单周期 CPU 设计实验报告

刘宇成

1120223593

目录

单周期 CPU 设计实验报告

1. 实验内容

1. 掌握 RISCV 指令格式与编码，掌握 RISCV 汇编转换成机器代码的方法。
2. 掌握单周期 CPU 的数据通路与控制器设计方法。
3. 掌握硬件描述语言与 EDA 工具。
4. 掌握基本测试与调试方法。

2. 实验目的

1. 掌握 RISCV 指令格式与编码，掌握 RISCV 汇编转换成机器代码的方法。
2. 掌握单周期 CPU 的数据通路与控制器设计方法。
3. 掌握硬件描述语言与 EDA 工具。
4. 掌握基本测试与调试方法。

3. 实验环境

1. 硬件描述语言 Verilog HDL
2. Vivado 2019.2 版本。
3. RISC-V 仿真器：RARS

4. 实验步骤

1. 确定指令集指令条数、指令格式及编码，按照序号 1 填充表格 1。1 条指令一行，可以扩充，最后给出指令总数目。

表格 1 指令功能与数目

序号	操作码	助记符	功能	描述
1	011 0011	Add	R[rd]←R[rs]+R[rt]; PC←PC+4	加法
2	011 0011	Sub	R[rd]←R[rs]-R[rt]; PC←PC+4	减法
3	001 0011	ORI	R[rd]←R[rs] | imme; PC←PC+4	立即数或
4	110 0011	BEQ	if(R[rs]==R[rt]) PC←PC+offset; PC←PC+4	相等时分支
5	110 0011	BGE	if(R[rs]>R[rt]) PC←PC+offset; PC←PC+4	大于时分支
6	000 0011	LW	R[rd]←R[rs+imme]; PC←PC+4	读取字
7	010 0011	SW	R[rs+imme]←R[rd]; PC←PC+4	保存字
8	001 0011	ADDI	R[rd]←R[rs] + imme; PC←PC+4	立即数加
9	001 0111	AUIPC	R[rd] ← imme[31:12]<<12	立即数拓展

