Experiment No 6

Implementation of RISC-V Assembler

6.1 Objectives:

After completing this experiment, student will be able to:

- Convert Assembly language into Machine code.
- Explore different Instruction formats of RISC-V.
- Describe different fields i.e Fuc3, Func7, Opcode, rs1, and rs2 etc.

6.2 Background Theory

In the RISC-V architecture, machine code consists of binary instructions that the CPU can directly execute. Each instruction typically corresponds to a single operation or action performed by the CPU. Understanding the basics of RISC-V machine code involves knowing its instruction format, instruction types, and how assembly language instructions are translated into machine code.

Basics of RISC-V Machine Code:

6.2.1 Instruction Format

RISC-V instructions have a fixed-length encoding, which simplifies instruction decoding and execution. The instruction format includes fields for opcode, registers, immediate values, and other necessary information. The most common instruction formats in RISC-V are R-type (register), I-type (immediate), S-type (store), B-type (branch), U-type (upper immediate), and J-type (jump).

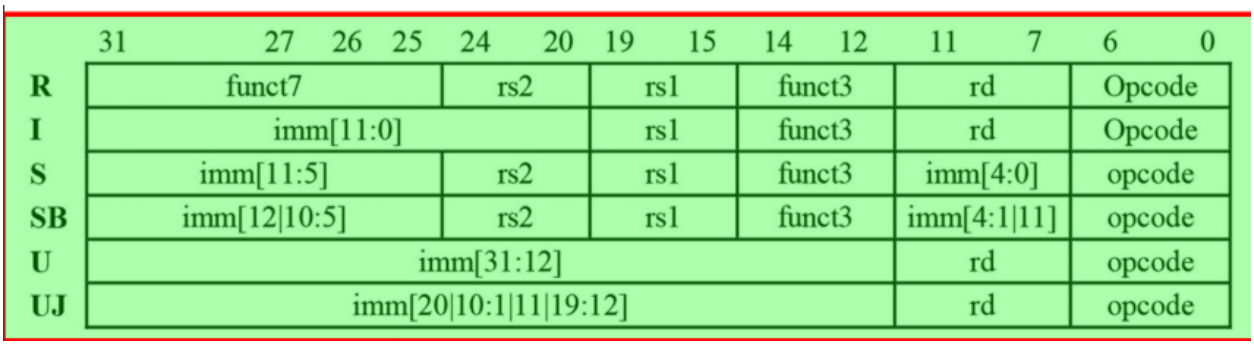
6.2.2 Instruction Types

RISC-V instructions can be categorized into various types based on their operations:

- Arithmetic and logical instructions (e.g., add, sub, and, or)
- Data transfer instructions (e.g., load, store)
- Control transfer instructions (e.g., branch, jump)
- System instructions (e.g., trap, syscall)

6.2.2 Instruction Encoding

Each RISC-V instruction is encoded into a fixed-length binary representation. The opcode field specifies the operation to be performed, while other fields encode register operands, immediate values, or memory addresses.



	31 27	26 25	24 20	19 15	14 12	11 7	6 0
R	funct7		rs2	rs1	funct3	rd	Opcode
I	imm[11:0]			rs1	funct3	rd	Opcode
S	imm[11:5]		rs2	rs1	funct3	imm[4:0]	opcode
SB	imm[12|10:5]		rs2	rs1	funct3	imm[4:1|11]	opcode
U	imm[31:12]					rd	opcode
UJ	imm[20|10:1|11|19:12]					rd	opcode

Figure 6.1: 4-Instruction Encoding

6.2.4 Conversion from Assembly Language to Machine Code:

The process of converting assembly language instructions into machine code involves several steps:

6.2.4.1 Parsing Assembly Instructions

The assembler parses assembly language instructions to identify the mnemonic operation (e.g., add, lw) and its operands (registers, immediate values, memory addresses).

6.2.4.2 Mapping Mnemonics to OpCodes

Each assembly language mnemonic corresponds to a specific opcode in the RISC-V instruction set. The assembler maps these mnemonics to their respective opcodes to determine the operation to be performed.

6.2.4.3 Encoding Registers and Immediate Values

Registers and immediate values specified in assembly instructions are encoded into binary representations according to the instruction format. Registers are typically represented by their register numbers, while immediate values are encoded directly or using sign-extension for signed values.

6.2.4.4 Generating Machine Code

Using the opcode and encoded operands, the assembler generates the binary machine code for each assembly instruction. It constructs the binary instruction by arranging the opcode and operand fields according to the instruction format.

