**PROGRAM CODE:**

MOV SI, 3000

MOV DI, 4002

MOV AX, [SI]

INC SI

INC SI

MOV [DI], 0000

ADD AX, [SI]

JNC 2014

MOV [DI], 0001

.DEC DI

DEC DI

MOV [DI], AX

HLT

**INPUT:**

3000: D5

3001: 7F

3002: 56

3003: EA

**OUTPUT:**

4000: 2B

4001: 6A

4002: 01

Name: A Theertha Date:

Roll No: 19

Class: CS5A

**EXPERIMENT- 3.2 (i)**

**16-bit Addition**

**AIM:**

Write a program to perform 16-bit addition using the 8086 training kit.

**ALGORITHM:**

Step-1: Start.

Step-2: Let SI hold the address of the memory location where the input is stored.

Step-3: Let DI hold the address of the memory location where the output will be stored.

Step-4: Transfer the content at the address that has been stored in the SI register to the AL register.

Step-5: Transfer the content at the address just after that to the AH register.

Step-6: Increment SI twice.

Step-7: Store the number zero at the address that has been stored in the DI register. It is used to indicate that the addition operation is assumed to have no carry.

Step-8: Add the content of the memory location whose address has been stored in the SI register and the one immediately after that with the content of the AX register. The sum will be stored in the AX register.

Step-9: If a carry was generated, store the number one in the memory location whose address is stored in the DI register.

Step-10: Decrement DI twice.

Step-11: Transfer the content of the AL register to the address stored in the DI register.

Step-12: Transfer the content of the AH register to the address immediately after that.

Step-13: Stop.

**RESULT:**

We have successfully performed 16-bit addition using the 8086 trainer kit.

**PROGRAM CODE:**

MOV SI, 3000

MOV DI, 4000

MOV AX, [SI]

INC SI

INC SI

MOV [DI], 0000

SUB AX, [SI]

JNC 2014

MOV [DI], 0001

.INC DI

MOV [DI], AX

HLT

**INPUT:**

3000: D5

3001: 7F

3002: 56

3003: EA

**OUTPUT:**

4000: 01

4001: 7F

4002: 95

Name: A Theertha Date:

Roll No: 19

Class: CS5A

**EXPERIMENT- 3.2 (ii)**

**16-bit Subtraction**

**AIM:**

Write a program to perform 16-bit subtraction using the 8086 training kit.

**ALGORITHM:**

Step-1: Start.

Step-2: Let SI hold the address of the memory location where the input is stored.

Step-3: Let DI hold the address of the memory location where the output will be stored.

Step-4: Transfer the content of the memory location at the address that has been stored in the SI register to the AL register.

Step-5: Transfer the content stored at the address just after that to the AH register.

Step-6: Increment SI twice.

Step-7: Transfer zero to the address that has been stored in the DI register. It is used to indicate that the result is not in the two’s complement form.

Step-8: Subtract the content of the memory location whose address has been stored in the SI register and the one immediately after that from the content of the AX register. The difference will be stored in the AX register.

Step-9: If a carry was generated, store the number one to the memory location whose address is stored in the DI register. This is used to indicate that the result is in two’s complement form.

Step-10: Increment DI.

Step-11: Transfer the content of the AL register to the address stored in the DI register.

Step-12: Transfer the content of the AH register to the address immediately after that.

Step-13: Stop.

**RESULT:**

We have successfully performed 16-bit subtraction using the 8086 trainer kit.

**PROGRAM CODE:**

MOV SI, 3000

MOV DI, 4000

MOV AX, [SI]

INC SI

INC SI

MOV BX, [SI]

MUL BX

MOV [DI], AX

INC DI

INC DI

MOV [DI], DX

HLT

**INPUT:**

3000: FF

3001: FF

3002: FF

3003: FF

**OUTPUT:**

4000: 01

4001: 00

4002: FE

4003: FF

Name: A Theertha Date:

Roll No: 19

Class: CS5A

**EXPERIMENT- 3.2 (iii)**

**16-bit Multiplication**

**AIM:**

Write a program to perform 16-bit multiplication using the 8086 training kit.

**ALGORITHM:**

Step-1: Start.

