Phase 3 of the Project – ECE 4120/5120 (Spring 2025)

Due Date: April 28, 2025

(Worth 50% of the Project, + Bonus of about 20%)

Goals: 1. To design control unit, additional hardware for handling branch instruction, another memory unit for data memory and auxiliary multiplexers

2. Modify ALU to handle branch instructions
3. Combine all the phases of the project for **single cycle implementation** (Figure 1).
4. To **implement the piplined architecture** leveraging the single cycle processor design without the forwarding unit and hazard unit (Figure 2).
5. Bonus: Implement **Forwarding** and demonstrate it for data-hazards (Figure 3). Next implement hazard unit for inserting bubbles and demonstrate it for between load word and R-type instructions. Up to 20% worth of points if you successfully implement, and demonstrate **both**. ***You'll not be penalized at all for not finishing the bonus tasks***.

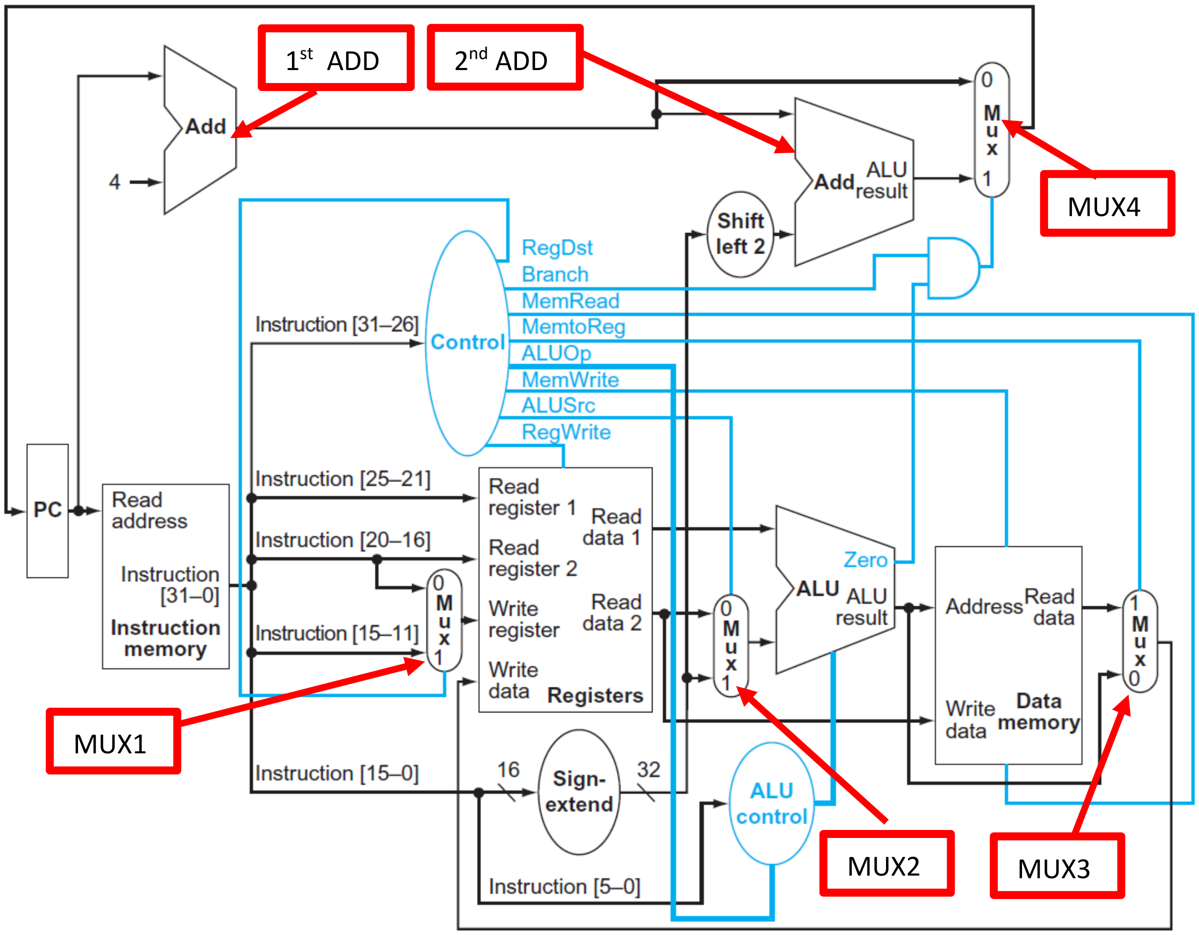

Figure 1. Single Cycle Processor (Required to get full credit)

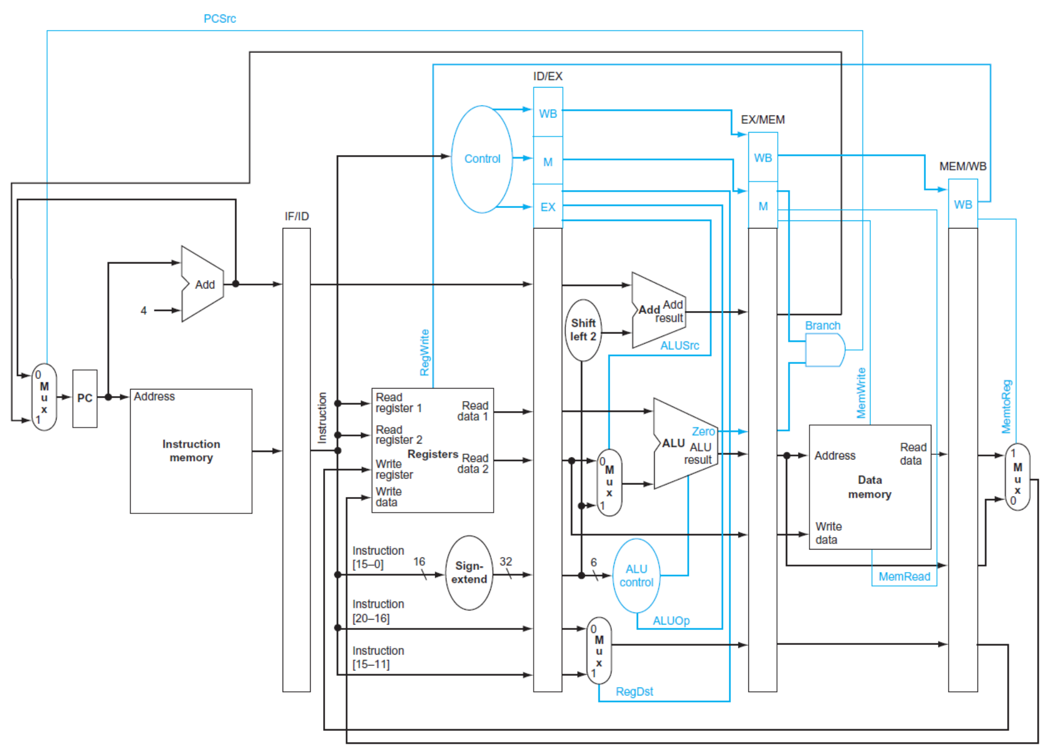

Figure 2. Simple Pipelined Architecture (Required to get full credit)

