ASSIGNMENT

CT063-3-2

COMPUTER ARCHITECTURE

APU3F2209EEE-CE-TE APD3F2209-CE-EEE-TE

HAND OUT DATE: 28 SEPTEMBER 2022

HAND IN DATE: 08 DECEMBER 2022

WEIGHTAGE: 60%

Name	**CHARRAN TRIDANDAPANI**
TP NO	**TP062467**
Lecturer Name	**TS. UMAPATHY EAGANATHAN**
Hand out date	**28th September**
Hand in date	**20 th December**

Table of Contents

PART-1 CODE

INTRODUCTION

Assembly is an example of a low-level language; it's essentially just machine code. In most cases, the instructions must be converted to machine code before the computer may act on them. Mnemonics are used in assembly language to convey information and instructions for a specific task. Specific computer-based activity the assembly language's symbolic programming is much simpler to pick up and use than machine code. It's also simpler to fix bugs and make changes to the code. When you need to make optimal use of your CPU, such as when creating an OS, a game, or a control application, assembly programming is a great choice. Unfortunately, due to the machine-dependency of assembly language, the source code created for one computer may not work on other systems with different hardware configurations.

Files written in assembly language may be saved as.asm in any text editor. To compile the source code into machine language, an assembler is needed. Borland Turbo Assembler (TASM) is the most popular assembler. After an assembly programme has been compiled into an object file, a linker will join the two into a single executable (.exe). To debugging, a debugger provides means of tracking the execution of a programme in memory.

Assembly language is avoided nowadays because:

1. Development time: Writing in assembly language takes longer.

2. Debugging and verifying: More mistakes make debugging harder.

Assembly code is platform specific. Using it between platforms is difficult.

4. Maintainability: Unstructured spaghetti code makes assembly code harder to adapt and maintain.

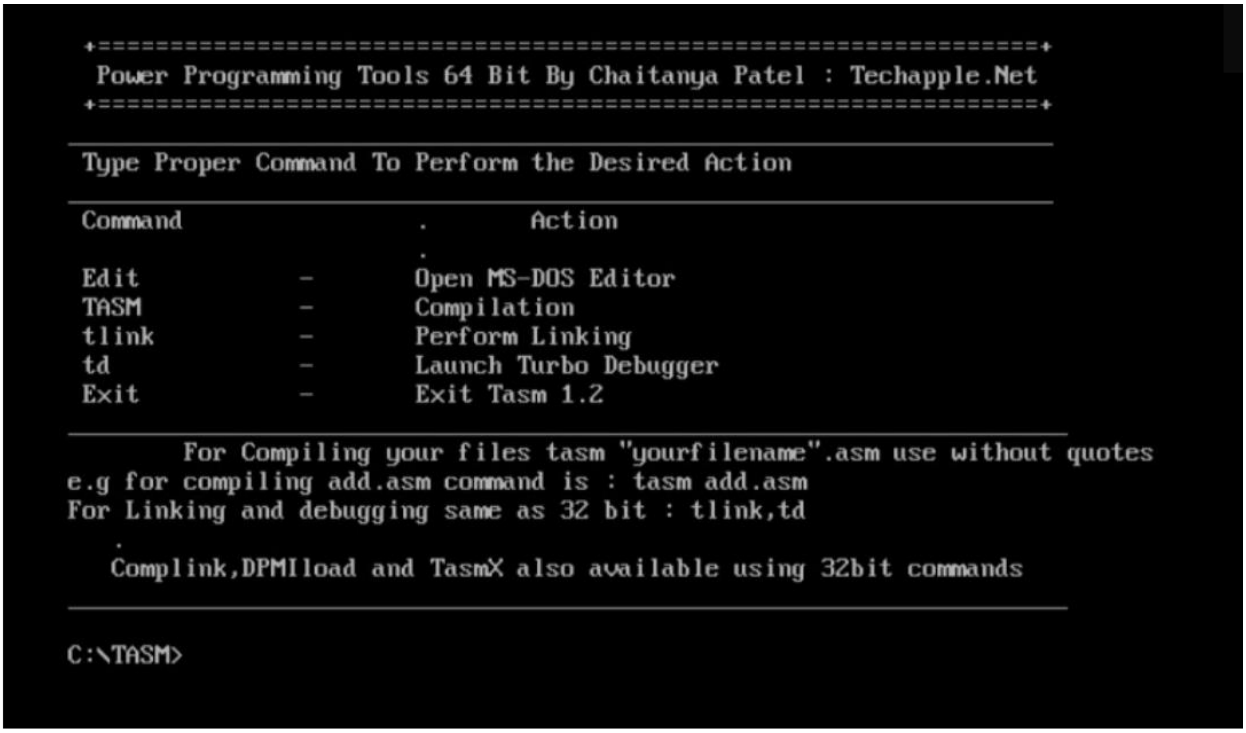
Documentation and coding style are essential. List assembly language's drawbacks.

In this assignment, as a programmer in APU.inc, you are required to develop a program that displays five different forms of shapes by using assembly language, which is TASM. You are also expected to demonstrate creativity in developing the program using assembly language prototype that can deal with the requirements of the developer as well as addressing.

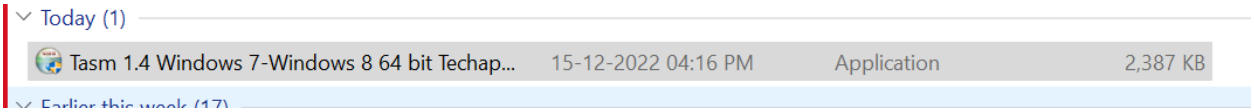
Additionally, before developing the program, demonstrated step-by-step instructions for TASM installation and assembly language procedure and explain the concepts.

TASM INSTALLATION PROCESS

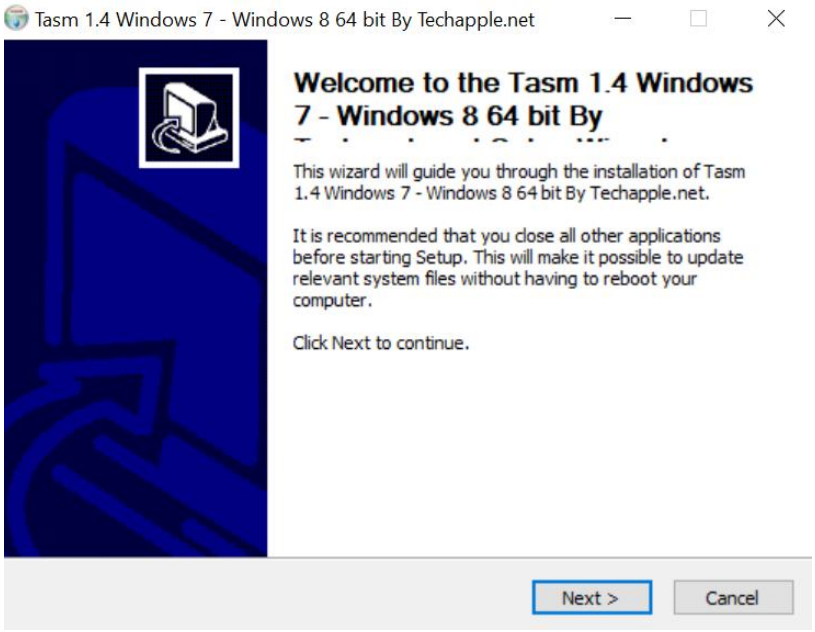
Firstly downloaded TASM from the this link

