UNIVERSITI TUNKU ABDUL RAHMAN

FACULTY OF INFORMATION & COMMUNICATION TECHNOLOGY

GROUP ASSIGNMENT SESSION 202305

UCCD1133 INTRODUCTION TO COMPUTER ORGANISATION AND ARCHITECTURE

Group 83

Group Members:

1. Chin Zi Wei (2105873)

2. Look Zheng Hong (2106397)

3. Tan Yi Xin (2101990)

4. Khor Ming Keat (2104771)

5. Damien Chuah Wen Xian (2106379)

Table of Contents

List of Tables and Figures

List of Figures

Introduction

An assembly program that consists of six equations that allow the user to have options to choose which equation to execute has been designed and implemented. In the assembly program, it allows the user to select the equation by entering the corresponding equation number (1-6). Then, the program will prompt the user for required input values, perform calculations and display the answer.

In this report, under part A, it consists of detail explanation on the complete workable assembly code that represent the equation given in the program. Besides that, it includes a flowchart that outline the structure of the assembly program and also a pseudocode that describe the program, and the instructions set, registers, system call and addressing modes involved. Whereas in part B, it consists of detail description on the overview of x86 processor and critical comparison between the x86 processor and MIPS32 processor.

Chapter 1: Part A - Assembly language with detailed description

1.1 Complete Assembly Code Represent Equation

```
# Equation (1) (A AND B) OR C
performEq1:

    jal getInput        # jump to getInput instructions

    # Calculate (A AND B) OR C
    and $t1, $s1, $s2
    or $t2, $t1, $s3

    jal printresult     # jump to printresult instructions

    j while
```

Figure 1.1

In Equation 1 ((A AND B) OR C), it will run the and instruction which mask the bits value of value in register $s1 (A) and $s2 (B) and store the result in $t1. Then, it will run the or instruction which combine bit fields of value in $t1 (result of A and B) with $s3 (C) and store the result in $t2.

```
# Equation (2) (A XOR B) AND C
performEq2:

    jal getInput        # jump to getInput instructions

    # Calculate (A XOR B) AND C
    xor $t1, $s1, $s2
    and $t2, $t1, $s3

    jal printresult     # jump to printresult instructions

    j while
```

Figure 1.2

In Equation 2 ((A XOR B) AND C), it will run the xor instruction which compare the bits value and produce bit value 1 if both bits value of value in register $s1 (A) and $s2 (B) different and vice versa and store the result in $t1. Then, it will run the and instruction which mask bit values of value in $t1 (result of A and B) and $s3 (C) and store the result in $t2.

```
# Equation (3) A AND (B NOR C)
performEq3:

    jal getInput        # jump to getInput instructions

    # Calculate A AND (B NOR C)
    nor $t1, $s2, $s3
    and $t2, $s1, $t1

    jal printresult     # jump to printresult instructions

    j while
```

Figure 1.3

In Equation 3 (A AND (B NOR C)), it will run the nor instruction which combine bit fields of value in register $s2 (B) with $s3 (C) then invert them and store the result in $t1. Then, it will run the and instruction which mask the bits value of value in register $s1 (A) and $t2 (result of B and C) and store the result in $t2.

```
# Equation (4) (A + B ) - C
performEq4:

    jal getInput        # jump to getInput instructions

    # Calculate (A + B) - C
    add $t1, $s1, $s2
    sub $t2, $t1, $s3

    jal printresult     # jump to printresult instructions

    j while
```

Figure 1.4

In Equation 4 ((A + B) - C), it will run the add instruction which add the value in register $s1 (A) and $s2 (B) and store the result in $t1. Then, it will run the subtract instruction which subtract the value in $t1 (result of A and B) with $s3 (C) and store the result in $t2.

```
# Equation (5) (A * B) + C
performEq5:

    jal getInput        # jump to getInput instructions

    # Calculate (A * B) + C
    mul $t1, $s1, $s2
    add $t2, $t1, $s3

    jal printresult     # jump to printresult instructions

    j while
```

Figure 1.5

In Equation 5 ((A * B) + C), it will run the multiply instruction which multiply the value in register $s1 (A) and $s2 (B) and store the result in $t1. Then, it will run the add instruction which add the value in $t1 (result of A and B) and $s3 (C) and store the result in $t2.

```
# Equation (6) (A - B) * C
performEq6:

    jal getInput        # jump to getInput instructions

    # Calculate (A - B) * C
    sub $t1, $s1, $s2
    mul $t2, $t1, $s3

    jal printresult     # jump to printresult instructions

    j while
```

Figure 1.6

In Equation 6 ((A - B) * C), it will run the subtract instruction which subtract the value in register $s1 (A) with $s2 (B) and store the result in $t1. Then, it will run the multiply instruction which multiply the value in $t1 (result of A and B) and $s3 (C) and store the result in $t2.

