**计算机组成原理课程设计**

**实验报告**

**多周期/流水线处理器设计与实现**

**小组成员**

**班级：XXXX**

二零一七年九月十八号

**多周期/流水线处理器设计与实现**

1. **设计目的**
2. 了解现代计算机硬件设计的基本流程和方法。
3. 了解典型的 RISC 处理器 MIPS 的体系结构。
4. 了解汇编语言到机器语言到计算机执行软硬的逻辑关系。
5. 掌握处理器的设计原理和方法。
6. 培养寄存器级硬件故障的检错和排错能力。
7. **设计内容**

设计多周期或者流水线控制的处理器，支持 MIPS 指令子集:Lui，Addiu，Add， Lw，Sw，Beq，j，Ori，Addi，Bgtz， Jal，Jr， Sub， And， XOR， BGEZ，OR， Slt等。

1. **设备器材**

操作系统：Windows 10

仿真软件：ModelSim

1. **成员与分工**

|  |  |  |  |
| --- | --- | --- | --- |
| 成员 | 分工 | 完成情况 |  |
| 马昱程（组长） | 结合多周期处理器结构图画出状态转换图，构造FSM，合并FSM获取各控制信号的表达式，编写代码实现，撰写报告 | 完成 |  |
| 陈嘉曦 | 结合多周期处理器结构图画出状态转换图，构造FSM，合并FSM获取各控制信号的表达式，编写代码实现，撰写报告 | 完成 |  |
| 贺熙芮 | 合并FSM获取各控制信号的表达式，撰写报告 | 完成 |  |
| 李立鲲 | 编写代码实现，撰写报告 | 完成 |  |
| 李文瑞 | 编写代码实现，撰写报告 | 完成 |  |
| 刘凌缘 | 合并FSM获取各控制信号的表达式，撰写报告 | 完成 |  |

1. **设计原理及内容**

在多周期处理器设计中，我们把指令的执行分成多个阶段，每个阶段在一个时钟周期内完成，时钟周期以最复杂阶段所花时间为准，尽量分成大致相等的若干阶段，规定每个阶段最多只能完成1次访存或寄存器堆读/写或ALU，每步都设置存储元件，每部执行结果都在下个时钟开始保存到相应单元，通过加入寄存器来切分数据通路从而实现阶段的切分。

设计过程中，在理解了多周期处理器的工作原理之后，我们参考了老师所给的资料中的多周期处理器结构图，并在此基础上设计了7个状态的状态转换图，并借此画出了FSM（原有8个状态，调试过程中发现多余一个后删除，但是在FSM设计时仍然为8个状态，但是s6状态是多余状态）。

