**实验9报告**

蔡润泽、付轶凡

箱子号：45

一、实验任务（10%）

在实验八的 CPU 代码基础上，完成:

a) CPU 增加 BREAK 指令，也就是增加 break 例外支持。

b) CPU 增加地址错、整数溢出、保留指令例外支持。

c) CPU 增加 CP0 寄存器 COUNT、COMPARE、BADVADDR。

d) CPU 增加时钟中断支持，时钟中断要求固定绑定在硬件中断 5 号上，也就是 CAUSE 对应的 IP7 上。

e) CPU 增加 6 个硬件中断支持，编号为 0~5，对应 CAUSE 的 IP7~IP2。

f) CPU 增加 2 个软件中断支持，对应 CAUSE 的 IP1~IP0。

g) 完成 lab9 功能测试。

h) 推荐在 lab9 的 myCPU 上运行记忆游戏程序，并正确运行。

二、实验设计（40%）

（一）总体设计思路

硬件设计图如下：

**图片包含 地图, 屏幕截图

描述已自动生成**

图1.1 硬件结构设计图

如图1.2的流水示意图，在代码设计中，主要有9个部分，包括五级流水、ALU、寄存器堆，HI/LO协处理器（用来储存乘法指令的高32位、低32位，以及除法指令的余数、商），以及cp0寄存器。该设计使用了四个IP，Inst\_RAM和Data\_RAM（采用同步RAM），mydiv和mydivu（分别计算有符号、无符号除法）。并且采用Tools模块进行译码。此外，WB模块与cp0寄存器相连形成了数据通路（cp0寄存器代码置于wb\_stage模块中）。

图片包含 天空

描述已自动生成

图1.2 流水示意图

（二）重要模块1设计：算数逻辑单元（ALU）模块

1. 工作原理

将CPU中的运算处理进行模块化，方便外界调用。同时模块化的ALU设计便于在其中增加新的运算功能，提高代码的扩展性。在本次实验中，加入了alu\_overflow的检测变量，并将结果输出，其判断代码如下：

assign alu\_overflow = (adder\_result[32]!=adder\_result[31]);【采用33位加法】

1. 接口定义

input [15:0] alu\_op, //输入运算符

input [31:0] alu\_src1, //输入数据1

input [31:0] alu\_src2, //输入数据2

output [31:0] alu\_result, //输出结果

output [31:0] alu\_hi\_result , //输出存入hi寄存器的结果

output [31:0] alu\_lo\_result //输出存入lo寄存器的结果

output alu\_overflow

1. 功能描述

采用16位的独热码对ALU进行控制，根据独热码进行16项（相较之前的实验增添了有符号、无符号乘除法）不同的算数逻辑运算操作，并将结果传回给exe阶段。并输出overflow标志位。

（三）重要模块2设计：寄存器堆（Reg\_File）模块

1. 工作原理

将32个32位宽的寄存器堆模块化，以实现两读一写，同步读异步写的操作。

1. 接口定义

input clk,

// READ PORT 1

input [ 4:0] raddr1,

output [31:0] rdata1,

// READ PORT 2

input [ 4:0] raddr2,

output [31:0] rdata2,

// WRITE PORT

input we, //write enable, HIGH valid

input [ 4:0] waddr,

input [31:0] wdata

1. 功能描述

当写使能信号为1时，在写回阶段对寄存器堆进行写入。同时，对于两个读端口信号，进行异步读取，将输出结果传递给ID阶段。

（四）重要模块3设计：取指阶段（IF\_stage）模块

1. 工作原理

将取指操作模块化，IF从inst\_ram中读出指令，并且在该周期模块之内处理PC的值，并且将指令传递给bus传递给ID模块，ID模块在下一周期再接收 。在IF阶段判断是否出现PC地址错误例外，若有，则将fs\_exc\_type例外类型标识进行修改，代码如下：

assign fs\_exc\_type = (fs\_pc[1:0]!=2'd0)?`ADEL\_IF:0;

