A picture containing graphics, graphic design, astronomy, art

Description automatically generated

**INDIVIDUAL ASSIGNMENT**

**TECHNOLOGY PARK MALAYSIA**

**CT073-3-2-CSLLT**

**COMPUTER SYSTEMS AND LOW LEVEL TECHNIQUES**

|  |  |  |
| --- | --- | --- |
| **Name** | **:** | Jacky Wong Chun Kit |
| **Intake Code** | **:** | APD2F2211CS(CYB) |
| **Project Title** | **:** | CyberFresh Tech Inventory Management System |
| **Date Assigned** | **:** | 22 May 2023 |
| **Date Completed** | **:** | 10 August 2023 |
| **Weightage** | **:** | 50% |

**A picture containing text, screenshot, font

Description automatically generated**

**Table of Contents**

[1 Introduction on Assembly Language 3](#_Toc142169575)

[2 Research and Analysis 4](#_Toc142169576)

[2.1 Importance And Utilization of Assembly Language 4](#_Toc142169577)

[2.2 Utilization of Low-level Language in Cybersecurity and Forensics Field 5](#_Toc142169578)

[3 System Design 9](#_Toc142169579)

[4 System Screenshot 10](#_Toc142169580)

[4.1 Program Compilation Stage 10](#_Toc142169581)

[4.2 System Main Menu 13](#_Toc142169582)

[4.3 View Inventory 14](#_Toc142169583)

[4.4 Order Device 14](#_Toc142169584)

[4.5 Sell Device 16](#_Toc142169585)

[4.6 Sort Inventory By Categories 18](#_Toc142169586)

[4.7 Generate Report 19](#_Toc142169587)

[4.8 Exit Program 20](#_Toc142169588)

[5 Conclusion 21](#_Toc142169589)

[6 References 22](#_Toc142169590)

[7 Appendices 23](#_Toc142169591)

# Introduction on Assembly Language

A low-level programming language called Assembly Language aids to deliver an immediate representation and understanding of machine code instructions (Pedamkar, 2023). It acts as a type of machine language that’s human-understandable, applying mnemonic commands and symbolic labels to describe the program’s operations and logics. However, it’s regarded as a challenge to employ as they demand an in-depth understanding of machine language. Therefore, the programmers which expertise in Assembly Language can perform instructions at most basic level and exercise fine-grained dominate over hardware. Additionally, developers design it for specific processors and might create it by compiling the source codes or writing it from zero. It’s such a wonderful tool for programmers which looking for in-depth comprehension and control because they can learn insightful stuffs about the operations of the processors inside a computer because by digging into its complexities. A wide variety of amazing tasks can be accomplished by the programmers by integrating numerous basic code operations into workable software. Several terms that needed to be comprehensive in obtaining the concept of low level programming language are interpreter which is a program that reads and runs code, and compiler which is a translator that translates the high-level language code to low-level language object code. Through this assignment, the assembler interpreter tool that utilized for creating the novel CyberFresh Tech Inventory Management System will be Turbo Assembler (TASM) Programming Language version 1.4 (8086) and the source code from the assembler file will be displayed and justified accordingly.

# Research and Analysis

## Importance And Utilization of Assembly Language

Assembly Language as the low-level language plays an important role in numerous aspects of hardware communication and computer programming nowadays. Although it’s less user-friendly than high-level languages, it provides several importance aspects that proved it as an invaluable tool for both programmers and developers.

One of the primary importance of Assembly Language is its flexibility in improving system performance. Readable machine instructions can be converted one by one from assembler instructions. The computer processor will execute the provided code to accomplish specific user's system task requirements, and there are no additional loops or redundant functions bloat the produced assembler code (Schmidt, 2016). Since the assembly code always generated by hand for certain hardware, it attains levels of performance that higher-level languages won't because of the level of abstraction. Low-level abstraction which is the close-to-hardware abstraction of a computer system's fundamental hardware enables programmers to design a customized code that makes use of a certain computer system's hardware features (Kanade, 2023). For instance, sorting or searching algorithms can be written in the assembly language for opening more functionalities to the novel designed system. Additionally, Assembly Language provides the basis for comprehending computer operations at a mechanical level, which can be invaluable for beginners like students. Proficiency in assembly language can facilitate in understanding machine language, which is composed of binary commands (0s and 1s) that interpreted by the computer’s CPU (Team, 2017). The utilization of mnemonic instructions is brief and simple phrase to users that the computer’s CPU can grasp and that stand in for precise instructions. For instance, a common mnemonic instruction “MOV” represents ‘move’ in data movement, and is utilized to transfer data between two locations (Kanade, 2023).

