Qianhui Wang U7070170
Linxi Li U7095375

BPIPE

Since there is only one memory in QuAC ISA, we decided to combine the EXE and MEM stage in MIPS, that makes 4 stages: fetch, decode (reg_file read), execution and writeback (reg_file write). And we set 3 PPRs between these stages. We connect 2D port of RAM to the start of circuit as “instruction memory” in MIPS, as it also explains the lack of MEM stage. We put the “set high move low” circuit in EXE stage instead of ID stage. Because this reduces data that need to store without adding process time and makes the hardware simpler.

There are two structural hazards. First, the read and write of reg_file, we use the policy “first half write, second half read”. We connect the raising clock with reg_file and memory (for write), and we negate the clock for PPRs, so they can read in falling edge, PC don’t need to push back. Second, since the combined memory, if the fetch and EXE stages need to read and write simultaneously, and the address for read and write are the same, the reading need to be pushed back and insert nop. To detect structure hazard, check the stall signal set high, and IF address equal to store address. At this time, insert nop without changing the PC address. After the stall, read out from the same address and PC enable set low. To nop, just make all control unit signals zero. Actually, just set write enable and stall signal zero is enough, to prevent the content of reg_file and memory change.

DPIPE

The “true dependence” in “register-use” hazard and “load-use” hazard in MIPS. Since QuAC combined EXE and MEM, only 1 hazard here (less overlap in shorter pipeline), so we put the forwarding unit in EXE stage. We add forwarding path so reduce nop. If data dependence is detected in Rd, the circuit will pass the data from last instruction by forwarding path, else the data will be directly from reg_file. One difference from MIPS is, this already contain the “load-use” situation. Since “load-use” is mainly for memory, the ALU, memory and “set high move low” circuit are parallel without any connection, so we use multiplexer to choose one of them.

CPIPE

We set the branch detection unit in ID stage. If decode stage detected “Beq”, then use a new address, jump to next branch. Null the end instruction in IF/ID register, send nop to next cycle, jump to the correct instruction. We put the PC in reg_file, it can be written in by ALU or memory load. The conditional execution needs the flag of the most recent instruction in ALU, at the time that the ALU produces. But the reg_file have a half cycle delay, and the conditional execution will miss the flag if store to the flag register first, so we need a multiplexer to select. We also store the flag from ALU into flag register in reg_file. Because this keeps the most recent flag, even if the last several instructions didn’t use ALU.