**开发者说明**：

12311047 胡耘宁 测试场景一100-111，测试场景二000-011，CPU的debug，数码管显示 50%

12311043 魏宇晴 测试场景一000-011，测试场景二100-111，CPU的连线和debug 50%

**开发计划日程安排和实施情况：**

2025.5.10 CPU子模块连线以及初步设计

2025.5.15 CPU每个子模块的仿真文件

2025.5.27 设计好CPU（debug以及顶层模块仿真）

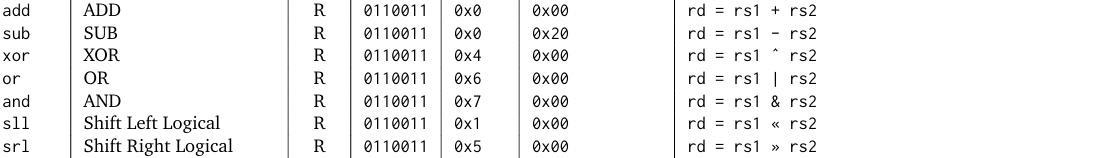
2025.5.29 完成测试场景一

2025.6.2 完成测试场景二

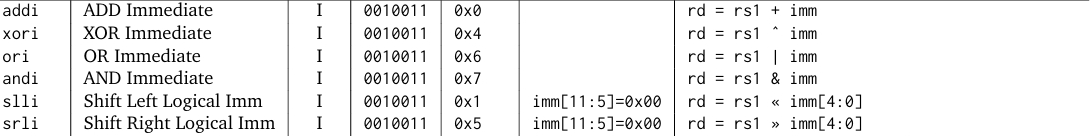
**CPU架构设计说明：**

CPU特性：

ISA：





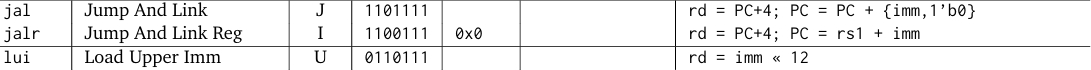












寄存器：32个32位寄存器

1. 单周期CPU，暂不支持pipeline。
2. 寻址空间设计：属于哈佛结构。

寻址单位：字节寻址（4字节）

指令空间：本系统的指令空间基于 RISC-V 32 位 ISA，支持R类型，I类型（opcode=0010011，0000011，1100111），S类型，B类型，J类型，U类型（lui）指令的执行

数据空间的大小：储存器（32位），寄存器（32位）

栈空间的基地址：CPU通过栈指针寄存器追踪当前指令，在汇编文件末尾添加循环跳转防止爆栈

1. 对外设IO的支持：采用MMIO。

LED 外设地址：0xFFFFFC60（led灯T6-K2）, 0xFFFFFC64（数码管）, 0xFFFFFC68（数码管）

Switch 外设地址：0xFFFFFC70（拨码开关P5-R1）, 0xFFFFFC74（拨码开关U3-V2）, 0xFFFFFC78（拨码开关T5），0xFFFFFC80（拨码开关T3）

