**体系结构实验第二周实验报告**

小组成员：许诗瑶20023105、刘朝润20023114、刘晓航20020070

**一、通过测试程序比较指令**

1、使用C语言编写 冒泡排序 程序，详见附件test\_program/BubbleSort/BubbleSort.c,使用编译工具链生成同文件夹下BubbleSort.out和BubbleSort.s文件，将编译后的可执行程序反汇编后生成BubbleSort.txt文件。

2、使用RV汇编指令手动编写 冒泡排序 程序，详见附件test\_program/BubbleSort/BubbleSortAss.txt。

3、使用C语言编写 计算第n个斐波那契数 程序，详见附件test\_program/Fibonacci/Fibonacci.c,使用编译工具链生成同文件夹下Fibonacci.out和Fibonacci.s文件，将编译后的可执行程序反汇编后生成Fibonacci.txt文件。

4、使用RV汇编指令手动编写 冒泡排序 程序，详见附件test\_program/ Fibonacci / Fibonacci\_ass.txt。

5、比较C语言编译后生成的BubbleSort.s和Fibonacci.s汇编指令与使用RV汇编手动编写的BubbleSortAss.txt与Fibonacci\_ass.txt指令差别：

从指令应用层面上而言，C语言编译生成的汇编程序是基于操作系统层次，其开头有一些汇编指示符，如.file、.text、.globl等汇编器的命令，以及一些用于指向指令跳转标签.L2、.L3；而手动编写的RV汇编指令是基于机器层次，即可以直接送入程序存储器的简单指令，没有指示符以及标签等，跳转是通过branches类指令的立即数对PC值更改实现。

从使用的寄存器而言，C语言编译生成的汇编程序遵循了RISCV函数调用中的相关规定，如下表所示：



而我们人为编写的RV汇编指令使用寄存器时直接使用其名称（x0、x1、x2等），只默认x0值固定为0，其他寄存器不做约束直接使用，更接近机器底层。

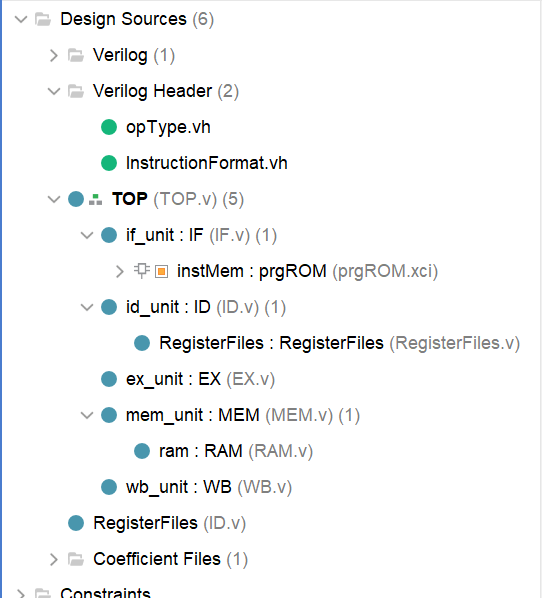
从指令数量上而言，C语言编译生成的汇编程序指令条数更多，观察冒泡排序代码可发现，由于C语言编写时使用了for循环，编译时经过循环展开，指令条数是手动编写的汇编指令的几倍，以及C语言编写时有函数调用等过程，更增加了指令数量。

从指令类型上而言，我们手动编写的汇编指令只参考RV32I基本指令集，而C语言汇编出的指令是通过伪指令表示，但其本质还是基于基础指令，但汇编后也出现了一些如mv的特权指令、li的16位RVC集的指令等，指令系统更大。通过对Fibonacci程序增加输出，观察汇编出的结果，指令数量增加了许多，最明显的是增价了.string用于输出的字符串等待被调用使用。

**二、实现五级流水线**

使用Verilog语言，本周实现了基于取指IF、译码ID、执行EXE、访存MEM、写回WB的五级流水线，目前只实现了基于RV32I指令集的基本功能，经测试能成功执行手动编写的计算Fibonacci程序产生正确结果。代码详见RISCV文件夹，文件结构为Vivado项目工程文件结构。

