**实 验 报 告**

实验名称： RISC-V 32I的实现与测试

学 员： 邱哲豪 学 号： 20023062

目录

[一、 实验要求 3](#_Toc92999310)

[二、 环境配置 3](#_Toc92999311)

[三、 实验步骤 3](#_Toc92999312)

[(一) 指令系统设计 3](#_Toc92999313)

[(二) 五级流水线CPU的设计与实现 4](#_Toc92999314)

[1. 总体架构 4](#_Toc92999315)

[2. IF级 4](#_Toc92999316)

[3. ID级 5](#_Toc92999317)

[4. EXE级 6](#_Toc92999318)

[5. MEM级 7](#_Toc92999319)

[6. WB级 8](#_Toc92999320)

[四.系统验证与测试 8](#_Toc92999321)

[（一）编译 8](#_Toc92999322)

[（二）spike模拟器 9](#_Toc92999323)

[五、实验结果分析 9](#_Toc92999324)

[1. Fibonacci 10](#_Toc92999325)

[（1） . C程序实现计算第n个斐波那契数 10](#_Toc92999326)

[（2）. RV汇编实现计算第n个斐波那契数 12](#_Toc92999327)

[2. BubbleSort 13](#_Toc92999328)

[（1）C程序实现冒泡排序 13](#_Toc92999329)

[（2）. RV汇编实现冒泡排序 16](#_Toc92999330)

[3、实验结果及分析 17](#_Toc92999331)

[（1）C程序计算第n个斐波那契数的结果及分析 17](#_Toc92999332)

[（2）RV汇编计算第n个斐波那契数的结果及分析 18](#_Toc92999333)

[（3）C程序实现冒泡排序的结果及分析 18](#_Toc92999334)

[（4）RV汇编实现冒泡排序的结果及分析 18](#_Toc92999335)

# 实验要求

1. 实现RISC-V的子集，即所有32位的I类指令共47条；

2. 对所实现的CPU验证所设计的指令系统。

# 环境配置

vivado 2019.2

riscv-gnu-toolchain

# 实验步骤

## 指令系统设计

本次设计中指令集包括RV32I的条指令，具体如表一所示。

**表1.指令集**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Name** | **Fmt** | **RV32I Base** | **Name** | **Fmt** | **RV32I Base** |
| ADD | R | ADD rd,rs1,rs2 | ADDI | I | ADDI rd,rs1,imm |
| SUB | R | SUB rd,rs1,rs2 | LUI | U | LUI rd,imm |
| AUIPC | U | AUIPC rd,imm | SLL | R | SLL rd,rs1,rs2 |
| SLLI | I | SLLI rd,rs1,shamt | SRL | R | SRL rd,rs1,rs2 |
| SRLI | I | SRLI rd,rs1,shamt | SRA | R | SRA rd,rs1,rs2 |
| SRAI | I | SRAI rd,rs1,shamt | XOR | R | XOR rd,rs1,rs2 |
| XORI | I | XORI rd,rs1,imm | OR | R | OR rd,rs1,rs2 |
| ORI | I | ORI rd,rs1,imm | AND | R | AND rd,rs1,rs2 |
| ANDI | I | ANDI rd,rs1,imm | SLT | R | SLT rd,rs1,rs2 |
| SLTI | I | SLTI rd,rs1,imm | SLTU | R | SLTU rd,rs1,rs2 |
| SLTIU | I | SLTIU rd,rs1,imm | BEQ | B | BEQ rs1,rs2,imm |
| BNE | B | BNE rs1,rs2,imm | BLT | B | BLT rs1,rs2,imm |
| BGE | B | BGE rs1,rs2,imm | BLTU | B | BLTU rs1,rs2,imm |
| BGEU | B | BGEU rs1,rs2,imm | JAL | J | JAL rd,imm |
| JALR | J | JALR rd,rs1,imm | CSRRW | I | CSRRW rd,csr,rs1 |
| CSRRS | I | CSRRS rd,csr,r1 | CSRRC | I | CSRRC rd,csr,rs1 |
| CSRRWI | I | CSRRWI rd,csr,imm | CSRRSI | I | CSRRSI rd,csr,imm |
| CSRRCI | I | CSRRCI rd,csr,imm | LB | I | LB rd,rs1,imm |
| LH | I | LH rd,rs1,imm | LBU | I | LBU rd,rs1,imm |
| LHU | I | LHU rd,rs1,imm | LW | I | LW rd,rs1,imm |
| SB | S | SB rs1,rs2,imm | SH | S | SH rs1,rs2,imm |
| SW | S | SW rs1,rs2,imm |  |  |  |

