



VE370 Intro to Computer Organization

Project 2 Team Report

FA 2020

Name	Stud. ID
Zhou Zhanpeng	518021910594
Liu Yihua	518021910998
Shen Yang	518370910027
Peng Haotian	518370910107

Abstract

In this report, we demonstrate our implementation of a 5-stage pipelined processor based on MIPS instruction set and Verilog HDL modeling. Our project implements a pipelined processor in MIPS architecture with a forwarding unit and two hazard detecting unit to resolve hazard issues. We use Verilog HDL to model the processor and verify our design by Vivado simulation included in this report. Besides, we also demonstrate our design on a FPGA board.

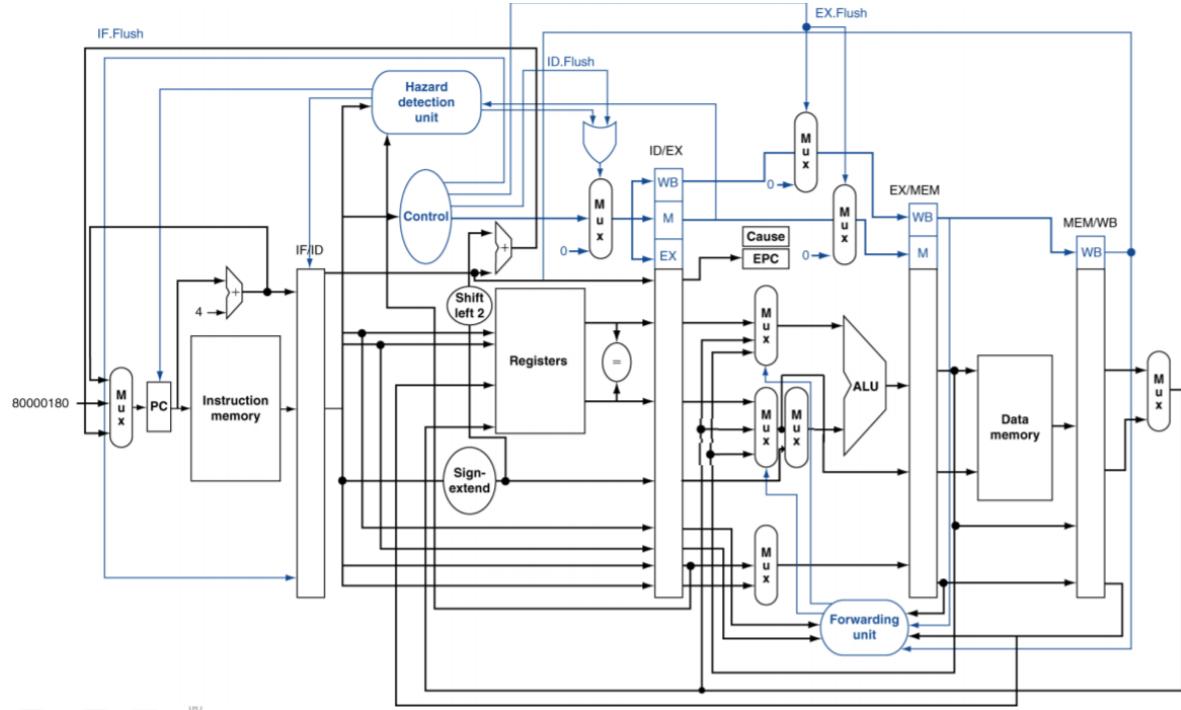
Introduction

In this project, we implement a 5-stage pipelined processor based on MIPS instruction set with a forwarding unit and two hazard detecting unit to resolve hazard issues. MIPS is an open source instruction set architecture with high performance that is widely used in embedded systems. Pipelined processor is implemented by a technique called instruction pipelining that is implemented by dividing instructions into a series of pipelines so that the processor's cycle time is reduced. It is simple, reliable, and fast.

To realize instruction pipelining, we divide the handling of instructions into 5 stages: IF (instruction fetch), ID (instruction decode), EX (execute), MEM (memory access), and (register write back). Between them, we have 4 stage registers: IF/ID register, ID/EX register, EX/MEM register, MEM/WB register. In the first stage, we acquire PC address and instruction from instruction memory; in the second stage, we acquire control signals, do register reading or writing if applicable, and detect hazards; in the third stage, we do ALU (Arithmetic Logic Unit) calculation and forwarding if applicable; in the fourth stage, we read or write data from the data memory; in the fifth stage, we select a write-back signal.

Description

Overview



Basically, our modeling and implementation is following above picture. However, some part is added and some part is deleted due to our specialized design. For example, to support **j** type instruction, we add an another mux before PC to select the jump target address. Also, EX.flush signal has no usage in our design then we simply remove it.

Next, we will go deeper about how we model the stage register, memory and how to resolve data hazard and control hazard. We will not talk about how to model a mux or comparator in later sections because they are simple to implement.

Stage Register

In pipelined processor, we use four stage registers to process signals: IF/ID, ID/EX, EX/MEM, MEM/WB. The names of wires and registers we use in state registers with their corresponding descriptions are list below.

Specially, the stage register is triggered by rising edge of clock. And for IF/ID & ID/EX, there are two control signal to flush all the content or just control signals passed in to zero.

	Name	Descriptions
input	clk	clock signal
	IFID_write	whether to write IF/ID stage register
	IF_flush	whether to flush IF stage
	IF_instr	instruction from the instruction memory in IF stage
	IF_pcplus4	the value of PC + 4 in IF stage
output	IFID_instr	instruction output from IF/ID stage register
	IFID_pcplus4	the value of PC + 4 output from IF/ID stage register

Table 1. IF/ID stage register.

	Name	Descriptions
input	clk	clock signal
	IF_flush	whether to flush IF stage
	regReadData1ID	registers read data 1 in ID stage
	regReadData2ID	registers read data 2 in ID stage
	signExtendID	output of sign-extend in ID stage
	registerRsID	the value of register rs in ID stage
	registerRtID	the value of register rt in ID stage
	registerRdID	the value of register rd in ID stage
	aluOpID	control signal ALUOp in ID stage
	regDstID	control signal RegDst in ID stage
	memReadID	control signal MemRead in ID stage
	memtoRegID	control signal MemtoReg in ID stage
	memWriteID	control signal MemWrite in ID stage
	aluSrcID	control signal ALUSrc in ID stage
	regWriteID	control signal RegWrite in ID stage
output	regReadData1EX	registers read data 1 in EX stage
	regReadData2EX	registers read data 2 in EX stage
	signExtendEX	output of sign-extend in EX stage
	registerRsEX	the value of register rs in EX stage
	registerRtEX	the value of register rt in EX stage
	registerRdEX	the value of register rd in EX stage
	aluOpEX	control signal ALUOp in EX stage
	regDstEX	control signal RegDst in EX stage
	memReadEX	control signal MemRead in EX stage
	memtoRegEX	control signal MemtoReg in EX stage
	memWriteEX	control signal MemWrite in EX stage
	aluSrcEX	control signal ALUSrc in EX stage
	regWriteEX	control signal RegWrite in EX stage

Table 2. ID/EX stage register.

	Name	Descriptions
input	Clock	clock signal
	EX_MemRead	control signal MemRead in EX stage
	EX_MemtoReg	control signal MemtoReg in EX stage
	EX_MemWrite	control signal MemWrite in EX stage
	EX_RegWrite	control signal RegWrite in EX stage
	EX_MUX8_out	the destination register in EX stage
	EX_ALU_result	ALU result as an output of ALU in EX stage
	EX_MUX6_out	the value of R[rt] in EX stage
	MEM_MemRead	control signal MemRead in MEM stage
	MEM_MemtoReg	control signal MemtoReg in MEM stage
output	MEM_MemWrite	control signal MemWrite in MEM stage
	MEM_RegWrite	control signal RegWrite in MEM stage
	MEM_MUX8_out	the destination register in MEM stage
	MEM_ALU_result	ALU result as an output of ALU in MEM stage
	MEM_MUX6_out	the value of R[rt] in MEM stage

Table 3. EX/MEM stage register.

	Name	Descriptions
input	Clock	clock signal
	MEM_RegWrite	control signal RegWrite in MEM stage
	MEM_MemtoReg	control signal MemtoReg in MEM stage
	MEM_MUX8_out	the destination register in MEM stage
	MEM_Data_memory_Read_data	data memory read data in MEM stage
	MEM_ALU_result	ALU result as an output of ALU in MEM stage
	WB_MemtoReg	control signal MemtoReg in WB stage
	WB_RegWrite	control signal RegWrite in WB stage
	WB_MUX8_out	the destination register in WB stage
	WB_Data_memory_Read_data	data memory read data in WB stage
output	WB_ALU_result	ALU result as an output of ALU in WB stage

Table 4. MEM/WB stage register.

Memory

The implementation of the memory part includes **Data Memory** (module *data_mem* in the source code), **Instruction Memory** (module *Instr_mem* in the source code) and **Register File** (module *reg_file* in the source file). There are some typical designing details in this project:

1. The data is byte-addressable. Though this project is based on word addresses, the real memory units supported by MIPS should be byte-addressable. There are some typical MIPS instructions that needs the addresses of byte, such as lb, lbu and lh. Though those instruction are not needed in this project, we still consider this condition in case we could develop more instructions through this project in future.
2. The data memory and register file are driven by the negative edge of clock in this project. This design may lengthen the critical path, but it could avoid some hazards:

```
always @(negedge clk) begin
    if (mem_write) begin
        memory[read_addr] = write_data[word - 1:word - byte];
        memory[read_addr + 1] = write_data[word - byte - 1:word - 2*byte];
        memory[read_addr + 2] = write_data[word - 2*byte - 1:word - 3*byte];
        memory[read_addr + 3] = write_data[word - 3*byte - 1:0];
    end
end
```

Take a part of the implementation of data memory as an example. If we update data in the positive edge, which is the beginning of a new clock cycle, the value of some variables may stay as the former clock cycle while some already change. If the write data has changed and the address of the write data stays the same, then wrong data will be updated into the write address of the last instruction. With the time of the first half of the clock cycle, this hidden hazard could be solved.

The detailed descriptions of the three memory units are shown below:

Data Memory

Data memory is used to store most data. In this project, due to the restriction of the software, we limit the capacity as 1000 words (4KB). The module of data memory takes 6 variables, including 5 inputs (*read_addr*, *write_data*, *mem_write*, *mem_read*, *clk*) and 1 output (*read_data*). The module will read or write data through a 32×1000 register called *memory*. If *mem_write* = 1, the output variable *read_data* will be updated with the data stores in *read_addr* in the *memory*.

Instruction Memory

Instruction memory is used to stores the MIPS instructions in a program in words. The PC could get the data of an instruction by searching the address of that data. In this project, we limit the capacity of the instruction memory as 42 words, which means it can hold at most 42 instructions. The module of instruction memory takes 2 variables, including 1 input (*read_addr*) and 1 output (*instruction*). To simulate a program, the data in the memory is initialized as follows:

```

00000 00001000 00000000 00100000 //addi $t0, $zero, 0x20
00000 00001001 00000000 00110111 //addi $t1, $zero, 0x37
00001 00001001 10000000 00100100 //and $s0, $t0, $t1
00001 00001001 10000000 00100101 //or $s0, $t0, $t1
01100 00010000 00000000 00000100 //sw $s0, 4($zero)
01100 00001000 00000000 00000100 //sw $t0, 8($zero)
00001 00001001 10001000 00100000 //add $s1, $t0, $t1
00001 00001001 10010000 00100010 //sub $s2, $t0, $t1
10010 00110010 00000000 00001001 //beq $s1, $s2, error0
01100 00010001 00000000 00000100 //lw $s1, 4($zero)
10010 00110010 00000000 01001000 //andi $s2, $s1, 0x48
10010 00110010 00000000 00001001 //beq $s1, $s2, error1
01100 00010011 00000000 00001000 //lw $s3, 8($zero)
10010 00010011 00000000 00001010 //beq $s0, $s3, error2
00010 01010001 10100000 00101010 //slt $s4, $s2, $s1 (Last)
10010 10000000 00000000 00001111 //beq $s4, $0, EXIT
00010 00100000 10010000 00100000 //add $s2, $s1, $0
01000 00000000 00000000 00001110 //j Last
00000 00001000 00000000 00000000 //addi $t0, $0, 0(error0)
00000 000001001 00000000 00000000 //addi $t1, $0, 0
01000 00000000 00000000 00011111 //j EXIT
00000 00001000 00000000 00000001 //addi $t0, $0, 1(error1)
00000 000001001 00000000 00000001 //addi $t1, $0, 1
01000 00000000 00000000 00011111 //j EXIT
00000 00001000 00000000 00000010 //addi $t0, $0, 2(error2)
00000 000001001 00000000 00000010 //addi $t1, $0, 2
01000 00000000 00000000 00011111 //j EXIT
00000 00001000 00000000 00000011 //addi $t0, $0, 3(error3)
00000 000001001 00000000 00000011 //addi $t1, $0, 3
01000 00000000 00000000 00011111 //j EXIT

```

The module will read the *read_addr* and find the correspond instruction to update in *instruction*.

Register File

Register file is used to store the data of the 32 most commonly used registers. The order of these registers is shown below:

NAME	NMBR	USE	STORE?
\$zero	0	The Constant Value 0	N.A.
\$at	1	Assembler Temporary	No
\$v0-\$v1	2-3	Values for Function Results and Expression Evaluation	No
\$a0-\$a3	4-7	Arguments	No
\$t0-\$t7	8-15	Temporaries	No
\$s0-\$s7	16-23	Saved Temporaries	Yes
\$t8-\$t9	24-25	Temporaries	No
\$k0-\$k1	26-27	Reserved for OS Kernel	No
\$gp	28	Global Pointer	Yes
\$sp	29	Stack Pointer	Yes
\$fp	30	Frame Pointer	Yes
\$ra	31	Return Address	Yes
\$f0-\$f31	0-31	Floating Point Registers	Yes

In this project, the module of data memory takes 8 variables, including 6 inputs (*read_addr1*, *read_addr2*, *write_addr*, *write_data*, *regwrite*, *clk*) and 2 outputs (*read_data1*, *read_data2*). To simplify the process, the data of every register is initialized as 32 bits 0. The *write_data* will be updated to correspond register at each negative edge. The *read_data* will be assigned to correspond register whenever the *rrad_addr* changes (read is asynchronous while write is synchronous).

