a) What are the values of the control signals for the instruction as listed in the below table?

Instruction	RegDst	Branch	MemRead	MemReg	ALUop	MemWrite	ALUsrc	RegWrite
add $s1, $S3, $S2	1	0	0	0	0010	0	0	1
Addi $S3, $S2, 50	0	0	0	0	10	0	1	1
Lw $t1, 32($s1)	0	0	1	1	00	0	1	1
Sw $s1, 8($s3)	x	0	0	x	00	1	1	0
beq $s0, $s1,120	x	1	0	x	01	0	0	0

1 indicates that it's being used in the instruction
0 indicates that it is not being used in the instruction
X is the dont care term

add $s1, $S3, $S2
The add (s1) instruction goes into the [15-11] field and then gets decoded. S3 and S2 go into [25-21] and [20-16]. Finally they go into the ALU and get added and the result is sent to get outputted in regWrite.

Addi $S3, $S2, 50
The s2 register goes into [25-21] and the decoded and sent into the ALU. 50 is then sent into the ALU after being converted to binary. Finally, the ALU adds value 50 and s2. The Result in the sent back to the register S3 in regWrite

Lw $t1, 32($s1)
s1 goes into [25-21] and then decoded and sent into the ALU. 32 is then sent to the ALU after being converted to binary. The values then go into data memory. Finally after being computed, the result is sent to regWrite and memReg when it is called upon by the user.

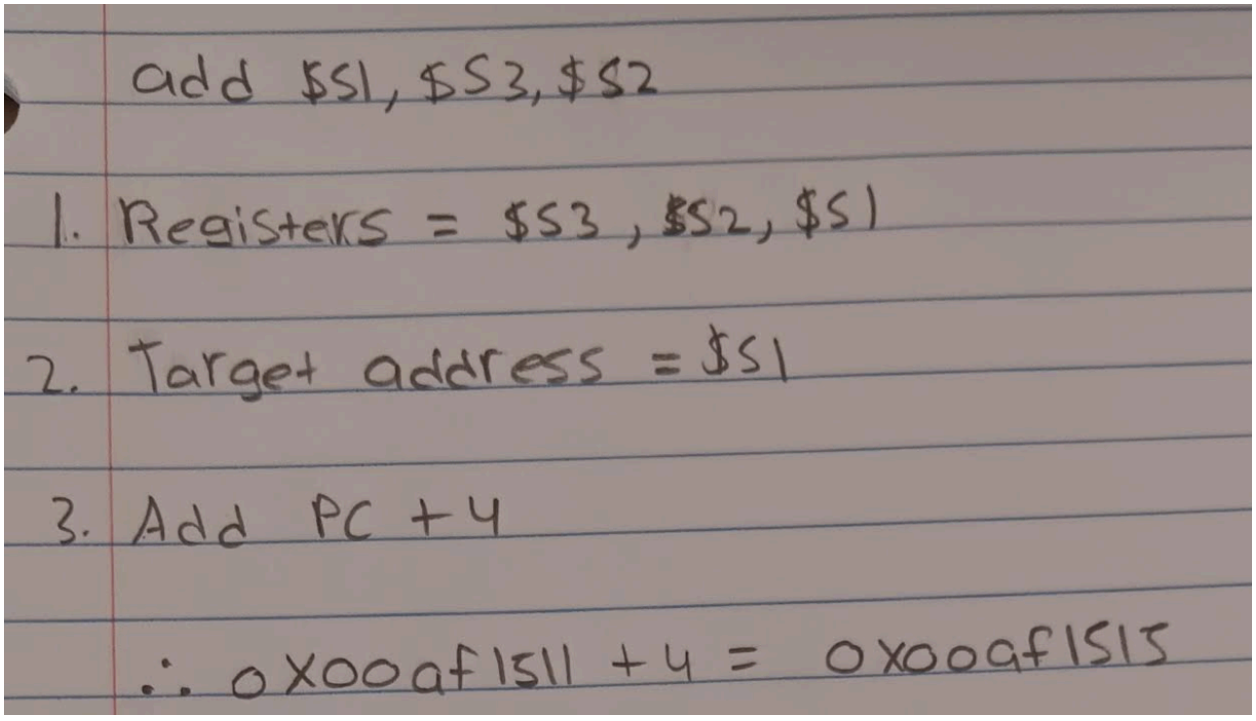
Sw $s1, 8($s3)
s3 goes into [25-21] and then decoded and sent into the ALU. 8 is converted into binary and sent into the ALU. The ALU adds 8 and s3. Finally after being computed the value from s1 is stored into data memory

beq $s0, $s1,120
s0 and s1 goes into [25-21] and [20-16] and then sent to the ALU. The ALU checks if they are equal and sees if the condition for the branch is met. 120 is converted to binary and if the branch condition is met, then it moves on

