Individual Assignment

TECHNOLOGY PARK MALAYSIA

CT073-3-2 CSLLT

Computer System Low Level Techniques

APU2F2109CS(CYB)

HAND OUT DATE : 16 – MARCH – 2022

HAND IN DATE : 27 – MAY – 2022

STUDENT NAME : TAN KOK FEENG

STUDENT ID : TP061435

Learning Outcome of the Assignment (CLO:3)

Propose a working example of a program in machine level language using any appropriate assembly programming tool (A5, PLO6)

Instructions:

1. No marks will be awarded for the entire assignment if any part of it is found to be copied directly from printed materials or from another student.
2. All submissions should be made ***online*** on or before the due date.
3. Any late submissions after the deadline will not be entertained.
4. Zero (0) mark will be awarded for late submission, unless extenuating circumstances are upheld.

Table of Contents

List of Figure

1.0 Abstract

This study will be discussing about contribution of assembly language, and the difference between low-level and high-level programming language, and how reverse engineer help in steganography, and also with a prototype to show how to create an simple applications using assembly language along side with the example and source code. This study aim is to understand why assembly language is important and how assembly language can help in reverse engineer and how to use assembly language.

2.0 Introduction to Assembly Language

Assembly language is a low-level programming language. It assists in the translation of programming language into machine code (Pedamkar, n.d.). An assembler is a programme that translates assembly code into machine code that can be executed on a computer (Pedamkar, n.d.). Assembly language has been used to decode instructions and pass them on to machine code for execution. It is mostly determined by the system's structure, whether it be the operating system or the computer architecture (Pedamkar, n.d.). Assembly Language mainly consists of mnemonic processor instructions or data and other statements or instructions (Pedamkar, n.d.).

3.0 Contribution of assembly programming language

Assembly language is a low-level and old programming language. However, assembly language are having lot of contribution towards applications nowadays, in the section below, it will talk about the contribution of assembly language in different recent technology such as embedded system, real time system, device drivers, malware analysis, 8051 microcontroller.

3.1 Embedded System

An embedded system is a microprocessor-based system used in devices to control and monitor the functioning of the device's components (n.d., n.d.). Embedded systems are mostly designed to perform specific task, which can be done by using assembly language (n.d., n.d.). Based on a study, without a good grasp of assembly language and computer architecture, transitioning to embedded systems development is challenging (Ebrahim, 2010). As embedded computing pervades almost every aspect of modern technology, from intelligent devices of all kinds to remote and autonomous systems, a lack of understanding becomes a major impediment to seeing and seizing the many opportunities for participation in this fascinating and rapidly expanding field (Ebrahim, 2010). Based on the findings, assembly language is crucial in embedded systems, either in recently or in the past, there are still lot of situations there are going to need assembly language. Therefore, to understand how to handle an embedded systems and make automation, learning assembly language is an inevitable path.

3.2 Real Time System

A real-time application (RTA) is a software that operates in a time that the user perceives to be immediate or current. The delay must be within a certain threshold, commonly measured in seconds (TechTarget Contributor, 2008). Based on a study, Assembly language is mostly used for highly specific and time-sensitive applications, such as rapid, real-time device drivers (Al-Aubidy, 2011). Assembly-language applications are often quicker than those written in other programming languages (Al-Aubidy, 2011). C is a widely utilised programming language in most computer control applications. It's a powerful language that allows programmers to do low-level tasks without having to resort to assembly (Al-Aubidy, 2011). Based on the findings, assembly language are very useful in real-time application, which are still being used nowadays, but most time, assembly language replaced by C programming for the real time system nowadays. It might seems that assembly language is being disuse for

the real-time application nowadays, but it is still important to understand the concept of assembly language to understand why and how other language can be applied for real time system, also one important thing, on the findings above, assembly language are still and always faster than any other programming language, therefore, to fully eliminate assembly language from real time system is mostly impossible at least for now.

3.3 Device Driver

The majority of operating system kernels are made up of device drivers. They work in a highly privileged environment, similar to other elements of the operating system, and may create disaster if something goes wrong. The relationship between the operating system and the device that they are controlling is controlled by device drivers (Advanced Industrial Control Technology, 2010). Also, A device driver is a software interface that enables operating systems and computer applications to access and interact with hardware functionalities without having to acknowledge prices information about the hardware in use (TEACH COMPUTER SCIENCE Corporation, 2022). Example of device drivers are graphics cards, computer printers, gamepad, scanners, etc. Device drivers were written in assembly language in the early ages of software and computer. Assembly language is a low-level language that could access hardware and CPU instructions (TEACH COMPUTER SCIENCE Corporation, 2022). But nowadays, more programmer prefer writing code using C++ or C programming language (TEACH COMPUTER SCIENCE Corporation, 2022). Based on the findings, assembly language is crucial in the early stage of the device driver technology, without assembly language, the device driver cannot have today result, therefore assembly language is still a crucial part for device driver engineer to study and understand, but not necessary to be used because there are more convenient language out there to be used.

3.4 Malware Analysis

Malware is a term for malicious software that is meant to do harm to a computer system without the individual's consent (Bhojani, 2014). The process of discovering the purpose and features of a specific malware sample, such as a virus, worm, or Trojan horse, is known as malware analysis. This is an essential step in developing effective detection systems for dangerous code (Bhojani, 2014). Based on a study, The disassembly of a binary is usually the most important element of static analysis. This is accomplished with the use of tools that can convert machine code to assembly language (Bhojani, 2014). Based on the findings, malware

analysis tools can help the forensic teams to quickly find out the source or how the malware is created, to create those tools, understanding assembly language is a crucial part for the creator of the tools to proper analysis a malware binary file. Therefore, it is important for cyber security student to understand how assembly work, with that the student can understand how the malware is created in a better way.

3.5 8051 Microcontroller

A microcontroller is a type of computer chip that is designed to control electrical devices. It's kept on a single integrated circuit that's dedicated to completing a single task and running a single programme (Williams, 2022). Based on a study, The 8051 is the first microcontroller of Intel Corporation's MCS-51 series, which was launched near the end of the 1970s (Rose, n.d.). 8051 microcontroller is an 8-bit microcontroller is a fact that been brought out in this website (SHARMA, 2021). Based on a website, Machine Language is represented in Assembly Language, which is a pseudo-English form. The 8051 Microcontroller Assembly Language is made up of Mnemonics, which are English-like words, and Hexadecimal codes. It's also a low-level language that necessitates a thorough grasp of the Microcontroller's design (Electronics Hub Organization, 2017). Based on the findings, to use 8051 microcontroller, it is important for the engineer to understand assembly language, assembly language stand an important role in the 8051 microcontroller which mean without assembly language, people are unable to use microcontroller nowadays.

