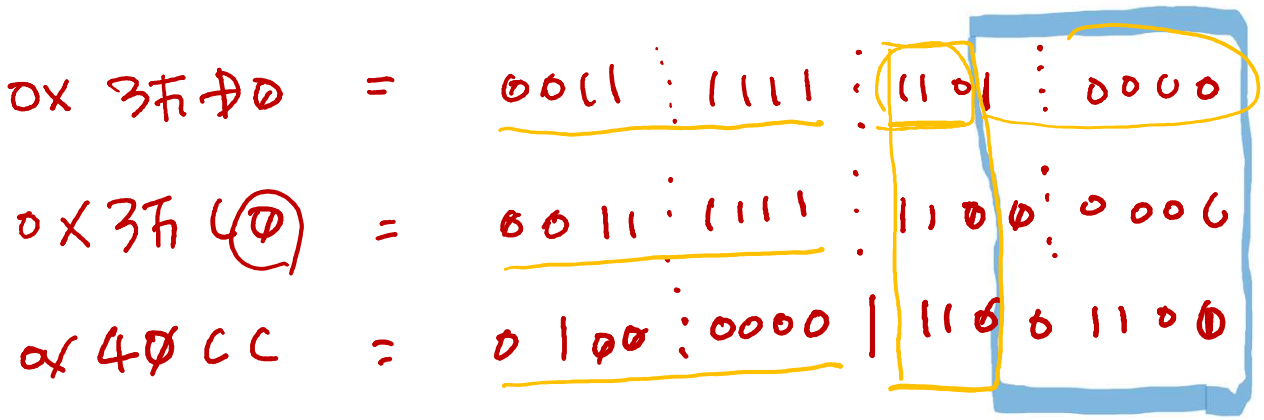
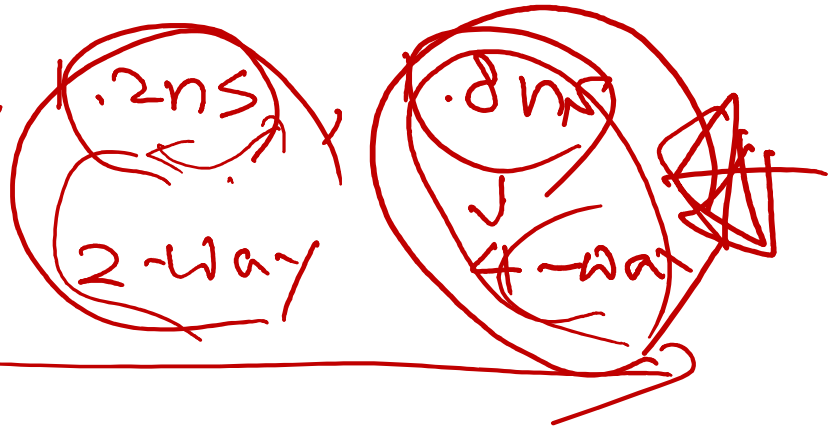

0x 3FD0 = 0011 1111 1101 0000
0x3FD6 = 0011 1111 1101 0110
0x40CC = 0100 0000 1100 1100

1. 1 ns → DM



1 GHz = ⎍ (1 ns)

a 1

b 2

c 2

2. AMAT = (2) + 0.1 x 100 cycles

↳ 2x

5.

6 bit to identify one byte in a set

a. 10 bits to id one set in DM cache

b. 8 bits to id one set in 4-way cache.

a) 10+6

b 8+6

6. a) 32 − 15 = 17 bits ⇒ tag.

b) 32 − 13 = 19 bits × 4

1. $\log_2(32) = 5$ bits

2. $32 \times 2^{10} = 2^{15}/2^5 = 2^{10}$ lines $= 1024$

3 a) 1024 sets

b) $1024/4 = 256$ sets

4.

a) $\log_2(1024) =$ 10 bits

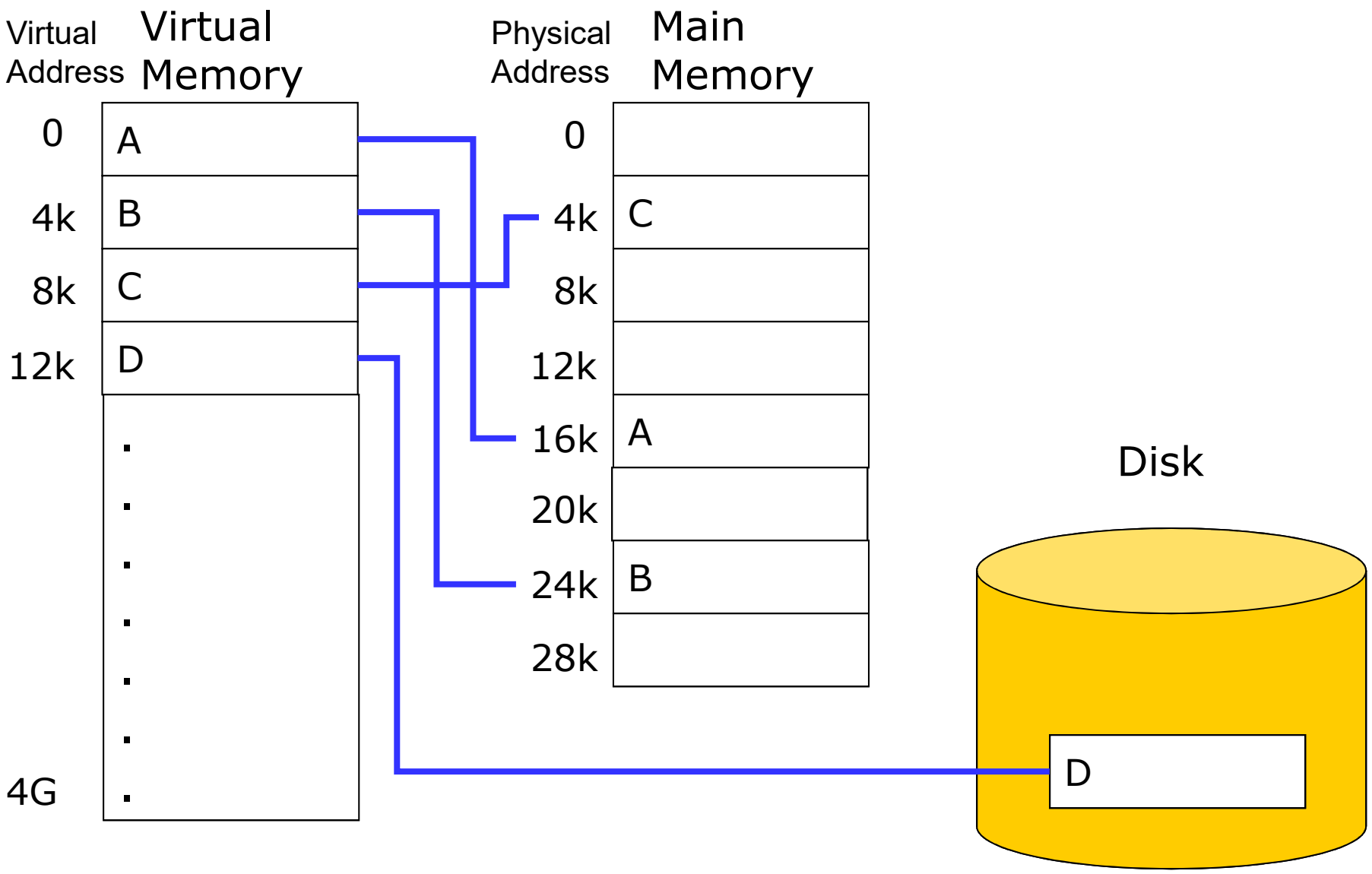
b) $\log_2(256) = 8$ bits

Lecture 8:
Pipeline Overview

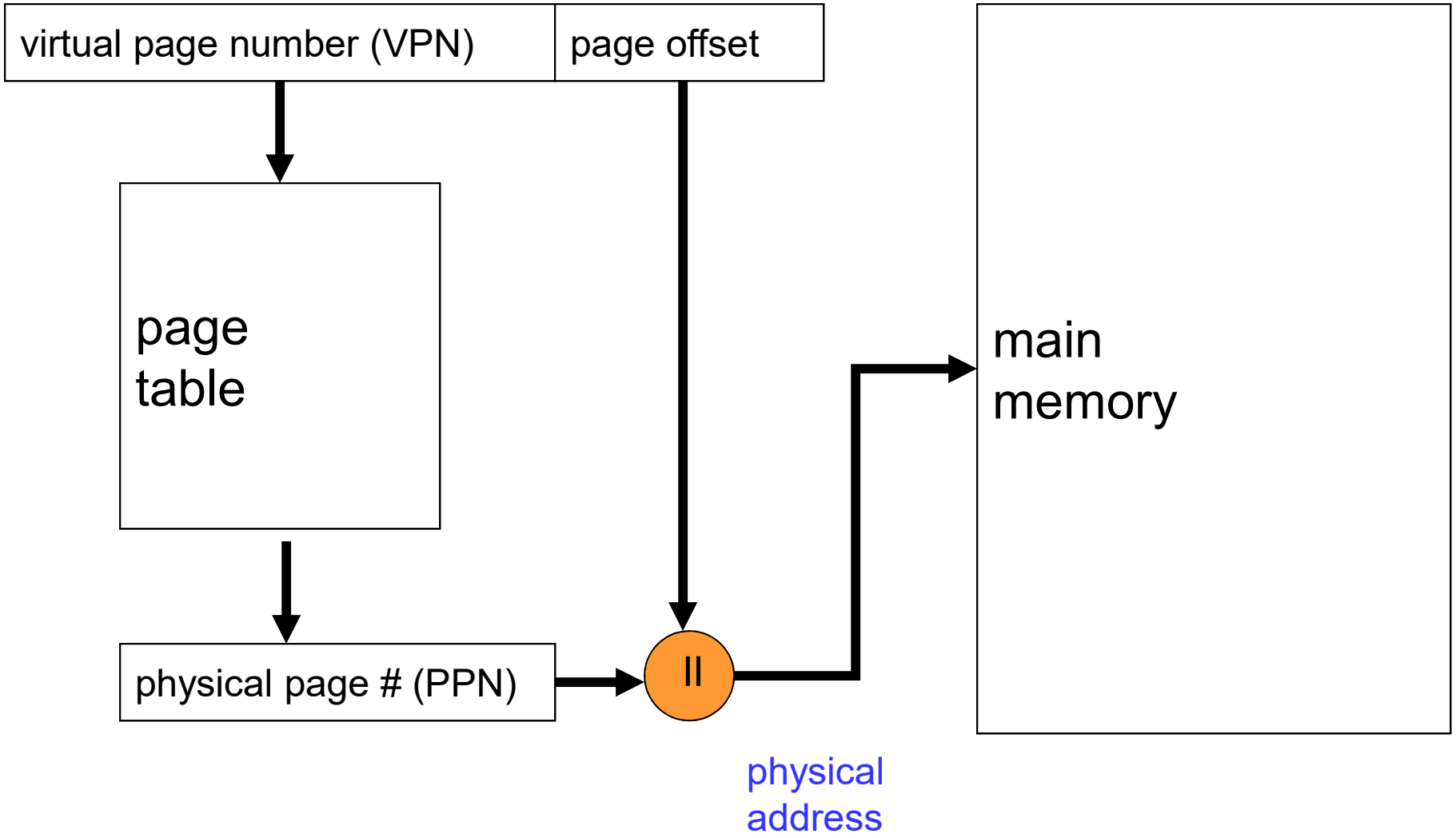
Review: Advantages of Virtual Memory

- translation
  - ✓ program can be given consistent view of memory, even though physical memory is scrambled
  - ✓ only the most important part of program ("working set") must be in physical memory
  - ✓ contiguous structures (like stacks) use only as much physical memory as necessary yet grow later
- protection
  - ✓ different threads (or processes) protected from each other
  - ✓ different pages can be given special behavior
    - (read only, invisible to user programs, etc.).
  - ✓ kernel data protected from user programs
  - ✓ very important for protection from malicious programs
    - far more "viruses" under Microsoft Windows
- sharing
  - ✓ map the same physical page to multiple users ("shared memory")

