

# 1 体会 1

本次实验内容为更改现有 CPU 使之能够在我们的开发环境中能够正常工作。实验本身需要的代码量并不是很大，但与自己动手写 CPU 不一样的是——改造现有 CPU 需要阅读别人的程序，完全搞明白别人的代码功能与代码逻辑以及各模块之间的连接关系，某种意义上来说这是比自己写 CPU 更难的一件事情。因为这需要对五阶段流水线 CPU 的原理有一个更好的理解。

这份下发的 CPU 文件实现可谓是高效简洁，CPU 部分仅仅使用了一千余行就完成了 MIPS45 条指令五阶段流水线 CPU，而我上学期的 MIPS54 条指令动态流水线 CPU 的实现却用了将近 3000 余行，很显然如此高效简洁的代码很有值得我学习的地方：

(1). 流水段信号使用：两个流水段之间的流水寄存器可以通过一个信号来实现。在这份代码中，流水线之间需要传递的信号被合并到一起直接传递到下一个流水段上，与我自己实现的代码比起来，省去了五个流水寄存器的定义、调用和接口的传递性赋值，这大大的减少了代码量、减少了信号的定义量。当然，这一做法也一个问题，那便是代码的阅读难度提升了——这份代码中经常使用合并信号的某几位作为需要的控制信号，这就使得意义不是十分的直观，需要提前定好类似于“协议”一样的东西以保证使用的正确性。

(2). 宏定义使用：宏定义的使用方便程序的修改与移植。在这份 CPU 中，使用了不少宏定义，这使得修改一些重要参数(如指令存储器的起始地址、指令的标号)等都十分方便。在我自己实现的 CPU 中，除了对指令的定义意外没有使用到宏定义而是直接在程序中写死，这就使得更改某些重要参数不太方便——修改需要查询源程序的每一处以保证每个使用的地方都被修改，而且需要保证仅仅修改想要被修改的常量而不是那些值相同意义却不同的常量。

(3). debug 艺术：不管是软件程序还是硬件程序，debug 都是一门艺术。对于硬件程序，直接下板验证、波形图仿真验证、逻辑分析仪验证等都是行之有效的调试手段，合理配套使用能够极大地帮助自己寻找问题所在。例如在本次大作业中，在进行各种信号输入输出选择时，就十分依靠 vavido 的仿真波形，为了能具体了解模块内部运行细节，比如 ALU、Pcreg、Regfile、CP0、Hi、Lo 等数据的变化，就可以在不影响模块功能的情况下，将一些内部信号也输出处理，这样使得模块透明化，便于直观观察，尽快找出问题所在。

通过此次的 CPU 设计实验，让我对 CPU 内部组成以及指令在 CPU 部件上如何运作有了一个更深的理解。在设计多周期 CPU 的过程中，感觉慢慢可以理清思路，并且对现实生活中 CPU 的快速运作有了更深的认识。也明白了下一步需要设计的东西通过此次实验，让我们对 CPU 有了更深的理解，而不只是纸上谈兵。

## 2 体会 2

在本次实验当中，首先体会的比较深刻的地方就是要选对正确的方向。

一开始，自己还打算修改自己的 cpu54，但是发现实在是工作量过于庞大。在思考，和同学们进行讨论之后，决定修改自己动手写 CPU 的程序，使其能够下板完成任务。

在编写代码的过程中，我深刻的意识到了合理的缩进的重要性，不知道是由于反复粘贴还是怎么回事，其实自己动手写 CPU 当中的代码并不是很清晰，起码缩进有时候乱七八糟的，也不知道是不是因为整个结构比较复杂，导致 vscode 不能正确的表示出 begin 和 end 的对应关系，在捋清楚整个 CPU 的工作过程上，我还是花了不少时间的。

此外，我感受到了有经验还是十分重要的，在我基本上完成了工作之后，很多同学来问我问题，我都比较好的能够解决，并且能比较迅速的帮助同学们找到正确的思路。想到自己如果编写的时候也有人指导，或许就不用在凌晨对着 Vivado 当中的波形逐个观察，逐指令的调试寄存器当中的值，并和 Mars 相比较。

