**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172班

姓名：刘梓瑄 学号：41724039 实验日期：2019 年 5 月 15 日

**实验名称：**

一．取指单元模块（IFU）的设计

**实验要求：**

用VerilogHDL或VHDL语言补充完整取指单元模块，要求：

支持指令的连续读取；

支持任意给定目标地址指令的读取；

支持满足分支指令条件下指令的读取。

写出仿真代码，根据仿真波形分析模块的功能，验证其正确性。代码应有适当的注释，并在实验报告中体现；报告中需要有设计分析过程，仿真实现的设计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：**

IFU实现的是取指模块的功能，其输入为来自Control的控制信号ct\_branch,ct\_jump,alu\_zero以及时钟控制信号clk和复位控制信号rst。输出即为32位的指令。通过各控制信号确定程序计数器pc的跳转情况。其中特别注意beq分支指令和跳转j指令执行前后pc的变化和指令读取情况的变化：如果只是每个时钟取出一条指令，因为此单周期CPU按字节编址，PC执行加4操作，beq指令和j指令则需要修改PC的值。

该模块的输入端口为：

1. 时钟脉冲信号clk。
2. 复位信号rst。
3. 控制信号：alu\_zero,ct\_branch,ct\_jump;

输出端口为：

1. 指令本身：inst[31:0]。

**实验内容与步骤：**

模块补充代码：

if(ct\_jump)

pc<={pc[31:28],address,2'b00};

else if(ct\_branch && alu\_zero)

pc<=pc+(ext\_data[15:0]<<2);

else

pc<=pc+4;

补充代码的编写思路：

jump指令是I型指令，需要执行的操作是：PC={PC[31:28],address,2’b00};.

beq指令是I型指令，需要执行的操作是：PC=PC+立即数<<2;

非上述两种情况时执行PC=PC+4;

依照上述信息完成对相关代码的补充。

仿真代码：testbench.v文件如下

module ifu\_tb();

reg clk, rst;

reg alu\_zero,ct\_branch,ct\_jump;

wire[31:0] inst;

IFU ifu0(clk,rst,alu\_zero,ct\_branch,ct\_jump,inst);

//初始化

initial begin

clk = 0;

rst = 1;

alu\_zero=0;

ct\_branch=0;

ct\_jump=0;

end

always begin

#5 clk = ~clk;//时钟周期

end

initial begin

#1 rst = 0;

#20 rst = 1;//初始化

#50 alu\_zero=1;

ct\_branch=1;

#40 alu\_zero=0;

ct\_branch=0;

#60 ct\_jump=1;

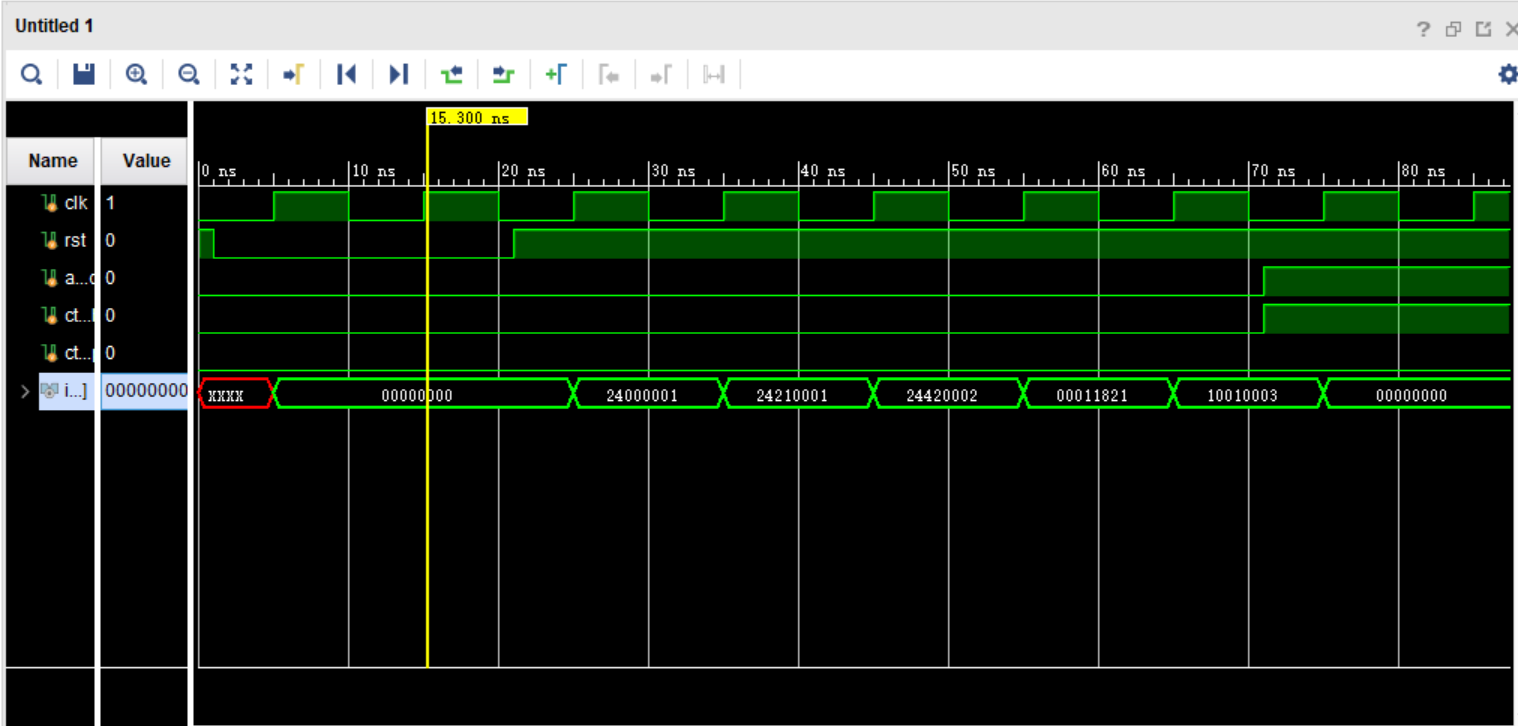
#3000 $stop;//运行时间上限

end

endmodule

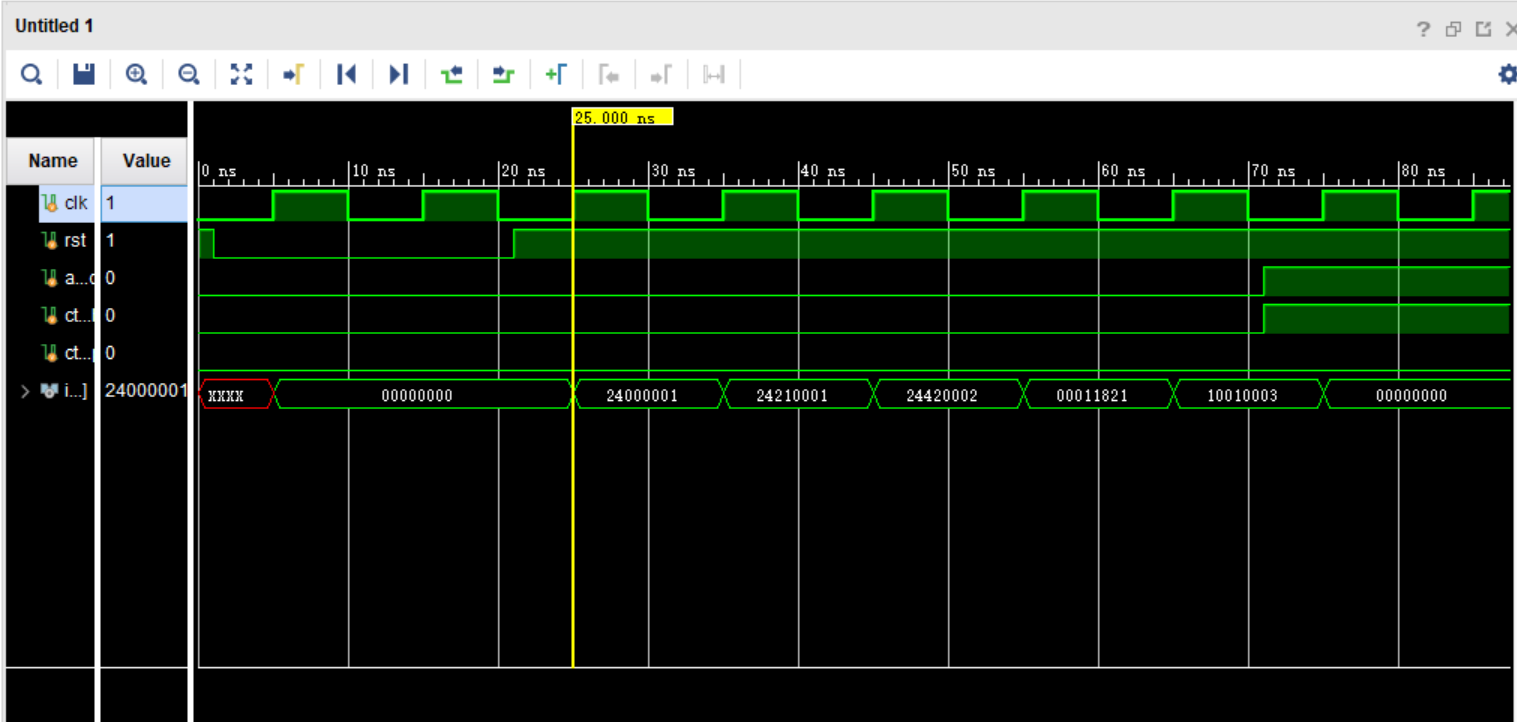
**实验数据：**

波形仿真图1：



如图所示，此时rst信号为无效状态0，过此时指令在时钟脉冲的上升沿只完成了初始化（即初始值赋值为零），而不发生指令的跳转。

波形仿真图2：



此时rst信号为有效状态1，在第一个时钟脉冲的上升沿开始指令的读取，且此时ct\_jump,ct\_branch,ct\_zero均为无效状态，故接下来在时钟脉冲的上升沿不断顺序读取指令，指令文件inst.data如下：

