The RISC-V 32I project aims to implement a 32-bit RISC-V processor that supports the base integer instruction set (I), as specified by the RISC-V Foundation. The processor design follows the principles of reduced instruction set computing (RISC) and incorporates several key features, such as a load/store architecture, a fixed instruction size, and a large register file.

The project involves the development of a complete processor pipeline, including an instruction fetch unit, an instruction decode unit, an execution unit, a memory access unit, and a write-back unit. The pipeline is designed to support pipelining and employed with data forwarding mechanism.

The processor is implemented in Verilog RTL flow, which enables designers to create a detailed hardware description of the processor's behaviour. The Verilog RTL implementation includes a simulation environment, which allows developers to verify the correctness of the design and test its functionality.

Contents

Chapter 1

Introduction

There has been drastic changes in chip technology and implementations in a single chip are widespread now. With these, the significance of Instruction Set Architectures (ISAs) has increased rapidly. Reduced Instruction Set RISC based ARM architectures already dominated the mobile systems like Android and Apple. RISC-V is a open ISA based on RISC.

In this work, we represent a hardware design architecture for RV32I base integer system for 32-bit addresses. RV32I is a form of RISC-V that was designed to be adequate for generating compiler targets and supporting modern system environments.

Chapter 2

RISC-V Architecture

The design of the RISCV architecture is done using the Register Transfer Level (RTL). In RTL code, the circuit is defined using operations and transfer of data between different registers. This code can then be synthesized to generate a schematic which produces a graphical representation of the modules used and the connections in-between them.

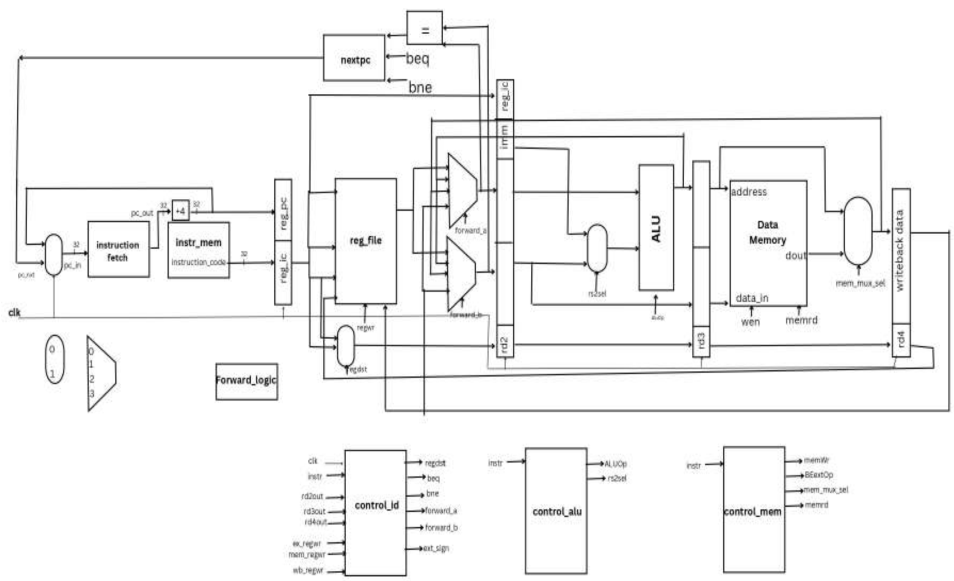

Figure 2.1: Block Diagram of RISC-V 32I Architecture

Stages Of Pipeline:

1. IF: Instruction Fetch Unit
2. ID: Instruction Decode Unit
3. EX: Execution Unit
4. MEM: Memory Unit
5. WB: Write Back Unit

2.1 Stage1: IF (Instruction Fetch)

This unit consists of three modules:
1. 2:1 Multiplexer
2. Instruction Fetch
3. Instruction Memory

2.1.1 2:1 Multiplexer

MUX selects the next PC i.e, PC+4 or the PC location from branch instructions based on the select line pcsrc generated from the control_id unit

2.1.2 Instruction Fetch

It is nothing but a register. At positive edge of the clock, the address which is pointing to the instruction memory gets updated.

