计算机组成原理课程设计

题　目　　五级流水线处理器设计

学生姓名	陈明富
学　　号	161710228
学　　院	计算机科学与技术学院
专　　业	计算机科学与技术
班　　级	1617102
指导教师	施慧彬

二〇一九年七月

南京航空航天大学
实验报告诚信承诺书

本人郑重声明：所呈交的毕业设计（论文）（题目：带冒险流水线CP设计 36条　　）是本人在导师的指导下独立进行研究所取得的成果。尽本人所知，除了毕业设计（论文）中特别加以标注引用的内容外，本毕业设计（论文）不包含任何其他个人或集体已经发表或撰写的成果作品。

作者签名：陈明富　2019 年 7 月 7日
（学号）：161710228

流水线 MIPS CPU课程设计报告

摘　要

为了提升对单周期 MIPS CPU 的理解能力和动手能力，充分地理解和掌握计算机组成原理课程的相关知识。深入理解计算机内部指令的执行原理和实现方式，设计简单的单周期CPU，计算机指令执行过程中控制信号和数据通路部件的联系，基本部件如存储器、组合逻辑元件和控制器的实现和处理方式。通过处理数据冒险、控制冒险解决流水线的冲突问题，设计出符合硬件逻辑要求的仿真。

关键词：流水线，MIPS，数据冒险，控制

Pipeline MIPS CPU Course Design Report

Abstract

In order to improve the understanding and hands-on ability of the single-cycle MIPS CPU, fully understand and master the knowledge of the computer composition principle course. In-depth understanding of the execution principle and implementation of computer internal instructions, design of a simple single-cycle CPU, the connection of control signals and data path components in the execution of computer instructions, basic components such as memory, combinatorial logic components and controller implementation and processing. By dealing with data adventures, controlling risk-solving pipeline conflicts, and designing simulations that meet hardware logic requirements.

Key Words：pipeline ；MIP; data hazard; control

目 录

1. 课程设计说明........7
  1.1 流水线设计说明........7
2. 模块化和层次化设计说明........8
  2.1 设计图........8
  2.2 模块化设计图........8
  2.3 层次化设计........9
3. 具体模块化定义........10
  3.1 PC 模块........10
  3.2 NPC 模块........10
  3.3 IM_4k 模块........10
  3.4 ALU 模块........11
  3.5 MUX 模块........11
  3.6 RegFile 模块........12
  3.7 dm_4k 模块........12
  3.8 Ctrl 模块........13
  3.9 Extend 模块........14
  3.10 IFID 寄存器........14
  3.11 IDEX 寄存器........14
  3.12 EXMEM 寄存器........16
  3.13 MEMWB 寄存器........17
  3.14 MIPS 模块........18
  3.15 Forward 模块........18
  3.16 load_use 模块........18
  3.17 controlhazard 模块........18
4. 测试代码及结果........18
  4.1 具体 Verilog 代码实现........18
  4.2 ModelSim 模拟及测试........38
  4.3 测试仿真结果........39
5. 心得体会........42
参 考 文 献........43
附 录........43

1. 课程设计说明

1.1 流水线设计说明

处理了一条指令的执行过程被分成五个阶段，每个阶段由相应的功能部件完成。如果将各阶段看成相应的流水段，则指令的执行过程就构成了一条指令流水线。

（1） Ifetch 阶段:将PC的值作为地址到指令存储器IM中取指令，并计算PC+4，送到PC输入端。

（2） Reg/Dec 阶段：根据指令中的Rs和Rt的值到寄存器堆中取出相应寄存器的值，同时对指令中的操作码op字段进行译码，生成相应的控制信号。寄存器堆可看成是寄存器读口和寄存器写口两个功能部件。

（3） Exec 阶段：由具体指令确定，不同的指令经过ID段译码后得到不同的控制信号，用来控制执行部件进行不同的操作。
（4） Mem 阶段：由具体指令确定，有Branch和MemWr两个控制信号。
（5） Wr阶段: 由具体指令确定，将数据写到寄存器中。

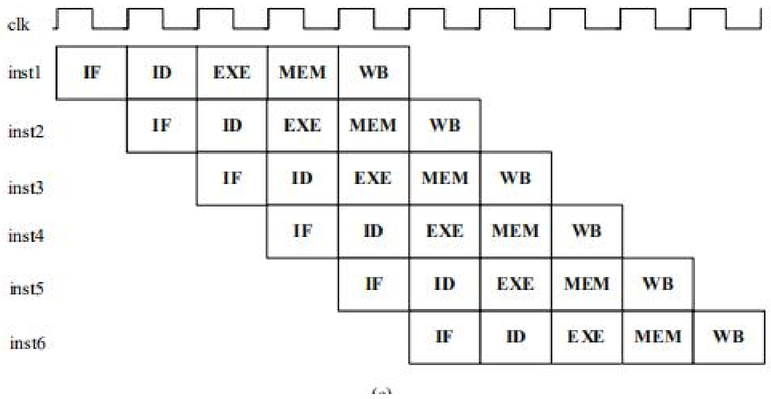


图1.1 5级流水线CPU时空图

2. 模块化和层次化设计说明

2.1 设计图

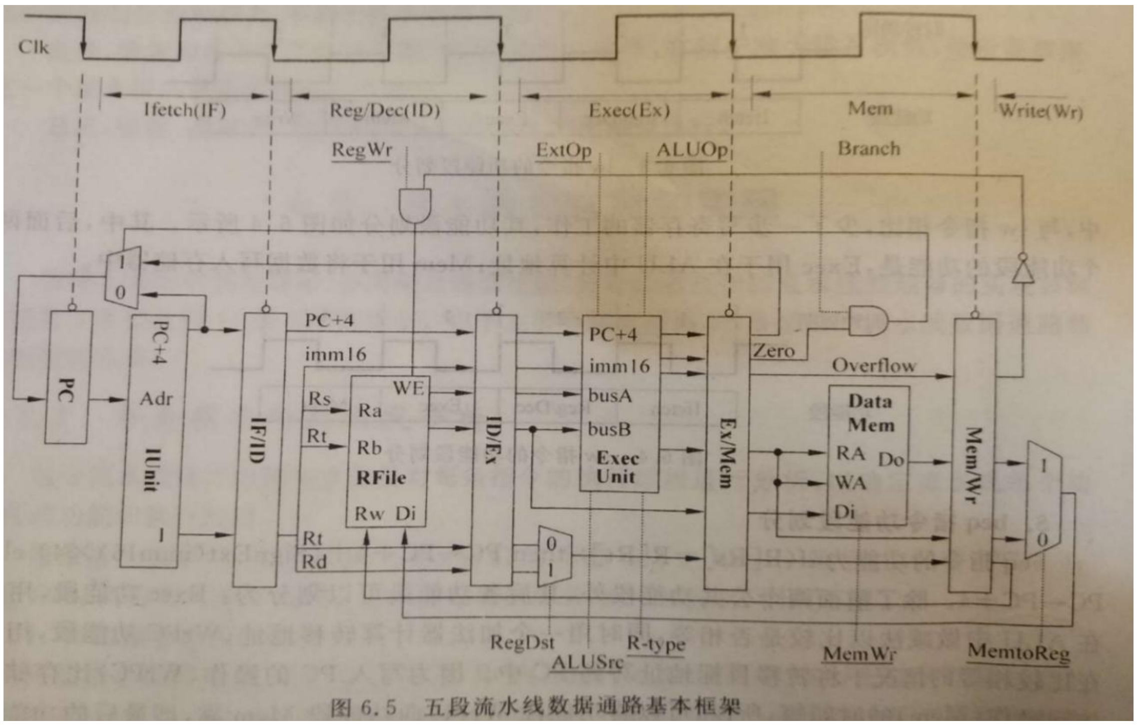


图1.1 5级流水线数据通路基本框架

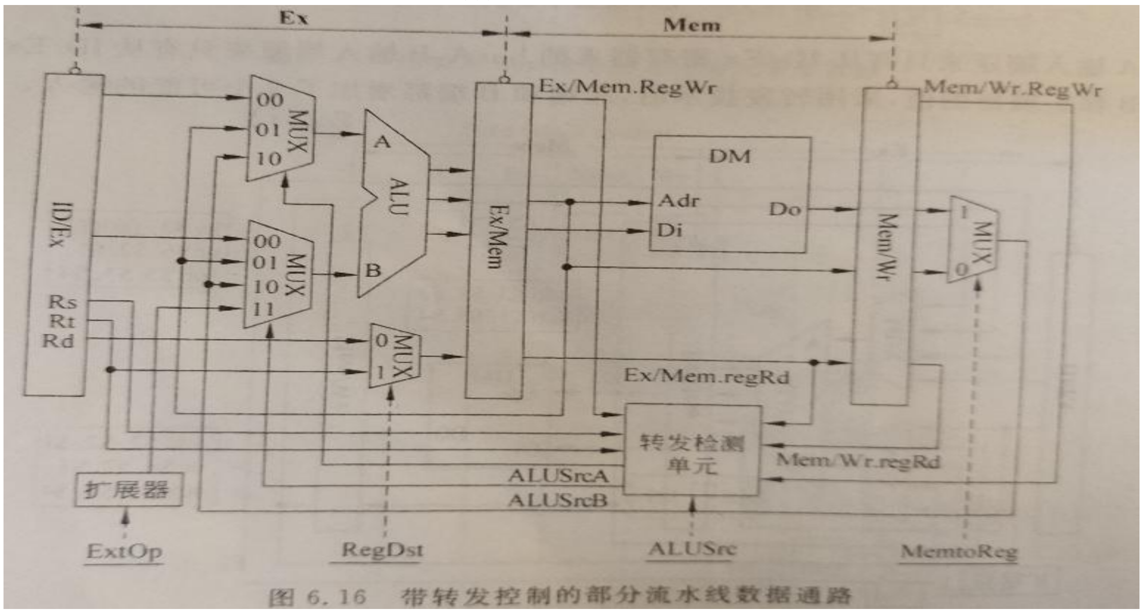


图1.2 带转发的流水线部分数据通路

2.2 模块化设计图

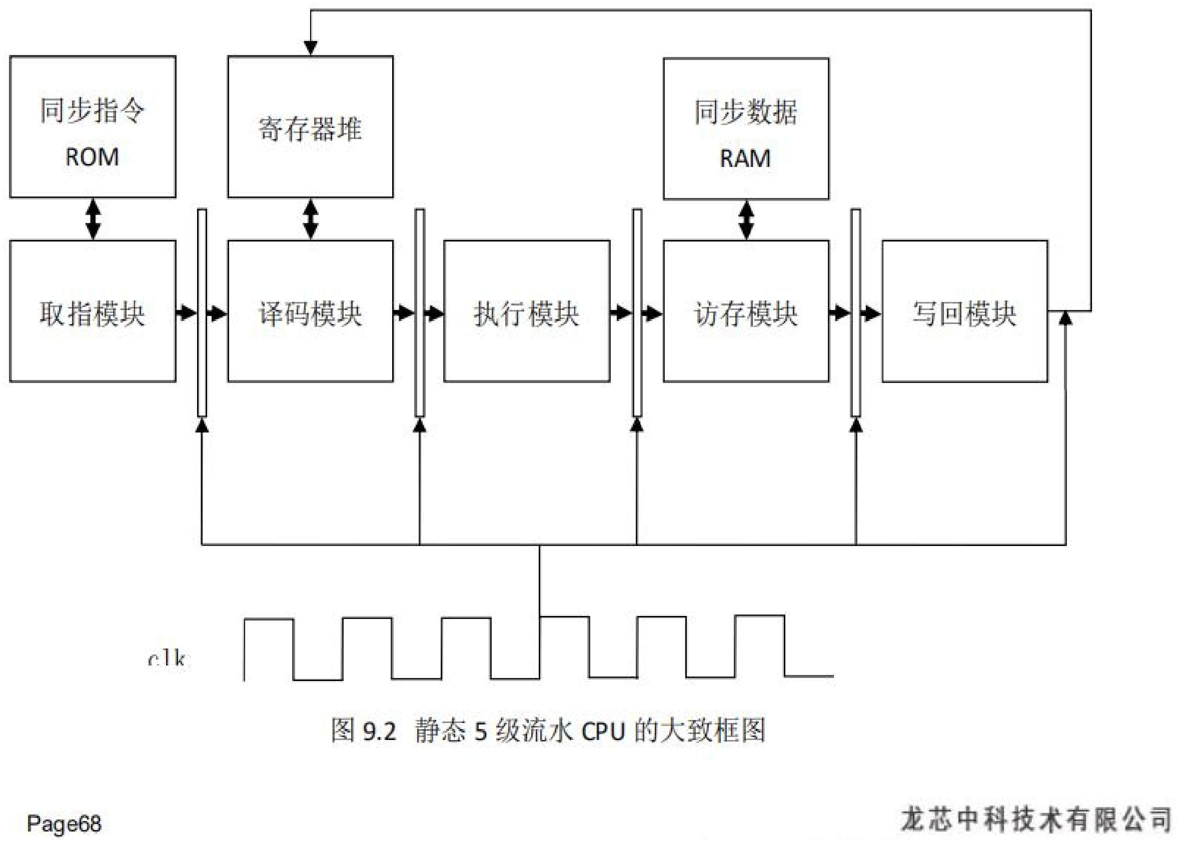
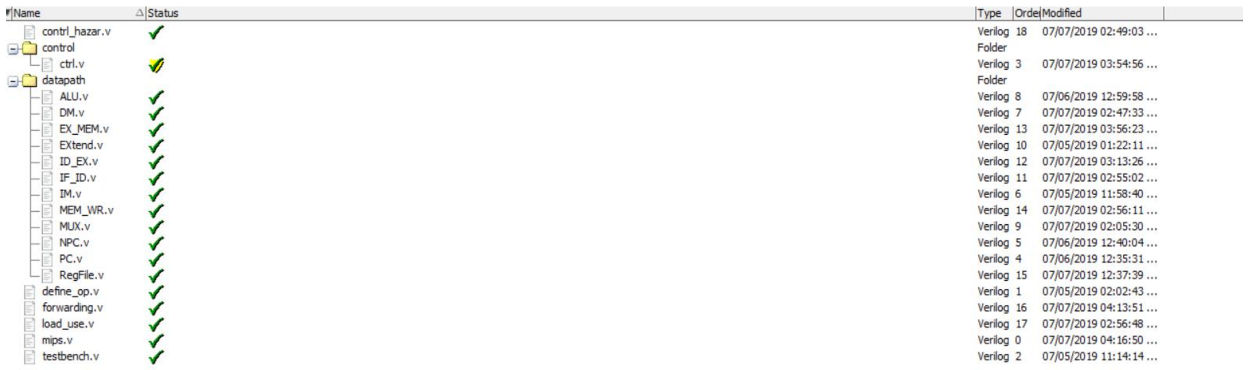


