UNIVERSITI TUNKU ABDUL RAHMAN

FACULTY OF INFORMATION & COMMUNICATION TECHNOLOGY

GROUP ASSIGNMENT SESSION 202305

BACHELOR OF COMPUTER SCIENCE (HONS)

UCCD1133 INTRODUCTION TO COMPUTER ORGANISATION AND ARCHITECTURE

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1 Introduction

Those who work in the fields of computer programming and processor design really need to have the ability to develop assembler programmes that are effective and can be used for a variety of purposes. Computer programing is defined as the process of writing a set of instructions called a program that directs the computer to perform a specific task. Besides that, a modern processor has more than one instruction, and these are encoded as machine code which the processor only can reads from memory to manipulate the data, read and write to memory. *Southwell, S. (n.d.).* Furthermore, there will be six equations included in this assembly code that we are currently building. The user will be requested to enter the needed input value before the calculation is conducted and the result is shown. Three of the six equations in the set contain two different logical operations and need the user to input three values. These equations are part of the set that was given to the user. In addition to that, each of the other three equations involves two different arithmetic operations, and it is the responsibility of the user to give three different numbers as input. In addition to this, there will be a flowchart and pseudocode included in the documentation that defines the programme as well as the instruction set (instruction class, instruction format), registers, system call, and addressing modes that are used.

In addition, after the application has been created, there will be an explanation of a central processing unit (CPU) that is not a MIPS processor. MIPS processor is a measurement of the speed of the processor. *(Sheldon, R. 2023, March 1)* To provide just a few examples of processor architectures, there is ARM, SPARC, x86, and so on and so on. Since the MIPS design is quite widely known and used, we are going to investigate other alternative processor designs in order to broaden our perspective and broaden our options. After that, a comparison and in-depth description one of the MIPS processors which is MIPS32 CPU in regard to the processor that was selected will be supplied. When we have a better understanding of the complex world that is the basis of modern computing, it is because we have a greater awareness of the features, strengths, and weaknesses of the many designs of processors. For example, when we have a better knowledge of the intricate world that is the foundation of modern computing, we have a better knowledge of the various designs of processors. Finally, we are decided to explain and understand what ARM architecture is.

Chapter 2 Part A – Assembly language with detailed description

2.1 Assembly Code

```
.data
A:      .word 0
B:      .word 0
C:      .word 0
Result: .word 0
Prompt: .asciiz "Enter value: "
Menu:   .asciiz "Select an equation (1-6):\n1.
(A AND B) OR C\n2. (A OR B) AND
(NOT C)\n3. (A XOR B) AND C\n4. (A +
B) - C\n5. (A * B) + C\n6. (A / B) + (A %
B) * C\n"
InvalidChoiceMsg: .asciiz "Invalid choice.
Please enter a number again.\n"
ZeroDivisionError: .asciiz "Error: Division
by zero\n"
ResultText: .asciiz "Result: "
ContinueMsg: .asciiz "\n\nDo you want to
continue? (1: Yes, 2: No): "

.text
.globl main

main:
   li $v0, 4
   la $a0, Menu
   syscall

input_loop:
   li $v0, 4
   la $a0, Prompt
   syscall

   li $v0, 5
   syscall
   move $t0, $v0

   li $t1, 1
   li $t2, 6
   blt $t0, $t1, invalid_choice
   bgt $t0, $t2, invalid_choice

   beq $t0, 1, equation
   beq $t0, 2, equation
   beq $t0, 3, equation
   beq $t0, 4, equation
   beq $t0, 5, equation
   beq $t0, 6, equation
j continue

invalid_choice:
   li $v0, 4
   la $a0, InvalidChoiceMsg
   syscall
   j input_loop

equation:
   li $v0, 4
   la $a0, Prompt
   syscall
   li $v0, 5
   syscall
   sw $v0, A

   li $v0, 4
   la $a0, Prompt
   syscall
   li $v0, 5
   syscall
   sw $v0, B

   li $v0, 4
   la $a0, Prompt
   syscall
   li $v0, 5
   syscall
   sw $v0, C

   lw $t1, A
   lw $t2, B
   lw $t3, C

   beq $t0, 1, eq1
   beq $t0, 2, eq2
   beq $t0, 3, eq3
   beq $t0, 4, eq4
   beq $t0, 5, eq5
   beq $t0, 6, eq6

eq1:
   and $t4, $t1, $t2
   or $t4, $t4, $t3
   j done
```

```
eq2:
   or $t4, $t1, $t2
   not $t3, $t3
   and $t4, $t4, $t3

eq3:
   xor $t4, $t1, $t2
   and $t4, $t4, $t3
   j done

eq4:
   add $t4, $t1, $t2
   sub $t4, $t4, $t3
   j done

eq5:
   mul $t4, $t1, $t2
   add $t4, $t4, $t3
   j done

eq6:
   beqz $t2, divide_zero
   div $t1, $t2
   mflo $t4
   mfhi $t5
   mul $t5, $t5, $t3
   add $t4, $t4, $t5
   j done

divide_zero:
   li $v0, 4
   la $a0, ZeroDivisionError
   syscall
   j continue

done:
   li $v0, 4
   la $a0, ResultText
   syscall
   move $a0, $t4
   li $v0, 1
   syscall
j continue

continue:
   li $v0, 4
   la $a0, ContinueMsg
   syscall
   li $v0, 5
   syscall
   move $t0, $v0
   beq $t0, 1, show_menu  # Show the main menu if user chooses to continue
   beq $t0, 2, exit_program  # Exit the program
   j continue  # Repeat the "done" loop if an invalid choice is made

show_menu:
   li $v0, 4
   la $a0, Menu
   syscall
   li $v0, 5
   move $t0, $v0
   j input_loop  # Jump to the input loop to get the user's choice for equation

exit_program:
   li $v0, 10
   syscall
```

