实验五

一、实验名称

简单的类 MIPS 单周期处理器实现-整体调试

二、实验目的

完成单周期的类 MIPS 处理器

三、实验范围

1. ISE 的使用

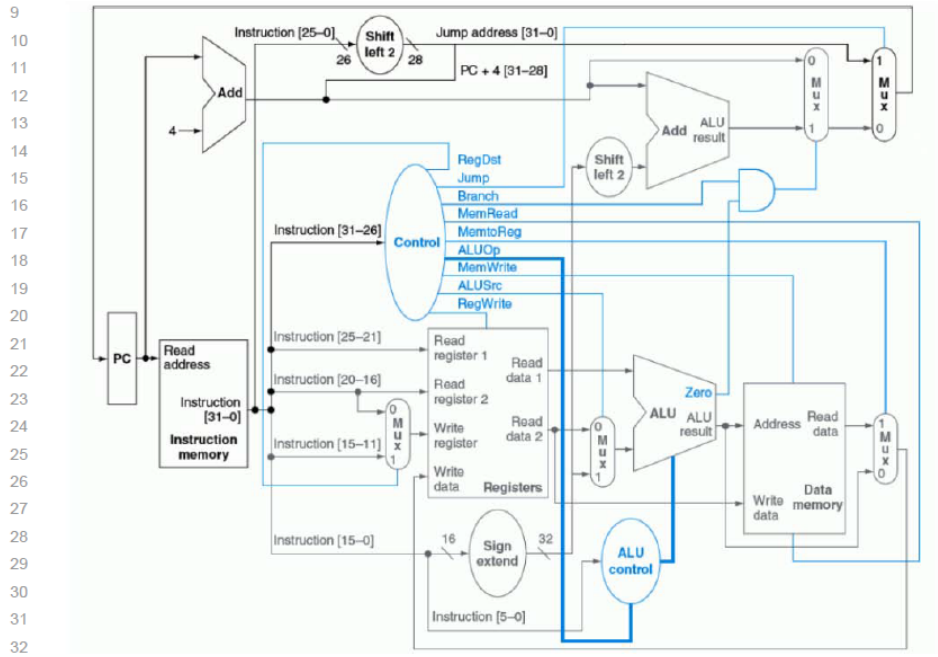
2. Xilinx Spartan 3E 实验板的使用

3. 使用 VerilogHDL 进行逻辑设计

4. 仿真测试、下载验证

四、主要的设计思想和测试仿真

以下为单周期处理器的原图



本次实验将前两次实验写好的一些模块整合起来，再加入之前未完成的元件，最后完成单周期处理器的实现。

我没有再写其他模块，其余元件的实现全部在 top 模块中实现。

1. 取指令

取指令只需要取指地址 PC 即可，由于我是 32 位读取，而 PC 的变化是每次加 4，所以需要 PC 右移 2 位。所以取指令时写成这样：

always @(PC)

INST = IM[PC >> 2];

（但是不知为何，在上板子的时候，这样写始终无法上板成功，下面会做一些变化）。

2. 中间逻辑单元

主要参照实验指导书的写法

定义 wire 类型的变量，利用 assign 语句进行连线。

比如多选器的实现

```
wire [31:0] INPUT2;

assign INPUT2 = ALUSRC?DATA:READDATA2;
```

移位操作

```
wire [31:0] addshift;

assign addshift = DATA << 2;
```

加法器

```
wire [31:0] ADDRES;

assign ADDRES = pcPlus + addshift;
```

按照原理图逐步实现即可，在这里就不一一列举了。

3. reset 信号的处理

为了避免多个 always 块导致赋值时的混乱。在时钟上升沿来临时进行 reset 的判断，若 reset==1，则 PC 置 0，否则 PC = nextPC

4. 模块实例化，连接模块

实例化前两次实验中编写的模块，实例化的过程中连接模块的端口。

我采用实验指导书上推荐的实例化方法， 即在连接时用“.”符号，表明原模块是定义时规定的端口名：

模块 模块名（.端口 1 名（信号 1）， .端口 2 名（信号 2））

以 data_memory 为例，关键代码如下：

```
data_memory mainmemory(
                    .clock_in(clk),
                    .address(ALURES),
                    .writeData(READDATA2),
                    .memWrite(MEM_WRITE),
                    .memRead(MEM_READ),
                    .readData(READDATA));
```

以上便是主要的设计方法。

仿真测试：

在编写完成实验五后，但花在测试上同样需要很多的时间，需要自己去写一些指令来验证是否实现无误。

通过$readmemb 来读取文件中的二进制指令，

```
10101100011000100000000000000011    //sw $2, 3($3)
00001000000000000000000000000011    //jump
10101100011001100000000000000011    //sw $6, 3($3)（跳过该指令）
10001100000000010000000000001010    //lw $1, 10($0)
10001100000001000000000000001000    //lw $4, 8($0)
```

