

# Control Hazards

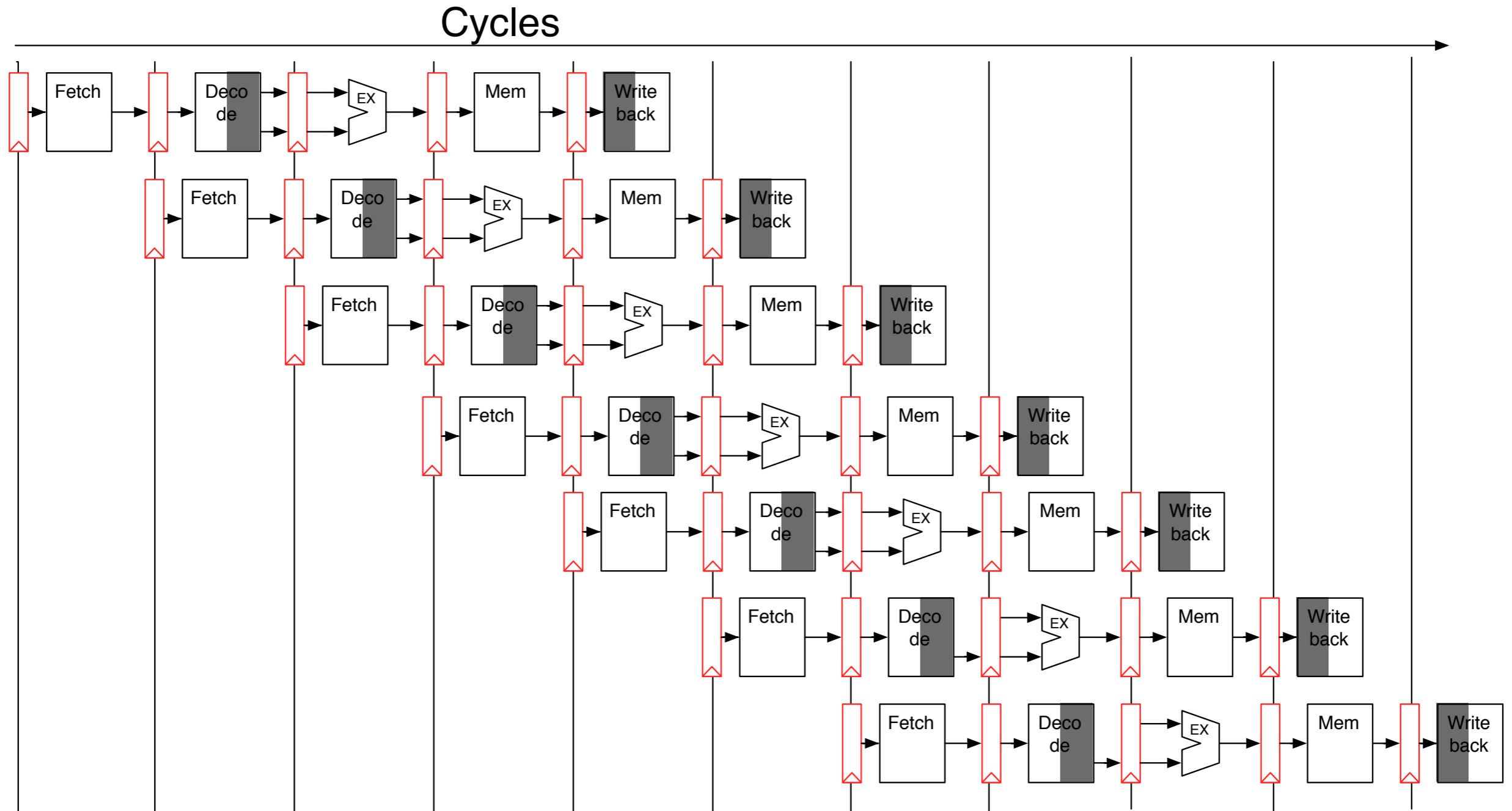
# Today

- Control Hazards
- DRAM

# Key Points: Control Hazards

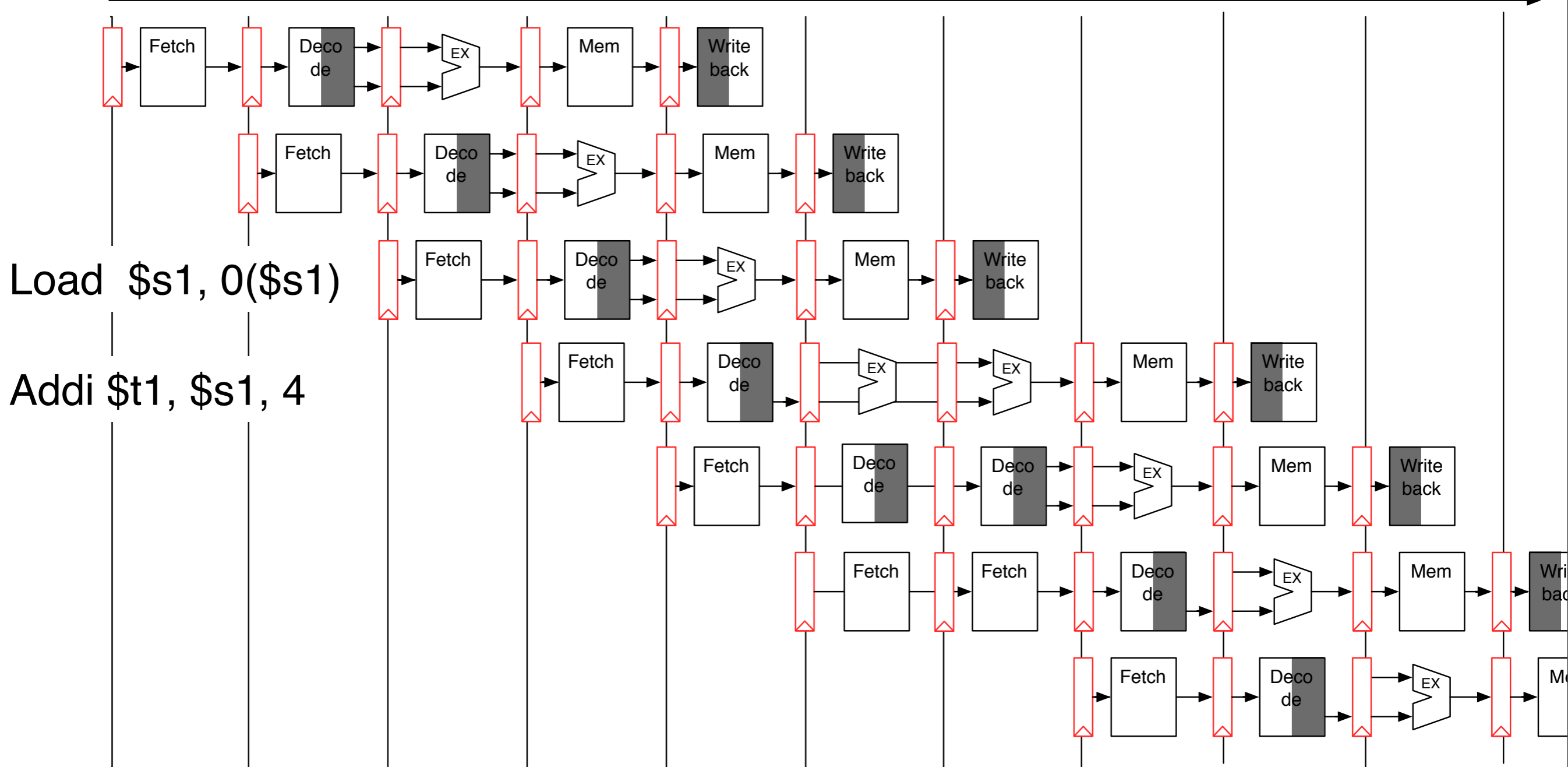
- Control occur when we don't know what the next instruction is
- Mostly caused by branches
- Strategies for dealing with them
  - Stall
  - Guess!
    - Leads to speculation
    - Flushing the pipeline
    - Strategies for making better guesses
- Understand the difference between stall and flush

