第四次实验报告

20307130112 马成

一、 实现 CSR 寄存器

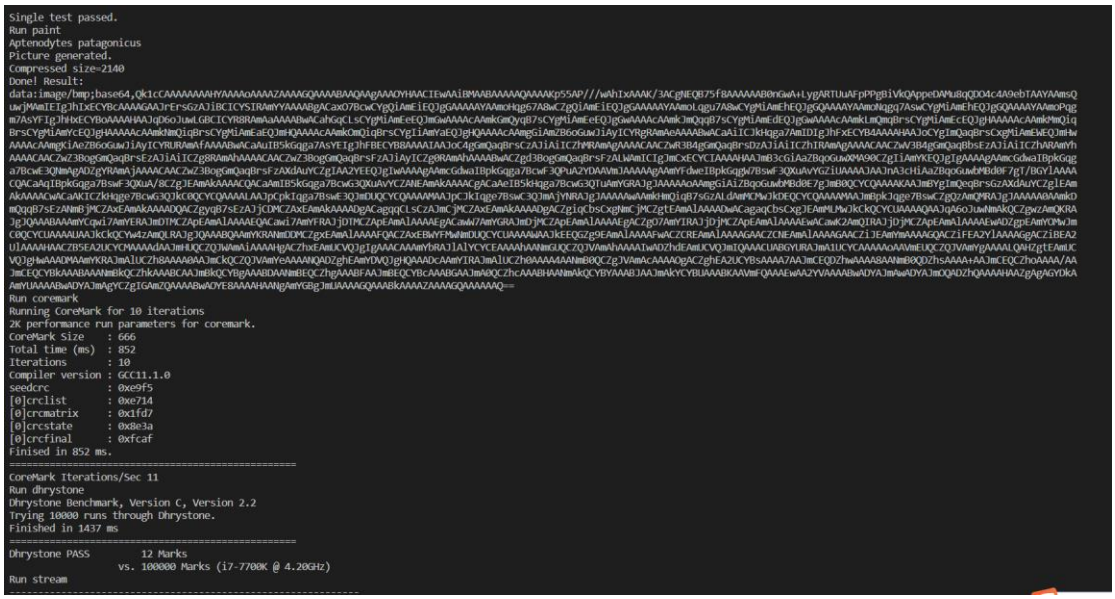
1. CSR 的整体架构参考了老师给出的代码和 regfile 的相关操作
2. CSR 的功能主要我分成了四块，第一是检测异常，第二是检测中断，第三是检测指令中的 CSR 指令并对之进行处理，第四就是对 MRET 的处理
3. 对于 CSR 指令的读操作，几乎和 regfile 完全一致，直接使用组合逻辑读出即可，对于写操作，我为了保证 ALU 模块可以正常的计算需要写入 rd 的值，我将对于 CSR 中写入值的计算直接写在了 CSR 寄存器模块内。先计算结果，然后也是类似于 regfile 写指令的操作更改 CSR 的对应值
4. 使用组合逻辑对上述的四种情况按顺序一一进行考虑(其实这四种情况是互斥的，不会出现一条指令兼具几种情况的问题)，这里对于 CSR 寄存器的修改完全参照文档即可。

