# EE-309: Microprocessors

## IITB-RISC

## The Pipelined Implementation

Design Report

# TEAM 37

Team members:

1.Pyla Ramya :22B1305
2.Yashaswini Kaluguri :22B3911
3.Sreechandana Nistala :22B1301
4.Vanka Mahithanvitha :22B1239

# WORK DISTRIBUTION:

Pyla Ramya: Datapath, hazard management(designing),ideation, I_memory,Testbench, Debugging,report
Yashaswini: Datapath, hazard management(designing), ideation, Register file, D_memory, Debugging,report
Sree Chandana: ideation, adder, report, sixteenbitadd, sixteenbitand, alu, Debugging
Mahithanvitha: ideation, sign extension, shift by 1, LLI, 1-bit-adder,alu

## The six states:

## Instruction Fetch:

IR is fetched from instruction memory (Read only I_memory) and stored in pipeline register (IR_IF). Only when IF_en = 1, values update in pipeline register. Pc gets updated/stored in pipeline register.

## Instruction Decode:

Provides control signals according to the current instruction (all the enables are set). Contains SE6, SE9 and Shift by 1 whose values after computing (according to the instruction) gets stored in pipeline register.  LLI is functional unit required for LLI instruction.

## Register Read:

Register File takes the addresses A1_ID,A2_ID from the pipeline registers and outputs DO1,DO2 respectively. Register file has extra 8 inputs of LM to reduce the wastage of number of cycles for LM instruction. For immediate dependency load instruction, input of addresses are taken care of in this stage.

## Instruction Execute:

ALU performs ADD/NAND operation depending on the control signal. Adder is used for the instructions where carry bit also should be added with the regA and regB 16 bit values. Carry and zero flags are updated only when enabled. Hazard management via forwarding the data is also done in this state.

## Memory Access:

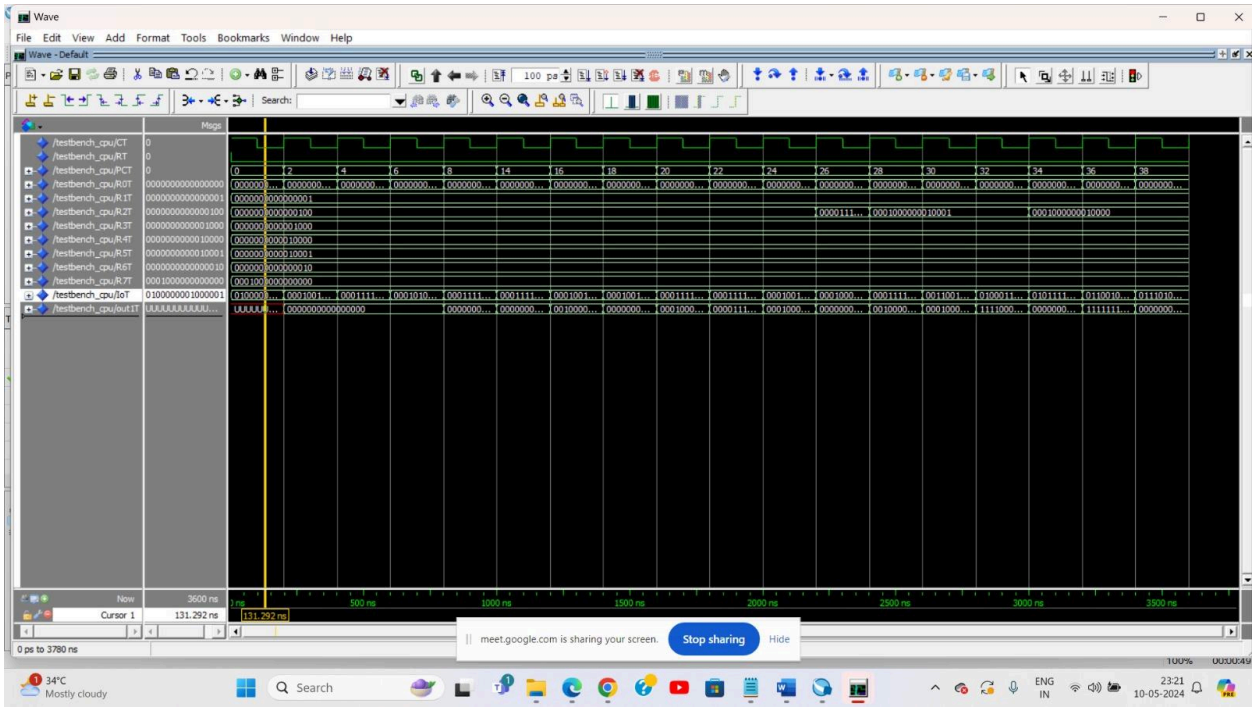
Data memory is readable and writable. It has extra 8 inputs and outputs of SM and LM, not to waste 8 cycles during performing LM or SM instructions. All the 8 16-bit data from memory is taken at a time.

