EE116C/CS151B
Fall 2017

Instructor: Professor Lei He
TA: Yuan Liang

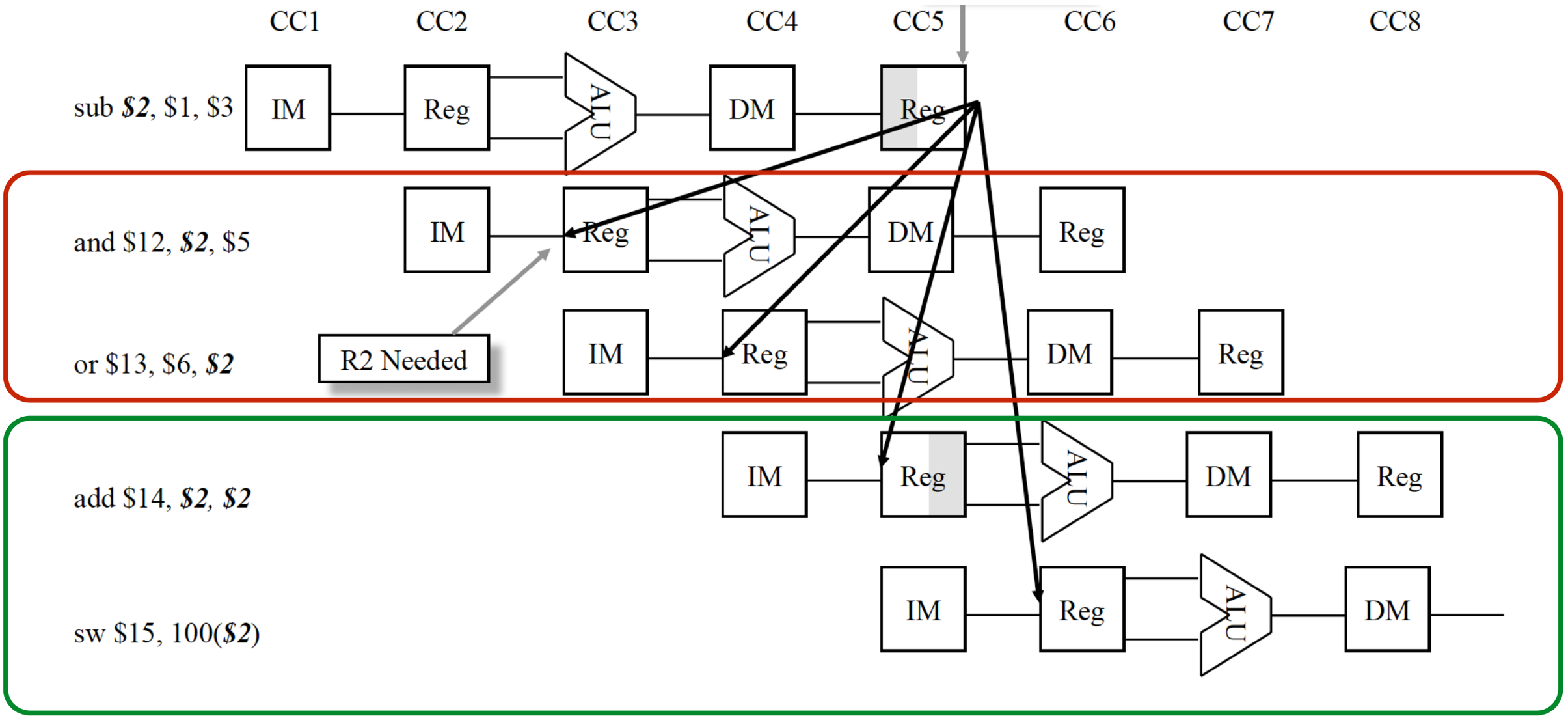
Review

- **Structural hazard (we have mentioned)**
- **Data hazards**
  - what is data hazard
  - how to solve it
- **Control hazards**
  - what is control hazard
  - how to solve it
- **Instruction Level Parallelism (ILP)**
  - Super-pipelining
    - static scheduling (at compiler)
    - dynamic scheduling (at hardware)

Review

- **Data hazards**
  - When a result is needed in the pipeline before it is available, a “data hazard” occurs.

Review

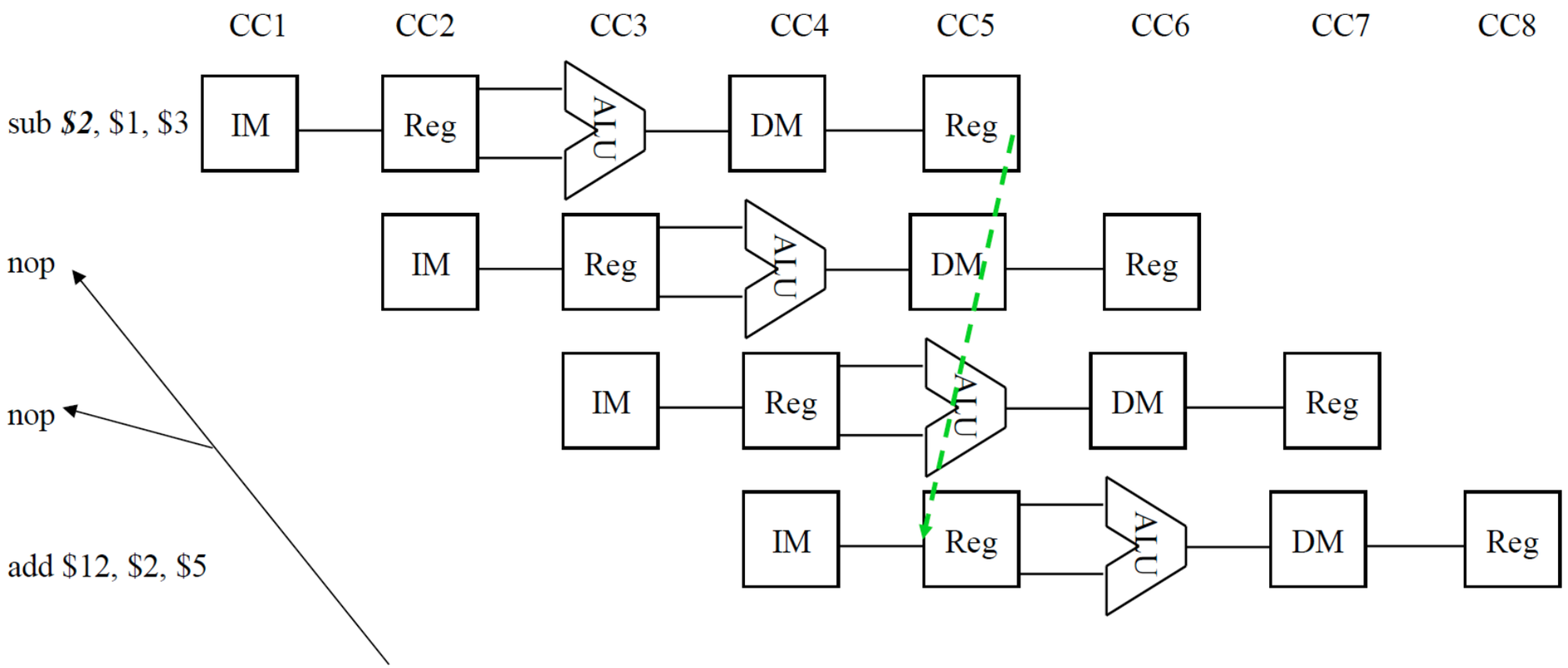

CC1
CC2
CC3
CC4
CC5
CC6
CC7
CC8
sub $2, $1, $3
IM
Reg
ALU
DM
Reg
and $12, $2, $5
IM
Reg
ALU
DM
Reg
or $13, $6, $2
R2 Needed
IM
Reg
ALU
DM
Reg
add $14, $2, $2
IM
Reg
ALU
DM
Reg
sw $15, 100($2)
IM
Reg
ALU
DM

Review

- **Solutions**
  - In Software
    insert independent instructions (or no-ops)
    re-ordering (not covered here)
  - In Hardware
    insert bubbles (i.e. stall the pipeline)
    data forwarding
    register transparent

Review

- **Insert no-ops or other instructions**



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

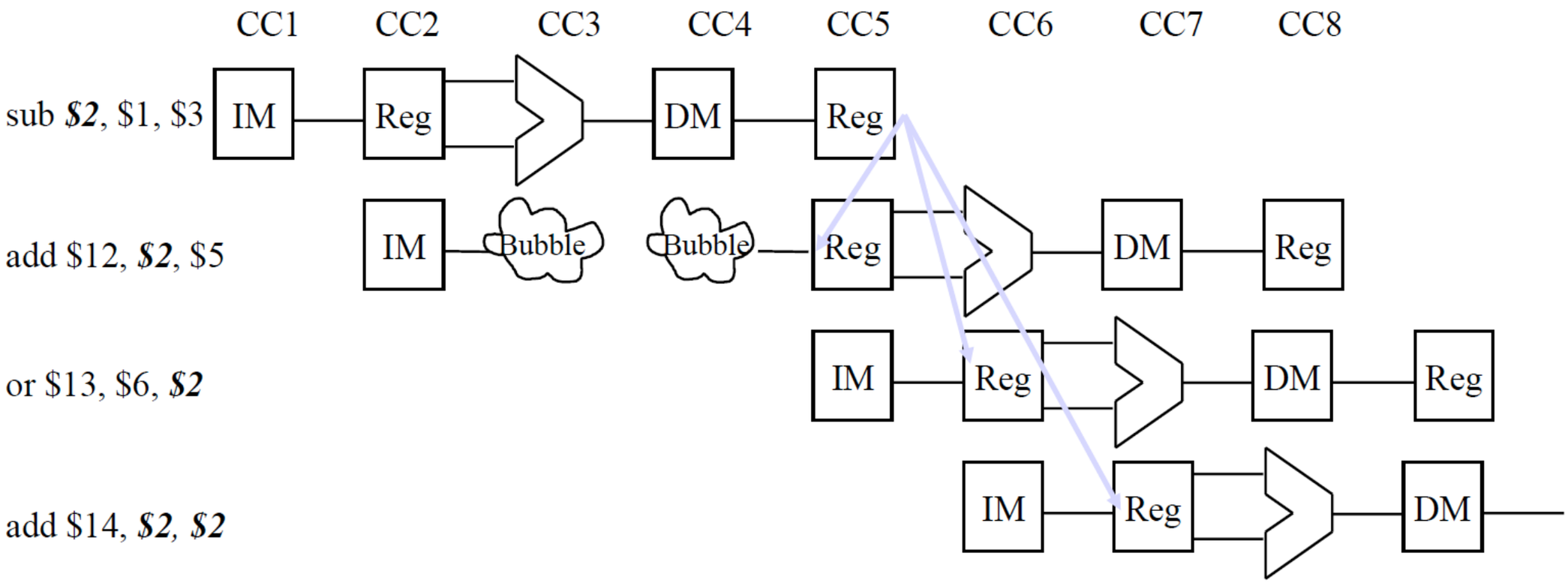
Review

- **Insert no-ops or other instructions**

- When you program. Burden when coding.

