**实验7报告**

蔡润泽、付轶凡

箱子号：45

一、实验任务（10%）

本次实验任务是要在实验六的 CPU 代码基础上添加更多的指令，具体包括转移类指令 BGEZ、BGTZ、BLEZ、BLTZ、J、 BLTZAL、BGEZAL、JALR，乘除运算类指令 MULT、MULTU、DIV、DIVU，以及访存指令 LB、LBU、 LH、LHU、LWL、LWR、SB、SH、SWL、SWR。运行 func\_lab7，要求成功通过仿真和上板验证。

二、实验设计（40%）

（一）总体设计思路

硬件设计图如下：

**图片包含 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8个模块，包括五级流水、ALU、寄存器堆以及cp0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）。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。

图片包含 音乐

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。

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

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。

（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 。

1. 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 |
| fs\_to\_ds\_valid | OUT | 1 | IF模块可以向ID模块传值 |
| fs\_to\_ds\_bus | OUT | 64 | 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 。

（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”。

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 | 64 | 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 | 153 | 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 | 40 | WB模块向ID模块传递的需要写回REG FILE的信息 |
| es\_to\_ms\_bus | IN | 108 | EXE模块向MEM模块传递数据 |
| ms\_to\_ws\_bus | IN | 70 | 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 |

相较上次实验，本次实验的代码设计中在ds\_to\_es\_bus中增添了load\_type、vaddr\_2、store\_type三个信号，用来描述load、store指令类型以及虚拟地址，所以ds\_to\_es\_bus总线的位宽扩充至153。

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。另外，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。

（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。

在本次任务中，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。

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 | 153 | 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 | 108 | 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 |

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。

另外es\_to\_ms\_bus相较上次新增添了es\_load\_type、es\_vaddr\_2、es\_rt\_value三个信号，给MEM阶段传load指令类型、虚拟地址以及rt寄存器的值（用来完成lwl和lwr指令写rt寄存器值的拼接）。

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。

（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。

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 | 108 | 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 | 70 | 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 |

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。

（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。

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 | 70 | MEM模块向WB模块传递数据 |
| ws\_to\_rf\_bus | OUT | 40 | 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 |

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。

（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18日 20：00-20：30 阅读讲义

10月18日 20：30-22：00 设计代码

10月19日 10：00-11：30 继续实现代码

10月20日 14：00-15：30 调试bug

10月20日19：30-22：30 撰写实验报告

（二）错误记录

1、错误1：trace比对节点出现错误

（1）错误现象

运行仿真，比对trace，发现PC、wb\_rf\_wnum、wb\_rf\_wdata值全都不一样，如图3.1所示：

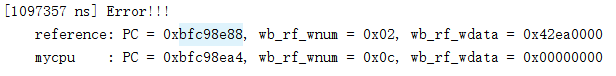


图3.1 当前对比节点不匹配

（2）分析定位过程

通过查看trace的对比节点，发现：在PC=0XBFC98EA4时，wb\_rf\_wnum、wb\_rf\_wdata的值正确，而在PC=0XBFC98E88时，wb\_rf\_wnum、wb\_rf\_wdata的值也正确。此时出现的错误应该为在不该进行trace对比时进行了trace对比，导致reference和mycpu出现了不匹配。通过查看波形，确认出错PC处对应的指令涉及到跳转指令，如图3.2.

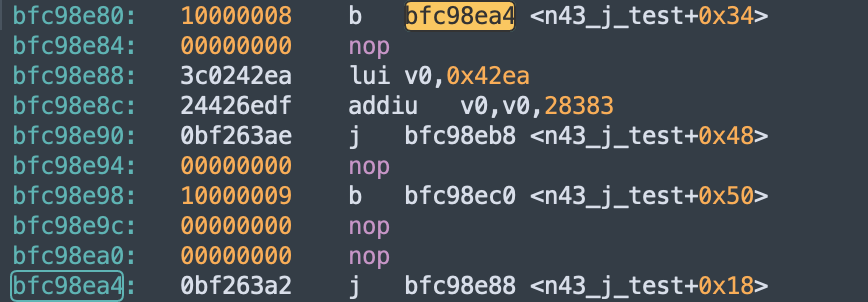


图3.2 相关汇编指令

分析指令，只有在通用寄存器写使能时，才会进行trace比对。因此通过查看通用寄存器写使能信号，如图3.3,发现：在译码阶段gr\_we信号缺少了~inst\_j，同时发现 inst\_bgez写成了isnt\_bgez。

