**计算机系统结构试验**

**Lab05: 简单的类 MIPS 单周期处理器的实现**

姓名：N/A

摘要

在Lab05中，我实现了MIPS单周期处理器的主要功能，支持31条指令的译码、执行功能，并在此基础之上实现了拨码乘法器。实验包括了Program Counter、Branch Control、Immediate Extension等功能的设计和实现，以及综合各部件的Top模块。通过本次实验，我进一步加深了对Verilog语言的理解和运用，掌握MIPS处理器的组成、设计和实现方法，并且能够使用仿真工具进行验证和调试，给我带来宝贵的经验和收获。

目录

[摘要 1](#_Toc167661546)

[**1.** 实验目的 3](#_Toc167661547)

[**2.** 原理分析 3](#_Toc167661548)

[2.1 Vivado工程的基本组成 3](#_Toc167661549)

[2.2 ProgramCounter模块的原理 4](#_Toc167661550)

[2.3 InstructionMemory模块和DataMemory模块的原理 4](#_Toc167661551)

[2.4 ImmediateExtension模块的原理 4](#_Toc167661552)

[2.5 Control模块，ALUControl模块和BranchControl模块的原理 4](#_Toc167661553)

[2.6 Registers模块的原理 4](#_Toc167661554)

[2.7 ALU模块的原理 4](#_Toc167661555)

[**3.** 功能实现 5](#_Toc167661556)

[3.1 ProgramCounter模块的实现 5](#_Toc167661557)

[3.2 InstructionMemory模块和DataMemory模块的实现 5](#_Toc167661558)

[3.3 ImmediateExtension模块的实现 6](#_Toc167661559)

[3.4 Control模块，ALUControl模块和BranchControl模块的实现 7](#_Toc167661560)

[3.5 Registers模块的实现 8](#_Toc167661561)

[3.6 ALU模块的实现 8](#_Toc167661562)

[3.7 Top模块的实现 9](#_Toc167661563)

[**4.** 结果验证 9](#_Toc167661564)

[4.1 乘法代码的测试 9](#_Toc167661565)

[4.2 其他指令的测试 10](#_Toc167661566)

[4.2.1 beq，bne指令 10](#_Toc167661567)

[4.2.2 j，jr，jal指令 10](#_Toc167661568)

[4.2.3 sll，srl，sra指令 11](#_Toc167661569)

[4.2.4 addi，addiu，lui指令 11](#_Toc167661570)

[4.2.5 slt，sltu指令 11](#_Toc167661571)

[4.3 拨码乘法器的测试 12](#_Toc167661572)

[**5.** 总结与反思 13](#_Toc167661573)

**1.** 实验目的

（1）理解简单的类MIPS单周期处理器的工作原理(即几类基本指令执行时所需的数据通路和与之对应的控制线路及其各功能部件间的互联定义、逻辑选择关系)；

（2）完成简单的类 MIPS 单周期处理器；

1）9条MIPS指令（lw, sw, beq, add, sub, and, or, slt, j）CPU的实现与调试。2）拓展至16条指令（增加addi, andi, ori, sll, srl, jal, jr）CPU的实现与调试。