6.2.4.5 Outputting Machine Code

Once all assembly instructions are translated into machine code, the assembler outputs the binary instructions in a format suitable for execution by the CPU. This could be in the form of a binary file, hexadecimal representation, or directly loaded into memory.

By following these steps, the assembler converts human-readable assembly language code into machine-executable binary instructions, allowing the CPU to execute the desired operations defined by the programmer.

6.2.5 Opcode

In computer architecture, an opcode, short for "operation code," is a fundamental component of machine language instructions. It specifies the operation to be performed by the CPU or processor. The opcode field of an instruction determines the type of operation or instruction that the CPU should execute.

Here are some key points about opcodes:

1. **Instruction Type:** Opcodes define the type of operation to be performed, such as arithmetic operations (add, subtract), logical operations (AND, OR), data transfer (load, store), control transfer (branch, jump), and more.

2. **Binary Representation:** Opcodes are represented as binary numbers in the machine code instructions. Each opcode corresponds to a unique binary pattern, which the CPU recognizes and interprets as a specific operation.

3. **Fixed Length:** In many instruction sets, including RISC-V, opcodes have fixed lengths. This simplifies the decoding process, as the CPU can quickly determine the type of operation without needing to analyze variable-length opcode fields.

4. **Instruction Set Architecture (ISA):** The opcode space is defined by the instruction set architecture (ISA) of a processor. Each ISA specifies a set of opcodes along with their corresponding operations and formats.

5. **Example:** In the RISC-V instruction set, the opcode field typically occupies the most significant bits of an instruction. For example, the opcode 0110011 corresponds to R-type instructions, which include arithmetic and logical operations.

So, the opcode is a crucial part of machine language instructions, as it directs the CPU to perform specific operations according to the program's requirements. It serves as a fundamental building block in the execution of programs on a computer system.

In the RISC-V instruction set architecture (ISA), different operations are represented by distinct opcodes. Here are some examples of opcodes for various operations in RISC-V:

1. **Arithmetic and Logical Operations (R-Type Instructions):**

- **add**: **0110011** (Binary)
- **sub**: **0110011**
- **and**: **0110011**
- **or**: **0110011**
- **xor**: **0110011**
- **slt** (Set Less Than): **0110011**
- **sltu** (Set Less Than Unsigned): **0110011**
- **sll** (Shift Left Logical): **0110011**
- **srl** (Shift Right Logical): **0110011**
- **sra** (Shift Right Arithmetic): **0110011**

2. **Data Transfer Operations (I-Type Instructions):**
    - **lw** (Load Word): **0000011** (Binary)
    - **sw** (Store Word): **0100011**
    - **lb** (Load Byte): **0000011**
    - **sb** (Store Byte): **0100011**
3. **Branch and Jump Operations (B-Type and J-Type Instructions):**
    - **beq** (Branch if Equal): **1100011** (Binary)
    - **bne** (Branch if Not Equal): **1100011**
    - **blt** (Branch if Less Than): **1100011**
    - **bge** (Branch if Greater Than or Equal): **1100011**
    - **jal** (Jump and Link): **1101111**
    - **jalr** (Jump and Link Register): **1100111**
4. **Immediate-Type Operations (I-Type Instructions):**
    - **addi** (Add Immediate): **0010011** (Binary)
    - **andi** (AND Immediate): **0010011**

- **ori** (OR Immediate): **0010011**
- **xori** (XOR Immediate): **0010011**
- **slti** (Set Less Than Immediate): **0010011**
- **sltiu** (Set Less Than Immediate Unsigned): **0010011**
- **slli** (Shift Left Logical Immediate): **0010011**
- **srli** (Shift Right Logical Immediate): **0010011**
- **srai** (Shift Right Arithmetic Immediate): **0010011**

5. **System Operations (System-Type Instructions):**
    - **ecall**: **1110011** (Binary)
    - **ebreak**: **1110011**
    - System calls and other privileged instructions have opcodes within the **1110011** opcode space.

These are just some examples of RISC-V opcodes for different operations. Each opcode corresponds to a specific operation or instruction, and they play a crucial role in the decoding and execution of instructions by the CPU in a RISC-V processor.

6.3 Understanding how to convert an assembly code into machine code

Let's walk through an example of converting a simple RISC-V assembly language instruction into machine code.

Example: Assembly to Machine Code Conversion

Consider the following RISC-V assembly instruction:

add x1, x2, x3

This instruction adds the values in registers x2 and x3 and stores the result in register x1.