2.1.3 Instruction Memory

The instructions are stored in the Instruction memory. Each instruction is of 32-bit. Those instructions are from standard RV32I Base Instruction Set[1]

2.2 IF - ID Pipeline Registers

Two registers are included. One is for instruction code to decode type of instruction, immediate address, ALU function source and destination registers and other is for PC location to calculate next address of Branch instructions

2.3 Stage2: ID (Instruction Decode)

This unit consists of seven modules:
1. Decode stage control unit
2. Register file
3. 2 4:1 Multiplexer
4. Sign Extension
5. Comparator
6. NextPC
7. 2:1 Multiplexer

2.3.1 Decode stage control unit

This unit generates control signal for the modules in this unit like regdst, forward_a, forward_b etc

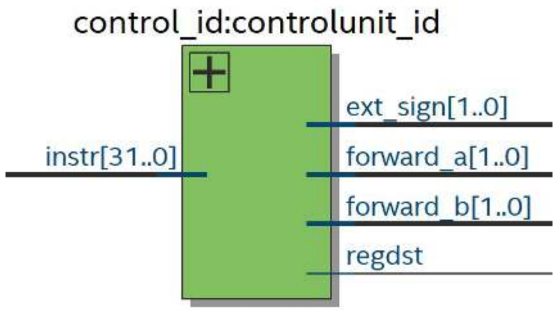

Figure 2.2: Control decode block diagram

2.3.2 Register file

If fetches the register contents for the given source registers//

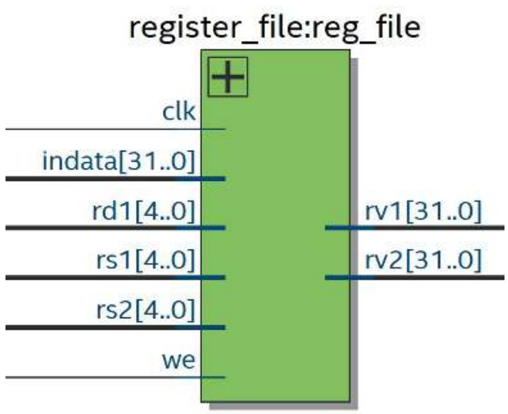

Figure 2.3: Regsiter file block diagram

2.3.3 2 4:1 MUX

Two MUXes are used to select which stage ouput as taken as inputs that are required for ALU

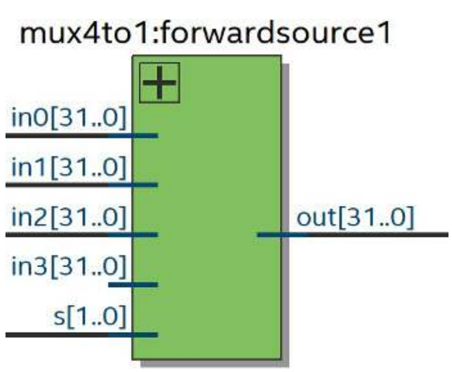

Figure 2.4: Forward source 1 block diagram

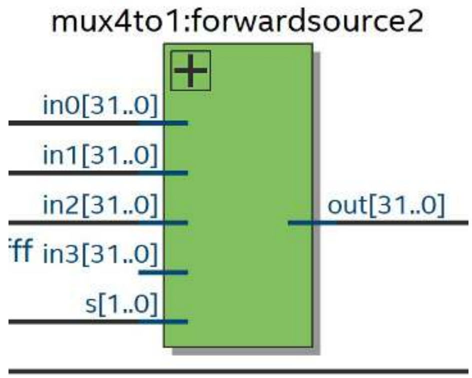

Figure 2.5: Forward source 2 block diagram

2.3.4 Sign Extension

Sign extension is needed to extend the immediate address

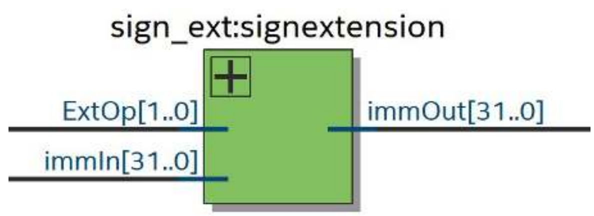

Figure 2.6: sign extension block diagram

2.3.5 Comparator

This unit checks if both the registers are having same content and generates zero flag if equal

2.3.6 NextPC

Based on the control signals beq,bne generated from control_id module its get to know the type of instruction and then it calculates the next PC address relative to its base address

2.3.7 2:1 Multiplexer

This MUX is used to select the destination register from Rt and Rd

2.4 ID - EX Pipeline Registers

Five registers are included for immediate address, two inputs of ALU, destination register, instruction code

2.5 Stage3: EX (Execution)

This unit consists of 3 module:
1. 2:1 Multiplexer
2. ALU (Arithmetic Logic Unit)
3. Control Unit

2.5.1 2:1 Mux

This unit is used to select the second source to the input depending on the type of input.