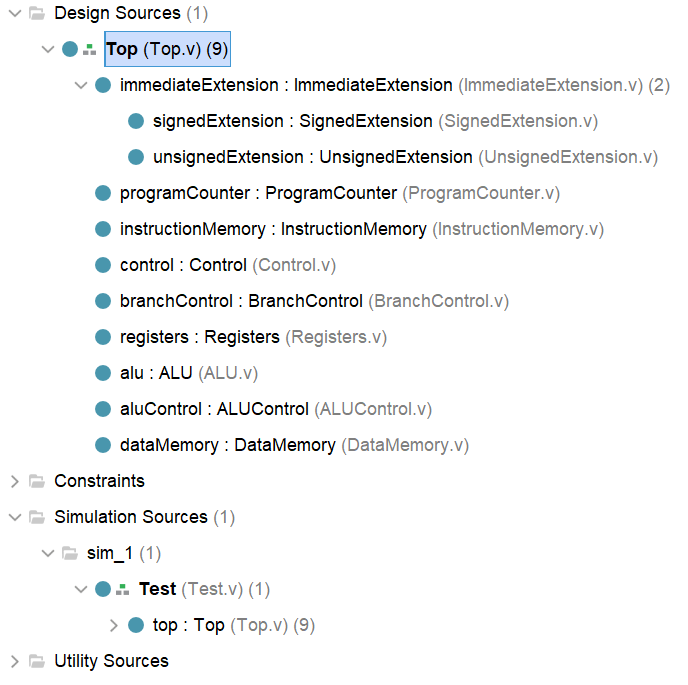
（3）乘法代码的仿真测试；

（4）上板验证拨码乘法器。

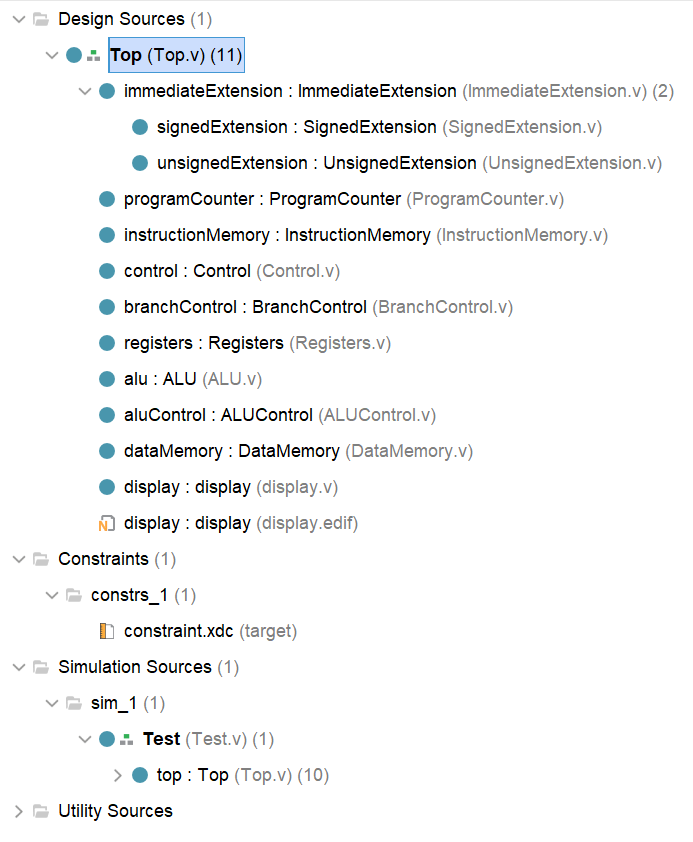
**2.** 原理分析

2.1 Vivado工程的基本组成

仿真测试的Vivado工程的结构如下：



拨码乘法器的Vivado工程的结构如下：



2.2 ProgramCounter模块的原理

Program Counter (PC)模块是用于跟踪程序执行位置的关键组件，PC模块会保存当前正在执行的指令的内存地址。每执行完一条指令，PC会自动增加4字节，指向下一条要执行的指令。

当遇到跳转指令(如jal、jalr、beq等)时,PC会根据跳转地址更新自己的值,使程序跳转到目标位置继续执行。跳转地址由指令中的立即数解码得到，或者从31号寄存器ra读取。因此PC模块需要接收指令立即数、31号寄存器以及各跳转控制信号的输入。

2.3 InstructionMemory模块和DataMemory模块的原理

InstructionMemory模块将指令内存中相应地址的数据读出，需注意PC按字节编址，而Memory按4字节编址，因此需要将输入PC右移2位。DataMemory在此基础之上还有写入功能，当接收到写使能信号时将读入数据写入相应位置。

2.4 ImmediateExtension模块的原理

ImmediateExtension模块将I-type指令中的立即数扩展到32位，由于指令的不同，立即数有无符号数和有符号数两种，因此分别实现了UnsignedExtension和SignedExtension两个子模块。模块通过指令的opCode判断扩展方式并输出相应结果。

2.5 Control模块，ALUControl模块和BranchControl模块的原理

Control模块进行指令的译码，R-type指令含义与funct字段有关，转发给ALUControl进行进一步判断，其余指令设置相应的regDst，ALUSrc，memToReg，regWrite，memRead，memWrite和ALUOp控制信号。

ALUControl模块根据ALUOp和funct字段译码，得到ALU控制信号，控制ALU的计算行为。ALUOp和ALU控制信号可以自行编写。

BranchControl模块使用opCode和funct字段判断当前指令是否属于跳转指令，并设置相应的控制信号位。

2.6 Registers模块的原理

Registers模块类似于DataMemory模块，但有两个地址输入和两个寄存器值输出。需要注意0号寄存器应当保持值为0不变，因此要忽略对0号寄存器的写入。

2.7 ALU模块的原理

ALU模块根据ALUControl模块约定的ALUCtr信号对两个输入进行运算，运算类型包括and，or，xor，nor，add，sub，slt，sltu，sll，srl，sra，sllv，srlv，srav，lui。由于sll，srl，sra指令需要读取shamt字段，ALU模块也需要一个来自指令的指令。

**3.** 功能实现

3.1 ProgramCounter模块的实现

Program Counter (PC)模块接收三种跳转的立即数：j和jal指令接收Instruction[25:0]，beq和bne指令接收有符号扩展的Instruction[15:0]，jr指令接收31号寄存器ra的值。

模块首先对jumpAddr和branchAddr进行译码。因为指令地址均为4的倍数，jumpImm首先要左移2位，jump指令会保留PC + 4的最高4位，因此这里加上 PC + 4的最高四位。branchImm同理左移2位，由于branch指令的立即数表示offset，因此加上PC + 4。

