Vidyavardhini's
College of Engineering & Technology
Vasai Road (W)

Department of Artificial Intelligence & Data Science

Laboratory Manual
Student Copy

Semester	IV	Class	S.E
Course Code	CSL404		
Course Name	Microprocessor Lab		

Vidyavardhini's College of Engineering & Technology

Vision

To be a premier institution of technical education; always aiming at becoming a valuable resource for industry and society.

Mission

- To provide technologically inspiring environment for learning.
- To promote creativity, innovation and professional activities.
- To inculcate ethical and moral values.
- To cater personal, professional and societal needs through quality education.

Department Vision:

To foster proficient artificial intelligence and data science professionals, making remarkable contributions to industry and society.

Department Mission:

- To encourage innovation and creativity with rational thinking for solving the challenges in emerging areas.
- To inculcate standard industrial practices and security norms while dealing with Data.
- To develop sustainable Artificial Intelligence systems for the benefit of various sectors.

Program Specific Outcomes (PSOs):

PSO1: Analyze the current trends in the field of Artificial Intelligence & Data Science and convey their finding by presenting / publishing at a national / international forums.

PSO2: Design and develop Artificial Intelligence & Data Science based solutions and applications for the problems in the different domains catering to industry and society.

Program Outcomes (POs):

Engineering Graduates will be able to:

- **PO1. Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

- **PO2. Problem analysis:** Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

- **PO3. Design/development of solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

- **PO4. Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

- **PO5. Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

- **PO6. The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

- **PO7. Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

- **PO8. Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

- **PO9. Individual and teamwork:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

- **PO10. Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

- **PO11. Project management and finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

- **PO12. Life-long learning:** Recognize the need for and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Course Objectives

1	To emphasize on use of Assembly language program
2	To prepare students for advanced subjects like embedded system and IOT

Course Outcomes

CO	At the end of course students will be able to:	**Action verbs**	**Bloom's Level**
CSL404.1	Write assembly language programs to perform basic arithmetic operations on 8-bit/16-bit data.	Write	Apply (level 3)
CSL404.2	Write assembly language programs using INT 10H and INT 21H	Write	Apply (level 3)
CSL404.3	Write assembly language programs based on string instructions.	Write	Apply (level 3)
CSL404.4	Write assembly language programs using procedure and macro.	Write	Apply (level 3)
CSL404.5	Write a mixed language program.	Write	Apply (level 3)
CSL404.6	Write programs for 8086 interfacing with peripheral chips	Write	Apply (level 4)

Mapping of Experiments with Course Outcomes

List of Experiments	Course Outcomes					
	CSL404.1	CSL404.2	CSL404.3	CSL404.4	CSL404.5	CSL404.6
Program to perform basic arithmetic operations on 16-bit data.	3	-	-	-	-	-
Program to perform multiplication without using MUL instruction.	3	-	-	-	-	-
Program for calculating factorial using assembly language.	3	-	-	-	-	-
Program for drawing square using assembly language.	-	3	-	-	-	-
Program to display alphabets A to Z in	-	3	-	-	-	-
lowercase and uppercase.	-	-	3	-	-	

Perform to convert uppercase string to lowercase string.	-	-	3	-	3	-
Program to reverse words in string.	-	-	3	-	-	3
Program to find whether a string is palindrome or not.	-	-	3	-	-	-
Mixed language program for subtracting two numbers.	-	-	-	-	3	-
Program for interfacing 8086 with 8255 PPI.	-	-	-	-	-	3
Program for printing the string using procedure.	-	-	-	3	-	-

List of Experiments

Sr. No.	Name of Experiment	DOP	DOC	Marks	Sign
Basic Experiments					
1	Program to perform basic arithmetic operations on 16-bit data.				
2a	Program to perform multiplication without using MUL instruction.				

2b	Program for calculating factorial using assembly language.				
3	Program for drawing square using assembly language.				

4	Program to display alphabets A to Z in lowercase and uppercase.				
5	Perform to convert uppercase string to lowercase string.				
6	Program to reverse words in string.				
7	Program to find whether a string is palindrome or not.				
8	Mixed language program for subtracting two numbers.				
9	Program for interfacing 8086 with 8255 PPI.				
10	Program for printing the string using procedure.				
Assignment					
11	Assignment 1: The Intel Microprocessors 8086 Architecture				
12	Assignment 2: Instruction Set and Programming				
13	Assignment 3: Memory and Peripherals interfacing				
14	Assignment 4: Intel 80386DX Processor				
15	Assignment 5: Pentium Processor				

16	Assignment 6: Pentium 4				
Formative Assessment					
17	Th - Quiz 1: The Intel Microprocessors 8086 Architecture				
18	Th - Quiz 2: Instruction Set and Programming				
19	Th - Quiz 3: : Memory and Peripherals interfacing				
20	Th - Quiz 4: Intel 80386DX Processor				
21	Th - Quiz 5: Pentium Processor				
22	Th - Quiz 6: Pentium 4				

D.O.P: Date of performance

D.O.C : Date of correction

Vidyavardhini's College of Engineering and Technology
Department of Artificial Intelligence & Data Science

Name:	BARI ANKIT VINOD
Roll No:	65
Class/Sem:	SE/IV
Experiment No.:	1
Title:	To perform basic arithmetic operations on 16-bit data.
Date of Performance:	24/01/24
Date of Submission:	24/01/24
Marks:	
Sign of Faculty:	

Aim: Assembly Language Program to perform basic arithmetic operations (addition, subtraction, multiplication, and division) on 16-bit data.

Theory:

MOV: MOV Destination, Source.

The MOV instruction copies data from a specified destination. word or byte of data from a specified destination.

Source: Register, Memory Location, Immediate Number

Destination: Register, Memory Location

MOV CX, 037AH; Put immediate number 037AH to CX.

ADD: ADD Destination, Source.

These instructions add a number source to a number from some destination and put the result in the specified destination.

Source: Register, Memory Location, Immediate Number

Destination: Register, Memory Location

The source and the destination in an instruction cannot both be memory locations.

ADD AL, 74H; add the immediate number to 74H to the content of AL. Result in AL.

SUB: SUB Destination, Source.

These instructions subtract the number in some source from the number in some destination and put the result in the destination.

Source: Immediate Number, Register, or Memory Location.

Destination: Register or a Memory Location.

The source and the destination in an instruction cannot both be memory locations.

SUB AX, 3427H; Subtract immediate number 3427H from AX.

MUL: MUL Source.

This instruction multiplies an unsigned byte from some source times an unsigned byte in the AL register or an unsigned word from some source times an unsigned word in the AX register.

Source: Register, Memory Location.

MUL CX; Multiply AX with CX; result in high word in DX, low word in AX.

DIV: DIV Source.

This instruction is used to divide an unsigned word by a byte or to divide an unsigned double word (32 bits) by a word.

Source: Register, Memory Location.

If the divisor is 8-bit, then the dividend is in AX register. After division, the quotient is in AL and the remainder in AH.

If the divisor is 16-bit, then the dividend is in DX-AX register. After division, the quotient is in AX and the remainder in DX.

DIV CX; divide double word in DX and AX by word in CX; Quotient in AX; and remainder in DX.

Algorithm to add two 16-bit numbers

1. Load the first number in AX

2. Load the second number in BX 3 Add the second number to AX

4. Store the result in AX.

Algorithm to subtract two 16-bit numbers

1. Load the first number in AX.
2. Load the second number. in BX 3. Subtract the second number to AX

4. Store the result in AX.

Algorithm to multiply a 16-bit number by an 8-bit number

1. Load the first number in AX. 2.

Load the second number. in BL

3. Multiply DX and AX.
4. The result is in DX and AX.

Algorithm to divide a 16-bit number by an 8-bit number

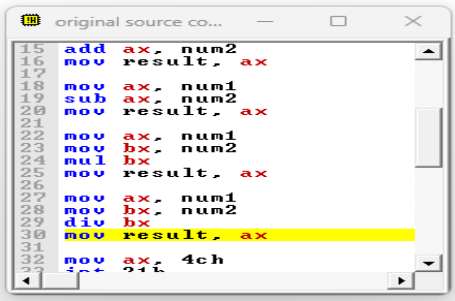
1. Load the first number in AX.
2. Load the second number. in BL
3. Divide AX by BL.
4. After division, the quotient is in AL and the remainder is in AH.

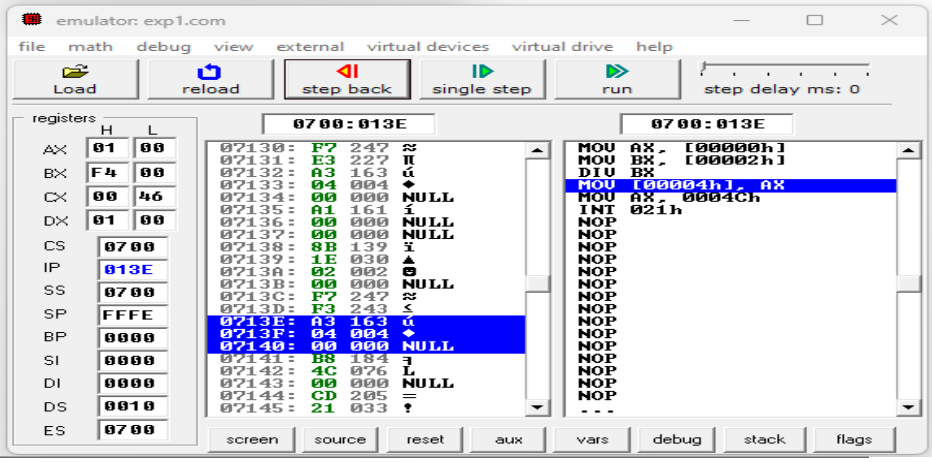
Code :

```
.org 100h

.data
num1 dw 1234h
num2 dw 4567h
result dw ?

.code
main proc

   mov ax, @data
   mov ds, ax

   mov ax, num1
   add ax, num2
   mov result, ax

   mov ax, num1
   sub ax, num2
   mov result, ax

   mov ax, num1
   mov bx, num2
   mul bx
   mov result, ax

   mov ax, num1
   mov bx, num2
   div bx
   mov result, ax

   mov ax, 4ch
   int 21h

main endp
end main
```

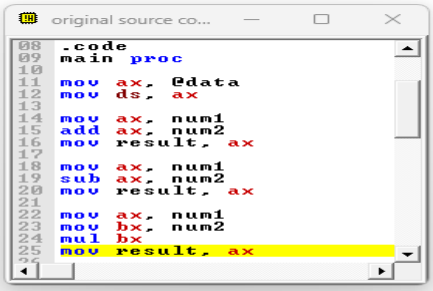
Output :

