**实验2报告**

第32小组

袁峥、施璠

一、实验任务

此次实验需要对上学期计算机组成原理实验课中完成的多周期CPU进行改进，总体任务是将原来的一个异步数据指令存储器进行拆分，拆分成两个同步存储器，分别存放数据和指令，并更改相应的控制信号使CPU正常运行。

具体实实验任务如下：

1. 迁移组成原理研讨课上完成的多周期 CPU 设计到本学期实验平台上。
2. 要求多周期 CPU 支持以下 15 条机器指令： LUI、 ADDU、 ADDIU、 BEQ、 BNE、 LW、 OR、 SLT、 SLTI、SLTIU、 SLL、 SW、 J、 JAL、 JR。
3. 要求多周期 CPU 对外访存接口为取指、 数据访问分开的两个 SRAM 接口， 且 SRAM 接口是同步读、同步写的。
4. 要求多周期 CPU 顶层连出部分 debug 信号， 以供验证平台使用。
5. 要求多周期 CPU 复位从虚拟地址 0xbfc00000 处取指。
6. 要求多周期 CPU 虚实地址转换采用： 虚即是实， 也就是转换过程不需要对虚拟地址作变换。
7. 要求多周期 CPU将每条指令对应的 PC每个周期都带下去， 一直带到写回级（ 最后一级） ， 这个信息，暂时 mycpu 里不会用到，但验证平台会使用到。
8. 要求实现 MIPS 架构的延迟槽技术，可以认为软件在延迟槽放置的永远为 NOP 指令。
9. 要求多周期 CPU 只实现一个操作模式：核心模式， 不要求实现其他操作模式。

10、不要求支持例外和中断。

11、不要求实现任何系统控制寄存器。

12、要求将多周期 CPU 嵌入到提供的一个简化系统 SoC\_lite 上。 完成仿真和上板功能验证。

二、实验设计

（一）设计方案

1、总体设计思路

相对于上学期完成的多周期CPU设计，此次实验需要对其中一些细节进行修改。

首先由于此次实验将原先的一个数据指令存储器拆分成了两个，因此在存储器控制信号方面需要将两份存储器的控制信号分开，于是便要增加一些控制信号。

由于存储器从异步变成了同步，所以在存储器中读取数据时时序发生了变化，原先用于寄存数据的MemDataReg不再需要，因为原来的读取数据操作可以理解为瞬间完成，因此需要放入寄存器使其保持到下一个周期，但改成同步存储器之后，读取的数据本身就已经在时钟上升沿才被读出，所以可以将其去除。

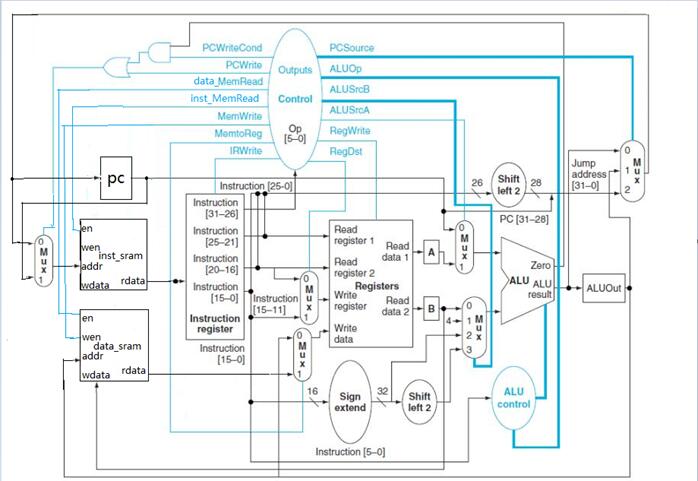
在指令存储器的读地址信号上也要做一些修改，原先使将PC寄存器中的信号连到该接口，但是改成同步指令存储器后，指令存储器的读地址接口必须提前准备好，因此将此前的新地址多路选择器信号直接接到读地址口，同时将原PC寄存器的写控制信号也作为读地址口的控制信号，如果要改写PC，则同时改写读地址，否则读地址保存为PC的值，保证两者一致，并且能够提前完成读地址准备。

此外，此次实验还修改了复位时的状态机，一方面是将复位时的PC信号修改为了任务书中要求的0xbfc00000，另一方面增加了一个新的状态，专门用来完成复位，此状态不做任何操作，仅为了保留一拍PC的值，因为如果直接将复位时的状态更改为所有指令的第一个状态（即取指和PC+4），则由于PC没有提前准备好而导致读取的指令有误，所以特地新增一个状态来保持PC值，之后再跳转到取指和PC+4状态，保证可以顺利读出指令。

