实验报告

实验名称（RISC-V 基本整数指令集 RV32I 的 CPU 设计）

班级：保密 1601 学号：201608060130 姓名：其美卓嘎

实验目标

实现单周期 CPU 的设计

实验要求

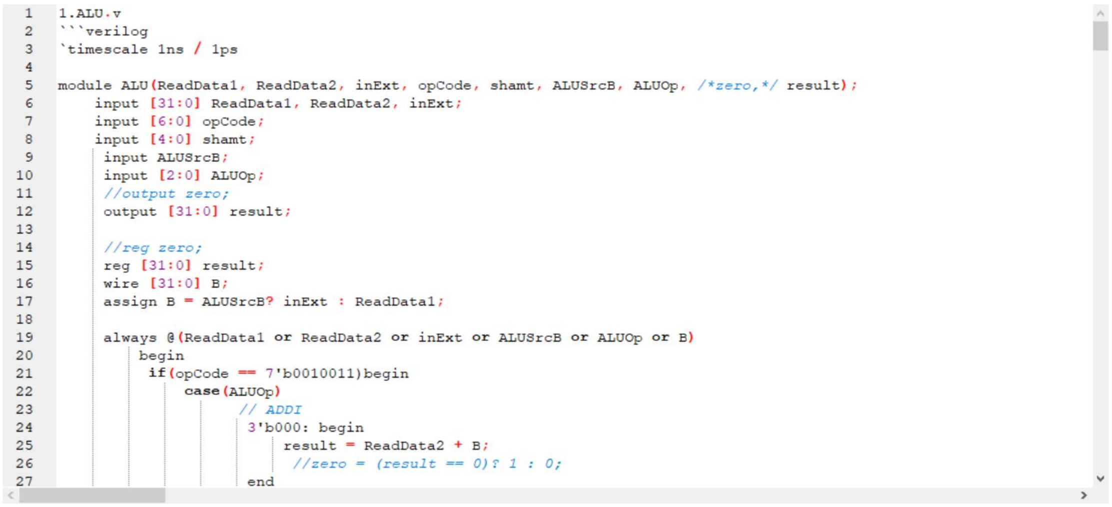
- 采用 vhdl 编写程序
- 模拟器的输入是二进制的机器指令文件
- 模拟器的输出是 CPU 各个寄存器的状态和相关的存储器单元状态

