THE UNIVERSITY OF DODOMA

COLLEGE OF INFORMATICS AND VIRTUAL EDUCATION

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Assignment Title: FINAL ASSIGNMENT WORK

Course Name: COMPUTER ORGANIZATION AND ARCHITECTURE I

Course Code: CT 211

Group Number: 7

S/N	NAME	REGISTRATION NUMBER
1	JOSHUA MWAKIPESILE	T24-03-15836
2	PRAISE MASHIKU	T24-03-10358
3	VICTORIA SEPHANIA	T24-03-17605
4	SALEH NASSOR	T24-03-20761
5	SAMWEL SALIM	T23-03-16714

Contents

Design of Instruction Sets: Memory Addressing and Instruction Encoding

Abstract

This report discusses the design of instruction sets in computer organization and architecture, focusing on memory addressing and instruction encoding. An instruction set defines the operations a processor can perform and serves as the main interface between software and hardware. Instruction set design determines how instructions are structured, how data is accessed, and how efficiently a processor executes programs. Understanding instruction sets is important because they directly affect CPU performance, complexity, flexibility, and compatibility between hardware and software.

The report explains how instructions are represented in binary form through instruction formats and encoding. It describes key components of an instruction, including the opcode and operands, and compares fixed length and variable length instruction formats. The report also examines memory addressing and addressing modes, explaining how operands are located in immediate values, registers, or memory. Common addressing modes such as immediate, register, direct, indirect, and indexed addressing are discussed along with the concept of effective address calculation. Finally, the report highlights instruction set design trade-offs from an architectural perspective, including the balance between speed, simplicity, instruction size, and hardware complexity, as well as the differences between RISC and CISC architectures.

1. Introduction

An instruction set defines the complete set of machine-level instructions that a processor can understand, interpret, and execute. It represents the language through which software communicates with hardware and forms a critical part of computer organization and architecture. The design of an instruction set has a direct impact on processor performance, complexity, flexibility, and efficiency.

Instruction set design focuses on defining the operations supported by the CPU, how instructions are formatted and encoded, and how operands are accessed during execution. Two important aspects of this design are memory addressing and instruction encoding. Memory addressing determines how and where data operands are located, while instruction encoding specifies how instructions are represented in binary form so that the processor can decode and execute them correctly.

This report discusses the concept of instruction sets and instruction set architecture (ISA), examines instruction formats and encoding techniques, and explains various memory addressing

modes used in modern processors. The goal is to provide a clear understanding of how these components contribute to effective instruction set design.

2. Instruction Sets and Instruction Set Architecture (ISA)

An instruction set is the collection of binary-coded instructions that specify the operations a CPU can perform, such as arithmetic, logical operations, data transfer, and control flow. It also defines how operands are accessed and how instructions are structured internally. In this sense, the instruction set acts as the interface between hardware and software.

An Instruction Set Architecture (ISA) is the formal specification of an instruction set. It defines the rules that govern how software communicates with the processor, including supported instructions, instruction formats, addressing modes, and available registers. Examples of common ISAs include x86, ARM, and MIPS. While hardware implementations may differ internally, processors that share the same ISA can execute the same machine-level programs.

The purpose of an instruction set is to provide a standardized way for programmers and compilers to control the CPU. Without a defined instruction set, software would have no consistent method to interact with hardware components.

3. Components of an Instruction

A machine instruction is generally composed of two main parts: the opcode and one or more operands.

3.1 Opcode

The opcode (operation code) specifies the action to be performed by the CPU. Examples include ADD, SUB, LOAD, STORE, and JUMP. During execution, the processor decodes the opcode to determine the required operation.

3.2 Operand(s)

Operands specify the data to be used or the location of the data. An operand may represent a register, a memory location, or an immediate value embedded directly within the instruction. The way operands are specified depends on the addressing mode used by the instruction.

4. Instruction Formats and Encoding

Instruction formats and encoding describe how machine instructions are represented internally in binary form. An instruction format specifies the layout of bits within an instruction, indicating how many bits are allocated to the opcode and how many are used for operand or address fields.

