计算机体系结构课程第 二 次实验报告

**实验名称：多周期CPU的实现设计**

学号：2011743 姓名：高祎珂

1. 实验目的
2. 在单周期CPU实验完成的提前下，理解多周期的概念。
3. 熟悉并掌握多周期CPU的原理和设计。
4. 进一步提升运用verilog语言进行电路设计的能力。
5. 为后续实现流水线cpu的课程设计打下基础。
6. 实验内容说明

多周期CPU是指，一条指令需要花费多个周期才能完成所有操作，在每个周期内只做一部分操作，比如：取指、译码、执行、访存、写回，此时，一条指令执行完，共需5个周期，每个周期只做一部分操作。本次实验是需要用到之前实验的结果的，比如ALU模块、寄存器堆模块、指令ROM模块和数据RAM模块，其中ROM和RAM使用调用库IP实例化的同步存储器，本次实验就是将实验六所实现的单周期CPU划分为多周期的，并扩展指令条数。

在之前的实验中生成的同步RAM和ROM，都是在发送地址后的下一拍才能获得对应数据的。故在使用同步存储器时，从指令和数据存储器中读取数据就需要等待一拍时钟了，即取指令需要两拍时间，load操作也需要两拍时间。在真实的处理器系统中，取指令和访存其实都是需要多拍时钟的。

执行的指令如下：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **指令类型** | **汇编指令** | **指令码** | **源操作**  **数1** | **源操作**  **数2** | **目的寄**  **存器** | **功能描述** |
| R型指令 | addu rd , rs , rt | 000000 rs | rt | rd |00000|100001 | [rs] | [rt] | rd | GPR[rd]=GPR[rs]+GPR[rt] |
| subu rd , rs , rt | 000000 rs | rt | rd |00000|100010 | [rs] | [rt] | rd | GPR[rd]=GPR[rs]-GPR[rt] |
| and rd , rs , rt | 000000 rs | rt | rd |00000|100100 | [rs] | [rt] | rd | GPR[rd]=GPR[rs] & GPR[rt] |
| or rd , rs , rt | 000000 rs | rt | rd |00000|100101 | [rs] | [rt] | rd | GPR[rd]=GPR[rs]｜GPR[rt] |
| nor rd , rs , rt | 000000 rs | rt | rd |00000|100111 | [rs] | [rt] | rd | GPR[rd]=(!GPR[rs]&GPR[rt])｜(!GPR[rt]&GPR[rs]) |
| sll rd , rs , rt | 000000 rs | rt | rd |00000|000000 | [rs] | [rt] | rd | GPR[rd]=GPR[rs]<<GPR[rt]zero |
| srl rd , rs , rt | 000000 rs | rt | rd |00000|000010 | [rs] | [rt] | rd | GPR[rd]=zeroGPR[rs]>>GPR[rt] |
| sra rd , rs , rt | 000000 rs | rt | rd |00000|000011 | [rs] | [rt] | rd | GPR[rd]=signGPR[rs]>>GPR[rt] |
| jr rs | 000000 rs | 00000 | 00000 |00000|000011 | [rs] |  |  | PC=GPR[rs] |
| I型指令 | addiu rt,rs,imm | 001001 rs | rt |imm | [rs] | sign\_ext(imm) | rt | GPR[rt]=GPR[rs]+sign\_ext(imm) |
| andiu rt,rs,imm | 001100 rs | rt |imm | [rs] | sign\_ext(imm) | rt | GPR[rt]=GPR[rs]&sign\_ext(imm) |
| oriu rt,rs,imm | 001101 rs | rt |imm | [rs] | sign\_ext(imm) | rt | GPR[rt]=GPR[rs] | sign\_ext(imm) |
| xoriu rt,rs,imm | 001110 rs | rt |imm | [rs] | sign\_ext(imm) | rt | GPR[rt]=(!GPR[rs]& sign\_ext(imm))｜(! sign\_ext(imm)&GPR[rs]) |
| lw rt,rs,imm | 100011 rs | rt |imm | [rs] | sign\_ext(imm) | rt | GPR[rt]=MEM[GPR[rs]+sign\_ext(imm)] |
| sw rt,rs,imm | 101011 rs | rt |imm | [rs] | sign\_ext(imm) | rt | MEM[GPR[rs]+sign\_ext(imm)]=GPR[rt] |
| beq rt,rs,imm | 000100 rs | rt |imm | [rs] | sign\_ext(imm) | rt | if(GPR[rs]==GPR[rt])  PC=sign\_ext(imm)\*4 |
| bne rt,rs,imm | 000101 rs | rt |imm | [rs] | sign\_ext(imm) | rt | if(GPR[rs]==GPR[rt])  PC=sign\_ext(imm)\*4 |
| lui rt,rs,imm | 001111 00000 | rt |imm |  | sign\_ext(imm) | rt | GPR[rt]=sign\_ext(imm),zero |
| J型指令 | j target | 000010|target | PC | target |  | 跳转, PC={PC[31:28],target,2’b00} |
| jal target | 001100|target | PC | target |  | 调用, PC={PC[31:28],target,2’b00} |

