EE309: IITB-RISC

Akshay Kaushal , Raushan Kumar , Monu Kumar Yadav

May 10, 2022

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

Project 2 (Bonus)

Design a 6 stage pipelined processor, IITB-RISC-22, 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-22* 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 forwarding and branch prediction technique.

Group: Group of FOUR

Submission deadline: 14th April 2021 (Thursday) 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 17 instructions. They are illustrated in the figure below.

R Type Instruction format

Opcode	Register A (RA)	Register B (RB)	Register B (RB)	Unused	Condition (CZ)
(4 bit)	(3 bit)	(3-bit)	(3-bit)	(1 bit)	(2 bit)

I Type Instruction format

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

J Type Instruction format

Opcode	Register A (RA)	Immediate
(4 bit)	(3 bit)	(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_00	RA	9 bit Immediate			
LW:	01_11	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 (right to left)			
SM:	11_01	RA	0 + 8 bits corresponding to Reg R0 to R7 (right to left)			
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

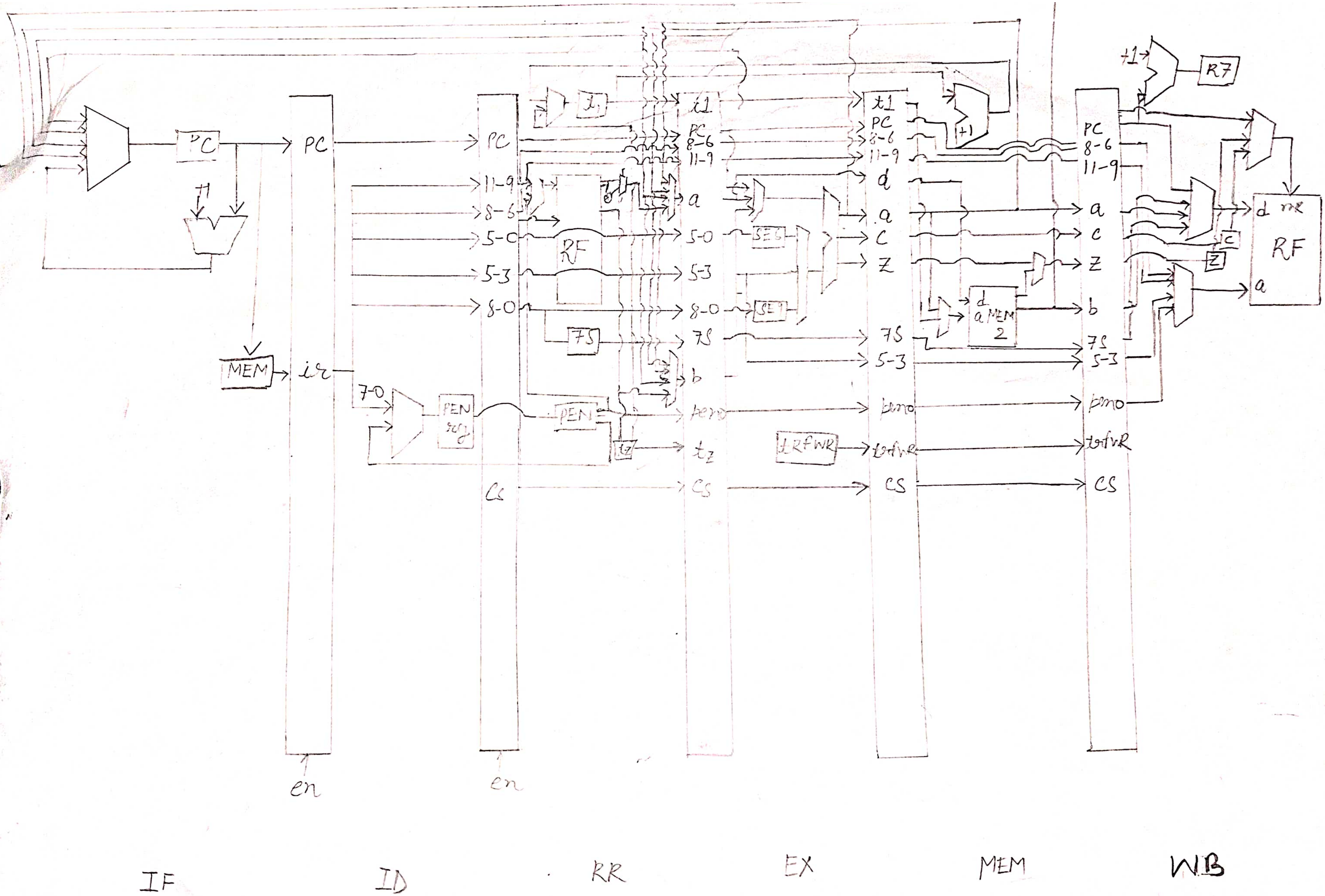
RC: Register C

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 right to left, 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 right to left, 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.
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



DATAPATH

ALL REGISTER WITH THEIR BITS

1.IF_ID→[32-0]

a).en→IF_ID[33]

b).PC→IF_ID[32-16]

c).IR→IF_ID[15-0]

2.ID_RR→[29-0]

a).en→ID_RR[29]

b).Cs→ID_RR[28]

c).IF_ID[32-16]→ID_RR[27-12]

d).IF_ID[11-0]→ID_RR[11-0]

3.RR_EX→[112-0]

a).en→RR_EX[112]

b).Cs→RR_EX[111]

c).PEN_0→RR_EX[110-108]

d)ID_RR[27-12]→RR_EX[107-92]

e)T1 →RR_EX[91-76]

f)A→RR_EX[75-60]

g)B→RR_EX[59-44]

h)T2→RR_EX[43-28]

i)7S→RR_EX[27-12]

j).ID_RR[11-0]→RR_EX[11-0]

4.EX_MEM→[99-0]

a).en→EX_MEM[99]

b).Cs→EX_MEM[98]

c).RR_EX[110-108]→EX_MEM[97-95]

d)TRFWR→EX_MEM[94]

e)T1 → RR_EX[93-78]

f)D→EX_MEM[77-62]

g)PC→EX_MEM[61-46] //PC FROM PREVIOUS REGISTOR

h)C→EX_MEM[45]

i)Z→EX_MEM[44]

j)7S→EX_MEM[43-28]

k)ALU_OUT→EX_MEM[27-12]

l).RR_EX[91-80] → EX_MEM[11-0]

4.MEM_WB→[83-0]

a).en→MEM_WB[83]

b).Cs→MEM_WB[82]

c).B→MEM_WB[81-66]

d)TRFWR→MEM_WB[65]

e)PC →MEM_WB[64-49]

f)7S→MEM_WB[48-33]

g)ALU_OUT→MEM_WB[32-17] //PC FROM PREVIOUS REGISTER

h)C→MEM_WB[16]

i)Z→MEM_WB[15]

j)PEN_0→MEM_WB[14-12]

k).RR_EX[91-80] → MEM_WB[11-0]

