CPSC 313: Computer Hardware and Operating Systems

Unit 2: Pipelining

Pipelining: Hazards

Administration

- Quiz 1 retakes happening until Saturday.
- Quiz 2:
  - Registration is open
  - Info/Practice available at 17:00 on October 7th.
- Lab 3: Make lots of progress on this; it's challenging!
- Lab 4: Releases Friday (not as challenging)

Today

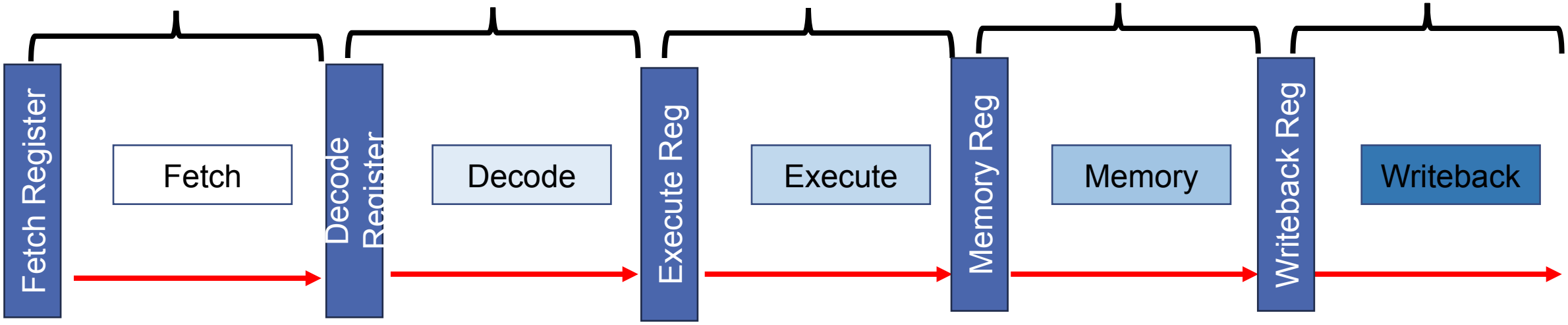
- Learning outcomes
  - Precisely describe instructions for the y86 pipelined implementation
  - Identify data hazards
  - Examine various mechanisms to deal with them.

Describing Instruction Execution: SEQ

MRMOVQ

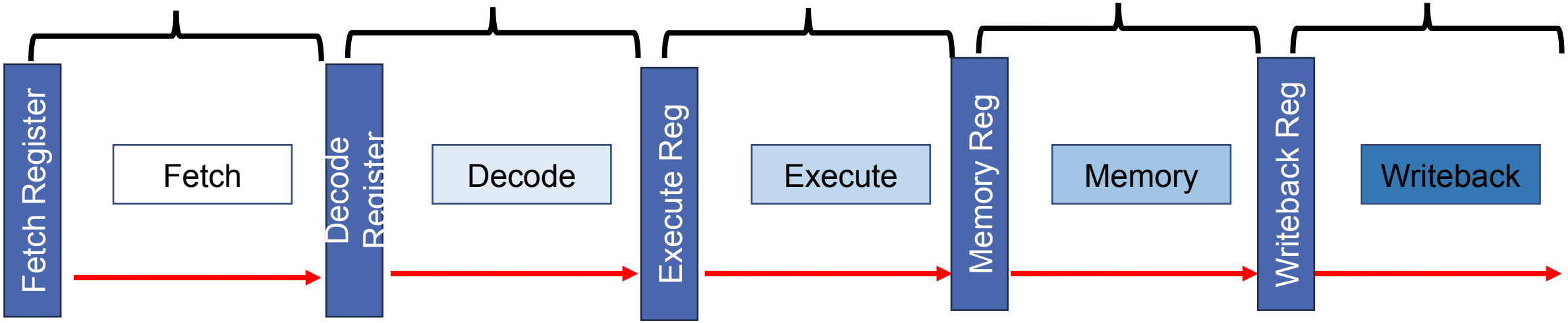
icode:ifun = M_1[PC] rA:rB = M_1[PC+1] valC = M_8[PC+2] valP = PC + 10	FETCH
valB = R[rB]	DECODE
valE = valB + valC	EXECUTE
valM = M_8[valE]	MEMORY
R[rA] = valM calcPC = valP	WRITEBACK

Pipeline Execution Cartoon Style



1. Pipeline registers hold values for their stage
2. Stages execute in parallel
3. Signals flow from the pipeline register and do not enter the next pipeline register until the clock ticks

Pipeline Execution Cartoon Style



During a stage, we use the contents of that stage's pipeline register to compute the values that will go into the next stage's pipeline register.
E.g., the Memory stage takes the contents of the Memory register as INPUT and outputs contents that go into the Writeback register.

Assume R[%rax] = 0xCAD
Assume R[%rbp] = 0x1000

Describing Instruction Execution: PIPE

Values in red are specific to this instruction and input values

MRMOVQ

icode:ifun = M_1[PC] rA:rB = M_1[PC+1] valC = M_8[PC+2] valP = PC + 10	FETCH
valB = R[rB]	DECODE
valE = valB + valC	EXECUTE
valM = M_8[valE]	MEMORY
R[rA] = valM calcPC = valP	WRITEBACK

MRMOVQ 8(%rbp), %rax

D_icode:ifun = M_1[PC] = 5:0 **D_rA:rB** = M_1[PC+1] = 0:5 **F_calcPC = valP** = 0xA	**D_valC** = M_8[PC+2] = 8 **D_valP** = PC + 10 = 0xA
E_icode = **E_ifun** = **E_valC** = **E_valA** =	**E_valB** = **E_dstE** = **E_dstM** =
M_icode = **M_Cnd** = **M_valE** =	**M_valA** = **M_dstE** = **M_dstM** =
W_icode = **W_valE** = **W_valM** = **W_dstE** = **W_dstM** =	**W_Cnd** =
R[W_dstE] = **R[W_dstM]** =	

UPPERCASE_ indicates this is the stage's input.

Note: We will skim this, not discuss every value!