1. 实验原理图

多周期CPU设计在单周期CPU基础上，主要做两部分改进。第一部分是控制单元，增加控制电路使每一个时钟只有一个阶段的电路产生的结果有效，并锁存上一阶段的结果用于后续阶段的运行；第二部分是数据通路，增加实现新增指令的电路。

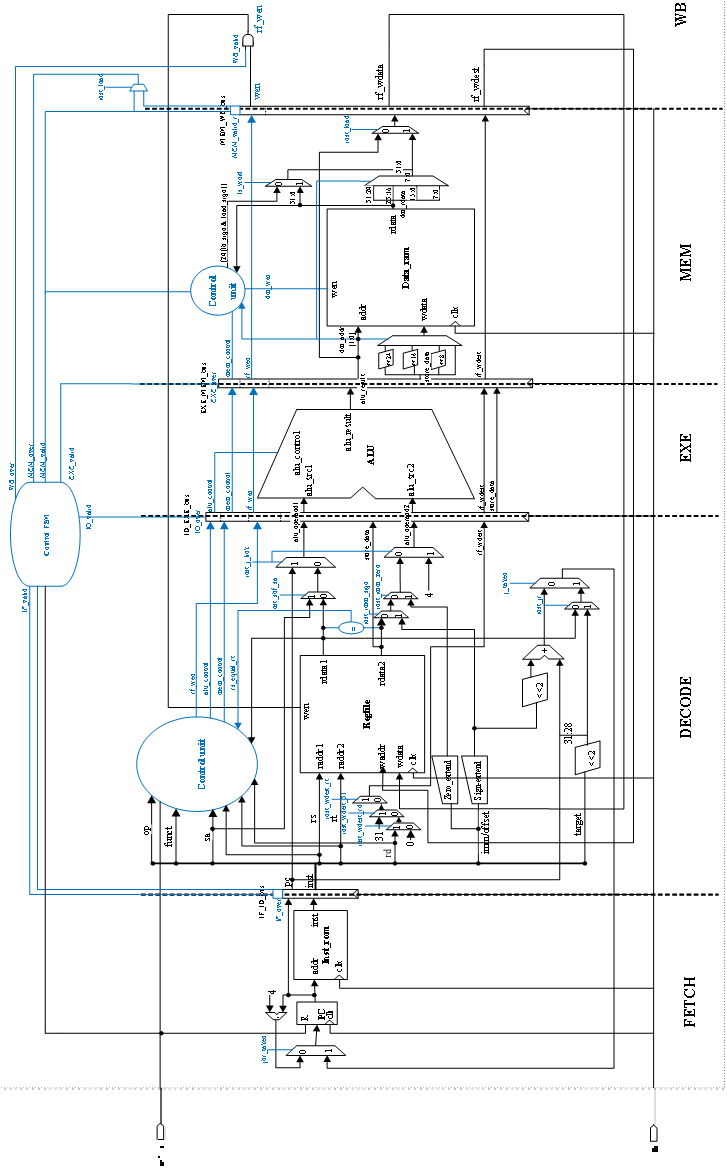
第一部分的改进主要是增加状态机控制及增加各阶段之间的用于锁存的寄存器。由于有5个阶段，状态机共有6个状态：空闲（IDLE）、取指（FETCH）、译码（DECODE）、执行（EXE）、访存（MEM）、写回（WB），如下图：

