Indian Institute of Technology Bombay

Department of Electrical Engineering

EE739: Processor Design

Course Project

Problem Statement

Design a 6-stage pipelined processor, IITB-RISC-25, whose instruction set architecture is provided. IITB-RISC-25 is a 16-bit pipelined processor designed for teaching, based on the Little Computer Architecture. It features 8 general-purpose registers (R0-R7), with R0 also serving as the Program Counter (PC). The processor follows a 6-stage pipeline consisting of

1. **Instruction Fetch (IF)**
2. **Instruction Decode (ID)**
3. **Register Read (RR)**
4. **Execute (EX)**
5. **Memory Access (MEM)**
6. **Write Back (WB)**

The architecture should be optimized for performance, i.e., should include hazard mitigation techniques. Hence, it should have implemented a forwarding mechanism. Implementation of branch predictor is optional.

The processor supports predicated instruction execution, multiple load/store operations, and three instruction formats (R, I, and J-type). With a focus on efficiency, this design ensures improved performance while maintaining simplicity for educational use.

This project is intended for a group of four, allowing for effective task distribution in implementing instruction sets, pipeline design, hazard handling, and testing.

Please note that unused opcode combinations must not be utilized to mitigate hazards or handle any corner cases under any circumstances.

IITB-RISC Instruction Set Architecture:

IITB-RISC-25 is a simple 16-bit computer designed for teaching, based on the Little Computer Architecture. The IITB-RISC-25 model features an 8-register, 16-bit system with general-purpose registers (R0 to R7), where R0 also serves as the Program Counter. The memory is a two-byte addressable one. Every address points to a content whose size is two bytes. This architecture includes a condition code register with two flags: Carry (C) and Zero (Z). Despite its simplicity, IITB-RISC-25 is capable of handling complex problems. It supports predicated instruction execution and multiple load/store operations. The system uses three instruction formats (R, I, and J types) and consists of 14 instructions

IITB-RISC Instruction Formats:

R-Type Instruction Format

Opcode (4 bits)	Register A (RA) (3 bits)	Register B (RB) (3 bits)	Register C (RC) (3 bits)	Complement (1 bit)	Condition (CZ) (2 bits)

I-Type Instruction Format

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

J-Type Instruction Format

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

Instruction Encoding Table:

Mnemonic	15-14	13-12	11-9	8-6	5-3	2-0
ADA	00	00	RA	RB	RC	0 00
ADC	00	00	RA	RB	RC	0 10
ADZ	00	00	RA	RB	RC	0 01
AWC	00	00	RA	RB	RC	0 11
ACA	00	00	RA	RB	RC	1 00
ACC	00	00	RA	RB	RC	1 10
ACZ	00	00	RA	RB	RC	1 01
ACW	00	00	RA	RB	RC	1 11
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
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 followed by 8-bit Immediate		
SM	01	11	RA	0 followed by 8-bit Immediate		
BEQ	10	00	RA	RB	6-bit Immediate	
BLT	10	01	RA	RB	6-bit Immediate	
BLE	10	10	RA	RB	6-bit Immediate	

JAL	10	11	RA	9-bit Immediate Offset	
JLR	11	00	RA	RB	000 000
JRI	11	01	RA	9-bit Immediate Offset	

Instruction Descriptions:

Mnemonic	Format	Description
ADA	**ada rc, ra, rb**	Add content of regB to regA and store the result in regC. It modifies C and Z flags
ADC	**adc rc, ra, rb**	Add content of regB to regA and store the result in regC, if the carry flag is set. It modifies C & Z flags
ADZ	**adz rc, ra, rb**	Add content of regB to regA and store the result in regC, if zero flag is set. It modifies C & Z flags
AWC	**awc rc, ra, rb**	Add content of regA to regB and Carry and store result in regC regC = regA + regB + Carry It modifies C & Z flags
ACA	**aca rc, ra, rb**	Add content of regA to complement of regA and store the result in regC. It modifies C and Z flags
ACC	**acc rc, ra, rb**	Add content of regA to Complement of regB and store result in regC, if carry flag is set. It modifies C & Z flags
ACZ	**acz rc, ra, rb**	Add content of regA to Complement of regB and store result in regC, if zero flag is set. It modifies C & Z flags
ACW	**acw rc, ra, rb**	Add content of regA to Complement of regB and Carry and store result in regC regC = regA + complement of regB + Carry It modifies C & Z flags
ADI	**adi rb, ra, imm6**	Add content of regA with Imm (sign extended) and store the result in regB. It modifies C and Z flags
NDU	**ndu rc, ra, rb**	NAND the content of regA to regB and store result in regC. It modifies Z flag
NDC	**ndc rc, ra, rb**	NAND the content of regA to regB and store result in regC if carry flag is set. It modifies Z flag

NDZ	**ndz rc, ra, rb**	NAND the content of regB to regA and store result in regC if zero flag is set. It modifies Z flag
NCU	**ncu rc, ra, rb**	NAND the content of regA to Complement of regB and store result in regC. It modifies Z flag
NCC	**ncc rc, ra, rb**	NAND the content of regA to complement regB and store result in regC if carry flag is set. It modifies Z flag
NCZ	**ncz rc, ra, rb**	NAND the content of regA to complement of regB and store result in regC, if zero flag is set. It modifies Z flag
LLI	**lli ra, Imm**	Place 9 bits immediate into least significant 9 bits of register A (RA) and higher 7 bits are assigned to zero
LW	**lw ra, rb, Imm**	Load value from memory into reg A. Memory address is formed by adding immediate 6 bits (signed) with content of reg B. It modifies zero flag.
SW	**sw ra, rb, Imm**	Store value from reg A into memory. Memory address is formed by adding immediate 6 bits (signed) with content of reg B.
LM	**lm ra, Imm**	The **Load Multiple (LM)** instruction loads multiple registers based on the bitmask specified in the **immediate field**. Each bit in the immediate field corresponds to a register (**R7 to R0**, from left to right). If a bit is set, the corresponding register is loaded from memory. The **base memory address** is provided in **register A**, and data is loaded into the selected registers from **consecutive memory locations**. The first register (starting from R7 if its bit is set) is loaded from the address in **Reg RA**, and subsequent registers (if selected) are loaded from **incremented addresses**.
SM	**sm ra, Imm**	The **Store Multiple (SM)** instruction stores the values of selected registers into memory based on the **immediate field**. Each bit in the immediate field corresponds to a register (**R7 to R0**, from left to right). If a bit is set, the corresponding register is stored in memory. The **base memory address** is provided in **Register RA**, and the selected registers are stored in **order of R7 to R0**. The

		first register (starting from **R7**, if its bit is set) is stored at the address in **Reg RA**, and subsequent registers (if selected) are stored at **incremented consecutive memory locations**.
BEQ	**beq ra, rb, Imm**	If content of reg A and regB are the same, branch to PC+Imm, where PC is the address of beq instruction
BLT	**blt ra, rb, Imm**	If content of reg A is less than content of regB, then it branches to PC+Imm, where PC is the address of beq instruction
BLE	**ble ra, rb, Imm**	If content of reg A is less than or equal to the content of regB, then it branches to PC+Imm, where PC is the address of beq instruction
JAL	**jal ra, Imm**	Branch to the address PC+ Imm. Store PC+1 into regA, where PC is the address of the jalr instruction
JLR	**jlr ra, rb**	Branch to the address in regB. Store PC+1 into regA, where PC is the address of the jlr instruction
JRI	**jri ra, Imm**	Branch to memory location given by the RA + Imm