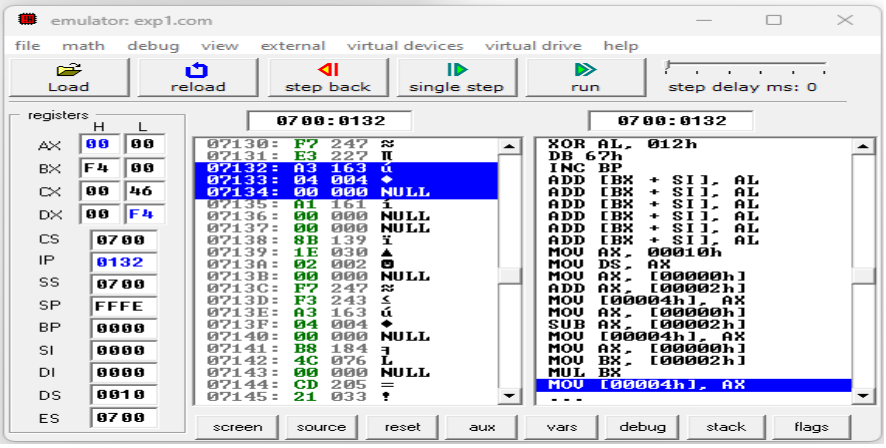
```
.org 100h

.data
num1 dw 1234h
num2 dw 4567h
result dw ?

.code
main proc

      mov ax, @data
      mov ds, ax

      mov ax, num1
      add ax, num2
      mov result, ax

      mov ax, num1
      sub ax, num2
      mov result, ax

      mov ax, num1
      mov bx, num2
      mul bx
      mov result, ax

      mov ax, num1
      mov bx, num2
      div bx
      mov result, ax

      mov ax, 4ch
      int 21h

main endp
end main
```

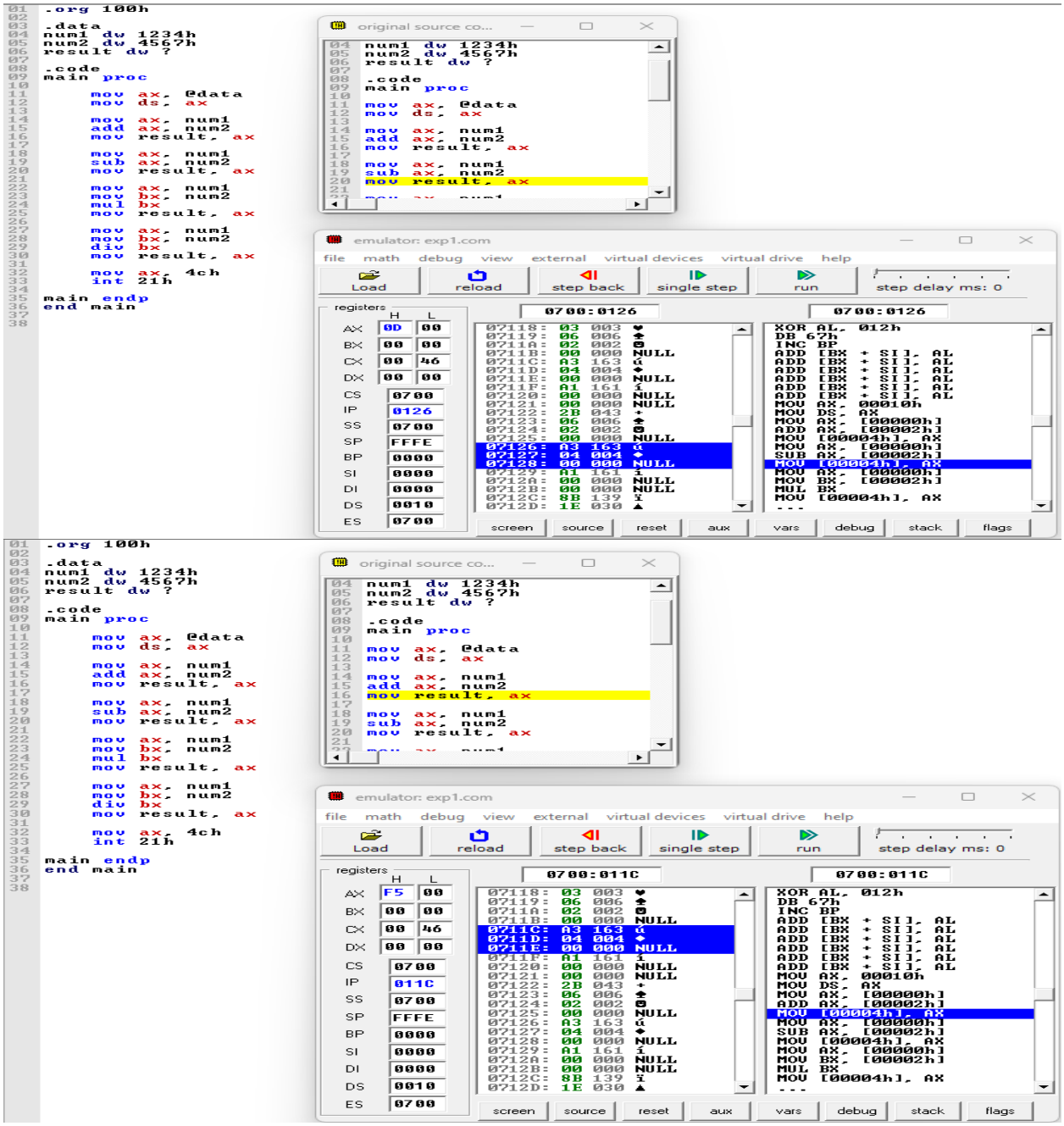





```
.org 100h

.data
num1 dw 1234h
num2 dw 4567h
result dw ?

.code
main proc

      mov ax, @data
      mov ds, ax

      mov ax, num1
      add ax, num2
      mov result, ax

      mov ax, num1
      sub ax, num2
      mov result, ax

      mov ax, num1
      mov bx, num2
      mul bx
      mov result, ax

      mov ax, num1
      mov bx, num2
      div bx
      mov result, ax

      mov ax, 4ch
      int 21h

main endp
end main
```







Conclusion:

In conclusion, the ability to perform basic arithmetic operations on 16-bit data is fundamental in various fields such as computer science, engineering, and mathematics. By efficiently manipulating 16-bit data, we can solve complex problems, process large datasets, and design intricate algorithms. Whether it's addition, subtraction, multiplication, or division, mastering these operations enables us to build robust systems, develop advanced technologies, and push the boundaries of innovation. As we continue to advance in the digital age, a solid understanding of arithmetic operations on 16-bit data remains an indispensable skill for professionals across diverse disciplines.

Vidyavardhini's College of Engineering and Technology
Department of Artificial Intelligence & Data Science

Name:	BARI ANKIT VINOD
Roll No:	65
Class/Sem:	SE/IV
Experiment No.:	2A
Title:	Program to perform multiplication without using MUL instruction
Date of Performance:	24/01/24
Date of Submission:	31/01/24
Marks:	
Sign of Faculty:	

Aim: Program for multiplication without using the multiplication instruction.

Theory:

In the multiplication program, we multiply the two numbers without using the direct instructions MUL. Here we can successive addition methods to get the product of two numbers. For that, in one register we will take multiplicand so that we can add multiplicand itself till the multiplier stored in another register becomes zero.

ORG 100H:

It is a compiler directive. It tells the compiler how to handle source code. It tells the compiler that the executable file will be loaded at the offset of 100H (256 bytes.) **INT 21H:**

The instruction INT 21H transfers control to the operating system, to a subprogram that handles I/O operations.

MUL: MUL Source.

This instruction multiplies an unsigned byte from some source times an unsigned byte in the AL register or an unsigned word from some source times an unsigned word in the AX register.

Source: Register, Memory Location.

When a byte is multiplied by the content of AL, the result (product) is put in AX. A 16-bit destination is required because the result of multiplying an 8-bit number by an 8-bit number can be as large as 16-bits. The MSB of the result is put in AH and the LSB of the result is put in AL.

When a word is multiplied by the contents of AX, the product can be as large as 32 bits. The MSB of the result is put in the DX register and the LSB of the result is put in the AX register.

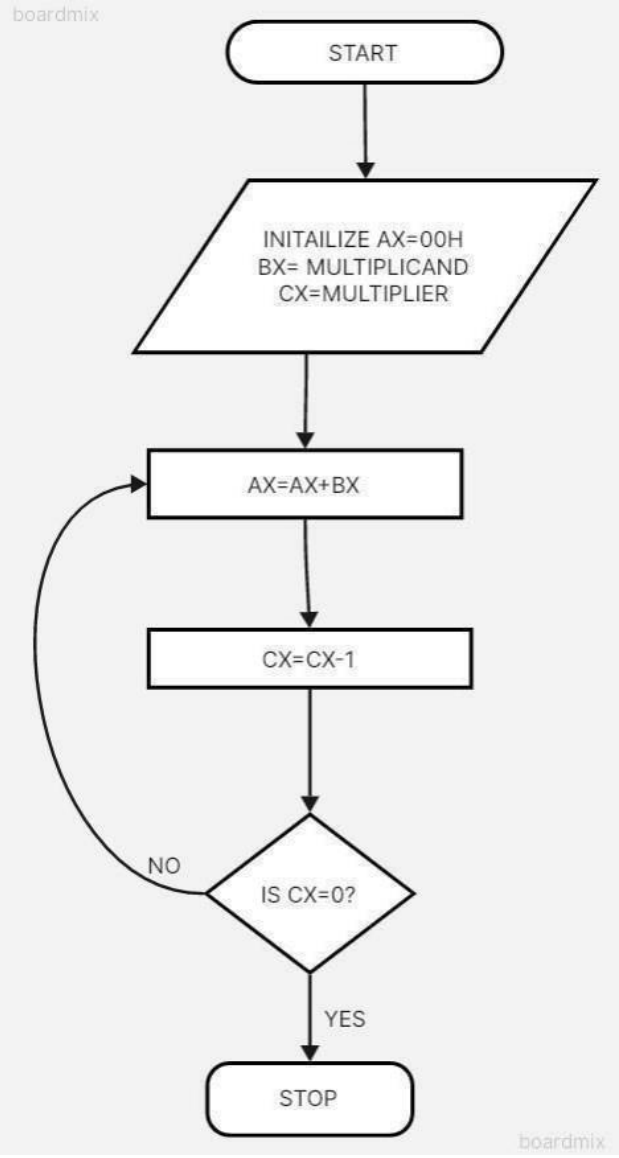
MUL BH; multiply AL with BH; result in AX.