多周期CPU的状态机

实验需要完成的内容是：

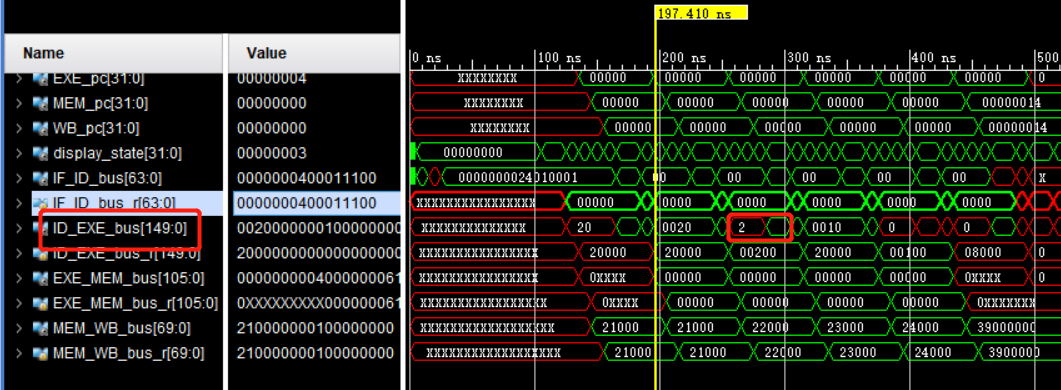
* 确认多周期CPU的设计框图的正确性；
* 编写verilog代码，实现多周期CPU；
* 编写一段汇编程序，并将其翻译为二进制，以coe文件的方式初始化到指令ROM中；
* 编写testbench，对模块进行仿真，观察实验波形，改进代码得出正确的波形；
* 对代码进行综合布局布线下载到实验箱里FPGA板上，进行上板验证；

多周期CPU的实现框图如下：



1. 实验步骤

在所给的源码中，存在一部分bug,经过初步仿真模拟，观察仿真结果，



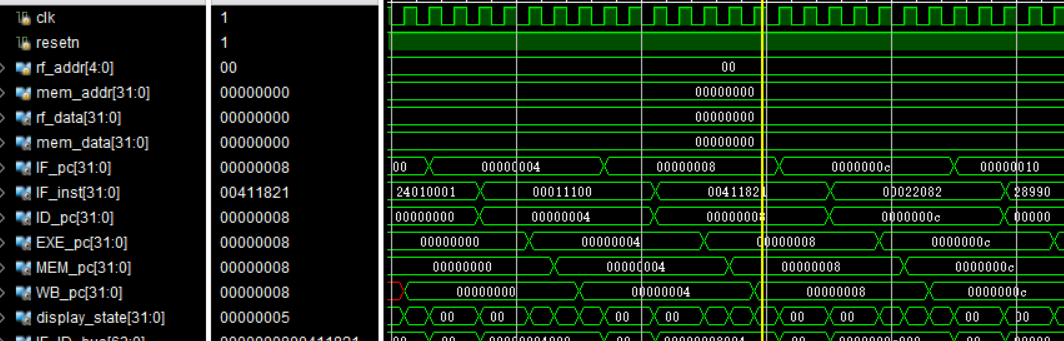
可以看到，该条对应的指令本是第六条bgez $25,#16，但此时分支地址处于不确定的状态，跳转出现了问题。

