|  |  |  |
| --- | --- | --- |
| 南农大  **计算机组成原理课程设计**  XH2 | | |
|  | 题 目: | 基于Verilog的32位MIPS指令集五级流水CPU的实现（个人项目）及Cache(小组中的工作) |
|  | 姓 名: | 邱日 |
|  | 学 院: | 信息科学技术学院 |
|  | 专 业: | 计算机科学与技术 |
|  | 班 级: | 计科151 |
|  | 学 号: | 19215116 |
|  | 指导教师: | 赵力 职称: 副教授 |
| 2017年 11 月 29 日 | | |
| 南京农业大学教务处制 | | |

**目录**

**1 课程设计实验目的**

1.1实验目的

1.2问题描述

**2 课程设计主要思路概括**

2．1 主要设计思路

**3 功能模块划分以及具体的设计思路**

3．1 小组中的工作

3．1．1 指令Cache模块

3．1．2数据Cache模块

3．1．3 Memory模块

3．2 个人的工作

3．2．1 取指ifetch模块

3．2．2 译码idecode模块

3．2．3 执行部件EU模块

3．2．4 存储Memory模块（包括写回到寄存器）

**4 心得体会**

**5 参考文献**

基于Verilog的32位MIPS指令集五级流水简单CPU的实现(个人项目)及Cache（小组中的工作）

计科专业学生 邱日

指导教师 赵力

# 摘要：

本课程设计我参与小组共同完成编写32位MIPS指令集CPU的工作,在小组工作中，我负责ICache、DCache及Memory模块的开发并编写相应testbench验证上述模块，在老师的指导下，我的这些模块成功通过测试且可独立检验，正确性可以保证。在整个小组的开发不能继续下去时，我另起炉灶，本着尽量简化的思想重新设计，从头编写了32位MIPS指令集的简单CPU的所有模块。由于前期耗去很多时间，后者没有来得及联调，但基本已完成，这个工作是可以继续完善的。

**引言：**

本系统利用硬件描述语言Verilog开发，采用哈佛结构，指令集MIPSR2000，尝试用最少的代码来实现一个非常简单，但“五脏俱全”的CPU。根据MIPS经典五级流水线，取指、译码、执行、内存读写、写回到寄存器Fetch,Decode,Execute，Memory ,Writeback。根据五级流水线，我将模块划分为ifetch取指模块，idecode译码模块（在实现上，此模块中包含了32个寄存器）、EU执行模块、Memory模块（包括了写回）、顶层模块StupidCPU。之前在小组中的工作中，我在赵老师的耐心指导下完成了ICache、DCache和Memory模块。由于老师的指导加上暑假时间充裕，对于在小组中的工作我编写了相应的测试程序，通过测试，可以确保其正确性。开学之后，课业繁重，我真的“学无余力”，也没有足够的时间了，对于个人写的CPU,还有待完善。

**1 课程设计实验目的**

1．1 实验目的

采用硬件描述语言，动手实现基于MIPS指令集的32位CPU。理论联系实际，加深对计算机组成原理的理解。

1．2 问题描述

1）硬件描述语言选择Verilog,开发工具选用iverilog和gtkwave,在Linux平台完成指令高速缓存模块ICache、数据高速缓存模块DCache及Memory的模块的编写及测试。

2）选用以上开发工具，完成32位处理器编写工作。

**2 课程设计主要思路概括**

2．1 主要设计思路

开始我是小组中的一员，小组共同完成32位基于MIPS指令集的CPU的编写。我在小组中负责Cache部分的编写，在老师的耐心指导下，我完成了指令Cache、数据Cache及Memory模块的编写，设计了一个比较直观的方案去测试这些模块并成功通过测试。后来小组项目遭遇困难，我费了很大力气尝试整合大家写的模块，但没有成功。正如老师指出的那样，小组技术路线存在很大问题。项目初期大家急于求成，在没有一个成熟设计方案的情况下，过早地将系统分模块了，写完后也不是每个模块都进行了单元测试，这导致系统整合和测试无法进行。

Dijkstra的名言“有效的程序员不应该浪费很多时间用于程序调试，他们应该一开始就不要把故障引入”影响我很深，在我日常写代码的过程中，如果遇到一个bug调试了较长时间没有得到解决，就说明这一部分开始编写时头脑中就没有清晰概念，设计是不合理的，我将会忍痛把这段代码全部重写，而不是在之前的基础上继续找bug。有时候我可能要重写五六遍，但这样可能要比在错误基础上调bug的时间少。

当小组的整合工作无法进行，我索性放弃所有代码，从头再写。这时候时间已经不是很充裕，个人想要完成四个人之前的目标，难度实在太大了。这时，我想到，小组失败的原因，很大程度上在于最初定的目标太高，实现时眼高手低，系统的复杂也给理解带来麻烦;这次到我独立实现，不必再追求性能上的高速度，能简化的都简化，所有与核心部分影响不大的都去掉，这样或许能尝试一下。比如说，虽然Cache部分很重要，但它对整个系统逻辑上的影响并不大，也就是说没有它系统照样存在，于是就把Cache去掉。之前小组中的陈尧负责做先行进位的运算器，但若直接用运算符，性能虽不如前者，但也能成立。这样降低要求之后，要实现之前的32位流水线CPU,几百行代码也能完成核心功能。实际上，我也用几百行简单的代码大体完成了整个系统，测试了部分指令，但要完善它时间实在来不及了。

**3 功能模块划分以及具体的设计思路**

3．1 **小组中的工作**

先谈我在小组中的工作，我在小组中完成了指令Cache、数据Cache和Memory模块

3．1．1 指令Cache模块

要编写Cache模块，不能凭空去写，必须要确定Cache和主存的容量等参数，开发板的信息没能找到，我征询了大家的意见，并请教了老师以后，确定了Cache和Memory的容量，在本系统中，Cache容量8KB,主存容量128MB,块大小我定的是8个字，字长是32bit,映射方法采用最简单的直接映射。

按字节编址，每块8个字，每个字4个字节，也就是32个字节,块内偏移 w = 5. 主存容量128MB(2^27B)，主存地址我使用27位。主存地址由14位标志(Tag)、8位行地址（line）以及5位的字地址（Word）组成。  
主存地址 （s+w=27位）： (s-r) + r + w = 27 bit

主存地址格式如下：

| **标志Tag** | **行地址Line** | **字地址Word** |
| --- | --- | --- |
| （s-r)=14 | r=8 | w=5 |