指令总数目	9

2. 参考图 1 设计并实现 CPU 数据通路，按照模块化设计方式，分子模块进行设计并在实验报告中记录仿真结果。

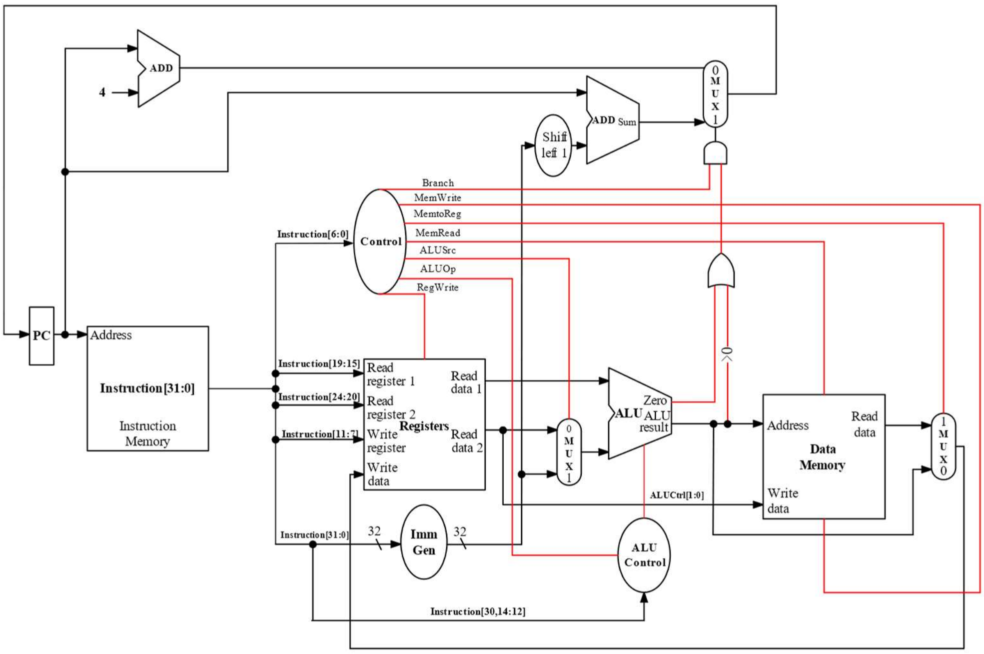


Figure 1 **单周期 CPU 数据通路**

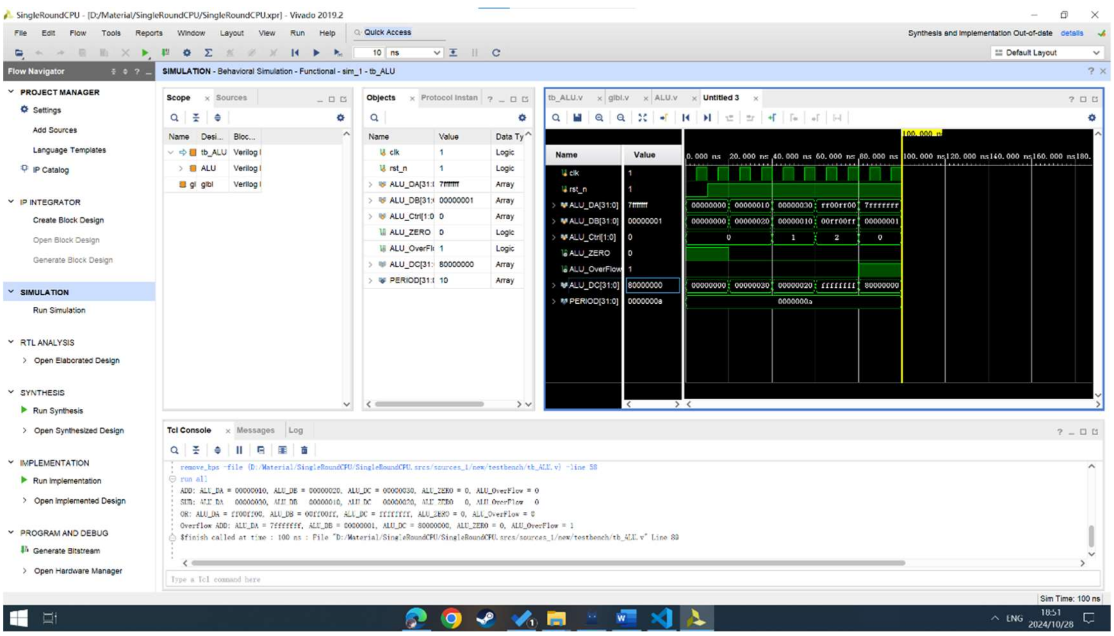


Figure 2 ALU

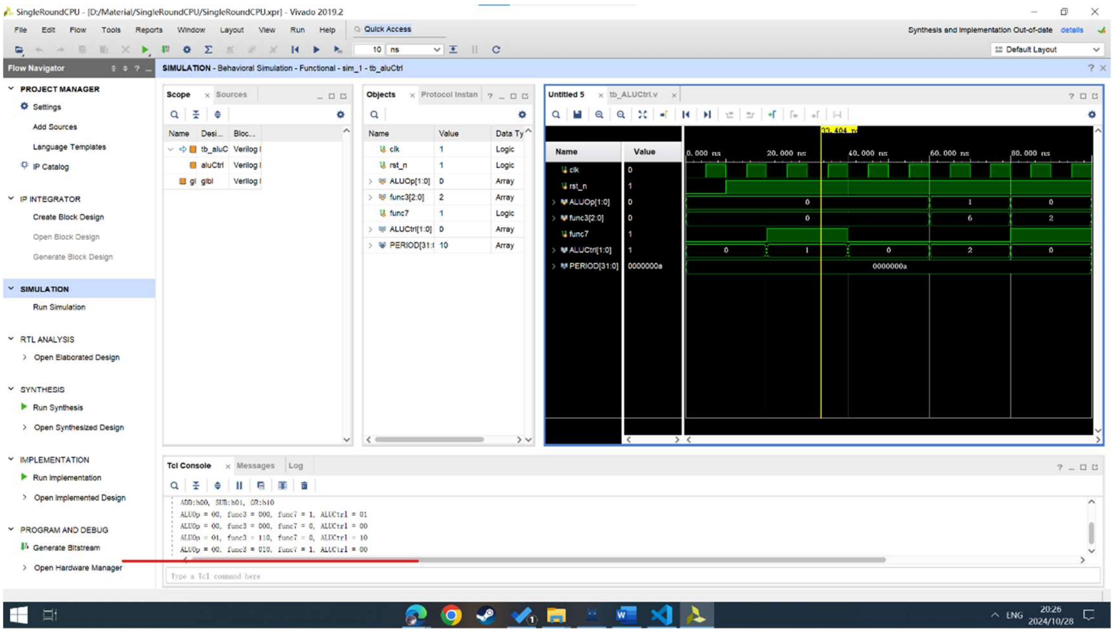


Figure 3 ALU 控制单元（ALUCtrl）

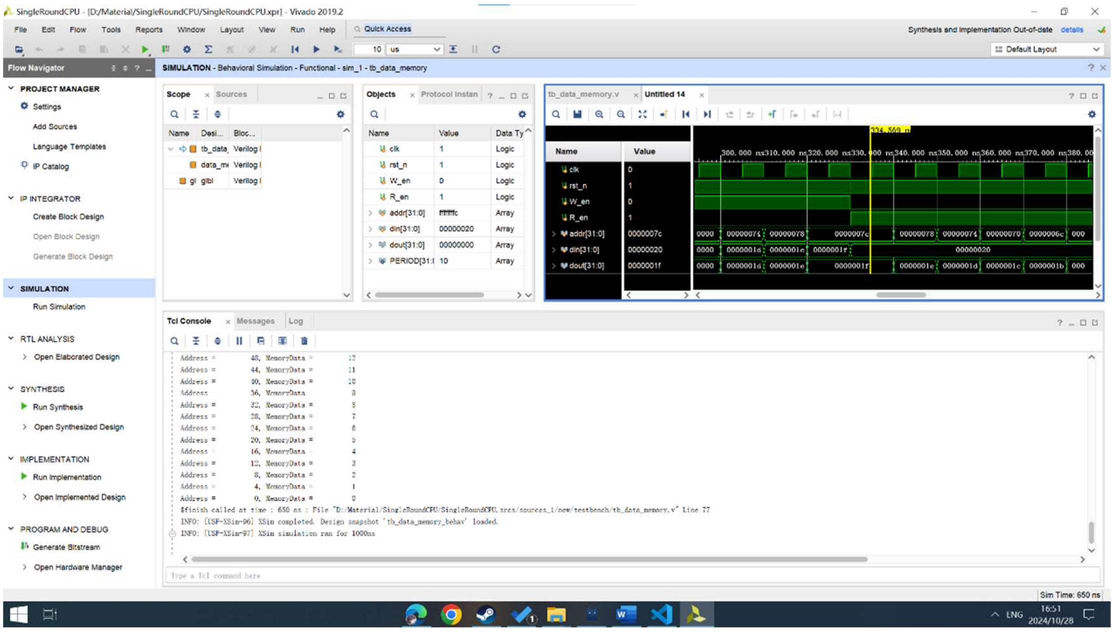


Figure 4 数据存储器（DataMemory）

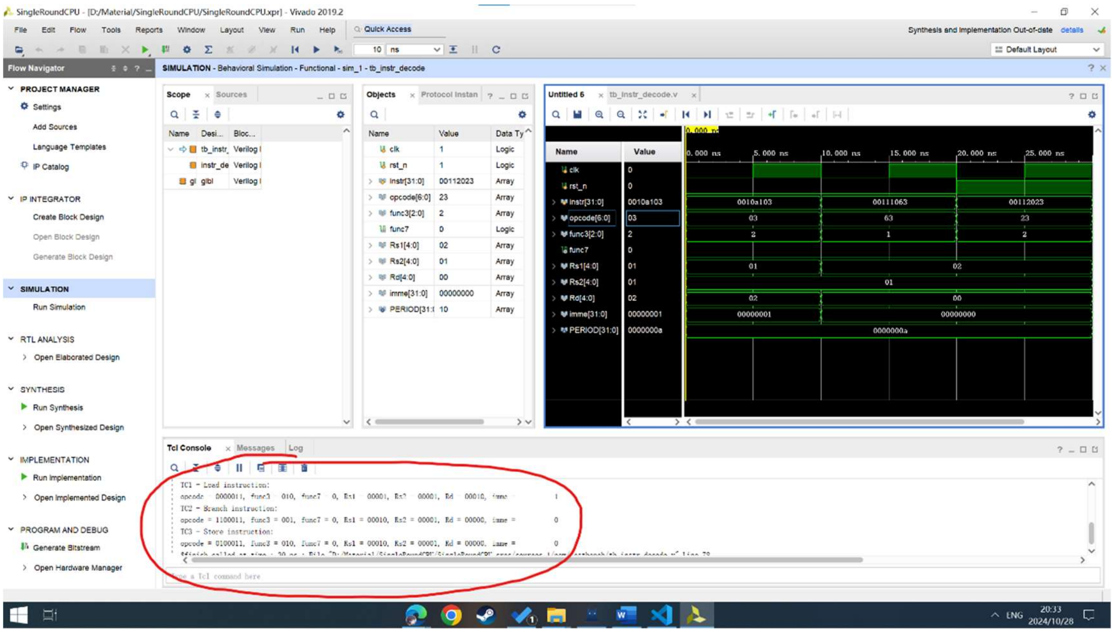


Figure 5 **指令译码器**（instr_decode）

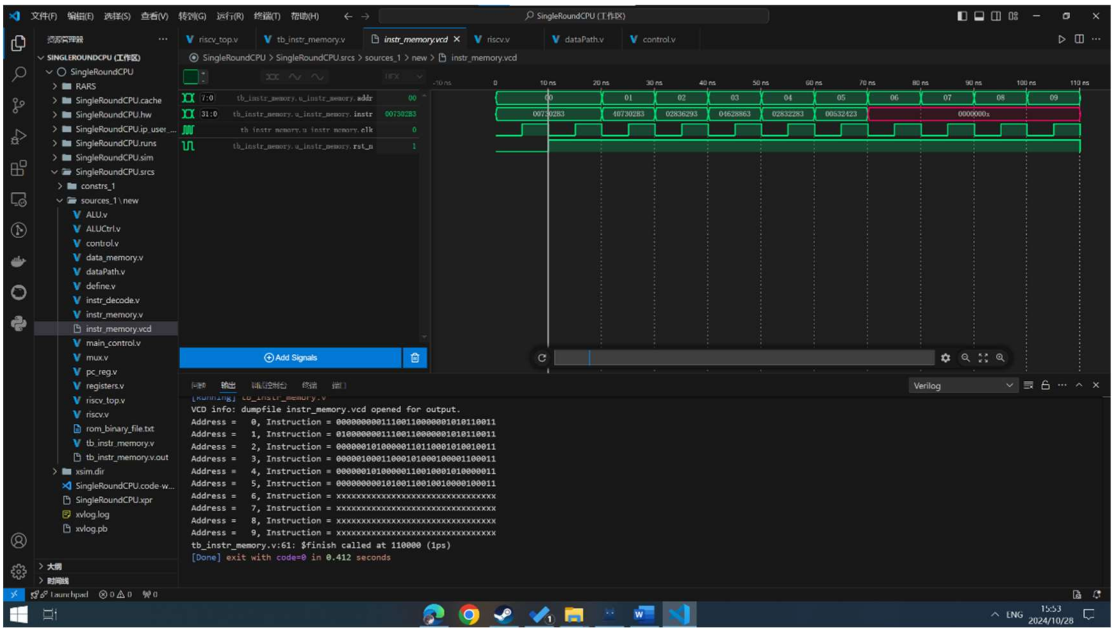