Step 1: Parsing Assembly Instruction

- Operation: **add**
- Operands: **x1**, **x2**, **x3**

Step 2: Mapping Mnemonics to OpCodes

The **add** operation corresponds to the **add** opcode in the RISC-V instruction set.

Step 3: Encoding Registers

- Register x1: Encoded as **001** (binary representation of register number)
- Register x2: Encoded as **010**
- Register x3: Encoded as **011**

Step 4: Generating Machine Code

The RISC-V R-type instruction format for **add** instructions is as follows:

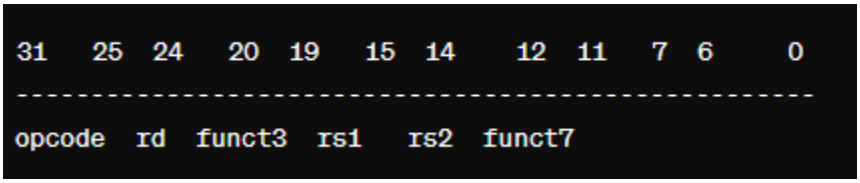


Figure 6.2: Machine Code Generation

- Opcode for **add**: **0110011**
- **rd** (destination register): Encoded register x1 (binary **001**)
- **rs1** (source register 1): Encoded register x2 (binary **010**)
- **rs2** (source register 2): Encoded register x3 (binary **011**)
- **funct3** for **add** instruction: **000**
- **funct7** for **add** instruction: **0000000**

Concatenating these fields together, the machine code instruction becomes:

0110011 00001 000 00010 00011 0000000

Step 5: Outputting Machine Code

The assembled machine code for the **add x1, x2, x3** instruction is **0x00A32333** in hexadecimal representation.

In this example, we converted the RISC-V assembly instruction **add x1, x2, x3** into its corresponding machine code **0x00A32333**. This process involves parsing the assembly instruction, mapping mnemonics to opcodes, encoding registers, generating the machine code according to the instruction format, and outputting the final machine code representation.

6.4 Conversion of Machine Code into Binary

The hexadecimal number 0x00A32333 represents a machine code instruction in RISC-V architecture. Let's break down this hexadecimal number into its binary representation and interpret its components:

Hexadecimal Representation:

0x00A32333

Binary Representation:

0000 0000 1010 0011 0010 0011 0011 0011

Interpretation:

The RISC-V instruction set architecture (ISA) typically divides machine code instructions into fields representing different parts of the instruction. Let's interpret the binary representation based on the standard RISC-V instruction formats.

Instruction Format:

RISC-V instructions can have different formats, such as R-type (register), I-type (immediate), S-type (store), B-type (branch), U-type (upper immediate), and J-type (jump). We'll need to identify the instruction format to interpret the binary representation accurately.

Interpretation Based on Opcode:

The opcode is the most significant part of the instruction and determines the operation to be performed. Let's extract the opcode from the binary representation.

In this case, the opcode appears to be 0000011. This opcode corresponds to the lw (Load Word) instruction in the RISC-V ISA.

Remaining Fields:

After the opcode, the binary representation contains fields for various components of the instruction, such as source and destination registers, immediate values, etc.

Since lw is an I-type instruction, it typically includes fields for the destination register (rd), base register (rs1), and immediate value (imm).

Interpreting the Load Word (lw) Instruction:

The lw instruction loads a 32-bit word from memory into a register.

The destination register (rd) is encoded as 00001.

The base register (rs1) is encoded as 00010.

The immediate value (imm) is encoded as 101000110010011.

Based on the interpretation, the machine code instruction 0x00A32333 corresponds to the RISC-V lw x1, 848(x2) instruction, where it loads a word from memory at address 848 plus the value in register x2 and stores it in register x1.

This breakdown illustrates how to interpret a hexadecimal machine code instruction in RISC-V and understand its components based on the RISC-V instruction format.

6.5 Lab Activities

6.5.4 Task-1: Convert at last three R-Type, S-Type and I-Type instructions into Machine code as well as explore its binary, Opcode, Functions, Operands and Destination Registers.

6.5.5 Task-2: Convert at least three B-Type and J-Type instructions into Machine code as well as explore its binary, Opcode, Functions, Operands and Destination Registers.

Lab Exercise and Summary

Summary should cover Introduction, Procedure, Data Analysis and Evaluation.

