ECE505 Final RISC-V Processor - Instruction

- Editor: @Xiao Zhang
- Date: Nov 4, 2022

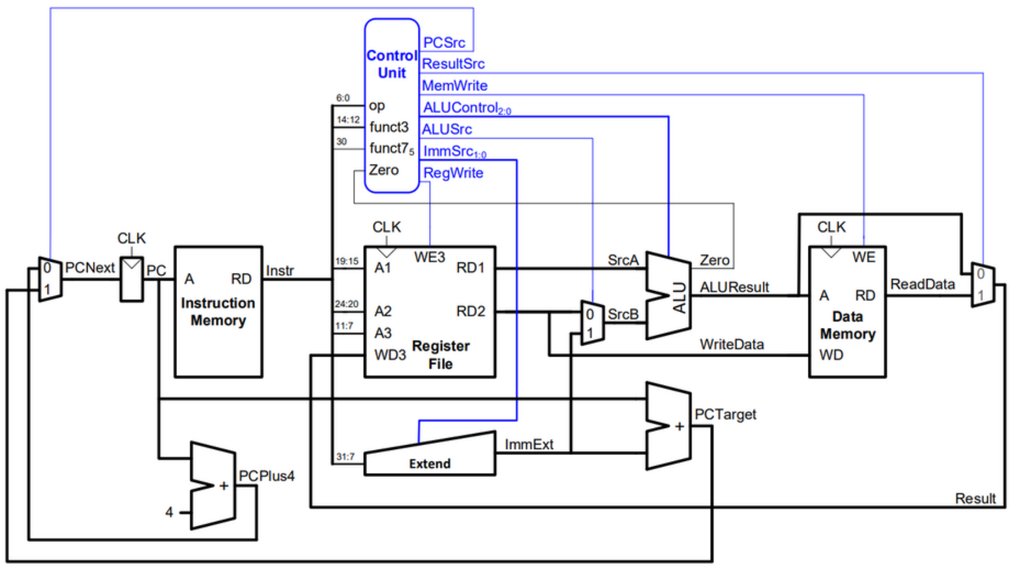
0. Pre-request

- HDL Coding Skills (Verilog/System Verilog)
- RISC-V Instruction Set Architecture (Chapter 6 of Book)
- RISC-V Micro-Architecture (Chapter 7 of Book)
- Xilinx Vivado IDE
  You could also use the *Intel Questa*, However, Vivado is recommended, since:
  > the supportive instructions are in the Vivado version
  > demo uses Xilinx IP core: single-port RAM and MMCM.
  (You could also learn how to use Intel's IP: RAM and PLL)
  > Xilinx Vivado has been pre-installed in PCs in **AK317**

1. Overview

The project aims to design a single-cycle **RSIC-V processor** with several general instructions by Verilog. The HDL design requires a logical level of verification in simulation and should pass at least **20Mhz** time constraint in the synthesis and implementation phases. (**Does not** require an actual FPGA board implementation). The key design idea is how to implement 5 stages process (F D E M W) in one clock cycle.

The overall architecture is shown as follows:



Tips:

1. **Machine codes:** Get from https://venus.kvakil.me/. You are also suggested to run the simulation to understand the details of RAM and registers.
2. **IP Cores**: It is required to instance the single-port RAM and MMCM (clockWizard) IP in the project. The instruction about how to generate IP cores can refer to "**Instruction - IP Core.pdf**" in supportive documents.

3. **Memory**: It is suggested to use a **register ROM** design to realize the Instruction Memory, a real single-port ROM also works. Size recommend:
   > ROM - 32 depth 32-bit width
   > RAM - 256 depth 32-bit width
4. **Clock**: The processor needs at least 2 - 3 (recommended) clocks in a different phase to realize the design.
   > CLK above is actually the same frequency clock with different phase shift
   > All the clocks should be the output of the MMCM / PLL
   > “*locked*“ signal should be considered when you instance the MMCM
5. **Module and Signal**: The other function modules and signals are self-design in Verilog. It is recommended to have the same module and signal name and set as in the book but not required.

2. Supportive Documents

```
├── README.pdf
├── Instruction - IP Core.pdf
├── Instruction - Steps.pdf
├── constrs
│   └── time.xdc
├── demoModule
└── tb
    ├── IP
    │   ├── tb_clkWizard.v
    │   └── tb_ram.v
    ├── screenshot
    ├── tb_ALU.v
    ├── tb_alu_control.v
    ├── tb_control_unit.v
    ├── tb_imm_gen.v
    ├── tb_reg_file.v
    ├── tb_reg_rom.v
    └── tb_top.v
```

Tips:

1. You could follow the `Instruction - IP Core.pdf` to learn instancing RAM and MMCM in Vivado
2. You could refer to the `Instruction - Steps.pdf` to build the processor step by step
3. The `demoModule` includes the demo code for submodules.
   > These codes are provided to help students who are not familiar with Verilog. If you have taken ***ECE2029,*** and ***ECE3829*** before, please feel free to design on your own.
   > The demo codes' port definition and structure are slightly different from the book and instruction. You do not need to follow the specific port or code structure design, feel free to realize the design in your own way
4. `tb` including the test bench to verify your submodules and top level.
   > It is suggested to verify every module correctly before you integrate them on the top level (*makes your debugging life much easier*).
   > `screenshot` includes the “correct” simulation result to help your design verification.
5. `constrs` contains the XDC file to define time (frequency) constraints for implementation. You only need it during the synthesis and implementation phases。
   > `period` in `time.xdc` defines the clock cycle period in a nanosecond.

> Start early and Good luck!