1.2 Flowchart

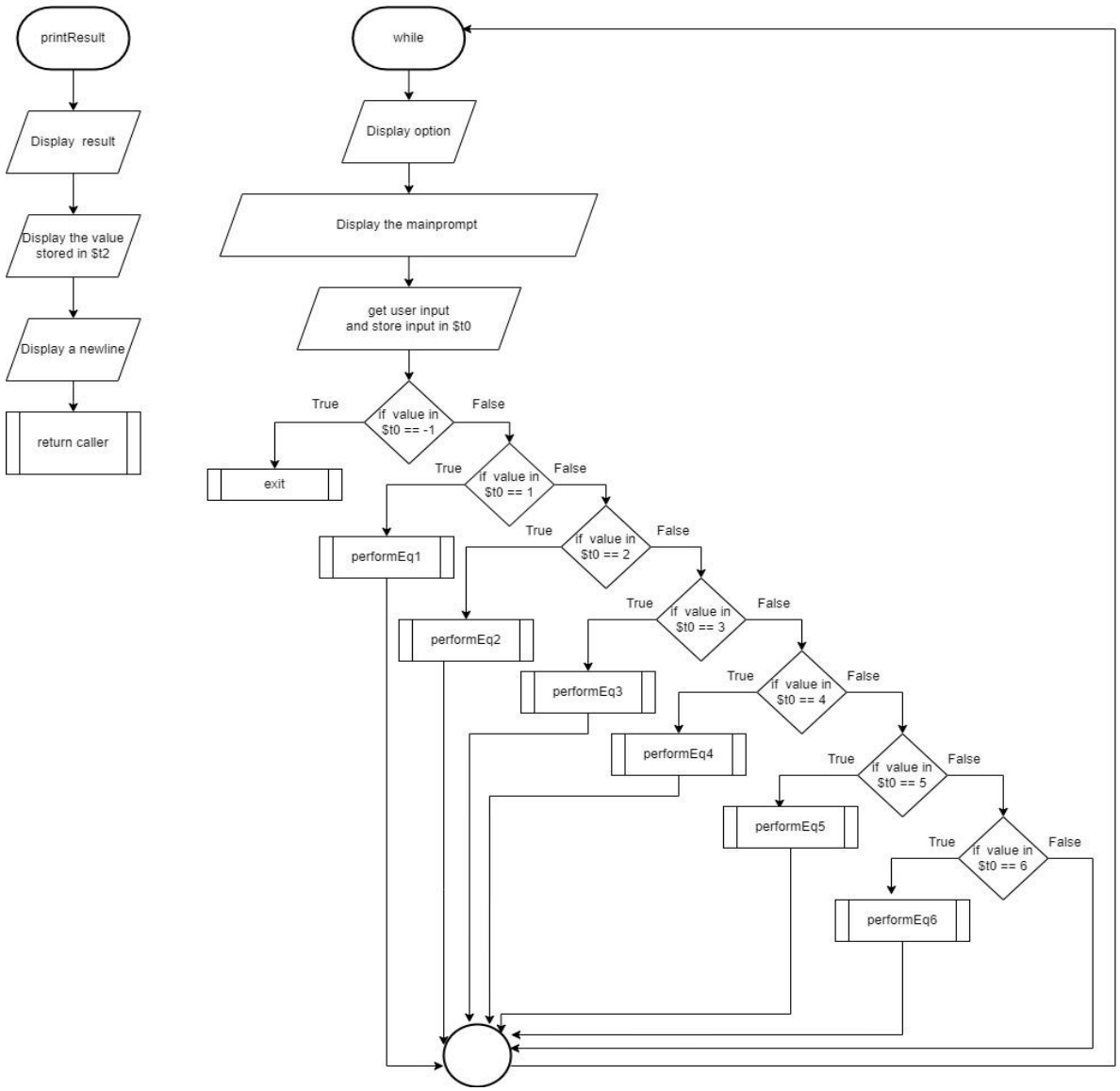

Figure 1.7

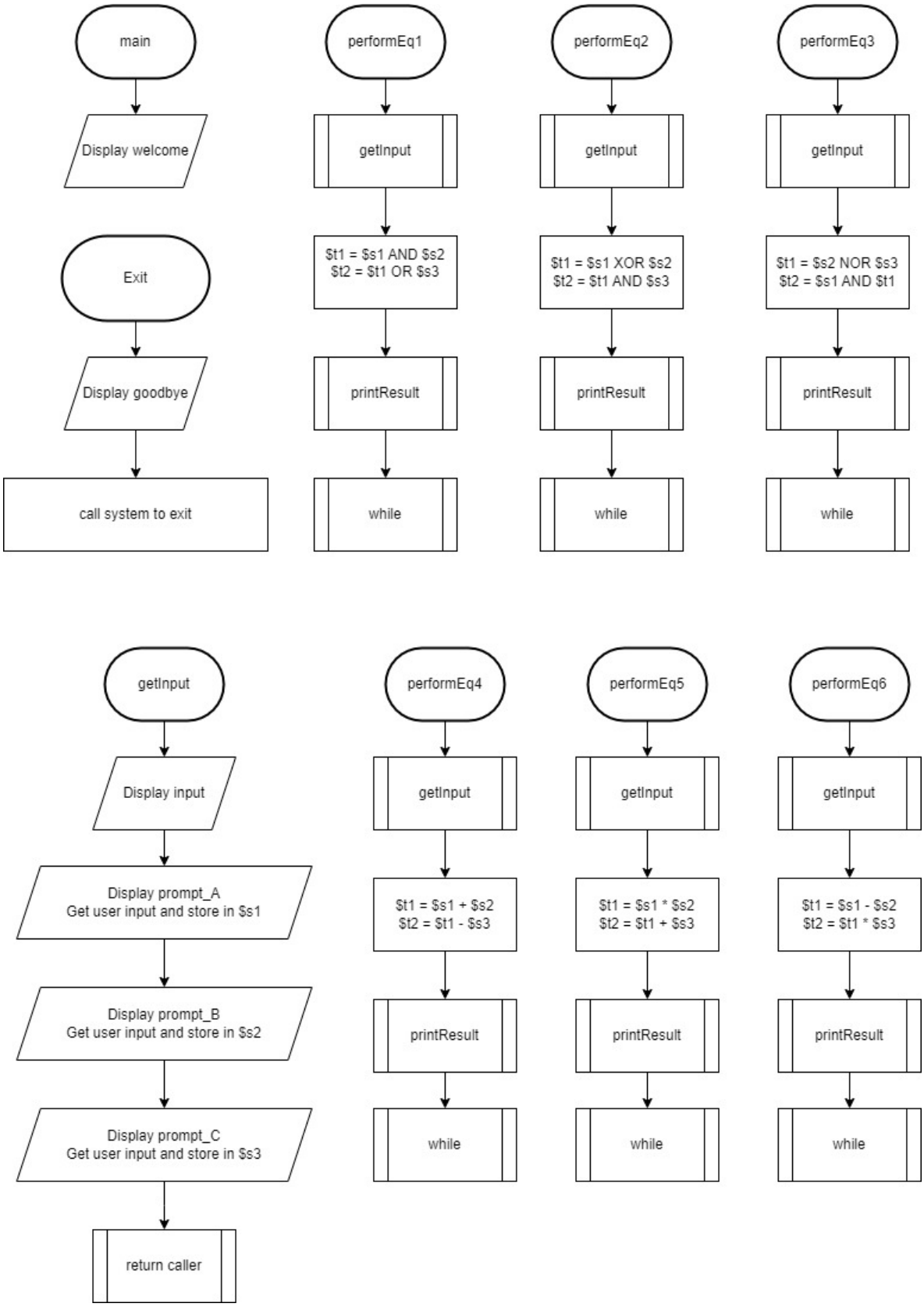
main
Display welcome
Exit
Display goodbye
call system to exit
performEq1
getInput
$t1 = $s1 AND $s2
$t2 = $t1 OR $s3
printResult
while
performEq2
getInput
$t1 = $s1 XOR $s2
$t2 = $t1 AND $s3
printResult
while
performEq3
getInput
$t1 = $s2 NOR $s3
$t2 = $s1 AND $t1
printResult
while
getInput
Display input
Display prompt_A
Get user input and store in $s1
Display prompt_B
Get user input and store in $s2
Display prompt_C
Get user input and store in $s3
return caller
performEq4
getInput
$t1 = $s1 + $s2
$t2 = $t1 - $s3
printResult
while
performEq5
getInput
$t1 = $s1 * $s2
$t2 = $t1 + $s3
printResult
while
performEq6
getInput
$t1 = $s1 - $s2
$t2 = $t1 * $s3
printResult
while

