1. Describe how you design the data path of instructions not referred in the lecture slides (jal, jalr, auipc, ...)
    a. Program counter (PC):

        The PC holds the address of the current instruction. At each clock cycle (except when stalled due to a multi-cycle instruction), the PC is updated to either:

        - PC + 4 for normal sequential execution
        - A branch/jump target address if the current instruction is a branch or jump
        - A jal/jalr target address for jump and link instructions

    b. Immediate Generator:

        The immediate generator extracts and sign-extends the immediate field from the instruction. Different instruction formats (I, S, B, U, J) have distinct rules for immediate extraction and sign-extension. For jump and branch instructions, the immediate is adjusted (shifted) as per RISC-V specification.

    c. ALU and ALU Control:

        The ALU performs arithmetic and logical operations based on ALU_control. For most instructions (add, sub, logical shifts, comparisons, lui, auipc), the ALU operates in one cycle. The ALU can:

        - Add/subtract for address calculations, arithmetic instructions
        - Shift operations (sll, srl, sra)
        - Set less than comparisons (slti)
        - Handle special instructions like lui (directly write upper immediate) and auipc (add PC and immediate shifted appropriately).

    d. Branch and Jump Logic:

        Branch instructions (beq, bne, blt, bge) use the ALU to compute a comparison result (often via subtract). If the condition is met, the PC jumps to PC + imm. Jumps (jal) and conditional branches share the logic that uses the branch_jal_target = PC + imm. Jalr uses (rs1_data + imm) & ~1

    e. Control Unit:

        The Control Unit decodes the opcode, funct3, and funct7 fields to determine the type of operation. It sets signals such as ALUSrc (whether second ALU operand is immediate or register), MemRead, MemWrite, MemtoReg, Branch, Jump, Jalr, and the ALUOp for the ALU.

        For instructions not covered in the lecture slides:

        - jal (J-type): Jump is asserted, ALUregWrite is asserted (to write PC+4 into rd).

- jalr (I-type jump): Jalr is asserted; it also uses ALUSrc since it needs imm.
- auipc (U-type): Treat like adding PC and a shifted immediate (ALU control 0111). Set ALUregWrite and ALUSrc to form the correct operand and pass the result back into rd.

2. Describe how you handle multi cycle instructions (mul, divu, remu)

   The MulDiv unit is a separate hardware block that handles multiplication and division/remainder operations, which do not complete in a single cycle. This unit works as follows:

   a. Initiation (valid signal):

      When a multiply or divide instruction is encountered (mul, divu, remu), the CPU sets mulDiv_valid. This signal initiates the multi-cycle operation. The Control Unit sets mulDiv_valid high and prevents the CPU's PC from advancing by stalling if necessary.

   b. Multi-Cycle Execution:

      The MulDiv module's state machine runs for multiple cycles (up to 32 cycles) to complete the multiplication or division:

      - MUL: Implements iterative add-and-shift steps to compute the product.
      - DIVU/REMU: Performs iterative subtract-and-shift to find the quotient and remainder.

      During these cycles, the main pipeline is effectively stalled. The PC and instruction fetch do not move forward. Only after MulDiv signals completion (mulDiv_ready) can the CPU continue.

   c. Completion and Write Back:

      Once the MulDiv unit finishes, it asserts ready. This makes mulDiv_ready high, which in turn allows the CPU to take the MulDiv_result and write it back into the destination register in the same cycle. The next cycle, the PC is allowed to advance to the next instruction, ending the stall period.