# Normal operation



# Stalling for Load

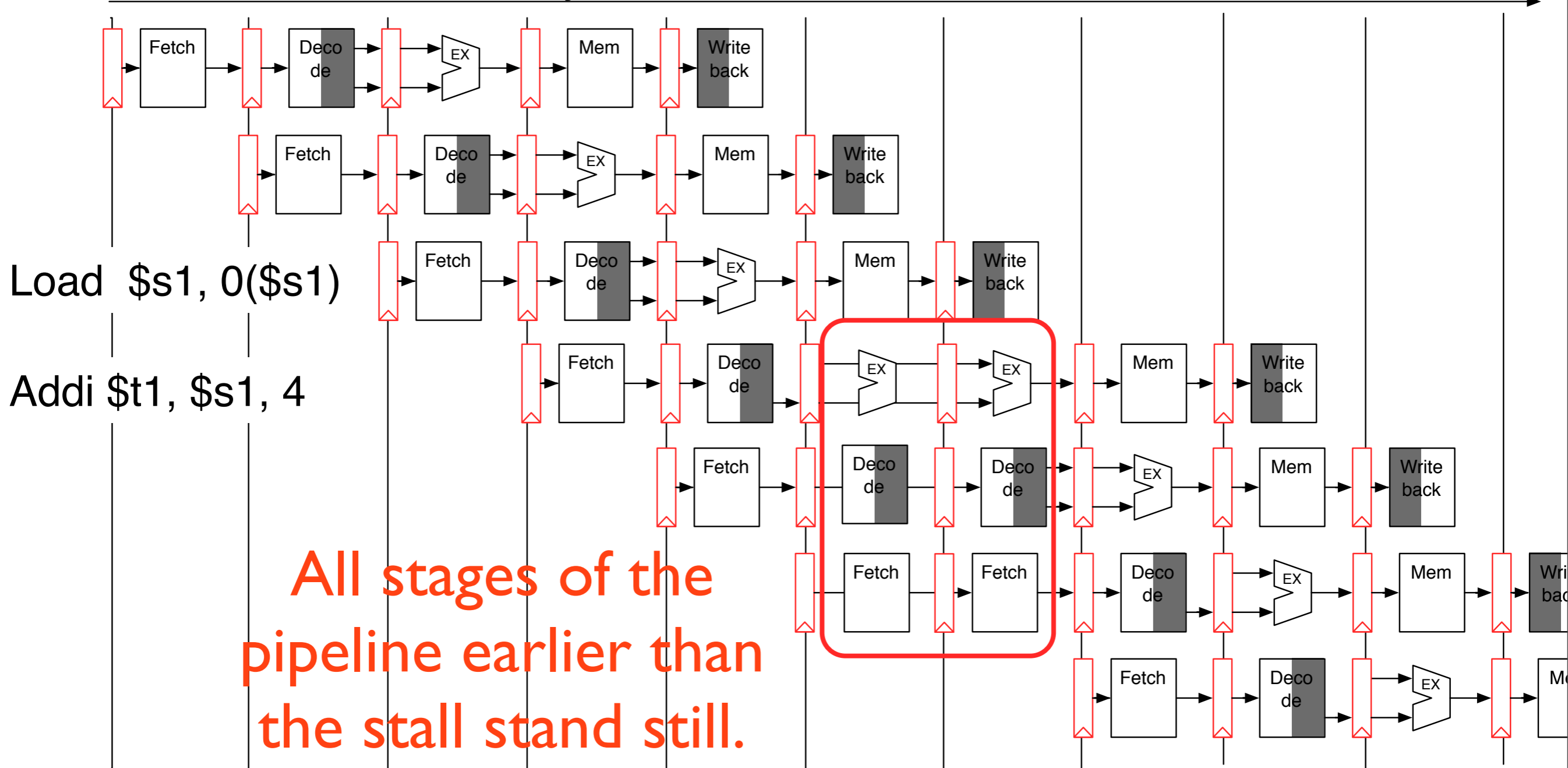
Cycles



To “stall” we insert a noop *in place of* the instruction and freeze the earlier stages of the pipeline

