CS61C Fall 2026

RISC-V Pipelining, Hazards

Discussion 9

1 Datapath Review

1.1 True or False? The single cycle datapath uses the outputs of all hardware units for each instruction.

False. All units are active in each cycle, but their output may be ignored (gated) by control signals.

1.2 True or False? It is possible to execute the stages of the single-cycle datapath in parallel to speed up execution of a single instruction.

False. Each stage depends on the value produced by the stage before it (e.g., instruction decode depends on the instruction fetched). We *can* execute the stages in parallel if we insert registers to pipeline the stages.

1.3 Which instruction(s) are responsible for the critical path?

Load instructions. These instructions use all 5 datapath stages.

1.4 Why is the single-cycle datapath inefficient?

At any given time, most of the parts of the single cycle datapath are not being used. Even though not every instruction exercises the critical path, the datapath can only be clocked as fast as the slowest instruction.

2 Performance Analysis

Register clk-to-q 30 ps	**Branch comp.** 75 ps	**DMEM write setup** 200 ps
Register setup 20 ps	**ALU** 200 ps	**Memory read** 250 ps
Register hold 10 ps	**Imm. Gen.** 15 ps	**Mux** 25 ps
RegFile read 100 ps	**RegFile setup** 20 ps	

Copied above are the same sample delays and setup times for each of the datapath components and registers. Use these, in conjunction with the pipelined datapath on the last page, to answer the questions below.

2.1 What would be the fastest possible clock time for a **single cycle** datapath? Recall that load instructions exercise the critical path.

HINT: $t_{\text{clk-cycle}} \geq t_{\text{clk-to-q}} + t_{\text{longest-combinational-path}} + t_{\text{setup}}$

$$\begin{aligned} t_{\text{clk}} &\geq t_{\text{PC clk-to-q}} + t_{\text{IMEM read}} + t_{\text{RF read}} + t_{\text{mux}} + t_{\text{ALU}} + t_{\text{DMEM read}} + t_{\text{mux}} + t_{\text{RF setup}} \\ &\geq 30 + 250 + 100 + 25 + 200 + 250 + 25 + 20 \\ &\geq 900 \text{ ps} \end{aligned}$$

Note that we take the maximum of the path from IMEM->Regfile->Mux->ALU and IMEM->ImmGen->Mux->ALU. With this hardware configuration, the longest path is through the Register file + ASel Mux.

2.2 What is the fastest possible clock time for a pipelined datapath?

HINT: First identify the critical path delay (longest path between two registers) for each stage.

$$\begin{aligned} \textbf{IF} &: t_{\text{PC clk-to-q}} + t_{\text{IMEM read}} + t_{\text{Reg setup}} = 30 + 250 + 20 = 300 \text{ ps} \\ \textbf{ID} &: t_{\text{Reg clk-to-q}} + t_{\text{RF read}} + t_{\text{Reg setup}} = 30 + 100 + 20 = 150 \text{ ps} \\ \textbf{EX} &: t_{\text{Reg clk-to-q}} + t_{\text{mux}} + t_{\text{ALU}} + t_{\text{Reg setup}} = 30 + 25 + 200 + 20 = 275 \text{ ps} \\ \textbf{MEM} &: t_{\text{Reg clk-to-q}} + t_{\text{DMEM read}} + t_{\text{Reg setup}} = 30 + 250 + 20 = 300 \text{ ps} \\ \textbf{WB} &: t_{\text{Reg clk-to-q}} + t_{\text{mux}} + t_{\text{RF setup}} = 30 + 25 + 20 = 75 \text{ ps} \end{aligned}$$

$$t_{\text{clk}} \geq \max(\textbf{IF}, \textbf{ID}, \textbf{EX}, \textbf{MEM}, \textbf{WB}) = 300 \text{ ps}$$

Note that for the ID and EX stages, the immediate generator and branch comparator delays are overshadowed by the longer delays of RegFile read and ALU respectively.