Figure 1.8

1.3 Pseudocode

getInput:

Print string of input 3 values message

Print string of input A message

Read integer input by user

Print string of input B message

Read integer input by user

Print string of input C message

Read integer input by user

Return to caller

printresult:

Print string of result message

Print integer (result of calculation) to screen

Print string of whitespace

Return to caller

Main:

Print string of welcome message

While:

Print string of options menu

Print string of input option message

Read integer input by user

If input == -1, then branch to exit

Else if input == 1, then branch to performEq1

Else if input == 2, then branch to performEq2

Else if input == 3, then branch to performEq3

Else if input == 4, then branch to performEq4

Else if input == 5, then branch to performEq5

Else if input == 6, then branch to performEq6

Else jump to while

performEq1:

Call getInput

Perform and operation on input A and B

Perform or operation on result of A and B, and input C

Call printresult

Jump to while

performEq2:

Call getInput

Perform exclusive or operation on input A and B

Perform and operation on result of A and B, and input C

Call printresult

Jump to while

performEq3:

Call getInput

Perform not or operation on input B and C

Perform and operation on result of B and C, and input A

Call printresult

Jump to while

performEq4:

Call getInput

Perform add operation on input A and B

Perform subtract operation on result of A and B, and input C

Call printresult

Jump to while

performEq5:

Call getInput

Perform multiple operation on input A and B

Perform add operation on result of A and B, and input C

Call printresult

Jump to while

performEq6:

Call getInput

Perform subtract operation on input A and B

Perform multiple operation on result of A and B, and input C

Call printresult

Jump to while

exit:

Print string of goodbye message

Exit the program

Description:

In the pseudocode, the print string function is performed by load the immediate 4 (system code for print string) and store it in $v0 with load the address of variable (storing string) and store it in $a0. Then, call the system console to perform the operation. Whereas, for the read integer function, it is performed by load the immediate 5 (system code for read integer) and store it in $v0 with call the system console to perform the operation. While for the print integer function, it is performed by load the immediate 1 (system code for print integer) and store it in $v0 with move the value in $t2 (storing result of calculation) into $a0. Then, call the system console to perform the operation. Besides that, for exit program function, it is performed by load the immediate 10 (system code for exit program) and store it in $v0 with call the system console to perform the operation.

On the other hand, conditional and unconditional branch instructions are used in the program which are jump (j), jump register (jr), jump and link (jal) and branch if equal (beq). In the pseudocode, jump to statement is equivalent to j instruction, return to caller statement is equivalent to jr instruction, call function statement is equivalent to jal instruction and if statement is equivalent to beq instruction. Both j and jal instructions are in J-Type instruction format which uses pseudo-direct addressing mode while jr instruction is in R-Type instruction format which uses register addressing mode. For beq instruction, it is in I-Type instruction format which uses PC-relative addressing mode.

Besides that, logical and arithmetic instructions are also used in the program which are and, or, xor, nor, add, subtract and multiply. In the pseudocode, those operations mentioned are aligned with their respective instruction such as and operation is aligned with and instruction. All of these 6 instructions mentioned are in R-Type instruction which uses register addressing mode.

