**中国科学技术大学计算机学院**

**《计算机组成原理实验》报告**



实验题目：\_\_\_\_\_\_\_综合设计\_\_\_\_\_\_\_\_

学生姓名：\_\_\_\_\_\_徐奥 谭骏飞\_\_\_\_\_\_

学生学号：\_PB20061343\_PB20061276\_

完成日期：\_\_\_\_2022年5月22日\_\_\_

计算机实验教学中心制

2020年09月

【实验目标】

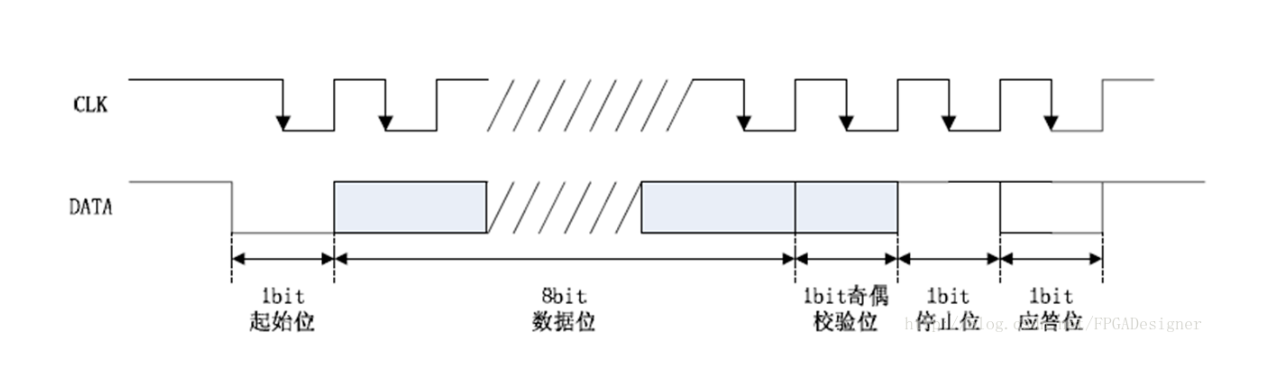
1. 理解计算机硬件系统的组成结构和工作原理
2. 掌握软硬件综合系统的设计和调试方法

【实验内容】

**内容1**.支持键盘输入调试指令

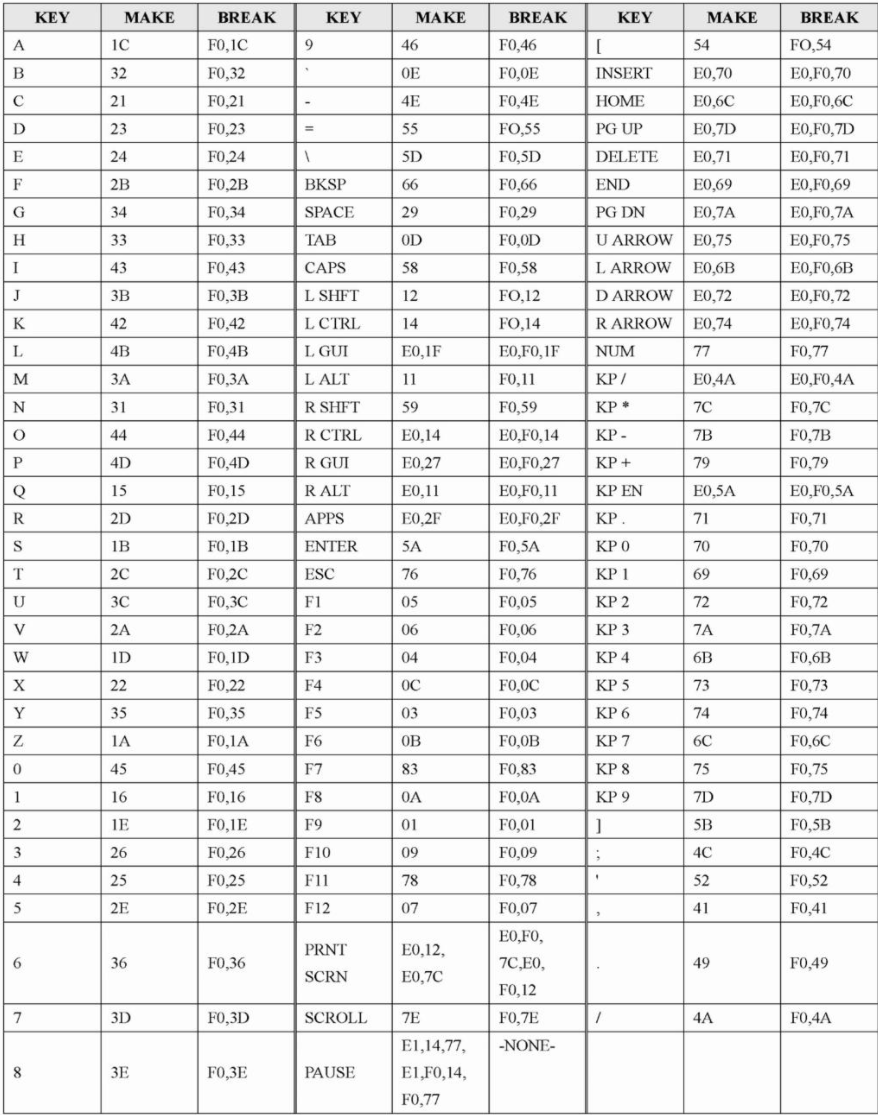
键盘使用PS2接口协议。PS2设备的clock和data都是集电极开路的，平时都是高电平。数据传送按帧格式，数据位在clock位高电平时准备好，在clock下降沿被读入。

