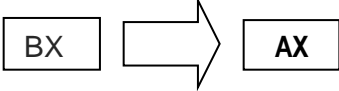
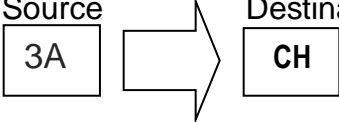
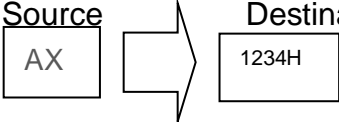
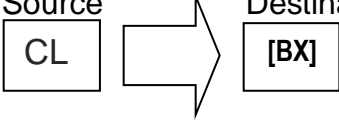


Activity No. 4																																																			
DATA TRANSFERS AND STRING INSTRUCTIONS																																																			
<b>1. Objective:</b>																																																			
To create programs using the data transfer and string instructions																																																			
<b>2. Intended Learning Outcomes (ILOs):</b>																																																			
<p>The students should be able to:</p> <p>2.1 Describe the data transfer instructions</p> <p>2.2 Describe the modes of addressing</p> <p>2.3 Create programs using the different transfers instructions</p> <p>2.4 Create programs using the string instructions</p>																																																			
<b>3. Discussion :</b>																																																			
<p style="text-align: center;"><b>DATA TRANSFER INSTRUCTIONS</b></p> <p>These instructions are used to move or transfer the values of the operators from a source to a destination. Table 3.1 lists the data transfer instructions and their description.</p> <table> <tr> <th>CODE</th><th>DESCRIPTION</th><th>CODE</th><th>DESCRIPTION</th></tr> <tr> <td><b>IN</b></td><td>Input byte or word from I/O port</td><td><b>LSS</b></td><td>Load pointer using SS</td></tr> <tr> <td><b>LAHF</b></td><td>Load AH from flags</td><td><b>MOVSX</b></td><td>Move with sign extended</td></tr> <tr> <td><b>LDS</b></td><td>Load pointer using data segment</td><td><b>MOVZX</b></td><td>Move with zero extended</td></tr> <tr> <td><b>LEA</b></td><td>Load effective address</td><td><b>POPAD</b></td><td>Pop all double (32-bit) register</td></tr> <tr> <td><b>LES</b></td><td>Load pointer using extra segment</td><td><b>POPD</b></td><td>Pop double register</td></tr> <tr> <td><b>MOV</b></td><td>Move to/from register/memory</td><td><b>POPFD</b></td><td>Pop double flag register</td></tr> <tr> <td><b>OUT</b></td><td>Output byet or word to I/O port</td><td><b>PUSHA</b></td><td>Push all double registers</td></tr> <tr> <td><b>POP</b></td><td>Pop word off stack</td><td><b>PUSHD</b></td><td>Push double register</td></tr> <tr> <td><b>POPF</b></td><td>Pop flags off stack</td><td><b>PUSHFD</b></td><td>Push double flag register</td></tr> <tr> <td><b>PUSH</b></td><td>Push word onto stack</td><td><b>BSWAP</b></td><td>Byte swap</td></tr> <tr> <td><b>PUSHF</b></td><td>Push flags onto stack</td><td><b>MOV</b></td><td>Move to/from control register</td></tr> </table>				CODE	DESCRIPTION	CODE	DESCRIPTION	<b>IN</b>	Input byte or word from I/O port	<b>LSS</b>	Load pointer using SS	<b>LAHF</b>	Load AH from flags	<b>MOVSX</b>	Move with sign extended	<b>LDS</b>	Load pointer using data segment	<b>MOVZX</b>	Move with zero extended	<b>LEA</b>	Load effective address	<b>POPAD</b>	Pop all double (32-bit) register	<b>LES</b>	Load pointer using extra segment	<b>POPD</b>	Pop double register	<b>MOV</b>	Move to/from register/memory	<b>POPFD</b>	Pop double flag register	<b>OUT</b>	Output byet or word to I/O port	<b>PUSHA</b>	Push all double registers	<b>POP</b>	Pop word off stack	<b>PUSHD</b>	Push double register	<b>POPF</b>	Pop flags off stack	<b>PUSHFD</b>	Push double flag register	<b>PUSH</b>	Push word onto stack	<b>BSWAP</b>	Byte swap	<b>PUSHF</b>	Push flags onto stack	<b>MOV</b>	Move to/from control register
CODE	DESCRIPTION	CODE	DESCRIPTION																																																
<b>IN</b>	Input byte or word from I/O port	<b>LSS</b>	Load pointer using SS																																																
<b>LAHF</b>	Load AH from flags	<b>MOVSX</b>	Move with sign extended																																																
<b>LDS</b>	Load pointer using data segment	<b>MOVZX</b>	Move with zero extended																																																
<b>LEA</b>	Load effective address	<b>POPAD</b>	Pop all double (32-bit) register																																																
<b>LES</b>	Load pointer using extra segment	<b>POPD</b>	Pop double register																																																
<b>MOV</b>	Move to/from register/memory	<b>POPFD</b>	Pop double flag register																																																
<b>OUT</b>	Output byet or word to I/O port	<b>PUSHA</b>	Push all double registers																																																
<b>POP</b>	Pop word off stack	<b>PUSHD</b>	Push double register																																																
<b>POPF</b>	Pop flags off stack	<b>PUSHFD</b>	Push double flag register																																																
<b>PUSH</b>	Push word onto stack	<b>BSWAP</b>	Byte swap																																																
<b>PUSHF</b>	Push flags onto stack	<b>MOV</b>	Move to/from control register																																																