Figure 6 **指令存储器**（instr_memory）

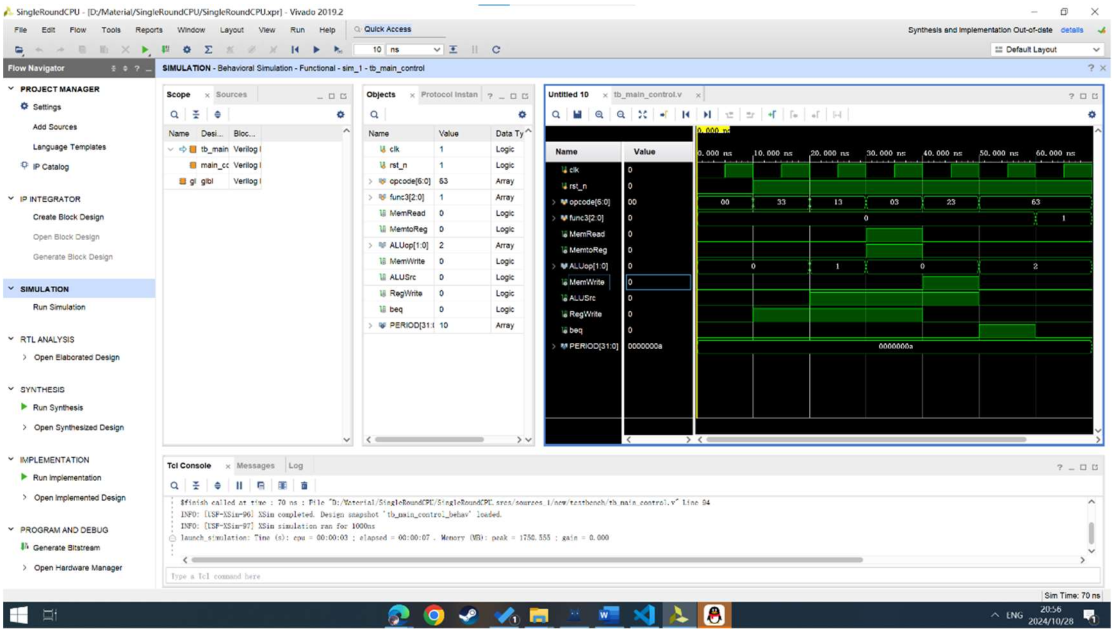


Figure 7 **主控制单元（main_control）**

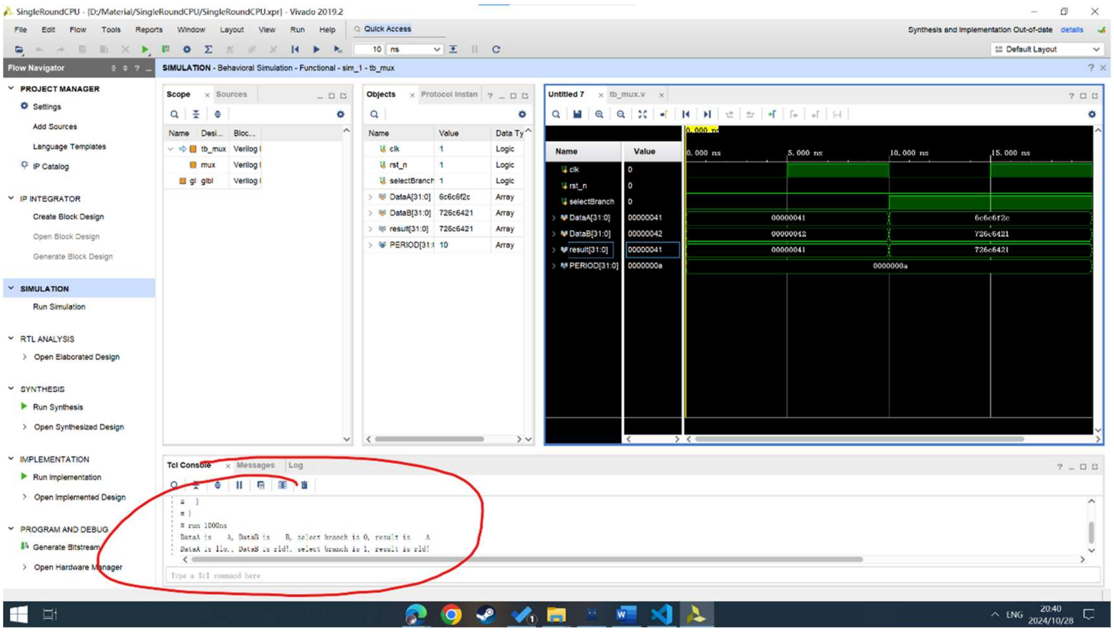


Figure 8 **二路选择器（Mux）**

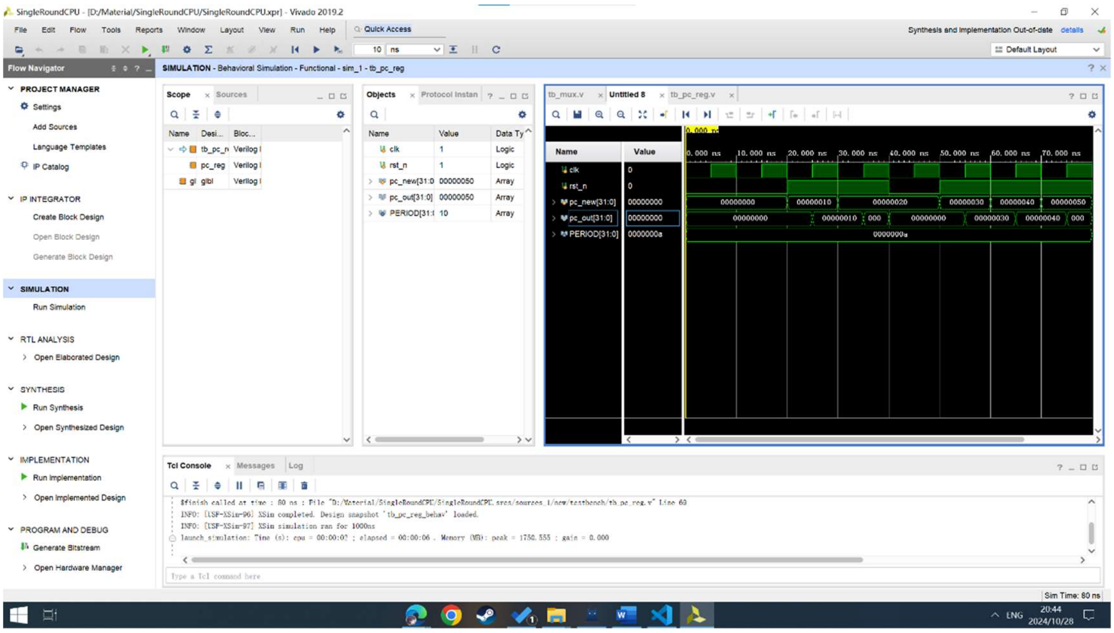


Figure 9 **程序计数器**（pc_reg）

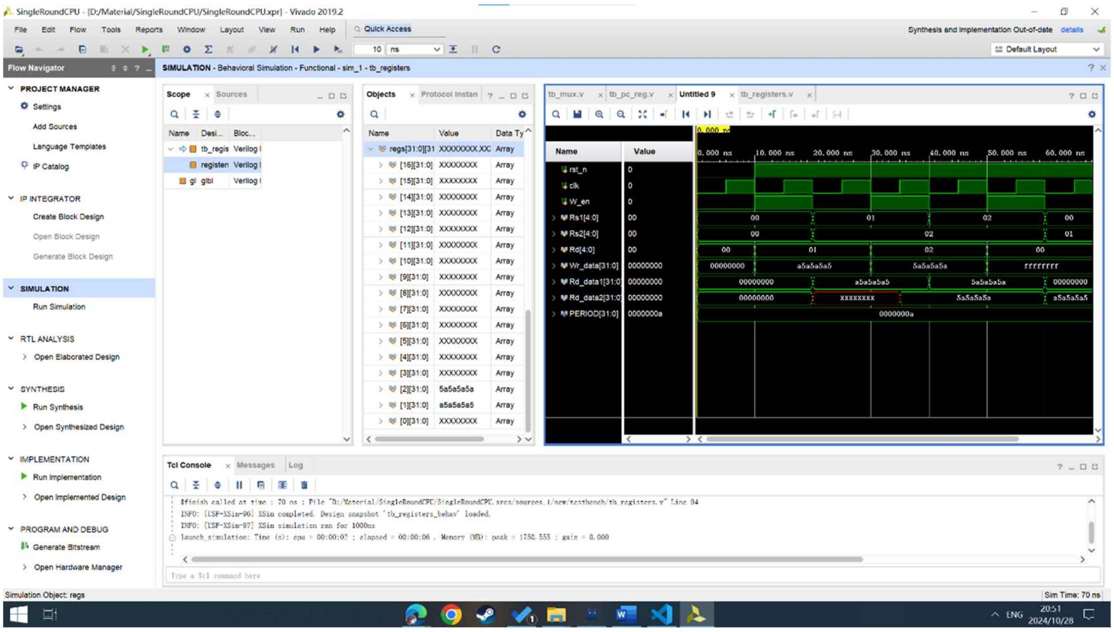


Figure 10 **寄存器堆**（registers）

3. 根据数据通路设计结果，填充并扩展表格 2。每个控制信号占一行，每条指令占 1 列。

表格 2 CPU 模型控制信号列表

操作码 op	**011 0011**	**011 0011**	**001 0011**	**110 0011**	**110 0011**
信号/指令	Add	Sub	ORI	BEQ	BGE
Branch	0	0	0	1	1
MemWrite	0	0	0	0	0
MemtoReg	0	0	0	0	0
MemRead	0	0	0	0	0

ALUSrc	0	0	1	0	0
ALUOp[1:0]	00	00	01	10	10
RegWrite	1	1	1	0	0
ALUctrl[1:0]	00(add)	01(sub)	10(or)	01(sub)	01(sub)
操作码 op	**000 0011**	**010 0011**	**001 0011**	**001 0111**	
信号/指令	LW	SW	Addi	AUIPC	
Branch	0	0	0	0	
MemWrite	0	1	0	0	
MemtoReg	1	0	0	0	
MemRead	1	0	0	0	
ALUSrc	1	1	1	1	
ALUOp[1:0]	00	00	01	11	
RegWrite	1	0	1	1	
ALUctrl[1:0]	00(add)	00(add)	00(add)	00(add)	