设计思路：代码编写前粗略画出流水线结构草图（但后期对结构细节改动较大，结构图与真正代码所实现的结构有所出入，此处不将草图予以展示），将流水线拆分为5级，每级一个主模块，约定和规范每级（每模块）的输入输出，最后通过顶层设计文件TOP.v将流水线模块整合。代码文件结构如下图所示：



遇到的重要问题及解决方案：

1、初步设计时将分支指令的执行提前到译码阶段，但在后续实现过程中发现与改设计冲突的各种逻辑问题，如不能将译码时期的分支判断控制在一周期内，数据冲突时PC值混乱等问题，最后还是决定将分支地址计算正常置于执行阶段。

2、程序存储器与数据存储器初步设计为调用ip和实现，但实现Store指令时发现存入数据位宽又一字节、半字、一字的差别，无法通过固定的32位位宽数据传入8位数据，于是将数据存储器改用手动使用寄存器实现。

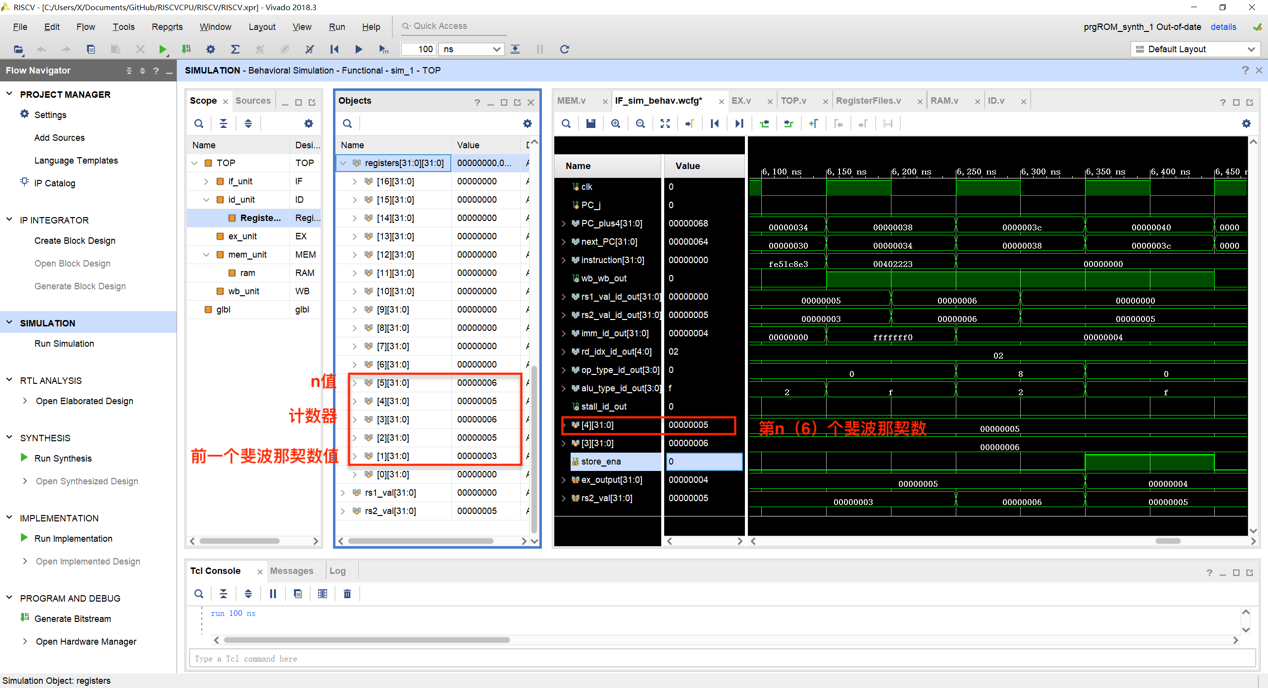
3、数据冲突的解决，通过维护一张寄存器状态表检测数据冲突的有无，由于本周只实现了最基本的流水线结构，暂时没有对冲突使用其他方案解决，数据冲突存在时直接通过stall信号产生停顿。