实验内容

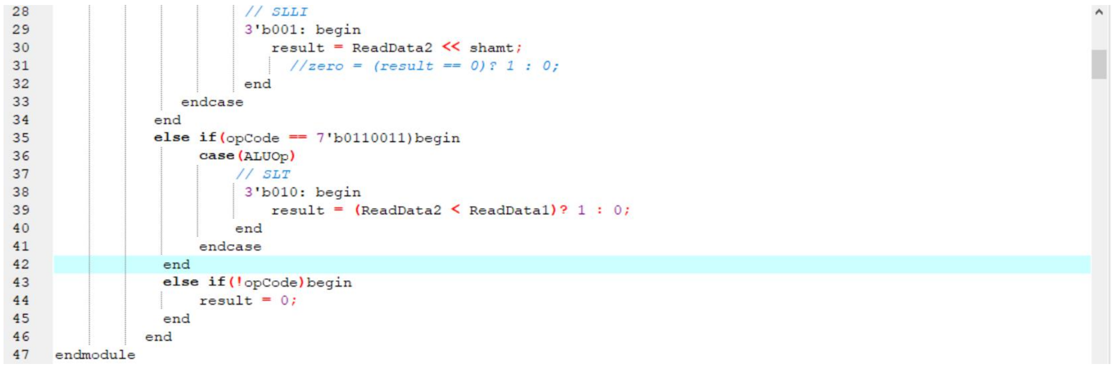
CPU 指令集：其中基本指令集有共 47 条指令集

程序框架

一、



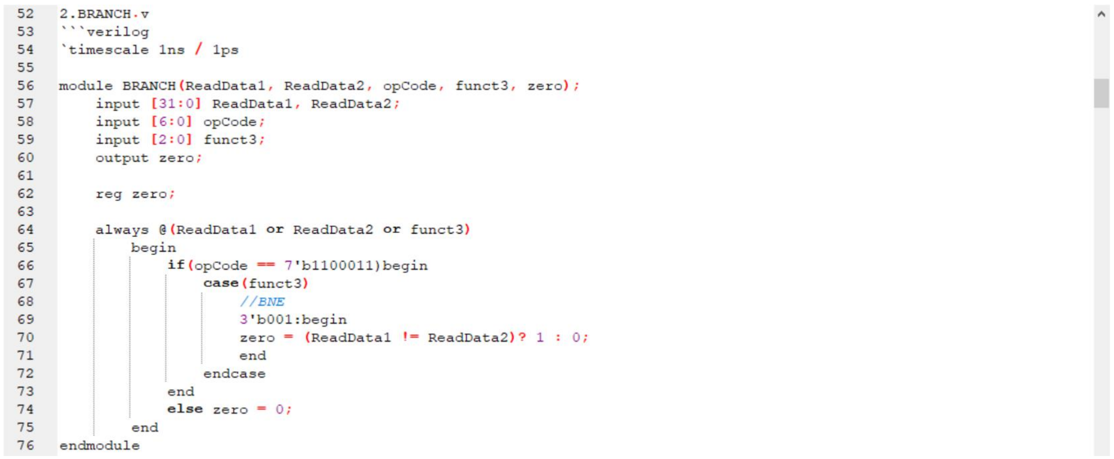
~~~
1.ALU.v
```verilog
`timescale 1ns / 1ps

module ALU(ReadData1, ReadData2, inExt, opCode, shamt, ALUSrcB, ALUOp, /*zero,*/ result);
    input [31:0] ReadData1, ReadData2, inExt;
    input [6:0] opCode;
    input [4:0] shamt;
     input ALUSrcB;
     input [2:0] ALUOp;
     //output zero;
     output [31:0] result;

     //reg zero;
     reg [31:0] result;
     wire [31:0] B;
     assign B = ALUSrcB? inExt : ReadData1;

     always @(ReadData1 or ReadData2 or inExt or ALUSrcB or ALUOp or B)
         begin
          if(opCode == 7'b0010011)begin
              case(ALUOp)
                    // ADDI
                     3'b000: begin
                         result = ReadData2 + B;
                          //zero = (result == 0)? 1 : 0;
                     end
~~~



