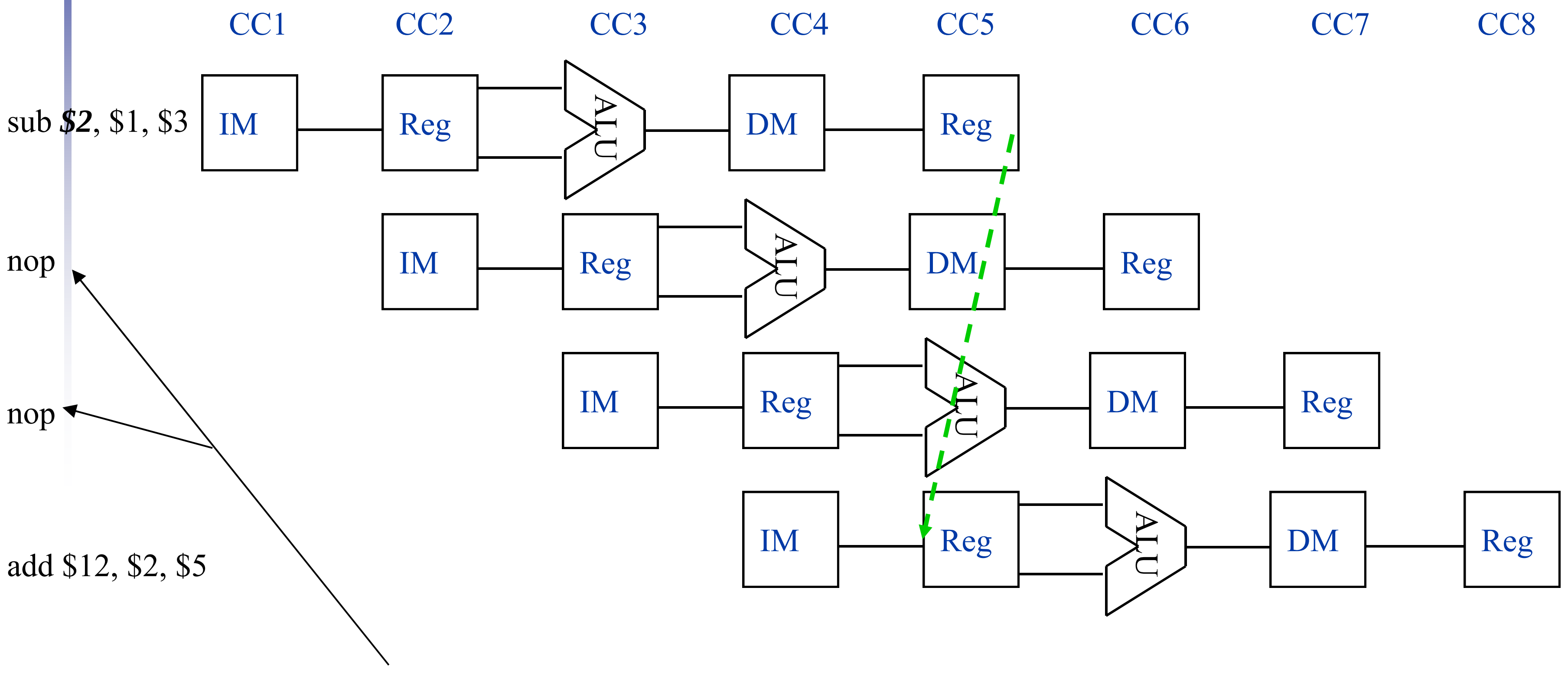
Chapter 4

The Processor

Dealing with Data Hazards

- In Software
  - insert independent instructions (or no-ops)
- In Hardware
  - insert bubbles (i.e. stall the pipeline)
  - data forwarding