Cache 地址 （8+5=13位） r=8 w=5

Cache地址是13位,Cache地址由8位的行地址（line）、5位的字地址组成。Cache地址格式如下：

| **行地址Line** | **字地址Word** |
| --- | --- |
| r=8 | w=5 |

PC是30位,对应32位地址线的高32位:A31~A2.因为每次访问的必须是32bit,即4个字节，（也决定了只能访问偶地址，且要是4的整数倍）故地址线的低两位A1=0,A0=0。由30位PC可以扩展成32位地址。

3．2数据Cache模块

此部分和指令Cache功能上有重叠的部分，但需要写回。我设了一个来控制位数组来判断每块是否装入。reg control[CACHE\_BLOCK\_NUM-1:0]; //控制位，其中 第0位 为装入位，0未装入，1已装入，需要初始化。RW表示当前的操作是写还是读，RW=1表示是读，否则是写。根据RW的不同采用不同的方案。

核心代码：

always @ ( posedge clk or reset) begin

|  |  |
| --- | --- |
|  | if(RW)begin |
|  | for(i = 0;i < 30; i++) |
|  | begin address[i+2] = PC[i]; address[1:0]=2'b00;end |
|  | index=address[index\_bits+offset\_bits-1:offset\_bits]; //8位索引（行号） |
|  | offset=address[offset\_bits-1:0];//5位块内偏移 |
|  | blockNumber=address[tag\_bits+index\_bits+offset\_bits-1:offset\_bits];//22位 |
|  | tag = address[tag\_bits+offset\_bits+index\_bits:offset\_bits+index\_bits]; |
|  | hit =0;//本次有没有命中，先初始化，默认没有命中 |
|  | if(control[index]) |
|  | begin |
|  | //如果已经装入并且命中 |
|  | //开始犯了CacheData没有初始化的错误 |
|  | CacheBuffer = CacheData[index]; |
|  | case (offset[5-1:2]) //4 3 2 //1 0 |
|  | 0: ibus[31:0]=CacheBuffer[32\*1-1:0]; |
|  | 1: ibus[31:0]=CacheBuffer[32\*2-1:32\*1]; |
|  | 2: ibus[31:0]=CacheBuffer[32\*3-1:32\*2]; |
|  | 3: ibus[31:0]=CacheBuffer[32\*4-1:32\*3]; |
|  | 4: ibus[31:0]=CacheBuffer[32\*5-1:32\*4]; |
|  | 5: ibus[31:0]=CacheBuffer[32\*6-1:32\*5]; |
|  | 6: ibus[31:0]=CacheBuffer[32\*7-1:32\*6]; |
|  | 7: ibus[31:0]=CacheBuffer[32\*8-1:32\*7]; |
|  | default: ;//已经穷举 |
|  | endcase |
|  | hit =1; |
|  | end |
|  |  |
|  | else |
|  | begin |
|  | //从主存里的块来替换 |
|  | CacheData[index] [32\*8-1:0]= MemoryBlockWire[32\*8-1:0]; |
|  | CacheBuffer = CacheData[index][32\*8-1:0]; |
|  | control[index]=1; |
|  | hit =0; |
|  | end |
|  | // $display("取到"); |
|  | end |
|  |  |
|  | else begin //如果不是读，那就进行写操作 |
|  | for(i = 0;i < 30; i++) |
|  | begin address[i+2] = PC[i]; address[1:0]=2'b00;end |
|  | index=address[index\_bits+offset\_bits-1:offset\_bits]; //8位索引（行号） |
|  | offset=address[offset\_bits-1:0];//5位块内偏移 |
|  | blockNumber=address[tag\_bits+index\_bits+offset\_bits-1:offset\_bits];//22位 |
|  | tag = address[tag\_bits+offset\_bits+index\_bits:offset\_bits+index\_bits]; |
|  | hit =0;//本次有没有命中，先初始化，默认没有命中 |
|  | // if(control[index]) |
|  | begin |
|  | //如果已经装入并且命中 |
|  | //开始犯了CacheData没有初始化的错误 |
|  | CacheBuffer = CacheData[index]; |
|  | case (offset[5-1:2]) //4 3 2 //1 0 |
|  | 0: CacheBuffer[32\*1-1:0]=writeData[31:0]; |
|  | 1: CacheBuffer[32\*2-1:32\*1]=writeData[31:0]; |
|  | 2: CacheBuffer[32\*3-1:32\*2]=writeData[31:0]; |
|  | 3: CacheBuffer[32\*4-1:32\*3]=writeData[31:0]; |
|  | 4: CacheBuffer[32\*5-1:32\*4]=writeData[31:0]; |
|  | 5: CacheBuffer[32\*6-1:32\*5]=writeData[31:0]; |
|  | 6: CacheBuffer[32\*7-1:32\*6]=writeData[31:0]; |
|  | 7: CacheBuffer[32\*8-1:32\*7]=writeData[31:0]; |
|  | default: ;//已经穷举 |
|  | endcase |
|  | CacheData[index] = CacheBuffer ; |
|  | // $display("\nCacheBuffer:%h",CacheBuffer); |
|  | // hit =1; |
|  | end |
|  | end//写操作～ |
|  | end |

3．3 Memory模块

这部分比较简单，该模块定义了主存及其读写逻辑。主存的本体是memory数组，reg [32\*8-1:0] memory[0:4194304];（128MB,4194304=2^22）开头我规定一块8个字，为此有reg [32\*8-1:0] MemoryBlock; MemoryBlock就是主存中对应的8个字。32位的data是该模块送出的数据。

