Final Project

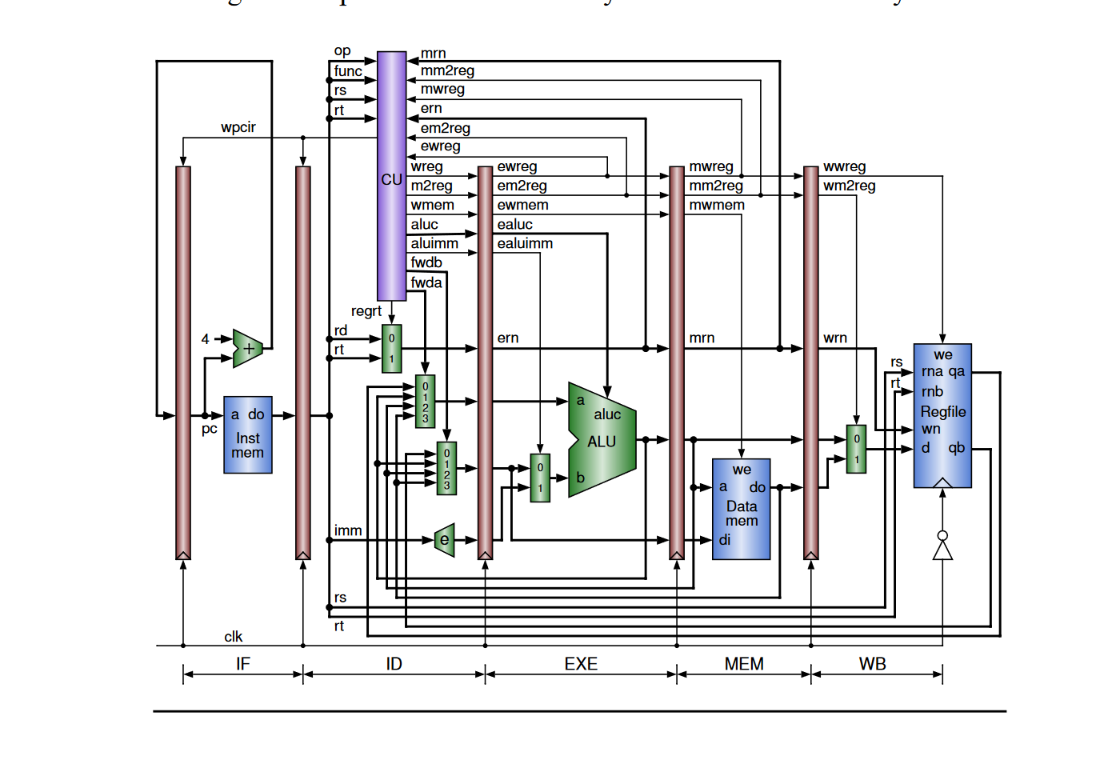
Computer Engineering 331

John Selnekovic

**Abstract:**

This project builds a pipelined cpu for the mips instruction set. We implement modules to handle each part the running an instruction. We do this by writing a stage to fetch the instructions, determine what the instructions do and fetch the data, execute the instruction then store the data. We pipe line it so that multiple instructions can happen at the same time so that we get increases throughput and implement forwarding to save us from errors that cause memory to be fetched in the next instruction before the previous ones update the value.

**Introduction:**



Each red bar represents a short-term data storage register for a clock cycle. In the IF stage we gather the instruction and then store it in the IF/ID register. In the ID stage we perform a number of things. The control unit determines what the alu does off of the func and op codes. It also determines what register to look in based on the instruction given. Also depending on the memory spaces used in previous instructions it determines if forwarding is necessary and tells Mux’s to pass the correct data through so that we do not access data before the new data is written to that spot. In the EXE stage the alu takes OpCode from the control unit and performs a calculation on the given data based on that opcode. This performs the action that the programmer wants. The MEM stage stores data when the control unit determines that it is necessary or it reads data from memory. The WB stage has data the finishing stage of the instruction and sends it back so memory hazards can be avoided.

At a high level this project takes us from the understanding of code we developed in prior years. When you learn to program you learn line by line execution. This works because engineers have developed CPU’s like these that hide the pipeline from us. If we did not use a pipeline we would have much slower computers because it lets multiple instructions be processed at once and handles memory errors for us making normal programmers jobs much easier.