采用的是轮询方式访问IO。CPU主动读取或写入IO端口，定时或根据条件判断是否需要与外设交互。

CPU接口：

1. 时钟：23MHz
2. 复位：rst
3. 无uart接口
4. 其他IO接口说明：

4.1  内存与IO控制信号

mRead（读内存控制信号，来自Controller）

mWrite（写内存控制信号，来自Controller）

ioRead（读IO控制信号，来自Controller）

ioWrite（写IO控制信号，来自Controller）

4.2  地址与数据总线

addr\_in [31:0]：ALU计算产生的地址输入，决定当前访存或IO的目标地址。

addr\_out [31:0]：输出到数据存储器的地址（通常直接等于addr\_in）。

m\_rdata [31:0]：从内存读出的数据。

io\_rdata [15:0]：从IO设备（开关）读出的数据。

r\_wdata [31:0]：最终送往寄存器文件的数据。

r\_rdata [31:0]：从寄存器文件读出的数据，用于写回内存或IO。

write\_data [31:0]：实际写入内存或IO的数据。

4.3  外设片选信号

LEDCtrl：LED外设片选信号（高有效），用于选择LED设备。

SwitchCtrl：开关（拨码）外设片选信号（高有效），用于选择Switch设备。

4.4  LED与数码管显示

ledout [15:0]：LED灯的输出状态。

num [7:0]、led [7:0]、led1 [7:0]：数码管的输出显示。

4.5  其他控制信号

sure\_lt：确认信号，用于控制IO读取的有效性（如防抖、状态锁存等）。

**方案分析说明：**

在CPU的设计中，硬件和软件的实现方式存在本质区别，主要体现在设计、性能、易修改性、复杂度以及适用场景等方面。硬件设计直接面向电路，能够精确控制数据路径、运算逻辑和时序，适用于高性能计算或定制化需求。这种方式可以实现高度并行化、低延迟的计算，但开发复杂度高，需要处理时钟同步、流水线冲突、功耗优化等问题，且通常需要硬件支持。

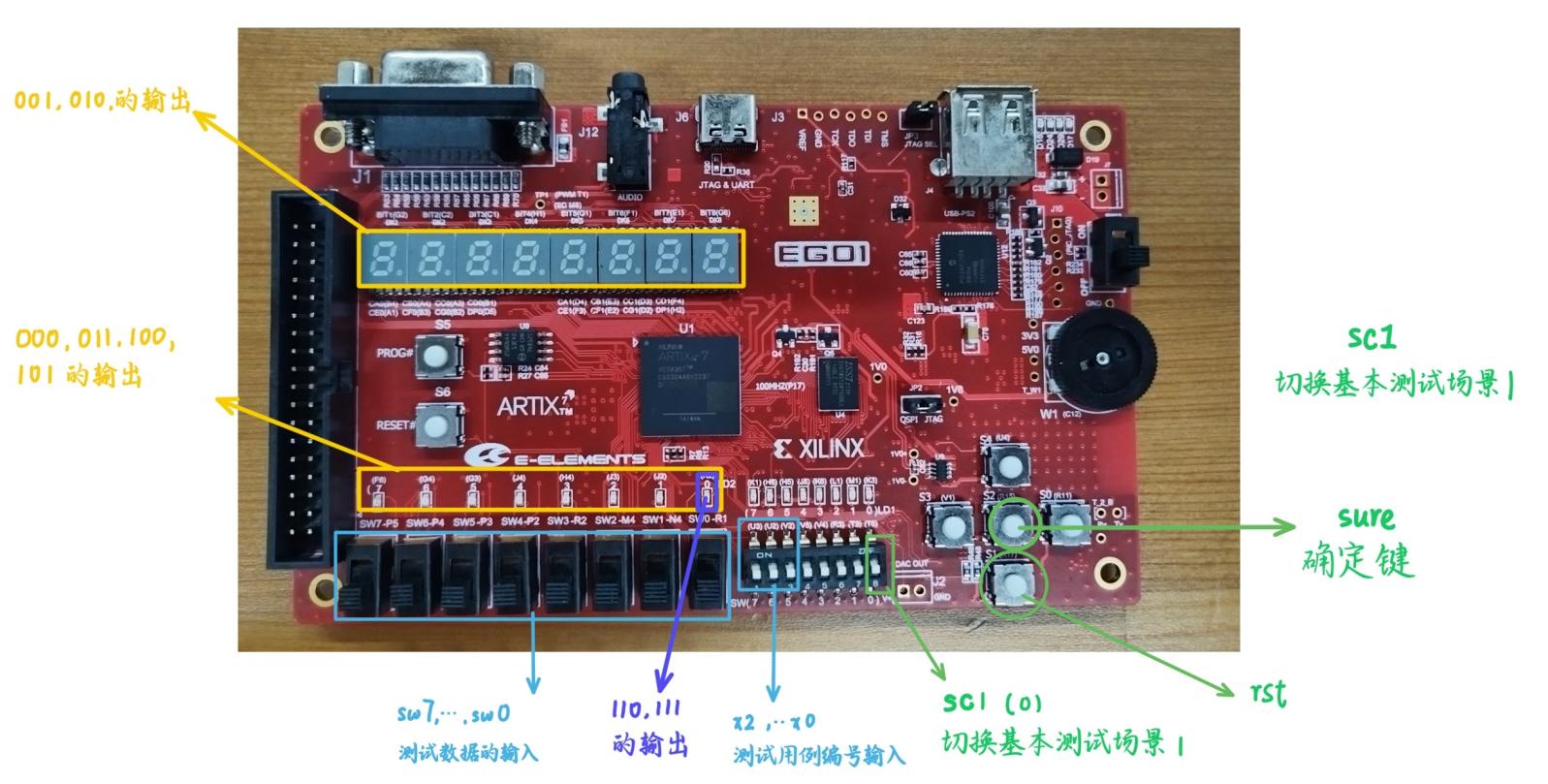
软件实现基于现有 CPU 的指令集架构ISA，通过调用预定义的指令完成计算，无需关心电路细节。这种方式开发效率高，但性能受限于 CPU 的硬件能力（如吞吐量、指令延迟等），灵活性较低（例如在浮点数计算时无法自定义浮点格式或优化计算流程等）

