Computer Architecture Lab 6

**CE/CS 321L/330L Computer Architecture and Organization**

**Lab 6: Core Processor Components – ALU and Register File Design**

**Objectives**

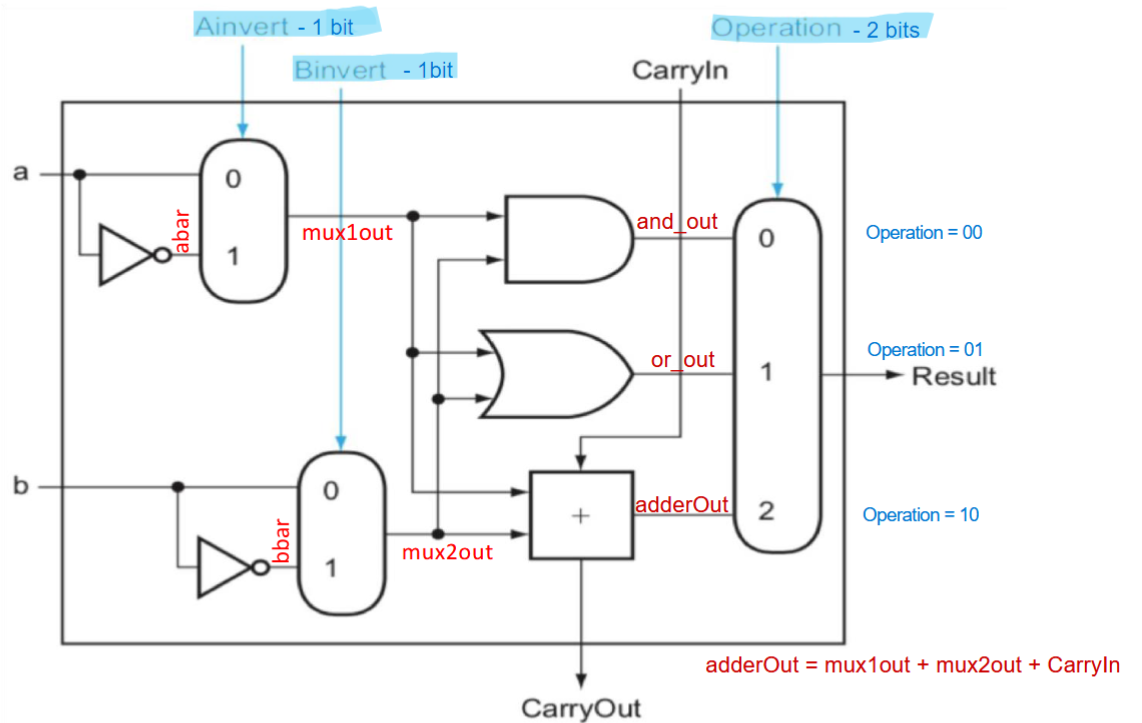
This lab aims to design and observe the functionality of processor components ALU and register file. Specifically, students will develop and interconnect a state machine that is capable of reading from a register file, computing output through ALU unit, and writing back to the register file. This lab combines the previously developed modules (debouncing logic, seven-segment display) to form a cohesive system.

**a) Introduction**

In this lab, you will integrate multiple processor modules to form a functioning 64-bit system. This includes the control logic that manages operation sequencing, address incrementing for register access, and the arithmetic logic unit (ALU) for performing operations.   
  
Previously developed modules such as the Debouncer, Delay Counter and Seven-Segment Display are reused here. New modules include State Control, Delay Counter, Address Counter, Register File, ALU Operation Generator, and the ALU itself.

**b) Task 1: Designing the Arithmetic and Logic Unit (ALU):**

The arithmetic logic unit (ALU) is the brain of the computer, the device that performs the  
arithmetic operations like addition and subtraction or logical operations like AND and OR. This  
lab constructs an ALU from four hardware building blocks (AND and OR gates, inverters, and  
multiplexors) and illustrates how combinational logic works.  
Because the RISC-V registers are 64 bits wide, we need a 64-bit-wide ALU. Let’s assume that we  
will connect 64 1-bit ALUs to create the desired ALU. We’ll therefore start by constructing a 1bit  
ALU, shown below in Figure 6.1:



(including the rest from the original lab 6)

Subtasks:

