EE380-001 Singlecycle Lab Project

Implementor's Notes

PAUL GRUBBS, #########, and #########

This project had us implementing a total of 4 MIPS instructions (or psuedo-MIPS instructions, as MUL has some oddities to it in MIPS that we aren't worrying about here) into the provided single-cycle MIPS Verilog code. Those instructions were MUL, which isn't technically it's valid MIPS implementation but is instead a slightly more rational one; BNE, which is basically just the inverse of the Branch Equal instruction; LB, an instruction to load a byte from memory into a register in Little-Endian ordering (due to the fact that we are on the odd-numbered Team 7) while retaining the sign of the number; and finally LBU, an instruction to load a byte from memory into a register in Little-Endian but treat it as unsigned. For some of the instructions we were provided with hints on which implementations to use while for others it was more useful to reference the existing implementations of other similar instructions.

ACM Reference Format:
Paul Grubbs, #########, and #########. 2025. EE380-001 Singlecycle Lab Project: Implementor's Notes. 1, 1 (March 2025), 2 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 GENERAL APPROACH

Due to our prior experience working on the Multi-cycle design in our previous project we had to spend much less time aclimating ourselves to the general processor layout than we did before. It was still useful for us to go over all of the define-macros and already implemented instructions to familiarize ourselves with the codebase, but for the most part this time around much more of the focus was on directly getting to work implementing the instructions rather than getting used to writing/reading complex Verilog or understanding how the processor is laid out. We decided to divide up the workload so that one person (Paul Grubbs) handled the first two (& easier) instructions and worked on filling out the bulk of the Implementors' Notes, one person ([######################]) handled implementing the much more complicated load-bytes instructions which were close enough to being the same as each other that it didn't make sense to divide them up as two distinct jobs, and finally the last person ([######################]) created the test-cases for each instruction to ensure that they functioned appropriately and would help with solving any issues after the fact. We decided that we would try to have everything initially done by the end of Spring Break, and then take the remaining days before the assigment was due to run the test cases and ensure that everything worked. This meant that we could take our time with the bulk of the work, and we only had to worry about finishing things up before it was due. (though due to complications during the break for many of us we didn't end up hitting that deadline perfectly, and some of the initial implementation work had to be finished up before we could begin the testing phase)

For the two easier instructions it was most practical to simply build from the other pre-implemented instructions since both BNE and MUL behaved near-identically to other pre-existing instructions. For the MUL instruction we were allowed to use Verilog's builtin multiply (*) operator which meant that we could more or less reuse the logic for a similar instruction like ADDU, and simply change the operator it used in the ALU module. For the BNE instruction we could similarly reuse most of the logic from the BEQ instruction to make implementation much easier. The primary implementational difficulty lied in the load-bytes instructions, though the two instructions were close enough for almost all of the logic to be the same between them, meaning it was more useful to consider them as implementing only one instruction and one minor modification to that instruction. Since our group number was odd we had to implement them in Little-Endian and there weren't any easily referencable pre-implemented instructions to work from, but conceptually the instructions were fairly simple and were easy enough to follow along with and only needing to really think about implementing one of them meant that it was reasonably doable. The general process we used for implementing the LB and LBU instructions was [######################]. Creating the test cases was a bit less immediately intuitive, but thankfully the instructions we had to implement weren't that complex and we were able to trust our Verilog compiler to get much of the true complexity down reliably, so even if it was a bit less conceptually simple it was still a relatively straightforward process. Overall our process for desinging the test cases was [######################].

2 TESTING PROCESS

Since we were only worried about getting our basic Verilog implementations correct our testing process mainly just involved running the test cases and seeing if our instructions behaved correctly when run with the Icarus Verilog compiler's output. Our test cases and outputs are as follows

[######################]

Authors' address: Paul Grubbs; #########; #########.

XXXX-XXXX/2025/3-ART $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

3 ISSUES

[########## IF YOU HAD ANY OTHER ISSUES DURING YOUR INITIAL IMPLEMENTATION PROCESS ADD THEM HERE ##########]

After running our test cases we found that our instructions [##########]