**北京科技大学实验报告**

学院：计算机与通信工程学院 专业：计算机科学与技术 班级：计194

姓名：王承开 学号：41924213 实验日期：2021年1月1日

**实验名称：**

单周期CPU指令扩展与仿真

**实验要求：**

用VerilogHDL或VHDL语言在原处理器基础上扩展两条指令，给出设计思路及扩展后的控制信号表格，仿真波形图，和对仿真波形的具体分析。最后提交该工程文件全部代码。代码应有适当的注释，并在实验报告中体现；报告中需要有指令的分析设计过程（一定包括对数据通路的分析），仿真验证过程需要有仿真波形图及波形分析。

**实验仪器：**

OS：Win10 64位

Software：Vivado2018.3开发工具

**实验原理：**

**实验内容与步骤：**

1. srlv指令的设计思路

srlv为r型指令，其功能当功能码是6’b000110，逻辑右移。

指令用法为：srlv rd，rt，rs。

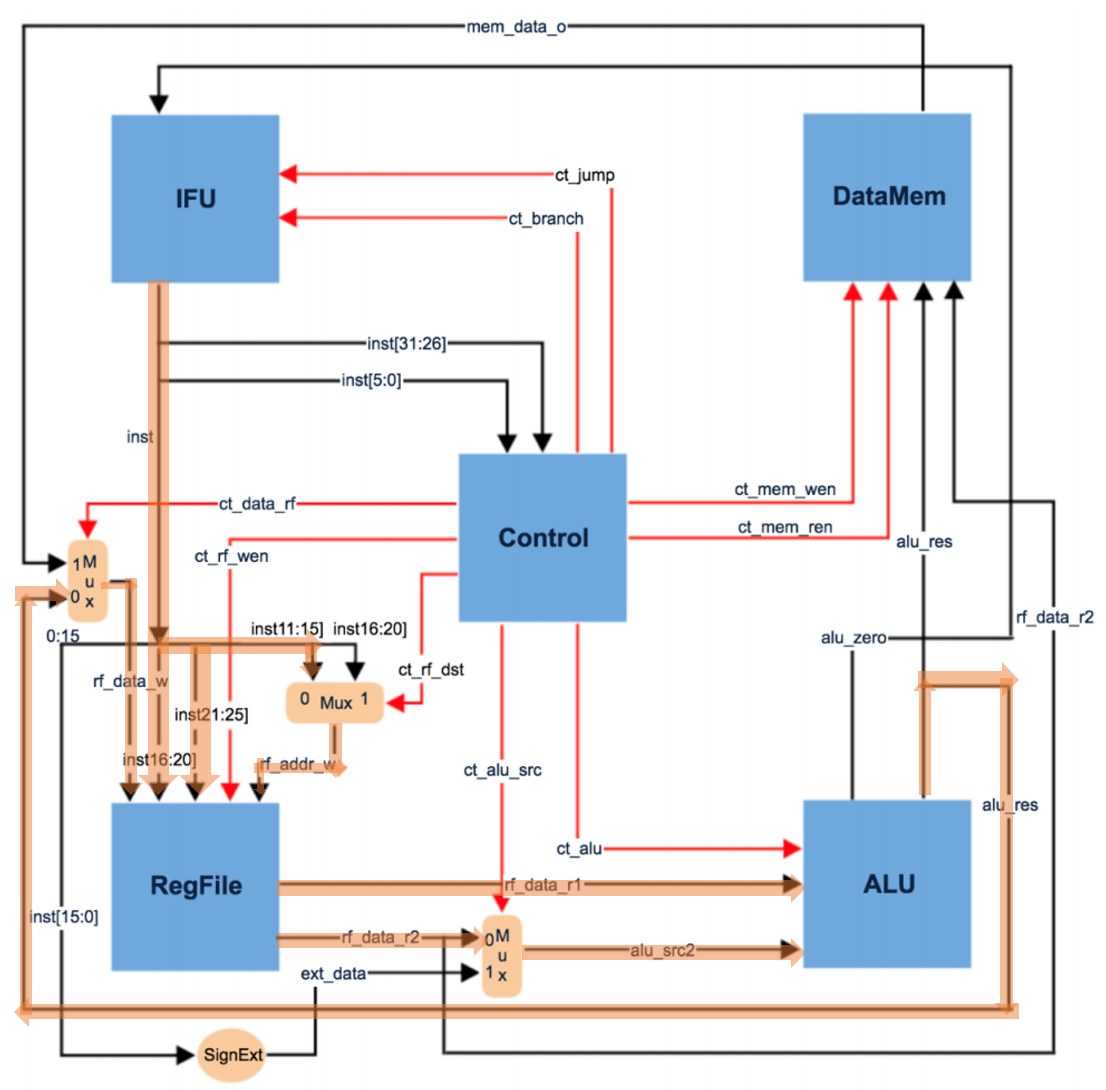
指令作用为：rd ← rt >> rs[4：0]（logic），将地址为rt的通用寄存器的值向右移位，空出来的位置使用0填充，结果保存到地址为rd的通用寄存器中。移位位数由地址为rs的寄存器值的第0-4bit确定。

