Q3) Provide detailed description of Pipeline Control Logic in context of four control cases

→ The four control cases to be handled by pipeline control logic are:

i) Load / Use Hazard
→ Pipeline stalls for one instruction cycle between an instruction that reads value from memory & instruction that uses it.

ii) Processing
→ The pipeline must stall until the ret instruction reaches writeback stage

iii) Mispredicted branches
→ By the time branch logic detects jump should not have been taken, several instructions at the branch would have started. These instructions must be cancelled & Fetching should begin at instruction following jump.

iv) Exceptions
→ When istruction causes exception, disable updation of programmer-visible state by later instruction & halt execution once the excepting instruction reaches write back.

⇒ These are the four control cases that need to be handled in Pipe Control Logic.

Desired handling

Load/use hazard

Only movmq & popq instructions read data from memory. When either of these instructions are in execute stage or the instruction requiring destination register is in decode stage, hold back the succeeding instruction in its decode stage & inject a bubble into execute on the next cycle. After that forwarding logic will resolve the hazard.

Implementation requires keeping registers F & D fixed, and injecting bubble into execute stage

Processing

For handling of processing of ret instruction, the pipeline should stall for 3 cycles till the return address is read & ret instruction passes through the memory stage.

Implementation requires control logic to inject bubble in the execute stage for three consecutive cycles.

Misinterpreted branches

If an incorrect branch is selected, it is done when jmp is in ~~decode~~ execute stage. So injecting two bubbles into decode & execute stage would cancel the two incorrectly fetched instructions

Implementation is same as the concept. injecting bubble in decode & execute stage.

Exception

In this, the implementation must match the desired ISA behaviour, with all previous instructions complete & none of the following instructions having any effect.

When exception occurs, record information as part of instructions status & continue fetching, decoding & executing instructions further.

As the excepting instruction reaches memory stage, steps to prevent modification of programmer-visible state by:

i) Disabling the setting of condition codes by instructions in execute stage
ii) Injecting bubbles into memory stage to disable writing to data memory
iii) Stalling write back stage when it has an excepting instruction, thus bringing pipeline to halt.

Combination

Using above given concepts, individual hazards can be stopped, but combinations might not be handled

Exceptions are enclosed in an exception-handling mechanism, so there is no need to handle them in combinations.

The combination of other ~~three~~ is given in graph below.

Condition	F	D	E	M	W
Processing	s	b	n	n	n
load/use hazard	s	s	b	n	n
Combination	s	b+s	b	n	n
Desired	s	s	b	n	n

n → normal
b → bubble
s → stall

Using combinations like this, one can get the desired result.