Step-2: Let SI hold the address of the memory location where the input is stored.

Step-3: Let DI hold the address of the memory location where the output will be stored.

Step-4: Transfer the content of the memory location at the address that has been stored in the SI register to the AL register.

Step-5: Transfer the content stored at the address just after that to the AH register.

Step-6: Increment SI twice.

Step-7: Transfer the content of the memory location at the address that has been stored in the SI register to the BL register.

Step-8: Transfer the content stored at the address just after that to the BH register.

Step-9: Multiply the content of the BX register with that of the AX register. The lower two bytes of the product will be stored in the AX register and the upper two bytes in the DX register.

Step-10: Transfer the content of the AL register to the address stored in the DI register.

Step-11: Transfer the content of the AH register to the address immediately after that.

Step-12: Increment DI twice.

Step-13: Transfer the content of the DL register to the address stored in the DI register.

Step-14: Transfer the content of the DH register to the address immediately after that.

Step-15: Stop.

**RESULT:**

We have successfully performed 16-bit multiplication using the 8086 trainer kit.

**PROGRAM CODE:**

MOV SI, 3000

MOV DI, 4000

MOV AX, [SI]

INC SI

INC SI

MOV BX, [SI]

MOV DX, 0000

DIV BX

MOV [DI], AX

INC DI

INC DI

MOV [DI], DX

HLT

**INPUT:**

3000: D5

3001: EA

3002: 1F

3003: 05

**OUTPUT:**

4000: 2D

4001: 00

4002: 62

4003: 04

Name: A Theertha Date:

Roll No: 19

Class: CS5A

**EXPERIMENT- 3.2 (iv)**

**16-bit Division**

**AIM:**

Write a program to perform 16-bit division using the 8086 training kit.

**ALGORITHM:**

Step-1: Start.

Step-2: Let SI hold the address of the memory location where the input is stored.

Step-3: Let DI hold the address of the memory location where the output will be stored.

Step-4: Transfer the content of the memory location at the address that has been stored in the SI register to the AL register.

Step-5: Transfer the content stored at the address just after that to the AH register. AX is now the dividend.

Step-6: Increment SI twice.

Step-7: Transfer the content of the memory location at the address that has been stored in the SI register to the BL register.

Step-8: Transfer the content stored at the address just after that to the BH register. BX is now the divisor.

Step-9: Divide the content of the BX register with that of the AX register. The quotient will be stored in the AX register and the remainder will be stored in the DX register.

Step-10: Transfer the content of the AL register to the address stored in the DI register.

Step-11: Transfer the content of the AH register to the address immediately after that.

Step-12: Increment DI twice.

Step-13: Transfer the content of the DL register to the address stored in the DI register.

Step-14: Transfer the content of the DH register to the address immediately after that.

Step-15: Stop.

**RESULT:**

We have successfully performed 16-bit division using the 8086 trainer kit.

**PROGRAM CODE:**

MOV SI, 3000

MOV DI, 4000

MOV [DI], 0000

MOV AX, [SI]

INC SI

INC SI

MOV CL, [SI]

INC SI

2010. CMP AX, [SI]

JNZ 201C

MOV [DI], 0001

INC DI

MOV [DI], SI

JMP 2025

201C INC SI

INC SI

DEC CL

CMP CL, 00

JNC 2010

2025 HLT

**INPUT:**

3000: 01

3001: 00

3002: 05

3003: 02

3004: 00

3005: 03

3006: 00

3007: 04

3008: 00

3009: 05

300A: 00

300B: 01

300C: 00

Name: A Theertha Date:

Roll No: 19

Class: CS5A

**EXPERIMENT- 3.3 (i)**

**Linear Search**

**AIM:**

Write a program to perform linear search on an array with 16-bit elements using the 8086 training kit.

**ALGORITHM:**

Step-1: Start.

Step-2: Let SI hold the address of the memory location where the input is stored.

Step-3: Let DI hold the address of the memory location where the output will be stored.

Step-4: Store the number zero at the address stored in the DI register. We are initially assuming that the element is not in the array.

Step-5: Transfer the content at the address that has been stored in the SI register to the AL register.

