浙江大学

本科实验报告

课程名称:	计算机组成
实验名称:	Lab4：单周期 CPU
姓　　名:	李宇晗
学　　号:	3240106155
专　　业:	计算机科学与技术
实验地点:	东教 509
指导教师:	林芃
报告日期:	2025/11/30

Lab4-0：CPU 顶层框架搭建

一、 目的和要求

- 确立 CPU 的对外接口标准（指令总线、数据总线、控制信号）。
- 规划内部核心模块（控制器与数据通路）的信号交互拓扑。
- 完成 SCPU 顶层模块的 RTL 结构描述。

二、 实验内容与实现

在 SCPU.v 中，我首先定义了 CPU 核心与外部存储器交互的 I/O 端口，并实例化了两个尚为空壳的子模块。

1. 顶层接口定义：
   - 输入：clk, rst, inst_in (指令), Data_in (内存读数据), MIO_ready。
   - 输出：PC_out (取指地址), Addr_out (数据地址), Data_out (写数据), MemRW (读写控制)。
2. 内部信号映射： 定义了连接控制器与数据通路的内部控制总线：
   - ImmSel, ALUSrc_B, MemtoReg (数据流控制)
   - Jump, Branch (控制流控制)
   - RegWrite, MemRW (存储控制)
   - ALU_Control (运算控制)
3. 模块实例化： 将 inst_in 的部分字段（如 Opcode, Funct3）连接至 SCPU_ctrl 的输入，将控制器的输出信号连接至 DataPath 的控制端口，形成闭环控制系统。

Lab4-1：数据通路子模块填充

一、 目的和要求

- 在已定义的 `DataPath` 接口框架下，实现指令执行所需的硬件逻辑。
- 集成算术逻辑单元、寄存器堆、立即数生成器及 PC 更新逻辑。

二、 关键模块实现

1. 立即数生成器 (ImmGen)： 根据 RISC-V 指令格式 (I/S/B/J 型)，将指令中的立即数部分进行拼接和符号扩展。
   - 代码逻辑：使用 `case(ImmSel)` 语句处理不同类型的位拼接，特别是 B-Type 和 J-Type 的乱序重组。

1. 数据流控制 (MUX)：
   - `ALUSrc_B`：选择 ALU 的第二个操作数是寄存器值 (`Data2`) 还是立即数 (`Imm`)。
   - `MemtoReg`：实现写回逻辑的多路选择，支持 ALU 结果、内存数据或 PC+4 (用于 JAL/JALR) 写回寄存器。

2. PC 更新逻辑： 实现了下一跳 PC (`PC_next`) 的计算逻辑：
   - 默认路径：`PC + 4`。
   - 分支/跳转路径：通过 `Branch & Zero` 或 `Jump` 信号控制 MUX，选择 `PC + Imm` 作为下一指令地址。

