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

一、实验目的

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

二、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：

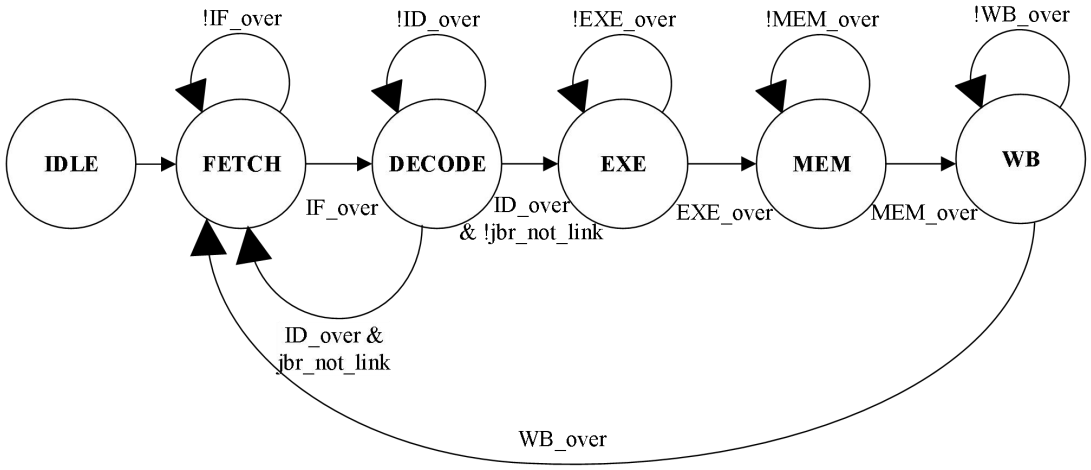
指令类型	汇编指令	指令码	源操作数 1	源操作数 2	目的寄存器	功能描述
R 型指令	addu rd , rs , rt	000000|rs|rt|rd|00000|100001	[rs]	[rt]	rd	GPR[rd]=GPR[rs]+GPR[rt]
	subu rd , rs , rt	000000|rs|rt|rd|00000|100010	[rs]	[rt]	rd	GPR[rd]=GPR[rs]-GPR[rt]
	and rd , rs , rt	000000|rs|rt|rd|00000|100100	[rs]	[rt]	rd	GPR[rd]=GPR[rs] & GPR[rt]
	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}

