EE 309 Project Report

Dhatri Mehta, Dhruvi Ganatra, Kartik Singhal, Snehaa Reddy

May 3, 2023

Contents

1 Introduction

The project required us to design a 6-stage pipelined processor, IITB-RISC-23, whose instruction set architecture was provided. IITB-RISC is a 16-bit very simple computer developed for teaching that is based on the Little Computer Architecture. IITB-RISC-23 is a 16-bit computer system with 8 registers. It follows the standard 6 stage pipelines (Instruction fetch, instruction decode, register read, execute, memory access, and write back). The architecture is optimized for performance, i.e., includes hazard mitigation techniques, in the form of forwarding mechanism.
This report consists of the instruction set provided, the state transition diagrams that our team drew, the code that describes the functioning of the CPU, and the simulation results.

2 Instructions

In computer science, an instruction set architecture (ISA), also called computer architecture, is an abstract model of a computer. A device that executes instructions described by that ISA, such as a CPU, is called an implementation. In general, an ISA defines the supported instructions, data types, registers, the hardware support for managing main memory, fundamental features (such as addressing modes), and the input/output model of a family of implementations of the ISA. Given below is the complete instruction set for IITB-CPU.

2.1 Instruction Set Architecture

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

2.2 Instruction Formats

The instruction format is a pattern of bits that control unit of the CPU can decode. The instruction format describes the layout of the instruction in terms of group of bits called fields of the instruction format. There are three machine-code instruction formats (R, I, and J type) and a total of 26 instructions. They are illustrated in the figure below.

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)

Figure 1: Types of instruction format

2.3 Instruction Encoding

Given below is the encoding of each instruction, mentioning the significance of each bit in the 16-bit length. The following abbreviations are to be kept in mind.
RA: Register A
RB: Register B
RC: Register C

ADA:	00_01	RA	RB	RC	0	00
ADC:	00_01	RA	RB	RC	0	10
ADZ:	00_01	RA	RB	RC	0	01
AWC:	00_01	RA	RB	RC	0	11
ACA:	00_01	RA	RB	RC	1	00
ACC:	00_01	RA	RB	RC	1	10
ACZ:	00_01	RA	RB	RC	1	01
ACW:	00_01	RA	RB	RC	1	11
ADI:	00_00	RA	RB	6 bit Immediate		
NDU:	00_10	RA	RB	RC	0	00
NDC:	00_10	RA	RB	RC	0	10
NDZ:	00_10	RA	RB	RC	0	01
NCU:	00_10	RA	RB	RC	1	00
NCC:	00_10	RA	RB	RC	1	10
NCZ:	00_10	RA	RB	RC	1	01
LLI:	00_11	RA	9 bit Immediate			
LW:	01_00	RA	RB	6 bit Immediate		
SW:	01_01	RA	RB	6 bit Immediate		
LM:	01_10	RA	0 + 8 bits corresponding to Reg R0 to R7 (left to right)			
SM:	01_11	RA	0 + 8 bits corresponding to Reg R0 to R7 (left to right)			
BEQ:	10_00	RA	RB	6 bit Immediate		
BLT	10_01	RA	RB	6 bit Immediate		
BLE	10_01	RA	RB	6 bit Immediate		

JAL:	11_00	RA	9 bit Immediate offset	
JLR:	11_01	RA	RB	000_000
JRI	11_11	RA	9 bit Immediate offset	

3 Implementation

The first step of implementing the processor was drawing the datapath.We sketched a rough datapath starting with the ADD instructions, and kept on augmenting it with other components whenever needed. In this manner, we finally came up with the datapath given below.

3.1 Components

1. **RRINDECIDER** : Decides the input pipeline register for the RR stage component

2. **RR** : Takes the instruction as an input and returns the address given by the priority encoder, and resets the last 8 bits of the instruction. Basically, it contains the entire description of the priority encoder.

