Term Project Submission 1 - Student Names

- C Akshay Kumar (192SP006)
- K Kiran Kumar (192SP012)

Progress

- We have created a Multi Cycle MIPS Processor using procedural assigments in a single block.
- As of now, the ISA is just 4 Instructions ADD, SUB, AND and OR.

Specifications

- Five Stage pipelining with **IF, ID, EXE, MEM and WB**
- **32**-bit Program Counter.
- **32** Registers of each **32** bit size.
- **4** Pipeline Registers of IFID-64, IDEX-127, EXMEM-69, MEMWB-37 bits respectively to store the **data between blocks**.
- **3** Pipeline Registers of EX-11, MEM-4 and WB-1 bits respectively to store **control signals**.
- **32 bit** wide instruction and data memory of size 1024.
- Posedge Triggered clock of Time period = **20ns**.
- Main Control which generates control signals is kept in **ID stage** itself.

Problems Faced

- Since we are using procedural assignments there is a problem while implementation.
- We are giving the instructions into the instruction memory by manually converting them into bits and assigning them before the first pipeline block starts.

Future work

- We will try to modify the code so that it can be Implemented.
- Find a dynamic way of storing instructions into memory given the code.
- We will design a ISA with all kinds of instructions.
- To improve the throughput we will deploy **Data Forwarding**, **Branch Target Buffer** etc.
- If time permits, we will also try to design basic compiler that can **Unroll**, **Rename** and try to solve some dependencies in the code given.