2.2 Flowchart

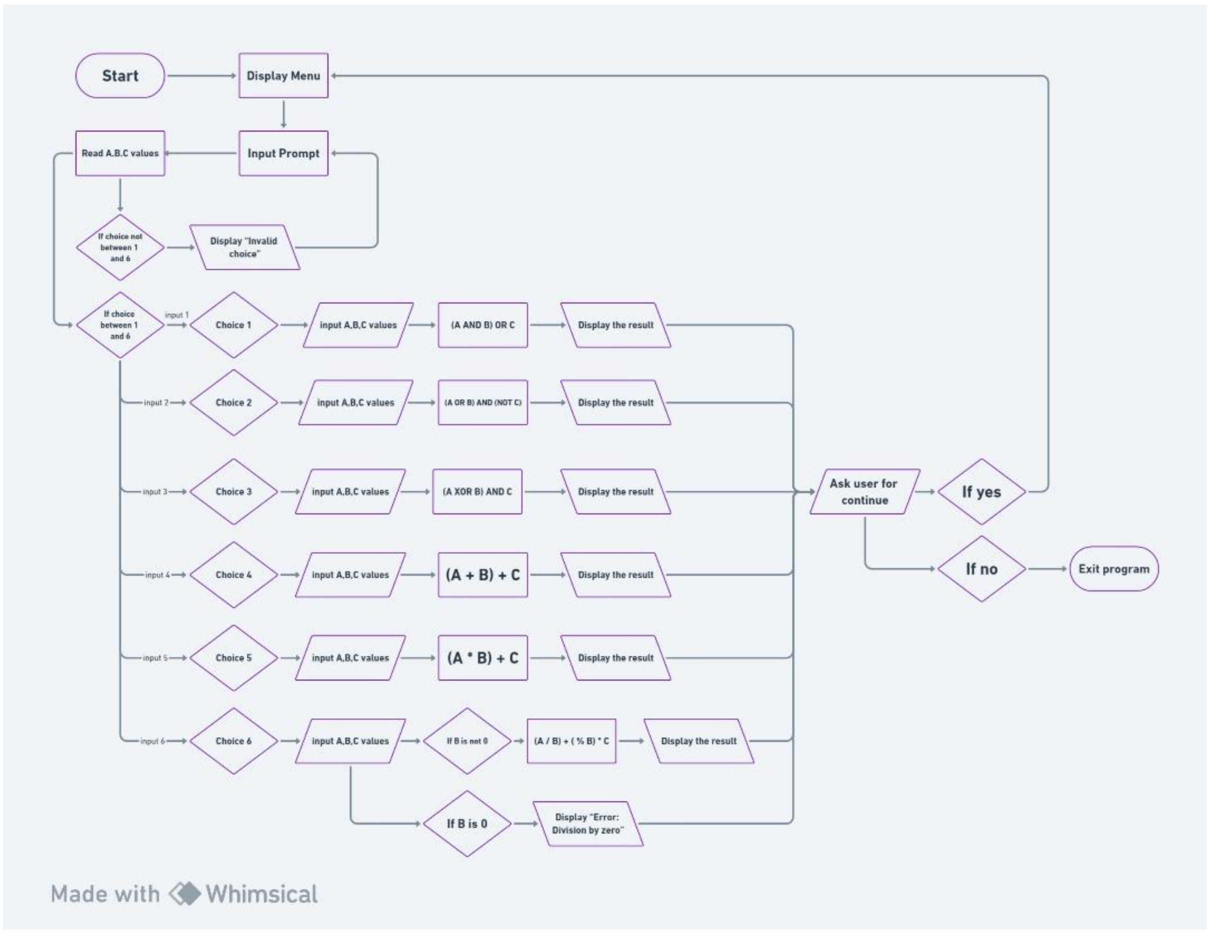


Figure 1 Flowchart of MIPS assembly code

2.3 Psuedocode

```
Data Section:
A, B, C, Result: Initialize to 0
Prompt: "Enter value: "
Menu: Display equation menu options
InvalidChoiceMsg: "Invalid choice. Please enter a number again."
ZeroDivisionError: "Error: Division by zero"
ResultText: "Result: "
ContinueMsg: "Do you want to continue? (1: Yes, 2: No): "

Main Function:
  Display Menu
  Loop:
    Display "Enter value: "
    Get User Input and store in t0
    Check if t0 is between 1 and 6
    If not, display InvalidChoiceMsg and repeat loop
    If valid choice, proceed to equation selection

  Equation Function:
    Depending on t0 value (equation choice), perform the following steps:
    1. (A AND B) OR C
    2. (A OR B) AND (NOT C)
    3. (A XOR B) AND C
    4. (A + B) - C
    5. (A * B) + C
    6. (A / B) + (A % B) * C

  Handle Divide by Zero:
    Check if division by zero would occur
    If so, display ZeroDivisionError message and return

  Display Result:
    Display "Result: " followed by the computed result
    Display ContinueMsg
    Get User Input (1 for Yes, 2 for No)
    If 1, return to the main menu
    If 2, exit the program

  Continue Loop:
    Display Menu
    Get User Input (1-6)
    Repeat the entire process
```

2.4 Code Explanation

This MIPS assembly program consists of six mathematical equations: AB+C, A+B-C, A-BC, (A and B) XOR C, (A and B) OR C, and (A NOR B) AND C. Users are given the option to choose which equation to execute. The program begins by displaying the menu of equations using the Menu string. It then reads the user's choice via a system call (syscall) and stores it in register $t0. To ensure the user's input is within the valid