Instruction	IF	ID	RR	EX	MEM	WB
ADD	PC++→ IF_ID[32-16] MEM→IF_ID[15-0]	IF_ID[11-0]→ID_RR[11-0] IF_ID[32-16]→ID_RR[27-12] IF_ID[5-3] → ID_RR[5-3]	ID_RR[11-9]→RF_A1 ID_RR[8-6]→RF_A2 RF_D1 → A RF_D2 → B ID_RR[5-3]→RR_EX[5-3]	A → ALU_A B → ALU_B ALU_OUT→ EX_MEM[27-12] RR_EX[5-3]→ EX_MEM[5-3]	EX_MEM[27-12] →MEM_WB[32-17] EX_MEM[5-3] → MEM_WB[5-3]	MEM_WB[32-17] →RF_WD MEM_WB[5-3] →RF_A3
ADC	PC++→ IF_ID[32-16] MEM→IF_ID[15-0]	IF_ID[11-0]→ID_RR[11-0] IF_ID[32-16]→ID_RR[27-12] IF_ID[5-3] → ID_RR[5-3]	ID_RR[11-9]→RF_A1 ID_RR[8-6]→RF_A2 RF_D1 → A RF_D2 → B ID_RR[5-3]→RR_EX[5-3]	A → ALU_A B → ALU_B IF C==1; ALU_OUT→ EX_MEM[27-12] RR_EX[5-3]→ EX_MEM[5-3]	EX_MEM[27-12] →MEM_WB[32-17] EX_MEM[5-3] → MEM_WB[5-3]	MEM_WB[32-17] →RF_WD MEM_WB[5-3] →RF_A3
ADZ	PC++→ IF_ID[32-16] MEM→IF_ID[15-0]	IF_ID[11-0]→ID_RR[11-0] IF_ID[32-16]→ID_RR[27-12] IF_ID[5-3] → ID_RR[5-3]	ID_RR[11-9]→RF_A1 ID_RR[8-6]→RF_A2 RF_D1 → A RF_D2 → B ID_RR[5-3]→RR_EX[5-3]	A → ALU_A B → ALU_B IF Z==1; ALU_OUT→ EX_MEM[27-12] RR_EX[5-3]→ EX_MEM[5-3]	EX_MEM[27-12] →MEM_WB[32-17] EX_MEM[5-3] → MEM_WB[5-3]	MEM_WB[32-17] →RF_WD MEM_WB[5-3] →RF_A3

NDU	PC++→ IF_ID[32-16] MEM→IF_ID[15-0]	IF_ID[11-0]→ID_RR[11-0] IF_ID[32-16]→ID_RR[27-12] IF_ID[5-3] → ID_RR[5-3]	ID_RR[11-9]→RF_A1 ID_RR[8-6]→RF_A2 RF_D1 → A RF_D2 → B ID_RR[5-3]→RR_EX[5-3]	A → ALU_A B → ALU_B ALU_OUT→ EX_MEM[27-12] RR_EX[5-3]→ EX_MEM[5-3]	EX_MEM[27-12] →MEM_WB[32-17] EX_MEM[5-3] → MEM_WB[5-3]	MEM_WB[32-17] →RF_WD MEM_WB[5-3] →RF_A3