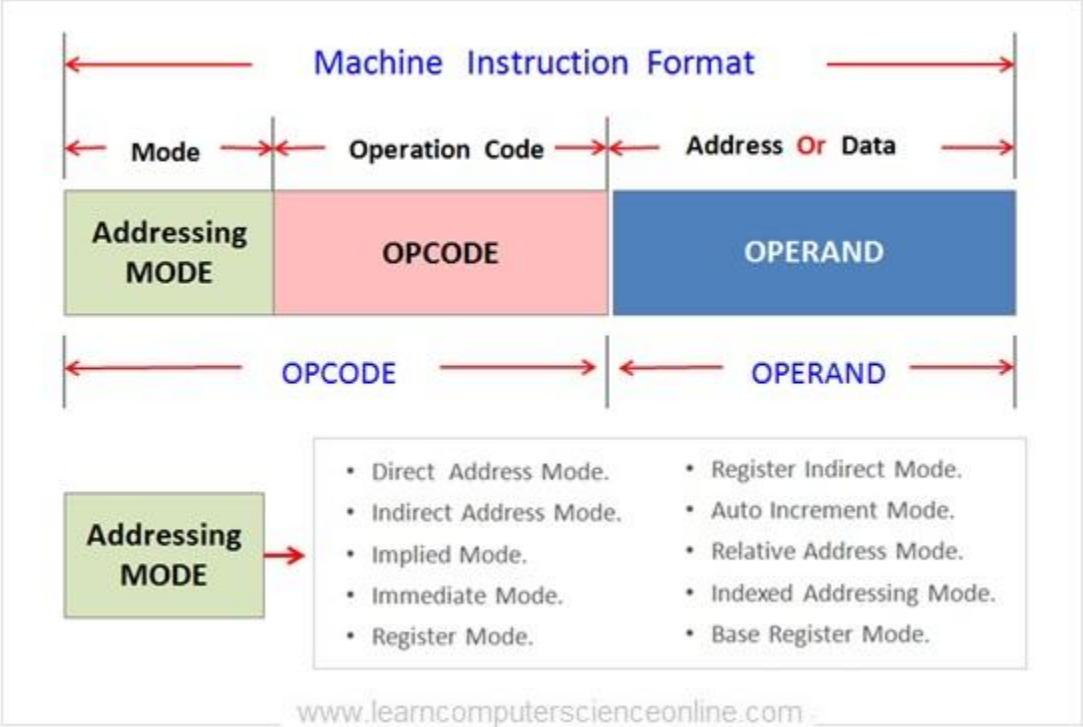

Figure 1 : General Machine Instruction Format showing opcode, addressing mode, and operand fields

Instruction formats may be classified as fixed-length or variable-length. Fixed-length instructions use the same number of bits for all instructions, simplifying instruction decoding and improving execution speed. Variable-length instructions allow instructions of different sizes, which can improve code density but increase decoding complexity.

Instructions are divided into fields, each serving a specific purpose. The opcode field identifies the operation to be performed, while operand or address fields specify where the data is located. Instructions may also be classified based on the number of operands they use, such as zero-address, one-address, two-address, or three-address instructions.

Instruction encoding involves several design trade-offs. Shorter instructions reduce memory usage and improve instruction fetch speed, while longer instructions can support more complex operations and addressing modes. Techniques such as expanding opcodes are used to increase the

number of available instructions without increasing instruction length, assigning shorter opcodes to frequently used instructions and longer opcodes to less common ones.

5. Memory Addressing and Addressing Modes

Memory addressing refers to the method by which an instruction specifies the location of the data required for execution. The operand is the actual data being processed, while the addressing mode determines how the operand is accessed. When the operand resides in memory, the processor computes an effective address, which is the actual memory address from which the data is fetched.

5.1 Immediate Addressing Mode

In immediate addressing mode, the operand is included directly within the instruction. Since no memory access is required to fetch the operand, execution is very fast. However, the size of the operand is limited by the instruction length.