Control Unit

The control unit is used to generate control signals from different 6 bits opcodes. From the structure of pipeline in VE370 lectures, the control unit should generate 9 outputs as follows:

```

initial begin
    ALUop = 2'b00; RegDst = 0; Jump = 0; Branch = 0; MemRead = 0; MemToReg = 0; MemWrite = 0; ALUSrc = 0; RegWrite = 0;
end

```

In this project, to support both `beq` and `bne` instructions, we introduce another output called `beq`. `Beq` will only be 1 when the instruction is `beq`. If both `branch` and `beq` signals equal to 1 we could determine the instruction as `beq`. If `branch` signal equals to 1 while `beq` signal equal to 0, then the instruction is `bne`.

ALU Related Modules

This part describes the ALU related modules in the pipelined processor, including the `ALUop` signal, the ALU control unit and the ALU unit. The processor first produces a 2-bit `ALUop` signal by the control unit. Then the `ALUop` signal is processed by the ALU control unit which will output a 4-bit ALU control signal. Finally the ALU unit executes the computation according to the ALU control signal.

ALUOp

`ALUop` signal is a 2-bit signal, produced by the control unit according to different instruction. `ALUop` signal is one of the inputs of the ALU control unit, which determines the type of computation of the ALU unit. How `ALUop` is designed is listed in the table below, together with the output result of the ALU control unit. R-type instructions share the same `ALUop` signal 10, to further identify different R-type instructions funct code need to be checked later. `lw`, `sw` and `addi` share `ALUop` of 00. `beq` and `bne` share `ALUop` of 01. `andi` produces `ALUop` of 11.

ALU Control Unit

ALU control unit takes two inputs, `ALUop` and `funct`, and outputs an 4-bit ALU control signal. The design is in the table below. Notice that actually `j` type instruction doesn't need to perform ALU operation, thus, we just randomly picked one `ALUop` and corresponding ALU control for it.

Op code	Operation	ALUop	Funct	ALU control	ALU function
<code>lw</code> (100011)	load word	00	xxxxxx	0010	add
<code>sw</code> (101011)	save word	00	xxxxxx	0010	add
<code>addi</code> (001000)	add immediate	00	xxxxxx	0010	add
<code>beq</code> (000100)	branch if equal	01	xxxxxx	0110	subtract
<code>bne</code> (000101)	branch if not equal	01	xxxxxx	0110	subtract
<code>andi</code> (001100)	and immediate	11	xxxxxx	0000	and
R-type (000000)	add	10	100000	0010	add
	sub	10	100010	0110	subtract
	and	10	100100	0000	and
	or	10	100101	0001	or
	slt	10	101010	0111	slt
<code>j</code>	jump	00	xxxxxx	0010	add

In this project, 12 instructions are to be implemented, including `lw`, `sw`, `beq`, `bne`, `add`, `sub`, `AND`, `OR`, `slt`, `addi`, `andi`, `j`. The funct code of all the I-type instructions doesn't matter, marked as `xxxxxx` in the table. For the R-type instructions, different ALU control signals are determined by the funct code.

`lw`, `sw`, `addi`, `add` require the ALU unit to do add function, so they share an ALU control code of `0010`. Likewise, `beq`, `bne`, `sub` require subtract function, output `0110`. `andi`, `AND` require `AND` function, output `0000`. `OR` outputs `0001`, indicating `OR` function. `slt` outputs `0111`, indicating `slt` function.

ALU Unit

The ALU unit takes the ALU control signal from the previous ALU control unit, to determine the type of computation.

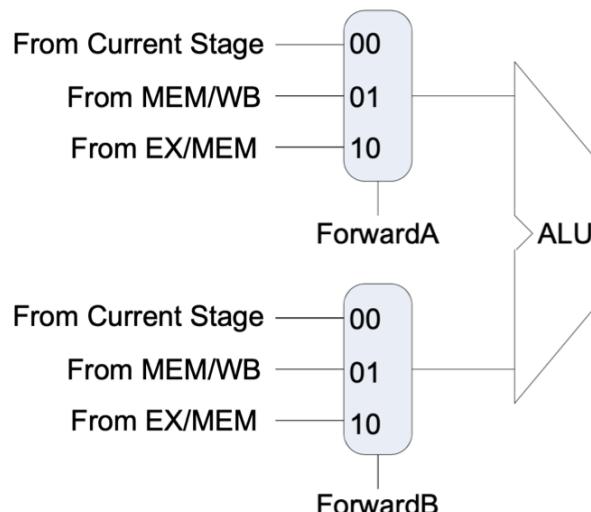
ALU control	ALU function
0000	and
0001	or
0010	add
0110	subtract
0111	slt

Data Hazard

This part describes how data hazards in this pipelined processor are solved. The data hazards that will occur can be categorized into three kinds, R-type data hazard, data hazard on branch, and data hazard on `lw`.

Forwarding Unit

Data hazard caused by R-type instructions is the simplest and can be solved by a forwarding unit, see the figure below. The unit controls the inputs of the ALU unit to make sure the value of the register being used is updated, if the previous one or two instruction changes the value of `Rd` (`Rd` is not `$0`), and the previous `Rd` is the `Rs` or `Rt` of the current instruction.



The logic of the forwarding unit is as follows:

Initially forwardA = 0, forwardB = 0.

- For 1&2 hazard:
 - If (regWrite.MEM && regRdMEM && (regRdMEM == regRsEX))
 - forwardA = 2'b10
 - If (regWrite.MEM && regRdMEM && (regRdMEM == regRtEX))
 - forwardB = 2'b10
- For 1&3 hazard (notice that we need to make sure there is no 1&2 hazard):
 - If (regWrite.WB && regRdWB && (regRdWB == regRsEX) && forwardA != 2'b10)
 - forwardA = 2'b01
 - If (regWrite.WB && regRdWB && (regRdWB == regRtEX) && forwardB != 2'b10)
 - forwardB = 2'b01;

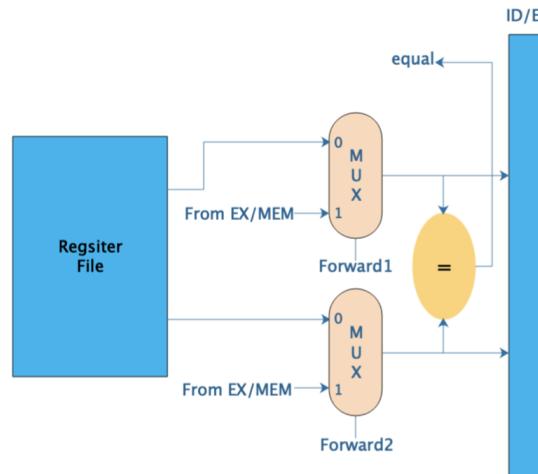
This unit is named as `R_forward` in Appendix.

Data Hazard on Branch

Since the branch instruction result will be produced by the comparator early in the ID stage, the forwarding unit is not enough to solve the data hazard caused by beq and bne. Here 5 more signals are introduced, `forward1`, `forward2`, `PCwrite`, `IFID_write`, and `ID_flush`.

Initially `forward1 = 0`, `forward2 = 0`, `PCwrite = 1`, `IFID_write = 1`, and `ID_flush = 0`.

Amongst, `forward1` and `forward2` are two select signals of muxes, as shown below. If the instruction in the MEM stage writes back to `Rd` (`Rd` is not \$0), the instruction in ID stage is beq/bne, and `Rd` of the MEM stage is `Rs/Rt` in the ID stage, then `forward1/forward2 = 1`.



Specifically:

```
if (EXMEM_regWrite && !EXMEM_memRead && ID_branch) begin
    if (EXMEM_regRd == ID_regRs) forward1 = 1;
    if (EXMEM_regRd == ID_regRt) forward2 = 1;
end
```

`PCwrite`, `IFID_write`, and `ID_flush` are used to insert stalls.

If the EX stage instruction before beq/bne (currently in the ID stage) writes back to `Rd` (`Rd != $0`), and `Rd` of the EX stage is `Rs/Rt` in the ID stage, then insert one stall, making `PCwrite = 0`, `IFID_write = 0` and `ID_flush = 1`.

Else if the MEM stage instruction reads data from the data memory, meaning it will write back to `Rd` (`Rd != $0`) in next clock cycle; and the ID stage instruction is `beq/bne`, and `Rd` of the MEM stage is `Rs/Rt` in the ID stage, insert one stall as well.

Specifically:

```

if (IDEX_regWrite && ID_branch) begin
    if (EX_regDst == ID_regRs || EX_regDst == ID_regRt) begin
        PCWrite = 0; IFID_write = 0; ID_flush = 1;
    end
end

if (EXMEM_memRead && ID_branch) begin
    if (EXMEM_regRd == ID_regRs || EXMEM_regRd == ID_regRt) begin
        PCWrite = 0; IFID_write = 0; ID_flush = 1;
    end
end

```

This unit is named as `brc_hazard` in Appendix.

`lw` Data hazard

The third kind of data hazard is caused by the `lw` instruction. Different from R-type instructions, the earliest time the `lw` instruction produces its result is the MEM stage. While forwarding back in time is impossible, a stall after the `lw` instruction is necessary. Again we use three signals to control stalls, `PCwrite`, `IFID_write`, and `ID_flush`.

Initially `PCwrite = 1`, `IFID_write = 1`, and `ID_flush = 0`.

If the EX stage instruction is `lw`, and `Rd` (`Rd != $0`) of `lw` is `Rs/Rt` of the ID stage instruction, insert a stall.

Specifically:

```

if (IDEX_memRead && (IFID_regRs == INDEX_regRt || IFID_regRt == INDEX_regRt)) begin
    PCwrite      = 0;
    ID_flush     = 1;
    IFID_write   = 0;
end

```

Control Hazard

Simply, control hazard is resolved by flush the whole instruction and other content in `ID/EX` stage register. Firstly, based on `branch` and `beq` signals generated by control unit, we know if there is a `beq` instruction in ID stage right now. Then, we will compare the two register's content, if the comparator gives 1 when `beq = 1` or gives 0 when `beq = 0`, we will simply flush the content in `ID/EX` stage register right now, this mechanism is called **Assume not taken**.

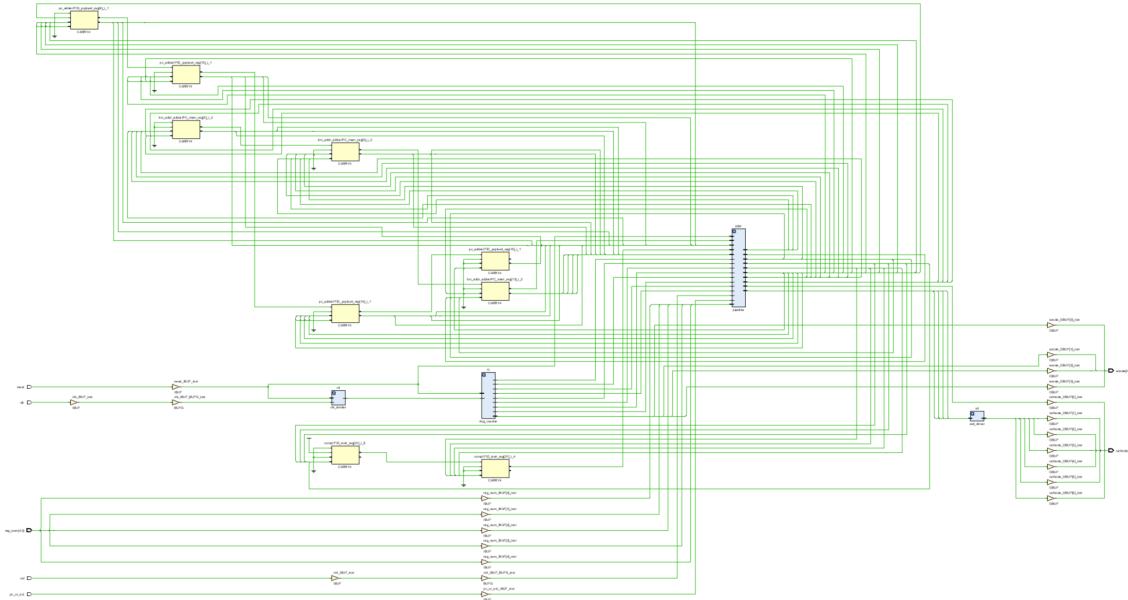
However, if control unit gives `jump = 1`, which means there is a j-type instruction in `ID` stage, we will directly flush all the content in `ID/EX` stage register.

Implementation & Simulation Results

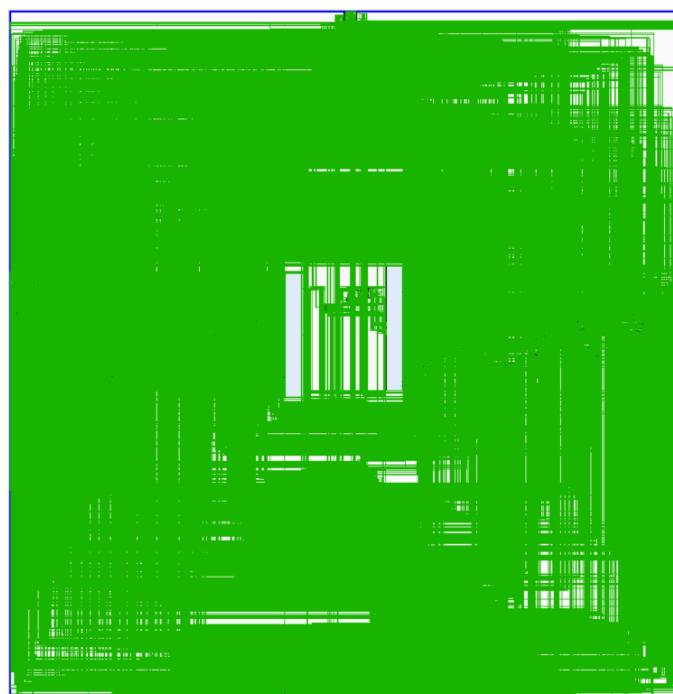
RTL Schematic

Based on our design described above, we implement our pipeline processor in `vivado` which gives the schematic of our design shown as below.

This is the whole picture of our design including pipeline processor and ssd_driver and other thing.



This one is the internal design of the pipeline processor.



