测试 CPU

> 通过课下测试并**不意味着你的设计完全不存在问题**，建议自行构造测试用例以验证设计的正确性（如果进行了相关工作，请写入设计文档，课上问答时将酌情加分）。

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”一节中，我们初步介绍了测试程序设计的一些要点并给出了一个参考的测试样例。相比于课上强测，Pre 中给出的测试样例在量和强度上都存在一定不足，下面将就课下指令功能测试进行一些建议和补充。

计算类指令功能测试

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

- 为更正确地进行测试，如果利用 Mars 进行对拍，应该将 Mars 和自己的 CPU 内存保持一致，因此需要在 Mars 中将 Memory Configuration 设置为 Compact,Data at Address 0。才能让 Mars 的 PC 起始于 0x3000, DM 起始于 0x0000。
- 为更合理地进行测试，如果利用 Mars 进行对拍，单个测试点最多指令数不可以超过 1024 条，这个限制来源于 Mars 的 .text 段长度限制，有兴趣的同学可以思考其原因，并考虑能否通过修改 Mars 使其支持更大的指令容量以符合题目要求（例如 IM 容量）并进行测试。
- 为更有效地进行测试，同学们可以在测试程序开始时将 31 个寄存器初始化成非 0 值，这往往有助于发现 Bug，同样的，有兴趣的同学可以思考其原因。
- DM 的地址范围为 0x0000 - 0x2fff. 生成 Load/Store 类指令时注意不要地址越界，这个范围同样来源于 Mars 的 .data。
- 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;
```

其对应的汇编语言可以为以下代码：

```
# 假设此时的 $t0 维护存入内存位置的 “指针”，$t1 保存的是结果
sw $t1, 0($t0)       # 将 result 保存到内存位置
addi $t0, $t0, 4     # 维护 index 指针
```

- 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：

  保存不同数值：

  ```
  if (branch) save[index++] = result1;
  else save[index++] = result2;
  ```

  其对应的汇编语言可以为以下代码：

  ```
  # 假设此时的 $t0 维护存入内存位置的 “指针”，$t1 和 $t2 分别保存两种不同的数值
  # branch 表示一种分支指令，如：beq $s0, $s1
  branch, save_result1
  nop
  sw $t2, 0($t0)       # 将 result2 保存到内存位置
  addi $t0, $t0, 4     # 维护 index 指针
  j end_branch
  nop

  save_result1:
  sw $t1, 0($t0)       # 将 result1 保存到内存位置
  addi $t0, $t0, 4     # 维护 index 指针

  end_branch:
  #.......
  ```

  保存不同数量的值：

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

  其对应的汇编语言可以为以下代码：

  ```
  # 假设此时的 $t0 维护存入内存位置的 “指针”，$t1 用于保存一个数值
  #  branch 表示一种分支指令，如：beq $s0, $s1
  branch, save_result
  nop
  j end_branch
  nop

  save_result:
  sw $t1, 0($t0)       # 存入 result
  ```

```
addi $t0, $t0, 4      # 维护 index 指针

end_branch:
sw $t1, 0($t0)        # 存入 result
addi $t0, $t0, 4      # 维护 index 指针
#......
```

- 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。
- 此外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。在这里建议学有余力的同学尝试在 Pre 部分涉及的Logisim 自动化测试方法。在后面的 Project 中，同学们也可以采用类似的思想辅助进行自动化测试。
- 为了避免课上产生不必要的 Bug，如果在测试时修改了 IM 和 DM 的规格，**请一定保证在进入课上时进行恢复！**

思考题

阅读 Pre 的“MIPS 指令集及汇编语言”一节中给出的测试样例，评价其强度（可从各个指令的覆盖情况，单一指令各种行为的覆盖情况等方面分析），并指出具体的不足之处。