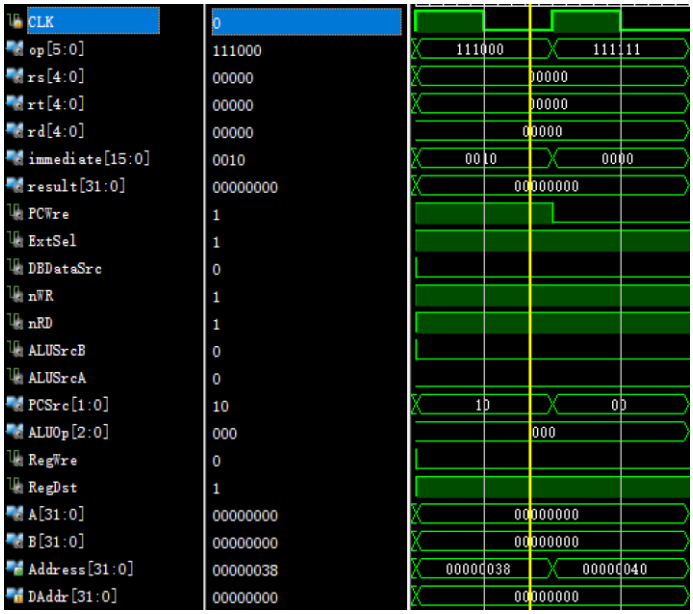


PCSrc为10，对PC执行为{(pc+4)[31:28],addr[27:2],2{0}}（即0x00000040），在CLK的下个上升沿到来时Address变为00000040，跳过了0000003C，执行half指令，PCwre变为0，停机。

3、实现：

数码管显示格式：左边两位数码管 BB：右边两位数码管 BB。

SW_in = 00:显示 当前 PC 值:下条指令 PC 值

SW_in = 01:显示 RS 寄存器地址:RS 寄存器数据

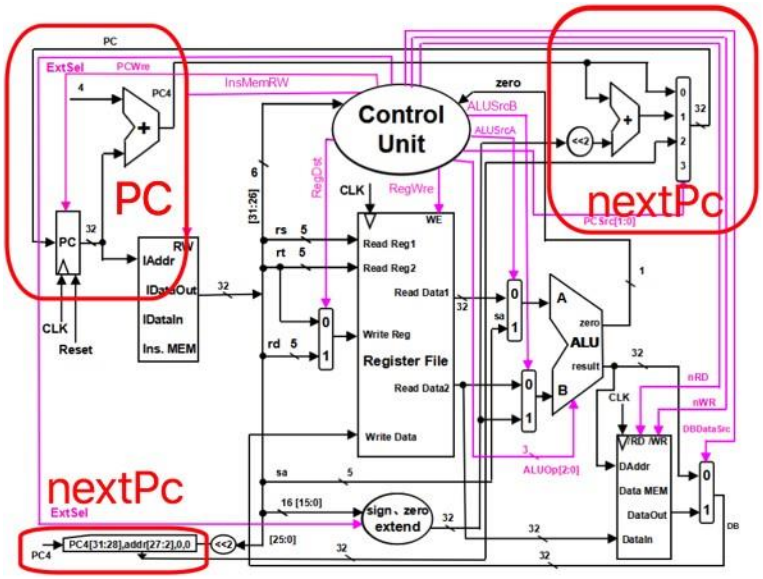
SW_in = 10:显示 RT 寄存器地址:RT 寄存器数据

SW_in = 11:显示 ALU 结果输出 :DB 总线数据

由于前面设计的 CPU 顶层模块 I/O ports 较多，需要根据实验板上的接口再设计一个更顶层的模块。且为了输出上述内容，需要将其添加到原来的顶层模块输出接口。

为了输出 nextPC，需要对前面设计的 PC 模块进行拆分，拆分为 PC 模块和 nextPc 模块。

```
module SingleCycleCPU(
      input CLK,
      input Reset,
      output [31:0] PCAddress,
      output [31:0] nextPC,
      output [4:0] rs,
      output [4:0] rt,
      output [31:0] RegisterFileReadData1,
      output [31:0] RegisterFileReadData2,
      output [31:0] ALUResult,
      output [31:0] DataMemoryDataOut
   );
```



