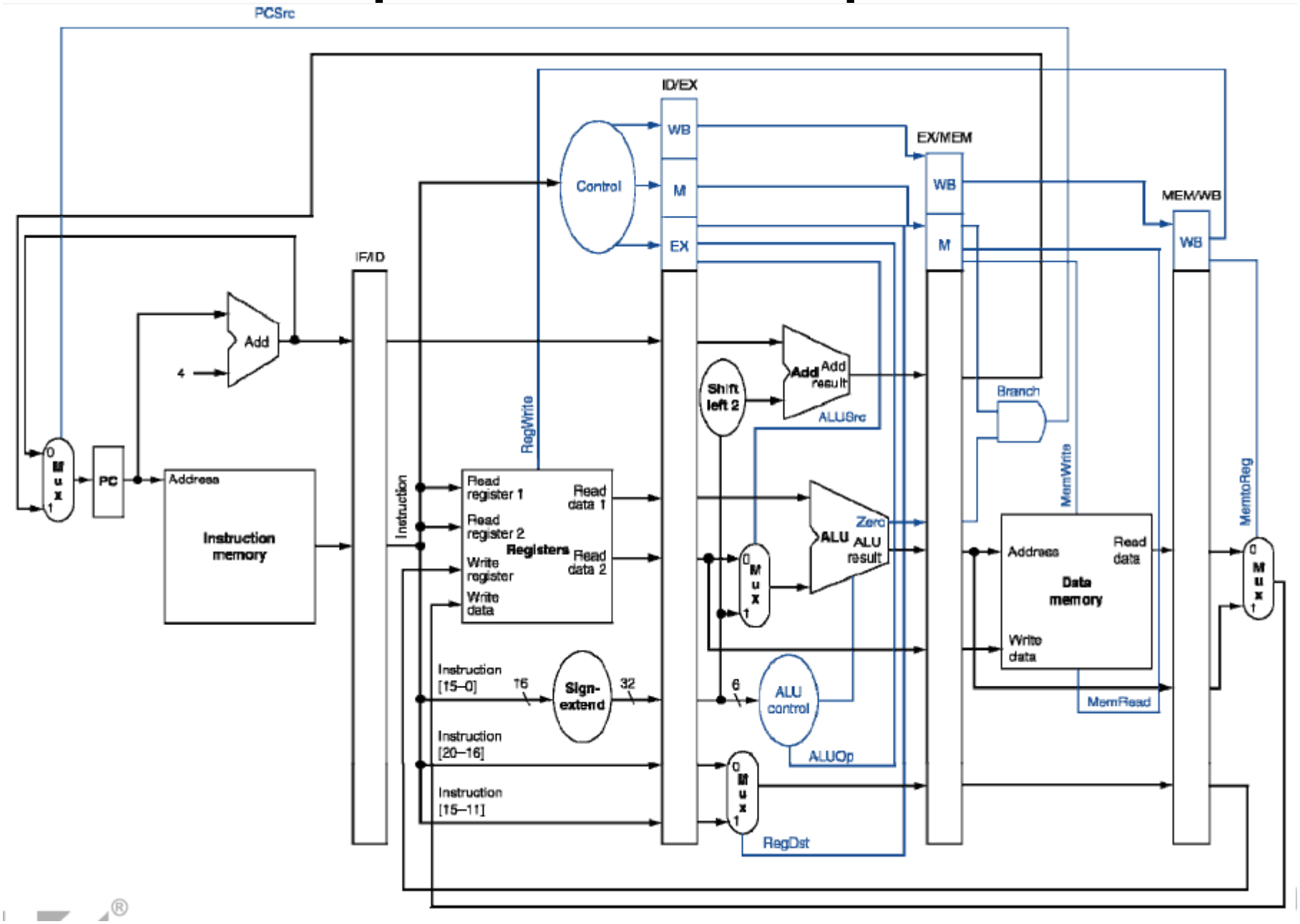
CS M151B:
Computer Systems Architecture

Week 6

Pipelined Datapath



Pipelined Datapath

- Recall that this ideal pipeline has some issues.
  - Data Hazards
  - Control Hazards
  - Alcoholism

Data Hazard: Dependencies

- Consider the following operations:
  1. lw $t0, 4($s0)
  2. add $t0, $t1, $t0
  3. lw $t2, 8($s0)
  4. add $t2, $t3, $t2
  5. add $t0, $t2, $t0
  6. sw $t0, 0($s0)
- What are the true dependencies?

Data Hazard: Dependencies

- Consider the following operations:
  1. lw $t0, 4($s0)
  2. add $t0, $t1, $t0
  3. lw $t2, 8($s0)
  4. add $t2, $t3, $t2
  5. add $t0, $t2, $t0
  6. sw $t0, 0($s0)
- What are the true dependencies?

Data Hazard: Dependencies

- Let's focus in on one

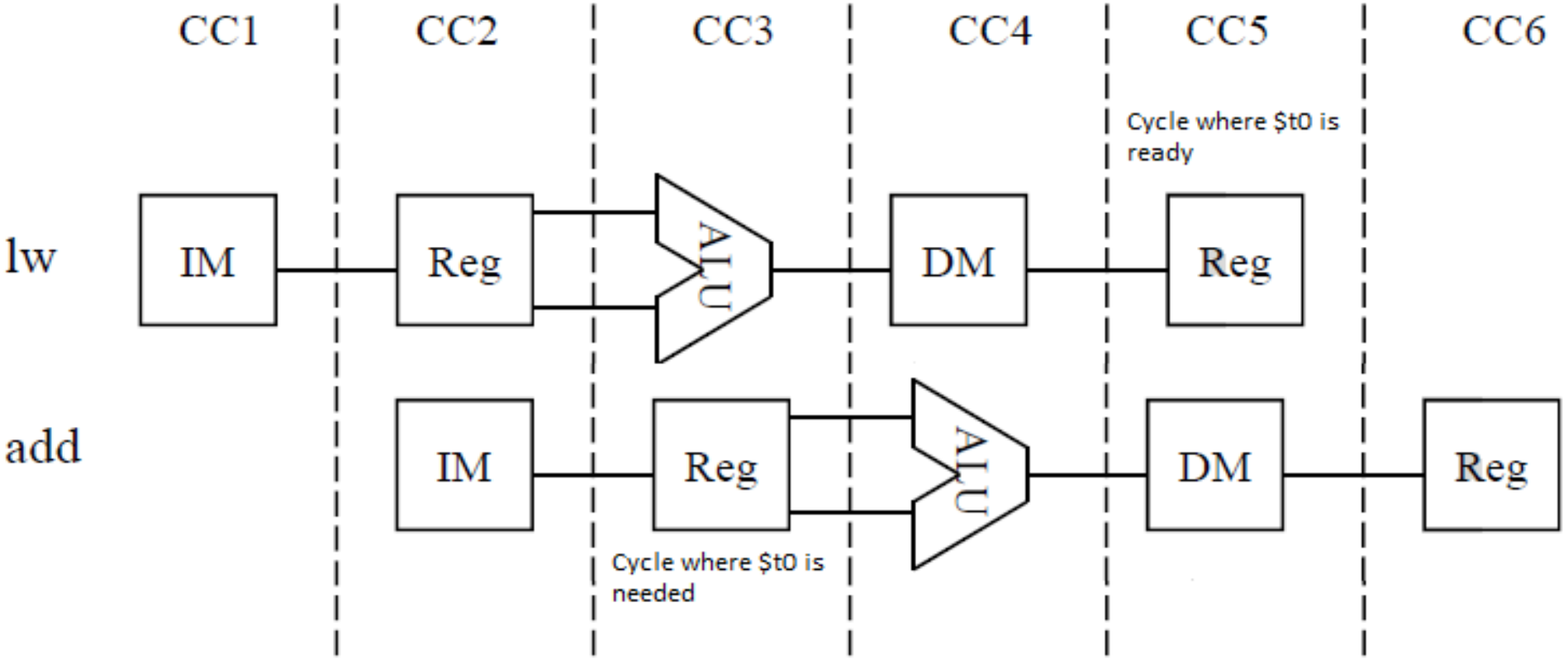
  1. lw $t0, 4($s0)

  2. add $t0, $t1, $t0

- What would it look like if we attempted to use the pipeline as is?