**TestBench:**

`timescale 1ns / 1ps

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

// Company:

// Engineer:

//

// Create Date: 10/15/2020 04:47:46 PM

// 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\_tb;

wire [31:0] pc\_debug\_out, pc\_PCreg\_out\_cpu, instruction\_IFIDreg\_out\_cpu;

//idexe reg out

wire ewreg\_IDEXEreg\_out\_cpu, ememtoreg\_IDEXEreg\_out\_cpu, ewmem\_IDEXEreg\_out\_cpu;

wire [1:0] fwda,fwdb;

wire [3:0] ealuc\_IDEXEreg\_out\_cpu;

wire ealuimm\_IDEXEreg\_out\_cpu;

wire [5:0] rdrt\_IDEXEreg\_out\_cpu;

wire [31:0] qa\_IDEXEreg\_out\_cpu, qb\_IDEXEreg\_out\_cpu, sext\_IDEXEreg\_out\_cpu;

//exemem reg outs

wire mwreg\_EXEMEMreg\_out\_cpu, mmtoreg\_EXEMEMreg\_out\_cpu, mwmem\_EXEMEMreg\_out\_cpu;

wire [5:0] rdrt\_EXEMEMreg\_out\_cpu;

wire [31:0] alu\_EXEMEMreg\_out\_cpu, qb\_EXEMEMreg\_out\_cpu;

//memwb reg out

wire wwreg\_MEMWBreg\_out\_cpu, wmtoreg\_MEMWBreg\_out\_cpu;

wire [5:0] rdrt\_MEMWBreg\_out\_cpu;

wire [31:0] alu\_MEMWBreg\_out\_cpu, do\_MEMWBreg\_out\_cpu;

CPU cpu\_tb( clk\_tb,

pc\_debug\_out, pc\_PCreg\_out\_cpu, instruction\_IFIDreg\_out\_cpu,fwdb, fwda, ewreg\_IDEXEreg\_out\_cpu, ememtoreg\_IDEXEreg\_out\_cpu, ewmem\_IDEXEreg\_out\_cpu,

ealuc\_IDEXEreg\_out\_cpu, ealuimm\_IDEXEreg\_out\_cpu, rdrt\_IDEXEreg\_out\_cpu, qa\_IDEXEreg\_out\_cpu, qb\_IDEXEreg\_out\_cpu, sext\_IDEXEreg\_out\_cpu,

mwreg\_EXEMEMreg\_out\_cpu, mmtoreg\_EXEMEMreg\_out\_cpu, mwmem\_EXEMEMreg\_out\_cpu, rdrt\_EXEMEMreg\_out\_cpu, alu\_EXEMEMreg\_out\_cpu, qb\_EXEMEMreg\_out\_cpu,

wwreg\_MEMWBreg\_out\_cpu, wmtoreg\_MEMWBreg\_out\_cpu, rdrt\_MEMWBreg\_out\_cpu, alu\_MEMWBreg\_out\_cpu, do\_MEMWBreg\_out\_cpu );

initial

begin

clk\_tb = 0;

