Shaheed Benazir Bhutto University
Shaheed Benazirabad
KNOWLEDGE - COMMITMENT - LEADERSHIP

Computer Organization & Assembly Language

Lab Manual

STUDENT NAME

Abdul Haseeb Memon

ROLL NUMBER

24-BSCS-43

SUBJECT TEACHER

Prof: Abdul Samad Jamali

DATE

27-MAY-2025

DEPARTMENTOFCOMPUTERSCIENCE

SHAHEED BENAZIR BHUTTO UNIVERSITY, SHAHEED BENAZIR ABAD

CONTENT

S:No	***Objectives***	***Pg/no***
01	***INTRODUCTION***	*3-5*
02	***Chapter 4**: INTRODUCTION TO IBM PC ASSEMBLY LANGUAGE* • *Exercise* • *Examples*	*6-18*
03	***Chapter 6**: FLOW OF CONTROL STRUCTURE* • *Exercise* • *Examples*	*19-32*
04	***Chapter 7**: LOGIC, SHIFT, & ROTATE INSTRUCTIONS* • *Exercise* • *Examples*	*33-41*
05	***Chapter 8**: THE STACK & INTRODUCTION TO PROCEDURES* • *Exercise* • *Examples*	*42-58*

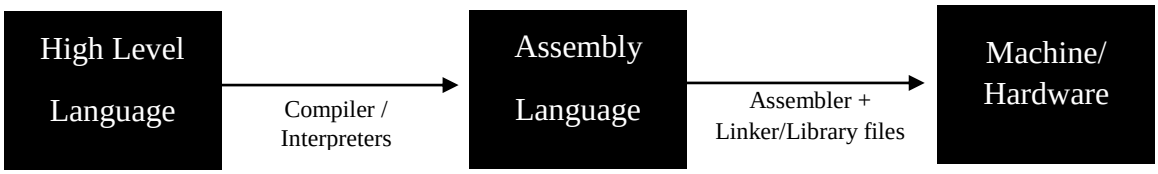
LAB MANUAL

Assembly Language &How to Assemble, Link& Execute the Assembly Language Program?

Assembly Language:

Assembly language is a low-level programming language that uses symbolic code and mnemonics to represent machine-level instructions specific to a computer's architecture. It allows direct control of hardware through human-readable instructions that correspond closely to binary machine code.

Flow of General Program:



Flow of Assembly Program:

Assembler (Software):

A program that translates assembly language code into machine code.

Eg: Masm (Microsoft Assembler), Tasm (Turbo Assembler), etc.

Object File:

An object file is a binary file that contains machine code and data produced by an assembler from source code (such as C or assembly).

Executable File:

An executable file is a binary file that contains a program in a format the operating system can load and run directly.

Assembling Assembly Code:

For Masm Assembler:

- Syntax: masm filename.asm

```
C:\MASM\BIN>masm PGM4_2.asm;
Microsoft (R) MASM Compatibility Driver
Copyright (C) Microsoft Corp 1993.  All rights reserved.

 Invoking: ML.EXE /I. /Zm /c /Ta PGM4_2.asm

Microsoft (R) Macro Assembler Version 6.14.8444
Copyright (C) Microsoft Corp 1981-1997.  All rights reserved.

 Assembling: PGM4_2.asm
```

Creating Object File:

For Masm Assembler:

- Syntax:no any syntax is there as it is created once the program assembled but it needs to link.

```
C:\MASM\BIN>dir
 Volume in drive C has no label.
 Volume Serial Number is 56F2-5005

 Directory of C:\MASM\BIN

12/05/2025  11:27 am               160 PGM4_2.obj
```

Linking the Object file:

For Masm Assembler:

Syntax:link filename.obj

```
C:\MASM\BIN>link PGM4_2.obj;

Microsoft (R) Segmented Executable Linker  Version 5.60.339 Dec  5 1994
Copyright (C) Microsoft Corp 1984-1993.  All rights reserved.


C:\MASM\BIN>
```

Executing Executable File:

For Masm Assembler:

Syntax:filename.exe

```
C:\MASM\BIN>PGM4_2.exe
HELLO!
C:\MASM\BIN>S_
```

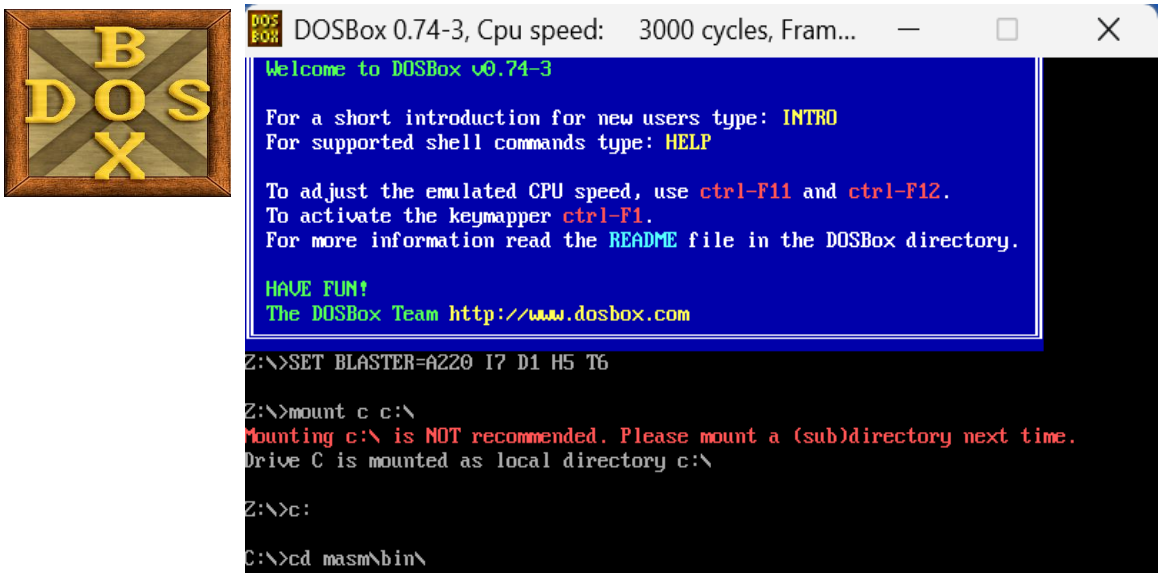
IDE :(Integrated Development Environments):

It is a software application that provides tools for writing, testing, and debugging code, typically including a code editor, compiler, and debugger in one interface.

Eg: Notebook.

Terminal:

A terminal is a command-line interface (CLI) used to interact with the operating system, where programs can be run, including through tools like DOSBox (for running old DOS programs) and CMD (Windows Command Prompt).



Chapter 4:

INTRODUCTION TO IBM PC ASSEMBLY LANGUAGE

EXAMPLES

Example1: Program1 Listing PGM4_ 1.ASM?

TITLE PGM4_1: ECHO PROGRAM

ALGORITHM:

This assembly program performs the following actions:

1. Displays the character '?' to prompt the user.
2. Reads a single character input from the user.
3. Stores the input character in register BL.
4. Prints a new line (carriage return and line feed).
5. Displays the input character back to the user.
6. Terminates the program.

CODE:

```
.model small
.stack 100h
.code
main proc
mov ah,2
mov dl,'?'
int 21h
mov ah,1
int 21h
mov  bl ,  al
mov  ah,2
mov  dl,0Dh
int 21h
mov dl,0Ah
int 21h
mov dl,bl
int 21h
mov ah,4ch
int 21h
main endp
end main
```

OUTPUT:

```
C:\MASM\BIN>hy.exe
?A
A
C:\MASM\BIN>S
```

Example 2: Program2 Listing PGM4_ 2.ASM?

TITLE PGM4_2: PRINT STRING PROGRAM

ALGORITHM:

This assembly program does the following:

1. Initializes the data segment using mov ax,@data and mov ds,ax.
2. Loads the address of the string msg ('HELLO! $') into DX.
3. Displays the string using DOS interrupt INT 21h with function AH = 9 (which prints characters until it sees $).
4. Terminates the program with INT 21h and AH = 4Ch.

CODE:

```
.model small
.stack 100h
.data
msg db 'HELLO! $'
.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
```

OUTPUT

```
C:\MASM\BIN>PGM4_2.exe
HELLO!
```

Example3: Program listing PGM4_3.ASM?