图1.3 5级流水线CPU框图

2.3 层次化设计



Name	Status	Type	Order	Modified
contrl_hazar.v	✓	Verilog	18	07/07/2019 02:49:03 ...
control		Folder		
ctrl.v	✓	Verilog	3	07/07/2019 03:54:56 ...
datapath		Folder		
ALU.v	✓	Verilog	8	07/06/2019 12:59:58 ...
DM.v	✓	Verilog	7	07/07/2019 02:47:33 ...
EX_MEM.v	✓	Verilog	13	07/07/2019 03:56:23 ...
EXtend.v	✓	Verilog	10	07/05/2019 01:22:11 ...
ID_EX.v	✓	Verilog	12	07/07/2019 03:13:26 ...
IF_ID.v	✓	Verilog	11	07/07/2019 02:55:02 ...
IM.v	✓	Verilog	6	07/05/2019 11:58:40 ...
MEM_WR.v	✓	Verilog	14	07/07/2019 02:56:11 ...
MUX.v	✓	Verilog	9	07/07/2019 02:05:30 ...
NPC.v	✓	Verilog	5	07/06/2019 12:40:04 ...
PC.v	✓	Verilog	4	07/06/2019 12:35:31 ...
RegFile.v	✓	Verilog	15	07/07/2019 12:37:39 ...
define_op.v	✓	Verilog	1	07/05/2019 02:02:43 ...
forwarding.v	✓	Verilog	16	07/07/2019 04:13:51 ...
load_use.v	✓	Verilog	17	07/07/2019 02:56:48 ...
mips.v	✓	Verilog	0	07/07/2019 04:16:50 ...
testbench.v	✓	Verilog	2	07/05/2019 11:14:14 ...

图1.4 流水线文件层次框图

3. 具体模块化定义

3.1 PC模块

（1）基本描述描述：

用来输出当前指令的地址

（2）接口定义：

信号名	方向	描述
clk	I	时钟信号
reset	I	复位信号
bubble	I	阻塞信号
NPC [31:0]	I	下一地址
PC [31:0]	O	输出地址

表1.1　PC模块定义

（3）功能定义：

序号	功能名称	功能描述
1	复位	rst=1 时，将 pc 置为 0X0000_3000
2	输出地址	时钟信号到来时，将 NPC 赋给 PC

表1.2 PC功能定义

3.2 NPC模块

（1）基本描述：

计算下一条指令并输出

（2）接口定义：

信号名	方向	描述
Zero	I	ALU 计算结果：1 表示当前两寄存器(rs,rt)值相等;0 表示不相等。
Branch	I	分支指令，1为是，0为否
Jump	I	是否为J指令，1为是，0为否
Imm16[15:0]	I	需要扩展的立即数
Target[15:0	I	J跳转的立即数
PC [31:2]	I	当前正常的指令地址
NPC [31:2]	O	输出地址

表1.3 NPC模块定义

（3）功能定义

序号	功能名称	功能描述
1	输出地址	正常顺序的取指地址
2	输出地址	根据zero与Branch的值输出下一条指令的地址
3	输出J跳转地址	根据jump和target 的值输出下一条指令的地址

表1.4 NPC功能定义

3.3 im_4k模块

（1）基本描述：

取指令模块，指令内存大小为 4K，初始化从 data.txt 载入指令。根据输入的指令地址，输出当前位置存储的指令。

（2）接口定义：

信号名	方向	描述
Addr[31:2]	I	指令地址
Dout[31:0]	O	指令

表1.5 im_4k模块定义

（3）功能定义

序号	功能名称	功能描述
1	载入指令	初始化载入data.txt中的指令
2	输出指令	根据指令地址，输出指令

表1.6 im_4k功能定义

3.4 ALU模块

（1）基本描述：

实现基本的R型指令以及其他需要进行ALU运算的指令，如 addu，subu，slt，and，nor， or，xor，sll，srl，sltu ，sllv，sra， srav，srlv，lui，slti，sltiu 等基本操作。

（2）接口定义：

信号名	方向	描述
A[31:0]	I	输入数据端口
B[31:0]	I	输入数据端口
Shf[4:0]	I	移位操作
ALUOp[4:0]	I	选择ALU操作
Result[31:0]	O	计算结果
Zero	O	0标志

表1.7 ALU模块定义

（3）功能定义

序号	功能名称	功能描述
1	输出计算结果	根据 alu 控制信号，输出A 与 B的计算结果
2	输出零标识位	根据计算结果判断是否为0,1表示为0；0表示不为0
3	基本运算	进行加法、减法、逻辑左右移、变量移、与或非、异或等操作

表1.8 ALU功能定义

3.5 MUX模块

（1）基本描述：

根据输入信号选择不同的输出结果，有不同位宽的选择器；有位宽为5的选择rd和rt，以及32位的选择器。

（2）接口定义：

信号名	方向	描述
Muxop	I	选择控制信号
Cin_a[4:0]	I	待选择a
Cin_b[4:0]	I	待选择b

表1.9 MUX模块定义

（3）功能定义

序号	功能名称	功能描述
1	5位的选择器	选择输出对应信号的5位结果
2	32位的选择器	选择输出对应信号的32位结果

表1.10 MUX功能定义

3.6 RegFile模块

（1）基本描述：

在时钟信号控制下，根据输入的两个寄存器地址，输出相应寄存器的值，根据寄存器写信号

和寄存器地址，将输入的数据选择写入寄存器。

（2）接口定义：

信号名	方向	描述
CLK	I	时钟信号
WEn	I	写使能，在时钟上升沿到来时，为1则写数据
RA[4:0]	I	Rs寄存器
RB[4:0]	I	Rt寄存器
RW[4:0]	I	被写数据的寄存器Rd或者Rt
busW[31:0]	I	待写的数据
busA[31:0]	O	Rs寄存器存储的内容
busB[31:0]	O	Rt寄存器存储的内容
CLK	I	时钟信号
WEn	I	写使能，在时钟上升沿到来时，为1则写数据
RA[4:0]	I	Rs寄存器

表1.11　RegFile模块定义

（3）功能定义

序号	功能名称	功能描述
1	写数据到对应寄存器中	根据写使能和时钟，写数据到制定的寄存器
2	读取对应的寄存器内的数据	在时钟控制下，读取对应寄存器中的内容数据

表1.12　RegFile功能定义

3.7 dm_4k模块

（1）基本描述：

用于存储数据的的数据存储器，内存。

（2）接口定义：

信号名	方向	描述
Address[11：0]	I	地址线
Bitop	I	bit操作
Extop	I	Bit操作时的外扩展信号
CLK	I	时钟信号
WE	I	写使能
din[31:0]	I	待写数据
dout[31:0]	O	输出数据

表1.13　dm_4k模块定义

（3）功能定义

序号	功能名称	功能描述
1	读数据内存	根据地址读出内存中的数据，word

2	读数据内存	根据地址读出内存中的数据，bit字节

表1.14　dm_4k功能定义

3.8 Ctrl模块

（1）基本描述：

根据指令中的op和func得到，R型指令的基本是一致的，特别地处理逻辑右移和逻辑左移以及算术右移时需要产生shfop标志；因为bgez和bltz的op域相同，无法产生可以区分的控制信号，所以需要再根据rt产生ALUop。

（2）接口定义：

信号名	方向	描述
op[5:0]	I	指令高6位
func[5:0]	I	指令的func段
rt[4:0]	I	rt寄存器
zero	I	ALU中得到的0标志
PCWre	O	PC修改标志
ALUSrc	O	ALU端选择立即数或者busB
RegWr	O	寄存器写使能信号
RegDst	O	选择rd或rt
MentoReg	O	选择ALU输出端或者内存数据信号
MenWr	O	DM写使能
ExtOp	O	符号扩展信号
cin	O	进位
Branch	O	分支指令信号
Jump	O	j跳转指令信号
Shfop	O	逻辑左右移和算术右移信号
Bitop	O	dm的bit操作信号
ALUOp[4:0]	O	ALUop控制信号

表1.15　RegFile模块定义

（3）功能定义

序号	功能名称	功能描述
1	输出控制信号	根据指令的op域和func段等，输出所有的控制信号
2	输出几个逻辑和算术位移信号	Shfop用于几个逻辑位移
3	生成bgez和bltz的ALUop	根据rt生成bgez和bltz

表1.16　Ctrl功能定义

3.9 Extend模块

（1）基本描述：

将16位的立即数进行符号或者0扩展为32位，将8位的立即数符号或者0扩展为32位.

（2）接口定义：

信号名	方向	描述
Extop	I	符号或0扩展控制信号
Immediate	I	立即数输入
Extendimmdiate	O	扩展输出

表1.17　Extend模块定义

（3）功能定义：

序号	功能名称	功能描述
1	符号扩展为32位	根据控制信号进行扩展
2	0扩展为32位	根据控制信号进行扩展

表1.18　Extend功能定义

3.10 IFID寄存器

（1）基本描述：

IFID流水段寄存器用于实现取指与译码阶段，，时钟信号上升沿时把信号传给下一流水段或寄存器。

（2）接口定义：

信号名	方向	描述
Clk	I	时钟信号
flush	I	清空信号，置为0
stall	I	阻塞信号
Pc_in	I	输入地址
Inst_in	I	输入指令
Pc_out	O	译码阶段的指令对应的地址
Inst_out	O	译码阶段指令

表1.19　IFID模块定义

3.11 IDEX寄存器

（1）基本描述：

将16位的立即数进行符号或者0扩展为32位，将8位的立即数符号或者0扩展为32位.

（2）接口定义：

信号名	方向	描述
clk	I	时钟信号
flush	I	清空信号
stall	I	阻塞信号
RegWr_in	I	寄存器写使能信号
RegDst_in	I	选择写寄存器
ALUsrc_in	I	ALUB输入端选择信号
MemWr_in	I	Dm写使能信号
MemRead_in	I	DM读数据使能信号
Extop	I	符号扩展信号

MemtoReg_in	I	选择写回寄存器堆的数据端口
Bitop_in	I	bit操作信号
Branch_in	I	分支跳转指令信号
Jump_in	I	J跳转
ALUop[4:0]	I	ALUop域
Rs_in[4:0]	I	Rs
Rt_in[4:0]	I	Rt
Rd_in[4:0]	I	Rd
Target_in[25:0]	I	J跳转的立即数
Imm16_in[15:0]	I	Branch的立即数
Inst_in[31:0]	I	指令
busA_in[31:0]	I	寄存器堆中读取busA
busB_in[31:0]	I	寄存器堆中读取busA
RegWr_out	0	对应的输出信号，等待用于下一流水段的使用
RegDst_out	0	
ALUsrc_out	0	
MemWr_out	0	
MemRead_out	0	
Extop_out	0	
MemtoReg_out	0	
Bitop_out	0	
Branch_out	0	
Jump_out	0	
ALUop_out[4:0]	0	
Rs_out[4:0]	0	
Rs_out[4:0]	0	
Rs_iout4:0]	0	
Target_out[25:0]	0	
Imm16_out[15:0]	0	
Inst_out[31:0]	0	
busA_out[31:0]	0	
busB_out[31:0]	0	

表1.20　IDEX模块定义

3.12 EXMEM寄存器

（1）基本描述：

将16位的立即数进行符号或者0扩展为32位，将8位的立即数符号或者0扩展为32位.

