中国科学技术大学计算机学院

《计算机组成与原理》报告

实验题目：运算器与排序

学生姓名：雷雨轩

学生学号：PB18111791

完成日期：2020-4-28

【实验目的】

- 掌握算术逻辑单元（ALU）的功能，加/减运算时溢出、进位/借位、零标志的形成及其应用；
- 掌握数据通路和控制器的设计和描述方法。

【逻辑设计分析与代码】

ALU的设计:

- 理解 reg或wire类型的数据(以无符号方式看其值，但以补码的规则进行加减运算) 相加或者相减的操作具体实现(a,b两数): 相加即竖式相加，a-b即a的二进制加上b的二进制取反加1所得数，所以进位借位可通过{cf,y}=a+b一目了然。

  若认为a,b是无符号数(不用考虑溢出位），相加的话则只用看进位即可；相减若a<b则会有cf=1，而且a-b后所得y即为a-b十进制负数结果的补码表示（eg：4(4'b0100)-6(4'b0110)=-2，所以y=4'b1110)

  若认为a,b是有符号数，相加的话: 两正数加后y最高位为1则溢出，或两负数加后最高位为0则溢出，其他情况则没溢出； 相减的话: 正-负后y最高位为1则溢出，负-正后最高位为0则溢出，其余情况没溢出

- 归约或： 如果存在位值为1，那么结果为1；如果存在位x或z，结果为x；否则结果为0(即从左到右每两位或运算所得结果再与下一位求或)

  归约或非: 与归约或结果相反

- 对于标志位: 若其可取任意值，则都默认取0；此外，为了更好地实现sort，alu中增加了sf标志位(结果最高位为1时置1，否则置0)

- 代码及注释

```
module alu
    #(parameter WIDTH=32)    //数据宽度
    (output reg [WIDTH-1:0]y,    //运算结果
    output reg zf,               //零标志
    output reg cf,               //进位/借位标志
    output reg of,               //溢出标志(针对有符号数时的判断)
    output reg sf,               //最高位结果标志，结果最高位为1则sf置1，否则置0
    input [WIDTH-1:0] a,b,     //两操作数
    input [2:0] m             //操作类型
    );
//若某个标志位可以为任意值x,则将其赋值为0
```

```
always@(*)
begin
    case(m)
        3'b000:
        begin
            {cf,y} = a+b;
            //溢出位of的分析参见ppt和报告(ALU设计)
            of = (~a[WIDTH-1] & ~b[WIDTH-1] & y[WIDTH-1])|(a[WIDTH-1] &
b[WIDTH-1]                  & ~y[WIDTH-1]);
            zf = ~| y;
            sf = y[WIDTH-1];
        end
        3'b001:
        begin
            {cf,y} = a-b;
            //溢出位of的分析参见ppt和报告(ALU设计)
            of = (~a[WIDTH-1] & b[WIDTH-1] & y[WIDTH-1])|(a[WIDTH-1] &
~b[WIDTH-1]                   & ~y[WIDTH-1]);
            zf = ~| y;
            sf = y[WIDTH-1];
        end
        3'b010:
        begin
            y=a&b;
            cf=0; of=0; zf=~|y;
            sf = y[WIDTH-1];
        end
        3'b011:
        begin
            y=a|b;
            cf=0; of=0; zf=~|y;
            sf = y[WIDTH-1];
        end
        3'b100:
        begin
            y=a^b;
            cf=0; of=0; zf=~|y;
            sf = y[WIDTH-1];
        end
        default://此时所有结果全置为0
        begin
            y=0;
            cf=0; of=0; zf=0;
            sf = 0;
        end
    endcase
end
endmodule
```

排序电路的设计

- 实验的关键是画出数据通路的逻辑框图，再写出控制器的状态图；

  与无符号数比较时的区别在于，通过alu计算X-Y来判断X与Y大小须通过 of,sf,zf三个标志位，所以相应的en0-3四个信号需据此做调整；

有符号数比较

X与Y关系	X-Y后标志 OF SF ZF
X=Y	0 0 1
X>Y	0 0 0 1 1 0
X<Y	0 1 0 1 0 0

参见上图分析:

若X>Y,则X-Y有两种可能:

- X为正且Y为负，并且所得结果溢出(of=1)，最高位为1(sf=1),zf=0
- 结果未溢出，则结果应为4位有符号正数，of=0,sf=0,zf=0

若X<Y，则X-Y也有两种可能

- X为负且Y为正，并且所得结果溢出(of=1)，最高位为0(sf=0),zf=0
- 结果未溢出，结果应为负数，of=0,sf=1,zf=0

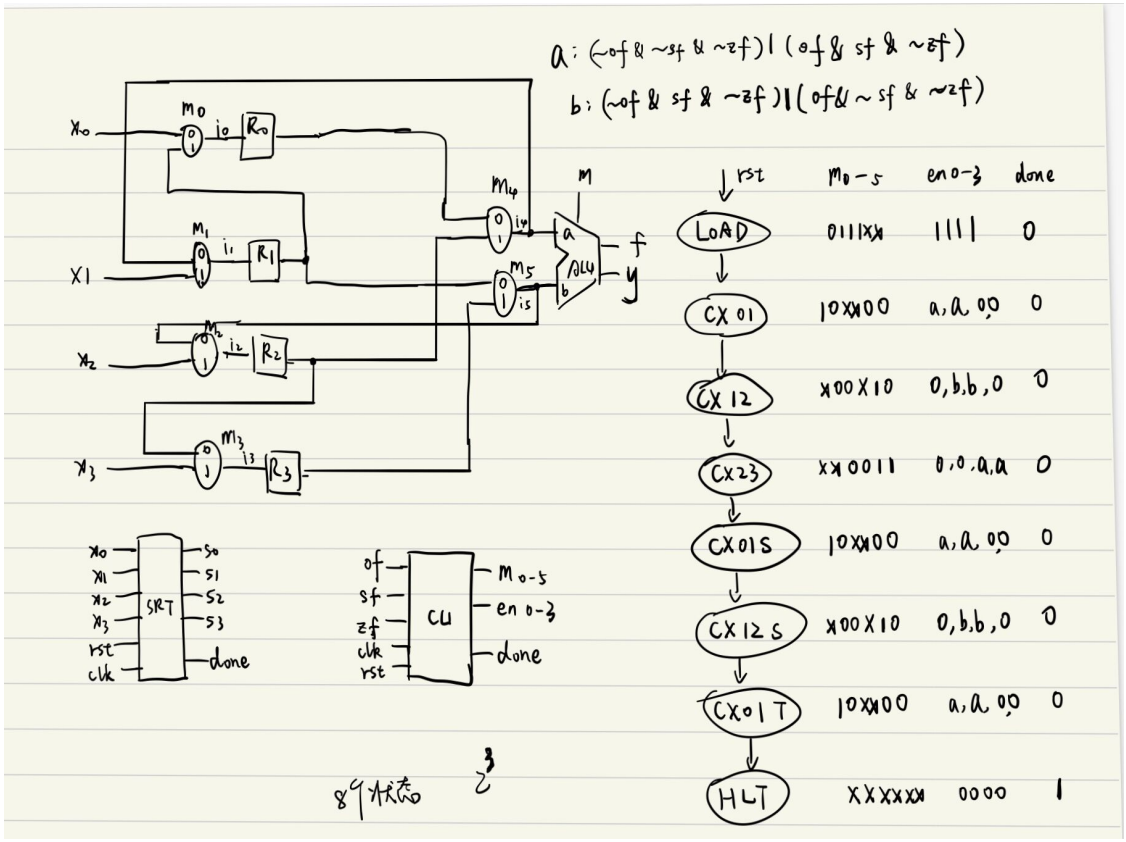
据此可设置en0-3的值

因为是四个数比较，所以需4个寄存器存数，6个二选一mux；比较次数则是第一轮3次比较；第二轮两次比较；第三轮一次比较；比较完毕后排序结束，控制器进入HLT状态保持，done信号置1，此时s0-3输出即为排序完成后的数据;直到遇到rst复位信号，开始下一轮排序

- 数据通路逻辑框图及控制器状态图、逻辑符号

  m0-5是mux选择信号，i0-5是中间信号，r0-r3是寄存器输出，en0-3是寄存器使能信号

  要理解整个排序电路工作流程:比如在LOAD状态时，设置好相应m0-5和en0-3，此时数据通路中的数据已经在线上了，当时钟上升沿来到时，状态变为CX01的同时寄存器也更新为初始数据，然后在下一时钟周期上升沿到来之前设置好m0-5和en0-3，把相关数据放到线上；当上升沿又来临时，状态变为CX12，寄存器再次更新，这里的更新则是根据CX01状态下的m和en信号；此时，在状态CX12下，寄存器里的值即为0,1号数据比较并交换后的结果，此时再设置m和en信号，依次类推