Assume R[%rax] = 0xCAD
Assume R[%rbp] = 0x1000

Describing Instruction Execution: PIPE

Values in red are specific to this instruction and input values

MRMOVQ

icode:ifun = M_1[PC] rA:rB = M_1[PC+1] valC = M_8[PC+2] valP = PC + 10	FETCH
valB = R[rB]	DECODE
valE = valB + valC	EXECUTE
valM = M_8[valE]	MEMORY
R[rA] = valM calcPC = valP	WRITEBACK

MRMOVQ 8(%rbp), %rax

D_icode:ifun = M_1[PC] = 5:0 **D_rA:rB** = M_1[PC+1] = 0:5 **F_calcPC = valP** = 0xA	**D_valC** = M_8[PC+2] = 8 **D_valP** = PC + 10 = 0xA
E_icode = D_icode = 5 **E_ifun** = D_ifun = 0 **E_valC** = D_valC = 8 **E_valA** = *Don't Care* = *<none>*	**E_valB** = *R[rB]* = *0x1000* **E_dstE** = *F* **E_dstM** = *rA* = *0*
M_icode = **M_Cnd** = **M_valE** =	**M_valA** = **M_dstE** = **M_dstM** =
W_icode = **W_valE** = **W_valM** = **W_dstE** = **W_dstM** =	**W_Cnd** =
R[W_dstE] = **R[W_dstM]** =	

Assume R[%rax] = 0xCAD
Assume R[%rbp] = 0x1000

Describing Instruction Execution: PIPE

Values in red are specific to this instruction and input values

MRMOVQ

icode:ifun = M_1[PC] rA:rB = M_1[PC+1] valC = M_8[PC+2] valP = PC + 10	FETCH
valB = R[rB]	DECODE
valE = valB + valC	EXECUTE
valM = M_8[valE]	MEMORY
R[rA] = valM calcPC = valP	WRITEBACK

MRMOVQ 8(%rbp), %rax

D_icode:ifun = M_1[PC] = 5:0 **D_rA:rB** = M_1[PC+1] = 0:5 **F_calcPC = valP** = 0xA	**D_valC** = M_8[PC+2] = 8 **D_valP** = PC + 10 = 0xA
E_icode = D_icode = 5 **E_ifun** = D_ifun = 0 **E_valC** = D_valC = 8 **E_valA** = Don't Care = <none>	**E_valB** = R[rB] = 0x1000 **E_dstE** = F **E_dstM** = rA = 0
M_icode = E_icode = 5 **M_Cnd** = *Cond(CC, E_ifun)* **M_valE** = *E_valC + E_valB* = *0x1008*	**M_valA** = E_valA = <none> **M_dstE** = E_dstE = F **M_dstM** = E_dstM = 0
W_icode = **W_valE** = **W_valM** = **W_dstE** = **W_dstM** =	**W_Cnd** =
R[W_dstE] = **R[W_dstM]** =	

Assume R[%rax] = 0xCAD
Assume R[%rbp] = 0x1000

Describing Instruction Execution: PIPE

Values in red are specific to this instruction and input values

MRMOVQ

icode:ifun = M_1[PC] rA:rB = M_1[PC+1] valC = M_8[PC+2] valP = PC + 10	FETCH
valB = R[rB]	DECODE
valE = valB + valC	EXECUTE
valM = M_8[valE]	MEMORY
R[rA] = valM calcPC = valP	WRITEBACK

MRMOVQ 8(%rbp), %rax

D_icode:ifun = M_1[PC] = 5:0 **D_rA:rB** = M_1[PC+1] = 0:5 **F_calcPC = valP** = 0xA	**D_valC** = M_8[PC+2] = 8 **D_valP** = PC + 10 = 0xA
E_icode = D_icode = 5 **E_ifun** = D_ifun = 0 **E_valC** = D_valC = 8 **E_valA** = Don't Care = <none>	**E_valB** = R[rB] = 0x1000 **E_dstE** = F **E_dstM** = **rA** = 0
M_icode = E_icode = 5 **M_Cnd** = Cond(CC, E_ifun) **M_valE** = E_valC + E_valB = 0x1008	**M_valA** = E_valA = <none> **M_dstE** = E_dstE = F **M_dstM** = E_dstM = 0
W_icode = M_icode = 5 **W_valE** = M_valE = 0x1008 **W_valM** = *M_8[M_valE] = M_8[0x1008]* **W_dstE** = M_dstE = F **W_dstM** = M_dstM = 0	**W_Cnd** = M_Cnd
R[W_dstE] = **R[W_dstM]** =	

Assume R[%rax] = 0xCAD
Assume R[%rbp] = 0x1000

Describing Instruction Execution: PIPE

Values in red are specific to this instruction and input values

MRMOVQ

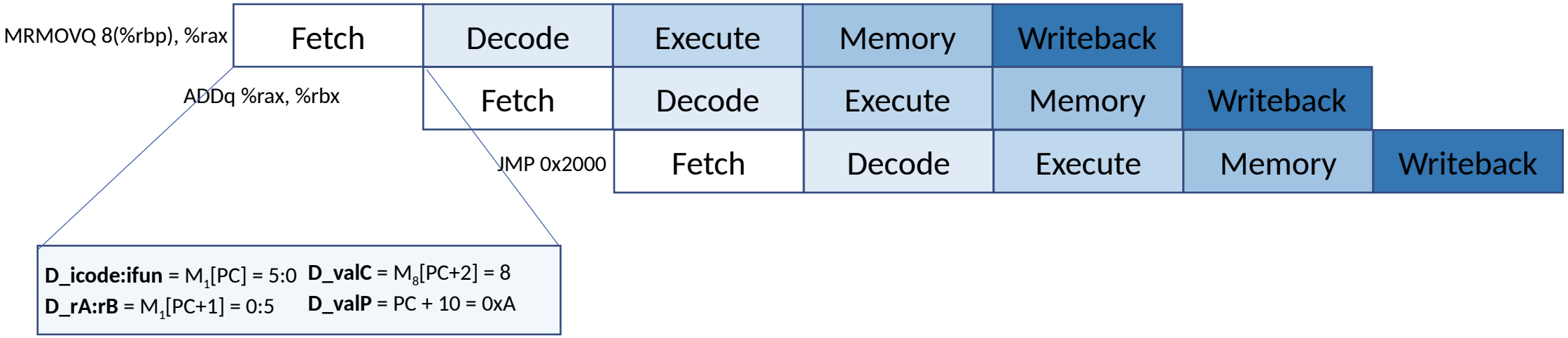
icode:ifun = M_1[PC] rA:rB = M_1[PC+1] valC = M_8[PC+2] valP = PC + 10	FETCH
valB = R[rB]	DECODE
valE = valB + valC	EXECUTE
valM = M_8[valE]	MEMORY
R[rA] = valM calcPC = valP	WRITEBACK