Review

- **Insert stalls to pipeline (stop IF from fetching new instructions)**



Review

- **Insert stalls to pipeline**

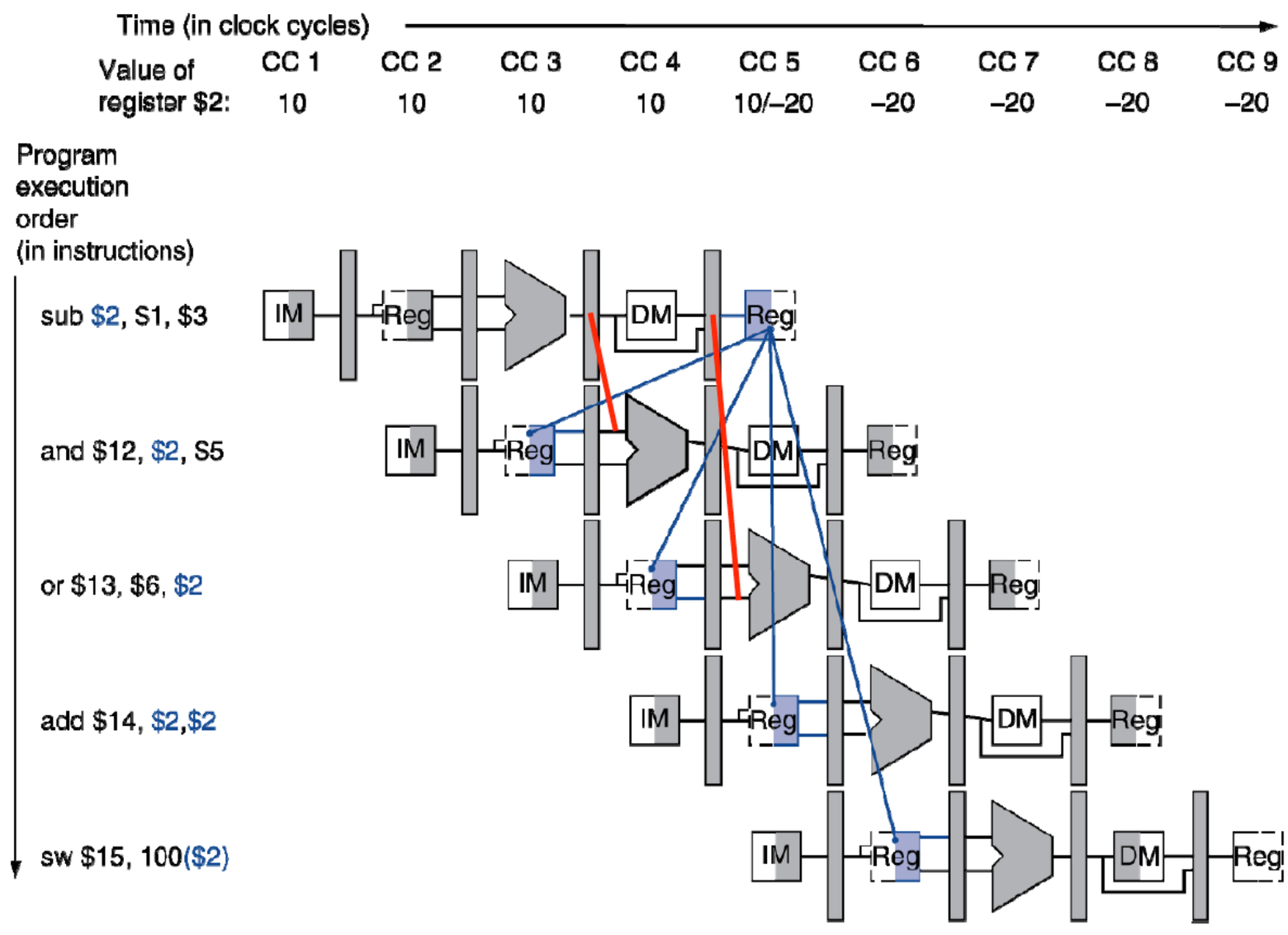
  - Prevent the IF and ID stages from proceeding
    – don’t write the PC (PCWrite = 0)
    – don’t rewrite IF/ID register (IF/IDWrite = 0)
  - Insert “nops”
    – set all control signals propagating to EX/MEM/WB to zero

Review

- **Data forwarding**

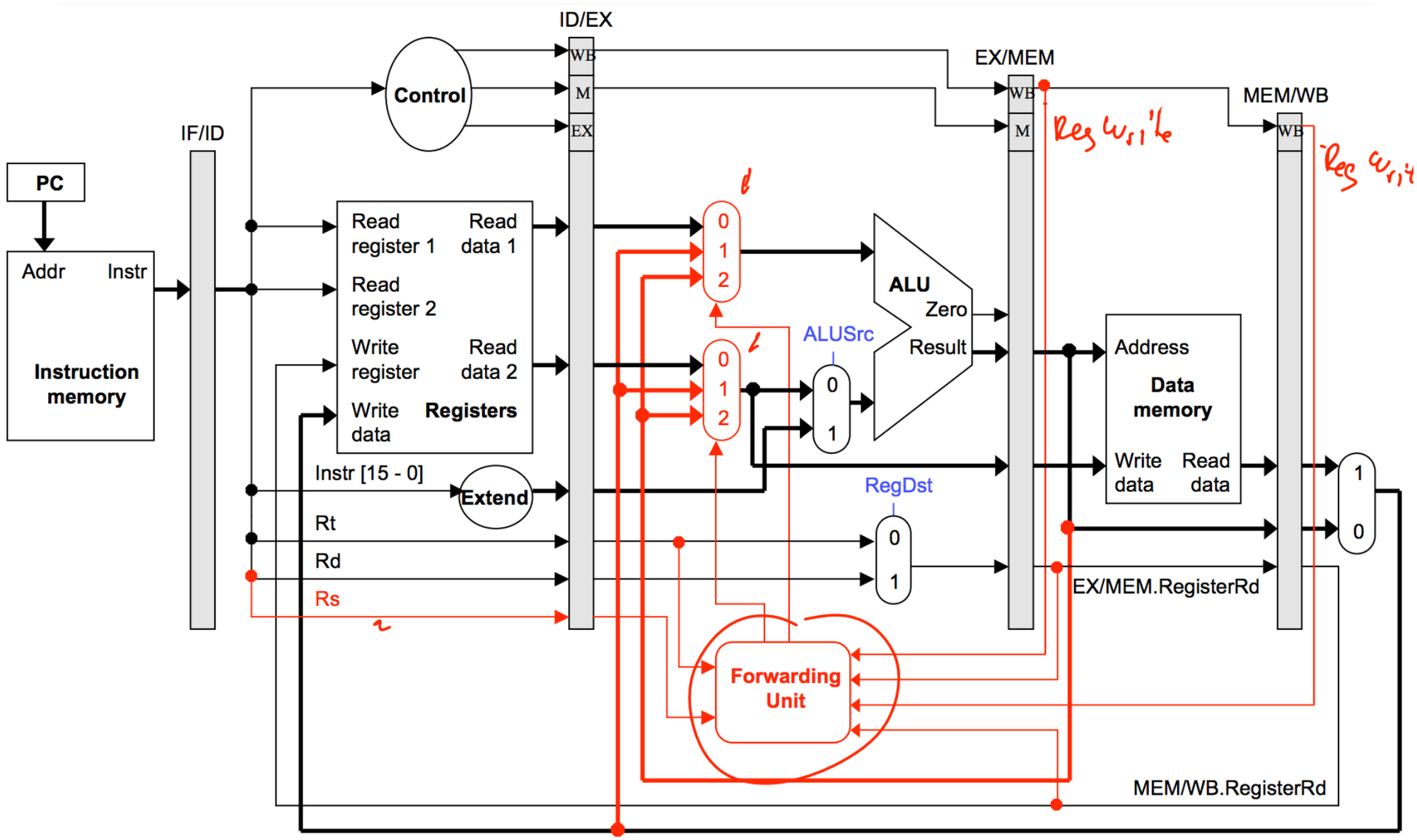
Forward from EX/MEM or MEM/WB

As alu input
As mem input



Review

- **Data forwarding**



Review

- **Data forwarding**

EX/MEM.RegisterRd = ID/EX.RegisterRs
EX/MEM.RegisterRd = ID/EX.RegisterRt
MEM/WB.RegisterRd = ID/EX.RegisterRs
MEM/WB.RegisterRd = ID/EX.RegisterRt

EX/MEM.RegWrite, MEM/WB.RegWrite

EX/MEM.RegisterRd ≠ 0,
MEM/WB.RegisterRd ≠ 0

Review

- **Data forwarding**

EX hazard

- if (EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0)
  and (EX/MEM.RegisterRd = ID/EX.RegisterRs))
  ForwardA = 10
- if (EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0)
  and (EX/MEM.RegisterRd = ID/EX.RegisterRt))
  ForwardB = 10