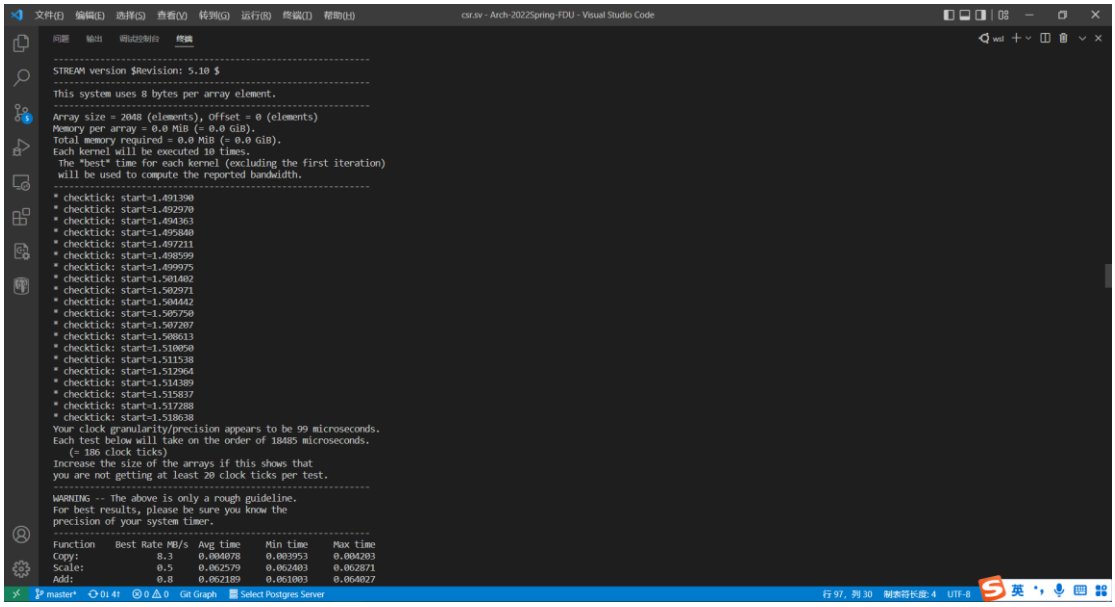
二、 对于流水线的修改

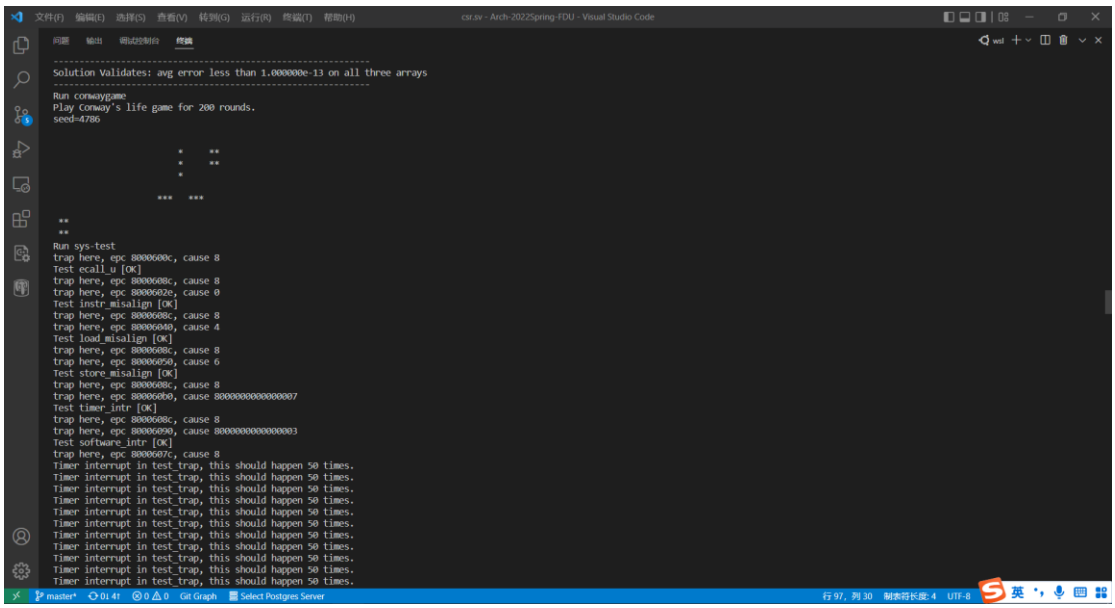
1. 对于 CSR 类型指令的处理：在 decoder 中加入对于 CSR、MRET、ECALL 指令的识别。在 decode 阶段通过向外传递 csrra，通过 csrrd 读取 CSR 对应的值，并通过 immediate 模块对 dataD 的 srca 和 srcb 进行布局，方便后续的处理。将 x[rs1]或 zmm[4:0]存储在 dataD.csr 中，CSR 的对应值会在后续的 ALU 运算中自动被存入 result 中不必担心。ECALL 和 MRET 指令对流水线不需要太多的改动，只需要识别即可。由于 CSR 指令或者是任何会改变 CSR 寄存器的操作都会导致整个流水线的刷新，所以其实我们不必考虑 CSR 寄存器的数据冲突问题，因为就算冲突了，后面出现的指令也会被刷新不会被执行。
2. 识别异常：在 FETCH 阶段可能导致的异常是 pc[1:0]!=2’b00 将 dataF.error=FETCHERROR，同时将 ireq.valid 和 stopf 等进行相应的修改，后续的异常还有在 decode 阶段未找到相应指令的解析方式，在 memery 阶段根据所取数据的 msize 发现地址未对齐。对这些异常指令的处理方式类似于 FETCH 阶段的处理，当发现你获得的 data 中已经是一条错误指令，将他看做费指令不予理会即可。
3. 流水线的刷新：由于 cbus 的合理性要求，dreq 和 ireq 不能在中途改变。因此在处理异常的时候我分成两个步骤进行处理。当出现异常但是 stopm 为 1 的时候，说明这时候 memory 阶段的 dreq 还在取指令不能变化，所以所有的寄存器保持原样不变，由于这是 stopm 自己就可以完成的，所以不需要额外操作。当 stopm 为 1 的时候，将 decode 和 execute 的指令 flush，但是 ireq 的情况还未知，继续让他取指令，memory 中由于 execute 的指令被刷新，dreq 不再工作，但是 dataM 还要保持不变以维持 csr 的信息来源。指导 stopf 也为 0 表示所有的流水线的取内存操作全部完成，这时候将所有流水线寄存器全部刷新即可。这时我再对 csr 寄存器进行一些列操作，同时按照要求取出新的 PC，在 fetch 阶段更新 pc_next 即可

