实验平台说明

特征	说明
操作系统版本和系统类型	Windows 10 64 bit
Quartus 版本	Quartus Prime 13.1.0
验证方式	Simnios 仿真

自查清单

部分	完成度
Part1	100%
Part2	100%
Part3	100%
Part4	100%
Part5	0%

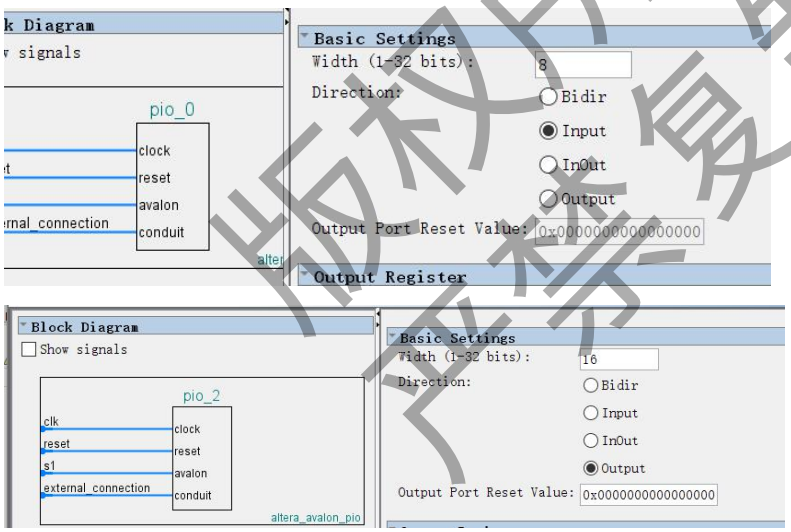
第一部分　Part 1/Lab1

一.题目分析

使用 Quartus ii 建立一个的 nios ii 系统。使用 qsys 并包含以下部分，JTAG,RAM,两路输出 PIO，一路输入 PIO。然后自动分配器件中的基地址。生成系统后，在模块中例化生成的 nios ii 系统。搜索到正点原子 FPGA 教程中有 nios ii 中生成系统的详细步骤，这里只需要按照既定步骤建立即可。

二．实验过程

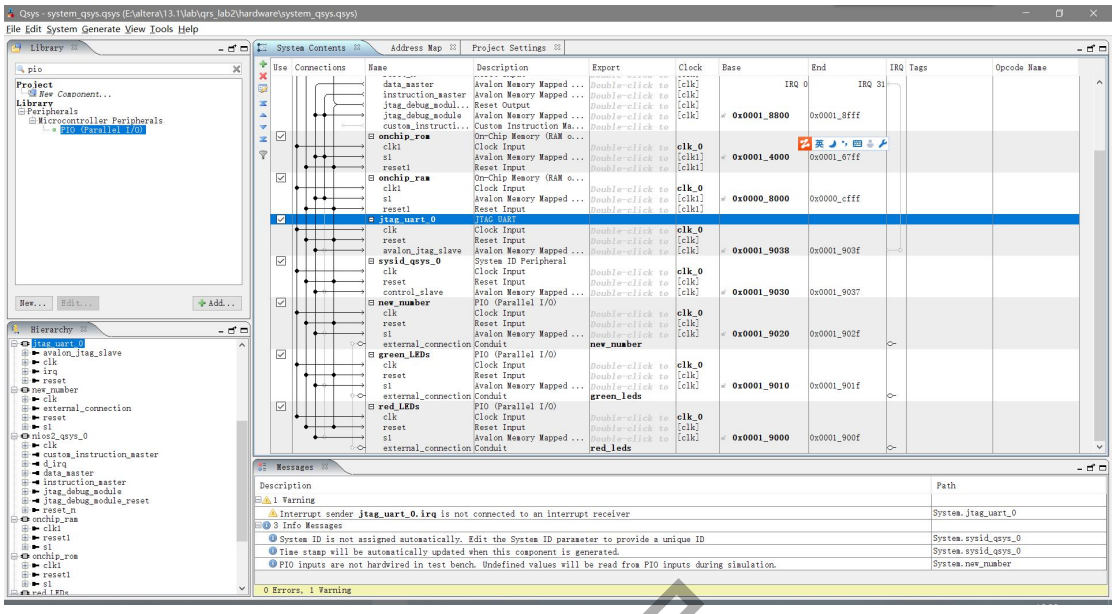
1.类似于 lab1 中的 part1 部分，只是多出以下两个部分：PIO 器件。