Step-6: Transfer the content at the address just after that to the AH register.

Step-7: Increment SI twice.

Step-8: Transfer the content at the address present in the SI register to the CL register.

Step-9: Increment SI.

**OUTPUT:**

4000: 01

4001: 0B

4002: 30

Step-10: Compare the content of the AX register with that of the content at the address stored in the SI register and the one immediately after it. If they are not equal, go to Step-14.

Step-11: Store the number one at the address stored in the DI register. This is done to show that we have found the element.

Step-12: Increment DI and transfer the lower two bytes stored in the SI register to the address that is present in the DI register.

Step-13: Transfer the upper two bytes stored in the SI register to the address immediately after that and go to Step-16.

Step-14: Increment SI twice and decrement CL.

Step-15: If CL holds a non-negative value, go to Step-5.

Step-16: Stop.

**RESULT:**

We have successfully performed linear search on an array with 16-bit elements using the 8086 training kit.

# **PROGRAM CODE:**

# MOV SI, 4000

# MOV BX, [SI]

# DEC BX

# MOV SI, 4000

# MOV CX, [SI]

# DEC CX

# INC SI

# MOV AX, [SI]

# INC SI

# INC SI

# CMP AX, [SI]

# JC 201E

# XCHG AX, [SI]

# DEC SI

# DEC SI

# XCHG AX, [SI]

# INC SI

# INC SI

# DEC CX

# JNZ 2007

# DEC BX

Name: A Theertha Date:

Roll No: 19

Class: CS5A

**EXPERIMENT NO 3.3(ii)**

# **SORTING OF 16 BIT NUMBERS USING 8086 TRAINER KIT**

# **AIM:**

To perform sorting of 16 – bit numbers using 8086 trainer kit.

**ALGORITHM:**

Step-1: Start.

Step-2: Initialise the registers SI, BX, CX and AX.

Step-3: Load SI with the address location 4000.

Step-4: Load BX with the value at the memory location pointed by SI.

Step-5: Decrement the value of the content in BX register.

Step-6: Store the decremented value back in memory address pointed by SI.

Step-7: Load SI with the address 4000.

Step-8: Load CX with the value at memory address pointed by SI.

Step-9: Decrement the value in CX.

Step-10: Increment the value in SI.

Step-11: Load AX with the value at the memory address pointed by SI.

Step-12: Increment SI twice to point to the next memory address.

Step-13: Compare AX with the value at the memory address pointed by SI.

Step-14: If AX is less than the value at the memory address, jump to label 201E.

Step-15: Exchange the value of AX and the memory address pointed by SI.

# **INPUT:**

# Memory Address Value

# 4000 05

# 4001 10

# 4002 15

# 4003 23

# 4004 22

# 4005 40

# 4006 80

# 4007 00

# 4008 30

# 4009 00

# 400A 10

# **OUTPUT:**

# Memory Address Value

# 4001 00

# 4002 10

# 4003 10

# 4004 15

# 4005 23

# 4006 22

# 4007 00

Step-16: Decrement SI twice.

# Step-17: Exchange the values of AX and the memory address pointed by SI.

# Step-18: Increment SI twice.

# Step-19: Decrement CX.

# Step-20: If CX is not zero, jump to label 2010

# Step-21: Decrement BX.

# Step-22: If BX is not zero, jump to label 2007.

# Step-23: Halt the execution.

# **RESULT:**

# Assembly program to perform sorting of 16 – bit numbers have been executed successfully.

# 4008 30

# 4009 40

# 400A 80

**PROGRAM CODE:**

**(i)String Palindrome**

DATA SEGMENT

MSG1 DB 10, 13, 'ENTER THE STRING: $'

MSG2 DB 10, 13, 'THE STRING IS PALINDROME$'

MSG3 DB 10, 13, 'THE STRING IS NOT PALINDROME$'

STR1 DB 50 DUP(0)

DATA ENDS

CODE SEGMENT

ASSUME CS: CODE, DS: DATA

START:

MOV AX, DATA

MOV DS, AX

LEA DX, MSG1

MOV AH, 09H

INT 21H

LEA SI, STR1

LEA DI, STR1

MOV AH, 01H

NEXT:

INT 21H

CMP AL, 0DH

JE TERMINATE

MOV [DI], AL

INC DI

JMP NEXT

TERMINATE:

MOV AL, '$'

MOV [DI], AL

Name: A Theertha Date:

Roll No: 19

Class: CS5A

**EXPERIMENT 3.5**

**STRING MANIPULATIONS USING 8086 EMULATOR**

**AIM:**

Implementation of String manipulations using 8086 emulator.