编写测试程序也是重要的，敢于测试，面对错误，才有可能能够解决。

最后，我认为整个作业从后面开始，就会变得比较困难了。由于《自己动手写 CPU》当中使用的开发板的型号似乎与我们不同，同学们也许很难沿着这本书上的路径走下去。而至于隔壁班想要移植 Ubuntu，我也认为很难。移植一个操作系统是很困难的事情，由于前面罕有前人，我可能甚至找不到合理的 debug 的方法，更不清楚操作系统需要什么指令来让我实现，我觉得从写 CPU 到这一步，期间所需要迈过的东西似乎是难于登天的...更何况我们甚至要求实现简单的 GUI 界面。操作系统从开机开始，就有进程调度等事务。一旦 CPU 出现问题，可能甚至无法开机。对于大家来说，也许只能寻找 github 上是否有成功的移植经验，并加以学习了。总之我还是认为，第一次移植操作系统，能开机就算成功。

## 3 体会 3

在本次 cpu 改造实验中，我在之前带 2bit 分支预测器的 54 条动态流水 cpu 的基础上扩充指令至 89 条，完善了协处理器 CP0，实现时钟中断。在改造过程中，我进一步加深理解了流水线技术的原理，对于不同类型指令的理解也更加深入了。本次实验中的重点在于时钟中断的实现，主要是修改 cpu 的控制器模块和 cp0 模块，参照中断指令 break 实现了发生时钟中断时向时钟中断例程的跳转。在本次实验中，我还对动态流水 CPU 的代码进行了部分重构与修复，包括：将 cpu 更改为小端模式以适应 MARS 的测试，优化陷入指令，修复了移位指令等，提高了 cpu 的稳定性和代码的易读性。总而言之，经过这次实验，我进一步提高了我的硬件编程技能，给今后的更加复杂的硬件设计（实现系统 wishbone 总线和相关硬件控制器）积累了宝贵的经验。

## 4 体会 4

总的来说，本次实验是参考《自己动手写 CPU》前 11 章内容和示例代码的基础上完成的；由于过往 54 条 MIPS 指令集 CPU 的设计和参考书相似，因此大致可以将扩展工作分为如下的几个部分：

- 部分扩展指令集（MOVN, NOP 等）只需要增加相应的指令译码过程和部分控制信号即可；
- 但有些扩展指令集（MADD 等）需要按照参考书的设计思路重新构建数据通路组件，来执行两阶段的“先乘后加”类型的指令，其中翻来覆去的过程还是挺痛苦的；
- 同样的部分扩展指令集（B, TGE 等）是在已存在同类型指令部件的基础上进行修改，比如 B 系列指令只需要增加判断控制信号和相应代码逻辑即能完成扩展；
- 但类似的扩展指令集（LL）却由于 LWL 系列指令的繁杂需要大量重构数据通路，这里我为了未来添加总线方便而直接沿用了参考书中的编写方式；
- 另外，由于 CP0 部件并未过往进行过太多测试，这里我也同样地直接沿用了参考书中的 CP0 设计结构，并且最终按照测试文件的规定修改了部分代码；

同时，在经历了若干个学期的项目调试工作，本次实验的前仿真调试和下板验证过程都较为顺利，出现的许多错误也都大致能够参考过往经验来解决；

另外，本次实验我将 Vivado 从 2016.2 版本升级到了 2019.2 版本，全新版本的界面确实更好看了，但也因为我电脑 C 盘剩余容量不足的原因而导致可以分配给 Vivado 进行前仿真的 Cache 文件夹大小捉襟见肘，前仿真过程可能一不注意就会因为 inspect 的变量过多而导致 Vivado 闪退，多少也算是一个比较头疼的问题。

## 5 体会 5

本次实验整体上来说并不是很难，主要就是围绕着 35 条指令的添加开展，实验的开展过程也显得颇为枯燥，从我个人的观感来看，我这几天已经把 ID、EXE、MEM 和 WB 模块来来回回地改了快二三十遍，每家几条指令就要重头改一遍，还要配置好对应的系统结构，这一过程属实让我体会到了“枯燥”的意义所在。

诚然，不带脑子的机械劳动，导致的结果就是在测试的时候状况百出，最让我头疼的是 func\_test4，我从晚上十二点开始 debug，一直 debug 到了早上七点半还是没有 de 出来，要不是看孔同学起床抱着电脑找他指点了一番，我感觉我可能这次实验就卡死在 func\_test4 上了。

