TECHNICAL REPORT WRITING

INSTRUCTIONS AND INSTRUCTION SEQUENCING, ADDRESSING MODE

NAME: RUPAK SARKAR

MAKAUT ROLL NO.: 14271024036

STREAM: MASTER OF COMPUTER APPLICATION

SUBJECT NAME: COMPUTER ORGANIZATION AND ARCHITECTURE

SUBJECT CODE: MCAN-103

COLLEGE NAME: MEGHNAD SAHA INSTITUTE OF TECHNOLOGY

UNIVERSITY NAME: MAULANA ABUL KALAM AZAD UNIVERSITY OF TECHNOLOGY

SUMMARY

In computer organization and architecture, instructions are binary-encoded operations executed by the CPU, consisting of an opcode (operation code) and operands (data or addresses). Instruction sequencing determines the order of execution, typically sequential but alterable through control instructions like jumps and branches, with modern processors often using pipelining for efficiency. Addressing modes specify how operands are accessed, offering flexibility in instruction design. Common modes include immediate addressing (operand directly in the instruction), direct addressing (explicit memory address), indirect addressing (address points to the operand's location), and register addressing (operand stored in a register). These concepts form the foundation of efficient data handling and execution in computing systems.

This report will cover how instructions, instruction sequence and addressing mode operates in computer architecture.

CONTENTS

INTRODUCTION

Instructions

Instructions are binary-encoded commands that specify operations for the CPU to execute. They form the fundamental units of work in a computer system and are part of the instruction set architecture (ISA) of the processor. Each instruction typically includes two main components:

Opcode (Operation Code): Specifies the type of operation to be performed, such as addition, subtraction, data transfer, or comparison.

Operands: Provide the data or memory addresses on which the operation will act. Operands can be immediate values, registers, or memory locations, depending on the addressing mode.

Instructions are categorized into different types based on their functionality:

- **Data Transfer Instructions** (e.g., LOAD, STORE) move data between memory, registers, and I/O devices.
- **Arithmetic and Logic Instructions** (e.g., ADD, SUB, AND, OR) perform calculations and logical operations.
- **Control Instructions** (e.g., JUMP, CALL, RETURN) modify the sequence of execution.
- **Input/Output Instructions** (e.g., IN, OUT) handle interactions with peripheral devices.

The execution of instructions follows the fetch-decode-execute cycle, where the CPU retrieves the instruction from memory, decodes its meaning, and performs the specified operation. Instructions are key to implementing algorithms and controlling hardware behaviour in a computer system.

Instruction Sequencing

In Computer Organization and Architecture (COA), **instruction sequencing** refers to the process of determining the order in which instructions are fetched, decoded, and executed by the CPU. It ensures that the program executes in a logical and intended sequence, enabling correct results and efficient execution.

By default, instructions are executed sequentially, one after another, based on their order in memory. This is governed by the **Program Counter (PC)**, which keeps track of the memory address of the next instruction to execute. However, instruction sequencing can be altered using control instructions that modify the flow of execution. These include:

1. **Branch Instructions**: Conditional (e.g., IF) or unconditional (e.g., JUMP) instructions that redirect execution to a different memory address.
2. **Loops**: Repeatedly execute a block of instructions until a condition is met.
3. **Function Calls and Returns**: Temporarily divert execution to a subroutine (e.g., CALL) and return to the main program afterward (e.g., RETURN).

Modern processors often enhance instruction sequencing through techniques like **pipelining**, where multiple instructions are processed simultaneously at different stages (fetch, decode, execute), and **out-of-order execution**, where instructions are executed based on resource availability rather than their order in memory. These techniques improve CPU performance and throughput while maintaining program correctness.

Addressing Mode

In **Computer Organization and Architecture (COA)**, an **addressing mode** specifies how the operand of an instruction is accessed or referenced. It defines the method used to calculate the effective address of the data that the instruction operates on. Addressing modes provide flexibility in instruction design, enabling efficient data handling and compact program representation.

Types of Addressing Modes

The different types of Addressing Modes are as follows –

1. **Immediate Addressing**:
   - The operand is directly specified within the instruction itself.
   - Example: ADD R1, #5 (adds the value 5 directly to the contents of register R1).
2. **Direct Addressing**:
   - The instruction specifies the memory address where the operand is stored.
   - Example: LOAD R1, 1000 (loads the data at memory address 1000 into register R1).
3. **Indirect Addressing**:
   - The instruction specifies a memory location or register that contains the address of the operand.
   - Example: LOAD R1, (500) (uses the value at address 500 as the address to fetch the operand).
4. **Register Addressing**:
   - The operand is located in a processor register, and the instruction specifies which register to use.
   - Example: MOV R1, R2 (copies the contents of R2 into R1).
5. **Register Indirect Addressing**:
   - A register contains the address of the operand in memory.

    - Example: LOAD R1, (R2) (uses the address in R2 to load data into R1).

6. **Indexed Addressing**:
    - The effective address is calculated by adding a constant offset to the contents of a register.
    - Example: LOAD R1, 1000(R2) (loads data from the address 1000 + contents of R2).
7. **Base-Offset Addressing**:
    - Similar to indexed addressing, but a base register and an offset are combined to calculate the effective address.
8. **Relative Addressing**:
    - The effective address is calculated as the sum of the program counter (PC) and an offset. Commonly used in branching instructions.
    - Example: JUMP 20 (jumps to the instruction located 20 bytes ahead of the current instruction).

Addressing modes enhance the capability of the instruction set by allowing a variety of ways to access data, optimizing performance and memory usage in different scenarios.

CONCLUSION

In conclusion, **instructions**, **instruction sequencing**, and **addressing modes** are fundamental concepts in computer organization and architecture, forming the backbone of how processors execute programs and handle data. Instructions serve as the essential commands that direct the CPU to perform specific operations, while instruction sequencing ensures these operations are executed in a logical and efficient order, supporting both sequential and non-sequential program flows. Addressing modes further enhance the versatility of instructions by defining diverse methods to access operands, allowing for compact, flexible, and efficient program design. Together, these concepts underpin the efficient execution of algorithms and tasks, optimizing processor performance and resource utilization. A solid understanding of these principles is critical for designing robust and high-performing computer systems.

BIBLIOGRAPHY

- https://en.wikipedia.org/
- https://www.geeksforgeeks.org/

ACKNOWLEDGEMENT

I would like to acknowledge all those without whom this project would not have been successful. Firstly, I would wish to thank our Mathematics teacher Mrs. Sankhamita Sinha who guided me throughout the project and gave her immense support. She made us understand how to successfully complete this project and without her, the project would not have been complete.

This project has been a source to learn and bring our theoretical knowledge to the real-life world. So, I would really acknowledge her help and guidance for this project.

I would also like to thank my parents who have always been there whenever needed. Once again, thanks to everyone for making this project successful.