For the sequence of instructions shown below, show how they would progress through the pipeline.

For all of these problems:

- Stalls are indicated by placing the code of the stage where the hazard would be discovered in the succeeding square
- We will assume a standard 5 stage pipeline
  - (IF = Instruction Fetch, ID = Instruction Decode, EX = Execute, M = Memory Access, WB = Write Back)
- Assume that each stage of the pipeline takes just 1 clock cycle to finish.

Example 1:

- Assume that forwarding **HAS NOT** been implemented
- Assume that you **CANNOT** read and write a register in the same clock cycle

Instruction	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Add $5, $3, $4	**IF**	**ID**	**EX**	**M**	**WB**												
Add $6, $5, $7		**IF**	**ID**	**ID**	**ID**	**ID**	**EX**	**M**	**W**	**Add must wait until $5 written by previous add; reads $5 in ID stage**							
LW $7, 0($6)			**IF**	**IF**	**IF**	**IF**	**ID**	**ID**	**ID**	**ID**	**EX**	**M**	**WB**	**LW stalled by prior add; needs $6 too – reads in CC #10**			
SUB $1, $2, $3							**IF**	**IF**	**IF**	**IF**	**ID**	**EX**	**M**	**WB**	**Pipeline full so SUB can't go**		
Add $9, $7, $8											**IF**	**ID**	**ID**	**ID**	**EX**	**M**	**WB**

(Last add must wait for $7 from LW)
(CPI = 17 /5 = 3.4 – not very good)

Example 2:

- Let's do the same problem as before, but now assume that **forwarding HAS been implemented**
- Assume that you CANNOT read and write from the same register in the register file in the same clock cycle

Instruction	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Add $5, $3, $4	**IF**	**ID**	**EX**	**M**	**WB**								**Data to be loaded into $5 available at end of CC 3 / Beginning of CC 4**				
Add $6, $5, $7		**IF**	**ID**	**EX**	**M**	**WB**							**Add gets data for $5 directly from output of ALU**				
LW $7, 0($6)			**IF**	**ID**	**EX**	**M**	**WB**						**Lw gets $6 data directly from output of ALU**				
SUB $1, $2, $3				**IF**	**ID**	**EX**	**M**	**WB**					**No dependencies on any other instructions**				
Add $9, $7, $8					**IF**	**ID**	**EX**	**M**	**WB**				**Add gets $7 data from latch between memory and writeback stage (its an input to ALU)**				

(Ability to forward eliminates *all* stalls!)
(CPI = 9 / 5 = 1.8 – much better – and would drop toward 0 as more instructions were executed.)

Example 3:

- Like Example 2, assume that **forwarding HAS been implemented**
- Assume that you **CAN** read and write a register in the same clock cycle

Instruction	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Add $1, $6, $9	**IF**	**ID**	**EX**	**M**	**WB**												
Add $6, $2, $4		**IF**	**ID**	**EX**	**M**	**WB**											
LW $7, 0($6)			**IF**	**ID**	**EX**	**M**	**WB**					**LW instruction producing result that will be stored in $7; even with forwarding must stall; data not available until end of CC #6 and needed at beginning of CC #6**					
SUB $1, $7, $8				**IF**	**ID**	**ID**	**EX**	**M**	**WB**								
Add $9, $1, $8					**IF**	**IF**	**ID**	**EX**	**M**	**WB**							