# Pipelined Processor Design

04/25/07

Luke Harvey (50%) and Stephanie Spielbauer (50%)
Department of Electrical and Computer Engineering
Iowa State University

CprE 305
Dr. Chris Chu
Lab TA: Ray Souvik

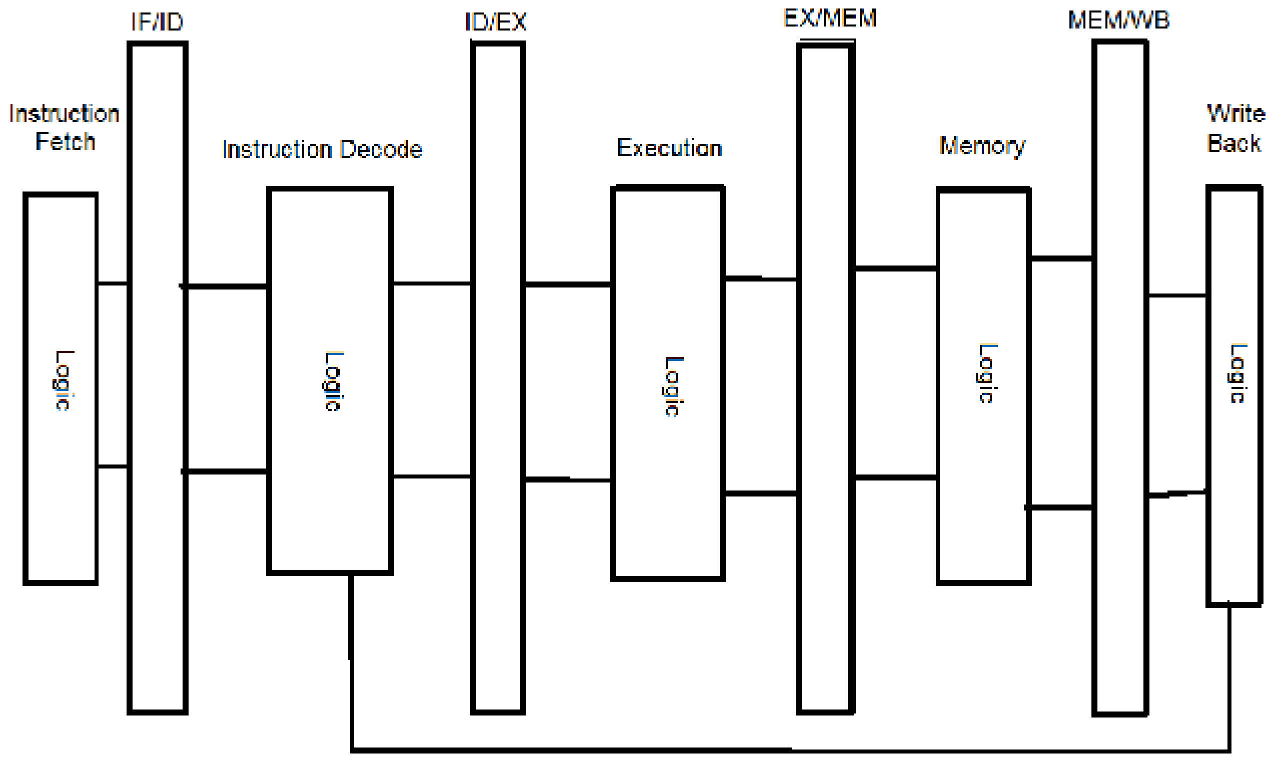
**Keywords:** Central Processing Unit(CPU), Pipelined, Pipeline Registers, Instruction Fetch Stage, Instruction Decode Stage, Execution Stage, Memory Stage, Write Back Stage, Verilog, Test Bench, Fibonacci, MIPS Instruction Set, Final Project, Design Methods, Module Design

**Abstract:** A final project for CprE 305 at Iowa State University is presented. The project's purpose was to design and simulate a MIPS(Microprocessor without Interlocked Pipeline Stages) pipelined Central Processing Unit(CPU) using the verilog hardware description language. The objectives of the project consisted of furthering our understanding of pipelining and processor design and to further understand the MIPS instruction set. Discussions and results of the projects are included in this report.

**Purpose of the machine**

The machine was designed with the MIPS instruction set in mind. Its purpose was to be able to take a binary representation of a MIPS instruction and convert it into microcode. The microcode controls the actions of the processor and pipeline. Pipelining is a process of multitasking instructions in the same data path. An instruction can be loaded into the data path at every clock cycle, even though each instruction takes up to 5 cycles to complete. At every clock cycle, each instruction is stored into a different stage in the pipeline. Doing this does not affect any of the other instructions in the processor because they are in different stages. Each stage contains a different part of the data path and has a specific function. This allows the user of the processor to fully utilize all components of the data path simultaneously, causing an increase in the throughput of their program.

There are 5 stages in our design. These are Instruction Fetch(IF), Instruction Decode(ID), Execution(EX), Memory(MEM), and Write Back(WB). These stages are separated by special registers called pipeline registers. The purpose of these registers is to separate the stages of the instructions so that there is no conflicting data due to multiple instructions being executed simultaneously. They are named after the stages that they are placed in-between: IF/ID Register, ID/EX Register, EX/MEM Register, and MEM/WB Register.



In the instruction fetch stage the instruction is loaded from memory. The hardware decides upon the address of the instruction in memory depending on the control signals from the other various stages. This instruction is written into the IF/ID register at the next positive clock cycle. Writing to this register causes the next stage(ID) to begin its operations on the instruction.

The instruction decode stage processes the instruction that was fetched in the IF stage. From the instruction control signals are determined by using the first 6 bits of the instruction, referred to as the OpCode. These control signals are divided up into their corresponding stages that they control and are passed along by the pipeline registers with the other data. Part of the instruction is the address of the registers to be used. These register addresses are placed into our register file and the values of those registers are output into the ID/EX register.

The execution stage performs the operations on the data that is passed in from the ID/EX register. It contains the Arithmetic Logic Unit(ALU). The ALU is the calculator of the data path. It performs the actual mathematical operation such as adding and subtracting. The result of the ALU operation is stored into the EX/MEM Register along with the control for the MEM and WB stages forwarded from the ID/EX Register.

The memory stage's purpose is to read from and write to from the data memory. The control signals passed in from the EX/MEM register determine which, if either, of the operations to do. The output of the memory. Is written into the MEM/WB register along with the WB control that is passed from the EX/MEM register.

The write back stage is responsible for taking the writing the data that we have just computed or loaded from memory out of the EX/MEM register and writing it to a one of the registers in the register file.

Also, since there are going to be multiple instructions in the pipeline at any given time, a data hazard detection unit and a forwarding unit were added to avoid data dependencies within the pipeline. The forwarding unit takes a value from the stage ahead of it and uses it for its source value if the instruction ahead of it is writing to a register that the current instruction uses. This allows the CPU to perform faster because it doesn't have to wait around for the instruction to finish before it begins. The hazard detection unit is used for the MIPS branch and load word instructions to stall the pipeline in the event that the next instruction needs its result.

## Instruction Set Definition

| Name | Discription | Operation in Verilog | Type | Opcode | Funct |
|---|---|---|---|---|---|
| add | Add | R[rd] = R[rs] + R[rt] | R | $0_{hex}$ | $20_{hex}$ |
| addi | Add Immediate | R[rt] = R[rs] + SignExtImm | I | $8_{hex}$ | - |
| and | And | R[rd] = R[rs] & R[rt] | R | $0_{hex}$ | $24_{hex}$ |
| beq | Branch on equal to | if(R[rs]==R[rt]) PC=PC +4+BranchAddr | I | $4_{hex}$ | - |
| bne | Branch on not equal | if(R[rs]!=R[rt]) PC=PC +4+BranchAddr | I | $5_{hex}$ | - |
| div | Divide | R[rd] = R[rs] / R[rt] | R | $0_{hex}$ | $1A_{hex}$ |
| j | jump | PC = JumpAddr | J | $2_{hex}$ | - |
| lw | Load Word | R[rt] = M[R[rs] +SignExtImm] | I | $23_{hex}$ | - |
| mult | Multiply | R[rd] = R[rs] * R[rt] | R | $0_{hex}$ | $18_{hex}$ |
| nor | Nor | R[rd] = ~(R[rs] \| R[rt]) | R | $0_{hex}$ | $27_{hex}$ |
| or | Or | R[rd] = R[rs] \| R[rt] | R | $0_{hex}$ | $25_{hex}$ |
| ori | Or Immediate | R[rt] = R[rs] \| SignExtImm | I | $D_{hex}$ | - |
| slt | Set Less Than | R[rd] = (R[rs]<R[rt]) ? 1:0 | R | $0_{hex}$ | $2A_{hex}$ |
| sub | Subtract | R[rd] = R[rs] - R[rt] | R | $0_{hex}$ | $22_{hex}$ |
| sw | Store Word | M[R[rs] +SignExtImm] = R[rt] | I | $2B_{hex}$ | - |

## Instruction format

### R-type

| | | | | | |
|---|---|---|---|---|---|
| opcode | Rs | Rt | Rd | shamnt | funct |
| 31 26 | 25 21 | 20 16 | 15 11 | 10 6 | 5 0 |

### I-type

| | | | |
|---|---|---|---|
| opcode | Rs | Rt | immedate value |
| 31 26 | 25 21 | 20 16 | 15 0 |

