DESIGN OF A 5-STAGE PIPELINED RISC-V PROCESSOR

Abinaya G[1], Asst. Prof. Dr S. Lakshmi[2]
[1] Dept of VLSI Design
[2] Guide, Department of VLSI Design
[1,2] Thirumalai Engineering College, Kanchipuram, Tamil Nadu, India

Abstract- *This paper presents a modified design of RISC-V RV32I 32-bit microprocessor. The RISC-V processor consists of a flexible 5 stage pipelined processor with certain techniques for hazard management. The processor consists of 6 blocks- fetch block, decode block, control logic block, memory block, register block and the ALU block. The processor is then pipelined into 5 stages – the fetch stage, the decode stage, the execute stage, the memory stage and the write backstage. After pipelining, the pipeline hazards such as data hazards and control hazards are managed by using data forwarding from previous stages of the pipeline and by introducing delay slots for control transfer instructions. After that, the schematic for the processor is obtained by feeding the Verilog code in cadence software. Also, there is a future scope of implementing new instructions that can combine the functions of two separate instructions into one single instruction, which are considered extensions to the existing ISA of RV32I.*

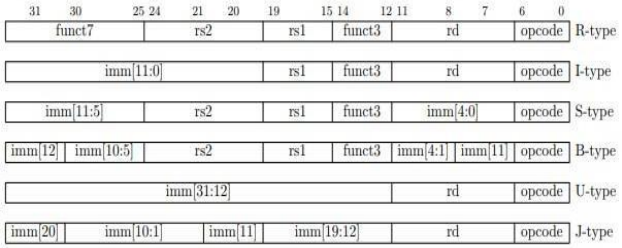
Keywords: RV32I, Flexibility, 5 stage pipelined, hazard management, 5 clock cycles.

I. INTRODUCTION

Reduced Instruction Set Computers (RISC) are meant to use a reduced instruction set in order to have a reduced execution time. They can carry out minor tasks in response to commands. These processors are more efficient at completing commands.

The format used in this project is RV32I which is v2.0 of RISC V. It is an optimized ISA for creating RISC machines. This ISA can support almost all modern operations and features. It has 32 general purpose registers x0 to x3. x0 is hardwired to the constant 0. There is one more register with a specific purpose, called the Program Counter or PC. The PC is responsible for fetching the next instruction into the processor from the memory.

The Instruction Set Format is shown in Figure 1.



II. INSTRUCTION SET ARCHITECTURE OR ISA

The RISC-V ISA consists of a total of 47 instructions, from which we have implemented 20 instructions. The Instruction Set consists of 6 major instruction types: r-type, s-type, i-type, u-type, b-type and j-type. Register type instructions perform operations between two source registers rs1 and rs2, and the result is stored in a destination register rd. Immediate type instructions perform operations between rs1 and an immediate value and store the result in rd. Store type instructions store data inside the memory instead of a destination register. Upper immediate type instructions store a 32-bit upper immediate value directly into rd. Jump and Branch instructions are control transfer instructions which change the next PC value.

Architecture of RISC-V Our proposed RISC-V 32-bit architecture involves 6 blocks that are interconnected and communicate with each other. These blocks are:

- Instruction fetch Block
- Instruction Decode Block
- ALU Block
- Memory Block
- Register Block
- Control Logic Block

Instruction fetch block: The fetch block is responsible for retrieving instructions from the memory based on the current program counter (PC) value. It reads the instruction stored at the memory location pointed to by the PC and loads

it into a buffer for further processing. It sends an RD signal to the memory and when the signal is acknowledged by the memory, the memory sends the instruction to the decode block, where it is decoded. Also, the logic for the next PC is present there.

