CPSC 313: Computer Hardware and Operating Systems

Unit 2: Pipelining

Pipelining: Forwarding

Administration

- Quiz 1 retakes happening until Saturday.
- Quiz 2:
  - Registration is open
  - Info/Practice available at 17:00 on October 7th.
- Lab 3: due Sunday
- Lab 4: Released today

Today

- Learning outcomes
  - Identify places in the implementation where we can forward signals to avoid stalling the processor.

Recall

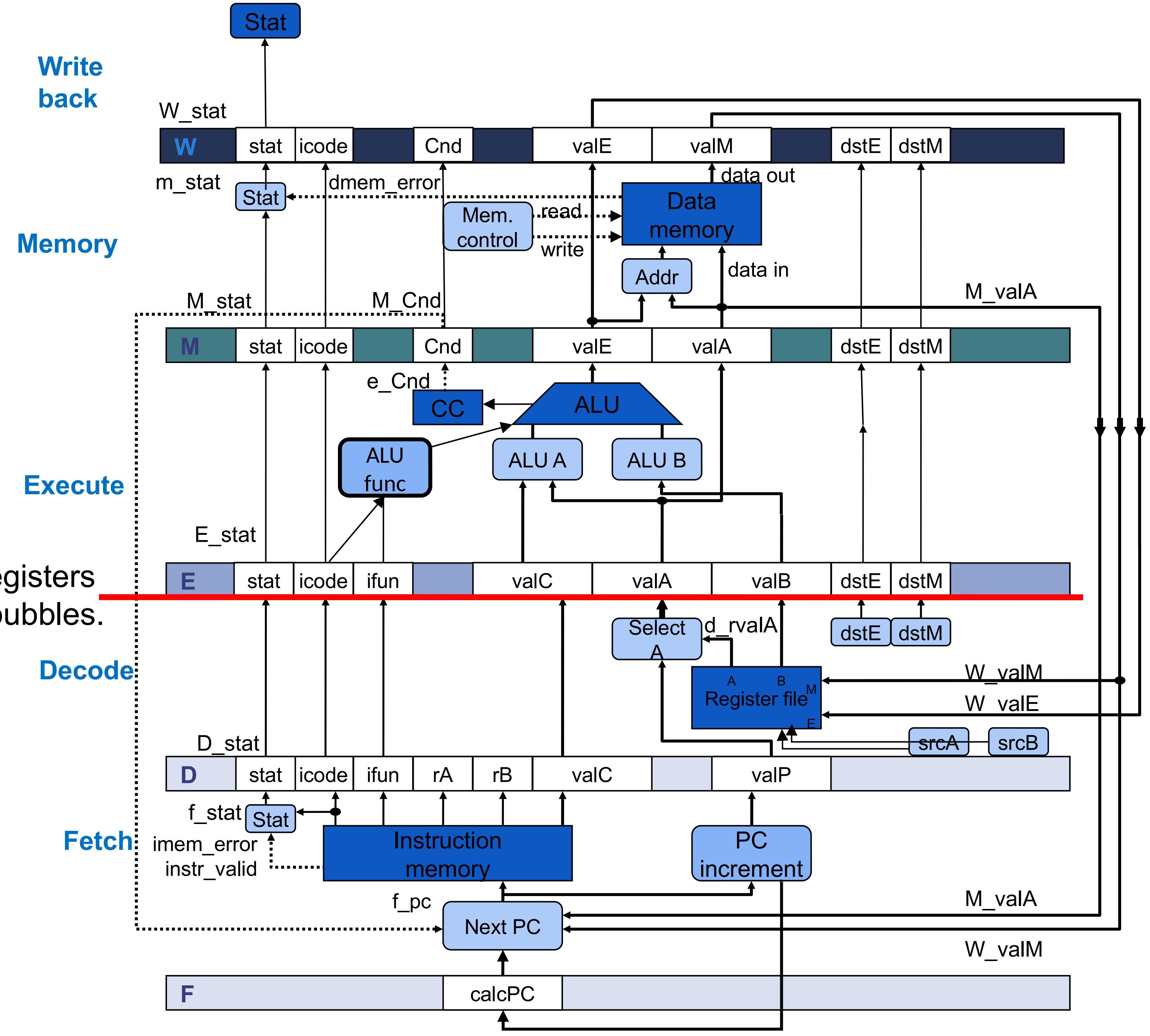
Consider the following sequence:

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

Until %rbx is written back: F and D pipeline registers maintain their values while E inserts bubbles.

With stalling:

addq	F	D	E	M	W		
subq		F	D	D	D	D	E

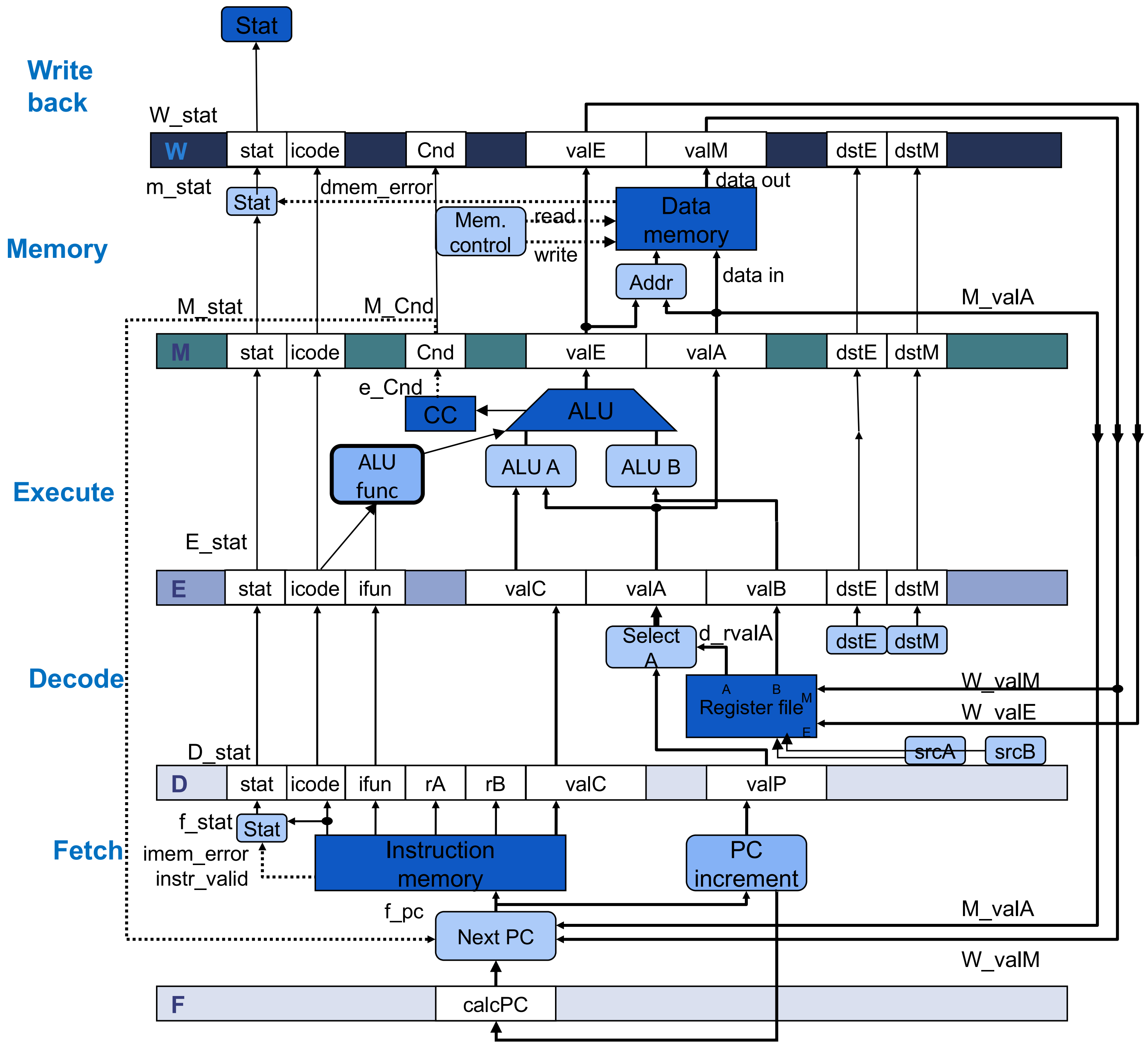


A Fix?

