Constructing a Data Path for Control Instructions

HW3 (Part 1) due tonight;
Check In 5 on Friday

Image credit: https://www.tripadvisor.com/Attraction_Review-g29125-d1519659-Reviews-Bridge_To_Nowhere-Azusa_California.html

Happy April Fools Day 😅

Image credit: https://www.chesapeakebaymagazine.com/ntsb-johns-hopkins-engineers-call-for-ship-strike-evaluations-on-vulnerable-bay-bridge/

Outline

- Extending the data path for control instructions
- Handling “offset” based branch targets
- Pipelining the control-instruction data path

Case Study: Branching in RISC-V

Software

Conditional Jumps

Unconditional Jumps

31 ... 25	24 ... 20	19 ... 15	14 ... 12	11 ... 7	6 ... 0	
funct7	rs2	rs1	funct3	rd	opcode	R-type
imm[11:0]		rs1	funct3	rd	opcode	I-type
imm[11:5]	rs2	rs1	funct3	imm[4:0]	opcode	S-type
imm[12|10:5]	rs2	rs1	funct3	imm[4:1|11]	opcode	B-type
imm[31:12]				rd	opcode	U-type
imm[20|10:1|11|19:12]				rd	opcode	J-type

What goes in the immediate field?

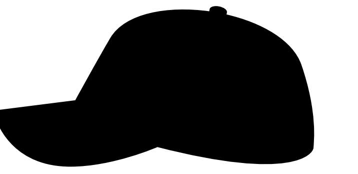
1️⃣ bne r1, r2, 0xff00
2️⃣ blt r1, r2, 0xff00
"if-statements" and loops

1️⃣ jmp 0xff00
2️⃣ jalr
function calls and returns

Direct or Indirect Offset

Updating the Data Path

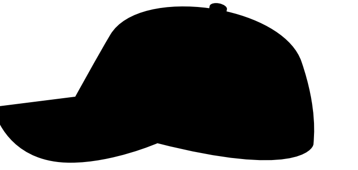
How do we update our data path for unconditional jumps with direct targets?



Hardware

MUX
4
Adder
PC
Instruction Memory
Decoder Unit
MUX
Register File
MUX
ALU
MUX
Data Memory
MUX

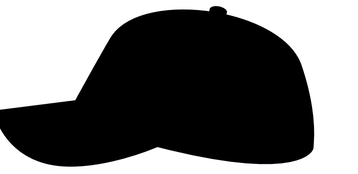
Pitfalls of Direct Targets



Hardware

- The maximum immediate value that we can encode in an unconditional jump is 2^{20}… if this is the maximum value of a direct target, then we cannot have programs with binaries larger than 1MB
- To jump further than 2^{20} using direct branch targets, we can “chain” multiple unconditional jumps together in the compilation of the binary! That is, we jump to a jump until we get to the instruction that we want
  - ❌ adds lots of complexity to the compiler
  - ❌ different instructions to call the same function…
- In practice most architectures use indirect branch targets, so we need to include additional components in the data path to perform this computation!

Computing Indirect Branch Targets



Hardware

How do we update our data path for unconditional jumps with indirect targets?

What should the offset in the instruction be?

