《计算机组成原理实验》

实验报告

学 院 名 称 ：	计算机学院
专业（班级）：	计算机科学与技术
学 生 姓 名 ：	刘冠麟
学　　号 ：	20319045
时　　间 ：	2022 年 12 月 28 日

成 绩 ：

流水线CPU设计与实现

一. 实验目的

1) 了解流水线CPU基本功能部件的设计与实现方法，

2) 了解提高CPU性能的方法。

3) 掌握流水线MIPS微处理器的工作原理。

4) 理解数据冒险、控制冒险的概念以及流水线冲突的解决方法。

5) 掌握流水线MIPS微处理器的测试方法。

二. 实验内容

本实验实现了兼容MIPS指令集的32位五级流水线处理器，该处理器能至少处理49条指令。在本实验中进行的实验内容大体如下：

①在单周期CPU的基础上进行改进，增加了IF/ID模块、ID/EX模块、EX/MEM模块、MEM/WB模块等流水线中间寄存器，并增加了部分时钟控制，改进为了五级整数流水线CPU。

②在上一个单周期CPU的实验完成的指令的基础上增加实现了许多新指令。

③实现了转发，解决了数据冒险并且改善了部分控制冒险的现象。

④实现了至少49条指令，且大多数指令能在一个周期的时间内完成。

⑤完成了除法、乘法以及按半字存取、按字存取等难度较高的指令。

三. 实验原理

1.概述：流水线是指将计算机指令处理过程拆分为多个步骤，并通过多个硬件处理单元并行执行来加快指令的执行速度。指令的每步有各自独立的电路来处理，每完成一步，就进到下一步，而前一步则处理后续指令。

采用流水线技术后，并没有加速单条指令的执行，每条指令的操作步骤一个也不能少，只是多条指令的不同操作步骤同时执行，因而从总体上看加快了指令流速度，缩短了程序执行时间。流水线本质的工作原理是加快了数据的吞吐量，虽然单独执行每一条指令的时间并没有缩短，但是利用并行的原理，当要处理多条指令时加快了整体的处理速度。

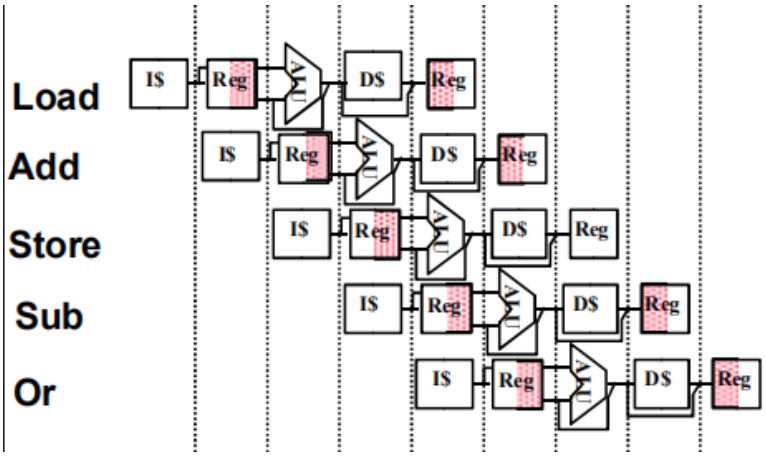


Figure 1 流水线示意图

显然，流水线级数越多，每级所花的时间越短，时钟周期就可以设计的越短，指令速度越快，指令平均执行时间也就越短。本次实验实现了五级流水线CPU，具体的运行过程如下所示：

①取指令（IF）阶段——②译码（ID）阶段——③执行（EX）阶段——④访存（MEM）阶段——⑤写回（WB）阶段。

其中cpu在各个阶段进行的操作为：

①取指令(IF)：根据程序计数器PC中的指令地址，从存储器中取出一条指令，同时，PC根据指令字长度自动递增产生下一条指令所需要的指令地址，但遇到“地址转移”指令时，则控制器把“转移地址”送入PC，当然得到的“地址”需要做些变换才送入PC。

②指令译码(ID)：对取指令操作中得到的指令进行分析并译码，确定这条指令需要完成的操作，从而产生相应的操作控制信号，用于驱动执行状态中的各种操作。

③指令执行(EXE)：根据指令译码得到的操作控制信号，具体地执行指令动作，然后转移到结果写回状态。

④存储器访问(MEM)：所有需要访问存储器的操作都将在这个步骤中执行，该步骤给出存储器的数据地址，把数据写入到存储器中数据地址所指定的存储单元或者从存储器中得到数据地址单元中的数据。

⑤结果写回(WB)：指令执行的结果或者访问存储器中得到的数据写回相应的目的寄存器中。

2.MIPS指令集

本实验采用的指令集格式为MIPS指令集。MIPS的含义是武内锁流水线微处理器（Microprocessor without Interlocked Piped Stages），是上世纪80年代诞生的RISC CPU的重要代表。

MIPS指令集的指令大体可以分成以下几种格式：

①R型指令：

具体操作由op、func结合指定，rs，rt是源寄存器的编号，rd是目的寄存器的编号。MIPS32架构中有32个通用寄存器，使用5位编码就可以全部表示，所以rs，rt，rd的宽度都是5位。Shamt只在移位指令中使用，用来指定移位位数。

31-26	25-21	20-16	15-11	10-6	5-0
OP	RS	RT	RD	SHMAT	FUNCT

②I型指令：

具体操作由op指定，指令的低16位是立即数，运算时要将其拓展至32位，然后作为其中一个源操作数参与运算。

31-26	25-21	20-16	15-0
OP	RS	RT	IMM

③J类型指令：

具体操作由op指定，一般是跳转指令，低26位是字地址，用于产生跳转的目标地址。

31-26	25-0
OP	Address

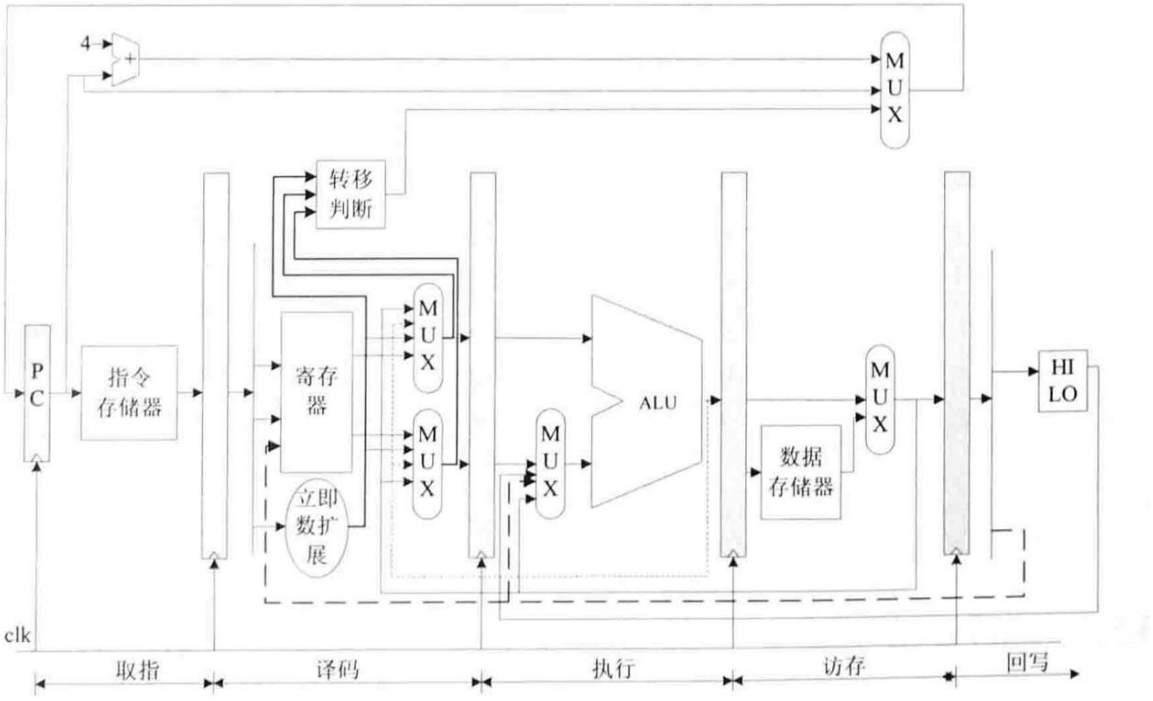
其中，各个字段的具体作用为：

名称	作用
OP	操作码，用以判定输入的是什么指令，然后cpu再进行对应的操作
RS	为第一个源操作数寄存器
RT	为第二个源操作数寄存器，或目的操作数寄存器
RD	用于存放操作结果的目的寄存器

SHMAT	位移量，移位指令(如SLL，SRL等)都根据此字段确定左移或者右移多少位。
FUNCT	功能码。当指令为R型指令时用以确定具体属于哪一条R型指令。
IMM	立即数。I型指令中由指令直接输入的立即数。

3. 总体结构设计：

（1）CPU数据通路图：

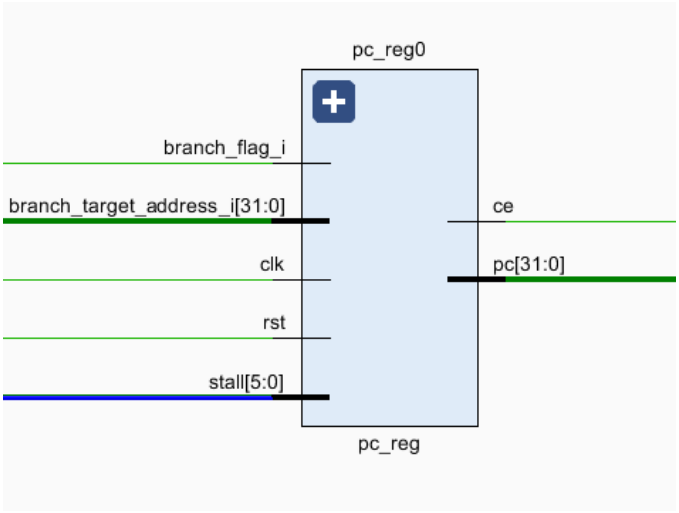


（2）各个模块的说明与详细作用：

根据取指、译码、执行、访存、回写五个阶段对本次实验中实现的CPU各个子模块的详细说明：

1）取指阶段：

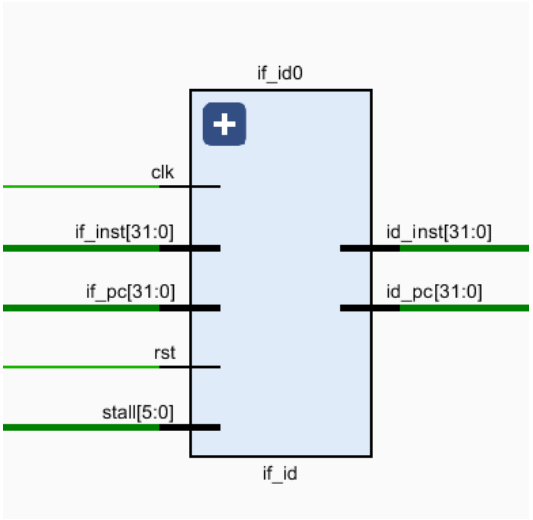
①pc_reg模块：Pc_reg模块的作用是给出指令地址，当指令存储器使能的时候，pc的值每时钟周期加4.