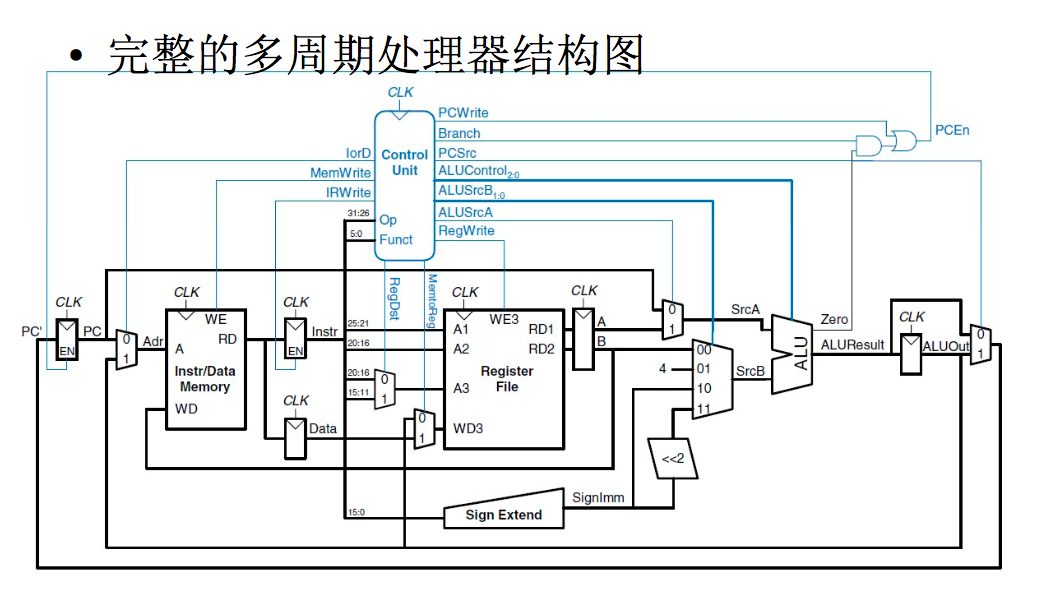


图5-1

一共有18张状态转换图和FSM，这里只给出lw指令的状态转换图和FSM，完整的状态转换图和FSM请参考一起打包的文件。

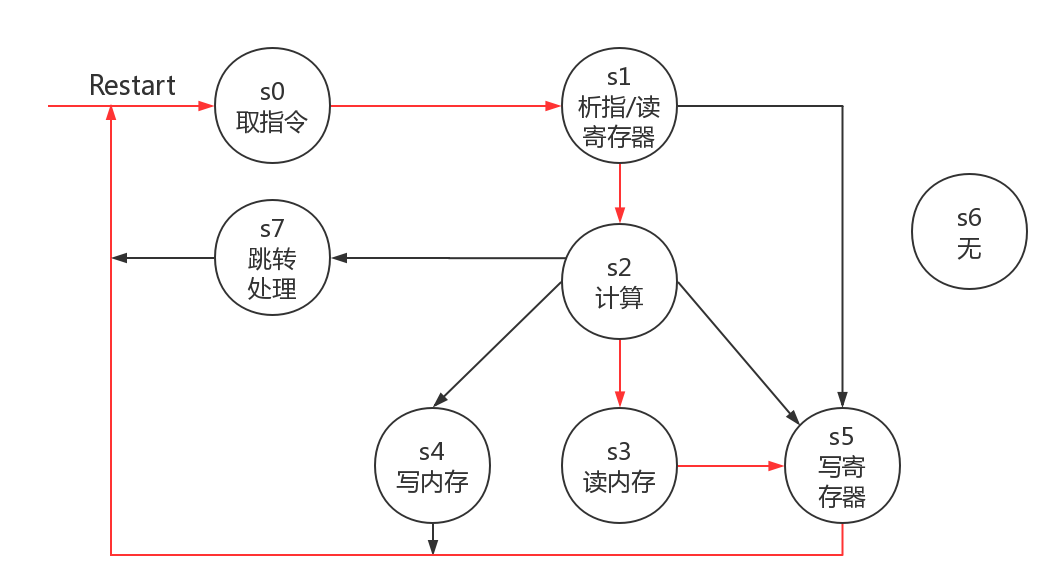


图5-2

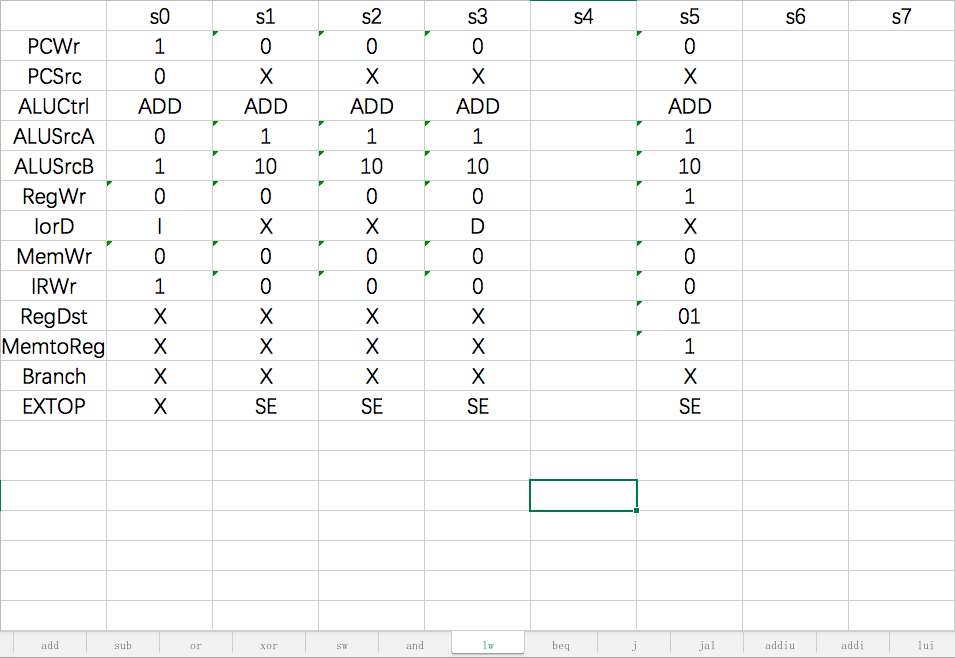


图5-3

1. **设计步骤**

## 画出状态转换图

多周期和单周期最大的区别在于多周期处理器将指令的执行分成了多个阶段，每个阶段在一个时钟周期内实现，因此我们需要确定多周期处理器最多有几个状态，并且根据具体的指令，来确定每条指令的状态转换图。设计过程中，我们参考了资料中的多周期处理器结构图，并以此构造出了s0~s7八个状态，后调试过程中发现s6为多余状态，于是删除，最终剩余7个状态，7个状态分别为：