传输1byte数据的过程为：键盘准备好数据以后，输出起始位0到数据线上，然后输出8bit数据位，1bit校验位和1bit输出停止位。如下图



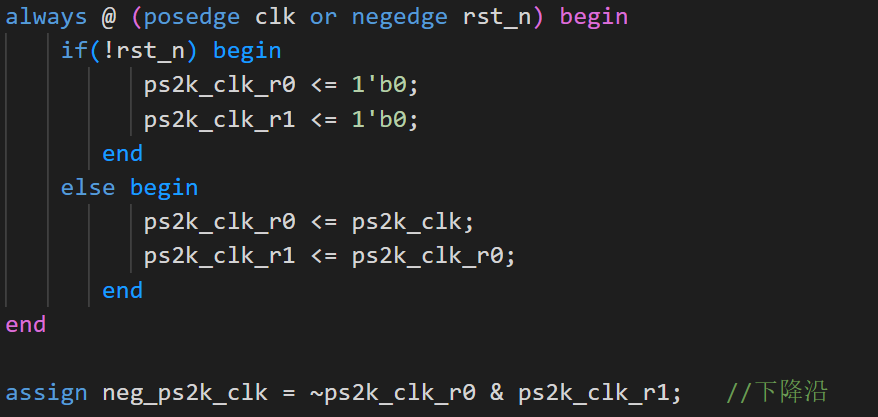
**图1**

键盘的按键被按下时，会发送一个字节，即通码。按键在释放时，会发送两个字节，即断码。每一个按键都有唯一的通码和断码，据此可以判断按下的是哪个键，从而执行对应的功能。通码断码对照表如下