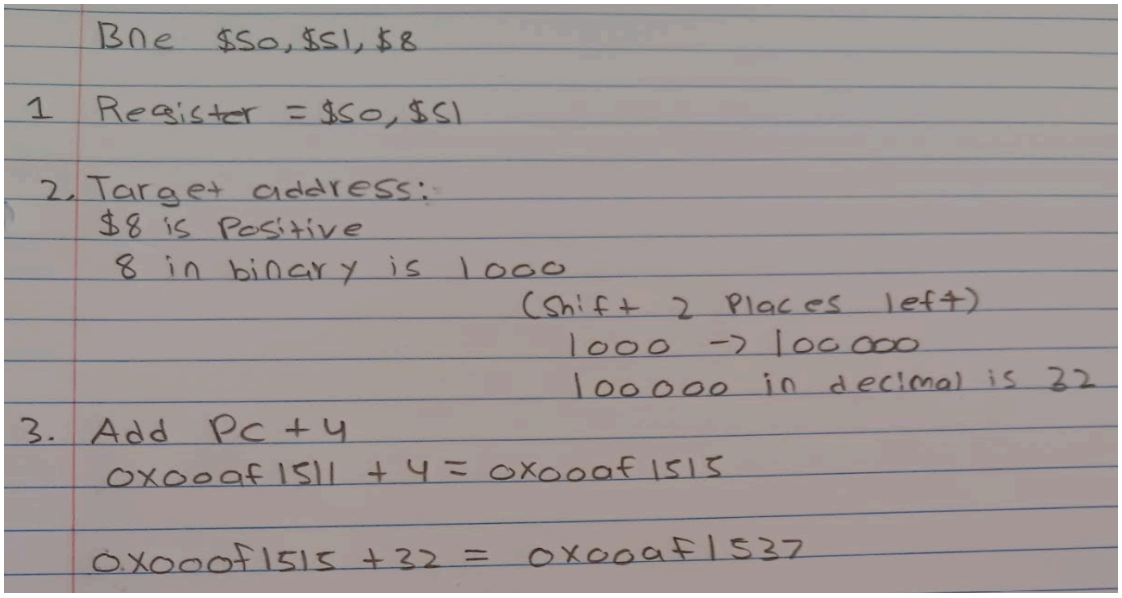
b) Assume that the PC value for all instructions is 0x00af1511. What is the value of the Program Counter (PC) after executing the following instructions? Assume that all conditions are true

Instruction	PC
Add $s1, $S3, $S2	0x00af1515
Bne $s0, $s1, $8	0x00af1537
Lw $t1, 32($s1)	0x00af1515
Sw $s1, 8($s3)	0x00af1515
Beq $s0, $s1, 4	0x00af1531
J 2000	0x00af1515