（2）接口定义：

信号名	方向	信号名
clk	I	时钟信号
flush	I	清空信号
stall	I	阻塞信号
RegWr_in	I	寄存器写使能信号
RegDst_in	I	选择写寄存器
ALUsrc_in	I	ALUB输入端选择信号
MemWr_in	I	Dm写使能信号
MemRead_in	I	DM读数据使能信号
Extop	I	符号扩展信号
MemtoReg_in	I	选择写回寄存器堆的数据端口
Bitop_in	I	bit操作信号
Branch_in	I	分支跳转指令信号
Jump_in	I	J跳转
Zero	I	0标志
Rs_in[4:0]	I	Rs
Rt_in[4:0]	I	Rt
Rd_in[4:0]	I	Rd
Target_in[25:0]	I	J跳转的立即数
Imm16_in[15:0]	I	Branch的立即数
Inst_in[31:0]	I	指令
Result_in[31:0]	I	ALU运算结果
busB_in[31:0]	I	寄存器堆中读取busA
RegWr_out	0	对应的输出信号，等待用于下一流水段的使用
RegDst_out	0	
ALUsrc_out	0	
MemWr_out	0	
MemRead_out	0	
Extop_out	0	
MemtoReg_out	0	
Bitop_out	0	
Branch_out	0	
Jump_out	0	
Zero	0	
Rs_out[4:0]	0	

Rs_out[4:0]	0	
Rs_iout4:0]	0	
Target_out[25:0]	0	
Imm16_out[15:0]	0	
Inst_out[31:0]	0	
Result_out[31:0]	0	
busB_out[31:0]	0	

表1.21　EXMEM模块定义

3.13 MEMWB寄存器

（1）基本描述：

将16位的立即数进行符号或者0扩展为32位，将8位的立即数符号或者0扩展为32位.

（2）接口定义：

信号名	方向	信号名
clk	I	时钟信号
stall	I	阻塞信号
RegWr_in	I	寄存器写使能信号
RegDst_in	I	选择写寄存器
MemtoReg_in	I	选择写回寄存器堆的数据端口
Rt_in[4:0]	I	Rt
Rd_in[4:0]	I	Rd
Result_in[31:0]	I	ALU运算结果
Dout_in[31:0]	I	DM中取出的数据
RegWr_out	0	对应的输出信号，等待用于下一流水段的使用
RegDst_out	0	
MemtoReg_out	0	
Rt_out[4:0]	0	
Rd_iout4:0]	0	
Result_out[31:0]	0	
_out[31:0]	0	

表1.22　MEMWB模块定义

3.14 MIPS模块

（1）基本描述：

顶层模块，用于连接各模块。

（2）接口定义：

信号名	方向	描述
Clk	I	时钟信号
reset	I	复位信号

表1.23 MIPS模块定义

3.15 Forward模块

（1）基本描述：

处理数据冒险，用于转发数据。

（2）接口定义：

信号名	方向	描述
Clk	I	时钟信号
reset	I	复位信号

表1.24 Forward模块定义

3.16 load_use模块

（1）基本描述：

处理数据冒险，用于转发数据。

（2）接口定义：

信号名	方向	描述
Clk	I	时钟信号
reset	I	复位信号

表1.25 load_use模块定义

3.17 controlhazard模块

（1）基本描述：

处理数据冒险，用于转发数据。

（2）接口定义：

信号名	方向	描述
Clk	I	时钟信号
reset	I	复位信号

表1.25 controlhazard模块定义

4. 测试代码及结果

4.1 具体Verilog代码实现

PC模块

```
module PC(CLK,reset,bubble,NPC,PC);
   input CLK;              // 时钟
   input reset;            // 重置信号
   input bubble;           // 阻塞信号
   input [31:2] NPC;       // PC指令地址
   output reg[31:2] PC ;
         initial begin
                PC <= 32'h0000_3000;
         end

         always@(posedge CLK)
          begin
                if (reset == 1)  PC <= 32'h0000_3000;
                else
			begin
				if (bubble)  PC <= PC;  //阻塞PC地址保持不变
				else  PC <= NPC;
			end
		end
Endmodule
```

NPC模块

```
module NPC(PC,Zero,Branch,Jump,target,imm16,Npc);
input	Zero;
input	Branch;
input	Jump;
input [31:2] PC;
input [15:0] imm16;
input [25:0] target;
output[31:2] Npc;
wire  [31:0] imm32;
wire  [31:2] JNPC;
wire  [31:2] BNPC;
wire  [31:2] NNPC;

Extend16 extnpc(1,imm16,imm32);
assign BNPC = PC+1+imm32[29:0];
assign JNPC = {PC[31:28],target};
assign NNPC =  PC + 1;
assign Npc  = Jump?JNPC:((Branch&Zero)?BNPC:NNPC);
endmodule
```

EXtend模块

```
//16位->32位
module Extend16(
input Extop,
input [15:0] immediate,
 output [31:0] extendImmediate
);
assign extendImmediate[15:0] = immediate;
assign extendImmediate[31:16] = Extop ? (immediate[15] ? 16'hffff : 16'h0000) : 16'h0000;
endmodule
```

MUX模块

```
//16位->32位

//选择写寄存器 rd or rt
// 5位2路
module Mux2_5(
		input MuxOp,
   input [4:0] cin_a,
   input [4:0] cin_b,
   output [4:0] out
   );

assign out = MuxOp ?  cin_b : cin_a;

endmodule

//32位2路
module Mux2_32(
            input MuxOp,
    input [31:0] cin_a,
    input [31:0] cin_b,
    output [31:0] out
    );

assign out = MuxOp ? cin_b : cin_a;

endmodule


//32位3路
module Mux3_32(
            input [1:0]  MuxOp,
    input [31:0] cin_a,
    input [31:0] cin_b,
    input [31:0] cin_c,
    output[31:0] out
    );

   reg [31:0] reg_out;

always @( * ) begin
        case (MuxOp)
            2'b00: reg_out = cin_a;
            2'b01: reg_out = cin_b;
            2'b10: reg_out = cin_c;
            default: ;
        endcase
    end

    assign out =  reg_out;
endmodule

//32位4路
module Mux4_32(
            input [1:0]  MuxOp,
    input [31:0] cin_a,
    input [31:0] cin_b,
    input [31:0] cin_c,
    input [31:0] cin_d,
    output[31:0] out
    );

   reg [31:0] reg_out;

always @( * ) begin
        case ( MuxOp )
            2'b00: reg_out = cin_a;
            2'b01: reg_out = cin_b;
            2'b10: reg_out = cin_c;
            2'b11: reg_out = cin_d;
            default: ;
        endcase
    end
    assign out =  reg_out;
endmodule
```

Im_4k模块

```
module im_4k(
   input  [11:2] addr,    //  读取指令存储地址
   output [31:0] dout ); //  指令内容
reg[31:0] im[0:1023];      //  指令存储器
initial
          begin
                  $readmemh("data.txt", im);  //读取文件中的指令 b-2进制 h-16进制
          end
          assign dout  = im[addr];
endmodule
```

dm_4k模块

```
module dm_4k( addr,Bitop,Extop,din, we,MemRead, clk, dout ) ;
input [11:0] addr ;
input Bitop;        // 判断是否进行字节的读写
input Extop;        // bit读时扩展
input MemRead;      // 读使能
input [31:0] din ;  // 待写数据
input we;           //写使能
input clk ;
output reg [31:0] dout ;  //输出数据  bit时候需要进行位扩展
reg [31:0] dm[0:1023];
reg [1:0] pos;   //地址的后两位 据此选择bit
reg [7:0] bit;
reg [31:0] Ext_bit;

always@(clk)
begin
          pos <= addr[1:0];
   if(we)
                  begin
                     if(!Bitop)
                     begin
                          dm[addr] <= din;
                      end
                     else
               case(pos)
            2'b00:  dm[addr[11:2]]  =   {dm[addr[11:2]][31:8],din[7:0] };
                                    2'b01:  dm[addr[11:2]]  =
{dm[addr[11:2]][31:16],din[7:0],dm[addr[11:2]][7:0] };
                                    2'b10:  dm[addr[11:2]]  =
{dm[addr[11:2]][31:25],din[7:0],dm[addr[11:2]][15:0] };
            2'b11:  dm[addr[11:2]]  =   {din[7:0],dm[addr[11:2]][23:0] };
         endcase
                   end
                  // 读内存
         else
   begin
                  if(MemRead)
                  begin
                     if(!Bitop)    dout <= dm[addr[11:2]];
                     else
                     begin
         case(pos)
            2'b00:  bit  <=   dm[addr[11:2]][7:0];
```

```
                                    2'b01:  bit  <=   dm[addr[11:2]][15:8];
                                    2'b10:  bit  <=   dm[addr[11:2]][23:16];
          2'b11:  bit  <=   dm[addr[11:2]][31:24];
                              endcase
          if(Extop)
             begin
                if(bit[7])  Ext_bit <=  {24'hffffff,bit};
                 else
                 Ext_bit <=  {24'h000000,bit};
              end
         else    Ext_bit =  {24'b0,bit};
         dout <= Ext_bit;
         end
               end
        end
end
endmodule
```

ALU模块

```
module Alu(A,B,ALUop,shf,Result,Zero);
input[31:0]  A,B;
input[4:0]   ALUop;
input [4:0]  shf; //移位
output[31:0] Result;
reg[31:0]    Result;
output reg   Zero; //0标识
initial
begin
 Zero = 0;
end
always@(ALUop or A or B)
begin
  case (ALUop)
   5'b00000: Result = A+B;                             // addu
   5'b00001: Result =  $signed(A) < 32'b0 ? 1:0;             // bgez 大于或等于0跳转(小于，zero标志为1)
   5'b00010: Result= A-B;                              // sub,subu,beq
   5'b00011: Result =  $signed(A)>0 ? 0:1;                  // bgtz  大于0跳转
   5'b00100: Result = ($unsigned(A)<$unsigned(B))?1:0;        // sltu
   5'b00101: Result = ($signed(A)<$signed(B))?1:0;           // slt
   5'b00110: Result = A&B;                             // and
   5'b00111: Result = ~(A|B);                           // nor
   5'b01000: Result = A|B;                            // or
   5'b01001: Result = A^B;                             // xor
   5'b01010: Result = B<<shf;                           // sll
   5'b01011: Result = B<<A;                            // sllv
   5'b01100: Result = $signed(B)>>shf;                     // sra,srl
   5'b01101: Result = $signed(B)>>A;                      // srav
   5'b01110: Result = $unsigned(B)>>A;                     // srlv
   5'b01111: Result =  $signed(A) > 0 ? 1:0;                // blez  小于或等于0跳转
   5'b10000: Result =  $signed(A)<0 ? 0:1;                  // bltz 小于0跳转
   5'b10001: Result = B<<5'b10000;                        // lui 左移16位
   5'b10010: Result =  A!= B ? 0:1;                        // bne
   5'b10010: Result =  A;                              // jalr,jr
  endcase
Zero =(Result)?0:1 ;
end
endmodule
```

RegFile模块

```
module RF(clk,WEn,RW,RA,RB,busW,busA,busB);
input clk,WEn;
input[4:0] RW,RA,RB;
input[31:0] busW;
output [31:0]  busA,busB;
reg[31:0]  regfile[0:31];
integer i;
          initial
           begin
                    for(i = 0; i < 31; i = i + 1)  regfile[i] <= 0;
           end
///时钟下降沿到来写
////先写后读
always@(clk )
  begin
    if(WEn)
           regfile[RW] <= busW;
   end
  assign busA = regfile[RA];
  assign  busB = regfile[RB] ;
Endmodule
```

IFID_REG模块

```
module IFID (clk,flush,stall,pc_in ,inst_in, pc_out,inst_out);
input        clk;
input        flush;//清空
input        stall;
input  [31:0] pc_in; // 当前取指令地址
input   [31:0] inst_in;//
output reg [31:0] pc_out;  //下一条指令地址
output reg[31:0] inst_out;//

reg [31:0]pc;
reg [31:0] inst;
 always @(posedge clk )
   begin
    if(flush==1)
      begin
         pc <=0;
        inst <= 0;
      end
     else if(stall==1)
       begin
         inst_out <= inst;
         pc_out<= pc;
        end
     else
       begin
         pc = pc_in;
         inst = inst_in;
         inst_out <= inst_in;
         pc_out <= pc_in;
       end
  end
endmodule
```

