CMPE 140 – Lab Assignment 8

Haonan Wang
Computer Engineering
Department, San Jose State University

Written By:
Alejandro Chavez-Guerrero 008838349
Patrick Lu 011948001
Chujie Meng 011489296
Wenyi Cai 011232000

Introduction

The purpose of this lab was to convert the fifteen-instruction 32-bit single-cycle MIPS processor into a five-stage, pipelined design. We also had to interface the processor with a factorial accelerator and a general-purpose I/O unit using memory-mapped interface registers. Upon completion we made sure that we were able to verify that we had everything working by looking at the simulation waveforms and also by testing on our FPGA board.

Design Methodology

Task 1: Design pipelined MIPS architecture and SoC interface schematic

The five stages of a processor operation are instruction fetch(IF), instruction decode (ID), execution(EXE), memory access(MEN), and writeback(WB). When operation starts, registers will be run as IF/ID, ID/EXE,EXE/MEM, and MEM/WB. In the pipeline design, an instruction will go into the IF stage and then ID, EXE… However, different from traditional design, when the instruction processes to ID stage, the next instruction can process IF stage. During the whole operation, the control unit will keep sending signals to help instructions move forward to stages. The final design was shown in Figure 1. We also did the comparison between 5-stage pipeline design and the single cycle design, and the result showed that 5-stage pipeline was 3 times faster than the single cycle design. The final result will be shown in table 1:

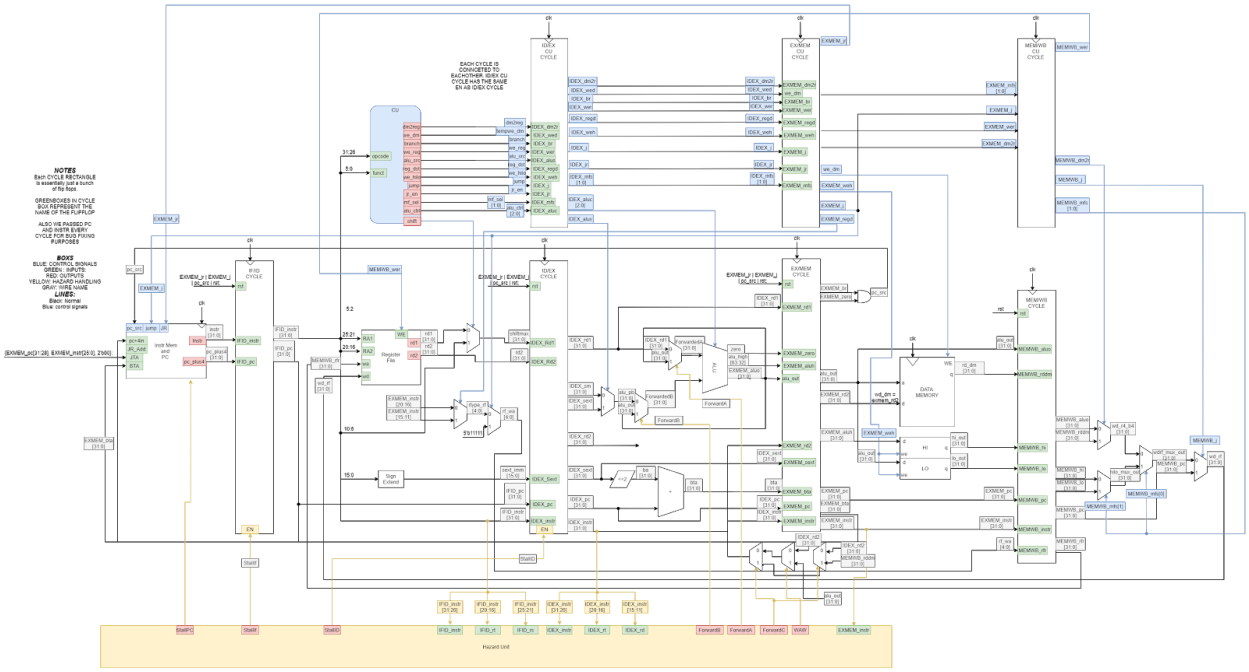


Figure 1. 5-stage pipeline design.

.

Task 2: Create tables for MIPS control unit and performance analysis of hardware accelerated n!

Based on the Figure 1 design, we also created the control unit table which was shown in Table 1. There are three types for MIPS Assembly Instructions, and we categorized them and divided 14 pieces for each instruction included instr, opcode, funct, branch, jump,reg_dst, we_reg, alu_src, we_dm, dm2reg, alu_ctrl, jr_en, mf_sel, and we_hilo. During the design process, we noticed that only J-type instructions require jump signal, only beq instruction requires branch signal, and only multu instruction requires we_hilo signal.

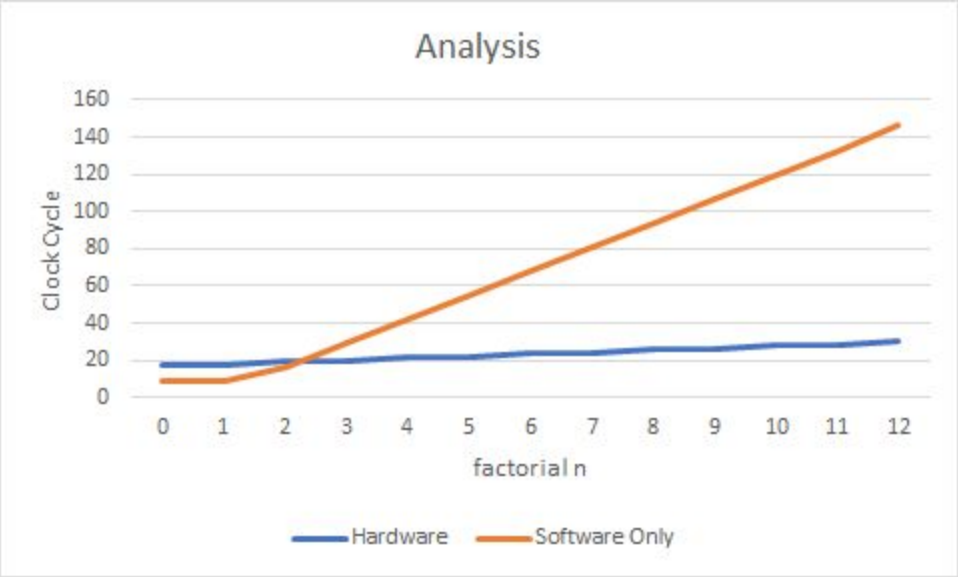
	Instr	opcode	funct	branch	jump	reg_dst	we_reg	alu_src	we_dm	dm2reg	alu_ctrl[2:0]	jr_en	mf_sel[1:0]	we_hilo
R-Type	add	0000_00	10_0000	0	0	1	1	0	0	0	3'b010	0	0	0
	sub	0000_00	10_0010	0	0	1	1	0	0	0	3'b110	0	0	0
	and	0000_00	10_0100	0	0	1	1	0	0	0	3'b000	0	0	0
	or	0000_00	10_0101	0	0	1	1	0	0	0	3'b001	0	0	0
	slt	0000_00	10_1010	0	0	1	1	0	0	0	3'b111	0	0	0
	sll	0000_00	00_0000	0	0	1	1	0	0	0	3'b100	0	2'b00	0
	srl	0000_00	00_0010	0	0	1	1	0	0	0	3'b101	0	2'b00	0
	jr	0000_00	00_1000	0	0	1	1	0	0	0	3'b000	1	2'b00	0
	multu	0000_00	01_1001	0	0	1	1	0	0	0	3'b011	0	2'b00	1
	mfhi	0000_00	01_0000	0	0	1	1	0	0	0	3'b000	0	2'b01	0
	mflo	0000_00	01_0010	0	0	1	1	0	0	0	3'b000	0	2'b11	0
J-type	jal	0000_11	xxxx	0	1	0	1	0	0	0	3'b000	0	2'b00	0
	j	0000_10	xxxx	0	1	0	1	0	0	0	3'b000	0	0	0
I-type	addi	0010_00	xxxx	0	0	0	1	1	0	0	3'b010	0	0	0
	lw	1000_11	xxxx	0	0	0	1	1	0	1	3'b010	0	0	0
	sw	1010_11	xxxx	0	0	0	0	1	1	0	3'b010	0	0	0
	beq	0001_00	xxxx	1	0	0	0	0	0	0	3'b110	0	0	0

Table1. Control Unite Table.

We also did the compared 5-stage pipeline design in hardware and software design for accelerated n!, and the result showed that the hardware design was 3 times faster than the software design. The final result will be shown in table 2 and graph 1:

Input	CC	
c	hardware	software
0	18	9
1	18	9
2	20	16
3	20	29
4	22	42
5	22	55
6	24	68
7	24	81
8	26	94
9	26	107
10	28	120
11	28	133
12	30	146
mean	23.54	69.92

Table 2. Clock Cycle Analysis between 5-stage pipeline and single clock cycle



Graph 1: Clock Cycle Analysis between 5-stage pipeline and single clock cycle

Task 3: Unit-level simulation waveforms

The simulation outcome was shown in Figure 2. We mainly rescued the code from the previous lab.