Lab4-2：控制器逻辑实现

一、 目的和要求

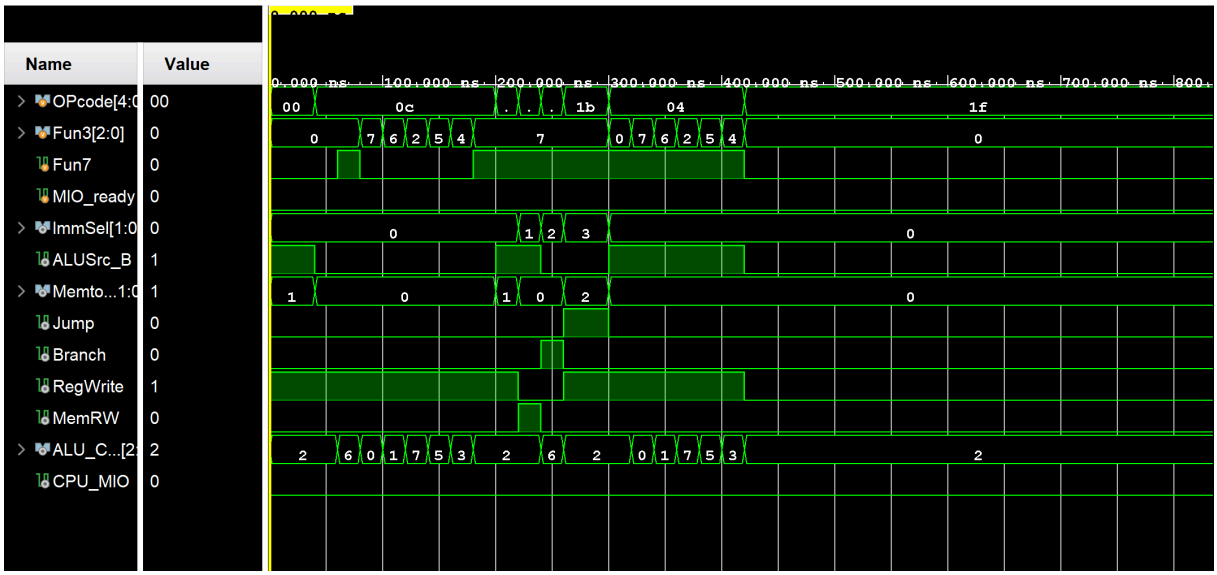
- 设计 CPU 的“大脑”，根据 `Opcode`, `Funct3`, `Funct7` 产生各模块所需的控制信号。
- 实现两级译码机制，简化控制逻辑复杂度。

二、 实现原理

控制器 `SCPU_ctrl` 采用纯组合逻辑设计，分为两个层级：

1. 主控制器 (Main Controller)：
   - 输入：`Opcode` (`inst[6:2]`)。
   - 输出：全局控制信号 (`RegWrite`, `MemRW`, `Branch`, `Jump` 等) 及 2 位中间信号 `ALUOp`。
   - 逻辑示例：对于 `lw` 指令，设置 `ALUSrc_B=1` (选立即数)，`MemtoReg=1` (选内存)，`RegWrite=1`。
2. ALU 解码器 (ALU Decoder)：
   - 输入：`ALUOp`, `Funct3`, `Funct7`。
   - 输出：直接控制 ALU 的 `ALU_Control` 信号。
   - 逻辑示例：当 `ALUOp=10` (R-Type) 且 `Funct7` 第 5 位为 1 时，输出减法控制码。