Data Hazard: Dependencies

- Let's focus in on one
  1. lw $t0, 4($s0)
  2. add $t0, $t1, $t0



Data Hazard: Dependencies

- Note: We can write to a register and read from that newly updated register in a single cycle. Thus, just as long as the dependent instruction reads from the register at the same cycle as the one that is writing to the register, it works.

Data Hazard: Dependencies

- Software Solutions
- Hardware Solutions

Data Hazard: Software Solution

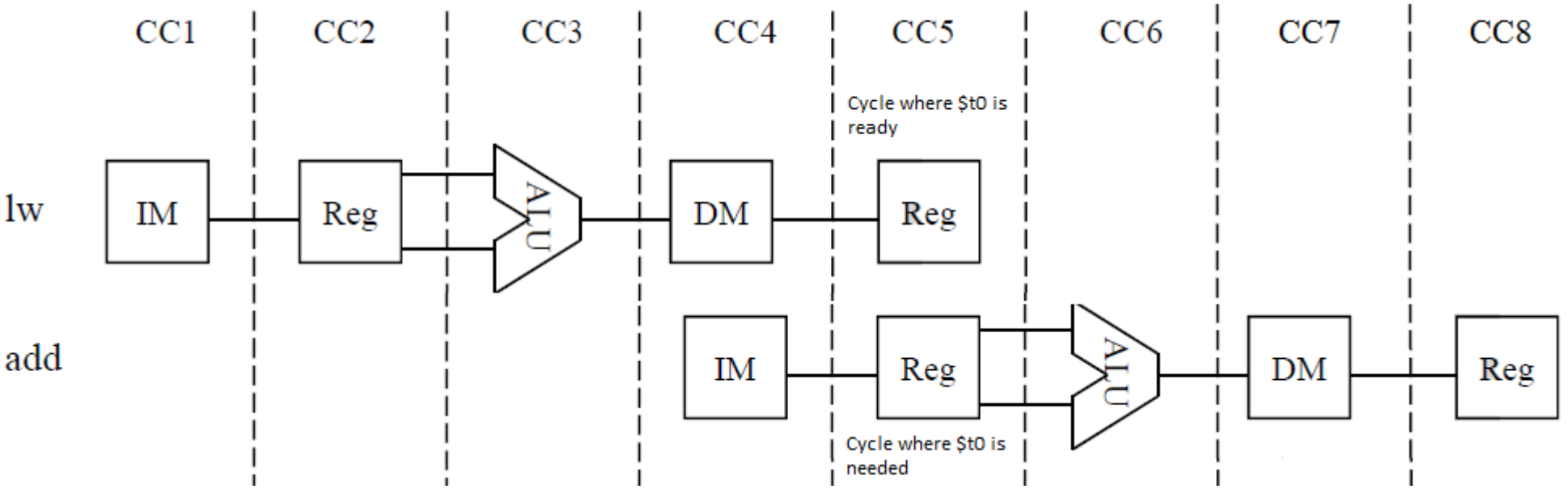
- Our goal is to ensure that the add instruction only reads from the register files after the load has loaded it into the register
- From the software perspective, we won't change the microarchitecture.
- The only thing we can really do is to control the instructions that go into the pipeline.
- If only we had some instruction that did no operation....

Data Hazard: Software Solution

- nop – An instruction that does nothing but take up space.
- Using nop's will allow us to delay the add until the load has loaded to the register.

Data Hazard: Software Solution

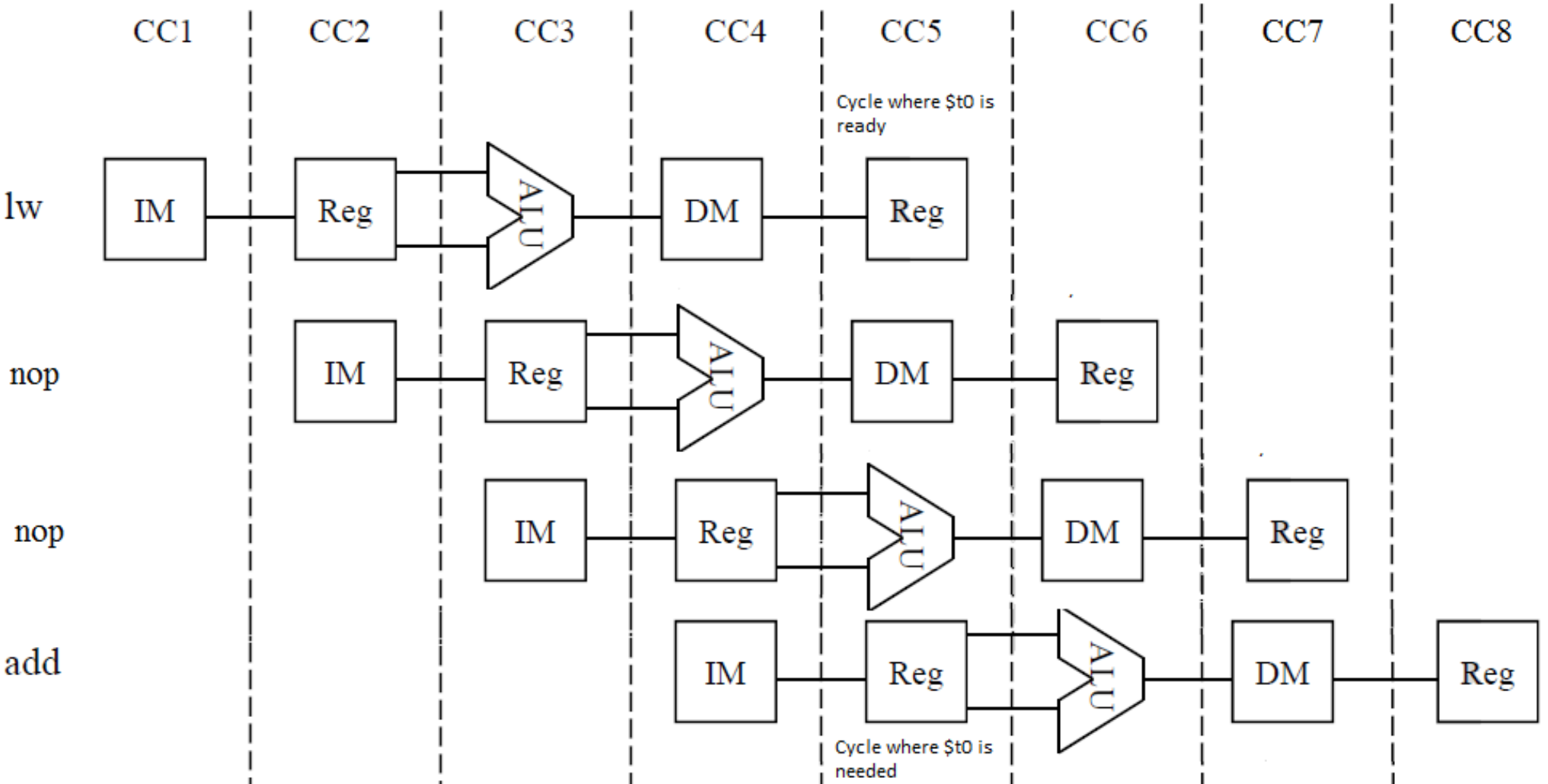
- We want something like:



- How many nops necessary?