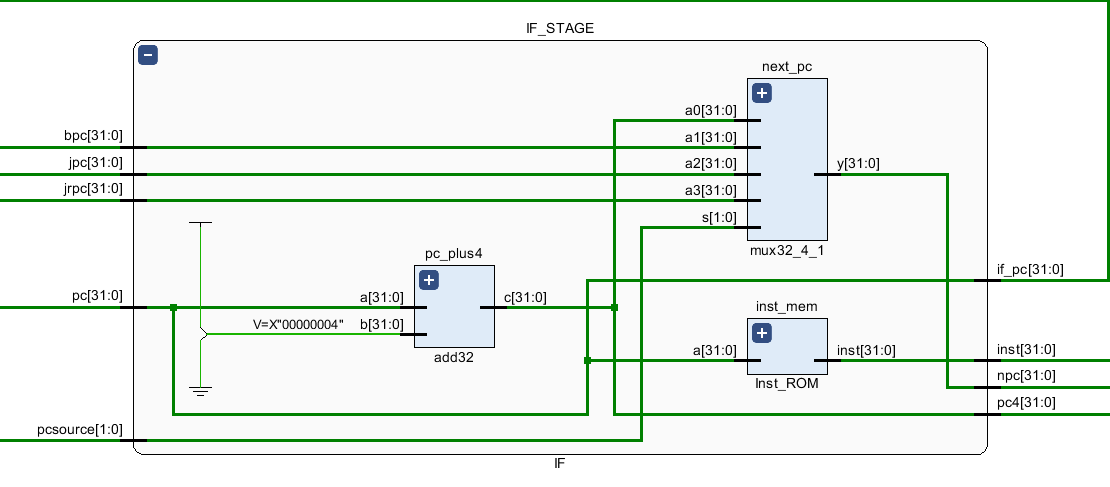
## 五级流水线CPU的设计与实现

### 总体架构



上图为本次实现的RISC-V核的总体架构成，主要实现了基础的五级流水线的各个部件，包括取指、译码、执行、访存、写回以及各级之间的4个流水线段寄存器。

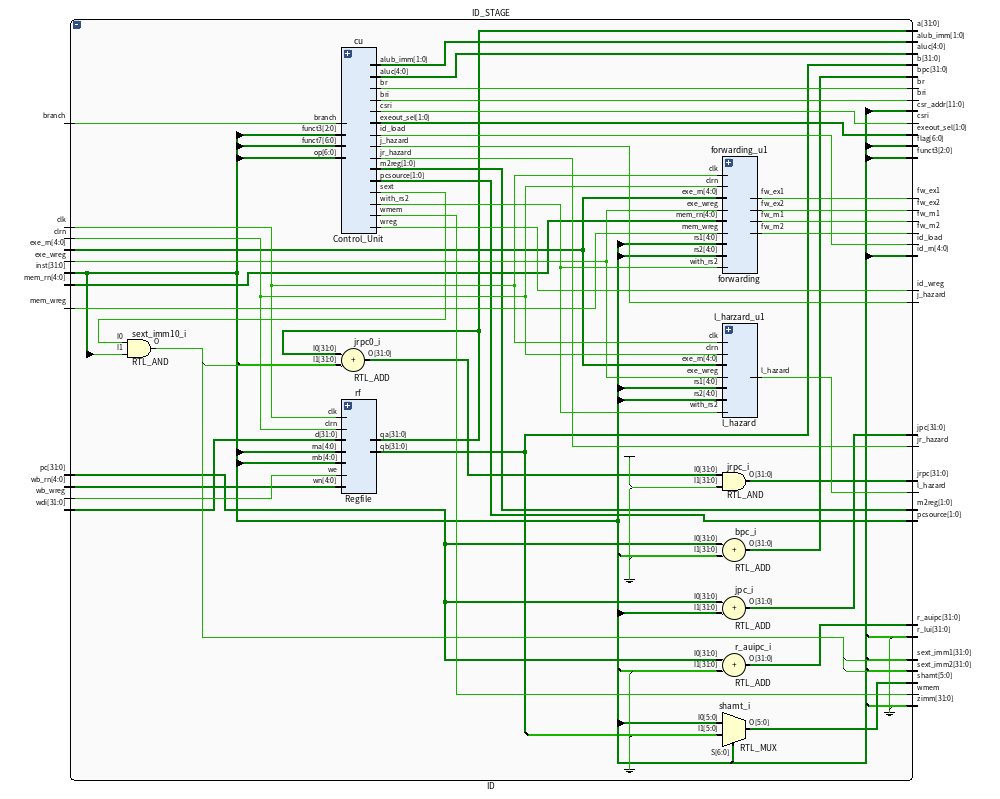
### IF级



主要包括三个部分：PC、指令寄存器、下地址选择部件。

* PC：用于保存当前正在执行的指令。
* 下地址选择部件：一个32位四选一选择器，由译码产生的信号pcsource控制选择下条指令地址，可选地址包括如下四个：pc4（pc+4）；bpc（B型指令跳转地址）；jpc（jal指令跳转地址）；jrpc（jalr指令跳转地址）。
* 指令寄存器：实际实现中是定义了64个32位wire型变量，根据指令地址输出相应的指令。

### ID级

主要包括五个部件：译码逻辑、控制器、寄存器堆、lu冒险检测模块和旁路模块

* 译码逻辑：ID级的顶层部件，将指令译码生成指令操作码（包括funct3和funct7）、源操作数（包括立即数及源寄存器号）、目的寄存器号、分支指令跳转地址、CSR类指令地址等。
* 控制器：根据指令操作码生成指令执行器件的各种控制信号，包括npc选择信号pcsource、符号扩展标志sext、EXE级输出选择信号exeout\_sel、ALU控制信号aluc、ALU部件操作数b选择信号alub\_imm、写寄存器堆信号wreg、写存储器信号wmem、B型指令标志bri、CSR类指令标志csri。
* 寄存器堆：包括32个32位寄存器。读寄存器堆是根据输入的源寄存器号输出对应源操作数；当写寄存器堆信号有效时，根据WB级传回的目的寄存器号将数据写入相应寄存器中。
* lu冒险检测模块：包含检测load -use产生的RAW数据真相关的硬件逻辑，并给出冒险的处理信号

