CPU Design Part 3: WB Stage

CMPEN 331 001 Lab 5

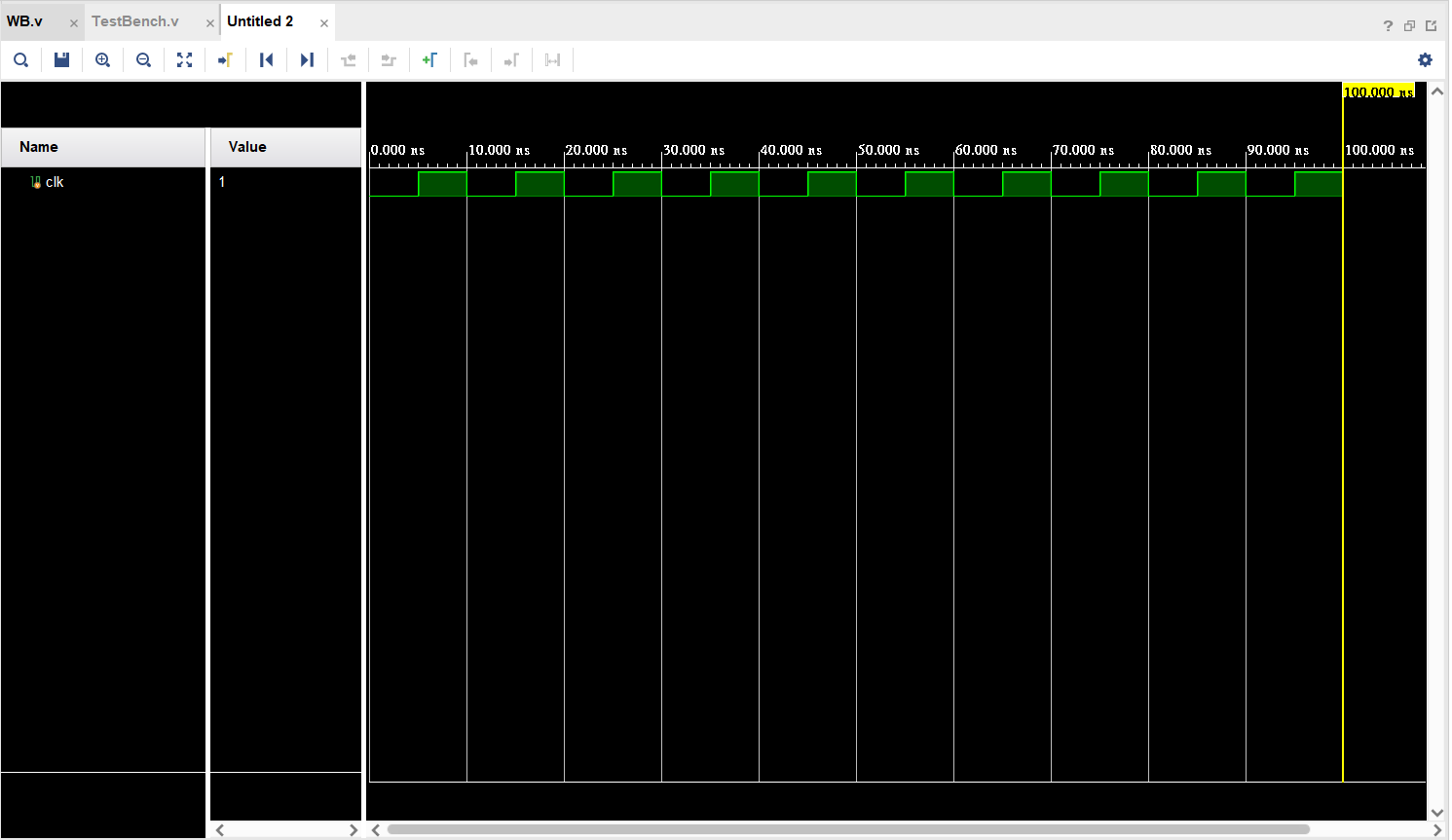
Summer 2021

Shao-Ju Wang

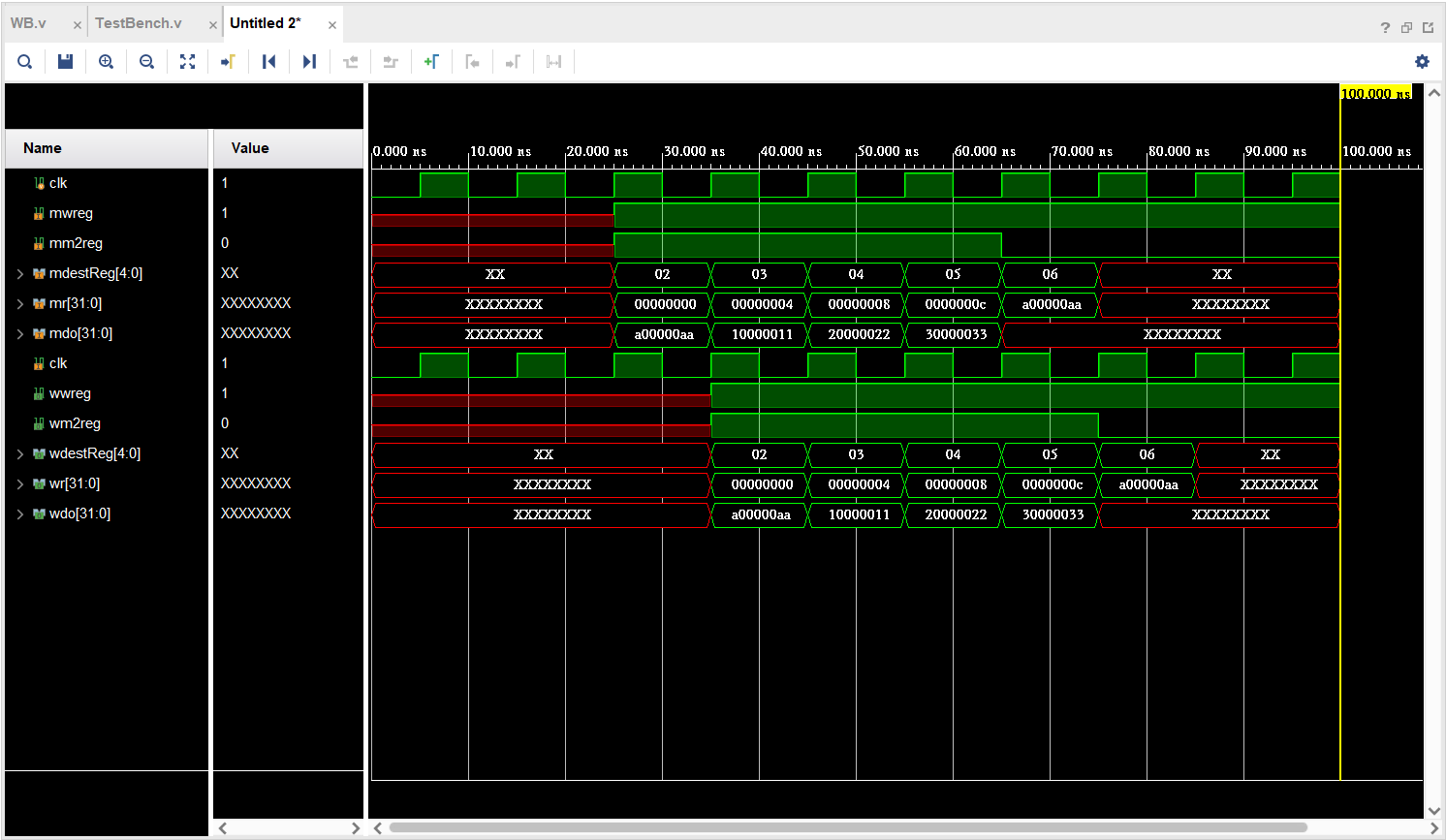