Consider the following sequence:

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

By what point does the subq need the value of %rbx?

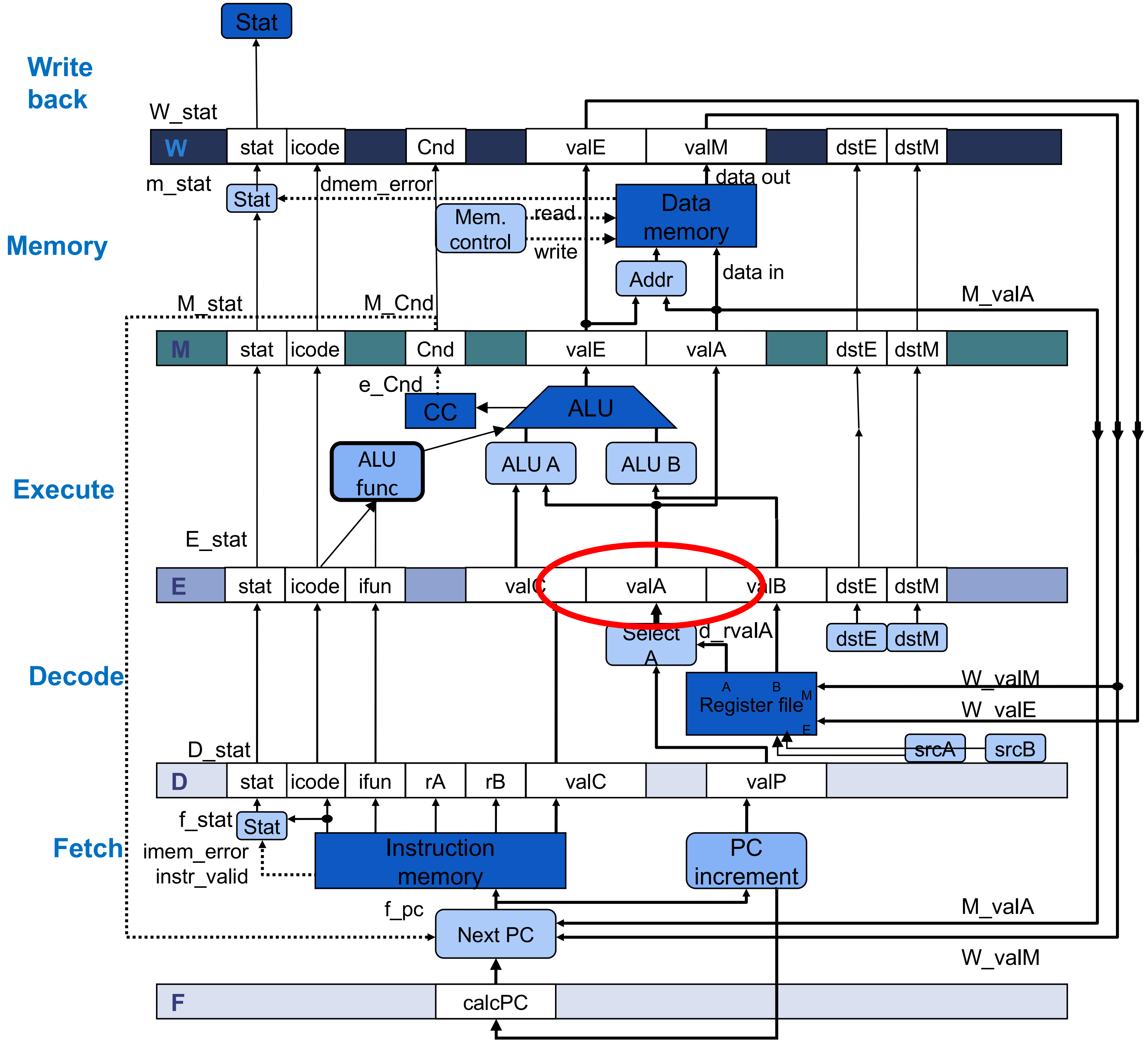


A Fix?

Consider the following sequence:

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

By what point does the subq need the value of %rbx?



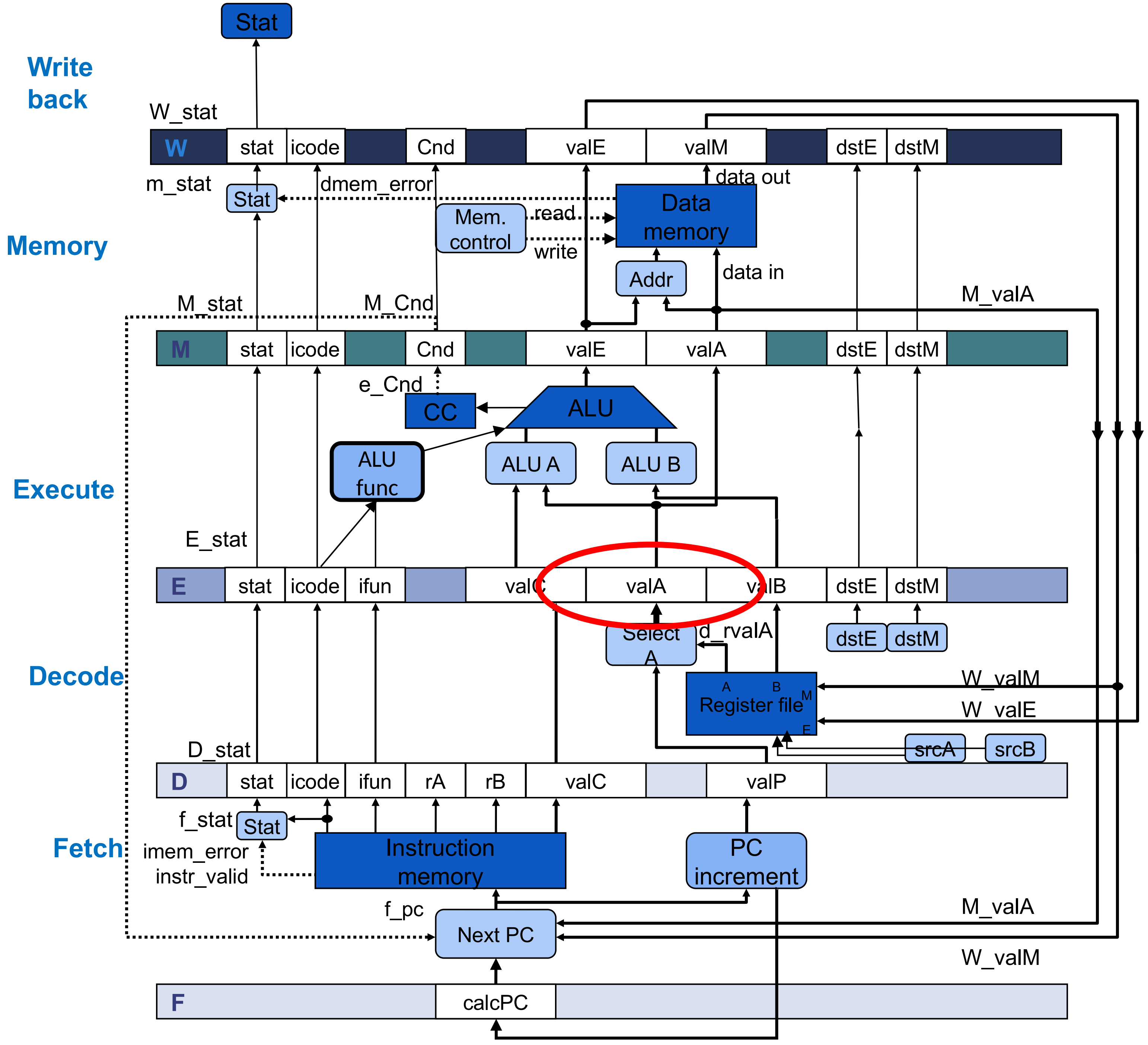
A Fix?

Consider the following sequence:

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

By what point does the subq need the value of %rbx?

At what point does the addq have the value of %rbx?



