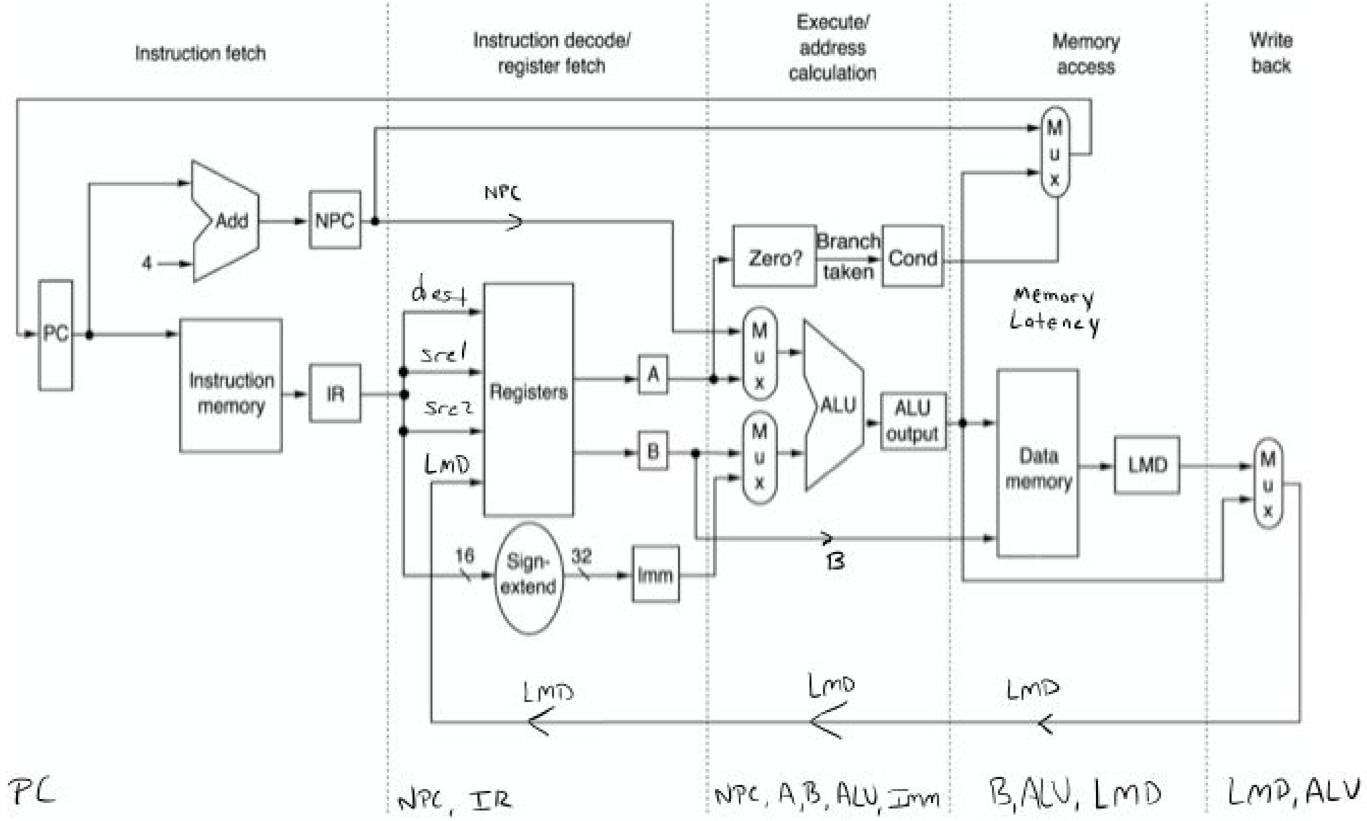
Bennett James
ECE563-001

5 Stage Pipelined MIPS Data Path

When designing and building a 5 stage pipelined MIPS data path there are many factors that must be accounted for to achieve successful integration. Several of these that were accounted for in this project were data hazards, memory latency, and control hazards. For the overall design of my pipeline data path. I used the diagram below, noting the registers that will need to be iterated through each stage to determine the registers that need to be used and how to populate the registers with the previous stages outputs.