00000000

//第0条指令

24000001

//addiu $0 $0 1

24210001

//addiu $1 $1 1

24420002

//addiu $2 $2 $2

00011821

//addiu $3 $0 $1

10010003

//beq $0 $1 8

08000002

//跳转到第2条指令

00000000

00000000

24820001

24820003

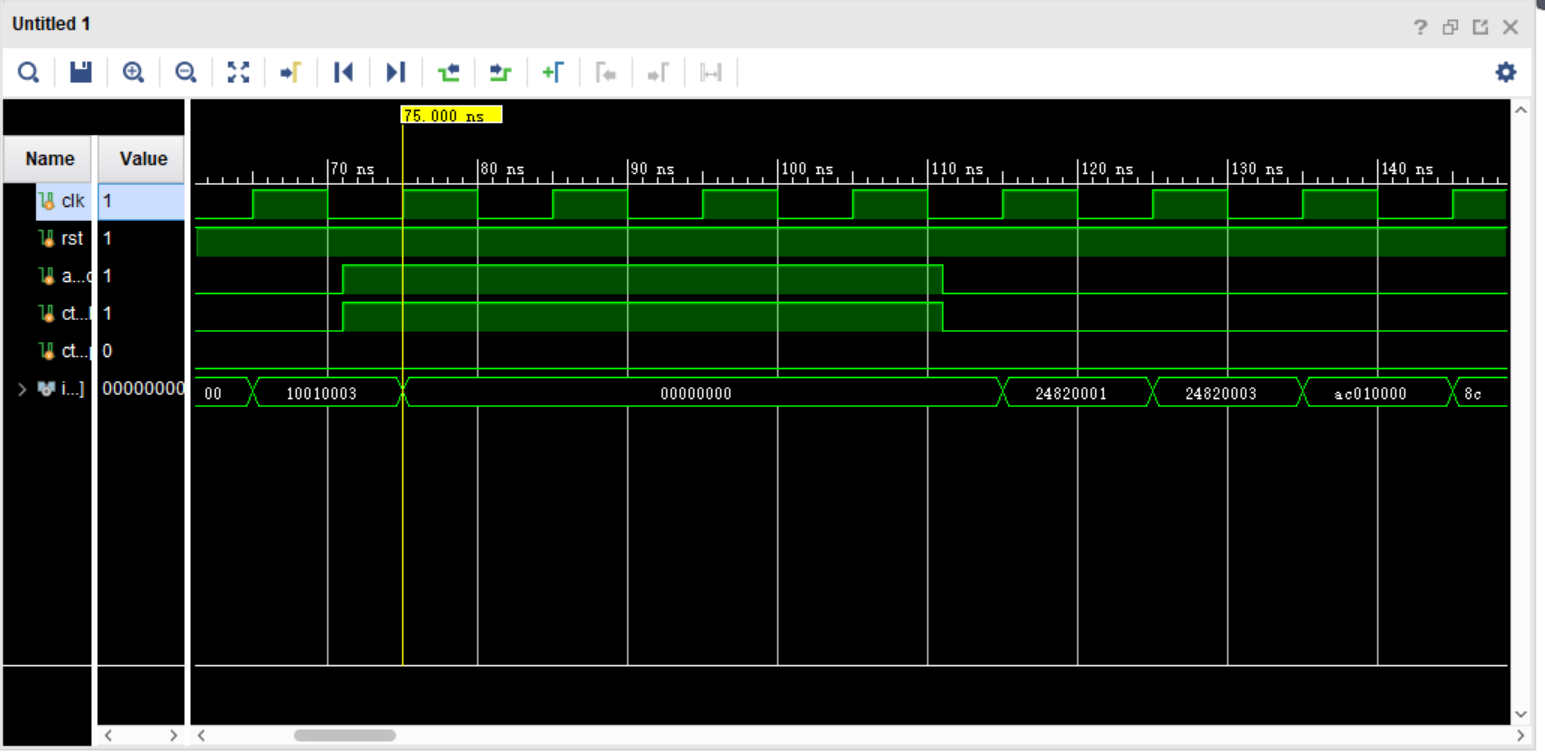
ac010000

8c030000

ac020001

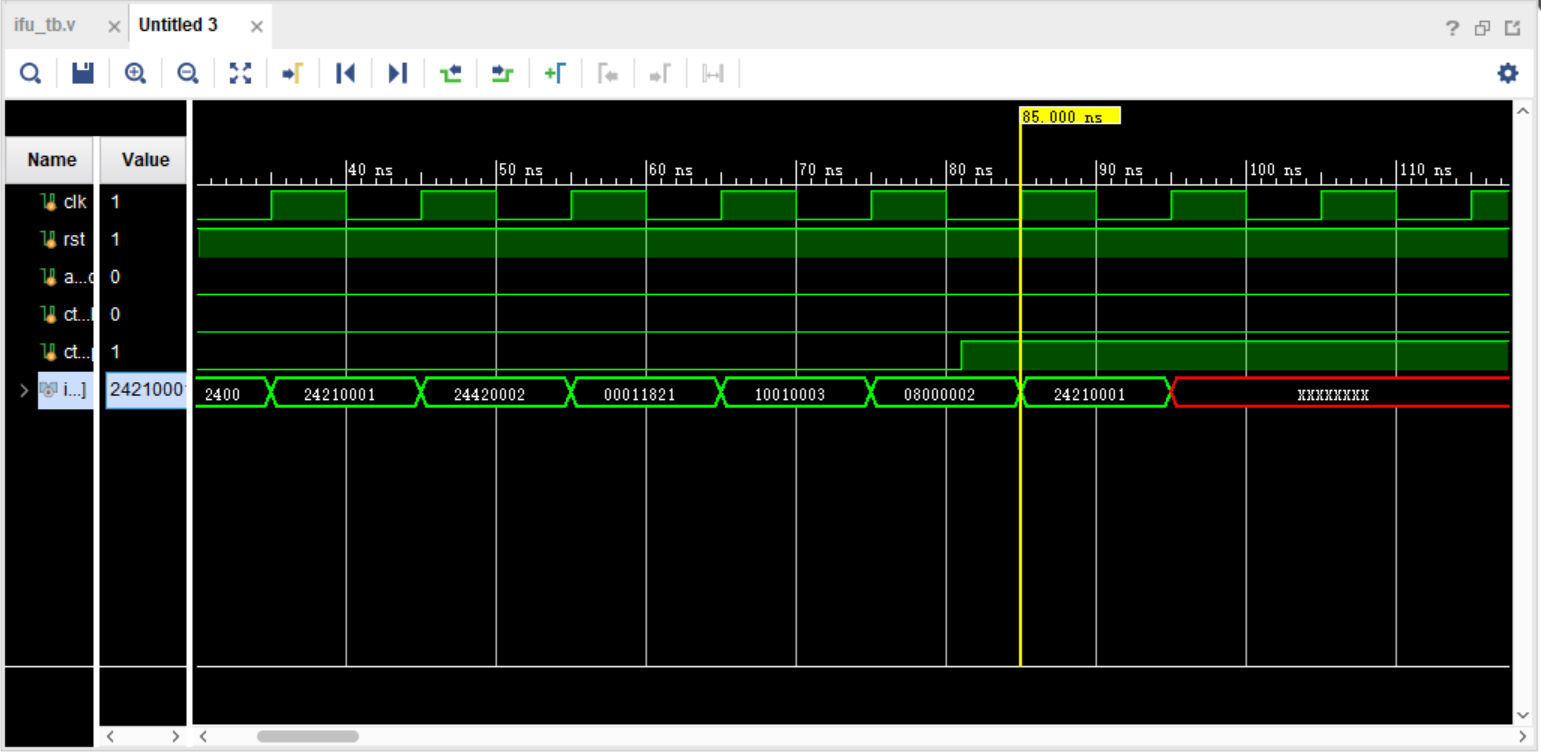
8c030001

波形仿真图3：



此时ct\_zero和ct\_branch指令均为有效信号1，在rst为有效信号1时，在时钟脉冲的上升沿执行分支指令，此时指令的执行顺序不再为顺序执行，而是直接开始执行分支指令，从波形图中可以看出，在执行完指令10010003后分支指令信号有效，接下来开始执行00000000,00000000,24820001…符合设计要求。

波形仿真图4：



从图中分析可知：在指令顺序执行至08000002指令时，ct\_jump变成有效控制信号1，此时执行beq指令，指令的内容是跳转至第二条指令开始执行，而从波形图中可以看出，在08000002指令执行完毕后，开始执行指令24210001，确为inst.data文件中的第二条指令，符合设计要求。值得做出解释的是，在24210001指令执行完后inst端口输出的高阻状态是因为ct\_jump控制信号此时还处于有效状态，故指令未能按顺序执行（即pc没有执行加4操作），符合设计要求。

**实验结果与分析：**

IFU模块成功实现了取指令的功能，输出inst即为相应的指令信号，inst长度为32位，该MIPS指令集采用32-bit编码，本模块主要实现了：

1. 加载指令文件到存储器，通过$readmeemh函数实现。
2. 根据指令类型，实现了三种寻址方式；
3. 寄存器直接寻址，源操作数或目的操作数直接指明相应的寄存器编号，直接读取操作数（如：addiu指令）。
4. 基址加偏移量寻址：通过寄存器与偏移量直接相加得到地址。
5. PC相对寻址（如分支指令beq，跳转指令j），即通过PC与地址便宜相加得到地址值。

实验结果分析：从上述仿真波形的分析中可知，来自Control模块的控制信号支持IFU模块执行顺序读取，分支指令，跳转指令，指令成功读取，符合IFU模块的设计要求。

**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172班

姓名：刘梓瑄 学号：41724039 实验日期：2019年5月15日

**实验名称：**

二．寄存器堆模块（Regfile）的设计

**实验要求：**

用VerilogHDL或VHDL语言补充完整寄存器堆模块，要求：

支持写入存储器32位数据；

支持从存储器读出32位数据；

写出仿真代码，根据仿真波形分析模块的功能，验证其正确性。代码应有适当的注释，并在实验报告中体现；报告中需要有设计分析过程，仿真实现的设计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：**

RegFile模块完成的是寄存器堆及其读写操作，需要根据地址读出或写入数据，读操作一次读出两个寄存器的值，它们的地址分别为指令的rs,rt字段，执行写操作时，地址不同对应的指令不同，通过二选一数据选择器执行选择操作。

该模块的输入端口为：

1. 时钟控制信号clk
2. 写入的控制信号：rf\_wen
3. 寄存器的地址字段rf\_addr\_r1,rf\_addr\_r2
4. 写入内容的地址字段和数据值rf\_addr\_w和rf\_data\_w

输出端口为：

* 1. 寄存器内存放的数据rf\_data\_r1和rf\_data\_r2;

**实验内容与步骤：**

模块补充代码：

if(rf\_wen)

begin

file[rf\_addr\_w]<= rf\_data\_w;//写入数据

end

代码解释：rf\_wen信号有效（其值为1）代表此时进行写操作，将输入的数据内容rf\_data\_w写入地址为rf\_addr\_w的寄存器之中。

仿真代码：

module rf\_tb();

//输入端口

reg clk;

reg rf\_wen;

reg[4:0] rf\_addr\_r1;

reg[4:0] rf\_addr\_r2;

reg[4:0] rf\_addr\_w;

reg[31:0] rf\_data\_w;

//输出端口

wire[31:0] rf\_data\_r1;

wire[31:0] rf\_data\_r2;

//实例化

RegFile rf0(clk,rf\_wen,rf\_addr\_r1,rf\_addr\_r2,rf\_addr\_w,rf\_data\_w,rf\_data\_r1,rf\_data\_r2);

//初始化

initial begin

clk=0;

rf\_wen=0;

rf\_addr\_r1=0;

rf\_addr\_r2=0;

rf\_data\_w=0;

rf\_addr\_w=0;

#100 rf\_wen=1;

#10 rf\_data\_w=119;

rf\_addr\_w=1;//1号寄存器写入值119

#50 rf\_data\_w=34;

rf\_addr\_w=2;//2号寄存器写入值34

#25 rf\_wen=1;

#50 rf\_addr\_r1=1;

rf\_addr\_r2=2;

end

//时钟控制

always begin

#25 clk=~clk;

