IITB-RISC-23

EE309 COURSE PROJECT

ANUJA SATHE – 210070010
AYAN DAS- 210070016
BHAKTI MATSYAPAL - 210070021
DEBASISH PANDA – 21d070021

Contents

Overview of the project:

Designing a 6-stage pipelined processor, IITB-RISC-23, as per the ISA provided. IITB-RISC is a 16-bit very simple computer developed for the teaching that is based on the Little Computer Architecture. The 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., it includes hazard mitigation techniques which include feed-forwarding paths.

IITB-CPU Instruction Set Architecture

IITB-RISC is a 16-bit very simple computer developed for the 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 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. The architecture allows predicated instruction execution and multiple load and store execution. There are three machine-code instruction formats (R, I, and J type) and a total of 14 instructions.

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

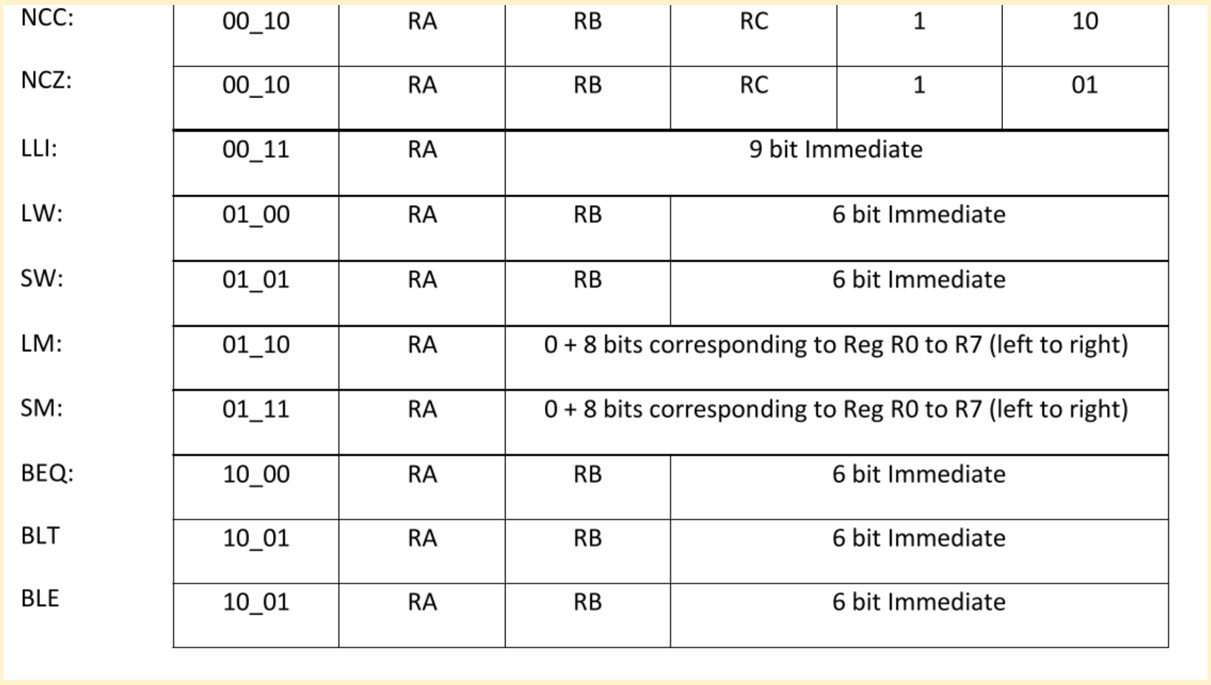
Opcode (4 bit)	Register A (RA) (3 bit)	Register C (RC) (3-bit)	Immediate (6 bits signed)