Data Hazard: Software Solution

- We need 2 consecutive nops to accomplish this. Will we ever need more?



Data Hazard: Software Solution

- For any instruction that saves something to register, there must be two nops between the instruction that writes to the register and the instruction that reads from the register
- This is for conflicts regarding reading and writing registers. Is there a similar issue with reading and writing to memory?
- Ex.

  1. sw $t0, 4($s0)
  2. lw $t2, 4($s0)

Data Hazard: Software Solution

1. sw $t0, 4($s0)
2. lw $t2, 4($s0)

- The sw will write to memory in MEM stage.
- The lw must read from memory in the MEM stage.
- If the lw instruction succeeds the sw, the lw will not have access to the MEM stage before the sw. No problem!

Data Hazard: Software Solution

- Tradeoffs of using nops?

Data Hazard: Software Solution

- Tradeoffs of using nops?
  - IC goes up because you're inserting instructions.
  - CPI remains 1
  - CT does not concern itself with petty matters of software.
  - Less portable. The number of necessary nops in this pipeline case was 2, but this is not true for all microarchitectures. Therefore, a compiler that inserts nops must know what the microarchitecture is.

Data Hazard: Software Solution

- Need two nops between write and read.
- Of course, more generally, this just means you need any two instructions between the write and read to load the pipeline.
- Perhaps we could move independent instructions between dependent ones.

Data Hazard: Software Solution

- Reordering. For code blocks, reorder the code so that execution remains the same, but fewer or no nops are necessary.

Data Hazard: Software Solution

1. lw $t0, 4($s0)
2. add $t0, $t1, $t0
3. lw $t2, 8($s0)

<REORDER>

1. lw $t0, 4($s0)
2. lw $t2, 8($s0)
3. add $t0, $t1, $t0

- Did it work?

Data Hazard: Software Solution

- Did it work?
  - Sort of... We need one nop now rather than two
- Tradeoffs?

Data Hazard: Software Solution

- Tradeoffs?
  - Reordering alone is not sufficient in all cases. We will need to have some nops in some cases. The most we can really hope for is that reordering will allow us to reduce the amount of necessary nops.
  - With reordering alone, IC stays the same, but since it essentially must be paired with nops, IC will most likely increase.
  - CPI remains 1.

Data Hazard: Software Solution

- Tradeoffs?
  - Can we move this load above this store?
  - 1. sw $t0 0($s0)
  - 2. lw $t1 4($s1)

Data Hazard: Software Solution

- Tradeoffs?
  - Can we move the load above the store?
  - 1. sw $t0 0($s0)
  - 2. lw $t1 4($s1)
  - What if $s0 is 0 and $s1 is -4? That would be a dependency! Sometimes hard to tell if you can move loads above stores.

Data Hazard: Software Solution

- Tradeoffs?
  - Generally speaking, reordering is not always easy or possible.
  - Can only reorder instructions within a non-branching control flow.

Data Hazard: Software Solution

nop only

1. lw $t0, 4($s0)
2. add $t0, $t1, $t0
3. lw $t2, 8($s0)
4. add $t2, $t3, $t2
5. add $t0, $t2, $t0
6. sw $t0, 0($s0)

Data Hazard: Software Solution

nop only

1. lw $t0, 4($s0)
2. add $t0, $t1, $t0
3. lw $t2, 8($s0)
4. add $t2, $t3, $t2
5. add $t0, $t2, $t0
6. sw $t0, 0($s0)

1. lw $t0, 4($s0)
2. nop
3. nop
4. add $t0, $t1, $t0
5. lw $t2, 8($s0)
6. nop
7. nop
8. add $t2, $t3, $t2
9. nop
10. nop
11. add $t0, $t2, $t0
12. nop
13. nop
14. sw $t0, 0($s0)

14 instructions, 18 cycles. Lame. Let's try reordering as well

Data Hazard: Software Solution

Reorder

1. lw $t0, 4($s0)
2. add $t0, $t1, $t0
3. lw $t2, 8($s0)
4. add $t2, $t3, $t2
5. add $t0, $t2, $t0
6. sw $t0, 0($s0)

1. lw $t0, 4($s0)
2. lw $t2, 8($s0)
3. add $t0, $t1, $t0
4. add $t2, $t3, $t2
5. add $t0, $t2, $t0
6. sw $t0, 0($s0)

Data Hazard: Software Solution

Now add nop

1. lw $t0, 4($s0)
2. lw $t2, 8($s0)
3. add $t0, $t1, $t0
4. add $t2, $t3, $t2
5. add $t0, $t2, $t0
6. sw $t0, 0($s0)

1. lw $t0, 4($s0)
2. lw $t2, 8($s0)
3. nop
4. add $t0, $t1, $t0
5. add $t2, $t3, $t2
6. nop
7. nop
8. add $t0, $t2, $t0
9. nop
10. nop
11. sw $t0, 0($s0)

11 instructions, 15 cycles. Eh...

Data Hazard: Software Solution

- Ultimately, the “software solution” consists of nops, which are required to make the code work.
- The reordering is something of an optimization.
- It's a simpler solution than changing the microarchitecture, but maybe it's not a better solution.

Data Hazard: Clever Segue

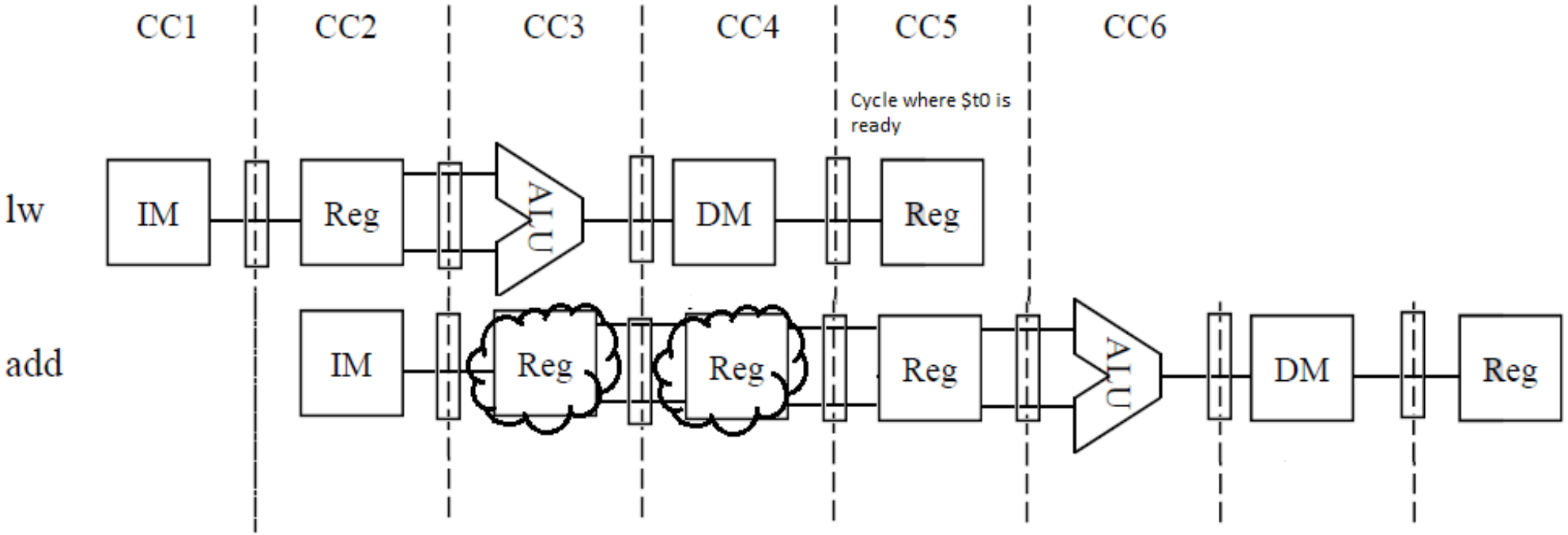
- Instead of changing the applications and instructions, the other alternative is to run the same instructions but alter how each instruction is executed.
- For instance, rather than have the pipeline stall by pushing in instructions that do nothing, simply modify the microarchitecture so that when there's a dependency, pause execution.
- Consider:

  1. lw $t0, 4($s0)

  2. add $t0, $t1, $t0

Data Hazard: Hardware Solution

- Stalling solution.
- The add stalls in Reg(ID) stage after detecting a dependency.



Data Hazard: Hardware Solution

- Notice that this pure stalling solution is similar to the pure nop solution since they both aim to delay the instruction long enough so that the read from register happens on the same cycle as the write to register
- As a result, we expect the latency to be the same between pure stalling and pure nop.

Data Hazard: Hardware Solution

stalls only

1. lw $t0, 4($s0)
2. add $t0, $t1, $t0
3. lw $t2, 8($s0)
4. add $t2, $t3, $t2
5. add $t0, $t2, $t0
6. sw $t0, 0($s0)

18 cycles total...

1. 5 cycles
2. 1 cycle + 2 cycles stalls.
3. 1 cycle
4. 1 cycle + 2 cycles stall.
5. 1 cycle + 2 cycles stall
6. 1 cycle + 2 cycles stall

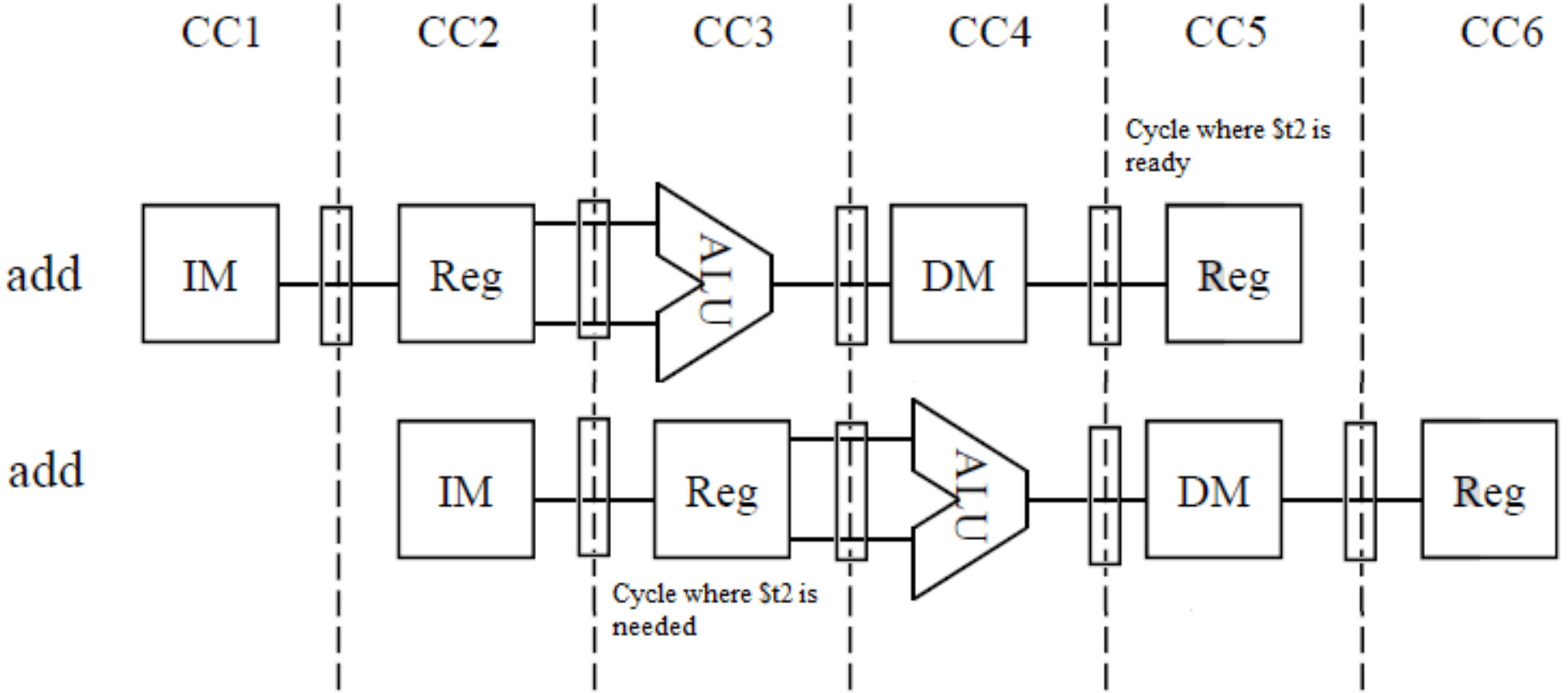
Data Hazard: Hardware Solution

- That's not really any better. In the software, we could also reorder to optimize.
- However, the hardware also has a few other tricks up its sleeve.
- Consider