## Write Back:

Register file write takes the Aw_3(address 3 to write data in it) and write the data in it. Reg_wr is the enable signal, only when it is 1 data gets written in the address.

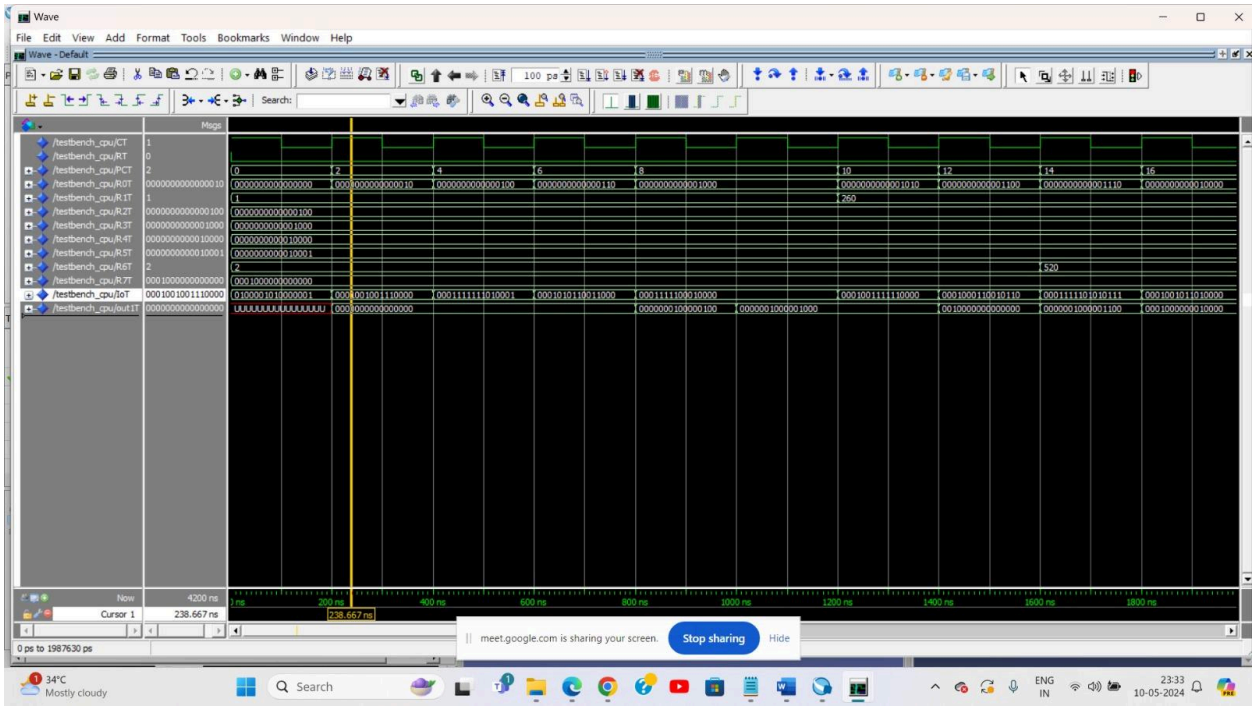
Flushes are used when there is wastage of cycles. When the previously calibrated data stored in pipeline is not required(like in unconditional jump), flush is enabled to erase the current data in pipeline registers.

