

North South University

*Project title: Design a 12-bit Custom RISC-V Microprocessor*

ISA Design

Course: CSE 332 - Computer Organization & Design

Section – 06

Submitted to:

Submitted by: Group Members

|  |  |
| --- | --- |
| Name | ID |
| Kazi Aniya Ahmed | 2011775642 |
|  |  |

**Introduction:** We have to perform a task to design a 12-bit RISC type of CPU.

**How many types of instruction ?**

There will be 3 types of instruction

* R-Type,
* I-Type,
* J-Type

**Describe each of the formats-**

A 12 bit ISA with following fields. We need to design a register file for this ISA. The formats of the instruction are as follows:

**(R-Type) ISA Format:**

|  |  |  |  |
| --- | --- | --- | --- |
| **opcode** | **rd** | **rs** | **rt** |
| 3 bit | 3 bit | 3bit | 3 bit |

**(I-Type) ISA Format:**

|  |  |  |  |
| --- | --- | --- | --- |
| **opcode** | **rd** | **rt** | **Immediate** |
| 3 bit | 3 bit | 3 bit | 3 bit |

**(J-Type) ISA Format:**

|  |  |
| --- | --- |
| **opcode** | **target** |
| 6 bit | 6 bit |

**How many Operands?**

There we will be use two operands for this ISA, which we are represent as d and s. The types of operands will be

- Register based

- Memory based

**How many Operations?**

We allocated 3 bits for the opcode, so the number of instructions can be executed is 23 or 8.

**Types of Operations?** There will be in total four different types operations.

The categories are:

- Arithmetic

- Logical

- Data transfer

- Conditional Branch

- Unconditional Branch

List the opcodes and respective binary values:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Catagory** | **Operation** | **Name** | **Opcode** | **Type** | **Syntax** | **Comments** |
| Arithmetic | Addition as Subtraction | ***sub*** | 000 | R | **sub $rs, $rt** | Two register operands |
| Arithmetic | Addition | ***add*** | 001 | R | **add $rs, $rt** | Two register operands |
| Arithmetic | Addition immediate | ***addi*** | 010 | I | **addi $rs, const.** | Used to add constants |
| Conditional | Branch on equal | ***beq*** | 011 | I | **Beq $rd, offset** | Check equality if else condition |
| Logical | Shift left logical | ***sll*** | 100 | R | **Sll $rd, offset** | Shift left by constant |
| Data Transfer | Store word | ***sw*** | 101 | I | **Sw offset($rs)** | Load data from memory to register |
| Unconditional | Jump | ***Jmp*** | 110 | j | **Jmp offset** | Jump to Given location |
| Data Transfer | load word | ***lw*** | 111 | I | **lw $rs, offset** | Load data from memory to register |

# List of registers

As we have allocated 3 bits for the registers so we will have 23 = 8 registers and all of them will be store type. We have selected registers from $zero, R1-R7 and assigned 3 bits for each of the registers as we know in the instruction field in MIPS containing the register rs, rt and rd contains 3 bits.

|  |  |  |
| --- | --- | --- |
| Register name | Reg. Number | Binary Value |
| **R0** | 0 | 000 |
| **R1** | 1 | 001 |
| **R2** | 2 | 010 |
| **R3** | 3 | 011 |
| **R4** | 4 | 110 |
| **R5** | 5 | 111 |
| **R6** | 6 | 110 |
| **R7** | 7 | 111 |

Detailed Instructions and Their OperationsInstruction Description

**Add**: It adds two registers and stores the result in the first register.

• Operation: **d = s + t**

# • Syntax: add $rs, $rt (*$rd = $rs +$rd* )

# Sub : Each bit of input is subtracted from the corresponding bit of input and the difference appears at the output of each full adder along with any borrow out.

• Operation: **d = s – t**

# • Syntax: sub $rs, $rt ( *$rd = $rs -$rt* )

**Addi** : It adds a value from register with an given integer value and stores the result in destination register.

# • Operation: d = s + constant

# • Syntax: addi $rs, Constant(x) (*$rd = $rs +const(x).*)

**lw:** It loads required value from the memory to the register for calculation

.

* Operation: d = M [s + offset]
* Syntax: **lw $rs, offset (** *$rd =Mem[$rs+offset]* **)**

**sw:** It stores specific value from register to memory.

# • Operation: M[d + offset] = s

• Syntax: **sw $rs, offset (** *Mem[$rs+offset] =$rd* **)**

**beq:** It checks whether the values of two register s are same or not. If it is same it performs the following operation • Operation: if (d == s) jump to offset else goto next line

## • Syntax: beq $rd, offset (if $rd == R0, goto offset location )

**Sll :** Shift left by constant we can use sll for multiplication

# sll $rd,const. ( $rd = $rd << const. )

**Translating Some codes using our Designed 8 Bit ISA**

Assigned register for individual variable: ***R0, R1, R2, R3***

1. g = A[i]

**sll R2,** 1 # here R2 = i, i = i\*2

**add R1, R2** # here, R1 = base address of A, R2 = i

**lw R1, 0 #** we have fetched the value that was stored in A[i]

**sub R3,R3** #initialize R3 =0 for moving data into R3

**add R3, R2** # here we stored the value of A[i] in R3

1. g = g + c

**add R1, R2**

1. g = g – c **sub R1, R2**