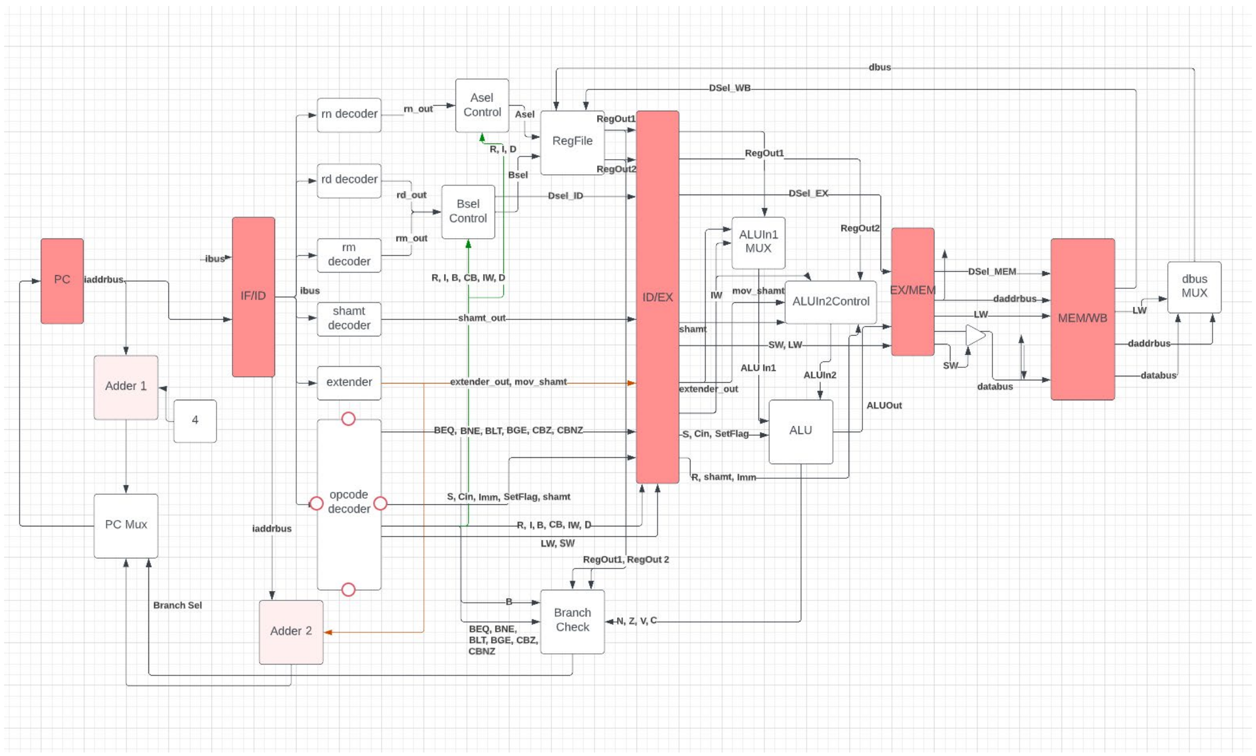
Aryan Mehrotra
Final Project
EECE3324

I. Description

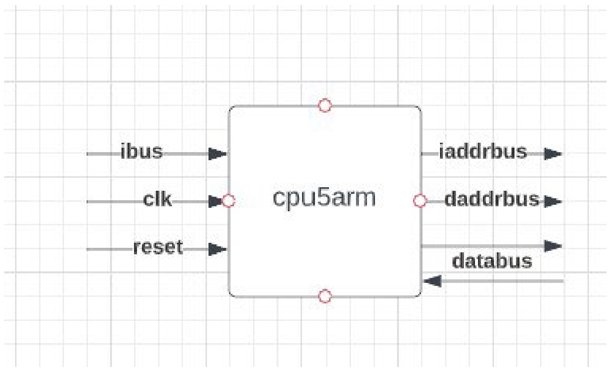
The goal of this assignment is to make a 5-stage pipelined CPU with the ARM LEGV8 instruction set. The MIPS pipelined CPU can be used as a reference for the high-level design, however the ARM LEGV8 has its own instruction reference card or green card which will be needed to be used. This assignment would provide experience in designing a CPU individually, and to test the skills acquired over the semester.

II. Schematic

a. Pipelined Connections



b. Block Diagram of Top Module



Aryan Mehrotra
Final Project
EECE3324

III. ARM LEGv8 Explanation

a. Instruction Fetch

The instruction fetch stage is responsible for pulling the ibus in and sending it through the pipeline. It's a 32-bit binary input which will be decoded in the next stage in-order to carry out a set of supported instructions. This stage also sends the iaddrbus through to the decode stage. At the start of the clock, the iaddrbus is set to 0 when the reset signal is high. In the following clock cycles, the reset is low and the iaddrbus increments using adder modules in the instruction fetch and decode stage. After being clocked through the program counter D Flip-Flop, the iaddrbus is send to an adder module which increments it by 4. The output of the adder is sent to the PC_mux. Depending on the select signal from the Branch_Check module, the mux sends correct iaddrbus out. The ibus, and the iaddrbus from the PC are sent through the D Flip-Flop connecting the fetch and decode stages. At the posedge of the clk, they are cycled through to the decode stage.

b. Instruction Decode

This stage interprets and breaks down the ibus to carry out instructions. There are several submodules to help with that. The rn_decoder module takes in the 32-bit ibus and pulls out 5-bits from it at ibus[9:5]. The green card given indicates that the identifier Rn register is present at those bits. The Rn register is only required for r-type, i-type and d-type instruction formats. To filter for that, the output of the rn_decoder is sent to the Asel_control module which only send the Rn value to the register file if the current instruction is of r, i, or d type. The module takes in control bits R, I, D and if any one of those are active it sends the output of the rn_decoder module to the register file. While it would be possible to just send the output of the rn_decoder to the RegFile without checking for instruction type, I felt that having a control signal would be helpful to reduce possible error. If the instruction format isn't one of the 3 types specified, we set Asel to all Zs.

The rm_decoder and rd_decoder pull out the rm and rd values from the ibus. The rm value is only needed for r-type instructions. The rd value is needed for all instruction formats except the b-type. The Bselect for the register file is chosen from either rm or rd. The outputs of the decoders get sent to the Bsel_control module. It takes in various control bits which signify which instructions are active. It takes in control bits R, I, B, DT, IW & CB and uses them to output the Bsel to the RegFile and the Dselect through the pipeline. For R type, it sends rm to Bsel and rd to Dsel. I & IW type have rd to Dsel and Bsel set to all Zs since they use an immediate format. D type & CB type sets Bsel and Dsel to rd, while it also uses an immediate format it needs a Bsel value set. B type instructions set Bsel to Zs, and Dsel to the zero register as no writeback is necessary.

The register file takes in three 32-bit inputs: Asel, Bsel & Dsel; one 64-bit input which is dbus, and it outputs two 64-bit values RegOut1 (abus) & RegOut2 (bbus). The Asel and Bsel values tell the register file which registers are needed to be read so that the values stored there can be sent out. The two outputs are sent through the ID/EX D Flip-Flop to the execute stage. They are also sent to the Branch_Check module which will be covered. The Dsel & dbus value are sent from the writeback stage. The Dsel value tells the file which register needs to be written to, and the dbus tells the file what value is going to be written. Reading and writing in the register file happens on the negative edge of the clock in-order to speed up the frequency of the processor. Additionally, unlike the MIPS register file, the zero register for LEGV8 is set to R31 AKA XZR. R0 is just a regular register that can be written to. Another difference is that the registers store thirty-two 64-bit values, whereas in MIPS we could store thirty-two 32-bit values.

The shamt decoder takes in the ibus and only pulls out the shift amount which are ibus[15:10]. It sends out a 6-bit output called shamt_out directly to the ID/EX flip flop where it will be used in the execute stage. I felt it would be better to have a separate module for this operation instead combining it with one of the other decoders for troubleshooting purposes mainly. It helped me rule out areas which didn’t have issues and ensure that the overlap between decoder modules would be reduced.

The extender module extends the immediate values in the ibus. It takes in control bits from the opcode decoder to specify the instruction from and sets the output depending on the type of instruction. The output of the extender also goes to the second adder to increment the iaddrbus. Different instruction types have different formats to extend the immediate values. The formats can be found in the green card provided. I, IW, D, B, CB type instructions have an immediate value in their ibus. I & IW type 0-pad the immediate to reach a 64-bit value. D, B & CB type sign extend the immediate value. B & CB type also multiply by 4 since they are branch instructions. The extender module also acts as a multiplexer of sorts as it only sends out 1 extended value depending on the instruction type. It additionally sends out a mov_shamt value for all instruction types. This was needed in-order to implement the MOVZ instruction. I couldn’t think of another way to integrate, and I didn’t want to add another module just for this as it would clutter up the decode stage which is why I’m passing the mov_shamt through this module. A better way could have been to do it through the shamt module to localize all shift amounts in one place, but I want to re-implement it as it was working.

The opcode decoder breaks down the ibus and sets control bits depending the on the operation encoded in the ibus. It uses an always block and case statements to match up opcodes, if there is a match it sends out control bits to help the other modules carry out decoding or to select certain wires. If no match is found, it sends out a generic set of control bits to signify NOP or no operation. The opcode decoder outputs the instruction type: R, I, IW, B, D, CB. It sends out control bits to signify the type of branch carried out: BEQ, BNE, CBZ, CBNZ, BLT, BGE. It also sends the select signal to the ALU, LW & SW bits for load & store operations, Cin, Imm to signify an instruction with an immediate, and the SetFlag control bit which lets the ALU know to set flags for branching. Many of the control bits are used by other modules in the decode stage, such as by Asel_control and Bsel_control. The Branch_Check module uses some to tell if a branch instruction is being carried out, and all the bits are sent through the ID/EX flip-flop to be used in the execute stage.

The branch check takes in flags from the ALU, both outputs from the register file and control bits from the opcode decoder. LEGV8 uses flags to signify if a branch is taken or not. Specific combinations are needed in order to branch, and the combinations are different for each instruction. Through the use of an always block and if-statements, the Branch_Check module outputs a select signal which is used by the PC mux to tell if the iaddrbus is going to be the branched one or not. There are 4 flags output from the ALU and passed directly to this module: N to signify a negative ALU Output, V to signify overflow in the ALU Output, C if the ALU Output has a carry out, and Z if the ALU Output is 0. The BEQ, BNE, BGE, BLT, CBZ. CBNZ all have different requirements to enable the branch. CBZ and CBNZ use the RegOut values, the other instructions use the flags to determine if a branch is needed. The branch check also sees if a B-type instruction was passed through. If that's the case, then it always branches.

In the decode stage, the iaddrbus gets sent directly to another adder module. The other input of this adder module the the extender_out wire. These two are added together and sent to the PC_mux. If the Branch_Check outputs a select signal that is high, the result of this adder is sent to iaddrbus, otherwise the result of the adder in the fetch stage is sent.

c. Execute

The ALUIn2_control takes in RegOut2, shamt_out, mov_shamt, extender_out, R, IW, shamt_ins, Imm and outputs ALUInput2. This acts as a multiplexer of sorts to determine one of the ALU inputs. The output of this goes to the ALU bbus which is the second input in the ALU. An always block and if statements are used to determine what is sent out to the ALU. Shamt_ins comes from the opcode decoder and it signifies LSL or LSR being done. If it's an r-type instruction and shamt_ins is high, the ALU outputs shamt_out. If it's just an r-type instruction, the output is the RegOut2. If it's an immediate type and iw type, the ALUInput2 receives mov_shamt. In all other cases, the ALUInput2 receives extender_out. This helps filter the input depending on the instruction type and operation currently being done.

The ALU_Aoper multiplexer takes in the RegOut1, extender_out and the IW control bit. If IW is high, a MOVZ instruction is being done and the extender out is passed to the ALUInput 1 or ALU abus. In all other cases RegOut1 is passed through to ALUInput 1. This was necessary as the MOVZ instruction needs the immediate to be shifted, and not the value of a register which is why the multiplexer controls the input to the ALU abus.

The ALU takes in two inputs abus and bbus, along with the S, Cin and SetFlag control bits. The abus and bbus contain the operands, the S control wire is a 3-bit binary digit. Different operations in the ALU are programmed to different combinations of S. The Cin is a 1-bit input which is high only for subtract operations. The SetFlag is a 1-bit input which tells the ALU to output the flags N, Z, V, C in addition to the ALUOutput. SetFlag is only active for instructions that end with an S, i.e ADDS, ANDS, ANDIS, SUBS, etc. The full list can be found in the green card. The output of the ALU is sent through to the memory stage, and the flags are sent back to the decode stage to be used in the branch check module. This particular ALU uses the look-ahead carry format in order to improve efficiency of the operations. Since we're dealing with 64-bit values, having a regular cascade adder would take significantly longer to finish operations.

d. Memory

The memory stage has two of our outputs, the daddrbus and the databus. The MEM/WB flip-flop sends the ALUOutput to the daddrbus directly and that value is send out to the user. It is also sent through to the MEM/WB flip flop where it will go to the writeback stage. No modification or control bits are needed to regulate the daddrbus in the memory stage.

The databus takes in the RegOut2 value from the register file. This is fed through a tri-state buffer which is controlled by the SW control signal. This is active only when the store word operation is being run. When SW is active, the RegOut2 value is sent out through the databus. When SW is inactive, the databus output is all Zs. Either RegOut2 or all Zs will be sent through the MEM/WB flip flop to the write-back stage. In the case of the LDUR instruction, the databus acts as an input which takes in a 64-bit value to be sent through to the writeback stage. Since SW would be inactive, we don't need to worry about interference from the RegOut2 value.

Additionally, the Dsel value and LW control bit are also pipelined through this stage to the Writeback stage.

e. Writeback

The writeback stage consists of a multiplexer, the databus, daddrbus, Dsel, and the LW control bit. The Dsel is sent directly to the register file with no further pipelining.

The databus, daddrbus and the LW control bit are sent to a 2x1 multiplexer with LW acting as the select signal. When LW is active, the databus value will be output from the mux which will be sent to the register file as the dbus value. This is because the value is loaded in from databus in the memory stage. When LW is inactive, the daddrbus will be sent from the multiplexer to dbus as the databus is set to all Zs.

IV. Verilog Code

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Aryan Mehrotra
//
// Module Name: PC
// Description: Program Counter D Flip-Flop
// Project Name: Final Project
//////////////////////////////////////////////////////////////////////////////////


module PC(clk, reset, pc_in, pc_out);
   input [63:0] pc_in;
   input clk, reset;
   output reg [63:0] pc_out;

   always@(posedge clk or posedge reset) begin
      if (reset) begin
         pc_out <= 64'h0000000000000000;
      end
      else if (clk) begin
         pc_out <= pc_in;
      end
   end
endmodule

`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Aryan Mehrotra
//
// Module Name: PC_Adder1
// Description: Adds 4 to the program counter
// Project Name: Final Project
//////////////////////////////////////////////////////////////////////////////////


module PC_Adder1(in_1, out_1);
   input[63:0] in_1;
   output[63:0] out_1;

   assign out_1 = in_1 + 64'h0000000000000004;
```

```
endmodule

`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Aryan Mehrotra
//
// Module Name: IF_ID_DFF
// Description: D Flip-Flop from instruction fetch to instruction decode stage
// Project Name: Final Project
//////////////////////////////////////////////////////////////////////////////////


module IF_ID_DFF(ibus_in, clk, PC_mux_in1, ibus_out, adder2_in1);
    input [31:0] ibus_in;
    input clk;
    input [63:0] PC_mux_in1;
    output reg [31:0] ibus_out;
    output reg [63:0] adder2_in1;

    always@(posedge clk) begin
        ibus_out = ibus_in;
        adder2_in1 = PC_mux_in1;
    end

endmodule

`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Aryan Mehrotra
//
// Module Name: rn_decoder
// Description: Pulls out Rn values from r_type, i_type and d_type formats
// Project Name: Final Project
//////////////////////////////////////////////////////////////////////////////////


module rn_decoder(ibus, rn_out);
    input[31:0] ibus;
    output[31:0] rn_out;

    wire[4:0] rn_val = ibus[9:5];
    integer rn_int;
    always@(rn_val) begin
        rn_int = rn_val;
    end

    assign rn_out = 32'b1 << rn_int;
endmodule

`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Aryan Mehrotra
//
// Module Name: Asel_control
// Description: Module to control the Asel input from decoders
// Project Name: Final Project
//////////////////////////////////////////////////////////////////////////////////


module Asel_control(rn_decoder_in, r_type, i_type, d_type, Asel);
    input[31:0] rn_decoder_in;
    input r_type, i_type, d_type;
    output[31:0] Asel;

    assign Asel = (r_type || i_type || d_type) ? rn_decoder_in : 32'bzzzzzzzz;

endmodule

`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Aryan Mehrotra
//
// Module Name: rd_decoder
// Description: Pulls out Rd/Rt values from r_type, i_type and d_type, cb_type and
iw_type formats
// Project Name: Final Project
//////////////////////////////////////////////////////////////////////////////////


module rd_decoder(ibus, rd_out);
    input[31:0] ibus;
    output[31:0] rd_out;

    wire[4:0] rd_val = ibus[4:0];
    integer rd_int;
```

```
   always@(rd_val) begin
      rd_int = rd_val;
   end

   assign rd_out = 32'b1 << rd_int;
endmodule

`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Aryan Mehrotra
//
// Module Name: rm_decoder
// Description: Pulls out Rm values from r_type formats
// Project Name: Final Project
//////////////////////////////////////////////////////////////////////////////////


module rm_decoder(ibus, rm_out);
   input[31:0] ibus;
   output[31:0] rm_out;

   wire[4:0] rm_val = ibus[20:16];
   integer rm_int;

   always@(rm_val) begin
      rm_int = rm_val;
   end

   assign rm_out = 32'b1 << rm_int;
endmodule

`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Aryan Mehrotra
//
// Module Name: Bsel_control
// Description: Module to control Bselect input from decoders
// Project Name: Final Project
//////////////////////////////////////////////////////////////////////////////////
```

```
module Bsel_control(rd_decoder_in, rm_decoder_in, r_type, i_type, d_type, b_type,
cb_type, iw_type, Bsel, Dsel_ID);
   input[31:0] rd_decoder_in, rm_decoder_in;
   input r_type, i_type, d_type, b_type, cb_type, iw_type;
   output reg[31:0] Bsel, Dsel_ID;

   always@(rd_decoder_in, rm_decoder_in, r_type, i_type, d_type, b_type, cb_type,
iw_type) begin
      if(r_type) begin
         Bsel = rm_decoder_in;
         Dsel_ID = rd_decoder_in;
      end
      else if(i_type || iw_type) begin
          Bsel = 32'hzzzzzzzz;
          Dsel_ID = rd_decoder_in;
      end
      else if(d_type) begin
         Bsel = rd_decoder_in;
         Dsel_ID = rd_decoder_in;
      end
      else if(cb_type) begin
          Bsel = rd_decoder_in;
          Dsel_ID = rd_decoder_in;
      end
      else if(b_type) begin
          Bsel = 32'hzzzzzzzz;
          Dsel_ID = 32'h80000000;
      end
   end
endmodule