Chapter 2: Part B- Overview of a processor

2.1 x86 processor

In computing, x86 is refers to the family of microprocessors developed by Intel Corporation, which is based on Intel's 8086 and 8088 processors. On June 8, 1978, Intel has introduced the 8086, which is a new 16-bits microprocessor. At the same time, the born of x86 architecture has created a new era. The x86 instruction set was specifically developed by the Intel for its first 16-bits CPU, which is referred to the i8086. The CPU- i8088 amplified in first PC in the world which is launched by IBM Corporation in 1981 also uses the x86 instructions. (Mia, 2023)

It is one of the most widely used Central Processing Unit (CPU) architecture for desktops, laptops, and servers. Despite the fact that other architectures exist and are gaining market share with mobile devices like smartphones and even Apple, beginning to include its M1 chip in the latest MacBook's and Mac Mini, this architecture persists to be the standard CPU architecture for modern computers systems with the exception of embedded and mobile devices. This architecture supports 64-bits, 32-bits, and 16-bits (Phillips, 2021).

Besides, an x86 server refers to a computer that uses an x86 central processing unit (CPU) architecture. Also, it may also use an x86-64 CPU, which is backward compatible with the x86 CPUs. An x86 architecture processor is a type of microprocessor which is suitable for the x86 instruction set. It is a collection of instructions that can be executed by x86-compatible microprocessors and will be used to manage the x86 microprocessor operations. An x86 microprocessors can run on almost every type of computer, including desktops, servers, laptops as well as supercomputers.

With the constant improvement of CPU technology, Intel has invented the upgraded versions of the i80386, i80486 until today's Pentium 4 series. However, all Intel CPUs still use the x86 instructions to guarantee that the computers are able to run various applications developed in the past to protect and preserve the software resources.

In comparison to additional servers, the x86 server design has several advantages, including ability to be used in various system and cost. To elaborate further, network infrastructures, cloud computing and data centres all use the server with x86 architecture. This

technology has been utilized for many years to perform various purposes. For example, providing web pages, hosting database, and running crucial applications.

The main advantage of using the x86 processors is that they are compatible with a variety of software in the market. This signifies that businesses can avoid worrying about compatibility issues when running legacy applications on x86 processors. In addition, x86 processors are able to carry out more tasks simultaneously than alternative processors, thus making them suit better with business with heavy workloads.

Furthermore, the x86 instruction set also provides outstanding compatibility with software and operating systems. As a consequence, servers running x86 processors became the ideal choice for mixed environment as they are guaranteed to run any operating system or software flawlessly. Moreover, the fact that x86 processors are energy efficient, helps business to save more on their energy bills by minimize the energy consumptions.

Last but not least, x86 processors are reliable and more secure. Thus, it is an excellent option for companies that required to protect their data (Mulford, 2022).

2.2 Differences between MIPS32 and x86 processors

1. RISC vs CISC

The main difference between a MIPS32 processor and a x86 processor is the type of microprocessor architecture adopted by the processors. MIPS32 processor belongs to the family of reduced instruction set computer (RISC) while x86 processor belongs to the family of complex instruction set computer (CISC).

RISC adopted by MIPS32 processor is a microprocessor designed to execute a limited set of computer instructions, allowing it to achieve greater speed, perform millions of instructions per second, or MIPS. If it is designed to execute a larger set of computer instructions, the microprocessor tends to be more complicated and operate in a slower speed as it requires additional transistor and circuitry. By having a smaller instruction set, RISC allows operating system and application programmers to develop code in a easier way. Besides, RISC also supports pipelining where a number of instructions can be executed concurrently, and thus the instruction throughput and system performance can be increased (Kirvan, 2023).

On the other hand, CISC adopted by x86 processor is designed to execute a large set of computer instructions. These instructions can vary in many ways, such as length, from simple tasks to multi-step operations, where longer tasks taking more clock cycle. In CISC, a single instruction can carry out multiple operations. For example, an instruction may tell the processor the load values from two different memory banks, multiply the values retrieved, and store the result. Integrating complex instructions directly into processor's instruction set require the software compiler to generate fewer lines of assembly code, and thus reduces the amount of memory required to store these instructions (Christensson, 2022).

RISC and CISC are two different approaches to instruction set design. Both RISC and CISC has its own advantages and disadvantages when compared to each other. CISC contain more instructions that can tackle more complex tasks over several clock cycles, while RISC includes contain fewer instructions that each take a single clock cycle. When multiple RISC instructions are chained together to complete the same task as a single CISC instruction can do, the number of clock cycles required is often the same. Moreover, RISC require less transistors to store the instruction sets compared to CISC as CISC have a larger set of instructions. RISC processor can use the transistors for memory storage or use lesser transistors for a less-complex chip design, which results in a smaller, produce less heat, and require less power chip (Christensson, 2022).