在时序部分，通过输入的控制信号以及ALU的zero flag进行相应的地址跳转，否则PC = PC + 4。

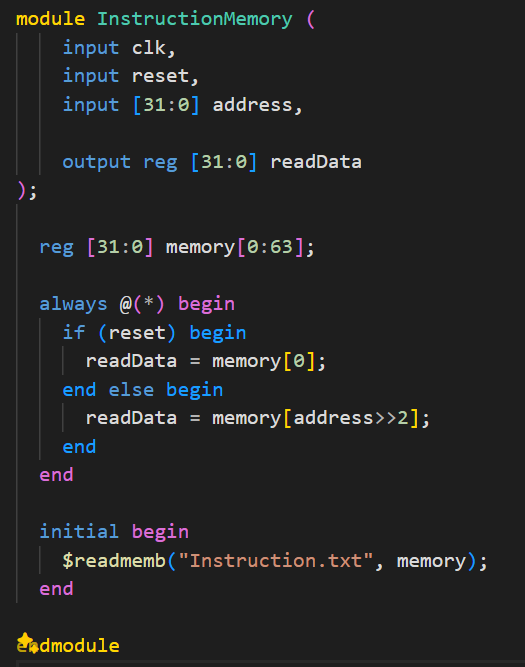
图形用户界面, 文本

描述已自动生成 文本

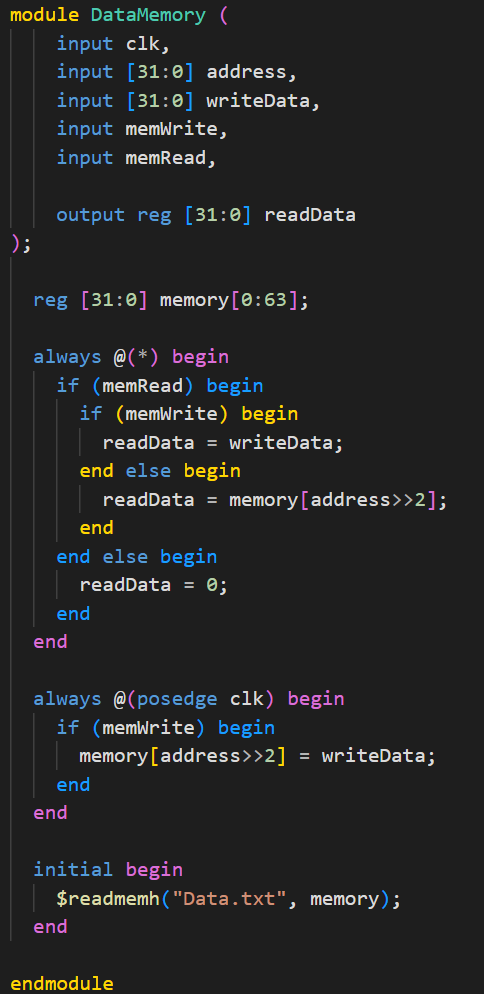
描述已自动生成

3.2 InstructionMemory模块和DataMemory模块的实现

InstructionMemory模块较简单，初始化时读入Instruction.txt的内容，代码如下：

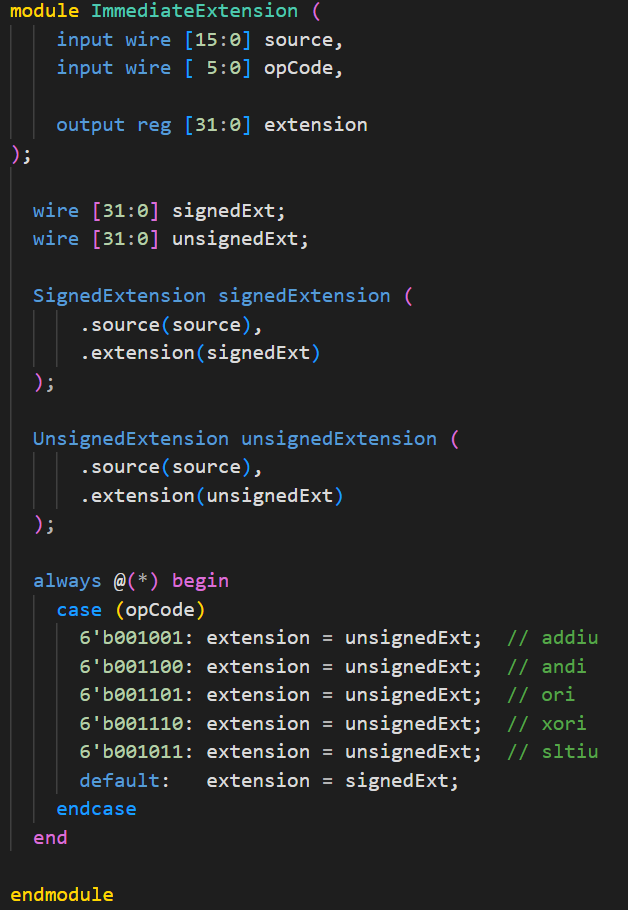


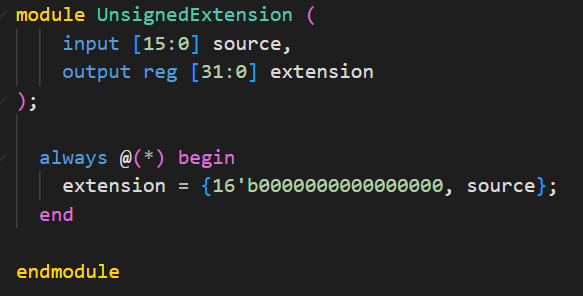
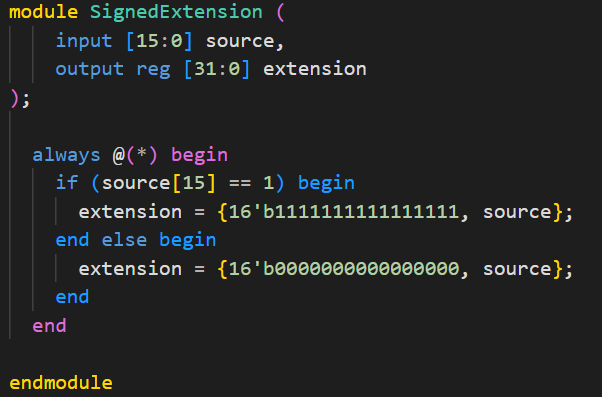
DataMemory模块需要注意hazard问题，读数据和其他部件不能在同一时钟沿，否则地址和数据将在同一时钟沿更新。此外，当read和write同时使能时，选择直接将写入数据forward到输出。代码如下：



3.3 ImmediateExtension模块的实现

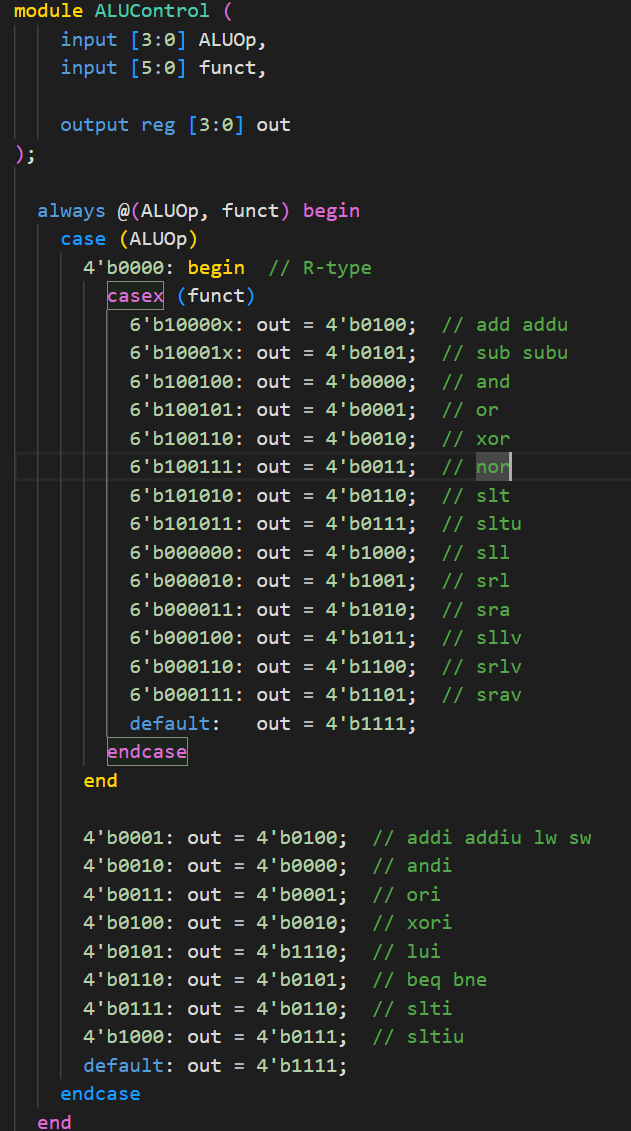
ImmediateExtension模块接收opCode的输入，判断指令中立即数应做无符号扩展还是有符号扩展。在所有指令中，addiu、andi、ori、xori和sltiu指令需要无符号扩展，其余指令均为有符号扩展，代码实现如下：



