合肥工業大學

软件学院

《计算机系统基础》

实验五—单周期 CPU 设计与实现报告

姓名：刘铭源

学号：2018214937

班级：软件工程 18-4 班

指导老师：安鑫

时间：2020.7.4 日

注：使用说明

1.将文件压缩包解压，然后使用 modelsim 打开工程文件即可运行

2.使用了循环语句需要使用 stop 语句暂停才可以看到结果

3.可以修改主存 memory 的代码初始化主存原始数据：

```
data[10'b0000000000] = 16'h0004;
data[10'b0000000001] = 16'h0015;
data[10'b0000000010] = 16'h0016;
```

4.可以修改 IRStorage 文件代码修改运行指令原指令：

```
data[0] = 16'b0010000000000001;  // LDA 00000 00001
data[1] = 16'b0010000000000010;  // LDA 00000 00010
data[2] = 16'b0001110000000100;  // STA 00000 00100
data[3] = 16'b0010000000000001;  // LDA 00000 00001
data[4] = 16'b0010000000000100;  // LDA 00000 00100
data[5] = 16'b0000010000000000;  // CLA
data[6] = 16'b0010000000000100;  // LDA 00000 00100
data[7] = 16'b0000100000000000;  // COM
data[8] = 16'b0000110000000000;  // SHR
data[9] = 16'b0001000000000000;  // CSL
data[10]= 16'b0010000000000001;  // LDA 00000 00001
data[11]= 16'b0001100000000010;  // ADD
data[12]= 16'b0010010000001110;  // JMP
data[13]= 16'b0010000000000001;  // LDA 00000 00001
data[14]= 16'b0010000000000010;  // LDA 00000 00010
data[15]= 16'b0000100000000000;  // COM
data[16]= 16'b0010000000000010;  // BAN 00000 00010
data[17]= 16'b0010000000000001;  // LDA 00000 00001
data[18]= 16'b0010000000000100;  // LDA 00000 00100
```