`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Aryan Mehrotra
//
// Module Name: shamt
// Description: Pulls out shift amount from R-Type instructions
// Project Name: Final Project
//////////////////////////////////////////////////////////////////////////////////


module shamt(ibus, shamt_out);
   input[31:0] ibus;
```

```
    output[5:0] shamt_out;

    assign shamt_out = ibus[15:10];
endmodule

`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Aryan Mehrotra
//
// Module Name: Extender
// Description: Zero extends or sign extends immidiates or branch addresses
// Project Name: Final Project
//////////////////////////////////////////////////////////////////////////////////


module Extender(ibus, i_type, d_type, b_type, cb_type, iw_type, mov_shamt,
extender_out);
    input[31:0] ibus;
    input i_type, d_type, b_type, cb_type, iw_type;
    output reg [63:0] extender_out, mov_shamt;

    reg[11:0] ALU_Imm;
    reg[25:0] BR_address;
    reg[15:0] MOV_immediate;

    always@(ibus, i_type, d_type, b_type, cb_type, iw_type) begin
        if(i_type) begin
            ALU_Imm = ibus[21:10];
            mov_shamt = {58'b0 , ibus[22:21] << 4};
            extender_out = {52'b0, ALU_Imm};
        end
        if(d_type) begin
            mov_shamt = {58'b0 , ibus[22:21] << 4};
            extender_out = {{55{ibus[20]}}, ibus[20:12]};
        end
        if(b_type) begin
            mov_shamt = {58'b0 , ibus[22:21] << 4};
            extender_out = {{36{ibus[25]}}, ibus[25:0], 2'b0};
        end
        if(cb_type) begin
            mov_shamt = {58'b0 , ibus[22:21] << 4};
            extender_out = {{43{ibus[23]}}, ibus[23:5], 2'b0};
        end
```

```
      if(iw_type) begin
         MOV_immediate = ibus[20:5];
         mov_shamt = {58'b0 , ibus[22:21] << 4};
         extender_out = {48'b0, MOV_immediate};
      end
   end
endmodule

`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Aryan Mehrotra
//
// Module Name: RegFile
// Description: Thirty-two 64-bit registers to make-up the register file
// Project Name: Final Project
//////////////////////////////////////////////////////////////////////////////////


module RegFile(Aselect, Bselect, DSelect, abus, bbus, dbus, clk);
   // Declare input and output variables
   input[31:0] Aselect, Bselect, DSelect;
   input[63:0] dbus;
   input clk;
   output[63:0] abus, bbus;

  // instantiate zero register
  ZeroReg zeroreg(Aselect[31], Bselect[31], clk, DSelect[31], abus, bbus);

  // generate the other 31 registers using RegDFF32 and ith bits of inputs
  genvar i; // <-- genvar to use in iteration
  generate
   for (i = 0; i < 31; i = i + 1) begin
      RegFile_DFF reg64(Aselect[i], Bselect[i], DSelect[i], clk, abus, bbus, dbus);
      end
  endgenerate

endmodule

`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Aryan Mehrotra
//
// Module Name: RegFile_DFF
// Description: Designated neg-edge D Flip-Flops for RegFile
// Project Name: Final Project
//////////////////////////////////////////////////////////////////////////////////

module RegFile_DFF(Asel, Bsel, Dselect, clk, abus, bbus, dbus);
   // Declare input and output variables
   input[63:0] dbus;
   input clk, Dselect, Asel, Bsel;
   reg[63:0] Q;
   output[63:0] abus, bbus;
   assign newclk = clk & Dselect;
   /*
   always @(negedge newclk) begin
      Q = dbus;
   end
   */
   // Set-up modified clk to assign to Q
   always @(negedge clk) begin
      if (Dselect==1'b1)
         Q = dbus;
   end

   // Set-up tri-state buffer
   assign abus = Asel ? Q : 64'bz;
   assign bbus = Bsel ? Q : 64'bz;
endmodule
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Aryan Mehrotra
//
// Module Name: ZeroReg
// Description: Designated Zero Register for RegFile
// Project Name: Final Project
//////////////////////////////////////////////////////////////////////////////////


module ZeroReg(Asel, Bsel, clk, Dselect, abus, bbus);
   // Declare input and output variables
   input clk, Dselect, Asel, Bsel;
   reg[63:0] Q;
   output[63:0] abus, bbus;
   assign newclk = clk & Dselect;
   /*
```

```
    always @(negedge newclk) begin
        Q = 64'b0;
    end
    */
    // Set-up modified clk to assign to Q
    always @(negedge clk) begin
        Q = 64'b0;
    end

    // Set-up tri-state buffer
    assign abus = Asel ? Q : 64'bz;
    assign bbus = Bsel ? Q : 64'bz;
endmodule
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Aryan Mehrotra
//
// Module Name: Branch_Check
// Description: Checks the RegFile outputs and opcode signals to see if a branch is needed
// Project Name: Final Project
//////////////////////////////////////////////////////////////////////////////////


module Branch_Check(RegOut1, RegOut2, N, Z, V, C, zcomp, nzcomp, BEQ, BNE,
BLT, BGE, mux_sel, b_type);
    input[63:0] RegOut1, RegOut2;
    input N, Z, V, C, zcomp, nzcomp, BEQ, BNE, BLT, BGE, b_type;
    output reg mux_sel;

    always@(RegOut1, RegOut2, N, Z, V, C, zcomp, nzcomp, BEQ, BNE, BLT, BGE,
b_type) begin
        if(BLT && (N != V)) begin
            mux_sel <= 1'b1;
        end
        else if(BGE && (N == V)) begin
            mux_sel <= 1'b1;
        end
        else if(BEQ && (Z == 1)) begin
            mux_sel <= 1'b1;
        end
        else if(BNE && (Z == 0)) begin
            mux_sel <= 1'b1;
        end
        else if(zcomp && (RegOut2 == 0)) begin
            mux_sel <= 1'b1;
        end
        else if(nzcomp && (RegOut2 != 0)) begin
            mux_sel <= 1'b1;
        end
        else if(b_type) begin
            mux_sel <= 1'b1;
        end
        else begin
            mux_sel <= 1'b0;
        end
    end
endmodule
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Aryan Mehrotra
//
// Module Name: PC_Adder2
// Description: Adds branch offset to the program counter
// Project Name: Final Project
//////////////////////////////////////////////////////////////////////////////////


module PC_Adder2(in_1, in_2, out_1);
    input[63:0] in_1, in_2;
    output[63:0] out_1;

    assign out_1 = in_1 + in_2;
endmodule
`timescale 1ns / 1ps



module PC_mux(mux_in1, mux_in2, sel, mux_out);
    //Input and output declaration
    input [63:0] mux_in1, mux_in2;
    input sel;
    output reg [63:0] mux_out;

    //If select (sel) is 1, output mux_in2, otherwise output mux_in1
     always@(mux_in1, mux_in2, sel) begin
        mux_out = mux_in1;
        if(sel) begin
            mux_out = mux_in2;
        end
    end
endmodule
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Aryan Mehrotra
//
// Module Name: ID_EX_DFF
// Description: D Flip-Flop from instruction decode to execute stage
// Project Name: Final Project
//////////////////////////////////////////////////////////////////////////////////


module ID_EX_DFF(RegOut1, RegOut2_ID, Imm_ID, S_ID, Cin_ID, SW_ID,
LW_ID,iw_type_ID, BEQ_ID, BNE_ID, BLT_ID, BGE_ID, shamt_ID,
                CBZ_ID, CBNZ_ID, SetFlag_ID, extender_out_ID, Dsel_ID, clk, ALUInput1,
RegOut2_EX, S_EX, Imm_EX, SW_EX,
                LW_EX, Cin_EX, Dsel_EX, extender_out_EX, BEQ_EX, BNE_EX,
BLT_EX, BGE_EX, CBZ_EX, CBNZ_EX, shamt_EX, SetFlag_EX,
                r_type_ID, r_type_EX, shamt_ins_ID, shamt_ins_EX, mov_shamt_EX,
mov_shamt_ID, iw_type_EX);

    input BEQ_ID, BNE_ID, BLT_ID, BGE_ID, CBZ_ID, CBNZ_ID, SetFlag_ID, clk,
Imm_ID, Cin_ID, SW_ID, LW_ID, r_type_ID,
            shamt_ins_ID, iw_type_ID;
    input[2:0] S_ID;
    input[5:0] shamt_ID;
    input[31:0] Dsel_ID;
    input[63:0] RegOut1, RegOut2_ID, extender_out_ID, mov_shamt_ID;
    output reg BEQ_EX, BNE_EX, BLT_EX, BGE_EX, CBZ_EX, CBNZ_EX,
SetFlag_EX, Imm_EX, SW_EX, LW_EX, Cin_EX,
               r_type_EX, shamt_ins_EX, iw_type_EX;
    output reg[2:0] S_EX;
    output reg[5:0] shamt_EX;
    output reg[31:0] Dsel_EX;
    output reg[63:0]  ALUInput1, RegOut2_EX, extender_out_EX, mov_shamt_EX;

    always@(posedge clk) begin
        BEQ_EX = BEQ_ID;
        BNE_EX = BNE_ID;
        BLT_EX = BLT_ID;
        BGE_EX = BGE_ID;
        CBZ_EX = CBZ_ID;
        CBNZ_EX = CBNZ_ID;
        r_type_EX = r_type_ID;
        iw_type_EX = iw_type_ID;
        shamt_ins_EX = shamt_ins_ID;
        SetFlag_EX = SetFlag_ID;
        Imm_EX = Imm_ID;
        SW_EX = SW_ID;
        LW_EX = LW_ID;
        Cin_EX = Cin_ID;
        S_EX = S_ID;
        shamt_EX = shamt_ID;
        Dsel_EX = Dsel_ID;
        ALUInput1 = RegOut1;
        RegOut2_EX = RegOut2_ID;
        mov_shamt_EX = mov_shamt_ID;
        extender_out_EX = extender_out_ID;
    end
endmodule
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Aryan Mehrotra
//
// Module Name: ALUIn2_control
// Description: Module to control ALU Input 2 depending on instruction type
// Project Name: Final Project
//////////////////////////////////////////////////////////////////////////////////


module ALUIn2_control(RegOut2_EX, shamt_EX, iw_type, mov_shamt_EX,
extender_out_EX, r_type, shamt_ins, Imm_EX, ALUInput2);
    input r_type, shamt_ins, Imm_EX, iw_type;
    input[5:0] shamt_EX;
    input[63:0] RegOut2_EX, extender_out_EX, mov_shamt_EX;
    output reg[63:0] ALUInput2;

    always@(RegOut2_EX, shamt_EX, mov_shamt_EX, extender_out_EX, r_type,
shamt_ins, Imm_EX) begin
        //LSR or LSL
        if(r_type & shamt_ins) begin
            ALUInput2 = shamt_EX;
        end
        //All other R-types
        else if(r_type) begin
           ALUInput2 = RegOut2_EX;
        end
        else if(Imm_EX) begin
           //MOVZ specifically
           if(iw_type) begin
              ALUInput2 = mov_shamt_EX;
           end
           //All other I-types
           else begin
              ALUInput2 = extender_out_EX;
           end
        end
        //Default
        else begin
              ALUInput2 = extender_out_EX;
        end
     end
endmodule
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Aryan Mehrotra
//
// Module Name: mux64
// Description: 2 x 1 multiplexer which takes in 2 64-bit inputs
// Project Name: Final Project
//////////////////////////////////////////////////////////////////////////////////


module mux64(mux_in1, mux_in2, sel, mux_out);
   //Input and output declaration
   input [63:0] mux_in1, mux_in2;
   input sel;
   output [63:0] mux_out;

   //If select (sel) is 1, output mux_in2, otherwise output mux_in1
   assign mux_out = sel ? mux_in2 : mux_in1;

endmodule
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Aryan Mehrotra
```

Aryan Mehrotra
Final Project
EECE3324

```
//
// Module Name: ALU
// Description: Arithmetic Logic Unit with 64-bit support and ARM flags
// Project Name: Final Project
//////////////////////////////////////////////////////////////////////////////


module ALU(ALU_abus, ALU_bbus, Cin, S, ALU_Out, N, Z, V, C, SetFlag);

   input[63:0] ALU_abus, ALU_bbus;
   input Cin, SetFlag;
   input [2:0] S;
   output reg [63:0] ALU_Out;
   output reg N, Z, V, C;
   wire [63:0] temp_out;
   wire [63:0] c_1, g, p;
   wire gout, pout;
   wire V_t, C_t;

  alu_cell alucell[63:0] (
    .d(temp_out),
    .g(g),
    .p(p),
    .a(ALU_abus),
    .b(ALU_bbus),
    .c_1(c_1),
    .S(S)
  );

  lac6 laclevel6(
    .c_1(c_1),
    .gout(gout),
    .pout(pout),
    .Cin(Cin),
    .g(g),
    .p(p)
  );

  overflow over(
    .c_1(c_1),
    .gout(gout),
    .pout(pout),
    .Cin(Cin),
    .Cout(C_t),
    .V(V_t)
  );


  always@(temp_out, SetFlag, S) begin
      assign ALU_Out = temp_out;

      if(S == 3'b101) begin
          assign ALU_Out = ALU_abus << ALU_bbus;
      end
      if(S == 3'b111) begin
          assign ALU_Out = ALU_abus >>> ALU_bbus;
      end
      if(SetFlag) begin
          N = ALU_Out[63];
          Z = (ALU_Out == 0) ? 1:0;
          V = V_t;
          C = C_t;
      end
  end
endmodule

module alu_cell (d, g, p, a, b, c_1, S);
  output d, g, p;
  input a, b, c_1;
  input [2:0] S;
  reg g,d,p,cint,bint;

  always @(a,b,c_1,S,p,g) begin
    bint = S[0] ^ b;
    g = a & bint;
    p = a ^ bint;
    cint = S[1] & c_1;

           //expand this module to handle S[2]
    if(S[2] == 0)
      begin
          d = p ^ cint;
      end
    else if (S[2] == 1)
      begin
          if(S[1] == 0 && S[0] == 0)
```

```
                begin
                    d = a | b;
                end
            else if(S[1] == 0 && S[0] == 1)
                begin
                    d = a << b;
                end
            else if(S[1] == 1 && S[0] == 0)
                begin
                    d = a & b;
                end
            else if(S[1] == 1 && S[0] == 1)
                begin
                    d = a >>> b;
                end
        end
    end

endmodule

module overflow (c_1, gout, pout, Cin, Cout, V);
    input[63:0] c_1;
    input gout, pout, Cin;
    output Cout, V;

    assign Cout = gout | (pout & Cin);
    assign V = Cout ^ c_1[63];
endmodule

module lac(c_1, gout, pout, Cin, g, p);

   output [1:0] c_1;
   output gout;
   output pout;
   input Cin;
   input [1:0] g;
   input [1:0] p;

   assign c_1[0] = Cin;
   assign c_1[1] = g[0] | ( p[0] & Cin );
   assign gout = g[1] | ( p[1] & g[0] );
   assign pout = p[1] & p[0];
```

```
endmodule

module lac2 (c_1, gout, pout, Cin, g, p);
  output [3:0] c_1;
  output gout, pout;
  input Cin;
  input [3:0] g, p;

  wire [1:0] cint, gint, pint;

  lac leaf0(
    .c_1(c_1[1:0]),
    .gout(gint[0]),
    .pout(pint[0]),
    .Cin(cint[0]),
    .g(g[1:0]),
    .p(p[1:0])
  );

  lac leaf1(
    .c_1(c_1[3:2]),
    .gout(gint[1]),
    .pout(pint[1]),
    .Cin(cint[1]),
    .g(g[3:2]),
    .p(p[3:2])
  );

  lac root(
    .c_1(cint),
    .gout(gout),
    .pout(pout),
    .Cin(Cin),
    .g(gint),
    .p(pint)
  );
endmodule


module lac3 (c_1, gout, pout, Cin, g, p);
  output [7:0] c_1;
  output gout, pout;
  input Cin;
  input [7:0] g, p;

  wire [1:0] cint, gint, pint;

  lac2 leaf0(
    .c_1(c_1[3:0]),
    .gout(gint[0]),
    .pout(pint[0]),
    .Cin(cint[0]),
    .g(g[3:0]),
    .p(p[3:0])
  );

  lac2 leaf1(
    .c_1(c_1[7:4]),
    .gout(gint[1]),
    .pout(pint[1]),
    .Cin(cint[1]),
    .g(g[7:4]),
    .p(p[7:4])
  );

  lac root(
    .c_1(cint),
    .gout(gout),
    .pout(pout),
    .Cin(Cin),
    .g(gint),
    .p(pint)
  );
endmodule


module lac4 (c_1, gout, pout, Cin, g, p);
   output [15:0] c_1;
   output gout, pout;
   input Cin;
   input [15:0] g, p;

   wire [1:0] cint, gint, pint;

   lac3 leaf0(
    .c_1(c_1[7:0]),
    .gout(gint[0]),
    .pout(pint[0]),
    .Cin(cint[0]),
    .g(g[7:0]),
    .p(p[7:0])
    );

  lac3 leaf1(
    .c_1(c_1[15:8]),
    .gout(gint[1]),
    .pout(pint[1]),
    .Cin(cint[1]),
    .g(g[15:8]),
    .p(p[15:8])
    );

  lac root(
    .c_1(cint),
    .gout(gout),
    .pout(pout),
    .Cin(Cin),
    .g(gint),
    .p(pint)
    );

endmodule


module lac5 (c_1, gout, pout, Cin, g, p);
  output [31:0] c_1;
  output gout, pout;
  input Cin;
  input [31:0] g, p;

  wire [1:0] cint, gint, pint;

  lac4 leaf0(
    .c_1(c_1[15:0]),
    .gout(gint[0]),
    .pout(pint[0]),
    .Cin(cint[0]),
    .g(g[15:0]),
    .p(p[15:0])
  );

  lac4 leaf1(
    .c_1(c_1[31:16]),
    .gout(gint[1]),
    .pout(pint[1]),
    .Cin(cint[1]),
    .g(g[31:16]),
    .p(p[31:16])
  );

