计算机组成原理 实验报告

姓名：龚小航　学号：PB18151866　实验日期：2020-6-17

一、实验题目：

Lab06 综合设计

二、实验目的

- 理解计算机系统的组成结构和工作原理；
- 理解计算机总线和接口的结构和功能；
- 掌握软硬件综合系统的设计和调试方法；
- 具体目标：

设计实现一个简单的计算机应用系统：利用设计好的 CPU，再选择合适的外设完成一个有实际应用功能的简单系统。

三、实验平台：

Vivado

四、实验过程：

1. 设计目标：

综合实验设计一个简单的 CPU 控制系统：

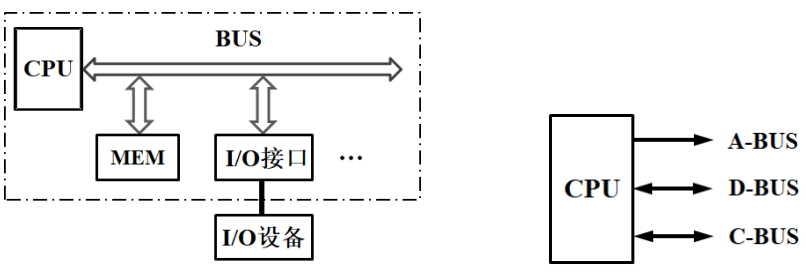
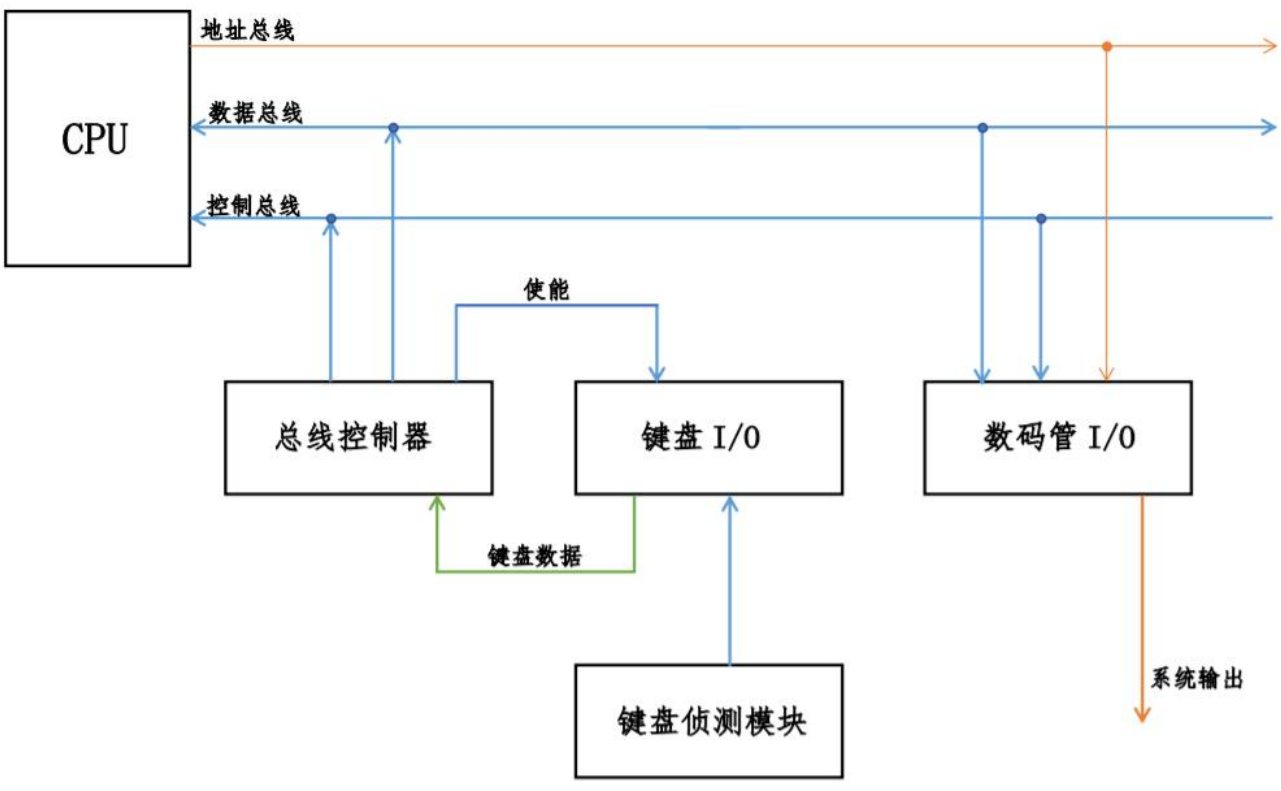


