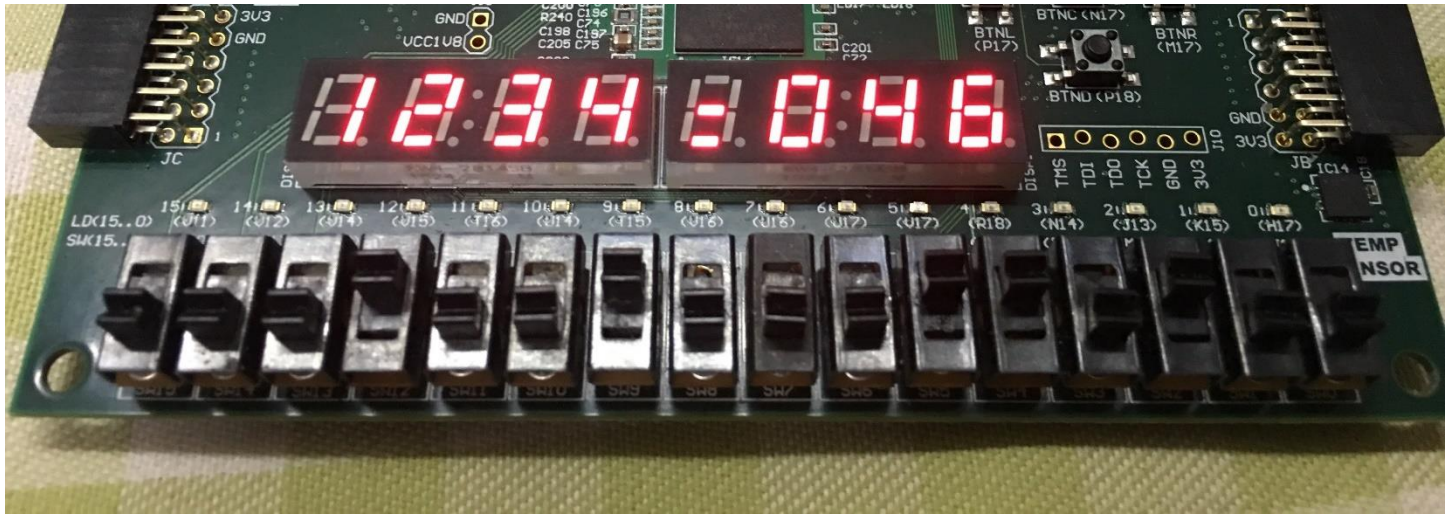
计算机组成和体系结构实验

I/O接口设计



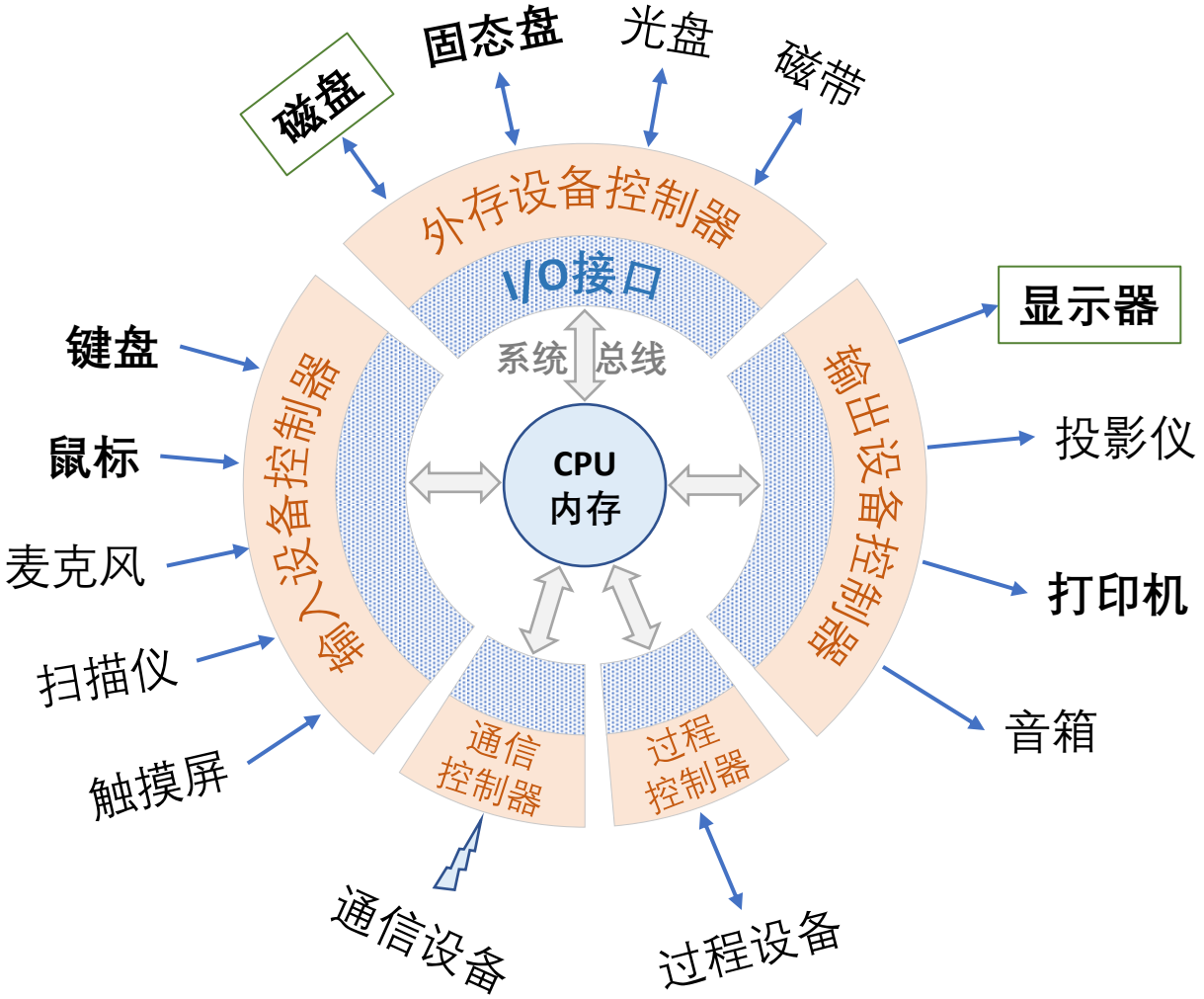
xgsun@fudan.edu.cn

孙晓光

2024-11-10

外围设备

计算机中除**CPU**和**内存**以外的所有设备



接口技术

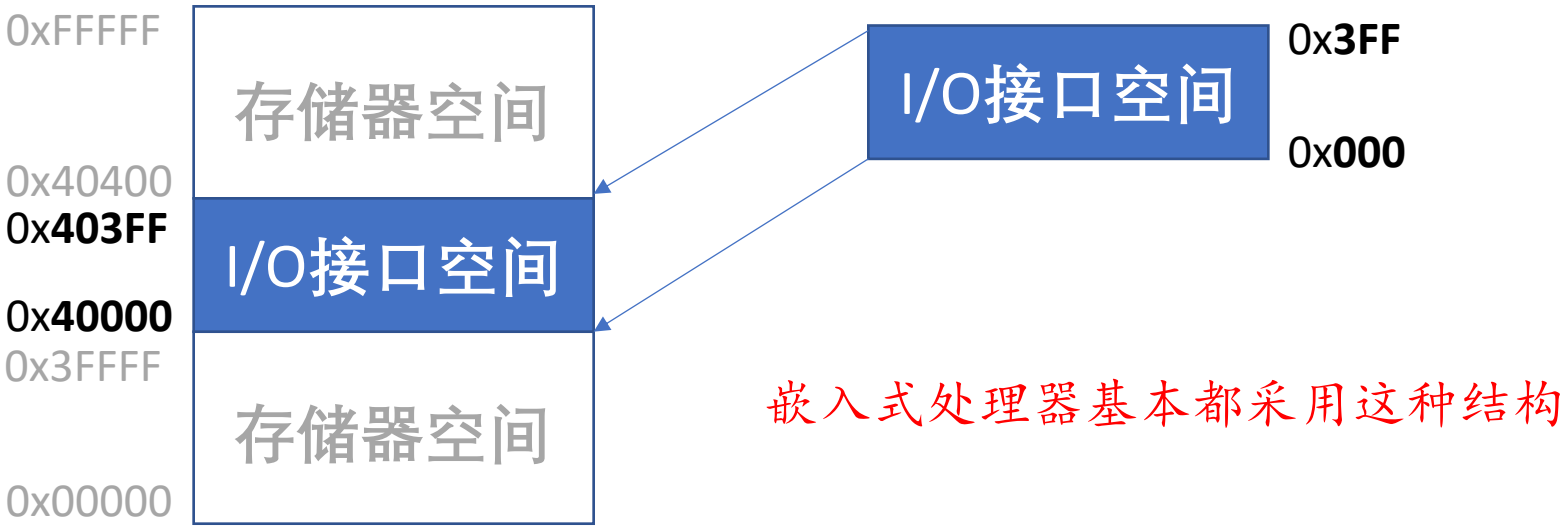
- 输入/输出（**I/O**）：计算机与外界的信息交换。
- 计算机与外界的信息交换是通过**I/O设备**进行。
- 一种**I/O设备**与计算机就需要一个连接电路：**I/O接口**　　通信的桥梁
  - **设备识别**：处理器如何寻址外部设备
  - **设备通信**：处理器如何与外设交换信息，进行**状态**、**控制信号**、**数据** 交换。
- **接口控制方式**：
  - **查询方式**：处理器在传送数据之前查询是否允许传送数据.
  - 中断方式
  - DMA方式

接口中有3类寄存器：

数据端口、状态端口、控制端口

I/O接口结构

- **标准I/O结构**：接口为专有设备，不同于普通存储器。
- **存储器映像I/O接口**：将**I/O接口的地址空间**映像到**存储器的部分地址空间**

