南京理工大学

NANJING UNIVERSITY OF SCIENCE & TECHNOLOGY

计算机组成原理

实验四报告

班　　级 9191062301

学生姓名 孙傲歆

学　　号 919106840333

任课教师 王晓峰

2021 年 11 月

实验四　单周期 CPU 实验

1.实验目的：

1.理解 MIPS 指令结构，理解 MIPS 指令集中常用指令的功能和编码，学会对这些 指令进行归纳分类。

2.了解熟悉 MIPS 体系的处理器结构，如延迟槽，哈佛结构的概念。

3.熟悉并掌握单周期 CPU 的原理和设计。

4.进一步加强运用 verilog 语言进行电路设计的能力。

5.为后续设计多周期 cpu 的实验打下基础。

2.实验设备：

1.装有 Xilinx Vivado 的计算机一台

2.LS-CPU-EXB-002 教学系统实验箱一套

3.实验内容：

单周期 CPU 就是在 1 个 CPU 周期内执行完一条指令，执行指令所需的微操作由控 制器产生，结构简单，容易设计。指令系统可包括如下指令类型：

运算类指令：算术运算、逻辑运算、移位运算，如 ADD 、SUB、AND、NOT 、SLL、SRL 等等。

传送类指令：寄存器之间的传送、立即数传送，如 MOVZ、MOVN、MFHI 等等。

访存类指令：读存储器指令、写存储器指令，如 LB、LW、SB、SW 等等。

控制类指令：无条件转移、有条件转移，如 J、JR、JAL、BEQ 等等。

本实验要求：根据提供的 32 位 MIPS 框架，至少调试出 5 条指令（运算类、传送类、访存类、 控制转移类等）的运行波形。

4.实验原理及思路：

1）实验原理

设计中所有寄存器和存储器都是异步读同步写的，即读出数据不需要时钟控制，但写入数据需时钟控制。故单周期 CPU 的运作即：在一个时钟周期内，根据 PC 值从指令 ROM 中读出相应的 指令，将指令译码后从寄存器堆中读出需要的操作数，送往 ALU 模块，ALU 模块运算得到结果。如果是 store 指令，则 ALU 运算结果为数据存储的地址，就向数据 RAM 发出写请求，在下一个时钟上升沿真正写入到数据存储器。如果是 load 指令，则 ALU 运算结果为数据存储的地址，根据该值从数据存 RAM 中读出数据，送往寄存器堆，根据目的寄存器发出写请求，在下一个时钟上升沿真正写入到寄存器堆中。如果非 load/store 操作，若有写寄存器堆的操作，则直接将 ALU 运算结果送往寄存器堆，根据目的寄存器发出写请求，在下一个时钟上升沿真正写入到寄存器堆中。如果是分支跳转指令，则是需

要将结果写入到 pc 寄存器中的。下图为 MIPS 三种指令类型格式：

R 类型：

31-26	25-21	20-16	15-11	10-6	5-0
op	rs	rt	rd	sa	func

I 类型：

31-26	25-21	20-16	15-0
op	rs	rt	immediate

J 类型：

31-26	25-0
op	address

2）实验思路

按照要求删改 inst_rom.v 的指令，并保证 R、I、J 类型的指令至少出现一条。其次在 tb.v 加入寄存器以及内存单元的数据读取代码，并且每 10ns（一个周期）读取一次，来观察寄存器以及内存单元的数据变化，以此来验证指令的运行情况。

5. 实验代码：

由于本实验已经给出了具体代码且代码量过大，所以这里仅展示修改部分的代码。

1）inst_rom.v

```
module inst_rom(
 input          [4 :0] addr, //  指令地址
 output reg [31:0] inst          //  指令
 );
 wire [31:0] inst_rom[19:0];
 assign inst_rom[ 0] = 32'h24010004;
 assign inst_rom[ 1] = 32'h00011100;
 assign inst_rom[ 2] = 32'h00411821;
 assign inst_rom[ 3] = 32'h00022082;
 assign inst_rom[ 4] = 32'h00642823;
 assign inst_rom[ 5] = 32'hAC250013;
 assign inst_rom[ 6] = 32'h08000000;
 always @(*)
 begin
      case (addr)
           5'd0 : inst <= inst_rom[0 ];
           5'd1 : inst <= inst_rom[1 ];
           5'd2 : inst <= inst_rom[2 ];
           5'd3 : inst <= inst_rom[3 ];
           5'd4 : inst <= inst_rom[4 ];
           5'd5 : inst <= inst_rom[5 ];
           5'd6 : inst <= inst_rom[6 ];

           default: inst <= 32'd0;
```

```
            endcase
        end
endmodule
```