J Type Instruction format

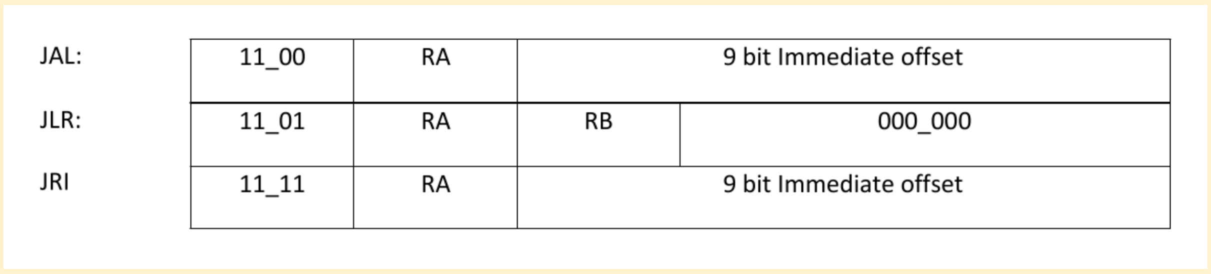
Opcode (4 bit)	Register A (RA) (3 bit)	Immediate (9 bits signed)

Instructions Encoding:

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	

Design:

Instruction executions:

1. <u>ADA</u>:

PC $\rightarrow$ MEM_ADDR, ALU2_A	ALU1:ADD
MEM1 _DATA$_{11-9}$ $\rightarrow$ RF_A1	MEM1_RD=1
MEM1 _DATA$_{8-6}$ $\rightarrow$ RF_A2	RF_RD=1
RF_D1 $\rightarrow$ ALU1_A	RF_WR=1
ALU1_C $\rightarrow$ RF_D3	CMP=0
MEM1 _DATA$_{5-3}$ $\rightarrow$ RF_A3	MEM2_RD=0
+2 $\rightarrow$ ALU2_B	MEM2_WR=0
ALU2_C $\rightarrow$ PC	

2. <u>ADC</u>:

PC $\rightarrow$ MEM-1_ADDR, ALU2_A	ALU1:ADD
	MEM1_RD=1
if (C==1) : ADA	RF_RD=1
else :	RD_WR=1
+2 $\rightarrow$ ALU2_B	CMP=0
ALU2_C $\rightarrow$ PC	MEM2_RD=0
	MEM2_WR=0

3. <u>ADZ</u>:

PC $\rightarrow$ MEM1_ADDR, ALU2_A	ALU1:ADD
if (Z==1) : ADA	MEM1_RD=1
else :	RF_RD=1
+2 $\rightarrow$ ALU2_B	RD_WR=1
ALU2_C $\rightarrow$ PC	CMP=0
	MEM2_RD=0
	MEM2_WR=0

4. <u>AWC</u>:

PC $\rightarrow$ MEM1 _ ADDR, ALU2_A	ALU1: ADD
MEM1_DATA$_{11-9}$ $\rightarrow$ RF_A1	MEM1_RD=1

MEM1_DATA$_{8-6}$ → RF_A2
RF_D1 → ALU1_A
RF_D2 → ALU1_B
ALU1_C → ALU4_A
C → SE1 → ALU4_B
ALU4_C → RF_D3
MEM1_DATA$_{5-3}$ → RF_A3
+2 → ALU2_B
ALU2_C → PC
	RF_WR=1
	RF_RD=1
	CMP=0
	MEM2_RD=0
	MEM2_WR=0

5. <u>ACA</u>:
   REPLACED CMP=1 IN ADA
6. <u>ACC</u>:
   REPLACED CMP=1 IN ADC
7. <u>ACZ</u>:
   REPLACED CMP=1 IN ADZ
8. <u>ACW</u>:
   REPLACED CMP=1 IN AWC
9. <u>ADI</u>:

PC → MEM1 _ADDR , ALU2_A
MEM1 _DATA$_{11-9}$ → RF_A1
RF_D1 → ALU1_A
MEM1 _DATA$_{5-0}$ → SE6 → ALU1_B
MEM1_DATA$_{8-6}$ → RF_A3
ALU1_C → RF_D3
+2 → ALU2_B
ALU2_C → PC
	MEM1_RD=1
	ALU1: ADD
	RF_RD=1
	RF_WR=1
	CMP=0
	MEM2_RD=0
	MEM2_WR=0

