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:

7. Postlab:

1. Explain CMPSB/CMPSW , LODS/STOS instructions

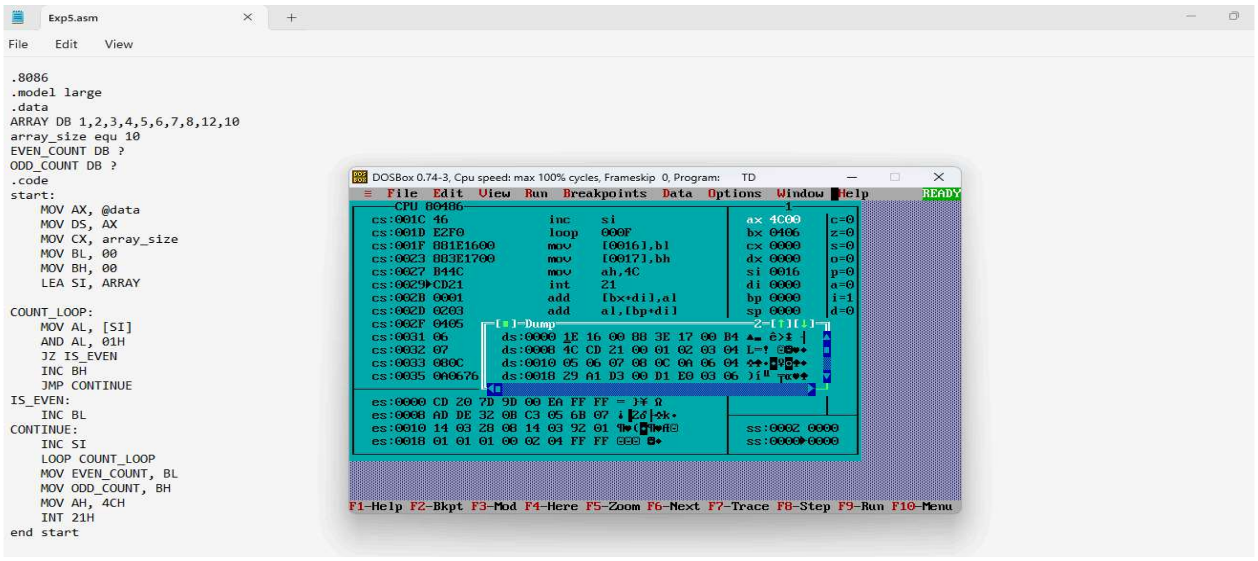
1

CODE:

```
.8086
.model large
.data
ARRAY DB 1,2,3,4,5,6,7,8,12,10
array_size equ 10
EVEN_COUNT DB ?
ODD_COUNT DB ?
.code
start:
    MOV AX, @data
    MOV DS, AX
    MOV CX, array_size
    MOV BL, 00
    MOV BH, 00
    LEA SI, ARRAY

COUNT_LOOP:
    MOV AL, [SI]
    AND AL, 01H
    JZ IS_EVEN
    INC BH
    JMP CONTINUE
IS_EVEN:
    INC BL
CONTINUE:
    INC SI
    LOOP COUNT_LOOP
    MOV EVEN_COUNT, BL
    MOV ODD_COUNT, BH
    MOV AH, 4CH
    INT 21H
end start
```

OUTPUT:



```
.8086
.model large
.data
ARRAY DB 1,2,3,4,5,6,7,8,12,10
array_size equ 10
EVEN_COUNT DB ?
ODD_COUNT DB ?
.code
start:
    MOV AX, @data
    MOV DS, AX
    MOV CX, array_size
    MOV BL, 00
    MOV BH, 00
    LEA SI, ARRAY

COUNT_LOOP:
    MOV AL, [SI]
    AND AL, 01H
    JZ IS_EVEN
    INC BH
    JMP CONTINUE
IS_EVEN:
    INC BL
CONTINUE:
    INC SI
    LOOP COUNT_LOOP
    MOV EVEN_COUNT, BL
    MOV ODD_COUNT, BH
    MOV AH, 4CH
    INT 21H
end start
```

DOSBox 0.74-3, Cpu speed: max 100% cycles, Frameskip 0, Program: TD
≡ File Edit View Run Breakpoints Data Options Window Help READY