2）tb.v

```
module tb;
        reg clk;
        reg resetn;
        reg [4:0] rf_addr;
        reg [31:0] mem_addr;
        wire [31:0] rf_data;
        wire [31:0] mem_data;
        wire [31:0] cpu_pc;
        wire [31:0] cpu_inst;
        single_cycle_cpu uut (
                .clk(clk),
                .resetn(resetn),
                .rf_addr(rf_addr),
                .mem_addr(mem_addr),
                .rf_data(rf_data),
                .mem_data(mem_data),
                .cpu_pc(cpu_pc),
                .cpu_inst(cpu_inst)
        );
        initial begin
                // Initialize Inputs
                clk = 0;
                resetn = 0;
                rf_addr = 0;
                mem_addr = 0;

              #100;
             resetn = 1;
             rf_addr=5'h00000001;
             mem_addr=32'h00000014;
              #10;
             resetn = 1;
             rf_addr=5'h00000002;
             mem_addr=32'h00000014;
              #10;
             resetn = 1;
             rf_addr=5'h00000003;
             mem_addr=32'h00000014;
              #10;
```

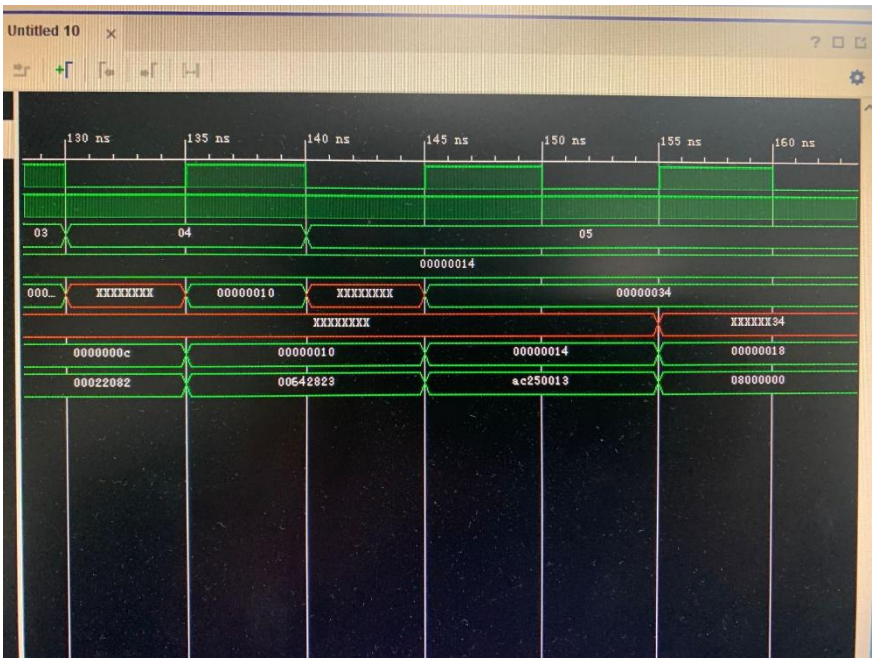
```
        resetn = 1;
        rf_addr=5'h00000004;
        mem_addr=32'h00000014;
         #10;
        resetn = 1;
        rf_addr=5'h00000005;
        mem_addr=32'h00000014;
    end
    always #5 clk=~clk;
endmodule
```