10. <u>NDU</u>:

PC → MEM1_ADDR, ALU2_A
MEM1_DATA$_{11-9}$ → RF_A1
MEM1_DATA$_{8-6}$ → RF_A2
RF_D1 → ALU1_A
RF_D2 → ALU1_B
ALU1_C → RF_D3
+2 → ALU2_B
ALU2_C → PC
	MEM1_RD=1
	ALU1: NAND
	RF_RD=1
	RF_WR=1
	CMP=0
	MEM2_RD=0
	MEM2_WR=0

11. <u>NDC</u>:

PC → MEM1_ADDR, ALU2_A
if (C==1) : NDU
else:
+2 → ALU2_B
ALU2_C → PC
	MEM1_RD=1
	ALU1:NAND
	RF_RD=1
	RF_WR=1
	CMP=0
	MEM2_RD=0
	MEM2_WR=0

12. NDZ:

PC → MEM1_ADDR, ALU2_A	MEM1_RD=1
if (Z==1) : NDU	ALU:NAND
else:	RF_RD=1
+2 → ALU2_B	RF_WR=1
ALU2_C → PC	CMP=0
	MEM2_RD=0
	MEM2_WR=0

13. NCU:

REPLACED CMP=1 IN NDU

14. NCC:

REPLACED CMP=1 IN NDC

15. NCZ:

REPLACED CMP=1 IN NDZ

16. LLI:

PC → MEM1_ADDR, ALU2_A	MEM1_RD=1
+2 → ALU2_B	RF_RD=0
ALU2_C → PC	RF_WR=1
MEM1_DATA$_{8-0}$ → SE9 → RF_D3	CMP=0
MEM1_DATA$_{11-9}$ → RF_A3	ALU:ADD
	MEM2_RD=0
	MEM2_WR=0

17. LW:

PC → MEM1_ADDR, ALU2_A	MEM1_RD= 1
+2 → ALU2_B	ALU1: ADD
ALU2_C → PC	RF_RD=1
MEM1_DATA$_{8-6}$ → RF_A1	RF_WR= 1
RF_D1 → ALU1_A	CMP=0
MEM1_DATA$_{5-0}$ → SE6 → ALU1_B	MEM2_RD=1
ALU1_C → MEM2_ADDR	MEM2_WR=0
MEM2_DATA → RF_D3	
MEM1_DATA → RF_A3	

18. SW:

PC → MEM1_ADDR, ALU2_A	MEM1_RD=1
+2 → ALU2_B	ALU1:ADD
ALU2_C → PC	RF_RD=1
MEM1_DATA$_{5-0}$ → SE6 → ALU1_B	RF_WR=0
ALU1_C → MEM2_ADDR	MEM2_WR=1
MEM1_DATA$_{11-9}$ → RF_A2	MEM2_RD=0
RF_D2 → MEM2_DATA	CMP=0

19. BEQ:

PC → MEM1_ADDR, ALU2_A	MEM1_RD=1
+2 → ALU2_B	ALU3: XOR
MEM1_DATA$_{11-9}$ → RF_A1	RF_RD=1
MEM1_DATA$_{8-6}$ → RF_A2	RF_WR=0
RF_D1 → ALU1_A	MEM2_WR= 0
RF_D2 → ALU1_B	MEM2_RD=0
ALU2_C → ALU3_A	CMP=0
MEM1_DATA$_{5-0}$ → SE6 → S-1 → ALU3_B	
if (ALU1_Z==1) : ALU3_C → PC	
else:	
ALU2_C → PC	