MRMOVQ 8(%rbp), %rax

D_icode:ifun = M_1[PC] = 5:0 **D_rA:rB** = M_1[PC+1] = 0:5 **F_calcPC = valP** = 0xA	**D_valC** = M_8[PC+2] = 8 **D_valP** = PC + 10 = 0xA
E_icode = D_icode = 5 **E_ifun** = D_ifun = 0 **E_valC** = D_valC = 8 **E_valA** = Don't Care = <none>	**E_valB** = R[rB] = 0x1000 **E_dstE** = F **E_dstM** = rA = 0
M_icode = E_icode = 5 **M_Cnd** = Cond(CC, E_ifun) **M_valE** = E_valC + E_valB = 0x1008	**M_valA** = E_valA = <none> **M_dstE** = E_dstE = F **M_dstM** = E_dstM = 0
W_icode = M_icode = 5 **W_valE** = M_valE = 0x1008 **W_valM** = M_8[M_valE] = M_8[0x1008] **W_dstE** = M_dstE = F **W_dstM** = M_dstM = 0	**W_Cnd** = M_Cnd
R[W_dstE] = **R[W_dstM] =** *W_valM* = M_8[0x1008]	

What can go Wrong? Hazards

Assume R[%rax] = 0xCAD
Assume R[%rbp] = 0x1000
Assume R[%rbx] = 0x4
Assume M8[0x1008] = 0x44

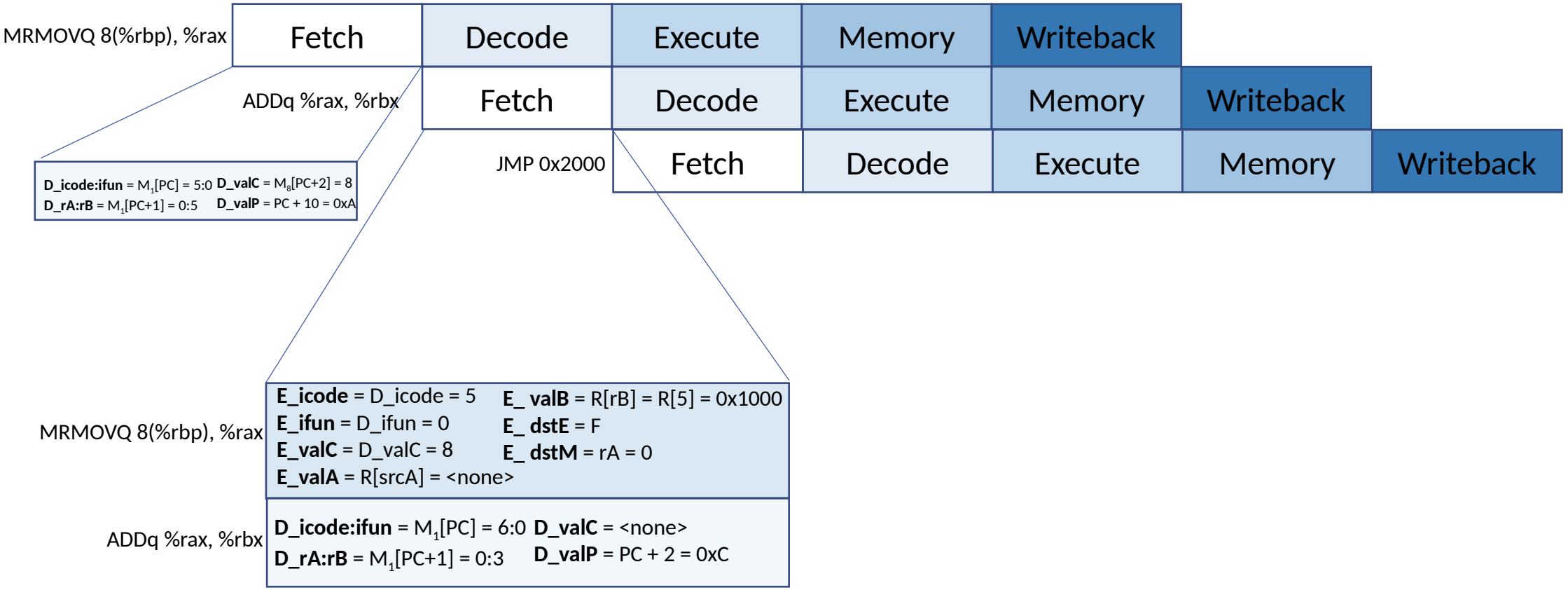


Consider the following sequence of instructions:
MRMOVQ 8(%rbp), %rax
ADDQ %rax, %rbx
JMP 0x2000

Hazards

Consider the following sequence of instructions:
MRMOVQ 8(%rbp), %rax
ADDQ %rax, %rbx
JMP 0x2000