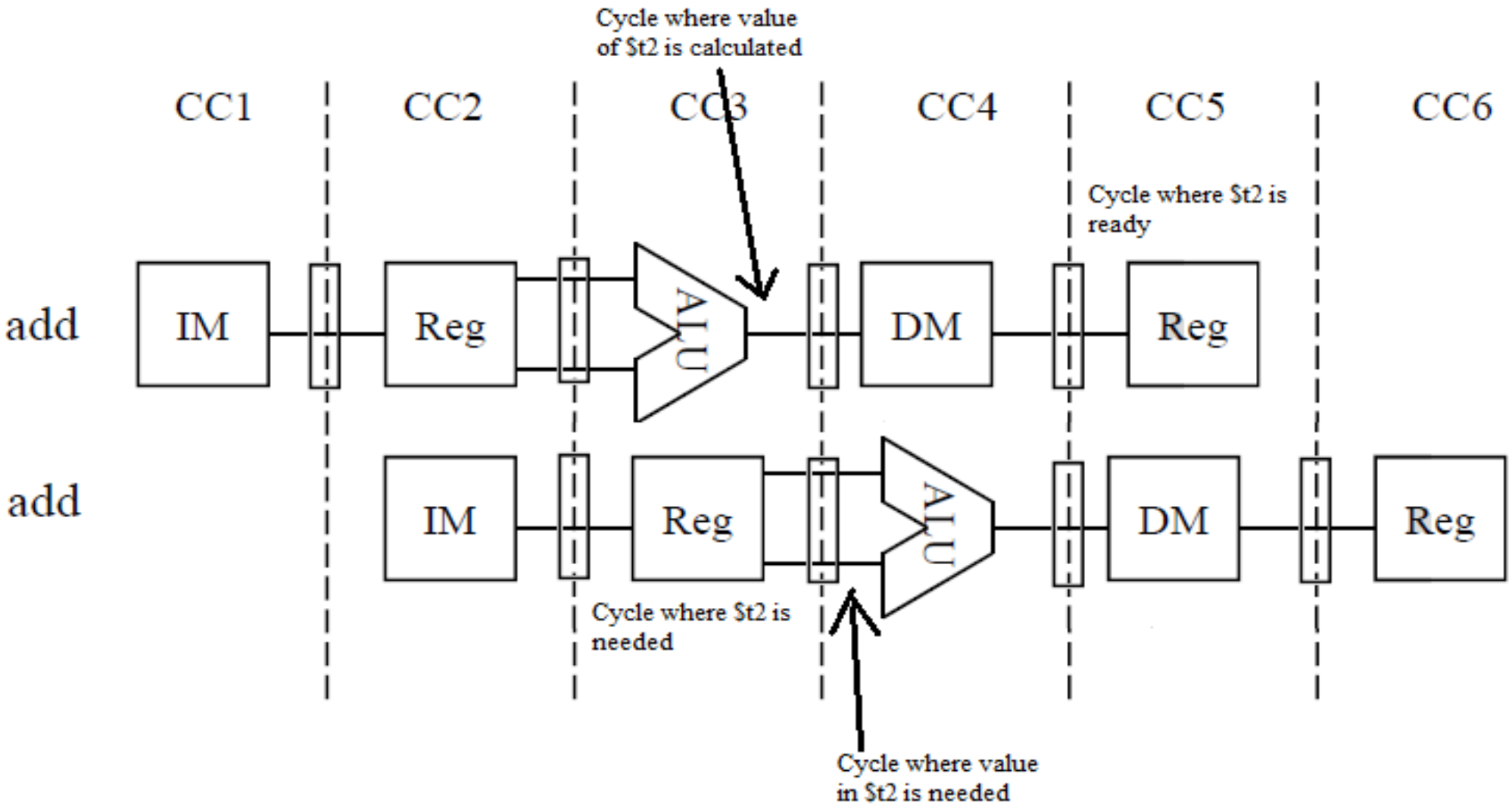
  add $t2, $t3, $t2

  add $t0, $t2, $t0

Data Hazard: Hardware Solution



Data Hazard: Hardware Solution



Data Hazard: Hardware Solution

- In reality, the calculated value is technically ready by the time it is needed. As a solution, we can have additional hardware to “forward” information.
- You're forwarding information from an instruction to subsequent instructions.
- You're moving data backwards in the pipeline.
- We do this by storing additional information in the EX/MEM and MEM/WB latches.

Data Hazard: Hardware Solution

- If we have the following set of instructions:

  1. [R-Type] $t0, $t1, $t1

  2. [R/I-Type] $t2, $t0, $t0

  ...this can be resolved by forwarding information (Rd, RegWrite, calculated value) from EX/MEM stage (instruction 1) to the EX stage of instruction 2.

Data Hazard: Hardware Solution

- If we have the following set of instructions:

  1. [R-Type] $t0, $t1, $t1
  2. <Other independent instruction>
  3. [R/I-Type] $t2, $t0, $t0

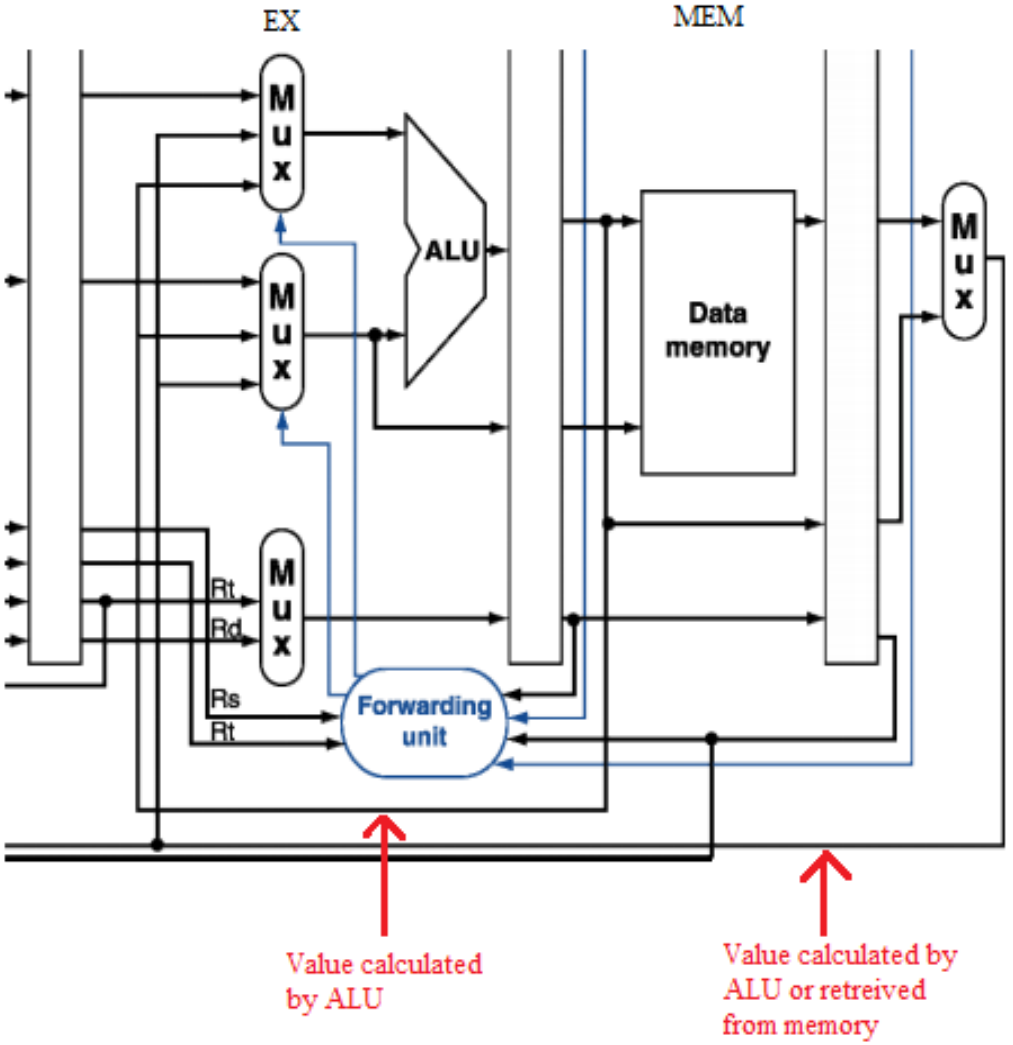
  ...this can be resolved by forwarding information (Rd, RegWrite) from MEM/WB stage (instruction 1) to the EX stage of instruction 1.

Data Hazard: Hardware Solution

- When do we need to forward?
  - Previous instruction is writing to register (EX/MEM.RegWrite or MEM/WB.RegWrite is 1)
  - Previous instruction's destination register (EX/MEM.Rd or MEM/WB.Rd) is the same as one of the current instruction's source register (ID/EX.Rt or ID/EX.Rs)
  - Previous instruction's destination register is not the zero register.

Data Hazard: Hardware Solution

- Forward values by passing data from later stages to the ALU of earlier stage



Data Hazard: Hardware Solution

- Note:
  - add $t0, $s0, $s1
  - add $t1, $s2, $s3
  - add $t2, $t0, $t1
  - We will need to be able to forward from two instructions.
- Note 2:
  - add $t0, $s0, $s1
  - add $t0, $s2, $s3
  - add $t2, $t0, $t1
  - The last add depends on the action of the second add, not the first. Prioritize forwarding from nearest instruction.