20. BLT :

REPLACED ALU1_Z BY ALU1_C IN BEQ

21. BLE:

REPLACED ALU1_Z BY (ALU1_Z \|\| ALU1_C) IN BEQ

22. JAL:

PC → MEM1_ADDR, ALU2_A	MEM1_RD=1
+2 → ALU2_A	ALU3 : ADD
MEM1_DATA$_{11-9}$ → RF_A3	RF_RD= 0
ALU2_C → ALU3_A, RF_A3	RF_WR= 1
MEM1_ DATA$_{8-0}$ → SE9 → S-1 → ALU3_B	CMP=0
ALU3_C → PC	MEM2_RD=0
	MEM2_WR=0

23. JLR:

PC → MEM1_ADDR, ALU2_A	MEM1_RD=1
+2 → ALU2_B	RF_RD=1
MEM1_DATA$_{11-9}$ → RF_A3	RD_WR=1
ALU2_C → ALU3_A	CMP=0
+0 → ALU3_B	MEM2_RD=0
ALU3_C → RF_D3	MEM2_WR=0
MEM1_DATA$_{8-6}$ → RF_A1	ALU3:ADD
RF_D1 → PC	

24. JLI:

PC → ALU2_A	MEM1_RD=1
+2 → ALU2_B	RF_RD=1
ALU2_C → PC	RF_WR=0
MEM1_DATA$_{8-0}$ → RF_A1	CMP=0
RF_D1 → ALU1_A	MEM2_RD=0
MEM1_DATA$_{8-0}$ → SE9 → S-1 → ALU1_B	MEM2_WR=0
ALU1_C → PC	ALU1:ADD

25. <u>LM:</u>

PC $\rightarrow$ MEM1_ADDR, ALU2_A	MEM1_RD=1
+2 $\rightarrow$ ALU2_B	RF_RD=1
ALU2_C $\rightarrow$ PC	RF_WR=1
MEM1_DATA$_{11-9}$ $\rightarrow$ RF_A1	ALU1: ADD
for i: 0 to 7 :	CMP=0
[if i==0:	MEM2_RD=1
if imm(i) ==1:	MEM2_WR=0S
RF_D1 $\rightarrow$ MEM2_ADDR	
MEM2_DATA_OUT $\rightarrow$ RF_D3	
111 $\rightarrow$ RF_D3	
RF_D1 $\rightarrow$ ALU1_A	
01 $\rightarrow$ ALU1_B	
ALU1_C $\rightarrow$ RF_D3	
MEM1_DATA$_{11-9}$ $\rightarrow$ RF_A3]	
THE ABOVE BLOCK IS REPEATED WITH 111 REPLACED BY 110 AND SO ON TILL i==7	

26. <u>SM:</u>

PC $\rightarrow$ MEM1_ADDR, ALU2_A	MEM1_RD=1
+2 $\rightarrow$ ALU2_B	CMP=0
ALU2_C $\rightarrow$ PC	RF_WR=1
MEM1_DATA$_{11-9}$ $\rightarrow$ RF_A1	RF_RD=1
for i: 0 to 7:	MEM2_WR=1
[if i==0:	MEM2_RD=0
if imm(i)==1:	ALU1: ADD
RF_D1 $\rightarrow$ MEM2_ADDR	
111 $\rightarrow$ RF_A2	
RF_D2 $\rightarrow$ MEM2_DATA_IN	
RF_D1 $\rightarrow$ ALU1_A	
01 $\rightarrow$ ALU1_B	
ALU1_C $\rightarrow$ RF_D3	
MEM1_DATA$_{11-9}$ $\rightarrow$ RF_A3] REPEATED SAME AS ABOVE TILL i==7	

Control Variables:

