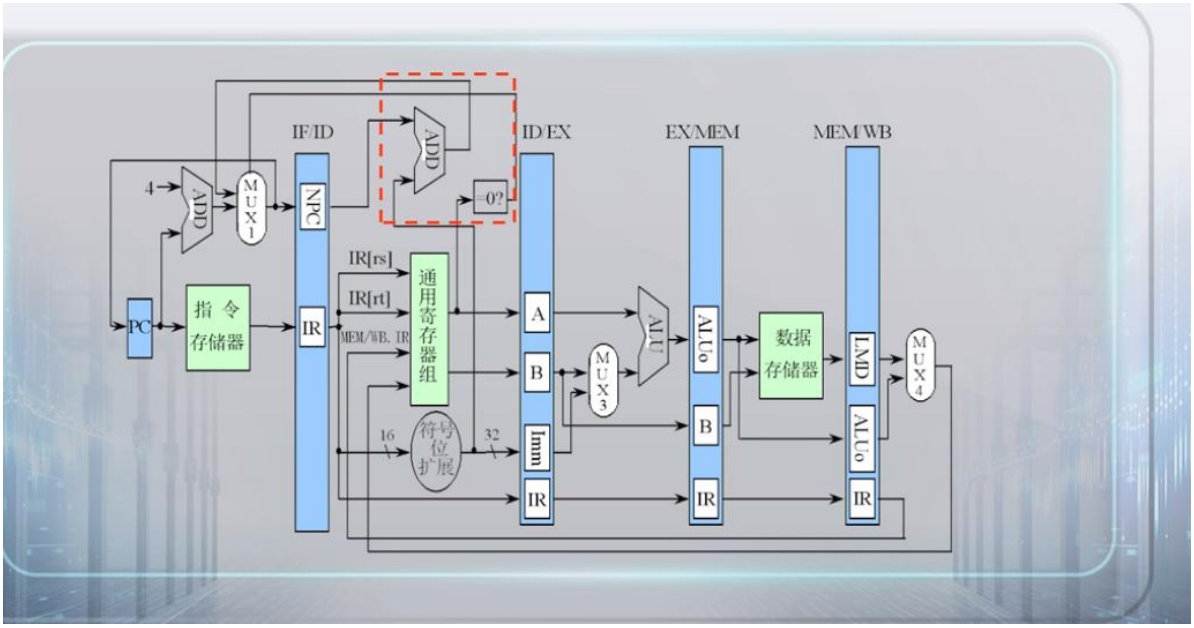
How to Start

**You can edit the line 53 in `src\testBench.v` to ensure show the clock counts after all instructions are end.*

```
make run
// clean previous build
make clear
```

Basic Idea

- Use registers between different stage modules to store key information, such as opcode, register data, address and so on.
- Connect the register after EX stage and MEM stage back to the ID stage to solve R type hazard. That is, directly push the data back to the next R type instruction.
- Use stall generation to delay the following instruction to handle the hazard of `lw`.
- In `ID` stage, try to handle `beq` and `j` (and so on), which we can boost the clock consumption with only one clock loss.



**The figure shows the abstract architect of the CPU design, but not my exact design.*

Details

PC

- If it receives the `branchEnable`, it means it needs to make a jump to `branchAddress`. If not, it makes increment of 4 following the clock.

IF_ID

- If it receive the `stall` signal to pause the dataflow, it gives zero command to the next stage, meaning do nothing. If not, it conveys key data from IF to ID.

ID

1. Use the opcode and function to decide what to do.
2. Fetch the register data according to address
    - If no hazard encounters, fetch from general registers.
    - If R type hazard, fetch from the register after EX or MEM stage.
    - If `sw` hazard, fetch from MEM stage.
    - If `lw` hazard, push some stall (`stall_gen.v`).
3. (*Only branch jump relatives*) If need to jump, set the following stage do nothing, and set the PC to the right place.

ID_EX

- Similar to **IF_ID**.

EX

- Do calculation or logical operation according to the opcode
- If the data needs to be written to / fetched from RAM, put them into `memoryAddress` and `memoryData`. If not, put it to result.
- Set the `writeEnable` and `writeAddress` correctly to push them back to register in WB stage.

EX_MEM

- Similar to **IF_ID**.

MEM

- Use the `MainMemory` API according to the opcode.

MEM_WB

- Similar to **IF_ID**.

WB

- The `Registers` in file `ID.v` flushes following the clock, putting the data in WB stage into correct place.

Miscellaneous

- `CPU.v` encloses the core modules of the CPU. And transfer the data between these modules.
- `driver.v` encloses the CPU and RAM. And transfer the data between them.
- `testBench.v` is used for testbench.
- You could also see my GitHub for the source code.

Reference

CSDN