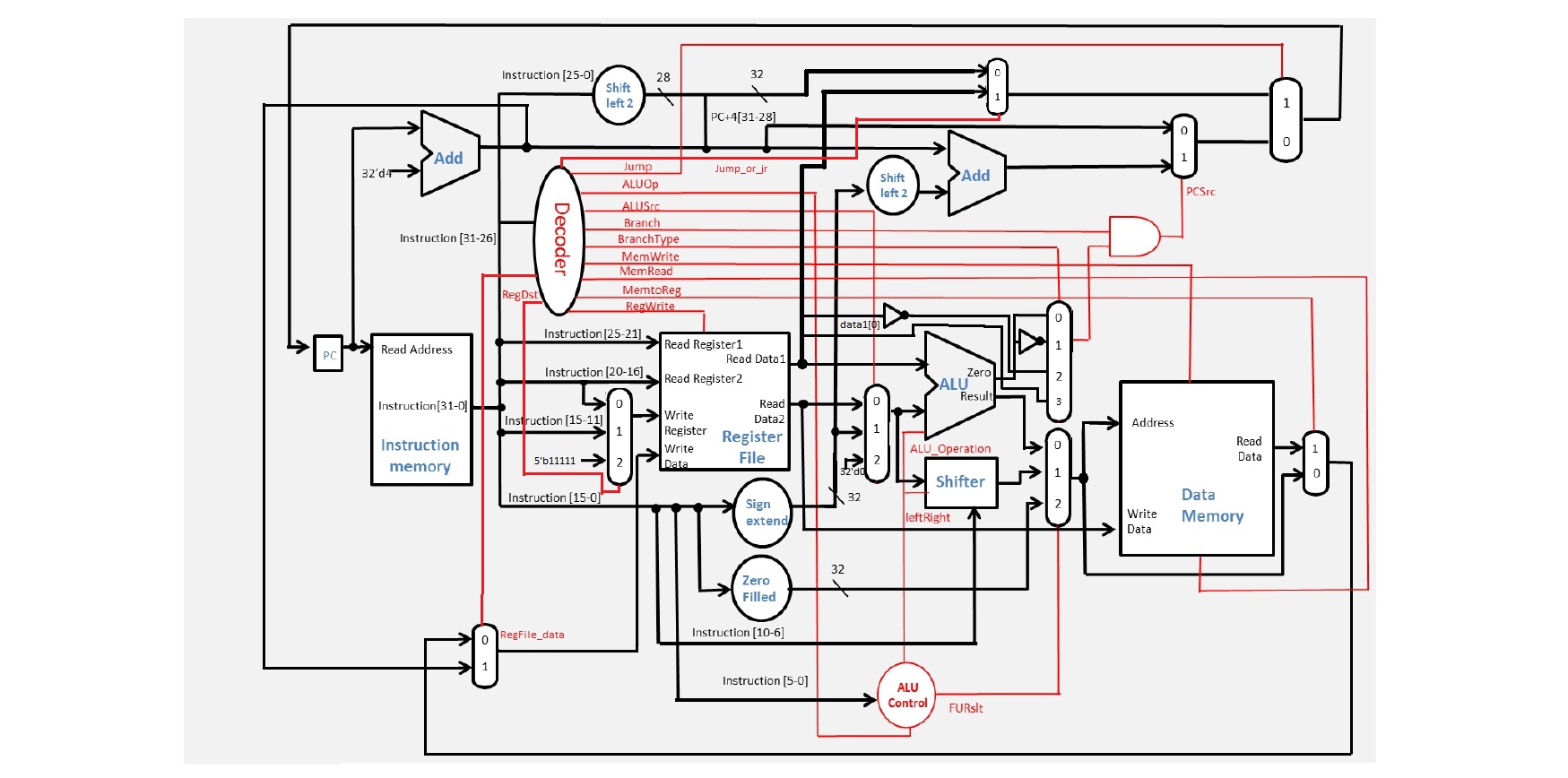
**Computer Organization**

**Student ID: 0416106 Name: 彭敬樺**

**Architecture diagrams:**

**Hardware module analysis:**- Adder: This module is used to calculate the sum of 2 31-bit value.  
- ALU: Arithmetic logical unit in a CPU. Used to calculate the result needed for the instruction, such as: add, sub, and, or, nor, slt, addi, and etc.  
- ALU\_1bit: Parts of the module included inside the module “ALU”, used to calculate only 1 bit value.  
- ALU\_bit31: The last bit module in the module “ALU”, usage is quite the same with the normal 1-bit ALU, however, with the additional function to determine zero and overflow.  
- ALU\_Ctrl: Used to give instruction to the ALU, so that it knows what to be calculated from the given machine code. ALU\_Ctrl provides ALU with 4-bit value to maintain its correctness in calculation result.

- Data\_Memory.v: Used to access, write, and read the value in the wanted memory address.  
- Decoder: Used to control the multiplexer and ALU\_Ctrl to suit the condition needed by the given machine code.  
- Full\_adder: Similar to adder, however, have the capability to receive and return carry.  
- Instr\_Memory: Module used to catch the instruction given its PC.

- Jump\_address.v: Module designed to calculate the desired address if jump instruction is used.  
- Mux2to1: A multiplexer to choose 1 value from 2 existing choice.  
- Mux3to1: A multiplexer to choose 1 value from 3 existing choice.

- Mux4to1: A multiplexer to choose 1 value from 4 existing choice.  
- Program\_Counter: The parts of the module that will maintain the value of the PC so that instruction can progress in the wanted order.  
- Reg\_File: Parts of the Simple\_Single\_CPU, used to get the data in the given address of its storage. It also have the function to write a data into a given address.

- Shift\_left\_2.v: Module used to calculate the offset of address given by the constant parts of the instruction.  
- Shifter: Shift the value given as input left or right as many as the given amount of shamt.  
- Sign\_Extend: Extend the sign bit of the 16-bit value into 32-bit.  
- Simple\_Single\_CPU: The modules that wrap all finished module to gain the wanted result.  
- Zero\_Filled: Fill the lower 16-bit value with 0(zero) and fill the upper 16-bit with input value.

**Finished part:**There are 21 modules used to make this project:  
- Adder.v  
- ALU.v  
- ALU\_1bit.v  
- ALU\_bit31.v  
- ALU\_Ctrl.v

- Data\_Memory.v  
- Decoder.v  
- Full\_adder.v  
- Instr\_Memory.v

- Jump\_address.v  
- Mux2to1.v  
- Mux3to1.v

- Mux4to1.v  
- Program Counter.v  
- Reg\_File.v

- Shift\_left\_2.v  
- Shifter.v  
- Sign\_Extend.v  
- Simple\_Single\_CPU.v  
- TestBench.v  
- Zero\_Filled.v

**Problems you met and solutions:  
-** Problem: Mistaken the value of ALUSrc when branch-like instruction is used.

Solution: Change how the decoder intercept the instruction branch and recalculate.   
- Problem: Bit number mismatch when connecting one module with another  
Solution: By analyzing the required bit length that is given by the output module with the one requiring the value.

**Summary:**

In this project, I have no major problem in completing it. Declaring the number of bit for every variable can be a hard task due to the number of wire needed to be maintained after the size of CPU increases to this stage. Due to the existence of many control signals, when a certain instruction is being calculated, value from the other parts of the CPU will not reach and disrupt the value of existing register or memory. Decoder takes the major position to ensure condition above could happen by tweaking the value of multiplexer’s select wire.

In conclusion, detailed and elaborate observation is the key to complete this assignment. Due to the size of the hardware needed to be maintained, mistake is quite intolerable.