3. **MEMACCESS** : Decides the values of input data and address of the data memory

4. **loader**: Selects the data that will be read into the register file, along with its address, based on if-else statements, that condition solely on the opcode of instructions

5. **RF_C_Z** : Selectively used to enable/disable writing into the Register File, the C and the Z flag, based on instruction opcode

6. **Stall_and_throw** : Implements stalling (in case of load) and throwing of instructions (in case of taken branches and jumps) by checking value of CHECKBIT and THROWBIT, set by load_hazards and PC_MUX_Control_unit, respectively

7. **load_hazards** : Detects presence of load hazard by considering opcode of instructions (to check if write back is taking place), and comparing whether the source register of an instruction is the same as the destination register of an earlier instruction

8. **Executor** : Looks at the opcode of instruction and decides (via cases) what will be the inputs to ALU2, and also what function ALU2 will perform in the execution stage of that instruction. It basically outputs the control select lines for the 2 MUXes at the inputs of ALU2, as well as the ALU select lines.

9. **Forwarding unit (FU)** : Takes the ALU2 mux signals computed by the execution unit and determines if there is any data dependency. If yes, then the FU amends the mux signals accordingly. This is implemented using carefully structured if-else statements by taking all possibilities of data hazards into account. It is divided into:

    - FwdA : Forwarding unit that provides control signals for the mux at input A of ALU 2. Looks at both 1-length and 2-length dependency
    - FwdB : Forwarding unit that provides control signals for the mux at input B of ALU 2. Looks at both 1-length and 2-length dependency

10. **PC_MUX_Control_unit** : Provides control signals to the MUX that decides the input to PC. Takes in the instruction, and the C and Z flags as input, and in addition to determining which ALU output feeds the PC input, also sets the value of the THROWBIT, which has been used for discarding instructions in case of taken branches and jumps

11. **SignExtender** : Sits in the execution stage in front of the MUX for ALU B instructions. Performs all types of extensions, namely, 9 bit to 16 bit (signed and unsigned), and 6 bit to 16 bit (signed), by looking at the instruction opcode

12. **RegisterFile** : Contains 8 registers with 16 bits (2 bytes) capacity each, R0 being the PC. Has 4 input ports (3 for address, and 1 for data) and 2 output ports (both for data) for accessing the registers R! to R7.Has separate input-output ports for accessing R0, since that stores PC. Thus, the register file has 2 write enable signals, for the normal register file functionality, and one for the PC functionality

13. **register_1bit** : 1-bit register to create the C and Z flags

14. **PipelineRegister** : A 123-bit register to hold important values for an instruction in the pipeline, when it passes from one pipeline stage to another. Given below is the pipeline register division:

Table 1: Pipeline register division

Bits	**Function**
00-15	Instruction
16-31	Data1 (content of reg A)
32-47	Data2 (content of reg B)
48	(currently vestigial)
49	Dmem_we
50	C_WE
51	Z_WE
52	RF_WE
53-68	Data3 (That has to be stored in the memory in SM)
69-84	PC
85-100	ALU_output
101-102	Vestigial
103-105	Address given by priority encoder
107-122	Mem_Access output

15. **MUXes**:

    - **Mux_4to1_16bit** : A 4-to-1 MUX, used at the 3 locations in the datapath. The location, function and controlling entity of each MUX is explained in a table at the end.
    - **Mux_2to1_3bit** : A 2-to-1 MUX, used at the A1 input of the register file, used to select address of data1
    - **Mux_2to1_8bit** : A 2-to-1 MUX, used at the output of the RR-EX pipeline register
    - **Mux_2to1_16bit** : A 2-to-1 MUX, used at the input of EX-MEM pipeline register and output of RR-EX pipeline register

16. **InstructionMemory** : Takes in PC as input and outputs instruction corresponding to that address.

17. **Datapath** : This contains the complete collection of functional units such as the ALU, memories, register file, hazard detection units etc. The images are given at the end of the report.

18. **Datamemory** : Takes in data and its address as input, outputs data. If Memory_write_enable is high then we write to the Memory (i.e. input to the Memory), if low then we read from it.