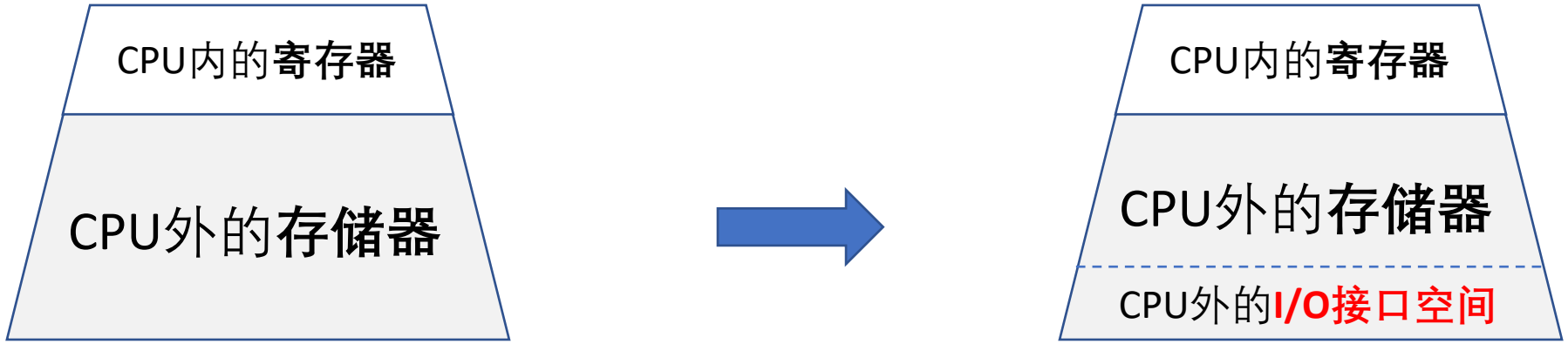


处理器不需要提供专门的接口控制总线，可采用与访问**存储器**一样的方式访问I/O接口。

存储器映像 I/O 寻址方式

特点

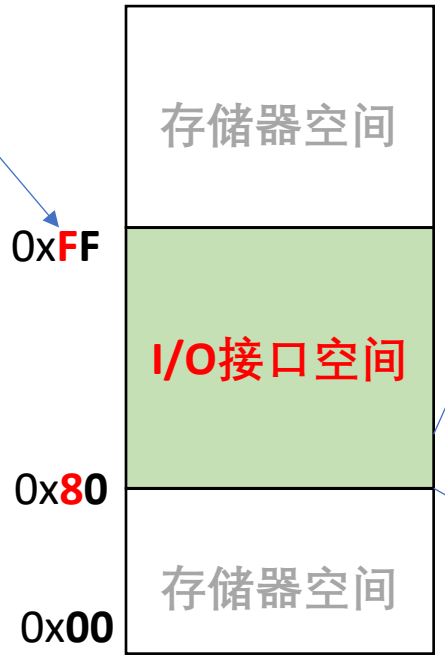
- I/O接口与存储器**共用**同一个地址空间;
- 每一个I/O设备占用存储器空间的一个地址;
- CPU利用lw、sw等指令对I/O设备的管理;
- CPU利用**存储器读写信号**对I/O设备进行读写控制。



存储器映像I/O寻址 具体方案

- **高位地址参与译码的方法**

  产生存储器读写信号和I/O接口读写信号



如，dmem的空间调整为：

```
logic [31:0] RAM [255:0]; // 8位[0-FF]地址空间
```

DataAdr[3:2]

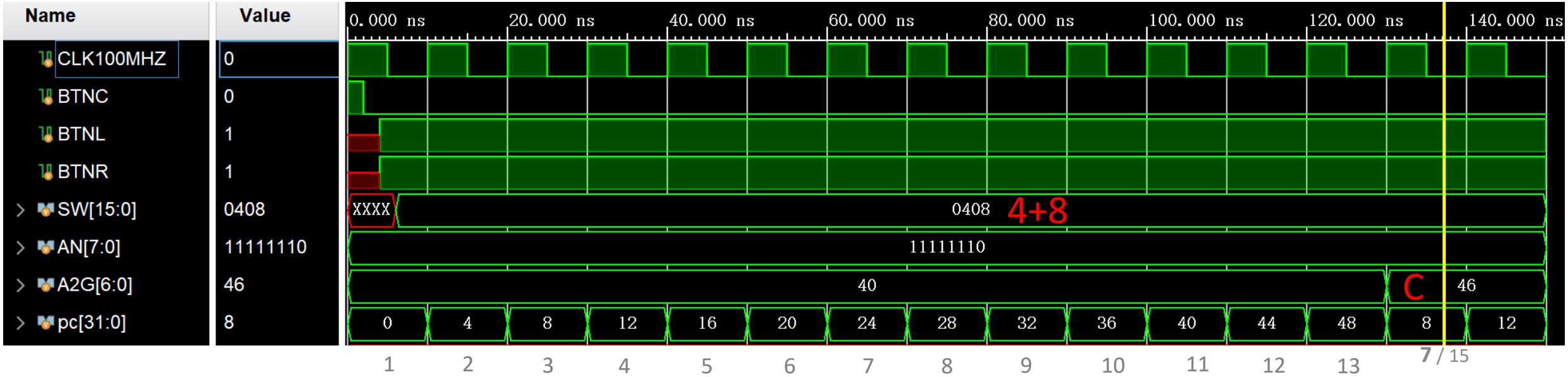
Switch端口高 [7:0]	0x8C (1000 11 00)$_2$	数据端口
Switch端口低 [7:0]	0x88 (1000 10 00)$_2$	
led端口 [11:0]	0x84 (1000 01 00)$_2$	
状态端口 [1:0]	0x80 (1000 00 00)$_2$	状态端口

```
//Whether reading from IO is enabled.
assign pRead  = (addr[7] == 1'b1) ? 1 : 0; //0x80
//Whether writing to IO is enabled.
assign pWrite = (addr[7] == 1'b1) ? writeEN : 0;
//写入数据存储器的开关
assign mWrite = writeEN & (addr[7] == 1'b0);
```