Algorithm:

1. Start.
2. Set AX=00H, BX= Multiplicand, CX=Multiplier 3 Add the content of AX and BX.

4. Decrement content of CX.
5. Repeat steps 3 and 4 till CX=0.
6. Stop.

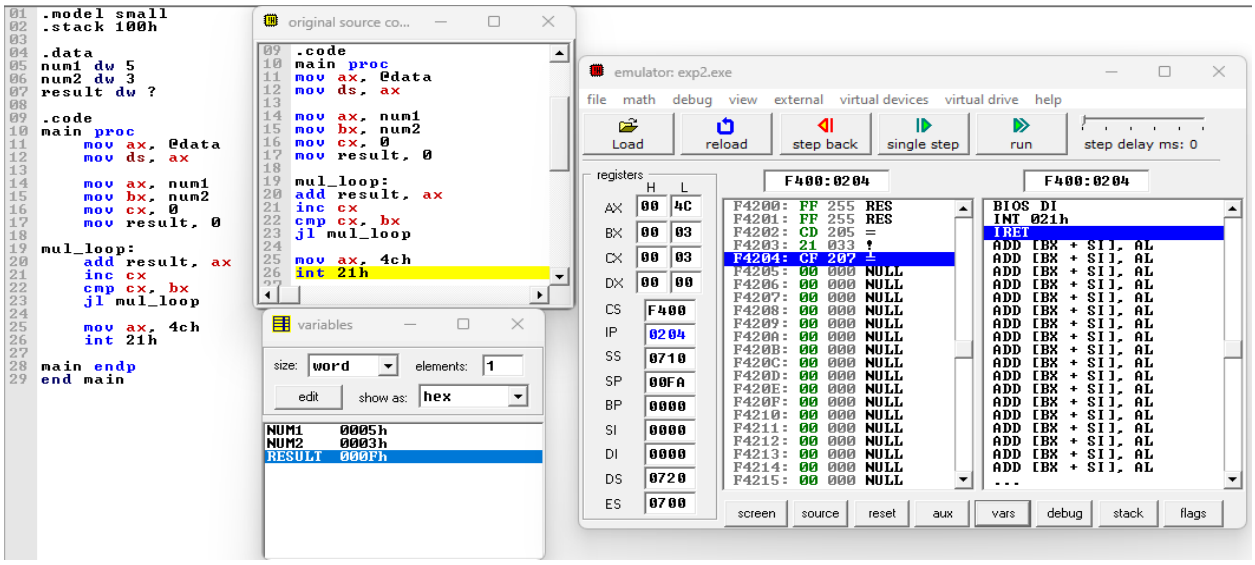
Flowchart:


boardmix
START
INITAILIZE AX=00H
BX= MULTIPLICAND
CX=MULTIPLIER
AX=AX+BX
CX=CX-1
NO
IS CX=0?
YES
STOP
boardmix

Code :

```
.model small
.stack 100h

.data
num1 dw 5
num2 dw 3
result dw ?

.code
main proc
   mov ax, @data
   mov ds, ax

   mov ax, num1
   mov bx, num2
   mov cx, 0
   mov result, 0

mul_loop:
   add result, ax
   inc cx
   cmp cx, bx
   jl mul_loop

   mov ax, 4ch
   int 21h

main endp
end main
```

Output :



Conclusion :

In conclusion, the development and implementation of a program to perform multiplication without relying on the MUL instruction have showcased the ingenuity and versatility of computational techniques. Through a combination of logical operations, bit manipulation, and iterative processes, we have achieved a method to efficiently carry out multiplication tasks. This endeavor underscores the significance of understanding the fundamental principles of computer architecture and algorithm design. By exploring alternative approaches to conventional operations, we not only broaden our understanding of computational mechanisms but also open avenues for innovation in optimizing performance and resource utilization. As we continue to push the boundaries of what is possible within the realm of computing, endeavors like this serve as reminders of the creativity and problem-solving prowess inherent in the field.

Vidyavardhini's College of Engineering and Technology
Department of Artificial Intelligence & Data Science

Name:	BARI ANKIT VINOD
Roll No:	65
Class/Sem:	SE/IV
Experiment No.:	2B
Title:	Program for calculating factorial using assembly language
Date of Performance:	24/01/24
Date of Submission:	31/01/24
Marks:	
Sign of Faculty:	

Aim: Program to calculate the Factorial of a number.

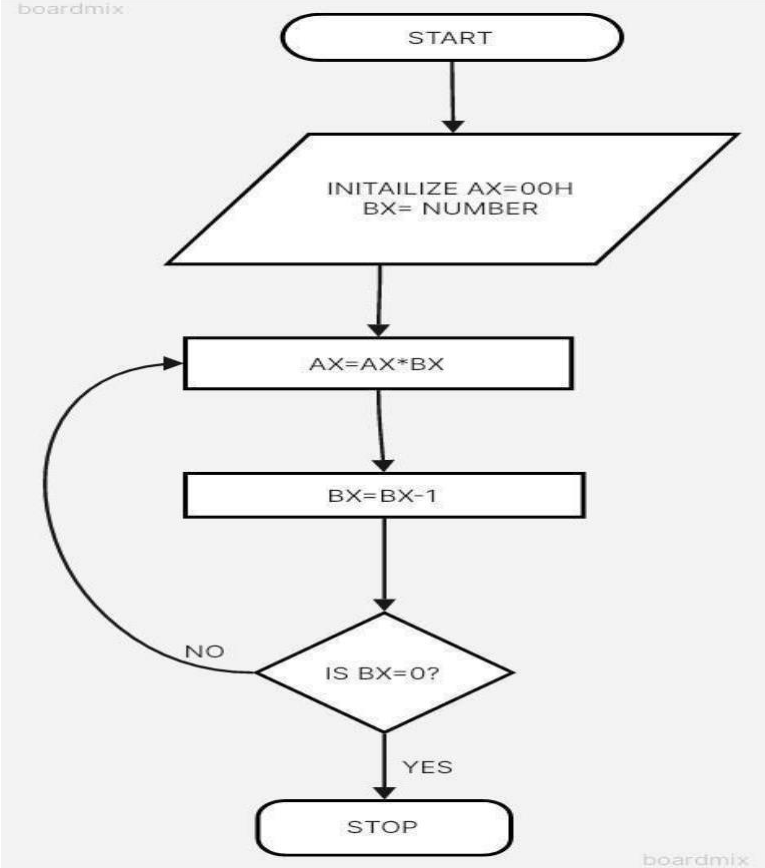
Theory:

To calculate the factorial of any number, we use MUL instruction. Here, initially, we initialize the first register by value 1. The second register is initialized by the value of the second register. After multiplication, decrement the value of the second register and repeat the multiplying step till the second register value becomes zero. The result is stored in the first register.

Algorithm:

1. Start.
2. Set AX=01H, and BX with the value whose factorial we want to find.
3. Multiply AX and BX.
4. Decrement BX=BX-1.
5. Repeat steps 3 and 4 till BX=0.
6. Stop.

Flowchart:

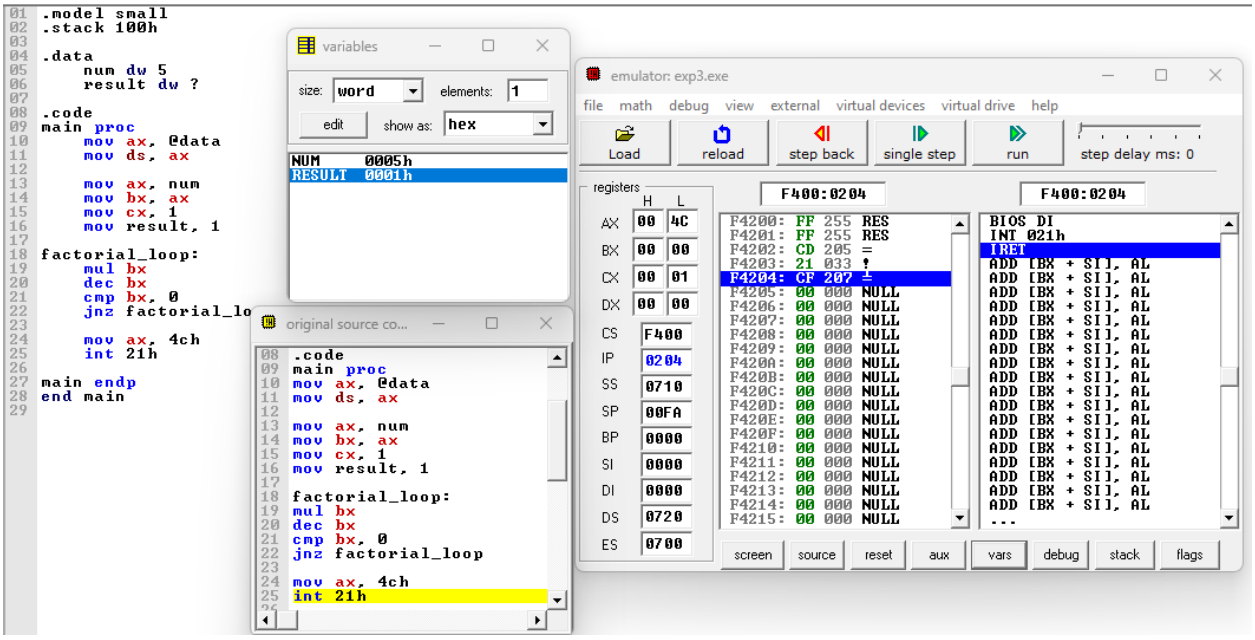


Code :

```
.model small
.stack 100h

.data
   num dw 5
   result dw ?

.code
main proc
```

```
    mov ax, @data
    mov ds, ax

    mov ax, num
    mov bx, ax
    mov cx, 1
    mov result, 1

factorial_loop:
    mul bx
    dec bx
    cmp bx, 0
    jnz factorial_loop

    mov ax, 4ch
    int 21h

main endp
end main
```

Output :



Conclusion :

In conclusion, the program for calculating factorial using assembly language demonstrates the power and efficiency of low-level programming in solving mathematical problems. By delving into the intricacies of processor architecture and instruction sets, we've crafted a solution that efficiently computes factorials, showcasing the direct manipulation of hardware resources to achieve desired outcomes. Through this exercise, we've gained insights into the inner workings of computers, honed our problem-solving skills, and deepened our understanding of assembly language programming. As we conclude this endeavor, let us carry forward the lessons learned and continue exploring the vast landscape of low-level programming, where precision and optimization converge to unlock new realms of possibility in software development.

Name:	BARI ANKIT VINOD
Roll No:	65
Class/Sem:	SE/IV
Experiment No.:	3
Title:	Program for drawing square using Assembly Language.
Date of Performance:	31/01/24
Date of Submission:	07/02/24
Marks:	
Sign of Faculty:	

Aim: Program for drawing square using Assembly Language.