Review: Virtual vs. Physical Address Space

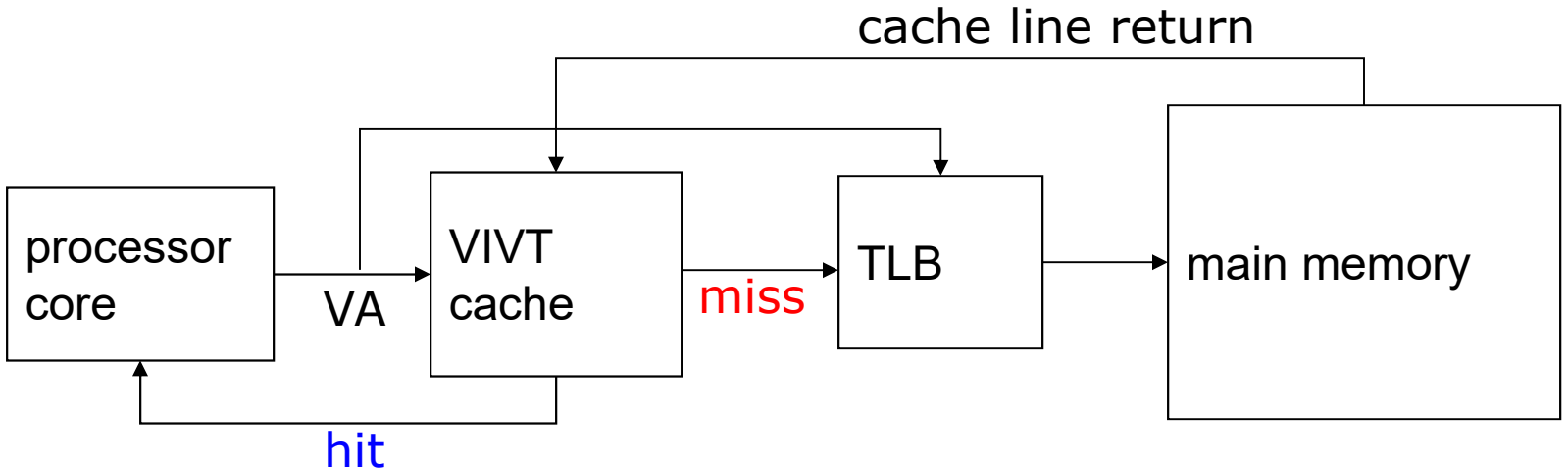


Review: Page Table and Address Translation



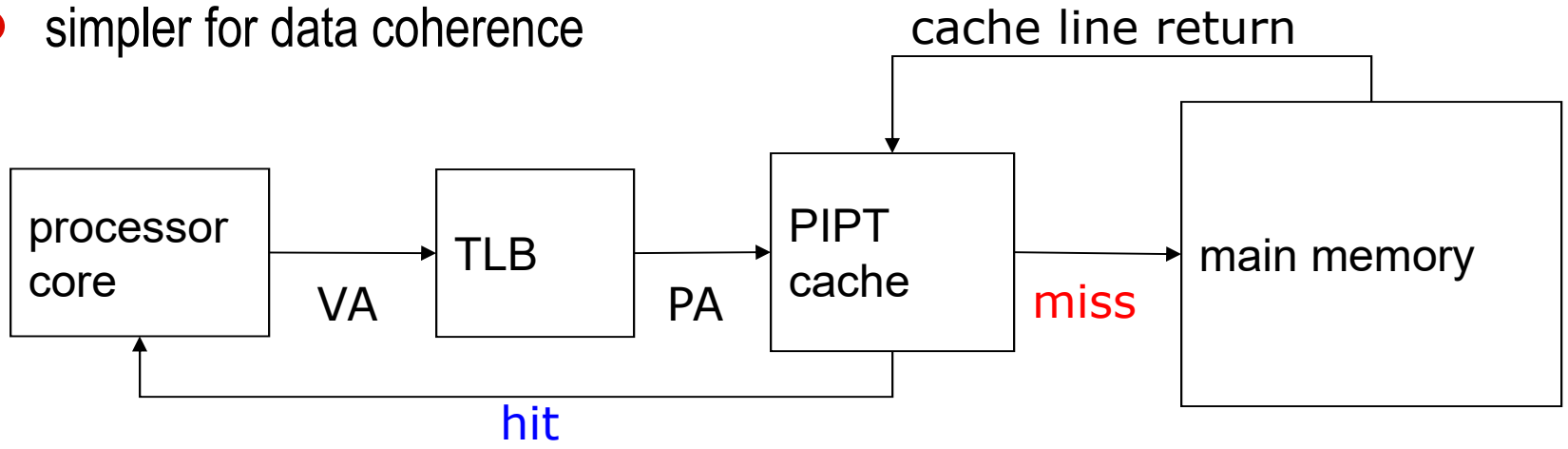
Review: Virtually-Indexed Virtually-Tagged

- fast cache access
- only require address translation when going to memory (miss)
- issues?



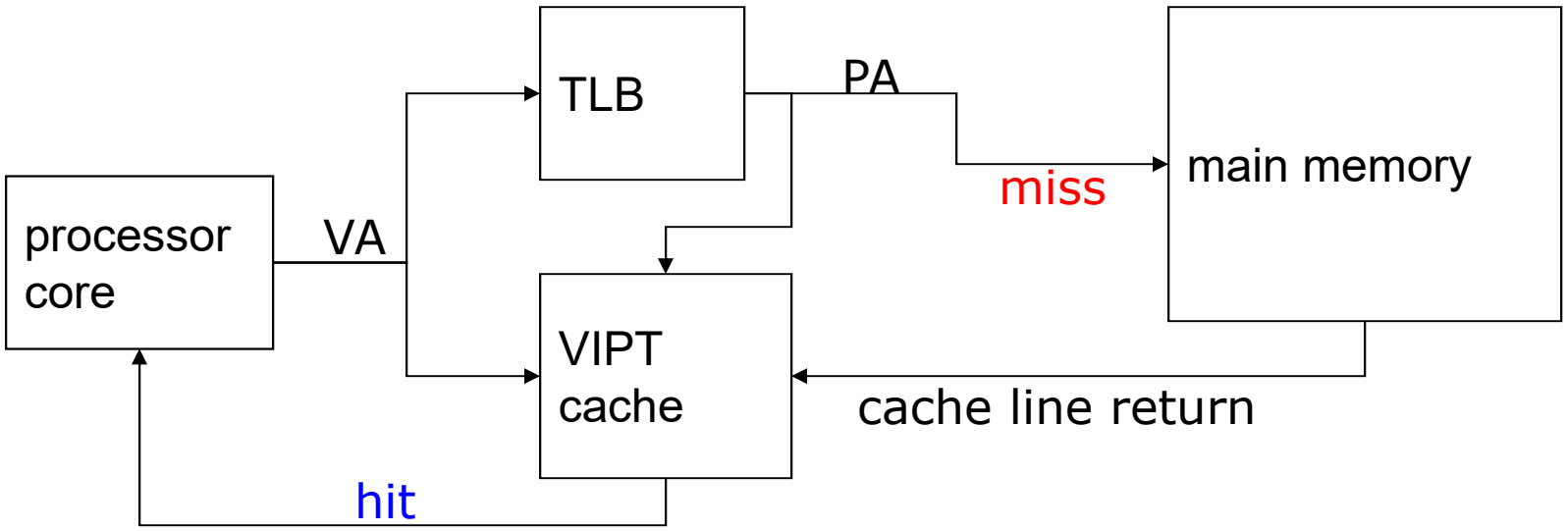
Review: Physically-Indexed Physically-Tagged

- slower, always translate address before accessing memory
- simpler for data coherence