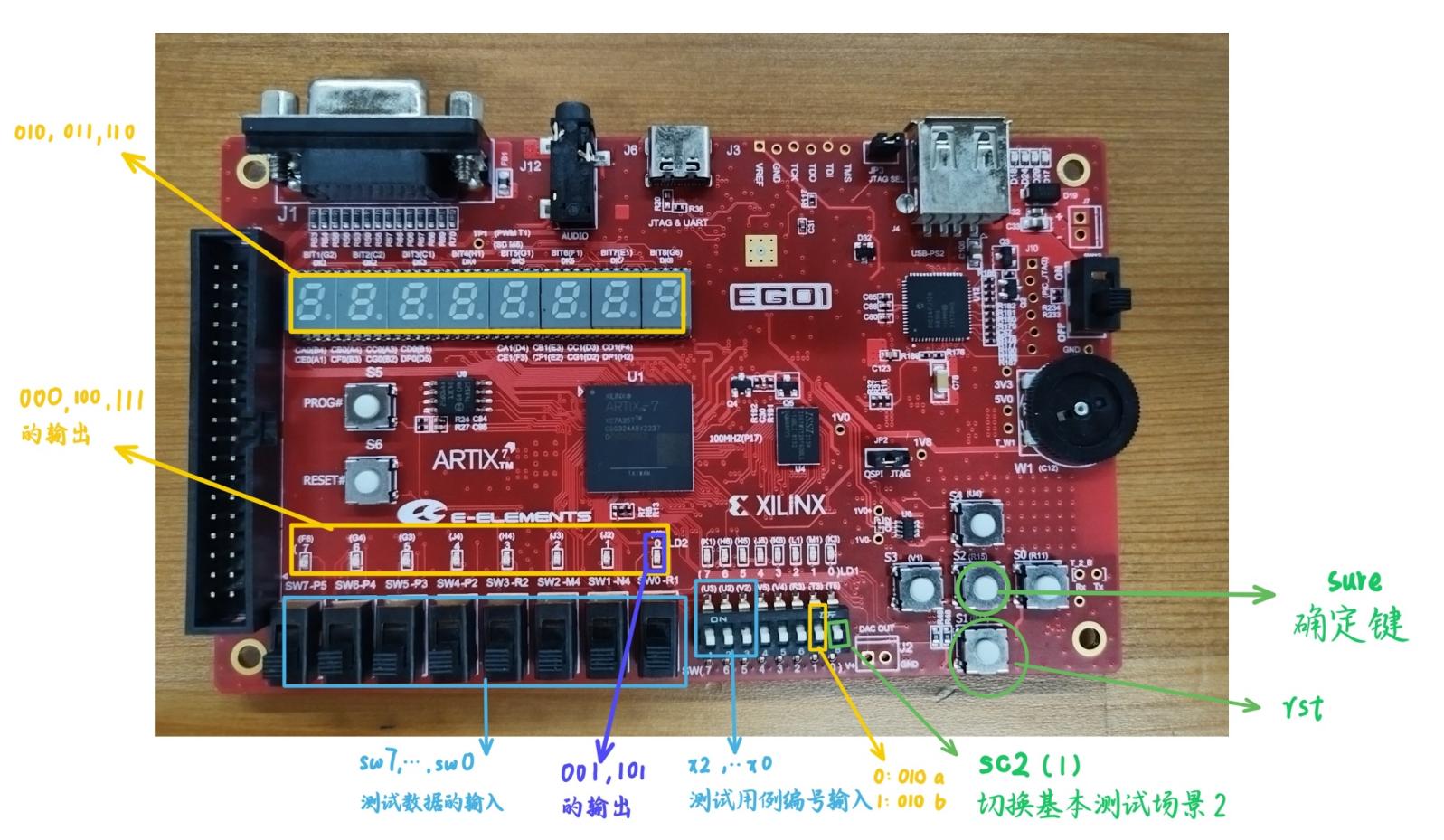
以浮点数相加后取整为例：硬件实现可以设计专用逻辑，直接送入加法器，再截断浮点小数位，甚至简化数据路径以减少时钟周期；而软件实现则需分步执行：先转换回浮点，再相加，最后调用指令取整，整个过程可能涉及多次寄存器访问并存在流水线停顿，效率较低。

