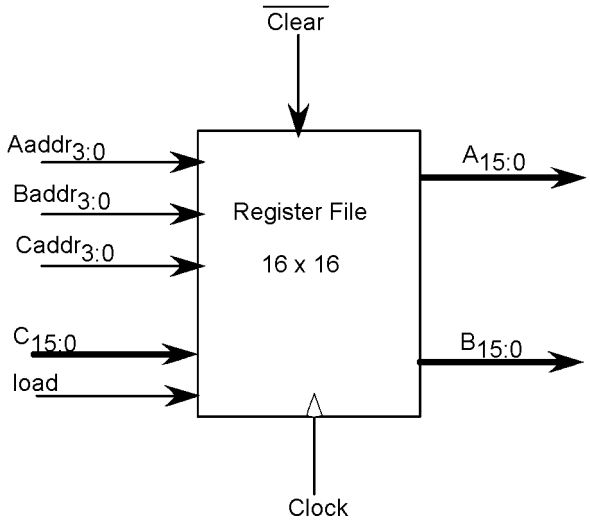
Milestone 2

Objective

The objective of this lab is to develop behavioral models for a register file and an ALU. These components are basic building blocks for your CPU.

Lab Exercise

Develop a behavioral model of a single input, dual output register file. A block diagram of the register file is shown below.



The following module definition defines the template interface for your register file:

```
module reg_file(
     A,                  -- input data port
     Aaddr,              -- register select for input A
     B,C,                --  output data ports
     Bddr,Caddr,         -- register select for outputsB and C respectively
     load,               -- synchronous load enable
     clear,              --  asynchronous, negative logic clear signal
     clk);               --  rising edge triggered clock
endmodule
```

Signals A, B and C represent the single data input and dual data outputs respectively. Caddr, Aaddr, and Baddr are the register numbers for storing input C data and for outputting A and B data, respectively. When the load signal is asserted, a rising edge on clk causes the data input on C to be stored in the register identified by Caddr. If the load signal is not asserted, a rising edge has no effect. When the clear signal is asserted, all register values should be set to 0. Note that clear is asynchronous and is negative logic.

Implement the behavioral module for the register file entity. Represent your registers as an array of words and use a structured procedure to update register contents and register file outputs.

Implement a test bench for your register file to demonstrate correctness.

Project

Develop a behavioral model of a sequential ALU to support the KURM instruction set (see previous lab sheet). Your ALU should perform operations that support the mathematical and logical instructions as well as offset calculation for memory access and branching. To support these requirements, your ALU should minimally implement:

1. Unsigned addition of 16-bit numbers
2. 2s-complement addition/subtraction of 16-bit numbers
3. Logical “and”
4. Logical “or”
5. Set on less than
6. Branch if not equal

In addition to a result, your ALU should generate control outputs for word equivalence and word less than. Your ALU should provide for carry in, carry out and a 2s-complement overflow indicator. The following module defines a template for your ALU:

```
module alu (
    x, y,       -- dual inputs
    z,          -- single word result
    c_in,       -- carry in
    c_out,      -- carry out
    lt, eq, gt,                 -- comparison indicator bits
    overflow,                   -- overflow indicator
    c);         -- operation select
endmodule;
```

The x and y inputs and the z output represent the two data inputs and one data output of the ALU. The `c_in` and `c_out` bits represent carry in and carry out, respectively. The three comparison indicator bits, lt, eq, and gt, are asserted when $x < y$, $x = y$ and $x > y$ are true, respectively. The overflow bit is asserted when the add or subtract operations generate an overflow, as defined in standard 2s-compliment mathematics. Finally, c is a vector of three control bits that select the operation the ALU will perform.

Use the encoding for control inputs shown in the table below. You may also add other operations that you feel might be helpful in designing your CPU. Further note that the comparison bits are always output without regard to the operation being performed.

ALU Control Lines	Function
0 0 0	And
0 0 1	Or
0 1 0	Add
0 1 1	Subtract
1 1 1	Set-on-less-than

Implement a test bench for your ALU model to demonstrate correctness.