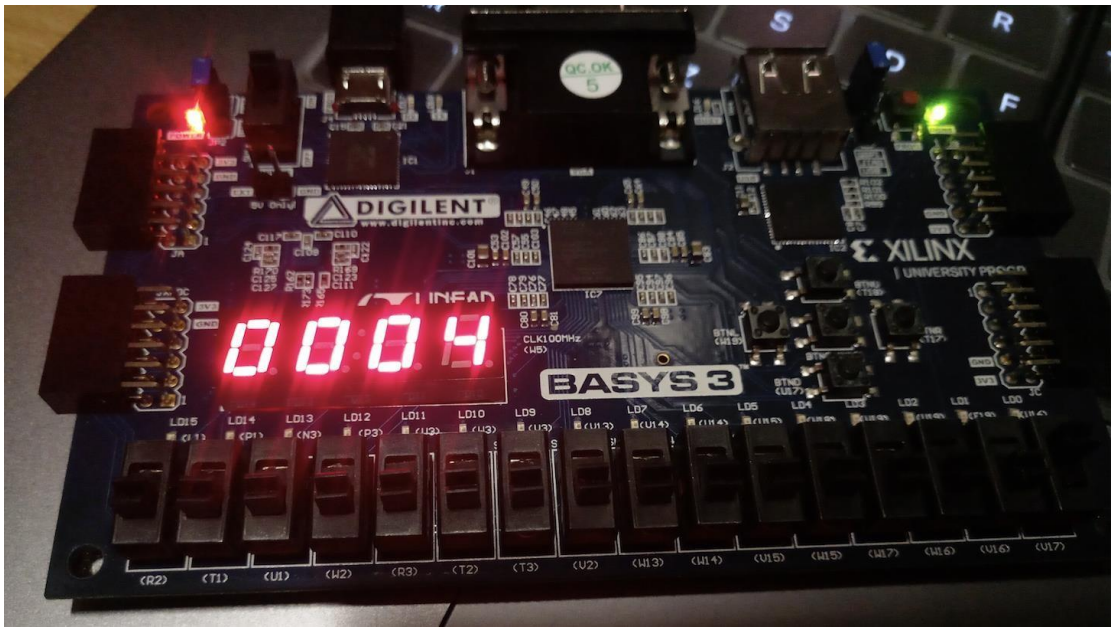

```
module PC(
       input CLK,
       input Reset,
       input [31:0] nextPC,
       input PCWre,
       output reg [31:0] Address
    );
module nextPc(
       input [1:0] PCSrc,
       input [31:0] Address,
       input [31:0] immediate,
       input [25:0] address,
       output reg [31:0] nextPC
    );
```

此外，还需要添加时钟分频、按键触发、开关控制、数码管显示模块，最终最顶层的模块:

```
Clock_div clock_div(W5,CLK);

    Button button(CLK,button,trigger);

SingleCycleCPU

singleCycleCPU(trigger,Reset,PCAddress,nextPC,rs,rt,RegisterFileReadData1,Re

gisterFileReadData2,ALUResult,DataMemoryDataOut);     Src

src(src,PCAddress[7:0],nextPC[7:0],rs,rt,RegisterFileReadData1[7:0],Register

FileReadData2[7:0],ALUResult[7:0],DataMemoryDataOut[7:0],data);

    Display display(CLK,Reset,data,seg,an,dp);
```

再根据接口修改一下仿真代码，观察波形确定无误后，进行综合、实现和烧写。

板上 Reset(右侧 V17)置 1，SW_in(左侧 R2,T1)开始测试。



列举其中几条指令：

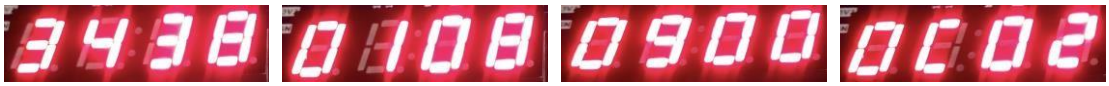
地址	汇编程序	指令代码					
		op（6）	**rs(5)**	**rt(5)**	**rd(5)/immediate (16)**	**16** 进制数代码	
0x00000000	addi $1,$0,8	000001	00000	00001	0000 0000 0000 1000	=	04010008