19. **ALU** : The ALU performs three primary functions as described below:

    (a) ADD (00) : This function is highly specialized. With the use of if-else statements (that check the 'complement' bit in the instruction), the ADD function is made to:

        i. Carry out addition of contents of reg A with the complement of contents of reg B, for instructions like ACA, ACC, ACZ

        ii. Carry out addition of contents of reg A, complement of reg B as well as the carry bit, for the instruction ACW

        iii. Carry out addition of contents of reg A, reg B as well as the carry bit, for the instruction AWC

        iv. Carry out simple addition of contents of reg A and reg B (This is the setting in which ALU 1 and ALU 3 permanently operate)

    (b) NAND (01) : Similar to ADD, NAND also takes care of the following:

        i. Carries out bitwise NAND of contents of reg A with complement of reg B (in case of NCU, NCC, NCZ instructions)

        ii. Carries out bitwise NAND of contents of reg A with contents of reg B (in case of NCU, NCC, NCZ instructions)

    (c) SUBB (10) : Subtracts contents of reg B from those of reg A (if reg A is fed to ALU2 A, and reg B to ALU2 B)

3.2 Results

Given below are the simulation results for instruction 01010010 01000000, where r1 has value x0001.

Location	Function	Controlled by
PC input	Determines which ALU output feeds the PC input	PC_MUX_Control_unit
ALU2 input A	Decides whether the input to ALU2 (A) is content of reg A or content of reg B or PC	FwdA
ALU2 input B	Decides whether the input to ALU2 (B) is output of SignExtender or content of reg B or +1	FwdB

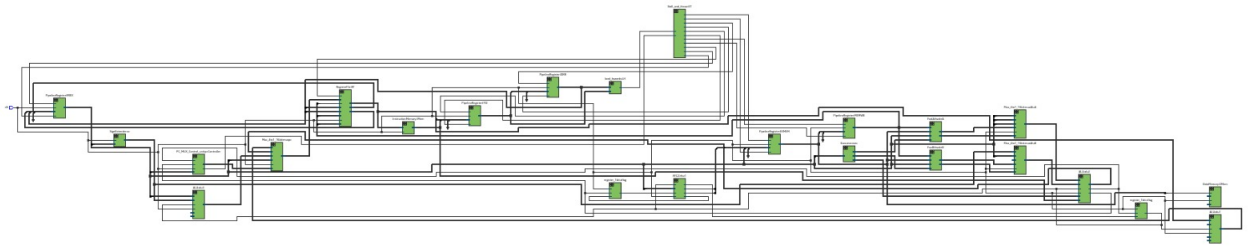

Figure 2: Complete datapath (as seen on RTL viewer)