NDC	PC++→ IF_ID[32-16] MEM→IF_ID[15-0]	IF_ID[11-0]→ID_RR[11-0] IF_ID[32-16]→ID_RR[27-12] IF_ID[5-3] → ID_RR[5-3]	ID_RR[11-9]→RF_A1 ID_RR[8-6]→RF_A2 RF_D1 → A RF_D2 → B ID_RR[5-3]→RR_EX[5-3]	A → ALU_A B → ALU_B IF C==1; ALU_OUT→ EX_MEM[27-12] RR_EX[5-3]→ EX_MEM[5-3]	EX_MEM[27-12] →MEM_WB[32-17] EX_MEM[5-3] → MEM_WB[5-3]	MEM_WB[32-17] →RF_WD MEM_WB[5-3] →RF_A3
NDZ	PC++→ IF_ID[32-16] MEM→IF_ID[15-0]	IF_ID[11-0]→ID_RR[11-0] IF_ID[32-16]→ID_RR[27-12] IF_ID[5-3] → ID_RR[5-3]	ID_RR[11-9]→RF_A1 ID_RR[8-6]→RF_A2 RF_D1 → A RF_D2 → B ID_RR[5-3]→RR_EX[5-3]	A → ALU_A B → ALU_B IF Z==1; ALU_OUT→ EX_MEM[27-12] RR_EX[5-3]→ EX_MEM[5-3]	EX_MEM[27-12] →MEM_WB[32-17] EX_MEM[5-3] → MEM_WB[5-3]	MEM_WB[32-17] →RF_WD MEM_WB[5-3] →RF_A3
ADL	PC++→ IF_ID[32-16] MEM→IF_ID[15-0]	IF_ID[11-0]→ID_RR[11-0] IF_ID[32-16]→ID_RR[27-12] IF_ID[5-3] → ID_RR[5-3]	ID_RR[11-9]→RF_A1 ID_RR[8-6]→RF_A2 RF_D1 →S1→ A RF_D2 → B ID_RR[5-3]→RR_EX[5-3]	A → ALU_A B → ALU_B ALU_OUT→ EX_MEM[27-12] RR_EX[5-3]→ EX_MEM[5-3]	EX_MEM[27-12] →MEM_WB[32-17] EX_MEM[5-3] → MEM_WB[5-3]	MEM_WB[32-17] →RF_WD MEM_WB[5-3] →RF_A3
ADI	PC++→ IF_ID[32-16] MEM→IF_ID[15-0]	IF_ID[11-0]→ID_RR[11-0] IF_ID[32-16]→ID_RR[27-12] IF_ID[11-9] → ID_RR[11-9]	ID_RR[5-0]→RR_EX[5-0] ID_RR[8-6]→RF_A1 RF_D1 → A ID_RR[11-9]→RR_EX[11-9]	RR_EX[5-0]→SE6→ALU_B A→ALU_A ALU_OUT→EX_MEM[27-12] RR_EX[11-9]→ EX_MEM[11-9]	EX_MEM[27-12] →MEM_WB[32-17] RR_EX[11-9]→ EX_MEM[11-9]	MEM_WB[32-17] →RF_WD MEM_WB[11-9] →RF_A3
LW	PC++→ IF_ID[32-16] MEM→IF_ID[15-0]	IF_ID[11-0]→ID_RR[11-0] IF_ID[32-16]→ID_RR[27-12] IF_ID[11-9] → ID_RR[11-9]	ID_RR[5-0]→RR_EX[5-0] ID_RR[8-6]→RF_A1 RF_D1 → B ID_RR[11-9]→RR_EX[11-9]	RR_EX[5-0]→SE6→ALU_B B→ALU_A ALU_OUT→EX_MEM[27-12] RR_EX[11-9]→ EX_MEM[11-9]	EX_MEM[27-12] →ADDRM RR_EX[11-9]→ EX_MEM[11-9] RD → B	B →RF_WD MEM_WB[11-9] →RF_A3