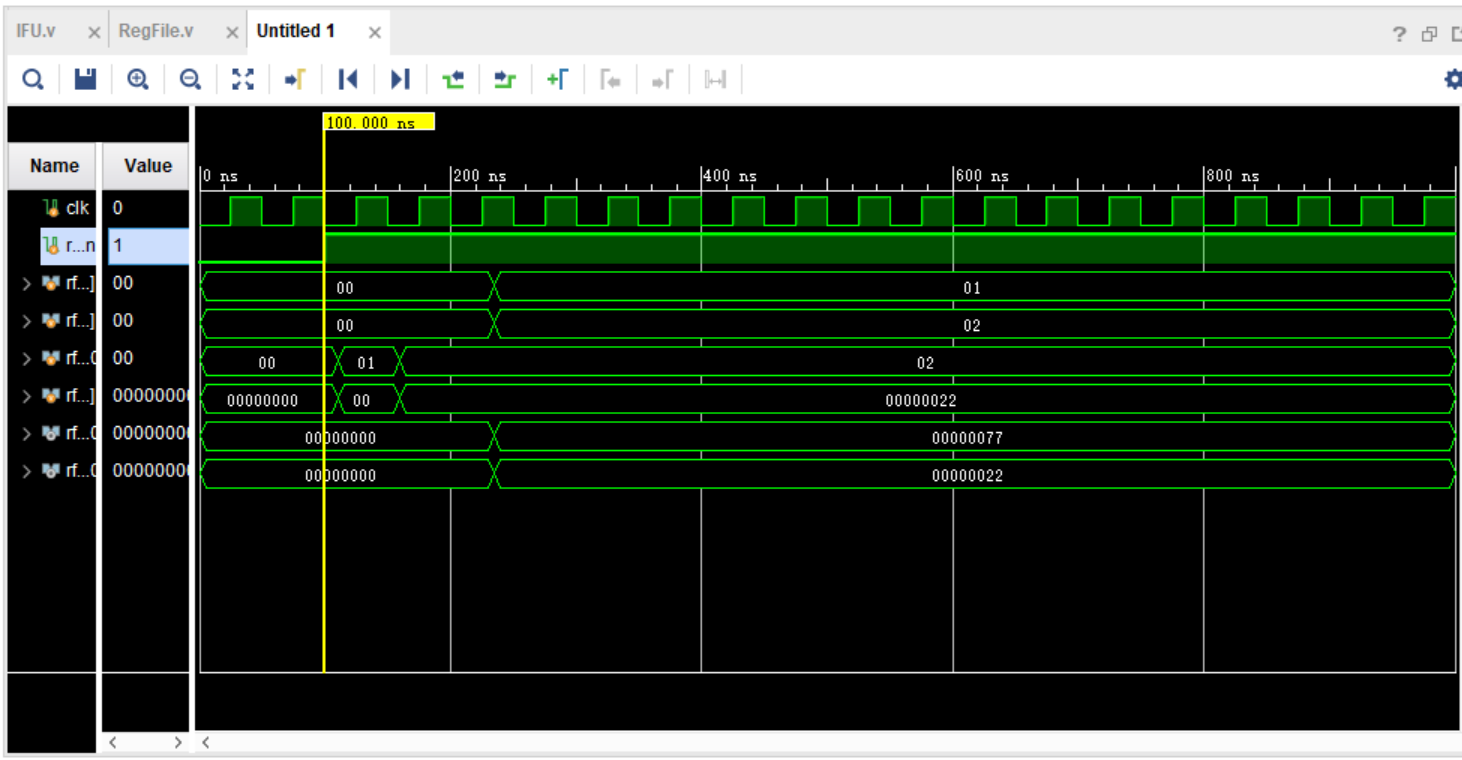
end

endmodule

仿真代码解释:首先为仿真设置时钟信号，RegFile模块部分最核心的操作有两个：①rf\_data\_r1=file[rf\_addr\_r1];和rf\_data\_r2=file[rf\_addr\_r2];，即当寄存器地址为rf\_addr\_r1和rf\_addr\_r2时输出寄存器对应的内容：rf\_data\_r1和rf\_data\_r2.②：当写操作信号rf\_wen有效（值为1时）将输入数据rf\_data\_w输给此时编号为rf\_addr\_w的寄存器；故tb文件的设计核心就是运用这两个读写操作，使写入内容的寄存器号和读出内容的寄存器号相一致，随后验证输入的数据是否能正确输出。（注意此次单周期MIPS中共有32个寄存器，故寄存器的编号赋值范围是：0—31）。

**实验数据：**

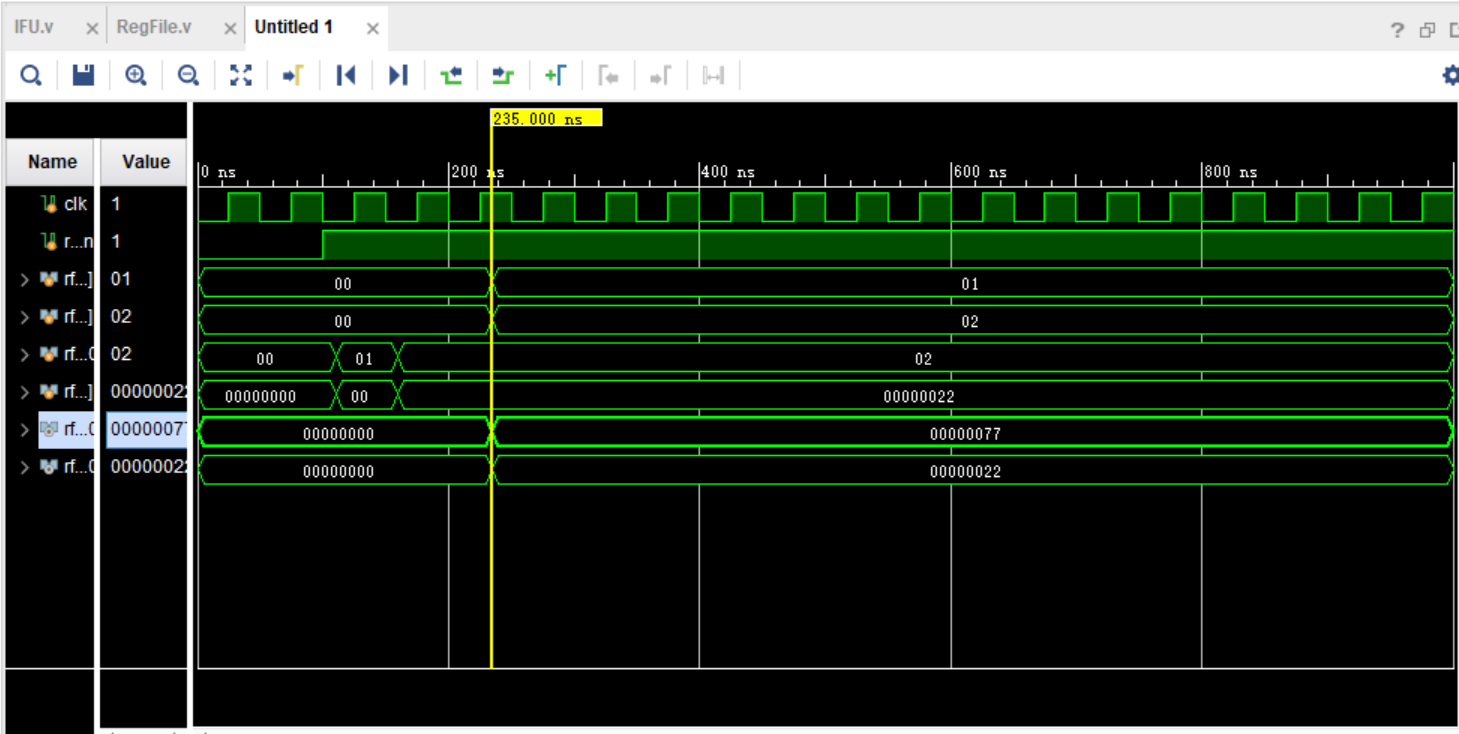
仿真波形图1：



仿真波形1分析：

如图所示，此时写信号（rf\_wen）无效（值为0），故不进行任何操作，寄存器的值也均为0；

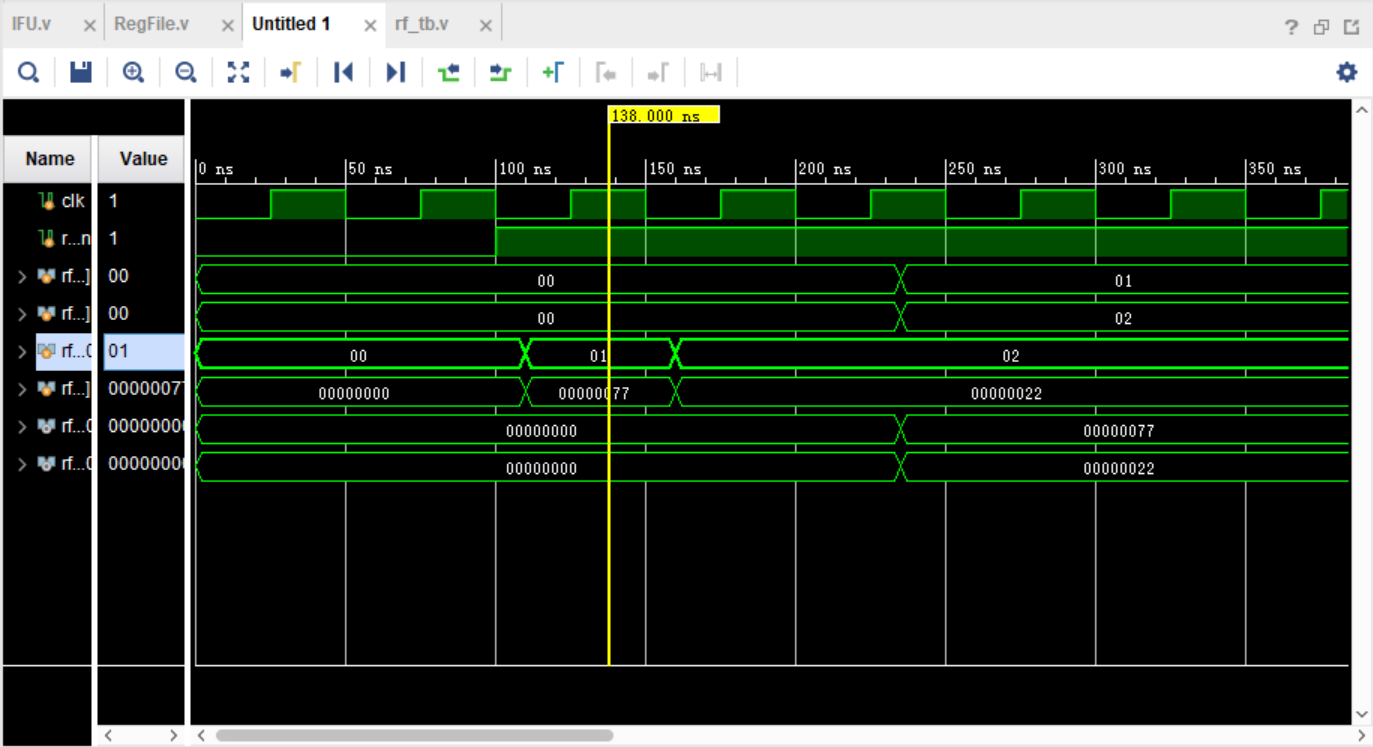
仿真波形图2：



仿真波形2分析：

如图所示，在写信号（rf\_wen）有效时给rf\_data\_w赋值22H，给rf\_addr\_w赋值2，其含义是：将值22H写入编号为2的寄存器中；在tb文件中设置经过几个系统时间后，给rf\_addr\_r2端子赋值为2，意为读取此时编号为2的寄存器中的值，检查rf\_data\_r2端子，输出值为22H，证明刚才写入的数据成功从该寄存器中读出，符合设计要求。

