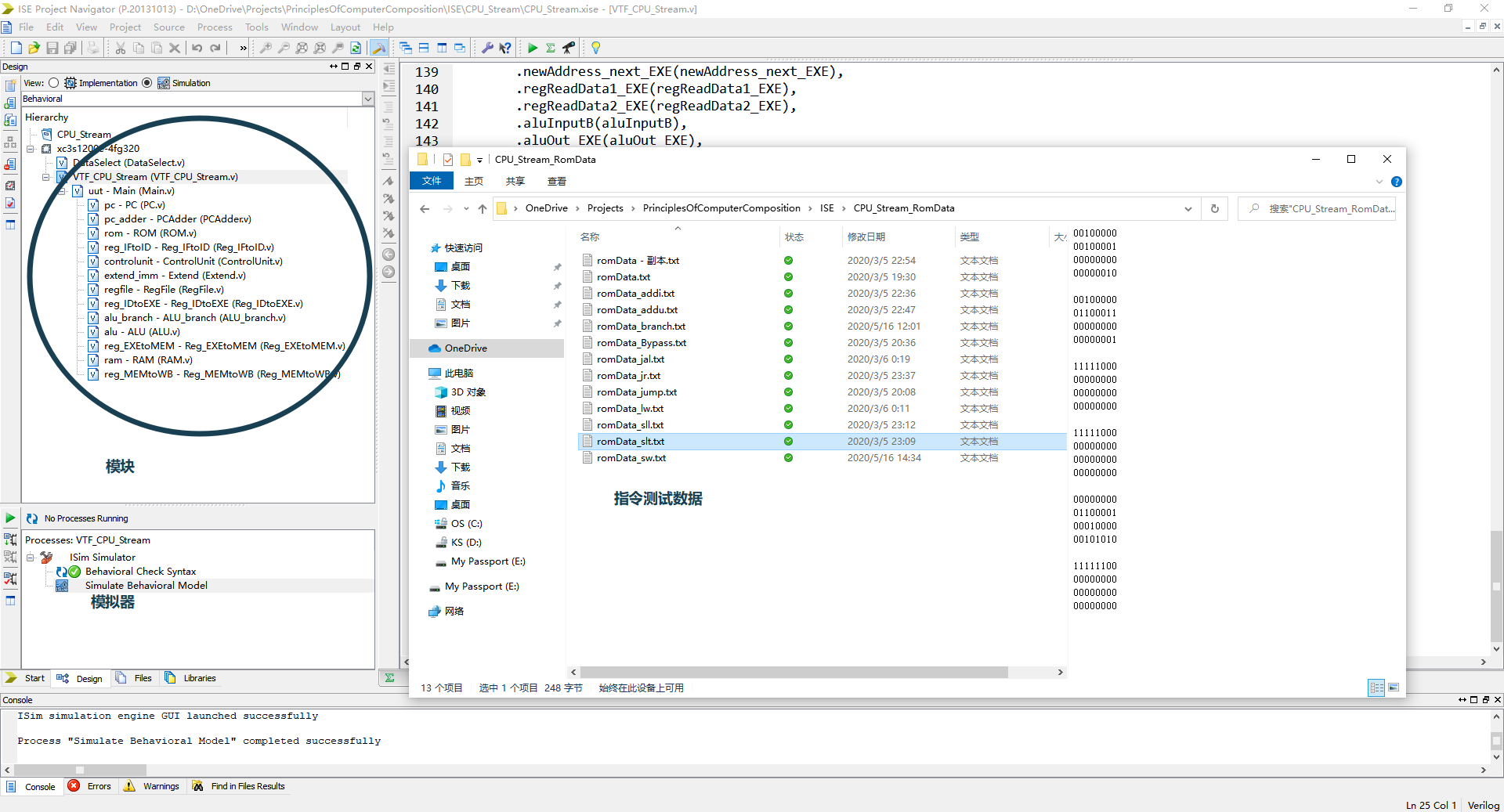
**作品简介：**

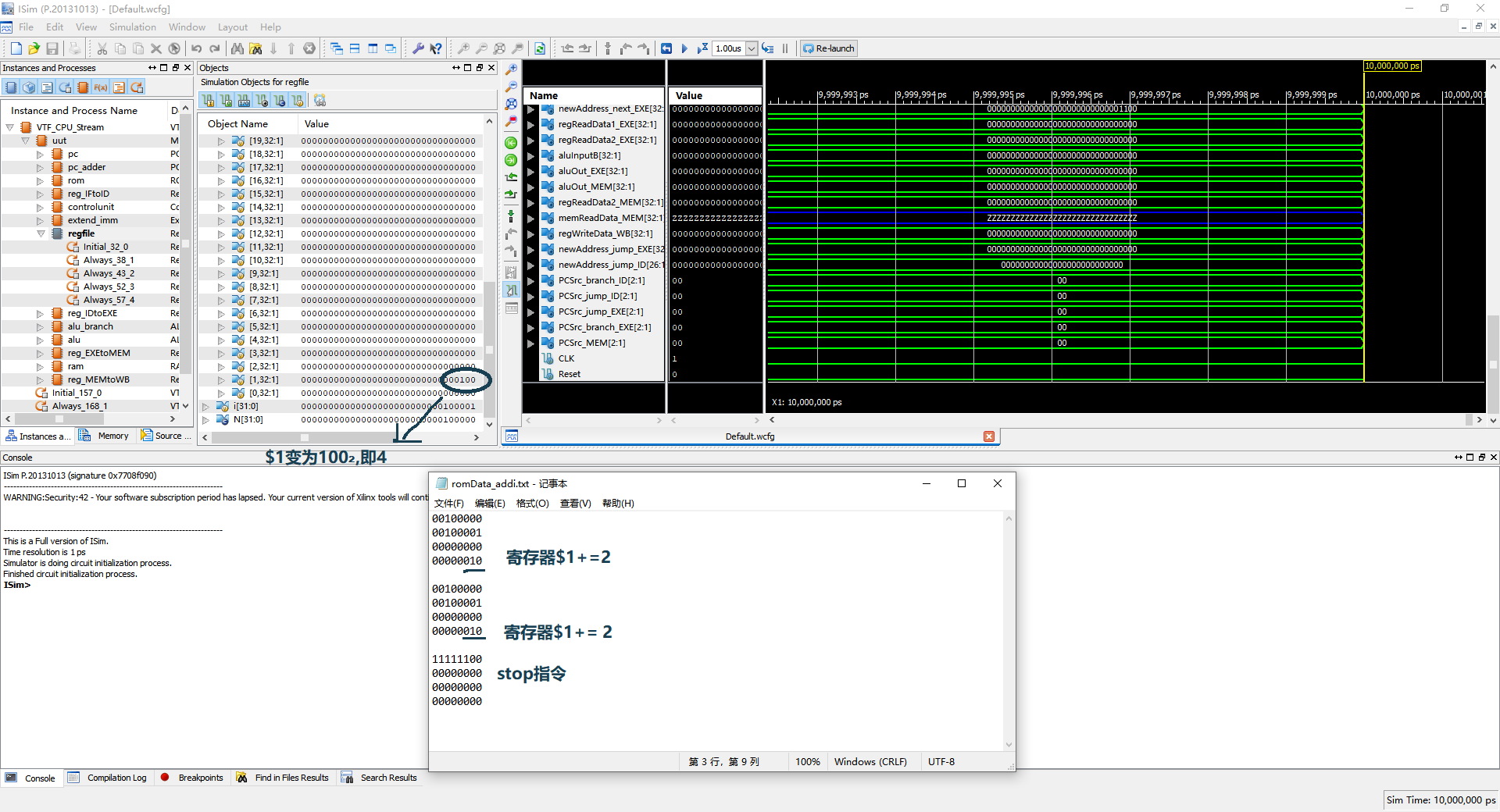