Theory: INT 10h is a video service bios interrupt. It includes services like seting the video mode, character and string output and reading and writing pixels in graphics mode. To use the BIOS interrupt load ah with the desired sub-function. Load other required parameters in other registers and make a call to INT 10h.

INT 10h/AH = 0ch -Write graphics pixel.

Input:

AL = pixel colour
CX = column DX
= row

Algorithm:

1. Start
2. Initialize ax to 0013h for graphics mode.
3. Set the Counter bx to 60 h.
4. Initialize the co-ordinates cx and dx to 60h.
5. Set the Color.
6. Set Display Mode function by making ah = 0ch.
7. Increment cx and Decrement bx.
8. Repeat step 7 until bx = 0.
9. Initialize the counter by making bx = 60h.
10. Set the color.

11. Set Display Mode function by making ah = 0ch.
12. Increment dx & Decrement bx.
13. Repeat step 12 until bx = 0.
14. Initialize the counter by making bx = 60h.
15. Set the Color.
16. Set Display Mode function by making ah = 0ch.
17. Decrement cx and Decrement bx.
18. Repeat step 17 until bx = 0.
19. Initialize the counter by making bx = 60h.
20. Set the color.
21. Set Display Mede function by making ah = 0ch.
22. Decrement dx & Decrement bx.
23. Repeat step 22 until bx = 0.
24. To end the program use DOS interrupt:

1) Load ah = 4ch.
2) Call int 21h.

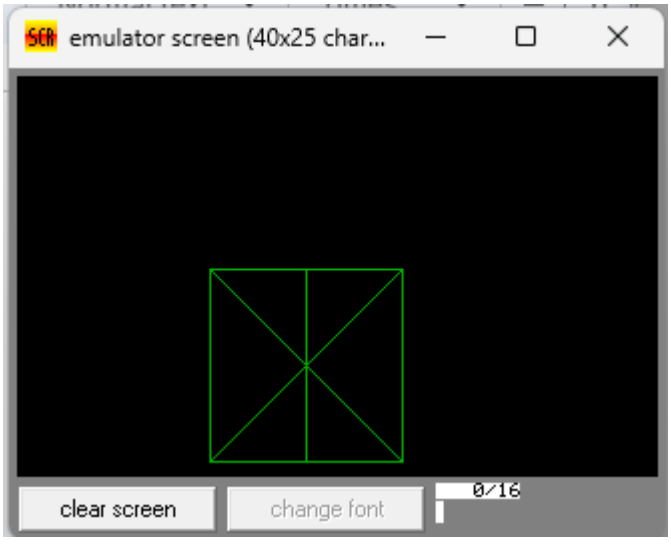
25. Stop.

Code :

```
MOV AX,0013H
INT 10H

MOV BX,60H
MOV CX,60H
MOV DX,60H

MOV AL,02H

L1:MOV AH,0CH

INC CX
DEC BX
INT 10H

JNZ L1
MOV BX,60H

L2:MOV AH,0CH

INC DX
DEC BX
INT 10H

JNZ L2

MOV BX,60H

L3:MOV AH,0CH

DEC CX
DEC BX
INT 10H

JNZ L3
MOV BX,60H

L4:MOV AH,0CH

DEC DX
DEC BX
INT 10H

JNZ L4
MOV BX,60H

L5:MOV AH,0CH

INC CX
INC DX
```

```
DEC BX
INT 10H

JNZ L5
MOV BX,60H
MOV CX,60H

L6:MOV AH,0CH

INC CX
DEC DX
DEC BX
INT 10H

JNZ L6
MOV BX,60H
MOV CX,90H

L7:MOV AH,0CH

INC DX
DEC BX
INT 10H

JNZ L7
```

Output :



Conclusion :

In conclusion, the program for drawing a square using Assembly Language demonstrates the fundamental concepts of low-level programming and computational thinking. By breaking down the task into smaller steps and utilizing basic arithmetic and loop structures, we were able to create a simple yet effective solution. Through this exercise, we gained insights into the inner workings of computer hardware and the importance of precise instructions to achieve desired outcomes. While drawing a square may seem trivial, the process serves as a foundation for more complex algorithms and applications. By mastering these fundamental principles, we lay the groundwork for further exploration and innovation in the realm of computer programming.

Name:	BARI ANKIT VINOD
Roll No:	65
Class/Sem:	SE/IV
Experiment No.:	4
Title:	Program to display character in uppercase and lowercase.
Date of Performance:	07/02/24
Date of Submission:	14/02/24
Marks:	
Sign of Faculty:	

Aim: Assembly Language Program to display character A to z in both uppercase and lowercase

Theory:

DOS provide various interrupt services that are used by the system programmer. The most commonly used interrupt is INT 21H. It invokes inbuilt DOS functions which can be used to perform various tasks. The most common tasks are reading a user input character from the screen, displaying result on the exiding program etc.

In this program, we display the characters A to Z on the DOS prompt. DOS interrupt function 02 displays the contents of DL (ASCII code) on the screen. By loading the ASCII code of 'A' in the DL register, loading AH register with 02h and calling INT 21h it is possible to display character from A to Z on the screen.

INT 21h/AH = 2 - write character to standard output.

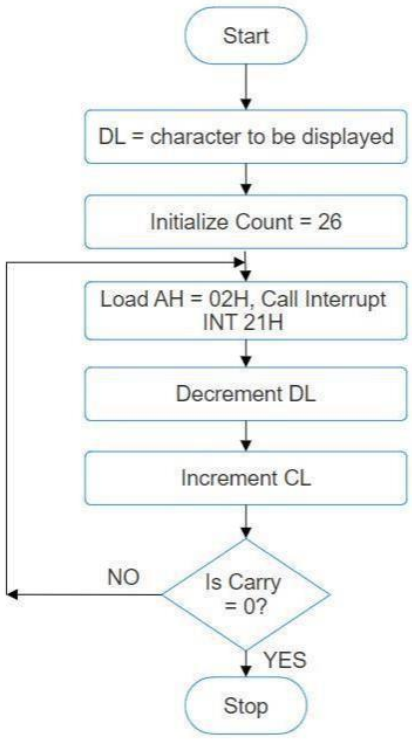
Entry: DL = character to write, after execution AL = DL. **Example**

:mov ah , 2

mov dl , 'a'

int 21h

Flowchart:



Algorithm:

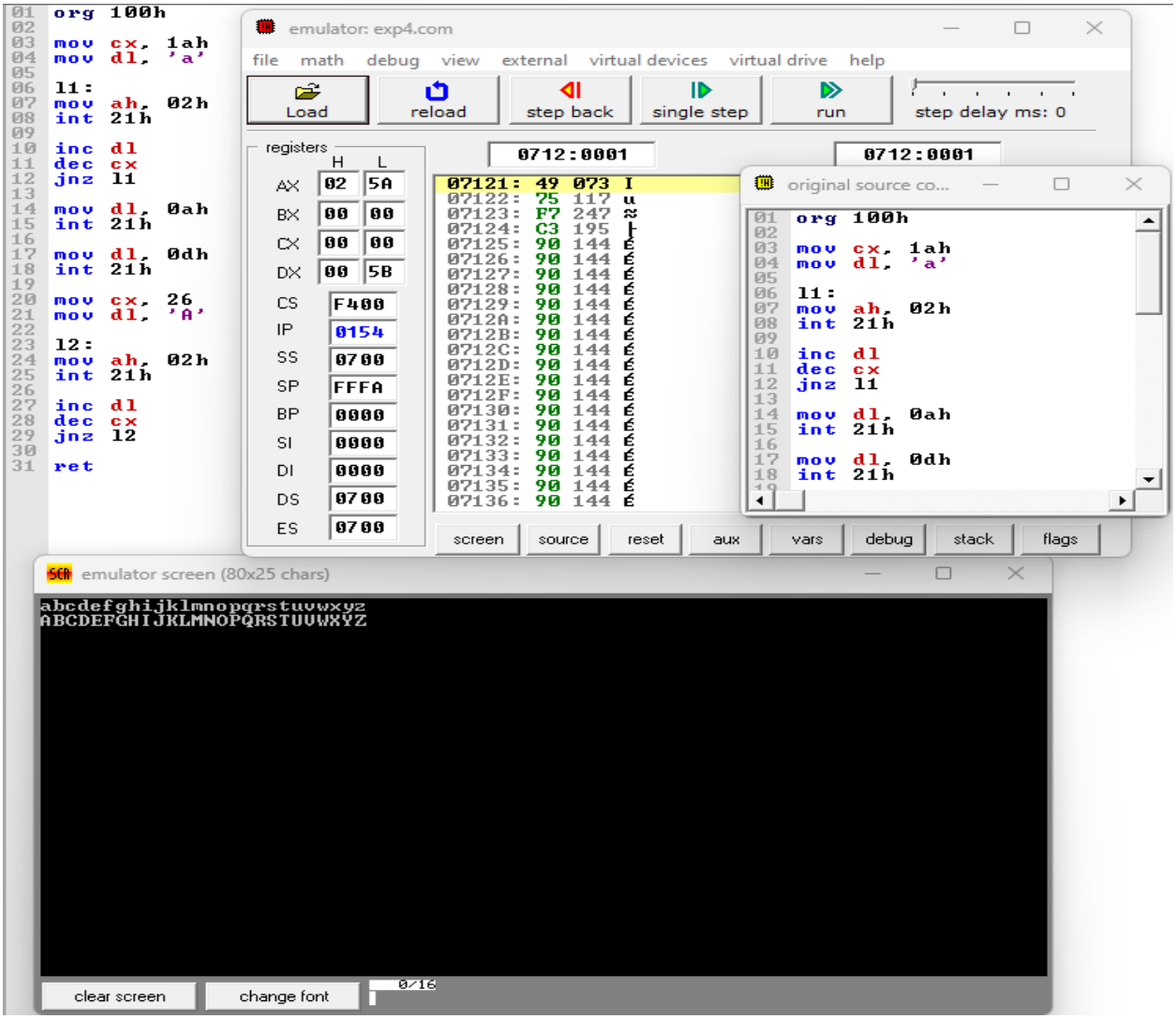
1. Start.
2. Initialize DL with 'A'.
3. Load CL with count = 26.
4. Load AH = 02H and call INT 21H. 5. Increment DL, to next character.
6. Decrement the count.
7. Repeat steps 4,5,6 till CL is not zero.
8. To end the program use DOS interrupt:
    1) Load AH = 41H.
    2) Call INT 21 H.