图 1 带 I/O 系统的逻辑图

本实验设计的应用系统实现通过外设输入 CPU 的运行指令，控制系统的运行状态。在这个系统中，CPU 不断通过程序查询方式轮询 I/O 接口是否准备好数据；键盘可以向 CPU 的当前地址中存入 16 进制形式的 32 位 MIPS 指令，当键盘 I/O 接收到完整的一条指令时就将自身的标志位设置为高电平，此时键盘数据通过总线控制器经由数据总线和控制总线传入 CPU 中，向 CPU 的当前 pc 指向的地址中写入该指令并立即执行，而需要向任意地址写入某数据时可以写入 j 指令实现；同一个地址中后写入的数据会覆盖先写入的数据。CPU 的运行情况可以用其他外设体现出来，此处选择的是七段数码管。只需要用其他未使用的键盘按键来控制数码管显示的内容即可。当输入完整的指令后，七段数码管会将输入的指令以十六进制的形式显示在八位的七段数码管上，如果按住键盘上的 ENTER 键，七段数码管则显示当前的 pc 值。本实验的 SSEG 模块已经加入了分频，因此在仿真的少量系统时钟周期内不能看出 SSGE_CA 与 SSEG_AN 的变化，在实物开发板上可以看出。

设计逻辑图如下所示：



图中所有模块全部采用例化实现。顶层模块 LAB6_TOP 仅声明信号且例化模块。

由于 CPU 已经在之前的实验中验证了完整性和正确性，本次实验重点在于外设的使用设计与同步。

具体的代码附于源文件中，并标有注释。

2. 设计实现：

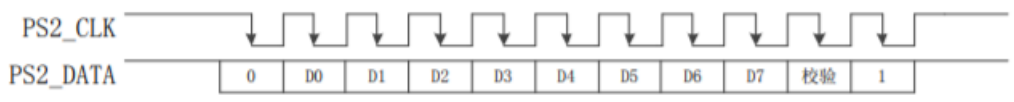
以下为顶层模块的连接，这是整个程序的总体与框架。

```
module LAB6_TOP(
    input clk,
    input rst,
    input ps2_clk,ps2_data,

    output [7:0] SSEG_CA,
    output [7:0] SSEG_AN
    );

    wire [31:0] IO_DATA_din,IO_DATA_dout ,PC;
    wire readIO_finished_in,readIO_finished_out, SSEG_MODE;
    wire  PRESS_0,PRESS_1,PRESS_2,PRESS_3,PRESS_4, PRESS_5,PRESS_6,PRESS_7,PRESS_8,
          PRESS_9,PRESS_A,PRESS_B,PRESS_C, PRESS_D, PRESS_E,PRESS_F,PRESS_ENTER;

    TOP UCPU (clk, rst, readIO_finished_out, IO_DATA_dout, PC);
    //以下为两个IO设备模块，即键盘输入与数码管输出模块//
    KeyBoard_TOP UKEY (clk, rst, ps2_clk, ps2_data, PRESS_0, PRESS_1, PRESS_2,PRESS_3, PRESS_4, PRESS_5, PRESS_6, PRESS_7,
                       PRESS_8, PRESS_9, PRESS_A, PRESS_B, PRESS_C, PRESS_D, PRESS_E, PRESS_F, PRESS_ENTER);
    KeyBoard_IO UKEYIO (clk, PRESS_0, PRESS_1, PRESS_2,PRESS_3, PRESS_4, PRESS_5, PRESS_6, PRESS_7,
                       PRESS_8, PRESS_9, PRESS_A, PRESS_B, PRESS_C, PRESS_D, PRESS_E, PRESS_F, PRESS_ENTER,
                       IO_DATA_din, readIO_finished_in, SSEG_MODE);
    SSEG_part USSEG (clk, IO_DATA_dout, PC, SSEG_MODE, SSEG_CA, SSEG_AN);
        ////////////////////////////////////////////////
    BUS_CONTROL UBUS (IO_DATA_din,readIO_finished_in,IO_DATA_dout,readIO_finished_out);

endmodule
```

