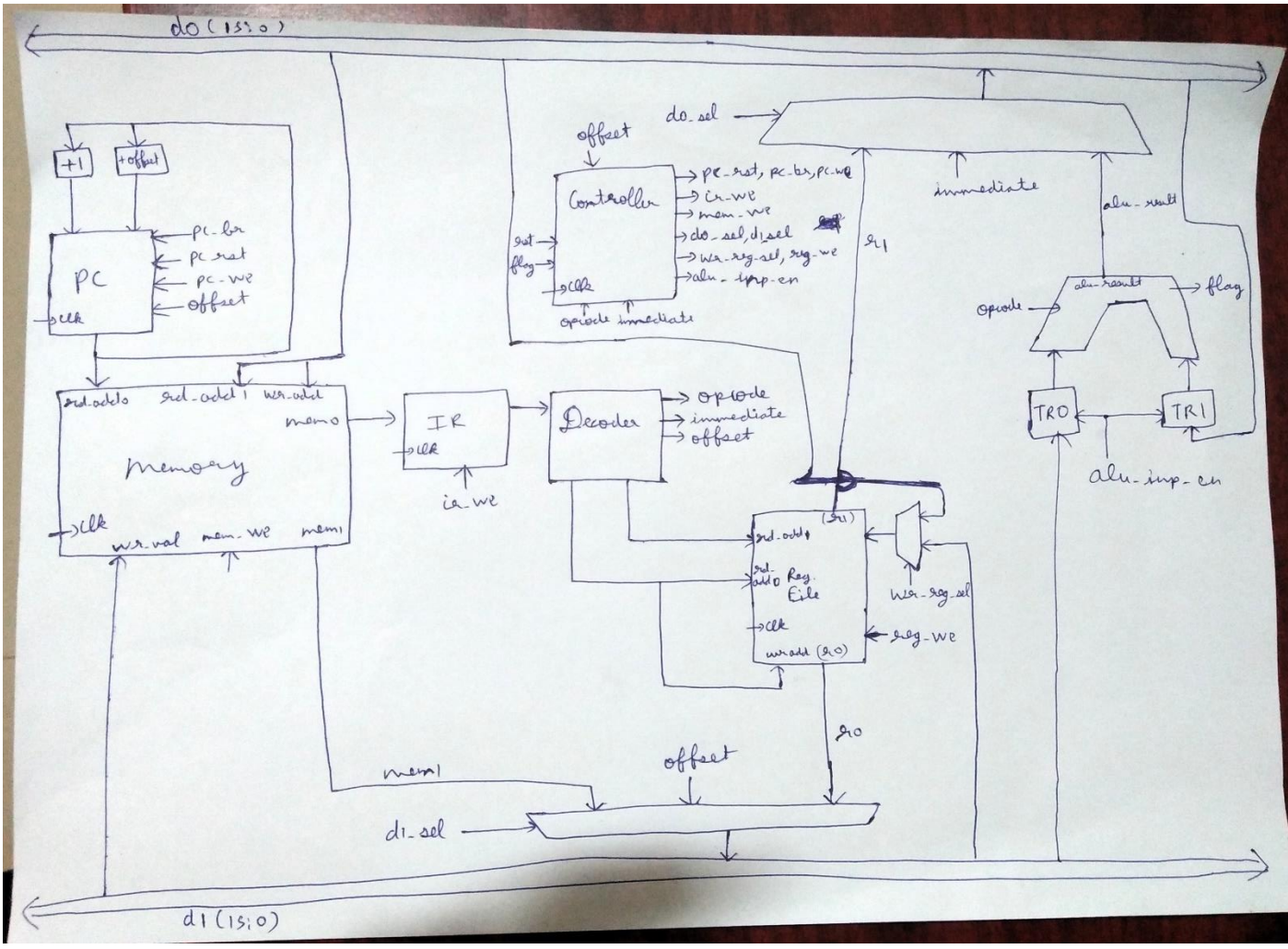
**Name: Gaurav Somani**

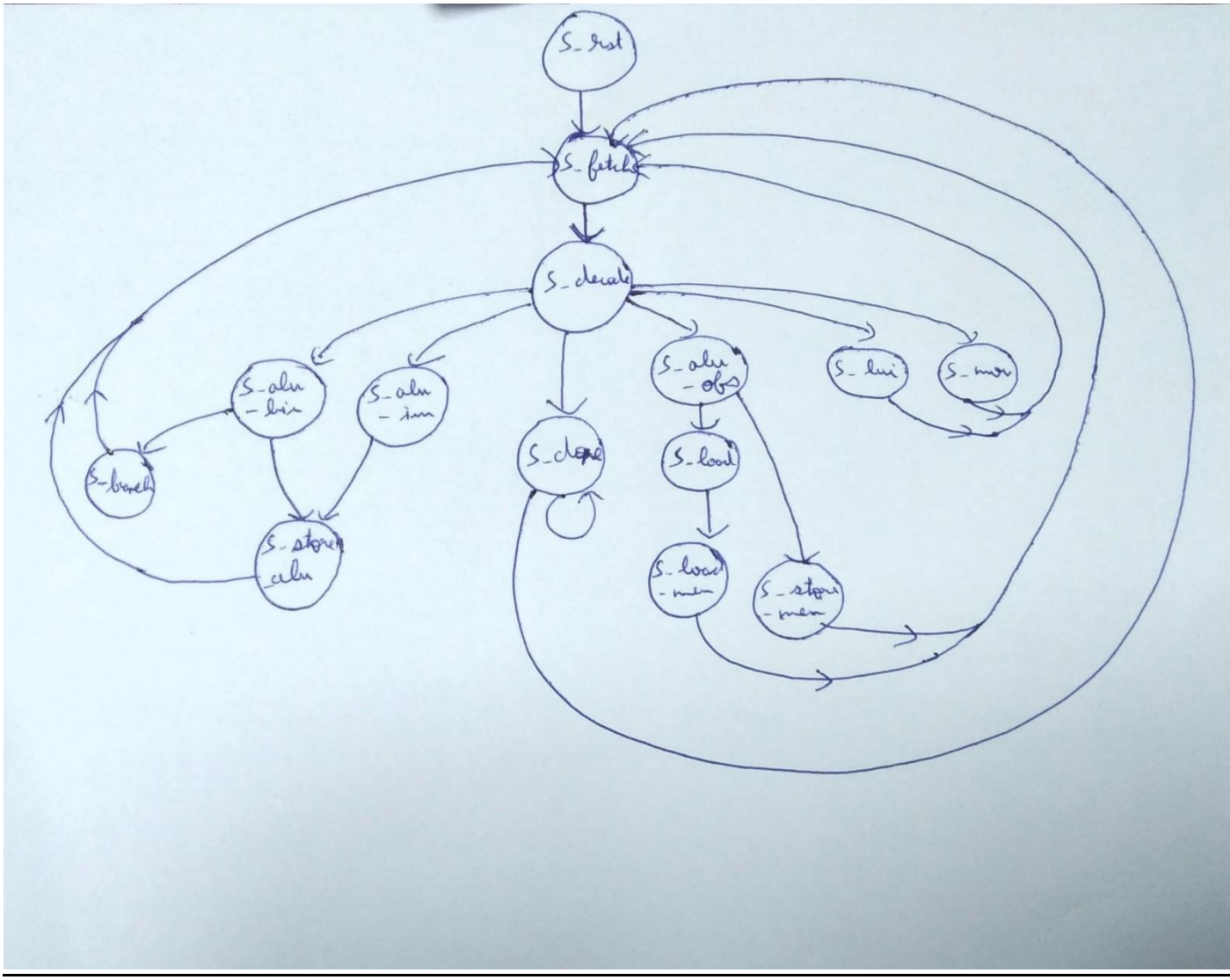
**Sr. No. 16082**

**Course : M.Tech (CeNSE)**

## Block schematic of the datapath



## FSM of Controller




FSM of the controller is expressed in the VHDL process below with the states shown in above diagram

```
fsm: process (ar,br,im,op,flag,pr_state,clk)
   begin
      case pr_state is
         when s_rst =>
                  pc_br <= '-'; pc_rst <='1'; pc_we <= '0';
            mem_we <= '0';ir_we <= '0';reg_we <= '0';
            d0_sel <= "--"; d1_sel <= "--"; done <= '0';alu_inp_en <= '0';wr_reg_sel <= '-';
            nx_state <=s_fetch;

         when s_fetch =>
            pc_rst <='0'; pc_we <= '0';
            mem_we <= '0';ir_we <= '1';reg_we <= '0';
            d0_sel <= "--"; d1_sel <= "--"; done <= '0';alu_inp_en <= '0'; wr_reg_sel <= '-';
            nx_state <=s_decode;

         when s_decode =>
                  pc_br <= '-'; pc_rst <='0'; pc_we <= '0';
            mem_we <= '0';ir_we <= '0';reg_we <= '0';
            d0_sel <= "--"; d1_sel <= "--"; alu_inp_en <= '0'; wr_reg_sel <= '0';done <= '0';
            if(ar = '1' or br ='1') then
                  nx_state <= s_alu_bin ;
            elsif( im = '1' ) then
                  nx_state <= s_alu_im;
            elsif( op = load or op = store ) then
                  nx_state <= s_alu_ofs;
            elsif( op = lui ) then
                  nx_state <= s_lui;
            elsif( op = mov ) then
                  nx_state <= s_mov;
            else