检测的依据为：

* 旁路模块：用于提前前面级流水线中获取后面指令的源操作数，减少流水线的阻塞，加快指令执行速率。
* 旁路检测依据：

1. E\_Rs == EX/MEM.Rd，

判断EX级指令的rs1字段是否和MEM级指令的目的寄存器号相同

2.E\_Rs == MEM/WB.Rd，

判断EX级指令的rs1字段是否和WB级指令的目的寄存器号相同

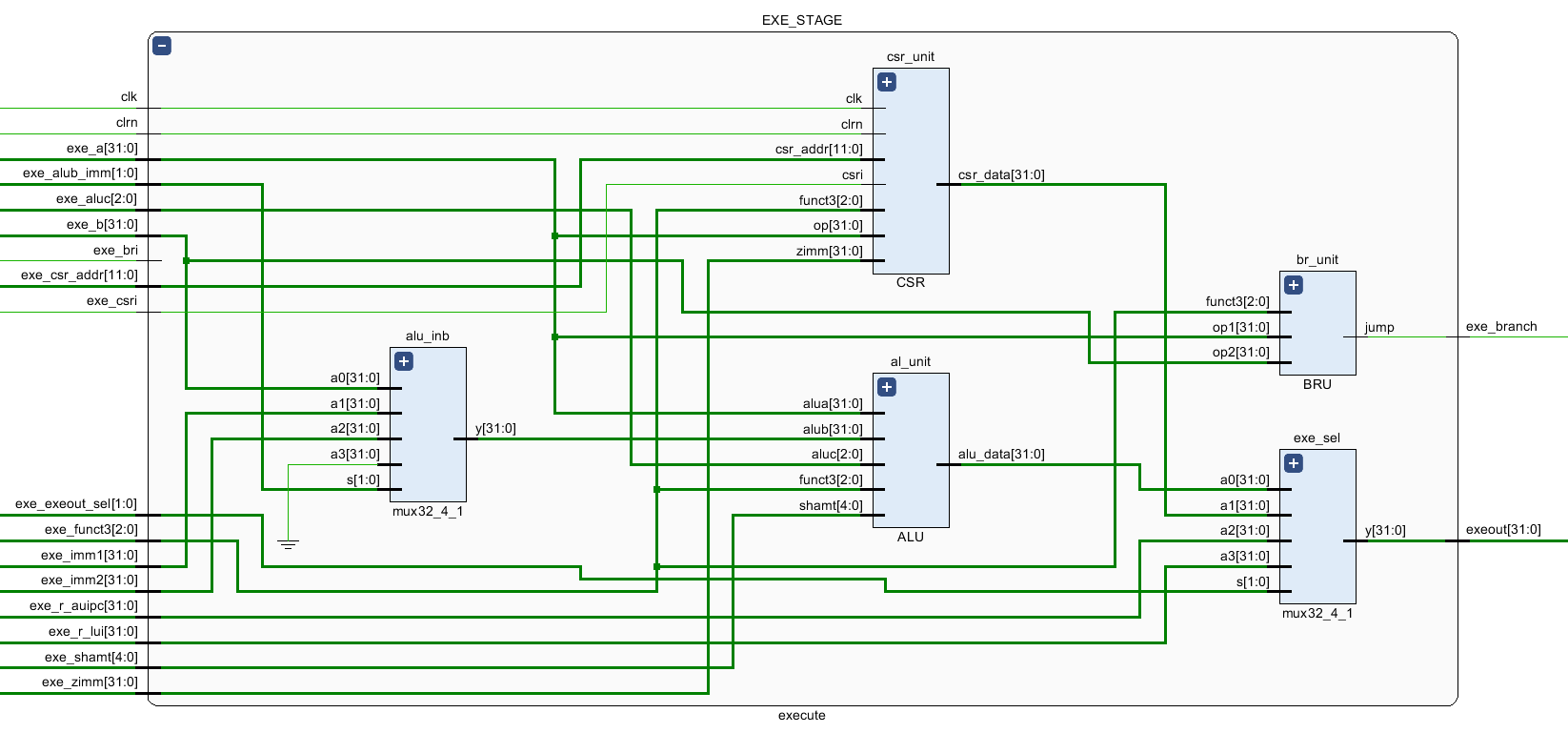
3.(E\_Rt == EX/MEM.Rd) & （with\_rs2）

判断EX级指令的rs2字段是否和MEM级指令的目的寄存器号相同

[条件d] (E\_Rt == MEM/WB.Rd) & (（with\_rs2）)

判断EX级指令的rs2字段是否和WB级指令的目的寄存器号相同

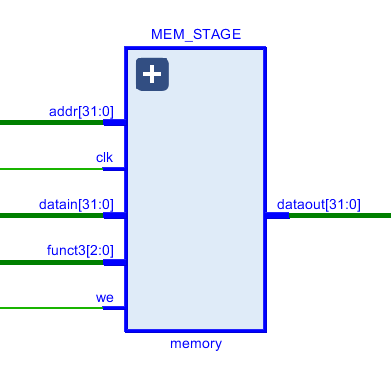
### EXE级



主要包括三个部件：ALU、BRU、CSR。

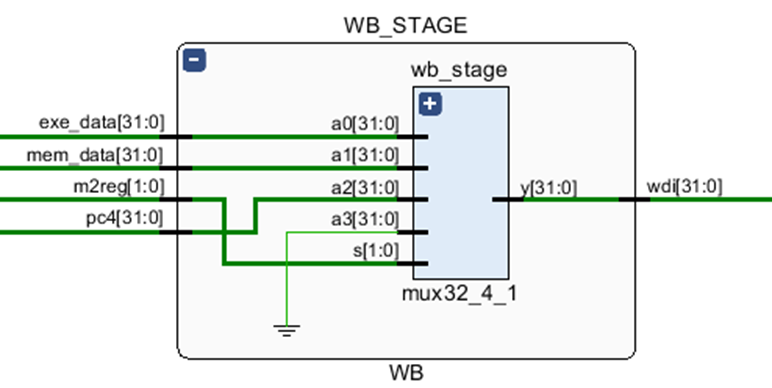
* ALU：执行基本的算数、逻辑运算，包括“+”、“-”、“&”、“|”、“^”、“>>”、“<<”、“>>>”，由ALU控制信号aluc[2:0]选择执行相应的操作。其中alua是来自rs1的寄存器数据，alub根据alub\_imm信号选择，可能是rs2的寄存器是数据、I型指令中的立即数，或S型指令中的立即数。
* BRU：执行B型指令，根据funct3选择执行相应的运算，并根据执行结果对分支跳转信号branch赋值。
* CSR：执行CSR指令，该部件中包括CSR寄存器及对CSR寄存器的读写逻辑；根据funct3选择执行相应的运算。

### MEM级



在实现中，Data\_Mem是由256个7位reg型变量组成，每个数据32位，采用4个连续的reg[7:0]存储，小端模式，在读写时根据funct3判断是读取一个字节或一个半字或一个字。