TITLE PGM4_3: CASE CONVERSION PROGRAM

ALGORITHM:

This assembly program converts a lowercase letter to uppercase and displays it. Here's a concise step-by-step description:

1. Initializes the data segment.
2. Displays the message: "ENTER A LOWER CASE LETTER:".
3. Reads a single character from the user (assumed to be lowercase).
4. Converts the character to uppercase by subtracting 20h (hex) from its ASCII code.
5. Stores the result in the char variable.
6. Displays the message: "IN UPPER CASE IT IS: ", followed by the converted character (due to char DB ?, '$'). 7.Exits the program

CODE:

```
.model small
.stack 100h
.data
cr EQU 0Dh
lf EQU 0Ah
msg1 DB 'ENTER A LOWER CASE LETTER: $'
msg2 DB 0Dh,0Ah,'IN UPPER CASE IT IS: '
char DB ?,'$'
.code
main proc
mov ax,@data
mov ds,ax
LEA dx,msg1
mov ah,9
int 21h
mov ah,1
int 21h
sub al,20h
mov char,al
LEA dx,msg2
mov ah,9
int 21h
mov ah,4ch
int 21h
main endp
end main
```

OUTPUT:

```
C:\MASM\BIN>PGM4_3.exe
ENTER A LOWER CASE LETTER: a
IN UPPER CASE IT IS: A
```

8. 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
THE SUM OF 2 AND 7 is 9

ALGORITHM:

1. Initializes the data segment.
2. Displays the prompt '?' to ask for input.
3. Reads the first ASCII digit, stores it in digit1, converts it to a number, and stores the result in BL.
4. Reads the second ASCII digit, stores it in digit2, converts it to a number, and stores the result in BH.
5. Adds the two digits and stores the result in sum.
6. Displays the result in this format:

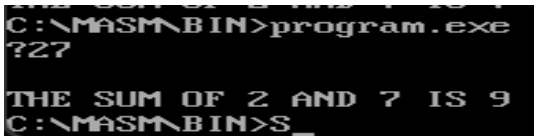
CODE:

```
.model small
.stack 100h
.data
msg1 db 13,10, 'THE SUM OF ', '$'
msg2 db ' AND ', '$'
msg3 db ' IS ', '$'
newline db 13,10, '$'
digit1 db ?  ; to store first ASCII digit
digit2 db ?  ; to store second ASCII digit
sum db ?  ; to store sum
.code
main proc
mov ax, @data
mov ds, ax
; Display '?'
mov ah, 2
mov dl, '?'
int 21h
mov ah, 1  ; === Read first digit ===
int 21h
mov digit1, al          ; store ASCII character
sub al, '0'             ; convert ASCII to number
```

```
mov bl, al              ; store in BL
; === Read second digit ===
mov ah, 1
int 21h
mov digit2, al
sub al, '0'
mov bh, al              ; store in BH


; === Calculate sum ===
add bl, bh              ; BL = digit1 + digit2
mov sum, bl             ; store sum
mov ah, 9  ; === Print output ,New line
lea dx, newline
int 21h
lea dx, msg1  ; "THE SUM OF "
int 21h
mov dl, digit1
mov ah, 2
int 21h; " AND "
lea dx, msg2
mov ah, 9
int 21h
; Print digit2 (ASCII)
mov dl, digit2
mov ah, 2
int 21h
; " IS "
lea dx, msg3
mov ah, 9
int 21h
; Print sum (convert to ASCII)
mov al, sum
add al, '0'
mov dl, al
mov ah, 2
int 21h
; Exit
mov ah, 4Ch
int 21h
main endp
end main
```

OUTPUT:



9. 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

ALGORITHM:

1. Initializes the data segment using mov ax, @data and mov ds, ax.
2. Displays the prompt "ENTER THREE INITIALS:" using DOS interrupt INT 21h (function AH = 9).
3. Reads three characters (initials) from the user:
4. It uses a loop (read_loop) to read each character with INT 21h (function AH = 1), storing them in the initials array.
5. Prints each of the three initials on a new line:
6. A loop (print_loop) is used to print a newline first (using INT 21h, function AH = 9 for the newline), then each character is printed using INT 21h (function AH = 2).
7. Exits the program using INT 21h (function AH = 4Ch).

CODE:

```
.model small
.stack 100h
.data
   prompt db 'ENTER THREE INITIALS: $'
   newline db 13, 10, '$'  ; Carriage return + Line feed
   initials db 3 dup(?)    ; Space to store 3 initials
.code
main proc
   mov ax, @data
   mov ds, ax
   ; Display prompt
   mov ah, 9
   lea dx, prompt
   int 21h
   ; === Read 3 characters ===
   mov cx, 3         ; counter for 3 characters
   lea si, initials
read_loop:
   mov ah, 1         ; read character
   int 21h
```

```
    mov [si], al       ; store in initials
    inc si
    loop read_loop
    ; === Print each character on a new line ===
    lea si, initials
    mov cx, 3
print_loop:
; Print newline
    mov ah, 9
    lea dx, newline
    int 21h
    ; Print character
    mov dl, [si]
    mov ah, 2
    int 21h
    inc si
    loop print_loop
    mov ah, 4Ch
    int 21h
main endp
end main
```

OUTPUT:

```
C:\MASM\BIN>program2.exe
ENTER THREE INITIALS: HAS
H
A
S
```

10. Write a program to read one of the hex digits A-F, and display it on the next line in decimal.

Sample execution:
ENTER A HEX DIGIT: C

IN DECIMAL IT IS 12

ALGORITHM:

1. Initializes the data segment using mov ax, @data and mov ds, ax.
2. Displays the prompt "ENTER A HEX DIGIT:" using DOS interrupt INT 21h with function AH = 9.
3. Reads a single character (hex digit) from the user:
4. Uses INT 21h with function AH = 1 to read the input character and store it in the digit variable.

5. Converts the input character:
6. If the character is between 'A' and 'F', it adjusts the ASCII value by subtracting 'A' (to make 'A' = 0, 'B' = 1, ..., 'F' = 5) and then adds 10 to get a decimal value between 10 and 15.
7. The result is stored in the value variable.
8. Prints a newline using INT 21h with function AH = 9.
9. Displays the message "IN DECIMAL IT IS:" using INT 21h with function ah=9.
10. Converts the decimal value (between 10 and 15) to ASCII and prints it:
11. If the decimal value is 10 or greater, it prints the tens digit ('1' for values 10–15) followed by the units digit (0–9).
12. If the value is less than 10, it directly converts it to ASCII and prints the number.
13. Exits the program using INT 21h with function AH = 4Ch.

CODE:

```
.model small
.stack 100h
.data
   prompt db 'ENTER A HEX DIGIT: $'
   newline db 13,10,'$'
   msg db 'IN DECIMAL IT IS $'
   digit db ?
   value db ? ; decimal value (0–15)
.code
main proc
   mov ax, @data
   mov ds, ax
   ; Show prompt
   mov ah, 9
   lea dx, prompt
   int 21h
   ; Read one character
   mov ah, 1
   int 21h
   mov digit, al
; Convert ASCII hex 'A'-'F' to decimal (10–15)
   sub al, 'A'          ; 'A' = 0, 'B' = 1, ..., 'F' = 5
   add al, 10           ; adjust to 10–15
   mov value, al
   ; Print newline
   mov ah, 9
   lea dx, newline
   int 21h
   ; Show message
   lea dx, msg
   mov ah, 9
   int 21h
```

```
    ; Convert value to ASCII and print
    ; value is 10–15, so we split into two digits if needed
    mov al, value
    cmp al, 10
    jl one_digit
    ; Print tens digit (1)
    mov dl, '1'
    mov ah, 2
    int 21h
    ; Print units digit
    mov al, value
    sub al, 10
    add al, '0'
    mov dl, al
    mov ah, 2
    int 21h
    jmp done
one_digit:
    ; Just print 0–9
    add al, '0'
    mov dl, al
    mov ah, 2
    int 21h
done:
    ; Exit
    mov ah, 4Ch
    int 21h
main endp
end main
```

OUTPUT:

```
C:\MASM\BIN>program3.exe
ENTER A HEX DIGIT: A
IN DECIMAL IT IS 10
```

11. Write *a* program to display *a* 10 x 10 solid box of asterisks.
***Hint:* declare *a* string in the data segment that specifies the box, and display it with INT 2lh, function 9h.**

ALGORITHM:

1. Initializes the data segment using mov ax, @data and mov ds, ax.
2. Defines a string line containing 10 asterisks followed by a newline (**********), stored in line db '**********', 13, 10, '$'.
3. Prints the string line 10 times using DOS interrupt INT 21h with function AH = 9:
4. The program repeatedly loads the address of line into the DX register and calls INT 21h to display it. This is done 10 times.
5. Exits the program using INT 21h with function AH = 4Ch

CODE:

```
.model small
.stack 100h
.data
   line db '**********', 13, 10, '$' ; One line of 10 asterisks + newline
.code
main proc
   mov ax, @data
   mov ds, ax
   mov ah, 9
   lea dx, line
   int 21h
   lea dx, line
   int 21h
   lea dx, line
   int 21h
   lea dx, line
   int 21h
   lea dx, line
   int 21h
   lea dx, line
   int 21h
   lea dx, line
   int 21h
   lea dx, line
   int 21h
   lea dx, line
   int 21h
   lea dx, line
   int 21h
   lea dx, line
   int 21h
   mov ah, 4Ch
   int 21h
main endp
end main
```

OUTPUT:

12. Write a program to (a) display"?", (b) read three initials, (c) display them in the middle of an 11 x 11 box of asterisks, and (d) beep the computer.