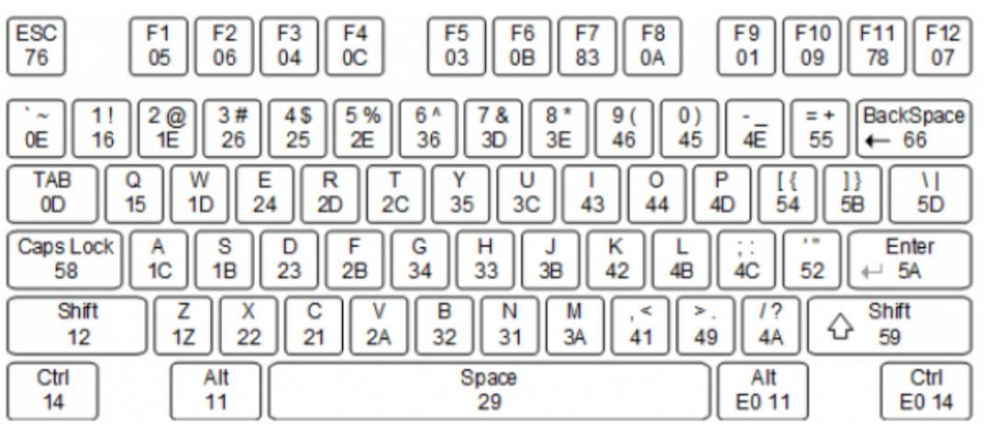
可见 LAB6_TOP 就是对设计逻辑图的例化连线。

以下是键盘 I/O 与外设的设计：

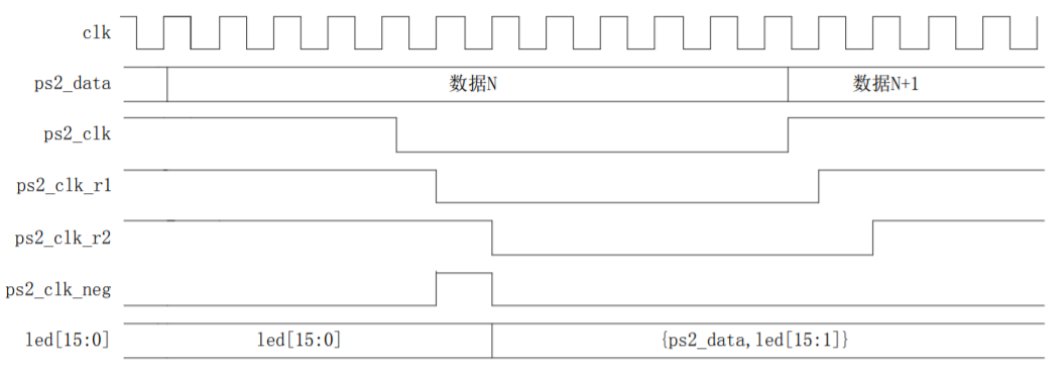
键盘传输原理如下所示：当 PS2 设备向主机发送数据时，会首先检测 ps2_clk 时钟信号是否为高电平，然后连续发送 11 个时钟负脉冲，并在数据线上发送 11bit 的数据，包括起始位（1bit）、数据位（8bit）、校验位（1bit）和停止位（1bit），主机可在 PS2_CLK 信号的下降沿对数据信号进行采样，其时序图如下所示：



在键盘按键、松开时都会向主机发送数据，某按键按下时，会发送该键对应的通码，一般来说是一个 8 位二进制数据；当某键被松开时，发送该键的断码，即 F0+通码。键盘内部已经自带去抖动模块，输出的是“干净”的信号。标准 ps2 键盘对应的通码如下图所示：

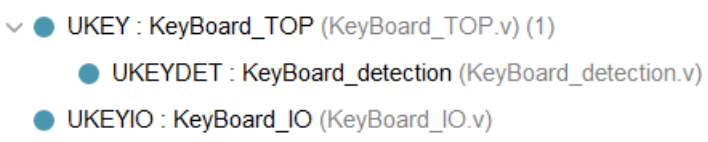


例如当‘a’被按下时，发送一个字节“1C”，该按键松开时，发送两个字节“F0 1C”。整个系统的时序示意图如下所示：



需要注意的是，ps2_clk 的频率一般远低于系统时钟频率。在常见的系统中，系统时钟 clk 为 100MHz，I/O 时钟 ps2_clk 为 10KHz 左右。