另外，fs\_pc的取指也根据WB阶段传回的信号，增加了两种选择的可能，即跳转到中断入口地址和处理eret，修改后的代码如下：

1. always @(posedge clk) begin
2. **if** (reset) begin
3. fs\_pc <= 32'hbfbffffc;  //trick: to make nextpc be 0xbfc00000 during reset
4. end
5. **else** **if** (to\_fs\_valid && ws\_ex  )  begin
6. fs\_pc <= 32'hbfc0037c;
7. end
8. **else** **if** (to\_fs\_valid && eret\_flush) begin
9. fs\_pc <= cp0\_epc-4;
10. end
11. **else** **if** (to\_fs\_valid && fs\_allowin) begin
12. fs\_pc <= nextpc;
13. end
14. end
15. 接口定义

表2.1 IF\_stage接口定义

| **名称** | **方向** | **位宽** | **功能描述** |
| --- | --- | --- | --- |
| clk | IN | 1 | 时钟信号 |
| reset | IN | 1 | 复位信号 |
| ds\_allowin | IN | 1 | ID模块允许接受IF传值 |
| br\_bus | IN | 33 | 输入是否跳转和branch的target |
| ws\_to\_fs\_bus | IN | 34 | 将WB阶段的ws\_ex,eret\_flush\_cp0\_epc传递给IF阶段 |
| fs\_to\_ds\_valid | OUT | 1 | IF模块可以向ID模块传值 |
| fs\_to\_ds\_bus | OUT | 68 | IF模块向ID模块传递数据（指令码和地址） |
| inst\_sram\_en | OUT | 1 | Inst\_sram读使能 |
| inst\_sram\_wen | OUT | 4 | Inst\_sram写使能，此处恒为0 |
| inst\_sram\_addr | OUT | 32 | Inst\_sram目标地址 |
| inst\_sram\_wdata | OUT | 32 | Inst\_sram写数据 |
| inst\_sram\_rdata | IN | 32 | Inst\_sram读数据 |

1. 功能描述

PC在收到reset信号时设为偏移量32'hbfbffffc，并且在该周期模块之内处理PC的值，PC值的变化根据br\_bus取出来决定是否跳转还是加4。IF模块当inst\_sram\_en读使能信号为1时，将处理后的next\_pc作为地址传递给inst\_sram, 并从inst\_ram中读出指令。IF模块将取出的指令和地址传递给ID模块，ID模块在下一周期再接收 。在IF阶段判断是否出现PC地址错误例外。fs\_pc的取指也根据WB阶段传回的信号，增加了两种选择的可能，即跳转到中断入口地址和处理eret。

（五）重要模块4设计：译码阶段（ID\_stage）模块

1. 工作原理

将从IF模块获取的指令进行译码，获得指令格式类型、ALU操作类型、是否需要加载、写回内存和参与运算数据的值、是否需要加载协处理器cp0的hi和lo寄存器、是否需要写回协处理器cp0的hi和lo寄存器、跳转的目标PC。判断PC是否需要跳转，并将结果返还给IF模块。将译码后的数据和控制信号传递给EXE模块,EXE模块在下一时钟周期接受。另外，写回阶段的数据也通过该模块传递给寄存器堆。

CPU数据通路增加旁路设计，来让前面的指令直接把已经生成出来的结果直接转给后面的指令。在本设计中，采用了“流水级组合逻辑的结果传递到译码级寄存器读出处”的方案。并通过后续阶段的valid信号和gr\_we信号来控制ID模块中，rs\_value和rt\_value的值。