4. 根据表格 2 控制信号完成控制器设计，可以采用微程序方式或者组合逻辑硬链接方式。前者在实验报告中提供微指令的格式，并提供控制存储器的内容文件。在实验报告中提供逻辑表达式。

B_type	$7'b1100011$
$load$	$7'b0000011$
$store$	$7'b0100011$
I_type	$7'b0010011$
R_type	$7'b0110011$
$auipc$	$7'b0010111$
定义如上变量	
$MemRead$	$\sim\wedge(opcode \wedge `load)$
$MemWrite$	$\sim\wedge(opcode \wedge `store)$
$RegWrite$	$\sim\wedge(opcode \wedge `load) \vert \sim\wedge(opcode \wedge `I\_type) \vert$ $\sim\wedge(opcode \wedge `R\_type) \vert \sim\wedge(opcode \wedge `auipc)$
$ALUSrc$	$\sim\wedge(opcode \wedge `load) \vert \sim\wedge(opcode \wedge `store) \vert$ $\sim\wedge(opcode \wedge `I\_type) \vert \sim\wedge(opcode \wedge `auipc)$
$MemtoReg$	$\sim\wedge(opcode \wedge `load)$
$ALUop[1]$	$\sim\wedge(opcode \wedge `B\_type) \vert \sim\wedge(opcode \wedge `auipc)$
$ALUop[0]$	$\sim\wedge(opcode \wedge `I\_type) \vert \sim\wedge(opcode \wedge `auipc)$
beq	$\sim\wedge(opcode \wedge `B\_type)\ \&\ \sim\wedge(func3 \wedge 3'b000)$
bge	$\sim\wedge(opcode \wedge `B\_type)\ \&\ \sim\wedge(func3 \wedge 3'b101)$

5. 用 RISC-V 汇编编写测试程序：实现对 5 个整数的排序。可以采用直接输入，或者将数据初始化到数据存储器的实现方式。在实验报告中画出程序流程图，并且将源代码写入表格 3。用 RARS 仿真器进行仿真调试，并且保留机器码和调试结果。根据 RARS 的内容填写并且补充表格 4。

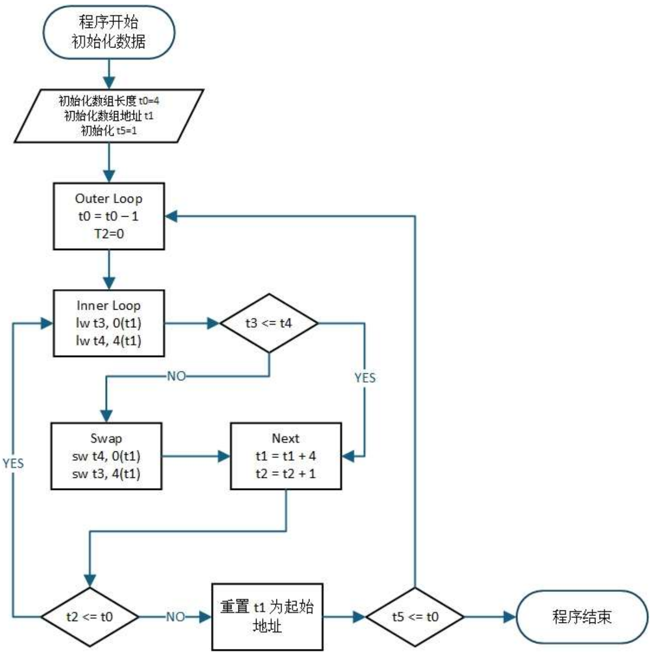


Figure 11 rars **程序流程图**

表格 3

RISC-V 汇编源代码粘贴处：

```
 .data
data_start: .word 5, 3, 8, 1, 7  # 初始化 5 个待排序的整数

.text
main:
    li t0, 4               # t0 = 数组长度
    la t1, data_start      # t1 = 数组起始地址
    li t5, 1               # 将立即数 1 存入 t5 寄存器

outer_loop:
    addi t0, t0, -1        # 每轮排序后长度减少
    li t2, 0               # t2 = 内层循环计数

inner_loop:
    lw t3, 0(t1)           # 加载当前元素
    lw t4, 4(t1)           # 加载下一个元素
    ble t3, t4, no_swap    # 若前者 <= 后者，跳过交换

    # 交换 t3 和 t4
    sw t4, 0(t1)
    sw t3, 4(t1)

no_swap:
    addi t1, t1, 4         # 下一个元素
```

```
    addi t2, t2, 1
    ble t2, t0, inner_loop  # 若未达内层循环限制，继续

    la t1, data_start     # 重置数组地址
    ble t5, t0, outer_loop # 若仍需排序，继续

end:
    # 排序完成
```

表格 4 指令与对应的机器码

序号	指令格式	具体指令	funct7 (31-25)	rs2 (24-20)	rs1 (19-15)	funct3 (14-12)	Rd (11-7)	Op (6-0)
1	addi rd, rs, imm	addi x5, x0, 4	0000000	00100	00000	000	00101	0010011
2	auipc rd, imm	auipc x6, 0x0000fc10	0000111	11100	00100	000	00110	0010111
3	addi rd, rs, imm	addi x6, x6, 0xfffffffc	1111111	11100	01100	000	00110	0010011
4	addi rd, rs, imm	addi x30, x0, 1	0000000	00001	00000	000	11110	0010011
5	addi rd, rs, imm	addi x5, x5, -1	1111111	11111	01010	000	00101	0010011
6	addi rd, rs, imm	addi x7, x0, 0	0000000	00000	00000	000	00111	0010011
7	lw rd, offset(rs1)	lw x28, 0(x6)	0000000	00000	01100	010	11100	0000011
8	lw rd, offset(rs1)	lw x29, 4(x6)	0000000	00100	01100	010	11101	0000011
9	addi rd, rs, imm	addi x29, x28, 0x0000000c	0000000	11100	11011	101	01100	1100011
10	sw rs2, offset(rs1)	sw x29, 0(x6)	0000000	11101	01100	010	00000	0100011
11	sw rs2, offset(rs1)	sw x28, 4(x6)	0000000	11100	01100	010	00100	0100011
12	addi rd, rs, imm	addi x6, x6, 4	0000000	00100	01100	000	00110	0010011
13	addi rd, rs, imm	addi x7, x7, 1	0000000	00001	01110	000	00111	0010011
14	addi rd, rs, imm	addi x5, x7, 0xffffffe4	1111111	00111	01011	101	00101	1100011
15	auipc rd, imm	auipc x6, 0x0000fc10	0000111	11100	00100	000	00110	0010111
16	addi rd, rs, imm	addi x6, x6, 0xffffffc8	1111110	01000	01100	000	00110	0010011
17	bge rs1, rs2, offset	bge x5, x30, 0xffffffd0	1111110	11110	01011	101	10001	1100011

