EE309 - Microprocessors

IITB-RISC-23

DESIGN REPORT

A 6-stage pipelined processor , with given ISA has been designed and implemented in VHDL.The architecture is also optimized for performance including hazard mitigation techniques.

MADE BY:
Shivansh Gupta , 21d070067
Vasu Soni , 21d070087
Pulkit, 21d070052
Devesh Soni, 21d070025

The DATA-PATH

A separate pdf is attached for datapath.

DATA-PATH COMPONENTS

➔ Instruction Fetch

- ◆ PC: stored as register 0 in register file , stores the PC value of current instruction .
- ◆ IM: Instruction memory , takes PC input and gives the instruction as indicated by PC value
- ◆ ALU2: takes 16 bit input ,used to update PC to PC+2 .

➔ Instruction Decode

- ◆ Generates signal for Control signals for multiplexers in the pipeline stages.
- ◆ Makes decision like:
  - Sign Extended value of the immediate data in the instruction if present .
  - Control Signal for SE which decides if to sign extend 6 bit or 9 bit immediate data
  - The ALU control bits which tells the ALU of its operations
  - Flag control bits which enables or disables the flag registers
  - Clear bit for when JAL instruction arrives to clear the pipeline register (IF-ID)

➔ Register Read

- Register File: Contains 8 registers of 16 bits each , which stores data or memory location etc .
- Hazard_MUX1(blue): this mux is used on correction of immediate dependencies for operand addresses defined by bits 8-6. Inputs are all data forwarding paths .
- Hazard_MUX2(blue): this mux is used on correction of immediate dependencies for operand addresses defined by bits 5-3. Inputs are all data forwarding paths .
- LM_SM mux1: choses between new op-code and the opcode for LM-SM. This is done because the LM-SM state should be present for 8 cycles.
- LM_SM mux2: choses the input bits 8-0 , the immediate bits for pipeline register 3 .It has two inputs ,either usual Imm bits or shifted 8 bits .The control line for this mux are coming from Pipeline register 3 and implemented using if-else conditions in the code .
- 9 bit_Zero _extender: selects immediate bits from pipeline register 3 . This corrects the immediate dependencies in LLI instruction

➔ Execution

- ALU1: performs ADD,NAND , comparator and Complement operation . It also updates the flags such as carry , zero and overflow flags
- ALU1_mux_A: takes Ra and Rb values as immediate is fix to alu_mux_B input .
- ALU1_mux_B: takes constants, Rb , sign extended Imm6 bits and sign extended Imm9 bits.

- ALU1_out_MUX: used for LM-SM instruction.
- 6bit_SE: extends the imm6 bits of I type instruction
- 9bit_SE: extends the Imm9 bits of J type instruction
- 1bit_shifter:used in LM-SM stage to get the bit which is set and corresponding register in register file is selected
- 1_subtractor: used in LM-SM state to decrease additional three bits which we have stored in pipeline register 4 . These three bits indicate the rf_a3 value. These three bits will be 111 until instruction LM-SM is executed.
- Shifter mux: selects if direct 8 bits or shifted 8 bits are to be transferred.

➔ Memory Write/Read

- Data_memory: data memory from which the instruction reads data writes data.
- Mem_di_mux: takes input as data of registers in register file or the output of ALU1 and sends it to mem_di port .
- Mem_write_mux: takes usual input 1/0 or takes value of the shifted 8th bit from shifter .
- ALU5: used for LM-SM instruction , to update the memory address to consecutive memory addresses . This ALU takes input as ALU5's output.
- ALU5_B_mux: control bit is the 8th shifted bit from shifter

➔ Write Back

- ALU3: calculates PC+2*Imm for instructions like BEQ, BLU,BLT and JAL
- ALU3_B_mux: selects 6Imm bits or 9Imm bits corresponding
- ALU4: Updates pc , PC+2.
- 9_Zero extender: Zero extends 9Imm bits for J type instructions.
- 6_Sign Extender: Sign Extends 6Imm bits for I type instructions.
- 9_Sign Extender:Sign Extends 9Imm bits for J type instructions.
- Rf_a3_mux: have a separate input for LM-SM states.
- Rf_d3_mux: select values to be entered in register.
- Pc_in: 4 select line 2 are PC+2 and two from the write back stage.
- Rf_write_mux: have separate select line for LM-SM state.

PIPELINE REGISTERS:

Pipeline Register	Components	Length(bits)
PIPE1	Instruction	16 (15-0)
	PC	16 (31-16)
PIPE2	Instruction	16(15-0)
	PC	16(PC)
PIPE3	Instruction	16(15-0)
	PC	16(31-16)
	DATA A1	16(47-32)

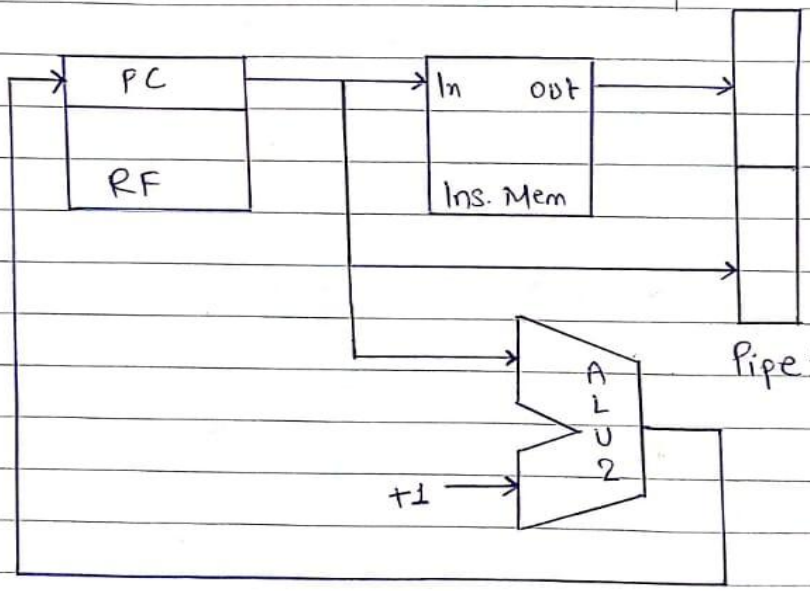
	DATA A2	16(63-48)
PIPE4	Instruction	16(15-0)
	PC	16(31-16)
	DATA A1	16(47-32)
	DATA A2	16(63-48)
	ALU OUT	16(79-64)
	CZ ALU OUT	2(81-80)
	RFA3 value	3(84-82)
PIPE5	Instruction	16(15-0)
	PC	16(31-16)
	DATA A1	16(47-32)
	DATA A2	16(63-48)
	ALU OUT	16(79-64)
	CZ ALU OUT	2(81-80)
	MEM DOUT	16(97-82)
	sig_1b_sub_in	3(100-98)