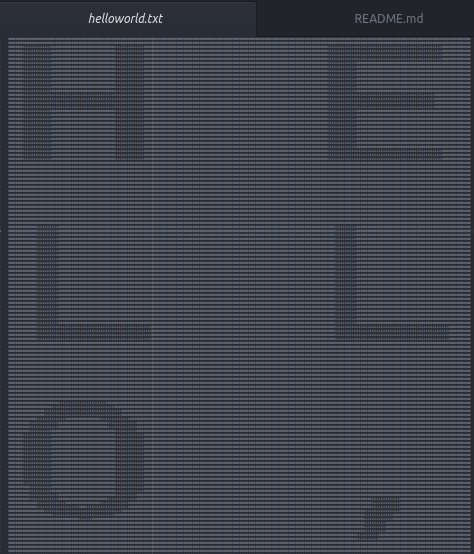
3．4 小组中Cache工作的测试模块

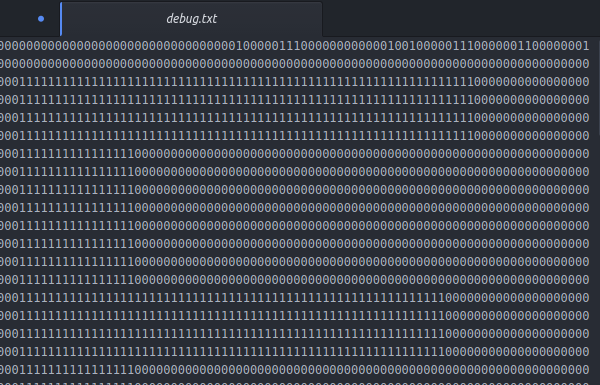
Memory模块的测试模块是QMemory\_tb，ICache的测试模块是ICache\_tb， DCache的测试模块DCache\_tb。

Cache和主存写完了，不过如果不写测试的程序不能保证正确性，因为Verilog描述的是电路，没有语法错误,不代表程序正确。我在Linux环境使用的gtkwave工具可以很方便地看波形，不过，每次一一核对，看得我头晕眼花，于是我就想出了一个简单可行而又直观的测试方法。

这和我的SRT项目有点关系，SRT项目要求我使用Python，学完Python语法我需要小项目来练手，我就给自己出了个题目，写个爬虫去爬自己所有QQ好友的说说和相册存到MySQL数据库。暑假里几乎每一天我都把绝大部分时间用在计组课设上，就个人兴趣而言，组成原理本身是很枯燥的，当我实在不想做它的时候，我就去搞搞这个作为娱乐。有一天看波形看不下去转而写Python爬虫的时候，想起学Python时有个简单有趣的入门项目，图片转字符画，这其实是很容易的，网上也有很多例子。我何不用一张图片来初始化我的主存呢？那样我只要观察我从Cache中读出的数据是否能够组成一个ASCII字符画就能比较容易地看出对错来了。

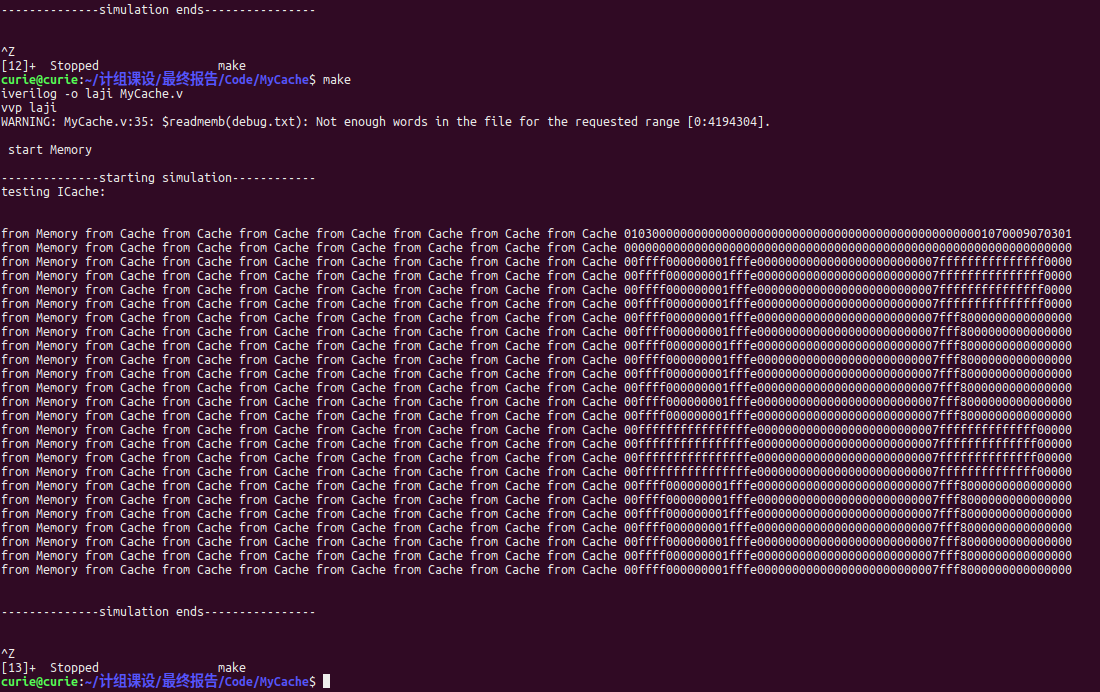
当然，如何用Python转字符画不是重点，如何用它验证Cache的读写是否正确才是目的。我先得到HELLOWORLD的图，然后用Python将它转成我想要尺寸的字符画，这里我得到的是一个名为Memory32\*8\*256.txt的文本文件。注意一下这个尺寸，它的每行是32×8个0或1,因为我是一个块8个字，字长32位。Verilog仿真支持系统功能调用，可以通过$readmemb这个函数从文本读入数据来初始化主存( $readmemb("debug.txt",memory);) 我的文本是256行，实际上，要更多行不是不可以，但没有必要。每次仿真要把128MB的主存都初始化一遍，也是可以的，不过速度太慢，对我测试也没有什么帮助，我就只初始化前面部分主存单元。



HELLOWORLD由01组成的字符画

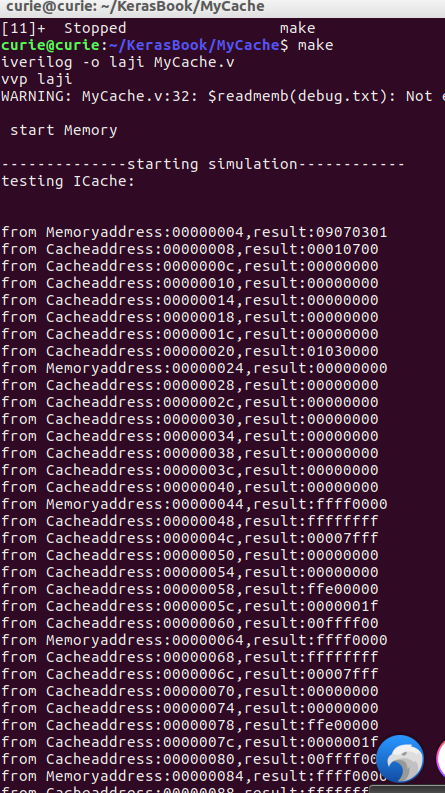
为了方便观察初始的几位,把右上角的数改得特殊一点

测试结果预测：Cache与主存以块为单位交换，每块8个字， 这里地址按顺序增长，先从主存读数据，并调一块到Cache, 这样下面剩下的7个字，可以每次从Cache读一个字（32bit）， 然后又调入新的一块，由此类推。故：一次读主存，隔7次从 Cache读的，然后又从主存读，如此循环往复。