仿真波形3：



仿真波形3分析：

如图所示，在写信号（rf\_wen）有效时给rf\_data\_w赋值77H，给rf\_addr\_w赋值1，其含义是：将值77H写入编号为1的寄存器中；在tb文件中设置经过几个系统时间后，给rf\_addr\_r1端子赋值为1，意为读取此时编号为1的寄存器中的值，检查rf\_data\_r1端子，输出值为77H，证明刚才写入的数据成功从该寄存器中读出，符合设计要求。

**实验结果与分析：**

所设计的MIPS的寄存器堆是32个32位的寄存器（对应代码中的file[31:0]）,RegFile模块的设计是基于在单周期MIPS中，要求能在给出地址的同时就能在出口得到对应的数据；为了节约时钟，此次实验中选取了clk信号的下边沿来检查此时的控制信号（rf\_wen）状态而需不需要改写寄存器中的内容，rf\_wen信号会在时钟脉冲的下边沿接受检查，如果它的值为1，那么就将rf\_data\_w的32位比特数据写入rf\_addr\_w表示的寄存器中，其他四个信号代表读取寄存器地址和数据输出端口，通过仿真波形，在读入地址和读出地址相一致的情况下，输出端数据等于输入端数据，证明设计正确，符合RegFile模块的设计要求。

**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172班

姓名：刘梓瑄 学号：41724039 实验日期：2019年5月15日

**实验名称：**

三．算术逻辑单元模块（ALU）的设计

**实验要求：**

用VerilogHDL或VHDL语言补充完整算术逻辑单元模块。并写出仿真代码，根据仿真波形分析模块的功能，验证其正确性。代码应有适当的注释，并在实验报告中体现；报告中需要有设计分析过程，仿真实现的设计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：**

ALU模块是算术逻辑单元，它可以对两个操作数执行算术运算和逻辑运算。要区分具体进行哪一种运算，就需要输入一个选择控制信号：alu\_ct，由4位二进制代码控制运算的具体类型，4位二进制代码意味着可以编码16种运算。输入信号还有两个操作数，其中一个直接来自寄存器堆RegFile中读出的数据rf\_addr\_r1，另一个可能来自于寄存器堆的输出，也有可能是指令中的立即数符号扩展的结果，这取决于当前指令的指令类型，需要一个二选一数据选择器区分数据的来源。Control模块会给出熟宣器的控制信号alu\_src，。值得特别注意的是，ALU的输出除了运算结果还有一个标志信号用于判断分支指令beq的运算结果。

本模块的输入端口为：

* 1. 置位控制端子rst。
  2. 4位控制信号alu\_ct。
  3. 32位的两个操作数：alu\_src1和slu\_src2。

输出端口为：

* 1. 用于判断分支指令的标志信号alu\_zero。
  2. 运算结果输出：32位的信号alu\_res。

**实验内容与步骤：**

模块补充代码：

case(alu\_ct)

4'b0010:alu\_res=alu\_src1+alu\_src2;//执行加法运算

4'b0110:alu\_res=alu\_src1-alu\_src2;//执行减法运算

仿真代码：

module alu\_tb();

reg rst;

reg[3:0] alu\_ct;

reg[31:0] alu\_src1,alu\_src2;

wire alu\_zero;

wire[31:0] alu\_res;

ALU alu0(rst,alu\_ct,alu\_src1,alu\_src2,alu\_zero,alu\_res);

initial begin

rst=1;

alu\_ct=4'b0010;

alu\_src1=32'b00000000000000000000000000010110;

alu\_src2=32'b00000000000000000000000000010011;

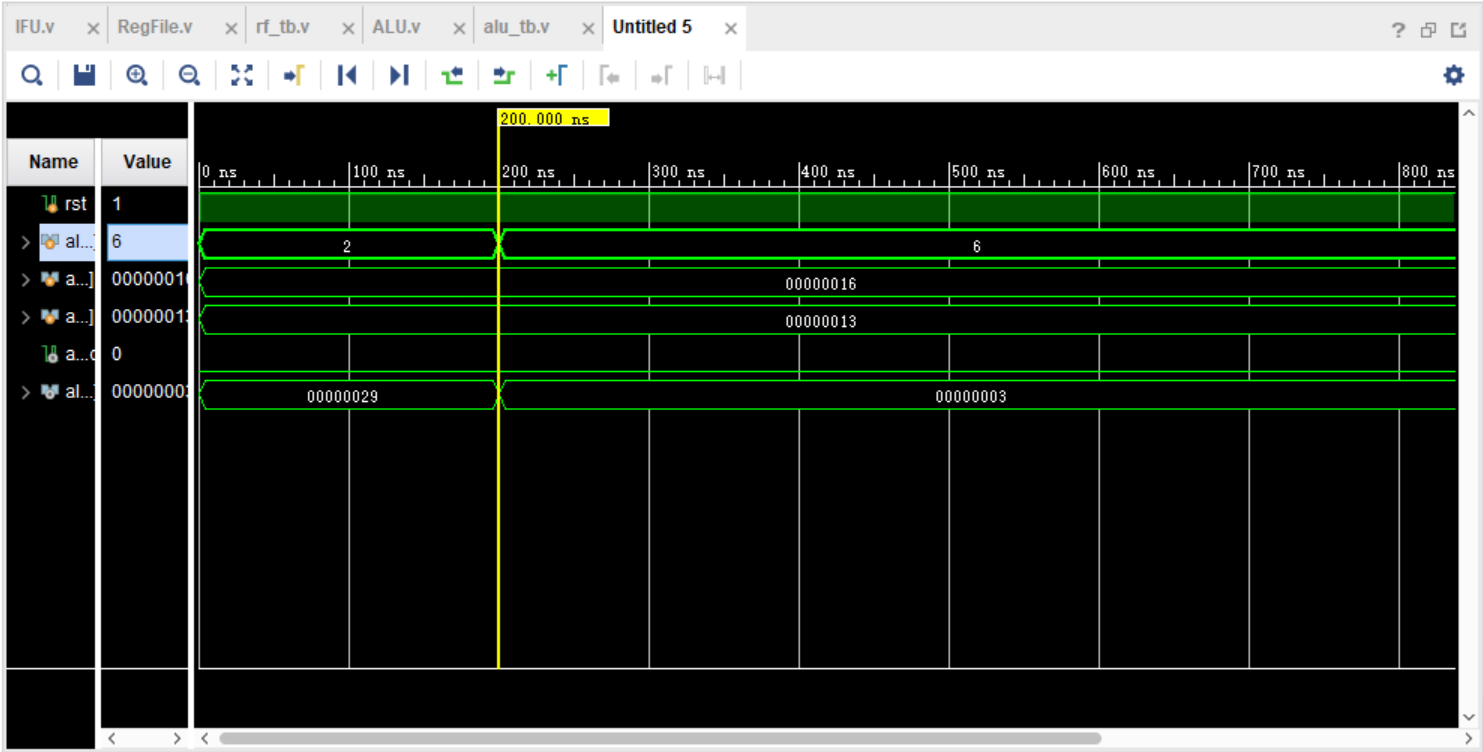
#200 alu\_ct=4'b0110;

end

endmodule

**实验数据：**

仿真波形图：



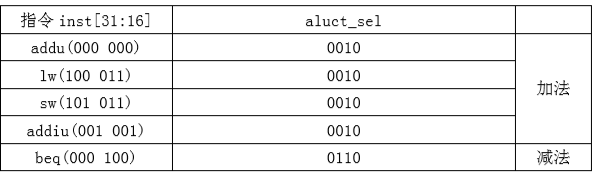
**实验结果与分析：**

仿真波形分析，及实验结果分析

如仿真波形图所示，本次ALU模块的设计中与时钟脉冲clk无关，初始时给控制信号alu\_ct赋值4位二进制代码0010，按设计要求，此时应执行加法运算，给俩个操作数alu\_src1和alu\_src2分别赋值为16和13，此时结果输出端（alu\_res）输出值为29，符合加法运算法则。一定的系统时间后，将控制端子（alu\_ct）的值修改为4位二进制代码0110，按设计要求，此时应执行减法运算，两个操作数的赋值不发生变化，此时结果输出端（alu\_res）输出值为3，符合减法运算法则，综上，完成了算术逻辑单元ALU模块的设计与实现。

实验结论：

在这次单周期MIPS的设计中，ALU主要用于执行算术运算和逻辑运算，算术运算包括加减法，逻辑运算包括比较等。本次仿真中未设置时钟脉冲是因为在单周期CPU中要求实现的是在一个时钟周期能够完成的指令，所以我的设计中，ALU的输出总是与此时的输入和控制信号相对应，ALU中参与运算的指令有R型和I型两种：R型指令中，操作类型由op段和funct段共同确定；I型指令中，指令类型则仅由op段确定，在控制模块control中通过这两段确定指令类型，通过自定义的方式编辑四位的控制信号alu\_ct，控制信号对应表如下：



beq指令的具体含义是在两个输入相等时跳转，采用减法来模拟，若两个数相等，标志信号alu\_zero的值为1（不等时值为0），标志信号alu\_zero作为输入端子进入IFU取值模块，配合其他的控制信号，即可以检测是否能实现跳转功能。

**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172班

姓名：刘梓瑄 学号：41724039 实验日期：2019年5月15日

**实验名称：**

四．数据存储模块（DataMem）的设计

**实验要求：**

用VerilogHDL或VHDL语言补充完整数据存储模块，并写出仿真代码，根据仿真波形分析模块的功能，验证其正确性。代码应有适当的注释，并在实验报告中体现；报告中需要有设计分析过程，仿真实现的设计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：**

DataMem模块需要存储运算后的数据，或者直接存储来自寄存器堆读出的数据，所以在这个模块中主要实现并验证它的读写功能。  
 ①读操作的实现：lw指令需要使用DataMem的读操作从DataMem读取到指定的寄存器当中，读操作的地址经ALU算术逻辑单元运算后得到alu\_res，它是由lw中的rs字段寄存器的值加上offset字段符号拓展后的值，DataMem模块设计中需要使用到来自Control给出读使能信号mem\_ren,读出的数据从端口mem\_data\_o输出。

②写操作的实现：sw指令需要使用DataMem的写操作将指定寄存器中的数据存入到DataMem模块中，写入的地址与读操作获得地址的方法一致，写入的数据是rt寄存器中的值，写操作由写使能信号mem\_wen控制。

