(Re-)Introducing Control Hazards

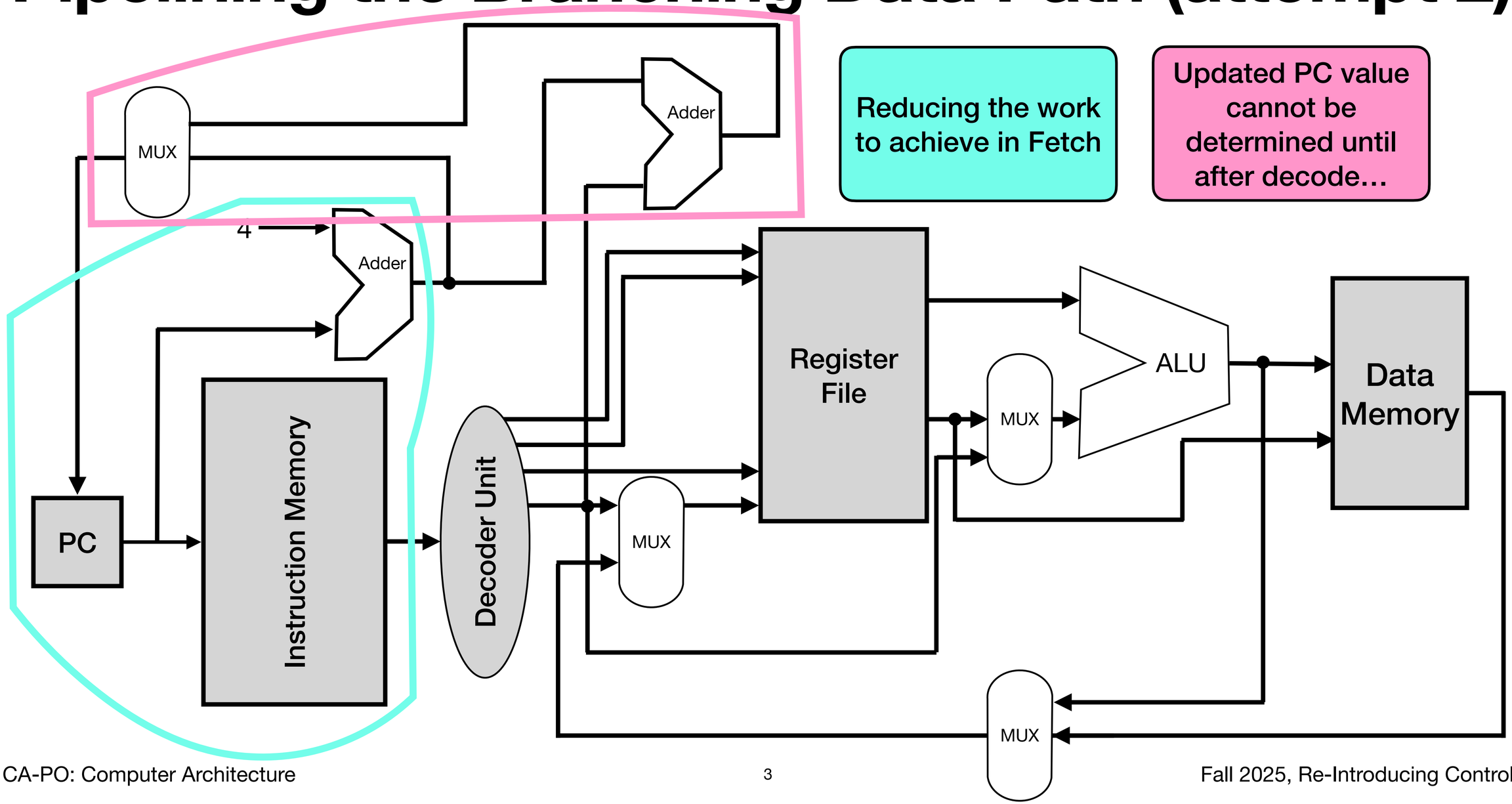
1️⃣ Colloquium today!
2️⃣ Check-In 5 in class
3️⃣ Homework 3 to be released Monday