  lac root(
    .c_1(cint),
    .gout(gout),
    .pout(pout),
    .Cin(Cin),
    .g(gint),
    .p(pint)
    );
endmodule

module lac6 (c_1, gout, pout, Cin, g, p);
  output [63:0] c_1;
  output gout, pout;
  input Cin;
  input [63:0] g, p;

  wire [1:0] cint, gint, pint;

  lac5 leaf0(
    .c_1(c_1[31:0]),
    .gout(gint[0]),
    .pout(pint[0]),
    .Cin(cint[0]),
    .g(g[31:0]),
    .p(p[31:0])
  );

  lac5 leaf1(
    .c_1(c_1[63:32]),
    .gout(gint[1]),
    .pout(pint[1]),
    .Cin(cint[1]),
    .g(g[63:32]),
    .p(p[63:32])
  );

  lac root(
    .c_1(cint),
    .gout(gout),
    .pout(pout),
    .Cin(Cin),
    .g(gint),
    .p(pint)
    );
endmodule
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Aryan Mehrotra
//
// Module Name: EX_MEM_DFF
// Description: D Flip-Flop from execute stage to memory stage
// Project Name: Final Project
//////////////////////////////////////////////////////////////////////////////////


module EX_MEM_DFF(ALUOutput, Dsel_EX, RegOut2_EX, SW_EX, LW_EX,
BEQ_EX, BNE_EX, BLT_EX, BGE_EX, zcomp_EX, nzcomp_EX,
                 clk, daddrbus, Dsel_MEM, SW_MEM, LW_MEM, databus_in, BEQ_MEM,
BNE_MEM, BLT_MEM, BGE_MEM, zcomp_MEM, nzcomp_MEM);
  //Declare inputs and outputs
  input[31:0] Dsel_EX;
  input[63:0] ALUOutput, RegOut2_EX;
  input clk, SW_EX, LW_EX, BEQ_EX, BNE_EX, BLT_EX, BGE_EX, zcomp_EX,
nzcomp_EX;
  output reg[31:0] Dsel_MEM;
  output reg[63:0] daddrbus, databus_in;
  output reg SW_MEM, LW_MEM, BEQ_MEM, BNE_MEM, BLT_MEM,
BGE_MEM, zcomp_MEM, nzcomp_MEM;

   always @(posedge clk) begin
        SW_MEM = SW_EX;
        LW_MEM = LW_EX;
        BEQ_MEM = BEQ_EX;
        BNE_MEM = BNE_EX;
        BLT_MEM = BLT_EX;
        BGE_MEM = BGE_EX;
        zcomp_MEM = zcomp_EX;
        nzcomp_MEM = nzcomp_EX;
        Dsel_MEM = Dsel_EX;
        databus_in = RegOut2_EX;
        daddrbus = ALUOutput;
    end
endmodule
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Aryan Mehrotra
//
// Module Name: Data_Mem
// Description: Tri-State Buffer for databus access to data memory
// Project Name: Final Project
//////////////////////////////////////////////////////////////////////////////////


module Data_Mem(b_oper, databus, SW_MEM);
    inout[63:0] databus;
    input[63:0] b_oper;
    input SW_MEM;

    assign databus = SW_MEM ?  b_oper : 64'bz;

endmodule
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Aryan Mehrotra
//
// Module Name: MEM_WB_DFF
// Description: D Flip-Flop from memory stage to write-back stage
// Project Name: Final Project
//////////////////////////////////////////////////////////////////////////////////


module MEM_WB_DFF(daddrbus_in, databus_in, Dselect_in, clk, SW_in, LW_in,
BEQ_in, BNE_in, BLT_in, BGE_in,
              zcomp_in, nzcomp_in, daddrbus_out, databus_out, Dselect_out, LW_out);
    //Declare inputs and outputs
    input[31:0] Dselect_in;
    input[63:0] daddrbus_in, databus_in;
    input clk, SW_in, LW_in, BEQ_in, BNE_in, BLT_in, BGE_in, zcomp_in, nzcomp_in;
    output reg[31:0] Dselect_out;
    output reg[63:0] daddrbus_out, databus_out;
    output reg LW_out;



    always @(posedge clk) begin
        Dselect_out = (SW_in || BEQ_in || BNE_in || BLT_in || BGE_in || zcomp_in ||
nzcomp_in) ? 32'h80000000 : Dselect_in;
        LW_out = LW_in;
        databus_out = databus_in;
        daddrbus_out = daddrbus_in;
    end

endmodule
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Aryan Mehrotra
//
// Module Name: mux64
// Description: 2 x 1 multiplexer which takes in 2 64-bit inputs
// Project Name: Final Project
//////////////////////////////////////////////////////////////////////////////////


module mux64(mux_in1, mux_in2, sel, mux_out);
    //Input and output declaration
    input [63:0] mux_in1, mux_in2;
    input sel;
    output [63:0] mux_out;

    //If select (sel) is 1, output mux_in2, otherwise output mux_in1
    assign mux_out = sel ? mux_in2 : mux_in1;

endmodule
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Aryan Mehrotra
//
// Module Name: opcode_decoder
// Description: Gives the instruction type from the operation code in ibus
// Project Name: Final Project
//////////////////////////////////////////////////////////////////////////////////
```

```
module opcode_decoder(ibus, S_ID, Cin_ID, Imm_ID,
                    SetFlag_ID, SW_ID, LW_ID, r_type,
                    i_type, d_type, b_type, cb_type, iw_type,
                    zcomp, nzcomp, BEQ, BNE, BLT, BGE, shamt_ins);
   //Instantiate inputs and outputs
   input[31:0] ibus;
   output reg [2:0] S_ID;
   output reg Cin_ID, Imm_ID, SetFlag_ID, SW_ID, LW_ID, r_type, i_type, d_type,
b_type, cb_type, iw_type,
            zcomp, nzcomp, BEQ, BNE, BLT, BGE, shamt_ins;

   wire[10:0] r_code = ibus[31:21];
   wire[9:0] i_code = ibus[31:22];
   wire[10:0] d_code = ibus[31:21];
   wire[5:0] b_code = ibus[31:26];
   wire[8:0] cb_code = ibus[31:24];
   wire[10:0] iw_code = ibus[31:21];

   always@(ibus) begin
      case(r_code)
         //ADD
         11'b00101000000: begin
            S_ID = 3'b010;
            Cin_ID = 1'b0;
            Imm_ID = 1'b0;
            SetFlag_ID = 1'b0;
            r_type = 1'b1;
            i_type = 1'b0;
            d_type = 1'b0;
            b_type = 1'b0;
            cb_type = 1'b0;
            iw_type = 1'b0;
            zcomp = 1'b0;
            nzcomp = 1'b0;
            BEQ = 1'b0;
            BNE = 1'b0;
            BLT = 1'b0;
            BGE = 1'b0;
            SW_ID = 1'b0;
            LW_ID = 1'b0;
            shamt_ins = 1'b0;
```

```
end
//ADDS
11'b00101000001: begin
    S_ID = 3'b010;
    Cin_ID = 1'b0;
    Imm_ID = 1'b0;
    SetFlag_ID = 1'b1;
    r_type = 1'b1;
    i_type = 1'b0;
    d_type = 1'b0;
    b_type = 1'b0;
    cb_type = 1'b0;
    iw_type = 1'b0;
    zcomp = 1'b0;
    nzcomp = 1'b0;
    BEQ = 1'b0;
    BNE = 1'b0;
    BLT = 1'b0;
    BGE = 1'b0;
    SW_ID = 1'b0;
    LW_ID = 1'b0;
    shamt_ins = 1'b0;
end
//AND
11'b00101000010: begin
    S_ID = 3'b110;
    Cin_ID = 1'b0;
    Imm_ID = 1'b0;
    SetFlag_ID = 1'b0;
    r_type = 1'b1;
    i_type = 1'b0;
    d_type = 1'b0;
    b_type = 1'b0;
    cb_type = 1'b0;
    iw_type = 1'b0;
    zcomp = 1'b0;
    nzcomp = 1'b0;
    BEQ = 1'b0;
    BNE = 1'b0;
    BLT = 1'b0;
    BGE = 1'b0;
    SW_ID = 1'b0;
    LW_ID = 1'b0;
    shamt_ins = 1'b0;
end
//ANDS
11'b00101000011: begin
    S_ID = 3'b110;
    Cin_ID = 1'b0;
    Imm_ID = 1'b0;
    SetFlag_ID = 1'b1;
    r_type = 1'b1;
    i_type = 1'b0;
    d_type = 1'b0;
    b_type = 1'b0;
    cb_type = 1'b0;
    iw_type = 1'b0;
    zcomp = 1'b0;
    nzcomp = 1'b0;
    BEQ = 1'b0;
    BNE = 1'b0;
    BLT = 1'b0;
    BGE = 1'b0;
    SW_ID = 1'b0;
    LW_ID = 1'b0;
    shamt_ins = 1'b0;
end
//EOR
11'b00101000100: begin
    S_ID = 3'b000;
    Cin_ID = 1'b0;
    Imm_ID = 1'b0;
    SetFlag_ID = 1'b0;
    r_type = 1'b1;
    i_type = 1'b0;
    d_type = 1'b0;
    b_type = 1'b0;
    cb_type = 1'b0;
    iw_type = 1'b0;
    zcomp = 1'b0;
    nzcomp = 1'b0;
    BEQ = 1'b0;
    BNE = 1'b0;
    BLT = 1'b0;
    BGE = 1'b0;
    SW_ID = 1'b0;
    LW_ID = 1'b0;
    shamt_ins = 1'b0;
end
//ENOR
11'b00101000101: begin
    S_ID = 3'b001;
    Cin_ID = 1'b0;
    Imm_ID = 1'b0;
    SetFlag_ID = 1'b0;
    r_type = 1'b1;
    i_type = 1'b0;
    d_type = 1'b0;
    b_type = 1'b0;
    cb_type = 1'b0;
    iw_type = 1'b0;
    zcomp = 1'b0;
    nzcomp = 1'b0;
    BEQ = 1'b0;
    BNE = 1'b0;
    BLT = 1'b0;
    BGE = 1'b0;
    SW_ID = 1'b0;
    LW_ID = 1'b0;
    shamt_ins = 1'b0;
end
//LSL
11'b00101000110: begin
    S_ID = 3'b101;
    Cin_ID = 1'b0;
    Imm_ID = 1'b0;
    SetFlag_ID = 1'b0;
    r_type = 1'b1;
    i_type = 1'b0;
    d_type = 1'b0;
    b_type = 1'b0;
    cb_type = 1'b0;
    iw_type = 1'b0;
    zcomp = 1'b0;
    nzcomp = 1'b0;
    BEQ = 1'b0;
    BNE = 1'b0;
    BLT = 1'b0;
    BGE = 1'b0;
    SW_ID = 1'b0;
    LW_ID = 1'b0;
    shamt_ins = 1'b1;
end
//LSR
11'b00101000111: begin
    S_ID = 3'b111;
    Cin_ID = 1'b0;
    Imm_ID = 1'b0;
    SetFlag_ID = 1'b0;
    r_type = 1'b1;
    i_type = 1'b0;
    d_type = 1'b0;
    b_type = 1'b0;
    cb_type = 1'b0;
    iw_type = 1'b0;
    zcomp = 1'b0;
    nzcomp = 1'b0;
    BEQ = 1'b0;
    BNE = 1'b0;
    BLT = 1'b0;
    BGE = 1'b0;
    SW_ID = 1'b0;
    LW_ID = 1'b0;
    shamt_ins = 1'b1;
end
//ORR
11'b00101001000: begin
    S_ID = 3'b100;
    Cin_ID = 1'b0;
    Imm_ID = 1'b0;
    SetFlag_ID = 1'b0;
    r_type = 1'b1;
    i_type = 1'b0;
    d_type = 1'b0;
    b_type = 1'b0;
    cb_type = 1'b0;
    iw_type = 1'b0;
    zcomp = 1'b0;
    nzcomp = 1'b0;
    BEQ = 1'b0;
    BNE = 1'b0;
    BLT = 1'b0;
    BGE = 1'b0;
    SW_ID = 1'b0;
    LW_ID = 1'b0;
    shamt_ins = 1'b0;
end
//SUB
11'b00101001001: begin
    S_ID = 3'b011;
    Cin_ID = 1'b1;
    Imm_ID = 1'b0;
    SetFlag_ID = 1'b0;
    r_type = 1'b1;
    i_type = 1'b0;
    d_type = 1'b0;
    b_type = 1'b0;
    cb_type = 1'b0;
    iw_type = 1'b0;
    zcomp = 1'b0;
    nzcomp = 1'b0;
    BEQ = 1'b0;
    BNE = 1'b0;
    BLT = 1'b0;
    BGE = 1'b0;
    SW_ID = 1'b0;
    LW_ID = 1'b0;
    shamt_ins = 1'b0;
end
//SUBS
11'b00101001010: begin
    S_ID = 3'b011;
    Cin_ID = 1'b1;
    Imm_ID = 1'b0;
    SetFlag_ID = 1'b1;
    r_type = 1'b1;
    i_type = 1'b0;
    d_type = 1'b0;
    b_type = 1'b0;
    cb_type = 1'b0;
    iw_type = 1'b0;
    zcomp = 1'b0;
    nzcomp = 1'b0;
    BEQ = 1'b0;
    BNE = 1'b0;
```

```
        BLT = 1'b0;
        BGE = 1'b0;
        SW_ID = 1'b0;
        LW_ID = 1'b0;
        shamt_ins = 1'b0;
    end
    11'b00000000000: begin
        S_ID = 3'bxxx;
        Cin_ID = 1'bx;
        Imm_ID = 1'b0;
        SetFlag_ID = 1'b0;
        r_type = 1'b0;
        i_type = 1'b0;
        d_type = 1'b0;
        b_type = 1'b0;
        cb_type = 1'b0;
        iw_type = 1'b0;
        zcomp = 1'b0;
        nzcomp = 1'b0;
        BEQ = 1'b0;
        BNE = 1'b0;
        BLT = 1'b0;
        BGE = 1'b0;
        SW_ID = 1'b0;
        LW_ID = 1'b0;
        shamt_ins = 1'b0;
    end
endcase
case(i_code)
    //ADDI
    10'b1000100000: begin
        S_ID = 3'b010;
        Cin_ID = 1'b0;
        Imm_ID = 1'b1;
        SetFlag_ID = 1'b0;
        r_type = 1'b0;
        i_type = 1'b1;
        d_type = 1'b0;
        b_type = 1'b0;
        cb_type = 1'b0;
        iw_type = 1'b0;
        zcomp = 1'b0;
        nzcomp = 1'b0;
    BEQ = 1'b0;
    BNE = 1'b0;
    BLT = 1'b0;
    BGE = 1'b0;
    SW_ID = 1'b0;
    LW_ID = 1'b0;
    shamt_ins = 1'b0;
end
//ADDIS
10'b1000100001: begin
    S_ID = 3'b010;
    Cin_ID = 1'b0;
    Imm_ID = 1'b1;
    SetFlag_ID = 1'b1;
    r_type = 1'b0;
    i_type = 1'b1;
    d_type = 1'b0;
    b_type = 1'b0;
    cb_type = 1'b0;
    iw_type = 1'b0;
    zcomp = 1'b0;
    nzcomp = 1'b0;
    BEQ = 1'b0;
    BNE = 1'b0;
    BLT = 1'b0;
    BGE = 1'b0;
    SW_ID = 1'b0;
    LW_ID = 1'b0;
    shamt_ins = 1'b0;
end
//ANDI
10'b1000100010: begin
    S_ID = 3'b110;
    Cin_ID = 1'b0;
    Imm_ID = 1'b1;
    SetFlag_ID = 1'b0;
    r_type = 1'b0;
    i_type = 1'b1;
    d_type = 1'b0;
    b_type = 1'b0;
    cb_type = 1'b0;
    iw_type = 1'b0;
    zcomp = 1'b0;
    nzcomp = 1'b0;
    BEQ = 1'b0;
    BNE = 1'b0;
    BLT = 1'b0;
    BGE = 1'b0;
    SW_ID = 1'b0;
    LW_ID = 1'b0;
    shamt_ins = 1'b0;
end
//ANDIS
10'b1000100011: begin
    S_ID = 3'b110;
    Cin_ID = 1'b0;
    Imm_ID = 1'b1;
    SetFlag_ID = 1'b1;
    r_type = 1'b0;
    i_type = 1'b1;
    d_type = 1'b0;
    b_type = 1'b0;
    cb_type = 1'b0;
    iw_type = 1'b0;
    zcomp = 1'b0;
    nzcomp = 1'b0;
    BEQ = 1'b0;
    BNE = 1'b0;
    BLT = 1'b0;
    BGE = 1'b0;
    SW_ID = 1'b0;
    LW_ID = 1'b0;
    shamt_ins = 1'b0;
end
//EORI
10'b1000100100: begin
    S_ID = 3'b000;
    Cin_ID = 1'b0;
    Imm_ID = 1'b1;
    SetFlag_ID = 1'b0;
    r_type = 1'b0;
    i_type = 1'b1;
    d_type = 1'b0;
    b_type = 1'b0;
    cb_type = 1'b0;
    iw_type = 1'b0;
    zcomp = 1'b0;
    nzcomp = 1'b0;
    BEQ = 1'b0;
    BNE = 1'b0;
    BLT = 1'b0;
    BGE = 1'b0;
    SW_ID = 1'b0;
    LW_ID = 1'b0;
    shamt_ins = 1'b0;
end
//ENORI
10'b1000100101: begin
    S_ID = 3'b001;
    Cin_ID = 1'b0;
    Imm_ID = 1'b1;
    SetFlag_ID = 1'b0;
    r_type = 1'b0;
    i_type = 1'b1;
    d_type = 1'b0;
    b_type = 1'b0;
    cb_type = 1'b0;
    iw_type = 1'b0;
    zcomp = 1'b0;
    nzcomp = 1'b0;
    BEQ = 1'b0;
    BNE = 1'b0;
    BLT = 1'b0;
    BGE = 1'b0;
    SW_ID = 1'b0;
    LW_ID = 1'b0;
    shamt_ins = 1'b0;
end
//ORRI
10'b1000100110: begin
    S_ID = 3'b100;
    Cin_ID = 1'b0;
    Imm_ID = 1'b1;
    SetFlag_ID = 1'b0;
    r_type = 1'b0;
    i_type = 1'b1;
    d_type = 1'b0;
    b_type = 1'b0;
    cb_type = 1'b0;
    iw_type = 1'b0;
    zcomp = 1'b0;
    nzcomp = 1'b0;
    BEQ = 1'b0;
    BNE = 1'b0;
    BLT = 1'b0;
    BGE = 1'b0;
    SW_ID = 1'b0;
    LW_ID = 1'b0;
    shamt_ins = 1'b0;
end
//SUBI
10'b1000100111: begin
    S_ID = 3'b011;
    Cin_ID = 1'b1;
    Imm_ID = 1'b1;
    SetFlag_ID = 1'b0;
    r_type = 1'b0;
    i_type = 1'b1;
    d_type = 1'b0;
    b_type = 1'b0;
    cb_type = 1'b0;
    iw_type = 1'b0;
    zcomp = 1'b0;
    nzcomp = 1'b0;
    BEQ = 1'b0;
    BNE = 1'b0;
    BLT = 1'b0;
    BGE = 1'b0;
    SW_ID = 1'b0;
    LW_ID = 1'b0;
    shamt_ins = 1'b0;
end
//SUBIS
10'b1000101000: begin
    S_ID = 3'b011;
    Cin_ID = 1'b1;
    Imm_ID = 1'b1;
    SetFlag_ID = 1'b1;
    r_type = 1'b0;
    i_type = 1'b1;
    d_type = 1'b0;
    b_type = 1'b0;
```

