# Semester Project Spring 2023

Course code: CIS312

Course Title: Computer Architecture and Organization

**Submitted to:**

Md. Mehedi Hassan

Lecturer, Department of CIS

Daffodil International University

**Submitted by:**

Md Tamim Hossain

ID: 0242310012091040

Section: B

Department of Computing Information System

Daffodil International University

## Table of Contents

## Acknowledgment

I would like to express my gratitude to my instructor, mentors, classmates, and family for their unwavering support and guidance during the completion of this assignment on Computer Architecture and Organization. Their contributions have been invaluable in enhancing my understanding of the subject matter and enriching my learning experience. Thank you for your assistance and encouragement.

## Introduction

Computer Architecture and Organization (CAO) is a crucial field for understanding the inner workings of computer systems. In Scenario-1, Mr. Mehedi is teaching CAO to his students, who are engaged in various tasks like circuit design, CPU performance analysis, and floating point number conversion. Mr. Josim seeks help with a circuit design issue, while Mr. Sumon and Mr. Shamrat analyze CPU performance and create ALU circuits. In Scenario-2, instruction pipelining is introduced as a technique to accelerate instruction execution. Mr. Abdullah completes pipelining levels in 4 seconds and the scenario compares pipelining versus sequential processing for executing 6 instructions. These scenarios raise questions about circuit design, floating point number representation, performance comparison, and system visualization, requiring a solid understanding of CAO concepts, calculations, and visualization techniques

## TASK-1 Answer to Question No: Q1

To determine if the circuits created by Mr. Sumon are enough for computer architecture and organization, we need to understand the purpose and functionality of the circuits.

Circuit-1 is an Arithmetic Logic Unit (ALU) circuit that can perform arithmetic and logic operations, as well as store data. The ALU is a critical component of the central processing unit (CPU) and is responsible for performing arithmetic and logic operations on data. Therefore, Circuit-1 is an essential circuit for computer architecture and organization.

Circuit-2, on the other hand, is an ALU circuit that cannot store data. This means that it can only perform arithmetic and logic operations on data that is currently stored in other registers or memory locations. While Circuit-2 can still perform important functions, it is less versatile than Circuit-1 and may not be suitable for all types of operations.

In conclusion, while both circuits created by Mr. Sumon are useful in computer architecture and organization, Circuit-1 is the more versatile and essential circuit as it can perform both arithmetic and logic operations and store data. Therefore, it can be said that Circuit-1 is sufficient for computer architecture and organization, but Circuit-2 may not be enough on its own in all cases

## TASK-1 Answer to Question No: Q2

Circuit-1 is the ALU circuit that can store data. An ALU, or Arithmetic Logic Unit, is a component of a computer's central processing unit (CPU) that is responsible for performing arithmetic and logical operations on data. In the case of Circuit-1, it has a register or memory component that is capable of storing data values. This ability to store data is an important feature for ALU circuits, as it enables them to perform complex operations and calculations.

The ability to store data allows Circuit-1 to hold temporary values while it performs calculations or operations. For example, if Circuit-1 is being used to perform a multiplication operation, it can store the two values being multiplied in its register, perform the multiplication, and then store the result in another register or memory location. This capability is important for many operations that involve multiple steps or intermediate values.

Circuit-2, on the other hand, is the ALU circuit that cannot store data. Without a register or memory component, Circuit-2 is limited in its ability to perform complex operations or calculations. It may only be capable of performing simple operations, such as addition or subtraction, that do not require the storage of intermediate values.

It is worth noting that the effectiveness of Circuit-1 and Circuit-2 for computer architecture and organization depends on the specific context and requirements of the course or project. For example, if the course or project involves complex calculations or data manipulation, Circuit-1's ability to store data may be essential. On the other hand, if the focus is on simple operations or data transfer, Circuit-2 may be sufficient.