IDEX_REG模块

```
module IDEX
(clk,flush,stall,RegWr_in,RegDst_in,ALUsrc_in,MemWr_in,MemRead_in,Extop_in,MemToReg_in,Bitop_i
n,Branch_in,Jump_in,
ALUop_in,rs_in,rt_in,rd_in,target_in,imm16_in,inst_in,busA_in,busB_in,
RegWr_out,RegDst_out,ALUsrc_out,MemWr_out,MemRead_out,Extop_out,MemToReg_out,Bitop_out,Br
anch_out,Jump_out,
ALUop_out,rs_out,rt_out,rd_out,target_out,imm16_out,inst_out,busA_out,busB_out
);

input       clk;//时钟信号
input       flush;//清空
input       stall; //阻塞
////Input：
input  RegWr_in;
input  RegDst_in;
input  ALUsrc_in;
input  MemWr_in;
input  MemRead_in;
input  Extop_in;
input  MemToReg_in;
input  Bitop_in;
input  Branch_in;
input  Jump_in;
input  [4:0]  ALUop_in;
input  [4:0]  rs_in ;
input  [4:0]  rt_in;
input  [4:0]  rd_in;
input  [25:0] target_in;
input  [15:0] imm16_in;
input  [31:0] inst_in;
input  [31:0] busA_in;
input  [31:0] busB_in;
//// Output:
output reg RegWr_out;
output reg RegDst_out;
output reg ALUsrc_out;
output reg MemWr_out;
output reg MemRead_out;
output reg Extop_out;
output reg MemToReg_out;
output reg Bitop_out;
output reg Branch_out;
output reg Jump_out;
output reg [4:0] ALUop_out;
output reg [4:0]  rs_out ;
output reg [4:0] rt_out;
output reg [4:0] rd_out;
output reg [25:0] target_out;
output reg [15:0] imm16_out;
output reg [31:0] inst_out;
output reg [31:0] busA_out;
output reg [31:0] busB_out;

        always@(posedge clk)
   begin
    if ( flush )  //清空信号
```

```
        begin
        RegWr_out  <= 0;
        RegDst_out <= 0;
        ALUsrc_out <= 0;
        MemWr_out <= 0;
        MemRead_out <= 0 ;
        Extop_out <= 0;
        MemToReg_out <= 0;
        Bitop_out <= 0;
        Branch_out <= 0;
        Jump_out <= 0;
        ALUop_out <= 0;
        rs_out <= 0;
        rd_out <= 0;
        rt_out <= 0;
        target_out <= 0;
        imm16_out <= 0;
        inst_out <= 0;
        busA_out <= 0;
        busB_out <= 0;
        end
      else if(!stall) //阻塞信号
         begin
           RegWr_out  <= RegWr_in;
           RegDst_out <= RegDst_in;
           ALUsrc_out <= ALUsrc_in;
           MemWr_out <= MemWr_in;
           MemRead_out <= MemRead_in;
           Extop_out <= Extop_in;
           MemToReg_out <= MemToReg_in;
           Bitop_out <= Bitop_in;
           Branch_out <= Branch_in;
           Jump_out <= Jump_in;
           ALUop_out <= ALUop_in;
           rd_out <= rd_in;
           rs_out <= rs_in;
           rt_out <= rt_in;
           target_out <= target_in;
           imm16_out <= imm16_in;
           inst_out <= inst_in;
           busA_out <= busA_in;
           busB_out <= busB_in;
         end
   end
endmodule
```

EXMEM_REG模块

```
module EXMEM
(clk,flush,stall,RegWr_in,RegDst_in,MemWr_in,MemRead_in,MemtoReg_in,Branch_in,Jump_in,Zero_in,
Extop_in,Bitop_in,rs_in,rt_in,rd_in,busB_in,Result_in,
RegWr_out,RegDst_out,MemWr_out,MemRead_out,MemtoReg_out,Branch_out,Jump_out,Zero_out,Extop
_out,Bitop_out,rs_out,rt_out,rd_out,busB_out,Result_out
);
input clk;
input flush;
input stall;
/////INput
input RegWr_in;
input RegDst_in;
input MemWr_in;
input MemRead_in;
input MemtoReg_in;
input Branch_in;
input Jump_in;
input Zero_in;
input Extop_in;
input Bitop_in;
input [4:0] rs_in;
input [4:0] rt_in;
input [4:0] rd_in;
input [31:0] busB_in;
input [31:0] Result_in;
/////Output
output reg RegWr_out;
output reg RegDst_out;
output reg MemWr_out;
output reg MemRead_out;
output reg MemtoReg_out;
output reg Branch_out;
output reg Jump_out;
output reg Zero_out;
output reg Extop_out;
output reg Bitop_out;
output reg [4:0]rs_out;
output reg [4:0]rt_out;
output reg [4:0]rd_out;
output reg [31:0] busB_out;
output reg [31:0] Result_out;

always @(posedge clk )
   begin
      if (flush) //清空
        begin
            RegWr_out <= 0;
            RegDst_out <= 0;
            MemWr_out <= 0;
            MemRead_out <= 0 ;
            MemtoReg_out <= 0;
            Branch_out <= 0;
            Jump_out  <= 0;
            Zero_out  <= 0;
            Extop_out <= 0;
            Bitop_out <= 0;
            rs_out    <= 0;
            rt_out    <= 0;
            rd_out    <= 0;
            busB_out   <= 0;
            Result_out <= 0;
        end
      if(!stall) //非阻塞
        begin
            RegWr_out <= RegWr_in;
             RegDst_out <= RegDst_in;
            MemWr_out <= MemWr_in;
            MemRead_out <= MemRead_in;
            MemtoReg_out <= MemtoReg_in;
            Branch_out <= Branch_in;
            Jump_out  <= Jump_in;
            Zero_out  <= Zero_in;
            Extop_out <= Extop_in;
            Bitop_out <= Bitop_in;
            rs_out   <= rs_in;
            rt_out   <= rt_in;
            rd_out    <= rd_in;
            busB_out  <= busB_in;
            Result_out <= Result_in;
        end
   end
endmodule
```

MEMWB_REG模块

```
module MEMWB( clk,stall,
RegWr_in,RegDst_in,MemToReg_in,rt_in,rd_in,Dout_in,Result_in,
RegWr_out,RegDst_out,MemToReg_out,rt_out,rd_out,Dout_out,Result_out
);

input  clk;
input  stall;
////Input
input RegWr_in; //Regfile写
input RegDst_in;//选择rd或者rt
input MemToReg_in;//选择写Reg的内容
input [4:0] rt_in;
input [4:0] rd_in;
input [31:0]  Dout_in; //DM读出的内容
input  [31:0] Result_in; //ALU计算结果，DM地址或者Reg待写内容
////Output
output reg RegWr_out;
output reg RegDst_out;
output reg MemToReg_out;
output reg [4:0] rt_out;
output reg [4:0] rd_out;
output reg [31:0] Dout_out;
output reg [31:0] Result_out;

 always @(posedge clk )
  begin
    if(stall)
      begin
      RegWr_out <= 0;
      RegDst_out <= 0;
      MemToReg_out <= 0;
      rt_out <= 0;
      rd_out <= 0;
      Dout_out <= 0;
      Result_out <= 0;
      end
     else
      begin
      RegWr_out <= RegWr_in;
      RegDst_out <= RegDst_in;
      MemToReg_out <= MemToReg_in;
      rt_out <= rt_in;
      rd_out <= rd_in;
```

```
        Dout_out <= Dout_in;
        Result_out <=Result_in;
        end
  end
Endmodule
```

Forward模块

```
//转发主要解决
//采用助教文档中的案例
//从Reg寄存器中读取数据，数据还未被写入，即读取的是旧值
module DataHazard(Mem_RegWr, Wr_RegWr,MemtoReg,Mem_Rw, Ex_Rs, Ex_Rt, Wr_Rw,
              Ex_ALUSrc, Ex_imm32, Ex_busA, Ex_busB, Mem_Result,Wr_Result, Wr_din,
              A, B);
input        Mem_RegWr, Wr_RegWr,MemtoReg, Ex_ALUSrc;
input  [4:0]  Mem_Rw, Ex_Rs, Ex_Rt, Wr_Rw;
input [31:0]  Ex_imm32; // 立即数
input [31:0]  Ex_busA;
input [31:0]  Ex_busB;
input [31:0]  Mem_Result,Wr_Result;
input [31:0]  Wr_din;
output [31:0] A, B; //ALU的A和B输入端口
wire         C1A, C1B, C2A, C2B,B2;
wire  [31:0]  busB;
reg   [1:0]  ALUSrcA, ALUSrcB;
initial
begin
   ALUSrcA = 0;
   ALUSrcB = 0;
end
assign C1A = Mem_RegWr&&(Mem_Rw != 0)&&(Mem_Rw == Ex_Rs)||Wr_RegWr&&(Mem_Rw !=
0)&&(Mem_Rw == Ex_Rs);  //选择Result
assign C1B = Wr_RegWr&&(Mem_Rw != 0)&&(Mem_Rw == Ex_Rt)|| Mem_RegWr&&(Mem_Rw !=
0)&&(Mem_Rw == Ex_Rt); // 选择Result
/*注意到此处助教给出代码的不足
C1B判定条件
原： Wr_RegWr&&(Mem_Rw != 0)&&(Mem_Rw == Ex_Rt)
修改后：Wr_RegWr&&(Mem_Rw != 0)&&(Mem_Rw == Ex_Rt)|| Mem_RegWr&&(Mem_Rw !=
0)&&(Mem_Rw == Ex_Rt)
*/
assign C2A = MemtoReg&&(Wr_Rw != 0)&&(Mem_Rw != Ex_Rs)&&(Wr_Rw == Ex_Rs); //选择
DM_out
assign C2B = MemtoReg&&(Wr_Rw != 0)&&(Mem_Rw != Ex_Rt)&&(Wr_Rw == Ex_Rt); //选择DM_out
assign  B2 = Wr_RegWr&&(Wr_Rw==Ex_Rt);
always@(C1A or C2A)
begin
  if (C1A !=1 && C2A != 1) ALUSrcA <= 2'b00;
  if (C1A == 1) ALUSrcA <= 2'b01;
  if (C2A == 1) ALUSrcA <= 2'b10;
end
always@(C1B or C2B)
begin
   if (C1B != 1 && C2B != 1) ALUSrcB <= 2'b00;
   if (C1B == 1) ALUSrcB <= 2'b01;
   if (C2B == 1) ALUSrcB <= 2'b10;
end
assign A = (ALUSrcA == 2'b00) ? Ex_busA :(ALUSrcA == 2'b01) ? Mem_Result :(ALUSrcA == 2'b10) ?
Wr_din : 0 ;
```

```
/*ALU输入A端：
 00 正常情况，选择Ex_busA，不发生数据冒险
 01 选择 Mem_Result
 10 选择Wr_din 即从Dm中读取出的数据，这时候应该 MemtoReg = 1
*/
assign busB = B2 ?Wr_Result: Ex_busB;
assign B = Ex_ALUSrc ? Ex_imm32 :((ALUSrcB == 2'b00) ? busB:((ALUSrcB == 2'b01) ?
Mem_Result :((ALUSrcB == 2'b10) ?  Wr_din : 0)));
endmodule
```

Loaduse模块

```
/* 课本P211
需要实现的三个操作：
1.将ID/EX寄存器的控制信号清零
2.保持IF/ID寄存器的的值不变，在下一个时钟到来重新进行译码
3.保持PC的值不变，使得Load后面的第二条指令在下一时钟周期重新取指令
*/
//冒险产生的结果
//保持PC不变 pc_stall = 1
//清空ID/EX IDEX_flush = 1
//保持IFID不变 IFID_stall = 1
module Load_use(IDEX_MemRead,IFID_rt,IFID_rs,IDEX_rt,pc_stall,IFID_stall,IDEX_flush);
input IDEX_MemRead;
input [4:0]IFID_rt,IFID_rs,IDEX_rt;
output reg pc_stall;
output reg IFID_stall;
output reg IDEX_flush;

always @(1)
begin
            if((IDEX_MemRead==1&&(IDEX_rt==IFID_rs||IDEX_rt==IFID_rt)))
             begin
        pc_stall= 1;
               IFID_stall = 1;
               IDEX_flush = 1;
             end
            else
             begin
       pc_stall= 0;
              IFID_stall = 0;
              IDEX_flush = 0;
                  end
end
endmodule
```