Theoretical Results

In this report, we use `InstructionMem_for_P2_Demo_bonus.txt` to simulate our pipeline processor. Following is the theoretical results:

cycle	Instruction	IF	ID	EX	MEM	WB	Results
1	1. addi \$t0, \$zero, 0x20	1	x	x	x	x	
2	2. addi \$t1, \$zero, 0x37	2	1	x	x	x	
3	3. and \$s0, \$t0, \$t1	3	2	1	x	x	
4	4. or \$s0, \$t0, \$t1	4	3	2	1	x	
5	5. sw \$s0, 4(\$zero)	5	4	3	2	1	\$t0 = 0x20
6	6. sw \$t0, 8(\$zero)	6	5	4	3	2	\$t1 = 0x37
7	7. add \$s1, \$t0, \$t1	7	6	5	4	3	\$s0 = 0x20
8	8. sub \$s2, \$t0, \$t1	8	7	6	5	4	\$s0 = 0x37; Data.Mem[4] = 0x37
9	9. beq \$s1, \$s2, error0	9	8	7	6	5	Data.Mem[8] = 0x20
10	10. lw \$s1, 4(\$zero)	10	9	8	7	6	
11		10	9	nop	8	7	\$s1 = 0x57
12	11. andi \$s2, \$s1, 0x48	11	10	9	nop	8	\$s2 = 0xffffffffe9
13	12. beq \$s1, \$s2, error1	12	11	10	9	nop	
14		12	11	nop	10	9	
15	13. lw \$s3, 8(\$zero)	13	12	11	nop	10	\$s1 = 0x37
16		13	12	nop	11	nop	
17	14. beq \$s0, \$s3, error2	14	13	12	nop	11	\$s2 = 0x0
18	15. slt \$s4, \$s2, \$s1 (Last)	15	14	13	12	nop	
19		15	14	nop	13	12	
20		15	14	nop	nop	13	\$s3 = 0x20

cycle	Instruction	IF	ID	EX	MEM	WB	Results
21	16. beq \$s4, \$0, EXIT	16	15	14	nop	nop	
22	17. add \$s2, \$s1, \$0	17	16	15	14	nop	
23		17	16	nop	15	14	
24	18. j Last	18	17	16	nop	15	\$s4 = 0x1
25	19. addi \$t0, \$0, 0(error0)	19	18	17	16	nop	
26	15. slt \$s4, \$s2, \$s1 (Last)	15	flush	18	17	16	
27	16. beq \$s4, \$0, EXIT	16	15	flush	18	17	\$s2 = 0x37
28	17. add \$s2, \$s1, \$0	17	16	15	flush	18	
29		17	16	nop	15	flush	
30	EXIT	EXIT	flush	16	nop	15	
31	EXIT	EXIT	EXIT	flush	16	nop	
32	EXIT	EXIT	EXIT	EXIT	flush	16	
33	EXIT	EXIT	EXIT	EXIT	EXIT	flush	

Textual Simulation Results

Also, we generated the following textual result which shows the content of register at each clock cycle, which concides with the theorectial result.

```
=====
Time: 0, CLK = 0, PC = 00000000
[$s0] = 00000000, [$s1] = 00000000, [$s2] = 00000000
[$s3] = 00000000, [$s4] = 00000000, [$s5] = 00000000
[$s6] = 00000000, [$s7] = 00000000, [$t0] = 00000000
[$t1] = 00000000, [$t2] = 00000000, [$t3] = 00000000
[$t4] = 00000000, [$t5] = 00000000, [$t6] = 00000000
[$t7] = 00000000, [$t8] = 00000000, [$t9] = 00000000
Time: 50, CLK = 1, PC = 00000000
[$s0] = 00000000, [$s1] = 00000000, [$s2] = 00000000
[$s3] = 00000000, [$s4] = 00000000, [$s5] = 00000000
[$s6] = 00000000, [$s7] = 00000000, [$t0] = 00000000
[$t1] = 00000000, [$t2] = 00000000, [$t3] = 00000000
[$t4] = 00000000, [$t5] = 00000000, [$t6] = 00000000
[$t7] = 00000000, [$t8] = 00000000, [$t9] = 00000000
=====
```

cycle	Instruction	IF	ID	EX	MEM	WB	Results
1	1. addi \$t0, \$zero, 0x20	1	x	x	x	x	

```

=====
Time:          100, CLK = 0, PC = 00000004
[$s0] = 00000000, [$s1] = 00000000, [$s2] = 00000000
[$s3] = 00000000, [$s4] = 00000000, [$s5] = 00000000
[$s6] = 00000000, [$s7] = 00000000, [$t0] = 00000000
[$t1] = 00000000, [$t2] = 00000000, [$t3] = 00000000
[$t4] = 00000000, [$t5] = 00000000, [$t6] = 00000000
[$t7] = 00000000, [$t8] = 00000000, [$t9] = 00000000
Time:          150, CLK = 1, PC = 00000004
[$s0] = 00000000, [$s1] = 00000000, [$s2] = 00000000
[$s3] = 00000000, [$s4] = 00000000, [$s5] = 00000000
[$s6] = 00000000, [$s7] = 00000000, [$t0] = 00000000
[$t1] = 00000000, [$t2] = 00000000, [$t3] = 00000000
[$t4] = 00000000, [$t5] = 00000000, [$t6] = 00000000
[$t7] = 00000000, [$t8] = 00000000, [$t9] = 00000000
=====
```

cycle	Instruction	IF	ID	EX	MEM	WB	Results
2	2. addi \$t1, \$zero, 0x37	2	1	x	x	x	

```

=====
Time:          200, CLK = 0, PC = 00000008
[$s0] = 00000000, [$s1] = 00000000, [$s2] = 00000000
[$s3] = 00000000, [$s4] = 00000000, [$s5] = 00000000
[$s6] = 00000000, [$s7] = 00000000, [$t0] = 00000000
[$t1] = 00000000, [$t2] = 00000000, [$t3] = 00000000
[$t4] = 00000000, [$t5] = 00000000, [$t6] = 00000000
[$t7] = 00000000, [$t8] = 00000000, [$t9] = 00000000
Time:          250, CLK = 1, PC = 00000008
[$s0] = 00000000, [$s1] = 00000000, [$s2] = 00000000
[$s3] = 00000000, [$s4] = 00000000, [$s5] = 00000000
[$s6] = 00000000, [$s7] = 00000000, [$t0] = 00000000
[$t1] = 00000000, [$t2] = 00000000, [$t3] = 00000000
[$t4] = 00000000, [$t5] = 00000000, [$t6] = 00000000
[$t7] = 00000000, [$t8] = 00000000, [$t9] = 00000000
=====
```

cycle	Instruction	IF	ID	EX	MEM	WB	Results
3	3. and \$s0, \$t0, \$t1	3	2	1	x	x	

```

=====
Time:          300, CLK = 0, PC = 0000000c
[$s0] = 00000000, [$s1] = 00000000, [$s2] = 00000000
[$s3] = 00000000, [$s4] = 00000000, [$s5] = 00000000
[$s6] = 00000000, [$s7] = 00000000, [$t0] = 00000000
[$t1] = 00000000, [$t2] = 00000000, [$t3] = 00000000
[$t4] = 00000000, [$t5] = 00000000, [$t6] = 00000000
[$t7] = 00000000, [$t8] = 00000000, [$t9] = 00000000
Time:          350, CLK = 1, PC = 0000000c
[$s0] = 00000000, [$s1] = 00000000, [$s2] = 00000000
[$s3] = 00000000, [$s4] = 00000000, [$s5] = 00000000
[$s6] = 00000000, [$s7] = 00000000, [$t0] = 00000000
[$t1] = 00000000, [$t2] = 00000000, [$t3] = 00000000
[$t4] = 00000000, [$t5] = 00000000, [$t6] = 00000000
[$t7] = 00000000, [$t8] = 00000000, [$t9] = 00000000
=====
```

cycle	Instruction	IF	ID	EX	MEM	WB	Results
4	4. or \$s0, \$t0, \$t1	4	3	2	1	x	

```
=====
Time:          400, CLK = 0, PC = 00000010
[$s0] = 00000000, [$s1] = 00000000, [$s2] = 00000000
[$s3] = 00000000, [$s4] = 00000000, [$s5] = 00000000
[$s6] = 00000000, [$s7] = 00000000, [$t0] = 00000000
[$t1] = 00000000, [$t2] = 00000000, [$t3] = 00000000
[$t4] = 00000000, [$t5] = 00000000, [$t6] = 00000000
[$t7] = 00000000, [$t8] = 00000000, [$t9] = 00000000
Time:          450, CLK = 1, PC = 00000010
[$s0] = 00000000, [$s1] = 00000000, [$s2] = 00000000
[$s3] = 00000000, [$s4] = 00000000, [$s5] = 00000000
[$s6] = 00000000, [$s7] = 00000000, [$t0] = 00000020
[$t1] = 00000000, [$t2] = 00000000, [$t3] = 00000000
[$t4] = 00000000, [$t5] = 00000000, [$t6] = 00000000
[$t7] = 00000000, [$t8] = 00000000, [$t9] = 00000000
=====
```

cycle	Instruction	IF	ID	EX	MEM	WB	Results
5	5. sw \$s0 , 4(\$zero)	5	4	3	2	1	\$t0 = 0x20

```
=====
Time:          500, CLK = 0, PC = 00000014
[$s0] = 00000000, [$s1] = 00000000, [$s2] = 00000000
[$s3] = 00000000, [$s4] = 00000000, [$s5] = 00000000
[$s6] = 00000000, [$s7] = 00000000, [$t0] = 00000020
[$t1] = 00000000, [$t2] = 00000000, [$t3] = 00000000
[$t4] = 00000000, [$t5] = 00000000, [$t6] = 00000000
[$t7] = 00000000, [$t8] = 00000000, [$t9] = 00000000
Time:          550, CLK = 1, PC = 00000014
[$s0] = 00000000, [$s1] = 00000000, [$s2] = 00000000
[$s3] = 00000000, [$s4] = 00000000, [$s5] = 00000000
[$s6] = 00000000, [$s7] = 00000000, [$t0] = 00000020
[$t1] = 00000037, [$t2] = 00000000, [$t3] = 00000000
[$t4] = 00000000, [$t5] = 00000000, [$t6] = 00000000
[$t7] = 00000000, [$t8] = 00000000, [$t9] = 00000000
=====
```

cycle	Instruction	IF	ID	EX	MEM	WB	Results
6	6. sw \$t0 , 8(\$zero)	6	5	4	3	2	\$t1 = 0x37

```
=====
Time:          600, CLK = 0, PC = 00000018
[$s0] = 00000000, [$s1] = 00000000, [$s2] = 00000000
[$s3] = 00000000, [$s4] = 00000000, [$s5] = 00000000
[$s6] = 00000000, [$s7] = 00000000, [$t0] = 00000020
[$t1] = 00000037, [$t2] = 00000000, [$t3] = 00000000
[$t4] = 00000000, [$t5] = 00000000, [$t6] = 00000000
[$t7] = 00000000, [$t8] = 00000000, [$t9] = 00000000
Time:          650, CLK = 1, PC = 00000018
[$s0] = 00000020, [$s1] = 00000000, [$s2] = 00000000
[$s3] = 00000000, [$s4] = 00000000, [$s5] = 00000000
[$s6] = 00000000, [$s7] = 00000000, [$t0] = 00000020
[$t1] = 00000037, [$t2] = 00000000, [$t3] = 00000000
[$t4] = 00000000, [$t5] = 00000000, [$t6] = 00000000
[$t7] = 00000000, [$t8] = 00000000, [$t9] = 00000000
=====
```

cycle	Instruction	IF	ID	EX	MEM	WB	Results
7	7. add \$s1 , \$t0 , \$t1	7	6	5	4	3	\$s0 = 0x20

```
=====
Time:          700, CLK = 0, PC = 0000001c
[$s0] = 00000020, [$s1] = 00000000, [$s2] = 00000000
[$s3] = 00000000, [$s4] = 00000000, [$s5] = 00000000
[$s6] = 00000000, [$s7] = 00000000, [$t0] = 00000020
[$t1] = 00000037, [$t2] = 00000000, [$t3] = 00000000
[$t4] = 00000000, [$t5] = 00000000, [$t6] = 00000000
[$t7] = 00000000, [$t8] = 00000000, [$t9] = 00000000
Time:          750, CLK = 1, PC = 0000001c
[$s0] = 00000037, [$s1] = 00000000, [$s2] = 00000000
[$s3] = 00000000, [$s4] = 00000000, [$s5] = 00000000
[$s6] = 00000000, [$s7] = 00000000, [$t0] = 00000020
[$t1] = 00000037, [$t2] = 00000000, [$t3] = 00000000
[$t4] = 00000000, [$t5] = 00000000, [$t6] = 00000000
[$t7] = 00000000, [$t8] = 00000000, [$t9] = 00000000
=====
```

cycle	Instruction	IF	ID	EX	MEM	WB	Results
8	8. sub \$s2, \$t0, \$t1	8	7	6	5	4	\$s0 = 0x37; Data.Mem[4] = 0x37

```
=====
Time:          800, CLK = 0, PC = 00000020
[$s0] = 00000037, [$s1] = 00000000, [$s2] = 00000000
[$s3] = 00000000, [$s4] = 00000000, [$s5] = 00000000
[$s6] = 00000000, [$s7] = 00000000, [$t0] = 00000020
[$t1] = 00000037, [$t2] = 00000000, [$t3] = 00000000
[$t4] = 00000000, [$t5] = 00000000, [$t6] = 00000000
[$t7] = 00000000, [$t8] = 00000000, [$t9] = 00000000
Time:          850, CLK = 1, PC = 00000020
[$s0] = 00000037, [$s1] = 00000000, [$s2] = 00000000
[$s3] = 00000000, [$s4] = 00000000, [$s5] = 00000000
[$s6] = 00000000, [$s7] = 00000000, [$t0] = 00000020
[$t1] = 00000037, [$t2] = 00000000, [$t3] = 00000000
[$t4] = 00000000, [$t5] = 00000000, [$t6] = 00000000
[$t7] = 00000000, [$t8] = 00000000, [$t9] = 00000000
=====
```