### J-type

| | |
|---|---|
| opcode | address |
| 31 26 | 25 0 |

## Design methodology

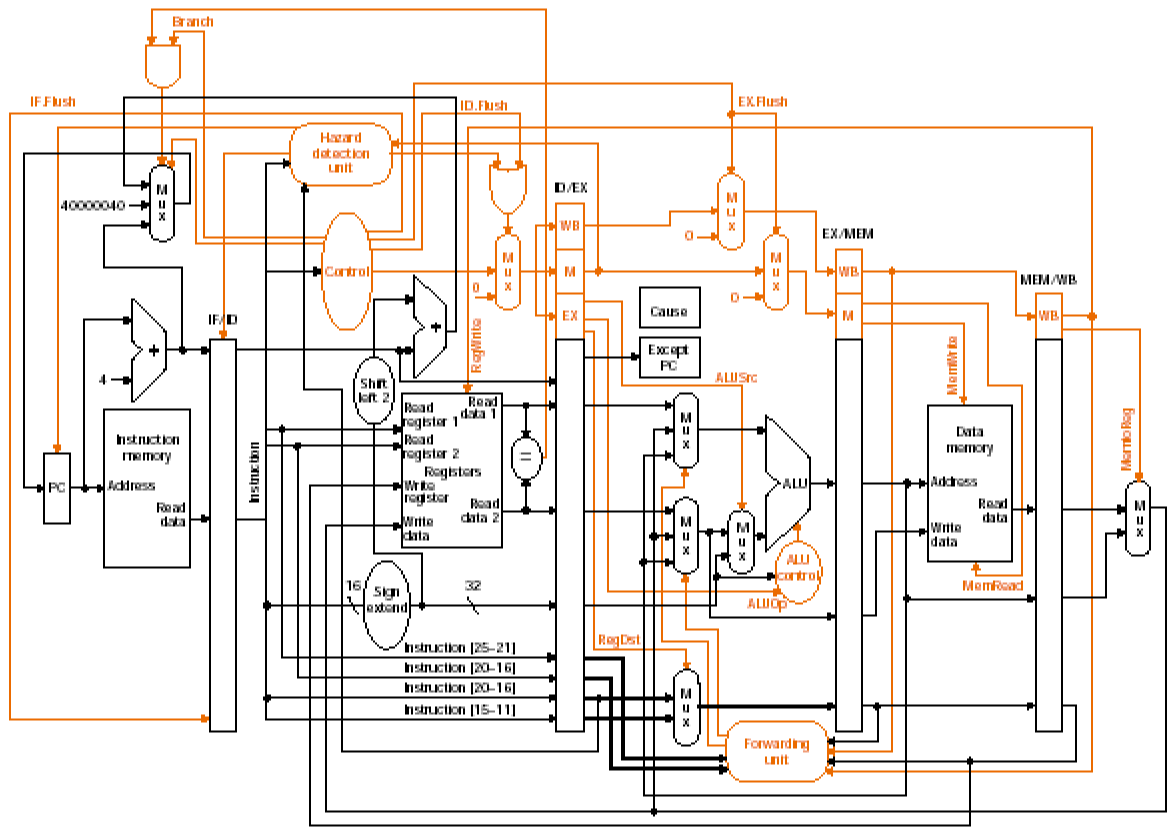
The planning of the project was broken down into 5 steps. Each step was completed before going on to the next step. The testing steps are described in testing methodology below. The steps were done in order as follows:

1. Create modules that are the components of the pipelined CPU
2. Test individual modules for functionality
3. Piece all of the individual modules together
4. Test complete CPU with Fibonacci sequence program
5. Add support and test for one instruction at a time
6. Create program to demonstrate functionality of CPU

Our pipeline implementation of the CPU was subdivided into modules which were created before being pieced together. Fortunately for us, we had created many of the basic modules before in lab while creating a single-cycle and multi-cycle CPUs. The single-cycle CPU was the most similar design to our pipeline schematic. For this reason we decided to begin with the basic single cycle design and then add in the pipeline registers and make changes to that data path. Each specialized pipeline register had to be created. Also created were the hazard detection unit and the forwarding unit. The control was also completely different from both the single-cycle and multi-cycle design so a new control was added. Test values were then put thorough the modules to test their functionality.

Once all of the modules were completed, we created the final CPU module that to joined all of the individual modules together with logic. This was the most challenging step, because of the changes in bit widths and extra control wires that had to be added. In hardware and also in verilog, all of the logic operates simultaneously. This means that we had to make sure that our external clock, which our test bench provided, had to be controlling the correct modules at the correct time. All of the pipeline registers were clocked so that they would write on the positive edge of the clock and the register file itself wrote on the positive edge and negative edge. The final data path picture is shown below. Once all of the modules were put together and fully debugged we ran the Fibonacci sequence test bench provided to us in lab to test and debug our working CPU.

The Fibonacci sequence only used 5 instructions so we still needed to implement another 10. All of the r-type instructions were the easiest to implement and we did those first because it just required adding an operation to the ALU and a change in signals to the control modules. No extra logic was added for these beyond that. The branch not equal instruction was added next because it simply had to work just like a branch on equal instruction but just the opposite. The immediate instructions were added last and were the most difficult to implement. They required a lot of extra logic and control signals because they used an immediate value though multiplexer to get them value into the ALU instead of a register from the register file. All of the instructions were tested individually as they were added and the Fibonacci sequence was tested every so often to make sure that adding support for an instruction did not cause any problems for the other instructions.



**Datapath for Pipelined Processor**

**Testing methodology**:

We used two test benches to test our design. The first was the Fibonacci sequence that we used to test our single-cycle and multi-cycle implementations for earlier labs. This program, however, did not use all of the instructions that we had to implement.

The actual test bench we created didn't do anything useful. The test bench loaded some values from memory and used them to test all of our instructions. Some of these instructions were dependent upon previous instructions, and some weren't. Each instruction got stored into a different register so that we could just run the whole program and look at the end result to decide if we ended up with the correct values.

We also tested our lab informally as we added new instructions. The first instructions we implemented partially came from the earlier labs about single-cycle and multi-cycle. We used the Fibonacci sequence program to test these first instructions. After we got those working, we used a really simple test bench program that basically just took one instruction and made sure we got the correct answer. The final test bench with all the instructions in it made sure that we could run them together to make sure that the hazards were being handled correctly.

## Conclusion

This project helped us understand how a pipeline processor works. It showed us the importance of dividing the instruction into stages and showed us how CPU time can be saved using this approach as opposed to a single-cycle or a multi-cycle implementation.

This was a difficult project, mostly because of its size. There were so many different modules that we needed to use together. There were also a lot of different wires we needed to keep track of. The timing was also a pretty big issue. We needed to make sure that everything changed at the same time. Deciding which of the components needed to be clocked and which ones didn't was a pretty major part of this project.

This was an interesting project, but we could have used more time to work on it. A suggestion would be to not have the last lab be quite so involved (maybe not implement both the single-cycle and the multi-cycle), so that we had more time to dedicate to this lab.

The final code was compiled and simulated in Quartus II. The RTL view of the logic and modules is below.

## Lessons Learned

We learned that the pipeline processor is the most difficult type to implement. This was because our processor had to be able to simultaneously work on five different instructions at the same time. It needed to decide if there were data hazards or whether or not it predicted a branch correctly. We needed it to be able to flush an instruction if it wasn't the correct one (because it predicted the branch incorrectly) and be able to stall an instruction if the data for the new instruction was dependent on data that hadn't been calculated yet. We also needed it to be able to forward data from instructions that hadn't been completed yet, so that the pipeline could proceed as soon as possible.

Having a picture to look at while coding with all the components and wires labeled was really useful(see diagram of data path above). Since there were so many different things to keep track of, having the wires labeled on the diagram would have helped save time looking through our modules to figure out the name. We had a diagram that kept track of most of this, but not all. Also, we didn't have a diagram that implemented all of the instructions we needed to, so we had to go through and add things to get the extra instructions to work.

Also, little mistakes can take a long time to debug. Most of the time we spent on this project was actually debugging, rather than writing the code. Logical errors are the most costly errors to make because only debugging can discover them. To avoid logical errors we had to make sure that we knew exactly how the pipelined processor worked causing us to learn a lot about pipelined processors while working on this lab.

## Appendix A: Verilog Code & Test-bench.

### Top level CPU Module:

```
module cpu(clock);
    input clock;

    //debugging vars
    reg [31:0] cycle;

    //IF vars
    wire [31:0] nextpc,IFpc_plus_4,IFinst;
    reg [31:0] pc;

    //ID vars
    wire PCSrc;
    wire [4:0] IDRegRs,IDRegRt,IDRegRd;
    wire [31:0] IDpc_plus_4,IDinst;
    wire [31:0] IDRegAout, IDRegBout;
    wire [31:0] IDimm_value,BranchAddr,PCMuxOut,JumpTarget;

    //control vars in ID stage
    wire PCWrite,IFIDWrite,HazMuxCon,jump,bne,imm,andi,ori,addi;
    wire [8:0] IDcontrol,ConOut;

    //EX vars
    wire [1:0] EXWB,ForwardA,ForwardB,aluop;
    wire [2:0] EXM;
    wire [3:0] EXEX,ALUCon;
    wire [4:0] EXRegRs,EXRegRt,EXRegRd,regtopass;
    wire [31:0] EXRegAout,EXRegBout,EXimm_value, b_value;
    wire [31:0] EXALUOut,ALUSrcA,ALUSrcB;

    //MEM vars
    wire [1:0] MEMWB;
    wire [2:0] MEMM;
    wire [4:0] MEMRegRd;
    wire [31:0] MEMALUOut,MEMWriteData,MEMReadData;

    //WB vars
    wire [1:0] WBWB;
    wire [4:0] WBRegRd;
    wire [31:0] datatowrite,WBReadData,WBALUOut;


    //initial conditions
    initial begin
       pc = 0;
       cycle = 0;
    end

    //debugging variable
    always@(posedge clock)
    begin
       cycle = cycle + 1;
    end

    /**
     * Instruction Fetch (IF)
     */
    assign PCSrc = ((IDRegAout==IDRegBout)&IDcontrol[6])|((IDRegAout!=IDRegBout)&bne);
    assign IFFlush = PCSrc|jump;
    assign IFpc_plus_4 = pc + 4;

    assign nextpc = PCSrc ? BranchAddr : PCMuxOut;
    always @ (posedge clock) begin
       if(PCWrite)
       begin
          pc = nextpc; //update pc
          $display("PC: %d",pc);
       end
       else
          $display("Skipped writting to PC - nop"); //nop dont update
    end

    InstructMem IM(pc,IFinst);

    IFID IFIDreg(IFFlush,clock,IFIDWrite,IFpc_plus_4,IFinst,IDinst,IDpc_plus_4);
    /**
     * Instruction Decode (ID)
     */
        assign IDRegRs[4:0]=IDinst[25:21];
        assign IDRegRt[4:0]=IDinst[20:16];
        assign IDRegRd[4:0]=IDinst[15:11];
        assign IDimm_value =
{IDinst[15],IDinst[15],IDinst[15],IDinst[15],IDinst[15],IDinst[15],IDinst[15],IDinst[15],IDi
nst[15],IDinst[15],IDinst[15],IDinst[15],IDinst[15],IDinst[15],IDinst[15],IDinst[15],IDinst[
15:0]};
        assign BranchAddr = (IDimm_value << 2) + IDpc_plus_4;
        assign JumpTarget[31:28] = IFpc_plus_4[31:28];
        assign JumpTarget[27:2] = IDinst[25:0];
        assign JumpTarget[1:0] = 0;

        assign IDcontrol = HazMuxCon ? ConOut : 0;
    assign PCMuxOut = jump ? JumpTarget : IFpc_plus_4;

        HazardUnit HU(IDRegRs,IDRegRt,EXRegRt,EXM[1],PCWrite,IFIDWrite,HazMuxCon);
        Control thecontrol(IDinst[31:26],ConOut,jump,bne,imm,andi,ori,addi);
    Registers
piperegs(clock,WBWB[0],datatowrite,WBRegRd,IDRegRs,IDRegRt,IDRegAout,IDRegBout);

    IDEX
IDEXreg(clock,IDcontrol[8:7],IDcontrol[6:4],IDcontrol[3:0],IDRegAout,IDRegBout,IDimm_value,I
DRegRs,IDRegRt,IDRegRd,EXWB,EXM,EXEX,EXRegAout,EXRegBout,EXimm_value,EXRegRs,EXRegRt,EXRegRd
);
    /**
     * Execution (EX)
     */
    assign regtopass = EXEX[3] ? EXRegRd : EXRegRt;
    assign b_value = EXEX[2] ? EXimm_value : EXRegBout;

    BIGMUX2 MUX0(ForwardA,EXRegAout,datatowrite,MEMALUOut,0,ALUSrcA);
    BIGMUX2 MUX1(ForwardB,b_value,datatowrite,MEMALUOut,0,ALUSrcB);
    ForwardUnit FU(MEMRegRd,WBRegRd,EXRegRs, EXRegRt, MEMWB[0], WBWB[0], ForwardA,
ForwardB);
    // ALU control
        assign aluop[0] =
(~IDinst[31]&~IDinst[30]&~IDinst[29]&IDinst[28]&~IDinst[27]&~IDinst[26])|(imm);
        assign aluop[1] =
(~IDinst[31]&~IDinst[30]&~IDinst[29]&~IDinst[28]&~IDinst[27]&~IDinst[26])|(imm);
    ALUControl ALUcontrol(andi,ori,addi,EXEX[1:0],EXimm_value[5:0],ALUCon);
    ALU theALU(ALUCon,ALUSrcA,ALUSrcB,EXALUOut);

    EXMEM
EXMEMreg(clock,EXWB,EXM,EXALUOut,regtopass,EXRegBout,MEMM,MEMWB,MEMALUOut,MEMRegRd,MEMWriteD
ata);
    /**
     * Memory (Mem)
     */
    DATAMEM DM(MEMM[0],MEMM[1],MEMALUOut,MEMWriteData,MEMReadData);
    MEMWB
MEMWBreg(clock,MEMWB,MEMReadData,MEMALUOut,MEMRegRd,WBWB,WBReadData,WBALUOut,WBRegRd);
    /**
     * Write Back (WB)
     */
    assign datatowrite = WBWB[1] ? WBReadData : WBALUOut;


endmodule
```