Instruction Decode Block: The decode block receives the instruction fetched from memory and extracts essential fields such as opcode, source register identifiers, destination register identifiers, funct3 and funct7, immediate values, and operation types. The 7-bit opcode corresponds to instruction type, the funct 3 along with funct 7 represents the operation to be performed. The 5-bit rd represents the address location of the destination register, the 5-bit rs1 and rs2 represents the address location of the source registers, the 11-bit immediate value is sign extended to 32-bit immediate value, and the same goes for the 7-bit offset 1 and the 5-bit offset 2, used in store type instructions. The 22-bit upper immediate value is also sign extended to 32 bit and used in u type instructions.

ALU Block: The ALU processes data from registers or memory based on the instruction's requirements. For example, it can perform operations like adding two register values (add), bitwise AND between two operands (and), or compare two values (compare). The data is stored in rd, for r type, I type, u type and j type instructions, whereas it is stored in memory for s type instructions.

Memory Block: The memory block stores both instructions and data required by the processor. Instructions are fetched from memory during the instruction fetch stage, when RD is active. Data (such as variables, arrays, and stack contents) are accessed during load/store operations executed.

Register Block: Registers are small, high speed storage locations directly accessible by the processor's arithmetic and logic units (ALUs). They store operands for arithmetic and other data needed during instruction execution. Registers are typically organized as a register file, where each register has a unique identifier (register number) and can hold a fixed number of bits (e.g., 32 bits in a 32-bit RISC-V architecture). The registers rs1, rs2 and rd are present in the register file only. The first register x0 always contains a 32-bit 0s as data. There are 32 registers x0 to x31.

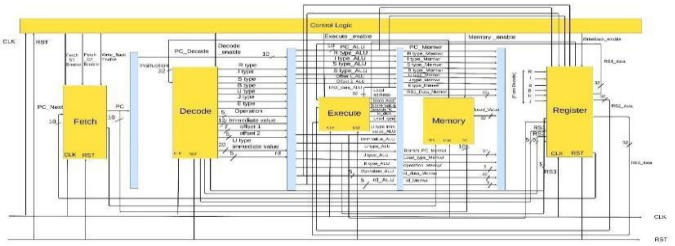
Control Logic Block: the control logic block is the central controller of a RISC-V processor, orchestrating the operation of pipeline stages, generating control signals, managing hazards, and optimizing performance. Its effective design and implementation are essential for achieving efficient and correct execution of instructions within the processor pipeline. The Control Logic Block provides Enable (EN) signals to all the blocks, thus sequentially enabling the blocks in a non-pipelined architecture. But in a pipelined architecture, all the blocks are always enabled at all the times, for pipeline to occur.

III. PIPELINING OF THE PROCESSOR

Pipeline ensures that all operations within the processor components are synchronized to the clock (clk) using positive edge triggers. This ensures that the pipeline stages progress synchronously, aligning with the processor's clock cycle. In order to efficiently pipeline the processor, we need to register the signals that V. Simulation Results are needed by the subsequent blocks, like the Program Counter. By registering the signals and forwarding it to subsequent blocks, we can bring in the next data by storing the intermediate value in the register. During pipeline, we may experience the following hazards:

- Data hazards
- Control hazards
- Structural hazards

To detect and resolve data hazards by implementing data forwarding logic, i.e., when an instruction requires data that is being produced by a previous instruction in the pipeline, forward the data directly from the execution stage to the dependent stage. If the data has been changed by the previous instruction, then that data has to replace the current source register data, if the same register that has been used as a destination register in the previous instruction is called upon as the source register in the in the current instruction.



Block Diagram of the Proposed Processor is shown in Figure 2.

To address control hazards (e.g., branches), we can predict branch outcomes early in the pipeline or using branch prediction mechanisms to minimize stalls. Implement techniques like branch target prediction to predict the target address of branch instructions before they are fully decoded, allowing subsequent instructions to proceed without waiting for branch resolution. But this is very complex. Instead, the delay slots occurring in the pipeline is a necessary price that most processors pay for simplicity in hardware and functionality. Delay slots occur due to the PC incrementing even before the instruction has been decoded as a branch type

or jump type by the processor. Due to this, three clock cycles are being wasted. Structural Hazards: It is common to avoid structural hazards by ensuring that resources (e.g., ALU, memory units) are efficiently shared among pipeline stages. Use techniques like resource duplication or scheduling to allocate resources dynamically based on pipeline stage requirements, reducing contention and hazards.