ALGORITHM:

1. Initializes the data segment using mov ax, @data and mov ds, ax.
2. Displays the prompt (?) to ask the user for input:
3. The string? is displayed using DOS interrupt INT 21h with function AH = 9.
4. Reads three initials from the user:
5. The program reads three characters (initials) using INT 21h with function AH = 1 and stores them in the initials array.
6. Prints a newline to separate the prompt and the box:
7. A newline (13, 10) is printed using DOS interrupt INT 21h with function AH = 9.
8. Prints the top part of the box:
9. The program prints a line of 11 asterisks (**********), followed by a newline, 5 times using INT 21h with function AH = 9.
10. Prints the middle line of the box with initials:
11. Prints the left padding (*****) using INT 21h with function AH = 9.
12. Then, it prints the three initials using INT 21h with function AH = 2 (printing each character individually).
13. Prints the right padding (***) followed by a newline using INT 21h with function AH = 9.
14. Prints the bottom part of the box:
15. Prints a line of 11 asterisks (**********), followed by a newline, 5 times using INT 21h with function AH = 9.
16. Beep the computer:
17. The program makes a beep sound (ASCII Bell, 07h) using INT 21h with function AH = 2.
18. Exits the program using INT 21h with function AH = 4Ch.

CODE:

```
.model small
.stack 100h
.data
   prompt      db '?', '$'
   newline     db 13,10,'$'
   starline    db '**********',13,10,'$' ; 11 stars + newline
   leftpad     db '*****', '$'          ; 5 asterisks before initials
   rightpad    db '***',13,10,'$'       ; 3 asterisks after initials + newline
   initials    db 3 dup(?)              ; to store the initials
.code
main proc
   mov ax, @data
```

```
    mov ds, ax
    ; (a) Display '?'
    mov ah, 9
    lea dx, prompt
    int 21h
    ; (b) Read 3 initials
    mov ah, 1
    int 21h
mov initials[0], al
    mov ah, 1
    int 21h
    mov initials[1], al
    mov ah, 1
    int 21h
    mov initials[2], al
    ; Newline before box
    mov ah, 9
    lea dx, newline
    int 21h
    ; (c) Print top 5 lines of asterisks (5 times)
    mov ah, 9
    lea dx, starline
    int 21h
    lea dx, starline
    int 21h
    lea dx, starline
    int 21h
    lea dx, starline
    int 21h
    lea dx, starline
    int 21h
    ; --- Middle line: *****XYZ***
    lea dx, leftpad
    mov ah, 9
    int 21h
; Print the 3 initials
    mov ah, 2
    mov dl, initials[0]
    int 21h
    mov dl, initials[1]
    int 21h
    mov dl, initials[2]
    int 21h
```

```
    ; Print the right pad
    lea dx, rightpad
    mov ah, 9
    int 21h
    ; Print bottom 5 lines of asterisks (5 times)
    lea dx, starline
    int 21h
    lea dx, starline
    int 21h
    lea dx, starline
    int 21h
    lea dx, starline
    int 21h
    lea dx, starline
    int 21h
    ; (d) Beep the computer (ASCII Bell = 07h)
    mov ah, 2
    mov dl, 7
    int 21h
  ; Exit
    mov ah, 4Ch
    int 21h
main endp
end main
```

OUTPUT

Chapter 6

FLOW CONTROL INSRUCTIONS

EXAMPLES

Example1: Program1 Listing PGM6_ 1.ASM?

TITLE PGM6 1: IBM CHARACTER DISPLAY

ALGORITHM:

1. Initializes AH = 2 to use DOS function for character output (INT 21h).
2. Sets CX = 256 to control the loop for 256 iterations.
3. Starts DL = 0 (the first ASCII character).
4. Loop (PRINT_LOOP):

- Calls INT 21h to print the character in DL.
- Increments DL to move to the next ASCII character.
- Decrements CX and continues until CX = 0.

5.Terminates the program with INT 21h, AH = 4Ch.

CODE:

```
.model small
.stack 100h
.code
main proc
mov ah,2
mov cx,256
mov dl,0
PRINT_LOOP:
        int 21h
        inc dl
        dec cx
        jnz PRINT_LOOP
mov ah,4ch
int 21h
main endp
end main
```

OUTPUT:

Roll No: 24BSCS 43

Example2: Program2 Listing PGM6_2.ASM?

TITLE PGM6 1: FIRST AND LAST CAPITALS

ALGORITHM:

1. Initialize data segment with mov ax, @data and mov ds, ax.
2. Display prompt: "Type a line of text:"
3. Reads characters one-by-one until Enter (carriage return, 0Dh) is pressed.
4. Checks if each character is a capital letter:
   - If between 'A' and 'Z', it's a capital letter.
   - The first capital found is saved in FIRST (if not already set).
   - The last capital is always updated with the current one.
5. After Enter is pressed:
   - If no capital letters were found (FIRST = 0), displays: "No capital letters found!"

CODE:

```
.model small
.stack 100h
.data
PROMPT DB 'Type a line of text: $'
NOCAP_MSG DB 0Dh, 0Ah, 'No capital letters found! $'
FIRST_MSG DB 0Dh, 0Ah, 'First Capital = $'
LAST_MSG DB 0Dh, 0Ah, 'Last Capital = $'
FIRST DB 0      ; Initialize FIRST to null (0)
LAST DB 0       ; Initialize LAST to null (0)
.code
main proc
   mov ax, @data
   mov ds, ax    ; Prompt user for input
   mov ah, 9
   lea dx, PROMPT
   int 21h    ; Read user input
   mov ah, 1       ; DOS function: Read character
WHILE_:
   int 21h        ; Get input character (AL contains the character)
   cmp al, 0Dh     ; Check if Enter key (Carriage Return) is pressed
   je END_WHILE    ; Exit loop if Enter is pressed
   ; Check if character is uppercase letter between 'A' and 'Z'
   cmp al, 'A'     ; Compare AL with 'A'
   jl CONTINUE     ; If AL < 'A', continue (not a capital letter)
   cmp al, 'Z'     ; Compare AL with 'Z'
```

```
    jg CONTINUE     ; If AL > 'Z', continue (not a capital letter)
    ; If the character is a capital letter, update FIRST and LAST
    ; Check and update FIRST (if first time found)
    cmp byte ptr FIRST, 0
    je UPDATE_FIRST ; If FIRST is still null, update with current letter
    jmp CONTINUE    ; If FIRST is already set, continue checking
UPDATE_FIRST:
    mov byte ptr FIRST, al
    jmp CONTINUE

CONTINUE:
    ; Check and update LAST (always update with most recent capital letter)
    mov byte ptr LAST, al
    jmp WHILE_
END_WHILE:
    ; Display the result
    mov ah, 9
    cmp byte ptr FIRST, 0  ; Check if no capital letter was found
    je NO_CAPITALS         ; If no capital letter, display "No capital letters found!"
    ; Display first capital
    lea dx, FIRST_MSG
    int 21h
mov dl, [FIRST]
    mov ah, 2
    int 21h
    ; Display last capital
    lea dx, LAST_MSG
    int 21h
    mov dl, [LAST]
    mov ah, 2
    int 21h
    jmp END_PROGRAM

NO_CAPITALS:
    lea dx, NOCAP_MSG
    int 21h
END_PROGRAM:
    mov ah, 4Ch    ; DOS function to terminate program
    int 21h
main endp
end main
```

OUTPUT:

```
C:\MASM\BIN>PGM6_2.exe
Type a line of text: haseeb memon is here

No capital letters found!
```

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

ALGORITHM:

1. Displays a? prompt
2. Reads two capital letters
3. Shows them on the next line in alphabetical order

CODE:

```
.model small
.stack 100h
.data
crlf db 13,10,'$'
.code
main proc
   mov ax, @data
   mov ds, ax
   ; Display '?'
   mov ah, 2
   mov dl, '?'
   int 21h
   ; Read first letter into AL, store in BL
   mov ah, 1
   int 21h
   mov bl, al
   ; Read second letter into AL, store in BH
   mov ah, 1
   int 21h
   mov bh, al
   ; Sort: if BL > BH, swap
   cmp bl, bh
   jbe skip_swap
   xchg bl, bh
skip_swap:
   ; New line
   mov ah, 9
   lea dx, crlf
   int 21h
   ; Print letters in order
   mov ah, 2
   mov dl, bl
   int 21h
```

```
 mov dl, bh
   int 21h
   ; Exit
   mov ah, 4Ch
   int 21h
main endp
end main
```

OUTPUT:

```
C:\MASM\BIN>chp6_.exe
?AB
AB
```

9. Write *a* program to display the extended ASCII characters (ASCHI codes 80h to FFh). Display 10 characters per line, separated by blanks. Stop after the extended characters have been displayed once

ALGORITHM:

1) Initialize the starting ASCII code to 80h.
2) Loop from 80h to FFh:
   - Print the character.
   - Print a space character.
   - Count how many characters printed on the current line.
   - If 10 characters printed, print a newline (carriage return + line feed).
3) Stop after reaching FFh.