三、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：

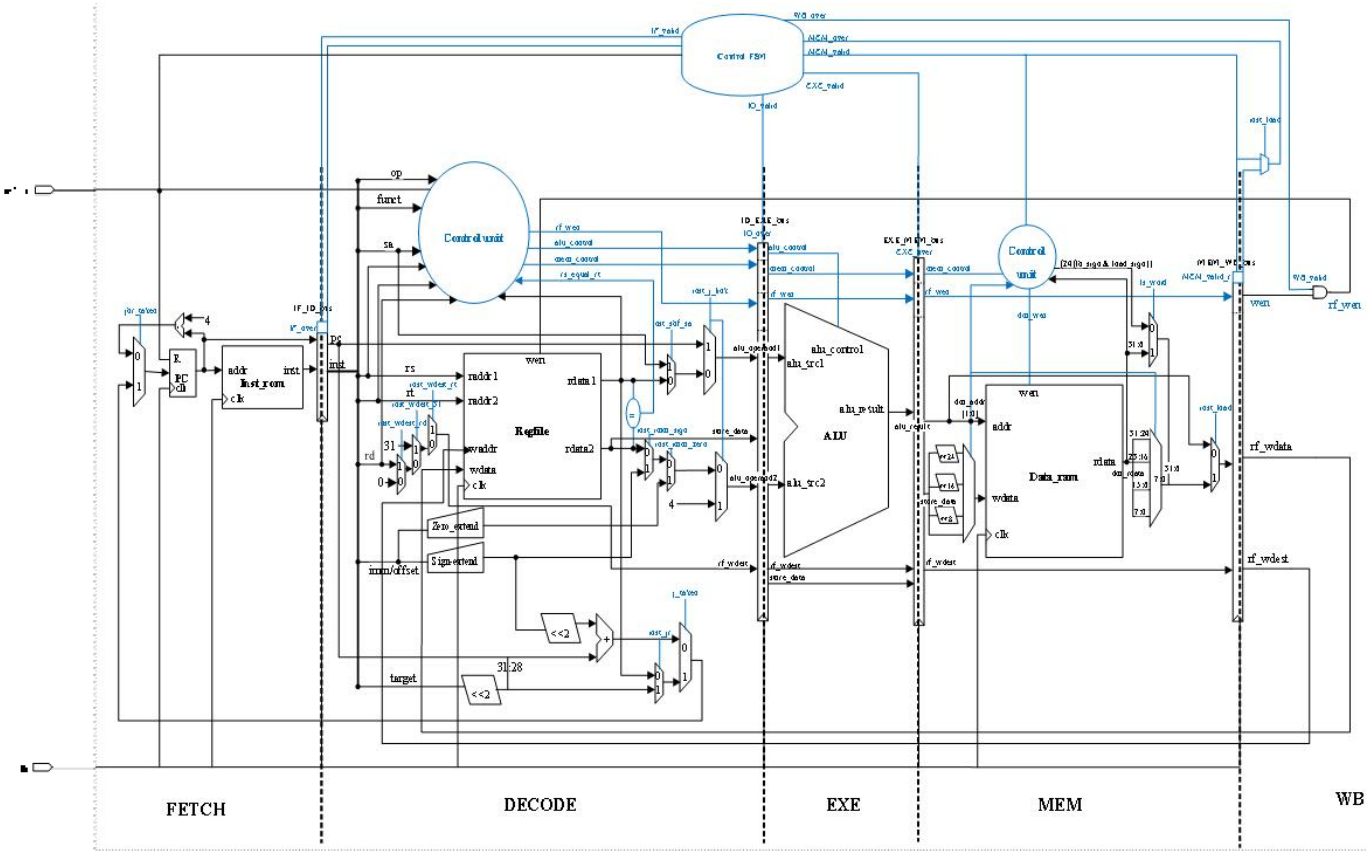


多周期 CPU 的状态机

实验需要完成的内容是：

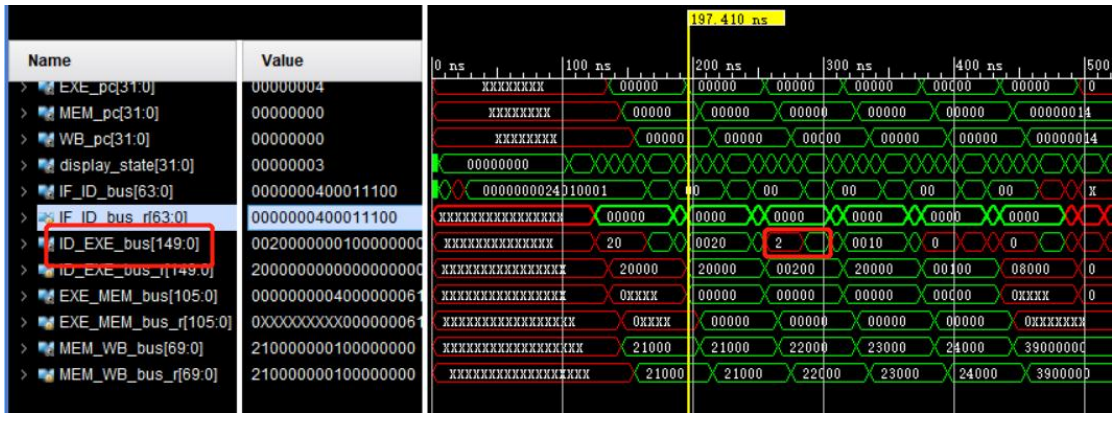
- 确认多周期 CPU 的设计框图的正确性；
- 编写 verilog 代码，实现多周期 CPU；
- 编写一段汇编程序，并将其翻译为二进制，以 coe 文件的方式初始化到指令 ROM 中；
- 编写 testbench，对模块进行仿真，观察实验波形，改进代码得出正确的波形；
- 对代码进行综合布局布线下载到实验箱里 FPGA 板上，进行上板验证；

多周期 CPU 的实现框图如下：



四、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。

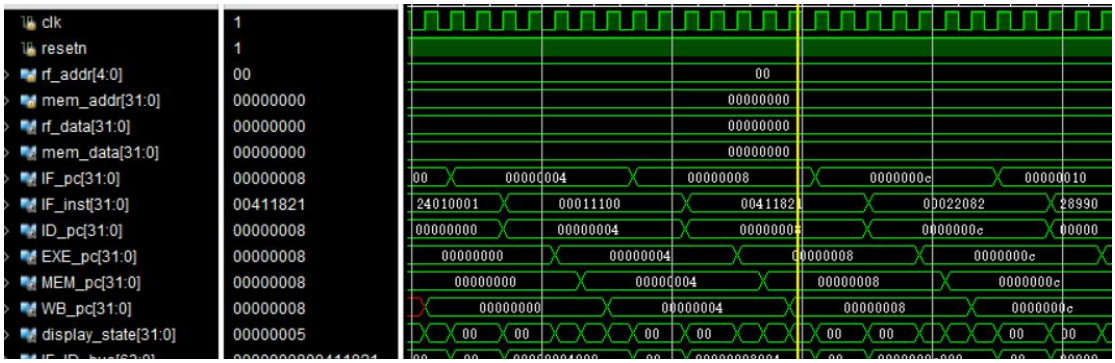
对仿真图分析可知，指令实际上传输得到的 inst 指令比预估的总是慢一拍，这导致指令的之前其实是慢一条的，但是由于组合逻辑实际上一直在向前递进这就导致了指令的错误重叠，从仿真图可以看出 IF 从 IP 核中取到指令内容这个过