In a nutshell, assembly language have been contribute in may field such as Embedded system, real time system, device driver, malware analysis and 8051 microcontroller, without the help of assembly language, most of the computer technology nowadays are unable to be done, as assembly language stand an important role in the very low/base of computing technology, which assembly language are still contributing many till todays.

4.0 Evaluation of the low-level programming language

There are two type of programming language, high-level and low-level. Well-known high-level programming are programming language such as Python, Java, JavaScript, and many more. In contrast, low-level programming language or also known as assembly language, and why is assembly language still exist? In this section, the difference between high-level programming language and low-level programming language, and the importance of assembly language in cyber security will be brought out.

4.1 Difference between high-level and low-level programming language (assembly language)

Section	High-level language	Assembly language
Conversion	For the conversion process, a high-level language requires an interpreter/compiler.	The conversion procedure in the assembly language needs the use of an assembler.
Process of Conversion	Convert a high-level language to an assembly language, which is subsequently translated into a machine-level language for the computer.	Converting an assembly language to a machine language
Machine Dependency	A high-level language is a form of language that is not machine dependent.	Assembly is a machine-dependent programming language.
Code	For functionality, it takes use of English sentences.	For operation, it makes use of mnemonic codes.
Accuracy	Accuracy is generally lower.	It is more accurate.
Performance	In comparison, the performance is not very good.	In general, an assembly language exceeds any high-level language.
Time for code execution	Because it must execute a massive code, it takes longer to complete.	Execution takes less time

Ease of understanding	An assembly language's code is simple to debug and learn.	Debugging and understanding assembly language code is quite challenging.

Table 1, Differences between high-level and low-level programming language (Byju community, n.d.)

As the table shows, there are differences between high-level programming and assembly language. In another word, they are meant for different positions, and in the section below, what role does each high-level programming language and assembly language stand in the cyber security field.

4.2 Python

Python is a powerful programming language that allows user to write code in object-oriented, functional, or imperative styles. Because of its readability and ease of use, it is great for beginners (Ramdas, 2019). Based on a study, Python has long been the language of choice in the field of cyber security. The final script does not need to be compiled by programmers because it is a server-side scripting language. It's a catch-all word that's used in a variety of cyber security scenarios (Faizan, 2021). Python is an important programming language for cyber security professionals since it can be used to detect malware, do penetration testing, scan for threats, and analyse them (Faizan, 2021).

4.3 Java

Java was one of the first languages used in the development of several significant operating systems, including Solaris, Linux, macOS, and Microsoft Windows. It is extensively utilised in a variety of industries since it can power both new and old web servers (Faizan, 2021). Based on a study, Java is used by skilled ethical hackers to build and create complicated, ethical apps and is used by cyber adversaries to reverse-engineer commercial software programmes to uncover and exploit security weaknesses (Faizan, 2021). Also, Java may be used by ethical hackers to create vulnerability testing programmes that can operate on a range of platforms (Faizan, 2021).

4.4 JavaScript

JavaScript is one of the most common languages within internet sites up to 95 percent mentioned in a study (Faizan, 2021)JavaScript is used by front-end developers, full-stack developers, back-end developers, and others. It is the world's most versatile as well as the most commonly spoken language (Faizan, 2021). While users are visiting a website, JavaScript allows programmers to utilise any code, boosting the site's functionality. On the other hand, it may have potentially hazardous functionality that the visitor is ignorant of. If the website is hacked, malicious software might be utilised to launch a programme (Faizan, 2021). In terms of security, JavaScript is a smart choice if the administrator wishes to acquire cookies, misuse event handlers, and do cross-site scripting (Faizan, 2021).

4.5 Assembly Language

Any low-level language that aids in the analysis and understanding of malware is known as an assembly language (Faizan, 2021). Based on a study, Slammer, an assembly-based virus that inflicted service neglect on a significant number of websites, wreaked havoc and slowed online traffic in 2003. A protection overflow hole in Microsoft's SQL server was exploited by the virus. Despite the fact that the problem did not arise overnight — it took months for a cure to be released - many organisations neglected to implement it, allowing the defect to propagate (Faizan, 2021). Assembly is a useful programming language for cyber security experts that need to analyse malware and figure out how it works. Because cyber security professionals are constantly defending against both traditional and current malware, it's vital to understand how malware operates (Faizan, 2021).

In a conclusion, each programming language has its advantages and disadvantage, programming language are most likely to be used in cyber security field, but low-level programming language such as assembly language are also essential if in need to malware analysis or reverse engineering. Therefore, every programming language is important and they stand important role in different position especially for the cyber security field.

5.0 Significance of reverse engineer in image steganography and usage of IDA tool

In this section, the significance of reverse engineering in image steganography and the usage of IDA Pro tool will be discussed. Steganography is the art and science of transmitting information in such a way that the existence of the transmission is undetectable (Lee & Lee, 2020). Which mean steganography is hidden a message inside any type of medium such as art, music, image, etc. It's not the same as cryptography, which is the technique of rendering things unintelligible unless the cryptography key is known (Lee & Lee, 2020). Steganography encrypts a message and hides it in a cover medium, but no one knows about it if only he is aware of it (Lee & Lee, 2020). Reverse engineering, often known as back engineering, is the act of breaking down items such as aeroplanes, software, equipment, and architectural structures into their component parts in order to get design knowledge (A, 2021). In most situations, reverse engineering entails the dismantling of individual components of larger items (A, 2021). Based on a study, a reverse engineering tool known as IDA Pro tool can be used in reverse engineering in steganography (Lee & Lee, 2020). Reverse engineers frequently utilise the Interactive Disassembler (IDA) as a debugger and disassembler to examine programmes (Lee & Lee, 2020). Based on a study, with the help of IDA, the internal operation process may be inversely examined, and the operating mechanism and algorithm used in the executable file can be inversely calculated using the disassembly process (Lee & Lee, 2020). Also, by using IDA tool, the user can examine the encrypted message's encryption technique and concealment mechanism (Lee & Lee, 2020). In addition, IDA generates a flow chart for functions to assist the developer in determining which code routes are used in certain situations (Lee & Lee, 2020). IDA examines the exported and imported functions and builds XRefs between them automatically (Lee & Lee, 2020). This allows the user to search for a certain import library and have all instances of that method call automatically listed in a window (Lee & Lee, 2020). Another key feature is that IDA parses all strings in the executable automatically and displays them in a distinct sub-view (Lee & Lee, 2020). This enables fast navigation through the code by searching for a string that is known to be used at a certain place in the programme (Lee & Lee, 2020). Based on a study, with the help of reverse engineering in steganography (or using the IDA tools), can increase the effectiveness of analysis an steganography faster than the traditional steganalysis technique (Lee & Lee, 2020). In a nutshell, reverse engineering provides better performance in steganography analysis by reversing it using tools such as IDA