range of integer numbers from 1 to 6, the program performs a series of conditional checks. If the input is not valid, it displays an error message (InvalidChoiceMsg) and redirects the user to the input loop (input_loop). Once a valid choice is made, the program proceeds to read three integer values: A, B, and C, prompting the user with appropriate messages and storing the values in registers $s1 (A), $s2 (B), and $s3 (C).

After successfully reading the values, the program uses a series of conditional branches to determine which equation to execute based on the user's choice. Each equation is implemented in its respective label (eq1 to eq6) and performs the necessary mathematical operations. The results are stored in $t4. Following the calculation, the program displays the result using the ResultText string followed by the actual result stored in $t4. Finally, the program asks the user if they want to continue or exit. If the user chooses to continue (by entering 1), the program loops back to the main menu. In contrast, if the user chooses to exit (by entering 2), the program displays a farewell message using the ContinueMsg string and terminates. This code provides a user-friendly, menu-driven interface for executing mathematical equations, ensuring input validation and error handling.

Chapter 3 Part B – Overview on a processor

3.1 An overview of ARM Architecture

The ARM (Advanced RISC Machine) processor, designed by ARM Holdings, a British semiconductor company, stands as a testament to cutting-edge microprocessor technology. This overview will delve into the key characteristics, architecture, and applications of ARM processors, with references to specific case studies and credible sources to ensure clarity and reliability.