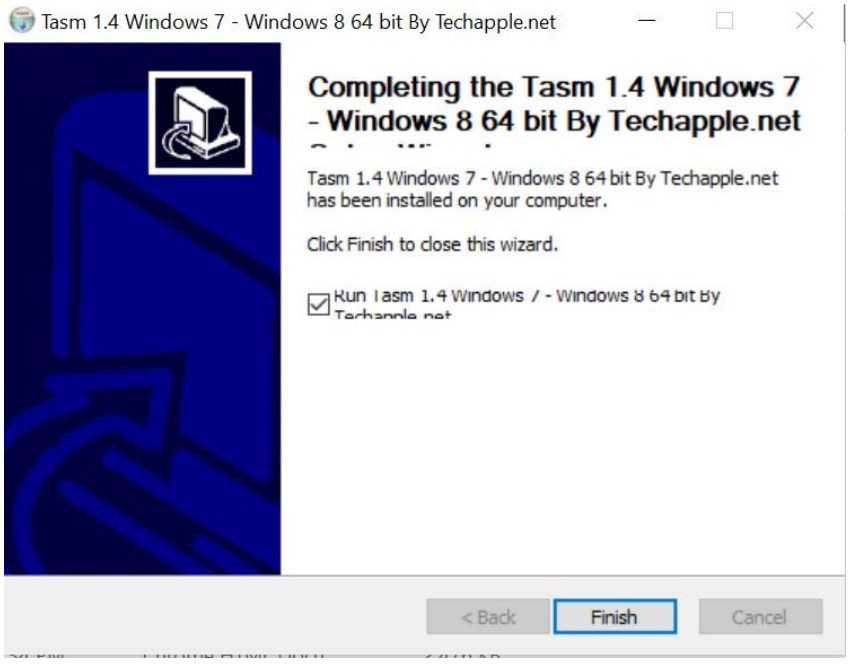


Click here to Download The Installation Package Tasm 1.4 Windows 7|Windows 8|8.1 & Windows 10 64bit Version



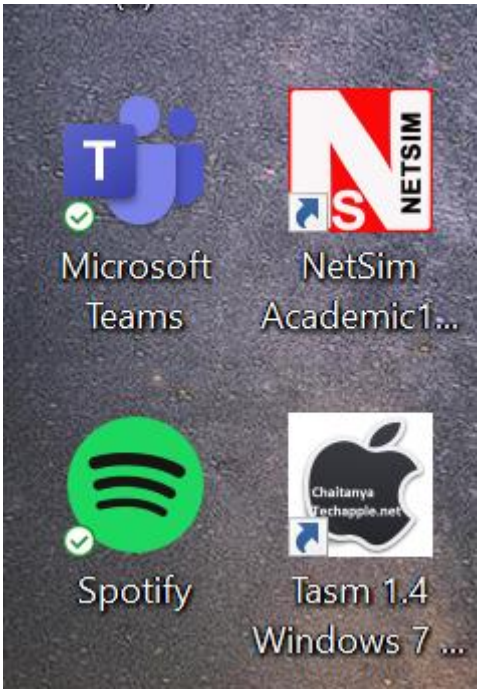
Then, it's pretty simple: just keep clicking "Next," make sure it says "C: tasm 1.4," and don't change where the software is installed.





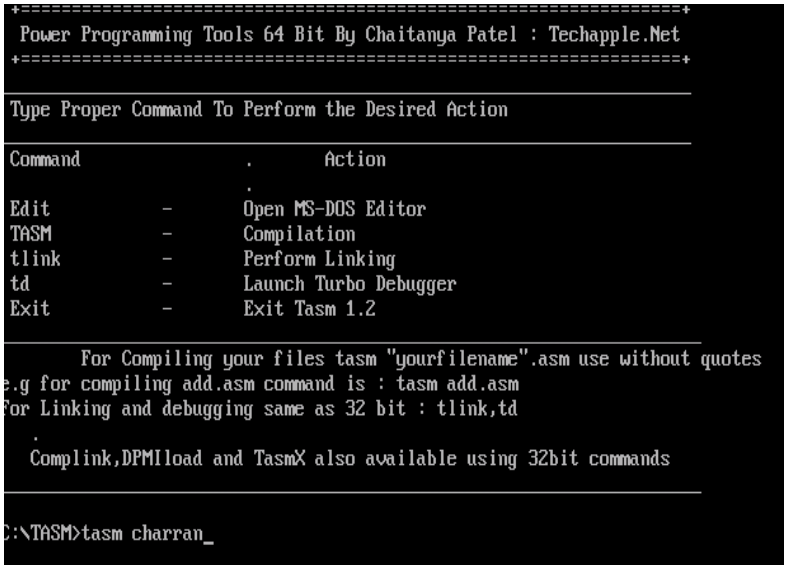
Directory for installing Tasm version 1.4.

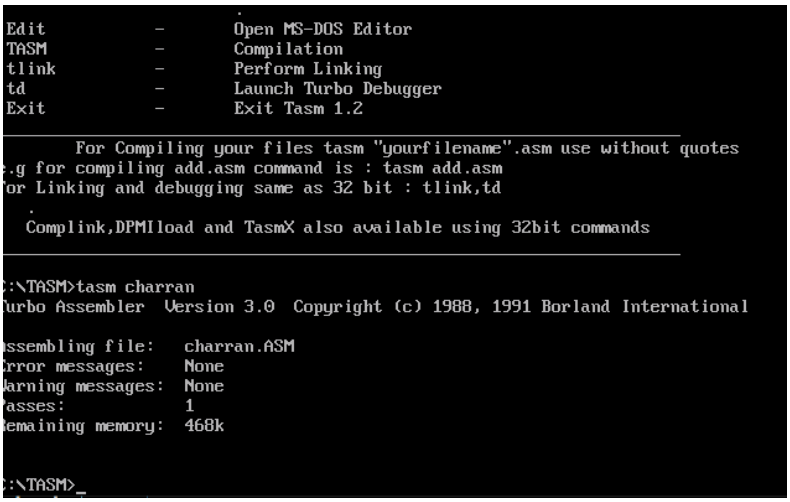
Now, start the program by double-clicking the Tasm 1.4 shortcut that was placed on your desktop.



To Debugging Tasm, Tlink, and Execution

For compilation – tasm(file name). Asm





For edit code – edit (file name).asm

```
C:\TASM>edit charran.asm
```

For linking - tlink (filename)

```
Turbo Assembler  Version 3.0  Copyright (c) 1988, 1991 Borland International

Assembling file:   charran.ASM
Error messages:    None
Warning messages:  None
Passes:            1
Remaining memory:  468k


C:\TASM>tlink charran
Turbo Link  Version 2.0  Copyright (c) 1987, 1988 Borland International
```

To run program – (file name)

```
C:\TASM>charran_
```

Flow control

```
int 21h
cmp al,49
jne pyramido
call number ; call the number procedure
jmp begin
```

In this code, the value in register AL is compared to 49 using the CMP (compare) instruction. While the CMP command subtracts the two numbers, it discards the result. It does a subtraction and then either sets or clears the zero flag. A zero flag is set if the result of the subtraction is zero. The zero flag is reset if the result is greater than zero.

If a comparison does not return true, the JNE (jump if not equal) pyramido command will cause the program to skip to a new address. If the zero flag (ZF) is not present, the JNE instruction will execute the designated instruction. The JNE instruction will be skipped, and execution will proceed if the zero flag is set.

To invoke a previously defined procedure or subroutine, use the CALL (number) command. The CALL command causes the program to immediately transfer control to the designated subroutine or function. At the conclusion of the called subroutine or procedure, the code will jump back to the instruction that followed the CALL instruction.

Finally, we have the JMP (jump) command, which is used to immediately skip to a new point in the code. Here, we utilize the JMP command to skip forward to the one with the name "begin"

Together, the value in register AL is compared to 49 in this code snippet. The zero flag will be reset, and the program will proceed to the "pyramido" instruction after the JNE instruction if the values are not identical. The zero flag is set and the JNE instruction is disregarded if the values are equal. After the program calls a subroutine or number procedure, it will immediately execute the instruction with the label "begin."

```
pyramido:
            cmp al,50
            jne square_box
            call recto1 ; call the design procedure
            jmp begin
```

For this purpose, the CMP (compare) instruction is used to compare the value in register AL to the value 50. While the CMP command subtracts the two numbers, it discards the result. It does a subtraction and then either sets or clears the zero flag. A zero flag is set if the result of the subtraction is zero. The zero flag is reset if the result is greater than zero.

