32位MIPS综合实验测试文档

原子小组

January 2, 2016

目　　录

1 引言

1.1 编写目的

测试是程序开发的一个重要部分。对硬件开发来说，测试的难度比软件开发更高，问题也更隐蔽。其重要性不亚于设计和实现部分。因此书写本测试文档，意在列出测试需求并设计测试样例，对结果进行整理和汇总，并记录这其中遇见的问题。

1.2 测试方案

本项目的测试需求为从小到大，从局部到整体，从单指令到多指令，直到最后能跑起操作系统。总体测试设计如下：

1. 单元测试
2. 指令片段测试
3. 冲突测试
4. 系统测试

说明：
在测试过程中，若无特殊说明，均为将时钟接入单步时钟进行调试。通过32个拨码开关来选择此时想查看的变量， 并通过16个LED灯查看结果。为了能够使16位灯显示32位结果，我们在拨码开关上留下一位用于控制LED灯显示为高16位还是低16位。

2 单元测试

2.1 译码模块

2.1.1 模块说明

本模块用于将32位指令码翻译成具有具体功能的指令，并将对应的信号传递给ALU模块。本模块的测试重点在于能不能正确取出指令，并能正确解码指令。

2.1.2 测试方法

将需要测试的指令写入ROM中，然后令CPU从ROM中取出指令。然后通过LED灯查看取出的指令是否正确，并可以查看译码出来的控制信号或操作码等信号是否正确。

2.1.3 测试样例

所有65条指令码+1条不存在指令码

2.1.4 测试结果

cache指令发现了译码错误的问题，之后将其改成了特殊指令单独处理。

mtc0和mfc0在参考MIPS32指令规范时发现其指令码与实验指导文档上有出入。最后选定了规范译码方法。

2.2 执行模块

2.2.1 测试方法

将指令分类为逻辑运算指令、算数操作指令、转移指令、加载存储指令和其它指令五种类型，分别进行测试。通过查看这一阶段算出的最终结果判断是否执行正确。

2.2.2 测试样例

测试样例1至测试样例9，注意到了边界条件。

2.2.3 测试结果

所有样例可正确运行，其中mult指令可在一个周期内给出结果。

2.3 访存模块

2.3.1 测试方法

在外设没有完成的情况下用数组的方法模拟RAM，在其中进行读写。

2.3.2 测试样例

测试样例10

2.3.3 测试结果

测试发现sb指令、lb指令、lbu指令指令在处理地址后两位时候把各种情况的对应处理写反了。修改后测试通过

2.4 CP0模块

2.4.1 测试方法

使用mtc0、mfc0指令对CP0寄存器进行操作。

2.4.2 测试样例

测试样例12

2.4.3 测试结果

样例正确。

2.5 异常处理模块

2.5.1 测试方法

修改异常入口，将异常入口放在0x40的位置，主程序从0x100地址开始，从而使得主程序部分与异常处理程序部分不会发生冲突。

2.5.2 测试样例

测试样例14至17

2.5.3 测试结果

在发生异常时，PC能正确进入异常处理例程，对相应CP0寄存器进行赋值，并能正确通过eret指令返回到epc寄存器存储的地址中继续执行。

2.6 MMU模块

2.6.1 模块说明

此模块用于进行虚拟地址到物理地址的转换。

2.6.2 测试方法

通过加载存储指令访问各外设对应地址，看是否能选到对应外设。

2.6.3 测试样例

各外设一个样例，共9个样例。

2.6.4 测试结果

样例均正确。

3 指令片段测试

3.1 逻辑、移位操作与空指令的测试

具体样例见测试样例1至4。
cache指令出现译码错误情况，修改方法为将其作为特殊指令处理。

3.2 移动操作指令的测试

具体样例见测试样例8至9.

CPU设计时加上了延迟槽设计，故在延迟槽中放入普通指令而不是nop指令进行测试。测试结果均正确。

3.3 算术操作指令的测试

具体样例见测试样例7.

测试包括溢出情况。因异常处理中没有加入溢出的情况，所以当发生溢出时不做任何处理，也不进行操作。

3.4 转移指令的测试

具体样例见测试样例12.

