PROJECT

Harsh Mehta

CMPEN 331

Spring 2021

8th May 2021

ABSTRACT

For the final project, our primary motive was to eliminate the hazarding problems faced earlier in Lab 5. The project is a succession to a chain of labs, lab 3 → lab 4 → lab 5 → project. In order to negate the hazard control, this project focused on implementing forwarding. When a jump or a branch instruction is executed in a pipelined CPU, control hazards occur. Control hazards that are caused as a result of conditional branch instructions are more consequential compared to the jump instructions. There are two ways to deal with them, cancel them and delayed branch which would allow the instructions to be executed. This was accomplished by the addition of multiplexers as well as code to the control unit and regfile design. Control hazard negation was accomplished using two mux components. Furthermore, a design implementation that would verify the correctness of the pipelined CPU was made, aiming to check 20 instructions. It tests a subroutine in which four 32-bit memory words are summed by a for loop. After returning from the subroutine, the sum is stored in the data memory by a sw instruction. A code pattern that causes pipeline stall is also prepared within the loop.

INTRODUCTION

Hazard detection and elimination are the key concepts applied in this project. I was implemented on the processor designed starting from lab 3 till lab 5. There was 5 design stages, instruction fetching stage (IF), instruction decoding stage (ID), execution stage (EX), memory access stage (MEM), and writeback stage (WB) involving the processor with pipelining. Each of these stages accomplishes an important function. Pipelining is a technique involving multiple instructions overlapping during execution. This prevents the bottlenecking of the CPU to a single execution at once. With the ability for a new instruction to enter every clock cycle and taking 5 cycles to pass through the pipeline, high efficiency is achieved. Due to a very efficient pipelining design, the CPU can produce an optimum result every cycle.