tools by disassembler the image and analysis it reversed and find out the message being hidden in the images. Therefore, reverse engineer stands an important role in the field of forensic in steganography.

6.0 Flowchart of System Design

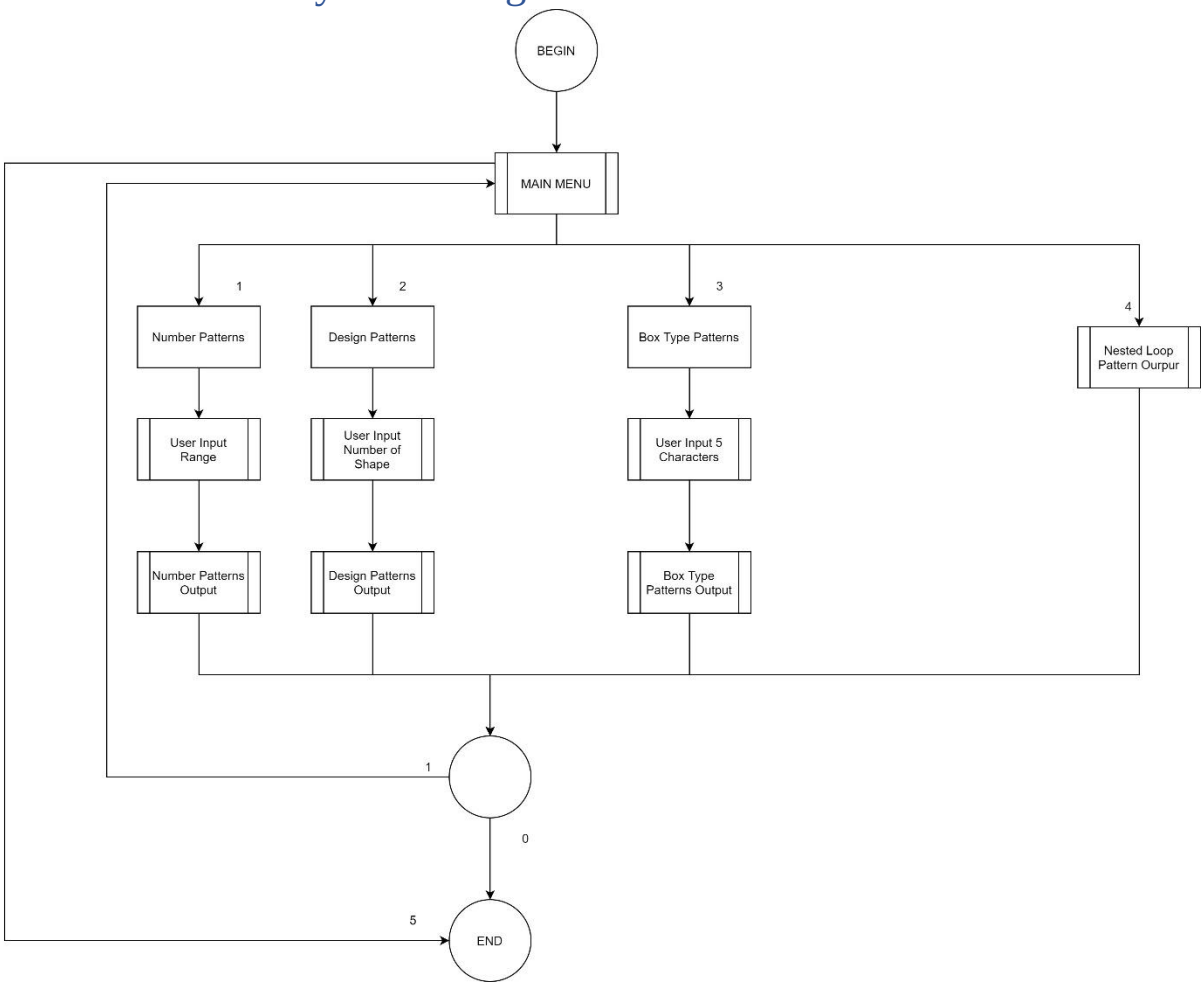

Figure 1, Flowchart of System Design

In this system, there are mainly 5 sections, the first section is the “Main menu”, this is where the program starts and allowed user to choose different output. And the next 4 sections are “Number Patterns”, “Design Patterns”, “Box Type Patterns” and “Nested Loop Patterns”. In Number Patterns, the user can insert 1 to 9 for the range of the output as a diamond shape. In Design Patterns, the user can choose how many shapes to be displayed in between 1 to 9. In Box Type Patterns, user can enter 5 character and display it in a “Box” shape. Lastly, in Nested Loop Patterns, user is allowed to choose between 4 different pattern and display it on the screen. Lastly, the user can choose the exit the program.

7.0 System Assumption & Screenshots

7.1 System Assumption

This system is created for APU Event, which allowed the APU's Event Management Unit to modify the output based on their requirement in the future days.

7.1.1 System Capabilities

This system is allowed to perform tasks below:

1. Displaying diamond shape output (number patterns) with randomized colour
2. Display Design Patterns with given number of shapes
3. Display 5 Characters in a Box form and allowed user to insert any 5 characters
4. Display 4 different nested loop patterns with constant character or increasing number.