ARM is a family of reduced instruction set computing (RISC) architectures and microprocessor designs. Its primary purpose is to provide energy-efficient and high-performance computing solutions across a wide range of applications, including mobile devices, embedded systems, servers, and supercomputers *(ARM Holdings, n.d.)*. ARM's goal is to deliver a versatile and power-efficient architecture that can be customized for various computing needs while maintaining compatibility and scalability.

ARM architecture comprises a set of CPU designs, instruction sets, and related technologies. At its core, ARM offers a diverse range of CPU cores tailored to different computing needs. For example, the Cortex-A series provides high-performance application processors suitable for smartphones and servers, while the Cortex-R series targets real-time embedded systems. Meanwhile, the Cortex-M series excels in microcontroller applications *(ARM Developer, n.d.)*. ARM's instruction sets are renowned for their simplicity and efficiency, featuring a fixed-length format that aids in power efficiency and improved performance.

Furthermore, ARM-based System-on-Chip (SoC) designs are ubiquitous, combining CPUs with components like GPUs, memory controllers, and peripherals on a single chip. Of note is the ARMv8-A architecture, ARM's latest offering, which introduces 64-bit computing capabilities while ensuring backward compatibility with existing 32-bit software. This addresses both high-performance and power-efficient computing needs.

One of ARM's defining features is its exceptional energy efficiency, particularly well-suited for environments where power consumption is critical, such as mobile devices and battery-operated gadgets *(Smith, 2020)*. ARM achieves this by optimizing instruction execution and minimizing power draw during idle states. This focus on energy efficiency significantly extends battery life, reduces heat generation, and facilitates fanless designs in various devices.

Regarding scalability, manufacturers can select from a broad spectrum of CPU cores and configurations to precisely match their specific performance and power requirements. Whether it's a high-performance Cortex-A core for a flagship smartphone or a power-sipping Cortex-M core for a low-power IoT sensor, ARM's flexibility allows for tailored solutions.

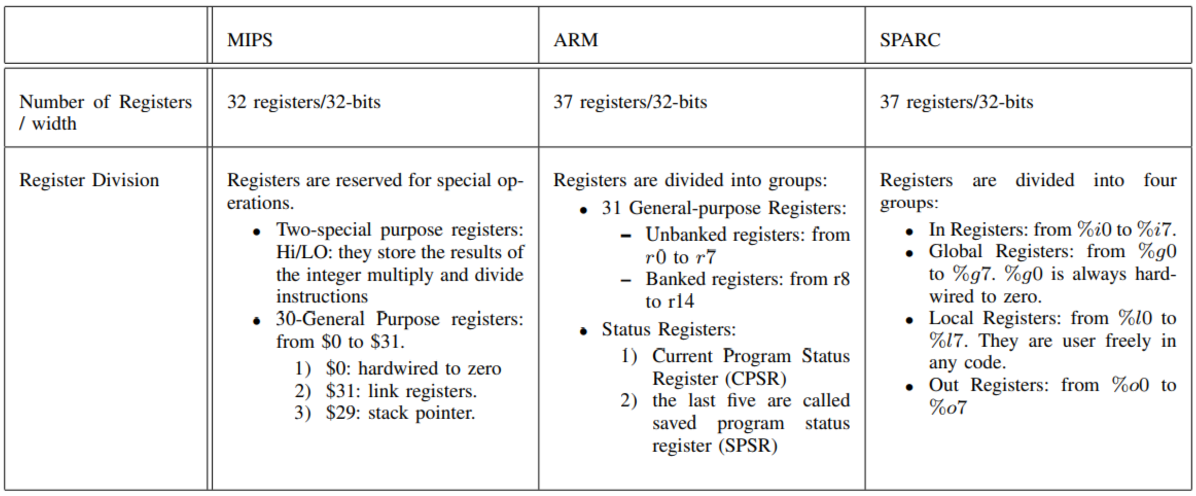
ARM processors find applications in an astonishingly diverse range of devices. Beyond the ubiquitous presence in smartphones and tablets, ARM powers automotive systems, including advanced driver assistance systems (ADAS) and in-car infotainment. For instance, Tesla's Autopilot system, known for its groundbreaking autonomous driving capabilities, utilizes ARM-based processors for real-time processing and control *(Tesla, 2017).* ARM architecture also drives the seamless operation of smart TVs, enabling crisp and responsive displays. In the realm of home automation systems, ARM's efficiency ensures reliable performance. Wearables like the Fitbit fitness tracker rely on ARM architecture for their energy-efficient operation, extending battery life and enhancing user convenience *(Fitbit, n.d.).*