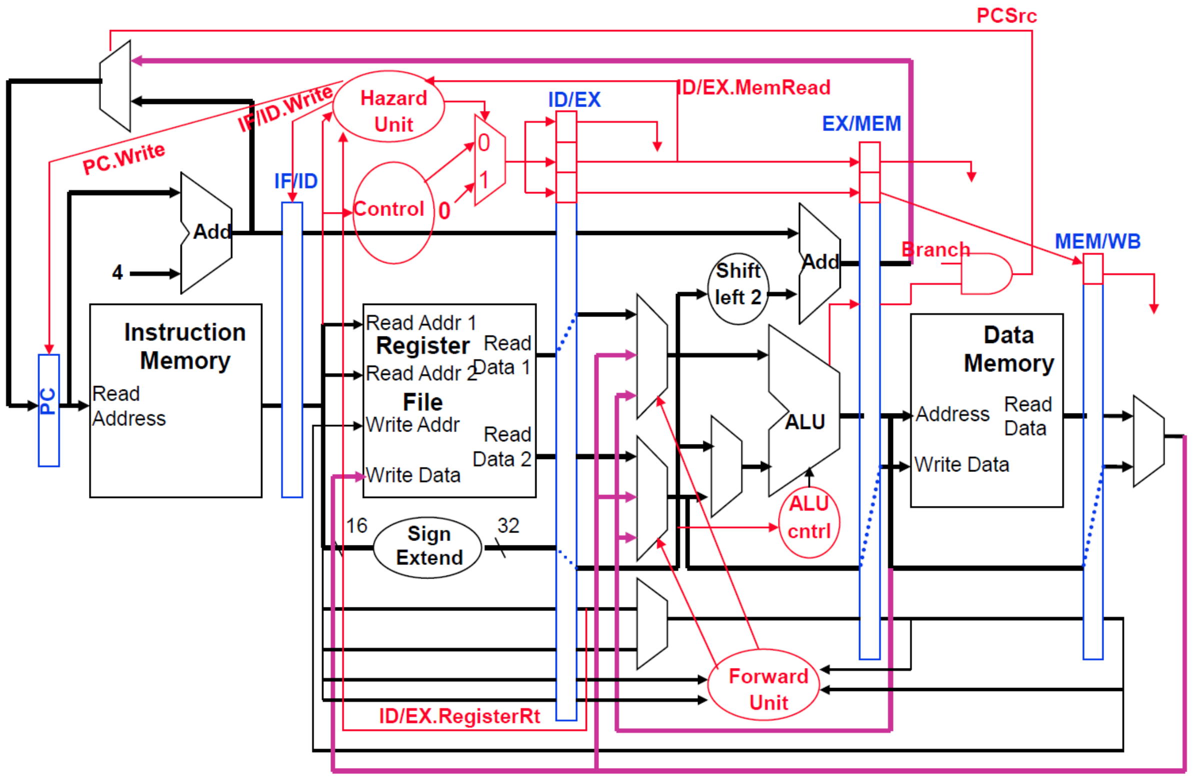

Figure 3. Pipeline with Forwarding and Hazard Units (Bonus)

High level Description of each unit:

You have already designed following units:

Phase 1. PC, Instruction Memory, 1st Add units
Phase 2. Register, Sign Extend, Mux 2, ALU

In this phase you need to design and modify following units for Single Cycle Implementation:

1. **Control unit**, which has opcode as input and 8 outputs:
    a. Regdst → select signal for Mux1 to decide write address operand
    b. Branch → Indicates whether the instruction is a branch or not. This signal feeds to MUX4 select bit via AND gate.
    c. RegWrite → To write value into the register
    d. ALU Src → Select signal for MUX 2 (this MUX is already included in Phase 2 of the project), which decides the 2nd operand input to the ALU
    e. And f. **MemRead and MemWrite** → These signals are bolded because it needs to be connected with the ***wren*** of the memory mega function (Hint. You may not need to use both the signals).
    g. MemtoReg is the select signal to Mux3
    h. Branch is the input to AND gate
    i. ALUop decides what operation to perform in ALU, note that this is an input to ALU control unit.

2. Modify the ALU code given in phase 2 of the project to accommodate branch operation.
3. 2nd Add unit.
4. Three Multiplexers Mux 1, Mux 3 and Mux 4
5. ALU control Unit (you can use of COD Figures 4.12 and 4.13 to implement it).
6. For Figure 2 (and bonus part) you need to implement the intermediate registers.