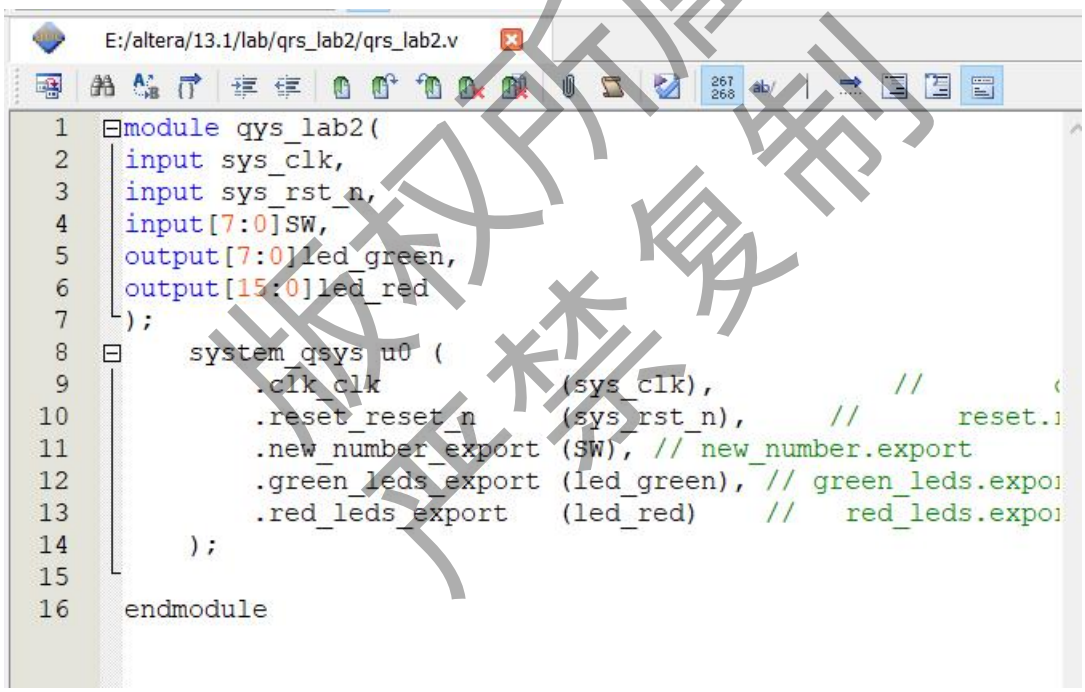
2.连接器件并将端口引出，这里参考了正点原子的方法，双击并起名。

3.自动分配地址，并把地址储存好可以看到地址如下。

```
led_red,0X00019000
led_green,0X00019010
sw,0X00019020
```



4.编写.V 文件，例化系统。



```
module qys_lab2(
input sys_clk,
input sys_rst_n,
input[7:0]SW,
output[7:0]led_green,
output[15:0]led_red
);
    system_qsys u0 (
        .clk_clk              (sys_clk),          //
        .reset_reset_n        (sys_rst_n),     //      reset.
        .new_number_export    (SW), // new_number.export
        .green_leds_export    (led_green), // green_leds.expo
        .red_leds_export      (led_red)    //   red_leds.expo
    );

endmodule
```


三．代码详解

该部分无代码。

四．组长点评

本部分较为简单，按照正点原子的例子，一步一步来即可。

第二部分　Part 2/Lab1

一 .题目分析

PART2，主要使用汇编来 LED 进行操作，由于手边没有板子，只能仿真。因为 8 位输出 REDLED 及 16 个 GREENLED 的亮灭，在汇编中主要由地址中的值来调节，即在相应寄存器下进行调节即可。所以本汇编程序主要就是将 SW 的数据累积并转移到 LED 相应地址。