                  nx_state <= s_done;
            end if;
```

```
when s_mov =>
   pc_br <= '0'; pc_rst <='0'; pc_we <= '1';
   mem_we <= '0';ir_we <= '0';reg_we <= '1';
   d0_sel <= "01"; d1_sel <= "--"; done <= '0';alu_inp_en <= '0';wr_reg_sel <= '0';
   nx_state <= s_fetch;

when s_alu_bin =>
   pc_rst <='0';pc_br <= '0'; pc_we <='0';
   mem_we <= '0';ir_we <= '0';reg_we <= '0';
   d0_sel <= "01"; d1_sel <= "00"; done <= '0';alu_inp_en <= '1';wr_reg_sel <= '0';
   if(br ='1') then
      nx_state <= s_branch;
   else
       nx_state <= s_store_alu;
   end if;

when s_branch =>
   pc_rst <='0'; pc_we <= '1';
   mem_we <= '0';ir_we <= '0';reg_we <= '0';
   d0_sel <= "--"; d1_sel <= "--"; done <= '0';alu_inp_en <= '0';wr_reg_sel <= '0';
   if(flag = '1') then
         pc_br <= '1';
   else
         pc_br <= '0';
   end if;
   nx_state <= s_fetch;

 when s_lui =>
   pc_br <= '0'; pc_rst <='0'; pc_we <= '1';
   mem_we <= '0';ir_we <= '0';reg_we <= '1';
   d0_sel <= "10"; d1_sel <= "--"; done <= '0';alu_inp_en <= '0';wr_reg_sel <= '0';
   nx_state <= s_fetch;
```

```
when s_alu_im =>
   pc_br <= '-'; pc_rst <='0'; pc_we <= '0';
   mem_we <= '0';ir_we <= '0';reg_we <= '0';
   d0_sel <= "10"; d1_sel <= "00"; done <= '0';alu_inp_en <= '1';wr_reg_sel <= '-';
   nx_state <= s_store_alu;

when s_alu_ofs =>
   pc_br <= '-'; pc_rst <='0'; pc_we <= '0';
   mem_we <= '0';ir_we <= '0';reg_we <= '0';
   d0_sel <= "01"; d1_sel <= "01"; done <= '0';alu_inp_en <= '1';wr_reg_sel <= '-';
   if(op = load) then
      nx_state <= s_load;
   else