**<u>Pipeline Registers Modules:</u>**
**<u>IF/ID:</u>**

```
module IFID(flush,clock,IFIDWrite,PC_Plus4,Inst,InstReg,PC_Plus4Reg);
    input [31:0] PC_Plus4,Inst;
    input clock,IFIDWrite,flush;
    output [31:0] InstReg, PC_Plus4Reg;

    reg [31:0] InstReg, PC_Plus4Reg;

    initial begin
        InstReg = 0;
        PC_Plus4Reg = 0;
    end

    always@(posedge clock)
    begin
        if(flush)
        begin
           InstReg <= 0;
           PC_Plus4Reg <=0;
        end
        else if(IFIDWrite)
        begin
           InstReg <= Inst;
           PC_Plus4Reg <= PC_Plus4;
        end
    end

endmodule
```

**<u>ID/EX:</u>**

```
module IDEX(clock,WB,M,EX,DataA,DataB,imm_value,RegRs,RegRt,RegRd,WBreg,Mreg,EXreg,DataAreg,
DataBreg,imm_valuereg,RegRsreg,RegRtreg,RegRdreg);
    input clock;
    input [1:0] WB;
    input [2:0] M;
    input [3:0] EX;
    input [4:0] RegRs,RegRt,RegRd;
    input [31:0] DataA,DataB,imm_value;
    output [1:0] WBreg;
    output [2:0] Mreg;
    output [3:0] EXreg;
    output [4:0] RegRsreg,RegRtreg,RegRdreg;
    output [31:0] DataAreg,DataBreg,imm_valuereg;


    reg [1:0] WBreg;
    reg [2:0] Mreg;
    reg [3:0] EXreg;
    reg [31:0] DataAreg,DataBreg,imm_valuereg;
    reg [4:0] RegRsreg,RegRtreg,RegRdreg;

    initial begin
        WBreg = 0;
        Mreg = 0;
        EXreg = 0;
        DataAreg = 0;
        DataBreg = 0;
        imm_valuereg = 0;
        RegRsreg = 0;
        RegRtreg = 0;
        RegRdreg = 0;
    end

    always@(posedge clock)
    begin
        WBreg <= WB;
        Mreg <= M;
        EXreg <= EX;
        DataAreg <= DataA;
        DataBreg <= DataB;
        imm_valuereg <= imm_value;
        RegRsreg <= RegRs;
        RegRtreg <= RegRt;
        RegRdreg <= RegRd;
    end
```

endmodule

## EX/MEM:

```
module EXMEM(clock,WB,M,ALUOut,RegRD,WriteDataIn,Mreg,WBreg,ALUreg,RegRDreg,WriteDataOut);
   input clock;
   input [1:0] WB;
   input [2:0] M;
   input [4:0] RegRD;
   input [31:0] ALUOut,WriteDataIn;
   output [1:0] WBreg;
   output [2:0] Mreg;
   output [31:0] ALUreg,WriteDataOut;
   output [4:0] RegRDreg;

   reg [1:0] WBreg;
   reg [2:0] Mreg;
   reg [31:0] ALUreg,WriteDataOut;
   reg [4:0] RegRDreg;

   initial begin
      WBreg=0;
      Mreg=0;
      ALUreg=0;
      WriteDataOut=0;
      RegRDreg=0;
   end


    always@(posedge clock)
    begin
        WBreg <= WB;
        Mreg <= M;
        ALUreg <= ALUOut;
        RegRDreg <= RegRD;
        WriteDataOut <= WriteDataIn;
    end
```

endmodule

## MEM/WB:

```
module MEMWB(clock,WB,Memout,ALUOut,RegRD,WBreg,Memreg,ALUreg,RegRDreg);
   input clock;
   input [1:0] WB;
   input [4:0] RegRD;
   input [31:0] Memout,ALUOut;
   output [1:0] WBreg;
   output [31:0] Memreg,ALUreg;
   output [4:0] RegRDreg;

   reg [1:0] WBreg;
   reg [31:0] Memreg,ALUreg;
   reg [4:0] RegRDreg;

   initial begin
      WBreg = 0;
      Memreg = 0;
      ALUreg = 0;
      RegRDreg = 0;

   end

    always@(posedge clock)
    begin
        WBreg <= WB;
        Memreg <= Memout;
        ALUreg <= ALUOut;
        RegRDreg <= RegRD;
    end

endmodule
```

## Instruction Memory Module:

```
module InstructMem(PC,Inst);
   input [31:0] PC;
   output [31:0] Inst;

   reg [31:0] regfile[511:0];//32 32-bit register

   assign Inst = regfile[PC]; //assigns output to instruction

endmodule
```

## Register File Module:

```
module Registers(clock,WE,InData,WrReg,ReadA,ReadB,OutA,OutB);
   input [4:0] WrReg, ReadA, ReadB;
   input WE,clock;
   input [31:0] InData;
   output [31:0] OutA,OutB;

   reg [31:0] OutA, OutB;//2 32-bit output reg
   reg [31:0] regfile[31:0];//32 32-bit registers

   initial begin
      OutA = -20572; //random values for initial
      OutB = -398567;
   end

   always@(clock,InData,WrReg,WE)
   begin
    if(WE && clock)
      begin
       regfile[WrReg]<=InData;//write to register
       $display("Does WrReg: %d Data: %d",WrReg,InData);
      end
   end

   always @ (clock,ReadA,ReadB,WrReg)
   begin
      if(~clock)
      begin
    OutA <= regfile[ReadA];//read values from registers
    OutB <= regfile[ReadB];
    $monitor ("R3: %d R4: %d R5 %d R6: %d R7: %d R8 %d R9: %d R10: %d R11 %d R12: %d R13: %d R14 %d",regfile[3],regfile[4],regfile[5],regfile[6],regfile[7],regfile[8],regfile[9],regfile[10],regfile[11],regfile[12],regfile[13],regfile[14]);
    end
   end


endmodule
```

