《数字逻辑与计算机组成》实验报告六. 单周期 CPU 设计与测试

1 实验

1.1 控制器设计实验

1.1.1 实验内容

RV32I 指令集中包含 47 条基础指令，涵盖了整数运算、存储器访问、控制转移和系统控制几个大类。本次实验中需要实现除了系统控制类的 10 条指令外的 37 条指令分为整数运算指令 21 条)、控制转移指令 8 条)和存储器访问指令 8 条)。

1.1.2 实验整体方案设计

按照控制信号表，分别根据不同的指令码得出控制信号.
控制信号如图。本实验我采用了最简单的方式，先用比较器判断是哪类指令，再按照位数每个位数没用卡诺图化简直接连最小项。

表 6.1 RV32I 指令控制信号列表

指令	类型	op[6:0]	func3	func7[5]	ExtOp	RegWr	ALUASrc	ALUBSrc	ALUctr
lui	U	0110111	×	×	001	1	×	10	1111
auipc	U	0010111	×	×	001	1	1	10	0000
addi	I	0010011	000	×	000	1	0	10	0000
slti	I	0010011	010	×	000	1	0	10	0010
sltiu	I	0010011	011	×	000	1	0	10	0011
xori	I	0010011	100	×	000	1	0	10	0100
ori	I	0010011	110	×	000	1	0	10	0110
andi	I	0010011	111	×	000	1	0	10	0111
slli	I	0010011	001	0	000	1	0	10	0001
srli	I	0010011	101	0	000	1	0	10	0101
srai	I	0010011	101	1	000	1	0	10	1101
add	R	0110011	000	0	×	1	0	00	0000
sub	R	0110011	000	1	×	1	0	00	1000
sll	R	0110011	001	0	×	1	0	00	0001
slt	R	0110011	010	0	×	1	0	00	0010
sltu	R	0110011	011	0	×	1	0	00	0011

图 1: 控制信号 1

xor	R	0110011	100	0	×	1	0	00	0100
srl	R	0110011	101	0	×	1	0	00	0101
sra	R	0110011	101	1	×	1	0	00	1101
or	R	0110011	110	0	×	1	0	00	0110
and	R	0110011	111	0	×	1	0	00	0111
jal	J	1101111	×	×	100	1	1	01	0000
jalr	I	1100111	000	×	000	1	1	01	0000
beq	B	1100011	000	×	011	0	0	00	0010
bne	B	1100011	001	×	011	0	0	00	0010
blt	B	1100011	100	×	011	0	0	00	0010
bge	B	1100011	101	×	011	0	0	00	0010
bltu	B	1100011	110	×	011	0	0	00	0011
bgeu	B	1100011	111	×	011	0	0	00	0011
lb	I	0000011	000	×	000	1	0	10	0000
lh	I	0000011	001	×	000	1	0	10	0000
lw	I	0000011	010	×	000	1	0	10	0000
lbu	I	0000011	100	×	000	1	0	10	0000
lhu	I	0000011	101	×	000	1	0	10	0000
sb	S	0100011	000	×	010	0	0	10	0000
sh	S	0100011	001	×	010	0	0	10	0000
sw	S	0100011	010	×	010	0	0	10	0000

图 2: 控制信号 2

表 6.1 RV32I 指令控制信号列表（续）

指令	类型	op[6:0]	func3	func7[5]	Branch	MemtoReg	MemWr	MemOp
lui	U	0110111	×	×	000	0	0	×
auipc	U	0010111	×	×	000	0	0	×
addi	I	0010011	000	×	000	0	0	×
slti	I	0010011	010	×	000	0	0	×
sltiu	I	0010011	011	×	000	0	0	×
xori	I	0010011	100	×	000	0	0	×
ori	I	0010011	110	×	000	0	0	×
andi	I	0010011	111	×	000	0	0	×
slli	I	0010011	001	0	000	0	0	×
srli	I	0010011	101	0	000	0	0	×
srai	I	0010011	101	1	000	0	0	×
add	R	0110011	000	0	000	0	0	×
sub	R	0110011	000	1	000	0	0	×
sll	R	0110011	001	0	000	0	0	×
slt	R	0110011	010	0	000	0	0	×
sltu	R	0110011	011	0	000	0	0	×

图 3: 控制信号 3

xor	R	0110011	100	0	000	0	0	×
srl	R	0110011	101	0	000	0	0	×
sra	R	0110011	101	1	000	0	0	×
or	R	0110011	110	0	000	0	0	×
and	R	0110011	111	0	000	0	0	×
jal	J	1101111	×	×	001	0	0	×
jalr	I	1100111	000	×	010	0	0	×
beq	B	1100011	000	×	100	×	0	×
bne	B	1100011	001	×	101	×	0	×
blt	B	1100011	100	×	110	×	0	×
bge	B	1100011	101	×	111	×	0	×
bltu	B	1100011	110	×	110	×	0	×
bgeu	B	1100011	111	×	111	×	0	×
lb	I	0000011	000	×	000	1	0	101
lh	I	0000011	001	×	000	1	0	110
lw	I	0000011	010	×	000	1	0	000
lbu	I	0000011	100	×	000	1	0	001
lhu	I	0000011	101	×	000	1	0	010
sb	S	0100011	000	×	000	×	1	101
sh	S	0100011	001	×	000	×	1	110
sw	S	0100011	010	×	000	×	1	000

图 4: 控制信号 4

opcode,funct3,funct7[5]	拼
Extop	宽度为 3 位,选
RegWr	宽度为 1 位,控制是否对寄
ALUASrc	宽度为 1 位,选择 ALU 输入端 A
ALUBSrc	宽度为 2 位,选择 ALU 输入端 B 的来源。为 00 时选择 BusB ,为 01 时选择常数 4 (用
ALUCtr	:宽度为 4 位
Branch	:宽度为 3 位,说明跳转指令
MemToReg	:宽度为 1 位,选择寄存器 rd 写回数据来源
MemWr	宽度为 1 位,控制是否对数
MemOp	宽度为 3 位,

表 1: 功能表

1.1.3 实验原理图和电路图

本实验不需要原理图,只需要真值表,最小项。用 opcode,funct3,funct7[5] 来化简最小项。
最小项:

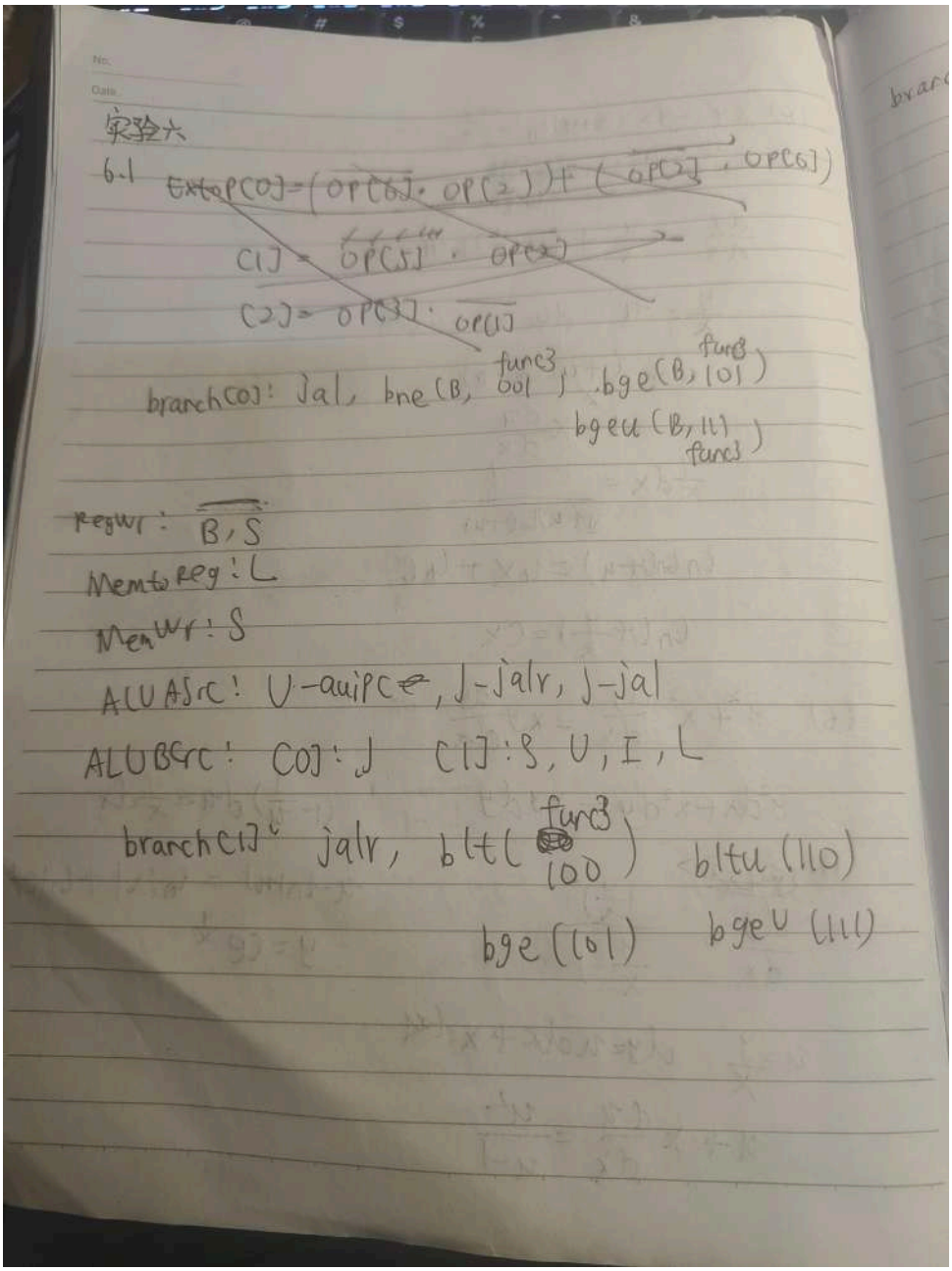
实验六
6.1
branch[0]: jal, bne(B, 001) bge(B, 101) func3
bgeu(B, 111) func3
RegWr: B, S
MemtoReg: L
MemWr: S
ALUASrc: U-auipc, J-jalr, J-jal
ALUBSrc: [0]: J [1]: S, U, I, L
branch[1]: jalr, blt(100) func3 bltu(110)
bge(101) bgeu(111)