```
                    // SLLI
                    3'b001: begin
                        result = ReadData2 << shamt;
                        //zero = (result == 0)? 1 : 0;
                    end
            endcase
        end
        else if(opCode == 7'b0110011)begin
            case(ALUOp)
                // SLT
                3'b010: begin
                    result = (ReadData2 < ReadData1)? 1 : 0;
                end
            endcase
        end
        else if(!opCode)begin
            result = 0;
        end
    end
endmodule
```

ALU 模块设置了 3 个 32 位的输入信号（ReadData1, ReadData2, inExt），分别表示 rs 寄存器的值、rt 寄存器的值和扩展后的立即数的值；设置了 7 位的输入信号 opCode、5 位的输入信号 shamt、3 位的输入信号 ALUOp（其实就是后面的 funct3 子集）以及输入的控制信号 ALUSrcB。该模块根据控制信号 ALUSrcB 来判断 ALU 命令时 ADD 类型还是 ADDI 类型选择 rs 寄存器的值或立即数的值。在 always 下，根据 opCode 可以判断是否 ALU 类型的译码而是否继续往下执行，最后根据 ALUOp（funct3）执行相应的指令内容。

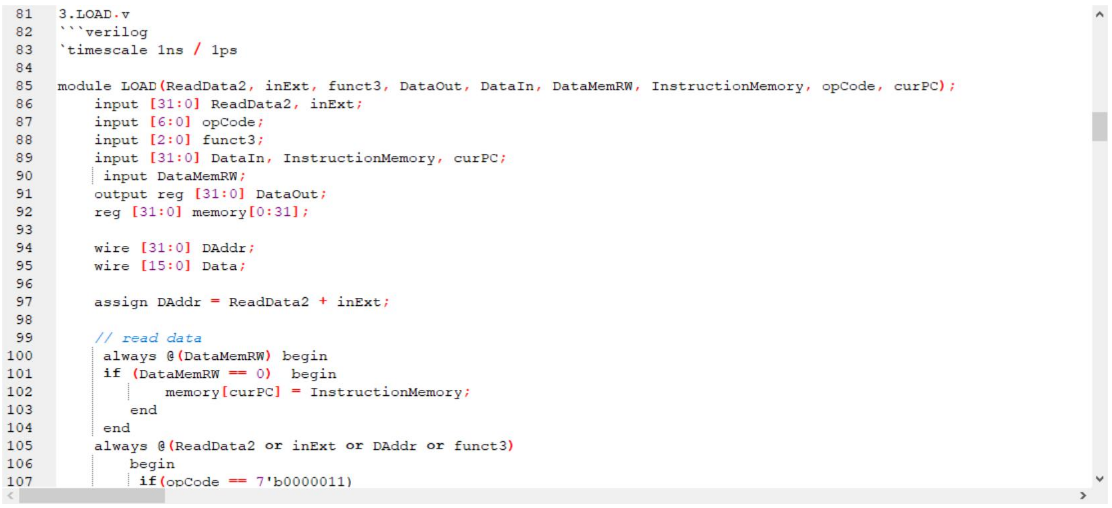
二、



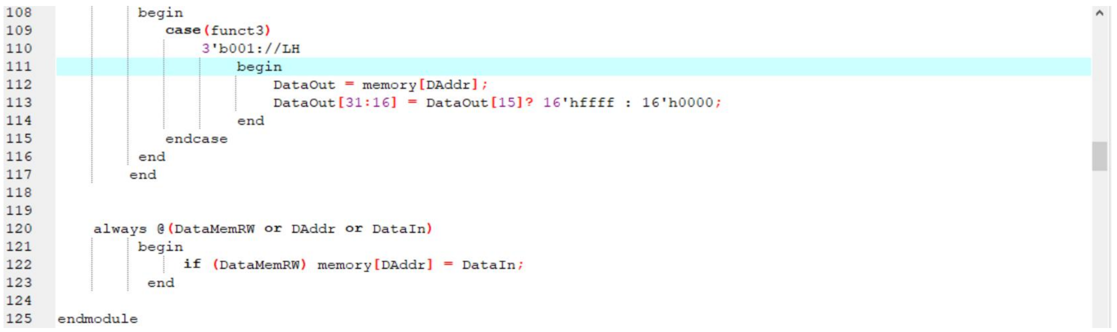
~~~
2.BRANCH.v
```verilog
`timescale 1ns / 1ps

module BRANCH(ReadData1, ReadData2, opCode, funct3, zero);
    input [31:0] ReadData1, ReadData2;
    input [6:0] opCode;
    input [2:0] funct3;
    output zero;

    reg zero;

    always @(ReadData1 or ReadData2 or funct3)
        begin
            if(opCode == 7'b1100011)begin
                case(funct3)
                    //BNE
                    3'b001:begin
                    zero = (ReadData1 != ReadData2)? 1 : 0;
                    end
                endcase
            end
            else zero = 0;
        end
endmodule
~~~

BRANCH 模块根据 rs 和 rt 寄存器里面的值是否相等将控制信号置为 1 或 0，zero 为 1 并且 PCSrc 为 1 会执行跳转指令（后面会提及到）。

三、



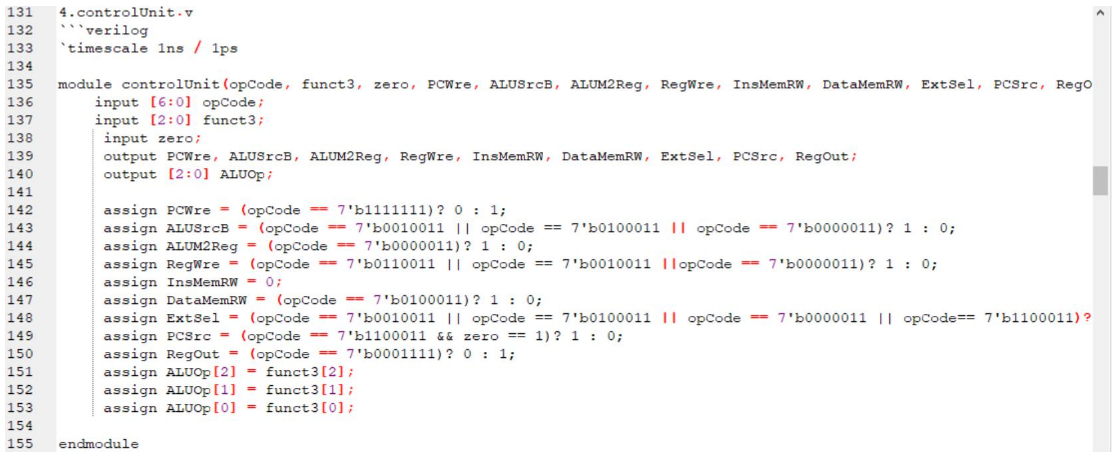
~~~
3.LOAD.v
```verilog
`timescale 1ns / 1ps