\*All Codes are at the bottom of this report as it’s too long.

**Waveform:**

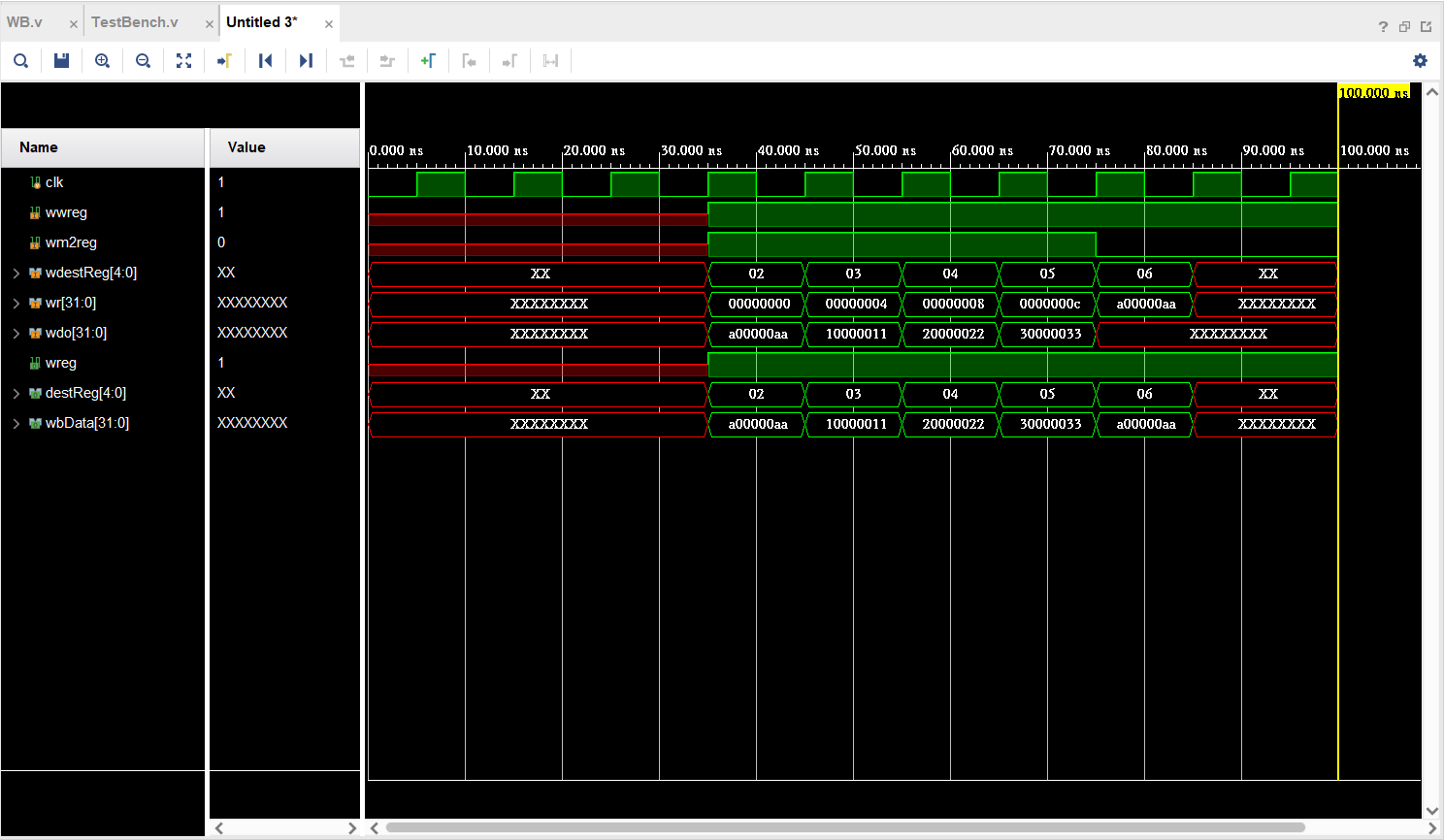
Raw/Default Output:



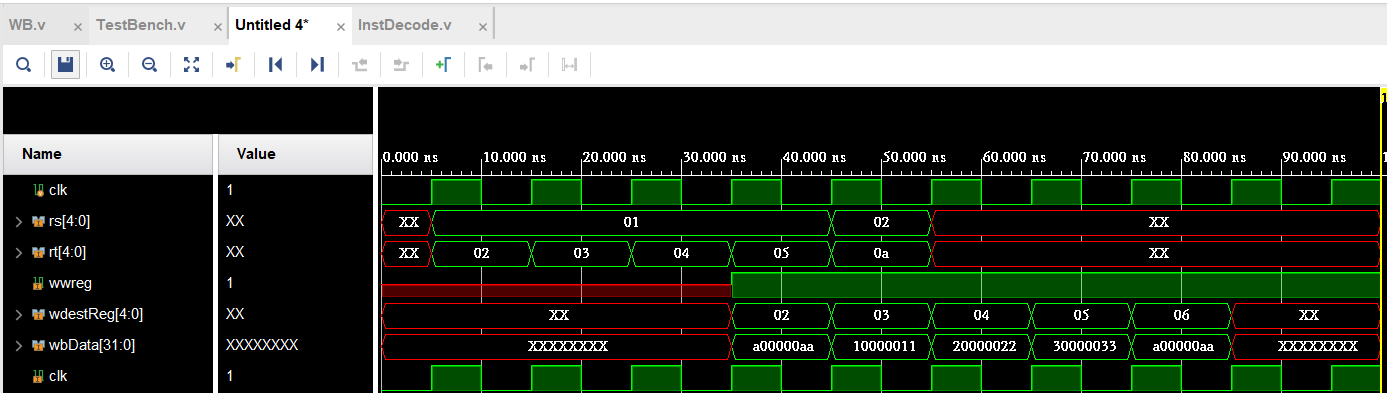
Mem/WB:



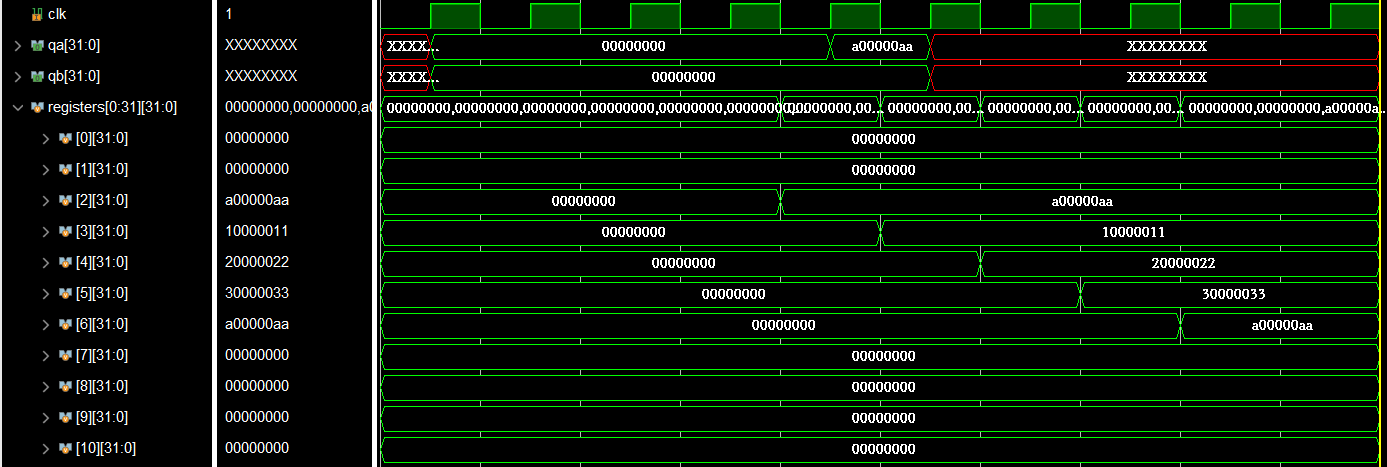
WB:



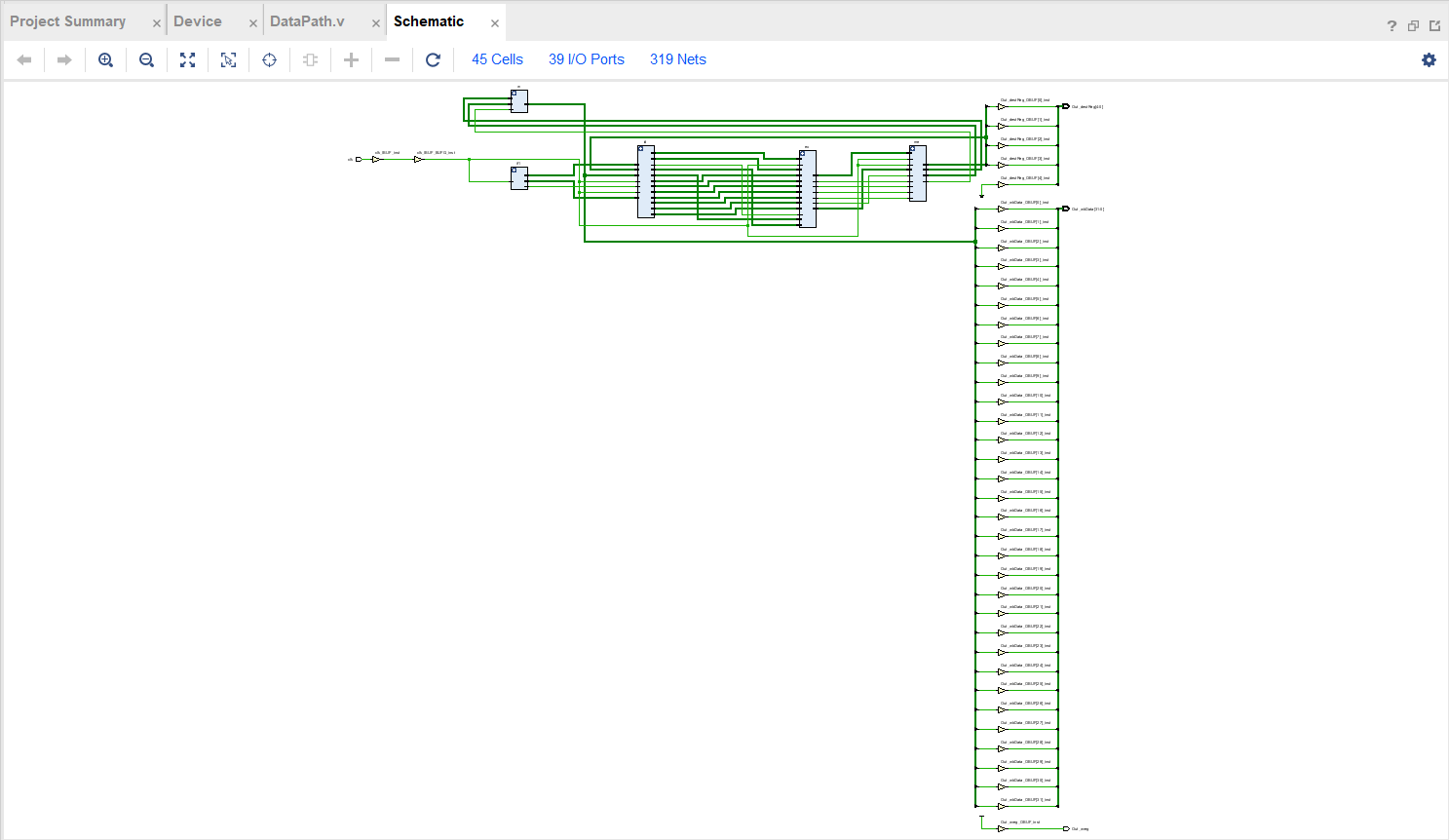
RegFile (Input):



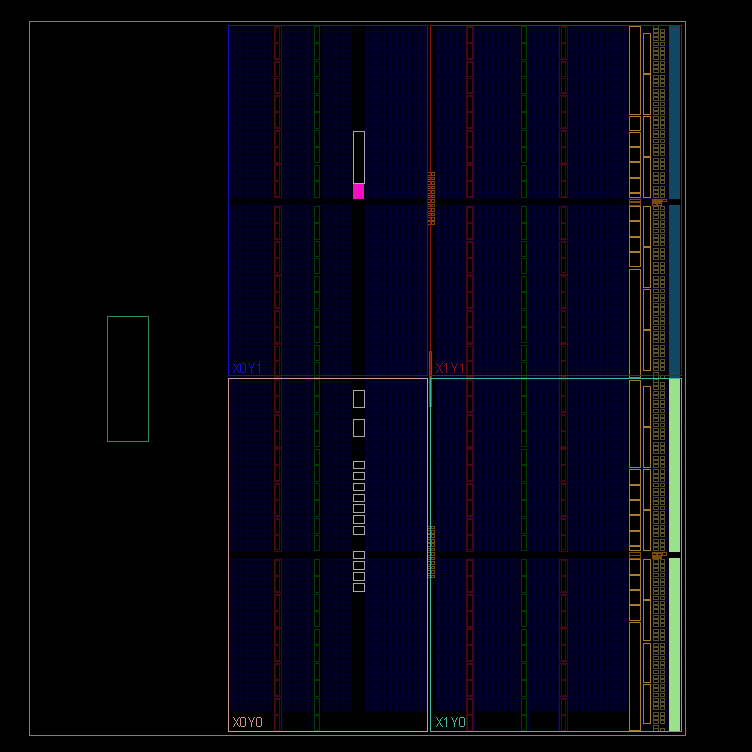
RegFile (Registers):