4、对分支跳转的处理，通过冲刷流水线，将正确分支前错误的取指和译码冲刷掉，返回正确的PC值等。

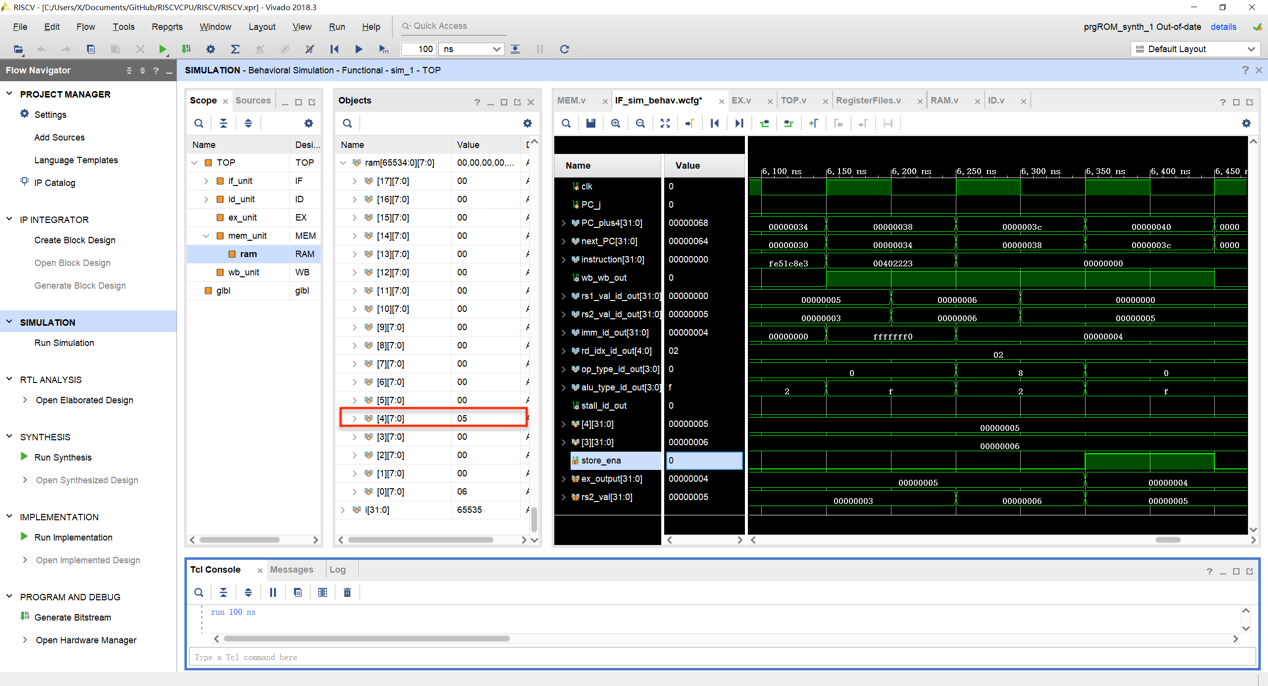
测试程序及结果：

将计算第n个斐波那契数的程序指令二进制文件Fibonacci.coe导入程序存储器中，将需要读取的n值，设置为6，存入数据存储器中，最终测试结果如下：

下图为执行到第6个斐波那契数时，寄存器中的相关值：



下图为执行到写回第6个斐波那契数时，数据存储器RAM中的的值，可得第6个斐波那契数（值为5）写入mem[1]（ram[4:7]）中：



后续改进：

1、代码规范化，由于实现过程不断调试，代码不够规整，可读性不高，各级模块的端口信号错综复杂，有待进一步调整；

2、丰富解决冲突的方法，当前仅实现对数据冲突的执行停顿操作，后续打算进一步实现旁路等其他解决方法。

附件说明：附件为包含

1、项目说明README.md

2、用于测试的程序fibonacci.coe

3、斐波那契和冒泡排序的test\_program文件夹

4、指令集相关的ISA文件夹

5、RISCV项目工程