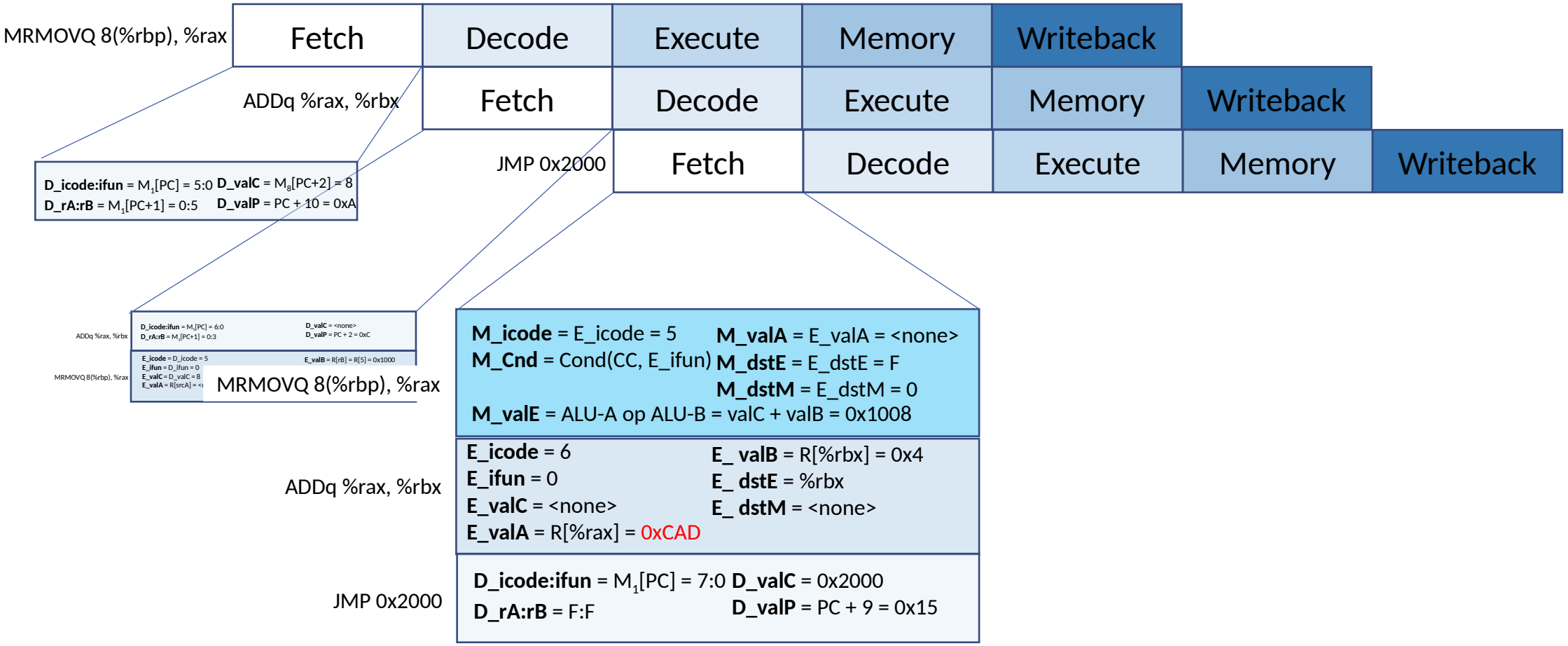
Assume R[%rax] = 0xCAD
Assume R[%rbp] = 0x1000
Assume R[%rbx] = 0x4
Assume M8[0x1008] = 0x44



Hazards

Consider the following sequence of instructions:
MRMOVQ 8(%rbp), %rax
ADDQ %rax, %rbx
JMP 0x2000

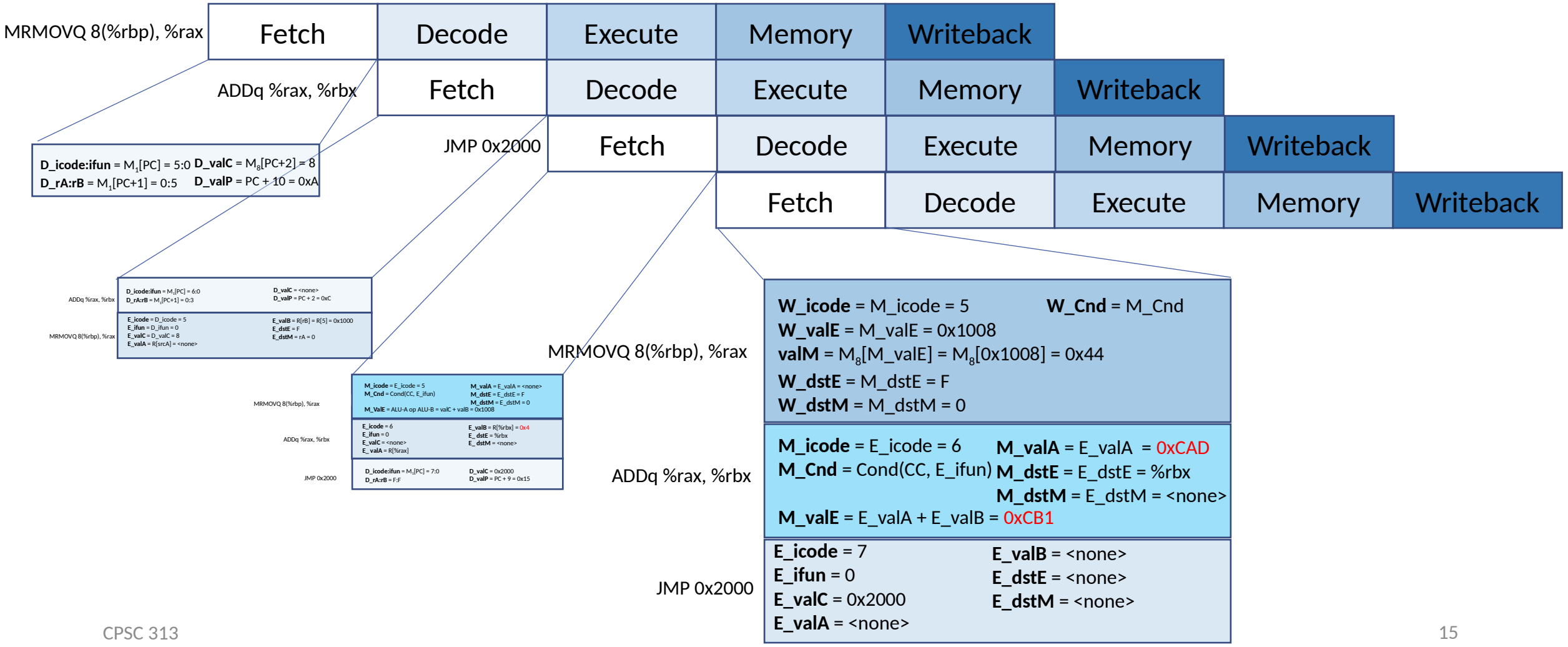
Assume R[%rax] = 0xCAD
Assume R[%rbp] = 0x1000
Assume R[%rbx] = 0x4
Assume M8[0x1008] = 0x44



Hazards

Consider the following sequence of instructions:
MRMOVQ 8(%rbp), %rax
ADDQ %rax, %rbx
JMP 0x2000

Assume R[%rax] = 0xCAD
Assume R[%rbp] = 0x1000
Assume R[%rbx] = 0x4
Assume M8[0x1008] = 0x44



Hazard (definition)

- **Hazards** are problems that arise in pipelining when information needed by an instruction is not available when it needs it.
  - Unless mitigated, hazards can lead to incorrect results
- Types of hazards:
  - **Data hazard**: When data dependencies (e.g., one instruction reads from a register that another one writes) potentially lead to incorrect behavior.
  - **Control hazard**: When control dependencies (e.g., a conditional jump determines the address of the next instruction to execute) potentially lead to incorrect behavior.