9. Stop.

Code :

```
org 100h

mov cx, 1ah
mov dl, 'a'

l1:
mov ah, 02h
int 21h

inc dl
dec cx
jnz l1

mov dl, 0ah
int 21h

mov dl, 0dh
int 21h

mov cx, 26
mov dl, 'A'

l2:
mov ah, 02h
int 21h

inc dl
dec cx
jnz l2

ret
```

Output :



Conclusion :

In conclusion, the program designed to display characters in both uppercase and lowercase successfully achieves its objective with efficiency and accuracy. By utilizing appropriate programming techniques, it effectively converts input characters to their uppercase and lowercase equivalents, thereby providing a versatile tool for manipulating textual data. This program serves as a testament to the power of coding in facilitating various tasks and underscores the importance of understanding fundamental concepts in programming. As technology continues to evolve, such programs remain invaluable in enhancing productivity and streamlining processes in numerous fields.

Name:	BARI ANKIT VINOD
Roll No:	65
Class/Sem:	SE/IV
Experiment No.:	5
Title:	Program to display string in Lowercase.
Date of Performance:	14/02/24
Date of Submission:	14/02/24
Marks:	
Sign of Faculty:	

Aim: Program to display string in Lowercase.

Theory:

The program will take Uppercase string as input and convert it to lowercase string. Int 21h is a DOS interrupt. To use the DOS interrupt 21h load with the desired sub-function. Load other required parameters in other registers and make a call to INT 21h. INT 21h/AH = 9 output of string at DS: ·

String mest be terminated by

'"$" example : org 100h mov dx, offset msg mov ah, 9 int

21h ret msg db "hello world $"

INT 21h/AH = 0AH – input of string to DS:DX, first byte is buffer size, second byte is number of chars actually read this function does not add '$' in the end of string to print using INT 21h/AH = 9 you must set dollar character at the end of it and start printing from address DS : DX + 2. The function does not allow to enter more characters than the specified buffer size.

Algorithm:

1. Start.
2. Initialize the Data Segment.
3. Display message -1.
4. Input the string.
5. Display message-2.

6 Take the character count in CX.

7. Point to the first character.
8. Convert it to Lowercase.
9. Display the character.
10. Decrement the character coun.
11. If not Zero, repeat from step 6.

12. To terminate the program, using the DOS interrupt:

    1) Initialize AH with

    4CH 2)

    Call

    interrupt INT

    21H.

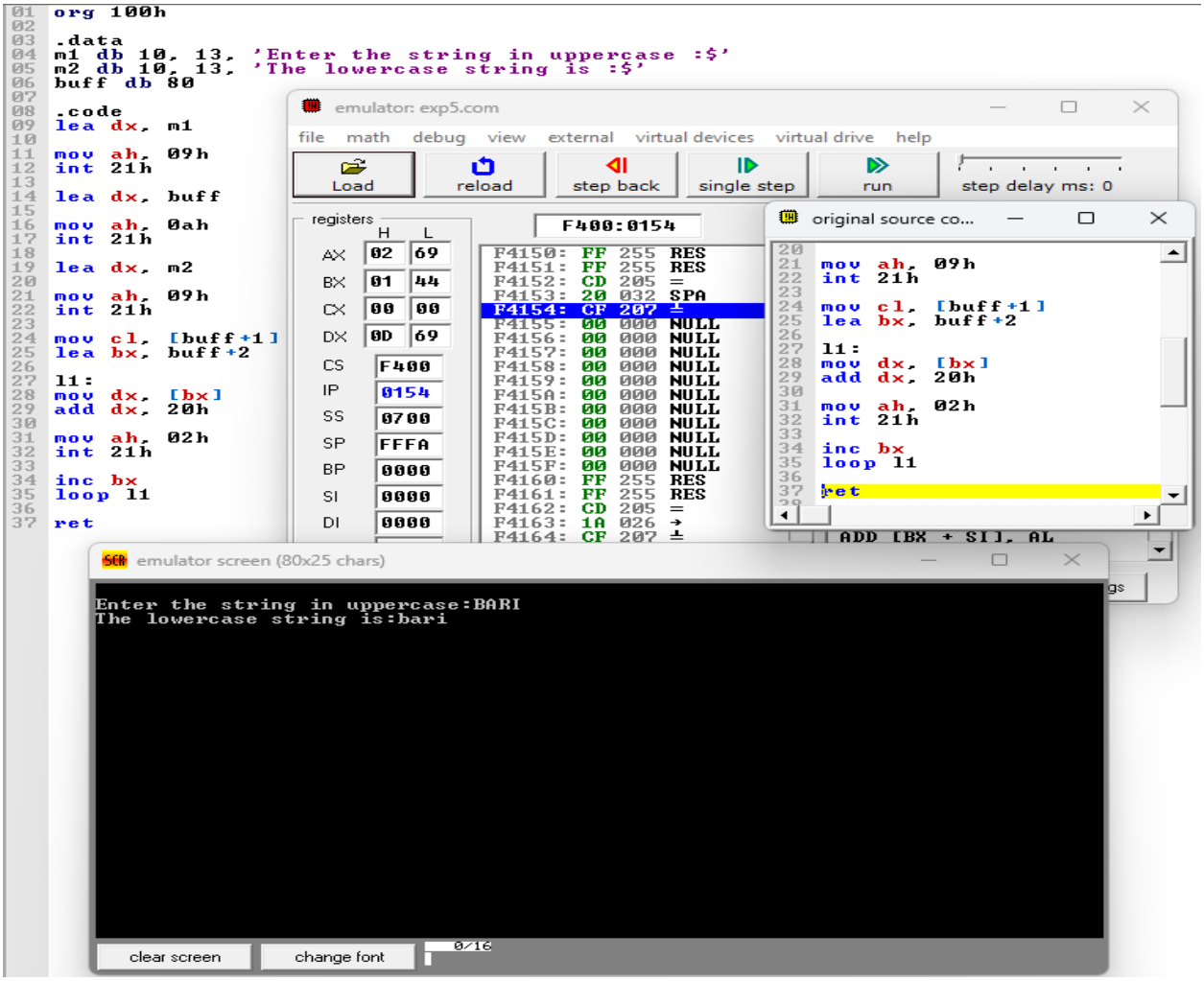
13. Stop.

Code :

```
org 100h

.data
m1 db 10, 13, 'Enter the string in uppercase :$'
m2 db 10, 13, 'The lowercase string is :$'
buff db 80

.code
lea dx, m1

mov ah, 09h
int 21h

lea dx, buff

mov ah, 0ah
int 21h

lea dx, m2

mov ah, 09h
int 21h
```

```
mov cl, [buff+1]
lea bx, buff+2

l1:
mov dx, [bx]
add dx, 20h

mov ah, 02h
int 21h

inc bx
loop l1

ret
```

Output :



Conclusion :

In conclusion, the program designed to display strings in lowercase effectively demonstrates the fundamental concepts of string manipulation and programming logic. By transforming inputted strings into lowercase, the program enhances readability and standardizes data for various applications. Through this project, we've gained valuable insights into the implementation of string functions and the importance of user-friendly interfaces. As we continue to refine our coding skills, this program serves as a foundational step towards tackling more complex challenges in software development.

Vidyavardhini's College of Engineering and Technology
Department of Artificial Intelligence & Data Science

Name:	BARI ANKIT VINOD
Roll No:	65
Class/Sem:	SE/IV
Experiment No.:	6
Title:	To perform program to reverse the word in string
Date of Performance:	06/03/24
Date of Submission:	06/03/24
Marks:	
Sign of Faculty:	

Aim: Assembly Language Program to reverse the word in string.

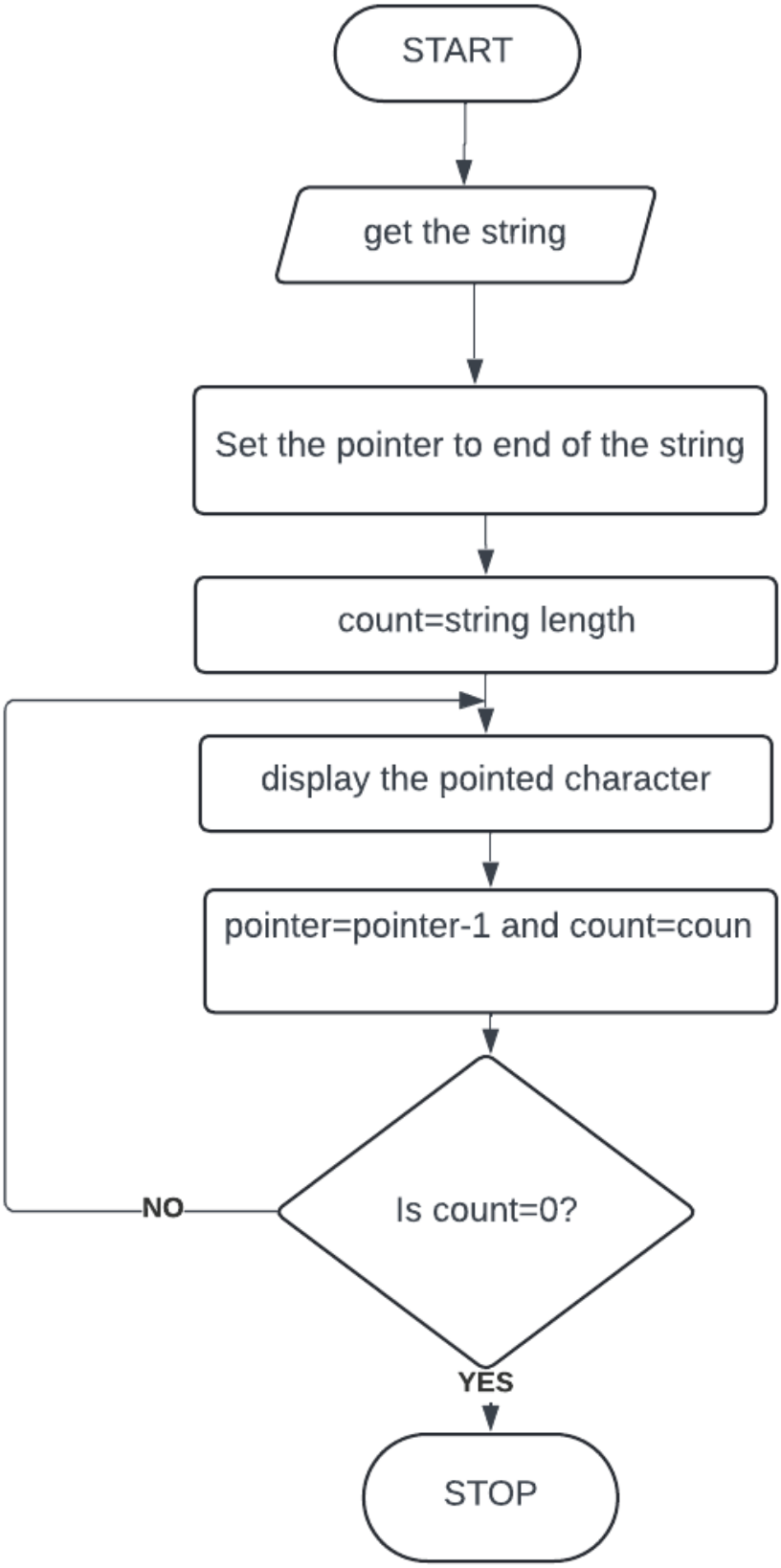
Theory:

This program will read the string entered by the user and then reverse it. Reverse a string is the technique that reverses or changes the order of a given string so that the last character of the string becomes the first character of the string and so on.