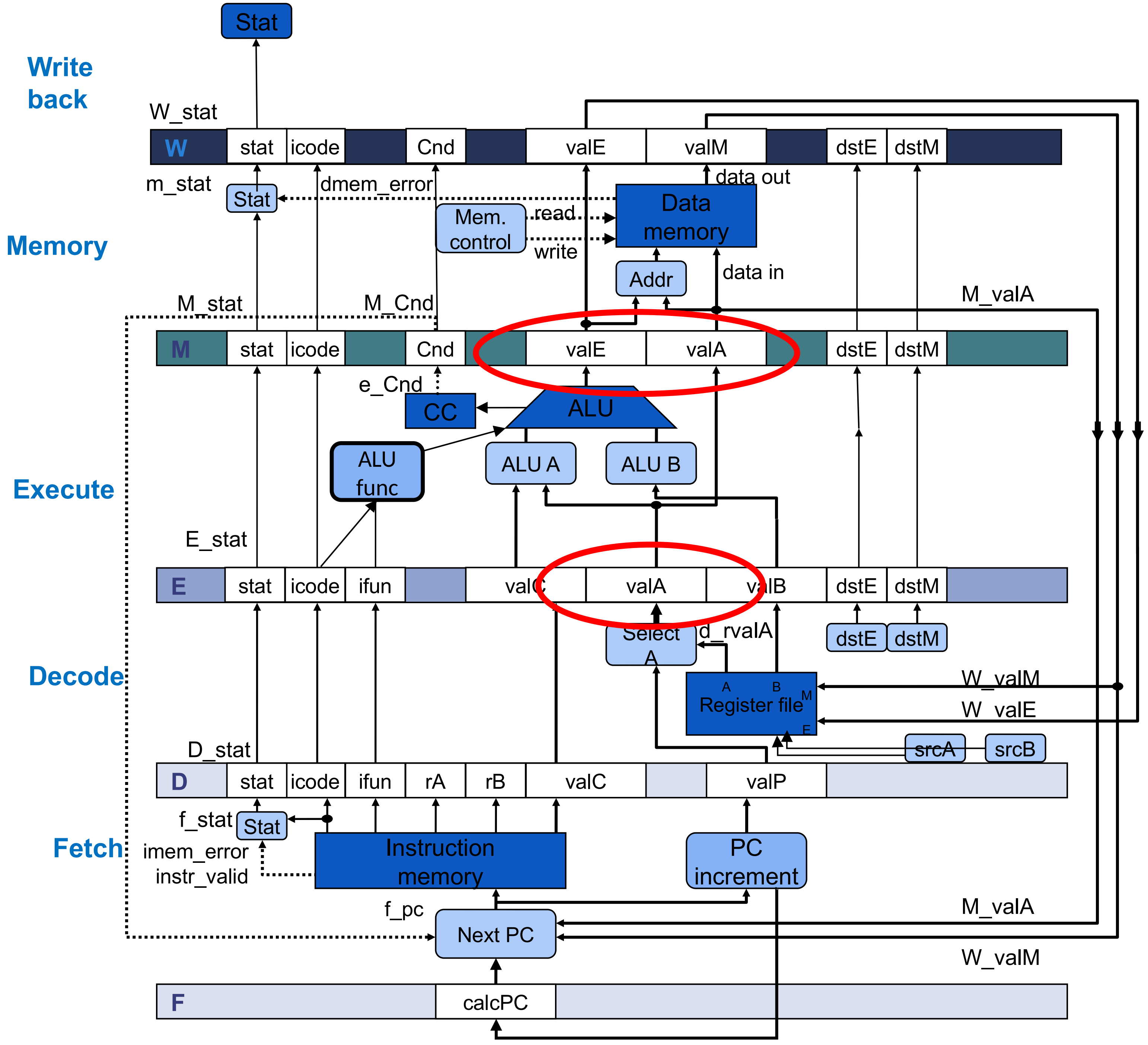
A Fix?

Consider the following sequence:

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

By what point does the subq need the value of %rbx?

At what point does the addq have the value of %rbx?



Forwarding (1)

Consider the following sequence:

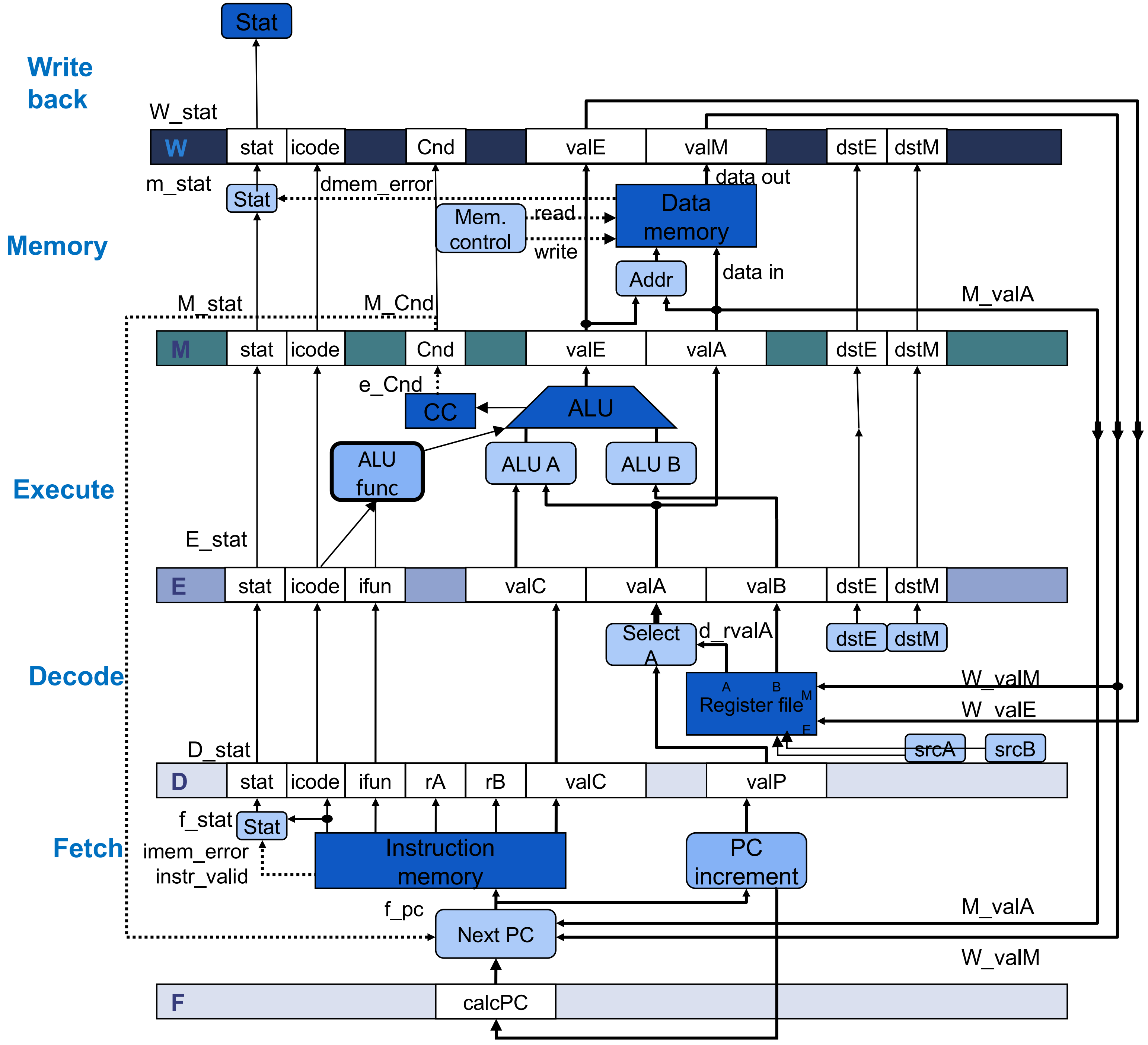
addq %rax, %rbx
subq %rbx, %rcx

We **have** the value we need before it's written into the register file, so could we somehow use the signal e_valE?

Q1: When do we NEED the value for the subq?

