**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：Win7 64位

Software：Vivado2018.1开发工具

**实验原理：**

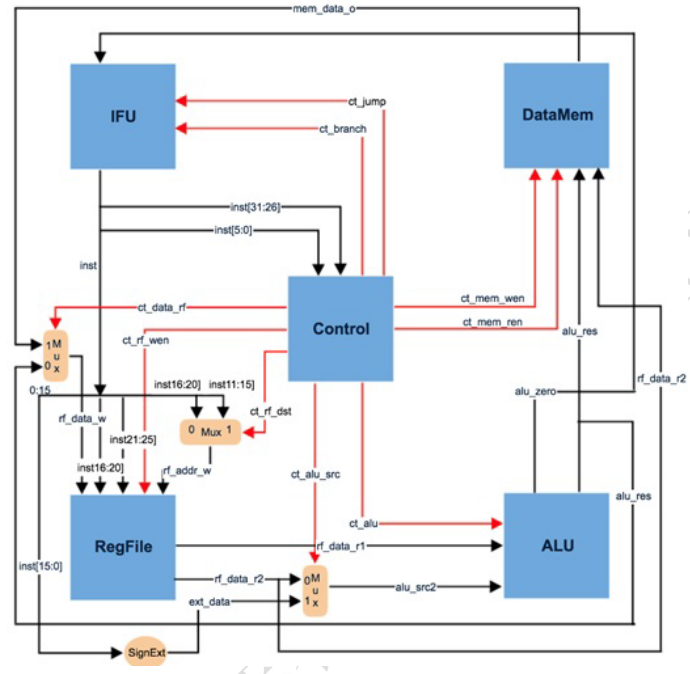
R型指令的实现思路：首先将数据传入寄存器7和寄存器8中，再由算术逻辑单元进行运算，得到结果，再将结果传入寄存器9中。每次在时钟信号的下降沿进行触发写入操作，rst为高电平复位。得到指令之后，需要ALU运算器来完成操作。

Subu指令：执行的操作是 $rd=$rs - $rt；

And指令：执行的操作是 $rd=$rs & $rt；

Or指令：执行的操作是 $rd=$rs | $rt；

Xor指令：执行的操作是 $rd=$rs ^ $rt；



1.IFU 模块完成指令取值，将功能码部分传入控制模块中；

2.将 ct\_alu 设置为Subu/And/Or/Xor操作的操作码（ct\_alu\_src 默认为 0，即第二个操作数来自寄存器堆），使得 ALU 进行指定的运算；

3.将 ct\_rf\_wen 置为 1，使得寄存器堆可写（ct\_data\_rf 默认为 0，即写入内容来源自 ALU 运算结果）

4.将 ct\_rf\_dst 置为 1，指定存储位置为 rd，将运算结果写入相应位置的寄存器中。

**实验内容与步骤：**

扩展后的控制信号表：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 扩展指令 | op | rs | rt | rd | shamt | func | 控制信号 |
| Subu | 00000000 | 00111 | 01000 | 01001 | 00000 | 100011 | 00E84823 |
| And | 00000000 | 00111 | 01000 | 01001 | 00000 | 100100 | 00E84824 |
| Or | 00000000 | 00111 | 01000 | 01001 | 00000 | 100101 | 00E84825 |
| Xor | 00000000 | 00111 | 01000 | 01001 | 00000 | 100110 | 00E84826 |

关键代码：

ALU模块：

`timescale 1ns / 1ps

module ALU(

input rst,

input[3:0] alu\_ct,

input[31:0] alu\_src1,alu\_src2,

output alu\_zero,

output reg [31:0] alu\_res

);

assign alu\_zero=(alu\_res==0)?1:0;

always@(\*)

if(!rst)begin

alu\_res=32'b0;

end

else begin

case(alu\_ct)

4'b0010:

alu\_res =(alu\_src1+alu\_src2);//执行加法操作

4'b0110:

alu\_res =(alu\_src1-alu\_src2);//执行减法操作

4'b0001:

alu\_res = alu\_src1 & alu\_src2;//执行与操作

4'b0011:

alu\_res = alu\_src1 | alu\_src2;//执行或操作

4'b0100:

alu\_res = alu\_src1 ^ alu\_src2;//执行异或操作

default: alu\_res=32'b0;

endcase

end

endmodule

ALUCt模块：

`timescale 1ns / 1ps

module ALUCt(

input rst,

input[5:0] funct,

input[1:0] alu\_ct\_op,

output reg[3:0] alu\_ct

);

always@(\*)

if(!rst)

alu\_ct=0;

else

case(alu\_ct\_op)

2'b00:alu\_ct=4'b0010;

2'b01:alu\_ct=4'b0110;

2'b10:begin case(funct)//funct为功能码，由ALU模块传入得到

6'b100001:alu\_ct=4'b0010;//功能码为100001，应执行加法操作，将alu\_ct设置成0010，传入ALU模块中。

6'b100011:alu\_ct=4'b0110; //功能码为100011，应执行剑法操作，将alu\_ct设置成0110，传入ALU模块中。

6'b100100:alu\_ct=4'b0001;//功能码为100100，应执行与操作，将alu\_ct设置成0001，传入ALU模块中。

6'b100101:alu\_ct=4'b0011; //功能码为100101，应执行或操作，将alu\_ct设置成0011，传入ALU模块中。

6'b100110:alu\_ct=4'b0100; //功能码为100110，应执行异或操作，将alu\_ct设置成0100，传入ALU模块中。

default: alu\_ct=0;

endcase end

default: alu\_ct=0;

endcase

endmodule

**仿真代码：**

`timescale 1ns / 1ps

module cpu\_tb();

reg clk=0;

reg rst=0;

initial begin

forever #10 clk=~clk;

end

initial begin

rst = 0;