# HAZARD MANAGEMENT:



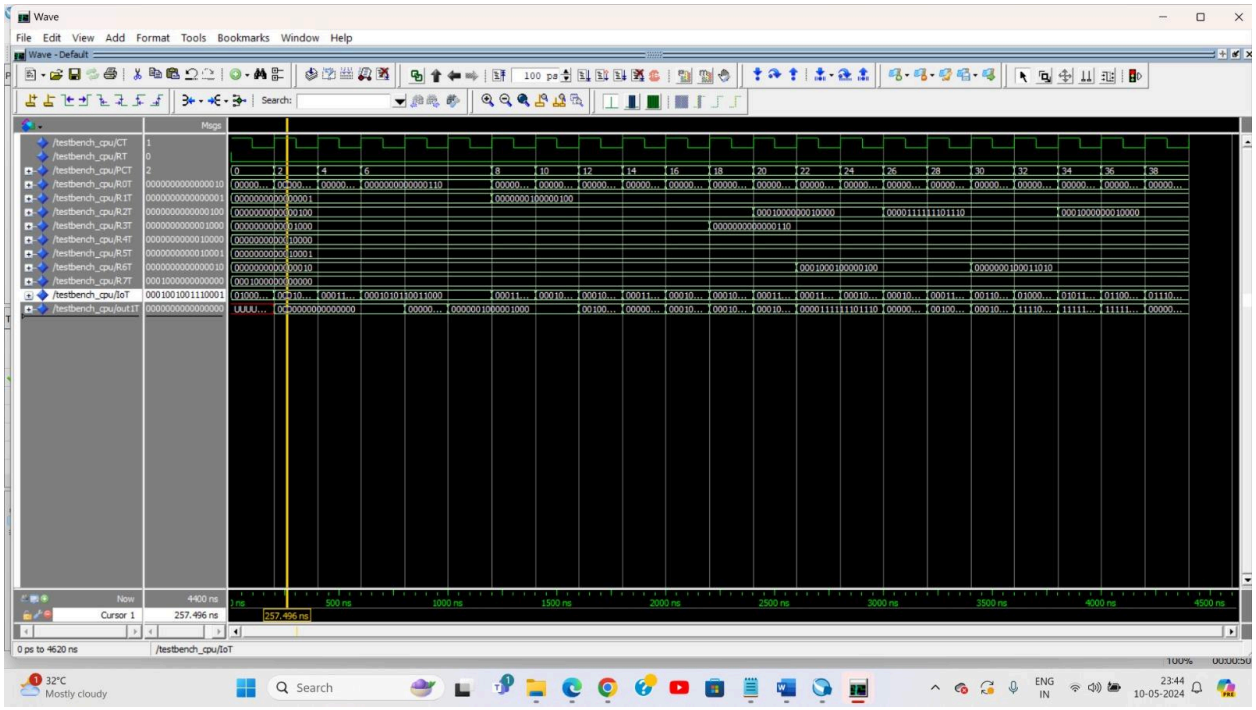

## LW hazard when R0 is destination (detected from Memory-Access stage)

Unwanted instructions are fetched before LW writes back in PC. So, three instructions have to be flushed and data is forwarded to IF stage such that PC gets modified immediately.
The data to be written into the register is read from the Memory access stage itself.