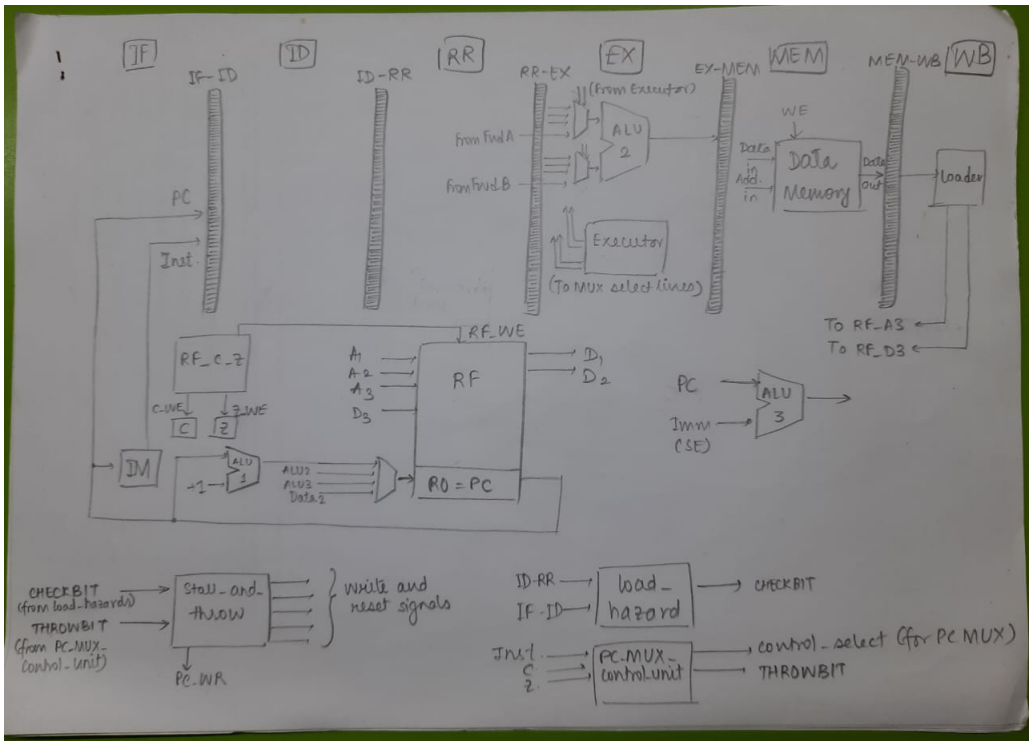

Figure 3: Complete datapath (Hand-drawn image, to emphasise all the components used)

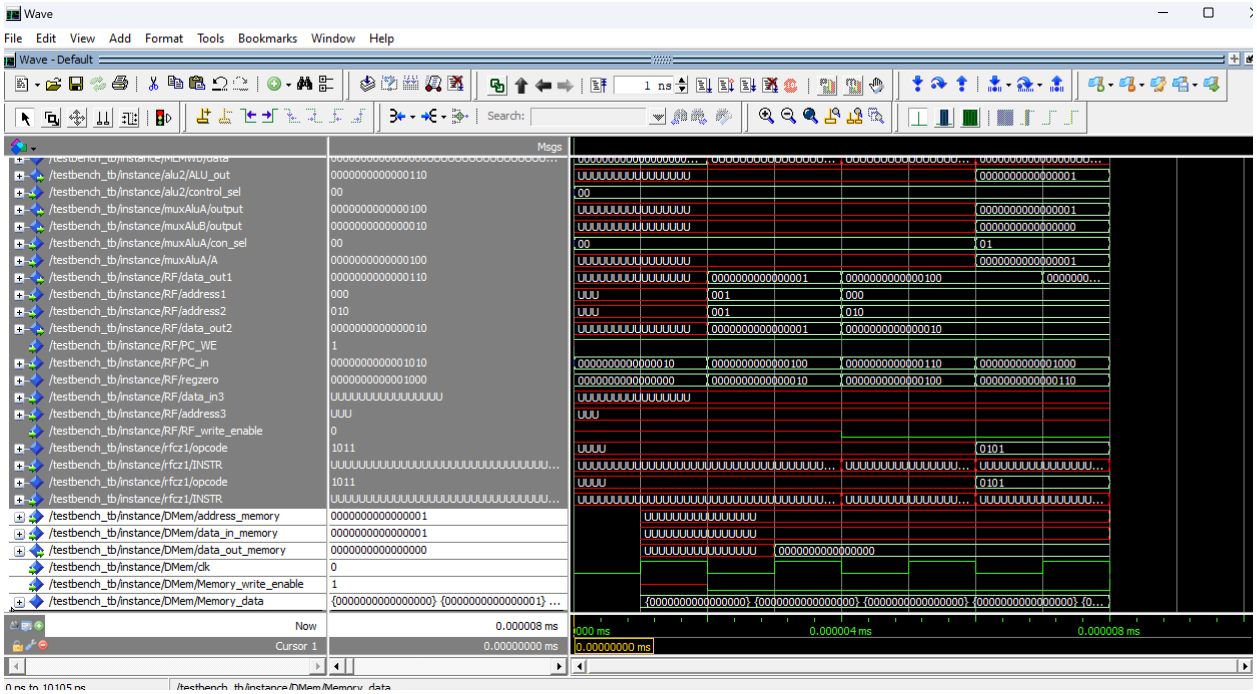

Figure 4: Waveforms for SW instruction