在debug信号的准备上，由于是在寄存器写信号为1时与标准输出进行比较，因此debug中的PC信号要保留为原指令的地址，其余信号分别接相应的寄存器堆的控制信号即可。

整个实验一共分为四个模块，第一个为CPU的主模块，进行整体调度，第二个为ALU模块，进行一些运算操作，第三个为寄存器堆模块，用来完成寄存器的读写，第四个为控制信号模块（Control Logic Unit），用来生成CPU中的各个控制信号并控制状态机的转换。

整体设计图如下：



2、模块1设计

模块1为CPU的总体调度模块，由它来控制模块2（ALU模块）、模块3（寄存器堆模块）和模块4(控制信号模块)的任务。

（1）工作原理

将整个设计拆分为不同模块，不同模块完成不同任务，其中模块1完成CPU功能的整体调度，主要为控制信号的通路设计，而其余的一些子任务（如控制信号的产生、ALU、寄存器堆）交给其他模块，这样可以使整个任务结构清晰，并且便于调试。

（2）接口定义

| **名称** | **方向** | **位宽** | **功能描述** |
| --- | --- | --- | --- |
| clk | IN | 1 | 时钟信号，来自 clk\_pll 的输出时钟 |
| resetn | IN | 1 | 复位信号， 低电平同步复位 |
| inst\_sram\_en | OUT | 1 | ram 使能信号，高电平有效 |
| inst\_sram\_wen | OUT | 4 | ram 字节写使能信号，高电平有效 |
| inst\_sram\_addr | OUT | 32 | ram 读写地址，字节寻址 |
| inst\_sram\_wdata | OUT | 32 | ram 写数据 |
| inst\_sram\_rdata | IN | 32 | ram 读数据 |
| data\_sram\_en | OUT | 1 | ram 使能信号，高电平有效 |
| data\_sram\_wen | OUT | 4 | ram 字节写使能信号，高电平有效 |
| data\_sram\_addr | OUT | 32 | ram 读写地址，字节寻址 |
| data\_sram\_wdata | OUT | 32 | ram 写数据 |
| data\_sram\_rdata | IN | 32 | ram 读数据 |
| debug\_wb\_pc | OUT | 32 | 写回级（ 多周期最后一级） 的 PC， 因而需要 mycpu 里将 PC 一  路带到写回级 |
| debug\_wb\_rf\_wen | OUT | 4 | 写回级写寄存器堆(regfiles)的写使能，为字节写使能，如果  mycpu写 regfiles为单字节写使能，则将写使能扩展成 4位即可。 |
| debug\_wb\_rf\_wnum | OUT | 5 | 写回级写 regfiles 的目的寄存器号 |
| debug\_wb\_rf\_wdata | OUT | 32 | 写回级写 regfiles 的写数据 |

（3）功能描述

由clk信号来控制时钟，resetn信号控制复位。

在该模块内部，由PC寄存器来存储PC值，同时由多路选择器从模块末端（即上一条指令的最后一个周期）给出的PCnext信号及PC寄存器来选择指令存储器的读地址接口，再将读出的指令存储到InstructionReg，此处由IRWrite控制是否要更新该寄存器，接着将该寄存器中的指令进行解码，并根据不同的指令情况将解码后的不同部分送往不同地方，其中解码后的寄存器地址送到寄存器堆的读地址，根据不同的指令情况，寄存器堆的写地址和写数据不同，具体可见上图结构图。接着根据不同指令完成相应操作，操作由控制信号模块所给出的控制信号和状态机决定。在整个模块中，会由ALU模块进行数据计算，由寄存器堆模块完成读写寄存器堆，由控制信号模块控制不同寄存器的读写和信号通路。

整个指令完成共分为5个部分，分别为取指并更新PC、解码与读取寄存器堆、执行、访存和写回，每条指令的通路不同，具体由控制信号逻辑给出的控制信号和状态机确定。

debug信号从该模块内部选择信号接出，debug\_wb\_pc保存该条指令的PC地址，debug\_wb\_rf\_wen、debug\_wb\_rf\_wnum和debug\_wb\_rf\_wdata分为接到寄存器堆的写使能、写寄存器地址和写数据，用于调试和验证。

