So here's the final version of our 5-stage pipelined data path.

To deal with data hazards we've added stall logic to the IF and RF input registers.

We've also added bypass muxes on the output of the register file read ports so we can route values from later in the data path if we need to access a register value that's been computed but not yet written to the register file.

We also made a provision to insert NOPs into the pipeline after the RF stage if the IF and RF stages are stalled.

To deal with control hazards, we speculate that the next instruction is at PC+4.

But for JMPs and taken branches, that guess is wrong so we added a provision for annulling the instruction in the IF stage.

To deal with exceptions and interrupts we added instruction muxes in all but the final pipeline stage.

An instruction that causes an exception is replaced by our magic BNE instruction to capture its PC+4 value.

And instructions in earlier stages are annulled.

All this extra circuitry has been added to ensure that pipelined execution gives the same result as unpipelined execution.

The use of bypassing and branch prediction ensures that data and control hazards have only a small negative impact on the effective CPI.

This means that the much shorter clock period translates to a large increase in instruction throughput.

It's worth remembering the strategies we used to deal with hazards: stalling, bypassing and speculation.

Most execution issues can be dealt with using one of these strategies, so keep these in mind if you ever need to design a high-performance pipelined system.

This completes our discussion of pipelining.

We'll explore other avenues to higher processor performance in the last lecture, which discusses parallel processing.