2.3 At steady state (i.e. ignore the first few cycles), how many instructions finish every cycle in the single-cycle datapath? What about the pipelined datapath, assuming no hazards occur?

In a single-cycle datapath, 1 instruction completes every cycle. For a pipelined datapath at steady state, 1 instruction finishes the WB stage and completes every cycle too. Hence, both datapaths have cycles per instruction (CPI) of approximately 1.

Note that in a real-world pipelined datapath, CPI is often greater than 1 due to stalls from data, and control hazards. However, techniques like forwarding and branch prediction can mitigate this.

2.4 What is the fastest possible clock frequency of the single-cycle datapath compared to that of the pipelined datapath? Based on this, how do their instruction **throughputs** differ, assuming no hazards occur?

$$\text{Clock frequency} = \frac{1}{\text{Clock period}}$$

$$\text{Throughput} = \frac{\text{Clock frequency}}{\text{Cycles per instruction (CPI)}}$$

Note that the CPI for both is ~1. For the single-cycle datapath,

$$\text{Clock frequency} = \frac{1}{900 \text{ ps}} \approx 1.11 \text{ GHz}$$

$$\text{Throughput} \approx 1.11 \text{ billion instructions per second}$$

For the pipelined datapath,

$$\text{Clock frequency} = \frac{1}{300 \text{ ps}} \approx 3.33 \text{ GHz}$$

$$\text{Throughput} \approx 3.33 \text{ billion instructions per second}$$

Notice that by introducing pipelining and reducing the critical path delay, we increase the throughput, allowing more instructions to complete each second! In reality, throughput for the pipelined datapath will often be less than calculated due to hazards.

2.5 What is the speedup from the single cycle datapath to the pipelined datapath? Why is the speedup less than 5x?

$$\text{Speedup} = \frac{\text{Pipelined throughput}}{\text{Single-cycle throughput}} = \frac{3.33 \text{ GHz}}{1.11 \text{ GHz}} = 3\text{x}$$

Alternatively,

$$\text{Speedup} = \frac{\text{Single-cycle clock time}}{\text{Pipelined clock time}} = \frac{900 \text{ ps}}{300 \text{ ps}} = 3\text{x}$$

The speedup is less than 5 because:

1) the necessity of adding pipeline registers, which have clk-to-q and setup times

2) the need to set the clock to the *maximum* of the five stages.

Note: Hazards require additional logic to resolve which would result in an even smaller performance increase.

3 Solving Data Hazards

One of the costs of pipelining is that it introduces pipeline hazards. Hazards, generally, are issues with something in the CPU's instruction pipeline that could cause the next instruction to execute incorrectly. Recall that **data hazards** are caused by data dependencies between instructions. In CS 61C, where we always assume that instructions go through the processor in order, we see data hazards when an instruction reads a register before a previous instruction has finished writing to that register.

For all questions, assume **no branch prediction** or **write-then-read** in one cycle for the RegFile.

Forwarding

Most data hazards can be resolved by forwarding, which is when the result of the EX or MEM stage is sent to the EX stage for a following instruction to use.

Note: There are two ways of forwarding - **MEM to EX** and **WB to EX**. How are each of these implemented in hardware? We add 2 wires: one from the beginning of the MEM stage for the output of the ALU (right after the EX/MEM pipelined register) and one from the beginning of the WB stage (right after the MEM/WB pipelined register). Both of these wires will connect to the A/B muxes in the EX stage.

3.1 Look for data hazards in the code below, and figure out how forwarding could be used to solve them.

Instruction	C1	C2	C3	C4	C5	C6	C7
`1. addi t0, a0, -1`	IF	ID	EX	MEM	WB		
`2. and s2, t0, a0`		IF	ID	EX	MEM	WB	
`3. sltiu a0, t0, 5`			IF	ID	EX	MEM	WB

There are two data hazards, between instructions 1 and 2, and between instructions 1 and 3. The first could be resolved by forwarding the ALU output in the MEM stage to the beginning of the EX stage in C4, and the second could be resolved by forwarding the ALU output in the WB stage in C5 to the beginning of the EX stage in C5.