7.1.2 System Manual

1. To navigate through the system please use only number for your selection
2. Every selection input can only have 1 number
3. Selection number is in between 1 to 4, any other than that will exit the program
4. Except Box Type Pattern which allowed user to input 5 characters to be displayed
5. User can choose what to display again after display once
6. Example of Output please refer to 7.2.1 System Screenshots section

7.2 System Screenshots

7.2.1 System Banner

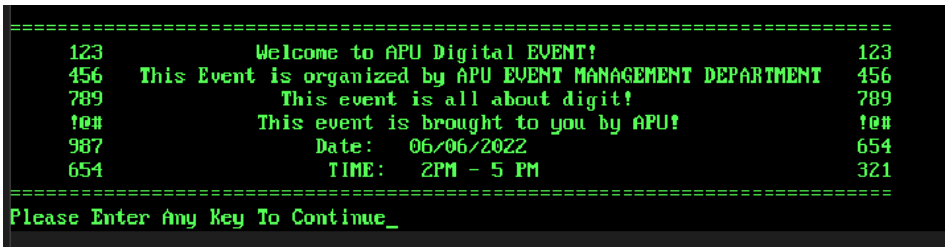

Figure 2, System Banner

7.2.2 System Main Menu

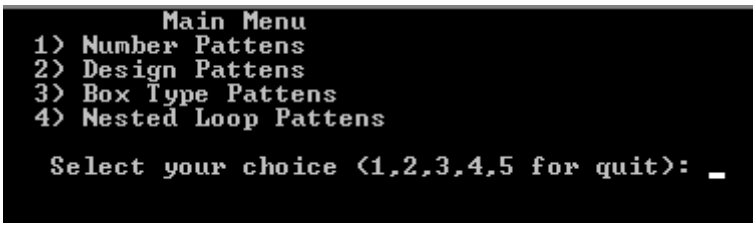

Figure 3, System Main Menu

7.2.3 Number Patterns Menu

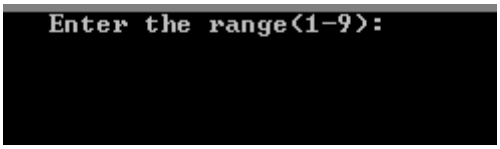

Figure 4, Number Pattern Menu

7.2.4 Example of Number Patterns Output

Figure 5, Number Patterns Output

7.2.5 Design Patterns Menu

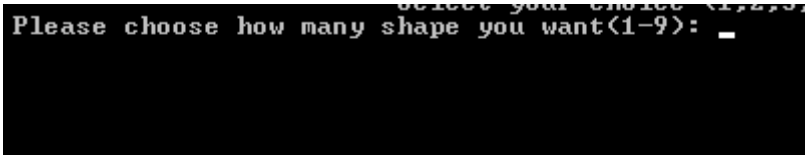

Figure 6, Design Pattern Menu

7.2.6 Example of Design Patterns Output

Figure 7, Design Pattern Output

7.2.7 Box Type Patterns Menu

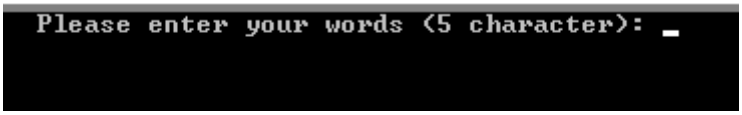

Figure 8, Box Type Patterns Menu

7.2.8 Example of Box Type Patterns Output

Figure 9, Box Type Patterns Output

7.2.9 Example Nested Loop Pattern Output

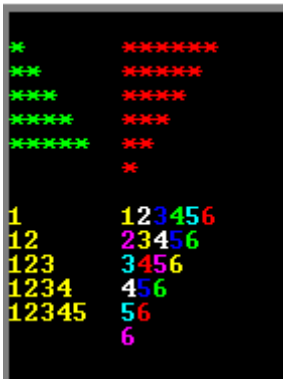

Figure 10, Nested Loop Patterns Output

8.0 Source Code

8.1 Initializing source code

```
.model small

.stack 300h

.data

 ##############Variables go here###########

.code

###############Codes go here###############
```

8.2 Variables

8.2.1 General Variables

```
   count  db ?

   range   db ?

   number  db ?

   colour db 9

banner_continue db "Please Enter Any Key To Continue$"

  banner   db "================================================================================",13,10

           db "    123          Welcome to APU Digital EVENT!                 123",13,10

           db "    456   This Event is organized by APU EVENT MANAGEMENT   DEPARTMENT   456",13,10

           db "    789            This event is all about digit!              789",13,10

           db "    !@#          This event is brought to you by APU!          !@#",13,10

           db "    987              Date:   06/06/2022                      654",13,10

           db "    654              TIME:   2PM - 5 PM                      321",13,10

           db "================================================================================$",13,10

mainMenu      db “Main Menu",13,10

                  db "1) Number Pattens",13,10

                  db "2) Design Pattens",13,10

                  db "3) Box Type Pattens",13,10

                   db "4) Nested Loop Pattens$"

   choiceLine  db " Select your choice (1,2,3,4,5 for quit): $"

   againLine    db " Choose again? (1 for YES): $"

   endLine      db " Thank you for using this program$"
```

8.2.2 Number Pattern Variables

```
messageRange db " Enter the range(1-9): $"

display_count db ?

halfnumber db ?
```

8.2.3 Design Pattern Variables

```
messageDesign     db " Please choose how many shape you want(1-9): $"

shapesize         db ?
```

8.2.4 Box Pattern Variables

```
char1             db ?

char2             db ?

char3             db ?

char4             db ?

char5             db ?

messageBox        db " Please enter your words (5 character): $"
```

8.3General Functions

8.3.1 AGAIN function

```
AGAIN proc
    call Newline
    mov ah,09h
    lea dx,againLine
    int 21h
    mov ah, 01h
    int 21h
    MOV BH, AL
    sub bh,48
    mov count,bh
    CALL Newline
    cmp count,1
    jne again_end
    jmp start
    again_end:
    jmp end
    ret
AGAIN endp
```

8.3.2 RESET function

```
RESET proc
    mov ax,0000
    mov bx,0000
    mov cx,0000
    mov dx,0000
    ret
RESET endp
```

8.3.3 CLS function

```
CLS proc
    MOV AH, 06h    ; Scroll up function
    XOR AL, AL     ; Clear entire screen
    XOR CX, CX     ; Upper left corner CH=row, CL=column
    MOV DX, 184FH  ; lower right corner DH=row, DL=column
    MOV BH, 0Fh
    INT 10H
    retCLS endp
```