3、模块2设计

模块2为ALU模块

（1）工作原理

ALU模块输入为数据A、B和操作信号ALUop，根据不同的ALUop完成不同的算逻运行，并输出结果及溢出、进位和判零信号，其中加法采用超前进位加法，而减法采用取反加一将其变为加法。

（2）接口定义

| **名称** | **方向** | **位宽** | **功能描述** |
| --- | --- | --- | --- |
| A | IN | 32 | 数据信号，为第一个操作数 |
| B | IN | 32 | 数据信号，为第二个操作数 |
| ALUop | IN | 3 | 控制信号，表示将要进行的算逻操作 |
| Overflow | OUT | 1 | 标志位，为1表示算术运算溢出 |
| CarryOut | OUT | 1 | 标志位，为1表示算术运算产生进位 |
| Zero | OUT | 1 | 标志位，为1表示算逻运算结果为零 |
| Result | OUT | 32 | 数据信号，为算逻运算的结果 |

（3）功能描述

根据不同的操作信号ALUop，在模块内部进行不同的算逻操作，其中还有一个加法器模块，采用了超前进位加法，可以提高运算效率，在进行减法时，将第二个操作数进行取反加一操作，从而将减法转变为加法。整个模块一共可以完成与（AND）、或(OR)、加(ADD)、减（SUB）、尾部添零（LUI）、左移(SLL)、无符号小于比较（SLTIU）和有符号小于比较（SLT）。

4、模块3设计

模块3为寄存器堆模块

（1）工作原理

从控制信号控制寄存器堆的读写，其实时间保存住寄存器堆中的数据。复位信号来控制寄存器堆的清零。

（2）接口定义

| **名称** | **方向** | **位宽** | **功能描述** |
| --- | --- | --- | --- |
| clk | IN | 1 | 时钟信号 |
| resetn | IN | 1 | 复位信号，低电平有效，进行寄存器堆清零 |
| waddr | IN | 5 | 写地址信号 |
| raddr1 | IN | 5 | 读地址信号1 |
| raddr2 | IN | 5 | 读地址信号2 |
| wen | IN | 1 | 写使能信号，高电平有效 |
| wdata | IN | 32 | 写数据信号 |
| rdata1 | OUT | 32 | 读数据信号1 |
| rdata2 | OUT | 32 | 读数据信号2 |

（3）功能描述

该寄存器堆为异步寄存器，根据读写地址和写使能信号进行寄存器对的读数据可以瞬间完成，写数据需要在时钟上升沿完成。0号寄存器一直保持为数据零，不接受写数据。

5、模块4设计

模块4为控制信号

（1）工作原理

根据输入的指令信号及当前状态，更新状态机状态，转移至下一状态，并给出相信的控制信号。通过不断的状态转移及相应的控制信号来完成相应的指令操作。

（2）接口定义

| **名称** | **方向** | **位宽** | **功能描述** |
| --- | --- | --- | --- |
| Operator | IN | 6 | 指令信号，为指令的前六位，用来区分不同指令 |
| SpecialBits | IN | 2 | 指令信号，为指令的第4和6位，用来进一步区分SLT（11） ADD/OR（10） SLL（00） JR（01）指令 |
| State | IN | 4 | 当前状态机状态 |
| PCWrite | OUT | 1 | PC寄存器写使能信号1，高电平有效，用来控制无条件跳转 |
| PCWriteCond1 | OUT | 1 | PC寄存器写使能信号2，高电平有效，用来控制BEQ条件跳转 |
| PCWriteCond2 | OUT | 1 | PC寄存器写使能信号3，高电平有效，用来控制BNE条件跳转 |
| InstMemRead | OUT | 1 | 指令存储器读使能信号，高电平有效 |
| DataMemRead | OUT | 1 | 数据存储器读使能信号，高电平有效 |
| MemWrite | OUT | 4 | 数据存储器写使能信号，高电平有效 |
| IRWrite | OUT | 1 | 指令寄存器写使能信号，高电平有效 |
| MemtoReg | OUT | 2 | 寄存器堆写数据选择信号，用来选择寄存器堆写数据来源 |
| PCSource | OUT | 2 | PC更新选择信号，用来选择更新的PC值 |
| ALUOp | OUT | 2 | ALU算逻运算选择信号，用来选择ALU进行的操作 |
| ALUSrcB | OUT | 2 | ALU输入数据B的选择信号，用来控制数据B的来源 |
| ALUSrcA | OUT | 2 | ALU输入数据A的选择信号，用来控制数据A的来源 |
| RegWrite | OUT | 1 | 寄存器堆写使能信号，高电平有效 |
| RegDst | OUT | 2 | 寄存器堆写地址选择信号，用来选择写地址来源 |
| NewState | OUT | 4 | 下一状态机状态，根据指令信号和当前状态进行更新 |