MUX

4

Adder

Adder

PC

Instruction Memory

Decoder Unit

MUX

Register File

MUX

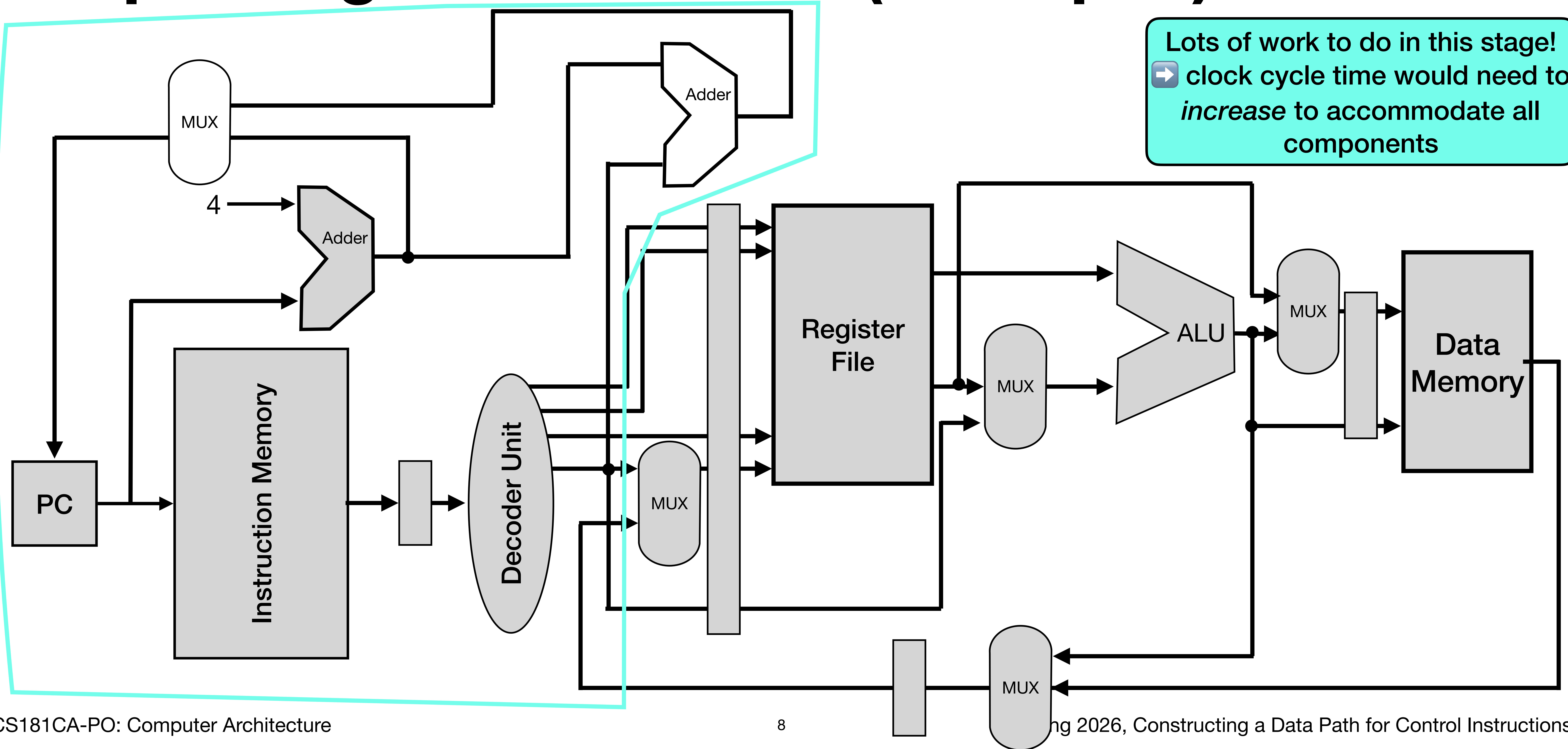
ALU

MUX

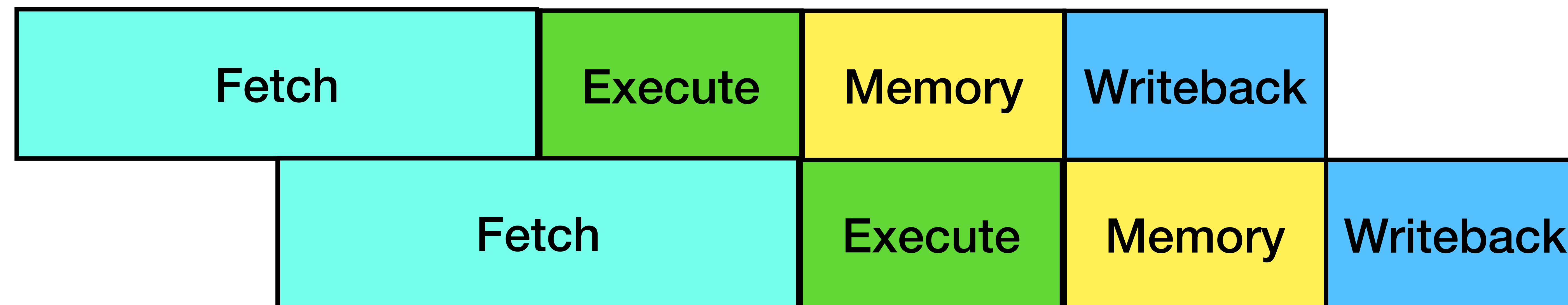
Data Memory

MUX

Pipelining the Data Path (attempt 1)