MEM hazard

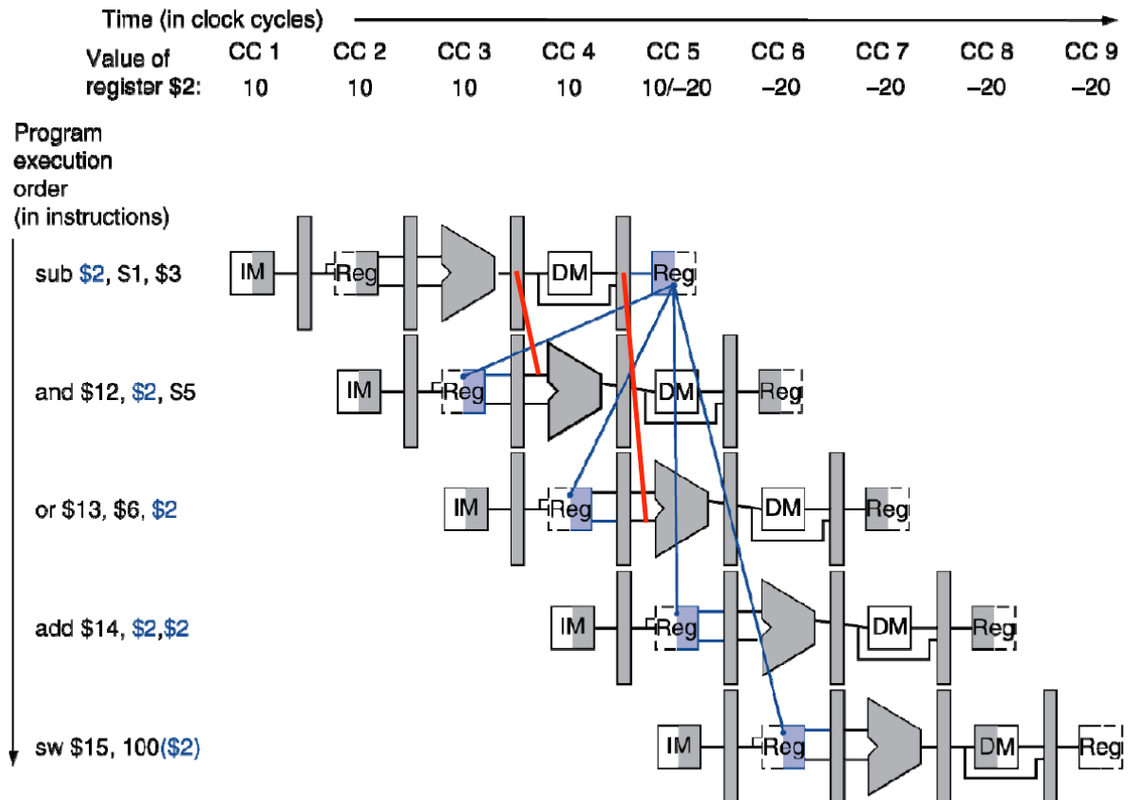
- if (MEM/WB.RegWrite and (MEM/WB.RegisterRd ≠ 0)
  and (MEM/WB.RegisterRd = ID/EX.RegisterRs))
  ForwardA = 01
- if (MEM/WB.RegWrite and (MEM/WB.RegisterRd ≠ 0)
  and (MEM/WB.RegisterRd = ID/EX.RegisterRt))
  ForwardB = 01

Review

- **Register transparent**

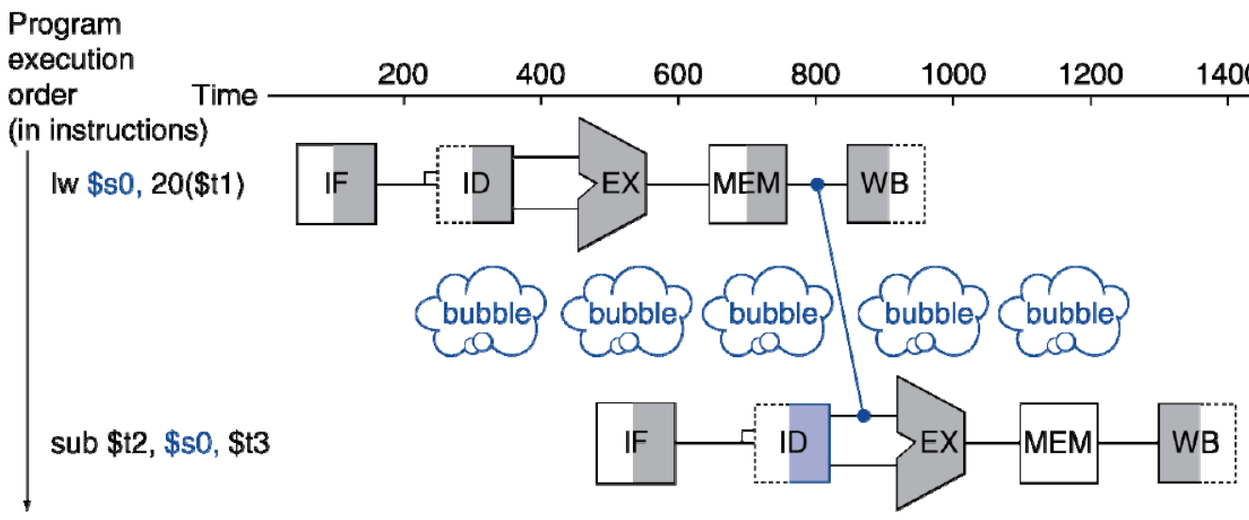
Register written can be used in the current cycle.
(they are the same thing / fast read/write)

But memory read cannot.



Review

- **Data forwarding**
  - Cannot solve with all the hazards.



- **Real solution**
  - Combination of no-ops, stalls, and forwarding, register transparent.

Review

- **Control hazards**
  - one instructions determines whether another gets executed or not.

```
beq $6, $7, somewhere
add $2, $1, $3
somewhere: add $1, $2, $3
```

pollute.

- **Ground truth: the branches are decided in the MEM stage**

Review

- **Solutions**
  - In Software

    Insert independent instructions (or no-ops)
  - In Hardware

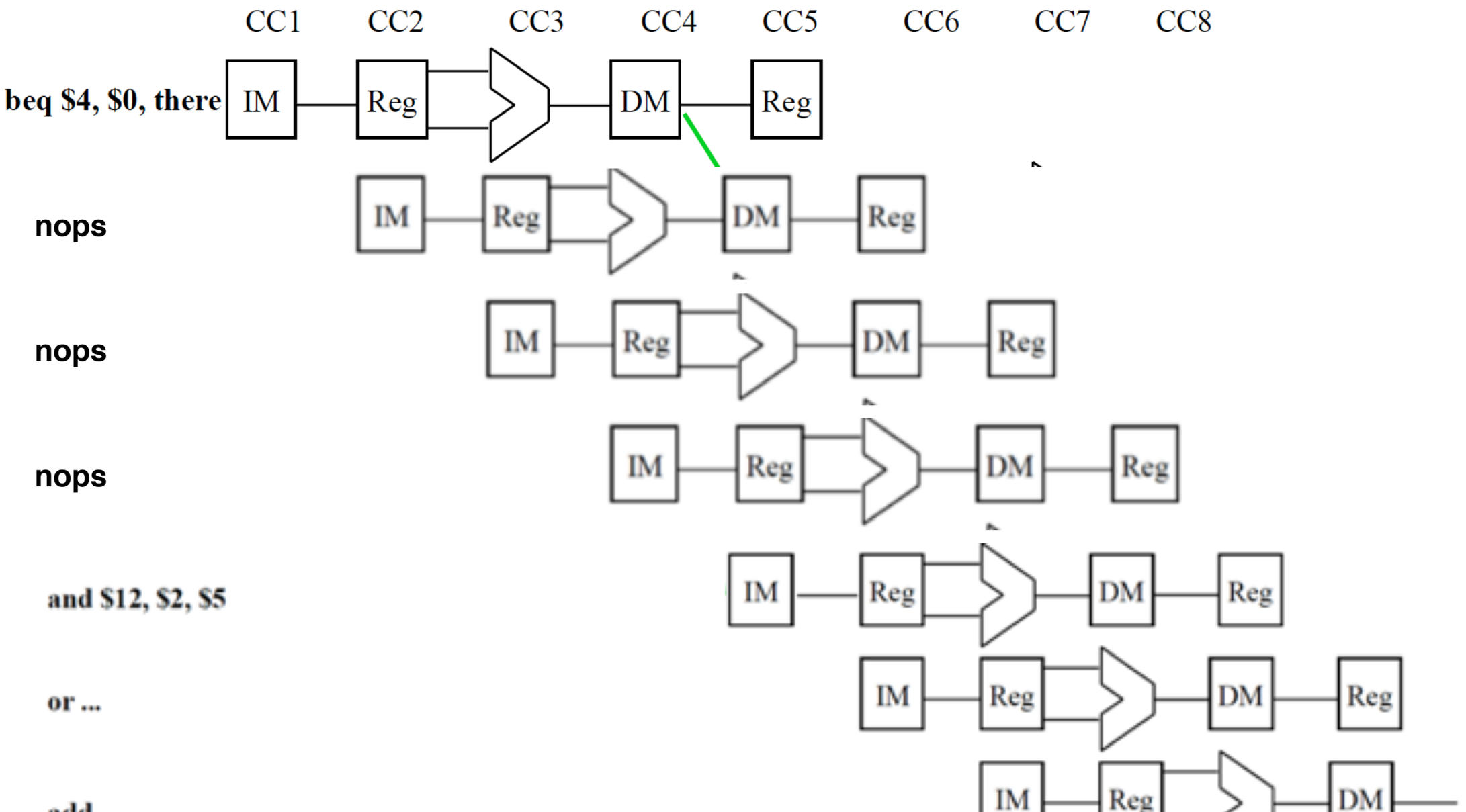
    Insert instructions (i.e. stall the pipeline, or you can find something to put there)

    Early detection

    Guess which direction, start executing chosen path (but be prepared to undo any mistakes!)