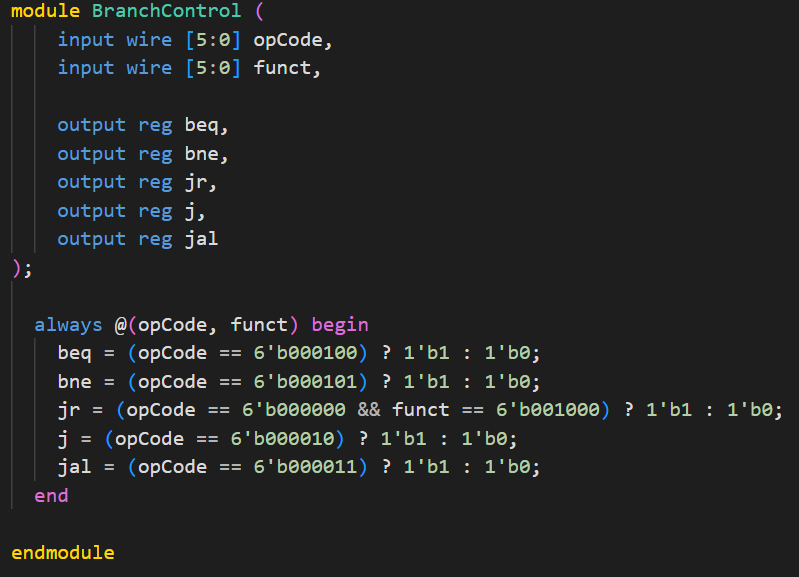


3.4 Control模块，ALUControl模块和BranchControl模块的实现

Control模块根据输入指令的opCode输出regDst、ALUSrc、memToReg、regWrite、memRead、memWrite控制信号以及ALUOp。ALUControl模块对于非R-type指令根据ALUOp选取ALUCtr，对于R-type指令根据funct字段选取ALUCtr。非R-type指令的ALUOp可以自行选取，只要和Control模块保持一致即可。Control模块代码此处省略，ALUControl模块代码如下：