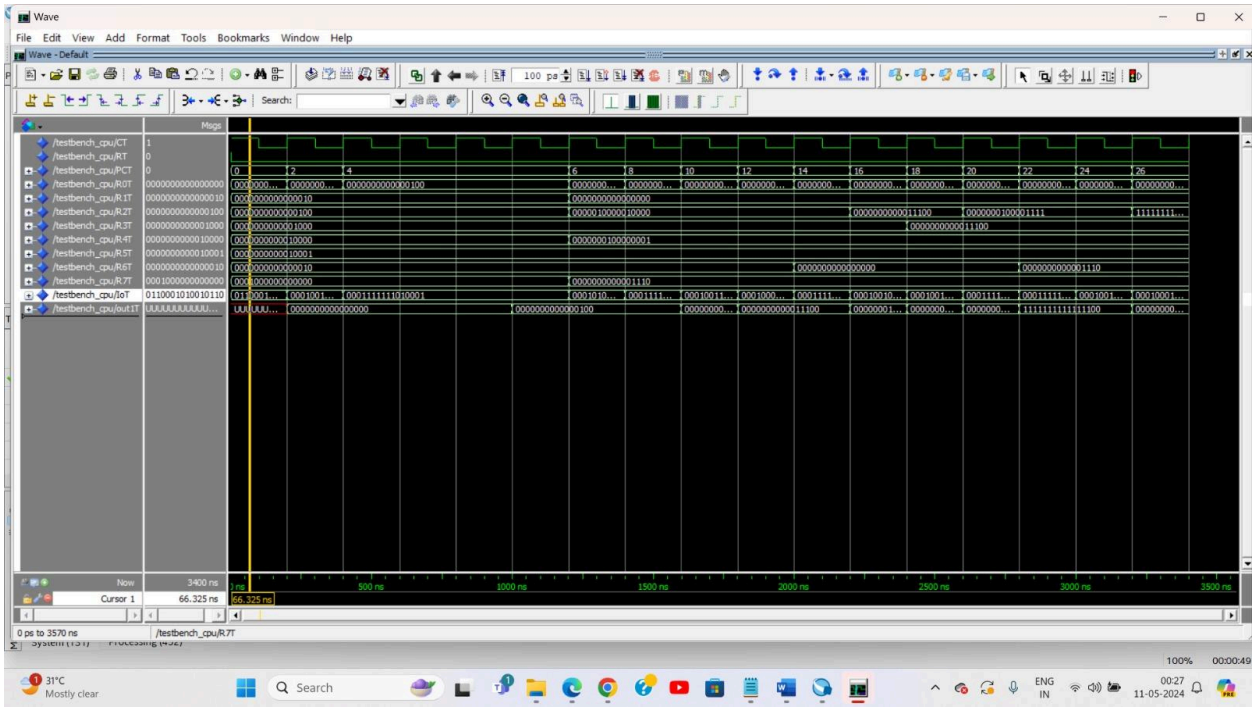

# LW(at Mem) followed by dependency using ALU(at Ex) then haz_load_dep hazard is detected and rectified

If there is an immediate dependent instruction after load,forwarding is not a sole solution since the data to be consumed is not produced itself.
Therefore, there must be a stalling for one cycle followed by forwarding of the data.