8.3.4 DisplayColour function

```
DisplayColour proc
    mov  bh, 0  ;Display page
    mov  ah, 02h  ;SetCursorPosition
    int  10h
    mov  ah, 09h
    int  10h
    ret
DisplayColour endp
```

8.3.5 increase_colour function

```
increase_colour proc
    cmp colour,15
    je  colour_limit_jump
    inc colour
    ret
    colour_limit_jump:
        mov colour,9
        ret
increase_colour endp
```

8.3.6 Newline function

```
Newline proc
```

```
    mov ah, 02h
    mov DL, 13
    int 21h
    mov ah, 02h
    mov DL, 10
    int 21h
    ret
Newline endp
```

8.4 Menu Functions

8.4.1 main_menu function

```
main_menu proc
    mov ah,09h
    lea dx,mainMenu
    int 21h
    CALL Newline
    CALL Newline
    mov ah,09h
    lea dx,choiceLine
    int 21h
    mov ah, 01h
    int 21h
    MOV BH, AL
    sub bh,48
    mov count,bh
    CALL Newline
    cmp count,1
    jne main_menu_choice_2
    jmp  numberPattern
    main_menu_choice_2:
    cmp count,2
    jne main_menu_choice_3
    jmp design_pattern
    main_menu_choice_3:
    cmp count,3
    jne main_menu_choice_4
    jmp boxTypePattern
    main_menu_choice_4:
    cmp count,4
    jne main_menu_choice_end
```

```
jmp nestedLoopPattern_menu
main_menu_choice_end:
cmp count,5
jmp end
ret
main_menu endp
```

8.4.2 nestedLoopPattern_menu function

```
    CALL nestedLoopPattern01
    CALL nestedLoopPattern02
    CALL nestedLoopPattern03
    CALL nestedLoopPattern04
    CALL AGAIN
```

8.4.3 banner_menu function

```
banner_menu proc
    MOV AH, 06h    ; Scroll up function
    XOR AL, AL     ; Clear entire screen
    XOR CX, CX     ; Upper left corner CH=row, CL=column
    MOV DX, 184FH  ; lower right corner DH=row, DL=column
    MOV BH, colour   ;Colour
    INT 10H
    POP CX
    mov ah,09h
    lea dx,banner
    int 21h
    CALL Newline
    mov ah,09h
    lea dx,banner_continue
    int 21h
    mov ah, 01h
    int 21h
    jmp start
    ret
banner_menu endp
```

8.5 Number Pattern Functions

In this pattern it allowed user to select what is the maximum number of the diamond shape can include (1~9) and using the functions below to print it out accordingly, by using 2 nested loop concept for the algorithm, the output will be colourized as well.

8.5.1 upper_diamond function

```
upper_diamond proc
    mov dl,0 ;col0
    mov dh,1 ;row1
    mov cx,0
    mov bx,0
    mov cl, range
    mov bl, range
    mov display_count,2
    mov number,0
    add number,48
    mov halfnumber,1
    jmp upper_diamond_start
    upper_diamond_loop_2:
        inc dh ;increasing row
        call Newline
        mov dl,0 ;reseting column
        add display_count,2
        add halfnumber,1
        dec bx
        mov number,0
        add number,48
    upper_diamond_start:
    upper_diamond_loop1:   ;spacing
        push dx ;new_pushing dx
        mov dh,0 ;clearing dh
        mov dl,' ' ;spacing
        mov ah,02h
```

```
    int 21h
    pop dx ; new_poping dx
    inc dl ;increasing col
    cmp bx,0
    jne upper_diamond_loop_zone
    jmp end
  upper_diamond_loop_zone:
  loop upper_diamond_loop1
     mov al,display_count ;new
     mov count,al        ;new
     upper_diamond_loop_3: ;displaying number
     dec count
     mov al,number ;new
     push cx       ;new
     push bx       ;new
     mov bl,colour
     mov cx,1
     CALL DisplayColour
     inc dl ;increasing col
     CALL increase_colour
     pop bx
     pop cx
     mov al,count
     cmp al,halfnumber
     ja  upper_diamond_increase
     dec number
     jmp upper_diamond_skip
     upper_diamond_increase:
       inc number
     upper_diamond_skip:
     cmp count,1
```

```
        jne upper_diamond_loop_3
        mov cx,bx
    loop upper_diamond_loop_2
    ret
upper_diamond endp
```

8.5.2 lower_diamond function

```
lower_diamond proc
    CALL Newline
    inc dh ;new row
    add display_count,2
    mov bx,0
    mov bl,range
    inc bl
    mov halfnumber,bl
    inc halfnumber
    mov dl,0;reset column
    mov cx,1
    mov al,display_count
    mov count,al
    mov number,0
    add number,48
    lower_diamond_loop_1:   ;displaying number
      dec count
      mov al,number ;new
        push cx       ;new
        push bx       ;new
        mov bl,colour
        mov cx,1
        CALL DisplayColour
        inc dl ;increasing col
        CALL increase_colour
```

```
        pop bx
        pop cx
        mov al,count
        cmp al,halfnumber
       jae  lower_diamond_increase
       dec number
       jmp lower_diamond_skip
       lower_diamond_increase:
       inc number
       lower_diamond_skip:
      cmp count,1
      jne lower_diamond_loop_1
  lower_diamond_loop_2:
     inc dh
     CALL Newline
     mov dl,0 ;reseting row
     sub display_count,2
     mov al,display_count
     mov count,al
     dec bx
     mov number,0
     add number,48
     dec halfnumber
  lower_diamond_loop_3:
     push dx ;new_pushing dx
     mov dh,0 ;clearing dh
     mov dl,' '
     mov ah,02h
     int 21h
     pop dx ; new_poping dx
     inc dl ;increasing col
```

```
    loop lower_diamond_loop_3
      mov cx,0
      mov cl,range
      add cl,2
      sub cx,bx
      CMP bx,0
      je diamond_end
      cmp count,1
      jne lower_diamond_loop_1
    loop lower_diamond_loop_2
    diamond_end:
    ret
lower_diamond endp
```

8.5.3 numberPattern function

```
numberPattern proc
    CALL CLS
    mov ah,09h
    lea dx,messageRange
    int 21h
    mov ah, 01h
    int 21h
    MOV BH, AL
    sub bh,48
    mov range,bh
    CALL Newline
    CALL upper_diamond
    CALL lower_diamond
    CALL RESET
    CALL AGAIN
    ret
numberPattern endp
```

8.6 Design Pattern Functions

In this pattern it allowed user to select what is the maximum number of shapes can appear (1~9) and using the functions below to print it out accordingly. Each function means for each line of the output.