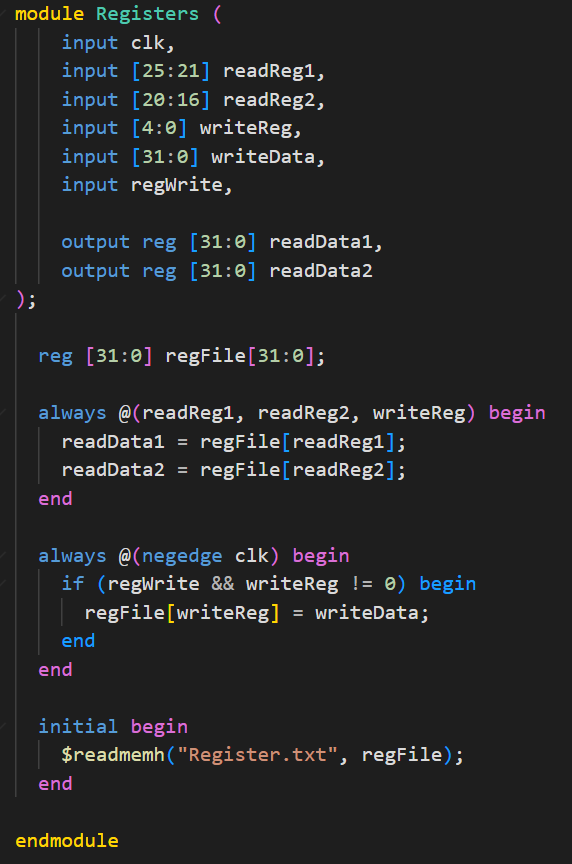


BranchControl模块根据指令的opCode和funct字段判断跳转指令类型，代码如下：



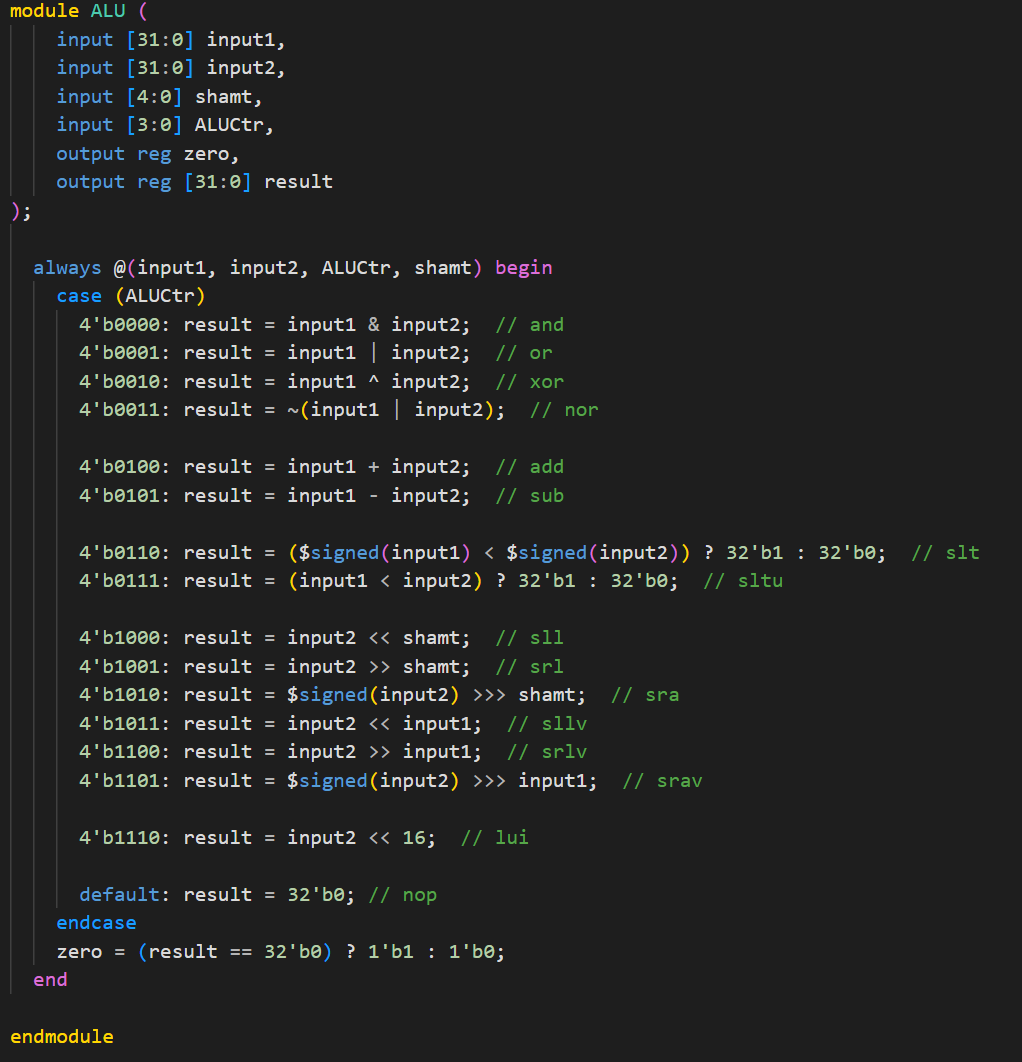
3.5 Registers模块的实现

由于不确定writeReg，writeData，regWrite 信号的先后次序，采用时钟的下降沿作为写操作的同步信号，防止发生错误。因此模块逻辑为：当readReg1，readReg2，writeReg其中一个发生变化时，将寄存器内值读到输出；当遇到clk下降沿时，若regWrite使能，则将输入写入寄存器。代码如下：



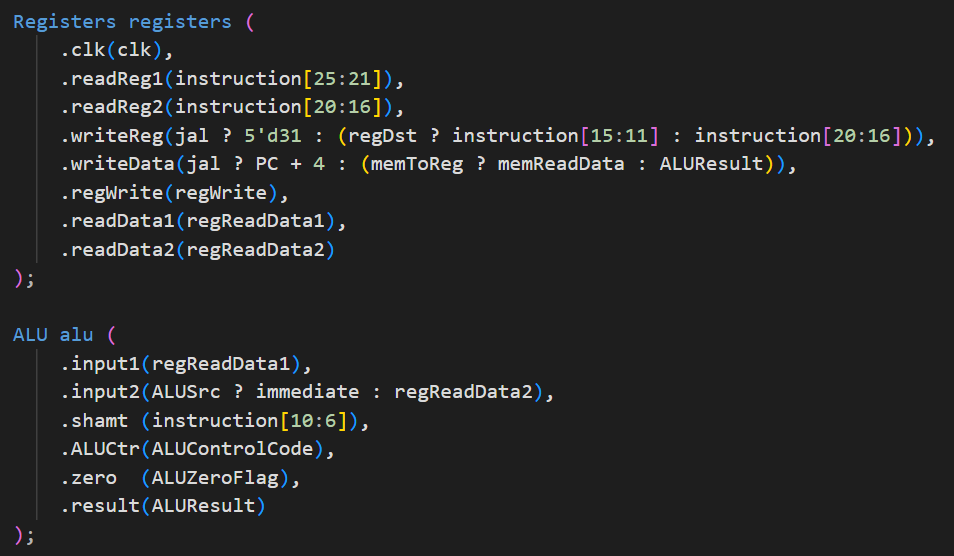
3.6 ALU模块的实现

ALU模块根据ALUCtr的控制信号进行相应运算，ALUCtr信号也可自行约定。需要注意不同指令的有符号数版本与无符号数版本的区别。代码如下：



3.7 Top模块的实现

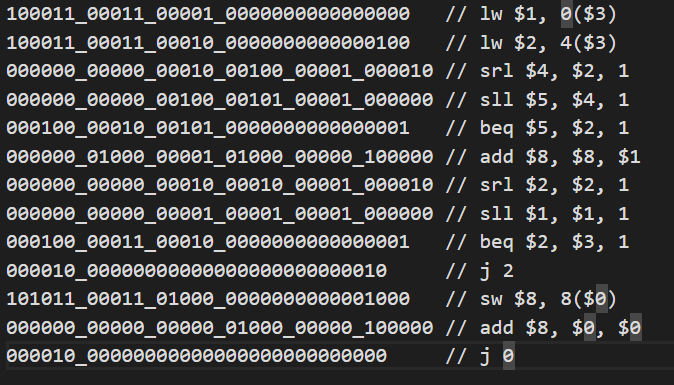
Top模块连接上述模块，根据Control模块的控制信号输出选择相应的输入。部分代码如下：