#25 rst=1;

end

CPU cpu(

.clk(clk),

.rst(rst)

);

endmodule

**实验数据：**

**测试用汇编代码：**

nop

Addiu $7, $0, 7 //x0=7

Addiu $8,$0, 4 //x1=4

Addu $9, $7, $8 //x2=x0+x1=11=b

Subu $9, $7, $8 //x2=x0-x1=3

And $9, $7, $8 //x2=x0&x1

Or $9, $7, $8 //x2=x0|x1

Xor $9, $7,$8 //x2=x0^x1

**对应机器码：**

0000 0000 0000 0000 0000 0000 0000 0000 //nop

0010 0101 0010 0111 0000 0000 0000 0111 // Addiu $7, $0, 7

0010 0101 0010 1000 0000 0000 0000 0100 // Addiu $8,$0, 4

0000 0000 1110 1000 0100 1000 0010 0001 // Addu $9, $7, $8

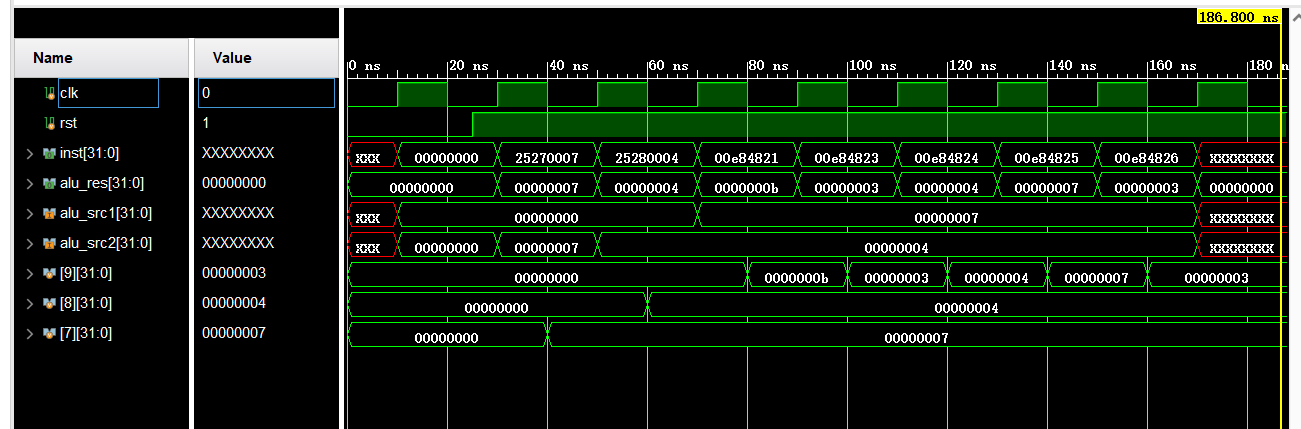
0000 0000 1110 1000 0100 1000 0010 0011 // Subu $9, $7, $8

0000 0000 1110 1000 0100 1000 0010 0100 // And $9, $7, $8

0000 0000 1110 1000 0100 1000 0010 0101 // Or $9, $7, $8

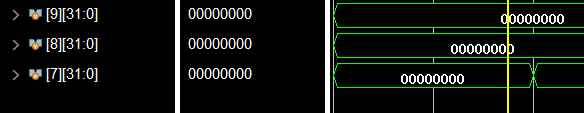
0000 0000 1110 1000 0100 1000 0010 0110 / Xor $9, $7,$8

**仿真波形图：**

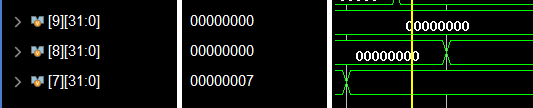


**实验结果与分析：**

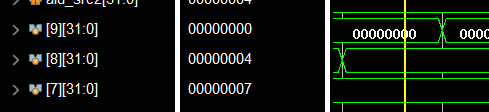
最开始$7，$8，$9都为0



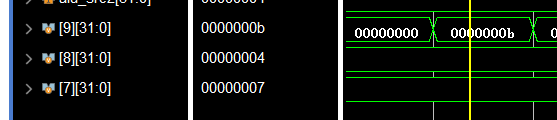
第一个指令将$7传入$7中



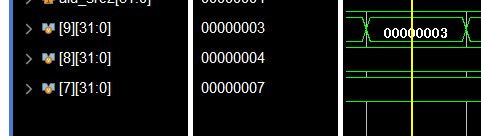
第二个指令将$4传入$8中



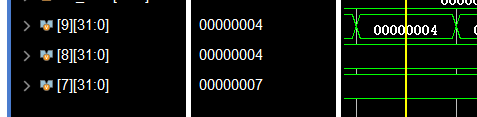
第三个指令将$7和$8进行Addu相加，并传入$9中



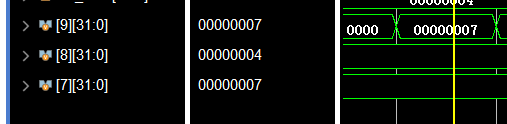
第四个指令将$7和$8进行Subu相减，并传入$9中



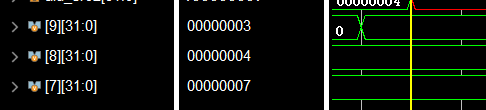
第五个指令将$7和$8进行And，并传入$9中



第六个指令将$7和$8进行Or，并传入$9中



第七个指令将$7和$8进行Xor，并传入$9中



本次实验拓展了加法、减法、与、或、异或指令，经过仿真图形验证，可验证该指令实现。