5.根据运行的工程文件开始观看波形图去查看，验证数据的正确性。

一、实验目的

通过设计并实现支持 10 条指令的 CPU，进一步理解和掌握 CPU 设计得基本原理和过程。

二、实验内容

设计和实现一个支持如下十条指令的单周期 CPU。

1.非访存指令

A.清除累加器指令 CLA

B.累加器取反指令 COM

C.算术右移一位指令 SHR：将累加器 ACC 中的数右移一位，结果放回 ACC

D.循环左移一位指令 CSL：对累加器中的数据进行操作

E.停机指令 STP

2.访存指令

A.加法指令 ADD X：[X] + [ACC] –〉ACC，X 为存储器地址，直接寻址

B.存数指令 STA X，采用直接寻址方式

C.取数指令 LDA X，采用直接寻址

3.转移类指令

A.无条件转移指令 JMP imm：signExt(imm) -> PC。

B.有条件转移（负则转）指令 BAN X: ACC 最高位为 1 则（PC）+ X -> PC,否则 PC 不变。

三、实验原理

1.指令格式约定

指令长度为 16 位。

存储字长为 16 位。

机器字长为 16 位。

所有的指令的操作码为前 6 位。

地址长度为 10 位，均为直接寻址。

2.根据功能和格式完成 CPU 的数据通路设计。

（1）需要实现的指令

A.清除累加器指令 CLA

操作码为 000 001

B.累加器取反指令 COM

操作码为 000 010

C.算术右移一位指令 SHR

操作码为 000 011

D.循环左移一位指令 CSL

操作码为 000 100 E.停机指令 STP 操作码为 000 101

E.加法指令 ADD X

操作码为 000 110

F.存数指令 STA X

操作码为 000 111，X 为 10 位的地址，位于指令的后 10 位

G.取数指令 LDA X

操作码为 001 000，X 为 10 位的地址，位于指令的后 10 位

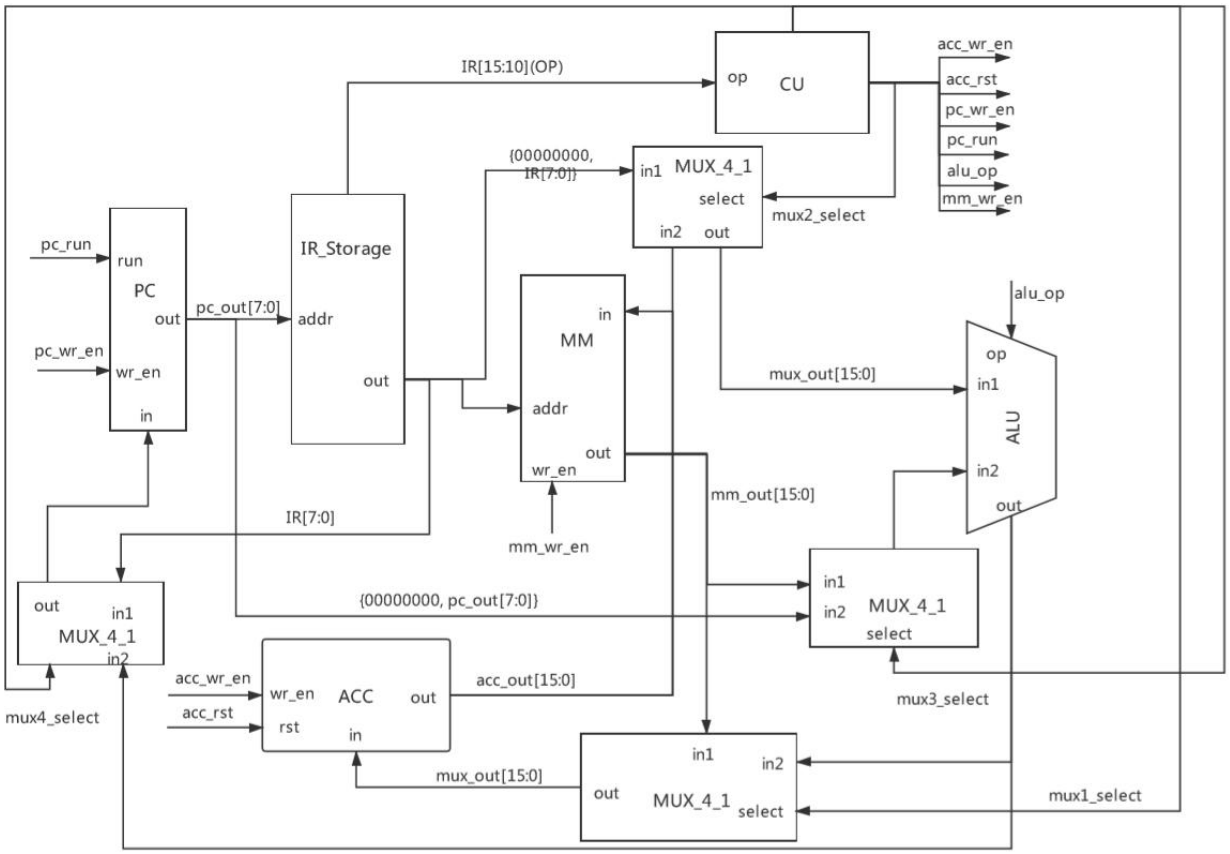
H.无条件转移指令 JMP imm

操作码为 001 001，imm 为 10 位的立即数，位于指令的后 10 位

I.有条件转移（负则转）指令 BAN imm

操作码为 001 010，imm 为 10 位的立即数，位于指令的后 10 位

（2）CPU 原理图



四、实验步骤

1.CPU 各个部门的代码

（1）PC 模块：当 PC 模块的 run 为高电平时，在每个时钟信号的下沿会自动加 1，还可以通过 rst 信号来将 pc 置 0

```
module pc_8b(
    input wire clk, rst, run, wr_en,
    input wire [7:0] in,
    output wire [7:0] out
);
    reg [7:0] pc;
    initial begin
        pc = 8'b00000000;
    end
    assign out = pc;

    always@ (negedge clk) begin
        case (rst)
```

```
            1'b0:
            begin
               case (wr_en)
                  1'b0: pc = pc + run;
                  1'b1: pc = in;
               endcase
            end
            0'b1: pc = 0;
            default: pc = 0;
         endcase
      end
   endmodule
```

（2）指令存储模块：用于存放待执行的指令，通过 addr 来获取对应的指令

```
   module IRStorage_16b(
      input wire [7:0] addr,
      output wire [15:0] out
   );
      reg [15:0] data [1023:0];
      initial begin
         data[0] = 16'b0010000000000001;    // LDA 00000 00001
         data[1] = 16'b0010000000000010;    // LDA 00000 00010
         data[2] = 16'b0001110000000100;    // STA 00000 00100
         data[3] = 16'b0010000000000001;    // LDA 00000 00001
         data[4] = 16'b0010000000000100;    // LDA 00000 00100
         data[5] = 16'b0000010000000000;    // CLA
         data[6] = 16'b0010000000000100;    // LDA 00000 00100
         data[7] = 16'b0000100000000000;    // COM
         data[8] = 16'b0000110000000000;    // SHR
         data[9] = 16'b0001000000000000;    // CSL
         data[10]= 16'b0010000000000001;    // LDA 00000 00001
         data[11]= 16'b0001100000000010;    // ADD
         data[12]= 16'b0010010000001110;    // JMP
         data[13]= 16'b0010000000000001;    // LDA 00000 00001
         data[14]= 16'b0010000000000010;    // LDA 00000 00010
         data[15]= 16'b0000100000000000;    // COM
         data[16]= 16'b0010000000000010;    // BAN 00000 00010
         data[17]= 16'b0010000000000001;    // LDA 00000 00001
```

```
        data[18]= 16'b0010000000000100;    // LDA 00000 00100
    end
    assign out = data[addr];
Endmodule
```