Individual State Diagrams:

for R type Instructions [Stage 1]

PC ⟶ Instr. Memory, Pipe1, ALU2-A
Instr. ⟶ Pipe1
+1 ⟶ ALU2-B
ALU2-C ⟶ PC_in

for remaining instructions also except for given below same process



for the write-back stage if we have

JAL then
ALU3-C ⟶ PC_in

for JLR
Data 1 (Pipe5) ⟶ PC_in

JRI
ALU1:OUT (Pipe5) ⟶ PC_in

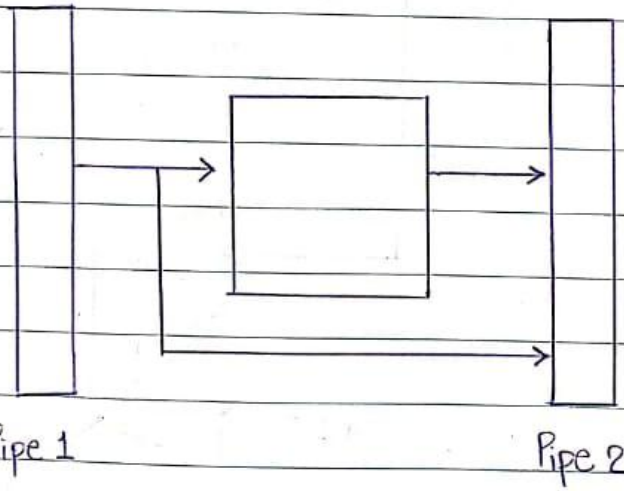
BEQ, BLE, BLT
if condition satisfied
ALU3-C ⟶ PC_in

Stage-2

for R type instructions

Pipe 1 ⟶ Pipe 2
Pipe 1 ⟶ Controls.

The above is same for I, J type.



Stage 3:

for R, I, J type:

ADA, ADC, ADZ, AWC, ACA, ACC, ACZ, ACW, NDU, NDC, NDZ, NCU, NCC, NCZ, LW, SW, BEQ, BLE, BLT, JLR, JRI

(1) Pipe2 [8-6] ⟶ RFA1

(2a) Pipe 2 [5-3] ⟶ RFA2

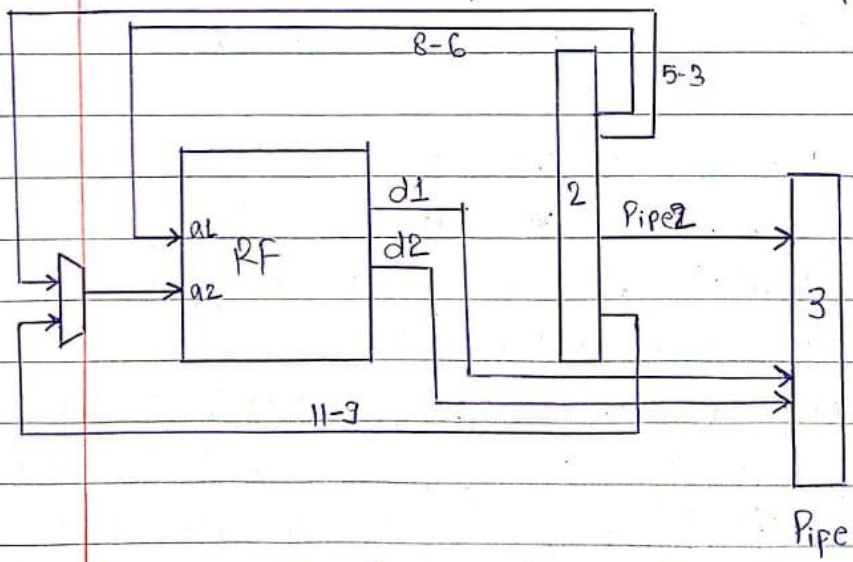
& Only for SW, BEQ, BLT, BLE, JRI

(2b) Pipe2 [11-9] ⟶ RFA2

(3) RFA1 ⟶ Pipe3

(4) RFA2 ⟶ Pipe3

} These are without dependency



* Here data-forwarding isn't shown it is shown later.
* for ADI we will be selecting RFA2 ← (5-3) but won't be used later.
* for LLI also the same

Stage-4

The operation for ALU1 is being decided by the instruction in Pipe-3.

