### Instruction Set Architecture (ISA) Design

Processor Architecture Style

#### RISC Architecture - The architecture has simplified instruction set, making instructions easier to decode and execute, thus; the simplicity allows for faster execution of instructions.

Registers Bank

* **General-Purpose Registers**: 4 registers
* **Special Purpose Registers**: Include dedicated ones for specific tasks, like a program counter (PC), and a stack pointer (SP).

|  |  |  |
| --- | --- | --- |
| **Register** | **Binary** | **Use** |
| zero | 000 | Constant value 0 |
| s1 | 001 | Saved register 1 |
| a0 | 010 | Argument register 0 |
| sp | 011 | Stack pointer |
| ra | 100 | Return address |
| s0 | 101 | Saved register 0 |
| v0 | 110 | Return value register 0 |

Addressing Modes

* **Immediate Addressing**: Great for constants, like setting up initial values.
* **Direct Addressing**: Access specific memory locations directly, useful for reading/writing messages.
* **Register Addressing**: Perform operations on data stored in registers to boost speed and efficiency.
* **Indirect Addressing**: Handy for accessing data structures like arrays or linked lists.

Memory Architecture, Endianness, and Alignment

* **Memory Architecture**: Harvard Architecture, the CPU can fetch an instruction and access data in the same clock cycle, and this leads to faster execution cycles, memory bottlenecks are also reduced.
* **Endianness**:Big endian.
* **Alignment**: Aligned, ensures faster memory access.

Instruction Format

* **Simple Format**: Keep it straightforward to minimize complexity.

#### Instruction Types

**Arithmetic Operations**: Basic math operations needed for processing transactions, like adding balances.

Examples: ADD, SUB

**Logical Operations**: Used for error checking and ensuring data integrity.

Examples: AND, OR, NOT, XOR

**Control Flow Instructions**: Manage the flow of execution, like conditional branching for error handling.

Examples: JUMP, CALL, RETURN

#### Power Management Instructions

* **Low-Power States**: Instructions for entering low-power modes, like SLEEP and WAKE.
* **Dynamic Voltage Scaling or Clock Gating**: Include instructions to adjust the processor’s voltage and frequency dynamically to save power.

#### Error Detection and Correction

* **Error Checking**: Instructions for checking and correcting errors, like CHECK\_ERROR and CORRECT.
* **Redundancy Checks**: Implement mechanisms like CRC (Cyclic Redundancy Check) to ensure data integrity during communication.

Instruction Set Table

| **Instruction** | **Opcode** | **Operand1** | **Operand2** | **Description** |
| --- | --- | --- | --- | --- |
| LOAD A, addr | 0000 | A | addr | Load value from memory address addr into register A |
| STORE A, addr | 0001 | A | addr | Store value from register A into memory address addr |
| ADD A, B, C | 0010 | A | B, C | Add values in registers A and B, and store in C |
| SUB A, B, C | 0011 | A | B, C | Subtract values in registers A and B, and store in C |
| JUMP addr | 0111 | addr | - | Jump to the specified memory address |
| SLEEP | 1000 | - | - | Enter low-power mode |
| CHECK\_ERROR | 1001 | - | - | Check for errors in the last operation |
| beq A, B, offset | 1010 | A,B | offset | If values in register A and B are equal, set PC to relative branch |

Conclusion

This project focused on designing an 8-bit processor simulator for low-cost, battery-efficient communication devices. By using a simple RISC architecture, we ensured the processor is efficient and easy to implement.

We included a small set of general-purpose and special-purpose registers,to make the processor flexible. The Harvard architecture organization helps optimize memory access.

Instruction set covers essential operations like arithmetic, logic, control flow, power management, and error detection, making the processor reliable and efficient. Power management instructions help to enhance performance and reliability.

This design is well-suited for communication devices in areas with limited infrastructure, helping improve connectivity and financial services.

**References**

1. Patterson, D.A. and Hennessy, J.L., 2014. *Computer organization and design: The hardware/software interface*. 5th ed. Amsterdam: Elsevier.