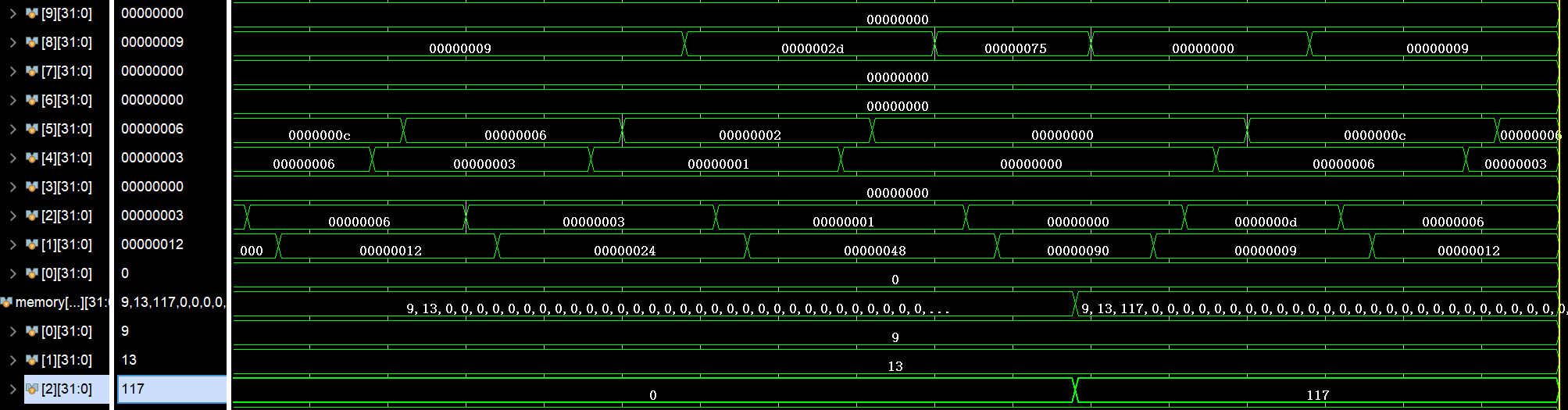
**4.** 结果验证

4.1 乘法代码的测试

编写Instruction.txt文件如下：



测试结果如图所示：

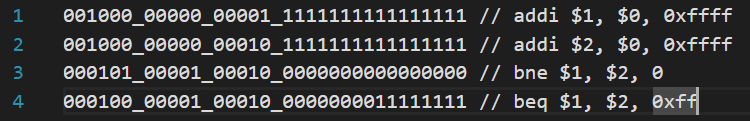


输出和计算过程符合预期。

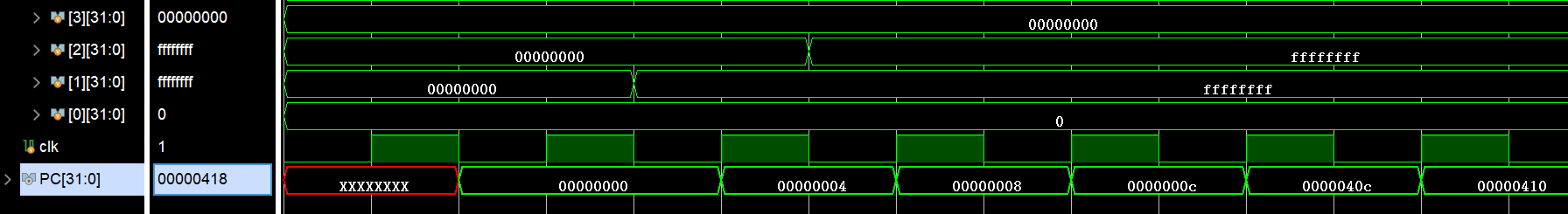
4.2 其他指令的测试

4.2.1 beq，bne指令

编写Instruction.txt文件如下：



测试结果如图所示：



测试代码将1号、2号寄存器设为0xffff。在bne指令中，分支条件不满足，因此PC由0x08增加到0x0c；在beq指令中，分支条件满足，PC = PC + 0x4 + 0xff << 2 = 0x40C。仿真符合预期。

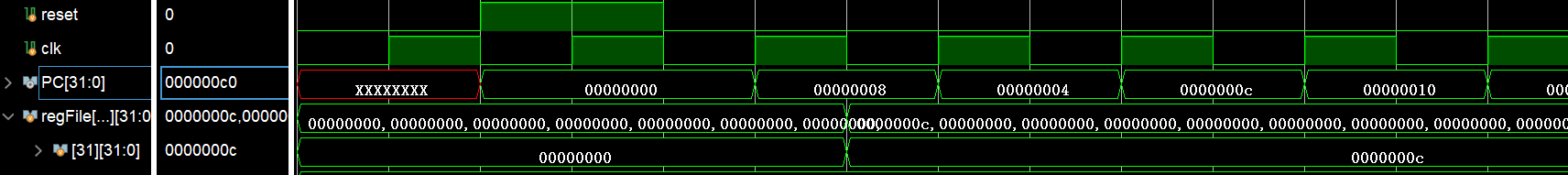
4.2.2 j，jr，jal指令

编写Instruction.txt文件如下：

图形用户界面, 文本

描述已自动生成

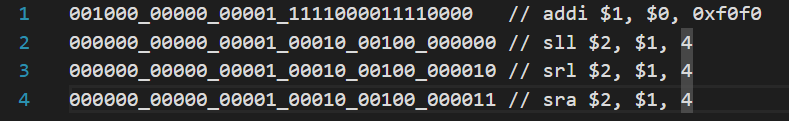
测试结果如图所示：



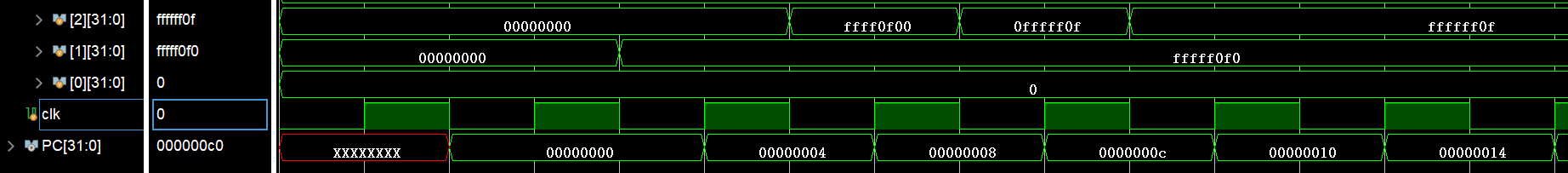
测试代码先使用j 2无条件跳转到PC = 2 << 2 = 8处，此时执行第三条指令jal 1，将PC + 4 = 0xc存入ra寄存器，并跳转到PC = 1 << 2 = 4处。接下来执行第二条指令jr $31将PC设为ra的值0xc。仿真符合预期。