输入与输出信号的说明：

接口名	作用
Clk	时钟信号
Rst	复位信号
Pc	要读取的指令的地址
Ce	指令存储器使能信号
Branch_target_address_i	转移到的目标地址
Branch_flag_i	是否发生转移
Stall	阻塞信号

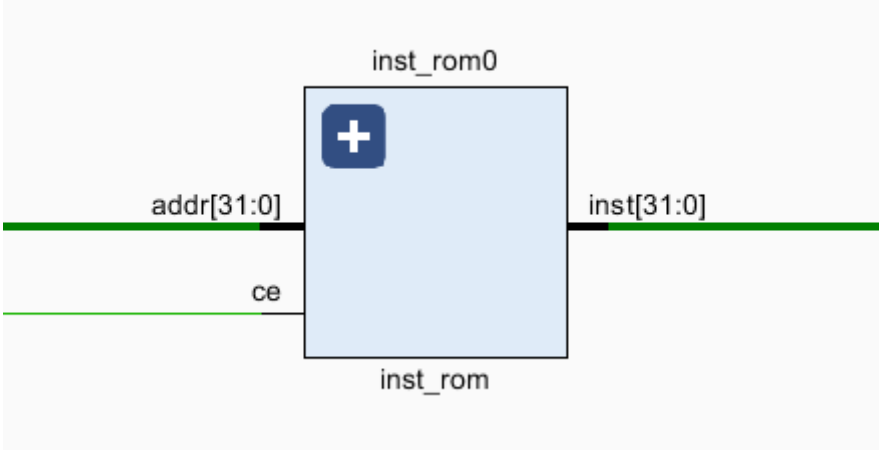
②IF/ID模块：实现取指与译码阶段之间的寄存器，将取值阶段的结果（取得的指令、指令地址等信息）在下一个时钟传递到译码阶段。



输入与输出信号的说明：

接口名	作用
Rst	复位信号
Clk	时钟信号
If_pc	取指阶段取得的指令对应的地址
If_inst	取值阶段取得的指令
Id_pc	译码阶段的指令对应的地址
Id_inst	译码阶段的指令
Stall	阻塞信号

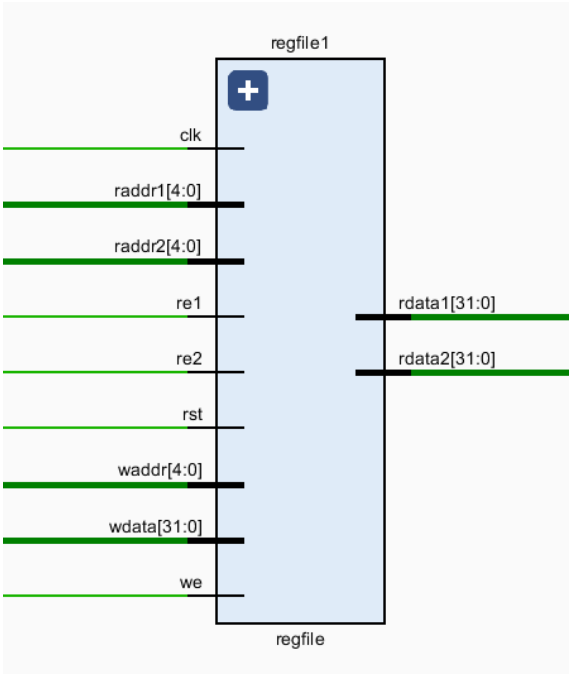
③inst_rom模块：也即指令存储器模块，该模块作用较为简单，即读取对应的txt文本中16进制的指令的机器码并以inst传输给译码模块。



端口名称	作用
Addr	当前指令的pc值
Ce	指令存储器使能信号
Inst	读取到的指令

2）译码阶段：

①regfile模块：实现了32个32位通用整数寄存器，可以同时进行两个寄存器的读操作和一个寄存器的写操作。



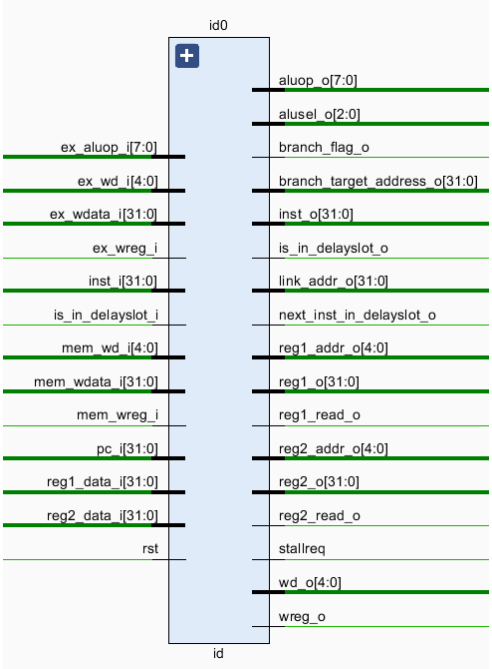
输入输出端口说明如下：

接口名	作用
Rst	复位信号，高电平有效
Clk	时钟信号
Waddr	要写入的寄存器地址
Wdata	要写入的数据
We	写使能信号
Addr1	第一个读寄存器端口要读取的寄存器的地址
Re1	第一个读寄存器端口读使能信号
Rdata1	第一个读寄存器端口输出的寄存器值
Raddr2	第二个读寄存器端口要读取的寄存器的地址
Re2	第二个读寄存器端口读使能信号
Rdata2	第二个读寄存器端口输出的寄存器值

在regfile模块内部定义了一个二维的向量，元数个数是32，每个元素的宽度也是32，由此定义了32个32位的寄存器堆。同时又通过读取读使能信号以及写使能信号来判断是否进行对应的寄存器读与寄存器写。

②ID模块：ID模块的作用是对指令进行译码，得到最终运算的类型、子类型、源操作数1、

源操作数2、要写入的目的寄存器地址等信息。

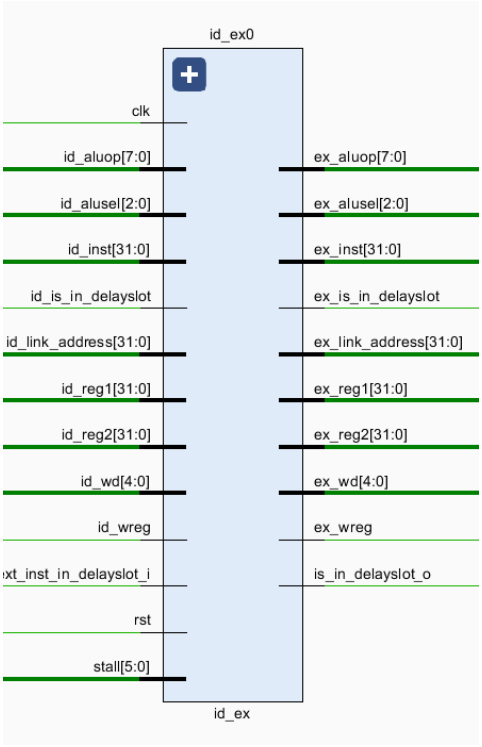


部分输入及输出端口的说明：

端口名字	作用
Rst	复位信号
Pc_i	译码阶段的指令对应的地址
Inst_i	译码阶段的指令
Reg1_data_i	从regfile输入的第一个读寄存器端口的输入
Reg2_data_i	从regfile输入的第二个读寄存器端口的输入
Reg1_read_o	Regfile模块的第一个读寄存器端口的读使能信号
Reg2_read_o	Regfile模块的第二个读寄存器端口的读使能信号
Reg1_addr_o	Regfile模块的第一个读寄存器端口的读地址信号
Reg2_addr_o	Regfile模块的第二个读寄存器端口的读地址信号
Alu_op	译码阶段的指令要进行的运算的子类型
Alusel_o	译码阶段的指令要进行的运算的类型

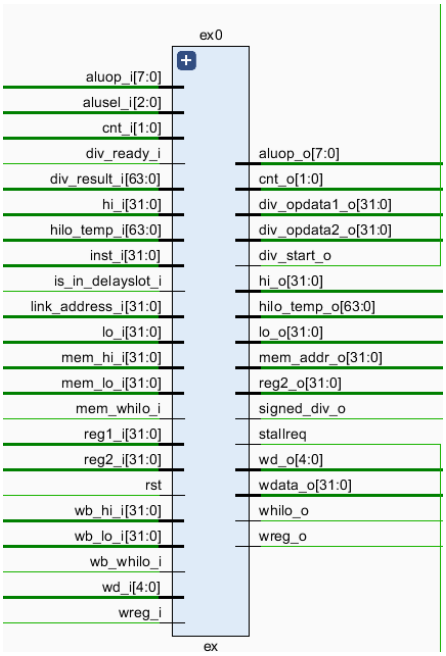
Reg1_o	源操作数1
Reg2_o	源操作数2
Wd_0	要写入的目的寄存器地址
Wreg_0	是否要写入目的寄存器
Branch_target_address_i	转移到的目标地址
Branch_flag_i	是否发生转移

③ID/EX模块：ID模块的输出连接到ID/EX模块，该模块的作用是将译码阶段取得的运算类型、源操作数、要写的目的寄存器等结果，在下一个时钟周期传递到流水线阶段。