## ALU Module:

```
module ALU(ALUCon,DataA,DataB,Result);
    input [3:0] ALUCon;
    input [31:0] DataA,DataB;
    output [31:0] Result;

    reg [31:0] Result;
    reg Zero;

    initial begin
        Result = 32'd0;
    end

    always@(ALUCon,DataA,DataB)
    begin
      case(ALUCon)
          4'b0000://and
             Result <= DataA&DataB;

          4'b0001://or
             Result <= DataA|DataB;

          4'b0010://add
              Result <= DataA+DataB;

          4'b0011://multiply
              Result <= DataA*DataB;
          4'b0100://nor
              begin
                 Result[0] <= !(DataA[0]|DataB[0]);
                 Result[1] <= !(DataA[1]|DataB[1]);
                 Result[2] <= !(DataA[2]|DataB[2]);
                 Result[3] <= !(DataA[3]|DataB[3]);
                 Result[4] <= !(DataA[4]|DataB[4]);
                 Result[5] <= !(DataA[5]|DataB[5]);
                 Result[6] <= !(DataA[6]|DataB[6]);
                 Result[7] <= !(DataA[7]|DataB[7]);
                 Result[8] <= !(DataA[8]|DataB[8]);
                 Result[9] <= !(DataA[9]|DataB[9]);
                 Result[10] <= !(DataA[10]|DataB[10]);
                 Result[11] <= !(DataA[11]|DataB[11]);
                 Result[12] <= !(DataA[12]|DataB[12]);
                 Result[13] <= !(DataA[13]|DataB[13]);
                 Result[14] <= !(DataA[14]|DataB[14]);
                 Result[15] <= !(DataA[15]|DataB[15]);
                 Result[16] <= !(DataA[16]|DataB[16]);
                 Result[17] <= !(DataA[17]|DataB[17]);
                 Result[18] <= !(DataA[18]|DataB[18]);
                 Result[19] <= !(DataA[19]|DataB[19]);
                 Result[20] <= !(DataA[20]|DataB[20]);
                 Result[21] <= !(DataA[21]|DataB[21]);
                 Result[22] <= !(DataA[22]|DataB[22]);
                 Result[23] <= !(DataA[23]|DataB[23]);
                 Result[24] <= !(DataA[24]|DataB[24]);
                 Result[25] <= !(DataA[25]|DataB[25]);
                 Result[26] <= !(DataA[26]|DataB[26]);
                 Result[27] <= !(DataA[27]|DataB[27]);
                 Result[28] <= !(DataA[28]|DataB[28]);
                 Result[29] <= !(DataA[29]|DataB[29]);
                 Result[30] <= !(DataA[30]|DataB[30]);
                 Result[31] <= !(DataA[31]|DataB[31]);
              end

          4'b0101://divide
             Result <= DataA/DataB;

          4'b0110://sub
             Result <= DataA-DataB;

          4'b0111://slt
             Result = DataA<DataB ? 1:0;

          4'b1000://sll
             Result <= (DataA<<DataB);

          4'b0110://srl
             Result <= (DataA>>DataB);

          default: //error
          begin
              $display("ALUERROR");
              Result = 0;
          end

      endcase
    end

endmodule
```

## Data Memory Module:

```
module DATAMEM(MemWrite,MemRead,Addr,Wdata,Rdata);
   input [31:0] Addr,Wdata;
   input MemWrite,MemRead;
   output [31:0] Rdata;

   reg [31:0] Rdata;
   reg [31:0] regfile[511:0];//32 32-bit registers


   always@(Addr,Wdata,MemWrite,MemRead)
   if(MemWrite)
   begin
     $display("Writing %d -> Addr: %d",Wdata,Addr);
     regfile[Addr]<=Wdata; //memory write
   end

   always@(Addr,Wdata,MemWrite,MemRead)
   if(MemRead)
     Rdata <= regfile[Addr];//memory read

endmodule
```

## ALU Control Module:

```
module ALUControl(andi,ori,addi,ALUOp,funct,ALUCon);
    input [1:0] ALUOp;
    input [5:0] funct;
    input andi,ori,addi;
    output [3:0] ALUCon;

    reg [3:0] ALUCon;

    always@(ALUOp or funct or andi or ori or addi)
    begin
      case(ALUOp)
        2'b00://lw or sw
           ALUCon = 4'b0010;

        2'b01://beq
           ALUCon = 4'b0110;

        2'b10://R-type
        begin
           if(funct==6'b100100)
              ALUCon = 4'b0000;//and
           if(funct==6'b100101)
              ALUCon = 4'b0001;//or
           if(funct==6'b100000)
              ALUCon = 4'b0010;//add
           if(funct==6'b011000)
              ALUCon = 4'b0011;//multi
           if(funct==6'b100111)
              ALUCon = 4'b0100;//nor
           if(funct==6'b011010)
              ALUCon = 4'b0101;//div
           if(funct==6'b100010)
              ALUCon = 4'b0110;//sub
           if(funct==6'b101010)
              ALUCon = 4'b0111;//slt
        end
      2'b11://immediate
      begin
          if(andi)begin
             ALUCon = 4'b0000;//andi
         end
          if(ori) begin
             ALUCon = 4'b0001;//ori
         end
          if(addi)
             ALUCon = 4'b0010;//addi
      end
    endcase
    end


endmodule
```

## Control Module:

```
module Control(Op,Out,j,bne,imm,andi,ori,addi);
   input [5:0] Op;
   output[8:0] Out;
   output j,bne,imm,andi,ori,addi;

   wire regdst,alusrc,memtoreg,regwrite,memread,memwrite,branch;

   //determines type of instruction
   wire r = ~Op[5]&~Op[4]&~Op[3]&~Op[2]&~Op[1]&~Op[0];
		wire lw = Op[5]&~Op[4]&~Op[3]&~Op[2]&Op[1]&Op[0];
		wire sw = Op[5]&~Op[4]&Op[3]&~Op[2]&Op[1]&Op[0];
		wire beq = ~Op[5]&~Op[4]&~Op[3]&Op[2]&~Op[1]&~Op[0];
		wire bne = ~Op[5]&~Op[4]&~Op[3]&Op[2]&~Op[1]&Op[0];
		wire j = ~Op[5]&~Op[4]&~Op[3]&~Op[2]&Op[1]&~Op[0];
   wire andi = ~Op[5]&~Op[4]&Op[3]&Op[2]&~Op[1]&~Op[0];
   wire ori = ~Op[5]&~Op[4]&Op[3]&Op[2]&~Op[1]&Op[0];
   wire addi = ~Op[5]&~Op[4]&Op[3]&~Op[2]&~Op[1]&~Op[0];
   wire imm = andi|ori|addi; //immediate value type

   //seperate control arrays for reference
   wire [3:0] EXE;
   wire [2:0] M;
   wire [1:0] WB;

		// microcode control
		assign regdst = r;
		assign alusrc = lw|sw|imm;
		assign memtoreg = lw;
		assign regwrite = r|lw|imm;
		assign memread = lw;
		assign memwrite = sw;
		assign branch = beq;

		// EXE control
		assign EXE[3] = regdst;
		assign EXE[2] = alusrc;
		assign EXE[1] = r;
		assign EXE[0] = beq;

		//M control
		assign M[2] = branch;
		assign M[1] = memread;
		assign M[0] = memwrite;

		//WB control
		assign WB[1] = memtoreg; //not same as diagram
		assign WB[0] = regwrite;

		//output control
		assign Out[8:7] = WB;
		assign Out[6:4] = M;
		assign Out[3:0] = EXE;

endmodule
```

**Forwarding Unit Module:**

```
module ForwardUnit(MEMRegRd,WBRegRd,EXRegRs,EXRegRt, MEM_RegWrite, WB_RegWrite, ForwardA, ForwardB);
  input[4:0] MEMRegRd,WBRegRd,EXRegRs,EXRegRt;
  input MEM_RegWrite, WB_RegWrite;
  output[1:0] ForwardA, ForwardB;

  reg[1:0] ForwardA, ForwardB;

  //Forward A
  always@(MEM_RegWrite or MEMRegRd or EXRegRs or WB_RegWrite or WBRegRd)
  begin
    if((MEM_RegWrite)&&(MEMRegRd != 0)&&(MEMRegRd == EXRegRs))
      ForwardA = 2'b10;
    else if((WB_RegWrite)&&(WBRegRd != 0)&&(WBRegRd == EXRegRs)&&(MEMRegRd != EXRegRs) )
      ForwardA = 2'b01;
    else
      ForwardA = 2'b00;
  end

  //Forward B
  always@(WB_RegWrite or WBRegRd or EXRegRt or MEMRegRd or MEM_RegWrite)
  begin
    if((WB_RegWrite)&&(WBRegRd != 0)&&(WBRegRd == EXRegRt)&&(MEMRegRd != EXRegRt) )
      ForwardB = 2'b01;
    else if((MEM_RegWrite)&&(MEMRegRd != 0)&&(MEMRegRd == EXRegRt))
      ForwardB = 2'b10;
    else
      ForwardB = 2'b00;
  end

endmodulemodule ForwardUnit(MEMRegRd,WBRegRd,EXRegRs,EXRegRt, MEM_RegWrite, WB_RegWrite, ForwardA, ForwardB);
```

```
  input[4:0] MEMRegRd,WBRegRd,EXRegRs,EXRegRt;
  input MEM_RegWrite, WB_RegWrite;
  output[1:0] ForwardA, ForwardB;

  reg[1:0] ForwardA, ForwardB;

  //Forward A
  always@(MEM_RegWrite or MEMRegRd or EXRegRs or WB_RegWrite or WBRegRd)
  begin
    if((MEM_RegWrite)&&(MEMRegRd != 0)&&(MEMRegRd == EXRegRs))
      ForwardA = 2'b10;
    else if((WB_RegWrite)&&(WBRegRd != 0)&&(WBRegRd == EXRegRs)&&(MEMRegRd != EXRegRs) )
      ForwardA = 2'b01;
    else
      ForwardA = 2'b00;
  end

  //Forward B
  always@(WB_RegWrite or WBRegRd or EXRegRt or MEMRegRd or MEM_RegWrite)
  begin
    if((WB_RegWrite)&&(WBRegRd != 0)&&(WBRegRd == EXRegRt)&&(MEMRegRd != EXRegRt) )
      ForwardB = 2'b01;
    else if((MEM_RegWrite)&&(MEMRegRd != 0)&&(MEMRegRd == EXRegRt))
      ForwardB = 2'b10;
    else
      ForwardB = 2'b00;
  end

endmodule
```

## Hazard Detection Unit Module:

```
module HazardUnit(IDRegRs,IDRegRt,EXRegRt,EXMemRead,PCWrite,IFIDWrite,HazMuxCon);
    input [4:0] IDRegRs,IDRegRt,EXRegRt;
    input EXMemRead;
    output PCWrite, IFIDWrite, HazMuxCon;

    reg PCWrite, IFIDWrite, HazMuxCon;

    always@(IDRegRs,IDRegRt,EXRegRt,EXMemRead)
    if(EXMemRead&((EXRegRt == IDRegRs)|(EXRegRt == IDRegRt)))
      begin//stall
         PCWrite = 0;
         IFIDWrite = 0;
         HazMuxCon = 1;
      end
    else
      begin//no stall
         PCWrite = 1;
         IFIDWrite = 1;
         HazMuxCon = 1;

      end

endmodule
```

## Multiplexer Module:

```
module BIGMUX2(A,X0,X1,X2,X3,Out);//non-clocked mux
input [1:0] A;
input [31:0] X3,X2,X1,X0;
output [31:0] Out;

reg [31:0] Out;

always@(A,X3,X2,X1,X0)
begin
  case(A)
      2'b00:
        Out <= X0;
      2'b01:
        Out <= X1;
      2'b10:
        Out <= X2;
      2'b11:
        Out <= X3;
  endcase
end

endmodule
```