8.6.1 first_design function

```
first_design proc
    mov cx,3
    mov ah,02h
    loop_first_design_1:
        mov dl,' '
        int 21h
    loop loop_first_design_1
    mov dl,'#'
    int 21h
    mov cx,2
    int 21h
    loop_first_design_2:
        mov dl,' '
        int 21h
    loop loop_first_design_2
    dec bl
    ret
first_design endp
```

8.6.2 second_design function

```
second_design proc
    mov cx,4
    mov dh,0
    mov ah,02h
    loop_second_design_1:
        cmp dh,2
        je skip_second_design_1_1
```

```
        mov dl,' '
        int 21h
        jmp skip_second_design_1_2
    skip_second_design_1_1:
        mov dl,'#'
        int 21h
    skip_second_design_1_2:
        inc dh
        loop loop_second_design_1
        mov cx,3
        mov dh,0
    loop_second_design_2:
        cmp dh,1
        je skip_second_design_2_1
        mov dl,' '
        int 21h
        jmp skip_second_design_2_2
    skip_second_design_2_1:
        mov dl,'#'
        int 21h
    skip_second_design_2_2:
        inc dh
        loop loop_second_design_2
        dec bl
    ret
second_design endp
```

8.6.3 third_design function

```
third_design proc
    mov cx,4
    mov dh,0
    mov ah,02h
    loop_third_design_1:
        cmp dh,1
        je skip_third_design_1_1
        mov dl,' '
        int 21h
        jmp skip_third_design_1_2
    skip_third_design_1_1:
        mov dl,'#'
        int 21h
    skip_third_design_1_2:
        inc dh
        loop loop_third_design_1
        mov cx,3
        mov dh,0
    loop_third_design_2:
        cmp dh,2
        je skip_third_design_2_1
        mov dl,' '
        int 21h
        jmp skip_third_design_2_2
    skip_third_design_2_1:
        mov dl,'#'
        int 21h
    skip_third_design_2_2:
        inc dh
        loop loop_third_design_2
```

```
        dec bl
    ret
third_design endp
```

8.6.4 fourth_design function

```
fourth_design proc
    mov cx,4
    mov dh,0
    mov ah,02h
    loop_fourth_design_1:
        cmp dh,0
        je skip_fourth_design_1_1
        mov dl,' '
        int 21h
        jmp skip_fourth_design_1_2
    skip_fourth_design_1_1:
        mov dl,'#'
        int 21h
    skip_fourth_design_1_2:
        inc dh
        loop loop_fourth_design_1
        mov cx,3
        mov dh,0
    loop_fourth_design_2:
        cmp dh,3
        je skip_fourth_design_2_1
        mov dl,' '
        int 21h
        jmp skip_fourth_design_2_2
    skip_fourth_design_2_1:
        mov dl,'#'
```

```
        int 21h
    skip_fourth_design_2_2:
        inc dh
        loop loop_fourth_design_2
        dec bl
        cmp bl,0
        je skip_last_design_1_1
        jmp skip_last_design_1_2

    skip_last_design_1_1:
            mov dl,'#'
            int 21h
    skip_last_design_1_2:
    ret
fourth_design endp
```

8.6.5 design_pattern function

```
design_pattern proc
    mov ax,0
    mov bx,0
    mov cx,0
    mov dx,0
    mov ah,09h
    lea dx,messageDesign
    int 21h
    mov ah, 01h
    int 21h
    MOV BL, AL
    SUB BL,48
    MOV shapesize,bl
    CALL Newline
    mov bl,shapesize
    DESIGN_START_FIRST_1:
    CALL first_design
    cmp bl,0
    jne DESIGN_START_FIRST_1
    CALL Newline
    mov bl,shapesize
    DESIGN_START_SECOND_1:
    CALL second_design
    cmp bl,0
    jne DESIGN_START_SECOND_1
    CALL Newline
    mov bl,shapesize
    DESIGN_START_THIRD_1:
    CALL third_design
    cmp bl,0
```

```
    jne DESIGN_START_THIRD_1
    CALL Newline
    mov bl,shapesize
    DESIGN_START_FOURTH:
    CALL fourth_design
    cmp bl,0
    jne DESIGN_START_FOURTH
    CALL Newline
    mov bl,shapesize
    DESIGN_START_THIRD_2:
    CALL third_design
    cmp bl,0
    jne DESIGN_START_THIRD_2
    CALL Newline
    mov bl,shapesize
    DESIGN_START_SECOND_2:
    CALL second_design
    cmp bl,0
    jne DESIGN_START_SECOND_2
    CALL Newline
    mov bl,shapesize
    DESIGN_START_FIRST_2:
    CALL first_design
    cmp bl,0
    jne DESIGN_START_FIRST_2
    CALL RESET
    CALL AGAIn
design_pattern endp
```

8.7 Box Pattern Functions

In this pattern it allowed user to select what character to be display (must be 5 character) and using the functions below to print it out as a box accordingly. Each function means for each line of the output, the output will be colourized as well.