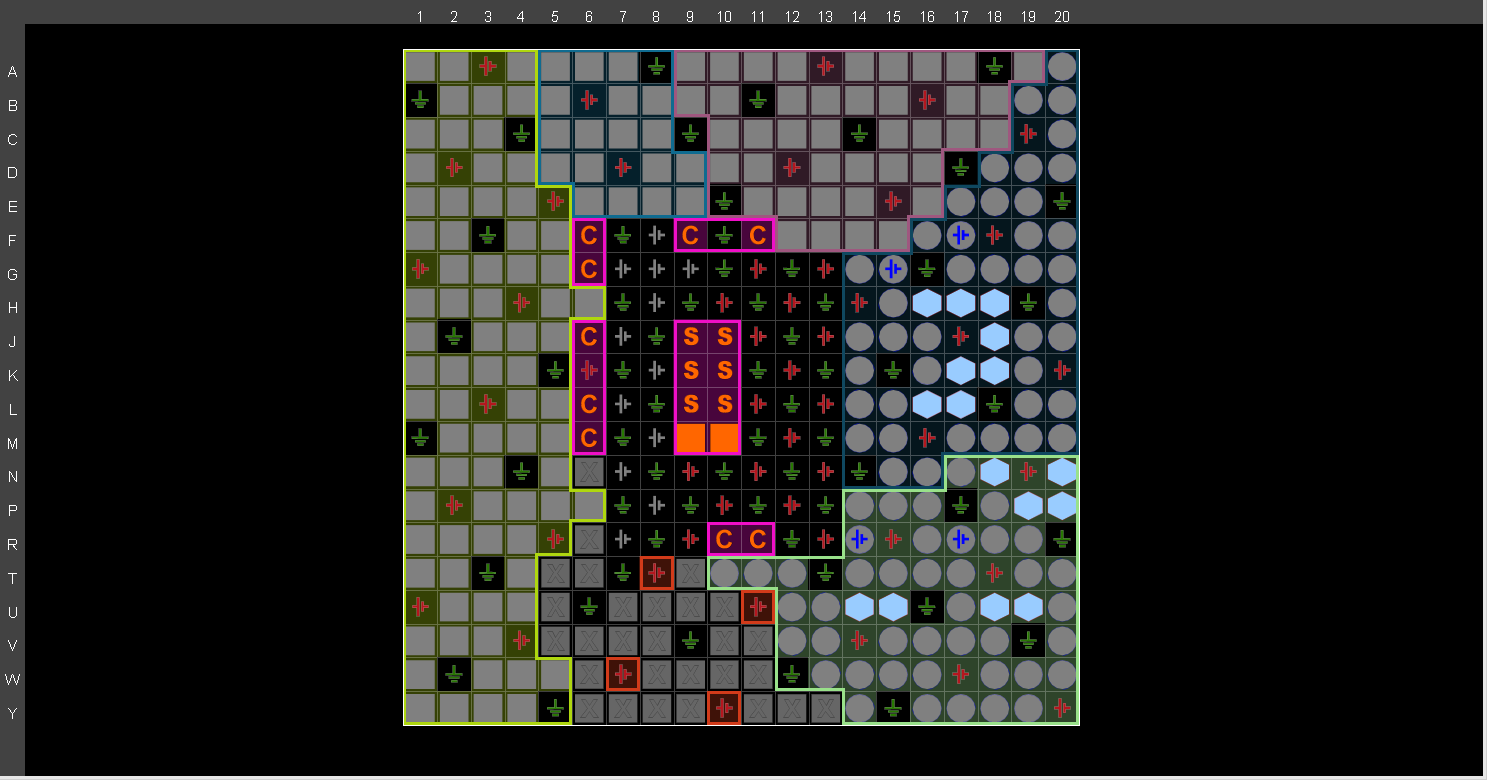
**Design Schematic from synthesis (Datapath output is added, connected to WB output to form schematic):**



**Floor Planning:**



**I/O Planning:**



**Code** (Collapsible)**:**

**IF (InstFetch.v):**

`timescale 1ns/1ps

module pc(

input [31:0] nextPC ,

input clk,

output reg [31:0] pc

);

initial pc = 100;

always @(posedge clk) begin

pc <= nextPC;

end

endmodule

module instMem(

input [31:0] pc,

output reg [31:0] instOut

);

reg [31:0] memory [0:63];

initial begin // Temp Assigned value

memory[25] = {

6'b100011,

5'b00001,

5'b00010,

5'b00000,

5'b00000,

6'b000000

};

memory[26] = {

6'b100011,

5'b00001,

5'b00011,

5'b00000,

5'b00000,

6'b000100

};

memory[27] = {

6'b100011,

5'b00001,

5'b00100,

5'b00000,

5'b00000,

6'b001000

};