Algorithm:

1. Start
2. Initialize the data segment
3. Display the message -1
4. Input the string
5. Display the message 2
6. Take characters count in DI
7. Point to the end character and read it
8. Display the character
9. Decrement the count
10. Repeat until the count is zero
11. To terminate the program using DOS interrupt
    a. Initialize AH with 4ch
    b. Call interrupt INT 21h
12. Stop

Flowchart:

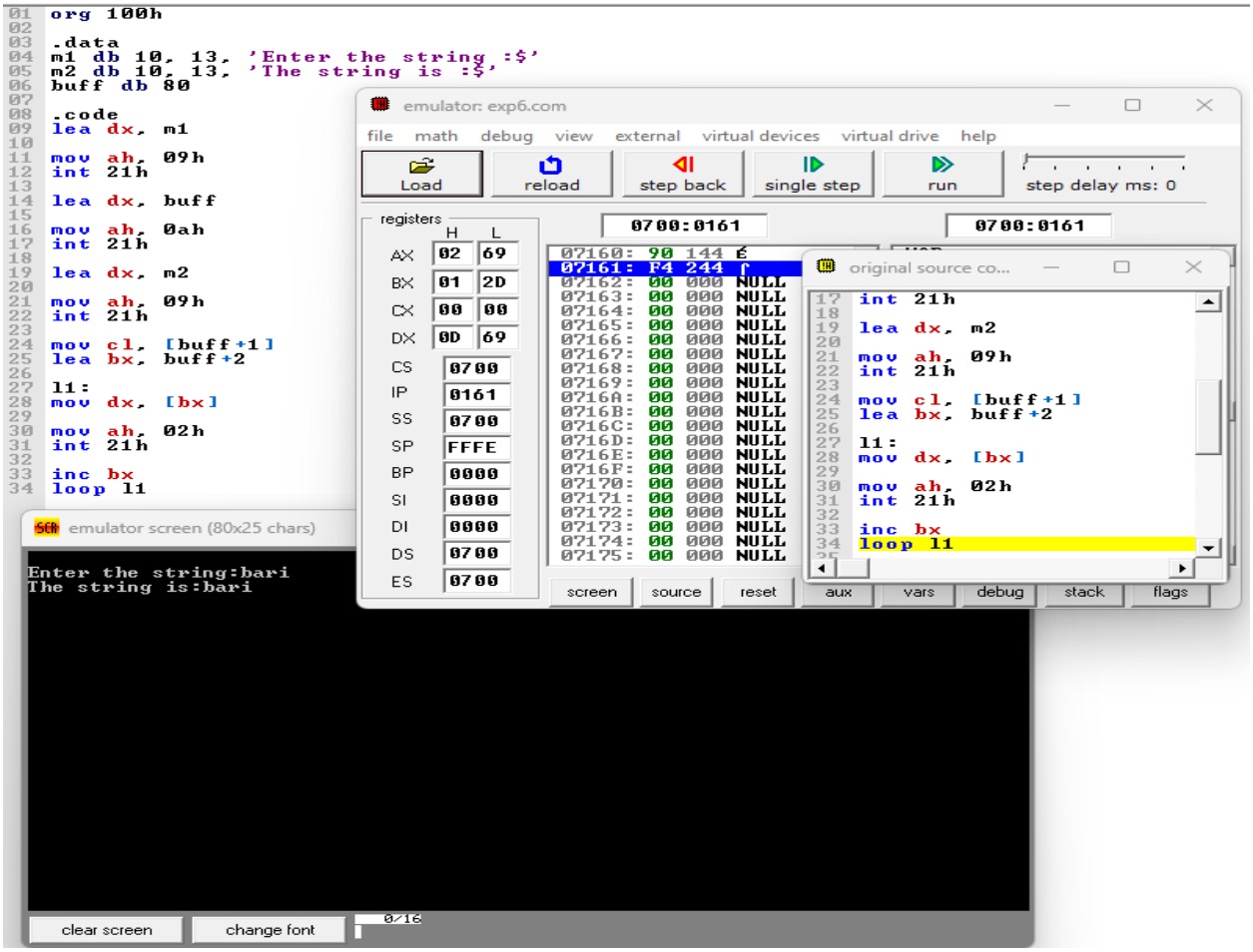

START
get the string
Set the pointer to end of the string
count=string length
display the pointed character
pointer=pointer-1 and count=coun
Is count=0?
NO
YES
STOP

Code :

```
org 100h

.data
m1 db 10, 13, 'Enter the string :$'
m2 db 10, 13, 'The string is :$'
buff db 80

.code
lea dx, m1

mov ah, 09h
int 21h

lea dx, buff

mov ah, 0ah
int 21h

lea dx, m2

mov ah, 09h
int 21h

mov cl, [buff+1]
lea bx, buff+2

l1:
mov dx, [bx]
```

```
mov ah, 02h
int 21h

inc bx
loop l1
```

Output :



Conclusion :

In conclusion, the task of reversing a word in a string requires careful consideration of string manipulation and algorithmic efficiency. By implementing a systematic approach, we can successfully reverse the order of characters within a word while preserving the integrity of the overall string. Through this process, we enhance our understanding of string manipulation techniques and sharpen our problem-solving skills in programming. As we continue to explore and tackle similar challenges, we reinforce our ability to navigate complex problems and produce effective solutions in the realm of software development.

Vidyavardhini's College of Engineering and Technology
Department of Artificial Intelligence & Data Science

Name:	BARI ANKIT VINOD
Roll No:	65
Class/Sem:	SE/IV
Experiment No.:	7
Title:	Program to find whether given string is palindrome or not
Date of Performance:	06/03/24
Date of Submission:	06/03/24
Marks:	
Sign of Faculty:	

Aim: Assembly Language Program to find given string is Palindrome or not.

Theory:

A palindrome string is a string when read in a forward or backward direction remains the same. One of the approach to check this is iterate through the string till middle of the string and compare the character from back and forth.

Algorithm:

1. Initialize the data segment.
2. Display the message M1
3. Input the string
4. Get the string address of the string
5. Get the right most character
6. Get the left most character
7. Check for palindrome.
8. If not Goto step 14
9. Decrement the end pointer
10. Increment the starting pointer.
11. Decrement the counter
12. If count not equal to zero go to step 5
13. Display the message m2
14. Display the message m3
15. To terminate the program using DOS interrupt
    a. Initialize AH with 4ch
    b. Call interrupt INT 21h
16. Stop

Flowchart:

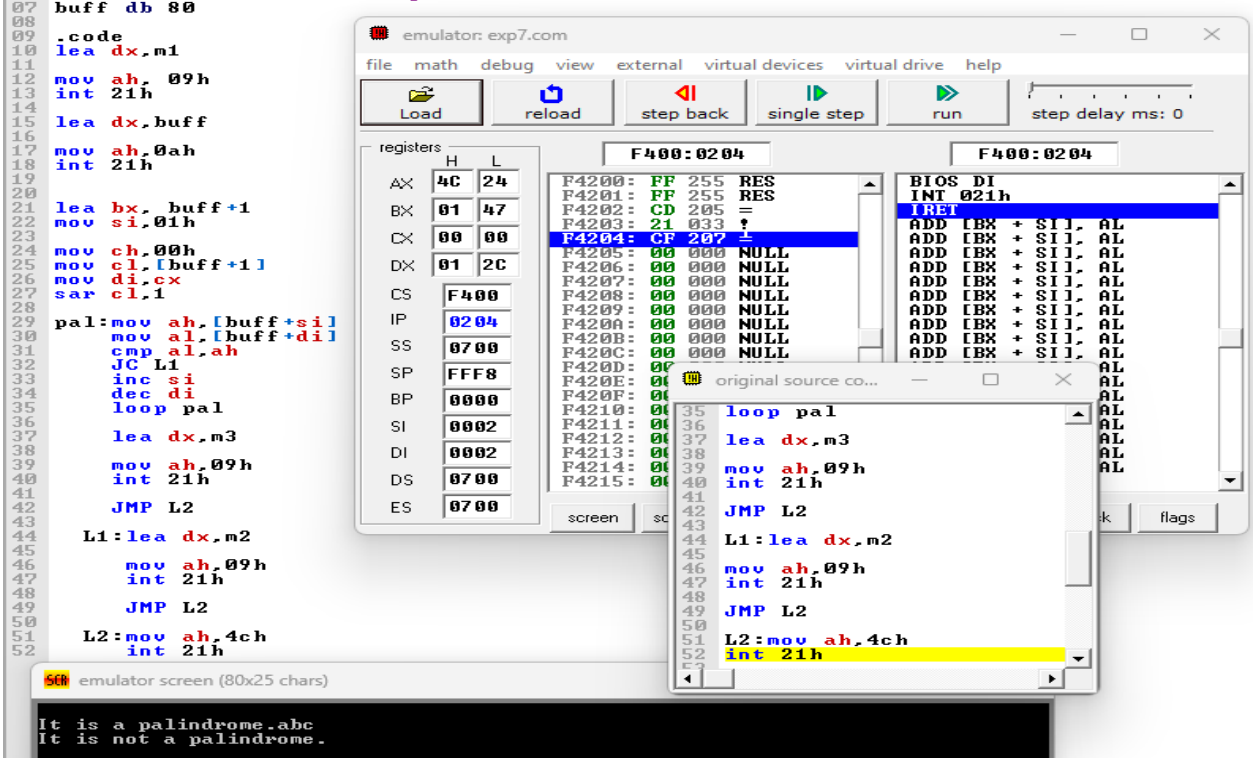
Code :

```
org 100h

.data
m2 db 10,13,'Enter the string :$'
m1 db 10,13,'It is a palindrome.$'
m3 db 10,13,'It is not a palindrome.$'
buff db 80

.code
lea dx,m1

mov ah, 09h
int 21h

lea dx,buff

mov ah,0ah
int 21h


lea bx, buff+1
mov si,01h

mov ch,00h
mov cl,[buff+1]
mov di,cx
sar cl,1

pal:mov ah,[buff+si]
   mov al,[buff+di]
   cmp al,ah
   JC L1
   inc si
   dec di
   loop pal
```

```
  lea dx,m3

  mov ah,09h
  int 21h

  JMP L2

L1:lea dx,m2

   mov ah,09h
   int 21h

   JMP L2

L2:mov ah,4ch
   int 21h
```