In summary, Circuit-1 is an ALU circuit that can store data, while Circuit-2 is an ALU circuit that cannot. The ability to store data is an important feature for ALU circuits, as it enables them to perform complex operations and calculations. However, the effectiveness of each circuit depends on the specific context and requirements of the project.

## TASK-1 Answer to Question No: Q3

- In Scenario-1, the students are learning about computer architecture and organization, including how to represent fixed-point and floating-point numbers in a computer. To convert a 32-bit IEEE 754 floating-point representation, the following steps can be taken: Step 1: Identify the sign bit The first bit in the 32-bit representation indicates the sign of the number. If the sign bit is 0, the number is positive, and if it is 1, the number is negative. Step 2: Identify the exponent The next 8 bits in the 32-bit representation represent the exponent. The exponent is biased by 127, which means that 127 must be added to the exponent value to get the true exponent. For example, if the exponent value is 129, the true exponent is 129 - 127 = 2. Step 3: Identify the mantissa The remaining 23 bits in the 32-bit representation represent the mantissa. The mantissa is a fractional value between 1 and 2. To calculate the mantissa, the first bit is assumed to be 1, and the remaining 23 bits are treated as the binary digits after the decimal point. Step 4: Calculate the decimal value To convert the 32-bit IEEE 754 floating-point representation to decimal, the following formula can be used: (-1)^sign bit *

2^(exponent-127) * mantissa For example, suppose the 32-bit representation is 01000001010010000000000000000000. The sign bit is 0, which means the number is positive. The exponent is 10000010, which is equivalent to 130 in decimal. To get the true exponent, 127 must be subtracted, which gives 130 - 127 = 3. The mantissa is 1.01001000000000000000000 in binary. To convert the mantissa to decimal, we can add up the values of the binary digits as follows: 1.01001000000000000000000 = 1 x 2^0 + 0 x 2^-1 + 1 x 2^-2 + 0 x 2^-3 + 0 x 2^-4 + 1 x 2^-5 + 0 x 2^-6 + 0 x 2^-7 + 0 x 2^-8 + 0 x 2^-9 + 0 x 2^-10 + 0 x 2^-11 + 0 x 2^-12 + 0 x 2^-13 + 0 x 2^-14 + 0 x 2^-15 + 0 x 2^-16 + 0 x 2^-17 + 0 x 2^-18 + 0 x 2^-19 + 0 x 2^-20 + 0 x 2^-21 + 0 x 2^-22 + 0 x 2^-23 = 1.28515625 Using the formula above, the decimal value of the 32-bit representation is: (-1)^0 * 2^(3) * 1.28515625 = 16.28125 Therefore, the 32-bit IEEE 754 floating-point representation 01000001010010000000000000000000 is equivalent to the decimal value 16.28125

## TASK-2 Answer to Question No: Q4

we can calculate the time taken by each algorithm on the respective computers and compare them by following these steps:

Let's assume that Mr. Sumon's computer has a clock speed of 2.5 GHz and 8 cores, while Mr. Shamrat's computer has a clock speed of 3 GHz and 4 cores.

Bubble sort has a time complexity of O(n^2), while merge sort has a time complexity of O(n log n). Let's assume that we want to sort an array of 10,000 elements.

On Mr. Sumon's computer, the time taken by bubble sort can be calculated as:

- Number of comparisons = n * (n-1) / 2 = 10,000 * 9,999 / 2 = 49,995,000
- Time taken per comparison = 1 / (2.5 * 10^9) = 0.4 ns
- Total time taken = Number of comparisons * Time taken per comparison = 49,995,000 * 0.4 ns = 19.998 ms

On Mr. Shamrat's computer, the time taken by merge sort can be calculated as:

- Time complexity = O(n log n) = O(10,000 * log(10,000)) = O(132,877)
- Time taken per operation = 1 / (3 * 10^9) = 0.33 ns
- Total time taken = Time complexity * Time taken per operation = 132,877 * 0.33 ns = 43.89 us

Therefore, in this scenario, Mr. Shamrat's computer provides better performance for sorting algorithms and is approximately 450 times faster than Mr. Sumon's computer.