NEXYS4 DDR

31 28	27 24	23 20	19 16	15 12	11 8	7 4	3 0
0	4	0	8	=	0	0	C

8个SW[15:8]　8个SW[7:0]　12位led[11:0]



CPU查询方式I/O输入输出

增加两个I/O设备：16位开关输入、3个七段数码管(led[11:0])加法结果输出

- 按下BTNR：开关已拨好，可输入新数据， status[1]=1
- 按下BTNL：led已准备好，可输出新数据，status[0]=1

【注】实际上，这种设备是不需要查询状态就可以直接输入输出。

Switch端口高 [7:0]	0x8C $(1000\,11\,00)_2$
Switch端口低 [7:0]	0x88 $(1000\,10\,00)_2$
led端口 [11:0]	0x84 $(1000\,01\,00)_2$
状态端口 [1:0]	0x80 $(1000\,00\,00)_2$

- **数据端口**：

  2个输入数据端口、

  1个输出数据端口。

- **状态端口**： 1—准备好

switch	LEDs
status[1]	status[0]

两个16进制数相加的测试汇编代码：

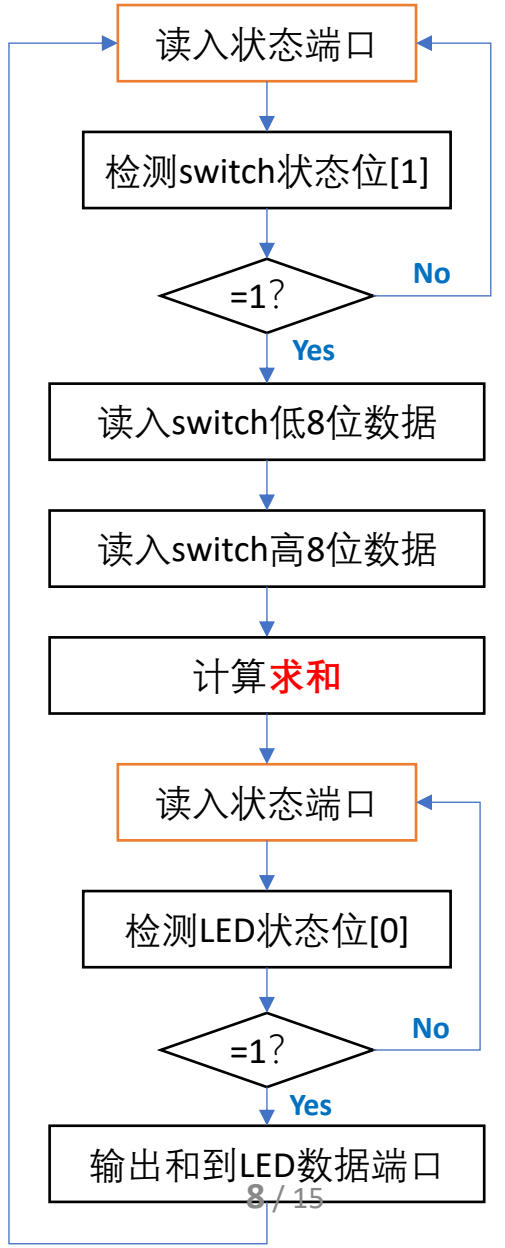
```
main:
    addi $s0, $0, 0
    sw   $s0, 0x80($0)
chkSwitch:
    lw   $s1, 0x80($0)
    andi $s2, $s1, 0x2
    beq  $s2, $0, chkSwitch
    lw   $s3, 0x88($0)
    lw   $s4, 0x8C($0)
    add  $s5, $s4, $s3
chkLED:
    lw   $s1, 0x80($0)
    andi $s2, $s1, 0x1
    beq  $s2, $0, chkLED
    sw   $s5, 0x84($0)
    j    chkSwitch
```