面向MIPS32指令集设计的流水式单线程CPU。进阶支持部分数据旁路功能。

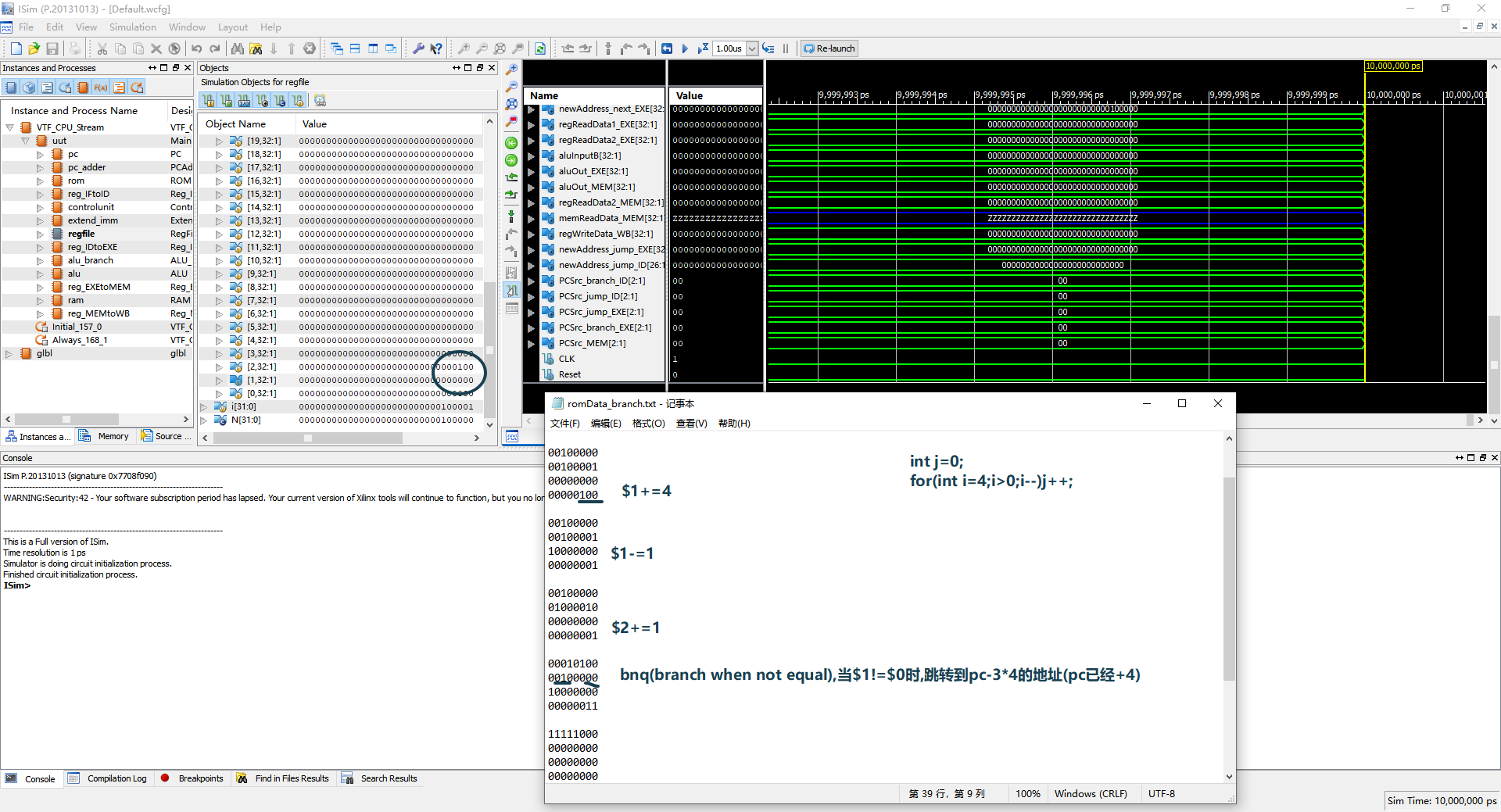
借助ISE Design Suite等工具模拟CPU运行和存储系统的实现及其余事项。