但实打实地讲，这次实验的过程中，我还是颇有一些惊喜的，比如在 LL 和 SC 指令的添加过程中，我第一次从硬件的角度理解了上学期操作系统课程所讲的信号量的处理；还有之后的自陷指令，我才真正意义上明白了操作系统所讲的中断在 CPU 硬件上是怎么实现的。

总的来说，通过这次试验，在一定程度上我是感受到了操作系统作为软硬件分界线的存在的，也是第一次用这种实际参与的方式体会到这种软硬件的交界面所在，纵使本次实验还是令人感到枯燥乏味，但能看到这最后的成功的喜悦，以及过程中能体会到一点融会贯通的味道，也算是颇有收获吧。

P.S. 希望这是最后一次在 CPU 里面加指令，我这辈子都不再往哪个动态流水里加指令了，简直是折磨。

## 6 体会 6

在 54 条 MIPS CPU 的基础上，我花了几周时间逐步完成了这次实验，在完成实验的过

程中，我遇到过很多的困难，为了解决这些困难，我学习了很多知识，查阅了很多资料。在解决这些问题的过程中，不仅让我解决了实验中所遇到的问题，最终完成了此次实验，也让我收获了很多。

最初，只是对 54 条指令比较熟悉，对 CPU 的协处理器、异常处理有一些粗浅的认识，于是通过《自己动手写 CPU (雷思磊)》、《计算机原理与设计—— Verilog HDL 版》等书籍学习了一些相关的理论知识，也查阅了很多网上的资料，了解了如何实现时钟中断等功能，才开始上手此次实验，在上手编程后，才发现将理论编程代码会遇到重重困难。例如，对相关算法的实现中编写的代码不够严谨，存在漏洞，带来了种种问题。

除了在设计 CPU 时遇到的问题，在最后验证的时候也遇到了很多问题，这其中的很多困难困扰了我很长时间。在后仿真和下板的过程中，总是遇到很多莫名其妙的问题，在不断地解决后，达到了实验验证的目的。

虽然解决这些问题的过程中花费了我很多的精力，但同时我也收获了很多知识和技能，例如：

- ①在实践的过程中，我丰富了相关的理论知识。
- ②在将理论付诸实践的过程中，我对 CPU 的理解更加透彻了。
- ③在使用 Vivado 等工具 Debug 的过程中，极大地提高了我对相关语言和工具的熟练掌握程度，可以帮助我更高效地完成后续的实验。
- ④在仿真的过程中，我学会对 ModelSim 的更多应用。在查看子模块变量值、计算周期数、指令数等过程中，体会到了 ModelSim 丰富和强大的功能，对这些功能的熟悉使用，是我在技能上很大的收货。

总之，这次实验给我带来了很多困扰，花费了很多精力，但也收获颇丰，相信在这次实验中收获的知识和学习的技能在未来我这门课的学习、甚至生活和工作中仍然发挥着作用。

## 7 体会 7

一是加深了对 mips cpu 系统结构和各种冲突解决方式的理解，上个学期做的时候对数据前推等冲突解决方式的理解还不深入。这学期在做 ll, sc 指令时出现了对 ll bit 的先读取再根据值访存的情况，需要考虑到 ll bit 的读写主要放在写回阶段，sc 指令需要在访存阶段

就做出是否存储的决策；第二个情况是 sc 指令紧跟 ll 指令引起的 RAW 数据冲突的解决，加深了我对整个系统结构设计的理解以及为什么要这样设计。

二是上个学期做动态流水线的时候还没有学习操作系统，在学操作系统的时候也对 PV 操作下达到指令层究竟应该怎么实现很感兴趣。在这学期搭 cpu 的时候着重关注了相关指令，还有异常相关的指令里有些是跳转当前指令，有些是跳转下一条指令，这样的处理和操作系统中对异常和中断的处理结合起来了。

## 8 体会 8

经过本次 89 条 MIPS 指令动态流水线 CPU 的设计，我们在 54 条 MIPS 指令 CPU 的基础上更加深入地了解到了 CPU 的原理以及 MIPS 操作指令集。

使用 Verilog 扩充设计 89 条 MIPS 指令 CPU 的过程，我们不光复习巩固了自大二以来的硬件设计的一般方法，还通过亲自实现，对 CPU 和 MIPS 指令集有了更深的了解。当然，本次实验设计，也暴露出了自身存在的一些问题。下面逐一分析。

