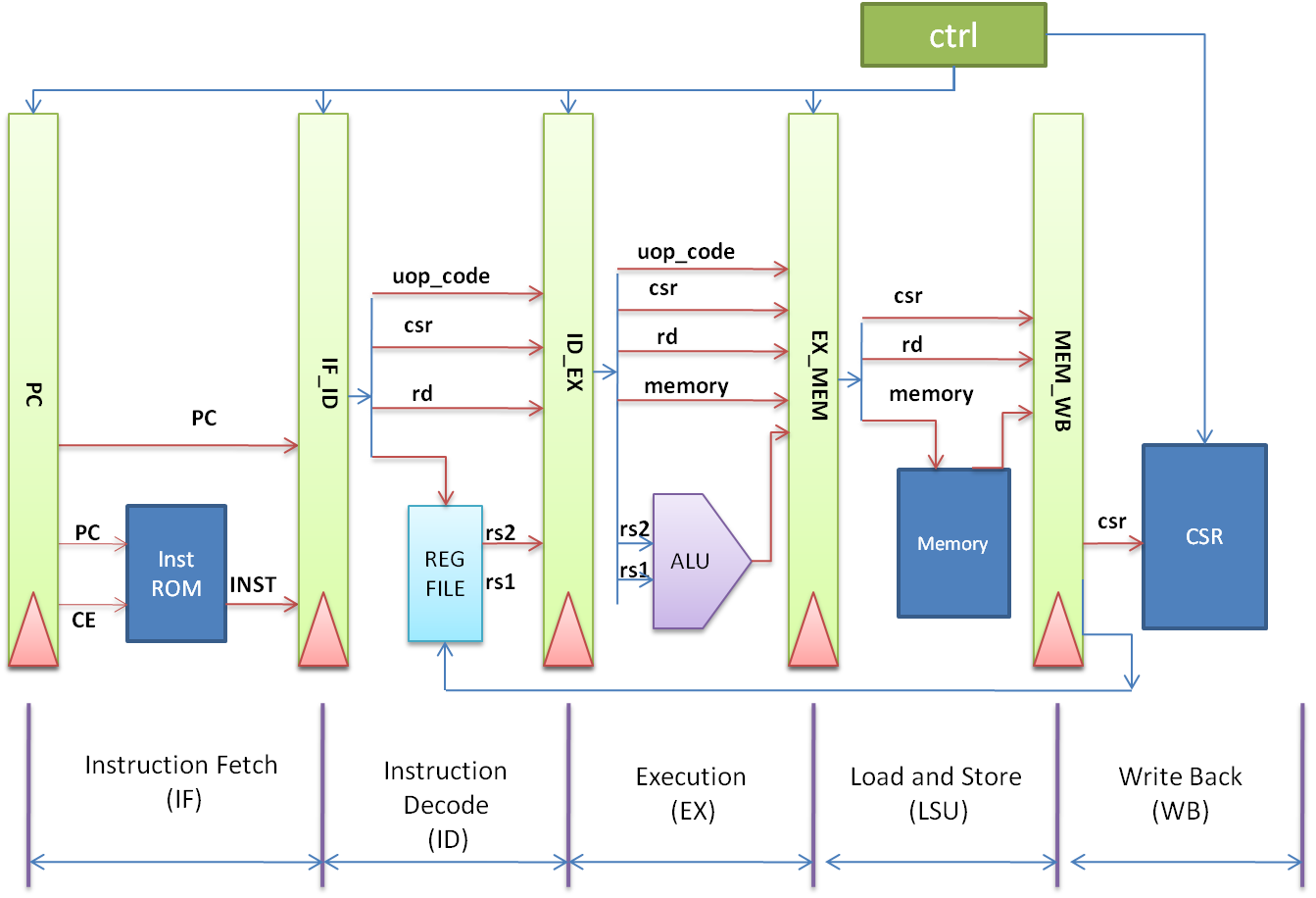
This CPU is a training-target implementation of rv32im, designed to be simple and easy to understand. However, most of the technologies of RISC-V CPU design are still addressed in this practice (named: Kookaburra), including the classic pipeline structure, dealing with data/structure/control hazards, precise exception and interrupt, pipeline stalling and flushing etc. The self-explained Verilog code accompanied with ample comments makes it is easy to expand a variety of new features, for instance, adding i-cache, data-cache units, supporting machine and user privileges as well as applying the branch prediction and etc.