另外对于ready\_go信号，当译码级的指令和处在执行级的LW指令相关时，需要设置成“0”。

在ID译码级需要标记三种类型的例外：无效指令、系统调用以及BREAK，并将其传给后续阶段。

1. 接口定义

表2.2 ID\_stage接口定义

| **名称** | **方向** | **位宽** | **功能描述** |
| --- | --- | --- | --- |
| clk | IN | 1 | 时钟信号 |
| reset | IN | 1 | 复位信号 |
| es\_allowin | IN | 1 | EXE模块允许接受ID传值 |
| ds\_allowin | OUT | 1 | 允许IF模块向ID模块传递数据 |
| fs\_to\_ds\_valid | IN | 1 | IF模块可以向ID模块传值 |
| fs\_to\_ds\_bus | IN | 68 | IF模块向ID模块传递数据（指令码及地址） |
| ds\_to\_es\_valid | OUT | 1 | 允许ID模块向ES模块传递数据 |
| ds\_to\_es\_bus | OUT | 168 | ID模块向EXE模块传递数据 |
| br\_bus | OUT | 33 | 输出是否跳转和branch的target给IF模块 |
| ws\_to\_rf\_bus | IN | 43 | WB模块向ID模块传递的需要写回REG FILE的信息 |
| es\_to\_ms\_bus | IN | 154 | EXE模块向MEM模块传递数据 |
| ms\_to\_ws\_bus | IN | 118 | MEM模块向WB模块传递数据 |
| es\_to\_ms\_valid | IN | 1 | EXE模块可以向MEM模块传值 |
| ms\_to\_ws\_valid | IN | 1 | MEM模块可以向WB模块传值 |
| out\_es\_valid | IN | 1 | 接收es\_valid是否为1 |
| out\_ms\_valid | IN | 1 | 接收ms\_valid是否为1 |

1. 功能描述

将从IF模块获取的指令进行译码，获得指令格式类型、ALU操作类型、是否需要加载、写回内存和参与运算数据的值、是否需要加载协处理器cp0的hi和lo寄存器、是否需要写回协处理器cp0的hi和lo寄存器、跳转的目标PC。判断PC是否需要跳转，并将结果返还给IF模块。将译码后的数据和控制信号在传递给EXE模块,EXE模块在下一时钟周期接受。写回阶段的数据也通过该模块传递给寄存器堆。并且采用前递的方式来减少CPU的阻塞，缩短运行时间。在ID译码级需要标记三种类型的例外：无效指令、系统调用以及BREAK，并将其传给后续阶段。

（六）重要模块5设计：执行阶段（EXE\_stage）模块

1. 工作原理

将从ID模块获取的指令相应的执行。将执行后的ALU结果和前阶段传递的通用寄存器写使能、写地址控制信号、PC传通过总线传递给MEM模块,MEM模块在下一周期接收新值。另外，load指令的发出读信号处理也在EXE阶段完成，EXE模块将数据传递给MEM模块，在下一周期WB阶段进行写回。输出数据RAM的写信号和数据。

此阶段在进行算数逻辑运算时，需要调用ALU模块。

除了对ALU模块需要新增有符号、无符号乘法外，还需要在此阶段调用我们生成的mydiv和mydivu两个除法器IP完成新增的有符号、无符号除法运算，以及对除法器中的控制信号进行相应的设置。

EXE阶段需要标记OVERFLOW例外和内存写地址错误例外，并将例外标记传到后续阶段，例外标记信号代码如下：

assign OV\_EXE = es\_inst\_ov & es\_alu\_overflow ; //es\_inst\_ov由ID传来，为inst\_add | inst\_addi | inst\_sub

assign es\_exc\_type = (ds\_exc\_type==4'd0 && OV\_EXE )? `OV:

(ds\_exc\_type==4'd0 && ADES\_EXE)? `ADES: ds\_exc\_type;

assign ADES\_EXE = (es\_store\_type==`SW\_TYPE && es\_vaddr[1:0]!=2'b00)?1:

(es\_store\_type==`SH\_TYPE && es\_vaddr[0] !=1'b0) ?1:

0;