CODE:

```
.model small
.stack 100h
.data
charCount db 0       ; Counter for 10 characters per line
.code
main proc
   mov ax, @data
   mov ds, ax
   mov bl, 80h       ; Start from ASCII code 80h
print_loop:
   ; Print character in BL
   mov ah, 02h       ; DOS function: Display character
   mov dl, bl
   int 21h
   ; Print space
   mov dl, ' '
```

```
    int 21h
    ; Update counter
    inc charCount
    cmp charCount, 10
    jne skip_newline
    ; Print newline (CR + LF)
    mov dl, 13      ; Carriage Return
    int 21h
    mov dl, 10      ; Line Feed
    int 21h
    mov charCount, 0  ; Reset counter
skip_newline:
    inc bl          ; Move to next character
    cmp bl, 0FFh
    jne print_loop
    ; Exit to DOS
    mov ah, 4Ch
    int 21h
main endp
end main
```

OUTPUT:

10. 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.i 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

ALGORITHM:

1) Start the program and prepare memory and variables.
2) Display a message asking the user to enter a hex digit.
3) Read the character input from the keyboard.
4) Check if it is valid:
   - If between '0' and '9', convert to decimal (subtract '0').
   - If between 'A' and 'F', convert to decimal (subtract 'A' and add 10).
   - If between 'a' and 'f', prompt again (not allowed unless uppercase).
   - If not valid, display an error and ask again. 5)Display the decimal value.
6) Ask the user if they want to repeat.
7) Check the response:
   - If Y or y, go back to step 2.
   - Otherwise, exit.

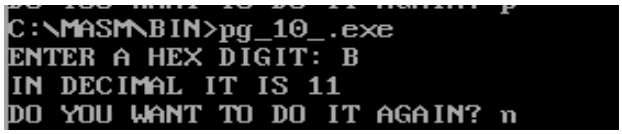
CODE:

```
.model small
.stack 100h
.data
msg_prompt db 'ENTER A HEX DIGIT: $'
msg_error  db 13,10,'ILLEGAL CHARACTER - ENTER 0 .. 9 OR A .. F: $'
msg_result db 13,10,'IN DECIMAL IT IS $'
msg_again  db 13,10,'DO YOU WANT TO DO IT AGAIN? $'
newline    db 13,10,'$'
.code
main:
   mov ax, @data
   mov ds, ax
start:
   ; Display prompt
   mov ah, 09h
   lea dx, msg_prompt
   int 21h
get_input:
; Read input character
   mov ah, 01h
   int 21h
   mov bl, al      ; Store input in BL
   mov ah, 0       ; Clear upper byte
   ; Validate input
   cmp bl, '0'
   jl invalid
   cmp bl, '9'
```

```
    jle is_digit
    cmp bl, 'A'
    jl invalid
    cmp bl, 'F'
    jle is_letter
invalid:
    ; Show error
    mov ah, 09h
    lea dx, msg_error
    int 21h
    jmp get_input
is_digit:
    sub bl, '0'     ; Convert ASCII '0'-'9' to number 0-9
    jmp display_result
is_letter:
    sub bl, 'A'
    add bl, 10      ; Convert 'A'-'F' to 10-15
display_result:
    ; Show result message
    mov ah, 09h
    lea dx, msg_result
    int 21h
    ; Convert value in BL to ASCII for output
    mov ax, 0
    mov al, bl
    aam             ; AH = tens, AL = ones
    add ax, 3030h   ; Convert to ASCII
    ; Display tens digit
    mov dl, ah
    mov ah, 02h
    int 21h
    ; Display ones digit
    mov dl, al
    mov ah, 02h
    int 21h
ask_again:
    ; Ask if user wants to do it again
    mov ah, 09h
    lea dx, msg_again
    int 21h
  ; Get response
    mov ah, 01h
    int 21h
```

```
    cmp al, 'y'
    je start
    cmp al, 'Y'
    je start
    ; Exit
    mov ah, 4Ch
    int 21h
end main
```

OUTPUT:



11. Do programming exercise 10, except that if the user fails to enter a hex-digit character In three tries, display a message and terminate the program.

ALGORITHM:

1)Start the program and set a retry counter to 0
2)Prompt user to input a hex digit 3)If valid:

- Reset retry counter
- Display decimal equivalent
- Ask to repeat 4)If invalid:
- Increase retry counter
- If less than 3: re-prompt
- If 3 invalid attempts: show error and exit

CODE:

```
.model small
.stack 100h
.data
msg_prompt db 'ENTER A HEX DIGIT: $'
msg_error  db 13,10,'ILLEGAL CHARACTER - ENTER 0 .. 9 OR A .. F: $'
msg_fail   db 13,10,'TOO MANY INVALID ATTEMPTS - TERMINATING.$'
msg_result db 13,10,'IN DECIMAL IT IS $'
msg_again  db 13,10,'DO YOU WANT TO DO IT AGAIN? $'
.code
main proc
    mov ax, @data
    mov ds, ax
```

```
start:
    mov cx, 0        ; Retry counter = 0
prompt_input:
    ; Prompt for input
    mov ah, 09h
    lea dx, msg_prompt
    int 21h
get_input:
    mov ah, 01h
    int 21h
    mov bl, al       ; Save input
    mov ah, 0        ; Clear upper byte
    ; Validate
    cmp bl, '0'
    jl invalid
    cmp bl, '9'
    jle is_digit
    cmp bl, 'A'
    jl invalid
    cmp bl, 'F'
    jle is_letter
invalid:
    inc cx
    cmp cx, 3
    je too_many_attempts
    mov ah, 09h
    lea dx, msg_error
    int 21h
    jmp prompt_input
is_digit:
    sub bl, '0'
    jmp display_result
is_letter:
    sub bl, 'A'
    add bl, 10
display_result:
    ; Reset retry counter for next round
    mov cx, 0
    mov ah, 09h
    lea dx, msg_result
    int 21h
    ; Convert to decimal ASCII
    mov ax, 0
```

```
    mov al, bl
    aam           ; AH = tens, AL = ones
    add ax, 3030h   ; Convert to ASCII
    ; Display digits
    mov dl, ah
    mov ah, 02h
    int 21h
    mov dl, al
    mov ah, 02h
    int 21h
ask_again:
    mov ah, 09h
    lea dx, msg_again
    int 21h
    mov ah, 01h
    int 21h
    cmp al, 'y'
    je start
    cmp al, 'Y'
    je start
    ; Exit
    mov ah, 4Ch
    int 21h
too_many_attempts:
    mov ah, 09h
    lea dx, msg_fail
    int 21h
    mov ah, 4Ch
    int 21h
main endp
end main
```

OUTPUT:

```
C:\MASM\BIN>pg_11_.exe
ENTER A HEX DIGIT: h
ILLEGAL CHARACTER - ENTER 0 .. 9 OR A .. F: ENTER A HEX DIGIT: u
ILLEGAL CHARACTER - ENTER 0 .. 9 OR A .. F: ENTER A HEX DIGIT: j
TOO MANY INVALID ATTEMPTS - TERMINATING.
C:\MASM\BIN>S_
```

12. (hard) Write a program that reads a string of capital letters, ending with a carriage return, and displays the longest sequence of consecutive alphabetically increasing capital letters read.

