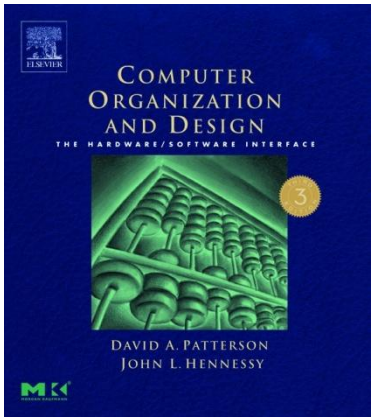
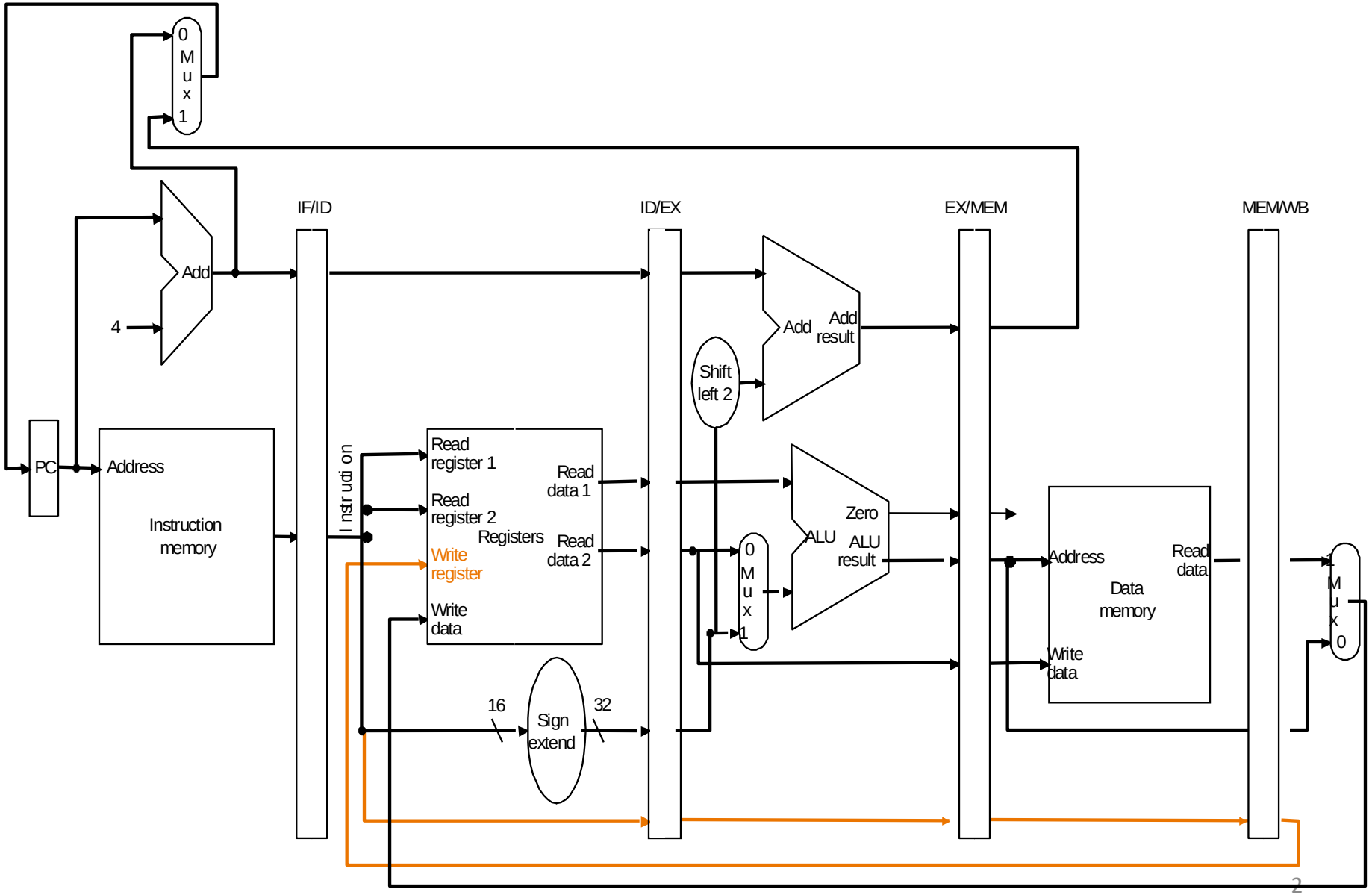
MIPS pipelining