00000000001000100001100000100000　　//add $3，$1, $2

00010000001000110000000000000001　　//beq $1, $3, 1

10101100011001000000000000000011　　//sw $4, 3($3) 这条指令将被跳过

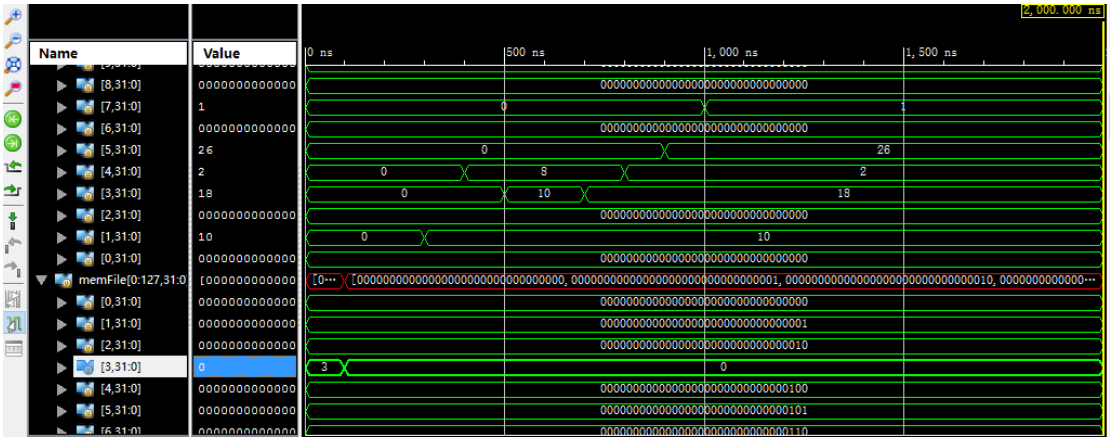
00000000001001000001100000100000　　//add $3, $1, $4

00000000001000110010000000100100　　//and $4, $1, $3

00000000011000010010100000100101　　//or $5, $3, $1

00000000100000010011100000101010　　//slt $7, $4, $1

仿真后的波形为（只显示相关寄存器和存储器）



五、实验上板验证

我上板的思想比较简单，就是显示 PC 和寄存器，通过两个开关 00,01,10,11 四种情况来选择 PC 和 3 个寄存器的显示，通过 LED 灯的闪烁观察变化。

上板时主要要做的改动如下：

1. 将寄存器，存储器，指令由文件读取的方式改成在程序中初始化。

2. 加入分时模块，本身的时钟频率过大，无法进行观察。分频主要参照实验二中的形式编写：

```
module timekiller(clockIn, clockOut);
    input clockIn;
    output clockOut;
    reg clockOut;

    reg [23:0] buffer;

    always @(posedge clockIn)
    begin
        buffer <= buffer + 1;
        clockOut <= &buffer;
    end
endmodule
```

3. 在 register 模块中添加一个输入：PC 和两个输出：switch，led

```
        //开关控制 led 灯显示寄存器或 PC 的值
        case(switch)
        0: led[7:0] = PC[7:0];
        1: led[7:0] = regFile[1][7:0];
        2: led[7:0] = regFile[2][7:0];
        3: led[7:0] = regFile[3][7:0];
```

同时在 top 模块中需要连线

.switch(switch),

.PC(PC),

.led(led));

最后能实现 PC 和寄存器值的显示，同时可以 push 一个 button 实现清零操作。

但是，因为我在一个时间内只能观察一个寄存器的变化，而一条指令是对多个寄存器的操作，而我无法同时看出一套指令过后每个寄存器的变化。这是我这个上板的缺点所在。

六、遇到的问题和心得体会

这次实验五可以说是我花时间最多的一个实验，几乎使用了两周的时间才算是搞定。

最初在仿真模拟时，一开始用 load word 指令来测试，总是出错，前前后后改了很久始终一筹莫展。然后再用 store word 进行尝试是可以的，add 操作又出现错误，于是怀疑是写回寄存器时出现了错误，结果果然在连线时变量名写错了，改过来以后就可以了。

然而过了几天上课时准备给老师验证时，突然 clk， reset 又变成了 x，于是一上午又在调试，后来才知道是我在读入指令时为了看起来方便，在二进制代码后面添加了注释从而导致读文件时有可能会出错，删除注释即可。在这里又消耗了大量的时间。

搞定了仿真以后，上板同样困难重重，由于时间不够了，我就拿

电子技术实验课上使用的小板子在寝室里进行实验。由于小板子存储量不够，空间不能开太大，所以要减小空间的申请，但是就在我认为已经成功的时候，寄存器的显示始终都没有发生变化，又陷入了一筹莫展，最后问其他的同学有可能是指令读取的问题，使用 8 位读取，4 个字节构成一条指令便可以成功显示寄存器的变化。

总而言之，这次实验五遇到了许多困难，也花了许多的时间，有好几次都怀疑是不是 ISE 这个软件有问题，但其实都是自己在一些细节上不注意导致的错误。不过最终总算是做出来效果，还是觉得很有收获。