首先是之前 54 条 MIPS 指令的动态 CPU，当时在编写的时候，没有使用类似 define.v 的文件对总线条数总线宽度等信息进行统一定义，在再次改写的时候，曾不止一次将重要的线路信息写错（如[31:0]打成[32:0]或[31:9]）。虽然我们使用的是 4 字节的宽度（即 32bit），但不排除今后还需改写扩充的可能。因此更加规范的文档和编程习惯显得极其重要。关于 MIPS 指令集 CPU 的大端小端问题，以前并没有关注过，一直认为 MIPS 指令集 CPU 应该是小端的。通过这次数据存储指令的编写和书中大端代码的学习，我们对大端存储和小端存储有了更多的了解，并且能够将大小端存储进行互相改写。并深刻认识到大端存储和小端存储在存储数据的差别以及互通。

## 9 体会 9

本次实验前期由于不太明确实现目标有点摸不着头脑，但随着老师对测试要求的不断明确和同学们的讨论与合作努力，发现其实并没有想象中的困难。特别感谢李同学和孔同学为

我们提供的大端测试代码以及在实验过程中的一些提示和指导，非常有用。

另外 Vivado2020 相比 Vivado2016 在综合速度上有一个比较大的提升，能在很大程度上提升 debug 的效率。同时仿真其实比下板重要许多，下板一次 5min，找一个错又得重来，相比之下 5s 的仿真要灵活许多，同时可以查看的变量也更多，是更优秀的 debug 方式。

本次改造 CPU 可以算是基本成功，希望在之后移植操作系统的过程中能和大家充分的合作讨论提升开发效率，最终能做出一点成果，也为下一级的同学提供一些参考。

## 10 体会 10

### 1、实验体会

本次实验将 54 条 CPU 完善到 89 条，收获颇多。书中对于数据存储器、存储相关指令的设计和分析十分巧妙，让我受益匪浅，还有像乘累加 MADD、MADDU，乘累减 MSUB、MSUBU 指令，它们需要在指令执行阶段花费多个周期进行计算，但过程又和除法 DIV、DIVU 的实现有所不同，在实现这些指令的过程我学到了许多。同时后面的指令设计中断处理，增加了时钟中断等，进一步增加了异常相关指令，完善了异常处理的结构，同时最后由于 MARS4.5 中无法执行异常相关的代码，需要自己在仿真测试中自己逐个指令查看，也加深了对于异常处理的指令、结构的理解。本次老师设计的测试文件用着十分舒服，也让我学习到设计测试文件的一些方法和格式，有助于日后可能需要的一些测试。这里记录和分享一些在实验过程中遇到的一些问题和解决方法。

### 2、实验遇到的问题和解决方法

#### 1) 问题一

问题：

使用上学期写好的 54 条动态流水线 cpu，执行第一个测试 test1，仿真结果没有问题，但是下板结果不对。执行上学期的 coe 文件，仿真结果和下板结果都正常。

解决方法：

由于仿真时是没有问题的，所以只能通过下板把中间值打出来进行调试。将中间的多个寄存器值显示在数码管上（通过开关选择要显示的寄存器），测试之后，发现部分指令执行有误，检查实现代码，并没有问题。最后想到本学期更新了 vivado，从 2016.2 更新到 2019.2，当前项目是在 2016 版的基础上，重新设置 ip 核得到的，所以尝试使用 vivado 2019.2 建立一

一个新的项目，然后把对应的文件复制后添加进去并重新设置 ip 核，之后下板，测试 test1 通过。

### 2) 问题二

问题：

multi-driven net 的问题，按照以往经验应该是对应端口在多个 always 中被赋值，但是本次报错出现在同一行解决方法：

上网查询相关资料，看到有这样的情况：

所以排查 multi-driven 端口后续的信号，最终发现是在 CPU.v 文件中端口接错了注意：multi-driven net 的问题在 vivado 中仅为 criticalwarning，也就是说有这样的情况仍然可以仿真、下板，原先我也没有在意这个问题，但是后来发现 test10 中仿真通过，但是下板结果不通过，才反过来检查这个问题，修改好之后仿真和下板结果一致。所以在实验过程中也要注意 warning 的信息。

