PIPELINE PROCESSOR IITB-RISC

FINAL REPORT

DATAPATH: The six stage pipeline datapath consists of 5 pipeline registers, two 16-bit adders, one Priority encoder, one zero decoder and an ALU. The Pipeline is divided into 6 stages -

1. Fetch Stage
2. Decode Stage
3. Register File Read Stage
4. Execute (ALU) Stage
5. Data Memory Stage
6. Regsiter File Write Stage

Forwarding Logic – Forwarding is being implemented at four places

- **Forwarding logic from ALU -Stage to Register-Read stage :** The opcode at Pipeline Register 2(PR2) is analysed first whether it is going to read a register or not. Then opcode at PR3 is analysed for its destination address if it matches with any of the source operands of PR2. If the match happens and the opcode is one of add,adc,adz,adi,ndu,ndc,ndz,lhi etc. whose output is available in the execute stage itself then the data is forwarded from the output of ALU stage itself to the register read stage so that the operand values read in the PR3 registers are correct.

- **Forwarding from the Data Memory Stage to Register Read stage :** In case the source operands(at least one) doesn't matches with the destination address of PR3, then we have checked for the same condition at PR4 and if it matches then data is forwarded from the PR4 result register to the same MUX logic to which the data from execute stage is forwarded.

- **Forwarding from the Write-Back Stage to the Register Read stage:** In case the source operands (at least one) doesn't matches with the destination address of execute and Data Memory stage, the matching is checked at the PR5 register write-back address and if it matches, data is forwarded from PR5 result to the MUX logic mentioned above

- The MUX logic in the register read stage gets inputs from the three stages mentioned above and selects the appropriate value of operands

d1 and d2 to be stored in PR3 register. This kind of forwarding covers all cases except the LW and LM case where data is available at the Memory Stage case and is forwarded to the ALU stage and not the register read stage.

- **Forwarding logic from Data Memory Stage to ALU Stage :** This forwarding has been implemented when we have LW or LM at the data memory stage and any instruction at the execute stage. If a match happens between the destination address of PR4 register and source address(es) of PR3 registers then we stall the pipeline for one cycle i.e we make PC_en=PR1_en=PR2_en=PR3_en=PR4_en=0 so that only LW/LM instruction propagates forward, takes its data from the data memory and after one cycle this data is forwarded to the ALU(execute) stage. Also after stalling for one cycle the instructions at PR4 and PR5 stages are the same hence for flushing the PR4 instruction we change its opcode to “1111” in the same cycle so that when it propagates forward it doesn’t writes to any register/memory. After stalling the desired data is available at PR5_result which gets forwarded to MUX logic (mux16 or mux20) which are in ALU stage and determines the correct input to the ALU. If LW/LM (destination, source) matching happens at PR5 i.e Write back stage then we forward the PR5_result to ALU_MUX_logic without stalling.

Logic for LM/SM instructions :

- **Logic for LM/SM loop:** When LM/SM instruction appears at register read stage and the last 8 bits of LM/SM instructions are having at least two ones then we have stalled the pipeline before PR2 i.e made PC_en=0,PR1_en=0, PR2_PC_en=0. So considering a case suppose that last 8 bits are 00111100 (MSB to LSB) . These 8 bits are input to the priority encoder in the registerread stage which outputs 010 (the first register which needs to be written or whose value has to be stored) and using the zero decoder block and AND16 block we scrap that bit in the next clock cycle so that the output of priority encoder next time is 011. Scraping basically means after priority encoder detects registers from last 8 bits which need to be loaded or stored, the zero decoder generates a binary number which when multiplied with previous 8-bit data converts the used bit to 0 leaving other bits unchanged. Thus each unit of LM/SM propagates forward(in my above example there are 4 such units).

If the last 8 bits are all zero or is having only single one then we don't need to stall the pipeline before PR2 (it can continue moving forward).

Detailed Imlementation of LM/SM

- **Start of the loop:** Start of the loop is detected by checking the opcode at PR2. If it is equal to LM/SM then we disable the enable pins of PR2, PR1 except IR of PR2. other instructions cannot enter the pipeline at this moment because mux0 now rejects output from PR1 and PR1 itself is disabled. Now PC value at PR3 and PR4 are observed If they are not equal, then it means that LM/SM has just entered in pipeline for the first time (1^{st} iteration of loop). It means we have to add "0" to the address. If they are equal it means it has completed 1^{st} loop so we select 1 from second input of alu and calculate the second address. And this way the loop goes on and addresses are computed sequentially.

- **End of loop :** End of the loop is detected by detecting the output of multiplier, if it is equal to "0" it means it is the last iteration so we set the enable pins of PC ,PR1 and PR2_PC. And now mux0 will select output of PR1. And this way pipeline starts accepting new instructions.

Use of Valid/Truth bit for conditional instructions which can modify carry and zero flags :

- **Truth bit:** For the truthness of adc, adz, ndc, ndz we have defined a truth bit in PR4 which is set to 1 when result of these instructions have to be accepted. For example, if an add instruction is coming after adc and suppose there is a dependency(forwarding has to be done) then add instruction has to know what it has to do(whether to accept the new value which will be written by adc/adz/ndc/ndz or read the already stored value in register file) based on this truth bit. And also at the time of write back first this truth bit is checked in case of adc,adz,ndc,ndz. If truth bit is one at PR5 for these instructions only then the results will be written. For other instructions there is no need to check the truth bit.

- **The opcode "1111" for flushing :** Changing the opcode to "1111" for invalid instructions is being done in a number of cases -
  First when the destination is itself R7 then we are detecting it at the execute stage for (alu based instructions) and at the data memory stage for the LW/LM instructions. Then since writing to R7 is a kind of jump instruction we need to flush the previous pipeline registers and for that

their opcode is changed so that later they don’t write to data memory or Register files.
We are using this opcode to detect whether one clock cycle delay for stalling has happened or not. Like in case of LW/LM/SM we are doing stalling based on operands matching. This opcode is assigned to PR4 when stalling happens. After one clock cycle opcode at PR4 is checked if it is “1111” and opcode at PR5 is LW/LM/SM then it knows that stalling has happened and the pipeline starts recruiting instructions again.

** In the demo, I showed the LM and SM instructions individually to the RA but due to time constraints couldn’t show LM followed by SM instruction. But I tried it after demo and it is working perfectly as expected.I am attaching the screenshot of the simulation along with the project

*** In the code folder the datapath_fsm is out top-level entity

** **The pipeline datapath is there in the zip folder.**

Team:

SAQIB AZIM : 150070031

AWANISH KUMAR : 15D070037

ANIRUDH KUMAR YADIKI : 150070056