cycle	Instruction	IF	ID	EX	MEM	WB	Results
9	9. beq \$s1, \$s2, error0	9	8	7	6	5	Data.Mem[8] = 0x20

```
=====
Time:          900, CLK = 0, PC = 00000024
[$s0] = 00000037, [$s1] = 00000000, [$s2] = 00000000
[$s3] = 00000000, [$s4] = 00000000, [$s5] = 00000000
[$s6] = 00000000, [$s7] = 00000000, [$t0] = 00000020
[$t1] = 00000037, [$t2] = 00000000, [$t3] = 00000000
[$t4] = 00000000, [$t5] = 00000000, [$t6] = 00000000
[$t7] = 00000000, [$t8] = 00000000, [$t9] = 00000000
Time:          950, CLK = 1, PC = 00000024
[$s0] = 00000037, [$s1] = 00000000, [$s2] = 00000000
[$s3] = 00000000, [$s4] = 00000000, [$s5] = 00000000
[$s6] = 00000000, [$s7] = 00000000, [$t0] = 00000020
[$t1] = 00000037, [$t2] = 00000000, [$t3] = 00000000
[$t4] = 00000000, [$t5] = 00000000, [$t6] = 00000000
[$t7] = 00000000, [$t8] = 00000000, [$t9] = 00000000
=====
```

cycle	Instruction	IF	ID	EX	MEM	WB	Results
10	10. lw \$s1, 4(\$zero)	10	9	8	7	6	

```
=====
Time:          1000, CLK = 0, PC = 00000024
[$s0] = 00000037, [$s1] = 00000000, [$s2] = 00000000
[$s3] = 00000000, [$s4] = 00000000, [$s5] = 00000000
[$s6] = 00000000, [$s7] = 00000000, [$t0] = 00000020
[$t1] = 00000037, [$t2] = 00000000, [$t3] = 00000000
[$t4] = 00000000, [$t5] = 00000000, [$t6] = 00000000
[$t7] = 00000000, [$t8] = 00000000, [$t9] = 00000000
INFO: [USF-XSim-96] XSim completed. Design snapshot 's
INFO: [USF-XSim-97] XSim simulation ran for 1000ns
launch_simulation: Time (s): cpu = 00:00:08 ; elapsed
run all
Time:          1050, CLK = 1, PC = 00000024
[$s0] = 00000037, [$s1] = 00000057, [$s2] = 00000000
[$s3] = 00000000, [$s4] = 00000000, [$s5] = 00000000
[$s6] = 00000000, [$s7] = 00000000, [$t0] = 00000020
[$t1] = 00000037, [$t2] = 00000000, [$t3] = 00000000
[$t4] = 00000000, [$t5] = 00000000, [$t6] = 00000000
[$t7] = 00000000, [$t8] = 00000000, [$t9] = 00000000
=====
```

cycle	Instruction	IF	ID	EX	MEM	WB	Results
11		10	9	nop	8	7	\$s1 = 0x57

```
=====
Time:          1100, CLK = 0, PC = 00000028
[$s0] = 00000037, [$s1] = 00000057, [$s2] = 00000000
[$s3] = 00000000, [$s4] = 00000000, [$s5] = 00000000
[$s6] = 00000000, [$s7] = 00000000, [$t0] = 00000020
[$t1] = 00000037, [$t2] = 00000000, [$t3] = 00000000
[$t4] = 00000000, [$t5] = 00000000, [$t6] = 00000000
[$t7] = 00000000, [$t8] = 00000000, [$t9] = 00000000
Time:          1150, CLK = 1, PC = 00000028
[$s0] = 00000037, [$s1] = 00000057, [$s2] = ffffffe9
[$s3] = 00000000, [$s4] = 00000000, [$s5] = 00000000
[$s6] = 00000000, [$s7] = 00000000, [$t0] = 00000020
[$t1] = 00000037, [$t2] = 00000000, [$t3] = 00000000
[$t4] = 00000000, [$t5] = 00000000, [$t6] = 00000000
[$t7] = 00000000, [$t8] = 00000000, [$t9] = 00000000
=====
```

cycle	Instruction	IF	ID	EX	MEM	WB	Results
12	11. andi \$s2, \$s1, 0x48	11	10	9	nop	8	\$s2 = 0xfffffe9

```
=====
Time:          1200, CLK = 0, PC = 0000002c
[$s0] = 00000037, [$s1] = 00000057, [$s2] = ffffffe9
[$s3] = 00000000, [$s4] = 00000000, [$s5] = 00000000
[$s6] = 00000000, [$s7] = 00000000, [$t0] = 00000020
[$t1] = 00000037, [$t2] = 00000000, [$t3] = 00000000
[$t4] = 00000000, [$t5] = 00000000, [$t6] = 00000000
[$t7] = 00000000, [$t8] = 00000000, [$t9] = 00000000
Time:          1250, CLK = 1, PC = 0000002c
[$s0] = 00000037, [$s1] = 00000057, [$s2] = ffffffe9
[$s3] = 00000000, [$s4] = 00000000, [$s5] = 00000000
[$s6] = 00000000, [$s7] = 00000000, [$t0] = 00000020
[$t1] = 00000037, [$t2] = 00000000, [$t3] = 00000000
[$t4] = 00000000, [$t5] = 00000000, [$t6] = 00000000
[$t7] = 00000000, [$t8] = 00000000, [$t9] = 00000000
=====
```

cycle	Instruction	IF	ID	EX	MEM	WB	Results
13	12. beq \$s1, \$s2, error1	12	11	10	9	nop	

```
=====
Time:          1300, CLK = 0, PC = 00000030
[$s0] = 00000037, [$s1] = 00000057, [$s2] = ffffffe9
[$s3] = 00000000, [$s4] = 00000000, [$s5] = 00000000
[$s6] = 00000000, [$s7] = 00000000, [$t0] = 00000020
[$t1] = 00000037, [$t2] = 00000000, [$t3] = 00000000
[$t4] = 00000000, [$t5] = 00000000, [$t6] = 00000000
[$t7] = 00000000, [$t8] = 00000000, [$t9] = 00000000
Time:          1350, CLK = 1, PC = 00000030
[$s0] = 00000037, [$s1] = 00000057, [$s2] = ffffffe9
[$s3] = 00000000, [$s4] = 00000000, [$s5] = 00000000
[$s6] = 00000000, [$s7] = 00000000, [$t0] = 00000020
[$t1] = 00000037, [$t2] = 00000000, [$t3] = 00000000
[$t4] = 00000000, [$t5] = 00000000, [$t6] = 00000000
[$t7] = 00000000, [$t8] = 00000000, [$t9] = 00000000
=====
```

cycle	Instruction	IF	ID	EX	MEM	WB	Results
14		12	11	nop	10	9	

```
=====
Time:          1400, CLK = 0, PC = 00000034
[$s0] = 00000037, [$s1] = 00000057, [$s2] = ffffffe9
[$s3] = 00000000, [$s4] = 00000000, [$s5] = 00000000
[$s6] = 00000000, [$s7] = 00000000, [$t0] = 00000020
[$t1] = 00000037, [$t2] = 00000000, [$t3] = 00000000
[$t4] = 00000000, [$t5] = 00000000, [$t6] = 00000000
[$t7] = 00000000, [$t8] = 00000000, [$t9] = 00000000
Time:          1450, CLK = 1, PC = 00000034
[$s0] = 00000037, [$s1] = 00000037, [$s2] = ffffffe9
[$s3] = 00000000, [$s4] = 00000000, [$s5] = 00000000
[$s6] = 00000000, [$s7] = 00000000, [$t0] = 00000020
[$t1] = 00000037, [$t2] = 00000000, [$t3] = 00000000
[$t4] = 00000000, [$t5] = 00000000, [$t6] = 00000000
[$t7] = 00000000, [$t8] = 00000000, [$t9] = 00000000
=====
```

cycle	Instruction	IF	ID	EX	MEM	WB	Results
15	13. lw \$s3, 8(\$zero)	13	12	11	nop	10	\$s1 = 0x37

```
=====
Time:          1500, CLK = 0, PC = 00000034
[$s0] = 00000037, [$s1] = 00000037, [$s2] = ffffffe9
[$s3] = 00000000, [$s4] = 00000000, [$s5] = 00000000
[$s6] = 00000000, [$s7] = 00000000, [$t0] = 00000020
[$t1] = 00000037, [$t2] = 00000000, [$t3] = 00000000
[$t4] = 00000000, [$t5] = 00000000, [$t6] = 00000000
[$t7] = 00000000, [$t8] = 00000000, [$t9] = 00000000
Time:          1550, CLK = 1, PC = 00000034
[$s0] = 00000037, [$s1] = 00000037, [$s2] = ffffffe9
[$s3] = 00000020, [$s4] = 00000000, [$s5] = 00000000
[$s6] = 00000000, [$s7] = 00000000, [$t0] = 00000020
[$t1] = 00000037, [$t2] = 00000000, [$t3] = 00000000
[$t4] = 00000000, [$t5] = 00000000, [$t6] = 00000000
[$t7] = 00000000, [$t8] = 00000000, [$t9] = 00000000
=====
```

cycle	Instruction	IF	ID	EX	MEM	WB	Results
16		13	12	nop	11	nop	

```
=====
Time:          1600, CLK = 0, PC = 00000038
[$s0] = 00000037, [$s1] = 00000037, [$s2] = ffffffe9
[$s3] = 00000020, [$s4] = 00000000, [$s5] = 00000000
[$s6] = 00000000, [$s7] = 00000000, [$t0] = 00000020
[$t1] = 00000037, [$t2] = 00000000, [$t3] = 00000000
[$t4] = 00000000, [$t5] = 00000000, [$t6] = 00000000
[$t7] = 00000000, [$t8] = 00000000, [$t9] = 00000000
Time:          1650, CLK = 1, PC = 00000038
[$s0] = 00000037, [$s1] = 00000037, [$s2] = 00000000
[$s3] = 00000020, [$s4] = 00000000, [$s5] = 00000000
[$s6] = 00000000, [$s7] = 00000000, [$t0] = 00000020
[$t1] = 00000037, [$t2] = 00000000, [$t3] = 00000000
[$t4] = 00000000, [$t5] = 00000000, [$t6] = 00000000
[$t7] = 00000000, [$t8] = 00000000, [$t9] = 00000000
=====
```

cycle	Instruction	IF	ID	EX	MEM	WB	Results
17	14. beq \$s0, \$s3, error2	14	13	12	nop	11	\$s2 = 0x0

```
=====
Time:          1700, CLK = 0, PC = 0000003c
[$s0] = 00000037, [$s1] = 00000037, [$s2] = 00000000
[$s3] = 00000020, [$s4] = 00000000, [$s5] = 00000000
[$s6] = 00000000, [$s7] = 00000000, [$t0] = 00000020
[$t1] = 00000037, [$t2] = 00000000, [$t3] = 00000000
[$t4] = 00000000, [$t5] = 00000000, [$t6] = 00000000
[$t7] = 00000000, [$t8] = 00000000, [$t9] = 00000000
Time:          1750, CLK = 1, PC = 0000003c
[$s0] = 00000037, [$s1] = 00000037, [$s2] = 00000000
[$s3] = 00000020, [$s4] = 00000000, [$s5] = 00000000
[$s6] = 00000000, [$s7] = 00000000, [$t0] = 00000020
[$t1] = 00000037, [$t2] = 00000000, [$t3] = 00000000
[$t4] = 00000000, [$t5] = 00000000, [$t6] = 00000000
[$t7] = 00000000, [$t8] = 00000000, [$t9] = 00000000
=====
```

cycle	Instruction	IF	ID	EX	MEM	WB	Results
18	15. slt \$s4, \$s2, \$s1 (Last)	15	14	13	12	nop	

```
=====
Time:          1800, CLK = 0, PC = 00000040
[$s0] = 00000037, [$s1] = 00000037, [$s2] = 00000000
[$s3] = 00000020, [$s4] = 00000000, [$s5] = 00000000
[$s6] = 00000000, [$s7] = 00000000, [$t0] = 00000020
[$t1] = 00000037, [$t2] = 00000000, [$t3] = 00000000
[$t4] = 00000000, [$t5] = 00000000, [$t6] = 00000000
[$t7] = 00000000, [$t8] = 00000000, [$t9] = 00000000
Time:          1850, CLK = 1, PC = 00000040
[$s0] = 00000037, [$s1] = 00000037, [$s2] = 00000000
[$s3] = 00000020, [$s4] = 00000000, [$s5] = 00000000
[$s6] = 00000000, [$s7] = 00000000, [$t0] = 00000020
[$t1] = 00000037, [$t2] = 00000000, [$t3] = 00000000
[$t4] = 00000000, [$t5] = 00000000, [$t6] = 00000000
[$t7] = 00000000, [$t8] = 00000000, [$t9] = 00000000
=====
```

cycle	Instruction	IF	ID	EX	MEM	WB	Results
19		15	14	nop	13	12	

```
=====
Time:          1900, CLK = 0, PC = 00000040
[$s0] = 00000037, [$s1] = 00000037, [$s2] = 00000000
[$s3] = 00000020, [$s4] = 00000000, [$s5] = 00000000
[$s6] = 00000000, [$s7] = 00000000, [$t0] = 00000020
[$t1] = 00000037, [$t2] = 00000000, [$t3] = 00000000
[$t4] = 00000000, [$t5] = 00000000, [$t6] = 00000000
[$t7] = 00000000, [$t8] = 00000000, [$t9] = 00000000
Time:          1950, CLK = 1, PC = 00000040
[$s0] = 00000037, [$s1] = 00000037, [$s2] = 00000000
[$s3] = 00000020, [$s4] = 00000000, [$s5] = 00000000
[$s6] = 00000000, [$s7] = 00000000, [$t0] = 00000020
[$t1] = 00000037, [$t2] = 00000000, [$t3] = 00000000
[$t4] = 00000000, [$t5] = 00000000, [$t6] = 00000000
[$t7] = 00000000, [$t8] = 00000000, [$t9] = 00000000
=====
```

cycle	Instruction	IF	ID	EX	MEM	WB	Results
20		15	14	nop	nop	13	[\$s3] = 0x20

```
=====
Time:          2000, CLK = 0, PC = 00000044
[$s0] = 00000037, [$s1] = 00000037, [$s2] = 00000000
[$s3] = 00000020, [$s4] = 00000000, [$s5] = 00000000
[$s6] = 00000000, [$s7] = 00000000, [$t0] = 00000020
[$t1] = 00000037, [$t2] = 00000000, [$t3] = 00000000
[$t4] = 00000000, [$t5] = 00000000, [$t6] = 00000000
[$t7] = 00000000, [$t8] = 00000000, [$t9] = 00000000
Time:          2050, CLK = 1, PC = 00000044
[$s0] = 00000037, [$s1] = 00000037, [$s2] = 00000000
[$s3] = 00000020, [$s4] = 00000001, [$s5] = 00000000
[$s6] = 00000000, [$s7] = 00000000, [$t0] = 00000020
[$t1] = 00000037, [$t2] = 00000000, [$t3] = 00000000
[$t4] = 00000000, [$t5] = 00000000, [$t6] = 00000000
[$t7] = 00000000, [$t8] = 00000000, [$t9] = 00000000
=====
```