VERILOG DESIGN CODE

```
`timescale 1ns / 1ps

module instruction(
input [31:0]addr,
output reg [31:0] data
   );
wire [31:0] rom [0:63];

 assign rom[6'h00] = 32'h3c010000; // (00) main: lui $1, 0
 assign rom[6'h01] = 32'h34240050; // (04) ori $4, $1, 80
 assign rom[6'h02] = 32'h0c00001b; // (08) call: jal sum
 assign rom[6'h03] = 32'h20050004; // (0c) dslot1: addi $5, $0, 4
 assign rom[6'h04] = 32'hac820000; // (10) return: sw $2, 0($4)
 assign rom[6'h05] = 32'h8c890000; // (14) lw $9, 0($4)
 assign rom[6'h06] = 32'h01244022; // (18) sub $8, $9, $4
 assign rom[6'h07] = 32'h20050003; // (1c) addi $5, $0, 3
 assign rom[6'h08] = 32'h20a5ffff; // (20) loop2: addi $5, $5, -1
 assign rom[6'h09] = 32'h34a8ffff; // (24) ori $8, $5, 0xffff
 assign rom[6'h0a] = 32'h39085555; // (28) xori $8, $8, 0x5555
 assign rom[6'h0b] = 32'h2009ffff; // (2c) addi $9, $0, -1
 assign rom[6'h0c] = 32'h312affff; // (30) andi $10,$9,0xffff
 assign rom[6'h0d] = 32'h01493025; // (34) or $6, $10, $9
 assign rom[6'h0e] = 32'h01494026; // (38) xor $8, $10, $9
 assign rom[6'h0f] = 32'h01463824; // (3c) and $7, $10, $6
 assign rom[6'h10] = 32'h10a00003; // (40) beq $5, $0, shift
 assign rom[6'h11] = 32'h00000000; // (44) dslot2: nop
 assign rom[6'h12] = 32'h08000008; // (48) j loop2
 assign rom[6'h13] = 32'h00000000; // (4c) dslot3: nop
 assign rom[6'h14] = 32'h2005ffff; // (50) shift: addi $5, $0, -1
 assign rom[6'h15] = 32'h000543c0; // (54) sll $8, $5, 15
 assign rom[6'h16] = 32'h00084400; // (58) sll $8, $8, 16
 assign rom[6'h17] = 32'h00084403; // (5c) sra $8, $8, 16
 assign rom[6'h18] = 32'h000843c2; // (60) srl $8, $8, 15
 assign rom[6'h19] = 32'h08000019; // (64) finish: j finish
 assign rom[6'h1a] = 32'h00000000; // (68) dslot4: nop

 assign rom[6'h1b] = 32'h00004020; // (6c) sum: add $8, $0, $0
 assign rom[6'h1c] = 32'h8c890000; // (70) loop: lw $9, 0($4)
 assign rom[6'h1d] = 32'h01094020; // (74) stall: add $8, $8, $9
 assign rom[6'h1e] = 32'h20a5ffff; // (78) addi $5, $5, -1
 assign rom[6'h1f] = 32'h14a0fffc; // (7c) bne $5, $0, loop
 assign rom[6'h20] = 32'h20840004; // (80) dslot5: addi $4, $4, 4
 assign rom[6'h21] = 32'h03e00008; // (84) jr $31
 assign rom[6'h22] = 32'h00081000; // (88) dslot6: sll $2, $8, 0

always @(addr)
begin
data = rom[addr[7:2]];
end
endmodule

/////////////////////////////////////////////////////////////////////////////////////////
`timescale 1ns / 1ps

module Adder(
input clk,
input [31:0] PC,
input [31:0] in1,
input [31:0]bpc,
input [31:0] da,
input [1:0] pcsrc,
output reg [31:0] PC_out
    );
reg [31:0] PC4=32'd100;
wire [31:0] PC_o;
MUX4to1 PC_4(PC4,bpc,da,PC,pcsrc,PC_o);
always @(*)
begin
if(PC<120)
PC4=PC + in1;
PC_out=PC_o;
end



endmodule

module MUX4to1(
input [31:0] a0,
input [31:0] a1,
input [31:0] a2,
input [31:0] a3,
input [1:0] sel,
output reg [31:0] y