SW	PC++→ IF_ID[32-16] MEM→IF_ID[15-0]	IF_ID[11-0]→ID_RR[11-0] IF_ID[32-16]→ID_RR[27-12] IF_ID[11-9] → ID_RR[11-9]	ID_RR[5-0]→RR_EX[5-0] ID_RR[8-6]→RF_A1 RF_D2 → A ID_RR[11-9]→RF_A2	RR_EX[5-0]→SE6→ALU_B B→ALU_A ALU_OUT→EX_MEM[27-12] A→D RR_EX[11-9]→ EX_MEM[11-9]	EX_MEM[27-12] →ADDRM RR_EX[11-9]→ EX_MEM[11-9] RD → B D→WD	B →RF_WD MEM_WB[11-9] →RF_A3
BEQ	PC++→ IF_ID[32-16] MEM→IF_ID[15-0]	IF_ID[11-9]→ID_RR[11-9] IF_ID[8-6]→ID_RR[8-6] IF_ID[32-16]→ID_RR[27-12] IF_ID[5-0]→ID_RR[5-0]	ID_RR[11-9]→RF_A1 ID_RR[11-9]→RF-A2 RF_A1→A RF_A2→B ID_RR[27-12]→RR_EX[107-92] ID_RR[5-0]→RR_EX[5-0]	A→ALU_A B→ALU_B ALU_OUT→EX_MEM[27-12] IF ALU_OUT==0; RR_EX[107-92]→ALU_A RR_EX[5-0]→SE6 SE6→ALU_B ALU_OUT→PC	NOP	NOP
JLR	PC++→ IF_ID[32-16] MEM→IF_ID[15-0]	IF_ID[11-9]→ID_RR[11-9] IF_ID[8-6]→ID_RR[8-6] IF_ID[32-16]→ID_RR[27-12] IF_ID[5-0]→ID_RR[5-0]==”000000”	ID_RR[11-9]→RR_EX[11-9] ID_RR[27-12]→RR_EX[107-92] ID_RR[8-6] →RF_A1 RF_D1 → T1 T1 → ALU3_A +1 → ALU3_B ALU3_OUT → PC	RR_EX[11-9] → EX_MEM[11-9] RR_EX[107-92]→ALU_A +1 → ALU_B ALU_OUT→EX_MEM[27-12]	EX_MEM[11-9] → MEM_WB[11-9] EX_MEM[27-12]→ MEM_WB[32-17]	MEM_WB[11-9] →RF_A3 MEM_WB[32-17]→RF_WD
LHI	PC++→ IF_ID[32-16] MEM→IF_ID[15-0]	IF_ID[8-0]→ID_RR[8-0] IF_ID[11-9]→ID_RR[11-9]	ID_RR[8-0]→RR_EX[8-0] ID_RR[11-9]→RR_EX[11-9]	RR_EX[8-0] → SE16→S7→ALU_OUT RR_EX[11-9] → EX_MEM[11-9]	ALU_OUT→MEM_WB[32-17] EX_MEM[11-9] → MEM_WB[11-9]	MEM_WB[32-17]→RF_WD MEM_WB[11-9] →RF_A3
JRI	PC++→ IF_ID[32-16] MEM→IF_ID[15-0	IF_ID[11-9]→ID_RR[11-9] F_ID[8-0]→ID_RR[8-0]	ID_RR[11-9]→RF_A1 ID_RR[8-0]→RR_EX[8-0]	RF_D1 → A , A → ALU_A RR_EX[8-0] →SE9→ALU_B ALU_OUT → EX_MEM[27-12] ALU_OUT → PC	NOP	NOP
LM	PC++→ IF_ID[32-16] MEM→IF_ID[15-0]	IF_ID[11-9]→ID_RR[11-9] IF_ID[7-0]→T0→PEN REG	WHILE(PEN_1!=0): PEN_REG→PEN PEN_O → RR_EX[110-108] PEN_1 →RR_EX[11-9] ID_RR[11-9]→RF_A2→A	RR_EX[110-108]→ EX_MEM[97-95] RR_EX[11-9]→ EX_MEM[11-9] A →D	EX_MEM[97-95] → MEM_WB[14-12] EX_MEM[11-9] → MEM_WB[11-9] D → ADDRM	MEM_WB[14-12]→RF_A3
SM	PC++→ IF_ID[32-16] MEM→IF_ID[15-0]	IF_ID[7-0]→T0→PEN REG IF_ID[11-9]→ID_RR[11-9]	WHILE(PEN_1!=0): ID_RR[11-9]→RF_A2→A PEN_O →RF_A1 RF_A1 → RF_D1 → A	RR_EX[110-108]→ EX_MEM[97-95] RR_EX[11-9]→ EX_MEM[11-9] B →D	EX_MEM[97-95] → MEM_WB[14-12] EX_MEM[11-9] → MEM_WB[11-9] D → ADDRM B →D	MEM_WB[14-12]→RF_A3

