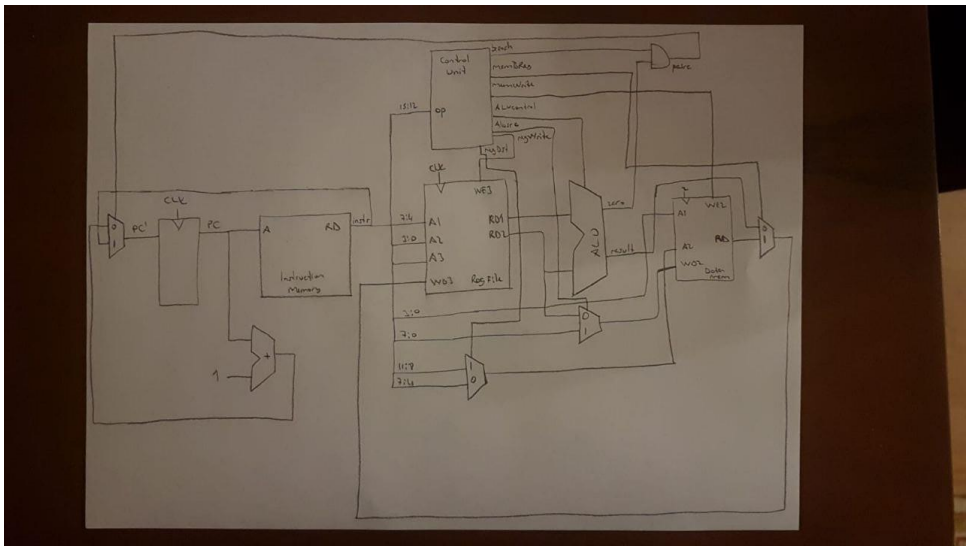
Ahmet Faruk Ulutaş - CS223-1 – 21803717

Project Report

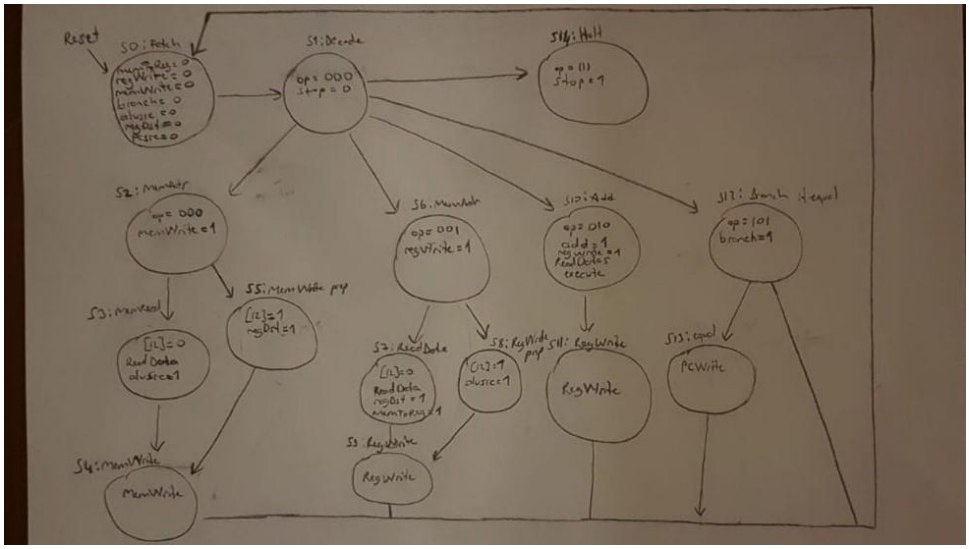
25.12.2020

1)



In the datapath part, I watched the address of the instruction memory, the instruction coming from pc, and then the instruction from the read data section and distributing it in the data path. While distributing instruction in Datapath, I created the paths respectively according to the operators we have. While doing this, I used the register file for the transformation of the data at hand, the address in the data memory, the write and read sections for the recorded data. If store operation is used, I take the data from the register file or instruction and save it to the data memory. If the load operation was used, I took the data from the instruction or data memory and loaded it into the register file. If add was used, if add, beq operation was used, I compared the inputs and jumped accordingly with the requested instruction pc. If more than one data is going to the same place, I made it possible to operate according to the operator with the controller using mux. With ALU, I have achieved the result by collecting or subtracting data from operators such as add and beq as input. In the controller section, I determined which operator to run by inputting the first three bits from the instruction to op. Then, according to the flow of the operator in the datapath, I made it possible to adjust the path by outputting outputs from the control unit to the relevant places. Also, I have updated the pc by incrementing one bit or according to the jump state so that the instructions that will come in the pc section can be read.

2)



In the state diagram section, I first ensured that the initial values are fetched with the reset button. Later, I made it possible to go to the initial operation by decoding and then I drew the diagrams of the store, load, add, beq, halt operations respectively. In these diagrams, if the immediate bit in store and load is 0, I performed the writing after reading the data. If the Immediate bit is 1, I performed direct writing. I made the sum in the section where the Add operation is performed. I set the branch to one in the section where the beq operation takes place, and if they are equal, I executed the pcwrite operation and then skipped the requested instruction. If not, transactions continued as they were. I stopped all operations by bringing the stop status to 1 in the halt operation and ended the diagram. Finally, I made sure to return to the first state and start over at the end of all operations except Halt.

3)

```
module controller(input logic [3:0] op,
                  input logic reset,
                  input logic zero,
                  output logic memToReg, memWrite,
                  output logic pcsrc, alusrc,
                  output logic regDst, regWrite,
                  output logic [2:0] alucontrol, stop);


   logic [2:0] aluop;
   logic branch;
   logic stop;


  // initial begin
```

```
//   stop = 0;
// end

 always_comb
 begin
    if(reset) begin
       regWrite <= 0;
       memWrite <= 0;
    end

    case(op[3:1])
       3'b000: begin
          if(op[0] == 0)
          begin
             branch = 0;
             memToReg = 0;
             memWrite = 1;
             alusrc = 0;
             regDst = 1;
             regWrite = 0;
             alucontrol = 3'b000;
          end
          else if(op[0] == 1)
          begin
             branch = 0;
             memToReg = 0;
             memWrite = 1;
             alusrc = 1;
             regDst = 0;
             regWrite = 0;
             alucontrol = 3'b000;
```

```
    end
end
3'b001: begin
    if(op[0] == 0)
    begin
        branch = 0;
        memToReg = 1;
        memWrite = 0;
        alusrc = 0;
        regDst = 1;
        regWrite = 1;
        alucontrol = 3'b001;
    end
    else if(op[0] == 1)
    begin
        branch = 0;
        memToReg = 0;
        memWrite = 0;
        alusrc = 1;
        regDst = 0;
        regWrite = 1;
        alucontrol = 3'b001;
    end
end
3'b010: begin
    branch = 0;
    memToReg = 0;
    memWrite = 0;
    alusrc = 0;
    regDst = 0;
    regWrite = 1;
```

```
            alucontrol = 3'b010;
        end
        3'b101: begin
            branch = 1;
            memToReg = 0;
            memWrite = 0;
            alusrc = 0;
            regDst = 0;
            regWrite = 0;
            alucontrol = 3'b101;
        end
        3'b111: begin
            stop = 1;
        end
        default: begin
            memToReg = 0;
            memWrite = 0;
            alusrc = 0;
            regDst = 0;
            regWrite = 0;
            alucontrol = 3'b100;
        end
      endcase
  end

  assign pcsrc = branch & zero;

endmodule
```