1. 接口定义

表2.3 EXE\_stage接口定义

| **名称** | **方向** | **位宽** | **功能描述** |
| --- | --- | --- | --- |
| clk | IN | 1 | 时钟信号 |
| reset | IN | 1 | 复位信号 |
| ms\_allowin | IN | 1 | MEM模块允许接受EXE传值 |
| es\_allowin | OUT | 1 | EXE模块允许接受ID传值 |
| ds\_to\_es\_valid | IN | 1 | ID模块可以向EXE模块传值 |
| ds\_to\_es\_bus | IN | 168 | ID模块向EXE模块传递数据 |
| es\_to\_ms\_valid | OUT | 1 | EXE模块可以向MEM模块传值 |
| es\_to\_ms\_bus | OUT | 154 | EXE模块向MEM模块传递数据 |
| data\_sram\_en | OUT | 1 | data\_sram读使能 |
| data\_sram\_wen | OUT | 4 | data\_sram写使能 |
| data\_sram\_addr | OUT | 32 | data\_sram目标地址 |
| data\_sram\_wdata | OUT | 32 | data\_sram写数据 |
| out\_es\_valid | OUT | 1 | 将es\_valid的值传递给ID模块 |
| ws\_ex | IN | 1 | WB阶段有例外标记 |
| ms\_exc\_type | IN | 4 | MEM阶段例外类型 |
| ws\_exc\_type | IN | 4 | WB阶段里外类型 |
| ms\_eret | IN | 1 | MEM阶段有eret指令 |
| eret\_flush | IN | 1 | eret刷新流水线信号 |

mydiv和mydivu除法器IP数据通道定义如下：

s\_axis\_divisor\_tdata[31:0]; //除数

s\_axis\_divisor\_tready; //除数应答信号

s\_axis\_divisor\_tvalid; //除数请求信号

s\_axis\_dividend\_tdata[31:0]; //被除数

s\_axis\_dividend\_tready; //被除数应答信号

s\_axis\_dividend\_tvalid; //被除数请求信号

m\_axis\_dout\_tdata[63:0]; //商和余数，其中高32位为余数、低32位为商

m\_axis\_dout\_tvalid; //结果有效信号，表示除法已经算完，可以取得结果

aclk; //时钟信号

在除法指令处于执行流水级且没有对除法器成功输入数据的时候，同时将 s\_axis\_dividend\_tvalid 和s\_axis\_divisor\_tvalid 置为 1，向除法器IP发送调用除法器的请求。当发现s\_axis\_dividend\_tready 和 s\_axis\_divisor\_tready 反馈为 1 后，tvalid和tready成功握手，并需要在下一拍将s\_axis\_dividend\_tvalid和s\_axis\_di vior\_tvalid清为0，以此保证一个除法操作只调用一次除法器IP，避免除法器给CPU送多个结果从而导致出错。成功握手后，数据传入除法器的各个数据通路，8拍后，除法器产生结果并将m\_axis\_dout\_tvalid信号置1，表示可以取得除法结果。

为了避免除法指令与之后的指令产生“写后读”相关，考虑到除法器IP需要8拍才能拿到结果，所以如果执行流水级这一拍执行的是除法指令，需要将执行级阻塞住（即修改es\_ready\_go信号），拿到除法结果后再释放，因为通常程序中的除法指令较少，这样的阻塞设计不会太影响整体的性能。

1. 功能描述

将从ID模块获取的指令相应的执行。将执行后和前阶段传递的数据控制信号通过数据总线在下一时钟周期更新传给MEM模块。另外，load指令的发出读信号处理也在EXE阶段完成，EXE模块将数据传递给MEM模块，在下一周期进行写回。输出数据RAM的写信号和数据。EXE阶段需要标记OVERFLOW例外和内存写地址错误例外，并将例外标记传到后续阶段。

（六）重要模块6设计：访存阶段（MEM\_stage）模块

1. 工作原理

将从EXE模块获取的访存指令相应的执行。根据es\_to\_ms\_bs中的是否数据来自数据RAM信号确定是否有访存取出的数据，并将相应指令的最终结果和前阶段传递的通用寄存器写使能、写地址控制信号、PC传通过总线在下一时钟周期更新给WB模块。MEM阶段需要标记是否有LOAD地址错例外，并将结果传递给WS阶段。标记代码如下：

assign adel\_mem = (ms\_load\_type==`LW\_TYPE&&ms\_vaddr[1:0]!=2'b00)?1:

(ms\_load\_type==`LH\_TYPE&&ms\_vaddr[0]!=1'b0)?1:

(ms\_load\_type==`LHU\_TYPE&&ms\_vaddr[0]!=1'b0)?1:

0;

assign ms\_exc\_type = (es\_exc\_type==4'd0 && adel\_mem)? `ADEL\_MEM: es\_exc\_type;

1. 接口定义

表2.4 MEM\_stage接口定义

| **名称** | **方向** | **位宽** | **功能描述** |
| --- | --- | --- | --- |
| clk | IN | 1 | 时钟信号 |
| reset | IN | 1 | 复位信号 |
| ws\_allowin | IN | 1 | WB模块允许接受MEM传值 |
| ms\_allowin | OUT | 1 | MEM模块允许接受EXE传值 |
| es\_to\_ms\_valid | IN | 1 | EXE模块可以向MEM模块传值 |
| es\_to\_ms\_bus | IN | 154 | EXE模块向MEM模块传递数据 |
| ms\_to\_ws\_valid | OUT | 1 | MEM模块可以向EXE模块传值 |
| ms\_to\_ws\_bus | OUT | 118 | MEM模块向WB模块传递数据 |
| data\_sram\_rdata | OUT | 32 | data\_sram读出的数据 |
| out\_ms\_valid | OUT | 1 | 将ms\_valid的值传递给ID模块 |
| ws\_ex | IN | 1 | WB阶段有例外标记 |
| ms\_exc\_type | OUT | 4 | MEM阶段例外类型 |
| ms\_eret | OUT | 1 | MEM阶段有eret指令 |
| eret\_flush | IN | 1 | eret刷新流水线信号 |

1. 功能描述

将从EXE模块获取的访存指令相应的执行。确定是否有访存指令，并将相应指令的数据和前阶段传递的数据控制信号传通过总线在下一时钟周期更新给WB模块。

MEM阶段需要标记是否有LOAD地址错例外，并将结果传递给WS阶段。

（七）重要模块7设计：访存阶段（WB\_stage）模块

1. 工作原理

将从MEM模块获取的写回指令相应的执行。确定是否有写回指令，并进行相应的操作。同时，该模块将PC、寄存器堆写使能、地址、和写回结果传递给debug模块，用于调试CPU的正确性。在本次设计中，cp0寄存器位于WB文件下，即在WB级后处理例外。遇到例外时，因向之前的流水级发出eret\_flush信号来刷新流水线。相应流水级据此控制相关PC和写使能。

表2.5 WB\_stage接口定义

| **名称** | **方向** | **位宽** | **功能描述** |
| --- | --- | --- | --- |
| clk | IN | 1 | 时钟信号 |
| reset | IN | 1 | 复位信号 |
| ws\_allowin | OUT | 1 | WB模块允许接受MEM传值 |
| ms\_to\_ws\_valid | IN | 1 | MEM模块可以向WB模块传值 |
| ms\_to\_ws\_bus | IN | 84 | MEM模块向WB模块传递数据 |
| ws\_to\_rf\_bus | OUT | 73 | WB模块向寄存器堆模块（通过ID模块）传递数据（包括ws\_gr\_we、ws\_valid等） |
| debug\_wb\_pc | OUT | 32 | debug显示PC |
| Debug\_wb\_rf\_wen | OUT | 4 | debug显示寄存器堆写使能 |
| Debug\_wb\_rf\_wnum | OUT | 5 | debug显示寄存器堆写地址 |
| Debug\_wb\_rf\_wdata | OUT | 32 | debug显示寄存器堆写数据 |
| ws\_ex | OUT | 1 | WB阶段有例外标记 |
| ws\_exc\_type | OUT | 4 | WS阶段例外类型 |
| eret\_flush | OUT | 1 | eret刷新流水线信号 |

1. 功能描述

将从MEM模块获取的写回指令相应的执行。确定是否有写回指令，并进行相应的操作。同时，该模块将PC、寄存器堆写使能、地址、和写回结果传递给debug模块，用于调试CPU的正确性。在WB流水级之后处理例外，并更新相应cp0寄存器的值。遇到例外时，因向之前的流水级发出eret\_flush信号来刷新流水线。相应流水级据此控制相关PC和写使能。