s0：取指令

s1：分析指令、取寄存器值

s2：alu计算（包括算术运算、逻辑运算、地址运算）

s3：读内存

s4：写内存

s5：写寄存器

s7：跳转处理（用于处理beq、bgez、bgtz一类指令）

其中所有指令以s0状态为起点，并且在一个时钟周期后进入s1状态，随后根据具体指令跳转到不同的状态，以lw指令为例，lw指令的指令说明为：

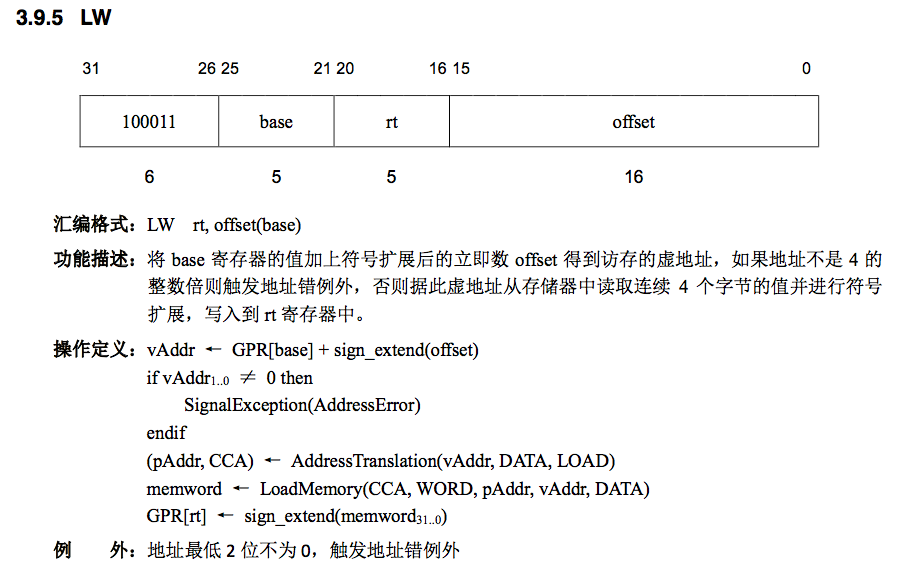


图6-1

可以看到，lw指令需要计算访存的地址，所以下一个状态应该是s2状态进行alu运算来获取内存地址，获取地址后进入下一个s3状态进行访存，最后进入s5状态，将访存的结果存入寄存器，所以最后lw的状态转换图为：