cycle	Instruction	IF	ID	EX	MEM	WB	Results
21	16. beq \$s4, \$0 , EXIT	16	15	14	nop	nop	

```
=====
Time:          2100, CLK = 0, PC = 00000048
[$s0] = 00000037, [$s1] = 00000037, [$s2] = 00000000
[$s3] = 00000020, [$s4] = 00000001, [$s5] = 00000000
[$s6] = 00000000, [$s7] = 00000000, [$t0] = 00000020
[$t1] = 00000037, [$t2] = 00000000, [$t3] = 00000000
[$t4] = 00000000, [$t5] = 00000000, [$t6] = 00000000
[$t7] = 00000000, [$t8] = 00000000, [$t9] = 00000000
Time:          2150, CLK = 1, PC = 00000048
[$s0] = 00000037, [$s1] = 00000037, [$s2] = 00000000
[$s3] = 00000020, [$s4] = 00000001, [$s5] = 00000000
[$s6] = 00000000, [$s7] = 00000000, [$t0] = 00000020
[$t1] = 00000037, [$t2] = 00000000, [$t3] = 00000000
[$t4] = 00000000, [$t5] = 00000000, [$t6] = 00000000
[$t7] = 00000000, [$t8] = 00000000, [$t9] = 00000000
=====
```

cycle	Instruction	IF	ID	EX	MEM	WB	Results
22	17. add \$s2 , \$s1 , \$0	17	16	15	14	nop	

```
=====
Time: 2200, CLK = 0, PC = 00000038
[$s0] = 00000037, [$s1] = 00000037, [$s2] = 00000000
[$s3] = 00000020, [$s4] = 00000001, [$s5] = 00000000
[$s6] = 00000000, [$s7] = 00000000, [$t0] = 00000020
[$t1] = 00000037, [$t2] = 00000000, [$t3] = 00000000
[$t4] = 00000000, [$t5] = 00000000, [$t6] = 00000000
[$t7] = 00000000, [$t8] = 00000000, [$t9] = 00000000
Time: 2250, CLK = 1, PC = 00000038
[$s0] = 00000037, [$s1] = 00000037, [$s2] = 00000000
[$s3] = 00000020, [$s4] = 00000001, [$s5] = 00000000
[$s6] = 00000000, [$s7] = 00000000, [$t0] = 00000020
[$t1] = 00000037, [$t2] = 00000000, [$t3] = 00000000
[$t4] = 00000000, [$t5] = 00000000, [$t6] = 00000000
[$t7] = 00000000, [$t8] = 00000000, [$t9] = 00000000
=====
```

cycle	Instruction	IF	ID	EX	MEM	WB	Results
23		17	16	nop	15	14	

```
=====
Time: 2300, CLK = 0, PC = 0000003c
[$s0] = 00000037, [$s1] = 00000037, [$s2] = 00000000
[$s3] = 00000020, [$s4] = 00000001, [$s5] = 00000000
[$s6] = 00000000, [$s7] = 00000000, [$t0] = 00000020
[$t1] = 00000037, [$t2] = 00000000, [$t3] = 00000000
[$t4] = 00000000, [$t5] = 00000000, [$t6] = 00000000
[$t7] = 00000000, [$t8] = 00000000, [$t9] = 00000000
Time: 2350, CLK = 1, PC = 0000003c
[$s0] = 00000037, [$s1] = 00000037, [$s2] = 00000037
[$s3] = 00000020, [$s4] = 00000001, [$s5] = 00000000
[$s6] = 00000000, [$s7] = 00000000, [$t0] = 00000020
[$t1] = 00000037, [$t2] = 00000000, [$t3] = 00000000
[$t4] = 00000000, [$t5] = 00000000, [$t6] = 00000000
[$t7] = 00000000, [$t8] = 00000000, [$t9] = 00000000
=====
```

cycle	Instruction	IF	ID	EX	MEM	WB	Results
24	18. j Last	18	17	16	nop	15	\$s4 = 0x1

```
=====
Time: 2400, CLK = 0, PC = 00000040
[$s0] = 00000037, [$s1] = 00000037, [$s2] = 00000037
[$s3] = 00000020, [$s4] = 00000001, [$s5] = 00000000
[$s6] = 00000000, [$s7] = 00000000, [$t0] = 00000020
[$t1] = 00000037, [$t2] = 00000000, [$t3] = 00000000
[$t4] = 00000000, [$t5] = 00000000, [$t6] = 00000000
[$t7] = 00000000, [$t8] = 00000000, [$t9] = 00000000
Time: 2450, CLK = 1, PC = 00000040
[$s0] = 00000037, [$s1] = 00000037, [$s2] = 00000037
[$s3] = 00000020, [$s4] = 00000001, [$s5] = 00000000
[$s6] = 00000000, [$s7] = 00000000, [$t0] = 00000020
[$t1] = 00000037, [$t2] = 00000000, [$t3] = 00000000
[$t4] = 00000000, [$t5] = 00000000, [$t6] = 00000000
[$t7] = 00000000, [$t8] = 00000000, [$t9] = 00000000
=====
```

cycle	Instruction	IF	ID	EX	MEM	WB	Results
25	19. addi \$t0, \$0, 0(error0)	19	18	17	16	nop	

```
=====
Time:          2500, CLK = 0, PC = 00000040
[$s0] = 00000037, [$s1] = 00000037, [$s2] = 00000037
[$s3] = 00000020, [$s4] = 00000001, [$s5] = 00000000
[$s6] = 00000000, [$s7] = 00000000, [$t0] = 00000020
[$t1] = 00000037, [$t2] = 00000000, [$t3] = 00000000
[$t4] = 00000000, [$t5] = 00000000, [$t6] = 00000000
[$t7] = 00000000, [$t8] = 00000000, [$t9] = 00000000
Time:          2550, CLK = 1, PC = 00000040
[$s0] = 00000037, [$s1] = 00000037, [$s2] = 00000037
[$s3] = 00000020, [$s4] = 00000001, [$s5] = 00000000
[$s6] = 00000000, [$s7] = 00000000, [$t0] = 00000020
[$t1] = 00000037, [$t2] = 00000000, [$t3] = 00000000
[$t4] = 00000000, [$t5] = 00000000, [$t6] = 00000000
[$t7] = 00000000, [$t8] = 00000000, [$t9] = 00000000
=====
```

cycle	Instruction	IF	ID	EX	MEM	WB	Results
26	15. slt \$s4, \$s2, \$s1 (Last)	15	flush	18	17	16	

```
=====
Time:          2600, CLK = 0, PC = 0000007c
[$s0] = 00000037, [$s1] = 00000037, [$s2] = 00000037
[$s3] = 00000020, [$s4] = 00000001, [$s5] = 00000000
[$s6] = 00000000, [$s7] = 00000000, [$t0] = 00000020
[$t1] = 00000037, [$t2] = 00000000, [$t3] = 00000000
[$t4] = 00000000, [$t5] = 00000000, [$t6] = 00000000
[$t7] = 00000000, [$t8] = 00000000, [$t9] = 00000000
Time:          2650, CLK = 1, PC = 0000007c
[$s0] = 00000037, [$s1] = 00000037, [$s2] = 00000037
[$s3] = 00000020, [$s4] = 00000000, [$s5] = 00000000
[$s6] = 00000000, [$s7] = 00000000, [$t0] = 00000020
[$t1] = 00000037, [$t2] = 00000000, [$t3] = 00000000
[$t4] = 00000000, [$t5] = 00000000, [$t6] = 00000000
[$t7] = 00000000, [$t8] = 00000000, [$t9] = 00000000
=====
```

cycle	Instruction	IF	ID	EX	MEM	WB	Results
27	16. beq \$s4, \$0, EXIT	16	15	flush	18	17	\$s2 = 0x37

```
=====
Time:          2700, CLK = 0, PC = 00000080
[$s0] = 00000037, [$s1] = 00000037, [$s2] = 00000037
[$s3] = 00000020, [$s4] = 00000000, [$s5] = 00000000
[$s6] = 00000000, [$s7] = 00000000, [$t0] = 00000020
[$t1] = 00000037, [$t2] = 00000000, [$t3] = 00000000
[$t4] = 00000000, [$t5] = 00000000, [$t6] = 00000000
[$t7] = 00000000, [$t8] = 00000000, [$t9] = 00000000
Time:          2750, CLK = 1, PC = 00000080
[$s0] = 00000037, [$s1] = 00000037, [$s2] = 00000037
[$s3] = 00000020, [$s4] = 00000000, [$s5] = 00000000
[$s6] = 00000000, [$s7] = 00000000, [$t0] = 00000020
[$t1] = 00000037, [$t2] = 00000000, [$t3] = 00000000
[$t4] = 00000000, [$t5] = 00000000, [$t6] = 00000000
[$t7] = 00000000, [$t8] = 00000000, [$t9] = 00000000
=====
```

cycle	Instruction	IF	ID	EX	MEM	WB	Results
28	17. add \$s2,\$s1, \$0	17	16	15	flush	18	

```
=====
Time:          2800, CLK = 0, PC = 00000084
[$s0] = 00000037, [$s1] = 00000037, [$s2] = 00000037
[$s3] = 00000020, [$s4] = 00000000, [$s5] = 00000000
[$s6] = 00000000, [$s7] = 00000000, [$t0] = 00000020
[$t1] = 00000037, [$t2] = 00000000, [$t3] = 00000000
[$t4] = 00000000, [$t5] = 00000000, [$t6] = 00000000
[$t7] = 00000000, [$t8] = 00000000, [$t9] = 00000000
Time:          2850, CLK = 1, PC = 00000084
[$s0] = 00000037, [$s1] = 00000037, [$s2] = 00000037
[$s3] = 00000020, [$s4] = 00000000, [$s5] = 00000000
[$s6] = 00000000, [$s7] = 00000000, [$t0] = 00000020
[$t1] = 00000037, [$t2] = 00000000, [$t3] = 00000000
[$t4] = 00000000, [$t5] = 00000000, [$t6] = 00000000
[$t7] = 00000000, [$t8] = 00000000, [$t9] = 00000000
=====
```

cycle	Instruction	IF	ID	EX	MEM	WB	Results
29		17	16	nop	15	flush	

Conclusion & Discussion

In this project, we developed a 32-bit pipelined MIPS processor in Verilog. We simulate the program in Vivado and implemented it on FPGA board. Beyond the lw hazard and forwarding unit, we also resolved beq related hazards. In general, the program went smooth, but there are still some concerns:

1. To avoid hazards as much as possible, we update the data in the memory on negative edge of the clock, which may lengthen the critical path and impact the performance.
2. When consider the control hazards, we only introduce IF. Flush. Without considering ID. Flush and EX. Flush, we couldn't solve control hazards ahead of ID stage.

The performance of the program can be improved by introducing more instructions (there are lw, sw, add, addi, sub, and, andi, or, slt, beq, bne and j now), extend the capacity of data memory and instruction memory, and reducing the critical path, to list a few. These improvements may need more stages and more complicated hazard detection unit, which could be implemented in the future.

Reference

[1] Zheng, G., 2020. *Ve370 Introduction To Computer Organization Project 2*.

Appendix

Source Code

ALU.v

```
1 `timescale 1ns / 1ps
2
3
4 module ALU(result, a, b, ALU_control);
5     parameter word = 32;
6     input [word - 1:0] a, b;
7     input [3:0] ALU_control;
8     output [word - 1:0] result;
9
10    reg [word - 1:0] result;
```

```

12     initial begin
13         result = 0;
14     end
15
16     always @(a or b or ALU_control) begin
17         case (ALU_control)
18             // add
19             4'b0010: result = a + b;
20             // sub
21             4'b0110: result = a - b;
22             // and
23             4'b0000: result = a & b;
24             // or
25             4'b0001: result = a | b;
26             // slt
27             4'b0111: result = (a < b)?1:0;
28             default: result = 0;
29         endcase
30     end
31 endmodule

```

ALU_control.v

```

1 `timescale 1ns / 1ps
2
3 module ALUcontrol(ALU_control, funct, ALU_op);
4     input [1:0] ALU_op;
5     input [5:0] funct;
6     output [3:0] ALU_control;
7
8     reg [3:0] ALU_control;
9
10    initial begin
11        ALU_control = 0;
12    end
13
14    always @ (funct or ALU_op) begin
15        case (ALU_op)
16            2'b00: ALU_control = 4'b0010;
17            2'b01: ALU_control = 4'b0110;
18            2'b11: ALU_control = 4'b0000;
19            2'b10: begin
20                case (funct)
21                    6'b100000: ALU_control = 4'b0010;
22                    6'b100010: ALU_control = 4'b0110;
23                    6'b100101: ALU_control = 4'b0001;
24                    6'b101010: ALU_control = 4'b0111;
25                    6'b100100: ALU_control = 4'b0000;
26                    default: ALU_control = 4'b0000;
27                endcase
28            end
29            default: ALU_control <= 4'b0000;
30        endcase
31    end
32 endmodule