If a comparison does not return true, the JNE (jump if not equal) command will cause the programme to skip to a new address. If the zero flag (ZF) is not present, the JNE instruction will execute the "square box" instruction. The JNE instruction will be skipped, and execution will proceed if the zero flag is set.

To invoke a previously defined procedure or subroutine, use the CALL (call retro 1 command. The CALL command causes the programme to immediately transfer control to the designated subroutine or function. At the conclusion of the called subroutine or procedure, the code will jump back to the instruction that followed the CALL instruction. After the JNE command, the programme will execute the "square_box " or method.

Finally, we have the JMP (jump) command, which is used to immediately skip to a new point in the code. The JMP instruction is used to immediately execute the instruction with the label "begin" in this code snippet.

Overall, this code snippet checks whether the value in register AL is less than or equal to 50. The zero flag will be removed and the JNE will be set if the values are not equal.

```
square_box:
            cmp al,51
            jne nested_triangles
            call square_box1 ; call the square_box procedure
            jmp begin
```

For this purpose, the CMP (compare) instruction is used to compare the value in register AL to the value 5. While the CMP command subtracts the two numbers, it discards the result. It does a subtraction and then either sets or clears the zero flag. A zero flag is set if the result of the subtraction is zero. The zero flag is reset if the result is greater than zero.

If a comparison does not return true, the JNE (jump if not equal) command will cause the programme to skip to a new address. If the zero flag (ZF) is not present, the JNE instruction will execute the "nested_triangle " instruction. The JNE instruction will be skipped, and execution will proceed if the zero flag is set.

To invoke a previously defined procedure or subroutine, use the CALL (call square_box 1 command. The CALL command causes the programme to immediately transfer control to the designated subroutine or function. At the conclusion of the called subroutine or procedure, the code will jump back to the instruction that followed the CALL instruction. After the JNE command, the programme will execute the "s " or method.

Finally, we have the JMP (jump) command, which is used to immediately skip to a new point in the code. The JMP instruction is used to immediately execute the instruction with the label "begin" in this code snippet.

Overall, this code snippet checks whether the value in register AL is less than or equal to 51. The zero flag will be removed and the JNE will be set if the values are not equal.

```
nested_triangles:
                cmp al,52
                jne tri
                call nest  ; call the nested_triangles procedure
                jmp begin
```

For this purpose, the CMP (compare) instruction is used to compare the value in register AL to the value 52. While the CMP command subtracts the two numbers, it discards the result. It does a subtraction and then either sets or clears the zero flag. A zero flag is set if the result of the subtraction is zero. The zero flag is reset if the result is greater than zero.

If a comparison does not return true, the JNE (jump if not equal) command will cause the programme to skip to a new address. If the zero flag (ZF) is not present, the JNE instruction will execute the "tri " instruction. The JNE instruction will be skipped, and execution will proceed if the zero flag is set.

To invoke a previously defined procedure or subroutine, use the CALL (call nest command. The CALL command causes the programme to immediately transfer control to the designated subroutine or function. At the conclusion of the called subroutine or procedure, the code will jump back to the instruction that followed the CALL instruction. After the JNE command, the programme will execute method.

Finally, we have the JMP (jump) command, which is used to immediately skip to a new point in the code. The JMP instruction is used to immediately execute the instruction with the label "begin" in this code snippet.

Overall, this code snippet checks whether the value in register AL is less than or equal to 52. The zero flag will be removed and the JNE will be set if the values are not equal.

```
tri:
      cmp al,53
      jne end_program
      call tri1
      jmp begin
```

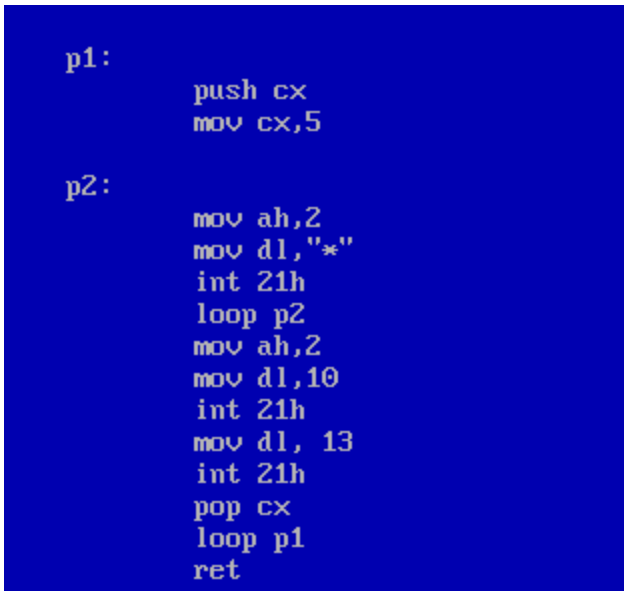
With the CMP (compare) instruction, we check whether the value in register AL is greater than or equal to 53 in this code fragment. While the CMP command subtracts the two numbers, it discards the result. It does a subtraction and then either sets or clears the zero flag. A zero flag is set if the result of the subtraction is zero. The zero flag is reset if the result is greater than zero.

If a comparison does not return true, the JNE (jump if not equal) command will cause the programme to skip to a new address. If the zero flag (ZF) is not set, the JNE instruction will execute the instruction designated "ending." The JNE instruction will be skipped, and execution will proceed if the zero flag is set.

To invoke a previously defined procedure or subroutine, use the CALL (call tri 1) command. The CALL command causes the programme to immediately transfer control to the designated subroutine or function. At the conclusion of the called subroutine or procedure, the code will jump back to the instruction that followed the CALL instruction. After the JNE instruction, the programme will execute the "end program " subroutine or method.

Last but not least, we have the JMP (jump) command, which is used to immediately skip to a new point in the code. The JMP instruction is used to immediately execute the instruction with the label "begin" in this code snippet.

Over all, this code fragment checks whether the value in register AL is equal to 53. If the values are not equal, the JNE instruction will clear the zero flag and forward control to the "ending" instruction. The zero flag is set and the JNE instruction is disregarded if the values are equal. The code will then unconditionally jump to the "begin" instruction after calling a "" subroutine or method.



```
p1:
        push cx
        mov cx,5

p2:
        mov ah,2
        mov dl,"*"
        int 21h
        loop p2
        mov ah,2
        mov dl,10
        int 21h
        mov dl, 13
        int 21h
        pop cx
        loop p1
        ret
```

A 5 is entered into the cx register.

Doing a while loop to display the character '*' until the loop counter cx equals 0. This is accomplished by calling the DOS int 21h interrupt service.

Executing the int 21h interrupt service in DOS to show the line feed and carriage return characters.

Restoring the cx register with the value that was placed into the stack.

Looping repeatedly until the loop counter cx equals 0.

Each of the five rows of asterisks will be shown on a separate line thanks to this code. This programme begins with the cx register set to 5, which controls the number of asterisks shown on each line. Single asterisk rows are shown by the loop at label p2, whereas multiple rows are displayed by the loop at label p1. If the loop counter is greater than zero, the loop instruction decreases it and returns to the start of the loop.

```
t1:
        PUSH CX
        MOV AH,2
        MOV DL,32
t2:
        INT 21H
        LOOP t2
        MOV CX,BX
        MOV DL,'^'
t3:
        int 21h
        loop t3
        mov ah,2
        mov dl,10
        int 21h
        mov dl,13
        int 21h
        INC BX
        inc bx
        pop cx
        loop t1
        ret
```

Looping until the loop counter cx equals 0 to display a space character through the DOS int 21h interrupt service.

The cx register has been set to the contents of the bx register.

Looping to display the letter " until the loop counter cx equals 0. This is done by using the DOS int 21h interrupt service.

Executing the int 21h interrupt service in DOS to show the line feed and carriage return characters.

Adding a doubling increment to the bx register.