（八）重要模块8设计：HI/LO寄存器模块

1、工作原理

记录乘法指令生成的完整的64位乘积，以及除法指令生成的各32位的商和余数。

表2.6 hi/lo寄存器模块接口定义

| **名称** | **方向** | **位宽** | **功能描述** |
| --- | --- | --- | --- |
| clk | IN | 1 | 时钟信号 |
| rd\_hi | OUT | 32 | 从cp0模块中读取hi寄存器的值 |
| rd\_lo | OUT | 32 | 从cp0模块中读取lo寄存器的值 |
| hi\_we | IN | 1 | 写hi寄存器使能 |
| lo\_we | IN | 1 | 写lo寄存器使能 |
| wd\_hi | IN | 32 | 写hi寄存器数据，本实验中为乘法指令生成结果的高32位以及除法指令生成结果的余数 |
| wd\_lo | IN | 32 | 写lo寄存器数据，本实验中为乘法指令生成结果的低32位以及除法指令生成结果的商 |

2、功能描述

类似于寄存器堆regfile模块，实现对hi、lo寄存器的同步写和异步读。

三、实验过程（50%）

（一）实验流水账

10月31日 20：00-20：30 阅读讲义

10月31日 20：30-22：00 设计代码

11月2日 10：00-11：30 继续实现代码

11月2日 14：00-15：30 调试bug

11月2日19：30-22：30 撰写实验报告

（二）错误记录

1、错误1：写寄存器的值错误

（1）错误现象

运行仿真，比对trace，发现wb\_rf\_wdata值有X，如图3.1所示：

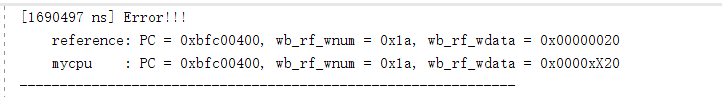


图3.1 wb\_rf\_wdata有误

（2）分析定位过程

通过查看trace的对比节点，发现：在PC=0XBFC00400，相关的汇编指令如图3.2：



图3.2 相关汇编指令

分析指令，并观察如图3.3的波形，发现是cp0\_cause的值出现问题，定位到源码如图3.4：



图3.3 cp0\_cause值的错误

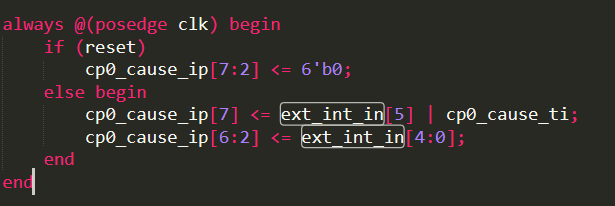


图3.4 cp0\_cause源码

（3）错误原因

ext\_int\_in未赋初值。

（4）修正效果

这次实验将ext\_int\_in全部置为0，这个错误被解决。

2、错误2：PC值出错

（1）错误现象

运行仿真，比对trace，发现PC值出错，如图3.5所示：

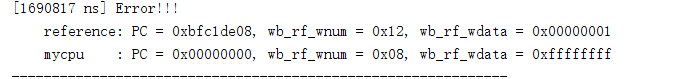


图3.5 PC出错

（2）分析定位过程

如图3.5，通过查看对应处的汇编指令，发现导致错误的原因应该是syscall执行完eret之后跳回到的epc值出错，观察如图3.6的波形，发现cp0\_epc的值为0。