在本次实验中，键盘部分的关系如图所示：



探测模块的作用为接收外部输入，即键盘给出的 ps2_clk 以及 ps2_data 信号，并将最近接收到的 24 位信号存储在该模块的输出变量[23:0]key 中。Key 的更新只有当接收到 8 位信号以后才会变化一次，以确保中间不会出现冒险。新传入的数据存放在高位，即 key[23:16],并将原来的数据右移 8 位。

KeyBoard_TOP 模块用于译码，对探测模块传入的 key 值分析并生成相应的按键按下信号。按键被按下的判定准则可以根据 ps2 键盘码的特点，设定为 key[23:16]=通码且 key[15:8]≠F0.根据功能设计，0~F 键信号我们不关心它们按了多久，而是关注它们被按

了几次以及何时按下。因此可以对其取边沿处理，只要在键盘上按下，不论何时松开都立刻认为这个键被按下一次并记录；而用于控制数码管显示内容的 ENTER 键需要实现按住时显示 pc 值，因此不需要对这个信号取边沿。

KeyBoard_IO 模块是键盘外设与总线连接的处理模块，外设通过 I/O 接口连接至总线并于 CPU 相连。这个模块记录了外设的工作方式，在本实验中即是对接收到的按键信号打包处理，并在接收到完整的 32 位 MIPS 指令后将 I/O 就绪标志设为高电平，同时将数据与控制信号经由总线传至 CPU 与数码管 I/O。

其余就是输数码管模块。为使八位数码管显示不同的内容，必须时分复用。引入一个分频变量[18:0]Reg_N，兼以位选功能。每个时钟周期（DBU 外接板载时钟，100MHz）将 Reg_N 的值+1，该变量的最高三位作为位选信号，对应 8 个数码管，且每根数码管占用的显示时长相等。

```
64  //////////////////////////////////////////////////////////////////////////////////
65   localparam N = 18; //使用低位对100Mhz的时钟进行分频
66      reg [N-1:0] regN; //高位作为控制信号，低位为计数器，对时钟进行分频
67      reg [3:0] hex_in; //段选控制信号
68      reg [3:0] hex0,hex1,hex2,hex3,hex4,hex5,hex6,hex7;
69      initial hex0=0; initial hex1=0;initial hex2=0; initial hex3=0;
70      initial hex4=0; initial hex5=0;initial hex6=0; initial hex7=0;
71      initial regN=0;
72  //////////////////////////////////////////////////////////////////////////////////
```

写入设计的逻辑：

```
22    always@(posedge clk)begin
23       if(SSEG_MODE==0)
24         {hex7,hex6,hex5,hex4,hex3,hex2,hex1,hex0}<=IO_DATA;
25       else
26         {hex7,hex6,hex5,hex4,hex3,hex2,hex1,hex0}<=PC;
27    end
```

最后在下方声明位选以及要显示的内容，数码管部分就完成了。

```
142   //数码管输出
143    always@(*)
144    begin
145     case(regN[N-1:N-3])
146           3'b000:begin
147               SSEG_AN = 8'b11111110; //选中第1个数码管
148               hex_in = hex0; //数码管显示的数字由hex_in控制，显示hex0输入的数字;
149            end
150           3'b001:begin
151               SSEG_AN = 8'b11111101; //选中第2个数码管
152               hex_in = hex1;
153            end
154           3'b010:begin
155             SSEG_AN = 8'b11111011; //选中第3个数码管
156             hex_in = hex2;
157            end
158           3'b011:begin
159             SSEG_AN = 8'b11111011; //选中第4个数码管
160             hex_in = hex3;
161            end
162           3'b100:begin
163               SSEG_AN = 8'b11101111; //选中第5个数码管
164               hex_in = hex4;
165            end
166           3'b101:begin
167               SSEG_AN = 8'b11011111; //选中第6个数码管
168               hex_in = hex5;
169            end
170           3'b110:begin
171             SSEG_AN = 8'b10111111; //选中第7个数码管
172             hex_in = hex6;
173          end
174         3'b111:begin
175             SSEG_AN = 8'b01111111; //选中第8个数码管
176             hex_in = hex7;
177          end
178         default: SSEG_AN=8'b11111111;
179      endcase
180   end
```