3.2 Imagine you are a hardware designer working on a CPU's forwarding control logic. How many instructions after the `addi` instruction could be affected by data hazards created by this `addi` instruction?

Three instructions. For example, with the `addi` instruction, any instruction that uses t0 that has its ID stage in C3, C4, or C5 will not have the result of `addi`'s writeback in C5. If, however, we are allowed to assume write-then-read to registers, then it would only affect two instructions since the ID stage of instruction 4 would be allowed to line up with the WB stage of instruction 1.

Stalls

3.3 Identify the data hazards in the code below. One of them cannot be solved with forwarding—why? What can we do to solve this hazard?

Instruction	C1	C2	C3	C4	C5	C6	C7	C8
`1. addi s0, s0, 1`	IF	ID	EX	MEM	WB			

Instruction	C1	C2	C3	C4	C5	C6	C7	C8
`2. addi t0, t0, 4`		IF	ID	EX	MEM	WB		
`3. lw t1, 0(t0)`			IF	ID	EX	MEM	WB	
`4. add t2, t1, x0`				IF	ID	EX	MEM	WB

There are two data hazards in the code. The first hazard is between instructions 2 and 3, from t0, and the second is between instructions 3 and 4, from t1. The hazard between instructions 2 and 3 can be resolved with forwarding, but the hazard between instructions 3 and 4 cannot be resolved with only forwarding. This is because even with forwarding, instruction 4 needs the result of instruction 3 at the beginning of C6, and it won't be ready until the end of C6.

In addition to forwarding, we can fix this by stalling: insert a `nop` (no-operation) between instructions 3 and 4.

3.4 Say you are the compiler and can re-order instructions to minimize data hazards while guaranteeing the same output. How can you fix the code above?

Reorder the instructions 2-3-1-4, because instruction 1 has no dependencies.

Control Hazards

Control hazards are caused by jump and branch instructions, because for all jumps and some branches, the next PC is not PC + 4, but the result of the ALU available after the EX stage. We could stall the pipeline for control hazards, but this decreases performance.

3.5 Identify the control hazards in the code below. How can we resolve them?

Instruction	C1	C2	C3	C4	C5	C6	C7	C8	C9
`1. beq s0, s1, loop`	IF	ID	EX	MEM	WB				
`2. addi t0, t0, 4`		IF	ID	EX	MEM	WB			
`3. ori t1, t1, 7`			IF	ID	EX	MEM	WB		
`4. slli sp, sp, 2`				IF	ID	EX	MEM	WB	
`5. addi a0, t0 2`					IF	ID	EX	MEM	WB

There are three control hazards in the code. The first hazard is between instructions 1 and 2 because `addi t0, t0, 4` may not get executed if the branch condition is true. The second hazard is between instructions 1 and 3 for the same reason as above and similarly between instructions 1 and 4. The branch condition and ALU outputs are available at the start of the `MEM` stage (look at the pipeline register placement!) in C4, so we have to stall for 3 cycles. There is no control hazard between instructions 1 and 5 because there is no need to stall instruction 5 if the branch is not taken.

We can fix the hazards by stalling: insert three NOPs (no-operation) after the first instruction.

3.6 Besides stalling, what can we do to resolve control hazards?

We can try to predict which way branches will go, and if this prediction is incorrect, flush the pipeline and continue with the correct instruction. Not using branch prediction will always incur 3 stalls, while using branch prediction could save 3 stalls on a correct prediction.

4 Hazards Practice

Given the RISC-V code below and a 5-stage pipelined CPU with no forwarding, how many hazards would there be? What types are each hazard? Consider all possible hazards between all instructions.

