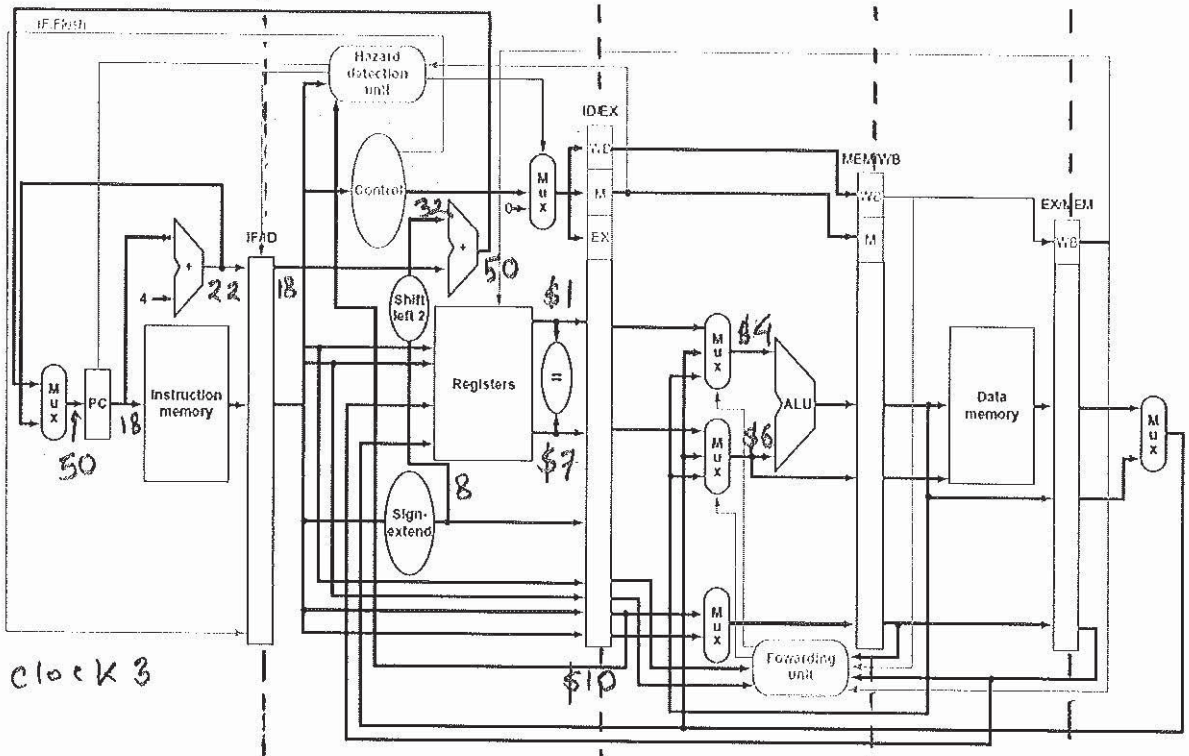
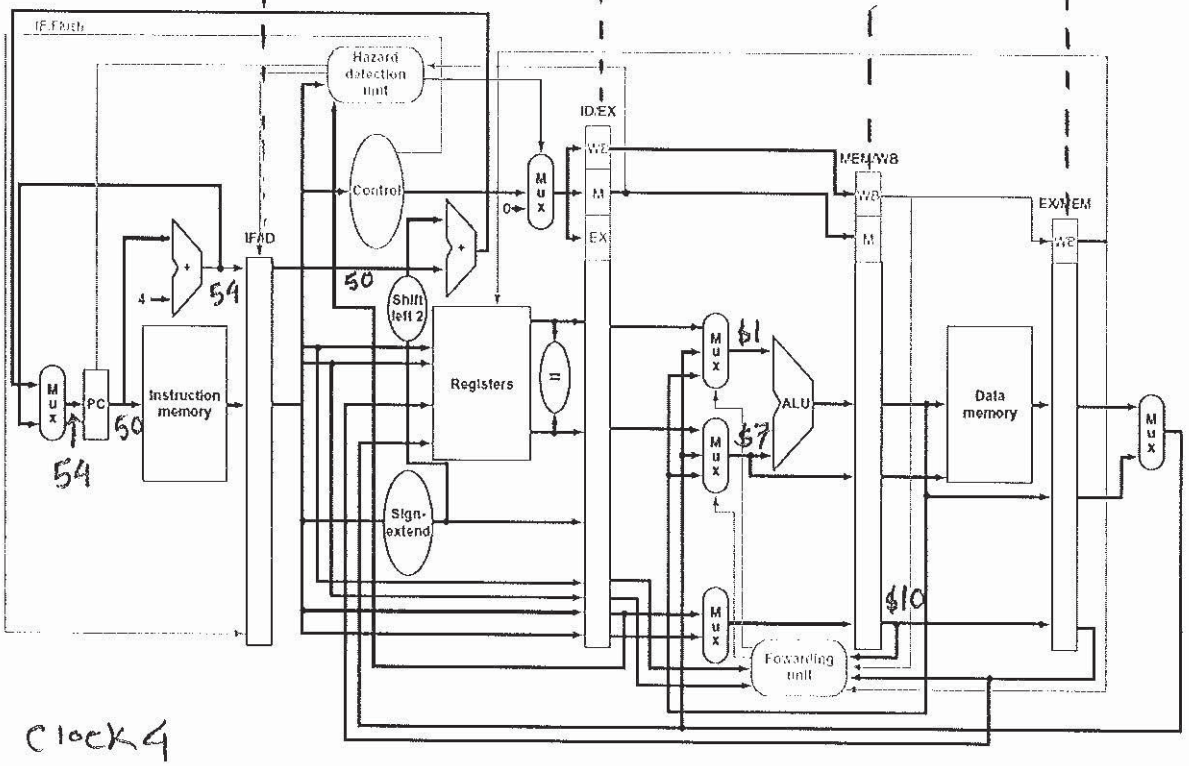
Q1