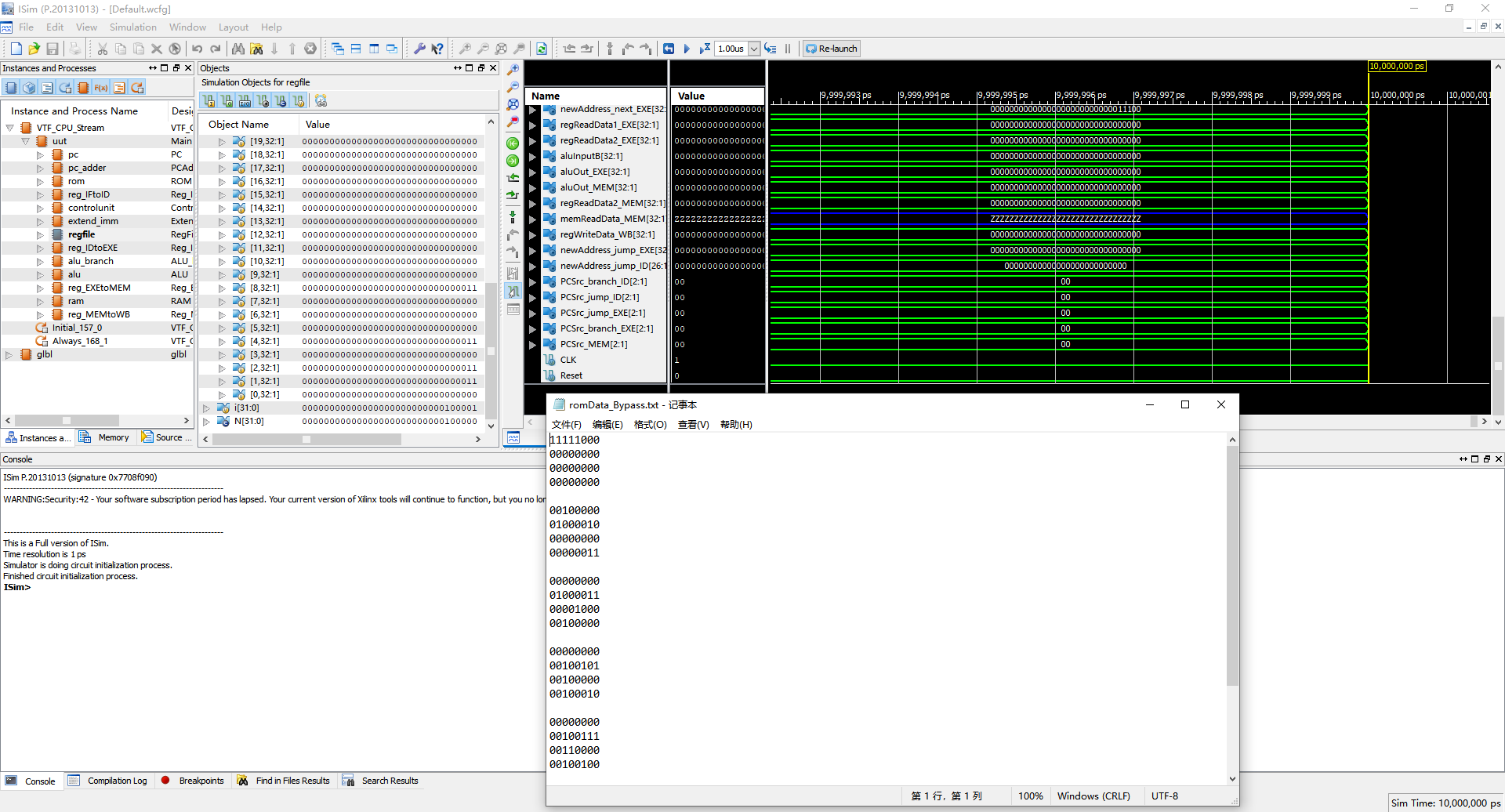
现代硬件设计越来越趋同于程序设计。在VHDL与Verilog两大主流硬件设计语言中，选择Verilog实现产品，因其与c语言高度相似，便于管控、移植。