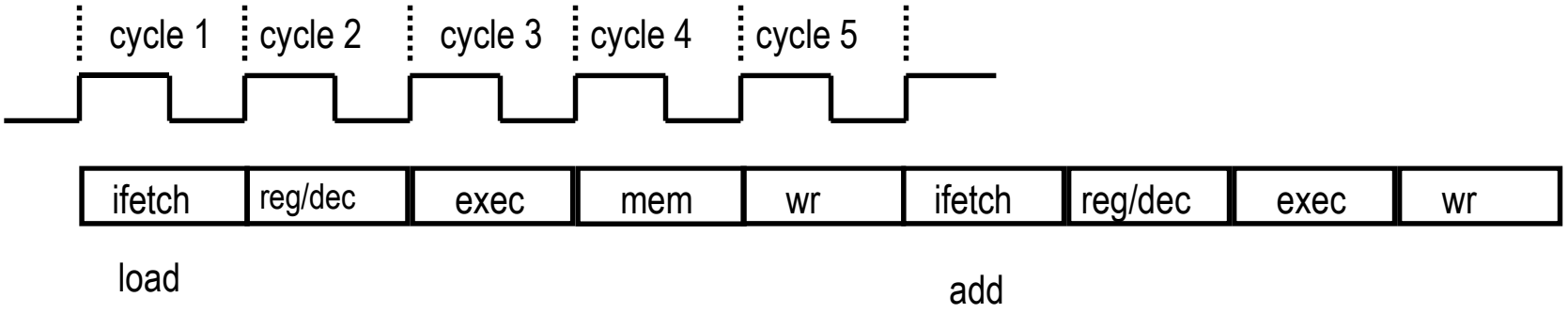


Review: Virtually-Indexed Physically-Tagged

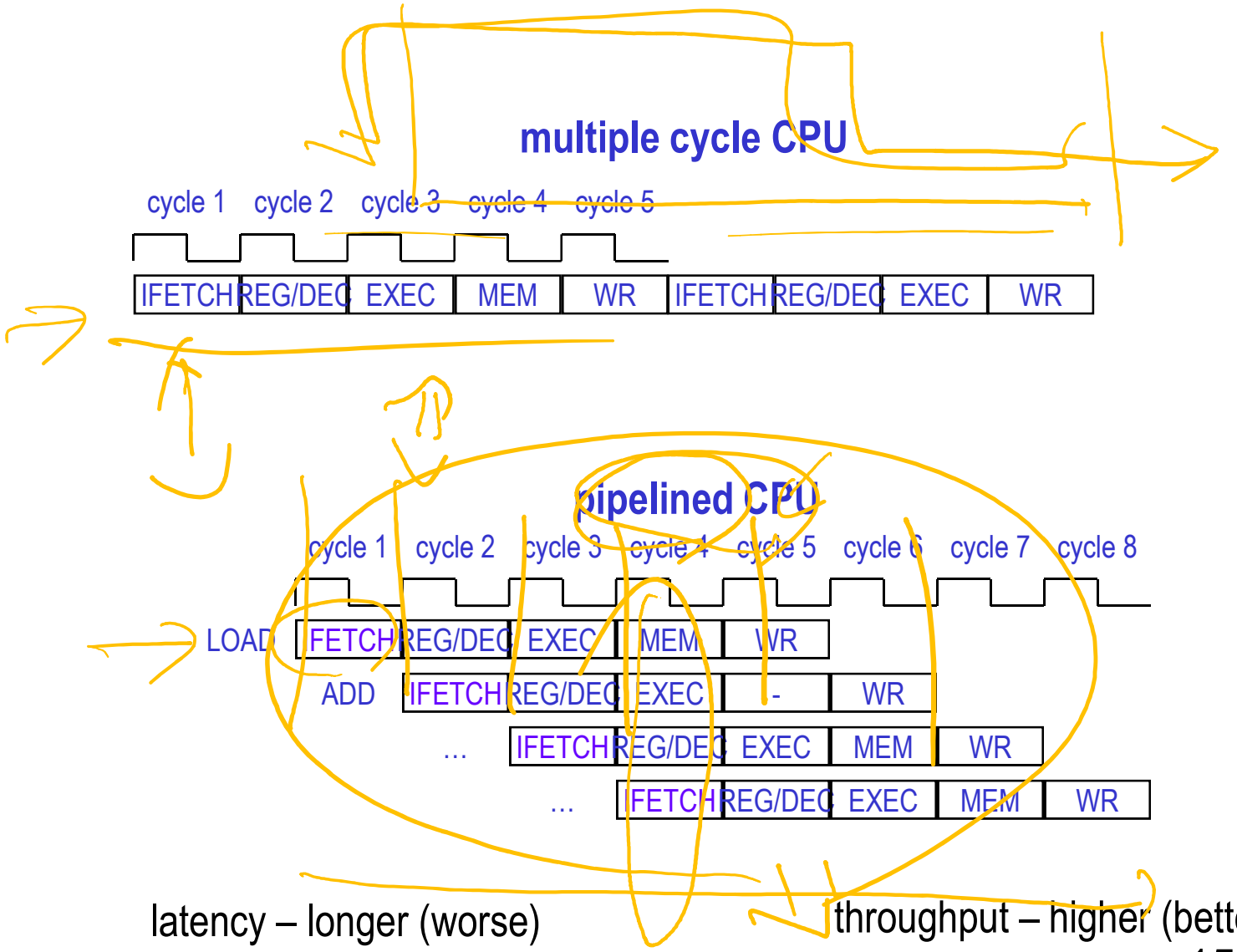
- gain benefit of a VIVT and PIPT
- parallel access to TLB and VIPT cache
- no homonym
  - ✓ how about synonym?



Review -- Instruction Latencies



Instruction Latencies and Throughput



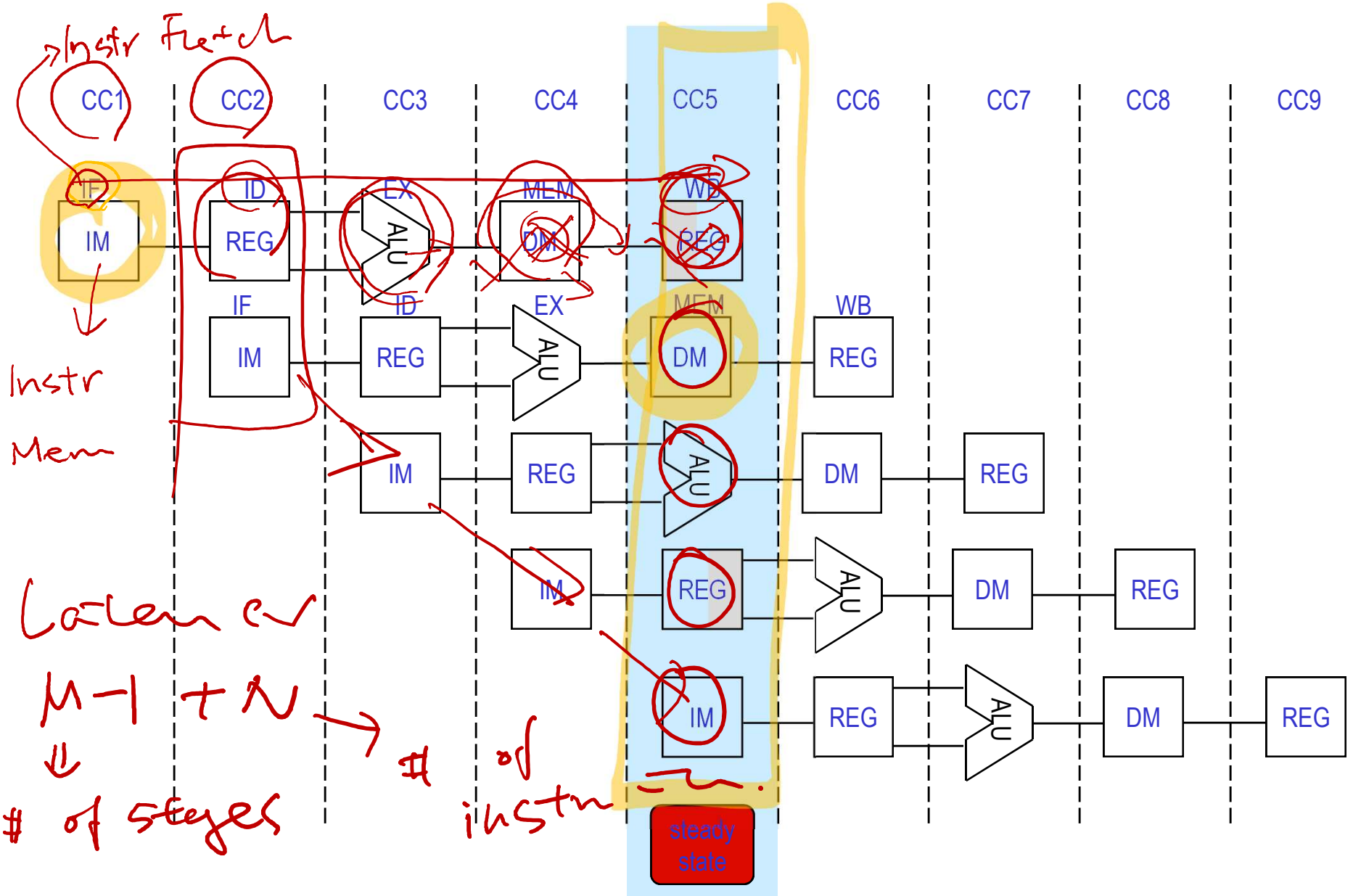
latency – longer (worse)

throughput – higher (better)

Pipelining - Analogy

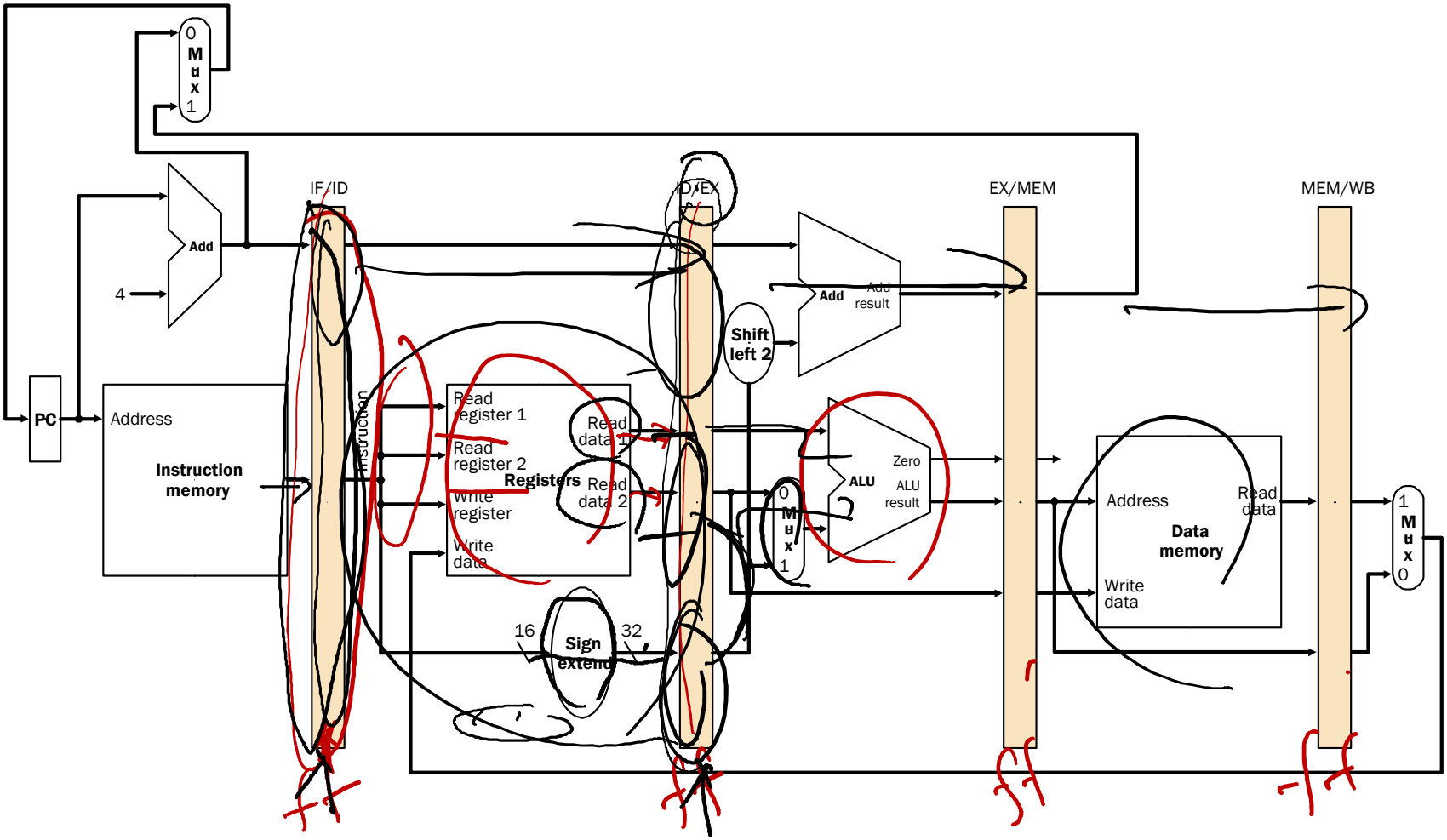
- https://www.youtube.com/watch?v=ecCt6HPlPeA