    );
```

```
always @(*)
begin
case(sel)
0: y=a0;
1: y=a1;
2: y=a2;
3: y=a3;
endcase
end
endmodule

module fetch(
input clk,
input [31:0] instruction_data,
input [31:0] pc,
output reg [31:0] IF_ID,
output reg [31:0] dpc4

    );
//          reg [31:0] data=32'b0;

always @(posedge clk)
begin

//          data<=instruction_data;
            IF_ID<=instruction_data;
            dpc4<=pc;

end


endmodule

/////////////////////////////////////////////////////////////////////////////////
`timescale 1ns / 1ps

module Decode(
input clk,
input [31:0] IF_ID,
input [4:0] mrn,
input  mm2reg,
input  mwreg,
input [4:0] ern_ff,
input  em2reg_ff,
input  ewreg_ff,
input wwreg,
input [4:0] wrn,
input [31:0] mmo,
input [31:0] malu,
input [31:0] ealu,
input [31:0] wdi,
input [31:0] dpc4,
output reg ewreg,
output reg em2reg,
output reg ewmem,
output reg [3:0] ealuc,
output reg ealuimm,
output reg ejal,
output reg eshift,
output reg [1:0] pcsrc,
output reg [4:0] ern,
output reg [31:0] ea,
output reg [31:0] eb,
output reg [31:0] eimm,
output reg [31:0] ID_EXE,
output reg [31:0] epc4,
output reg [31:0] bpc,
output reg [31:0] daa
   );
          wire regRt,wreg,m2reg,wmem,aluimm,jal,shift;
          wire [3:0]aluc;
          wire [1:0] pcsrc_f,fwda,fwdb;
          wire [4:0] drn;
          wire [31:0] da,db,qa;
          wire [31:0] qb;
          reg [31:0] imm;
Control_unit
C1(IF_ID[31:26],IF_ID[5:0],mrn,mm2reg,mwreg,ern_ff,em2reg_ff,ewreg_ff,wreg,m2reg,wmem,
aluc,aluimm,regRt,jal,shift,pcsrc_f,fwda,fwdb);

assign drn=regRt? IF_ID[20:16]:IF_ID[15:11];

reg_file R1(clk,IF_ID[25:21],wwreg,wrn,wdi,qa,qb);

         MUX4to1 mux_fwda(qa,ealu,malu,mmo,fwda,da);
         MUX4to1 mux_fwdb(qb,ealu,malu,mmo,fwdb,db);
```

```
always @(posedge clk)
begin

        ID_EXE<=IF_ID;
        ern<=drn;
        ewreg<=wreg;
        em2reg<=m2reg;
        ewmem<=wmem;
        ealuc<=aluc;
        ealuimm<=aluimm;
        ejal<=jal;
        eshift<=shift;
        epc4<=dpc4;
        eb<=db;
        ea<=da;
        eimm<=imm;

end

always @(*)
begin
imm={{16{IF_ID[15]}},IF_ID[15:0]};
pcsrc=pcsrc_f;
bpc=dpc4+imm;
daa=da;
end
endmodule


////////////////////////////////////////////////////////////////////////////////
`timescale 1ns / 1ps

module Control_unit(
input [5:0] op,
input [5:0] func,
input [4:0] mrn,
input  mm2reg,
input  mwreg,
input  ern_ff,
input  em2reg_ff,
input  ewreg_ff,
output reg wreg,
output reg m2reg,
output reg wmem,
output reg [3:0] aluc,
output reg aluimm,
output reg regRt,
output reg jal,
output reg shift,
output reg [1:0] pcsrc,
output reg [1:0] fwda,
output reg [1:0] fwdb
    );
always @ (*)
begin
case(op)


                    // ADD
                    6'b100000: begin
                              aluc = 4'b0010;
                              wreg = 1;
                              m2reg = 0;