Ctrlhazard模块

```
//处理分支跳转指令
/*Branch & Jump
bne  beq
bgez bgtz blez bltz
J
*/
module Ctrl_Hazar(clk,Sign,Jump,pc_stall,IFID_flush);
input   clk;
input   Sign; ///Branch 地址跳转条件
input   Jump;
output  reg pc_stall;//清空信号
output  reg IFID_flush;

initial begin
        pc_stall = 0;
        IFID_flush = 0;
        end

always @(1)
begin
      //jump flush
                if(Jump==1||Sign==1)   //或者Branch满足跳转
                    begin
             pc_stall = 1;
             IFID_flush = 1;/// IFID——flush
           end
        else
        begin
        pc_stall = 0;
        IFID_flush = 0;
        end
end
Endmodule
```

define模块

```
// define OP

`define RTYPE     6'b000000

`define LB        6'b100000
`define LBU       6'b100100
`define LW        6'b100011
`define SB        6'b101000
`define SW        6'b101011

`define ADDI      6'b001000
`define ADDIU     6'b001001
`define ANDI      6'b001100
`define ORI       6'b001101
`define XORI      6'b001110
`define LUI       6'b001111
`define SLTI      6'b001010
`define SLTIU     6'b001011

`define BEQ       6'b000100
`define BNE       6'b000101
`define BGEZ      6'b000001
`define BGTZ      6'b000111
`define BLEZ      6'b000110
`define BLTZ      6'b000001

`define J         6'b000010
//`define JAL       6'b000011

// func域
`define ADD_func     6'b100000
`define ADDU_func    6'b100001
`define SUB_func     6'b100010
`define SUBU_func    6'b100011
```

```
`define AND_func     6'b100100
`define NOR_func     6'b100111
`define OR_func      6'b100101
`define XOR_func     6'b100110
`define SLT_func     6'b101010
`define SLTU_func    6'b101011
`define SLL_func     6'b000000
`define SRL_func     6'b000010
`define SRA_func     6'b000011
`define SLLV_func    6'b000100
`define SRLV_func    6'b000110
`define SRAV_func    6'b000111
`define JR_func      6'b001000
`define JALR_func    6'b001001

`define BGEZ_Rt      5'b00001
`define BLTZ_Rt      5'b00000
```

Crtl模块

```
  `include "define_op.v"
module
Ctrl(op,func,rt,ALUSrc,RegWr,RegDst,MemtoReg,MemWr,MemRead,Extop,Branch,Jump,Bitop,ALUop);
   input [5:0] op;
   input [5:0] func;
   input [4:0] rt;  //用于区别 bltz与bgez

   output reg ALUSrc;    //ALU选择
   output reg RegWr;    //Regfile写使能
   output reg RegDst;    //Regfile写地址选择，rd or rt
   output reg MemtoReg; //写回结果内容选择
   output reg MemWr;   //存储器写使能
   output reg MemRead;
   output reg Extop;    //符号扩展
   output reg Branch;
   output reg Jump;
   output reg Bitop;    //DM的字节
   output reg [4:0] ALUop; //ALU操作
initial
begin Branch = 0;
     Jump  = 0 ;
     Bitop  =0;
end
always@(op or func )
begin
     Branch = 0;
     Jump  = 0 ;
     Bitop  =0;
     MemRead = 0;
          case(op)
                    `RTYPE: //R型指令 000000
                    begin
     ALUSrc = 0;
                         RegDst = 1 ;
                         MemWr = 0;
                         MemtoReg = 0;
        Branch = 0;
        Jump = 0;
        Bitop = 0;
```

```
            case(func)
`ADD_func:   //6'b100000
   begin
   RegWr = 1;
   ALUop = 5'b00000;
   end
`ADDU_func :   //6'b100001
   begin
   RegWr = 1;
   ALUop = 5'b00000;
   end
`SUB_func  :   //6'b100010
   begin
   RegWr = 1;
   ALUop = 5'b00010;
   end
`SUBU_func :   //6'b100011
   begin
   RegWr = 1;
   ALUop = 5'b00010;
   end
`AND_func  :   //6'b100100
   begin
   RegWr = 1;
   ALUop = 5'b00110;
   end
`NOR_func  :   //6'b100111
   begin
   RegWr = 1;
   ALUop = 5'b00111;
   end
`OR_func   :   //6'b100101
   begin
   RegWr = 1;
   ALUop = 5'b01000;
   end
`XOR_func  :   //6'b100110
   begin
   RegWr = 1;
   ALUop = 5'b01001;
   end
`SLT_func  :   //6'b101010
   begin
   RegWr = 1;
   ALUop = 5'b00101;
   end
`SLTU_func :  //6'b101011
   begin
   RegWr = 1;
   ALUop = 5'b00100;
   end
`SLL_func  :  //6'b000000
   begin
   RegWr = 1;
   ALUop = 5'b01010;
   end
`SRL_func  :   //6'b000010
   begin
   RegWr = 1;
```

```
        ALUop = 5'b01100;
        end
     `SRA_func  :   //6'b000011
        begin
        RegWr = 1;
        ALUop = 5'b01100;
        end
     `SLLV_func :   //6'b000100
        begin
        RegWr = 1;
        ALUop = 5'b01011;
        end
     `SRLV_func :   //6'b000110
        begin
        RegWr = 1;
        ALUop = 5'b01110;
        end
     `SRAV_func :  //6'b000111
        begin
        RegWr = 1;
        ALUop = 5'b01101;
        end
    // `JR_func   :   //6'b001000
    // `JALR_func :   //6'b001001
    default:;
endcase
                end
  `ADDI ://addi 6'b001000
   begin
   RegWr = 1;
   RegDst = 0;
   ALUSrc = 1;
         MemWr = 0;
         MemtoReg = 0;
         Extop = 1;
   Jump = 0;
   Branch = 0;
         ALUop = 5'b00000;
   end

  `ADDIU://addiu  6'b001001
   begin
   RegWr = 1;
   RegDst = 0; // rt
   ALUSrc = 1;
          MemWr = 0;
          MemtoReg = 0;
          Extop = 1;
   Jump = 0;
   Branch = 0;
          ALUop = 5'b00000;
   end

  `ANDI://andi 6'b001100
   begin
   RegWr = 1;
   RegDst = 0;
   ALUSrc = 1;
   MemWr = 0;
```

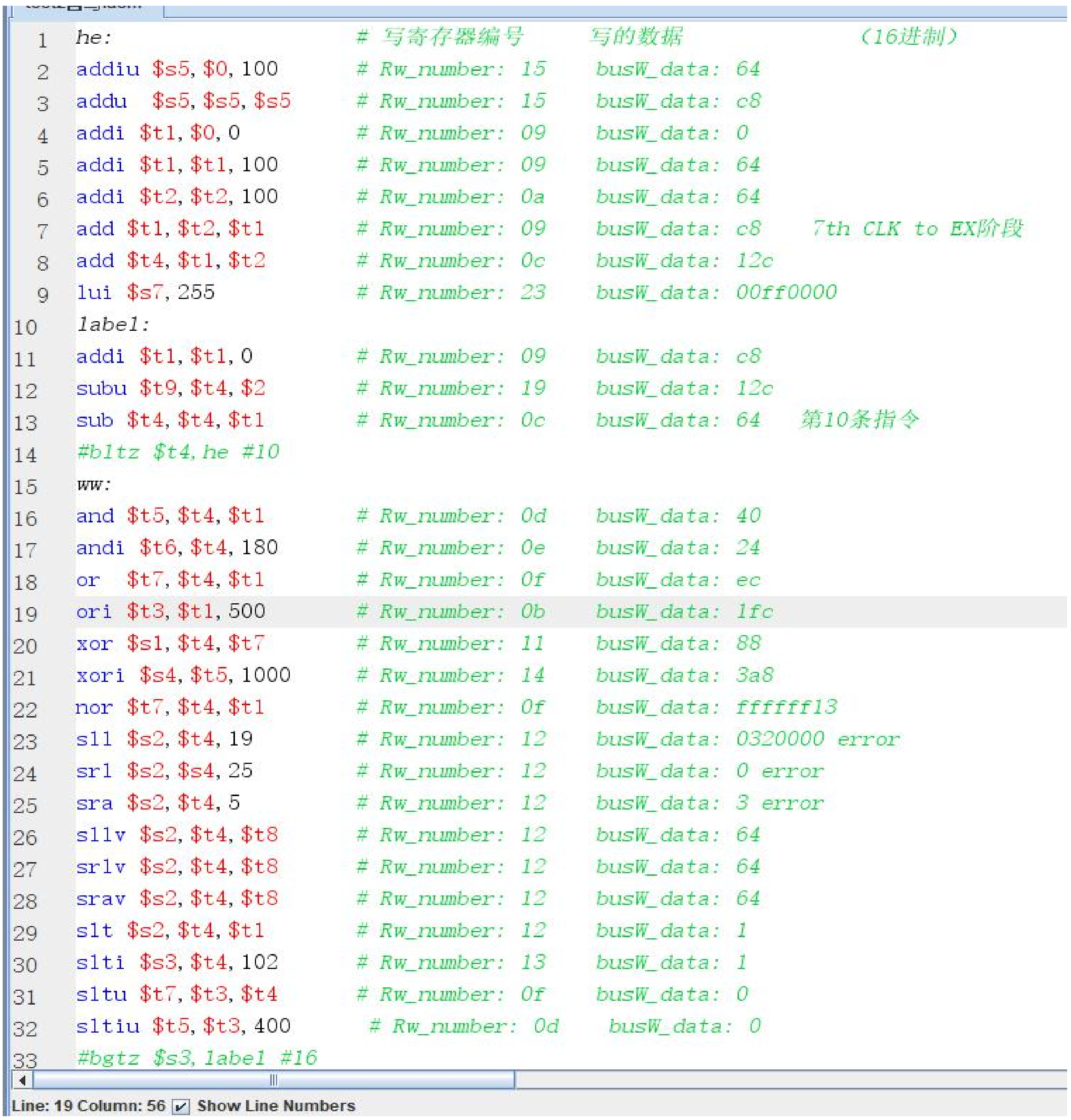
```
 MemtoReg = 0;
 Extop = 0;
 Jump = 0;
 Branch = 0;
 ALUop = 5'b00110;//and
 end

`ORI://ori  6'b001101
 begin
 RegWr = 1;
 RegDst = 0;
 ALUSrc = 1;
 MemWr = 0;
 MemtoReg = 0;
 Extop = 0;
 Jump = 0;
 Branch = 0;
 ALUop = 5'b01000;//or
 end

`XORI://xori 6'b001110
 begin
 RegWr = 1;
 RegDst = 0;
 ALUSrc = 1;
 MemWr = 0;
 MemtoReg = 0;
 Extop = 0;
 Jump = 0;
 Branch = 0;
 ALUop = 5'b01001;//xor
 end

`LW ://lw  6'b100011
 begin
 RegWr = 1;
 RegDst = 0;
 ALUSrc = 1;
 MemWr = 0;
 MemRead = 1;
 MemtoReg = 1;
 Extop = 1;
 Jump = 0;
 Branch = 0;
 ALUop = 5'b00000;
 Bitop  = 0;
 end

`SW://sw 6'b101011
 begin
 RegWr = 0;
 ALUSrc = 1;
 MemWr = 1;
 MemtoReg = 0;
 Extop = 1;
 Jump = 0;
 Branch = 0;
 ALUop = 5'b00000;
 Bitop = 1;
```

```
  end

`LUI://lui  6'b001111
  begin
  RegWr = 1;
  RegDst = 0;
  ALUSrc = 1;
  MemWr = 0;
  MemtoReg = 0;
  Extop = 0;
  Jump = 0;
  Branch = 0;
  ALUop = 5'b10001;
  end

`LBU://lbu 6'b100100
  begin
  RegWr = 1;
  RegDst = 0;
  ALUSrc = 1;
  MemWr = 0;
  MemtoReg = 1;
  Extop = 0;
  Jump = 0;
  Branch = 0;
  ALUop = 5'b00000;
  Bitop = 1;
  end

`LB://lb 6'b100000
  begin
  RegWr = 1;
  RegDst = 0;
  ALUSrc = 1;
  MemWr = 0;
  MemtoReg = 1;
  Extop = 0;
  Jump = 0;
  Branch = 0;
  ALUop = 5'b00000;
  Bitop = 1;
  end

`SB://sb  6'b101000
  begin
  RegWr = 0;
  ALUSrc = 1;
  MemWr = 1;
  Extop = 1;
  Jump = 0;
  Branch = 0;
  ALUop = 5'b00000;
  Bitop = 1;
  end

`SLTI://slti 6'b001010
  begin
  RegWr = 1;
  RegDst = 0;
  Extop = 1;
  MemWr = 0;
  MemtoReg = 0;
  ALUSrc = 1;
  Jump = 0;
  Branch = 0;
  ALUop = 5'b00101;
  end

`SLTIU://sltiu 6'b001011
  begin
  RegWr = 1;
  RegDst = 0;
  Extop = 1;
  MemWr = 0;
  MemtoReg = 0;
  ALUSrc = 1;
  Jump = 0;
  Branch = 0;
  ALUop = 5'b00100;
  end
 6'b000001://bgez区别
  if(rt==`BGEZ_Rt)
  begin
  RegWr = 0;
  ALUSrc = 0;
  MemWr = 0;
  ALUop = 5'b00001;
  Branch = 1;
  Jump = 0;
  end

  else //bltz区别
  begin
  RegWr = 0;
  ALUSrc = 0;
  MemWr = 0;
  Branch = 1;
  ALUop = 5'b10000;
  Jump = 0;
  end

`BGTZ://bgtz 6'b000111
  begin
  RegWr = 0;
  ALUSrc = 0;
  MemWr = 0;
  Branch = 1;
  ALUop = 5'b00011;
  Jump = 0;
  end

 `BLEZ://blez  6'b000110
  begin
  RegWr = 0;
  ALUSrc = 0;
  MemWr = 0;
  Branch = 1;
```

```
            ALUop = 5'b01111;
            Jump = 0;
            end

          `BEQ://beq  6'b000100
            begin
            RegWr = 0;
            RegDst = 0;
            ALUSrc = 0;
            MemWr = 0;
            Branch = 1;
            Jump = 0;
            ALUop = 5'b00010;
            end

          `BNE:   // bne 6'b000101
            begin
            RegWr = 0;
            RegDst = 0;
            ALUSrc = 0;
            MemWr = 0;
            Branch = 1;
            Jump = 0;
            ALUop = 5'b10010;
            end

          `J://j   6'b000010
            begin
            RegWr = 0;
            ALUSrc = 0;
            MemWr = 0;
            Branch = 0;
            Jump = 1;
            end

    default:;
    endcase
end


endmodule
```