# Stalling for Load

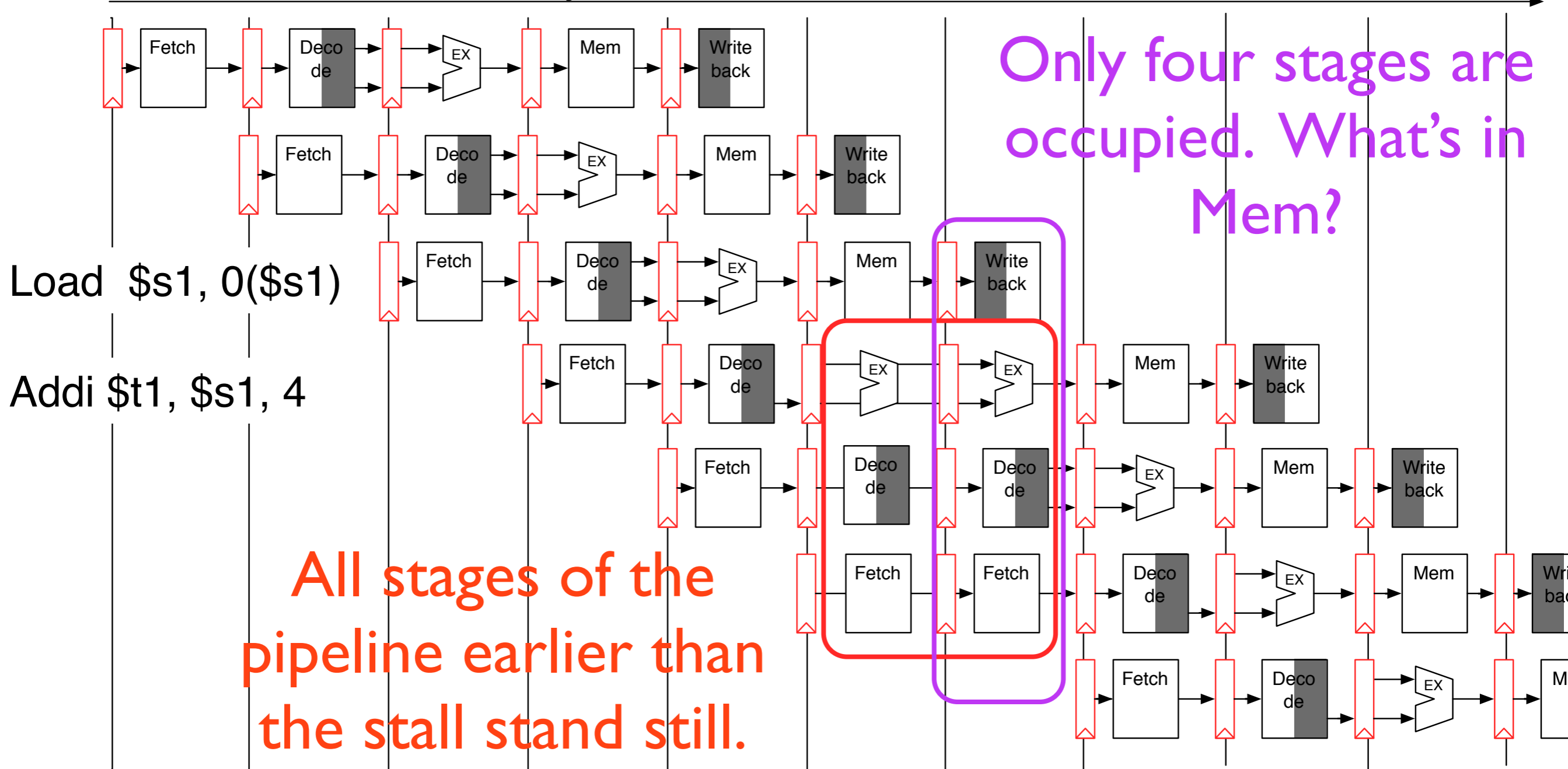
Cycles



To “stall” we insert a noop *in place of* the instruction and freeze the earlier stages of the pipeline

# Stalling for Load

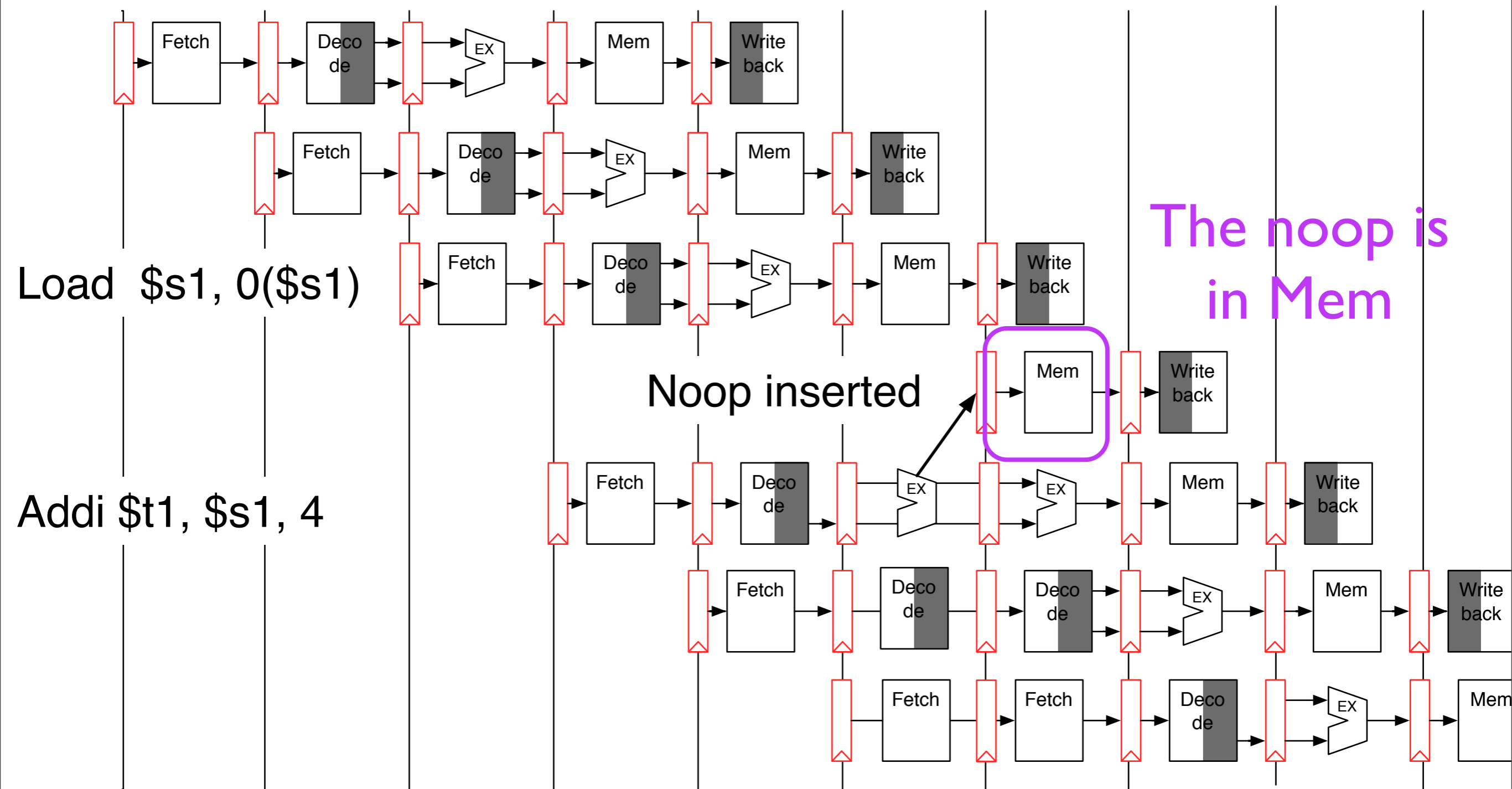
Cycles



To “stall” we insert a noop *in place of* the instruction and freeze the earlier stages of the pipeline

