IITB-RISC-22

6 stage pipelined processor

Design Report

Electrical Engineering Department (EED)

Indian Institute of Technology Bombay

April 2022

A.Sathvik 213070070

Table of contents

Architecture Description **3**

Components in Data Path stagewise **9**

Instruction Fetch (IF) 9

Instruction Decoder (ID) 9

Register Read (RR) 9

Execution (EX) 10

Memory Write/Read (MM) 11

Write Back (WB) 11

Load (Load multiple/all) and Store (Store multiple/all) block: (LM_SM Block) 12

Inputs 12

Outputs 13

FSM Logic for Load(Load multiple or Load all): 13

FSM Logic for Store (store multiple or store all): 14

Control signals used for stagewise **14**

Instruction Fetch (IF) 14

Instruction Decode (ID) 14

Register Read (RR) 15

Execution (EX) 16

Data Memory Write/Read (MM) 16

Write Back (WB) 17

Hazards **17**

Timing Diagram **17**

Data Forwarding 18

Instruction-Wise Hazards and Mitigation 18

Arithmetic Instructions: 18

Jump and Branch Instructions: 19

Load Instructions: 19

Blocks for hazard Mitigation 19

Stalling Block: 19

EX Stage Block: 19

MM Stage Block: 20

Hazard and Condition Control Block 20

RTL VIEW **21**

Testbench **22**

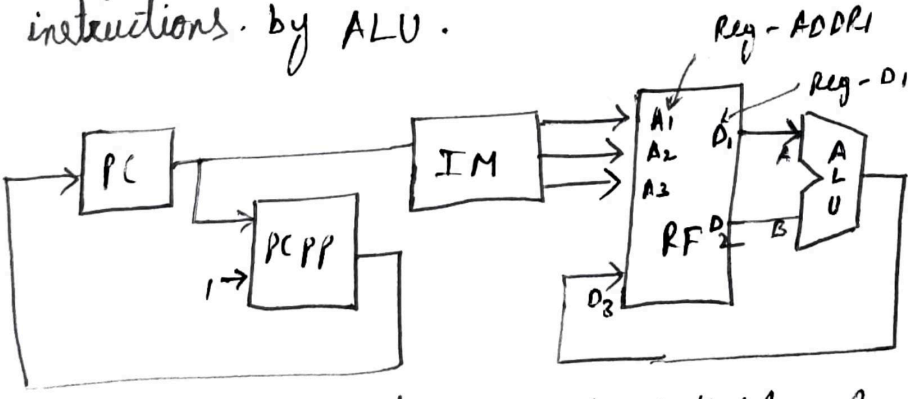
Simulation results **23**

Architecture Description

R-type instruction (ADD, ADC, ADZ, NDU, NDC, ADL, NDZ)

PC → PCPP-A, IM
+1 → PCPP-B
PCPP → RC
IM - D_{11-9} → Reg - ADDR1
IM - D_{8-6} → Reg - ADDR2
Reg - D1 → ALU - A
Reg - D2 → ALU - B
ALU → Reg - D3
IM - D_{5-3} → Reg - ADDR3

* For NAND, Addition control of ALU changed accordingly
* carry and zero flags are set in this type of instructions. by ALU.

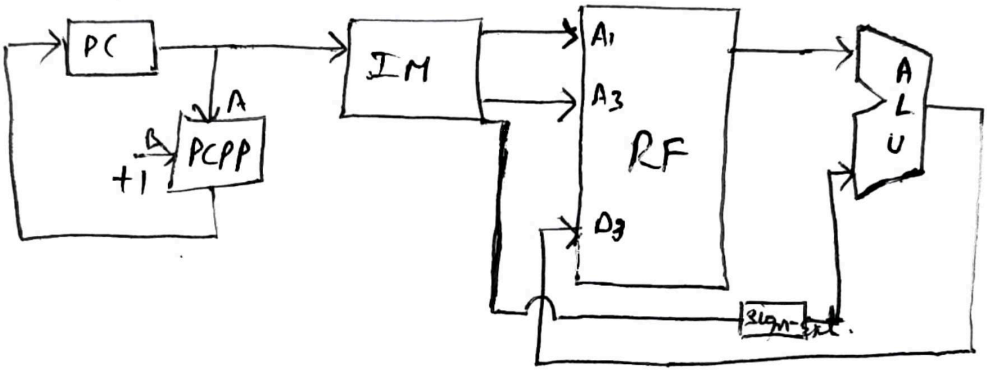


* For ADL instruction we left shift the Reg - D2 and apply to ALU-B and remaining things are same as usual as above instruction. of type Addition
* For conditional ADD and NAND we do the computation using opcode and CZ flag (condition) at decoder stage.
* in execution stage if we find carry or zero flag is set & not and then based on this flag condition we flush pipeline and take in next instruction.

ADI

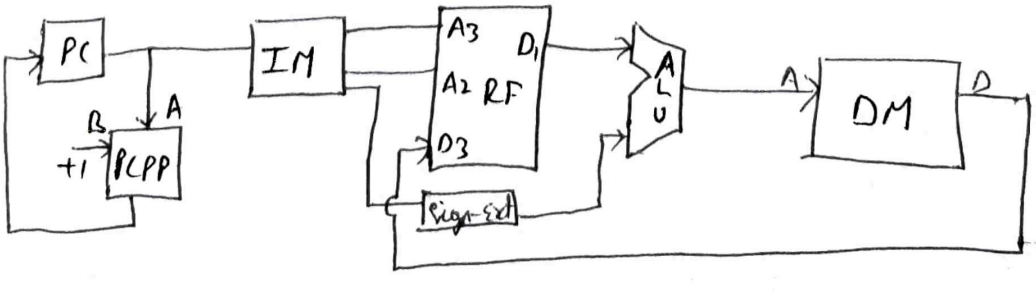
PC → PCPP-A , IM
+1 → PCPP-B
PCPP → PC
IM-D_{11-9} → Reg-ADDR1
IM-D_{8-6} → Reg-ADDR3
IM-D_{5-0} → Sign-ext → ALU-B
Reg-D_1 → ALU-A
ALU → Reg-D_3.

