计算机组成原理课程设计实验三

多周期 CPU 基于硬件开发板的测试

学院： 计算机学院

班级： 07111503 07111506

组员： CMQ

DWY

HSF

LW

WZW

WDT

目录

多周期CPU基于硬件开发板的测试

一、 实验目的

1、了解现代计算机硬件设计的基本流程和方法。

2、了解典型的RISC 处理器MIPS 的体系结构。

3、了解汇编语言到机器语言到计算机执行软硬的逻辑关系。

4、掌握处理器的设计原理和方法。

5、培养寄存器级硬件故障的检错和排错能力。

二、 实验分工

2.1 分工

王紫薇/陈牧乔：CPU输入输出设计

王铎暾/侯思凡：数码管&LED设计与控制

杜文仪/罗　薇：仿真调试与下板测试

2.2 组长联系方式

陈牧乔：15201692532

三、 实验环境

硬件环境：Xilinx Artix-7系列开发板 xc7a35tcsg324-1

软件环境：Vivado v2017.2 (64-bit)

系统环境：Windows 10

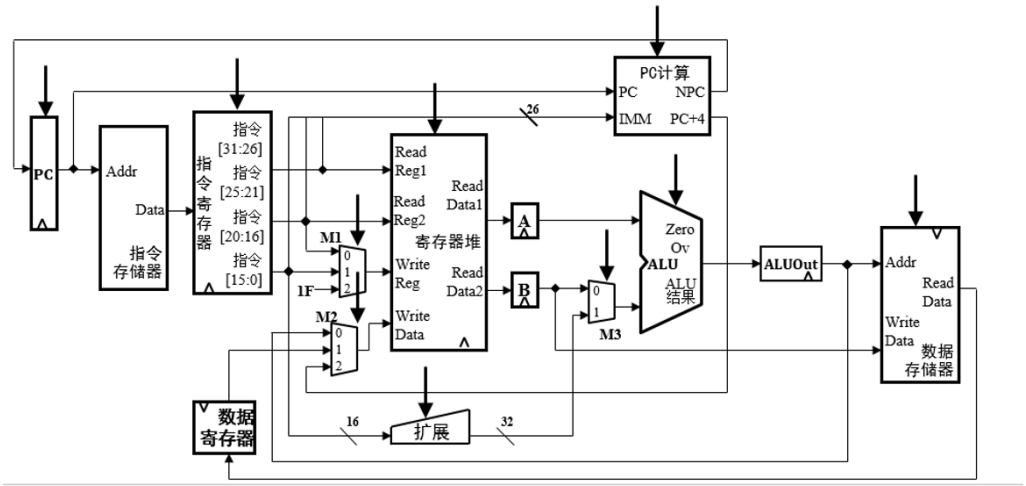
四、 实验原理

4.1 多周期 CPU

多周期 CPU 区别于单周期处理器的设计，指令的执行完成不是耗费整个周期，而是被分成多个阶段，每个阶段在一个时钟周期内完成，使得整个时钟周期降低到最复杂的阶段所花时间的 1/4 至 1/3(取决于所分阶段的数量)。利用使能信号对每个阶段的访存和读写进行一个限制，每一步的结果都能在下一个时钟周期保存到相应的单元（主要是寄存器相应位置中），对阶段的切分主要利用加入的寄存器来实现，但同时也需要考虑整个状态的转换。