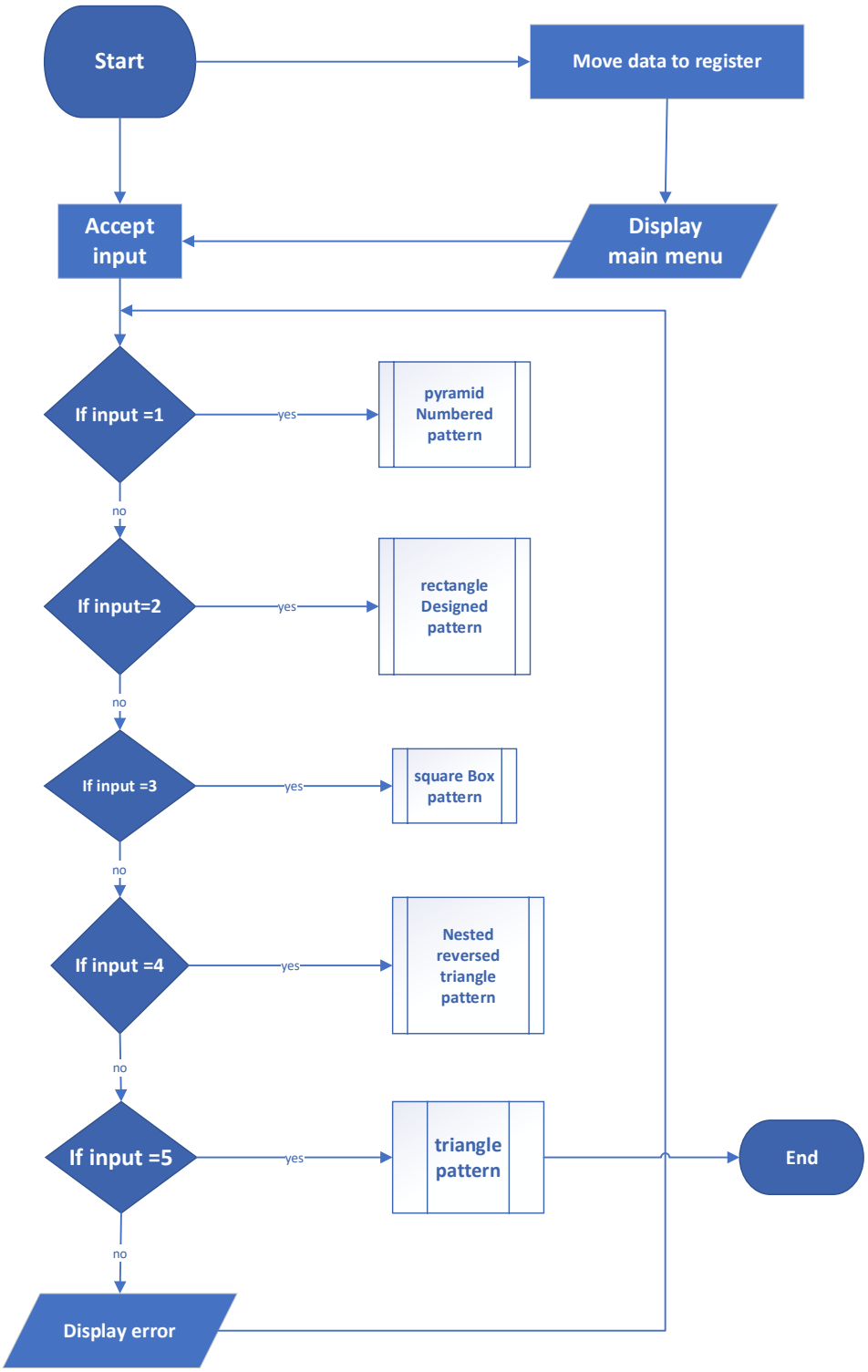
Restoring the cx register with the value that was placed into the stack.

Looping repeatedly until the loop counter cx equals 0.

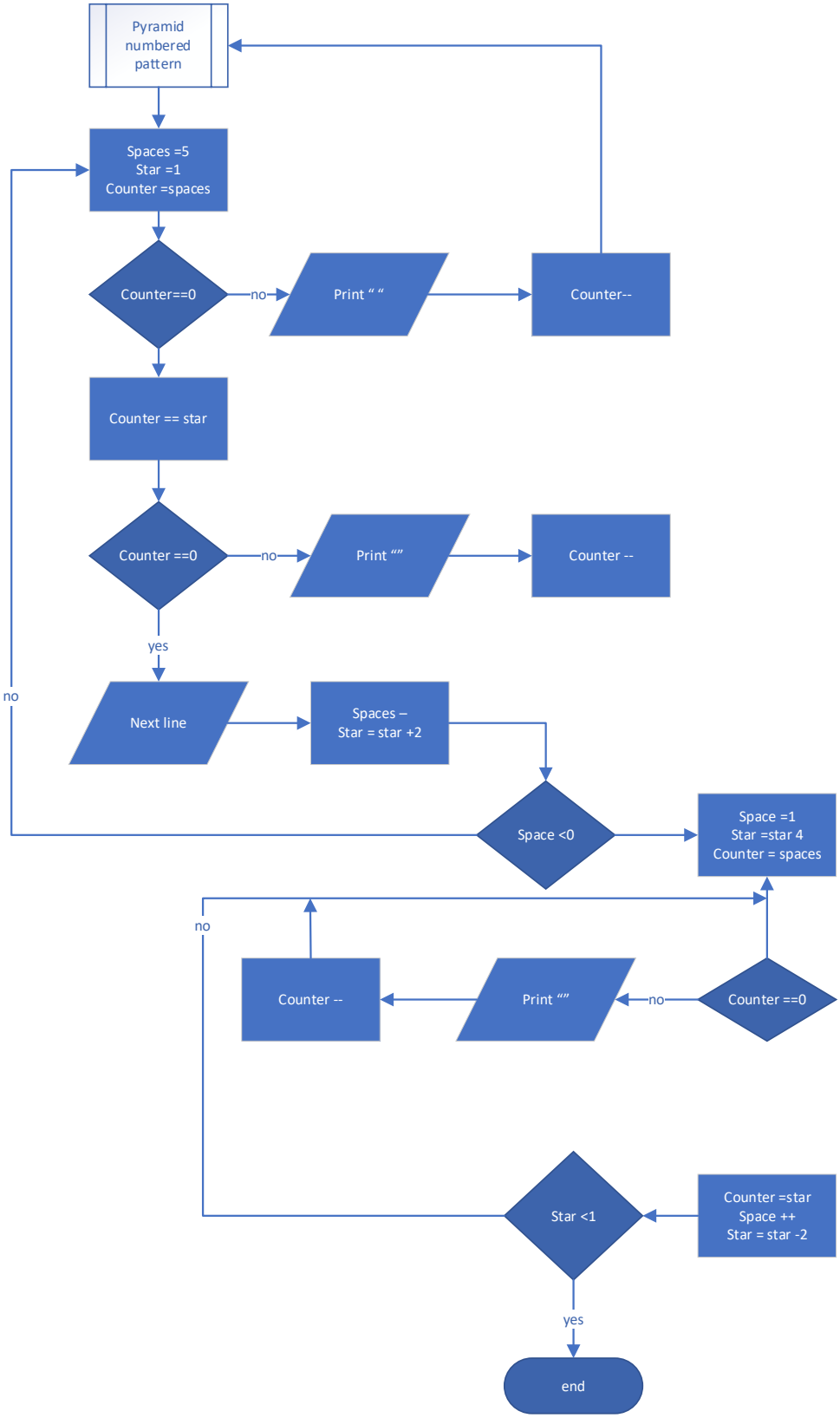
The following code will print 5 columns of text, one column per line. In the first row, we'll have a single space followed by a single tilde (), in the second row, we'll have two spaces followed by two tildes (), and so on. Every time through the loop shown by label t1, the bx register is increased by 2, resulting in a different number of spaces and " characters being displayed on the screen. The" characters are shown by the loop at label t3, while the spaces are shown by the loop at label t2. If the loop counter is greater than zero, the loop instruction decreases it and returns to the start of the loop.