6. 将表格 4 的机器码写入指令存储器中，启动 Vivado 程序执行仿真，在实验报告中记录每条指令的仿真波形以及结果分析。对比执行每条指令后相关寄存器或者存储单元中保存的值和 RARS 软件执行的结果，填充表格 5，验证执行结果的正确性。

表格 5 Vivado 仿真结果与 RARS 仿真结果比较

指令序列	单周期处理器仿真结果	RARS 仿真结果
addi x5, x0, 4	Register[5]Change From 4 To 4	Register[5]Change From 4 To 4
auipc x6, 0x0000fc10	Register[6]Change From x To 4	Register[6]Change From x To 4
addi x6, x6, 0xfffffffc	Register[6]Change From 4 To 0	Register[6]Change From 4 To 0
addi x30, x0, 1	Register[30]Change From x To 1	Register[30]Change From x To 1
addi x5, x5, -1	Register[5]Change From 4 To 3	Register[5]Change From 4 To 3
addi x7, x0, 0	Register[7]Change From x To 0	Register[7]Change From x To 0
lw x28, 0(x6)	Register[28]Change From x To 5	Register[28]Change From x To 5
lw x29, 4(x6)	Register[29]Change From x To 3	Register[29]Change From x To 3
addi x29, x28, 0x0000000c	Register[12]Change From x To 4294967294	Register[12]Change From x To 4294967294
sw x29, 0(x6)	DataMemory[0]Change From 5 To 3	DataMemory[0]Change From 5 To 3
sw x28, 4(x6)	DataMemory[4]Change From 3 To 5	DataMemory[4]Change From 3 To 5
addi x6, x6, 4	Register[6]Change From 0 To 4	Register[6]Change From 0 To 4
addi x7, x7, 1	Register[7]Change From 0 To 1	Register[7]Change From 0 To 1
addi x5, x7, 0xffffffe4	Register[5]Change From 3 To 2	Register[5]Change From 3 To 2
lw x28, 0(x6)	Register[28]Change From 5 To 5	Register[28]Change From 5 To 5
lw x29, 4(x6)	Register[29]Change From 3 To 8	Register[29]Change From 3 To 8
addi x29, x28, 0x0000000c	Register[12]Change From x To 3	Register[12]Change From x To 3
addi x6, x6, 4	Register[6]Change From 4 To 8	Register[6]Change From 4 To 8
addi x7, x7, 1	Register[7]Change From 1 To 2	Register[7]Change From 1 To 2

addi x5, x7, 0xffffffe4	Register[5]Change From 3 To 1	Register[5]Change From 3 To 1
lw x28, 0(x6)	Register[28]Change From 5 To 8	Register[28]Change From 5 To 8
lw x29, 4(x6)	Register[29]Change From 8 To 1	Register[29]Change From 8 To 1
addi x29, x28, 0x0000000c	Register[12]Change From x To 4294967289	Register[12]Change From x To 4294967289
sw x29, 0(x6)	DataMemory[8]Change From 8 To 1	DataMemory[8]Change From 8 To 1
sw x28, 4(x6)	DataMemory[12]Change From 1 To 8	DataMemory[12]Change From 1 To 8
addi x6, x6, 4	Register[6]Change From 8 To 12	Register[6]Change From 8 To 12
addi x7, x7, 1	Register[7]Change From 2 To 3	Register[7]Change From 2 To 3
addi x5, x7, 0xffffffe4	Register[5]Change From 3 To 0	Register[5]Change From 3 To 0
lw x28, 0(x6)	Register[28]Change From 8 To 8	Register[28]Change From 8 To 8
lw x29, 4(x6)	Register[29]Change From 1 To 7	Register[29]Change From 1 To 7
addi x29, x28, 0x0000000c	Register[12]Change From x To 4294967295	Register[12]Change From x To 4294967295
sw x29, 0(x6)	DataMemory[12]Change From 8 To 7	DataMemory[12]Change From 8 To 7
sw x28, 4(x6)	DataMemory[16]Change From 7 To 8	DataMemory[16]Change From 7 To 8
addi x6, x6, 4	Register[6]Change From 12 To 16	Register[6]Change From 12 To 16
addi x7, x7, 1	Register[7]Change From 3 To 4	Register[7]Change From 3 To 4
addi x5, x7, 0xffffffe4	Register[5]Change From 3 To 4294967295	Register[5]Change From 3 To 4294967295
auipc x6, 0x0000fc10	Register[6]Change From 16 To 56	Register[6]Change From 16 To 56
addi x6, x6, 0xffffffc8	Register[6]Change From 56 To 0	Register[6]Change From 56 To 0
bge x5, x30, 0xffffffd0	Register[17]Change From x To 2	Register[17]Change From x To 2
addi x5, x5, -1	Register[5]Change From 3 To 2	Register[5]Change From 3 To 2

addi x7, x0, 0	Register[7]Change From 4 To 0	Register[7]Change From 4 To 0
lw x28, 0(x6)	Register[28]Change From 8 To 3	Register[28]Change From 8 To 3
lw x29, 4(x6)	Register[29]Change From 7 To 5	Register[29]Change From 7 To 5
addi x29, x28, 0x0000000c	Register[12]Change From x To 2	Register[12]Change From x To 2
addi x6, x6, 4	Register[6]Change From 0 To 4	Register[6]Change From 0 To 4
addi x7, x7, 1	Register[7]Change From 0 To 1	Register[7]Change From 0 To 1
addi x5, x7, 0xffffffe4	Register[5]Change From 2 To 1	Register[5]Change From 2 To 1
lw x28, 0(x6)	Register[28]Change From 3 To 5	Register[28]Change From 3 To 5
lw x29, 4(x6)	Register[29]Change From 5 To 1	Register[29]Change From 5 To 1
addi x29, x28, 0x0000000c	Register[12]Change From x To 4294967292	Register[12]Change From x To 4294967292
sw x29, 0(x6)	DataMemory[4]Change From 5 To 1	DataMemory[4]Change From 5 To 1
sw x28, 4(x6)	DataMemory[8]Change From 1 To 5	DataMemory[8]Change From 1 To 5
addi x6, x6, 4	Register[6]Change From 4 To 8	Register[6]Change From 4 To 8
addi x7, x7, 1	Register[7]Change From 1 To 2	Register[7]Change From 1 To 2
addi x5, x7, 0xffffffe4	Register[5]Change From 2 To 0	Register[5]Change From 2 To 0
lw x28, 0(x6)	Register[28]Change From 5 To 5	Register[28]Change From 5 To 5
lw x29, 4(x6)	Register[29]Change From 1 To 7	Register[29]Change From 1 To 7
addi x29, x28, 0x0000000c	Register[12]Change From x To 2	Register[12]Change From x To 2
addi x6, x6, 4	Register[6]Change From 8 To 12	Register[6]Change From 8 To 12
addi x7, x7, 1	Register[7]Change From 2 To 3	Register[7]Change From 2 To 3
addi x5, x7, 0xffffffe4	Register[5]Change From 2 To 4294967295	Register[5]Change From 2 To 4294967295

