一、　实验目的和要求

Object:

1. Familiar with the design method of the basic logic components, and master the working principle and design method of the basic components.
2. Understand the principles of Datapath and master methods of Datapath design.
3. Understand the principles of Single-cycle clock CPU and master methods of Single-cycle clock CPU design
4. master methods of program verification of CPU.

Requirement:

1. Design the Datapath, bring together the basic units into Single-cycle clock CPU.
2. Verify the CPU with program and observe the execution of program

二、　实验内容和原理

2.1 The Overall Design Graph of single-cycle CPU

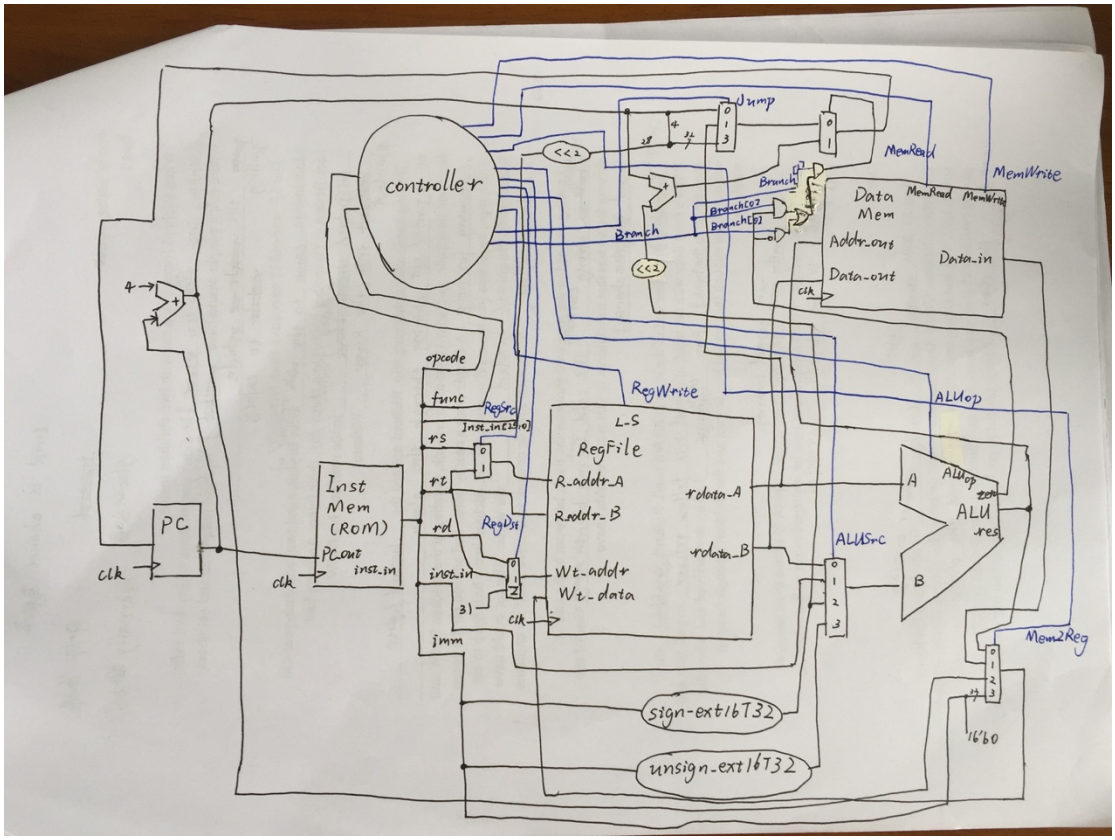

Figure 1　the Design Graph

2.2 Extended ALU

To implement 19 instructions, I have made some extension to the original ALU module.

