Lab 13: Pipelining with Branching or Forwarding

Jiasen Zhou and Jon Johnston

May 5, 2019

1 Executive Summary

The purpose of this lab was to add the register buffers between each stage and ultimately to get a barebones pipelined datapath working. The iFetch, iDecode, iExecute, iMemory and iWriteback stages were all connected together with buffers placed at the beginning of each module to ensure data would be passed at the correct time. This pipeline was tested with the 10 test instructions, excluding NOPs added between some instructions. 4 NOPs were added after 6 of the instructions since the datapath did not have data forwarding or hazard detection, so the datapath must wait until the writeback stage to continue operation. After comparing the results of the simulation against the expected results, this lab was successful.

2 Test Report

To verify operation of the pipeline, this lab requires 1 test bench.

1. Pipeline simulation

2.1 Pipeline Test Bench

The Pipeline Test Bench contains:

1. Inputs
   (a) branch_target - branch address
   (b) pc_src - the control of branch mux
   (c) reset - set the current pc to be 0

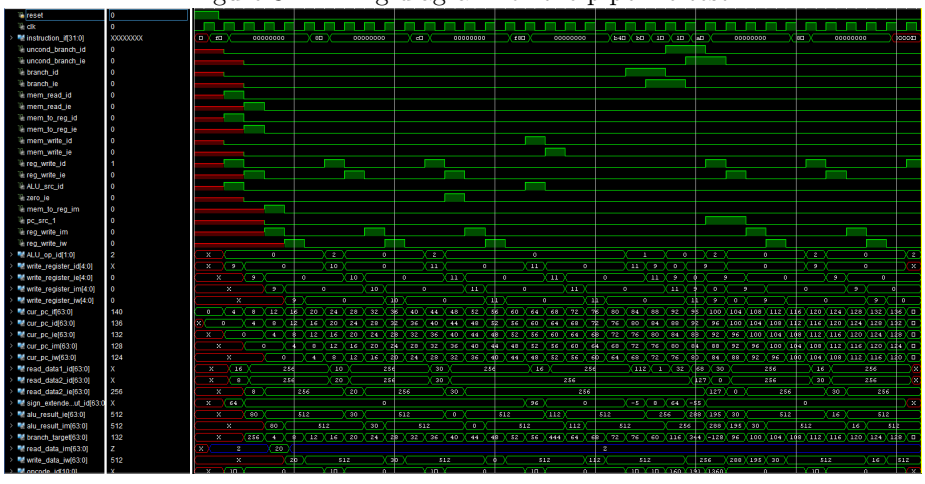
Figure 1: Pipeline buffer table.

Fetch				Decode				Execute				Memory				Write Back			
Input	Src	Output	Dst	Input	Src	Output	Dst	Input	Src	Output	Dst	Input	Src	Output	Dst	Input	Src	Output	Dst
pc_src	iM													pc_src	iF				
branch_target	iE									branch_target	iF								
		cur_pc_if	iD	cur_pc_if	iF	cur_pc_id	iE	cur_pc_id	iD	cur_pc_ie	iM	cur_pc_ie	iE	cur_pc_im	iW				
		instruction_if	iD	instruction_if	iF														
				write_data_iw	iW													write_data_iw	iD
				write_register_iw	iW	write_register_id	iW	write_register_id		write_register_ie		write_register_ie		write_register_im		write_register_im	iD	write_data_iw	iD
				reg_write_iw	iW	reg_write_id	iW	reg_write_id		reg_write_ie		reg_write_ie		reg_write_im		reg_write_im	iD	reg_write_iw	iD
						sign_extended_output_id	iE	sign_extended_output_id	iD										
						uncondbranch_id	iM	uncondbranch_id		uncondbranch_ie		uncondbranch_ie	iD						
						branch_id	iM	branch_id		branch_ie		branch_ie	iD						
						mem_read_id	iM	mem_read_id		mem_read_ie		mem_read_ie	iD						
						mem_to_reg_id	iW	mem_to_reg_id	iD	mem_to_reg_ie	iW	mem_to_reg_ie	iD	mem_to_reg_im	iW	mem_to_reg_im	iD		
						alu_op_id	iE	alu_op_id	iD										
						mem_write_id	iM	mem_write_id		mem_write_ie		mem_write_ie	iD						
						alu_src_id	iD	alu_src_id	iD										
						read_data1_id	iE	read_data1_id	iD										
						read_data2_id	iE/iM	read_data2_id	iD	read_data2_ie		read_data2_ie	iD						
						opcode_id	iE	opcode_id	iD										
										alu_result_ie	iM/iW	alu_result_ie	iE	alu_result_im		alu_result_im	iE		
										zero_ie	iM	zero_ie	iE						
														read_data_im	iW	read_data_im	iM		