With stalling:

addq	F	D	E	M	W		
subq		F	D	D	D	D	E



Forwarding (2)

Consider the following sequence:

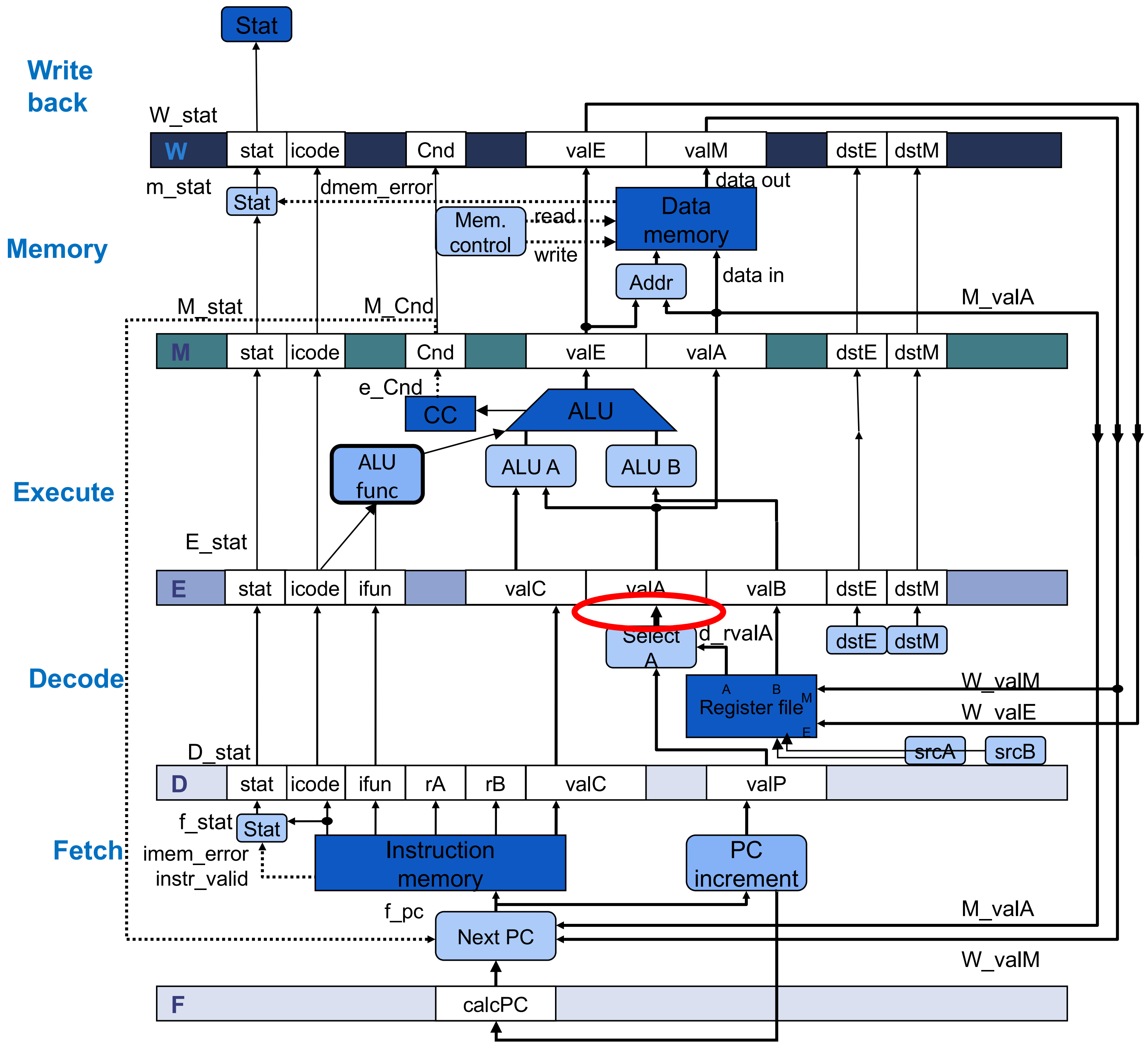
addq %rax, %rbx
subq %rbx, %rcx

We **have** the value we need before it's written into the register file, so could we somehow use the signal e_valE?

Q1: When do we NEED the value for the subq?

With stalling:

addq	F	D	E	M	W		
subq		F	D	D	D	D	E



Forwarding (2)

Consider the following sequence:

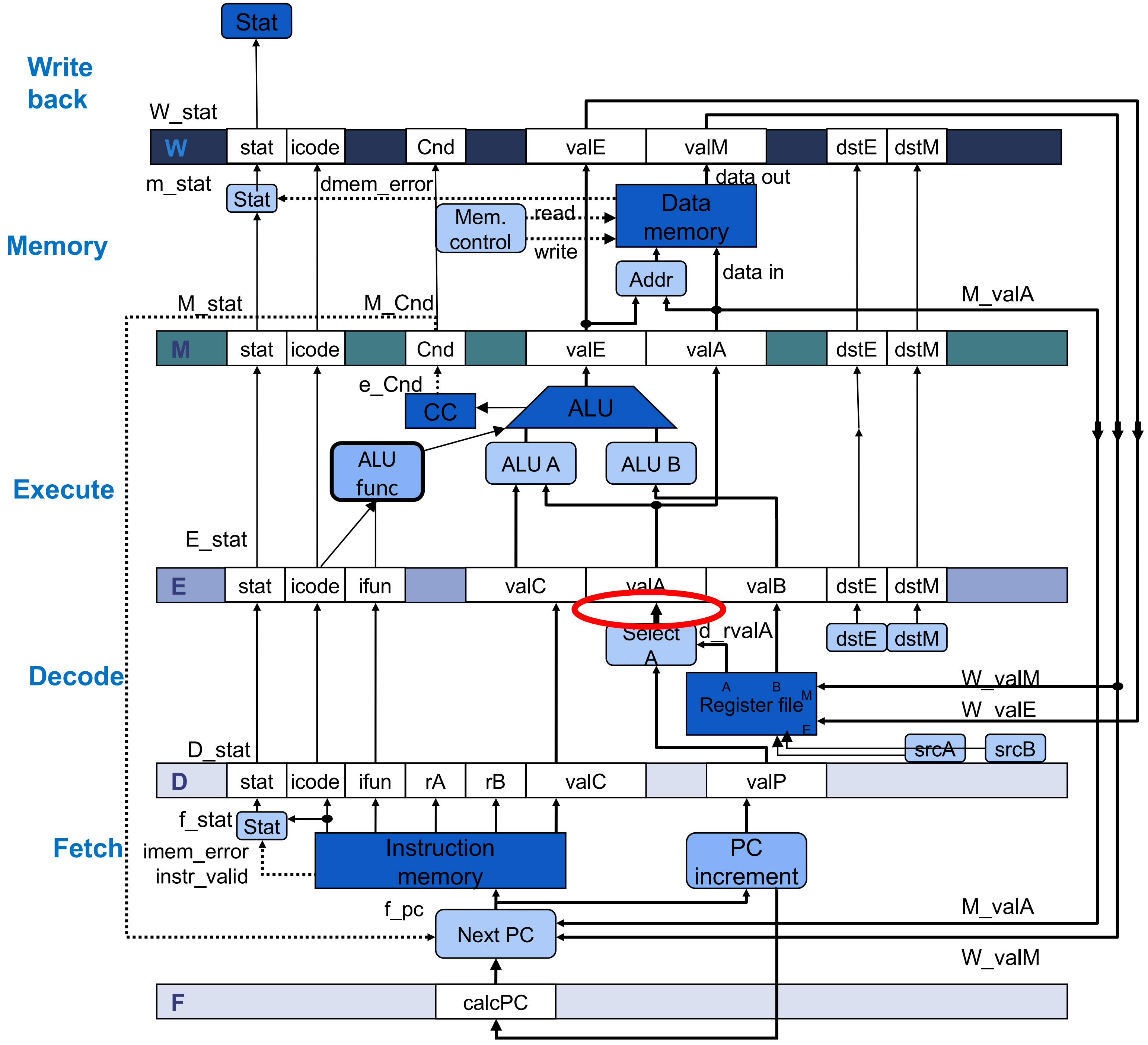
addq %rax, %rbx
subq %rbx, %rcx

We **have** the value we need before it's written into the register file, so could we somehow use the signal e_valE?

Q1: When do we NEED the value for the subq?
Q2: When do we HAVE the value from addq?