3）执行阶段：

①EX模块：根据译码阶段的结果，进行指定的运算，给出运算结果。EX模块会从ID/EX模块得到运算类型与运算子类型、源操作数1与源操作数2、要写的目的寄存器地址等。然后EX模块会根据这些数据进行运算，最后再进行输出。

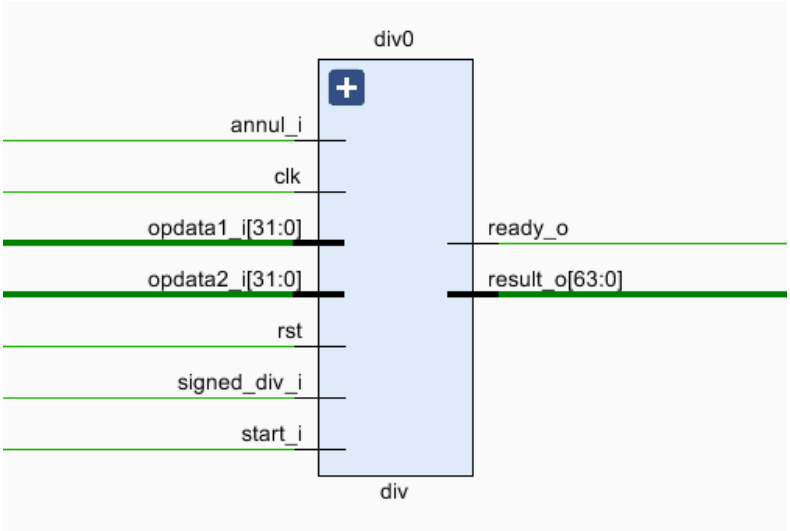


部分输入输出接口的说明：

接口名称	作用
Rst	复位信号
Alusel_i	执行阶段要进行的运算的类型
Aluop_i	执行阶段要进行的运算的子类型
Reg1_i	参与运算的源操作数1
Reg2_i	参与运算的源操作数2
Wd_i	指令执行要写入的目的寄存器
Wreg_i	是否有要写入的目的寄存器
Wd_o	执行阶段最终要写入的目的寄存器的地址
Wreg_o	执行阶段的指令最终是否有要写入的目的寄存器
Wdata_o	执行阶段的指令最终要写入目的寄存器的值
Hi_i	HI寄存器的值
Lo_i	LO寄存器的值
Mem_whilo_i	访存时是否要写HI、LO寄存器
Mem_hi_i	处于访存阶段的指令要写入HI寄存器的值

Mem_lo_i	处于访存阶段的指令要写入LO寄存器的值
Wb_whilo_i	回写阶段的指令是否要写HI、LO寄存器
Wb_hi_i	处于回写阶段的指令要写入HI寄存器的值
Wb_lo_i	处于回写阶段的指令要写入LO寄存器的值
Hi_o	执行阶段的指令要写入HI寄存器的值
Lo_o	执行阶段的指令要写入LO寄存器的值
Whilo_o	执行阶段的指令是否要写HI、LO寄存器
Hilo_temp_i	第一个执行周期得到的乘法结果
Cnt_i	当前处于执行阶段的第几个时钟周期
Hilo_temp_o	第一个执行周期得到的乘法结果
Cnt_o	下一个时钟周期处于执行阶段的第几个时钟周期

②div除法模块：div模块为专门用来进行除法运算的模块，其中实现采用试商法的32位除法运算。当流水线执行阶段的ex模块发现当前指令是除法指令时，首先暂停流水线，然后将被除数、除数等信息送到div模块，开始除法运算，当div模块运算完成后再将运算结果运送到ex模块。

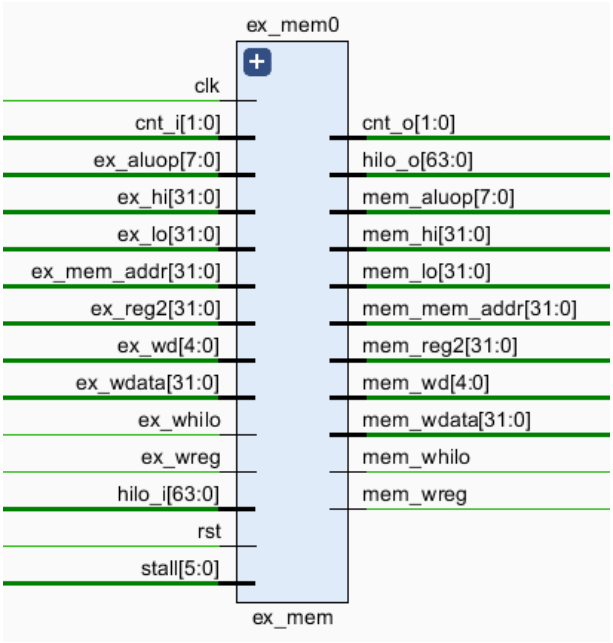


输入输出接口的描述：

接口名称	作用
Rst	复位信号

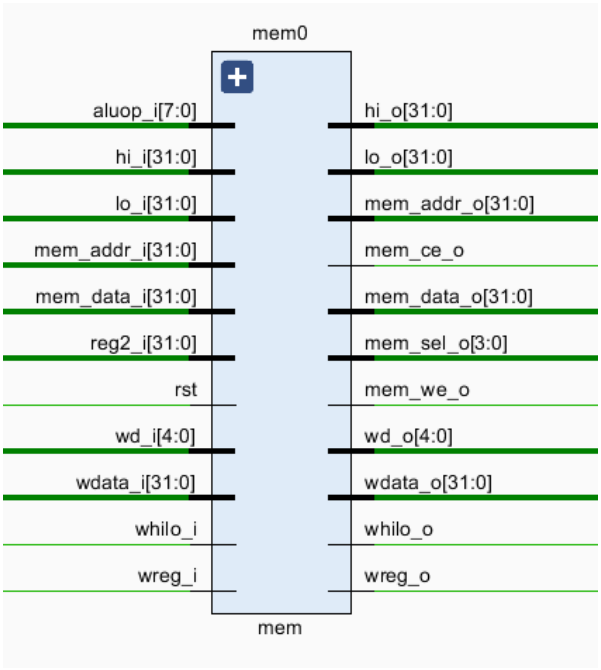
Clk	时钟信号
Signed_div_i	是否有符号除法
Opdata1_i	被除数
Opdata2_i	除数
Start	是否开始除法运算
Annul_i	是否取消除法运算
Result_o	除法运算结果
Reday_o	除法运算是否结束

③EX/MEM模块：EX模块的输出连接到EX/MEM模块，该模块的作用是将执行阶段取得的运算结果在下一个时钟传递到流水线访存阶段。

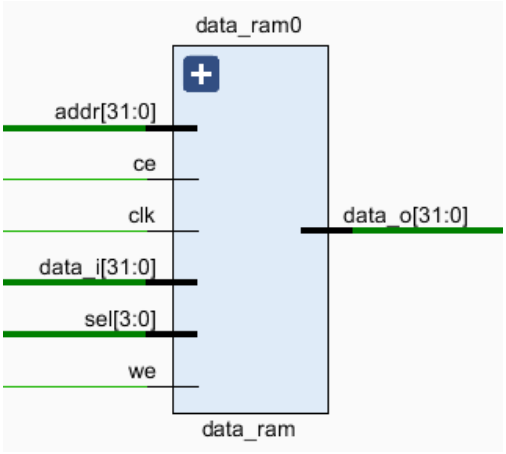


4）访存阶段

①MEM模块：如果输入的指令是加载、存储指令，那么会对数据存储器进行访问。

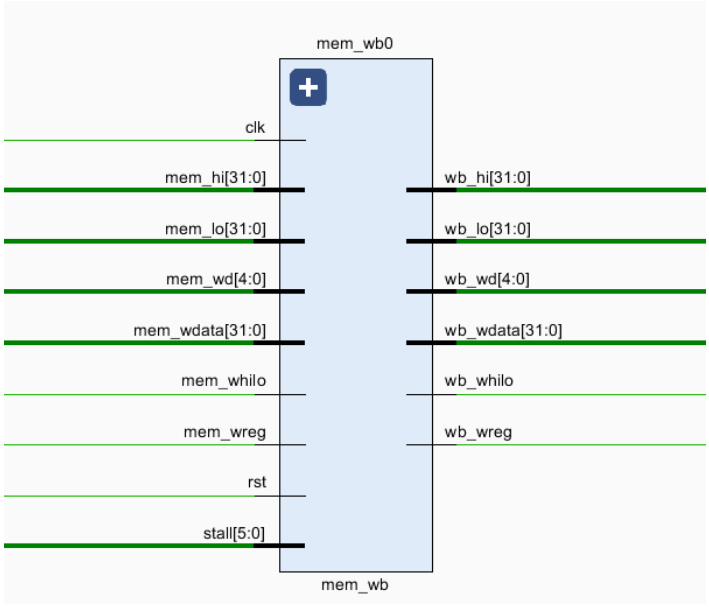


②data_ram模块：也即数据存储器模块。



为了实现对数据存储器按字节寻址，在设计的时候使用了4个8位存储器代替一个32位存储器，读操作时，从4个8位存储器中各读出一个字节，组合为一个32为数据输出，写操作时，依据sel的值，修改其中特定存储器对应的字节即可。

③MEM/WB模块：实现访存与回写阶段之间的寄存器，将访存阶段的结果在下一个时钟周期传递到回写阶段。



5）回写阶段

经过上面模块的传递，指令的运算结果会直接进入回写阶段，这个阶段实际上实在regfile模块中实现的，MEM/WB模块的输出wb_wreg、wb_wd、wb_wdata连接到Regfile模块，分别连接到写使能端口we、写操作目的寄存器端口waddr、写入数据端口wdata，所以会将指令的运算结果写入目的寄存器。

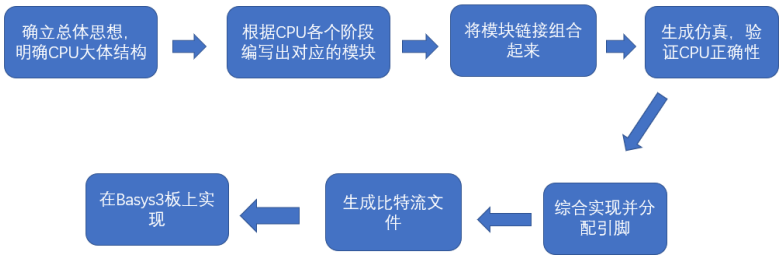
四. 实验器材

电脑一台，Xilinx Vivado 软件一套，Basys3板一块。

五. 实验过程与结果

1.CPU设计思想与方法：

（1）CPU设计的总体思想：模块化编程。



①总的来说，就是首先根据在理论课上已经学过的知识，确定流水线CPU的五个部分（取指令（IF）、译码（ID）、执行（EX）、访存（MEM）阶段、写回（WB）），并且清晰明确每个部分的具体作用。