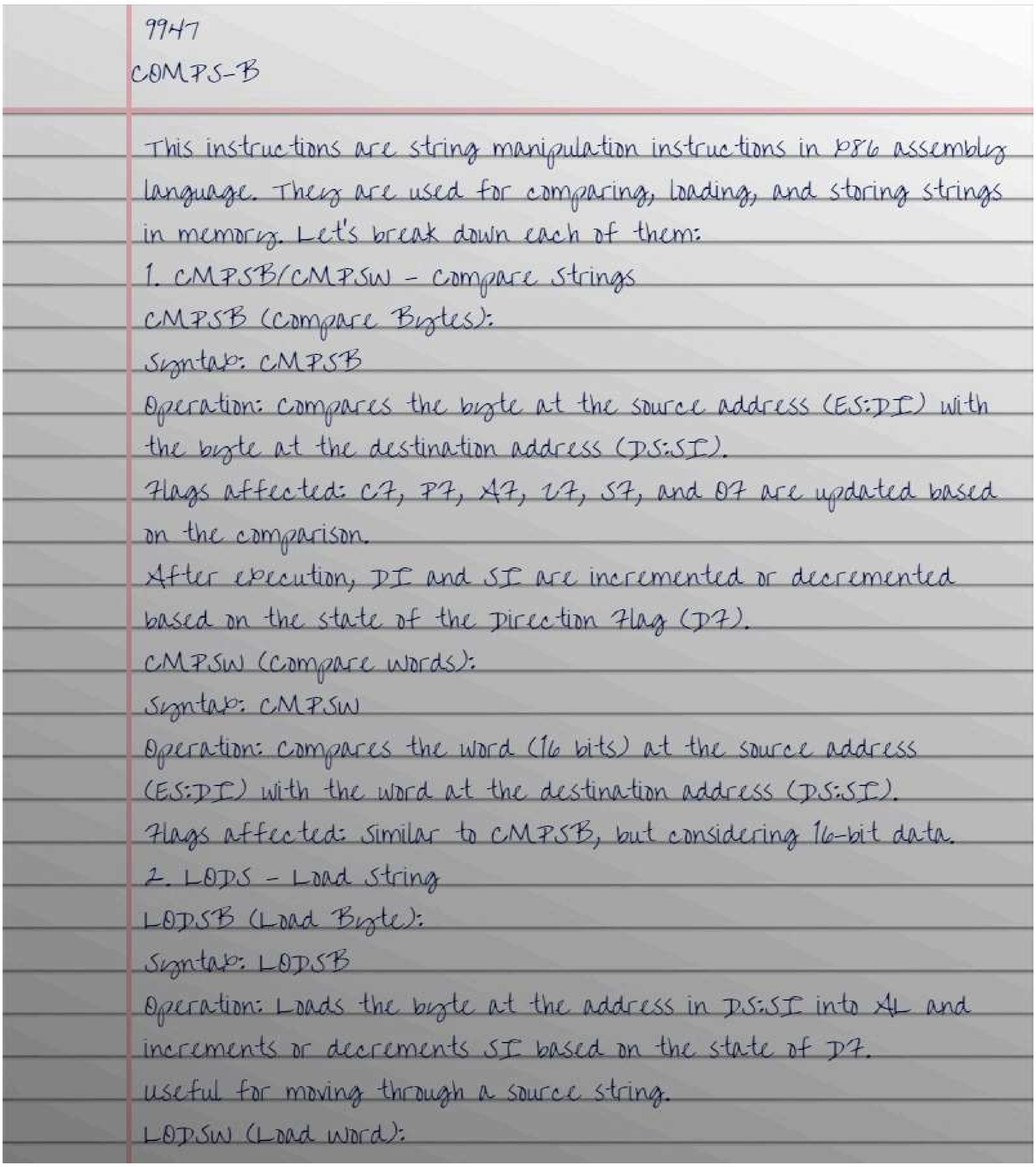
```
CPU 80486                                            1
 cs:001C 46              inc    si               ax 4C00   c=0
 cs:001D E2F0            loop   000F             bx 0406   z=0
 cs:001F 881E1600        mov    [0016],bl        cx 0000   s=0
 cs:0023 883E1700        mov    [0017],bh        dx 0000   o=0
 cs:0027 B44C            mov    ah,4C            si 0016   p=0
 cs:0029►CD21            int    21               di 0000   a=0
 cs:002B 0001            add    [bx+di],al       bp 0000   i=1
 cs:002D 0203            add    al,[bp+di]       sp 0000   d=0
 cs:002F 0405      [■]=Dump                     2=[↑][↓]
 cs:0031 06          ds:0000 1E 16 00 88 3E 17 00 B4
 cs:0032 07          ds:0008 4C CD 21 00 01 02 03 04
 cs:0033 080C        ds:0010 05 06 07 08 0C 0A 06 04
 cs:0035 0A0676      ds:0018 29 A1 D3 00 D1 E0 03 06

 es:0000 CD 20 7D 9D 00 EA FF FF
 es:0008 AD DE 32 0B C3 05 6B 07
 es:0010 14 03 28 08 14 03 92 01                  ss:0002 0000
 es:0018 01 01 01 00 02 04 FF FF                  ss:0000►0000
```

