For this problem, assume that you have a fully functioning 5-stage pipelined beta with full bypassing and annulment of branch delay slots as presented in lecture.

This beta has been running the program shown here for a while.

The actual functionality of this program is not so important for this problem, but lets just review it quickly.

This program begins by initializing R1 to 0 before entering the loop.

R1 represents the index of the array element currently being accessed.

Within the loop, the value of that array element is loaded into R0.

R1 is then incremented by 4 in order to point to the next element in the array.

We then compare the array element that was just loaded into R0 with the updated index in R1 and if they are equal, then we repeat the loop.

If they are not equal, then we store the current value of R1 into a memory location called index to remember which index value satisfied the compare instruction.

We want to understand how this program would run on our beta.

In order to do this, we will create a pipeline diagram showing the execution of this program.

A pipeline diagram demonstrates which instruction is currently being executed in each of the 5 pipeline stages.

Our rows indicate the pipeline stage that the instruction is in.

There are five pipeline stages.

The first is IF, or instruction fetch, which fetches the next instruction from memory.

The second is RF, or register file stage which reads the source operands of the instruction.

Next comes the ALU stage where all required arithmetic and logic unit operations are executed.

The fourth stage is the MEM stage where we can begin accessing memory for a load or store operation because the address of the memory location was computed in the ALU stage.

Finally, the last stage is WB, or the write back stage where the results are written back into the register file.

The columns in a pipeline diagram represent the execution cycles.

Our loop begins with a LD operation, so we see our LD instruction in the IF stage in cycle 1001.

The LD operation then proceeds down the 5 stages of the pipelined beta.

Next comes the ADDC instruction.

Since there is no dependency between the LD and the ADDC instruction, the ADDC instruction begins in cycle 1002 and proceeds through all the 5 stages of the beta pipeline as well.

Next comes the CMPEQ instruction.

When we reach the CMPEQ instruction, we are met with our first data hazard caused by the fact that the LD is updating R0, and the CMPEQ wants to read this new value of R0.

Recall, that a LD does not produce its value until the WB stage of the pipeline.

This means that even with full bypassing logic, the CMPEQ instruction cannot read register R0 until the LD is in the WB stage.

So we must initiate a stall of the pipeline in cycle 1004.

The stall can be seen in our pipeline diagram in cycle 1005 where the CMPEQ has remained in the RF stage and we have inserted a NOP in place of the CMPEQ that was coming down the pipe one cycle earlier.

The instruction that follows the CMPEQ is the BNE.

Notice that it entered the IF stage in cycle 1004, but it too was stalled by the CMPEQ, so the BNE remains in the IF stage while the CMPEQ is stuck in the RF stage.

In cycle 1005, the CMPEQ is able to complete the read of its operands by using the bypass path from the WB stage to read the updated value of R0, and by using the bypass path from the MEM stage to read the updated value of R1 produced by the ADDC instruction.

In cycle 1006, the CMPEQ instruction moves on to the ALU stage and the BNE can move on to the RF stage.

Since the CMPEQ is going to update the value of R2 which is the register that the BNE is trying to read, we need to make use of the bypass path from the ALU stage to the RF stage in order to provide the BNE with the result of the CMPEQ instruction in cycle 1006.

The RF stage is also the stage when Z is generated.

The Z signal tells the beta whether or not a register is equal to zero.

This means that by the end of the RF stage in cycle 1006, the BNE will know whether it is repeating the loop or not.

We now illustrate what happens to the pipeline diagram if the loop is repeated.

In cycle 1006, the ST instruction enters the IF stage of the pipeline because until we resolve whether a branch is taken or not, we assume that we should continue fetching the next instruction.

If the BNE determines that it should branch back to LOOP, then this ST instruction which was just fetched must be annulled by inserting a NOP in its place.

The annulment is initiated in cycle 1006 and shows up as a NOP in the RF stage in cycle 1007.

In cycle 1007, we also see that we now fetch the first instruction of the loop which is the LD instruction so that we can repeat the loop.

Here is a complete pipeline diagram showing repeated execution of the loop in our sample code together with the bypass paths being used as well as the initiation of stalls and annulment of branch delay slots.

We are now ready to answer a few questions about the execution of this loop on our beta.

The first question we want to consider is which of the registers R0, R1, and/or R2 were read at least once directly from the register file rather than through a bypass path?

Looking back at our completed pipeline diagram, we see that the LD and ADDC instructions did not get their operands through bypass paths.

Since both of those instructions read R1, that means that register R1 was read at least once directly from the register file.

R0 which is only read by the CMPEQ always comes from a bypass path.

Similarly, R2, which is only read by the BNE, always comes from a bypass path as well.

Next, we want to identify the cycle in which stall was set to 1 in the pipelined beta hardware.

This occurs in the cycle where the stall is initiated which was in cycle 1004.

At the end of that cycle the instructions that are currently in the IF and RF stage are stalled by not allowing a load of a new value into the instruction registers of that pipeline stage.

Next, we want to determine in which cycle was ANNUL_IF != 0?

Recall that the ANNUL_STAGE control signals specify when an annulment is initiated in that particular stage.

In order to initiate an annulment, then the instruction that is currently in the IF stage is replaced with a NOP.

This occurs in the IF stage when we need to annul a branch delay slot.

In our example this occurs in cycle 1006.

In which cycle was ANNUL_RF != 0?

This question is asking when an annulment was initiated in the RF stage.

This occurred when the CMPEQ instruction needed to be stalled in the RF stage.

In order to fill the pipeline bubbles, a NOP is inserted into the pipeline in place of the CMPEQ instruction that was in the RF stage in cycle 1004.

The stall and thus the setting of ANNUL_RF != 0 occurs in cycle 1004.

In which cycle was ANNUL_ALU != 0?

In other words, in which cycle did we initiate the replacement of an instruction in the ALU stage with a NOP?

This does not occur in our example.

Next, we want to consider our bypass paths.

In which cycle was either bypass coming from the ALU stage?

In cycle 1006, the BNE reads the result of the CMPEQ instruction from the ALU stage.

In which cycle was either bypass coming from the MEM stage?

In cycle 1005, the CMPEQ reads the result of the ADDC instruction from the MEM stage.

Finally, in which cycle was either bypass coming from the WB stage?

In cycle 1005, the CMPEQ reads the result of the LD instruction from the WB stage.