（3）功能描述

根据当前状态及当前指令来选择下一状态，并更新控制信号的值，整个模块整体上由一个case语句构成，根据状态机来确定当前状态的下一状态。

（二）验证方案

1、总体验证思路

实验验证分为两部分，分别为仿真验证和上板验证。

仿真验证根据自己的CPU的debug信号与标准结果是否一致，如果一致则会输出“PASS!!!”，否则会输出错误错误的信号进行调试。

上板验证将生成的bit文件编程到FPGA开发板上并运行，观察开发板上的LED灯与数码管的显示，如果与标准结果一致则验证成功，否则验证失败。

(1) 在 Linux 下使用交叉编译工具编译测试 func，得到编译结果。

(2) Vivado 仿真运行 cpu132\_gettrace， inst\_ram 加载 func 用于仿真的编译结果， 生成参考结果 trace\_ref.txt。

(3) Vivado 仿真运行 mycpu， inst\_ram 加载 func 用于仿真的编译结果， 参考仿真结果是 error 还是 pass。

(4) 第(3)步中，如果 error，则进行仿真 debug， 重复第(3)、 (4)步。

(5) 第(3)步中， 如果 pass，则 Vivado综合实现 mycpu， 此时 inst\_ram需要加载 func用于综合实现的编译结果，进行上板验证，观察实验箱上数码管显示结果， 判断是否正确。

(6) 第(5)步中， 如果判断上板正确，则 myCPU 验证成功，结束。

2、验证环境

仿真验证：

(1) 在 Linux 下使用交叉编译工具编译测试 func，得到编译结果。

(2) Vivado 仿真运行 cpu132\_gettrace， inst\_ram 加载 func 用于仿真的编译结果， 生成参考结果 trace\_ref.txt。

(3) Vivado 仿真运行 mycpu， inst\_ram 加载 func 用于仿真的编译结果， 参考仿真结果是 error 还是 pass。

(4) 第(3)步中，如果 error，则进行仿真 debug， 重复第(3)、 (4)步。

上板验证：

(5) 第(3)步中， 如果 pass，则 Vivado综合实现 mycpu， 此时 inst\_ram需要加载 func用于综合实现的编译结果，进行上板验证，观察实验箱上数码管显示结果， 判断是否正确。

(6) 第(5)步中， 如果判断上板正确，则 myCPU 验证成功，结束。

3、验证计划

| **编号** | **功能点描述** | **考核标准** |
| --- | --- | --- |
| 1 | 在仿真环境中进行仿真验证，通过自带的比较结果判断是否成功 | 运行结果输出：PASS!!! |
| 2 | 在FPGA开发板上进行验证，观察数码管及LED灯的变化 | 判断数码管的输出与LED灯的变化是否与标准情况一致 |

三、实验实现

（一）实现交付说明

本实验所修改的代码在 lab2\_32\mycpu\_verify\rtl\myCPU目录下：

alu.v ALU模块

reg\_file.v 寄存器堆 模块

mycpu.v 多周期CPU

（二）实现说明

mycpu调用的模块包括ALU，寄存器堆，控制逻辑单元，其中控制逻辑在mycpu.v中。

其他两个模块分别在alu.v 和 reg\_file.v中。

四、实验测试

（一）测试过程

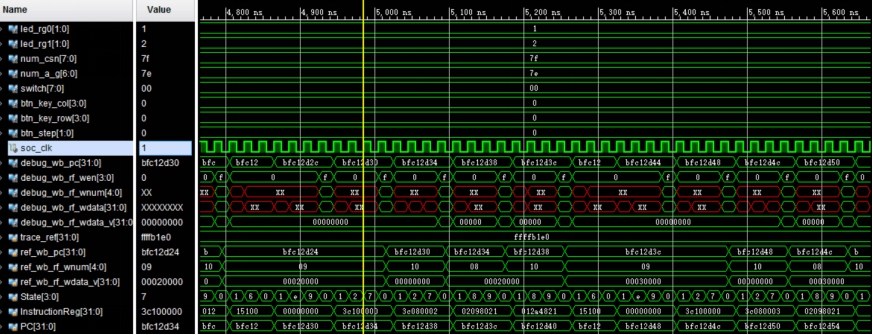
第一个碰到的问题时debug信号给的不太对，主要是debug\_wb\_pc信号的问题，后来根据标准信号看出，每次都是在debug\_wb\_rf\_wen信号为高电平时进行比较，因此需要用寄存器将当前指令PC寄存，并晚一拍连给debug\_wb\_pc信号，这样便可以解决这个问题。

