..

0000:1003  
Dyna-86>A 1000  
0000:1000 MOV AX,F0  
0000:1003 MOV BX,AA  
0000:1006 SUB AX,BX  
0000:1008 INT 3  
0000:1009  
Dyna-86>G1000

Break at FFFF:1009  
Dyna-86>R  
AX=0046 BX=00AA CX=0000 DX=0000 SP=06FB BP=0000 SI=0000  
DI=0000 CS=FFFF DS=0000 SS=0000 ES=0000 IP=1009 FL=F012 Dyna-  
86>A 1000  
0000:1000 MOV AX,02  
0000:1003 MOV BX,05  
0000:1006 MUL BX  
0000:1008 INT 3  
0000:1009  
Dyna-86>G 1000

Break at FFFF:1009  
Dyna-86>  
Dyna-86>R  
AX=000A BX=0005 CX=0000 DX=0000 SP=06F9 BP=0000 SI=0000  
DI=0000 CS=FFFF DS=0000 SS=0000 ES=0000 IP=1009 FL=F046  
Dyna-86>A 1200  
0000:1200 MOV AX,FE  
0000:1203 MOV BX,02  
0000:1206 DIV BX

0000:1208 INT 03  
0000:1209  
Dyna-86>G1200

Break at FFFF:1209  
Dyna-86>  
Dyna-86>R  
AX=007F BX=0002 CX=0000 DX=0000 SP=06F7 BP=0000 SI=0000  
DI=0000 CS=FFFF DS=0000 SS=0000 ES=0000 IP=1209 FL=F047  
Dyna-86>A 1000  
0000:1000 MOV AX,0F  
0000:1003 MOV BX,F0  
0000:1006 OR AX,BX  
0000:1008 INT 3  
0000:1009  
Dyna-86>G 1000

Break at FFFF:1009  
Dyna-86>R  
AX=00FF BX=00F0 CX=0000 DX=0000 SP=06F5 BP=0000 SI=0000  
DI=0000 CS=FFFF DS=0000 SS=0000 ES=0000 IP=1009 FL=F006  
Dyna-86>A 1000  
0000:1000 MOC. .V A  
Error ! Invalid operand or Addressing mode  
0000:1000  
Dyna-86>A 1000  
0000:1000 MOV AX,55  
0000:1003 MOV BX,AA  
0000:1006 AND AX,BX  
0000:1008 INT 3  
0000:1009  
Dyna-86>G1000

Break at FFFF:1009  
Dyna-86>R  
AX=0000 BX=00AA CX=0000 DX=0000 SP=06F3 BP=0000 SI=0000  
DI=0000 CS=FFFF DS=0000 SS=0000 ES=0000 IP=1009 FL=F046  
Dyna-86>A1. .  
0000:1009  
Dyna-86>A 1000  
0000:1000 MOV AX, 03  
0000:1003 MOV BX,05  
0000:1006 XOR AX,BX  
0000:1008 INT 3  
0000:1009  
Dyna-86>G 1000

Break at FFFF:1009

Dyna-86>R

AX=0006 BX=0005 CX=0000 DX=0000 SP=06F1 BP=0000 SI=0000  
DI=0000 CS=FFFF DS=0000 SS=0000 ES=0000 IP=1009 FL=F006

Dyna-86>

Dyna-86>

Dyna-86>A 1000

0000:1000 MOV AX,55  
0000:1003 NOT AX  
0000:1005 INT 03  
0000:1006

Dyna-86>G 1000

Break at FFFF:1006

Dyna-86>R

AX=FFAA BX=0005 CX=0000 DX=0000 SP=06EF BP=0000 SI=0000  
DI=0000 CS=FFFF DS=0000 SS=0000 ES=0000 IP=1006 FL=F006 Dyna-  
86>A 1000

0000:1000 MOV AX, FF  
0000:1003 MOV BX, 00  
0000:1006 INC BX  
0000:1007 DEC AX  
0000:1008 JNZ 1006  
0000:100A INT 3  
0000:100B

Dyna-86>G 1000

Break at FFFF:100B

Dyna-86>R

AX=0000 BX=00FF CX=0000 DX=0000 SP=06ED BP=0000 SI=0000  
DI=0000 CS=FFFF DS=0000 SS=0000 ES=0000 IP=100B FL=F046

Dyna-86>A 1000

0000:1000 MOV AX,80  
0000:1003 OUT 33,AX  
0000:1005 MOV AX,55  
0000:1008 OUT 30,AX  
0000:100A MOV AX,AA  
0000:100D OUT 31,AX  
0000:100F MOV AX,0F  
0000:1012 OUT 32,AX  
0000:1014 INT 3

0000:1015

Dyna-86>G 1000

Break at FFFF:1015  
Dyna-86>A 1000  
0000:1000 MOV AX,80

0000:1003  
Dyna-86>A 1000  
0000:1000 MOV AX,80  
0000:1003 OUT 33,AX  
0000:1005  
Dyna-86>  
Dyna-86>A 1000  
0000:1000 MOV AX,80  
0000:1003 OUT 33,AX  
0000:1005 MOV AX,57  
0000:1008 OUT 30,AX  
0000:100A MOV AX,BB  
0000:100D OUT 31,AX

0000:100F  
Dyna-86>A 1000  
0000:1000 MOV AX,80  
0000:1003 OUT 33,AX  
0000:1005 MOV AX,56  
0000:1008 OUT 30,AX  
0000:100A MOV AX,1C  
0000:100D OUT 31,AX  
0000:100F MOV AX,FF  
0000:1012 OUT 32,AX  
0000:1014 INT 3  
0000:1015  
Dyna-86>G1000

Break at FFFF:1015  
Dyna-86>