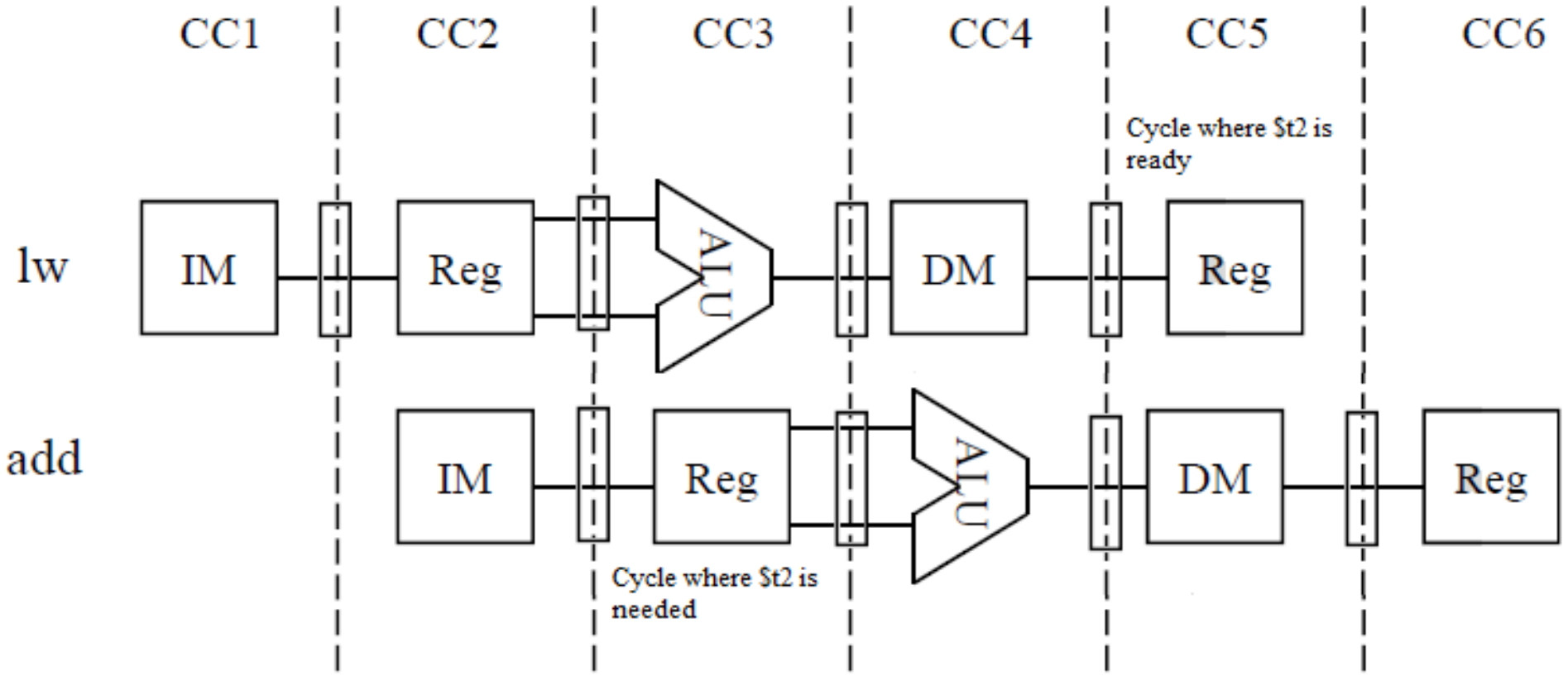
Data Hazard: Hardware Solution

- Now consider

  lw $t2, 0($t3)

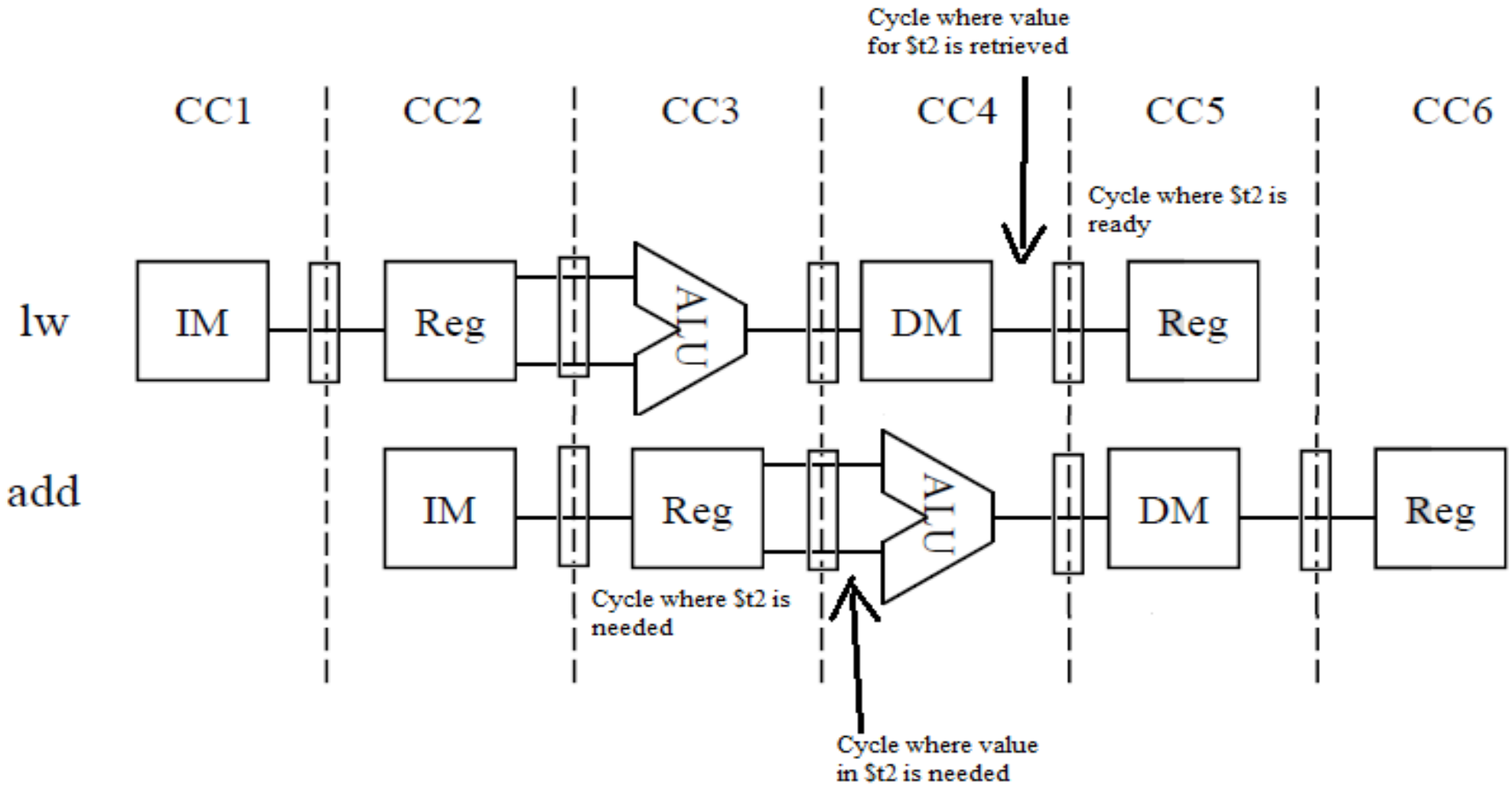
  add $t0, $t2, $t0

Data Hazard: Hardware Solution

- 

Data Hazard: Hardware Solution

- Whoops...