With stalling:

addq	F	D	E	M	W		
subq		F	D	D	D	D	E



Forwarding (2)

Consider the following sequence:

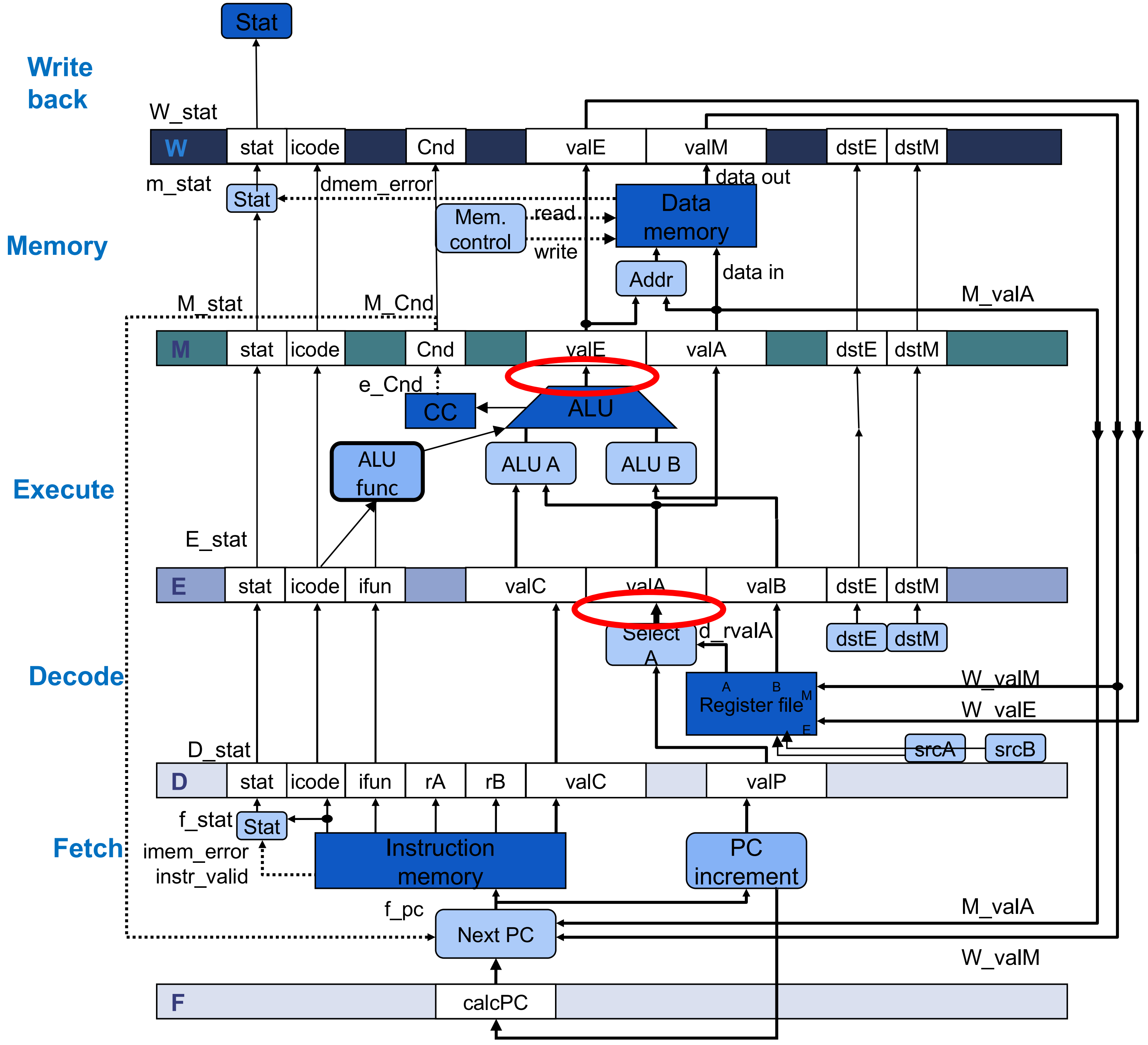
addq %rax, %rbx
subq %rbx, %rcx

We **have** the value we need before it's written into the register file, so could we somehow use the signal e_valE?

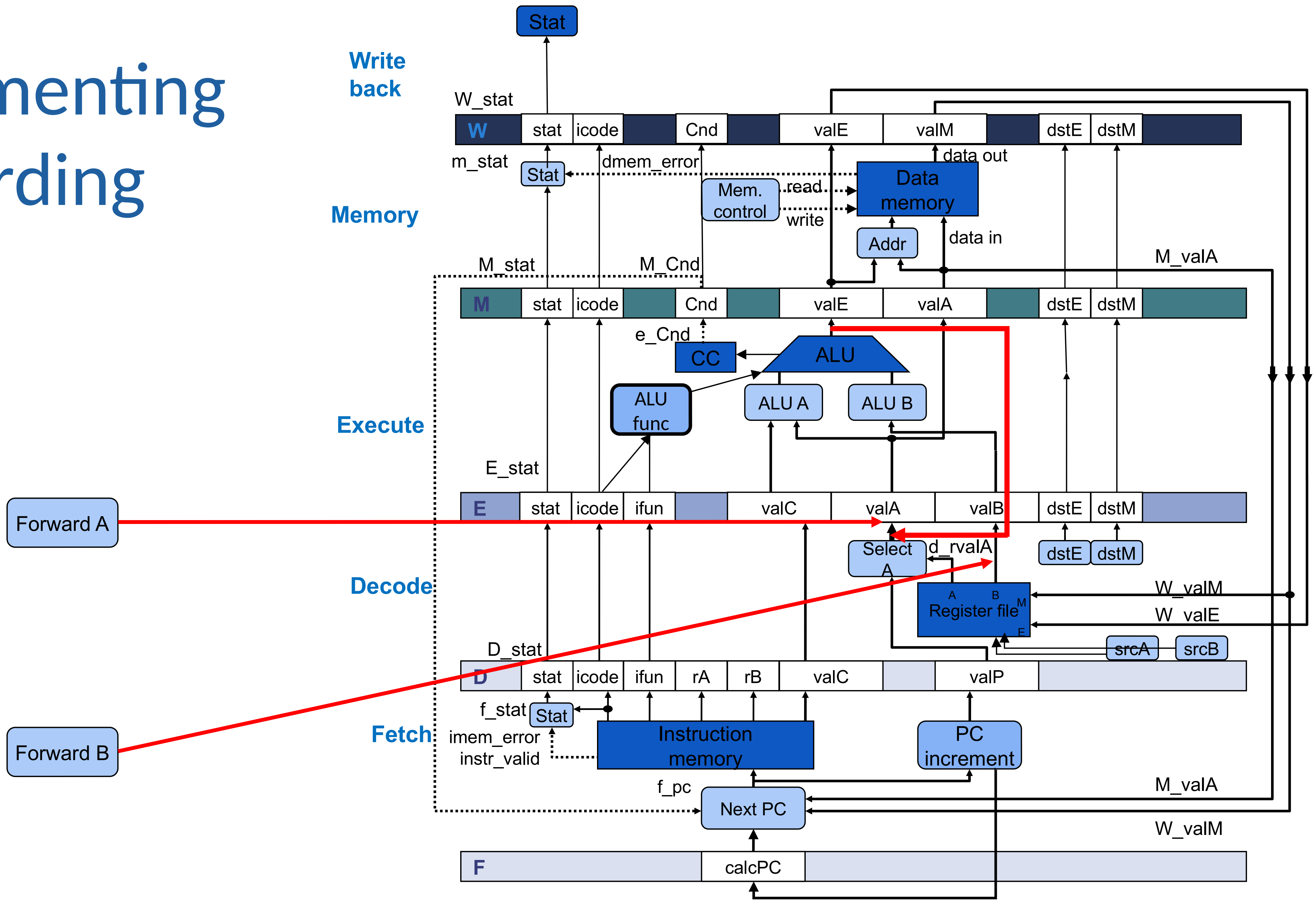
Q1: When do we NEED the value for the subq?
Q2: When do we HAVE the value from addq?

