Indian Institute of Technology Bombay

EE309, 2024

Report On

Pipelined IITB RISC CPU

Course Instructor: Prof. Virendra Singh

Group Members:

Prajwal Nayak	22B4246
Sarvadnya Nandkumar Purkar	22b4232
Shikhar Moondra	22B0688
Siddick Mohammed Hamza Khatri	22b4241

Abstract

This project focuses on the detailed design and implementation of a

sophisticated 6-stage pipelined processor, the IITB-RISC-23, tailored specifically for educational purposes within the context of computer architecture and digital systems. The processor is based on a 16-bit instruction set architecture (ISA), featuring 8 general-purpose registers (R0 to R7) where R0 is reserved as the Program Counter.

Contents

R Type Instruction format

Opcode (4 bit)	Register A (RA) (3 bit)	Register B (RB) (3-bit)	Register C (RC) (3-bit)	Comple -ment (1 bit)	Condition (CZ) (2 bit)

I Type Instruction format

Opcode (4 bit)	Register A (RA) (3 bit)	Register C (RC) (3-bit)	Immediate (6 bits signed)

J Type Instruction format

Opcode (4 bit)	Register A (RA) (3 bit)	Immediate (9 bits signed)

1 Introduction

IITB-RISC is a 16-bit very simple computer developed for teaching that is based on the Little Computer Architecture. The IITB-RISC-23 is an 8-register, 16-bit computer system. It has 8 general-purpose registers (R0 to R7). Register R0 always stores the a Program Counter. All addresses are byte addresses and instructions. Always it fetches two bytes for instruction and data. This architecture uses condition code register which has two flags Carry flag (C) and Zero flag (Z). The IITB-RISC-23 is very simple, but it is general enough to solve complex problems.

2 List of Components Used

2.1 Arithmetic Logic Unit

The Arithmetic Logic Unit (ALU) is a crucial component within the central processing unit (CPU) of a computer, responsible for executing arithmetic and logical operations on binary data.

2.2 Decoders

They take in the IRs of different stages and give out the control signals required in that particular stage.

2.3 Writer

This unit decides whether we should keep the bubble of the stage on or off, and it dies this by taking in result from the hazard detection unit and the decoders form stage 2.It also manages when the pc will be updated.(Because if a Hazard is caused then we might need to put the IR1 IR2 and IR3 into buuble).

2.4 Hazard Detection Unit

A hazard detection unit is a crucial component within a pipelined processor responsible for identifying and managing hazards that arise due to dependencies between instructions. Hazards can include data dependencies (e.g., read-after-write) and control flow changes (e.g., branches). The hazard detection unit monitors the pipeline stages to detect these hazards and implements strategies to resolve them. For data hazards, the unit may stall the pipeline, use data forwarding techniques to supply needed data or reorder instructions. Control hazards are managed by predicting branch outcomes and flushing incorrect instructions if predictions are wrong. The functions that could generate hazards include consecutive R-type instructions in which the destination of the first one is the source of the next, conditional and unconditional jumps, load and store.

2.5 LMSM Unit

The LMSM unit is a specialized unit designed to handle LM and SM instructions within a processor. This unit takes the instruction as input and produces two outputs. First, it processes the immediate value from the instruction and determines the index of the rightmost 1 bit within that value, outputting this index as a 3-bit value. Additionally, the LMSM unit generates an 8-bit output where the bit corresponding to the identified index of the rightmost 1 bit is 0, and all other bits are set to 1. This output effectively masks out the bit at the identified index.

2.6 Register File

In computer architecture, a register file is a crucial component that stores data used by the CPU during instruction execution. In this project, having 8 registers numbered from 0 to 7, with register 0 designated as the program counter (PC), reflects a typical setup found in processor designs. Register 0, acting as the PC, holds the memory address of the next instruction to be fetched and executed.