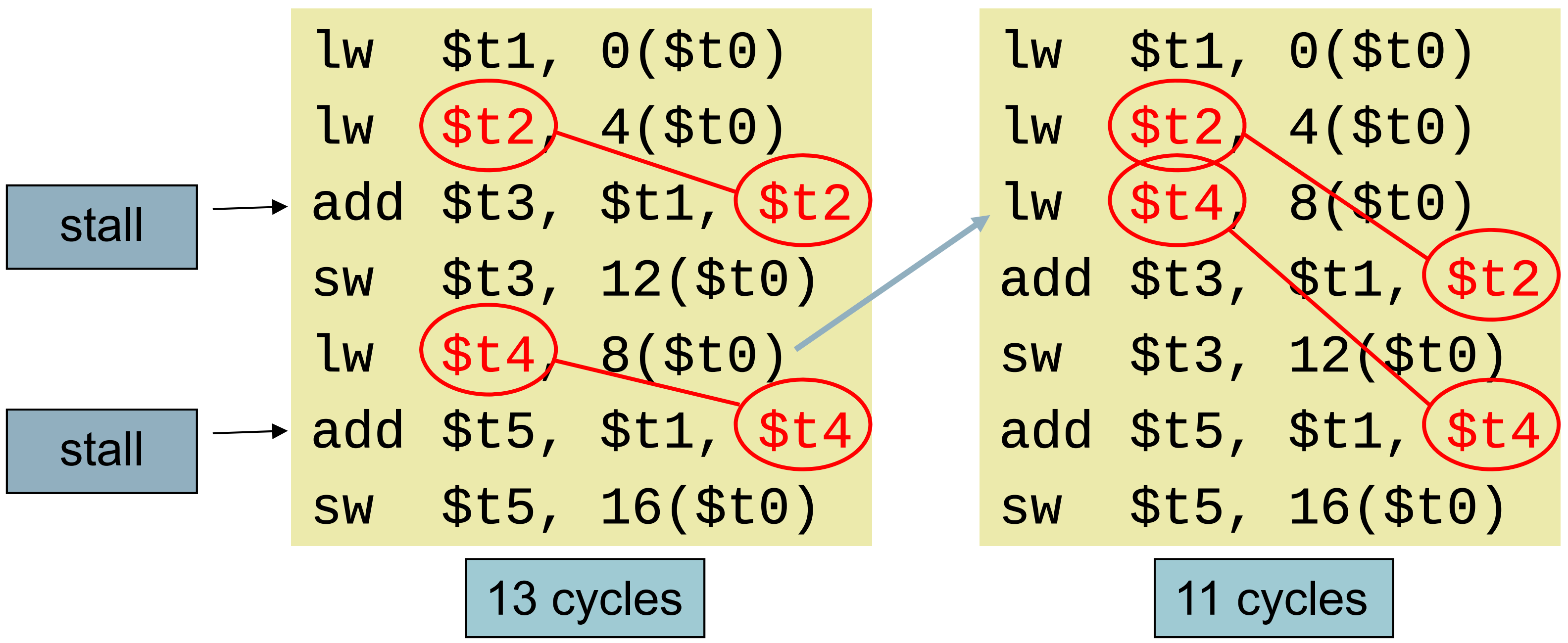
Dealing with Data Hazards in Software



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

Code Scheduling to Avoid Stalls

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



```
lw   $t1, 0($t0)
lw   $t2, 4($t0)
add  $t3, $t1, $t2
sw   $t3, 12($t0)
lw   $t4, 8($t0)
add  $t5, $t1, $t4
sw   $t5, 16($t0)
```

```
lw   $t1, 0($t0)
lw   $t2, 4($t0)
lw   $t4, 8($t0)
add  $t3, $t1, $t2
sw   $t3, 12($t0)
add  $t5, $t1, $t4
sw   $t5, 16($t0)
```

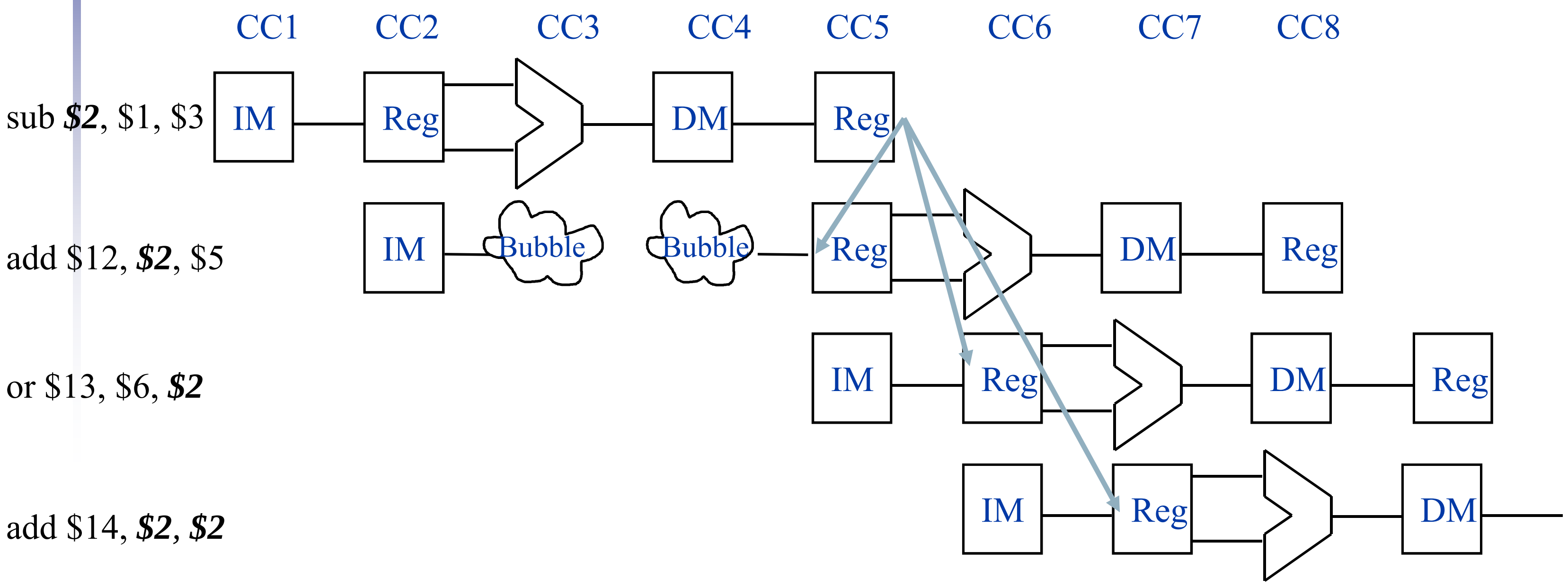
Where are No-ops needed?