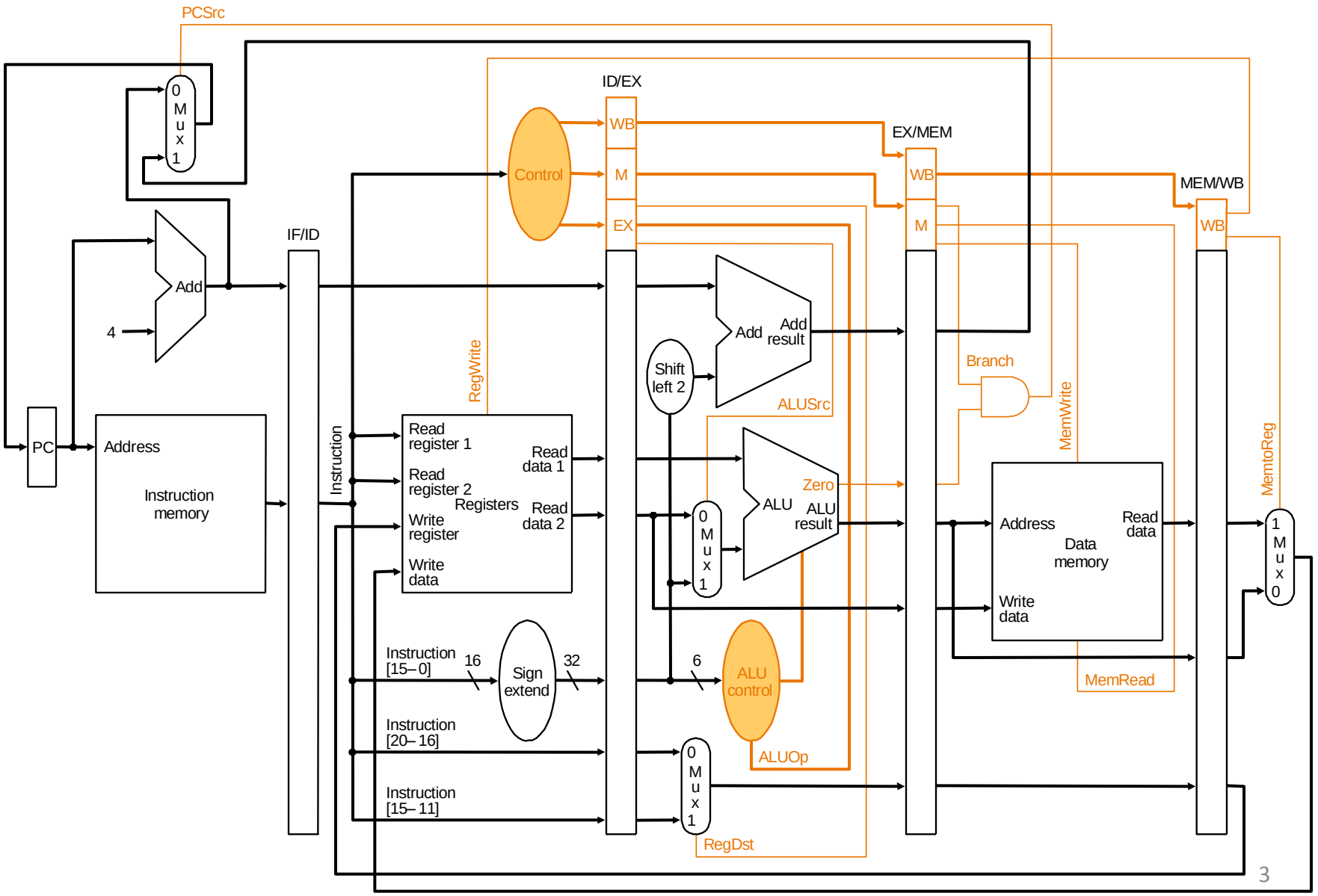
Pictures from:



MIPS Datapath



Datapath with Control



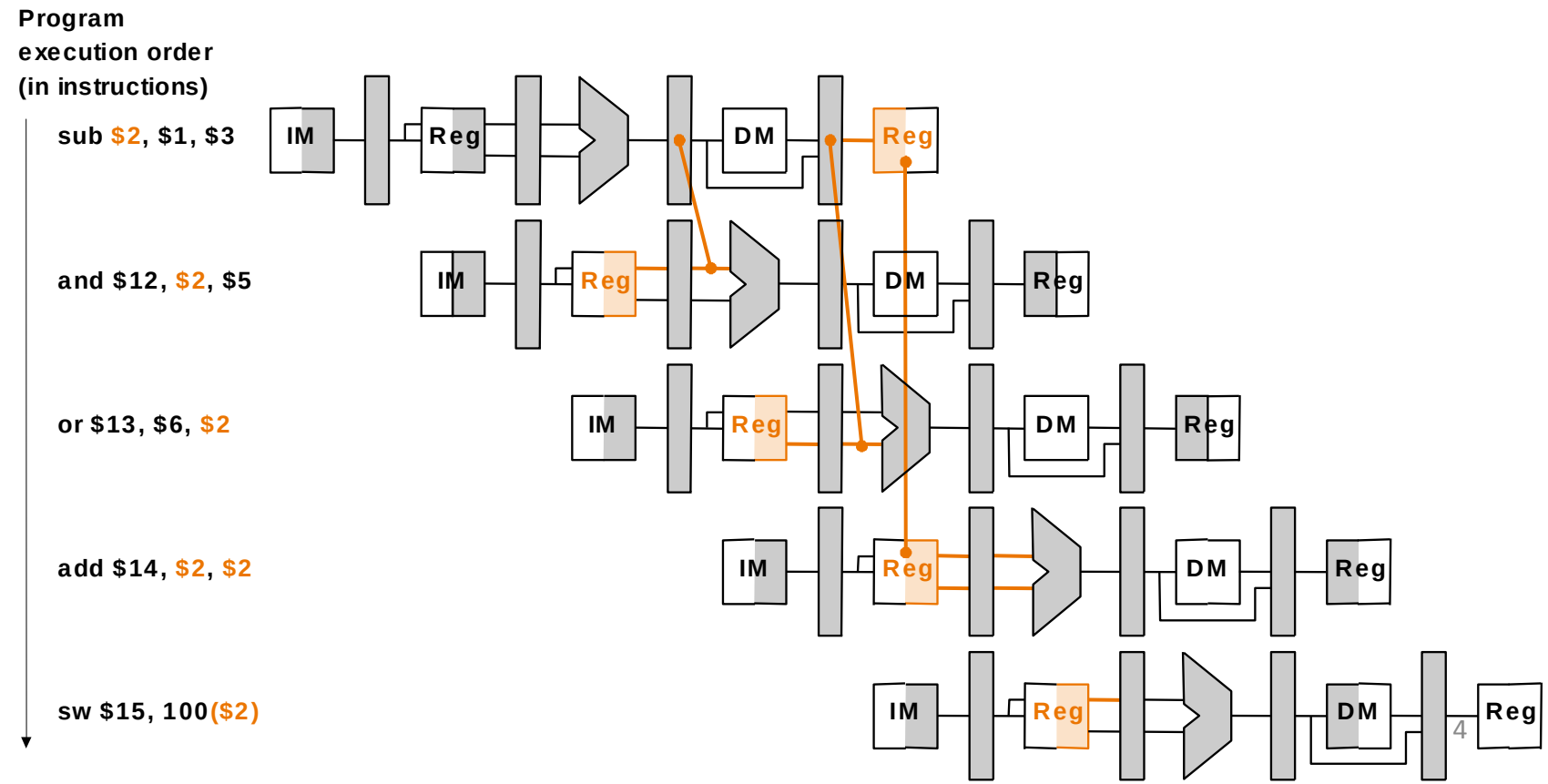
Forwarding:

Use temporary results, don't wait for them to be written

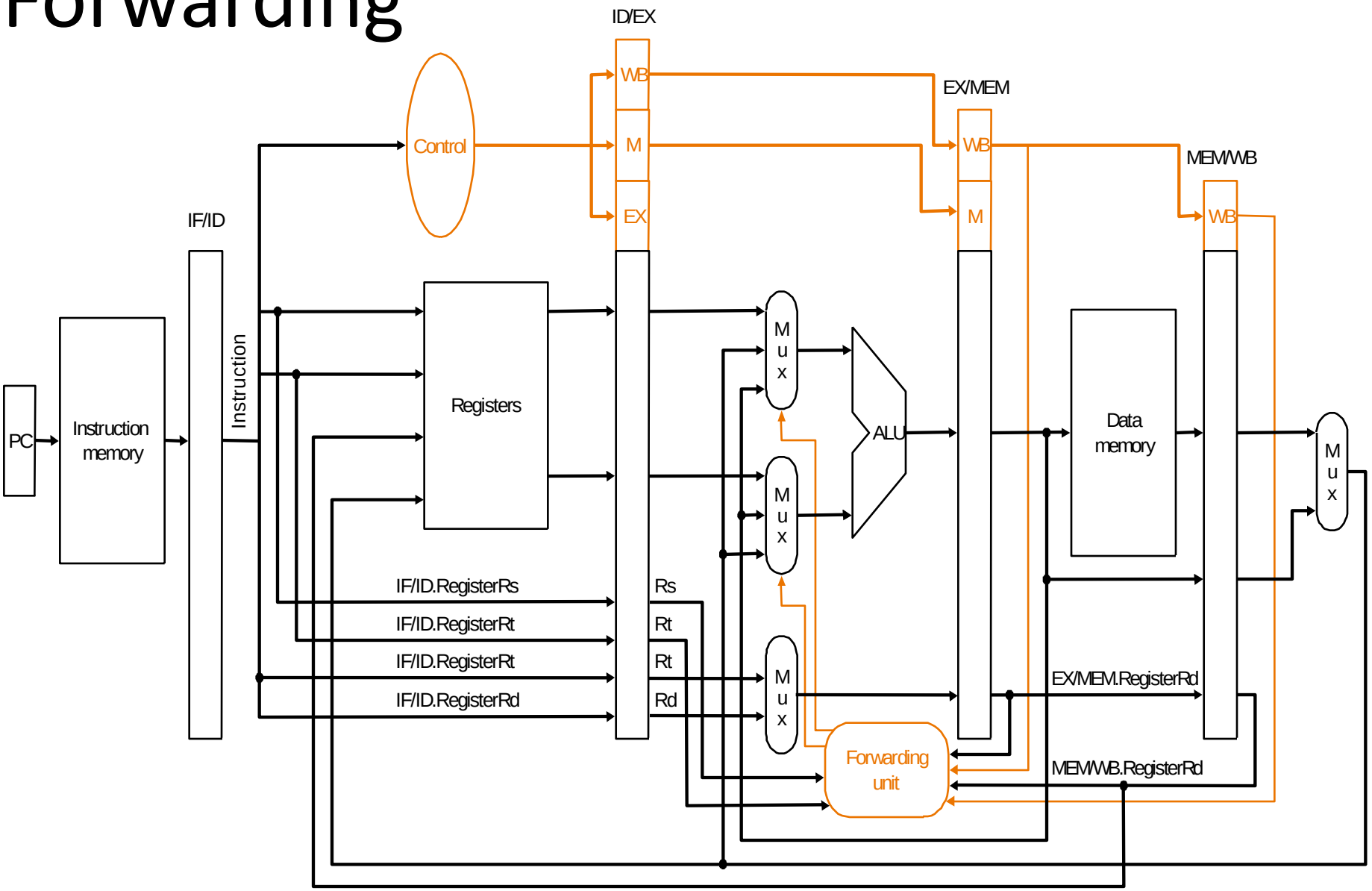
- register file forwarding to handle read/write to same register
- ALU forwarding

