CS 61C Summer 2024 — RISC-V Pipelining

Discussion 8

1 Pre-Check

This section is designed as a conceptual check for you to determine if you conceptually understand and have any misconceptions about this topic. Please answer true/false to the following questions, and include an explanation:

1.1 By pipelining the CPU datapath, each single instruction will execute faster (latency is reduced), resulting in a speed-up in performance.

False. Because we implement registers between each stage of the datapath, the time it takes for an instruction to finish execution through the 5 stages will be longer than the single-cycle datapath we were first introduced with. A single instruction will take multiple clock cycles to get through all the stages, with the clock cycle based on the stage with the longest timing.

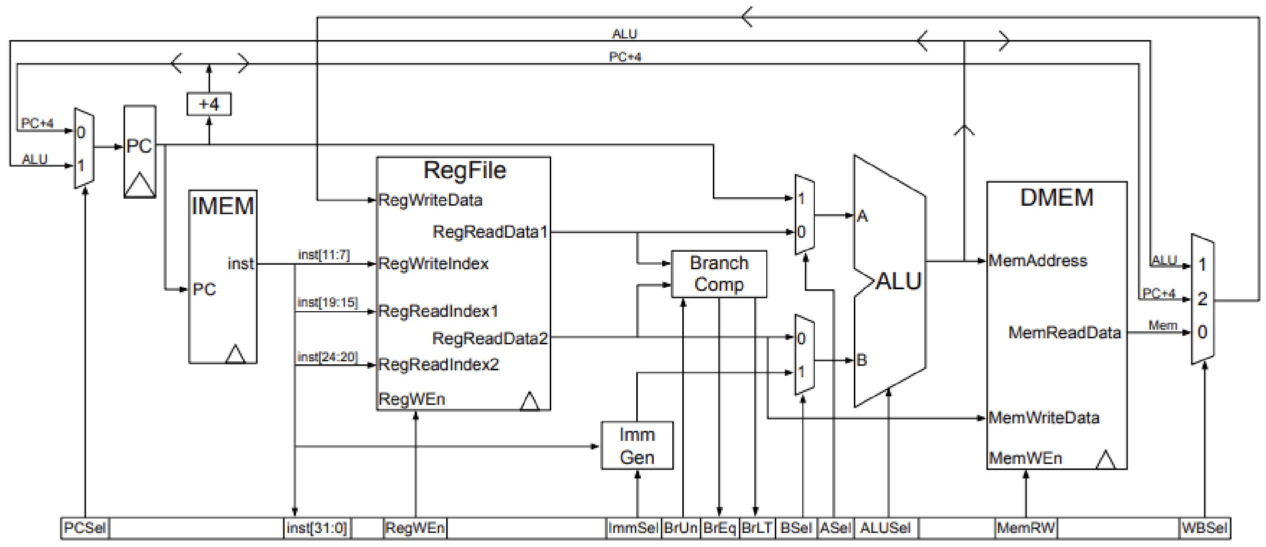
1.2 A pipelined CPU datapath results in instructions being executed with higher throughput (than the single-cycle CPU).

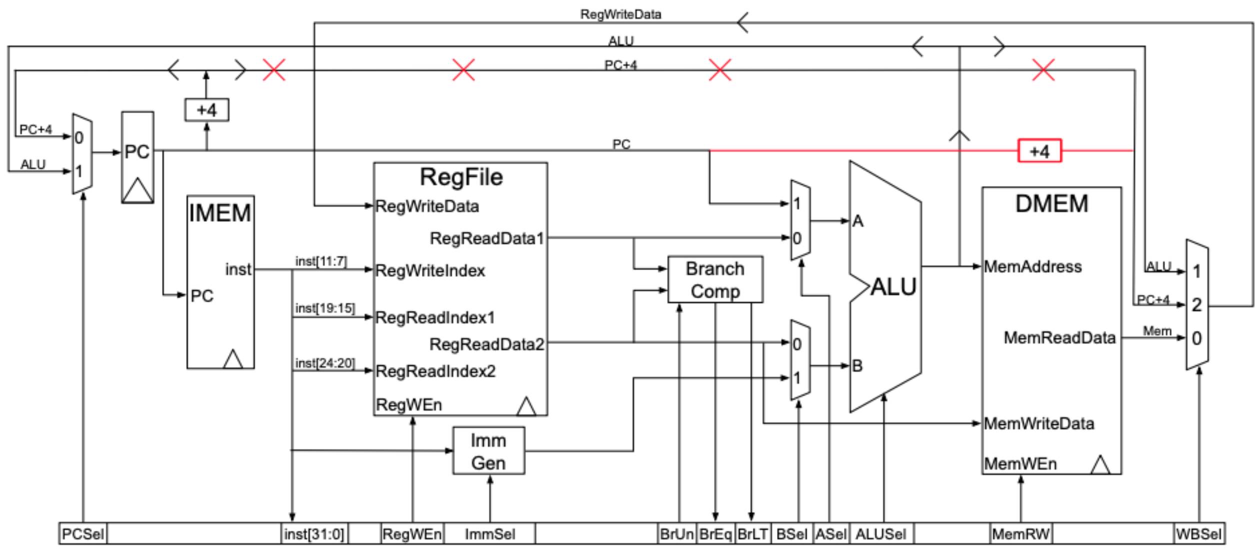
True. Recall that throughput is the number of instructions processed per unit time. Pipelining results in a higher throughput because more instructions are run at once, which utilizes more parts of the datapath simultaneously.

