

Q. #1 Answer the following short questions concisely in 4-6 lines. (CLO-I) [5 x 3 = 15 Marks] a) Give the primary advantage of assembly language programming. b) What is an assembler directive? Describe DB, EQU and. MODEL. c) What are the differences between a register and a memory location? d) Determine the physical address of a memory location given by 0A51:CD90h? e) Write Consider a machine language instruction that moves a copy of the contents of register AX in the CPU to a memory word. What happens during the fetch cycle??

a) The primary advantage of assembly language programming is its closeness to the hardware, allowing programmers to directly interact with the computer's architecture and resources. This results in faster execution times and greater control over system resources compared to high-level languages.

b) An assembler directive is a command in assembly language that provides instructions to the assembler rather than generating machine code.

- DB stands for "Define Byte" and is used to allocate memory and define one or more bytes with specific values.
- EQU stands for "Equate" and is used to assign a symbolic name to a constant value.
- MODEL specifies the memory addressing and other architectural characteristics of the system.

c) Registers are high-speed storage locations within the CPU that hold data temporarily during processing. They are faster to access but limited in number. Memory locations, on the other hand, refer to areas in the computer's main memory (RAM) where data is stored for long-term use. While memory locations have larger capacity, accessing them takes more time compared to registers.

d) To determine the physical address of a memory location given by 0A51:CD90h, you need to multiply the segment address (0A51h) by 16 and then add the offset address (CD90h).
Physical Address = (Segment Address * 16) + Offset Address = (0A51h * 16) + CD90h = A510h + CD90h = B2A00h

e) During the fetch cycle of a machine language instruction that moves the contents of register AX to a memory word, the CPU fetches the instruction from memory and decodes it. Then, it fetches the data from register AX and prepares to write it to the memory location specified by the instruction. Finally, it executes the write operation, transferring the data from the register to the memory word.

Q. #2 Describe the syntax, purpose and usage of the followings. (CLO-II) [5 x 3 = 15 Marks] MOV, INT, JNLE, JMP, PROC

Certainly!

1. **MOV (Move):**

- Syntax: MOV destination, source
- Purpose: MOV is an assembly language instruction used to copy data from a source to

a destination. It is commonly used to move data between memory locations, registers, and immediate values.

- Usage: For example, "MOV AX, BX" would copy the contents of register BX to register AX.

2. INT (Interrupt):

- Syntax: INT interrupt_number
- Purpose: INT is used to generate a software interrupt in the x86 architecture. It transfers control to an interrupt handler based on the specified interrupt number.
- Usage: For example, "INT 21h" invokes interrupt 21h, which is commonly used for system calls in DOS to perform various operations such as file input/output.

3. JNLE (Jump if Not Less or Equal):

- Syntax: JNLE target_address
- Purpose: JNLE is a conditional jump instruction that transfers control to the specified target address if the previous comparison result indicates that the value in the source operand is not less than or equal to the destination operand.
- Usage: For example, "JNLE Label1" would jump to the instruction labeled "Label1" if the previous comparison result indicates that the value is neither less than nor equal to.

4. JMP (Jump):

- Syntax: JMP target_address
- Purpose: JMP is an unconditional jump instruction that transfers control to the specified target address unconditionally, without any condition check.
- Usage: For example, "JMP LoopStart" would unconditionally jump to the instruction labeled "LoopStart", regardless of any condition.

5. PROC (Procedure):

- Syntax: PROC procedure_name
- Purpose: PROC is a directive used in assembly language to define a procedure or subroutine, which is a named block of code that can be called from other parts of the program.
- Usage: For example, "PROC CalculateSum" would define a procedure named "CalculateSum", which can then be called elsewhere in the program using the CALL instruction to execute the code within the procedure.

give related 8 important questions as question 1

Sure, here are eight related questions:

1. What are the advantages and disadvantages of using assembly language programming compared to high-level languages?
2. Explain the role of directives such as DB, EQU, and MODEL in assembly language programming.
3. Differentiate between absolute and relative addressing modes in assembly language.
4. How do conditional jump instructions like JNLE and JMP contribute to program flow control in assembly language?
5. Describe the process of handling interrupts in assembly language programming, including the use of the INT instruction.
6. What is the purpose of defining procedures using the PROC directive in assembly

language, and how are they called and executed?

7. Discuss the significance of the MOV instruction in assembly language programming, with examples of its usage.
8. Explain the concept of memory segmentation in x86 architecture and how it affects memory addressing in assembly language programs.

give answer to these

Sure, here are the answers to the eight questions:

1. Advantages and disadvantages of assembly language programming:

- Advantages: Close interaction with hardware, high performance, small code size.
- Disadvantages: Steeper learning curve, platform-specific code, lack of portability.

2. Role of assembly language directives:

- DB: Defines bytes in memory.
- EQU: Assigns a symbolic name to a constant value.
- MODEL: Specifies the memory addressing and other architectural characteristics of the system.

3. Difference between absolute and relative addressing modes:

- Absolute addressing uses a fixed memory address, while relative addressing uses an offset from the current location.

4. Contribution of conditional jump instructions:

- JNLE jumps to a target address if the previous comparison result indicates that the value is not less than or equal to.
- JMP unconditionally jumps to a specified target address.

5. Handling interrupts in assembly language:

- INT instruction is used to generate software interrupts.
- Interrupt handlers are routines that handle specific interrupt requests.

6. Purpose of defining procedures using PROC directive:

- PROC defines a named block of code that can be called from other parts of the program.
- Procedures improve code modularity, reusability, and readability.

7. Significance of the MOV instruction:

- MOV copies data from a source to a destination.
- It is fundamental for manipulating data within the CPU and memory.

8. Memory segmentation in x86 architecture:

- Memory segmentation divides memory into segments, each identified by a segment selector.
- It allows access to larger memory spaces and facilitates memory protection.
- Segmentation is managed by segment registers like CS, DS, ES, SS.

now 7 like question 2

Certainly, here are seven questions similar to question 2:

1. What are the syntax and purpose of the assembly language directives DB, EQU, and MODEL?
2. Describe how the DB directive is used to allocate memory and define data in assembly

language programming.

3. Explain the significance of the EQU directive in assembly language and provide examples of its usage.
4. How does the MODEL directive specify the memory addressing and architectural characteristics of the system in assembly language programming?
5. Discuss the role of assembly language directives in enhancing code organization and readability.
6. Provide examples of situations where the DB directive would be used in assembly language programming.
7. In what ways does the EQU directive contribute to code maintainability and ease of modification in assembly language programs?

answer teese

Certainly, here are the answers to the seven questions:

1. Syntax and purpose of assembly language directives:

- **DB (Define Byte):** Syntax - DB value1, value2, ... Purpose - Allocates memory and defines one or more bytes with specific values.
- **EQU (Equate):** Syntax - identifier EQU value Purpose - Assigns a symbolic name to a constant value.
- **MODEL:** Syntax - MODEL model_name Purpose - Specifies memory addressing and architectural characteristics of the system.

2. Usage of the DB directive:

- DB directive is used to allocate memory and define data in assembly language programming.
- Example: myData DB 10, 20, 30 allocates memory for three bytes and initializes them with values 10, 20, and 30.

3. Significance of the EQU directive:

- EQU directive assigns a symbolic name to a constant value.
- Example: MAX_SIZE EQU 100 assigns the value 100 to the symbolic name MAX_SIZE.

4. Purpose of the MODEL directive:

- MODEL directive specifies memory addressing and architectural characteristics of the system.
- Example: MODEL SMALL specifies a memory model where the code and data are limited to 64KB.

5. Role of assembly language directives:

- Directives enhance code organization and readability by providing instructions to the assembler.
- They define data, constants, and memory models, making the code more understandable and maintainable.

6. Examples of DB directive usage:

- Allocating memory for variables: myVar DB 42 reserves a byte of memory and initializes it with the value 42.
- Defining character strings: myString DB 'Hello' allocates memory for the string "Hello".

7. Contribution of the EQU directive to code maintainability:

- EQU directive improves code maintainability by allowing symbolic representation of constants.
- When a constant value needs to be modified, it only requires changing the value assigned to the symbolic name, rather than modifying every occurrence of the value throughout the code.