- 代码及注释

```
module sort
    #(localparam N = 4)           //数据宽度
    (output [N-1:0] s0,s1,s2,s3,     //排序后四个数据(递增)
    output reg done,              //排序结束标志
    input [N-1:0] x0,x1,x2,x3,     //原始输入数据
    input clk,rst            //时钟(上升沿有效)，复位(高电平有效)
    );

parameter SUB=3'b001;//alu的操作信号
//状态图的8个状态
parameter LOAD=3'b000,
CX01=3'b001,CX12=3'b010,CX23=3'b011,CX01S=3'b100,CX12S=3'b101,CX01T=3'b110,H
LT=3'b111;
//中间信号，i0-i5是对应mux2的输出，对应寄存器的输入;r0-3是对应寄存器输出
wire [3:0] i0,i1,i2,i3,i4,i5,r0,r1,r2,r3;
//en0-3为寄存器使能信号，m0-5为mux2选择信号
reg en0,en1,en2,en3,m0,m1,m2,m3,m4,m5;
//标志位
wire of,zf,sf;

reg [2:0] current_state,next_state;

//DATA Path,模块实例化参加报告中的数据通路逻辑框图
register #(.WIDTH(4)) R0
    (.clk(clk),
    .rst(rst),
    .en(en0),
    .d(i0),
    .q(r0));
register #(.WIDTH(4)) R1
    (.clk(clk),
    .rst(rst),
```

```
    .en(en1),
    .d(i1),
    .q(r1));
register #(.WIDTH(4)) R2
    (.clk(clk),
    .rst(rst),
    .en(en2),
    .d(i2),
    .q(r2));
register #(.WIDTH(4)) R3
    (.clk(clk),
    .rst(rst),
    .en(en3),
    .d(i3),
    .q(r3));
alu #(.WIDTH(4)) ALU
    (.y(),
    .zf(zf),
    .cf(),
    .of(of),
    .sf(sf),
    .a(i4),
    .b(i5),
    .m(SUB));
mux2 #(.WIDTH(4)) M0
    (.y(i0),
    .a(x0),
    .b(r1),
    .s(m0));
mux2 #(.WIDTH(4)) M1
    (.y(i1),
    .a(i4),
    .b(x1),
    .s(m1));
mux2 #(.WIDTH(4)) M2
    (.y(i2),
    .a(i5),
    .b(x2),
    .s(m2));
mux2 #(.WIDTH(4)) M3
    (.y(i3),
    .a(r2),
    .b(x3),
    .s(m3));
mux2 #(.WIDTH(4)) M4
    (.y(i4),
    .a(r0),
    .b(r2),
    .s(m4));
mux2 #(.WIDTH(4)) M5
    (.y(i5),
    .a(r1),
    .b(r3),
    .s(m5));


//Control Unit
always@(posedge clk,posedge rst)
```

```
    if(rst) current_state<=LOAD;
    else
        current_state <= next_state;

always@(*) //状态变换规则
begin
    case(current_state)
        LOAD:next_state=CX01;
        CX01:next_state=CX12;
        CX12:next_state=CX23;
        CX23:next_state=CX01S;
        CX01S:next_state=CX12S;
        CX12S:next_state=CX01T;
        CX01T:next_state=HLT;
        HLT:next_state=HLT;
        default:next_state=HLT;
    endcase
end

always@(*)//控制信号及组合逻辑输出
begin
    {m0,m1,m2,m3,m4,m5,en0,en1,en2,en3,done}=11'b0;
    case(current_state)
        LOAD:{m0,m1,m2,m3,en0,en1,en2,en3}=8'b01111111;
        CX01:
        begin
            {m0,m1,m4,m5}=4'b1000;
            en0=(~of & ~sf & ~zf)|(of & sf & ~zf);
            en1=(~of & ~sf & ~zf)|(of & sf & ~zf);
        end
        CX12:
        begin
            {m1,m2,m4,m5}=4'b0010;
            en1=(~of & sf & ~zf)|(of & ~sf & ~zf);
            en2=(~of & sf & ~zf)|(of & ~sf & ~zf);
        end
        CX23:
        begin
            {m2,m3,m4,m5}=4'b0011;
            en2=(~of & ~sf & ~zf)|(of & sf & ~zf);
            en3=(~of & ~sf & ~zf)|(of & sf & ~zf);
        end
        CX01S:
        begin
            {m0,m1,m4,m5}=4'b1000;
            en0=(~of & ~sf & ~zf)|(of & sf & ~zf);
            en1=(~of & ~sf & ~zf)|(of & sf & ~zf);
        end
        CX12S:
        begin
            {m1,m2,m4,m5}=4'b0010;
            en1=(~of & sf & ~zf)|(of & ~sf & ~zf);
            en2=(~of & sf & ~zf)|(of & ~sf & ~zf);
        end
        CX01T:
        begin
            {m0,m1,m4,m5}=4'b1000;
            en0=(~of & ~sf & ~zf)|(of & sf & ~zf);
```

```
            en1=(~of & ~sf & ~zf)|(of & sf & ~zf);
        end
        HLT:done=1;
        default:;
    endcase
end

//s0-3输出即为寄存器里内容(done=1时表明排序完成，此时s0-3才为排序成功后的数据
assign s0=r0;
assign s1=r1;
assign s2=r2;
assign s3=r3;

endmodule
```