module LOAD(ReadData2, inExt, funct3, DataOut, DataIn, DataMemRW, InstructionMemory, opCode, curPC);
    input [31:0] ReadData2, inExt;
    input [6:0] opCode;
    input [2:0] funct3;
    input [31:0] DataIn, InstructionMemory, curPC;
     input DataMemRW;
    output reg [31:0] DataOut;
    reg [31:0] memory[0:31];

    wire [31:0] DAddr;
    wire [15:0] Data;

    assign DAddr = ReadData2 + inExt;

    // read data
     always @(DataMemRW) begin
     if (DataMemRW == 0)  begin
            memory[curPC] = InstructionMemory;
        end
     end
    always @(ReadData2 or inExt or DAddr or funct3)
        begin
         if(opCode == 7'b0000011)
~~~



```verilog
            begin
               case(funct3)
                   3'b001://LH
                       begin
                           DataOut = memory[DAddr];
                           DataOut[31:16] = DataOut[15]? 16'hffff : 16'h0000;
                       end
               endcase
            end
          end


     always @(DataMemRW or DAddr or DataIn)
            begin
                 if (DataMemRW) memory[DAddr] = DataIn;
             end

endmodule
```

LOAD 指令单元模块会有一点不同，因为除了本身有数据存储模块外，还需要一个指令存储模块，所以 LOAD 模块里面会包含有存储功能，以方便当指令是读功能的时候，可以读取相应所需要的内容。所以在该模块里面声明了 memory 来存储指令内容。

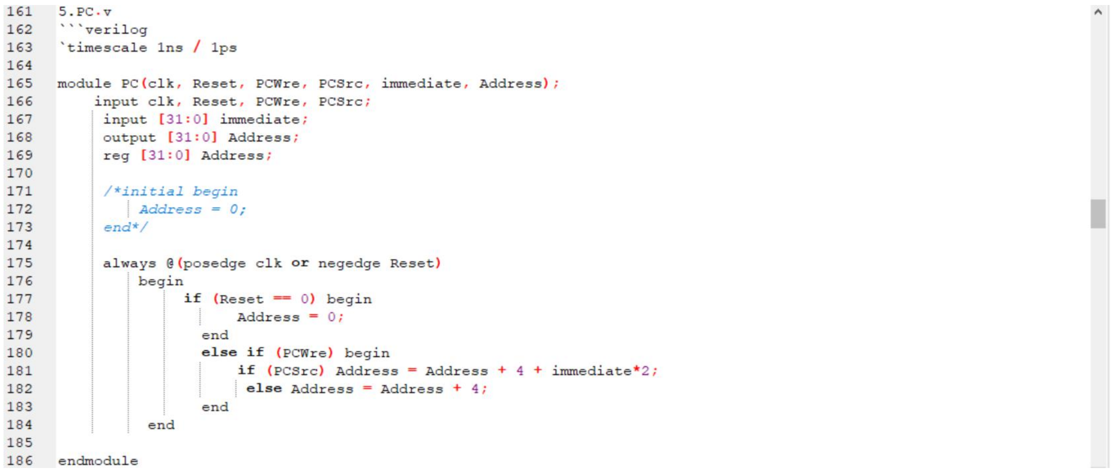
四、