多周期 CPU 的数据通路如下：

数据通路：



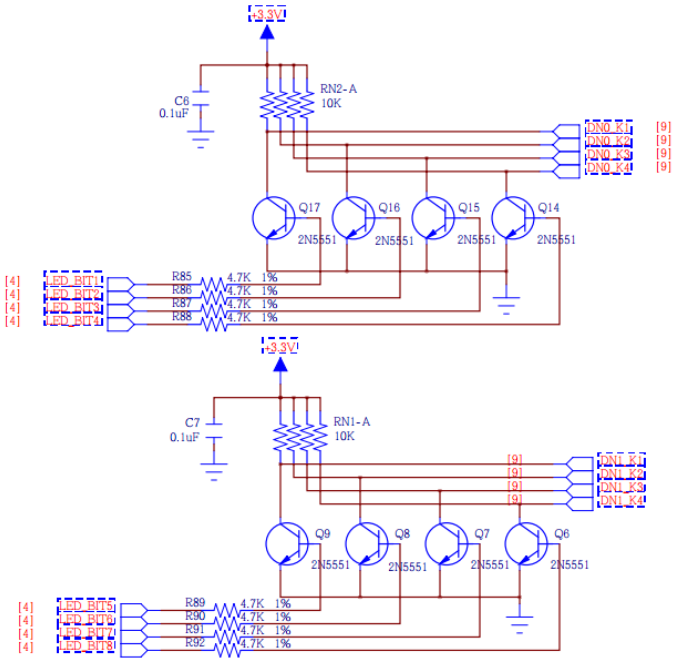
4.2 七段数码管显示原理

在 EGO1 用户手册中，有对数码管的管脚使用的描述。在该开发板中，数码管为共阴极数码管，即公共极输入低电平。共阴极由三极管驱动，FPGA 需要提供正向信号。同时段选端连接高电平，数码管上的对应位置才可以被点亮。因此，FPGA 输出有效的片选信号和段选信号都应该是高电平。

用户手中数码管的管脚配置如下：

名称	原理图标号	FPGA IO PIN
A0	LED0_CA	B4
B0	LED0_CB	A4
C0	LED0_CC	A3
D0	LED0_CD	B1
E0	LED0_CE	A1
F0	LED0_CF	B3
G0	LED0_CG	B2
DP0	LED0_DP	D5
A1	LED1_CA	D4
B1	LED1_CB	E3
C1	LED1_CC	D3
D1	LED1_CD	F4
E1	LED1_CE	F3
F1	LED1_CF	E2
G1	LED1_CG	D2
DP1	LED1_DP	H2
DN0_K1	LED_BIT1	G2
DN0_K2	LED_BIT2	C2
DN0_K3	LED_BIT3	C1
DN0_K4	LED_BIT4	H1
DN1_K1	LED_BIT5	G1
DN1_K2	LED_BIT6	F1
DN1_K3	LED_BIT7	E1
DN1_K4	LED_BIT8	G6

在开发板资料 EES-338-V0.1-out 中，片选信号的电路如下：



由管脚配置可以看出，LED_BIT 为片选信号，被选择的段接高电平即写 1，其余写 0，其对应的数码管被点亮。而 LED0 与 LED1 为段选信号，分别控制数码管的前 4 个与后 4 个的内容。

项目中数码管和 LED 灯配合完成多周期 CPU 数据存储的输出功能，所要显

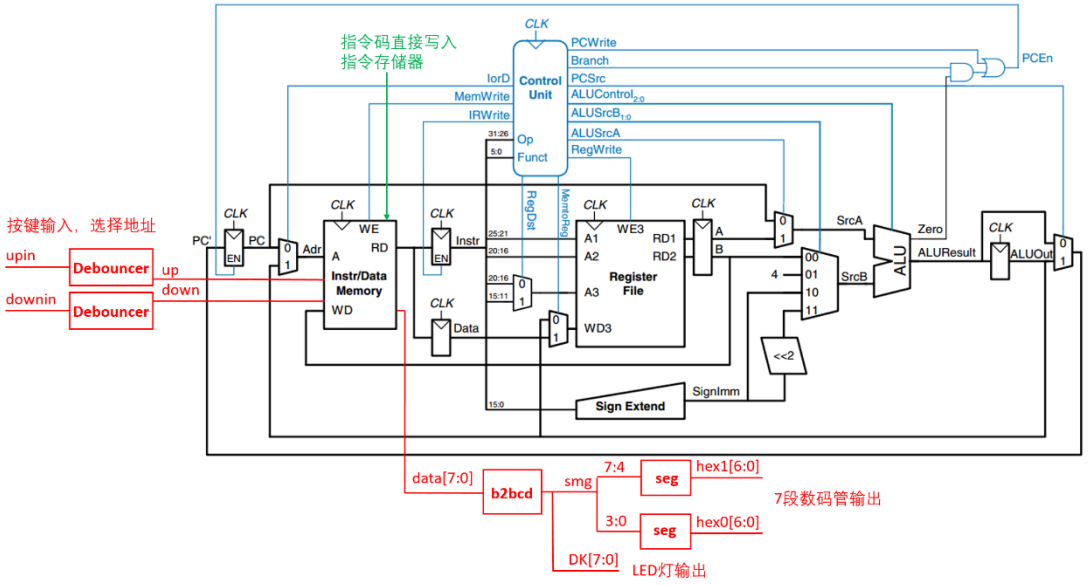
示的数字范围为 0 至 99（十进制）。我们令前四位数码管显示数字的十位（四个七段数码管显示相同的十进制数值），后四位数码管显示数字的个位（同上），使用了段选信号对数码管的显示进行控制。

