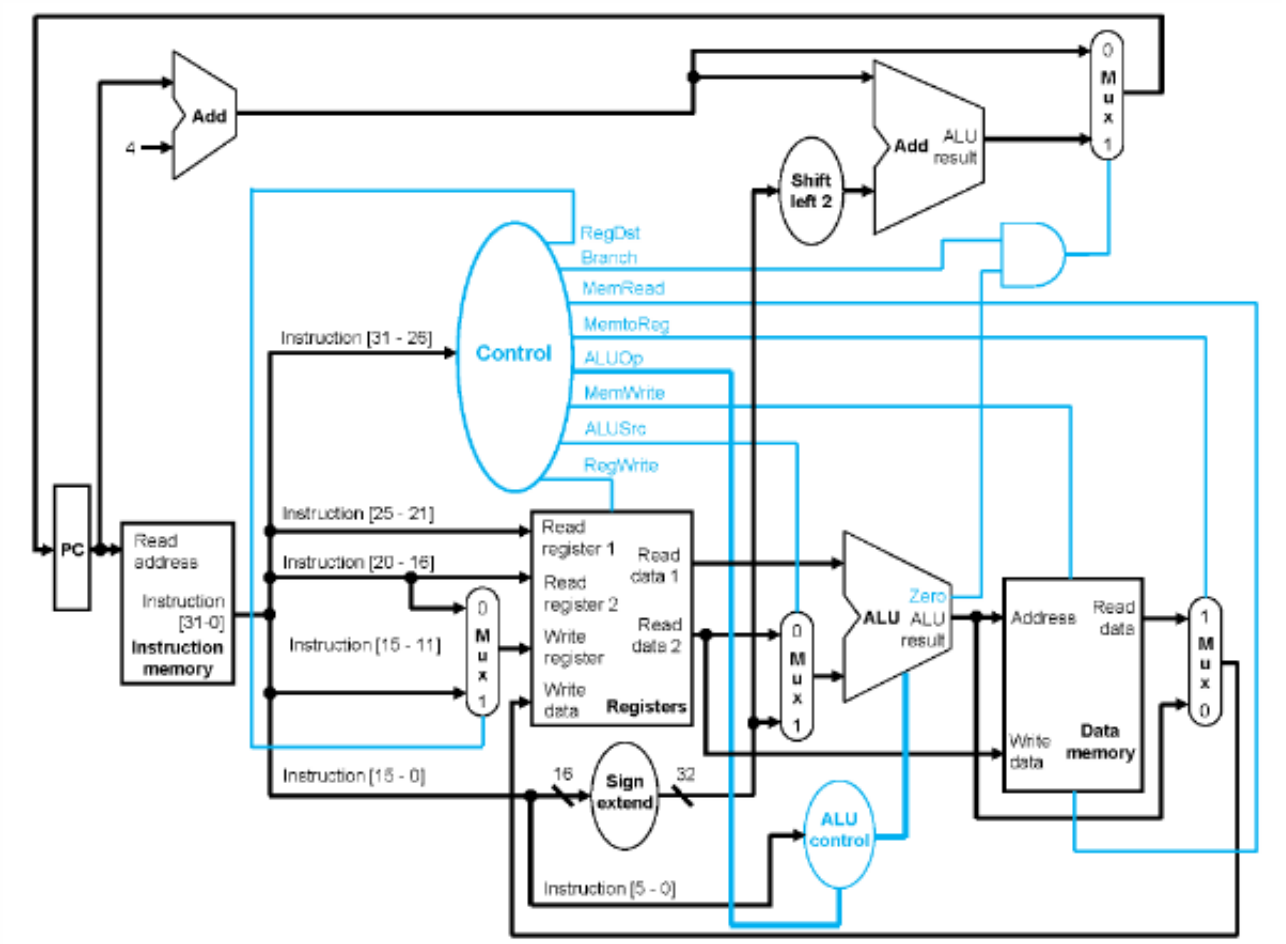
COMPUTER ARCHITECTURE

Single Cycle Implementation of MIPS Architecture

Name: Mihir Vora
ID: 2018A3PS0755G

Circuit Diagram:



Different Modules and their function:

1) **PC (Program Counter)** – Gives the address in the memory where the instruction to be executed is present. Also calculates the address for the next instruction to be executed.
2) **InstMem (Instruction Memory)** – In this module the instruction memory has been defined by storing some instructions in a few memory locations. It chooses the instruction to be executed depending on the address received from the Program counter.
3) **MainControl** – It takes opcode part of the instruction as input and generates all the control signals like ALUSrc, MemWrite, RegWrite, MemRead, Branch, RegDst, MemtoReg, ALUOp.
4) **Mux1** – It makes the decision as to which bits of the instruction hold the information about the Write register by using RegDst control signal.
5) **RegFile** – Depending on the instruction it chooses appropriate register numbers and assigns the content of those registers to ReadData1 and ReadData2. Also chooses the appropriate register to write the result and does the work of storing the result into that register. Some registers have been assigned values in this module so as to use them while simulating some instructions.