# Inserting Noops

Cycles



The noop is in Mem

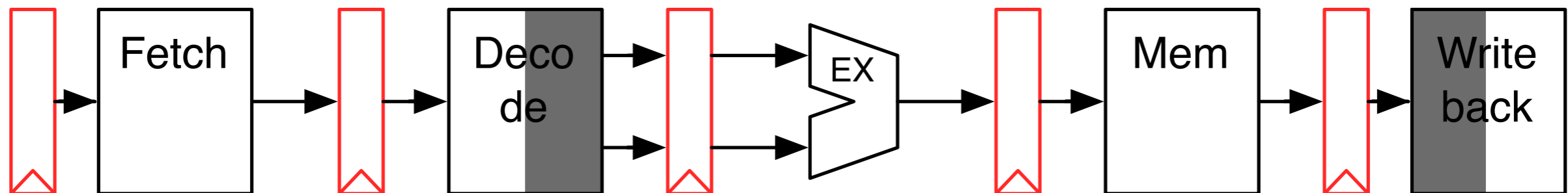
Noop inserted

To "stall" we insert a noop *in place of* the instruction and freeze the earlier stages of the pipeline

# Control Hazards

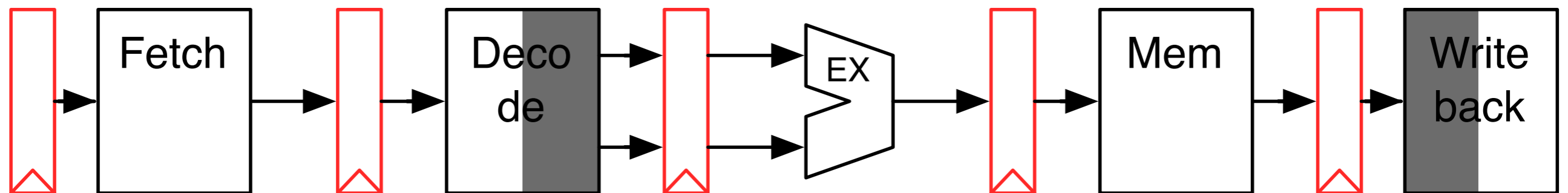
- Computing the new PC

```
add $s1, $s3, $s2
sub $s6, $s5, $s2
beq $s6, $s7, somewhere
and $s2, $s3, $s1
```



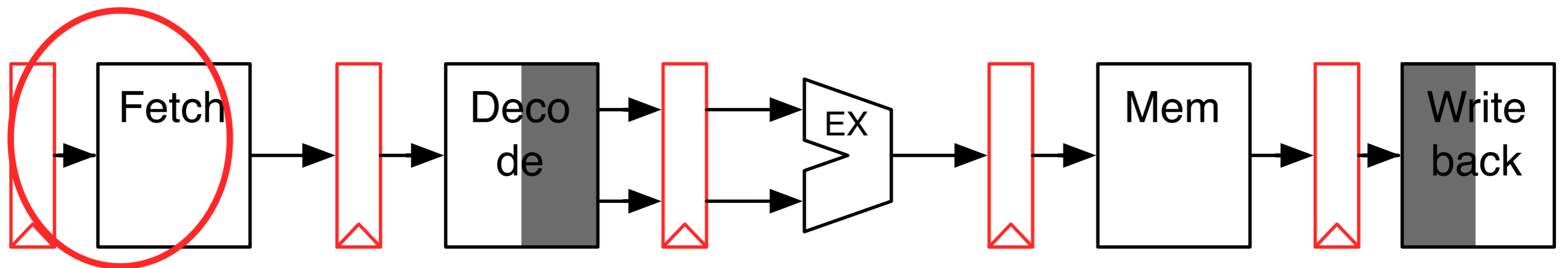
# Computing the PC

- Non-branch instruction
  - $PC = PC + 4$
- When is PC ready?



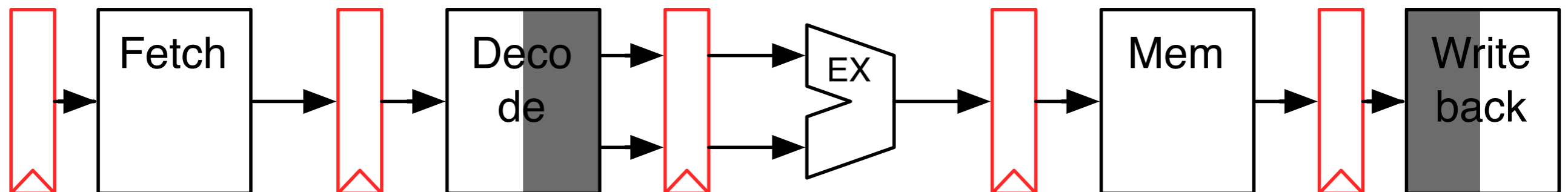
# Computing the PC

- Non-branch instruction
  - $PC = PC + 4$
- When is PC ready?