Hex_in 决定了该时刻数码管显示的内容；SSEG_AN 决定哪个数码管发光。

```
73    //数码管输出部分
74    always@(*)begin
75        case(hex_in)
76                        4'h0: SSEG_CA[7:0] = 8'b00000011; //共阳极数码管
77                        4'h1: SSEG_CA[7:0] = 8'b10011111;
78                        4'h2: SSEG_CA[7:0] = 8'b00100101;
79                        4'h3: SSEG_CA[7:0] = 8'b00001101;
80                        4'h4: SSEG_CA[7:0] = 8'b10011001;
81                        4'h5: SSEG_CA[7:0] = 8'b01001001;
82                        4'h6: SSEG_CA[7:0] = 8'b01000001;
83                        4'h7: SSEG_CA[7:0] = 8'b00011111;
84                        4'h8: SSEG_CA[7:0] = 8'b00000001;
85                        4'h9: SSEG_CA[7:0] = 8'b00001001;  //以上为数字
86
87                        4'ha: SSEG_CA[7:0] = 8'b00010001;//A
88                        4'hb: SSEG_CA[7:0] = 8'b11000001;//b
89                        4'hc: SSEG_CA[7:0] = 8'b01100011;//C
90                        4'hd: SSEG_CA[7:0] = 8'b10000101;//d
91                        4'he: SSEG_CA[7:0] = 8'b01100001;//E
92                    default: SSEG_CA[7:0] = 8'b01110001;//F
93            endcase
94    end
95
96  endmodule
```

五、实验结果：

为了验证系统功能的完整性与正确性，只需要给出模拟 ps2 键盘的输入信号即可。指令存储器和数据存储器的内容均可以通过执行指令得到任意形式的更改。

仿真文件需要给出系统时钟 clk，系统复位信号 rst，键盘时钟信号 ps2_clk 以及键盘数据信号 ps2_data.

因此，仿真信号需要给出 ps2_clk 的 11 个负脉冲系列，以及对应的传输数据。这一部分比较繁琐冗长。

```
module LAB6_SIM();
  reg clk, rst, ps2_clk, ps2_data;
  wire [7:0] SSEG_CA,SSEG_AN;
  parameter PERIOD = 5,       //时钟周期长度
            CYCLE = 150;
  LAB6_TOP test(clk,rst,ps2_clk, ps2_data,SSEG_CA,SSEG_AN);

  initial begin
        clk = 0;
        repeat (500 * CYCLE)
            #(PERIOD/5) clk = ~clk;

  end
  initial begin
    rst=1; #(1) rst=0;
  end

  initial begin
        ps2_clk = 1;
        #(PERIOD*2)
        repeat (22)
            #(PERIOD) ps2_clk = ~ps2_clk;
        #(PERIOD*2)
        repeat (22)
            #(PERIOD) ps2_clk = ~ps2_clk;
        #(PERIOD*2)
        repeat (22)
            #(PERIOD) ps2_clk = ~ps2_clk;
        #(PERIOD*2)
```

未截取部分仍然是若干个 ps2_clk 的变化描述，与上面展示的重复。

```
  initial begin //第一位2
          ps2_data=0;
    #(10) ps2_data=0;  //起始位
    #(10) ps2_data=0;  //数据开始
    #(10) ps2_data=1;
    #(10) ps2_data=1;
    #(10) ps2_data=1;
    #(10) ps2_data=1;
    #(10) ps2_data=0;
    #(10) ps2_data=0;
    #(10) ps2_data=0;  //8位数据，1E对应键盘2
    #(10) ps2_data=0; //校验位
    #(10) ps2_data=1; //终止位

    #(20) ps2_data=0;  //起始位
    #(10) ps2_data=0;  //数据开始
    #(10) ps2_data=0;
    #(10) ps2_data=0;
    #(10) ps2_data=0;
    #(10) ps2_data=1;
    #(10) ps2_data=1;
    #(10) ps2_data=1;
    #(10) ps2_data=1;  //8位数据，F0
    #(10) ps2_data=0; //校验位
    #(10) ps2_data=1; //终止位

    #(20) ps2_data=0;  //起始位
    #(10) ps2_data=0;  //数据开始
    #(10) ps2_data=1;
    #(10) ps2_data=1;
    #(10) ps2_data=1;
    #(10) ps2_data=1;
    #(10) ps2_data=0;
    #(10) ps2_data=0;
    #(10) ps2_data=0;  //8位数据，1E对应键盘2
    #(10) ps2_data=0; //校验位
    #(10) ps2_data=1; //终止位
```