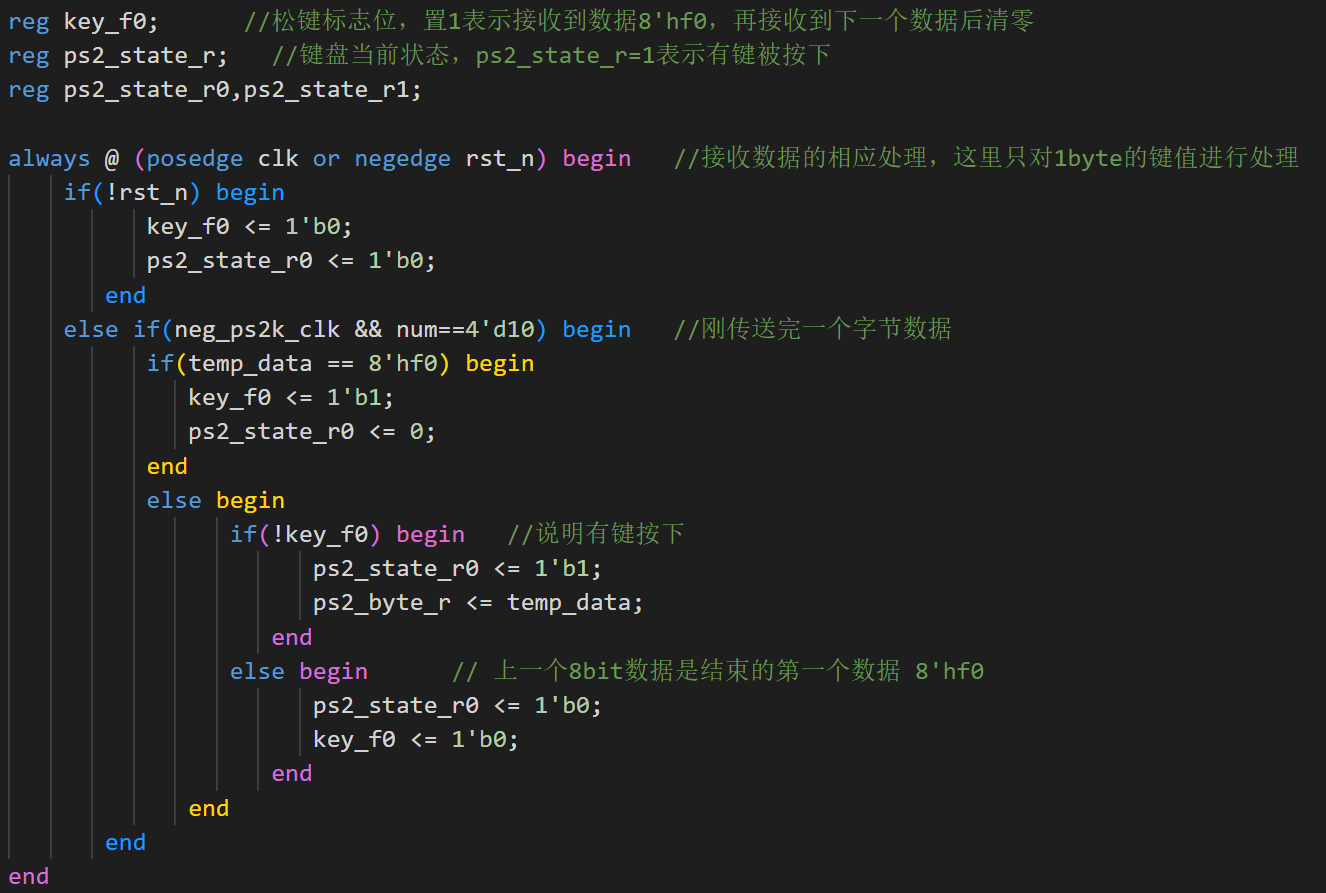
**图2**

在具体代码实现中，需要关注键盘的时钟下降沿，在其每个时钟下降沿获取1bit信息。

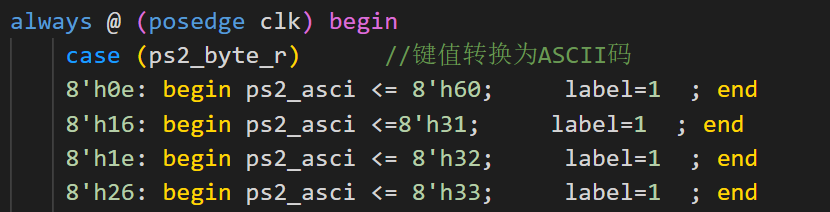


**图3**

当获得一个低电平后，就开始连续读入8bit数据位，并于通码断码对照表比对，判断其信息。可以观察到的一个特点是，2byte断码的第一个byte恒为8’hf0，可以通过这一特点判断是通码还是断码。读取通码后，将其转化为ASCII码传给PDU



**图4**



**图5**

在PDU模块中定义一个kb\_cmd数组，记录来自键盘的ASCII码字符，这里规定以行为单位划分调试指令，即当读到回车的ASCII码时，判定为一条指令输入完成，之后进行指令译码与具体操作。

与PDU原本的调制功能对应，我们设置了以下调试指令：

1. step：单步运行
2. cont + 数字：连续运行，数字即为设置的断点PC值
3. chek + 数字：查看某个具体的内容，包括各类功能寄存器、通用寄存器和内存
4. chek：查看下一个内容，即当上一条指令的数组部分不存在时，默认查看下一项内容
5. data + num：作为外设，向CPU输入内容（相当于原来的开关输入）

以上指令均与原PDU的调试button兼容，也即两类方法均可用。例如说，首先用button查看了通用寄存器x2的内容，然后键盘输入chek，即可查看寄存器x3的内容，再次按下chek的button，可继续查看寄存器x4的内容。以上查看的结果，仍使用FPGA的数码管进行显示。

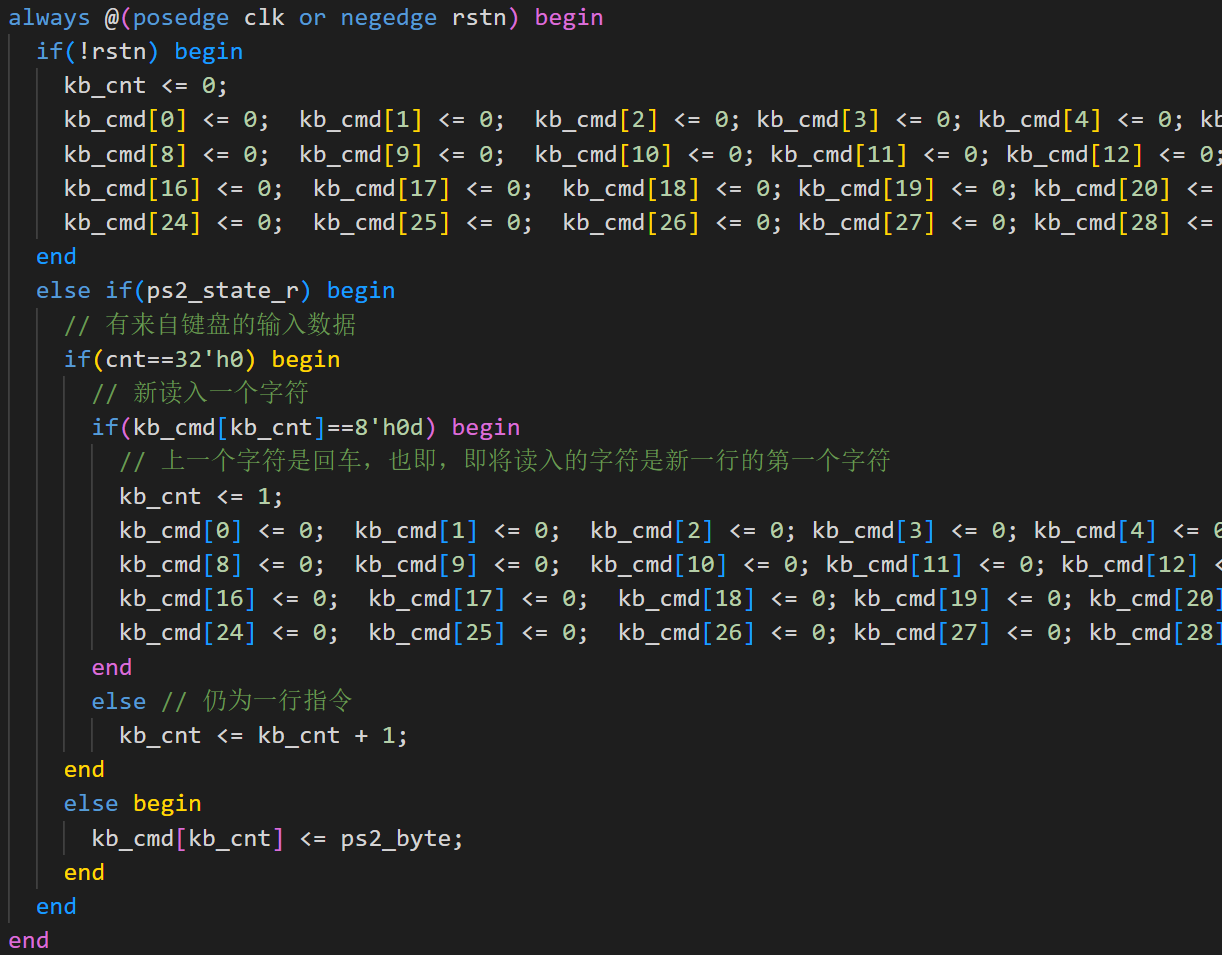
在这一部分功能实现的过程中，遇到的最大困难就是键盘和FPGA的时钟不同步问题。传输过程中的不同步问题还可以通过取键盘的时钟下降沿完成，即只有遇到键盘的时钟下降沿，才读入1bit的数据。