- sub $2, $1,$3
- and $4, $2,$5
- or $8, $2,$6
- add $9, $4,$2
- slt $1, $6,$7

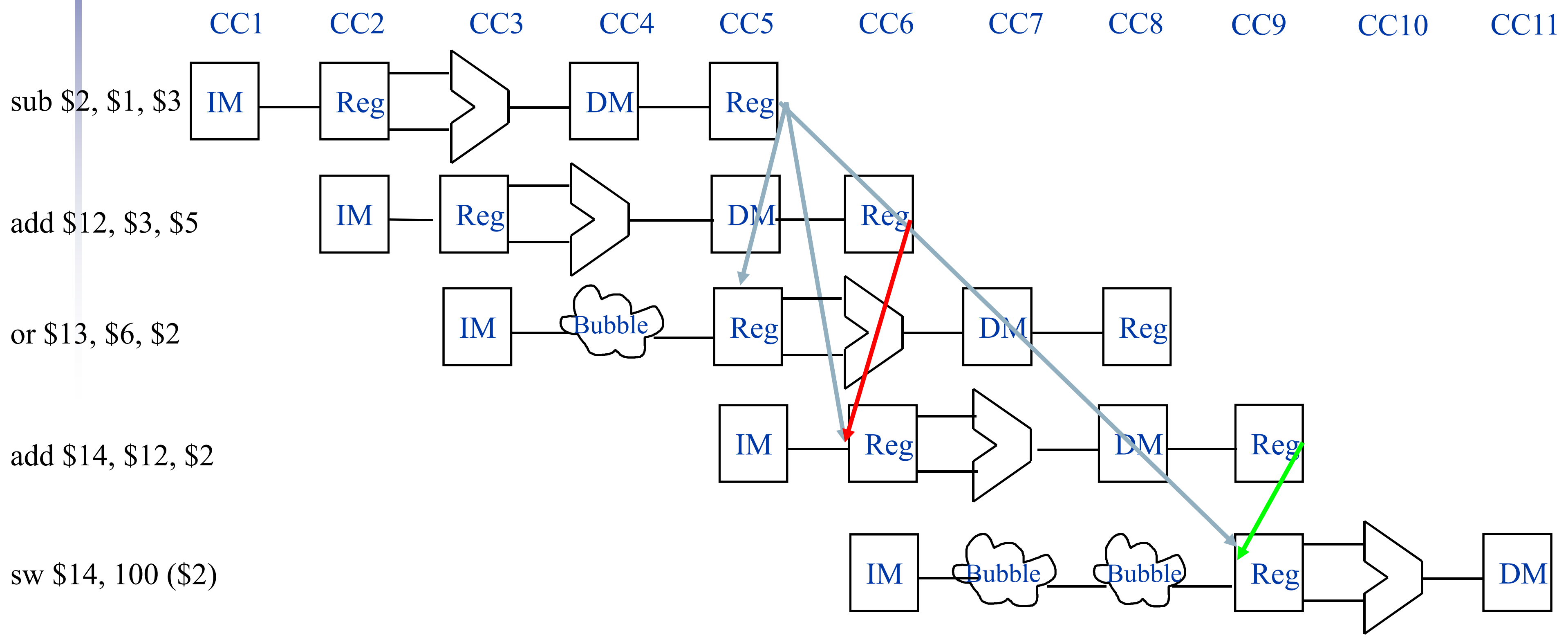
- Are no-ops really necessary?