三、 仿真波形



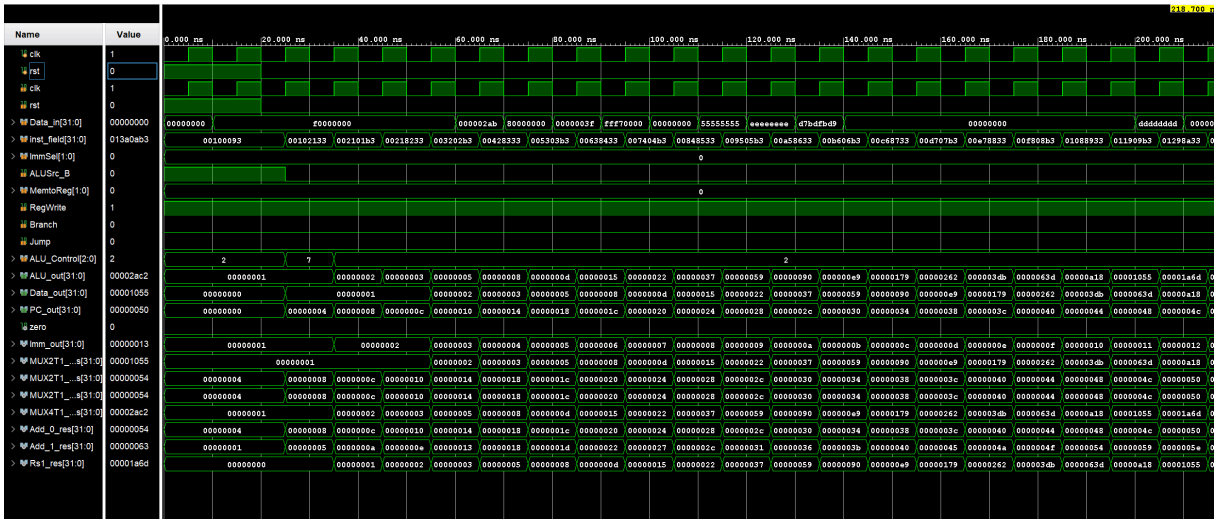
仿真与验证平台构建

为确保设计正确性，在完成 RTL 编码后，设计了仿真平台并进行了硬件上板验证。

一、 仿真平台设计

设计了 Testbench 模拟 CPU 的运行环境：

- 时钟与复位：产生 100MHz 时钟及复位信号。
- 虚拟存储器：在 Testbench 中实例化了指令存储器 (ROM) 和数据存储器 (RAM) 的行为模型，加载 `.coe` 文件初始化指令和数据。
- 波形观测：通过 Vivado Simulator 观察 `PC`, `ALU_out`, `RegWrite` 等关键信号的波形跳变，验证指令执行流是否符合预期。



二、 硬件验证环境 (基于 **Lab 2 CSSTE**)

硬件验证沿用了 Lab 2 设计的 计算机系统综合测试环境 (CSSTE) 框架。

1. 环境集成：将设计好的 `SCPU` 核心模块实例化到 `CSSTE` 顶层模块中，替代原有的逻辑。`CSSTE` 提供了 VRAM、GPIO 控制器及 VGA 显示驱动。

1. 外设交互：
   - 输入：利用开发板开关 (`SW`) 控制 CPU 时钟（单步模式）及复位。
   - 输出：通过七段数码管实时显示 PC 值、ALU 结果或内存数据。
   - MMIO：将 LED 灯映射到特定内存地址 (`0xF0000000`)，验证 CPU 对外设的控制能力。

硬件测试结果

一、 ALU 指令与寄存器堆测试

1.1) 测试程序设计

测试代码实现了斐波那契数列计算序列，完整指令流及其对应的机器码如下：

Step	PC	机器码	汇编指令	预期结果 (Reg)
1	00	00100093	addi x1, x0, 1	x1 = 00000001
2	04	00102133	slt x2, x0, x1	x2 = 00000001
3	08	001101B3	add x3, x2, x1	x3 = 00000002
4	0C	00218233	add x4, x3, x2	x4 = 00000003
5	10	003202B3	add x5, x4, x3	x5 = 00000005
6	14	00528333	add x6, x5, x5	x6 = 0000000A
7	18	005303B3	add x7, x6, x5	x7 = 0000000F
8	1C	0063F433	and x8, x7, x6	x8 = 0000000A

Table 1: ALU 测试指令序列设计 (含机器码)

1.2) 上板验证结果

在单步调试模式下，观测寄存器堆写入端口的数据，所有步骤均与预期一致。

Step	PC	汇编指令	实际观测值 (Reg)	微架构行为分析
1	00	addi x1, x0, 1	x1 = 00000001	ImmGen 生成立即数 1，ALU 加法旁路输出
2	04	slt x2, x0, x1	x2 = 00000001	比较运算：0 < 1 为真，结果置 1
3	08	add x3, x2, x1	x3 = 00000002	R-Type 加法：1 + 1 = 2
4	0C	add x4, x3, x2	x4 = 00000003	数据依赖验证：读取前两步结果 (2+1)
5	10	add x5, x4, x3	x5 = 00000005	斐波那契逻辑：3 + 2 = 5
6	14	add x6, x5, x5	x6 = 0000000A	自加运算：5 + 5 = 10 (Hex 0A)
7	18	add x7, x6, x5	x7 = 0000000F	累加运算：10 + 5 = 15 (Hex 0F)
8	1C	and x8, x7, x6	x8 = 0000000A	逻辑与验证：0F & 0A = 0A

