IITB CPU

Project Report for Indian Institute of Technology, Bombay - EE224: Digital Design (2022)

Aditya Kabare
Dept. of Electrical Engineering
Indian Institute of Technology, Bombay
Mumbai, Maharashtra
21D070009@iitb.ac.in

Anuj Gupta
Dept. of Electrical Engineering
Indian Institute of Technology, Bombay
Mumbai, Maharashtra
21D070014@iitb.ac.in

Mrudul Jambhulkar
Dept. of Electrical Engineering
Indian Institute of Technology, Bombay
Mumbai, Maharashtra
21D070044@iitb.ac.in

Saurav Kumar
Dept. of Electrical Engineering
Indian Institute of Technology, Bombay
Mumbai, Maharashtra
21D070063@iitb.ac.in

***Abstract*—This document is a report of our project aiming at designing a computer system whose instruction set architecture was provided. We used VHDL as HDL to implement a 16-bit very simple computer. The IITB-CPU is a 8-register, 16-bit computer system, i.e., it can process 16 bits at a time.**

I. Introduction

IITB-CPU is a 16-bit very simple computer based on the little Computer Architecture. It is an 8-register, 16-bit computer system and has 8 general-purpose registers (R0 to R7). For simplicity, we created datapath, controller and various components separately including the program counter which stores next instruction. Our job was to design the processor to implement the given 15 instructions. This architecture uses condition code register which has two flags Carry flag (c) and Zero flag (z) and there are three machine-code instruction formats (R, I, and J type) which are given in figure 1.
Instructions Encoding is given in figure 2.

R Type Instruction format

Opcode (4 bit)	Register A (RA) (3 bit)	Register B (RB) (3-bit)	Register C (RC) (3-bit)	Unused (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)

Fig. 1. Machine-code Instructions

II. Design

We followed a standard practice of dividing the code into datapath and controller where datapath includes the actual arrangement and controller is basically a Finite State Machine.

ADD:	00_00	RA	RB	RC	0	00
ADC:	00_00	RA	RB	RC	0	10
ADZ:	00_00	RA	RB	RC	0	01
ADI:	00_01	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
LHI:	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 R7 to R0			
SM:	01_11	RA	0 + 8 bits corresponding to Reg R7 to R0			
BEQ:	11_00	RA	RB	6 bit Immediate		
JAL:	10_00	RA	9 bit Immediate offset			
JLR:	10_01	RA	RB	000_000		

Fig. 2. Instructions Encoding

A. Controller

Controller is a 20 states finite state machine which takes flag signals as input and puts the control signals. It basically works with the datapath to implement the processor. Moreover, each states puts its own control signals which are necessary for the datapath to function. The control signals are :-

★ t1_E, ★ pc_select, ★ pc_E, ★ alu_op_select, ★ alu_a_select, ★ alu_b_select, ★ mem_add_select, ★ mem_data_select, ★ mem_write, ★ mem_read, ★ rf_a2_select, ★ rf_a3_select, ★ rf_d3_select, ★ t2_select, ★ t2_E, ★ t3_select, ★ t3_E, ★ t4_E, ★ t5_select, ★ t5_E.

B. Data Path

A datapath is a collection of functional units such as arithmetic logic units, memory, registers etc that perform data processing operations. It takes control signals as input and outputs the flags. The schematic is given in figure 3.

1) Components Used: We used various components and multiplexers to design the datapath. The components are :=

- **ALU** - It is designed to perform addition, subtraction and NAND operations.
- **continue_flag_tracker** - Getting to next instruction in various cases is contrained, that's what this flag solves.
- **AND Gate** - Normal 8 bit and gate.
- **data_postfix** - It is basically a extender which concatenate the vector according to the instruction.
- **decoder_for_Encoder** - It decodes the result of priority encoder in a special way which is helpful in proper implementation of the instruction.
- **op_code_finder** - It takes and instruction as input and outputs the first 4 MSBs.
- **PE** - Normal Priority Encoder.
- **Memory** - A storage element with read & write enables, address in, data in and data out ports along with clock and reset.
- **register File** - A storage element with read & write enables, three address in, data in and two data outs ports along with clock and reset.
- **t1** - A component which synchonizes the instruction register with enable.
- **Temporary Register** - Register to store 16-bit temporary data.
- **Temporary Register 3 bit** - Register to store 3-bit temporary data.
- **Temporary Register 8 bit** - Register to store 8-bit temporary data.
- **SE6** - Sign extension of 6-bit data.
- **SE9** - Sign extension of 9-bit data.
- **Zero_checker** - It results 1 if input is 0 else results 0.