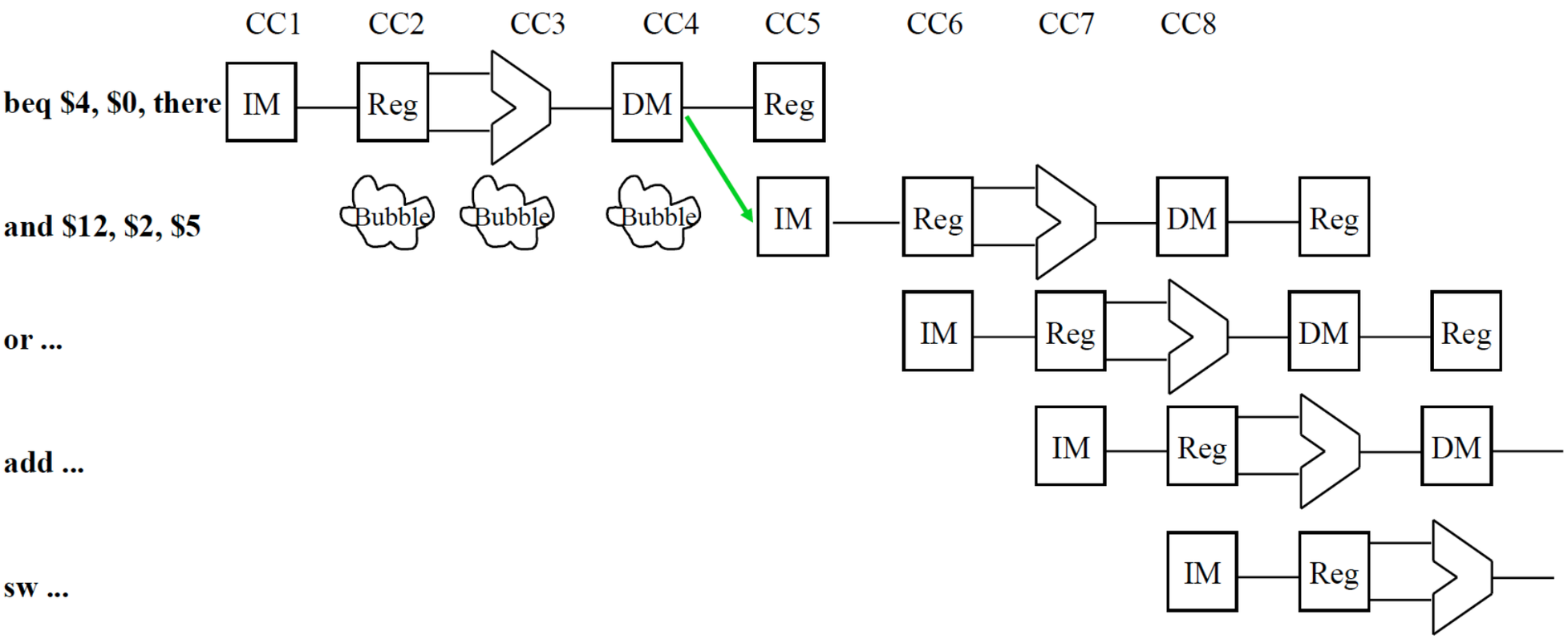
Review

- **Solutions: insert nops**



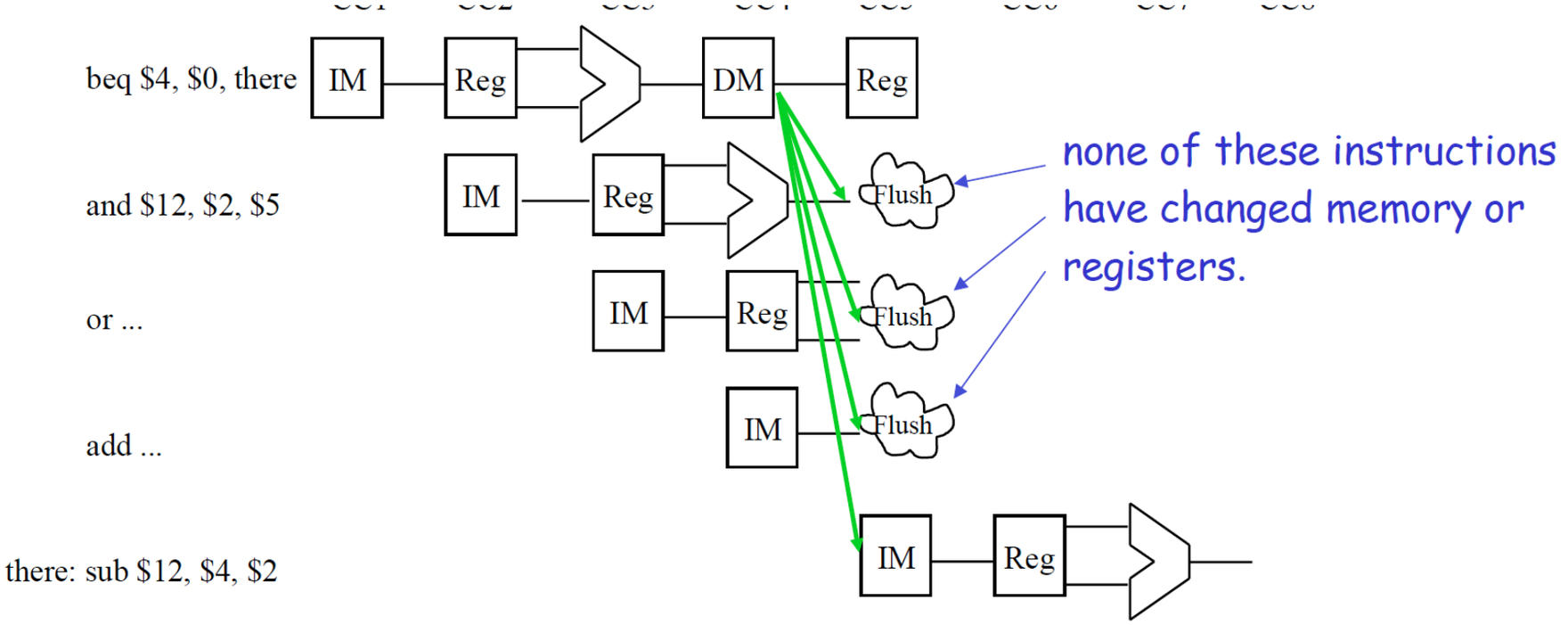
Review

- **Solutions: insert stalls**



Review

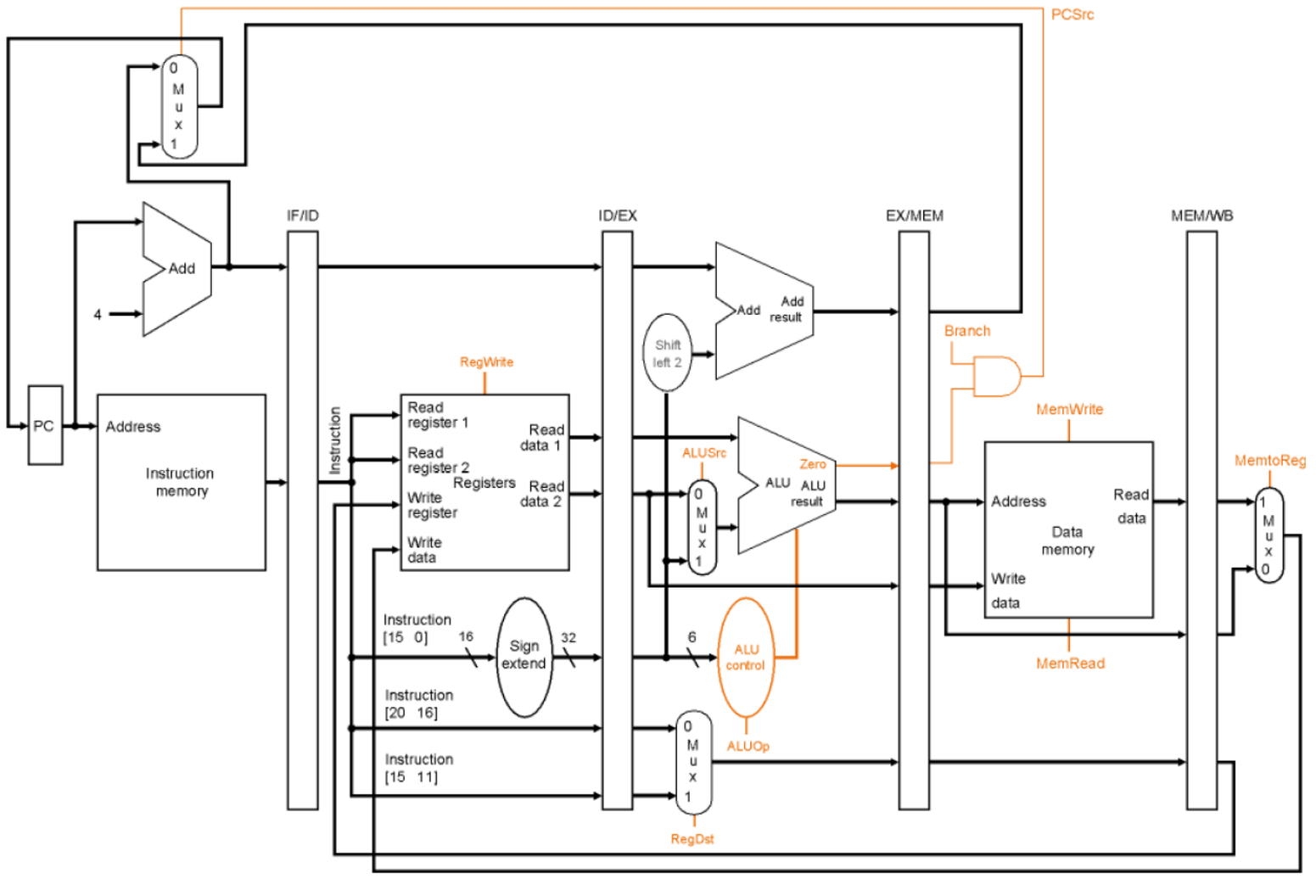
- **Solutions: guess**



Review

- **Solutions: Early detection**

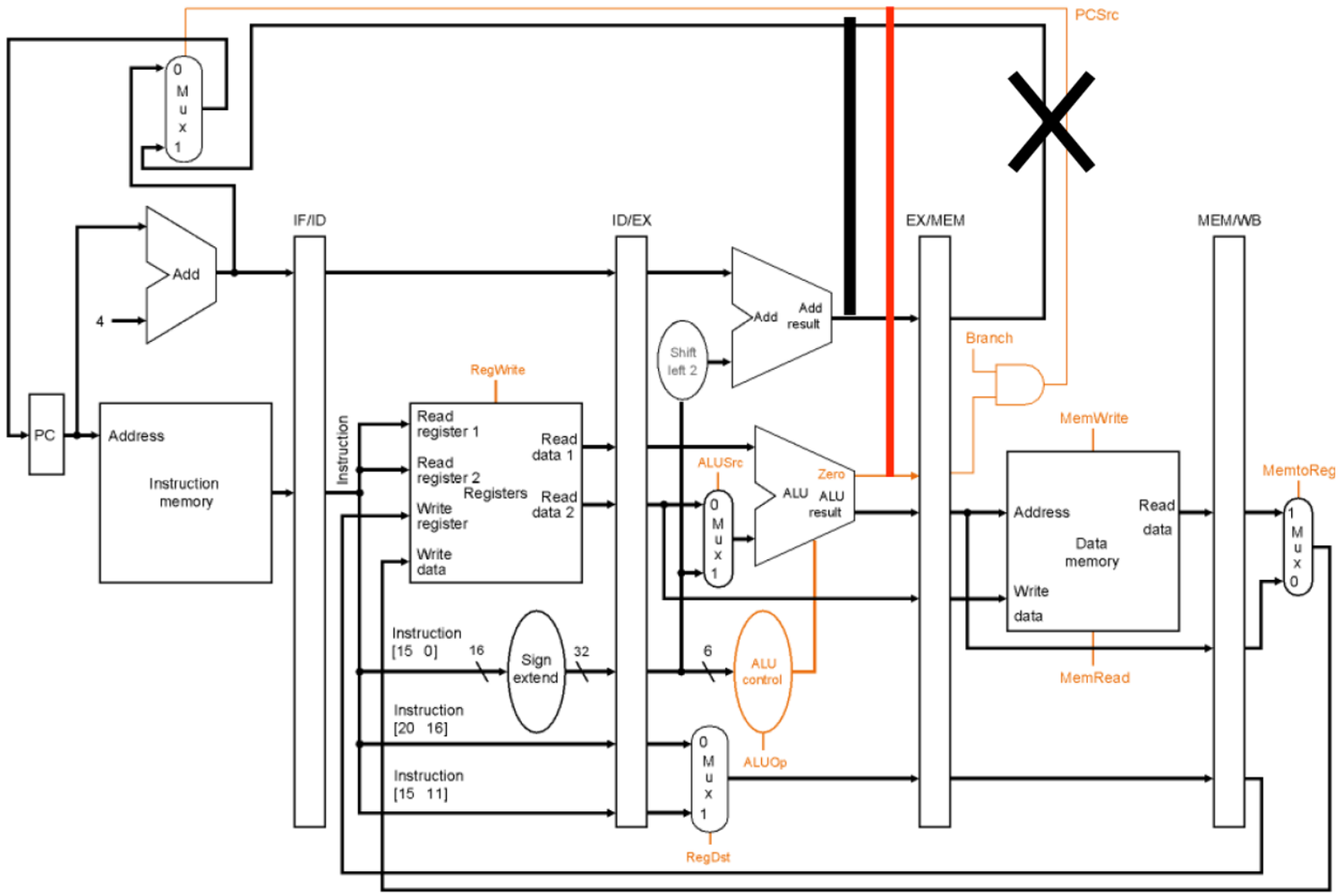
Early detection: 3 cycles delay
IF ID EX MEM WB
IF ID EX MEM WB