for ADA, ADC, ADZ, AWC, ACA, ACC, ACZ, ACW
NDU, NDC, NDZ, NCU, NCC, NCZ, BEQ, BLE, BLT

Pipe 3 (data 1) ⟶ ALU1-A
Pipe 3 (data 2) ⟶ ALU1-B

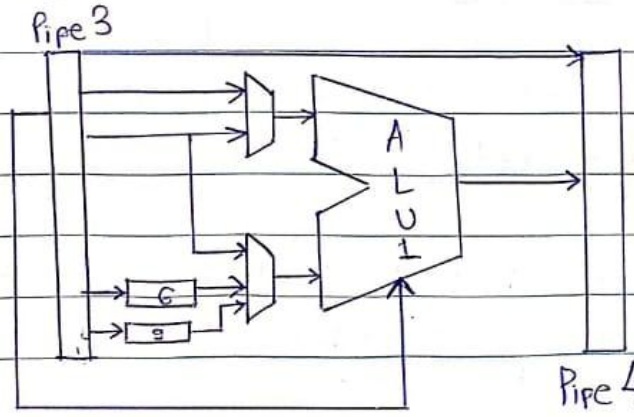
for ADI, LW, SW,

Pipe3 [data 1] ⟶ ALU1-A
Pipe3 [imm.] ⟶ ALU1-B [Sign extended]

for JRI

Pipe 3 [data 2] ⟶ ALU1-A
Pipe 3 [Imm 8-0] ⟶ SE ⟶ ALU1-B

* for LLI no ALU is needed



* The CZ bit will be written in ALU1 & be given to Pipe 4 along with output

Stage 4 [Memory Access]

Pipe 4 → Pipe 5

for SW, SM

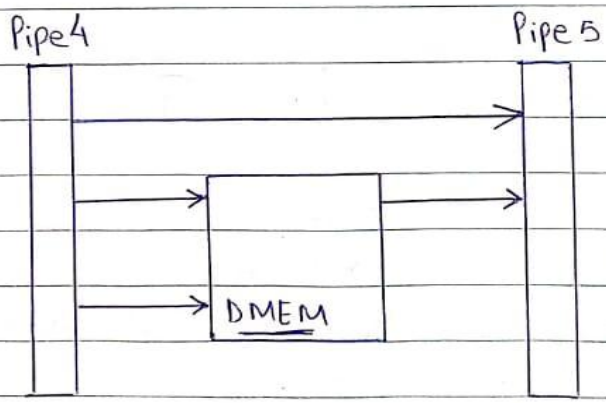
SW,

Pipe 4 (ALU-out) → Mem_Add (Data memory)

Pipe 4 (data 2) → Mem- write

(memory is write enabled only for SW, SM)

.P



for LW, LM

Pipe 4 (ALU-out) → Mem_Add

Mem_data → Pipe 5

Stage 5 (Write Back)

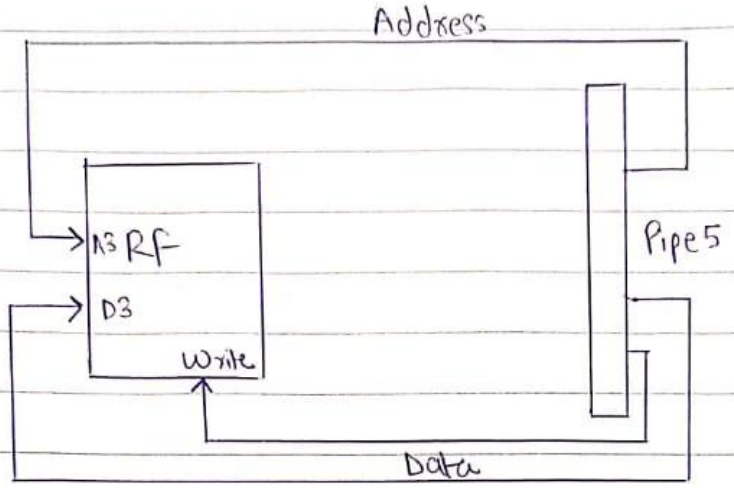
Here we have instruction, including the data in Pipe 5 so we select the write address from instruction, and see wheter this instruction writes or not for enabling rf_write and provide the data.

for ADA, AWC, ACA, ACW, ADI, NDU, NCU, NCC, LW, JAL, JLR, LLI

Pipe 5 [data] ⟶ RF_D3
Pipe5 [Address] ⟶ RF_A3
rf_write ⟶ Enabled

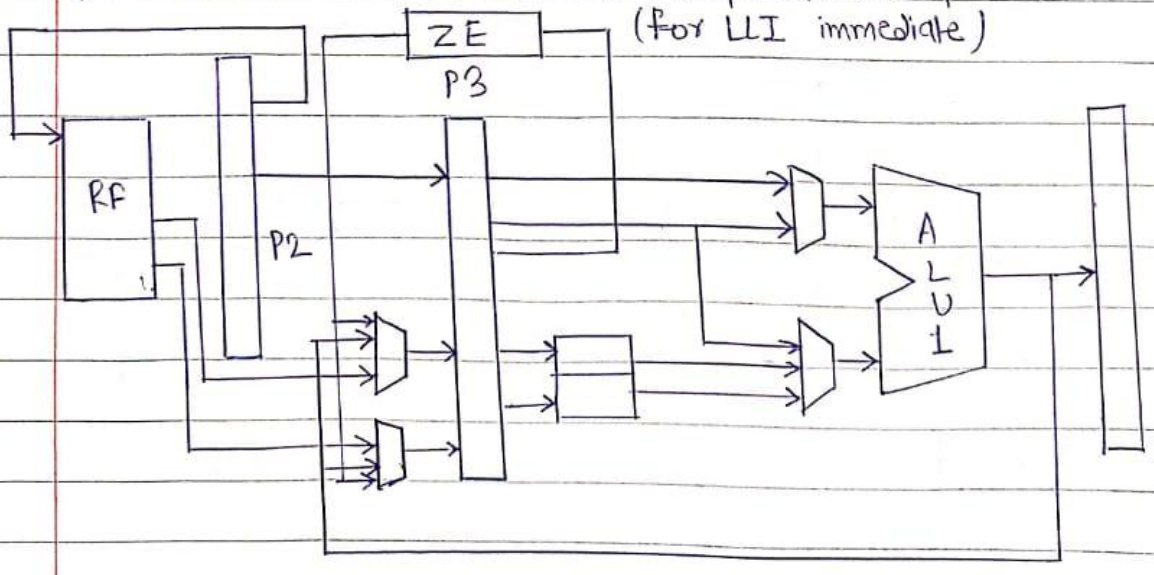
for ADC, ADZ, ACC, ACZ, NDC, NCZ

Pipe 5 [data] ⟶ RF-d3
Pipe5 [Address] ⟶ RF-A3
if the conditions are met then
    rf_write ⟶ Enabled
else
    rf-write ⟶ Disabled.