Constraints:

1. Use another instantiation of the same memory IP for data memory and modify as required.
2. PC, Instruction Memory, Register, Control Unit, Data Memory all **should be clocked with the same clock source** for the single cycle implementation.
3. Preload the registers, using lw command or addi command.
4. For testing purposes, follow the test bench requirements provided below.

Test Bench Requirements:

Write testbench to execute following sets of MIPS instructions (for single cycle and pipelined implementation, implement these instructions with branch not taken assumption).

i. beq $t1, $t2, Equal # if t1 equals t2 then go to Equal (v instruction)
ii. add $t1, $t1, $t2
iii. sw $t3, 100($t2)
iv. or $t1, $t4, $t2
v. Equal: …..

<u>For bonus points Forward Unit:</u> Implement all the four instructions: assume the case of branch not taken and show forwarding and stalling operation between lw and add instructions, and forwarding between add and or instructions.

i. beq $t1, $t2, Equal # if t1 equals t2 then go to Equal (v instruction)
ii. add $t2, $t1, $t2
iii. sw $t3, 100($t2)
iv. add $t4, $t1, $t2
v. Equal: …..

<u>For bonus points Forward Unit AND Hazard Unit:</u> Implement all the four instructions: assume the case of branch not taken

i. beq $t1, $t2, Equal # if t1 equals t2 then go to Equal (vi instruction)
ii. lw $t1, 8($t2)
iii. add $t2, $t1, $t2
iv. sw $t3, 100($t2)
v. add $t4, $t1, $t2
vi. Equal: …..

Deliverables:

I. A pdf file containing following:

1. Modified block diagrams for two cases (three cases for bonus points), showing the clock signals, number of bits provided to each unit and control signals.
2. Elaboration on VHDL implementation of each unit
3. Flow summary, RTL view and Technology map view
4. Elaboration on test bench: e.g. how you confirmed the execution of these instructions, how you loaded the registers, etc.
5. Elaboration on the waveform and snapshot of the waveform should also be included. Show ***both loading the register*** and ***writing the result into the registers and memory*** using the waveforms (use as many waveform as you require to elaborate, but don't clutter all of them into one zoomed out waveform). Proper labeling of the waveform is important – bottom line if I cannot understand the waveform and your elaboration for each important event is not described and labeled properly then you are not going to get the full credit.
6. It is normal rather encouraged to show separate waveforms for each instruction – please make sure that your .qpf submission matches with what you are submitting as the final phase – otherwise explain what differences should I expect.
7. State the improvements you noticed using pipelining in latency, throughput and Fmax.
8. State the hardware overhead related to pipeline only.
9. **<u>For Bonus points:</u>** In addition to above items, state the **<u>performance penalty</u>** associated with having the forwarding unit only or having both the forwarding and hazard units.
10. **<u>For Bonus points:</u>** In addition to above items, state the **<u>hardware overhead</u>** associated with having the forwarding unit only or having both the forwarding and hazard units.

II. **A zipped VHDL Implementation of the Project**: A folder containing all the files generated by Quartus II, including the all the .vhd, should be submitted to the dropbox via ilearn. Name the submitted folder as your lastname1_lastname2_S25_phase3.

Presentation and Demonstration:

In the last week of the classes (4/29 and 4/31) we are going to have presentations and demonstrations. Each group is going to elaborate on the key achievements for five minutes (it'll be timed – details are given below). Then each group will show the best simulation result to me over Altera-ModelSim (details are given below).