2. Applications within an IoT Gateway

MIPS processors is widely used in IoT gateway embedded systems due to its high performance, lower power consumption, cost effective, and its scalability. It plays a crucial role in ensuring low latency and high-speed data processing for edge computing in IoT gateways, which is important when apply in smart cities, transportation systems, and surveillance. Besides, MIPS offers high performance and energy efficient computing in wearable medical devices, including fitness trackers, smartwatches, and remote patient monitoring devices (Mia, 2023).

X86 processors is a popular option when choosing a processor for IoT gateway because of its high performance and wide availability. It is also popular for desktop and server applications. X86-based IoT gateways can be used to connect and manage smart city devices vary from traffic lights, sensors, and security cameras. X86 architecture is ideal for handling large amounts of data generated by these devices because of its high processing power. In terms

or healthcare industries, the high processing power of x86 based IoT gateways can handle complex medical data and algorithms, and thus it can be used for enabling remote patient monitoring, wearable devices, and medical imaging, so that the quality of patient care can be improved. Moreover, x86_based IoT gateways are widely used in education for interactive whiteboards ad classroom management systems. Learning experience for students and teachers can be further enhanced when x86_based IoT gateways are integrated with educational software and tools (Mia, 2023).

In short, MIPS is an ideal architecture for low power consumption IoT devices such as smart home appliances because of its efficiency and low power consumption. While on the other hand, x86 architecture is ideal for IoT gateways that require high performance power, such as edge computing and AI applications as a result of its powerful processing power and ability to handle complex tasks and applications (Mia, 2023).

Conclusion

In the assembly program, it includes 6 unique equations that execute different operation type to perform calculation based on the input values from the user. There are 3 equations that are performing logical operations which are Equation 1 ((A AND B) OR C), Equation 2 ((A XOR B) AND C) and Equation 3 (A AND (B NOR C)). Besides that, there are also 3 equations that performing arithmetic operations which are Equation 4 ((A + B) – C), Equation 5 ((A * B) + C) and Equation 6 ((A - B) * C). Thus, the user is allowed to choose the equation by entering the corresponding equation number (1-6) and input the required values then get the answer from the calculation.

On the other hand, an overview with detailed description on the x86 processor based on its background, application, component and evolution are included. In addition, the advantages of x86 processor compared to others MIPS processor are also considered in the overview. Besides that, a description of critical comparison between the x86 processor and MIPS32 processor is included. In the description, it mainly compares the 2 processors based on their type of microprocessor architecture adopted (RISC & CISC) and their applications within an IoT Gateway.

References

Christensson, P. (2022). CISC. *techterms.com*. https://techterms.com/definition/cisc

Kirvan, P. (2023). RISC (reduced instruction set computer). *WhatIs.com*. https://www.techtarget.com/whatis/definition/RISC-reduced-instruction-set-computer

Mia. (2023). MIPS, ARM, X86, NPU, what's the best hardware platform for your IoT gateway? *DusunIoT*. https://www.dusuniot.com/blog/mips-vs-arm-vs-x86-vs-npu-the-best-hardware-platform-for-iot-gateways/

Mulford, J. (2022, August 3). *What is x86 Architecture?* Mulcas. https://mulcas.com/what-is-x86-architecture/

Phillips, T. (2021, April 20). *A Hacker's Tour of the X86 CPU Architecture.* Secure Ideas. https://www.secureideas.com/blog/2021/04/a-hackers-tour-of-the-x86-cpu-architecture.html