【注】还需要扩展andi指令

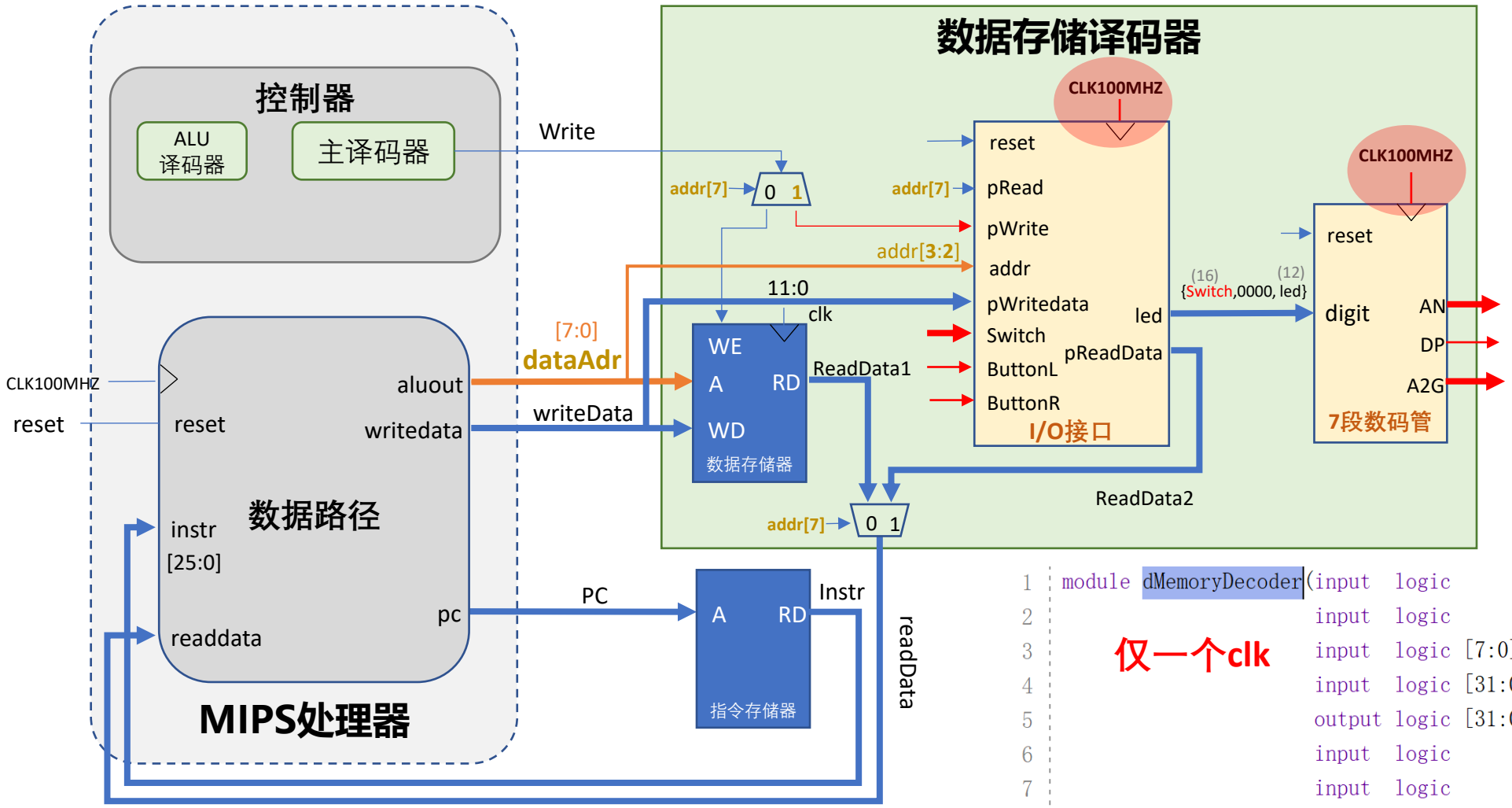
```
20100000
ac100080
8c110080
32320002
1240fffd
8c130088
8c14008c
0293a820
8c110080
32320001
1240fffd
ac150084
08000002
```

仿真代码

```
1  `timescale 1ns / 1ps
2  module testbench( );
3      logic CLK100MHZ,BTNC;//reset
4      logic BTNL,BTNR;
5      logic [15:0] SW;
6      logic [7:0]  AN;
7      logic [6:0]  A2G;
8
9      // instantiate device to be tested
10     Top T(CLK100MHZ, BTNC, BTNL, BTNR, SW, AN, A2G);
11
12     // initialize test
13     initial    begin
14         #0; BTNC <= 1;
15         #2; BTNC <= 0;
16         #2; BTNL <= 1; BTNR <= 1;
17         #2; SW<=16'b00000100_00001000;//4+8
18     end
19
20     // generate clock to sequence tests
21     always    begin
22         CLK100MHZ <= 1; # 5; CLK100MHZ <= 0; # 5;  end
23 endmodule
```



增加I/O接口的MIPS单周期处理器




```
1  module dMemoryDecoder(input  logic         clk,
2                        input  logic         writeEN,
3                        input  logic [7:0]   addr,
4                        input  logic [31:0]  writeData,
5                        output logic [31:0]  readData,
6                        input  logic         reset,
7                        input  logic         btnL,
8                        input  logic         btnR,
9                        input  logic [15:0]  switch,
10                       output logic [7:0]   an,
11                       output logic [6:0]   a2g );
```