第二个碰到的问题是一开始没有将原设计中的MemDataReg寄存器去掉，该寄存器用来存储从数据存储器中读出的数据，在原先的异步存储器中，由于读数据没有延时，可以认为是立刻得出结果，因此需要用寄存器将其结果暂存，用于后一周期使用，但是在同步存储器中，本身读出数据就需要等到上升沿才完成，因此此时如果再加入一个寄存器，便会反而慢一个周期，从而导致LW指令的实现出现问题，因此去掉这一个寄存器便可以解决问题。

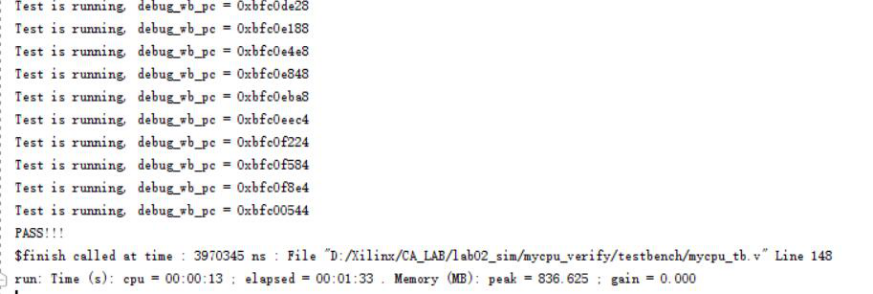
还有的问题就是在仿真和上板测试是遇到问题，首先先进行了仿真测试，这时候没有问题，然后生成bit文件后上板测试，数码管没有显示出变化的过程，此后根据老师的邮件中的解决步骤进行操作，也一直没有解决这个问题，而且此时再进行仿真的话会报错。因此后来只能创建两个工程，一个用来仿真测试，一个用来上板测试，希望老师能在下次实验中给出解决此类问题的完整方案与步骤。

（二）测试结果

1、仿真验证

数据正常更新，仿真波形如下

Tcl console显示程序运行成功，“pass！！！”