Table 2: ALU 功能硬件执行完整记录

二、 LW/SW 指令与存储器测试

2.1) 访存指令流设计

设计了“两读两写”的测试序列，验证数据在 Memory 与 Register 之间的无损传输。

Step	PC	机器码	汇编指令	MemRW 预期
1	00	03402B03	lw x22, 0x34(x0)	0 (Read)
2	04	04802B83	lw x23, 0x48(x0)	0 (Read)
3	08	05605823	sw x22, 0x50(x0)	1 (Write)
4	0C	03705923	sw x23, 0x32(x0)	1 (Write)
5	10	00000033	add x0, x0, x0	0 (Read)

Table 3: 存储器读写指令流设计 (含机器码)

2.2) 读写通路验证

通过观测 CPU 对外的数据总线 (Data_out) 和地址总线 (Addr_out)，验证了 Load-Store 逻辑的闭环。

Step	PC	Addr Bus (Hex)	Data Bus (Hex)	总线行为分析
1	00	00000034	- (High Z)	ALU 正确计算读地址 34h，MemRW=0
2	04	00000048	- (High Z)	ALU 正确计算读地址 48h，MemRW=0
3	08	00000050	55555555	写操作：寄存器 x22 的值 (来自地址 34h) 被写出
4	0C	00000032	AAAAAAAA	写操作：寄存器 x23 的值 (来自地址 48h) 被写出
5	10	-	-	NOP 指令，总线状态保持稳定，测试结束

Table 4: 存储器接口信号物理观测结果

三、 动态 LW/SW 测试

3.1) 控制逻辑设计

测试程序通过向高位地址写入数据来驱动外设。下表展示了包含初始化和两轮循环的完整执行流设计。

Step	PC	机器码	汇编指令	操作目的
1	00	E0000137	lui x2, 0xE0000	初始化数码管基址
2	04	F00001B7	lui x3, 0xF0000	初始化 LED 基址
3	08	00000093	addi x1, x0, 0	初始化计数器 x1=0
4	0C	00108093	addi x1, x1, 1	计数 +1 (x1=1)
5	10	00112023	sw x1, 0(x2)	写数码管 (Value=1)
6	14	0011A023	sw x1, 0(x3)	写 LED (Value=1)
7	18	FE000AE3	beq x0, x0, -8	跳转回 PC=0C
8	0C	00108093	addi x1, x1, 1	Loop 2: 计数 +1 (x1=2)
9	10	00112023	sw x1, 0(x2)	Loop 2: 写数码管 (Value=2)
10	14	0011A023	sw x1, 0(x3)	Loop 2: 写 LED (Value=2)

Table 5: MMIO 动态循环逻辑展开 (含机器码)

3.2) 物理现象观测

将比特流下载至 FPGA，开启手动单步时钟，观测外设状态变化。

Step	PC (Seg-7)	LED 状态	微架构与外设分析
1-3	00 -> 02	全灭	完成基址与计数器初始化
4	03	全灭	执行 ADDI，内部寄存器更新为 1
5	04	全灭	执行 SW 数码管，数码管锁存值 1
6	05	LED[0] 亮	执行 SW LED，地址 F0... 命中，LED 更新
7	06 -> 03	保持	执行 BEQ，PC 成功跳转回 03 (字地址 0C)
8	03	保持	Loop 2: 内部寄存器更新为 2
9	04	保持	Loop 2: 数码管更新显示为 2
10	05	LED[1] 亮	Loop 2: LED 更新为二进制 10，MMIO 验证通过