### WB级



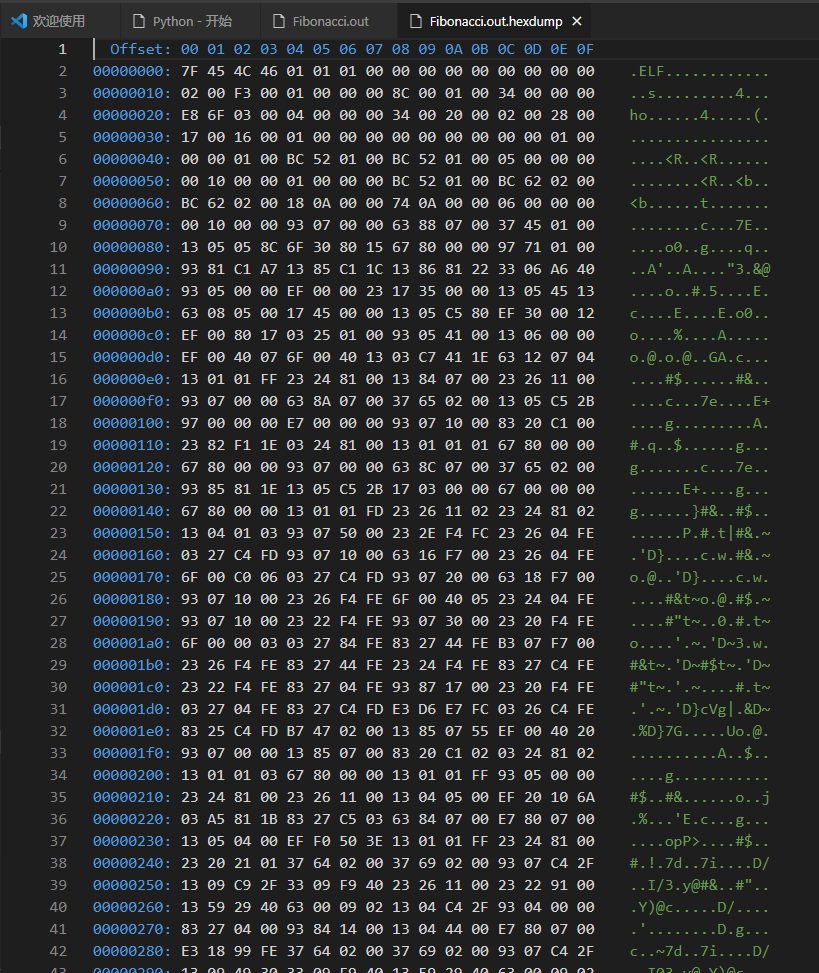
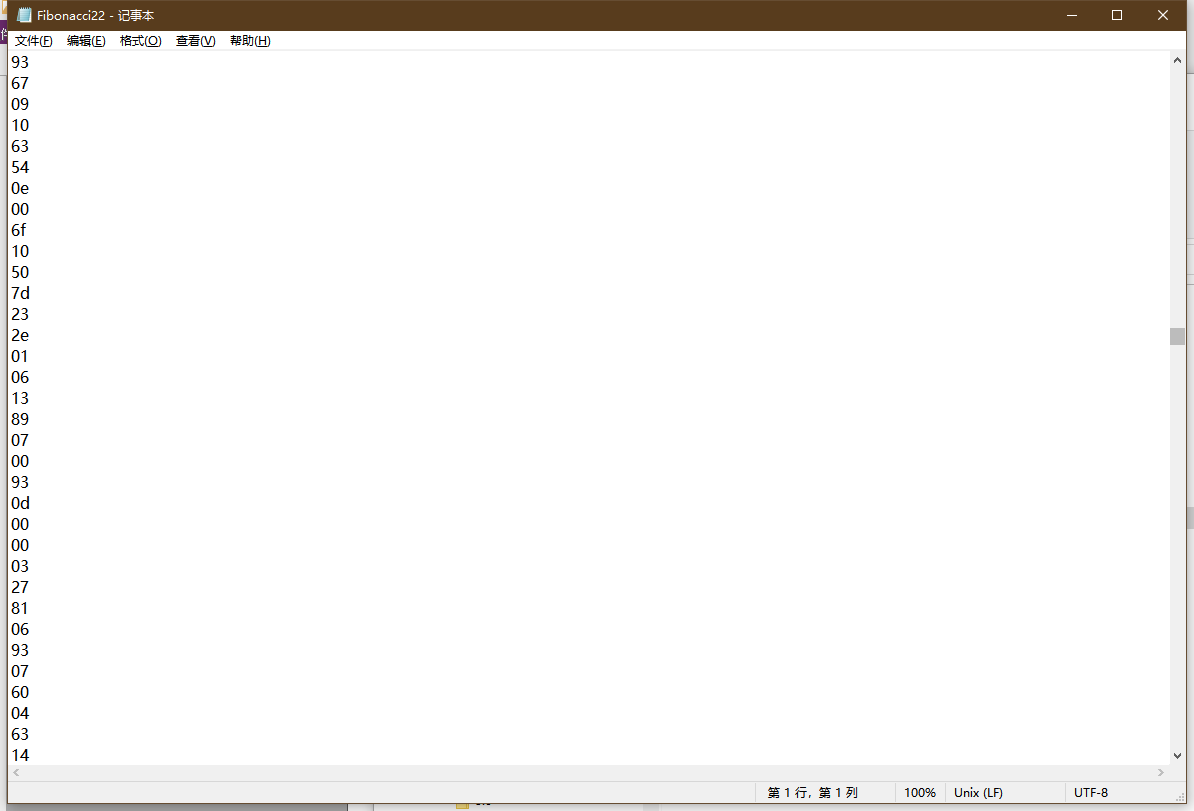
该部件的主要功能是将执行结果写回寄存器堆，根据信号m2reg判断写入的数据是来自EXE执行的结果还是load指令从Data\_Mem中读出的数据，将要写入的数据和目的寄存器号输入到ID级。

# 四.系统验证与测试

本实验选用BubbleSort和Fibonacci两种算法对该系统进行验证

## （一）编译

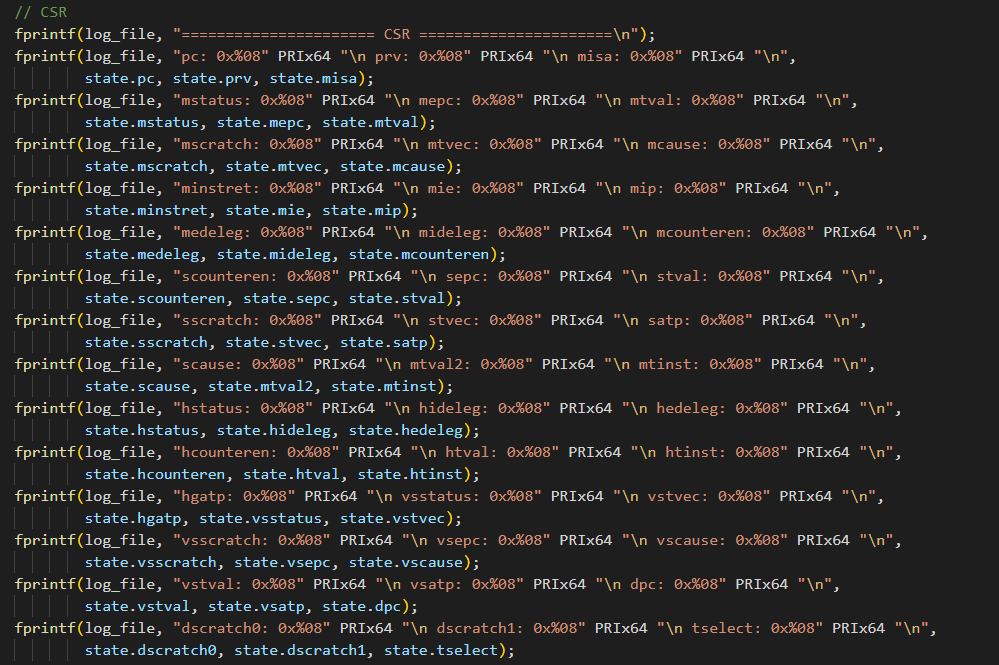
首先我们对程序BubbleSort和Fibonacci通过工具链toolchains编译生成.out文件，后通过objdump生成反汇编文件。由于 riscv32-unknown-elf-objdump 仅可转换 elf 可执行文件为汇编代码，但是会丢失了elf 中数据段的内容，因此通过命令./elf2hex 将 elf 文件直接转换成 16 进制码文件。获得相关16进制数据。



## （二）spike模拟器

为了知道真实的程序在运行中的真实状态变化与CSR寄存器变化，以及为了引做对比，我们使用了spike模拟器与PK内核运行作为参照，本身spike运行后不会打印csr指令中mepc等寄存器的内容，但是我们修改spike模拟器的代码后可以实现打印CSR寄存器值的功能。