Handling Data Hazards in Hardware

- Stall the pipeline



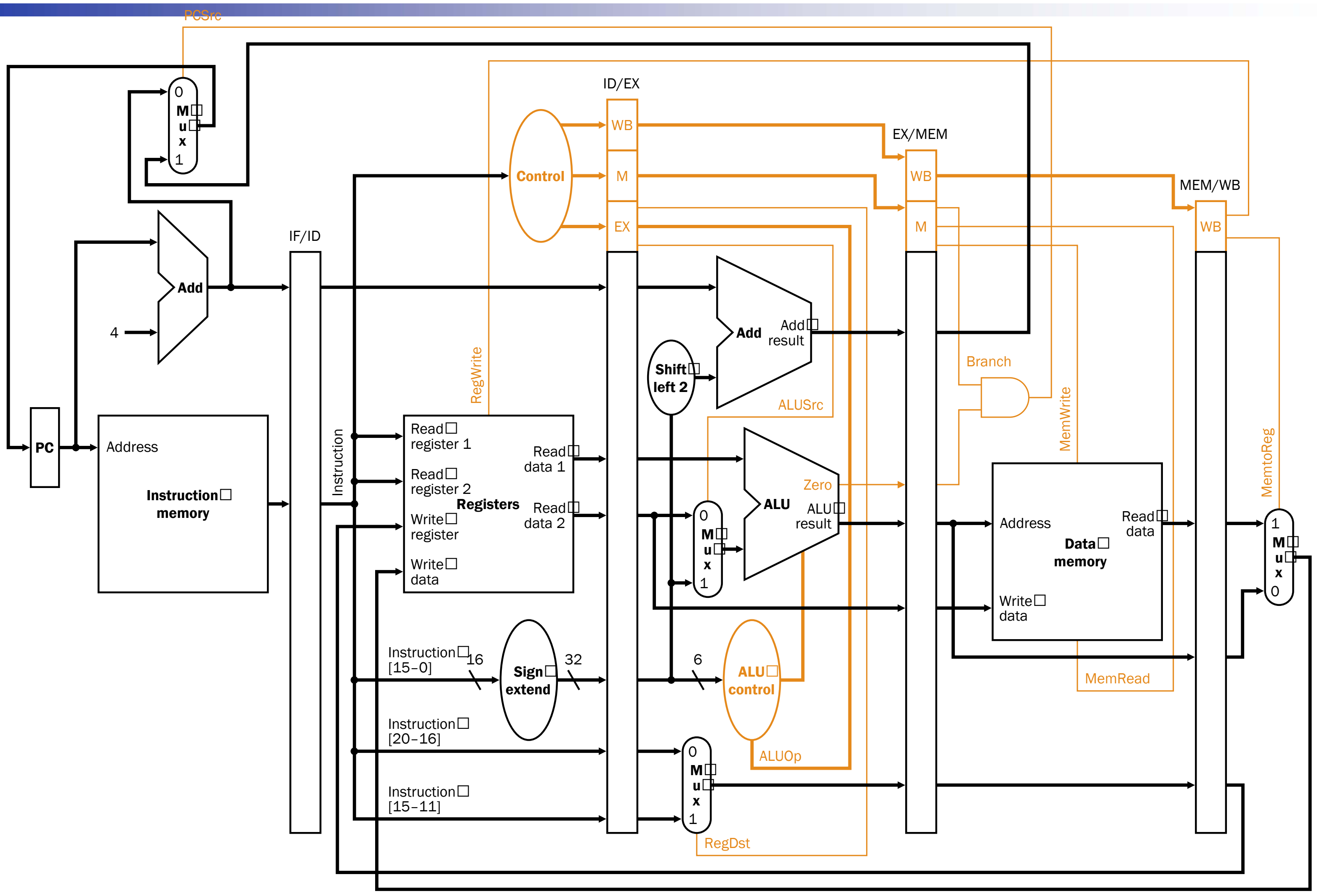
Handling Data Hazards in Hardware



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
      - the ID stage because we can’t go on until the dependent instruction completes correctly
      - the IF stage because we do not want to lose any instructions.

The Pipeline



What comparisons tell us when to stall?

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 “nops”
  - set all control signals propagating to EX/MEM/WB to zero