## Test Bench A - Fibonacci:

```
module Pipelined_TestBench;

  reg Clock;
  integer i;

  initial begin
     Clock = 1;
  end
  //clock controls
  always begin
                   Clock = ~Clock;
                   #25;
        end

  initial begin

        // Instr Memory intialization
        pipelined.IM.regfile[0] = 32'h8c030000;
        pipelined.IM.regfile[4] = 32'h8c040001;
        pipelined.IM.regfile[8] = 32'h8c050002;
        pipelined.IM.regfile[12] = 32'h8c010002;
        pipelined.IM.regfile[16] = 32'h10600004;
        pipelined.IM.regfile[20] = 32'h00852020;
        pipelined.IM.regfile[24] = 32'h00852822;
        pipelined.IM.regfile[28] = 32'h00611820;
        pipelined.IM.regfile[32] = 32'h1000fffb;
        pipelined.IM.regfile[36] = 32'hac040006;

        // Data Memory intialization
        pipelined.DM.regfile[0] = 32'd8;
        pipelined.DM.regfile[1] = 32'd1;
        pipelined.DM.regfile[2] = -32'd1;
        pipelined.DM.regfile[3] = 0;

        pipelined.piperegs.regfile[0] = 0;

        // Register File initialization
        for (i = 0; i < 32; i = i + 1)
                pipelined.piperegs.regfile[i] = 32'd0;


 end
 //Instantiate cpu
  cpu pipelined(Clock);

endmodule
```

## Test Bench B – Program that tests all 15 instructions (does not do anything useful):

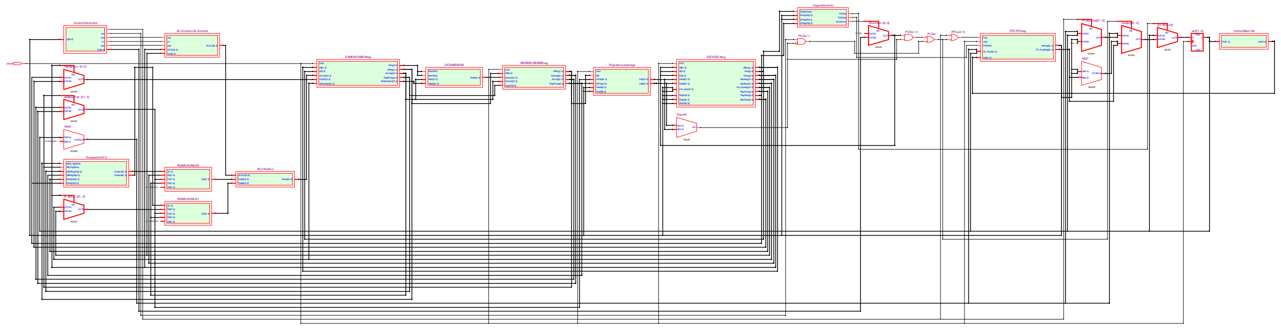
```
module Pipelined_TestBench;

  reg Clock;
  integer i;

  initial begin
     Clock = 1;
  end
  //clock controls
  always begin
                        Clock = ~Clock;
                        #25;
           end

  initial begin

           // Instr Memory intialization
           pipelined.IM.regfile[0] = 32'h8C030000; //lw R3,0(R1)
           pipelined.IM.regfile[4] = 32'h8C040001;//lw R4,1(R0)
           pipelined.IM.regfile[8] = 32'h00642820;//add R5,R3,R4
           pipelined.IM.regfile[12] = 32'h00A43022;//sub R6,R5,R4
           pipelined.IM.regfile[16] = 32'h00643824;//and R7,R3,R4
           pipelined.IM.regfile[20] = 32'h00644025;//or R8,R3,R4
           pipelined.IM.regfile[24] = 32'h00644827;//nor R9,R3,R4
           pipelined.IM.regfile[28] = 32'h00C5502A;//slt R10,R6,R5
           pipelined.IM.regfile[32] = 32'h80000008;//j startloop
           pipelined.IM.regfile[36] = 32'h2063FFFF;//loop: addi R3,R3,-1
           pipelined.IM.regfile[40] = 32'h14E3FFFE;//startloop: bne R3,R7,-2
           pipelined.IM.regfile[44] = 32'h01295818;//mult R11,R9,R9
           pipelined.IM.regfile[48] = 32'h0166601A;//div R12,R11,R6
           pipelined.IM.regfile[52] = 32'h34CE0002;//ori R14,R6,2
           pipelined.IM.regfile[56] = 32'h11CC0000;//beq R14,R12, next
           pipelined.IM.regfile[60] = 32'hADCE0006;//sw

           // Data Memory intialization
           pipelined.DM.regfile[0] = 32'd8;
           pipelined.DM.regfile[1] = 32'd1;

           pipelined.piperegs.regfile[0] = 0;

           // Register File initialization
           for (i = 0; i < 32; i = i + 1)
                    pipelined.piperegs.regfile[i] = 32'd0;


 end
 //Instantiate cpu
  cpu pipelined(Clock);

endmodule
```

