IITB-RISC Pipelined
EE309 Course Project

Manan Garg, Roll. no: 210070052

May 3, 2023

1 Datapath

The diagram below shows the basic Datapath design. The Purple bars represent the Pipeline Registers, red elements represent hazard-dealing hardware, green represents memory of any kind and grey represents combinational logic.

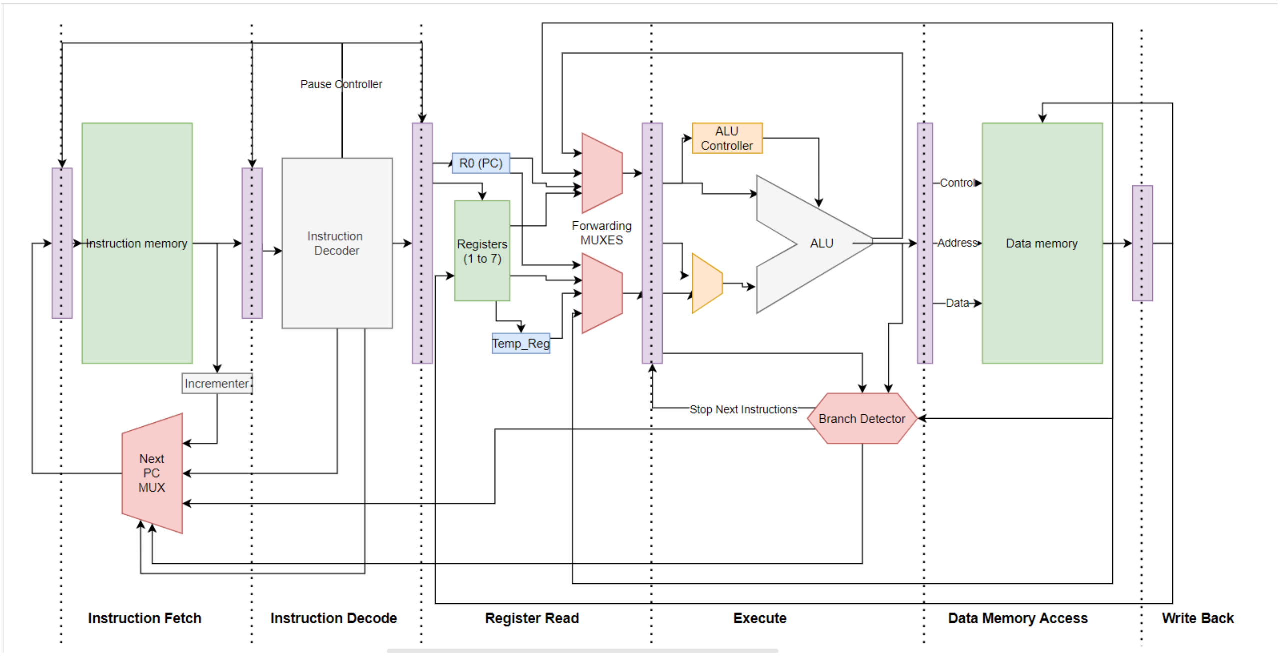

Figure 1: Data Path Design

2 Hazards

2.1 Register 0

This Register is being updated every cycle, and will be updated along with any Registers being updated due to the instruction. Normal forwarding would not be practical here. My solution is to send the instruction address along the Pipeline, which is always the correct value for R0. If the value of R0 is needed, this value is read and used.

2.2 Forwarding

The implementation for this is very standard. The control signals for the MUXes used for forwarding is simply that the Register address requested is equal to one of the register addresses the further stages are going to write into. We prioritize Exec stage forwarding over Data Mem stage forwarding and so on. R0 being special results in the following:

- The correct value for R0 is already present. Even if a further stage is going to write into R0, we use the already propogated value.
- Since sending R0 as the write addresses for the future stages basically results in nothing, we use the value 0 to signal that no forwarding is required due to that stage.

2.3 Forwarding - Immediate load dependency

In the instruction decode stage, we remember whether the previous instruction was a Load instruction and the register address it would load into. If for the current instruction one of the operands is the same, then we pause the pipeline for the IF, ID and RR stages. This results in two copies of the load instruction, which causes no issues. We then use the forwarding mechanism to get the correct operand from the Data Mem stage.

2.4 Branching - JAL

We can detect this branch and the compute the correct address in the ID stage itself. We freeze the IF to ID pipeline register for a cycle to trash the incorrect instruction, and send the correct address to the IF stage to get the correct instruction. The only work left in the rest of the pipeline is writing to a register, easily accomplished by the already created DataPath.

2.5 Branching - Compare, JRI, JLR and Writing to R0

For all these branches, we can get the the correct branching address only in the Exec stage. For this, we have a branch controller which sends the correct branch address to the IF stage and **freezes the RR -¿ Exec pipeline registers** for 3 cycles, purging the incorrect instructions.

- For the compare instructions, the Exec stage only computes whether we have to take the branch. The address to branch to is pre-computed in the ID stage, and is propogated along the pipeline and sent to IF if the branch is taken.
- Note: the comparison is done assuming the numbers are signed 16 bit integers.
- R0 branching can also occur due to a load instruction. Whenever R0 occurs as the write operand for a Load instruction, we use the Load dependency freeze to make 2 copies of the load instruction, and then when the *second copy* is in the Exec stage, branch is taken using address from data mem stage. The pipeline is again stopped for 3 cycles to purge the incorrect instructions.

- Any freezing of the pipeline in earlier stages is deactivated for to ensure the incorrect instructions are purged.

2.6 Load and Store Multiple

These instructions freeze the pipeline for IF stage, and the ID stage converts this instruction into consecutive Loads (or Stores). We also use a temporary register in the RR stage to ensure that if a load changes the initial value of the operand register, we still get expected behaviour.

3 Testing

- To test, you can write ”assembly” in file code.txt (in the testing directory) and run the script.ps1 power shell script to get the state of the Data Memory at the end of execution in file output.txt. A python script will also calculate the expected output(in expected_outputtxt), so that the two can be easily compared. (Of course, the ultimate decision of whether the simulated output is correct is with the evaluator, the expected output file is for convenience only.)

- You will need the quartus software installed, along with python and will have to enable external powershell scripts for this method of testing to work on your machine.

- Both the data and instruction memory can hold a maximum of 64 words in this simulation.

- Ensure that there are no infinite loops in the program written, as otherwise the python script calculating the output will get stuck. The printing of final Data Memory is done when the Program Counter goes beyond the written instructions.

- The registers in the python script are initialized to 0. This could potentially cause differences in expected and simulated output. I suggest loading registers with some values (like with LLI Instruction) to ensure that testing is smooth.

- Some sample programs are given in file Samples.txt. I have tested on more programs, these are some that have edge cases.

- If running the script causes issues or you are not on windows, you could try directly passing the commands in the script file to a terminal. The script assumes it has been called in the same directory it is present in.