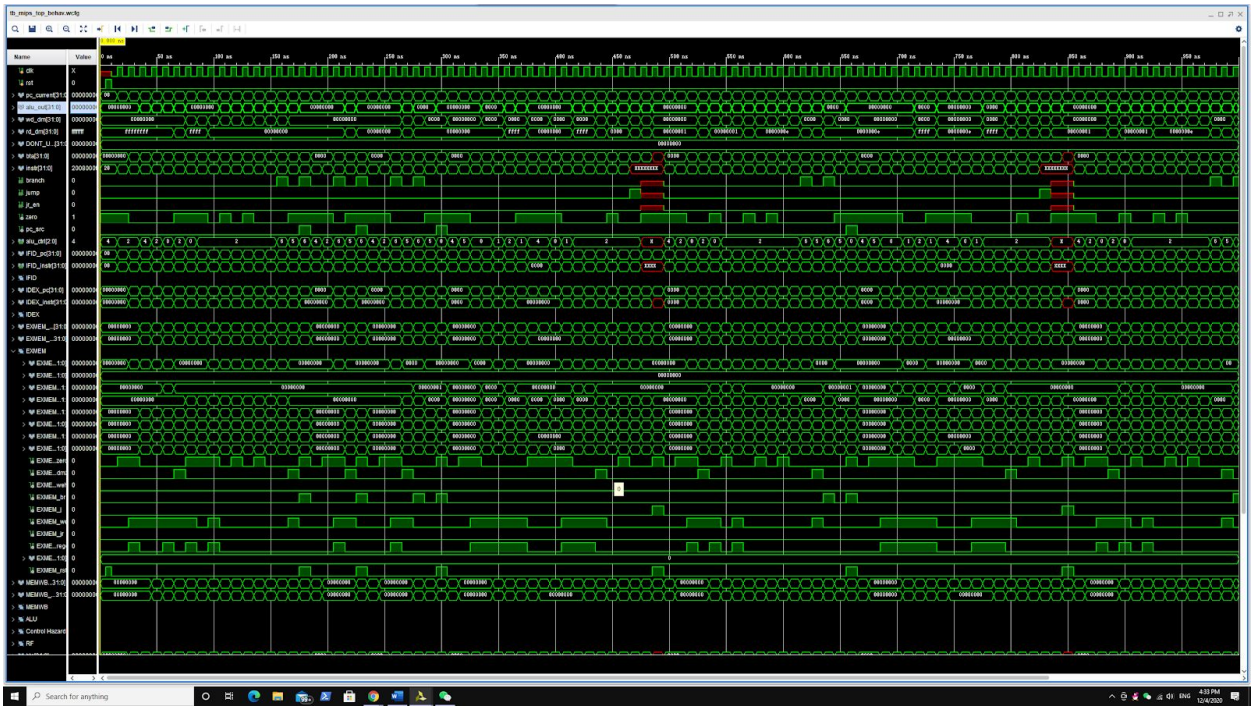


Figure 2 Final simulation outcome waveform

The whole simulation will start from pc_curr = 0 to pc_curr = 24. After that, the branch signal will be activated and it will jump to beq instruction which is instr = 01a96824, then it will

go back to the previous instruction by jr and keep operating. The details design was shown in Figure 3

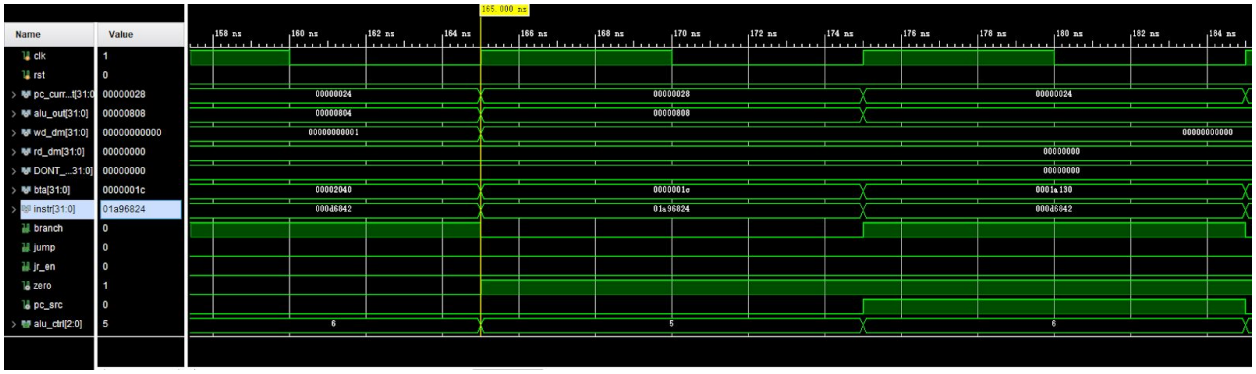


Figure 3: pc_curr = 24, branch jump

Task 4: FPGA validation

We also successfully implemented the validation on the FPGA board. Figure 4 showed the hi-bi for the outcome of n =12. The yellow rectangle label indicated the input bit and the hi/low switch. The blue part was used to rest and the red part was the processing button. Also, the input limitation was set to be 12, once the input number exceeded, the result will be freezed. In addition, all the output result will be presented as Hex number.

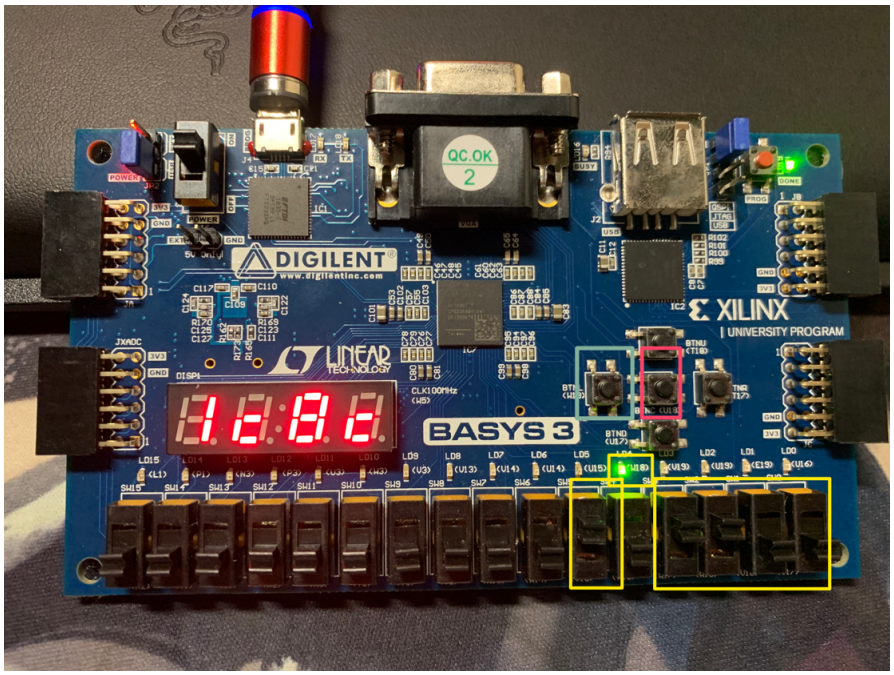


Figure4. Factorial Acceleration n = 12. (high part)

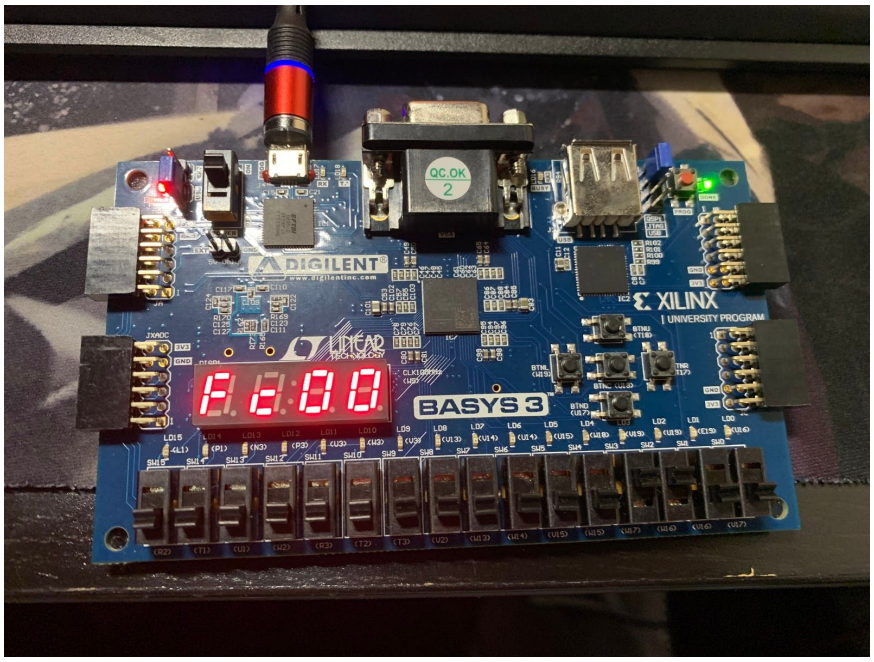


Figure 5. Factorial Acceleration n = 12. (low part)

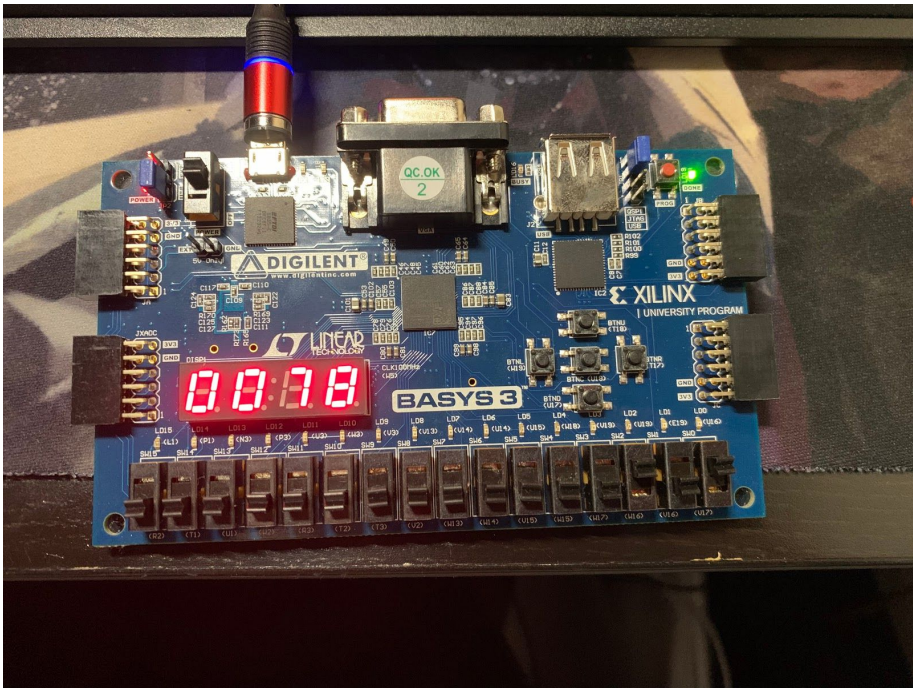


Figure 6. Factorial Acceleration n = 5.

