**ECE 4250/7250**

**VHDL & Programmable Logic Devices**

**MizzouRisc : Shift to Least Significant Bit**

**Fall 2014**

**Angelique Taylor, Seth Kurtenbach, Dylan Samson**

**Introduction:**

The objective of this project is to develop a new MizzouRisc instruction “Shift to Least Significant bit” (stl). The syntax of this instruction is STL Rs, Rd. After this instruction is executed, the contents of Rs are shifted to the right, and the most significant bit is replaced by its previous value. An internal register is use to hold the contents of Rs. The contents of this internal register are copied into Rd with a shift, the least significant bit disappearing. The most significant bit from Rs is copied to the most significant bit of Rd.

The shift to least significant bit instruction was done by modifying the pre-existing MizzouRisc files. These files consisted of alu.vhd, controller.vhd, rom.vdh, and common.vhd. Additional opcode was added to these files to execute this instruction, in addition to providing the syntax, and the word organization.

**Methods:**

**Alu.vhd File**

The alu.vhd file is where the shift takes place. There was a modification made in the behave architecture in the alu.vhd file that consisted of adding an opcode for the shift to least significant bit instruction. Figure 1 displays the change made in this file on line 55. The sh\_reg signal is a 33 bit internal logic bit vector (in which the most significant bit is ignored) that is used to shift 31 of the left most bits in Rs to the 31 rightmost bits in the result. The most significant bit of Rs is appended to the result so that it is the most significant bit.



Figure 1: alu.vhd

Figure 1 shows the altered portion of the code, specifically line 55. Here, *result* is a 33 bit vector assigned bit values in the following way: We are concerned only with the right most 32 bits, so the first bit can be assigned any value. Here, we assign it sh\_reg(31), which is the most significant bit of Rs, the number being shifted. This value is concatenated to sh\_reg(31), meaning the 32nd bit of the new value is assigned the 32nd bit of Rs. To execute the shift, the value of *result*(*i)* is assigned *sh\_reg(i+1)*. Thus, Rd(0) = Rs(1), etc, resulting in a shift in the direction of the least significant bit.

**Controller.vhd File**

The controller.vhd file is used to develop the word organization of the new instruction. Because the word organization for the instruction to shift to the most significant bit is identical to the new instruction, it was not necessary to develop a new organization. Therefore, as shown is Figure 2 on lines 36 and 42, stl\_opcode uses the same opcode\_addr as stm\_opcode: 8. The word organization for both STM and the new STL is displayed in Figure 3.



Figure 2: controller.vhd. Lines 36 and 42 show identical opcode\_addr assignments for stl\_opcode and stm\_opcode.



Figure 3: Word Organization for STL, denoted by opcode\_addr 8.

**Common.vhd File**

The common.vhd file is used to set the unique opcode number for the new instruction. The opcode number that was chosen is opcode = 1; Therefore, the modictaion that was made on the behave architecture of the common.vhd file was adding the stl\_opcode standard logic vector, which contains four bits, is set to “0001.” This is displayed on line 8 in Figure 4. This is used to identify the new instruction when it is broken down into machine code and inserted in the rom.vhd file.

Figure 4: common.vhd

**Rom.vhd File**

The rom.vhd file is use to test the instructions in MizzouRisc. There were three simulated test. Each test was done by modifying the porgram1 architecture of this file. Each instruction sets up register R2 to be Rs and register R1 as Rd.

Test 1

In this test, immediate data “1234” is placed in register R3 low ( R3 = 0000 1234). Then immediate data, “8846” into register R4 high ( R4 = 8846 0000). The next instruction or’s R4 with R3 and place the result in R1 ( R1 = 8846 1234 ). The final step is to shift the contents of register R1 to the least significant bit

( R1 = C423091A ) . This code from the rom.vhd file is displayed in Figure 5.

|  |  |  |
| --- | --- | --- |
| Line Number: | MizzouRisc Instruction: | Machine Translation: |
| 24 | load l#1234, R3 | 83001234(h) |
| 25 | load h#8846, R4 | 84208846(h) |
| 26 | or R4, R3, R1 | 41340000(h) |
| 27 | Stl R1, R2 | 12010000(h) |



Figure 5 : Test 1

Test 2

The next test executes similar instructions, but with different immediate values. In this test, immediate data “4321” is placed in register R3 low ( R3 = 0000 4321). Then immediate data, “0FFF” into register R4 high ( R4 = 0FFF 0000). The next instruction or’s R4 with R3 and place the result in R1 ( R1 = 0FFF 4321 ). The final step is to shift the contents of register R1 to the least significant bit

( R1 = 07FFA190 ) . This code from the rom.vhd file is displayed in Figure 6.

|  |  |  |
| --- | --- | --- |
| Line Number: | MizzouRisc Instruction: | Machine Translation: |
| 24 | load l#4321, R3 | 83004321(h) |
| 25 | load h#0FFF, R4 | 84200FFF(h) |
| 26 | or R4, R3, R1 | 41340000(h) |
| 27 | Stl R1, R2 | 12010000(h) |



Figure 6 : Test 2

Test 3

The final test once again executes similar instructions, but with different immediate values. In this test, immediate data “F0F0” is placed in register R3 low ( R3 = 0000 F0F0). Then immediate data, “7722” into register R4 high ( R4 = 7722 0000). The next instruction or’s R4 with R3 and place the result in R1 ( R1 = 7722 F0F0 ). The final step is to shift the contents of register R1 to the least significant bit

( R1 = 3B917878 ) . This code from the rom.vhd file is displayed in Figure 7.

|  |  |  |
| --- | --- | --- |
| Line Number: | MizzouRisc Instruction: | Machine Translation: |
| 24 | load l#F0F0, R3 | 8300F0F0(h) |
| 25 | load h#7722, R4 | 84207722(h) |
| 26 | or R4, R3, R1 | 41340000(h) |
| 27 | Stl R1, R2 | 12010000(h) |



Figure 7 : Test 3

**Results:**

**Discussion:**

**Conclusion:**