System design using flow chart

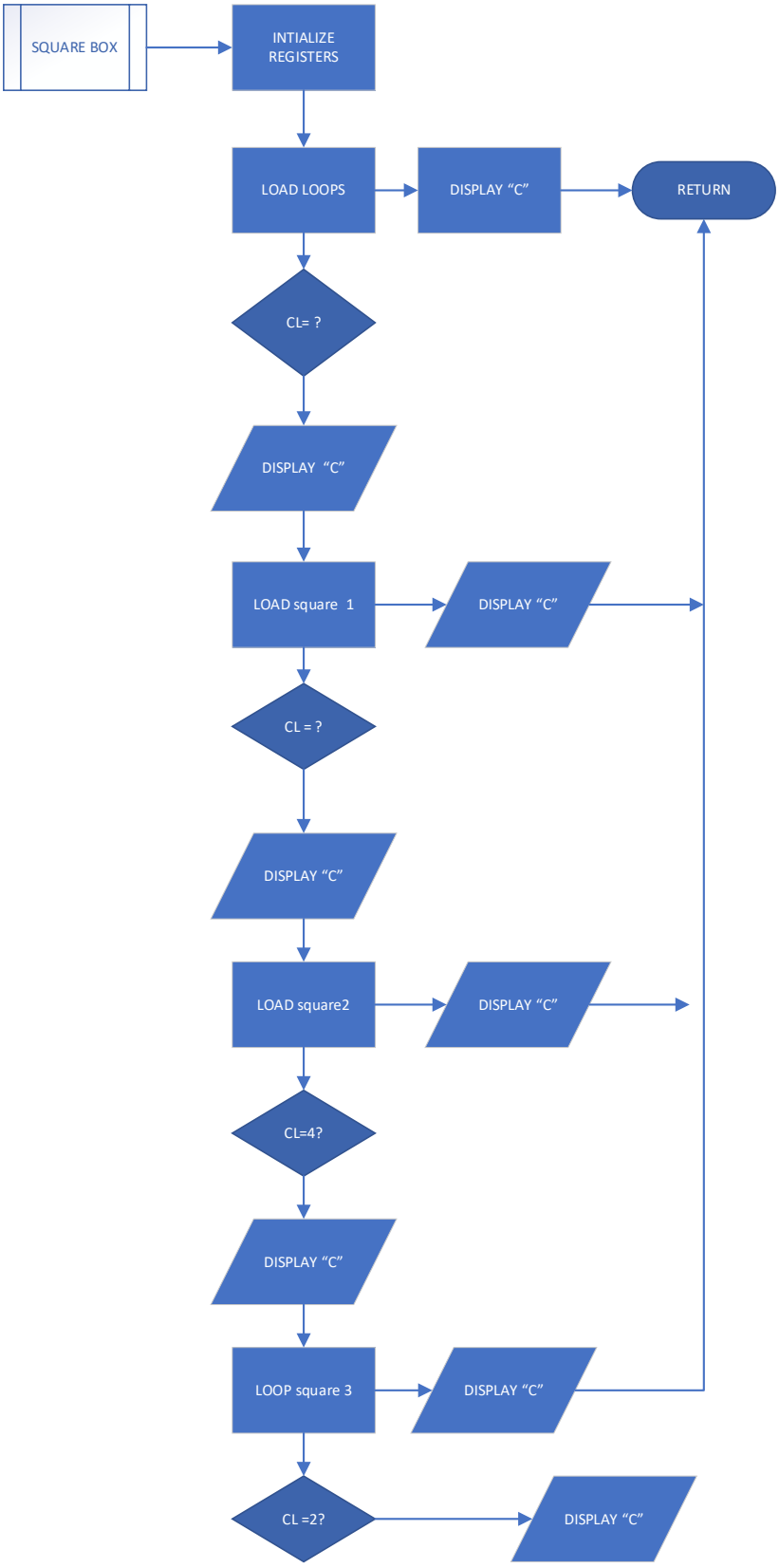
Main menu flow



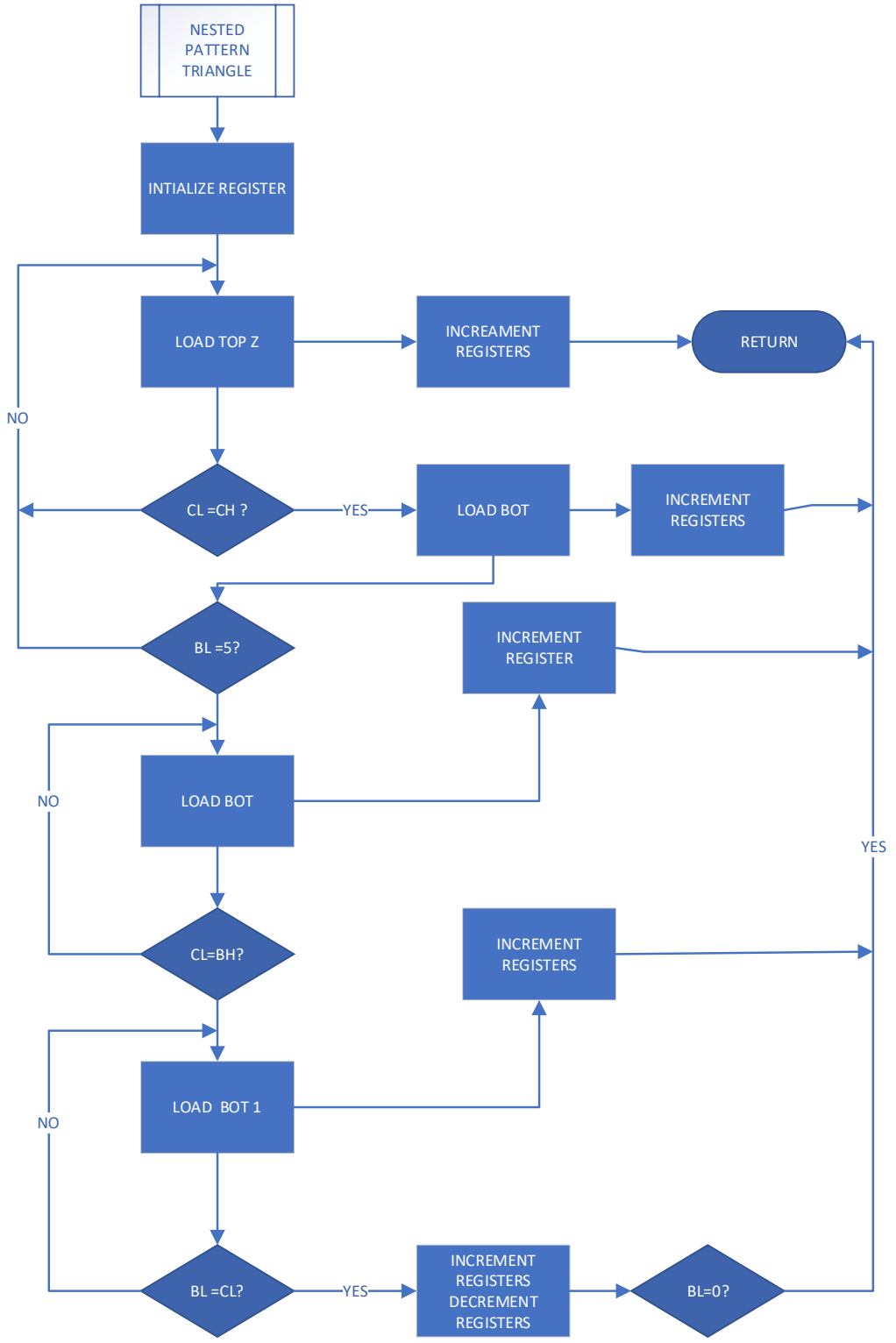
Pyramid number pattern



Square box pattern



Nested reversed triangles



System screenshots

When the user run the program, this is the main menu which will be produced on the screen