Review

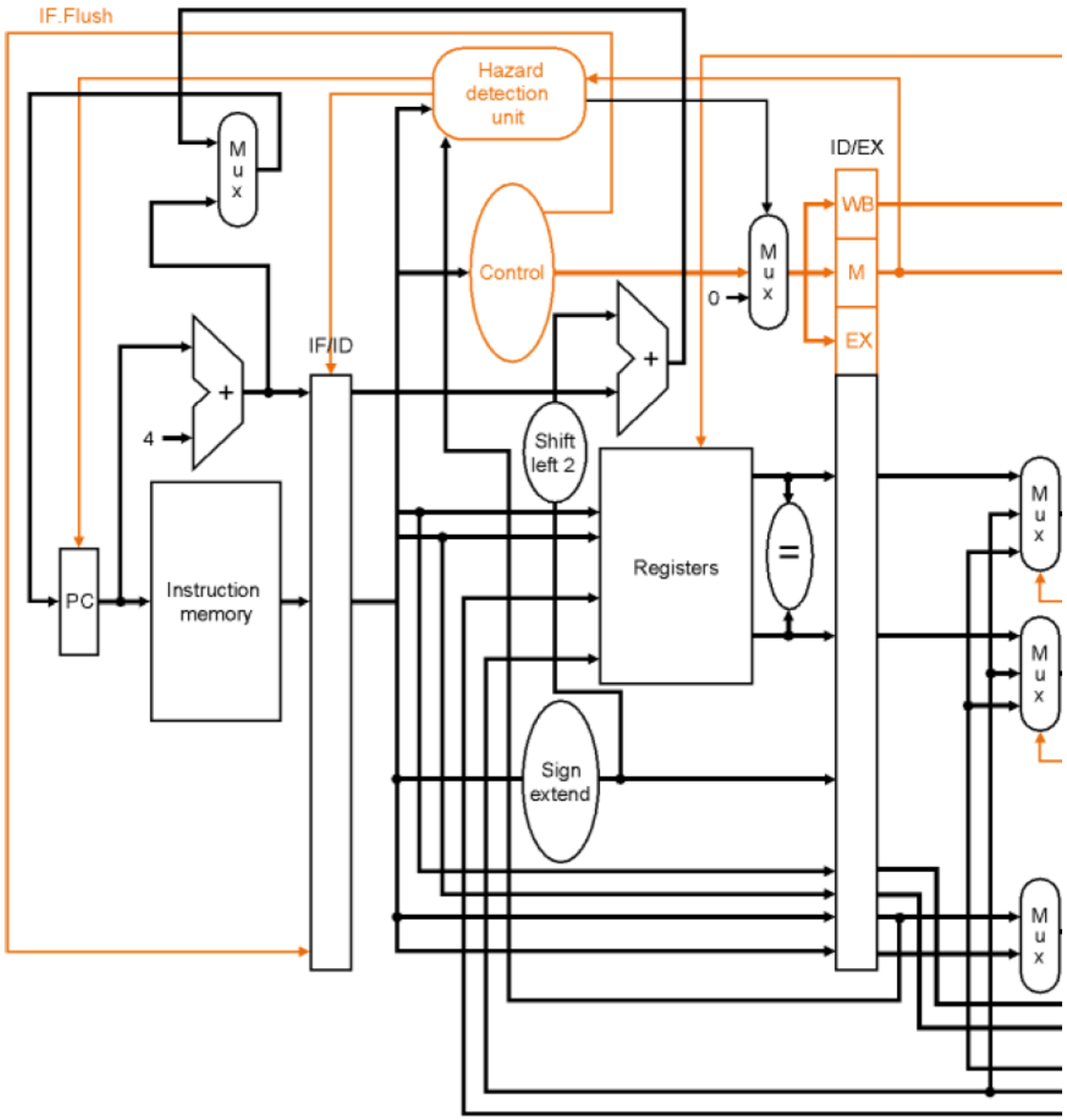
- **Solutions**

Early detection: 2 cycles delay
IF ID EX MEM WB
IF ID EX MEM WB



Review

- **Solutions**

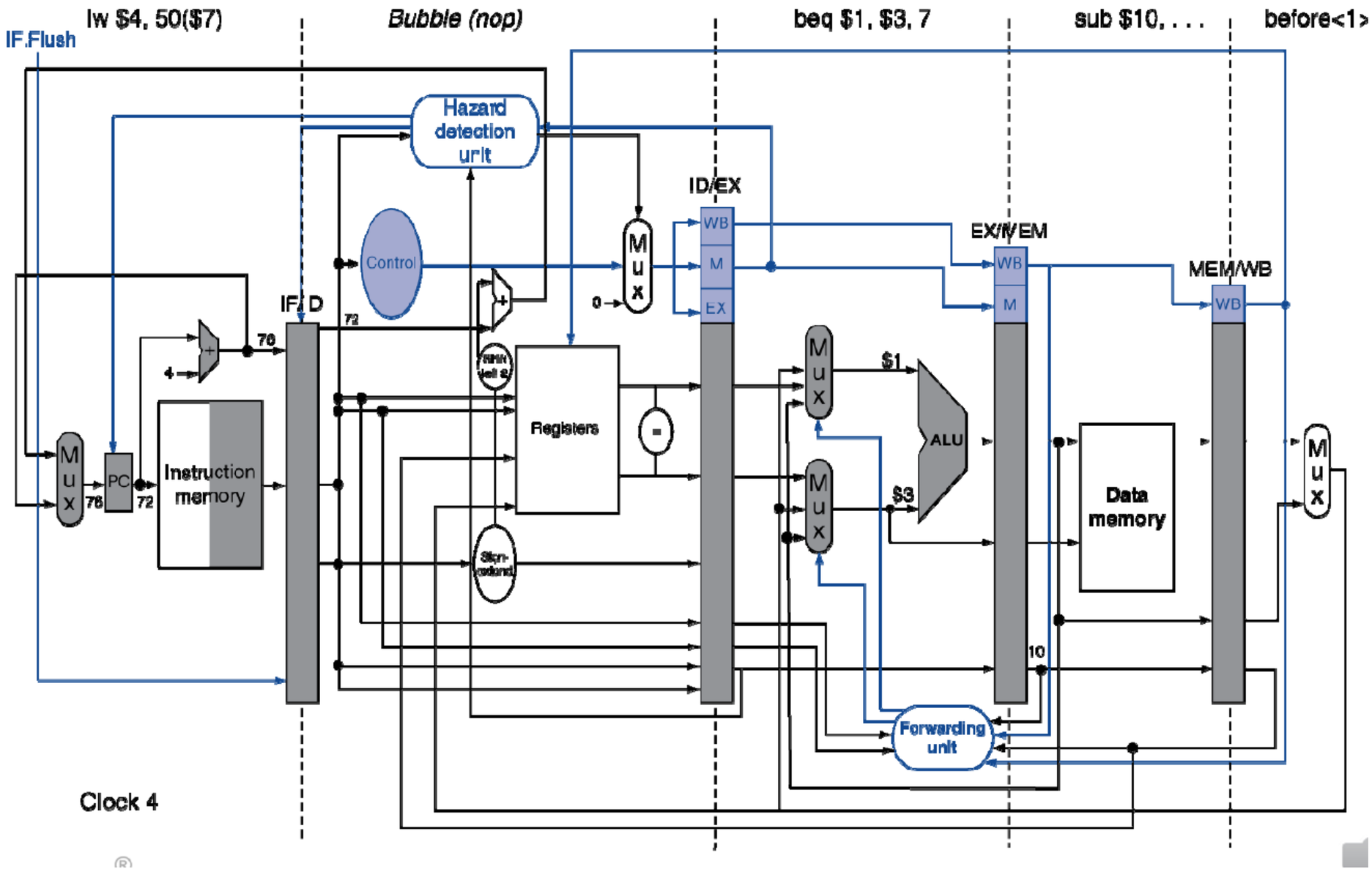


Early detection: 1 cycles delay
IF ID EX MEM WB
IF ID EX MEM WB

Review

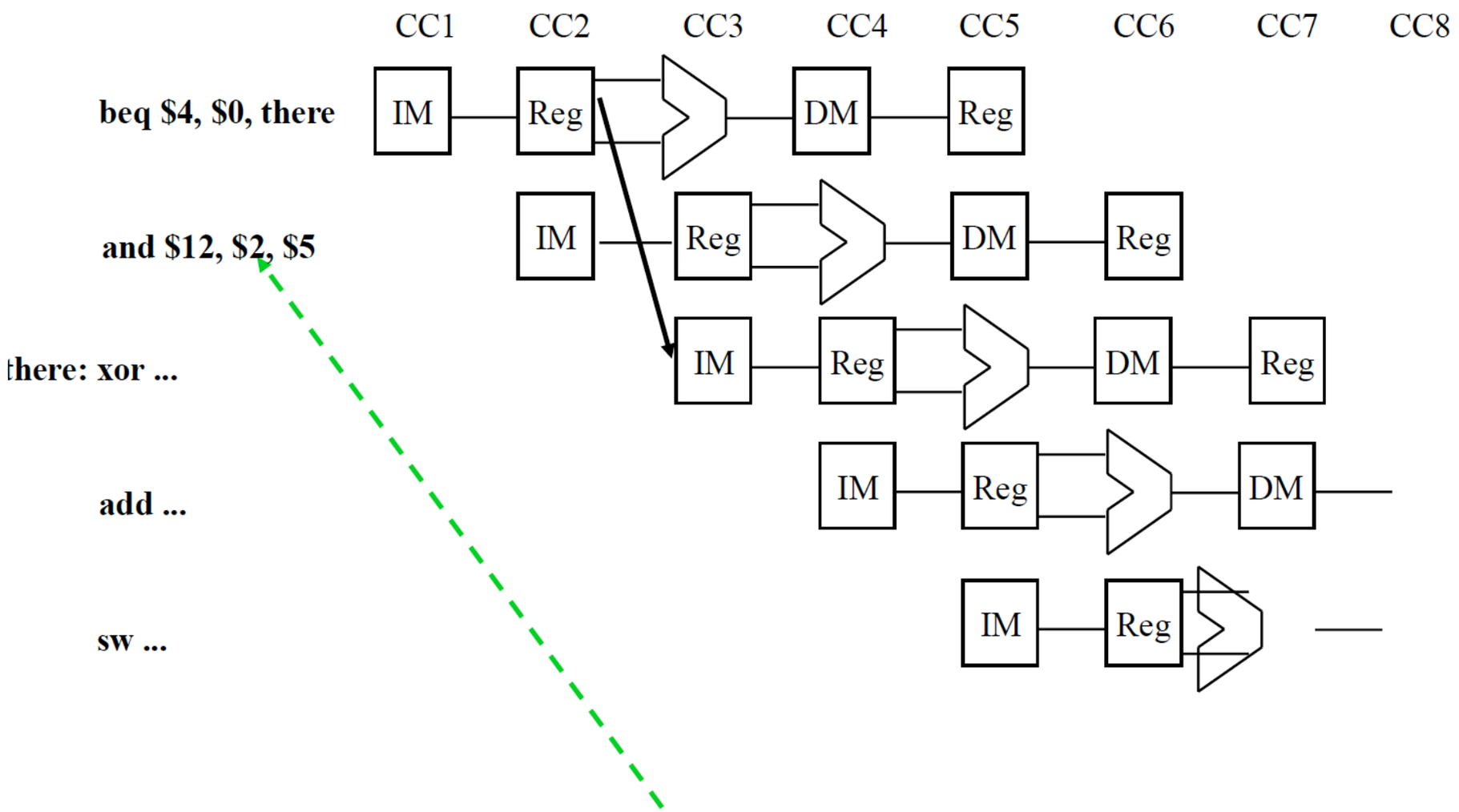
- **Real Solution**
  - We have longer pipeline in real processors. -> stall penalty becomes unacceptable.