五、 实验设计

5.1 实验数据通路图

本实验在实验二多周期 CPU 的基础上进行改进，结合开发板的特性，添加输入输出控制，使实验二中所得仿真成果得以下载到开发板，完成多周期 CPU 实验的板上验证工作。

在多周期 CPU 的数据通路上进行增改，所得到的数据通路图如下：其中绿色和红色部分为本实验所实现的内容。



5.2 输入输出信号一览表

	信号	配置设备管脚	功能说明

输入信号	Rst	1 个按键	作为系统清零，重置输入
	寄存器选择信号	上下两个按键	用于选择数据存储器的地址
输出信号	LED[7:0]	8 个 LED 灯	二进制显示指定地址数据存储器内容的后 8 位
	Seg	七段数码管	和 LED 灯同步显示数字，可显示两位十进制数

5.3 指令码写入

在指令存储器部件 im 中，对前 24 个存储器进行 assign 赋值，将 fibonacci 的指令码直接写入指令存储器。

```
assign   im[0] = 32'h20080000;
assign   im[1] = 32'h200d0054;
assign   im[2] = 32'h8dad0000;
assign   im[3] = 32'h240a0001;
assign   im[4] = 32'had0a0000;
assign   im[5] = 32'had0a0004;
assign   im[6] = 32'h340e0002;
assign   im[7] = 32'h01ae4822;
assign   im[8] = 32'h340f0001;
assign   im[9] = 32'h012f602a;
assign   im[10] = 32'h118f0007;
assign   im[11] = 32'h8d040000;
assign   im[12] = 32'h8d050004;
assign   im[13] = 32'h0c000c16;
assign   im[14] = 32'had020008;
assign   im[15] = 32'h21080004;
assign   im[16] = 32'h2129ffff;
assign   im[17] = 32'h08000c09;
assign   im[18] = 32'h3c0eabcd;
assign   im[19] = 32'h21080008;
assign   im[20] = 32'had0e0000;
assign   im[21] = 32'h08000c15;
assign   im[22] = 32'h00851020;
assign   im[23] = 32'h03e00008;
```

5.4 按键输入

在数据通路 mips 中，添加 upin 和 downin 信号输入，分别对应开发板的上下两个按键的信号。

```
input upin;
input downin;
```

由于检测的是按下按键时的边沿信号，因此要添加去抖动部件，对输入信号 upin 和 downin 进行处理，输出去抖动的边沿信号 up 和 down。输出信号在检测到按键按下后的一个时钟周期内为 1，其余时间为 0。

```
Debouncer Up(clk, rst_n,upin,up);
Debouncer Down(clk, rst_n,downin,down);
```

up 和 down 信号输入到数据存储器 DM 部件中，用于选择输出数据的地址。

5.5 数据选取

在数据存储模块中，需要根据输入的 up 和 down 信号选择存储器的地址，并将存储器的数据输出。通过语句 i<=i+up-down 实现输入信号到数据存储器地址的转换，并通过语句 data<=dm[i]将指定存储器的数据从 DM 模块输出。具体的实现方法如下：

```
always@(posedge clk or negedge rst)
begin
    if(rst)
    begin
        i<=0;
        data<=dm[i];
        dm[21]<=32'd20;
    end
    else
    begin
        data<=dm[i];
        i<=i+up-down;
        dm[21]<=32'd20;
    end
end
```