4.2 ModelSim模拟及测试

（1） 使用自己写的测试代码，如下所示



```
he:                      # 写寄存器编号     写的数据              （16进制）
addiu $s5,$0,100         # Rw_number: 15    busW_data: 64
addu  $s5,$s5,$s5        # Rw_number: 15    busW_data: c8
addi $t1,$0,0            # Rw_number: 09    busW_data: 0
addi $t1,$t1,100         # Rw_number: 09    busW_data: 64
addi $t2,$t2,100         # Rw_number: 0a    busW_data: 64
add $t1,$t2,$t1          # Rw_number: 09    busW_data: c8     7th CLK to EX阶段
add $t4,$t1,$t2          # Rw_number: 0c    busW_data: 12c
lui $s7,255              # Rw_number: 23    busW_data: 00ff0000
label:
addi $t1,$t1,0           # Rw_number: 09    busW_data: c8
subu $t9,$t4,$2          # Rw_number: 19    busW_data: 12c
sub $t4,$t4,$t1          # Rw_number: 0c    busW_data: 64    第10条指令
#bltz $t4,he #10
ww:
and $t5,$t4,$t1          # Rw_number: 0d    busW_data: 40
andi $t6,$t4,180         # Rw_number: 0e    busW_data: 24
or  $t7,$t4,$t1          # Rw_number: 0f    busW_data: ec
ori $t3,$t1,500          # Rw_number: 0b    busW_data: 1fc
xor $s1,$t4,$t7          # Rw_number: 11    busW_data: 88
xori $s4,$t5,1000        # Rw_number: 14    busW_data: 3a8
nor $t7,$t4,$t1          # Rw_number: 0f    busW_data: ffffff13
sll $s2,$t4,19           # Rw_number: 12    busW_data: 0320000 error
srl $s2,$s4,25           # Rw_number: 12    busW_data: 0 error
sra $s2,$t4,5            # Rw_number: 12    busW_data: 3 error
sllv $s2,$t4,$t8         # Rw_number: 12    busW_data: 64
srlv $s2,$t4,$t8         # Rw_number: 12    busW_data: 64
srav $s2,$t4,$t8         # Rw_number: 12    busW_data: 64
slt $s2,$t4,$t1          # Rw_number: 12    busW_data: 1
slti $s3,$t4,102         # Rw_number: 13    busW_data: 1
sltu $t7,$t3,$t4         # Rw_number: 0f    busW_data: 0
sltiu $t5,$t3,400         # Rw_number: 0d    busW_data: 0
#bgtz $s3,label #16
```

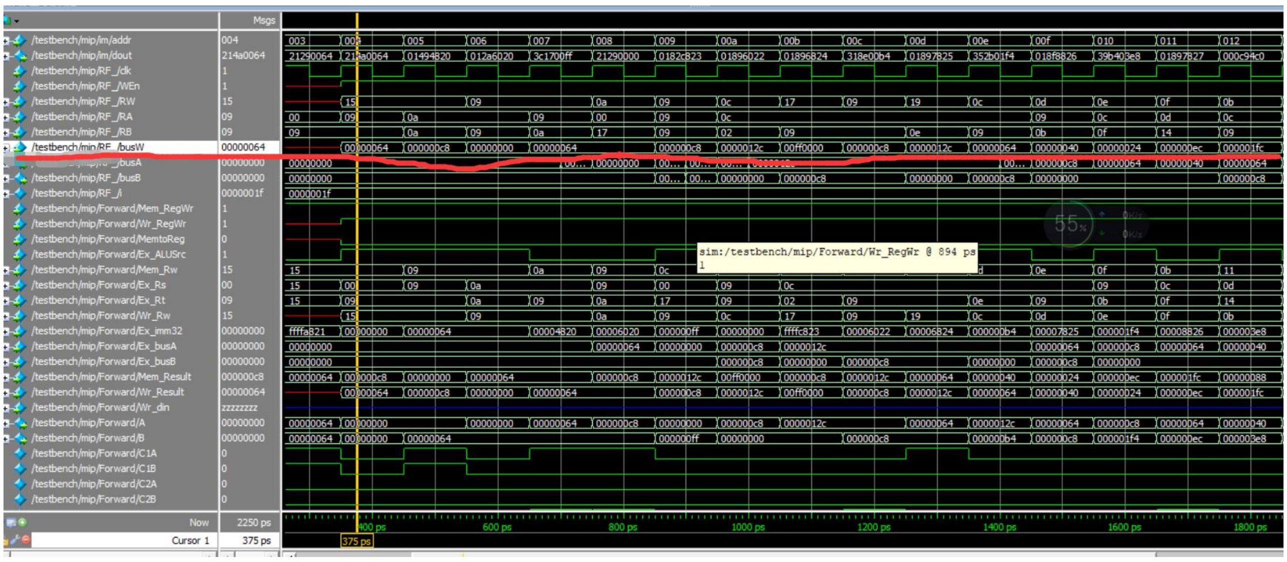
具体测试代码：test.asm

```
he:             # 写寄存器编号     写的数据           （16进制）
addiu $s5,$0,100      # Rw_number: 15    busW_data: 64
addu  $s5,$s5,$s5     # Rw_number: 15    busW_data: c8
addi $t1,$0,0         # Rw_number: 09    busW_data: 0
addi $t1,$t1,100      # Rw_number: 09    busW_data: 64
addi $t2,$t2,100      # Rw_number: 0a    busW_data: 64
add $t1,$t2,$t1       # Rw_number: 09    busW_data: c8    7th CLK to EX阶段
add $t4,$t1,$t2       # Rw_number: 0c    busW_data: 12c
lui $s7,255           # Rw_number: 23    busW_data: 00ff0000
```

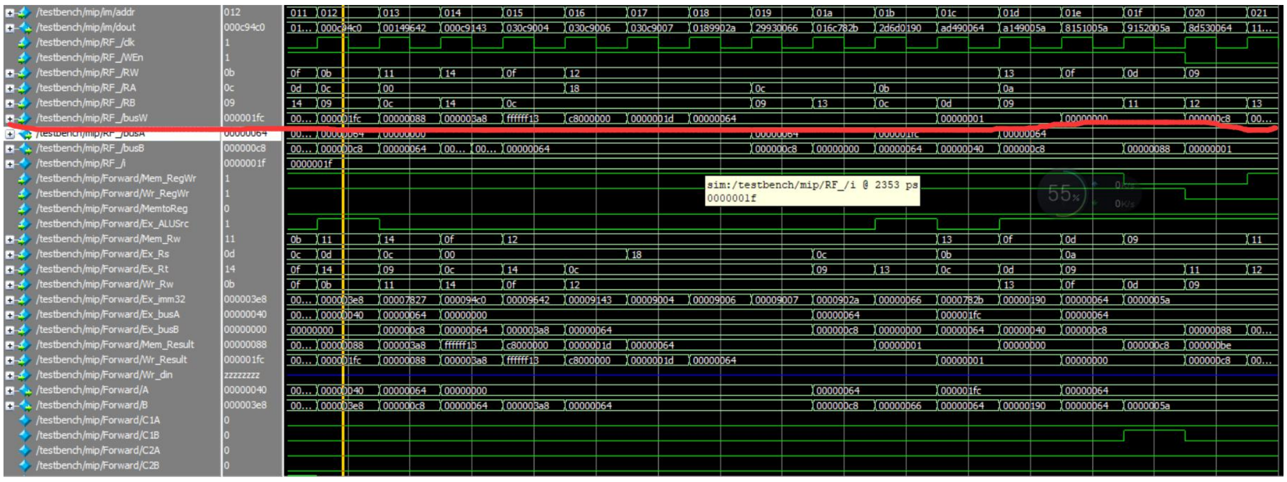
```
label:
addi $t1,$t1,0        # Rw_number: 09    busW_data: c8
subu $t9,$t4,$2       # Rw_number: 19    busW_data: 12c
sub $t4,$t4,$t1       # Rw_number: 0c    busW_data: 64   第10条指令
#bltz $t4,he #10
ww:
and $t5,$t4,$t1       # Rw_number: 0d    busW_data: 40
andi $t6,$t4,180      # Rw_number: 0e    busW_data: 24
or  $t7,$t4,$t1       # Rw_number: 0f    busW_data: ec
ori $t3,$t1,500       # Rw_number: 0b    busW_data: 1fc
xor $s1,$t4,$t7       # Rw_number: 11    busW_data: 88
xori $s4,$t5,1000     # Rw_number: 14    busW_data: 3a8
nor $t7,$t4,$t1       # Rw_number: 0f    busW_data: ffffff13
sll $s2,$t4,19        # Rw_number: 12    busW_data: 0320000 error
srl $s2,$s4,25        # Rw_number: 12    busW_data: 0 error
sra $s2,$t4,5         # Rw_number: 12    busW_data: 3 error
sllv $s2,$t4,$t8      # Rw_number: 12    busW_data: 64
srlv $s2,$t4,$t8      # Rw_number: 12    busW_data: 64
srav $s2,$t4,$t8      # Rw_number: 12    busW_data: 64
slt $s2,$t4,$t1       # Rw_number: 12    busW_data: 1
slti $s3,$t4,102      # Rw_number: 13    busW_data: 1
sltu $t7,$t3,$t4      # Rw_number: 0f    busW_data: 0
sltiu $t5,$t3,400     # Rw_number: 0d    busW_data: 0
#bgtz $s3,label #16
sw  $t1,100($t2)      # Datamem: c8     data: c8
lw  $s3,100($t2)      # Rw_number: 13    busW_data: c8
beq $t4,$t2,label     #第一次会跳转至label          第20条指令
j he                  #地址转为0
```

