1. Logo, company name

   Description automatically generatedNational University of Sciences and Technology
2. School of Electrical Engineering and Computer Science
3. Department of Electrical Engineering

|  |  |
| --- | --- |
| **EE-321: Computer Architecture and Organization** | |
| **Faculty Member** | **Semester** |
| Dr. Muhammad Imran | 7th |
| **Class/Section** | **Date** |
| CAO | Dec 28, 2024 |

1. **Project: AES Integration with RV32I**

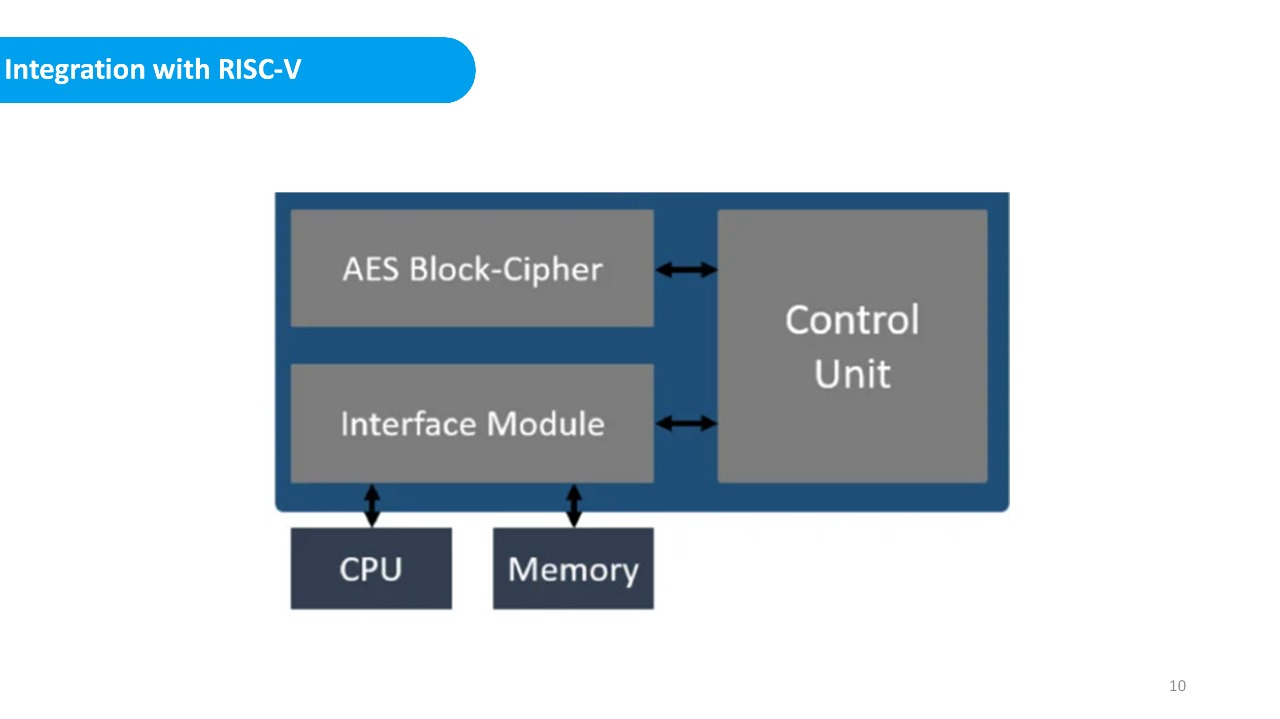
|  |  |
| --- | --- |
| **Name** | **Registration No.** |
| Muhammad Majid Ferozi | 386488 |
| Hassaan Ahmad | 369212 |
| Muhammad Saad Jawad | 372985 |

**Report on RISC-V RV32I ISA**

**Introduction**

RISC-V (Reduced Instruction Set Computing - Version 5) is an open-standard instruction set architecture (ISA) that is modular and extensible. Among its subsets, RV32I is the base integer instruction set for 32-bit implementations. It provides a foundation for both embedded systems and general-purpose computing.

RV32I defines 32 registers, each 32 bits wide, where the register x0 is hardwired to 0. This report covers the instruction set, pipelining, hazard detection and handling mechanisms, as well as branch prediction techniques in RISC-V RV32I.



**Basic Instruction Set**

The RV32I instruction set includes the following major categories:

**1. Arithmetic Instructions**

* **ADD**: Performs integer addition.
  + Syntax: ADD rd, rs1, rs2
  + Function: rd = rs1 + rs2
* **SUB**: Performs integer subtraction.
  + Syntax: SUB rd, rs1, rs2
  + Function: rd = rs1 - rs2
* **MUL** (if M-extension is included): Multiplies integers.
  + Syntax: MUL rd, rs1, rs2
  + Function: rd = rs1 \* rs2

**2. Logical Instructions**

* **AND**: Performs bitwise AND.
  + Syntax: AND rd, rs1, rs2
  + Function: rd = rs1 & rs2
* **OR**: Performs bitwise OR.
  + Syntax: OR rd, rs1, rs2
  + Function: rd = rs1 | rs2
* **XOR**: Performs bitwise XOR.
  + Syntax: XOR rd, rs1, rs2
  + Function: rd = rs1 ^ rs2