首先观察srlv的指令结构：

与已实现的R型指令对比发现，其区别仅在于func的值不同，以及两种指令对寄存器堆中的数据的计算方式不一样，于是我们给出以下思路：

1. 通过ifu模块取指
2. 将指令输入到control模块并解析数据，并在aluct模块中解析到alu的操作
3. 把解析好的寄存器地址以及对寄存器的读写操作输入到regfile模块
4. 在regfile模块中读取到数据后送入alu模块进行计算，将计算后的结果返回至regfile模块，即完成指令

根据以上思路设计出该指令的数据通路如下图半透明橙色路径：



在代码实现过程中，需要在aluct中添加对funct的识别关键代码如下：

case(funct)

  //当指令中 funct 段为 100001 时，alu\_ct 出 4’b001（执行加法操作）。

  6'b100001: alu\_ct = 4'b0010;

  //funct 段为000110时，alu\_ct 输出 4'b0011（执行逻辑右移）。

  6'b000110: alu\_ct = 4'b0011;

  default:

begin

alu\_ct = 0;

end

endcase

另在alu模块要添加对数据逻辑右移的操作，关键代码如下：

case(alu\_ct)

    4'b0011://逻辑右移

    begin

        alu\_res = alu\_src2 >> alu\_src1;

    end

为仿真的有效性，对代码实行的顶层仿真，代码如下：

`timescale 1ns / 1ps

module test(

    );

reg clk,rst;

CPU cpu(clk,rst);

initial

begin

    clk = 0;

    rst = 0;

    #100 rst = 1;

end

always #20 clk = ~clk;

