RISC-V Class Project Phase 9 – Branch Optimization

Phase 9 of the Class Project modifies the branch logic to reduce the number of cycles lost to flushing on a taken branch. It adds an additional branch function to the design created in standardname8.

1. Copy the Base Project and Rename It

Start from the standardname8 project and rename it to standardname9.

2. Block Diagram Modifications for BEQ/BNE/JAL/JALR

Figure 1 shows the high-level architecture of the existing RISC-V design. Note that the branch decision is made in the ME stage in all cases, which results in three instructions being flushed on any taken branch or jump.

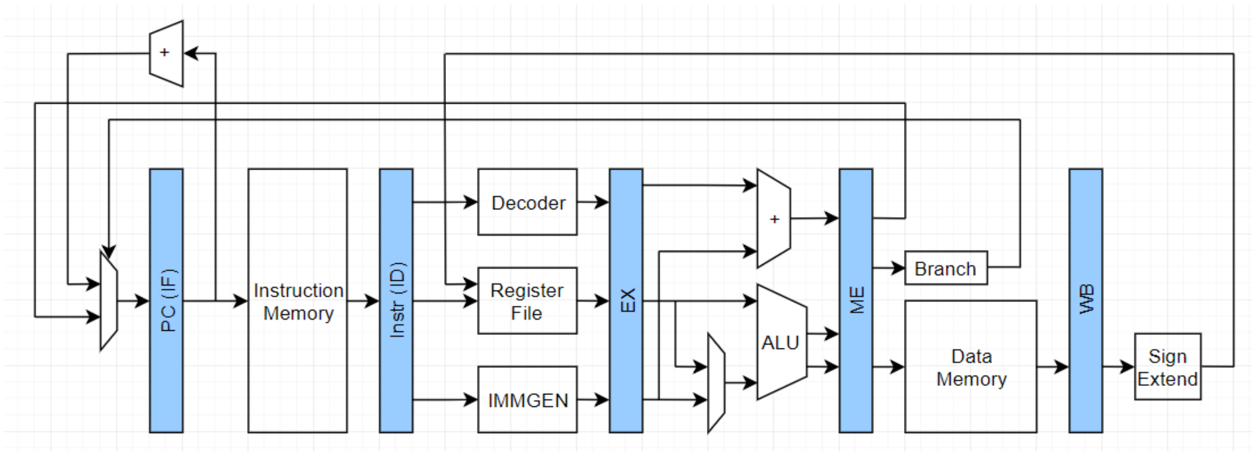

Figure 1

With some additional hardware some of the branch instructions can be modified so that only two instructions are flushed on a taken branch. Figure 2 is a block diagram of a modification which will allow some of the instructions (BEQ, BNE, JAL and JALR) to have this improvement. There are several straightforward changes which are required.

1) Add an XOR function in parallel with the ALU (“X” in Figure 2) to produce a faster zero output.
2) Add a new branch function in the EX stage which produces a new branch signal s_ex_take_branch. This branch function should execute the branch for BEQ/BNE/JAL/JALR.
3) Modify the existing branch function in the ME stage to ignore those instructions.
4) Add new enums to brnchop in ca_defines to handle the new cases which are required (Specific to the branch operations that are now handled in the EX stage).
5) Modify the decoder to create the correct values on s_id_brnchop.
6) Add a third input to the s_if_nextpc multiplexor (lines 44 and 45 in ca_pipe1_if.codal) in the IF stage selected by s_ex_take_branch. Select the appropriate signal for the branch address in that case.
7) Update the clear signal assignments to handle the new function.

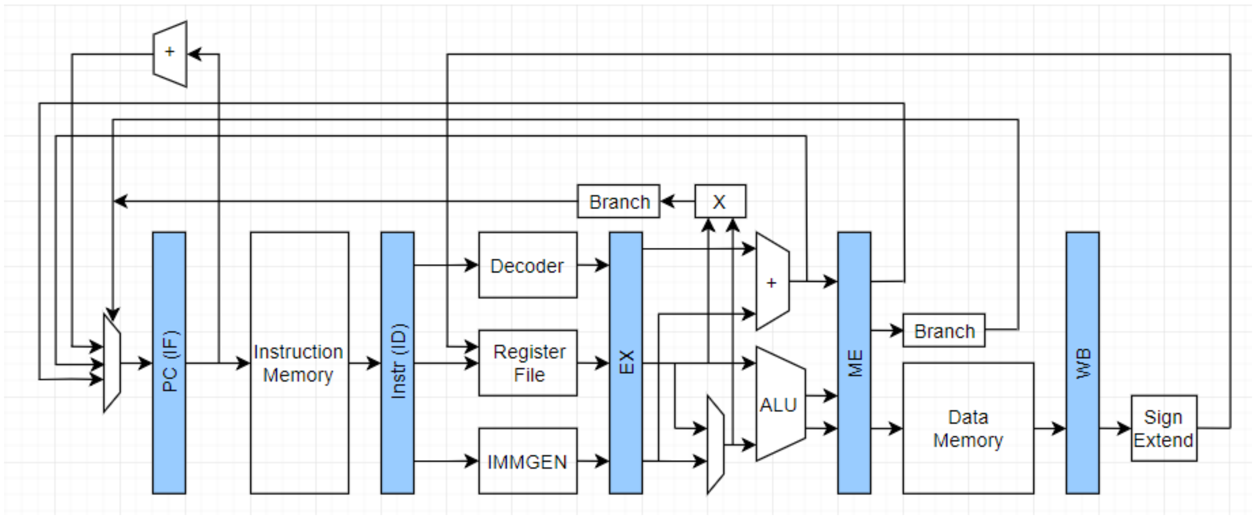

Figure 2

3. Additional Modifications for JAL

An additional optimization will be added for JAL. Since JAL does not use the ALU, the branch decision there can be made in the ID stage instead of the EX stage. Make a similar set of modifications to those specified in Section 2, changing only the behavior for JAL. Note that this will change some of the new logic created in Section 2.

4. Build the Hardware

Once the hardware changes are made, build SDK (ca) until the design is correct.

5. Run the Test Program

Once both the SDK (CA) builds successfully, the next step is to run the test program.

Import the phase's test by entering the following command on a terminal: git clone https://github.com/tamisil/phase9_test.git

This program is has a test that will thoroughly exercise the branch functions. First run phase9_test on the SDK(ca) of Phase 8, and note the number of clock cycles required. This number must be very close to 28,082 (within 1 or 2 cycles). Then run phase9_test on your phase 9 SDK (ca), and debug until it runs successfully. Just as with the other Phases, the test completes successfully if it runs to completion with no breakpoints, the value in register x10 is 58 (0x3a) and the value in register x25 is 1. In addition, the number of clock cycles required should be lower than with phase 8. If the number is not much lower, there is an error in the Phase 9 implementation.

It may be valuable to implement and test only the changes described in Section 2 first, then build, run and debug

6. Scoring

Phase 9 is worth 5% of the project score as a bonus (2.5% of the overall grade). There are no bonus or deduction points.

7. Exporting the Project

Once the test program is running for the correct number of cycles, the project's model folder should be downloaded and renamed to standardname9.tar