另附上选择器和寄存器代码

```
module mux2
    #(parameter WIDTH = 32)//数据宽度
    (output [WIDTH-1:0] y, //输出数据
    input [WIDTH-1:0] a,b, //两路输入数据
    input s                 //数据选择控制
    );

assign y=s?b:a;  //二选一: s=1,选b；s=0，选a
endmodule
```

```
module register
    #(parameter WIDTH = 32,RST_VALUE = 0)
    (input clk,rst,en,          //时钟(上升沿有效)，复位(高电平有效)，使能(高电平有效)
    input [WIDTH-1:0] d,        //输入数据
    output reg [WIDTH-1:0] q);  //输出数据

always@(posedge clk,posedge rst)
    if(rst) q<=RST_VALUE;
    else if(en)
        q<=d;

endmodule
```

【仿真结果及分析】

ALU功能仿真

- 仿真代码

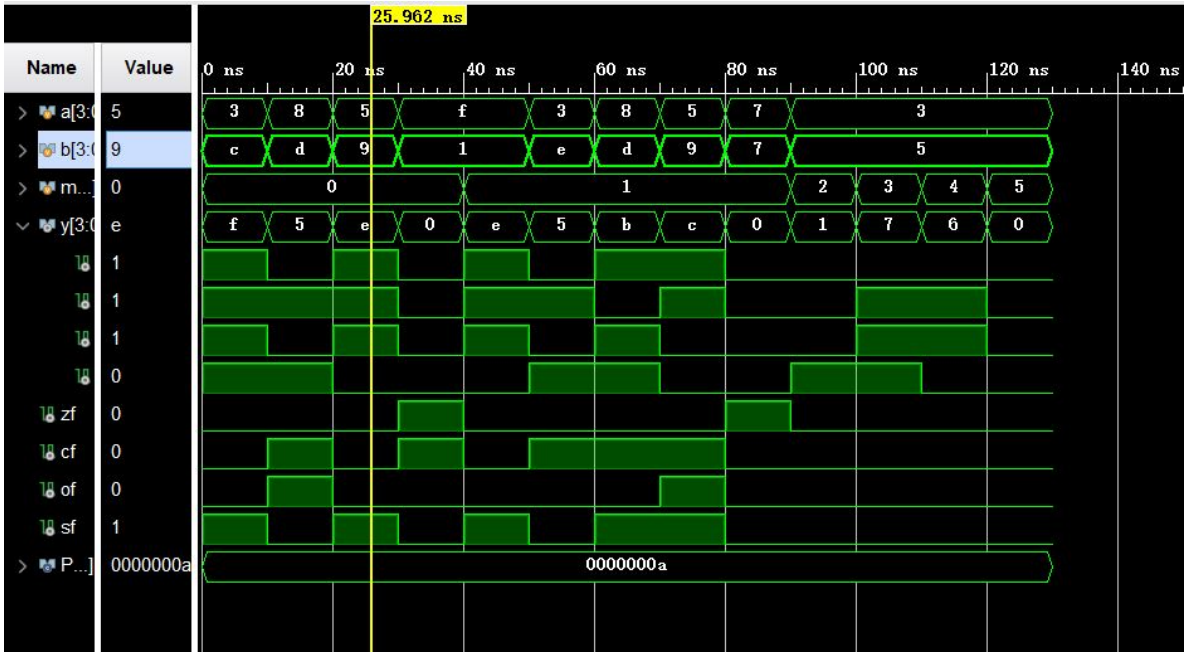
```
`timescale 1ns / 100ps

module alu_tb;
    reg [3:0] a, b; //两操作数
    reg [2:0] m;    //操作类型
    wire [3:0] y;   //运算结果
    wire zf, cf, of,sf; //零标志，进位/借位标志，溢出标志，最高位标志

    parameter PERIOD = 10;//数据变化的时间间隔
```

```
    alu #(4) ALU(.a(a), .b(b), .m(m), .y(y), .cf(cf), .of(of),
.zf(zf),.sf(sf));

    initial begin
        m = 0;              //ADD
        a = 4'b0011;
        b = 4'b1100;

        #PERIOD;
        a = 4'b1000;
        b = 4'b1101;

        #PERIOD;
        a = 4'b0101;
        b = 4'b1001;

        #PERIOD;
        a = 4'b1111;
        b = 4'b0001;

        #PERIOD m = 1;  //SUB

        #PERIOD;
        a = 4'b0011;
        b = 4'b1110;

        #PERIOD;
        a = 4'b1000;
        b = 4'b1101;

        #PERIOD;
        a = 4'b0101;
        b = 4'b1001;

        #PERIOD;
        a = 4'b0111;
        b = 4'b0111;

        #PERIOD m = 2;  //AND
        a = 4'b0011;
        b = 4'b0101;

        #PERIOD m = 3;  //OR
        #PERIOD m = 4;  //XOR
        #PERIOD m = 5;  //other
        #PERIOD;
        $finish;
    end
endmodule
```