Cache测试结果

可以看到，Cache的确从主存中读出了正确的数值，组成了一幅和主存看起来一样的Helloworld字符画（这里我没有展示出全部，但已经能说明问题），而且也是读一次主存，接下来每隔7次再读一次主存，中间的都从Cache中取数。测试程序中我的地址依次增长，这符合预期，测试通过。

O

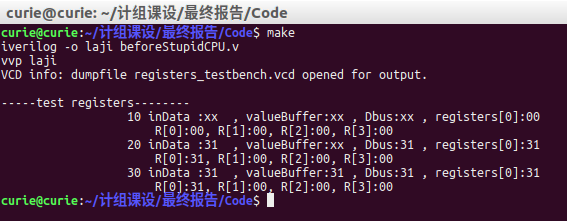
还可以把地址打印出来，这样因为排版的问题，没法看到那张字符画，但仔细比对也是正确的。

3．2 **个人的工作**

谈完了我在小组的工作，再谈我个人的32位CPU工作。前期小组在设计上出了偏差，目标过于宏伟，写出来的代码我只能参考，实际使用的时候不如自己全部从头重写。虽然还是32位CPU,不过我尽量简化，目标比之前小组的小了很多。比如，我也抛弃了我自己写的Cache和Memory,新系统不使用Cache,降低难度;Memory模块因为新的设计重新编写，将写回也融入此模块。对于之前陈尧费了一番力气做的先行进位运算器，也仅用运算符代替。这样做之后，系统得到了很大简化。我发现如果不考虑效率的话，32位CPU或许也能用几百行代码完成。

写一个32位CPU对于计组课设而言是比较大的项目了，之前走了很多弯路，浪费了不少时间。等到确定要我自己来完成整个项目时，时间已差不多了。等我重写所有模块代码，搭起了整个框架，剩下的时间已经不够了。

目前的情况是，已完成取指模块ifetch的编写和测试，包含写回到寄存器的Memory模块的编写和测试，完成取指模块idecode和执行部件EU核心代码的编写。其中idecode和EU要涉及所有MIPS指令，我没有彻底写完，MIPS指令集有R、I、 J这三类指令，只是完成add、sub、load、store等有代表性指令的实现。



写回到寄存器的简单测试

从上图中可以看到数据indata,缓冲valueBuffer和总线Dbus以及部分寄存器在store过程中的变化情况。

**模块功能**

ifetch 模块

取指模块，我采用哈佛结构指令和数据分开这里有一个数组instructionMem，就是指令存储器。

reg [31:0] instructionMem[127:0];

idecode 模块

译码模块,这里还包括了32个32位寄存器

reg [31:0]Register[0:31];//寄存器们

EU模块

执行模块，这里本该把所有MIPS指令都实现，但限于时间关系，未能完成。仅实现add,load和store。

Memory模块

此模块含数据存储器DataMemory ，此模块还包括了写回到寄存器。

reg [31:0]DataMemory [0:127];//数据存储器

**4心得体会**

首先感谢赵老师的耐心指导，感谢小组的其他同学。感谢你们对我的帮助，我也感到荣幸能与大家合作。

此次组成原理的课程设计，是一个投入多产出少的过程。大家和我走了很多弯路，投入了很多时间精力，最后做出来的成果却远达不到预期，这是失败。在这个痛苦的过程中，我们的系统能力得到了培养，我们对硬件的了解更加深刻，这是收获。

我反思过程中遭遇失败的原因，有一些心得体会。

关于计算机组成原理的课程设计，这次大部分同学是利用Proteus软件来展开设计的。Proteus仿真软件相对于实验箱连电路，已经是很大的进步，给教学带来一个可以验证想法的平台，是很有帮助的。然而，由于硬件的复杂性和并行性，用软件去模拟硬件本来就是一个极为困难而且有很大局限性的事情，这导致了Proteus在使用的过程中出现很多意外的问题。几乎每个使用过它的同学都会有类似的抱怨，“这个工程我昨天打开还能用，今天就运行不对了”，或者是“我完全照着书上连的电路，可是得不到正确的结果”，“我只不过把线挪了一下就不对了”。抛开同学们连线时的失误，从众人反映的情况看，Proteus的确不那么好用。我们可能能力有限，不知道它到底错在哪。

我在使用Proteus时也遭遇这些麻烦，而且由于自身对原理的认识很肤浅，出了错不能很快解决，常常不知道到底是软件的原因还是自己的原因。我想，如果我决定使用Proteus来做，时间将大部分浪费在研究如何使用这个平台上，等到课设完成，我可能是学会了如何使用Proteus这个软件，而不是弄懂了计算机的组成原理。

关于平台的选用，我是很慎重的。实际上，在学习计算机组成原理这门课的时候，也就是今年3月份左右，我就在考虑使用何种平台去完成课设，并进行尝试。有三种可行的方案：一是Proteus仿真软件，这个我实在不想用；二是利用硬件描述语言，借助相应平台来设计；三是用高级语言来模拟。

使用高级语言来仿真，这个可能是最容易完成此次课设的方法。但是我从一开始就不打算走这条路。使用高级语言仿真，并非不需要努力，但是，这种方法是对硬件的建模。模型本身只有好坏，没有对错。它反映的是建模者自己的理解，只要自圆其说就行。硬件本身却是有对错的，0和1是客观的。用高级语言写，或许能提高数据结构水平，但对组成原理的理解帮助不是很大。