Spike模拟器为指令行为级模拟的模拟器，能够看到寄存器中值的变化，因此此模拟器能够满足我们实验的相关要求。



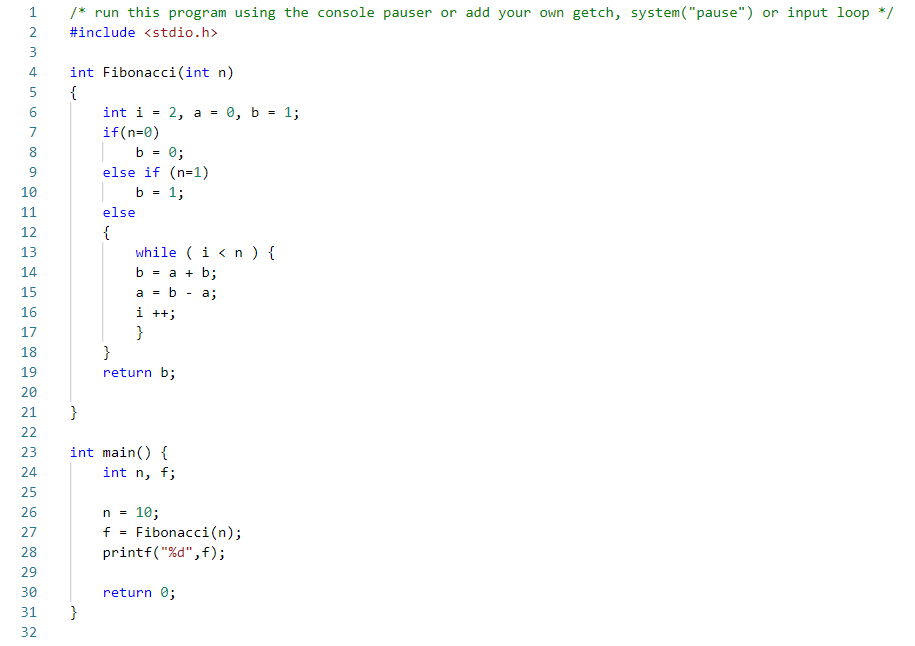
# 五、实验结果分析

准备通过模拟器的log文件与通过VIVADO将可执行文件写入指令存储器中，同时用系统的仿真波形做对比，查看指令与寄存器值的变化是否一致来验证正确性。

### Fibonacci

1. . C程序实现计算第n个斐波那契数

C程序如图3.1所示：



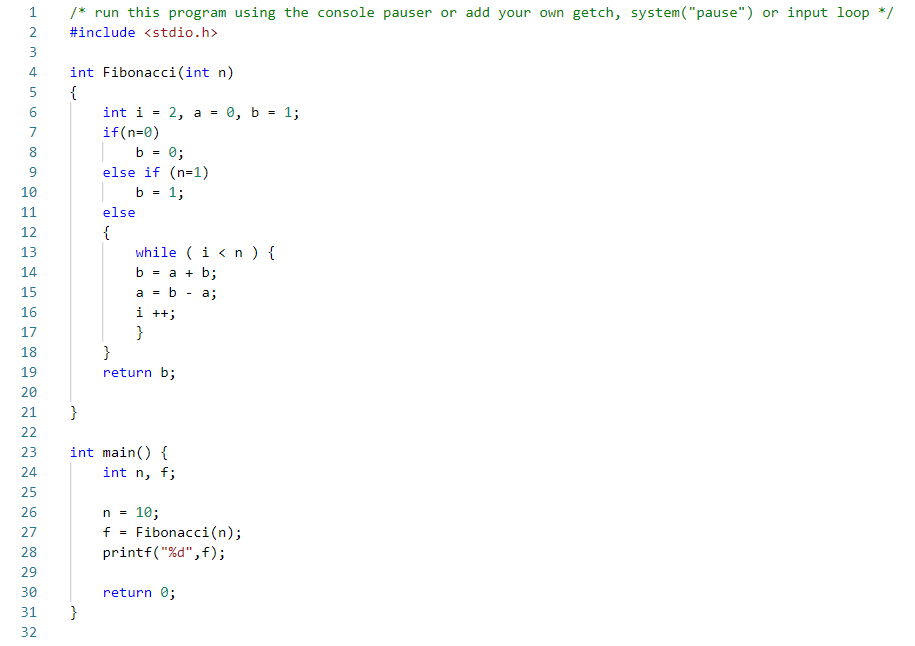
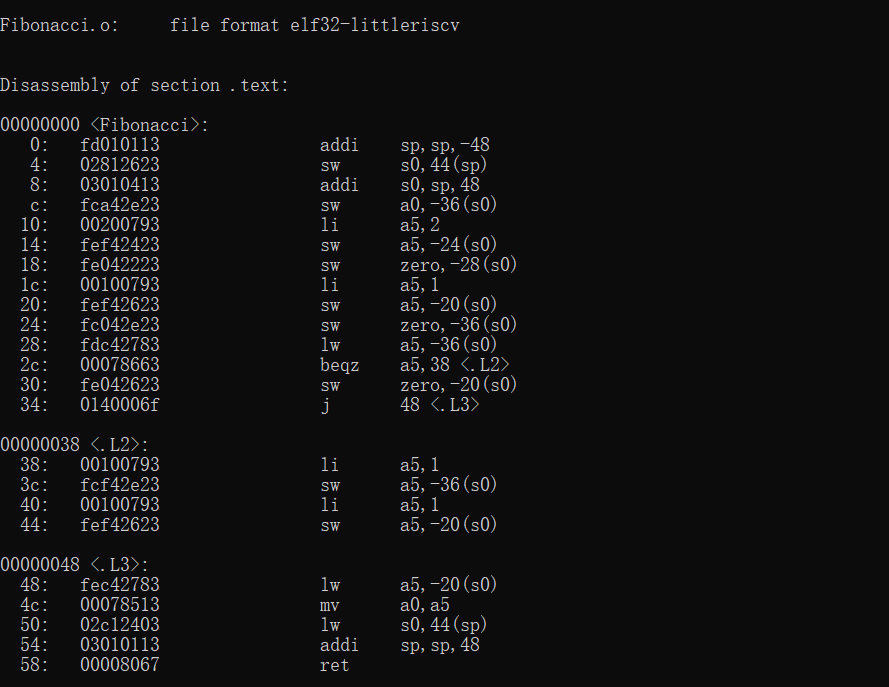