仅一个clk

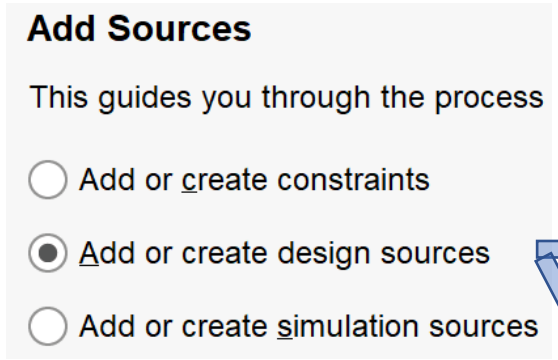
代码整体结构

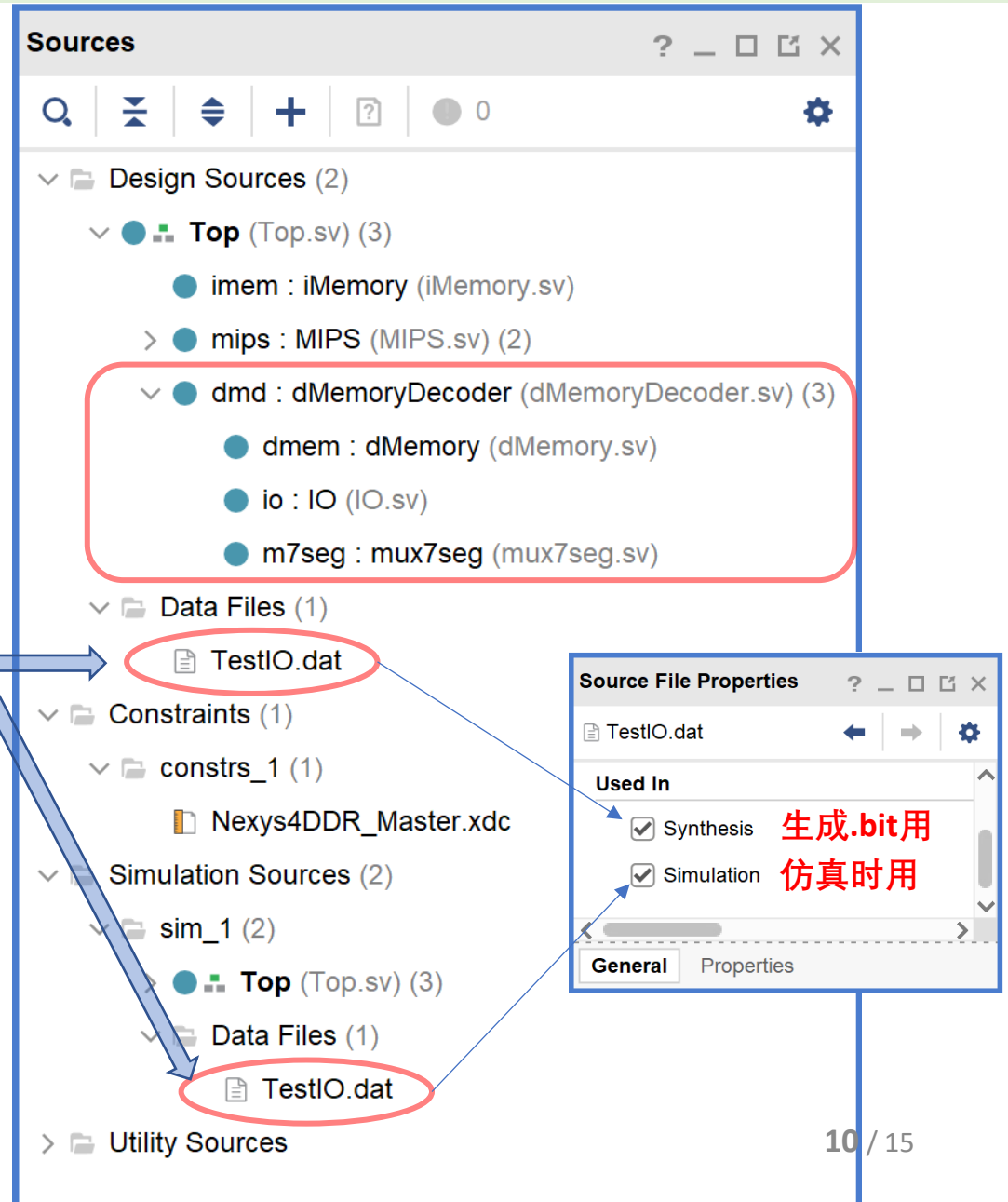
```
1  module Top(input  logic CLK100MHZ,
2             input  logic BTNC,        //reset
3             input  logic BTNL,        //SW input data
4             input  logic BTNR,        //七段数码管显示
5             input  logic [15:0] SW,   //a:SW[15:8], b:SW[7:0]
6             output logic [7:0]  AN,
7             output logic [6:0]  A2G );
8
9      logic [31:0] pc, instr;
10     iMemory imem(.a(pc[7:2]),
11                  .rd(instr)); // output
12
13     logic Write;   //写信号：可能是memWrite，也可能是ioWrite
14     logic [31:0] dataAdr, writeData, readData;
15
16     MIPS mips(.clk(CLK100MHZ),
17               .reset(BTNC),
18               .pc(pc),                 // output
19               .instr(instr),
20               .memwrite(Write),        // output
21               .aluout(dataAdr),        // output
22               .writedata(writeData), // output
23               .readdata(readData));
24
25     dMemoryDecoder dmd(.clk(CLK100MHZ),
26                        .writeEN(Write),
27                        .addr(dataAdr[7:0]),
28                        .writeData(writeData),
29                        .readData(readData), //output
30                        .reset(BTNC),
31                        .btnL(BTNL),
32                        .btnR(BTNR),
33                        .switch(SW),
34                        .an(AN),              //output
35                        .a2g(A2G) );          //output
36 endmodule
```