```

EX_MEM.v

```

1 module EX_MEM (MEM_MemRead, MEM_MemtoReg, MEM_MemWrite, MEM_RegWrite,
2   MEM_ALU_result, MEM_MUX6_out, MEM_MUX8_out,
3   Clock, EX_MemRead, EX_MemtoReg, EX_Memwrite, EX_Regwrite,
4   EX_ALU_result, EX_MUX6_out, EX_MUX8_out);
5   parameter rwidth = 5;
6   parameter word = 32;
7   input          clock,           EX_MemRead,      EX_MemtoReg,
8   EX_MemWrite,  EX_RegWrite;
9   input [rwidth - 1:0] EX_MUX8_out;
10  input [word - 1:0]  EX_ALU_result,   EX_MUX6_out;
11  output          MEM_MemRead,     MEM_MemtoReg,   MEM_Memwrite,
12  MEM_RegWrite;
13  output [rwidth - 1:0] MEM_MUX8_out;
14  output [word - 1:0]  MEM_ALU_result,  MEM_MUX6_out;
15  reg             MEM_MemRead,     MEM_MemtoReg,   MEM_Memwrite,
16  MEM_RegWrite;
17  reg             [rwidth - 1:0] MEM_MUX8_out;
18  reg             [word - 1:0]  MEM_ALU_result,  MEM_MUX6_out;
19
20 initial begin
21   MEM_MemRead    <= 0;
22   MEM_MemtoReg   <= 0;
23   MEM_MemWrite   <= 0;
24   MEM_RegWrite   <= 0;
25   MEM_ALU_result <= 0;
26   MEM_MUX6_out   <= 0;
27   MEM_MUX8_out   <= 0;
28 end
29
30 always @ (posedge Clock) begin
31   MEM_MemRead    <= EX_MemRead;
32   MEM_MemtoReg   <= EX_MemtoReg;
33   MEM_MemWrite   <= EX_Memwrite;
34   MEM_RegWrite   <= EX_Regwrite;
35   MEM_ALU_result <= EX_ALU_result;
36   MEM_MUX6_out   <= EX_MUX6_out;
37   MEM_MUX8_out   <= EX_MUX8_out;
38 end
39 endmodule

```

ID_EX.v

```

1 module ID_EX(EX_RegDst, EX_MemRead, EX_MemtoReg, EX_ALUOp, EX_Memwrite,
2   EX_ALUSrc,
3   EX_RegWrite, EX_Registers_Read_data_1,
4   EX_Registers_Read_data_2,
5   EX_Sign_extend_out, ID_EX_RegisterRs, ID_EX_RegisterRt,
6   ID_EX_RegisterRd,
7   Clock, ID_EX_Flush, ID_RegDst, ID_MemRead, ID_MemtoReg,
8   ID_ALUOp,
9   ID_MemWrite, ID_ALUSrc, ID_RegWrite, ID_Registers_Read_data_1,
10  ID_Registers_Read_data_2, ID_Sign_extend_out, IF_ID_RegisterRs,
11  IF_ID_RegisterRt, IF_ID_RegisterRd);
12  parameter sel = 2;      // sel means the width of ALUOp
13  parameter rwidth = 5;   // rwidth means the width of the a register
14  number
15  parameter word = 32;

```

```

10    input                               clock, ID_EX_Flush,           ID_RegDst,
11          ID_MemRead,                  ID_MemtoReg,      ID_MemWrite,     ID_ALUSrc,
12          ID_RegWrite;
13          input [sel - 1:0]           ID_ALUOp;
14          input [rwidth - 1:0]        IF_ID_RegisterRs,
15          IF_ID_RegisterRt,          IF_ID_RegisterRd;
16          input [word - 1:0]         ID_Registers_Read_data_1,
17          ID_Registers_Read_data_2, ID_Sign_extend_out;
18          output                   EX_RegDst,             EX_MemRead,
19          EX_MemtoReg,      EX_MemWrite,   EX_ALUSrc,      EX_RegWrite;
20          output [sel - 1:0]           EX_ALUOp;
21          output [rwidth - 1:0]        ID_EX_RegisterRs,
22          ID_EX_RegisterRt,          ID_EX_RegisterRd;
23          output [word - 1:0]         EX_Registers_Read_data_1,
24          EX_Registers_Read_data_2, EX_Sign_extend_out;
25          reg                      EX_RegDst,             EX_MemRead,
26          EX_MemtoReg,      EX_MemWrite,   EX_ALUSrc,      EX_RegWrite;
27          reg [sel - 1:0]            EX_ALUOp;
28          reg [rwidth - 1:0]          ID_EX_RegisterRs,
29          ID_EX_RegisterRt,          ID_EX_RegisterRd;
30          reg [word - 1:0]           EX_Registers_Read_data_1,
31          EX_Registers_Read_data_2, EX_Sign_extend_out;
32
33      initial begin
34          EX_RegDst    <= 0;
35          EX_MemRead   <= 0;
36          EX_MemtoReg  <= 0;
37          EX_ALUOp     <= 0;
38          EX_MemWrite  <= 0;
39          EX_ALUSrc    <= 0;
40          EX_RegWrite  <= 0;
41          EX_Registers_Read_data_1  <= 0;
42          EX_Registers_Read_data_2  <= 0;
43          EX_Sign_extend_out       <= 0;
44          ID_EX_RegisterRs        <= 0;
45          ID_EX_RegisterRt        <= 0;
46          ID_EX_RegisterRd        <= 0;
47
48      end
49
50      always @ (posedge clock) begin
51          if (ID_EX_Flush == 1'b1) begin
52              EX_RegDst    <= 0;
53              EX_MemRead   <= 0;
54              EX_MemtoReg  <= 0;
55              EX_ALUOp     <= 0;
56              EX_MemWrite  <= 0;
57              EX_ALUSrc    <= 0;
58              EX_RegWrite  <= 0;
59          end
60          else begin
61              EX_RegDst    <= ID_RegDst;
62              EX_MemRead   <= ID_MemRead;
63              EX_MemtoReg  <= ID_MemtoReg;
64              EX_ALUOp     <= ID_ALUOp;
65              EX_MemWrite  <= ID_MemWrite;
66              EX_ALUSrc    <= ID_ALUSrc;
67              EX_RegWrite  <= ID_RegWrite;
68          end
69      end

```

```

58      EX_Registers_Read_data_1    <= ID_Registers_Read_data_1;
59      EX_Registers_Read_data_2    <= ID_Registers_Read_data_2;
60      EX_Sign_extend_out        <= ID_Sign_extend_out;
61      ID_EX_RegisterRs         <= IF_ID_RegisterRs;
62      ID_EX_RegisterRt         <= IF_ID_RegisterRt;
63      ID_EX_RegisterRd         <= IF_ID_RegisterRd;
64  end
65 endmodule

```

IF_ID.v

```

1 `timescale 1ns / 1ps
2
3 module IF_ID(IFID_pcplus4, IFID_instr, IF_pcplus4, IF_instr, IF_flush,
4 IFID_write, clk);
5     parameter word = 32;
6
7     input [word - 1:0] IF_pcplus4, IF_instr;
8     input IF_flush, IFID_write, clk;
9     output [word - 1:0] IFID_pcplus4, IFID_instr;
10
11    reg [word - 1:0] IFID_pcplus4, IFID_instr;
12
13    initial begin
14        IFID_instr <= 0;
15        IFID_pcplus4 <= 0;
16    end
17
18    always @(posedge clk) begin
19        if (IF_flush) begin
20            IFID_pcplus4 <= 0;
21            IFID_instr <= 0;
22        end
23        else if (IFID_write) begin
24            IFID_pcplus4 <= IF_pcplus4;
25            IFID_instr <= IF_instr;
26        end
27    end
28 endmodule

```

MEM_WB.v

```

1 module MEM_WB (WB_MemtoReg, WB_RegWrite, WB_Data_memory_Read_data,
2 WB_ALU_result, WB_MUX8_out,
3             Clock, MEM_RegWrite, MEM_MemtoReg, MEM_Data_memory_Read_data,
4 MEM_ALU_result, MEM_MUX8_out);
5     parameter rwidth = 5;
6     parameter word = 32;
7     input Clock, MEM_RegWrite, MEM_MemtoReg;
8     input [rwidth - 1:0] MEM_MUX8_out;
9     input [word - 1:0] MEM_Data_memory_Read_data, MEM_ALU_result;
10    output [rwidth - 1:0] WB_MUX8_out;
11    output [word - 1:0] WB_Data_memory_Read_data, WB_ALU_result;
12    reg [rwidth - 1:0] WB_MemtoReg, WB_RegWrite;
13    reg [rwidth - 1:0] WB_MUX8_out;

```

```

13    reg      [word - 1:0]    WB_Data_memory_Read_data,    WB_ALU_result;
14
15    initial begin
16        WB_MemtoReg           <= 0;
17        WB_RegWrite          <= 0;
18        WB_Data_memory_Read_data <= 0;
19        WB_ALU_result         <= 0;
20        WB_MUX8_out          <= 0;
21    end
22
23    always @ (posedge Clock) begin
24        WB_MemtoReg           <= MEM_MemtoReg;
25        WB_RegWrite          <= MEM_RegWrite;
26        WB_Data_memory_Read_data <= MEM_Data_memory_Read_data;
27        WB_ALU_result         <= MEM_ALU_result;
28        WB_MUX8_out          <= MEM_MUX8_out;
29    end
30 endmodule

```

PC.v

```

1 `timescale 1ns / 1ps
2
3 module PC(curr, clk, next, PCWrite, reset);
4     parameter word = 32;
5
6     input      [word - 1:0]      next;
7     input                  clk,    PCWrite,    reset;
8     output     [word - 1:0]      curr;
9
10    reg       [word - 1:0]      PC_mem;
11
12
13    always @(posedge clk or posedge reset) begin
14        if (reset == 1'b1) begin
15            PC_mem = 0;
16        end
17        else begin
18            if (PCWrite) begin
19                PC_mem = next;
20            end
21        end
22    end
23
24    assign curr = PC_mem;
25 endmodule

```

R_forward.v

```

1 `timescale 1ns / 1ps
2
3 module R_forward(forward_A, forward_B, EXMEM_regWrite, MEMWB_regWrite,
4                   EXMEM_regRd, MEMWB_regRd, INDEX_RegRs, INDEX_RegRt, clk);
5     input                      EXMEM_regWrite,
6             MEMWB_regWrite,    clk;
7     input      [4:0]           EXMEM_regRd,    MEMWB_regRd,    INDEX_RegRs,
8             INDEX_RegRt;

```

```

6   output  [1:0]      forward_A,      forward_B;
7   reg     [1:0]      forward_A,      forward_B;
8
9   initial begin
10    forward_A = 0;
11    forward_B = 0;
12  end
13
14  always @(`negedge clk) begin
15    // 1 & 2 hazard
16    forward_A[1] = EXMEM_regWrite & (EXMEM_regRd != 0) & (EXMEM_regRd
17 == INDEX_regRs);
18    forward_B[1] = EXMEM_regWrite & (EXMEM_regRd != 0) & (EXMEM_regRd
19 == INDEX_regRt);
20
21    // 1 & 3 hazard (need to make sure no 1 & 2 hazard)
22    forward_A[0] = MEMWB_regWrite & (MEMWB_regRd != 0) & (MEMWB_regRd
23 == INDEX_regRs) & (~(EXMEM_regWrite & EXMEM_regRd & (EXMEM_regRd ==
24 INDEX_regRs)));
25    forward_B[0] = MEMWB_regWrite & (MEMWB_regRd != 0) & (MEMWB_regRd
26 == INDEX_regRt) & (~(EXMEM_regWrite & EXMEM_regRd & (EXMEM_regRd ==
27 INDEX_regRt)));
28  end
29
30 endmodule

```

adder.v

```

1 `timescale 1ns / 1ps
2
3 module adder(result, a, b);
4   parameter word = 32;
5
6   input  [word - 1:0]   a,      b;
7   output [word - 1:0]   result;
8
9   reg    [word - 1:0]   result;
10
11  initial begin
12    result = 0;
13  end
14
15  always @(a, b) begin
16    result = a + b;
17  end
18 endmodule

```

brc_hazard.v

```

1 `timescale 1ns / 1ps
2
3 module brc_hazard(forward1, forward2, PCWrite, IFID_write, ID_flush,
4                   ID_branch, INDEX_regWrite, EXMEM_regWrite, EXMEM_memRead,
5                   EX_regDst, EXMEM_regRd, ID_regRt, ID_regRs, clk);
6   input          ID_branch, INDEX_regWrite, EXMEM_regWrite,
7         EXMEM_memRead, clk;

```

```

6      input [4:0]      EX_regDst,   EXMEM_regRd,      ID_regRt,
7      ID_regRs;
8      output         forward1,   forward2,        PCwrite,
9      IFID_write,    ID_flush;
10
11     reg             forward1,   forward2,        PCwrite,
12     IFID_write,    ID_flush;
13
14
15     initial begin
16         forward1 = 0; forward2 = 0; PCwrite = 1; IFID_write = 1; ID_flush =
17 0;
18     end
19
20
21     always @(`negedge` clk) begin
22         forward1 = 0; forward2 = 0; PCwrite = 1; IFID_write = 1; ID_flush =
23 0;
24         if (INDEX_regWrite && ID_branch) begin
25             if (EX_regDst == ID_regRs || EX_regDst == ID_regRt) begin
26                 PCwrite = 0; IFID_write = 0; ID_flush = 1;
27             end
28         end
29         if (EXMEM_regWrite && !EXMEM_memRead && ID_branch) begin
30             if (EXMEM_regRd == ID_regRs) forward1 = 1;
31             if (EXMEM_regRd == ID_regRt) forward2 = 1;
32         end
33         if (EXMEM_memRead && ID_branch) begin
34             if (EXMEM_regRd == ID_regRs || EXMEM_regRd == ID_regRt) begin
35                 PCwrite = 0; IFID_write = 0; ID_flush = 1;
36             end
37         end
38     end
39
40     endmodule //brc_hazard

```

clk_divider.v

```

1 `timescale 1ns / 1ps
2
3 module clk_divider(gate, clk, reset);
4     input clk, reset;
5     output gate;
6     reg [19:0] Q;
7
8     assign gate = (&Q[5:0]) & (&Q[10:7]) & Q[13] & Q[16] & Q[19];
9
10    always @ (posedge reset or posedge clk) begin
11        if (reset == 1'b1) Q <= 0;
12        else if (gate == 1'b1) Q <= 0;
13        else Q <= Q + 1;
14    end
15 endmodule

```

comparator.v

```

1 `timescale 1ns / 1ps
2
3 module comparator (equal, a, b);

```

```

4   parameter word = 32;
5   input   [word - 1:0] a,      b;
6   output          equal;
7
8   reg           equal;
9
10  initial begin
11    equal = 1'b0;
12  end
13  always @(a or b) begin
14    if (a == b) equal = 1'b1;
15    else        equal = 1'b0;
16  end
17
18 endmodule //comparator