4.2.3 sll，srl，sra指令

编写Instruction.txt文件如下：



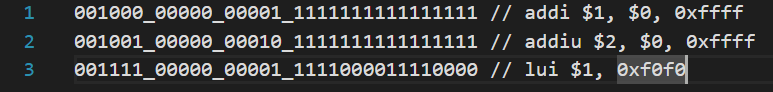
测试结果如图所示：



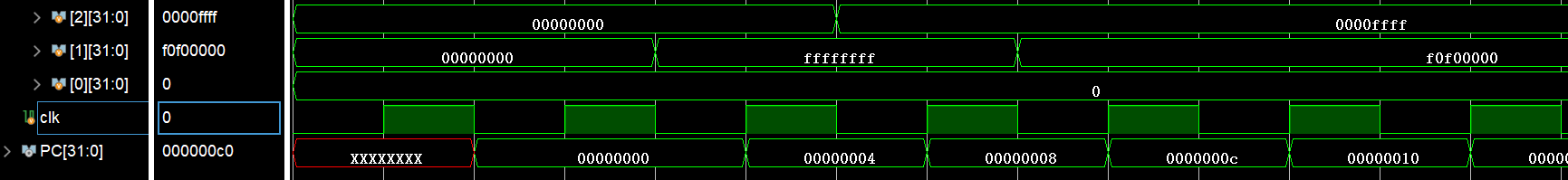
测试代码将1号寄存器设为0xfffff0f0（有符号数扩展）。在sll指令中，1号寄存器左移4位得到0xffff0f00。在srl指令中，1号寄存器右移四位得到0x0fffff0f。在sra指令中，1号寄存器右移4位，并在高位填充符号位1，得到0xffffff0f。仿真符合预期。

4.2.4 addi，addiu，lui指令

编写Instruction.txt文件如下：



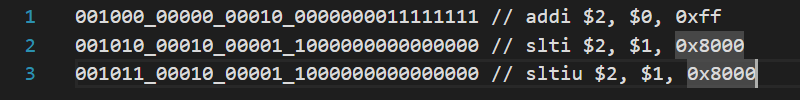
测试结果如图所示：



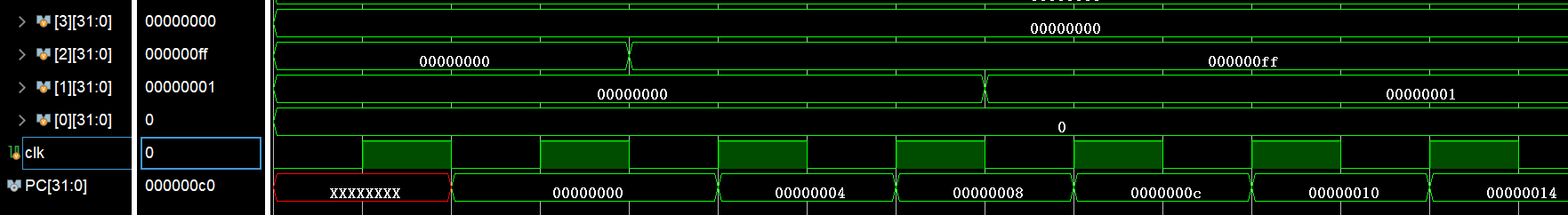
仿真代码中addi指令将0xffff做有符号数扩展得到0xffffffff，addiu指令将0xffff做无符号数扩展得到0x0000ffff。lui指令将立即数0xf0f0填入高16位，并把低16位清零得到0xf0f00000。仿真符合预期。