Program Code

Question 1: .

```
1
2  `timescale 1ns / 1ps
3  //////////////////////////////////////////////////////////////////////////////////
4  // Company:
5  // Engineer:
6  //
7  // Create Date: 02/22/2020 12:04:58 PM
8  // Design Name: MIPS Processor Design
9  // Module Name: processor
10 // Project Name: Term Project - Prof M S Bhatt
11 // Target Devices:
12 // Tool Versions:
13 // Description:
14 //
15 // Dependencies:
16 //
17 // Revision:
18 // Revision 0.01 - File Created
19 // Additional Comments:
20 //
21 //////////////////////////////////////////////////////////////////////////////////
22
23
24 module processor(clock);
25     // CLOCK
26     input clock;
27     // Program Counter
28     reg [31:0] PC;
29
30     // PIPELINE REGISTERS
31     reg [63:0] IFID;
32     reg [126:0] IDEX;
33     reg [68:0] EXMEM;
34     reg [36:0] MEMWB;
35
36     // Control Signals Pipeline Registers
37     reg [10:0]EX; // All Control Signals except RegDest
38     reg [3:0]MEM; // Control signals like MemRead, MemWrite, MemtoReg, RegWrite
39     reg WB; // RegWrite
40
41     // Intruction and Data Memory
42     reg [31:0]InsMem[0:1023], DatMem[0:1023];
43     // Registers 32 registers of 32 bits each
44     reg [31:0]Reg[0:31];
45     // Other Connections
46     reg [4:0] Rs, Rt, Rd;
47     reg [5:0] opcode, func;
48     reg RegDst,MemtoReg;
49     reg [31:0] DataOut;
50     integer i;
51
52     initial
53         begin
54         InsMem[0] = 32'b00000000001000100001100000000000; // ADD R1 R2 R3
55 ________InsMem[1] = 32'b00000000100001010011000000000010; // SUB R4 R5 R6
56         InsMem[2] = 32'b00000000111010000100100000000100; // AND R7 R8 R9
57
```

```
        //␣Lets␣store␣R1=␣1,␣R2␣=␣2,␣R3␣=␣0,␣R4␣=␣5,R5␣=␣4,␣R6␣=␣1,␣R7␣=␣7,␣R8␣=␣8,␣R9␣=␣0
        //␣After␣Execution␣R3␣=␣3,␣R6␣=␣1,␣R9␣=␣0
        PC␣=␣32'h00000000; // Address of First Instructiom
        // Initialize all the PIPELINE REGISTERS and CONTROL REGISTERS TO ZERO
        IFID = 64'd0;
        IDEX␣=␣127'd0;
        EXMEM = 69'd0;
        MEMWB␣=␣37'd0;

        EX = 11'd0;
        MEM␣=␣4'd0;
        WB = 1'b0;
        RegDst␣=␣1'b1;

        for(i=0;i<=31;i=i+1)
            Reg[i] = i; // Initializing the Register Bank

        Reg[5] = 32'd4;
        Reg[4]␣=␣32'd1;

        end
    always @(posedge clock)
        begin
            // Instruction Fetch
            IFID[63:32] <= InsMem[PC];
            IFID[31:0] <= PC;
            PC <= PC+1;

            // Instruction Decode

            // Control Signal Generation
            opcode <= IFID[63:58]; // Entire Opcode
            func <= IFID[37:32]; // Last Four Bits of
            // R-Type Instructions
            if(opcode==6'b000000)
                begin
                RegDst␣<=␣1;␣//␣Destination␣Register␣for␣R␣type␣is␣always␣Rd
                    if(func␣==␣6'b000000)
                        begin
                            RegDst = 1'b1;
                            EX[0]␣<=␣0;␣//␣Extention␣Operation
                            EX[1]␣<=␣0;␣//␣ALU␣Source
                            EX[3:2]␣<=␣2'b00; // ALU Control ~~~~~~~~~~ ADD R-Type Instru
                            EX[6:4] <= 3'b000;␣//␣J␣Beq␣Bne
                            EX[9:7]␣<=␣3'b000; // MemRead, Memwrite, MemtoReg
                            EX[10] <= 1; // RegWrite
                        end
                    if(func == 6'b000010)
                        begin
                            RegDst␣=␣1'b1;
                                EX[0] <= 0; // Extention Operation
                                EX[1] <= 0; // ALU Source
                                EX[3:2] <= 2'b01;␣//␣ALU␣Control␣~~~~~~~~~~␣SUB␣R-Type␣In
                            EX[6:4]␣<=␣3'b000; // J Beq Bne
                                EX[9:7] <= 3'b000;␣//␣MemRead,␣Memwrite,␣MemtoReg
                            EX[10]␣<=␣1;␣//␣RegWrite
                        end
                    if(func␣==␣6'b000100)
                            begin
                                RegDst = 1'b1;
                            EX[0]␣<=␣0;␣//␣Extention␣Operation
```

```
                                    EX[1] <= 0; // ALU Source
                                    EX[3:2] <= 2'b10; // ALU Control ~~~~~~~~~~ AND R-Type In
                                    EX[6:4] <= 3'b000; // J Beq Bne
                                    EX[9:7] <= 3'b000; // MemRead, Memwrite, MemtoReg
                                    EX[10] <= 1; // RegWrite
                                end

                        if(func == 6'b000110)
                            begin
                                    RegDst = 1'b1;
                                    EX[0] <= 0; // Extention Operation
                                    EX[1] <= 0; // ALU Source
                                    EX[3:2] <= 2'b11; // ALU Control ~~~~~~~~~~ OR R-Type Ins
                                    EX[6:4] <= 3'b000; // J Beq Bne
                                    EX[9:7] <= 3'b000; // MemRead, Memwrite, MemtoReg
                                    EX[10] <= 1; // RegWrite
                            end
                    end
                else
                    EX <= 11'b10000000000;


                Rs <= IFID[57:53]; // Source Register
                Rt <= IFID[52:48]; // Second Source Register
                Rd <= RegDst?IFID[47:43]:MEMWB[36:32]; // Destination Register
                IDEX[31:0] <= PC;
                IDEX[57:32] <= IFID[47:32];
                IDEX[89:58] <= Reg[Rs];
                IDEX[121:90] <= Reg[Rt];
                IDEX[126:122] <= Rd; // pipeline registers



                // Execute or Address Calculation

                // Control Signal Copy
                MEM[2:0] <= EX[9:7];
                MEM[3] <= EX[10];

                case(EX[3:2])
                    2'b00: EXMEM[31:0] <= IDEX[121:90] + IDEX[89:58]; // ADD
                    2'b01: EXMEM[31:0] <= IDEX[121:90] - IDEX[89:58]; // SUB
                    2'b10: EXMEM[31:0] <= IDEX[121:90] & IDEX[89:58]; // AND
                    2'b11: EXMEM[31:0] <= IDEX[121:90] | IDEX[89:58]; // OR

                endcase

                EXMEM[68:64] <= IDEX[126:122];
                EXMEM[63:32] <= IDEX[121:90]; // Pipeline registers

                // Memory Stage
                WB <= MEM[3];

                // NOT COMPLETE
                DataOut <= DatMem[EXMEM[31:0]];
                MEMWB[31:0] <= EX[9]?DataOut:EXMEM[31:0];
                MEMWB[36:32] <= EXMEM[68:64]; // Pipeline registers


                // Write back Stage
                if(WB)
```

```
180                     begin
181                         Reg[MEMWB[36:32]] <= MEMWB[31:0];
182                     end
183                 $display(Reg[6]);
184 
185 
186             end
187 
188 endmodule
```