* carry and zero flags updated here aswell.



LW

PC → PCPP-A , IM
+1 → PCPP-B
PCPP → PC
IM-D_{11-9} → Reg-$ADDR_3$
IM-D_{8-6} → Reg-$ADDR_1$
Reg-D_1 → ALU-A
~~MEM~~ IM-D_{5-0} → Sign-ext → ALU-B
ALU → DM-A
Reg+D_1 → DM-D
DM-D → Reg-D_3



Zero flag Modified here

SW ⇒

PC → PCPP-A, IM

+1 → PCPP-B

PCPP → PC

IM - D_{11-9} → Reg - ADDR1

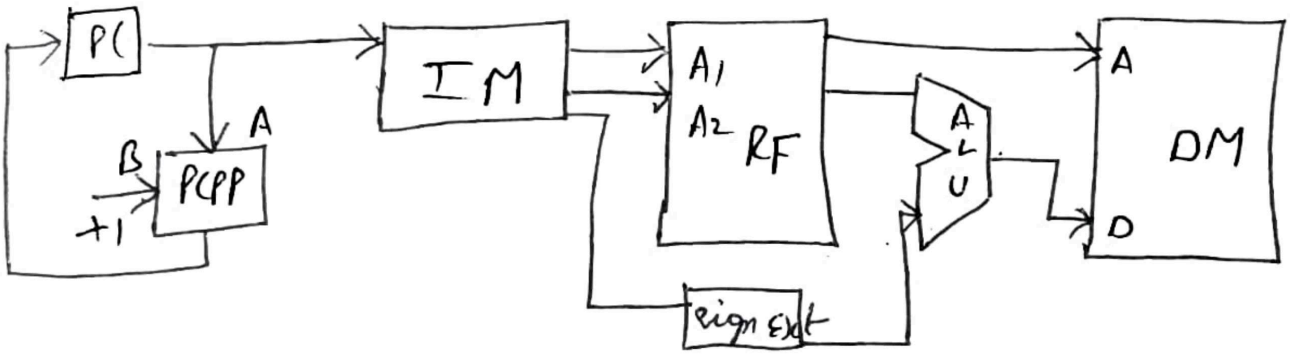
IM - D_{8-6} → Reg - ADDR2

Reg - D_2 → ALU - A

IM - D_{5-0} → sign-ext (SE) → ALU - B

ALU → DM - A

Reg - D_1 → DM - D.



JAL

PC → PCPP-A, IM, Reg ← PC

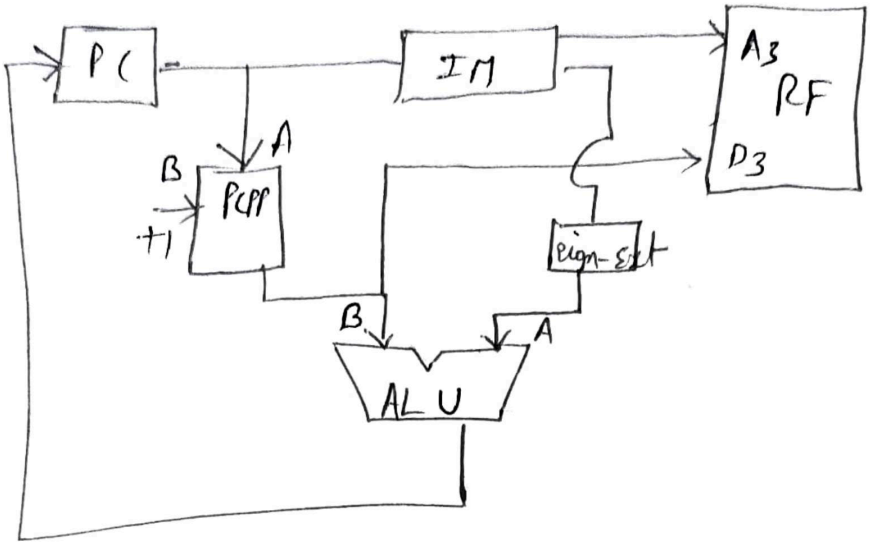
+1 → PCPP-B

PCPP → Reg - D_3, ALU - B

IM - D_{11-9} → Reg - ADDR3

IM - D_{8-0} → sign-ext (SE) → ALU - A

ALU → PC.



JLR:

PC → IM , PCPP-A

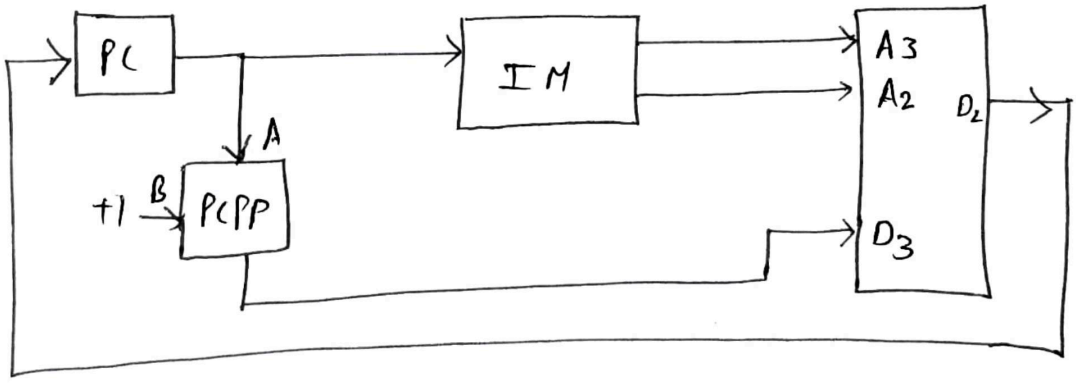
IM-D11-9 → Reg-ADDR3

IM-D8-6 → Reg-ADDR2

Reg-D2 → PC

+1 → PCPP-B

PCPP → Reg-D3



JRI:

PC → IM , PCPP-A

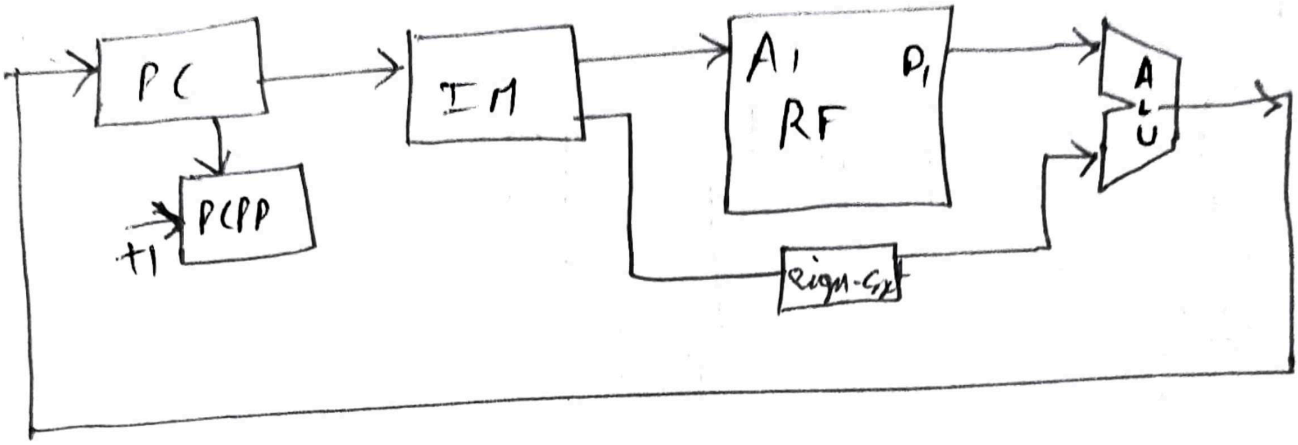
+1 → PCPP-B

IM-D11-9 → Reg-ADDR1

Reg-D1 → ALU-A

IM-D8-0 → ~~ALU-B~~ Sign-Ext → ALU-B

ALU → PC



BEQ:

$PC \rightarrow PCPP\text{-}A\ ,\ IM\ ,\ ALU_2\text{-}A$

$+1 \rightarrow PCPP\text{-}B$

$IM\text{-}D_{11-9} \rightarrow Reg\text{-}ADDR1$

$IM\text{-}D_{8-6} \rightarrow Reg\text{-}ADDR_2$

$Reg\text{-}D_1 \rightarrow ALU_1\text{-}A$

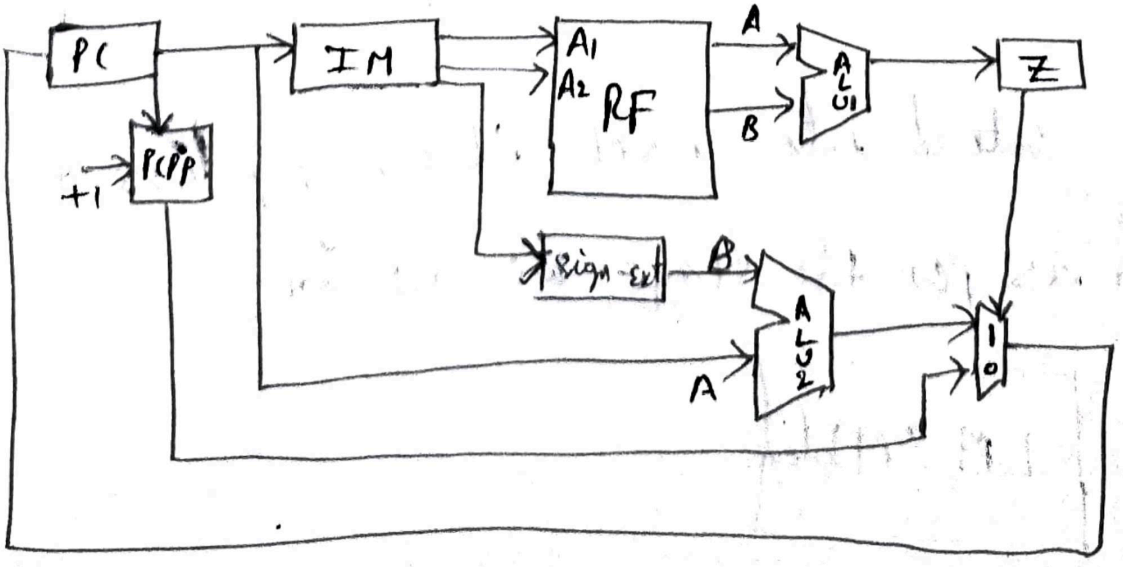
$Reg\text{-}D_2 \rightarrow ALU_1\text{-}B$

~~PCPP → ALU2 → A~~

$IM\text{-}D_{5-0} \rightarrow Sign\text{-}ext \rightarrow ALU_2\text{-}B$

if $ALU_1 - Z = 1$ then $ALU_2 \rightarrow PC$

else $PCPP \rightarrow PC$



LHI :

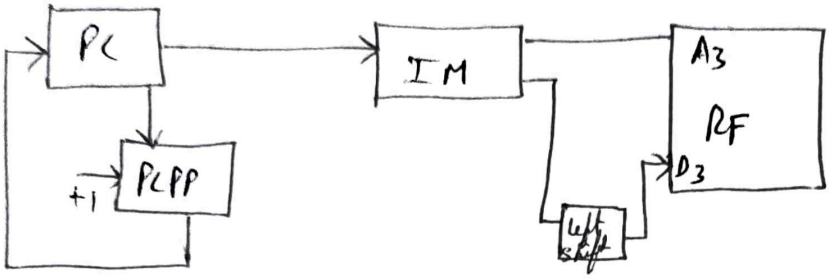
PC → PCPP-A , IM

+1 → PCPP

PCPP → PC

IM − D_{11-9} → Reg - ADDR 3

IM − D_{8-0} → Left shift → Reg - D_3



LM, SM, LA and SA

PC → PCPP-A , IM

+1 → PCPP-B

PCPP → PC

IM − D_{11-9} → RF-A_1 (Reg_ADDR1)