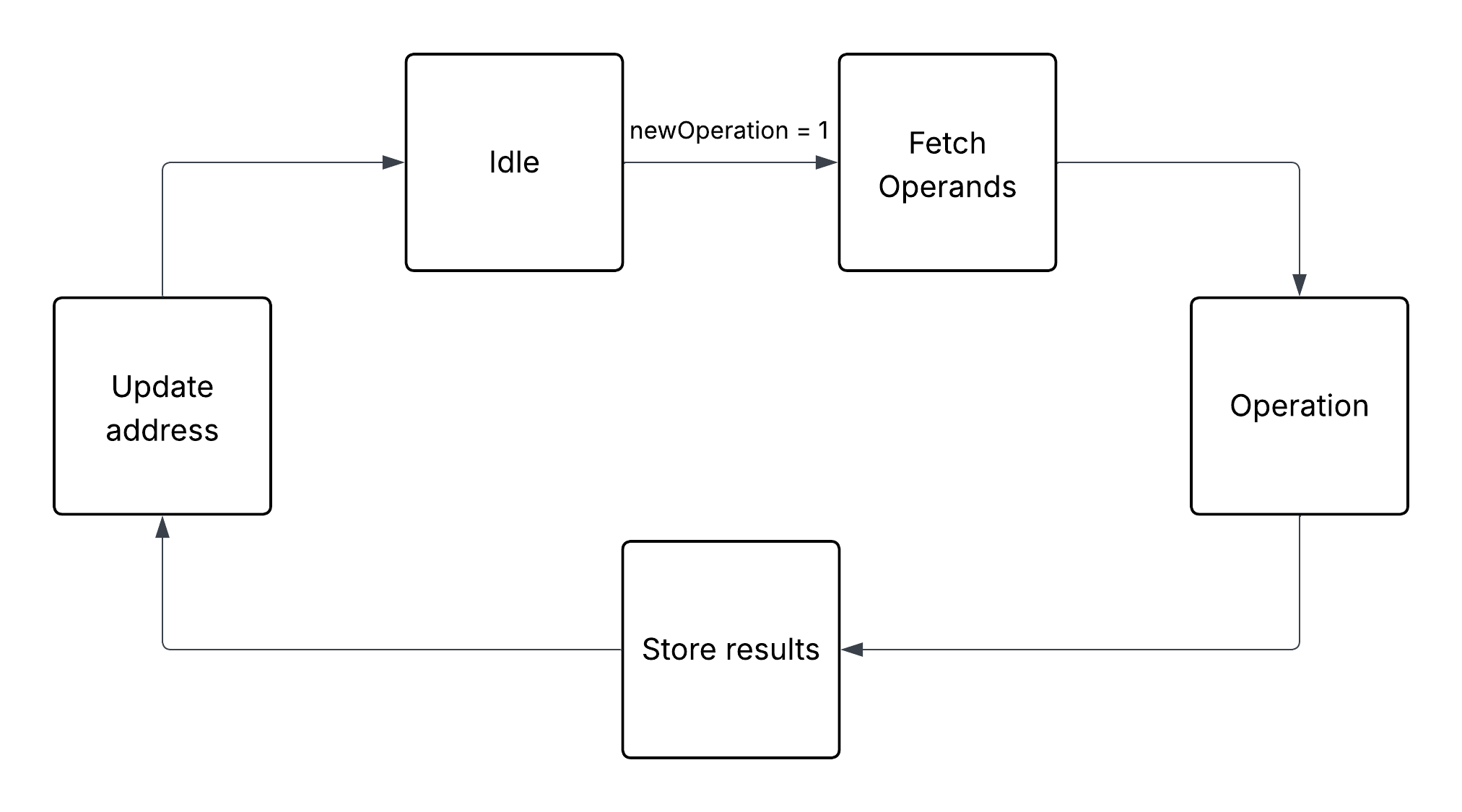
* Design 1 bit ALU
* Design and test 8 bit ALU
* Design and test 64 bit ALU

**c) Task 2: Register File**

**c) Task 3: State Control Unit**

In this task, you will design and implement the State Control Unit. It uses a finite state machine (FSM) to manage signals that control data flow and operation timing.  
The control unit ensures that operand fetching, ALU operations, and result storage occur in the correct sequence.

What the FSM can look like (where newOperation is an external signal):



You may use a test bench to verify that the FSM works as expected.

**d) Task 4: Delay and Address Counters**

The FPGA clock operates on 100MHz. If operations are not spaced out correctly, you may never be able to observe each computation result. As a result, we introduce delays to be able to observe each output.

|  |
| --- |
| We space out each operation by 10s. A pulse (newOperation) must thus go high every 10s. Delay is calculated to be 1000000000 clock cycles. |

(The delay module from the previous lab can be reused).

Design also an address module, that increments address by 2 each time an indicator signal (incrementAddress) is high.

Module Interface Example (Address Counter):

***module addressCounter (***

***input clk, rst, incrementAddress,***

***output [4:0] address***

***);***

**e) Task 5: Top Module Integration and 7-Segment Display**

In this final task, you will integrate all developed modules into a single top module. The top module connects the debouncer, state control, counters, register file, ALU, and display unit. Upon pressing the control button, a new operation cycle begins, and the resulting computation is shown on the seven-segment display.

Top Module Interface Example:

***module top(  
 input clk, btnC,  
 output [6:0] seg,  
 output [3:0] an,  
 output [15:0] led  
);***