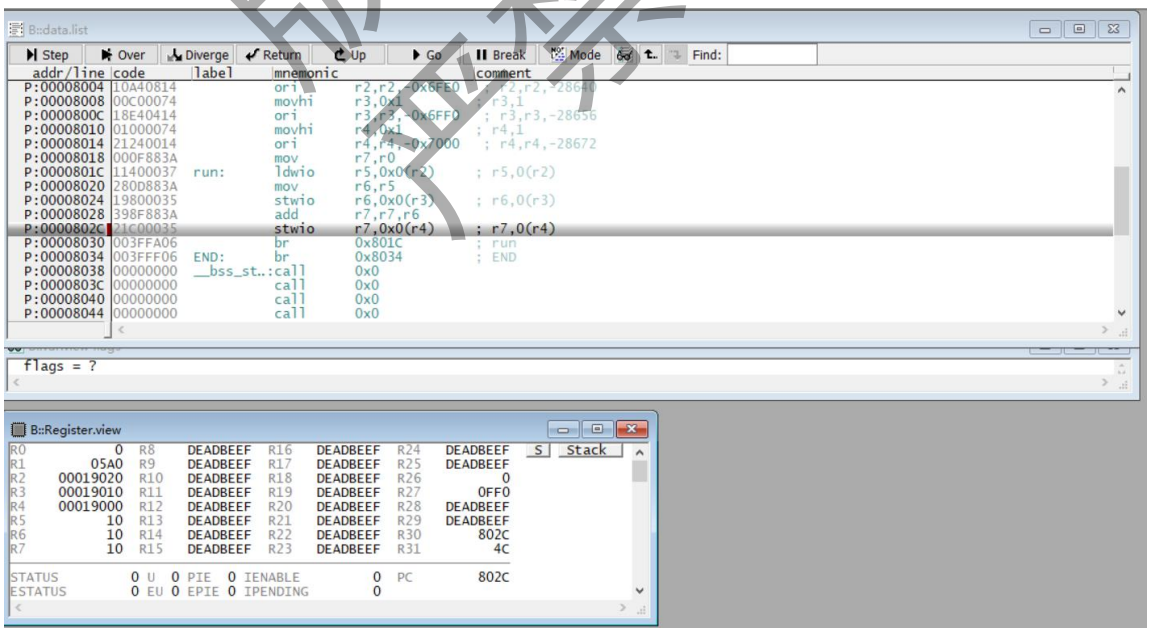
二．实验过程

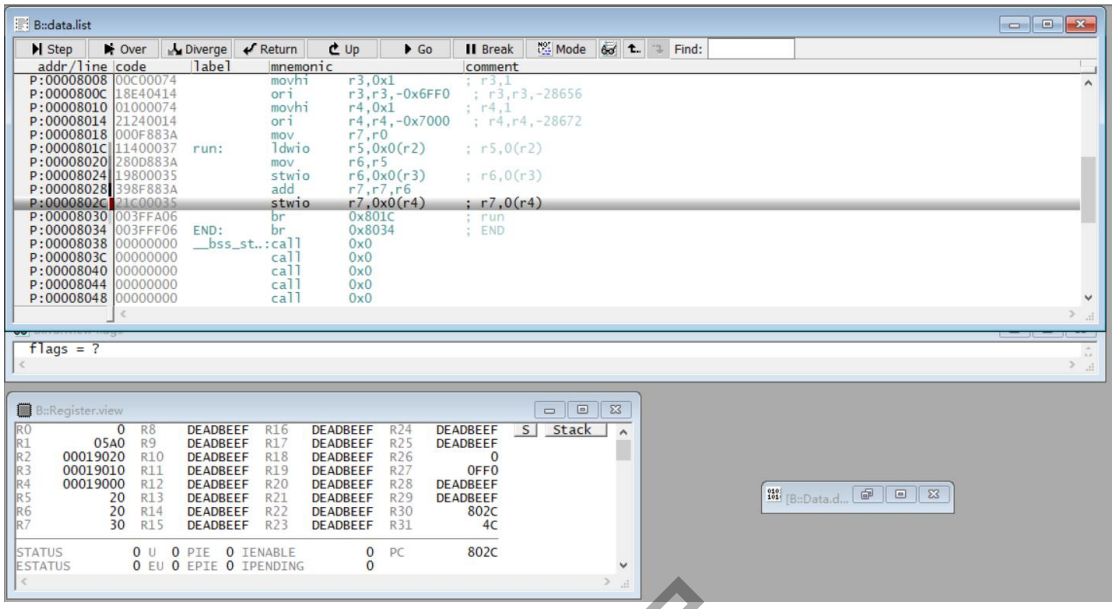
1.编写代码

lab2_part.s

2.仿真

3.观察寄存器及地址值，发现结果正确。R5,R6,R7 中，R5 为 SW 值，R6 为 LEDGREEN 的值，R7 为 LEDRED 的值。





三．代码详解

对每一句代码进行详细解释。

```
.include "nios_macros.s"

.text
.equ led_red,0X00019000//REDLED 地址
.equ led_green,0X00019010//GREENLED 地址
.equ sw,0X00019020//SW 地址

.global _start
_start:
        movia r2,sw //将 SW 的地址转移到 R2 中
        movia r3,led_green//将 GREENLED 的地址转移到 R3 中
        movia r4,led_red//将 REDLED 的地址转移到 R3 中
        mov r7,r0 //给 R7 清 0
run:
        ldwio r5,0(r2) //从 R2 所指地址中装载值到 R5
        mov r6,r5 //把 R5 中的值转移到 R6
        stwio r6,0(r3)//把 R6 中的值转移到 R3 所指地址中。
        add r7,r7,r6//    r7 的原始值加上 R6 赋给 R7
        stwio r7,0(r4)//把 R7 中的值转移到 R3 所指地址中
        br run//回到 RUN
END:
        br END
.end
```

四．组长点评

本实验难度主要在于各种寄存器的转移，只要用好 ldwio 和 sdwio 即可，一个是从 SW 中装载，向是从 LED 中赋值。

第三部分　Part4 **Part 3/Lab1**

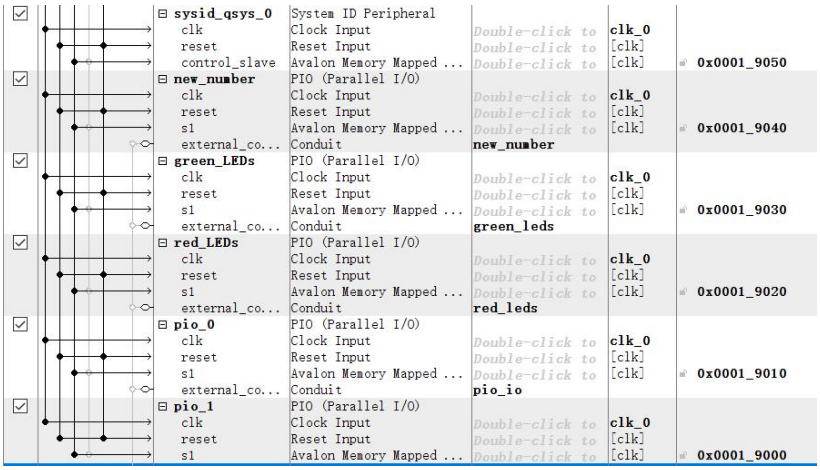
一 .题目分析

PART3,PART4 相对于 PART2，主要增加了用按键控制计数，和使用汇编来对 HEX 数码管进行操作，由于手边没有板子，只能进行仿真。因为 8 位数码管，在汇编中主要由地址中的值来调节，即在相应地址下进行调节即可。所以本汇编程序主要就是检测按键及，将 SW 的数据累积并转移到 HEX 相应地址。