Pipelined Execution Timing



Execution in a Pipelined Datapath

instruction fetch | instruction decode/ register fetch | `sub $15, $4, $1` | `lw $12, 1000($4)` | `add $10, $1, $2`



Speed up calculation

- assuming the stages are perfectly balanced
  - ✓ time between instructions (pipelined) =
  - ✓ time between instructions (non-pipelined) / # of stages =

⇓

single cycle m.

Pipelining Example 1

- a processor that takes 5ns to execute an instruction is pipelined into 5 equal stages. the latch between each stage has a delay of 0.25 ns.
  - ✓ what is the minimum clock cycle time of the pipelined processor?
    - 5/5+0.25 = 1.25 ns
  - ✓ what is the maximum speedup that can be achieved by this pipelined processor? (compared to the original processor)
    - 4× (1 instr every 1.25ns vs 1 instr every 5 ns)
  - ✓ can we have much deeper pipelining?
    - if we divide into 10 stages, the clock will be 0.75 ns and the speedup will at most 6.7×, diminishing return!

Pipelining Example 2

- a non-pipelined processor takes 5ns to execute an instruction. if I want clock the processor at 2GHz, how many stages should I pipeline this processor into if each latch has a 0.25ns delay?
  - ✓ what is the maximum speedup that can be achieved by the pipelined processor running at 2GHz? (compared to the original single cycle processor)
  - ✓ what is the average latency of an instruction?
  - ✓ how many stages if I want to clock the processor at 5GHz?

$$\frac{5ns}{X} + 0.25ns = 0.5ns$$

$$0.25ns$$

$$\frac{5}{0.25} = X \quad \text{20 stages.}$$

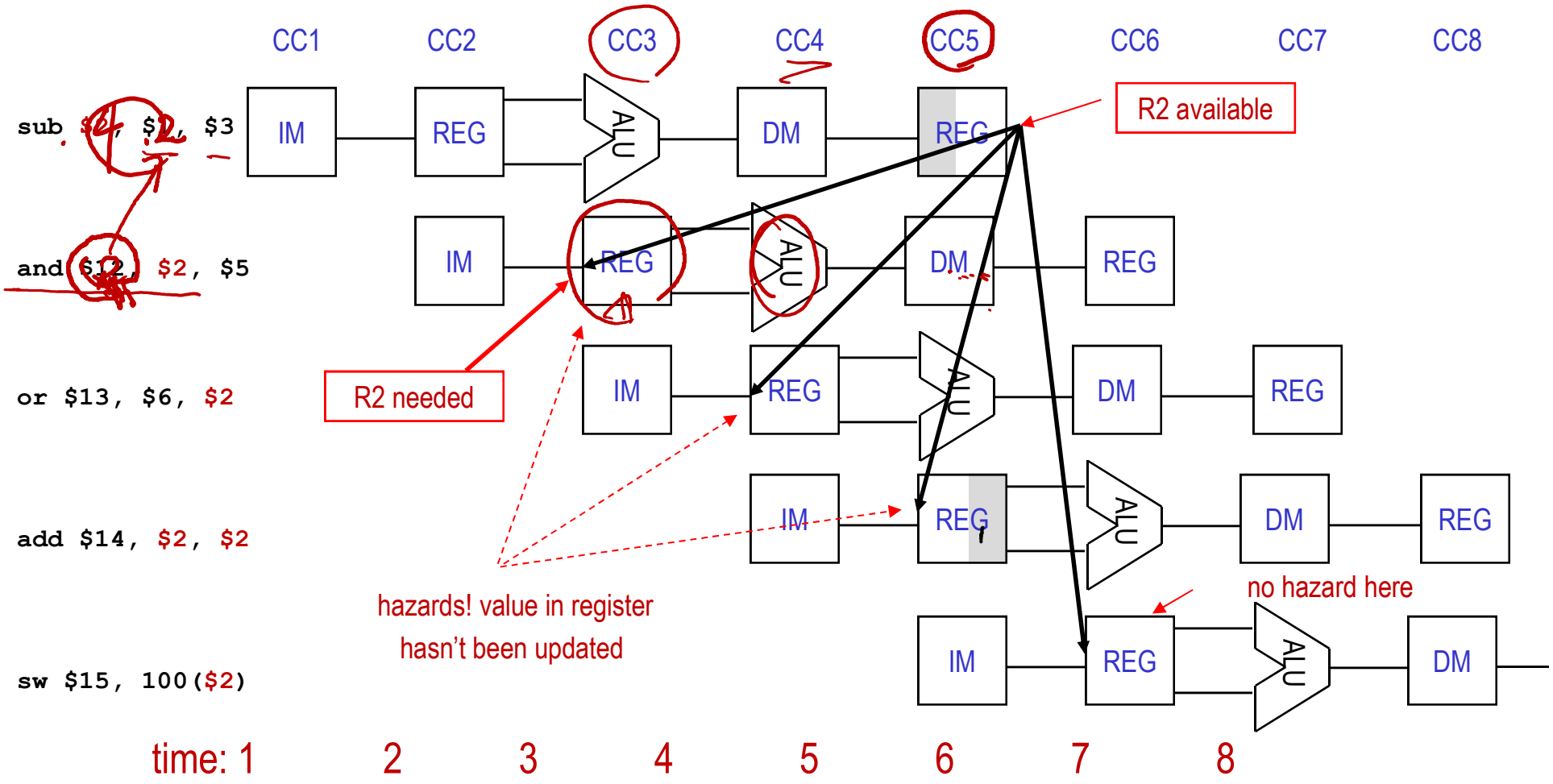
Hazards

- situations that prevent starting the next instruction in the next cycle
  - ✓ structure hazards
    - a required resource is busy
  - ✓ data hazard
    - need to wait for previous instruction to complete its data read/write
  - ✓ control hazard
    - deciding on control action depends on previous instruction

Data Hazards

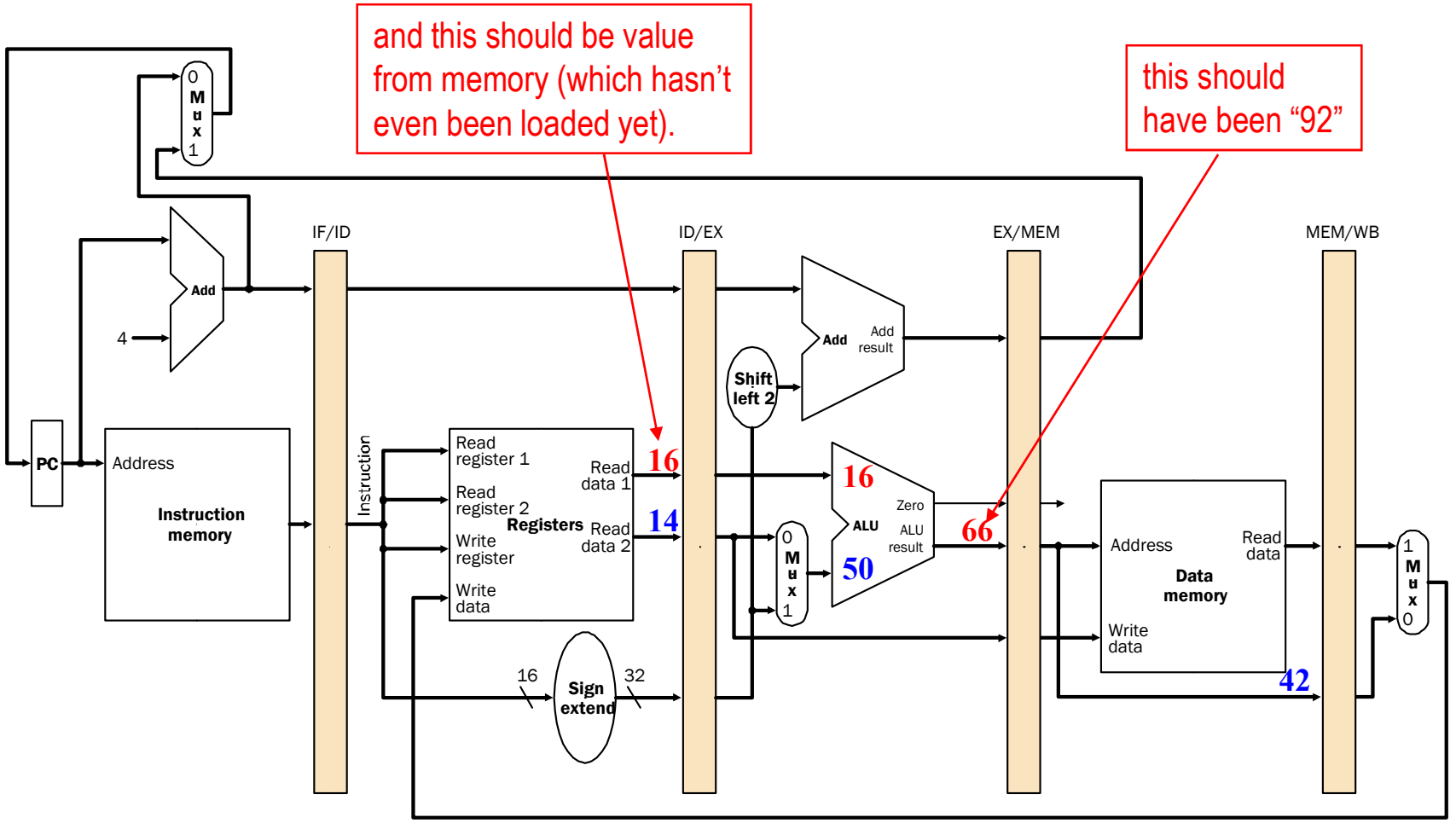
revisit on Thursday.