实践是检验真理的唯一标准，我认为，只有通过第二条路去搞硬件才能真正让组成原理课设发挥它应有的作用，即更深入地理解底层。

相关的两大厂商有Altair与Xilinx，Altera公司的Quartus工具有几个G,我在校园网下下载几天也下载不下来，当时没有考虑;Xilinx家的Vivado更夸张，有几十G,而且并不是免费的，我也没有条件去用它。硬件描述语言VHDL和Verilog之间，还是Verilog更平易近人一些，于是在三月份的时候我就决定来学一学Verilog,校园网下不了大东西，我最终设法找到了iverilog和gtkwave这对工具，加起来只有5M大小，很轻量，用起来也方便，再加上语法高亮的编辑器sublime，编程环境就有了。Verilog是硬件描述语言，和那些高级语言有本质不同，即它是并行的，最后主要以看波形为验证方法。好在Verilog语法相对简单，我在学习组成原理这门课的时候就初步学会了写testbench,自己实现了32位ALU并完成验证,这个工作也比较简单，我课余没有花多少时间就完成了简易ALU，因为书上类似例子很多，8位ALU和32位ALU也没有什么本质区别。

之后我询问老师到底应当用什么开发工具，老师告诉我，业界用的比较多的是modelsim,并且还在群里发了简单的Verilog代码以作参考。我自己就下了一个modelsim来试试，这个东西也没有什么不好，但是比较大，有几个G，安装好打开后我电脑内存占用也很大，比较卡，我没法再开浏览器查查资料什么的，用起来不方便，于是不久就卸载了。

此后我也没有再学Verilog,直到有一天，陈尧到我们宿舍来聊天，谈到了我可以参加他们小组来合作完成计组课设。有这么一个比赛，这个比赛我们基本没有希望拿奖，但是大赛会提供实验的环境，发放一个FPGA开发板，最后会将我们的Verilog代码烧写到硬件，真正做一个CPU。

FPGA开发板和Verilog，这是最吸引我的地方。真正接触到硬件，这是少有的机会，我一定要把握好。至于参加比赛拿奖，这我从来没指望过，我们不过是普通211的学生，根本没有可能。但是我相信四个人合作，最后是能够做出东西的。另外，我知道，大三的课业很繁重，我将“学无余力”，我可以先利用好比较充裕的暑假时间把计组课设完成，以腾出下学期的时间迎接新的挑战。

我爽快地答应了。关于大赛，我是“明知不可而为之”，人总得尝试一点新东西嘛。当时因为开会和我数学建模的时间冲突，我没有到场听要求，现在看来这是不对的，开始时很有必要弄清楚到底要干什么，不然会很被动。会后同学告诉我有Vivado这个软件并展示给我看那块开发板。

大赛官方给出的Vivado软件真的大到超乎想象，光压缩包就二十多个G,安装空间还要更大。在校园网的网速下，我永远也下不下来，最后只能麻烦同学到网吧帮我们下好，安装起来也很费时。但最要命的是：这个软件如此之大，运行起来很非常非常吃内存。而我自己的老电脑配置很低，根本带不动这么吃内存的软件。实际上，因为我的电脑用Windows比较卡，我早已放弃使用Windows,日常就用Linux，这样才比较流畅。最后我的Cache也是在Linux系统中使用iverilog加gtkwave来做的。

为了完成项目，我不得不到本来就很卡的Windows里去跑极吃内存的Vivado软件，这个过程浪费了我太多的时间。打开Vivado,新建工程，把一段已经写好的最简单的加法器代码复制粘贴过去，运行，把波形图打开来看。这个没有难度，但是却很耗时间，我干等它的进度条不停地转圈，我没有夸张，这个过程真的可能要将近七八分钟！

用Vivado真的卡到我绝望，暑假我花了很长的时间来熟悉和使用它，客观一点说，我花在等Vivado软件上的时间或许已经足够用C语言写完计组课设了。然而这个等Vivado的时间是完全被浪费的。穷则思变，我发现这个Vivado虽然大，并没有给我带来任何好处。我在使用的时候，光压缩包就有20个G的Vivado还不如我之前三月份在Windows下装的只有5MB大小的iverilog+gtkwave这两个非常小的工具。我不如继续用我的iverilog。因为我基本上还是用Linux,这次我在Linux系统装了gtkwave和iverilog。这个大小也大概5MB,用起来毫不占内存，但是真的比二十几个G的Vivado好用。

我认真思考其中原因，大赛官方提供Vivado这么一个超大型软件的意义在于，能够将代码真正下载到开发板，从波形或实际的LED显示屏验证实验正确与否，这就不光是仿真了。Vivado这么大的原因是它要完成布线综合的工作,这才是最耗时的地方。我记得在今年3月20号那天，我曾经询问过老师有没有一款EDA软件，它可以将代码直接转化成电路图。当时老师还没有接触到Vivado,我也觉得不可能有这么厉害的软件。其实，Vivado就是一个这么厉害的软件，我在实际使用的过程中的确看到了代码对应的电路图。

Vivado好比一把牛刀，而我们才刚刚入门Verilog，Vivado是大材小用了。Verilog是一门硬件描述语言。我是这样理解iverilog和gtkwave工具的:iverilog(icarus verilog)是一个仿真和综合的工具，实际上是个编译器，编译出目标代码由相应的Verilog预处理器程序来解释执行，其对应的Linux命令是vvp 那个编译后的目标代码，（vvp即Verilog Preprocessor） 。gtkwave顾名思义，用来看波形的，Verilog在过程语句中$dumpfile出vcd文件，gtkwave可以显示这个vcd文件对应的波形。 Vivado的最大意义在于下载到开发板和综合出详细电路图，据我推测,它的时间主要也就花在这几个地方，对于一个专业的真正搞硬件的人来说这很重要，然而暑假在家，开发板不在我手上，（当然，我也不会用），Vivado的这些功能根本没有用，我花很大的时间代价勉强用Vivado没有意义。

就初学者用Verilog开发一个玩具CPU而言，一个编译器（严格地说或许不能称之编译器）、一个波形查看器、一个语法高亮的编辑器其实已经足够了，能用modelsim这样的集成工具是更好。不过这些EDA软件都普遍的大，一般都是几个G起步，我也尝试下载安装，比起Vivado，的确流畅很多，不过还是很卡。或许有人要和我争论，”我也用的是modelsim，但感觉很好啊，怎么没有遇到你所说那些情况？“。这不是我嫌弃各种工具，而是我真的“用不起”它们。举个例子，我并非不会使用VS,但是我早就把它卸载，现在只用gcc在命令行编译了，这一方面有个人喜好的原因，另一方面，我的电脑实在不支持，开了VS就基本上别想打开浏览器了。我的电脑已经用了五六年，日常使用是没问题的，但容易发烫，在夏天最热的那几天，我其实都不敢打开电脑，怕它烧坏。为了做计组课设，我不得不在暑假最热的天打开电脑里的Vivado,即便开了排风扇，结果电脑还是黑屏了两次，使我很害怕，我总不能因为课设把我的电脑弄坏啊。