And the Verilog code for the extended ALU module is as following:

```
module ALU(
      input[31:0] A, B,
       input[3:0] ALU_operation,
      output reg[31:0] res,
      output zero
    );

   wire  [31:0]  res_and,  res_or,  res_xor,  res_nor,
res_srl, res_add, res_sub, res_sll;
   reg [31:0] res_slt;

    always @ (*)
         case (ALU_operation)
            4'b0000: res=res_and;
            4'b0001: res=res_or;
            4'b0010: res=res_add;
      4'b0011: res=res_xor;
      4'b0100: res=res_nor;
      4'b0101: res=res_srl;
            4'b0110: res=res_sub;
      4'b0111: res=res_slt;
      4'b1000: res=res_sll;
         endcase

   assign zero = (res == 0);

   parameter one = 32'h00000001, zero_0 = 32'h00000000;

    assign res_and = A&B;
    assign res_or = A|B;
    assign res_nor = ~(A | B);
    assign res_add = A+B;
    assign res_sub = A-B;
    assign res_xor = A^B;
    assign res_srl = A >> B[10:6];
    assign res_sll = A << B[10:6];

   always @(*) begin
```

```
    if(A[31]==1 && B[31]==1) // 都是负数
      res_slt =(A < B) ? zero_0: one;
   else if(A[31]==0 && B[31]==0)
       res_slt =(A < B) ? one : zero_0;   // 都是正数
   else  // 一正一负
      res_slt =(A[31] == 1) ? one : zero_0;
   end

endmodule
```

The ALU_operation is extended to 4-bit so that the ALU module can work with more types of arithmetical operations.

2.3 Control Signals for each instruction

Table 1　the control signals of instructions

	[3:0]ALUop	[1:0]RegDst	RegSrc	[1:0]Branch	[1:0]Jump	MemRead	MemWrite	[1:0]Mem2Reg	RegWrite	[1:0]ALUSrc
add	0010	00	0	00	00	0	0	00	1	00
sub	0110	00	0	00	00	0	0	00	1	00
and	0000	00	0	00	00	0	0	00	1	00
or	0001	00	0	00	00	0	0	00	1	00
nor	0100	00	0	00	00	0	0	00	1	00
xor	0011	00	0	00	00	0	0	00	1	00
slt	0111	00	0	00	00	0	0	00	1	00
sll	1000	00	1	00	00	0	0	00	1	01
srl	0101	00	1	00	00	0	0	00	1	01
jr	x	x	0	00	01	0	0	x	0	x
lui	x	01	x	00	00	0	0	11	1	x
lw	0010	01	0	00	00	1	0	01	1	10
sw	0010	x	0	00	00	0	1	x	0	10
addi	0010	01	0	00	00	0	0	00	1	10
ori	0001	01	0	00	00	0	0	00	1	11
beq	0110	x	0	01	00	0	0	x	0	00
bne	0110	x	0	11	00	0	0	x	0	00
j	x	x	x	00	11	0	0	x	0	x
jal	x	10	x	00	11	0	0	10	1	x

And the following Verilog code will show you that how the control signals work:

```
if(RegSrc==1'b0)
         rs<=inst_in[25:21];
      else
         rs<=inst_in[20:16];
```

The “RegSrc” is the control signal for the mux, which is used to choose the input of “R_addr_A” port of the RegFile module. Most of instructions use Rs (inst_in[25:21]) as the first operand. However, sll and srl will use Rt(inst_in[20:16]) as their first

operand.

```
if(RegDst==2'b00)
         rd<=inst_in[15:11];
      else if(RegDst==2'b01)
         rd<=inst_in[20:16];
      else if(RegDst==2'b10)
         rd<=5'b11111;
```

The “RegDst” is the control signal for the mux, which is used to choose the input of “Wt_addr” port of the RegFile module. Most of instructions use Rd (inst_in[15:11]) as the destination register. Others use Rt(inst_in[20:16]). Specially, jal will write to $ra($r31).

```
if(ALUSrc==2'b00)
         ALU_inB<=RegData_outB;
      else if(ALUSrc==2'b01)
         ALU_inB<=inst_in;
      else if(ALUSrc==2'b10)
         ALU_inB<=SignExt;
      else
         ALU_inB<=({16'b0,inst_in[15:0]});
```

