
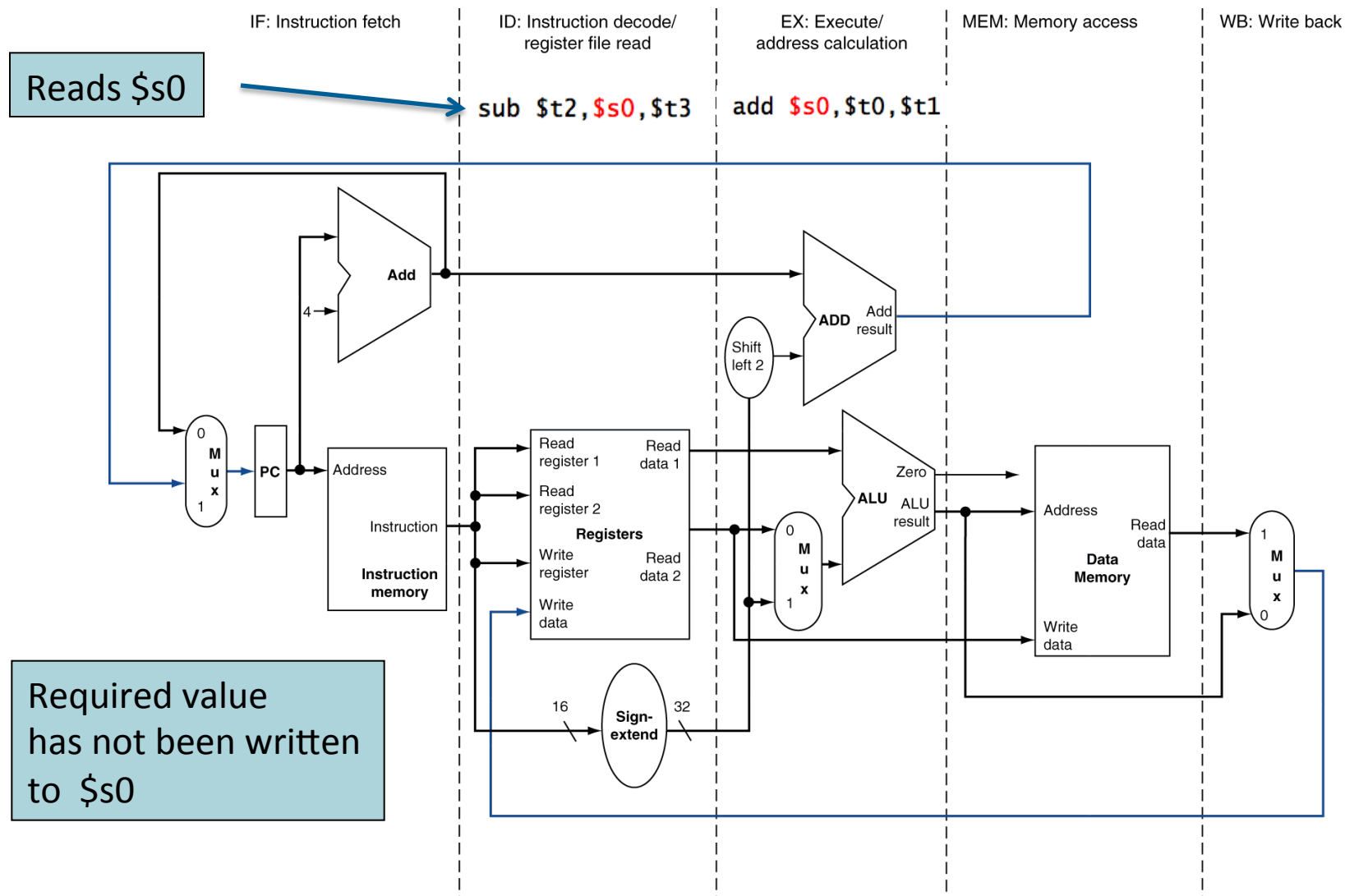


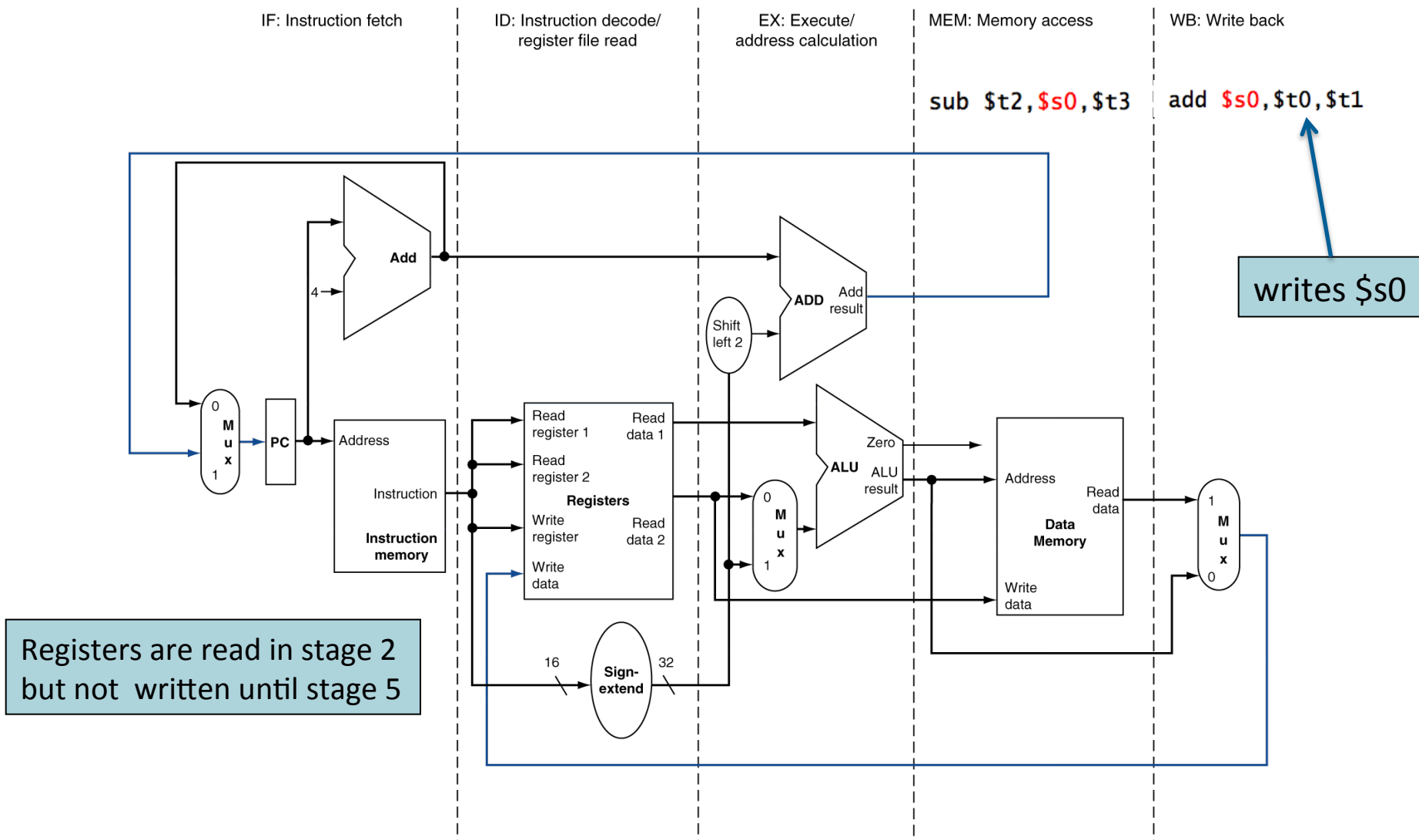
- An instruction depends on a result from a previous instruction