Table 6: MMIO 动态上板验证结论

Lab4-3：CPU 设计-指令集扩展

一、 实验目的与要求

1. 掌握 RISC-V 指令集架构扩展方法：深入理解不同指令格式（R, I, S, B, U, J）的数据通路差异，学会通过修改硬件描述语言（Verilog）来支持新指令。
2. 实现 31 条 RISC-V 指令的 CPU：将 CPU 功能扩展为支持完整的算术逻辑、移位、比较及各类跳转指令。
   - 运算扩展：`and/or/xor` 系列逻辑运算，`sll/srl/sra` 系列移位运算，`slt/sltu` 系列比较运算。
   - 控制流扩展：`bne` (不相等跳转), `jal` (J 型跳转), `jalr` (寄存器跳转), `lui` (U 型立即数加载)。
3. 系统集成与验证：重新设计控制器与数据通路的接口，利用 Vivado 仿真及 FPGA 上板验证扩展后 CPU 的正确性。

二、 实验内容与原理

1. JALR 指令：I 型指令，跳转目标为 `Reg[rs1] + Imm`。需在 Next PC 逻辑中引入 ALU 计算结果作为跳转地址。
2. LUI 指令：U 型指令，用于加载高 20 位立即数。数据通路需支持将立即数不经 ALU 计算直接写入寄存器。
3. BNE 指令：与 BEQ 逻辑相反。需增加控制信号配合 Zero 标志位，实现“结果非零时跳转”的逻辑。

三、 实现方法、步骤与调试

3.1) 控制器设计 **(`SCPU_ctrl_more.v`)**

控制器沿用两级译码结构，重点针对新增指令扩充控制信号：

1. 信号扩展：将 `Jump` 扩展为 2 位（`01`: JAL, `10`: JALR），新增 `BranchN` 信号处理 BNE，`ALU_Control` 扩展至 4 位以容纳更多运算。
2. 主解码逻辑：
   - LUI：设置 `MemtoReg` 为自定义状态（如 `11`），使立即数直通写回。
   - JAL/JALR：通过 `Jump` 信号区分跳转目标来源（PC+Imm 或 ALU_out）。
   - BNE：通过 `Funct3` 区分 BEQ/BNE，置位 `BranchN`。
3. ALU 解码逻辑：完善 `case` 语句，利用 `Funct7[5]` 区分逻辑右移 (`srl`) 和算术右移 (`sra`)。

3.2) 数据通路设计 **(`DataPath_more.v`)**

针对新增的数据流向，重点改造了多路选择器（MUX）逻辑：

1. Next PC 逻辑升级：将 Next PC 的选择逻辑改为级联结构。优先级如下：
   - 若 `Jump=2` (JALR)，选择 `ALU_out`。

- 若 `Jump=1` (JAL) 或满足分支条件 `(Branch & Zero) | (BranchN & ~Zero)`，选择 `PC + Imm`。
- 默认选择 `PC + 4`。

2. 寄存器写回扩展：扩展 Write Back MUX，新增第四路输入连接 `Imm_out`，专门服务于 `LUI` 指令。

3.3) ALU 模块扩展 (`ALU_more.v`)

利用 Verilog 高层运算符简化实现：

- 移位运算：使用 `>>>` 实现算术右移（需配合 `$signed`），`>>` 和 `<<` 实现逻辑移位。
- 比较运算：`SLT` 使用带符号比较，`SLTU` 使用无符号比较。
- 逻辑运算：补全 `XOR`, `OR`, `AND` 等位运算逻辑。

四、 结果与分析

4.1) 测试程序设计 (Test Program Design)