auipc x6, 0x0000fc10	Register[6]Change From 12 To 56	Register[6]Change From 12 To 56
addi x6, x6, 0xffffffc8	Register[6]Change From 56 To 0	Register[6]Change From 56 To 0
bge x5, x30, 0xffffffd0	Register[17]Change From x To 1	Register[17]Change From x To 1
addi x5, x5, -1	Register[5]Change From 2 To 1	Register[5]Change From 2 To 1
addi x7, x0, 0	Register[7]Change From 3 To 0	Register[7]Change From 3 To 0
lw x28, 0(x6)	Register[28]Change From 5 To 3	Register[28]Change From 5 To 3
lw x29, 4(x6)	Register[29]Change From 7 To 1	Register[29]Change From 7 To 1
addi x29, x28, 0x0000000c	Register[12]Change From x To 4294967294	Register[12]Change From x To 4294967294
sw x29, 0(x6)	DataMemory[0]Change From 3 To 1	DataMemory[0]Change From 3 To 1
sw x28, 4(x6)	DataMemory[4]Change From 1 To 3	DataMemory[4]Change From 1 To 3
addi x6, x6, 4	Register[6]Change From 0 To 4	Register[6]Change From 0 To 4
addi x7, x7, 1	Register[7]Change From 0 To 1	Register[7]Change From 0 To 1
addi x5, x7, 0xffffffe4	Register[5]Change From 1 To 0	Register[5]Change From 1 To 0
lw x28, 0(x6)	Register[28]Change From 3 To 3	Register[28]Change From 3 To 3
lw x29, 4(x6)	Register[29]Change From 1 To 5	Register[29]Change From 1 To 5
addi x29, x28, 0x0000000c	Register[12]Change From x To 2	Register[12]Change From x To 2
addi x6, x6, 4	Register[6]Change From 4 To 8	Register[6]Change From 4 To 8
addi x7, x7, 1	Register[7]Change From 1 To 2	Register[7]Change From 1 To 2
addi x5, x7, 0xffffffe4	Register[5]Change From 1 To 4294967295	Register[5]Change From 1 To 4294967295
auipc x6, 0x0000fc10	Register[6]Change From 8 To 56	Register[6]Change From 8 To 56
addi x6, x6, 0xffffffc8	Register[6]Change From 56 To 0	Register[6]Change From 56 To 0

bge x5, x30, 0xffffffd0	Register[17]Change From x To 0	Register[17]Change From x To 0
addi x5, x5, -1	Register[5]Change From 1 To 0	Register[5]Change From 1 To 0
addi x7, x0, 0	Register[7]Change From 2 To 0	Register[7]Change From 2 To 0
lw x28, 0(x6)	Register[28]Change From 3 To 1	Register[28]Change From 3 To 1
lw x29, 4(x6)	Register[29]Change From 5 To 3	Register[29]Change From 5 To 3
addi x29, x28, 0x0000000c	Register[12]Change From x To 2	Register[12]Change From x To 2
addi x6, x6, 4	Register[6]Change From 0 To 4	Register[6]Change From 0 To 4
addi x7, x7, 1	Register[7]Change From 0 To 1	Register[7]Change From 0 To 1
addi x5, x7, 0xffffffe4	Register[5]Change From 0 To 4294967295	Register[5]Change From 0 To 4294967295
auipc x6, 0x0000fc10	Register[6]Change From 4 To 56	Register[6]Change From 4 To 56
addi x6, x6, 0xffffffc8	Register[6]Change From 56 To 0	Register[6]Change From 56 To 0
bge x5, x30, 0xffffffd0	Register[17]Change From x To 4294967295	Register[17]Change From x To 4294967295