Time (in clock cycles) →

	CC 1	CC 2	CC 3	CC 4	CC 5	CC 6	CC 7	CC 8	CC 9
Value of register $2 :	10	10	10	10	10/– 20	– 20	– 20	– 20	– 20
Value of EX/MEM :	X	X	X	– 20	X	X	X	X	X
Value of MEM/WB :	X	X	X	X	– 20	X	X	X	X



Forwarding


ID/EX
EX/MEM
MEM/WB
Control
WB
M
EX
IF/ID
PC
Instruction memory
Instruction
Registers
Mux
ALU
Data memory
IF/ID.RegisterRs
IF/ID.RegisterRt
IF/ID.RegisterRt
IF/ID.RegisterRd
Rs
Rt
Rt
Rd
EX/MEM.RegisterRd
MEM/WB.RegisterRd
Forwarding unit

Forwarding check

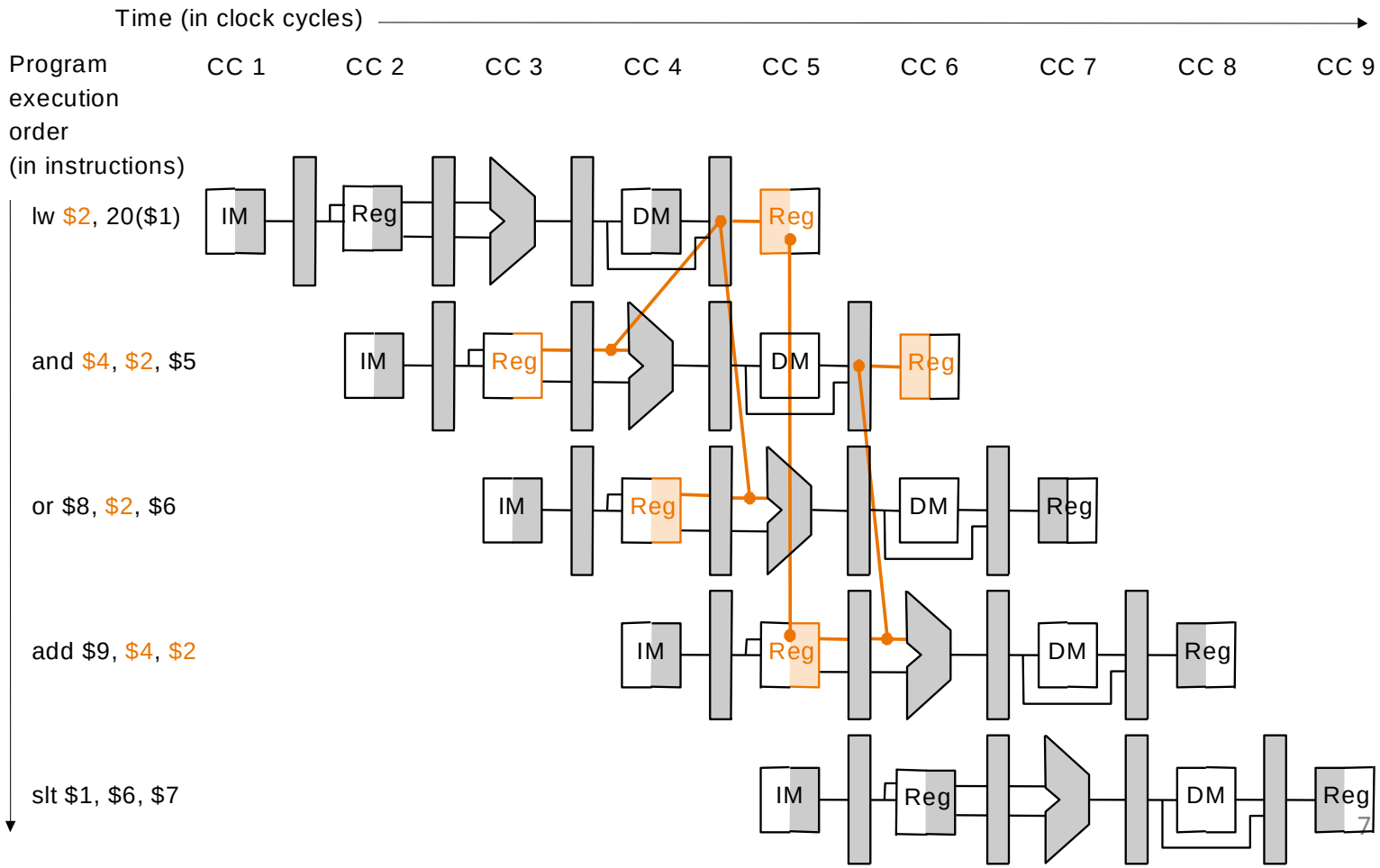
- Check for matching register-ids:
- For each source-id of operation in the EX-stage check if there is a matching pending dest-id