With stalling:

addq	F	D	E	M	W		
subq		F	D	D	D	D	E

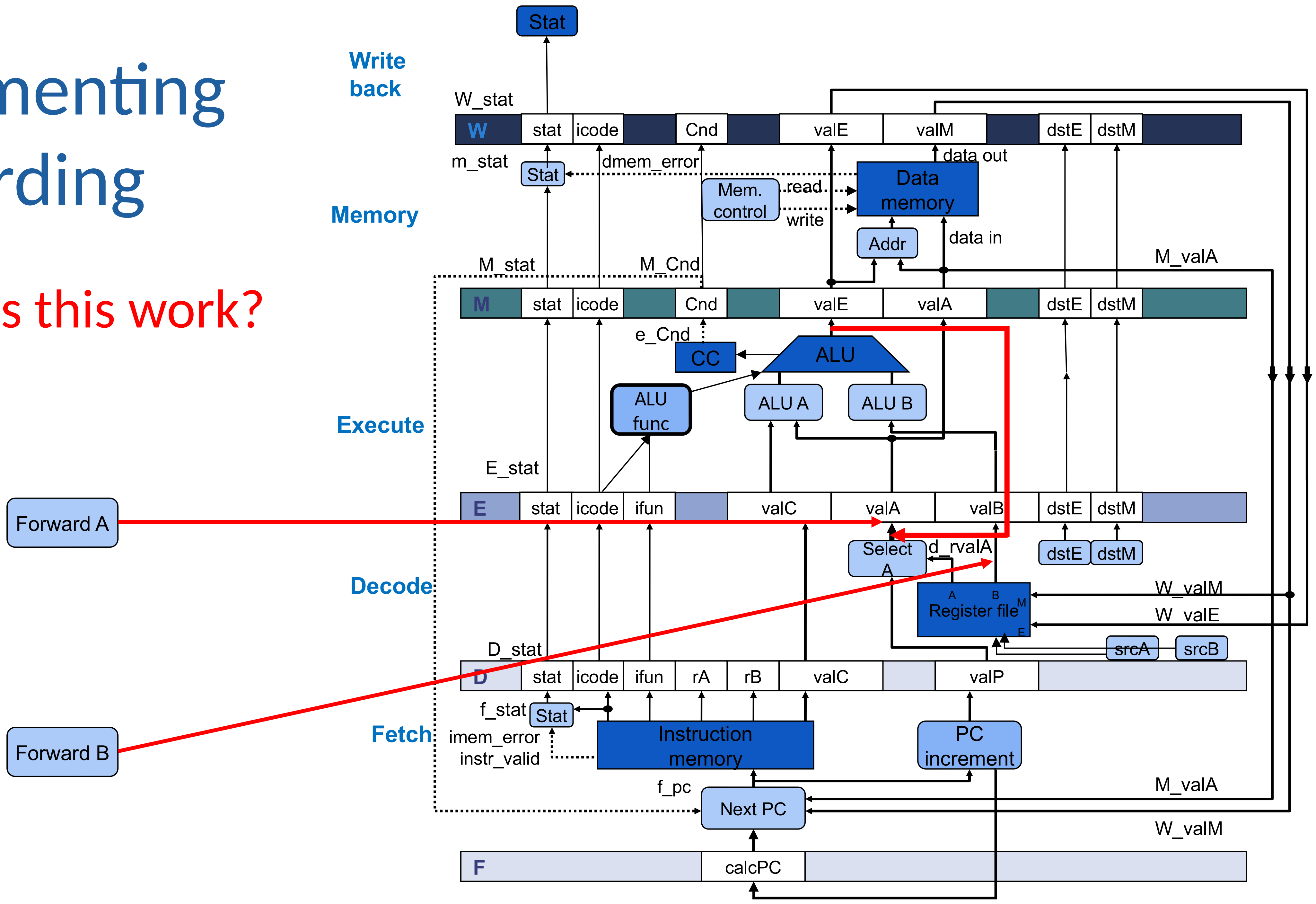


Implementing Forwarding



Implementing Forwarding

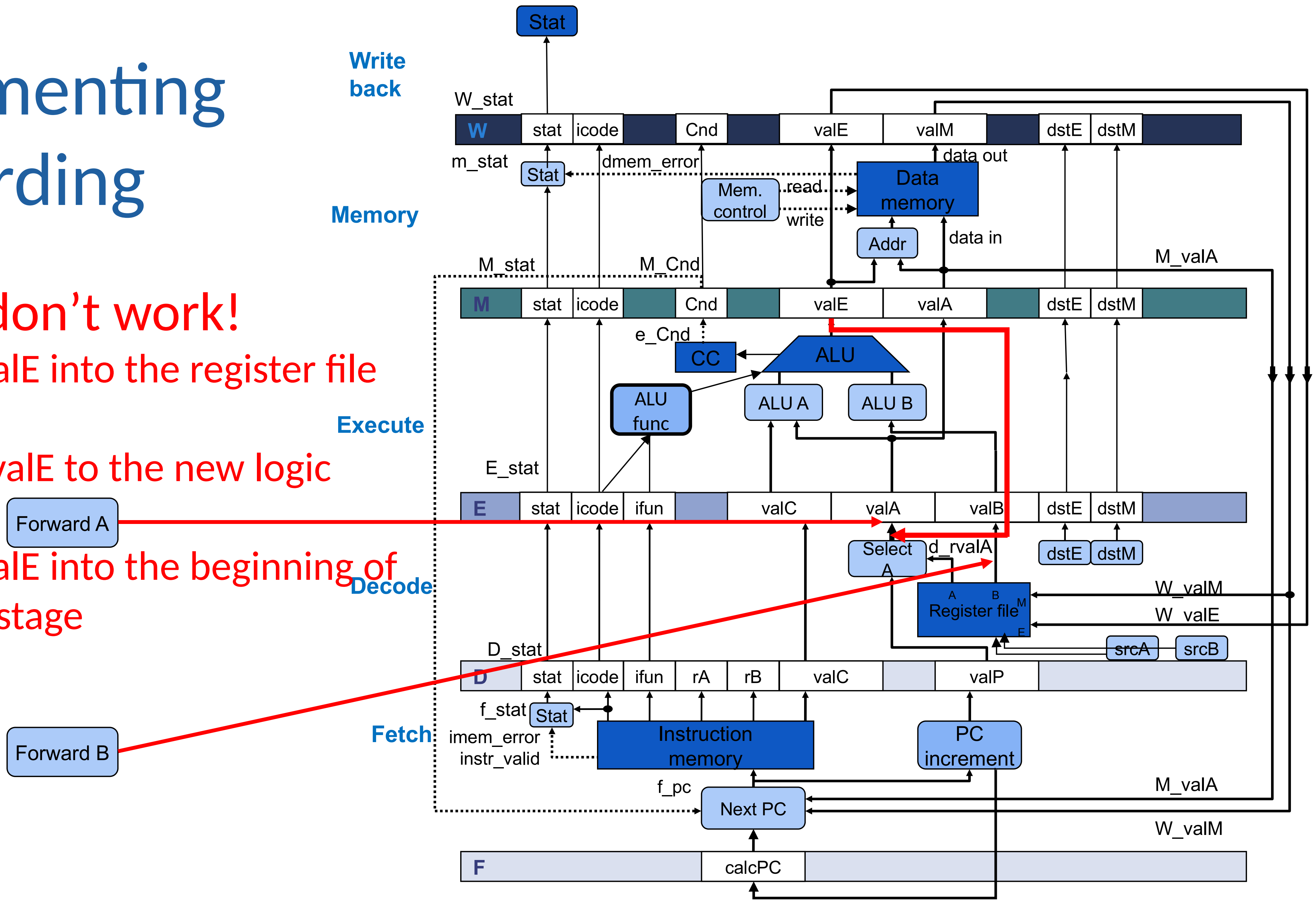
Why does this work?



Implementing Forwarding

Things that don't work!