The utilization of Assembly Language has been essential in numerous areas of computing, playing a crucial role in the optimization and deep knowledge to system functions. For instance, it’s usually utilized within the operation system, especially those that need direct hardware interfaces. It guarantees that the components run as effectively as possible and have full control over the system’s resources. The language’s influence is equally notable in video game creation nowadays. The early developers largely relied on it to obtain maximum performance when considering the limited hardware capabilities of the early generation’s gaming computer. Additionally, it has been essential in the broad field of embedded and real-time systems, which including anything from commonplace equipment, including microwave ovens to complex medical instructions (Team, 2017). The equipment usually employs microcontrollers which the firmware is designed in assembly code form for ensuring the effective use of scarce resources while being adapted for certain purposes.

## Utilization of Low-level Language in Cybersecurity and Forensics Field

Novice programmers frequently disregard Assembly Language, as it’s one of the most sophisticated programming languages in cybersecurity nowadays. It’s crucial to master this language if the programmers wish to develop an efficient cyber defence solutions or fully comprehend the cyber-attacks’ internal operations (Sharma, n.d.). The field of malware analysis is at the cutting-edge of the interaction. Majority of the latest dangerous programmes including viruses and trojans, are written in high-level languages, but they are distributed in compiled form. Hence, it’s necessary to disassemble them into assembler code to understand how they operate (Sikorski, 2012). Additionally, current malware frequently employs strategies including code evasion and perplexity to avoid detection. It highlights the need for computer analysts to be well-versed in low-level languages, prompting them to undo the evasive measures and restore the malware back to its unprocessed and unaltered state for further malware analysis (Brumley, 2005). Reverse engineering weaves a complex web in conjunction with malware analysis. Cybersecurity enthusiasts regularly conduct countless difficult job of disassembling software programmes, digging into their assembly representations to highlight any vulnerabilities that might be exploited maliciously (Eilam, 2005). When manufacturers release certain patches to repair the discovered vulnerabilities, a huge challenge arises that the attackers could painstakingly reverse engineer the patches, learning more about flows, and designing plans to attack victims who aren’t using updates immediately (Dullien, 2009). Furthermore, reverse engineering is very crucial in situations where software analysis must be conducted because a component of confidential software is suspected of having backdoors or other undocumented features. Moreover, forensics field relies largely on low-level language. Although it’s challenging in understanding binary-level file structures, but it’s essential for recovering lost or damaged information. In case a security breach had happened, forensics analysts can utilize their knowledge of low-level operations to perform a detailed study of the damaged system, assisting in detection of the attacker’s activities and retrieval of corrupted data.

*A screenshot of a computer

Description automatically generated*

*Figure 2.2.1 – CODE\_0 section for ASSG.EXE*

Ghidra is a disassembly tool created by NSA and last released in year 2019, allowing malware experts to analyse within malware code without running it plus providing details about its behaviour (Fox, 2022). The figure above (Refer to Figure 2.2.1) shows created assembler code after analysing the .EXE extension file through Ghidra. Based on the output of the analysed disassembled code, the code provided looks to be the DOS executable’s starting point and seems to be sending out certain kind of user input and comparison. According to the first code line, ‘1000:0000’ is the memory address with the instruction’s location, which containing the segment and offset within the segment, while ‘b8 3b 10’ are the hexadecimal codes representing the machine code that can be executed by the CPU. ‘b8’ represents ‘MOV’ (move) operation, and ‘3b 10’ represents ‘0x103b’ operand. Therefore, ‘MOV AX, 0x103b’ copies and contains the operand value into AX which representing an accumulator register.