Approaches to Hazard Handling

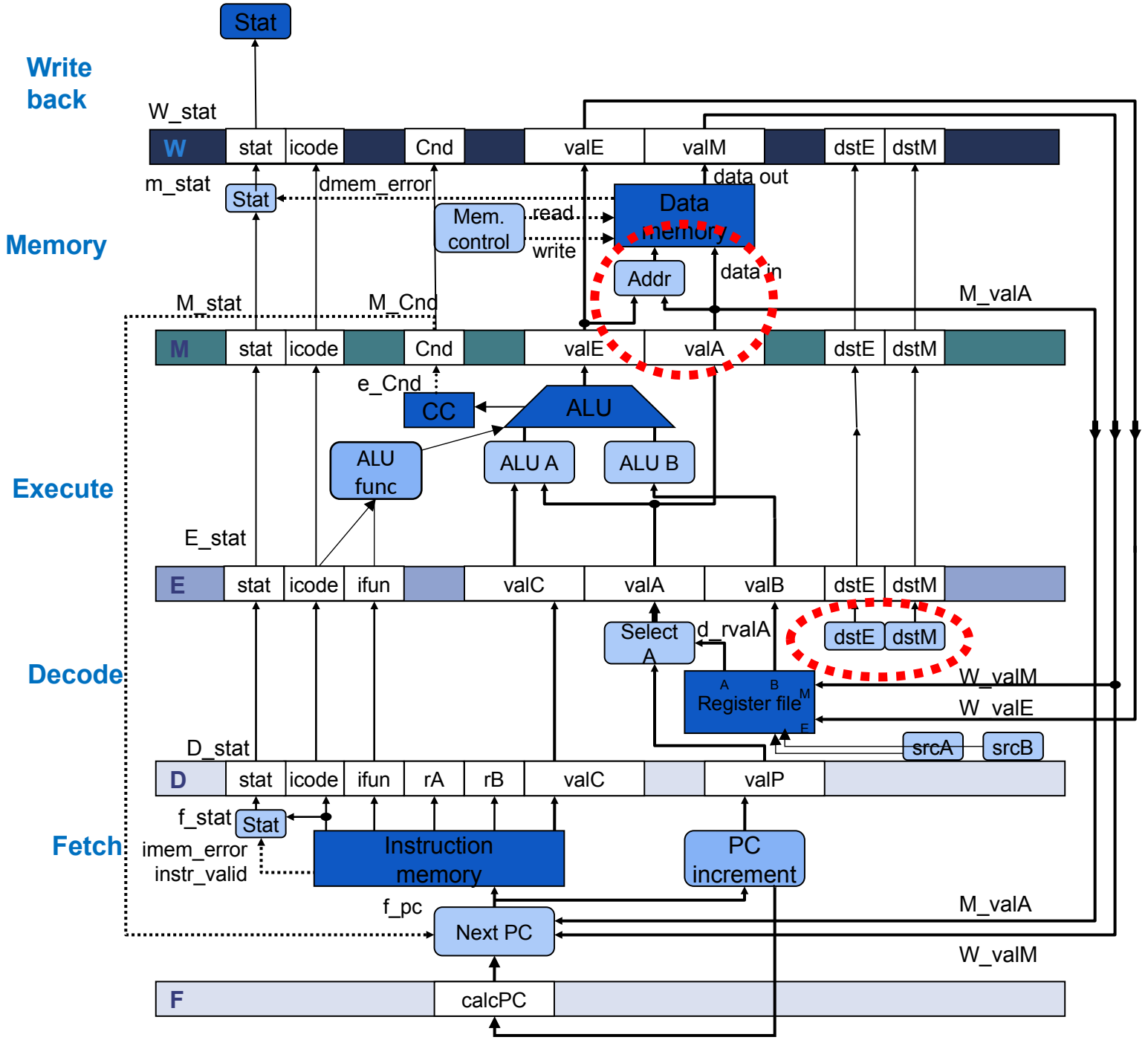
- Pure software (today)
- Delaying execution in hardware (stalling) (next Pre-class)
- Value forwarding in hardware (addresses data hazards) (next class)
- Branch prediction (addresses [some] control hazards) (class after that one)

Y86 Pipeline

Public Service Announcements:

- This is an abstraction; details are missing! (E.g., what are `dstE` and `dstM`'s inputs?)
- There are some small changes from the sequential version. Especially:

  For `popq` (and `ret`), we place `%rsp`'s value in both `valA` and `valB`.