对仿真图分析可知，指令实际上传输得到的inst指令比预估的总是慢一拍，这导致指令的之前其实是慢一条的，但是由于组合逻辑实际上一直在向前递进这就导致了指令的错误重叠，从仿真图可以看出IF从IP核中取到指令内容这个过程实际上发生在IF阶段结束，ID阶段即将上跳沿进入的时机，那么这就会导致IF\_ID\_bus\_r这条总线中保存的PC值与inst内容实际上是不对应的，inst内容比PC慢一条指令。

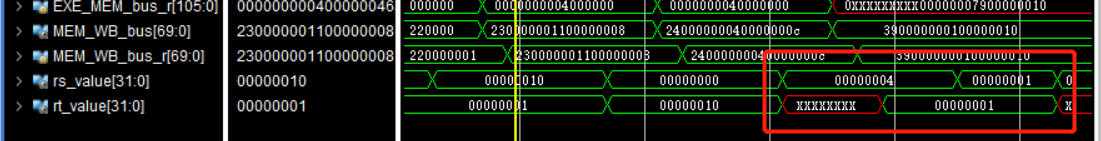
查看代码后发现，传值和状态转换为ID的过程受同一个信号影响，因此通过修改ID\_valid的值来确保转换状态时IF\_ID\_bus的值完全被传入到IF\_ID\_bus\_r中。



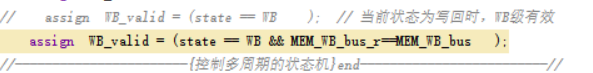
修改之后，可以发现跳转指令执行时，寄存器值已经正常了。



但是此时观察波形，lw指令依旧无法正确满足需要，对于lw $10,#20($1) [$10] = 0000\_0011H，此时rt\_value不确定，对于上一次错误，此时想到，会不会也是访问IP核，数据内容没有及时取出导致的。