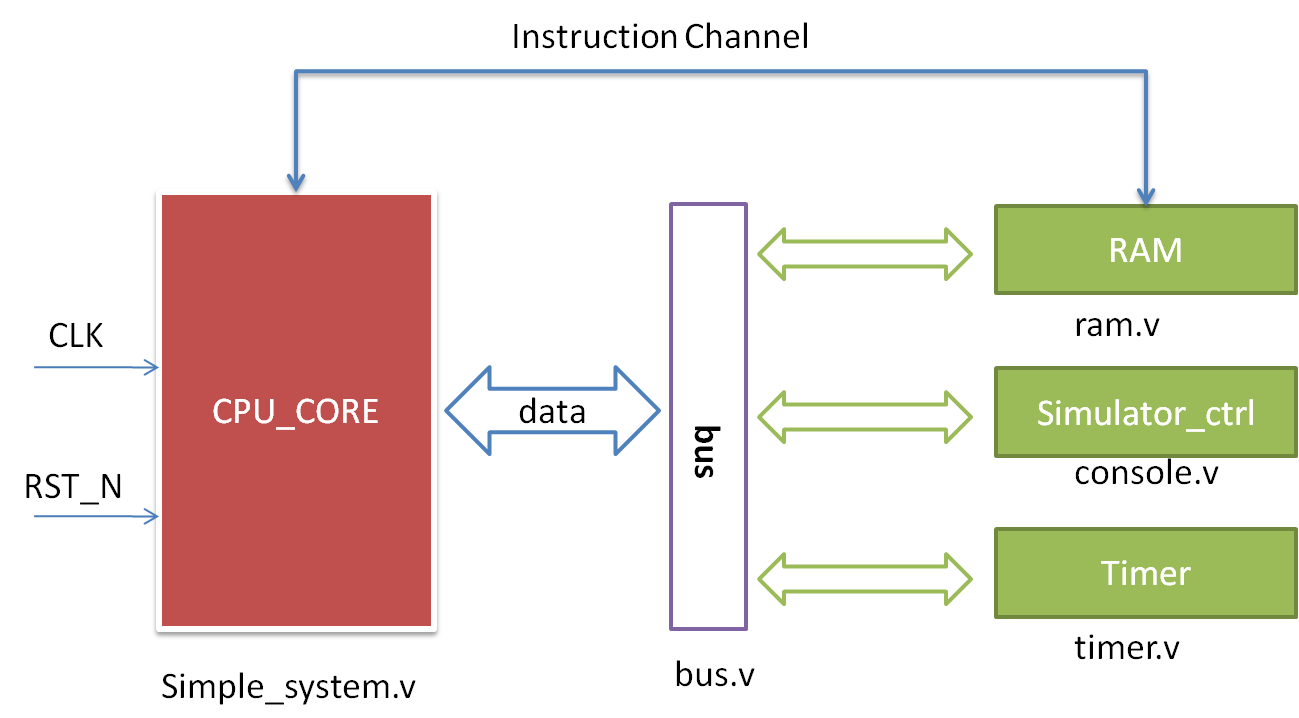
1. The micro-architecture

The implementation did not dwell on the micro-architecture design, it has adopted the typical 5 stages pipeline (instruction fetch, instruction decoder, execute, LSU and write back), with the control unit in charge of coordinating all the components as well as handling the exception and interruptions.



2. Test bench

The test bench consists of the CPU core, the data bus as well as some other necessary peripheral devices, including a dual-port RAM (one port for instruction fetching, the other for data access), the simulator ctrl (acting as the console to display the debug information), and a timer to produce interrupts. The bus, simulator ctrl and the timer are from the open-source project ibex provided by lowRISC.

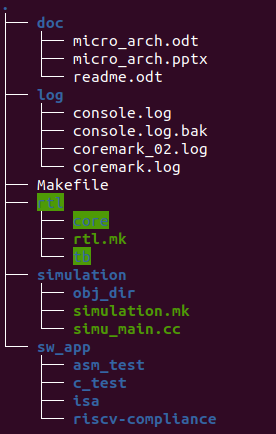


3. verification

3.1 requirements

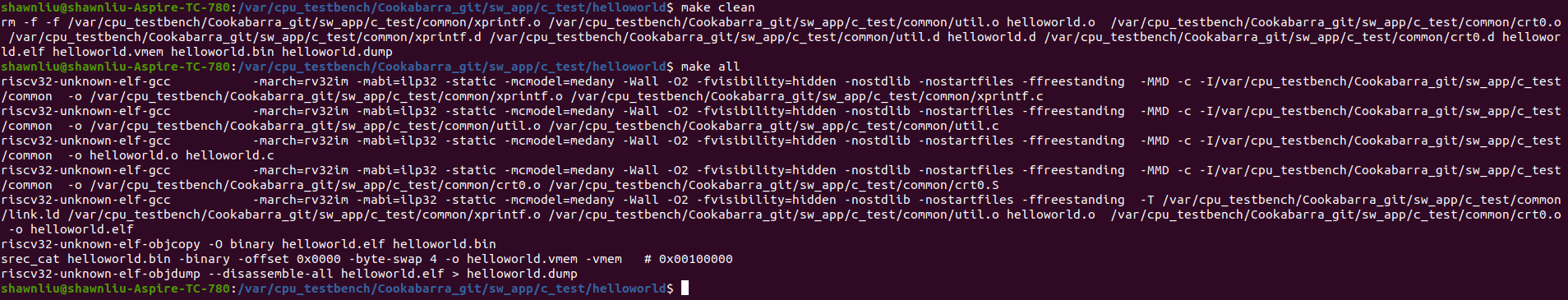
To run the simulation, the two components (verilator and rv32 gcc toolchain) are necessary, these are accessible from their official website.

