Pipelining

New Components

- $branch\_addr\_alu$ to calculate the address of the branch in ID stage
- IF_ID register bus
- ID_EX register bus
- EX_MEM register bus
- MEM_WB register bus
- $multALU$ as the multiply unit
- $multALUbus$ as the register bus inside the multALU unit pipelined

Modifications in CSRfile

NB: This section can be skipped as it has no visible effects on the execution, it is just optimization

- The input instruction is used for both reading and writing to the CSRfile. But now each will be done in a different stage
- Reading is done in the EX stage
- Writing is done in the WB stage
- To solve this issue I added an additional instruction_WB input port. This will take the instruction of the WB register, and the initial instruction will therefore only be used for reading
- We also modified the instruction counter to read from the instruction_WB not EX
- The condition of instr_word_WB(0)=0 or 1 is used to make sure we don't count if the signal is undefined

Flushing

→ I need to flush the instruction fetched after a taken branch or a jump instruction

- I can do this relying on the c_PCSrc, if it is '1' it means I want to choose the addres coming from the branch_addr alu so it means I shoudn't have taken the pcplus4 hence I need to flush.
- To flush all i need to do is just inside the IF_ID register I put an if condition and set every output port to 0 in case c_PCSrc ='1'

New Control signals

- Input:
  - MStall coming from multALU unit
  - MNop coming from multALU unit
- Output:
  - Stall: This is gonna be used for stalling in general. For our current purposes, MStall will tell us there is an instruction in the multiply unit. We check if the current instruction in ID is a M extension instruction we let it pass so "Stall<=0", else we stall it so "Stall<=1"
  - Nop: This is gonna be used for sendind NOP to EX/MEM in general. For our current purposes, Nop<=MNop
  - MExt: This is '1' when the instruction i have is a M extension instruction

Flushing

→ I need to flush the instruction fetched after a taken branch or a jump instruction

- I can do this relying on the c_PCSrc, if it is '1' it means I want to choose the addres coming from the branch_addr alu so it means I shoudn't have taken the pcplus4 hence I need to flush.
- To flush all i need to do is just inside the IF_ID register I put an if condition and set every output port to 0 in case c_PCSrc ='1'

M extension instructions

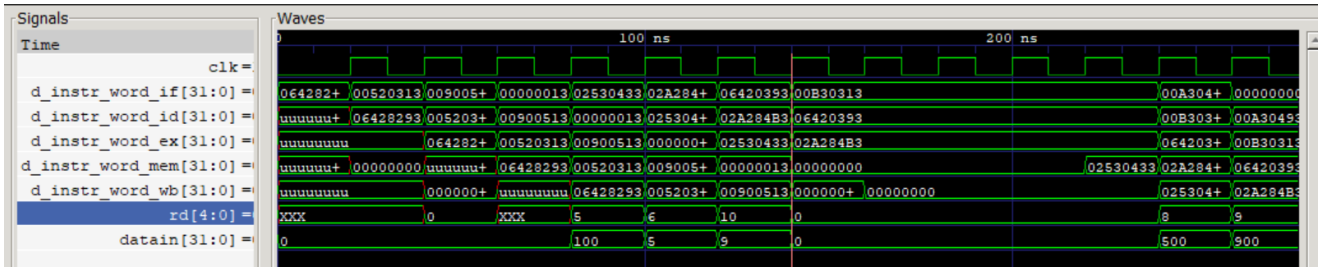
- He asked us to do a multiply unit that simulates 5 clock cycles to give a result
- I did a new multALUbus as well to simulate the pipelining inside the multiply unit alu
- When it comes to pipelining:
  - We made it in a way to support multiply pipelining
  - The only signals that we care to preserve from one multiply to the next are the result and the instruction. So we inputed the instruction in the mult unit, made it go throught the five stages and at the entrance of the EX_MEM bus there are 2 muxes that ensure we take the instruction and the result from the multiply unit
- When it comes to stalling:
  - MStall will go to 1 when MExt_EX1 becomes 1
  - MStall will go back to zero as soon as the instruction reaches the last execution stage (EX5) in the mult unit at the condition that the instruction in the EX4 and the instruction in EX5 are the same one.
  - This is to account for pipelined multiplication
- When it comes to NOP:
  - I only sent NOP to EX/MEM because the MEM/WB will inherit the NOP from it
  - As soon as the first multiply that entered the pipeline arrive to EX5, NOP turns back to 0

Testing the multiply pipelined - test_04

The RISC-V code:

```
06428293 addi x5 x5 100
00520313 addi x6 x4 5
00900513 addi x10 x0 9
00000013 nop
02530433 mul x8 x6 x5
02a284b3 mul x9 x5 x10
06420393 addi x7 x4 100
00b30313 addi x6 x6 11
00a30493 addi x9 x6 10
```

Please take the red line in the picture below as the cursor reference for the explanation below



- We can first observe in the EX stage that the first multiply instuction enters but does not stall anything as the second multiply enters right after it
- After the second multiply we can observe everything being stalled for 5 cc.

- The addi that is after the second mult gets stalled in the ID and the addi that is after it gets stalled in the IF
- After the NOP passes through the mem stage, everything after it is just empty instructions (zeros) until the first multiply finishes executing, then it gets sent to mem and everyting continues normally
- Finally we can observe through rd and datain (these are the signals being sent to rf) that all instructions are being executed correctly, and all values are being written in the register file correctly