or $13, $2, $6 | beq $1, $7, 8 | add $10, $4, $6 | before <17 | before <17



clock 3

sw $4, 50($7) | bubble (nop) | beq $1, $7, 8 | add $10, $4, $6



clock 4

4.8.1 Pipelined

ID stage takes the longest (i.e. 350ps)

∴ clock cycle time = 350 ps

Non-pipelined

(IF + ID + Ex + MEM + WB)

clock cycle time = (250 + 350 + 150 + 300 + 200) ps

⇒ clock cycle time: 1250 ps

4.8.2 LW uses all stages

Non-pipelined

total latency = (250 + 350 + 150 + 300 + 200) ps

⇒ total latency = 1250 ps

Pipelined

Through put is improved, and latency only depends on the longest stage length.

⇒ total latency = (5 stages) · (ID stage)

= (5)(350 ps)

⇒ total latency = 1750 ps

4.8.3

Since ID stage is the longest, if you split its stage into two new stages, the time for each of those stages will be halved (i.e. 350/2 = 175ps). After the split, the new clock cycle time will depend on the longest stage, which is now MEM stage.

=> New clock cycle time after ID split = 300ps

4.8.4 Only "lw" and "sw" instructions deal with data memory.

Utilization = lw + sw = (20 + 15) %

=> Utilization = 35%

4.8.5 Only "lw" and "alu" instructions are utilized in this case.

Utilization = lw + alu = (20 + 45) %

=> Utilization = 65%

Solution:

4.13.1

	Instruction Sequence	Dependences
a.	I1: SW R16,-100(R6) I2: LW R4,8(R16) I3: ADD R5,R4,R4	RAW on R4 from I2 to I3

4.13.2 In the basic five-stage pipeline WAR and WAW dependences do not cause any hazards. Without forwarding, any RAW dependence between an instruction and the next two instructions (if register read happens in the second half of the clock cycle and the register write happens in the first half). The code that eliminates these hazards by inserting NOP instructions is:

	Instruction Sequence	

a.	SW R16,-100(R6)
	LW R4,8(R16)
	NOP
	NOP
	ADD R5,R4,R4
		Delay I3 to avoid RAW hazard on R4 from I2

4.13.3 With full forwarding, an ALU instruction can forward a value to the EX stage of the next instruction without a hazard. However, a load cannot forward to the EX stage of the next instruction (but can to the instruction after that). The code that eliminates these hazards by inserting NOP instructions is:

	Instruction Sequence	

a.	SW R16,-100(R6)
	LW R4,8(R16)
	NOP
	ADD R5,R4,R4
		Delay I3 to avoid RAW hazard on R4 from I2
		Value for R4 is forwarded from I2 now

4.13.4 The total execution time is the clock cycle time times the number of cycles. Without any stalls, a three-instruction sequence executes in 7 cycles (5 to complete the first instruction, then one per instruction). The execution without forwarding must add a stall for every NOP we had in 4.13.2, and execution forwarding must add a stall cycle for every NOP we had in 4.13.3. Overall, we get:

	No Forwarding	With Forwarding	Speedup Due to Forwarding
a.	(7 + 2) × 250ps = 2250ps	(7 + 1) × 300ps = 2400ps	0.94 (This is really a slowdown)

4.13.5 With ALU-ALU-only forwarding, an ALU instruction can forward to the next instruction, but not to the second-next instruction (because that would be forwarding from MEM to EX). A load cannot forward at all, because it determines the data value in MEM stage, when it is too late for ALU-ALU forwarding. We have:

	Instruction Sequence	

a.	`SW R16,-100(R6)`
	`LW R4,8(R16)`
	`ADD R5,R4,R4`
		ALU-ALU forwarding of R4 from I2

4.13.6

	No Forwarding	With ALU-ALU Forwarding Only	Speedup with ALU-ALU Forwarding
a.	(7 + 2) × 250ps = 2250ps	7 × 290ps = 2030ps	1.11