Indian Institute of Technology Mandi

Single-cycle RISC-V processor for solving Linear system of equations

Group 3

Name	Enrollment no.	Branch
Chinmay Patel	B23487	VLSI
Garvit Garg	B23488	VLSI
Rushabh Lodha	B23495	VLSI
Jigeesha Sur	B22212	Electrical
Vani Dhiman	B23505	VLSI
Talla Bhavana	B23471	VLSI
Parv A. Joshi	B23090	VLSI

The Problem Statement

Design your own custom single-cycle RISC-V processor using Verilog-HDL by choosing instructions based on your own consideration to achieve a **Linear system of equation solver (at-least 3 variables).**

Overview

The implementation follows the RV32I base integer instruction set and completes all stages (fetch, decode, execute, memory, write-back) in a single clock cycle. The processor includes:

1. Program Counter (PC)
2. Instruction Memory
3. Instruction Decoder
4. 32 General-purpose registers(32 × 32-bit)
5. ALU (Arithmetic Logic Unit)
6. Data Memory
7. Control Unit
8. MUXes and ALU Control

Steps to achieve the solution:

1. Begin by writing C code: To solve a system of 3 linear equations with 3 variables, represented in augmented matrix form (3×4). It uses Gaussian elimination with partial pivoting and avoids floating-point operations by:

- Multiplying via repeated addition

- Dividing via repeated subtraction

Matrix initialization as:

```
A = {
  {1, 2, 3, 4},
  {2, 3, 4, 5},
  {3, 4, 5, 6}
}
```

Represented as:

$1x + 2y + 3z = 4$

$2x + 3y + 4z = 5$

$3x + 4y + 5z = 6$

Why It’s Easy to run for our process

- No floats
- Fixed-size arrays
- Controlled branching

2. Cross compilation to assembly

We used the RISC-V GCC toolchain to convert the C code into assembly code which will generate the RISC-V assembly version of your C code.

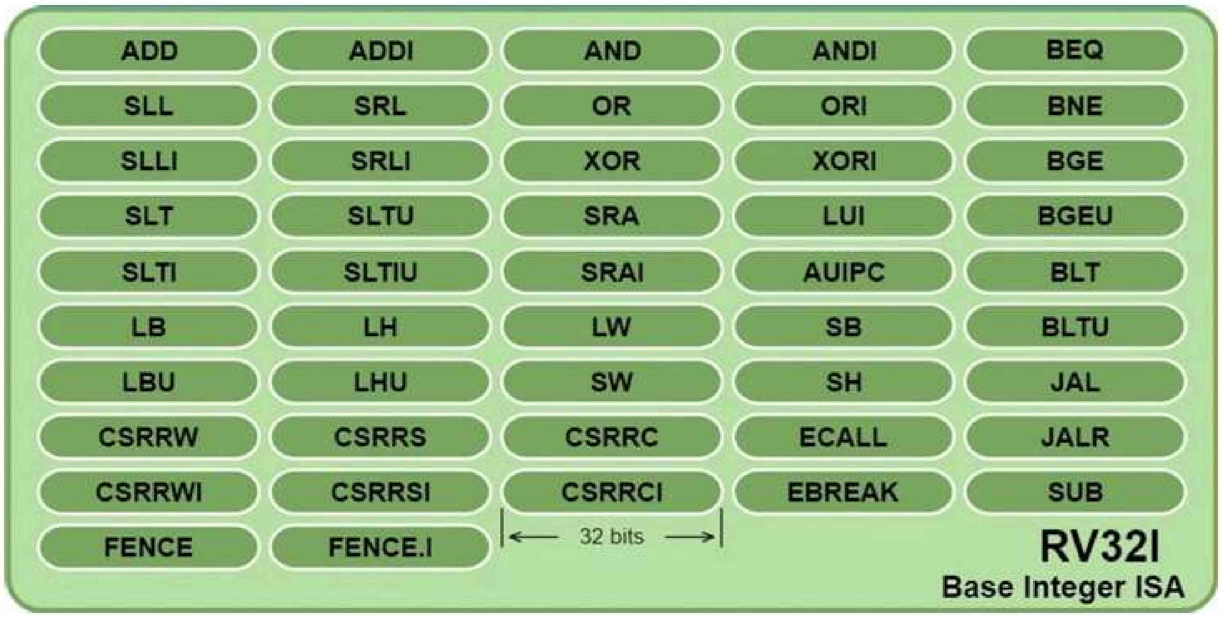
3. Replacing all pseudo instructions with their actual RISC-V instructions.

The main pseudo instructions in the code are

Pseudo	**Replaced With**

li (load immediate)	lui and addi
mv (move)	addi with zero
ret (return)	jalr with zero

<u>Required Base Instructions</u>



ADD	ADDI	AND	ANDI	BEQ
SLL	SRL	OR	ORI	BNE
SLLI	SRLI	XOR	XORI	BGE
SLT	SLTU	SRA	LUI	BGEU
SLTI	SLTIU	SRAI	AUIPC	BLT
LB	LH	LW	SB	BLTU
LBU	LHU	SW	SH	JAL
CSRRW	CSRRS	CSRRC	ECALL	JALR
CSRRWI	CSRRSI	CSRRCI	EBREAK	SUB
FENCE	FENCE.I			

← 32 bits →

RV32I
Base Integer ISA

Arithmetic and Logic

Instruction	Purpose
add	Basic addition(used in loops and array calculations)
sub	Subtraction (used in division and elimination)
addi	Add immediate (loop counters, constants)

lui	Load upper immediate (for li replacement)
slli	Shift left
jal	Jump And Link
beq	Branch if equal to

Memory Access

Instruction	Purpose
lw	Load Word (read array elements like A[i][j])
sw	Store word (write back updated matrix values)

4. Testing

We tested the processor using individual instructions. The output is stored in the registers or data memory printed to the console upon simulation end. To run the simulation:

1. Load files in Vivado
2. Compile all modules
3. Run simulation and observe output

Acknowledgement and Contributions

We sincerely acknowledge our course instructor, Dr Bikram Paul, for his guidance. The team's collective efforts have made this project a success. Below mentioned are the contributions:

- Rushabh Lodha: GNU compilation, algorithm simplification
- Parv A. Joshi: Data Memory Integration, Verification, Debugging
- Chinmay Patel: Instruction memory, Control Unit design, Testing, debugging
- Garvit Garg: Data memory setup, PC and Immediate, TB development (also got TCL console running)
- Jigeesha Sur: Algorithm design, Documentation and report
- Vani Dhiman: Main and ALU decoder, Instruction verification
- Talla Bhavana: Main and ALU decoder, Instruction verification