②根据每个部分的具体作用编写对应的模块。

比如对于取指令IF阶段，需要有一个模块对文件进行取指并输出，由此编写出指令

寄存器Ins_rom模块与PC_reg模块；而对于译码ID阶段，则需要有一个部分对由指令寄存器输出的指令进行解析，知晓是什么指令，然后再根据这个指令输出对应的信号，以控制后面的部分的操作，由此编写出ID控制模块，根据输入指令的op码与funct码确定具体的控制信号，并输出到各个部分中控制cpu的运行；又比如说对于执行EX阶段，则需要根据前面的信号对输入的数据进行具体的处理，由此编写出EX模块，用以根据控制信号与指令类别对输入的指令进行具体操作并输出运算结果。

③将各个模块链接起来。

根据流水线cpu的五个阶段编写好对应的模块以后，这些模块是相对独立的，如果需要编写出完整的cpu，这个时候就需要将这些模块像“搭积木”一样拼接起来，将各个模块的输入与输出对应，链接成一整个完整的cpu。

而与单周期CPU所不同的是，流水线CPU的各个阶段的模块之间存在流水线寄存器。在流水线处理器中，一条指令被拆分成若干小步骤和数据来进行处理，每个流水线阶段都会具有流水线寄存器来存储数据。流水线寄存器通常用于存储中间结果，以便在流水线处理的过程中将其传递到下一个阶段。所以考虑到流水线寄存器的存在，与单周期CPU相比，流水线CPU的“模块链接”部分则更为复杂。

④最后根据需要对cpu进行优化。

完成最基础的cpu设计以后，可以再根据需要对cpu进行改进与完善，比如说要增加指令，则在ID模块中增加对应的指令，在EX模块中增加相对应的计算；又比如说要解决流水线cpu的数据冒险现象，则也要在相应的模块上增加转发、判断条件等，将流水线cpu进行不断完善。

⑤对CPU进行仿真验证。

为了验证CPU设计的正确性，需要建立仿真文件并对CPU进行仿真检验，测试已经添加的指令以保证设计无误，为后续在basys3板上的实现作铺垫。

⑥最后分配引脚，综合实现并生成比特流文件然后下载到basys3板上

(2）部分重要模块的代码与原理说明：

①inst_rom模块代码说明：

```
12      reg[`InstBus]  inst_mem[0:`InstMemNum-1];
13
14      initial $readmemh("E:/Xilinx Project/inst_rom.txt",inst_mem);
15
16      always @ (*) begin
17          if (ce == `ChipDisable) begin
18              inst <= `ZeroWord;
19        end else begin
20            inst <= inst_mem[addr[`InstMemNumLog2+1:2]];
21          end
22      end
23
24  endmodule
```

在初始化指令存储器时，使用了$readmemh，表示从ins_rom文件中读取数据以初始化inst_mem，而inst_mem正是上一行定义的数组。Ins_rom.txt文件是使用mars软件，在其中输入想要运行的指令然后生成的16进制指令机器码。文件中每行存储了一条32位宽度、用16进制表示的指令，系统函数$readmemh会将inst_rom.txt中的数据依次填写到inst_mem数组中。

②在本实验中，数据冒险的解决方式：

对于R型指令：

1）相隔两条指令存在数据冒险（即流水线译码、回写阶段存在数据相关）这种情况在regfile中得到了解决：

```
end else if((raddr1 == waddr) && (we == `WriteEnable)
                && (re1 == `ReadEnable)) begin
    rdata1 <= wdata;
end else if((raddr2 == waddr) && (we == `WriteEnable)
                && (re2 == `ReadEnable)) begin
    rdata2 <= wdata;
```

在读操作中有一个判断，如果要读取的寄存器是在下一个时钟周期上升沿要写入的寄存器，那么就将要写入的数据直接作为结果输出。如此就解决了像个两条指令存在数据相关的情况。

2）对于相邻的或者只相隔一条指令的指令间存在的数据冒险，采用了数据前推的方式用以解决，即将计算结果从其产生处直接送到其他指令需要处或所有需要的功能单元处，由此避免数据冒险现象。为了实现这一效果，需要将处于流水线执行阶段的指令的运算结果以及处于流水线访存阶段的指令的运算结果等信息送到译码ID模块。在译码阶段的ID模块中会依据需要送入的信息进行，综合判断，解决数据冒险现象，给出最后要参与运算的操作数。

```
always @ (*) begin
        stallreq_for_reg1_loadrelate <= `NoStop;
    if(rst == `RstEnable) begin
        reg1_o <= `ZeroWord;
    end else if(pre_inst_is_load == 1'b1 && ex_wd_i == reg1_addr_o
                        && reg1_read_o == 1'b1 ) begin
      stallreq_for_reg1_loadrelate <= `Stop;
    end else if((reg1_read_o == 1'b1) && (ex_wreg_i == 1'b1)
                        && (ex_wd_i == reg1_addr_o)) begin
        reg1_o <= ex_wdata_i;
    end else if((reg1_read_o == 1'b1) && (mem_wreg_i == 1'b1)
                        && (mem_wd_i == reg1_addr_o)) begin
        reg1_o <= mem_wdata_i;
  end else if(reg1_read_o == 1'b1) begin
    reg1_o <= reg1_data_i;
  end else if(reg1_read_o == 1'b0) begin
    reg1_o <= imm;
  end else begin
    reg1_o <= `ZeroWord;
  end
end
```

给reg1_o赋值的过程增加了两种情况：如果regfile模块读端口1要读取的寄存器就是执行阶段要写的寄存器，那么直接把执行阶段的结果ex_wdata_i作为reg1_0的值；如果regfile模块读端口1要读取的寄存器就是访存阶段要写的寄存器，那么直接把访存阶段的结果mem_wdata_i作为reg1_0的值.

给reg2_o赋值的过程也同理。

对于R型指令数据冒险解决效果的测试：

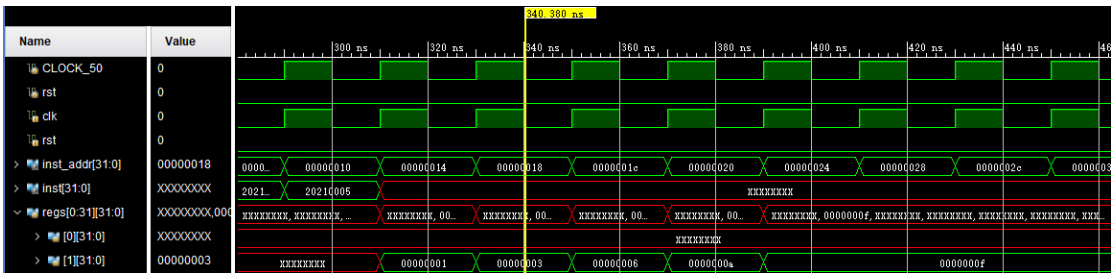
测试的指令为：

test

```
1  addi  $1,$0,1
2  addi  $1,$1,2
3  addi  $1,$1,3
4  addi  $1,$1,4
5  addi  $1,$1,5
```

各条指令间都存在数据依赖。

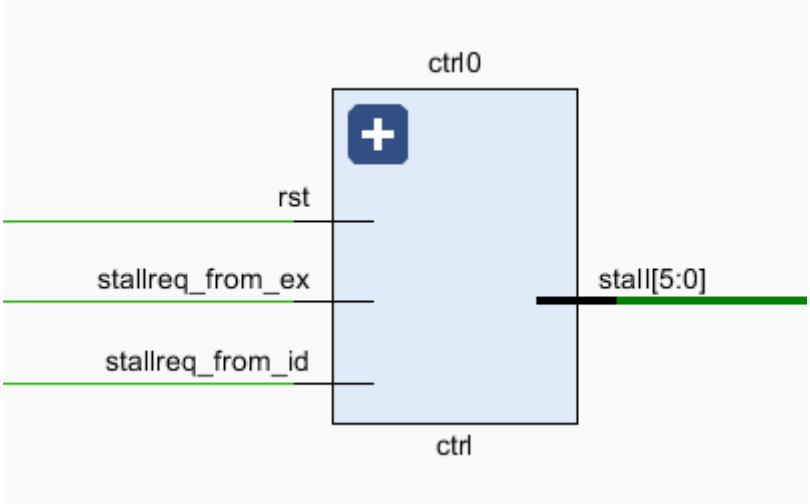
测试结果为



可以从结果的波形图中看到$1中的值符合变化规律，即能在不阻塞流水线时钟周期的情况下成功计算出正确的结果。

③流水线暂停机制的实现：

本实验流水线的暂停机制通过CTRL来实现



接口名称	作用
Rst	复位信号
Stallreq_from_id	处于译码阶段的指令是否请求流水线暂停
Stall_from_ex	处于执行阶段的指令是否请求流水线暂停
Stall	暂停流水线控制信号

```
always @ (*) begin
    if(rst == `RstEnable) begin
        stall <= 6'b000000;
    end else if(stallreq_from_ex == `Stop) begin
        stall <= 6'b001111;
    end else if(stallreq_from_id == `Stop) begin
        stall <= 6'b000111;
    end else begin
        stall <= 6'b000000;
    end    //if
end      //always
```

当流水线处于执行阶段的指令请求暂停时，要求取指、译码、执行阶段暂停，而访存、回写阶段继续。设置stall为6’b001111；当当流水线处于译码阶段的指令请求暂停时，要求取指、译码阶段暂停，而执行、访存、回写阶段继续。设置stall为6’ b001111；其余情况不暂停流水线，设置stall为6’b000000.

