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：

srlv	000000	rs	rt	rd	00000	000110	srlv $1, $2, $3	$1=$2>>$3	(rd)<-(rt)>>(rs),rs=$3,rt=$2,rd=$1（逻辑右移）

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：

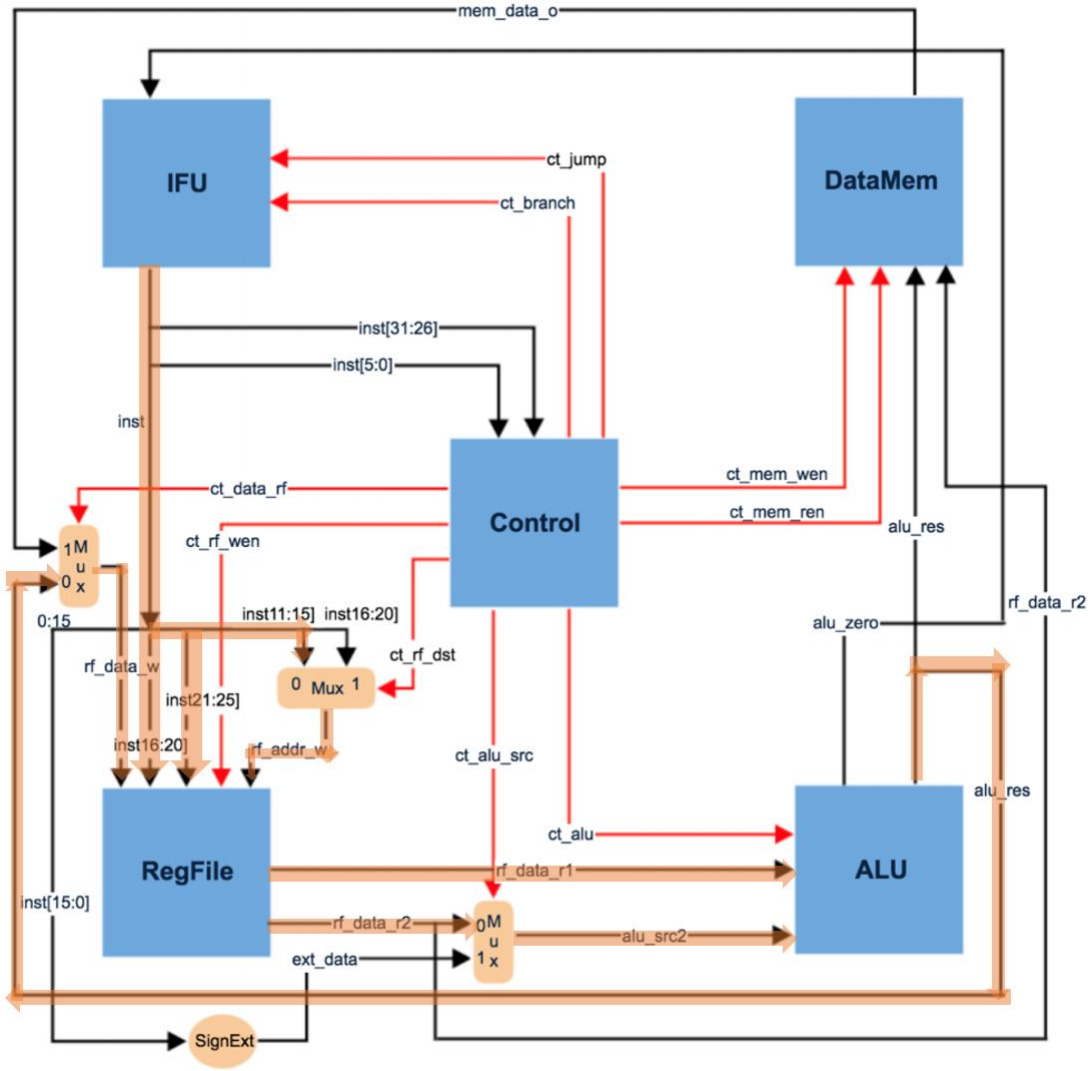
（1）通过 ifu 模块取指

（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

2、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：

xori	001110	rs	rt	immediate	xori $1, $2, 10	$1 = $2 ^ 10	(rt)<-(rs)^(zero-extend)immediate,rt=$1,rs=$2