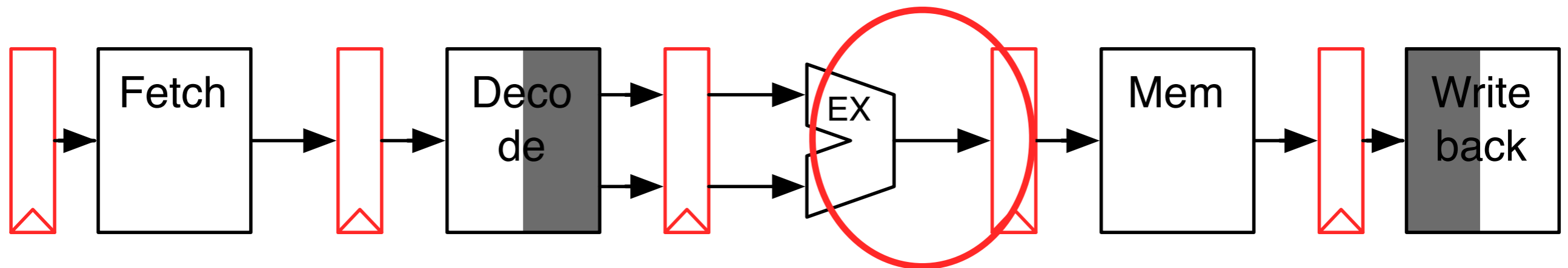
# Computing the PC

- Branch instructions
  - `bne $s1, $s2, offset`
  - `if ($s1 != $s2) { PC = PC + offset } else { PC = PC + 4; }`
- When is the value ready?



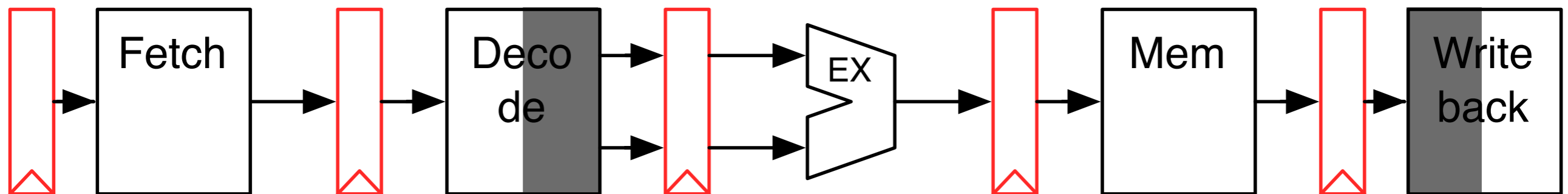
# Computing the PC

- Branch instructions
  - `bne $s1, $s2, offset`
  - `if ($s1 != $s2) { PC = PC + offset } else { PC = PC + 4; }`
- When is the value ready?



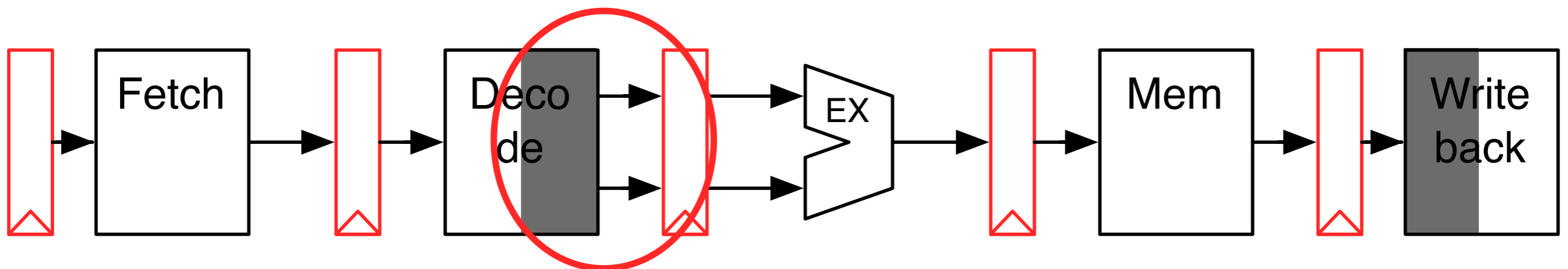
# Computing the PC

- But wait!
- When do we know *whether* the instruction is a branch?