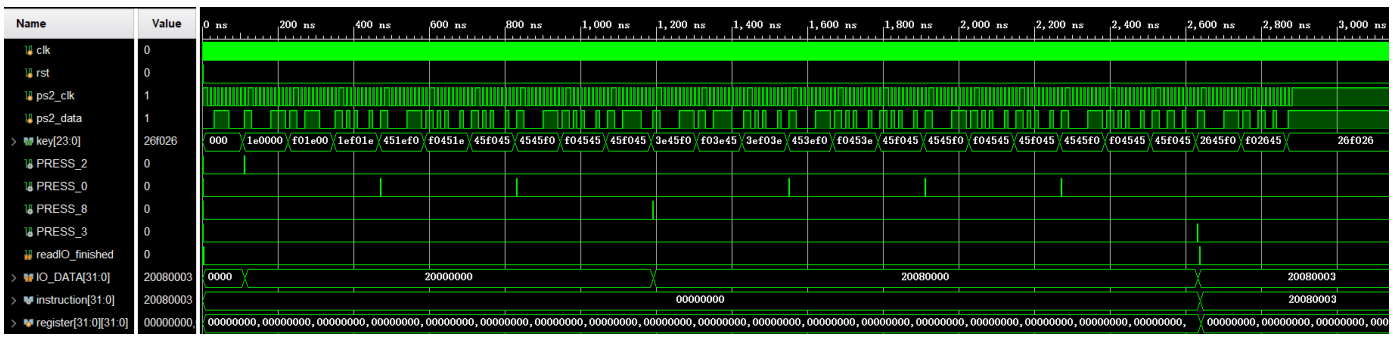
这部分是模拟按键信号的数据值，每一次按键需要上面这样的一个块。作一条指令的仿真需要按键 8 次，即有 8 块上面这样的数据块。其余具体代码见附件源文件。

在开发板上验证时，可以接入键盘简单的输入多条指令。测试程序使用之前测试 CPU 的简单 MIPS 程序。为避免增加若干条 sw 语句与更改数条 j，beq 指令的跳转地址，可以初始化其数据存储器。将这个测试程序写成汇编代码形式如下所示。在 vivado 中仿真只用到了第一条指令。

