ECEN 3100

Digital Design Interfacing

Lab Assignment # 9

Implementation of Data Forwarding in 5-Stage Pipeline CPU

Cameron Biniamow

Gopal Bohora

University of Nebraska-Lincoln

Department of Electrical and Computer Engineering

Peter Kiewit Institute

Due: 11/12/2020

Introduction

We have learned about data dependence and data hazards in the lecture and possible solution for data hazard. When an instruction depends on the results of previous instruction by overlapping of instructions in the pipeline the data hazard raises. For example

ADD R2, R3, R6

SUB R5, R2, R4

In the above instruction, the SUB instruction uses the destination register R2 of previous ADD instruction and SUB instruction executes with current value of R2 which is not correct because it is the final addition result which creates a problem called data hazard. To prevent this problem, we implement the data forwarding to our design. In this lab we will simulate and test the data dependence of our design using our previous 5-stage pipeline module in Lab 7 and Lab 8 and we will generate the waveform as well. Finally, implemented the data forwarding to overcome the data hazard problem.

Implementation

Lab 9 operates by using previously written code in Labs 7 & 8. Due to this, mainly the forwarding unit is the only addition to the program. The forwarding unit operates by checking the destination register in both the memory and writeback stages and checks to see if it is the same register as either the source or target register in the execute stage.

Specifically, the forwarding unit initially checks if RegWrite is HIGH in the EX/MEM pipeline and if the destination register in the EX/MEM pipeline is the same as the source register in the ID/EX pipeline. Given that both conditions are met, the forwarding unit outputs 2'b10 through ForwardA. This means that the multiplexer for input A for the ALU is the ALU result from the EX/MEM pipeline. Given the previous conditions were not true, the forwarding unit checks if RegWrite in the MEM/WB pipeline is HIGH and if the destination register in the MEM/WB pipeline matches the source register in the ID/EX pipeline. If these conditions are met, the forwarding unit outputs 2'b01 through ForwardA. This means that input A of the ALU will be the writeback data. If neither case has their conditions met, the forwarding unit outputs 2'b00 through ForwardA, which means that input A of the ALU will be the read data from the register file at the register source address.

For input B of the ALU, the same logic is used to determine which data to use. This is performed by checking if RegWrite of the EX/MEM pipeline is HIGH and if the destination

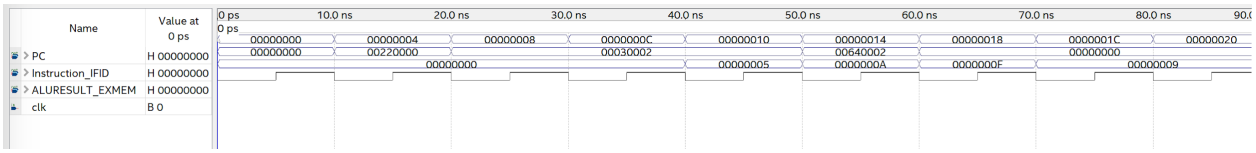
register in the EX/MEM pipeline matches the target register of the ID/EX pipeline. Given these conditions are met, the forwarding unit outputs 2'b10 through ForwardB. This means that the B input of the ALU is the ALU result from the EX/MEM pipeline. If the previous conditions are not met, the forwarding unit checks if RegWrite of the MEM/WB pipeline is HIGH and if the destination register of the MEM/WB pipeline is the same as the target register of the ID/EX pipeline. Given these conditions are met, the forwarding unit outputs 2'b01 through ForwardB. This means that the B input for the ALU is the writeback data. In the case that all previous conditions were not met, the forwarding unit will output 2'b00 through ForwardB. This means that input B of the ALU is the read data from the register file at the address of the target register of the IF/ID pipeline.

Use of the forwarding unit allows for consecutive instructions where the destination register of a previous instruction is used as either the source or target register of the current instruction. After the forwarding unit has been implemented, the remaining portion of the 5-stage pipelined CPU designed in previous labs operates the same.

Results

As can be seen below, the generated waveform is shown. Through analyzing the produced waveform, it has been determined that the data forwarding 5-stage pipelined CPU operates as expected and without error. This determination has been made due to the fact the CPU executes three consecutive instructions where the destination register in the previous instruction is used as either the source or target register in the following instruction. By comparing the results in the waveform to the table below, correct results are verified.