2.7 Memory

Memory in computing refers to the electronic components used to store and retrieve digital data and instructions. Two main types of memory are data memory (RAM), which holds data being actively processed by the CPU, and instruction memory (ROM or Flash memory), which stores program instructions for the CPU to execute.

2.8 Shifter

A left shifter is a digital circuit that performs a left shift by one bit on an input binary number, effectively multiplying the number by 2. In this operation, the bits of the input number are shifted to the left by one position, with the most significant bit (MSB) being discarded, and a 0 bit inserted into the least significant bit (LSB) position.

2.9 Sign Extender

A sign extender, also known as a sign extension unit or sign extension circuit, is a digital circuit used to extend the sign bit of a binary number

For example, if we have an 8-bit signed number represented as where b_7 is the sign bit, and we want to extend it to a 16-bit signed number, the sign extender would replicate b_7 (the sign bit) across bits b_8 across b_{15}, resulting in $b_7b_7b_7b_7b_7b_7b_7b_7b_7b_6b_5b_4b_3b_2b_1$.

2.10 Multiplexer

A multiplexer (MUX) is a fundamental digital circuit used in electronics and computing to select one of many input signals and route it to a single output. Think of it as a data selector.

2.11 PC Update Unit

The PC (Program Counter) update unit, typically implemented as a simple adder circuit within a CPU, performs the essential task of incrementing the program counter value by 1 after fetching an instruction. This unit ensures that the CPU can effectively proceed to fetch the next instruction in sequence from memory.

2.12 Testbench

A testbench is a key component used in digital design and verification processes to ensure the correctness and functionality of hardware designs or modules before physical implementation. It consists of a separate module that interacts with the design under test (DUT).

2.13 A few example testcases

In this particular case, the testcase was a carefully constructed memory that will check collisions in code and see to it that the right output is given in the right place. We tested on various instructions and few of them are mentioned below :

```
signal memory_storage : array_of_vectors :=
(0 => "0001101111001000", -- this is r1 = r5 + r7 with no condition
1 => "0001101111001010", --ADC or Add if carry = 1   r1 = r5 + r7
2 => "0001101111001001", --ADZ or ADD if zero = 1  r1 = r5 + r7
3 => "0001101111001011", --AWC or ADD with carry r1 = r5 + r7 + carry
others => "0000000000000000");
```