In conclusion, the ARM processor family's success can be attributed to its energy efficiency, scalability, and versatility, making it a top choice for a broad spectrum of applications. Its influence extends from mobile devices to high-performance computing, solidifying its place as a leading processor architecture in the modern computing landscape. As technology continues to evolve, ARM processors are poised to play a pivotal role in shaping the future of computing devices and systems.

3.2 Comparison between MIPS32 and ARM

MIPS and ARM, two stalwarts in the realm of microprocessors, offer distinct architectures catering to a wide array of applications. This comparison delves into the architectural nuances, operating modes, addressing modes, and stack implementations of these processors.

MIPS32 is a 32-bit implementation of the MIPS architecture. In terms of architectural differences, MIPS32 and ARM, two prevalent processor architectures, diverge in their approaches. MIPS32, a RISC-based design, boasts a fixed-length instruction format, making decoding simple but demanding more instructions for complex operations. Conversely, ARM, another RISC architecture, employs a variable-length instruction set, enabling dense code but requiring more complex decoding. Additionally, MIPS32 often uses load-store architecture, necessitating separate instructions for memory access, while ARM's load-store architecture mandates separate instructions for data processing and memory operations. These differences influence their respective performance, power efficiency, and applicability in diverse computing devices.

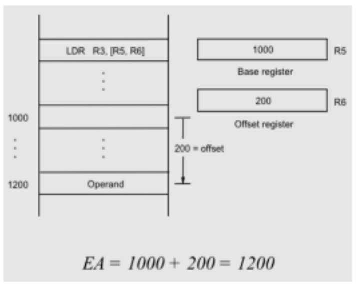


	MIPS	ARM	SPARC
Number of Registers / width	32 registers/32-bits	37 registers/32-bits	37 registers/32-bits
Register Division	Registers are reserved for special operations. • Two-special purpose registers: Hi/LO: they store the results of the integer multiply and divide instructions • 30-General Purpose registers: from \$0 to \$31. 1) \$0: hardwired to zero 2) \$31: link registers. 3) \$29: stack pointer.	Registers are divided into groups: • 31 General-purpose Registers: – Unbanked registers: from $r0$ to $r7$ – Banked registers: from r8 to r14 • Status Registers: 1) Current Program Status Register (CPSR) 2) the last five are called saved program status register (SPSR)	Registers are divided into four groups: • In Registers: from $\%i0$ to $\%i7$. • Global Registers: from $\%g0$ to $\%g7$. $\%g0$ is always hard-wired to zero. • Local Registers: from $\%l0$ to $\%l7$. They are user freely in any code. • Out Registers: from $\%o0$ to $\%o7$

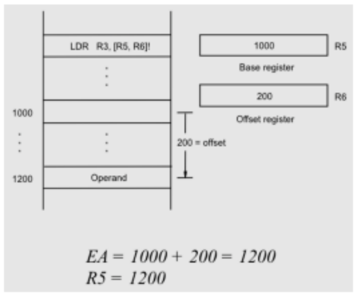
Table 1 Registers Convention Comparison

Table 1 illustrates the registers conventions in MIPS32 and ARM architectures *("MIPS, ARM, and SPARC. - An Architecture Comparison", 2014)*. MIPS32 features 32 registers, while ARM employs 37 registers. In MIPS, registers are divided into special-purpose and general-purpose categories, with specific roles like the stack pointer and link register. In ARM, the registers are organized into 31 general-purpose registers and several status registers, allowing for greater versatility and adaptability. The choice between these conventions depends on the requirements of the specific application at hand.