④除法的实现

本实验设计的CPU为了实现除法指令专门设计了div除法模块，用以计算除法。Div模块使用的除法指令实现思路是试商法，试商法原理较为简单易懂（在老师的课上也着重讲过），但是缺点是速度较慢，对于32位的除法，至少需要32个时钟周期才能得到除法结果。

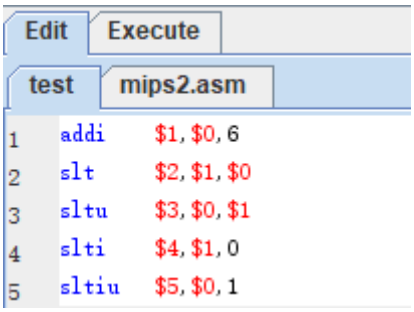
大体思想是设被除数是m，除数是n，商保存在s中，被除数的位数是k，首先取出被除数的最高位，使用被除数的最高位减去除数n，如果结果大于0则商的s[k]位1，反之为0.如果上一步得出的结果是0，表示当前的被减数小于除数，则取出被除数剩下的值的最高位，与当前被减数组合作为下一轮的被减数；如果上一步得出的结果是1，表示当前的被减数大于除数，则利用上一步中减法的结果与被除数剩下的值的最高位组合为下一轮的被减数，然后令k-1；新的被减数减去除数，如果结果大于等于0，则商的s[k]为1，否则s[k]为0，后面的步骤重复前面两步，直到k=1.

（3）验证设计CPU的正确性：

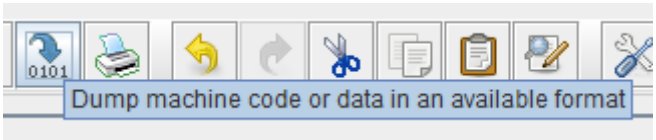
为了验证设计CPU的正确性，在本实验报告中将对实现的每一条指令进行相应的仿真测试，并通过仿真测试的波形图来检验指令是否正常执行。

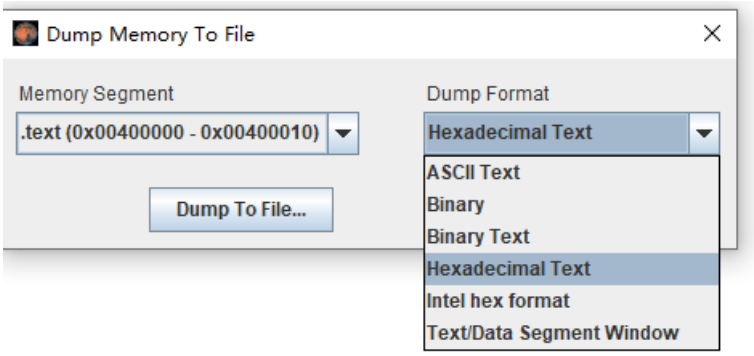
由于要进行测试的指令数量较多，所以采用分类测试，按照指令的类别来进行分组测试，以检验CPU的正确性。

具体的测试方法为，利用在实验三汇编实验中使用过的Mars软件，输入想要进行测试的指令。

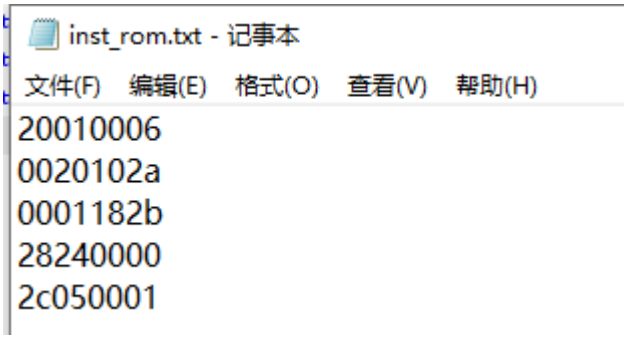


然后点击运行后再点击生成对应的十六进制机器码到对应的文件上。





此处生成的存放十六进制机器码的记事本文件就是再cpu中inst_rom模块中读取的文件，cpu要执行的指令从其中读取。



最后再进行仿真，然后查看波形图即可。

1）测试类别：算术指令

①第一组测试的指令程序为

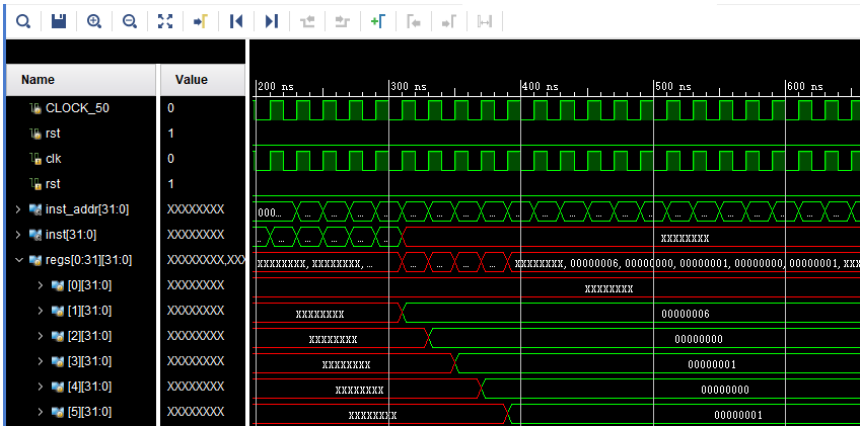
addi $1,$0,6

slt $2,$1,$0

sltu $3,$0,$1

slti $4,$1,0

sltiu $5,$0,1



该测试段总共测试了addi,slt,sltu,slti,sltiu五条指令。

可以看到，第一条指令为addi指令，为立即数加，效果是将寄存器中的值与立即数相加并存放到对应的寄存器中。可以看到0寄存器中的值默认为0，与立即数6相加后存放到了寄存器1当中，可以在波形图中看到1寄存器的值变为了6，结果正确；第二条指令是slt小于则设置指令，比较两个寄存器中存放的值的大小，如果1寄存器的值小于0寄存器的值，则将2寄存器设置为1，否则设置为0.可以知道1寄存器的值为6，比0大，所以可以看到波形图中2寄存器的值为0；sltu为小于无符号数则设置，如果0寄存器比1寄存器小则将3寄存器设置为1，由前面的结果可以知道0号寄存器中存放的值是0，1号寄存器中存放的值是6，0比6小所以3号寄存器的值为1，由波形图知结果正确。

Slti为立即数小于则设置，将rs寄存器中存放到的值与立即数进行比较，如果小于则将rt寄存器中的数设置为1，否则设置为0.sltiu为小于立即数（无符号数）则设置，与slti的区别在于立即数为无符号数。可以从知道6不小于0，而0小于1，可以看到波形图中寄存器4的值为0，寄存器5的值为1.

②第二组测试的指令程序为：

addi $1,$0,6

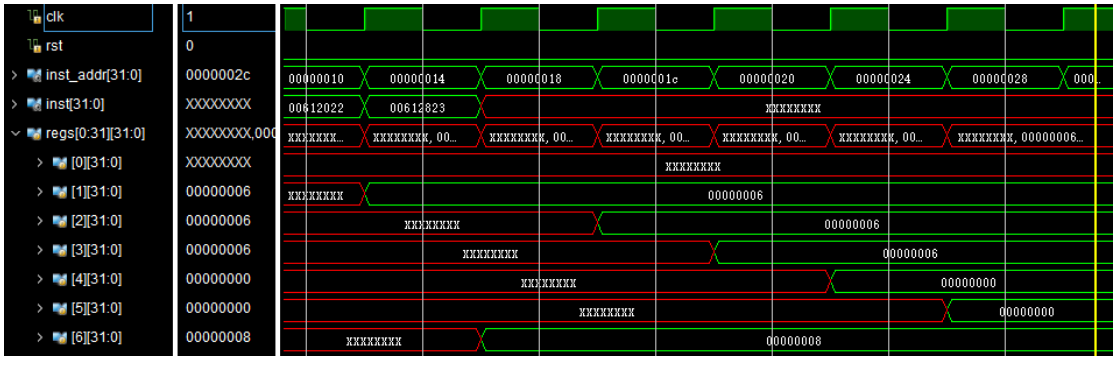
addiu $6,$1,2

add $2,$1,$0

addu$3,$2,$0

sub $4,$3,$1

subu$5,$3,$1



该测试段总共测试了addiu,add,addu,sub,subu五条指令。

第一条addi指令在上一程序测试段已经测试过了，不再赘述。

第二条为addiu指令，为无符号立即数加，addiu $6,$1,2的作用是将寄存器1中的值与立即数2相加再存放到寄存器6中，可以知道此时寄存器1中的值为6，6+2=8，可以在波形图中

看到寄存器6的值为8，结果正确。

第三条第四条指令为add,addu指令，执行的指令分别为add $2,$1,$0，addu $3,$2,$0。即将寄存器1中的值与寄存器0中的值相加并存放在寄存器2中、将寄存器2中的值与寄存器0中的值相加存放到寄存器3中。可以在波形图中看到寄存器2和寄存器3的值依次变为了6.

第五和第六条指令为sub,subu指令，执行的指令分别为sub $4,$3,$1，subu $5,$3,$1。即将寄存器1中的值与寄存器3中的值相减并存放在寄存器4中、将寄存器1中的值与寄存器3中的值相减并存放在寄存器5中.可以在波形图中看到寄存器4和寄存器5的值依次变为了0.