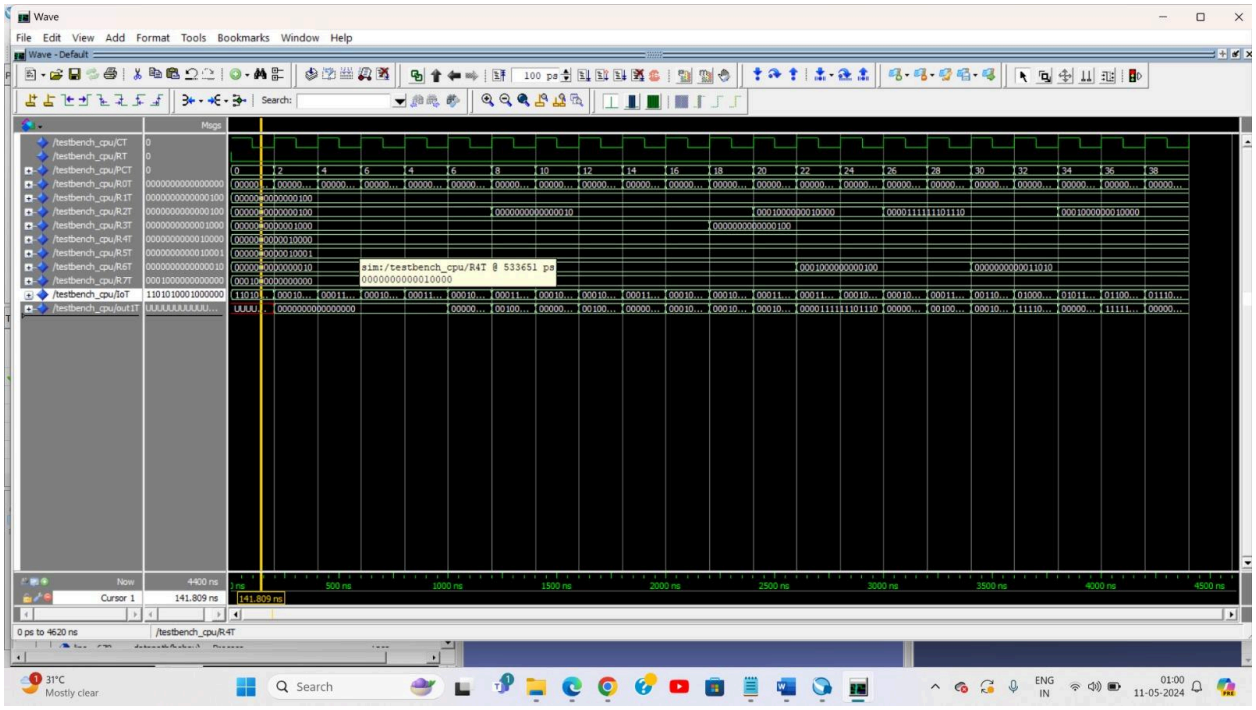

# ADZ/NDZ at RR stage dependent on LW(at Ex stage) haz_lw_zero_dep hazard is detected and rectified by stalling

LW has to modify the zero flag,but it is possible only after reading from the memory.ADZ/NDZ needs the status of zero(yet to be computed/modified) in execute stage.This is solved by stalling for two cycles.