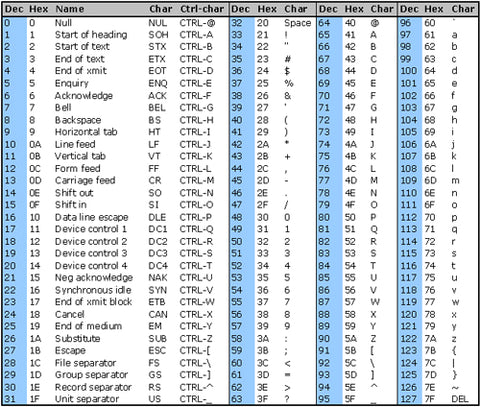
For the third code line, ‘e8’ stands for ‘CALL’ operation, while ‘1d 01’ stands for both ‘011d’ in hexadecimal and ‘285’ in decimal form. A subroutine which might be a method or function is called using the ‘CALL’ instruction. Therefore, ‘FUN\_1000\_0125’ which specifies address ‘1000:0125’ of a subroutine function will be called with instruction and returns at ‘1000:0008’ after the subroutine finished its execution.

For the seventh code line, ‘74’ stands for ‘JZ’ (Jump if Zero) which is one of the conditional jump operations, while ‘16’ stands for both hexadecimal and ‘22’ in decimal. ‘LAB\_1000\_0026’ is a label and it located at address ‘1000:0026’. ‘JZ’ operation verifies the zero flag in the CPU’s flag register. If the result of the last operation is zero after comparing value from AL, then the jump is made, otherwise the following next operation is conducted.

A screenshot of a computer

Description automatically generated

*Figure 2.2.2 – Data section for ASSG.EXE*



*Figure 2.2.3 – ASCII Hex to Dec Code Conversion Table* (ASCII Chart, n.d.)

It’s containing a section of ASCII-encoded text based on the data segment provided (Refer to Figure 2.2.2). The instance of offset addresses are stated from ‘103b:0014’ to ‘103b:0077’ for specifying memory location. The ‘??’ symbol means that this byte was deconstructed and it unable to identify a clear command for it. Therefore, it’s considering them as raw data rather than processing it as a command. The two-character hexadecimal values indicate the actual byte values kept at the memory addresses and classified as raw data values. Next, the characters after hexadecimal values are representations of ASCII bytes. If a hex byte matches to a printable ASCII character, then it can be displayed to the console screen. For instance, hexadecimal ‘50h’ represents character ‘P’ and ‘61h’ represents ‘a’.

# System Design

A diagram of a flowchart

Description automatically generated

# System Screenshot

## Program Compilation Stage

A screenshot of a computer

Description automatically generated

*Figure 4.1.1 – Program Creation with ‘edit’ command*

The assembly language source code is firstly created or altered at this step. ‘edit’ command typically opens the specified .asm file in DOS TASM text editor when utilizing MS-DOS.

A computer screen shot of a blue screen

Description automatically generated

*Figure 4.1.2 – Written Source Code*

The assembly source code is written based on the text editor given and ready for compilation.

A screenshot of a computer

Description automatically generated

*Figure 4.1.3 – Program Compilation with Assembly Compiler*

*A screenshot of a computer

Description automatically generated*

*Figure 4.1.4 – Generated Object File (filename.obj)*

The assembly code that written in the .asm file is converted and done compiled to machine language object code using TASM 1.4 at this step. As a result, an object file with .obj extension is generated.

A screenshot of a computer

Description automatically generated

*Figure 4.1.5 – Program Linking with Assembly Linker*

*A screenshot of a computer

Description automatically generated*

*Figure 4.1.6 – Generated Application and Linker Address Map (filename.EXE and filename.MAP)*

Linking is the process of merging one or more object files and addressing references using Turbo Linker (TLINK). As a result, an executable program file and linker address map are produced for DOS environment. In case there is any linking error, it should be fixed before the executable is generated.

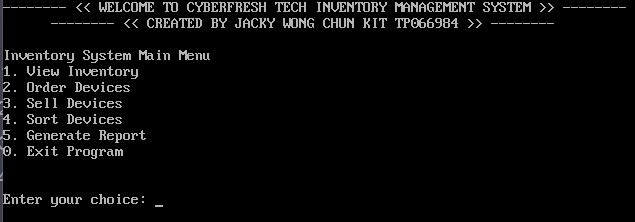
A screenshot of a computer

Description automatically generated

*Figure 4.1.7 – Program Execution*

The created executable program can be run, and the desired output or operations can be performed.

## System Main Menu



*Figure 4.2.1 – Main Menu Interface for User*

During the program execution, the user has been led to the main menu first. It’s designed to be an intuitive and user-friendly interface which providing clear functionalities of the system. There are a total of 5 functionalities for the user to prompt accordingly, which including view, order, sell, sort devices inside the inventory, and generate report based on the sales of the day. Additionally, user can exit the program safely as well by an input away.