（3）ALU 模块：通过 op 信号选择相应的运算，在 out 端口进行输出

```
module alu_16b(
    input wire [15:0] in1, in2,
    input wire carry_in,
    input wire [2:0] op,
    output reg [15:0] out,
    output reg carry_out
);
    always@* begin
        case (op)
            3'b000: {carry_out, out} = in1 + in2 + carry_in;
            3'b001: {carry_out, out} = in1 - in2 - carry_in;
            3'b010: {carry_out, out} = in1 & in2;
            3'b011: {carry_out, out} = in1 | in2;
            3'b100:
            begin
                {carry_out, out} = in1 << 1;
                out[0] = carry_out;
            end
            3'b101: {carry_out, out} = in1 >> 1;
            3'b110: out = ~in1;
        endcase
    end
endmodule
```

（4）ACC 模块:用于暂存运算数据，可以通过 rst 信号来清零，当 wr_en 为高电平时，在时钟下边沿会将 in 端口的数据写入

```
module acc_16b(
    input wire clk,
    input wire [15:0] in,
    input wire wr_en,
    input wire rst,
    output wire [15:0] out
);
```

```
    reg [15:0] data;
    initial begin
        data = 0;
    end
    assign out = data;

    always @(negedge clk) begin
        case (rst)
            1'b1: data = 0;
            1'b0:
            begin
                case (wr_en)
                    1'b1: data = in;
                endcase
            end
        endcase
    end
Endmodule
```

（5）主存模块 memory 模块：用于存放数据，通过 addr 来访问，当 wr_en 为高电平时，在时钟下边沿会将 in 端口的数据写入

```
module memory_16b(
    input wire [9:0] addr,
    input wire wr_en, clk,
    input wire [15:0] in,
    output wire [15:0] out
);
    reg [15:0] data [1023:0];

    initial begin
        data[10'b0000000000] = 16'h0004;
        data[10'b0000000001] = 16'h0015;
        data[10'b0000000010] = 16'h0016;
    end

    assign out = data[addr];
    always @ (negedge clk) begin
        if(wr_en)
            data[addr] = in;
```

```
    end
endmodule
```

（6）多路选择器：用于当端口冲突时，在不同的指令中选择不同的数据来源

```
module mux_41_16b(
    input wire [15:0] in1, in2, in3, in4,
    output reg [15:0] out,
    input wire [1:0] select
);
    always@* begin
        case (select)
            2'b00: out = in1;
            2'b01: out = in2;
            2'b10: out = in3;
            2'b11: out = in4;
            default: out = 4'bx;
        endcase
    end
endmodule
```