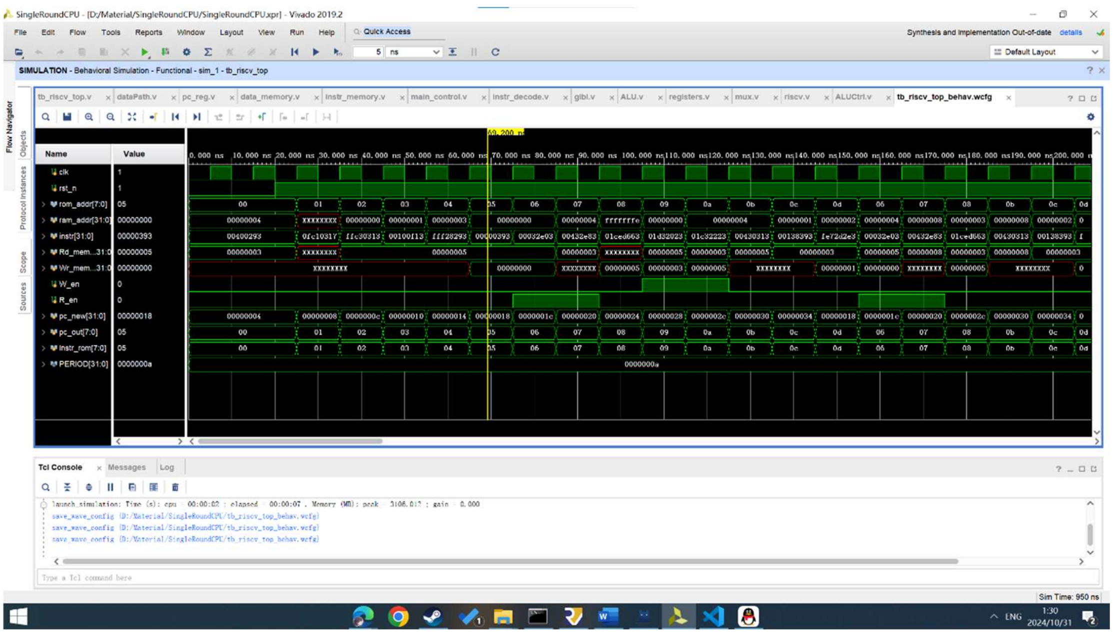
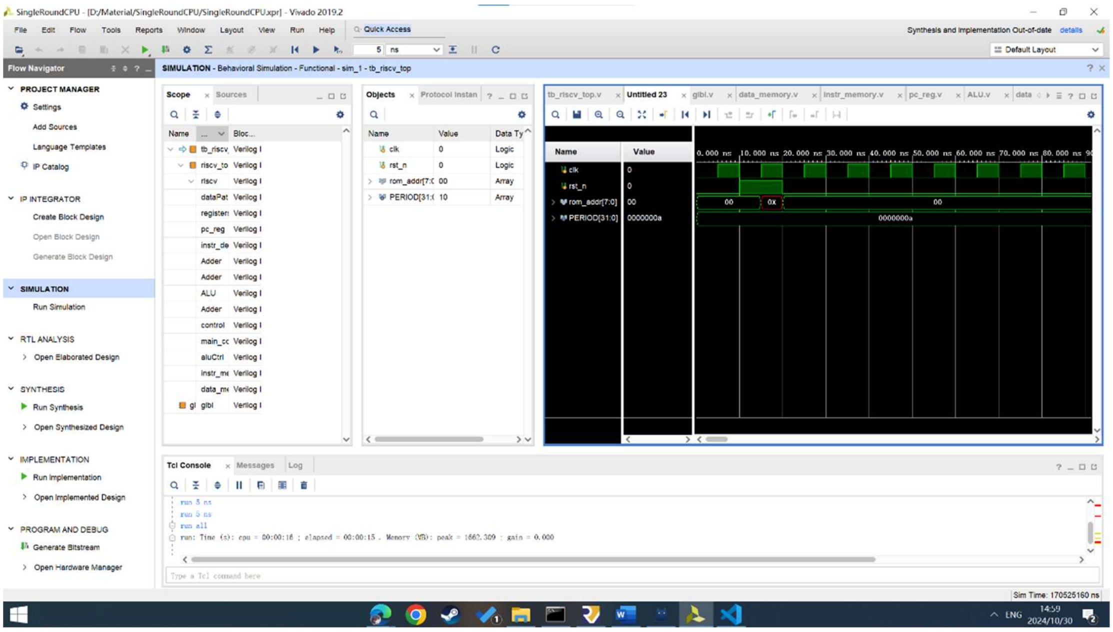


Figure 12 实验流程波形图

5. 实验结果与分析

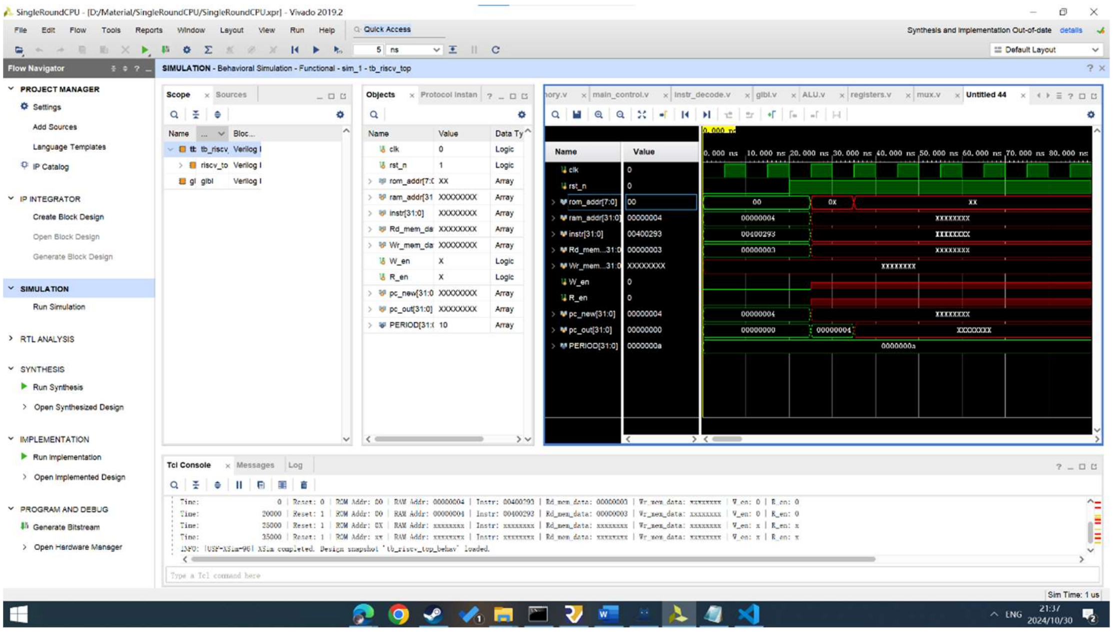
实验过程中，我在子模块设计和仿真过程中都没有遇到什么问题。但是，在总体仿真的时候遇到了几个 BUG。见如下：



图中一条指令执行后(rom_addr)，下一个周期时没有变化。

错误分析：后续排查出原因得知是 rst_n 信号没有释放，导致译码器不断重置。

此外是另一个 BUG，如下：



图中一条指令执行完后，下一个周期新的指令 pc_out 地址已经生成，但是 rom_addr 却赋值错误，导致后续无法找到指令。这是个可稳定复现的问题。

错误分析：说来话长，在 tb_riscv_top 中的这一行

```
wire   [7:0]  rom_addr                        ;
```

即指令寄存器的读取地址，我将其初始化为 0。此时的我还没意识到这个操作导致的严重问题，仿真时，程序在执行第一条指令地址 0x00 后，下一指令地址已生成为 0x04，但是图中得到的 rom_addr 却为 0x0X，也就是为指令赋值时出现了错误。后续我意识到，wire 类型相当于一根导线，初始值应该为高组态 Z，不应该人为初始化，而应该采用 reg+assign 的方式初始化。

6. 心得体会

通过本次实验，我学习到 RISCV 指令格式与编码，掌握 RISCV 汇编转换成机器代码的方法，并初步学会 verilog 和 vivado 的使用，掌握基本测试与调试方法。此外，学会单周期 CPU 的数据通路与控制器设计方法，理解并掌握了单周期 CPU 的设计和仿真实验流程。在此过程中，我深刻理解 CPU 处理指令的流程，并提高了我的代码能力和工程能力。

7. 参考资料

1. 王党辉等译，[美] David A. Patterson, John L. Hennessy 计算机组成与设计-硬件/软件接口（原书第五版），北京：机械工业出版社，2016 年
2. AMD 官网 Vivado 教学视频
3. 钉钉指导视频