- 数据分析: 对仿真代码里的计组数据计算后得到理论上的运算结果及标志位结果

- 仿真结果及分析:

  以黄色竖线那里为例，此时刻两操作数a=4'b0101,b=4'b1001, 运算结果y=4'b1110,标志位zf=0,cf=0,of=0,sf=1,与上面数据分析的图中ADD的第三组数据做对比，完全一致，说明仿真结果正确，其余组分析同理。



排序电路功能仿真

- 仿真代码

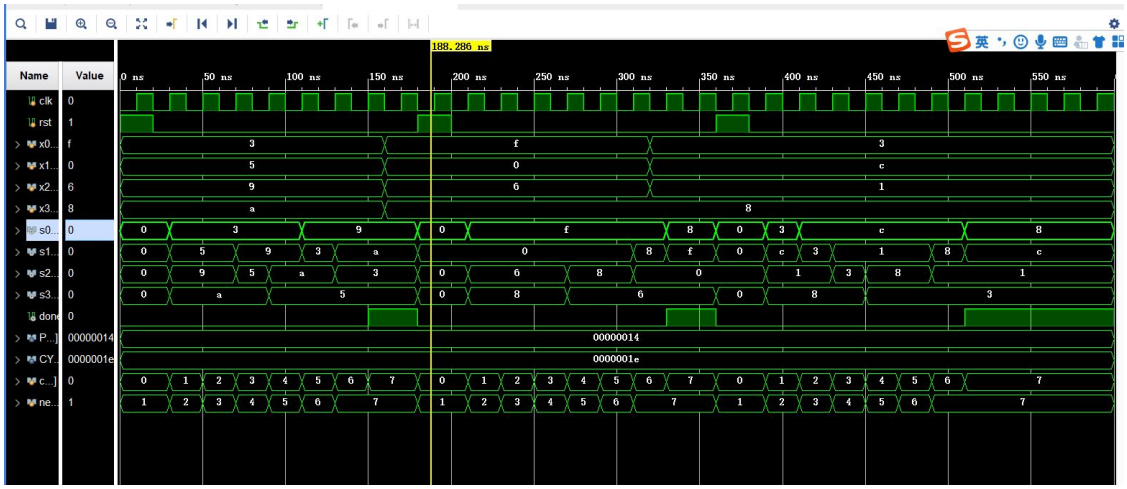
```
`timescale 1ns / 100ps

module sort_tb;
    reg clk, rst;
```

```
    reg [3:0] x0, x1, x2, x3;
    wire [3:0] s0, s1, s2, s3;
    wire done;


parameter PERIOD = 20,  //时钟周期长度
CYCLE = 30;     //时钟个数

    sort SORT(s0, s1, s2, s3, done, x0, x1, x2, x3,  clk, rst);

    initial
    begin
        clk = 0;
        repeat (2 * CYCLE)
            #(PERIOD/2) clk = ~clk;
        $finish;
    end

    initial
    begin
    rst = 1;
    #PERIOD rst = 0;

    #(PERIOD*8) rst = 1;
    #PERIOD rst = 0;

    #(PERIOD*8) rst = 1;
    #PERIOD rst = 0;
    end

    initial
    begin
    x0 = 4'b0011;//3
    x1 = 4'b0101;//5
    x2 = 4'b1001;//-7
    x3 = 4'b1010;//-6

    #(PERIOD*8);
    x0 = 4'b1111;//-1
    x1 = 4'b0000;//0
    x2 = 4'b0110;//6
    x3 = 4'b1000;//-8
    #(PERIOD*8);
    x0 = 4'b0011;//3
    x1 = 4'b1100;//-4
    x2 = 4'b0001;//1
    x3 = 4'b1000;//-8
end
endmodule
```