但是传输完成后，获得的1byte数据会一直存在，持续很多个FPGA的时钟周期，直到下一次键盘产生输入。而我们预期只需要取一次进行保存。所以这里的处理不能简单的每个时钟cnt加一，而应传送一个键盘被按下的信号，这个信号也会持续数个时钟周期，但只需要取第一个有效周期即可。

另一个需要解决的问题是指令字符数组的清零问题，理论上是每次输入回车后清零。具体实现上，由于我们需要由指令产生的调试信号（如chek、step）等可以持续几个时钟周期，而如果一旦读入回车立刻将指令字符数组清零，那调试信号最多持续一个时钟周期，这样的调试信号实际运行的效果并不好，经常不会被执行。所以，我们将清零操作放在输入回车后的下一次1byte数据的输入时，也即先清空指令字符数组，再读入。读入比原来延迟了一个时钟周期，不过并不会产生影响。

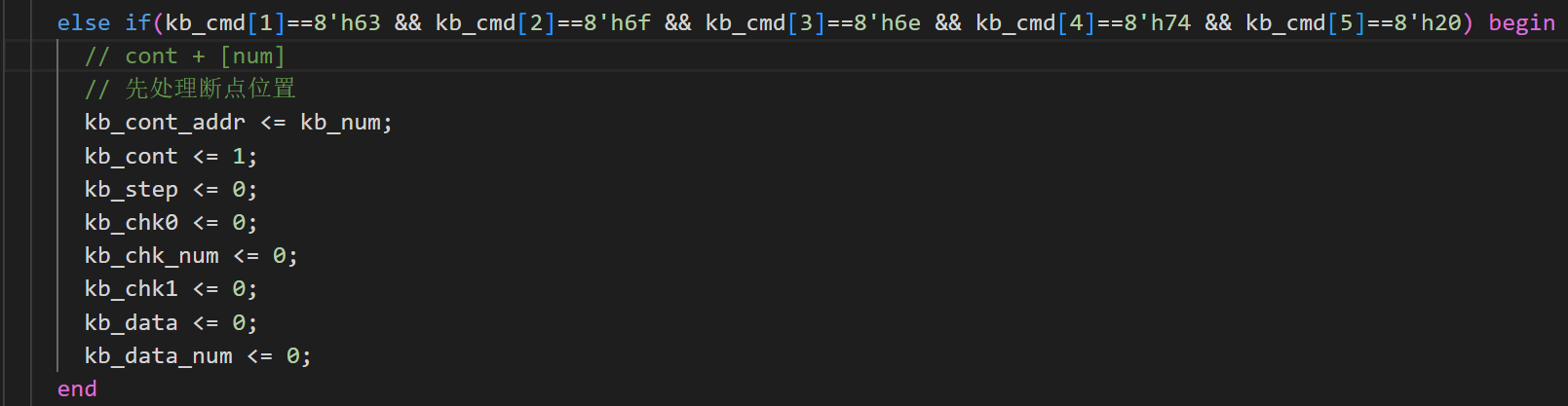
以下以cont+num为例，简述键盘输入指令的具体执行过程。

键盘首先向FPGA传送多位数据，包括起始位、8bit数据位、校验位和终止位，经处理后我们获得1byte有效数字，对应一个ASCII码。此时键盘有按键按下的标记信号置为有效，先后执行cnt+1，和1byte数据存入指令字符数组。代码如下：