PC	Instruction	Instruction Code	ALU Result
0000 0000H	R0 = R1 & R2	0022 0000H	0000 0000H
0000 0004H	R0 = R0 + R3	0003 0002H	0000 0005H
0000 0008H	R0 = R0 + R3	0003 0002H	0000 000AH
0000 000CH	R0 = R0 + R3	0003 0002H	0000 000FH
0000 0010H	R0 = R3 + R4	0064 0002H	0000 0009H



Conclusion

In conclusion, the main purpose of this lab was to design and simulate the data dependence of our designed sequences of instruction which includes the data dependence and data hazards. The example of data dependence is mentioned above in the instruction section. We came to know that when the data hazard occurs then it prevents the next instruction from executing during its designated clock cycle, so as soon as a data hazard occurs it reduces the performance of the CPU. Hence, to prevent such kinds of problems we implement data forwarding. For this lab we used the same code that we generated in the lab 7 and 8. We generated the correct waveform which determines that the designed 5-stage pipeline CPU is functioning correctly.

Appendix

Figure I: Data Forwarding 5-Stage Pipelined CPU Verilog Source Code

```
// CAMERON BINIAMOW & GOPAL BOHORA
// ECEN 3100
// LAB 9: DATA FORWARDING
// DUE: 11/12/2020



/*============================================================*/
/*============================================================*/
/*=================MAIN=======================================*/
/*============================================================*/
/*============================================================*/

`timescale 1ns / 1ps

module Lab9(
input clk,
output [31:0] PC,
output [31:0] Instruction_IFID,
output [4:0] Rs_IDEX,
output [4:0] Rt_IDEX,
output [4:0] Rd_IDEX,
output [31:0] DATA1_IDEX,
output [31:0] DATA2_IDEX,
output [8:0] Control_IDEX,
output [31:0] ALURESULT_EXMEM,
output [4:0] Rd_EXMEM,
output [8:0] Controls_EXMEM
    );

       reg [31:0] imem [31:0];

       //*********************INSTRUCTIIONS*********************//
       //*******************************************************//

       initial
       begin
              imem[0] = 32'h00220000;                          // R0 = R1 & R2
              imem[1] = 32'h00030002;                          // R0 = R0 + R3
              imem[2] = 32'h00030002;                          // R0 = R0 + R3
              imem[3] = 32'h00030002;                          // R0 = R0 + R3
              imem[4] = 32'h00640002;                          // R0 = R3 - R4
       end

       //*******************************************************//


/*============================================================*/
/*============================================================*/
/*=================INSTRUCTION FETCH (IF)=====================*/
/*============================================================*/
/*============================================================*/

       reg [31:0] PC_reg = 0;
       reg [31:0] instruction_IFID;


       //*********************IF/ID PIPELINE********************//
       //*******************************************************//

       always @(negedge clk)
```

```
	begin
		PC_reg <= PC_reg + 4;					// ADD 4 TO PC
		instruction_IFID <= imem[(PC_reg/4)];	// LOAD INSTRUCTION FROM MEM
	end

	//*********************************************************//


	//*********************OUTPUTS*****************************//
	//*********************************************************//

	assign PC = PC_reg;							// OUTPUT PROGRAM COUNTER
	assign Instruction_IFID = instruction_IFID;	// OUTPUT INSTRUCTION

	//*********************************************************//


/*=============================================================*/
/*=============================================================*/
/*==================INSTRUCTION DECODE (ID)====================*/
/*=============================================================*/
/*=============================================================*/

	reg [31:0] RF [31:0];
	reg [4:0] rs_IDEX, rt_IDEX, rd_IDEX;
	reg [31:0] Data1, Data2, Data1_IDEX, Data2_IDEX, instruction_IDEX;
	reg [8:0] ControlLines_IDEX, ControlLines_ID;
	reg [5:0] Function_IDEX;
	wire [4:0] rs_ID, rt_ID, rd_ID, read1, read2;
	wire [5:0] opcode_ID, function_ID;


	//*********************REGISTER FILE***********************//
	//*********************************************************//

	initial
		// INITIAL REGISTER FILE VALUES
	begin
		RF[0] = 32'h00000000;
		RF[1] = 32'h00000000;
		RF[2] = 32'hFFFFFFFF;
		RF[3] = 32'h00000005;