```
            cb_type = 1'b0;
            iw_type = 1'b0;
            zcomp = 1'b0;
            nzcomp = 1'b0;
            BEQ = 1'b0;
            BNE = 1'b0;
            BLT = 1'b0;
            BGE = 1'b0;
            SW_ID = 1'b0;
            LW_ID = 1'b0;
            shamt_ins = 1'b0;
        end
        10'b0000000000: begin
            S_ID = 3'bxxx;
            Cin_ID = 1'bx;
            Imm_ID = 1'b0;
            SetFlag_ID = 1'b0;
            r_type = 1'b0;
            i_type = 1'b0;
            d_type = 1'b0;
            b_type = 1'b0;
            cb_type = 1'b0;
            iw_type = 1'b0;
            zcomp = 1'b0;
            nzcomp = 1'b0;
            BEQ = 1'b0;
            BNE = 1'b0;
            BLT = 1'b0;
            BGE = 1'b0;
            SW_ID = 1'b0;
            LW_ID = 1'b0;
            shamt_ins = 1'b0;
        end
    endcase
    case(d_code)
        //LDUR
        11'b11010000000: begin
            S_ID = 3'b010;
            Cin_ID = 1'b0;
            Imm_ID = 1'b0;
            SetFlag_ID = 1'b0;
            r_type = 1'b0;
            i_type = 1'b0;
    d_type = 1'b1;
    b_type = 1'b0;
    cb_type = 1'b0;
    iw_type = 1'b0;
    zcomp = 1'b0;
    nzcomp = 1'b0;
    BEQ = 1'b0;
    BNE = 1'b0;
    BLT = 1'b0;
    BGE = 1'b0;
    SW_ID = 1'b0;
    LW_ID = 1'b1;
    shamt_ins = 1'b0;
end
//STUR
11'b11010000001: begin
    S_ID = 3'b010;
    Cin_ID = 1'b0;
    Imm_ID = 1'b0;
    SetFlag_ID = 1'b0;
    r_type = 1'b0;
    i_type = 1'b0;
    d_type = 1'b1;
    b_type = 1'b0;
    cb_type = 1'b0;
    iw_type = 1'b0;
    zcomp = 1'b0;
    nzcomp = 1'b0;
    BEQ = 1'b0;
    BNE = 1'b0;
    BLT = 1'b0;
    BGE = 1'b0;
    SW_ID = 1'b1;
    LW_ID = 1'b0;
    shamt_ins = 1'b0;
end
11'b00000000000: begin
    S_ID = 3'bxxx;
    Cin_ID = 1'bx;
    Imm_ID = 1'b0;
    SetFlag_ID = 1'b0;
    r_type = 1'b0;
    i_type = 1'b0;
```

```
            d_type = 1'b0;
            b_type = 1'b0;
            cb_type = 1'b0;
            iw_type = 1'b0;
            zcomp = 1'b0;
            nzcomp = 1'b0;
            BEQ = 1'b0;
            BNE = 1'b0;
            BLT = 1'b0;
            BGE = 1'b0;
            SW_ID = 1'b0;
            LW_ID = 1'b0;
            shamt_ins = 1'b0;
        end
    endcase
    case(iw_code)
        //MOVZ
        9'b110010101: begin
            S_ID = 3'b101;
            Cin_ID = 1'b0;
            Imm_ID = 1'b1;
            SetFlag_ID = 1'b0;
            r_type = 1'b0;
            i_type = 1'b0;
            d_type = 1'b0;
            b_type = 1'b0;
            cb_type = 1'b0;
            iw_type = 1'b1;
            zcomp = 1'b0;
            nzcomp = 1'b0;
            BEQ = 1'b0;
            BNE = 1'b0;
            BLT = 1'b0;
            BGE = 1'b0;
            SW_ID = 1'b0;
            LW_ID = 1'b0;
            shamt_ins = 1'b0;
        end
        9'b000000000: begin
            S_ID = 3'bxxx;
            Cin_ID = 1'bx;
            Imm_ID = 1'b0;
            SetFlag_ID = 1'b0;
            r_type = 1'b0;
            i_type = 1'b0;
            d_type = 1'b0;
            b_type = 1'b0;
            cb_type = 1'b0;
            iw_type = 1'b0;
            zcomp = 1'b0;
            nzcomp = 1'b0;
            BEQ = 1'b0;
            BNE = 1'b0;
            BLT = 1'b0;
            BGE = 1'b0;
            SW_ID = 1'b0;
            LW_ID = 1'b0;
            shamt_ins = 1'b0;
        end
    endcase
    case(b_code)
        //B
        6'b000011: begin
            S_ID = 3'bxxx;
            Cin_ID = 1'bx;
            Imm_ID = 1'b0;
            SetFlag_ID = 1'b0;
            r_type = 1'b0;
            i_type = 1'b0;
            d_type = 1'b0;
            b_type = 1'b1;
            cb_type = 1'b0;
            iw_type = 1'b0;
            zcomp = 1'b0;
            nzcomp = 1'b0;
            BEQ = 1'b0;
            BNE = 1'b0;
            BLT = 1'b0;
            BGE = 1'b0;
            SW_ID = 1'b0;
            LW_ID = 1'b0;
            shamt_ins = 1'b0;
        end
        6'b000000: begin
            S_ID = 3'bxxx;
            Cin_ID = 1'bx;
            Imm_ID = 1'b0;
            SetFlag_ID = 1'b0;
            r_type = 1'b0;
            i_type = 1'b0;
            d_type = 1'b0;
            b_type = 1'b0;
            cb_type = 1'b0;
            iw_type = 1'b0;
            zcomp = 1'b0;
            nzcomp = 1'b0;
            BEQ = 1'b0;
            BNE = 1'b0;
            BLT = 1'b0;
            BGE = 1'b0;
            SW_ID = 1'b0;
            LW_ID = 1'b0;
            shamt_ins = 1'b0;
        end
    endcase
    case(cb_code)
        //CBZ
        8'b11110100: begin
            S_ID = 3'b010;
            Cin_ID = 1'b0;
            Imm_ID = 1'b1;
            SetFlag_ID = 1'b0;
            r_type = 1'b0;
            i_type = 1'b0;
            d_type = 1'b0;
            b_type = 1'b0;
            cb_type = 1'b1;
            iw_type = 1'b0;
            zcomp = 1'b1;
            nzcomp = 1'b0;
            BEQ = 1'b0;
            BNE = 1'b0;
            BLT = 1'b0;
            BGE = 1'b0;
            SW_ID = 1'b0;
            LW_ID = 1'b0;
            shamt_ins = 1'b0;
        end
        //CBNZ
```

```
8'b11110101: begin
   S_ID = 3'b010;
   Cin_ID = 1'b0;
   Imm_ID = 1'b1;
   SetFlag_ID = 1'b0;
   r_type = 1'b0;
   i_type = 1'b0;
   d_type = 1'b0;
   b_type = 1'b0;
   cb_type = 1'b1;
   iw_type = 1'b0;
   zcomp = 1'b0;
   nzcomp = 1'b1;
   BEQ = 1'b0;
   BNE = 1'b0;
   BLT = 1'b0;
   BGE = 1'b0;
   SW_ID = 1'b0;
   LW_ID = 1'b0;
   shamt_ins = 1'b0;
end
//B.EQ
8'b01110100: begin
   S_ID = 3'b010;
   Cin_ID = 1'b0;
   Imm_ID = 1'b1;
   SetFlag_ID = 1'b0;
   r_type = 1'b0;
   i_type = 1'b0;
   d_type = 1'b0;
   b_type = 1'b0;
   cb_type = 1'b1;
   iw_type = 1'b0;
   zcomp = 1'b0;
   nzcomp = 1'b0;
   BEQ = 1'b1;
   BNE = 1'b0;
   BLT = 1'b0;
   BGE = 1'b0;
   SW_ID = 1'b0;
   LW_ID = 1'b0;
   shamt_ins = 1'b0;
end
```

```
//B.NE
8'b01110101: begin
    S_ID = 3'b010;
    Cin_ID = 1'b0;
    Imm_ID = 1'b1;
    SetFlag_ID = 1'b0;
    r_type = 1'b0;
    i_type = 1'b0;
    d_type = 1'b0;
    b_type = 1'b0;
    cb_type = 1'b1;
    iw_type = 1'b0;
    zcomp = 1'b0;
    nzcomp = 1'b0;
    BEQ = 1'b0;
    BNE = 1'b1;
    BLT = 1'b0;
    BGE = 1'b0;
    SW_ID = 1'b0;
    LW_ID = 1'b0;
    shamt_ins = 1'b0;
end
//B.LT (Signed)
8'b01110110: begin
    S_ID = 3'b010;
    Cin_ID = 1'b0;
    Imm_ID = 1'b1;
    SetFlag_ID = 1'b0;
    r_type = 1'b0;
    i_type = 1'b0;
    d_type = 1'b0;
    b_type = 1'b0;
    cb_type = 1'b1;
    iw_type = 1'b0;
    zcomp = 1'b0;
    nzcomp = 1'b0;
    BEQ = 1'b0;
    BNE = 1'b0;
    BLT = 1'b1;
    BGE = 1'b0;
    SW_ID = 1'b0;
    LW_ID = 1'b0;
    shamt_ins = 1'b0;
```

```
end
//B.GE (Signed)
8'b01110111: begin
    S_ID = 3'b010;
    Cin_ID = 1'b0;
    Imm_ID = 1'b1;
    SetFlag_ID = 1'b0;
    r_type = 1'b0;
    i_type = 1'b0;
    d_type = 1'b0;
    b_type = 1'b0;
    cb_type = 1'b1;
    iw_type = 1'b0;
    zcomp = 1'b0;
    nzcomp = 1'b0;
    BEQ = 1'b0;
    BNE = 1'b0;
    BLT = 1'b0;
    BGE = 1'b1;
    SW_ID = 1'b0;
    LW_ID = 1'b0;
    shamt_ins = 1'b0;
end
8'b00000000: begin
    S_ID = 3'bxxx;
    Cin_ID = 1'bx;
    Imm_ID = 1'b0;
    SetFlag_ID = 1'b0;
    r_type = 1'b0;
    i_type = 1'b0;
    d_type = 1'b0;
    b_type = 1'b0;
    cb_type = 1'b0;
    iw_type = 1'b0;
    zcomp = 1'b0;
    nzcomp = 1'b0;
    BEQ = 1'b0;
    BNE = 1'b0;
    BLT = 1'b0;
    BGE = 1'b0;
    SW_ID = 1'b0;
    LW_ID = 1'b0;
    shamt_ins = 1'b0;
```