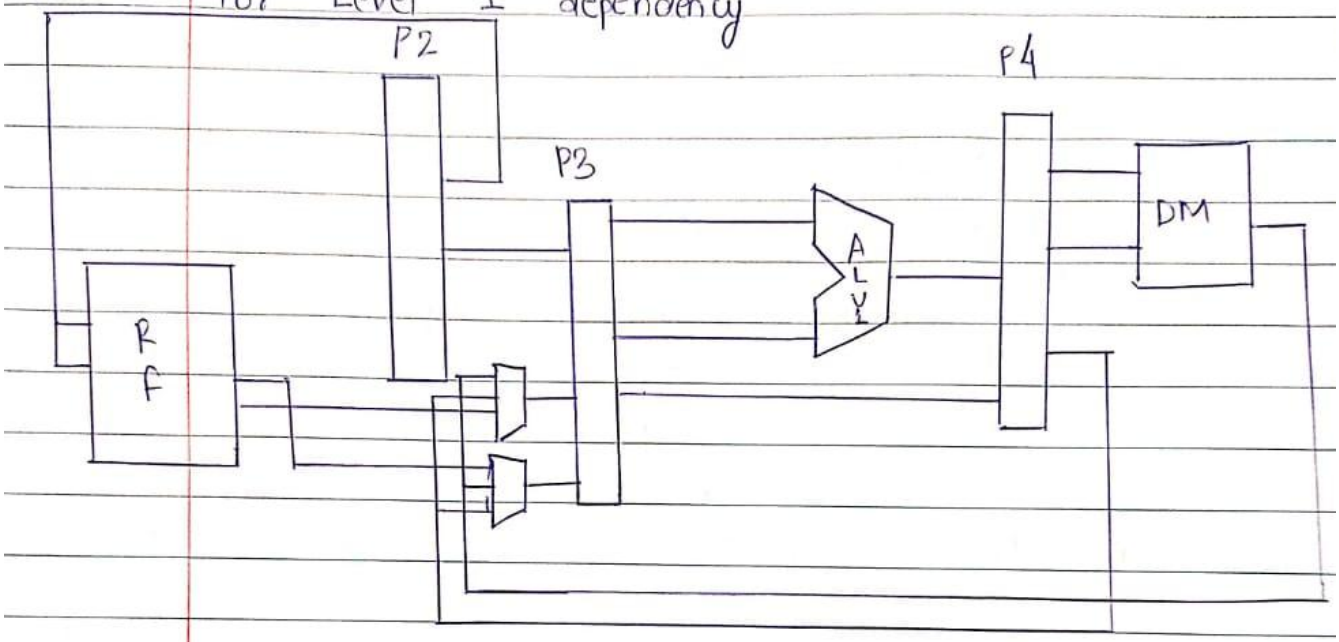
Data forwarding :-

If in Pipe 2 we read data which is modified by instruction in Pipe 3, Pipe 4, Pipe 5 then,
for P3
In case of Arithmetic dependencies,

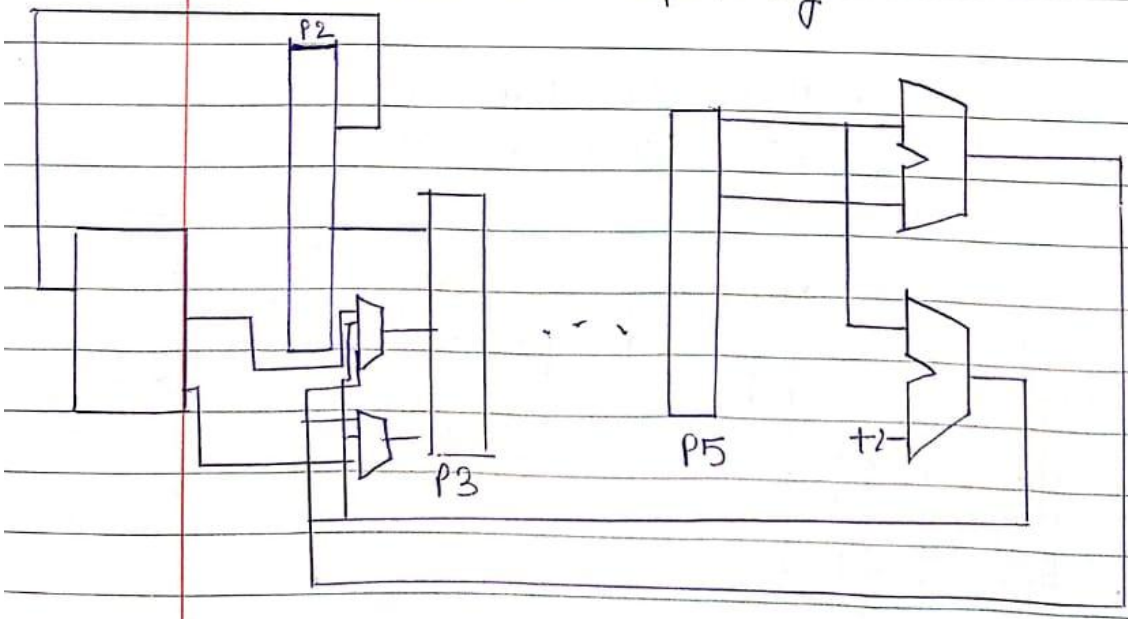


This is for the Immediate dependency

for Level 1 dependency



for Level 2 dependency



for LM-SM : (J-type)

Stage 1: [Instruction Fetch]

PC → Instruction Memory, Pipe 1, Alu2-A
Instr → Pipe 1
+1 → Alu2-B
Alu2-C → PC_in