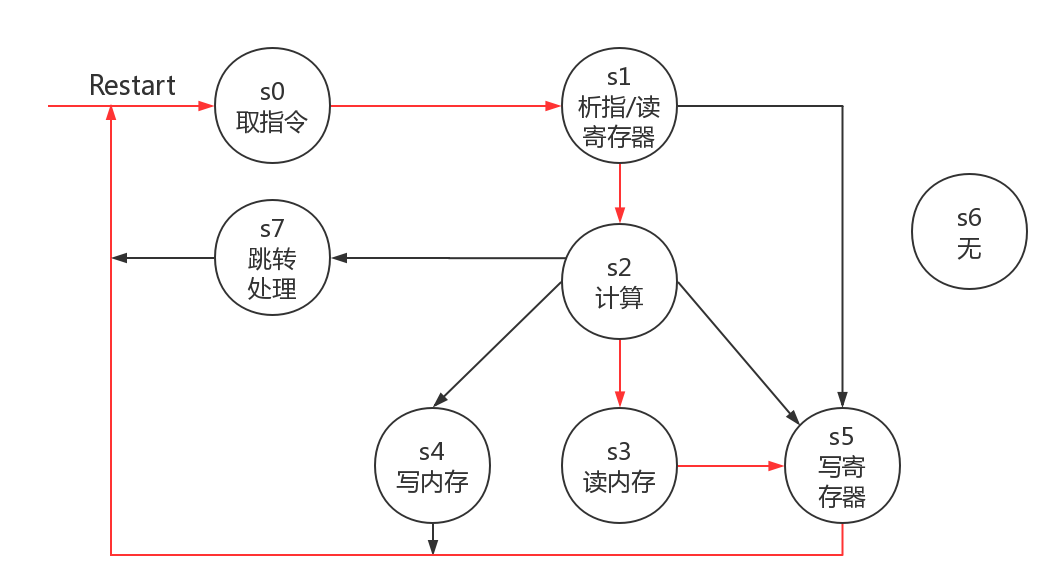


图6-2

一共十八个指令，对应十八个状态转换图，篇幅原因就不在此一一列举了，可参考一起打包的状态转换图文件夹。

## 根据状态转换图构造FSM

画出所有指令的状态转换图后，就可以根据状态转换图来构造各个指令的FSM了。因为我们将地址计算整合到了ALU中，所以和ALU有关的控制信号就相对复杂，同时，指令寄存器和数据寄存器放在一起，所以需要给一个控制信号来确定当前是读指令还是读内存，根据多周期处理器的结构图，我们共有13个控制信号，分别为：