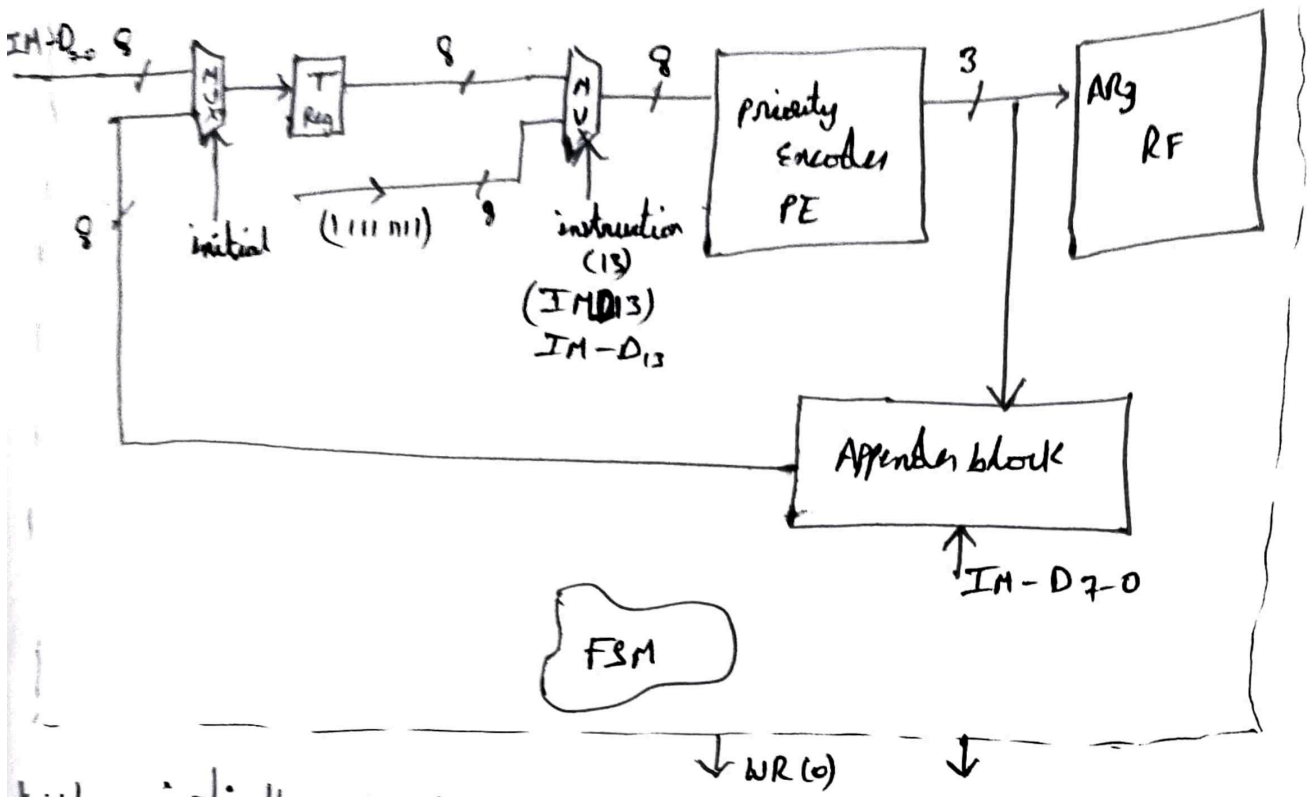
IM − D_{7-0} → LMSM Block

LMSM Block → WR(0), disable

* if opcode is of (LM or LA) and (SM or SA) then the control signals are defined by using FSM based.

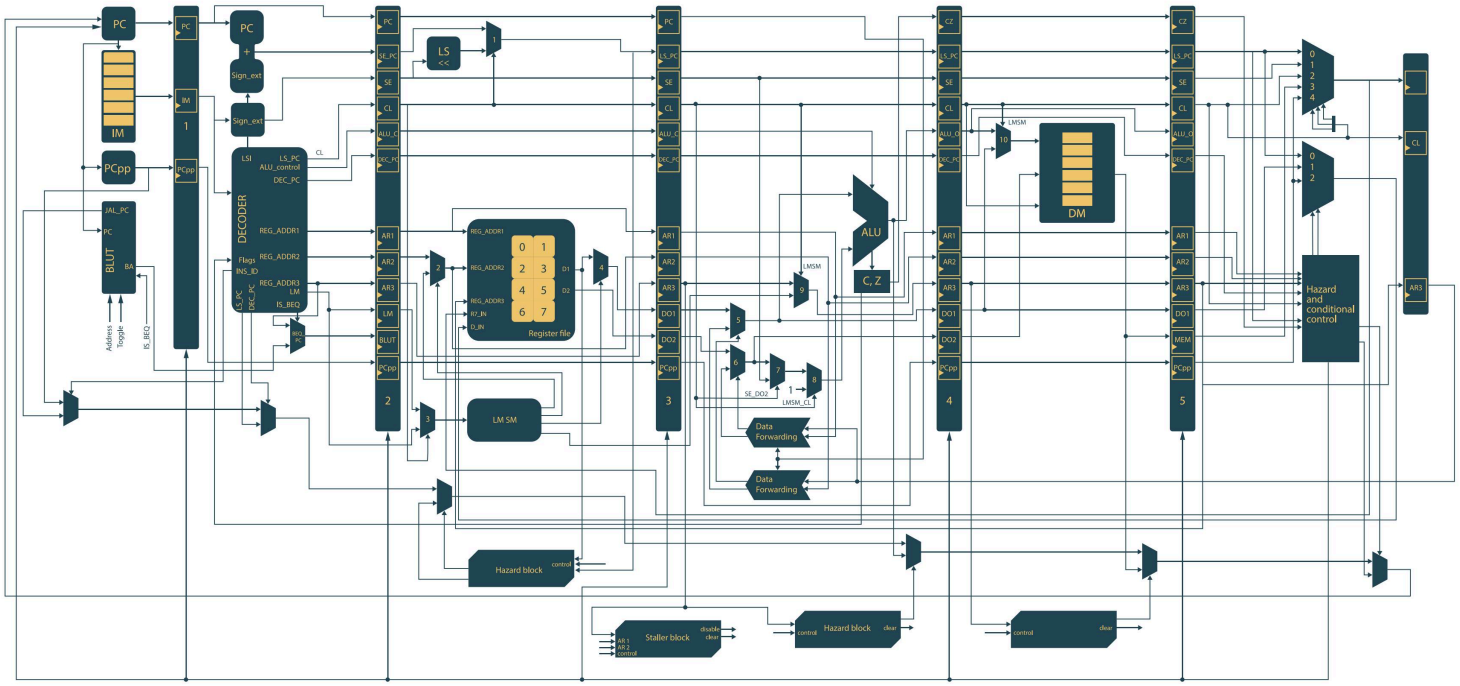
* if we are entered into FSM then, pipelines are disabled as per the stage we are in.