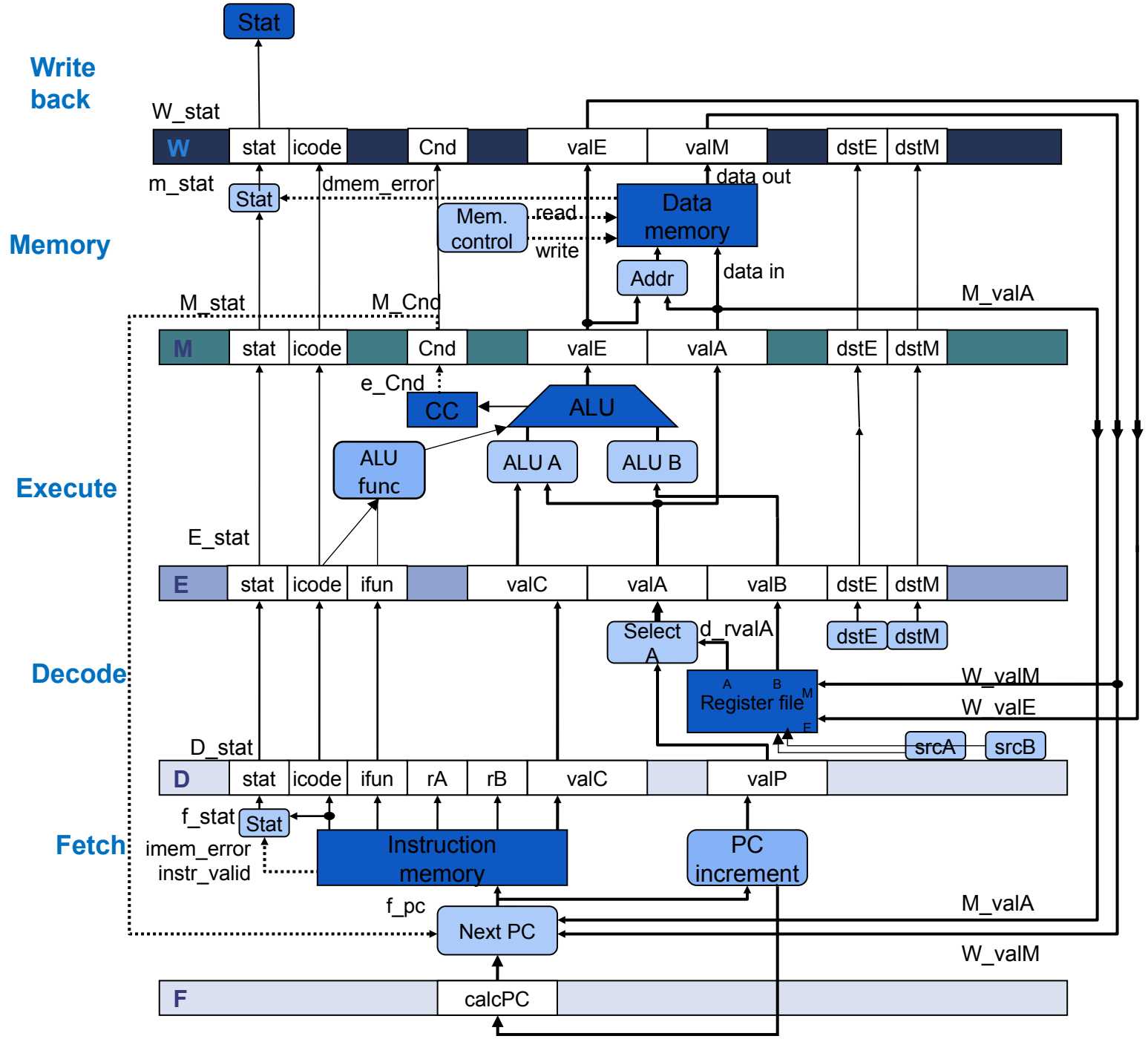


Y86 Pipeline

addq %rax, %rbx
subq %rbx, %rcx

Assume that the addq enters the pipeline at time T = 1, and the subq enters the pipeline at time T+1 = 2.

Q1: During what stage of the subq do we need the value of %rbx?



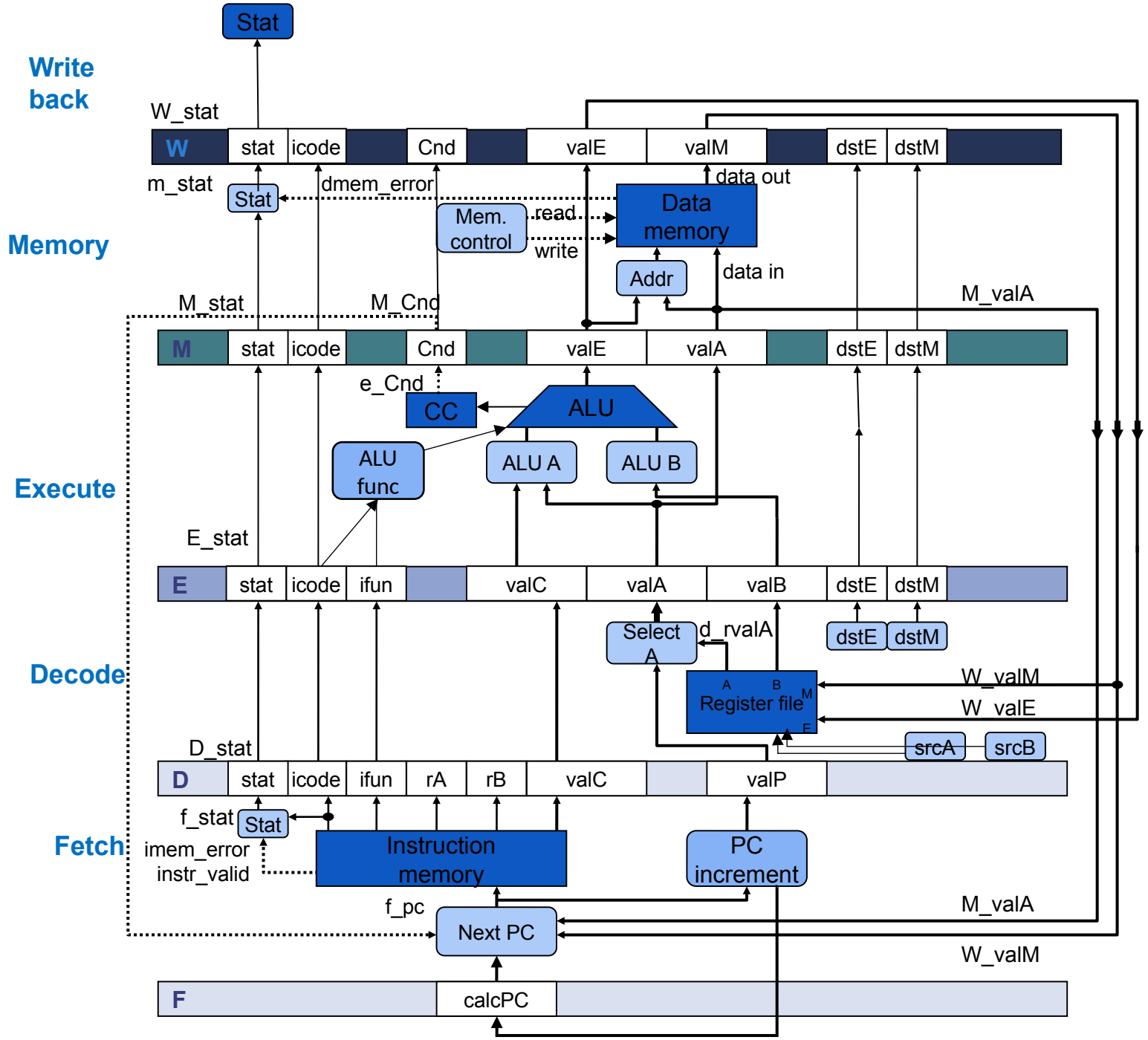
Y86 Pipeline

addq %rax, %rbx
subq %rbx, %rcx

Assume that the addq enters the pipeline at time T = 1, and the subq enters the pipeline at time T+1 = 2.

Q1: During what stage of the subq do we need the value of %rbx?
We need it in the subq's Decode (when the addq is in the Execute stage) (T=3)

Q2: During what stage would the addq normally write its new value into %rbx?