end

always begin

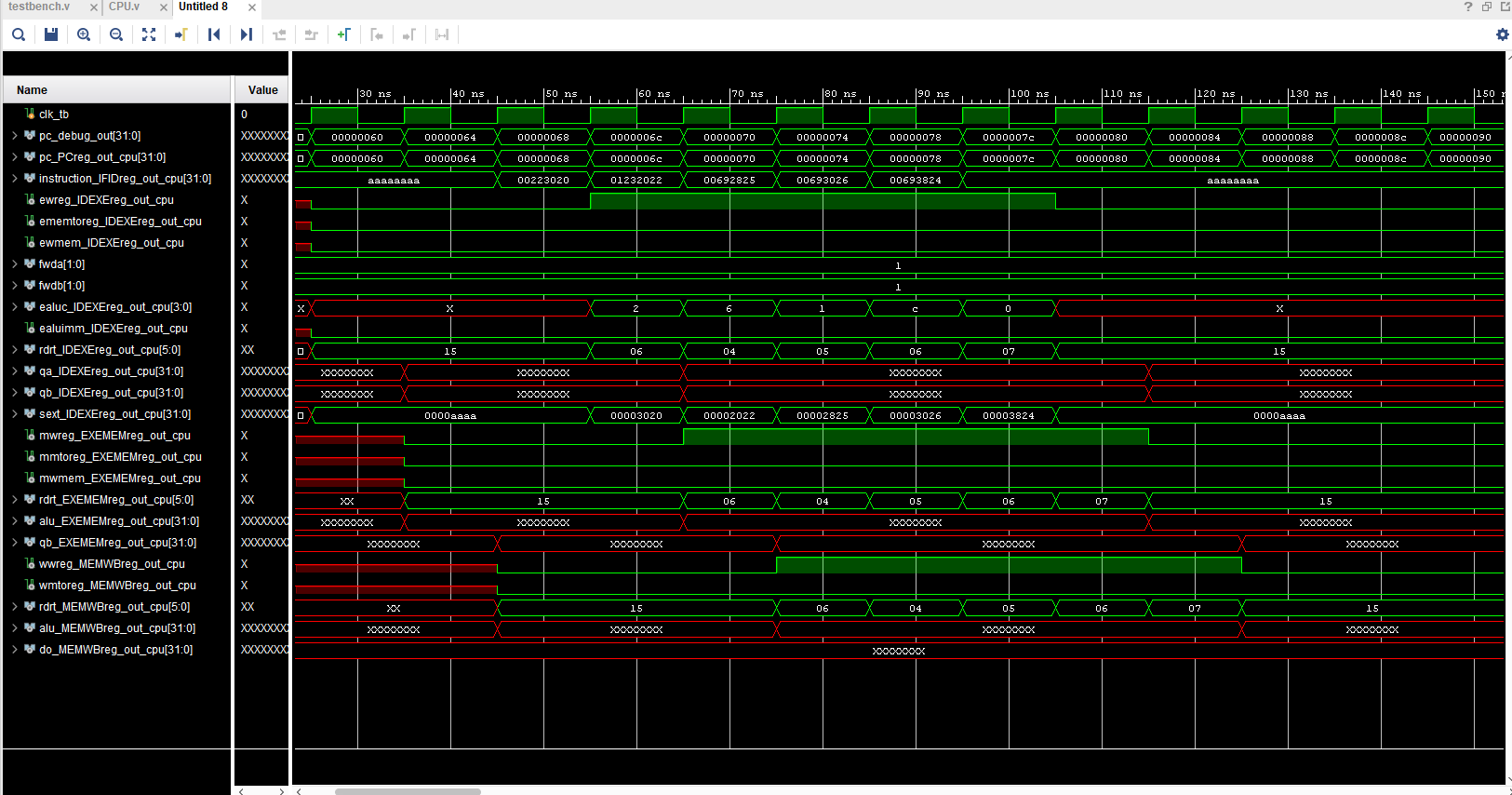
#5;

clk\_tb = ~clk\_tb;

//regrt\_tb = ~regrt\_tb;

end

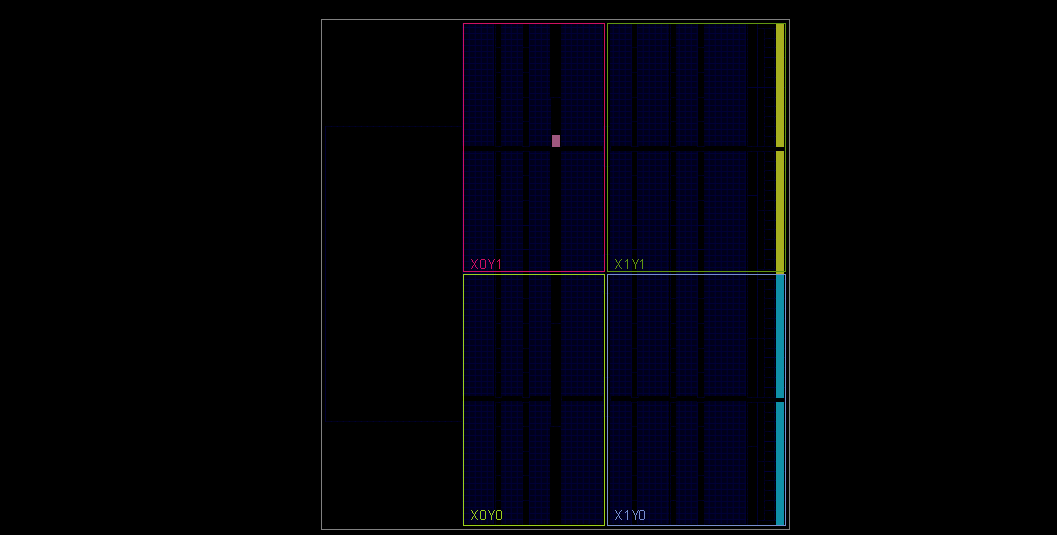
endmodule

**WaveForms:**

**I/O Planning:**



**Floor Planning:**



**Schematic:**