图 5: 1

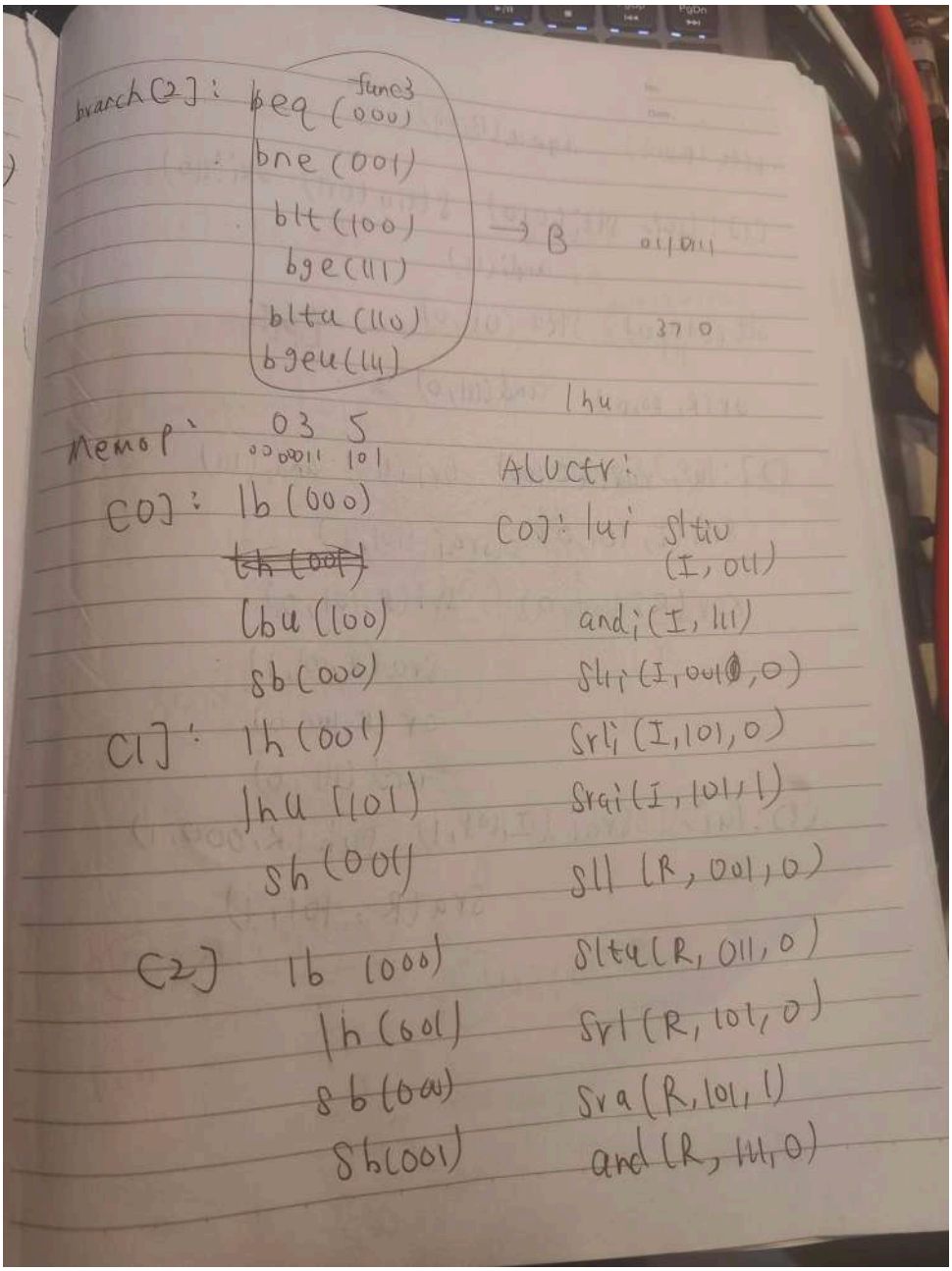
branch[2]: beq (000)
bne (001)
blt (100)
bge (111)
bltu (110)
bgeu (111)
func3
→ B
lhu
Memop: 03 5
00 0011 101
ALUctr:
[0]: lb (000)
lh (001)
lbu (100)
sb (000)
[1]: lh (001)
lhu (101)
sh (001)
[2] lb (000)
lh (001)
sb (000)
sh (001)
lui
(I, 011)
and; (I, 111)
(I, 101, 0)
(I, 101, 1)
(R, 001, 0)
(R, 011, 0)
(R, 101, 0)
(R, 101, 1)
(R, 111, 0)

图 6: 2

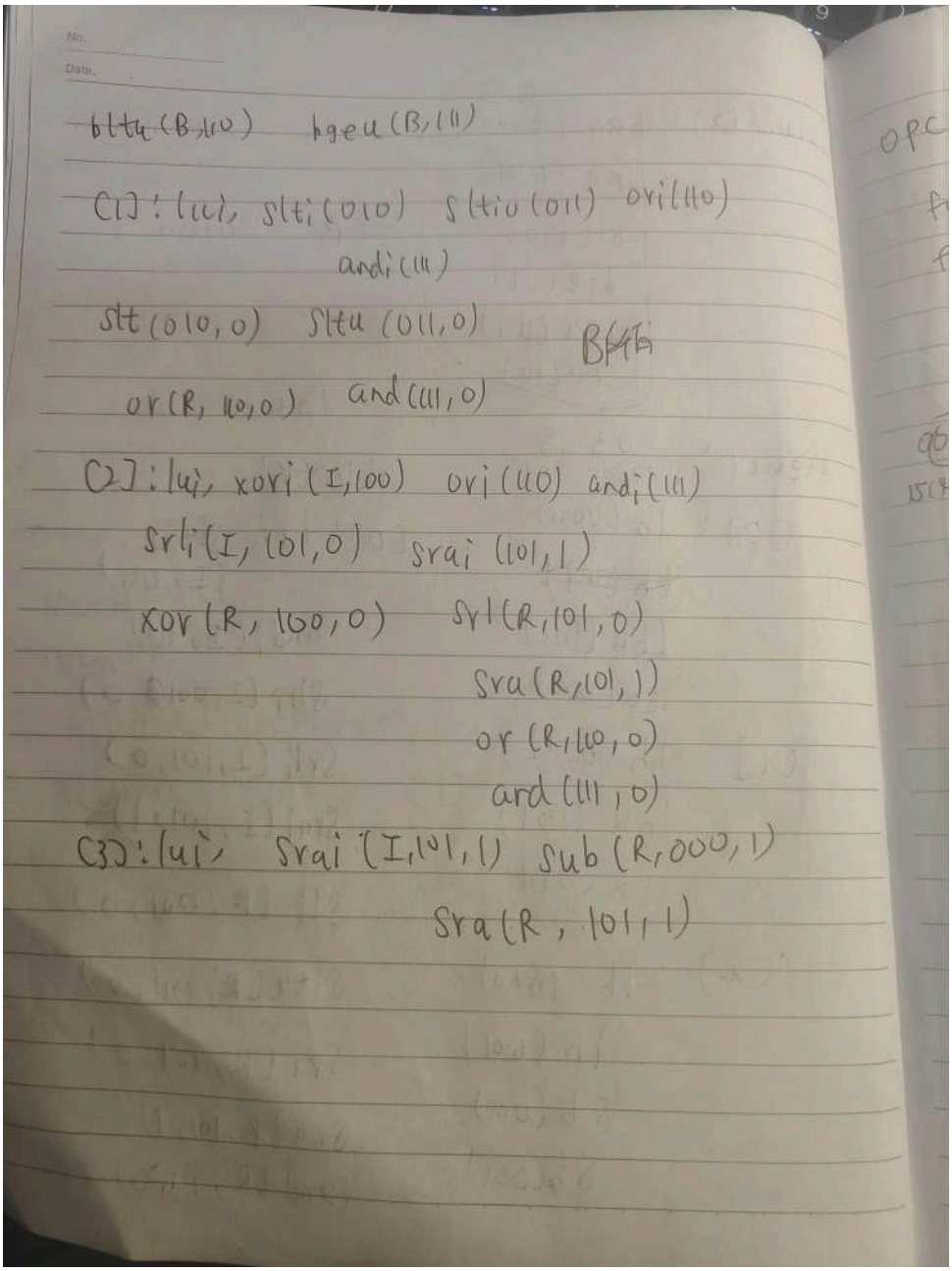
bltu(B,110) bgeu(B,111)
(1): lui, slti(010) sltiu(011) ori(110)
andi(111)
slt(010,0) sltu(011,0)
or(R,110,0) and(111,0)
(2): lui, xori(I,100) ori(110) andi(111)
srli(I,101,0) srai(101,1)
xor(R,100,0) srl(R,101,0)
sra(R,101,1)
or(R,110,0)
and(111,0)
(3): lui, srai(I,101,1) sub(R,000,1)
sra(R,101,1)