③第三组测试的指令程序为：

```
addi    $1,$0,2
addi    $2,$0,4
multu   $1,$0
mult    $1,$2
mul     $3,$1,$2
madd    $2,$3
maddu   $3,$2
msub    $3,$2
msubu   $2,$3
div     $5,$3,$1
divu    $6,$3,$2
```

divu $6,$3,$2该组指令测试了mult、multu、mul、madd、maddu、msub、msubu、div、divu等九条指令。

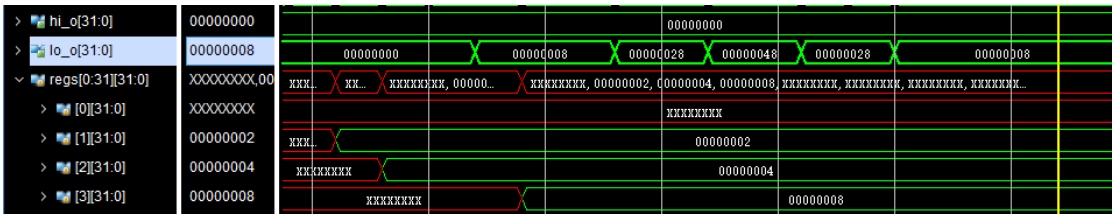
第三条指令mult为乘法指令，第四条指令multu为无符号乘法指令，是将rs寄存器中存放的值以及rt寄存器中存放的值相乘，乘积的低位字和高位字分别存入寄存器lo和hi中。

第四条指令将寄存器1中的值与寄存器0中的值0相乘，可以看到高位寄存器和低位寄存器的值都变为了0，结果正确。可以看到执行第四条指令时先将寄存器1中的值赋值为了2，寄存器2中的值赋值为了4，二者相乘的结果为8，执行之后低位寄存器的值为8，高位寄存器的值为0.结果正确。

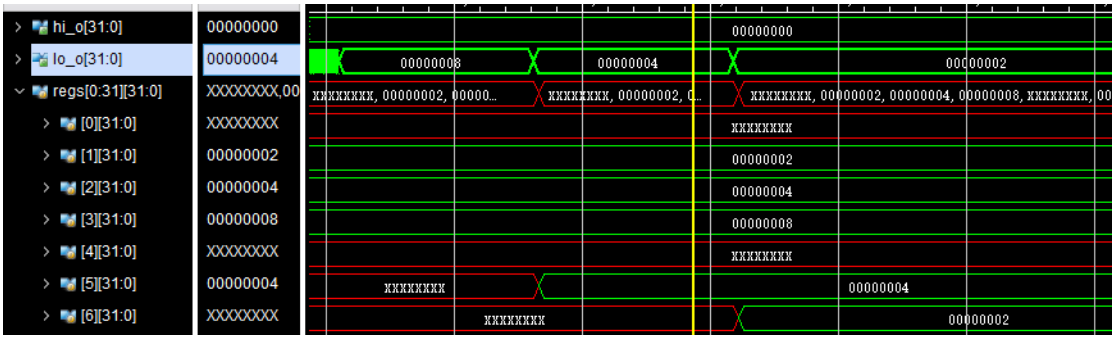
第五条指令mul为不带溢出位的乘法，是将rs寄存器和rt寄存器的乘积结果的低32位存入寄存器rd。可以看到执行后寄存器3的值变为了8，结果正确。

第六第七条指令madd、maddu为乘加与无符号乘加，是将寄存器rs和rt的乘积所得的64位结果与连接寄存器lo、hi中的64位相加。执行madd　$2,$3，寄存器2中的值为2，寄存器3中的值为8 ，二者相乘得到的16进制数为0x20，再与lo、hi中存放的值相加，得到28，结果正确。紧接着又执行maddu　$2,$3，即再加一次0x20，此时可以看到lo寄存器中存放的值变为了48，结果正确。

第八第九条指令执行了msub　$3,$2，msubu $2,$3，即连续减去两次0x20，可以看到此时lo中的值先从48变为了28，再变回了8，结果正确。



第十第十一条指令是验证除法的指令，执行了div　$5,$3,$1和divu　$6,$3,$2 两条指令，这两条指令的作用是将rs寄存器中的值除以rt寄存器中的值，并且将运算结果存储到rd寄存器当中。可以知道当执行这两条指令时，寄存器3、寄存器2和寄存器1的值分别为8、4和2.两条指令的计算结果分别为4和2，从波形图中可以看到寄存器5的值变为了4，寄存器6的值变为了2，而用以存储乘除法结果的高低位寄存器也有了对应的变化，可以得知运算结果正确。



最后为了再一次更精确地验证64位乘法指令正确性，选择进行有溢出的乘法，观察高位寄存器是否存储对应的数值。

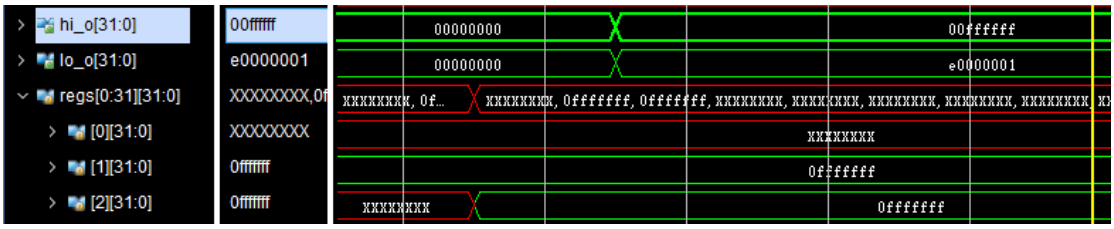
addi　　$1,$0,268435455

addi　　$2,$0,268435455

mult　　$1,$2

此时，1寄存器和2寄存器中储存的值都为0x0FFFFFFF，两者相乘后的结果超过了32位，可

以看到hi寄存器中此时的值变为了0x00FFFFFF，低位寄存器的值为0xE0000001，结果正确。



2）测试类别：逻辑指令

ori $1,$0,7

andi $2,$1,3

xori $3,$1,3

lui $4,65535

and $5,$1,$2

or $6,$1,$0

xor $7,$1,$2

nor $8,$1,$2

本组共测试了ori，andi，xori，lui，and，or，xor，nor共八条指令。

第一条指令为ori，立即数或运算，作用是将rs寄存器中的值与立即数相或，运算结果存储到rd寄存器中。ori $1,$0,7，此时寄存器0中的值为0，与7相或运算结果为7，可以看到寄存器1中的值变为了7，运算正确。

第二条指令为andi指令，立即数与运算，作用是将rs寄存器中的值与立即数相与，运算结果存储到rd寄存器中。andi $2,$1,3，此时寄存器1中的值为7，与3相与运算结果为3，可以看到寄存器2中的值变为了3，运算正确。

第三条指令为xori指令，也即立即数异或指令，相同的位置为0，不相同的位置为1. xori $3,$1,3，可以看到寄存器1中的值为7，与立即数3只有第三位不相同（7为1，3为0）所以运算结果为4，由波形图可知运算正确。

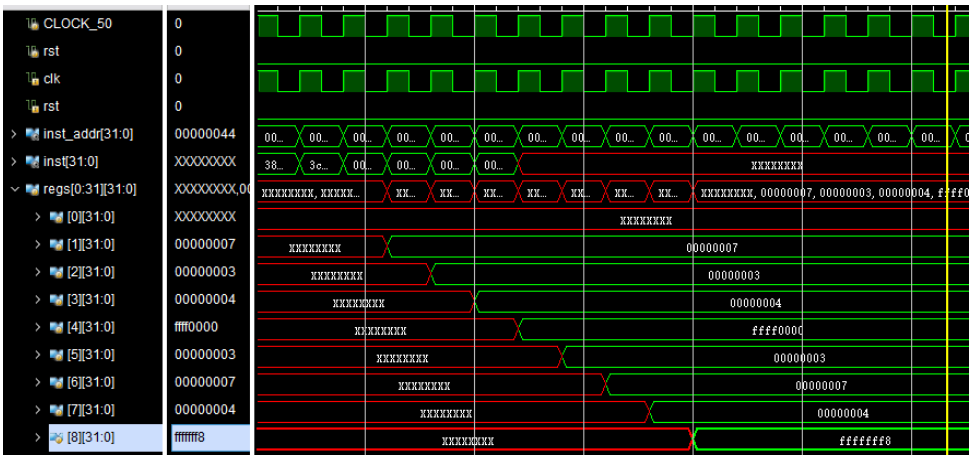
第四条指令为lui指令，为立即数高位取指令，将立即数的低半字位存入寄存器rt的高半字地址，并将寄存器的低位值置为0. lui $4,65535即将0xffff置于寄存器4的高16中，可以看到波形图中寄存器4的结果变为了0xffff0000，运算结果正确。

第五条指令为寄存器与指令，and $5,$1,$2是将寄存器1中的值与寄存器2中的值相与然后存放到寄存器5中。可以在波形图中看到寄存器5中的值变为了3，运算结果正确。

第六条指令是寄存器或指令，or　　$6,$1,$0将寄存器1中的值和寄存器0中的值相与然后存放到寄存器6中。可以看到寄存器6中的值变为了7，运算结果正确。

第七条指令为寄存器异或指令，xor　　$7,$1,$2将寄存器1和寄存器2中的值相异或再存放到寄存器7中，由前面的指令可知运算结果应该为4，在波形图中可以看到寄存器7的值变为了7，运算结果正确。

第八条指令为或非指令，进行按位或再取反的操作，nor　$8,$1,$2 的结果应该为0xfffffff8，由波形图结果可以知道结果运算正确。



3）移位指令

addi $1,$0,1

sll　$2,$1,2

sllv　$3,$2,$1

srl　$4,$3,1

srlv　$5,$4,$1

addi $6,$0,-16

sra　$7,$6,2

srav $8,$6,$1

本组指令程序共测试了sll，sllv，srl，srlv，sra和srav共六条指令

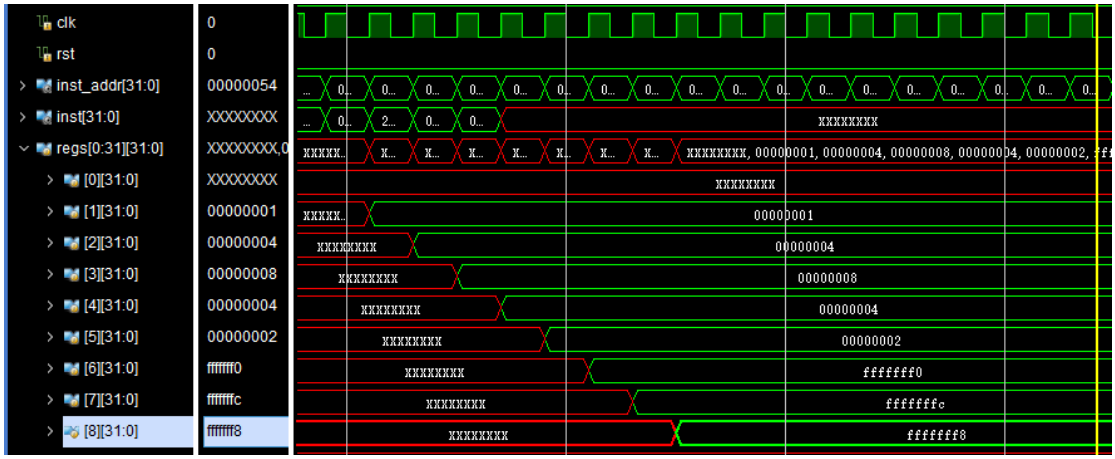
可以看到第二条指令为sll指令，作用是将寄存器1中存放的值向左移动两位，然后再存放到寄存器2中，可知此时寄存器1中的值为1，移动两位后为4，可以从波形图中清楚地看到寄存器2的值变为了4，运算结果正确。