2 Pipelining Registers

Recall the five stages: In the **IF** stage, we use the Program Counter to access our instruction as it is stored in IMEM. Then, we separate the distinct parts we need from the instruction bits in the **ID** stage and generate our immediate, the register values from the RegFile, and other control signals. Afterwards, using these values and signals, we complete the necessary ALU operations in the **EX** stage. Next, anything we do in regards with DMEM (not to be confused with RegFile or IMEM) is done in the **MEM** stage, before we hit the **WB** stage, where we write the computed value that we want back into the return register in the RegFile.

In order to pipeline, we separate the datapath into 5 discrete stages. These 5 stages, divided by registers, allow operation of different stages of the datapath in the same clock period. Different instructions can use different stages at a time. At each clock cycle, the necessary inputs into a particular stage are sampled at the rising clock edge (and available after the clk-to-q delay). After the stage operates on the inputs, the corresponding outputs are fed into pipeline registers for the next stage. Note, pipeline registers may also be required to pass information that may not be necessary for the next immediate stage, but some future stage.





2.1 Two diagrams are provided above. The topmost one is the standard single cycle datapath. The second is a modified version. Compare these two diagrams and explain the difference.

In the modified version, there is no wire that connects the output of the +4 block close to the PC register to the WB mux. Instead, there is an additional +4 block, which is located in the MEM stage. It takes as input the wire carrying the PC signal (extended from the wire that feeds into the 1 input of the ASel mux). The output is PC + 4, which feeds into the WB mux.

2.2 Using the modified single-cycle datapath as reference provided above, think about the information that needs to be passed along from stage to stage. Which pipeline registers are required at the end of each stage?

IF to ID:

- PC : The most adjacent stage in which the PC signal is used later on is the EX stage where PC is the input into the ASel mux.
- Inst : input into the RegFile, ImmGen, and control logic of the ID stage.

ID to EX:

- PC : input into the ASel mux
- RegReadData1 : is an input into the ASel mux.
- RegReadData2 : is an input into the BSel mux.
- Imm : is an input into the BSel mux.
- Inst : is required to compute the control logic for that particular instruction being executed in a particular stage. Therefore, the values generated by the control logic will be different in each stage depending on the input instruction. What would happen if the Inst signal was not passed along? If each stage involves a different instruction, is it correct for all stages to have the same control logic?

EX to MEM:

- PC : input into the +4 block in the MEM stage.
- ALUOut : is an input into DMEM.
- RegReadData2 : is an input into DMEM,
- Inst : input into next stage's control logic.

MEM to WB:

- PC + 4: input into WBSel mux.
- ALUOut : input into WBSel mux.
- MEM : input into WBSel mux.
- Inst : input into next stage's control logic.

$\boxed{2.3}$ Looking at the way PC is passed through the datapath, there are two places where +4 is added to the PC, once in the **IF** and **MEM** stage. Why do we add +4 to the PC again in the memory stage?

We add +4 to the PC again in the memory stage so we don't need to pass both PC and PC+4 along the whole pipeline. This would use more registers, adding unnecessary hardware. We also can't just pass only PC+4 through the pipeline, as we need the original PC value in operands like `auipc`.

3 Performance Analysis

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

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

3.1 What would be the fastest possible clock time for a single cycle datapath? Recall from last week's discussion that one instruction which exercises the critical path is `lw`.
(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 the delay in the immediate generator as well as the branch comparator are omitted because the immediate generator and branch comparison is done in parallel with the RegFile read and ALU computation respectively, the latter two taking much longer time.

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

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

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

Again, the immediate generator and branch comparator delays are overshadowed by the longer delays of RegFile read and ALU.

3.3 What is the speedup from the single cycle datapath to the pipelined datapath? Why is the speedup less than 5×?

$\frac{900 \text{ ps}}{300 \text{ ps}}$, or a 3 times speedup. The speedup is less than 5 because of (1) the necessity of adding pipeline registers, which have clk-to-q and setup times, and (2) the need to set the clock to the maximum of the five stages, which take different amounts of time.

Note: Due to hazards, which require additional logic to resolve, the actual speedup would likely be even less than 3 times.