Indian Institute of Technology Bombay
Department of Electrical Engineering
EE-309: Microprocessors

Project

Design a 6-stage pipelined processor, *IITB-RISC-23*, 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-23* is a 16-bit computer system with 8 registers. It should follow the standard 6 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 implemented forwarding mechanism. Implementation of branch predictor is optional.

Group: Group of FOUR

Submission deadline: 3rd May 2023 (Wednesday) 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-23* is an 8-register, 16-bit computer system. It has 8 general-purpose registers (R0 to R7). Register R0 is 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. 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)

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		

val of flag or what?

1011

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

RA: Register A

RB: Register B

RC: Register C

zero flag is used to check the result of an arithmetic operation, including bitwise logical instructions. It is set to 1, or true, if an arithmetic result is zero, and reset otherwise.

Instruction Description

Mnemonic	Name & Format	Assembly	Action
ADA	ADD (R)	*ada 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*
AWC	Add with carry (R)	*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	ADD (R)	*aca rc, ra, rb*	Add content of regA to complement of regA and store result in regC. *It modifies C and Z flags*
ACC	Add if carry set (R)	*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	Add if zero set (R)	*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	Add with carry (R)	*acw rc,ra,rb*	Add content of regA to Complement of regB and Carry and store result in regC regC = regA + compement of regB + Carry *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 regA to regB and store result in regC. *It modifies Z flag*
NDC	Nand if carry set (R)	*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	Nand if zero set (R)	*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	Nand (R)	*ncu rc, ra, rb*	NAND the content of regA to Complement of regB and store result in regC. *It modifies Z flag*
NCC	Nand if carry set (R)	*ncc rc, ra, rb*	NAND the content of regA to complement of regB and store result in regC if carry flag is set. *It modifies Z flag*
NCZ	Nand if zero set (R)	*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	Load lower immediate (J)	*lli ra, Imm*	Place 9 bits immediate into leat significant 9 bits of register A (RA) and higher 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 (signed) 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 (signed) 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 from left to right) in reverse order from right to left, i.e, registers from R7 to R0 if corresponding bit is set. Memory address is given in reg A. Registers which are expected to be loaded from consecutive memory 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 from left to right) in reverse order from right to left, i.e, registers from R7 to R0 if corresponding bit is set. Memory address is given in reg A. Registers which are expected to store must be stored to consecutive addresses.
BEQ	Branch on Equality (I)	beq ra, rb, Imm	If content of reg A and regB are the same, branch to PC+Imm*2, where PC is the address of beq instruction
BLT	Branch on Less Than (I)	blt ra, rb, Imm	If content of reg A is less than content of regB, then it branches to PC+Imm*2, where PC is the address of beq instruction
BLE	Branch on Less or Equal (I)	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*2, where PC is the address of beq instruction
JAL	Jump and Link (J)	jalr ra, Imm	Branch to the address PC+ Imm*2. Store PC+2 into regA, where PC is the address of the jalr instruction

JLR	Jump and Link to Register (I)	jlr ra, rb	Branch to the address in regB. Store PC+2 into regA, where PC is the address of the jlr instruction
JRI	Jump to register (J)	jri ra, Imm	Branch to memory location given by the RA + Imm*2