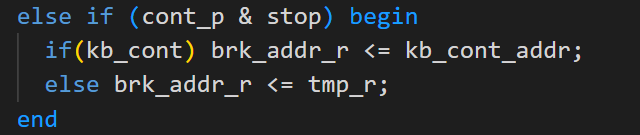
**图6**

若键盘传来的信号为断码，那么置ps2\_state\_r为低电平。一个字符的读入完成。当读到回车时，会解码执行，在本例中，那就应该是cont+num：

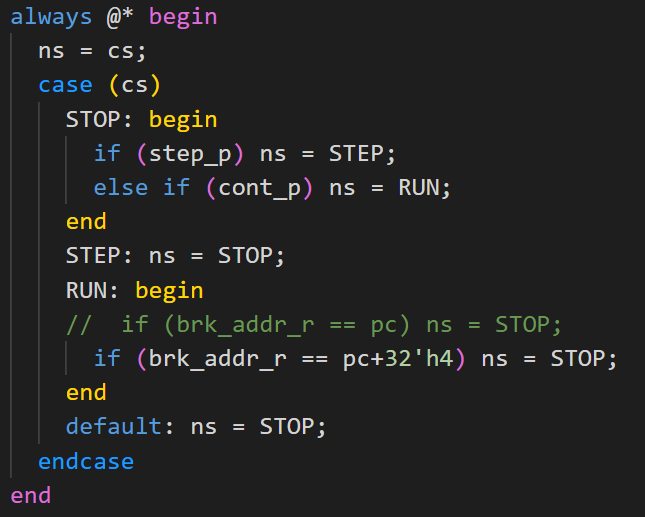


**图7**

置相应的信号为有效，注意，其他信号需要置为无效。之后，就是写入相应的变量中，具体去运行。



**图8**

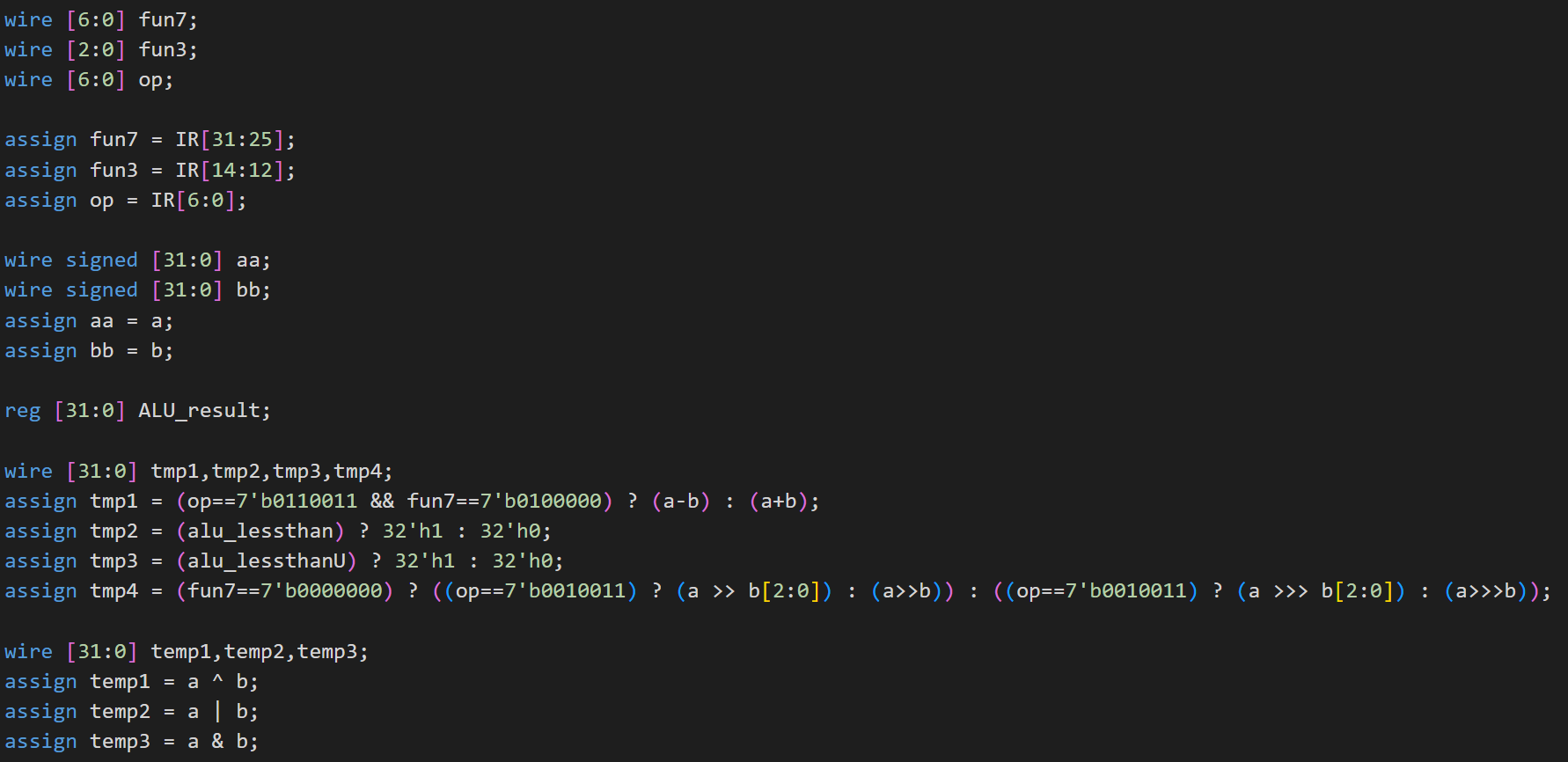


**图9**

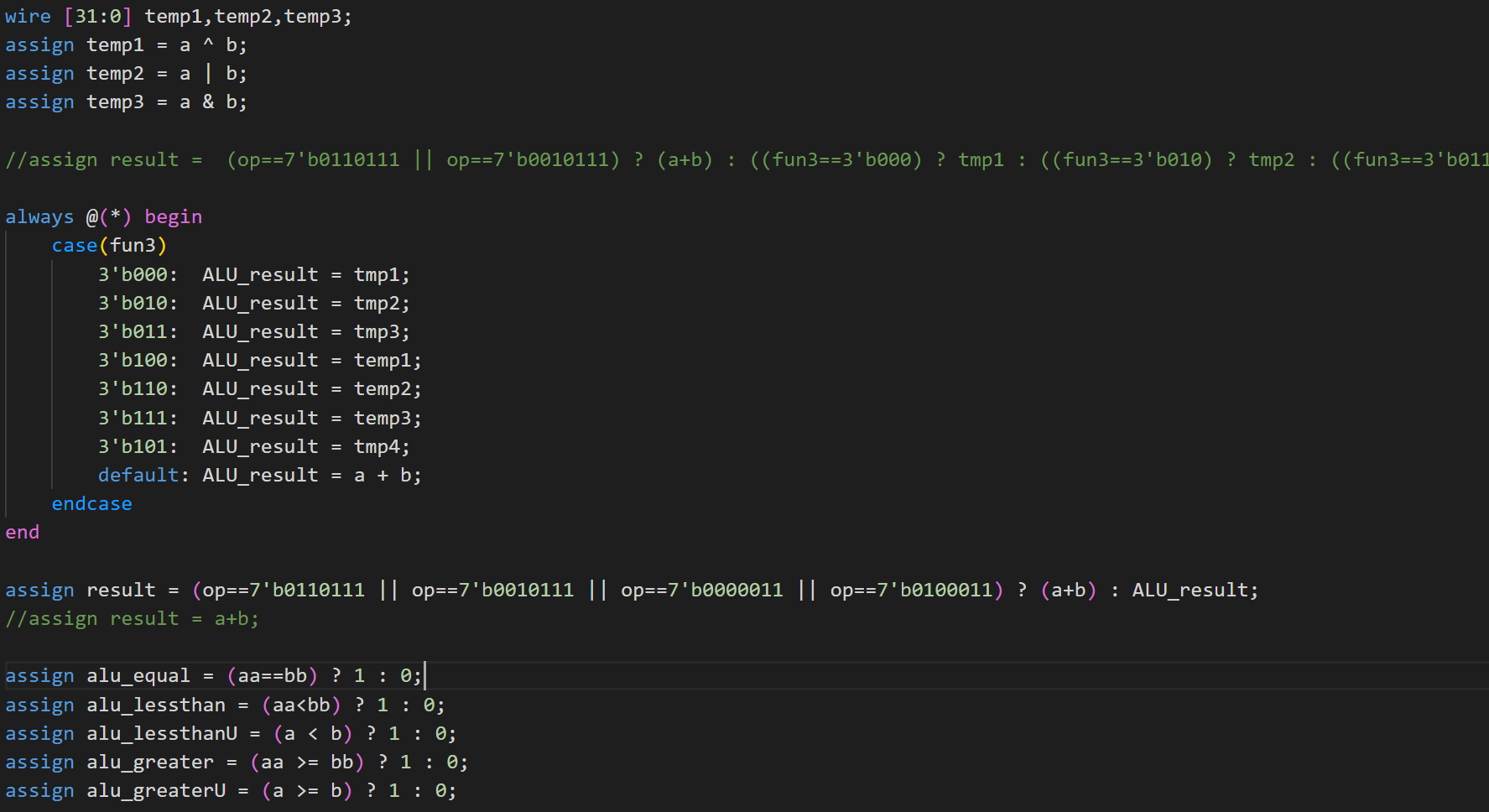
**内容2**.扩展指令至37条

原本仅支持10条指令，我们将PPT上剩下的27条指令都加入了CPU。包括6条逻辑运算指令、4条条件分支指令、6条load&store指令、6条移位运算指令、4条比较置位指令以及一个大数运算指令lui.

具体的实现，主要涉及到了ALU部分的扩充，load和store的扩充、以及条件分支情况的扩充。其中，扩展后的ALU核心代码如下