**Appendix B**: Simulation results.
**Test Bench A Output – Fibonacci :**

```
# R3:        0 R4:        0 R5        0 R6:        0 R7:        0 R8        0 R9:        0 R10:        0 R11        0 R12:        0 R13:        0 R14        0
# PC:        4
# R3:        0 R4:        0 R5        0 R6:        0 R7:        0 R8        0 R9:        0 R10:        0 R11        0 R12:        0 R13:        0 R14        0
# PC:        8
# R3:        0 R4:        0 R5        0 R6:        0 R7:        0 R8        0 R9:        0 R10:        0 R11        0 R12:        0 R13:        0 R14        0
# PC:       12
# Does WrReg:  0 Data:         0
# R3:        0 R4:        0 R5        0 R6:        0 R7:        0 R8        0 R9:        0 R10:        0 R11        0 R12:        0 R13:        0 R14        0
# PC:       16
# Does WrReg:  0 Data:         0
# Does WrReg:  3 Data:         0
# Does WrReg:  3 Data:         8
# R3:        8 R4:        0 R5        0 R6:        0 R7:        0 R8        0 R9:        0 R10:        0 R11        0 R12:        0 R13:        0 R14        0
# R3:        8 R4:        0 R5        0 R6:        0 R7:        0 R8        0 R9:        0 R10:        0 R11        0 R12:        0 R13:        0 R14        0
# PC:       20
# Does WrReg:  3 Data:         8
# Does WrReg:  4 Data:         8
# Does WrReg:  4 Data:         1
# R3:        8 R4:        1 R5        0 R6:        0 R7:        0 R8        0 R9:        0 R10:        0 R11        0 R12:        0 R13:        0 R14        0
# R3:        8 R4:        1 R5        0 R6:        0 R7:        0 R8        0 R9:        0 R10:        0 R11        0 R12:        0 R13:        0 R14        0
# PC:       24
# Does WrReg:  4 Data:         1
# Does WrReg:  5 Data:         1
# Does WrReg:  5 Data: 4294967295
# R3:        8 R4:        1 R5 4294967295 R6:        0 R7:        0 R8        0 R9:        0 R10:        0 R11        0 R12:        0 R13:        0 R14        0
# R3:        8 R4:        1 R5 4294967295 R6:        0 R7:        0 R8        0 R9:        0 R10:        0 R11        0 R12:        0 R13:        0 R14        0
# PC:       28
# Does WrReg:  5 Data: 4294967295
# Does WrReg:  1 Data: 4294967295
# R3:        8 R4:        1 R5 4294967295 R6:        0 R7:        0 R8        0 R9:        0 R10:        0 R11        0 R12:        0 R13:        0 R14        0
# PC:       32
# Does WrReg:  1 Data: 4294967295
# R3:        8 R4:        1 R5 4294967295 R6:        0 R7:        0 R8        0 R9:        0 R10:        0 R11        0 R12:        0 R13:        0 R14        0
# PC:       36
# Does WrReg:  4 Data:         8
# Does WrReg:  4 Data:         0
# R3:        8 R4:        0 R5 4294967295 R6:        0 R7:        0 R8        0 R9:        0 R10:        0 R11        0 R12:        0 R13:        0 R14        0
# R3:        8 R4:        0 R5 4294967295 R6:        0 R7:        0 R8        0 R9:        0 R10:        0 R11        0 R12:        0 R13:        0 R14        0
# PC:       16
# Does WrReg:  4 Data:         0
# Does WrReg:  5 Data:         0
# Does WrReg:  5 Data:         1
# R3:        8 R4:        0 R5        1 R6:        0 R7:        0 R8        0 R9:        0 R10:        0 R11        0 R12:        0 R13:        0 R14        0
# R3:        8 R4:        0 R5        1 R6:        0 R7:        0 R8        0 R9:        0 R10:        0 R11        0 R12:        0 R13:        0 R14        0
# PC:       20
# Does WrReg:  5 Data:         1
# Does WrReg:  3 Data:         1
# Does WrReg:  3 Data:         7
# R3:        7 R4:        0 R5        1 R6:        0 R7:        0 R8        0 R9:        0 R10:        0 R11        0 R12:        0 R13:        0 R14        0
# R3:        7 R4:        0 R5        1 R6:        0 R7:        0 R8        0 R9:        0 R10:        0 R11        0 R12:        0 R13:        0 R14        0
# PC:       24
# Does WrReg:  3 Data:         7
# R3:        7 R4:        0 R5        1 R6:        0 R7:        0 R8        0 R9:        0 R10:        0 R11        0 R12:        0 R13:        0 R14        0
# PC:       28
# Does WrReg:  0 Data:         0
# R3:        7 R4:        0 R5        1 R6:        0 R7:        0 R8        0 R9:        0 R10:        0 R11        0 R12:        0 R13:        0 R14        0
# PC:       32
# Does WrReg:  0 Data:         0
# R3:        7 R4:        0 R5        1 R6:        0 R7:        0 R8        0 R9:        0 R10:        0 R11        0 R12:        0 R13:        0 R14        0
# PC:       36
# Does WrReg:  4 Data:         7
# Does WrReg:  4 Data:         1
# R3:        7 R4:        1 R5        1 R6:        0 R7:        0 R8        0 R9:        0 R10:        0 R11        0 R12:        0 R13:        0 R14        0
# R3:        7 R4:        1 R5        1 R6:        0 R7:        0 R8        0 R9:        0 R10:        0 R11        0 R12:        0 R13:        0 R14        0
# PC:       16
# Does WrReg:  4 Data:         1
# Does WrReg:  5 Data:         1
# Does WrReg:  5 Data:         0
# R3:          7 R4:         1 R5         0 R6:         0 R7:         0 R8         0 R9:         0 R10:         0 R11         0 R12:         0 R13:         0 R14         0
# R3:          7 R4:         1 R5         0 R6:         0 R7:         0 R8         0 R9:         0 R10:         0 R11         0 R12:         0 R13:         0 R14         0
# PC:         20
# Does WrReg:  5 Data:         0
# Does WrReg:  3 Data:         0
# Does WrReg:  3 Data:         6
# R3:          6 R4:         1 R5         0 R6:         0 R7:         0 R8         0 R9:         0 R10:         0 R11         0 R12:         0 R13:         0 R14         0
# R3:          6 R4:         1 R5         0 R6:         0 R7:         0 R8         0 R9:         0 R10:         0 R11         0 R12:         0 R13:         0 R14         0
# PC:         24
# Does WrReg:  3 Data:         6
# R3:          6 R4:         1 R5         0 R6:         0 R7:         0 R8         0 R9:         0 R10:         0 R11         0 R12:         0 R13:         0 R14         0
# PC:         28
# Does WrReg:  0 Data:         0
# R3:          6 R4:         1 R5         0 R6:         0 R7:         0 R8         0 R9:         0 R10:         0 R11         0 R12:         0 R13:         0 R14         0
# PC:         32
# Does WrReg:  0 Data:         0
# R3:          6 R4:         1 R5         0 R6:         0 R7:         0 R8         0 R9:         0 R10:         0 R11         0 R12:         0 R13:         0 R14         0
# PC:         36
# Does WrReg:  4 Data:         6
# Does WrReg:  4 Data:         1
# R3:          6 R4:         1 R5         0 R6:         0 R7:         0 R8         0 R9:         0 R10:         0 R11         0 R12:         0 R13:         0 R14         0
# PC:         16
# Does WrReg:  4 Data:         1
# Does WrReg:  5 Data:         1
# R3:          6 R4:         1 R5         1 R6:         0 R7:         0 R8         0 R9:         0 R10:         0 R11         0 R12:         0 R13:         0 R14         0
# R3:          6 R4:         1 R5         1 R6:         0 R7:         0 R8         0 R9:         0 R10:         0 R11         0 R12:         0 R13:         0 R14         0
# PC:         20
# Does WrReg:  5 Data:         1
# Does WrReg:  3 Data:         1
# Does WrReg:  3 Data:         5
# R3:          5 R4:         1 R5         1 R6:         0 R7:         0 R8         0 R9:         0 R10:         0 R11         0 R12:         0 R13:         0 R14         0
# R3:          5 R4:         1 R5         1 R6:         0 R7:         0 R8         0 R9:         0 R10:         0 R11         0 R12:         0 R13:         0 R14         0
# PC:         24
# Does WrReg:  3 Data:         5
# R3:          5 R4:         1 R5         1 R6:         0 R7:         0 R8         0 R9:         0 R10:         0 R11         0 R12:         0 R13:         0 R14         0
# PC:         28
# Does WrReg:  0 Data:         0
# R3:          5 R4:         1 R5         1 R6:         0 R7:         0 R8         0 R9:         0 R10:         0 R11         0 R12:         0 R13:         0 R14         0
# PC:         32
# Does WrReg:  0 Data:         0
# R3:          5 R4:         1 R5         1 R6:         0 R7:         0 R8         0 R9:         0 R10:         0 R11         0 R12:         0 R13:         0 R14         0
# PC:         36
# Does WrReg:  4 Data:         5
# Does WrReg:  4 Data:         2
# R3:          5 R4:         2 R5         1 R6:         0 R7:         0 R8         0 R9:         0 R10:         0 R11         0 R12:         0 R13:         0 R14         0
# R3:          5 R4:         2 R5         1 R6:         0 R7:         0 R8         0 R9:         0 R10:         0 R11         0 R12:         0 R13:         0 R14         0
# PC:         16
# Does WrReg:  4 Data:         2
# Does WrReg:  5 Data:         2
# Does WrReg:  5 Data:         1
# R3:          5 R4:         2 R5         1 R6:         0 R7:         0 R8         0 R9:         0 R10:         0 R11         0 R12:         0 R13:         0 R14         0
# PC:         20
# Does WrReg:  5 Data:         1
# Does WrReg:  3 Data:         1
# Does WrReg:  3 Data:         4
# R3:          4 R4:         2 R5         1 R6:         0 R7:         0 R8         0 R9:         0 R10:         0 R11         0 