- when a result is needed in the pipeline before it is available, a data hazard occurs

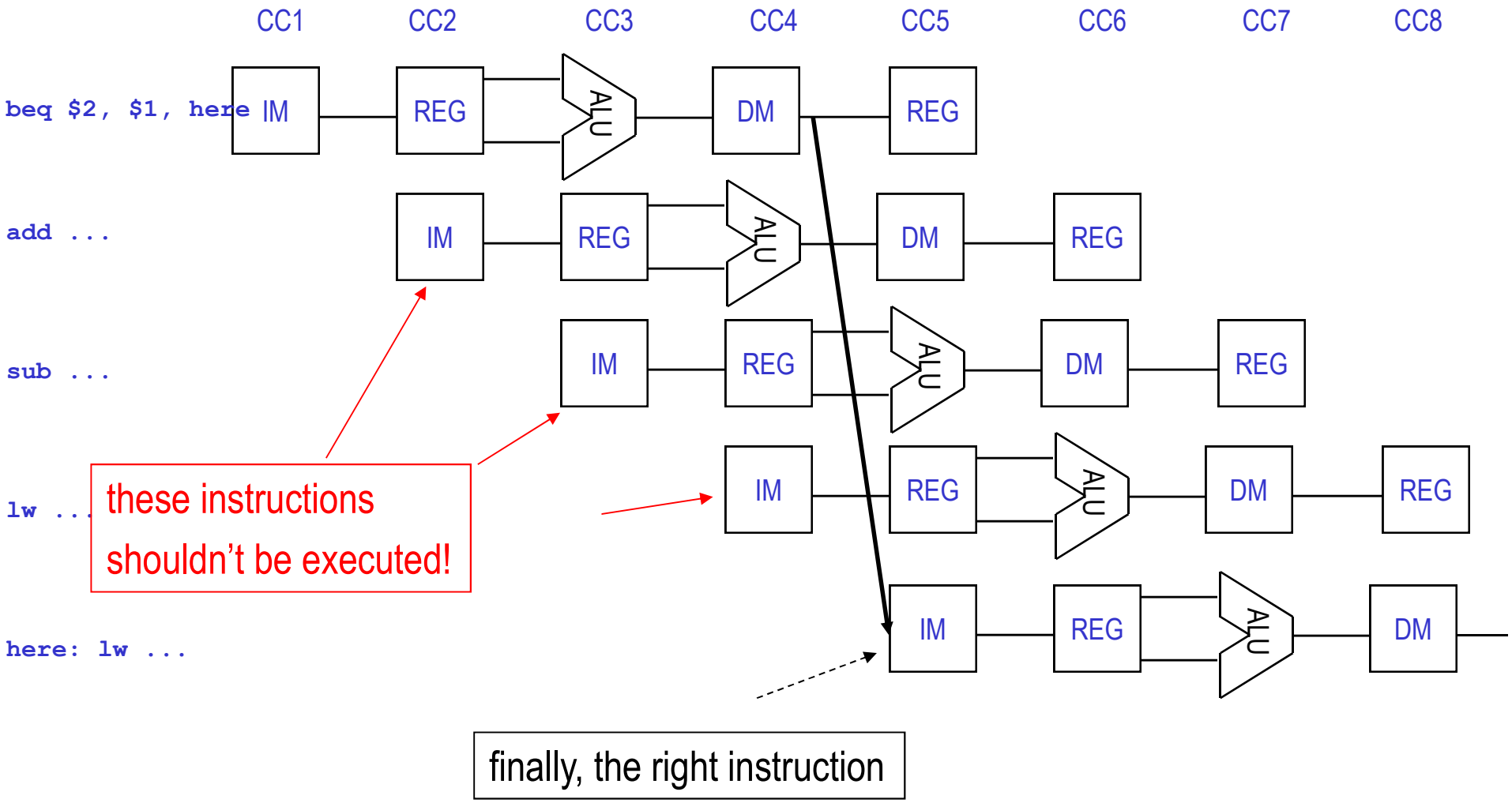


Data Hazard – Data Path View

```
add $10, $1, $2    add $11, $8, $7    lw $8, 52($3)    add $3, $10, $11    write back
```

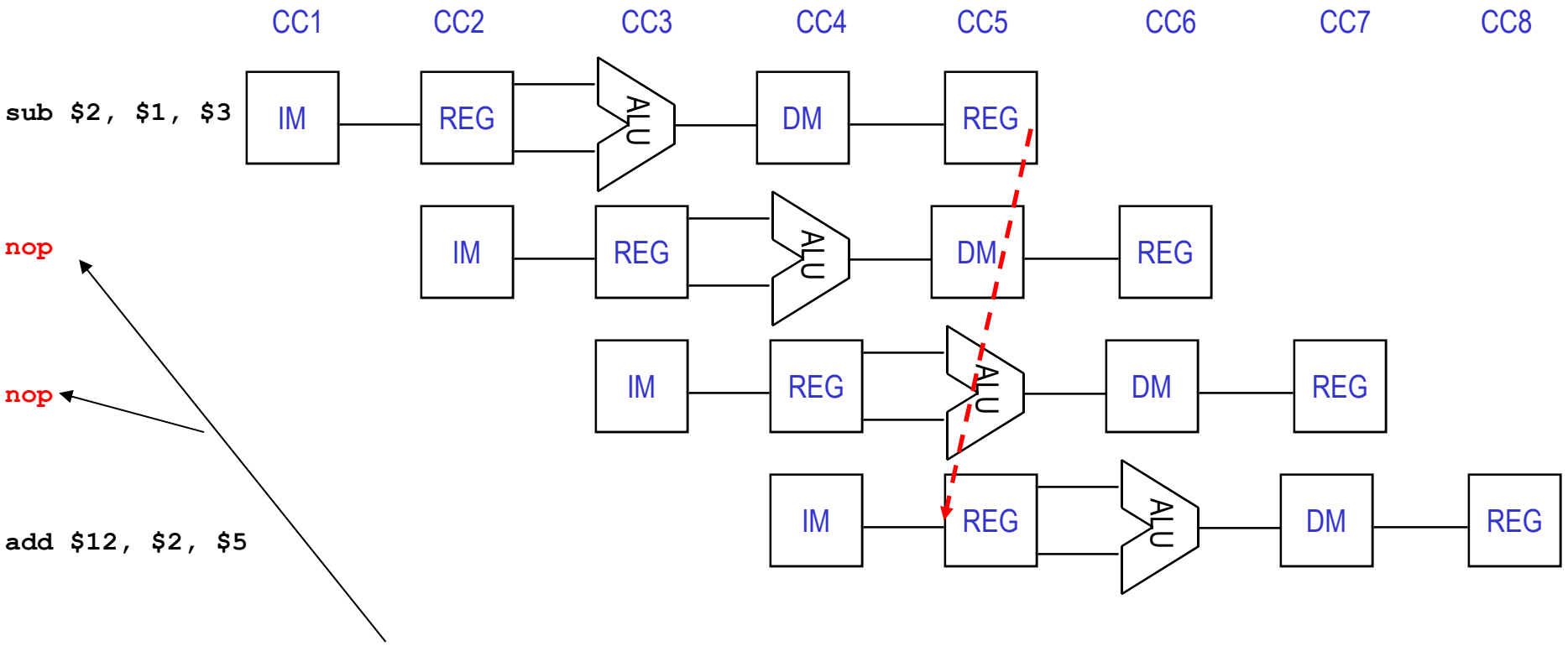


Branch Hazards



Dealing with Data Hazards in Software

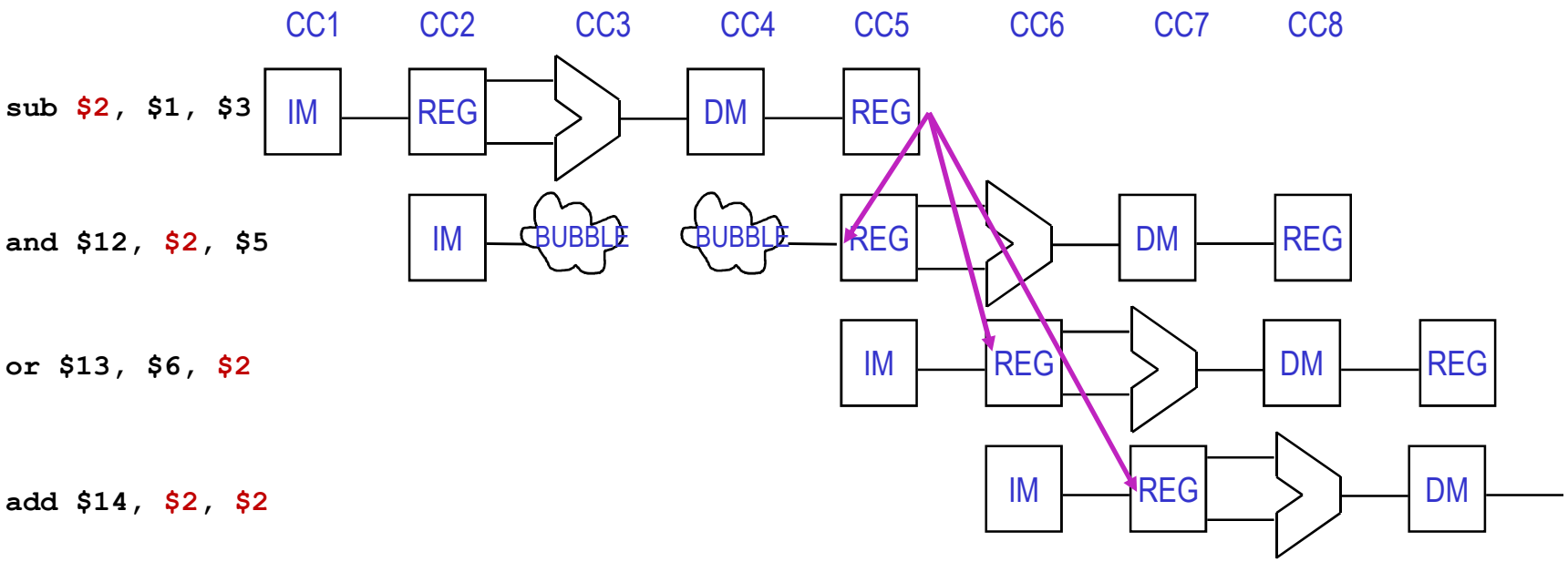
- insert **nop**



insert enough no-ops (or other instructions that don't use register 2) so that data hazard doesn't occur,