                              aluimm = 0;
                              regRt = 0;
                              wmem = 0;
                              pcsrc=0;
                              jal=0;
                              shift=0;
                              fwda=0;
                    fwdb=0;
                    end

                    // SUB
                    6'b100010: begin
                              aluc = 4'b0110;
                              wreg = 1;
                              m2reg = 0;
                              aluimm = 0;
                              regRt = 0;
                              wmem = 0;
                              pcsrc=0;
                              jal=0;
                              shift=0;
                              fwda=0;
                    fwdb=0;
                    end
```

```
// AND
6'b100100: begin
        aluc = 4'b0000;
        wreg = 1;
        m2reg = 0;
        aluimm = 0;
        regRt = 0;
        wmem = 0;
        fwda=0;
        pcsrc=0;
        jal=0;
        shift=0;
fwdb=0;
end


6'b100101: begin
        aluc = 4'b0001;
        wreg = 1;
        m2reg = 0;
        aluimm = 0;
        regRt = 0;
        wmem = 0;
        fwda=0;
        pcsrc=0;
        jal=0;
        shift=0;
fwdb=0;
end


6'b100110: begin
        aluc = 4'b0010;
        wreg = 1;
        m2reg = 0;
        aluimm = 0;
        regRt = 0;
        wmem = 0;
        pcsrc=0;
        jal=0;
        shift=0;
        fwda=0;
fwdb=0;
end
```

```
6'b100011: begin
        aluc = 4'b0010;
        wreg = 1;
        m2reg = 1;
        aluimm = 1;
        regRt = 1;
        wmem = 0;
        pcsrc=0;
        jal=0;
        shift=0;
        fwda=0;
        fwdb=0;
end

// SW
6'b101011: begin
        aluc = 4'b0010;
        wreg = 0;
        m2reg = 0;
        aluimm = 1;
        regRt = 1;
        wmem = 1;
        fwda=0;
        jal=0;
        shift=0;
        pcsrc=0;
        fwdb=0;
end


default: begin
        aluc = 4'b0000;
        wreg = 0;
        m2reg = 0;
        aluimm = 0;
        regRt = 0;
        wmem = 0;
        pcsrc=0;
        jal=0;
        shift=0;
        fwda=0;
        fwdb=0;
end
```

```
            endcase

end

endmodule


/////////////////////////////////////////////////////////////////////////////////
`timescale 1ns / 1ps

module reg_file(
input clk,
input [4:0] address,
input we,
input [4:0] wn,
input [31:0] d,
output reg [31:0] qa,
output reg [31:0] qb
    );

reg [31:0] register [31:0];
 integer i;
 initial
 begin
  for(i=0;i<32;i=i+1)
  begin
                    register[i] =0;
  end
   end



always @(negedge clk)
            begin
            if (we)
            register[wn]<=d;

end

always @(posedge clk)
            begin
            qa<=register[address];
            qb<=register[address];
end
```