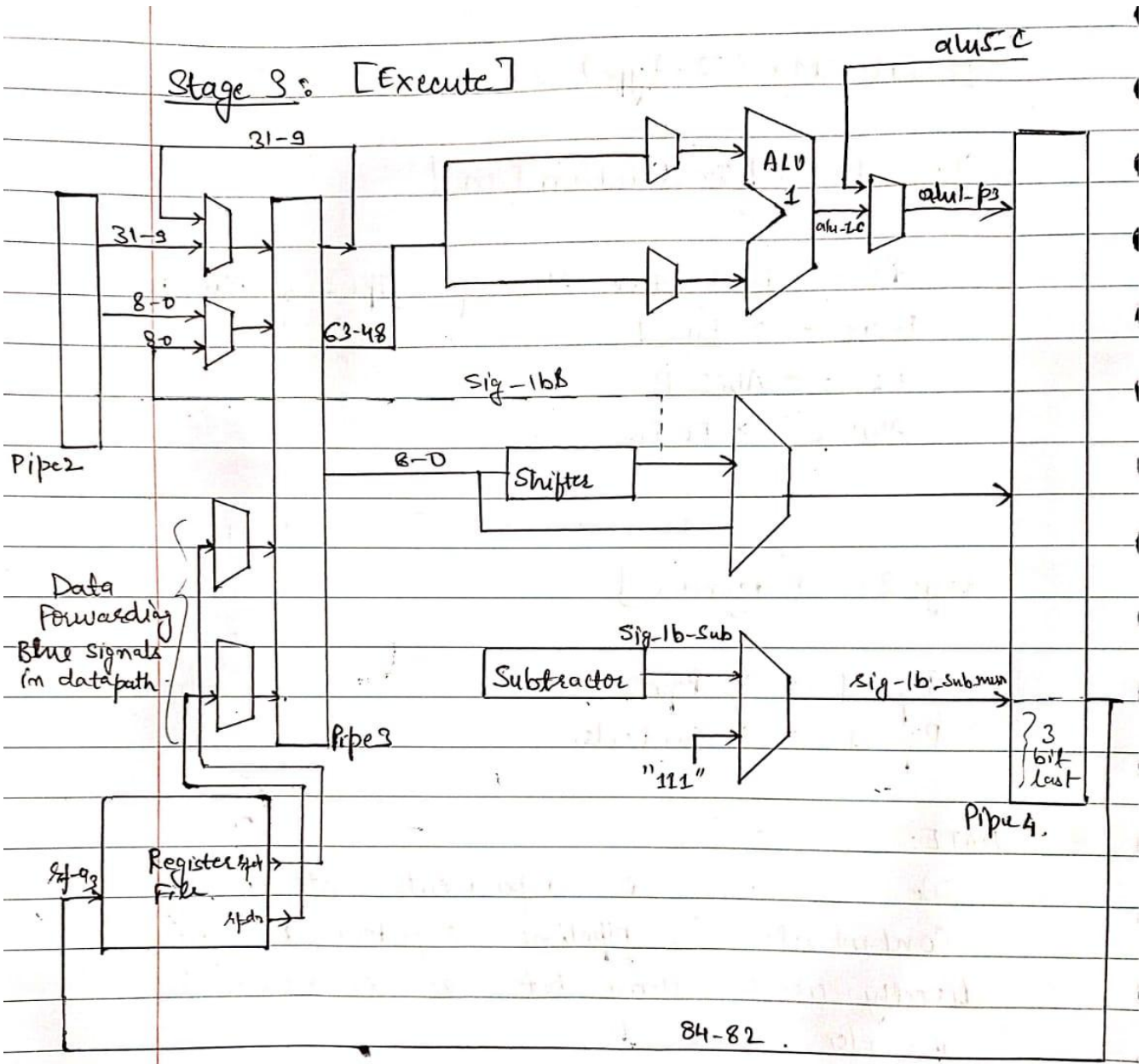
Stage 2: [decoder]

Pipe 1 → Pipe2
Pipe 1 → controls.

NOTE:

We haven't made a particular storage of control bits in pipeline registers. We have directly used those bits as conditions in if-else statements.
But as Sir has shown control bits in class we have shown them in our datapath.

Stage 3 : [Execute]



For LM

→ We have to maintain LM for 8 cycles

Pipe [31-9] → Pipe3 [31-9]

sig-lbs → Pipe3 [8-0]

sig-rf-d1 → Pipe3 [47-32]

Pipe3 [63-48] ⟶ alu_1A , alu_1B and operands.
alu_1_C ⟶ Pipe4 [79-64].
sig_lbs ⟶ pipe4 [8-0]
sig_lb_sub ⟶ sig_lb_sub_mux

```
if (sig_pipe4 (84 downto 82) /= "000") then
    sig_sub_ON <= '1'
else
    sig_sub_ON <= '0'
```