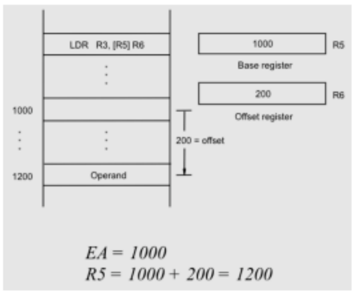
In terms of addressing modes between MIPS32 and ARM, MIPS32 processors support five addressing modes, offering flexibility for various computing tasks while ARM processors offer multiple addressing modes, providing flexibility and efficiency are shown in Figure 2 *("MIPS, ARM, and SPARC - An Architecture Comparison", 2014):*



(a) Pre-indexed addressing mode



(b) Pre-indexed addressing mode with write back



(c) Post-indexed addressing mode

Figure 2 ARM Addressing Modes

MIPS32 architecture employs five addressing modes to facilitate efficient and flexible instruction execution. Register addressing is predominantly utilized for calculating the effective addresses in jump register (jr) instructions. Immediate addressing enhances speed by allowing operations to be executed without memory access, with immediate values typically being 16-bit constants. PC-relative addressing is crucial for determining the timing of branch instructions, as it involves adding an offset to the Program Counter (PC). In the case of jump instructions, pseudo-direct addressing is employed, incorporating a 6-bit offset and a 26-bit target address, resulting in a 32-bit instruction. Additionally, base addressing is utilized in store and load word (sw & lw) instructions, functioning similarly to indirect addressing, with a register serving as a memory location pointer.

In contrast, ARM architecture offers a range of addressing modes to cater to diverse memory access requirements. Pre-indexed addressing involves using a register to offset the source/destination address, while pre-indexed addressing with write-back allows the new address to be saved in a register, denoted by an exclamation mark (!) in load instructions. Post-indexed addressing, akin to pre-indexed addressing with write-back, updates and stores the address only after the load or store operation. These modes are valuable for flexible memory access operations. Moreover, ARM supports program counter-relative addressing, enabling memory access relative to the Program Counter (r15), further enhancing the architecture's versatility in addressing memory locations.

In conclusion, the comparison between MIPS32 and ARM microprocessor architectures highlights their distinctive strengths and adaptability. MIPS32, with its energy-efficient RISC approach, excels in power conservation, making it well-suited for battery-powered and embedded systems. On the other hand, ARM's versatility, marked by its Thumb instruction set and flexible addressing modes, caters to a wide range of applications. The choice between these two architectures ultimately hinges on specific application requirements, with ARM's industry acceptance and performance efficiency often providing a competitive advantage. Additionally, the contrasting register conventions and addressing modes in MIPS32 and ARM offer developers a spectrum of options to tailor their choice to the precise needs of their projects. both MIPS32 and ARM have left a significant mark in the microprocessor landscape, each with its unique set of features catering to diverse computing demands.

Chapter 4 Conclusion

In conclusion, Part A of our study delved into the intricacies of MIPS assembly code by translating six mathematical equations, including AB+C, A+B-C, A-BC, (A and B) XOR C, (A and B) OR C, and (A NOR B) AND C, into efficient and functional assembly code. We provided a comprehensive overview, including flowcharts, pseudocode, and detailed code explanations to ensure a thorough understanding of the code's functionality and logic.

Moving on to Part B, we shifted our focus to the ARM architecture, a renowned and influential processor architecture alongside MIPS32. Our examination highlighted significant disparities between these two architectures, encompassing architectural nuances, register conventions, and addressing modes. MIPS32, with its RISC-based design, employs a fixed-length instruction format simplifying decoding but necessitating more instructions for complex operations. In contrast, ARM embraces a variable-length instruction set, enabling denser code while demanding more intricate decoding.

Register conventions also distinguish the two architectures, with MIPS32 offering 32 registers and ARM boasting 37, enhancing the latter's versatility and adaptability. Furthermore, MIPS32 predominantly employs a load-store architecture, requiring separate instructions for memory access, while ARM divides instructions for data processing and memory operations within its load-store architecture. This divergence is further exemplified in their addressing modes. MIPS32 supports five modes, including immediate and PC-relative addressing, while ARM offers a broader spectrum, encompassing pre-indexed and post-indexed addressing, resulting in greater flexibility and efficiency.

Conclusively, the choice between ARM and MIPS32 hinges on specific application requirements, with ARM's versatility and efficiency often conferring a competitive advantage. These architectural differences underscore the significant impact of both ARM and MIPS32 in shaping the dynamic microprocessor landscape.