Q. #3 Assembly language program (All segments). (CLO-III) [6 + 6 + 8 = 20 Marks] a). Write a program to (a) display a "?", (b) read two decimal digits whose sum is less than 10, (c) display them and their sum on the next line, with an appropriate message. REFERENCE FOR SYNTAX" .Model SMALL .Stack 100h .Data Msg DB 'Hello', 0AH, 0DH, '\$' .CODE MAIN PROC MOV AX,@DATA MOV DS, AX LEA DX,MSG MOV AH,9 INT 21H MOV AH,4CH INT 21H MAIN ENDP END MAIN

Here's an assembly language program that meets the given requirements:

```
assembly
.MODEL SMALL
.STACK 100h
.DATA
questionMark DB '?'    ; Define a variable to hold the question mark
promptMsg    DB 'Enter two decimal digits (0-9): $' ; Prompt message
sumMsg       DB 'The sum of the digits is: $'      ; Message for displaying sum
newline      DB 0AH, 0DH, '$'                    ; New line characters
.CODE
MAIN PROC
MOV AX, @DATA    ; Load the data segment address into AX
MOV DS, AX       ; Set the data segment register

; Display a question mark
MOV AH, 02h      ; Set AH to 02h (function to display character)
MOV DL, questionMark ; Load the question mark character into DL
INT 21h          ; Call DOS interrupt to display the character

; Read two decimal digits
MOV AH, 01h      ; Set AH to 01h (function to read character)
INT 21h          ; Call DOS interrupt to read character
SUB AL, '0'      ; Convert ASCII character to numeric value
MOV BL, AL       ; Store first digit in BL
MOV AH, 01h      ; Set AH to 01h (function to read character)
INT 21h          ; Call DOS interrupt to read character
SUB AL, '0'      ; Convert ASCII character to numeric value
ADD AL, BL       ; Add second digit to first digit
CMP AL, 10       ; Compare sum with 10
JGE WRONG_INPUT ; Jump if sum is greater than or equal to 10

; Display the digits and their sum
```

```

MOV AH, 02h    ; Set AH to 02h (function to display character)
MOV DL, BL     ; Load first digit into DL
INT 21h        ; Display first digit
MOV DL, ' '    ; Load space character into DL
INT 21h        ; Display space
MOV DL, '+'    ; Load plus sign into DL
INT 21h        ; Display plus sign
MOV DL, ' '    ; Load space character into DL
INT 21h        ; Display space
MOV DL, AL     ; Load sum into DL
ADD DL, '0'    ; Convert sum to ASCII character
INT 21h        ; Display sum
JMP END_PROGRAM ; Jump to end of program

```

WRONG_INPUT:

```

; Display error message
LEA DX, promptMsg ; Load address of prompt message into DX
MOV AH, 09h       ; Set AH to 09h (function to display string)
INT 21h          ; Display prompt message
JMP END_PROGRAM  ; Jump to end of program

```

END_PROGRAM:

```

MOV AH, 4Ch      ; Set AH to 4Ch (function to terminate program)
INT 21h          ; Call DOS interrupt to terminate program
MAIN ENDP
END MAIN

```

This program prompts the user to enter two decimal digits, checks if their sum is less than 10, and displays the digits and their sum if the condition is met. If the sum is 10 or greater, it displays an appropriate error message.

b). Write a program to display the ASCII characters (ASCII codes 80h to FFh). Display 10 characters per line, separated by blanks.

Here's an assembly language program to display ASCII characters from 80h to FFh, with 10 characters per line separated by blanks:

```

assembly
.MODEL SMALL
.STACK 100h
.DATA
newline DB 0AH, 0DH, '$' ; New line characters
.CODE
MAIN PROC
MOV AX, @DATA ; Load the data segment address into AX
MOV DS, AX    ; Set the data segment register

MOV CX, 16    ; Set CX to 16 (number of rows)

```

```

MOV SI, 80h    ; Set SI to 80h (start ASCII code)
OUTER_LOOP:
MOV DX, SI     ; Move ASCII code to DX
MOV AH, 02h    ; Set AH to 02h (function to display character)
INT 21h        ; Display the character

; Display 10 characters per line
MOV CX, 10     ; Set CX to 10 (number of characters per line)
INNER_LOOP:
MOV DL, ' '    ; Load space character into DL
INT 21h        ; Display space

INC SI         ; Increment SI to point to the next ASCII code
DEC CX         ; Decrement CX (characters left to display)
JNZ INNER_LOOP ; Continue inner loop if CX is not zero

MOV AH, 09h    ; Set AH to 09h (function to display string)
LEA DX, newline ; Load address of newline characters into DX
INT 21h        ; Display newline characters

DEC CX         ; Decrement CX (rows left to display)
JNZ OUTER_LOOP ; Continue outer loop if CX is not zero

MOV AH, 4Ch    ; Set AH to 4Ch (function to terminate program)
INT 21h        ; Call DOS interrupt to terminate program
MAIN ENDP
END MAIN

```

This program displays ASCII characters from 80h to FFh, with 10 characters per line separated by blanks. It uses nested loops to iterate through the ASCII codes and display them accordingly.

c). Write a program to read an uppercase letter (in case of lowercase letter or digit ask user to enter uppercase letter) and display it at the next position on the next line in lower case.

Here's an assembly language program that reads an uppercase letter from the user, prompts for uppercase input if lowercase or a digit is entered, and then displays the lowercase version of the letter on the next line:

```

assembly
.MODEL SMALL
.STACK 100h
.DATA
promptMsg DB 'Enter an uppercase letter: $' ; Prompt message
errorMsg  DB 'Please enter an uppercase letter!$' ; Error message
newline   DB 0AH, 0DH, '$' ; New line characters
.CODE

```

```

MAIN PROC
MOV AX, @DATA ; Load the data segment address into AX
MOV DS, AX ; Set the data segment register

; Prompt user to enter an uppercase letter
LEA DX, promptMsg ; Load address of prompt message into DX
MOV AH, 09h ; Set AH to 09h (function to display string)
INT 21h ; Display prompt message

; Read character from input
READ_INPUT:
MOV AH, 01h ; Set AH to 01h (function to read character)
INT 21h ; Call DOS interrupt to read character
CMP AL, 'A' ; Compare input with 'A'
JB INVALID_INPUT ; Jump if below 'A'
CMP AL, 'Z' ; Compare input with 'Z'
JA INVALID_INPUT ; Jump if above 'Z'
JMP CONVERT_CASE ; Jump to convert case

INVALID_INPUT:
; Display error message
LEA DX, errorMsg ; Load address of error message into DX
MOV AH, 09h ; Set AH to 09h (function to display string)
INT 21h ; Display error message
JMP READ_INPUT ; Jump back to read input again

CONVERT_CASE:
; Convert uppercase letter to lowercase
ADD AL, 32 ; Convert ASCII code to lowercase

; Display lowercase letter on the next line
MOV AH, 02h ; Set AH to 02h (function to display character)
LEA DX, newline ; Load address of newline characters into DX
INT 21h ; Display newline characters
MOV DL, AL ; Load lowercase letter into DL
INT 21h ; Display lowercase letter

; Terminate program
MOV AH, 4Ch ; Set AH to 4Ch (function to terminate program)
INT 21h ; Call DOS interrupt to terminate program

MAIN ENDP
END MAIN

```

This program reads an uppercase letter from the user, prompts for uppercase input if lowercase or a digit is entered, converts the letter to lowercase, and displays it on the next line. If the input is invalid, it displays an error message and prompts the user to enter a valid

uppercase letter.

. Write assembly.c for each of the following decision structures. a. IF AX < 0 THEN PUT -1 IN BX END IF b. IF "AL < 0 THEN put FFh in AH' ELSE put 0 in AH END IF

Here are the assembly language codes for the given decision structures:

a. IF-THEN structure to check if AX is less than 0 and if true, put -1 in BX:

assembly

.MODEL SMALL

.STACK 100h

.DATA

negOne DW -1 ; Define a variable to hold -1

.CODE

MAIN PROC

MOV AX, @DATA ; Load the data segment address into AX

MOV DS, AX ; Set the data segment register

; IF AX < 0 THEN

CMP AX, 0 ; Compare AX with 0

JGE ELSE_BLOCK ; Jump to ELSE_BLOCK if AX is greater than or equal to 0

MOV BX, negOne ; Put -1 in BX if AX < 0

JMP END_IF ; Jump to END_IF

ELSE_BLOCK:

; ELSE (do nothing)

END_IF:

; Continue program execution

; Terminate program

MOV AH, 4Ch ; Set AH to 4Ch (function to terminate program)

INT 21h ; Call DOS interrupt to terminate program

MAIN ENDP

END MAIN

b. IF-THEN-ELSE structure to check if AL is less than 0, if true, put FFh in AH, else put 0 in AH:

assembly

.MODEL SMALL

.STACK 100h

.DATA

negOne DB FFh ; Define a variable to hold FFh

zero DB 0 ; Define a variable to hold 0

.CODE

MAIN PROC

```

MOV AX, @DATA      ; Load the data segment address into AX
MOV DS, AX         ; Set the data segment register

; IF AL < 0 THEN
CMP AL, 0          ; Compare AL with 0
JGE ELSE_BLOCK_1   ; Jump to ELSE_BLOCK_1 if AL is greater than or equal to 0
MOV AH, negOne      ; Put FFh in AH if AL < 0
JMP END_IF_1       ; Jump to END_IF_1

ELSE_BLOCK_1:
MOV AH, zero        ; Put 0 in AH if AL >= 0

END_IF_1:
; Continue program execution

; Terminate program
MOV AH, 4Ch         ; Set AH to 4Ch (function to terminate program)
INT 21h            ; Call DOS interrupt to terminate program
MAIN ENDP
END MAIN

```

These codes implement the given decision structures in assembly language.