图 7: 3

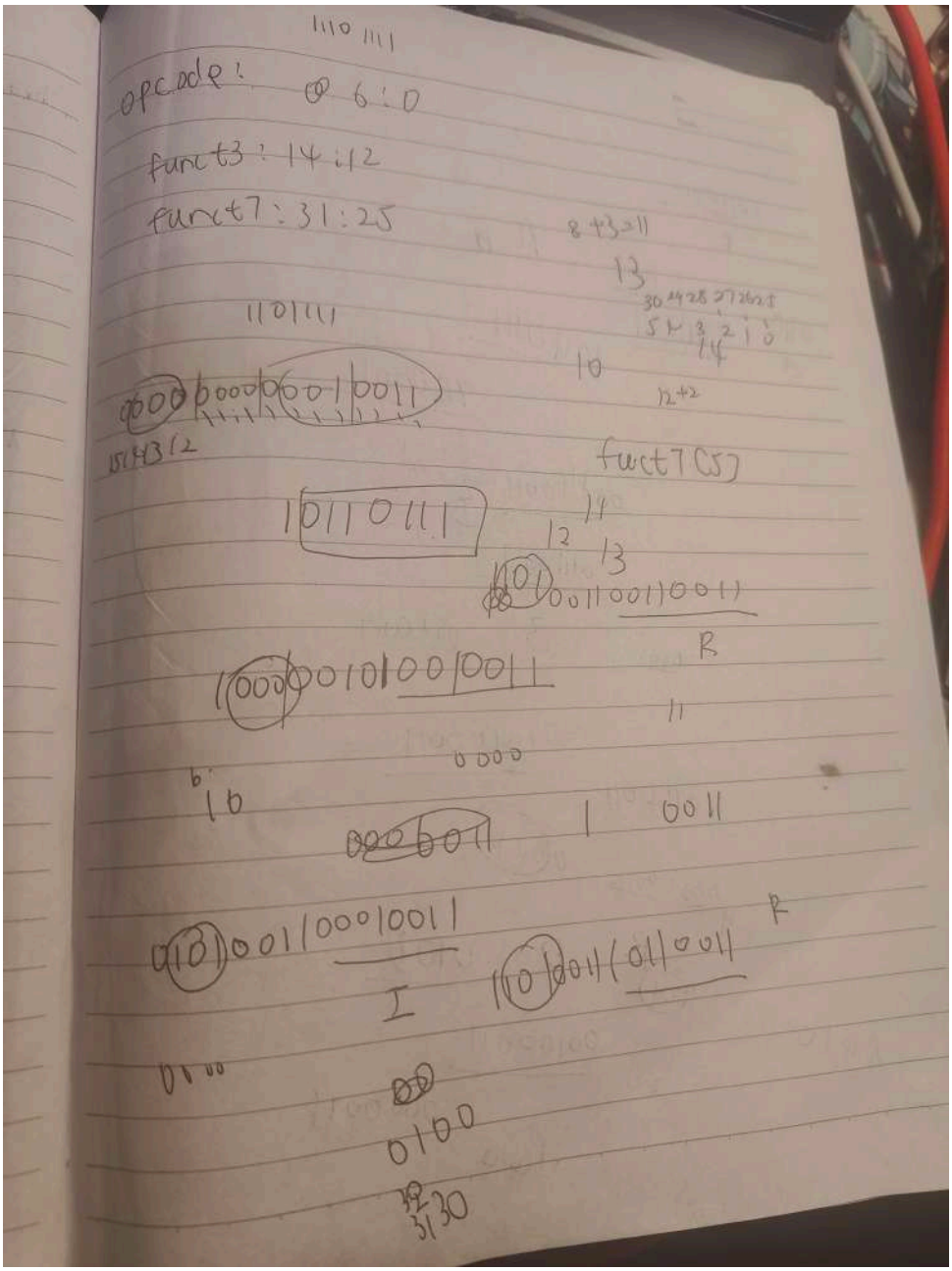

图 8: 4

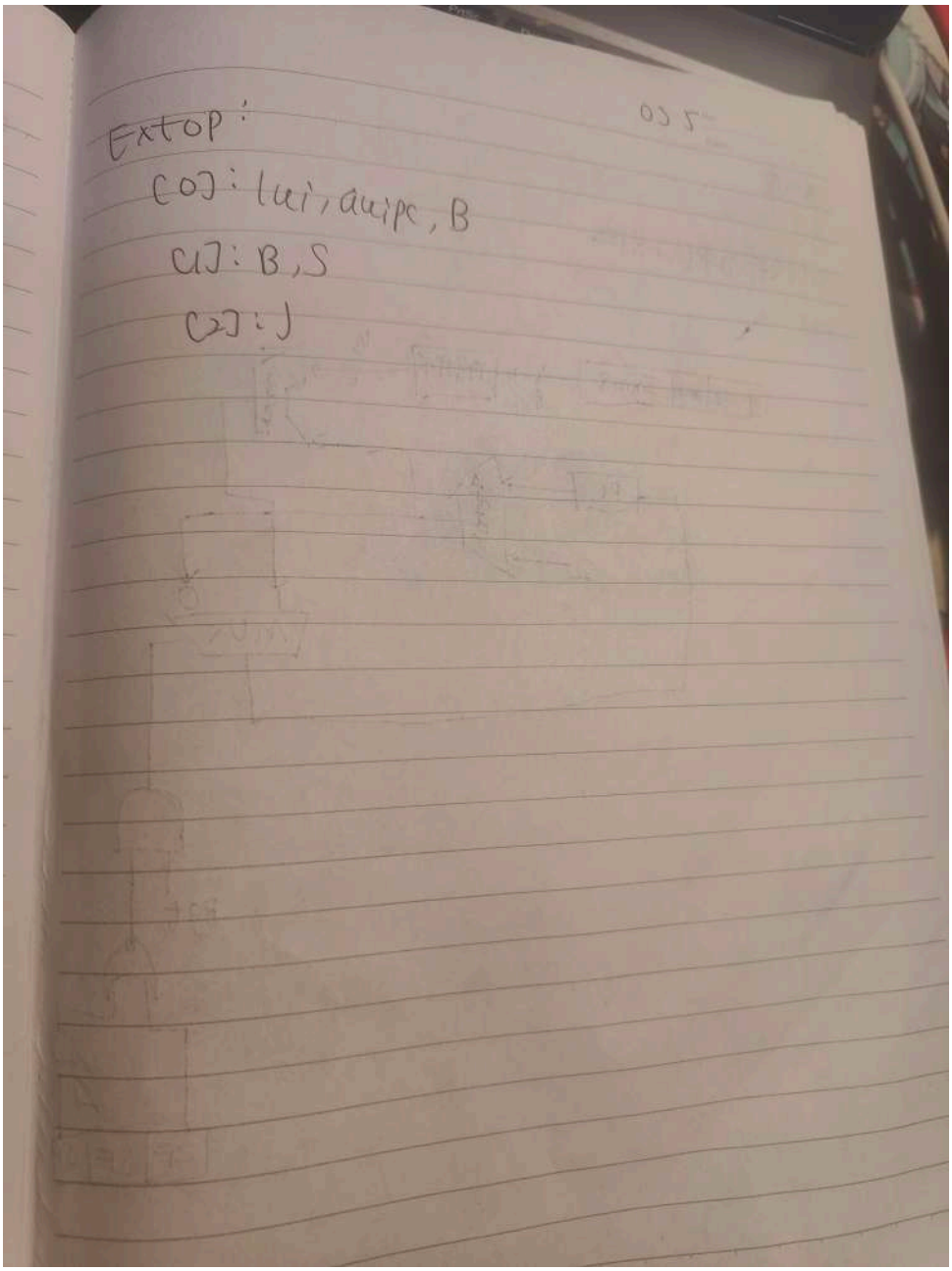

图 9: 5

电路图实现:

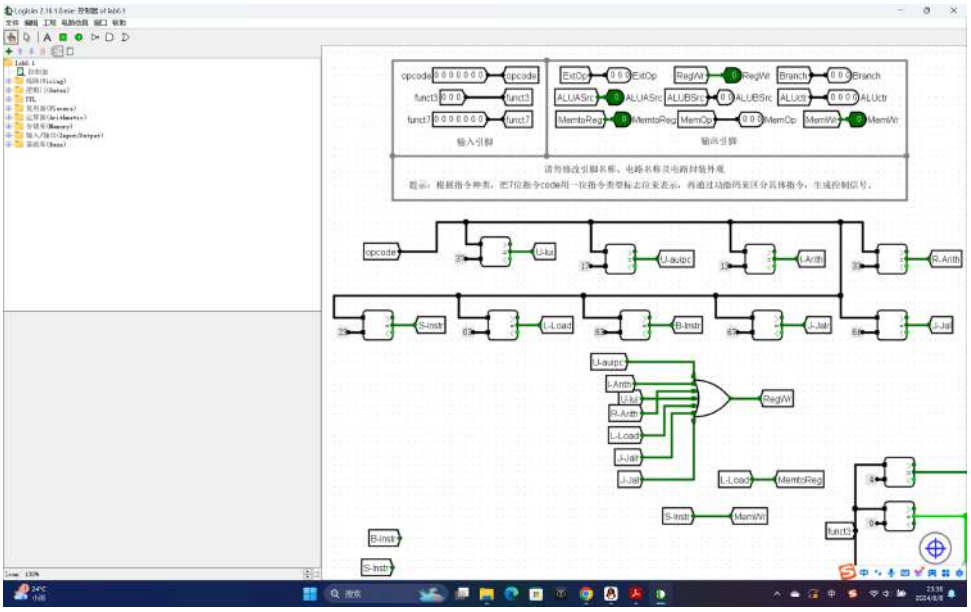

图 10: 电路图 1

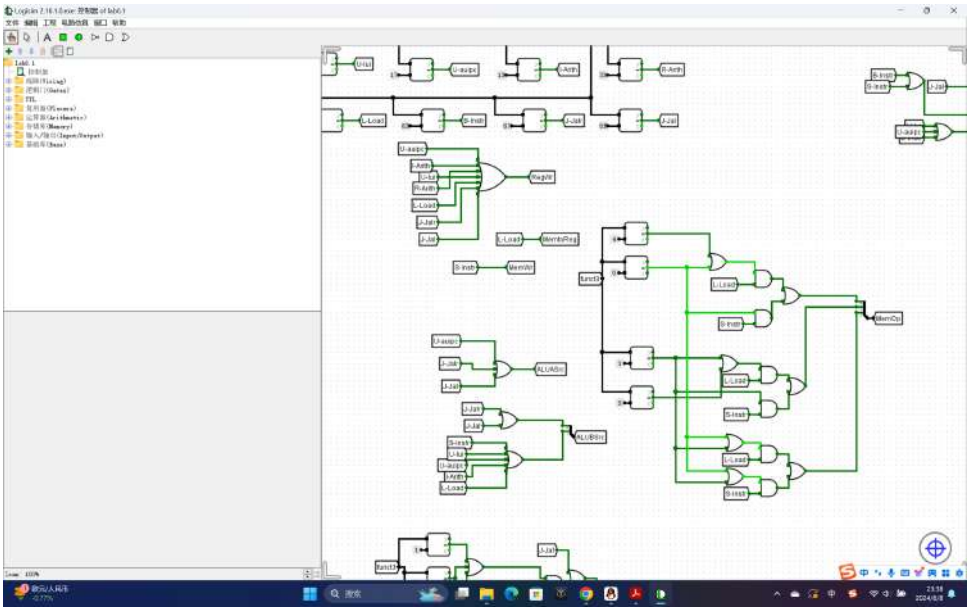

图 11: 电路图 2

1.1.4 实验数据仿真测试图

展示了初始状态和载入两个 ASCII 码的 LED。

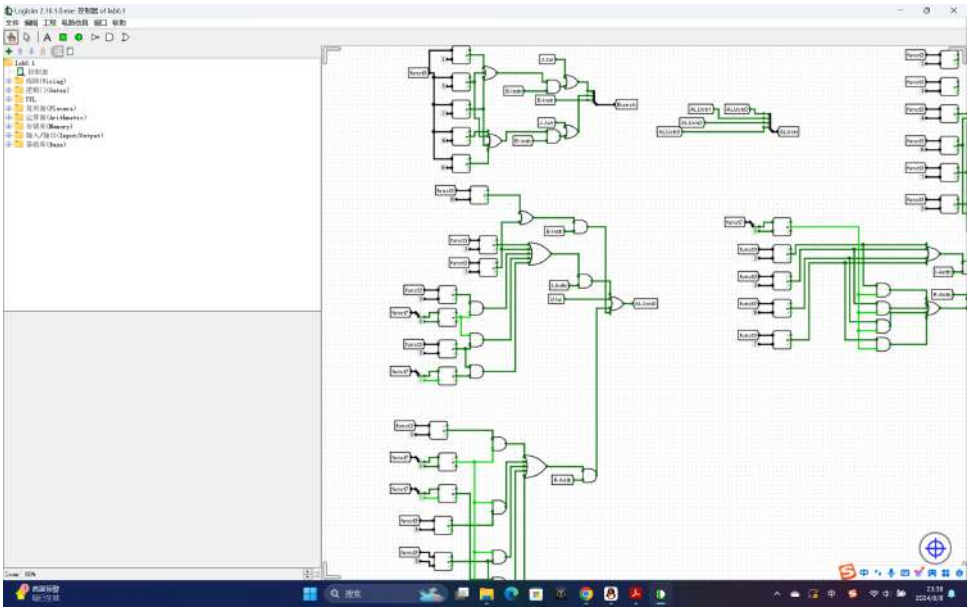

图 12: 模拟 1

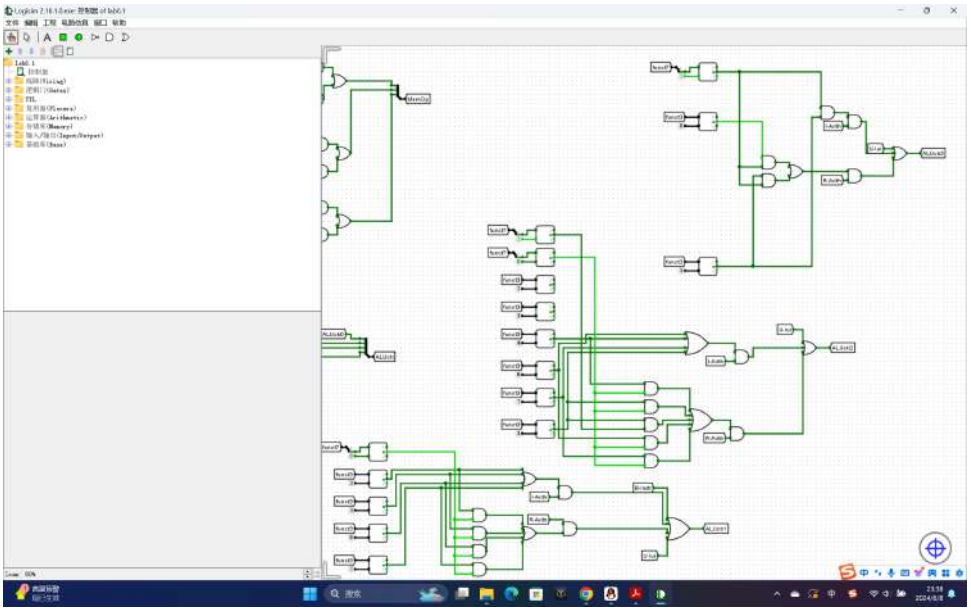

图 13: 模拟 2

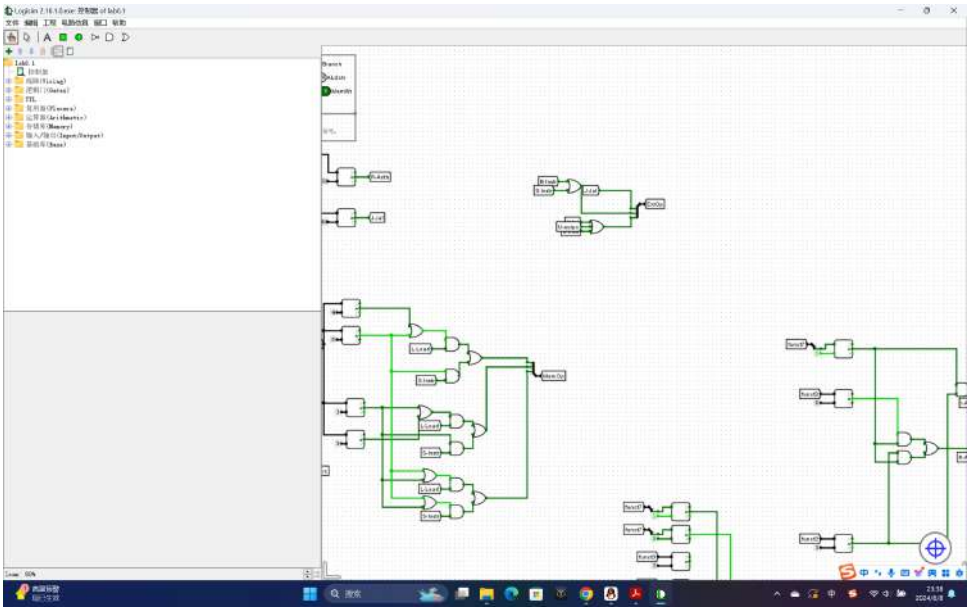

图 14: 模拟 3

功能表在前面。

1.1.5 错误现象及分析

在实验过程中未遇到任何错误。

1.2 单周期 CPU 设计实验

1.2.1 实验内容

为了保证 CPU 的正常执行,在系统中需要考虑复位信号 Reset、中止信号 Halt,还需要考虑状态元件的片选信号 sel 等。复位信号 Reset 用来初始化系统状态,保证 CPU 每次执行程序时,都能从相同的状态开始。当复位信号有效时,PC 寄存器的输入端为程序段初始地址,数据存储器清零端有效。CPU 一旦开始执行程序,下地址计算部件就不断计算下条指令的地址 PC 寄存器持续更新。为了观测当前程序执行结果,在程序执行结束时,需要中止当前程序的执行。本次实验中,使用 ecall 指令作为程序停止执行的指令,在汇编测试程序中,以 ecall 指令作为结束语句 ecall 指令的操作码为 1110011 0x73 。因此修改控制器设计增加一个输出信号 Halt 当操作码 opcode 为 0x73 时,中止信号 Halt 有效,赋值为 1。当中止信号有效时,使得 PC 寄存器的使能端无效,暂停 PC 输出。单周期 CPU 中的状态元件有三个:PC 寄存器、指令存储器和数据存储器。PC 寄存器在每个时钟周期都需要修改,因此 PC 寄存器的片选信号设置为高电平有效,且要始终有效。指令存储器 ROM 的片选信号 Sel 设置为高电平有效且要始终有效。数据存储器中每一片字节存储器 RAM 片选信号设置都设置为高电平有效,但是每一片 RAM 的片选信号的输入值需根据数据存储器读写格式控制信号 MemOp 和最低 2 位地址来决定。

1.2.2 实验整体方案设计

按照要求,控制器加了 ecall 指令,IFU 加了 HALT 信号控制。具体会在错误分析详解。

控制器	同 6.1
Halt	停止计数控制信号
数据通路	同实验 5

表 2: 功能表

1.2.3 实验原理图和电路图

不需要原理图。电路图实现:

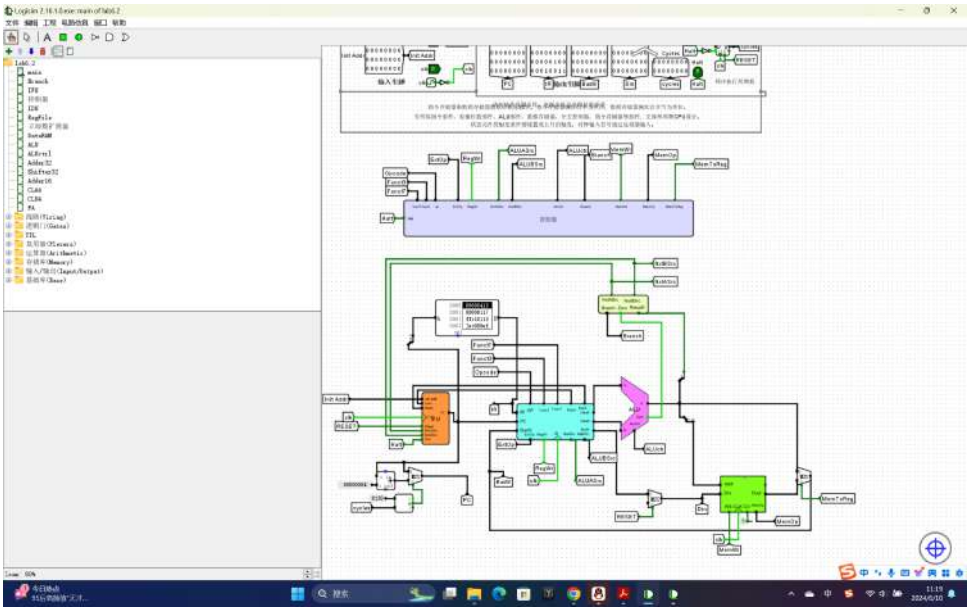

图 15: 主电路

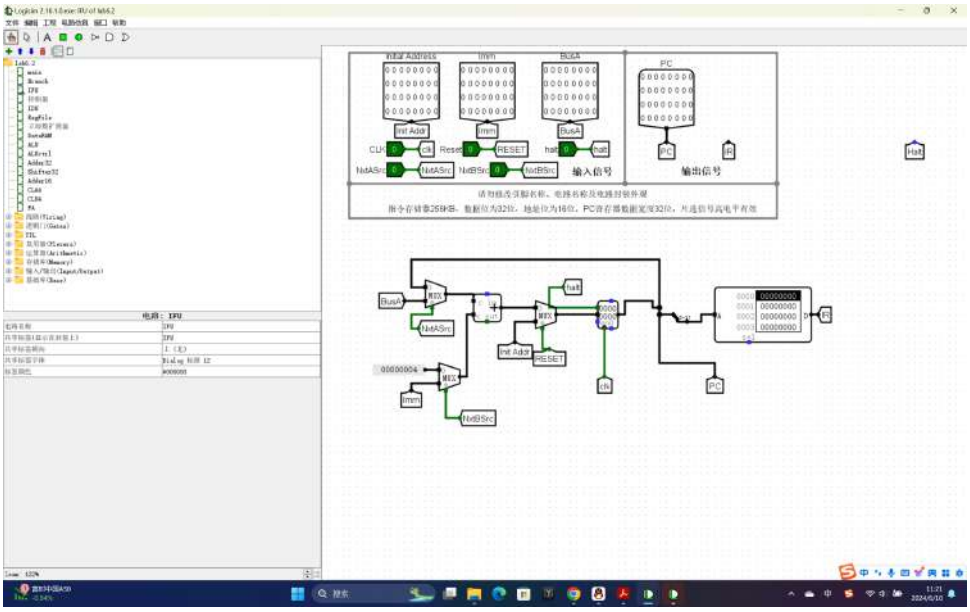

图 16: Halt(有错误)