本实验设计了包含 31 个步骤的综合测试序列，涵盖了算术、逻辑、移位、分支与跳转指令。完整指令序列及其机器码如下：

Step	PC	Machine Code	Assembly Instruction
1	00	00007293	andi x5, x0, 0
2	04	00007313	andi x6, x0, 0
3	08	88888137	lui x2, 0x88888
4	0C	00832183	lw x3, 8(x6)
5	10	0032A223	sw x3, 4(x5)
6	14	00402083	lw x1, 4(x0)
7	18	01C02383	lw x7, 28(x0)
8	1C	00338863	beq x7, x3, 16
9	2C	007282B3	add x5, x5, x7
10	30	00230333	add x6, x6, x2
11	34	00531463	bne x6, x5, 8
12	3C	40530433	sub x8, x6, x5
13	40	405304B3	sub x9, x6, x5
14	44	0080006F	jal x0, 8
15	4C	0072F533	and x10, x5, x7
16	50	00157593	andi x11, x10, 1
17	54	00B51463	bne x10, x11, 8
18	5C	00A5E5B3	or x11, x11, x10
19	60	0015E513	ori x10, x11, 1
20	64	00558463	beq x11, x5, 8
21	6C	00A5C633	xor x12, x11, x10

Step	PC	Machine Code	Assembly Instruction
22	70	00164613	xori x12, x12, 1
23	74	00B61463	bne x12, x11, 8
24	7C	0012D293	srli x5, x5, 1
25	80	00060463	beq x12, x0, 8
26	88	00129293	slli x5, x5, 1
27	8C	00B28463	beq x5, x11, 8
28	94	001026B3	slt x13, x0, x1
29	98	00503733	sltu x14, x0, x5
30	9C	F65FF06F	jal x0, -156

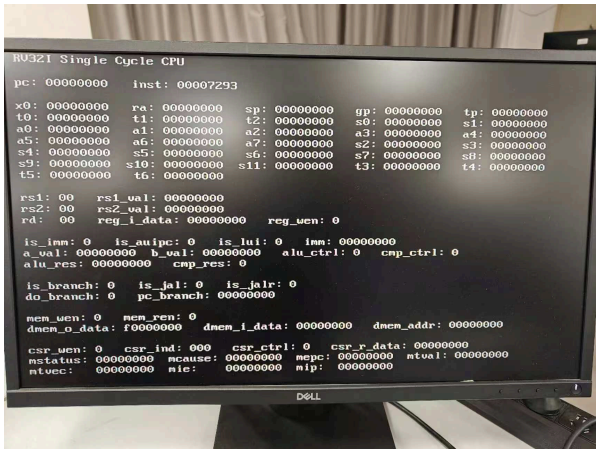
4.2) 执行结果综合记录 **(Execution Trace)**

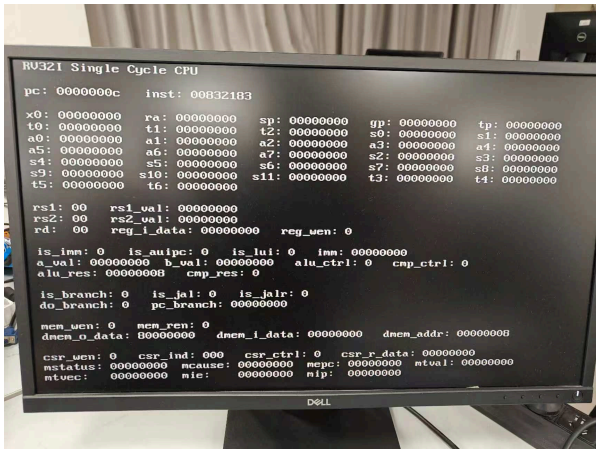
下表记录了 FPGA 上板单步调试的完整过程。重点验证了跳转指令（PC 突变）和运算指令的寄存器写回值。