Figure 2: Expected Results of the pipeline test.

	Instruction 1	Instruction 2	Instruction 3	Instruction 4	Instruction 5	Instruction 6	Instruction 7	Instruction 8	Instruction 9	Instruction 10
Instruction	LDUR X9, [X22, #64]	ADD X10, X19, X9	SUB X11, X20, X10	STUR X11, [X22, #96]	CBZ X11, -5	CBZ X9, 8	B 64	B -55	ORR X9, X10, X21	AND X9, X22, X10
Machine Instruction (hex)	F84402C9	8B09026A	CB0A028B	F80602CB	B4FFFF6B	b4000109	14000040	17FFFFC9	AA150149	8A0A02C9
opcode (binary)	11111000010	10001011000	11001011000	11111000000	10110100	10110100	000101	000101	10101010000	10001010000
sign_extended_output (hex)	0000000000000040	0000000000000000	0000000000000000	0000000000000060	FFFFFFFFFFFFFFFB	0000000000000008	0000000000000040	FFFFFFFFFFFFFFC9	0000000000000000	0000000000000000
reg2_loc	0	0	0	1	1	1	0	0	0	0
branch	0	0	0	0	1	1	0	0	0	0
mem_read	1	0	0	0	0	0	0	0	0	0
mem_to_reg	1	0	0	0	0	0	0	0	0	0
alu_op	00	10	10	00	01	01	00	00	10	10
mem_write	0	0	0	1	0	0	0	0	0	0
alu_src	1	0	0	1	0	0	0	0	0	0
reg_write	1	1	1	0	0	0	0	0	1	1
uncondbranch	0	0	0	0	0	0	1	1	0	0
write_data (decimal)	20	30	0	X	X	X	X	X	30	16
read_data1 (decimal)	16	10	30	16	X	X	X	X	30	16
read_data2 (decimal)	X	20	30	0	0	20	X	X	0	30
branch_target	X	X	X	X	60	116	344	-128	32	36
alu_result	80	30	0	112	0	20	X	X	30	16
zero	0	0	1	0	1	0	0	0	0	0
PC actual	0	20	40	60	80	84	88	92	96	116

Figure 3: Timing diagram for the pipeline test.