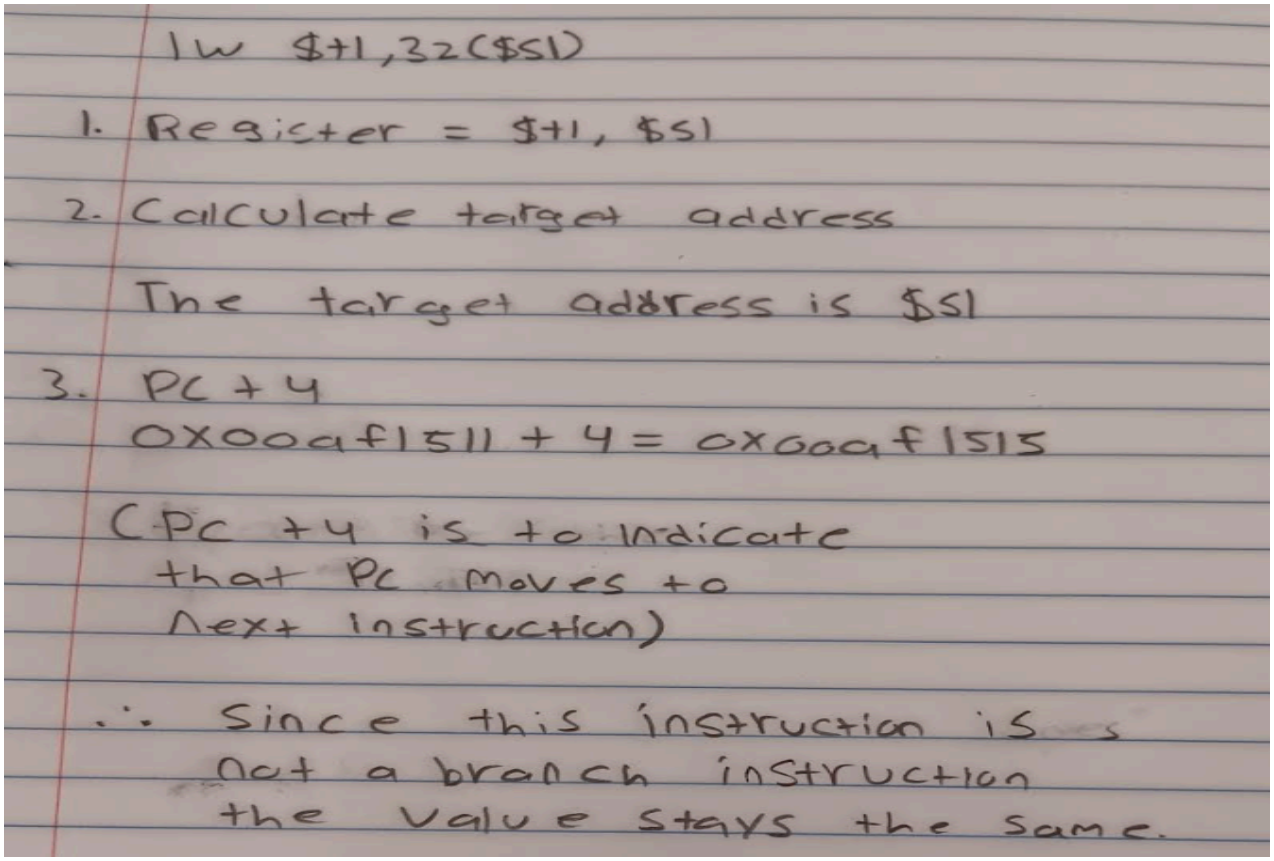
Add $s1, $S3, $S2



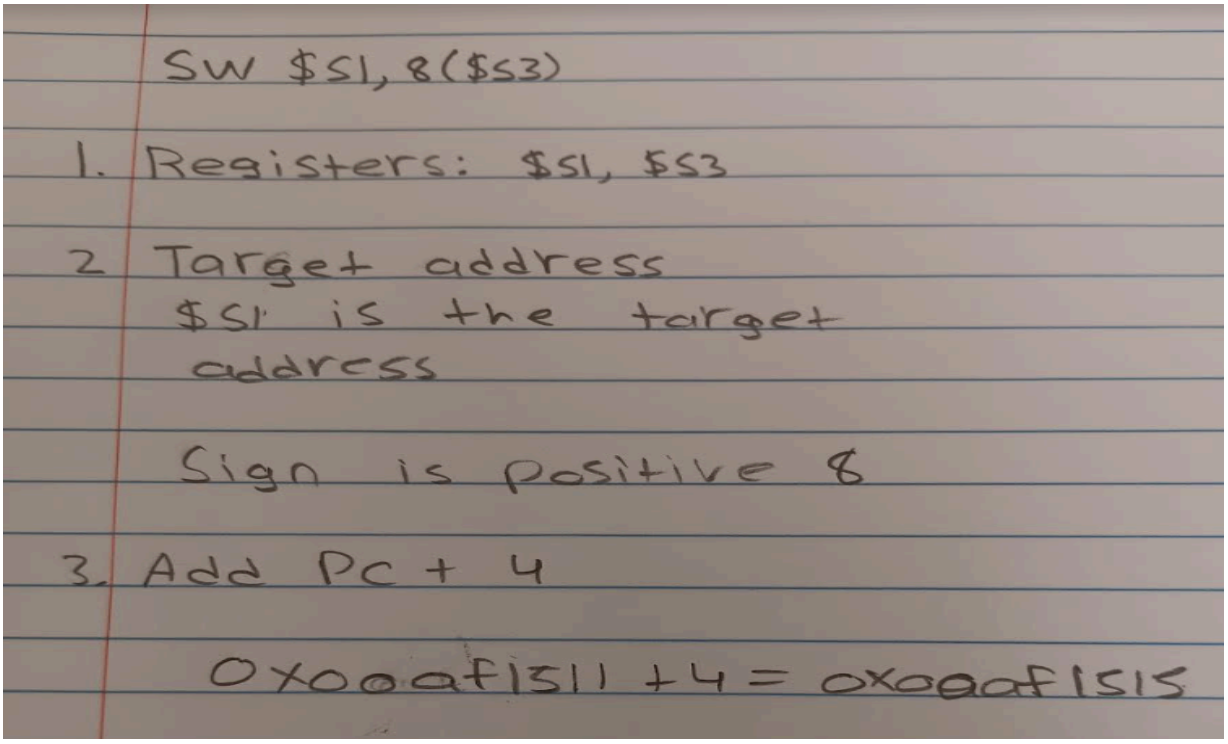
Bne $s0, $s1, $8



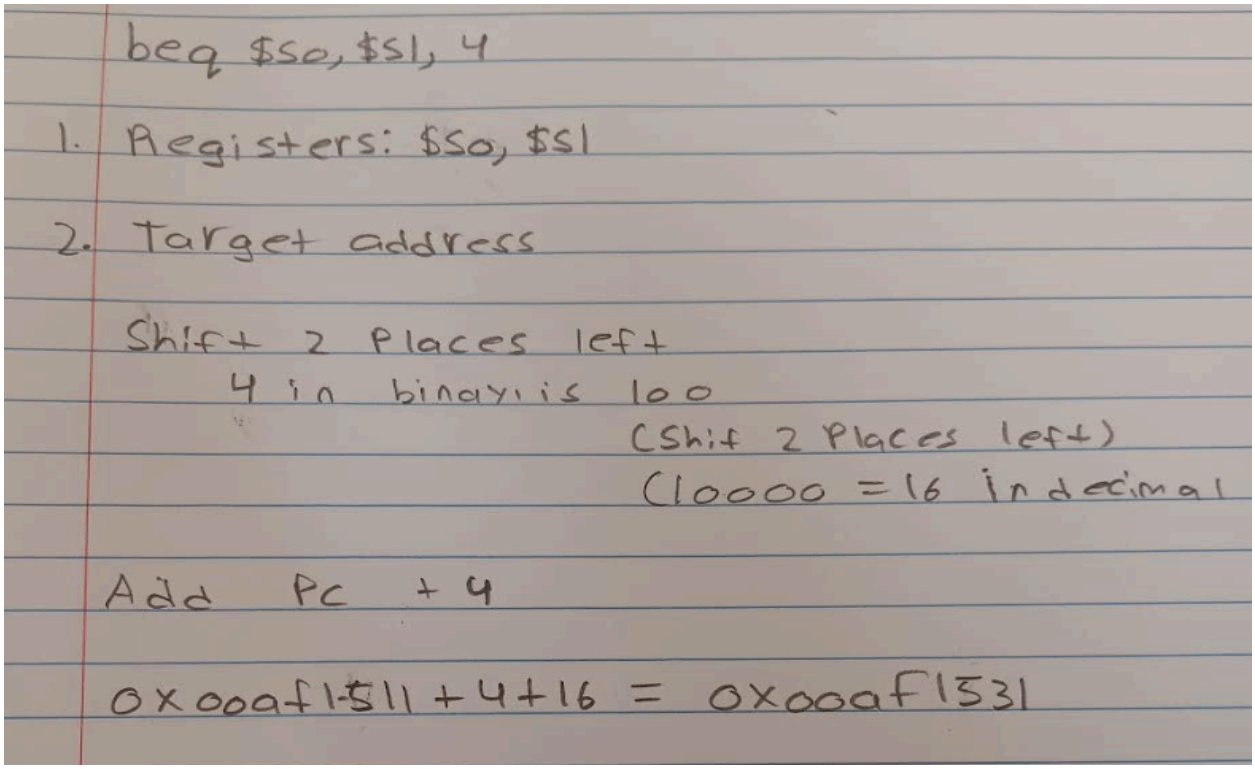
Lw $t1, 32($s1)



sw $s1, 8($s3)



Beq $s0, $s1, 4



j 2000

PC + 4

Therefore: 0x00af1511 + 4 = 0x00af1515

Since the instruction itself is very vague, it jumps to 2000. I concluded that in order for it to jump, you would add 4 since 4 bytes are needed to move in an instruction.

