ALU Sim

Background

The use of co-simulators has become more prevalent in their use as opposed to more expensive proprietary simulators in industry. See these blog posts from the RISC organization in 2021 and 2023. This has allowed for the software industry to keep pace with the CPU design steps using tools such as cocotb.

Project Steps

There are three main parts to this project.

1. Puzzles (20)
2. Comparisons (15)
3. Multiplication (40)
4. Division (40)

For multiplication and division, all the calculations must be completed using 32-bit registers. Modify your algorithms accordingly. Assume that all inputs are positive numbers.

All comparison checks should first be calculated via the ALU. All comparisons completed in native python syntax (ex. `if dut.zero.value`) should be completed "C-style" where any number with a value other than zero `0` is `True`.

Puzzles

Puzzles are individually 5 points each, but will be required for the successful completion of the multiplication and division problems.

- `perform_not`: Perform the logic operation `~`. This is not a RISC instruction, so a method has been provided for you to wrap the ALU operations.
- `perform_negate`: Perform the two's complement negation. A wrapper method will be provided. You may re-use the `not` method.
- `perform_sub`: The `sub` instruction was not provided to you, but exists with the use of the field `funct7`. Use the `perform_negate` method to pipeline this instruction in this wrapper method.
- `set_gt`: The logical check for greater than or equal '>' is required for greater than or equal `>=`. This operation does not exist directly as an instruction however, the logic must exist since there is a `bg` instruction for branching. Create the logical equivalent using the ALU.
- `set_gte`: The logical check for greater than or equal `>=` is required in the algorithm for division. This operation does not exist directly as an instruction however, the logic must exist since there is a `bge` instruction for branching. Create the logical equivalent using the ALU. You may use the function `set_gt` to pipeline your instruction.

Perform the operations such that the next instruction clocks out the result on signal `d` or `zero`.

Comparisons

Two comparisons have already been completed as part of the puzzles. These will be included as the base instruction set. However, if we are interested in floating point operations the accompanying instruction set will have to be implemented. These comparisons make use of the `exp` and `frac` fields, but will still assume a positive number.

- `f_set_e`: The logical check for two floating point numbers being equal `==`.
- `f_set_lt`: The logical check for floating point less than `<`.
- `f_set_lte`: The logical check for floating point less than or equal `<=`.

Perform the operations such that the next instruction clocks out the result on signal `d` or `zero`.

Multiplication

In this problem, you will perform multiplication as described by the instruction `mul rd, rs1, rs2` using the algorithm described in class from Chapter 3.3. You will require an `if-else`, however you may not write a conditional statement in your `python` code. You must use the output from the ALU's `d` or `zero` signals in the conditional. This relies on `python`'s use of `0` for `False` and `1` or any value other than `0` as `True`. This is the same logical condition used in `C`.

There are `7` different steps of the multiplication algorithm as described in the text, allowing `5` points for partial credit.

Division

In this problem, you will perform division as described by the instruction `div rd, rs1, rs2` using the algorithm described in class from Chapter 3.4. The requirements for the conditionals in this problem are the same as for `multiplication`. One added complexity is the requirement of the conditional for greater than or equal (`>=`).

There are `8` steps to the division algorithm as described in the text.