```
            end
          endcase

        end


      endmodule
```

V. Test Bench

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Aryan Mehrotra
//
// Module Name: cpu5arm_AryanM_tb
// Description: Test bench for ARM LEGV8 CPU
// Project Name: Final Project
//////////////////////////////////////////////////////////////////////////////////


 module cpu5arm_AryanM_tb();
   reg  [31:0] instrbus;
   reg  [31:0] instrbusin[0:209];
   wire [63:0] iaddrbus, daddrbus;
   reg  [63:0] iaddrbusout[0:209], daddrbusout[0:209];
   wire [63:0] databus;
   reg  [63:0] databusk, databusin[0:209], databusout[0:209];
   reg         clk, reset;
   reg         clkd;

   reg [63:0] dontcare, activez;
   reg [24*8:1] iname[0:209];
   integer error, k, ntests;

   //R-Format
   parameter ADD   = 11'b00101000000;
   parameter ADDS  = 11'b00101000001;
   parameter AND   = 11'b00101000010;
   parameter ANDS  = 11'b00101000011;
   parameter EOR   = 11'b00101000100;
   parameter ENOR  = 11'b00101000101;
   parameter LSL   = 11'b00101000110;
   parameter LSR   = 11'b00101000111;
   parameter ORR   = 11'b00101001000;
   parameter SUB   = 11'b00101001001;
parameter SUBS  = 11'b00101001010;
//I-Format
parameter ADDI  = 10'b1000100000;
parameter ADDIS = 10'b1000100001;
parameter ANDI  = 10'b1000100010;
parameter ANDIS = 10'b1000100011;
parameter EORI  = 10'b1000100100;
parameter ENORI = 10'b1000100101;
parameter ORRI  = 10'b1000100110;
parameter SUBI  = 10'b1000100111;
parameter SUBIS = 10'b1000101000;
//D-Format
parameter LDUR  = 11'b11010000000;
parameter STUR  = 11'b11010000001;
//IM-Format
parameter MOVZ  = 9'b110010101;
//B-Format
parameter B     = 6'b000011;
//CB-Format
parameter CBZ   = 8'b11110100;
parameter CBNZ  = 8'b11110101;
parameter BEQ   = 8'b01110100;
parameter BNE   = 8'b01110101;
parameter BLT   = 8'b01110110;
parameter BGE   = 8'b01110111;

// Registers
parameter X0    = 5'b00000;
parameter X1    = 5'b00001;
parameter X2    = 5'b00010;
parameter X3    = 5'b00011;
parameter X4    = 5'b00100;
parameter X5    = 5'b00101;
parameter X6    = 5'b00110;
parameter X7    = 5'b00111;
parameter X8    = 5'b01000;
parameter X9    = 5'b01001;
parameter X10   = 5'b01010;
parameter X11   = 5'b01011;
parameter X12   = 5'b01100;
parameter X13   = 5'b01101;
parameter X14   = 5'b01110;
parameter X15   = 5'b01111;
parameter X16   = 5'b10000;
parameter X17   = 5'b10001;
parameter X18   = 5'b10010;
parameter X19   = 5'b10011;
parameter X20   = 5'b10100;
parameter X21   = 5'b10101;
parameter X22   = 5'b10110;
parameter X23   = 5'b10111;
parameter X24   = 5'b11000;
parameter X25   = 5'b11001;
parameter X26   = 5'b11010;
parameter X27   = 5'b11011;
parameter X28   = 5'b11100;
parameter X29   = 5'b11101;
parameter X30   = 5'b11110;
parameter XZR   = 5'b11111;

cpu5arm dut(.ibus(instrbus),
          .clk(clk),
          .reset(reset),
          .iaddrbus(iaddrbus),
          .daddrbus(daddrbus),
          .databus(databus));

initial begin
// This test file runs the following program.
iname[0] =  "ADDI   X5, XZR, #-1";
iname[1] =  "ADDI   X8, XZR, #15";
iname[2] =  "ADDI   X10, XZR, #20";
iname[3] =  "LDUR   X11, [XZR, #60]";
iname[4] =  "LDUR   X12, [XZR, #20]";
iname[5] =  "EORI   X13, XZR, A8C";
iname[6] =  "STUR   X5, [X10, #2A]";
iname[7] =  "STUR   X8, [X5, #0]";
iname[8] =  "ADD    X0, X10, X11";
iname[9] =  "ORR    X1, X5, X8";
iname[10] = "AND    X2, X11, X12";
iname[11] = "ORRI   X3, X13, F0F";
iname[12] = "SUB    X4, X11, X10";
iname[13] = "SUB    X6, X12, X0";
iname[14] = "SUB    X9, X0, X10";
iname[15] = "EOR    X7, X13, X3";
iname[16] = "ENOR   X16, X3, X2";
```

```
iname[17] = "EOR    X20, X5, X10";
iname[18] = "ORRI   X19, X3, 902";
iname[19] = "ORRI   X18, X6, 000";
iname[20] = "AND    X17, X3, X4";
iname[21] = "ANDI   X21, X6, 672";
iname[22] = "ANDI   X22, X7, AAB";
iname[23] = "ADDI   X23, X13, CDD";
iname[24] = "SUBI   X30, X12, 010";
iname[25] = "ADDI   X29, X0, 439";
iname[26] = "SUBI   X28, X16, C10";
iname[27] = "EORI   X27, X9, F5B";
iname[28] = "ENORI  X26, X7, FED";
iname[29] = "EORI   X25, X20, 9B6";
iname[30] = "ENORI  X24, X22, 6A2";
iname[31] = "ADD    X14, X6, X7";
iname[32] = "ADD    X15, X13, X12";
iname[33] = "ADD    X9, X29, X16";
iname[34] = "AND    X1, X20, X23";
iname[35] = "AND    X2, X4, X18";
iname[36] = "AND    X4, X24, X27";
iname[37] = "EOR    X6, X3, X10";
iname[38] = "EOR    X10, X14, X19";
iname[39] = "ENOR   X11, X21, X25";
iname[40] = "ENOR   X12, X20, X27";
iname[41] = "EOR    X5, X7, X22";
iname[42] = "ENOR   X13, X23, X30";
iname[43] = "LSL    X3, X7, #5";
iname[44] = "LSL    X7, X14, #0";
iname[45] = "LSL    X14, X17, #2";
iname[46] = "LSR    X15, X15, #8";
iname[47] = "LSR    X18, X6, #0";
iname[48] = "LSL    X16, X4, #7";
iname[49] = "LSR    X19, X3, #2";
iname[50] = "LSL    X17, X1, #4";
iname[51] = "LSR    X20, X9, #0";
iname[52] = "ORR    X22, X26, X4";
iname[53] = "ORR    X23, X5, X8";
iname[54] = "ORR    X24, X9, X10";
iname[55] = "SUBI   X25, X1, 044";
iname[56] = "SUBI   X26, X9, FCB";
iname[57] = "ENORI  X21, X14, #56C";
iname[58] = "ENORI  X27, X11, #71A";
iname[59] = "EORI   X28, X0, #BAE";
```

```
iname[60] = "LDUR   X29, [XZR, #F4]";
iname[61] = "LDUR   X30, [X14, #78]";
iname[62] = "LDUR   X0, [XZR, #FA]";
iname[63] = "STUR   XZR, [XZR, #7C]";
iname[64] = "STUR   XZR, [XZR, #FF]";
iname[65] = "STUR   XZR, [XZR, #9D]";
iname[66] = "ANDI   X8, X7, #87F";
iname[67] = "ANDI   X3, X19, #3E5";
iname[68] = "LSR    X6, X7, #1";
iname[69] = "ADD    X7, X14, X9";
iname[70] = "SUB    X8, X24, X25";
iname[71] = "SUB    X9, X13, X18";
iname[72] = "ENOR   X10, X30, X29";
iname[73] = "ORR    X11, X30, X29";
iname[74] = "EORI   X12, X24, #0FF";
iname[75] = "ENORI  X13, X23, #FF0";
iname[76] = "ORRI   X14, X18, #100";
iname[77] = "ORRI   X15, X20, #34B";
iname[78] = "SUBI   X16, X19, #810";
iname[79] = "ANDI   X17, X27, #FFF";
iname[80] = "ADD    X18, X0, X1";
iname[81] = "ADD    X19, X2, X3";
iname[82] = "AND    X20, X30, X29";
iname[83] = "AND    X21, X29, X5";
iname[84] = "EOR    X22, X7, X8";
iname[85] = "EOR    X23, X9, X9";
iname[86] = "ENOR   X24, X10, X15";
iname[87] = "ENOR   X25, X11, X12";
iname[88] = "LSL    X26, X6, #2";
iname[89] = "LSL    X27, X5, #4";
iname[90] = "LSR    X28, X0, #6";
iname[91] = "LSR    X29, X2, #10";
iname[92] = "ORR    X30, X14, X16";
iname[93] = "ORR    X0, X10, X12";
iname[94] = "SUB    X1, X16, X17";
iname[95] = "SUB    X2, X18, X19";
iname[96] = "ADDI   X3, X9, #0A2";
iname[97] = "ADDI   X4, X10, #19E";
iname[98] = "EORI   X5, X11, #908";
iname[99] = "EORI   X6, X25, #ABE";
iname[100] = "ENORI X7, X26, #1BC";
iname[101] = "ENORI X8, X27, #234";
iname[102] = "ORRI  X9, X28, #FDC";
```

```
iname[103] = "ORRI  X10, X29, #024";
iname[104] = "SUBI  X11, X30, #BBA";
iname[105] = "SUBI  X12, X16, #987";
iname[106] = "LDUR  X13, [XZR, #5A]";
iname[107] = "LDUR  X14, [XZR, #FD]";
iname[108] = "STUR  X15, [XZR, #9B]";
iname[109] = "STUR  X16, [XZR, #78]";
iname[110] = "ANDI  X17, X5, #984";
iname[111] = "ANDI  X18, X6, #FEE";
iname[112] = "ADDS  X0, X21, X20";
iname[113] = "BEQ   #10";       //Branch not taken
iname[114] = "NOP";
iname[115] = "ADD   X1, X23, X23";
iname[116] = "ADDS  X2, X15, X28";
iname[117] = "BLT   #2";        //Branch taken
iname[118] = "NOP";
iname[119] = "NOP";
iname[120] = "NOP";
iname[121] = "ADDS  X4, X28, X30";
iname[122] = "BGE #5";          //Branch taken
iname[123] = "NOP";
iname[124] = "NOP";
iname[125] = "NOP";
iname[126] = "B #20";           //Branch taken
iname[127] = "NOP";
iname[128] = "NOP";
iname[129] = "NOP";
iname[130] = "ANDS  X4, X19, X22";
iname[131] = "BLT   #20";       //Branch not taken
iname[132] = "NOP";
iname[133] = "CBZ   X23, #10";  //Branch taken
iname[134] = "NOP";
iname[135] = "CBNZ  X27, #15";  //Branch taken
iname[136] = "NOP";
iname[137] = "CBNZ  X23, #80";  //Branch not taken
iname[138] = "NOP";
iname[139] = "B     #10";       //Branch taken
iname[140] = "NOP";
iname[141] = "NOP";
iname[142] = "SUBS  X5, X20, X21";
iname[143] = "BNE   #10";       //Branch taken
iname[144] = "NOP";
iname[145] = "NOP";
```

```
iname[146] = "NOP";
iname[147] = "CBZ   X15, #20";  //Branch not taken
iname[148] = "NOP";
iname[149] = "CBZ   X30, #0";   //Branch not taken
iname[150] = "NOP";
iname[151] = "CBNZ  X20, #0";   //Branch taken
iname[152] = "NOP";
iname[153] = "NOP";
iname[154] = "ADDIS X15, X18, #9A";
iname[155] = "BEQ   #80";       //Branch not taken
iname[156] = "NOP";
iname[157] = "SUB   X16, X6, X10";
iname[158] = "ADDIS X19, X5, #8AC";
iname[159] = "BLT   #20";       //Branch not taken
iname[160] = "NOP";
iname[161] = "ADDI  X20, X8, #09E";
iname[162] = "ADDIS X21, X12, #FEA";
iname[163] = "CBZ   X1, #10";   //Branch taken
iname[164] = "NOP ";
iname[165] = "NOP ";
iname[166] = "B     #0";        //Branch taken
iname[167] = "NOP";
iname[168] = "CBNZ  X1, #10";   //Branch not taken
iname[169] = "NOP";
iname[170] = "ANDIS X22, X6, #ABC";
iname[171] = "BEQ   #1";        //Branch not taken
iname[172] = "NOP ";
iname[173] = "ANDIS X23, X9, #765";
iname[174] = "BNE   #1";        //Branch taken
iname[175] = "NOP";
iname[176] = "NOP";
iname[177] = "NOP";
iname[178] = "CBNZ  X1, #12";    //Branch not taken
iname[179] = "NOP";
iname[180] = "MOVZ  X23, #3A3";
iname[181] = "CBZ   X16, #5";    //Branch not taken
iname[182] = "NOP";
iname[183] = "MOVZ  X24, #0";
iname[184] = "SUBIS X25, X13, #AAA";
iname[185] = "CBNZ  X1, #9";     //Branch not taken
iname[186] = "NOP";
iname[187] = "SUBIS X26, X8, #0";
iname[188] = "SUBIS X27, X3, #EFE";
```

```
iname[189] = "BNE   #14";       //Branch taken
iname[190] = "NOP";
iname[191] = "NOP";
iname[192] = "NOP";
iname[193] = "MOVZ  X28, #9AE";
iname[194] = "ADD   X29, X0, X1";
iname[195] = "AND   X30, X3, X2";
iname[196] = "SUB   X5, X2, X7";
iname[197] = "ORR   X0, X10, X11";
iname[198] = "ANDI  X1, X1, X4";
iname[199] = "ADDI  X2, X20, #90A";
iname[200] = "SUBI  X4, X3, #AFF";
iname[201] = "ORRI  X6, X8, #00A";
iname[202] = "EOR   X7, X17, X18";
iname[203] = "EORI  X8, X22, #00C";
iname[204] = "ADD   X9, X27, X28";
iname[205] = "SUB   X10, X27, X25";
iname[206] = "AND   X11, X20, X21";
iname[207] = "EOR   X12, X26, X15";
iname[208] = "ORR   X13, X2, X16";
iname[209] = "ADDI  X14, X28, #E45";


//* ADDI   X5, XZR, #-1
iaddrbusout[0] = 64'h0000000000000000;
//          opcode ALU_imm  Source Dest
instrbusin[0]={ADDI, 12'hFFF, XZR, X5};

daddrbusout[0] = dontcare;
databusin[0] = activez;
databusout[0] = dontcare;

//ADDI   X8, XZR, #15
iaddrbusout[1] = 64'h0000000000000004;
//          opcode ALU_imm  Source Dest
instrbusin[1]={ADDI, 12'h015, XZR, X8};

daddrbusout[1] = dontcare;
databusin[1] = activez;
databusout[1] = dontcare;

//ADDI   X10, XZR, #20
iaddrbusout[2] = 64'h0000000000000008;
```

```
//          opcode ALU_imm  Source Dest
instrbusin[2]={ADDI, 12'h020, XZR, X10};

daddrbusout[2] = dontcare;
databusin[2] = activez;
databusout[2] = dontcare;

//LDUR   X11, [XZR, #60]
iaddrbusout[3] = 64'h000000000000000C;
//          opcode offset         OP   Source Dest
instrbusin[3]={LDUR, 9'b001100000, 2'b00, XZR, X11};

daddrbusout[3] = 64'h0000000000000060;
databusin[3] = 64'hAAAAAAAAAAAAAAAA;
databusout[3] = dontcare;

//LDUR   X12, [XZR, #20]
iaddrbusout[4] = 64'h0000000000000010;
//          opcode offset     OP   Source Dest
instrbusin[4]={LDUR, 9'b000100000, 2'b00, XZR, X12};

daddrbusout[4] = 64'h0000000000000020;
databusin[4] = 64'hDDDDDDDDDDDDDDDD;
databusout[4] = dontcare;

//EORI   X13, XZR, A8C
iaddrbusout[5] = 64'h0000000000000014;
//          opcode ALU_imm  Source Dest
instrbusin[5]={EORI, 12'hA8C, XZR, X10};

daddrbusout[5] = 64'h0000000000000A8C;
databusin[5] = activez;
databusout[5] = dontcare;

//STUR   X5, [X10, #2A]
iaddrbusout[6] = 64'h0000000000000018;
//          opcode offset     OP   Source Dest
instrbusin[6]={STUR, 9'b000101010, 2'b00, X10, X5};

daddrbusout[6] = 64'h000000000000004A;
databusin[6] = activez;
databusout[6] = 64'h0000000000000FFF;
```

```
//STUR   X8, [X5, #0]
iaddrbusout[7] = 64'h000000000000001C;
//          opcode offset     OP   Source Dest
instrbusin[7]={STUR, 9'b000000000, 2'b00, X5, X8};

daddrbusout[7] = 64'h0000000000000FFF;
databusin[7] = activez;
databusout[7] = 64'h0000000000000015;

//ADD    X0, X10, X11
iaddrbusout[8] = 64'h0000000000000020;
//            opcode Source2 Shamt Source1 Dest
instrbusin[8] ={ADD, X11, 6'b000000, X10, X0};

daddrbusout[8] = dontcare;
databusin[8] = activez;
databusout[8] = dontcare;

//ORR    X1, X5, X8
iaddrbusout[9] = 64'h0000000000000024;
//            opcode Source2 Shamt Source1 Dest
instrbusin[9] ={ORR, X8, 6'b000000, X5, X1};

daddrbusout[9] = dontcare;
databusin[9] = activez;
databusout[9] = dontcare;

//AND    X2, X11, X12
iaddrbusout[10] = 64'h0000000000000028;
//            opcode Source2 Shamt Source1 Dest
instrbusin[10] ={AND, X12, 6'b000000, X2, X11};

daddrbusout[10] = dontcare;
databusin[10] = activez;
databusout[10] = dontcare;

//ORRI   X3, X13, F0F
iaddrbusout[11] = 64'h000000000000002C;
//          opcode ALU_imm  Source Dest
instrbusin[11]={ORRI, 12'hF0F, X13, X3};

daddrbusout[11] = dontcare;
databusin[11] = activez;
```

```
databusout[11] = dontcare;

//SUB    X4, X11, X10
iaddrbusout[12] = 64'h0000000000000030;
//            opcode Source2 Shamt Source1 Dest
instrbusin[12] ={SUB, X10, 6'b000000, X12, X4};

daddrbusout[12] = dontcare;
databusin[12] = activez;
databusout[12] = dontcare;

//SUB    X6, X12, X0
iaddrbusout[13] = 64'h0000000000000034;
//            opcode Source2 Shamt Source1 Dest
instrbusin[13] ={SUB, X0, 6'b000000, X12, X6};

daddrbusout[13] = dontcare;
databusin[13] = activez;
databusout[13] = dontcare;

//SUB    X9, X0, X10
iaddrbusout[14] = 64'h0000000000000038;
//            opcode Source2 Shamt Source1 Dest
instrbusin[14] ={SUB, X10, 6'b000000, X0, X9};

daddrbusout[14] = dontcare;
databusin[14] = activez;
databusout[14] = dontcare;

//EOR    X7, X13, X3
iaddrbusout[15] = 64'h000000000000003C;
//            opcode Source2 Shamt Source1 Dest
instrbusin[15] ={EOR, X3, 6'b000000, X13, X7};

daddrbusout[15] = dontcare;
databusin[15] = activez;
databusout[15] = dontcare;

//ENOR   X16, X3, X2
iaddrbusout[16] = 64'h0000000000000040;
//            opcode Source2 Shamt Source1 Dest
instrbusin[16] ={ENOR, X2, 6'b000000, X3, X16};
```

```
daddrbusout[16] = dontcare;
databusin[16] = activez;
databusout[16] = dontcare;

//EOR    X20, X5, X10
iaddrbusout[17] = 64'h0000000000000044;
//            opcode Source2 Shamt Source1 Dest
instrbusin[17] ={EOR, X10, 6'b000000, X5, X20};

daddrbusout[17] = dontcare;
databusin[17] = activez;
databusout[17] = dontcare;

//ORRI   X19, X3, 902
iaddrbusout[18] = 64'h0000000000000048;
//          opcode ALU_imm  Source Dest
instrbusin[18]={ORRI, 12'h902, X3, X19};

daddrbusout[18] = dontcare;
databusin[18] = activez;
databusout[18] = dontcare;

//ORRI   X18, X6, 000
iaddrbusout[19] = 64'h000000000000004C;
//          opcode ALU_imm  Source Dest
instrbusin[19]={ORRI, 12'h000, X6, X18};

daddrbusout[19] = dontcare;
databusin[19] = activez;
databusout[19] = dontcare;

//AND    X17, X3, X4
iaddrbusout[20] = 64'h0000000000000050;
//            opcode Source2 Shamt Source1 Dest
instrbusin[20] ={AND, X4, 6'b000000, X3, X17};

daddrbusout[20] = dontcare;
databusin[20] = activez;
databusout[20] = dontcare;

//ANDI   X21, X6, 672
iaddrbusout[21] = 64'h0000000000000054;
//          opcode ALU_imm  Source Dest
```

```
instrbusin[21]={ANDI, 12'h672, X6, X21};

daddrbusout[21] = dontcare;
databusin[21] = activez;
databusout[21] = dontcare;

//ANDI   X22, X7, AAB
iaddrbusout[22] = 64'h0000000000000058;
//          opcode ALU_imm  Source Dest
instrbusin[22]={ANDI, 12'hAAB, X7, X22};

daddrbusout[22] = dontcare;
databusin[22] = activez;