1. String Palindrome
2. String Reversal

**ALGORITHM:**

**(i)String Palindrome**

Step 0: Start

Step 1: Define the data segment:

1.1: Initialize MSG1 with the user prompt "ENTER THE STRING :".

1.2: Initialize MSG2 with message "THE STRING IS PALINDROME".

1.3: Initialize MSG3 with message "THE STRING IS NOT PALINDROME".

1.4: Define a buffer STR1 of 50 bytes initialized to 0.

Step 2: Begin the code segment and set up segment registers.

2.1: Use ASSUME directive to link the CS to code and DS to data.

2.2: Define START as the entry point of the program.

Step 3: Initialize the data segment.

3.1: Move the address of data into AX and then move AX to DS.

Step 4: Display the input prompt message.

4.1: Load the offset of MSG1 into DX.

DOTHIS:

DEC DI

MOV AL, [SI]

CMP [DI], AL

JNE NOTPALINDROME

INC SI

CMP SI, DI

JL DOTHIS

PALINDROME:

MOV AH, 09H

LEA DX, MSG2

INT 21H

JMP XX

NOTPALINDROME:

MOV AH, 09H

LEA DX, MSG3

INT 21H

XX:

MOV AH, 4CH

INT 21H

CODE ENDS

END START

4.2: Set AH to 09H to call DOS function for printing a string.

4.3: Call interrupt 21H to display "ENTER THE STRING :".

Step 5 : Load the address of STR1 into the pointers SI and DI.

Step 6: Start reading characters from user input.

6.1: Set AH to 01H to use the DOS function for reading a character.

Step 7: Read each character in a loop.

7.1: Call interrupt 21H to read a character and store it in AL.

* 1. : Check if the entered character is the "Enter" key.

7.3: If it is "Enter," jump to TERMINATE.

7.4: If not then proceed to the next step.

Step 8: Store the character in STR1.

8.1: Move the character in AL to the location pointed to by DI.

* 1. : Increment DI to move to the next position in the buffer.

8.3: Jump back to NEXT to continue reading the next character.

Step 9: Mark the end of the string.

* 1. :In TERMINATE, store a dollar sign ($) at the end of STR1.

Step 10: Initialize the comparison process.

* 1. : Decrement DI by 1 to point DI to the last character of the string.
  2. : Load the character at SI into AL to start the palindrome check from the beginning of the string.

Step 11: Compare characters for palindrome check.

11.1: Compare the character at SI with the character at DI.

11.2: If they are not equal, jump to NOTPALINDROME.

11.3: If they are equal, proceed to the next step.

Step 12: Increment and decrement pointers.

12.1: Increment SI to move to the next character from the beginning.

12.2: Check if SI is still less than DI. If true, jump back to DOTHIS to repeat the comparison.

12.3: If SI has reached or crossed DI, it means the string is a palindrome.

Step 13: Display the palindrome message.

**(ii)String Reversal**

DATA SEGMENT

MSG1 DB 10, 13, "ENTER THE STRING : $"

MSG2 DB 10, 13, “REVERSED STRING : $"

STR1 DB 1 DUP(0)

DATA ENDS

CODE SEGMENT

ASSUME CS: CODE, DS: DATA

START:

MOV AX, DATA

MOV DS, AX

LEA DX, MSG1

MOV AH, 09H

INT 21H

LEA SI, STR1

LEA DI, STR1

MOV AH, 01H

NEXT:

INT 21H

CMP AL, 0DH

JE TERMINATE

MOV [DI], AL

INC DI

JMP NEXT

TERMINATE:

MOV AL, "'"

MOV [DI], AL

INC DI

13.1: In PALINDROME, set AH to 09H, load the offset of MSG2 to DX

13.2: Call interrupt 21H to display "THE STRING IS PALINDROME".