8.7.1 first_char function

```
first_char proc
    mov count,9
    mov  dl, 0 ;Column
    mov cx,1    ;set to display 1 word pertime
    first_char_start:
    mov  al, char1
    CALL DisplayColour
    inc dl ;next coloumn
    dec count
    cmp count,0
    je first_char_end
    jmp first_char_start
    first_char_end
    ret
first_char endp
```

8.7.2 second_char function

```
second_char proc
    mov count,9
    mov dl,0 ;column
    mov cx,1
    second_char_start:
        cmp count,9
        je skip_second_char1
        cmp count,1
        je skip_second_char1
    skip_second_char2:
        mov al,char2
        jmp start_loop_second_char
    skip_second_char1:
        mov al,char1
    start_loop_second_char:
        CALL DisplayColour
        inc dl ;next coloumn
        dec count
        cmp count,0
        jne second_char_start
        ret
second_char endp
```

8.7.3 third_char function

```
third_char proc
    mov count,9
    mov dl,0 ;column
    mov cx,1
    third_char_start:
        cmp count,9
        je skip_third_char1
        cmp count,1
        je skip_third_char1
        cmp count,8
        je skip_third_char2
        cmp count,2
        je skip_third_char2
    skip_third_char3:
        mov al,char3
        jmp start_loop_third_char
    skip_third_char2:
        mov al,char2
        jmp start_loop_third_char
    skip_third_char1:
        mov al,char1
    start_loop_third_char:
        CALL DisplayColour
        inc dl ;next coloumn
        dec count
        cmp count,0
        jne third_char_start
        ret
third_char endp
```

8.7.4 fourth_char function

```
fourth_char proc
   mov count,9
   mov dl,0 ;column
   mov cx,1
   fourth_char_start:
      cmp count,9
      je skip_fourth_char1
      cmp count,1
      je skip_fourth_char1
      cmp count,8
      je skip_fourth_char2
      cmp count,2
      je skip_fourth_char2
      cmp count,7
      je skip_fourth_char3
      cmp count,3
      je skip_fourth_char3
   skip_fourth_char4:
      mov al,char4
      jmp start_loop_fourth_char
   skip_fourth_char3:
      mov al,char3
      jmp start_loop_fourth_char
   skip_fourth_char2:
      mov al,char2
      jmp start_loop_fourth_char
   skip_fourth_char1:
      mov al,char1
   start_loop_fourth_char:
      CALL DisplayColour
```

```
        inc dl ;next coloumn
        dec count
        cmp count,0
        jne fourth_char_start
        ret
fourth_char endp
```

8.7.5 fifth_char function

```
fifth_char proc
   mov count,9
   mov dl,0 ;column
   mov cx,1
   fifth_char_start:
      cmp count,9
      je skip_fifth_char1
      cmp count,1
      je skip_fifth_char1
      cmp count,8
      je skip_fifth_char2
      cmp count,2
      je skip_fifth_char2
      cmp count,7
      je skip_fifth_char3
      cmp count,3
      je skip_fifth_char3
      cmp count,6
      je skip_fifth_char4
      cmp count,4
      je skip_fifth_char4
   skip_fifth_char5:
      mov al,char5
      jmp start_loop_fifth_char
   skip_fifth_char4:
      mov al,char4
      jmp start_loop_fifth_char
   skip_fifth_char3:
      mov al,char3
      jmp start_loop_fifth_char
```

```
    skip_fifth_char2:
        mov al,char2
        jmp start_loop_fifth_char
    skip_fifth_char1:
        mov al,char1
    start_loop_fifth_char:
        CALL DisplayColour
        inc dl ;next coloumn
        dec count
        cmp count,0
        jne fifth_char_start
        ret
fifth_char endp
```

8.7.6 boxTypePattern function

```
boxTypePattern proc
    CALL CLS
    mov ah,09h
    lea dx,messageBox
    int 21h
    mov  bl, 2  ;Color is red
    mov ah, 01h
    int 21h
    MOV char1, AL
    mov ah, 01h
    int 21h
    MOV char2, AL
    mov ah, 01h
    int 21h
    MOV char3, AL
    mov ah, 01h
    int 21h
    MOV char4, AL
    mov ah, 01h
    int 21h
    MOV char5, AL
    mov  dh, 1  ;Row
    call Newline
    call first_char
    inc  bl; change colour
    inc  dh; nex Row
    call Newline
    call second_char
    inc  bl; change colour
    inc  dh; nex Row
    call Newline
```

```
call third_char
inc  bl; change colour
inc  dh; nex Row
call Newline
call fourth_char
inc  bl; change colour
inc  dh; nex Row
call Newline
call fifth_char
inc  bl; change colour
inc  dh; nex Row
call Newline
call fifth_char
inc  bl; change colour
inc  dh; nex Row
call Newline
call fourth_char
inc  bl; change colour
inc  dh; nex Row
call Newline
call third_char
inc  bl; change colour
inc  dh; nex Row
call Newline
call second_char
inc  bl; change colour
inc  dh; nex Row
call Newline
call first_char
CALL RESET
CALL AGAIN
```

```
boxTypePattern endp
```

8.8 Nested Loop Pattern Function

In this pattern, there will be 4 different pattern to be printed based on user selection, all of them are using nested loop concept to achieve the output, the output will be colourized as well.

8.8.1 nestedLoopPattern01 function

```
nestedLoopPattern01 proc
   CALL CLS
   mov colour,10
   mov dh,1
   mov dl,0
   mov cx,1
   mov bx,5
   jmp NP01start
   NP01L1:
      dec bx
      CMP bx,4
      JE  NP01L2
      CALL Newline
      inc dh
      mov dl,0
   NP01L2:
      push dx
      push cx
      push bx
      mov bl,colour
      mov al,"*"
      mov cx,1
      CALL DisplayColour
      pop bx
      pop cx
      pop dx
```

```
        inc dl
    NP01start:
    loop NP01L2
    mov cx,7
    sub cx,bx
    CMP bx,0
    JE  nestedLoopPattern01_end ;jmp to end
    loop NP01L1
    nestedLoopPattern01_end:
        ret
nestedLoopPattern01 endp
```

8.8.2 nestedLoopPattern02 function

```
nestedLoopPattern02 proc
   mov colour,12
   mov cx, 7
   mov bx, 6
   mov dh,1
   mov dl,0
   jmp NP02start
   NP02L1:
      dec bx
      CALL Newline
      mov dl,0
      inc dh
   NP02L2:
      push dx
      push cx
      push bx
      mov bl,colour
      mov al,"*"
      mov cx,1
      CALL DisplayColour
      pop bx
      pop cx
      pop dx
      inc dl
   NP02start:
      loop NP02L2
      mov cx,bx
      loop NP02L1

      ret
nestedLoopPattern02 endp
```