Instruction	Opcode	PCWR	MEM1RD	MUX9	RFRD	RFWR	MUX1	MUX2	MUX3	MUX4	MUX5	MUX7	S11	S12	CMP	CIN	ZIN	MUX6	MUX10	CV	EN	MEM2RD	MEM2WR	MUX8
ADA	0001	1	1	0	1	1	1	0	01	1	11	01	0	0	0	0	0	1	10	00	11	0	0	0
ADC	0001	1	1	0	1	1	1	0	01	1	11	01	0	0	0	1	0	1	10	00	11	0	0	0
ADZ	0001	1	1	0	1	1	1	0	01	1	11	01	0	0	0	0	1	1	10	00	11	0	0	0
AWC	0001	1	1	0	1	1	1	0	01	1	11	01	0	0	0	1	1	1	11	00	11	0	0	0
ACA	0001	1	1	0	1	1	1	0	01	1	11	01	0	0	1	0	0	1	11	00	11	0	0	0
ACC	0001	1	1	0	1	1	1	0	01	1	11	01	0	0	1	1	0	1	11	00	11	0	0	0
ACZ	0001	1	1	0	1	1	1	0	01	1	11	01	0	0	1	0	1	1	11	00	11	0	0	0
ACW	0001	1	1	0	1	1	1	0	01	1	11	01	0	0	1	1	1	1	11	00	11	0	0	0
ADI	0000	1	1	0	1	1	1	X	10	1	10	01	0	0	0	X	X	1	10	00	11	0	0	0
NDU	0010	1	1	0	1	1	1	0	01	1	11	01	0	0	0	0	0	1	10	01	01	0	0	0
NDC	0010	1	1	0	1	1	1	0	01	1	11	01	0	0	0	1	0	1	10	01	01	0	0	0
NDZ	0010	1	1	0	1	1	1	0	01	1	11	01	0	0	0	0	1	1	10	01	01	0	0	0
NCU	0010	1	1	0	1	1	1	0	01	1	11	01	0	0	1	0	0	1	10	01	01	0	0	0
NCC	0010	1	1	0	1	1	1	0	01	1	11	01	0	0	1	1	0	1	10	01	01	0	0	0
NCZ	0010	1	1	0	1	1	1	0	01	1	11	01	0	0	1	0	1	1	10	01	01	0	0	0
LLI	0011	1	1	0	0	1	X	X	11	0	00	01	0	0	0	X	X	1	10	00	00	0	0	0
LW	0100	1	1	0	1	1	0	X	11	1	10	01	0	0	0	X	X	1	X	00	00	1	1	1
SW	0101	1	1	0	1	1	0	1	00	1	11	01	0	0	0	X	X	1	01	00	00	1	1	0
LM	0110																							
SM	0111																							
BEQ	1000	1	1		1	0	1	0	X	1	11	11	0	0	0	X	X	1	X	00	00	0	0	X
BLT	1001	1	1		1	0	1	0	X	1	11	11	0	0	0	X	X	1	X	00	00	0	0	X
BLE	1001	1	1		1	0	1	0	X	1	11	11	0	0	0	X	X	1	X	00	00	0	0	X
JAL	1100	1	1	1	0	1	X	X	11	X	X	10	0	0	0	X	X	1	01	X	00	0	0	0
JLR	1101	1	1	1	1	1	0	X	11	X	X	01	0	0	0	X	X	1	01	X	00	0	0	0
JRI	1111	1	1	1	X	0	1	X	X	1	01	01	0	0	0	X	X	1	X	X	00	0	0	X

Entities:

1. ALU1, ALU2, ALU3: ALUs meant for arithmetic and logical operations. There is also an additional ALU4 inside EX stage, however it's code is not there in the entities folder exclusively.
2. Branch_Jump_control: To control MUX9's output when branch or jump instructions happen.
3. Data_memory: Memory block 2, used to store data.
4. Instruction_memory: Memory block 1, used to store instructions.
5. data_hazard_1, data_hazard_2, data_hazard_3: These blocks check for data dependencies and do the requisite data forwarding.
6. Pipe_Register_control: To control the flow of information through the pipeline registers and stall the stages, as and when required.
7. Register_file: Contains 16-bit registers from R0-R7.
8. SE1: Sign extender for 1-bit data.
9. SE6: Sign extender for 6-bit data.
10. SE9: Sign extender for 9-bit data.
11. Shifter: 1-bit left shifter.
12. Stages: Contains the code required for each stage. Note that we do not need to design an entity for WB stage explicitly.

GITHUB Link