13.3: Jump to the end of the program.

Step 14: Display the non-palindrome message.

* 1. : In NOTPALINDROME, set AH to 09H and load the offset of MSG3 into DX.
  2. :Call interrupt 21H to display "THE STRING IS NOT PALINDROME".

Step 15: Label XX marks the end of the program:

15.1: Set AH to 4CH to call DOS function for program termination.

15.2: Call interrupt 21H to exit.

Step 16: Stop

**(ii)String Reversal**

Step 0: Start

Step 1: Define the data segment.

1.1: Initialize MSG1 with the user prompt "ENTER THE STRING : ".

1.2: Initialize MSG2 with the message "REVERSED STRING : ".

* 1. : STR1 is defined as a 1-byte buffer initialized to 0 to hold the user-input string.

Step 2: Begin the code segment and set up segment registers.

* 1. : Use ASSUME directive to link CS to CODE and DS to DATA.

2.2: Define START as the entry point of the program.

Step 3: Initialize the data segment.

* 1. : Move the address of DATA into AX, then move AX into DS to set up the data segment.

Step 4: Display the input prompt message.

* 1. : Load the offset of MSG1 into DX.
  2. : Set AH to 09H to use DOS function for printing a string.

4.3: Call interrupt 21H to display "Enter a string:".

Step 5: Set up pointers for input.

MOV AL, "."

MOV [DI], AL

INC DI

MOV AL, "$"

MOV [DI], AL

DEC DI

DEC DI

DEC DI

DOTHIS:

MOV AL, [SI]

XCHG AL, [DI]

XCHG AL, [SI]

INC SI

DEC DI

CMP SI, DI

JL DOTHIS

LEA DX, MSG2

MOV AH, 09H

INT 21H

LEA DX, STR1

MOV AH, 09H

INT 21H

MOV AH, 4CH

INT 21H

CODE ENDS

END START

* 1. : Load the address of STR1 into both SI and DI. SI will serve as the beginning pointer, while DI will be used to store each character as it is read.

Step 6: Start reading characters from user input.

6.1: Set AH to 01H to use DOS function to read a single character.

Step 7: Read each character in a loop.

7.1: Call interrupt 21H to read a character into AL.

* 1. : Compare AL with 0DH to check if "Enter" was pressed.
  2. : If "Enter" is detected, jump to TERMINATE.
  3. : If not then proceed to the next step.

Step 8: Store the character in STR1.

8.1: Move the character in AL into the memory location pointed to by DI.

8.2: Increment DI to move to the next position in the buffer.

8.3: Jump back to NEXT to continue reading the next character.

Step 9: Mark the end of the string.

9.1 : In TERMINATE, add a single quote (') to the end of the string by moving it into AL and storing it at the location pointed to by DI.

* 1. : Increment DI to move to the next position.
  2. : Store a period (.) and a dollar sign ($) at the end to terminate the string.

Step 10: Set up pointers for reversing the string.

* 1. : Decrement DI three times to position it at the last character before the single quote, to prepare for the reversing process.

Step 11: Start reversing the string.

11.1: Load the character at SI into AL.

11.2: Exchange the character in AL with the character at DI.

11.3: Move the value in AL back to the memory location at SI.

11.4: Increment SI to move forward in the string.

11.5: Decrement DI to move backward in the string.

Step 12: Continue reversing until SI meets or exceeds DI.

12.1: Compare SI and DI.

**OUTPUT:**

**(i)String Palindrome**

ENTER THE STRING : MALAYALAM

THE STRING IS PALINDROME

ENTER THE STRING : MISSISSIPPI

THE STRING IS NOT PALINDROME

**(ii)String Reversal**

ENTER THE STRING : MALAYALAM

REVERSED STRING : MALAYALAM

ENTER THE STRING : MISSISSIPPI

REVERSED STRING : IPPISSISSIM

* 1. : If SI is less than DI, repeat the reversal process by jumping back to DOTHIS.

Step 13: Display the message "The reversed string is '".

13.1: Load the offset of MSG2 into DX.

13.2: Set AH to 09H and call interrupt 21H to display the message.