1. forward e_valE into the register file
2. forward M_valE to the new logic
3. forward e_valE into the beginning of the execute stage

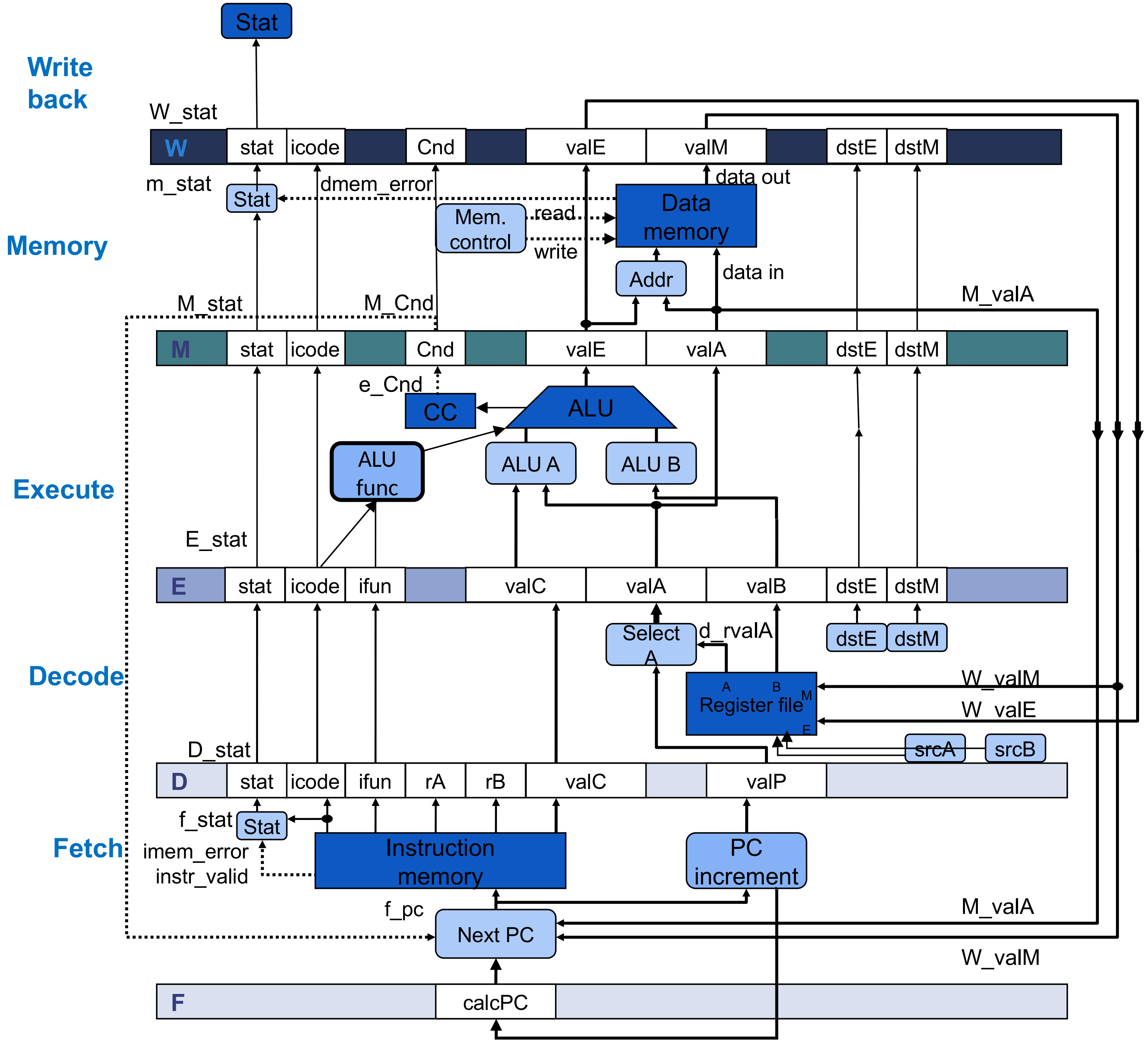


Hey, Quick Q..

Consider the following sequence:

addq %rax, %rbx
irmovq 10, %rdx
subq %rbx, %rcx

NOW at what point does the subq need the value of %rbx? Same spot?

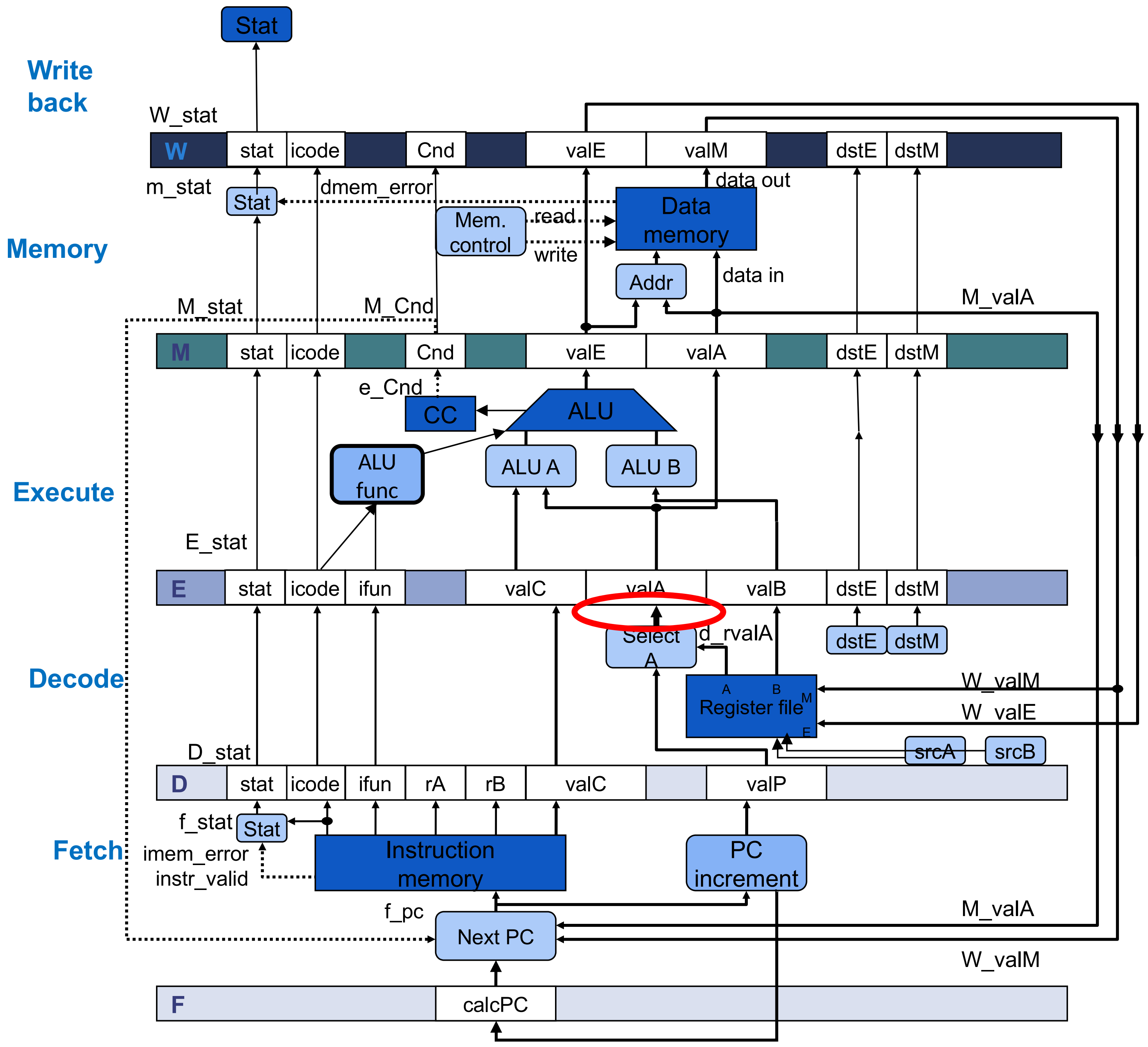


Hey, Quick Q..

Consider the following sequence:

addq %rax, %rbx
irmovq 10, %rdx
subq %rbx, %rcx

NOW at what point does the subq need the value of %rbx? Same spot?