Handling Data Hazards in Hardware

- stall the fetch and insert bubbles

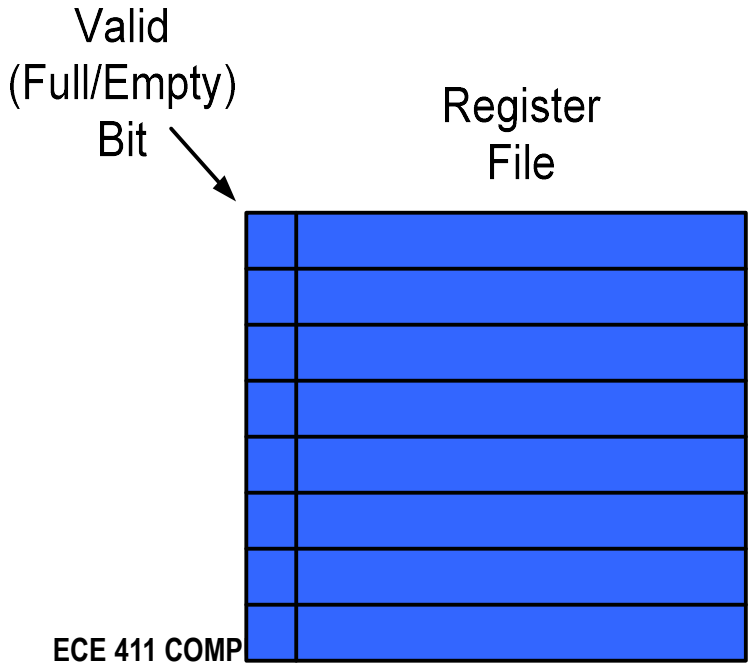


Pipeline Stalls

- to insure proper pipeline execution in light of register dependences, we must:
  - ✓ detect the hazard
  - ✓ stall the pipeline
    - prevent the IF and ID stages from making progress
    - insert “no-ops” into later stages

Register Scoreboard

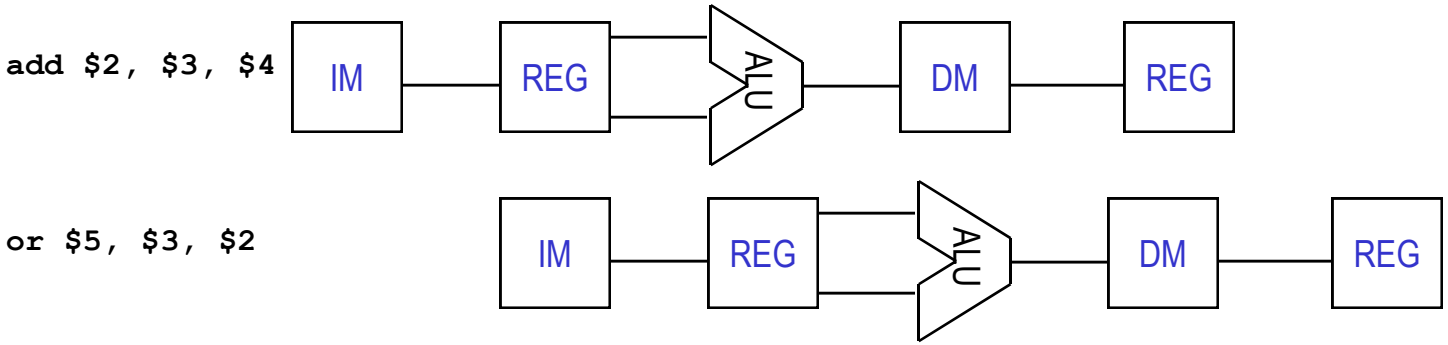
- track operand availability
  - ✓ add valid bit to each register in the reg file
  - ✓ HW clears valid bit when an instr writing the reg issues (leaves decode/reg read stage)
  - ✓ HW sets valid bit when an instr writing the reg completes
  - ✓ instructions are not allowed to issue if any of their source regs are invalid



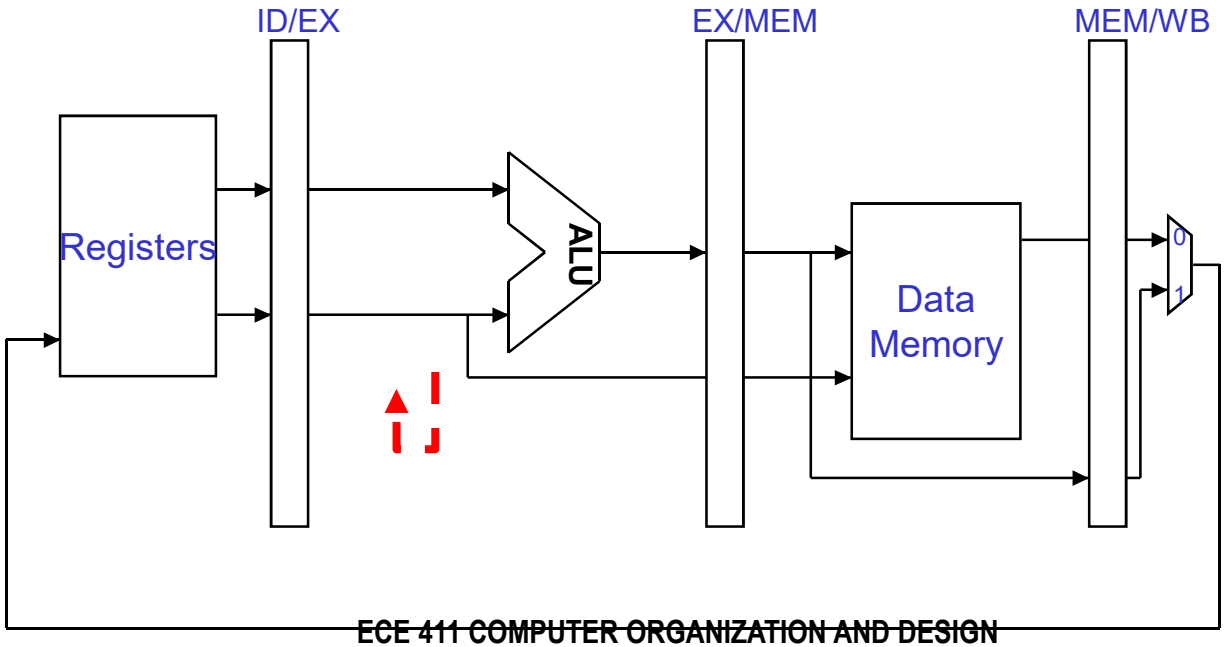
Stalling the Pipeline

- prevent the IF and ID stages from proceeding
  - ✓ don't write the PC (`PCWrite = 0`)
  - ✓ don't rewrite IF/ID register (`IF/IDWrite = 0`)
- insert **`nop`**
  - ✓ Set all control signals propagating to EX/MEM/WB to zero

