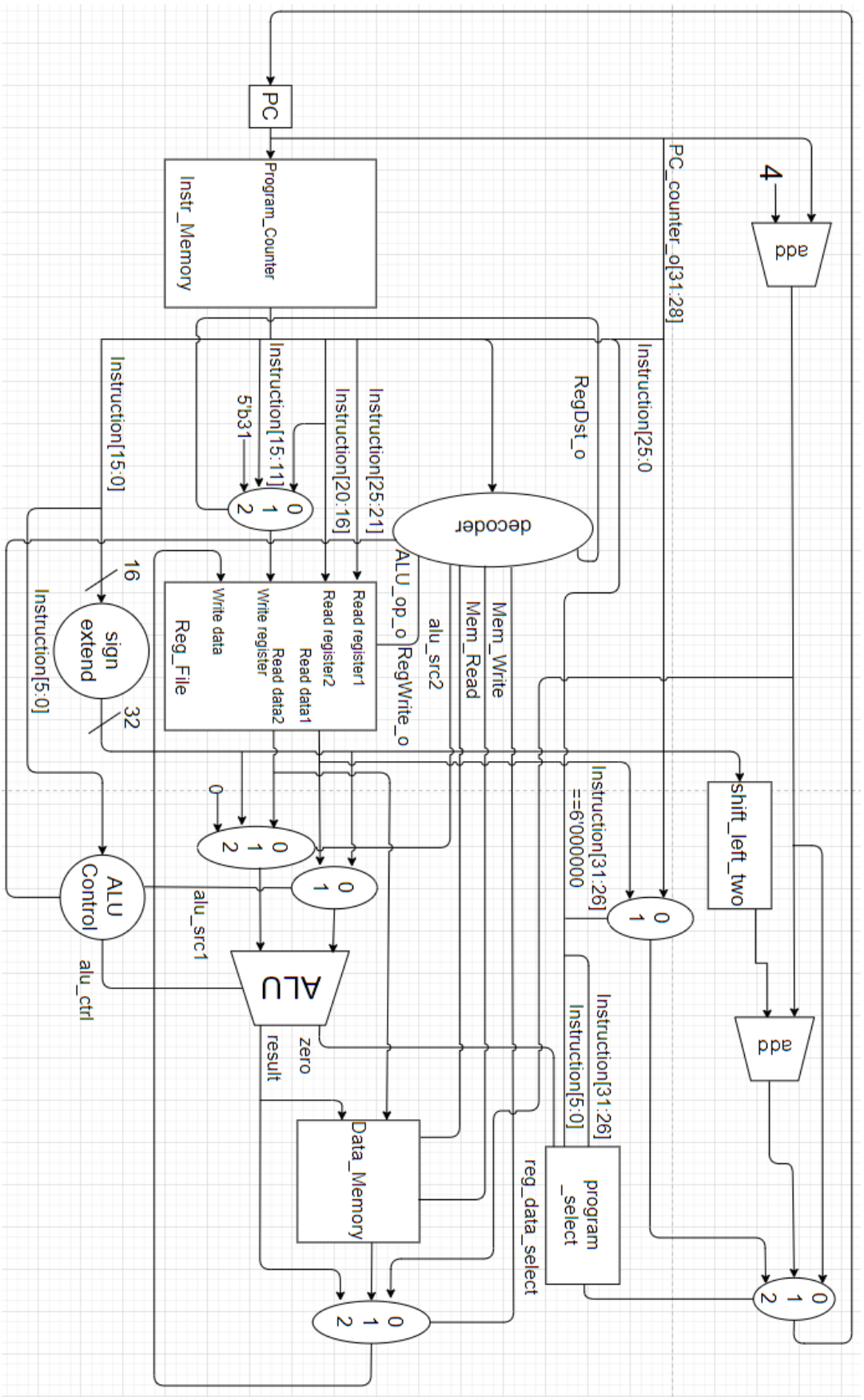
Diagram


PC
Program_Counter
Instr_Memory
4
add
PC_counter_o[31:28]
Instruction[25:0
RegDst_o
decoder
Instruction[25:21]
Instruction[20:16]
Instruction[15:11]
5'b31
Instruction[15:0]
ALU_op_o
RegWrite_o
alu_src2
Mem_Write
Mem_Read
Write data
Read register1
Read register2
Write register
Read data1
Read data2
Reg_File
16
sign extend
32
Instruction[5:0]
shift_left_two
Instruction[31:26]
==6'000000
ALU Control
alu_src1
alu_ctrl
ALU
zero
result
Data_Memory
Instruction[31:26]
Instruction[5:0]
program _select
reg_data_select

Implementation

- We add the Data_Memory module after ALU, by taking in the input from ALU result and RT_data, and the additional control Mem_Write, Mem_Read which the decoder get from looking into the opcode of every instruction to determine whether to save or load the word in or out of the memory

- After the Data_Memory we put a 3-to-1 MUX to determine whether the output of ALU result(R-Type), Data_Memory(lw), or the next line address(jal) should be save into RD_data

- We change the MUX in front of the Reg_file from 2-to-1 to 3-to-1, adding 5'd31 to the additional channel, for jal function need to store the address in r31($ra)

- We add a 2-to-1 MUX to determine if the program is taking the R-Type jump, or j, jal(those two have their address save in the same position), and we can get the overall jump address, and put the result into another 3-to-1 MUX with the normal and branch address input

- We abandon the branch_eq and branch_o in LAB02 and replace them with a additional module PC_selector to be the select of the 3-to-1 MUX that determine the next address the program should count to, by the analyzing the instruction and the output of the ALU, we determine that we should go with the normal address, the branch address, or the jump address

Question

- // j Jump //16
  00001000000000000000000000011001
  // beq $t1, $t2, Loop //108
  00010001010010011111111111111110
  // bne $t1, $0, outer//92
  00010100000010011111111111110000
  // bgtz $at, next_turn //84
  00011100001000000000000000000001

- **Jr :** let the program jump from the current address to the address saved in RS_data
  Jal : let the program jump from the current address to the address in the instruction, at the same time, save the next line address to r31($ra), so that the program can jump back to the current address
  Ex : jal (26 bit address)
  .
  .
  .
  jr $ra

- Yes
  slt $t1 $rs $rt
  beq $t1 $0 offset
- Yes
  Jr, jal, add, or, ori, slt, beq, sra, sll, mul, lw, sw
  Advantage : the CPU have less space saving instruction format, giving it additional capability in running the code, which makes the program run faster
  Disadvantage : the complexity of the code will increase enormously, as every simple instruction need to be replace by more than two other instructions

Contribution

- **J**
- **Jr**
- **Jal**

Discussions

Actually, I start working on the lab before my teammate did, but as I didn’t make the naming of the wire clear enough to understand, when my teammate started working on the lab, he couldn’t understand what I had done, what’s worst, I couldn’t fully understand my work at that time. Therefore, in the end, most my work has been deleted as be replaced by easily understanding code. Next time, I should make good naming and comments to make everything clear to avoid the same situation happen again.