Hurdles in instruction pipeline There are three types of hazards:

1. Structural hazard:- if there are resorces conflict between two instruction in the same cycle

2. Data hazard:- if there are data dependencies i.e one instruction needs data which is yet to be produce by another instruction in the same cycle

3.Control hazard:-decision about the given instruction need more cycle to take any action.This hazard come because of branch and jump instruction

Procedure of removing structural hazards:

1. Separating instruction and data memories

2. Implementing separate adders for PC increment and offset addition

3. Each instruction uses ALU in at most 1 cycle

4. One instruction can read from Rf while other can write into Rf in the same cycle

5. We can delay next instructions for some cycle so that the same device cannot conflict with two instruction in same cycle. To ensure this, we can use data forwarding.

Procedure of removing data hazards:

1. To determine the instruction that has data dependencies in the same cycle, we can introduce NOP instruction(bubbles-it will delay for some time and do nothing) in the pipeline datapath.

2. We can delay next instructions for some cycle so that the same device cannot conflict with two instruction in same cycle. To ensure this, we can use data forwarding.

Procedure of removable of control hazards:

1. To determine the instruction that has data dependencies in the same cycle, we can introduce nope instruction(bubbles-it will delay for some time and do nothing) in the pipeline datapath.

2. We will flush in line instruction that we don't want to execute if branch or jump instruction will occurs.

3. To better do the control hazard, we use branch prediction technique.

Note: Hazard handling requires detection, stalling and flushing.