  - 1. Predict (static/dynamic prediction) outcome of branch, and only stall (flush) if prediction is wrong.
  - 2. Fetch instructions after brach with no delay.
  - 3. Hardware for early detection.

Review



- **Predict + early detection + fill (1 delay) with instructions after brach**

Review



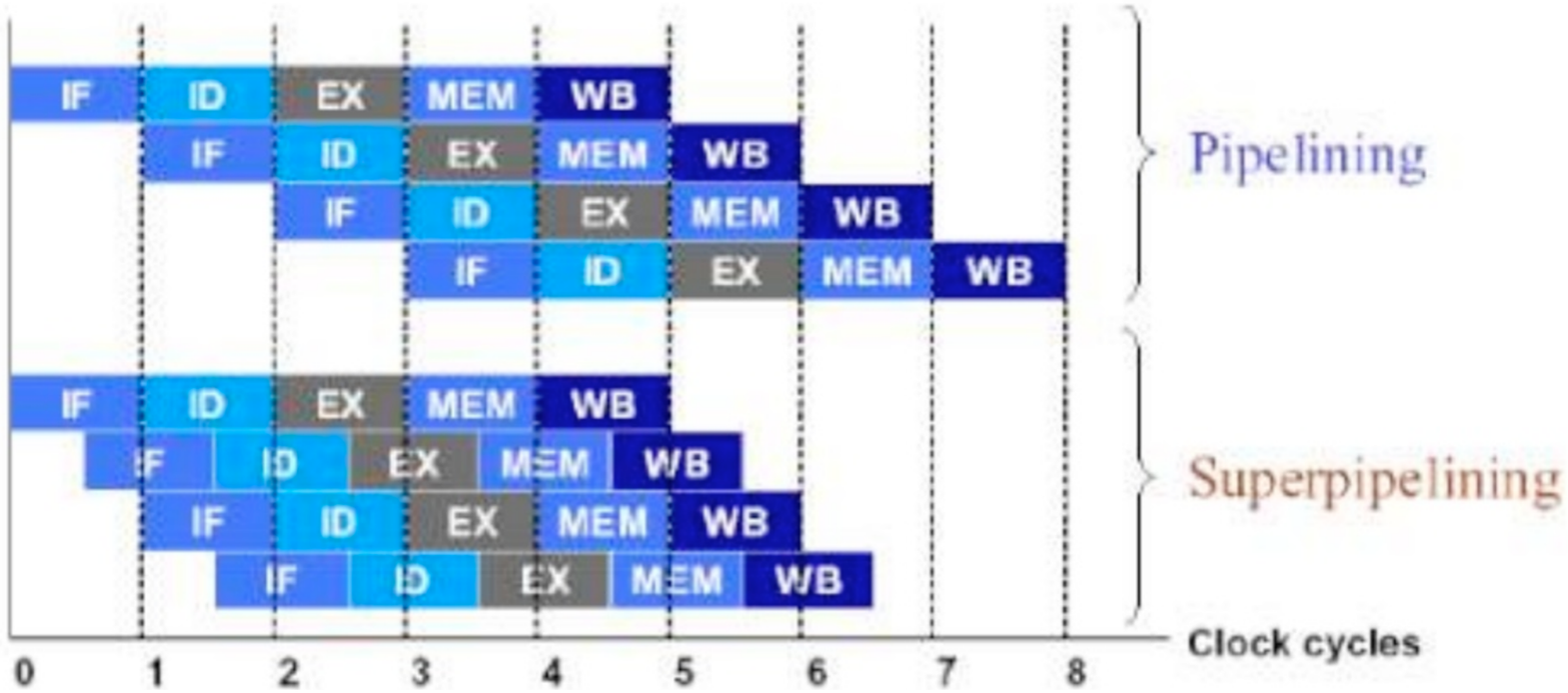
- **Predict + early detection + fill (1 delay) with instructions after brach**