databusout[22] = dontcare;

//ADDI   X23, X13, CDD
iaddrbusout[23] = 64'h000000000000005C;
//          opcode ALU_imm  Source Dest
instrbusin[23]={ADDI, 12'hCDD, X13, X23};

daddrbusout[23] = dontcare;
databusin[23] = activez;
databusout[23] = dontcare;

//SUBI   X30, X12, 010
iaddrbusout[24] = 64'h0000000000000060;
//          opcode ALU_imm  Source Dest
instrbusin[24]={SUBI, 12'h010, X12, X30};

daddrbusout[24] = dontcare;
databusin[24] = activez;
databusout[24] = dontcare;

//ADDI   X29, X0, 439
iaddrbusout[25] = 64'h0000000000000064;
//          opcode ALU_imm  Source Dest
instrbusin[25]={ADDI, 12'h439, X0, X29};

daddrbusout[25] = dontcare;
databusin[25] = activez;
databusout[25] = dontcare;

//SUBI   X28, X16, C10
```

```
iaddrbusout[26] = 64'h0000000000000068;
//          opcode ALU_imm  Source Dest
instrbusin[26]={SUBI, 12'hC10, X16, X28};

daddrbusout[26] = dontcare;
databusin[26] = activez;
databusout[26] = dontcare;

//EORI   X27, X9, F5B
iaddrbusout[27] = 64'h000000000000006C;
//          opcode ALU_imm  Source Dest
instrbusin[27]={EORI, 12'hF5B, X9, X27};

daddrbusout[27] = dontcare;
databusin[27] = activez;
databusout[27] = dontcare;

//ENORI  X26, X7, FED
iaddrbusout[28] = 64'h0000000000000070;
//          opcode ALU_imm  Source Dest
instrbusin[28]={ENORI, 12'hFED, X7, X26};

daddrbusout[28] = dontcare;
databusin[28] = activez;
databusout[28] = dontcare;

//EORI   X25, X20, 9B6
iaddrbusout[29] = 64'h0000000000000074;
//          opcode ALU_imm  Source Dest
instrbusin[29]={EORI, 12'h9B6, X20, X25};

daddrbusout[29] = dontcare;
databusin[29] = activez;
databusout[29] = dontcare;

//ENORI  X24, X22, 6A2
iaddrbusout[30] = 64'h0000000000000078;
//          opcode ALU_imm  Source Dest
instrbusin[30]={ENORI, 12'h6A2, X22, X24};

daddrbusout[30] = dontcare;
databusin[30] = activez;
databusout[30] = dontcare;
```

```
//ADD    X14, X6, X7
iaddrbusout[31] = 64'h000000000000007C;
//            opcode Source2 Shamt Source1 Dest
instrbusin[31] ={ADD, X7, 6'b000000, X6, X14};

daddrbusout[31] = dontcare;
databusin[31] = activez;
databusout[31] = dontcare;

//ADD    X15, X13, X12
iaddrbusout[32] = 64'h0000000000000080;
//            opcode Source2 Shamt Source1 Dest
instrbusin[32] ={ADD, X12, 6'b000000, X13, X15};

daddrbusout[32] = dontcare;
databusin[32] = activez;
databusout[32] = dontcare;

//ADD    X9, X29, X16
iaddrbusout[33] = 64'h0000000000000084;
//            opcode Source2 Shamt Source1 Dest
instrbusin[33] ={ADD, X16, 6'b000000, X29, X9};

daddrbusout[33] = dontcare;
databusin[33] = activez;
databusout[33] = dontcare;

//AND    X1, X20, X23
iaddrbusout[34] = 64'h0000000000000088;
//            opcode Source2 Shamt Source1 Dest
instrbusin[34] ={AND, X23, 6'b000000, X20, X1};

daddrbusout[34] = dontcare;
databusin[34] = activez;
databusout[34] = dontcare;

//AND    X2, X4, X18
iaddrbusout[35] = 64'h000000000000008C;
//            opcode Source2 Shamt Source1 Dest
instrbusin[35] ={AND, X18, 6'b000000, X4, X2};

daddrbusout[35] = dontcare;
```

```
databusin[35] = activez;
databusout[35] = dontcare;

//AND    X4, X24, X27
iaddrbusout[36] = 64'h0000000000000090;
//            opcode Source2 Shamt Source1 Dest
instrbusin[36] ={AND, X27, 6'b000000, X24, X4};

daddrbusout[36] = dontcare;
databusin[36] = activez;
databusout[36] = dontcare;

//EOR    X6, X3, X10
iaddrbusout[37] = 64'h0000000000000094;
//            opcode Source2 Shamt Source1 Dest
instrbusin[37] ={EOR, X10, 6'b000000, X3, X6};

daddrbusout[37] = dontcare;
databusin[37] = activez;
databusout[37] = dontcare;

//EOR    X10, X14, X19
iaddrbusout[38] = 64'h0000000000000098;
//            opcode Source2 Shamt Source1 Dest
instrbusin[38] ={EOR, X19, 6'b000000, X14, X10};

daddrbusout[38] = dontcare;
databusin[38] = activez;
databusout[38] = dontcare;

//ENOR   X11, X21, X25
iaddrbusout[39] = 64'h000000000000009C;
//            opcode Source2 Shamt Source1 Dest
instrbusin[39] ={ENOR, X25, 6'b000000, X21, X11};

daddrbusout[39] = dontcare;
databusin[39] = activez;
databusout[39] = dontcare;

//ENOR   X12, X20, X27
iaddrbusout[40] = 64'h00000000000000A0;
//            opcode Source2 Shamt Source1 Dest
instrbusin[40] ={ENOR, X27, 6'b000000, X20, X12};
```

```
daddrbusout[40] = dontcare;
databusin[40] = activez;
databusout[40] = dontcare;

//EOR    X5, X7, X22
iaddrbusout[41] = 64'h00000000000000A4;
//            opcode Source2 Shamt Source1 Dest
instrbusin[41] ={EOR, X22, 6'b000000, X7, X5};

daddrbusout[41] = dontcare;
databusin[41] = activez;
databusout[41] = dontcare;

//ENOR   X13, X23, X30
iaddrbusout[42] = 64'h00000000000000A8;
//            opcode Source2 Shamt Source1 Dest
instrbusin[42] ={ENOR, X30, 6'b000000, X23, X13};

daddrbusout[42] = dontcare;
databusin[42] = activez;
databusout[42] = dontcare;

//LSL    X3, X7, #5
iaddrbusout[43] = 64'h00000000000000AC;
//            opcode Source2 Shamt Source1 Dest
instrbusin[43] ={LSL, XZR, 6'b000101, X7, X3};

daddrbusout[43] = dontcare;
databusin[43] = activez;
databusout[43] = dontcare;

//LSL    X7, X14, #0
iaddrbusout[44] = 64'h00000000000000B0;
//            opcode Source2 Shamt Source1 Dest
instrbusin[44] ={LSL, XZR, 6'b000000, X14, X7};

daddrbusout[44] = dontcare;
databusin[44] = activez;
databusout[44] = dontcare;

//LSL    X14, X17, #2
iaddrbusout[45] = 64'h00000000000000B4;
```

```
//		opcode Source2 Shamt Source1 Dest
instrbusin[45] ={LSL, XZR, 6'b000010, X17, X14};

daddrbusout[45] = dontcare;
databusin[45] = activez;
databusout[45] = dontcare;

//LSR	X15, X15, #8
iaddrbusout[46] = 64'h00000000000000B8;
//		opcode Source2 Shamt Source1 Dest
instrbusin[8] ={LSR, XZR, 6'b001000, X15, X15};

daddrbusout[46] = dontcare;
databusin[46] = activez;
databusout[46] = dontcare;

//LSR	X18, X6, #0
iaddrbusout[47] = 64'h00000000000000BC;
//		opcode Source2 Shamt Source1 Dest
instrbusin[47] ={LSR, XZR, 6'b000000, X6, X18};

daddrbusout[47] = dontcare;
databusin[47] = activez;
databusout[47] = dontcare;

//LSL	X16, X4, #7
iaddrbusout[48] = 64'h00000000000000C0;
//		opcode Source2 Shamt Source1 Dest
instrbusin[48] ={LSL, XZR, 6'b000111, X4, X16};

daddrbusout[48] = dontcare;
databusin[48] = activez;
databusout[48] = dontcare;

//LSR	X19, X3, #2
iaddrbusout[49] = 64'h00000000000000C4;
//		opcode Source2 Shamt Source1 Dest
instrbusin[49] ={LSR, XZR, 6'b000010, X3, X19};

daddrbusout[49] = dontcare;
databusin[49] = activez;
databusout[49] = dontcare;
```

```
//LSL    X17, X1, #4
iaddrbusout[50] = 64'h00000000000000C8;
//           opcode Source2 Shamt Source1 Dest
instrbusin[50] ={LSL, XZR, 6'b000100, X1, X17};

daddrbusout[50] = dontcare;
databusin[50] = activez;
databusout[50] = dontcare;

//LSR    X20, X9, #0
iaddrbusout[51] = 64'h00000000000000CC;
//           opcode Source2 Shamt Source1 Dest
instrbusin[51] ={LSR, XZR, 6'b000000, X9, X20};

daddrbusout[51] = dontcare;
databusin[51] = activez;
databusout[51] = dontcare;

//ORR    X22, X26, X4
iaddrbusout[52] = 64'h00000000000000D0;
//           opcode Source2 Shamt Source1 Dest
instrbusin[52] ={ORR, X4, 6'b000000, X26, X22};

daddrbusout[52] = dontcare;
databusin[52] = activez;
databusout[52] = dontcare;

//ORR    X23, X5, X8
iaddrbusout[53] = 64'h00000000000000D4;
//           opcode Source2 Shamt Source1 Dest
instrbusin[53] ={ORR, X8, 6'b000000, X5, X23};

daddrbusout[53] = dontcare;
databusin[53] = activez;
databusout[53] = dontcare;

//ORR    X24, X9, X10
iaddrbusout[54] = 64'h00000000000000D8;
//           opcode Source2 Shamt Source1 Dest
instrbusin[54] ={ORR, X10, 6'b000000, X9, X24};

daddrbusout[54] = dontcare;
databusin[54] = activez;
```

```
databusout[54] = dontcare;

//SUBI   X25, X4, 044
iaddrbusout[55] = 64'h00000000000000DC;
//          opcode ALU_imm  Source Dest
instrbusin[55]={SUBI, 12'h044, X4, X25};

daddrbusout[55] = dontcare;
databusin[55] = activez;
databusout[55] = dontcare;

//SUBI   X26, X9, FCB
iaddrbusout[56] = 64'h00000000000000E0;
//          opcode ALU_imm  Source Dest
instrbusin[56]={SUBI, 12'hFCB, X9, X26};

daddrbusout[56] = dontcare;
databusin[56] = activez;
databusout[56] = dontcare;

//ENORI  X21, X14, #56C
iaddrbusout[57] = 64'h00000000000000E4;
//          opcode ALU_imm  Source Dest
instrbusin[57]={ENORI, 12'h56C, X14, X21};

daddrbusout[57] = dontcare;
databusin[57] = activez;
databusout[57] = dontcare;

//ENORI  X27, X11, #71A
iaddrbusout[58] = 64'h00000000000000E8;
//          opcode ALU_imm  Source Dest
instrbusin[58]={ENORI, 12'h71A, X11, X27};

daddrbusout[58] = dontcare;
databusin[58] = activez;
databusout[58] = dontcare;

//EORI   X28, X0, #BAE
iaddrbusout[59] = 64'h00000000000000EC;
//          opcode ALU_imm  Source Dest
instrbusin[59]={EORI, 12'hBAE, X0, X28};
```

```
daddrbusout[59] = dontcare;
databusin[59] = activez;
databusout[59] = dontcare;

//LDUR   X29, [XZR, #F4]
iaddrbusout[60] = 64'h00000000000000F0;
//          opcode offset         OP   Source Dest
instrbusin[60]={LDUR, 9'b011110100, 2'b00, XZR, X29};

daddrbusout[60] = 64'h00000000000000F4;
databusin[60] = 64'h0000FFFF0000EEEE;
databusout[60] = dontcare;

//LDUR   X30, [XZR, #78]
iaddrbusout[61] = 64'h00000000000000F4;
//          opcode offset         OP   Source Dest
instrbusin[61]={LDUR, 9'b001111000, 2'b00, XZR, X30};

daddrbusout[61] = 64'h0000000000000078;
databusin[61] = 64'h000012349ABC5678;
databusout[61] = dontcare;

//LDUR   X0, [XZR, #FA]
iaddrbusout[62] = 64'h00000000000000F8;
//          opcode offset         OP   Source Dest
instrbusin[62]={LDUR, 9'b011111010, 2'b00, XZR, X0};

daddrbusout[62] = 64'h00000000000000FA;
databusin[62] = 64'h000000000000FEDC;
databusout[62] = dontcare;

//STUR   XZR, [XZR, #7C]
iaddrbusout[63] = 64'h00000000000000FC;
//          opcode offset         OP   Source Dest
instrbusin[63]={STUR, 9'b001111100, 2'b00, XZR, XZR};

daddrbusout[63] = 64'h000000000000007C;
databusin[63] = activez;
databusout[63] = 64'h0000000000000000;

//STUR   XZR, [XZR, #FF]
iaddrbusout[64] = 64'h0000000000000100;
//          opcode offset         OP   Source Dest
```

```
instrbusin[64]={STUR, 9'b011111111, 2'b00, XZR, XZR};

daddrbusout[64] = 64'h00000000000000FF;
databusin[64] = activez;
databusout[64] = 64'h0000000000000000;

//STUR   XZR, [XZR, #9D]
iaddrbusout[65] = 64'h0000000000000104;
//          opcode offset         OP   Source Dest
instrbusin[65]={STUR, 9'b010011101, 2'b00, XZR, XZR};

daddrbusout[65] = dontcare;
databusin[65] = activez;
databusout[65] = activez;

//ANDI   X8, X7, #87F
iaddrbusout[66] = 64'h0000000000000108;
//          opcode ALU_imm  Source Dest
instrbusin[66]={ANDI, 12'h87F, X7, X8};

daddrbusout[66] = dontcare;
databusin[66] = activez;
databusout[66] = dontcare;

//ANDI   X3, X19, #3E5
iaddrbusout[67] = 64'h000000000000010C;
//          opcode ALU_imm  Source Dest
instrbusin[67]={ANDI, 12'h3E5, X19, X3};

daddrbusout[67] = dontcare;
databusin[67] = activez;
databusout[67] = dontcare;

//LSR    X6, X7, #1
iaddrbusout[68] = 64'h0000000000000110;
//           opcode Source2 Shamt Source1 Dest
instrbusin[68] ={LSR, XZR, 6'b000001, X7, X6};

daddrbusout[68] = dontcare;
databusin[68] = activez;
databusout[68] = dontcare;

//ADD    X7, X14, X9
```

```
iaddrbusout[69] = 64'h0000000000000114;
//            opcode Source2 Shamt Source1 Dest
instrbusin[69] ={ADD, X9, 6'b000000, X14, X7};

daddrbusout[69] = dontcare;
databusin[69] = activez;
databusout[69] = dontcare;

//SUB    X8, X24, X25
iaddrbusout[70] = 64'h0000000000000118;
//            opcode Source2 Shamt Source1 Dest
instrbusin[70] ={SUB, X25, 6'b000000, X24, X8};

daddrbusout[70] = dontcare;
databusin[70] = activez;
databusout[70] = dontcare;

//SUB    X9, X13, X18
iaddrbusout[71] = 64'h000000000000011C;
//            opcode Source2 Shamt Source1 Dest
instrbusin[70] ={SUB, X18, 6'b000000, X13, X9};

daddrbusout[71] = dontcare;
databusin[71] = activez;
databusout[71] = dontcare;

//ENOR   X10, X30, X29
iaddrbusout[72] = 64'h0000000000000120;
//            opcode Source2 Shamt Source1 Dest
instrbusin[72] ={ENOR, X29, 6'b000000, X30, X10};

daddrbusout[72] = dontcare;
databusin[72] = activez;
databusout[72] = dontcare;

//ORR    X11, X30, X29
iaddrbusout[73] = 64'h0000000000000124;
//            opcode Source2 Shamt Source1 Dest
instrbusin[73] ={ORR, X29, 6'b000000, X30, X11};

daddrbusout[73] = dontcare;
databusin[73] = activez;
databusout[73] = dontcare;
```

```
//EORI   X12, X24, #0FF
iaddrbusout[74] = 64'h0000000000000128;
//          opcode ALU_imm  Source Dest
instrbusin[74]={EORI, 12'h0FF, X24, X12};

daddrbusout[74] = dontcare;
databusin[74] = activez;
databusout[74] = dontcare;

//ENORI  X13, X23, #FF0
iaddrbusout[75] = 64'h000000000000012C;
//          opcode ALU_imm  Source Dest
instrbusin[75]={ENORI, 12'hFF0, X23, X13};

daddrbusout[75] = dontcare;
databusin[75] = activez;
databusout[75] = dontcare;

//ORRI   X14, X18, #100
iaddrbusout[76] = 64'h0000000000000130;
//          opcode ALU_imm  Source Dest
instrbusin[76]={ORRI, 12'h100, X18, X14};

daddrbusout[76] = dontcare;
databusin[76] = activez;
databusout[76] = dontcare;

//ORRI   X15, X20, #34B
iaddrbusout[77] = 64'h0000000000000134;
//          opcode ALU_imm  Source Dest
instrbusin[77]={ORRI, 12'h34B, X20, X15};

daddrbusout[77] = dontcare;
databusin[77] = activez;
databusout[77] = dontcare;

//SUBI   X16, X19, #810
iaddrbusout[78] = 64'h0000000000000138;
//          opcode ALU_imm  Source Dest
instrbusin[78]={SUBI, 12'h810, X19, X16};

daddrbusout[78] = dontcare;
```

```
databusin[78] = activez;
databusout[78] = dontcare;

//ANDI   X17, X27, #FFF
iaddrbusout[79] = 64'h000000000000013C;
//          opcode ALU_imm  Source Dest
instrbusin[79]={ANDI, 12'hFFF, X27, X17};

daddrbusout[79] = dontcare;
databusin[79] = activez;
databusout[79] = dontcare;

//ADD    X18, X0, X1
iaddrbusout[80] = 64'h0000000000000140;
//            opcode Source2 Shamt Source1 Dest
instrbusin[80] ={ADD, X1, 6'b000000, X0, X18};

daddrbusout[80] = dontcare;
databusin[80] = activez;
databusout[80] = dontcare;

//ADD    X19, X2, X3
iaddrbusout[81] = 64'h0000000000000144;
//            opcode Source2 Shamt Source1 Dest
instrbusin[81] ={ADD, X3, 6'b000000, X2, X19};

daddrbusout[81] = dontcare;
databusin[81] = activez;
databusout[81] = dontcare;

//AND    X20, X30, X29
iaddrbusout[82] = 64'h0000000000000148;
//            opcode Source2 Shamt Source1 Dest
instrbusin[82] ={AND, X29, 6'b000000, X30, X20};

daddrbusout[82] = dontcare;
databusin[82] = activez;
databusout[82] = dontcare;

//AND    X21, X29, X5
iaddrbusout[83] = 64'h000000000000014C;
//            opcode Source2 Shamt Source1 Dest
instrbusin[83] ={AND, X5, 6'b000000, X29, X21};
```

```
daddrbusout[83] = dontcare;
databusin[83] = activez;
databusout[83] = dontcare;

//EOR    X22, X7, X8
iaddrbusout[84] = 64'h0000000000000150;
//            opcode Source2 Shamt Source1 Dest
instrbusin[84] ={EOR, X8, 6'b000000, X7, X22};

daddrbusout[84] = dontcare;
databusin[84] = activez;
databusout[84] = dontcare;

//EOR    X23, X9, X9
iaddrbusout[85] = 64'h0000000000000154;
//            opcode Source2 Shamt Source1 Dest
instrbusin[85] ={EOR, X9, 6'b000000, X9, X23};

daddrbusout[85] = dontcare;
databusin[85] = activez;
databusout[85] = dontcare;

//ENOR   X24, X10, X15
iaddrbusout[86] = 64'h0000000000000158;
//            opcode Source2 Shamt Source1 Dest
instrbusin[86] ={ENOR, X15, 6'b000000, X10, X24};

daddrbusout[86] = dontcare;
databusin[86] = activez;
databusout[86] = dontcare;

//ENOR   X25, X11, X12
iaddrbusout[87] = 64'h000000000000015C;
//            opcode Source2 Shamt Source1 Dest
instrbusin[87] ={ENOR, X12, 6'b000000, X11, X25};

daddrbusout[87] = dontcare;
databusin[87] = activez;
databusout[87] = dontcare;

//LSL    X26, X6, #2
iaddrbusout[88] = 64'h0000000000000160;
```

```
//          opcode Source2 Shamt Source1 Dest
instrbusin[88] ={LSL, XZR, 6'b000010, X6, X26};