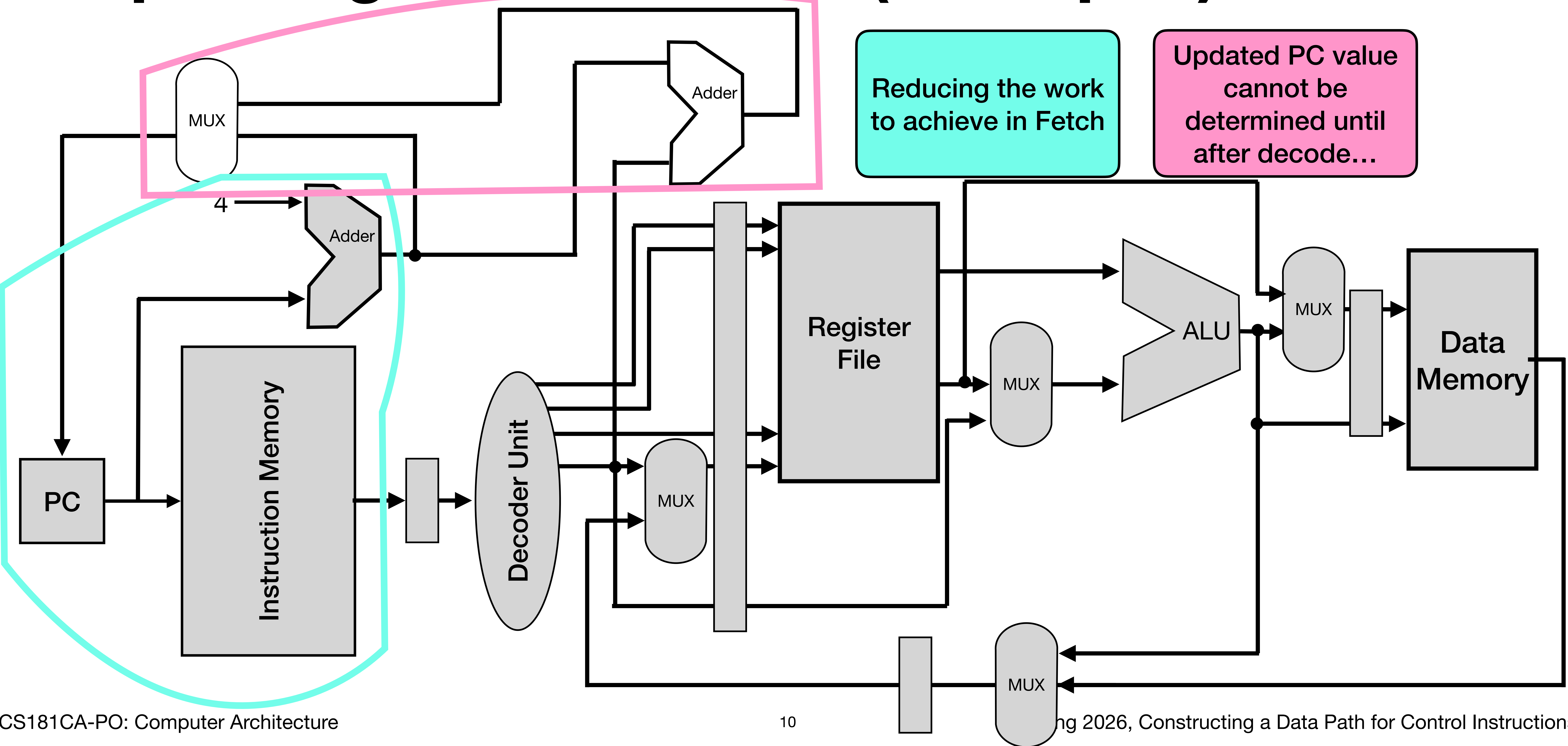


Pipelining the Data Path (attempt 1)