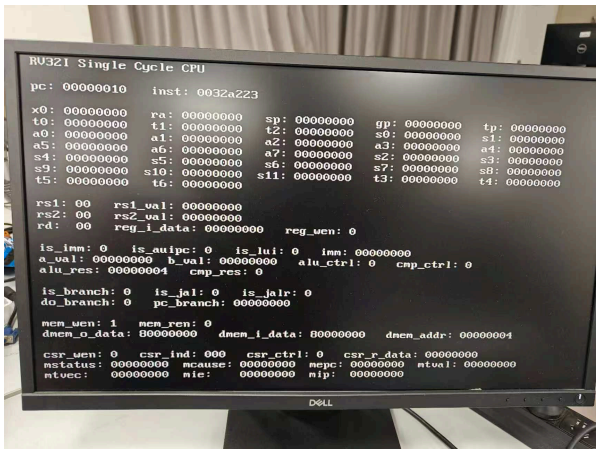
Step	PC	汇编指令	观测结果	微架构行为与结果分析
1	00	andi x5, x0, 0	x5 = 00000000	初始化清零, ALU 逻辑与功能正常
2	04	andi x6, x0, 0	x6 = 00000000	初始化清零
3	08	lui x2, 0x88888	x2 = 88888000	LUI 验证: ImmGen 左移 12 位, 直通写回
4	0C	lw x3, 8(x6)	x3 = 80000000	基址寻址: Addr=0+8, 读 RAM 成功
5	10	sw x3, 4(x5)	MemRW = 1	Store 验证: 数据 80000000 写入地址 4
6	14	lw x1, 4(x0)	x1 = 80000000	Load 验证: 从地址 4 正确回读刚才写入的数据
7	18	lw x7, 28(x0)	x7 = 80000000	读取常量数据
8	1C	beq x7, x3, 16	PC -> 2C	BEQ 跳转成功: x7==x3, Branch=1, 跳过中间指令
9	2C	add x5, x5, x7	x5 = 80000000	跳转落地验证: x5=0+80000000
10	30	add x6, x6, x2	x6 = 88888000	寄存器累加正确
11	34	bne x6, x5, 8	PC -> 3C	BNE 跳转成功: x6!=x5, BranchN=1 生效
12	3C	sub x8, x6, x5	x8 = 08888000	减法验证: 88888000 - 80000000 = 08888000
13	40	sub x9, x6, x5	x9 = 08888000	重复指令验证稳定性

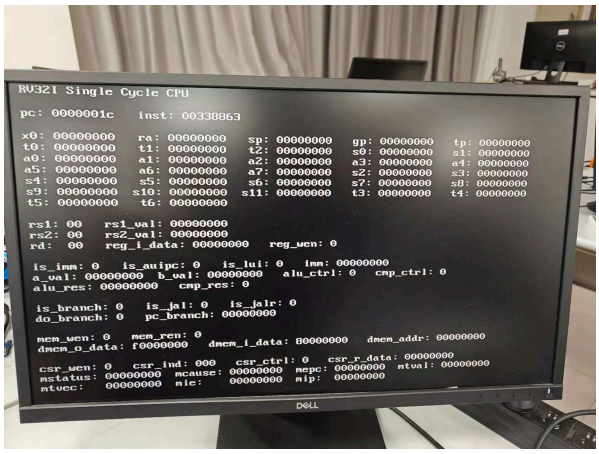
Step	PC	汇编指令	观测结果	微架构行为与结果分析
14	44	jal x0, 8	PC -> 4C	JAL 跳转成功：无条件相对跳转 PC+8
15	4C	and x10, x5, x7	x10 = 80000000	逻辑与：80.. & 80.. = 80..
16	50	andi x11, x10, 1	x11 = 00000000	逻辑与立即数：末位为 0
17	54	bne x10, x11, 8	PC -> 5C	BNE 跳转成功：条件成立
18	5C	or x11, x11, x10	x11 = 80000000	逻辑或：0 | 80.. = 80..
19	60	ori x10, x11, 1	x10 = 80000001	逻辑或立即数：80.. | 1 = 80..01
20	64	beq x11, x5, 8	PC -> 6C	BEQ 跳转成功：条件成立
21	6C	xor x12, x11, x10	x12 = 00000001	异或验证：80..00 ^ 80..01 = 1
22	70	xori x12, x12, 1	x12 = 00000000	异或立即数：1 ^ 1 = 0
23	74	bne x12, x11, 8	PC -> 7C	BNE 跳转成功
24	7C	srli x5, x5, 1	x5 = 40000000	逻辑右移：Funct7 正确译码，高位补 0
25	80	beq x12, x0, 8	PC -> 88	BEQ 跳转成功：0==0
26	88	slli x5, x5, 1	x5 = 80000000	逻辑左移：40.. << 1 = 80..
27	8C	beq x5, x11, 8	PC -> 94	BEQ 跳转成功
28	94	slt x13, x0, x1	x13 = 00000000	有符号比较：0 < −2^31 (负数) 为假
29	98	sltu x14, x0, x5	x14 = 00000001	无符号比较:0 < LargeNum 为真
30	9C	jal x0, -156	PC -> 00	J-Type 负向跳转：PC 回滚至 00，循环测试通过

