EECS 151 FPGA CPU: Checkpoint 1

Niko Dittmar, Justin Hoang

October 2025

1 Questions

1.1 How many stages is the datapath you've drawn?

How many cycles does it take to execute 1 instruction?

It it five stages, each instruction will take 5 cycles to execute, but on average, we will execute one instruction per cycle as they are being run in parallel.

1.2 How do you handle ALU → ALU hazards?

```
addi x1, x2, 100
addi x2, x1, 100
```

We can solve this hazard by forwarding the ALU result to the ALU input. We will do this anytime the instruction in the EX stage uses a register that was written to in the M stage. So whenever `rd == rs1` (A input mux) or `rd == rs2` (B input mux).

1.3 How do you handle ALU → MEM hazards?

```
addi x1, x2, 100
sw x1, 0(x3)
```

To handle ALU to memory data hazards, we will forward the data from the WB stage to the M stage. This will happen when the M stage uses the register that was written to in the WB stage. So whenever `rd == rs2`

1.4 How do you handle MEM → ALU hazards?

```
lw x1, 0(x3)
addi x1, x1, 100
```

To handle memory to ALU data hazards, we will need to stall for one cycle. Then, we will forward the data from the WB stage to the EX stage. We will do this whenever `rd == rs1` or `rd == rs2` and the instruction in the M stage is a load type instruction.

1.5 How do you handle MEM → MEM hazards?

```
lw x1, 0(x2)
sw x1, 4(x2)
```

To handle memory to memory data hazards for the `rs2` register, we will forward the result from the WB stage to the M stage. This will happen whenever `rs2 == rd`.

Also consider:

```
lw x1, 0(x2)
sw x3, 0(x1)
```

To handle memory to memory data hazards for the `rs1` register, we will need to stall for one cycle and then we can forward the result from WB to the EX stage so we can correctly calculate the address.

1.6 Do you need special handling for 2 cycle apart hazards?

```
addi x1, x2, 100
nop
addi x1, x1, 100
```

To handle data hazards that are two cycles apart, we will forward the data from the WB stage to the EX stage. This will happen when the EX stage uses a register that was written to in the WB stage. So whenever `rd == rs1` (A input mux) or `rd == rs2` (B input mux).

Importantly, the M stage forwarding will take precedence over the WB stage forwarding, in the event that we have several instructions all operating on the same instructions in a row.

1.7 How do you handle branch control hazards?

What is the mispredict latency, what prediction scheme are you using, are you just injecting NOPs until the branch is resolved, what about data hazards in the branch?

For branches, we will predict if they are taken or not in IF so that we can load the requisite instructions in the following cycle. If the prediction is determined to be incorrect in the EX stage we will flush the instructions in the IF and ID stage. The EX will be connected to the PC input mux so we can forward the corrected instruction to be fetched in the next clock cycle.

1.8 How do you handle jump control hazards?

Consider `jal` and `jalr` separately. What optimizations can be made to special-case handle `jal`?

We will calculate the `jal` and `jalr` addresses in the ID stage. This will allow us to immediately fetch the next instrucition with no bubbles (as our IMEM and BIOS memory is synchronous). To handle data hazards for `jalr`, we add forwarding from the three following pipeline stages with earlier stages holding priority. We then allow the rest of the instruction to continue on through the pipeline so that we can write PC + 4 to to the RegFile.

1.9 What is the most likely critical path in your design?

The ciritical path of out design is likely the EX stage with the ALU. Most of the memory is stored on the on-chip Block RAMs which is relatively fast. On the other hand, certain operations such as adding can be expensive on an FPGA.

1.10 Where do the UART modules, instruction, and cycle counters go?

How are you going to drive `uart_tx_data_in_valid` and `uart_rx_data_out_ready` (give logic expressions)?

Because we are using memory mapped I/O, the UART module can be thought of as simply another memory module, the control logic will determine if the UART module should be written to or read based on the address within the provided instruction. The instruction and cycle counters will be within this module as a register. The UART module will act as a "fire and forget" allowing it to run in one cycle, then continue on transmitting the message in continuing cycles, if we try to read from it before it is ready, we will stall the entire pipeline. When it is sending out messages, it will manage the ready valid interface.

1.11 What is the role of the CSR register? Where does it go?

The role of the CSR register is to determine if the tests for the RISC-V ISA suite passed. We check the value at the end of the test, a 1 indicates a pass, a value greater than 1 indicates failure. We can put it in the EX stage as we will only try to write values from the immediate generator and the register file.

1.12 When do we read from BIOS for instructions?

When do we read from IMem for instructions? How do we switch from BIOS address space to IMem address space? In which case can we write to IMem, and why do we need to write to IMem? How do we know if a memory instruction is intended for DMem or any IO device?

We read from BIOS right after reset, since the reset sets the PC to the base of the BIOS Memory 0x40000000. On the other hand, we fetch from IMEM when the MSB of the PC address is 0x1. We can switch from BIOS address space to IMEM address space using a jump instruction. We can write to the IMEM when our PC is in BIOS memory, we need this capability to load user programs (from the UART), before jumping to them.