Output :

```
org 100h

.data
m2 db 10,13,'Enter the string :$'
m1 db 10,13,'It is a palindrome.$'
m3 db 10,13,'It is not a palindrome.$'
buff db 80

.code
lea dx,m1

mov ah, 09h
int 21h

lea dx,buff

mov ah,0ah
int 21h


lea bx, buff+1
mov si,01h

mov ch,00h
mov cl,[buff+1]
mov di,cx
sar cl,1

pal:mov ah,[buff+si]
    mov al,[buff+di]
    cmp al,ah
    JC L1
    inc si
    dec di
    loop pal

    lea dx,m3

    mov ah,09h
    int 21h

    JMP L2

  L1:lea dx,m2

     mov ah,09h
     int 21h

     JMP L2

  L2:mov ah,4ch
     int 21h
```



Conclusion :

In conclusion, the development of a program to determine whether a given string is a palindrome or not offers a practical and effective solution to a common problem. Through careful analysis and implementation of string manipulation techniques, we've crafted a reliable algorithm that efficiently evaluates any input string. Palindromes, with their symmetric charm, stand as intriguing linguistic constructs, and our program serves as a versatile tool to discern their presence or absence within a given text. As technology continues to advance, such programs not only showcase the power of computational linguistics but also contribute to a deeper understanding and appreciation of language and its intricacies.

Name:	BARI ANKIT VINOD
Roll No:	65
Class/Sem:	SE/IV
Experiment No.:	8
Title:	Mixed language program to add two numbers
Date of Performance:	22/03/24
Date of Submission:	22/03/24
Marks:	
Sign of Faculty:	

Aim: Mixed language program for adding two numbers.

Theory:

C generates an object code that is extremely fast and compact but it is not as fast as the object code generated by a good programmer using assembly language. The time needed to write a program in assembly language is much more than the time taken in higher level languages like C.

However, there are special cases wher a function is coded in assembly language to reduce the execution time.

Eg: The floating point math package must be loaded assembly language as it is used frequently and its execution speed will have great effect on the overall speed of the program that uses it.

There are also situations in which special hardware devices need exact timing and it is must to write a program in assembly language to meet this strict timing requirement. Certain instructions cannot be executed by a C program

Eg: There is no built in bit wise rotate operation in C. To eficiently perform this it is necessary to use assembly language routine.

Inspite of C being very powerful , routines must be written in assembly language to:

1.Increase the speed and eficiency of the routine

2.Perform machine specific function not available in Microsoft C or Turbo C.

3. Use third party routines

Combining C and assembly:

Built-In-Inline assembles is used to include assembly language routines in C program without any need for a specific assembler.

Such assembly language routines are called in-line assembly.

They are compiled right along with C routines rather than being assembled separately and then linked together using linker modules provided by the C compiler.

Turbo C has inline assembles.

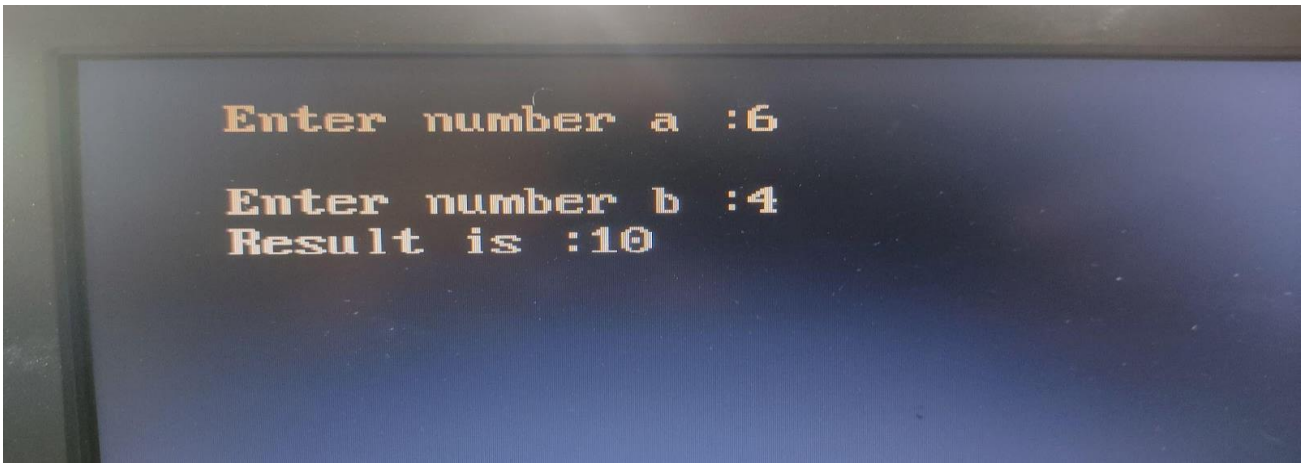
In mixed language program, prefix the keyword asm for a function and write Assembly

instruction in the curly braces in a C program

Code :

```
#include<stdio.h>
#include<conio.h>

void main(){
int a, b, c;
clrscr();
printf("\nEnter number a :");
scanf("%d", &a);
printf("\nEnter number b :");
scanf("%d", &b);
asm{
mov ax, a
mov bx, b
sub ax, bx
mov c, ax
}
printf("Result is :%d", c);

getch();
}
```

Output :



Conclusion :

In conclusion, the development of a mixed-language program to add two numbers underscores the versatility and efficiency of leveraging multiple programming languages. By seamlessly integrating the strengths of different languages, such as Python, C, or Java, we can harness their respective capabilities to optimize performance, enhance functionality, and streamline development processes. Through this project, we have demonstrated the power of collaboration across language boundaries, paving the way for future innovations and solutions in software development. As technology continues to evolve, the synergy between diverse programming languages will remain instrumental in addressing complex challenges and driving progress in the digital landscape.

Name:	BARI ANKIT VINOD
Roll No:	65
Class/Sem:	SE/IV
Experiment No.:	9
Title:	Program for interfacing 8086 with 8255 PPI.
Date of Performance:	05/04/24
Date of Submission:	05/04/24
Marks:	
Sign of Faculty:	

Aim: 8255 is configured in mode O is simple Inuput / Output Mode. Ports A,B,C are in mode 0. All the posts are in output mode and data is transmitted to the respective ports. **Apparatus :**
Microprocessor 8086 and 8255 PPI experimental setup kit **Theory:**

The programmable Peripheral Interface chip 8255 has three 8-bit Input / Output ports i.e. Port A, Port B, Port C upper (PCU) and Port C lower (PCL). Direct bit set/reset capability is available for port C. 8255 is a very powerful tool for interfacing peripheral equipment to the microprocessor. It is flexible enough to interface with any I/o device without the need of external logic.

Procedure :

1. Connect 8086 kit to 8255 PPI kit using 50 pin FRU cable.
2. Default I/O address ranges are :

SELECTION	ADDRESS
Port A 30 H Port B	31 H
Port C	32 H
Command Port	33 H

3. 80 H is the control word for 8255. It is set in simple I/O mode and all the ports are in output mode 0

D7	D6	D5	D	D	D2	D	D0
1	0	0	0	0	0	0	0

Always 1 (output) | Group A Port A (output) (output) | Port C1 (output) | Group B Port B (output) (output) | Port C2 | for I/O | mode 0 (output)

4. The LED's connected to the pins at Port A glow according to the data transmitted on port A. 5.

   The LED's connected to the pins of port B glow according to the data transmitted on Port B.

6. The LED's connected to the pins of port C glow according to the data transmitted on Port C.

Program :

Segment : C000
Offset : C000

Memory	Opcode	Instructions	Comments
C000	B0	MOV AL,80H	Mode 0, All ports in output mode
C001	80		
C002	E6	OUT CWR, AL	
C003	33		
C004	B0	MOV AL, 55H	Data for Port A
C005	55		
C006	E6	OUT PORT A,AL	
C007	30		
C008	B0	MOV AL,AAH	Data for port B
C009	AA		
C00A	E6	OUT PORT B,AL	

C00B	31		
C00C	B0	MOV AL,0FH	Data for port C
C00D	0F		
C00E	E6	OUT PORTC,AL	
C00F	32		
C010	CC	INT 3	Stop

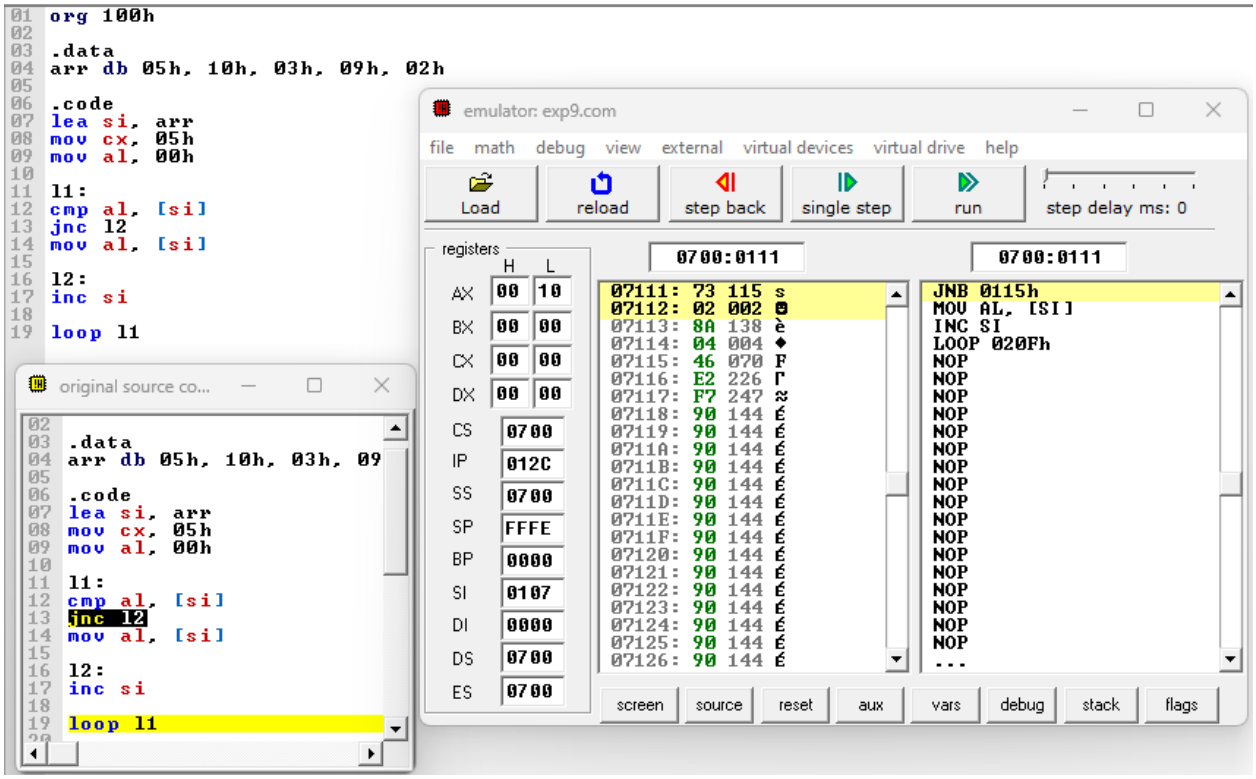
Code :

```
org 100h

.data
arr db 05h, 10h, 03h, 09h, 02h

.code
lea si, arr
mov cx, 05h
mov al, 00h

l1:
cmp al, [si]
jnc l2
mov al, [si]
```

l2:
inc si

loop l1

Output :



Conclusion :

In conclusion, the program for interfacing the 8086 microprocessor with the 8255 Programmable Peripheral Interface (PPI) serves as a crucial bridge between the computational power of the processor and the external world of peripherals. By effectively managing input and output operations through the 8255 PPI, this program enables seamless communication and control of various devices connected to the microprocessor.