本模块的输入端子有：

* + 1. 时钟脉冲信号clk。
    2. 写控制信号：mem\_wen。
    3. 读控制信号：mem\_ren。
    4. 进行读写操作的地址信号：mem\_addr[31:0]。
    5. 输入数据：mem\_data\_i[31:0]。

本模块的输出端子有：

数据输出端子：

1. mem\_data\_o[31:0]。

**实验内容与步骤：**

模块补充代码：

assign mem\_data\_o = mem\_ren ? ({data\_mem3[mem\_addr[31:2]], data\_mem2[mem\_addr[31:2]], data\_mem1[mem\_addr[31:2]], data\_mem0[mem\_addr[31:2]]}):32'bz;

仿真代码：

clk=0;

mem\_wen=0;

mem\_ren=0;

mem\_addr=0;

mem\_data\_i=0;

end

parameter PERIOD=20;

always begin

clk=0;

#(PERIOD/2) clk=~clk;

#(PERIOD/2);

end

always begin

#60 mem\_wen=1;

mem\_data\_i=32'b00000000000000000000001000110000;

mem\_addr=1;

#60 mem\_ren=1;

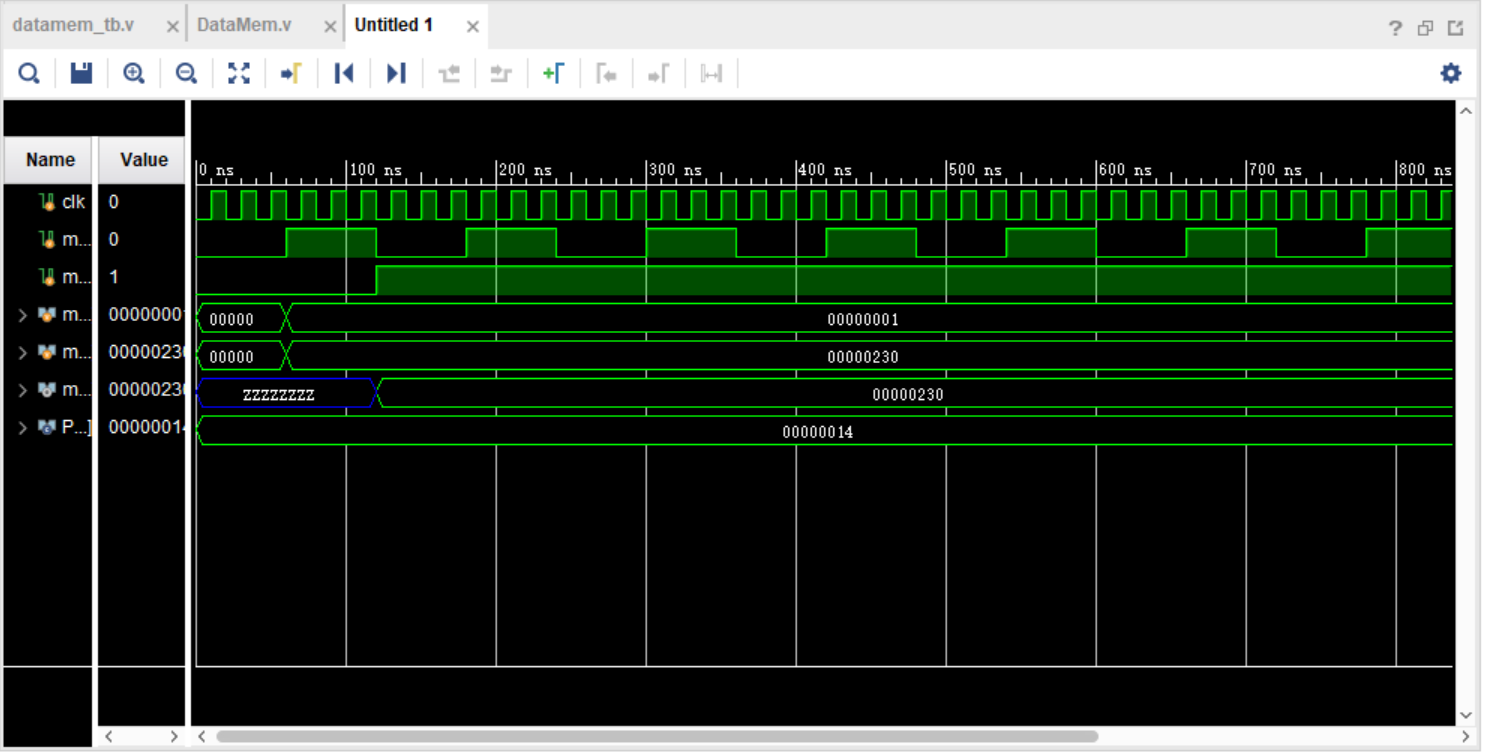
mem\_wen=0;

end

endmodule

**实验数据：**

仿真波形图：



**实验结果与分析：**

仿真波形分析，及实验结果分析

初始时写信号mem\_ren无效（值为0），写信号mem\_wen在时钟脉冲的下降沿有效（值为1），实现将数据从数据输入端mem\_data\_i写入mem\_addr表示的内存地址，随后写信号mem\_ren有效（值为1），将mem\_addr内存地址中存入的数据传送到数据输出端口mem\_data\_o;从波形图中可以看出，在读信号有效的时候，mem\_data\_i输入端口的值为230，随后在仿真波形中控制写信号mem\_wen有效，mem\_data\_o输出端输出内容230，说明该部分功能设计正确，Datamem模块功能设计正常。至此，DataMem模块完成了可以存储运算后的数据，或者直接存储来自寄存器堆数据的功能，DataMem完成了数据读写与存储功能，该模块是CPU中的数据存储器，可用于存储运算结果或操作数，其输出端口输出的数据再经由数据选择器送入寄存器模块，等待下一次参与运算。

**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172班

姓名：刘梓瑄 学号：41724039 实验日期：2019年5月15日

**实验名称：**

五．控制器模块（Control）的设计

**实验要求：**

用VerilogHDL或VHDL语言补充完整控制器模块Control和ALU控制模块ALUCt，并分别写出仿真代码，根据仿真波形分析模块的功能，验证其正确性。代码应有适当的注释，并在实验报告中体现；报告中需要有设计分析过程，仿真实现的设计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：**

Control分为两部分完成设计，该模块的输入是指令的高六位与指令的低六位，即op码字段和func码字段，输出是所用用于控制指令执行的控制信号。最高六位（op）码字段用于区分不同的指令，最低六位（func）码字段用于区分R型指令的不同操作，因为本次设计中实现的不止有R型指令，所以将指令低六位和由Control模块根据不同指令生成的2bit的ct\_alu\_op控制信号一起送入子模块ALUCT（ALU控制），ALUCT返回4bit的alu\_ct信号由Control输出。

**实验内容与步骤：**

1. Control模块的实现

输入信号：

<1>指令的高六位(op)字段。

<2>指令的低六位(func)字段。

<3>控制信号rst。

输出信号：

<1>各条指令按真值表所对应的控制端子信号。

模块补充代码：

//与阵

assign inst\_r=(!ct\_inst[5])&&(!ct\_inst[4])&&(!ct\_inst[3])&&(!ct\_inst[2])&&(!ct\_inst[1])&&(!ct\_inst[0]);

assign inst\_lw=(ct\_inst[5])&&(!ct\_inst[4])&&(!ct\_inst[3])&&(!ct\_inst[2])&&(ct\_inst[1])&&(ct\_inst[0]);

assign inst\_sw=(ct\_inst[5])&&(!ct\_inst[4])&&(ct\_inst[3])&&(!ct\_inst[2])&&(ct\_inst[1])&&(ct\_inst[0]);

assign inst\_beq=(!ct\_inst[5])&&(!ct\_inst[4])&&(!ct\_inst[3])&&(ct\_inst[2])&&(!ct\_inst[1])&&(!ct\_inst[0]);

assign inst\_j=(!ct\_inst[5])&&(!ct\_inst[4])&&(!ct\_inst[3])&&(!ct\_inst[2])&&(ct\_inst[1])&&(!ct\_inst[0]);

assign inst\_addiu=(!ct\_inst[5])&&(!ct\_inst[4])&&(ct\_inst[3])&&(!ct\_inst[2])&&(!ct\_inst[1])&&(ct\_inst[0]);

//或阵

assign ct\_rf\_dst=rst?inst\_r:0;

assign ct\_rf\_wen=rst?inst\_r||inst\_lw||inst\_addiu:0;

assign ct\_alu\_src=inst\_lw||inst\_sw||inst\_addiu;

assign ct\_alu\_op[1:0]={inst\_r,inst\_beq};

assign ct\_branch= rst?inst\_beq:0;

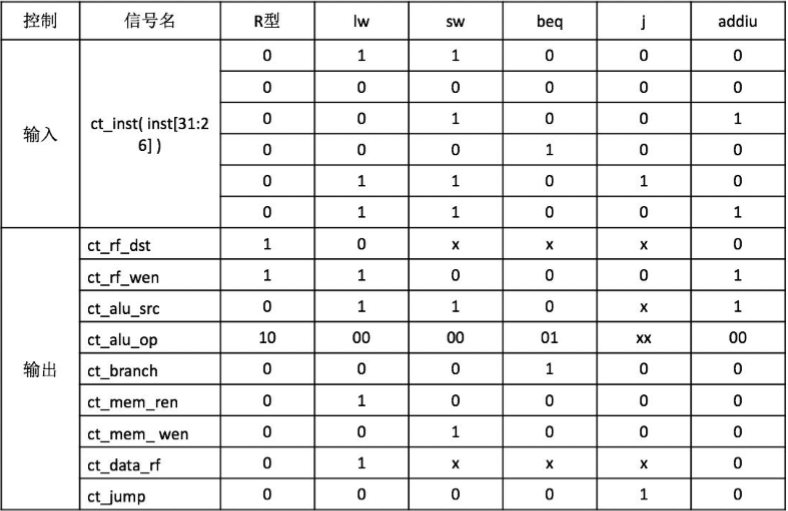
assign ct\_mem\_ren = rst?inst\_lw:0;

assign ct\_mem\_wen=rst?inst\_sw:0;

assign ct\_data\_rf=rst?inst\_lw:0;

assign ct\_jump=rst?inst\_j:0;

设计思路，依照各指令与其控制信号的对照表：



与阵实现了前6位的op段字码，或阵对照上述真值表实现控制信号的逻辑表达式，如上述真值表，可以实现不同指令对应不同的控制信号，从而通过这些控制信号实现不同的操作，从而达到实现指令功能的目的。

仿真代码：

module ctrl\_tb();

//inputs

reg rst;

reg[5:0] ct\_inst;

reg[5:0] aluct\_inst;

//outputs

wire ct\_rf\_dst;

wire ct\_rf\_wen;

wire ct\_alu\_src;

wire ct\_alu;

wire ct\_mem\_wen;

wire ct\_mem\_ren;

wire ct\_data\_rf;

wire ct\_branch;

wire ct\_jump;

//实例化

ctrl ct0(rst,ct\_inst,aluct\_inst,ct\_rf\_dst,ct\_rf\_wen,ct\_alu\_src,ct\_alu,ct\_mem\_wen,ct\_mem\_ren,ct\_data\_rf,ct\_branch,ct\_jump);