图片包含 物体

描述已自动生成

图3.3 错误的gr\_we代码

（3）错误原因

译码阶段gr\_we信号缺少了~inst\_j，同时发现 inst\_bgez写成了isnt\_bgez。

（4）修正效果

在gr\_we信号的末尾加上“& ~inst\_j”，并将“isnt\_bgez”修改为“inst\_bgez”。修改后进行仿真，该问题被解决。

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3.4所示：

图片包含 屏幕截图

描述已自动生成

图3.4 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在PC=0xbfc58934处，应该实现跳转，而该设计的CPU没有出现跳转。

图片包含 文字

描述已自动生成

图3.5 对应处的汇编指令

通过查看ID阶段对BGEZ指令的处理，发现：原来设计的代码中，对于rs\_value和0作比较时，没有改成有符号数，导致比较结果全是rs\_value恒大于等于0。

图片包含 墙壁

描述已自动生成

图3.6 ID阶段对应代码

（3）错误原因

rs\_value和0作比较时，没有改成有符号数

（4）修正效果

如图3.7代码所示，在源代码的基础上增加了$sigend（），修正完毕后，该问题得到解决。

图片包含 文字

描述已自动生成

图3.7 对rs\_value增加了$sigend（）

3、错误3：trace比对写寄存器的值有误

（1）错误现象

运行仿真，比对trace，在PC值为0xbfc6e0a4发生了写寄存器值错误，如图3.8：

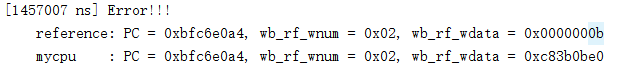


图3.8 wb\_rf\_wdata出错

（2）分析定位过程

在test.s中找到对应指令为“lb v0,14241(t0)”，说明lb指令写值有误，观察mycpu的wb\_rf\_wdata，发现这不可能是lb指令应该出现的结果，于是抓取MEM阶段的load\_type信号到波形中，发现此时该信号为9，在我们设置的宏定义中，load\_type只有3bit，最多为7，如下图：

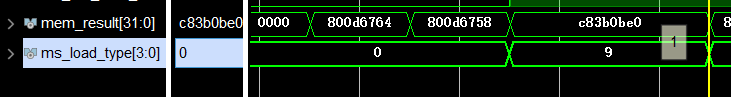


图3.9 ms\_load\_type值有误

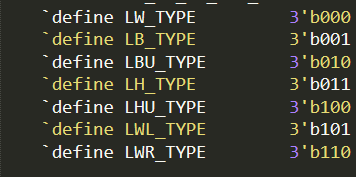


图3.10 mycpu.h中关于load指令类型的宏定义

立即想到是ms\_load\_type位宽错误，从波形图中也可以发现ms\_load\_type的位宽设置成了4。

（3）错误原因

ms\_load\_type位宽错误。

（4）修正效果

将ms\_load\_type位宽改为3，再次仿真，如图3.11，所有测试顺利通过。

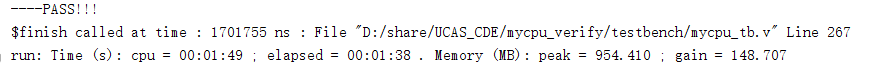


图3.11 测试通过

四、实验总结（可选）

本次实验也需要自己通过查看MIPS手册来添加一些新的指令。在设计代码的过程中，依旧需要保持细致，不然还是会出现例如变量名写错等问题。另外，由于本周时间有限，在LWL/LWR指令设计的选择上，只选择了可以复用上学期组成原理课数据通路的方案二，希望能在下次实验中，能够对这一块进行优化。