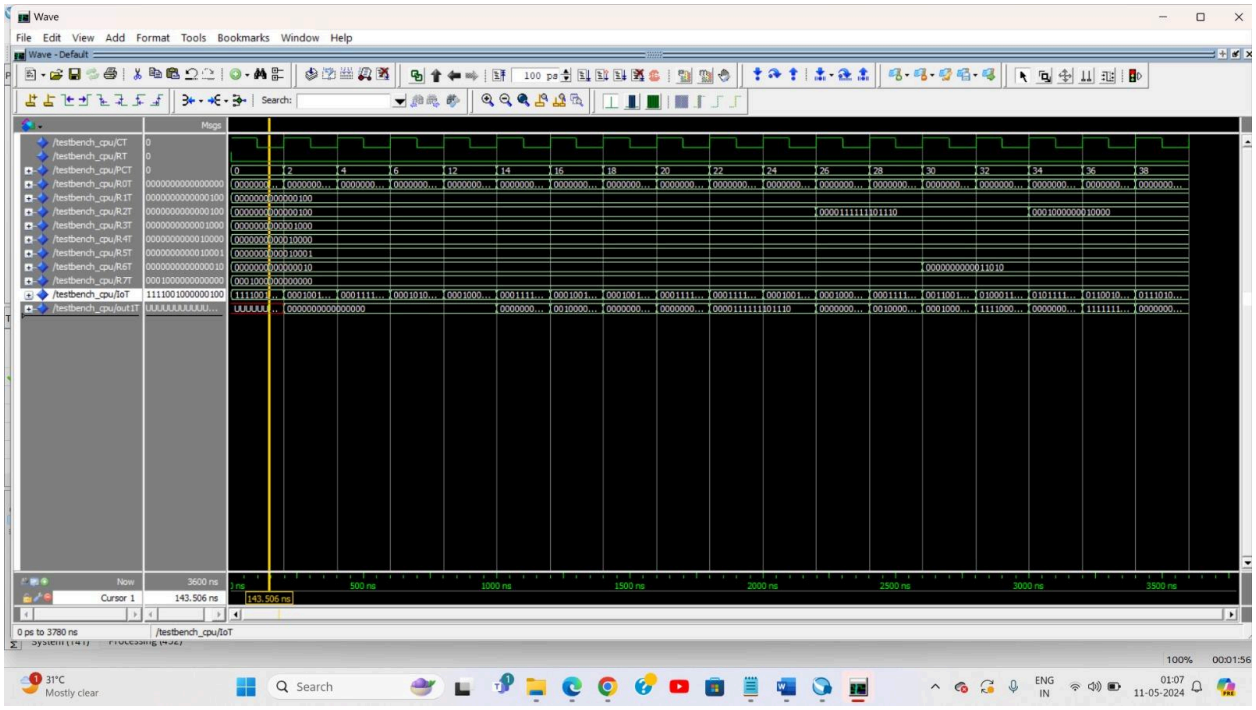

LM_SM Instruction always stalling of 3 cycles is done to avoid hazards haz_lmsm1,haz_lmsm2,haz_lmsm3, LmSm stalls are used for this purpose.

Loading and Storing of LM and SM instructions respectively is done in One cycle.But to avoid unwanted dependencies of later instructions on the unmodified registers or memory,stalling is done for three cycles until writing back is done.



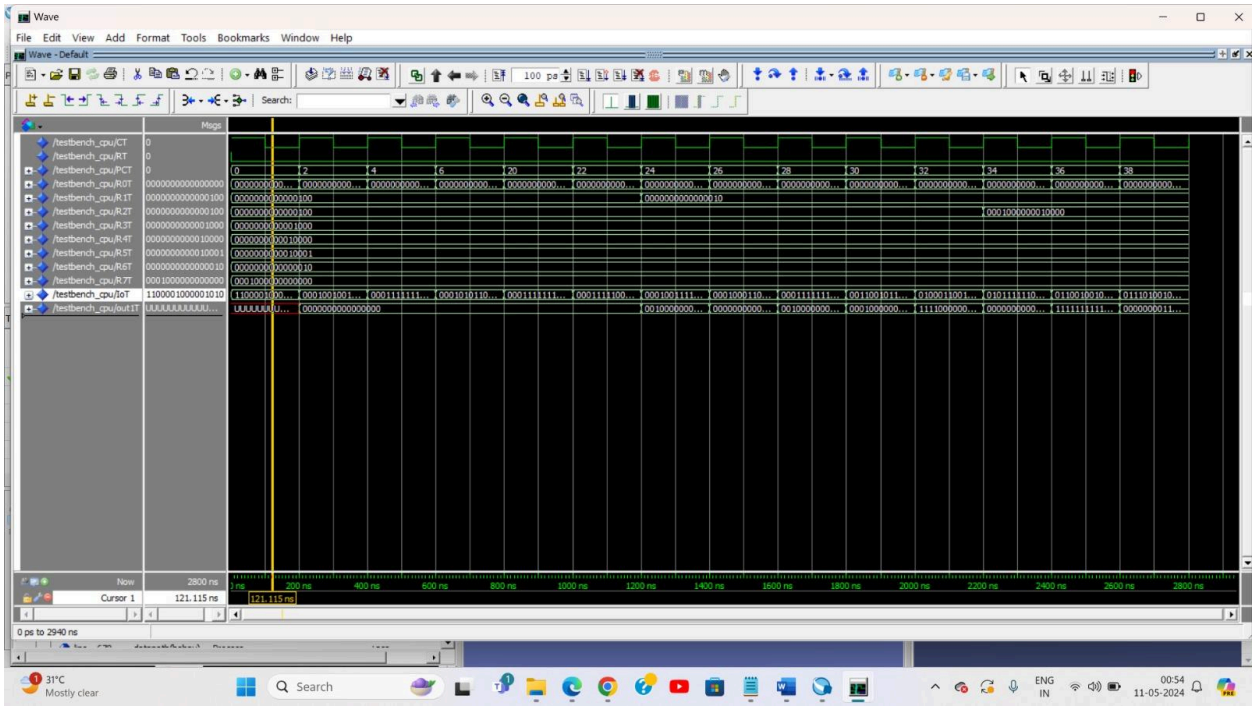

# Haz_jlr_ex : this is used to detect and rectify hazard due to JLR instruction which basically flushes unwanted instructions

JLR branches the PC depending on the data in the register which is which is read in RR stage.By that time, two unwanted instructions would already have been fetched.
These unwanted instructions are flushed and modified PC value is forwarded.