4.2.5 slt，sltu指令

编写Instruction.txt文件如下：



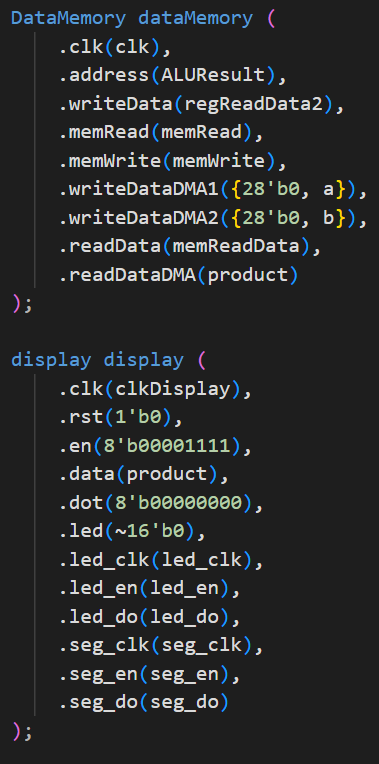
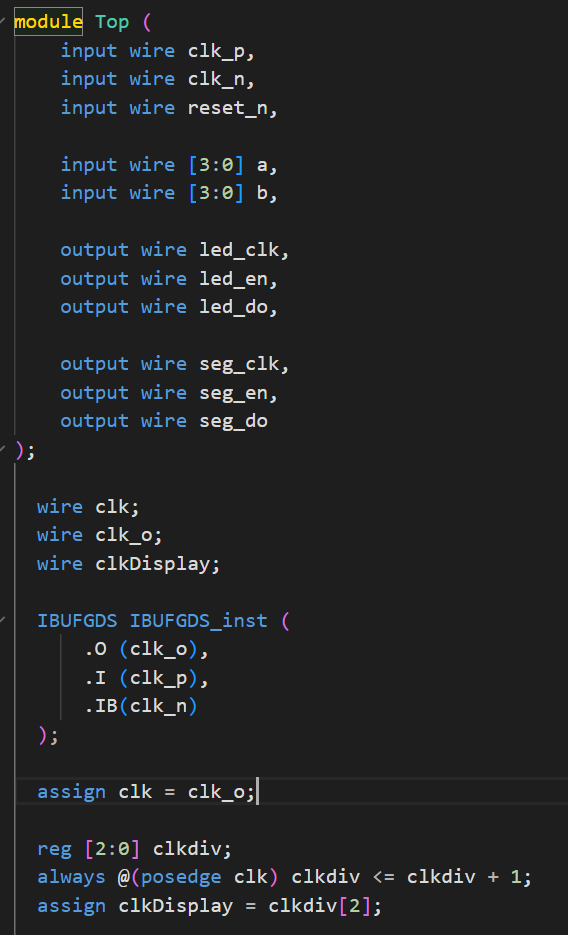
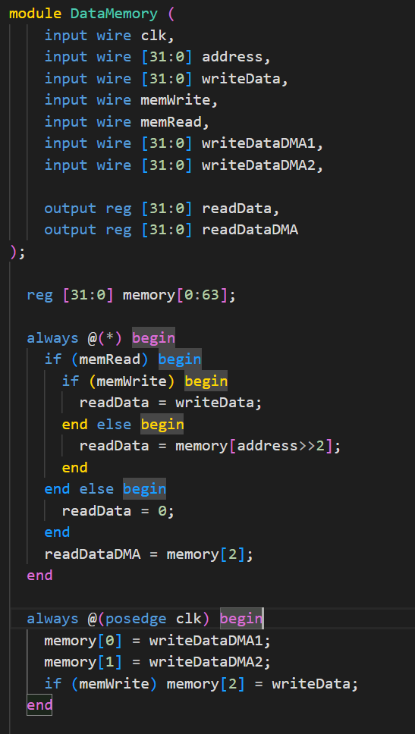
测试结果如图所示：



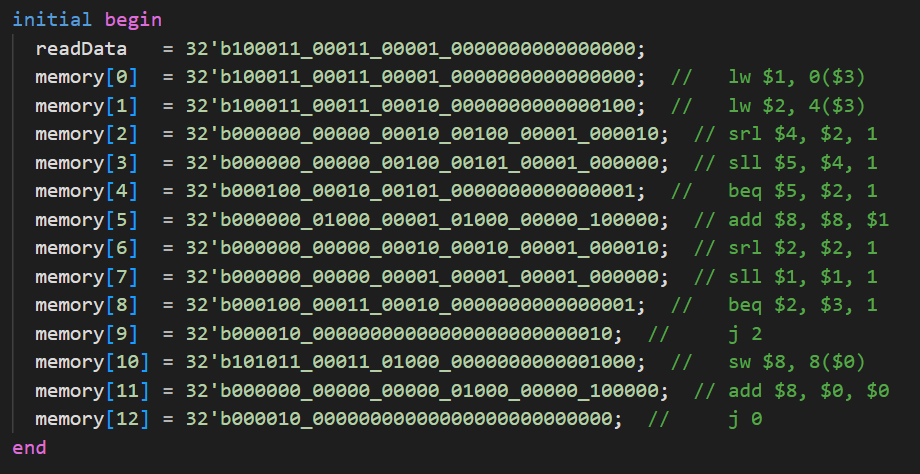
测试代码把2号寄存器设为0x00ff。在slti指令中，0x8000做有符号数扩展，为负数，0x00ff < 0xffff8000 不成立，故1号寄存器不设为1。在sltiu指令中，0x8000做无符号数扩展 0x00ff < 0xffff成立，故1号寄存器设为1。仿真符合预期。

4.3 拨码乘法器的测试

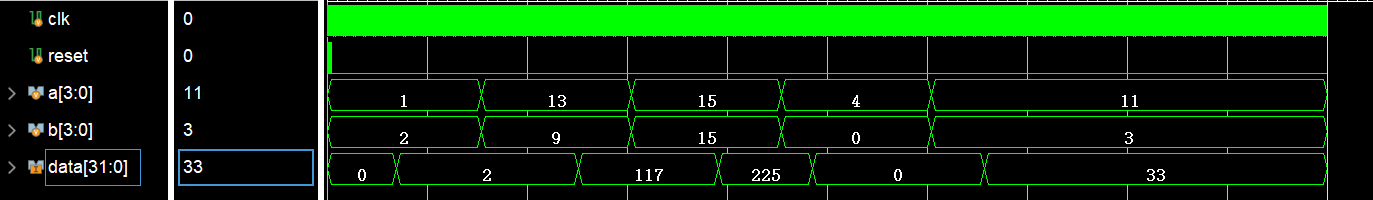
拨码乘法器需要引入display模块IP核并修改相应模块。实验中直接将拨码输入连接到memory[0]与memory[1]，并将memory[2]连接至display数据。修改代码如下：



修改Instruction Memory如下：



仿真结果如下：



上板结果通过了检查。

**5.** 总结与反思

在Lab05中，我深入了解了MIPS单周期处理器的各部件功能和关系，通过实现Program Counter和Branch Control等功能，我熟悉了处理器取指、译码、读写、计算等部分的流程。通过使用Vivado开发环境，我能够更好地编写仿真文件调试Verilog HDL的代码。通过这次实验，我掌握了使用读写文件的方式初始化寄存器的方法，并学习了genvar功能。

我要感谢课程组为我们提供的详细指导书，它为我提供了清晰的实验步骤，使我能够更好地理解和实践所学的知识。通过这次实验，我不仅巩固了Verilog和MIPS处理器的基础知识，还为的学习和设计打下了坚实的基础。