6. 结果分析：

本实验的测试代码为以下 7 条汇编指令（左为具体 MIPS 汇编指令，右为对应的 16 进制机器码）：

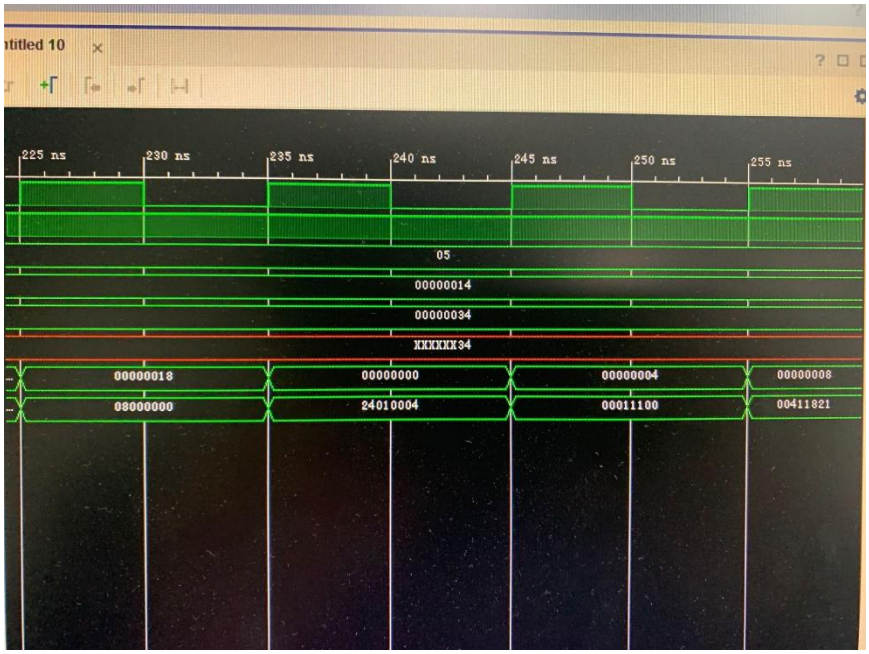
```
addiu $1 ,$0,#4        24010004
sll   $2 ,$1,#4        00011100
addu  $3 ,$2,$1        00411821
srl   $4 ,$2,#2        00022082
subu  $5 ,$3,$4        00642823
sw    $5 ,#19($1)      AC250013
j     00H              08000000
```

进行仿真，获得以下结果：



分析**上图**，我们可以看到，pc（第 7 行数据）自动增量，每次+4，而每个时钟周期都会从指令寄存器取出一条指令（第 8 行数据），这里我们设置 100-110nm 访问 1 号寄存器，110ns-120ns 访问 2 号寄存器，以此类推一直到 5 号为止（第 3 行数据）。而内存我们一直访问 00000014H 内存单元（第 4 行数据）。

可以发现，随着指令的执行，各个寄存器的内容确实在改变且与上述指令保持一致（第 5 行数据）。当指令运行到 sw 指令时，我们发现内存单元的值也发生了变化，从 XXXXXXXX 变成了 XXXXXX34，即将 5 号寄存器的值存入 00000014H 内存单元（第 6 行数据）。



如**上图**所示，指令继续执行，当执行到最后一条跳转指令时，pc 变成了 00H，也就是说我们通过无条件跳转指令成功改变了 pc 的值。此后指令将按照上述步骤循环往复执行下去。

7.心得体会：

由于老师给出了具体代码，且仅让我们进行仿真而没有上板验证，在读懂代码的大概逻辑后，很快便完成了实验。

虽然此次实验很快完成了，但其内容却是四次实验中最难的，前三次实验做的加法器、运算器、存储器实验都是这个单周期 CPU 实验的一小部分。虽然没有进行具体的代码编写工作，但此次实验任然让我受益匪浅。

首先通过本次实验我更加充分地理解了 CPU 内部的工作原理，从取指、译码、执行再到访存、写回，这样一个井然有序的过程。我不禁感叹那些从事 CPU 指令集开发、CPU 硬件开发的人们是多么的聪慧啊！CPU 就如同人的大脑一般控制着机器有条不紊的执行一条条指令，我们现在用的各种电子设备，像是平板电脑、笔记本电脑、PC 机以及智能手机都离不开它。

其次，此次试验，或者说这四次实验为我以后的课程如嵌入式系统、硬件课程设计等等，打下了良好的基础。说实话，上学期的计算机逻辑基础实验虽然也是使用 verilog 实现各种功能，但其远没有计算机组成原理的实验这样具体且有条理。可以看出，四次实验是一个层层嵌套，由浅入深的过程。在这样一个过程中，我更好地理解了计算机内部硬件结构的逻辑以及具体实验方法。

总之，我从计组实验中学到的不仅仅是知识，更是一种学习、探索的方法。