3. Snapshot the Register table using Design Compiler

```
dc_shell> read_verilog CPU.v
Loading db file '/usr/cad/synopsys/synthesis/cur/libraries/syn/gtech.db'
Loading db file '/usr/cad/synopsys/synthesis/cur/libraries/syn/standard.sldb'
  Loading link library 'gtech'
Loading verilog file '/home/raid7_2/userb10/b10172/CA/final_project/CPU.v'
Detecting input file type automatically (-rtl or -netlist).
Reading with Presto HDL Compiler (equivalent to -rtl option).
Running PRESTO HDLC
Warning: Can't read link_library file 'your_library.db'. (UID-3)
Compiling source file /home/raid7_2/userb10/b10172/CA/final_project/CPU.v

Inferred memory devices in process
        in routine CPU line 190 in file
                '/home/raid7_2/userb10/b10172/CA/final_project/CPU.v'.
===============================================================================
|    Register Name    |   Type    | Width | Bus | MB | AR | AS | SR | SS | ST |
===============================================================================
|       PC_reg        | Flip-flop |  31   |  Y  | N  | Y  | N  | N  | N  | N  |
|       PC_reg        | Flip-flop |   1   |  N  | N  | N  | Y  | N  | N  | N  |
===============================================================================
Warning:  /home/raid7_2/userb10/b10172/CA/final_project/CPU.v:225: signed to unsigned conversion occurs. (VER-318)
Warning:  /home/raid7_2/userb10/b10172/CA/final_project/CPU.v:232: signed to unsigned conversion occurs. (VER-318)

Inferred memory devices in process
        in routine reg_file line 228 in file
                '/home/raid7_2/userb10/b10172/CA/final_project/CPU.v'.
===============================================================================
|    Register Name    |   Type    | Width | Bus | MB | AR | AS | SR | SS | ST |
===============================================================================
|       mem_reg       | Flip-flop |  995  |  Y  | N  | Y  | N  | N  | N  | N  |
|       mem_reg       | Flip-flop |  29   |  Y  | N  | N  | Y  | N  | N  | N  |
===============================================================================
Statistics for MUX_OPs
=====================================================
| block name/line  | Inputs | Outputs | # sel inputs |
=====================================================
|   reg_file/220   |   32   |   32    |      5       |
|   reg_file/221   |   32   |   32    |      5       |
=====================================================
```

```
Statistics for case statements in always block at line 279 in file
        '/home/raid7_2/userb10/b10172/CA/final_project/CPU.v'
============================================
|           Line           |  full/ parallel  |
============================================
|           292            |    auto/auto     |
|           298            |    auto/auto     |
|           306            |    auto/auto     |
|           324            |    auto/auto     |
============================================

Statistics for case statements in always block at line 390 in file
        '/home/raid7_2/userb10/b10172/CA/final_project/CPU.v'
============================================
|           Line           |  full/ parallel  |
============================================
|           391            |    auto/auto     |
============================================
Warning:  /home/raid7_2/userb10/b10172/CA/final_project/CPU.v:443: signed to unsigned assignment occurs. (VER-318)

Statistics for case statements in always block at line 437 in file
        '/home/raid7_2/userb10/b10172/CA/final_project/CPU.v'
============================================
|           Line           |  full/ parallel  |
============================================
|           438            |    auto/auto     |
============================================

Statistics for case statements in always block at line 478 in file
        '/home/raid7_2/userb10/b10172/CA/final_project/CPU.v'
============================================
|           Line           |  full/ parallel  |
============================================
|           480            |    auto/auto     |
|           483            |    auto/auto     |
============================================
```

```
Statistics for case statements in always block at line 542 in file
        '/home/raid7_2/userb10/b10172/CA/final_project/CPU.v'
===============================================
|           Line           |  full/ parallel  |
===============================================
|           549            |     no/auto      |
|           552            |     no/auto      |
===============================================

Inferred memory devices in process
        in routine MulDiv line 509 in file
                '/home/raid7_2/userb10/b10172/CA/final_project/CPU.v'.
===========================================================================================
|    Register Name    |   Type    | Width | Bus | MB | AR | AS | SR | SS | ST |
===========================================================================================
|      state_reg      | Flip-flop |   2   |  Y  | N  |  Y  | N  | N  | N  | N  |
===========================================================================================

Inferred memory devices in process
        in routine MulDiv line 517 in file
                '/home/raid7_2/userb10/b10172/CA/final_project/CPU.v'.
===========================================================================================
|    Register Name    |   Type    | Width | Bus | MB | AR | AS | SR | SS | ST |
===========================================================================================
|  calc_counter_reg   | Flip-flop |   6   |  Y  | N  |  Y  | N  | N  | N  | N  |
===========================================================================================

Inferred memory devices in process
        in routine MulDiv line 542 in file
                '/home/raid7_2/userb10/b10172/CA/final_project/CPU.v'.
===========================================================================================
|    Register Name    |   Type    | Width | Bus | MB | AR | AS | SR | SS | ST |
===========================================================================================
|  multiplicand_reg   | Flip-flop |  32   |  Y  | N  |  Y  | N  | N  | N  | N  |
|     divisor_reg     | Flip-flop |  64   |  Y  | N  |  Y  | N  | N  | N  | N  |
|     product_reg     | Flip-flop |  65   |  Y  | N  |  Y  | N  | N  | N  | N  |
|    remainder_reg    | Flip-flop |  65   |  Y  | N  |  Y  | N  | N  | N  | N  |
===========================================================================================
Presto compilation completed successfully.
Current design is now '/home/raid7_2/userb10/b10172/CA/final_project/CPU.db:CPU'
Loaded 6 designs.
Current design is 'CPU'.
CPU reg_file ControlUnit ImmediateGenerator ALU MulDiv
```

4. Describe your observation

The interaction between the MulDiv module and the Control Unit can impact the system's scalability.

a. Complexity of Control Logic:

When the CPU only supports basic ALU operations, branches, and jumps, the Control Unit's logic remains relatively simple, primarily determining control signals within a single cycle based on the opcode, funct3, and funct7. However, with the addition of multi-cycle instructions such as mul, divu, and remu, the Control Unit must account for the entry and exit timing of these instructions and generate handshake signals like mulDiv_valid and mulDiv_ready.

As more complex, multi-cycle instructions are added (for example, other multi-cycle operations, specialized accelerators, or floating-point units), the Control Unit needs to coordinate an increasing number of multi-cycle modules, potentially leading to an overly large state machine that becomes difficult to maintain and extend.

b. Expansion Issues with Stall Mechanism:

Currently, the stall mechanism is relatively simple: when a MulDiv instruction is encountered, the PC is not updated until mulDiv_ready is high. If in the future there is a need to handle multiple different types of multi-cycle operations simultaneously (such as floating-point operations or other specialized instructions), each operation might have different completion times. If multiple stall conditions need to be considered concurrently, the Control Unit may need to maintain a more complex arbitration mechanism to ensure the correctness and timing of the instruction flow, further increasing the burden on the control logic.