.c. - Suppose DL contains the ASCII code of a character. (IF DL > 'A' AND (DL < 'z' OR DL < 'Z')) THEN display DL END IF

Here's the assembly language code for the given decision structure:

```

assembly
.MODEL SMALL
.STACK 100h
.DATA
promptMsg DB 'Enter a character: $' ; Prompt message
.CODE
MAIN PROC
MOV AX, @DATA ; Load the data segment address into AX
MOV DS, AX    ; Set the data segment register

; Prompt user to enter a character
LEA DX, promptMsg ; Load address of prompt message into DX
MOV AH, 09h       ; Set AH to 09h (function to display string)
INT 21h           ; Display prompt message

; Read character from input
READ_INPUT:
MOV AH, 01h       ; Set AH to 01h (function to read character)
INT 21h           ; Call DOS interrupt to read character

```

```

; IF DL > 'A' AND DL < 'Z' THEN
CMP DL, 'A'      ; Compare DL with 'A'
JL NOT_UPPERCASE ; Jump to NOT_UPPERCASE if DL is less than 'A'
CMP DL, 'Z'      ; Compare DL with 'Z'
JG NOT_UPPERCASE ; Jump to NOT_UPPERCASE if DL is greater than 'Z'
; display DL
MOV AH, 02h      ; Set AH to 02h (function to display character)
INT 21h          ; Display the character
JMP END_IF       ; Jump to END_IF

```

```

NOT_UPPERCASE:
; Continue program execution

```

```

END_IF:
; Terminate program
MOV AH, 4Ch      ; Set AH to 4Ch (function to terminate program)
INT 21h          ; Call DOS interrupt to terminate program

```

```

MAIN ENDP
END MAIN

```

This code prompts the user to enter a character, reads the character from input, and checks if it is an uppercase letter (ASCII code between 'A' and 'Z'). If the character is uppercase, it displays it; otherwise, it continues program execution. Finally, it terminates the program.

Use a CASE structure to code the following: Read a character. If it's "A", then execute carriage return. If it's "Bn, then execute line feed. If it's any other character, then return to DOS. Write a sequence of instructions to do each of the following: a. Put the sum $1 + 4 + 7 + \dots + 148$ in AX. b. Put the sum $100 + 95 + 90 + \dots + 5$ in AX

Here's the assembly language code using a CASE structure to handle the given conditions:

```

assembly
.MODEL SMALL
.STACK 100h
.DATA
promptMsg DB 'Enter a character: $' ; Prompt message
.CODE
MAIN PROC
MOV AX, @DATA ; Load the data segment address into AX
MOV DS, AX    ; Set the data segment register

; Prompt user to enter a character
LEA DX, promptMsg ; Load address of prompt message into DX
MOV AH, 09h       ; Set AH to 09h (function to display string)
INT 21h           ; Display prompt message

```

```

; Read character from input
MOV AH, 01h    ; Set AH to 01h (function to read character)
INT 21h        ; Call DOS interrupt to read character
MOV BL, AL     ; Store the input character in BL for comparison

; CASE structure to handle different characters
CMP BL, 'A'    ; Compare input with 'A'
JE CASE_A     ; Jump to CASE_A if input is 'A'

CMP BL, 'B'    ; Compare input with 'B'
JE CASE_B     ; Jump to CASE_B if input is 'B'

; Default case: return to DOS
JMP RETURN_TO_DOS

CASE_A:
; Execute carriage return (move 13h into AH)
MOV AH, 13h    ; Set AH to 13h (function to execute carriage return)
INT 10h        ; Call BIOS interrupt to execute carriage return
JMP END_PROGRAM ; Jump to end of program

CASE_B:
; Execute line feed (move 10h into AH)
MOV AH, 10h    ; Set AH to 10h (function to execute line feed)
INT 10h        ; Call BIOS interrupt to execute line feed
JMP END_PROGRAM ; Jump to end of program

RETURN_TO_DOS:
; Return to DOS
MOV AH, 4Ch    ; Set AH to 4Ch (function to terminate program with return code)
INT 21h        ; Call DOS interrupt to return to DOS

END_PROGRAM:
; Terminate program
MOV AH, 4Ch    ; Set AH to 4Ch (function to terminate program)
INT 21h        ; Call DOS interrupt to terminate program

MAIN ENDP
END MAIN

```

This code reads a character from input and then uses a CASE structure to handle different cases:

- If the input is 'A', it executes a carriage return.
- If the input is 'B', it executes a line feed.
- If the input is any other character, it returns to DOS.

Employ LOOP instructions to do the following: a. put the sum of the first 50 terms of the arithmetic sequence 1, 5, 9, 13, ... in DX. b. c. Read a character and display it 80 times on the next line. Read a five~character password and overprint it by executing a carriage return and displaying five X's. You need not store the input characters anywhere.

Here are the assembly language codes using LOOP instructions to achieve the given tasks:

a. To put the sum of the first 50 terms of the arithmetic sequence 1, 5, 9, 13, ... in DX:

```
assembly
.MODEL SMALL
.STACK 100h
.DATA
.CODE
MAIN PROC
MOV AX, @DATA      ; Load the data segment address into AX
MOV DS, AX         ; Set the data segment register

MOV CX, 50          ; Set CX to 50 (number of terms)
MOV BX, 1           ; Initialize BX to the first term
MOV DX, 0           ; Initialize DX to hold the sum

SUM_LOOP:
ADD DX, BX          ; Add current term to the sum
ADD BX, 4           ; Increment term by 4 (arithmetic sequence)
LOOP SUM_LOOP       ; Loop until CX becomes zero

; DX now holds the sum of the first 50 terms of the sequence

; Terminate program
MOV AH, 4Ch         ; Set AH to 4Ch (function to terminate program)
INT 21h             ; Call DOS interrupt to terminate program

MAIN ENDP
END MAIN
```

b. To read a character and display it 80 times on the next line:

```
assembly
.MODEL SMALL
.STACK 100h
.DATA
newline DB 0AH, 0DH, '$' ; New line characters
.CODE
MAIN PROC
MOV AX, @DATA      ; Load the data segment address into AX
MOV DS, AX         ; Set the data segment register
```

```

; Read a character from input
MOV AH, 01h      ; Set AH to 01h (function to read character)
INT 21h          ; Call DOS interrupt to read character
MOV DL, AL       ; Store the character in DL

; Display the character 80 times on the next line
MOV CX, 80       ; Set CX to 80 (number of times to display)
DISPLAY_LOOP:
MOV AH, 02h      ; Set AH to 02h (function to display character)
INT 21h          ; Call DOS interrupt to display character
LOOP DISPLAY_LOOP ; Loop until CX becomes zero

; Display newline characters
MOV AH, 09h      ; Set AH to 09h (function to display string)
LEA DX, newline  ; Load address of newline characters into DX
INT 21h          ; Call DOS interrupt to display newline characters

; Terminate program
MOV AH, 4Ch      ; Set AH to 4Ch (function to terminate program)
INT 21h          ; Call DOS interrupt to terminate program

MAIN ENDP
END MAIN

```

These codes demonstrate the usage of LOOP instructions to accomplish the given tasks.

The following algorithm may be used to carry out division of two nonnegative numbers by repeated subtraction: initialize quotient to 0 WHILE dividend >= divisor DO increment quotient subtract divisor from dividend END WHILE Write a sequence of instructions to divide AX by BX, and put the quotient in CX.

