Final Project – Progress Report 2

CS 385 – Computer Architecture

Daniel Kostecki, Thi Nguyen, and Sonia Leonato Soiras

**Tasks:**

Daniel Kostecki:

* Gate level conversions in the regfile
* Regfile implementation in project
* Report and adjusted diagrams
* Code comments

Thi Nguyen:

* 16-bit implementation of multiplexors using 4x1 mux
* Regfile D-flip-flops
* CPU module
* Main Control Unit
* I-type operations for Report 2
* Testing program

Sonia Leonato Soiras:

* ALU conversions to 16-bit
* Gate level conversions in the ALU (4x1 mux and 2x1 mux)
* Truth table
* Code debugging
* Testing program

**Descriptions:**

The instruction set architecture currently in place for our 16-bit machine allows for R-type and I-type instructions. The list of R-type instructions that are currently supported and their respective opcode are and(0000), sub(0001), add(0010), or(0011), and slt(0111). The opcodes of the I-type instructions are addi(0100), lw(0101), sw(0110), beq(1000), bne(1001).

R-type format: I-type format:

|  |  |  |  |
| --- | --- | --- | --- |
| Op | Rs | Rt | Address/Value |
| 4 | 2 | 2 | 8 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Op | Rs | Rt | Rd | Unused |
| 4 | 2 | 2 | 2 | 6 |

The format of the instructions is dependent on the type if instruction. For example, in the r-type format we first get the op code (listed above). This is followed by the two registers which are being used. The final register is the destination. Because we have a 4-bit opcode followed by 3 2-bit registers this leaves us with 6-bits at the end of every r-type that is unused. For the i-type format this changes. The op code is still 4-bit, and there are still two registers (rs being used and rt being destination) but this is followed by an 8-bit value or address. For our current implementation, this will always be a value, as we have only implemented the addi instruction presently.

**Logic Diagrams/Truth Tables:**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Instr** | **RegDst** | **ALUSrc** | **MemReg** | **RegWrite** | **MemRead** | **MemWrite** | **Branch** | **ALUOp1** | **ALUOp2** |
| **Rtype** | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 |
| **lw** | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| **sw** | X | 1 | X | 0 | 0 | 1 | 0 | 0 | 0 |
| **beq** | X | 0 | X | 0 | 0 | 0 | 1/0 | 0 | 1 |
| **bne** | X | 0 | X | 0 | 0 | 0 | 0/1 | 0 | 1 |
| **addi** | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |

**\*\*\* Missing logic diagram \*\*\***

**Verilog Code:**

**Test results:**