R12:         0 R13:         0 R14         0
# R3:          4 R4:         2 R5         1 R6:         0 R7:         0 R8         0 R9:         0 R10:         0 R11         0 R12:         0 R13:         0 R14         0
# PC:         24
# Does WrReg:  3 Data:         4
# R3:          4 R4:         2 R5         1 R6:         0 R7:         0 R8         0 R9:         0 R10:         0 R11         0 R12:         0 R13:         0 R14         0
# PC:         28
# Does WrReg:  0 Data:         0
# R3:          4 R4:         2 R5         1 R6:         0 R7:         0 R8         0 R9:         0 R10:         0 R11         0 R12:         0 R13:         0 R14         0
# PC:         32
# Does WrReg:  0 Data:         0
# R3:          4 R4:         2 R5         1 R6:         0 R7:         0 R8         0 R9:         0 R10:         0 R11         0 R12:         0 R13:         0 R14         0
# PC:         36
# Does WrReg:  4 Data:         4
# Does WrReg:  4 Data:         3
# R3:          4 R4:         3 R5         1 R6:         0 R7:         0 R8         0 R9:         0 R10:         0 R11         0 R12:         0 R13:         0 R14         0
# R3:          4 R4:         3 R5         1 R6:         0 R7:         0 R8         0 R9:         0 R10:         0 R11         0 R12:         0 R13:         0 R14         0
# PC:         16
# Does WrReg:  4 Data:         3
# Does WrReg:  5 Data:         3
# Does WrReg:  5 Data:         2
# R3:          4 R4:         3 R5         2 R6:         0 R7:         0 R8         0 R9:         0 R10:          0 R11         0 R12:          0 R13:          0 R14         0
# R3:          4 R4:         3 R5         2 R6:         0 R7:         0 R8         0 R9:         0 R10:          0 R11         0 R12:          0 R13:          0 R14         0
# PC:         20
# Does WrReg:  5 Data:         2
# Does WrReg:  3 Data:         2
# Does WrReg:  3 Data:         3
# R3:          3 R4:         3 R5         2 R6:         0 R7:         0 R8         0 R9:         0 R10:          0 R11         0 R12:          0 R13:          0 R14         0
# R3:          3 R4:         3 R5         2 R6:         0 R7:         0 R8         0 R9:         0 R10:          0 R11         0 R12:          0 R13:          0 R14         0
# PC:         24
# Does WrReg:  3 Data:         3
# R3:          3 R4:         3 R5         2 R6:         0 R7:         0 R8         0 R9:         0 R10:          0 R11         0 R12:          0 R13:          0 R14         0
# PC:         28
# Does WrReg:  0 Data:         0
# R3:          3 R4:         3 R5         2 R6:         0 R7:         0 R8         0 R9:         0 R10:          0 R11         0 R12:          0 R13:          0 R14         0
# PC:         32
# Does WrReg:  0 Data:         0
# R3:          3 R4:         3 R5         2 R6:         0 R7:         0 R8         0 R9:         0 R10:          0 R11         0 R12:          0 R13:          0 R14         0
# PC:         36
# Does WrReg:  4 Data:         3
# Does WrReg:  4 Data:         5
# R3:          3 R4:         5 R5         2 R6:         0 R7:         0 R8         0 R9:         0 R10:          0 R11         0 R12:          0 R13:          0 R14         0
# R3:          3 R4:         5 R5         2 R6:         0 R7:         0 R8         0 R9:         0 R10:          0 R11         0 R12:          0 R13:          0 R14         0
# PC:         16
# Does WrReg:  4 Data:         5
# Does WrReg:  5 Data:         5
# Does WrReg:  5 Data:         3
# R3:          3 R4:         5 R5         3 R6:         0 R7:         0 R8         0 R9:         0 R10:          0 R11         0 R12:          0 R13:          0 R14         0
# R3:          3 R4:         5 R5         3 R6:         0 R7:         0 R8         0 R9:         0 R10:          0 R11         0 R12:          0 R13:          0 R14         0
# PC:         20
# Does WrReg:  5 Data:         3
# Does WrReg:  3 Data:         3
# Does WrReg:  3 Data:         2
# R3:          2 R4:         5 R5         3 R6:         0 R7:         0 R8         0 R9:         0 R10:          0 R11         0 R12:          0 R13:          0 R14         0
# R3:          2 R4:         5 R5         3 R6:         0 R7:         0 R8         0 R9:         0 R10:          0 R11         0 R12:          0 R13:          0 R14         0
# PC:         24
# Does WrReg:  3 Data:         2
# R3:          2 R4:         5 R5         3 R6:         0 R7:         0 R8         0 R9:         0 R10:          0 R11         0 R12:          0 R13:          0 R14         0
# PC:         28
# Does WrReg:  0 Data:         0
# R3:          2 R4:         5 R5         3 R6:         0 R7:         0 R8         0 R9:         0 R10:          0 R11         0 R12:          0 R13:          0 R14         0
# PC:         32
# Does WrReg:  0 Data:         0
# R3:          2 R4:         5 R5         3 R6:         0 R7:         0 R8         0 R9:         0 R10:          0 R11         0 R12:          0 R13:          0 R14         0
# PC:         36
# Does WrReg:  4 Data:         2
# Does WrReg:  4 Data:         8
# R3:          2 R4:         8 R5         3 R6:         0 R7:         0 R8         0 R9:         0 R10:          0 R11         0 R12:          0 R13:          0 R14         0
# R3:          2 R4:         8 R5         3 R6:         0 R7:         0 R8         0 R9:         0 R10:          0 R11         0 R12:          0 R13:          0 R14         0
# PC:         16
# Does WrReg:  4 Data:         8
# Does WrReg:  5 Data:         8
# Does WrReg:  5 Data:         5
# R3:          2 R4:         8 R5         5 R6:         0 R7:         0 R8         0 R9:         0 R10:          0 R11         0 R12:          0 R13:          0 R14         0
# R3:          2 R4:         8 R5         5 R6:         0 R7:         0 R8         0 R9:         0 R10:          0 R11         0 R12:          0 R13:          0 R14         0
# PC:         20
# Does WrReg:  5 Data:         5
# Does WrReg:  3 Data:         5
# Does WrReg:  3 Data:         1
# R3:          1 R4:         8 R5         5 R6:         0 R7:         0 R8         0 R9:         0 R10:          0 R11         0 R12:          0 R13:          0 R14         0
# R3:          1 R4:         8 R5         5 R6:         0 R7:         0 R8         0 R9:         0 R10:          0 R11         0 R12:          0 R13:          0 R14         0
# PC:         24
# Does WrReg:  3 Data:         1
# R3:          1 R4:         8 R5         5 R6:         0 R7:         0 R8         0 R9:         0 R10:          0 R11         0 R12:          0 R13:          0 R14         0
# PC:         28
# Does WrReg:  0 Data:         0
# R3:          1 R4:         8 R5         5 R6:         0 R7:         0 R8         0 R9:         0 R10:          0 R11         0 R12:          0 R13:          0 R14         0
# PC:         32
# Does WrReg:  0 Data:         0
# R3:          1 R4:         8 R5         5 R6:         0 R7:         0 R8         0 R9:         0 R10:          0 R11         0 R12:          0 R13:          0 R14         0
# PC:         36
# Does WrReg:  4 Data:         1
# Does WrReg:  4 Data:        13
# R3:          1 R4:        13 R5          5 R6:          0 R7:          0 R8          0 R9:          0 R10:          0 R11          0 R12:          0 R13:          0 R14          0
# R3:          1 R4:        13 R5          5 R6:          0 R7:          0 R8          0 R9:          0 R10:          0 R11          0 R12:          0 R13:          0 R14          0
# PC:        16
# Does WrReg:  4 Data:        13
# Does WrReg:  5 Data:        13
# Does WrReg:  5 Data:         8
# R3:          1 R4:        13 R5          8 R6:          0 R7:          0 R8          0 R9:          0 R10:          0 R11          0 R12:          0 R13:          0 R14          0
# R3:          1 R4:        13 R5          8 R6:          0 R7:          0 R8          0 R9:          0 R10:          0 R11          0 R12:          0 R13:          0 R14          0
# PC:        20
# Does WrReg:  5 Data:         8
# Does WrReg:  3 Data:         8
# Does WrReg:  3 Data:         0
# R3:          0 R4:        13 R5          8 R6:          0 R7:          0 R8          0 R9:          0 R10:          0 R11          0 R12:          0 R13:          0 R14          0
# R3:          0 R4:        13 R5          8 R6:          0 R7:          0 R8          0 R9:          0 R10:          0 R11          0 R12:          0 R13:          0 R14          0
# PC:        36
# Does WrReg:  3 Data:         0
# R3:          0 R4:        13 R5          8 R6:          0 R7:          0 R8          0 R9:          0 R10:          0 R11          0 R12:          0 R13:          0 R14          0
# PC:        40
# Does WrReg:  0 Data:         0
# R3:          0 R4:        13 R5          8 R6:          0 R7:          0 R8          0 R9:          0 R10:          0 R11          0 R12:          0 R13:          0 R14          0
# PC:        44
# Does WrReg:  0 Data:         0
# R3:          0 R4:        13 R5          8 R6:          0 R7:          0 R8          0 R9:          0 R10:          0 R11          0 R12:          0 R13:          0 R14          0
# PC:         x
# Does WrReg:  0 Data:         0
# Writing        13 -> Addr:         6
```