Data Hazard: Hardware Solution

- Typically the result of a module is stored in the subsequent latch. The result of MEM is stored in the MEM/WB latch. Then you'd have to forward it from that latch.
- It seems though, that you could also simply do the forwarding at the end of the MEM stage.
- This will may increase the CT since then the EX stage could then have to wait for an entire MEM stage before being able to start.
- Let's just stall.
- What are the conditions for stalling here?

Data Hazard: Hardware Solution

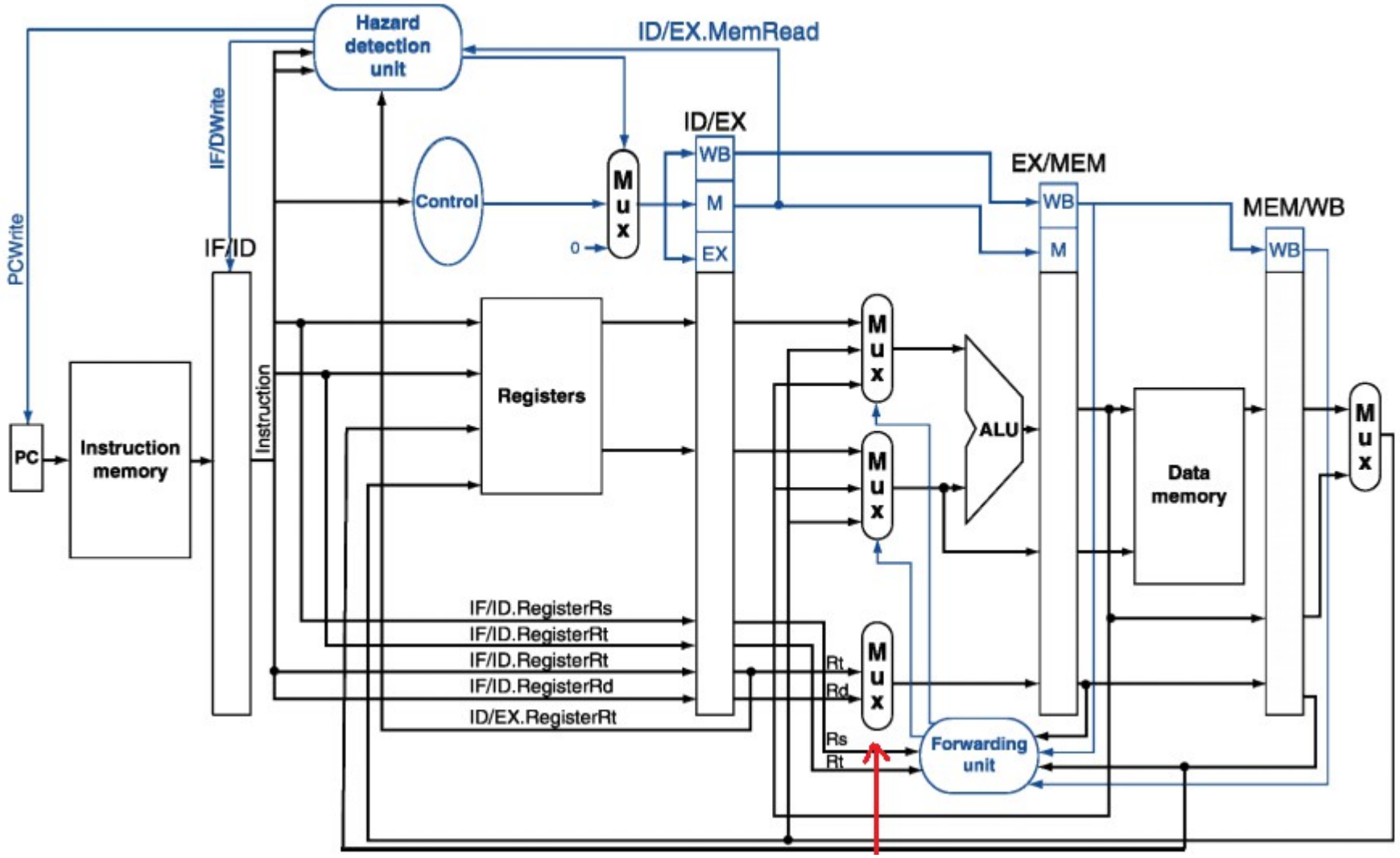
- What are the conditions for stalling here?
  - Recall this is a specialized case for loads
  - If the load and dependent instruction have an instruction between then, we don't need to stall.
  - We can just use the forwarding mechanism for R-Types.
  - ...or can we?

Data Hazard: Hardware Solution

- The previous mechanism for forwarding from an R-Type checked for the instructions Rd register.
- The destination of load is an Rt register.
- But it's okay.

Data Hazard: Hardware Solution

- The forwarding mechanism actually considers Rd and Rt based on RegDst



Data Hazard: Hardware Solution

- What are the conditions for stalling here?
  - We can check if we need to stall an instruction in the ID stage since we won't know before then what registers we'll be using.
  - We need to stall if current source registers (IF/ID.Rs or IF/ID.Rt) is the same as the previous instruction's destination register (IF/EX.Rt).
  - Also check ID/EX.MemRead. Do we also need to check ID/EX.RegWrite?

Data Hazard: Hardware Solution

- What are the conditions for stalling here?
  - Also check ID/EX.MemRead. Do we also need to check ID/EX.RegWrite?
  - Not in our current implementation where the only way to read from memory is via lw and lw will always write to a register.