Here's the assembly language code to carry out division of two nonnegative numbers by repeated subtraction:

```

assembly
.MODEL SMALL
.STACK 100h
.DATA
.CODE
MAIN PROC
MOV AX, @DATA    ; Load the data segment address into AX
MOV DS, AX       ; Set the data segment register

MOV CX, 0        ; Initialize quotient to 0
DIVISION_LOOP:

```

```

CMP AX, BX      ; Compare dividend (AX) with divisor (BX)
JL END_DIVISION ; Jump to END_DIVISION if dividend < divisor
INC CX          ; Increment quotient
SUB AX, BX      ; Subtract divisor from dividend
JMP DIVISION_LOOP ; Repeat the loop

```

```

END_DIVISION:
MOV DX, CX      ; Move quotient to DX

```

```

; Terminate program
MOV AH, 4Ch     ; Set AH to 4Ch (function to terminate program)
INT 21h         ; Call DOS interrupt to terminate program

```

```

MAIN ENDP
END MAIN

```

This code initializes the quotient to 0 and then enters a loop where it compares the dividend (AX) with the divisor (BX). If the dividend is greater than or equal to the divisor, it increments the quotient by 1 and subtracts the divisor from the dividend. The loop continues until the dividend is less than the divisor. Finally, it moves the quotient to DX and terminates the program.

The following algorithm may be used to carry out multiplication of two positive numbers M and N by repeated addition: initialize product to 0 REPEAT add M to product decrement N UNTIL N = 0 Write a sequence of instructions to multiply AX by BX, and put the product in DX. You may ignore the possibility of overflow.

Here's the assembly language code to carry out multiplication of two positive numbers by repeated addition:

```

assembly
.MODEL SMALL
.STACK 100h
.DATA
.CODE
MAIN PROC
MOV AX, @DATA ; Load the data segment address into AX
MOV DS, AX    ; Set the data segment register

MOV CX, 0     ; Initialize product to 0
MULTIPLICATION_LOOP:
TEST BX, BX   ; Check if BX (N) is zero
JZ END_MULTIPLICATION ; If N is zero, exit the loop

ADD CX, AX    ; Add M (AX) to product (CX)
DEC BX       ; Decrement N (BX)

```

JMP MULTIPLICATION_LOOP ; Repeat the loop

END_MULTIPLICATION:

MOV DX, CX ; Move product to DX

; Terminate program

MOV AH, 4Ch ; Set AH to 4Ch (function to terminate program)

INT 21h ; Call DOS interrupt to terminate program

MAIN ENDP

END MAIN

This code initializes the product to 0 and then enters a loop where it adds M (AX) to the product (CX) and decrements N (BX) until N becomes zero. Finally, it moves the product to DX and terminates the program.

Write a program to display a "?", read two capital letters, and display them on the next line in alphabetical order.

Here's an assembly language program to achieve the given task:

assembly

.MODEL SMALL

.STACK 100h

.DATA

promptMsg DB 'Enter two capital letters: \$' ; Prompt message

errorMsg DB 'Please enter capital letters!\$' ; Error message

newline DB 0AH, 0DH, '\$' ; New line characters

.CODE

MAIN PROC

MOV AX, @DATA ; Load the data segment address into AX

MOV DS, AX ; Set the data segment register

; Display a question mark

MOV AH, 02h ; Set AH to 02h (function to display character)

MOV DL, '?' ; Load the question mark character into DL

INT 21h ; Call DOS interrupt to display the character

; Prompt user to enter two capital letters

LEA DX, promptMsg ; Load address of prompt message into DX

MOV AH, 09h ; Set AH to 09h (function to display string)

INT 21h ; Display prompt message

; Read the first capital letter

READ_FIRST_LETTER:

MOV AH, 01h ; Set AH to 01h (function to read character)

INT 21h ; Call DOS interrupt to read character

CMP AL, 'A' ; Check if input is a capital letter


```

JB INVALID_INPUT    ; Jump if below 'A'
CMP AL, 'Z'         ; Check if input is a capital letter
JA INVALID_INPUT    ; Jump if above 'Z'
MOV BL, AL          ; Store the first capital letter
; Read the second capital letter
READ_SECOND_LETTER:
MOV AH, 01h         ; Set AH to 01h (function to read character)
INT 21h            ; Call DOS interrupt to read character
CMP AL, 'A'         ; Check if input is a capital letter
JB INVALID_INPUT    ; Jump if below 'A'
CMP AL, 'Z'         ; Check if input is a capital letter
JA INVALID_INPUT    ; Jump if above 'Z'
MOV BH, AL          ; Store the second capital letter

; Display the capital letters in alphabetical order
MOV AH, 02h         ; Set AH to 02h (function to display character)
MOV DL, BL          ; Load the first capital letter into DL
INT 21h            ; Display the first capital letter
MOV DL, BH          ; Load the second capital letter into DL
INT 21h            ; Display the second capital letter

; Display newline characters
MOV AH, 09h         ; Set AH to 09h (function to display string)
LEA DX, newline     ; Load address of newline characters into DX
INT 21h            ; Call DOS interrupt to display newline characters

; Terminate program
MOV AH, 4Ch         ; Set AH to 4Ch (function to terminate program)
INT 21h            ; Call DOS interrupt to terminate program

INVALID_INPUT:
; Display error message
LEA DX, errorMsg    ; Load address of error message into DX
MOV AH, 09h         ; Set AH to 09h (function to display string)
INT 21h            ; Display error message
JMP READ_FIRST_LETTER ; Jump back to read the first capital letter again

MAIN ENDP
END MAIN

```

This program displays a question mark, prompts the user to enter two capital letters, reads and validates the input, displays the letters on the next line in alphabetical order, and then terminates the program. If the input is invalid, it displays an error message and prompts the user to enter the letters again.

Write a program that will prompt the user to enter a hex digit character ("0" ... "9" or "A" ... "F"), display it on the next line in decimal, and ask the

user i.e he or she wants to do it again. If the user types "y" or "Y", the program repeats; If the user types anything else, the program terminates. If the user enters an illegal character, prompt the user to try again. ENTER A HEX DIGIT: 9 IN DECIMAL IS IT 9 DO YOU WANT TO DO IT AGAIN? y ENTER A HEX DIGIT: c ILLEGAL CHARACTER - ENTER 0 .. 9 OR A .. F: C IN DECIMAL IT IS 12 DO YOU WANT TO DO IT AGAIN? N

Here's an assembly language program to accomplish the given task:

```
assembly
.MODEL SMALL
.STACK 100h
.DATA
promptMsg    DB 'ENTER A HEX DIGIT: $'      ; Prompt message
decimalMsg    DB 'IN DECIMAL IT IS: $'      ; Message for decimal conversion
illegalMsg    DB 'ILLEGAL CHARACTER - ENTER 0 .. 9 OR A .. F: $' ; Error message
repeatMsg     DB 'DO YOU WANT TO DO IT AGAIN? $' ; Prompt message for repetition
newline       DB 0AH, 0DH, '$'              ; New line characters
.CODE
MAIN PROC
MOV AX, @DATA    ; Load the data segment address into AX
MOV DS, AX       ; Set the data segment register

CONVERSION_LOOP:
; Prompt user to enter a hex digit
LEA DX, promptMsg ; Load address of prompt message into DX
MOV AH, 09h       ; Set AH to 09h (function to display string)
INT 21h           ; Display prompt message

; Read a character from input
READ_INPUT:
MOV AH, 01h       ; Set AH to 01h (function to read character)
INT 21h           ; Call DOS interrupt to read character
CMP AL, '0'       ; Check if input is between '0' and '9'
JB INVALID_INPUT  ; Jump if below '0'
CMP AL, '9'       ; Check if input is between '0' and '9'
JBE CONVERT_DECIMAL ; Jump if between '0' and '9'
CMP AL, 'A'       ; Check if input is between 'A' and 'F'
JB INVALID_INPUT  ; Jump if below 'A'
CMP AL, 'F'       ; Check if input is between 'A' and 'F'
JA INVALID_INPUT  ; Jump if above 'F'

CONVERT_DECIMAL:
; Convert hex digit to decimal
CMP AL, 'A'       ; Check if input is between 'A' and 'F'
JB SKIP_SUBTRACTION ; Jump if below 'A'
```

```

SUB AL, 7      ; Adjust ASCII value to convert 'A'-'F' to decimal (10-15)
SKIP_SUBTRACTION:
SUB AL, '0'    ; Convert ASCII character to numeric value
MOV BL, AL     ; Store the decimal value

; Display the decimal value
LEA DX, decimalMsg ; Load address of decimal message into DX
MOV AH, 09h    ; Set AH to 09h (function to display string)
INT 21h        ; Display decimal message
ADD BL, 30h    ; Convert decimal value to ASCII character
MOV DL, BL     ; Load ASCII character into DL
MOV AH, 02h    ; Set AH to 02h (function to display character)
INT 21h        ; Display ASCII character

; Ask if the user wants to repeat the process
LEA DX, repeatMsg ; Load address of repeat message into DX
MOV AH, 09h    ; Set AH to 09h (function to display string)
INT 21h        ; Display repeat message

; Read the response from the user
MOV AH, 01h    ; Set AH to 01h (function to read character)
INT 21h        ; Call DOS interrupt to read character
CMP AL, 'y'    ; Check if user wants to repeat (input 'y' or 'Y')
JE CONVERSION_LOOP ; Jump to CONVERSION_LOOP if user wants to repeat

; Terminate program
MOV AH, 4Ch    ; Set AH to 4Ch (function to terminate program)
INT 21h        ; Call DOS interrupt to terminate program

INVALID_INPUT:
; Display error message
LEA DX, illegalMsg ; Load address of illegal message into DX
MOV AH, 09h    ; Set AH to 09h (function to display string)
INT 21h        ; Display illegal message
JMP READ_INPUT ; Jump back to read input again

MAIN ENDP
END MAIN

