pro1

请同学们查看代码、运行代码，回答如下问题：

- 内置的cache项数为：
- cache的组织形式为：（全相联？等等）
- cache的block_size为：
- cache的替换策略为：
- 内置的conv实现方案可以提高效率多少倍？

pro2

请同学们在上题的理解基础上，进行如下修改：

- 修改cache的静态指标，如项数、block_size，并分析对应的结果
- 修改卷积计算的6层循环的顺序，找到一种更优的循环方案
- 修改cache的替换策略为FIFO（可能需要额外设置变量来记录相关信息）

pro3（附加题）

请同学们在上题的基础上，结合理论课所将，对代码进行较大规模修改，完成一下任务：

- 修改cache的替换策略为Clock或LRU算法，如果你觉得他们效率太低，可自行设计替换算法，只需要效率比FIFO高即可
- 修改back_to_memory函数，给cache增加dirty位，防止进程结束后的无用写回

说明

pro1-pro3均需要在下发的代码上进行修改，需要新增数据结构时可任意新增，附加题可以修改原有的数据结构（如增加dirty位可能要修改cache的数据结构）

pro4（附加题、开放题）

事实上，6层循环计算卷积是相对低效的。同学们可以尝试大规模修改conv函数，并修改替换策略，以更好的提高效率

- CPU是通用性计算单元，因为大量随机访问与跳转，他的Cache替换策略必须具有普适性。即使是FIFO，在相当多的情况下也远好于完全随机。但是，卷积计算拥有如下两个特性
  - 访存顺序事先完全可预知
  - 不存在跳转、异常等情况

  因此，计算卷积的时候，CPU的通用性反而会降低性能。在cache容量一定（nums * block_size）的情况下，使用特殊的访存顺序和配套的cache策略可以实现比CPU高得多的性能。事实上，对于题目要求的任务，同学们可以做如下分析：
  - 对于某一个image的元素，他要与多少个core元素相乘？
  - 对于一个core元素，他要与多少个image元素相乘？
  - 能否使一个core元素在cache中停留尽量长的时间，当这个core元素被替换出去的时候，他将永远不会再被使用？
- 如果采用效率提升比例（代码中的统计方式）与cache容量的比，作为cache设计的有效性的话，同学们能否按照上面的提示，设计出一种大大超过**优化顺序版6重循环+clock**的方案呢？希望尝试的同学可以按照下发代码的思路，重新设计大部分内容，并写明设计的思路

扩展内容

- 对于pro4提出的问题，卷积运算是CNN中最大的运算量所在，因此使用专用硬件对其进行加速是一个非常有效益的方向。感兴趣的同学可以尝试利用logisim或Verilog搭建计算卷积的硬件设备。自主设计硬件来达到比pro4还要高数十倍的IPC提升