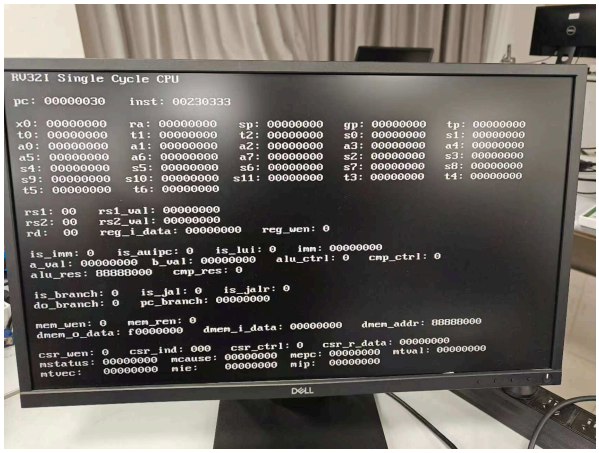
另附过程中部分 VGA 结果记录：

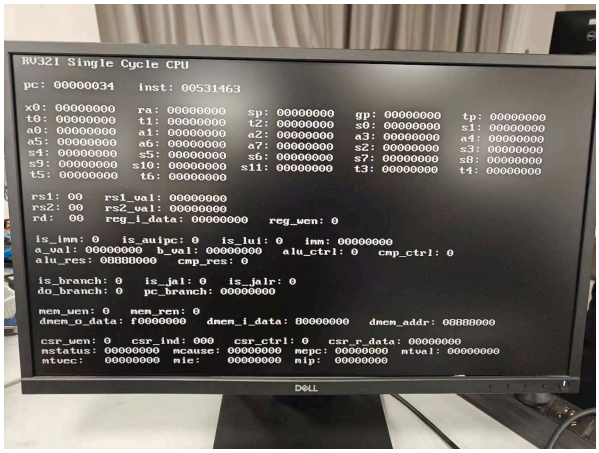












五、 讨论与心得

1. 控制信号优化：在扩展 JAL 和 JALR 时，没有盲目增加新信号，而是将 `Jump` 信号位宽扩展为 2 位。这不仅区分了跳转目标（PC+Imm vs ALU_out），还保持了接口的整洁，降低了译码逻辑的冗余度。
2. 数据通路复用：对于 `LUI` 指令，虽然可以通过 ALU 运算实现，但我选择了在写回级增加直通路径。这种设计虽然略微增加了 MUX 的复杂度，但解耦了 ALU 与 立即数生成逻辑，使得 ALU 专注于运算，结构更加清晰。
3. 实验总结：通过本次实验，成功构建了支持 31 条指令的单周期 CPU。利用 FPGA 单步调试功能，能够清晰地观测到每一条指令执行时的 PC 跳转和寄存器状态变化，验证了 CPU 对复杂控制流（如嵌套跳转、循环）的处理能力。