1.2.4 实验数据仿真测试图

只贴最后结果了。

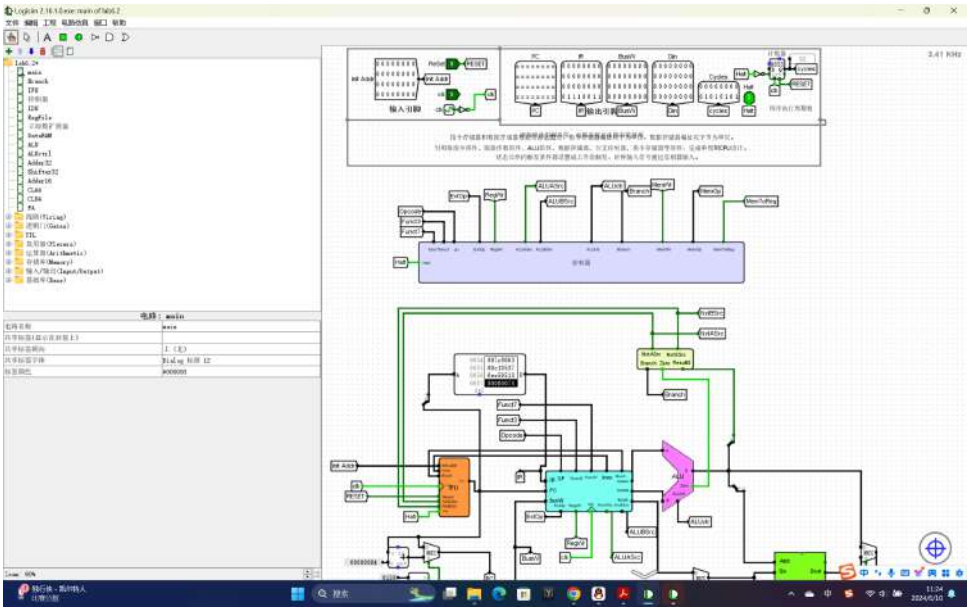

图 17: 1

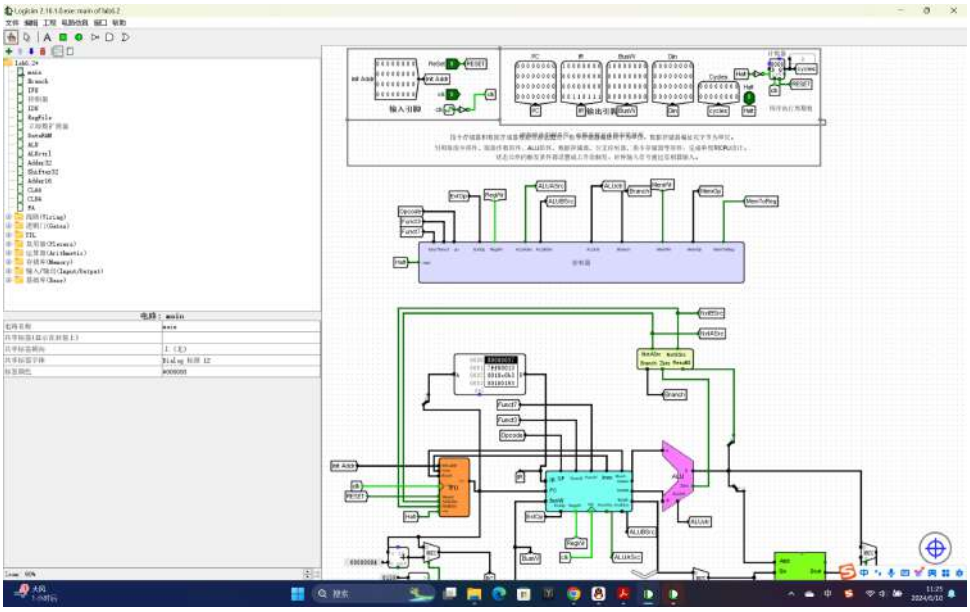

图 18: 2

1.2.5 错误现象及分析

最初遇到了遇到终止指令时仍然输出 PC 的问题(如图),发现是下降沿导致的问题,调成全部上升沿后解决。

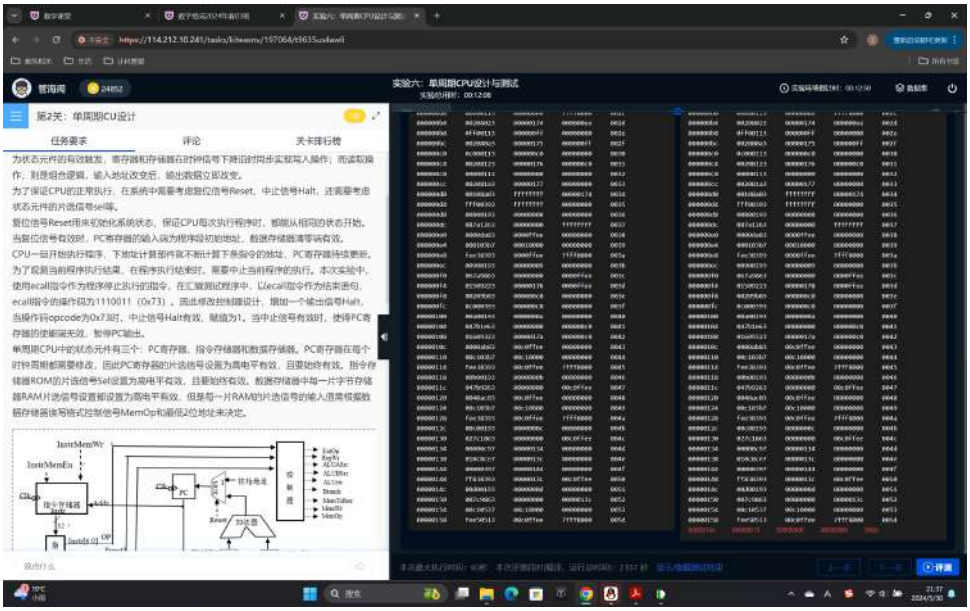

图 19: 下降沿使得指令多输出一行

1.3 用累加和程序验证 CPU 设计

1.3.1 实验内容

本次实验要求是先将计算累加和的 RV32I 汇编语言程序在 RARS 中调试通过，然后导出机器代码并在首行添加“V2.0 raw”语句。在单周期 CPU 的指令存储器中加载该计算累加和的机器代码数据镜像文件。在数据存储器的 0 号单元中设置参数 n。设置仿真使能，启动时钟信号，执行机器代码。程序执行结束后，在数据存储器的地址 4 号单元中观测累加和的计算结果，验证 CPU 设计和测试程序的正确性。