PCWr：控制是否写PC

PCSrc：控制写PC的值的来源，是当前运算结果还是之前运算结果

ALUCtrl：ALU执行何种操作

ALUSrcA：ALU操作数A的来源，取寄存器还是取PC值

ALUSrcB：ALU操作数B的来源，具体指令具体分析

RegWr：寄存器的写使能

IorD：读指令还是读内存

MemWr：内存写使能

IRWr：指令暂存器的写使能，当读内存时，写使能置0

RegDst：选择目标寄存器

MemtoReg：写寄存器的数据来源，来自内存或来自ALU的结果

Branch：跳转控制，在实现过程中我们并没有用到这个控制信号

EXTOP：符号扩展控制，0扩展或者符号扩展或者高位扩展

对于一条具体的指令，在不同的状态每个控制信号都有不同的值，所以我们对每个控制信号都需要构造一张FSM表来表现，同样以lw指令为例：

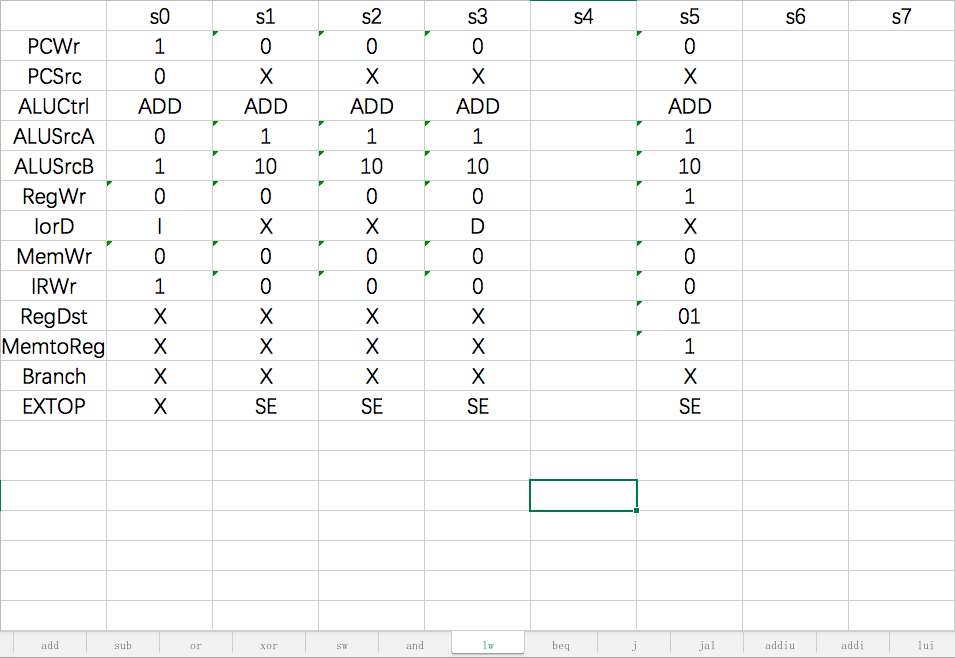


图6-3

在s0状态，因为是取指状态，所以我们将IorD、IRWr置为1，同时，取完当前指令后，PC应该指向下一条指令地址，即PC=PC+4，所以和ALU相关的三个控制信号也要做相应的设置，在之后的所有状态中，因为当前PC指针已经指向了下一条指令，所以IRWr控制信号均置为0，不能置为1，否则就会直接跳跳向下一条指令。在s1状态，我们做一个分析指令和取寄存器的操作，这里可以提前改变和ALU有关的控制信号，让ALU为计算做好准备。s2状态即进行alu运算获取内存地址。s3阶段因为需要读取内存，所以IorD控制信号置为D，即读内存操作，读取的数据存储在数据暂存器中。最后一个s5状态即为写寄存器状态，将寄存器写使能置1，根据指令格式设置RegDst控制信号，并且将MemtoReg控制信号置为1，即写寄存器的数据来源为内存数据。

对于每一条指令，都需要构造这么一个FSM，一共十八个FSM，同样因为篇幅原因就不在此一一列举了，可以在fsm.xlsx文件中看到所有指令的FSM。

## 合并FSM获取各控制信号的表达式并化简

在构造了所有FSM之后，就需要对所有FSM进行合并，获取各个控制信号的表达式，共有十八条指令，对照表格各个控制信号的所有真值抽取合并，获得一个真值表达式，进行化简之后即可得到各个控制信号的表达式，以IRWr为例，IRWr仅仅在s0状态为1，其他状态均为0，而所有指令有均以s0状态为起始状态，所以IRWr的表达式可以简单的写成IRWr = s0。