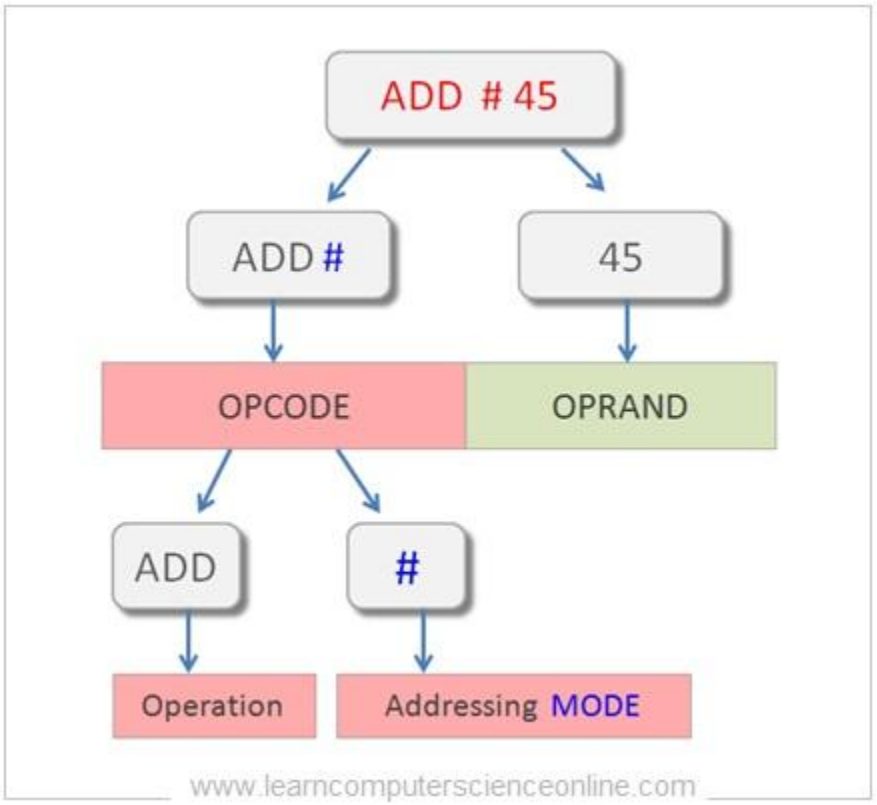

Figure 2: Example of immediate addressing such as ADD #45 showing opcode and operand separation

5.2 Register Addressing Mode

In register addressing mode, the operand is stored in a CPU register specified by the instruction. Accessing data from registers is very fast, but the number of registers is limited.

5.3 Direct Addressing Mode

In direct addressing mode, the instruction contains the memory address of the operand. The effective address is equal to the address specified in the instruction. This mode is simple but slower than register-based addressing due to memory access latency.

5.4 Indirect Addressing Mode

Indirect addressing mode specifies a memory location that contains the effective address of the operand. The processor must first read the effective address and then access memory again to retrieve the operand, resulting in slower execution but increased flexibility.

5.5 Register Indirect Addressing Mode

In register indirect addressing mode, the effective address is stored in a register rather than memory. This reduces memory accesses compared to indirect addressing while maintaining flexibility.

5.6 Indexed and Base Addressing Modes

In indexed or base addressing modes, the effective address is calculated by adding an offset value to the contents of a register. These modes are commonly used for accessing arrays, tables, and sequential data structures.

5.7 Stack Addressing Mode

Stack addressing mode implicitly accesses operands from the top of the stack using a stack pointer. Instructions do not explicitly specify operand addresses, simplifying instruction formats but limiting flexibility.

6. Instruction Set Design Trade-Offs

Instruction set design involves balancing multiple trade-offs related to performance, complexity, and efficiency. Simple instructions are easier and faster to decode and execute, enabling higher clock speeds and efficient pipelining. However, they may require more instructions to perform complex tasks. Complex instructions can reduce instruction count but increase hardware complexity and decoding time.

Another important trade-off is between instruction size and program size. Larger instructions can encode more functionality but consume more memory, while smaller instructions improve memory efficiency but may increase total instruction count. Addressing modes also present a trade-off between flexibility and hardware complexity, as supporting many addressing modes increases decoder complexity.

These trade-offs are reflected in the two major ISA design philosophies: Reduced Instruction Set Computer (RISC) and Complex Instruction Set Computer (CISC). RISC architectures emphasize simple, fixed-length instructions and extensive use of registers, while CISC architectures support complex, variable-length instructions and multiple addressing modes.

7. Conclusion

Instruction set design plays a fundamental role in computer architecture by defining how software interacts with hardware. Memory addressing and instruction encoding are key components of this design, determining how data is accessed and how instructions are represented internally. Different addressing modes and encoding strategies offer varying balances between speed, flexibility, and complexity. Understanding these concepts provides insight into why different instruction set architectures exist and how architectural decisions influence overall system performance and efficiency.

References

i) The essesntials of computer organization and architecture by Linda Null (Pennsylvania State University) and Julia Lobur (Pennsylvania State University) 2003 edition.

ii) https://www.learncomputerscienceonline.com/instruction-format/

iii) https://www.learncomputerscienceonline.com/instruction-format/

iv) https://youtu.be/jTa0w-MxFJE