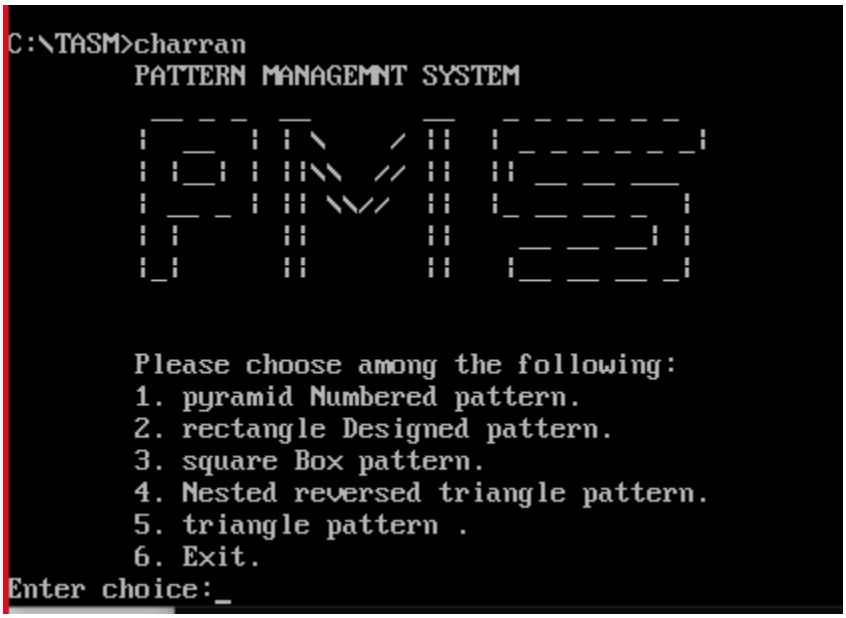
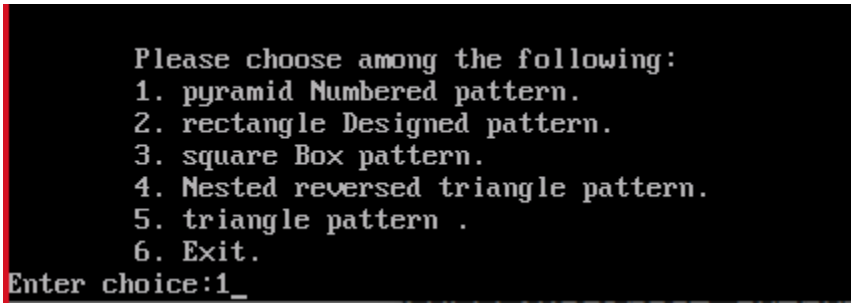


Figure 1 main menu

The five different forms of the shapes are the outputs, by asking the user to select which shape option user want to choose to produce



If the 1st option is chosen by clicking "1" which displays pyramid numbered pattern

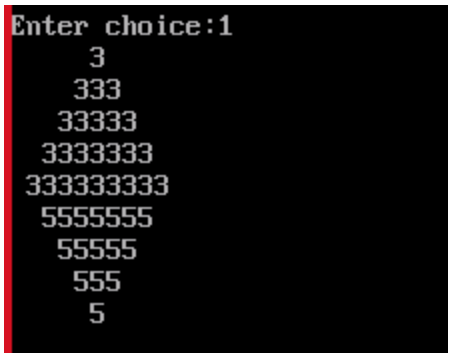


Figure 2 dimond

If the 2nd option is chosen by clicking “2” which displays rectangle designed pattern

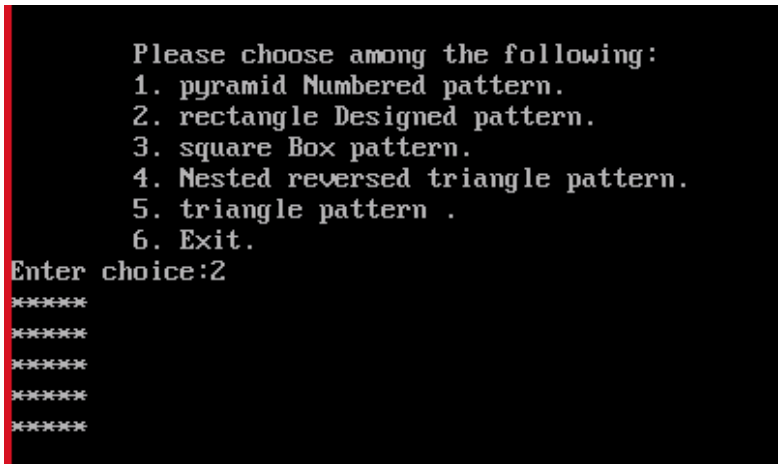


Figure 3 rectangle

If the 3rd option is chosen by clicking “3” which displays square box pattern

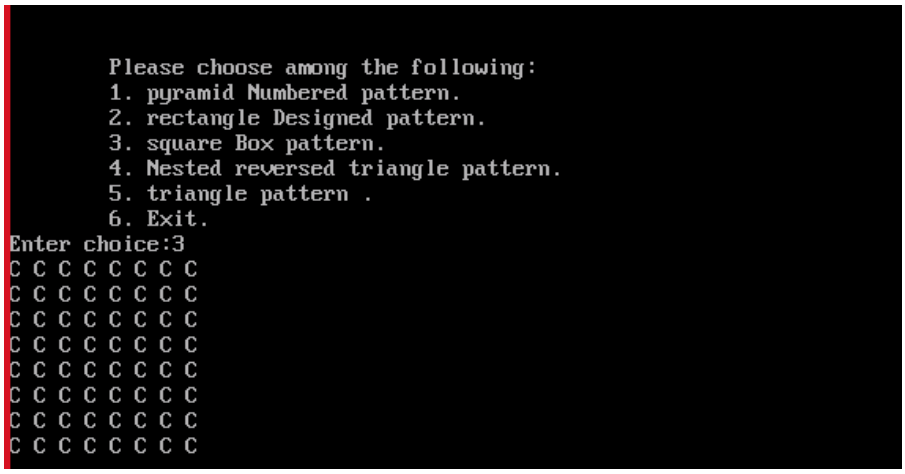


Figure 4 square

If the 4th option is chosen by clicking “4” which displays nested reversed triangle pattern