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。

```
//    assign  ID_valid = (state == DECODE);  // 当前状态为译码时，ID级有效
  assign  ID_valid = (state == DECODE&&IF_ID_bus_r==IF_ID_bus);
```

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。

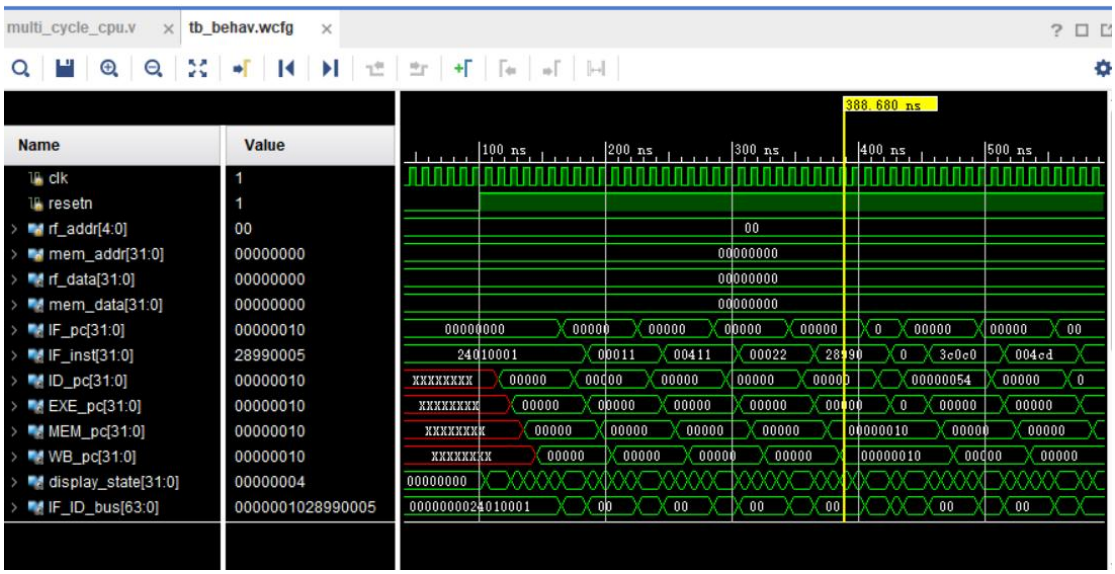
因此修改代码

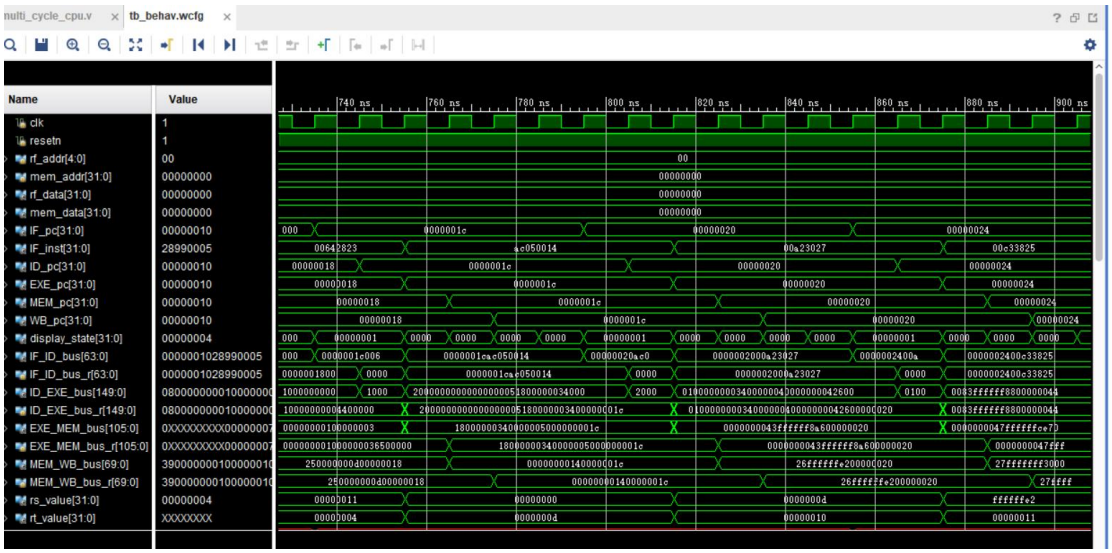
```
//    assign  WB_valid = (state == WB     );  // 当前状态为写回时，WB级有效
    assign  WB_valid = (state == WB && MEM_WB_bus_r==MEM_WB_bus    );
//----------------------{控制多周期的状态机}end----------------------//
```