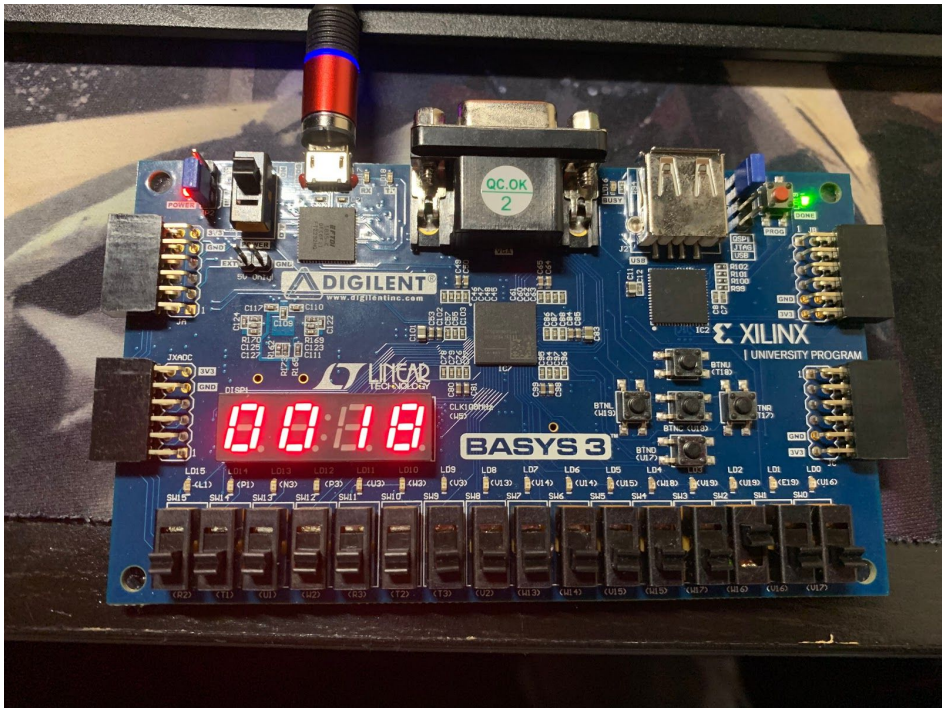


Figure 6. Factorial Acceleration n = 4.

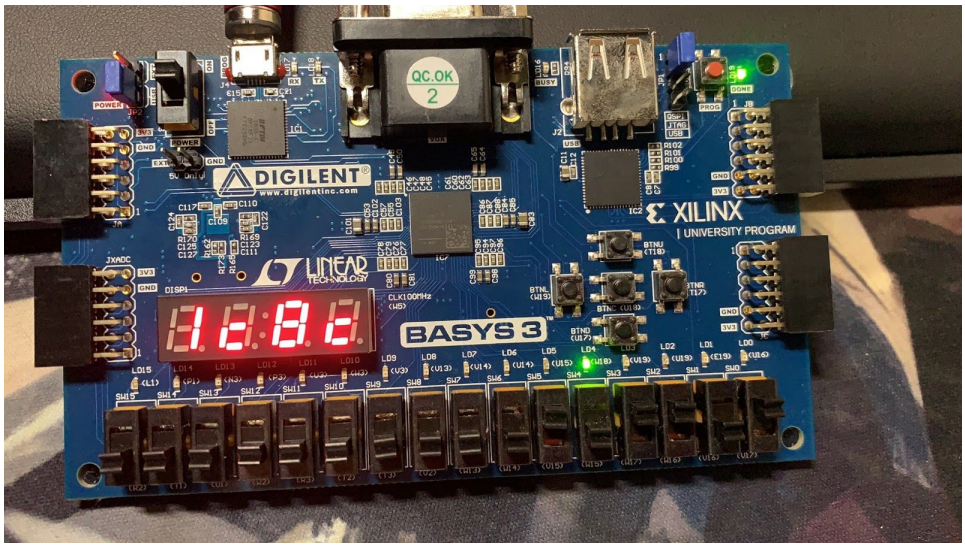


Figure 6. Factorial Acceleration n = 13, output answer freeze.

Successful Tasks

1. Extend single-cycle MIPS processor
2. Design diagrams for datapath and control unit
3. Create truth tables
4. Create test bench
5. Test via FPGA validation

Unsuccessful Tasks

None, lab was completed successfully.

Control Hazards

For control handling, we only handled the RAW, and load_use. For jump and branching, we just added 3 nops to it because we could not figure out how to change the hardware to support efficient handling.

For load use, the problem was that you would try to use a register that has loaded the instructions before. This is pretty much a read after write but instead of forwarding, you would need to delay one clock cycle so the lw instruction can finish the MEMWB part of the pipeline for the next instruction. The equation for this is: if(IDEX_instr == 6'b100011 (LW)) and IDEX_rt == IFID_rs or IFID_rt then you would stall for one clock cycle and NOP the IDEX stage. This will execute the IFID one clock cycle later and let the LW finish their cycle. The way we stalled it was that but we would cycle in Pc_current-4 back in the IFID so it would stay in the current address. This solution costs one clock cycle.

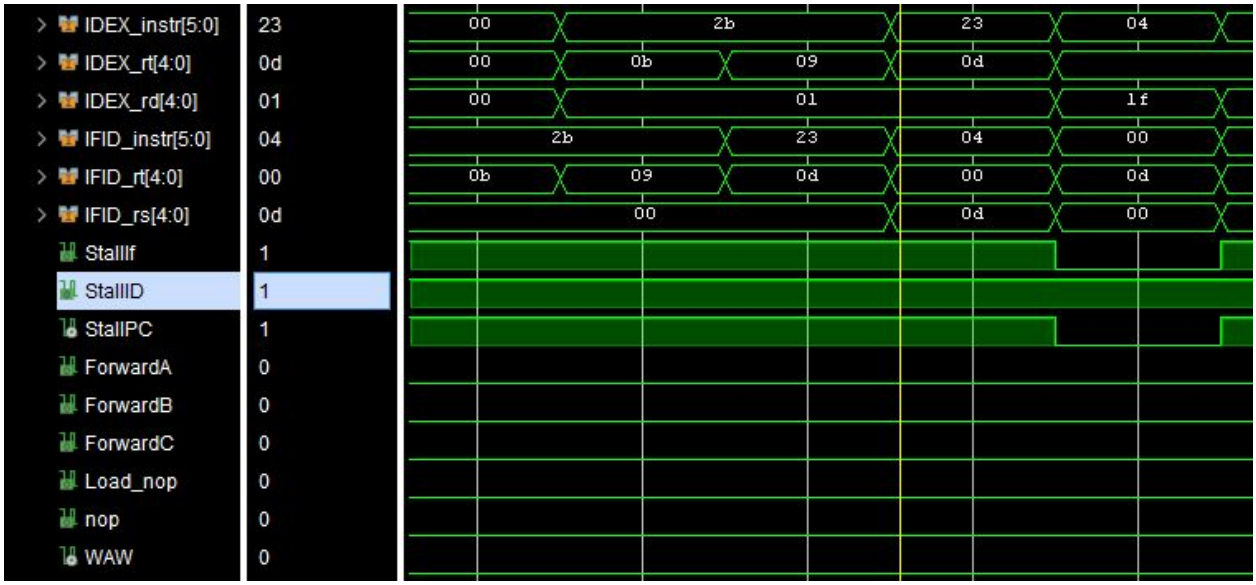


Figure 7: an example of the control hazard detecting the hazard. (Stalling is active low)

For read after write, we used data forwarding to handle this. We had to add 2 MUXs to the hardware in the beginning of the IDEX stage. One mux would handle the ALU_a input and the other would handle the ALU_b input. Each MUXs 1 choice was ALU_output and the 0 choice would be whatever was going into the ALU before. For forwarding, there are two different but similar scenarios, I-type and R-type. The reason why they are different is because I-type instructions store to register RT and R-type stores into RD. For R-type, we would forwardB when IDEX_rd == IFID_rt. We would forwardA when IDEX_rd == IFID_rs. We would not do this when IFID_instr is storeword because the storeword needs the value to come from somewhere else. If it is SW WAW = 1 which is just a badly named bus that forwards to

where the value for SW is needed. For I-type instructions, we would forwardB when IDEX_rt == IFID_rt and forwardA when IDEX_rt == IFID_rs.