当前 PC:下条指令 PC　RS 地址:RS 寄存器数据　RT 地址:RT 寄存器数据　ALU 结果:DB 数据

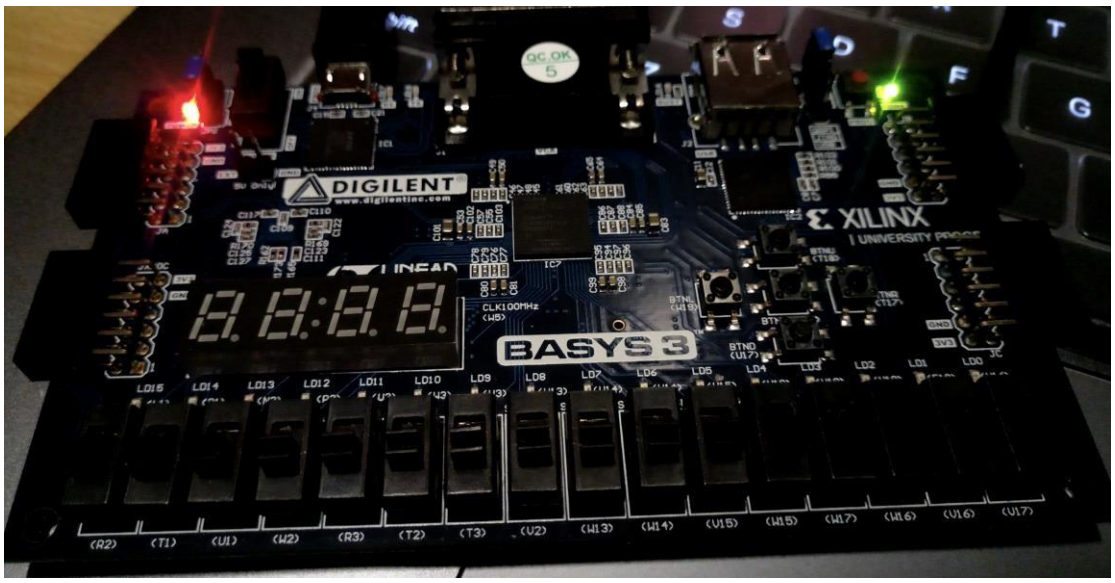
0x00000004	ori $2,$0,2	010000	00000	00010	0000 0000 0000 0010	=	40020002

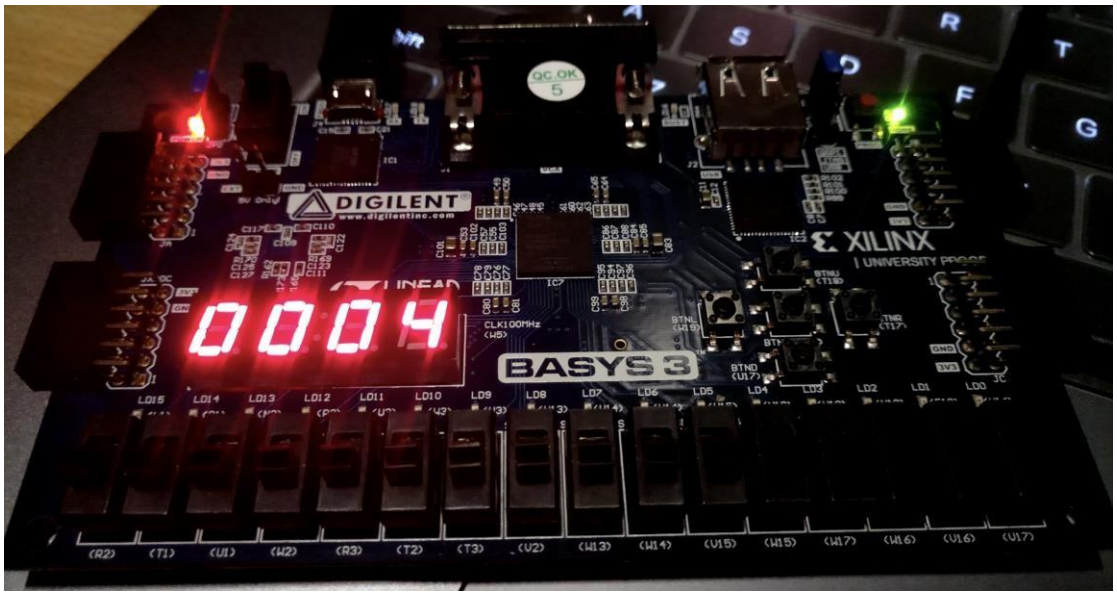
0x0000002C	beq $7,$1,-2 (=,转 28)	110000	00111	00001	1111 1111 1111 1110	=	C0E1FFFE