```

This program prompts the user to enter a hex digit, converts it to decimal, displays the decimal value, and then asks if the user wants to repeat the process. If the user enters an illegal character, it prompts the user to try again. If the user types 'y' or 'Y', the program repeats; otherwise, it terminates.

Do programming exercise ABOVE DID, except that if the user fails to enter a hex-digit character In three tries, display a message and terminate the program.

Here's the modified assembly language program that terminates if the user fails to enter a hex-digit character after three tries:

```
assembly
.MODEL SMALL
.STACK 100h
.DATA
promptMsg    DB 'ENTER A HEX DIGIT: $'      ; Prompt message
decimalMsg    DB 'IN DECIMAL IT IS: $'      ; Message for decimal conversion
illegalMsg    DB 'ILLEGAL CHARACTER - ENTER 0 .. 9 OR A .. F: $' ; Error message
repeatMsg     DB 'DO YOU WANT TO DO IT AGAIN? $' ; Prompt message for repetition
tryCountMsg   DB 'TRIES REMAINING: $'       ; Message for remaining tries
errorMsg      DB 'THREE UNSUCCESSFUL TRIES. TERMINATING PROGRAM.$' ; Error message
for three unsuccessful tries
newline       DB 0AH, 0DH, '$'             ; New line characters
.CODE
MAIN PROC
MOV AX, @DATA    ; Load the data segment address into AX
MOV DS, AX       ; Set the data segment register

MOV CX, 3        ; Initialize try count to 3

CONVERSION_LOOP:
; Prompt user to enter a hex digit
LEA DX, promptMsg ; Load address of prompt message into DX
MOV AH, 09h      ; Set AH to 09h (function to display string)
INT 21h          ; Display prompt message

; Read a character from input
READ_INPUT:
MOV AH, 01h      ; Set AH to 01h (function to read character)
INT 21h          ; Call DOS interrupt to read character
CMP AL, '0'      ; Check if input is between '0' and '9'
JB INVALID_INPUT ; Jump if below '0'
CMP AL, '9'      ; Check if input is between '0' and '9'
JBE CONVERT_DECIMAL ; Jump if between '0' and '9'
CMP AL, 'A'      ; Check if input is between 'A' and 'F'
JB INVALID_INPUT ; Jump if below 'A'
CMP AL, 'F'      ; Check if input is between 'A' and 'F'
JA INVALID_INPUT ; Jump if above 'F'

CONVERT_DECIMAL:
; Convert hex digit to decimal
CMP AL, 'A'      ; Check if input is between 'A' and 'F'
JB SKIP_SUBTRACTION ; Jump if below 'A'
SUB AL, 7        ; Adjust ASCII value to convert 'A'-'F' to decimal (10-15)
SKIP_SUBTRACTION:
```

```
SUB AL, '0'      ; Convert ASCII character to numeric value
MOV BL, AL       ; Store the decimal value
```

```
; Display the decimal value
```

```
LEA DX, decimalMsg ; Load address of decimal message into DX
MOV AH, 09h        ; Set AH to 09h (function to display string)
INT 21h            ; Display decimal message
ADD BL, 30h        ; Convert decimal value to ASCII character
MOV DL, BL         ; Load ASCII character into DL
MOV AH, 02h        ; Set AH to 02h (function to display character)
INT 21h            ; Display ASCII character
```

```
; Ask if the user wants to repeat the process
```

```
LEA DX, repeatMsg  ; Load address of repeat message into DX
MOV AH, 09h        ; Set AH to 09h (function to display string)
INT 21h            ; Display repeat message
```

```
; Read the response from the user
```

```
MOV AH, 01h        ; Set AH to 01h (function to read character)
INT 21h            ; Call DOS interrupt to read character
CMP AL, 'y'        ; Check if user wants to repeat (input 'y' or 'Y')
JE CONVERSION_LOOP ; Jump to CONVERSION_LOOP if user wants to repeat
CMP CX, 0          ; Check if try count is zero
JZ TERMINATE_PROGRAM ; If try count is zero, terminate the program
DEC CX            ; Decrement try count
; Display remaining try count
LEA DX, tryCountMsg ; Load address of try count message into DX
MOV AH, 09h        ; Set AH to 09h (function to display string)
INT 21h            ; Display try count message
MOV DL, '0'        ; Load ASCII character '0' into DL
ADD DL, CL         ; Add try count value to DL
MOV AH, 02h        ; Set AH to 02h (function to display character)
INT 21h            ; Display remaining try count
JMP READ_INPUT     ; Jump back to read input again
```

```
INVALID_INPUT:
```

```
; Display error message
```

```
LEA DX, illegalMsg ; Load address of illegal message into DX
MOV AH, 09h        ; Set AH to 09h (function to display string)
INT 21h            ; Display illegal message
JMP READ_INPUT     ; Jump back to read input again
```

```
TERMINATE_PROGRAM:
```

```
; Display error message for three unsuccessful tries
```

```
LEA DX, errorMsg   ; Load address of error message into DX
MOV AH, 09h        ; Set AH to 09h (function to display string)
```

```
INT 21h      ; Display error message
; Terminate program
MOV AH, 4Ch   ; Set AH to 4Ch (function to terminate program)
INT 21h      ; Call DOS interrupt to terminate program
```

```
MAIN ENDP
END MAIN
```

This program prompts the user to enter a hex digit, converts it to decimal, displays the decimal value, and then asks if the user wants to repeat the process. If the user enters an illegal character, it prompts the user to try again. If the user fails to enter a hex-digit character after three tries, it displays an error message and terminates the program.

For each of the following instructions, give the new destination contents and the new settings of CF, SF, tF, Pf, and OF. Suppose that the flags are initially 0 in each part at this question. ADD AX,BX where AX contains 7FFFh and BX contains 0001h

To evaluate the effect of the ADD instruction on the destination contents and the flags (CF, SF, ZF, PF, and OF), let's calculate the result step by step:

Given:

- AX = 7FFFh
- BX = 0001h

1. Performing the Addition:

```
makefile
AX = AX + BX
= 7FFFh + 0001h
= 8000h
```

2. Destination Contents:

- AX = 8000h

3. Flag Settings:

- **CF (Carry Flag):** 0 (No carry)
- **SF (Sign Flag):** 1 (Negative result, since the most significant bit of the result is set)
- **ZF (Zero Flag):** 0 (Result is non-zero)
- **PF (Parity Flag):** 0 (Parity bit count is even)
- **OF (Overflow Flag):** 1 (Overflow occurred, as the result is beyond the representable range of a signed 16-bit integer)

So, after the ADD AX, BX instruction:

- **Destination Contents:** AX = 8000h
- **Flag Settings:**
 - CF = 0
 - SF = 1
 - ZF = 0
 - PF = 0

- OF = 1

SUB AL,BL Where AL contains 01h and BL contains FFh

To analyze the effect of the SUB instruction on the destination contents and the flags (CF, SF, ZF, PF, and OF), let's perform the subtraction step by step:

Given:

- AL = 01h
- BL = FFh

1. Performing the Subtraction:

makefile

AL = AL - BL

= 01h - FFh

= 02h (after borrowing)

2. Destination Contents:

- AL = 02h

3. Flag Settings:

- **CF (Carry Flag):** 0 (No borrow)
- **SF (Sign Flag):** 0 (Positive result)
- **ZF (Zero Flag):** 0 (Result is non-zero)
- **PF (Parity Flag):** 0 (Parity bit count is even)
- **OF (Overflow Flag):** 0 (No overflow, since there is no signed overflow when subtracting a positive number from another positive number)