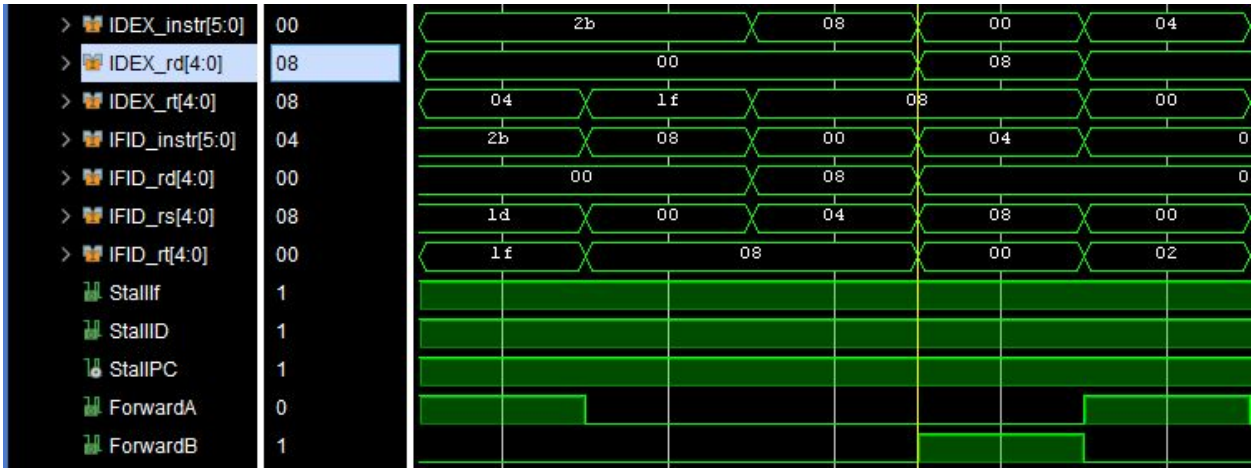


Figure 8. This shows an example of forwarding in factorial.asm (20-28)

Conclusion

Overall, we were able to complete this lab after several weeks of tasks and demos. Since this lab is the last lab of the semester we anticipated that it would challenge us and be the most difficult. One of the road blocks we faced was dealing with how to handle branch hazards, which we ultimately solved by reaching out to the professor for tips. Other than that, we were able to follow the tasks required for each of the four weeks that we had to complete this lab. We started by creating drafts of our pipelined MIPS microarchitecture and our SoC interface. Once we had those we created tables for the MIPS control unit and memory maps for our SoC interface. We followed that up by doing a performance analysis of hardware accelerated n!, and producing unit-level simulation waveforms. Lastly, we were able to complete the interface design for our SoC with the single-cycle MIPS processor, and the factorial unit and the simple GPIO. On our last week to complete the lab we finished by showing a demo of the completed integration of the SoC using a pipelined MIPS processor. We were able to demo our upgraded SoC on our FPGA board.