测试发现mthi、mtlo指令没有正常工作，根据信号检查后发现是顶层文件中hi、lo引脚没有连，连上后测试通过。

3.5 加载存储指令的测试

具体样例见测试样例10

初期测试时没有考虑好结构冲突的问题，以为能将指令和数据分开存储。导致后期写仲裁器处理结构冲突时遇到了很多问题。然而在不考虑结构冲突的情况下，加载存储指令均能正确执行。

3.6 协处理器访问指令的测试

具体样例见测试样例12

测试均通过。

3.7 异常相关指令的测试

具体样例见测试样例13至17

遇见的第一个问题为应触发异常时没有正确触发异常。经过查看信号值发现原因时异常号错误。最后修改异常号后测试通过。

第二个问题时测试syscall指令时没有触发异常。经过查看信号后发现原因是进行异常处理前没有清空流水线，导致流水线仍然在工作，清空流水线后测试通过。

第三个问题时测试地址未对齐异常时没有正确触发。经过查看信号后发现原因是原先的代码在进行异常种类判定时忽略了访存阶段的异常，将访存阶段的异常加入判定后测试通过。

此外发现了诸如CPU异常和无效指令异常的bit写反了这样的小bug，更正后测试通过。

4 特殊情况测试

4.1 与数据冲突有关的测试

与数据冲突有关的测试主要分为普通寄存器的数据冲突、HILO寄存器的数据冲突、CP0寄存器的数据冲突和load相关问题。

1. 普通寄存器的数据冲突

   普通寄存器的数据冲突是指连续两条指令访问同一寄存器，且第二条指令译码阶段需要依赖第一条指令得出的寄存器值时，由于第一条指令不能及时写回寄存器而导致的数据冲突。在这种情况下，只需令第一条指令在检测到相关问题时将数据前推回译码阶段即可。

2. HILO寄存器的数据冲突

   与普通寄存器的数据冲突类似，只需在检测到相关问题时令数据前推即可。

3. CP0寄存器的数据冲突

   与普通寄存器的数据冲突类似，只需在检测到相关问题时令数据前推即可。

4. load相关问题

   load相关问题是指load类指令后接branch类指令，branch类指令需要依赖load类指令所写回的寄存器值，而load类指令来不及写回寄存器导致的。在这种情况下数据前推也来不及，因此需要插入气泡，使得回写、访存、执行阶段继续执行，而取指和译码阶段暂停。这样load类指令可以在访存阶段得到正确的值并前推回译码阶段，使得branch类指令能得到正确值。

对这四种情况分别进行了测试，分别为测试样例2、6、12、11.
普通寄存器的数据冲突一开始没有解决，后来发现是因为某个变量的错别字。改正后测试通过。

4.2 与结构冲突有关的测试

结构冲突是指取指阶段和访存阶段同时需要访问存储器时产生的冲突。为了解决这个问题，我们在CPU与外设之间加入了总线，并在总线上实现了仲裁器。

仲裁器的修改耗时较长，最后确定了写法后测试通过。样例主要为加载存储类指令。对应样例为测试样例9

4.3 与控制冲突有关的测试

因为我们的CPU中引入了延时槽的设计，所以不会有控制冲突问题。为了测试延时槽中指令是否能正确执行，测试样例中延时槽中指令并不是nop而是其它运算指令。具体样例为测试样例8至9，测试均通过。

5 系统测试

系统测试是片段测试的扩展，主要为bootloader和ucore的测试。

5.1 bootloader测试

5.1.1 测试过程

将bootloader的程序烧进ROM中，设定取指的起始地址为ROM的起始地址，从而执行bootloader指令。

5.1.2 测试结果

程序一开始就遇到了写flash的指令，导致flash被修改从而无法正确执行后面的指令。经过检查后修改了bootloader的代码，从而解决了这个问题。

之后遇到了load相关问题，发现之前对仲裁器的设计还是不合理。修改了之后load相关问题顺利解决。

再之后遇到了从flash里读出指令放入RAM时指令错误的情况，检查后发现flash与主频使用同一时钟会导致错误。修改flash工作频率为50MHz后正确运行。

至此bootloader运行通过，CPU进入ucore内核态运行。

5.2 ucore测试

5.2.1 测试过程

在执行bootloader之前先将ucore的二进制文件烧入flash中，bootloader在执行时会将指令从flash中取出来放进RAM里，再从RAM里读取指令并继续执行ucore的指令。

5.2.2 测试结果

在运行ucore时遇到了不存在指令异常，检查后发现是mtc0指令和mfc0指令在译码上不符合MIPS32规范。修改后执行正确

重新运行ucore时遇到了访问不存在的CP0寄存器异常。问过助教之后发现是由于我们所使用的ucore中对CP0寄存器的定义与标准MIPS32定义不同，而我们CPU中与标准MIPS32定义是相同的。这个问题至今没有解决，也说明了我们前期对ucore了解不够，导致后期失利。