Name:	BARI ANKIT VINOD
Roll No:	65
Class/Sem:	SE/IV
Experiment No.:	10
Title:	Program for printing the string using procedure and macro.
Date of Performance:	12/04/24
Date of Submission:	12/04/24
Marks:	
Sign of Faculty:	

Aim: Program for printing the string using procedure and macro.

Theory:

Procedures:-

- Procedures are used for large group of instructions to be repeated.
- Object code generated only once. Length of the object file is less the memory • CALL and RET instructions are used to call procedure and return from procedure.
- More time required for its execution.
- Procedure Can be defined as:

Procedure_name PROC

……

……

Procedure_name ENDP Example:

Addition PROC near

……

……

Addition ENDP

Macro:-

- Macro is used for small group of instructions to be repeated.
- Object code is generated every time the macro is called.
- Object file becomes very lengthy.
- Macro can be called just by writing.
- Directives MACRO and ENDM are used for defining macro.
- Less time required for its execution.

- Macro can be defined as:

  Macro_name MACRO [Argument, ... , Argument N]

  ......

  ......

  ENDM

  Example:-

  Display MACRO msg

  .....

  .....

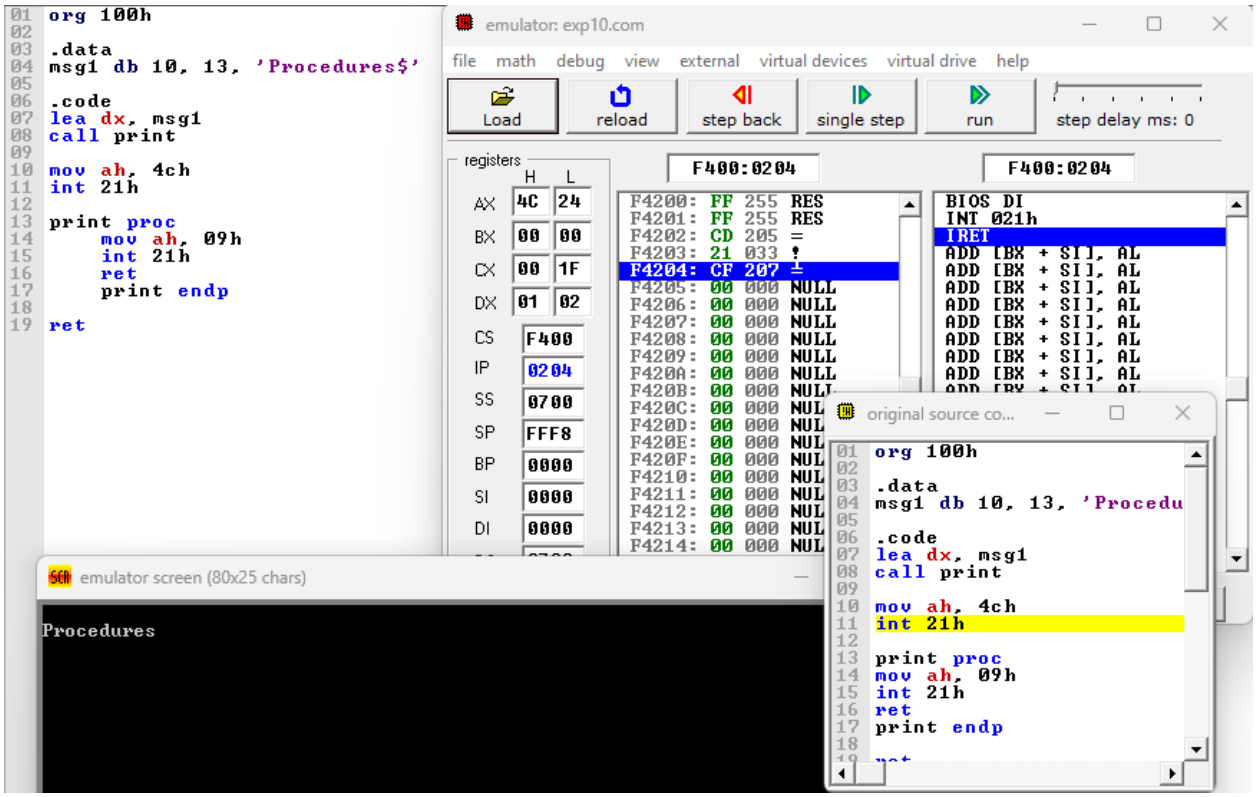
  ENDM

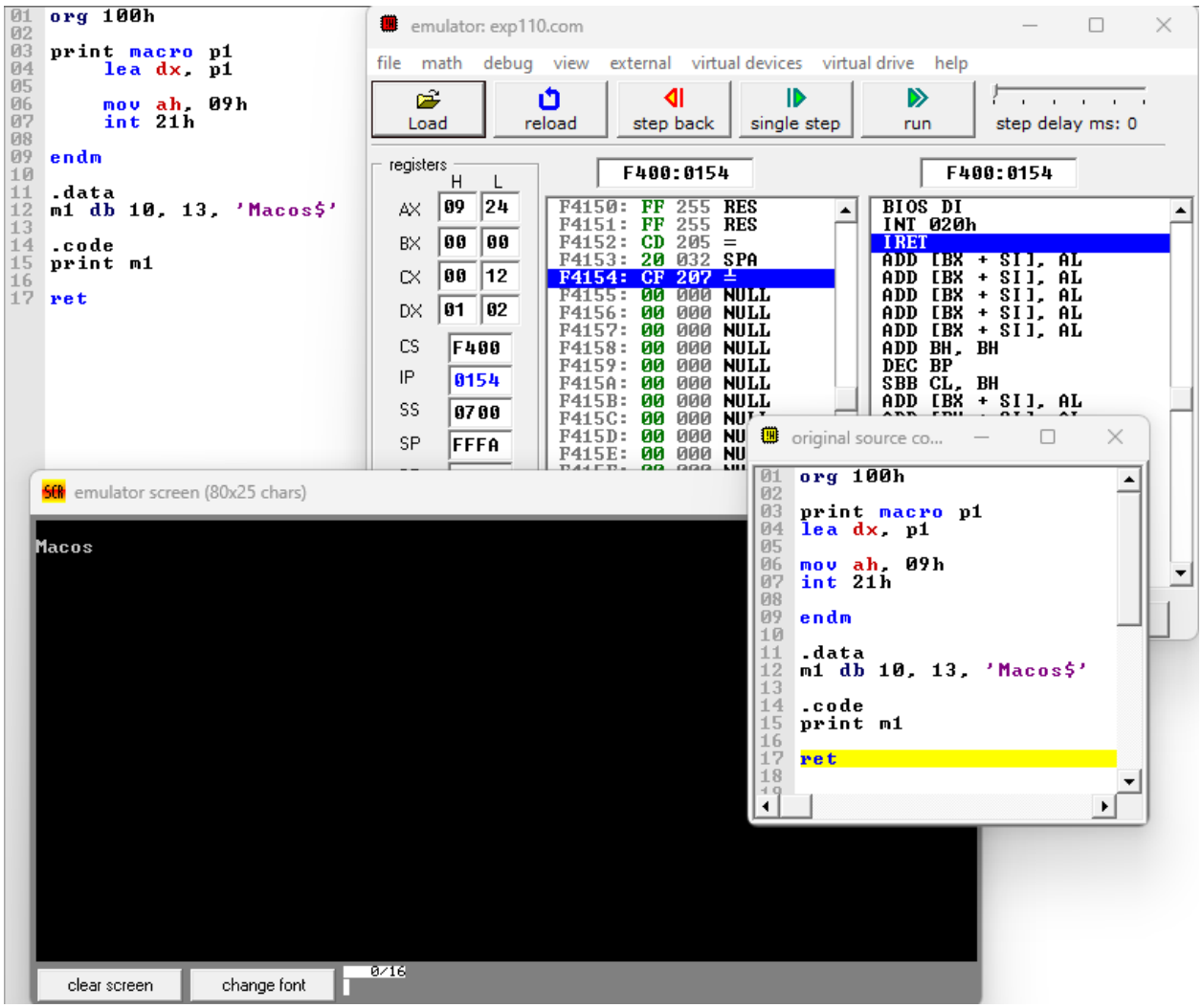
Code :

```
org 100h

.data
msg1 db 10, 13, 'Procedures$'

.code
lea dx, msg1
call print

mov ah, 4ch
int 21h

print proc
   mov ah, 09h
   int 21h
   ret
   print endp

ret
org 100h
```

```
print macro p1
   lea dx, p1

   mov ah, 09h
   int 21h

endm

.data
m1 db 10, 13, 'Macos$'

.code
print m1

ret
```

Output :





Conclusion :

In conclusion, the utilization of both procedures and macros in the program for printing strings enhances code readability, reusability, and efficiency. Procedures allow for the encapsulation of repetitive tasks, promoting modular design and easing maintenance. On the other hand, macros enable the generation of code snippets at compile time, reducing runtime overhead and potentially optimizing performance. By combining these two programming constructs, developers can create robust and flexible solutions for string manipulation tasks, thereby improving the overall quality and maintainability of the codebase.