Appendix

Datapath.v (Pretty much changed 80%)

```
module datapath (
        input  wire        clk,
        input  wire        rst,
        input  wire        branch,
        input  wire        jump,
        input  wire        reg_dst,
        input  wire        we_reg,
        input  wire        alu_src,
        input  wire        dm2reg,
        input  wire        we_hilo, shift,
        input  wire        jr_en,
        input  wire [1:0]  mf_sel,
        input  wire [2:0]  alu_ctrl,
        input  wire [4:0]  ra3,
        input  wire [31:0] instr,
        input  wire [31:0] rd_dm,
        output wire [31:0] pc_current,
        output wire [31:0] alu_out,
        output wire [31:0] wd_dm,
        output wire [31:0] rd3, IFID_instr,
        output reg         noop
    );

    wire [4:0]  rf_wa, rtype_rf;
    wire        pc_src;
    wire [31:0] pc_plus4, pc_minus4, pc_stall;
    wire [31:0] pc_plus8;
    wire [31:0] pc_pre;
    wire [31:0] pc_jmp;
    wire [31:0] pc_next;
    wire [31:0] sext_imm;
    wire [31:0] ba;
    wire [31:0] bta;
    wire [31:0] jta;
    wire [31:0] alu_pa;
    wire [31:0] alu_pb;
    wire [31:0] wd_rf;
    wire        zero;
    wire [31:0] alu_high;
```

```
    wire [31:0] shiftmux;
    wire [31:0] hi_out, lo_out, hilo_mux_out, wd_rf_b4, wdrf_mux_out
,rd2, rd1;

    //pipelining wires
    wire [31:0] IFID_pc;
    wire [31:0] IDEX_rd1, IDEX_rd2, IDEX_rd3, IDEX_sext, IDEX_sm,
IDEX_instr, IDEX_pc, IFID_nop;
    wire [4:0] IDEX_rfr , EXMEM_rfr , MEMWB_rfr;
    wire [3:0]  IDEX_jpc;
    wire [31:0] EXMEM_aluo, EXMEM_aluh, EXMEM_rd2, EXMEM_bta,
EXMEM_pc, EXMEM_instr, EXMEM_sext, EXMEM_rd1 , EDXMEM_jr, EXMEMTEMP;
    wire        EXMEM_zero;
    wire [31:0] MEMWB_aluo, MEMWB_rddm, MEMWB_hi, MEMWB_lo, MEMWB_pc,
MEMWB_instr;

    //Controls
    wire        IDEX_br , IDEX_j , IDEX_regd , IDEX_wer , IDEX_alus ,
IDEX_dm2r , IDEX_weh, IDEX_jr;
    wire [1:0]  IDEX_mfs;
    wire [2:0]  IDEX_aluc;

    wire        EXMEM_dm2r, EXMEM_weh, EXMEM_br, EXMEM_j, EXMEM_wer,
EXMEM_jr, EXMEM_regd;
    wire [1:0]  EXMEM_mfs;

    wire        MEMWB_dm2r, MEMWB_j, MEMWB_wer;
    wire [1:0]  MEMWB_mfs;

    //hazards
    reg Load_nop;
    wire        StallIf, StallID, ForwardA, ForwardB , ForwardC,
StallPC, Load_nop2, Load_nop3, WAW, StallEX, StallPC2;
    wire [31:0] ForwardedA, ForwardedB,ForwardedC, WAW_forwarded,
pc_staal;
    wire        MEMWB_rst, EXMEM_rst, IDEX_rst,IFID_rst, nop;

    assign pc_src = EXMEM_br & EXMEM_zero;
    assign ba = {IDEX_sext, 2'b00};
```

```
    assign jta = {EXMEM_pc[31:28], EXMEM_instr[25:0], 2'b00};

    // --- PC Logic --- //
    Reg pc_reg (
            .clk            (clk),
            .rst            (rst),
            .LD             (!Load_nop & !Load_nop2 & !Load_nop3),
            .D              (pc_staal),
            .Q              (pc_current)
        );

    adder pc_plus_4 (
            .a              (pc_current),
            .b              (32'd4),
            .y              (pc_plus4)
        );
    adder pc_plus_8 (
            .a              (pc_current),
            .b              (32'hFFFF_FFFC),
            .y              (pc_plus8)
        );

    adder pc_plus_br (
            .a              (IDEX_pc),
            .b              (ba),
            .y              (bta)
        );

    mux2 #(32) pc_src_mux (
            .sel            (pc_src),
            .a              (pc_plus4),
            .b              (EXMEM_bta),
            .y              (pc_pre)
        );

    mux2 #(32) pc_jmp_mux (
            .sel            (EXMEM_j),
            .a              (pc_pre),
            .b              (jta),
```

```
        .y              (pc_jmp)
    );
//new
mux2 #(32) pc_jr_mux (
        .sel            (EXMEM_jr),
        .a              (pc_jmp),
        .b              (EXMEM_rd1),
        .y              (pc_next)
    );
mux2 #(32) stallPC_mux (
        .sel            (!StallPC),
        .a              (pc_next),
        .b              (pc_plus8),
        .y              (pc_staal)
    );
 //end
// --- RF Logic --- //
mux2 #(5) rf_wa_mux (
        .sel            (EXMEM_regd),
        .a              (EXMEM_instr[20:16]),
        .b              (EXMEM_instr[15:11]),
        .y              (rtype_rf)
    );
//new
mux2 #(5) jal_rf_wa_mux(
        .sel            (EXMEM_j),
        .a              (rtype_rf),
        .b              (5'b11111),
        .y              (rf_wa)
    );
 mux2 #(32) ForwardingC (
    .sel            (ForwardC),
    .a              (rd2),
    .b              (MEMWB_rddm),
    .y              (ForwardedC)
);
//end
regfile rf (
        .clk            (clk),
```

```
            .we             (MEMWB_wer),
            .ra1            (IFID_instr[25:21]),
            .ra2            (IFID_instr[20:16]),
            .ra3            (ra3),
            .wa             (MEMWB_rfr),
            .wd             (wd_rf),
            .rd1            (rd1),
            .rd2            (rd2),
            .rd3            (rd3),
            .rst            (rst)
        );

    signext se (
            .a              (IFID_instr[15:0]),
            .y              (sext_imm)
        );


    // --- ALU Logic --- //
   //ADDED NEW THING
    mux2 #(32) alu_shift_mux (
            .sel            (shift),
            .a              (rd1),
            .b              ({27'b0, IFID_instr[10:6]}),
            .y              (shiftmux)
        );

   mux2 #(32) ForwardingA (
        .sel            (ForwardA),
        .a              (IDEX_rd1),
        .b              (alu_out),
        .y              (ForwardedA)
    );

   mux2 #(32) ForwardingB (
        .sel            (ForwardB),
        .a              (alu_pb),
        .b              (alu_out),
        .y              (ForwardedB)
```

```
    );

   //NEW THING END
    mux2 #(32) alu_pb_mux (
            .sel            (IDEX_alus),
            .a              (IDEX_rd2),
            .b              (IDEX_sext),
            .y              (alu_pb)
        );

    alu alu (
            .op             (IDEX_aluc),
            .a              (ForwardedA),
            .b              (ForwardedB),
            .zero           (zero),
            .y              ({alu_high,EXMEM_aluo})
        );

    // --- MEM Logic --- //
    mux2 #(32) rf_wd_mux (
            .sel            (MEMWB_dm2r),
            .a              (MEMWB_aluo),
            .b              (MEMWB_rddm),
            .y              (wd_rf_b4)
        );

    //NEW
    reg_w_en #(32) HI(
            .clk            (clk),
            .en             (EXMEM_weh),
            .d              (EXMEM_aluh),
            .q              (hi_out)
        );
    reg_w_en #(32) LO(
            .clk            (clk),
            .en             (EXMEM_weh),
            .d              (alu_out),
            .q              (lo_out)
        );
```

```
    mux2 #(32) hilo_mux(
            .sel            (MEMWB_mfs[1]),
            .a              (MEMWB_hi),
            .b              (MEMWB_lo),
            .y              (hilo_mux_out)
        );

    mux2 #(32) wd_rf_mux(
            .sel            (MEMWB_mfs[0]),
            .a              (wd_rf_b4),
            .b              (hilo_mux_out),
            .y              (wdrf_mux_out)
        );

    mux2 #(32) wdrf_mux (
            .sel            (MEMWB_j),
            .a              (wdrf_mux_out),
            .b              (MEMWB_pc),
            .y              (wd_rf)
        );

   mux2 #(32) SWERROR(
            .sel            (WAW),
            .a              (IDEX_rd2),
            .b              (alu_out),
            .y              (WAW_forwarded)
   );
  mux2 #(32) RD2ERROR(
            .sel            (ForwardC),
            .a              (WAW_forwarded),
            .b              (ForwardedC),
            .y              (EXMEMTEMP)
   );
    //NEW END

    // pipelining things
    //DP signals
    Reg IFID_PC (.D (pc_plus4), .Q (IFID_pc), .clk(clk), .LD(1'b1),
.rst(IFID_rst));
```

```
    Reg IFID_Instr (.D (IFID_nop) , .Q(IFID_instr), .clk(clk),
.LD(1'b1), .rst(IFID_rst));

    Reg IDEX_Rd1 (.D(shiftmux), .Q(IDEX_rd1) ,.clk(clk),
.LD(StallID), .rst(IDEX_rst));
    Reg IDEX_Rd2 (.D(rd2), .Q(IDEX_rd2) ,.clk(clk), .LD(StallID),
.rst(IDEX_rst));
    Reg IDEX_Rd3 (.D(rd3), .Q(IDEX_rd3) ,.clk(clk), .LD(StallID),
.rst(IDEX_rst));
    Reg IDEX_Sext (.D(sext_imm), .Q(IDEX_sext) ,.clk(clk),
.LD(StallID), .rst(IDEX_rst));
    Reg IDEX_SMUX (.D(shiftmux), .Q(IDEX_sm) ,.clk(clk),
.LD(StallID), .rst(IDEX_rst));
    Reg IDEX_PC (.D(IFID_pc), .Q(IDEX_pc) ,.clk(clk), .LD(StallID),
.rst(IDEX_rst));
    Reg IDEX_Instr (.D (IFID_instr) , .Q(IDEX_instr), .clk(clk),
.LD(StallID | Load_nop | Load_nop2 | Load_nop3), .rst(IDEX_rst |
!StallEX));


    Reg EXMEM_PC(.D(IDEX_pc), .Q(EXMEM_pc), .clk(clk), .LD(1'b1),
.rst(EXMEM_rst));
    Reg EXMEM_ALUO (.D(EXMEM_aluo), .Q(alu_out), .clk(clk),
.LD(1'b1), .rst(EXMEM_rst));
    Reg EXMEM_ALUH (.D(alu_high), .Q(EXMEM_aluh), .clk(clk),
.LD(1'b1), .rst(EXMEM_rst));
    Reg #(1) EXMEM_ALUZ (.D(zero), .Q(EXMEM_zero), .clk(clk),
.LD(1'b1), .rst(EXMEM_rst));
    Reg EXMEM_Rd2 (.D(EXMEMTEMP), .Q(EXMEM_rd2), .clk(clk),
.LD(1'b1), .rst(EXMEM_rst));
    Reg EXMEM_seXT(.D(IDEX_sext), .Q(EXMEM_sext), .clk(clk),
.LD(1'b1), .rst(EXMEM_rst));
    Reg EXMEM_BTA(.D(bta), .Q(EXMEM_bta), .clk(clk), .LD(1'b1),
.rst(EXMEM_rst));
    Reg EXMEM_Instr (.D(IDEX_instr), .Q(EXMEM_instr), .clk(clk),
.LD(1'b1), .rst(EXMEM_rst));
    Reg EXMEM_RD1 (.D(IDEX_rd1), .Q(EXMEM_rd1), .clk(clk), .LD(1'b1),
.rst(EXMEM_rst));
    assign wd_dm = EXMEM_rd2;
```

```
    Reg MEMWB_PC(.D(EXMEM_pc), .Q(MEMWB_pc), .clk(clk), .LD(1'b1),
.rst(MEMWB_rst));
    Reg MEMWB_ALUO (.D(alu_out), .Q(MEMWB_aluo), .clk(clk),
.LD(1'b1), .rst(MEMWB_rst));
    Reg MEMWB_RDDM (.D(rd_dm),.Q(MEMWB_rddm), .clk(clk), .LD(1'b1),
.rst(MEMWB_rst));
    Reg MEMWB_HI (.D(hi_out),.Q(MEMWB_hi), .clk(clk), .LD(1'b1),
.rst(MEMWB_rst));
    Reg MEMWB_LO (.D(lo_out),.Q(MEMWB_lo), .clk(clk), .LD(1'b1),
.rst(MEMWB_rst));
    Reg MEMWB_IInstr(.D(EXMEM_instr), .Q(MEMWB_instr), .clk(clk),
.LD(1'b1), .rst(MEMWB_rst));
    Reg #(5) MEMWB_RFR (.D(rf_wa), .Q(MEMWB_rfr) ,.clk(clk),
.LD(1'b1), .rst(MEMWB_rst));

    //CU signals
    Reg #(.WIDTH(1'b1)) IDEX_BR (.D (branch), .Q (IDEX_br),
.clk(clk), .LD(StallID), .rst(IDEX_rst | !StallEX));
    Reg #(.WIDTH(1'b1)) IDEX_REGD  (.D (reg_dst) , .Q(IDEX_regd),
.clk(clk), .LD(StallID), .rst(IDEX_rst | !StallEX));
    Reg #(.WIDTH(1'b1)) IDEX_WER  (.D (we_reg) , .Q(IDEX_wer),
.clk(clk), .LD(StallID), .rst(IDEX_rst | !StallEX ));
    Reg #(.WIDTH(1'b1)) IDEX_ALUS  (.D (alu_src) , .Q(IDEX_alus),
.clk(clk), .LD(StallID), .rst(IDEX_rst | !StallEX));
    Reg #(.WIDTH(1'b1)) IDEX_DM2R  (.D (dm2reg) , .Q(IDEX_dm2r),
.clk(clk), .LD(StallID), .rst(IDEX_rst | !StallEX));
    Reg #(.WIDTH(1'b1)) IDEX_HL  (.D (we_hilo) , .Q(IDEX_weh),
.clk(clk), .LD(StallID), .rst(IDEX_rst | !StallEX));
    Reg #(.WIDTH(2'b10)) IDEX_MFS  (.D (mf_sel) , .Q(IDEX_mfs),
.clk(clk), .LD(StallID), .rst(IDEX_rst | !StallEX));
    Reg #(.WIDTH(2'b11)) IDEX_ALUC  (.D (alu_ctrl) , .Q(IDEX_aluc),
.clk(clk), .LD(StallID), .rst(IDEX_rst | !StallEX));
    Reg #(.WIDTH(1'b1)) IDEX_J  (.D (jump) , .Q(IDEX_j), .clk(clk),
.LD(StallID), .rst(IDEX_rst | !StallEX));
    Reg #(.WIDTH(1'b1)) IDEX_JR  (.D (jr_en) , .Q(IDEX_jr),
.clk(clk), .LD(StallID), .rst(IDEX_rst | !StallEX));
```

```
    Reg #(.WIDTH(1'b1)) EXMEM_DM2R  (.D (IDEX_dm2r) , .Q(EXMEM_dm2r),
.clk(clk), .LD(1'b1), .rst(EXMEM_rst));
    Reg #(.WIDTH(1'b1)) EXMEM_BR  (.D (IDEX_br) , .Q(EXMEM_br),
.clk(clk), .LD(1'b1), .rst(EXMEM_rst));
    Reg #(.WIDTH(1'b1)) EXMEM_WER  (.D (IDEX_wer) , .Q(EXMEM_wer),
.clk(clk), .LD(1'b1), .rst(EXMEM_rst));
    Reg #(.WIDTH(1'b1)) EXMEM_HILO  (.D (IDEX_weh) , .Q(EXMEM_weh),
.clk(clk), .LD(1'b1), .rst(EXMEM_rst));
    Reg #(.WIDTH(2'b10)) EXMEM_MFS  (.D (IDEX_mfs) , .Q(EXMEM_mfs),
.clk(clk), .LD(1'b1), .rst(EXMEM_rst));
    Reg #(.WIDTH(1'b1)) EXMEM_J  (.D (IDEX_j) , .Q(EXMEM_j),
.clk(clk), .LD(1'b1), .rst(EXMEM_rst));
    Reg #(.WIDTH(1'b1)) EXMEM_JR  (.D (IDEX_jr) , .Q(EXMEM_jr),
.clk(clk), .LD(1'b1), .rst(EXMEM_rst));
    Reg #(.WIDTH(1'b1)) EXMEM_REGD  (.D (IDEX_regd) , .Q(EXMEM_regd),
.clk(clk), .LD(1'b1), .rst(EXMEM_rst));

    Reg #(.WIDTH(1'b1)) MEMWB_WER  (.D (EXMEM_wer) , .Q(MEMWB_wer),
.clk(clk), .LD(1'b1), .rst(MEMWB_rst));
    Reg #(.WIDTH(1'b1)) MEMWB_DM2R  (.D (EXMEM_dm2r) ,
.Q(MEMWB_dm2r), .clk(clk), .LD(1'b1), .rst(MEMWB_rst));
    Reg #(.WIDTH(2'b10)) MEMWB_MFS  (.D (EXMEM_mfs) , .Q(MEMWB_mfs),
.clk(clk), .LD(1'b1), .rst(MEMWB_rst));
    Reg #(.WIDTH(1'b1)) MEMWB_J  (.D (EXMEM_j) , .Q(MEMWB_j),
.clk(clk), .LD(1'b1), .rst(MEMWB_rst));

    assign MEMWB_rst = rst;
    assign EXMEM_rst = EXMEM_jr | EXMEM_j | pc_src | rst;
    assign IDEX_rst =  EXMEM_jr |EXMEM_j | pc_src | rst;
    assign IFID_rst =  EXMEM_jr |EXMEM_j | pc_src |rst;
    mux2 #(32) Load_nNOP (
            .sel            (!StallIf),
            .a              (instr),
            .b              (IDEX_instr),
            .y              (IFID_nop)
        );
     always@(posedge clk) begin
        noop<= 0;
        end
```

```
    Reg #(.WIDTH(1'b1)) delay (.D (Load_nop) , .Q(Load_nop2),
.clk(clk), .LD(1'b1), .rst(1'b0));
    Reg #(.WIDTH(1'b1)) delay2 (.D (Load_nop2) , .Q(Load_nop3),
.clk(clk), .LD(1'b1), .rst(1'b0));
    Reg #(.WIDTH(1'b1)) delayP2 (.D (StallPC) , .Q(StallPC2),
.clk(clk), .LD(1'b1), .rst(1'b0));
    initial Load_nop <= 0;
    control_hazards HAZARDS
(.clk(clk),.IDEX_instr(IDEX_instr[31:26]),
.IDEX_rt(IDEX_instr[20:16]), .IDEX_rd(IDEX_instr[15:11]),
.IFID_rt(IFID_instr[20:16]), .MEMWB_instr(MEMWB_instr[31:26]),
                             .IFID_rs(IFID_instr[25:21]),
.IFID_rd(IFID_instr[15:11]), .StallIf(StallIf),
.StallID(StallID),.StallPC(StallPC), .ForwardA(ForwardA),
.ForwardB(ForwardB), .WAW(WAW),
                             .IFID_instr(IFID_instr[31:26]),
.nop(nop), .MEMWB_rt(MEMWB_instr[20:16]),
.MEMWB_rd(MEMWB_instr[15:11]), .StallEx(StallEX),
.ForwardC(ForwardC), .EXMEM_instr(EXMEM_instr[31:26]),
                             .EXMEM_rt(EXMEM_instr[20:16]));
endmodule
```

