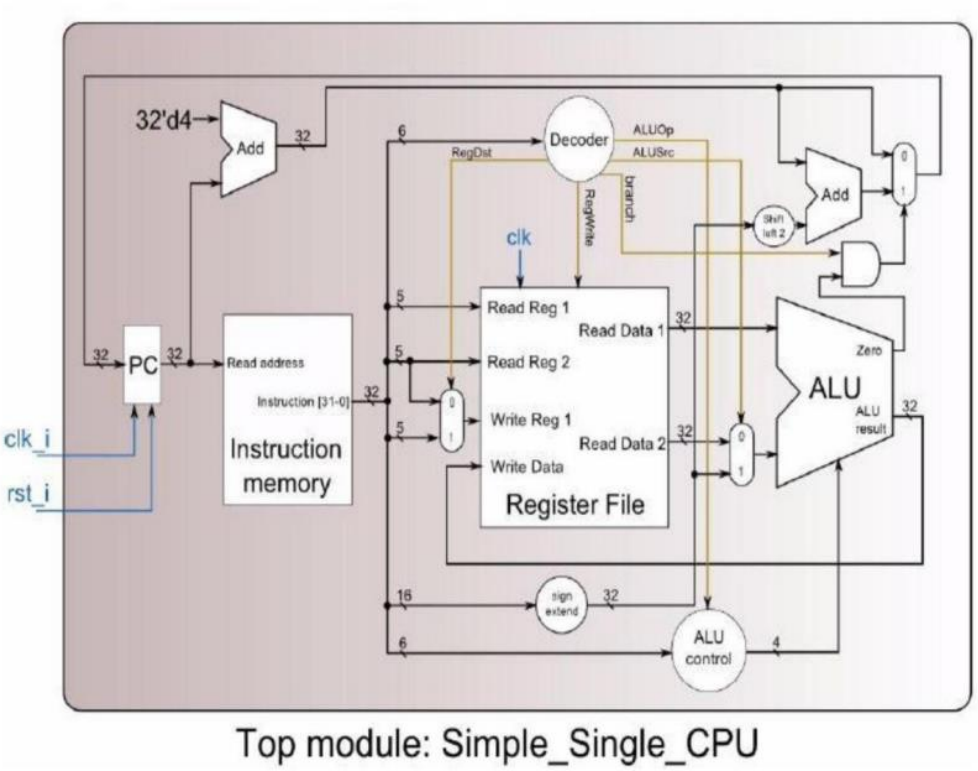
# Computer Organization Lab2

## Name:邱弘竣

## ID:109550136

### Architecture diagrams:




Top module: Simple_Single_CPU

- r-type
  - instruction[31:26] is sent to decoder
  - bits are sent into read reg 1 and read reg 2
  - Do the opeartion in ALU controlled by ALU control
  - the result will be sent to write data
  - PC = PC + 4
- addi
  - bits are sent to read reg 1 and sign extend
  - add two numbers from read data 1 and sign extend in ALU
  - the result will be sent to write data
- beq
  - bits are sent to read reg 1 and sign extend
  - subtract two numbers from read data 1 and sign extend in ALU

    - if two numbers are equal, then “zero”=1
    - PC = PC + 4 + (sign extend)*4 (zero=1 and branch =1)
- slti
    - bits are sent to read reg 1 and sign extend
    - compare two numbers from read data 1 and sign extend in ALU
    - if the condition is true, then “zero”=1
    - PC = PC + 4

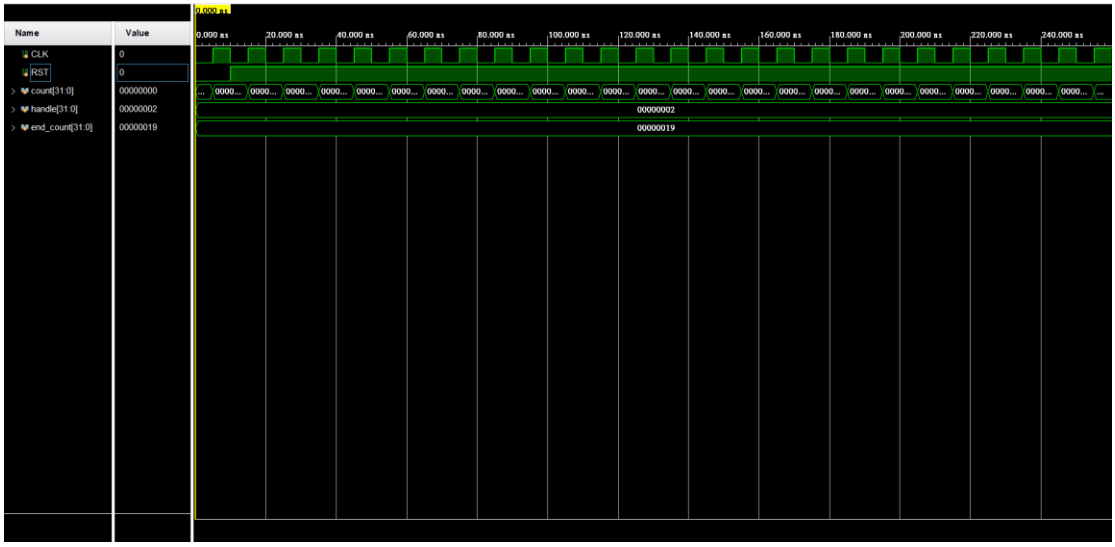
## Hardware module analysis:

**(explain how the design work and its pros and cons)**

- Adder.v
    - add two given numbers
- ALU_Ctrl.v
    - input funt_i and ALUOp_i
    - output ALUCtrl_o to control ALU to do different operations
- ALU.v
    - Given different ctrl_i, ALU can do different operations
- decoder.v
    - input instruction[31:26]
    - output RegWrite_o , ALU_op_o, ALUSrc_o, RegDst_o, Branch_o to control other modules to do different operations
- MUX_2to1.v
    - Given 0 or 1
    - do the selected things
- Shift_Left_Two_32.v
    - input a 32-bit data
    - output the data multiplied by 4
- Sign_Extend.v
    - convert 16-bit data to 32-bit
- Simple_Single_CPU.v
    - connect all the modules to implement all the MIPS code
- pros：easy to implement.
- cons：It’s very slow because it can only do one operation in one time.

## Finished part:

(show the screenshot of the simulation result and waveform, and explain it)




**Part1**

```
          2
r0=       0
r1=      10
r2=       4
r3=       0
r4=       0
r5=       6
r6=       0
r7=       0
r8=       0
r9=       0
r10=      0
r11=      0
r12=      0
```

**Part2**

```
          2
r0=       0
r1=       1
r2=       0
r3=       0
r4=       0
r5=       0
r6=       0
r7=      14
r8=       0
r9=      15
r10=      0
r11=      0
r12=      0
```

All the result are the same as CO_Lab_2.pdf

## Problems you met and solutions:

- In the begining, I didn't know what Decoder is. I tried to find out the answer in Teacher's ppt and .txt in our project. Instruction[31:26] is the intput of Decoder, deciding which type of operation is now chosen. Then I checked the code in CO_P2_test_data1.txt and compared it to CO_Lab_2.pdf. I finally understood how Decoder works.
- There are too many modules this time. In the begining I didn't how to do it because Simple_Single_CPU.v was almost empty. Then I checked the diagram in CO_Lab_2.pdf and then I understood how are these modules connect with one another.

## Summary:

Indeed, I'm not famiililar with verilog at all. However, after these labs, I think I understand verilog much more than before. Although software programming is still easier for me, I will try to improve myself in the future courses.