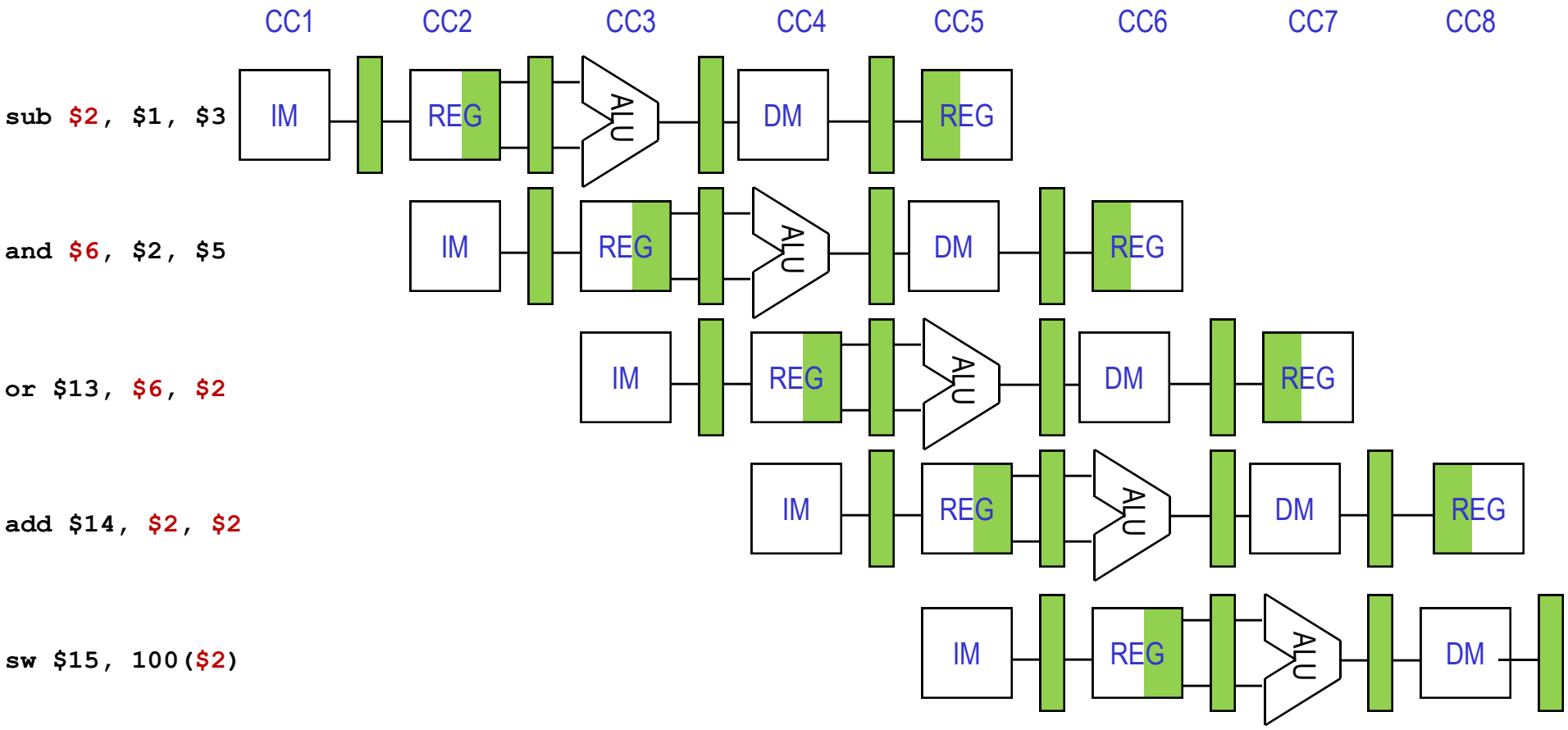
Reducing Data Hazards: Forwarding



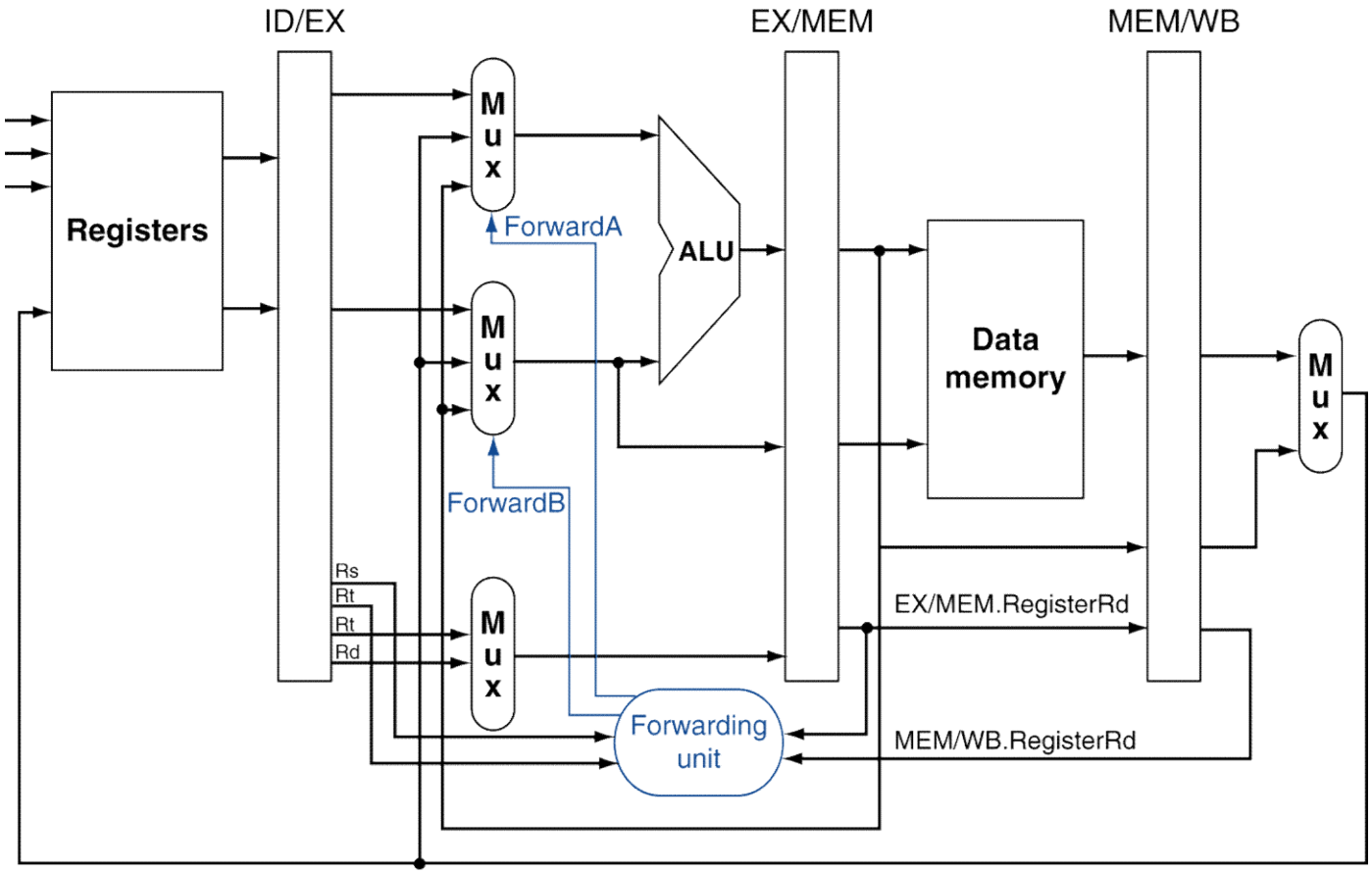
we could avoid stalling if we could get the ALU output from ADD to ALU input for the OR



Reducing Data Hazards: Forwarding



Forwarding Paths



b. With forwarding

Forwarding Conditions

- EX hazard
  - ✓ `if (EX/MEM.RegWrite & (EX/MEM.RegisterRd ≠ 0) & (EX/MEM.RegisterRd = ID/EX.RegisterRs)) ForwardA = 10`
  - ✓ `if (EX/MEM.RegWrite & (EX/MEM.RegisterRd ≠ 0) & (EX/MEM.RegisterRd = ID/EX.RegisterRt)) ForwardB = 10`
- MEM hazard
  - ✓ `if (MEM/WB.RegWrite & (MEM/WB.RegisterRd ≠ 0) & (MEM/WB.RegisterRd = ID/EX.RegisterRs)) ForwardA = 01`
  - ✓ `if (MEM/WB.RegWrite & (MEM/WB.RegisterRd ≠ 0) & (MEM/WB.RegisterRd = ID/EX.RegisterRt)) ForwardB = 01`

Double Data Hazard

- consider the sequence:
  - ✓ `add $1,$1,$2`
  - ✓ `add $1,$1,$3`
  - ✓ `add $1,$1,$4`
- both hazards occur
  - ✓ want to use the most recent
- revise MEM hazard condition
  - ✓ only fwd if ex hazard condition isn't true

Revised Forwarding Condition

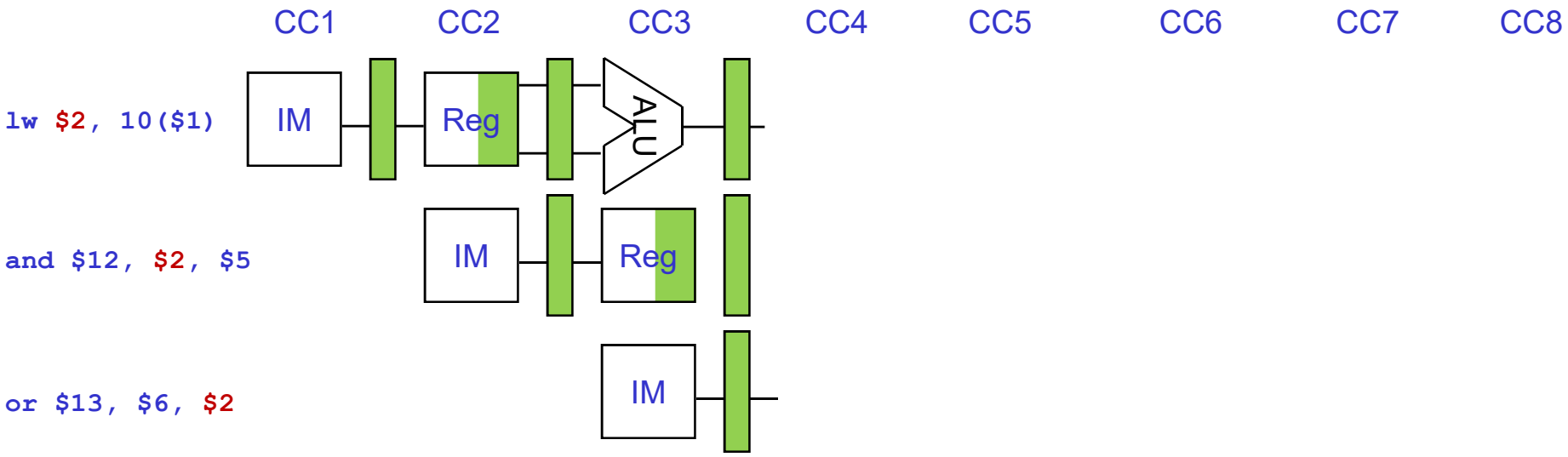
- MEM hazard
  - ✓ `if (MEM/WB.RegWrite & (MEM/WB.RegisterRd ≠ 0)`
    `& !(EX/MEM.RegWrite & (EX/MEM.RegisterRd ≠ 0)`
    `& (EX/MEM.RegisterRd != ID/EX.RegisterRs))`
    `& (MEM/WB.RegisterRd = ID/EX.RegisterRs))`
    `ForwardA = 01`
  - ✓ `if (MEM/WB.RegWrite & (MEM/WB.RegisterRd ≠ 0)`
    `& !(EX/MEM.RegWrite & (EX/MEM.RegisterRd ≠ 0)`
    `& (EX/MEM.RegisterRd != ID/EX.RegisterRt))`
    `& (MEM/WB.RegisterRd = ID/EX.RegisterRt))`
    `ForwardB = 01`

Forwarding Eliminate All Data Hazards?

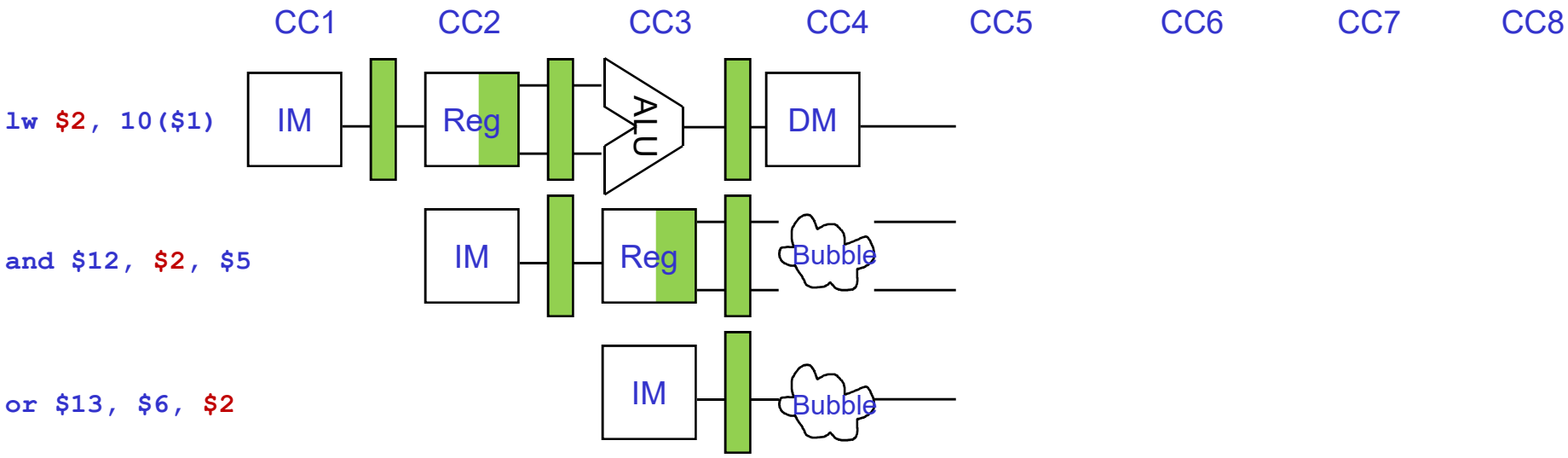
CC1 CC2 CC3 CC4 CC5 CC6 CC7 CC8

```
lw $2, 10($1)

and $12, $2, $5

or $13, $6, $2

add $14, $2, $2

sw $15, 100($2)
```