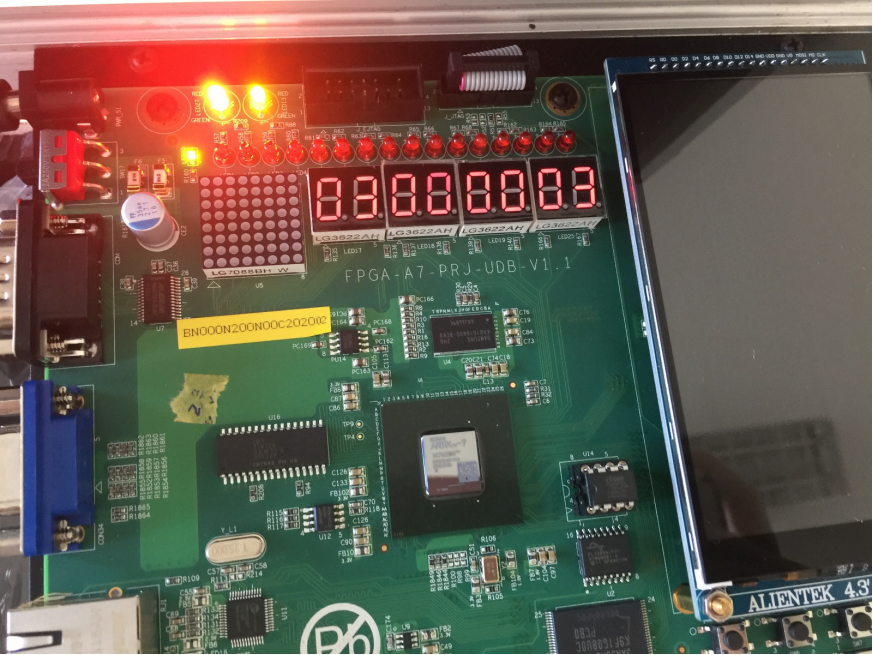


2、FPGA验证

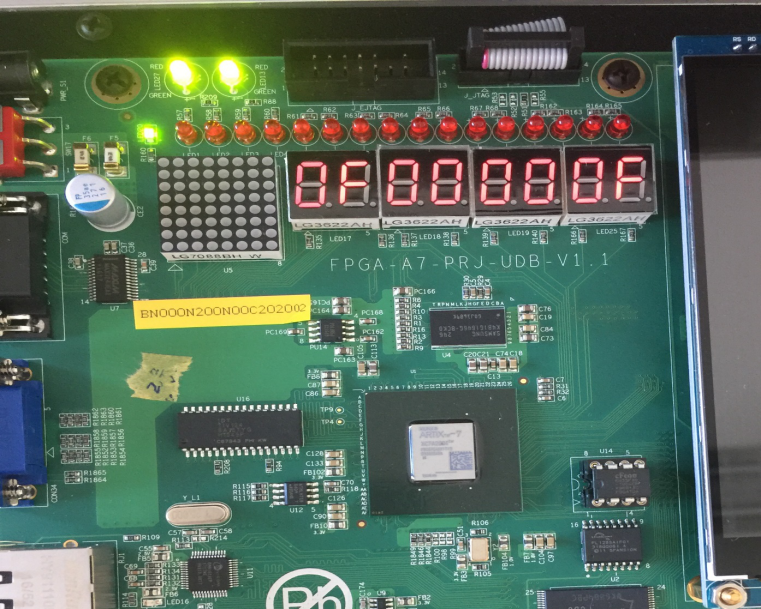
(1) 开始，单色 LED 全灭，双色 LED 灯一红一绿，数码管显示全 0；

起始状态，难以抓拍。

(2) 执行过程中，单色 LED全灭，双色 LED灯一红一绿，数码管高 8位和低 8位同步累加；



(3) 结束时，单色 LED全灭，双色 LED灯亮两绿，数码管高 8位和低 8位数值相同，对应测试功能点数目



五、成员分工

1. 交流讨论整体实验构想（3小时，共同完成）
2. 对上学期的多周期CPU代码进行修改（3小时，共同完成）
3. 调试及上板验证（5小时，共同完成）
4. 完成实验报告（4小时，共同完成）

六、实验总结

（一）组员：袁峥

本次试验整体上上学期计算机组成原理实验中的多周期CPU上进行修改，主要的难点在于要提前想清楚整个流程和结构，具体分析清楚存储器从异步到同步会给整个流程带来哪些影响，具体来说就是指令存储器的读地址需要提前给出，而数据存储器读出数据后的一个数据寄存器可以去掉。还有就是复位信号部分也需要进行一定的修改，让其空等一个周期，相当于让PC准备好，以保证下一拍能够顺利读取指令。

在改完代码进行调试的过程中，首先发现对debug\_wb\_pc信号的理解出现了一定的问题，因此对其又进行了修改。在上板调试过程中第一次没有出现数码管的动态显示，根据老师的邮件方法进行了反复尝试，最终也没能做到一个工程既可以用来仿真又可以用来上板测试，因此最后只能建了两个工程来分别完成。

此次实验的实现在时间上比较匆忙，在截止时间前才开始动工，低估了实验难度，以后要抓紧时间，提前开始实验，这样也能给调试留出更加充分的时间。

（二）组员：施璠

本次实验基于上学期计算机组成原理实验课的多周期处理器，结合上次课的同步RAM，进行修改。原来的momory模块把指令和数据放在一起，而此次的实验把memory一分为二成为inst\_sram 和 data\_sram. 主要的难点在于异步读数据改成了同步读，即在指令RAM和数据RAM中地址信号要提前给出。考虑到在原来的实验中有用到两个寄存器PC和Memdata起到延缓一拍的作用，在本次实验中去掉这两个寄存器，就使得addr信号提前给出。虽然我想在PC上动手，设了一个PC\_reg,多等待一个时钟周期，但是这次实验开始的比较晚，deadline将近也没调出来，最后用队友的代码完成了本次实验。通过本次实验，对前两个学习的关于verilog的知识差不多又找回来了，包括对同步读和异步读的理解，MIPS指令对应的状态机模型。教训就是在以后的实验中，要早动手，不拖到最后。