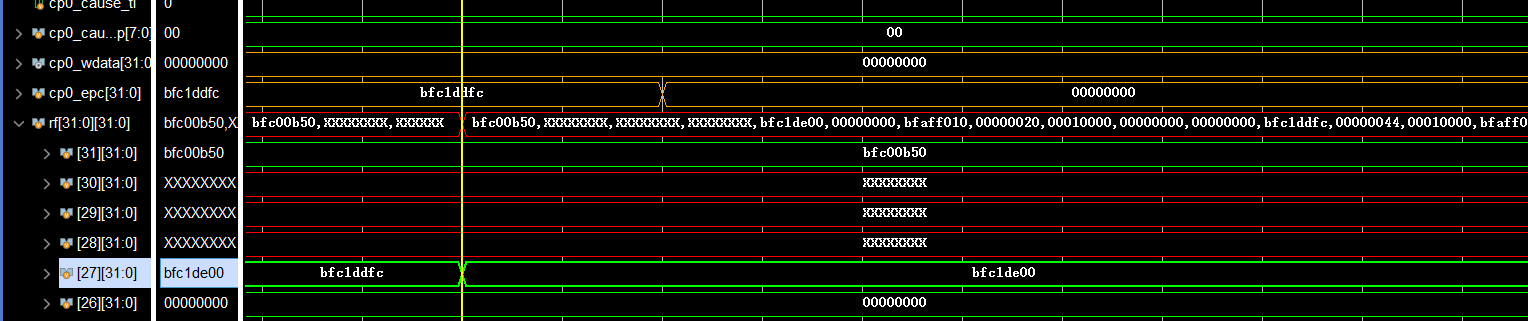


图3.6 cp0\_epc值出错

查看对应的汇编代码如图3.7，发现mtc0之后两拍就是eret，说明mtc0还没有把epc写入eret就跳转了，所以需要让eret阻塞在ID级。

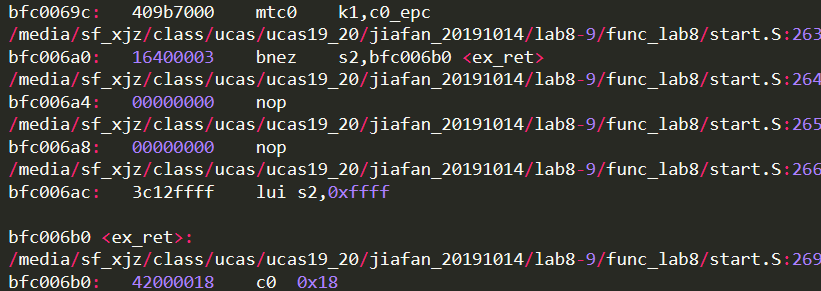


图3.7 对应汇编代码

（3）错误原因

mtc0和eret发生了写后读相关。

（4）修正效果

如图3.8代码所示，在ID级阻塞新增mtc0和eret的写后读相关判断，修正完毕后，该问题得到解决。

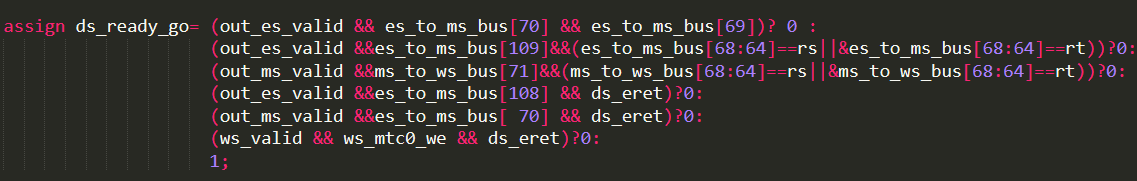


图3.8 ds\_ready\_go新增mtc0和eret写后读相关

所有测试顺利通过。

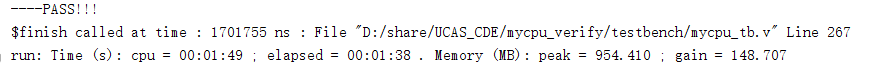


图3.9 测试通过

四、实验总结（可选）

本次实验需要处理例外操作，设计难度相较于前面几次来说较大。由于实验框架之前大致已经搭好，因此添加基本指令难度不大。但本次实验在处理例外时，需要考虑增加或更改数据通路来传递数据和信号，而设计自由度较高，因此在设计前期需要考虑清楚各类信号的传递。在设计过程中可能会设计一些冗杂的信号或者通路，这个需要我们在今后的实验中进行改正和优化。