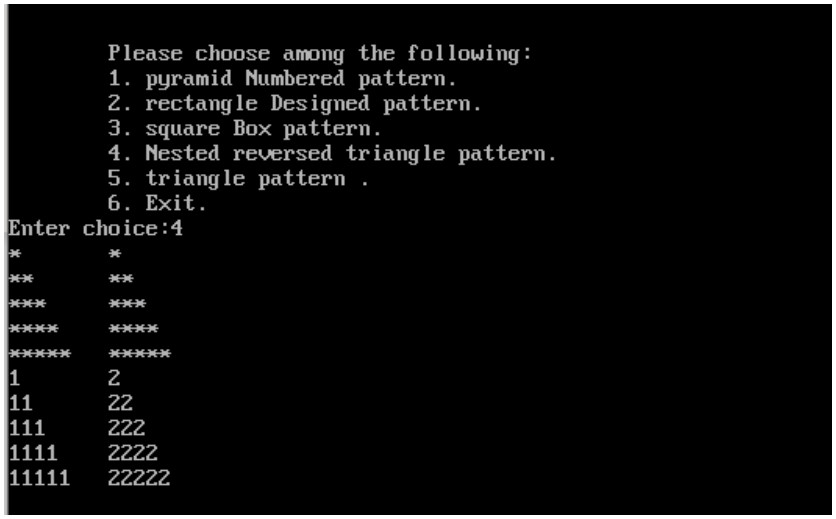


Figure 5 nested right angled triangles

If the 5th option is chosen by clicking “5” which displays triangle pattern

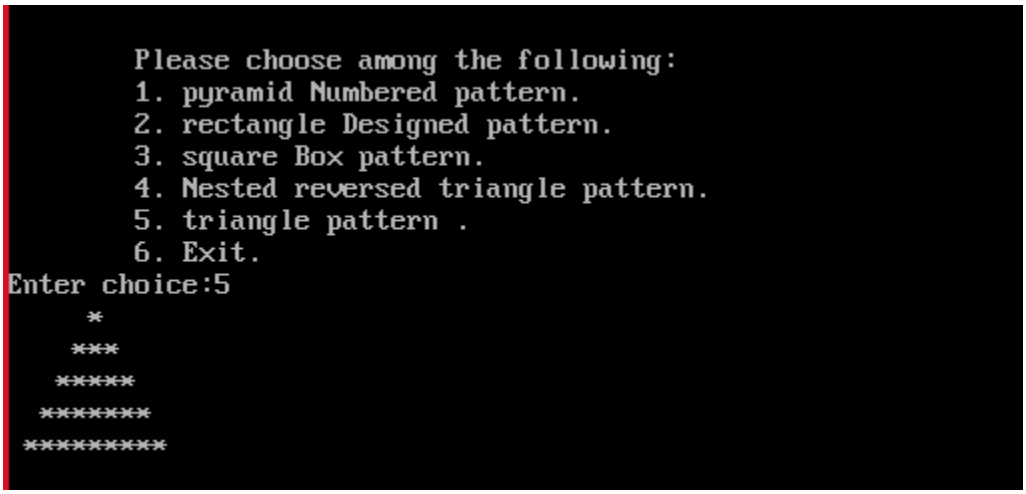


Figure 6 triangle

If the 6th option is chosen by “6” which exits the program

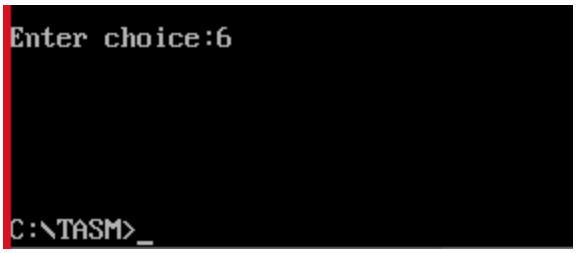


Figure 7 exit

Source code

```
.model small

.stack 100h

.data

tem_data DB  13,10,'$'

                        mess1 DB  09, 'PATTERN MANAGEMNT SYSTEM$'

                        mess2 DB  09,  ' __ _ _  __       __   _ _ _ _ _ _    $'

                        mess3 DB  09,  '|  __  | | \    / ||  | _ _ _ _ _ _|  $'

                        mess4 DB  09,  '| |__| | ||\\  // ||  || __ __ ___    $'

                        mess5 DB  09,  '| __ _ | || \\//  ||  |_ __ __ _  |   $'

                        mess6 DB  09,  '| |      ||       ||    __ __ __| |   $'

                        mess7 DB  09,  '|_|      ||       ||   |__ __ __ _|   $'


                        Option db 09,'Please choose among the following:$'

                        Option1 db 10,9, "1. pyramid Numbered pattern. $"

                        Option2 db 10,9, "2. rectangle Designed pattern. $"

                        Option3 db 10,9, "3. square square_box pattern.$"

                        Option4 db 10,9, "4. nested triangles reversed triangle
pattern. $"

                        Option5 db 10,9, "5. triangle pattern .$"

                        Option6 db 10,9, "6. Exit. $"

                        choice db 13,10, "Enter choice:$",13,10
```

```
star db ?
blank db ?



.CODE
;macro
showmessage Macro Mess
lea dx,Mess
mov ah,9h
int 21h
EndM
Main Proc
        mov ah,6 ;clear the screen
        mov al,0 ;fullup the screen
        int 10h
        mov ax,@data ; get  data segment
        mov ds,ax ; point DS to  the data segment
        mov ah,13h ; assign cursor to start somewhere
        mov al,0 ; check accumulator register at  level 0
        mov bh,0 ; check  base register at level 0
        mov cx,6 ; number of character displayed from the msg
        mov dh,3 ; number of vertical line beginning of the cursor
        mov dl,36 ; number of horizontal lines
        mov bp, offset mess1 ;display  the message
        int 10h
        mov ah,13h
        mov al,0
        mov bh,0
        mov bl,14
        mov cx,8
```

```
mov dh,4
mov dl,35
mov bp, offset mess1
int 10h
mov ah,13h
mov al,0
mov bh,0
mov bl,14
mov cx,2
mov dh,5
mov dl,35
mov bp, offset mess1
int 10h
mov ah,13h
mov al,0
mov bh,0
mov bl,14
mov cx,2
mov dh,5
mov dl,41
mov bp, offset mess1
int 10h
mov ah,13h
mov al,0
mov bh,0
mov bl,14
mov cx,2
mov dh,6
mov dl,35
mov bp, offset mess1
```

```
int 10h
mov ah,13h
mov al,0
mov bh,0
mov bl,14
mov cx,2
mov dh,6
mov dl,41
mov bp, offset mess1
int 10h
mov ah,13h
mov al,0
mov bh,0
mov bl,14
mov cx,2
mov dh,7
mov dl,35
mov bp, offset mess1
int 10h
mov ah,13h
mov al,0
mov bh,0
mov bl,14
mov cx,2
mov dh,7
mov dl,41
mov bp, offset mess1
int 10h
mov ah,13h
mov al,0
```

```
mov bh,0
mov bl,14
mov cx,2
mov dh,8
mov dl,35
mov bp, offset mess1
int 10h
mov ah,13h
mov al,0
mov bh,0
mov bl,14
mov cx,2
mov dh,8
mov dl,41
mov bp, offset mess1
int 10h
mov ah,13h
mov al,0
mov bh,0
mov bl,14
mov cx,8
mov dh,9
mov dl,35
mov bp, offset mess1
int 10h
mov ah,13h
mov al,0
mov bh,0
mov bl,14
mov cx,6
```

```
          mov dh,10
          mov dl,36
          mov bp, offset mess1
          int 10h
showmessage mess1
showmessage tem_data
showmessage mess2
showmessage tem_data
showmessage mess3
showmessage tem_data
showmessage mess4
showmessage tem_data
showmessage mess5
showmessage tem_data
showmessage mess6
showmessage tem_data
showmessage mess7
showmessage tem_data
showmessage tem_data
showmessage tem_data
showmessage Option
showmessage Option1
showmessage Option2
showmessage Option3
showmessage Option4
showmessage Option5
showmessage Option6

OptionList:
             mov ah,9
```

```
            showmessage choice
            mov ah,1
            int 21h
            cmp al,49
            jne pyramido
            call number ; call the number procedure
            jmp begin

pyramido:
            cmp al,50
            jne square_box
            call recto1 ; call the design procedure
            jmp begin

square_box:
            cmp al,51
            jne nested_triangles
            call square_box1 ; call the square_box procedure
            jmp begin

nested_triangles:
                  cmp al,52
                  jne tri
                  call nest  ; call the nested_triangles procedure
                  jmp begin

tri:
      cmp al,53
      jne end_program
      call tri1
```

```
      jmp begin




end_program :  ;to end the program
            cmp al,54
            jne error_msg
            jmp ending
error_msg: ; display error message in case of wrong input
         mov al,9
         int 21h


begin:;to loop the Option
      loop OptionList


ending:
       mov cx,5
top:
      mov dl,10
      mov ah,2
      int 21h
      loop top
      mov ah,4ch
      int 21h
main endp


;diomond number pattern procedure
number proc
            mov bl,09
            int 21h
```

```
            mov cx,5 ; set counter to 5
            mov bx,1  ; base value set to 1
dio1:
   push cx
dio2:
      mov ah,2 ;display character service
      mov dl,32 ;display space
      int 21h ;call DOS
      loop dio2
      mov cx, bx ;compare cx with bx
dio3:
      mov ah,2
      mov dl,'3' ;display number 9
      int 21h
      loop dio3
      mov ah,2
      mov dl,10 ;line feed
      int 21h
      mov dl,13 ;carriage return
      int 21h
      inc bx ;increment bx
      inc bx
      pop cx
      loop dio1
      mov cx,4
      mov bh,7 ;base reg value is 7
      mov bl,2 ;base value set to 2
      mov star, bh
      mov blank, bl
dio4:
```

```
        cmp blank, 0 ;compare blank with 0
        JE dio5 ;jump to dio5 if equal
        mov ah,2
        mov dl,32
        int 21h
        dec blank ;decrement blank by 1
        JMP dio4
dio5:
        mov ah,2
        mov dl,'5'
        int 21h
        dec star
        cmp star,0
        jne dio5
dio6:
        mov ah,2
        mov dl,10
        int 21h
        mov dl,13
        int 21h
        dec bh
        dec bh
        mov star, bh
        inc bl
        mov blank,bl
        loop dio4
        ret
number endp
```

```
;RECTANGLE DESIGN PATTERN PROCEDURE


recto1 proc
mov bl,09
int 21h
mov cx,5
mov bx,5


    p1:
            push cx
            mov cx,5


    p2:
            mov ah,2
            mov dl,"*"
            int 21h
            loop p2
            mov ah,2
            mov dl,10
            int 21h
            mov dl, 13
            int 21h
            pop cx
            loop p1
            ret
recto1 endp
```

```