Control_hazards.v (new file)

```
module control_hazards(
    input   wire    [4:0]   IDEX_rt, IDEX_rd, IFID_rt, IFID_rs,
IFID_rd, MEMWB_rt, MEMWB_rd, EXMEM_rt,
    input   wire    [5:0]   IDEX_instr, IFID_instr, MEMWB_instr,
EXMEM_instr,
    input   wire            clk,
    output  reg             StallIf, StallID, ForwardA,
ForwardB,ForwardC, WAW, nop, StallPC, Load_nop, StallEx
    );
    initial begin
    StallIf <= 1;
    StallID <= 1; //STALLING IS ACTIVE LOW
    StallPC <= 1;
```

```
    StallEx <= 1;
    Load_nop <= 0;
    ForwardA <= 1'b0;
    ForwardB <= 1'b0;
    ForwardC <= 0;
    nop <= 0;
    WAW <= 0;
    end
    always@ (posedge clk) begin
        if(WAW) WAW <= 0;
        if(StallIf == 0 || Load_nop) begin
           StallIf <= 1'b1;
           StallID <= 1'b1;
           StallPC <= 1'b1;
           StallEx <= 1'b1;
           Load_nop <= 0;
        end
        if(ForwardB | ForwardA | ForwardC) begin
            ForwardA <= 1'b0;
            ForwardB <= 1'b0;
            ForwardC <= 1'b0;
            end
//        if(IDEX_rt != 0 && IFID_rt != 0 && IFID_rd != 0 && IDEX_rd
!= 0 )
//            begin
            case(IDEX_instr)  //RAW and WAR
                6'b10_0011: begin // LOADWORD
                $display("LW IDEX : %b IFID: %b or %b", IDEX_rt,
IFID_rt, IFID_rs);
                    if((IDEX_rt == IFID_rt) || (IDEX_rt == IFID_rs))
begin //IF register loading from loadword is any of the operands of
next operation
                        $display("STALLED");
                        StallIf <=  0;
                        StallPC <= 0;
                        StallEx <= 0;
                    end
                    else begin
                        StallIf <= 1'b1;
```

```
                        StallID <= 1'b1;
                    end
                end
                6'b00_0000: begin //RTYPE
                $display("RTYPE IDEX : %b IFID: %b or %b", IDEX_rd,
IFID_rt, IFID_rs);
                    if((IDEX_rd == IFID_rt) && !(IFID_instr ==
6'b101011))begin //IF register loading from loadword is any of the
operands of next operation
                        ForwardA <= 1'b0;
                        ForwardB <= 1'b1;
                    end
                    if ((IDEX_rd == IFID_rs)&& !(IFID_instr ==
6'b101011)) begin
                        ForwardA <= 1'b1;
                        ForwardB <= 1'b0;
                    end
                    if((IDEX_rd == IFID_rd) || (IDEX_rd == IFID_rt))
begin // RAW ERROR W/SW
                        WAW <= 1;
                    end
                    if((IDEX_rd != IFID_rs) && (IDEX_rd != IFID_rt))
begin
                        ForwardA <= 1'b0;
                        ForwardB <= 1'b0;
                    end
                end
                6'b00_1000: begin //ITYPE
                $display("ITYPE IDEX : %b IFID: %b or %b", IDEX_rt,
IFID_rt, IFID_rs);
                    if((IDEX_rt == IFID_rt))begin //IF register
loading from loadword is any of the operands of next operation
                        ForwardA <= 1'b0;
                        ForwardB <= 1'b1;
                    end
                    if ((IDEX_rt == IFID_rs)) begin
                        ForwardA <= 1'b1;
                        ForwardB <= 1'b0;
                    end
```

```
                    if ((IDEX_rt != IFID_rs)&&(IDEX_rt !=
IFID_rt))begin
                        ForwardA <= 1'b0;
                        ForwardB <= 1'b0;
                    end
                end
                default: begin
                    StallIf <= 1'b1;
                    StallID <= 1'b1;
                    ForwardA <= 1'b0;
                    ForwardB <= 1'b0;
                end
            endcase

            case(EXMEM_instr)
                6'b10_0011: begin // LOADWORD
                    case(IFID_instr)
                        6'b101011: begin
                            if(IFID_rt == EXMEM_rt) ForwardC <= 1;
                        end
                    endcase
                end
            default:
                ForwardC <= 0;
            Endcase
            if(MEMWB_rd != 0) begin
            case(MEMWB_instr)
                6'b00_0000: begin
                    if(MEMWB_rd == IFID_rs || MEMWB_rd == IFID_rt)
begin
                   $display("STALLED");
                        StallIf <=  0;
                        StallPC <= 0;
                        StallEx <= 0;
                    end
                    else begin
                        StallIf <= 1'b1;
                        StallID <= 1'b1;
                    end
```

```
                end
                endcase
                end
        end
//    end
endmodule
```

Reg_load.v (new)

```
`timescale 1ns / 1ps
module reg_load #(parameter WIDTH = 32)(
    input wire           clk, rst, load,
    input wire  [WIDTH-1:0] D,
    output reg  [WIDTH-1:0] Q
    );

    always @(posedge clk, posedge rst)
    if(rst)
        Q <= 0;
    else if(load)
        Q <= D;
    else
        Q <= Q;
endmodule
```

Gpio_ad.v (new)

```
`timescale 1ns / 1ps

module gpio_ad(
    input wire  [1:0]   A,
    input wire          WE,
    output reg         WE1, WE2,
    output wire [1:0]   RdSel
    );
```

```
        always @(*) begin
        case (A)
        2'b00: begin    //Read input 1
            WE1 = 1'b0;
            WE2 = 1'b0;
        end
        2'b01: begin  //Read input 2
            WE1 = 1'b0;
            WE2 = 1'b0;
        end
        2'b10: begin  //Read and/or write input1
            WE1 = WE;
            WE2 = 1'b0;
        end
        2'b11: begin  //Read and/or write input2
            WE1 = 1'b0;
            WE2 = WE;
        end
        default: begin
            WE1 = 1'bx;
            WE2 = 1'bx;
            end
        endcase
    end
    assign RdSel = A;
endmodule
```