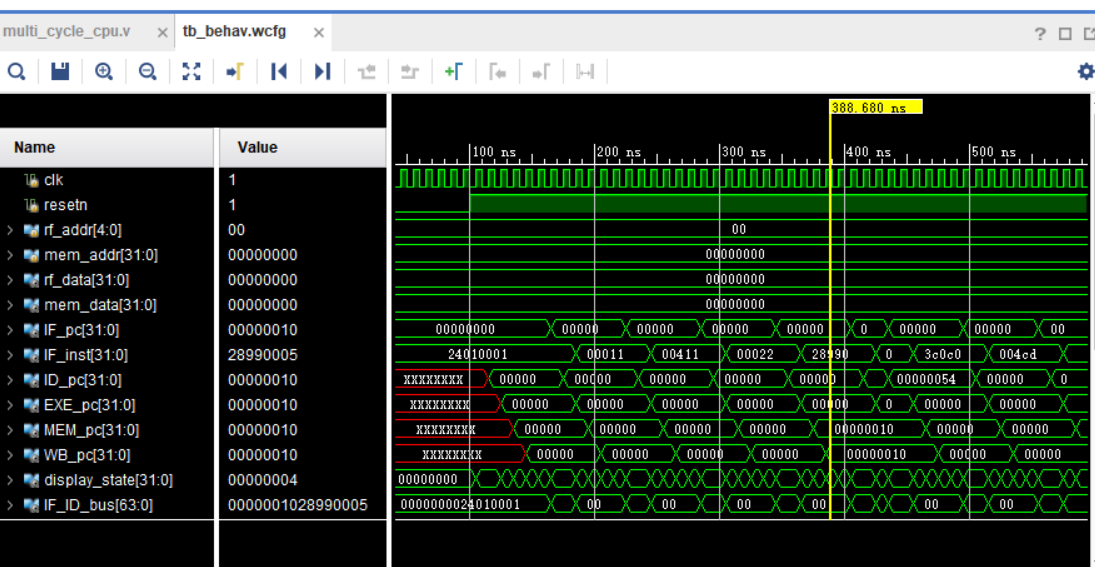
因此修改代码

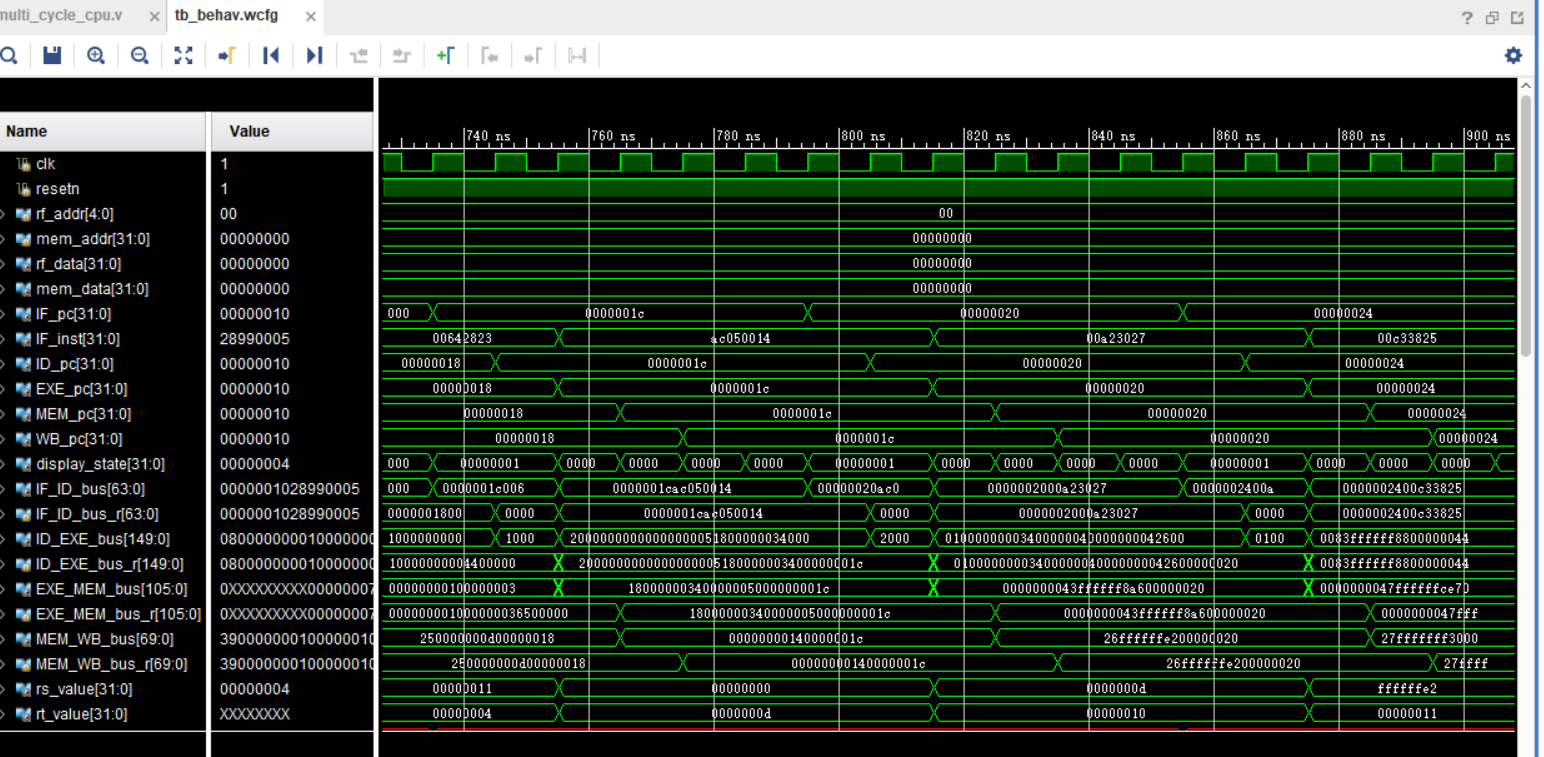


可以看到值恢复了正常。

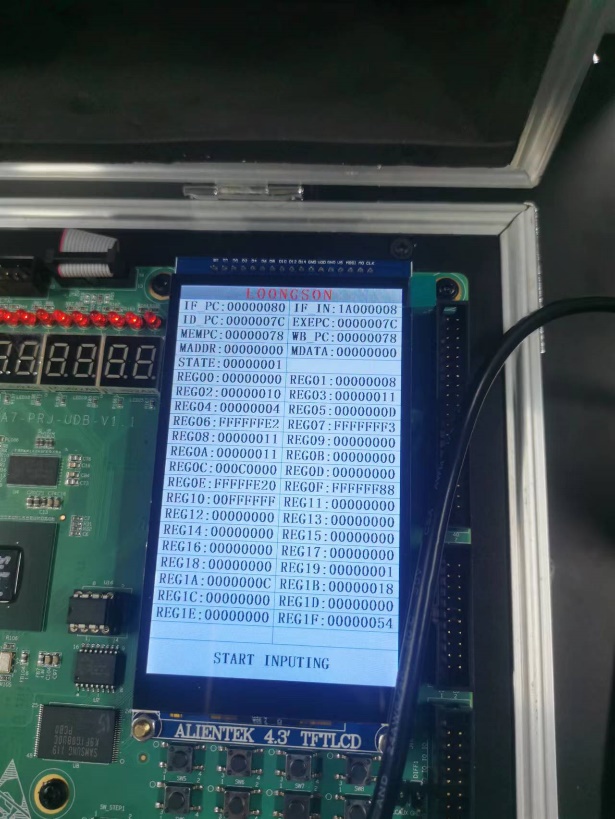
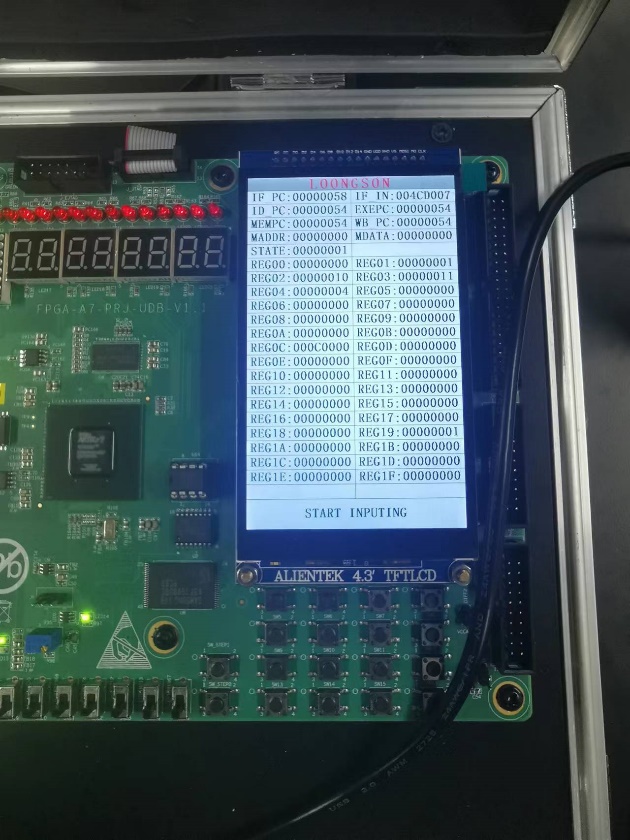
1. 实验结果

仿真结果：





上箱结果：



1. 总结

从单周期 CPU 到多周期 CPU 再到流水线 CPU 的实现是一个循序渐进的过程。先通过单周期 CPU 的实现基本完成各个模块的功能联合和实现，然后通过多周期 CPU的改进，将本来是一个 CLK 始终周期完成的任务，分步成为 5 个周期进行完成。多周期的变换关键在于始终周期，要判断每次在时钟上升沿将要发生时，数据有没有准确的传递到下一个阶段，一句此进行代码的修正。