LMSM Block



* When initially in the T-Reg (Temperary Reg) it stores the $IM-D_{7-0}$. later it will stores appender block output of 8 bit data.
* When $IM-D_{13}$ is = 0 then it performs LM, SM instructions execution based on $IM-D_{7-0}$ data. When $IM-D_{13}$ is = 1 then it performs LA, SA instructions execution and priority Encoder input at this instant = 1111.1111.
* priority Encoder selects the highest priority bit and encodes it to 3 bit vector.
* Based on this priority Encoder output bits RF-ARB (Reg-ADDRB) is selected..
* Appender block will set the corresponding bit of at Encoded value to "0" in the data that is available at that time in Appender block.
* Appender Block take initially data as $IM-D_{7-0}$.

Datapath for all instructions along with control signals



Pipeline components stage-wise description

IF ID		ID RR		RR EX		EX MM		MM WB	
Components	Length(bits)	Components	Length(bits)	Components	Length(bits)	Components	Length(bits)	Components	Length(bits)
PC	16	PC	16	PC	16	LSPC	16	LSPC	16
Instruction	16	SEPC	16	LS_PC	16	Sign_ext(SE)	16	Sign_ext(SE)	16
PCpp	16	Sign_ext(SE)	16	Sign_ext (SE)	16	CL(Control Lines)	8	CL(Control Lines)	7
		CL (Control Lines)	11	CL (Control Lines)	12	• Beq	1	• Beq	1
		• LS_PC	1	• Beq	1	• Write_back_mux	3	• Write_back_mux	3
		• Beq	1	• LW	1	• Valid	3	• Valid	3
		• LM	1	• Sign_d2	1	• LM_SM_control	1	Flag_Control	3
		• LW	1	• Write_back_mux	3	Flag _Control	3	CZ	2
		• Sign_d2	1	• Valid	3	CZ	2	Write	2
		• Write_back_mux	3	• LM_SM control	3	Write	2	Reg_ADDR1(AR1)	3
		• valid	3	ALU_Control	2	Reg_ADDR1(AR1)	3	Reg_ADDR2(AR2)	3
		Opcode	4	Flag_Control	3	Reg_ADDR2(AR2)	3	Reg_ADDR3(AR3)	3
		ALU_Control	2	CZ	2	Reg_ADDR3(AR3)	3	ALU_output	16
		Flag_Control	3	Write	2	Flags	3	Memory_output	16
		Condition bits (CZ)	2	BLUT(Branch Lookup		Reg_D1(DO1)	16	Reg_D1(DO1)	16
		Write Bits	2	Table)	4	Reg_D2(DO2)	16	PCpp	16
		Reg_ADDR1(AR1)	3	Reg_D1(DO1)	16	ALU_output	16		
		Reg_ADDR2(AR2)	3	Reg_D2(DO2)	16				
		Reg_ADDR3(AR3)	3	Reg_ADDR1(AR1)	3				
		PCpp	16	Reg_ADDR2(AR2)	3				
		LM input	8	Reg_ADDR3(AR3)	3				
		BLUT(Branch LUT)	4	PCpp	16				

Components in Data Path stagewise

Instruction Fetch (IF)

1. Program Counter(PC) : Stores the current instruction PC value.
2. PCpp : It increments the PC value by 1 => PC+1.
3. IM : Instruction Memory in which the instructions are stored.
4. MUX : one input is address given by the branch prediction table and another input is PCpp output .The selection signal is also generated by the branch prediction table block itself
5. Brach Prediction Table :Store the already executed Beq instruction Program counter (PCs) and the branch address along with one more recent history of the instruction (whether it was taken or not)

Instruction Decoder (ID)

1. Sign_ext (SE) : It sign extends the 6-bit or 9-bit immediate data to 16 bit.
2. Decoder : It gives the control signals according to the current instruction type.
3. Sign_ext+PC : It adds the Sign_ext value and the PC value of the current instruction.
4. MUX : one input is PC+1 and other input is (Sign_ext+PC) value.

Register Read (RR)

1. Register_file (RF) : contains the registers (R0-R7).
2. Left_shift (LS) : Left shifts the given input value or left shifts the given sign extended value.
3. MUX1 : One input is (Sign_ext+PC) and other input is Left_shift value.
4. MUX2 : The mux before Reg_ADDR2(AR2). For this mux one input is AR2 from IDRR register and the other input is output from LM_SM block.

5. MUX3 : The mux after Reg_D1(DO1). For this mux one input is DO1 and the other input is ALU output and selection signal is produced by LM_SM block.

6. Hazard_Mux : Controlled by Hazard_RR block.

7. Hazard_RR : The Hazard Block present in RR stage.Sends Input to the PC register according to the instruction.

    a. SE_LS during LHI and R7 is the destination register.

    b. Reg_D1(DO1) during the JLR instruction act as PC.

8. LM_SM block : It handles the LM and SM operation for LM and SM instructions .

9. LM_SM mux : Choses the input for the LM_SM block between the data after decoder block and from the ID_RR pipeline.

Execution (EX)

1. Flags : Contains the carry, zeros.

2. ALU : Perform Addition, NAND and Comparison operation.

3. MUX1 : The second mux before ALU second input. It steers inputs according to the arithmetic operation or address calculation.

4. MUX2 : The mux just before the second input of ALU. Controlled by the LM_SM block. Selects 1 during the LM or SM operation.