A screen shot of a computer

Description automatically generated

*Figure 4.2.2 – Invalid Input Message for User*

A self-explanatory message will be outputted to the user when they have prompted an invalid input. They will be instructed to enter any key to return to the main menu. This output is designed to preserve a seamless user experience and avoid program failure.

## View Inventory

A screenshot of a computer

Description automatically generated

*Figure 4.3.1 – View Inventory Interface for User*

If user inputs ‘1’ at the main menu, then the user is led to the View Inventory interface. User able to view the inventory list that currently available for sale in the company. Devices’ ID, name, quantity, and unit price are displayed to user in ease with table form. If the items that are less than 5 in quantity, items are highlighted in red with blinking effect for alerting the user that low in stock for items and require ordering from vendor. After viewing the inventory, the user is given choices of functionalities such as ordering, selling devices, and back to main menu as well.

## Order Device

A screenshot of a computer

Description automatically generated

*Figure 4.4.1 – Order Device Interface for User*

If user inputs ‘2’ at the main menu or any similar user navigation interface, then the user is led to the Order Device interface. User needs to input the device ID to order from vendor by reference to the IDs that listed in the inventory table.

A screenshot of a computer screen

Description automatically generated

*Figure 4.4.2 – Order Quantity Prompt from User*

User must input an order quantity between 1 to 9 after inputting the specific device ID. It’s assumed that user should submit input in the form of integer value. Since the system does not yet impose input type restriction, any non-integer input such as a character might resulted in unexpected system behaviour or problem during displaying the expected device quantity to user. Therefore, this assumption is crucial for the user to go by the instruction while inputting the quantity to guarantee that the system works properly.

A screenshot of a computer

Description automatically generated

*Figure 4.4.3 – Order Device Success for User*

After inputting the order quantity, then the user is notified with an order device success message that the device has been ordered successfully. As can see from the figure (Refer to Figure 4.4.3), the webcam device quantity is increased from 2 to 11. It also indicates that the system able to add the quantity to the current inventory quantity automatically. The webcam device quantity is not highlighted with red anymore as the quantity is greater than 5. Additionally, the system is assumed to allow user to process bulk device replenishments in the inventory which replenishing within 10 for all inventory stock quantities that less than 5. After done ordering device, the user is allowed to continue order device, sell device, or back to main menu by specific input.

## Sell Device

A screenshot of a computer

Description automatically generated

*Figure 4.5.1 – Sell Device Interface for User*

If user inputs ‘3’ at the main menu or any similar user navigation interface, then the user is led to the Sell Device interface. User needs to input the device ID to sell by reference to the IDs that listed in the inventory table.

A screenshot of a computer screen

Description automatically generated

*Figure 4.5.2 – Sell Quantity Prompt from User*

User must input a sell quantity between 1 to 9 after inputting the specific device ID. The same assumptions are made as with Order Device Interface to ensure that the system able to read the intended quantity from user correctly.

A screenshot of a computer

Description automatically generated

*Figure 4.5.3 – Sell Device Failed for User*

If the inputted sell quantity is larger than the inventory quantity for the selected device, a sell device failure message is issued to the user, stating that the device quantity is too low to sell. Hence, user must repeat the step and adjust the sell quantity if user still insists to sell device.

A screenshot of a computer

Description automatically generated

*Figure 4.5.4 – Sell Device Success for User*

If the inputted sell quantity is lesser than the inventory quantity for the selected device, a sell device success message is issued to the user, stating that the device has been sold successfully. As can see from the figure above (Refer to Figure 4.5.4), the headphone device quantity is decreased from 2 to 0. It also indicates that the system able to produce the item sale by auto deducting the quantity from the current inventory quantity. As the quantity is 0, therefore the headphone device quantity is still highlighted with red which indicating that a restock action is required from user. Furthermore, the system is assumed to allow user to sell multiple device items at a time. After done selling the device, user is allowed to continue sell device, order device, or back to main menu by specific input.

## Sort Inventory By Categories

A black screen with white text

Description automatically generated

*Figure 4.6.1 – Sort Inventory Interface for User*

If user inputs ‘4’ at the main menu, then the user is led to the Sort Inventory interface. User able to categorize the inventory based on specific criteria including finished stock devices and stocks need to order. User needs to enter a specific key to enable the system to sort the inventory based on user’s requirements.