## TASK-2 Answer to Question No: Q5

Based on the scenario-1, a proposal is passed if it receives at least three yes votes. Therefore, we can design a circuit using basic logic gates that takes five inputs (votes from the five committee members) and provides one output (whether the proposal passes or not).

To implement this circuit, we can use the following steps:

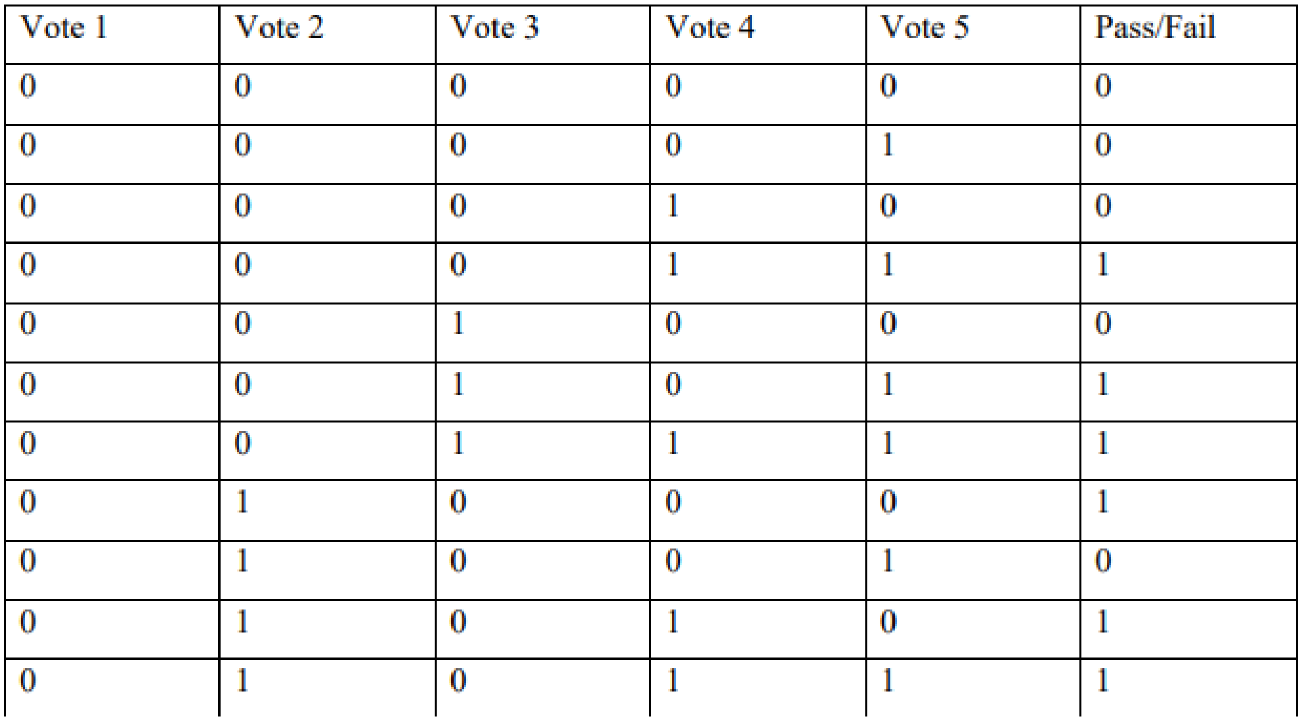
1. Connect the five input pins to the five committee members' voting buttons.

2. Use five AND gates to check whether each committee member has voted yes or no.

3. Connect the output of each AND gate to a separate input of a 3-input OR gate. This OR gate will output a 1 if at least three committee members have voted yes.