Gpio_top.v (new)

```
`timescale 1ns / 1ps

module gpio_top(
    input wire  [1:0]   A,
    input wire          WE, clk, rst,
    input wire  [31:0]  gpI1, gpI2,
    input wire  [31:0]   WD,
    output wire  [31:0]  RD,
```

```
    output wire [31:0] gpO1, gpO2
    );

    wire            WE1, WE2;
    wire    [1:0]   RdSel;

    gpio_ad address_decoder(.A(A), .WE(WE), .WE1(WE1), .WE2(WE2),
.RdSel(RdSel));
    mux4_1 selector (.zero(gpI1), .one(gpI2), .two({28'b0,gpO1}),
.three({28'b0,gpO2}), .sel(RdSel), .out(RD));
    reg_load  gp1Reg (.clk(clk), .rst(rst), .load(WE1), .D(WD),
.Q(gpO1));
    reg_load  gp2Reg (.clk(clk), .rst(rst), .load(WE2), .D(WD),
.Q(gpO2));
endmodule
```

Fact_ad.v (new)

```
`timescale 1ns / 1ps

module fact_ad(
    input wire  [1:0]   A,
    input wire          WE,
    output reg          WE1, WE2,
    output wire [1:0]   RdSel
    );

    always @(*) begin
        case (A)
        2'b00: begin    //Read/Write into Factorial. Input N [3:0],
output N!
            WE1 = WE;
            WE2 = 1'b0;
        end
        2'b01: begin  // Read/write the GO bit. Output the GO thing.
            WE1 = 1'b0;
            WE2 = WE;
```

```
        end
        2'b10: begin  //only read Control Outputs (Done, Err.)
            WE1 = 1'b0;
            WE2 = 1'b0;
        end
        2'b11: begin  //only read result (N!)
            WE1 = 1'b0;
            WE2 = 1'b0;
        end
        default: begin
            WE1 = 1'bx;
            WE2 = 1'bx;
            end
        endcase
    end
    assign RdSel = A;
endmodule
```

Fact_top.v (new)

```
`timescale 1ns / 1ps
module fact_top(
    input wire   [1:0]  A,
    input               WE, rst, clk,
    input wire   [3:0]  WD,
    output wire   [31:0] RD
    );

    reg         ResErr, ResDone;
    wire        WE1, WE2, Go, GoPulse,  Done, Err, GoPulseCmb;
    wire [31:0] nf, Result;
    wire [3:0]  n;
    wire [1:0]  RdSel;

    fact_ad Address_Decorder(.A(A), .WE(WE), .WE1(WE1), .WE2(WE2),
.RdSel(RdSel));
    fact    Factorial_Accelerator(.RST(rst), .CLK(clk), .GO(GoPulse),
```

```
.IN1(n), .DONE(Done),
                                .ERROR(Err), .UPPER(nf[31:16]),
.LOWER(nf[15:0]));

    reg_load #(32)inputNF (.clk(clk), .rst(rst), .load(Done), .D(nf),
.Q(Result));
    reg_load #(4) inputWd (.clk(clk), .rst(rst), .load(WE1), .D(WD),
.Q(n));
    reg_load #(1) inputGo (.clk(clk), .rst(rst), .load(WE2),
.D(WD[0]), .Q(Go));
    reg_load #(1) inputGoPulse (.clk(clk), .rst(rst), .load(1'b1),
.D(GoPulseCmb), .Q(GoPulse));
    mux4_1 ReadSelect (.zero({28'b0,n}), .one({31'b0,Go}),
.two({30'b0, ResErr, ResDone}), .three(Result), .sel(RdSel),
.out(RD));


    assign GoPulseCmb = WD[0] & WE2;
    always @(posedge clk, posedge rst)
    begin
        if(rst)
            ResDone <= 1'b0;
        else
            ResDone <= (!GoPulseCmb) & (Done | ResDone);
    end

    always @(posedge clk, posedge rst)
    begin
        if(rst)
            ResErr <= 1'b0;
        else
            ResErr <= (!GoPulseCmb) & (Err | ResErr);
    end
endmodule
```

Mips_top_ad.v (new)

```
`timescale 1ns / 1ps
module mips_top_ad(
    input   wire          WE,
    input   wire  [31:0]  A,
    output  reg         WE2,WE1, WEM,
    output  reg   [1:0]   RdSel
    );
    always @(*) begin
        case(A[11:8])
        4'b0000: begin
            WEM = WE & !(A[2]& A[3]& A[4]& A[5]& A[6]& A[7]);
            WE1 = 1'b0;
            WE2 = 1'b0;
            RdSel = 2'b00;
            end
        4'b1000: begin
            WEM = 1'b0;
            WE1 = WE & !(A[4]& A[5]& A[6]& A[7]);
            WE2 = 1'b0;
            RdSel = 2'b10;
            end
        4'b1001: begin
            WEM = 1'b0;
            WE1 = 1'b0;
            WE2 = WE & !(A[4]& A[5]& A[6]& A[7]);
            RdSel = 2'b11;
            end
        default: begin
            WEM = 1'bx;
            WE1 = 1'bx;
            WE2 = 1'bx;
            RdSel = 2'bxx;
            end
        endcase
    end
endmodule
```

Mips_top.v (modified)

```
module mips_top (
        input  wire        clk,
        input  wire        rst,
        input  wire [4:0]  ra3,
        input  wire [31:0] GPI2, GPI1,
        output wire        we_dm,
        output wire [31:0] pc_current,
        output wire [31:0] instr,
        output wire [31:0] alu_out,
        output wire [31:0] writeData,
        output wire [31:0] readData,
        output wire [31:0] rd3,
        output wire [31:0] GPO1, GPO2
    );

    wire [31:0] DONT_USE, rd_dm, FactData, GPIOData;
    wire        WE1,WE2, WEM, tempwe_dm;
    wire [1:0]  RdSel;

    wire        IDEX_wed, EXMEM_wed;
    mips mips (
            .clk            (clk),
            .rst            (rst),
            .ra3            (ra3),
            .instr          (instr),
            .rd_dm          (readData),
            .we_dm          (tempwe_dm),
            .pc_current     (pc_current),
            .alu_out        (alu_out),
            .wd_dm          (writeData),
            .rd3            (rd3)
        );

    imem imem (
            .a              (pc_current[7:2]),
            .y              (instr)
        );
```

```
    dmem dmem (
            .clk            (clk),
            .we             (we_dm),
            .a              (alu_out[7:2]),
            .d              (writeData),
            .q              (rd_dm),
            .rst            (rst)
        );

    //NEWWW
    mips_top_ad Address_decoder (.A(alu_out), .WE(we_dm),.WE1(WE1),
.WE2(WE2), .WEM (WEM), .RdSel(RdSel));
    mux4_1 MemoryChooser (.zero(rd_dm), .one(rd_dm), .two(FactData),
.three(GPIOData), .sel(RdSel), .out(readData));

    fact_top Factorial_Accelerator (.rst(rst), .clk(clk),
.A(alu_out[3:2]), .WE(WE1), .WD(writeData[3:0]),  .RD(FactData));
    gpio_top Gen_purp_IO (.rst(rst), .clk(clk), .A(alu_out[3:2]),
.WE(WE2), .WD(writeData),  .RD(GPIOData), .gpI1(GPI1) ,.gpI2(GPI2),
.gpO1(GPO1),.gpO2(GPO2));

    //pipeline
    Reg #(1) IDEX_WED  (.D (tempwe_dm) , .Q(IDEX_wed), .clk(clk),
.LD(1'b1));
    Reg #(1) EXMEM_WED  (.D (IDEX_wed) , .Q(we_dm), .clk(clk),
.LD(1'b1));

endmodule
```

Fpga_top.v (new)

```
`timescale 1ns / 1ps
module fpga_top(
        input  wire       clk, button,
        input  wire       rst,
        input  wire [4:0] switches,
```

```
    output wire [4:0] LED,
    output wire [3:0] LEDSEL,
    output wire [7:0] LEDOUT
);

wire [15:0] reg_hex;
wire        clk_sec;
wire        clk_5KHz;
wire        clk_pb;

wire [7:0]  digit0;
wire [7:0]  digit1;
wire [7:0]  digit2;
wire [7:0]  digit3;

wire [31:0] pc_current;
wire [31:0] instr;
wire [31:0] alu_out;
wire [31:0] wd_dm;
wire [31:0] rd_dm;
wire [31:0] dispData;
wire        we_dm;

wire [31:0] GPO1, GPO2;
wire       dispSE, factErr;

clk_gen clk_gen (
        .clk100MHz          (clk),
        .rst                (rst),
        .clk_4sec           (clk_sec),
        .clk_5KHz           (clk_5KHz)
    );
button_debouncer bd (
        .clk                (clk_5KHz),
        .button             (button),
        .debounced_button   (clk_pb)
    );
mips_top mips_top (
        .clk                (clk_5KHz),
```

```
            .rst                (rst),
            .ra3                (4'b0),
            .we_dm              (we_dm),
            .pc_current         (pc_current),
            .instr              (instr),
            .alu_out            (alu_out),
            .writeData            (wd_dm),
            .readData              (rd_dm),
            .rd3                (dispData),
            .GPI1               ({27'b0, switches[4:0]}),
            .GPI2               (GPO1),
            .GPO1               (GPO1),
            .GPO2               (GPO2)
        );

    assign dispSE  = GPO1[1];
    assign factErr = GPO1[0];
    /*
    switches[4:0] are used as the 3rd read address (ra3) of the RF,
    dispData is the register contents from the RF's 3rd read port
(rd3).
    */
    assign reg_hex = dispSE ? GPO2[31:16] : GPO2[15:0];
    hex_to_7seg hex3 (
            .HEX                (reg_hex[15:12]),
            .s                  (digit3)
        );

    hex_to_7seg hex2 (
            .HEX                (reg_hex[11:8]),
            .s                  (digit2)
        );

    hex_to_7seg hex1 (
            .HEX                (reg_hex[7:4]),
            .s                  (digit1)
        );

    hex_to_7seg hex0 (
```

```
            .HEX                (reg_hex[3:0]),
            .s                  (digit0)
        );

    led_mux led_mux (
            .clk                (clk_5KHz),
            .rst                (rst),
            .LED3               (digit3),
            .LED2               (digit2),
            .LED1               (digit1),
            .LED0               (digit0),
            .LEDSEL             (LEDSEL),
            .LEDOUT             (LEDOUT)
        );

        assign LED = { dispSE, {4{factErr}}};

endmodule
```

