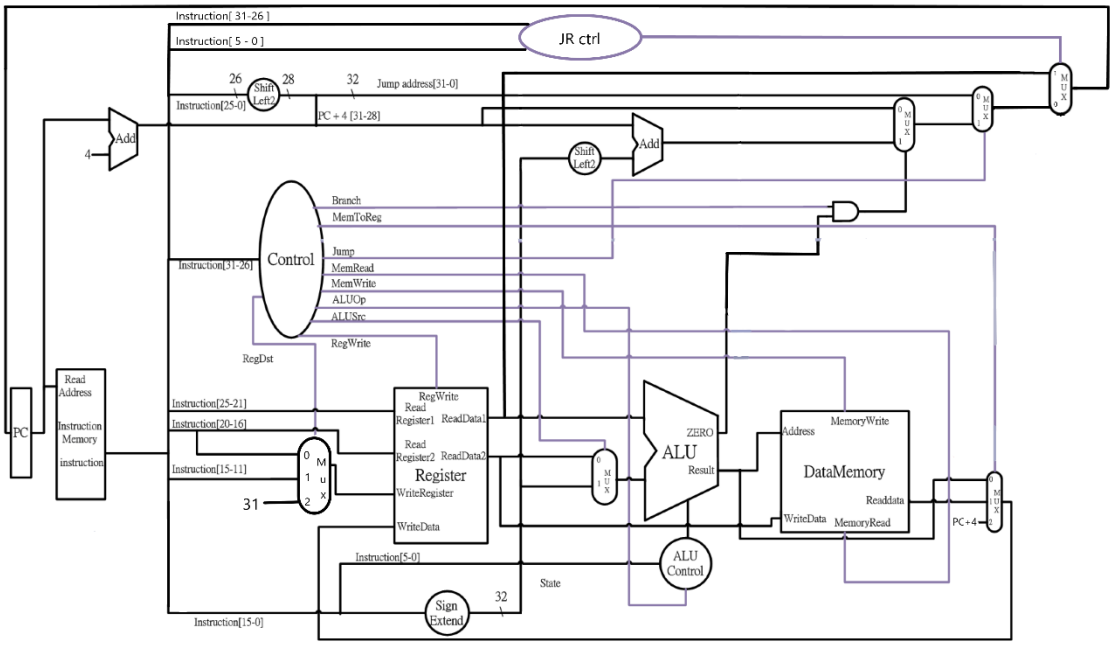
Computer Organization

Architecture diagrams:



Hardware module analysis:

- Top module: Simple_Single_CPU
  - Program Counter: “count” for the next instruction that needs to do in the following clock cycle.
  - Adder: there is 2 adder in this hardware-
    - first: PC+4
    - second: add the value of Program Counter for the next instruction with the address stated by instruction[15:0], when is the instruction that specified by branch operation.
  - Instruction memory: fetch the instructions from the test file according to the Program Counter.
  - Decoder: decode the opcode of the instruction and output to other module for use.
  - ALU Control: further decode the function code of the instruction and output to ALU for use.
  - Register File:performs the functions of getting the value from register according to the instructions and storing the result of ALU into the

register specified by current instruction.

- Sign Extend: extend the 16-bit of address stored in instruction into 32-bit.
- ALU: perform some operations such as
  - and
  - or
  - addition
  - subtraction
  - set on less than
  - nor
- multiplexor: for selecting the input according to the selector.

Result:

test data 1:

```
PC =          x
Data Memory =       1,      2,      0,      0,      0,      0,      0,      0
Data Memory =       0,      0,      0,      0,      0,      0,      0,      0
Data Memory =       0,      0,      0,      0,      0,      0,      0,      0
Data Memory =       0,      0,      0,      0,      0,      0,      0,      0
Registers
R0 =        0, R1 =     1, R2 =     2, R3 =     3, R4 =     4, R5 =     5, R6 =     1, R7 =     2
R8 =        4, R9 =     2, R10 =    0, R11 =    0, R12 =    0, R13 =    0, R14 =    0, R15 =    0
R16 =       0, R17 =    0, R18 =    0, R19 =    0, R20 =    0, R21 =    0, R22 =    0, R23 =    0
R24 =       0, R25 =    0, R26 =    0, R27 =    0, R28 =    0, R29 =    128, R30 =  0, R31 =    0
```

test data 2:

```
PC =          x
Data Memory =       0,      0,      0,      0,      0,      0,      0,      0
Data Memory =       0,      0,      0,      0,      0,      0,      0,      0
Data Memory =       0,      0,      0,      0,      68,     2,      1,      68
Data Memory =       2,      1,      68,     4,      3,      16,     0,      0
Registers
R0 =        0, R1 =     0, R2 =     5, R3 =     0, R4 =     0, R5 =     0, R6 =     0, R7 =     0
R8 =        0, R9 =     1, R10 =    0, R11 =    0, R12 =    0, R13 =    0, R14 =    0, R15 =    0
R16 =       0, R17 =    0, R18 =    0, R19 =    0, R20 =    0, R21 =    0, R22 =    0, R23 =    0
R24 =       0, R25 =    0, R26 =    0, R27 =    0, R28 =    0, R29 =    128, R30 =  0, R31 =    16
```

Summary:

For me, this Lab3 is harder than the lab before as the guidance given is not as much as before. to do the basic MIPS Simple Single CPU that operate *add, addi, sub, sub, and, or, slt, slti, beq,* added some more instructions such as *sw, lw, j, jal, and jr.*

When implementing the *jr* instruction, I was frustrated because this instruction has some special properties that differ from the others. It is an R-format instruction but did not use the ALU for operation (or else the r0 would be modified). Also, the

implementation of this instruction required it to read the register file and forward it to the PC, which cannot be done by any of the current existing modules. What has most frustrated me is, I can’t find any of the formal implementation of it in slides, nor the textbook. So I have to design it by myself (or Google it, to make sure my design is not too far from the formal one). But this is a nice experience for me to design this single cycle CPU and make me dive into the details of this kind of CPU.