//初始化

initial begin

rst=0;

ct\_inst=0;

aluct\_inst=0;

end

always begin

#60 rst=1;

ct\_inst=6'b001001;

#60

ct\_inst=6'b100011;

#60

ct\_inst=6'b101011;

#60

ct\_inst=6'b000100;

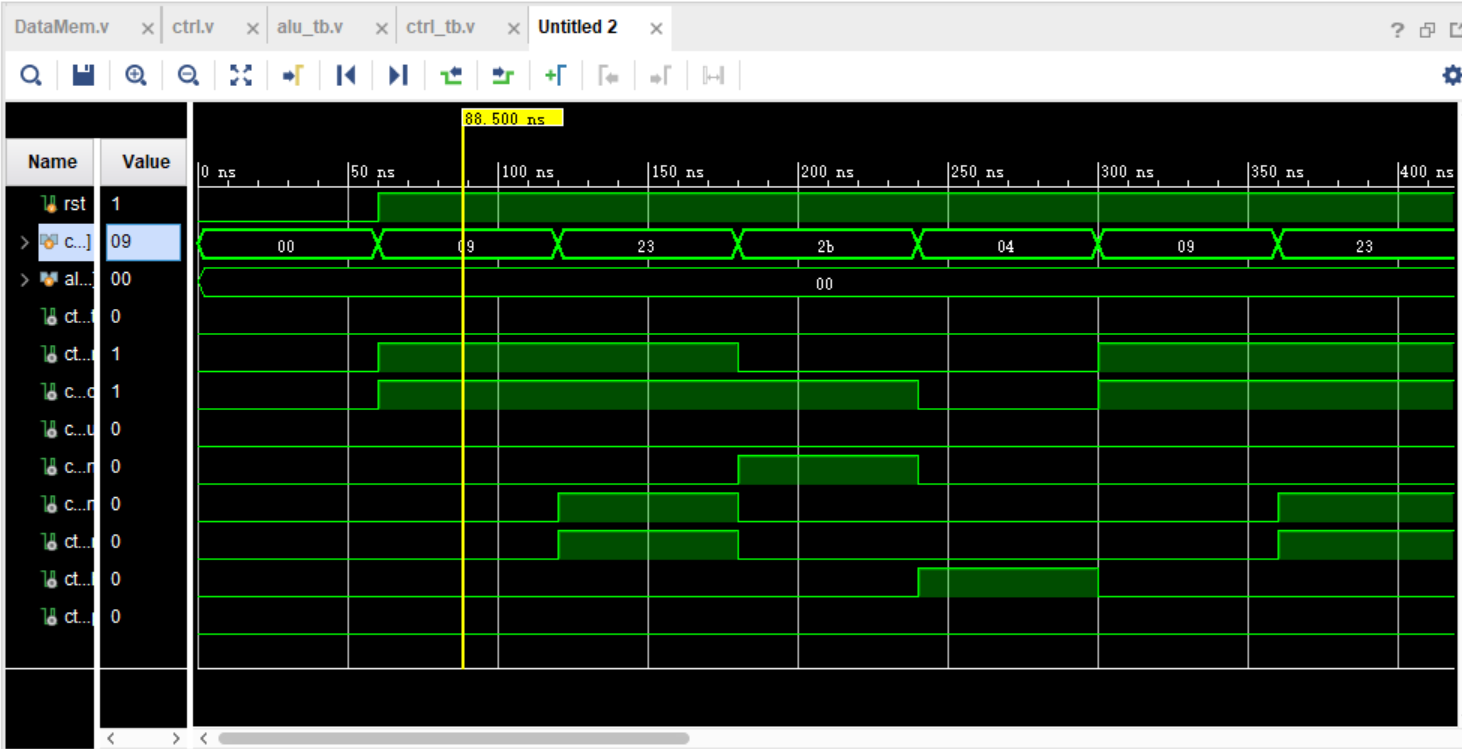
end

endmodule

仿真波形的设计思路：输入信号为指令的高六位，对照此时的输出端子的真值情况与真值表是否符合，从而可以验证该模块功能是否设计正确。

**实验数据：**

仿真波形图：



如图高六位为09时，对应指令的高六位为001001，此时ct\_rf\_dst真值为1，ct\_rf\_wen真值为1，ct\_alu\_src真值为0，ct\_alu\_op真值为10，ct\_branch真值为0，ct\_mem\_ren真值为0，ct\_mem\_wen真值为0，ct\_data\_rf真值为0，ct\_jump真值为0，设计与真值表相符。依次检查其他指令，各控制信号均与对应指令相符，说明Control模块设计正确。

1. ALUControl模块的实现：

输入端子：

<1>控制信号rst。

<2>指令的高六位（func）码字段。

<3>2位的控制信号alu\_ct\_op。

输出端子为：

控制具体运算内容的4位控制信号alu\_ct。

模块补充代码：

parameter ADD=6'b100001;

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)//R型指令

ADD:begin

alu\_ct<=4'b0010;

end

default:begin

end

设计思路：ALUCT是Control模块的子模块，指令中的低六位（func）字段和两位的ct\_alu\_op作为内部信号送到子模块ALUCT中，ALUCT中产生四位的控制信号ct\_alu信号由Control模块输出，送入算术逻辑单元ALU中定义具体的运算过程。

仿真代码：

module aluctrl\_tb();

//inputs

reg rst;

reg[5:0] funct;

reg[1:0] alu\_ct\_op;

//outputs

wire [3:0] alu\_ct;

//实例化

ALUCT aluct0(rst,funct,alu\_ct\_op,alu\_ct);

//初始化

initial begin

rst=0;

funct=0;

alu\_ct\_op=0;

end

always begin

#20 rst=1;

#60 funct=6'b100001;

alu\_ct\_op=2'b10;

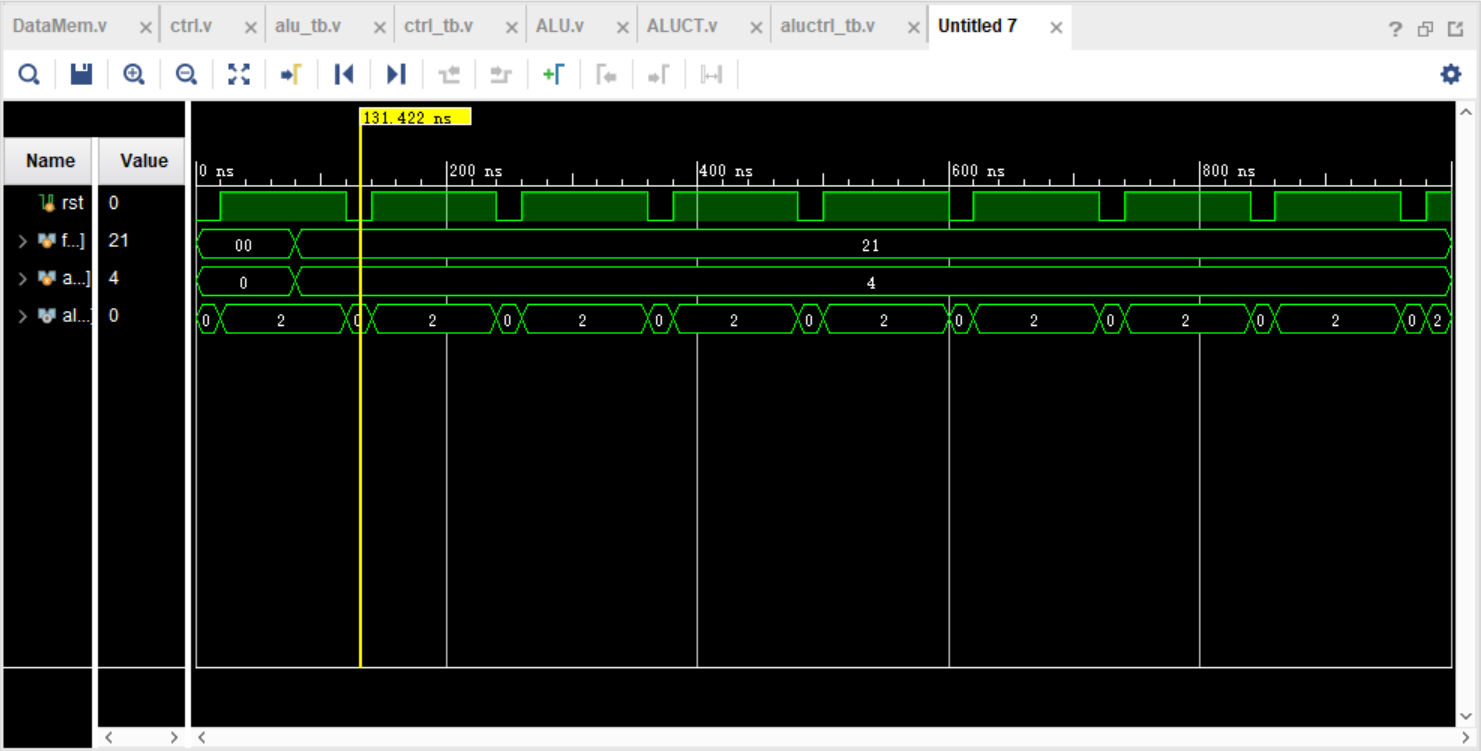
#40 rst=0;

end

endmodule

**实验数据：**

仿真波形图：



**实验结果与分析：**

仿真波形分析，及实验结果分析

从仿真波形图中，将func字段赋值一信号21，对应二进制代码为100001，另一控制信号alu\_ct字段应为三位二进制代码100（此时所用于验证波形图已经是完成拓展指令后的波形图，故alu\_ct\_op是三位的控制信号），从波形图中可以看出输出端子alu\_ct\_op的值为0010，对应执行加法操作，符合设计要求，说明ALUCT模块设计正确。

实验结论：Control模块主要是实现了控制各指令操作的控制端子的逻辑表达式的实现，对应已经设计好的控制端子真值表，不同指令下各控制端子的真值有所不同，对于I型指令，指令类型仅由op段决定。对于R型指令，具体指令则由op段和func段共同决定，R型指令生成相应的控制信号alu\_ct，该信号通过设计者自定义完成，送往ALU算术逻辑单元，用于控制具体的运算内容。

**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172班

姓名：刘梓瑄 学号：41724039 实验日期：2019年5月15日

**实验名称：**

六．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；报告中需要有设计分析过程，仿真实现的设计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：**

我此次拓展的两条指令分别是I型指令andi(实现逻辑与)，R型指令slt(实现比较大小)，分为以下几个步骤进行：

1. 分析不同的指令类型，指令字段分别对应二进制码的哪些部分，指令特点和操作的具体内容。
2. 依据以上分析列出拓展指令对应控制端子的真值表，修改相关代码。
3. 明确拓展指令实现内容，为MIPS添加实际的运算。
4. 设计仿真文件，将32位的二进制翻译为16位，设计新的数据文件。
5. 测试与分析。

