**Overview and Motivation**

* The ISA design will be an embedded system since it will be implanted in low mobile phones to help increase their performance, while maintaining low power consumption along with low cost.

**Architectural Design Choice**

* We use RISC style because it runs applications faster since it executes more instructions per second and it is also easier to pipeline.RISC favors simplicity and efficiency.
* RV64 are best choice for AI workloads since they balance performance, memory(greater than 4GB which is necessary for large AI models), and compatibility. While RV32 can only address 4GB or less memory. RV64 can support a wider range of data types and larger registers, which can improve performance for AI tasks that involve significant numerical computation.RV64 can handle more complex calculations and larger data-sets, making it more suitable for intensive AI tasks.
* 32-bit provides a good balance for general-purpose operations and model calculation. 64-bit which is essential for high-precision tasks that demand accuracy, such as deep learning and complex algorithms.
* **Immediate Addressing** - Immediate values are directly available in the instruction, reducing the need for additional memory access. Minimizes instruction cycles for accessing frequently used constants, enhancing overall execution speed. Ideal for constants (e.g., initialization parameters) that do not change, simplifying the code and reducing overhead. **PC-Relative Addressing** - Facilitates efficient implementation of loops and conditional statements, which are prevalent in AI algorithms. Enables easier code relocation, making it suitable for dynamic environments or systems with varying memory layouts. Reduces the need for absolute addressing calculations, speeding up execution of control flow operations. **Load/Store Addressing** - Essential for efficiently transferring large amounts of data between memory and registers, crucial for handling AI model parameters and data-sets. Supports the high data throughput required for training and inference in AI applications. Enables efficient access patterns that match the structure of AI workloads, improving cache utilization and reducing latency. **Base Addressing** **(Register Indirect)** - Allows dynamic access to arrays and matrices, which are common in AI models for weights and features. Facilitates working with variable-sized data structures and batch processing in neural networks. Supports efficient data management, especially when handling large data-sets or models. These modes enhance the overall efficiency of algorithms used in voice assistance, auto-correction, and offline translation, aligning with the computational demands of modern AI applications.
* Choosing little-endian for RV64 aligns with performance, compatibility, and ease of integration, making it well-suited for the demands of AI workloads. This choice helps ensure efficient processing and data handling in applications like voice assistance, auto-correction, and offline translation. In little-endian format, the least significant byte is accessed first, which can lead to performance benefits in certain arithmetic operations and data manipulations common in AI workloads.
* Both fixed-length and variable-length instruction formats have their advantages and disadvantages, a fixed-length instruction format is preferable for RV64 when targeting AI workloads. This choice enhances performance, simplifies instruction decoding, and supports efficient compiler optimization, making it well-suited for applications like voice assistance, auto-correction, and offline translation.

**Instruction Set Summary**

|  |  |  |  |
| --- | --- | --- | --- |
| **Category** | **Mnemonic(s)** | **Operands** | **Description** |
| **Voice Command** | VCALL | REG, STRING | Initiates a voice command for calling or messaging |
| Security unlock | SECUNLOCK | PIN, PASSWORD, REG | Unlocks device or transactions using security credentials. |
| Text prediction | PREDICT | REG, STRING | Suggests the next word or corrects spelling in messages. |

**Instruction Encoding Summary**

RISC-V 64-bit instruction encoding includes several formats, each designed for efficient processing. The R-type format uses 64 bits, with 7 bits for the opcode (which tells the CPU what to do), 5 bits for the destination register (rd), 3 bits for funct3 (indicating the operation type), 5 bits for the first source register (rs1), 5 bits for the second source register (rs2), and 7 bits for funct7 (additional details). The I-type format also has 64 bits, featuring a 12-bit immediate value, 5 bits for rs1, 3 bits for funct3, 5 bits for rd, and 7 bits for the opcode.

In the S-type format, the instruction is 64 bits, starting with the opcode, followed by 5 bits for rs2, 5 bits for rs1, 3 bits for funct3, and a 12-bit immediate split across different fields. The B-type format keeps the 64-bit structure, with 7 bits for the opcode, 5 bits each for rs2 and rs1, 3 bits for funct3, and a 12-bit immediate value in different parts.

The U-type format uses 20 bits for the immediate value, 5 bits for rd, and 7 bits for the opcode. Finally, the J-type format also has 64 bits, with 20 bits for the immediate value, 5 bits for rd, and 7 bits for the opcode. Overall, all formats use 7 bits for the opcode and 5 bits for registers, allowing for 32 registers, while immediate values differ by type, making decoding straightforward and execution efficient.

**Design Rationale & Trade-offs**

This RV64 ISA design deliberately prioritizes hardware simplicity and performance over code density, adhering to a strict RISC philosophy. By using a minimal, regular instruction set with fixed 32-bit encoding, it simplifies processor design—enabling efficient pipelining, lower power consumption, and cheaper implementation. The trade-off is potentially larger program sizes, but this is addressed through optional compression extensions. Built with future-proofing in mind, its modular structure allows for seamless addition of specialized instruction sets, ensuring long-term relevance and adaptability without sacrificing its core elegant simplicity.