memory[28] = {

6'b100011,

5'b00001,

5'b00101,

5'b00000,

5'b00000,

6'b001100

};

memory[29] = {

6'b000000,

5'b00010, //2

5'b01010, //10

5'b00110, //6

5'b00000,

6'b100000

};

end

always @ (\*) begin

instOut <= memory[pc[7:2]];

end

endmodule

module pcAdder(

input [31:0] pc,

//input [31:0] increment, reserved for j inst

output reg [31:0] nextPC

);

always @ (\*) begin

nextPC <= pc + 4;

end

endmodule

module IFIDreg(

input [31:0] instOut,

input clk,

output reg [31:0] dinstOut

);

always @ (posedge clk) begin

dinstOut <= instOut;

end

endmodule

module InstFetch(

input clk,

output [31:0] dinstOut

);

wire [31:0] wpc;

wire [31:0] wnextPC;

wire [31:0] winstOut;

pc pc1(wnextPC, clk, wpc);

pcAdder pcadd(wpc, wnextPC);

instMem inmem(wpc, winstOut);

IFIDreg IFID(winstOut, clk, dinstOut);

endmodule

**ID (InstDecode.v):**

module Control(

input [5:0] op,

input [5:0] func,

output reg wreg,

output reg m2reg,

output reg wmem,

output reg [3:0] aluc,

output reg aluImm,

output reg regrt

);

always @(\*) begin

case (op)

6'b000000: //r-type

begin

case (func)

6'b100000: //Add

begin

wreg <= 1;

m2reg <= 0;

wmem <= 0;

aluc <= 4'b0010;

aluImm <= 0;

regrt <= 0;

end

// 6'b100010: //sub

// begin

// wreg <= 1;

// m2reg <= 0;

// wmem <= 0;

// aluc <= 4'b0110;

// aluImm <= 0;

// regrt <= 0;

// end

// 6'b100100: //and

// begin

// wreg <= 1;

// m2reg <= 0;

// wmem <= 0;

// aluc <= 4'b0000;

// aluImm <= 0;

// regrt <= 0;

// end

// 6'b100101: //or

// begin

// wreg <= 1;

// m2reg <= 0;

// wmem <= 0;

// aluc <= 4'b0001;

// aluImm <= 0;

// regrt <= 0;

// end

// 6'b100110: //xor // not sure

// begin

// wreg <= 1;

// m2reg <= 0;

// wmem <= 0;

// aluc <= 0110; //Unsure

// aluImm <= 0;

// regrt <= 0;

// end

// 6'b000000: //sll

// begin

// end

// 6'b000010: //srl

// begin

// end

endcase

end

6'b100011: //lw

begin

wreg <= 1;

m2reg <= 1;

wmem <= 0;

aluc <= 4'b0010;

aluImm <= 1;

regrt <= 1;

end

// 6'b101011: //sw

// begin

// wreg <= 0;

// m2reg <= 0;

// wmem <= 1;

// aluc <= 4'b0010;

// aluImm <= 1;

// regrt <= 1;

// end

// 6'b001000: //addi //not sure

// begin

// wreg <= 1;

// m2reg <= 0;

// wmem <= 0;

// aluc <= 4'b0010;

// aluImm <= 1;

// regrt <= 1;

// end

// 6'b001100: //andi //not sure

// begin

// wreg <= 1;

// m2reg <= 0;

// wmem <= 0;

// aluc <= 0000;

// aluImm <= 1;

// regrt <= 1;

// end

// 6'b001101: //ori //not sure

// begin

// wreg <= 1;

// m2reg <= 0;

// wmem <= 0;

// aluc <= 0001;

// aluImm <= 1;

// regrt <= 1;

// end

// 6'b001110: //xori

// begin

// end

// 6'b000100: //beq

// begin

// end

// 6'b000101: //bne

// begin

// end

// 6'b001111: //lui

// begin

// end

// 6'b000010: //j

// begin

// end

// 6'b000011: //jal

// begin

// end

endcase

end

endmodule

module regRTMux(

input [4:0] rt,

input [4:0] rd,

input regrt,

output reg [4:0] destReg

);

always @ (\*) begin

if (~regrt) destReg <= rd;

else if (regrt) destReg <= rt;

end

endmodule

module regFile(

input [4:0] rs,

input [4:0] rt,

input wwreg,

input [4:0] wdestReg,

input [31:0] wbData,

input clk,

output reg [31:0] qa,

output reg [31:0] qb

);

reg [31:0] registers [0:31];

initial //Initialize RegFile

begin

registers[0] = 32'd0;

registers[1] = 32'd0;

registers[2] = 32'd0;

registers[3] = 32'd0;

registers[4] = 32'd0;

registers[5] = 32'd0;

registers[6] = 32'd0;

registers[7] = 32'd0;

registers[8] = 32'd0;

registers[9] = 32'd0;

registers[10] = 32'd0;

registers[11] = 32'd0;

registers[12] = 32'd0;

registers[13] = 32'd0;

registers[14] = 32'd0;

registers[15] = 32'd0;

registers[16] = 32'd0;

registers[17] = 32'd0;

registers[18] = 32'd0;

registers[19] = 32'd0;

registers[20] = 32'd0;

registers[21] = 32'd0;

registers[22] = 32'd0;