1.3.2 实验整体方案设计

1、编写汇编语言源程序。按照要求做就行。

2. 测试。

1.3.3 实验原理图和电路图

不需要原理图电路图功能表，看结果就行。包含了生成程序指令的过程和测试的结果。

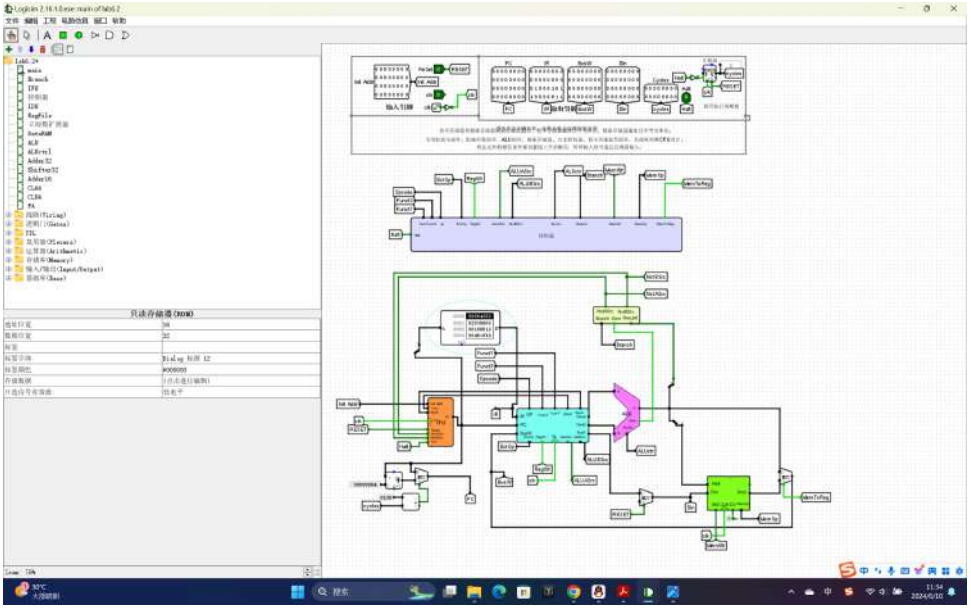

图 20: 1

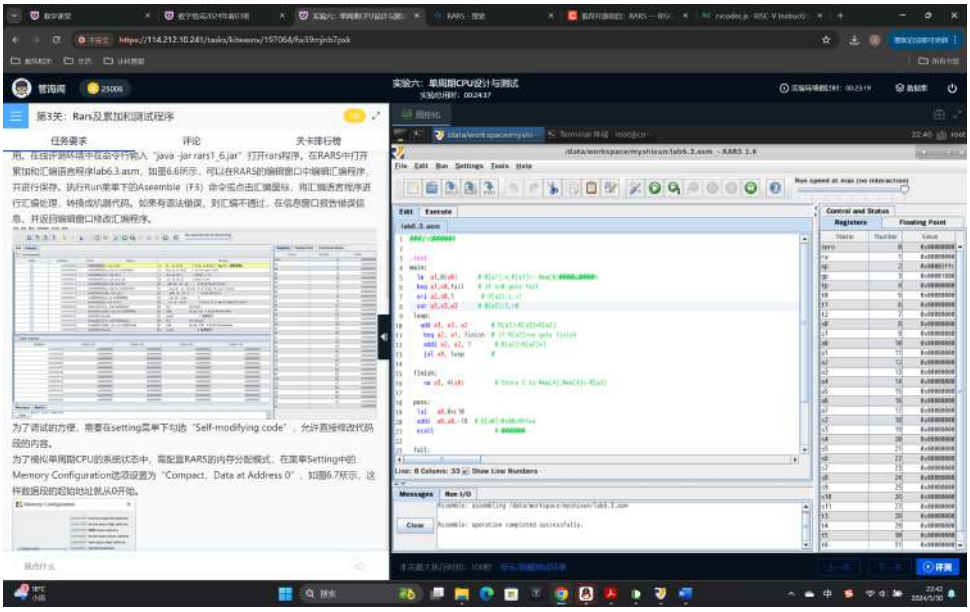

图 21: 2

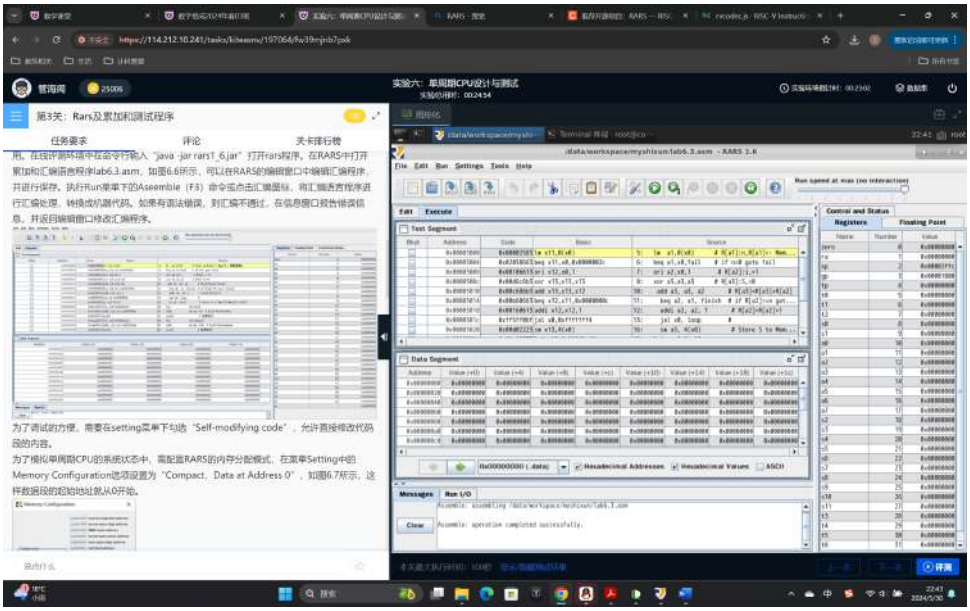

图 22: 3

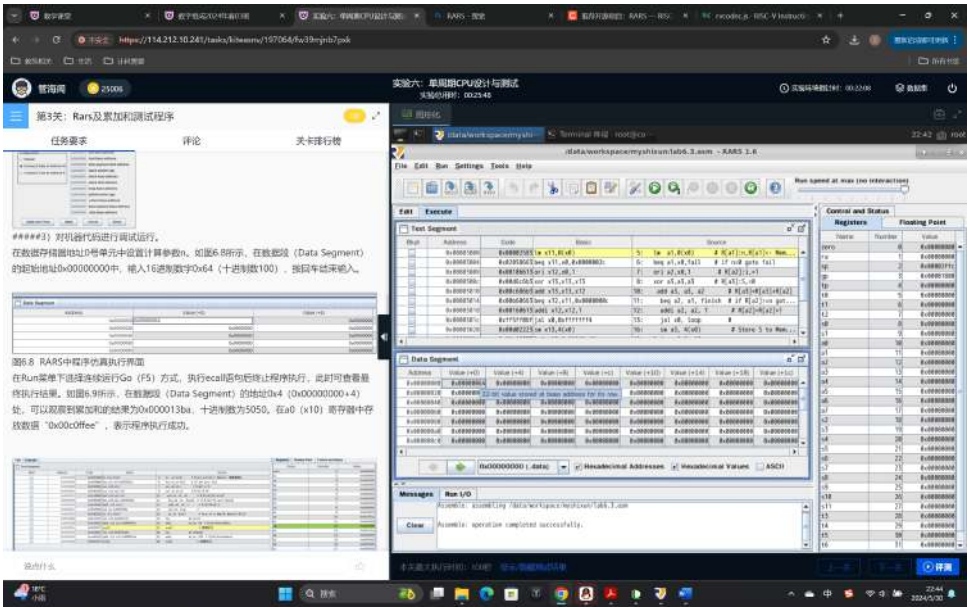

图 23: 4

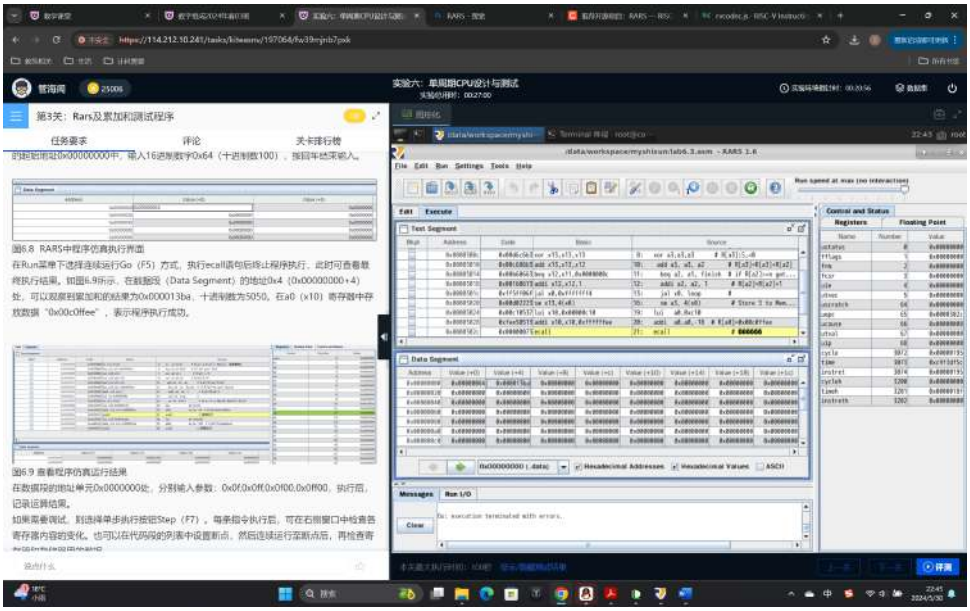

图 24: 5

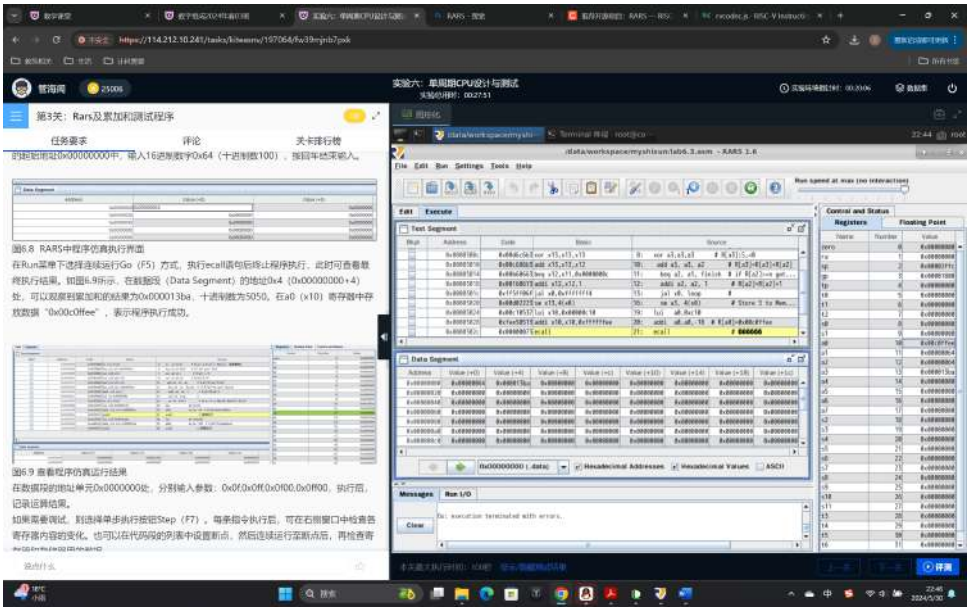

图 25: 6

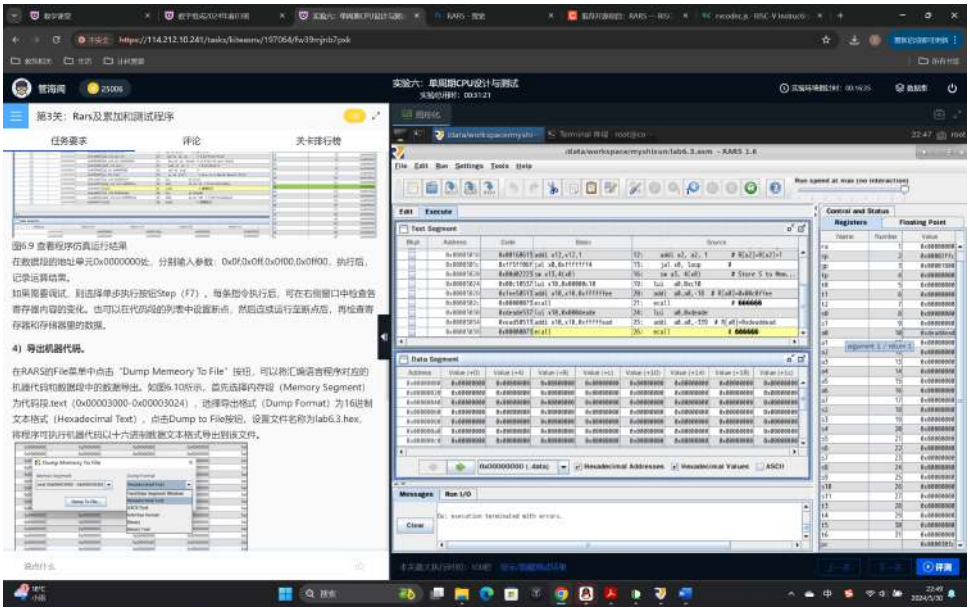

图 26: 7

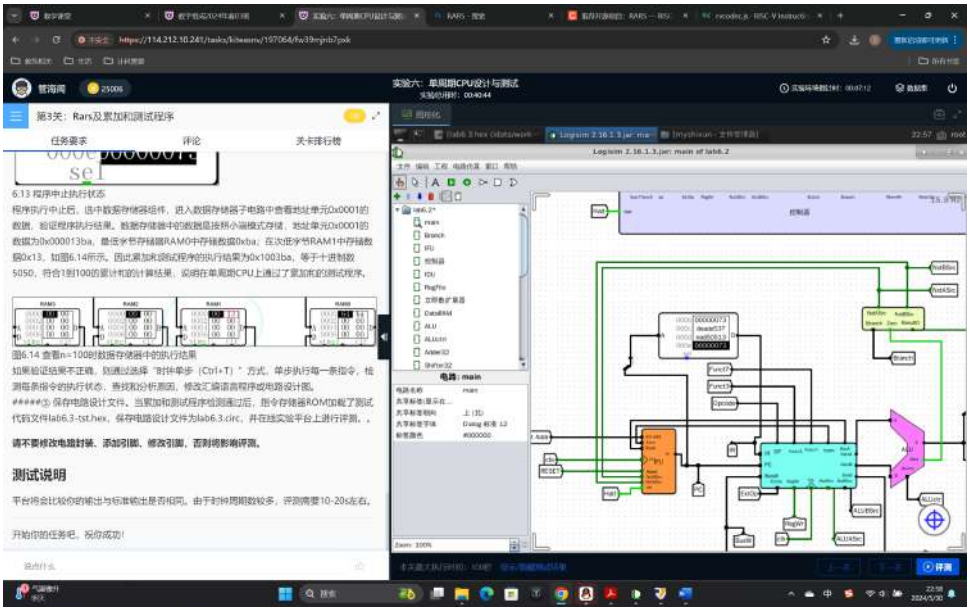

图 27: 8

1.3.4 错误现象及分析

发现 PC 在最后没有正确 + 4. 首先发现的是,dataram 的 LOAD 没有连接,不能存入数据;然后通过面向测试集,给 PC 输出加了个 150 计数器的比较器通过了测试,但是这样导致了 6.4 过不去,最后排查,实际上是 HALt 连错位置了,不应该连禁选,而应该连忽略时钟输入。修改后 6.4 通过,故 6.4 不再重复叙述这个问题。