~~~verilog
4.controlUnit.v
```verilog
`timescale 1ns / 1ps

module controlUnit(opCode, funct3, zero, PCWre, ALUSrcB, ALUM2Reg, RegWre, InsMemRW, DataMemRW, ExtSel, PCSrc, RegO
    input [6:0] opCode;
    input [2:0] funct3;
     input zero;
     output PCWre, ALUSrcB, ALUM2Reg, RegWre, InsMemRW, DataMemRW, ExtSel, PCSrc, RegOut;
     output [2:0] ALUOp;

     assign PCWre = (opCode == 7'b1111111)? 0 : 1;
     assign ALUSrcB = (opCode == 7'b0010011 || opCode == 7'b0100011 || opCode == 7'b0000011)? 1 : 0;
     assign ALUM2Reg = (opCode == 7'b0000011)? 1 : 0;
     assign RegWre = (opCode == 7'b0110011 || opCode == 7'b0010011 ||opCode == 7'b0000011)? 1 : 0;
     assign InsMemRW = 0;
     assign DataMemRW = (opCode == 7'b0100011)? 1 : 0;
     assign ExtSel = (opCode == 7'b0010011 || opCode == 7'b0100011 || opCode == 7'b0000011 || opCode== 7'b1100011)?
     assign PCSrc = (opCode == 7'b1100011 && zero == 1)? 1 : 0;
     assign RegOut = (opCode == 7'b0001111)? 0 : 1;
     assign ALUOp[2] = funct3[2];
     assign ALUOp[1] = funct3[1];
     assign ALUOp[0] = funct3[0];

endmodule
~~~

该模块是获取各功能的控制信号，PCWre 和 zero 同时置为 1 时执行 PC←PC+4+(sign-extend)immediate 操作；ALUSrcB 为 1 时获取来自 sign 或 zero 扩展的立即数，相关指令：addi、ori、sw、lw，否则获取来自寄存器堆 rs 输出，相关指令：add、sub、or、and、move、beq；ALUM2Reg 为 1 时获取来自数据存储器（Data MEM）的输出，相关指令：lw、lh 等，否则来自 ALU 运算结果的输出，相关指令：add、addi、sub、ori、or、and、move；RegWre 为 1 时寄存器组写使能，相关指令：add、addi、sub、ori、or、and、move、lw 等，否则无写寄存器组寄存器，相关指令：sw、halt；InsMemRW 为 0 时读指令存储器(Ins. Data)，初始化为 0；DataMemRW 为 1 时写数据存储器，相关指令：sw 等，否则读数据存储器，相关指令：lw 等；ExtSel 为 1 时进行立即数符号扩展，相关指令：addi、sw、lw、beq 等，否则进行零扩展；RegOut 为 1 时写寄存器组寄存器的地址，来自 rd 字段，相关指令：add、sub、and、or、move 等，否则写寄存器组寄存器的地址，来自 rt 字段。

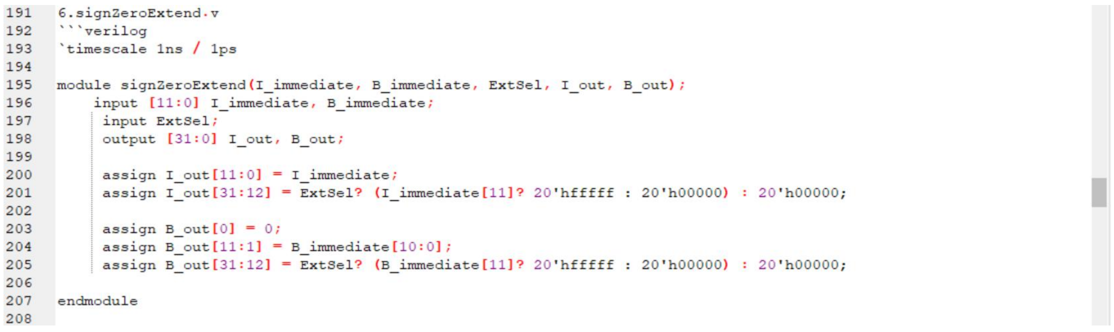
五



~~~
5.PC.v
```verilog
`timescale 1ns / 1ps

module PC(clk, Reset, PCWre, PCSrc, immediate, Address);
    input clk, Reset, PCWre, PCSrc;
     input [31:0] immediate;
     output [31:0] Address;
     reg [31:0] Address;