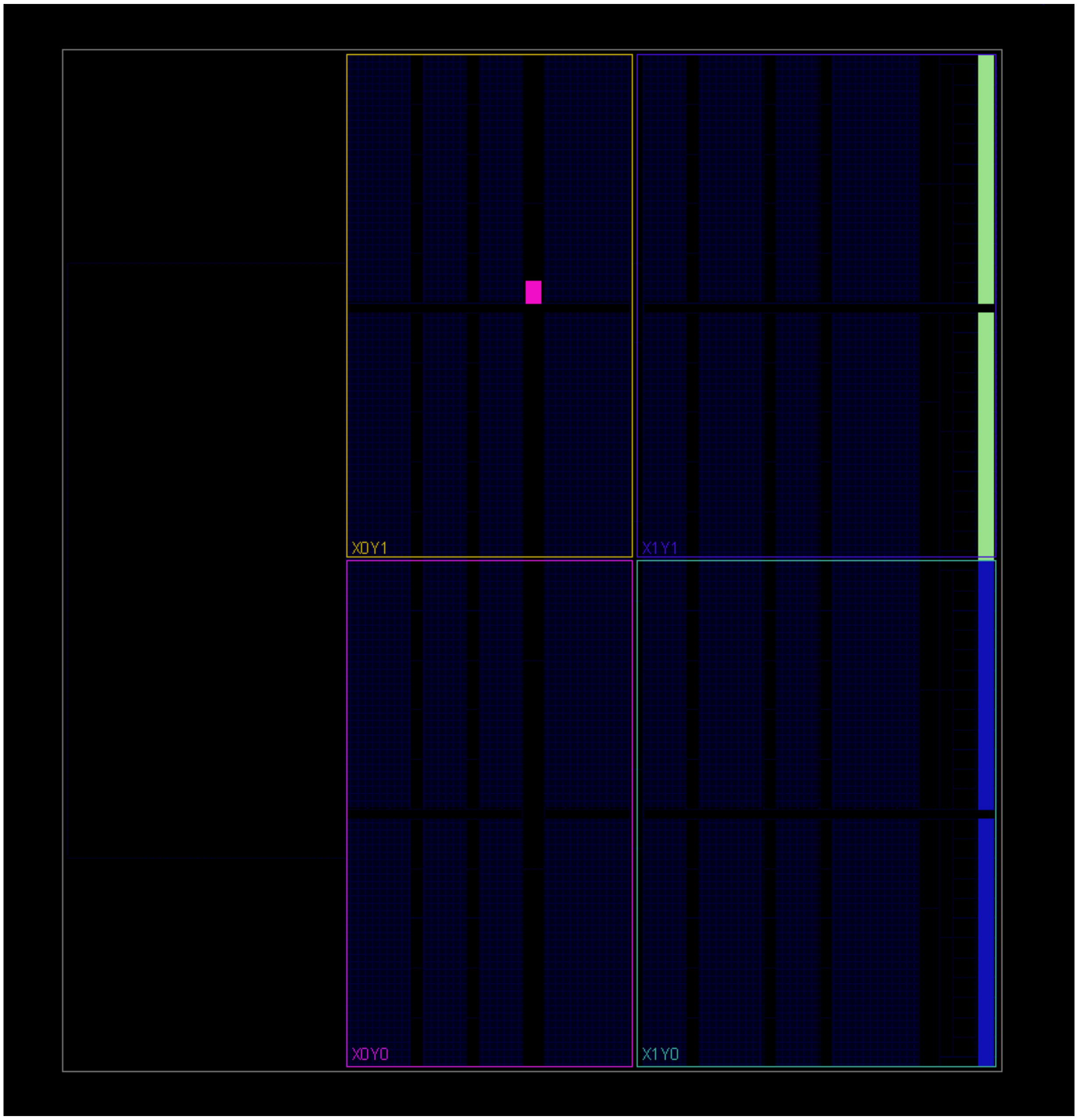
```
endmodule




////////////////////////////////////////////////////////////////////////////////
`timescale 1ns / 1ps

module Execution(
input clk,
input [31:0] ID1,
input ewreg,
input em2reg,
input ewmem,
input [3:0] ealuc,
input ealuimm,
input ejal,
input eshift,
input [4:0] ern,
input [31:0] epc4,
output [31:0] r,
output reg mwreg,
output reg mm2reg,
output reg mwmem,
input  [31:0] eimm,
output reg [31:0] malu,
input  [31:0] ea,
input  [31:0] eb,
output reg [31:0] di,
output reg [4:0] mrn,
output reg [31:0] EXE,
output  [31:0] ealu
    );
wire [31:0] a,b,ealu,epc8;
assign a=(eshift)? eimm:ea;
assign b=(ealuimm)? eimm:eb;
ALU ALU_unit(ealuc,a,b,r);

assign epc8=epc4+32'd4;
assign ealu= (ejal)? epc8:r;

always @(posedge clk)
begin
        mwreg<=ewreg;
        mm2reg<=em2reg;
        mwmem<=ewmem;
        EXE<=ID1;
        malu<=ealu;
        di<=eb;
        mrn<=ern;
end

endmodule

////////////////////////////////////////////////////////////////////////////////
`timescale 1ns / 1ps

module ALU(
input [3:0] aluc,
input [31:0] qa,
input [31:0] b,
output reg [31:0] r
    );
always @(*)
begin
case(aluc)
4'b0000:   r<=qa&b;
4'b0001:   r<=qa|b;
4'b0010:    r<=qa+b;
4'b0110:    r<=qa-b;
4'b0111:    r<=qa^b;
4'b1100:    r<=~(qa|b);
endcase
end
endmodule

////////////////////////////////////////////////////////////////////////////////
`timescale 1ns / 1ps

module Memory(
input clk,
input [31:0] EXE,
input mwreg,
input mm2reg,
input mwmem,
input [31:0] malu,
input [31:0] di,
input [4:0] mrn,
output reg wwreg,
output reg wm2reg,
output reg [31:0] wd0,
output reg [31:0] mmo,