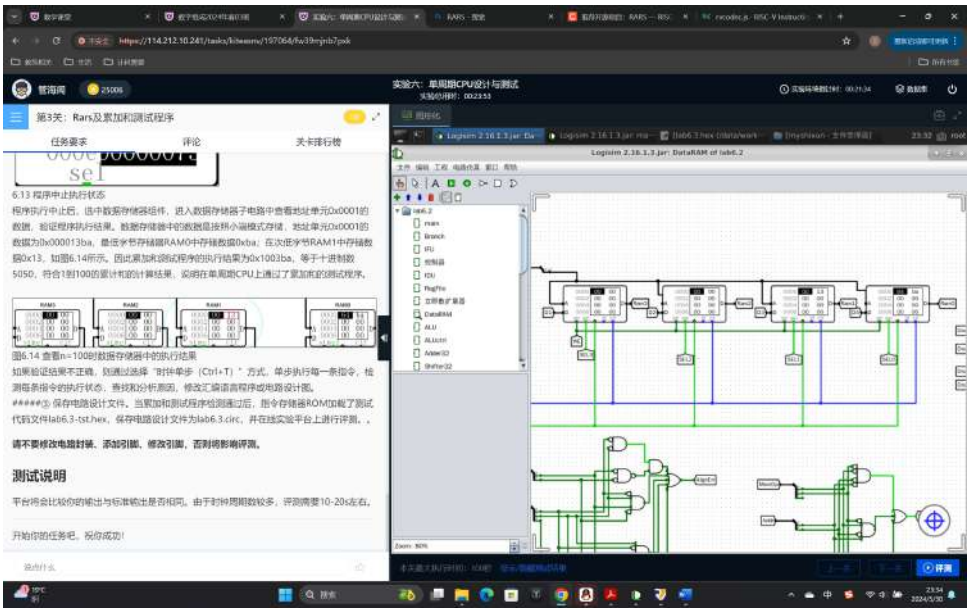

图 28: 错误 1

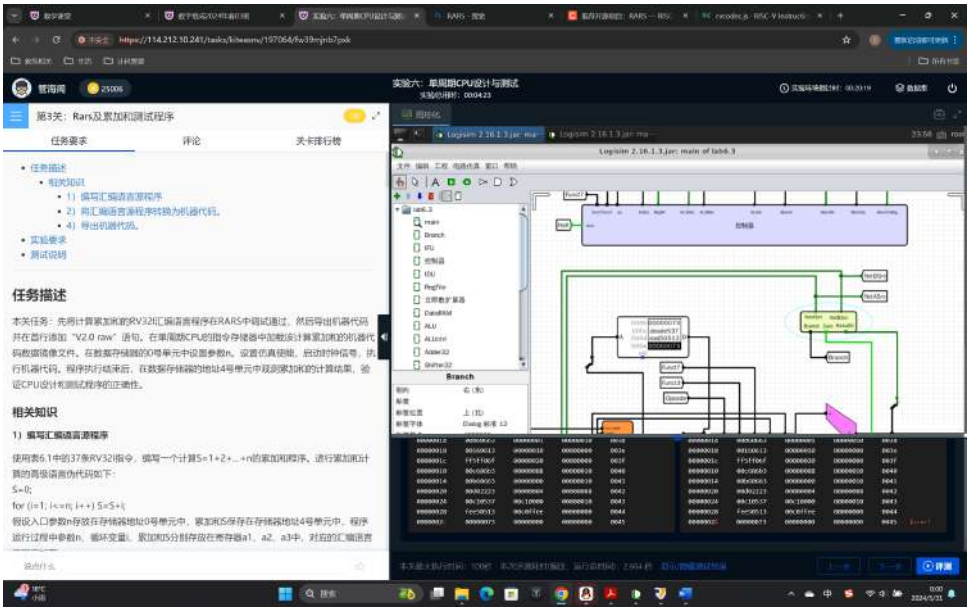

图 29: 错误 2

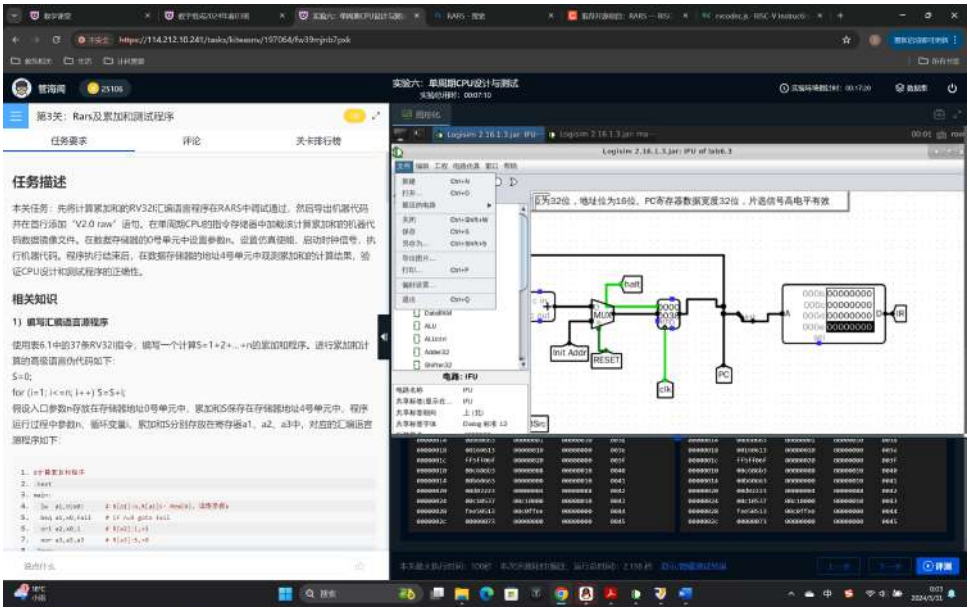

图 30: 错误 3

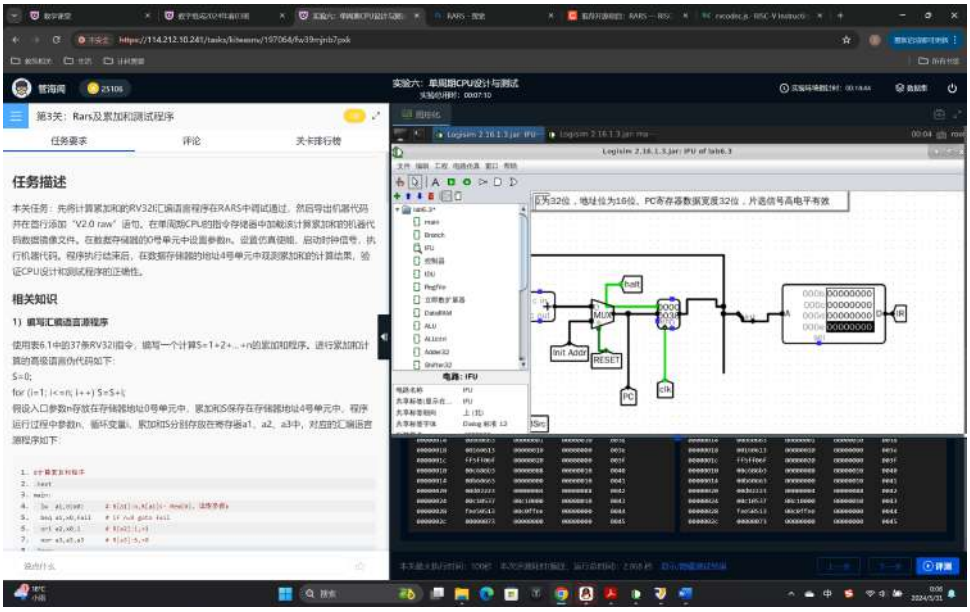

图 31: 错误 4

1.4 用冒泡排序程序进行 CPU 设计验证

1.4.1 实验内容

当冒泡测试程序检测通过后,在指令存储器 ROM 加载了测试代码文件 lab6.4-tst.hex,保存电路设计文件为 lab6.4.circ,并在线实验平台上进行评测。