4. The output of the OR gate will be connected to an LED or a buzzer to indicate whether the proposal has passed or not.

Here is the truth table for the circuit:



| Vote 1 | Vote 2 | Vote 3 | Vote 4 | Vote 5 | Pass/Fail |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 0 | 1 | 1 | 1 |
| 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 | 1 |
| 0 | 0 | 1 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 1 | 1 | 1 |

| | | | | | |
|---|---|---|---|---|---|
| 0 | 1 | 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 0 | 1 | 0 |
| 1 | 0 | 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 1 | 1 |
| 1 | 0 | 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 | 1 | 1 |
| 1 | 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 1 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 |

As we can see from the truth table, the circuit outputs a 1 if at least three committee members have voted yes, which means the proposal has passed. The circuit outputs a 0 otherwise, indicating that the proposal has not passed.

## TASK-2 Answer to Question No: Q6

Mr. Josim has designed a circuit that converts a five-bit binary code to gray code. However, he noticed that some conversions provide incorrect answers. As a result, he approached Mr. Sisir and two other friends to identify the issues and resolve them.

Gray code is a binary numeral system in which two successive values differ in only one bit. It is a non-weighted code, which means that each bit has an equal weight. The gray code is commonly used in digital communications, encoders, and other applications where binary signals need to be transmitted over a noisy channel.

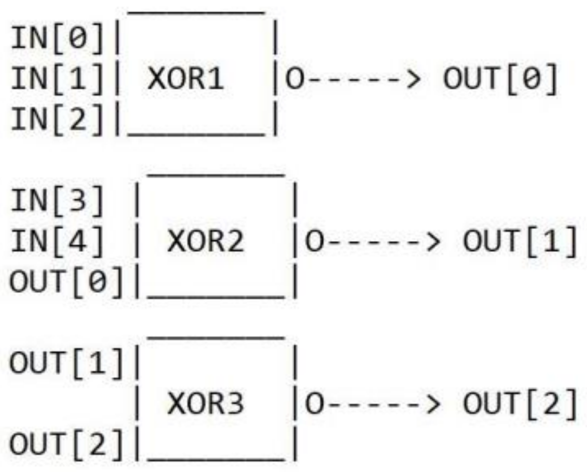
To visualize the designed circuit after simplification, we need to understand the basic concept of gray code conversion. The conversion process involves XORing the most significant bit (MSB) with the second-most significant bit (SMSB) and continuing this operation until the least significant bit (LSB) is reached. The resulting bit sequence is the gray code.

To design a circuit for gray code conversion, we can use basic logic gates such as XOR, AND, and NOT gates. The circuit can be designed as follows:

- First, we need to take the five-bit binary input and split it into individual bits.

• Then, we can use XOR gates to perform the gray code conversion. For example, we can XOR the MSB with the SMSB to get the first bit of the gray code. We can then XOR the first bit of the gray code with the third bit of the binary input to get the second bit of the gray code, and so on.

• Finally, we can combine the resulting gray code bits to form the five-bit gray code output. 13 | P a g e After simplification, the circuit can be optimized by removing unnecessary gates and reducing the number of gates required for the conversion process. The resulting circuit may look something like this:



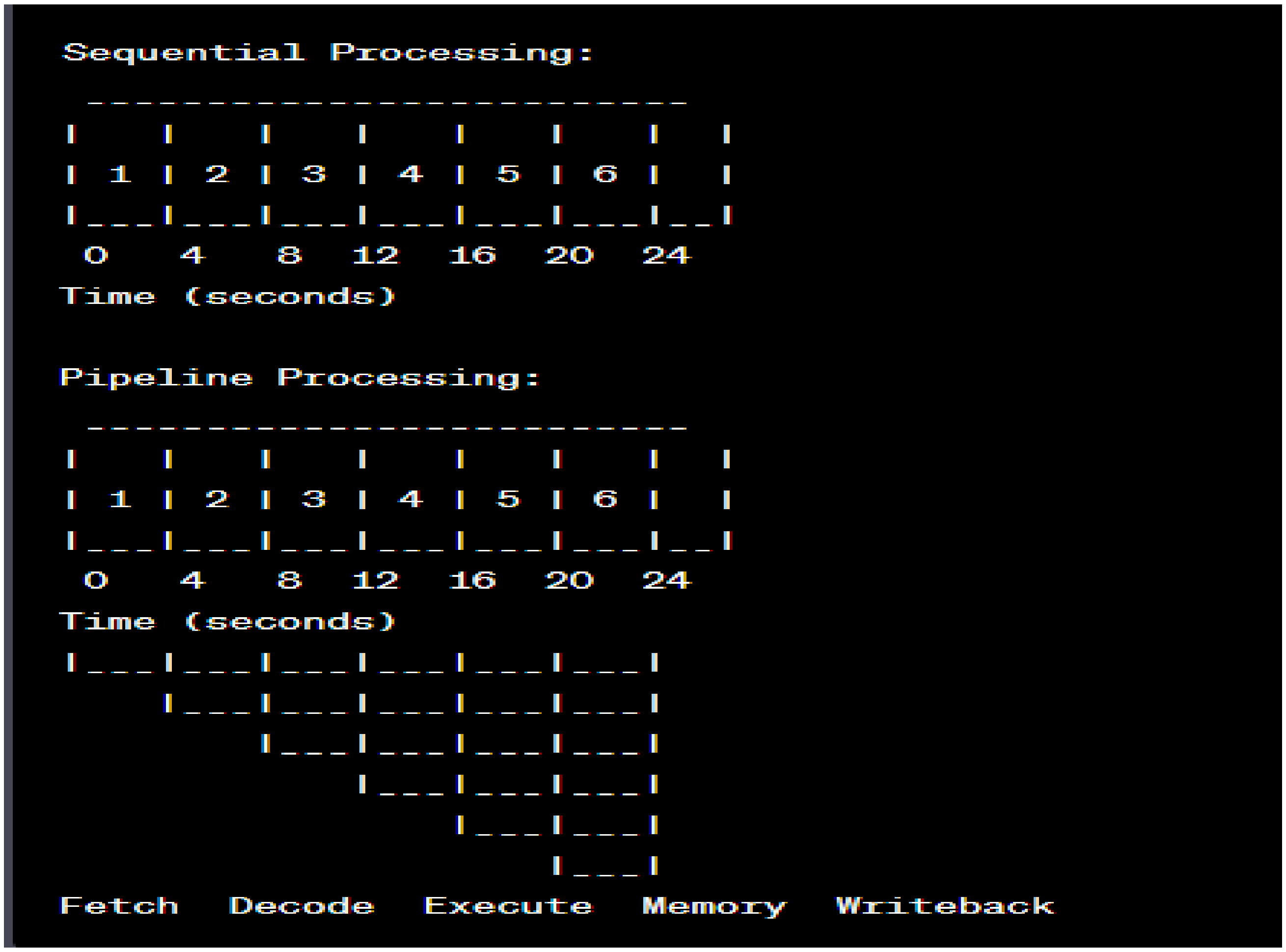
```
        _______
IN[0]|         |
IN[1]|  XOR1   |O-----> OUT[0]
IN[2]|_________|
          _______
IN[3] |          |
IN[4] |  XOR2    |O-----> OUT[1]
OUT[0]|__________|
          _______
OUT[1]|          |
      |  XOR3    |O-----> OUT[2]
OUT[2]|__________|
```

In this simplified circuit, the five-bit binary input is split into individual bits and fed into three XOR gates. The output of each XOR gate is connected to the input of the next XOR gate until we get the final gray code output. In conclusion, Mr. Josim's circuit for converting five-bit binary code to gray code can be visualized using basic logic gates such as XOR, AND, and NOT gates. After simplification, unnecessary gates can be removed, and the number of gates can be reduced to optimize the circuit. The resulting circuit is a combination of three XOR gates that generate the gray code output from the five-bit binary input