Step 14: Display the reversed string.

14.1: Load the offset of STR1 into DX.

14.2: Set AH to 09H and call interrupt 21H to display the reversed string.

Step 15: End the program.

15.1: Set AH to 4CH to use DOS function for program termination.

15.2: Call interrupt 21H to exit.

Step 16: Stop.

**RESULT:**

Successfully implemented string manipulation programs using 8086 emulator.

**PROGRAM CODE:**

0400: MOV AL,80

0402: OUT 46,AL

0404: MOV CL,04

0406: MOV BX,0500

0409: MOV AL,[BX]

040A: OUT 40,AL

040D: CALL 0415

0410: INC BX

0411: LOOPNZ 0409

0413: JMP 0404

0415: PUSH CX

0416: MOV CX, FFFF

0419: LOOPNZ 0419

041B: POP CX

041C: RET

**INPUT:**

MEMORY:

0500: 0A

0501: 06

0502: 05

0503: 09

0500: 09

0501: 05

0502: 06

0503: 0A

Name: A Theertha Date:

Roll No: 19

Class: CS5A

**EXPERIMENT 3.6**

**STEPPER MOTOR INTERFACING**

**AIM:**

Interfacing with stepper motor – Rotate through any given sequence.

**ALGORITHM:**

Step 0: Start

Step 1: Set AL register to 80 (MOV AL, 80).

Step 2: Output the value in AL to port 46 (OUT 46, AL).

Step 3: Set CL register to 04 (MOV CL, 04).

Step 4: Set BX register to the memory location 0500 (MOV BX, 0500).

Step 5: Load the value from the memory location pointed by BX into AL.

Step 6: Output the value in AL to port 40 (OUT 40, AL).

Step 7: Call the subroutine at address 0415 (CALL 0415).

Step 8: Increment the value in BX by 1 (INC BX).

Step 9: Loop back to Step 5 if the zero flag is not set (LOOPNZ 0409).

Step 10: Jump to the instruction at address 0404 (JMP 0404).

Step 11: Subroutine at address 0415:

11.1: Push the value in CX onto the stack (PUSH CX).

11.2: Set CX register to FFFF (MOV CX, FFFF).

11.3: Loop back to Step b if the zero flag is not set (LOOPNZ 0419).

11.4: Pop the value from the stack into CX (POP CX).

11.5: Return from the subroutine (RET).

Step 12: Stop.

**RESULT:**

Successful execution of the program and the output was verified.

**PROGRAM CODE:**

0400: MOV AL,00

0402: OUT 22,AL

0404: MOV AL,2D

0406: OUT 22,AL

0408: MOV AL,90

040A: OUT 22,AL

040C: MOV BX,041E

040F: MOV SI,0000

0412: MOV CX,0007

0415: MOV AL,[BX+SI]

0417: OUT 20,AL

0419: INC SI

041A: LOOPNZ 0415

041C: HLT

**INPUT:**

401E: 77

401F: 7F

4020: 39

4021: 3F

4022: 79

4023: 71

4024: 70

Name: A Theertha Date:

Roll No: 19

Class: CS5A

**EXPERIMENT 3.7**

**INTERFACING WITH 8279 DISPLAY**

**AIM:**

Interfacing with 8279(Static display implementation).

**ALGORITHM:**

Step 0: Start

Step 1: Set the value of register AL to 00.

Step 2: Output the value in register AL to the I/O port 22.

Step 3: Set the value of register AL to 2D.

Step 4: Output the value in register AL to the I/O port 22.

Step 5: Set the value of register AL to 90.

Step 6: Output the value in register AL to the I/O port 22.

Step 7: Load the memory address 041E into register BX.

Step 8: Set the value of register SI to 0000.

Step 9: Set the value of register CX to 0007.

Step 10: Enter a loop:

10.1: Load the byte at the memory address (BX + SI) into register AL.

10.2: Output the value in register AL to the I/O port 20.

10.3: Increment the value in register SI.

10.4: Decrement the value in register CX.

10.5: Repeat the loop if CX is not zero.

Step 11: Halt the processor.

Step 12: Stop

**RESULT:**

Successful execution of the program and the output was verified.