IV. SIMULATION RESULTS

Our Proposed work contains 20 instructions out of the 47 instructions from the RISC-V RV32I architecture. The mnemonics of the instructions have been attached below, along with the simulation results.

ADD x1, x10, x6 SUB x2, x10, x6 MUL x3, x10, x6 DIV x4, x10, x6 XOR x5, x10, x6 OR x7, x10, x6 AND x8, x10, x6 ADDI x15, x10, 14

XORI x16, x10, 14 ORI x17, x10, 14 ANDI x18, x10, 14 LW x19, 14(x10) SW x6, x10

LUI x20, 524287

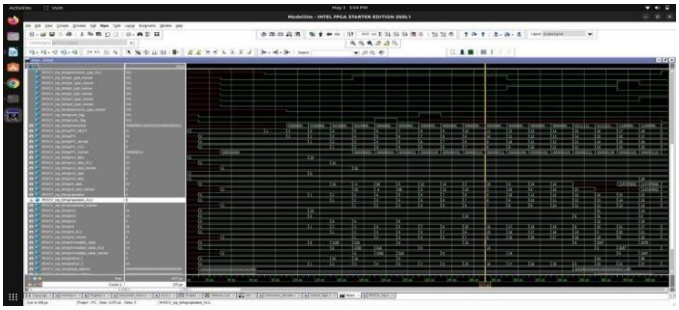
AUIPC x21, 524287

JAL x22, 4

JALR x23, 12(x10) BEQ x0, x0, 4 BNE x1, x0, 2 BLT x0, x1, 2 BGE x1, x0, 2

V. CONCLUSION AND FUTURE WORK

Based on RV32I, the design and verification of a 5-stage pipelined RISC processor, incorporating 20 instructions across various types and considering future enhancements such as branch prediction and data forwarding, represents a comprehensive approach to processor development. Through meticulous simulation and validation, including the utilization of tools like ModelSim and Cadence, critical functionalities have been verified and potential performance bottlenecks identified. The outlined future enhancements, ranging from branch prediction schemes to synthesis optimization in tools like Xilinx Vivado, underscore a commitment to improving execution speed, throughput, and overall efficiency. Ultimately, this work sets a solid foundation for the continued evolution of RISC processors, aligning with the demands of emerging technologies and computing paradigms. The Simulation Results are shown in Figure 3.



REFERENCES

[1] M. N. Topiwala and N. Saraswathi, “Implementation of a 32-bit MIPS Based RISC International Conference on Advanced Communications, Control and Computing Technologies,2014, pp. 979-983.

[2] S. P. Ritpurkar, M. N. Thakare and G. D. Korde, “Synthesis and Simulation of a 32Bit MIPS RISC Processor using VHDL” 2014 International Conference on Advances in Engineering & Technology Research (ICAETR - 2014), 2014, pp. 1-6.

[3] S. Palekar and N. Narkhede, “32-bit RISC Processor with Floating Point Unit for DSP Applications”,2016 IEEE International Conference on Recent Trendsin Electronics, Information & Communication Technology (RTEICT), 2016, pp. 2062-2066.

[4] A Raveendran, V. B. Patil, D. Selvakumar and V. Desalphine, “A RISCV Instruction Set Processor-Micro-architecture Design and Analysis”,2016 International Conference on VLSI Systems, Architectures, Technology and Applications (VLSI-SATA), 2016, pp. 1-7.

[5] R. J. L. Austria, A. L. Sambile, K. M. Villegas and J. N. T. Tabing, “Design of an 8 Bit Five Stage Pipelined RISC Microprocessor for Sensor Platform Application”, TENCON 2017 - 2017 IEEE Region 10 Conference, 2017, pp. 2110- 2115