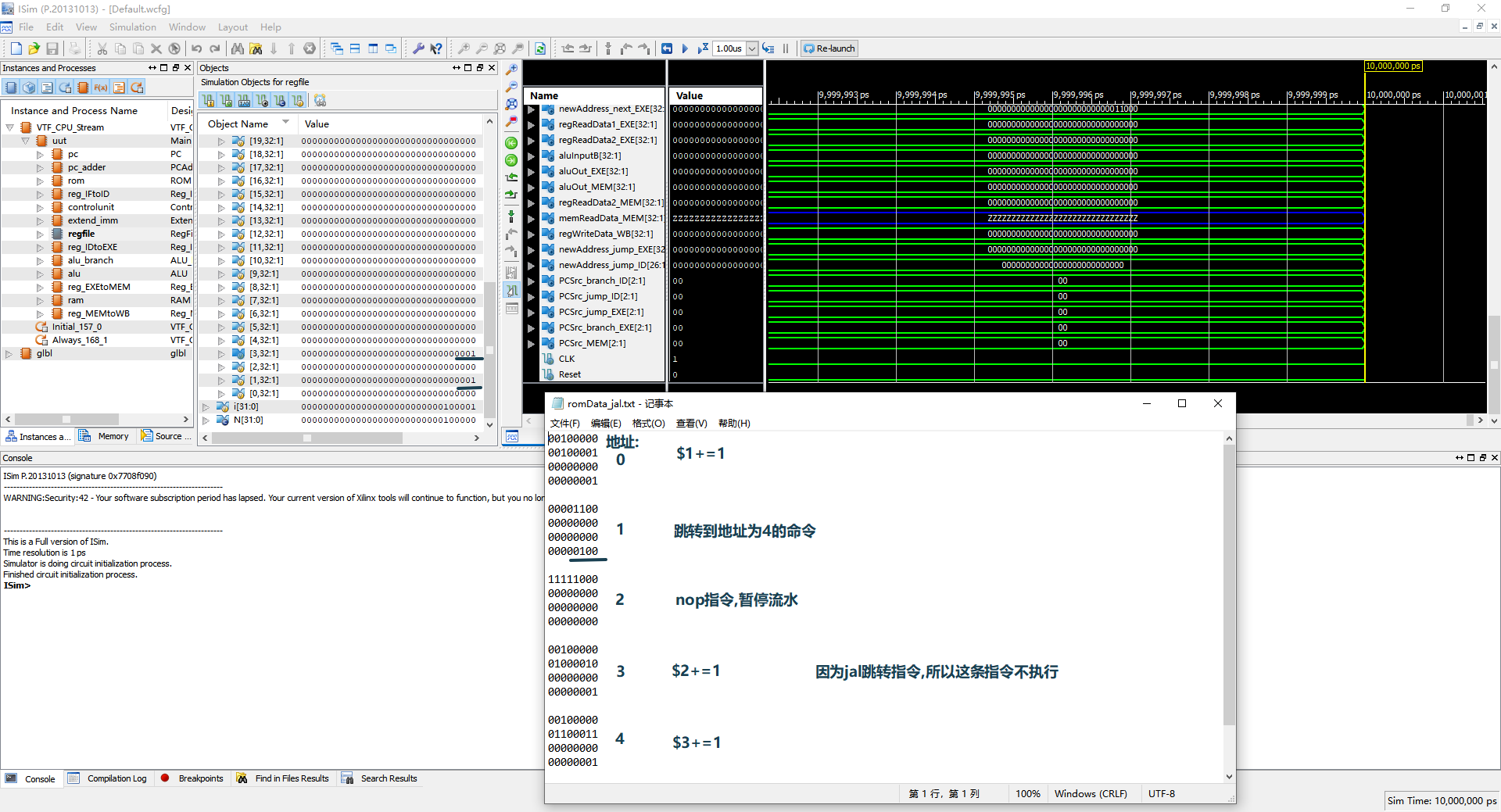
**作品效果图:**

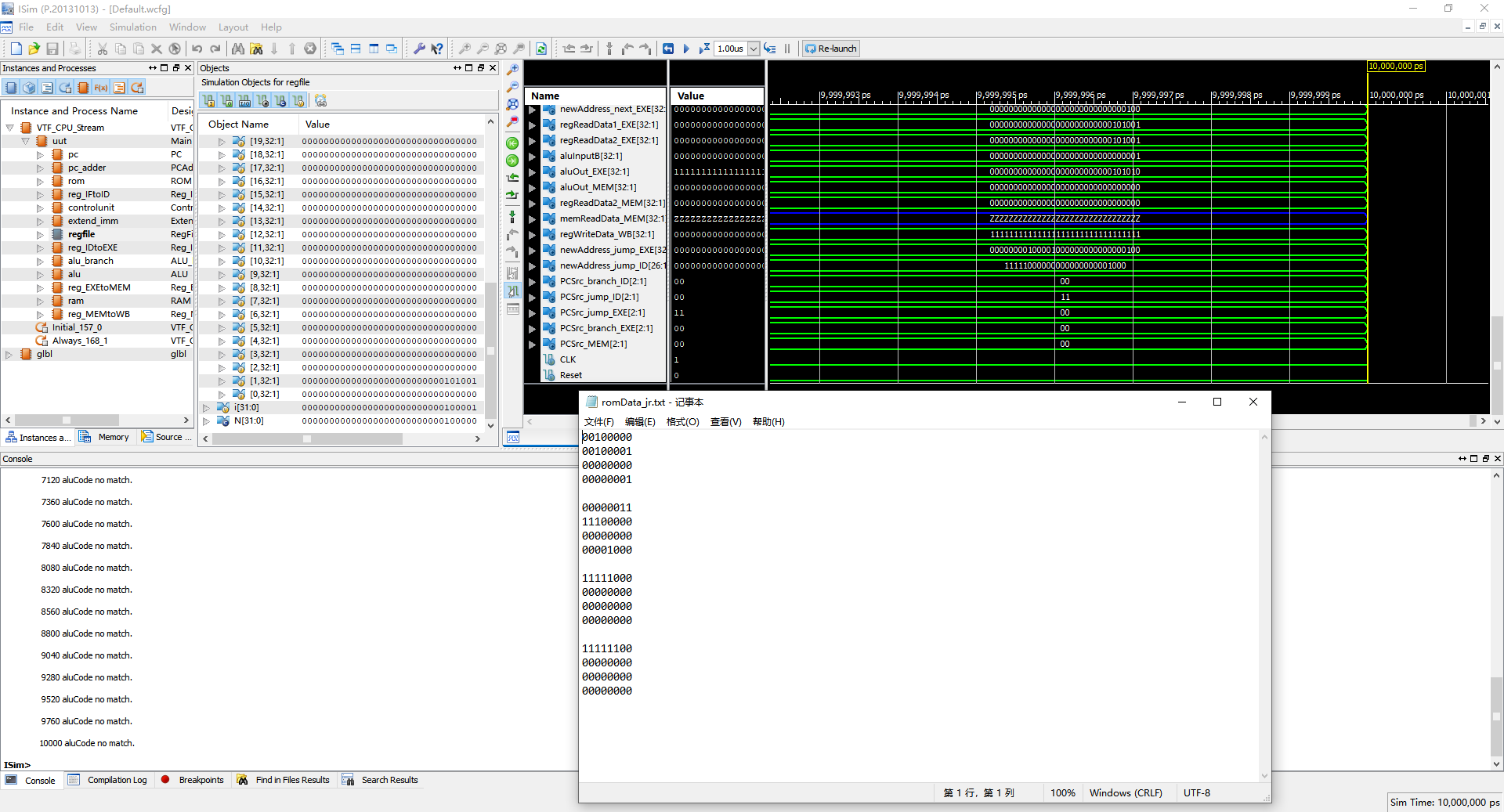
****

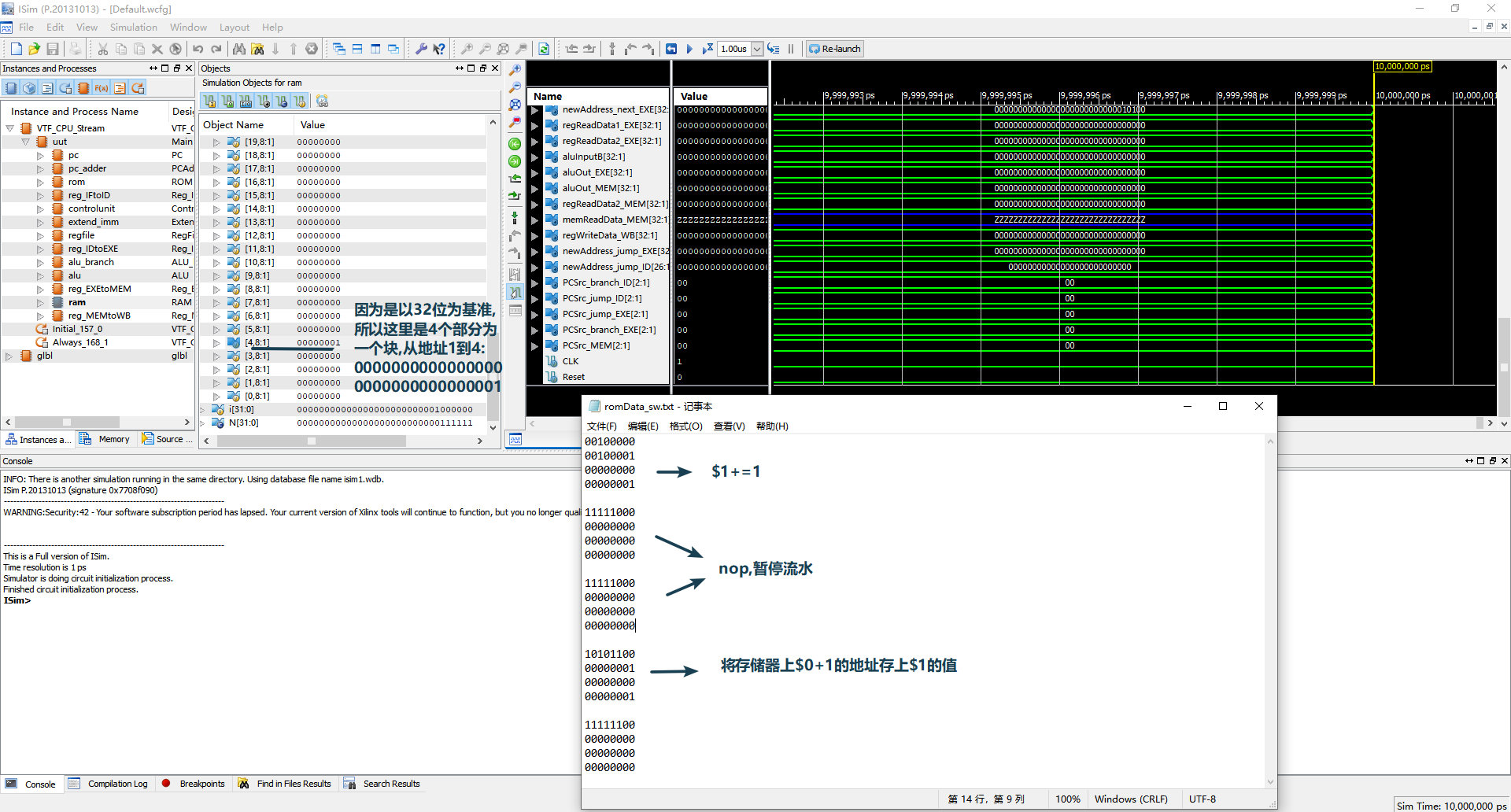
****

****

****

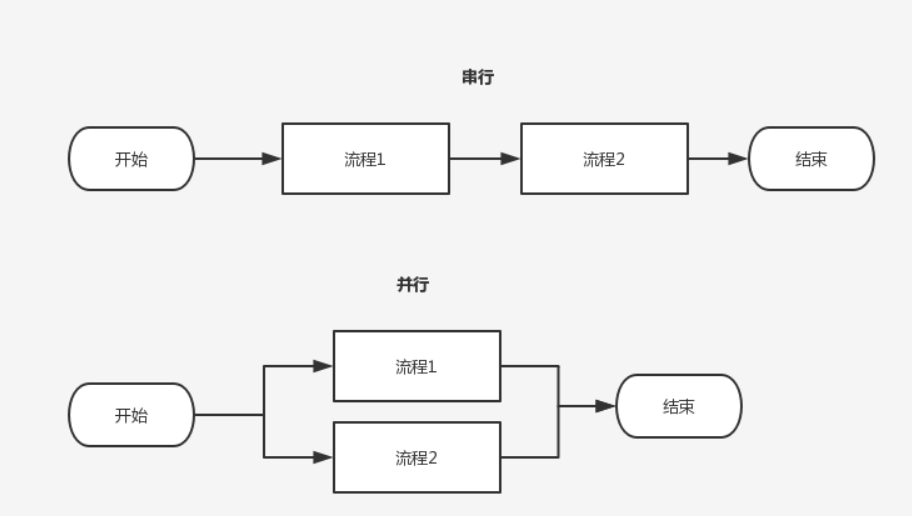
****

****

****

**设计思路:**

Cpu主要分为单周期,多周期和流水式cpu。现阶段主流为流水式cpu,它具有并行处理的特点(单/多周期cpu为串行,需要一整条指令结束后才执行下一条指令),每个元器件可以单独工作(即同一时间各个元器件可能在处理不同的指令),保证了执行效率。基于此,我们选择了流水式cpu设计作为参赛作品。