# Computing the PC

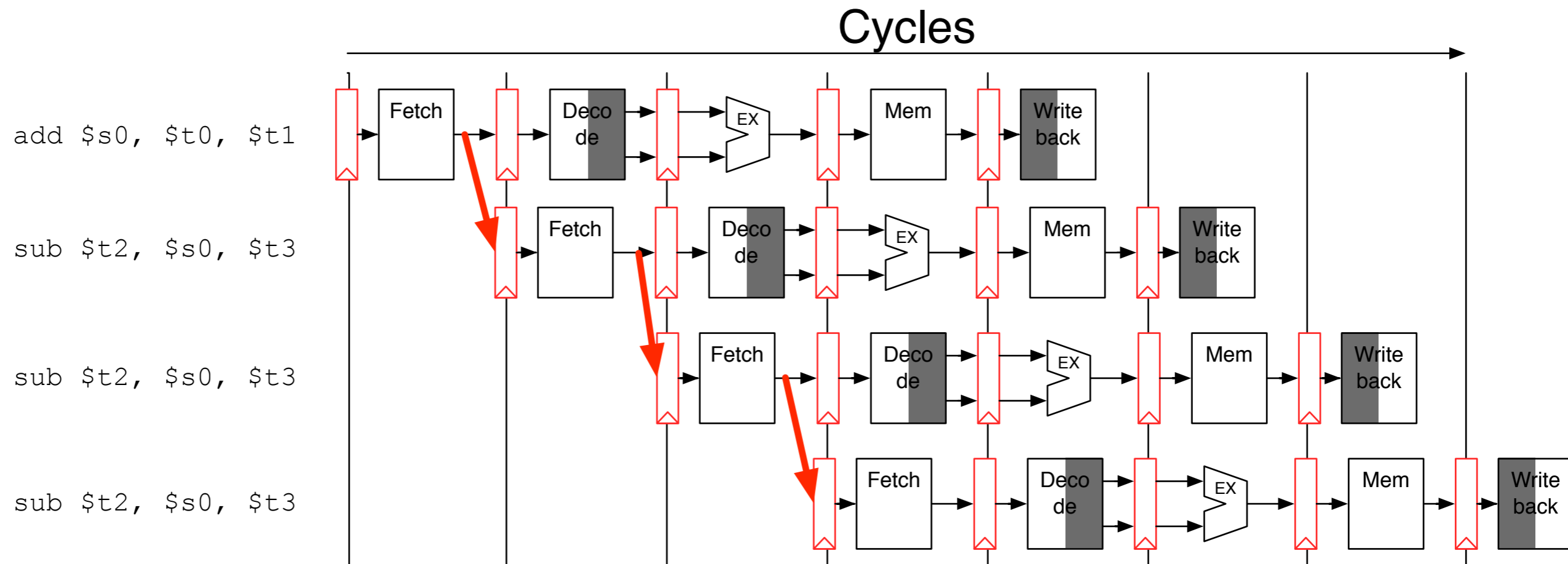
- But wait!
- When do we know *whether* the instruction is a branch?



# There is a constant control hazard

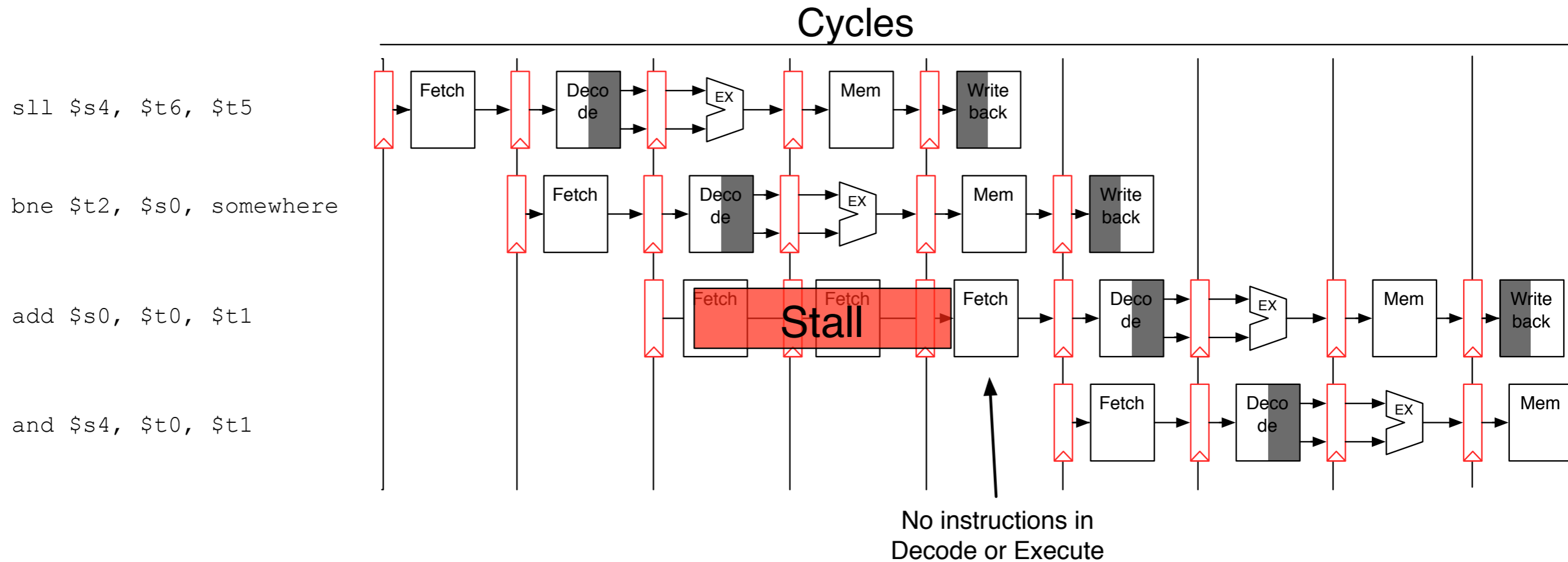
- We don't even know what kind of instruction we have until decode.
- What do we do?