配置一个轻量的能够支持我用Verilog的编程环境对我个人来说非常重要，实际上这也是可行的。我在我的Ubuntu（Linux）系统上安装了iverilog和gtkwave,这两个东西总共5MB,非常地小，用起来效率比Vivado高得太多了。iverilog是编译器，gtkwave是波形查看器，缺一个能够提示语法错误的编辑器。Ubuntu上我写代码都在Atom编辑器里写，它是可扩展的，尝试装了一个插件，出乎意料地好用，不仅语法高亮，在代码编写时就错误提示直接定位到出错的行，还有Vivado所没有的自动补全功能。

Iverilog和gtkwave都是命令行工具，要用好它们还需要知道相应的命令应该怎么写，在这个过程中我了解到Makefile如何写。只有写好Makefile（用这两个工具，Makefile中只需要两句话），我才能方便地使用。

iverilog+gtkwave再加上Atom编辑器与Verilog相关的插件，这个编程环境让我十分满意，我终于可以不用坐在哪里干等七八分钟才能看到代码的结果了。实际上，网上并没有谁给出这么一个解决方案，这个环境非常轻量，很方便，装起来也没有难度，但为了找到这么一个方法我摸索了很长时间，走了很多弯路。

饱受Vivado之苦的，不止我一个，小组中的王昱程也受不了Vivado这么慢的仿真速度，费了很大力气装上Vivado，但打开几次后受不了，改在记事本里写代码，然后让陈尧在电脑上运行帮她改语法错误。这个行为很可笑，她这样做和在纸上写代码没什么区别，但我知道她也是没有其他办法了。陈尧的电脑配置很高，是我电脑配置的几倍，不过我了解到他也受不了Vivado而换了另外一个替代品。我的环境很方便，开学后我与大家分享，两个女生因为没有Linux系统作罢，我教给了陈尧，他立即愉快地换用了新环境，等他足够熟悉，他还是用的传统的modelsim。

总之，大家开始被Vivado浪费了许多时间，换用其他环境之后才得到解决。之后仿真平台的问题不再困扰我了。

小组中当时给我分配的任务是做Cache。Cache这个工作听起来似乎不难：不就是数组的变换吗？而且还是最简单的方法，没有用其他那些复杂的替换算法。但是执行起来却有极大的难度，因为不应该首先做Cache，而是应当等整个CPU都做好，再去做Cache,那样难度不会有多大，应该是水到渠成的事情；另外在学生的项目中，其实做了一个功能完善的CPU也就差不多了，只有很少人会在做完CPU之后再去做Cache，我能参考的资料极少。

等我做完了Cache,我又读了老师发在qq群中《美国一流大学计算机组成与系统结构实验课程研究》的论文，这篇论文中谈到Stanford University对计组课设这门课的建设，首先它面向的对象主要有两类人群，一是CS专业也就是偏软的计科的，二是EE专业也就是偏硬的电子工程系。其中，它对电子工程系的在这门课的要求是这样的：

“EEl08作为EE108a的后续课程,强调系统的设计和实现,要求学生实现一个MIPS的微处理器，并在上面编写汇编程序,完成一个乒乓球的游戏,用FPGA实现。实验开始给出一些例如Instruction、Fetch、Decode、ALU、Register、File、VGA等基本模块,要求学生在这些模块上实现冲突检测和转(forwarding)等功能。最后要求学生加入高速缓存(Cache)模块,以实现一个直写方式(write—through)的数Cache(Data)并能统计Cache性能的模块。对于整个处理器系统的测试,实验要求学生编写一个乒乓球游戏的MIPS汇编程序,最终在FPGA实现的处理器统上运行测试这个游戏。整个Stanford University采用由浅入深的方式,从基本的数字电路设计开始,逐渐构建整个处理器系统。而在构建过程中,为简化学生的入门难度,开始的实验还提供了已经实现的模块,只要求学生理解和应用,然后再逐渐加入复杂模块的设计。这样的方式也是值得我们借鉴的。”

我引用了这么一大段话，是因为我觉得这才是真正正确的学习顺序。因为不正确的学习顺序，我走了很大的弯路。我在做Cache的时候感到绝望，给我的感觉就是，目前大家有个任务要造一个小楼，给我分配的任务是做高处的那一层楼，第一第二第三等层楼都没有做或正在做，我却要把最高的那个楼层和大家同时搭建好。

可能并不是最高的那个楼层比其他楼层难做，而是没有其他楼层作为基础，这个工作是根本无法进行的。这里有很强的依赖关系。可以看到，在论文中,Stanford University是先让学生完成Instruction、Fetch、Decode、ALU、Register、File、VGA这些模块，最后才是加入Cache。而我则是反过来，一开始我没有系统地学习整个系统并且在整个系统没有雏形的时候，上来直接写Cache,而这个Cache或许本身的绝对难度并不大，但是离开系统谈局部是很难做的。我也几乎没有任何可以参考的东西，每一本介绍Verilog语言的书都会介绍简单运算器还有Memory模块的制作方法，有的书甚至会给出32位ALU的详细代码，但是找不到这么一本书会讲Cache如何制作。

虽然优秀的袁清亮学长作为他的SRT项目写了处理器的代码可供我们参考，但仅就我Cache部分的工作而言，这个对我没有帮助，因为它根本就没有Cache模块。《自己动手写CPU》这本书非常详细，五六百页的篇幅，最后甚至在移植操作系统，但是它也同样没有Cache模块，等于说就算我把《自己动手写CPU》这本书从头到尾看过了，也依旧不知道如何去做Cache。胡皓翔同学很厉害，一个人的工作远远超过我们四个人合作成果的总和，我很敬佩他。他最终完成了32位超标量流水线CPU的制作，我也在第一次机组中期检查的现场认真看了他的报告。我没有他的源码，不知道他最终有没有Cache，但从当时的报告来看，他的32位超标量流水CPU也是没有Cache这个模块的。

