# 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	<b>Estimated Time</b>	Experiment No. 4– 02 Hours
Course & Semester	S.E. (Comps) – Sem. IV	Subject Name	Microprocessor
Chapter No.	2	Chapter Title	<b>Instruction Set and Programming</b>
<b>Experiment Type</b>	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:

```
CODE:
.8086
.model small
.data
STRING1 db 21H, 35H, 89H, 1EH, 5CH
.code
start:
```

MOV AX,@data

MOV DS,AX

MOV CH,04H

UP2: MOV CL,04H

LEA SI, STRING1

UP1: MOV AL,[SI]

MOV BL,[SI+1]

CMP AL, BL

JC DOWN

MOV DL,[SI+1]

XCHG [SI], DL

MOV [SI+1],DL

DOWN: INC SI

DEC CL

JNZ UP1

DEC CH

JNZ UP2

INT 3

end start

```
CPU 80486-
cs:0022 FECD
                        dec
                                ch
                                                    ax 085C
                                                                c=1
cs:0024 75E1
                                0007
                                                    b× 0089
                        jne
                                                                z=1
cs:0026 CC
                                03
                                                    cx 0000
                        int
                                                                s=0
cs:0027 001E2135
                                [3521],b1
                                                    d× 0021
                        add
                                                                o=0
cs:002B 5C
                                                    si 000C
                                                                p=1
                        pop
cs:002C 89BD00A3
                                [di-5D00],di
                        MOV
                                                    di 0000
                                                                a=0
                                                    bp 0000
cs:0030 BAZ6F8
                        MOV
                                dx, F826
                                                                i=1
                                                    sp 0000
cs:0033 C3
                                                               d=0
                        ret
cs:0034 F9
                                                    =2=[†][↓]=
               [_[ ■ ]=Dump=
                  ds:0000 75 EA FE CD 75 E1 CC 00 uΩ = uß |
cs:0035 C3
cs:0036 B80000
                  ds:0008 1E 21 35 5C 89 BD 00 A3 ▲!5\ë "ú
                  ds:0010 BA 26 F8 C3 F9 C3 B8 00 ||&°|- |-
cs:0039 26833E
cs:003F 7610
                  ds:0018 00 26 83 3E 06 00 02 76
es:0000 CD 20 7D 9D 00 EA FF FF = }¥ N
es:0008 AD DE 32 OB C3 05 6B 07 i 28 4k.
es:0010 14 03 28 08 14 03 92 01 TW ( TWHE
                                                    ss:0002 0000
es:0018 01 01 01 00 02 04 FF FF 000 0+
                                                    ss:00000 00000
```

#### Postlab:

1. Compare JMP and CMP instruction

1) The CMP (compare) and FMP (unconditional jump) instructions assembly language serve different purposes. Here's a compthe two instructions:  1. CMP (compare) Instruction:  • compares two operands by subtracting one from the other adjusts the flags (such as zero, sign, carry, overflow) the result of the subtraction.  • Does not after the destination or source operands.  • used in conditional execution and decision making, often combination with conditional jump instructions.  • syntax: CMP destination, source  2. FMP (unconditional Fump) Instruction:  • Transfers the flow of control to a specified label or manaderess unconditionalley.	ting one from the other.  gn, carry, overflow) based  wree operands. ecision making, often in actions.
1. CMP (compare) Instruction:  • compares two operands by subtracting one from the oth  • Adjusts the flags (such as zero, sign, carry, overflow)  the result of the subtraction.  • Does not after the destination or source operands.  • used in conditional execution and decision making, often combination with conditional jump instructions.  • Signtax: CMP destination, source  2. FMP (unconditional Fump) Instruction:  • Transfers the flow of control to a specified label or manaderess unconditionally.	gn, carry, overflow) based wrce operands. ecision making, often in actions.
<ul> <li>compares two operands by subtracting one from the oth</li> <li>Adjusts the flags (such as zero, sign, carry, overflow)</li> <li>the result of the subtraction.</li> <li>Does not alter the destination or source operands.</li> <li>used in conditional execution and decision making, often combination with conditional jump instructions.</li> <li>Signtax: CMP destination, source</li> <li>FMP (unconditional Fump) Instruction:</li> <li>Transfers the flow of control to a specified label or manderess unconditionally.</li> </ul>	gn, carry, overflow) based wrce operands. ecision making, often in actions.
<ul> <li>Adjusts the flags (such as zero, sign, carry, overflow) the result of the subtraction.</li> <li>Does not after the destination or source operands.</li> <li>Used in conditional execution and decision making, often combination with conditional jump instructions.</li> <li>Signtax: CMP destination, source</li> <li>FMP (unconditional Fump) Instruction:</li> <li>Transfers the flow of control to a specified label or me address unconditionally.</li> </ul>	gn, carry, overflow) based wrce operands. ecision making, often in actions.
<ul> <li>Does not after the destination or source operands.</li> <li>used in conditional execution and decision making, often combination with conditional jump instructions.</li> <li>Syntax: CMP destination, source</li> <li>FMP (unconditional Fump) Instruction:</li> <li>Transfers the flow of control to a specified label or me address unconditionally.</li> </ul>	ecision making, often in actions.
<ul> <li>used in conditional execution and decision making, often combination with conditional jump instructions.</li> <li>syntax: CMP destination, source</li> <li>JMP (unconditional Fump) Instruction:</li> <li>Transfers the flow of control to a specified label or maddress unconditionally.</li> </ul>	ecision making, often in actions.
combination with conditional jump instructions.  • Syntap: CMP destination, source  2. FMP (unconditional Fump) Instruction:  • Transfers the flow of control to a specified label or manadress unconditionally.	ictions.
Syntap: CMP destination, source  2. FMP (unconditional Fump) Instruction:      Transfers the flow of control to a specified label or manderess unconditionally.	
<ul> <li>Syntap: CMP destination, source</li> <li>2. FMP (unconditional Fump) Instruction:</li> <li>Transfers the flow of control to a specified label or me address unconditionally.</li> </ul>	
<ul> <li>Transfers the flow of control to a specified label or many address unconditionally.</li> </ul>	etion:
<ul> <li>Transfers the flow of control to a specified label or many address unconditionally.</li> </ul>	
•	specified label or memory
the contract to the contract t	
· Does not perform any comparison or evaluation of opera	or evaluation of operands.
· used to implement loops, switch statements, and to trav	tements, and to transfer
control to other parts of the program.	
· Syntap: FMP label or FMP memory_address	ry_address