二．实验过程

1.新建系统，添加一位 PIO 输入和 32 位 PIO 输出，分别命名为 PIO_IO 和 PIO_HEX。其中地址如下：（data 地址会在后面解释）

```
status,0x00019010
hex,0x00019000
led_red,0x00019020
led_green,0x00019030
sw,0x00019040
data,0x00006000
```

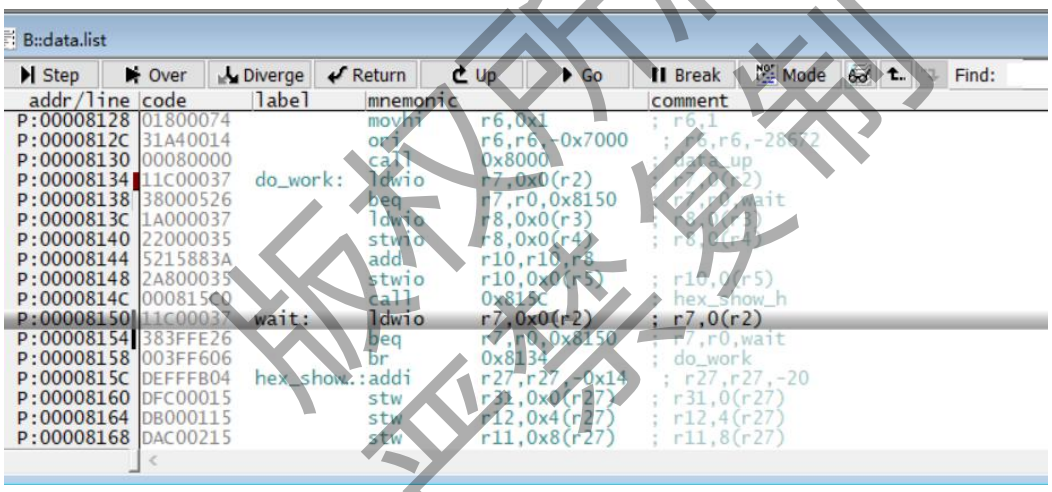


⊟ sysid_qsys_0	System ID Peripheral				
clk	Clock Input	Double-click to	clk_0		
reset	Reset Input	Double-click to	[clk]		
control_slave	Avalon Memory Mapped ...	Double-click to	[clk]		0x0001_9050
⊟ new_number	PIO (Parallel I/O)				
clk	Clock Input	Double-click to	clk_0		
reset	Reset Input	Double-click to	[clk]		
s1	Avalon Memory Mapped ...	Double-click to	[clk]		0x0001_9040
external_co...	Conduit	new_number			
⊟ green_LEDs	PIO (Parallel I/O)				
clk	Clock Input	Double-click to	clk_0		
reset	Reset Input	Double-click to	[clk]		
s1	Avalon Memory Mapped ...	Double-click to	[clk]		0x0001_9030
external_co...	Conduit	green_leds			
⊟ red_LEDs	PIO (Parallel I/O)				
clk	Clock Input	Double-click to	clk_0		
reset	Reset Input	Double-click to	[clk]		
s1	Avalon Memory Mapped ...	Double-click to	[clk]		0x0001_9020
external_co...	Conduit	red_leds			
⊟ pio_0	PIO (Parallel I/O)				
clk	Clock Input	Double-click to	clk_0		
reset	Reset Input	Double-click to	[clk]		
s1	Avalon Memory Mapped ...	Double-click to	[clk]		0x0001_9010
external_co...	Conduit	pio_io			
⊟ pio_1	PIO (Parallel I/O)				
clk	Clock Input	Double-click to	clk_0		
reset	Reset Input	Double-click to	[clk]		
s1	Avalon Memory Mapped ...	Double-click to	[clk]		0x0001_9000

