Indian Institute of Technology Bombay
Department of Electrical Engineering
EE-739: Processor Design

Project

Design a 5 stage pipelined processor, IITB-RISC, whose instruction set architecture is 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* is an 8-register, 16-bit computer system. It should follow the standard 5 stage pipelines (Instruction fetch, instruction decode & register read, execute, memory access, and write back). The architecture should be optimized for performance, i.e., should include hazard mitigation techniques. Hence, it should have forwarding and branch prediction technique.

Group: Group of FOUR

Submission deadline: 13th May 2021 (23:59 PM)

IITB-RISC 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* is an 8-register, 16-bit computer system. It has 8 general-purpose registers (R0 to R7). Register R7 is always stores Program Counter. All addresses are short word addresses (i.e. address 0 corresponds to the first two bytes of main memory, address 1 corresponds to the second two bytes of main memory, etc.). This architecture uses condition code register which has two flags Carry flag (*C*) and Zero flag (*Z*). The *IITB-RISC* 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. 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 B (RB) (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)

Instructions Encoding:

ADD:	00_01	RA	RB	RC	0	00
ADC:	00_01	RA	RB	RC	0	10
ADZ:	00_01	RA	RB	RC	0	01
ADL:	00_01	RA	RB	RC	0	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
LHI:	00_11	RA	9 bit Immediate			
LW:	01_00	RA	RB	6 bit Immediate		
SW:	01_01	RA	RB	6 bit Immediate		
LM:	11_00	RA	0 + 8 bits corresponding to Reg R0 to R7 (left to right)			
SM:	11_01	RA	0 + 8 bits corresponding to Reg R0 to R7 (left to right)			
LA:	11_10	RA	0_0000_0000			
SA:	11_11	RA	0_0000_0000			
BEQ:	10_00	RA	RB	6 bit Immediate		
JAL:	10_01	RA	9 bit Immediate offset			
JLR:	10_10	RA	RB	000_000		
JRI	10_11	RA	9 bit Immediate offset			

RA: Register A

RB: Register B

RC: Register C

Instruction Description

Mnemonic	Name & Format	Assembly	Action
ADD	ADD (R)	*add rc, ra, rb*	Add content of regB to regA and store result in regC. *It modifies C and Z flags*
ADC	Add if carry set (R)	*adc rc, ra, rb*	Add content of regB to regA and store result in regC, if carry flaf is set. *It modifies C & Z flags*
ADZ	Add if zero set (R)	*adz rc, ra, rb*	Add content of regB to regA and store result in regC, if zero flag is set. *It modifies C & Z flags*
ADL	Add with one bit left shift of RB (R)	*Adl rc,ra,rb*	Add content of regB (after one bit left shift) to regA and store result in regC *It modifies C & Z flags*
ADI	Add immediate (I)	*adi rb, ra, imm6*	Add content of regA with Imm (sign extended) and store result in regB. *It modifies C and Z flags*
NDU	Nand (R)	*ndu rc, ra, rb*	NAND the content of regB to regA and store result in regC. *It modifies Z flag*
NDC	Nand if carry set (R)	*ndc rc, ra, rb*	NAND the content of regB to regA and store result in regC if carry flag is set. *It modifies Z flag*

NDZ	Nand if zero set (R)	*ndc rc, ra, rb*	NAND the content of regB to regA and store result in regC if zero flag is set. *It modifies Z flag*
LHI	Load higher immediate (J)	*lhi ra, Imm*	Place 9 bits immediate into most significant 9 bits of register A (RA) and lower 7 bits are assigned to zero.
LW	Load (I)	*lw ra, rb, Imm*	Load value from memory into reg A. Memory address is formed by adding immediate 6 bits with content of red B. *It modifies zero flag.*
SW	Store (I)	*sw ra, rb, Imm*	Store value from reg A into memory. Memory address is formed by adding immediate 6 bits with content of red B.
LM	Load multiple (J)	*lw ra, Imm*	Load multiple registers whose address is given in the immediate field (one bit per register, R0 to R7) in order from left to right, i.e, registers from R0 to R7 if corresponding bit is set. Memory address is given in reg A. Registers are loaded from consecutive addresses.
SM	Store multiple (J)	sm, ra, Imm	Store multiple registers whose address is given in the immediate field (one bit per register, R0 to R7) in order from left to right, i.e, registers from R0 to R7 if corresponding bit is set. Memory address is given in reg A. Registers are stored to consecutive addresses.
LA	Load all (J)	*la ra*	Load value from successive memory locations into registers R0 to R6. Starting memory address is given by RA.
SA	Store all (J)	*sa ra*	Store values from registers R0 to R6 to successive memory locations starting from address given in RA

BEQ	Branch on Equality (I)	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
JAL	Jump and Link (I)	jalr ra, Imm	Branch to the address PC+ Imm. Store PC+1 into regA, where PC is the address of the jalr instruction
JLR	Jump and Link to Register (I)	jalr ra, rb	Branch to the address in regB. Store PC+1 into regA, where PC is the address of the jalr instruction
JRI	Jump to register (J)	jri ra, Imm	Branch to memory location given by the RA + Imm