RISC-V 4-Stage Pipeline Core (`riscv_core`)

This module is the **heart of your CPU**. It orchestrates how instructions flow through each pipeline stage, how data is moved/calculated, and controls overall sequencing.

High-Level Structure

Your pipeline has these four stages:

1. **Instruction Fetch (IF)**
2. **Instruction Decode (ID)**
3. **Execute (EXE)**
4. **Write Back (WB)**

Between each stage, a **pipeline register** (e.g., `IF_IDstage`, `ID_EXEstage`, `EXE_WBstage`) holds the necessary data to maintain smooth flow and correct timing.

Stage-by-Stage Flow

IF (Instruction Fetch) Stage

- **PC (Program Counter):**
  - Holds address of the *current* instruction.
  - Updated every cycle: usually `PC+4` (sequential), or to a branch/jump target address if needed.

- **Memory:**
  - Uses PC to fetch instruction from *instruction memory* (ROM loaded from `program.hex`).

- **IF/ID Pipeline Register (`IF_IDstage`):**
  - Captures PC and fetches instruction.
  - These values are passed cleanly to the next stage at each clock cycle.

ID (Instruction Decode) Stage

- **Extracting Fields:**
  - Splits the instruction into **rs1**, **rs2**, **rd** fields (source/dest register numbers).
  - Decodes the **immediate** field as required (I/S/B/J-type instructions, depending on opcode).

- **Register File (`regfile`):**
  - Reads the values from the register file for `rs1` and `rs2`.
  - Eventually (in the WB stage) receives a write-back of computed results.

- **Control Unit (`control`):**
  - Decodes the instruction and outputs control signals that direct how each part of the CPU should behave (ALU operation, register write enable, branching, memory operations, etc.).

- **ID/EXE Pipeline Register (`ID_EXEstage`):**
  - Captures all data, operands, and control signals above, forwarding them to EXE.

3. EXE (Execute) Stage

- **Operand Selection:**
  - Selects between a register value and an immediate value (as the second input) for the ALU, as directed by the control signal.

- **ALU:**
  - Computes the result (add, subtract, AND, OR, etc.) according to the decoded `aluop` signal.
  - Computes the result and sets the `zero` flag for branch decisions.

- **Branch Decision:**
  - If a branch instruction is detected and the ALU result (`zero`) is true, the next PC is set to the branch target; otherwise, it increments by 4 for sequential execution.

- **EXE/WB Pipeline Register (`EXE_WBstage`):**
  - Stores the ALU result, destination register, and control signal for write-back, forwarding them to the WB stage.

4. WB (Write Back) Stage

- **Write Data:**
  - Takes the result from the EXE stage (already stored in `EXE_WBstage`).
  - If the instruction requires it (`regwrite` signal), write the result into the destination register in the register file.

Pipeline Registers—Why They're Crucial

Each **pipeline register** (e.g., `IF_IDstage`, `ID_EXEstage`, `EXE_WBstage`) "freezes" the crucial outputs of one stage, handing them at the proper time to the next stage on the next clock cycle. This allows instructions to overlap without interfering—while one instruction is being fetched, another is being decoded, another is executing, and another is writing back.

Data and Control Flow Summary

- The **PC** determines which instruction to execute.
- The **fetched instruction and PC** are handed off in the IF/ID pipeline register.
- The **instruction is decoded**, operands fetched, control signals generated in ID, and latched for EXE.
- The **ALU computes** results, and branch logic checks if the PC needs to jump via the branch signal and zero flag.
- The **result, destination, and control signals** move to WB.
- The **register file** is written back with results (as required).

Special Notes

- **Branching:** The `assign pc_next = (branch_exe && zero) ? pc_exe + imm_exe : pc + 4;` line means:
  - If the instruction in EXE is a branch and the branch condition is true (e.g., `beq` and comparison was zero/equal), the PC is set to the branch target. Otherwise, the PC just increments to the next instruction.

Visual Summary

text
```
[PC] --fetch--> [memory] --|--> [IF/ID] --decode--> [regfile]
[control] [imm decoder] --|--> [ID/EXE] --execute--> [ALU] [branch]
--|--> [EXE/WB] --write-back--> [regfile]
```
Each "--|-->" is a pipeline register separating the major stages.