Example:

```
if (EX/MEM.RegWrite) ∧
   (EX/MEM.RegisterRd ≠ 0) ∧
   (EX/MEM.RegisterRd = ID/EX.RegisterRs)
then ForwardA = 10
```

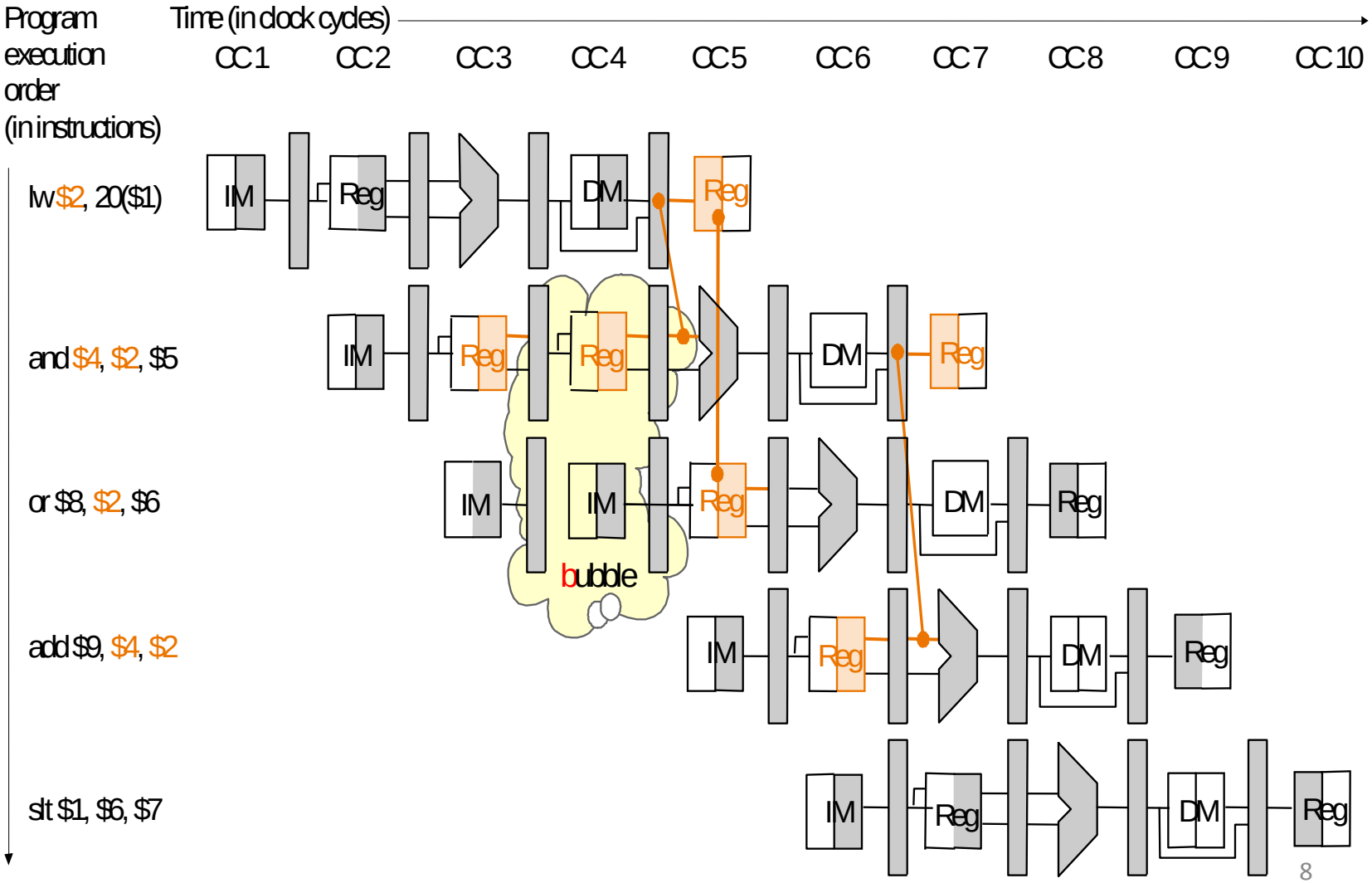
Can't always forward

- Load word can still cause a hazard:
  - an instruction tries to read register **r** following a load to the same **r**