5.6 数据输出

对于 led 灯的输出，直接将 data 的低 8 位赋给 led 即可：

```
//led
assign led=data[7:0];
```

对于数码管的输出，由于 data 是二进制表示，首先要通过 b2bcd 模块将 data 的低 8 位转换为 bcd 码：

```
b2bcd b2bcd(data[7:0],smg[7:0]);
```

输出为 smg[7:0]，其中高 4 位为十位的 bcd 码，低 4 位为个位的 bcd 码。接下来将十位和个位的 bcd 码分别作为两个 seg 模块输入，转化为七段数码管的数字显示信号。

```
seg seg1(rst_n,smg[7:4],hex1);
seg seg0(rst_n,smg[3:0],hex0);
```

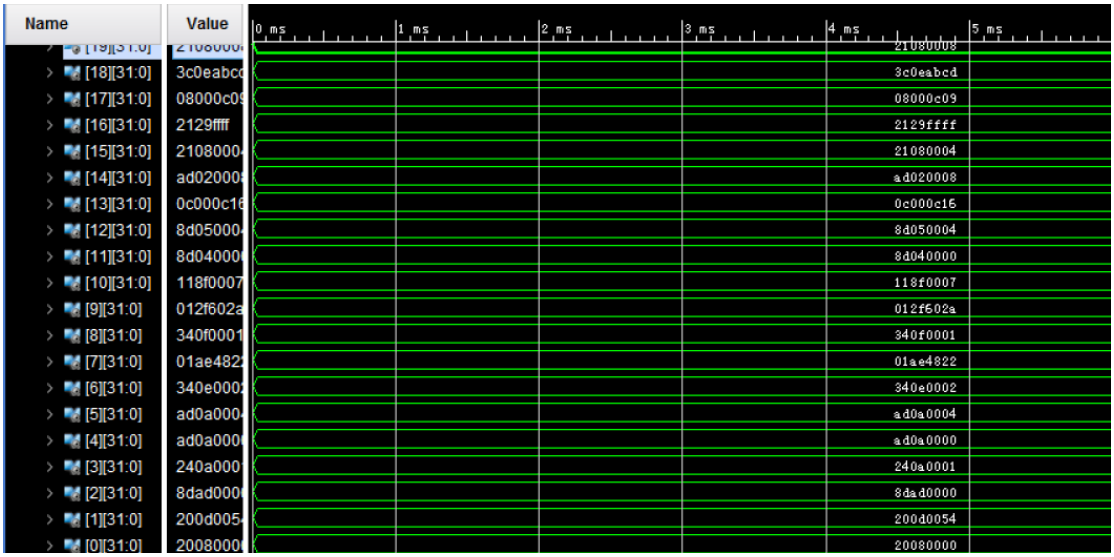
七段数码管的十进制数字表示如下：

```
case (data)
0    :    hex = 7'b011_1111;        // digit 0
1    :    hex = 7'b000_0110;        // digit 1
2    :    hex = 7'b101_1011;        // digit 2
3    :    hex = 7'b100_1111;        // digit 3
4    :    hex = 7'b110_0110;        // digit 4
5    :    hex = 7'b110_1101;        // digit 5
6    :    hex = 7'b111_1101;        // digit 6
7    :    hex = 7'b000_0111;        // digit 7
8    :    hex = 7'b111_1111;        // digit 8
9    :    hex = 7'b110_1111;        // digit 9
```