;square square_box pattern procedure
square_box1 proc
                  mov dl,10
                  int 21h
                  mov ah,2; display character service
                  mov bl,5; set register to 5
                  mov cl,0;cl register value set to 0
                  mov ch,8;ch register value is 8

;First line
squareline1:
            inc cl
            mov dl, 'C'
            int 21h
            mov dl, " "
            int 21h
            cmp cl, ch
            jne squareline1
            mov ah,2
            mov dl,00h
            int 21h
            mov dl,10
            int 21h

;Second line
mov dl,'C'
```

```
int 21h
mov dl," "
int 21h
mov cl,0
mov ch,6
squareline2:
            inc cl
            mov dl,'C'
            int 21h
            mov dl," "
            int 21h
            cmp cl,ch
            jne squareline2
            mov dl,'C'
            int 21h
            mov ah,2
            mov dl,00h
            int 21h
            mov dl,10
            int 21h

;Third line
mov dl,'C'
int 21h
mov dl," "
int 21h
mov dl,'C'
int 21h
mov dl," "
int 21h
```

```
mov cl,0
mov ch,4
squareline3:
            inc cl
            mov dl,'C'
            int 21h
            mov dl," "
            int 21h
            cmp cl,ch
            jne squareline3
            mov dl,'C'
            int 21h
            mov dl," "
            int 21h
            mov dl,'C'
            int 21h
            mov ah,2
            mov dl,00h
            int 21h
            mov dl,10
            int 21h

;Fourth line
mov dl,'C'
int 21h
mov dl," "
int 21h
mov dl,'C'
int 21h
mov dl," "
```

```
int 21h
mov dl,'C'
int 21h
mov dl," "
int 21h
mov cl, 0
mov ch, 2
squareline4:
            inc cl
            mov dl,'C'
            int 21h
            mov dl," "
            int 21h
            cmp cl,ch
            jne squareline4
            mov dl,'C'
            int 21h
            mov dl," "
            int 21h
            mov dl,'C'
            int 21h
            mov dl," "
            int 21h
            mov dl,'C'
            int 21h
            mov ah,2
            mov dl,00h
            int 21h
            mov dl,10
            int 21h
```

```
;Fifth line
mov dl,'C'
int 21h
mov dl," "
int 21h
mov dl,'C'
int 21h
mov dl," "
int 21h
mov dl,'C'
int 21h
mov dl," "
int 21h
mov cl,0
mov ch,2
squareline5:
            inc cl
            mov dl,'C'
            int 21h
            mov dl," "
            int 21h
            cmp cl,ch
            jne squareline5
            mov dl,'C'
            int 21h
            mov dl," "
            int 21h
            mov dl,'C'
            int 21h
```

```
            mov dl," "
            int 21h
            mov dl,'C'
            int 21h
            mov ah,2
            mov dl,00h
            int 21h
            mov dl,10
            int 21h

;Sixth line
mov dl,'C'
int 21h
mov dl," "
int 21h
mov dl,'C'
int 21h
mov dl," "
int 21h
mov cl,0
mov ch,4
squareline6:
            inc cl
            mov dl,'C'
            int 21h
            mov dl," "
            int 21h
            cmp cl,ch
            jne squareline6
            mov dl,'C'
```

```
            int 21h
            mov dl," "
            int 21h
            mov dl,'C'
            int 21h
            mov ah,2
            mov dl,00h
            int 21h
            mov dl,10
            int 21h

;Seventh line
mov dl,'C'
int 21h
mov dl," "
int 21h
mov cl,0
mov ch,6
squareline7:
            inc cl
            mov dl,'C'
            int 21h
            mov dl," "
            int 21h
            cmp cl,ch
            jne squareline7
            mov dl,'C'
            int 21h
            mov ah,2
            mov dl,00h
```

```
            int 21h
            mov dl,10
            int 21h

;8th line
mov cl,0
mov ch,8
squareline8:
            inc cl
            mov dl,'C'
            int 21h
            mov dl, " "
            int 21h
            cmp cl,ch
            jne squareline8
            mov ah,2
            mov dl,00h
            int 21h
            mov dl,10
            int 21h
            ret
square_box1 endp



;nested_triangles loop pattern procedureD
nest proc
         mov dl,10
         int 21h
         mov ah,2;display character function/service
```

```
          mov bl,10;base value set to 10 ( 10 rows)
          mov cl,0;cl register value set to 0 (counter register always 0)
          mov ch,1;ch register value is 1 (counter high bit reg) cl - row, ch-column
          mov bh,1;base reg value is 1
topz:
      inc cl
      mov dl,42
      int 21h
      cmp cl,ch
      jne topz
      mov dl,09
      int 21h
      mov cl,0


;display the second triangle
bot:
      inc cl
      mov dl,42
      int 21h
      cmp cl,ch
      jne bot
      mov dl,10
      int 21h
      mov cl,0
      inc ch
      dec bl
      cmp bl,5
      jne topz
      mov cl,0
```

```
topd:
      inc cl
      mov dl, 49
      int 21h
      cmp cl,bh
      jne topd
      mov dl,09
      int 21h
      mov cl,0

; display the second loop
bot1:
      inc cl
      mov dl,50
      int 21h
      cmp bh, cl
      jne bot1
      mov dl,10
      int 21h
      mov cl,0
      inc bh
      dec bl
      cmp bl,0
      jne topd
      ret
nest endp
```

```
;triangle procedure
tri1 PROC
            mov dl,10
            int 21h
            MOV BX,1
            MOV CX,5

   t1:
            PUSH CX
            MOV AH,2
            MOV DL,32
   t2:
            INT 21H
            LOOP t2
            MOV CX,BX
            MOV DL,'^'
  t3:
            int 21h
            loop t3
            mov ah,2
            mov dl,10
            int 21h
            mov dl,13
            int 21h
            INC BX
            inc bx
            pop cx
            loop t1
            ret
tri1 ENDP
```

```
end main
```

Conclusion

In this assignment we conclude that the using the low-level assembly language developed the 5 different forms shapes and explained about design flow and control flow of the code as a APU developer able to learn about and get experience with the assembly language skillset by the time the system development was done on schedule. The experience of writing code in Assembly has been rewarding, and it accurately portrays the benefits of using a low-level language. Programmer via study and practise of actual programming. As more experience and information is gained, a system developed in a low-level language like this might be useful.

References

https://www.scribd.com/document/433706243/Tasm. (n.d.). Retrieved December 20, 2022, from https://www.scribd.com/document/433706243/Tasm

https://www.instructables.com/How-to-run-TASM-and-Compile-x86-Assembly. . . (n.d.). Retrieved December 20, 2022, from https://www.instructables.com/How-to-run-TASM-and-Compile-x86-Assembly. . .