# Smart ISA design



- Make it very easy to tell if the instruction is a branch -- maybe a single bit or just a couple.
- Decoding these bits is nearly trivial.
- In MIPS the branches and jumps are opcodes 0-7, so if the high order bits are zero, it's a control instruction

# Dealing with Branches: Option 1 -- stall



- What does this do to our CPI?
- Speedup?

# Performance impact of stalling

- $ET = I * CPI * CT$
- Branches about about 1 in 5 instructions
- What's the CPI for branches?
  
- Amdah's law: Speedup =
- $ET =$

# Performance impact of stalling

- $ET = I * CPI * CT$
- Branches about about 1 in 5 instructions
- What's the CPI for branches?  $1 + 2 = 3$
  
- Amdah's law: Speedup =
- $ET =$

# Performance impact of stalling

- $ET = I * CPI * CT$
- Branches about about 1 in 5 instructions
- What's the CPI for branches?  $1 + 2 = 3$
  
- Amdah's law:  $Speedup = 1 / (.2 / (1/3) + (.8)) = 0.714$
- $ET =$

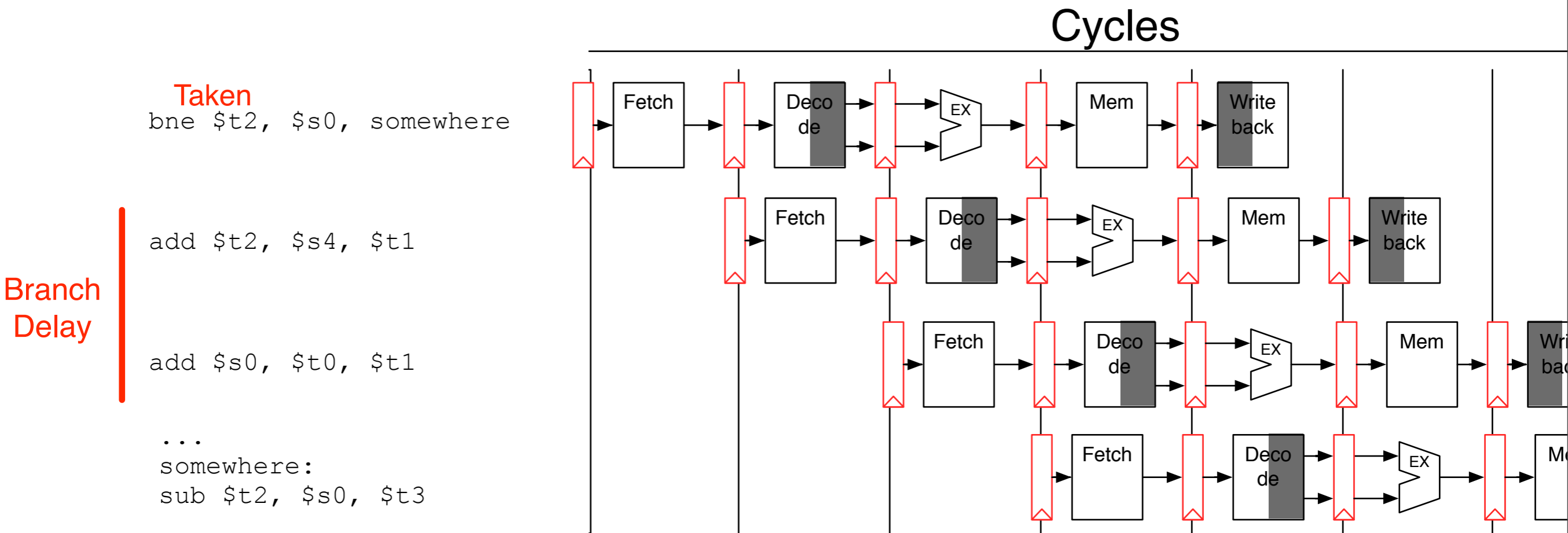
# Performance impact of stalling

- $ET = I * CPI * CT$
- Branches about about 1 in 5 instructions
- What's the CPI for branches?  $1 + 2 = 3$
  
- Amdah's law:  $Speedup = 1 / (.2 / (1/3) + (.8)) = 0.714$
- $ET = I * (.2 * 3 + .8 * 1) * CT = 1.4$

# Option 2: The compiler

- Use “branch delay” slots.
- The next  $N$  instructions after a branch are *always* executed
- Good
  - Simple hardware
- Bad
  - $N$  cannot change.

# Delay slots.



# Option 2: Simple Prediction

- Can a processor tell the future?
- For non-taken branches, the new PC is ready immediately.
- Let's just assume the branch is not taken
- Also called “branch prediction” or “control speculation”
- What if we are wrong?