Student's Signature: _____ **Date: _____**

Cognitive Activity

Q1. Show how the R-type instruction 'add x3, x1, x2' would be broken into its instruction fields. List the fields in order (opcode, rd, funct3, rs1, rs2, funct7) and state what each one identifies for this instruction. **(5)**

Ans: _____

_____

_____

_____

_____

Q2. The instructions 'add' and 'sub' share the same opcode and the same funct3 value in RISC-V. Which field actually distinguishes them, and why do you think the ISA is designed this way instead of giving every instruction its own unique opcode? **(5)**

Ans: _____

_____

_____

_____

_____

Obtained Marks	**/10**

Psychomotor Activity 1 — Experimental Setup

S. No	Criteria	Performance	Description	Marks
1	Tool Setup (4)	Level 4	Opens and configures Venus and selects the required environment correctly and independently.	4
		Level 3	Configures the required tool with minor errors or assistance.	3
		Level 2	Completes the setup with considerable assistance.	2
		Level 1	Setup is incomplete and requires substantial assistance.	1
2	File/Design Setup (4)	Level 4	Correctly creates/selects the required assembly file, Logisim circuit/design and required components for the RISC-V machine code translation.	4
		Level 3	File/design setup is mostly correct with minor errors or omissions.	3
		Level 2	Setup is partially completed and requires assistance.	2
		Level 1	Major errors or omissions are present in the setup.	1
3	Parameter Setup (2)	Level 2	Correctly defines required data, registers, parameters, components and initial conditions.	2
		Level 1	Most required inputs/components are correctly configured with minor errors.	1
			Total Obtained	/10

If a student is absent:

Criteria	Description	Marks
Absent	Student is absent from the laboratory activity.	0

Psychomotor Activity 2 — Implementation + Results

S. No	Criteria	Performance	Description	Marks
1	**Implementation (4)**	Level 4	Implements the required RISC-V machine code translation correctly and independently.	**4**
		Level 3	Implements the required program/design with minor errors or assistance.	**3**
		Level 2	Partially implements the program/design and requires considerable assistance.	**2**
		Level 1	Implementation is largely incomplete and requires substantial assistance.	**1**
2	**Execution/Simulation (2)**	Level 2	Executes the RISC-V program or simulates the Logisim design correctly and handles basic errors appropriately.	**2**
		Level 1	Executes/simulates the work with minor errors or assistance.	**1**
3	**Results/Output (4)**	Level 4	Obtains correct results and clearly demonstrates the required program output or circuit behavior.	**4**
		Level 3	Obtains mostly correct results with minor errors.	**3**
		Level 2	Obtains partially correct results with noticeable errors or omissions.	**2**
		Level 1	Results are largely incorrect or incomplete.	**1**
			Total Obtained	**/10**

If a student is absent:

Criteria	Description	Marks
Absent	Student is absent from the laboratory activity.	**0**

Affective Activity — Lab Report + Viva Voce

S. No	Criteria	Performance	Description	Marks
1	Lab Report Completeness & Organization (4)	Level 4	Report is complete, well organized and demonstrates full responsibility for the assigned work.	4
		Level 3	Report is complete with minor omissions or organizational issues and demonstrates satisfactory responsibility for the assigned work.	3
		Level 2	Report is partially complete with several omissions and demonstrates limited responsibility for the assigned work.	2
		Level 1	Report has major omissions or poor organization and demonstrates minimal responsibility for the assigned work.	1
2	Code/Design, Results & Conclusion (2)	Level 2	RISC-V machine code translation, results, and conclusion are correctly presented and discussed, demonstrating ownership of the assigned work.	2
		Level 1	RISC-V machine code translation, results, or conclusion are presented with minor errors or omissions.	1
3	Viva Voce (4)	Level 4	Clearly explains the assigned RISC-V machine code translation, working, and results, demonstrating individual responsibility, participation, and effective communication.	4
		Level 3	Explains most of the assigned program/design and results correctly, with minor difficulty, and demonstrates good individual participation.	3
		Level 2	Explains basic aspects of the assigned program/design with some prompting and demonstrates basic participation.	2
		Level 1	Demonstrates limited understanding or involvement and requires substantial prompting to explain the assigned work.	1
			Total Obtained	**/10**

The student has not submitted the report and has not appeared in the Viva Voce.

Criteria	Description	Marks
Lab Report and Results Not Submitted	Student has not submitted the report and results file.	**0**
Absent in Viva Voce	Student is absent from the viva voce.	**0**

Instructor's Signature: _____ **Date:** _____