体系结构实验课程第__一__次实报告

实验名称	**多周期 CPU 实现**			班级	李雨森
学生姓名	李娅琦	学号	2213603	指导老师	董前琨
实验地点	A304		实验时间	星期一 12：00-13：30	

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. 实验要求

1. 做好预习：
1） 复习单周期 CPU 的实验内容，归纳常用的 MIPS 指令，确定自己准备实现的 MIPS 指令，对其进行分析，完成表 8.1 的填写；
2） 依据自己设计中实现的指令，编写一段不少于 40 行的汇编程序，要求包含所有实现的指令，完成表 8.2 的填写；
3） 认真学习多周期的概念，了解流水线的概念，明白划分为多周期的意义；
4） 认真学习 CPU 各模块的功能，确认模块的划分。设计本次实验的方案，画出实验方案的设计框图，即补充完善图 8.1；
5） 如果对 FPGA 板了解的话，可确定设计中与 FPGA 板上交互的接口，画出包含外围模块的整体设计框图，即补充完善图 8.2。
2. 实验实施：
1） 确认多周期 CPU 的设计框图的正确性；
2） 编写 verilog 代码，将表 8.2 中自己编写的汇编程序翻译为二进制，以 coe 文件的方式初始化到指令 ROM 中；
3） 对该模块进行仿真，得出正确的波形，截图作为实验报告结果一项的材料，在仿真时需要将生成指令 ROM 时产生的.mif 文件拷贝到工程目录下，才能仿真成功；
4） 完成调用多周期 CPU 的外围模块的设计，并编写代码；
5） 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。
3. 实验检查：
1） 完成上板验证后，让指导老师或助教进行检查，进行现场演示。先解读表 8.2 中自己编写的汇编程序，然后采用手动输入时钟，每个周期查看 CPU 状态，按照检查人员的要求进行演示，检查指令运行结果的正确性，可对演示结果进行拍照作为实验报告结果一项的材料。
4. 实验报告的撰写：
1） 实验结束后，需按照规定的格式完成实验报告的撰写。

三．实验原理图

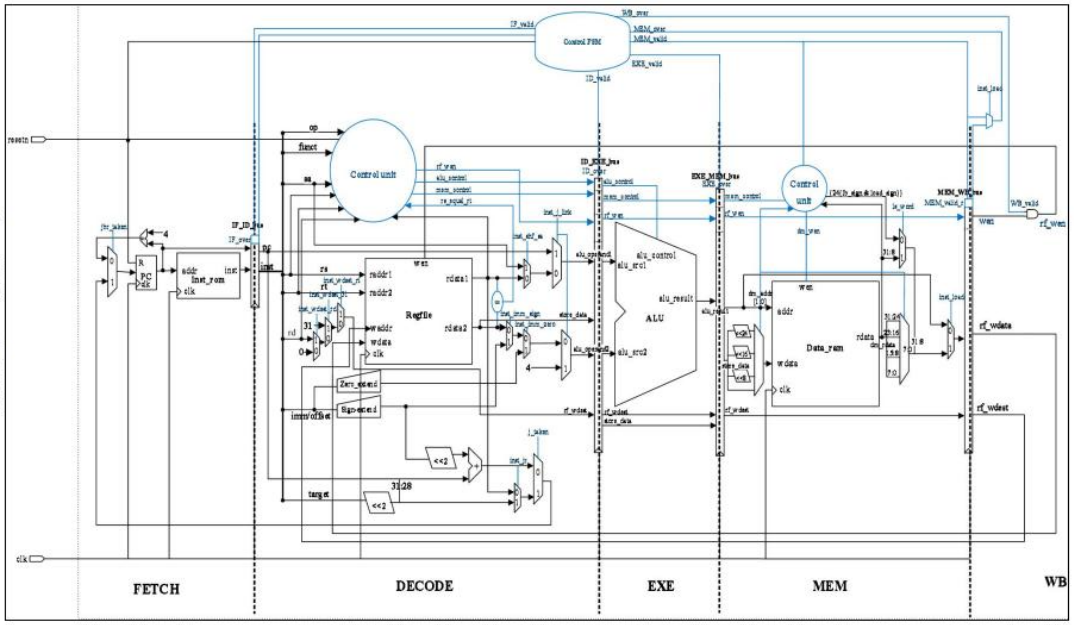


图 8.4 多周期 CPU 的实现框图

四．代码修改

进行仿真实验时，发现指令地址为 14H 后会发生错误，即跳转指令无法实现。分析代码可以发现，在多周期 CPU 中跳转指令需要额外的周期来处理，需要确保在跳转发生时，后续指令的取指不会立即开始，直到跳转逻辑完全处理完毕。因此将 IF 阶段再延迟一拍。

但延迟之后，在 14H 跳转至 54H 后会继续跳转至 96H，出现了连续跳转问题，可能是由于跳转指令未被正确地隔离，因此在此基础上插入一条空指令 NOP，其中空指令的 pc 为 pc+4（当然也可以设为 0）。

使用 insert_nop 标志用于控制是否插入空指令。当检测到跳转时，在更新 PC 到跳转地址同时设置 insert_nop 标志。在下一个时钟周期，如果 insert_nop 为真，则保持 PC 不变并在 IF_ID_bus 中插入空指令（NOP），这样实现了延迟槽的效果。这个空指令周期确保了跳转指令后能够正常执行预期的指令，而不是连续跳转或指令错位。