将转换的结果作为 hex1 和 hex0 输出，就可以实现数字在数码管的显示。

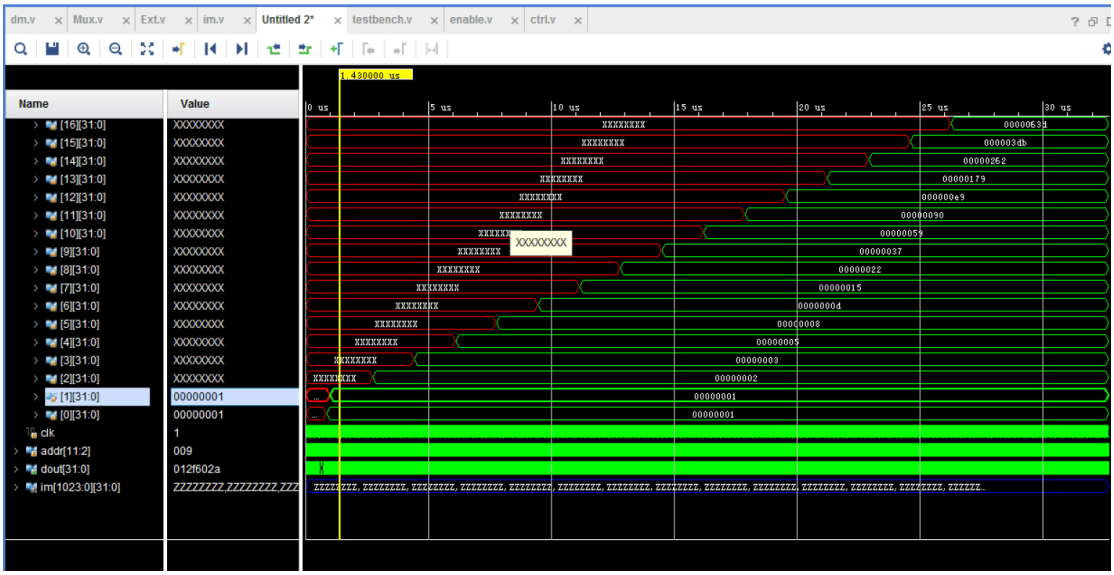
六、 实验仿真

6.1 指令码成功写入 im

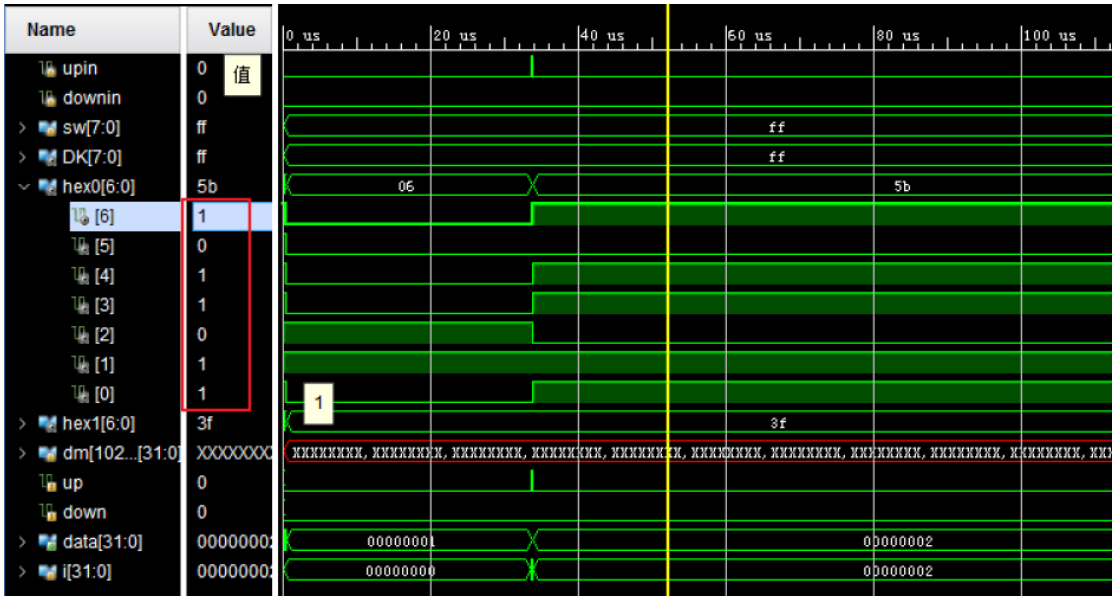


6.2 U_IM 模块成功读取指令码

6.3 斐波那契数列暂存在 dm 中

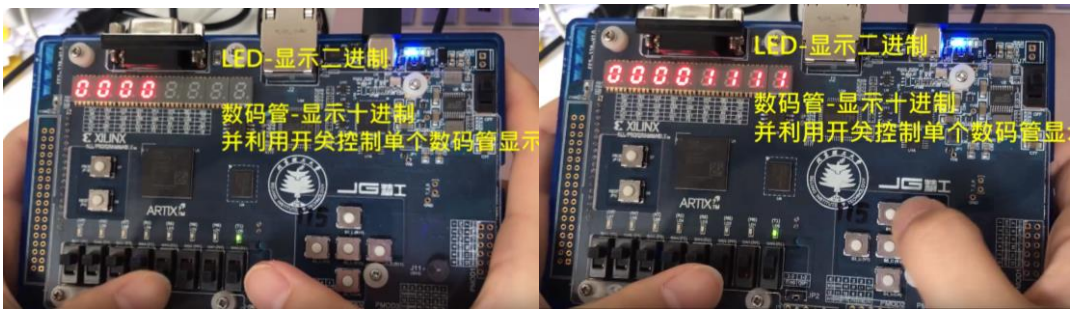


6.4　以取数“2”为例检查数码管编码



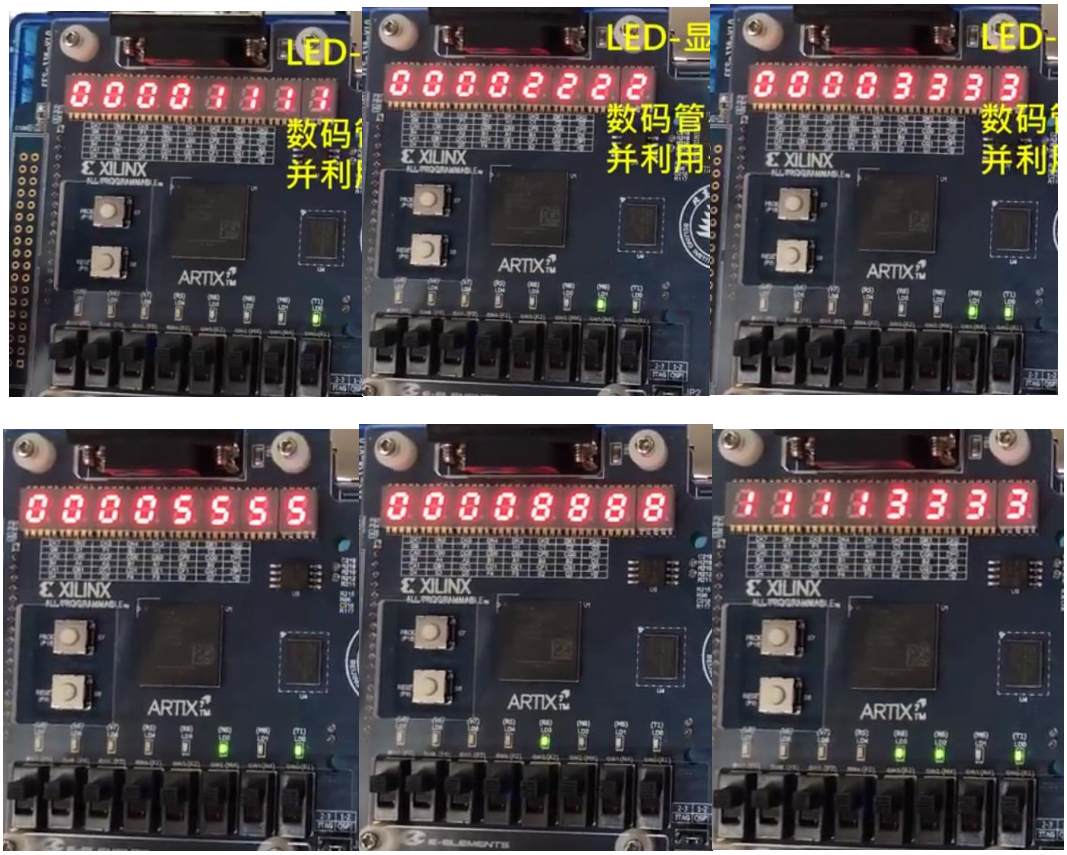
七、 实验结果

7.1 I/O 控制



通过 switch 开关控制数码管的亮暗——如左图，只有 sw[7:4]开关打开，故前四个数码管亮，后四个数码管暗；如右图，八个开关全都打开，八个数码管均亮。

7.2 数码管、LED 同步显示斐波那契数列



如上六张图，斐波那契数列前六个数在数码管上以十进制方式显示，LED 同步以二进制显示。

该显示由 up/down 控制选数。

八、 遇到的问题

8.1 指令码的输入

起初，我们仿照 testbench 的方法，按层次访问 im 模块，预备将指令码由 txt 直接写入 im 中，但是 vivado 不允许一个模块直接访问另一个模块的内部，并且 txt 内容无法直接写入开发板。然后，我们设计将指令写成 coe 文件，并且生成 rom，通过仿真的确实现了指令的读取，但是可能由于一些读写周期的问题，