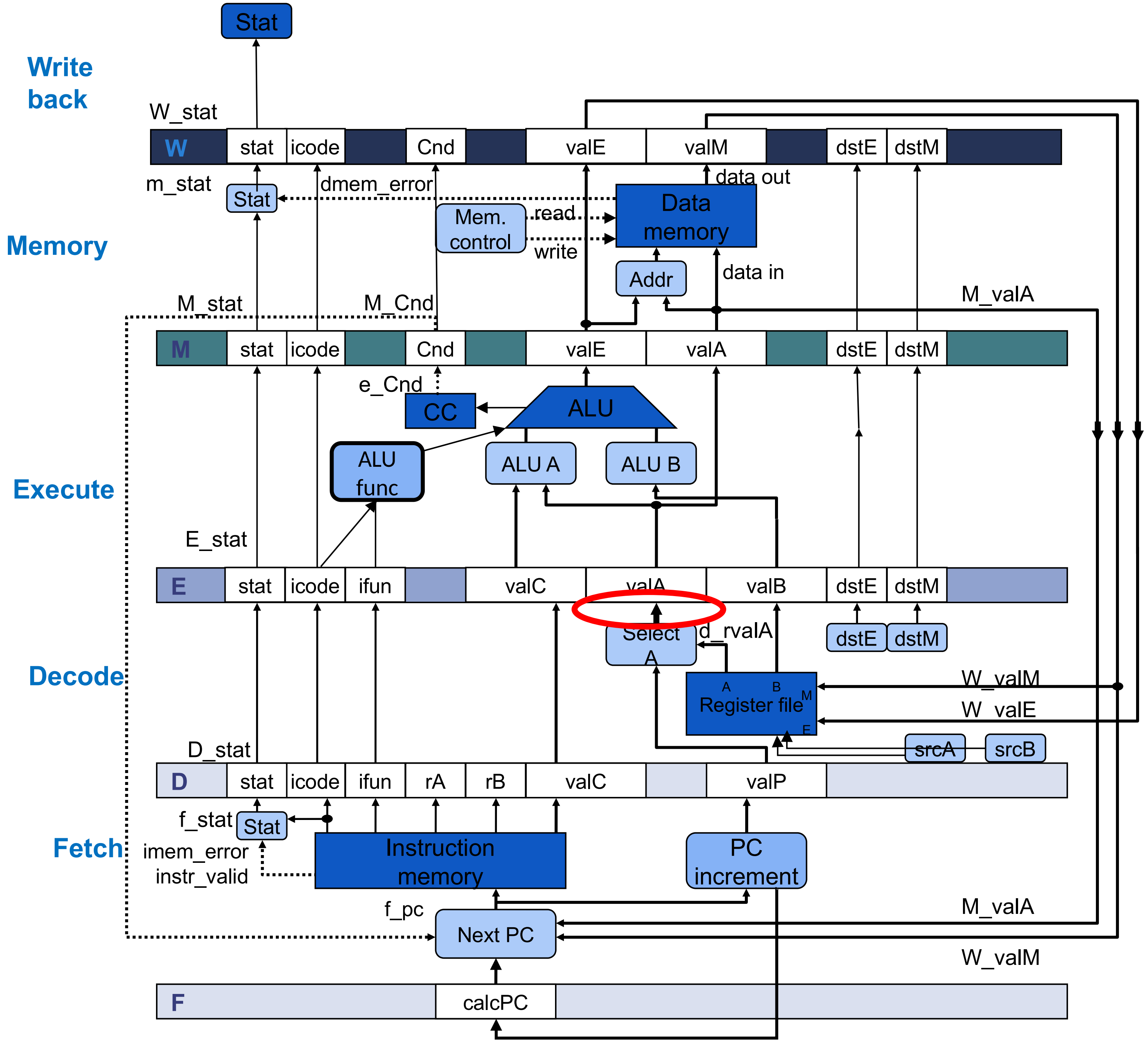
Hey, Quick Q..

Consider the following sequence:

addq %rax, %rbx
irmovq 10, %rdx
subq %rbx, %rcx

NOW at what point does the subq need the value of %rbx? Same spot?

Exactly where is the value of %rbx when it's needed? Same spot?



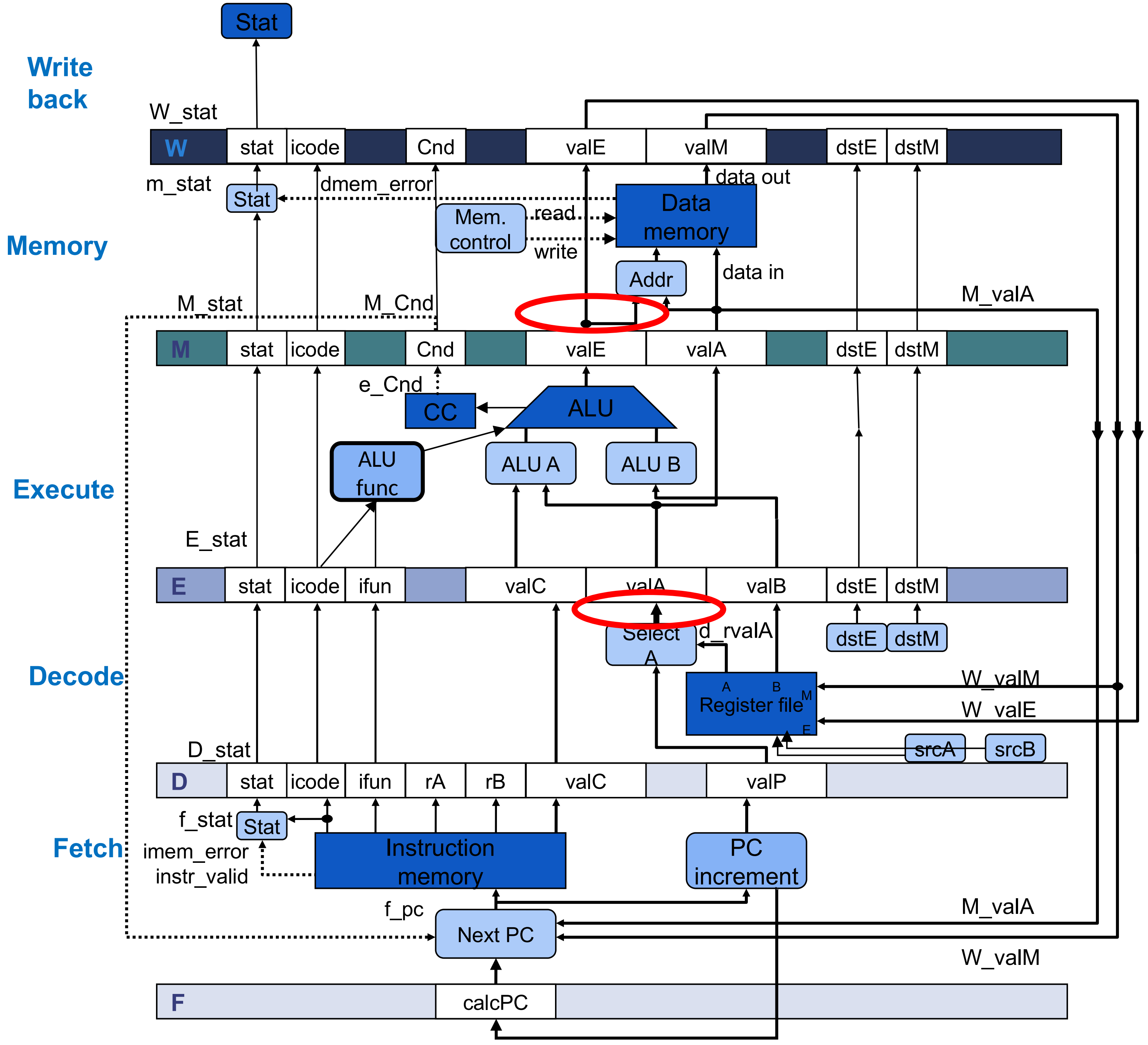
Hey, Quick Q..

Consider the following sequence:

addq %rax, %rbx
irmovq 10, %rdx
subq %rbx, %rcx

NOW at what point does the subq need the value of %rbx? Same spot?

Exactly where is the value of %rbx when it's needed? Same spot?



How does the processor keep track of all this?

In some sense, just like before: with logic blocks!

Logic that reasons about instructions at multiple stages and reasons about their interactions: messier, but not really different.

You can read about pipeline control logic in the text.

> For CPSC 313: no need to know the details, but you should a) know that such logic exists and b) conceptually know what it has to do and c) be able to talk about how it could be extended/changed as needed.

In-class Activity

Explore other forwarding options

Wrapping Up

- Forwarding lets us mitigate some of the hazards that occur due to pipelining.
- In early RISC processors, there was no stalling and no forwarding (chips could not contain as many transistors and wires).
  - The compiler was responsible for inserting nops into the assembly code to ensure correct execution!
  - When humans had to write assembly directly (which occurs in some of the lowest levels of the operating system), those programmers had to be acutely aware of the internals of the hardware.
- Today, the hardware is so complex that it has to take care of hazards, not people or compilers.