endmodule

仿真的指令代码如下：

00000000 // nop

24010002 // addiu $1, $0, 2

24030007 // addiu $3, $0, 7

00232806 //000000 00001 00011 00101 00000 000110 // srlv $5, $3, $1

1. xori指令的设计思路

xori当指令码是6’b001110，表示是xori指令，异或运算。

指令用法为：xori rt，rs，immediate。

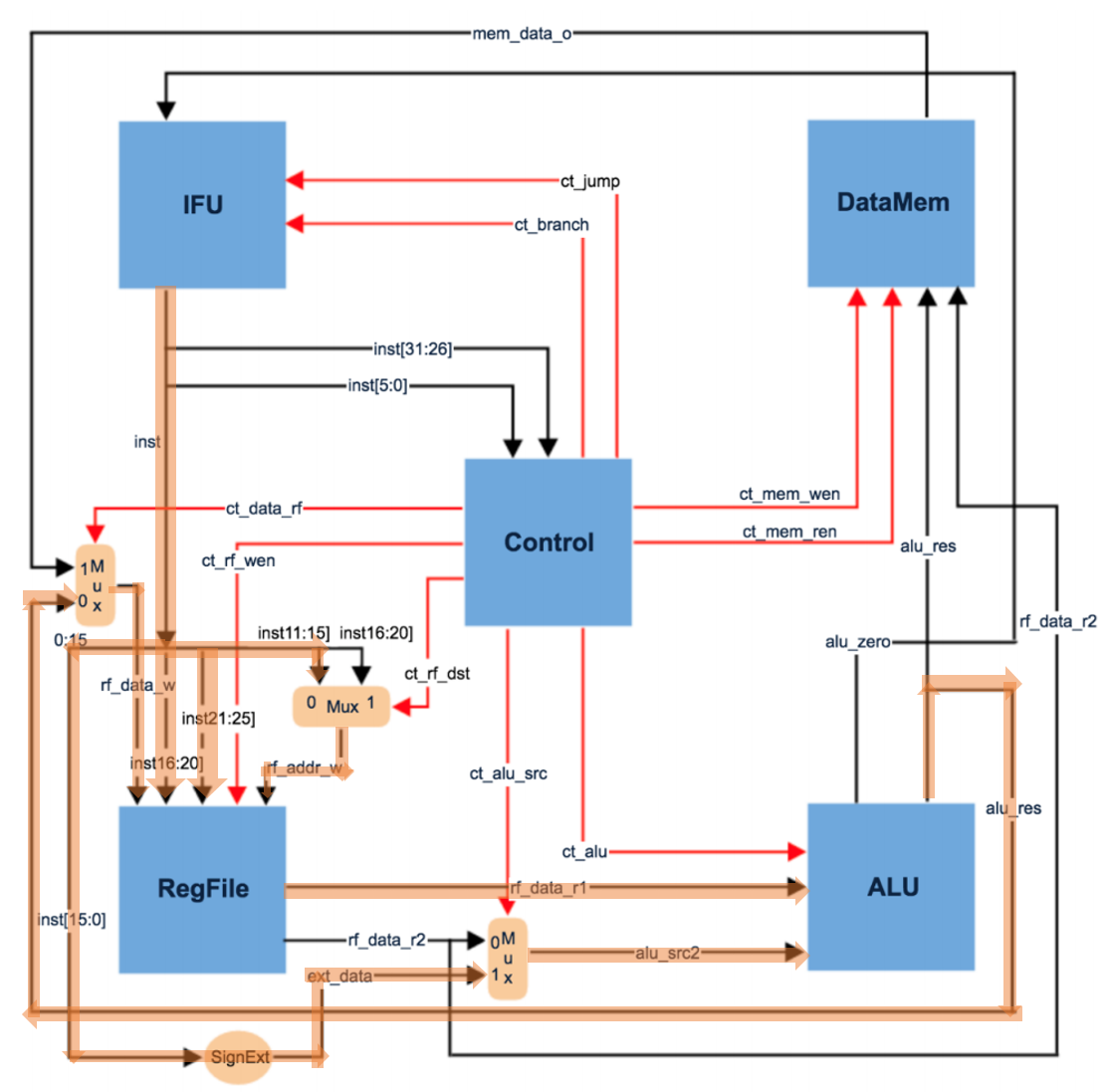
指令作用为：rt ← rs XOR zero\_extended（immediate），将地址为rs的通用寄存器的值与指令中立即数进行零扩展后的值进行逻辑“异或”运算，运算结果保存到地址为rt的通用寄存器中。

首先观察xori的指令结构：

与已实现的I型指令addiu对比发现，其区别在于立即数拓展的方式不同，以及两种指令对寄存器堆中的数据的计算方式不一样，于是我们给出以下思路：

1. 通过ifu模块取指
2. 将指令输入到control模块并解析数据，并在aluct模块中解析到alu的操作
3. 把解析好的寄存器地址以及对寄存器的读写操作输入到regfile模块
4. 在regfile模块中读取到数据后送入alu模块进行计算，将计算后的结果返回至regfile模块，即完成指令

根据以上思路设计出该指令的数据通路如下图半透明橙色路径：



在aluct中，我们发现原先代码的ct\_alu\_op的位数已无法满足需要，于是将ct\_alu\_op拓展成了3位，其赋值修改为：

assign ct\_alu\_op[2:0] = {inst\_r,inst\_beq,inst\_xori};

同时，在aluct模块中修改对alu\_ct\_op的case选择如下：

case(alu\_ct\_op)

3'b000:alu\_ct= 4'b0010;

3'b010:alu\_ct= 4'b0110;

3'b001:alu\_ct= 4'b0111;

3'b100:

begin

case(funct)

//当指令中 funct 段为 100001 时，alu\_ct 输出 4’b0010（执行加法操作）。

6'b100001: alu\_ct = 4'b0010;