指令执行错乱。最后，只得强制用 assign 语句，将指令码序列直接预存在 im 中。

8.2 结果的取出

按照指令执行的顺序，按顺序写入 dm 的内容就是斐波那契数列，但是开发板时钟太快，人眼无法跟上它的节奏。所以，我们决定通过手动按键访问数列，通过 up/down 键查看数列内容，所以让 dm 保持暂存数列，通多按键访问数组指定位置上的内容。

最开始的仿真发现计算永无止境的继续——当算满 1024 个数之后，计算仍在继续，并且循环着覆盖 dm 内容，所以原本算好暂存在 dm 的数在几纳秒之后便换成另一个数，后来我们发现是初始化不完全，初始化内容 应当包括强制给 dm[21]赋值为 20，命令指令只计算前 20 个斐波那契数。

8.3 数码管显示混乱

电路图看错了，编码 01 反了，仔细分析了一下当前亮了哪根管息着哪根管就发现了。

九、 实验心得

9.1 CMQ

实验三在实验二的基础上添加硬件开发板的输入输出，实现多周期 CPU 的开发板验证过程。这个实验对于 CPU 的设计没有更高的要求，而是需要灵活运用所掌握的硬件编程与调试知识。

实验三与实验二有很大的不同，实验二的前期工作比较艰难，需要充分理解多周期 CPU 的原理，而一旦理解并确定了方向，编码与仿真实现并不困难。而实验三在设计之初有很明确的目标，实现起来却有各种各样难以解决的 debug 问题，即使仿真成功，最后下板也不一定成功。这就是硬件编程的不确