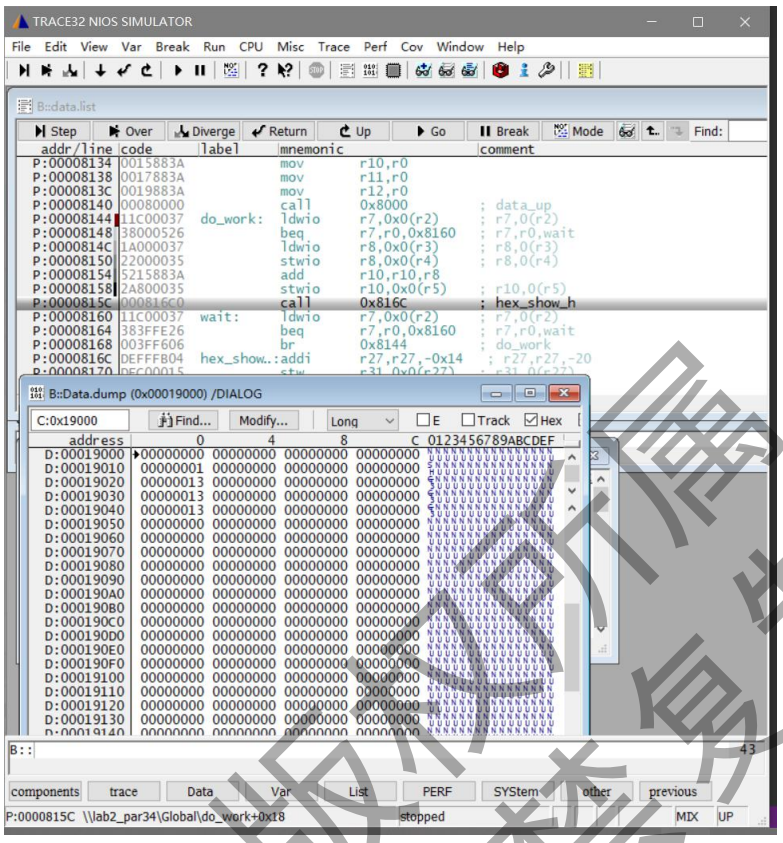
2.编写代码

lab2_par34.s

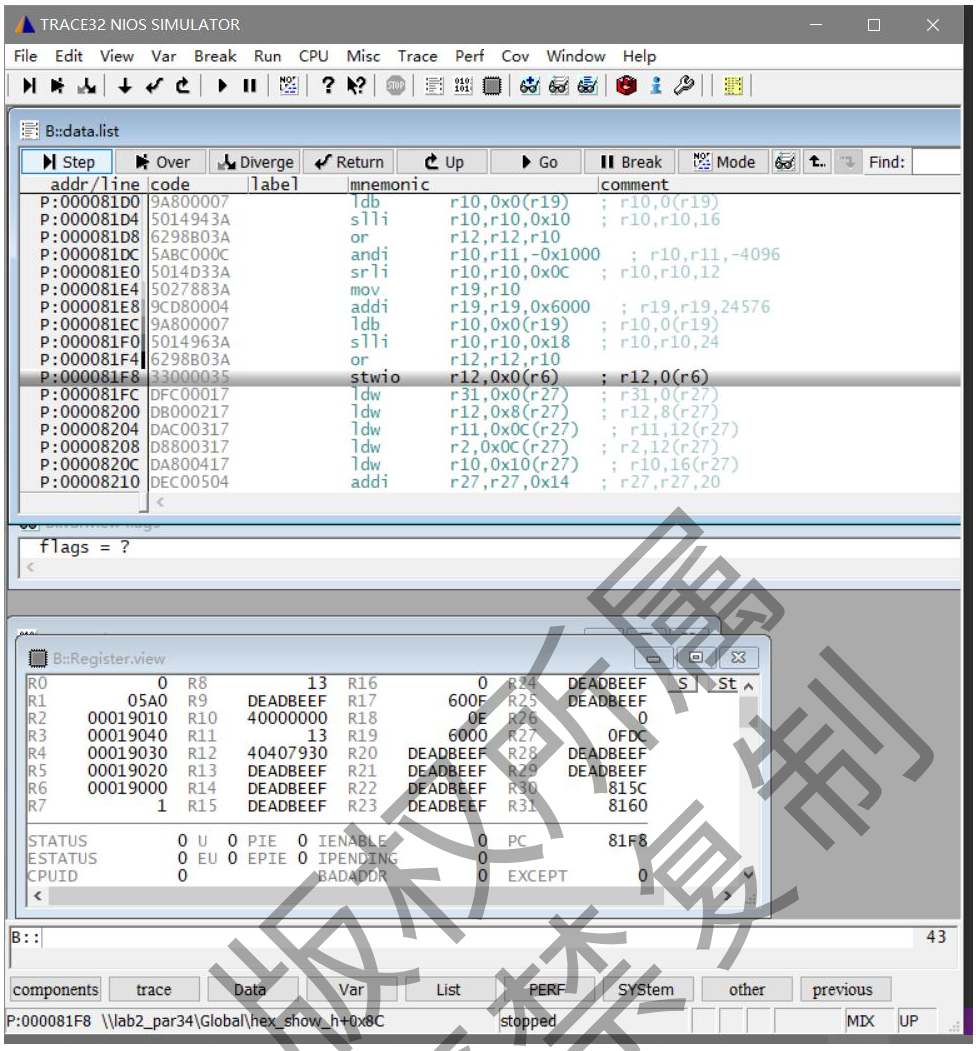
3.仿真



4.观察寄存器及地址值，发现结果正确。各地址中，19040 为 SW 值 0X00000013，19030 为 LEDGREEN 的值，19020 为 LEDRED 的值，结果正确。



5.观察地址 0X00019000 即寄存器 R12 的值，为 HEX。结果正确



三．代码详解

该 PART 代码十分复杂，这里按逻辑来讲：

首先列出表格，为数码管中 8 个 IO 为一组的值与数值 1234 •••• 的关系。

0	0x40
1	0x79
2	0x24
3	0x30
4	0x19
5	0x12
6	0x02

7	0x1f
8	0x00
9	0x18
A	0x08
b	0x03
c	0x46
d	0x21
E	0x06
F	0x0e

即假若要显示 13 则，地址 HEX 中必须为 0X40407930。

现在为了达到此目的来进行编程，主要思路为找一个 DATA 储存地址从 0X00006000 到 0x00006010 间的十六个地址中的 32 位空间储存该数组（从 0x40 到 0x0e）。然后若此时要显示 1,则查询 1+0X00006000 即可查询到对应的码值。代码如下：分句解释。

