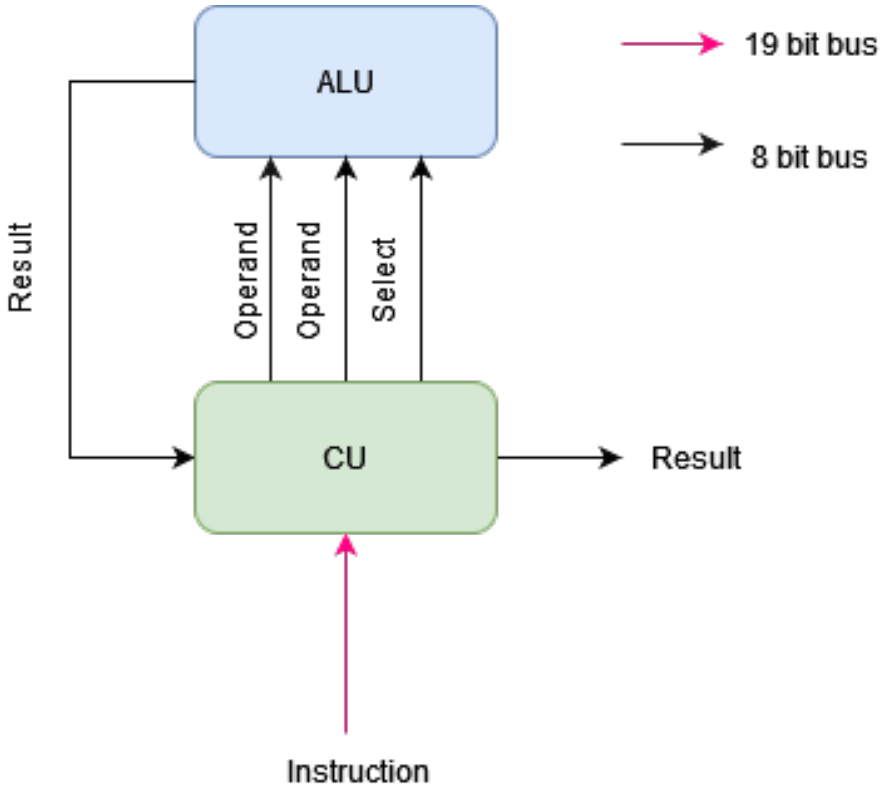
CS2310 — Verilog Lab Assignment
Creating a very simple CPU

Diwali 2022

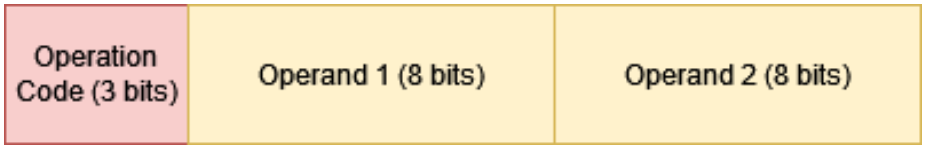
1 What our simple CPU looks like - a top level view?



In our simple CPU the CU (control unit) will read the instruction, and provide the operands and select line (which are used to determine the operations to be performed) to the ALU (arithmetic logic unit). It will then collect the result that the ALU generates, which is it's output.

The ALU on the other hand will use the select lines to determine the operation to be performed on the two (or one) operand, and give out a result. The details of the operations available in our ALU and the instruction format are in the next section.

2 What does the instruction look like?



The first 3 bits of our instruction are the operation code, since there are 3 bits we can implement a total of 8 operations here, though we will implement 7 operation here (feel free to implement an additional operation) which are as follows (with there respective Operation Codes):

- ADD - 001
- SUBTRACT - 010
- INCREMENT - 011
- DECREMENT - 100
- AND - 101
- OR - 110
- NOT - 111

The remaining 16 bits of the instruction are divided into 2 operands of 8 bits each. The operand represents the actual value (usually operands represents an address to memory, in which the actual value of the operand is stored, but in our simple we are not implementing memory). Also not that some operands like INCREMENT (011), DECREMENT (100), and NOT (111), need only one operand, in such cases we will ignore Operand 2 (i.e the last 8 bits)

3 The problem statement

The problem statement of the assignment is to create a simple cpu as define in Section 1, that follows the instruction set as described in Section 2, but with the below restrictions:

- You can only use structural description (i.e define interconnection between modules). Using procedural description (with if-else, while, case, etc) and functional description (writing the logic as a boolean expressing) is not allowed.

- In case of structural description, you can use the predefined basic modules like and, or, not, xor, buf, etc only. Other modules like adders, multiplexers, decoders, etc needs to be designed by you.

The above restriction are there so that you may appreciate a very fundamental concept in computer science (both in hardware and software) which is **abstraction** The test-bed for you CU may look something like this:

```
4   module cu_v2_tb;
5
6       reg [18:0] instruction;
7       wire [7:0] result;
8
9       cu_v2 uut (result, instruction);
10
11      initial begin
12
13          $dumpfile ("cu_v2_tb.vcd");
14          $dumpvars (0, cu_v2_tb);
15
16          instruction = 19'b0010010001100010100; #20; //Addition
17          instruction = 19'b0100010001100010100; #20; //Subtraction
18          instruction = 19'b0110010001100010100; #20; //Bitwise And
19          instruction = 19'b1000010001100010100; #20; //Bitwise Or
20          instruction = 19'b1010010001100010100; #20; //Bitwise Not
21          instruction = 19'b1100010001100010100; #20; //Increment
22          instruction = 19'b1110010001100010100; #20; //Decrement
23
24          $display ("Test Completed");
25      end
26
27  endmodule
```

4 A helping hand

Here we want to help you out with what might the components inside the ALU and CU look like, please feel free to implement it in any way you want as long as you fulfil the problem statement in Section 3.

4.1 Control Unit

The control unit is really simple in our design, the only major component needed here will be a 3 X 8 decoder to convert the 3 bit operation codes to the 8 bit select lines, that are needed as input for the ALU

4.2 Arithmetic Logic Unit

Here there are few things to consider:

- All the arithmetic operations, like ADD, SUBTRACT, INCREMENT, and DECREMENT can be done using a single 8 bit Adder.
- Note that subtraction is just the addition with 2's complement, so if subtraction is selected we need to find the inverse of the second operand
- INCREMENT is just addition with 8 bit 0, with carry in to the adder being 1. So we need to convert the second operand to all 0s (no matter what is passed in the second operand) and pass carry in as 1.
- Similarly in DECREMENT, 2's complement of 1 (which we will add to the 1st operand) has a special property, which can be exploited like in INCREMENTS case.
- AND, OR, NOT, are basic logic operations, only that it is bit-wise for 8 bits.
- You might require some multiplexers to select the output depending on the select lines

Beyond this you are on your own! Try to figure out a optimal design, and I hope the above tips will help you.
All the Best!!