More on Presentation:

1. Your presentation should have no more than five slides (excluding the title slide).
2. Slide 1. Explain whatever the final working project that you achieved – e.g. able to achieve complete integration of the three phases for single cycle implementation (Fig. 1), or similar statements for Fig. 2 and Fig. 3 – whatever you are able to achieve successfully.
3. Slide 2.
    a. Explain how you loaded the register values (e.g. addi, using test bench, etc.). Explain what values of registers are chosen and if there is any rationale to chosen values.
    b. Explain how you loaded the data memories (e.g. .mif, using test bench, etc.). Explain what values of data memory locations are chosen and if there is any rationale to chosen values.
    c. Explain what values you loaded to the instruction memories – machine code and assembly instruction.
4. Slide 3 – 5 Explain the best results you are able to achieve – some examples are given below:
    i. If you are only able to do single cycle implementation – then show whether beq is taken or not – and whether store instruction able to store data or not?
    ii. If you are able to implement simple pipelined – then show fourth, fifth and sixth clock cycles to indicate **pipelining is working** and **expected results are obtained at the respective clock cycles**. I am interested in seeing the values of ALU outputs and DATA memory.
5. If you are able to implement **bonus part (forwarding only)** then **you can have two additional slides to show forwarding is working.** I am interested in seeing the forwarding working between instructions ii & iii and ii & iv.
6. If you are able to implement **bonus part (both forwarding and hazard)** then **you can have three additional slides to show forwarding and hazard both are working.** I am interested in seeing the hazard unit for instruction ii & iii and forwarding working for instructions iii & iv and iii & v.

Demonstration of Successful Simulation

Each group needs to show on their computer following: Showing the waveform demonstrating whatever cases you presented in the slides - using Modelsim-Altera – as shown in the class. ----

Please keep your waveform that you want to demonstrate already showing up on your laptop so that you can demonstrate it right after your presentation on your computer.

Timing of the Presentation and other Logistics:

1. Each group will be given **five minutes** to do their power-point presentations ---- it will be **strictly timed!!**
2. For students with COMPLETELY FUNCTIONAL bonus part will be given extra **three minutes.**
3. Attendance in both the presentation sessions is mandatory, and I'll mark attendance on both days 4/23 and 4/25.
4. Please keep your waveform that you want to demonstrate, already executed and showing on your laptop so that you can demonstrate it right after your presentation without delay. Failure to do that will **result in penalty** to your grades. For student with COMPLETELY FUNCTIONAL bonus part can demonstrate ONLY the part that reflects successful implementation of bonus part.

============Sample Readme for Demonstration on the Board==================

****

Demo Instructions:

****

1. Deploy the <name of your project>(Simple Pipelined) project to the DE10-Lite board

using the instructions in the <name of your Readme Instructions for the Complete Project > file found in the project archive.

2. After flashing, with all switches in the DOWN (OFF) position,

press KEY0 to generate a clock pulse.

3. Repeatedly press KEY0 for as many clock cycles as desired. Observe the reg_wr_data values shown in hex on the 7-segment displays. Expected values are shown below. (You need to do further investigation on how to show the data for sw instruction).

KEY1 is a synchronous reset.

Note: See <name of your Readme Instructions for the Complete Project > for further explanation of board controls and features.

****

Expected Values (following values are just an example):

****

Clock Cycle	reg_wr_data (hex)	notes
0	0x000000	<- board startup
1	0x000000	<- synchronous reset
2	0x000000	
3	0x000000	
4	0x000000	

5	0x00000A	
6	0x00001C	
7	0x000037	
8	0x000000	
9	0xFFFFEE	<- reg_write = 0, mem_to_reg = X
10	0x000026	
11	0x000037	<- reg_write = 0, mem_to_reg = X
12	0x000000	
13	0x00003E	
14..	0x000000	<- repeats until PC overflows

==

Sample of Readme Instructions for the Complete Project

==

****

Project Organization:

****

mips_p3_sc = Single Cycle

mips_p3_pl = Simple Pipelined

mips_p3_fw = Pipelined with Forwarding

All projects have the same interface on the DE10-Lite board.