A screenshot of a computer

Description automatically generated

*Figure 4.6.2 – Finished Stock Devices for User*

If user inputs ‘1’ at the Sort Inventory interface, a list of devices with a recorded inventory quantity of 0 will be displayed. After viewing the sorted inventory list, user can input choice whether continue to order, sell devices, or back to the main menu.

A screenshot of a computer

Description automatically generated

*Figure 4.6.3 – Finished Stock Devices for User*

If user inputs ‘2’ at the Sort Inventory interface, a list of devices with a recorded inventory quantity of less than or equal to 5 will be displayed. After viewing the sorted inventory list, user can input choice whether continue to order, sell devices, or back to the main menu.

## Generate Report

A screenshot of a computer screen

Description automatically generated

*Figure 4.7.1 – Generate Report Interface for User*

If user inputs ‘5’ at the main menu, then the user is led to the Generate Report interface. User able to view all the devices that have been sold during the program execution. Devices’ ID, name, sold quantity, unit price, total earned for each device are displayed to user in ease with table form. The report is assumed that all items are set to default as 0 as the sold quantity in the inventory system so that user can clearly view which device that has been sold through the sales report generated. Additionally, the system is assumed to enable the user to view the generated report from time to time after the devices have been sold and recorded. Since the headphone devices have been sold 2 units (Refer to Figure 4.5.4), therefore the corresponding sold quantity is listed as 2 units as well. Hence, the calculation result of multiplying the sold quantity and the unit price is 16 as the total earned. After viewing the report, user can simply back to main menu or exit the program with specific input.

## Exit Program

A black screen with white text

Description automatically generated

*Figure 4.8.1 – Exit Program Interface for User*

If user inputs ‘0’ at the main menu or any user navigation interface, then the user is led to the Exit Program interface. The program is terminated and outputs a thank message to user for using the program.

# Conclusion

In conclusion, this assignment provided a valuable opportunity to investigate the complex yet essential nature of assembly language, its significance in several sectors, and its crucial role in the study of cybersecurity and forensics fields nowadays. The demonstration of how the direct hardware manipulation capabilities of assembly language might produce greatly effective and successful software can be clearly shown by creating a successful inventory management system. The strengths and versatilities of assembly language were also clearly discovered through this assignment by the inclusion of features like automated inventory tracking, automated alert of low stock quantity, and smart inventory restocking and reselling. Furthermore, this assignment demonstrated the assembly language’s adaptability by showing that it can be applied for higher-level programmes like the inventory management system in addition to system-level functionalities. Furthermore, this assignment aided to get into a greater understanding of low-level programming language ideas and how computer systems operate.

# References

*ASCII Chart*. (n.d.). Retrieved from CommFront: https://www.commfront.com/pages/ascii-chart

Brumley, D. B. (2005). Remote Timing Attacks are Practical. *Computer Networks*, 13.

Dullien, T. P. (2009). REIL: A platform-independent intermediate representation of disassembled code for static code analysis. 7.

Eilam, E. (2005). *Reversing: Secrets of Reverse Engineering.* Wiley Publishing.

Fox, N. (2022, March 21). *How to Use Ghidra to Reverse Engineer Malware*. Retrieved from Varonis: https://www.varonis.com/blog/how-to-use-ghidra

Kanade, V. (2023, June 15). *What Is Assembly Language? Working, Features, and Advantages*. Retrieved from spiceworks: https://www.spiceworks.com/tech/tech-general/articles/what-is-assembly-language/

Pedamkar, P. (2023, May 18). *What is Assembly Language?* Retrieved from EDUCBA: https://www.educba.com/what-is-assembly-language/

Schmidt, G. (2016). *Beginners Introduction to the Assembly Language of ATMELAVRMicroprocessors*. Retrieved from : http://www.avr-asm-download.de/beginner\_en.pdf

Sharma, A. (n.d.). *6 Best Cyber Security Programming Languages To Learn in 2022*. Retrieved from Hackr.io: https://hackr.io/blog/best-cyber-security-programming-languages

Sikorski, M. H. (2012). *Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software.* San Francisco: William Pollock.

Team, T. L. (2017, July 4). *Understanding Assembly Language: Importance, Pros & Cons*. Retrieved from Tweak Library: https://tweaklibrary.com/importance-of-assembly-language/

# Appendices

A white sheet with black text

Description automatically generated