Sample exec11tio11:
ENTER A STKING OF CAPITAL LETTERS: FGHADEFGHC
THE LONGEST CONSECUTIVELY INCREASING STRING IS:
DEFGH

ALGORITHM:

1) Read input characters into a buffer until Enter (ASCII 13) is pressed.
2) While reading, store only capital letters (A–Z).
3) Loop through the buffer and:
   - Track current increasing sequence (current_seq)
   - If next letter > previous letter, extend the current sequence •Else, compare length with max_seq, and update if current is longer 4)After reaching the end:
   - Compare final sequence one more time (in case it's the longest)

5)Display the longest increasing sequence

CODE:

```
.model small
.stack 100h
 .data
   prompt_1  db  'enter a string of capital letters : $'
   prompt_2  db  0dh,0ah,'the longest consecutive increasing string is : $'
   invalid   db  0dh,0ah,'invalid string of capital letters. try again : $'
.code
   main proc
     mov ax, @data              ; initialize ds
     mov ds, ax
     lea dx, prompt_1           ; load and display the string prompt_1
     mov ah, 9
     int 21h
     jmp @start                ; jump to label @start
     @try_again:                ; jump label
     lea dx, invalid           ; load and display the string invalid
     mov ah, 9
     int 21h
     @start:                   ; jump label
```

```
    mov ah, 1                ; set input function
    int 21h                  ; read a character
    cmp al, 0dh              ; compare al with cr
    je @try_again            ; jump to label @try_again if al=cr
    cmp al, 41h              ; compare al with 41h
    jb @try_again            ; jump to label @try_again if al<41h
    cmp al, 5ah              ; comapre al with 5ah
ja @try_again                ; jump to label @try_again if al>5ah
    mov bl, al               ; set bl=al
    mov bh, al               ; set bh=al
    mov dh, al               ; set dh=al
    mov dl, 1                ; set dl=1
    mov cl, 1                ; set cl=1
    @input:                  ; loop label
     int 21h                 ; read a character
     cmp al, 0dh             ; compare al with cr
     je @end_input           ; jump to label @end_input if al=cr
     cmp al, 41h             ; compare al with 41h
     jb @try_again           ; jump to label @try_again if al<41h
    cmp al, 5ah              ; comapre al with 5ah
     ja @try_again           ; jump to label @try_again if al>5ah
     inc bl                  ; set bl=bl+1
     cmp al, bl              ; compare al with bl
     jne @check_and_replace  ; jump to label @check_and_replace if al!=bl
     inc cl                  ; set cl=cl+1
     jmp @input              ; jump to label @input
     @check_and_replace:     ; jump label
     cmp cl, dl              ; compare cl with dl
     jle @skip_updation_1    ; jump to label @skip_updation_1 if cl<=dl
     mov dh, bh              ; set dh=bh
     mov dl, cl              ; set dl=cl
     @skip_updation_1:       ; jump label
     mov bh, al              ; set bh=al
     mov bl, al              ; set bl=al
     mov cl, 1               ; set cl=1
    jmp @input               ; jump to label @input
    @end_input:              ; jump label
    cmp cl, dl               ; compare cl with dl
    jle @skip_updation_2     ; jump to label @skip_updation_2 if cl<=dl
    mov dh, bh               ; set dh=bh
    mov dl, cl               ; set dl=cl
    @skip_updation_2:        ; jump label
    mov bx, dx               ; set bx=dx
```

```
    lea dx, prompt_2          ; load and display the string prompt_2
    mov ah, 9
    int 21h
    xor cx, cx                ; clear cx
    mov cl, bl                ; set cl=bl
    mov dl, bh                ; set dl=bh
    mov ah, 2                 ; set output function
    @output:                  ; loop label
      int 21h                 ; print a character
      inc dl                  ; set dl=dl+1
    loop @output              ; jump to label @output if cx!=0
    mov ah, 4ch               ; return control to dos
    int 21h
  main endp
end main
```

OUTPUT:

```
C:\MASM\BIN>pg_12_.exe
Enter a string of Capital Letters : FGHADEFGHC

The longest consecutive increasing string is : DEFGH
```

CHAPTER: 7

LOGIC, SHIFT & ROTATE INSTRUCTIONS

EXERCISE

8. Write a program that prompts the user to enter a character, and on subsequent lines prints its ASCII code in binary, and the number of 1 bits In Its ASCII code.

Sample execution:
TYPE A CHARACTER: A
THE ASCII CODE OF A IN BINARY IS 01000001
THE NUMBER OF l BITS IS 2

ALGORITHM:

1. Read Input Until Enter (ASCII 13)
   - Continuously read characters from the user until Enter is pressed.
   - Only store capital letters (A–Z) in the buffer.
2. Track Longest Increasing Sequence
   - Loop through the buffer:
   - Track the current increasing sequence.
   - If the next letter is greater than the previous, extend the sequence.
   - If not, compare the current sequence with the longest sequence found so far and update if necessary.
3. Final Comparison • After the loop, compare the last sequence with the longest to ensure it's recorded correctly.
4. Display the Longest Sequence
   - Print the longest increasing sequence found in the buffer.

CODE:

```
.model small
.stack 100h
.data
msg1 db 'TYPE A CHARACTER: $'
msg2 db 0Dh,0Ah, 'THE ASCII CODE OF ', 0
msg3 db 0Dh,0Ah, 'THE ASCII CODE OF CHARACTER IN BINARY IS $'
msg4 db 0Dh,0Ah, 'THE NUMBER OF 1 BITS IS $'
binary db 8 dup('0'), '$'
char db ?
digit db ?
.code
```

```
main proc
mov ax, @data
mov ds, ax
mov dx, offset msg1
mov ah, 09h
int 21h
mov ah, 01h
int 21h
mov char, al
mov bl, al
mov bh, 0
mov cx, 8
mov si, offset binary
next_bit:
mov dl, bl
and dl, 80h
cmp dl, 0
je store_zero
mov al, '1'
mov [si], al
inc bh
jmp continue
store_zero:
mov al, '0'
mov [si], al
continue:
inc si
shl bl, 1
loop next_bit
mov dx, offset msg2
mov ah, 09h
int 21h
mov al, char
mov dl, al
mov ah, 02h
int 21h
mov dx, offset msg3
mov ah, 09h
int 21h
mov dx, offset binary
mov ah, 09h
int 21h
mov dx, offset msg4
```

```
mov ah, 09h
int 21h
mov al, bh
add al, '0'
mov dl, al
mov ah, 02h
int 21h
mov ah, 4Ch
int 21h
main endp
end main
```

OUTPUT:

```
C:\MASM\BIN>chp7_.exe
TYPE A CHARACTER: H
THE ASCII CODE OF
THE ASCII CODE OF CHARACTER IN BINARY IS H
THE ASCII CODE OF CHARACTER IN BINARY IS 01001000
THE NUMBER OF 1 BITS IS 2
```

9. Write a program that prompts the user to enter a character and prints the ASCII code of the character in hex on the next line. Repeat this process until the user types a carriage return.

Sample execution:
TYPE A CHARACTER: Z
THE ASCII CODE OF Z IN HEX IS SA TYPE A
CHARACTER:

ALGORITHM:

1) Prompt the User for Input
   - Display the message: "TYPE A CHARACTER: " to the user.
2) Read the Input
   - Accept a single character input from the user.
   - If the character is a carriage return (ASCII 13), exit the program.
3) Convert the Character to its ASCII Code in Hex
   - Convert the ASCII code of the entered character into its hexadecimal representation.
4) Display the ASCII Code in Hex
   - Print the ASCII code of the character in hexadecimal format.
5) Repeat
   - Repeat the process until the user presses Enter (ASCII 13), which will terminate the loop.

CODE:

```
.model small
.stack 100h
.data
   prompt db 'TYPE A CHARACTER: $'
   result db 'THE ASCII CODE OF ? IN HEX IS $'
.code
main:
   mov ax, @data
   mov ds, ax
read_char:
   lea dx, prompt
   mov ah, 09h
   int 21h
   mov ah, 01h
   int 21h
   cmp al, 0Dh
   je exit_program
   push ax
   mov dl, 0Dh
   mov ah, 02h
   int 21h
   mov dl, 0Ah
   int 21h
   lea dx, result
   mov ah, 09h
   int 21h
   pop ax
push ax
   mov bx, offset result + 21
   mov [bx], al
   lea dx, result
   mov ah, 09h
   int 21h
   pop ax
   mov bl, al
   mov cl, 4
   shr al, cl
   call print_hex_digit
   mov al, bl
   and al, 0Fh
```

```
    call print_hex_digit
    mov dl, 0Dh
    mov ah, 02h
    int 21h
    mov dl, 0Ah
    int 21h
    jmp read_char
    mov ah, 4Ch
    int 21h
print_hex_digit:
    cmp al, 0Ah
    jb .is_digit
    add al, 7
.is_digit:
    add al, '0'
    mov dl, al
    mov ah, 02h
    int 21h
    ret
end main
```