当百度上找不到方案，google里没有参考，能找到的所有相关书籍都没有这块内容时，我只能回到组成原理这门课本身了。我要自我批评。当初老师布置了用C++仿真Cache的编程作业，小组合作。当时核心工作基本是由合作的另一个同学完成的，我自己做了一些无关紧要的工作，水一水就过去了，没有真正弄懂。要是当时认真搞懂了，Cache也不会这么难做。主存与cache的地址映射方式有全相联方式、直接方式和组相联方式三种。

老师当时给出了参考的C++代码，分别完成了以上三种功能。我要采用的是最简单的直接映射，于是再去读相应代码，终于有了一点头绪。

C++代码其实和Verilog有本质的不同，读懂老师的代码，我明确了到底要做什么，但此时我依然没有见到过一个真正的用Verilog写成的Cache。这就好比我从来没有看到过真正的猪长什么样，我却要凭空造出一个猪来。而且Cache离CPU最近，又要涉及主存，耦合性比较大。当时老师的代码也是先有这么一个系统的存在，然后再去做Cache。实际情况是，当时王雨琴在做控制器模块，王昱程在做主存模块，都没有做好，而我的工作必须建立在他们已经做好或者有一个统一接口规范的前提下。我也联系她们，但大家其实也在摸索阶段，没有明确的接口规范，事实上是关于整体系统，我没有一点信息的。

这就非常麻烦了，Cache是在系统中的，耦合性很大，而其他人都在做这个系统，我不可能等到大家把这个系统做完了，再去做Cache(实际上，我觉得真的应该等CPU做完了再做Cache)。我很痛苦，我怎么才能脱离系统，凭空造出Cache,等大家做完，我这个Cache又恰好合适呢？这实际上在割裂系统整体和局部的关系，是没法继续下去的。

当初王雨琴在项目之处，就和王昱程合作，费了很大的努力做出了一个数据通路图。这个工作其实是很重要的，这相当于把整个系统都提前规划好了，决定了系统架构。接下来大家的工作理应都建立在它之上。我在学习组成原理这门课的时候，没有弄懂时序和总线，考试时这种看数据通路图的题目也不会。我尝试理解这张图，但还是看不懂。它是有总线BUS1、BUS2、BUS3，但是缺少接口规范和相应的设计文档。说实话，王雨琴王昱程她们能弄出这张图，工作量已经很大了，理应我不应再有过多要求。然而，要是真正想要多人合作做好这个项目，这张图就必须很详细，线的位数也应当统一好。

我们当时的目标是做个基于MIPS指令集的带流水线的32位CPU。王雨琴她们设计的数据通路图很好，但我觉得他们应该是按用Proteus连线时的思路画的这张图，当我现在再来看这张图的时候，会发现一个严重的问题，它不是基于流水线的。MIPS经典五级流水有取指（Fetch）、移码（Decode）、执行（Execute)、存储（Memory） 、写回到寄存器（Writeback）这五级。可以看到这个数据通路图并未按照这个来组织，我觉得它是一个非流水的CPU。按照一个非流水的且定义模糊的数据通路图，是不可能写出一个流水CPU的。

我的理解或许有偏差。但是有一点我觉得是正确的，前期我们必须花足够多的时间去做设计，不要急着去写代码，要把设计图再画得详细一点，要写好设计文档。多人合作做一个项目和一个人编程还是不一样的。软件工程告诉我，“一个项目编写代码的时间越早，其完成日期可能就越晚”，“当一个项目已经延期的时候，增加更多人手不能使项目完成得更快”。

其实四个人的项目我之前从未遇到过，人多了，沟通交流的成本就越大。暑假我没有其他什么事情，一直在家，但是其他人可能没有这么多时间。我的工作虽然绝对难度不大，但是耦合性比较强，我不得不询问其他人模块的进度，不然我的工作没法做。实际情况是，由于大家不在学校，我比较难联系到大家。其实我们也在摸索阶段，不是能很快做出原型的。

王雨琴的控制器是整个小组的重中之重，实际上整个系统都是她设计的，我们不过是在做一些辅助性的工作。陈尧的运算器基本独立，耦合性较低，不太需要联系其他人。王昱程的存储器跟我的工作联系紧密，实际上，如果没有存储器读写控制模块是没有办法做Cache并测试的。我当时由于Vivado耽搁了很多时间，在没有使用轻量的iverilog+gtkwave之前我是没有条件舒服地编程的。王昱程就没有这么幸运了，以她的电脑配置，也是带不动Vivado的，然而她又对Linux缺乏了解，不太会配置那个轻量环境，所以一直没有能够有环境写代码，这也不能怪她。但是这给我带来麻烦，没有主存及主存控制逻辑，我怎么写Cache呢？

等了很多天，Memory模块还是没有。我很焦急，难道要等到暑假结束？Memory模块其实很简单，与其坐等，不如自己写个，于是写了Memory模块。我之后工作都基于自己写的这个Memory模块，不然，我将无法完成Cache的编写和测试。

在项目开始之初，我们就有了一个开发板。然而在开发过程中，没有人关心这个开发板。另一个问题是，也没有确定主存要做多大以及Cache要做多大。大家各自去做自己的模块，可能并不需要用到这些东西。但是我就处在一个非常尴尬的境地。Cache离CPU最近，又和Memory密切相关，耦合性很大。我必须关心王雨琴控制器模块和王昱程的Memory模块。关于Memory模块，我通过自己编写它来解决了这个问题；不过控制器难度是整个小组最大的，我总不能自己再去写个控制器，确定好控制器给我什么信号再去做。

整体与局部，系统与模块，这些都有千丝万缕的联系，割裂这些联系，哪怕去完成一些简单的工作都将是寸步难行的。当时我们都没有接触过FPGA这个领域，更没有多人合作编程的经验，没有软件工程的概念，到底要做什么自己也不清楚，不能有效地组织起来。初期的任务分配是有很大问题的。

就一个计算机系统而言，最重要的是控制器，也就是王雨琴的负责部分，这是重中之重，小组包括我的其他三人都是辅助作用，三个人任务加起来的重要性也没有她大。我做的Cache其实意义不大，或者说在我们当时的情况下根本没有必要去做一个Cache。陈尧做的是ALU,从组成原理上讲ALU或许是比较核心的部分了，但是我要说，其实也根本没有必要在早期这个系统还没有搭起来的时候让陈尧去做这个ALU，在Verilog语言中，这是一个加号，一个减号等就能搞定的事情。