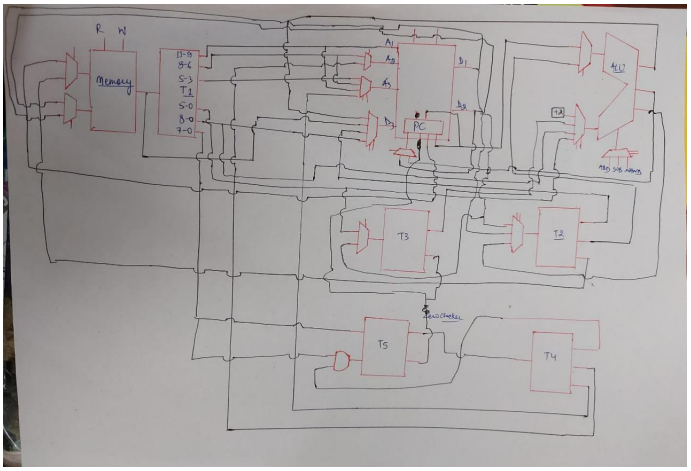

Fig. 3. Schematic

III. Simulation Results

We performed simulation of controller using regular test-bench & DUT and self prepared Tracefile to ensure proper working of all the states. The simulation results can be found below.

A. ADD

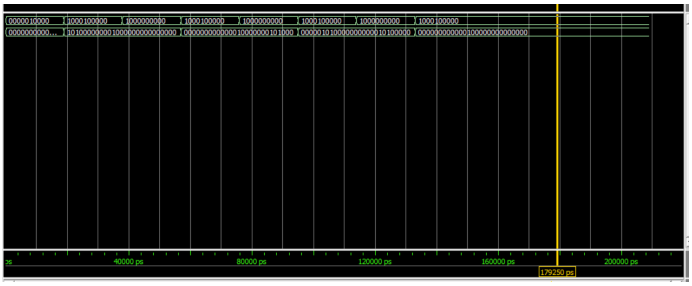

Fig. 4. ADD

B. ADC

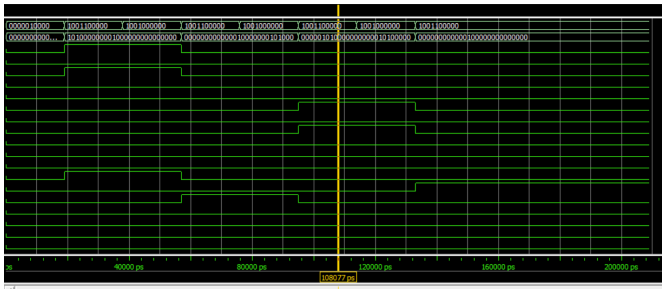

Fig. 5. ADC

C. ADZ

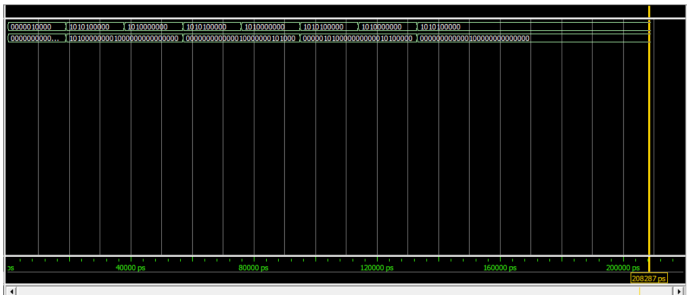

Fig. 6. ADZ

D. ADI

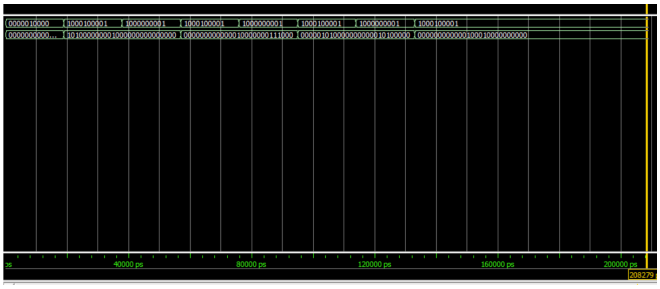

Fig. 7. ADI

E. NDU

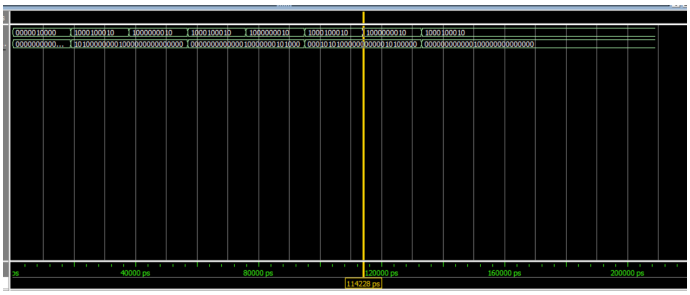

Fig. 8. NDU

F. NDC

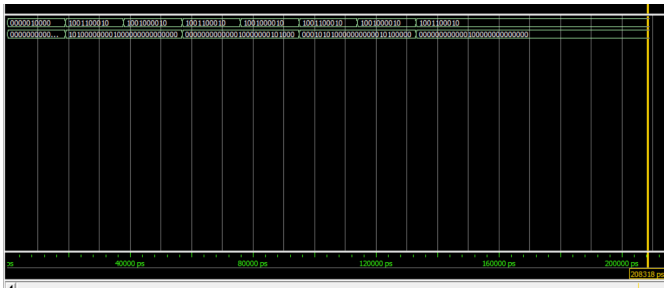