图表 1串行与并行

流水式cpu因其并行处理的特点,设计难度高于单/多周期cpu。本作品选用的解决方案主要为流水寄存器,并辅以数据旁路技术来提高其性能。

MIPS32是较为主流的架构,是典型的RISC(Reduced Instruction Set Computing)指令集(精简指令集),总共只有30多条指令。它便于实现和维护,为流水线而设计,所以选用它作为cpu的指令集。

(三大架构:MIPS,ARM,X86)

流水式cpu需要解决单/多周期cpu所没有的状态转移问题,我们使用流水寄存器处理它。MIPS指令架构一般将一条指令的执行分为5个阶段：取指，译码，执行，存储，写回。每个阶段占用一个周期。流水寄存器即在每两个阶段之间设置寄存器组，以传递状态，从而让指令流动起来。

并行处理和指令流动会导致某些数据冲突，如上一条指令为$1+=1，下一条为$2=$1，因并行处理且数据写回阶段处于最后，这时会出现$1所存值尚未改变而$2却要使用改变后的$1的值的问题。最简单的解决方法为暂停流水，等待写回后再执行下一条指令，但该方法将严重降低cpu效率。数据旁路技术可以很好地解决该问题，所以我们采用此方法。

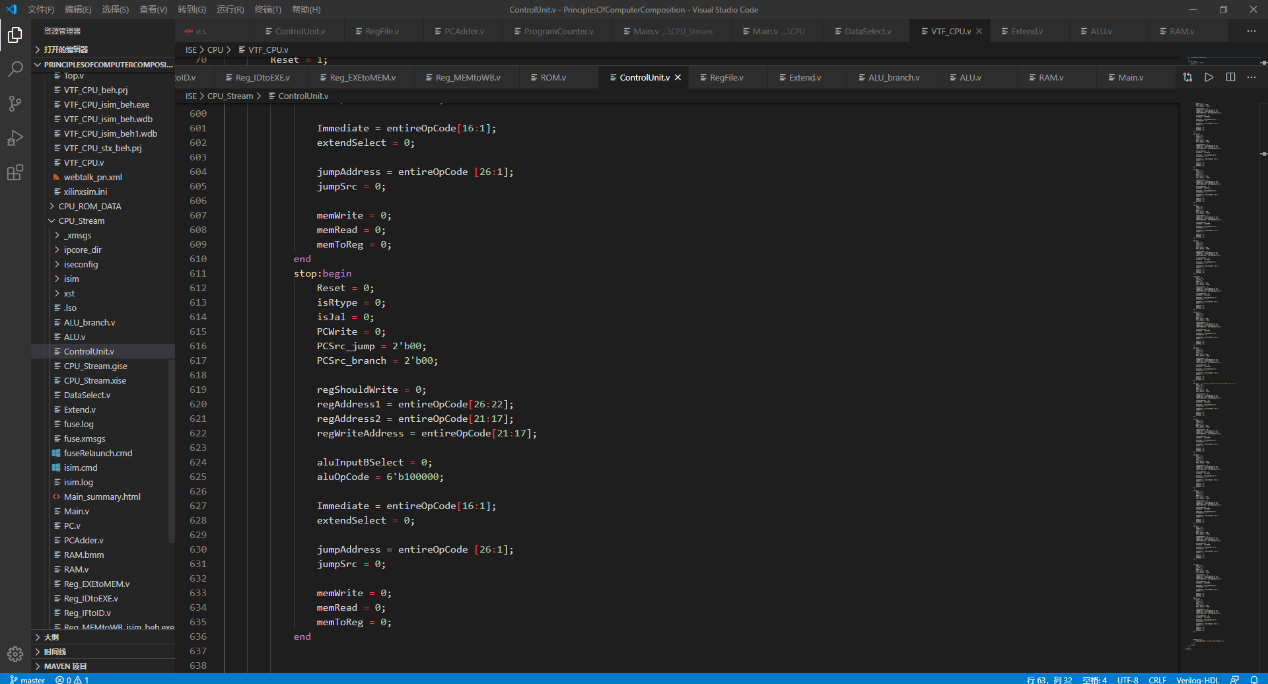
现阶段硬件设计无需考虑与或非门等底层级别的电路实现，借助硬件设计语言可以快速完成硬件设计。当前主流硬件设计语言为VHDL和Verilog，国内Verilog使用广泛，其语言构造与程序设计语言相似。