- Need a hazard detection unit to “stall” the load instruction

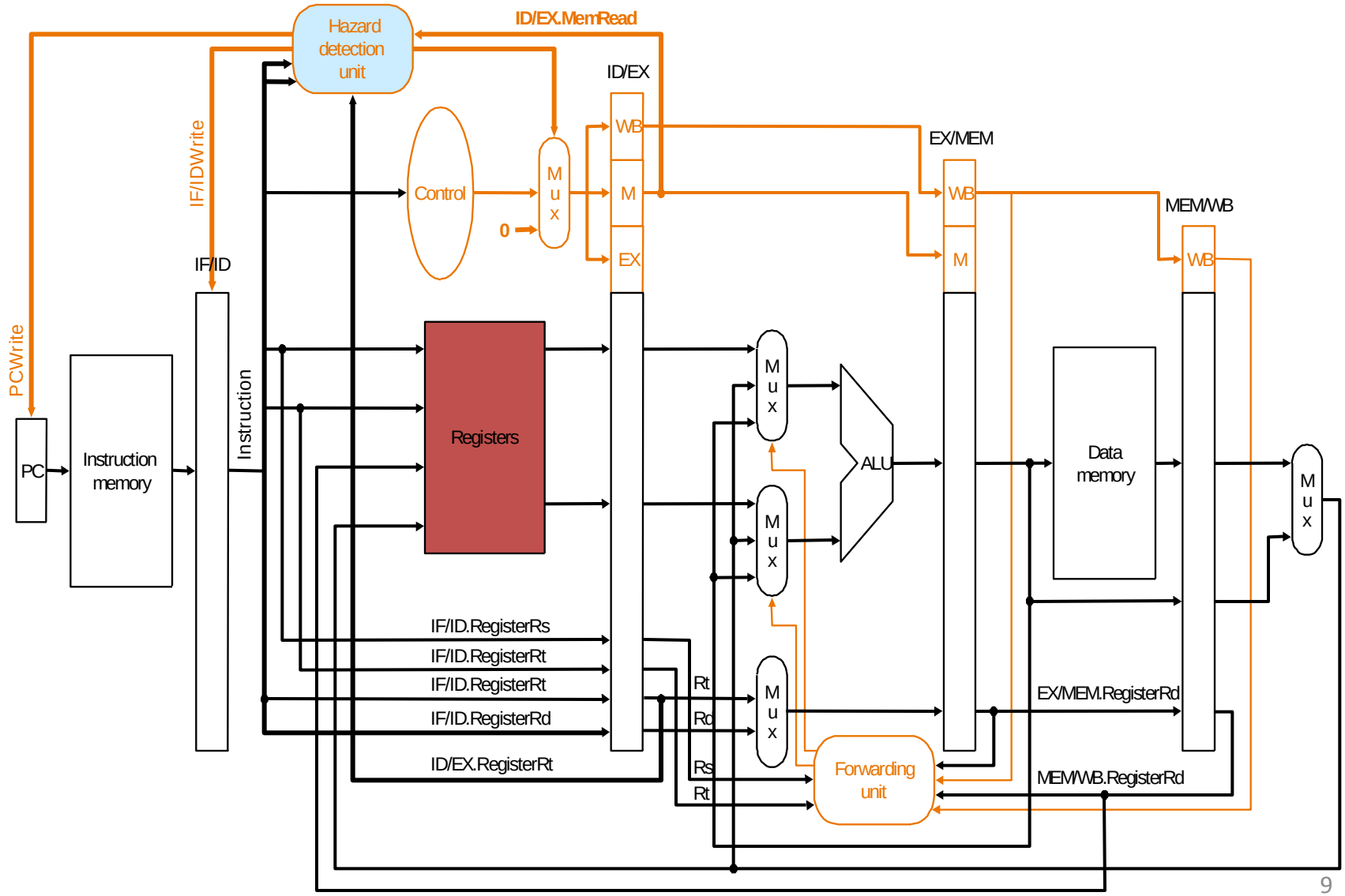


Stalling

We can stall the pipeline by keeping an instruction in the same stage



Hazard Detection Unit



Software only solution?