registers[23] = 32'd0;

registers[24] = 32'd0;

registers[25] = 32'd0;

registers[26] = 32'd0;

registers[27] = 32'd0;

registers[28] = 32'd0;

registers[29] = 32'd0;

registers[30] = 32'd0;

registers[31] = 32'd0;

end

always @ (\*) begin

qa = registers[rs];

qb = registers[rt];

end

always @ (negedge clk) begin

if (wwreg) begin

registers[wdestReg] <= wbData;

end

end

endmodule

module ImmExt(

input [15:0] imm,

output reg [31:0] imm32

);

always @ (\*) begin

imm32 <= {{17{imm[15]}}, imm[14:0]};

end

endmodule

module IDEXEreg(

input wreg,

input m2reg,

input wmem,

input [3:0] aluc,

input aluimm,

input [4:0] destReg,

input [31:0] qa,

input [31:0] qb,

input [31:0] imm32,

input clk,

output reg owreg,

output reg om2reg,

output reg owmem,

output reg [3:0] oaluc,

output reg oaluimm,

output reg [4:0] odestReg,

output reg [31:0] oqa,

output reg [31:0] oqb,

output reg [31:0] oimm32

);

always @ (posedge clk) begin

owreg <= wreg;

om2reg <= m2reg;

owmem <= wmem;

oaluc <= aluc;

oaluimm <= aluimm;

odestReg <= destReg;

oqa <= qa;

oqb <= qb;

oimm32 <= imm32;

end

endmodule

module InstDecode(

input [31:0] dinstOut,

input wwreg,

input [4:0] wdestReg,

input [31:0] wbData,

input clk,

output owreg,

output om2reg,

output owmem,

output [3:0] oaluc,

output oaluimm,

output [4:0] odestReg,

output [31:0] oqa,

output [31:0] oqb,

output [31:0] oimm32

);

wire wreg;

wire wm2reg;

wire wwmem;

wire [3:0] waluc;

wire waluimm;

wire [4:0] destReg;

wire [31:0] wqa;

wire [31:0] wqb;

wire [31:0] wimm32;

wire wrtMUX;

Control c(dinstOut[31:26], dinstOut[5:0], wreg, wm2reg, wwmem, waluc, waluimm, wrtMUX);

regRTMux rtMux(dinstOut[20:16], dinstOut[15:11], wrtMUX, destReg);

regFile rf(dinstOut[25:21], dinstOut[20:16], wwreg, wdestReg, wbData, clk, wqa, wqb);

ImmExt extend(dinstOut[15:0], wimm32);

IDEXEreg idexe1(wreg, wm2reg, wwmem, waluc, waluimm, destReg, wqa, wqb, wimm32, clk, owreg, om2reg,

owmem, oaluc, oaluimm, odestReg, oqa, oqb, oimm32);

endmodule

**Execution (Exe.v):**

module aluMUX (

input [31:0] eqb,

input [31:0] eimm,

input ealuimm,

output reg [31:0] b

);

always @ (\*)

begin

if (ealuimm) b <= eimm;

else b <= eqb;

end

endmodule

module Alu(

input [31:0] eqa,

input [31:0] b,

input [3:0] ealuc,

output reg [31:0] r

);

always @(\*)

begin

case (ealuc)

4'b0000: r <= eqa + b;

4'b0001: r <= eqa + b;

4'b0010: r <= eqa + b;

endcase

end

endmodule

module exemem(

input ewreg,

input em2reg,

input ewmem,

input [4:0] edestReg,

input [31:0] r,

input [31:0] eqb,

input clk,

output reg mwreg,

output reg mm2reg,

output reg mwmem,

output reg [4:0] mdestReg,

output reg [31:0] mr,

output reg [31:0] mqb

);

always @ (posedge clk) begin

mwreg <= ewreg;

mm2reg <= em2reg;

mwmem <= ewmem;

mdestReg <= edestReg;

mr <= r;

mqb <= eqb;

end

endmodule

module Exe(

input ewreg,

input em2reg,

input ewmem,

input [3:0] ealuc,

input ealuimm,

input [4:0] edestReg,

input [31:0] eqa,

input [31:0] eqb,

input [31:0] eimm32,

input clk,

output mwreg,

output mm2reg,

output mwmem,

output [4:0] mdestReg,

output [31:0] mr,

output [31:0] mqb

);

wire [31:0] rWire;

wire [31:0] bWire;

aluMUX almu(eqb, eimm32, ealuimm, bWire);

Alu alu(eqa, bWire, ealuc, rWire);

exemem em(ewreg, em2reg, ewmem, edestReg, rWire, eqb, clk, mwreg, mm2reg, mwmem, mdestReg, mr, mqb);

endmodule

**Memory (Mem.v):**

`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////

// Company:

// Engineer:

//

// Create Date: 2021/07/15 21:32:52

// Design Name:

// Module Name: Mem

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

//////////////////////////////////////////////////////////////////////////////////

module Main\_Mem(

input [31:0] mr,

input [31:0] mqb,

input mwmem,

input clk,

output reg [31:0] mdo

);

reg [31:0] mem [0:63];

initial begin

mem[0] = 32'hA00000AA;

mem[1] = 32'h10000011;

mem[2] = 32'h20000022;

mem[3] = 32'h30000033;

mem[4] = 32'h40000044;

mem[5] = 32'h50000055;

mem[6] = 32'h60000066;

mem[7] = 32'h70000077;

mem[8] = 32'h80000088;

mem[9] = 32'h90000099;

end

always @ (\*) begin

mdo <= mem[mr[7:2]];

end

always @ (negedge clk) begin

if (mwmem) begin

mem[mr[7:2]] <= mqb;

end

end

endmodule

module memwb(

input mwreg,

input mm2reg,

input [4:0] mdestReg,

input [31:0] mr,

input [31:0] mdo,

input clk,

output reg wwreg,

output reg wm2reg,

output reg [4:0] wdestReg,

output reg [31:0] wr,

output reg [31:0] wdo

);

always @ (posedge clk) begin

wwreg <= mwreg;

wm2reg <= mm2reg;

wdestReg <= mdestReg;

wr <= mr;

wdo <= mdo;

end

endmodule

module Mem(

input mwreg,

input mm2reg,

input mwmem,

input [4:0] mdestReg,

input [31:0] mr,

input [31:0] mqb,

input clk,

output wwreg,

output wm2reg,

output [4:0] wdestReg,

output [31:0] wr,

output [31:0] wdo

);

wire [31:0] doWire;

Main\_Mem m(mr, mqb, mwmem, clk, doWire);

memwb mw(mwreg, mm2reg, mdestReg, mr, doWire, clk, wwreg, wm2reg, wdestReg, wr, wdo);

endmodule

**Write Back (WB.v):**

`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////

// Company:

// Engineer:

//

// Create Date: 2021/07/31 12:42:51

// Design Name:

// Module Name: WB

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

//////////////////////////////////////////////////////////////////////////////////

module WbMUX(

input wm2reg,

input [31:0] wr,

input [31:0] wdo,

output [31:0] wbData

);

assign wbData = wm2reg ? wdo : wr;

endmodule

module WB(

input wwreg,

input wm2reg,

input [4:0] wdestReg,

input [31:0] wr,

input [31:0] wdo,

output wreg,

output [4:0] destReg,

output [31:0] wbData

);

assign wreg = wwreg;

assign destReg = wdestReg;

WbMUX wbm(wm2reg, wr, wdo, wbData);

endmodule

**Datapath (DataPath.v):**

`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////

// Company:

// Engineer:

//

// Create Date: 2021/07/04 10:25:31

// Design Name:

// Module Name: DataPath

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

//////////////////////////////////////////////////////////////////////////////////

module DataPath(

input clk,

//temp

// output WB\_wreg,

// output [4:0] WB\_destReg,

// output [31:0] WB\_wbData

// output wwreg,

// output wm2reg,

// output [4:0] wdestReg,

// output [31:0] wr,

// output [31:0] wdo

//Just for schematics

output Out\_wreg,

output [4:0] Out\_destReg,

output [31:0] Out\_wbData

);

//IF to ID

wire [31:0] inst;

// ID to Exe

wire [31:0] qa;

wire [31:0] qb;

wire [31:0] imm32;

wire wreg;

wire m2reg;

wire wmem;

wire aluimm;

wire [3:0] aluc;

wire [4:0] destReg;

//Exe to Mem

wire mwreg;

wire mm2reg;

wire mwmem;

wire [4:0] mdestReg;

wire [31:0] mr, mqb;

//Mem to WB

wire wwreg;

wire wm2reg1;

wire [4:0] wdestReg;

wire [31:0] wr;

wire [31:0] wdo;

//WB

wire WB\_wreg;

wire [4:0] WB\_destReg;

wire [31:0] WB\_wbData;

InstFetch if1(clk, inst);

InstDecode id(inst, WB\_wreg, WB\_destReg, WB\_wbData, clk, wreg, m2reg, wmem, aluc, aluimm, destReg, qa, qb, imm32);

Exe ex(wreg, m2reg, wmem, aluc, aluimm, destReg, qa, qb, imm32, clk, mwreg, mm2reg, mwmem, mdestReg, mr, mqb);

Mem me(mwreg, mm2reg, mwmem, mdestReg, mr, mqb, clk, wwreg, wm2reg1, wdestReg, wr, wdo);

WB w(wwreg, wm2reg1, wdestReg, wr, wdo, WB\_wreg, WB\_destReg, WB\_wbData);

//Just for schematics

assign Out\_wreg = WB\_wreg;

assign Out\_destReg = WB\_destReg;

assign Out\_wbData = WB\_wbData;

endmodule

**Test Bench (TB.v):**

`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////

// Company:

// Engineer:

//

// Create Date: 2021/07/04 16:12:05

// Design Name:

// Module Name: TestBench

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

//////////////////////////////////////////////////////////////////////////////////

module TestBench();

reg clk;

// wire TB\_wreg;

// wire TB\_m2reg;

// wire [4:0] TB\_wdestReg;

// wire [31:0] TB\_wr;

// wire [31:0] TB\_wdo;

DataPath dp (clk);

//, TB\_wreg, TB\_m2reg, TB\_wdestReg, TB\_wr, TB\_wdo

initial begin

clk = 0;

#100 $finish;

end

always begin

#5;

clk = ~clk;

end

endmodule

Note: In the Original code, the Datapath doesn’t have output, so TB is based on the output-less code. The Datapath with output above is just for generating schematics, other pictures/code are based on the original code.