The “ALUSrc” is the control signal for the mux, which is used to choose the input of “B” port of the ALU module. Most of instructions use Rt (RegData_outB) as the second operand. Some use the whole instruction code, such as sll and srl. Others use sign-extended immediate, such as lw. Specially, ori will use unsign-extended immediate.

```
if(Mem2Reg==2'b00)
         RegData_in<=Addr_out;
      else if(Mem2Reg==2'b01)
         RegData_in<=Data_in;
      else if(Mem2Reg==2'b10)
         RegData_in<=PC_Add_4;
      else
         RegData_in<={inst_in[15:0],16'b0};
```

The “Mem2Reg” is the control signal for the mux, which is used to choose the input of “Wt_data” port of the RegFile module. Most of instructions use the output of ALU (also

Addr_out). lw uses the input of RAM(Data_in). jal uses PC_Add_4. lui uses imm<<16.

```
if(Jump==2'b01)
         PC<=RegData_outA;
      else if(Jump==2'b11)
         PC<=PC_Jump;
       else    if(Jump==2'b00    &&    Branch==2'b01    &&
zero==1'b1 )
          PC <=PC_Branch;
       else    if(Jump==2'b00    &&    Branch==2'b11    &&
zero==1'b0 )
          PC <=PC_Branch;
       else PC <=PC_Add_4;
   end
```

This module is used to get the next PC. If the instruction is jr, the next PC will be Rs. If the instruction is j or jal, the next PC will be {PC_Add_4[31:28], inst_in[25:0]<<2}. If the instruction is beq and Rs equals to Rt, or the instruction is bne and Rs doesn't equals to Rt, the next PC will be (PC_Add_4+sign-extended imm<<2). Otherwise, the next PC will be PC_Add_4.

三、 实验过程和数据记录及结果分析

1. Delete the Data_path module, SCPU_ctrl module and SCPU module used in the previous experiment and clean up the project file.
2. Add the copy of ALU module and RegFile module into the project.
3. Make extension to the ALU module.
4. Create the new Verilog module "SCPU".
5. Then download bit file to the SWORD board and verify the design.
6. Spend a lot of time on debugging ☹

Finally, the SWORD board behaves normally, just like the result of the last experiment.

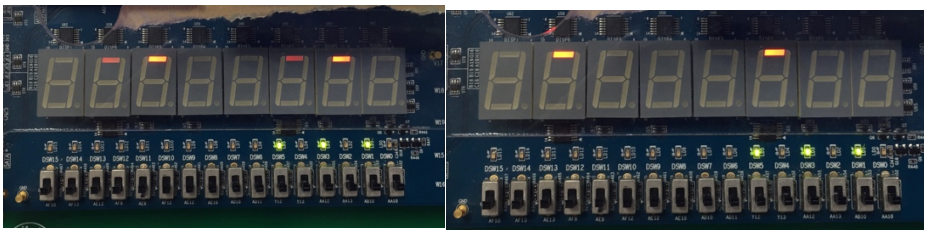

Figure 2 scroller

When SW[0] is 0, it's on the pattern mode. When SW[4:3] is 2'b00, we can see a scroller from left to right, from top to the bottom. The LED segments light row by row.

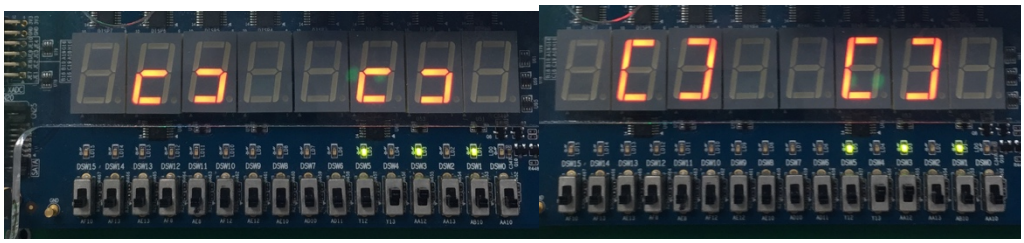

Figure 3 the changing rectangle

When SW[0] is 0 and SW[4:3] is 2'b11, we can two changing rectangles.