6) **SignExtend** – Uses bits 0:15 from the instruction and extends that to a 32 bit number which is used as an offset in lw, sw, beq instructions.
7) **Mux2** – Depending on the control signal input ALUSrc decides whether the second operand should come from ReadData2 or from the 32 bit sign extended value.
8) **ShiftLeft** – Used to calculate the new address in case of Branch instructions.
9) **ALUControl** – It takes the value of ALUOp control signals as input and using this along with the last 6 function bits of the instruction decided on the function the ALU will perform.
10) **Fpa** – This module is used to implement floating point addition when required and is considered to be a part of the ALU.
11) **ALU** – It takes two operands and gives a particular output ALUOp. The function performed on the operands depends on the signals generated by ALUControl. It also gives another output(zero) which is used to implement the beq instruction.
12) **Add_ALU** – Used to calculate the new address to branch to in case of Branch instructions.
13) **AndGate** – Gives the select input to Mux3 to decide which address to send to PC.
14) **Mux4** – Takes Pc_out, AddALU_out and decides which address to send to PC depending on AndGate result.
15) **DataMemory** - Used to write to or read from a particular memory location during lw and sw instructions. The memory address to access is given by ALU_out and whether to read or write data is decided by MemRead and MemWrite control signals. Values have been assigned to a few memory locations in this module.
16) **Mux3** – This mux is used to decide whether the output of the ALU is to be used to write to destination register or value from the memory location as in the case of lw instruction. This is decided by using MemtoReg control signal as select input.
17) **MIPS CPU –** This is the top module of the implementation which is used to connect all the above mentioned modules.

Instruction Memory:

```
Mem[0] = 32'h00221820; //add: R3, R1, R2

Mem[1] = 32'hAC010000; //sw: R1, 0(R0)

Mem[2] = 32'h8C240000; //lw R4, 0(R1)

Mem[3] = 32'h10210001; //beq R1, R1, 1

Mem[4] = 32'h00001820; //add R3, R0, R0

Mem[5] = 32'h00e84828; //floating point addition R9,R7,R8
```

For floating point addition I have used the opcode as 0 (R-type) and function code as 40. The instruction for it has been written accordingly.

Register values:

```
reg_mem[0] <= 4;

reg_mem[1] <= 7;

reg_mem[2] <= 21;

reg_mem[7] <= 32'h7F7FFFFF;
```

reg_mem[8] <= 32'h7F400001;

Data Memory:

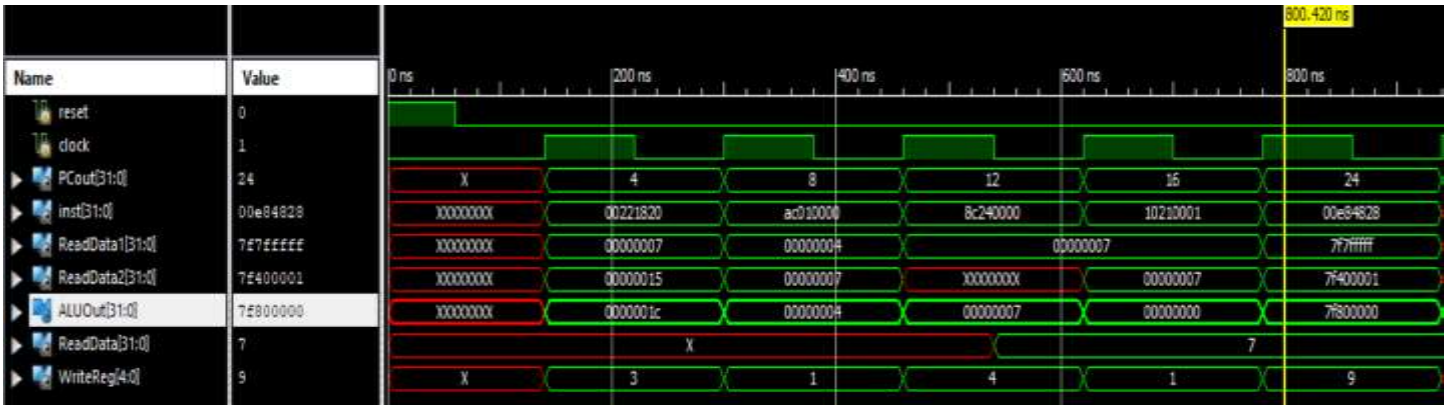
Mem[0] = 2;

Mem[1] = 3;

Mem[2] = 4;

Test Bench: Here the only inputs given are reset and the clock pulses.

Simulation Results:



Results Explained:

1st positive clock edge - Instruction is add: R3, R1, R2. It takes values from R1 and R2 which is 7 and 21 and the result of addition i.e. 28 is stored in R3.

2nd positive clock edge - Instruction is sw: R1, 0(R0). The word in R1 is 7 which is read and stored in the address given in R0 i.e. 4. This is the byte address. So now Mem[1] has 7.

3rd positive clock edge - Instruction is lw R4, 0(R1). Here ReadData2 shows xxxx because the R18 (Bits 16:20 of the instruction) hasn't been initialized. But ReadData2 won't be used as it is lw. The data in memory location of address stored in R1 will be written to R4. R1 has 7. But memory can be accessed only in multiples of 4. So Mem[1] (Bytes 4-7) is accessed which has 7. Hence ReadData is 7. This is written to R4.

4th positive clock edge – Instruction is beq R1, R1, 1. As R1 is going to be the same as R1, the branch will be taken. The new PC value becomes PC + 4 + 1*4 = PC + 8 = 24. Therefore the next instruction stored at Mem[4] will be skipped.

5th positive clock edge – Instruction is floating point addition R9,R7,R8. The two operands are taken from R7 and R8, their floating point addition is done and the final value is stored in R9.

Hence this MIPS implementation can execute R-type instructions (and, or, add, subtract, slt, nor, floating point addition), lw, sw and beq.