     /*initial begin
         Address = 0;
     end*/

     always @(posedge clk or negedge Reset)
         begin
             if (Reset == 0) begin
                 Address = 0;
               end
               else if (PCWre) begin
                   if (PCSrc) Address = Address + 4 + immediate*2;
                    else Address = Address + 4;
               end
         end

endmodule
~~~

简单的时钟输入信号和重置输入信号，根据控制信号判断地址修改的时候是否跟立即数有关。

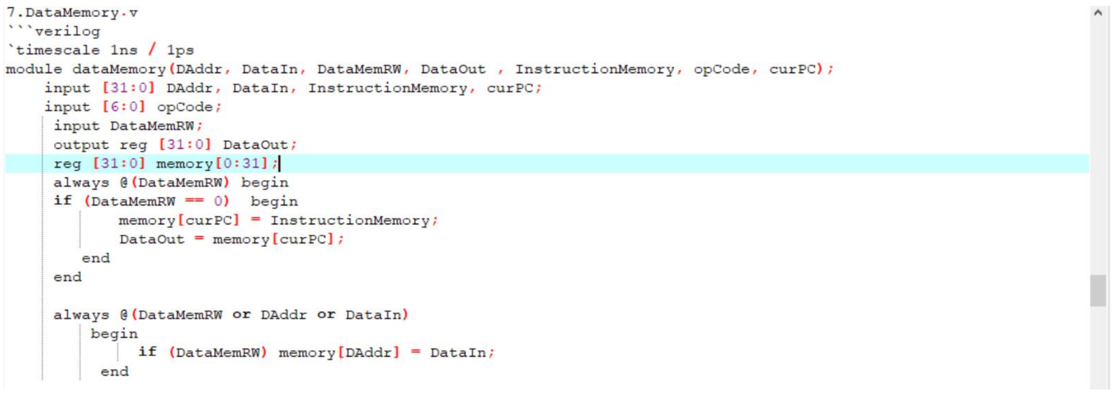
六、



~~~
6.signZeroExtend.v
```verilog
`timescale 1ns / 1ps

module signZeroExtend(I_immediate, B_immediate, ExtSel, I_out, B_out);
    input [11:0] I_immediate, B_immediate;
     input ExtSel;
     output [31:0] I_out, B_out;

     assign I_out[11:0] = I_immediate;
     assign I_out[31:12] = ExtSel? (I_immediate[11]? 20'hfffff : 20'h00000) : 20'h00000;

     assign B_out[0] = 0;
     assign B_out[11:1] = B_immediate[10:0];
     assign B_out[31:12] = ExtSel? (B_immediate[11]? 20'hfffff : 20'h00000) : 20'h00000;

endmodule
~~~