图3.1　计算第n个斐波那契数的C程序

分别用如图3.2所示的四条指令，将C程序进行预处理、编译、汇编和链接，得到的最终机器语言和汇编语言如图所示：



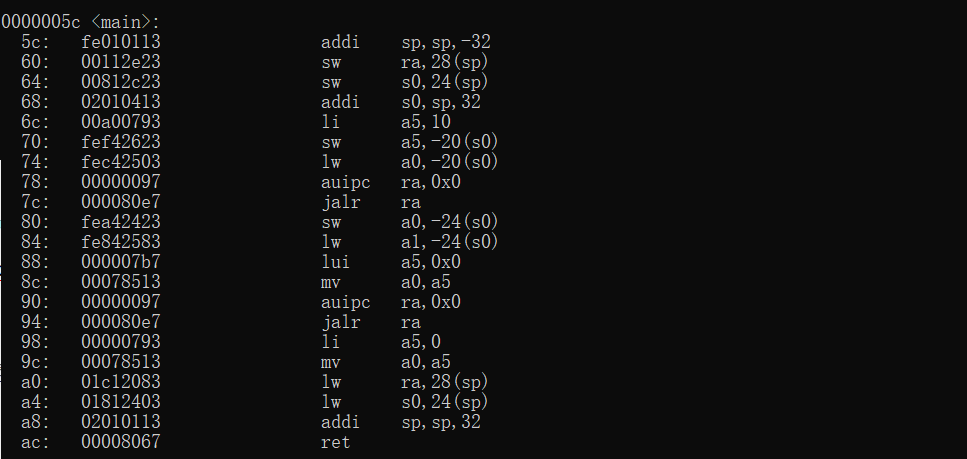


图3.2　C程序对应的机器语言和汇编语言

将图3.2中的机器码放入vivado中进行仿真，查看对应寄存器中的值，即可验证CPU中指令集编写的正确性。

（2）. RV汇编实现计算第n个斐波那契数

RV汇编程序如图3.3所示：

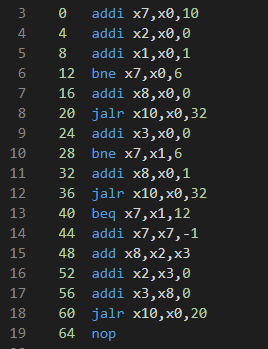


图3.3　计算第n个斐波那契数的RV汇编程序

将图3.4中的汇编代码转化为机器码放入vivado中进行仿真，查看对应寄存器中的值，即可验证CPU中指令集编写的正确性。

### 2. BubbleSort

（1）C程序实现冒泡排序

C程序如图3.5所示：

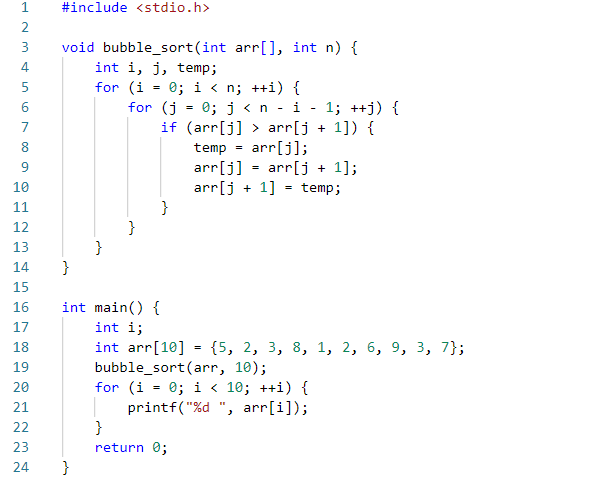
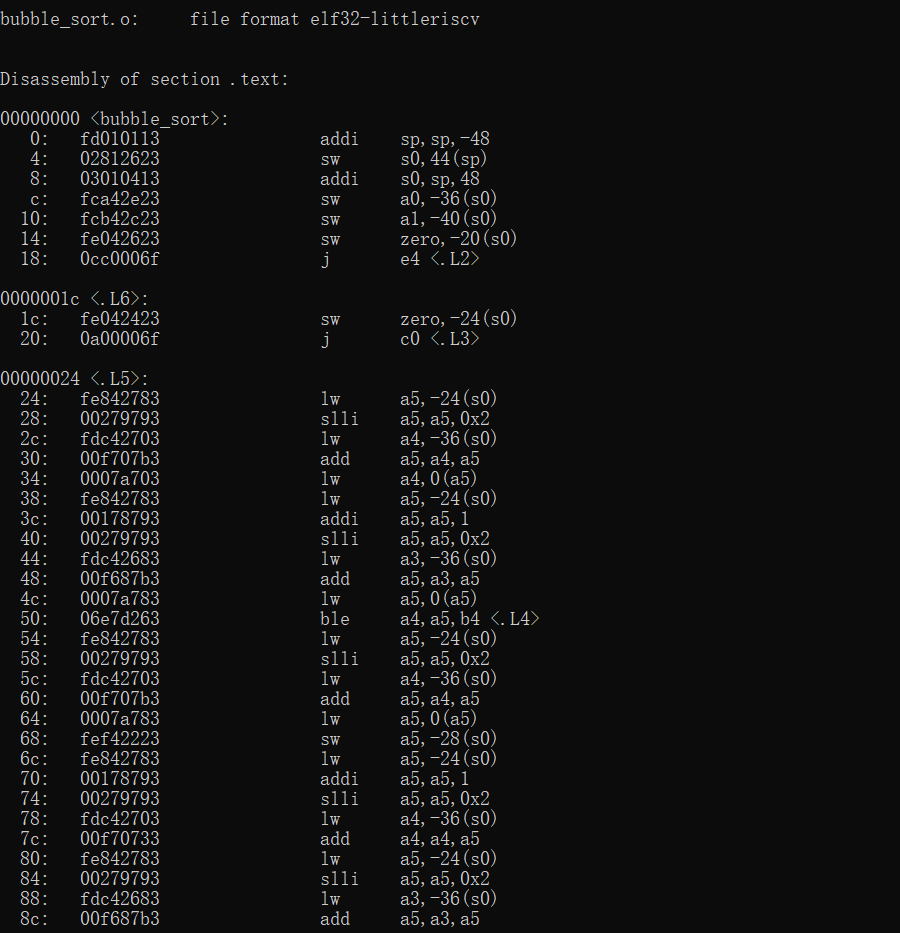