fpga_top.xdc(new)

```
# Clock Signal
set_property -dict {PACKAGE_PIN W5 IOSTANDARD LVCMOS33}
[get_ports {clk}];
create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5}
[get_ports {clk}];

# Buttons
set_property -dict {PACKAGE_PIN U18 IOSTANDARD LVCMOS33} [get_ports
{button}]; # Center Button
set_property -dict {PACKAGE_PIN W19 IOSTANDARD LVCMOS33} [get_ports
{rst}]; # Left Button

# Switches
set_property -dict {PACKAGE_PIN V17 IOSTANDARD LVCMOS33} [get_ports
{switches[0]}];  # Switch 0
set_property -dict {PACKAGE_PIN V16 IOSTANDARD LVCMOS33} [get_ports
```

```
{switches[1]}];  # Switch 1
set_property -dict {PACKAGE_PIN W16 IOSTANDARD LVCMOS33} [get_ports
{switches[2]}];  # Switch 2
set_property -dict {PACKAGE_PIN W17 IOSTANDARD LVCMOS33} [get_ports
{switches[3]}];  # Switch 3
set_property -dict {PACKAGE_PIN W15 IOSTANDARD LVCMOS33} [get_ports
{switches[4]}];  # Switch 4

# LEDs
set_property PACKAGE_PIN U16 [get_ports {LED[0]}]

     set_property IOSTANDARD LVCMOS33 [get_ports {LED[0]}]
set_property PACKAGE_PIN E19 [get_ports {LED[1]}]

     set_property IOSTANDARD LVCMOS33 [get_ports {LED[1]}]
set_property PACKAGE_PIN U19 [get_ports {LED[2]}]

     set_property IOSTANDARD LVCMOS33 [get_ports {LED[2]}]
set_property PACKAGE_PIN V19 [get_ports {LED[3]}]

     set_property IOSTANDARD LVCMOS33 [get_ports {LED[3]}]
set_property PACKAGE_PIN W18 [get_ports {LED[4]}]

     set_property IOSTANDARD LVCMOS33 [get_ports {LED[4]}]


# 7 segment display
set_property -dict {PACKAGE_PIN W7 IOSTANDARD LVCMOS33} [get_ports
{LEDOUT[0]}]; # CA
set_property -dict {PACKAGE_PIN W6 IOSTANDARD LVCMOS33} [get_ports
{LEDOUT[1]}]; # CB
set_property -dict {PACKAGE_PIN U8 IOSTANDARD LVCMOS33} [get_ports
{LEDOUT[2]}]; # CC
set_property -dict {PACKAGE_PIN V8 IOSTANDARD LVCMOS33} [get_ports
{LEDOUT[3]}]; # CD
set_property -dict {PACKAGE_PIN U5 IOSTANDARD LVCMOS33} [get_ports
{LEDOUT[4]}]; # CE
set_property -dict {PACKAGE_PIN V5 IOSTANDARD LVCMOS33} [get_ports
{LEDOUT[5]}]; # CF
```

```
set_property -dict {PACKAGE_PIN U7 IOSTANDARD LVCMOS33} [get_ports
{LEDOUT[6]}]; # CG
set_property -dict {PACKAGE_PIN V7 IOSTANDARD LVCMOS33} [get_ports
{LEDOUT[7]}]; # DP

set_property -dict {PACKAGE_PIN U2 IOSTANDARD LVCMOS33} [get_ports
{LEDSEL[0]}]; # AN0
set_property -dict {PACKAGE_PIN U4 IOSTANDARD LVCMOS33} [get_ports
{LEDSEL[1]}]; # AN1
set_property -dict {PACKAGE_PIN V4 IOSTANDARD LVCMOS33} [get_ports
{LEDSEL[2]}]; # AN2
set_property -dict {PACKAGE_PIN W4 IOSTANDARD LVCMOS33} [get_ports
{LEDSEL[3]}]; # AN3
```

Tb_mips_top.v (new)

```
module tb_mips_top;

    reg         clk;
    reg         rst;
    wire        we_dm;
    wire [31:0] pc_current;
    wire [31:0] instr;
    wire [31:0] alu_out;
    wire [31:0] wd_dm;
    wire [31:0] rd_dm, GPO1, GPO2;
    wire [31:0] DONT_USE;
    reg  [31:0] GPI1, GPI2;
    integer i;

    mips_top DUT (
            .clk            (clk),
            .rst            (rst),
            .we_dm          (we_dm),
            .ra3            (5'h0),
            .pc_current     (pc_current),
            .instr          (instr),
```

```
            .alu_out        (alu_out),
            .writeData        (wd_dm),
            .readData         (rd_dm),
            .GPI1           (GPI1),
            .GPI2           (GPI2),
            .GPO1           (GPO1),
            .GPO2           (GPO2),
            .rd3            (DONT_USE)
        );

    task tick;
    begin
        clk = 1'b0; #5;
        clk = 1'b1; #5;
    end
    endtask

    task reset;
    begin
        rst = 1'b0; #5;
        rst = 1'b1; #5;
        rst = 1'b0;
    end
    endtask

    initial begin
       reset;
//        for(i = 0; i < 13; i = i + 1)
//        begin
//            tick;
//        end
//        reset;
        GPI1 = 32'b100;
        GPI2 = GPO1;
        while(pc_current != 32'h64) tick;
        tick; tick; tick; tick; tick;
        $finish;
    end
```

```
endmodule
```

fact_top_tb.v(new)

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 11/15/2020 09:26:17 PM
// Design Name:
// Module Name: fact_top_tb
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module fact_top_tb;

    parameter PERIOD = 10;
    parameter real DUTY_CYCLE = 0.5;
    parameter OFFSET = 2;
    reg clkin;
    initial    // Clock process for clkin
    begin
        #OFFSET
```

```
          clkin = 1'b1;
        forever
        begin
            #(PERIOD-(PERIOD*DUTY_CYCLE)) clkin = ~clkin;
        end
    end

    parameter   //{A, WE}
        LOADIN = 3'b00_1,
        READIN = 3'b00_0,
        LOADGO = 3'b01_1,
        READGO = 3'b01_0,
        COUT   = 3'b10_0,
        OUTPUT = 3'b11_0;



    reg [1:0]  A;
    reg        WE, rst;
    reg [3:0]  WD;
    wire [31:0] RD;
    integer i, errorc, expected;
    fact_top UUT(.A(A), .WE(WE), .rst(rst), .WD(WD), .RD(RD),
.clk(clkin));

    initial begin
    rst = 1'b1;
    #10;
    rst = 1'b0;
    #10;
    expected = 1;
    errorc = 0;
        for(i = 1; i < 2**4; i = i +1)begin

        WD = i;
        expected = expected * i;
        {A,WE} = READIN;
        #10;
        if(i-1 != RD) begin
```

```
            errorc = errorc + 1;
            $display("READIN ERROR");
            end;
        {A,WE} = LOADIN;
        #10;
        WD = 1;
        {A,WE} = READGO;
        #10;
        {A,WE} = LOADGO;
        #50;
        {A,WE} = COUT;
        #10;
        while(RD[0] != 1'b1 && RD[1] != 1'b1) #10;
        {A,WE} = OUTPUT;
        #10;
        if(expected != RD && i < 13)begin
             errorc = errorc + 1;
             $display("%b",i);
            end
        #10;
        WD = 0;
        {A,WE} = LOADGO;
        #10;
            end
    end
endmodule
```

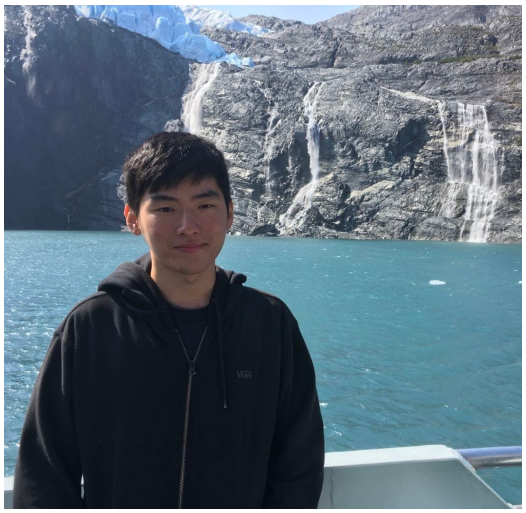
Group Members

Patrick Lu

This is my fourth year in San Jose State University. I love coding and it is pretty much my hobby, so classes starting again made my quarantine days a little less boring.

Alejandro Chavez-Guerrero

I am currently in my last semester at San Jose State University. In my free time I enjoy watching basketball and being outdoors.



Chujie Meng

I am a senior student in San Jose State University .I like reading books and watching cartoons in my free time.

Wenyi Cai

One more semester to finish Bachelor of Science Degree at San Jose State University. Currently focusing on embedded system design studying and may apply for further studying programs. Love Animes.