**3. Shift Instructions**

* **SLL**: Shifts left logical.
  + Syntax: SLL rd, rs1, rs2
  + Function: rd = rs1 << rs2
* **SRL**: Shifts right logical.
  + Syntax: SRL rd, rs1, rs2
  + Function: rd = rs1 >> rs2
* **SRA**: Shifts right arithmetic.
  + Syntax: SRA rd, rs1, rs2
  + Function: rd = rs1 >> rs2 (arithmetic)

**4. Load and Store Instructions**

* **LB**: Load byte.
  + Syntax: LB rd, offset(rs1)
  + Function: rd = sign\_extend(M[rs1 + offset])
* **SB**: Store byte.
  + Syntax: SB rs2, offset(rs1)
  + Function: M[rs1 + offset] = rs2

**5. Branch Instructions**

* **BEQ**: Branch if equal.
  + Syntax: BEQ rs1, rs2, offset
  + Function: if (rs1 == rs2) PC += offset
* **BNE**: Branch if not equal.
  + Syntax: BNE rs1, rs2, offset
  + Function: if (rs1 != rs2) PC += offset

**6. Immediate Instructions**

* **ADDI**: Adds immediate value.
  + Syntax: ADDI rd, rs1, imm
  + Function: rd = rs1 + imm
* **ANDI/ORI/XORI**: Perform bitwise operations with immediate values.

**Pipelining in RISC-V RV32I**

RISC-V employs a standard five-stage pipeline:

1. **Instruction Fetch (IF)**: Fetches the instruction from memory.
2. **Instruction Decode (ID)**: Decodes the instruction and reads registers.
3. **Execute (EX)**: Performs arithmetic/logic operations or calculates memory addresses.
4. **Memory Access (MEM)**: Reads from or writes to memory.
5. **Write Back (WB)**: Writes results back to the register file.

**Advantages of Pipelining**

* Increases instruction throughput.
* Reduces CPU idle time by overlapping stages.

**Challenges**

* Hazards, which include:
  + **Structural Hazards**: Caused by resource conflicts.
  + **Data Hazards**: Occur when instructions depend on results of previous instructions.
  + **Control Hazards**: Result from branch instructions.

**Hazard Detection and Handling**

**1. Structural Hazard Handling**

* **Solution**: Employ multiple memory units or pipelines to ensure resources are available for all stages.

**2. Data Hazard Handling**

* **Forwarding/Bypassing: Forward results directly from one pipeline stage to another without waiting for WB.**
* **Stalling: Introduce no-operation (NOP) instructions to resolve hazards temporarily.**

**Forwarding Unit**

**The forwarding unit is a hardware component used to resolve data hazards by redirecting data from later pipeline stages back to earlier ones. For instance, if an instruction in the EX stage requires the result of a prior instruction that is not yet written back to the register file, the forwarding unit bypasses the data directly from the MEM or WB stage to the EX stage.**

**Implementation**

1. **The forwarding unit monitors the source registers of the current instruction in the EX stage.**
2. **It compares these registers with the destination registers of instructions in later stages (MEM or WB).**
3. **If a match is found, the required data is forwarded to the EX stage inputs.**

**Example**

**Consider two instructions:**

1. **ADD x3, x1, x2**
2. **SUB x4, x3, x5**

**Without forwarding, the second instruction would stall until the ADD instruction completes WB. With forwarding, the result of ADD is directly sent from MEM/WB to the EX stage of SUB.**

A diagram of a system

Description automatically generated

**3. Control Hazard Handling**

* **Branch Delay Slots**: Insert independent instructions in delay slots following branch instructions.
* **Pipeline Flushing**: Flush incorrect instructions after a branch prediction fails.

**Branch Prediction**

Branch prediction minimizes pipeline stalls due to control hazards.

**1. Static Branch Prediction**

* **Fixed Prediction**: Always assumes a branch is taken or not taken.
* **Backward Branch Bias**: Predicts branches backward in loops as taken.

**2. Dynamic Branch Prediction**

* **One-Bit Predictor**: Maintains a single bit for each branch indicating whether it was taken or not during the last execution.
* **Two-Bit Predictor**: Uses two bits to reduce misprediction penalties by requiring two incorrect predictions to change the state.
* **Branch Target Buffer (BTB)**: Caches predicted target addresses.

**Example: Two-Bit Predictor State Diagram**

|  |  |  |  |
| --- | --- | --- | --- |
| **Current State** | **Prediction** | **Next State if Taken** | **Next State if Not Taken** |
| Strongly Taken | Taken | Strongly Taken | Weakly Taken |
| Weakly Taken | Taken | Strongly Taken | Weakly Not Taken |
| Weakly Not Taken | Not Taken | Weakly Taken | Strongly Not Taken |
| Strongly Not Taken | Not Taken | Weakly Not Taken | Strongly Not Taken |
|  |  |  |  |

A diagram of a machine

Description automatically generated

**Conclusion**

The RISC-V RV32I ISA provides a robust and modular framework for 32-bit architectures. Its simplicity and extensibility allow for efficient pipelining and hazard management. While static branch prediction suffices for simpler designs, dynamic techniques enhance performance in complex systems. With its open standard, RISC-V continues to empower developers to innovate across diverse computing domains.