图3.5　实现冒泡排序的C程序

分别用如图3.6所示的四条指令，将C程序进行预处理、编译、汇编和链接，得到的最终机器语言和汇编语言（部分）如图所示：



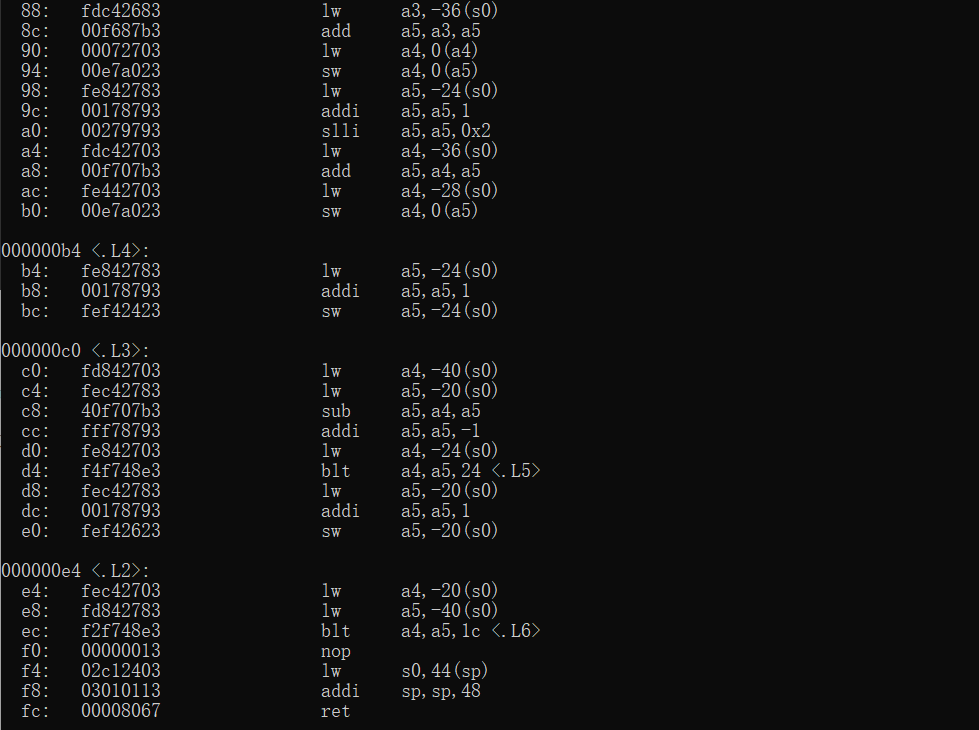


图3.6　C程序对应的机器语言和汇编语言将图3.6中的机器码放入vivado中进行仿真，查看对应寄存器中的值，即可验证CPU中指令集编写的正确性。

（2）. RV汇编实现冒泡排序

RV汇编程序如图3.7所示：

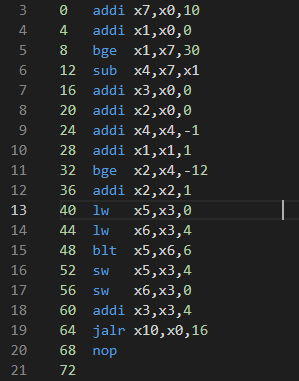


图3.7　实现冒泡排序的RV汇编程序

## 3、实验结果及分析

（1）C程序计算第n个斐波那契数的结果及分析

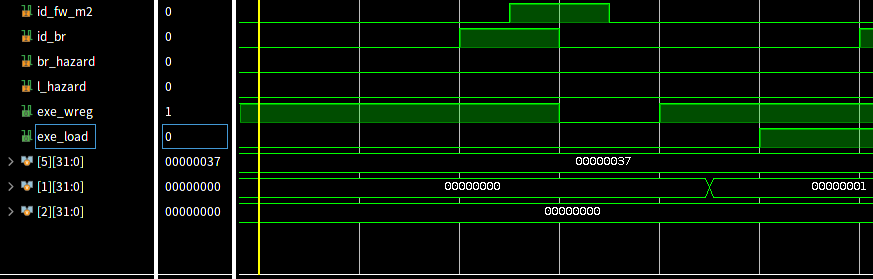


图4.1　C程序实现斐波那契的仿真结果

分析生成的汇编指令可知，第10个斐波那契数存放在第5号寄存器中。从图4.1可以看出，第5号寄存器中的值最终为37’h，转换为十进制为55’h，证明即验证CPU编写和调试的正确性。

（2）RV汇编计算第n个斐波那契数的结果及分析

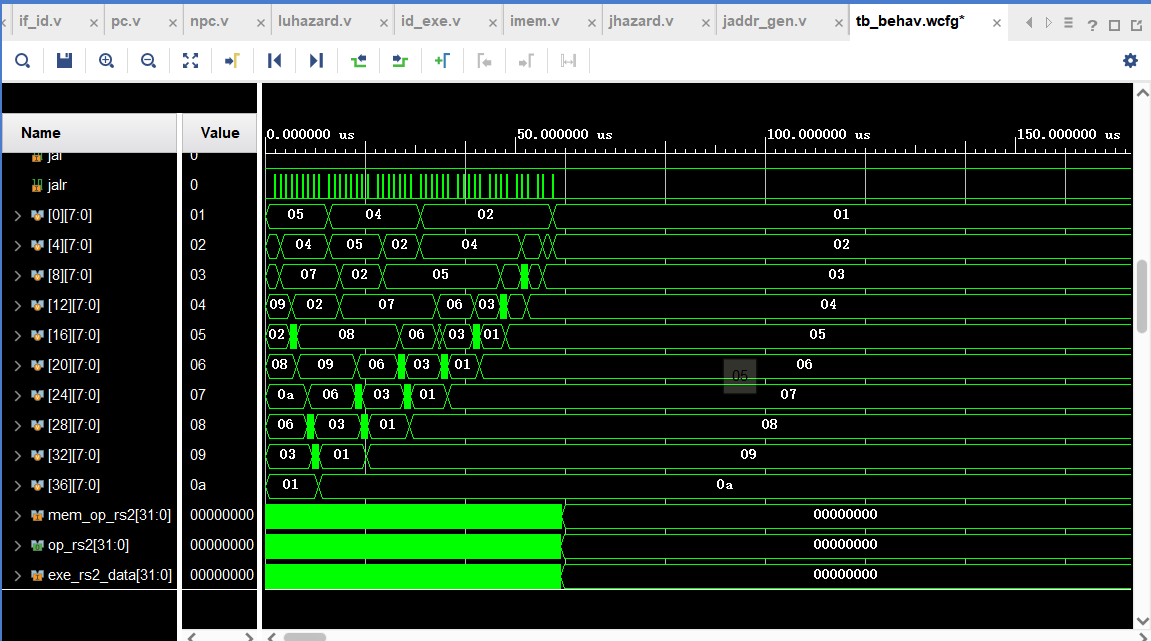
验证时，令N=10，即计算第10个斐波那契数的值。将图3.4中的机器码放入vivado进行模拟仿真，查看寄存器中最后的值是否为第10个斐波那契数，即是否为34。实验得到的部分寄存器波形如下图4.3所示：