5. Hazard_Mux : Mux controlled by the hazard logic block in execution stage.

6. Staller : When there is an immediate dependency after the load instruction. It stalls the pipeline registers

7. Forwarding Blocks : Checks if dependency present between RR_EX pipeline stage and the pipeline registers in the further stages. It controls the forward logic muxes accordingly.

8. Hazard_EX : When the destination is R7 during arithmetic instruction it controls the hazard. It loads the PC with the required value by controlling the hazard mux and flushes the pipeline as well.

Memory Write/Read (MM)

1. DM : Data memory from which the instruction writes data or read data.
2. Hazard_MUX : Mux which is being controlled by the hazard block in this stage(MM).
3. Hazard_MM : When the destination is R7 then checks the hazard during load instruction. By controlling the hazard mux it suitably loads the PC value.
4. MUX1 : Mux before address of data memory. Selects inputs between ALU output (address calculation) or Reg_D1(DO1) during LM or SM operation.

Write Back (WB)

1. Flags_user : The flags which are visible to user.
2. WB_mux : It chooses the input to be written in the register file. The register file inputs according to the instructions are
    i. Mem_out : During LW, LM instruction.
    ii. LS_PC : During LHI instruction.
    iii. ALU output : During arithmetic instructions.
    iv. PC+1 : During JAL, JLR instruction.
3. R7_mux : Chooses the input for R7 in the register file. R7 inputs according to the instructions are
    i. LS_PC : For Beq instruction when the branch is taken.
    ii. PC+1 : Normal Program execution.
    iii. Reg_D1(DO1) : During JLR instruction.
4. Hazard and Conditional Control : Flushes the pipeline if any of the below conditions are true as well care must be taken for the following cases
    i. When the destination is R7 for JLR instruction.
    ii. When the destination is R7 for JAL instruction.

iii. When the destination is R7 for conditional arithmetic instruction.

iv. Dependency present in previous pipeline registers and Conditional arithmetic instruction is not taken.

v. For Beq instruction Check whether the branch is taken or not.

5. Hazard_Mux : Decides the input to PC register and controlled by the Hazard and conditional Control logic block.

i. (Sign_ext+PC) when branch is taken for Beq instruction.

ii. PC+1 when JLR and JAL hazard is seen.

iii. Otherwise same default value.

Load (Load multiple/all) and Store (Store multiple/all) block: (LM_SM Block)

In this LM_SM block we have a priority encoder(PE) which will take the inputs from 2:1 mux output. Here one of the input to 2:1 mux is 8-bit immediate data from the instruction and another input to 2:1 mux is 11111111 and the selection signal is taken from Instruction opcode => Instruction[13] .The LM_SM block is used to generate the control signals to run the Load and Store multiple instructions along with Load all and store all instructions. Inherently, the instruction is a multi-cycle instruction and thus has to be performed by halting (stalling) the pipeline.

NOTE : The LM_SM block starts giving the address one cycle after it is activated.

The LM_SM block has the following inputs and outputs:

Inputs

- Clk and reset
- 8-bit data (from the instruction to be fed to the one of the input to 2:1 mux)
    - Note that there is another mux connected to it, since the inputs (which are the same) for LM and SM will be taken in different clock cycles.
- LM bit and SM bit

- The LM and SM bits specifically tell when the instruction is of type load or store.These bits are also used to activate the LM_SM block, so that priority encoder can start giving the address of register to store the corresponding data from memory location specified by the instruction itself.

Outputs

- ALU2_mux: Selection signal for the mux9 connected to the second input of ALU, decides when +1 should be the input to the ALU
- AR3_mux: Used in case of Load(load multiple or load all), this mux9 controls the data that is stored in register AR3 in EX_MM
- AR2_mux: Used in case of Store(store all or store multiple), this mux2 controls the input to the register file(RF) for AR2
- Register Address: Reg_ADDR2(AR2) in case of STORE(Store multiple or Store all), Reg_ADDR3(AR3) in case of LOAD (load multiple or load all).
- Clear and disable signals for the pipeline registers.
- Reg_D1_mux(RF_DO1_mux) : Selection signal for the mux4 connected to the Reg_D1(DO1) register in RR_EX
- mem_in_mux: Selection signal for the mux10 connected to the input to the address of the memory which is usually the output of the ALU except in Load(Load multiple or Load all) or Store (store multiple or store all) where it is DO1

The LM_SM block has been built using an FSM which goes into different states, depending on whether the instruction is Load(Load multiple or Load all) or Store (store multiple or store all).

FSM Logic for Load(Load multiple or Load all):

- The block gets activated when the current instruction is in the RR stage. Here it is in the S1 state. The bits that control memory input, Reg_ADDR3(AR3), and ALU are '1' and are put through the pipeline register.
- It then moves to the S2 state, where the mux connected to input of DO1 now starts accepting the output of ALU (DO1 + 1), and the block starts giving Reg_ADDR3(AR3) addresses, which will be written in the write back(WB) stage into the register file.The disable signal is high, since the registers are now disabled (RR_EX, ID_RR, IF_ID and PC are disabled).
    - **Note:** There is a special enable signal to DO1 in RR_EX since that has to be enabled during the LM_SM process.
- The whole cycle continues till the last bit in the input of priority encoder(PE) goes to zero, when the valid signal goes low, and one instruction currently in the RR_EX register has to be disabled.The disable signal goes low as well, meaning the pipeline flow has started again.

FSM Logic for Store (store multiple or store all):

- The block gets activated when the current instruction is in the instruction decode stage(ID).
- In the next clock cycle, the LM_SM block starts giving the Reg_ADDR2(AR2) address to begin Store(Store multiple or store all). The control bits follow the same pattern as Load(load multiple or load all), except the mux controlling the input to DO1 starts accepting the ALU output one cycle later because now it's in Store(Store multiple or store all) stage.When the valid signal goes low, the last set of data is available in the RR_EX register.The disable signal goes low as well, meaning the pipeline flow has started again.

Control signals used for stagewise

Instruction Fetch (IF)

- PC is incremented by PCpp and sent to PC register.
- Instruction is fetched from Instruction Memory (IM).
- Branch Prediction Table (BPT) is used whenever it is required.