由于此步骤均在纸面上完成，在此仅给出代码形式的结果



图6-4

其中s125等类似写法表示在s1或s2或s5状态。

## 代码实现

完成设计之后就是代码实现的过程，其中代码实现过程分为两部分：第一部分是编写代码，第二部分是进行仿真调试。

编写代码部分，我们根据多周期处理器的结构图，为每个寄存器、暂存器分别编写了一个模块，最后实例化即可。以ALU模块为例：

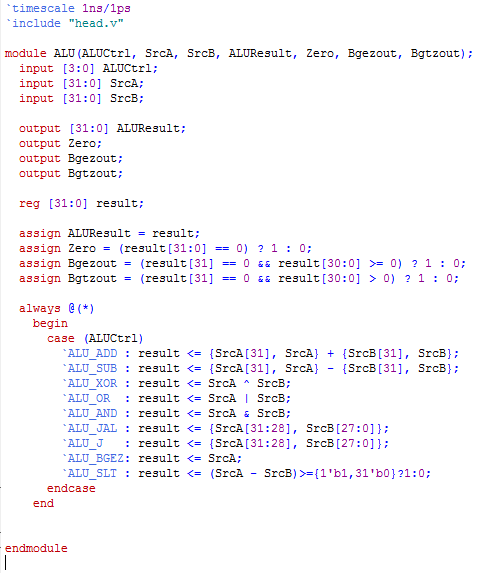
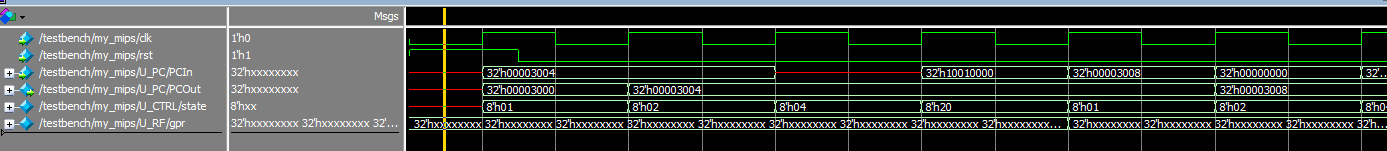


图6-5

共有三个控制信号输入，四个信号输出，其中Zero，Bgezout，Bgtzout三个信号又反向输出给控制模块，从而来控制跳转。

调试阶段，我们在ModelSim中进行了调试，根据state的波形以及各个模块的输入输出来确认代码的正确与否。



在波形中可以看到，PC指针在s0取指状态后就指向了下一条指令，即PC=PC+4

1. **遇到的问题及解决方法**

设计以及实现过程中，我们也遇到了很多问题：

1. 如何理解多周期

我们遇到的第一个问题就是该如何理解“多周期”这个概念？！我们最开始将“多周期”理解为了“流水线”，即多条指令并行执行，后发现理解有误，在经过小组讨论以及对老师给的资料进行进一步详细解读之后，我们重新理解了“多周期”的概念。

1. 状态转换图问题

在画状态转换图的过程中，我们小组内进行了讨论，因为将地址计算整合到了ALU中，所以在执行一些跳转指令比如j指令、jal指令、beq指令等会出现一些问题。最终讨论的结果是我们确定了八个状态，其中的s6状态就是专门为了处理jal指令中的指令跳转的，jal最初的状态转换是s0->s6->s5但是在后来调试的过程中发现跳过了分析指令的过程会导致该指令执行不正确，在讨论之后我们最终删除了s6状态，并重新画了jal指令的状态转换图和FSM。

1. **设计总结**

A：

作为组长，这次设计实验的压力还是比较大的，上学期的计算机组成原理课程、硬件系统设计课程算是为这次实验打下了一定的基础。

我个人认为这次实验最大的困难就是该如何去理解“多周期”这个概念，前面也提到了我们小组最开始将“多周期”和“流水线”两个概念混为一谈，实际上是不对的。在理解清楚“多周期”和“流水线”这两个概念之后，设计过程中的一些问题就能看的比较清楚，比如如何划分状态，状态之间怎么切换（通过时钟）等。

整个实验过程我都参与了其中，个人认为最困难的阶段是最后的调试阶段。在调试阶段，我们编写了一份包含了所有指令的代码来进行测试，可是我们最终的代码实现始终会出现问题，最后的结果始终不对，这时候就需要不断地去查看波形，一个时钟周期一个时钟周期地去看，查在哪条指令的哪个状态出了问题，再回去看状态转换图和FSM是否有问题再根据出现的问题进行相应的修改。一般到最后我们往往发现是设计的时候出现了问题，最典型的例子就是我们的jal指令，设计的时候就出现了问题，导致代码实现的时候指令执行的时候会朝一个完全错误的方向进行，最后通过修改状态转换图、FSM和相关代码后，问题最终得到了解决。如果当初我们设计的时候考虑更加周到，在最后的调试阶段也就不会这么辛苦，一条指令出现错误基本上就要推倒重来。今后我会更加注重设计的严谨，争取把问题解决在设计阶段。