<b>SAHF</b>	Store AH into flags	<b>POPA</b>	Pop all registers
<b>XCHG</b>	Exchange byte or word	<b>PUSHA</b>	Push all register
<b>XLAT</b>	Translate byte	<b>LFS</b>	Load pointer using FS
<b>INS</b>	Input string from port	<b>LGS</b>	Load pointer using GS
<b>OUTS</b>	Output string to port		

Table 3.1- The Data Transfer Instructions

Each instruction can be used with different modes of addressing. Addressing modes specifies the ways in which data is transferred. Table 3.1 shows the different addressing modes and their description.

Type	Description	Instruction	Address Generation
<b>Register</b>	Transfers a copy of a byte or word from the source register or contents of memory location to the destination register or memory location.	MOV AX, BX MOV AH, BH MOV CX, DX	<div>Source</div> <div>Destination</div> 
<b>Immediate</b>	Transfers the source, an immediate byte or word data into the destination register or memory location.	MOV CH, 3AH MOV AL, 22H MOV BX, 1234H	<div>Source</div> <div>Destination</div> 
<b>Direct</b>	Moves a byte or word between a memory location and a register.	MOV [1234H], AX MOV BX, [500H]	<div>Source</div> <div>Destination</div>  <p> <math>DS \times 10H + DISP</math>  <math>10000 + 1234H = 11234H</math> </p>
<b>Register Indirect</b>	Transfer a byte or word between a register and a memory location addressed by an index or base register	MOV [BX], CL MOV CX, [BX] MOV AL, [BX]	<div>Source</div> <div>Destination</div>  <p> <math>DS \times 10H + BX</math>  <math>10000 + 0300H = 10300H</math> </p>

<b>Base-plus-index Addressing</b>	Transfers a byte or word between a register and the memory location addressed by a register (BP or BX) plus an index register (DI or SI)	MOV [BX+SI],BP MOV BP, [BX+SI] MOV [BX+DI],CL	<div>Source</div> <div>BP</div> <div>Destination</div> <div>BX+SI</div> $DS \times 10H + BX + SI$ $10000 + 0300H + 0200H = 10500H$
<b>Register Relative Addressing</b>	Moves a byte or word between a register and the memory location addressed by an index or base register plus a displacement.	MOV CL, [BX+4] MOV AX, [BX+4]	<div>Source</div> <div>BX+4</div> <div>Destination</div> <div>CL</div> $DS \times 10H + BX + 4$ $10000 + 0300H + 4 = 10304H$
<b>Base-Index-plus-Displacement Addressing</b>	Transfer a byte or word between a register and the memory location addressed by an index register plus a displacement.	MOV ARRAY[BX+SI], DX	<div>Source</div> <div>BX+SI +4</div> <div>Destination</div> <div>CL</div> $DS \times 10H + ARRAY + BX + SI$ $10000 + 1000H + 0300H + 0200H = 11500H$