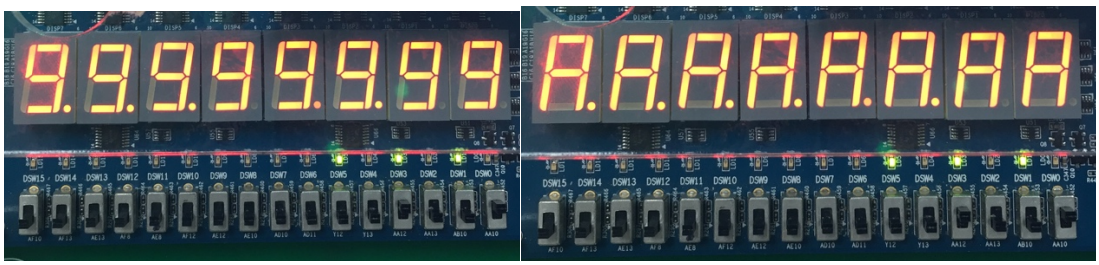

Figure 4 the content of RAM

When SW[1:0] is 2'b01 and SW[4:3] is 2'b01, we can see the content of RAM, which is written in the .coe file.

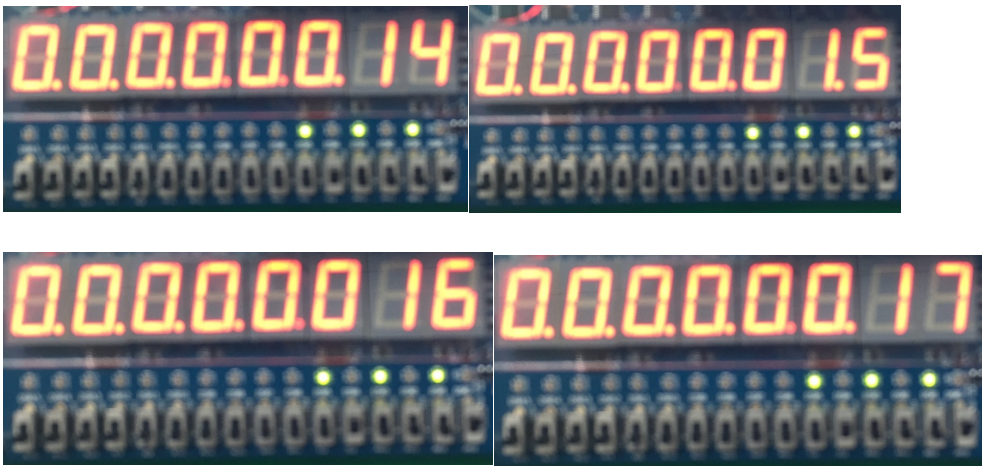

Figure 5 the counter

When SW[1:0] is 2'b01 and SW[4:3] is 2'b10, we can see the digit presented by

LED segments keeps increasing.

SW[2] controls the CPU clocks. When SW[2] is changed, the speed of pattern changing or text changing will be different.

四、 讨论与心得

It's very painful to debug in this experiment. I have spent hours and hours in the laboratory but still couldn't find out what was going wrong. When hardware is involved with it, it seems like a kind of black magic. You must be very careful about the timing. Otherwise, your project will fail even if logically it seems nothing is wrong.

And I really learn a lesson in the process of debugging:

1. Take care of PC and the next PC. Don't mix them up.
2. in the "always@" clause, use <= instead of =.
3. use ({xxx, xxx}) instead of {xxx, xxx}.
4. don't use nested if-else clause, just write if(xxx && xxx)

Believe or not, the ISE will regard some of my control signals as constant and ignore them if I don't follow the above rules.