CMSC 411—Project—Part b

For this second phase, you will be building an 8-bit ALU that implements the core requirements of the MIPS instruction set. The R-mode instructions that are handled by the ALU include:

- `add` – opcode/func: 0 / 20_{hex}; $d = $s + $t, w/overflow detect
- `addu` – opcode/func: 0 / 21_{hex}; $d = $s + $t
- `sub` – opcode/func: 0 / 22_{hex}; $d = $s - $t, w/overflow detect
- `subu` – opcode/func: 0 / 23_{hex}; $d = $s - $t
- `and` – opcode/func: 0 / 24_{hex}; $d = $s & $t
- `or` – opcode/func: 0 / 25_{hex}; $d = $s | $t
- `xor` – opcode/func: 0 / 26_{hex}; $d = $s ^ $t
- `nor` – opcode/func: 0 / 27_{hex}; $d = ~($s | $t)

There are just a few more, but the above are the only ones you will be implementing for this project; if you can accomplish this subset, you can feel confident that you have implemented a very capable ALU. Also, there are the immediate mode I-format versions, but recall that these differ from the R-mode instructions only in terms of where the input data is channeled from, so that part does not impact the ALU design.

Note that the opcodes for all of these instructions is 0, so the instruction decode logic will examine the "funct" part of the instruction to decode what should be done. All ALU instructions have an opcode of 0, indicating it is an R-format instruction. You are not responsible for this decoding logic. Notice that the function codes are in a narrow range: they are all between 20_{hex} and 27_{hex}F The instruction decode step will therefore just route the least significant bits of the instruction to your ALU. Assume that only the least significant 3 bits are routed to the ALU's control lines. (The true picture is somewhat more complex—I hope you don't mind my simplifying for this project ☺)

For this project, you will mainly be using low-level gates: AND, OR, NOT, XOR, etc.. You are allowed to use the full adder from the library, but you cannot set the "data bits" attribute above 1, meaning you must manually connect 8 of these together. You are not allowed to use the subtractor, though! You can even use multiplexers and decoders if you find them useful, but they must be at most 4-channel (i.e., no more than 2 selection bits)! Since the function code is 3 bits, i.e., 8-valued, using a 4-channel multiplexer is a challenge. And, no, you cannot gang multiple muxes together to simulate a larger mux—already thought of that.

Note that the ALU is a purely combinational circuit: you should not need any clock signals, flip flops, and the like. Remember the core design principle of an ALU: different parts of the circuit might be/usually are computing different answers (one part of the circuit might be computing the sum, while another is simultaneously computing the AND of the same inputs), and the second part of the circuit often just chooses which answer to present on the output. In some cases, though, the difference between two operations is implemented by modifying the input but using the same data path. For example, to implement ADD vs. SUB, since the latter can be

implemented via “add the 2’s-complement of the subtrahend”, so you would modify input B before routing it through the same full adder circuit as you would with addition.

The Assignment

You will be constructing an ALU that functions much the black box we’ve been using in our architecture diagrams for the last several weeks. However, like our register file from the first project, it will be scaled down a bit: you will only be implementing the R-format instructions, and not all of them. You will also *not* have to support multiplication or division—whew! You will support a word size of 8 bits, but with Logisim, that is not too difficult to scale. The circuit should have two input buses, representing the $s and $t inputs ($t is the addend/subtrahend). It should have an output bus that sends out the result of the calculations. It must also have an OVERFLOW line, which should be high IFF the command is sub or add, and the result overflowed/underflowed (remember the carry-in/carry-out rule from the beginning of the semester).

In detail, the inputs to your circuit should be:

- S-in: Input value $s: the addend1, or minuend: value to be added to/subtracted from
- T-in: Input value $t: the addend2, or subtrahend: value to be added or subtracted.
- Op: the low 3 bits of the “FUNCT” part of the instruction.

And the outputs from your circuit should be:

- D-out: The result of the requested ALU operation
- OF: overflow/underflow indicator. Should only be high when instruction is “add” or “sub”, and overflow/underflow has occurred. So, for example, this should never be high if the instruction was “addu”.

You have (almost) complete freedom to design the circuit as you wish, modulo the restrictions on adder and multiplexer widths described earlier. Here are some hints, though:

- Note that for (almost) all operations, each bit column is independent of the others. This is definitely true of logical operators such as AND and OR, but it is also true of ADD: for each 1-bit full adder, note that it takes the corresponding bits from the two inputs, its carry-in is from the previous bit adder, its carry-out goes to the next adder. Also note that for SUB, creating the 2’s complement of the subtrahend consists of inverting each bit of input T-in, and you can take care of the “add 1” part, not by treating bit 0 of the input specially, but by using the carry-in trick for the bit-0 adder. This means that you can create the circuit design for a single column, put it in a black box, then paste 8 copies of this into your circuit.
- It *might* be easier, depending on the rest of your design, to use one of two alternatives for routing control signals through your circuit: (a) a set of Boolean logic circuits to explicitly test for one or more specific patterns that go to various subcircuits; or (b) a simple decoder to convert the 3-bit function code into separate 8 control lines, each one “on” for a given function. Each alternative causes a different kind of complication because your output multiplexer is only allowed to be 4-input at most.

By default, Logisim starts up with the layout pane opened to the "Untitled→main" circuit. You should create your register file as new circuit at the top level (just right-click on the "Untitled" folder in the left pane, then select "new circuit…"), and call it "ALU411". That way, you can more easily integrate it as a component in other circuits.