3.1 Compile the CPU and test bench

there is a makefile in the top directory that can be used to build the whole test bench.



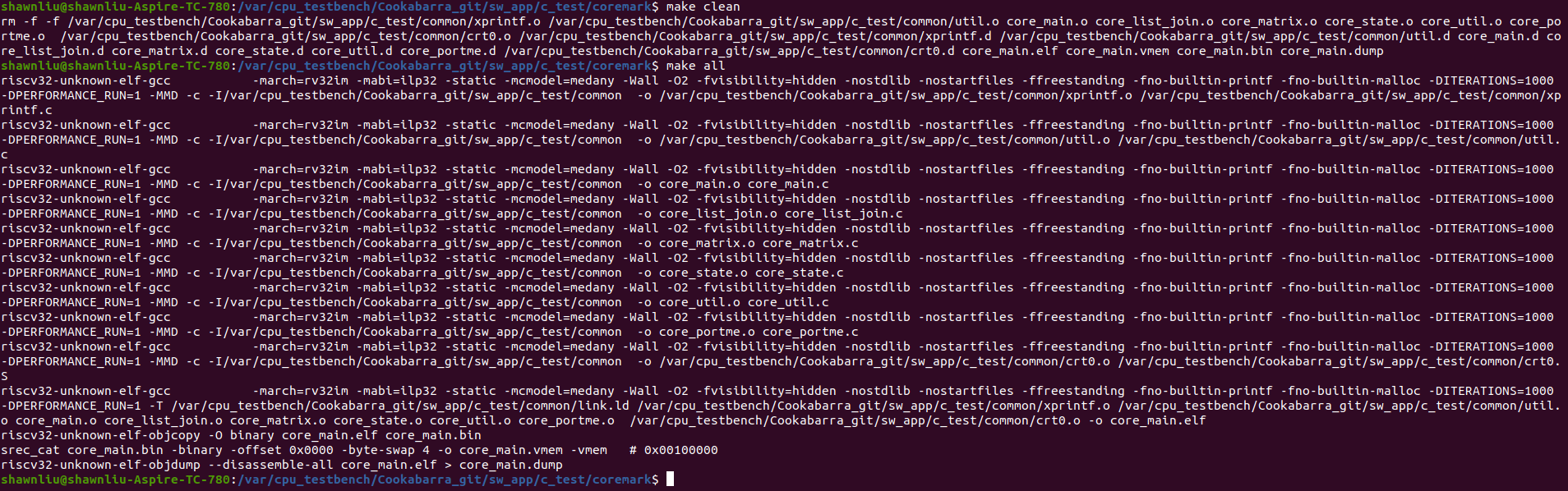
3.2 Test

  
you can run the hellworld test case via the following command:

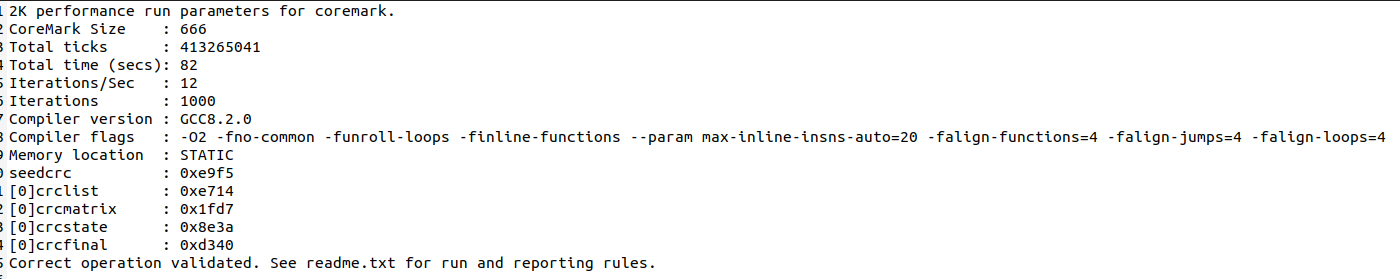
./tb ./sw\_app/c\_test/helloworld/helloworld.vmem

3.3 Coremarks

Tested with the coremark(downloaded from <https://github.com/eembc/coremark.git>)

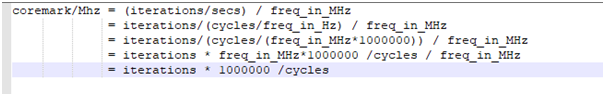
follow the command below to run the coremark:

./tb ./sw\_app/c\_test/coremark/core\_main.vmem



coremark/Mhz = 1000\*1000000.0/413265041.0 = 2.42

The formula to calculate the coremark/Mhz:



Note: Seen a lot of nop instruction were inserted into the pipeline when there is a branch or jump happened, the performance can be expected to improve significantly by introducing a branch prediction unit.