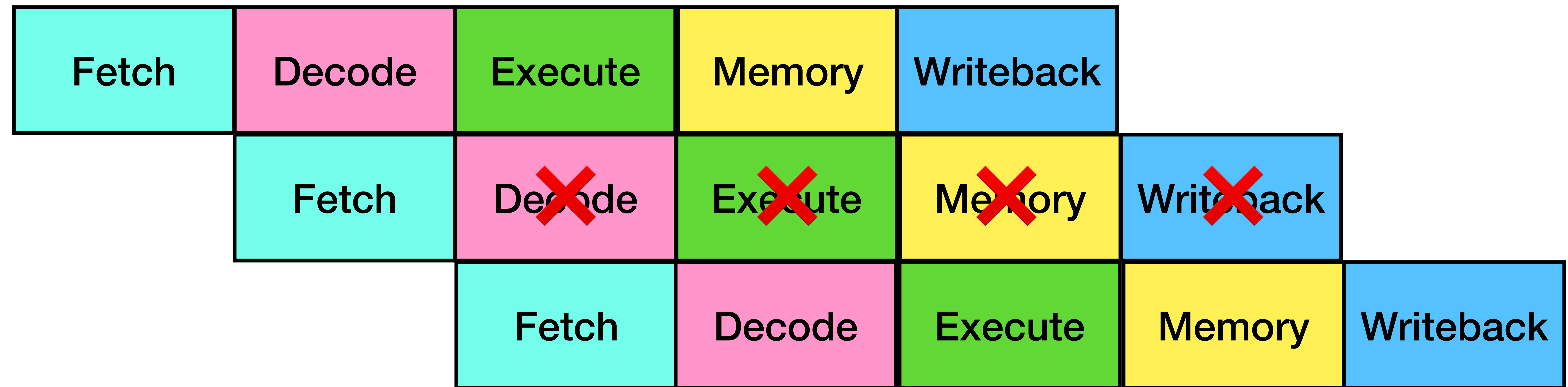
We cannot we start the memory operation for instruction 2 at this time. Why not?

Pipelining the Data Path (attempt 2)