output reg [31:0] walu,
output reg [4:0] wrn,
output reg [31:0] MEM
    );
wire [31:0] mmo_ff;
Data_Memory DM(mwmem,malu,di,mmo_ff);

always @(posedge clk)
begin
        MEM<=EXE;
        walu<=malu;
        wrn<=mrn;
        wd0<=mmo;
        wwreg<=mwreg;
        wm2reg<=mm2reg;
end
always @(*)
begin
mmo=mmo_ff;
end

endmodule

/////////////////////////////////////////////////////////////////////////////////
`timescale 1ns / 1ps

module Data_Memory(
input mwmem,
input [31:0] r,
input [31:0] di,
output reg [31:0] d0
    );

reg [31:0] memory [0:31];
 integer i=0;
initial
begin
 for(i=10;i<31;i=i+1)
  memory[i]=0;
  memory[0]=32'hA00000AA;
  memory[1]=32'h10000011;
  memory[2]=32'h20000022;
  memory[3]=32'h30000033;
  memory[4]=32'h40000044;
  memory[5]=32'h50000055;
  memory[6]=32'h60000066;
  memory[7]=32'h70000077;
  memory[8]=32'h80000088;
  memory[9]=32'h90000099;
  memory[5'h14] = 32'h000000a3; // (50) data[0] 0 + a3 = a3
  memory[5'h15] = 32'h00000027; // (54) data[1] a3 + 27 = ca
  memory[5'h16] = 32'h00000079; // (58) data[2] ca + 79 = 143
  memory[5'h17] = 32'h00000115; // (5c) data[3] 143 + 115 = 258

end

        always@(*)
        begin
        if(mwmem)
                memory[r[6:2]]<=di;

                d0<=memory[r[6:2]];
        end

endmodule

/////////////////////////////////////////////////////////////////////////////////
`timescale 1ns / 1ps

module write_back(
input wwreg,
input wm2reg,
input [31:0] walu,
input [31:0] wd0,
input [4:0] wrn,
output reg [31:0] wdi,
output reg we
   );


always @(*)
begin
        wdi<= wm2reg ? wd0:walu;
        we<= wwreg;
end
endmodule
```