(7)控制器：根据不同的 op 输入，发出不同的信号，从而控制各个组件， 达到各个组件协调工作，实现指令的目的

```
module CU(
    input wire [5:0] IR,
    output reg acc_wr_en,
    output reg acc_rst,
    // Notice: pc_rst pc_run pc_wr_en should be initialized
    output reg pc_rst,
    output reg pc_run,
    output reg pc_wr_en,
    output reg [2:0] alu_op,
    output reg mm_wr_en,
    output reg alu_carry_in,
    output reg alu_carry_out,
    output reg [1:0] mux1_select,
    output reg [1:0] mux2_select,
    output reg [1:0] mux3_select,
    output reg [1:0] mux4_select
);
```

```
/**
acc_wr_en
acc_rst
pc_rst
pc_run
pc_wr_en
alu_op
mm_wr_en
mux1_select
mux2_select
mux3_select
mux4_select
*/
initial begin
    pc_rst = 0;
    pc_run = 1;
    pc_wr_en = 0;
    alu_carry_in = 0;
    alu_carry_out = 0;
end
always@(IR) begin
    case(IR)
        6'b000001:        // CLA
        begin
            acc_wr_en = 0;
            acc_rst = 1;
            pc_rst = 0;
            pc_run = 1;
            pc_wr_en = 0;
            alu_op = 0;
            mm_wr_en = 0;
            mux1_select = 2'b00;
            mux2_select = 2'b00;
            mux3_select = 2'b00;
            mux4_select = 2'b00;
        end
        6'b000010:        // COM
        begin
```

```
    acc_wr_en = 1;
    acc_rst = 0;
    pc_rst = 0;
    pc_run = 1;
    pc_wr_en = 0;
    alu_op = 3'b110;
    mm_wr_en = 0;
    mux1_select = 2'b01;
    mux2_select = 2'b01;
    mux3_select = 2'b00;
    mux4_select = 2'b00;
end
6'b000011:        // SHR
begin
    acc_wr_en = 1;
    acc_rst = 0;
    pc_rst = 0;
    pc_run = 1;
    pc_wr_en = 0;
    alu_op = 3'b101;
    mm_wr_en = 0;
    mux1_select = 2'b01;
    mux2_select = 2'b01;
    mux3_select = 2'b00;
    mux4_select = 2'b00;
end
6'b000100:        // CSL
begin
    acc_wr_en = 1;
    acc_rst = 0;
    pc_rst = 0;
    pc_run = 1;
    pc_wr_en = 0;
    alu_op = 3'b100;
    mm_wr_en = 0;
    mux1_select = 2'b01;
    mux2_select = 2'b01;
    mux3_select = 2'b00;
```

```
    mux4_select = 2'b00;
end
6'b000101:      // STP
begin
    acc_wr_en = 0;
    acc_rst = 0;
    pc_rst = 1;
    pc_run = 0;
    pc_wr_en = 0;
    alu_op = 3'b000;
    mm_wr_en = 0;
    mux1_select = 2'b00;
    mux2_select = 2'b00;
    mux3_select = 2'b00;
    mux4_select = 2'b00;
end
6'b000110:      // ADD
begin
    acc_wr_en = 1;
    acc_rst = 0;
    pc_rst = 0;
    pc_run = 1;
    pc_wr_en = 0;
    alu_op = 0;
    mm_wr_en = 0;
    mux1_select = 2'b01;
    mux2_select = 2'b01;
    mux3_select = 2'b00;
    mux4_select = 2'b00;
end
6'b000111:      // STA
begin
    acc_wr_en = 0;
    acc_rst = 0;
    pc_rst = 0;
    pc_run = 1;
    pc_wr_en = 0;
    alu_op = 0;
```

```
    mm_wr_en = 1;
    mux1_select = 2'b00;
    mux2_select = 2'b00;
    mux3_select = 2'b00;
    mux4_select = 2'b00;
end
6'b001000:      // LDA
begin
    acc_wr_en = 1;
    acc_rst = 0;
    pc_rst = 0;
    pc_run = 1;
    pc_wr_en = 0;
    alu_op = 0;
    mm_wr_en = 0;
    mux1_select = 2'b00;
    mux2_select = 2'b00;
    mux3_select = 2'b00;
    mux4_select = 2'b00;
end
6'b001001:      // JMP
begin
    acc_wr_en = 0;
    acc_rst = 0;
    pc_rst = 0;
    pc_run = 1;
    pc_wr_en = 1;
    alu_op = 0;
    mm_wr_en = 0;
    mux1_select = 2'b00;
    mux2_select = 2'b00;
    mux3_select = 2'b00;
    mux4_select = 2'b00;
end
6'b001010:      // BAN
begin
    acc_wr_en = 0;
    acc_rst = 0;
```

```
            pc_rst = 0;
            pc_run = 1;
            pc_wr_en = 1;
            alu_op = 0;