OUTPUT:

```
C:\MASM\BIN>hassi_9.exe
TYPE A CHARACTER: h
THE ASCII CODE OF ? IN HEX IS THE ASCII CODE OF ? Ih HEX IS 68
TYPE A CHARACTER: A
THE ASCII CODE OF ? Ih HEX IS THE ASCII CODE OF ? IA HEX IS 41
TYPE A CHARACTER: S
THE ASCII CODE OF ? IA HEX IS THE ASCII CODE OF ? IS HEX IS 53
TYPE A CHARACTER: E
THE ASCII CODE OF ? IS HEX IS THE ASCII CODE OF ? IE HEX IS 45
TYPE A CHARACTER: E
THE ASCII CODE OF ? IE HEX IS THE ASCII CODE OF ? IE HEX IS 45
TYPE A CHARACTER: B
THE ASCII CODE OF ? IE HEX IS THE ASCII CODE OF ? IB HEX IS 42
TYPE A CHARACTER:

C:\MASM\BIN>S
```

12. Write a program that prompts the user to enter two.binary numbers of up to 8 digits each, and prints their sum on the next line in binary. If the user enters an illegal character, he or she should be prompted to begin again. Each input ends with a carriage return.
Sample execution:

TYPE 'A BINARY NUMBER, UP TO 8 DIGITS: 11001010

TYPE 'A BINARY NUMBER, UP TO 8 DIGITS: 10011100

THE BINARY SUM I~ 101100110

ALGORITHM:

1. Prompt User for First Binary Number The program displays:
   TYPE A BINARY NUMBER, UP TO 8 DIGITS:

   It waits for the user to type a binary number (up to 8 characters, using only '0' and

   '1').

   If the user enters any illegal character (like '2', 'A', or space), it prints an error and restarts.
2. Read and Store First Number
   Valid characters are stored in a buffer (b1) for processing.
3. Prompt User for Second Binary Number
   The program repeats the same input and validation process for a second binary number and stores it in another buffer (b2).
4. Convert and Add the Two Binary Numbers
5. The program starts from the rightmost digit of each buffer.
6. It adds corresponding bits from both numbers plus a carry bit.
7. It stores each result bit into the result buffer.
8. If there's a carry left after the last bit, it is added as the leftmost bit of the result.
   9.Remove Leading Zeros from the Result
10. The program skips all leading '0' characters in the result string.
11. If the result is all zeros, it still prints at least one '0'.
12. Print the Final Binary Sum The program prints:
    THE BINARY SUM IS 101100110

    (based on the example input 11001010 + 10011100)
13. Repeat or Exit
    After displaying the result, the program ends or could be modified to prompt for new inputs again.

CODE:

```
.model small
.stack 100h
.data
m1 db 'TYPE A BINARY NUMBER, UP TO 8 DIGITS: $'
m2 db 0Dh,0Ah,'THE BINARY SUM IS $'
m3 db 0Dh,0Ah,'ILLEGAL CHARACTER, TRY AGAIN$'
b1 db 9 dup(0)
b2 db 9 dup(0)
res db 9 dup('0'), '$'
.code
main:
   mov ax, @data
mov ds, ax
read1:
   lea dx, m1
   mov ah, 09h
   int 21h
   lea si, b1
   call readbin
   jc read1
read2:
   lea dx, m1
   mov ah, 09h
   int 21h
   lea si, b2
   call readbin
   jc read2
   call addbin
   lea dx, m2
   mov ah, 09h
   int 21h
   lea dx, res
   mov ah, 09h
   int 21h
   mov ah, 4Ch
   int 21h
readbin:
   xor cx, cx
rloop:
   mov ah, 01h
   int 21h
   cmp al, 13
```

```
    je done
    cmp al, '0'
    je store
    cmp al, '1'
    je store
    lea dx, m3
    mov ah, 09h
    int 21h
    stc
    ret
store:
    cmp cx, 8
    jae skip
    mov [si], al
    inc si
    inc cx
skip:
    jmp rloop
done:
    mov al, 0
    mov [si], al
  clc
    ret
addbin:
    lea si, b1
lea di, b2
    lea bx, res
    mov cx, 8
    xor dx, dx
    addloop:
        dec si
        dec di
        dec bx
        mov al, [si]
        mov bl, [di]
        sub al, '0'
        sub bl, '0'
        add al, bl
        add al, dl
        cmp al, 2
        jb no_carry
        sub al, 2
        mov dl, 1
```

```
        jmp storebit
    no_carry:
        mov dl, 0
    storebit:
        add al, '0'
        mov [bx], al
        loop addloop
    cmp dl, 1
    jne endadd
    dec bx
    mov byte ptr [bx], '1'
endadd:
    lea si, res
    lea di, res
    mov cx, 9
skipzero:
    mov al, [si]
    cmp al, '1'
    je doneprint
    inc si
    loop skipzero
doneprint:
    mov dx, si
    ret
end main
```

OUTPUT:

```
C:\MASM\BIN>new_one.exe
TYPE A BINARY NUMBER, UP TO 8 DIGITS: 10000000
TYPE A BINARY NUMBER, UP TO 8 DIGITS: 10000000

THE BINARY SUM IS 000000000
```

CHAPTER: 8

STACK & INTRODUCTION TO PROCEDURES

EXAMPLE

Example1: Program1 Listing PGM8_ 1.ASM?

TITLE PGM8_1: REVERSE THE GIVEN INPUT

ALGORITHM:

This assembly program performs the following actions:

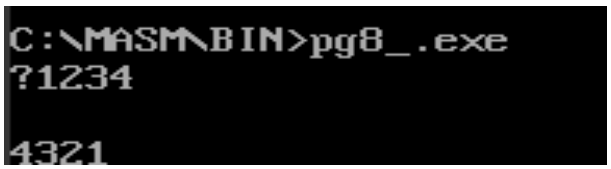
1) Display Prompt Character
   Displays a single? on screen to prompt the user for input.
2) Initialize Character Counter
   Clears CX to use it as a counter for how many characters are typed.
3) Read Characters from User
   Uses INT 21h, AH=01h to read one character at a time with echo.
   The loop continues until the Enter key (ASCII 13 / 0Dh) is pressed.
4) Push Each Character to Stack
   Each character typed (except Enter) is pushed to the stack, and CX is incremented.
5) End of Input
   Once Enter is pressed, the program prints a newline to move to the next line.
6) Check if Any Characters Were Entered
   If no characters were entered (CX = 0), the program exits immediately.
7) Pop and Print Characters in Reverse
   If characters were entered, they are popped from the stack one by one and printed, effectively displaying the input in reverse order.
8) Terminate Program
   Uses INT 21h, AH=4Ch to return control to DOS and end the program.

CODE:

```
.model small
.stack 100h
.code
main proc
   mov ah,2
   mov dl,'?'
   int 21h
   xor cx,cx
   mov ah,1
```

```
    int 21h
WHILE_:
    cmp al, 0Dh
    je END_WHILE
    push ax
inc cx
    mov ah,1
    int 21h
    jmp WHILE_
END_WHILE:
    mov ah,2
    mov dl,0Dh
    int 21h
    mov dl,0Ah
    int 21h
    jcxz EXIT
TOP:
    pop dx
    mov ah,2
    int 21h
    loop TOP
EXIT:
    mov ah,4ch
    int 21h
main endp
end main
```

OUTPUT:



Example2: Program2 Listing PGM8_ 2.ASM?

TITLE PGM8_2: MULTIPLICATION OF TWO NUMBERS

ALGORITHM:

1) Initialize Data Segment
    a. mov ax, @data and mov ds, ax set up the data segment so strings and buffers can be accessed.
2) Assign Numbers to Multiply
    a. mov ax, 2 and mov bx, 2: set the two numbers to multiply.
    b. These values can be changed as needed.
3) Call MULTIPLY Routine
    a. Performs multiplication of AX × BX using bitwise shift-and-add (manual binary multiply).
    b. The result is stored in DX.
4) Print Message
    a. Displays "RESULT: " using INT 21h with DOS function 09h.
5) Move Result to AX for Printing
    a. Since DX holds the result, it's copied to AX to convert and print.
6) Call PRINT_DECIMAL Routine
    a. Converts the binary number in AX to decimal digits.
    b. Stores ASCII characters of each digit in a buffer (digits) by repeated division by 10.
7) Print Decimal Digits
    a. Uses INT 21h, function 02h to print one character at a time from the buffer.
    b. Only prints the non-zero digits from the result.
8) Print Newline
    a. Moves cursor to the next line after the result using DOS newline (0Dh, 0Ah).
9) Terminate Program
    a. Exits cleanly using INT 21h, function 4Ch.