在实际操作中，我觉得只有这样才能快搭起一个系统的架子，之后可以去考虑加Cache，搞先行进位的运算器等等。我们应当采取增量模型或者螺旋模型的开发方法，不能一开始就考虑优化问题。老师或许对我们的期望值比较高，虽然知道不可能有什么结果，但还是希望能够尽量贴近官方的要求，把这个系统做得更快一点，所以一开始就要Cache模块，要先行进位。但是，我们的能力很有限，这些东西更像是锦上添花，而不是雪中送炭。一个计算机系统，若没有Cache模块，依然是一个完整的计算机系统；如果没有先行进位，简单地用Verilog的加号减号，计算机系统也能够搭起来。等搭起了一个大概能用的计算机系统以后，时序就不是什么大问题了，此时倘若时间还足够，可以考虑去加Cache模块，把简单的加号减号换成先行进位的复杂的ALU。此时再去做Cache应该没有多少难度，也方便测试了。

我并没有说Cache和ALU不重要，但是，一个对整体逻辑没有影响的东西（Cache）初期可以不用加，一个逻辑上可以有等效替代的东西（ALU）初期也没有必要弄得很复杂，那样反而会给测试带来麻烦。事实上，我和陈尧以及王昱程做的工作对整个计算机系统的逻辑没有什么影响，真正在做核心功能的只有王雨琴一个人，这是很不应该的。

由于动手写代码的时间过早，之前没有详细的设计，各个人在没有全局系统意识的情况下就钻到了自己的小任务中去，这导致了写出来的代码根本没法综合起来构成一个完整的系统。等到综合大家代码的时候，我也尝试去综合，但是根本没有办法。我详细阅读了所有人的代码，虽然没有能力整合，但除了王雨琴的部分代码，其他代码我是彻底了解的。王昱程因为Vivado环境可能浪费了一些时间，等她写出主存模块，我自己也做了主存和Cache,我是基于自己的Memory模块编写的Cache,功能是重合的；关于陈尧的运算器，具体实现起来各个系统可能不一样，但这部分功能比较确定，如果看成黑盒子的话，给外界的接口是比较规范的。我在几个月之前还没有这个项目的时候在业余也搞了32位ALU,比较Low,没有用先行进位，不过从功能上是等效的，运算器这部分代码也容易理解。王雨琴的控制器比较复杂，它包括了取指译码等很多东西，实际上，我觉得它本应当拆开成许多模块的。包括我自己在内的三个人其实在整个系统中没有多少作用，核心全在控制器上。

我不知道怎么去写一个控制器，拿到王雨琴的代码，也不知道怎么去验证它的正确性。王雨琴是系统的设计者，看她的代码我没能弄懂她的想法。开学后，我问起她，”我如何去验证你的代码，也就是说你的testbench写了吗？“，她感到惊讶，”什么是testbench?“,我继续解释说就是测试程序。”还要写测试程序啊？“，”那你如何保证你代码的正确性呢？“，”Vivado没有报错啊“。这时我才意识到，这个控制器并未经过测试，正确性不能得到保证。在各个模块通过各自的单元测试之前，不能进行系统测试，整合是做不了的。

Verilog是硬件描述语言，和高级语言有本质区别，通过编译检查不能保证它描述的电路能完成设计者预期逻辑功能。符合Verilog语法并不能说是完全正确，Verilog描述的电路正确性要靠设计者自己或他人设计测试方案写测试程序来验证，通过测试才能初步说明其正确性。

小组项目失败了，我觉得我们的失败主要是我们尝试了一个新的领域，直到胡皓翔同学的工作震惊了我们。

暑假结束后，胡同学展示了他的基于MIPS指令集的32位超标量流水线CPU，客观地说，这个工作远远超过了我们四个人合作的东西。胡同学很优秀，平时就很厉害，做出来的东西远远超过我，这很容易接受；然而他一个人超过了我们四个人的总和，这就是我们自身有问题了，话说”三个臭皮匠”还“赛过诸葛亮“呢。

我们虚心听胡同学介绍成功经验，他提到一本书《自己动手写CPU》,他自己学习这本书，期间他动手实践，循序渐进，先后实现了8位的到32位的再到32位超标量流水CPU。为了一个学分的课设，居然去做了三个CPU,这真是令人敬佩。

我意识到，我们前期真是”出生牛犊不怕虎“，明明连Verilog语法都是现学的，也没有系统地看什么关于CPU设计的东西，就准备写代码了，真是”[蚍蜉撼大树,可笑不自量](http://zhidao.baidu.com/question/445534200.html" \t "/home/curie/Documents\\x/_blank)“。事实上，老师在暑假开始之前就给我们推荐《自己动手写CPU》这本书，但大家一直没有去认真看。

当时老师介绍这本书的时候我倒打算慢慢看的，然而后来自己比较急功近利，只是粗略浏览了那书配套的博文就不看了。因为我的任务是做Cache,而用Verilog实现Cache，这个内容《自己动手写CPU》一书里没有，我就没有看，从而对系统没有一个全局的认识，影响了后面的开发。这是十分短视的行为。

除了急功近利不读书，还有一个问题就是初期目标定的过高，没有考虑到自己的实际水平，胡同学的超标量流水CPU也不是一蹴而就的，而是经过多次迭代开发。我们也应当采取类似方法，将一个大目标划分为一个一个小目标，一个一个去实现。

从优秀同学身上，从老师的教导中，从团队的合作中，组成原理的课程设计都给我很多宝贵经验。前期走了很多弯路，课设的结果并不理想，但我觉得，花了真功夫，走的冤枉路都不会冤枉，组成原理的课程设计初步培养了我们的系统能力，这是最宝贵的财富。

**5参考文献**

在原理上，参考了老师写的Cache的三种地址映射方式的C++代码。其他最有帮助的是威斯康星大学麦迪逊分校（wisc）的组成原理课程网站，后来我发现老师帮我们找到的MIPS指令结构手册的来源也是这个网站。

[1]http://pages.cs.wisc.edu/~karu/courses/cs552/spring2009/wiki/

[2]http://pages.cs.wisc.edu/~karu/courses/cs552/spring2009/wiki/index.php/Main/CacheSimulator

[3]https://github.com/nblintao/Computer-Architecture

[5]自己动手写CPU 雷思磊著

[4]Verilog\_HDL :A Guide to Digital Design and Synthesis数字设计与综合（第二版）[美]Samir Palnitkar著

[5]Verilog数字系统设计教程（第二版） 夏宇闻著