这样写

不宜错

导入方式：





I/O接口

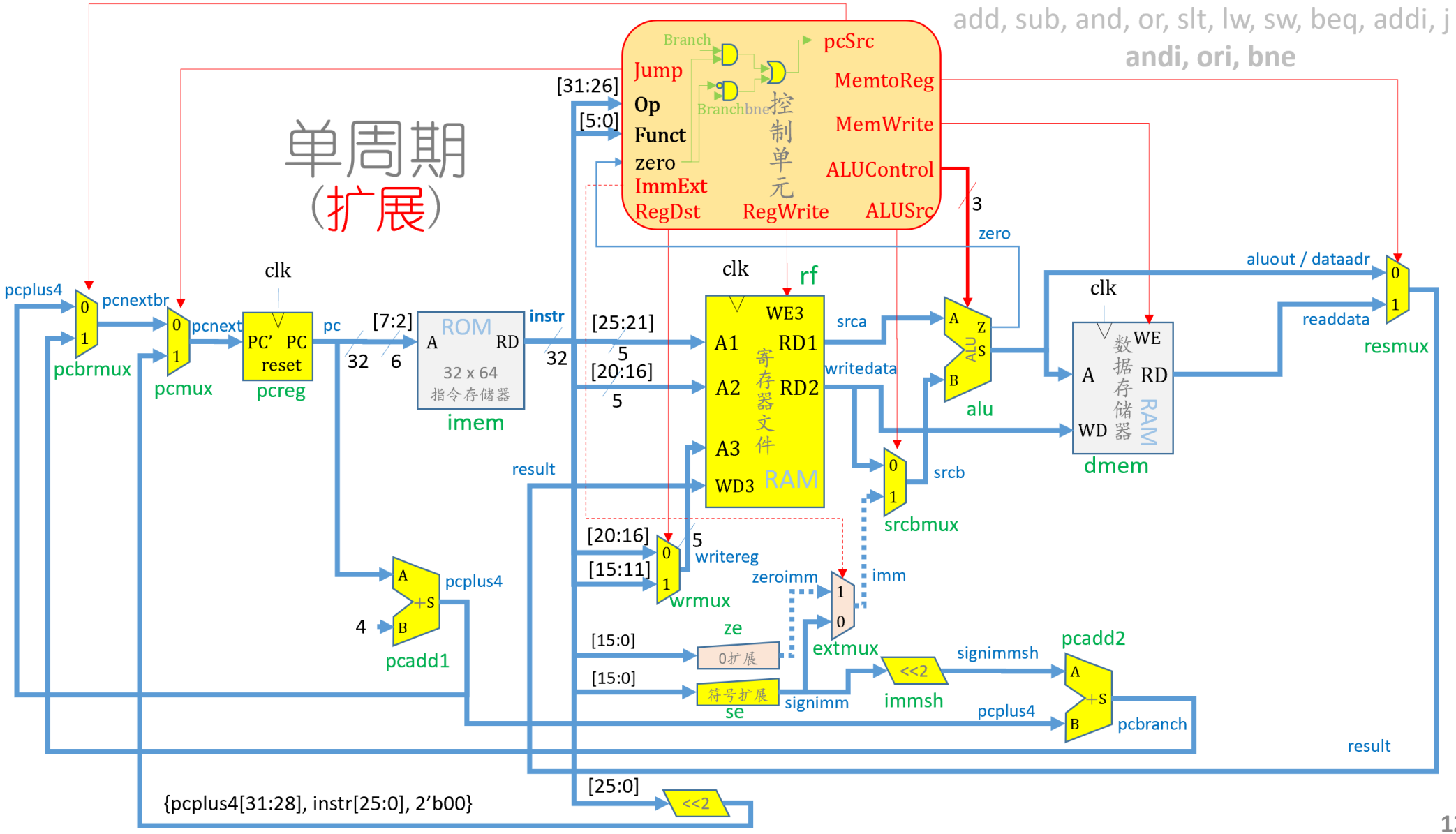
```
module IO(input  logic        clk,
          input  logic        reset,
          input  logic        pRead,
          input  logic        pWrite,
          input  logic [1:0]  addr,
          input  logic [11:0] pWriteData,
          output logic [31:0] pReadData,
          //上面与CPU相连，下面与外设相连
          input  logic        buttonL, //led输出
          input  logic        buttonR, //Switch输入
          input  logic [15:0] switch,  //直接显示
          output logic [11:0] led );

    logic [1:0]  status;
    logic [15:0] switch1;
    logic [11:0] led1;
    always_ff @(posedge clk) begin
        if (reset) begin
            status  <=  2'b00;
            led1    <= 12'h00;
            switch1 <= 16'h00;
        end
        else  begin
            //开关位置已经拨好，可以输入新数据
            if(buttonR) begin
                status[1] <= 1;
                switch1   <= switch;
            end
            //led显示已经准备好，可以输出新数据
            if(buttonL) begin
                status[0] <= 1;
                led       <= led1;
            end

            // 向数据输出端口输出(led)
            if(pWrite & (addr==2'b01)) begin
                led1      <= pWriteData;
                status[0] <= 0;
            end

        end //if
    end   //always_ff

    // 读数据
    always_comb
        if(pRead)
            // 11:数据输入端口(高),    10:数据输入端口(低)
            // 01:数据输出端口(led),   00:状态端口
            case(addr)
                2'b11:   pReadData = {24'b0, switch1[15:8]};
                2'b10:   pReadData = {24'b0, switch1[7:0]};
                2'b00:   pReadData = {24'b0, 6'b0, status};
                default: pReadData =  32'b0;
            endcase
        else
            pReadData =  32'b0;
endmodule
```