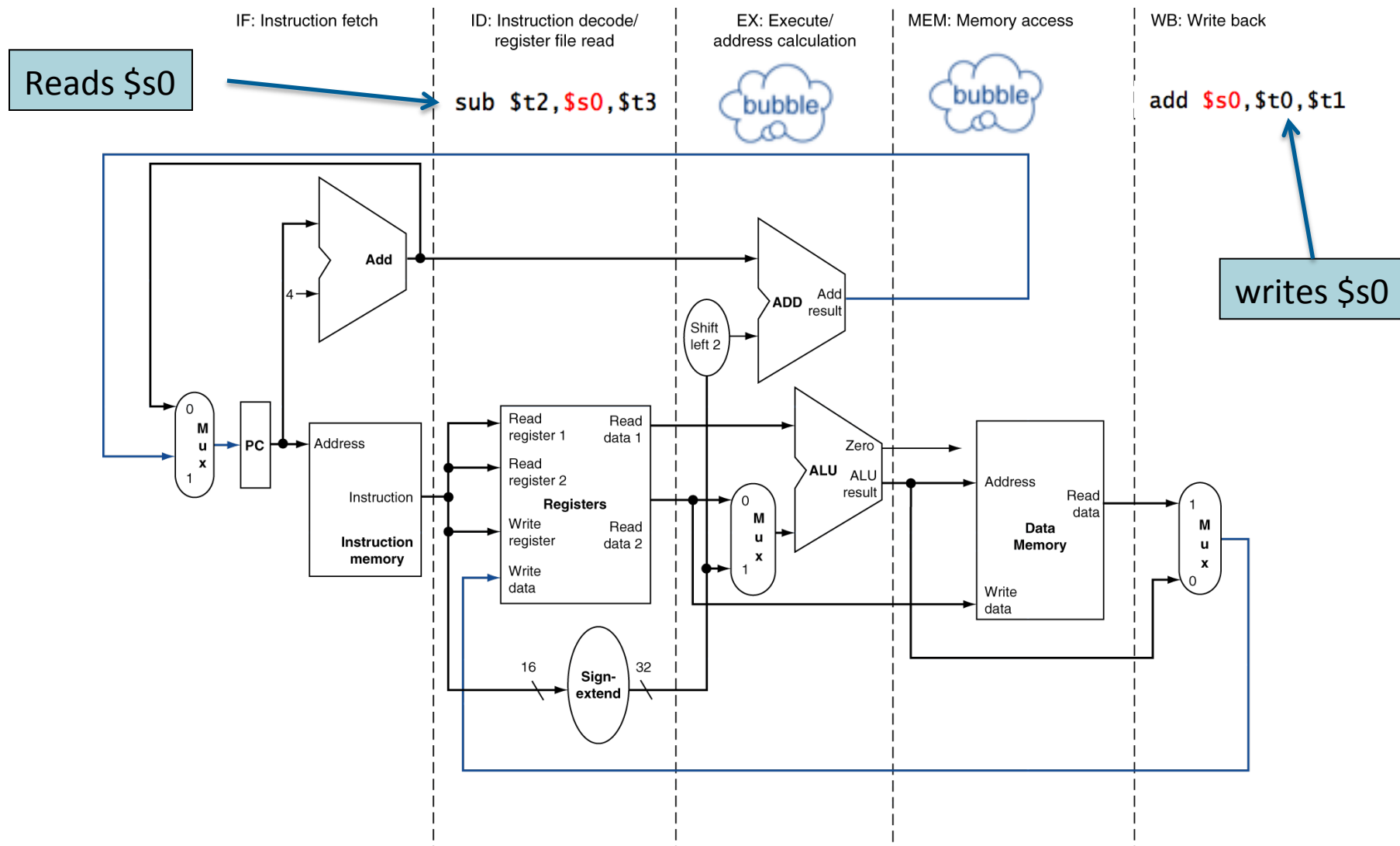
- add **\$s0**, \$t0, \$t1

- sub \$t2, **\$s0**, \$t3



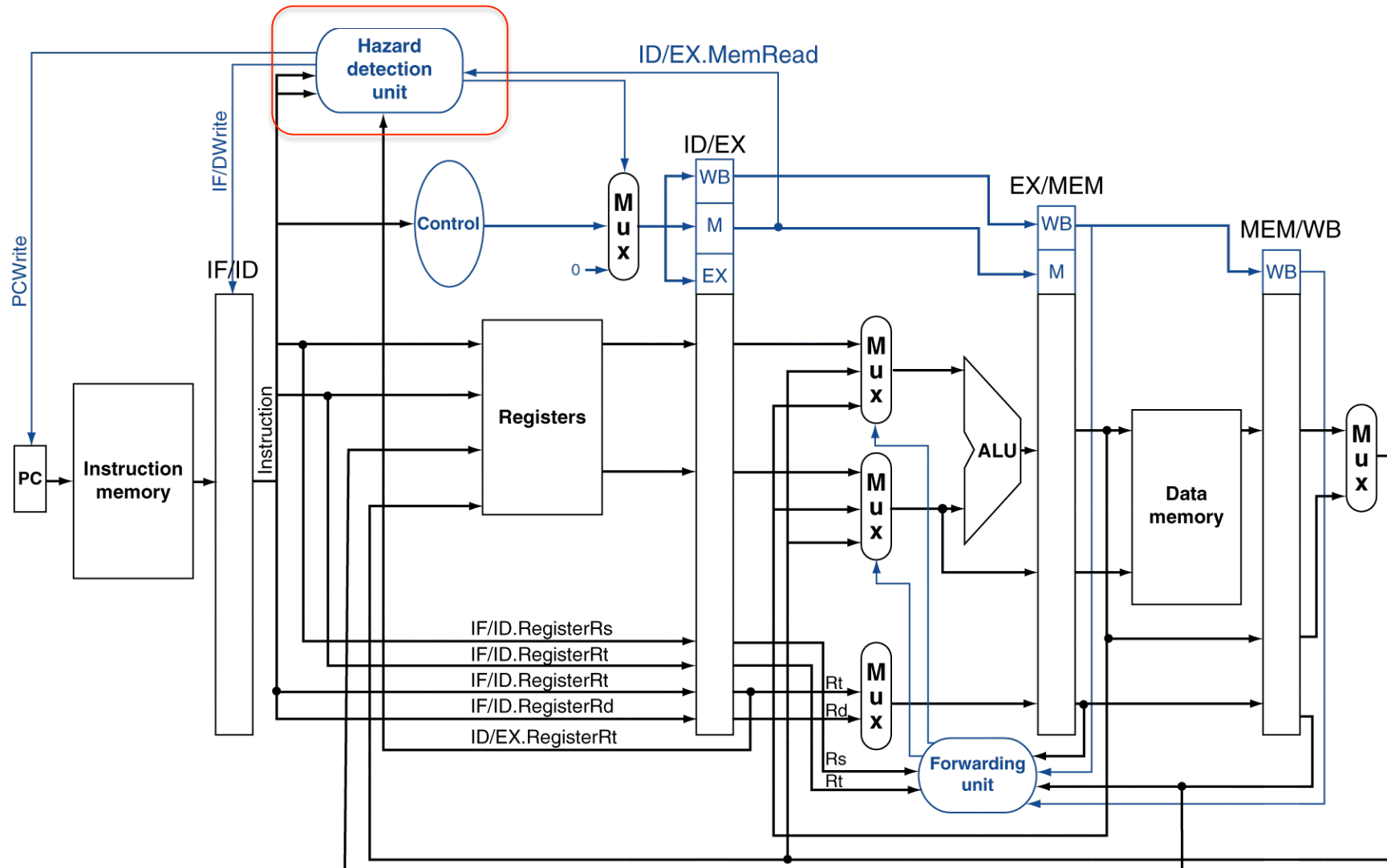






- Stalling is also known as *pipeline interlock*
 - Instructions in front continue to advance
 - Creates empty stages or *bubbles*
 - Consumes 2 extra clock cycles

- Without stalling, dependent instructions use stale data
 - Extra hardware is required to implement stalls
 - Hazard detection unit controls when stalls occur



Instruction in decode may be dependent

- If either of its 2 input registers matches:
- reg written by instruction in EXE or in MEM stage
- Hazard detection unit outputs:
 - 0 to allow normal control signals to be passed
 - 1 to instead substitute zero control signals (i.e. bubble)
- Instructions must write a result to a cause data hazard
 - Neither sw nor beq write a result

If (EX/MEM.RegWrite or MEM/WB.RegWrite)

- Does instruction in stage 4 or 5 write a register?

If (EX/MEM.Rd == ID/EX.Rs or EX/MEM.Rd == ID/EX.Rt) or













If (MEM/WB.Rd == ID/EX.Rs or MEM/WB.Rd == ID/EX.Rt)

- Result reg. match an input reg. for instruction in stage 2?

Hazard exists if both conditions are true


```
add  $s0, $t0, $t1  