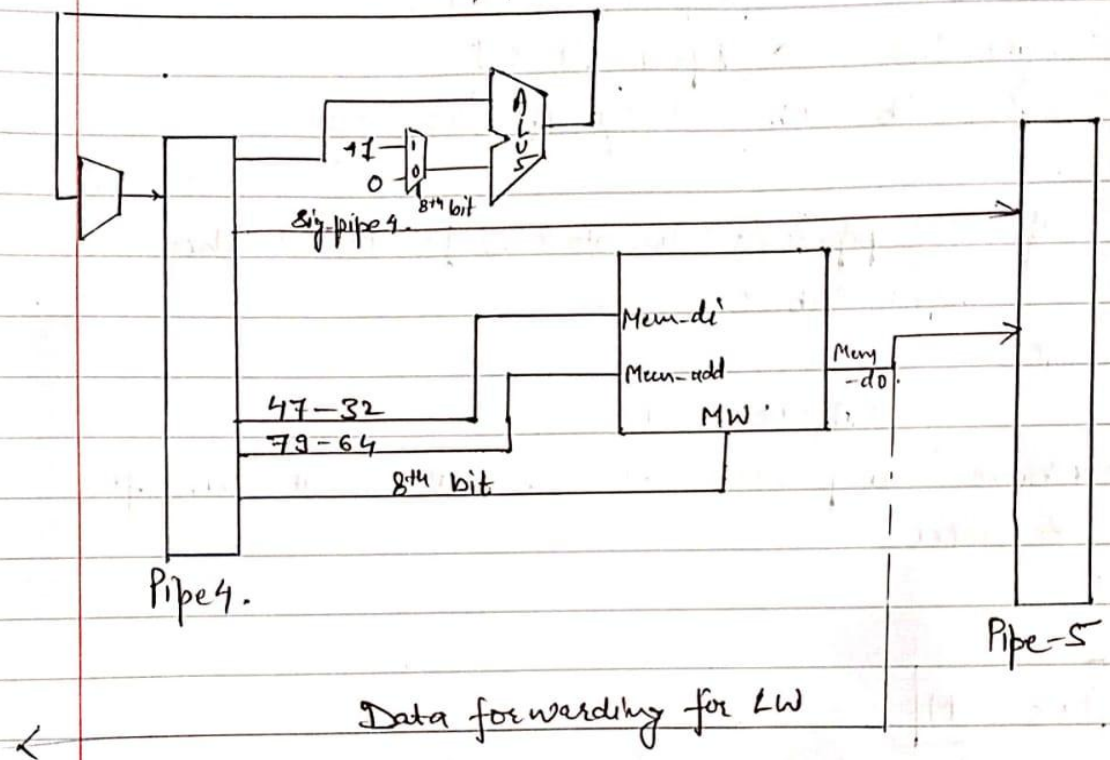
→ Above loop will ensure that it runs for 8 cycles.

For SM

only change in condition:

```
if (sig_Pipe5 (100 down to 98) /="000") then
    sig_sub_on <='1'
else
    sig_sub_on <='0'
```

Stage 4 : [Memory Excess]



→ In this stage we are updating the address dependency on the 8th bit

for LM: Pipe4 [79-64] ⟶ mem-add

mem-do ⟶ pipe5

Pipe [79-64] ⟶ alu5-A

+1/0 ⟶ alu5-B.

↓ if 8th bit = 1 ; ↳ if 8th bit = 0

alu5-c ⟶ pipe4 [79-64]

for SM: → pipe4 [79-64] → mem-add
pipe 4 [47-32] → mem-di
pipe 4 [8] → sig-MW

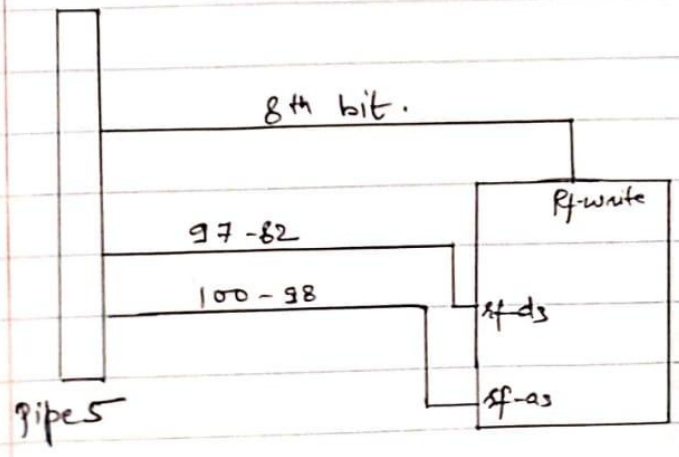
pipe 4 [79-64] → alu5-A
+1/0 → alu5-B
If 8th bit is 1 then do +1
If 8th bit is 0 then do +0

alu5-C → pipe 4 [79-64].

Stage 5: [Write back]

For LM:

pipe 5 [100-98] → sig-rf-a3
pipe 5 [8] → sig-rf-write
pipe 5 [97 downto 82] → sig-rf-d3.



HAZARD DETECTION AND MITIGATION:

- **R0 HAZARD:**
    - As per our understanding of the problem statement , we are considering that the programmer is highly skilled and given specifications of our design , he/she won't run instructions which are 'R0 hazard' specific. i.e. none of the instructions will have the destination address as "000" , which is the address for PC .
    - The above mentioned Hazard is very frequent for such a design in which the PC is in the register file itself .
- **Dependencies without R0:**
    - If the operand register for new instructions is the same as that of the destination register of previous instruction, let it be in the execution stage or memory stage or writeback stage, there will be errors and we can use forwarding directly from those stages. One thing to be noted is that in case of conditional execution like ADC, ADZ, NDC, NDZ if the respective flags are not set then there is no need for forwarding even if the destination register of previous instructions are the same.
        - For all these forwarding we are using two muxes and the output is to be given to pipeline reg3 ports data47_32 and data63_48.
        - For ADD, ADC, ADZ, NDU, NDC, NDZ ,ADI and other R-type instructions we are forwarding from ALU_1 and thus correcting IMMEDIATE dependencies.