//funct 段为000110时，alu\_ct 输出 4'b0011（执行逻辑右移）。

6'b000110: alu\_ct = 4'b0011;

default:

begin

alu\_ct = 0;

end

endcase

end

default:

begin

alu\_ct = 0;

end

endcase

在alu中添加异或的运算以及零拓展的情况：

case(alu\_ct)

begin

zero\_ext = {16'b0,alu\_src2[15:0]};

alu\_res = alu\_src1 ^ zero\_ext;

end

仿真指令如下：

00000000 // nop

24010002 // addiu $1, $0, 2

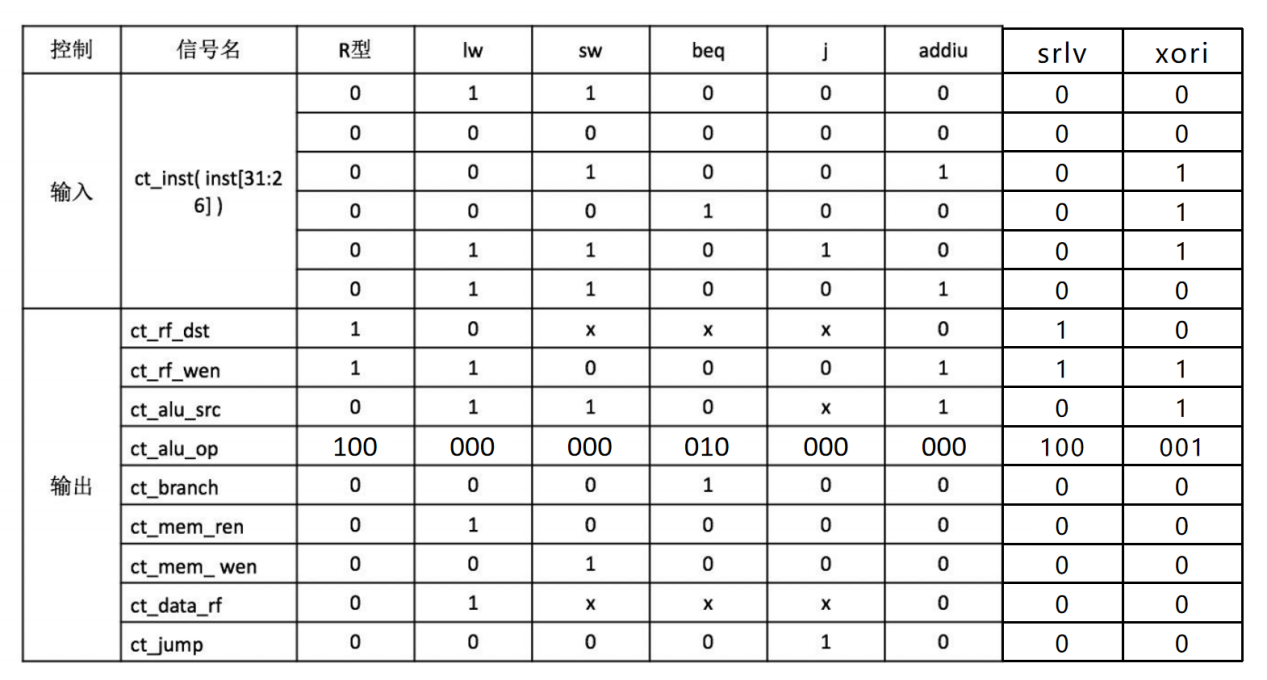
24030007 // addiu $3, $0, 7

00232806 //000000 00001 00011 00101 00000 000110 // srlv $5, $3, $1

3827000F //001110 00001 00111 0000000000001111// xori $7, $1, 15

3827FFFF //001110 00001 00111 1111111111111111// xori $7, $1, ffff

3、拓展后的控制信号表述



**实验数据：**

1. 用于测试的汇编代码，对应的机器码

00000000 // nop

24010002 //001001 00000 00001 00000 00000 000010 addiu $1, $0, 2

24030007 //001001 00000 00011 00000 00000 000111 addiu $3, $0, 7

