For the following code, explain what is happening in each stage of the pipelined processor during cycle 5.

	1	2	3	4	5	6	7	8	9
ADD X0, X31, #102	IF	DEC	EX	MEM	WB				
LDUR X1, [X0, #10]		IF	DEC	EX	MEM	WB			
STUR X0, [X0, #10]			IF	DEC	EX	MEM	WB		
CBZ X1, LOOP				IF	DEC	EX	MEM	WB	
EOR X6, X4, X1					IF	DEC	EX	MEM	WB

Instruction Fetch:

Bringing in the EOR instruction

Register Fetch/Decode:

Register file is reading X1. Forwarding unit decides to bring X1 in from the LDUR in the MEM stage.

Control logic processing the CBZ instruction into control signals.

Accelerated branches unit is determining the new PC in response to the CBZ instruction

Execute:

The ALU is computing the address for the STUR.

Memory:

Data is being read from the Data Memory for the LDUR

Write-back:

The result of the ADD is being written to register X0.

Our processor has a load delay slot, which means code cannot use the Rd register of a load in the instruction right after a load. What happens if we ignore this restriction? Specifically, assume the following code is given to the CPU. Assume MEM[32] = -4. What value will end up in register X5? Think carefully – this one is tricky!

```
ADDI  X3, X31, 24
LDUR X3, [X3, #8]
ADD   X5, X3, X3
```

When the ADD is in DECODE, forwarding grabs a value from EX, where the LDUR is being processed. The ALU has just computed the LDUR address, which is 24+8 = 32. Thus, the ADD believes X3 = 32. So, X5 becomes equal to 32+32 = 64.

The following code contains a "read after write" data hazard that is resolved by forwarding:

```
ADD   X2, X3, X4
ADD   X5, X2, X6
```

Consider the following code where a memory read occurs after a memory write:

	1	2	3	4	5	6
STUR X7, [X2, #100]	IF	DEC	EX	MEM	WB	
LDUR X8, [X2, #100]		IF	DEC	EX	MEM	WB

Does the code work correctly? Why/why not? Will the forwarding unit need to be altered to handle this code?

In cycle 4, the STUR writes a value into the Data Memory. This is finished by the end of cycle 4. The LDUR starts reading the data memory in cycle 5. Thus, the data is seen properly by the LDUR.

Code does work correctly, no need to change the forwarding unit.

In this question, we examine how pipelining affects the clock cycle time of the processor. Assume that individual stages of the datapath have the following latencies:

IF	ID	EX	MEM	WB
200ps	170ps	220ps	210ps	150ps

a.) What is the clock cycle time of the pipelined and single-cycle CPU?

b.) If we can split one stage of the pipelined datapath into two new stages, each with half the latency of the original stage, which stage would you split and what is the new clock cycle time of the processor? You can ignore hazards for answering this question.

a.) Pipelined cycle time = MAX(200, 170, 220, 210, 150) = 220ps

Single-cycle cycle time = 200 + 170 + 220 + 210 + 150 = 950ps

b.) EX - want to split the slowest stage

Cycle time = MAX(200, 170, 110, 110, 210, 150) = 210ps

5.) For the following code, explain what the register file and forwarding unit are doing during the fifth cycle of execution. If any comparisons are being made, mention them. Remember, use the forwarding unit from class, not the book.

Instruction	1	2	3	4	5				
`ADD  X1, X2, X3`	IF	DEC	EX	MEM	WB				
`STUR X2, [X1, #0]`		IF	DEC	EX	MEM	WB			
`LDUR X1, [X2, #4]`			IF	DEC	EX	MEM	WB		
`ADD  X2, X2, X3`				IF	DEC	EX	MEM	WB	
`EOR  X5, X4, X1`					IF	DEC	EX	MEM	WB

The 2nd add is Reading X2 + X3 in decode

The forwarding unit looks at the LDUR. It sets X1, which isn't needed in the add.

The forwarding unit looks at the STUR. It does not write to the regfile.

So, the 2nd add reads X2 and X3 from the regfile, and no forwarding.

The first add is writing its value to X1.