因暂无相应的程序编译工具，所有二进制指令均需人工编写。

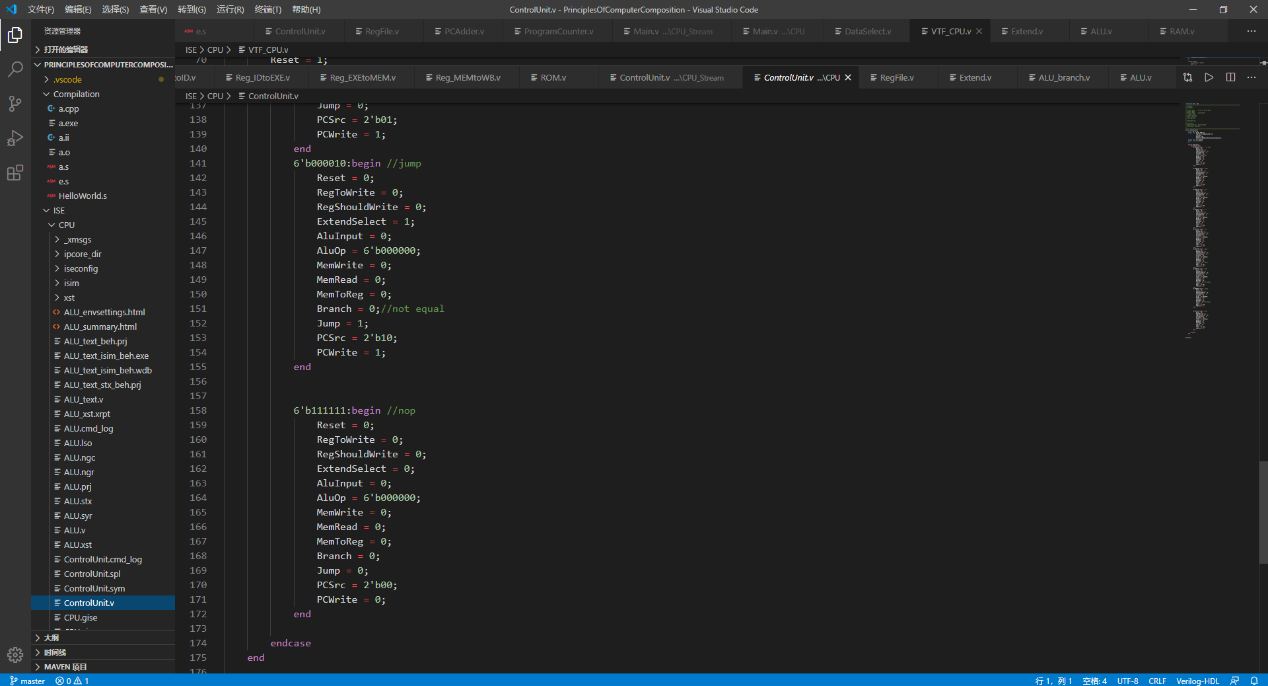
**作品重点难点:**

总体架构: cpu作为计算机的核心组件,需要统筹调控整个系统,包括输入输出设备,数据总线,存储系统等部件。产品设计时,需要创建完备的构造结构,难度不亚于大型软件开发。

控制器组件: cpu中的控制器作为中枢,管理cpu各个部件的运作。并且因流水cpu的并行处理特点,其控制器设计难度远大于单/多周期cpu。串行式cpu控制器只需要控制当前指令的所有控制信号，而并行式cpu控制器需处理每个阶段的控制信号，每个阶段执行不同的指令。



图表 2 流水cpu控制器



图表 3 单周期cpu控制器

数据/控制冲突问题和数据旁路技术:MIPS架构将一条指令的执行分为5个阶段,分别为取指,译码,执行,存储,写回。因其写回阶段处于最后,流水式cpu又为并行处理,所以这种架构有不少数据冲突问题需要解决。需要读取数据的第一个阶段为译码阶段,写回阶段位于其后第3个阶段。已知的数据/控制冲突有:寄存器读写冲突,寄存器-存储器读写冲突,存储器读写冲突,跳转指令冲突(过于复杂,暂停流水线)。

