P3 CPU 设计文档

21371064 权尚浩然

2022 年 10 月 27 日

1 设计草稿

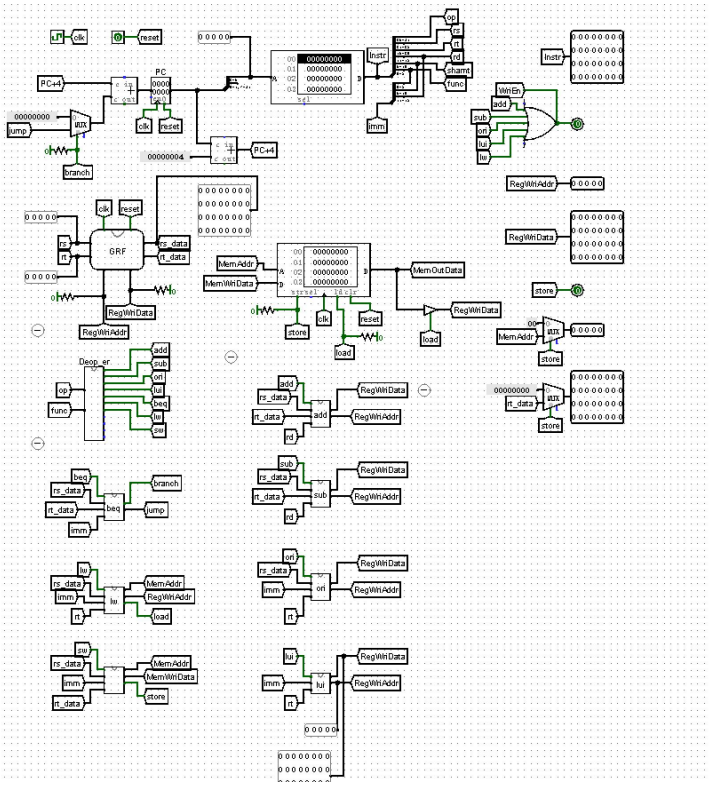

图 1: CPU 顶层设计

我搭了两个 CPU，第一个把指令划分成 R 型、I 型分别处理，但发现很多指令需要分别设计，于是在第二个版本中直接对每一条指令分别设计子电路。

可以发现，我设计的 CPU 没有总的 controller 和 ALU 单元，而是把这二者的功能下分到任意一个指令单元，在最上层只进行译码和分配的操作。在译码阶段，我们可以获得所有需要的寄存器数据信息，然后每一个指令单元接入它需要的输入即可，输出同理。

这样做的优势是在增添指令时比较方便，因为它直接面向指令。但需要指出的是当指令条数增多时，它的结构会比较臃肿，会有很多不必要的计算开销。这也反面反映出了 ALU 结构单元的作用：在较少指令时我们可以分别计算体现不到其优势，而指令多了后 ALU 单元可以被共用，但需要更加细致的设计。

2 测试方案

对于我们的设计，保证上层模块译码、取址没问题后，我们只需测试每条指令的子模块，故每条指令分别测试即可，此时我们可以手动输入一些指令来测试。

同时我还用 C++ 编写了一个随机生成指令的指令生成器，生成后分别输入到 mips 和 logisim 中执行，比较寄存器和主存数据，这样可以更全面地进行测试。

3 思考题解答

3.1 上面我们介绍了通过 FSM 理解单周期 CPU 的基本方法。请大家指出单周期 CPU 所用到的模块中，哪些发挥状态存储功能，哪些发挥状态转移功能。

RAM 和 GRF 发挥状态存储功能，ALU 和各指令子模块发挥状态转移功能。

3.2 现在我们的模块中 IM 使用 ROM，DM 使用 RAM，GRF 使用 Register，这种做法合理吗？请给出分析，若有改进意见也请一并给出。

合理。执行的指令在导入后不会改变也不应该改变，应该使用 ROM，课程中 32 字主存似乎可以使用 GRF 实现，但当主存变大后不现实，采用 RAM，GRF 需要频繁取值和存值，而且每一次取值可能要取两个位置的值，使用 RAM 实现较为牵强，采用 register 更合理。

3.3 在上述提示的模块之外，你是否在实际实现时设计了其他的模块？如果是的话，请给出介绍和设计的思路。

是。我还设计了译码装置便于分配指令，以及每个指令都有自己的计算单元。

3.4 事实上，实现 nop 空指令，我们并不需要将它加入控制信号真值表，为什么？

因为其所有控制信号输出均为 0。（此外，事实上 mips 中 nop 全 0 指令会以 sll $0 $0 0 被执行）

3.5 上文提到，MARS 不能导出 PC 与 DM 起始地址均为 0 的机器码。实际上，可以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。

在 logisim 中加入模块把 PC 和 DM 值减去初始值。

3.6 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。

没覆盖 sub 和 nop，其余各指令覆盖较为全面，可以起到初步测试的效果。