4.3 测试仿真结果

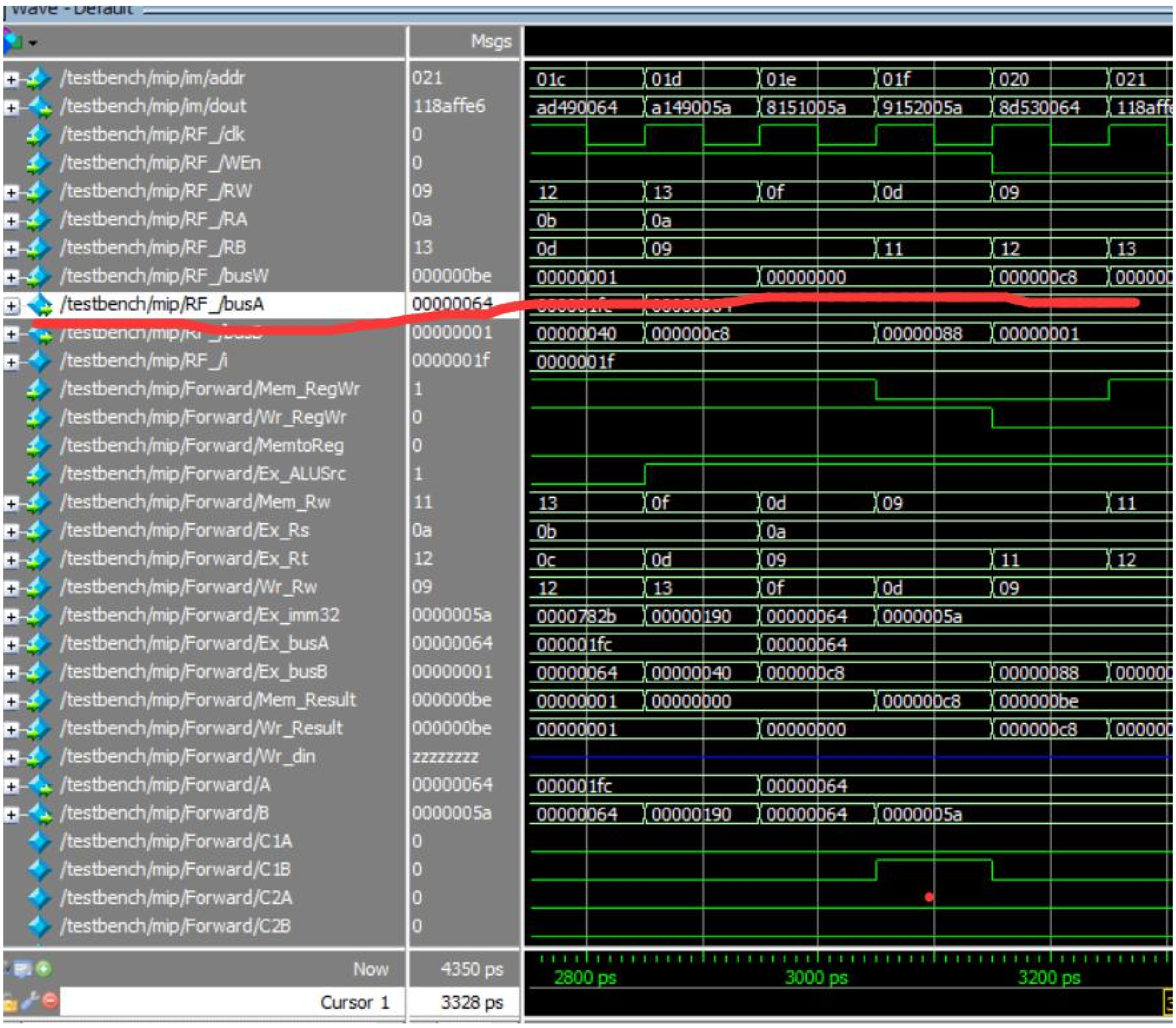
像我写的汇编代码里面的注释那样的，我们只需要通过观测寄存器内数据的变化或者从Regfile的BusW是否与我们预期的一样，则可以说明代码的正确性。



```
                                              Msgs
/testbench/mip/im/addr                        004
/testbench/mip/im/dout                        214a0064
/testbench/mip/RF_/clk                        1
/testbench/mip/RF_/WEn                        1
/testbench/mip/RF_/RW                         15
/testbench/mip/RF_/RA                         09
/testbench/mip/RF_/RB                         09
/testbench/mip/RF_/busW                       00000064
/testbench/mip/RF_/busA                       00000000
/testbench/mip/RF_/busB                       00000000
/testbench/mip/RF_/i                          0000001f
/testbench/mip/Forward/Mem_RegWr              1
/testbench/mip/Forward/Wr_RegWr               1
/testbench/mip/Forward/MemtoReg               0
/testbench/mip/Forward/Ex_ALUSrc              1
/testbench/mip/Forward/Mem_Rw                 15
/testbench/mip/Forward/Ex_Rs                  00
/testbench/mip/Forward/Ex_Rt                  09
/testbench/mip/Forward/Wr_Rw                  15
/testbench/mip/Forward/Ex_imm32               00000000
/testbench/mip/Forward/Ex_busA                00000000
/testbench/mip/Forward/Ex_busB                00000000
/testbench/mip/Forward/Mem_Result             000000c8
/testbench/mip/Forward/Wr_Result              00000064
/testbench/mip/Forward/Wr_din                 zzzzzzzz
/testbench/mip/Forward/A                      00000000
/testbench/mip/Forward/B                      00000000
/testbench/mip/Forward/C1A                    0
/testbench/mip/Forward/C1B                    0
/testbench/mip/Forward/C2A                    0
/testbench/mip/Forward/C2B                    0
                                     Now      2250 ps
                                Cursor 1      375 ps

sim:/testbench/mip/Forward/Wr_RegWr @ 894 ps
1
```



```
/testbench/mip/im/addr                        012
/testbench/mip/im/dout                        000c94c0
/testbench/mip/RF_/clk                        1
/testbench/mip/RF_/WEn                        1
/testbench/mip/RF_/RW                         0b
/testbench/mip/RF_/RA                         0c
/testbench/mip/RF_/RB                         09
/testbench/mip/RF_/busW                       000001fc
/testbench/mip/RF_/busA                       00000064
/testbench/mip/RF_/busB                       000000c8
/testbench/mip/RF_/i                          0000001f
/testbench/mip/Forward/Mem_RegWr              1
/testbench/mip/Forward/Wr_RegWr               1
/testbench/mip/Forward/MemtoReg               0
/testbench/mip/Forward/Ex_ALUSrc              1
/testbench/mip/Forward/Mem_Rw                 11
/testbench/mip/Forward/Ex_Rs                  0d
/testbench/mip/Forward/Ex_Rt                  14
/testbench/mip/Forward/Wr_Rw                  0b
/testbench/mip/Forward/Ex_imm32               000003e8
/testbench/mip/Forward/Ex_busA                00000040
/testbench/mip/Forward/Ex_busB                00000000
/testbench/mip/Forward/Mem_Result             00000088
/testbench/mip/Forward/Wr_Result              000001fc
/testbench/mip/Forward/Wr_din                 zzzzzzzz
/testbench/mip/Forward/A                      00000040
/testbench/mip/Forward/B                      000003e8
/testbench/mip/Forward/C1A                    0
/testbench/mip/Forward/C1B                    0
/testbench/mip/Forward/C2A                    0
/testbench/mip/Forward/C2B                    0

sim:/testbench/mip/RF_/i @ 2353 ps
0000001f
```



```
Wave - Default
                                     Msgs
/testbench/mip/im/addr               021
/testbench/mip/im/dout               118affe6
/testbench/mip/RF_/clk               0
/testbench/mip/RF_/WEn               0
/testbench/mip/RF_/RW                09
/testbench/mip/RF_/RA                0a
/testbench/mip/RF_/RB                13
/testbench/mip/RF_/busW              000000be
/testbench/mip/RF_/busA              00000064
/testbench/mip/RF_/busB              00000001
/testbench/mip/RF_/i                 0000001f
/testbench/mip/Forward/Mem_RegWr     1
/testbench/mip/Forward/Wr_RegWr      0
/testbench/mip/Forward/MemtoReg      0
/testbench/mip/Forward/Ex_ALUSrc     1
/testbench/mip/Forward/Mem_Rw        11
/testbench/mip/Forward/Ex_Rs         0a
/testbench/mip/Forward/Ex_Rt         12
/testbench/mip/Forward/Wr_Rw         09
/testbench/mip/Forward/Ex_imm32      0000005a
/testbench/mip/Forward/Ex_busA       00000064
/testbench/mip/Forward/Ex_busB       00000001
/testbench/mip/Forward/Mem_Result    000000be
/testbench/mip/Forward/Wr_Result     000000be
/testbench/mip/Forward/Wr_din        zzzzzzzz
/testbench/mip/Forward/A             00000064
/testbench/mip/Forward/B             0000005a
/testbench/mip/Forward/C1A           0
/testbench/mip/Forward/C1B           0
/testbench/mip/Forward/C2A           0
/testbench/mip/Forward/C2B           0
                          Now        4350 ps
                          Cursor 1   3328 ps
```

通过观测busW的值与mars里面得到正确的值进行比较，可以得出以下：

指令add,addu,addi,addiu,subu,sub,lui 7

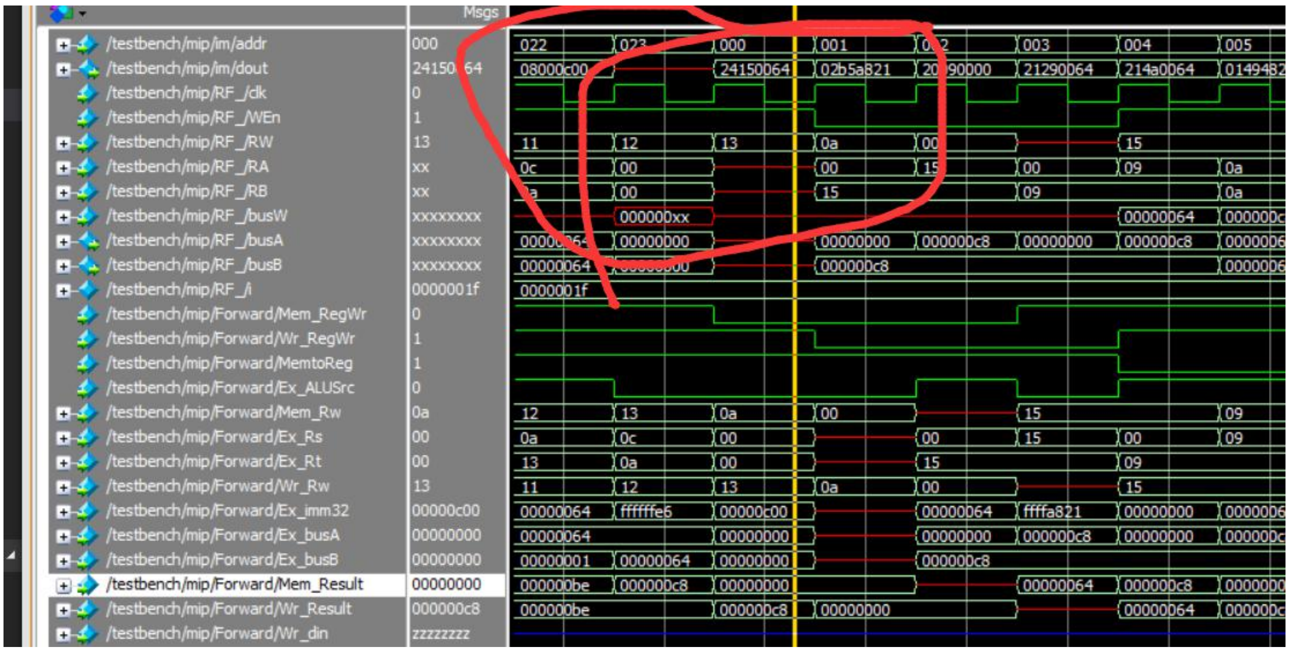
and,andi,or,ori,xor,xori,nor, 7

slt,slti,sltu,sltiu, 4

lw,lb,lbu, 3

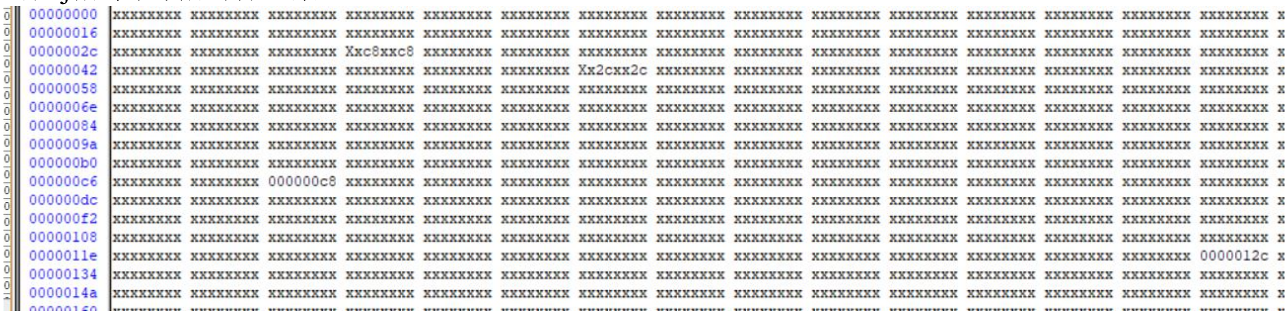
sll,sra,sllv,srav,srlv,srl,6

此处共计27条；

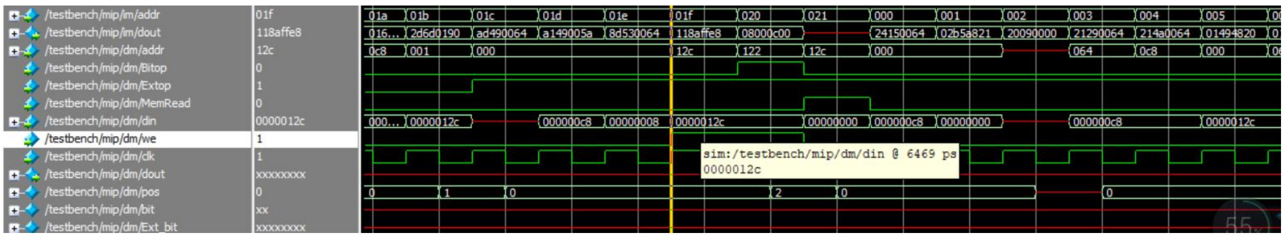


```
/testbench/mip/im/addr                000
/testbench/mip/im/dout                24150064
/testbench/mip/RF_/clk                0
/testbench/mip/RF_/WEn                1
/testbench/mip/RF_/RW                 13
/testbench/mip/RF_/RA                 xx
/testbench/mip/RF_/RB                 xx
/testbench/mip/RF_/busW               xxxxxxxx
/testbench/mip/RF_/busA               xxxxxxxx
/testbench/mip/RF_/busB               xxxxxxxx
/testbench/mip/RF_/i                  0000001f
/testbench/mip/Forward/Mem_RegWr      0
/testbench/mip/Forward/Wr_RegWr       1
/testbench/mip/Forward/MemtoReg       1
/testbench/mip/Forward/Ex_ALUSrc      0
/testbench/mip/Forward/Mem_Rw         0a
/testbench/mip/Forward/Ex_Rs          00
/testbench/mip/Forward/Ex_Rt          00
/testbench/mip/Forward/Wr_Rw          13
/testbench/mip/Forward/Ex_imm32       00000c00
/testbench/mip/Forward/Ex_busA        00000000
/testbench/mip/Forward/Ex_busB        00000000
/testbench/mip/Forward/Mem_Result     00000000
/testbench/mip/Forward/Wr_Result      000000c8
/testbench/mip/Forward/Wr_din         zzzzzzzz
```