Instruction Decode (ID)

Generated Signals:

- Sign Extended value of the Immediate data in the instruction if it is needed.
- The condition bits which indicates the presence of conditional arithmetic instruction.
- Flag control bits which enables or disables the flag registers.
- The addition of the current PC value and the sign extended value (SE + PC).
- Register write and Memory write signals.
- The ALU control bits tells which operations has to be performed by the ALU.
- Control line for the mux which decides the PC+1 or (SE+PC) value has to be sent to the PC register. When decoder encounters with JAL instruction (SE+PC) value is sent to PC.
- Clear bit for to clear the pipeline register (IF-ID) when JAL instruction arrives.

- Multiplexes Control signals in the other pipeline stages.
- Control Signal for Sign_ext(SE) which decides the sign extend 6 bit or 9 bit immediate data.
- Address of the register file Reg_ADDR1(AR1) ,Reg_ADDR2(AR2) ,Reg_ADDR3(AR3).
- Control Signal for the Brach Prediction Table (BPT) telling us whether the instruction is Beq type or not.
- The LM or SM data to be given to the LM_SM block.This contains the data of the registers whose data has to be either stored or loaded.
- Branch Prediction Table produces the index of the instruction from the table and also information on whether the branch was taken or not

Register Read (RR)

Signals Consumption:

- The selection signal for the mux which selects inputs between (SE+PC) and the left shifted value of the sign extended immediate data.
- Reg_ADDR1(AR1) and Reg_ADDR2(AR2) data in the register file.
- LM_SM block creates the selection signal for the mux before Reg_ADDR2(AR2) which selects the input between the original Reg_ADDR2(AR2) and Reg_ADDR2(AR2) created by the LM_SM block.
- LM or SM bit which activates the LM_SM block and also the LM or SM data is consumed by the LM_SM block.
- LM_SM block also generates the Data signal for the mux which decides the original Reg_D1(DO1) value or the auto incremented value of Reg_D1(DO1) by the LM_SM block. This mux output data is sent to the register file (RF) at Reg_D1(DO1).
- RR_PC created by the hazard block in RR stage which controls the mux which decides the input to PC should be PC+1 or left shifted Sign_ext(SE) (when R7 is Reg_ADDR3(AR3) in LHI) or DO1 (when the instruction is JLR)

Generated Signals:

- Reg_D1(DO1) and Reg_D2(DO2) data created from the register file

Execution (EX)

Signals Consumption:

- The selection signal (SE_DO2) for the mux7 which selects input between Reg_D1(DO2) or the sign extended(Sign_ext) value. Reg_D1 is used for arithmetic operations whereas sign extended value is used for address calculation.
- The selection signal for the hazard mux
- 2 selection signals for the forwarding mux5 and mux6 created by the forwarding logic block.
- The selection signal for the mux created by LM_SM block to store the created address in the register file for storing data during LM or LA from the memory
- The selection signal (LM_SM_CL) for the mux8 which is created by the LM_SM block to select the output of mux7 or to select 1 such that the required address is always incremented by 1 during the LM or SM or SA or LA instruction.

Generated Signals:

- The ALU output data after ALU operation was performed.
- The flag register values after the ALU operation was performed.

Data Memory Write/Read (MM)

Signals Consumption:

- Memory Write control signal for the data memory (DM) during instructions SA or SW or SM.
- The selection signal for the hazard mux in the Memory Write stage.
- The selection signal for the mux created by the LM_SM block which selects the inputs between the ALU output or the address calculated by the LM_SM block.

Generated Signals:

- The data after reading from the data memory.

Write Back (WB)

Signals Consumption:

- The selection signal for the mux which decides the input to R7 in register file(RF).This is created by the hazard block at this stage.
- User flag registers must be written by Flag values.
- We need to sent Flag control bits, Condition bits and Control Signals to the hazard logic block in Write back (WB) stage.
- The selection signals for the mux which decides the data (ALU output, left shifted sign extended value, (SE+PC), SE, PC+1, Memory out data) to be written in the register file(RF) according to the instruction.

The temporary register holds the data for the forwarding logic.The data after WB mux, AR3 and some of the control signals (valid, the control signals for WB mux) is sent to a temporary register.

Hazards

Detection and Removal

Timing Diagram

Time	IF	ID	RR	EX	MM	WB
t0	1					
t1	2	1				
t2	3	2	1			
t3	4	3	2	1		
t4	5	4	3	2	1	
t5	6	5	4	3	2	1
t6	7	6	5	4	3	2

Data Forwarding

Data is forwarded through two multiplexers as shown in the below figure and priority for checking forward Logic => 1) R7 2) EX_MM 3) MM_WB 4) WB

Each pipeline register holds valid bits corresponding to the operand and destination register address of corresponding instruction. Forwarding block takes these valid bits to determine dependency among instructions.

Each mux (i.e mux 5 and mux 6) is controlled by a separate data-forwarding unit which checks if any of the operand in EX stage depends on current value of PC or output of any other instruction in further stages and then finally provides corresponding data to ALU input.