      nx_state <= s_store_mem;
   end if;

when s_store_alu =>
            pc_br <= '0'; pc_rst <='0'; pc_we <= '1';
   mem_we <= '0';ir_we <= '0';reg_we <= '1';
   d0_sel <= "00"; d1_sel <= "--"; done <= '0';alu_inp_en <= '0';wr_reg_sel <= '0';
   nx_state <= s_fetch;

when s_load =>
            pc_br <= '-'; pc_rst <='0'; pc_we <= '0';
   mem_we <= '0';ir_we <= '0';reg_we <= '0';
   d0_sel <= "00"; d1_sel <= "--"; done <= '0';alu_inp_en <= '0';wr_reg_sel <= '-';
   nx_state <= s_load_mem;

when s_load_mem =>
            pc_br <= '0'; pc_rst <='0'; pc_we <= '1';
   mem_we <= '0';ir_we <= '0';reg_we <= '1';
   d0_sel <= "--"; d1_sel <= "10"; done <= '0';alu_inp_en <= '0';wr_reg_sel <= '1';
   nx_state <= s_fetch;
```

```
        when s_store_mem =>
                pc_br <= '0'; pc_rst <='0'; pc_we <= '1';
            mem_we <= '1';ir_we <= '0';reg_we <= '0';
            d0_sel <= "00"; d1_sel <= "00"; done <= '0';alu_inp_en <= '0';wr_reg_sel <= '-';
            nx_state <= s_fetch;

        when s_done =>
                pc_br <= '-'; pc_rst <='0'; pc_we <= '0';
            mem_we <= '0';ir_we <= '0';reg_we <= '0';
            d0_sel <= "--"; d1_sel <= "--"; done <= '1';alu_inp_en <= '0';wr_reg_sel <= '-';
            nx_state <= s_done;

        when others =>
            pc_br <= '-'; pc_rst <='-'; pc_we <= '-';
            mem_we <= '0';ir_we <= '0';reg_we <= '0';
            d0_sel <= "--"; d1_sel <= "--"; done <= '0';alu_inp_en <= '0';wr_reg_sel <= '-';
            nx_state <= s_rst;
    end case;
end process fsm;
```