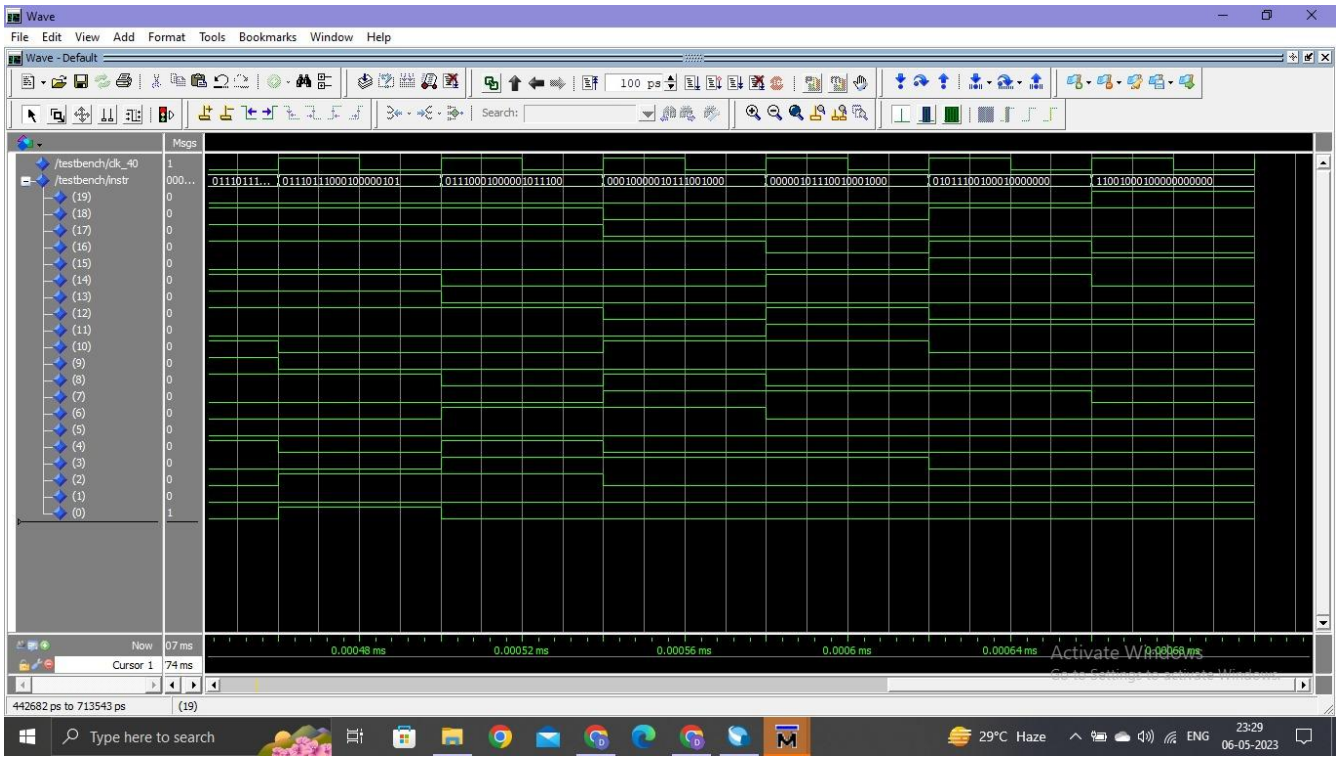
        - For ADD, ADC, ADZ, NDU, NDC, NDZ ,ADI and other R-type instructions we are also forwarding from 79-64 bits of pipeline register 4 and thus correcting LEVEL 2 dependencies.
        - For LW we are forwarding from the mem_d0 LEVEL 2 dependencies.
        - For JAL and JLR instructions , forwarding is the same as other R type instructions.
    - We can't correct LW immediate dependencies and so we have to stall the pipeline for 1 cycle so that load reaches memory stage and add or nand is in register-read stage and then we do the forwarding from mem_do.
- **LM-SM hazards:**

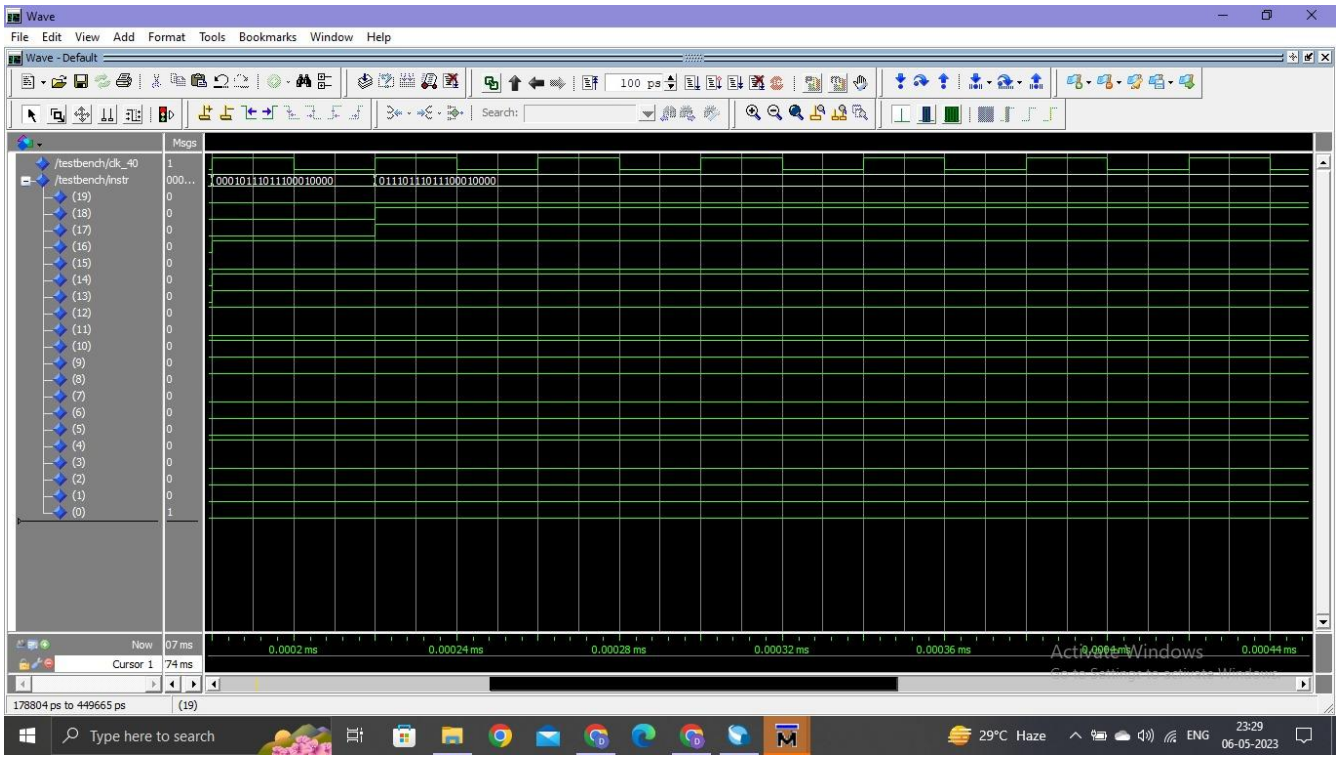
  In case of LM-SM instruction we have to stall pipeline register1, Pipeline register 2 and stop updating the PC .
    - LM**:**
        - Subtractor is turned ON when LM is in Pipe4
        - Wrp1 ,wrp2 (write enable for pipeline registers) takes input 0 when LM in pipe3.
        - PC updation stops when LM is in Pipe3.
        - Simultaneously transfer Shifter output into pipe4 when LM is in pipe3
        - When 8 cycles complete:
            - We are checking the above condition by checking sig_pipe4[84-82] != "000"