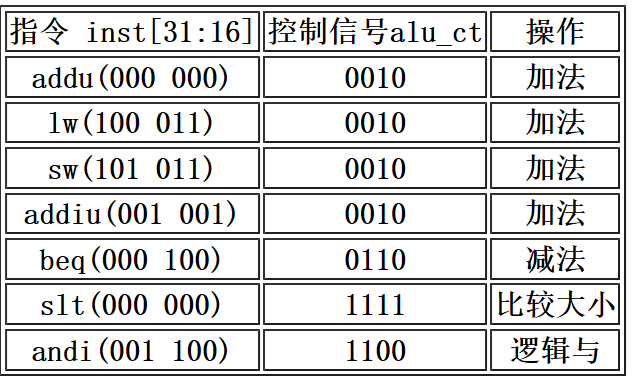
**实验内容与步骤：**

1. 分析不同的指令类型，指令字段分别对应二进制码的哪些部分，指令特点和操作的具体内容。

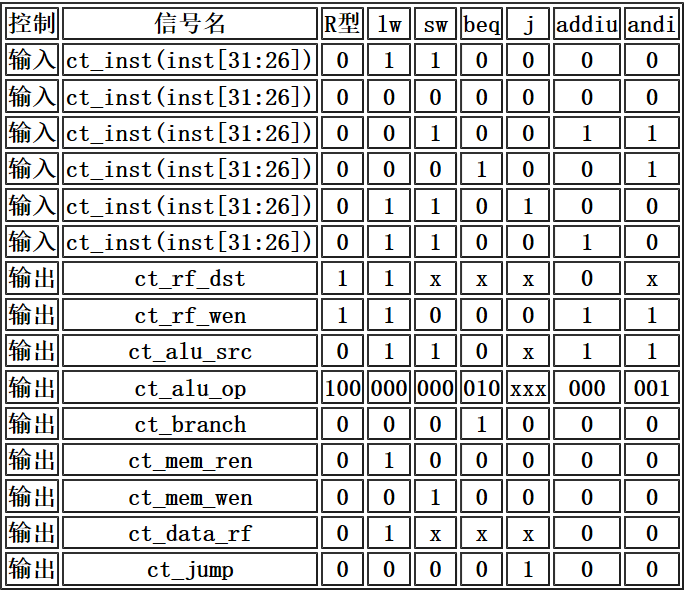
slt：R型指令，操作由op段和func段共同决定，op:000 000，func:101010，实现操作：两寄存器内的数比较大小

andi：I型指令，操作仅由op段决定，op：001100，实现操作：两寄存器内的数实现按位与。

1. 拓展指令后自定义的alu\_ct段控制信号参照表如下图：

****

拓展指令后各指令对应控制端子真值表如下图：

****

给出关键代码并加注释：

Control模块关键代码：

//与阵

assign inst\_r=(!ct\_inst[5])&&(!ct\_inst[4])&&(!ct\_inst[3])&&(!ct\_inst[2])&&(!ct\_inst[1])&&(!ct\_inst[0]);

assign inst\_lw=(ct\_inst[5])&&(!ct\_inst[4])&&(!ct\_inst[3])&&(!ct\_inst[2])&&(ct\_inst[1])&&(ct\_inst[0]);

assign inst\_sw=(ct\_inst[5])&&(!ct\_inst[4])&&(ct\_inst[3])&&(!ct\_inst[2])&&(ct\_inst[1])&&(ct\_inst[0]);

assign inst\_beq=(!ct\_inst[5])&&(!ct\_inst[4])&&(!ct\_inst[3])&&(ct\_inst[2])&&(!ct\_inst[1])&&(!ct\_inst[0]);

assign inst\_j=(!ct\_inst[5])&&(!ct\_inst[4])&&(!ct\_inst[3])&&(!ct\_inst[2])&&(ct\_inst[1])&&(!ct\_inst[0]);

assign inst\_addiu=(!ct\_inst[5])&&(!ct\_inst[4])&&(ct\_inst[3])&&(!ct\_inst[2])&&(!ct\_inst[1])&&(ct\_inst[0]);

assign inst\_andi=(!ct\_inst[5])&&(!ct\_inst[4])&&(ct\_inst[3])&&(ct\_inst[2])&&(!ct\_inst[1])&&(!ct\_inst[0]);

//或阵

assign ct\_rf\_dst=rst?inst\_r:0;

assign ct\_rf\_wen=rst?(inst\_r||inst\_lw||inst\_addiu||inst\_andi):0;

assign ct\_alu\_src=(inst\_lw||inst\_sw||inst\_addiu||inst\_andi);

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

assign ct\_branch= rst?inst\_beq:0;

assign ct\_mem\_ren = rst?inst\_lw:0;

assign ct\_mem\_wen=rst?inst\_sw:0;

assign ct\_data\_rf=rst?inst\_lw:0;

assign ct\_jump=rst?inst\_j:0;:

设计思路：按上述所列真值表修改信号，由于扩展了1条I型指令，将alu\_ct\_op控制信号修改为3位。

ALUCT模块关键代码：

case(alu\_ct\_op)

3'b000:alu\_ct<=4'b0010;

3'b001:alu\_ct<=4'b1100;

3'b010:alu\_ct<=4'b0110;

3'b100:begin case(funct)//R型指令

ADD:alu\_ct<=4'b0010;

COMP:alu\_ct<=4'b1111;//拓展slt指令

设计思路，添加一条I型指令andi和1条R型指令slt，并为它们自定义具体的ALU控制信号，各不同指令与控制信号alu\_ct的对应关系见上表。

ALU模块关键代码：

case(alu\_ct)

4'b0010:alu\_res=alu\_src1+alu\_src2;

4'b0110:alu\_res=alu\_src1-alu\_src2;

4'b1111:alu\_res=(alu\_src1<alu\_src2)?1:0;//拓展指令slt,两操作数比较大小

4'b1100:alu\_res=(alu\_src2&alu\_src1);//拓展指令andi,两操作数相与

设计思路：该模块主要实现拓展指令后，算术逻辑单元ALU所添加的两种运算：比较大小与逻辑与运算，该部分代码主要定义了这两种运算的具体实现。

给出关键仿真代码：

module top\_tb();

reg clk,rst;

//实例化

CPU cpu0(clk,rst);

//初始化

initial begin

clk=0;

rst=1;

end

always begin

#5 clk=~clk;//时钟周期

end

initial begin

#1 rst=0;

#20 rst=1;