		RF[4] = 32'h00000004;
		RF[5] = 32'h00000005;
	end

	//*********************************************************//


	//*********************DECODE INSTRUCTION******************//
	//*********************************************************//

	assign rs_ID = instruction_IFID[25:21];
	assign rt_ID = instruction_IFID[20:16];
	assign rd_ID = instruction_IFID[15:11];

	assign read1 = rs_ID;						// Rs ADDRESS
	assign read2 = rt_ID;						// Rt ADDRESS

	//*********************************************************//


	//*********************REG FILE DATA***********************//
	//*********************************************************//
```

```
	always @(posedge clk)
	begin
		Data1 <= RF[read1];                    // Rs DATA
		Data2 <= RF[read2];                    // Rt DATA
		RF[rd_MEMWB] <= DataMemRead_WB;        // WRITEBACK DATA
    end

	//*********************************************************//


	//*********************CONTROL LINES*********************//
	//*********************************************************//

	parameter Rformat = 6'b000000, LW = 6'b100011, SW = 6'b101011, BEQ = 6'b000100;

	assign opcode_ID = instruction_IFID[31:26];
	assign function_ID = instruction_IFID[5:0];

	always @(posedge clk)
	begin
		case (opcode_ID)
			Rformat:     ControlLines_ID = 9'b100100010;
			LW:          ControlLines_ID = 9'b011110000;
			SW:          ControlLines_ID = 9'b000000101;
			default:     ControlLines_ID = 9'b000000000;
		endcase
	end

	//*********************************************************//


	//*********************ID/EX PIPELINE*********************//

	always @(negedge clk)
	begin
		rs_IDEX <= rs_ID;
		rt_IDEX <= rt_ID;
		rd_IDEX <= rd_ID;
		Data1_IDEX <= Data1;
		Data2_IDEX <= Data2;
		ControlLines_IDEX <= ControlLines_ID;
		Function_IDEX <= function_ID;
		instruction_IDEX <= instruction_IFID;
	end

	//*********************************************************//


	//*********************OUTPUTS*****************************//
	//*********************************************************//

	assign Rs_IDEX = rs_IDEX;
	assign Rt_IDEX = rt_IDEX;
	assign Rd_IDEX = rd_IDEX;
	assign DATA1_IDEX = Data1_IDEX;
	assign DATA2_IDEX = Data2_IDEX;
	assign Control_IDEX = ControlLines_IDEX;

	//*********************************************************//



/*===========================================================*/
/*===========================================================*/
/*=====================EXECUTE (EX)==========================*/
```

```
/*=============================================================*/
/*=============================================================*/

	reg[31:0] ForwardA = 0, ForwardB = 0;
	reg[4:0] rd_EX, rd_EXMEM;
	reg[31:0] ControlLines_EXMEM, ALUResult_EXMEM, instruction_EXMEM, Data2_EXMEM;
	wire[31:0] A, B;
	wire[31:0] ALUResult_EX;
	wire[5:0] ALUCtrlWire;

	parameter Add = 6'b000010, Sub = 6'b000100, And = 6'b000000, Or = 6'b000001;


	//*********************DESTINATION REGISTER (EX)**********//
	//********************************************************//

	always @(posedge clk)
	begin
		if (ControlLines_IDEX[8])		// IF RegDst
		begin
			rd_EX <= rd_IDEX;
		end
		else
		begin
			rd_EX <= rt_IDEX;
		end
	end

	//********************************************************//


	//*********************FORWARDING UNIT********************//
	//********************************************************//

	always @(posedge clk)
					// FORWARD A
	begin
		if ((ControlLines_EXMEM[5]) && (rd_EXMEM == rs_IDEX))
					// IF EX RegWrite & DEST/SOURCE REGS ARE SAME
		begin
			ForwardA <= 2'b10;
					// SELECT ALURESULT
		end
		else if ((ControlLines_MEMWB[5]) && (rd_MEMWB == rs_IDEX))
					// IF MEM RegWrite & DEST/SOURCE REGS ARE SAME
		begin
			ForwardA <= 2'b01;
					// SELECT WRITE BACK DATA
		end
		else
		begin
			ForwardA <= 2'b00;
					// OTHERWISE USE DATA1 FROM REG FILE
		end
	end

	always @(posedge clk)
					// FORWARD B
	begin
		if ((ControlLines_EXMEM[5]) && (rd_EXMEM == rt_IDEX))
					// IF EX RegWrite & DEST/TARG REGS ARE SAME
		begin
			ForwardB <= 2'b10;
					// SELECT ALURESULT
		end
```

```
        else if ((ControlLines_MEMWB[5]) && (rd_MEMWB == rt_IDEX))
                                    // IF MEM RegWrite & DEST/TARG REGS ARE SAME
        begin
              ForwardB <= 2'b01;
                                    // SELECT WRITE BACK DATA
        end
        else
        begin
              ForwardB <= 2'b00;
                                    // OTHERWISE USE DATA1 FROM REG FILE