### 3) 问题三

问题：

执行到下面指令时，出现问题：

解决方法：

实际上在执行这里的指令时，出现了溢出情况，按照书中的处理，是会有溢出异常的，但是本次实验的测试文件，没有对溢出异常进行处理，所以这里就先不处理溢出情况，把检测溢出异常的代码先注释掉即可。

## 11 体会 11

通过本次实验，我学习了 CPU 的 89 条指令的指令作用以及实现方法，对流水线 cpu 各个模块的架构与功能有了更加清晰的认识。这次实验的过程比较曲折，主要原因是一开始没有确定好自己的方向。最开始想要改造之前写的 54 条指令 cpu，但是由于对指令作用以及与各个模块之间的关系理解得不清楚，导致改动的过程并不顺利。所以最后选择了《自己动手写 CPU》这本书作为指导，在实验的进行过程中，这本书的指导非常重要，书里详细地介绍每个指令的格式、作用、实现方式等，不仅使自己理解了新增的指令的工作原理，还

帮助我理清了之前写 54 条 cpu 时比较模糊的地方。当自己对这些知识有了更深刻的理解后，整个 cpu 架构也更加清晰了。用了很长时间才跳出来的坑主要有三个，首先是测试文件 test10 中 break 指令过不了，最后修改了中断起始地址的位置才通过测试。第二是分频之后，再进行仿真时会出现读不进去指令的问题，一步一步调试后发现是写 pc 时的问题，分频之后无法保证 clk 和 rst 的上升沿重合，导致 pc 无法初始化，稍微修改一下 cpu 的架构，加一个顶层文件就可以规避 clk 和 rst 周期的问题。最后，下板之后的结果和仿真的结果不一样，这个地方猜测可能的原因是由于分频过小导致读指令出现问题，改大分频即可。

总而言之，本次实验增加了自己对 cpu 与计算机系统结构相关知识的理解，提高了自己 verilog 编程的能力以及解决问题的能力。

## 12 体会 12

(1) 本次实验完成了 89 条指令 CPU 的改造。前期花了较多时间进行参考代码和参考书的阅读，然后再进行 CPU 代码的修改与仿真、下板调试。深刻体会到了如果原理不够清楚，编写代码将会十分困难。

(2) 本次实验是第五次进行 CPU 设计，从最初的 31 条指令 CPU，再到扩展指令数目到 54 条，后来到实现静态流水线、动态流水线 CPU，一路走来不断修善自己的代码，再到本课程开始阅读官方给出的教程，修改完整的 89 条 MIPS 指令的 CPU，一开始对 CPU 比较迷茫，虽然也画出了通路图、指令控制信号表，但总感觉理解还是不够清晰。后来随着时间推移和知识的学习，不断地复习、学习新的知识，一点点加深对 CPU 的理解。

(3) 硬件课程确实是不容易，真的需要花功夫。一味地抱怨难、要求高是没有用的，与其抱怨，不如脚踏实地做，迈开步子总比不迈强。而且很多时候，计算机系统实验课程报告硬件的调试时间特别长，往往修改代码中一个字符，重新综合下板就要十多分钟，然后还是错的，又要修改，周而复始。但是我觉得，要沉得住气，不要怕困难，正常做，等待的过程可以做点别的事情，不能因为一两次失败就放弃了。

(4) 在这里感谢班级的同学分享的测试文件，也感谢在群里分享测试文件修改思路的同学，是大家共同的智慧使得我们能完成 CPU 的改造。下次实验依旧艰辛，但我还是会努力尝试，加油！

## 13 体会 13

经过本次 89 条 MIPS 指令动态流水线 CPU 的设计，我们在 54 条 MIPS 指令 CPU 的基础上更加深入地了解到了 CPU 的原理以及 MIPS 操作指令集。

使用 Verilog 扩充设计 89 条 MIPS 指令 CPU 的过程，我们不光复习巩固了自大二以来的硬件设计的一般方法，还通过亲自实现，对 CPU 和 MIPS 指令集有了更深的了解。当然，本次实验设计，也暴露出了自身存在的一些问题。下面逐一分析。