Review

- **Exception handling**
- **Steps:**
    - 1. Save PC of offending (or interrupted) instruction (In MIPS: Exception Program Counter (EPC))
    - 2. Save indication of the problem (In MIPS: Cause register)
    - 3. Jump to handler at 8000 00180
- **Or, steps:**
    - 1. Save PC of offending (or interrupted) instruction (In MIPS: Exception Program Counter (EPC))
    - 2. jump to handler, whose address determined by the cause
- **Than, handler:**
    - 1. If restartable: Take corrective action; use EPC to return to program
    - 2. otherwise: Terminate program; Report error using EPC, cause
- **Moreover, for pipeling processor: (much the same as mis-predicted branch)**
    - 1. early detection to prevent other instructions from being clobbered
    - 2. Complete previous instructions
    - 3. Flush add and subsequent instructions
    - 4. + above steps

Review

- **Super pipelining (instruction-level parallelism)**



- CPI > 1
- Data dependency
- Procedural dependency
- Solution: **(all about speculation!, and thus, also unrolling)**
    - static scheduling (complier's matter)
    - dynamic scheduling (processor's matter)

Review

- **Super pipelining (instruction-level parallelism)**

```
Loop:   lw      $t0, 0($s1)
        addu    $t0, $t0, $s2
        sw      $t0, 0($s1)
        addi    $s1, $s1, -4
        bne     $s1, $zero, Loop
```

Insertion

	ALU or branch	Data transfer	Clock cycle
Loop:		lw $t0, 0($s1)	1
	addi $s1, $s1, -4		2
	addu $t0, $t0, $s2		3
	bne $s1, $zero, Loop	sw $t0, 4($s1)	4

```
Loop:   lw      $t0, 0($s1)
        addu    $t0, $t0, $s2
        sw      $t0, 0($s1)
        addi    $s1, $s1, -4
        lw      $t0, 0($s1)
        addu    $t0, $t0, $s2
        sw      $t0, 0($s1)
        addi    $s1, $s1, -4
        lw      $t0, 0($s1)
        addu    $t0, $t0, $s2
        sw      $t0, 0($s1)
        addi    $s1, $s1, -4
        lw      $t0, 0($s1)
        addu    $t0, $t0, $s2
        sw      $t0, 0($s1)
        addi    $s1, $s1, -4
        bne     $s1, $zero, Loop
```

Review

- **Super pipelining (instruction-level parallelism)**

Prediction

```
Loop:   lw      $t0, 0($s1)
        addu    $t0, $t0, $s2
        sw      $t0, 0($s1)
        addi    $s1, $s1, -4
        lw      $t0, 0($s1)
        addu    $t0, $t0, $s2
        sw      $t0, 0($s1)
        addi    $s1, $s1, -4
        lw      $t0, 0($s1)
        addu    $t0, $t0, $s2
        sw      $t0, 0($s1)
        addi    $s1, $s1, -4
        lw      $t0, 0($s1)
        addu    $t0, $t0, $s2
        sw      $t0, 0($s1)
        addi    $s1, $s1, -4
        bne     $s1, $zero, Loop
```

```
Loop:   lw      $t0, 0($s1)
        addu    $t0, $t0, $s2
        sw      $t0, 0($s1)
        lw      $t0, -4($s1)
        addu    $t0, $t0, $s2
        sw      $t0, -4($s1)
        lw      $t0, -8($s1)
        addu    $t0, $t0, $s2
        sw      $t0, -8($s1)
        lw      $t0, -12($s1)
        addu    $t0, $t0, $s2
        sw      $t0, -12($s1)
        addi    $s1, $s1, -16
        bne     $s1, $zero, Loop
```

Review

- **Super pipelining (instruction-level parallelism)**

Decoupling

```
Loop:   lw      $t0, 0($s1)
        addu    $t0, $t0, $s2
        sw      $t0, 0($s1)
        lw      $t0, -4($s1)
        addu    $t0, $t0, $s2
        sw      $t0, -4($s1)
        lw      $t0, -8($s1)
        addu    $t0, $t0, $s2
        sw      $t0, -8($s1)
        lw      $t0, -12($s1)
        addu    $t0, $t0, $s2
        sw      $t0, -12($s1)
        addi    $s1, $s1, -16
        bne     $s1, $zero, Loop
```

```
Loop:   lw      $t0, 0($s1)
        addu    $t0, $t0, $s2
        sw      $t0, 0($s1)
        lw      $t1, -4($s1)
        addu    $t1, $t1, $s2
        sw      $t1, -4($s1)
        lw      $t2, -8($s1)
        addu    $t2, $t2, $s2
        sw      $t2, -8($s1)
        lw      $t3, -12($s1)
        addu    $t3, $t3, $s2
        sw      $t3, -12($s1)
        addi    $s1, $s1, -16
        bne     $s1, $zero, Loop
```

Review

- **Super pipelining (instruction-level parallelism)**

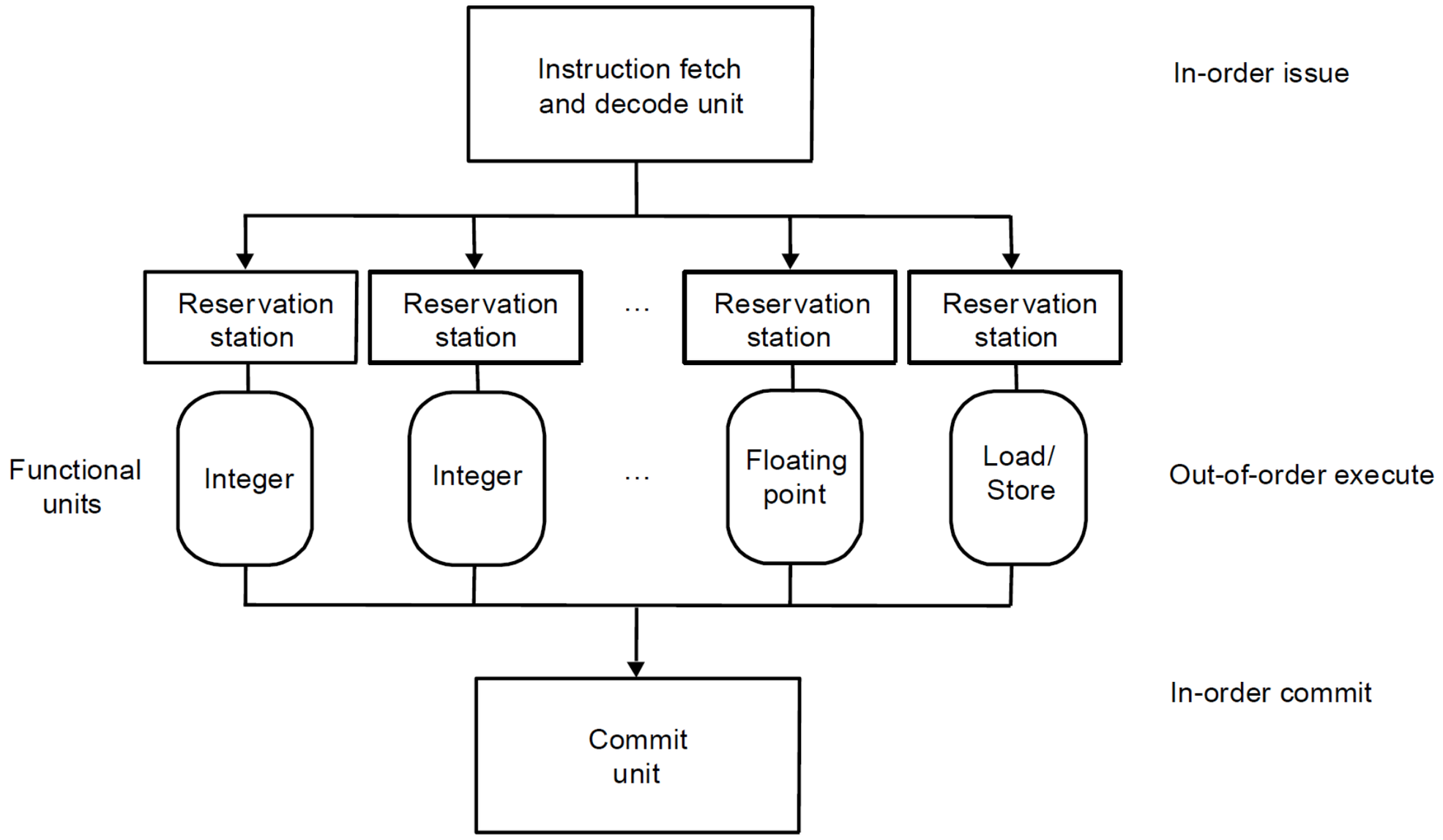
Out-of-order Insertion

	ALU or branch	Data transfer	Clock cycle
Loop:	addi $s1, $s1, -16	lw $t0, 0($s1)	1
		lw $t1, 12($s1)	2
	addu $t0, $t0, $s2	lw $t2, 8($s1)	3
	addu $t1, $t1, $s2	lw $t3, 4($s1)	4
	addu $t2, $t2, $s2	sw $t0, 16($s1)	5
	addu $t3, $t3, $s2	sw $t1, 12($s1)	6
		sw $t2, 8($s1)	7
	bne $s1, $zero, Loop	sw $t3, 4($s1)	8

```
Loop: lw   $t0, 0($s1)
      addu $t0, $t0, $s2
      sw   $t0, 0($s1)
      lw   $t1, -4($s1)
      addu $t1, $t1, $s2
      sw   $t1, -4($s1)
      lw   $t2, -8($s1)
      addu $t2, $t2, $s2
      sw   $t2, -8($s1)
      lw   $t3, -12($s1)
      addu $t3, $t3, $s2
      sw   $t3, -12($s1)
      addi $s1, $s1, -16
      bne  $s1, $zero, Loop
```