CODE:

```
.model small
.stack 100h
.data
msg db 'RESULT: $'
num db 6 dup(0)
.code
main proc
   mov ax, @data
   mov ds, ax
   mov ax, 0002h
```

```
    mov bx, 0002h
    call MULTIPLY
    lea dx, msg
   mov ah, 09h
    int 21h
    mov ax, dx
    lea di, num + 5
    mov cx, 0
convert:
    xor dx, dx
    mov bx, 10
    div bx
    add dl, '0'
    dec di
    mov [di], dl
    inc cx
    cmp ax, 0
    jne convert
    mov ah, 02h
print_loop:
    mov dl, [di]
    int 21h
    inc di
    loop print_loop
    mov ah, 4Ch
    int 21h
main endp
MULTIPLY proc
    push ax
    push bx
    xor dx, dx
REPEAT_LABEL:
    test bx, 1
    jz SKIP_ADD
    add dx, ax
SKIP_ADD:
    shl ax, 1
    shr bx, 1
    jnz REPEAT_LABEL
    pop bx
    pop ax
    ret
MULTIPLY endp
end main
```

OUTPUT:

```
C:\MASM\BIN>chp8.exe
RESULT: 4
```

EXERCISE: CHAPTER: 8

8. Writ; a program that lets the user type some text, consisting of words separated by blanks, ending with a carriage return, and dis plays the text in the same word order as entered, but with the let __, ters ·in each word reversed. For example, "this is a test" becomes "siht si a tset". Hint: modify program PGM8_2.ASM in section 8.3.

```
.MODEL SMALL
.STACK 100H
.DATA
   prompt DB 'Enter text (words separated by spaces): $'
   output DB 0DH, 0AH, 'Reversed words: $'
   buffer DB 100, ?, 100 DUP('$')  ; Input buffer
   wordBuffer DB 100 DUP(0)        ; Temporary word buffer

.CODE
MAIN PROC
   MOV AX, @DATA
   MOV DS, AX
   MOV ES, AX

   ; Display prompt
   MOV AH, 09H
   LEA DX, prompt
   INT 21H

   ; Read input string
   MOV AH, 0AH
   LEA DX, buffer
   INT 21H

   ; Display output label
   MOV AH, 09H
   LEA DX, output
   INT 21H

   ; Prepare to process input
   MOV SI, OFFSET buffer + 2  ; Start of actual input
   MOV DI, OFFSET wordBuffer  ; Temporary word buffer
   XOR CX, CX                 ; Word length counter
```

```
PROCESS_LOOP:
   MOV AL, [SI]       ; Get current character
   CMP AL, 0DH        ; Check for carriage return
   JE END_OF_INPUT    ; If found, we're done
   CMP AL, ' '        ; Check for space
   JE REVERSE_WORD    ; If space, reverse current word

   ; Store character in word buffer
   MOV [DI], AL
   INC DI
   INC CX             ; Increment word length
   INC SI
   JMP PROCESS_LOOP

REVERSE_WORD:
   ; Reverse and print the current word
   CALL REVERSE_AND_PRINT

   ; Print the space
   MOV DL, ' '
   MOV AH, 02H
   INT 21H

   ; Reset for next word
   MOV DI, OFFSET wordBuffer
   XOR CX, CX
   INC SI             ; Move past the space
   JMP PROCESS_LOOP

END_OF_INPUT:
   ; Reverse and print the last word if it exists
   CMP CX, 0
   JE EXIT_PROGRAM
   CALL REVERSE_AND_PRINT

EXIT_PROGRAM:
   ; Exit program
   MOV AH, 4CH
   INT 21H
MAIN ENDP

; Subroutine to reverse and print the current word
REVERSE_AND_PRINT PROC
```

```
    PUSH AX
    PUSH BX
    PUSH CX
    PUSH SI
    PUSH DI

    ; Check if we have a word to reverse
    CMP CX, 0
    JE END_REVERSE

    ; Point DI to end of word buffer
    MOV DI, OFFSET wordBuffer
    ADD DI, CX
    DEC DI          ; DI now points to last character

    ; Point SI to start of word buffer
    MOV SI, OFFSET wordBuffer

REVERSE_LOOP:
    CMP SI, DI
    JAE PRINT_WORD

    ; Swap characters
    MOV AL, [SI]
    MOV BL, [DI]
    MOV [SI], BL
    MOV [DI], AL

    ; Move pointers
    INC SI
    DEC DI
    JMP REVERSE_LOOP

PRINT_WORD:
    ; Print the reversed word
    MOV SI, OFFSET wordBuffer
    MOV CX, 0       ; Reset counter for STOSB

PRINT_LOOP:
    MOV AL, [SI]
    CMP AL, 0       ; Check for null terminator
    JE END_REVERSE
    MOV DL, AL
```

```
    MOV AH, 02H
    INT 21H
    INC SI
    JMP PRINT_LOOP

END_REVERSE:
    ; Clear word buffer
    MOV DI, OFFSET wordBuffer
    MOV CX, 100
    MOV AL, 0
    REP STOSB

    POP DI
    POP SI
    POP CX
    POP BX
    POP AX
    RET
REVERSE_AND_PRINT ENDP
END MAIN
```

OUTPUT:

```
D:\>TLINK D:\test
Turbo Link  Version 7.1.30.1. Copyright (c) 1987, 1996 Borland International

D:\>D:\test
Enter text (words separated by spaces): Abdul Hasseb Memon
Reversed words: ludbA bessaH nomeM
```

9.A problem in elementary algebra is to decide if an expression containing several kinds of brackets, such as [,], {, }, (,), is correctly bracketed. This is the case if (a) there are the same number of left and right brackets of each kind, and (b) when a right bracket appears, the most recent preceding unmatched left bracket should be of the same type. For example,
(a + [b - {c x (d - e)}] + f) is correctly bracketed, but
(a + (b - [c x (d - e)) + f] is not.
Correct bracketing can be decided by using a stack. The expression is scanned left to right. When a left bracket is encountered,
it is pushed onto the stack. When a right bracket is encountered,
the stack is popped (if the stack is empty, there are too many right brackets) and the brackets are compared. If they are of the same type, the scanning continues. If there is a mismatch, the expression is incorrectly bracketed. At the end of the expression, if the stack is empty, the expression is correctly bracketed. If the stack is not empty, there are too many left brackets.
Write a program that lets the user type in an algebraic expression, ending with a carriage return, that contains round (parentheses), square, and curly brackets. As the expression is being typed in, the program evaluates each character. If at any point the expression is incorrectly bracketed (too many right brackets or a mis-match between left and right brackets), the program tells the user to start over. After the carriage return is typed, if the expression is correct, the program displays "EXPRESSION IS CORRECT." If not, the program displays "TOO MANY LEFT BRACKETS." In both cases, the pro-gram asks the user if he or she wants to continue. If the user types 'Y', the program runs again.
Your program does not need to store the input string, only check it for correctness.

ENTER AN ALGEBRAIC EXPRESSION:
(a + b]
TOO MANY RIGHT BRACKETS. BEGIN AGAIN!
ENTER AN ALGEBRAIC EXPRESSION:
(a + [b +c] - d)
EXPRESSION IS CORRECT
TYPE Y IF YOU WANT TO CONTINUE: Y
ENTER AN ALGEBRAIC EXPRESSION:
{a + b x ([- c]) E- P)

BRACKET MISMATCH. BEGIN AGAIN!
ENTER AN ALGEBRAIC EXPRESSION:
((a + [b - {c + (d - e)}] + f)
TOO MANY LEFT BRACKETS. BEGIN AGAIN!
ENTER AN ALGEBRAIC EXPRESSION:
I'VE HAD ENOUGH
EXPRESSION IS CORRECT
TYPE Y IF YOU WANT TO CONTINUE: N