VERILOG TESTBENCH CODE

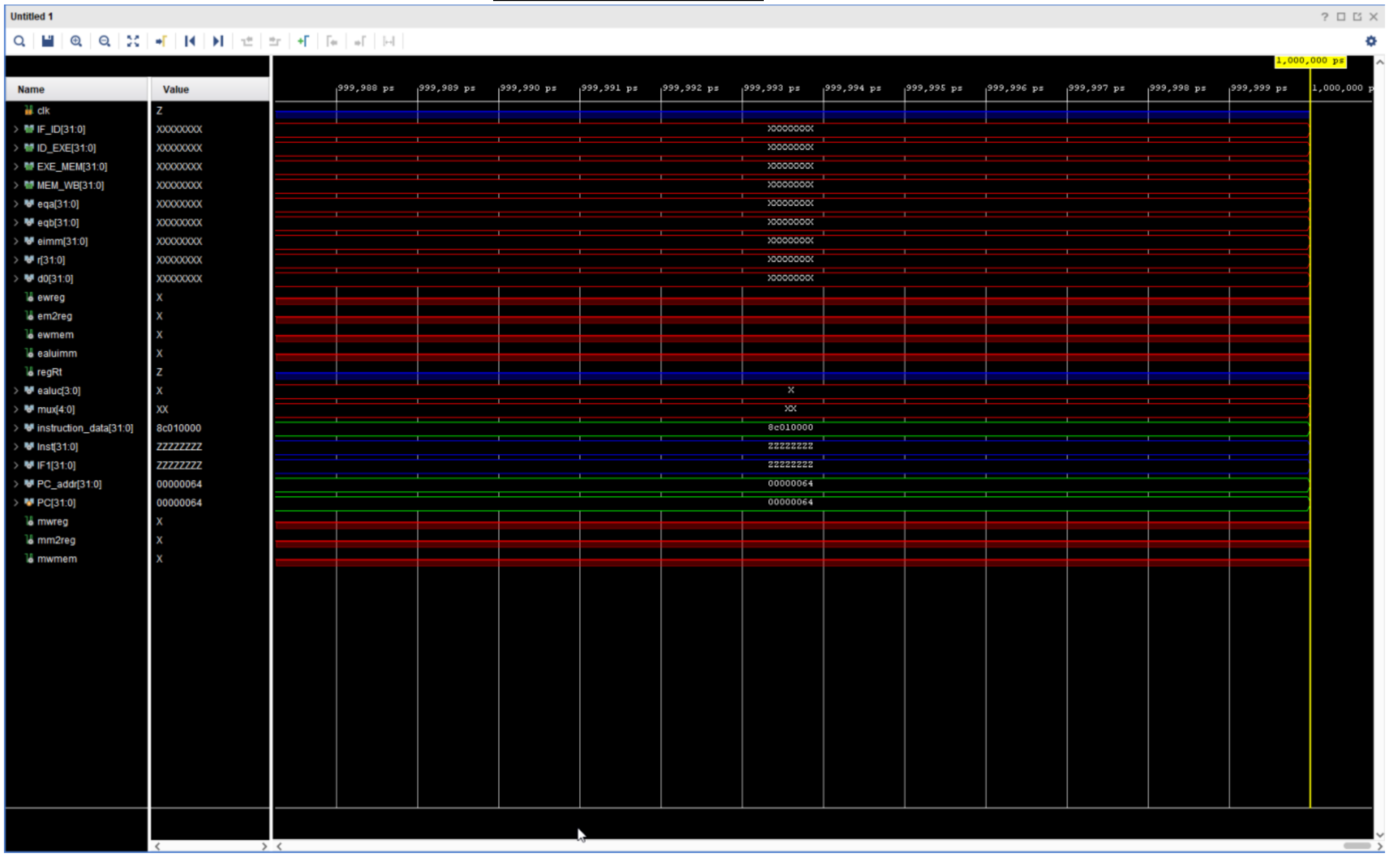
```
`timescale 1ns / 1ps

module Top(
input clk

  );
        reg [31:0]PC;
  wire [31:0] qa,qb,imm,r,d0,d,bpc,ealu,malu,walu,mmo,wdi,eimm,wd0;
        wire [31:0]di,da,db,ea,eb;
        wire wreg,m2reg,wmem,aluimm,regRt,wwreg,wm2reg,ejal,eshift;
        wire [1:0] pcsrc;
        wire [3:0] aluc,ealuc;
        wire [4:0] drn,ern,wrn,mrn,wn;
  wire [31:0] instruction_data,Inst,IF1,MEM1,EXE1,ID1;
        reg [31:0] IF,MEM,EXE,ID;
        wire [31:0]PC_addr_f,dpc4,epc4;
        reg [31:0] PC_addr=32'd100;


initial
        begin
        PC=100;
        end



instruction I1( PC_addr, instruction_data);
Adder Add_4(clk, PC,32'd4,bpc,da,pcsrc,PC_addr_f);
fetch F1(clk,instruction_data,PC,IF1,dpc4);
Decode
D1(clk,IF1,mrn,mm2reg,mwreg,ern,em2reg,ewreg,wwreg,wrn,mmo,malu,ealu,wdi,dpc4,ewreg,
em2reg,ewmem,ealuc,ealuimm,ejal,eshift,pcsrc,ern,ea,eb,eimm,ID1,epc4,bpc,da);
Execution
E1(clk,ID1,ewreg,em2reg,ewmem,ealuc,ealuimm,ejal,eshift,ern,epc4,r,mwreg,mm2reg,mwme
m,eimm,malu,ea,eb,di,mrn,EXE1,ealu);
Memory
M1(clk,EXE1,mwreg,mm2reg,mwmem,malu,di,mrn,wwreg,wm2reg,wd0,mmo,walu,wrn,MEM1
);
```

```
write_back WB(wwreg,wm2reg,walu,wd0,wrn,wdi,we);

always @(posedge clk)
begin
        PC=PC_addr;
        IF<=IF1;
        MEM<=MEM1;
        EXE<=EXE1;
        ID<=ID1;
end
always @(*)
begin
        PC_addr=PC_addr_f;
end
endmodule
```