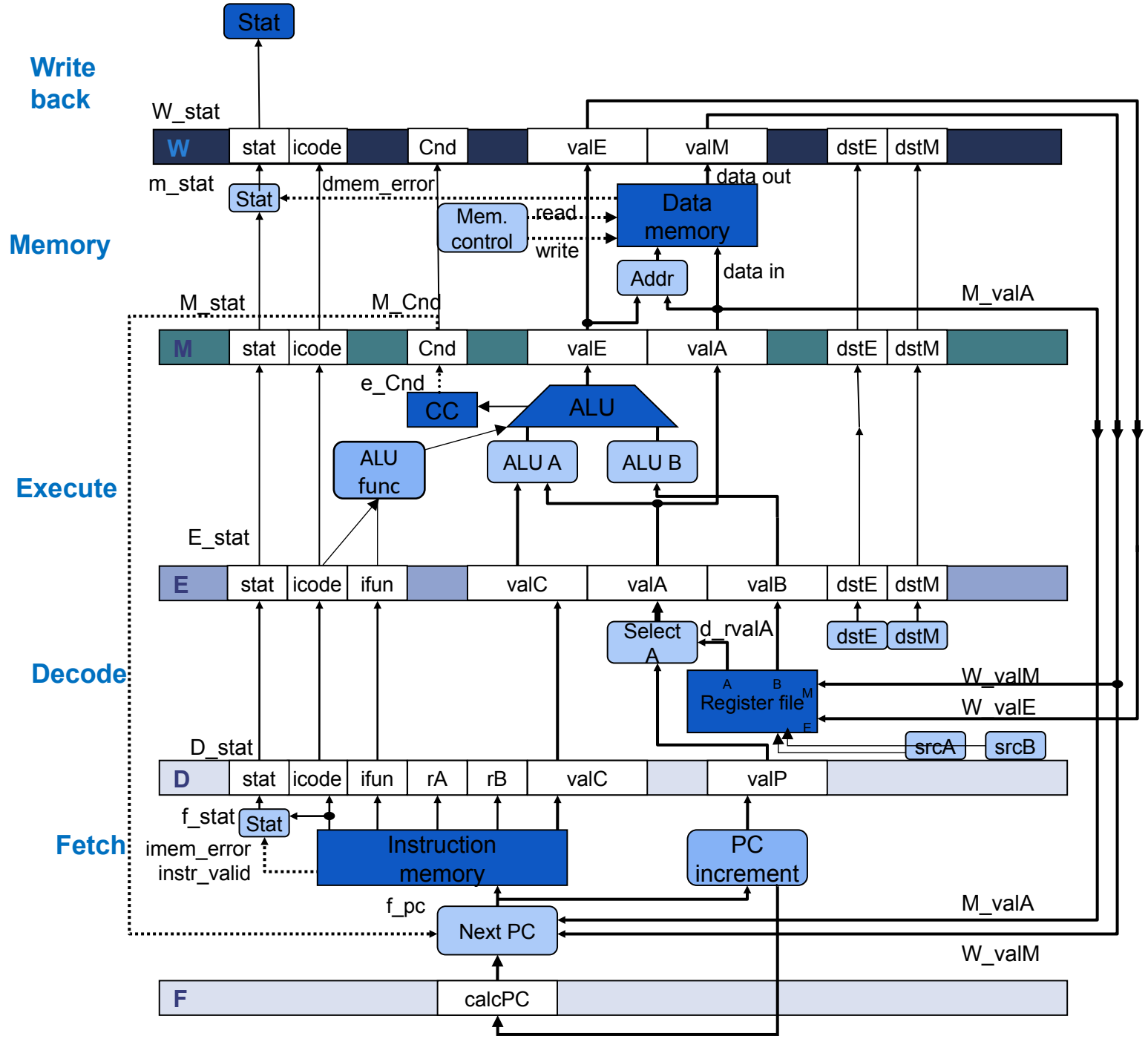
Y86 Pipeline

addq %rax, %rbx
subq %rbx, %rcx

Assume that the addq enters the pipeline at time T = 1, and the subq enters the pipeline at time T+1 = 2.

Q1: During what stage of the subq do we need the value of %rbx?
We need it in the subq's Decode (when the addq is in the Execute stage) (T=3)

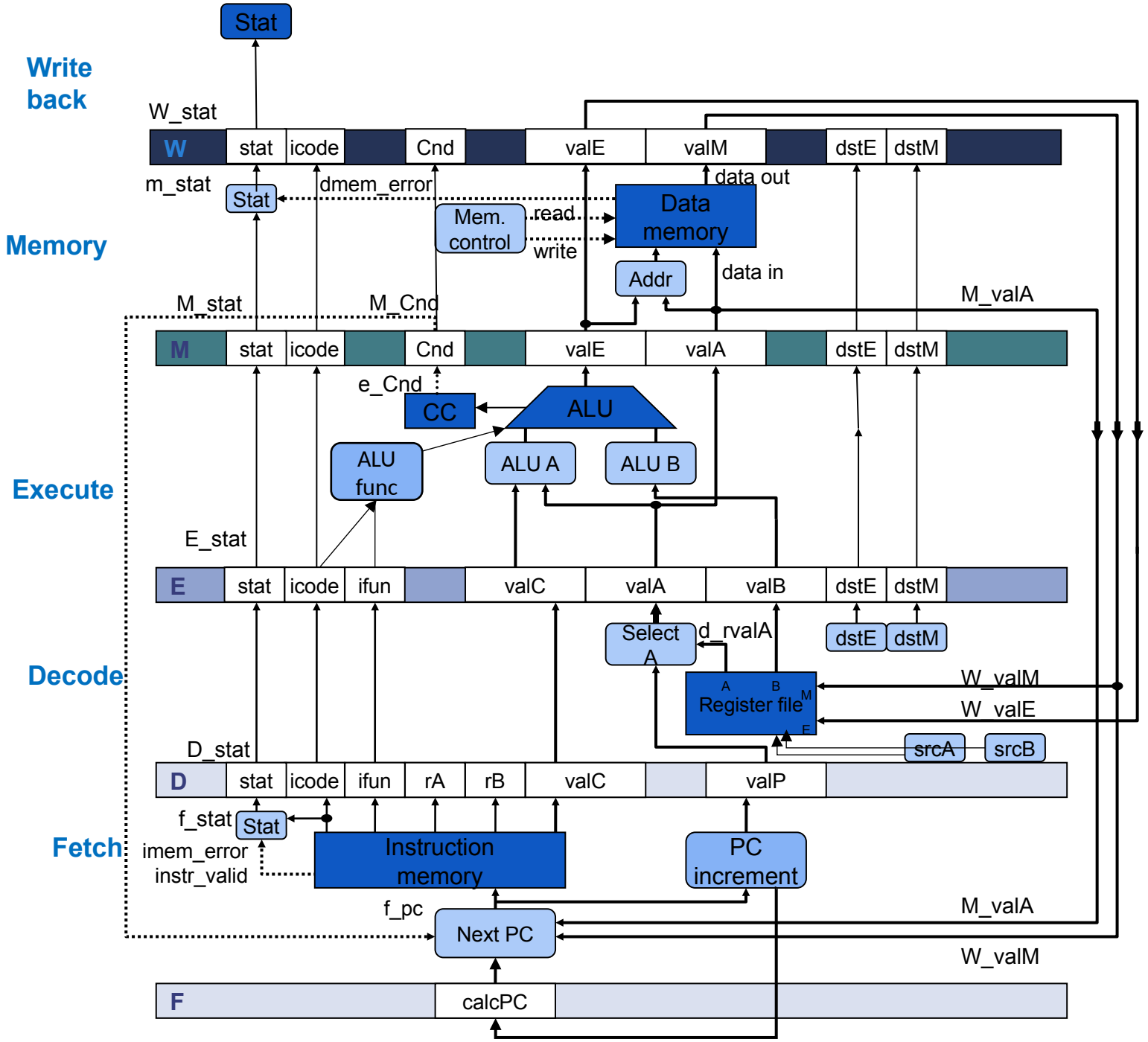
Q2: During what stage would the addq normally write its new value into %rbx?
The writeback stage: T+4 = 5



