**Project Report: MIPS Processor Design and Analysis**

**Group Members:**

1. **Sazid Hasan (ID: 2211513042)**
2. **Momtahina Quyyum(ID: 2211889042)**
3. **Adnan Arafat(ID: 2132456642)**
4. **Abir Bin Faruque (ID: 2131101642)**

**1. How many operands?**

In the MIPS instruction set architecture (ISA), instructions can have different numbers of operands, typically ranging from zero to three. Here are the types of instructions and their typical operand counts:

* **R-type (Register) instructions:** 3 operands (two source registers and one destination register)
* **I-type (Immediate) instructions:** 2 or 3 operands (one or two source registers, one immediate value, and one destination register)
* **J-type (Jump) instructions:** 1 operand (target address)

**2. Types of operands**

The MIPS processor uses a mix of register-based and memory-based operands:

* **Register-based operands:** Operands that are stored in the CPU registers.
* **Memory-based operands:** Operands that are stored in the main memory.

**3. How many operations? Why?**

The MIPS instruction set includes a variety of operations, each designed to perform a specific task efficiently. The number of operations is determined by the need to support a wide range of computational tasks, from basic arithmetic to complex control flow operations.

Here is a list of typical operations in the MIPS ISA:

* **Arithmetic operations:** Addition, subtraction, multiplication, and division.
* **Logical operations:** AND, OR, XOR, NOR.
* **Data transfer operations:** Load and store instructions.
* **Control flow operations:** Branch and jump instructions.
* **Shift operations:** Logical and arithmetic shifts.

**4. Types of operations**

The operations can be categorized into arithmetic, logical, branch, and data transfer operations. Below is a table listing the instructions, their types, opcodes, and functionality.

| **Instruction** | **Type** | **Opcode** | **Functionality** |
| --- | --- | --- | --- |
| ADD | Arithmetic | 000000 | Adds two registers and stores the result in a register |
| SUB | Arithmetic | 000000 | Subtracts one register from another |
| AND | Logical | 000000 | Bitwise AND between two registers |
| OR | Logical | 000000 | Bitwise OR between two registers |
| LW | Data Transfer | 100011 | Loads a word from memory into a register |
| SW | Data Transfer | 101011 | Stores a word from a register into memory |
| BEQ | Control Flow | 000100 | Branches if two registers are equal |
| J | Control Flow | 000010 | Jumps to a specified address |

**5. Instruction formats**

MIPS instructions can be categorized into three main formats: R-type, I-type, and J-type. Each format has a specific structure with different fields. Below are the formats with field names and bit lengths.

**R-Type Format**

| **Field Name** | **Bits** |
| --- | --- |
| opcode | 6 |
| rs | 5 |
| rt | 5 |
| rd | 5 |
| shamt (shift) | 5 |
| funct | 6 |

**I-Type Format**

| **Field Name** | **Bits** |
| --- | --- |
| opcode | 6 |
| rs | 5 |
| rt | 5 |
| immediate | 16 |

**J-Type Format**

| **Field Name** | **Bits** |
| --- | --- |
| opcode | 6 |
| address | 26 |

**6. List of registers**

MIPS has 32 general-purpose registers. Below is a table of these registers along with their initial values.

| **Register** | **Name** | **Initial Value** |
| --- | --- | --- |
| $zero | Zero | 0x00000000 |
| $at | AT | Assembler Temporary |
| $v0-$v1 | V | 0x00000000 |
| $a0-$a3 | A | Argument Registers |
| $t0-$t7 | T | Temporary Registers |
| $s0-$s7 | S | Saved Registers |
| $t8-$t9 | T | Temporary Registers |
| $k0-$k1 | K | Reserved for OS |
| $gp | GP | Global Pointer |
| $sp | SP | Stack Pointer |
| $fp | FP | Frame Pointer |
| $ra | RA | Return Address |

In a MIPS processor simulation, the initial values of the registers can be loaded from a file (registers.mem) during the testbench setup.

**Conclusion**

This project report outlines the operand types, operations, instruction formats, and registers used in an MIPS processor. The detailed analysis helps in understanding the structure and functionality of the MIPS architecture, making it easier to implement and simulate.

**All the codes have been attached to the file.**