Figure 3.2- Addressing Modes and their description

### STRING INSTRUCTION

CODE	DESCRIPTION	CODE	DESCRIPTION
CMPS	Compare byte or word string	REPE(REPZ)	Repeat while equal (zero)
LODS	Load byte or word string	REPNE(RepNZ)	Repeat while not equal (not zero)
MOVS	Move byte or word string	SCAS	Scan byte or word string
MOVSB (MOVSW)	Move byte string (word string)	STOS	Store byte or word string
REP	Repeat		

Figure 3.3- String Instructions

#### 4. Resources:

PC  
TASM

#### 5. Procedure:

Using any text editor, write the following programs. Execute and show the output. Use the space provide on the Data and Results.

1.

```

.MODEL SMALL
.STACK 64
.DATA
    MSG DB 'Hi! How are you?$',
.CODE
    MOV AX, @DATA
    MOV DS, AX

    MOV AH, 02H
    MOV DH, 12
    MOV DL, 18
    INT 10H

    MOV AH, 09H
    MOV BX, 97H
    MOV CX, 10H
    INT 10H

    MOV AH, 09H
    LEA DX, MSG
    INT 21H
    MOV AH, 4CH
    INT 21H
    END
2.

```

```

dosseg
.model small
.stack
.data
    msg1 db 13,10,"Enter a character:$"
    msg2 db 13,10,"The character you entered is:$"
.code
main proc
    mov ax,@data
    mov ds,ax

    lea dx,msg1
    mov ah,09h
    int 21h

    mov ah,01h
    int 21h
    mov bl,al

    lea dx,msg2
    mov ah,09h

```

```
int 21h
mov dl,bl
mov ah,02h
int 21h
```

```
mov ax,4c00h
int 21h
```

```
main endp
end
```

3.

```
dosseg
```

```
.model small
```

```
.stack
```

```
.data
```

```
msg1 db 13,10,"Enter a string with dollar symbol as a break:$"
```

```
msg2 db 13,10,"Reverse of the string is:$"
```

```
strg db 20 DUP(0)
```

```
restr db 20 DUP(0)
```

```
.code
```

```
main proc
```

```
mov ax,@data
```

```
mov ds,ax
```

```
mov es,ax
```

```
mov di,00
```

```
lea dx,msg1
```

```
mov ah,09h
```

```
int 21h
```

```
read:mov ah,01h
```

```
int 21h
```

```
cmp al,24h
```

```
    je next
    inc di
    mov strg[di],al
    jmp read
next: mov si,00
start:cmp di,0
    je dmsg2
    mov al,strg[di]
    mov restr[si],al
    inc si
    dec di
    jmp start
dmsg2:lea dx,msg2
    mov ah,09h
    int 21h
dis:mov al,restr[di]
    cmp al,0
    je ou
    mov dl,al
    mov ah,02h
    int 21h
    inc di
    jmp dis
ou: mov ax,4c00h
    int 21h
main endp
end
```

<b>Course: BET-CPET</b>	<b>Activity No.: 4</b>
<b>Group No.:</b>	<b>Section: 2A</b>
<b>Group Members: Joseph C. Arenas</b>	<b>Date Performed: March 18, 2025</b>
	<b>Date Submitted: March 19, 2025</b>
	<b>Instructor: Ma'am Delos Trinos</b>
<b>6. DATA AND RESULTS:</b>	
<p>Problem 1:</p> <ul style="list-style-type: none"> <li>a. V</li> <li>b. V</li> <li>c. I</li> <li>d. I</li> <li>e. I</li> </ul> <p>Problem 2:</p> <pre> DOSSEG .MODEL SMALL .STACK 64H  .DATA MSG DB 'Hi! How do you do?\$',  .CODE START:     MOV AX, @DATA     MOV DS, AX      MOV AH, 02H     MOV DH, 12     MOV DL, 18     INT 10H      MOV AH, 09H     MOV BX, 97H     MOV CX, 12H     INT 10H      MOV AH, 09H     LEA DX, MSG     INT 21H      MOV AH, 4CH     INT 21H     END START </pre>	