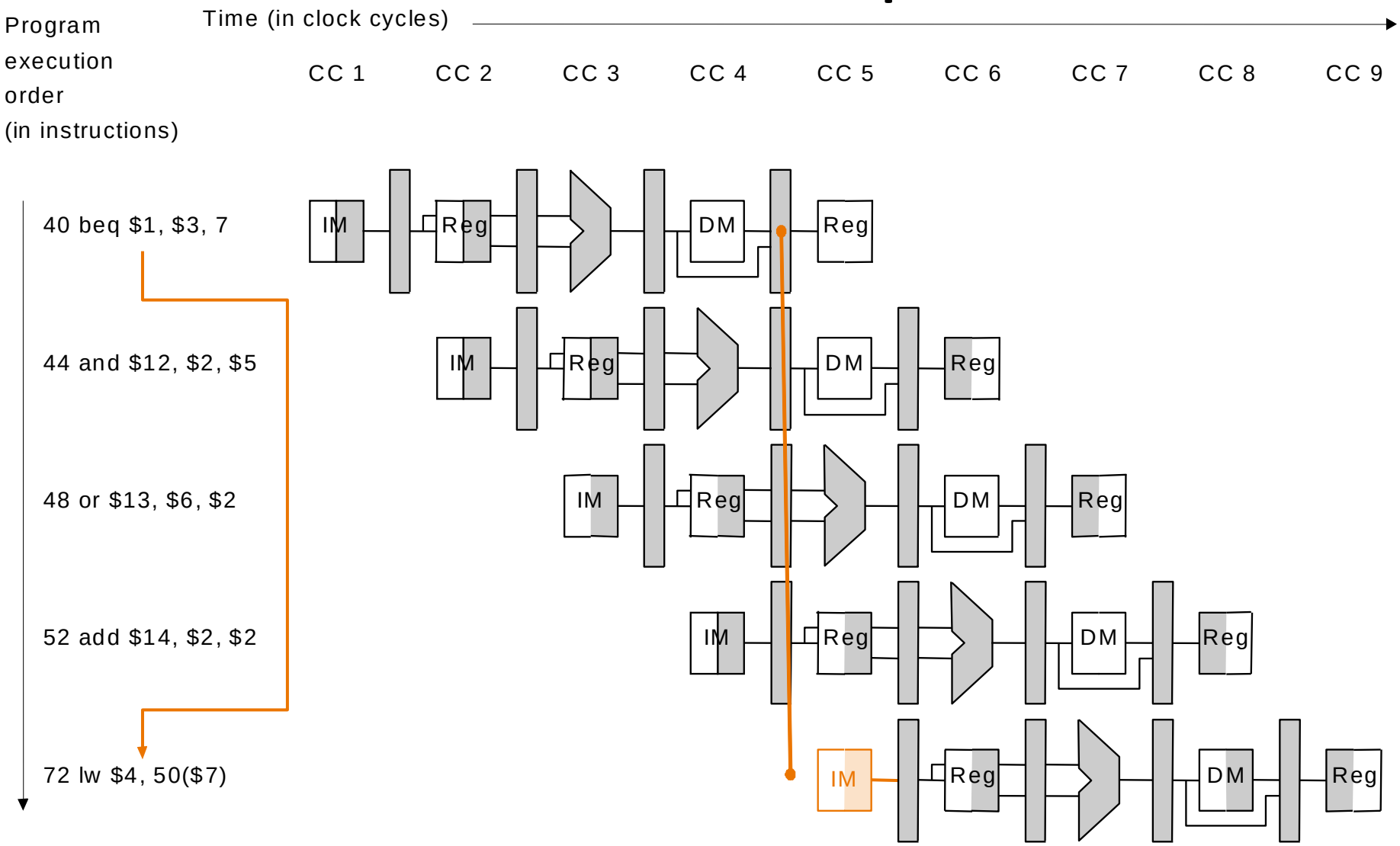
- Have compiler guarantee that no hazards occur
- Example: where do we insert the "NOPs" ?