混合架构（如自定义ISA）能结合两者优势：在硬件层面增加专用指令，再通过汇编调用，既提升性能又保持编程灵活性。最终选择需权衡开发成本、性能需求及可维护性。

在我们的CPU设计中，采取的是软件实现，例如在汇编语言中完成个12bit位宽的IEEE754编码的解析，浮点数的相加与取整，易于修改，灵活性更高。

**系统上板使用说明：**



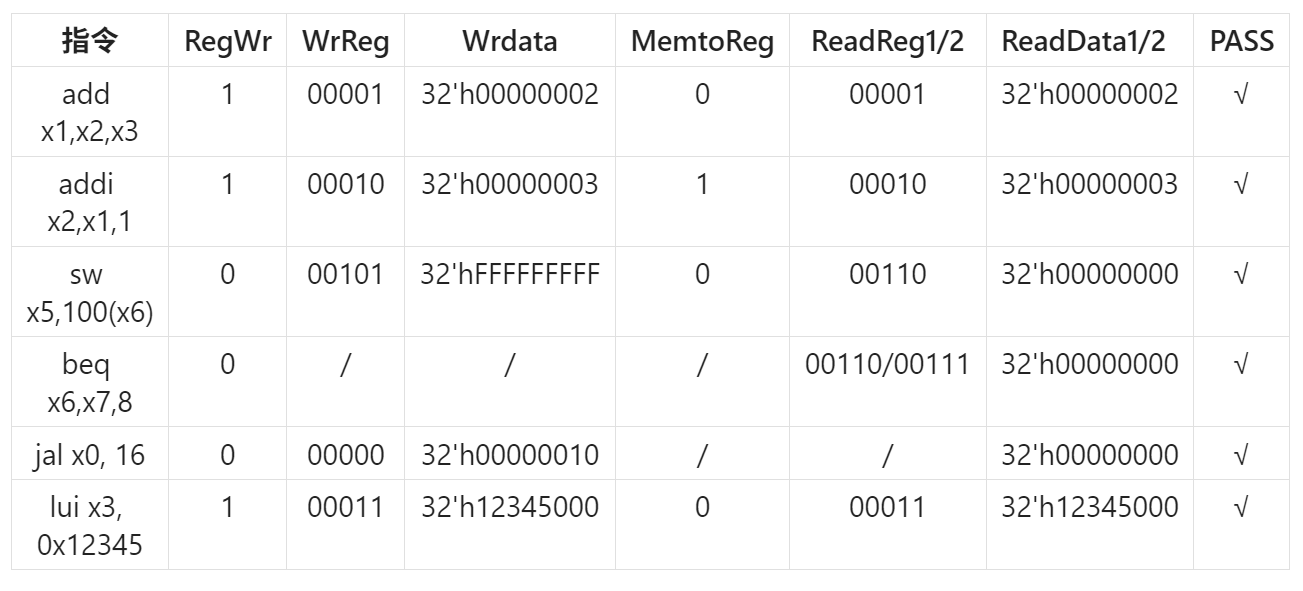