1.4.2 实验整体方案设计

同 6.3,看结果就行。

1.4.3 实验数据仿真测试图

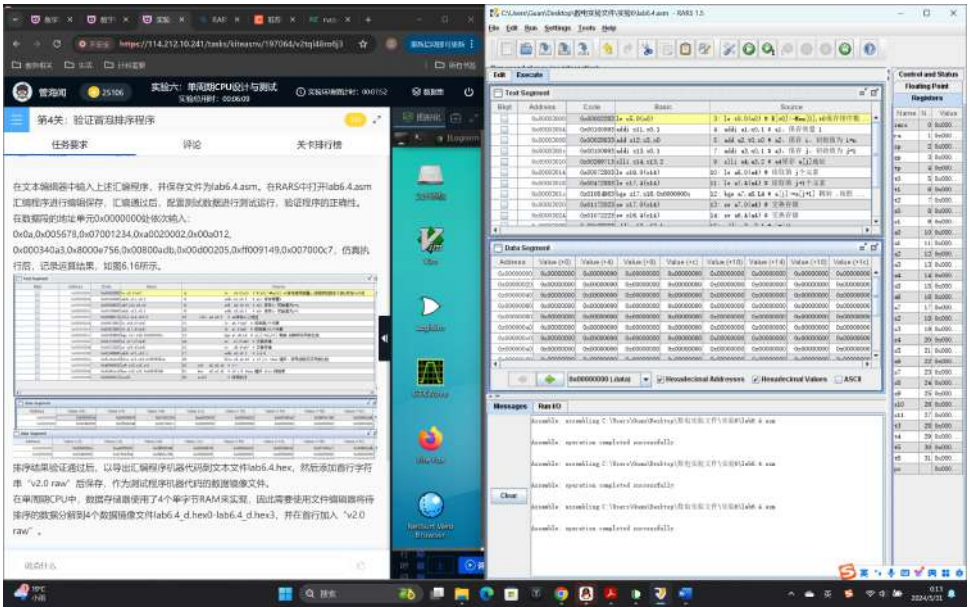

图 32: 1

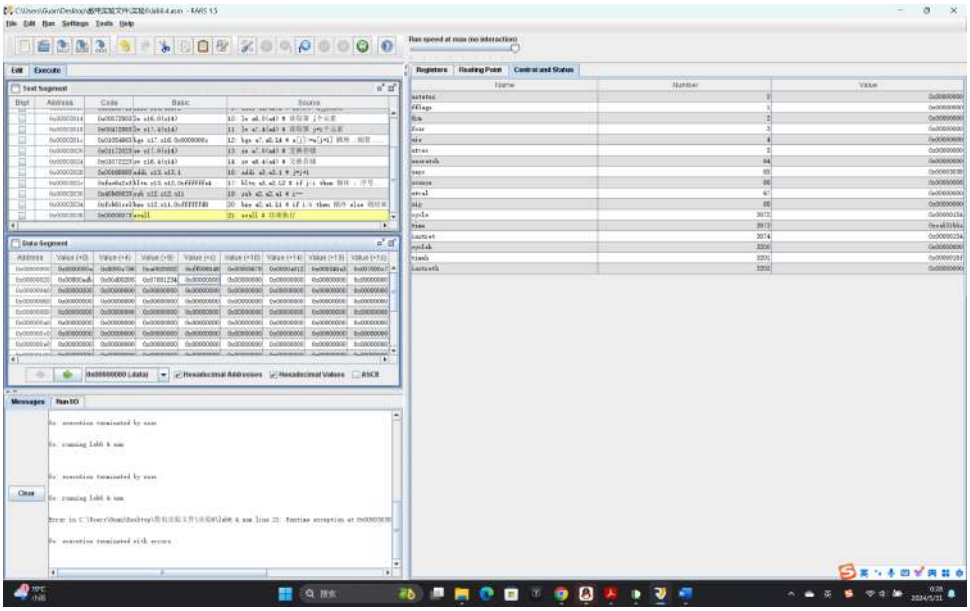

图 33: 2

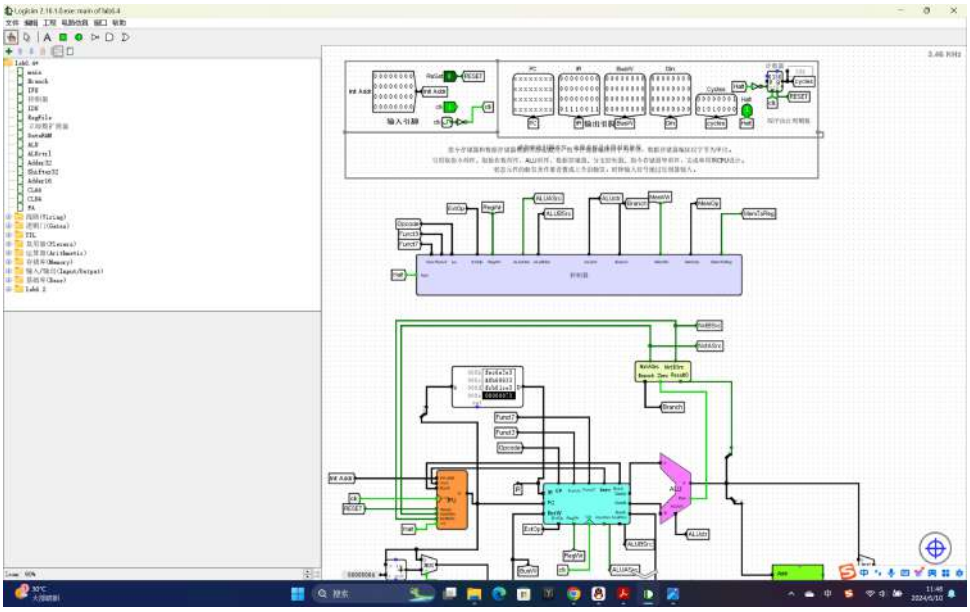

图 34: 3

1.4.4 错误现象及分析

除了 6.3 已经说过的错误外，还有群里说过的，大家都把 BRANCH 的含义看错了，改完后通过了。

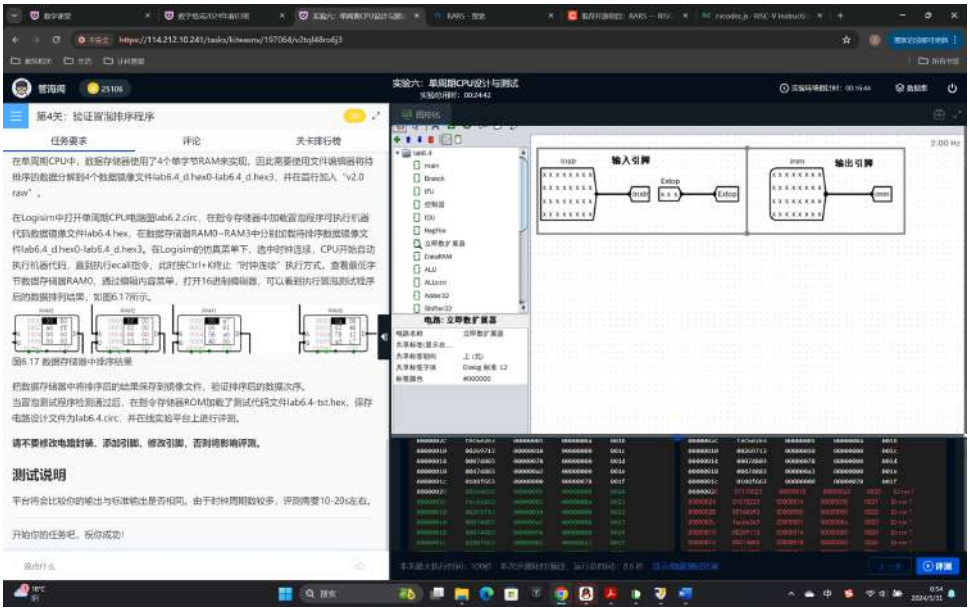

图 35: BRANCH 连错了

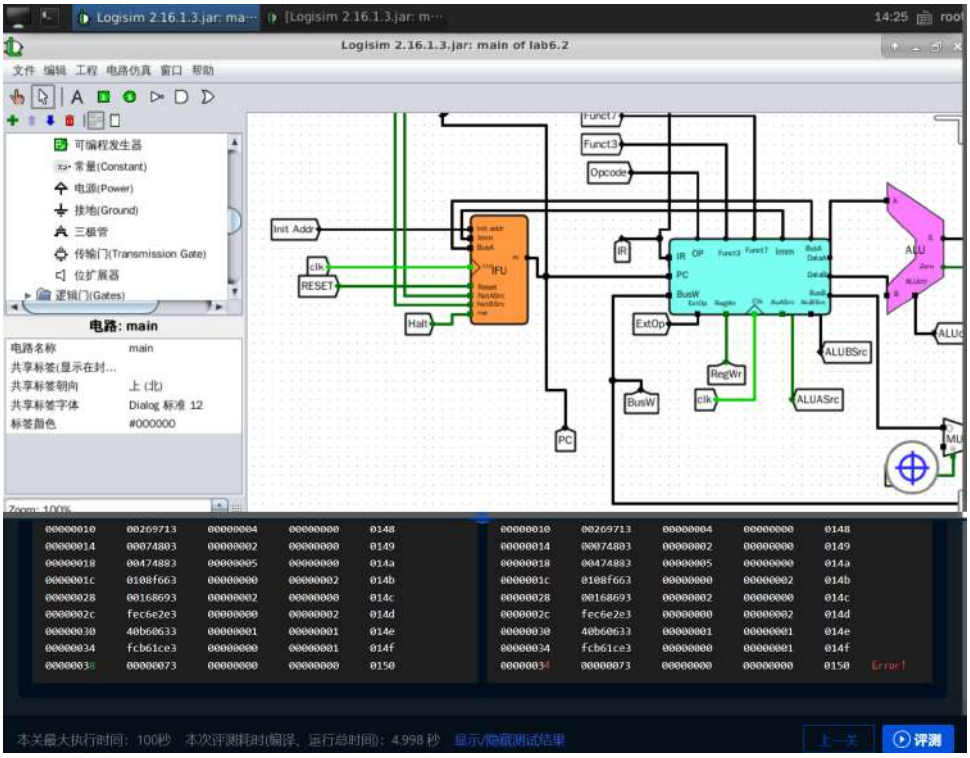

图 36: 2

1.5 官方测试集和 PA 测试

1.5.1 实验内容

1.5.2 实验整体方案设计

附在线下测试集的表中以及 bsort,qsort 跑程序后的结果(用群里的程序)。

2 思考题

2.1 思考题 1

1. 如何在单 CPU 上实现多任务处理,例如同时执行计算累加和与数据排序两个程序阐述思路。

单核 CPU 想要同时进行多任务处理,其实是 CPU 在轮流进行这几个进程,其中每个进程的运行时间很短,这样通过时间片的轮转来实现来实现多个任务同时处理。例如同时进行计算累加和数据排序两个程序,CPU 可先进行一段时间的累加然后再执行排序,接着再回到累加,以此类推,从而实现多任务处理。

2.2 思考题 2

2. 在 CPU 的基础上，如何实现键盘输入、TTY 输出部件等输入输出设备的数据访问，构建完整的计算机系统。

需要将输入输出设备连接到计算机系统的总线上。然后编写控制器程序，负责管理输入输出设备的数据传输。这样，CPU 可以通过总线向控制器发送命令，控制器将输入输出设备的数据传输给 CPU 或将 CPU 的输出传输给设备。

2.3 思考题 3

3. 如何在单周期 CPU 基础上实现多周期 CPU

3. 在多周期 CPU 中，通常需要引入状态机来管理指令的执行过程。每个指令的执行被分为不同的阶段，如指令获取、指令解码、执行、访存和写回等。每个阶段都对应一个时钟周期。通过在每个时钟周期执行不同的操作，可以实现多个指令的并行执行。在流水线 CPU 中，通常有指令获取、指令解码、执行、访存和写回等多个流水线阶段。每个阶段都有自己的功能并且相互独立，指令按照顺序在不同的阶段之间流动。通过流水线寄存器来保存每个阶段的数据，可以实现指令的连续执行。