2.5.2 ALU

This unit does all the Arithmetic Operations need be done.

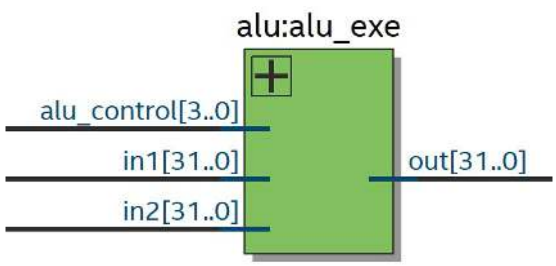

Figure 2.7: ALU block diagram

The operations included in the ALU unit are:
1. Addition
2. Subtraction
3. Bit wise AND, OR, XOR
4. Shifting Operations

2.5.3 Execution stage Control unit

This unit generates the control signals required for ALU unit.Such as

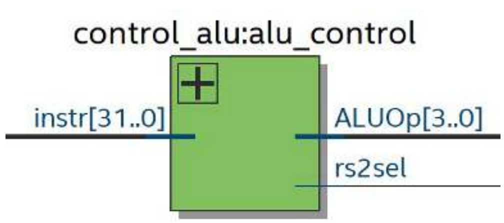

Figure 2.8: Control unit ALU block diagram

1. Selection of operation (ALUOp)
2. Selection of source (rs2sel)

2.6 EX - MEM Pipeline Registers

Four registers are included for Destination register, data to stored in memory,output of Execution Unit, Instruction Code

2.7 Stage4: MEM (Memory)

This unit consists of four modules:
1. Memory stage control unit
2. Byte select
3. Data Memory
4. 2:1 Multiplexer

2.7.1 Memory stage control unit

This unit generates the contol signals required in the memory stage.Such as

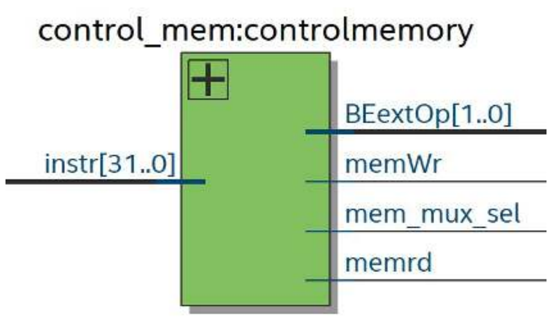

Figure 2.9: Control memory block diagram

1. memWr - when to write into a data memory
2. BEextOp - tell about which type store instruction it is
3. mem_mux_sel - selection signal used to select the signal between alu output or memory output (dout).
4. memrd - it is read enable for memory.

2.7.2 Byte select

This unit generates a signal which is used in selecting which part of the 32 bit data (byte or half word or word) has to be written based on the type of Store instructions.

2.7.3 Data Memory

It is the data memory where the data is stored and used when required.

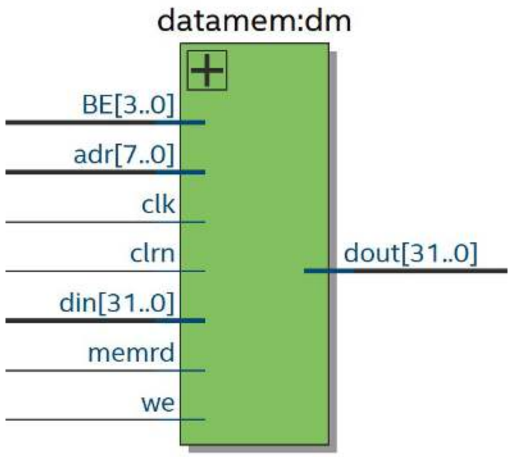

Figure 2.10: Data memory block diagram

The size of the memory can be varied as per requirement.
Depending upon the control signals either it read or write the data.
Byte selecting is possible with this memory type.

2.7.4 2:1 Multiplexer

This mux is used to select either the output of ALU or memory output (dout).