图4.3　RV汇编实现斐波那契的仿真结果

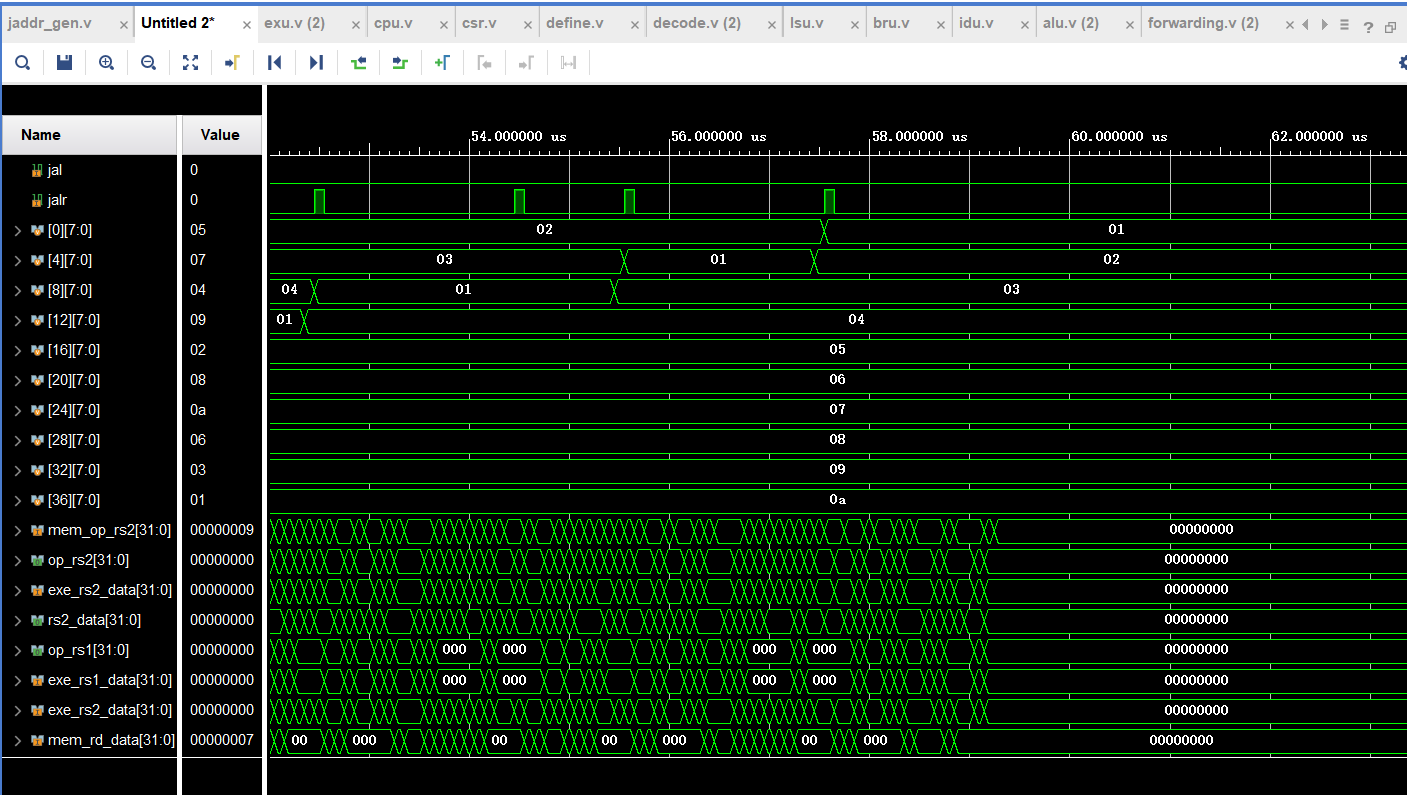
分析生成的汇编指令可知，第10个斐波那契数存放在第21号寄存器中。从图4.3可以看出，第8号寄存器中的值最终为37‘h，转换为10进制为55‘d，即验证CPU编写和调试的正确性。

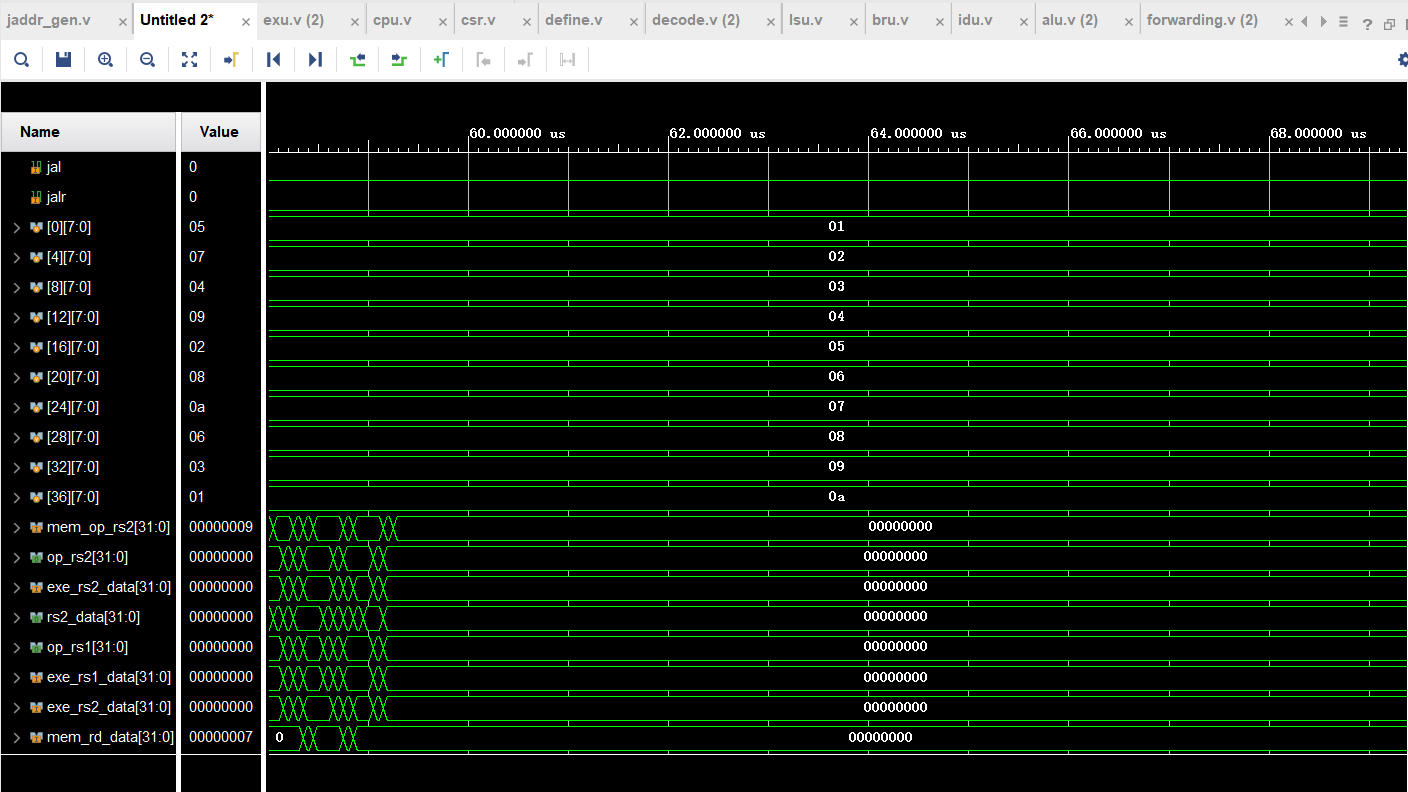
（3）C程序实现冒泡排序的结果及分析



从上述仿真结果可以验证最终的结果十个数据按从大到小的顺序排列，证明程序已在硬件中成功运行，即验证CPU编写和调试的正确性。

（4）RV汇编实现冒泡排序的结果及分析





从上述仿真结果可以验证最终的结果十个数据按从小到大的顺序排列，证明程序已在硬件中成功运行，即验证CPU编写和调试的正确性。