FLOOR PLANNING SCHEMATIC



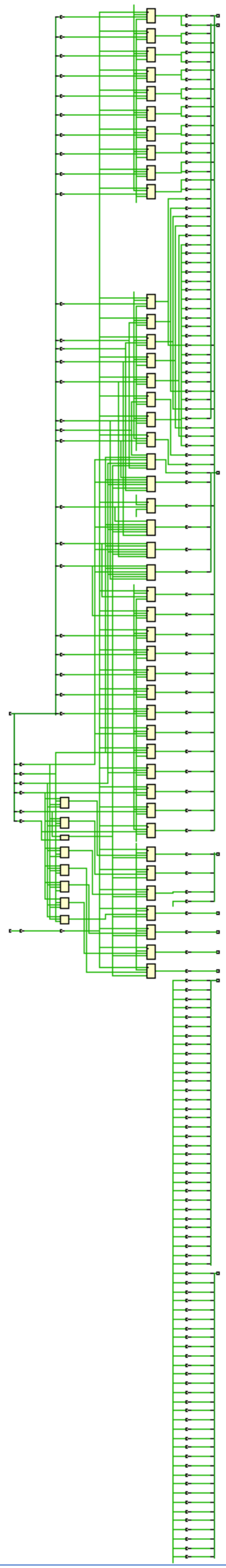
WAVEFORMS



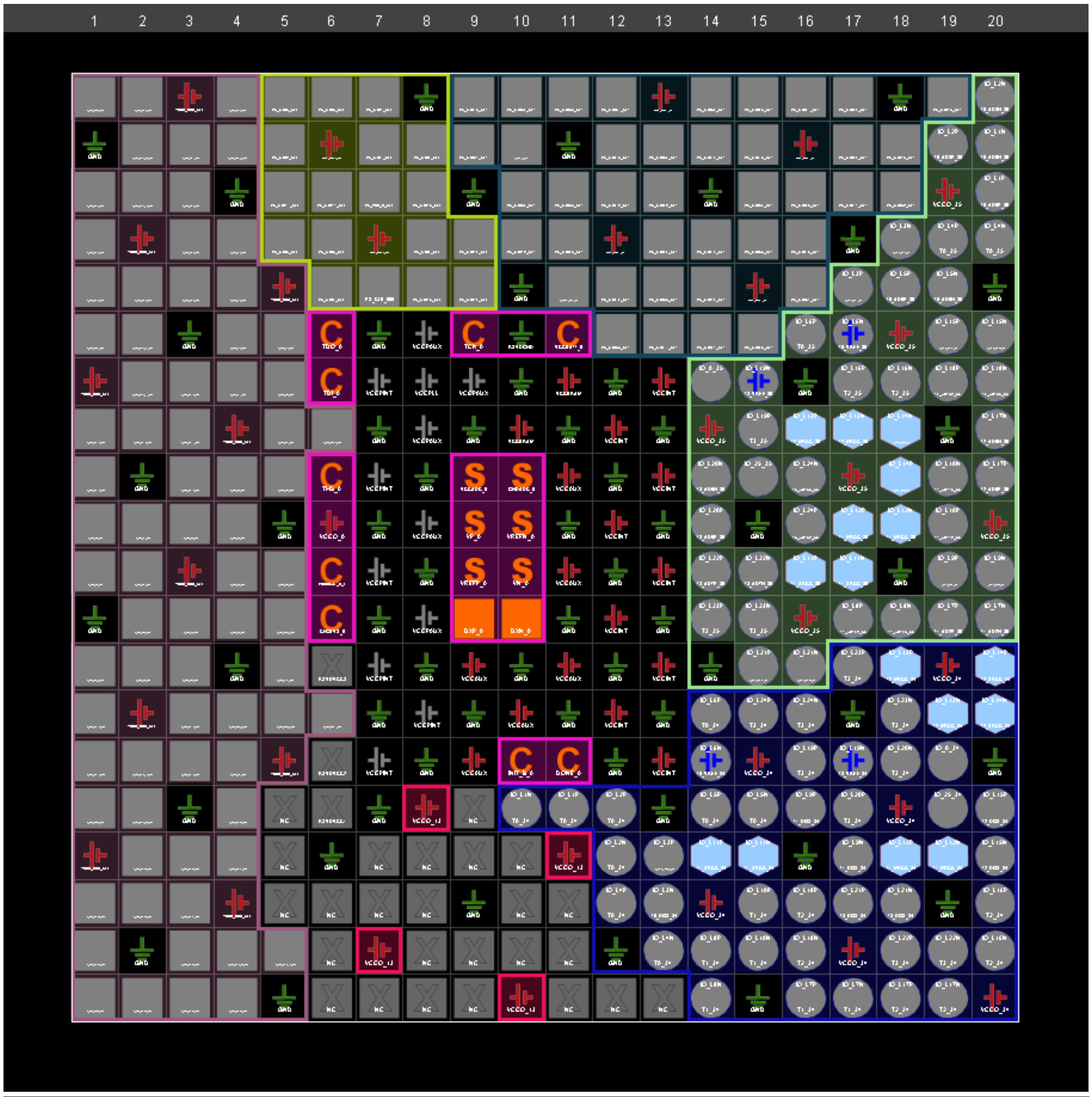
Untitled 1

Name	Value
clk	Z
IF_ID[31:0]	XXXXXXXX
ID_EXE[31:0]	XXXXXXXX
EXE_MEM[31:0]	XXXXXXXX
MEM_WB[31:0]	XXXXXXXX
eqa[31:0]	XXXXXXXX
eqb[31:0]	XXXXXXXX
eimm[31:0]	XXXXXXXX
r[31:0]	XXXXXXXX
d0[31:0]	XXXXXXXX
ewreg	X
em2reg	X
ewmem	X
ealuimm	X
regRt	Z
ealuc[3:0]	X
mux[4:0]	XX
instruction_data[31:0]	8c010000
Inst[31:0]	ZZZZZZZZ
IF1[31:0]	ZZZZZZZZ
PC_addr[31:0]	00000064
PC[31:0]	00000064
mwreg	X
mm2reg	X
mwmem	X

GATE SCHEMATIC



I/O PLANNING


1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20