可知j指令跳转成功；1条



```
00000000 xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx
00000016 xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx
0000002c xxxxxxxx xxxxxxxx xxxxxxxx Xxc8xxc8 xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx
00000042 xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx Xx2cxx2c xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx
00000058 xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx
0000006e xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx
00000084 xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx
0000009a xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx
000000b0 xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx
000000c6 xxxxxxxx xxxxxxxx 000000c8 xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx
000000dc xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx
000000f2 xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx
00000108 xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx
0000011e xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx 0000012c
00000134 xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx
0000014a xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx
```



```
/testbench/mip/im/addr       01f
/testbench/mip/im/dout       118affe8
/testbench/mip/dm/addr       12c
/testbench/mip/dm/Bitop      0
/testbench/mip/dm/Extop      1
/testbench/mip/dm/MemRead    0
/testbench/mip/dm/din        0000012c
/testbench/mip/dm/we         1
/testbench/mip/dm/clk        1
/testbench/mip/dm/dout       xxxxxxxx
/testbench/mip/dm/pos        0
/testbench/mip/dm/bit        xx
/testbench/mip/dm/Ext_bit    xxxxxxxx

sim:/testbench/mip/dm/din @ 6469 ps
0000012c
```

通过观测DM内存的数据可以发现，sw装载字和sb装载字节的正确性；（2条）
由于jr和jal单周期的时候也没写出来，所以没能预测，不知道对不对。

实现指令如下：

add， addu, addi, addiu,sub,subu, slt, slti,sltu,sltiu,and, nor, or, xor, andi, ori, xori,sll, srl,sllv, sra, srav, srlv, beq, bne, bgez, bgtz, blez, bltz, lw, sw, lui, j, lb, lbu, sb,

其中分支指令部分未能预测完全；共计在32条。

待验证：jalr, jr,

5. 心得体会

奇怪的是，写到这里意味着课设终于要结束了，不用在过着每天写到夜里两三点的困乏，有一种如释重负的感觉同时又有一丝丝的失落。与之前写C、C++和数据结构的课设不同的是，对于那些语言你是可以轻而易举的知道并且解决掉bug的，然而计算机组成原理课设在愉快的假期到来之前，可算是给我一个难忘的煎熬。是的，代码很快就写出来了，但是太多太多的bug了，而且处理冒险问题很让人头疼，别以为自己起的变量名字自己就能不搞乱。试试一个模块几十个

变量，然后又十几个模块，各个模块还不是独立的。不单是要求理论知识达到水准，动手实践的时候才能真正体现出是否掌握，大家都知道这发生数据冒险了需要转发到那去。可是，如何实现？如何一同的实现一类冒险，这可不是理论上那么简单的事情，而且还要尽量让自己的硬件模块是合理真实存在的。不得不说，这是一道坎。

因为ModelSim代码编辑不是特别的好用，所以使用了VScode进行编写，风格很清爽，而且相同的变量名选择时候回全部发亮，容易查找一些大小写错误，还有可以全局替代特定的变量，特别好；大大提高了工作效率。还有遇到了好多问题不会处理的时候，查阅资料和网上索引，以及向陈班长请教，总会得到正确的解决方案，交谈过程中收获不少。

通过这次课程设计，深有体会的一点是，一步一个脚印，慢慢地耐心地解决问题，不要浮躁。基本上我是这样完成的，先把各个模块写出来，在原来的单周期上进行修改，并且参照给的课设样例里面的内容。把那些提示的bug统统解决，还有通过VScode编辑器检查变量是否一致，不能跌在这种小错误上。暂时地，先不用去考虑带冒险的流水线，先把流水线写出来，认为的书写无冒险的流水线，看看正确吗？其次才是去考虑带冒险问题。先实现那些简单的R型或I型指令想一些跳转等指令先不考虑，定个小目标25条以上，然后再去解决，虽然最后可能没有36条，但是理论上是知道数据通路如何的，但是，哎有点复杂。

不逼自己，永远都不知道自身的潜力到底有多么的大；不熬夜到两三点，永远都不知道一天能有这么长。其实，回去审视自己这么认真努力地完成课设，不仅仅是一种自豪，更重要的是为又一次成功的坚守底线儿感到欣慰。是的，可能这份课设还存在一些我没能及时发现并且解决的bug，因为最后时间实在是不够了，有点后悔为什么不早点开写课设，可能也是最后考试太多了。但是，遗憾终究是会有的，能让我感到安慰的是真实的付出和认真对待。

参 考 文 献

[1]袁春风，等. 计算机组成与系统结构[M]. 第2 版，北京：清华大学出版社，2010.
[2]MIPS32® Architecture For Programmers Volume II: The MIPS32® Instruction Set[M]. Revision 2.50, Mountain View, CA: MIPS Technologies Inc., July 1, 2005.
[3]龙芯MIPS指令集及汇编程序设计

附　录

汇编指令	结果描述	机器指令的机器码	
		16进制	二进制
addiu $1, $0,#1	[$1] = 0000_0001H	24010001	0010_0100_0000_0001_0000_0000_0000_0001
sll $2, $1,#4	[$2] = 0000_0010H	00011100	0000_0000_0000_0001_0001_0001_0000_0000
addu $3, $2,$1	[$3] = 0000_0011H	00411821	0000_0000_0100_0001_0001_1000_0010_0001
srl $4, $2,#2	[$4] = 0000_0004H	00022082	0000_0000_0000_0010_0010_0000_1000_0010
slti $25,$4,#5	[$25] = 0000_0001H	28990005	0010_1000_1001_1001_0000_0000_0000_0101
bgez $25,#16	跳转到54H	07210010	0000_0111_0010_0001_0000_0000_0001_0000
subu $5, $3,$4	[$5] = 0000_000DH	00642823	0000_0000_0110_0100_0010_1000_0010_0011
sw $5, #20($0)	Mem[0000_0014H] = 0000_000DH	AC050014	1010_1100_0000_0101_0000_0000_0001_0100
nor $6, $5,$2	[$6] = FFFF_FFE2H	00A23027	0000_0000_1010_0010_0011_0000_0010_0111

or $7, $6,$3	[$7] = FFFF_FFF3H	00C33825	0000_0000_1100_0011_0011_1000_0010_0101
xor $8, $7,$6	[$8] = 0000_0011H	00E64026	0000_0000_1110_0110_0100_0000_0010_0110
sw $8, #28($0)	Mem[0000_001CH] = 0000_0011H	AC08001C	1010_1100_0000_1000_0000_0000_0001_1100
beq $8, $3,#2	跳转到38H	11030002	0001_0001_0000_0011_0000_0000_0000_0010
slt $9, $6,$7	不执行	00C7482A	0000_0000_1100_0111_0100_1000_0010_1010
addiu $1, $0,#8	[$1] = 0000_0008H	24010008	0010_0100_0000_0001_0000_0000_0000_1000
lw $10,#20($1)	[$10] = 0000_0011H	8C2A0014	1000_1100_0010_1010_0000_0000_0001_0100
bne $10,$5,#4	跳转到50H	15450004	0001_0101_0100_0101_0000_0000_0000_0100
and $11,$2,$1	不执行	00415824	0000_0000_0100_0001_0101_1000_0010_0100
sw $11,#28($1)	不执行	AC2B001C	1010_1100_0010_1011_0000_0000_0001_1100
sw $4, #16($1)	不执行	AC240010	1010_1100_0010_0100_0000_0000_0001_0000
jal #25	跳转到64H, [$31] = 0000_0054H	0C000019	0000_1100_0000_0000_0000_0000_0001_1001
lui $12,#12	[$12] = 000C_0000H	3C0C000C	0011_1100_0000_1100_0000_0000_0000_1100
srav $26,$12,$2	[$26] = 0000_000CH	004CD007	0000_0000_0100_1100_1101_0000_0000_0111
sllv $27,$26,$1	[$27] = 0000_0018H	003AD804	0000_0000_0011_1010_1101_1000_0000_0100
jalr $27	跳转到18H , [$31] = 0000_0064H	0360F809	0000_0011_0110_0000_1111_1000_0000_1001
sb $26,#5($3)	MEM[0000_0016H] = 000C_000DH	A07A0005	1010_0000_0111_1010_0000_0000_0000_0101
sltu $13,$3,$3	[$13] = 0000_0000H	0063682B	0000_0000_0110_0011_0110_1000_0010_1011
bgtz $13,#3	不跳转	1DA00003	0001_1101_1010_0000_0000_0000_0000_0011
sllv $14,$6,$4	[$14] =FFFF_FE20H	00867004	0000_0000_1000_0110_0111_0000_0000_0100
sra $15,$14,#2	[$15] =FFFF_FF88H	000E7883	0000_0000_0000_1110_0111_1000_1000_0011
srlv $16,$15,$1	[$16] =00FF_FFFFH	002F8006	0000_0000_0010_1111_1000_0000_0000_0110
blez $16,#8	不跳转	1A000008	0001_1010_0000_0000_0000_0000_0000_1000
srav $16,$15,$1	[$16] =FFFF_FFFFH	002F8007	0000_0000_0010_1111_1000_0000_0000_0111
addiu $11,$0,#140	[$11] = 0000_008CH	240B008C	0010_0100_0000_1011_0000_0000_1000_1100
bltz $16, #6	跳转到A0H	06000006	0000_0110_0000_0000_0000_0000_0000_0110
lw $28,#3($10)	[$28] = 000C_000DH /000C_880DH	8D5C0003	1000_1101_0101_1100_0000_0000_0000_0011
bne $28,$29,#7	不跳转/跳转ACH	179D0007	0001_0111_1001_1101_0000_0000_0000_0111
sb $15,#8($5)	Mem[0000_0015H] = 0000_0088H	A0AF0008	1010_0000_1010_1111_0000_0000_0000_1000
lb $18,#8($5)	[$18] =FFFF_FF88H	80B20008	1000_0000_1011_0010_0000_0000_0000_1000
lbu $19,#8($5)	[$19] = 0000_0088H	90B30008	1001_0000_1011_0011_0000_0000_0000_1000
sltiu $24,$15,#0xFFFF	[$24] = 0000_0001H	2DF8FFFF	0010_1101_1111_1000_1111_1111_1111_1111
or $29,$12,$5	[$29] = 000C000DH	0185E825	0000_0001_1000_0101_1110_1000_0010_0101
jr $11	跳转指令8CH	01600008	0000_0001_0110_0000_0000_0000_0000_1000
andi $20,$15,#0xFFFF	[$20] = 0000_FF88H	31F4FFFF	0011_0001_1111_0100_1111_1111_1111_1111
ori $21,$15,#0xFFFF	[$21] =FFFF_FFFFH	35F5FFFF	0011_0101_1111_0101_1111_1111_1111_1111
xori $22,$15,#0xFFFF	[$22] = FFFF_0077H	39F6FFFF	0011_1001_1111_0110_1111_1111_1111_1111
j #00H	跳转指令00H	08000000	0000_1000_0000_0000_0000_0000_0000_0000

自写的测试代码：

```
24150064
02b5a821
20090000
21290064
214a0064
01494820
012a6020
3c1700ff
21290000
```

```
0182c823
01896022
01896824
318e00b4
01897825
352b01f4
018f8826
39b403e8
01897827
000c94c0
00149642
000c9143
030c9004
030c9006
030c9007
0189902a
29930066
016c782b
2d6d0190
ad490064
8d530064
118affe9
08000c00
```