0x00000034	lw $9,4($1)	100111	00001	01001	0000 0000 0000 0100	=	9C290004



执行到 half 指令后，要重新从头执行，可以 Reset 置 0 后再置 1。





再次回到开始的状态。

一. 实验心得

通过对单周期CPU的设计，不仅加深了对CPU结构、数据通路的学习和理解，也初步学习了使用Verilog HDL语言进行编程设计，也熟悉了Vivado的使用。

1、wire和reg的区别：

i. 仿真时使用软件思路，Verilog HDL语言面对的是编译器，wire对应于连续赋值，如assign。reg对应于过程赋值，如always，initial。

ii. 综合时使用电路思路，Verilog HDL语言面对的是综合器，wire综合出来一般情况下是一根导线。Reg综合出来不一定是register，在always中以 always @ (a or b or c) 形式的，即不带时钟边沿的，综合出来是组合逻辑，只是net；以 always @ (posedge clk)

形式的，即带有边沿的，综合出来一般是时序逻辑，才是以Flip-Flop形式表示的register触发器。

iii．设计中，输入信号一般来说不能判断出上一级是寄存器输出还是组合逻辑输出，对于本级来说，就当成一根导线，即wire型。而输出信号是reg还是组合逻辑输出即wire和reg型都可以。但一般整个设计（即顶层模块）的外部输出是reg输出，这比较稳定、扇出能力好。

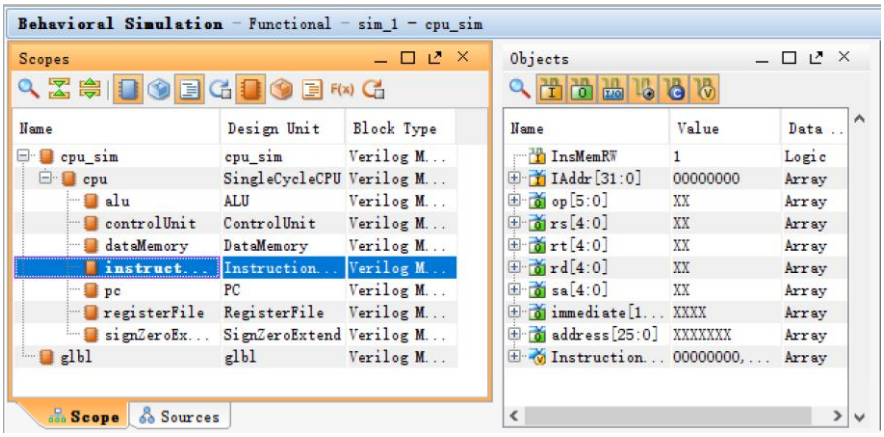
iv．assign语句应用wire，否则编译器报错；元件实例化时必须用wire型，因为wire为无逻辑连线，输入什么就的输出什么，如 assign　　c = a & b; 综合时c是连接到a & b综合成 a、b与门的输出线，综合出来的是与门而不是c，因此always语句对wire变量赋值编译器会报错。

v．reg型数据保持最后一次的赋值，而wire型数据需要持续的驱动。

vi．Verilog HDL语法规定调用子模块时，输出端口只能用wire类型变量进行映射。

2、Vivado的使用：

i．仿真时可通过Scope窗口观察各模块的变量的值。



ii．Messages窗口中可排查错误，有时应与Tcl Console相结合使用。