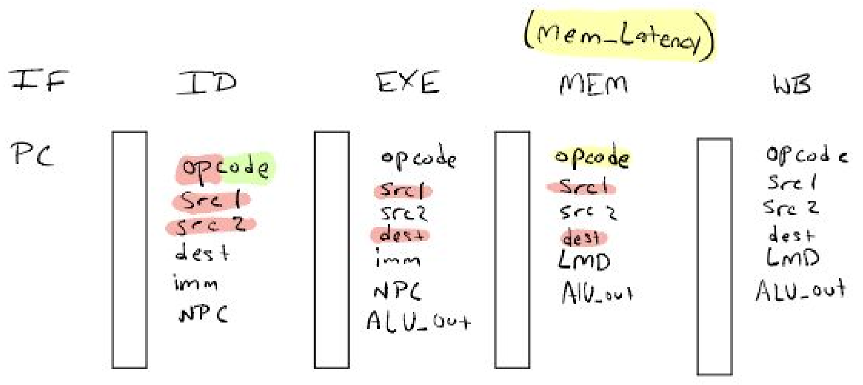


For the MIPS data path to reference the Special Purpose Registers or pipeline registers that include the PC, NPC, A, B, Immediate field, and several other key aspects that are pivotal to maintaining reliable reference points to these registers therefore I used a two dimensional array with the reference points of the first field being the register name such as PC, NPC, ALU_OUTPUT, LMD, etc. and the second field the stage of the pipeline that it would be inputting to such as EXE, MEM, WB, etc. All of the expressions that went into my fields were declared as enums to allow for easy readability and referencing. I implemented a similar structure for the general purpose registers R0-R31 but in a simpler manner considering this only needs a one dimensional array.

To handle the data I made five functions in the sim_pipe class that includes one for each stage name accordingly IF_pip_stage, ID_pip_stage, EX_pip_stage, MEM_pip_stage, and WB_pip_stage. Within the run function these were declared backwards as if declared in order data would flow straight through on one run. To handle the data as if there were no hazards I used mainly switch statements based upon the opcodes of the instruction. Depending on the stage of the pipeline it was in would determine which special registers would be filled. Testcase1 tested the full functionality of your ability to process data without any hazards.

Once the data path was constructed the main portion for the rest of this program is addressing three types of hazards which include data, structural and control hazards. To implement I use the diagram shown below to determine where in the pipeline the stalls would need to be inserted to allow for these hazards to be mitigated. The first hazards that were implemented were data hazards in these test cases; these were primary RAW or read after write hazards. Test Case 2 and 3 tested the implementation of these, for this type of hazard within my ID_pip_stage function I implemented a switch statement that would look at the opcode of multiple statements. These include the opcode of the instruction in the ID stage to then look at the registers being used and comparing with how those are being used as inputs to the EXE and MEM stages the inputs to the WB stage are negligible because they will be out of the pipeline and all hazards will be mitigated before the instruction in the ID stage will ever be executed upon. To implement this the pipeline was stalled at the ID stage with the EXE and further continued to be executed for a number of cycles depending on the necessary number of stalls needed. NOP opcodes were inserted into the EXE pipeline for each necessary stall.

Bennett James
ECE563-001



Hazard Priority

1. Structural
2. Data
3. Control

The next hazard to be accounted for was control hazards these are caused during a branch condition. To implement this I was again in the ID stage in which I stalled the IF stage and inserted NOP opcodes into the ID stage for 2 statements and the branch instruction moved down the pipeline and would then branch the PC in the memory stage if the conditions were met. This hazard did not have to look at any other opcode statements as the data hazard implementation method would take care of any register data hazards before the stalls here would even be inserted for the control hazard. This hazard was implemented in test case 4.

The final hazard that would be implemented was structural hazards. This hazard was implemented in the MEM_pip_stage. Depending on the data memory latency determined by the testcase the MEM_pip_stage would cause the whole pipeline to stall as during memory latency there are no stages that can continue to run. This was implemented in test case 5. This was also the highest priority hazard with data hazards being second and control hazards having the lowest priority.

Bennett James
ECE563-001

From an overall performance view. I ran all of the test cases outputting my outputs to their own files and used the diff command to compare the outputs expected vs. my achieved outputs and was able to achieve 100% accuracy on test cases one through six.

With regards to the Makefile I did not alter it at all or any of the test cases all changes in my code were done in the .cc or .h files as specified in the project specs.