Problem: Pipelining a single cycle datapath.

Given: The VHDL code for a single cycle datapath, broken up into 6 files, 1 file each for the fetch, decode, execute, and memory stages, 1 file for the control unit, and 1 top- level file connecting all the files and signals together. Fig. 1 shows a simulation waveform for the given datapath done using ModelSim, and Fig. 2, on the next page, shows a block diagram for it.

![A close up of text on a black background

Description automatically generated]()

**Fig. 1** The simulation waveform for the given single cycle datapath, run for 400ns with the ram and register array exposed.

A picture containing text, map

Description automatically generated

**Fig. 2** Block diagram for the single cycle datapath.

Find: Make changes to each of the files and/ or add new files such that instead of completing fetch, decode, execute, memory access, and writeback in one long clock cycle per instruction, the architecture completes one stage of one instruction in a clock cycle, while simultaneously completing the subsequent stages of the next instructions in the same clock cycle. The main data hazard to avoid is Read after Write.

Solution: I made the following changes to implement pipelining.

* The first change I made was to add internal registers between each of the five stages so that the hardware associated by one stage could pass on the control bits and data needed by the next stage to the register and move on to handling the next instruction. Thus, I created the IF\_ID register in FETCH.vhd, ID\_EX in DECODE.vhd, and so on.
* I implemented a hazard detection unit to take care of the Read after Write dependencies by stalling.
  + I made a new file called HAZARD.vhd that takes the current source register and the destination registers from the previous three instructions as inputs and detects a hazard if rs/ rt match with any of the destination registers.
  + It also takes in the opcodes from the previous two instructions to check if either of them were branch instructions and detects a hazard so that PC isn’t updated before branch calculation.
  + Once it detects a hazard, it stops PC+4 from updating by setting PCWrite to 0, stops the IF\_ID register load by setting IF\_IDWrite to 0, and sets all the control bits to 0.
  + Additional signals were defined for these delayed rd and opcode values as well as PCWrite and IF\_IDWrite.
* Modifications to the code were made such that Muxes, adders, etc. take the data they need from the appropriate internal registers so that they have correctly delayed versions of data and code bits as shown in the pipelining diagram in Fig. 3.

A close up of text on a black background

Description automatically generated

**Fig. 3** Block diagram for pipelined datapath.

Results: The vhdl code couldn’t be debugged in time, so a simulation of the pipeline is not available.