So, after the SUB AL, BL instruction:

- **Destination Contents:** AL = 02h
- **Flag Settings:**
 - CF = 0
 - SF = 0
 - ZF = 0
 - PF = 0
 - OF = 0

DEC AL Where A~ contains 00h

To analyze the effect of the DEC instruction on the destination contents and the flags (CF, SF, ZF, PF, and OF), let's perform the decrement operation step by step:

Given:

- AL = 00h

1. Performing the Decrement:

makefile

AL = AL - 1

= 00h - 1

= FFh (after borrow)

2. Destination Contents:

- AL = FFh

3. Flag Settings:

- **CF (Carry Flag):** 1 (Borrow occurred)
- **SF (Sign Flag):** 1 (Negative result)
- **ZF (Zero Flag):** 0 (Result is non-zero)
- **PF (Parity Flag):** 1 (Parity bit count is odd)
- **OF (Overflow Flag):** Not affected (DEC instruction doesn't affect the overflow flag)

So, after the DEC AL instruction:

- **Destination Contents:** AL = FFh
- **Flag Settings:**
 - CF = 1
 - SF = 1
 - ZF = 0
 - PF = 1
 - OF = Not affected

NEG AL Where AL contains 7Fh

To analyze the effect of the NEG instruction on the destination contents and the flags (CF, SF, ZF, PF, and OF), let's perform the negation operation step by step:

Given:

- AL = 7Fh

1. Performing the Negation:

makefile

AL = 0 - AL

= 0 - 7Fh

= 81h (after borrow)

2. Destination Contents:

- AL = 81h

3. Flag Settings:

- **CF (Carry Flag):** 1 (Borrow occurred)
- **SF (Sign Flag):** 1 (Negative result)
- **ZF (Zero Flag):** 0 (Result is non-zero)
- **PF (Parity Flag):** 0 (Parity bit count is even)
- **OF (Overflow Flag):** 1 (Overflow occurred, since there is no positive representation of 81h in a signed 8-bit integer)

So, after the NEG AL instruction:

- **Destination Contents:** AL = 81h
- **Flag Settings:**
 - CF = 1
 - SF = 1

- ZF = 0
- PF = 0
- OF = 1

XCHG AX,BX where AX contains 1ABCh and BX contains 712Ah

To analyze the effect of the XCHG instruction on the destination contents, let's swap the values of AX and BX:

Given:

- AX = 1ABCh
- BX = 712Ah

1. **Performing the Swap:**After the swap:

- AX = 712Ah
- BX = 1ABCh

2. **Flag Settings:** The XCHG instruction does not affect any flags (CF, SF, ZF, PF, AF, or OF).

So, after the XCHG AX, BX instruction:

- **Destination Contents:**
 - AX = 712Ah
 - BX = 1ABCh
- **Flag Settings:** Unaffected

NEG 'AX where AX contains 0001h

The NEG instruction negates the value of the operand. However, NEG operates directly on memory or registers, not on immediate values or constants. Therefore, you cannot directly apply NEG to the value 'AX' where AX contains 0001h.

If you want to negate the value stored in the AX register, you can do it like this:

Given:

- AX = 0001h

1. **Performing the Negation:**

makefile

AX = 0 - AX

= 0 - 0001h

= FFFFh (after borrow)

2. **Destination Contents:**

- AX = FFFFh

3. **Flag Settings:**

- **CF (Carry Flag):** 1 (Borrow occurred)
- **SF (Sign Flag):** 1 (Negative result)
- **ZF (Zero Flag):** 0 (Result is non-zero)
- **PF (Parity Flag):** 1 (Parity bit count is odd)

- **OF (Overflow Flag):** 1 (Overflow occurred, since there is no positive representation of FFFFh in a signed 16-bit integer)

So, after the NEG AX instruction:

- **Destination Contents:** AX = FFFFh
- **Flag Settings:**
 - CF = 1
 - SF = 1
 - ZF = 0
 - PF = 1
 - OF = 1

Suppose that AX and BX both contain positive numbers, and ADD AX, BX is executed. Show that there is a carry into the msb but no carry out of the msb if, and only if, signed overflow occurs.

Let's analyze the scenario where ADD AX, BX is executed, assuming both AX and BX contain positive numbers. We'll demonstrate that there is a carry into the Most Significant Bit (MSB) but no carry out of the MSB if, and only if, signed overflow occurs.

Given:

- AX and BX both contain positive numbers.

1. Performing the Addition:

makefile

AX = AX + BX

2. Carry into the MSB:

- If the addition of the least significant bits of AX and BX results in a carry, then there will be a carry into the MSB.

3. Carry out of the MSB:

- Since both AX and BX contain positive numbers, their MSBs are both 0. Therefore, if there's a carry into the MSB, it means the sum of the MSBs is 1. If the result is greater than the maximum positive number that can be represented by a signed number, then signed overflow occurs. This happens when the MSB of the result is 1.
- However, there's no carry out of the MSB if the result fits within the representable range of a signed number.

4. Conclusion:

- There is a carry into the MSB but no carry out of the MSB if, and only if, signed overflow occurs. This occurs when the result of the addition exceeds the maximum positive value that can be represented by a signed number.

Suppose AX and BX both contain negative numbers, and ADD AX, BX is executed. Show that there is a carry out of the msb but no carry into the msb if, and only if, signed overflow occurs.

Let's analyze the scenario where ADD AX, BX is executed, assuming both AX and BX contain negative numbers. We'll demonstrate that there is a carry out of the Most Significant Bit (MSB) but no carry into the MSB if, and only if, signed overflow occurs.

Given:

- AX and BX both contain negative numbers.

1. Performing the Addition:

makefile

AX = AX + BX

2. Carry out of the MSB:

- Since both AX and BX contain negative numbers, their MSBs are both 1. Therefore, if there's an overflow during addition, it will cause a carry out of the MSB.

3. Carry into the MSB:

- There will be no carry into the MSB if the result is less than the minimum negative number that can be represented by a signed number.

4. Conclusion:

- There is a carry out of the MSB but no carry into the MSB if, and only if, signed overflow occurs. This occurs when the result of the addition exceeds the maximum negative value that can be represented by a signed number.

Suppose ADD AX,BX is executed. In each of the following parts, the first number being added is the contents of AX, and the second number is the contents of BX. Give the resulting value of AX and tell whether signed or unsigned overflow occurred. 512Ch + 4185h

To calculate the result of ADD AX, BX and determine if signed or unsigned overflow occurred, we need to perform the addition:

Given:

- AX contains 512Ch
- BX contains 4185h

1. Performing the Addition: Since AX and BX contain positive numbers, we can add them directly.

= 92B1h

2. Signed Overflow:

- To check for signed overflow, we interpret AX and BX as signed numbers. If the sign of the result differs from the signs of both operands, signed overflow occurs.
- In this case, both AX and BX are positive numbers. The result is also positive, so there is no signed overflow.

3. Unsigned Overflow:

- To check for unsigned overflow, we interpret AX and BX as unsigned numbers. If the result exceeds the maximum value that can be represented by the data type, unsigned overflow occurs.
- In this case, the result 92B1h does not exceed the maximum value for a 16-bit

unsigned integer (FFFFh), so there is no unsigned overflow.

So, after ADD AX, BX, the resulting value of AX is 92B1h, and neither signed nor unsigned overflow occurred.

Write a program to (a) prompt the user, (b) read first, middle, and last initials of a person's name, and (c) display them down the left margin.