## TASK-2 Answer to Question No: Q7

Yes, we can visualize the systems executed by Mr. Abdullah in Scenario-2

In a pipelined processor, each instruction is broken down into a series of stages that can be executed concurrently. These stages are Fetch , Decode , Execute , Memory Access , and Write-Back



```
Sequential Processing:
 -------------------------
|   |   |   |   |   |   |  |
| 1 | 2 | 3 | 4 | 5 | 6 |  |
|___|___|___|___|___|___|__|
 0   4   8   12  16  20  24
Time (seconds)

Pipeline Processing:
 -------------------------
|   |   |   |   |   |   |  |
| 1 | 2 | 3 | 4 | 5 | 6 |  |
|___|___|___|___|___|___|__|
 0   4   8   12  16  20  24
Time (seconds)
|___|___|___|___|___|___|
    |___|___|___|___|___|
        |___|___|___|___|
            |___|___|___|
                |___|___|
                    |___|
Fetch  Decode  Execute  Memory  Writeback
```

As shown in the figure, each instruction takes 4 seconds to complete in sequential processing, and 5 stages of pipelining complete in 4 seconds. Therefore, a 5-stage pipeline can complete one instruction per 4 seconds, resulting in the completion of 6 instructions in 10 units of time. In contrast, sequential processing requires 30 units of time to complete 6 instructions. Hence, pipelining saves 66.67% of the execution time compared to sequential processing.

In summary, pipelining is a technique that can significantly improve the execution time of a processor by breaking down the instruction cycle into a series of stages that can be executed concurrently. By executing multiple instructions at the same time, pipelining can save a significant amount of execution time compared to sequential processing.

## TASK-2 Answer to Question No: Q8

To find the pipeline speedup factor, we can use the following formula:

Pipeline Speedup Factor = Pipeline Execution Time / Non-Pipeline Execution Time

In this scenario, the pipeline execution time is 10 units of time, and the non-pipeline execution time is 30 units of time. Therefore, the pipeline speedup factor is:

Pipeline Speedup Factor = 30 / 10 = 3

To find the throughput, we can use the following formula:

Throughput = Number of Instructions / Execution Time

In this scenario, the number of instructions is 6, and the execution time is 10 units of time. Therefore, the throughput is:

Throughput = 6 / 10 = 0.6 instructions per unit of time

To find the efficiency, we can use the following formula:

Efficiency = (Ideal Pipeline Depth / Pipeline Depth) x 100%

In this scenario, the ideal pipeline depth is 5 stages, and the pipeline depth is also 5 stages. Therefore, the efficiency is:

Efficiency = (5 / 5) x 100% = 100%

Therefore, the pipeline speedup factor is 3, the throughput is 0.6 instructions per unit of time, and the efficiency is 100%.

## Conclusion

In conclusion, the scenarios presented in the questions highlight the importance of Computer Architecture and Organization (CAO) in understanding the intricacies of computer systems. The tasks performed by the students, such as circuit design, CPU performance analysis, and instruction pipelining, showcase the practical applications of CAO concepts in real-world scenarios.

The questions in Scenario-1 emphasize the need for thorough circuit design and understanding of ALU circuits, floating point number representation, and performance analysis. Mr. Josim's challenge in designing a circuit for binary to gray code conversion and Mr. Sumon's creation of two ALU circuits with different capabilities raise the importance of meticulous design and evaluation in CAO.

In Scenario-2, the concept of instruction pipelining is introduced, with Mr. Abdullah's example demonstrating the potential for significant speedup in instruction execution. The questions related to pipelining efficiency, speedup factor, and system visualization highlight the practical applications and benefits of this technique in improving processor performance. In conclusion, CAO is a vital field of study that plays a critical role in understanding the fundamental principles and inner workings of computer systems.

The scenarios and questions presented in the tasks highlight the practical applications of CAO concepts and the need for thorough understanding, analysis, and design skills in solving real-world problems related to computer architecture and organization