# Timing Report

Clock Frequency : 100 MHz

Duty Cycle = 50 %

| Setup | | Hold | | Pulse Width | |
|---|---|---|---|---|---|
| Worst Negative Slack (WNS): | 3.518 ns | Worst Hold Slack (WHS): | 0.107 ns | Worst Pulse Width Slack (WPWS): | 4.500 ns |
| Total Negative Slack (TNS): | 0.000 ns | Total Hold Slack (THS): | 0.000 ns | Total Pulse Width Negative Slack (TPWS): | 0.000 ns |
| Number of Failing Endpoints: | 0 | Number of Failing Endpoints: | 0 | Number of Failing Endpoints: | 0 |
| Total Number of Endpoints: | 1427 | Total Number of Endpoints: | 1427 | Total Number of Endpoints: | 746 |

**All user specified timing constraints are met.**

# Utilisation

| Name | Slice LUTs (20800) | Slice Registers (41600) | F7 Muxes (16300) | Bonded IOB (106) | BUFGCTRL (32) |
|---|---|---|---|---|---|
| io_ctl | 617 | 762 | 192 | 19 | 1 |
| cp (cpu) | 617 | 762 | 192 | 0 | 0 |
| al (alu) | 6 | 0 | 0 | 0 | 0 |
| ctl (con | 276 | 5 | 80 | 0 | 0 |
| pr_cou | 144 | 5 | 64 | 0 | 0 |
| reg_file | 191 | 160 | 48 | 0 | 0 |