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：

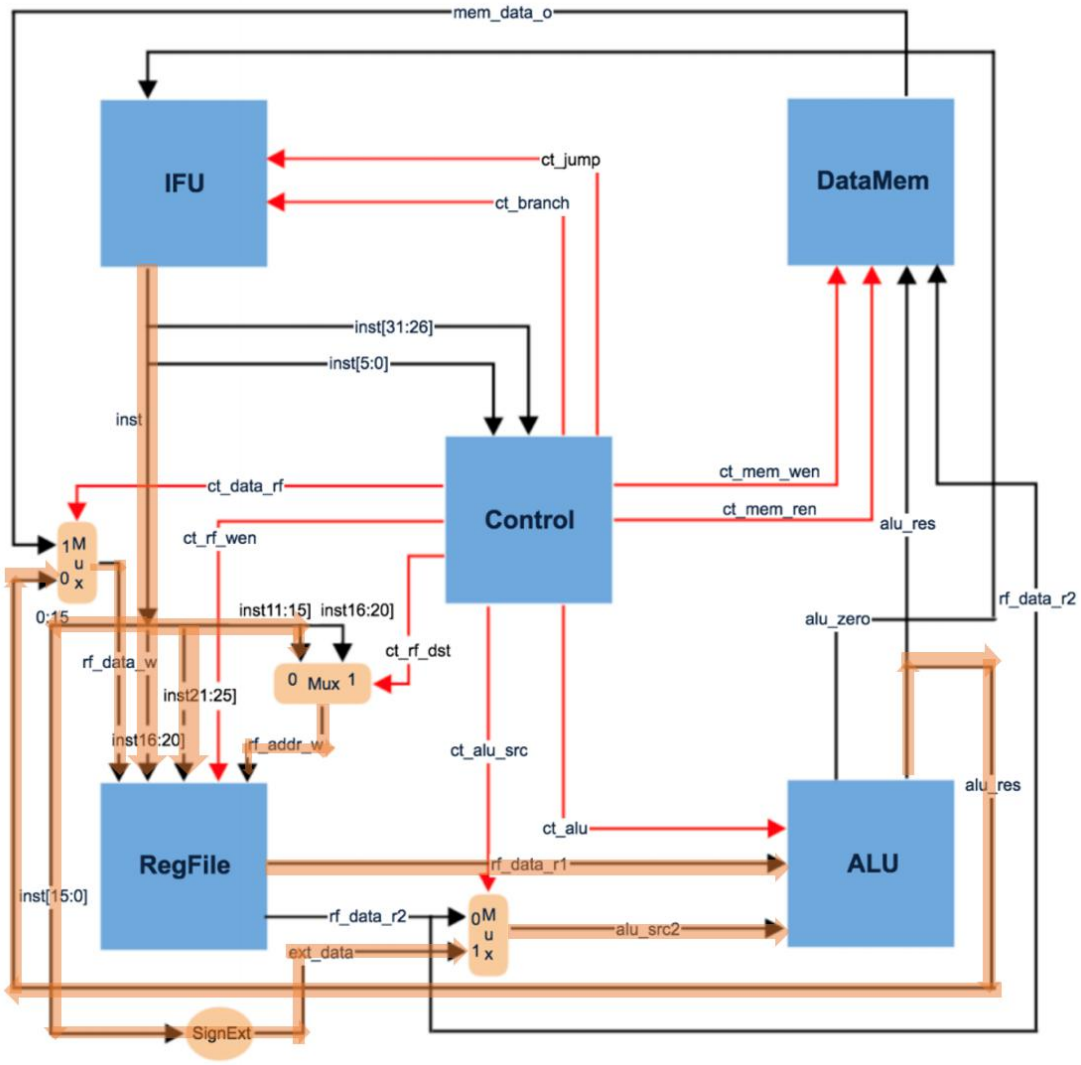
（1）通过 ifu 模块取指

（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

控制	信号名	R型	lw	sw	beq	j	addiu	srlv	xori
输入	ct_inst(inst[31:26])	0	1	1	0	0	0	0	0
		0	0	0	0	0	0	0	0
		0	0	1	0	0	1	0	1
		0	0	0	1	0	0	0	1
		0	1	1	0	1	0	0	1
		0	1	1	0	0	1	0	0
输出	ct_rf_dst	1	0	x	x	x	0	1	0
	ct_rf_wen	1	1	0	0	0	1	1	1
	ct_alu_src	0	1	1	0	x	1	0	1
	ct_alu_op	100	000	000	010	000	000	100	001
	ct_branch	0	0	0	1	0	0	0	0
	ct_mem_ren	0	1	0	0	0	0	0	0
	ct_mem_ wen	0	0	1	0	0	0	0	0
	ct_data_rf	0	1	x	x	x	0	0	0
	ct_jump	0	0	0	0	1	0	0	0

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

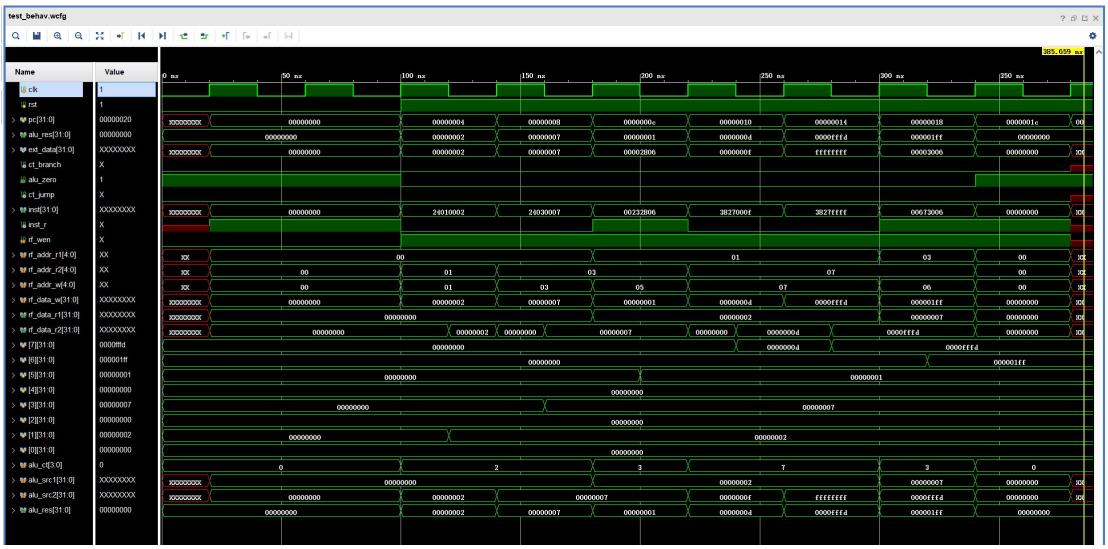
3827000F //001110 00001 00111 0000000000001111　xori $7, $1, 15

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

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

00000000 // nop

2、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。