## Test Bench B Output – Program that tests all 15 instructions (does not do anything useful):

```
# R3:          0 R4:         0 R5          0 R6:          0 R7:          0 R8          0 R9:          0 R10:          0 R11          0 R12:          0 R13:          0 R14          0
# PC:         4
# R3:          0 R4:         0 R5          0 R6:          0 R7:          0 R8          0 R9:          0 R10:          0 R11          0 R12:          0 R13:          0 R14          0
# PC:         8
# R3:          0 R4:         0 R5          0 R6:          0 R7:          0 R8          0 R9:          0 R10:          0 R11          0 R12:          0 R13:          0 R14          0
# PC:        12
# Does WrReg:  0 Data:         0
# R3:          0 R4:         0 R5          0 R6:          0 R7:          0 R8          0 R9:          0 R10:          0 R11          0 R12:          0 R13:          0 R14          0
# Skipped writting to PC - nop
# Does WrReg:  0 Data:         0
# Does WrReg:  3 Data:         0
# Does WrReg:  3 Data:         8
# R3:          8 R4:         0 R5          0 R6:          0 R7:          0 R8          0 R9:          0 R10:          0 R11          0 R12:          0 R13:          0 R14          0
# R3:          8 R4:         0 R5          0 R6:          0 R7:          0 R8          0 R9:          0 R10:          0 R11          0 R12:          0 R13:          0 R14          0
# PC:        16
# Does WrReg:  3 Data:         8
# Does WrReg:  4 Data:         8
# Does WrReg:  4 Data:         1
# R3:          8 R4:         1 R5          0 R6:          0 R7:          0 R8          0 R9:          0 R10:          0 R11          0 R12:          0 R13:          0 R14          0
# R3:          8 R4:         1 R5          0 R6:          0 R7:          0 R8          0 R9:          0 R10:          0 R11          0 R12:          0 R13:          0 R14          0
# PC:        20
# Does WrReg:  4 Data:         1
# Does WrReg:  5 Data:         1
# Does WrReg:  5 Data:         9
# R3:          8 R4:         1 R5          9 R6:          0 R7:          0 R8          0 R9:          0 R10:          0 R11          0 R12:          0 R13:          0 R14          0
# R3:          8 R4:         1 R5          9 R6:          0 R7:          0 R8          0 R9:          0 R10:          0 R11          0 R12:          0 R13:          0 R14          0
# PC:        24
# Does WrReg:  5 Data:         9
# R3:          8 R4:         1 R5          9 R6:          0 R7:          0 R8          0 R9:          0 R10:          0 R11          0 R12:          0 R13:          0 R14          0
# PC:        28
# Does WrReg:  5 Data:         9
# Does WrReg:  6 Data:         9
# Does WrReg:  6 Data:         8
# R3:          8 R4:         1 R5          9 R6:          8 R7:          0 R8          0 R9:          0 R10:          0 R11          0 R12:          0 R13:          0 R14          0
# R3:          8 R4:         1 R5          9 R6:          8 R7:          0 R8          0 R9:          0 R10:          0 R11          0 R12:          0 R13:          0 R14          0
# PC:        32
# Does WrReg:  6 Data:         8
# Does WrReg:  7 Data:         8
# Does WrReg:  7 Data:         0
# R3:          8 R4:          1 R5          9 R6:          8 R7:          0 R8          0 R9:          0 R10:          0 R11          0 R12:          0 R13:          0 R14          0
# PC:         36
# Does WrReg:  7 Data:          0
# Does WrReg:  8 Data:          0
# Does WrReg:  8 Data:          9
# R3:          8 R4:          1 R5          9 R6:          8 R7:          0 R8          9 R9:          0 R10:          0 R11          0 R12:          0 R13:          0 R14          0
# R3:          8 R4:          1 R5          9 R6:          8 R7:          0 R8          9 R9:          0 R10:          0 R11          0 R12:          0 R13:          0 R14          0
# PC:         40
# Does WrReg:  8 Data:          9
# Does WrReg:  9 Data:          9
# Does WrReg:  9 Data: 4294967286
# R3:          8 R4:          1 R5          9 R6:          8 R7:          0 R8          9 R9: 4294967286 R10:          0 R11          0 R12:          0 R13:          0 R14          0
# R3:          8 R4:          1 R5          9 R6:          8 R7:          0 R8          9 R9: 4294967286 R10:          0 R11          0 R12:          0 R13:          0 R14          0
# PC:         44
# Does WrReg:  9 Data: 4294967286
# Does WrReg: 10 Data: 4294967286
# Does WrReg: 10 Data:          1
# R3:          8 R4:          1 R5          9 R6:          8 R7:          0 R8          9 R9: 4294967286 R10:          1 R11          0 R12:          0 R13:          0 R14          0
# R3:          8 R4:          1 R5          9 R6:          8 R7:          0 R8          9 R9: 4294967286 R10:          1 R11          0 R12:          0 R13:          0 R14          0
# PC:         36
# Does WrReg: 10 Data:          1
# R3:          8 R4:          1 R5          9 R6:          8 R7:          0 R8          9 R9: 4294967286 R10:          1 R11          0 R12:          0 R13:          0 R14          0
# PC:         40
# Does WrReg:  3 Data:          0
# Does WrReg:  3 Data:          7
# R3:          7 R4:          1 R5          9 R6:          8 R7:          0 R8          9 R9: 4294967286 R10:          1 R11          0 R12:          0 R13:          0 R14          0
# R3:          7 R4:          1 R5          9 R6:          8 R7:          0 R8          9 R9: 4294967286 R10:          1 R11          0 R12:          0 R13:          0 R14          0
# PC:         44
# Does WrReg:  3 Data:          7
# R3:          7 R4:          1 R5          9 R6:          8 R7:          0 R8          9 R9: 4294967286 R10:          1 R11          0 R12:          0 R13:          0 R14          0
# PC:         36
# Does WrReg:  0 Data:          7
# Does WrReg:  0 Data:          0
# R3:          7 R4:          1 R5          9 R6:          8 R7:          0 R8          9 R9: 4294967286 R10:          1 R11          0 R12:          0 R13:          0 R14          0
# PC:         40
# Does WrReg:  0 Data:          0
# Does WrReg:  3 Data:          0
# Does WrReg:  3 Data:          6
# R3:          6 R4:          1 R5          9 R6:          8 R7:          0 R8          9 R9: 4294967286 R10:          1 R11          0 R12:          0 R13:          0 R14          0
# R3:          6 R4:          1 R5          9 R6:          8 R7:          0 R8          9 R9: 4294967286 R10:          1 R11          0 R12:          0 R13:          0 R14          0
# PC:         44
# Does WrReg:  3 Data:          6
# R3:          6 R4:          1 R5          9 R6:          8 R7:          0 R8          9 R9: 4294967286 R10:          1 R11          0 R12:          0 R13:          0 R14          0
# PC:         36
# Does WrReg:  0 Data:          6
# Does WrReg:  0 Data:          0
# R3:          6 R4:          1 R5          9 R6:          8 R7:          0 R8          9 R9: 4294967286 R10:          1 R11          0 R12:          0 R13:          0 R14          0
# PC:         40
# Does WrReg:  0 Data:          0
# Does WrReg:  3 Data:          0
# Does WrReg:  3 Data:          5
# R3:          5 R4:          1 R5          9 R6:          8 R7:          0 R8          9 R9: 4294967286 R10:          1 R11          0 R12:          0 R13:          0 R14          0
# R3:          5 R4:          1 R5          9 R6:          8 R7:          0 R8          9 R9: 4294967286 R10:          1 R11          0 R12:          0 R13:          0 R14          0
# PC:         44
# Does WrReg:  3 Data:          5
# R3:          5 R4:          1 R5          9 R6:          8 R7:          0 R8          9 R9: 4294967286 R10:          1 R11          0 R12:          0 R13:          0 R14          0
# PC:         36
# Does WrReg:  0 Data:          5
# Does WrReg:  0 Data:          0
# R3:          5 R4:          1 R5          9 R6:          8 R7:          0 R8          9 R9: 4294967286 R10:          1 R11          0 R12:          0 R13:          0 R14          0
# PC:         40
# Does WrReg:  0 Data:          0
# Does WrReg:  3 Data:          0
# Does WrReg:  3 Data:          4
# R3:          4 R4:          1 R5          9 R6:          8 R7:          0 R8          9 R9: 4294967286 R10:          1 R11          0 R12:          0 R13:          0 R14          0
# R3:          4 R4:          1 R5          9 R6:          8 R7:          0 R8          9 R9: 4294967286 R10:          1 R11          0 R12:          0 R13:          0 R14          0
# PC:         44
# Does WrReg:  3 Data:          4
# R3:          4 R4:          1 R5          9 R6:          8 R7:          0 R8          9 R9: 4294967286 R10:          1 R11          0 R12:          0 R13:          0 R14          0
# PC:         36
# Does WrReg:  0 Data:          4
# Does WrReg:  0 Data:          0
# R3:          4 R4:          1 R5          9 R6:          8 R7:          0 R8          9 R9: 4294967286 R10:          1 R11          0 R12:          0 R13:          0 R14          0
# PC:        40
# Does WrReg:  0 Data:         0
# Does WrReg:  3 Data:         0
# Does WrReg:  3 Data:         3
# R3:          3 R4:          1 R5          9 R6:          8 R7:          0 R8          9 R9: 4294967286 R10:          1 R11          0 R12:          0 R13:          0 R14          0
# R3:          3 R4:          1 R5          9 R6:          8 R7:          0 R8          9 R9: 4294967286 R10:          1 R11          0 R12:          0 R13:          0 R14          0
# PC:        44
# Does WrReg:  3 Data:         3
# R3:          3 R4:          1 R5          9 R6:          8 R7:          0 R8          9 R9: 4294967286 R10:          1 R11          0 R12:          0 R13:          0 R14          0
# PC:        36
# Does WrReg:  0 Data:         3
# Does WrReg:  0 Data:         0
# R3:          3 R4:          1 R5          9 R6:          8 R7:          0 R8          9 R9: 4294967286 R10:          1 R11          0 R12:          0 R13:          0 R14          0
# PC:        40
# Does WrReg:  0 Data:         0
# Does WrReg:  3 Data:         0
# Does WrReg:  3 Data:         2
# R3:          2 R4:          1 R5          9 R6:          8 R7:          0 R8          9 R9: 4294967286 R10:          1 R11          0 R12:          0 R13:          0 R14          0
# R3:          2 R4:          1 R5          9 R6:          8 R7:          0 R8          9 R9: 4294967286 R10:          1 R11          0 R12:          0 R13:          0 R14          0
# PC:        44
# Does WrReg:  3 Data:         2
# R3:          2 R4:          1 R5          9 R6:          8 R7:          0 R8          9 R9: 4294967286 R10:          1 R11          0 R12:          0 R13:          0 R14          0
# PC:        36
# Does WrReg:  0 Data:         2
# Does WrReg:  0 Data:         0
# R3:          2 R4:          1 R5          9 R6:          8 R7:          0 R8          9 R9: 4294967286 R10:          1 R11          0 R12:          0 R13:          0 R14          0
# PC:        40
# Does WrReg:  0 Data:         0
# Does WrReg:  3 Data:         0
# Does WrReg:  3 Data:         1
# R3:          1 R4:          1 R5          9 R6:          8 R7:          0 R8          9 R9: 4294967286 R10:          1 R11          0 R12:          0 R13:          0 R14          0
# R3:          1 R4:          1 R5          9 R6:          8 R7:          0 R8          9 R9: 4294967286 R10:          1 R11          0 R12:          0 R13:          0 R14          0
# PC:        44
# Does WrReg:  3 Data:         1
# R3:          1 R4:          1 R5          9 R6:          8 R7:          0 R8          9 R9: 4294967286 R10:          1 R11          0 R12:          0 R13:          0 R14          0
# PC:        36
# Does WrReg:  0 Data:         1
# Does WrReg:  0 Data:         0
# R3:          1 R4:          1 R5          9 R6:          8 R7:          0 R8          9 R9: 4294967286 R10:          1 R11          0 R12:          0 R13:          0 R14          0
# PC:        40
# Does WrReg:  0 Data:         0
# Does WrReg:  3 Data:         0
# R3:          0 R4:          1 R5          9 R6:          8 R7:          0 R8          9 R9: 4294967286 R10:          1 R11          0 R12:          0 R13:          0 R14          0
# R3:          0 R4:          1 R5          9 R6:          8 R7:          0 R8          9 R9: 4294967286 R10:          1 R11          0 R12:          0 R13:          0 R14          0
# PC:        44
# Does WrReg:  3 Data:         0
# R3:          0 R4:          1 R5          9 R6:          8 R7:          0 R8          9 R9: 4294967286 R10:          1 R11          0 R12:          0 R13:          0 R14          0
# PC:        48
# Does WrReg:  0 Data:         0
# R3:          0 R4:          1 R5          9 R6:          8 R7:          0 R8          9 R9: 4294967286 R10:          1 R11          0 R12:          0 R13:          0 R14          0
# PC:        52
# Does WrReg:  0 Data:         0
# Does WrReg:  3 Data:         0
# Does WrReg:  3 Data: 4294967295
# R3: 4294967295 R4:          1 R5          9 R6:          8 R7:          0 R8          9 R9: 4294967286 R10:          1 R11          0 R12:          0 R13:          0 R14
0
# R3: 4294967295 R4:          1 R5          9 R6:          8 R7:          0 R8          9 R9: 4294967286 R10:          1 R11          0 R12:          0 R13:          0 R14
0
# PC:        56
# Does WrReg:  3 Data: 4294967295
# R3: 4294967295 R4:          1 R5          9 R6:          8 R7:          0 R8          9 R9: 4294967286 R10:          1 R11          0 R12:          0 R13:          0 R14
0
# PC:        60
# Does WrReg: 11 Data: 4294967295
# Does WrReg: 11 Data:       100
# R3: 4294967295 R4:          1 R5          9 R6:          8 R7:          0 R8          9 R9: 4294967286 R10:          1 R11        100 R12:          0 R13:          0 R14
0
# R3: 4294967295 R4:          1 R5          9 R6:          8 R7:          0 R8          9 R9: 4294967286 R10:          1 R11        100 R12:          0 R13:          0 R14
0
# PC:        60
# Does WrReg: 11 Data:       100
# Does WrReg: 12 Data:       100
# Does WrReg: 12 Data:        12
# R3: 4294967295 R4:          1 R5          9 R6:          8 R7:          0 R8          9 R9: 4294967286 R10:          1 R11        100 R12:         12 R13:          0 R14
0
# R3: 4294967295 R4:          1 R5          9 R6:          8 R7:          0 R8          9 R9: 4294967286 R10:          1 R11        100 R12:         12 R13:          0 R14
0
# PC:         64
# Does WrReg: 12 Data:         12
# Does WrReg: 14 Data:         12
# Does WrReg: 14 Data:         10
# R3: 4294967295 R4:          1 R5          9 R6:          8 R7:          0 R8          9 R9: 4294967286 R10:          1 R11        100 R12:         12 R13:          0 R14
10
# R3: 4294967295 R4:          1 R5          9 R6:          8 R7:          0 R8          9 R9: 4294967286 R10:          1 R11        100 R12:         12 R13:          0 R14
10
# PC:         68
# Does WrReg: 14 Data:         10
# R3: 4294967295 R4:          1 R5          9 R6:          8 R7:          0 R8          9 R9: 4294967286 R10:          1 R11        100 R12:         12 R13:          0 R14
10
# PC:         72
# Does WrReg:  0 Data: 4294967294
# Writing         10 -> Addr:         16
# Does WrReg:  0 Data:          0
# R3: 4294967295 R4:          1 R5          9 R6:          8 R7:          0 R8          9 R9: 4294967286 R10:          1 R11        100 R12:         12 R13:          0 R14
10
# PC:          x
# Does WrReg:  0 Data:          0
# R3: 4294967295 R4:          1 R5          9 R6:          8 R7:          0 R8          9 R9: 4294967286 R10:          1 R11        100 R12:         12 R13:          0 R14
10
# PC:          x
# Does WrReg: 13 Data:         16
# Does WrReg: 13 Data:         10
# R3: 4294967295 R4:          1 R5          9 R6:          8 R7:          0 R8          9 R9: 4294967286 R10:          1 R11        100 R12:         12 R13:         10 R14
10
# R3: 4294967295 R4:          1 R5          9 R6:          8 R7:          0 R8          9 R9: 4294967286 R10:          1 R11        100 R12:         12 R13:         10 R14
10
# PC:          x
# Does WrReg: 13 Data:         10
```



**RTL simulation view of final datapath for our code**

**Appendix C**: Our common verilog mistakes

- Forgetting to place semicolon at end of line-

  This is our most common mistake in coding in any language. This however has an easy solution and the compiler catches the problem instantly.
- Not declaring a variable as a input/output-

  This often happends when we are coding and just place variables into modules without remembering they are inputs or outputs. The compiler also catches these mistakes.
- Incorrect bit sizes-

  When adding in extra bits for the control signals often times there were a lot of instances of that value being passed around to the various modules. It was difficult to locate each instance of the variable and grow/shrink the bit sizes.