RISC-V Core on FPGA
Zayam Tariq
Logan Liberty
Pranav the PhD Student

(BROAD) steps to take:

1. ISA
2. High Level uArch specifications/system features
3. High Level System Diagram of Core CPU, Bus, Peripherals, Memory, Controller FSMs, Arbiter.
4. Pipeline Diagram
5. Datapath Details

GOAL:

Run Linux, be able to interface with Keyboard and Mouse.

ISA

RISC-V "RV32IMZicsr" ISA (https://msyksphinz-self.github.io/riscv-isadoc/html/rvi.html#beq)

- 32 bit instructions
- Typical RISC-V RV32I core, with Multiplication and ZICSR extension
    - Allows multiplication, division, and privileged/supervisor mode to be enabled
- 32 Integer Registers. Register Definitions found here: https://msyksphinz-self.github.io/riscv-isadoc/html/regs.html

High Level uArch Specifications/System Features

High Level Specs

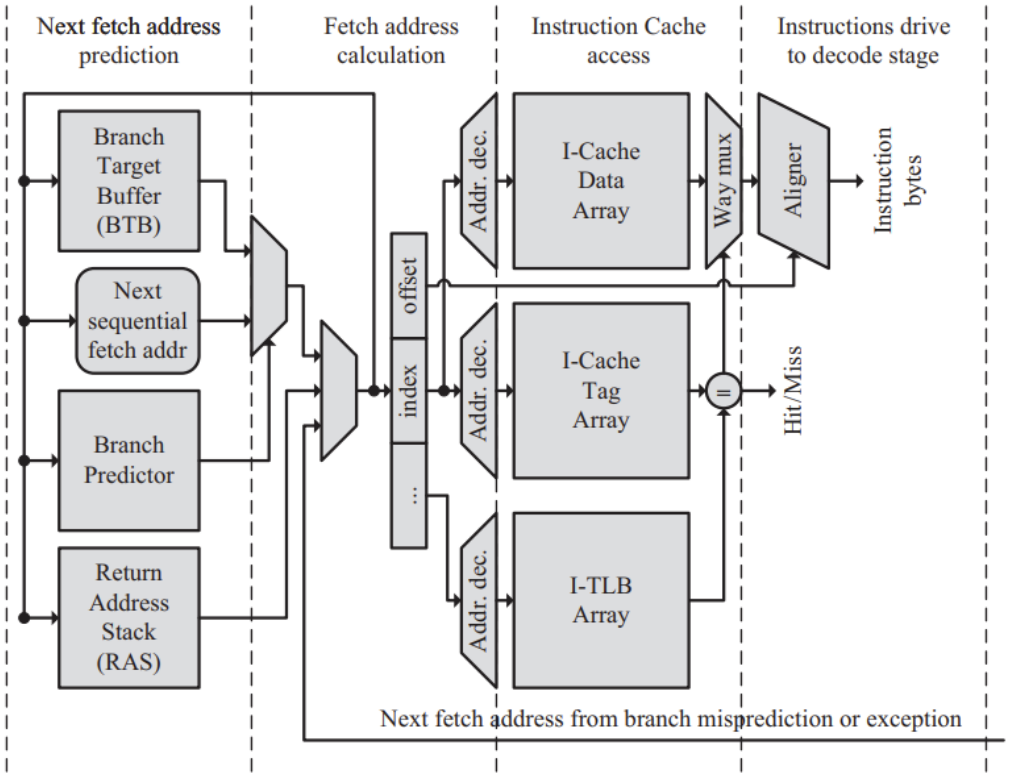
1. Pipelined
    a. Fetch, Decode, Rename, Issue, Execute, Writeback, Commit
    b. Accesses to Instruction Cache (Fetch Stage) will be further pipelined for higher clock frequency
        i. Parallel Tag and Data Array access
    c. Accesses to Data Cache (Execute Stage) will be further pipelined for higher clock frequency
        i. Serial Tag and Data Array access (Verify Tag exists -> Go to Data Array)
2. Out-of-Order
3. Superscalar

Cache Specifications

1. Separate FIRST-LEVEL Instruction and Data Caches
    a. Tag and Data Arrays for each
    b. N-way Set Associative (differing Ns based on optimal calculation)
    c. Second- and Third-Level caches much larger, hold both data and instructions
2. Address split into: [Tag, Index, Offset]
3. Instruction Cache:
    a. Blocking
    b. Single-Ported
    c. N-way Set Associative (calculate later)
    d. Pipelined
        i. Parallel TLB, tag and data array access. We want to get the instruction as fast as possible.
    e. Static Ordering (cache stores instructions in the conventional order)
    f. CONSIDERATION: If doing Superscalar, then we need to fetch MULTIPLE instructions in a single cycle (For example, by reading consecutive bytes from the cache that are part of the same cache block)
4. Data Cache:
    a. Non-Blocking
        i. MSHR table and LOAD/STORE table entries (extra hardware)
    b. Single-Ported
    c. N-way Set Associative (calculate later)
    d. Pipelined
        i. SERIAL tag array, and then data array accesses (The OOO processor will hide the latency of serial accesses)
5. Replacement Policy
    a. Pseudo-LRU Cache (pyramid bit implementation)

Pipeline Specifications

- **Fetch Unit:**



- Will be further pipelined in order to allow for an instruction to be fetched every cycle
- **STAGES:** Next Address Prediction, Next Address Calculation, Instruction Cache Access, Drive to Decode Stage

- **Decode Unit**
  - For RISC, decode is trivial
  - TODO: Will need to discuss having a fetch buffer and multiple decoders

- **Rename Unit (Allocation Stage)**
  - Reorder Buffer
  - Register Rename Table (RAT)
  - Architectural Register File
  - Read BEFORE Issuing

- **Issue Unit**
  - Load Ordering and Store Ordering for memory disambiguation
  - Reservation Stations for non-memory instructions
  - Arbiter separating between Memory and Non-memory operations

    - **Separate Load and Store queue for Memory operations. Sequential Loads, Sequential Stores, but Loads and Stores themselves may be Out of Order.**
    - Functional Units for Simple ALU, Multiplication Units, Division Units, Branch Calculation Unit, and CSR Calculation Unit.

- **Execution Unit**
    - Multiple Address Generation Unit for the Separate Load/Store pipelines
    - Multiple ALU Units
    - Multiplication and Division Units (not as common)
    - Branch Unit
    - Bypass Network Necessary (more of a datapath question ATP)

- **Commit Unit / Other Considerations**
    - ROB with Retire Register File (RRF)
    - When Branch Misprediction occurs, allow all previous instructions and the branch instruction in question to be committed before flushing in order to ensure that the architectural state in the RRF represents the application state after execution of the mispredicted branch. Then we restore the rename table by pointing all of its entries to the RRF, thus beginning the correct renaming of instructions.
    - **TODO: HOW WILL WE RESTORE OUR SPECULATIVE STATE?**

Other Specifications

Virtual Memory Hardware Requirements
- Virtually Indexed, Physically Tagged Cache that is accessed in parallel to the physical instruction cache.
- TODO: Page Size? Multi-Level Page Table?

Input/Output Ports
- Keyboard, Mouse, HDMI Display

Interrupt/Exception Handling
- Exceptions are handled at commit time of the criminal instruction, then all in-flight instructions are flushed, and then the speculative state is recovered.
- Interrupts also wait for current instruction to be completed before processing the desired interrupt.