2.8 MEM - WB Pipeline Registers

Four registers are included for Destination register, data from the memory unit that is needed to write back, output of Execution Unit, Instruction Code

2.9 Stage5: WB (Write Back)

This unit consists of three modules:

1. Write Back stage control unit
2. 2:1 Multiplexer
3. Write Back Extension

2.9.1 Write Back stage control unit

This unit generates the control signal required for Write Back stage. Such as

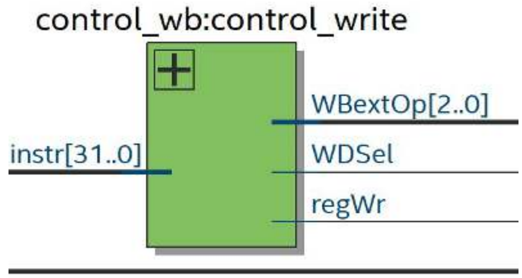

Figure 2.11: Control WriteBack block diagram

1. WDSel - used to select the destination address.
2. regWr - this acts as register file write enable signal.
3. WBextOp - this 3 bit signal is used to select whic byte or half word or word to write in the destination.

2.9.2 Write Back Extension

This unit takes the data to be written into register file as input and gives the data out based on the type of instructions (i.e. whether to write byte or half word or word).

2.9.3 2:1 Multiplexer

Mux selects which data to be written. i.e. the data which we got from ALU or from the WB extension stage.

Chapter 3

Instruction Set Architecture and Instructions Used

All the instructions are a fixed 32 bits in length and must be aligned on a four-byte boundary in memory.
The different type of instructions that the processor can handle are
1. Register to Register
2. Register to Immediate
3. Load and Store
4. Branch

3.1 Register to Register Instructions

Funct7(7)	Rs2(5)	Rs1(5)	Funct3(3)	Rd(5)	Opcode(7)

0000000	rs2	rs1	000	rd	0110011	ADD
0100000	rs2	rs1	000	rd	0110011	SUB
0000000	rs2	rs1	001	rd	0110011	SLL
0000000	rs2	rs1	010	rd	0110011	SLT
0000000	rs2	rs1	011	rd	0110011	SLTU
0000000	rs2	rs1	100	rd	0110011	XOR
0000000	rs2	rs1	101	rd	0110011	SRL
0100000	rs2	rs1	101	rd	0110011	SRA
0000000	rs2	rs1	110	rd	0110011	OR
0000000	rs2	rs1	111	rd	0110011	AND

Figure 3.1: Register to Register Instructions

3.2 Register to Immediate Instructions

Funct7(7)	Rs2(5)	Rs1(5)	Funct3(3)	Rd(5)	Opcode(7)

imm[11:0]		rs1	100	rd	0010011	XORI
imm[11:0]		rs1	110	rd	0010011	ORI
imm[11:0]		rs1	111	rd	0010011	ANDI
0000000	shamt	rs1	001	rd	0010011	SLLI
0000000	shamt	rs1	101	rd	0010011	SRLI
0100000	shamt	rs1	101	rd	0010011	SRAI

Figure 3.2: Register to Immediate type Instructions

3.3 Load Instructions

Immediate(12)	Rs1(5)	Funct3(3)	Rd(5)	Opcode(7)

imm[11:0]	rs1	000	rd	0000011	LB
imm[11:0]	rs1	001	rd	0000011	LH
imm[11:0]	rs1	010	rd	0000011	LW
imm[11:0]	rs1	100	rd	0000011	LBU
imm[11:0]	rs1	101	rd	0000011	LHU

Figure 3.3: Load Instructions

3.4 Store Instructions

Immediate(7)	Rs2(5)	Rs1(5)	Funct3(3)	Immediate(5)	Opcode(7)

imm[11:5]	rs2	rs1	000	imm[4:0]	0100011	SB
imm[11:5]	rs2	rs1	001	imm[4:0]	0100011	SH
imm[11:5]	rs2	rs1	010	imm[4:0]	0100011	SW

Figure 3.4: Store Instructions

3.5 Branch Instructions

Immediate(7)	Rs2(5)	Rs1(5)	Funct3(3)	Immediate(5)	Opcode(7)

imm[12|10:5]	rs2	rs1	000	imm[4:1|11]	1100011	BEQ
imm[12|10:5]	rs2	rs1	001	imm[4:1|11]	1100011	BNE
imm[12|10:5]	rs2	rs1	100	imm[4:1|11]	1100011	BLT
imm[12|10:5]	rs2	rs1	101	imm[4:1|11]	1100011	BGE
imm[12|10:5]	rs2	rs1	110	imm[4:1|11]	1100011	BLTU
imm[12|10:5]	rs2	rs1	111	imm[4:1|11]	1100011	BGEU

Figure 3.5: Branch Instructions