**图10**

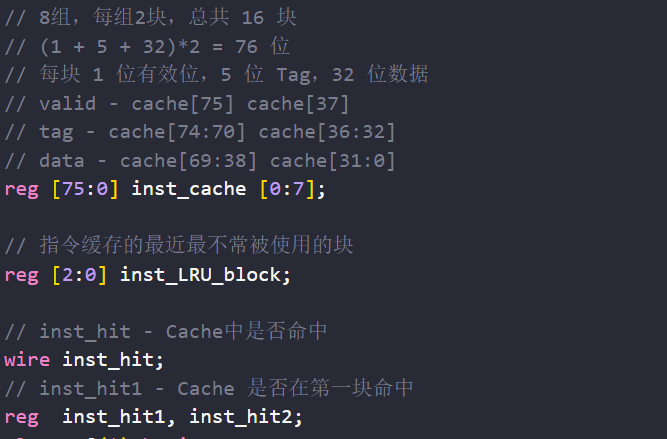


**图11**

这里我们遇到了一个BUG，它困扰了我们近五天。这个问题的具体体现是，扩展ALU后，原本的排序就无法正确排序，甚至不会发生任何数据交换。我们一直怀疑是always\* 的问题，所以甚至一度将上述代码全改为了assign，但问题仍然存在。最后通过单步调试，问题出在了lw指令，具体而言，load和store指令都会使用ALU，而它们只会使用加法，但是如果不在ALU中加特判的话，load和store自身的fun3就会把它们调用的ALU执行为其他算术逻辑指令。所以ALU还需加上load和store的特判。

**内容3**.Cache

从 Lab3 冒泡排序的汇编程序来看，程序会频繁地访问某一块区域的指令和数据，因此在 CPU里加入 Cache 可以利用汇编指令的空间局部性提升 CPU 的运行效率。本次实验中采用组相联 Cache，分为 8 组，每组 2 块，采用 LRU 的块替换策略。每个 Cache 块由 1 位有效位，5 位标签位和 32 位数据位组成。附上变量的定义。