            mm_wr_en = 0;
            mux1_select = 2'b00;
            mux2_select = 2'b00;
            mux3_select = 2'b01;
            mux4_select = 2'b01;
        end
    endcase
  end
Endmodule
```

（8）CPU：在这个文件中，声明了 pc，memory，acc，cu，alu 等模块。并且声明了几个 wire 类型的变量用于连接各个部件。

```
module CPU(
  input wire clk,
  input wire rst
);
  // PC
  wire [7:0] pc_out;
  wire pc_wr_en;
  wire pc_run;
  wire pc_rst;

  // IR
  wire [15:0] IR;

  // MM
  wire [15:0] mm_out;
  wire mm_wr_en;

  // ALU
  wire [15:0] alu_out;
  wire [2:0] alu_op;
  wire alu_carry_in;    // Not used, should be initialized in CPU start
  wire alu_carry_out;    // Not uesd, should be initialized in CPU start
```

```
// ACC
wire [15:0] acc_out;
wire acc_wr_en;
wire acc_rst;

// MUX_4_1
wire [1:0] mux1_select;
wire [1:0] mux2_select;
wire [1:0] mux3_select;
wire [1:0] mux4_select;
wire [15:0] mux1_out;
wire [15:0] mux2_out;
wire [15:0] mux3_out;
wire [15:0] mux4_out;

/**
module CU(
input wire [5:0] IR,
output reg acc_wr_en,
output reg acc_rst,
output reg pc_rst,
output reg pc_run,
output reg pc_wr_en,
output reg [2:0] alu_op,
output reg mm_wr_en,
output reg [1:0] mux1_select,
output reg [1:0] mux2_select,
output reg [1:0] mux3_select,
output reg [1:0] mux4_select
);
*/
CU cu_obj(
    .IR(IR[15:10]),
    .acc_wr_en(acc_wr_en),
    .acc_rst(acc_rst),
    .pc_rst(pc_rst),
    .pc_run(pc_run),
```

```
    .pc_wr_en(pc_wr_en),
    .alu_op(alu_op),
    .mm_wr_en(mm_wr_en),
    .alu_carry_in(alu_carry_in),
    .alu_carry_out(alu_carry_in),
    .mux1_select(mux1_select),
    .mux2_select(mux2_select),
    .mux3_select(mux3_select),
    .mux4_select(mux4_select)
);

/**
module pc_8b(
input wire clk, rst, run, wr_en,
input wire [7:0] in,
output wire [7:0] out
);
*/
// Notice: pc_rst pc_run pc_wr_en should be initialized
pc_8b pc_obj(
    .clk(clk), .rst(pc_rst), .run(pc_run), .wr_en(pc_wr_en),
    .in(mux4_out[7:0]),
    .out(pc_out)
);

/**
module IRStorage_16b(
input wire [7:0] addr,
output wire [15:0] out
);
*/
IRStorage_16b IRStorage_obj(
    .addr(pc_out),
    .out(IR)
);

/**
module alu_16b(
```

```
input wire [15:0] in1, in2,
input wire carry_in,
input wire [2:0] op,
output reg [15:0] out,
output reg carry_out
);
*/
alu_16b alu_obj(
    .in1(mux2_out), .in2(mux3_out),
    .carry_in(alu_carry_in),
    .op(alu_op),
    .out(alu_out),
    .carry_out(alu_carry_out)
);

/**
module acc_16b(
input wire clk,
input wire [15:0] in,
input wire wr_en,
input wire rst,
output wire [15:0] out
);
*/
acc_16b acc_obj(
    .clk(clk),
    .in(mux1_out),
    .wr_en(acc_wr_en),
    .rst(acc_rst),
    .out(acc_out)
);

/**
module memory_16b(
input wire [9:0] addr,
input wire wr_en, clk,
input wire [15:0] in,
output wire [15:0] out
```

```
);
*/
memory_16b mm_obj(
    .clk(clk),
    .wr_en(mm_wr_en),
    .addr(IR[9:0]),
    .in(acc_out),
    .out(mm_out)
);

/**
module mux_41_16b(
input wire [15:0] in1, in2, in3, in4,
output reg [15:0] out,
input wire [1:0] select
);
*/
mux_41_16b mux1(
    .select(mux1_select),
    .in1(mm_out),
    .in2(alu_out),
    .out(mux1_out)
);
mux_41_16b mux2(
    .select(mux2_select),
    .in1({8'b0, IR[7:0]}),
    .in2(acc_out),
    .out(mux2_out)
);
mux_41_16b mux3(
    .select(mux3_select),
    .in1(mm_out),
    .in2({8'b0, pc_out}),
    .out(mux3_out)
);
mux_41_16b mux4(
    .select(mux4_select),
    .in1({8'b0, IR[7:0]}),
```

```
        .in2(alu_out),
        .out(mux4_out)
    );
endmodule
```

