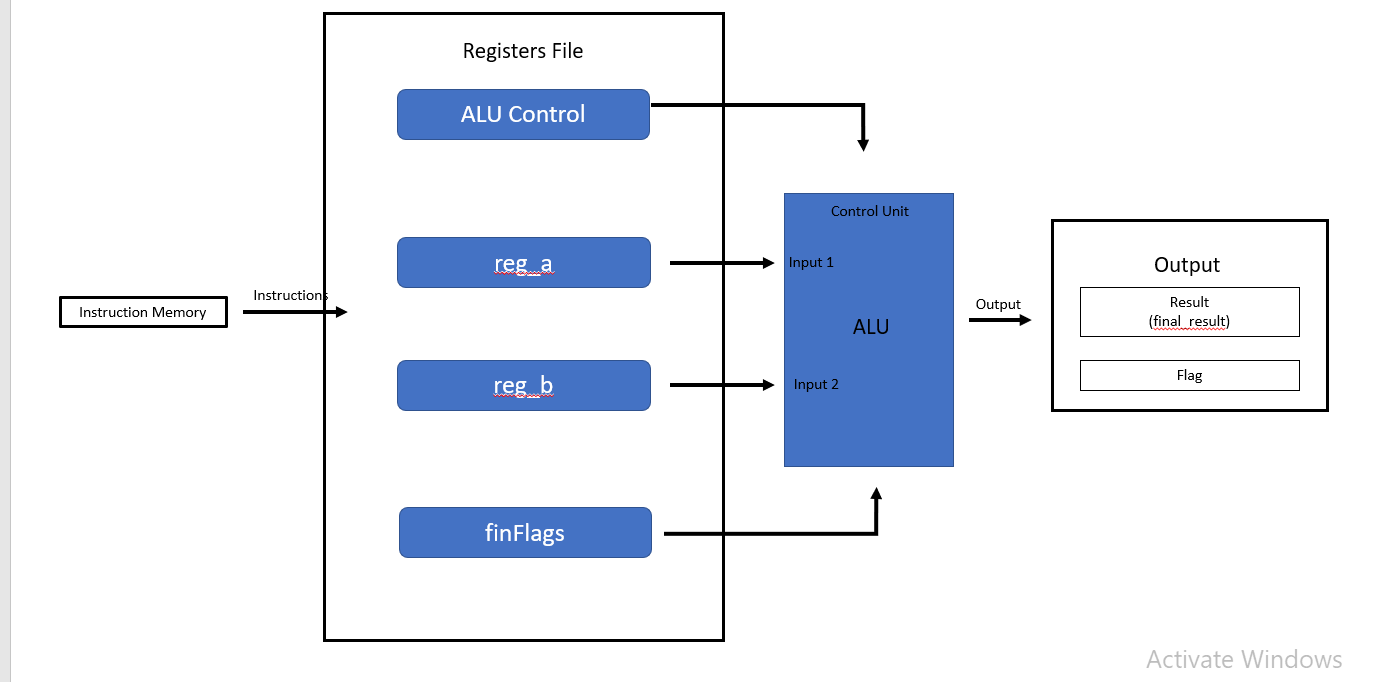
Project 2 Computer Science 3050

1. Flow Diagram



1. About

This program is going to implement a simple CPU with simple instruction parsing, Registers, and ALU functions.

1. Big Picture

From the project, I learn how to create an implementation of ALU and its functions by using Verilog. First this programme will do simple parsing on every machine code from MIPS instruction set. After the parsing process, ALU will receive fixed input. For example: 6-bit opcode, 2 32-bit register, etc. Next, we need to fetch RS and RT from the instructions. Then we need to create case for every opcode given with their corresponding functions. For example: A simple ADDU instruction set (RS + RT). After all case for every function is made, the program is basically done. Next, is to create a test file to confirm whether the function is all sync and functional.

1. Registers
2. Input Register

* Input [31:0] instCode – 32-bit register input instruction code register
* Input [31:0] reg\_a – 32-bit register for input register A
* Input [31:0] reg\_b – 32-bit register for input register B

1. Output Register

* Output [31:0] res – 32-bit register output for final ALU result
* Output [2:0] flag – 3-bit register output that shows final flags

1. ALU Control

* Reg [5:0] opcode – 6-bit register operator code used in R-type instruction set
* Reg [5:0] func – 6-bit register function used in I-type instruction set
* Reg [31:0] zero\_extend, sign\_extend

In I-type mode, a 32-bit register for a special instruction. The sign\_extend and zero\_extend functions are sign extensions that load the most important bit into the remaining bits. If sign\_extend is used for signed instructions (which can be either 1 or 0), zero\_extend is used for unsigned instructions (which can be either 1 or 0).

1. Output