可以看到值恢复了正常。

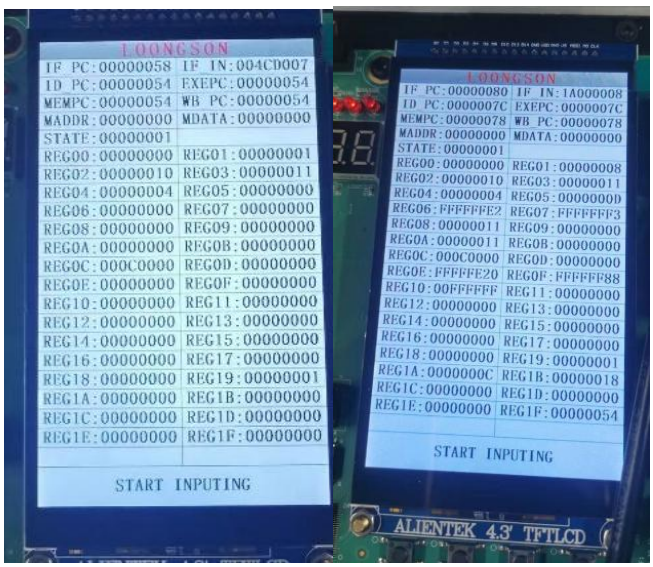
五、实验结果

仿真结果：


multi_cycle_cpu.v
tb_behav.wcfg
Name
Value
clk
resetn
rf_addr[4:0]
mem_addr[31:0]
rf_data[31:0]
mem_data[31:0]
IF_pc[31:0]
IF_inst[31:0]
ID_pc[31:0]
EXE_pc[31:0]
MEM_pc[31:0]
WB_pc[31:0]
display_state[31:0]
IF_ID_bus[63:0]
388.680 ns
100 ns
200 ns
300 ns
400 ns
500 ns


multi_cycle_cpu.v
tb_behav.wcfg
Name
Value
clk
resetn
rf_addr[4:0]
mem_addr[31:0]
rf_data[31:0]
mem_data[31:0]
IF_pc[31:0]
IF_inst[31:0]
ID_pc[31:0]
EXE_pc[31:0]
MEM_pc[31:0]
WB_pc[31:0]
display_state[31:0]
IF_ID_bus[63:0]
IF_ID_bus_r[63:0]
ID_EXE_bus[149:0]
ID_EXE_bus_r[149:0]
EXE_MEM_bus[105:0]
EXE_MEM_bus_r[105:0]
MEM_WB_bus[69:0]
MEM_WB_bus_r[69:0]
rs_value[31:0]
rt_value[31:0]
740 ns
760 ns
780 ns
800 ns
820 ns
840 ns
860 ns
880 ns
900 ns

上箱结果：


LOONGSON
IF PC:00000058 IF IN:004CD007
ID PC:00000054 EXEPC:00000054
MEMPC:00000054 WB PC:00000054
MADDR:00000000 MDATA:00000000
STATE:00000001
REG00:00000000 REG01:00000001
REG02:00000010 REG03:00000011
REG04:00000004 REG05:00000000
REG06:00000000 REG07:00000000
REG08:00000000 REG09:00000000
REG0A:00000000 REG0B:00000000
REG0C:000C0000 REG0D:00000000
REG0E:00000000 REG0F:00000000
REG10:00000000 REG11:00000000
REG12:00000000 REG13:00000000
REG14:00000000 REG15:00000000
REG16:00000000 REG17:00000000
REG18:00000000 REG19:00000001
REG1A:00000000 REG1B:00000000
REG1C:00000000 REG1D:00000000
REG1E:00000000 REG1F:00000000
START INPUTING
LOONGSON
IF PC:00000080 IF IN:1A000008
ID PC:0000007C EXEPC:0000007C
MEMPC:00000078 WB PC:00000078
MADDR:00000000 MDATA:00000000
STATE:00000001
REG00:00000000 REG01:00000008
REG02:00000010 REG03:00000011
REG04:00000004 REG05:0000000D
REG06:FFFFFFE2 REG07:FFFFFFF3
REG08:00000011 REG09:00000000
REG0A:00000011 REG0B:00000000
REG0C:000C0000 REG0D:00000000
REG0E:FFFFFE20 REG0F:FFFFFF88
REG10:00FFFFFF REG11:00000000
REG12:00000000 REG13:00000000
REG14:00000000 REG15:00000000
REG16:00000000 REG17:00000000
REG18:00000000 REG19:00000001
REG1A:0000000C REG1B:00000018
REG1C:00000000 REG1D:00000000
REG1E:00000000 REG1F:00000054
START INPUTING
ALIENTEK 4.3' TFTLCD

六、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。