定性，十分依赖具体的平台和硬件设备的物理条件。同时，verilog硬件描述语言与以C语言为代表的一系列编程语言在思维模式上有很大的不同，它的组合逻辑和时序逻辑更加贴近机器的运作方式，这也给我们编写代码的过程带来了很大的困扰。

其实整个debug过程就是一个逐步退让的过程，最开始模块化更强的方案没能跑出来（也没有报错），又一时找不出原因，只好破坏它的模块性，逐步深入模块内部，用更简单粗暴的方式实现预期的功能。虽然功能最后实现了，但是在代码的耦合性、可移植性上还是有一定的提升空间的。这需要更加深入的学习和更丰富的硬件实践经验。如果有机会希望在未来的某天能够解决这个历史遗留问题。

9.2 DWY

实验三在实验二的基础上又要自己发挥，一开始我们也有很多设想，但是在不断的讨论过程中又修改了很多次，最后完成的这份成果也让我很满意了，在和小组成员的沟通交流中也获得了许多团队合作的经验，再一次体会到了合作的力量比个人要大许多。收获很大，希望以后还有机会能体验合作带来的乐趣。

9.3 HSF

实验三是以实验二为基础的，即把指令码序列给CPU，让CPU计算出结果，然后通过开发板器件将结果显示出来。原本我们是仿照testbench的方法，将指令写给im，然后深入到dm中把结果取出。结果vivado的实现和modelsim的仿真在很多限制上不一样，不允许深入到模块内部写入或取出内容。然后我们考虑将各个参数暴露出来，也就是将我们需要的在接口中传递。由于dm太大，同样不允许暴露。最后我们只能修改CPU内部，将指令序列强制写进im，需要查看的结果也通过dm传递给顶层，驱动开发板的数码管、LED显示。