Problem 3:

DOSSEG

.MODEL SMALL

.STACK 200H

.DATA

MSG DB 0DH, 0AH, "ENTER 10 CHARACTERS (Please end with dollar sign): \$"

MSG1 DB 0DH, 0AH, "YOU HAVE ENTERED: \$"

CHARACS DB 11 DUP (0)

.CODE

START:

MOV AX, @DATA

MOV DS, AX

MOV DI, 00

MOV AH, 09H

LEA DX, MSG

INT 21H

READ:

MOV AH, 01H

INT 21H

CMP AL, 24H

JE DISPLAY\_MSG

MOV CHARACS[DI], AL

INC DI

JMP READ

DISPLAY\_MSG:

MOV AH, 09H

LEA DX, MSG1

INT 21H

MOV DI, 0

PRINT\_CHAR:

CMP DI, 10

JGE END\_TIME

MOV AH, 02H

MOV DL, CHARACS[DI]

INT 21H

INC DI



JMP PRINT\_CHAR

END\_TIME:

MOV AX, 4C00H

INT 21H

END START

Problem 4:

DOSSEG

.MODEL SMALL

.STACK 100H

.DATA

MSG1 DB 0DH, 0AH, "Enter a string with dollar symbol as a break: \$"

MSG2 DB 0DH, 0AH, "Modified string is: \$"

BUFFER DB 100 DUP(0)

.CODE

START:

MOV AX, @DATA

MOV DS, AX

MOV AH, 09H

LEA DX, MSG1

INT 21H

MOV SI, 0

READ\_STRING:

MOV AH, 01H

INT 21H

CMP AL, 24H

JE MODIFY\_STRING

MOV BUFFER[SI], AL

INC SI

JMP READ\_STRING

MODIFY\_STRING:

MOV BUFFER[SI], '\$'

MOV SI, 0

MOV AH, 09H

LEA DX, MSG2

INT 21H

CONVERT\_CASE:

MOV AL, BUFFER[SI]

CMP AL, '\$'

JE DISPLAY\_RESULT

CMP AL, 'a'

JB CHECK\_UPPERCASE

CMP AL, 'z'

JA CHECK\_UPPERCASE

SUB AL, 32

JMP STORE\_CHAR

CHECK\_UPPERCASE:

CMP AL, 'A'

JB STORE\_CHAR

CMP AL, 'Z'

JA STORE\_CHAR

ADD AL, 32

STORE\_CHAR:

MOV BUFFER[SI], AL

INC SI

JMP CONVERT\_CASE

DISPLAY\_RESULT:

MOV AH, 09H

LEA DX, BUFFER

INT 21H

MOV AX, 4C00H

INT 21H

END START

**PROBLEMS:**

1. Indicate whether each of the following is valid (V-valid) or invalid(I-invalid)
  - a. MOV DX, 7F65H
  - b. MOV SI, -1
  - c. MOV SS,DS
  - d. MOV SI, CL
  - e. MOV ECX, 6F23458H
2. Modify Program 1 to display the string "Hello!, How do you do?"
3. Develop and execute a program to read 10 characters from keyboard

**SampleOutput:**

Enter 10 characters:1234567890

4. Write a program which converts string lower case character to upper case characters and upper case character to lower case characters.

**SampleOutput:**

Enter a string with dollar symbol as a break:eNgliSH  
Modified string is:EnGLIsh

**7. Conclusion:**

In conclusion, the interrupt 21h function ah= 1 significantly helps developers to include a user input into their program. By utilizing this function, various tasks can be achieved by programmers in order to formulate programs for the end-user's benefit.

**8. Assessment (Rubric for Laboratory Performance):**