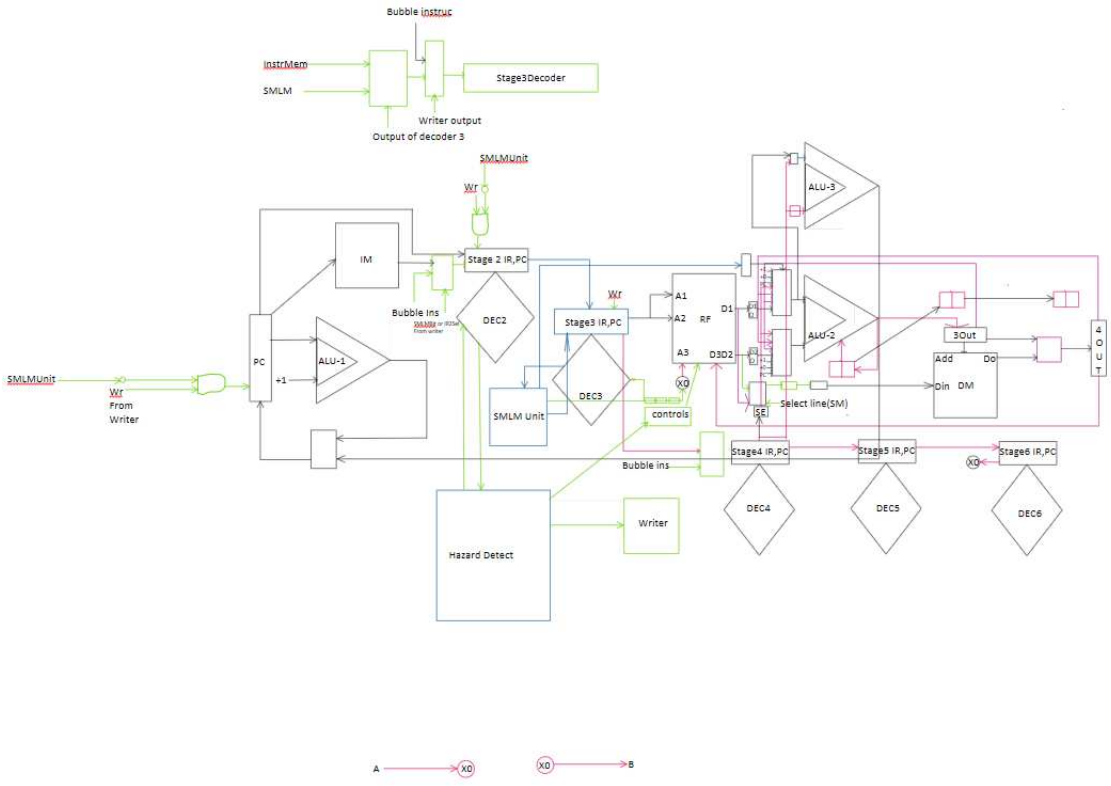
and

```
signal memory_storage : array_of_vectors :=
(0 => "0001101111001000", -- this is r1 = r5 + r7 with no condition
1 => "0001101111001010", --ADC or Add if carry = 1   r1 = r5 + r7
2 => "0110 010 0 11100000", -- LM from M[r3]->r5, M[r3 + 1]->r6, M[r3+2]->r7
3 => "0110 010 0 00011000",  -- LM from M[r3]->r3 and M[r3 + 1]->r4
4 => "0001101111001000", --r1 = r5 + r7 with no condition
5 => "0001 001 111 110 0 00", --r6 = r1 + r7 with no condition -- collision
6 => "0001 001 010 111 0 00", --r7 = r2 + r1 --collision with 4
```

```
7 => "0010 001 010 111 0 00", -- r7 = r2 nand r1
8 => "0011 010 000001110", -- LLI immediate in the lower values of R3
9 => "0111 010 000000111", --SM but values is taken from r3
10 => "1000 101 011 000001", -- BEQ for the WIN  no collision here though
others => "0000000000000000");
```

3 DataPath

The Datapath shows all the connections being made and gives a quick glimpse of the project.Here the datapath consists of the ALU1, 2, 3, Register file, Muxes, PC and various registers for storing the value of the IR and components like LMSM unit and Hazard detect unit and Writer.



The above figure can explain the functioning of how a instruction is processed and dealt with, each instruction takes one cycle in each of the stages and then gets stored in the pipeline register ahead. It also shows how forwarding will

be used to avoid unnecessary penalty in the system and lead to good Cycles per instruction.

4 Results

The Final product can execute all the instructions in the instruction set and provide accurate results.
The RISC architecture implemented here can tackle collisions in coding and continue operation without taking in penalty for the same.
The LM and SM operations require delay which is dependent on the number of writes we have to perform in that instance, but the logic implemented makes sure that no more than necessary delays happen.
The Branching and Jumping also take in a certain amount of delay that depends on whether they take the branch or not.
The Controller makes sure that all the control signals - Register write, Register read, Enable, Memory write etc work at the right time and place.