扩充立即数的单元模块，此处只处理了 Itype 类型和 Btype 类型的立即数。

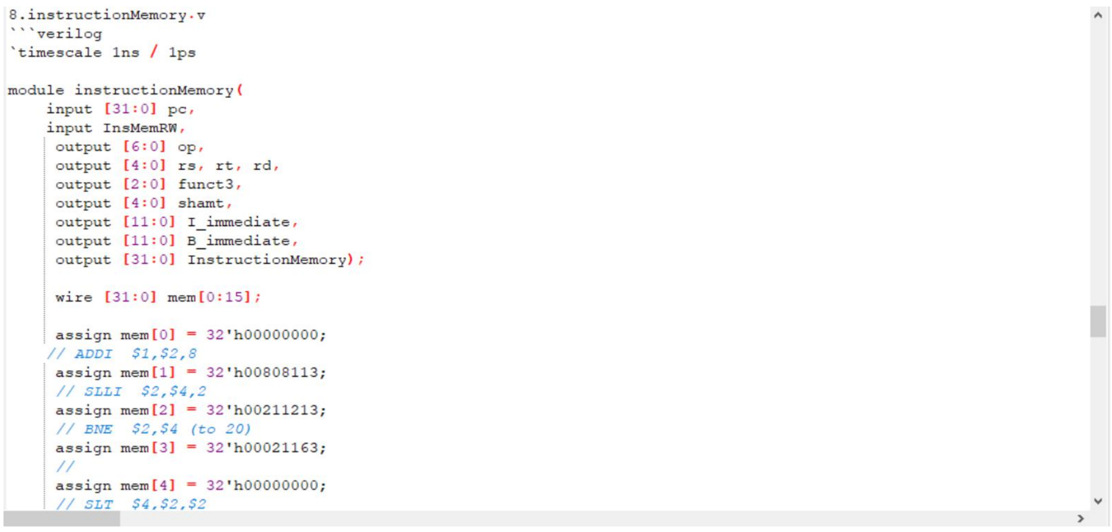
七、



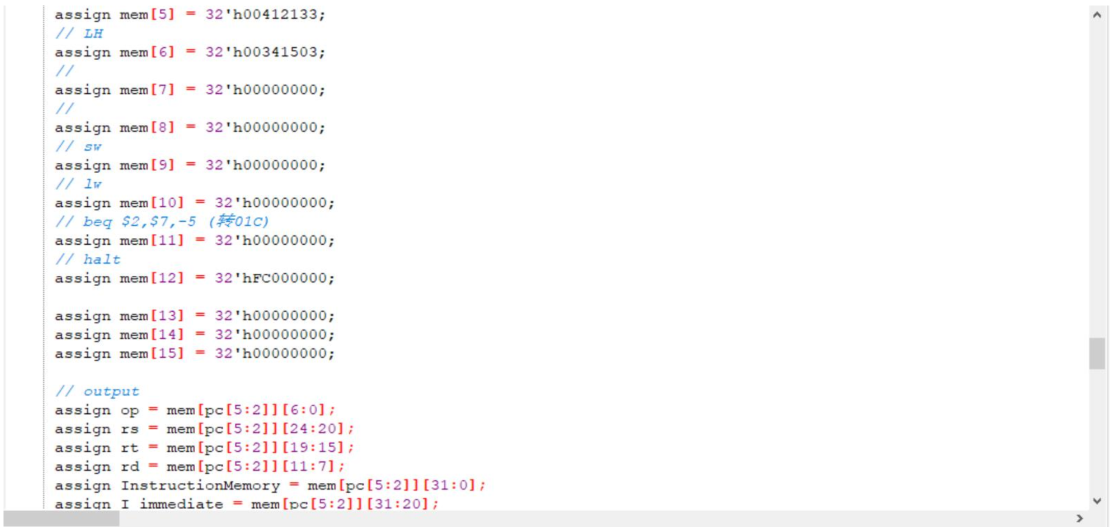
~~~
7.DataMemory.v
```verilog
`timescale 1ns / 1ps
module dataMemory(DAddr, DataIn, DataMemRW, DataOut , InstructionMemory, opCode, curPC);
    input [31:0] DAddr, DataIn, InstructionMemory, curPC;
    input [6:0] opCode;
     input DataMemRW;
     output reg [31:0] DataOut;
     reg [31:0] memory[0:31];
     always @(DataMemRW) begin
     if (DataMemRW == 0)  begin
            memory[curPC] = InstructionMemory;
            DataOut = memory[curPC];
         end
     end