整个过程曲折还是蛮多的，也是通过一步步折中最后决定了这个办法。不

过从解决问题的过程中，我们也充分体会到了模块化设计的思想，模块应当是封装好的黑盒，不被外界知道其内部内容，只通过模块间的通讯传递一些必要的参数。也是从实践中明白模块间高内聚低耦合的重要性和实现方法。虽然这个特性阻碍了我们实验的进行，不过不得不承认这对保护各个模块是很重要的。

9.4 LW

本模块要求为自己设计的CPU设计一个有趣的应用，在开发板上验证并实现。本次实验相当于是我们设计的多周期处理器的实际应用，让我对于处理器设计以及实现有了更清晰的了解，对于verilog语言的写法和使用更加熟悉。

9.5 WZW

模块三运用多周期CPU的框架，运行斐波那契示例并将结果呈现在开发板。这既是组成原理模块二的应用，也是汇编程序设计实验的应用。我们计划读取TXT形式指令码，将斐波那契数列结果暂存后输出大数码管和LED，但却遇到了读取的问题。后来尝试查到的解决办法做coe读取，却又遇到了时钟频率的问题，最终我们将指令直接写入im，

有了上学期硬件课程和之前“纯原创“的汇编I/O控制实验的经历之后，将结果显示到开发板变得轻松起来。我们方向键up&down用来控制结果的上下翻动。由于开发板数码管电路的局限性，我们只能显示到1~89，于是加了边界限制，超过限制不予显示。我们还将结果以二进制的形式显示在LED，使得实验更加完整。

整个过程在结果读取上遇到了很大的问题，进度一度受阻，仿真结果正确但是无论如何也没有结果。加上下板运行时间很长，临近截止日期等原因，大家不知所措更加焦头烂额。但这也让之后的成功更加有意义，甚至让我们有更大的动力加LED显示、边界控制等功能来完善实验。这次实验从最初的MIPS汇编到最后在多周期框架上运行下板，再到录制视频、汇报验收完成，整个小学期也画上了句号，让我受益匪浅。

9.6 WDT

经过计算机组成原理最后一个实践的完成，我也彻底完成了整个小学期的实践过程，能够清晰地看到整个小组和个人在整个实践过程中的学习和进步过程，到了这一次的下板验证，我们采用的是将运行多周期测试的斐波拉契的结果下板到数码管和led灯中，并能够控制显示不同的结果。我记得我当时在负责将多周期的dm结果进行输出，就遇到了瓶颈。因为多周期的设计是基于读取code.txt的机器码，与后续的交互设计是困难的。所以后面我们就将机器码直接放在im读取，然后结果直接读取dm进行保存，取得突破性进展。在这次实践中也是前面实践的一个充分运用，包括Verilog语言，汇编的下板，以及之前多周期的指令设计内容。如果之前多周期的指令设计有问题，在这一个模块的下板也将出现问题，因为我们是实践斐波拉契数列结果的下板，如果当时多周期设计有一定问题，则对结果的导出将遇到问题。所以我也充分认识到了实践的连贯性，和条理性对整个团队项目的重要性。而且我也进行了整个项目的视频录制和ppt制作的工作，也感受到一个项目要被充分认可，需要一个清晰的，能够充分表达团队想法的成果展示和demo。整个小学期的实践和设计让我受益匪浅，希望以后也能继续完成一些团队项目，加强自己！

十、 参考文献

[1] MIPS® Architecture For Programmers Volume II-A: The MIPS32® Instruction Set

[2] Digital Design and Computer Architecture

[3] d1MIPS指令集合和汇编数据通路ppt.ppt

[4] d2单周期控制和多周期处理器设计.ppt

[5] L自学2-ModelSim（自学）.pptx

[6] EES-338_UserManual_v1.0.pdf

[7] EES-338-V0.1-out.pdf