（9）CPU 测试文件：用于测试 CPU 的设计完成的指令

```
module CPU_tb;
    reg clk;
    reg reset;

    initial begin
        reset = 0;
        clk = 1;

        forever #50 clk = ~clk;
        #150 $stop;
    end

    CPU cpu_obj(
        .clk(clk), .rst(reset)
    );
endmodule
```

2.实验过程

（1）设计 CPU 原理图

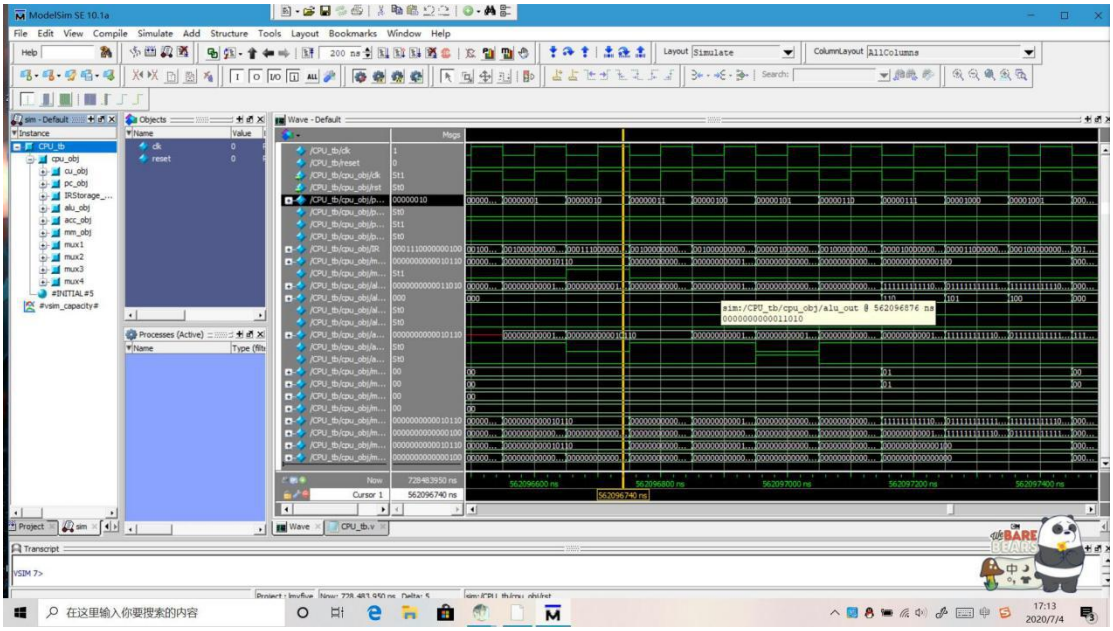
（2）编写各个模块的代码和测试代码

（3）运行程序

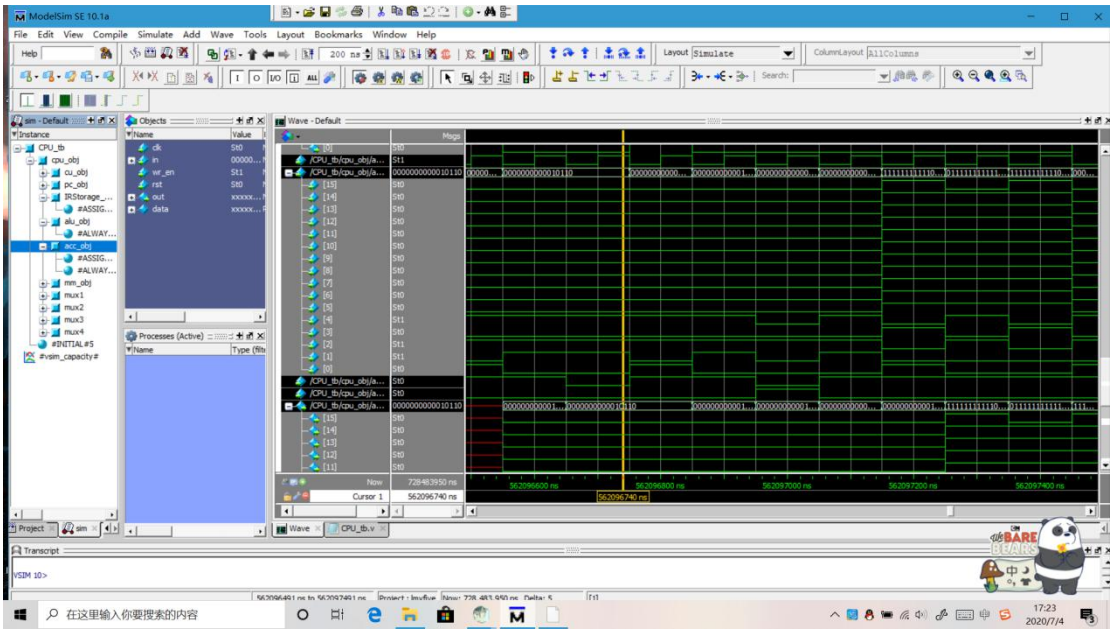
（4）观察结果

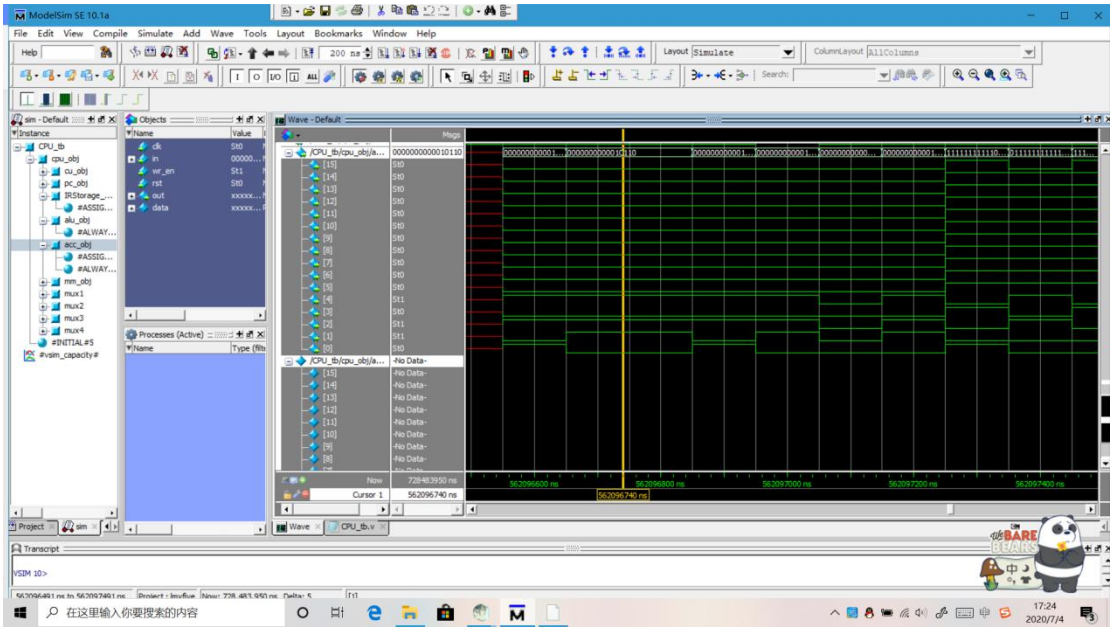
（5）各部分截图

A.CPU 截图

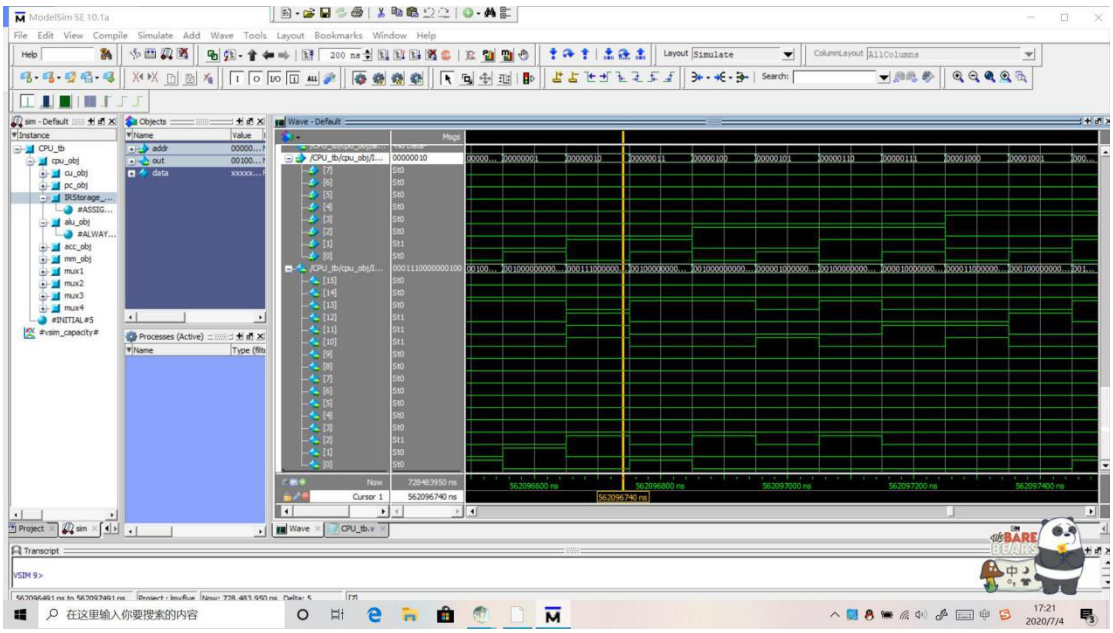