```

control.v

```

1 `timescale 1ns / 1ps
2
3 module control(RegDst, Jump, Branch, MemRead, MemtoReg, ALUop, MemWrite,
4   ALUSrc, RegWrite, Beq, op);
5   input  [5:0]  op;
6   output     RegDst, Jump, Branch, MemRead, MemtoReg, MemWrite,
7   ALUSrc, RegWrite, Beq;
8   output  [1:0]  ALUop;
9
10  reg           RegDst, Jump, Branch, MemRead, MemtoReg, MemWrite,
11  ALUSrc, RegWrite, Beq;
12  reg           [1:0]  ALUop;
13
14  initial begin
15    ALUop = 2'b00; RegDst = 0; Jump = 0; Branch = 0; MemRead = 0;
16    MemtoReg = 0; MemWrite = 0; ALUSrc = 0; RegWrite = 0; Beq = 0;
17  end
18
19  always @(op) begin
20    case (op)
21      // lw
22      6'b100011: begin
23        ALUop = 2'b00; RegDst = 0; Jump = 0; Branch = 0; MemRead =
24        1; MemtoReg = 1; MemWrite = 0; ALUSrc = 1; RegWrite = 1; Beq = 0;
25      end
26
27      // sw
28      6'b101011: begin
29        ALUop = 2'b00; RegDst = 0; Jump = 0; Branch = 0; MemRead =
30        0; MemtoReg = 0; MemWrite = 1; ALUSrc = 1; RegWrite = 0; Beq = 0;
31      end
32
33      // R-type
34      6'b000000: begin
35        ALUop = 2'b10; RegDst = 1; Jump = 0; Branch = 0; MemRead =
36        0; MemtoReg = 0; MemWrite = 0; ALUSrc = 0; RegWrite = 1; Beq = 0;
37      end
38
39      // addi

```

```

33         6'b001000: begin
34             ALUop = 2'b00; RegDst = 0; Jump = 0; Branch = 0; MemRead =
35             0; MemtoReg = 0; MemWrite = 0; ALUSrc = 1; RegWrite = 1; Beq = 0;
36             end
37
38             // andi
39             6'b001100: begin
40                 ALUop = 2'b11; RegDst = 0; Jump = 0; Branch = 0; MemRead =
41                 0; MemtoReg = 0; MemWrite = 0; ALUSrc = 1; RegWrite = 1; Beq = 0;
42                 end
43
44             // beq
45             6'b000100: begin
46                 ALUop = 2'b01; RegDst = 0; Jump = 0; Branch = 1; MemRead =
47                 0; MemtoReg = 0; MemWrite = 0; ALUSrc = 0; RegWrite = 0; Beq = 1;
48                 end
49
50             // bne
51             6'b000101: begin
52                 ALUop = 2'b01; RegDst = 0; Jump = 0; Branch = 1; MemRead =
53                 0; MemtoReg = 0; MemWrite = 0; ALUSrc = 0; RegWrite = 0; Beq = 0;
54                 end
55
56             // j
57             6'b000010: begin
58                 ALUop = 2'b00; RegDst = 0; Jump = 1; Branch = 0; MemRead =
59                 0; MemtoReg = 0; MemWrite = 0; ALUSrc = 0; RegWrite = 0; Beq = 0;
60                 end
61             endcase
62         end
63     endmodule

```

data_mem.v

```

1 `timescale 1ns / 1ps
2
3 module data_mem(read_data, read_addr, write_data, mem_write, mem_read, clk,
4 reset);
5     parameter word    = 32;
6     parameter byte    = 8;
7     parameter number = 1000;
8
9     input  [word - 1:0] read_addr, write_data;
10    input      mem_write, mem_read,   clk,      reset;
11    output [word - 1:0] read_data;
12
13    reg      [byte - 1:0] memory[number - 1:0];
14    reg      [word - 1:0] read_data;
15
16    integer          n;
17
18    always @(posedge clk or posedge reset) begin

```

```

18     if (reset == 1'b1) begin
19         for (n = 0; n < number; n = n + 1) begin
20             memory[n] = 0;
21         end
22     end
23     else if (mem_write) begin
24         memory[read_addr] = write_data[word - 1:word - byte];
25         memory[read_addr + 1] = write_data[word - byte - 1:word -
26 *byte];
26         memory[read_addr + 2] = write_data[word - 2*byte - 1:word -
27 *byte];
27         memory[read_addr + 3] = write_data[word - 3*byte - 1:0];
28     end
29 end
30
31 always @(*) begin
32     read_data = 'bz;
33     if (mem_read) begin
34         read_data = {memory[read_addr], memory[read_addr+1],
35         memory[read_addr+2], memory[read_addr+3]};
36     end
37 end
endmodule

```

demo.v

```

1 `timescale 1ns / 1ps
2
3 module demo(anode, cathode, reset, ctrl, reg_num, pc_or_not, clk);
4     input      reset,      ctrl,      pc_or_not,      clk;
5     input [4:0] reg_num;
6     output [3:0] anode;
7     output [6:0] cathode;
8
9     wire      div_clk;
10    clk_divider cd(div_clk, clk, reset);
11
12    ring_counter rc(anode, div_clk, reset);
13
14    wire [15:0] data;
15    pipeline   pipe(data, ctrl, reset, reg_num, pc_or_not);
16
17    wire [3:0] digit;
18    tri_buffer tb0(digit, data[15:12], anode[0]);
19    tri_buffer tb1(digit, data[11:8], anode[1]);
20    tri_buffer tb2(digit, data[7:4], anode[2]);
21    tri_buffer tb3(digit, data[3:0], anode[3]);
22
23    ssd_driver sd(cathode, digit);
24 endmodule

```

instr_mem.v

```

1 `timescale 1ns / 1ps
2
3 module instr_mem(instruction, read_addr);
4     parameter word = 32;

```



```

41          { mem[88], mem[89], mem[90], mem[91] } =
32'b00100000000010010000000000000001; // 23
42          { mem[92], mem[93], mem[94], mem[95] } =
32'b000010000000000000000000000000011111; // 24
43          { mem[96], mem[97], mem[98], mem[99] } =
32'b001000000000100000000000000000010; // 25
44          { mem[100], mem[101], mem[102], mem[103] } =
32'b001000000000100100000000000000010; // 26
45          { mem[104], mem[105], mem[106], mem[107] } =
32'b000010000000000000000000000000011111; // 27
46          { mem[108], mem[109], mem[110], mem[111] } =
32'b001000000000100000000000000000011; // 28
47          { mem[112], mem[113], mem[114], mem[115] } =
32'b001000000000100100000000000000011; // 29
48          { mem[116], mem[117], mem[118], mem[119] } =
32'b000010000000000000000000000000011111; // 30
49          // $readmemb("D:/JI/2020 fall/VE370 Intro to Computer
Organization/Projects/P2/InstructionMem_for_P2_Demo_bonus.txt", mem);
50      end
51
52      always @(read_addr) begin
53          instruction = {mem[read_addr], mem[read_addr+1], mem[read_addr+2],
mem[read_addr+3]};
54      end
55  endmodule

```

lw_hazard.v

```

1 `timescale 1ns / 1ps
2
3 module lw_hazard (PCWrite, IFID_write, ID_flush, INDEX_memRead, IFID_regRs,
IFID_regRt, INDEX_regRt, clk);
4     input [4:0] IFID_regRs, IFID_regRt, INDEX_regRt;
5     input INDEX_memRead, clk;
6     output PCWrite, IFID_write, ID_flush;
7
8     reg PCWrite, IFID_write, ID_flush;
9
10    initial begin
11        PCwrite      = 1;
12        ID_flush    = 0;
13        IFID_write  = 1;
14    end
15
16    always @ (negedge clk) begin
17        PCwrite      = 1;
18        ID_flush    = 0;
19        IFID_write  = 1;
20        if (INDEX_memRead && (IFID_regRs == INDEX_regRt || IFID_regRt ==
INDEX_regRt)) begin
21            PCWrite      = 0;
22            ID_flush    = 1;
23            IFID_write  = 0;
24        end
25    end
26 endmodule //lw_hazard

```

mux_by_2.v

```
1 `timescale 1ns / 1ps
2
3 module mux_by_2(F, sel, A, B);
4     parameter N = 32;
5
6     input          sel;
7     input [N-1 : 0] A, B;
8     output [N-1 : 0] F;
9
10    reg      [N-1 : 0] F;
11
12    initial begin
13        F = 0;
14    end
15
16    always @ (A, B, sel) begin
17        case (sel)
18            1'b0 : F = A;
19            1'b1 : F = B;
20            default : F = 0;
21        endcase
22    end
23 endmodule
```

mux_by_3.v

```
1 `timescale 1ns / 1ps
2
3 module mux_by_3(F, sel, A, B, C);
4     parameter N = 32;
5
6     input [1:0]      sel;
7     input [N-1 : 0] A, B, C;
8     output [N-1 : 0] F;
9
10    reg      [N-1 : 0] F;
11
12    initial begin
13        F = 0;
14    end
15
16    always @ (A, B, sel) begin
17        case (sel)
18            2'b00 : F = A;
19            2'b01 : F = B;
20            2'b10 : F = C;
21            2'b11 : F = C;
22            default : F = 0;
23        endcase
24    end
25 endmodule
```

pipeline.v

```

1 `timescale 1ns / 1ps
2
3 module pipeline (data, clk, reset, reg_num, pc_or_not);
4     parameter word = 32;
5     parameter half_w = 16;
6
7     input                      clk,
8     input [4:0]                 reg_num;
9     output [half_w - 1:0]       data;
10
11    // output
12    wire   [word - 1:0]         full_data,      tmp_reg,      curr;
13
14    assign   full_data = pc_or_not?curr:tmp_reg;
15    assign   data = full_data[half_w - 1:0];
16
17    // IF stage
18    wire   [word - 1:0]         next,          IF_pcplus4,
19                  IF_instr,        jump_target;
20    wire   [word - 1:0]         brc_target,     not_jump_target;
21    wire   PCWrite,             ID_real_brc,
22                  ID_jump,        IF_flush;
23
24    PC      pc(curr, clk, next, PCWrite, reset);
25    adder   pc_adder(IF_pcplus4, curr, 4);
26    mux_by_2 brc_mux(not_jump_target, ID_real_brc, IF_pcplus4,
27                      brc_target);
28    mux_by_2 jmp_mux(next, ID_jump, not_jump_target, jump_target);
29    instr_mem instrmem(IF_instr, curr);
30    assign   IF_flush = ID_real_brc | ID_jump;
31
32    // IF/ID stage register
33    wire   [word - 1:0]         IFID_pcplus4,   IFID_instr;
34    wire   IFID_write;
35
36    IF_ID      ifid(IFID_pcplus4, IFID_instr, IF_pcplus4, IF_instr,
37                      IF_flush, IFID_write, clk);
38
39    // ID stage
40    wire   ID_regDs,           ID_branch,
41    ID_memRead,                ID_beq;
42    wire   ID_memtoReg,        ID_memWrite,    ID_ALUSrc,
43    ID_regWrite,               ID_brc_part;
44    wire   MEMWB_memtoReg,     MEMWB_regwrite, reg_equal,
45    forward1,                 forward2;
46    wire   [1:0]                ID_ALUop;
47    wire   [4:0]                MEMWB_regRd;
48    wire   [word - 1:0]         ID_read_data1,  ID_read_data2,
49    ID_reg_data1,              ID_reg_data2,  ext_imt;
50    wire   [word - 1:0]         sh_ext_imt;
51    wire   [word - 1:0]         EXMEM_ALU_result,
52    EXMEM_val_regRt,           WB_write_data;
53    wire   [27:0]               jump_part;
54
55    control   ctrl(ID_regDst, ID_jump, ID_branch, ID_memRead,
56                  ID_memtoReg, ID_ALUop, ID_memWrite, ID_ALUSrc, ID_regWrite, ID_beq,
57                  IFID_instr[31:26]);

```

```

47    reg_file    regfile(tmp_reg, ID_read_data1, ID_read_data2,
48                      IFID_instr[25:21], IFID_instr[20:16], MEMWB_regRd, WB_write_data,
49                      MEMWB_regWrite, clk, reset, reg_num);
50    mux_by_2    read_data_mux1(ID_reg_data1, forward1, ID_read_data1,
51                               EXMEM_ALU_result);
52    mux_by_2    read_data_mux2(ID_reg_data2, forward2, ID_read_data2,
53                               EXMEM_ALU_result);
54    comparator  comp(reg_equal, ID_reg_data1, ID_reg_data2);
55    sign_ext   extended(ext_imt, IFID_instr[15:0]);
56    shifter_1  sh1(jump_part, IFID_instr[25:0]);
57    assign      jump_target = {IFID_pcplus4[31:28], jump_part};
58    shifter_2  sh2(sh_ext_imt, ext_imt);
59    adder      brc_addr_adder(brc_target, sh_ext_imt, IFID_pcplus4);
60    mux_by_2 #1 beqOrbne(ID_brc_part, ID_beq, ~reg_equal, reg_equal);
61    assign      ID_real_brc = ID_brc_part & ID_branch;
62
63    // ID/EX stage register
64    wire [word - 1:0]           IDEX_ext_imt, IDEX_reg_data1,
65    IDEX_reg_data2;
66    wire [4:0]                  IDEX_regRs, IDEX_regRt,
67    IDEX_regRd;
68    wire                      IDEX_regDst, IDEX_memRead,
69    IDEX_memtoReg, IDEX_memWrite;
70    wire                      IDEX_ALUSrc, IDEX_regWrite, ID_flush;
71    wire [1:0]                 IDEX_ALUOp;
72
73    ID_EX      idex(IDEIndex_regDst, IDEX_memRead, IDEX_memtoReg, IDEX_ALUOp,
74                      IDEX_memWrite, IDEX_ALUSrc,
75                      IDEX_ext_imt, IDEX_regRs, IDEX_regRt, IDEX_regRd,
76                      clk, ID_flush, ID_regDst, ID_memRead, ID_memtoReg,
77                      ID_ALUop, ID_memWrite, ID_ALUSrc,
78                      IDEX_regWrite, IDEX_reg_data1, IDEX_reg_data2, ext_imt,
79                      IFID_instr[25:21], IFID_instr[20:16], IFID_instr[15:11]);
80
81    // EX stage
82    wire [word - 1:0]           ALU_operand1, ALU_operand2,
83    ALU_operand2_part;
84    wire [4:0]                  EX_regDst;
85    wire [word - 1:0]           EX_ALU_result;
86    wire [3:0]                  ALU_control;
87    wire [1:0]                  forwardA, forwardB;
88
89    mux_by_2 #5 regDst_mux(EX_regDst, IDEX_regDst, IDEX_regRt,
90                           IDEX_regRd);
91    mux_by_3    ALU_mux1(ALU_operand1, forwardA, IDEX_reg_data1,
92                           WB_write_data, EXMEM_ALU_result);
93    mux_by_3    ALU_mux2(ALU_operand2_part, forwardB, IDEX_reg_data2,
94                           WB_write_data, EXMEM_ALU_result);
95    mux_by_2    ALU_mux3(ALU_operand2, IDEX_ALUSrc, ALU_operand2_part,
96                           IDEX_ext_imt);
97    ALUcontrol  alu_ctrl(ALU_control, IDEX_ext_imt[5:0], IDEX_ALUOp);
98    ALU        alu_com(EX_ALU_result, ALU_operand1, ALU_operand2,
99                         ALU_control);
100
101   // EX/MEM stage register
102   wire [4:0]                 EXMEM_regRd;