Sample execution: ENTER THREE INITIALS: JFK J f K

Here's an assembly language program to accomplish the given task:

```
assembly
.MODEL SMALL
.STACK 100h
.DATA
promptMsg    DB 'ENTER THREE INITIALS: $' ; Prompt message
initials     DB 3 DUP('$')                ; Array to store initials
newline      DB 0AH, 0DH, '$'            ; New line characters
.CODE
MAIN PROC
MOV AX, @DATA    ; Load the data segment address into AX
MOV DS, AX       ; Set the data segment register

; Prompt user to enter three initials
LEA DX, promptMsg ; Load address of prompt message into DX
MOV AH, 09h       ; Set AH to 09h (function to display string)
INT 21h           ; Display prompt message

; Read three initials from input
MOV CX, 3         ; Initialize loop counter to 3 (for three initials)
LEA DI, initials  ; Load address of initials array into DI
READ_INITIALS:
MOV AH, 01h       ; Set AH to 01h (function to read character)
INT 21h           ; Call DOS interrupt to read character
MOV [DI], AL      ; Store the initial in the array
INC DI            ; Move to the next element in the array
CMP AL, ' '       ; Check if the current character is a space
JNE READ_INITIALS ; If not, continue reading initials
LOOP READ_INITIALS ; Repeat until all three initials are read

; Display the initials down the left margin
MOV CX, 3         ; Reset loop counter to 3
LEA DI, initials  ; Reset DI to point to the beginning of the array
DISPLAY_INITIALS:
MOV DL, [DI]      ; Load the initial into DL
MOV AH, 02h       ; Set AH to 02h (function to display character)
INT 21h           ; Display the initial
```

```

MOV DL, newline    ; Load newline characters into DL
INT 21h            ; Display newline characters
INC DI             ; Move to the next element in the array
LOOP DISPLAY_INITIALS ; Repeat for all three initials

; Terminate program
MOV AH, 4Ch        ; Set AH to 4Ch (function to terminate program)
INT 21h            ; Call DOS interrupt to terminate program

MAIN ENDP
END MAIN

```

This program prompts the user to enter three initials, reads them from input, and then displays them down the left margin. Each initial is displayed on a separate line.

Write a program to read one of the hex digits A-I, and display it on the next line in decimal. Sample execution: ENTER A HEX DIGIT: C IN DECIMAL IT IS 12

Here's an assembly language program to accomplish the given task:

```

assembly
.MODEL SMALL
.STACK 100h
.DATA
promptMsg    DB 'ENTER A HEX DIGIT: $' ; Prompt message
decimalMsg   DB 'IN DECIMAL IT IS: $'  ; Message for decimal conversion
newline      DB 0AH, 0DH, '$'         ; New line characters
.CODE
MAIN PROC
MOV AX, @DATA ; Load the data segment address into AX
MOV DS, AX    ; Set the data segment register

; Prompt user to enter a hex digit
LEA DX, promptMsg ; Load address of prompt message into DX
MOV AH, 09h       ; Set AH to 09h (function to display string)
INT 21h           ; Display prompt message

; Read a character from input
READ_INPUT:
MOV AH, 01h       ; Set AH to 01h (function to read character)
INT 21h           ; Call DOS interrupt to read character

; Convert hex digit to decimal
CMP AL, 'A'       ; Check if input is between 'A' and 'I'
JB INVALID_INPUT  ; Jump if below 'A'
CMP AL, 'I'       ; Check if input is between 'A' and 'I'
JA INVALID_INPUT  ; Jump if above 'I'

```

SUB AL, 'A' - 1 ; Adjust ASCII value to convert 'A'-'I' to decimal (10-18)

; Display the decimal value

LEA DX, decimalMsg ; Load address of decimal message into DX

MOV AH, 09h ; Set AH to 09h (function to display string)

INT 21h ; Display decimal message

ADD AL, 30h ; Convert decimal value to ASCII character

MOV DL, AL ; Load ASCII character into DL

MOV AH, 02h ; Set AH to 02h (function to display character)

INT 21h ; Display ASCII character

; Display newline characters

MOV DX, OFFSET newline ; Load address of newline characters into DX

MOV AH, 09h ; Set AH to 09h (function to display string)

INT 21h ; Display newline characters

; Terminate program

MOV AH, 4Ch ; Set AH to 4Ch (function to terminate program)

INT 21h ; Call DOS interrupt to terminate program

INVALID_INPUT:

; Display error message

MOV DX, OFFSET newline ; Load address of newline characters into DX

MOV AH, 09h ; Set AH to 09h (function to display string)

INT 21h ; Display newline characters

MOV DX, OFFSET newline ; Load address of newline characters into DX

MOV AH, 09h ; Set AH to 09h (function to display string)

INT 21h ; Display newline characters

MOV DX, OFFSET newline ; Load address of newline characters into DX

MOV AH, 09h ; Set AH to 09h (function to display string)

INT 21h ; Display newline characters

MOV DX, OFFSET newline ; Load address of newline characters into DX

MOV AH, 09h ; Set AH to 09h (function to display string)

INT 21h ; Display newline characters

MOV DX, OFFSET newline ; Load address of newline characters into DX

MOV AH, 09h ; Set AH to 09h (function to display string)

INT 21h ; Display newline characters

MOV DX, OFFSET newline ; Load address of newline characters into DX

MOV AH, 09h ; Set AH to 09h (function to display string)

INT 21h ; Display newline characters

MOV DX, OFFSET newline ; Load address of newline characters into DX

MOV AH, 09h ; Set AH to 09h (function to display string)

INT 21h ; Display newline characters

MOV DX, OFFSET newline ; Load address of newline characters into DX

MOV AH, 09h ; Set AH to 09h (function to display string)

INT 21h ; Display newline characters

```
JMP READ_INPUT      ; Jump back to read input again
```

```
MAIN ENDP
```

```
END MAIN
```

This program prompts the user to enter a hex digit, reads it from input, converts it to decimal, and then displays the decimal value on the next line. If the user enters an invalid hex digit, it displays an error message and prompts the user to try again.

Write a program to (a) display a "?", (b) read two decimal digits whose sum is less than 10, (c) display them and their sum on the next line, with an appropriate message. Sample execution: ?27 '!'HE SUM OF 2 .'AND 7 IS 9

Here's an assembly language program to accomplish the given task:

```
assembly
```

```
.MODEL SMALL
```

```
.STACK 100h
```

```
.DATA
```

```
promptMsg    DB '?$'          ; Prompt message
```

```
errorMsg     DB 'INVALID INPUT!$' ; Error message
```

```
sumMsg       DB 'THE SUM OF ', '$' ; Message for sum output
```

```
newline      DB 0AH, 0DH, '$'    ; New line characters
```

```
.CODE
```

```
MAIN PROC
```

```
MOV AX, @DATA      ; Load the data segment address into AX
```

```
MOV DS, AX          ; Set the data segment register
```

```
; Display a "?"
```

```
LEA DX, promptMsg   ; Load address of prompt message into DX
```

```
MOV AH, 09h         ; Set AH to 09h (function to display string)
```

```
INT 21h             ; Display prompt message
```

```
; Read two decimal digits
```

```
MOV CX, 2           ; Initialize loop counter to 2 (for two digits)
```

```
XOR BX, BX          ; Initialize BX to store the sum
```

```
READ_DIGITS:
```

```
MOV AH, 01h         ; Set AH to 01h (function to read character)
```

```
INT 21h             ; Call DOS interrupt to read character
```

```
SUB AL, 30h         ; Convert ASCII digit to decimal
```

```
CMP AL, 0AH         ; Check if input is a valid decimal digit (0-9)
```

```
JA INVALID_INPUT    ; Jump if not a valid digit
```

```
ADD BX, AL           ; Add the digit to the sum in BX
```

```
LOOP READ_DIGITS    ; Repeat for two digits
```

```
; Check if the sum is less than 10
```

```
CMP BX, 0Ah         ; Compare sum with 10
```

```
JAE INVALID_SUM     ; Jump if sum is not less than 10
```

```

; Display the digits and their sum
LEA DX, sumMsg      ; Load address of sum message into DX
MOV AH, 09h         ; Set AH to 09h (function to display string)
INT 21h             ; Display sum message
ADD BL, 30h         ; Convert sum to ASCII digit
MOV DL, BL          ; Load ASCII digit into DL
MOV AH, 02h         ; Set AH to 02h (function to display character)
INT 21h             ; Display ASCII digit
MOV DL, newline     ; Load newline characters into DL
INT 21h             ; Display newline characters

; Terminate program
MOV AH, 4Ch         ; Set AH to 4Ch (function to terminate program)
INT 21h             ; Call DOS interrupt to terminate program

```

```

INVALID_INPUT:
; Display error message
LEA DX, errorMsg    ; Load address of error message into DX
MOV AH, 09h         ; Set AH to 09h (function to display string)
INT 21h             ; Display error message
JMP TERMINATE_PROGRAM ; Jump to terminate program

```

```

INVALID_SUM:
; Display error message
LEA DX, errorMsg    ; Load address of error message into DX
MOV AH, 09h         ; Set AH to 09h (function to display string)
INT 21h             ; Display error message

```

```

TERMINATE_PROGRAM:
; Terminate program
MOV AH, 4Ch         ; Set AH to 4Ch (function to terminate program)
INT 21h             ; Call DOS interrupt to terminate program

```

```

MAIN ENDP
END MAIN

```

This program prompts the user to enter two decimal digits, reads them from input, checks if their sum is less than 10, and displays the digits and their sum on the next line with an appropriate message. If the user enters an invalid input or the sum is not less than 10, it displays an error message and terminates the program.