第三条指令为sllv指令，为寄存器左移指令，sllv　$3,$2,$1的作用是将寄存器2中的值向左移动寄存器1中的值个位数，运算结果存储在寄存器3中，寄存器2中的值为4，寄存器1中的值为1，移动后的值为8，可以看到寄存器3中的值变为8，结果正确。

第四条指令为srl指令，为逻辑右移指令，srl　$4,$3,1的作用是将寄存器3中的值向左移动一位，运算结果存储在寄存器4中，寄存器3中的值为8，向右移动后的值为4，可以看到寄存器4中的值变为4，结果正确。

第五条指令为srlv指令，为寄存器右移指令，srlv　$5,$4,$1的作用是将寄存器4中的值向右移动寄存器1中的值个位数，运算结果存储在寄存器5中，寄存器4中的值为4，寄存器1中的值为1，移动后的值为2，可以看到寄存器5中的值变为2，结果正确。

第七条为算术右移指令，第八条为寄存器算术右移指令，与逻辑右移不同的是当向右移补最高位的时候，逻辑右移默认补0，而算术右移会根据被移动的数的最高位来决定补0还是补1. sra　$7,$6,2，srav $8,$6,$1。寄存器6中存放的值为-16，最高位为1，所以两条指令的运算结果最高位都补1，由图中波形图结果可以知道运算正确。



4）数据传输指令

addi　$1,$0,2

addi　$2,$0,4

mthi　$1

mtlo　$2

mfhi　$3

mflo　$4

movz　$5,$1,$0

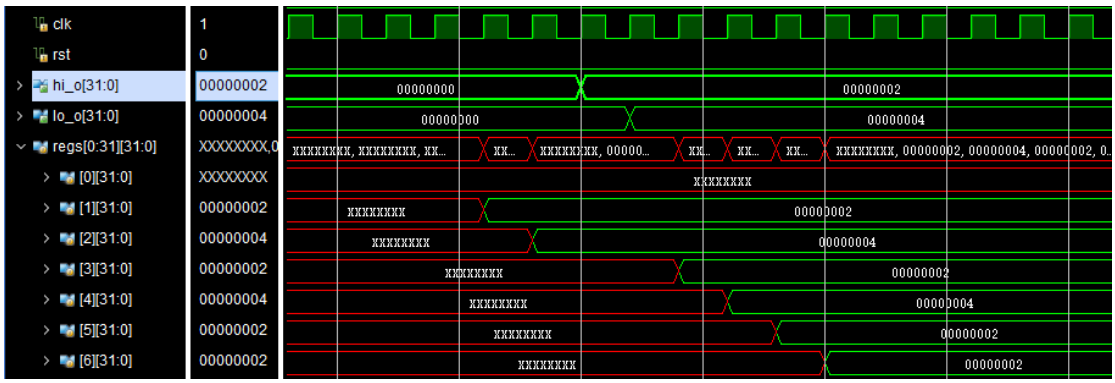
movn $6,$1,$1

本组指令程序共测试了mthi，mtlo，mfhi，mflo以及movz、movn六条指令

第三条指令是mthi，mthi $1的运算结果是将寄存器1中的值传送至hi寄存器。第四条指令是mtlo $2，运算结果是将寄存器2中的值传送至lo寄存器。由第一二条指令可以知道此时寄存器1和寄存器2中存放的值分别为2和4，由波形图可以看到执行指令之后高位寄存器的值变为了2，地位寄存器的值变为了4，运算结果正确。

第五第六条指令分别为mfhi和mflo，运算结果分别是将高位和低位寄存器中存放的值移动到寄存器3和寄存器4中，由波形图结果可知运算结果正确。

第七条指令为movz，零条件传送。如果寄存器rt的值为0，将寄存器rs中的数值传送到寄存器rd中，第八条指令为movn，非零条件传送。如果寄存器rt的值不为0，将寄存器rs中的数值传送到寄存器rd中。可以看到第七条指令中0寄存器为0，第八条指令中1寄存器不为0，波形图中5寄存器和6寄存器相继变为1寄存器中存放的值2，结果正确。



5）转移指令

mark0:

addi $1,$0,4

mark1:

addi $1,$1,-1

bne $1,$0,mark1

addi $2,$0,-2

mark2:

addi $2,$2,1

bltz $2,mark2

addi $3,$0,-1

mark3:

addi $3,$3,1

beq $3,$0,mark3

j mark0

本组测试指令程序测试了4条指令，分别为j指令、bne指令、beq指令以及bltz指令。

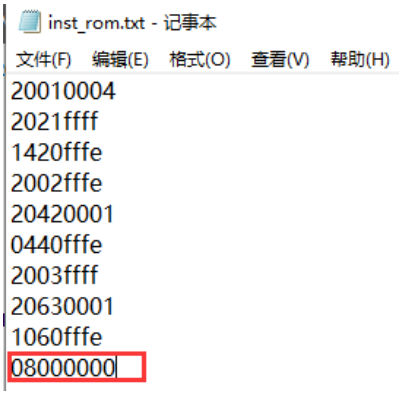
首先是mark1，bne$1,$0,mark1表示当寄存器1的值和寄存器0的值不相等的时候则相对跳转到mark1，而每次跳转后都会addi $1,$1,-1令寄存器1中的值相加，直到寄存器1中的值和寄存器0中的值相等，则不再跳转。可以从波形图中看到寄存器1中的值从4循环到了0后再下一轮大循环前不再变化。

然后是mark2，bltz $2,mark2表示当寄存器2中的值小于0时则相对跳转到mark2。紧接着执行addi $2,$2,1令2中的值加一。寄存器2中的值初始为2，当寄存器2中的值加为0时不满足小于0的跳转条件，不进行跳转，进入到mark3的阶段。

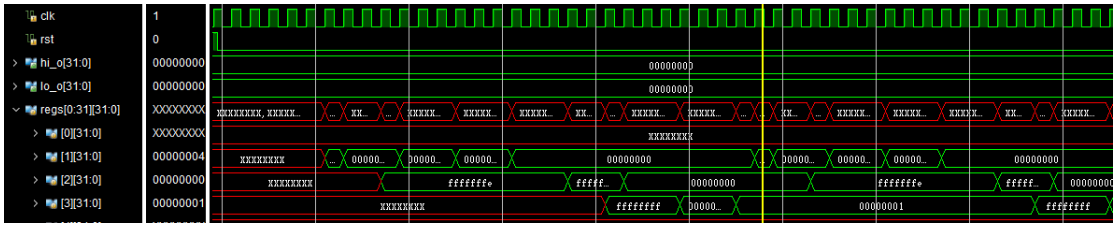
到了mark3后，寄存器3中初始值为-1，beq $3,$0,mark3意为当寄存器3中的值与寄存器0中的值相等时则跳转，可以看到寄存器3中初始为-1，加一后变为了0，第一次执行满足条件跳转到mark3，第二次执行时因为又执行了一次addi $3,$3,1所以不满足跳转条件，不进行跳转。顺序往下执行。

顺序执行到j后，j mark0意为无条件绝对跳转到mark0处的地址，所以可以看到波形图中后面一直在重复循环前面的几个小循环。因为mark0处在程序开头，每当程序运行到j指令时都会再次跳转到地址为00000000的位置，重复执行。

需要注意的是，因为mars软件中默认的地址不是从0开始的，所以想要在vivado中对j指令进行验证，需要手动修改生成的ins文件，将j指令跳转的地址修改为0.



将绝对跳转的地址修改为0.



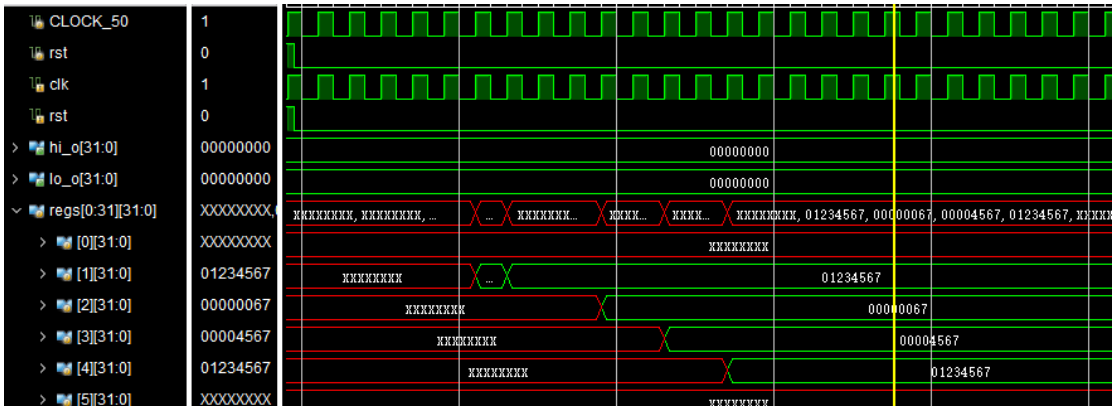
6）保存与读取指令

```
addi    $1,$0,19088743
sb  $1,0($0)
lb  $2,0($0)
sh  $1,4($0)
lh  $3,4($0)
sw  $1,8($0)
lw  $4,8($0)
```

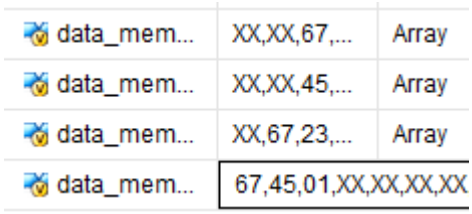
本组实验的指令程序总共测试了六组指令，分别是sb，sh，sw和lb，lh，lw。

本组指令的的第2至第3条指令分别将寄存器1（此时存放的值为0x01234567）中的低位字节（也即0x67）存放到编号为0的地址中，然后再用lb指令取出存放到寄存器2中。第四第五条指令先用sh指令将寄存器1（此时存放的值为0x01234567）中的低位半字（也即0x4567）存放到编号为4的地址中，然后再用lh指令取出存放到寄存器2中。最后再用sw指令将寄存器1中存放的值（0x01234567）存放到编号为8的地址中，然后再用lw指令取出数值存放到寄存器4当中。