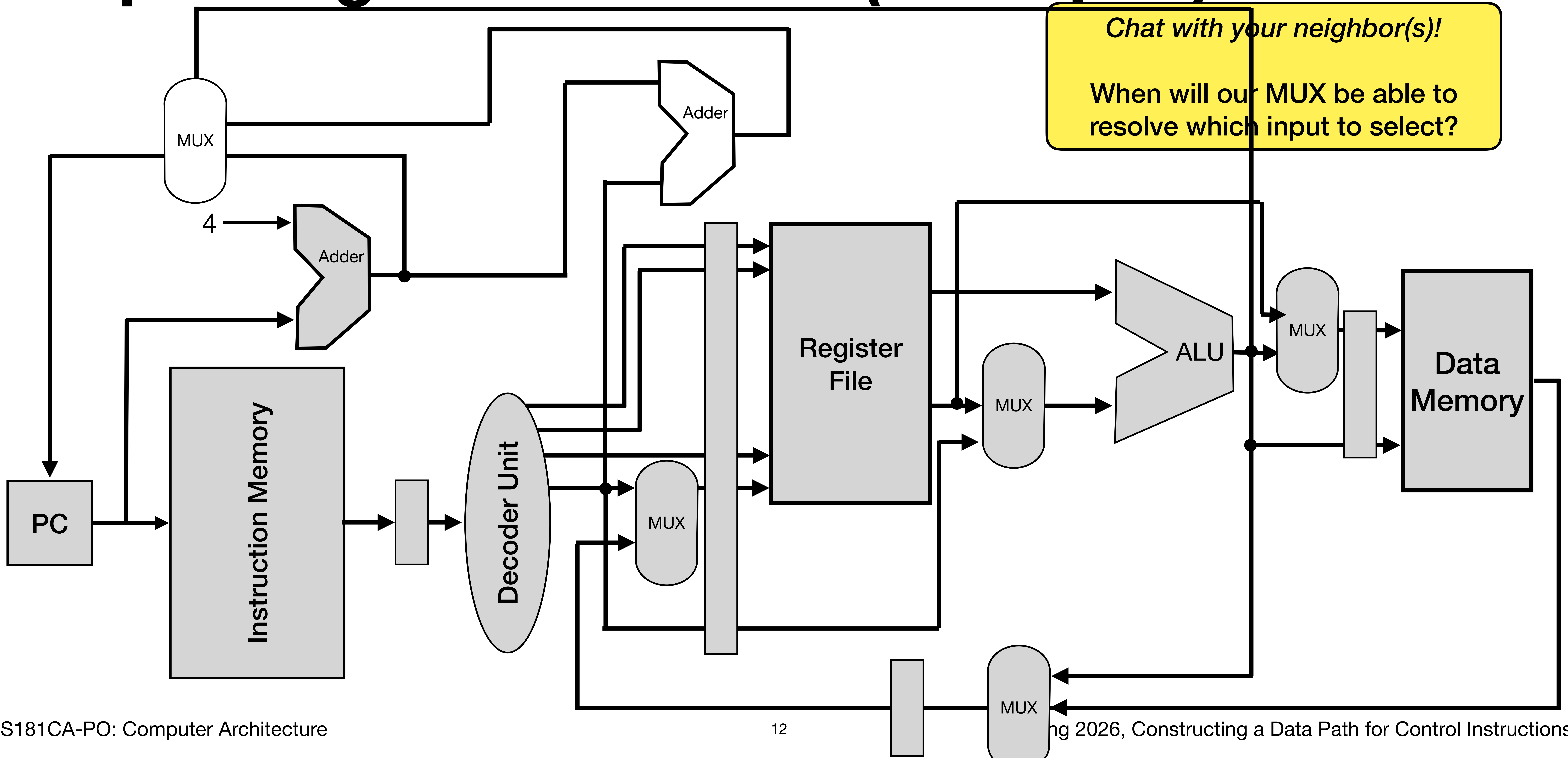
Pipelining the Data Path (attempt 2)

1️⃣ jmp 1
2️⃣ ldi r2, 30
3️⃣ end

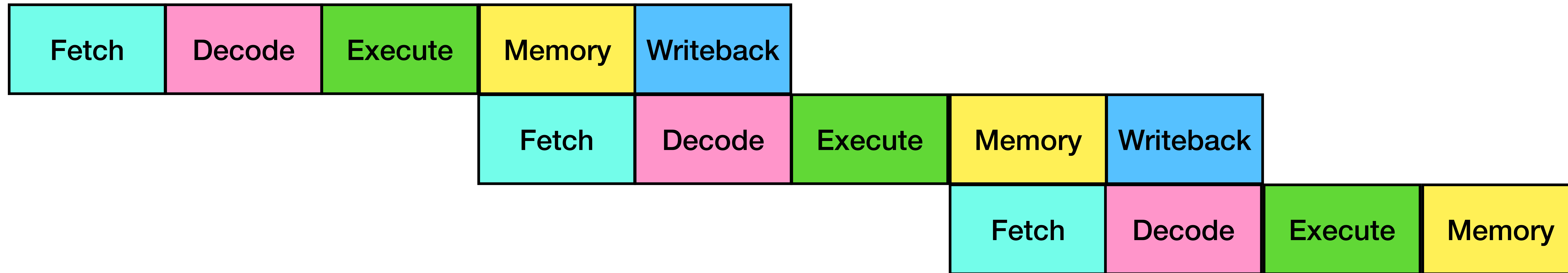


Takeaway: Inefficient!
There is now a control hazard in our data path… how should this be resolved?

Pipelining the Data Path (attempt 3)

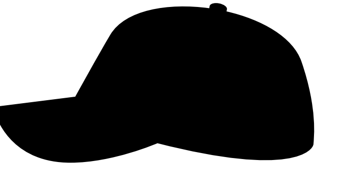


Pipelining the Branching Data Path (attempt 3)



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

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