🎃 Happy Halloween! 🎃

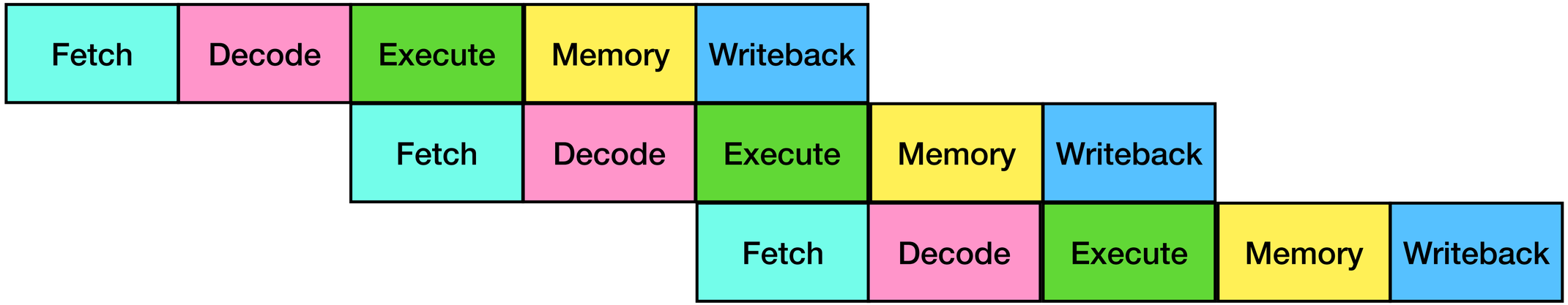
Outline

- Pipelining a data path with branch instructions
- (Re-) Introducing control hazards
- Strategies to work around control hazards

Pipelining the Branching Data Path (attempt 2)

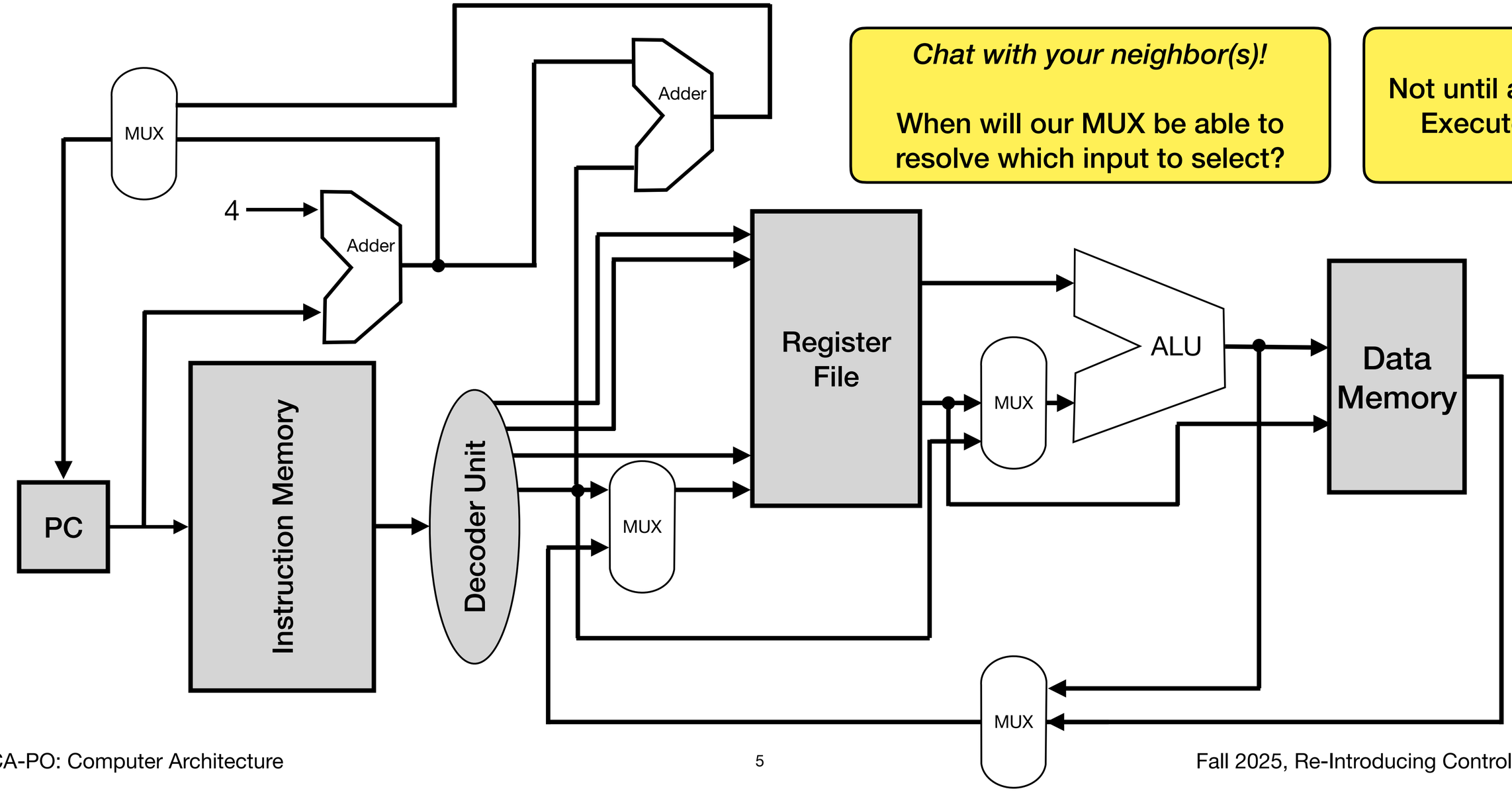


Pipelining the Branching Data Path (attempt 2)