sub  $t2, $s0, $t3  
slt  $t4, $t2, $0
```

- Hazard detection unit compares register numbers
- Sub must be stalled until add writes \$s0
- Slit must be stalled until sub writes \$t2
- Registers are written during the first half of a cycle
- Registers are read during the second half of a cycle
 - otherwise 3 rather than 2 bubbles would be required

Cycle	IF	ID	EX	MEM	WB
1	add \$s0,\$t0,\$t1				
2	sub \$t2,\$s0,\$t3	add \$s0,\$t0,\$t1			
3	slt \$t4,\$t2,\$0	sub \$t2,\$s0,\$t3	add \$s0,\$t0,\$t1		
4	slt \$t4,\$t2,\$0	sub \$t2,\$s0,\$t3		add \$s0,\$t0,\$t1	
5	slt \$t4,\$t2,\$0	sub \$t2,\$s0,\$t3			add \$s0,\$t0,\$t1
6		slt \$t4,\$t2,\$0	sub \$t2,\$s0,\$t3		
7		slt \$t4,\$t2,\$0		sub \$t2,\$s0,\$t3	
8		slt \$t4,\$t2,\$0			sub \$t2,\$s0,\$t3
9			slt \$t4,\$t2,\$0		
10				slt \$t4,\$t2,\$0	
11					slt \$t4,\$t2,\$0

Stalls add 4 extra cycles to the time required for these 3 instructions.

■ Software

- Compiler inserts useful instructions
 - 2 independent instructions between dependent instructions (code rearrangement)
 - 2 NOP instructions if useful ones can't be found

■ Hardware

- Required value is forwarded to dependent instruction
- Just-in-time replacement of stale ALU inputs