由波形图可知，寄存器2、3、4的值相继变成了0x67、0x4567以及0x1234567，可知运算结果正确。



Data_rom中data_mem存放的数值如图所示。



（4）实现：烧写到basys3板子上

1.指令执行采用单步（按键控制）执行方式，由开关（SW15、SW14）控制选择查看数码管上的相关信息，地址和数据。地址或数据的输出经以下模块代码转换后接到数码管上。

（1）首先编写数码管译码模块。译码模块将 CPU 运算的结果转换成 7 段数码管中各个数码管显示所需的高低电平信号,该单元的输入为 4-bit 位宽的二进制数。

```
case(Store)
    4'b0000:    Out=    8'b00000011;    //0
    4'b0001:    Out=    8'b10011111;    //1
    4'b0010:    Out=    8'b00100101;    //2
    4'b0011:    Out=    8'b00001101;    //3
    4'b0100:    Out=    8'b10011001;    //4
    4'b0101:    Out=    8'b01001001;    //5
    4'b0110:    Out=    8'b01000001;    //6
    4'b0111:    Out=    8'b00011111;    //7
    4'b1000:    Out=    8'b00000001;    //8
    4'b1001:    Out=    8'b00001001;    //9
    4'b1010:    Out=    8'b00010001;    //A
    4'b1011:    Out=    8'b11000001;    //b
    4'b1100:    Out=    8'b01100011;    //C
    4'b1101:    Out=    8'b10000101;    //d
    4'b1110:    Out=    8'b01100001;    //E
    4'b1111:    Out=    8'b01110001;    //F
    default:    Out=    8'b00000000;    //all light
endcase
```

（2）其次是按键消抖模块。Basys3 板采用的是机械按键，在按下按键时按键会出现人眼无法观测但是系统会检测到的抖动变化，这可能会使短时间内电平频繁变化，导致程序接收到许多错误的触发信号而出现许多不可知的错误。消抖操作是每当检测到 CLK 上升沿到来时检测一次当前电平信号并记录，同计数器开始计数，若在计数器达到 5000 之前电平发

生变化，则将计数器清零，若达到 5000，则将该记录电平取反输出。

（3）最后是顶层模块。

主要执行以下操作：

①对Basys3板系统时钟信号进行分频，分频的目的用于计数器；

②生成计数器，计数器用于产生4个数。这4数用于控制4个数码管；

③根据计数器产生的数生成数码管相应的位控信号（输出）和接收CPU来的相应数据；

④将从CPU 接收到的相应数据转换为数码管显示信号，再送往数码管显示（输出）。

2.代码编写完成后添加约束文件或者在I/O Ports界面分配引脚：

Project Summary | cpu.xdc

E:/Xilinx Project/PipelineCPU_2/PipelineCPU_2.srcs/constrs_1/new/cpu.xdc

```
set_property IOSTANDARD LVCMOS33 [get_ports CLK]
set_property IOSTANDARD LVCMOS33 [get_ports Reset]

set_property PACKAGE_PIN U2 [get_ports {AN[0]}]
set_property PACKAGE_PIN U4 [get_ports {AN[1]}]
set_property PACKAGE_PIN V4 [get_ports {AN[2]}]
set_property PACKAGE_PIN W4 [get_ports {AN[3]}]

set_property PACKAGE_PIN V7 [get_ports {Out[0]}]
set_property PACKAGE_PIN U7 [get_ports {Out[1]}]
set_property PACKAGE_PIN V5 [get_ports {Out[2]}]
set_property PACKAGE_PIN U5 [get_ports {Out[3]}]
set_property PACKAGE_PIN V8 [get_ports {Out[4]}]
set_property PACKAGE_PIN U8 [get_ports {Out[5]}]
set_property PACKAGE_PIN W6 [get_ports {Out[6]}]
set_property PACKAGE_PIN W7 [get_ports {Out[7]}]

set_property PACKAGE_PIN U1 [get_ports {SW[1]}]
set_property PACKAGE_PIN W2 [get_ports {SW[0]}]

set_property PACKAGE_PIN R2 [get_ports {Button}]
set_property PACKAGE_PIN W5 [get_ports {CLK}]
set_property PACKAGE_PIN T1 [get_ports {Reset}]

```

3.分配引脚完成后点击综合：

SYNTHESIS

Run Synthesis

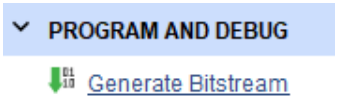
Open Synthesized Design

并进行实现

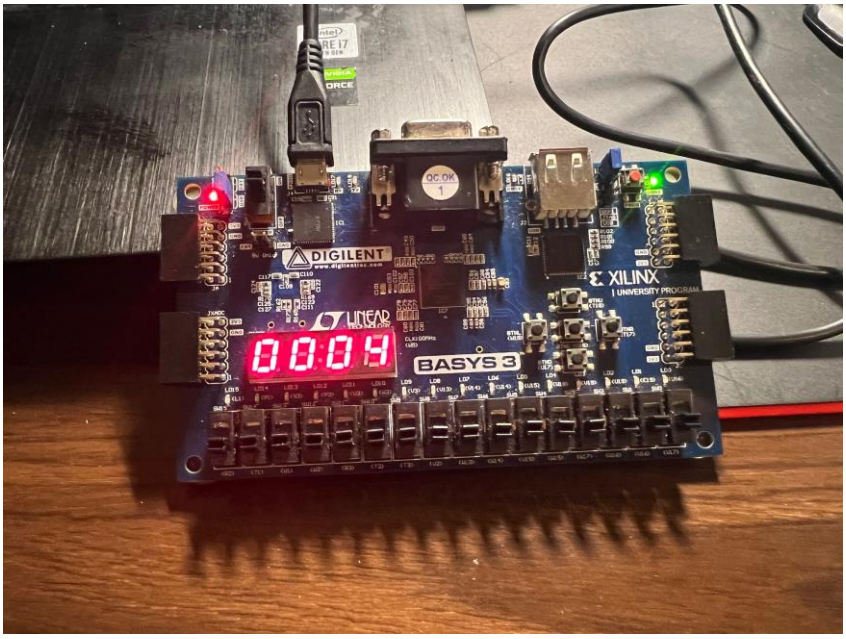
IMPLEMENTATION

Run Implementation

最后生成比特流文件



然后将比特流文件下载到basys3板上，再在板上运行cpu。



3.板上实现：

展示指令：

```
addiu   $1,$0,8
ori   $2,$0,2
add   $3,$2,$1
sub   $5,$3,$2
and   $4,$5,$2
```

①addiu $1,$0,8

第一条指令地址为0，nextpc=4

运算结果为0+8=8

执行后rt寄存器1的值为8

并且rs寄存器0的值为0

②ori　$2,$0,2

第二条指令地址为4，nextpc=8

此时rt寄存器$0的值为0

Rs寄存器$2的值为0

指令的运行结果为2

③add　$3,$2,$1

此时第三条指令的地址为8，nextpc=c

Rs寄存器$2的值为2

Rt寄存器1的值为8

执行结果应为2+8=0xa

此时运算结果为

结果正确

④sub　$5,$3,$2

第四条指令地址为c，nextpc=0x10

Rs寄存器$3的值为0xa

Rt寄存器$2的值为2

运算结果为0xa-2=8

可知结果正确。

⑤and　$4,$5,$2

第五条指令地址为0x10，nextpc=0x14

此时rs寄存器$5的值为8

Rt寄存器$2的值为2

1000和0010两者相与结果为0

可知运算结果正确

六．实验心得

体会和建议：

经过了整个学期的实验课程的学习，从简单的加法器的实现开始，到认识MIPS汇编程序的编写，再到cpu各个单独模块的编写与实验，然后再到独立编写出一个属于自己的单周期CPU，再到最后成功实现出一个能实现多条mips指令的流水线cpu，对于我来说无疑是巨大的挑战，在其中我认识到的最重要的一个道理就是要学会自学、学会自己去探索知识。在上计组实验课之前，我是完全没有接触过Verilog语言，对于vivado的使用更是一窍不通。而由于我是转专业学生，在这个学期之前没有经过系统性的计算机课程的学习，相对于其他同学，我的基础更为薄弱。与此同时，因为没有上过大一的数电课程（安排在大二下学期才补修），计组实验中涉及到的数电知识（如与非门、或非门等），我完全是一窍不通，这更是加大了在这门实验课程中我需要面临的挑战，从最开始的一张白纸一窍不通，到最后能成功

编写出一个完整的流水线CPU，我经历了重重困难，并在这个过程中锻炼了我的能力。

正所谓“师傅领进门，修行靠个人”，大学阶段的学习与初高中阶段的学习有很大差别，初高中阶段是老师手把手教你要做什么，怎么做，作为学生只需要听从老师的安排即可；而到了大学，许多知识和方法则需要自己主动去查阅、探索，老师只是起一个引导作用。在编写流水线cpu的过程中，极大地锻炼了我自学的能力，比如verilog语言的基础语法是什么？有什么需要注意的事项？又比如说对于vivado的使用，像如何建立模块、编写模块到如何使用ip核；怎样进行仿真、调试；怎样进行综合实现又如何连接引脚；以及最后如何下载比特流文件到板子上如何使用板子等。涉及的知识老师不可能面面俱到跟你一一讲清楚，这个时候就需要自己去自学，要学会自己弄清楚并解决问题。

与此同时，计组实验课也锻炼了我查阅资料的能力，懂得了上网搜索遇到的问题，许多网站比如知乎、CSDN等都有较为详细的解答。有时候自己摸索一天的问题，只需要自己查阅资料，在网上搜索一下就能迎刃而解。比如在综合实现过程中，我曾遇到生成的比特流文件无法下载到板子上的问题，对于这个问题我自己捣鼓了很久都没有解决，最后复制message中的错误信息到网上搜索，才知晓原来是文件选择的板子型号与实际使用的板子型号不对应，修改之后问题很快得到解决。

最后，经历编写cpu，因为是基于理论课的基础，在锻炼了我动手实践能力的同时，也反复温习了在理论课上学过的知识。对于流水线cpu的相关理论知识，比如流水线cpu分为哪几个部分，每个部分又有什么作用，mips指令集中指令的结构有哪些，又有什么特点等等。经历过这次试验以后我都了然于胸。在实验的同时又夯实了理论基础，我感到受益匪浅，这可能将是使我受益一生的知识。