This document focuses on the excellent achievement of the unit learning outcomes of the SIT111 module from week 1 to week 10. For the aim of presenting my comprehension of the material, I will succinctly outline the key ideas covered each week. The basis for the complicated concepts that will be covered in the following weeks was set in week one. In the first week, the main topics covered were the principles of computer systems, such as binary numbers, or bits, such as 1s and 0s, and boolean logic. The boolean logic was then implemented into virtual chips using a hardware language called Hardware Description Language, which was created specifically for this purpose. Since this language is declarative, code implementation designs are transformed into functionally-deficient structured code. The names of the chip, input, and output pins are specified in the headers of the HDL code file, and the code logic is implemented in the parts section. Next, we go to week 2, when I first encountered a hardware simulator. This hardware simulator generates the virtual chip picture by first receiving an HDL file as input. A test script is used to test this chip, and the results are compared to the compare file. Additionally, there are three different kinds of simulations: behavioral, script-based, and interactive. The goal of interactive simulation is to manually alter the inputs and observe if the outputs follow the chip's logic. I was able to test the accuracy of the chip utilizing an automated script testing method thanks to script-based simulation. The field of behavioral simulation involves advanced programming language skills. Thus, this week assisted me in progressively expanding my understanding of this course and illustrating the usefulness of hardware simulators. Let's go back to week 3, when I learned how computers construct arithmetic and logical functions using the ALU. Six control bits are used by the ALU to compute the calculation, which determines the kind of computation that will be done on the input. I was also given an explanation of how the ALU functions within. The fourth week then included insights on clocks, flip flops, registers, and sequential and combined logic circuits. The output of a sequential logic circuit depends on both the current inputs and the input history, whereas the output of a combinational logic circuit depends only on the current inputs. Flipflops are simple state-keeping devices with some storage capacity. A register is a set of flip flops combined. Depending on how many registers are used, multibit values can be stored in registers. Following the introduction of each of these ideas in week 4, week 5 covers RAM and the Programme Counter (PC), another kind of register. Data and instructions are stored in RAM. The programme counter selects the next command to be carried out. It's challenging to understand the more complicated ideas that are based on these fundamental ideas if you don't understand these basic principles. The sixth week is devoted to keyboard and screen mapping. RAM memory spaces designated for each display block, or display memory, can be turned on and off to show material on the screen. Memory spaces in RAM set aside for each pushed key on the keyboard that corresponds to a unique key code is known as keyboard memory. Core ideas including branching, variables, and pointers are the subject of week seven. Branching is just analyzing the boolean logic and
moving to a different place depending on whether or not a jump condition is given. Variables are named and value-containing containers. It's important to remember that several kinds of variables and data types are present in high level languages. However, there is only one kind of variable found in assembly language, and that is a 16-bit variable. Pointers are variables whose value is the address in memory. Harvard and Von-Neumann architecture are the primary topics

of discussion in week eight. Whereas Harvard design keeps data and instructions in different memory locations, Von-Neumann architecture keeps both in one. In week nine, students learn how to work with symbols, directions, and white spaces in an assembly programme. The assembler ignores white spaces—comments, in-line comments, and indentations—when translating them into machine code. As the course module came to an end in week 10, I had acquired all the knowledge necessary to finish it exceptionally well. By the time this module ended, I had also obtained practical experience with the instructional resource nand2tetris.

External Resources

1. Introduction to Computer System/1 INTRODUCTION TO COMPUTER SYSTEM. (n.d.). Available at: https://www.tmv.edu.in/pdf/Distance_education/BCA%20Books/BCA%20I%20SEM/BCA- 121%20Computer%20Fundamental.pdf [Accessed 22 Jan. 2024]. chrome-
2. extension://efaidnbmnnnibpcajpcglclefindmkaj/https://ncert.nic.in/textbook/pdf/kecs101.pdf
3. Bryant, R. and O'hallaron, D. (n.d.). *A Programmer's Perspective*. [online] Available at: https://www.cs.sfu.ca/~ashriram/Courses/CS295/assets/books/CSAPP_2016.pdf [Accessed14Feb.2024].
4. Edwards, S. (n.d.). *Fundamentals of Computer Systems Thinking Digitally*. [online] Available at: https://www.cs.columbia.edu/~sedwards/classes/2020/3827-summer/intro.pdf [Accessed 31 Dec. 2023].
5. Learning objectives. (n.d.). Available at: https://assets.cambridge.org/97813165/00743/excerpt/9781316500743_excerpt.pdf.
6. www.youtube.com. (n.d.). *Neso Academy - YouTube*. [online] Available at: https://www.youtube.com/@nesoacademy.