Review

- **Super pipelining (instruction-level parallelism)**



Review

- **Conclusion for pipeline**
- **CPU time = #instructions * cycles/instruction * cycle time**
    - Pros of having pipelining compared to **single cycle**
        - CT is reduced
        - CPI is the same (why)
    - Cons of having pipelining compared to **single cycle**
        - CPI can be larger than the ideal case one
        - hazards to deal with (software way, hardware way)

Sample question 1

- **Consider the data hazards:**

- what is the min. number of stalls needed for below functions? The tech we can use here are:1) stalls, 2) forwarding.

```
add $1, $2, $3
add $4, $5, $6
beq $1, $4, target
```

```
lw  $1, addr
add $4, $5, $6
beq $1, $4, target
```

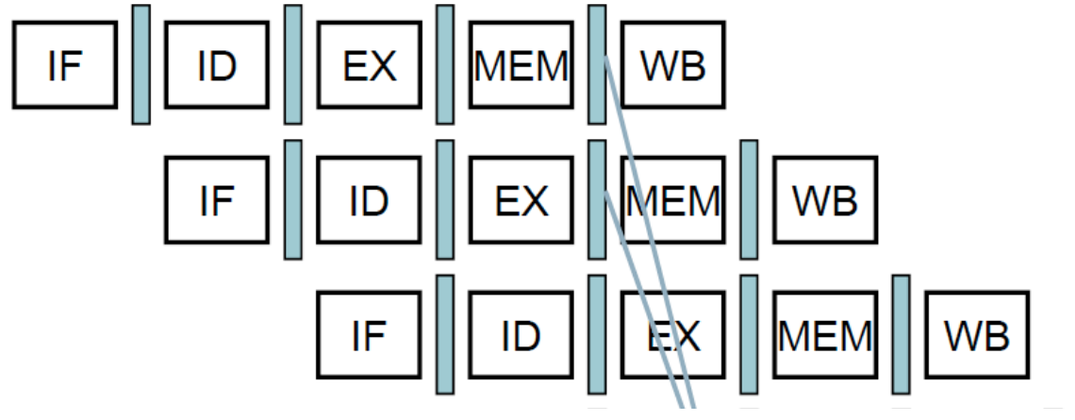
```
lw  $1, addr
beq $1, $0, target
```

Sample question 1

```
add $1, $2, $3

add $4, $5, $6

beq $1, $4, target
```



Sample question 1

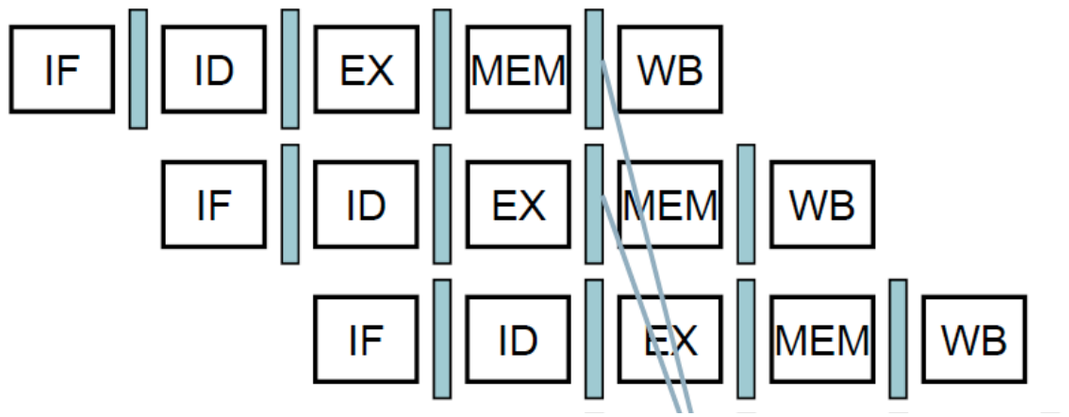
- **Consider the data hazards:**

- what is the min. number of stalls needed for below functions? The tech we can use here are:1) stalls, 2) forwarding.

```
lw  $1, addr

add $4, $5, $6

beq $1, $4, target
```



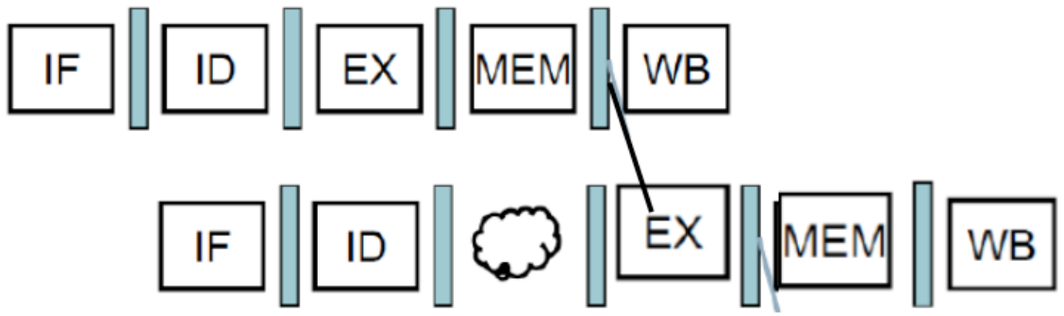
Sample question 1

- **Consider the data hazards:**

- what is the min. number of stalls needed for below functions? The tech we can use here are:1) stalls, 2) forwarding.

```
lw  $1, addr

beq $1, $0, target
```



Sample question 2

- **What is clock time (CT), instruction count(IC), cycle count(cc), cycle per instruction (CPI), CPU time for this program? (assume 1 ns for one-cycle data path, each stage takes the same time)**

```
sub   $2,  $1,  $3
add   $12, $3,  $5
nop
or    $13, $6,  $2
add   $14, $12, $2
nop
nop
sw    $14, 100($2)
```

- **1) For inserting nops.**
- **2) For inserting bubbles.**
- **3) For data forwarding.**

Sample question 2

Option 1: Compiler inserts NOPs

	1	2	3	4	5	6	7	8	9	10	11	12
sub $2,$1,$3	IF	ID	EX	MEM	WB							
add $12,$3,$5		IF	ID	EX	MEM	WB						
nop			IF	ID	EX	MEM	WB					
or $13,$6,$2				IF	ID	EX	MEM	WB				
add $14,$12,$2					IF	ID	EX	MEM	WB			
nop						IF	ID	EX	MEM	WB		
nop							IF	ID	EX	MEM	WB	
sw $14,100($2)								IF	ID	EX	MEM	WB

Performance

CT = 1 ns / 5 = 0.2 ns

IC = 5 + 3 = 8

CC = (5-1) + 8 = 12

CPI = 12 / 8 = 1.5

T = 12 * 0.2 ns = 2.4 ns

Sample question 2

Option 2: Hardware inserts bubbles (stalls)

	1	2	3	4	5	6	7	8	9	10	11	12
sub $2,$1,$3	IF	ID	EX	MEM	WB							
add $12,$3,$5		IF	ID	EX	MEM	WB						
BUBBLE					EX	MEM	WB					
or $13,$6,$2			IF	ID	ID	EX	MEM	WB				
add $14,$12,$2		IF	IF	ID	EX	MEM	WB					
BUBBLE						EX	MEM	WB				
BUBBLE							EX	MEM	WB			
sw $14,100($2)			IF	ID	ID	ID	EX	MEM	WB			

Performance

CT = 1 ns / 5 = 0.2 ns

IC = 5

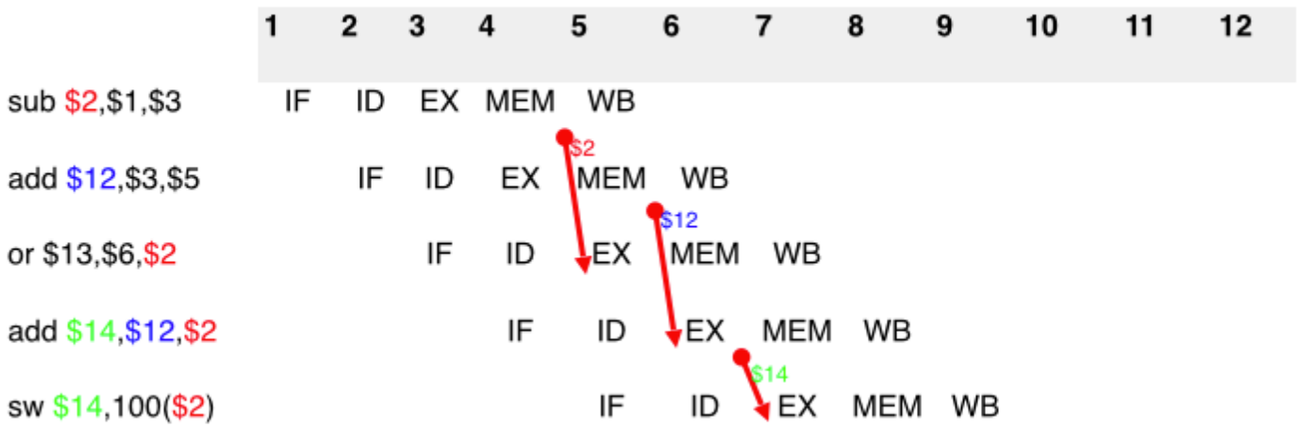
CC = (5-1) + 5 + 3 = 12

CPI = 12 / 5 = 2.4

T = 12 * 0.2 ns = 2.4 ns

Sample question 2

Option 3: Forwarding



Performance

CT = 1 ns / 5 = 0.2 ns

IC = 5

CC = (5-1) + 5 = 9

CPI = 9 / 5 = 1.8

T = 9 * 0.2 ns = 1.8 ns

Sample question 2

- **Single cycle data path design?**

	CT	IC	CC	CPI	T
Single-Cycle Datapath	1 ns	5	5	1	5 ns