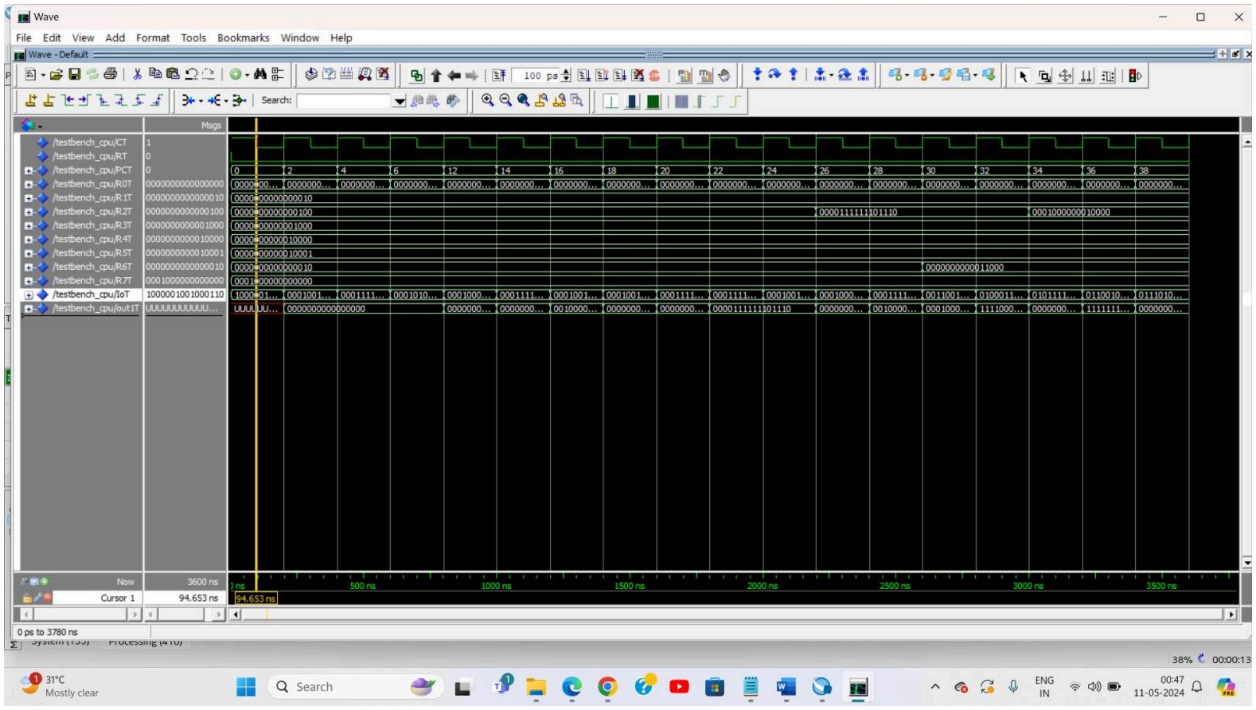

haz_jri JRI instruction: is used to detect and rectify hazard due to JRI instruction which basically flushes unwanted instructions.

JRI branches the instruction depending on the reg+immediate value which is computed in the execute stage.By that time, three unwanted instructions would already have been fetched.
These unwanted instructions are flushed and modified PC value is forwarded.




JAL instruction where instruction and this uses haz_jal_ex to avoid hazards and flushing unwanted instructions

JAL branches the PC depending on the immediate value which is computed in the execute stage.By that time, three unwanted instructions would already have been fetched.
These unwanted instructions are flushed and modified PC value is forwarded.




haz_beq,haz_blt,haz_ble hazards when conditions are met where we need to update PC to PC+imm*2 we flush next instuctions in this case

There is no hazard when the branch is not taken. But when the branch is taken, PC is modified. The modified value is calculated in execution stage. So, three unwanted instructions have to be flushed.

# Pipelined Architecture