**自测试说明：**

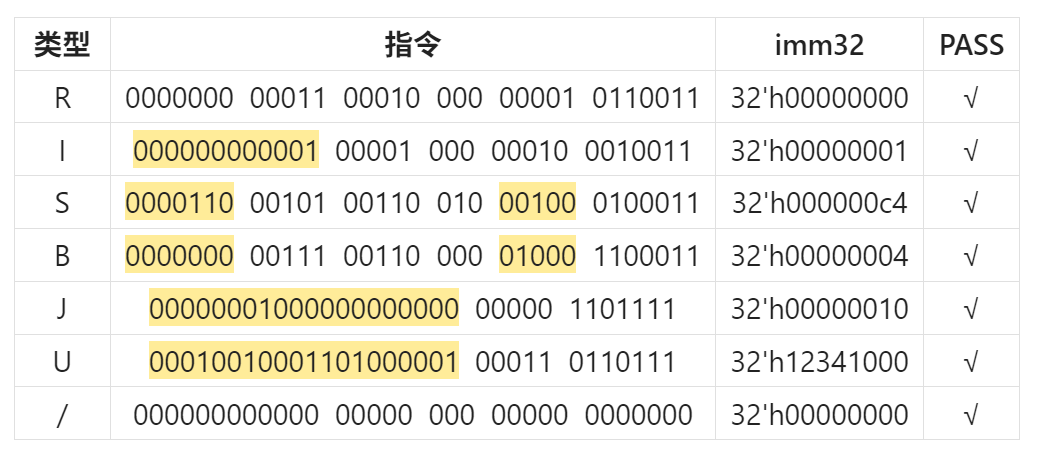




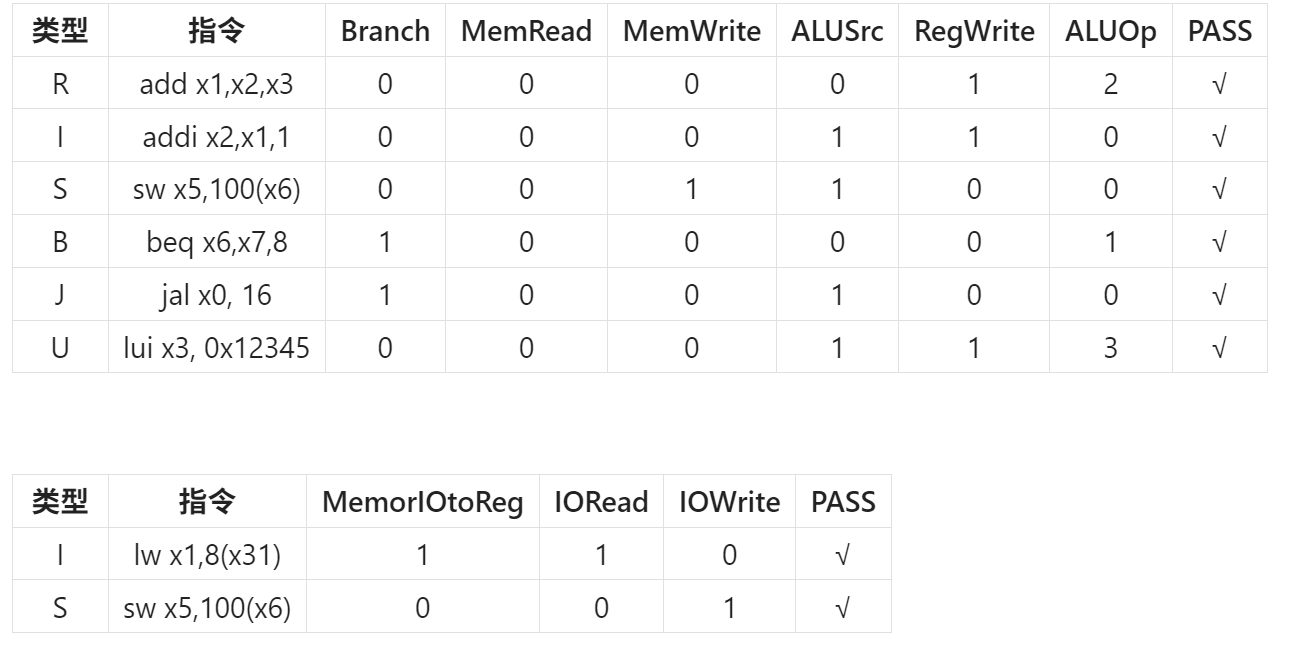
Registers\_tb:



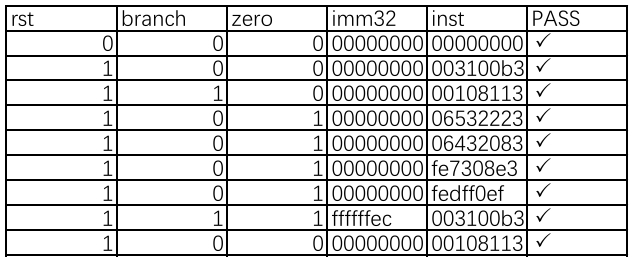
Imm\_gen\_tb:



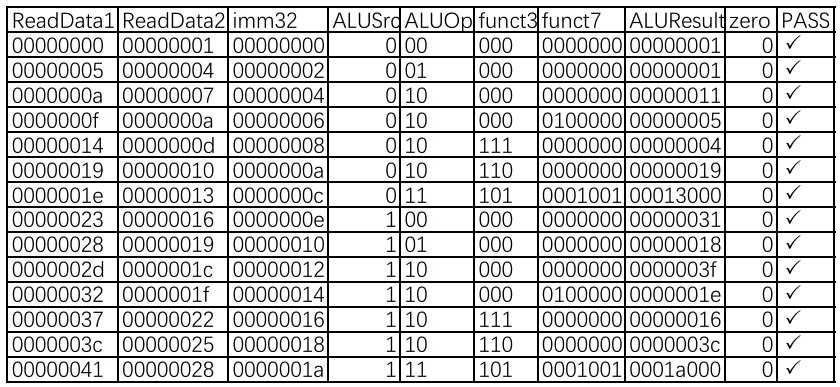
Controller\_tb:



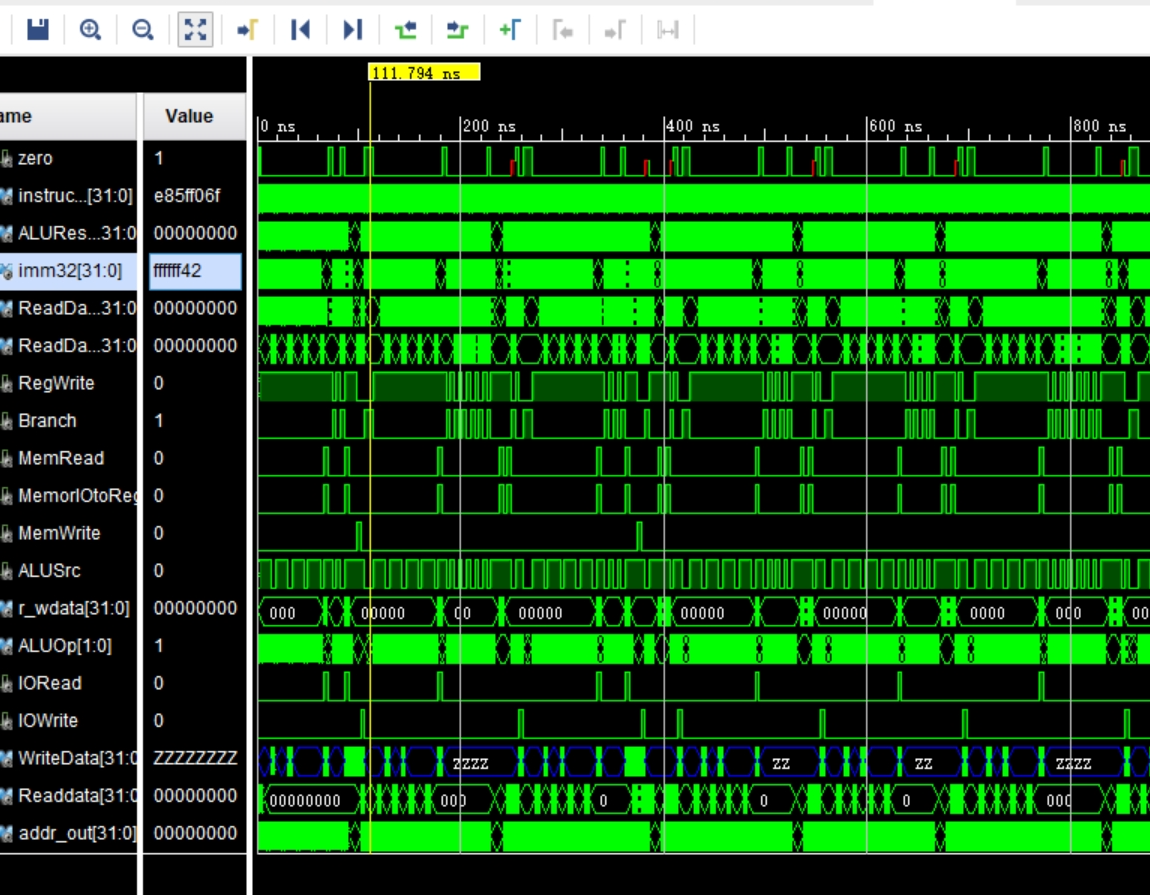
Ifecth\_tb:



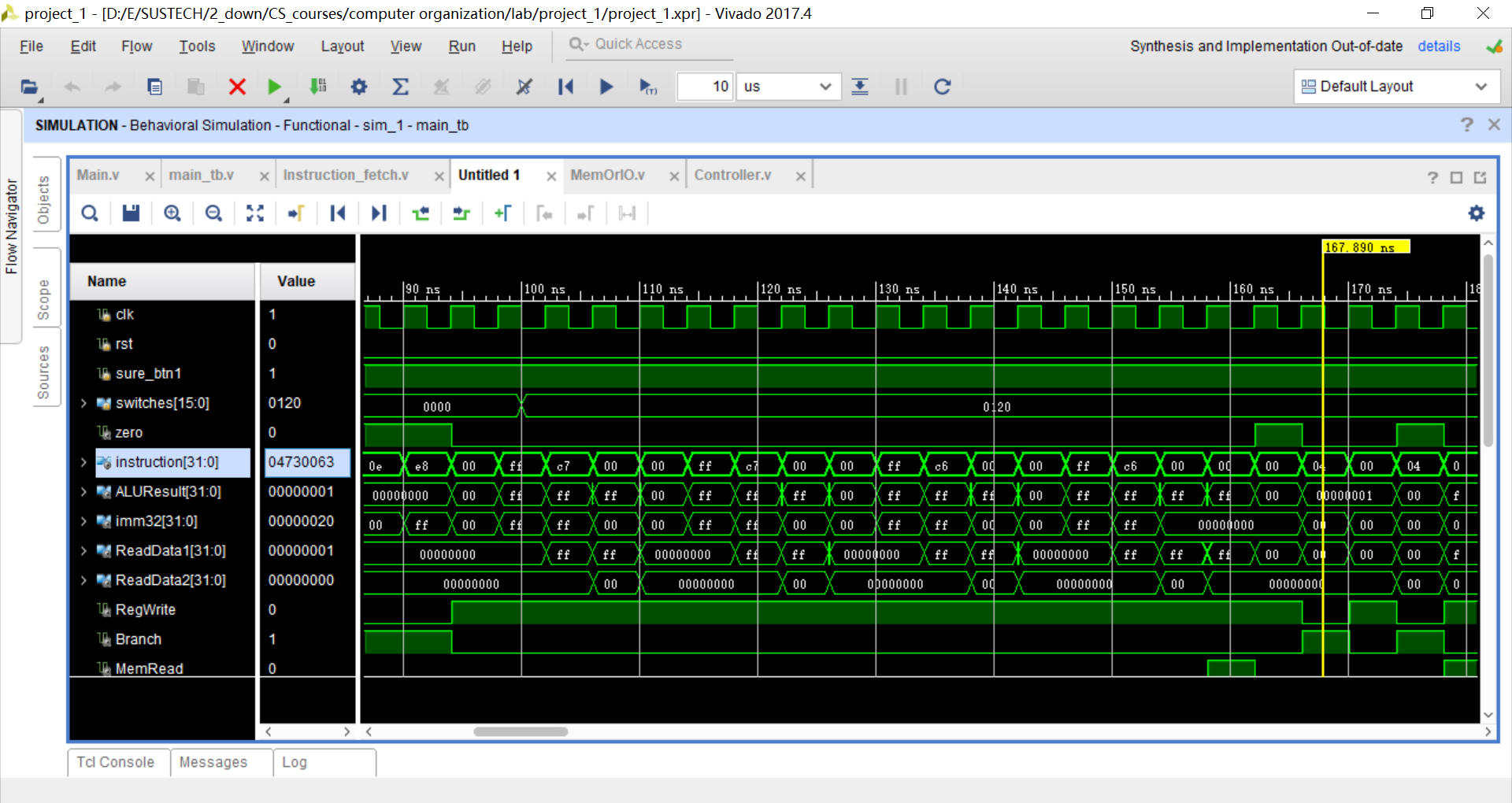
ALU\_tb:



CPU设计集成仿真：



测试场景一二集成仿真：



**开源及AI对于本次大作业的启发和帮助：**

MemOrIO和ioread以及leds子模块基于实验课课件生成，根据仿真结果以及CPU子模块之间的连线，调试端口的连接方式。

汇编代码中使用andi 指令，按位与，保留或提取特定位。srl 指令，逻辑右移，提取特定位。

**问题及总结：**

1. 单一接口不足以满足beq，bne，blt，bltu等同类型指令的判断与跳转，需要添加其它对应的接口。
2. MMIO模块中因为输出设备地址的选择，led灯与数码管的状态出现频闪，在硬件中修改为保持原状态或者在软件中修改为不同时点亮解决。
3. 开发过程中，熟悉了CPU的各个模块以及连接，在基础的CPU上添加了更复杂更多的指令的实现，熟悉了IP核的应用，汇编语言的使用以及汇编语言的逻辑内核。
4. 在CPU的结构，汇编文件的测试下，成功构建的完整的CPU系统。