首先是之前 54 条 MIPS 指令的动态 CPU，当时在编写的时候，没有使用类似 `define.v` 的文件对总线条数总线宽度等信息进行统一定义，在再次改写的时候，曾不止一次将重要的线路信息写错（如`[31:0]`打成`[32:0]`或`[31:9]`）。虽然我们使用的是 4 字节的宽度（即 32bit），但不排除今后还需改写扩充的可能。因此更加规范的文档和编程习惯显得极其重要。关于 MIPS 指令集 CPU 的大端小端问题，以前并没有关注过，一直认为 MIPS 指令集 CPU 应该是小端的。通过这次数据存储指令的编写和书中大端代码的学习，我们对大端存储和小端存储有了更多的了解，并且能够将大小端存储进行互相改写。并深刻认识到大端存储和小端存储在存储数据的差别以及互通。

## 14 体会 14

本次实验主要进行了 MIPS 89 条指令的动态流水线 CPU 设计，了解了动态流水线 CPU 设计的具体流程与方法。在设计过程中对计算机硬件了解更加深入。

指令流水线的设计的关键就是要将整个 CPU 的模块分为 5 个部分，分别为 IF, ID, MEM, EXE, WB。然后逐步实现各级部件，相邻部件之间用指令流水线连接。

在整个 CPU 设计完成后，需要进行大量数据的检测验证，以保证 CPU 的功能正确性。特别是态流水线中会发生的各种冲突，在动态流水线中，要通过数据前推机制进行解决。

相比于之前实现的 31 条指令单周期 CPU 和 54 条指令多周期 CPU，这次需要考虑的东西更加细致，要准确考虑到信号需要传到哪一级并使用。

硬件设计在当今计算机界十分重要的环节，作为软件的承载体，硬件设计要考虑到性能、功能、规范、体积等诸多因素。现代计算机的 CPU 指令和功能比这次实验所设计的要复杂得多，但是这次实验让我们了解了静态流水线的设计方法。在这个过程中，我受益颇多，知

识和技能得到进一步提高。

由于后续要对该 CPU 移植操作系统，所以 CPU 功能的完备性和正确性要得到充足的保障。

对于完备性，主要体现在指令要实现齐全。这次从 54 条指令扩充到 89 条指令，不仅仅是加指令那么简单，而是要对整体结构进行优化设计。其中最困难的部分应当是异常处理的逻辑了。后续要移植操作系统，那么异常这一部分必须要实现，而且要保证异常功能能正常进行。异常主要会对协处理器 CP0 进行处理，使得这次对 CP0 进行了大改。

对于正确性，这是更加困难的一件事情了。这次实验将 54 条指令扩充到 89 条指令，改代码花的时间倒不算很多，大量时间都花在 debug 上。我花了整整 3 天时间才将我的 CPU 修改到能通过测试。硬件测试不同于软件，它只能通过波形观察测试，这是一个十分繁琐的过程。当发现一个错误后，要定位到那个错误是一件挺折磨人的事情。由于这次涉及到的指令过多，又要一条条保证正确性，所以测试工作十分繁重。但是，没有一个人能保证他所设计的程序百分百正确无 bug，因为我们的测试永远无法百分百覆盖。我们能做的就是找一些容易出错的测试代码，在测试的量上做足功夫，来使得我们的 CPU 趋近于百分百正确。

人们会说，制造 CPU 芯片的光刻机是“人类智慧的结晶”。我觉得，设计 CPU 这样一个如此精巧的硬件，又何尝不是“智慧结晶”呢？

## 15 体会 15

通过这次实验，对流水线 CPU 有了进一步的认识和理解，也对新增加的 35 条指令有了一定的了解，这次实验最主要的过程是参照给出的测试程序，对代码进行 debug，这个过程很花费时间，但是也进一步认识到了指令的功能、执行过程等等，除此以外，进一步的掌

握了 vivado 的仿真工具，给测试带来了极大的便利，可以查看到所有的中间过程的变量，并根据变量的值的变化以及 MARS 的执行过程确定哪一步有问题，非常的方便。

总的来说，CPU 的实现是非常复杂繁琐的一个过程，从最开始的 31 条单周期指令的实现到现在 89 条指令流水线 CPU 的实现，没有想到最后真的能实现，但整个过程很痛苦，特别是使用了 vivado、modelsim 并不经常使用的工具，在报错和一些功能的使用上仍然不太清楚，甚至有些时候都不知道为什么这样做能成功，这样写就会报错，而且，底层的

硬件逻辑的实现上，和我预想的很不一样。感觉 CPU 将大家联系在了一起，不论认不认识，看见是同学，都会讨论两句 CPU，也确实在逐渐讨论的过程中，有了新的认识，也解决了很多问题。