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

1

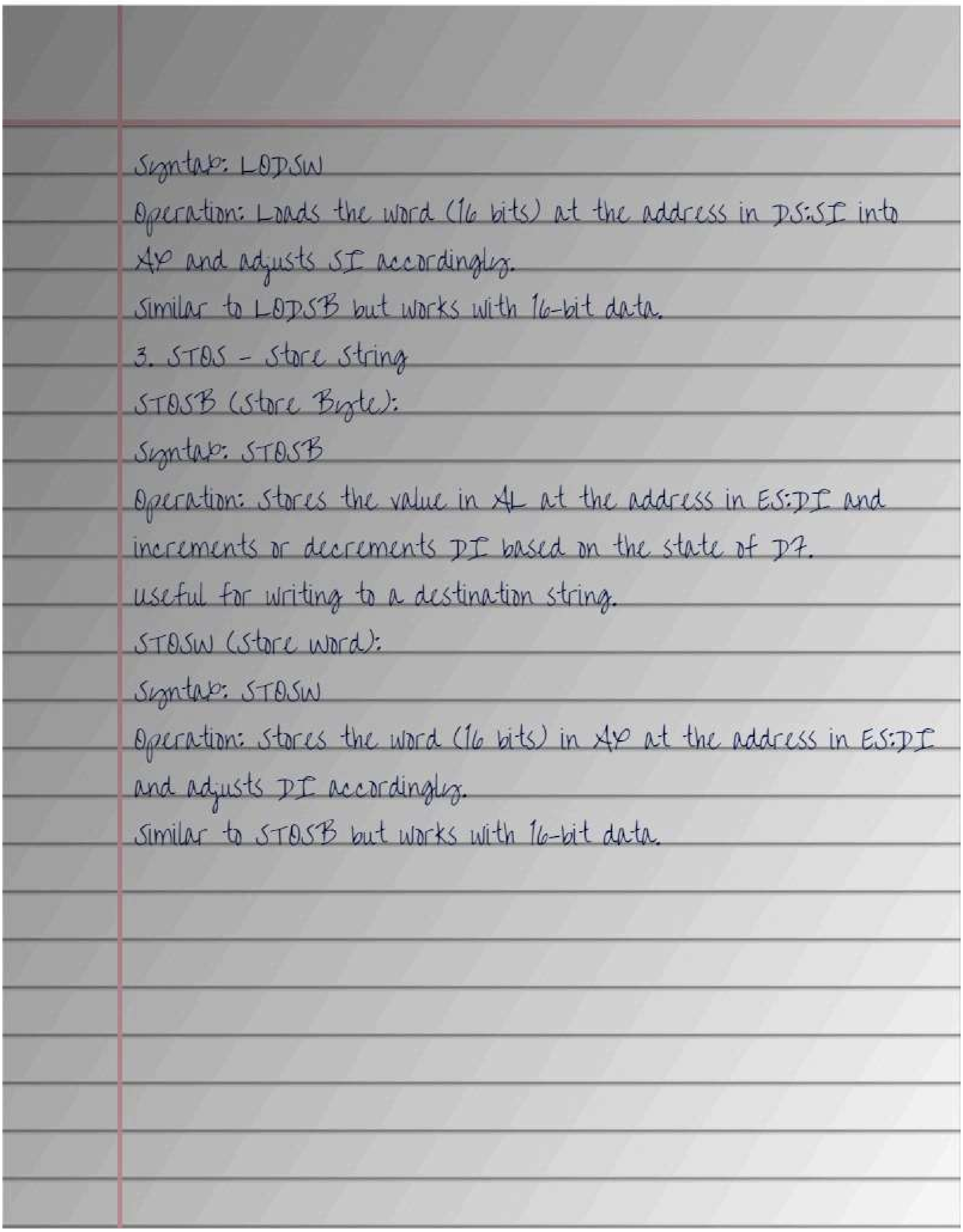
POSTLAB:



9947
COMPS-B

This instructions are string manipulation instructions in x86 assembly language. They are used for comparing, loading, and storing strings in memory. Let's break down each of them:

1. CMPSB/CMPSW - Compare strings
CMPSB (Compare Bytes):
Syntax: CMPSB
Operation: compares the byte at the source address (ES:DI) with the byte at the destination address (DS:SI).
Flags affected: CF, PF, AF, ZF, SF, and OF are updated based on the comparison.
After execution, DI and SI are incremented or decremented based on the state of the Direction Flag (DF).
CMPSW (Compare words):
Syntax: CMPSW
Operation: compares the word (16 bits) at the source address (ES:DI) with the word at the destination address (DS:SI).
Flags affected: Similar to CMPSB, but considering 16-bit data.
2. LODS - Load String
LODSB (Load Byte):
Syntax: LODSB
Operation: Loads the byte at the address in DS:SI into AL and increments or decrements SI based on the state of DF.
useful for moving through a source string.
LODSW (Load word):



Syntax: LODSW

Operation: Loads the word (16 bits) at the address in DS:SI into AX and adjusts SI accordingly.

Similar to LODSB but works with 16-bit data.

3. STOS - Store String

STOSB (Store Byte):

Syntax: STOSB

Operation: Stores the value in AL at the address in ES:DI and increments or decrements DI based on the state of DF.

useful for writing to a destination string.

STOSW (Store word):

Syntax: STOSW

Operation: Stores the word (16 bits) in AX at the address in ES:DI and adjusts DI accordingly.

Similar to STOSB but works with 16-bit data.