daddrbusout[88] = dontcare;
databusin[88] = activez;
databusout[88] = dontcare;

//LSL    X27, X5, #4
iaddrbusout[89] = 64'h0000000000000164;
//          opcode Source2 Shamt Source1 Dest
instrbusin[89] ={LSL, XZR, 6'b000100, X5, X27};

daddrbusout[89] = dontcare;
databusin[89] = activez;
databusout[89] = dontcare;

//LSR    X28, X0, #6
iaddrbusout[90] = 64'h0000000000000168;
//          opcode Source2 Shamt Source1 Dest
instrbusin[90] ={LSR, XZR, 6'b000110, X0, X28};

daddrbusout[90] = dontcare;
databusin[90] = activez;
databusout[90] = dontcare;

//LSR    X29, X2, #10
iaddrbusout[91] = 64'h000000000000016C;
//          opcode Source2 Shamt Source1 Dest
instrbusin[91] ={LSR, XZR, 6'b001010, X2, X29};

daddrbusout[91] = dontcare;
databusin[91] = activez;
databusout[91] = dontcare;

//ORR    X30, X14, X16
iaddrbusout[92] = 64'h0000000000000170;
//          opcode Source2 Shamt Source1 Dest
instrbusin[92] ={ORR, X16, 6'b000000, X14, X30};

daddrbusout[92] = dontcare;
databusin[92] = activez;
databusout[92] = dontcare;
```

```
//ORR    X0, X10, X12
iaddrbusout[93] = 64'h0000000000000174;
//            opcode Source2 Shamt Source1 Dest
instrbusin[93] ={ORR, X12, 6'b000000, X10, X0};

daddrbusout[93] = dontcare;
databusin[93] = activez;
databusout[93] = dontcare;

//SUB    X1, X16, X17
iaddrbusout[94] = 64'h0000000000000178;
//            opcode Source2 Shamt Source1 Dest
instrbusin[94] ={SUB, X17, 6'b000000, X16, X1};

daddrbusout[94] = dontcare;
databusin[94] = activez;
databusout[94] = dontcare;

//SUB    X2, X18, X19
iaddrbusout[95] = 64'h000000000000017C;
//            opcode Source2 Shamt Source1 Dest
instrbusin[95] ={SUB, X19, 6'b000000, X18, X2};

daddrbusout[95] = dontcare;
databusin[95] = activez;
databusout[95] = dontcare;

//ADDI   X3, X9, #0A2
iaddrbusout[96] = 64'h0000000000000180;
//          opcode ALU_imm  Source Dest
instrbusin[96]={ADDI, 12'h0A2, X9, X3};

daddrbusout[96] = dontcare;
databusin[96] = activez;
databusout[96] = dontcare;

//ADDI   X4, X10, #19E
iaddrbusout[97] = 64'h0000000000000184;
//          opcode ALU_imm  Source Dest
instrbusin[97]={ADDI, 12'h19E, X10, X4};

daddrbusout[97] = dontcare;
databusin[97] = activez;
```

```
databusout[97] = dontcare;

//EORI   X5, X11, #908
iaddrbusout[98] = 64'h0000000000000188;
//          opcode ALU_imm  Source Dest
instrbusin[98]={EORI, 12'h908, X11, X5};

daddrbusout[98] = dontcare;
databusin[98] = activez;
databusout[98] = dontcare;

//EORI   X6, X25, #ABE
iaddrbusout[99] = 64'h000000000000018C;
//          opcode ALU_imm  Source Dest
instrbusin[99]={EORI, 12'hABE, X25, X6};

daddrbusout[99] = dontcare;
databusin[99] = activez;
databusout[99] = dontcare;

//ENORI X7, X26, #1BC
iaddrbusout[100] = 64'h0000000000000190;
//          opcode ALU_imm  Source Dest
instrbusin[100]={ENORI, 12'h1BC, X26, X7};

daddrbusout[100] = dontcare;
databusin[100] = activez;
databusout[100] = dontcare;

//ENORI X8, X27, #234
iaddrbusout[101] = 64'h0000000000000194;
//          opcode ALU_imm  Source Dest
instrbusin[101]={ENORI, 12'h234, X27, X8};

daddrbusout[101] = dontcare;
databusin[101] = activez;
databusout[101] = dontcare;

//ORRI  X9, X28, #FDC
iaddrbusout[102] = 64'h0000000000000198;
//          opcode ALU_imm  Source Dest
instrbusin[102]={ORRI, 12'hFDC, X28, X9};
```

```
daddrbusout[102] = dontcare;
databusin[102] = activez;
databusout[102] = dontcare;

//ORRI  X10, X29, #024
iaddrbusout[103] = 64'h000000000000019C;
//          opcode ALU_imm  Source Dest
instrbusin[103]={ORRI, 12'h024, X29, X10};

daddrbusout[103] = dontcare;
databusin[103] = activez;
databusout[103] = dontcare;

//SUBI  X11, X30, #BBA
iaddrbusout[104] = 64'h00000000000001A0;
//          opcode ALU_imm  Source Dest
instrbusin[104]={SUBI, 12'hBBA, X30, X11};

daddrbusout[104] = dontcare;
databusin[104] = activez;
databusout[104] = dontcare;

//SUBI  X12, X16, #987
iaddrbusout[105] = 64'h00000000000001A4;
//          opcode ALU_imm  Source Dest
instrbusin[105]={SUBI, 12'h987, X16, X12};

daddrbusout[105] = dontcare;
databusin[105] = activez;
databusout[105] = dontcare;

//LDUR  X13, [XZR, #5A]
iaddrbusout[106] = 64'h00000000000001A8;
//          opcode offset        OP   Source Dest
instrbusin[106]={LDUR, 9'b001011010, 2'b00, XZR, X13};

daddrbusout[106] = 64'h000000000000005A;
databusin[106] = 64'h0000444400008888;
databusout[106] = dontcare;

//LDUR  X14, [XZR, #FD]
iaddrbusout[107] = 64'h00000000000001AC;
//          opcode offset        OP   Source Dest
```

```
instrbusin[65]={LDUR, 9'b011111101, 2'b00, X2, X14};

daddrbusout[107] = dontcare;
databusin[107] = 64'h6666000000002222;
databusout[107] = dontcare;

//STUR  X10, [XZR, #9B]
iaddrbusout[108] = 64'h00000000000001B0;
//          opcode offset         OP   Source Dest
instrbusin[108]={STUR, 9'b010011011, 2'b00, XZR, XZR};

daddrbusout[108] = 64'h000000000000009B;
databusin[108] = activez;
databusout[108] = 64'h0000000000000000;

//STUR  XZR, [X4, #78]
iaddrbusout[109] = 64'h00000000000001B4;
//          opcode offset         OP   Source Dest
instrbusin[109]={STUR, 9'b001111000, 2'b00, X4, XZR};

daddrbusout[109] = 64'hFFFF12346543497F;
databusin[109] = activez;
databusout[109] = 64'h0000000000000000;

//ANDI  X17, X5, #984
iaddrbusout[110] = 64'h00000000000001B8;
//          opcode ALU_imm  Source Dest
instrbusin[110]={ANDI, 12'h984, X5, X17};

daddrbusout[110] = dontcare;
databusin[110] = activez;
databusout[110] = dontcare;

//ANDI  X18, X6, #FEE
iaddrbusout[111] = 64'h00000000000001BC;
//          opcode ALU_imm  Source Dest
instrbusin[111]={ANDI, 12'hFEE, X6, X18};

daddrbusout[111] = dontcare;
databusin[111] = activez;
databusout[111] = dontcare;

//ADDS  X0, X21, X20
```

```
iaddrbusout[112] = 64'h00000000000001C0;
//            opcode Source2 Shamt Source1 Dest
instrbusin[112] ={ADDS, X20, 6'b000000, X21, X0};

daddrbusout[112] = dontcare;
databusin[112] = activez;
databusout[112] = dontcare;

//BEQ   #10       //Branch not taken
iaddrbusout[113] = 64'h00000000000001C4;
//              opcode  offset  dest reg
instrbusin[113] = {BEQ, 19'h28, XZR};

daddrbusout[113] = dontcare;
databusin[113] = activez;
databusout[113] = dontcare;

//NOP
iaddrbusout[114] = 64'h00000000000001C8;

instrbusin[114] = 32'b00000000000000000000000000000000;

daddrbusout[114] = dontcare;
databusin[114] = activez;
databusout[114] = dontcare;

//ADD X1, X23, X23
iaddrbusout[115] = 64'h00000000000001CC;
//            opcode Source2 Shamt Source1 Dest
instrbusin[115] ={ADD, X23, 6'b000000, X23, X1};

daddrbusout[115] = dontcare;
databusin[115] = activez;
databusout[115] = dontcare;

//ADDS X2, X15, X28
iaddrbusout[116] = 64'h00000000000001D0;
//            opcode Source2 Shamt Source1 Dest
instrbusin[116] ={ADDS, X28, 6'b000000, X15, X2};

daddrbusout[116] = dontcare;
databusin[116] = activez;
databusout[116] = dontcare;
```

```
//BLT #2        //Branch taken
iaddrbusout[117] = 64'h00000000000001D4;
//              opcode  offset  dest reg
instrbusin[117] = {BLT, 19'h8, XZR};

daddrbusout[117] = dontcare;
databusin[117] = activez;
databusout[117] = dontcare;

//NOP
iaddrbusout[118] = 64'h00000000000001D8;

instrbusin[118] = 32'b00000000000000000000000000000000;

daddrbusout[118] = dontcare;
databusin[118] = activez;
databusout[118] = dontcare;

//NOP
iaddrbusout[119] = 64'h00000000000001DC;

instrbusin[119] = 32'b00000000000000000000000000000000;

daddrbusout[119] = dontcare;
databusin[119] = activez;
databusout[119] = dontcare;

//NOP
iaddrbusout[120] = 64'h00000000000001E0;

instrbusin[120] = 32'b00000000000000000000000000000000;

daddrbusout[120] = dontcare;
databusin[120] = activez;
databusout[120] = dontcare;

//ADDS X4, X28, X30
iaddrbusout[121] = 64'h00000000000001E4;
//            opcode Source2 Shamt Source1 Dest
instrbusin[121] ={ADDS, X30, 6'b000000, X28, X4};

daddrbusout[121] = dontcare;
```

```
databusin[121] = activez;
databusout[121] = dontcare;

//BGE #5        //Branch not taken
iaddrbusout[122] = 64'h00000000000001E8;
//              opcode  offset  dest reg
instrbusin[122] = {BGE, 19'h14, XZR};

daddrbusout[122] = dontcare;
databusin[122] = activez;
databusout[122] = dontcare;

//NOP
iaddrbusout[123] = 64'h00000000000001EC;

instrbusin[123] = 32'b00000000000000000000000000000000;

daddrbusout[123] = dontcare;
databusin[123] = activez;
databusout[123] = dontcare;

//NOP
iaddrbusout[124] = 64'h00000000000001F0;

instrbusin[124] = 32'b00000000000000000000000000000000;

daddrbusout[124] = dontcare;
databusin[124] = activez;
databusout[124] = dontcare;

//NOP
iaddrbusout[125] = 64'h00000000000001F4;

instrbusin[125] = 32'b00000000000000000000000000000000;

daddrbusout[125] = dontcare;
databusin[125] = activez;
databusout[125] = dontcare;

//B #20         //Branch taken
iaddrbusout[126] = 64'h00000000000001F8;
//              opcode   Offset
instrbusin[126] = {B, 26'h50};
```

```
daddrbusout[126] = dontcare;
databusin[126] = activez;
databusout[126] = dontcare;

//NOP
iaddrbusout[127] = 64'h00000000000001FC;

instrbusin[127] = 32'b00000000000000000000000000000000;

daddrbusout[127] = dontcare;
databusin[127] = activez;
databusout[127] = dontcare;

//NOP
iaddrbusout[128] = 64'h0000000000000338;

instrbusin[128] = 32'b00000000000000000000000000000000;

daddrbusout[128] = dontcare;
databusin[128] = activez;
databusout[128] = dontcare;

//NOP
iaddrbusout[129] = 64'h000000000000033C;

instrbusin[129] = 32'b00000000000000000000000000000000;

daddrbusout[129] = dontcare;
databusin[129] = activez;
databusout[129] = dontcare;

//ANDS X4, X19, X22
iaddrbusout[130] = 64'h0000000000000340;
//            opcode Source2 Shamt Source1 Dest
instrbusin[130] ={ANDS, X22, 6'b000000, X19, X4};

daddrbusout[130] = dontcare;
databusin[130] = activez;
databusout[130] = dontcare;

//BLT #20       //Branch not taken
iaddrbusout[131] = 64'h0000000000000344;
```

```
//              opcode  offset  dest reg
instrbusin[131] = {BLT, 19'h50, XZR};

daddrbusout[131] = dontcare;
databusin[131] = activez;
databusout[131] = dontcare;

//NOP
iaddrbusout[132] = 64'h0000000000000348;

instrbusin[132] = 32'b00000000000000000000000000000000;

daddrbusout[132] = dontcare;
databusin[132] = activez;
databusout[132] = dontcare;

//CBZ X23, #10          //Branch NOT taken
iaddrbusout[133] = 64'h000000000000034C;
//              opcode  offset  dest reg
instrbusin[133] = {CBZ, 19'h28, XZR};

daddrbusout[133] = dontcare;
databusin[133] = activez;
databusout[133] = dontcare;

//NOP
iaddrbusout[134] = 64'h0000000000000350;

instrbusin[134] = 32'b00000000000000000000000000000000;

daddrbusout[134] = dontcare;
databusin[134] = activez;
databusout[134] = dontcare;

//CBNZ X27, #15     //Branch taken
iaddrbusout[135] = 64'h00000000000003EC;
//              opcode  offset  dest reg
instrbusin[135] = {CBNZ, 19'h3C, XZR};

daddrbusout[135] = dontcare;
databusin[135] = activez;
databusout[135] = dontcare;
```

```
//NOP
iaddrbusout[136] = 64'h00000000000003F0;

instrbusin[136] = 32'b00000000000000000000000000000000;

daddrbusout[136] = dontcare;
databusin[136] = activez;
databusout[136] = dontcare;

//CBNZ X23, #80     //Branch not taken
iaddrbusout[137] = 64'h00000000000003F4;
//              opcode  offset  dest reg
instrbusin[137] = {CBNZ, 19'h140, XZR};

daddrbusout[137] = dontcare;
databusin[137] = activez;
databusout[137] = dontcare;

//NOP
iaddrbusout[138] = 64'h00000000000003F8;

instrbusin[138] = 32'b00000000000000000000000000000000;

daddrbusout[138] = dontcare;
databusin[138] = activez;
databusout[138] = dontcare;

//B     #10         //Branch taken
iaddrbusout[139] = 64'h00000000000003FC;
//              opcode   Offset
instrbusin[139] = {B, 26'h28};

daddrbusout[139] = dontcare;
databusin[139] = activez;
databusout[139] = dontcare;

//NOP
iaddrbusout[140] = 64'h0000000000000400;

instrbusin[140] = 32'b00000000000000000000000000000000;

daddrbusout[140] = dontcare;
databusin[140] = activez;
```

```
databusout[140] = dontcare;

//NOP
iaddrbusout[141] = 64'h000000000000049C;

instrbusin[141] = 32'b00000000000000000000000000000000;

daddrbusout[141] = dontcare;
databusin[141] = activez;
databusout[141] = dontcare;

//SUBS X5, X20, X21
iaddrbusout[142] = 64'h00000000000004A0;
//            opcode Source2 Shamt Source1 Dest
instrbusin[142] ={SUBS, X21, 6'b000000, X20, X5};

daddrbusout[142] = dontcare;
databusin[142] = activez;
databusout[142] = dontcare;

//BNE #10           //Branch taken
iaddrbusout[143] = 64'h00000000000004A4;
//              opcode  offset  dest reg
instrbusin[143] = {BNE, 19'h28, XZR};

daddrbusout[143] = dontcare;
databusin[143] = activez;
databusout[143] = dontcare;

//NOP
iaddrbusout[144] = 64'h00000000000004A8;

instrbusin[144] = 32'b00000000000000000000000000000000;

daddrbusout[144] = dontcare;
databusin[144] = activez;
databusout[144] = dontcare;

//NOP
iaddrbusout[145] = 64'h00000000000004AC;

instrbusin[145] = 32'b00000000000000000000000000000000;
```

```
daddrbusout[145] = dontcare;
databusin[145] = activez;
databusout[145] = dontcare;

//NOP
iaddrbusout[146] = 64'h00000000000004B0;

instrbusin[146] = 32'b00000000000000000000000000000000;

daddrbusout[146] = dontcare;
databusin[146] = activez;
databusout[146] = dontcare;

//CBZ X15, #20      //Branch not taken
iaddrbusout[147] = 64'h00000000000004B4;
//              opcode  offset  dest reg
instrbusin[147] = {CBZ, 19'h50, XZR};

daddrbusout[147] = dontcare;
databusin[147] = activez;
databusout[147] = dontcare;

//NOP
iaddrbusout[148] = 64'h00000000000004B8;

instrbusin[148] = 32'b00000000000000000000000000000000;

daddrbusout[148] = dontcare;
databusin[148] = activez;
databusout[148] = dontcare;

//CBZ X30, #0       //Branch not taken
iaddrbusout[149] = 64'h00000000000005F4;
//              opcode  offset  dest reg
instrbusin[149] = {CBZ, 19'h0, XZR};

daddrbusout[149] = dontcare;
databusin[149] = activez;
databusout[149] = dontcare;

//NOP
iaddrbusout[150] = 64'h00000000000005F8;
```

```
instrbusin[150] = 32'b00000000000000000000000000000000;

daddrbusout[150] = dontcare;
databusin[150] = activez;
databusout[150] = dontcare;

//CBNZ X20, #0      //Branch taken
iaddrbusout[151] = 64'h00000000000005F4;
//                opcode  offset  dest reg
instrbusin[151] = {CBNZ, 19'h0, XZR};

daddrbusout[151] = dontcare;
databusin[151] = activez;
databusout[151] = dontcare;

//NOP
iaddrbusout[152] = 64'h00000000000005F8;

instrbusin[152] = 32'b00000000000000000000000000000000;

daddrbusout[152] = dontcare;
databusin[152] = activez;
databusout[152] = dontcare;

//NOP
iaddrbusout[153] = 64'h00000000000005FC;

instrbusin[153] = 32'b00000000000000000000000000000000;

daddrbusout[153] = dontcare;
databusin[153] = activez;
databusout[153] = dontcare;

//ADDIS X15, X18, #9A
iaddrbusout[154] = 64'h0000000000000600;
//           opcode ALU_imm  Source Dest
instrbusin[154]={ADDIS, 12'h09A, X18, X15};

daddrbusout[154] = dontcare;
databusin[154] = activez;
databusout[154] = dontcare;

//BEQ #80           //Branch not taken
```

```
iaddrbusout[155] = 64'h0000000000000604;
//              opcode  offset  dest reg
instrbusin[155] = {BEQ, 19'h140, XZR};

daddrbusout[155] = dontcare;
databusin[155] = activez;
databusout[155] = dontcare;

//NOP
iaddrbusout[156] = 64'h0000000000000608;

instrbusin[156] = 32'b00000000000000000000000000000000;

daddrbusout[156] = dontcare;
databusin[156] = activez;
databusout[156] = dontcare;

//SUB X16, X6, X10
iaddrbusout[157] = 64'h000000000000060C;
//            opcode Source2 Shamt Source1 Dest
instrbusin[157] ={SUB, X10, 6'b000000, X6, X16};

daddrbusout[157] = dontcare;
databusin[157] = activez;
databusout[157] = dontcare;

//ADDIS X19, X5, #8AC
iaddrbusout[158] = 64'h0000000000000610;
//          opcode ALU_imm  Source Dest
instrbusin[158]={ADDIS, 12'h8AC, X5, X19};