Data Hazard: Hardware Solution

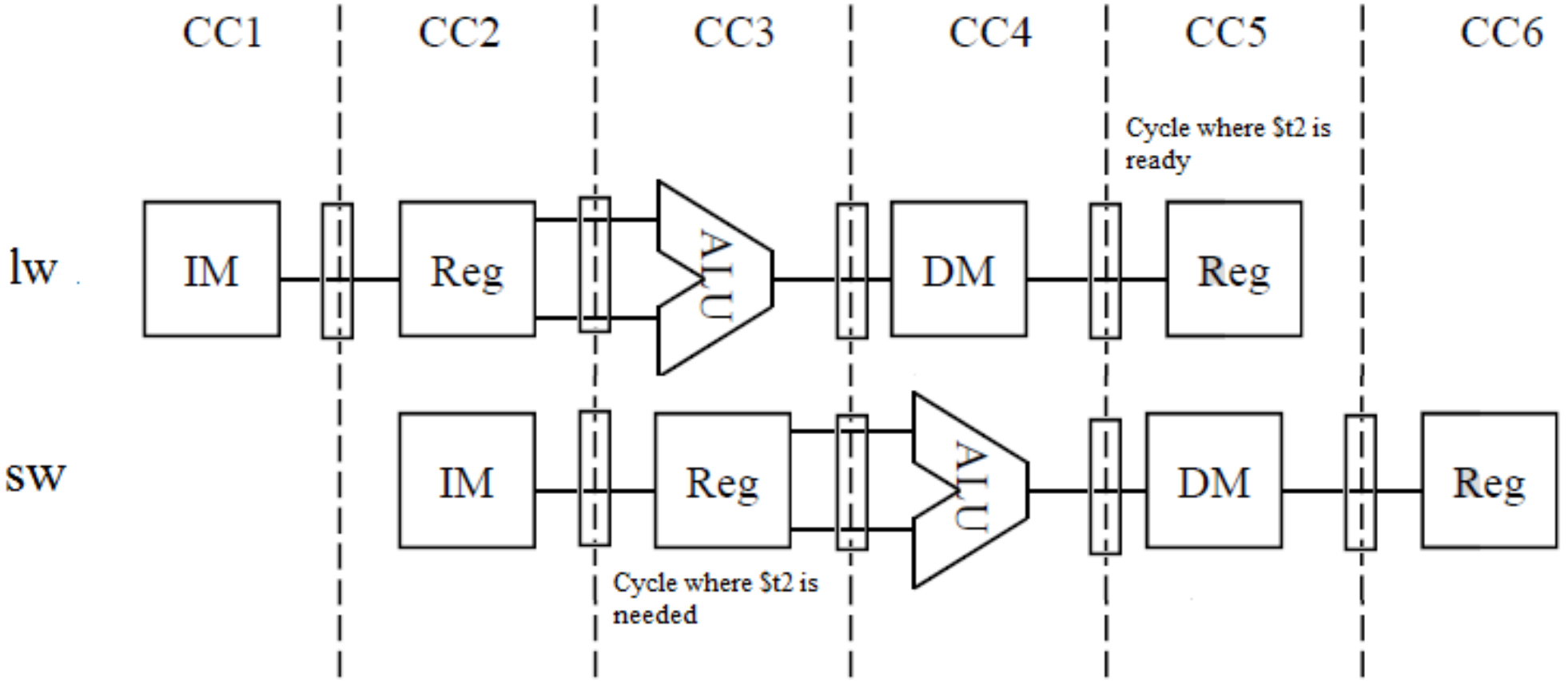
- Insert bubble by:
  - Pushing all 0's as control to the ID/EX latch.
  - PCWrite = 0 (don't get new instruction).
  - IF/IDWrite = 0 (instruction stored in IF/ID remains and doesn't get pushed out by new instruction).

Data Hazard: Hardware Solution

- Are we out of the woods yet?
- Consider:
- lw $t2, ($s1)
- sw $t2, ($s0)

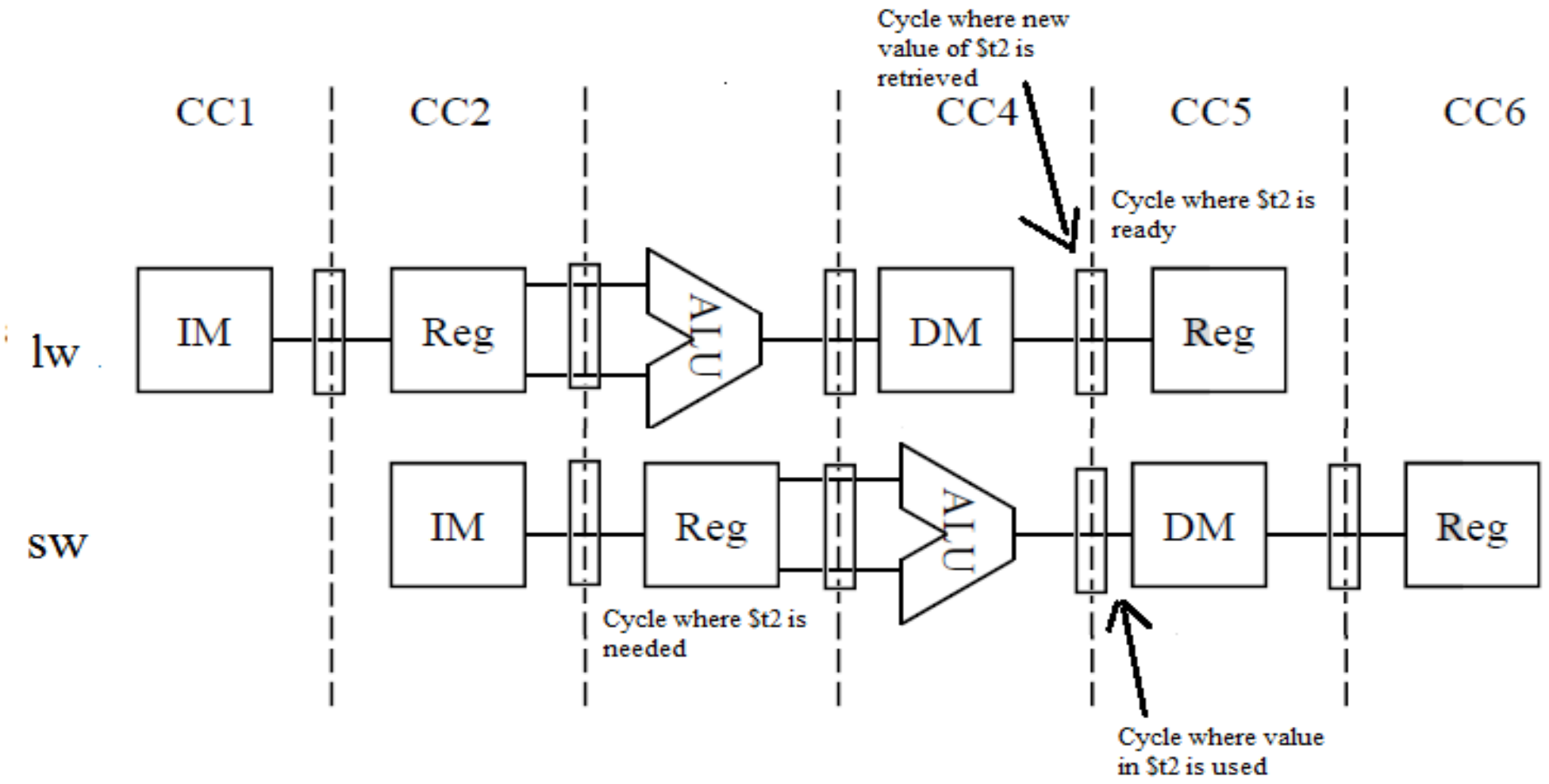
Data Hazard: Hardware Solution

-



Data Hazard: Hardware Solution

- We can't yet pass from MEM/WB to MEM stage.
- We will need to stall or add new forwarding logic



Data Hazard: Hardware Solution

- Quick tradeoff aside:
- Even for this very reduced set of instructions, we have to consider all of the permutations of instructions types and add logic to the pipeline to get each to work.
- Imagine doing this for a CISC machine.
- With a single cycle datapath, it was fairly easy to add new and complicated instructions, but it's much more difficult in the pipeline.

Data Hazard: Hardware Solution

- From here on out, we assume that we have full forwarding.
- Even full forwarding isn't good enough to handle loads followed by dependent instructions so we will need one cycle stalling.

Data Hazard: Hardware Solution

1. lw $t0, 4($s0)
2. add $t0, $t1, $t0
3. lw $t2, 8($s0)
4. add $t2, $t3, $t2
5. add $t0, $t2, $t0
6. sw $t0, 0($s0)

Data Hazard: Hardware Solution

1. lw $t0, 4($s0)
2. add $t0, $t1, $t0
3. lw $t2, 8($s0)
4. add $t2, $t3, $t2
5. add $t0, $t2, $t0
6. sw $t0, 0($s0)

- Identify actual control flow (branches taken or not taken)
- Identify dependencies
- If loads are followed by a dependent, the dependent will need to stall in ID stage.
- Consult example...

End of

The Sixth Week

-Four Weeks Remain-