Appendix

```
.data
welcome:    .asciiz "Welcome to our Program."

# Show menu and prompt input
options:    .asciiz "\n1: (A AND B) OR C\n2: (A XOR B) AND C\n3: A AND (B NOR C)\n4: (A + B) - C\n5: (A * B) + C\n6: (A - B) * C\n-1: quit\n"  #show menu
mainprompt: .asciiz "Please enter the equation: \n\n"
input:      .asciiz "Enter three values (integer) :\n"

# Message to prompt the user to input each value
prompt_A:   .asciiz "Enter value A: "
prompt_B:   .asciiz "Enter value B: "
prompt_C:   .asciiz "Enter value C: "

# Message to show result and for exit
result:     .asciiz "The result is: "
newline:    .asciiz "\n"
goodbye:    .asciiz "Thanks for using our program. Goodbye!"



.text
.globl main

# Function to get input
getInput:

   # Print message for input
   li $v0, 4
```

```
la $a0, input
syscall

# Display a message before prompting for user input A
li $v0, 4
la $a0, prompt_A
syscall

# Prompt for user input A and store in $v0
li $v0, 5
syscall

move $s1, $v0        # A

# Display a message before prompting for user input B
li $v0, 4
la $a0, prompt_B
syscall

# Prompt for user input B and store in $v0
li $v0, 5
syscall

move $s2, $v0        # B

# Display a message before prompting for user input C
li $v0, 4
la $a0, prompt_C
syscall
```

```
    # Prompt for user input C and store in $v0
    li $v0, 5
    syscall

    move $s3, $v0        # C

    jr $ra

# Print result message
printresult:

    # Print the result
    li $v0, 4
    la $a0, result
    syscall

    # Print the value
    li $v0, 1
    move $a0, $t2
    syscall

    li $v0, 4
    la $a0, newline
    syscall

    jr $ra

# Print welcome message
main:
```

```
    li $v0, 4           # 'li': load immediate, 'v0': system call code before making a system
call
    la $a0, welcome
    syscall             # perform the operation

# Print option and prompt
while:

    li $v0, 4           # load the system call code '4' into register v0
    la $a0, options     # load address of the options prompted into a0.
    syscall             # system call instruction is used to perform the operation specified in
v0.

    # Prompt for user input
    li $v0, 4
    la $a0, mainprompt
    syscall

    # Get user input (equation choice)
    li $v0, 5           # system call code=5(read integer input)
    syscall             # triggers a system cal based on the value in v0,=5
    move $t0, $v0

    # Branch by input options
    # Check for exit (-1). (if option == 1, exit;)
    beq $t0, -1, exit

    # Else if
    beq $t0, 1, performEq1
    beq $t0, 2, performEq2
    beq $t0, 3, performEq3
```

```
    beq $t0, 4, performEq4
    beq $t0, 5, performEq5
    beq $t0, 6, performEq6

    # Else (invalid input, loop again)
    j while

    # End of main

# Equation (1) (A AND B) OR C
performEq1:

    jal getInput          # jump to getInput instructions

    # Calculate (A AND B) OR C
    and $t1, $s1, $s2
    or $t2, $t1, $s3

    jal printresult       # jump to printresult instructions

    j while

# Equation (2) (A XOR B) AND C
performEq2:

    jal getInput          # jump to getInput instructions

    # Calculate (A XOR B) AND C
    xor $t1, $s1, $s2
    and $t2, $t1, $s3
```

```
    jal printresult         # jump to printresult instructions

    j while

# Equation (3) A AND (B NOR C)
performEq3:

    jal getInput            # jump to getInput instructions

    # Calculate A AND (B NOR C)
    nor $t1, $s2, $s3
    and $t2, $s1, $t1

    jal printresult         # jump to printresult instructions

    j while

# Equation (4) (A + B ) - C
performEq4:

    jal getInput            # jump to getInput instructions

    # Calculate (A + B) - C
    add $t1, $s1, $s2
    sub $t2, $t1, $s3

    jal printresult         # jump to printresult instructions

    j while
```

```
# Equation (5) (A * B) + C
performEq5:

    jal getInput          # jump to getInput instructions

    # Calculate (A * B) + C
    mul $t1, $s1, $s2
    add $t2, $t1, $s3

    jal printresult       # jump to printresult instructions

    j while

# Equation (6) (A - B) * C
performEq6:

    jal getInput          # jump to getInput instructions

    # Calculate (A - B) * C
    sub $t1, $s1, $s2
    mul $t2, $t1, $s3

    jal printresult       # jump to printresult instructions

    j while

# Exit and print goodbye message
exit:
```

```
# Print goodbye message and exit
li $v0, 4
la $a0, goodbye
syscall

# Exit program
li $v0, 10
syscall
```