How many stalls would there need to be in order to fix the data hazard(s) if the RegFile supports write-then-read? What about the control hazard(s) if we use branch prediction with perfect accuracy? There is no forwarding in this question.

Instruction	C1	C2	C3	C4	C5	C6	C7	C8	C9
1. `sub t1, s0, s1`	IF	ID	EX	MEM	WB				
2. `or s0, t0, t1`		IF	ID	EX	MEM	WB			
3. `sw s1, 100(s0)`			IF	ID	EX	MEM	WB		
4. `bgeu s0, s2, loop`				IF	ID	EX	MEM	WB	
5. `add t2, x0, x0`					IF	ID	EX	MEM	WB

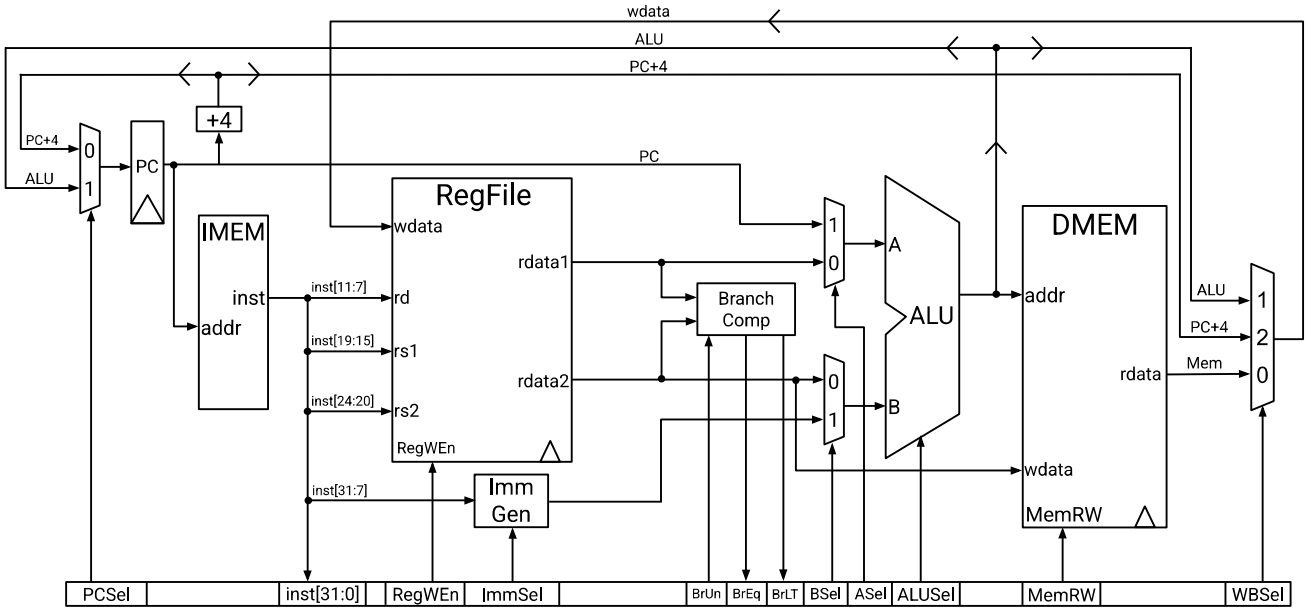
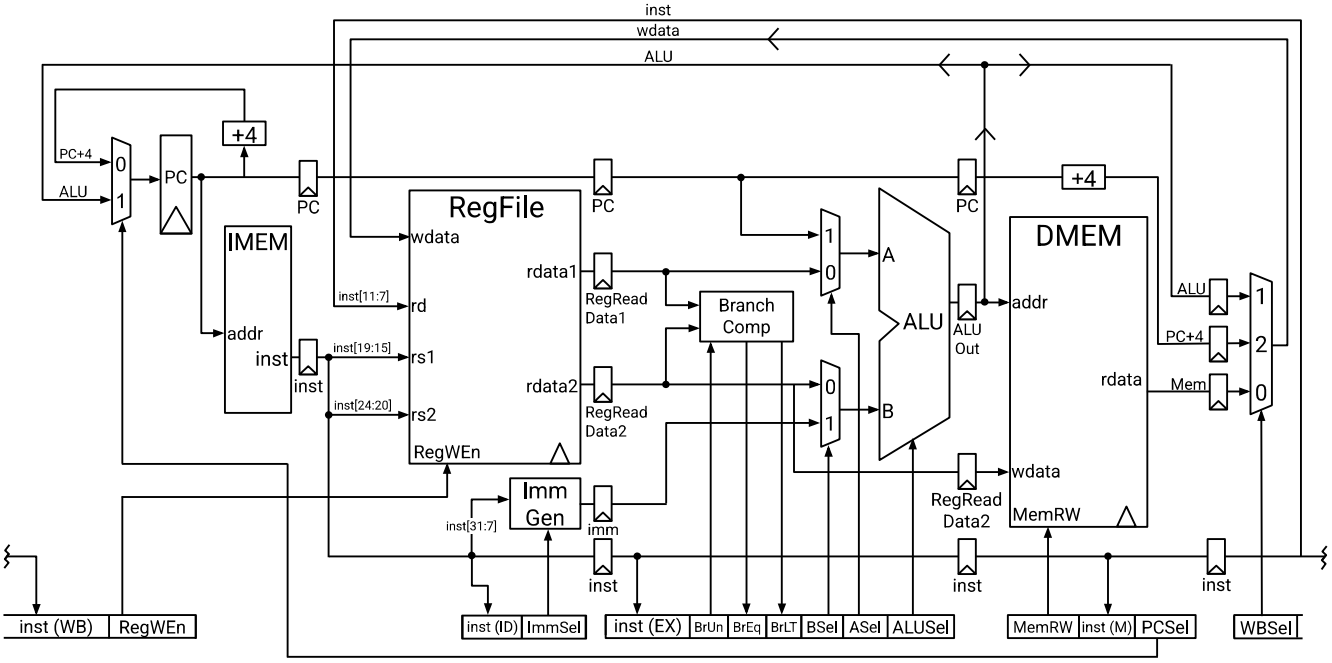
There are four hazards: between instructions 1 and 2 (data hazard from t1), instructions 2 and 3 (data hazard from s0), instructions 2 and 4 (from s0), and instructions 4 and 5 (a control hazard).