****

To Run ModelSim simulation:

****

1. Open <your qpf file> in Quartus
2. In menu bar click: Processing -> Start Compilation
3. After compilation, in menu bar click: Tools -> Run Simulation Tool -> Gate Level Simulation
4. ModelSim will open, run the simulation, and display the waveform.

****

To deploy to DE10-Lite board:

****

1. Open <your qpf file> in Quartus
2. In menu bar: Processing -> Start Compilation
3. In menu bar: Tools -> Programmer
4. Click "Hardware Setup"
5. Select "USB Blaster"
6. Click "Close"
7. Click "Start" to begin programming

****

Explanation of board controls/features:

****

KEY0:

Clock control in manual mode: Press this button to cause a clock pulse while in manual mode.

No effect in automatic mode.

KEY1:

Synchronous reset.

Manual mode: Hold down this button and press KEY0 to cause a reset.

Automatic mode: Hold down this button and wait for clock pulse to cause a reset.

SW(1..0):

"00" (down, down): manual mode. Press KEY0 for clock pulse.

"01" (down, up): automatic mode. 1 Hz clock is used.

"10" (up, down): automatic mode. 20 Hz clock is used.

"11" (up, up): automatic mode. 50 MHz clock is used.

SW(9..7):

"000" (down, down, down): LEDs display control signals: reg_write, alu_op_ctl (4-bit), alu_src (2-bit), pc_src, reset, clk.

"001" (down, down, up): LEDs display PC(11..2)

"010" (down, up, down): LEDs display imem_out(31..22)

"011" (down, up, up): LEDs display imem_out(21..12)

"100" (up, down, down): LEDs display imem_out(11..2)

"101" (up, down, up): LEDs display imem_out(9..0)

any other: LEDs are all OFF

SW(6..2):

Select 5b register address (in binary: up = 1, down = 0) to display on the 7-segment displays.

If you select register 0 (all switches down), reg_wr_data will be shown instead.

When register address is zero:

HEX(5..0): Shows reg_wr_data in hex (lower 24 bits)

When register address is non-zero:

HEX(5..0): Shows selected register value in decimal

****

Explanation of project files:

****

alu_control.vhd	ALU control unit. VHDL file.
alu.vhd	ALU unit. VHDL file.
bin2seg7.vhd	Binary to 7-segment modules. VHDL file.
binary_to_bcd_digit.vhd	Binary to BCD modules. VHDL file.
binary_to_bcd.vhd	Binary to BCD modules. VHDL file.
clock_div.vhd	Counter-based clock divider. VHDL file.

data_mem.cmp	Data memory IP file. VHDL component file.
data_mem.qip	Data memory IP file. Quartus IP file.
data_mem.vhd	Data memory IP file. VHDL file.
forw_control.vhd	Forwarding control unit. VHDL file.
instr_mem.cmp	Instruction memory IP file. VHDL component file.
instr_mem.mif	Instruction memory IP file. Initialization file.
instr_mem.qip	Instruction memory IP file. Quartus IP file.
instr_mem.vhd	Instruction memory IP file. VHDL file.
main_control.vhd	Main control unit. VHDL file.
mips_p3_xx_top.vhd	Top module for HW (DE10-Lite board) deployment. VHDL file.
mips_p3_xx.qpf	Quartus project file.
mips_p3_xx.qsf	Quartus settings file.
mips_p3_xx.sdc	Synopsys Design Constraint file.
pb_debounce.vhd	Synchronizer and debounce module for asynchronous inputs. VHDL file.
proc_tb.do	ModelSim script file. Ran by Quartus to set up and start my simulation.
proc_tb.vht	VHDL Testbench file for testing proc module.
proc.vhd	Processor module. VHDL file.
reg_file.vhd	Register file module. VHDL file.
reg_n.vhd	n-bit register module. VHDL file.
reset_control.vhd	Reset control module. VHDL file.
six_dig_disp.vhd	Display driver module. VHDL file.
wave.do	ModelSim script file. Sets up waveforms. Ran by proc_tb.do