- 仿真结果及分析

  如下图所示，一共测试了三组数据，黄色竖线左边为第一组数据运行过程

  首先在第一个时钟周期里复位至LOAD状态，将数加载入寄存器(数据为x0=3,x1=5,x2=-7,x3=-6,参见仿真代码)，之后7个时钟周期则按照控制器状态图一步一步走(可参考下图最后两行的current_state和next_state的值)，当运行至第8个时钟周期时，排序完成，观察黄线左边得到结果为(s0=-7,s1=-6,s2=3,s3=5),为正确的排序结果。其余两组数据分析同理，结果也正确。



【下载结果及分析】

暂无

【思考题】

- 如果要求排序后的数据是递减顺序，电路如何调整?

  只需让冒泡排序过程中的比较成功条件更改一下即可，比如:R1和R2比较，若R1小，则数据交换，按照前面的控制器状态图来看，就是把a，b两个信号的表达式互换即可

- 如果为了提高性能，使用两个ALU，电路如何调整?

  四个数的冒泡排序正常情况下是有三轮比较(CX01,CX12,CX23第一轮，CX01S,CX12S为第二轮，CX01T为第三轮)，那么若有两个ALU，可考虑并行比较，减少比较轮数，从而减少排序时间:

  CX01(ALU1) -> CX12(ALU1) -> CX23(ALU1)且CX01S(ALU2) -> CX12S(ALU1) -> CX01T(ALU1) ,这样一来提高了性能，减少了状态机一个状态(尽管效果不明显，但可以发现当排序数据个数增加到5，6，甚至10时，两个ALU这样的并行操作就性能提升很明显了)

【实验总结】

通过本次实验，完整的复习了模块化、层次化的设计方法，并新学习了参数化的设计方法，重温了Vivado仿真的使用，对于两段式FSM的代码结构及其功能更加清晰。

对于ALU设计这一块，因为上学期的学习中对于运算符的忽略，导致对于Verilog里reg和wire型变量+,-操作的计算方式产生困惑，经过一番资料查询，才成功解决知识盲区，深入的理解到了溢出和进位\借位标志符的形成方式。对于归约运算符的理解也存在一定偏差，也是查阅资料后解决。

对于排序设计这一块，整体难度不大，因为有三个无符号数排序的例子，所以能比较清晰地构建出4个有符号数排序的数据通路，不过需注意通过ALU的减法来比较两数大小的判断方法有所变化，这也体现在了控制器的状态图里。在写代码时，则注意格式规范，这样模块实例很多也仍然结构清晰。

此外，通过学习数据通路和控制器的设计与描述方法，也受益匪浅。

【意见建议】

对于本次实验整体较为满意，实验完成后自我感觉所达到的效果与实验目的基本一致，有温故而知新的意味，将上学期所学知识在本次实验中重温并融会贯通，并在此基础上增加了对于数据通路和控制器、ALU的设计思想的学习。

不过感觉实验文档稍微有点混乱，建议统一为一个.pdf文件，而不是分开放为.docs和.ppt，这样不方便查阅文档。