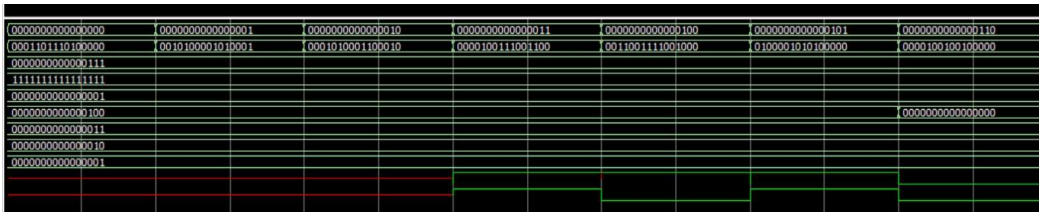


Figure 1: Addition happening with the delay of 6 cycles

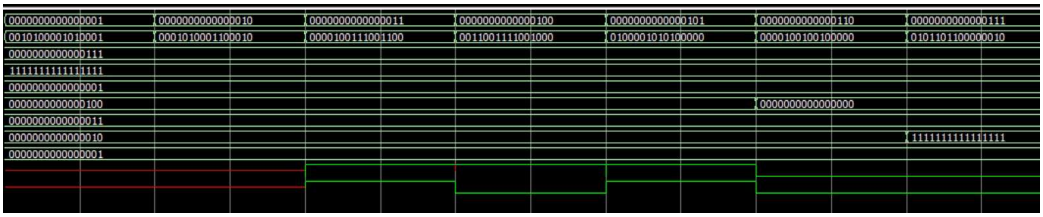


Figure 2: Conditional Nand operation

5 Simulation Results

Here we will show the working of the major instructions and the observed simulations and then we will look into some simulations showing the working of Hazard detect system.