#1000 $stop;//设置运行时间上限

end

endmodule

添加了新的指令文件inst1.data，用于验证两条拓展指令是否能正确实现设计功能。

inst1.data文件内容如下：

00000000

//第零条指令

24000001

//addiu $0 $0 1

//给零号寄存器赋立即数1

24420002

//addiu $2 $2 2

//给2号寄存器赋立即数2

30410003

//andi $2 $1 3

//验证与操作,2号寄存器中内容与立即数3做按位与，操作结果存入1号寄存器当中

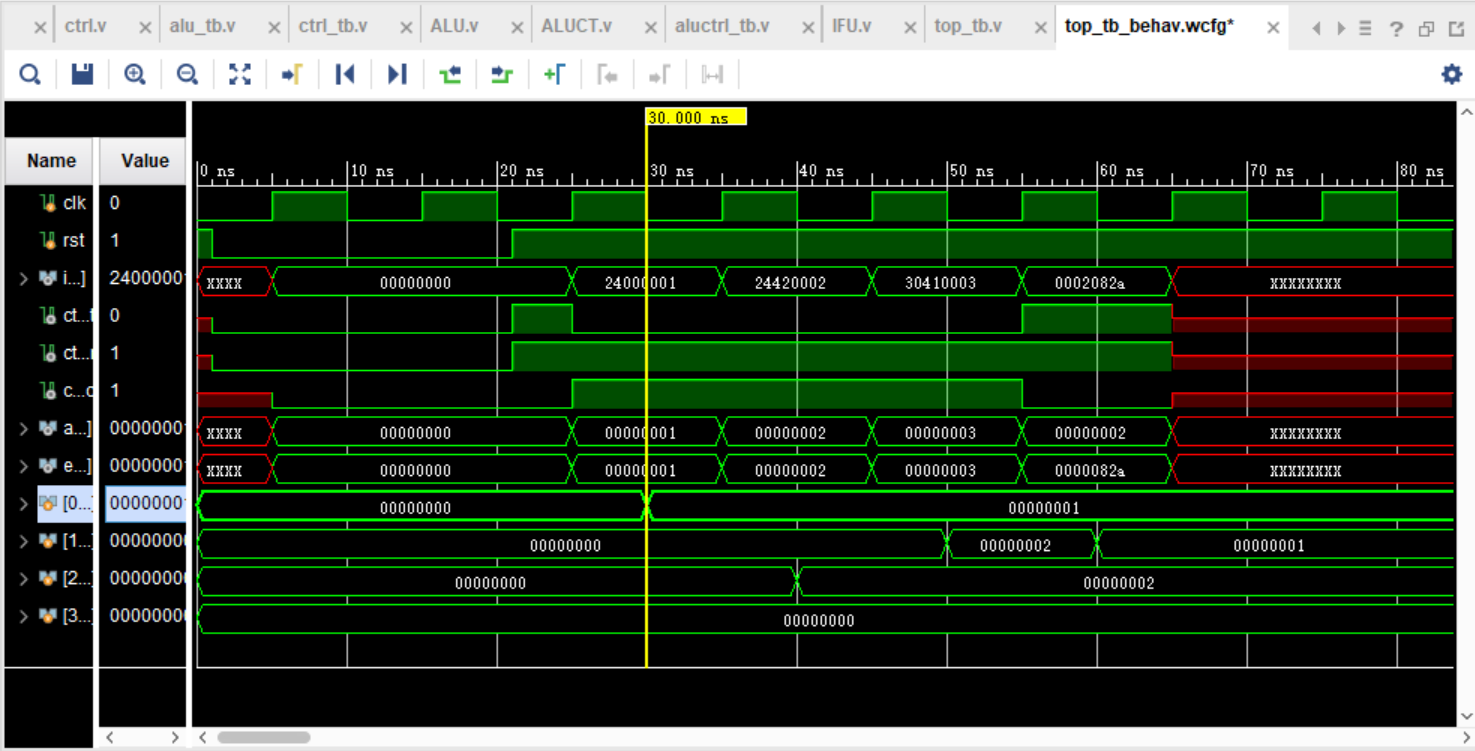
0002082a

//slt $0 $2 $1，0号寄存器和2号寄存器内容进行比较，比较结果放入1号寄存器中

**实验数据：**

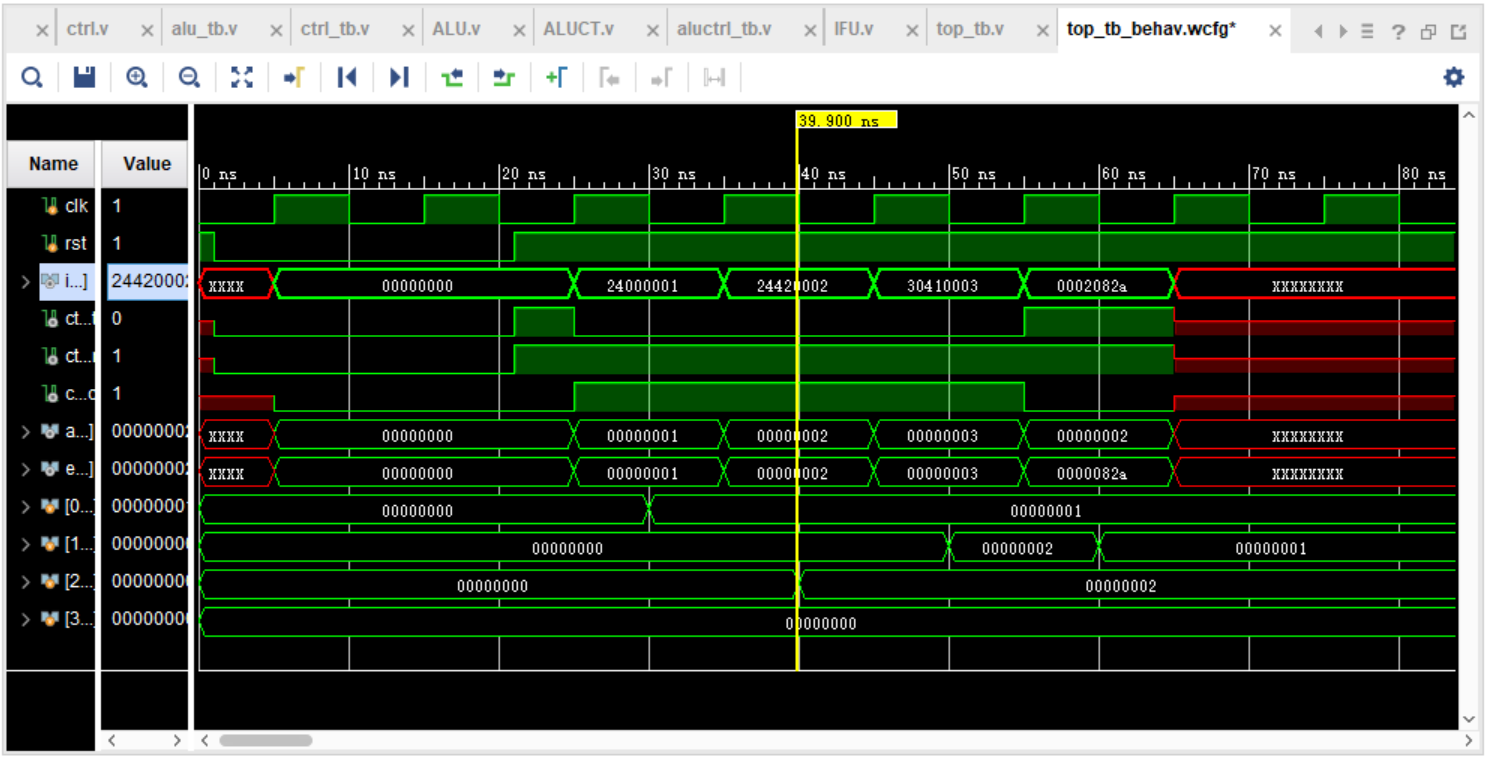
仿真波形图：

仿真波形图1：



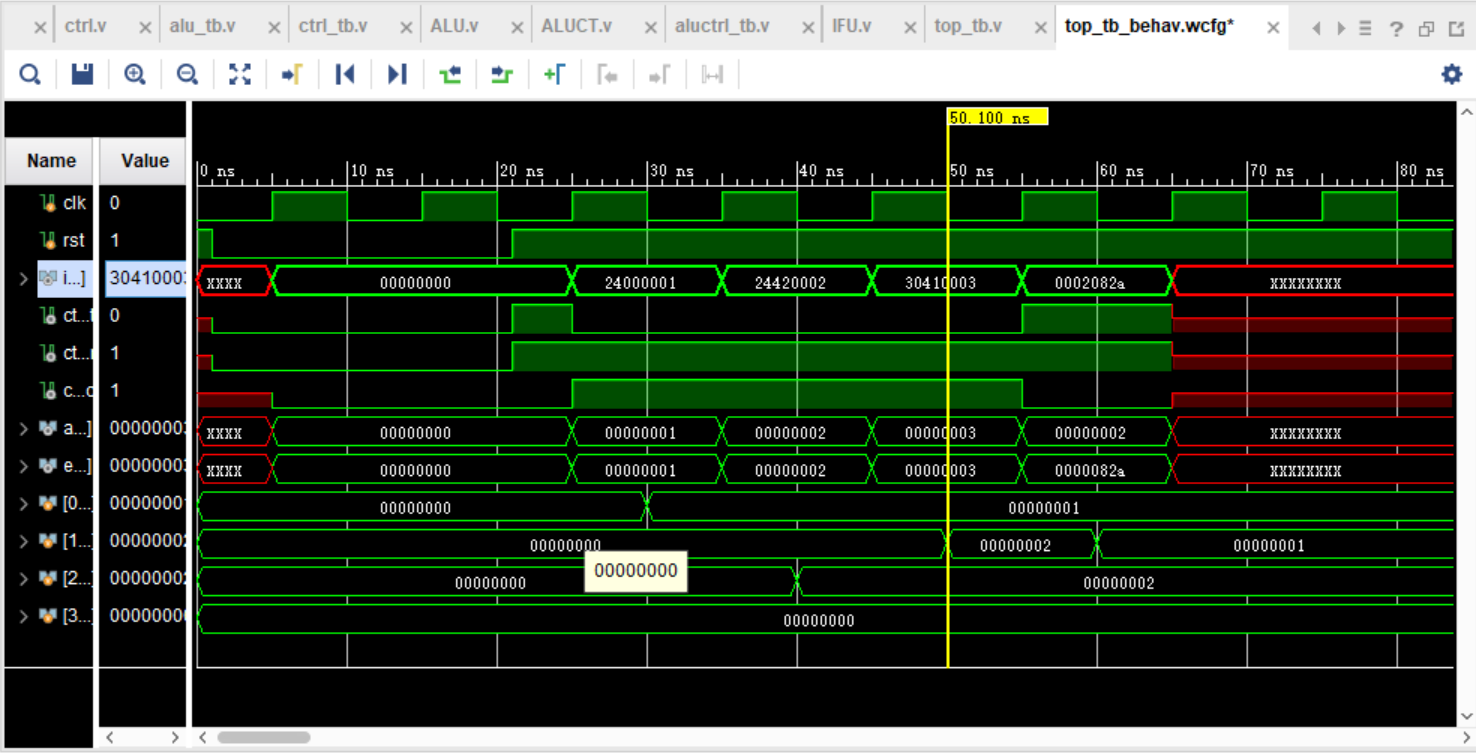
如图第一条指令执行前，0号寄存器内容为0，执行后内容为1，功能正常。

仿真波形图2：



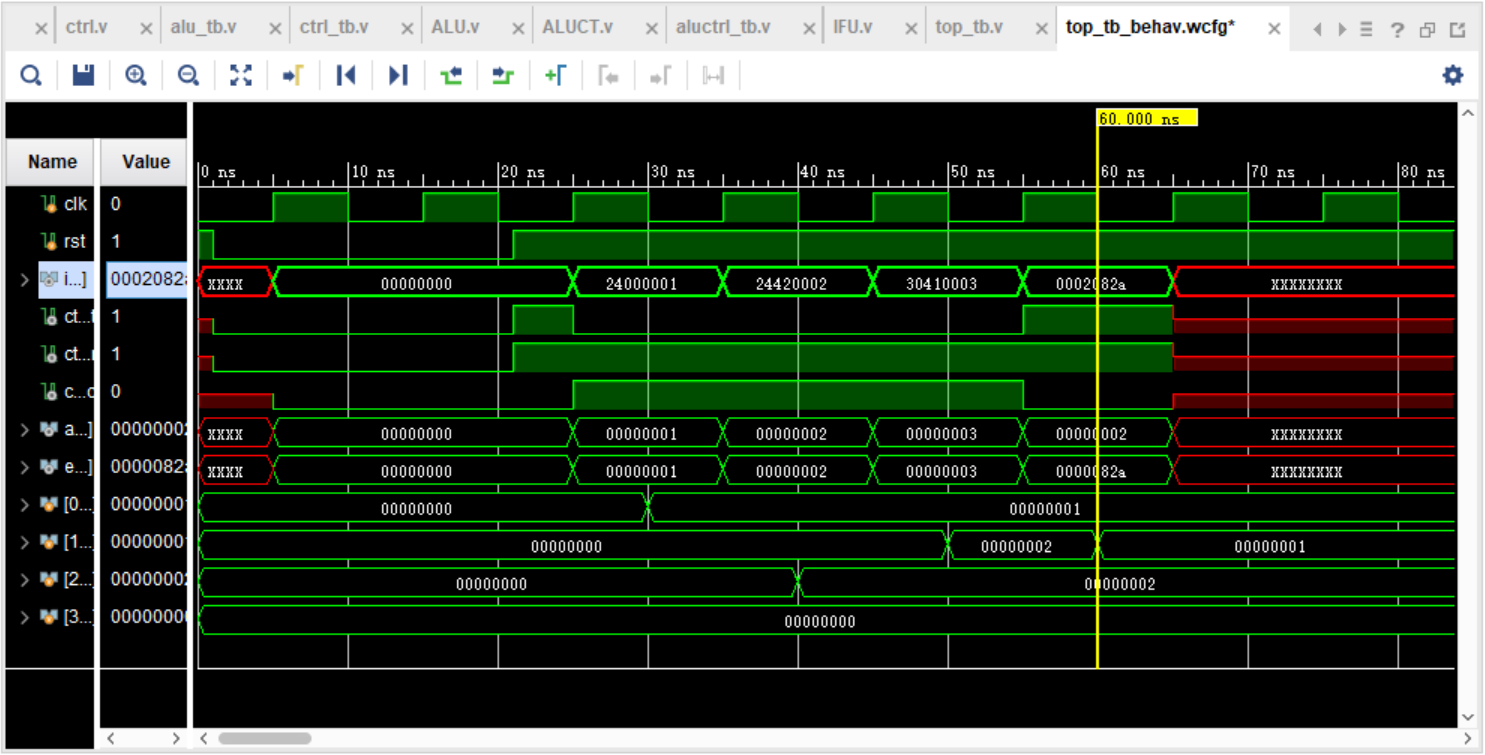
如图第二条指令执行前，2号寄存器内容为0，执行后为2，功能正常。

仿真波形图3：



如图第三条指令执行时，2号寄存器中内容2（10）与3（11）做按位与，结果为2，第三条指令执行后，1号寄存器内内容为2，设计正常。

仿真波形图4：



如图第四条指令执行时，0号寄存器内容1与2号寄存器内容2作比较，$0<$2，返回值为1 ，指令执行后，寄存器1内值变为1，功能正常。

**实验结果与分析：**

综上完成了两条指令：andi和slt指令的设计过程，且验证功能正确。完成了单周期MIPS的设计过程，这个MIPS共可实现8条指令，采用32位编码，处理器采用寄存器组结构（共含有32个寄存器），实现了三种寻址方式（寄存器寻址，基址加偏移寻址，PC相对寻址），共含有5大功能部件：IFU取值模块，RegFile寄存器堆模块，ALU算术逻辑单元，DataMem数据存储器模块，Control控制模块。本次单周期MIPS的设计，是通过给定的指令集，依据给定的数据通路和控制单元信号进行设计，以达到实现不同指令功能的目的。