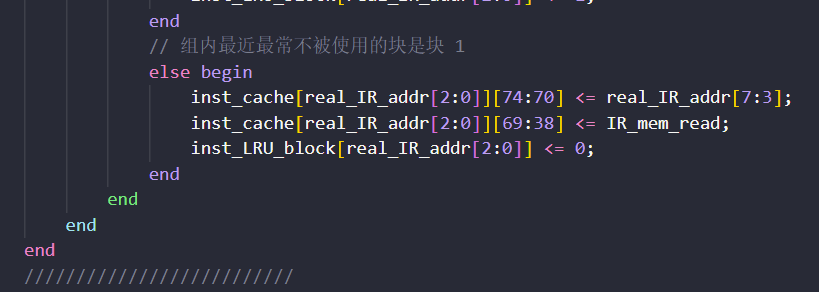


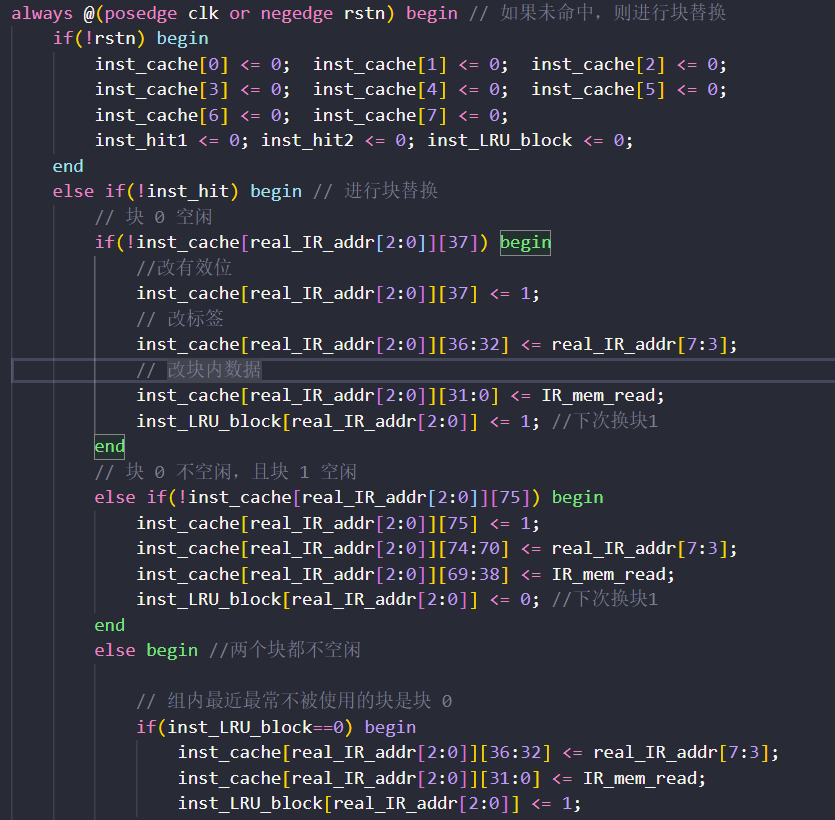
**图12**

以数据 Cache 为例说明其具体运行流程，CPU 在取数时会先去数据 Cache 中查找，如果在 Cache 中找到所需的数据，则将其送到相应的段间寄存器；如果没有找到，则待从内存中读出数据后，在接下来的时钟上升沿将读出的数据存入 Cache，如果 Cache 没有空余的块则进行块替换。当 CPU将数据写入内存时，采用写穿透策略，命中则在写入内存的同时写入相应的块，反之则在块替换以后将相关信息存入 Cache。

Cache 的块替换采用 LRU （最近最长未被使用）策略，用专门的变量记录下一次需要替换的 Cache 块，默认为 0。运行时若某个块被访问或载入数据，则将另一个块的编号赋给该变量，每次块替换时根据变量的值确定要替换哪个块。

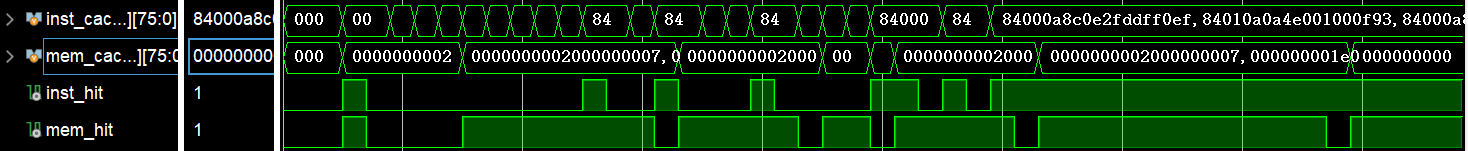
附上数据 Cache 的核心代码。

****

**图13**

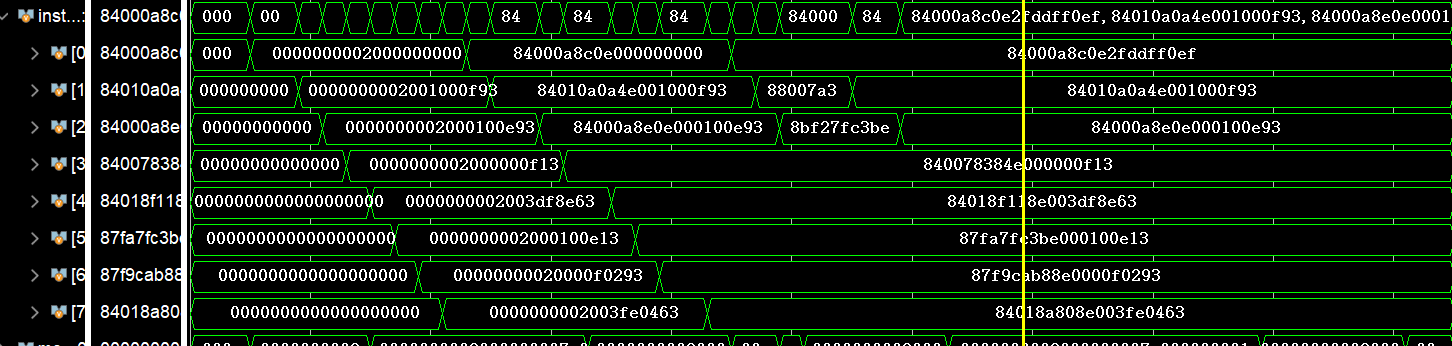
对冒泡排序的仿真结果如下，由此可见在程序起来后，指令和数据 Cache 的命中率都能达到较高的水平，从而减小访问内存的开销。