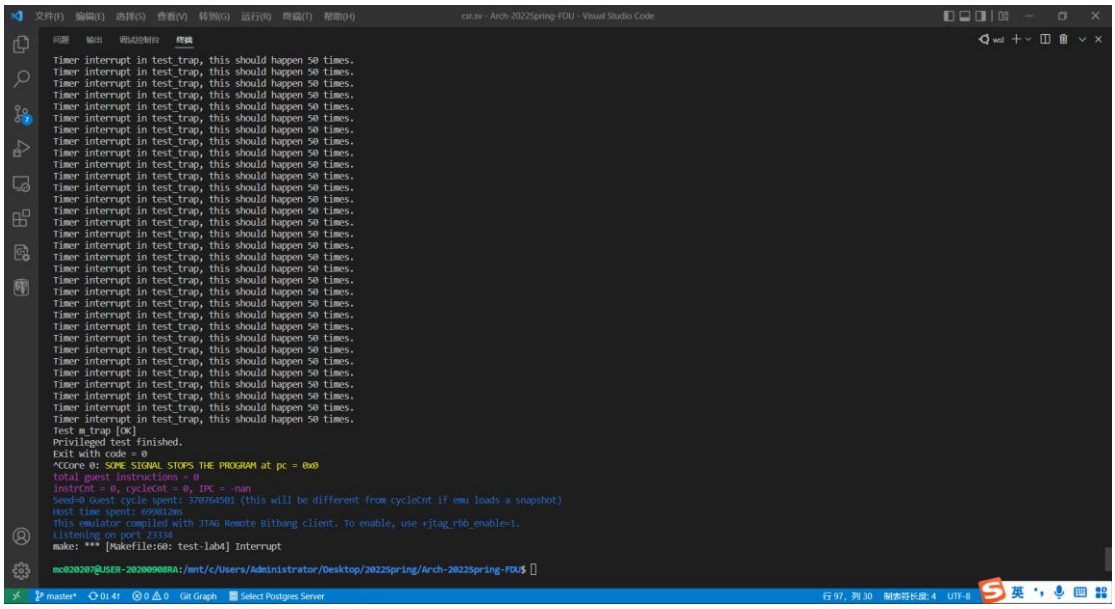
三、 通过截图

1. Test-lab4 TEST=all 通过截图






csr.sv - Arch-2022Spring-FDU - Visual Studio Code
问题 输出 调试控制台 终端
Solution Validates: avg error less than 1.000000e-13 on all three arrays
Run conwaygame
Play Conway's life game for 200 rounds.
seed=4786
Run sys-test
trap here, epc 8000600c, cause 8
Test ecall_u [OK]
trap here, epc 8000600c, cause 8
trap here, epc 8000602e, cause 0
Test instr_misalign [OK]
trap here, epc 8000600c, cause 8
trap here, epc 80006040, cause 4
Test load_misalign [OK]
trap here, epc 8000600c, cause 8
trap here, epc 80006050, cause 6
Test store_misalign [OK]
trap here, epc 8000600c, cause 8
trap here, epc 800060b0, cause 8000000000000007
Test timer_intr [OK]
trap here, epc 8000600c, cause 8
trap here, epc 80006090, cause 8000000000000003
Test software_intr [OK]
trap here, epc 8000607c, cause 8
Timer interrupt in test_trap, this should happen 50 times.
master*
Git Graph
Select Postgres Server
行 97，列 30
制表符长度: 4
UTF-8


csr.sv - Arch-2022Spring-FDU - Visual Studio Code
问题 输出 调试控制台 终端
Timer interrupt in test_trap, this should happen 50 times.
Test m_trap [OK]
Privileged test finished.
Exit with code = 0
^CCore 0: SOME SIGNAL STOPS THE PROGRAM at pc = 0x0
total guest instructions = 0
instrCnt = 0, cycleCnt = 0, IPC = -nan
Seed=0 Guest cycle spent: 370764501 (this will be different from cycleCnt if emu loads a snapshot)
Host time spent: 699812ms
This emulator compiled with JTAG Remote Bitbang client. To enable, use +jtag_rbb_enable=1.
Listening on port 23334
make: *** [Makefile:60: test-lab4] Interrupt
mc020207@USER-20200908RA:/mnt/c/Users/Administrator/Desktop/2022Spring/Arch-2022Spring-FDU$
master*
Git Graph
Select Postgres Server
行 97，列 30
制表符长度: 4
UTF-8