Fig. 9. NDC

G. NDZ

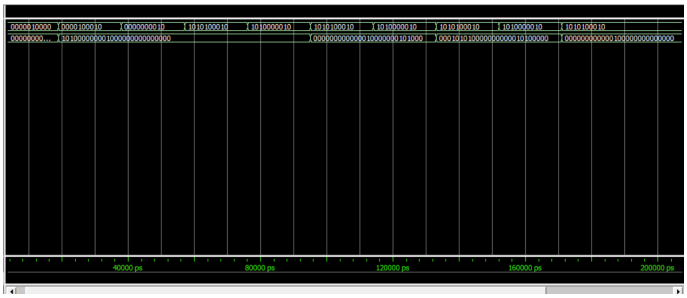

Fig. 10. NDZ

H. LHI

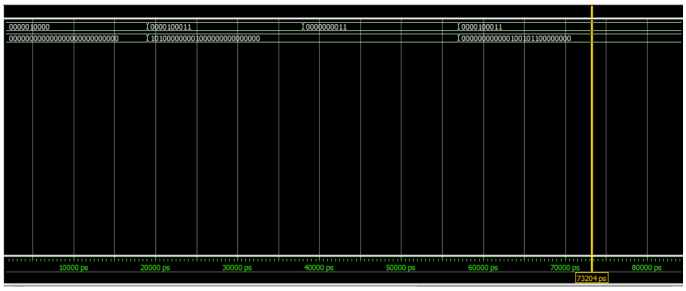

Fig. 11. LHI

I. LW

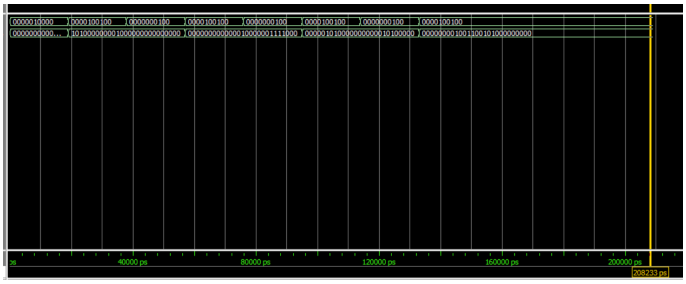

Fig. 12. LW

J. SW

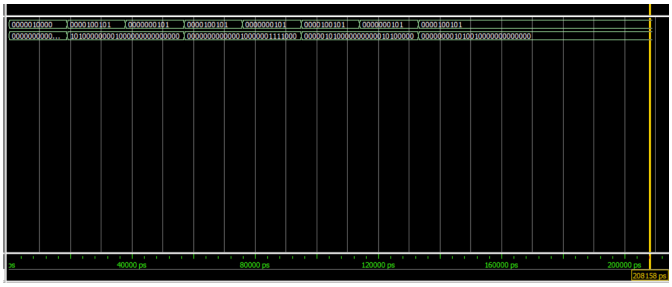

Fig. 13. SW

K. LM

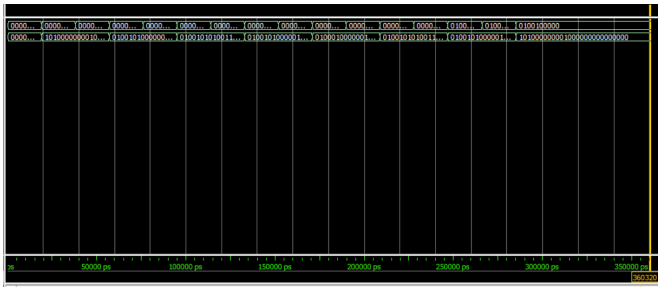

Fig. 14. LM

L. SM

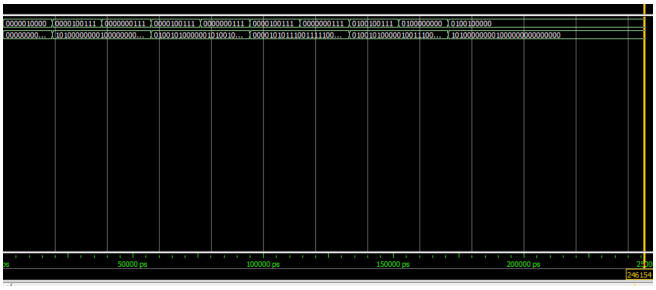

Fig. 15. SM

M. BEQ

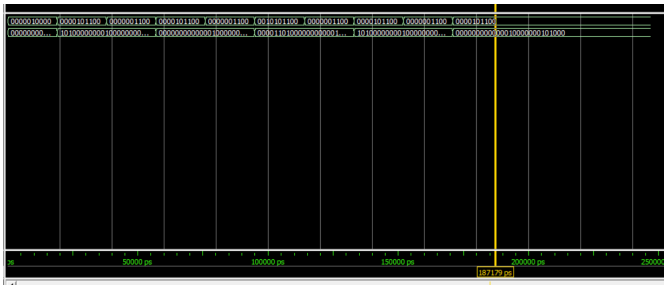

Fig. 16. ADI

N. JAL

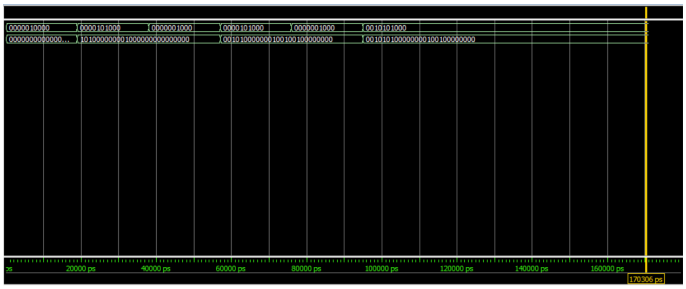

Fig. 17. JAL

O. JLR

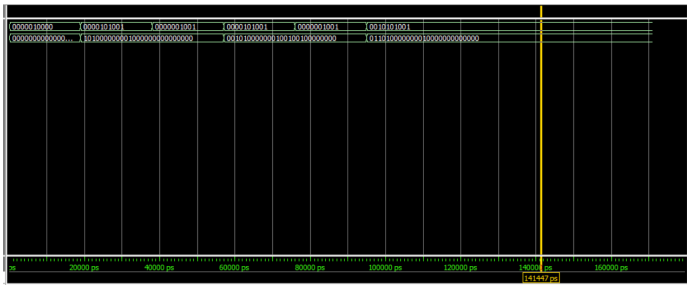

Fig. 18. JLR

IV. Conclusion

This Project taught us a lot about the CPU and its dependence on various components crucial for its flawless working. Various challenges were encountered at different points but we, with great enthusiasm, tackled most of them to present the best we could. Though, we performed simulation of controller to ensure correctness of state transition process, we were unable to compile and simulate the whole code together due to errors in testbench we made. Nevertheless, the project experience for all of us was tremendous and we learnt a lot in the process.

References

[1] https://www.youtube.com/watch?v=cIDoJtYdDVA
[2] http://people.sabanciuniv.edu/erkays/el310/ComplexSequential