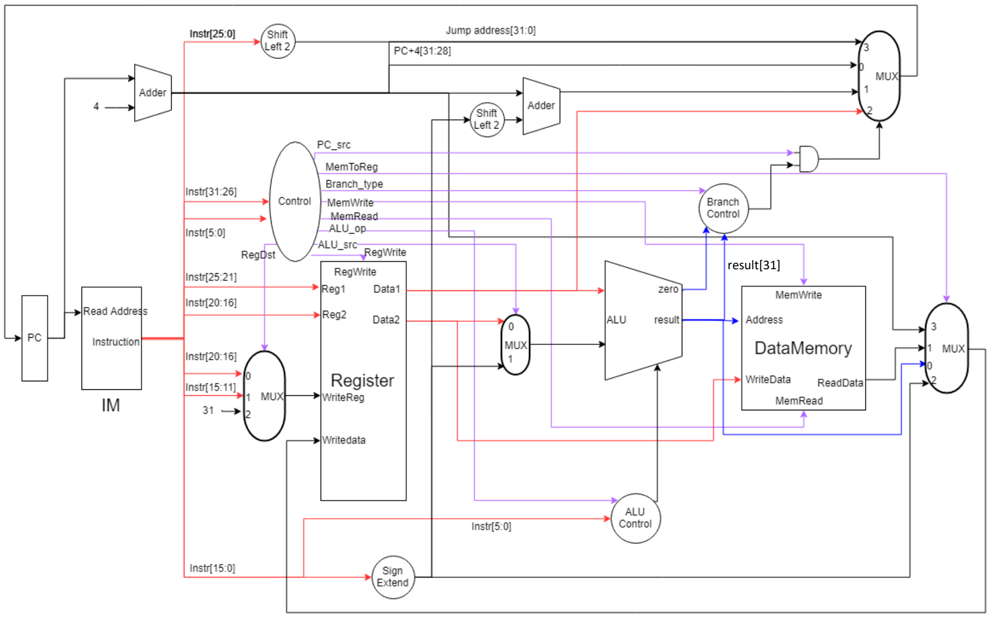
Computer Organization

Architecture diagram:



Detailed description of the implementation:

Decoder:

instr	RegDst	RegWrite	ALU_src	ALU_op(binary)	MemRead	MemWrite	Branch_type(binary)	MemToReg(binary)	PC_src(binary)
addu	1	1	0	010	0	0	x	00	00
addi	0	1	1	000	0	0	x	00	00
subu	1	1	0	010	0	0	x	00	00
and	1	1	0	010	0	0	x	00	00
or	1	1	0	010	0	0	x	00	00
slt	1	1	0	010	0	0	x	00	00
sltiu	1	1	1	011	0	0	x	00	00
beq	x	0	1	001	0	0	10	00	01
sra	1	1	1	010	0	0	x	00	00
srav	1	1	1	010	0	0	x	00	00
lui	0	1	0	101	0	0	x	00	00
ori	0	1	0	100	0	0	x	00	00
bne	x	0	1	001	0	0	11	00	01
lw	0	1	1	000	1	0	x	01	00
sw	x	0	1	000	0	1	x	00	00
j	x	0	x	111	0	0	x	00	11
mul	1	1	0	010	0	0	x	00	00
nop	x	0	x	010	0	0	x	00	00
jal	2	1	x	111	0	0	x	11	11
jr	x	0	x	010	0	0	x	00	10
ble	x	0	1	001	0	0	00	00	01
bltz	x	0	1	001	0	0	01	00	01
li	0	1	0	111	0	0	x	10	00

ALU_op

Sorted the instructions by what type of instruction and what the ALU will do:

R-type: addu, subu, and, or, slt, sra, srav, mul, nop

ALU addition: lw, sw, addi

ALU subtraction: beq, bne, ble, bltz

Nothing related to ALU: j, jal, li

The rest will be in their own group respectively

Branch Control:

Branch will be executed if the following condition is true:

1. Branch type = beq, alu_zero = 1, alu_result[31] = 0
2. Branch type = bne, alu_zero = 0
3. Branch type = ble, alu_zero = 1, alu_result[31] = 0
4. Branch type = ble, alu_zero = 0, alu_result[31] = 1
5. Branch type = bltz, alu_zero = 0, alu_result[31] = 1

PC source:

The PC source will be determined by PC_src&{1,Branch_Ctrl}.

Jump instructions won't be affected by the Branch_Ctrl

When it's a jump instruction, Branch_type will be $(10)_2$ and the ALU_result will be 0. So the Branch_Ctrl will be 1 and won't clear any bit of PC_src.

ALU control:

For R-type:

Alu control will be {funct_i[5] | funct_i[4],funct_i[2:0]}

For lw,sw,addi:

lw, sw and addi all need the ALU to do addition, so ALU control is $(1001)_2$ (same as addu)

For branches:

They need ALU to do subtraction to determine the relation, so ALU control is $(1011)_2$ (same as subu)

For jumps and li:

ALU isn't needed in these instructions, so ALU control is $(0000)_2$ (ALU will simply output 0)

For the rest:

They have their unique control bits.

lui: $(1111)_2$

ori: $(0110)_2$

sltiu:$(0101)_2$

ALU:

since we are allowed to use a 32-bit ALU, we just implement it by doing different operation depending on the input ctrl_i.

```
case(ctrl_i)
    //R type
    4'b1100/*12*/: result_o <= src1_i & src2_i;
    4'b1101/*13*/: result_o <= src1_i | src2_i;
    4'b1001/* 9*/: result_o <= src1_i + src2_i;
    4'b1011/*11*/: result_o <= src1_i - src2_i;
    4'b1010/*10*/: result_o <= src1_i < src2_i;
    4'b0011/* 3*/: result_o <= $signed(src2_i) >>> shamt_i; //sra
    4'b0111/* 7*/: result_o <= $signed(src2_i) >>> src1_i; //srav
    //I type
    4'b0101/* 5*/: result_o <= src1_i < (src2_i & 32'h0000FFFF); //sltiu
    4'b1111/*15*/: result_o <= src2_i << 16; //lui
    4'b0110/* 6*/: result_o <= src1_i | (src2_i & 32'h0000FFFF); //ori
    4'b1000/* 8*/: result_o <= src1_i * src2_i;
    4'b0000/* 0*/: result_o <= 32'd0;
```

Problems encountered and solutions:

For 23 instructions, how to use three bit to let ALU_Ctrl to know what the alu should do?

It'll be solved by grouping the instructions that ask the ALU to do the same thing.

How should I know when doing a jump, where can I get the address?

I just pull the last 6 bit of the instruction into the decoder so that it can determine whether it's a 'jr' or a 'j'.

Lesson learnt (if any):

Using Verilog, I can simply use if-else statement to implement the classification instead of using k map to get the classifier of each control signal. This is not only reader-friendly and also easy to implement.