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

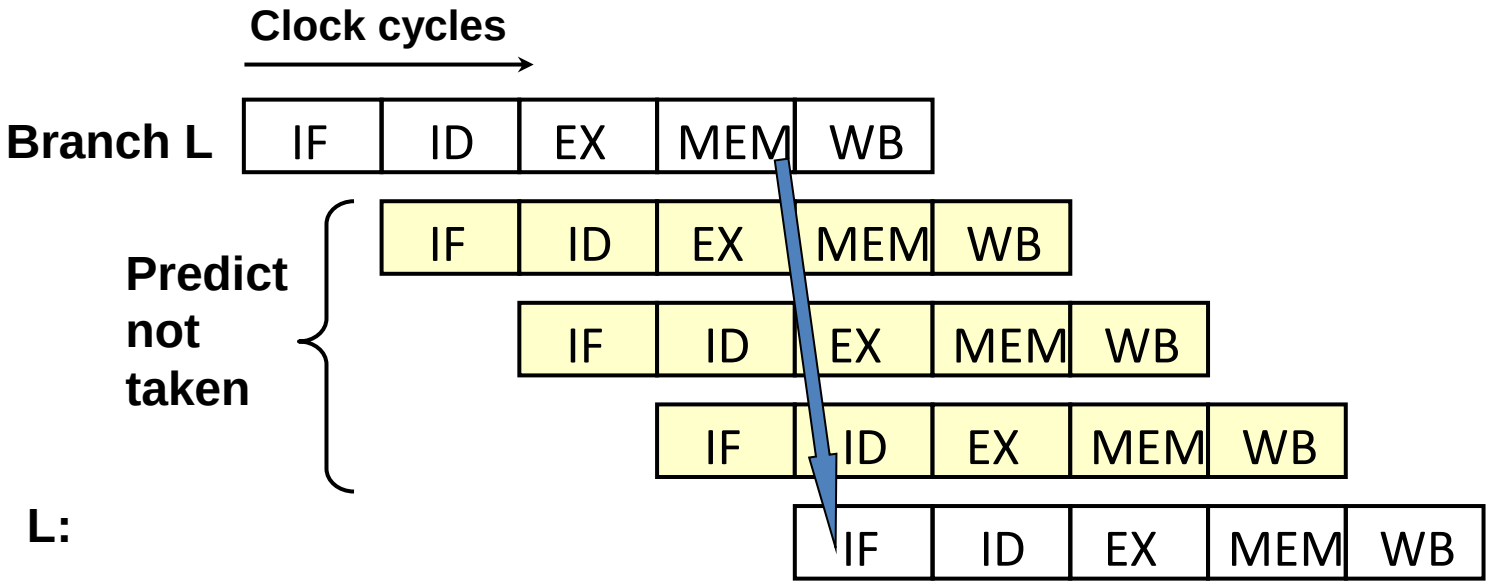
Problem: this really slows us down!

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

Branch example



Branch with predict not taken

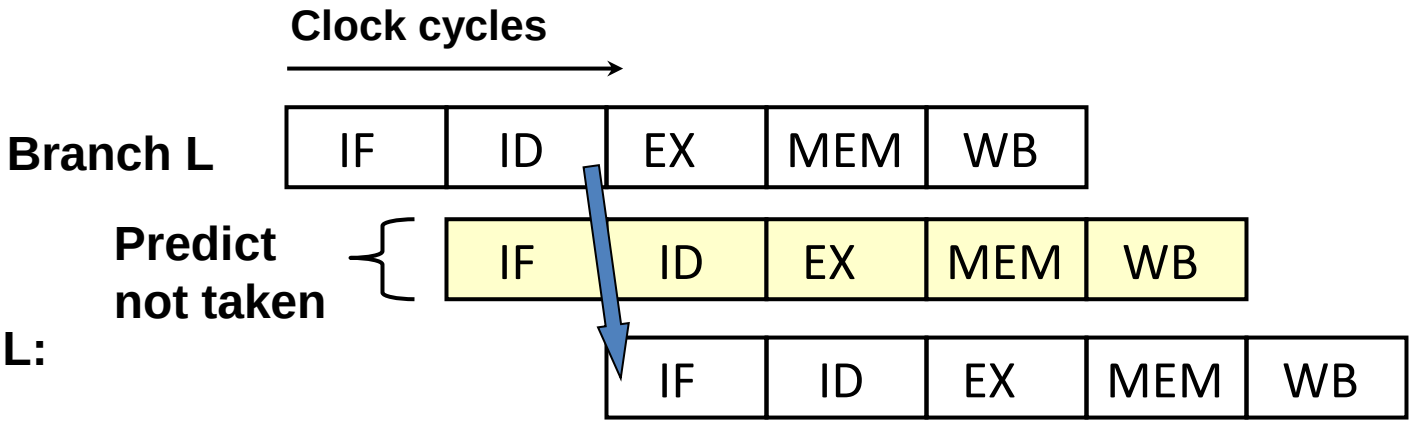


Branch speedup

Earlier address computation

Earlier condition calculation

Put both in the ID pipeline stage (adder + comparator)



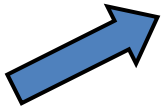
Scheduling data hazards example

Avoiding stalls:

Reordering instructions for following program

Code:

```
a = b + c
d = e - f
```



Unscheduled code:

```
Lw   R1,b
Lw   R2,c
Add  R3,R1,R2      interlock
Sw   a,R3
Lw   R1,e
Lw   R2,f
Sub  R4,R1,R2      interlock
Sw   d,R4
```

Scheduled code:

```
Lw   R1,b
Lw   R2,c
Lw   R5,e     extra reg. needed!
Add  R3,R1,R2
Lw   R2,f
Sw   a,R3
Sub  R4,R5,R2
Sw   d,R4
```

Scheduling control hazards

Add a branch delay slot:

the next instruction after a branch is **always** executed,

rely on compiler to "fill" the slot with something useful.

What to put in the delay slot?