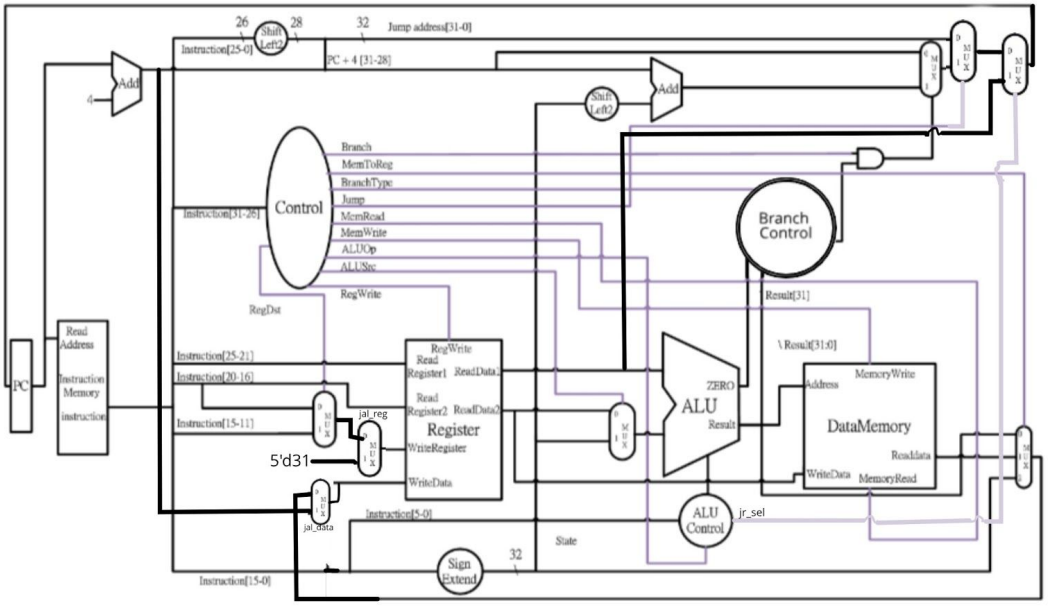
Computer Organization

I. **Architecture diagram:**



1. **Add Multiplexers:** There are more multiplexers need to add because there are more functions to be check.
   a. **Mux_Jal_Reg:** Choose register 31 for **JAL** to store PC+4 address.
   b. **Mux_Jal_Data:** Store the PC+4 in the register.
   c. **MuxToReg:** Check whether the data send to the register is from **ALU**, **DataMemory**, or **signExtend**.
   d. **Mux_Jump:** Check the next program address is Mux_branch or jump.
   e. **Mux_Jal:** Check the next program address is Mux_Jump or jal.

2. **Add Data Memory:** Store data.

II. The detailed description of the implementation:

1. Instruction

Instruction	OpCode	function	Type	ALU_op	ALU_Ctrl	ALU_func	Write Data address
lw	100 011	-	I-type	0000	0010	add	Rt
sw	101 011	-	I-type	0001	0010	add	Rt

Addu	000 000	100 001	R-type	0010	0010	add	Rd
Addi	001 000	-	I-type	0011	0010	add	Rt
Subu	000 000	100 011	R-type	0010	0011	sub	Rd
AND	000 000	100 100	R-type	0010	0000	AND	Rd
OR	000 000	100 101	R-type	0010	0001	OR	Rd
slt	000 000	101 010	R-type	0010	1010	set less than	Rd
sra	000 000	000 011	R-type	0010	1000	switch right	Rd
srav	000 000	000 111	R-type	0010	1001	switch right	Rd
beq	000 100	-	I-type	0100	0110	branch eq	-
bnez	000 101	-	I-type	0101	0111	branch n eq	-
li	001 111	-	I-type	0110	1100	load imm	Rt
ori	001 101	-	I-type	0111	0001	OR	Rt
j	000 010	-	I-type	-		jump	-
jal	000 011	-	I-type	-		jump and link	-
jr	000 000	001 000	R-type	-			-
ble	000110	-	I-type	1011		branch (<=)	
bnez	000101	-	I-type	1101		branch (!= 0)	
bltz	000 001	-	R-type	0010		branch (<0)	

※imm = immediate ; bif=branch if
※new added one is in red

2. The insight of control signal

Instr	Reg Write	RegDst	ALUOp	ALUsrc	Branch	Mem Read	Mem Write	Jump	Memto Reg
lw	1	0	0000	1	0	1	0	0	0
sw	0	x	0001	1	0	0	1	0	x

Addu	1	1	0010	0	0	0	0	0	1
Addi	1	0	0011	1	0	0	0	0	1
Subu	1	1	0010	0	0	0	0	0	1
AND	1	1	0010	0	0	0	0	0	1
OR	1	1	0010	0	0	0	0	0	1
slt	1	1	0010	0	0	0	0	0	1
sra	1	1	0010	0	0	0	0	0	1
srav	1	1	0010	0	0	0	0	0	1
beq	0	x	0100	0	1	0	0	0	x
bnq	0	x	0101	0	1	0	0	0	x
li	1	0	0110	1	0	0	0	0	1
ori	1	0	0111	1	0	0	0	0	1
j	0	x	-	x	0	0	0	1	x
jal	0	x	-	x	0	0	0	1	x
jr	0	x	-	x	0	0	0	1	x
ble	0	x	1011	0	1	0	0	0	x
bnez	0	x	1101	0	1	0	0	0	x
bltz	0	x	0010	0	1	0	0	0	x

III. **Problems encountered and solutions:**

- I thought that the ALUOp code seems more, so I try to change it from 3 bits to 4 bits.
- Here we need to add **Data Memory**. The data write back to register to have a new option, so I have to make a new 3 to 1 Multiplexer. Actually, I think that the sign extended one isn't used, but I still leave it on my diagram.
- There are more branch if instruction in this lab work. So I make a new module named **Branch Control**, to manage each kind of situation for branching.
- When implementing the **blez** function, I have met a problem. I use src1<src2 to decide whether it is less than zero. However, it makes a mistake because src1 seems to default as an unsigned value. So I change src1<src2 to src1-src2 and check the first bit. It works.
- In the origin, diagram has no jump function. These jump functions need more wire to implement. Especially **jal** and **jr**. First, I use a new Mux to detect whether it should jump after detecting whether it should branch. Second, I make a Mux for **jal** to store the **program counter** in **r31**. Last, I make a Muz after **jal Mux** to check the **jr** address.
- Instruction **jr** is an R-type instruction, so I have to use ALU controller to check if it is true.

IV. **Lesson learned (if any):**

1. More clearly about how jump and branch instruction works.
2. Have a better look about how high level language turn into machine code (such as fibonacci).
3. I sometimes make some unnecessary wire or options, making the diagram looks more complicated. I should plan the architecture more completely before I start to do it.
4. Gtkwave is really a god damn good application.