具体代码修改如下（fetch.v 文件）：

```
`timescale 1ns / 1ps
//*************************************************************************
*******
//     > 文件名: fetch.v
//     > 描述   : 多周期 CPU 的取指模块
//     > 作者   : LOONGSON
//     > 日期   : 2016-04-14
//*************************************************************************
*******
`define STARTADDR 32'd0      // 程序起始地址为 0
```

```
`define NOP 32'd0                  //空指令
module fetch(
    input                   clk,          // 时钟
    input                   resetn,       // 复位信号，低电平有效
    input                   IF_valid,     // 取指级有效信号
    input                   next_fetch,// 取下一条指令，用来锁存 PC 值
    input       [31:0] inst,          // inst_rom 取出的指令
    input       [32:0] jbr_bus,       // 跳转总线
    output      [31:0] inst_addr, // 发往 inst_rom 的取指地址
    output reg          IF_over,      // IF 模块执行完成
    output      [63:0] IF_ID_bus, // IF->ID 总线
    // 展示 PC 和取出的指令
    output      [31:0] IF_pc,
    output      [31:0] IF_inst
);
//-----{程序计数器 PC}begin
    wire [31:0] next_pc;
    wire [31:0] seq_pc;
    reg  [31:0] pc;
    reg  insert_nop;
    // 跳转 pc
    wire        jbr_taken;
    wire [31:0] jbr_target;
    assign {jbr_taken, jbr_target} = jbr_bus; // 跳转总线

    assign seq_pc[31:2]  = pc[31:2] + 1'b1;     // 下一指令地址：PC=PC+4
    assign seq_pc[1 :0]  = pc[1:0];

    // 新指令：若指令跳转，为跳转地址；否则为下一指令
    assign next_pc = jbr_taken ? jbr_target : seq_pc;

    always @(posedge clk)       // PC 程序计数器
    begin
        if (!resetn)
        begin
            pc <= `STARTADDR; // 复位，取程序起始地址
            insert_nop<=0;
        end
        else if (next_fetch)
        begin
            if (insert_nop)
            begin
                pc <= pc; // 保持 PC 不变（插入 NOP 周期）
                insert_nop <= 0; // 清除插入 NOP 标志
```

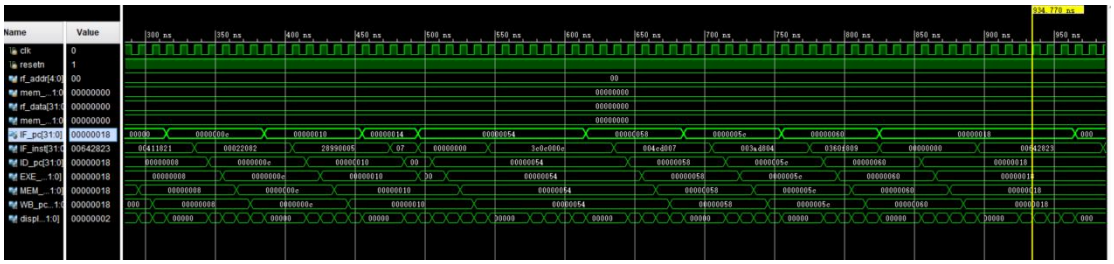
```
                end
                else
                begin
                    pc <= next_pc; //  更新 PC 为下一条指令
                    if (jbr_taken)
                    begin
                        insert_nop <= 1; //  设置插入 NOP 标志
                    end
                end
            end
        end
//-----{程序计数器 PC}end
//-----{发往 inst_rom 的取指地址}begin
        assign inst_addr = pc;
//-----{发往 inst_rom 的取指地址}end
//-----{IF 执行完成}begin
        //  由于指令 rom 为同步读写的,
        //  取数据时，有一拍延时
        //  即发地址的下一拍时钟才能得到对应的指令
        //  故取指模块需要两拍时间
        //  将 IF_valid 锁存一拍即是 IF_over 信号
        reg IF_over_d;    //  延迟的 IF_over 信号
        always @(posedge clk)
        begin
            if (!resetn)
            begin
                IF_over <= 0;
                IF_over_d <= 0;
            end
            else
            begin
                IF_over_d <= IF_valid;
                IF_over <= IF_over_d;    //  延迟一拍
            end
        end
//-----{IF 执行完成}end

//-----{IF->ID 总线}begin
        assign IF_ID_bus =insert_nop ? {pc, `NOP} : {pc, inst};
//-----{IF->ID 总线}end

//-----{展示 IF 模块的 PC 值和指令}begin
        assign IF_pc = pc;
        assign IF_inst = insert_nop ? `NOP : inst;
```

```
//-----{展示 IF 模块的 PC 值和指令}end
```

五．实验结果分析



从图中可以看出在 14H 时正确跳转至 54H，在 60H 时正确跳转至 18H，因此 bug 被正确解决。

六．总结感想

经过本次实验更加加深了对多周期 cpu 的理解以及每条指令的执行过程，更懂得了对跳转指令的处理需要特别考虑流水线中指令的顺序和同步问题，同时在理论的基础上成功实现了延迟槽（即在跳转指令后插入一个空指令），解决了连续跳转的问题。