Question 2. Recall that the clock cycle time in a non-pipelined processor is determined by the instruction (considering all stages) that takes the longest time to execute as no instruction can take more than one cycle.

In a pipelined datapath, the cycle time is determined by the individual stage that takes the longest time as each cycle will execute exactly one stage. Now assume the distribution of time required by independent stages are as follows:

IF	ID	EX	MEM	WB
250	350	150	300	200

Now answer the following questions:

a) What should be the clock cycle time for a non-pipelined and a pipelined processor? Provide justification for your answer.

Non-pipelined: Based on the laundry analogy taught in class, a non-pipelined version

involves having to do everything one by one and waiting for each stage to finish. Based on this, I can conclude that it will take 350 for the clock cycle time because that is the longest period of time in the given scenario. However, to complete the entire cycle in the pipeline, it would take 1250 because that is the sum of all the time added up.

Pipelined: In a pipelined processor, it would take 350 as the clock cycle time because that is the stage that takes the longest amount of time. So, as we recall, a pipeline involves everything happening in parallel. So that means even if a stage is faster than the slowest stage, the slowest stage will slow everything down since it needs to be completed in order to move on to the next cycle.

b) How long will it take to execute the lw instruction in a non-pipelined processor? Why?

Based on the laundry analogy taught in class, a non-pipelined version involves having to do everything one by one and waiting for each state to finish. Based on this, I can conclude that it will take all the stages to execute the LW instruction. Thus, 250 + 350 + 150 + 300 + 200 = 1250. Therefore, 1250 is the time it would take to complete the execution in a non-pipelined processor. This is because each stage needs to be completed one at a time and is not parallel at the same time.

c) How long will it take to execute the lw instruction in a pipelined processor? Why?

In a pipelined processor, everything is happening at the same time, meaning it's in parallel. Therefore, for the LW instruction to execute in a pipelined processor, all the stages would need to be completed, and since they are happening at the same time, we would look at the longest stage, which is the slowest. ID takes 350 as the time, which is the slowest, and everything would need to pass through this stage even if they were completed before. Therefore, to calculate how long it would take to execute the LW instruction, it would be based on the ID stage, which is 350. It does not matter how fast the other stages are; if the ID stage slows them down, everything else will also slow down.

Question 3. The following MIPS code might produce data hazard when executed. Study the code and answer the following questions:

lw $t0, 0($a0)
add $t2, $t0, $t1
sw $t2, 4($a0)

a) Explain the nature of the data hazard and identify the instructions involved.

Data hazards occur when an instruction in a pipeline is stalled due to them being dependent on each other to be executed to move on in the clock cycle. Essentially, the pipeline can not move forward due to the hazard. So for the given code, the LW instruction is dependent on the result of the add instruction. What this means is that in the pipeline, the LW instruction needs to wait for the add instructions result to move forward in the pipeline

b) Explain how the data hazard affects the pipeline stages.

How the data hazard affects the pipelines is by causing them to stall because the result of the add is dependent on the result of the LW instruction. Essentially, the add instruction starts the execution phase without the result of the LW instruction causing a hazard.

c) Discuss potential pipeline stalls or bubbles and their impact on performance.

How a pipeline stall can impact performance is by stalling the processor, which causes latency and the overall performance to deteriorate. In the specific case, it's going to cause a control hazard that occurs when the instruction cannot be executed in the proper pipeline clock cycle because the wrong instruction or register was fetched.

d) Discuss possible techniques for resolving the data hazard.

There are multiple ways to resolve the data hazard. One way I am familiar with is known as forwarding or bypassing. How it works is that it retrieves the missing data element from the internal buffers instead of waiting for that element to come from memory.

e) Propose modifications to the code to avoid or minimize the data hazard.

```
lw $t0, 0($a0)

lw $t1, 8($a0)     # Loading $t1 which will avoid the stall because now the add
    instruction has the necessary result, which means that the t1 will be available for the
    add instruction that uses it.


add $t2, $t0, $t1
```

```
sw $t2, 4($a0)
```