Assuming write-then-read, two stalls are needed between instructions 1 and 2, and two stalls are needed between instructions 2 and 3. For the control hazard, if we predicted correctly, then no stalls are needed, but if we predicted incorrectly, then we need 3 stalls to flush the pipeline. We don't need to stall for the hazard between 2 and 4 because stalling for instruction 3 already handles that.

Instruction	C1	C2	C3	C4	C5	C6	C7	C8	C9
`1. sub t1, s0, s1`	IF	ID	EX	MEM	WB				
`nop`		IF	X	X	X	X			
`nop`			IF	X	X	X	X		
`2. or s0, t0, t1`				IF	ID	EX	MEM	WB	
`nop`					IF	X	X	X	X
`nop`						IF	X	X	X
`3. sw s1, 100(s0)`							IF	ID	EX
`4. bgeu s0, s2, loop`								IF	ID
`nop`									IF

Instruction	...	C10	C11	C12	C13	C14	C15	C16	C17
`nop`	...	X							
`3. sw s1, 100(s0)`	...	MEM	WB						
`4. bgeu s0, s2, loop`	...	EX	MEM	WB					
`nop`	...	X	X	X	X				
`nop`	...	IF	X	X	X	X			
`nop`	...		IF	X	X	X	X		
`5. add t2, x0, x0`	...			IF	ID	EX	MEM	WB	

Single-Cycle Datapath Diagram



5-Stage Datapath Diagram