00232806 //000000 00001 00011 00101 00000 000110 srlv $5, $3, $1

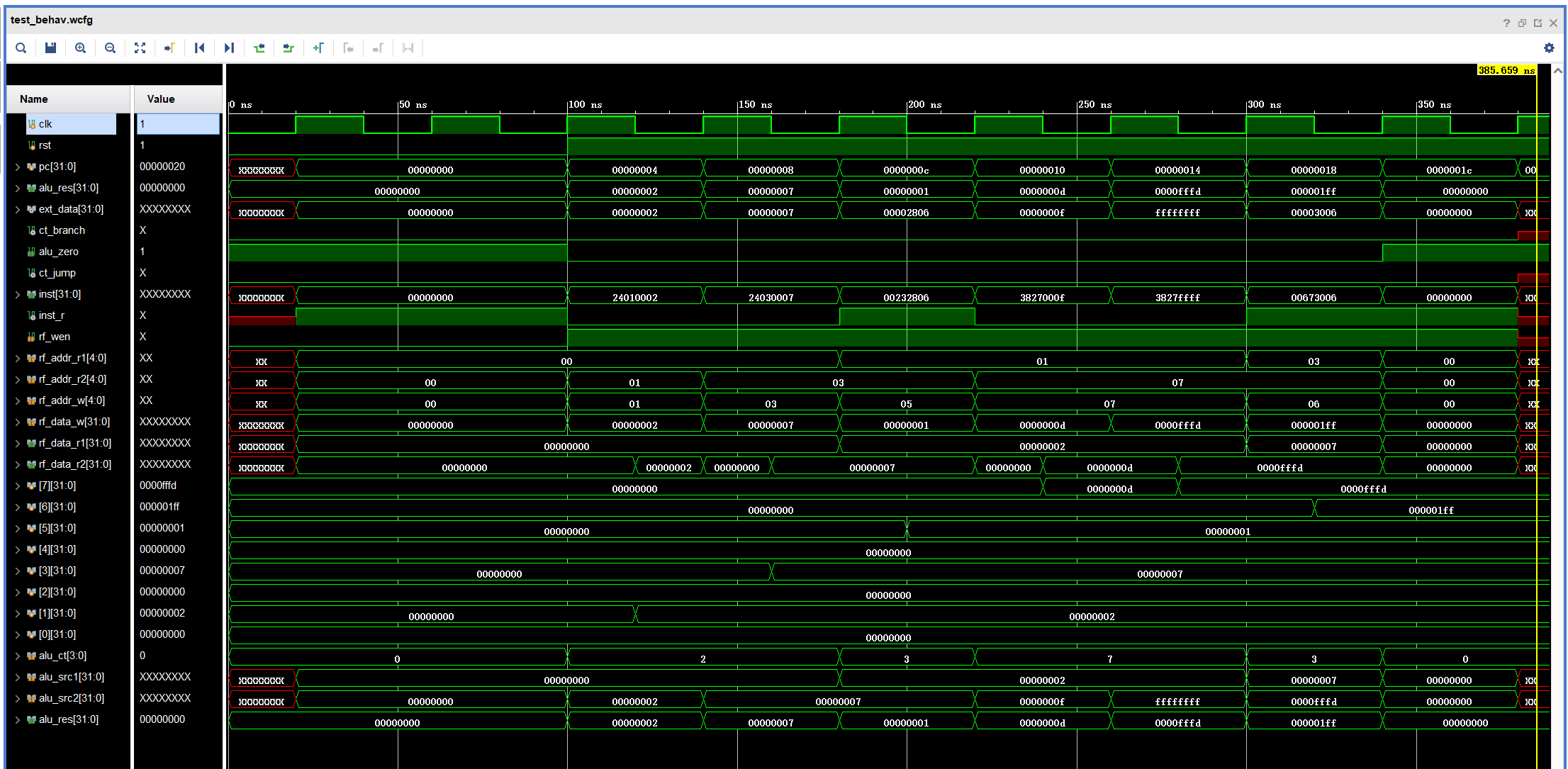
3827000F //001110 00001 00111 0000000000001111 xori $7, $1, 15

3827FFFF //001110 00001 00111 1111111111111111 xori $7, $1, ffff

00673006 //000000 00011 00111 00110 00000 000110 srlv $6, $7, $3

00000000 // nop

1. 仿真波形图。



**实验结果与分析：**

1. 在初始化后寄存器的值均为0，这是符合预期的
2. 观察pc取值和inst的值，可以发现正常读入机器码
3. 在2401002，addiu $1, $0, 2指令执行完后寄存器1的值变成了2，这是符合预期的
4. 在2403007，addiu $3, $0, 7指令执行完后寄存器3的值变成了7，这是符合预期的
5. 接下来时拓展语句的实现
6. 00232806，srlv $5, $3, $1指令执行完后寄存器5的值为1，是111>>2的结果，是符合预期的
7. 3827000F，xori $7, $1, 15指令执行完后寄存器7的值为d，是2^15的结果，是符合预期的
8. 3827FFFF，xori $7, $1, ffff指令执行完后寄存器7的值为fffd，零拓展有效，且结果是符合预期的
9. 00673006，srlv $6, $7, $3指令执行完后寄存器6的值为1ff，是fffd>>7的结果，符合预期。