Inefficient!
There is now a data hazard between the Decode (and compute new PC) stage and the Fetch stage on the PC register

Pipelining the Branching Data Path (attempt 3)

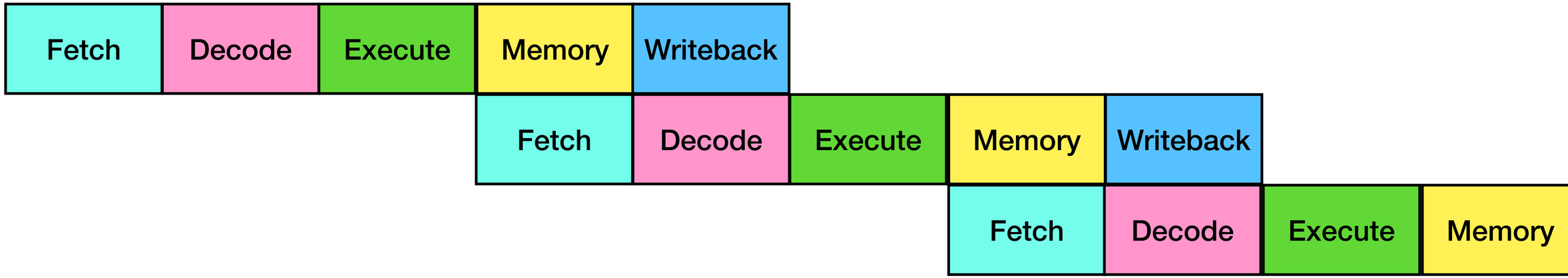


Chat with your neighbor(s)!

When will our MUX be able to resolve which input to select?

Not until after Execute!

Pipelining the Branching Data Path (attempt 3)

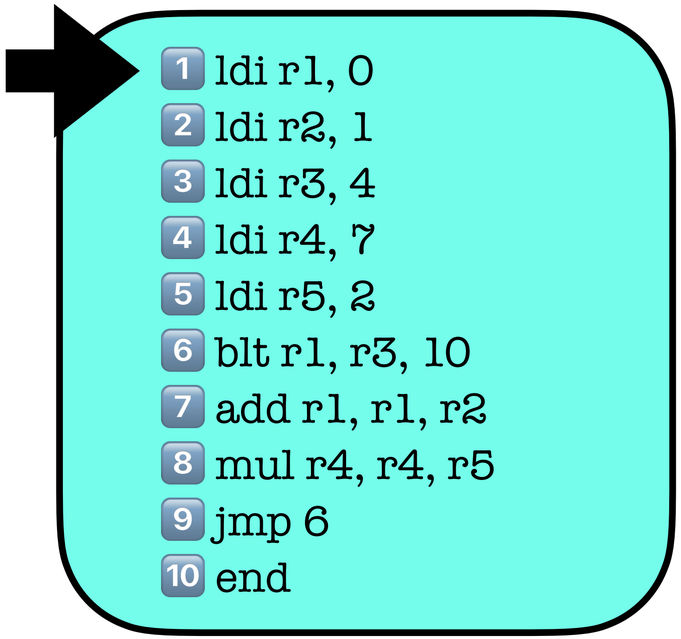


Inefficient!
The maximum achievable parallelism if we wait for the branch to be resolved is 2 instructions in the pipeline at a time!

Control Hazards

- If it takes several cycles to know what the appropriate next program counter value should be, then it may be the case that our processor executes instructions that are incorrect relative to the expected program behavior
- Executing instructions on the incorrect side of a branch is called a *control hazard* as it will lead to incorrect instructions in the pipeline
- If our processor implements a hazard checking unit, then the unit must also check to see if incorrect instructions are in the pipeline due to control hazards and appropriately stall/bubble the stages

Strategies to Workaround Control Hazards



We don’t know the answer!

We could wait or guess!

What to Fetch Next

- Seeing as we will not know the *true value* of the PC until after the branch is evaluated (which could be as late as after Execute), it may seem as though we have to delay our next fetch until we know the PC
- Alternatively, we can try optimistically making an *assumption* or *prediction* about what the next PC will be and correct ourselves later if we were wrong (we will talk about how to do this next week)
- For example, we could *assume* that the instruction is always either not a branch or that the branch is *not taken* whenever we reach a control instruction
- Alternatively, we could track certain behaviors to try to *predict* whether the branch was or was not taken

Takeaways

- By adding control instructions, our programs can become more robust but they also add complexity to the underlying hardware
- Updating the control flow requires new hardware logic to update the PC and pipelining logic must change accordingly
- By updating the pipeline, we introduce control hazards that must be mitigated