3 Code Appendix

Listing 1: Verilog code of pipeline.

```
`include "definitions.vh"

module pipeline;

    reg reset, pc_src;
    wire [`INSTR_LEN-1:0] instruction_if;
    wire uncond_branch_id, uncond_branch_ie,
         branch_id, branch_ie,
         mem_read_id, mem_read_ie,
         mem_to_reg_id, mem_to_reg_ie,
         mem_write_id, mem_write_ie,
         reg_write_id, reg_write_ie,
         ALU_src_id, clk,
         zero_ie, mem_to_reg_im, pc_src_1,
         reg_write_im, reg_write_iw;
    wire [1:0] ALU_op_id;
    wire [4:0] write_register_id, write_register_ie,
         write_register_im, write_register_iw;
    wire [`WORD-1:0]
         cur_pc_if, cur_pc_id, cur_pc_ie, cur_pc_im, cur_pc_iw,
         read_data1_id,
         read_data2_id, read_data2_ie,
         sign_extended_output_id,
         alu_result_ie, alu_result_im,
         branch_target,
         read_data_im,
         write_data_iw;
    wire [10:0] opcode_id;

    // Base Clock
    oscillator r_clk(.clk(clk));

    // Fetch Stage
    fetch fetch_mod(
        .clk(clk),
        .reset(reset),
        .branch_target(branch_target),
        .pc_src(pc_src),
        .instruction(instruction_if),
        .cur_pc(cur_pc_if));

    // Decode Stage
```

```
    iDecode decode_mod(
        .cur_pc_in(cur_pc_if),
        .cur_pc_out(cur_pc_id),
        .write_data(write_data_iw),
        .write_register_in(write_register_iw),
        .write_register_out(write_register_id),
        .reg_write_in(reg_write_iw),
        .reg_write_out(reg_write_id),
        .instruction(instruction_if),
        .uncond_branch(uncond_branch_id),
        .branch(branch_id),
        .mem_read(mem_read_id),
        .mem_to_reg(mem_to_reg_id),
        .mem_write(mem_write_id),
        .ALU_src(ALU_src_id),
        .write_clk(clk),
        .ALU_op(ALU_op_id),
        .read_data1(read_data1_id),
        .read_data2(read_data2_id),
        .sign_extended_output(sign_extended_output_id),
        .opcode(opcode_id));

    iExecute execute_mod(
        .clk(clk),
        .pc_in(cur_pc_id),
        .pc_out(cur_pc_ie),
        .write_register_in(write_register_id),
        .write_register_out(write_register_ie),
        .reg_write_in(reg_write_id),
        .reg_write_out(reg_write_ie),
        .uncond_branch_in(uncond_branch_id),
        .uncond_branch_out(uncond_branch_ie),
        .branch_in(branch_id),
        .branch_out(branch_ie),
        .mem_read_in(mem_read_id),
        .mem_read_out(mem_read_ie),
        .mem_to_reg_in(mem_to_reg_id),
        .mem_to_reg_out(mem_to_reg_ie),
        .mem_write_in(mem_write_id),
        .mem_write_out(mem_write_ie),
        .read_data1(read_data1_id),
        .read_data2_in(read_data2_id),
        .read_data2_out(read_data2_ie),
        .sign_extend(sign_extended_output_id),
        .opcode(opcode_id),
        .alu_op(ALU_op_id),
```

```
        .alu_src(ALU_src_id),
        .alu_result(alu_result_ie),
        .zero(zero_ie),
        .branch_target(branch_target));


    iMemory memory_mod(
        .im_clk(clk),
        .pc_in(cur_pc_ie),
        .pc_out(cur_pc_im),
        .alu_result_in(alu_result_ie),
        .alu_result_out(alu_result_im),
        .read_data2(read_data2_ie),
        .mem_read(mem_read_ie),
        .mem_write(mem_write_ie),
        .mem_to_reg_in(mem_to_reg_ie),
        .mem_to_reg_out(mem_to_reg_im),
        .write_register_in(write_register_ie),
        .write_register_out(write_register_im),
        .reg_write_in(reg_write_ie),
        .reg_write_out(reg_write_im),
        .zero(zero_ie),
        .branch(branch_ie),
        .uncond_branch(uncond_branch_ie),
        .read_data(read_data_im),
        .pc_src(pc_src_1));

    iWrite_back writeback_m(
        .iw_clk(clk),
        .read_data(read_data_im),
        .alu_result(alu_result_im),
        .MemtoReg(mem_to_reg_im),
        .write_data(write_data_iw),
        .pc_in(cur_pc_im),
        .write_register_in(write_register_im),
        .write_register_out(write_register_iw),
        .reg_write_in(reg_write_im),
        .reg_write_out(reg_write_iw),
        .pc_out(cur_pc_iw));

initial
    begin
       reset = 1;
       pc_src = 0;#5
       reset = 0; #140
   $finish;
```

```
    end

endmodule
```