        end
   end


                                    // ASSIGN VALUE TO INPUT 'A' OF ALU
   assign A = (ForwardA == 2'b01) ? DataMemRead_WB :
                              (ForwardA == 2'b10) ? ALUResult_EXMEM :
                              Data1_IDEX;

                                    // ASSIGN VALUE TO INPUT 'B' OF ALU
   assign B = (ForwardB == 2'b01) ? DataMemRead_WB :
                              (ForwardB == 2'b10) ? ALUResult_EXMEM :
                              Data2_IDEX;

   //*********************************************************//


   //*********************ALU*********************************//
   //*********************************************************//

   assign ALUCtrlWire = (ControlLines_IDEX[1:0] == 2'b00) ? Add :     // LW OR SW
                 (ControlLines_IDEX[1:0] == 2'b01) ? Sub :           // BEQ
                 Function_IDEX;                                      // R FORMAT

assign ALUResult_EX = (ALUCtrlWire == 0) ? A & B:
                      (ALUCtrlWire == 1) ? A | B:
                      (ALUCtrlWire == 2) ? A + B:
                      (ALUCtrlWire == 4) ? A - B:
                      0;                                    //DEFAULT VALUE
   //*********************************************************//


   //**********************EX/MEM PIPELINE********************//
   //*********************************************************//

   always @(negedge clk)
   begin
         ControlLines_EXMEM <= ControlLines_IDEX;
         ALUResult_EXMEM <= ALUResult_EX;
         rd_EXMEM <= rd_EX;
         instruction_EXMEM <= instruction_IDEX;
         Data2_EXMEM <= Data2_IDEX;

   end

   //*********************************************************//


   //**********************OUTPUTS****************************//
   //*********************************************************//

   assign ALURESULT_EXMEM = ALUResult_EXMEM;
   assign Rd_EXMEM = rd_EXMEM;
   assign Controls_EXMEM = ControlLines_EXMEM;
```

```
	//*********************************************************//


/*=============================================================*/
/*=============================================================*/
/*=====================MEMORY (MEM)============================*/
/*=============================================================*/
/*=============================================================*/

	reg [31:0] ControlLines_MEMWB, DataMemRead_MEMWB, ALUResult_MEMWB;
	reg [31:0] DataMemRead_MEM, instruction_MEMWB;
	reg [4:0] rd_MEMWB;
	reg [31:0] dmem [31:0];


	//*********************DATA MEMORY***********************//
	//*********************************************************//

	always @(posedge clk)
	begin
		if (ControlLines_EXMEM[4])                    // IF MemRead
		begin
			DataMemRead_MEM <= dmem[ALUResult_EXMEM];// LOAD DATA FROM MEMORY
		end
		else if (ControlLines_EXMEM[3])               // IF MemWrite
		begin
			dmem[ALUResult_EXMEM] <= Data2_EXMEM;   // STORE DATA IN MEMORY
		end
	end

	//*********************************************************//


	//*********************MEM/WB PIPELINE*******************//
	//*********************************************************//

	always @(negedge clk)
	begin
		ControlLines_MEMWB <= ControlLines_EXMEM;
		ALUResult_MEMWB <= ALUResult_EXMEM;
		rd_MEMWB <= rd_EXMEM;
		DataMemRead_MEMWB <= DataMemRead_MEM;
		instruction_MEMWB <= instruction_EXMEM;
	end

	//*********************************************************//


/*=============================================================*/
/*=============================================================*/
/*=====================WRITE BACK (WB)=========================*/
/*=============================================================*/
/*=============================================================*/

	wire [31:0] DataMemRead_WB;


	//*********************WRITEBACK DATA**********************//
	//*********************************************************//

	assign DataMemRead_WB = (ControlLines_MEMWB[6] == 1) ? ALUResult_MEMWB :
				DataMemRead_MEMWB;
	//*********************************************************//
endmodule
```

Figure II: Lab 9 Test Bench

```
module Lab9_tb;
	reg CLK;
	wire [31:0] PC;
	wire [31:0] Instruction;
	wire [31:0] ALUResult;

	Lab9 uut(
		.CLK(clk),
		.PC(PC_Reg),
		.Instruction(Instruction_IFID),
		.ALUResult(ALURESULT_EXMEM)
		);

		initial begin
			CLK = 0;
			#10;
			#400;
			$finish;
		end
		always
		begin
			#10;
			CLK = ~CLK;
		end

endmodule
```