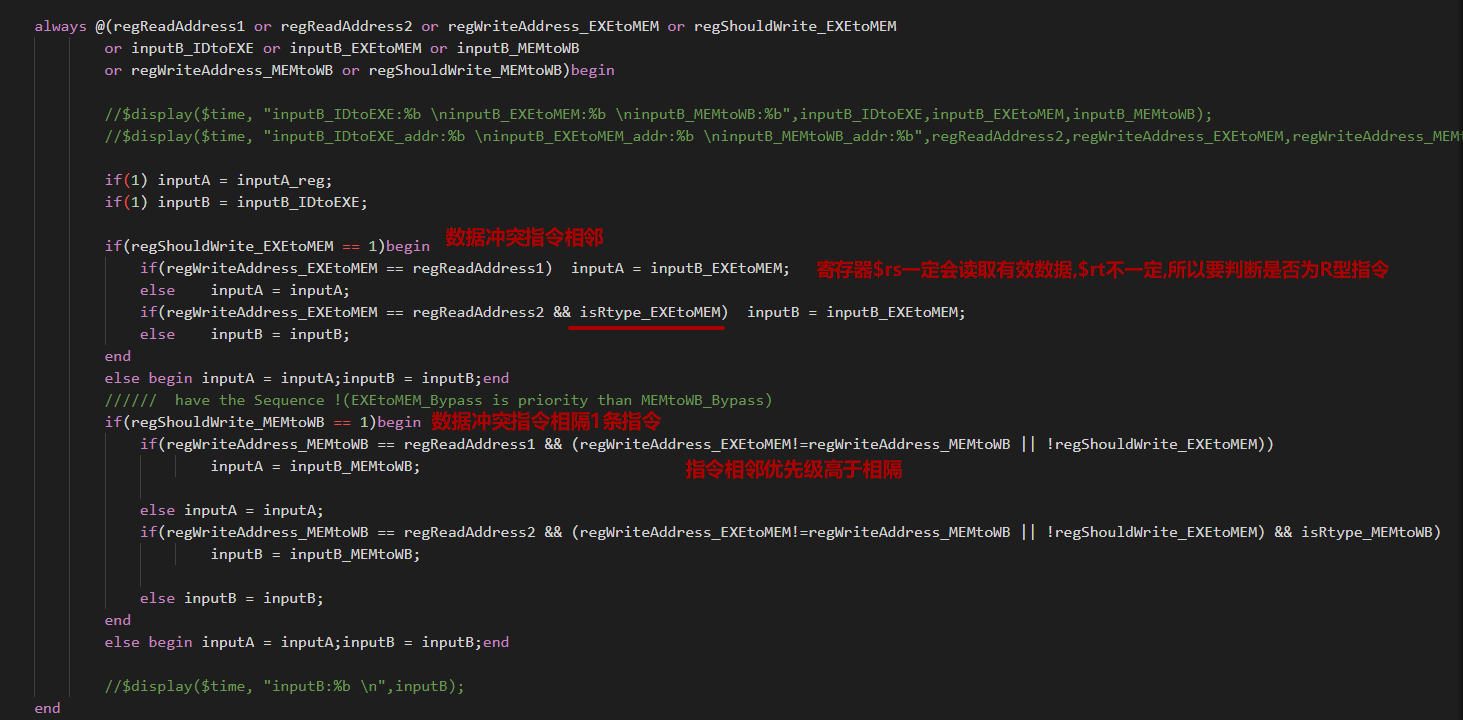
1. 寄存器读写冲突:考虑以下情况:

**指令1将$1+=1,指令2,3为空,指令4为$2=$1.**

当指令1为写回阶段时,指令4为译码阶段,需要读取$2的值.这时只需保证寄存器先写后读即可.我们将数据写回置于上升沿(posedge),数据读取置于下降沿(negedge),一个周期总是从上升沿开始,经过下降沿,再从上升沿到下一个周期。

**指令1将$1+=1,指令2为$2=$1.**

这时指令2为执行阶段,指令1仅到存储阶段.我们采用增加数据旁路来处理此问题,即在存储阶段增加数据通路至执行阶段,以更新最新的数据.数据旁路需要考虑较多问题:如何知道使用的数据被更新过,如何知道是否为无效数据。将需要更新的寄存器地址保持传输,在存储阶段传回与执行阶段所需使用的寄存器地址对比,若相同且寄存器写控制信号为true,则更新数据。



(解决寄存器读写冲突的数据旁路)

**指令1将$1+=1,指令2为其他,指令3为$2=$1.**

与上一种情况类似,这时指令1为写回阶段,指令3为执行阶段,注意,数据的读入在译码阶段已完成,所以指令3仍使用旧数据,需要旁路更新.处理方法与上一种情况类似,但优先级低于它(考虑指令1为$1=1,指令2为$1=2,指令3为$2=$1,最后使用应为$2=$1=2.).

1. 寄存器-存储器读写冲突:

**指令1为$1=1,指令2为$1数据存入存储器地址1.**

可处理,暂定为暂停流水线.

**指令1为存储器地址1数据存入$1,指令2为$2=$1.**

指令1为存储阶段时,指令2已经到达执行阶段,当完成存储器读取,执行阶段也已完成,所以该情况只能暂停流水线.

**指令1为存储器地址1数据存入$1,指令2为空,指令3为$2=$1.**

指令1为存储阶段,指令3为译码阶段,可增加数据旁路解决冲突

1. 存储器读写冲突:

先写后读即可.(若暂停流水线,则仍可先读后写)

**素材,源码说明:**

素材:

图表1: <https://www.cnblogs.com/gdufs/p/6831685.html>

未说明素材均为原创

源码:

待提供