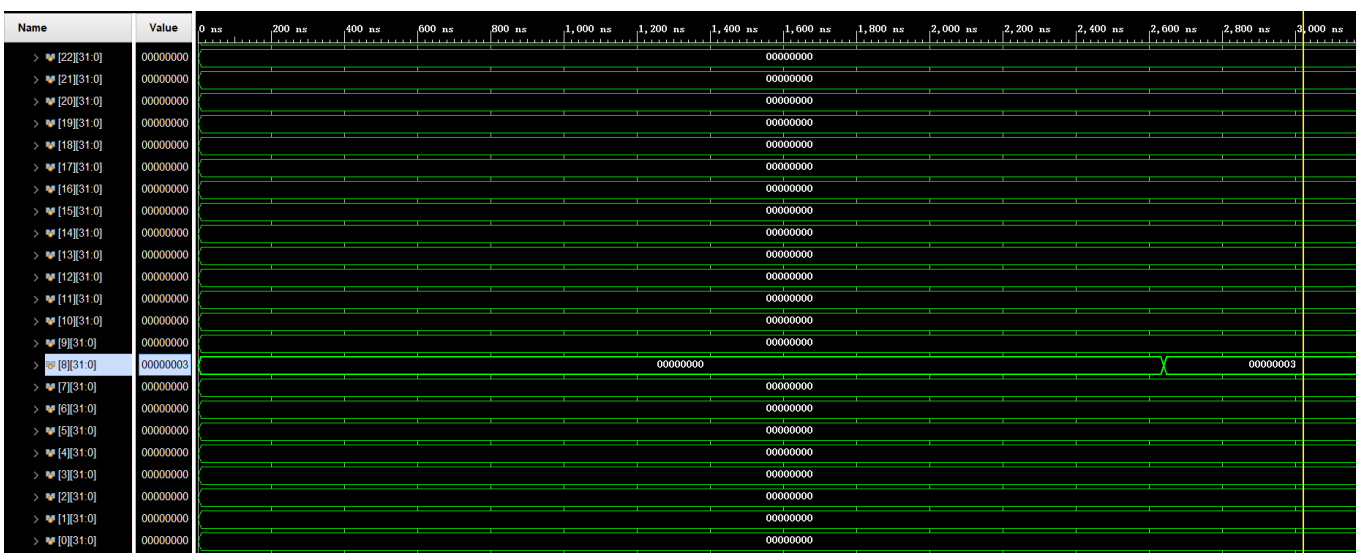
```
# 本文档存储器以字节编址
# 初始PC = 0x00000000

.data
    .word 0,6,0,8,0x80000000,0x80000100,0x100,5,0
                #编译成机器码时，编译器会在前面多加个0，所以后面lw指令地址会多加4

_start:
     addi $t0,$0,3              #t0=3                                    0
     addi $t1,$0,5              #t1=5                                    4
     addi $t2,$0,1              #t2=1                                    8
     addi $t3,$0,0              #t3=0                                    12

     add  $s0,$t1,$t0           #s0=t1+t0=8   测试add指令                16
     lw   $s1,12($0)            #                                        20
     beq  $s1,$s0,_next1        #正确跳到_next                           24

      j _fail

_next1:
     lw $t0, 16($0)             #t0 = 0x80000000  32
     lw $t1, 20($0)             #t1 = 0x80000100  36

     add  $s0,$t1,$t0           #s0 = 0x00000100 = 256                   40
     lw $s1, 24($0)             #                                        44
     beq  $s1,$s0,_next2        #正确跳到_success                        48

      j _fail

_next2:
     add $0, $0, $t2                #$0应该一直为0                       56
     beq $0,$t3,_success            #                                    60


_fail:
     sw   $t3,8($0) #失败通过看存储器地址0x08里值，若为0则测试不通过，最初地址0x08里值为0
        j    _fail

_success:
     sw   $t2,8($0)     #全部测试通过，存储器地址0x08里值为1
        j   _success

                #判断测试通过的条件是最后存储器地址 0x08 里值为 1，说明全部通过测试
```

以上是完整的 MIPS 程序，由于 CPU 的功能完整性与正确性已经在前面的实验中得到了验证。因此只需仿真某一条指令的通信即可。本例选择第一条指令 20080003，将 t0 寄存器的值赋为 3.

仿真结果如下：



再查看最后 8 号寄存器内的值是否为 3：



这一条指令的正确存储及执行可以说明该系统支持外设与 CPU 的通信，即可以正确的存储运行任意一条输入的指令。再结合之前 CPU 设计的正确性与完整性，可以断言该系统达到了设计目标。

六、心得体会：

本次实验需要设计简单的计算机应用系统，CPU 采用 LAB3 设计的单周期 CPU，实现了外设与 CPU 之间的通信。

综合实验的重点在于外设的使用与 I/O 接口的协调工作。这个系统的重点在于总线和总线控制器，以及 I/O 接口的设计。如何同步 CPU 与外设之间的工作是实际应用系统中非常重要的问题。一般来说 CPU 的工作速度、频率都远高于外设。例如在本例中，系统时钟一般为 100MHz，而键盘的时钟 ps2_clk 一般在 10KHz 左右，约为 1/10000.本实验采用了最简单的程序查询方法同步 CPU 与外设的工作，虽然在大型的系统中这样的方法会极大的拖慢 CPU 的运行速度导致 I/O 瓶颈的出现，但在这种小型的实验系统中也是可以接受的。

该系统还可以拓展为更完整的系统。由于键盘的通用性，只需要再编入一些键码，就可以实现类似 DBU 的功能。在有实物的情况下，还可以接入 VGA 模块，在屏幕上把当前 CPU 内的各种信号一起显示出来，这可以极大的增加调试的便捷性。