

**North South University**

**Department of Electrical and Computer Engineering**

**CSE332- Project-Summer2022**

**Tittle: Design a 10-bit Custom RISC-V Microprocessor**

**Submitted To:**

**Dr. Mainul Hossain (Mho1)**

Professor Department of Electrical and Computer Engineering

North South University

**Submitted By: [Section-10, Group-2]**

|  |  |
| --- | --- |
| **Azmine Mahtab Chowdhury** | **1911517642** |
| **Tahmina Sultana Sarah** | **2013233042** |
| **MD. Najmus Sakib** | **2021250042** |
| **Mahmodul Hasan Jobayer** | **2021910642** |

**Submission Date:** 29th August, 2022

# Part 1.1 ISA Design

**Introduction:** Our task was to design a 10-bit RISC type ISA. Objectives: Our objectives was to design a 10 Bit ISA which can solve a particular problem i. e. arithmetic addition, branching etc.

1. **How many types of instructions (R – Type, I – Type, J – Type)?**

Ans: In our project we have decided to use two of the three types of instructions; that is we are using only

Register Type (R- Type)

Immediate Type (I-Type)

1. **Describe each of the formats (field and field length).**

**R-TYPE:** Since, we are here to plan a 10 bit RISC type CPU, so for the ease of our design, we have taken 4 bit for opcode, 2 bit for rd, 2 bit for rs, 2 bit for rt.

-rs: Source register specifier

-rt: Soure/destination register speicifier (in this format we will use rt as another source register) -rd: Destination register specifier

|  |  |  |  |
| --- | --- | --- | --- |
| OP Code | rs | rt | rd |
| 4 bits | 2 bits | 2 bits | 2 bits |

**I – Type:** Again, for the ease of my design, I have taken 4 bit for OP Code, 2 bit for rs, 2 bit for rd and 2 bit for Immediate to perform operations such as addi, subi, load and store.

-rs: Source register specifier

-rt: destination register (for this type only)

Immediate: it’s for adding subtracting an immediate value, also for branching and address displacement.

|  |  |  |  |
| --- | --- | --- | --- |
| OP Code | rs | rt | Immediate |
| 4 bits | 2 | 2 bits | 2 bits |

**3. How many operands (2 operands, 3 operands)?**

- There are three operands, which we represented as d, s and t.

**4. How many operations?**

We have 7 operations. .

## List of registers

- As we have allocated 2 bits for the registers so we will have 2^2 = 4 registers and all of them will be store type

|  |  |  |  |
| --- | --- | --- | --- |
| **Register name** | **Type** | **Reg.Number** | **Binary Value** |
| **$s1** | **Saves values** | **0** | **00** |
| **$s2** | **Saves values** | **1** | **01** |
| **$s3** | **Saves values** | **2** | **10** |
| **$s4** | **Saves values** | **3** | **11** |

**5. Types of operations (Arithmetic, logical, branch type?? How many from each category? List the OP Codes and respective binary values)**

There will be in total 3 different types operations. The categories are:

* Arithmetic
* Logical
* Data transfer

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Category** | **Operation** | **Name** | **Type** | **Opcode** | **Syntax** |
| Logical | AND | AND | R | 0000 | AND $rd, $rs, $rt |
| Logical | OR | OR | R | 0001 | OR $rd, $rs, $rt |
| Arithmetic | Subtraction | sub | R | 0010 | sub $rd, $rs, $rt |
| Arithmetic | Addition | add | R | 0011 | add $rd, $rs, $rt |
| Arithmetic | Addition immediate | addi | I | 0100 | addi $rt, $rs, const |
| Data  Transfer | Load | lw | I | 0101 | lw $rt, $rs, offset |
| Data  Transfer | Store | sw | I | 0110 | sw $rs, $rt, offset |

Arithemetic Operation:

**add:** rd = rs+ rt

Operands A and B stored in register locations rs and rt are added and written to the destination register specified by rd.

**sub:** rd = rs - rt

Operand B (rt) is subtracted from Operand A (rs) and written to rd

**addi**: rd = rs + const

Operand A stored in register rs and an immediate value (constant) are added and written to rd

Logical\_Operation:

**OR:** rd = rs OR Rt

Operand A (rs) is bitwise xored with Operand B (rt) and written into rd

**AND:** rd = rs AND Rt

Operand A (rs) is bitwise anded with Operand B (rt) and written into rd

Data Transfer:

**lw**: rt= MEM[rs + offset]

the value inside rs is considered the base address of the memory and an offset value is added with it, which results in another address of a location and its value is loaded in rt register. **sw**: MEM[rt + offset] = rs

the value inside rt is considered the base address and an offset value is added with it, which gives an address of a location in the memory in which the value inside rs register is stored.