四．组长点评

```
.include "nios_macros.s"

.text
.equ status,0x00019010 //按键标志位的地址
.equ hex,0x00019000//数码管地址
.equ led_red,0x00019020//红 led 地址
.equ led_green,0x00019030//绿 led 地址
.equ sw,0x00019040//拨码开关地址
.equ data,0x00006000//码值地址

data_up://数据准备，将所有码值储存到 0X00006000 到 0x00006010 的地址中。

    movia r17,0x00006000//赋值立即数码值地址。
    movia r18,0x40//赋值立即数码值
    stw r18,0(r17)//把 R18 中的码值放入 R17 中的地址中。

    addi r17,r17,0x00000001//以下同理
    movia r18,0x79
    stw r18,0(r17)

    addi r17, r17,0x00000001
    movia r18,0x24
```

```
stw r18,0(r17)

addi r17,r17,0x00000001
movia r18,0x30
stw r18,0(r17)

addi r17,r17,0x00000001
movia r18,0x19
stw r18,0(r17)

addi r17,r17,0x00000001
movia r18,0x12
stw r18,0(r17)

addi r17,r17,0x00000001
movia r18,0x02
stw r18,0(r17)

addi r17,r17,0x00000001
movia r18,0x1f
stw r18,0(r17)

addi r17,r17,0x00000001
movia r18,0x00
stw r18,0(r17)

addi r17,r17,0x00000001
movia r18,0x18
stw r18,0(r17)

addi r17,r17,0x00000001
movia r18,0x08
stw r18,0(r17)

addi r17,r17,0x00000001
movia r18,0x03
stw r18,0(r17)

addi r17,r17,0x00000001
movia r18,0x46
stw r18,0(r17)

addi r17,r17,0x00000001
movia r18,0x21
```

```
    stw r18,0(r17)

    addi r17,r17,0x00000001
    movia r18,0x06
    stw r18,0(r17)

    addi r17,r17,0x00000001
    movia r18,0x0e
    stw r18,0(r17)
    br do_work

.global _start
_start: //程序开始运行。
    movia r2,status//转移地址到寄存器中
    movia r3,sw
    movia r4,led_green
    movia r5,led_red
    movia r6,hex

    mov r19,r0//给使用到的寄存器赋 0
    mov r10,r0
    mov r11,r0
    mov r12,r0

    call data_up//数据准备
do_work:
    ldwio r7,0(r2)//把 R2 中的值载入到 R7 中
    beq r7,r0,wait//比较与 R0，若 0 则转移到等待，并不断等待。
    ldwio r8,0(r3) //PART2 中内容
    stwio r8,0(r4) //PART2 中内容

    add r10,r10,r8 //计算累加，并储存到 R10 中
    stwio r10,0(r5) //计算累加并载入到 R5 中
    call hex_show_h
wait:
    ldwio r7,0(r2) //把 R2 中的值载入到 R7 中
    beq r7,r0,wait 比较与 R0，若 0 则转移到等待，并不断等待。
    br do_work//若不为 0 则继续运行 do_work

hex_show_h://以下为显示程序，R10 为累积值，以 0x0013 为例
    subi sp,sp,0x0014//此处保存 sp 借鉴詹冬霖大佬，其实我也不懂
    stw ra,(sp)
    stw r12,4(sp)
    stw r11,8(sp)
```

```
stw r2,12(sp)
stw r10,16(sp)
                    //r10=0x0013 为例
andi r10,r10,0xffff //r10|0xffff=0x0013
mov r11,r10        //r11=0x0013,r10=0x0013
andi r10,r11,0x000f //r10|0x000f=0x0003

mov r19,r10          //r19=r10=0x0003
addi r19,r19,0x00006000//r19=0x00006003
ldb r10,0(r19)//将 R19 所指地址中的值放入 R10 中，查表为 0X30

mov r12,r10         //R12=R10=0X30

andi r10,r11,0x00f0   //R10=0X0010
srli r10,r10,0x0004  //R10=0X0001

mov r19,r10          //r19=r10=0x0001
addi r19,r19,0x00006000//r19=0x00006001
ldb r10,0(r19) //将 R19 所指地址中的值放入 R10 中，查表为 0X79

slli r10,r10,0x0008 //R10=0X7900
or r12,r12,r10      //R12=0X7930
                //同理以下为两次运行为 0
andi r10,r11,0x0f00
srli r10,r10,0x0008

mov r19,r10
addi r19,r19,0x00006000
ldb r10,0(r19) //R10=0X40

slli r10,r10,0x0010
or r12,r12,r10

andi r10,r11,0xf000
srli r10,r10,0x000c

mov r19,r10
addi r19,r19,0x00006000
ldb r10,0(r19)//R10=0X40

slli r10,r10,0x0018
or r12,r12,r10

stwio r12,0(r6)   //R12=0X40407930       ，转移到 HEX 地址中
```

```
    ldw ra,(sp)
    ldw r12,8(sp)
    ldw r11,12(sp)
    ldw r2,12(sp)
    ldw r10,16(sp)
    addi sp,sp,0x0014
    br do_work

END:
        br END
.end
//
```

如此 PART34 完成

四．组长点评

本实验难度主要在于 HEX 的显示，需要想到如何填表，如何查询即可完成，过程比较复杂，需要查询一些网上的程序，但网上的程序大多基于 51，只要学会其思想，即可想到。