Solution 1

Manual: Insert nops.

Q3: How many will we need?

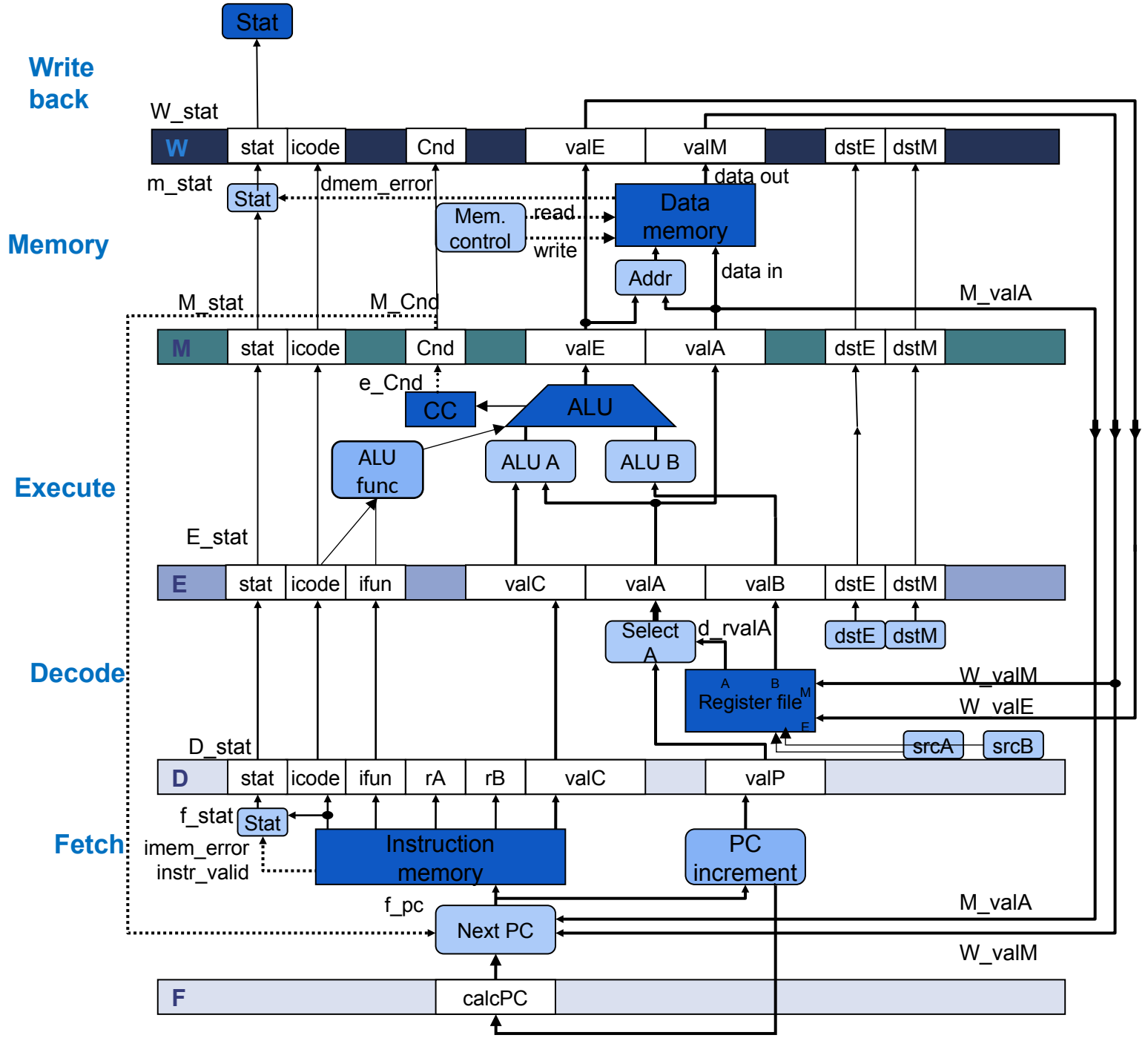


Solution 1

Manual: Insert nops.

Q3: How many will we need?

(T + 4 + 1) – (3) = 6 – 3 = 3



In-class activity

- You will need to do at least 5 examples to earn 100%
  - The more examples you do, the deeper will be your understanding
- Learning experts argue about whether or not it really takes 10,000 hours of practice to become an expert
  - But the number isn't 0 or 1, either!

Wrapping up

- Instructions move through the pipeline in lockstep
- **Hazards** occur when an instruction in the pipeline needs information that is not yet available.
  - **Data hazards**: an instruction reads data that an earlier, not-yet-retired instruction wrote; without some kind of mitigation it would be possible to read the wrong value.
  - **Control hazard**: an instruction in the Fetch stage does not yet know the address of the next instruction to execute.
- First mitigation technique: **insert nops** to avoid hazards.