daddrbusout[158] = dontcare;
databusin[158] = activez;
databusout[158] = dontcare;

//BLT #20          //Branch not taken
iaddrbusout[159] = 64'h0000000000000614;
//              opcode  offset  dest reg
instrbusin[159] = {BLT, 19'h50, XZR};

daddrbusout[159] = dontcare;
databusin[159] = activez;
databusout[159] = dontcare;
```

```
//NOP
iaddrbusout[160] = 64'h0000000000000618;

instrbusin[160] = 32'b00000000000000000000000000000000;

daddrbusout[160] = dontcare;
databusin[160] = activez;
databusout[160] = dontcare;

//ADDI X20, X8, #09E
iaddrbusout[161] = 64'h000000000000061C;
//          opcode ALU_imm  Source Dest
instrbusin[161]={ADDI, 12'h09E, X8, X20};

daddrbusout[161] = dontcare;
databusin[161] = activez;
databusout[161] = dontcare;

//ADDIS X21, X12, #FEA
iaddrbusout[162] = 64'h0000000000000620;
//          opcode ALU_imm  Source Dest
instrbusin[162]={ADDIS, 12'hFEA, X12, X21};

daddrbusout[162] = dontcare;
databusin[162] = activez;
databusout[162] = dontcare;

//CBZ X1, #10       //Branch taken
iaddrbusout[163] = 64'h0000000000000624;
//              opcode  offset  dest reg
instrbusin[163] = {BEQ, 19'h28, XZR};

daddrbusout[163] = dontcare;
databusin[163] = activez;
databusout[163] = dontcare;

//NOP
iaddrbusout[164] = 64'h0000000000000628;

instrbusin[164] = 32'b00000000000000000000000000000000;

daddrbusout[164] = dontcare;
```

```
databusin[164] = activez;
databusout[164] = dontcare;

//NOP
iaddrbusout[165] = 64'h000000000000062C;

instrbusin[165] = 32'b00000000000000000000000000000000;

daddrbusout[165] = dontcare;
databusin[165] = activez;
databusout[165] = dontcare;

//B #0            //Branch taken
iaddrbusout[166] = 64'h0000000000000630;
//              opcode   Offset
instrbusin[166] = {B, 26'h0};

daddrbusout[166] = dontcare;
databusin[166] = activez;
databusout[166] = dontcare;

//NOP
iaddrbusout[167] = 64'h0000000000000634;

instrbusin[167] = 32'b00000000000000000000000000000000;

daddrbusout[167] = dontcare;
databusin[167] = activez;
databusout[167] = dontcare;

//CBNZ X1, #10      //Branch not taken
iaddrbusout[168] = 64'h0000000000000630;
//               opcode  offset  dest reg
instrbusin[168] = {CBNZ, 19'h28, XZR};

daddrbusout[168] = dontcare;
databusin[168] = activez;
databusout[168] = dontcare;

//NOP
iaddrbusout[169] = 64'h0000000000000634;

instrbusin[169] = 32'b00000000000000000000000000000000;
```

```
daddrbusout[169] = dontcare;
databusin[169] = activez;
databusout[169] = dontcare;

//ANDIS X22, X6, #ABC
iaddrbusout[170] = 64'h0000000000000638;
//          opcode ALU_imm  Source Dest
instrbusin[170]={ANDIS, 12'hABC, X6, X22};

daddrbusout[170] = dontcare;
databusin[170] = activez;
databusout[170] = dontcare;

//BEQ #1            //Branch not taken
iaddrbusout[171] = 64'h000000000000063C;
//              opcode  offset  dest reg
instrbusin[171] = {BEQ, 19'h4, XZR};

daddrbusout[171] = dontcare;
databusin[171] = activez;
databusout[171] = dontcare;

//NOP
iaddrbusout[172] = 64'h0000000000000640;

instrbusin[172] = 32'b00000000000000000000000000000000;

daddrbusout[172] = dontcare;
databusin[172] = activez;
databusout[172] = dontcare;

//ANDIS X23, X9, #765
iaddrbusout[173] = 64'h0000000000000644;
//          opcode ALU_imm  Source Dest
instrbusin[173]={ANDIS, 12'h765, X9, X23};

daddrbusout[173] = dontcare;
databusin[173] = activez;
databusout[173] = dontcare;

//BNE #1        //Branch taken
iaddrbusout[174] = 64'h0000000000000648;
```

```
//              opcode  offset  dest reg
instrbusin[174] = {BNE, 19'h4, XZR};

daddrbusout[174] = dontcare;
databusin[174] = activez;
databusout[174] = dontcare;

//NOP
iaddrbusout[175] = 64'h000000000000064C;

instrbusin[175] = 32'b00000000000000000000000000000000;

daddrbusout[175] = dontcare;
databusin[175] = activez;
databusout[175] = dontcare;

//NOP
iaddrbusout[176] = 64'h0000000000000658;

instrbusin[176] = 32'b00000000000000000000000000000000;

daddrbusout[176] = dontcare;
databusin[176] = activez;
databusout[176] = dontcare;

//NOP
iaddrbusout[177] = 64'h000000000000065C;

instrbusin[177] = 32'b00000000000000000000000000000000;

daddrbusout[177] = dontcare;
databusin[177] = activez;
databusout[177] = dontcare;

//CBNZ X1, #12      //Branch not taken
iaddrbusout[178] = 64'h0000000000000660;
//              opcode  offset  dest reg
instrbusin[178] = {BEQ, 19'h30, XZR};

daddrbusout[178] = dontcare;
databusin[178] = activez;
databusout[178] = dontcare;
```

```
//NOP
iaddrbusout[179] = 64'h0000000000000664;

instrbusin[179] = 32'b00000000000000000000000000000000;

daddrbusout[179] = dontcare;
databusin[179] = activez;
databusout[179] = dontcare;

//MOVZ  X23, #3A3
iaddrbusout[180] = 64'h0000000000000668;
//              opcode         Imm      dest
instrbusin[180] = {MOVZ, 2'b10, 16'h3A3, X23};

daddrbusout[180] = dontcare;
databusin[180] = activez;
databusout[180] = dontcare;

//CBZ   X16, #5     //Branch taken
iaddrbusout[181] = 64'h000000000000066C;
//               opcode  offset  dest reg
instrbusin[181] = {CBZ, 19'h14, XZR};

daddrbusout[181] = dontcare;
databusin[181] = activez;
databusout[181] = dontcare;

//NOP
iaddrbusout[182] = 64'h0000000000000670;

instrbusin[182] = 32'b00000000000000000000000000000000;

daddrbusout[182] = dontcare;
databusin[182] = activez;
databusout[182] = dontcare;

//MOVZ  X24, #0
iaddrbusout[183] = 64'h00000000000006BC;
//              opcode         Imm      dest
instrbusin[183] = {MOVZ, 2'b10, 16'h000, X24};

daddrbusout[183] = dontcare;
databusin[183] = activez;
```

```
databusout[183] = dontcare;

//SUBIS X25, X13, #AAA
iaddrbusout[184] = 64'h00000000000006C0;
//          opcode ALU_imm  Source Dest
instrbusin[184]={SUBIS, 12'hAAA, X13, X25};

daddrbusout[184] = dontcare;
databusin[184] = activez;
databusout[184] = dontcare;

//CBNZ  X1, #9      //Branch not taken
iaddrbusout[185] = 64'h00000000000006C4;
//              opcode  offset  dest reg
instrbusin[185] = {CBNZ, 19'h24, XZR};

daddrbusout[185] = dontcare;
databusin[185] = activez;
databusout[185] = dontcare;

//NOP
iaddrbusout[186] = 64'h00000000000006C8;

instrbusin[186] = 32'b00000000000000000000000000000000;

daddrbusout[186] = dontcare;
databusin[186] = activez;
databusout[186] = dontcare;

//SUBIS X26, X8, #0
iaddrbusout[187] = 64'h00000000000006CC;
//          opcode ALU_imm  Source Dest
instrbusin[187]={SUBIS, 12'h000, X8, X26};

daddrbusout[187] = dontcare;
databusin[187] = activez;
databusout[187] = dontcare;

//SUBIS X27, X3, #EFE
iaddrbusout[188] = 64'h00000000000006D0;
//          opcode ALU_imm  Source Dest
instrbusin[188]={SUBIS, 12'hEFE, X3, X27};
```

```
daddrbusout[188] = dontcare;
databusin[188] = activez;
databusout[188] = dontcare;

//BNE #14          //Branch taken
iaddrbusout[189] = 64'h00000000000006D4;
//               opcode  offset  dest reg
instrbusin[189] = {BNE, 19'h38, XZR};

daddrbusout[189] = dontcare;
databusin[189] = activez;
databusout[189] = dontcare;

//NOP
iaddrbusout[190] = 64'h00000000000006D8;

instrbusin[190] = 32'b00000000000000000000000000000000;

daddrbusout[190] = dontcare;
databusin[190] = activez;
databusout[190] = dontcare;

//NOP
iaddrbusout[191] = 64'h00000000000006DC;

instrbusin[191] = 32'b00000000000000000000000000000000;

daddrbusout[191] = dontcare;
databusin[191] = activez;
databusout[191] = dontcare;

//NOP
iaddrbusout[192] = 64'h00000000000006E0;

instrbusin[192] = 32'b00000000000000000000000000000000;

daddrbusout[192] = dontcare;
databusin[192] = activez;
databusout[192] = dontcare;

//MOVZ X28, #9AE
iaddrbusout[193] = 64'h00000000000006E4;
//              opcode        Imm      dest
```

```
instrbusin[193] = {MOVZ, 2'b10, 16'h9AE, X28};

daddrbusout[193] = dontcare;
databusin[193] = activez;
databusout[193] = dontcare;

//ADD X29, X0, X1
iaddrbusout[194] = 64'h00000000000006E8;
//          opcode Source2 Shamt Source1 Dest
instrbusin[194] ={ADD, X1, 6'b000000, X0, X29};

daddrbusout[194] = dontcare;
databusin[194] = activez;
databusout[194] = dontcare;

//AND X30, X3, X2
iaddrbusout[195] = 64'h00000000000006EC;
//          opcode Source2 Shamt Source1 Dest
instrbusin[195] ={AND, X2, 6'b000000, X3, X30};

daddrbusout[195] = dontcare;
databusin[195] = activez;
databusout[195] = dontcare;

//SUB X5, X2, X7
iaddrbusout[196] = 64'h00000000000006F0;
//          opcode Source2 Shamt Source1 Dest
instrbusin[196] ={SUB, X7, 6'b000000, X2, X5};

daddrbusout[196] = dontcare;
databusin[196] = activez;
databusout[196] = dontcare;

//ORR X0, X10, X11
iaddrbusout[197] = 64'h00000000000006F4;
//          opcode Source2 Shamt Source1 Dest
instrbusin[197] ={ORR, X11, 6'b000000, X10, X0};

daddrbusout[197] = dontcare;
databusin[197] = activez;
databusout[197] = dontcare;

//AND X1, X1, X4
```

```
iaddrbusout[198] = 64'h00000000000006F8;
//            opcode Source2 Shamt Source1 Dest
instrbusin[198] ={SUB, X4, 6'b000000, X1, X1};

daddrbusout[198] = dontcare;
databusin[198] = activez;
databusout[198] = dontcare;

//ADDI X2, X20, #90A
iaddrbusout[199] = 64'h00000000000006FC;
//          opcode ALU_imm  Source Dest
instrbusin[199]={ADDI, 12'h90A, X20, X2};

daddrbusout[199] = dontcare;
databusin[199] = activez;
databusout[199] = dontcare;

//SUBI X4, X3, #AFF
iaddrbusout[200] = 64'h0000000000000700;
//          opcode ALU_imm  Source Dest
instrbusin[200]={SUBI, 12'hAFF, X3, X4};

daddrbusout[200] = dontcare;
databusin[200] = activez;
databusout[200] = dontcare;

//ORRI X6, X8, #00A
iaddrbusout[201] = 64'h0000000000000704;
//          opcode ALU_imm  Source Dest
instrbusin[201]={ORRI, 12'h00A, X8, X6};

daddrbusout[201] = dontcare;
databusin[201] = activez;
databusout[201] = dontcare;

//EOR X7, X17, X18
iaddrbusout[202] = 64'h0000000000000708;
//            opcode Source2 Shamt Source1 Dest
instrbusin[202] ={EOR, X18, 6'b000000, X17, X7};

daddrbusout[202] = dontcare;
databusin[202] = activez;
databusout[202] = dontcare;
```

```
//EORI X8, X22, #00C
iaddrbusout[203] = 64'h000000000000070C;
//          opcode ALU_imm  Source Dest
instrbusin[203]={EORI, 12'h00C, X22, X8};

daddrbusout[203] = dontcare;
databusin[203] = activez;
databusout[203] = dontcare;

//ADD X9, X27, X28
iaddrbusout[204] = 64'h0000000000000710;
//           opcode Source2 Shamt Source1 Dest
instrbusin[204] ={ADD, X28, 6'b000000, X27, X9};

daddrbusout[204] = dontcare;
databusin[204] = activez;
databusout[204] = dontcare;

//SUB X10, X27, X25
iaddrbusout[205] = 64'h0000000000000714;
//           opcode Source2 Shamt Source1 Dest
instrbusin[205] ={SUB, X25, 6'b000000, X27, X10};

daddrbusout[205] = dontcare;
databusin[205] = activez;
databusout[205] = dontcare;

//AND X11, X20, X21
iaddrbusout[206] = 64'h0000000000000718;
//           opcode Source2 Shamt Source1 Dest
instrbusin[206] ={SUB, X21, 6'b000000, X20, X11};

daddrbusout[206] = dontcare;
databusin[206] = activez;
databusout[206] = dontcare;

//EOR X12, X26, X15
iaddrbusout[207] = 64'h000000000000071C;
//           opcode Source2 Shamt Source1 Dest
instrbusin[207] ={EOR, X15, 6'b000000, X26, X12};

daddrbusout[207] = dontcare;
```

```
databusin[207] = activez;
databusout[207] = dontcare;

//ORR X13, X2, X16
iaddrbusout[208] = 64'h0000000000000720;
//           opcode Source2 Shamt Source1 Dest
instrbusin[208] ={ORR, X16, 6'b000000, X2, X13};

daddrbusout[208] = dontcare;
databusin[208] = activez;
databusout[208] = dontcare;

//ADDI X14, X28, #E45
iaddrbusout[209] = 64'h0000000000000724;
//          opcode ALU_imm  Source Dest
instrbusin[209]={ADDI, 12'hE45, X28, X14};

daddrbusout[209] = dontcare;
databusin[209] = activez;
databusout[209] = dontcare;


dontcare = 64'hx;
activez = 64'hz;

//ntests = 300;

$timeformat(-9,1,"ns",12);

end

//assumes positive edge FF.
//testbench reads databus when clk high, writes databus when clk low.
assign databus = clkd ? 64'bz : databusk;

//Change inputs in middle of period (falling edge).
initial begin
  error = 0;
  clkd =1;
  clk=1;
  $display ("Time=%t\n  clk=%b", $realtime, clk);
  databusk = 64'bz;
```

```
 //extended reset to set up PC MUX
 reset = 1;
 $display ("reset=%b", reset);
 #5
 clk=0;
 clkd=0;
 $display ("Time=%t\n  clk=%b", $realtime, clk);
 #5

 clk=1;
 clkd=1;
 $display ("Time=%t\n  clk=%b", $realtime, clk);
 #5
 clk=0;
 clkd=0;
 $display ("Time=%t\n  clk=%b", $realtime, clk);
 #5
 $display ("Time=%t\n  clk=%b", $realtime, clk);

for (k=0; k<= 209; k=k+1) begin
   clk=1;
   $display ("Time=%t\n  clk=%b", $realtime, clk);
   #2
   clkd=1;
   #3
   $display ("Time=%t\n  clk=%b", $realtime, clk);
   reset = 0;
   $display ("reset=%b", reset);


   //set load data for 3rd previous instruction
   if (k >=3)
     databusk = databusin[k-3];

   //check PC for this instruction
   if (k >= 0) begin
     $display ("  Testing PC for instruction %d", k);
     $display ("    Your iaddrbus =    %b", iaddrbus);
     $display ("    Correct iaddrbus = %b", iaddrbusout[k]);
     if (iaddrbusout[k] !== iaddrbus) begin
       $display ("    -------------ERROR. A Mismatch Has Occured-----------");
       error = error + 1;
     end
```

```
        end

        //put next instruction on ibus
        instrbus=instrbusin[k];
        $display ("  instrbus=%b %b %b %b %b for instruction %d: %s", instrbus[31:26],
instrbus[25:21], instrbus[20:16], instrbus[15:11], instrbus[10:0], k, iname[k]);

        //check data address from 3rd previous instruction
        if ( (k >= 3) && (daddrbusout[k-3] !== dontcare) ) begin
          $display ("  Testing data address for instruction %d:", k-3);
          $display ("  %s", iname[k-3]);
          $display ("    Your daddrbus =    %b", daddrbus);
          $display ("    Correct daddrbus = %b", daddrbusout[k-3]);
          if (daddrbusout[k-3] !== daddrbus) begin
            $display ("    -------------ERROR. A Mismatch Has Occured-----------");
            error = error + 1;
          end
        end

        //check store data from 3rd previous instruction
        if ( (k >= 3) && (databusout[k-3] !== dontcare) ) begin
          $display ("  Testing store data for instruction %d:", k-3);
          $display ("  %s", iname[k-3]);
          $display ("    Your databus =    %b", databus);
          $display ("    Correct databus = %b", databusout[k-3]);
          if (databusout[k-3] !== databus) begin
            $display ("    -------------ERROR. A Mismatch Has Occured-----------");
            error = error + 1;
          end
        end

        clk = 0;
        $display ("Time=%t\n  clk=%b", $realtime, clk);
        #2
        clkd = 0;
        #3
        $display ("Time=%t\n  clk=%b", $realtime, clk);
      end

      if ( error !== 0) begin
        $display("--------- SIMULATION UNSUCCESFUL - MISMATCHES HAVE
OCCURED ----------");
        $display(" No. Of Errors = %d", error);
```

```
   end
   if ( error == 0)
     $display("---------YOU DID IT!! SIMULATION SUCCESFULLY FINISHED------
----");
 end

endmodule
```

VI. Acquired Skills

This was a very interesting and challenging assignment to complete. I learned quite a bit about the amount of planning it takes to design a simple CPU. I ended up trying and scrapping several approaches to arrive on my final design. Designing the decode stage was the most challenging part of this assignment as we had to account for twice as many instruction formats as MIPS. Parsing the opcodes and other parts of the ibus efficiently was difficult. I still think my design could be improved by reducing the number of control bits and having the wires be multiple bits long, instead of many 1-bit control signals. I also should've taken notes as I was going along and kept my thoughts more organized because there are a staggering number of signals and intermediate wires to deal with in this project. Some more organization would've made the process a lot smoother.