Using only MOV, ADD, SUB, INC, DEC, and NEG, translate the following high-level language assignment statements into assembly language. A, B, and C are word variables.

- $A = B - A$
- $A = -(A + 1)$
- $C = A + B$
- $B = 3 \times B + 7$
- $A = B - A - 1$

Here are the translations of the given high-level language assignment statements into assembly language using only the specified instructions:

a. $A = B - A$

assembly

```
MOV AX, B    ; Move the value of B into AX
SUB AX, A    ; Subtract the value of A from AX
MOV A, AX    ; Move the result back into A
```

b. $A := (A + 1)$

assembly

```
MOV AX, A    ; Move the value of A into AX
INC AX       ; Increment AX (A + 1)
NEG AX       ; Negate AX (-(A + 1))
MOV A, AX    ; Move the result back into A
```

c. $C := A + B$

assembly

```
MOV AX, A    ; Move the value of A into AX
ADD AX, B    ; Add the value of B to AX
NEG AX       ; Negate AX (-(A + B))
MOV C, AX    ; Move the result back into C
```

d. $B = 3 * B + 7$

assembly

```
MOV AX, B    ; Move the value of B into AX
SHL AX, 1    ; Multiply AX by 2 (2 * B)
ADD AX, B    ; Add the original value of B to AX (3 * B)
ADD AX, 7    ; Add 7 to AX (3 * B + 7)
MOV B, AX    ; Move the result back into B
```

e. $A = B - A - 1$

assembly

```
MOV AX, B    ; Move the value of B into AX
SUB AX, A    ; Subtract the value of A from AX
DEC AX       ; Decrement AX (B - A - 1)
MOV A, AX    ; Move the result back into A
```

Write instructions (not a complete program) to do the following. a. Read a character, and display it at the T'.ext position on the same line. < b. • Read an uppercase letter (omit error checking), and display it at the next position on the same line in lower case.

Here are the instructions to accomplish the given tasks:

a. Read a character and display it at the next position on the same line:

assembly

MOV AH, 01h ; Function to read a character from standard input

INT 21h ; Read character into AL

MOV AH, 02h ; Function to display character

INT 21h ; Display character

b. Read an uppercase letter and display it at the next position on the same line in lowercase:

assembly

MOV AH, 01h ; Function to read a character from standard input

INT 21h ; Read character into AL

AND AL, 11011111b ; Convert uppercase letter to lowercase (toggle bit 5)

MOV AH, 02h ; Function to display character

INT 21h ; Display character

Tell whether each of the following instructions is legal or illegal. W1 and W2 are word variables, and B1 and B2 are byte variables. a. MOV DS, AX

The instruction "MOV DS, AX" is illegal because it attempts to move a 16-bit register (AX) directly into the 16-bit segment register (DS), which is not allowed. Segment registers such as DS can only be loaded using memory operands or immediate values. The correct way to load a segment register is:

assembly

MOV AX, some_segment_value

MOV DS, AX

Where "some_segment_value" can be a memory location containing a segment value or an immediate value.

ADD B1, B2

The instruction "ADD B1, B2" is legal. It adds the contents of the byte variable B2 to the byte variable B1. This operation is allowed because both B1 and B2 are byte-sized variables, and the ADD instruction supports adding byte-sized operands.

MOV W1, DS

The instruction "MOV W1, DS" is legal. It moves the value of the segment register DS into the word variable W1. This is allowed because both W1 and DS are word-sized entities, and the MOV instruction supports moving word-sized operands.

Which of the following names are legal in IBM PC assembly language? a. TWO WORDS b. c. d. e. f. g. 2. ?l Two words . @? \$145 LET'S _GO

In IBM PC assembly language, the following names are legal:

b. ?l (Contains only valid characters) c. Two words (Contains only valid characters and spaces) d. \$145 (Starts with a valid character and may contain numbers) e. LET'S_GO (Contains only valid characters and underscore)

List one special function for each of the data registers "AX, BX, CX, and

DX.

Here is one special function for each of the data registers AX, BX, CX, and DX:

- AX: Multiplication and Division
- BX: Base register in indexed addressing mode
- CX: Loop control in loop instructions
- DX: I/O operations, such as port addressing

What is a paragraph boundary?

In the context of IBM PC assembly language, a paragraph boundary refers to a memory addressing scheme where memory addresses are aligned to paragraph boundaries, which are multiples of 16 (decimal) or 10h (hexadecimal). This alignment is important for accessing memory efficiently and is commonly used in segmentation architectures like the one used in IBM PC-compatible systems. A paragraph boundary ensures that memory is accessed in a predictable and organized manner, which is crucial for the proper functioning of programs and operating systems on these systems.

What is the maximum length (in bytes) of an instruction for the 8086-based IBM PC?

The maximum length of an instruction for the 8086-based IBM PC is 6 bytes. The 8086 processor can handle instructions of variable lengths, but the maximum length is 6 bytes. This includes the opcode and any optional prefixes or additional bytes needed for addressing modes or operands. However, most instructions are shorter, typically ranging from 1 to 3 bytes in length.

Consider a machine language instruction that moves a copy of the contents of register AX in the CI'U to a memory word. What happens during a. the fetch cycle? b. the execute cycle?

During the fetch cycle:

a. The CPU fetches the machine language instruction from memory. This involves fetching the opcode of the instruction, which specifies the operation to be performed (in this case, moving the contents of register AX to memory), and any necessary operands or addressing information.

During the execute cycle:

b. The CPU decodes the fetched instruction and determines the action to be taken. In this case, it recognizes that the instruction is a move operation from register AX to memory. The CPU then executes the instruction by transferring the contents of register AX to the specified memory location. This involves accessing the memory address specified by the instruction and writing the contents of register AX to that address.

What is the function of a. the microprocessor? b. the buses?

a. The function of the microprocessor is to execute instructions stored in memory, perform arithmetic and logical operations, and manage data transfer between various components

of a computer system. It serves as the central processing unit (CPU) of a computer, processing instructions and performing computations necessary for the operation of software programs.

b. The buses in a computer system serve as communication pathways that allow data, instructions, and control signals to be transferred between different components, such as the CPU, memory, and peripheral devices. The buses consist of multiple lines that carry signals representing data, addresses, and control signals. The data bus carries data between the CPU and memory or between the CPU and I/O devices. The address bus carries memory addresses from the CPU to memory or I/O devices. The control bus carries control signals that coordinate the operation of different components, such as signals indicating read or write operations, interrupts, and clock signals. Overall, buses facilitate the exchange of information within a computer system, enabling the various components to work together effectively.

The two parts of the microprocessor are the EU and the BIU. a. What is the function of the EU? b. What is the function of the BIU?

a. The function of the EU, or Execution Unit, is to execute instructions. It performs arithmetic and logical operations on data, controls the flow of data within the CPU, and coordinates the execution of instructions according to the program being executed. The EU is responsible for fetching instructions from memory, decoding them, and executing the required operations.

b. The function of the BIU, or Bus Interface Unit, is to manage data transfer between the CPU and other components of the computer system, such as memory and peripheral devices. It handles the fetching of instructions and data from memory, as well as the storing of results back into memory. The BIU also manages the address bus and control signals, ensuring that data is transferred accurately and efficiently between the CPU and external devices.

In the microprocessor, what is the function of a. the IP? b. a. b. the ALU? What are the I/O ports used for?

In the microprocessor:

a. The function of the IP, or Instruction Pointer, is to store the memory address of the next instruction to be fetched and executed by the CPU. It keeps track of the location in memory where the next instruction is located, allowing the CPU to sequentially fetch and execute instructions in the correct order.

b. The function of the ALU, or Arithmetic Logic Unit, is to perform arithmetic and logical operations on data. It is responsible for executing instructions that involve arithmetic operations (such as addition, subtraction, multiplication, and division) and logical operations (such as AND, OR, and NOT) on binary data. The ALU processes data according to the instructions provided by the CPU, performing calculations and generating results.

Regarding I/O ports:

The I/O ports, or Input/Output ports, are used for communication between the microprocessor and peripheral devices, such as keyboards, mice, monitors, printers, and storage devices. These ports provide a means for transferring data between the CPU and external devices. Input ports receive data from external devices and transfer it to the CPU for processing, while output ports send data from the CPU to external devices for display or storage. The CPU communicates with peripheral devices through these ports using specific protocols and interfaces, allowing for the exchange of data and control signals between the microprocessor and the external world.