B.ACC 截图

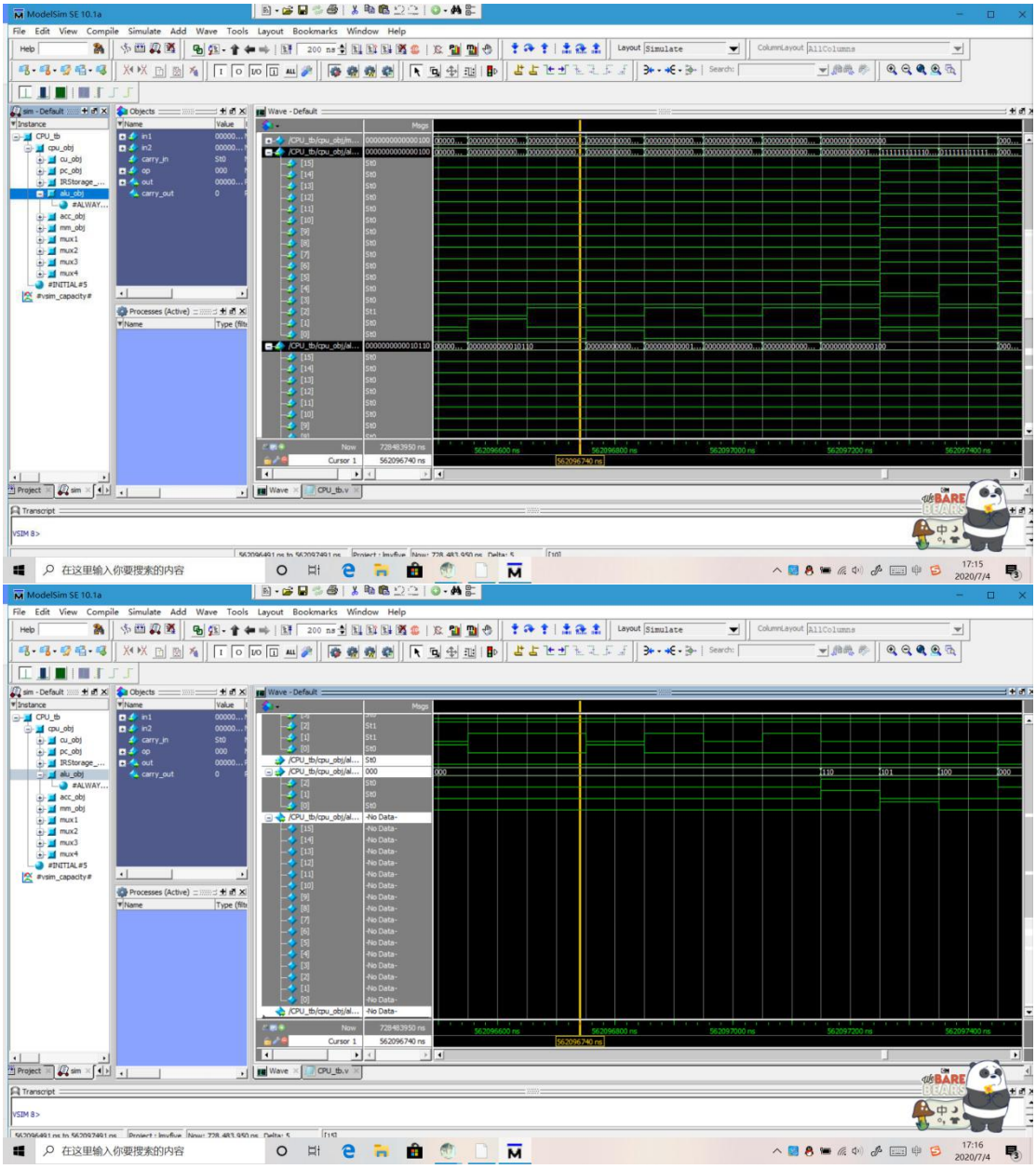



ModelSim SE 10.1a
File Edit View Compile Simulate Add Wave Tools Layout Bookmarks Window Help
Wave - Default
Objects
Processes (Active)
Transcript
VSIM 10>

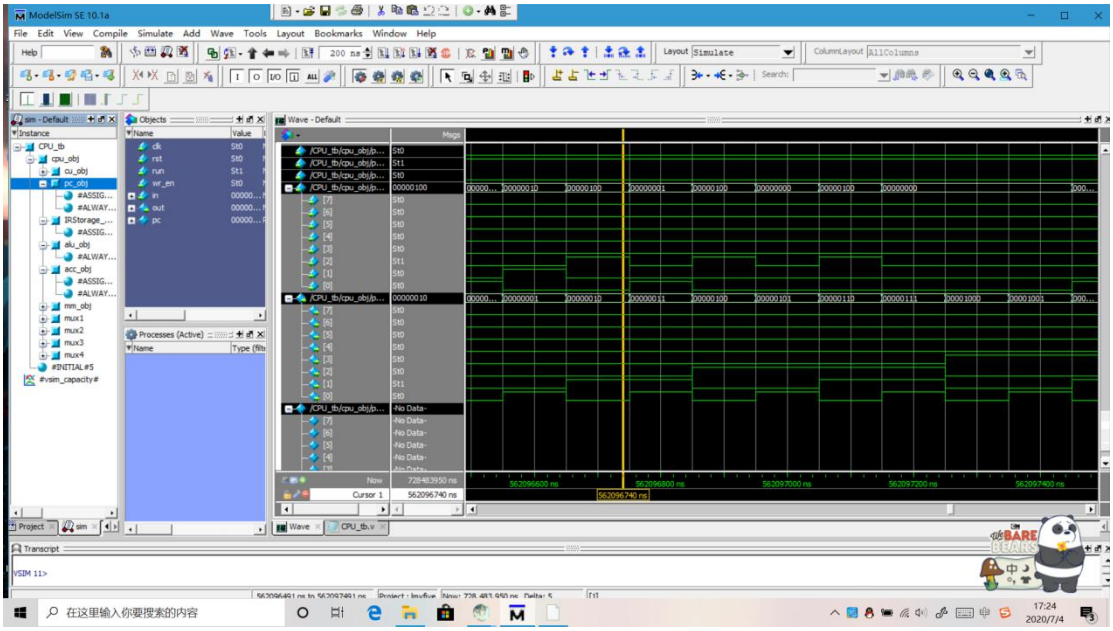
C.IR 截图


ModelSim SE 10.1a
File Edit View Compile Simulate Add Wave Tools Layout Bookmarks Window Help
Wave - Default
Objects
Processes (Active)
Transcript
VSIM 9>

D.ALU 截图


ModelSim SE 10.1a
File Edit View Compile Simulate Add Wave Tools Layout Bookmarks Window Help
Wave - Default
Objects
Processes (Active)
Transcript
VSIM 8>
在这里输入你要搜索的内容
17:15
2020/7/4
ModelSim SE 10.1a
File Edit View Compile Simulate Add Wave Tools Layout Bookmarks Window Help
Wave - Default
Objects
Processes (Active)
Transcript
VSIM 8>
在这里输入你要搜索的内容
17:16
2020/7/4

E.PC 截图



五、实验总结

因为本次疫情，没有回到学校做实验，在家实验我学会了使用 verilog 语言编写计算机各个部件，通过自己进行逻辑设计完成了单周期 CPU 的设计。同时在实验中发现对计算机组成原理的理解不够深刻，在实验编写时候总是出现各种问题，通过查阅各种文献、书籍、博客画 CPU 的结构图，然后网上寻找关于 CPU 的资料去一遍遍的修正，在把各个线路合并到一起时候还需要考虑冲突问题，设计多路选择器，最后编写代码进行测试。