B：

通过这次的实验，我对于计算机组成原理以及体系结构更加的了解，之前只是对理论有了解，但是对于细节，没有什么认知。通过实现多周期CPU，我明白了CPU如何进行对机器码进行硬件上的实现，以及CPU的数据通路。

实验中，我们并没有完全使用老师所给的数据通路以及状态图，而是对其进行了简化，实现了状态的减少和ALU的复用，为此我们对每个标志位的获得进行了重新的计算，为此我们付出了大多数的时间，但是当我们的CPU可以对编译出来的机器码进行正确的运行时，是十分开心的。

由于我们对某些指令出现了理解上的错误，因此也出现过错误，例如JAL，通过这次实验，我们对于各个指令的执行也有了更加深入的了解。

总而言之，多周期CPU的实现，对于我在计算机组成原理上有了很大的帮助。

C：

在本次实验我负责的是多周期代码的编写，用其他组员设计好的指令的通路图来实现具体的指令。在编写单周期的实验时，我并没有自己独立完成所有的代码部分，通过修改老师的样例即可达到目的，但是在多周期是自己要独立完成代码，虽然思路上大体和单周期一致，但是实践的时候还是写错了很多地方，经过了仔细的检查和组长的帮助克服了问题。本次实践中，我学习到了多周期cpu实现的原理，并且实践完成了多周期cpu，这其中少不了小组成员的共同努力，体会到了团队合作的乐趣。

D：

通过之前的单周期的实验，对硬件逻辑有了一定简单的了解，在多周期实验中，在之前的基础上进一步拓展，注意到了很多之前没有注意到的细节。在实验中还了解到了测试程序的方法，通过实践解决遇到的问题，在这个过程中更深一步的了解了处理器的设计原理和方法，同时排错能力有了一定的提高。

E：

在小学期我们学习了计算机组成原理课程设计这门课程，它是一门综合应用计算机组成原理课程的知识和内容来解决问题并实现功能的课程，

通过学习这门课的知识并完成实验的过程，了解了现代计算机硬件设计的基本流程和方法。了解了典型的RISC 处理器MIPS 的体系结构。了解了汇编语言到机器语言到计算机执行软硬的逻辑关系。掌握了处理器的设计原理和方法。培养了寄存器级硬件故障的检错和排错能力。

并且运用这些知识，成功地设计了设计多周期或者流水线控制的处理器，支持MIPS 指令子集：Lui，Addiu，Add，Lw，Sw，Beq，j，Ori，Addi，Bgtz， Jal，Jr， Sub， And， XOR， BGEZ，OR，Slt。

课程知识的实用性很强，因此实验就显得非常重要，刚开始做实验的时候，由于自己的理论知识基础不好，在实验过程遇到了许多的难题，也使我感到理论知识的重要性。但是我并没有气垒，在实验中发现问题，独立思考，团队合作，最终解决问题，从而也就加深我对理论知识的理解。

F：

通过之前的单周期设计，我已经简单的了解了如何实现一个CPU，多周期CPU在单周期CPU基础进行改进。基本框架是相同的，但是相比单周期CPU的设计最大的不同就是“多周期”这一概念，多周期CPU和单周期CPU的设计相同，但多周期CPU指的是将整个CPU的执行过程分成几个阶段，每个阶段用一个时钟去完成，然后开始下一条指令的执行，而每种指令执行时所用的时钟数不尽相同，由此也衍生出了其他的几个区别，比如，数据传输的延迟问题，增加的跳转指令等使得数据通路图变得复杂了很多。

多周期CPU是小组作业，虽然在实验中遇到了不少困难，如果仅凭一人之力，要完成这个实验肯定更加困难，但通过小组的共同努力，终于得出了满意的结果，感受到了团队协作的重要性。

1. **参考文献**
2. d1MIPS指令集合和汇编数据通路
3. d2单周期控制和多周期处理器
4. 57条MIPS指令说明
5. MIPS32 Architecture For Programmers Volume II/The MIPS32 Introduction Set
6. 数字设计和计算机体系结构（含目录）