与CPU相连

与外设之间的连线

Switch端口高 [7:0]	0x8C $(1000\,11\,00)_2$
Switch端口低 [7:0]	0x88 $(1000\,10\,00)_2$
led端口 [11:0]	0x84 $(1000\,01\,00)_2$
状态端口 [1:0]	0x80 $(1000\,00\,00)_2$

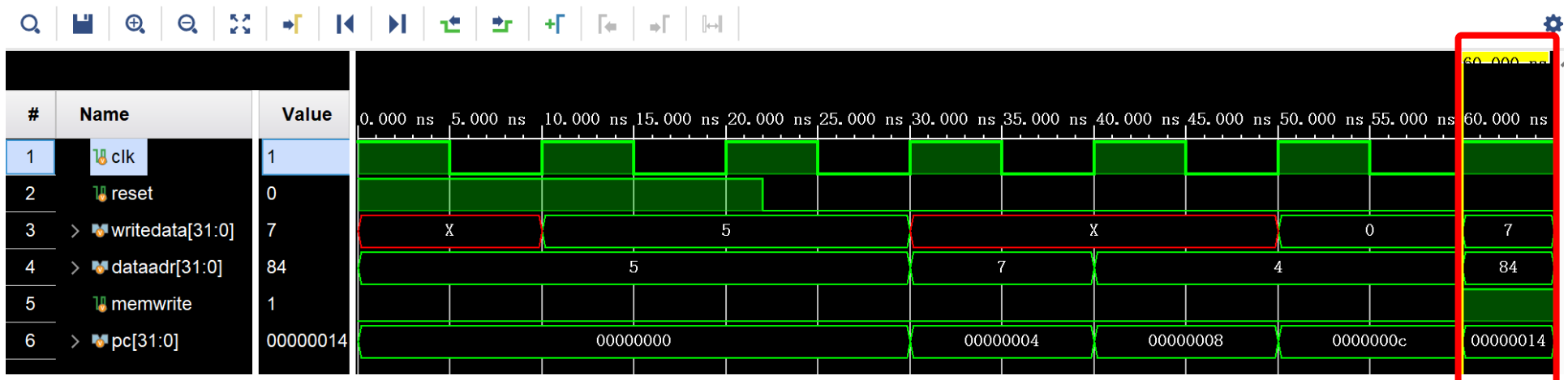
增加 andi、ori、bne 指令



主译码器真值表

指令	Opcode	RegWrite	RegDst	ALUSrc	Branch	MemWrite	MemtoReg	Jump	ALUOp	ImmExt	Branch Bne
R类型	000000	1	1	0	0	0	0	0	010	0	0
lw	100011	1	0	1	0	0	1	0	000	0	0
sw	101011	0	x	1	0	1	x	0	000	0	0
beq	000100	0	x	0	1	0	x	0	001	0	0
addi	001000	1	0	1	0	0	0	0	000	0	0
j	000010	0	x	x	x	0	x	1	xx	0	0
bne	000101	0	0	0	0	0	0	0	001	0	1
ori	001101	1	0	1	0	0	0	0	011	1	0
andi	001100	1	0	1	0	0	0	0	100	1	0

```
1  # Test the MIPS-Ext process
2  # andi, ori, bne
3  # if successful, it should
4  main:   addi $2, $0, 5
5          ori  $4, $2, 7
6          andi $5, $2, 4
7          bne  $5, $0, end
8          addi $4, $0, 1
9  end:    sw $4, 84($0)
```



测试汇编指令 + 仿真代码

测试汇编代码

```
# Test the MIPS-Ext processor
# andi, ori, bne
# if successful, it should write the value 7 to address 84
main:   addi $2, $0, 5      # initialize $2 = 5
        ori  $4, $2, 7      # 101 or  111 = 111 ($4)
        andi $5, $2, 4      # 101 and 100 = 100 ($5)
        bne  $5, $0, end    # should be taken
        addi $4, $0, 1      # shouldn't happen $4 = 1
end:    sw $4, 84($0)       # write mem[84] = 7
```

测试机器代码

memfileExt.txt

```
20020005
34440007
30450004
14a00001
20040001
ac040054
```

仿真代码

```
module testbench();
  logic        clk;
  logic        reset;
  logic [31:0] writedata, dataadr;
  logic        memwrite;

  // instantiate device to be tested
  top dut(clk, reset, writedata, dataadr, memwrite);

  // initialize test
  initial   begin
      reset <= 1; # 22; reset <= 0;
    end

  // generate clock to sequence tests
  always
    begin
      clk <= 1; # 5; clk <= 0; # 5;
    end

  // check that 7 gets written to address 84
  always@(negedge clk)
    begin
      if(memwrite) begin
        if(dataadr === 84 & writedata === 7) begin
          $display("Simulation succeeded");
          $stop;
        end else if (dataadr !== 80) begin
          $display("Simulation failed");
          $stop;
        end
      end
    end
endmodule
```

效果图

单击**中间按钮**(系统清零)，再单击**左按钮**(LED输出)，则**清零**；
单击**右侧按钮**(开关输入)，再单击**左按钮**(LED输出)，显示**相加**结果。