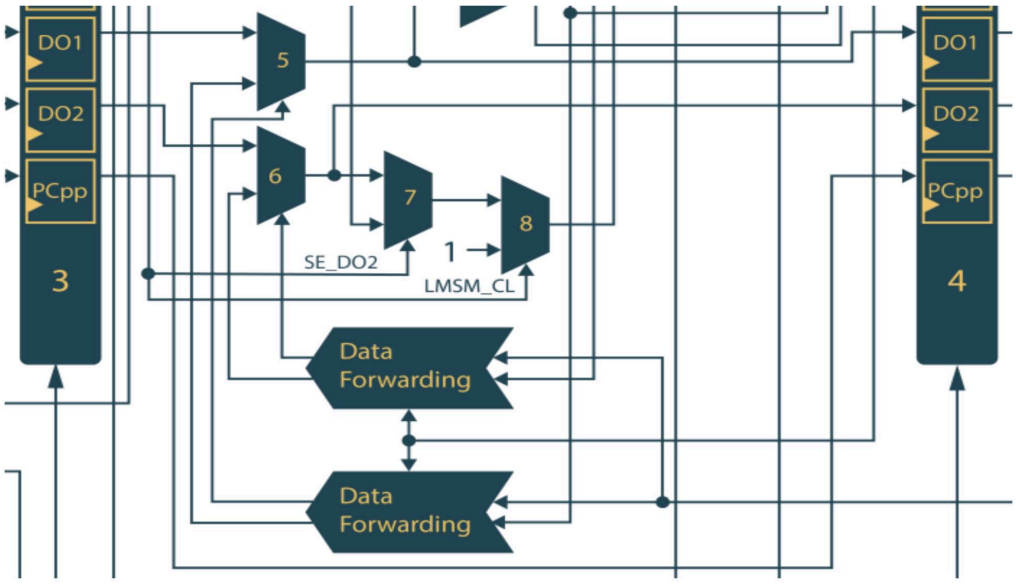


Fig 1: Indicates Data forwarding block

Instruction-Wise Hazards and Mitigation

Arithmetic Instructions:

- Arithmetic instructions not having R7 as destination
  - Only data forwarding is sufficient If instruction is unconditional.
  - Assume instruction to be taken and check instruction in writeback stage If instruction is conditional. If instruction is false and preceding instructions depend

on the current instruction, then flush the pipeline. Dependency can be found by Reg_ADDR1(AR1)/Reg_ADDR2(AR2) and Valid bits in pipeline registers.

- ❖ Arithmetic instructions having R7 as destination
    - ➢ Update PC in execution state and update R7 in writeback stage as well as flush the [IF-ID] and [ID-RR] registers If instruction is unconditional.
    - ➢ Assume instruction to be taken i.e. update PC in execution stage If instruction is conditional. If the instruction condition becomes false (checked in Writeback stage) then flush the pipeline.

Jump and Branch Instructions:

- ❖ JAL: Clear [IF-ID] register and update Program counter (PC) in instruction decode stage(ID) stage.
- ❖ JLR: Clear [IF-ID] and [ID-RR] register and update PC in RR stage.
- ❖ Beq: Beq instructions are assumed to be not-taken/taken based on the branch prediction table.
    - ➢ New entries values to the table are modified in the decode stage(ID).
    - ➢ Note that the condition is going to be checked in the WB stage.
    - ➢ If the condition becomes opposite to what was predicted then the table is accordingly modified (edited) and the program counter (PC) is updated and the pipeline is going to be flushed accordingly.
    - ➢ If it is not found in the branch prediction table then they are assumed to be not-taken by default as the address to the branch is unknown.

Load Instructions:

- ❖ Flush previous registers and update PC if instruction in MM stage is load type (LM,LA and LW) and destination register is R7.
- ❖ LW: If instruction in [RR-EX] is LW and instruction in [ID-RR] depends on its output then stall has to be performed.

Blocks for hazard Mitigation

Stalling Block:

- ❖ If instruction in [IF-ID] depends on its output and instruction in [ID-RR] is LW, then stall the instruction for one cycle.
- ❖ Delay SM_start bit through register and MUX if instruction in [IF-ID] is SM or SA.
- ❖ Clear enable of PC, [IF-ID], [ID-RR]. It will create 2 copies of LW instruction, one in [ID-RR] and one in [RR-EX]. So, pass the same signal through the register to clear LW instruction in the [ID-RR].

EX Stage Block:

- ❖ This block updates the PC in case of Arithmetic Instructions with R7 as destination.

- This block clears [ID-RR] and [IF-ID] registers and sends output to PC.

MM Stage Block:

- Update PC in MM stage if any load type instruction has R7 as destination.
- Clear [RR-EX] , [ID-RR] and [IF-ID] registers.

Hazard and Condition Control Block

1. Condition:

   Conditional Arithmetic Instruction not having R7 as output destination becomes false (Reg_ADDR3!=111) and preceding instructions depend on it. User Flags used for checking conditions for conditional arithmetic.

   Action:

   Clear register write signal and flush pipeline.

2. Condition:

   Conditional Arithmetic Instruction not having R7 as output destination becomes false.User Flags used for checking conditions for conditional arithmetic.

   Action:

   Write PC and R7 with PC+1. Clear register write signal and flush pipeline.

3. Condition:

   Beq condition becomes true. ALU Flags are used for checking conditions for Beq.

   Action:

   Write PC and R7 with (PC+SE) and Flush pipeline.

4. Condition:

   Jump and link to register R7 => (JLR R7 R7)

   Action:

   Write PC and R7 with PC+1 and Flush pipeline.

RTL View of 6 stage pipelined IITB RISC 2022

Testbench

Memory contents To test the instructions:

```
memory(0) <= 74;  - ADI R1 <- R0 + 10(DECIMAL)
memory(1) <= 1098 ;  - ADI R1 <- R2 +10
memory(2) <= 5712 ;   - ADD R2 <- R1 + R3
memory(3) <= 10448;   - NDU R2 <- R3 +R4
memory(4) <= 14879 ;    -LHI R5,31
memory(5) <= 23114 ;   - SW R5,R6,10
memory(6) <= 19466 ;     - LW R6,R0,10
memory(7) <= 6083 ;     - JRI R7,3
```

Testing load /store multiple instructions

```
memory(0)<= 74;          -ADI R0,R1,10
memory(1) <= 21003;     - SW R1,R0,10
memory(2) <= 21004;      -SW R2,R0,11
memory(3) <= 21005;       -SW R3,R0,12
memory(4) <= 21006;       -SW R4,R0,13
memory(5) <= 21007;       -SW R5,R0,14
memory(6) <= 0;           -ADI R0,R0,0
memory(7) <= 128;         -ADI R1,R0,0
memory(8) <= 192;         -ADI R2,R0,0
memory(9) <= 256;         -ADI R3,R0,0
memory(10) <=320;         -ADI R4,R0,0
memory(11) <= 384;        -ADI R5,R0,0
memory(12) <= 0;          -ADI R6,R0,0
memory(13) <= 49228;      -LM R0,10
memory(14) <= 26700;      -SM R0,10
```

Simulation results

For all instructions