注：图中 inst\_hit 和 cache\_hit 分别表示指令 Cache 和数据 Cache 是否命中。

****

**图15**

仿真过程中指令 Cache 的内容如下。

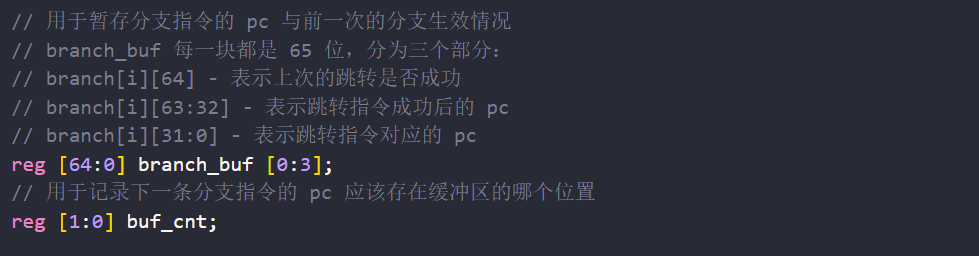
****

**图16**

**内容4**.动态分支预测

多数汇编程序中还会带有大量分支语句，采用从失败处调度时，会因为分支生效产生的停顿而带来性能损失，因此提升分支预测精度也可以对 CPU 的性能带来提升。

实验中用 1位转移预测缓冲器来实现动态分支预测。缓冲器的每一块分为 1 位的预测位（用于存上次该分支指令的跳转结果），32 位用于存分支指令的目标 PC，另外32 位用于存分支指令的 PC。附上变量的定义。

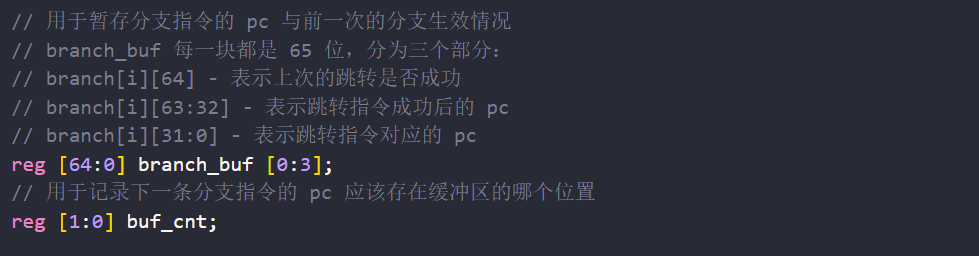


**图17**

其运行逻辑如下：当 IF段取到一条新的转移指令时，查看缓冲器是否有这条指令，如果指令已在缓冲器中，根据以前状态预测这条指令。当 EX 段获得一条转移指令时，也会去缓冲器里查找，如果找到且上一次的结果和此次的跳转结果不相同，则清空 IF/ID 段 和 ID/EX 段的段间寄存器，相同则继续运行；如果没有找到，则将该指令及其此次运行结果加入缓冲器（如果缓冲器满，则根据先进先出原则进行替换），并根据此次分支结果判断是否需要清空相应的段间寄存器。

附上缓冲器确定下一取指地址的代码。

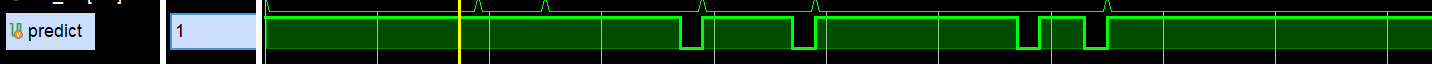




**图19**

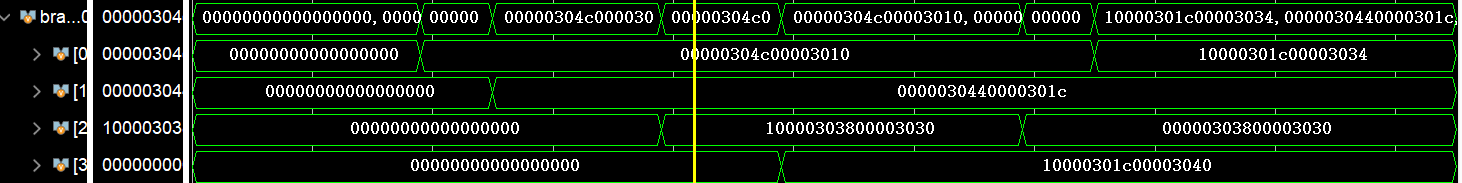
对冒泡排序的仿真结果如下，由此可见预测的失败率相对较低，减少了 CPU 因控制冒险产生的停顿。

注： predict 变量当且仅当需要因控制冒险产生停顿时才为 0。



**图20**

仿真过程中缓冲区的具体内容如下。



**图21**

【总结与思考】

1. 本次实验难度中等偏难，不过我们能够完善自己的CPU和PDU，收获还是很大，完善后的PDU支持键盘输入调试，完善后的CPU扩展了指令、增加了cache和动态分支预测。
2. Verilog属于硬件代码，每个变量、赋值和运算操作直接对应着实际的电路，所以通过本次实验以及之前的5次实验，我们能够从电路的层次深入体会CPU的执行逻辑，通过自己搭建CPU的方式，我们对CPU的理解得以加深。通过对CPU增加优化与动态分支预测，我们加深了对 Cache 和动态分支预测实现过程的理解。
3. 最后一次实验也是两人合作，与人合作是一项不可缺少的技能，在这个过程中，我们的沟通交流与任务分工的能力得到了锻炼，一个良好的合作应该是各取所长、协调配合，我们也十分感谢彼此的配合与协作。

【附录】

COD6.srcs 包含了设计文件、引脚约束文件

sort.asm 排序的汇编代码

test.asm 测试扩展的指令的汇编代码