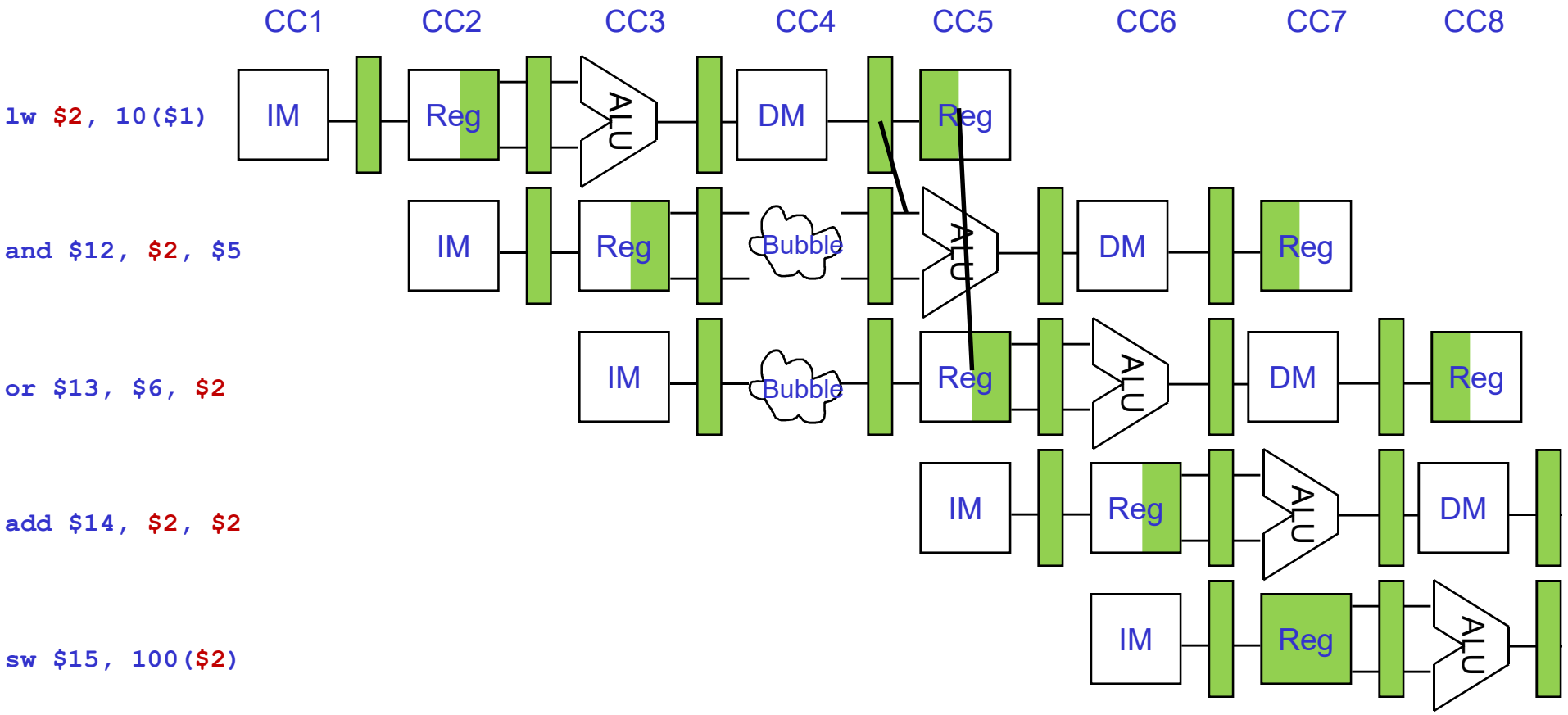
Forwarding Eliminate All Data Hazards?



Forwarding Eliminate All Data Hazards?



Forwarding Eliminate All Data Hazards?

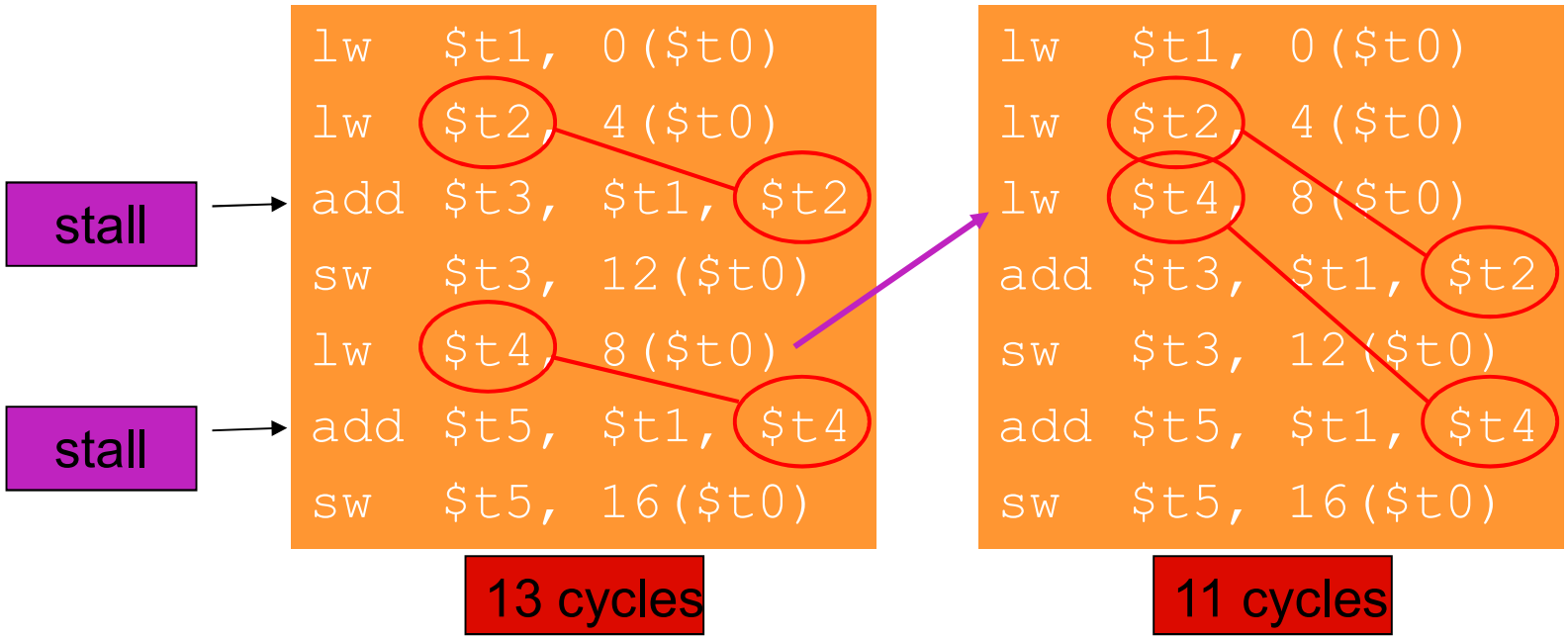


Try this one...

- show stalls and forwarding for this code
  - ✓ `add $3, $2, $1`
  - ✓ `lw $4, 100($3)`
  - ✓ `and $6, $4, $3`
  - ✓ `sub $7, $6, $2`

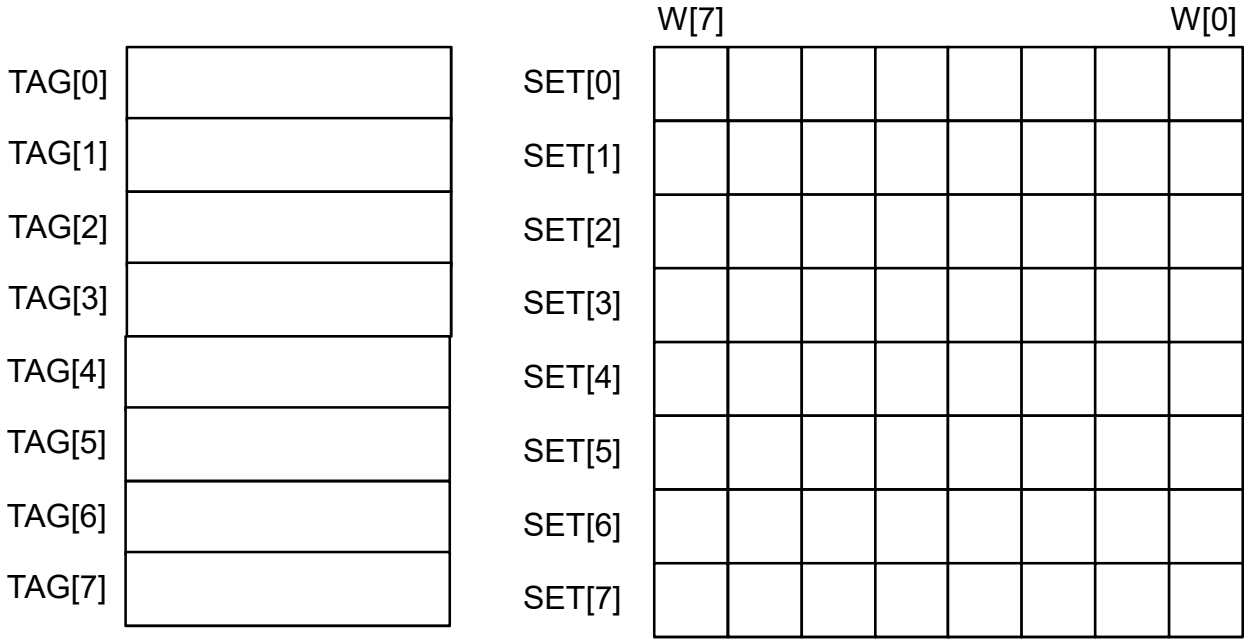
Code Scheduling to Avoid Stalls

- reorder code to avoid use of load result in the next instruction
- C code for `A = B + E; C = B + F;`



Announcement

- today's lecture: pipeline
  - ✓ Ch. 4.5 – 4.10 (HP1)
- next lecture: pipeline
  - ✓ Ch. 4.5 – 4.10 (HP1)
- MP assignment
  - ✓ HW1 due on 2/13 5pm
  - ✓ MP2 check-point due on 2/18 5pm


W[7]
W[0]
TAG[0]
TAG[1]
TAG[2]
TAG[3]
TAG[4]
TAG[5]
TAG[6]
TAG[7]
SET[0]
SET[1]
SET[2]
SET[3]
SET[4]
SET[5]
SET[6]
SET[7]