8.8.3 nestedLoopPattern03 function

```
nestedLoopPattern03 proc
    mov colour,14
    mov cx,1
    mov bx,5
    mov count,1
    mov dh,1
    mov dl,0
    jmp NP03start
    NP03L1:
        dec bx
        CMP bx,4
        JE  NP03L2
        CALL Newline
        mov count,1
        inc dh
        mov dl,0
    NP03L2:
        mov al,count
        mov number,al
        mov al,number
        add al,48
        inc count
        push dx
        push cx
        push bx
        mov bl,colour
        mov cx,1
        CALL DisplayColour
        pop bx
        pop cx
        pop dx
```

```
        inc dl
    NP03start:
        loop NP03L2
        mov cx,7
        sub cx,bx
        CMP bx,0
        JE  nestedLoopPattern03_end
    loop NP03L1
    nestedLoopPattern03_end:
        ret
nestedLoopPattern03 endp
```

8.8.4 nestedLoopPattern04 function

```
nestedLoopPattern04 proc
     mov count,1
     mov cx, 7
     mov bx, 6
     mov number, 1
     mov dl,0
     mov dh,1
     jmp NP04start
    NP04L1:
        dec bx
        CALL Newline
        inc count
        mov al,count
        mov number,al
        inc dh
        mov dl,0
    NP04L2:
        mov al,number
```

```
        add al,48
        inc number
        push bx
        push cx
        push dx
        mov cx,1
        mov bl,colour
        CALL DisplayColour
        CALL increase_colour
        pop dx
        pop cx
        pop bx
        inc dl
    NP04start:
        loop NP04L2
        mov cx,bx
        loop NP04L1
nestedLoopPattern04 endp
```

8.9 Main Function

```
Main proc
    start:
        mov ax,@data
        mov ds,ax
        call main_menu
    end:
        mov ah,4ch
        int 21h
Main endp
end main
```

9.0 Application limitation

There are several limitations in the application provided, which in the box type patterns only 5 characters is allowed to be input and must be 5 characters. And in the design pattern there are no colour output which can be improve or enhance in the future. But the application is working properly without any bugs and can be modify easily by future modifier for their new events.

10.0 Conclusion

In a conclusion, this study has been brought out the contribution of assembly language in recent applications by providing 5 recent application that are using assembly language, also the difference of low-level and high-level programming language and their particular role in cyber security/ forensic field, and also how reverse engineering help in steganography and the usage of IDA tools kit. Last but not least, this study provides an application and its source code as the example of using assembly language.

11.0 Self-Reflection

Throughout the research and analysis, me myself as the author of this study has learn a lot about the usage of assembly language and why it is still important nowadays especially in the cyber security field, also even though high-level language programming are mostly used in nowadays, but assembly language are still stand in an important role in this technology era. Also, I have learnt about the use of reverse engineering in steganography and how IDA tools kit provides the help in a convenient way to help reverse engineers to do their jobs. Also, by an application using assembly language are also a fun way to learn about how assembly language work such as different register has different meaning, and different interruption can cause different output. As for my own application I hope there will be enhancement such as clearer coding styles, or maybe implement object-oriented programming concept into assembly as I don’t know will it able to do that or not. I hope in the future I can learn more about how to use assembly language in ethical hacking, which is more on attacking side. Last but not least, thanks for anyone reaching this study to the end.

12.0 References

A, A. (2021). Reverse Engineering Research. -, 3.

Advanced Industrial Control Technology. (2010). Device Driver. -, 2.

Al-Aubidy, P. K. (2011). Computer Software Requirements for Real-Time Applications. -, 13.

Bhojani, N. (2014). Malware Analysis. -, 2.

Byju community. (n.d. n.d., n.d.). *Difference Between Assembly Language and High-Level Language*. Retrieved from BYJU's Exam Prep: https://byjus.com/gate/difference-between-assembly-language-and-high-level-language/#:~:text=The%20assembly%20language%20is%20a,machine-independent%20type%20of%20language.&text=It%20makes%20use%20of%20the,the%20English%20statements%20for%20operation.

Ebrahim, A. (n.d. n.d., 2010). *Assembly Language and Embedded Systems Development,*. Retrieved from mathscitech: http://mathscitech.orgarticlesassembly-value

Electronics Hub Organization. (25 November, 2017). *8051 Microcontroller Assembly Language Programming*. Retrieved from Electronicshub Logo: https://www.electronicshub.org/8051-microcontroller-assembly-language-programming/#:~:text=Assembly%20Language%20is%20a%20pseudo,the%20architecture%20of%20the%20Microcontroller.

Faizan, A. (2 February, 2021). *The Best Programming Languages for Cybersecurity in 2021*. Retrieved from flatironschool.com: https://flatironschool.com/blog/best-programming-languages-cyber-security/

Lee, H., & Lee, H.-W. (2020). New Approach on Steganalysis: Reverse-Engineering based Steganography SW Analysis. *Association for Computing Machinery*, 212, 213.

lo4d. (06 May, 2011). Retrieved from Emu8086: https://emu8086.en.lo4d.com/windows#:~:text=Tutorial....-,Emu8086%20is%20a%20Microprocessor%20Emulator%20with%20an%20integrated%208086%20Assembler,memory%20and%20input%2Foutput%20devices.

n.d. (n.d.). Advantages and Disadvantages of Using Assembly Language in Embedded Systems. -, 1.

Pedamkar, P. (n.d. n.d., n.d.). *What is Assembly Language?* Retrieved from EDUCBA: https://www.educba.com/what-is-assembly-language/

Ramdas, M. N. (2019). Basic Fundamental of Python Programming Language and the Bright Future. *AN INTERNATIONAL MULTIDISCIPLINARY QUARTERLY RESEARCH JOURNAL*, 72.

Rose, K. (n.d.). The 8051 Microcontroller. -, 1.

SHARMA, R. (20 April, 2021). *8051 Microcontroller*. Retrieved from electronics desk logo: https://electronicsdesk.com/8051-microcontroller.html

TEACH COMPUTER SCIENCE Corporation. (n.d. n.d., 2022). *Device Drivers.* Retrieved from TEACH COMPUTER SCIENCE: https://teachcomputerscience.com/device-drivers/

TechTarget Contributor. (n.d. April, 2008). *real-time application (RTA).* Retrieved from TECHTARGET NETWORK: https://www.techtarget.com/searchunifiedcommunications/definition/real-time-application-RTA#:~:text=A%20real%2Dtime%20application%20(RTA,value%2C%20usually%20measured%20in%20seconds.

Williams, L. (24 February, 2022). *Difference between Microprocessor and Microcontroller*. Retrieved from Guru99: https://www.guru99.com/difference-between-microprocessor-and-microcontroller.html