Solution:

```
.MODEL SMALL
.STACK 100h
.DATA
prompt1 DB "ENTER AN ALGEBRAIC EXPRESSION:$"
msg_right DB 13,10,"TOO MANY RIGHT BRACKETS. BEGIN AGAIN!",13,10,"$"
msg_mismatch DB 13,10,"BRACKET MISMATCH. BEGIN AGAIN!",13,10,"$"
msg_left DB 13,10,"TOO MANY LEFT BRACKETS. BEGIN AGAIN!",13,10,"$"
msg_correct DB 13,10,"EXPRESSION IS CORRECT",13,10,"$"
msg_continue DB "TYPE Y IF YOU WANT TO CONTINUE:$"
bracket_stack DB 50 DUP(?)
top DB 0
input_char DB ?
.CODE
START:
MOV AX, @DATA
MOV DS, AX
MAIN_LOOP:
CALL CLEAR_STACK
LEA DX, prompt1
CALL PRINT_STRING
READ_LOOP:
CALL READ_CHAR
CMP AL, 13
JE CHECK_END
MOV input_char, AL
CMP AL, '('
JE PUSH_BRACKET
CMP AL, '['
JE PUSH_BRACKET
CMP AL, '{'
JE PUSH_BRACKET
CMP AL, ')'
JE CHECK_RIGHT
CMP AL, ']'
JE CHECK_RIGHT
CMP AL, '}'
JE CHECK_RIGHT
JMP READ_LOOP
```

```
PUSH_BRACKET:
MOV BL, top
MOV AL, input_char
MOV bracket_stack[BX], AL
INC top
JMP READ_LOOP
CHECK_RIGHT:
CMP top, 0
JE TOO_MANY_RIGHT
DEC top
MOV BL, top
MOV AL, bracket_stack[BX]
MOV DL, input_char
CALL MATCH_BRACKETS
CMP AL, 0
JE MISMATCH
JMP READ_LOOP
CHECK_END:
CMP top, 0
JNE TOO_MANY_LEFT
LEA DX, msg_correct
CALL PRINT_STRING
JMP ASK_CONTINUE
TOO_MANY_RIGHT:
LEA DX, msg_right
CALL PRINT_STRING
JMP MAIN_LOOP
TOO_MANY_LEFT:
LEA DX, msg_left
CALL PRINT_STRING
JMP MAIN_LOOP
MISMATCH:
LEA DX, msg_mismatch
CALL PRINT_STRING
JMP MAIN_LOOP
ASK_CONTINUE:
LEA DX, msg_continue
CALL PRINT_STRING
CALL READ_CHAR
CMP AL, 'Y'
JNE EXIT_PROGRAM
JMP MAIN_LOOP
EXIT_PROGRAM:
MOV AH, 4CH
INT 21H
PRINT_STRING PROC
MOV AH, 09H
INT 21H
RET
PRINT_STRING ENDP
```

```
READ_CHAR PROC
MOV AH, 01H
INT 21H
RET
READ_CHAR ENDP
CLEAR_STACK PROC
MOV top, 0
RET
CLEAR_STACK ENDP
MATCH_BRACKETS PROC
CMP AL, '('
JE CMP_PAREN
CMP AL, '['
JE CMP_SQUARE
CMP AL, '{'
JE CMP_CURLY
MOV AL, 0
RET
CMP_PAREN:
CMP DL, ')'
JNE NO_MATCH
MOV AL, 1
RET
CMP_SQUARE:
CMP DL, ']'
JNE NO_MATCH
MOV AL, 1
RET
CMP_CURLY:
CMP DL, '}'
JNE NO_MATCH
MOV AL, 1
RET
NO_MATCH:
MOV AL, 0
RET
MATCH_BRACKETS ENDP
END START
```

OUTPUT:

```
):\>D:\test
ENTER AN ALGEBRAIC EXPRESSION:(a+b)+c

EXPRESSION IS CORRECT
TYPE Y IF YOU WANT TO CONTINUE:YENTER AN ALGEBRAIC EXPRESSION:(a+b))
TOO MANY RIGHT BRACKETS. BEGIN AGAIN!
ENTER AN ALGEBRAIC EXPRESSION:_
```

10. The following method can be used to generate random numbers in the range 1 to 32767. Start with any number in this range. Shift left once. Replace bit 0 by the XOR of bits 14 and 15. Clear bit 15. Write the following procedures: a. A procedure READ that ll'r in AX. .-\ proct>dure WRITE that displays AX in binary. You may use lhl' algorithm given in section 7.4. Write a program that displays a '?', calls READ to read a binary numl>cr, and calls RANDOM and WRITE to compute and display 100 random numbers. The numbers should be displayed four per line, with four blanks scparatlng the numbers

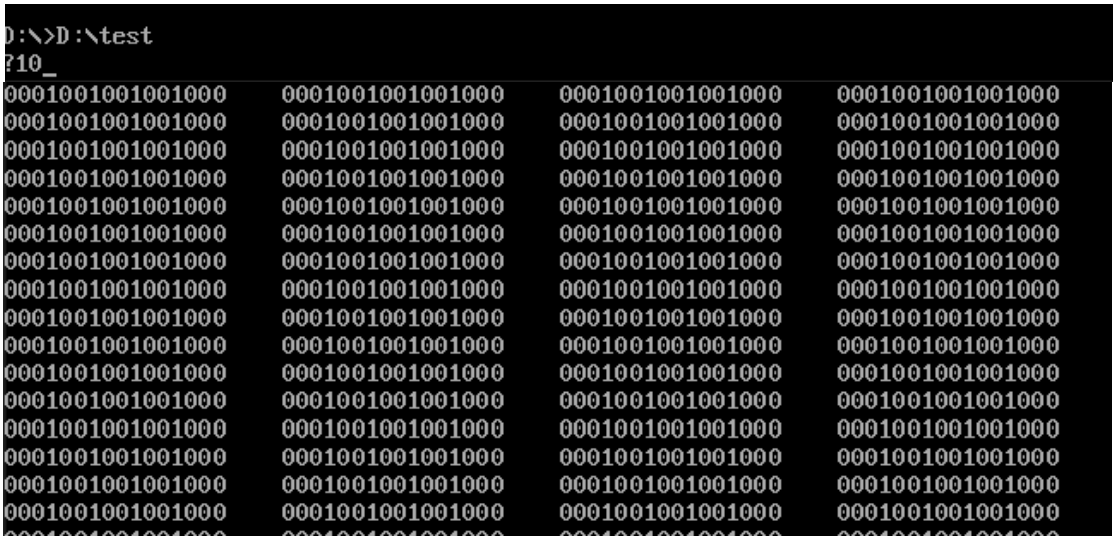
Solution:

```
.MODEL SMALL
.STACK 100H
.DATA
msg_prompt DB '?', '$'
msg_newline DB 13, 10, '$'
space4 DB ' ', '$'
bin_input DB 17 DUP('$') ; room for 16 bits + null
counter DB 0
.CODE
MAIN:
MOV AX, @DATA
MOV DS, AX
; Print '?'
LEA DX, msg_prompt
MOV AH, 09H
INT 21H
; Read input binary number
CALL READ
; Generate and print 100 random numbers
MOV CX, 100 ; loop counter
MOV BL, 0 ; per-line counter
NEXT_RANDOM:
CALL RANDOM
CALL WRITE
INC BL
CMP BL, 4
JL PRINT_SPACES
; Newline after 4 numbers
LEA DX, msg_newline
MOV AH, 09H
INT 21H
MOV BL, 0
JMP CONTINUE_LOOP
```

```
PRINT_SPACES:
LEA DX, space4
MOV AH, 09H
INT 21H
CONTINUE_LOOP:
LOOP NEXT_RANDOM
; Exit program
MOV AH, 4CH
INT 21H
; ----------------------------
; READ: Reads a 16-bit binary number from user and stores in AX
; ----------------------------
READ PROC
XOR AX, AX
XOR CX, CX
XOR BX, BX
READ_LOOP:
MOV AH, 01H ; read char
INT 21H
CMP AL, 13 ; Enter key?
JE DONE_READ
CMP AL, '0'
JE ADD_ZERO
CMP AL, '1'
JE ADD_ONE
JMP READ_LOOP ; ignore invalid input
ADD_ZERO:
SHL AX, 1
JMP READ_LOOP
ADD_ONE:
SHL AX, 1
OR AX, 1
JMP READ_LOOP
DONE_READ:
RET
READ ENDP
; ----------------------------
; RANDOM: Generates next pseudo-random number in AX
; ----------------------------
RANDOM PROC
PUSH CX
PUSH DX
; Get bits 14 and 15
MOV CX, AX
```

```
SHR CX, 14 ; bits 14 and 15 now in bit 0 and 1
AND CX, 00000011B
MOV DL, CL
SHR DL, 1
XOR CL, DL ; XOR bit 0 and 1 of CX => result in CL bit 0
SHL AX, 1 ; shift left
AND AX, 7FFFH ; clear bit 15
OR AL, CL ; set bit 0 to XOR result
POP DX
POP CX
RET
RANDOM ENDP
; ----------------------------
; WRITE: Displays binary value in AX (16-bit)
; ----------------------------
WRITE PROC
PUSH AX
PUSH CX
PUSH DX
PUSH BX ; save used register
MOV CX, 16
MOV BX, AX ; copy AX to BX, so AX is not destroyed
WRITE_LOOP:
SHL BX, 1
JC PRINT_ONE
MOV DL, '0'
JMP PRINT_CHAR
PRINT_ONE:
MOV DL, '1'
PRINT_CHAR:
MOV AH, 02H
INT 21H
LOOP WRITE_LOOP
POP BX
POP DX
POP CX
POP AX
RET
WRITE ENDP
END MAIN
```

OUTPUT:


D:\>D:\test
?10_
0001001001001000 0001001001001000 0001001001001000 0001001001001000