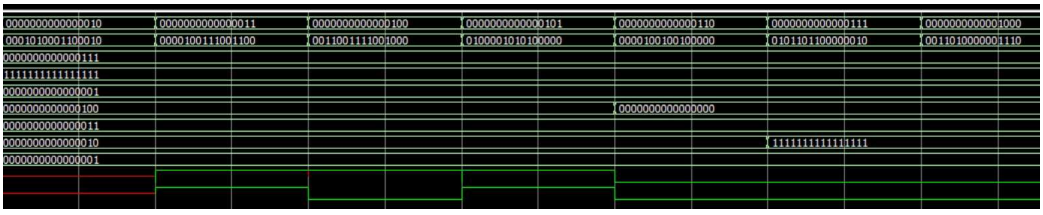


Figure 3: Conditional Addition operation

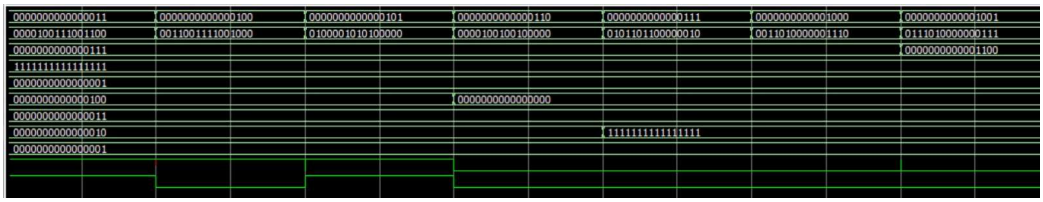


Figure 4: Immediate addition

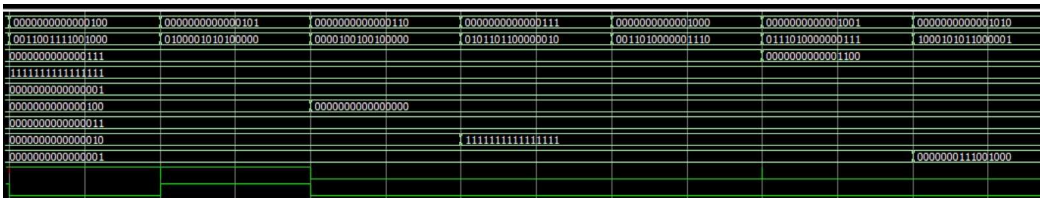


Figure 5: LLI instruction

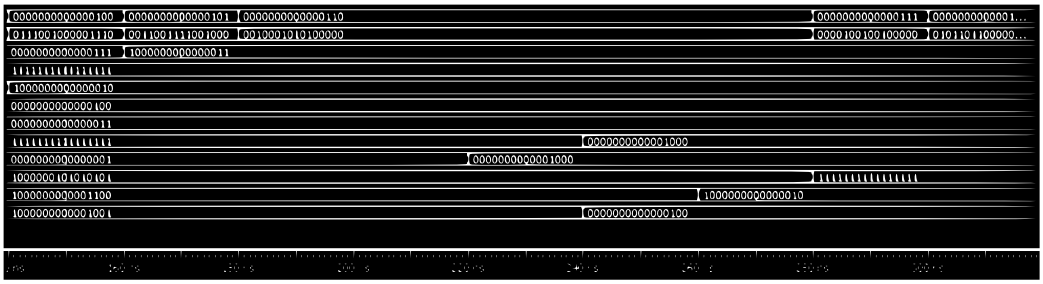


Figure 6: Store Many - Starts writing from the 5th cycle

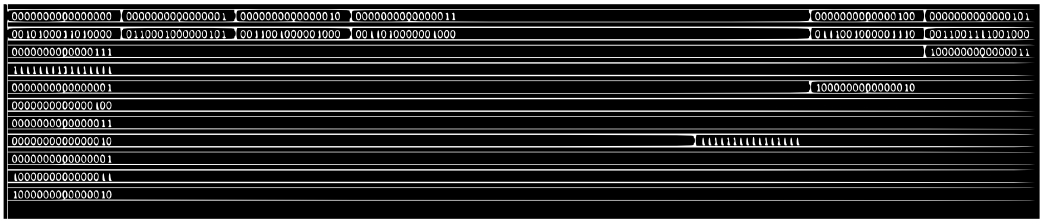


Figure 7: Load Many(LM)

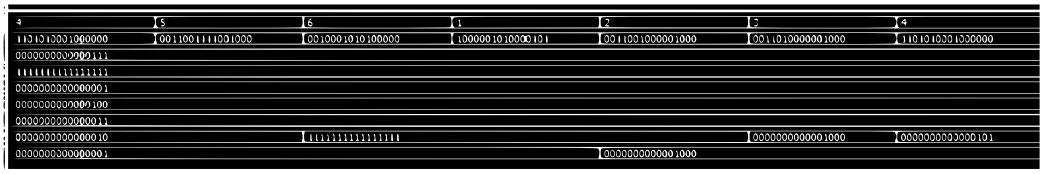


Figure 8: Jump instruction changing PC

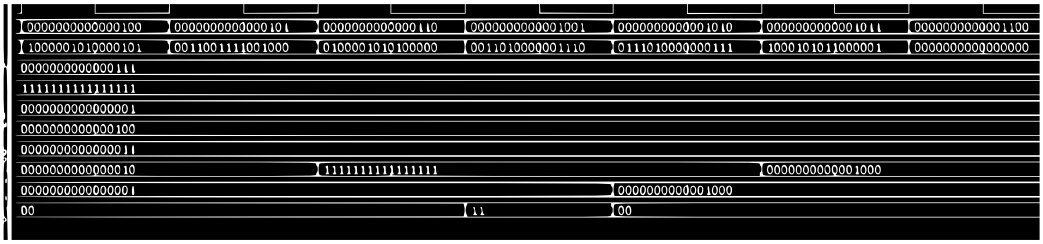


Figure 9: BEQ jumping

6 Contributions

- **Prajwal**: LMSM, Hazard Detection, Register File, Controls, Decoder4, Report.
- **Sarvadnya**: ALU Designing, Hazard Detection, memory, Controls, Decoder3.
- **Shikhar**: Control Unit Design, Hazard Detection Unit, Decoders 5 and 6, Report, Testing
- **Siddick**: Datapath, Controls, Decoder 2, Hazard Detection Interfacing, Writer, Testing