References

1. ARM Holdings. (n.d.). ARM architecture reference manual. ARM Limited.
2. ARM Developer. (n.d.). Retrieved from https://developer.arm.com/
3. Fitbit. (n.d.). Retrieved from https://www.fitbit.com/
4. Smith, J. (2020). The evolution of ARM architecture. *Computer Science Review*, 15, 25-38.
5. Tesla. (2017). Autopilot. Tesla, Inc. Retrieved from https://www.tesla.com/autopilot
6. El Kady, S., Khater, M., & Alhafnawi, M. (2014). MIPS, ARM, and SPARC - An Architecture Comparison.
7. Southwell, S. (n.d.). Processor Design #1: Overview. [LinkedIn Profile]. Retrieved from www.linkedin.com
8. Sheldon, R. (2023, March 1). Million Instructions Per Second (MIPS). IT Operations. Retrieved from https://www.techtarget.com/searchitoperations/definition/MIPS-million-instructions-per-second
9. "Computer Programming | Introduction to Computer Programming" (2023, July 8). Learn Computer Science. Retrieved from https://www.learncomputerscienceonline.com/computer-programming/

APPENDIX A

General requirement on Report

- The report must be formatted with a font size of 12pt if Times New Roman or a font size of 11pt if Arial and 1.5 line spacing. Please ensure the paragraphs are properly aligned/ justified.
- There should be List of Tables and List of Figures after the Table of Content.
- The report should be in chapters and the structure should not go beyond the second level. Instead of adding subsections at the third level you may use bullets if required.
- All information provided must be straight to the point, precise and all information must be accordingly cited and well presented. Avoid plagiarism.
- All figures and tables must have a title and referenced i.e. indicate the source.
- There might be slight variations in the order and content required, please do approach your relevant lecturer for future assistance.
- Please also include the following in your report
  - ✓ Page numbering on each page (Page X of Y)
  - ✓ Figure and table caption font size: Times New Roman,10 pt
  - ✓ Position of figure and table: centre aligned

Appendix B

Criteria	Fail	Marginal Fail	Pass	Credit	Distinction
	0-10	**11-12**	**13-16**	**17-20**	**21-25**
Assembly Program (25%)	• **Outlined** inappropriate program • Did not submit any assembly code.	• **Outlined** appropriate program but not working as needed. • Completed up to 25% of the program.	• **Outlined** appropriate program and working with some minor errors. • Completed up to 25% to 50% of the program	• **Outlined** appropriate program and working well without errors. • Completed up to 50% to 75% of the program.	• **Outlined** appropriate and unique program and working well without errors. • Extra ordinary program
Program Description (25%)	• **Exemplified** inappropriate flow chart, pseudocode and very poor explanation on the program description	• **Exemplified** appropriate flowchart and pseudocode but with very poor explanation on the program description	• **Exemplified** appropriate flowchart and pseudocode yet surface explanation on the program description • Errors in the explanations	• **Exemplified** appropriate flowchart and pseudocode and detailed explanation on the program description • Minor errors in the explanations	• **Exemplified** appropriate flowchart and pseudocode and detailed and unique explanation on the program description • No errors
Description on the processor and Comparison between Processors (25%)	• **Exemplified** improper /no description on the processor • **Exemplified** improper comparison between the processors	• **Exemplified** very brief description on the processor. Content is very general and mostly irrelevant. • **Exemplified** very brief comparison between the processors	• **Exemplified** average description on the processor. Content is relevant • **Exemplified** average and relevant comparison between the processors	• **Exemplified** good description on the processor. Content is relevant and detailed • **Exemplified** detailed and relevant comparison between the processors	• **Exemplified** excellent description on the processor. Content is relevant and extensive information. • **Exemplified** extensive and detailed comparison between the processors
Group Presentation (25%)	• Simulator is not working and not relevant. • **Clarification** was not provided for did not turn up for the presentation or unable to answer any of the questions	• Simulator is working but not relevant. • **Clarification** was minimum for the work done and responsibly answered up to 25% of the questions	• Simulator is working with some major errors. • **Clarification** was sufficient for the work done in the assignment and responsibly answered up to 25% - 50% of the questions	• Simulator works well with minor error • **Clarification** was clear and in detail for the work done in the assignment and responsibly answered up to 50% - 75% of the questions.	• Simulator works well for all aspects of input/output condition • **Clarification** was highly appropriate, and coverage of the complete work done in the assignment and was excellent and able to answers all questions