测试 CPU

测试 CPU

在完成 CPU 搭建后，进行 CPU 的测试是一个重要的环节，它可以检查出我们 CPU 设计和实现的错误。在 Pre 的 [“MIPS 指令集及汇编语言”](http://cscore.buaa.edu.cn/tutorial/mips/mips-6/mips6-1/) 部分中，我们简要介绍了测试程序设计的一些要点，而在分析同学们的课下提交时，我们也发现一些测试方面问题，下面将就课下指令功能测试进行一些补充。

计算类指令功能测试

* 寄存器数据方面，可以考虑以下情况：
  + 00 及附近的数：-2, -1, 0, 1, 2−2,−1,0,1,2
  + 3232 位数边界附近的数： -2147483648, -2147483647, 2147483646, 2147483647−2147483648,−2147483647,2147483646,2147483647
  + 3232 位数范围内的一些随机数：-1000786109, 1919156834, ...−1000786109,1919156834,...
* 无符号立即数方面，可以考虑以下情况：
  + 00 及附近的数：0, 1, 2, 30,1,2,3
  + 1616 位无符号数边界附近的数：65533, 65534, 6553565533,65534,65535
  + 1616 位无符号数范围内的一些随机数：25779, 42528, ...25779,42528,...
* 符号立即数 (P3 不涉及) 方面，可以考虑以下情况：
  + 00 及附近的数：-2, -1, 0, 1, 2−2,−1,0,1,2
  + 1616 位符号数边界附近的数：-32768, -32767, 32766, 32767−32768,−32767,32766,32767
  + 1616 位符号数范围内的一些随机数：-5329, 25299, ...−5329,25299,...
* 特别的，可注意测试目标寄存器是 \$0$0 的情况。

存取类指令功能测试

* offset 方面，可以考虑以下情况：
  + offset 是正数
  + offset 是零
  + offset 是负数
* $base 寄存器方面，可以考虑以下情况：
  + $base 寄存器中的值是正数
  + $base 寄存器中的值是零
  + $base 寄存器中的值是负数
* 特别的，对于 sw 指令，建议存入的 word 中，每个 byte 都不是零。
* 特别的，对于 lw 指令，可注意测试目标寄存器是 $0 的情况。

跳转类指令功能测试

* 对于非比较相关的部分，可以考虑以下情况：
  + 跳转，且目标在此跳转指令之前
  + 跳转，且目标是此跳转指令
  + 跳转，且目标在此跳转指令之后
  + 不跳转，且目标在此跳转指令之前
  + 不跳转，且目标是此跳转指令
  + 不跳转，且目标在此跳转指令之后
* 对于比较相关的部分，本质上依旧是构造寄存器数据，处理类似 “计算类指令功能测试”。

其它测试上的建议

* 为更方便的进行测试，同学们可以将 IM 调大至 10241024，以便一次测试执行更多的指令。
* 为更有效的进行测试，同学们可以在测试程序开始时将 3131 个寄存器初始化成非 00 值 (需要配合调大 IM)，这往往有助于发现 Bug，有兴趣的同学可以思考其原因。
* 在本实验中，我们要求 PC 和 DM 的起始地址均为 00，而 MARS 是不能够将两者的地址区间设定重叠的。因此大家可以修改生成的指令机器码，或者对本地的 Logisim 电路进行一些处理，来保证测试能够顺利进行，但请注意：
  + 课上测试时，要求起始地址为 0x00000000，因此采用此方式测试时，请保证起始地址**易于修改**，**以免课上测试时出现问题**。
  + 课上测试时，要求起始地址为 0x00000000，因此采用此方式测试时，请保证起始地址**易于修改**，**以免课上测试时出现问题**。
  + 课上测试时，要求起始地址为 0x00000000，因此采用此方式测试时，请保证起始地址**易于修改**，**以免课上测试时出现问题**。
* 事实上，在现代主流计算机中，数据存储器和指令存储器的起始地址**不应该重叠**，但在我们的设计中，由于采用分离存储器设计方案，因此可以暂时忽略这一点。
* 为验证自己设计的正确性，一种简易且可行的方式是：
  + 对直接产生结果的指令，将结果存入内存，同时增加维护存入内存位置的 “指针”，其逻辑类似以下代码：

save [index++] = result;

* + 对于不产生结果的指令，可以通过构造指令序列，使指令执行结果不同时 (如跳转 / 不跳转)，存入内存的值不同或值的数量不同，其逻辑类似以下代码：
  + **if** (branch) save [index++] = result1;

**else** save [index++] = result2;

**if** (branch) save [index++] = result;

save [index++] = result;

* + 测试程序运行结束后，将 Logisim 中 DM 中的内容和 MARS 中内存中的内容导出并比对，由于两者格式不同，同学们可以实现一个用于比对的程序，对于经过程序设计和数据结构训练的同学们而言，这并不难。
  + 使用这种方式时，同学们可以将 DM 也调大至 10241024 或更多。
  + 此外我们也可以通过其它方式进行正确性验证，同学们可以自行探索。
* 最后，虽然在实际开发中测试驱动开发是一种可行的模式，但在本课程中测试更多的是一种 “锦上添花”。我们鼓励同学们构造样例进行测试，但请优先保证**完成处理器本身的设计**并**通过课下测试**。

文档撰写建议

请在测试文档中阐述你采用的测试方案，将测试样例分类、有条理地列出 (对于测试样例较多或较长的情况，可只展示关键部分)。如果采用了自动生成程序或开发了较完整的评测系统，需要对这一系统的功能进行描述，并对其使用方法与实现原理进行说明。

**思考题**

上文提到，MARS 不能导出 PC 与 DM 起始地址均为 00 的机器码。实际上，可以通过为 DM 增添片选信号，来避免手工修改的麻烦。请查阅相关资料进行了解，并阐释为了解决这个问题，你最终采用的方法。

除了编写程序进行测试外，还有一种验证 CPU 设计正确性的办法 —— 形式验证。**形式验证**的含义是根据某个或某些形式规范或属性，使用数学的方法证明其正确性或非正确性。请搜索 “形式验证 (Formal Verification)”，了解相关内容后，简要阐述相比于测试，形式验证的优劣之处。