```

```

87      wire EXMEM_memRead, EXMEM_memtoReg,
88      EXMEM_memWrite, EXMEM_regWrite;
89
89      EX_MEMORY exmem(EXMEM_memRead, EXMEM_memtoReg, EXMEM_memWrite,
90      EXMEM_regWrite, EXMEM_ALU_result, EXMEM_val_regRt, EXMEM_regRd,
90      clk, IDEX_memRead, IDEX_memtoReg, IDEX_memWrite,
91      IDEX_regWrite, EX_ALU_result, ALU_operand2_part, EX_regDst);
91
92      // MEM stage
93      wire [word - 1:0] MEM_read_mem;
94
95      data_mem datamem(MEM_read_mem, EXMEM_ALU_result, EXMEM_val_regRt,
95      EXMEM_memWrite, EXMEM_memRead, clk, reset);
96
97      // MEM/WB stage register
98      wire [word - 1:0] MEMWB_ALU_result, MEMWB_read_mem;
99
100     MEM_WB memwb(MEMWB_memtoReg, MEMWB_regWrite, MEMWB_read_mem,
100     MEMWB_ALU_result, MEMWB_regRd,
101     clk, EXMEM_regWrite, EXMEM_memtoReg, MEM_read_mem,
101     EXMEM_ALU_result, EXMEM_regRd);
102
103      // WB stage
104      mux_by_2 mem2reg(WB_write_data, MEMWB_memtoReg, MEMWB_ALU_result,
104      MEMWB_read_mem);
105
106      // branch hazard detection
107      wire PCwrite1, IFID_write1;
108      wire ID_flush1;
109
110      brc_hazard brc_hazard_detect(forward1, forward2, PCwrite1,
110      IFID_write1, ID_flush1,
111      ID_branch, IDEX_regWrite, EXMEM_regWrite, EXMEM_memRead,
111      EX_regDst, EXMEM_regRd, IFID_instr[20:16], IFID_instr[25:21], clk);
112
113      // lw hazard detection
114      wire PCwrite2, IFID_write2, ID_flush2;
115
116      lw_hazard lw_hazard_detect(PCwrite2, IFID_write2, ID_flush2,
116      IDEX_memRead, IFID_instr[25:21], IFID_instr[20:16], IDEX_regRt, clk);
117
118      assign PCwrite = PCwrite1 & PCwrite2;
119      assign IFID_write = IFID_write1 & IFID_write2;
120      assign ID_flush = ID_flush1 | ID_flush2;
121
122      // R-type forwarding
123      R_forward forwarding(forwardA, forwardB, EXMEM_regWrite,
123      MEMWB_regWrite, EXMEM_regRd, MEMWB_regRd, IDEX_regRs, IDEX_regRt, clk);
124  endmodule //pipeline

```

ports.xdc

```

1 set_property IOSTANDARD LVCMOS33 [get_ports {anode[3]}]
2 set_property IOSTANDARD LVCMOS33 [get_ports {anode[2]}]
3 set_property IOSTANDARD LVCMOS33 [get_ports {anode[1]}]
4 set_property IOSTANDARD LVCMOS33 [get_ports {anode[0]}]
5 set_property IOSTANDARD LVCMOS33 [get_ports {cathode[6]}]

```

```

6 set_property IOSTANDARD LVC MOS33 [get_ports {cathode[5]}]
7 set_property IOSTANDARD LVC MOS33 [get_ports {cathode[4]}]
8 set_property IOSTANDARD LVC MOS33 [get_ports {cathode[3]}]
9 set_property IOSTANDARD LVC MOS33 [get_ports {cathode[2]}]
10 set_property IOSTANDARD LVC MOS33 [get_ports {cathode[1]}]
11 set_property IOSTANDARD LVC MOS33 [get_ports {cathode[0]}]
12 set_property IOSTANDARD LVC MOS33 [get_ports {reg_num[4]}]
13 set_property IOSTANDARD LVC MOS33 [get_ports {reg_num[3]}]
14 set_property IOSTANDARD LVC MOS33 [get_ports {reg_num[2]}]
15 set_property IOSTANDARD LVC MOS33 [get_ports {reg_num[1]}]
16 set_property IOSTANDARD LVC MOS33 [get_ports {reg_num[0]}]
17 set_property PACKAGE_PIN U4 [get_ports {anode[2]}]
18 set_property PACKAGE_PIN V4 [get_ports {anode[1]}]
19 set_property PACKAGE_PIN W4 [get_ports {anode[0]}]
20 set_property PACKAGE_PIN U2 [get_ports {anode[3]}]
21 set_property PACKAGE_PIN W7 [get_ports {cathode[6]}]
22 set_property PACKAGE_PIN W6 [get_ports {cathode[5]}]
23 set_property PACKAGE_PIN U8 [get_ports {cathode[4]}]
24 set_property PACKAGE_PIN V8 [get_ports {cathode[3]}]
25 set_property PACKAGE_PIN U5 [get_ports {cathode[2]}]
26 set_property PACKAGE_PIN V5 [get_ports {cathode[1]}]
27 set_property PACKAGE_PIN U7 [get_ports {cathode[0]}]
28 set_property PACKAGE_PIN W15 [get_ports {reg_num[4]}]
29 set_property PACKAGE_PIN W17 [get_ports {reg_num[3]}]
30 set_property PACKAGE_PIN W16 [get_ports {reg_num[2]}]
31 set_property PACKAGE_PIN V17 [get_ports {reg_num[0]}]
32 set_property PACKAGE_PIN V16 [get_ports {reg_num[1]}]
33 set_property IOSTANDARD LVC MOS33 [get_ports clk]
34 set_property IOSTANDARD LVC MOS33 [get_ports pc_or_not]
35 set_property IOSTANDARD LVC MOS33 [get_ports ctrl]
36 set_property IOSTANDARD LVC MOS33 [get_ports reset]
37 set_property PACKAGE_PIN W5 [get_ports clk]
38 set_property PACKAGE_PIN V15 [get_ports pc_or_not]
39 set_property PACKAGE_PIN W14 [get_ports ctrl]
40 set_property PACKAGE_PIN U18 [get_ports reset]
41 set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets ctrl_IBUF]

```

reg_file.v

```

1 `timescale 1ns / 1ps
2
3 module reg_file(reg_tmp, read_data1, read_data2, read_addr1, read_addr2,
4   write_addr, write_data, regwrite, clk, reset, reg_num);
5   parameter addr_size = 5;
6   parameter word = 32;
7
8   input  [addr_size - 1:0] read_addr1, read_addr2, write_addr,
9   reg_num;
10  input  [word - 1:0]      write_data;
11  input                regwrite,    clk,        reset;
12  output [word - 1:0]      read_data1,  read_data2,  reg_tmp;
13  integer               n;
14
15  always @(negedge clk or posedge reset) begin
16    if (reset == 1'b1) begin

```

```

17         for (n = 0; n < 2**addr_size; n = n + 1) begin
18             reg_mem[n] = 0;
19         end
20     end
21     else if (regwrite == 1'b1) begin
22         reg_mem[write_addr] = write_data;
23     end
24 end
25
26 assign read_data1 = reg_mem[read_addr1];
27 assign read_data2 = reg_mem[read_addr2];
28 assign reg_tmp    = reg_mem[reg_num];
29 endmodule

```

ring_counter.v

```

1 `timescale 1ns / 1ps
2
3 module ring_counter(Q, clk, reset);
4     input clk, reset;
5     output [3:0] Q;
6
7     wire CE, load;
8     assign CE = 1'b1;
9     assign load = 1'b0;
10    wire [1:0] D;
11
12    counter_4_bit #2 bit_2(D, clk, reset, load, CE);
13
14    decoder_2_to_4 deco(D, Q);
15 endmodule
16
17
18 module counter_4_bit(Q, clk, reset, load, CE);
19     parameter N = 4;
20     input clk, reset, load, CE;
21     output [N - 1:0] Q;
22
23     reg [N - 1:0] Q;
24
25     always @ (posedge clk or posedge reset) begin
26         if (reset == 1'b1) Q <= 0;
27         else if (load == 1'b1) Q <= 0;
28         else if (CE == 1'b1) Q <= Q + 1;
29         else Q <= Q;
30     end
31 endmodule
32
33 module decoder_2_to_4(I, D);
34     input [1:0] I;
35     output [3:0] D;
36
37     reg [3:0] D;
38     always @ (I) begin
39         case(I)
40             2'b00: D <= 4'b1110;
41             2'b01: D <= 4'b1101;

```

```
42      2'b10: D <= 4'b1011;
43      2'b11: D <= 4'b0111;
44      default: D <= 4'b1111;
45      endcase
46  end
47 endmodule
```

shifter_1.v

```
1 `timescale 1ns / 1ps
2
3 module shifter_1(out, in);
4     parameter in_size = 26;
5     parameter by = 2;
6
7     input [in_size - 1:0] in;
8     output [in_size - 1 + by:0] out;
9
10    assign out[in_size - 1+by:by] = in;
11    assign out[by - 1:0] = 0;
12 endmodule
```

shifter_2.v

```
1 `timescale 1ns / 1ps
2
3 module shifter_2(out, in);
4     parameter word = 32;
5     parameter by = 2;
6
7     input [word - 1:0] in;
8     output [word - 1:0] out;
9
10    assign out[word - 1:by] = in[word - 1 - by:0];
11    assign out[by - 1:0] = 0;
12 endmodule
```

sign_ext.v

```
1 `timescale 1ns / 1ps
2
3 module sign_ext(out, in);
4     parameter word = 32;
5     parameter half_word = 16;
6
7     input [half_word - 1:0] in;
8     output [word - 1:0] out;
9
10    assign out = {{half_word{in[half_word-1]}}, in};
11 endmodule
```

ssd_driver.v

```
1 `timescale 1ns / 1ps
2
```

```

3 module ssd_driver(cathode, Q);
4   input [3:0] Q;
5   output [6:0] cathode;
6
7   reg [6:0] cathode;
8
9   always @ (Q) begin
10    case(Q)
11      4'b0000: cathode <= 7'b0000001;
12      4'b0001: cathode <= 7'b1001111;
13      4'b0010: cathode <= 7'b0010010;
14      4'b0011: cathode <= 7'b0000110;
15      4'b0100: cathode <= 7'b1001100;
16      4'b0101: cathode <= 7'b0100100;
17      4'b0110: cathode <= 7'b0100000;
18      4'b0111: cathode <= 7'b0001111;
19      4'b1000: cathode <= 7'b0000000;
20      4'b1001: cathode <= 7'b0000100;
21      4'b1010: cathode <= 7'b0001000;
22      4'b1011: cathode <= 7'b1100000;
23      4'b1100: cathode <= 7'b1110010;
24      4'b1101: cathode <= 7'b1000010;
25      4'b1110: cathode <= 7'b0110000;
26      4'b1111: cathode <= 7'b0111000;
27      default cathode <= 7'b1111111;
28    endcase
29  end
30 endmodule

```

tri_buffer.v

```

1 `timescale 1ns / 1ps
2
3 module tri_buffer(D, I ,sel);
4   parameter N = 4;
5   input [N - 1:0] I;
6   input sel;
7   output [N - 1:0] D;
8
9   assign D = sel ? 'bz: I;
10 endmodule

```

Sample Instructions

InstructionMem_for_P2_Demo_bonus.txt

```

1 00100000 00001000 00000000 00100000 //addi $t0, $zero, 0x20
2 00100000 00001001 00000000 00110111 //addi $t1, $zero, 0x37
3 00000001 00001001 10000000 00100100 //and $s0, $t0, $t1
4 00000001 00001001 10000000 00100101 //or $s0, $t0, $t1
5 10101100 00010000 00000000 00000100 //sw $s0, 4($zero)
6 10101100 00001000 00000000 00001000 //sw $t0, 8($zero)
7 00000001 00001001 10001000 00100000 //add $s1, $t0, $t1
8 00000001 00001001 10010000 00100010 //sub $s2, $t0, $t1
9 00010010 00110010 00000000 00001001 //beq $s1, $s2, error0
10 10001100 00010001 00000000 00000100 //lw $s1, 4($zero)
11 00110010 00110010 00000000 01001000 //andi $s2, $s1, 0x48

```

```
12 00010010 00110010 00000000 00001001 //beq $s1, $s2, error1
13 10001100 00010011 00000000 00001000 //lw $s3, 8($zero)
14 00010010 00010011 00000000 00001010 //beq $s0, $s3, error2
15 00000010 01010001 10100000 00101010 //slt $s4, $s2, $s1 (Last)
16 00010010 10000000 00000000 00001111 //beq $s4, $0, EXIT
17 00000010 00100000 10010000 00100000 //add $s2, $s1, $0
18 00001000 00000000 00000000 00001110 //j Last
19 00100000 00001000 00000000 00000000 //addi $t0, $0, 0(error0)
20 00100000 00001001 00000000 00000000 //addi $t1, $0, 0
21 00001000 00000000 00000000 00011111 //j EXIT
22 00100000 00001000 00000000 00000001 //addi $t0, $0, 1(error1)
23 00100000 00001001 00000000 00000001 //addi $t1, $0, 1
24 00001000 00000000 00000000 00011111 //j EXIT
25 00100000 00001000 00000000 00000010 //addi $t0, $0, 2(error2)
26 00100000 00001001 00000000 00000010 //addi $t1, $0, 2
27 00001000 00000000 00000000 00011111 //j EXIT
28 00100000 00001000 00000000 00000011 //addi $t0, $0, 3(error3)
29 00100000 00001001 00000000 00000011 //addi $t1, $0, 3
30 00001000 00000000 00000000 00011111 //j EXIT
```