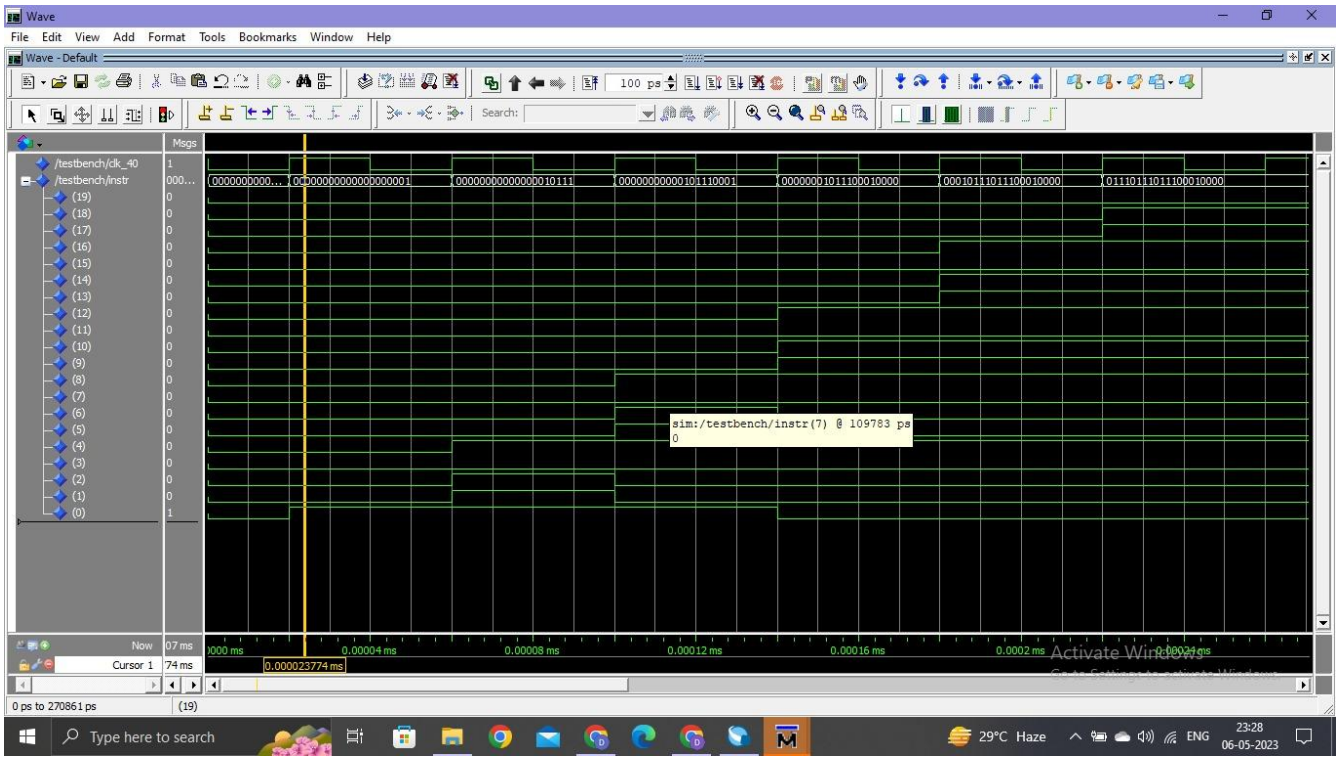
- After 8 cycles PC updation starts
- Subtractor is turned off , and bits 84-82 of pipeline register 3 are selected as “111”
- wrp<=1 , wrp2<= 1
- All muxes related to LM-SM , start passing normal signals for other instructions.

- SM:
  - Subtractor is turned ON when SM is in pipe2
  - Wrp1 , wrp2(write enable for pipeline registers) are set 0 when SM is in pipe 3
  - PC updation stops when SM is in pipe3
  - Simultaneously transfer Shifter output into pipe3 when SM is in pipe2
  - When 8 cycle completes:
    - We are checking the above condition by checking, sig_pipe5[100-98] != “000”.
  - After 8 cycles PC update starts.
  - Subtractor is turned OFF
  - wrp<= 1 , wrp2<= 1(enable write enable for pipeline registers)
  - All muxes related to LM-SM start passing Normal signals for other instructions

RTL SIMULATION RESULTS and TRANSCRIPT:







Sequence of Instructions shown:

1. ADA: opcode: 0001
2. SM: opcode: 0111
3. ADA: opcode: 0001
4. ADI: opcode: 0000
5. SW: opcode: 0101
6. JAL: opcode: 1100
7. BEQ: opcode: 1000
8. BEQ: opcode: 1000
9. ADI: opcode: 0000
10. ADI: opcode: 0000

NOTE: The above results show 10 different instructions.Picture second shows SM instruction which runs for 8 cycles .

NOTE: Transcript and datapath are separately attached as text and pdf files respectively in the zip folder .