     always @(DataMemRW or DAddr or DataIn)
         begin
             if (DataMemRW) memory[DAddr] = DataIn;
          end
~~~

endmodule 和 LOAD 单元模块的存储数据一样，这里增加了每次时钟周期查看当前存储的数据的功能，即输出信号 DataOut。

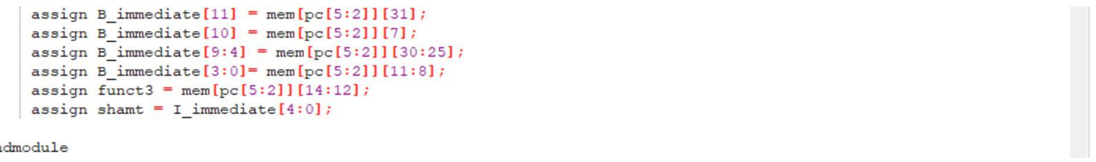
八、



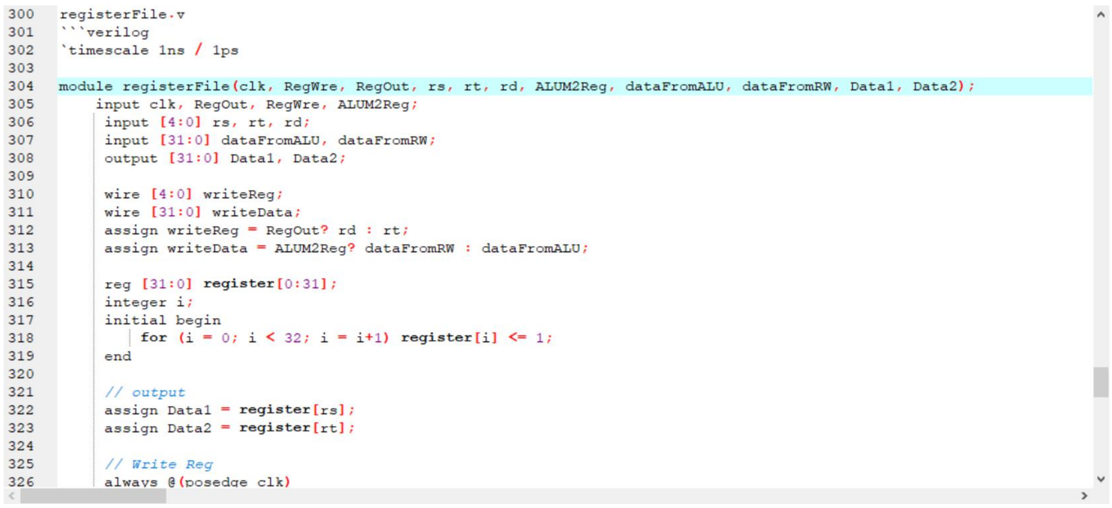
~~~
8.instructionMemory.v
```verilog
`timescale 1ns / 1ps

module instructionMemory(
    input [31:0] pc,
    input InsMemRW,
     output [6:0] op,
     output [4:0] rs, rt, rd,
     output [2:0] funct3,
     output [4:0] shamt,
     output [11:0] I_immediate,
     output [11:0] B_immediate,
     output [31:0] InstructionMemory);

     wire [31:0] mem[0:15];

     assign mem[0] = 32'h00000000;
    // ADDI  $1,$2,8
     assign mem[1] = 32'h00808113;
     // SLLI  $2,$4,2
     assign mem[2] = 32'h00211213;
     // BNE  $2,$4 (to 20)
     assign mem[3] = 32'h00021163;
     //
     assign mem[4] = 32'h00000000;
     // SLT  $4,$2,$2
~~~



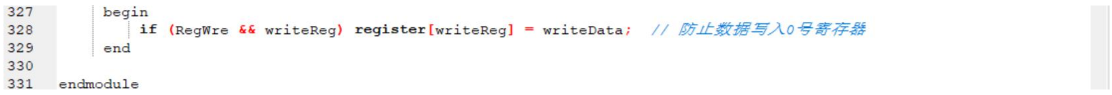
```
     assign mem[5] = 32'h00412133;
     // LH
     assign mem[6] = 32'h00341503;
     //
     assign mem[7] = 32'h00000000;
     //
     assign mem[8] = 32'h00000000;
     // sw
     assign mem[9] = 32'h00000000;
     // lw
     assign mem[10] = 32'h00000000;
     // beq $2,$7,-5 (转01C)
     assign mem[11] = 32'h00000000;
     // halt
     assign mem[12] = 32'hFC000000;

     assign mem[13] = 32'h00000000;
     assign mem[14] = 32'h00000000;
     assign mem[15] = 32'h00000000;

     // output
     assign op = mem[pc[5:2]][6:0];
     assign rs = mem[pc[5:2]][24:20];
     assign rt = mem[pc[5:2]][19:15];
     assign rd = mem[pc[5:2]][11:7];
     assign InstructionMemory = mem[pc[5:2]][31:0];
     assign I_immediate = mem[pc[5:2]][31:20];
```



```
 assign B_immediate[11] = mem[pc[5:2]][31];
 assign B_immediate[10] = mem[pc[5:2]][7];
 assign B_immediate[9:4] = mem[pc[5:2]][30:25];
 assign B_immediate[3:0]= mem[pc[5:2]][11:8];
 assign funct3 = mem[pc[5:2]][14:12];
 assign shamt = I_immediate[4:0];

endmodule
```

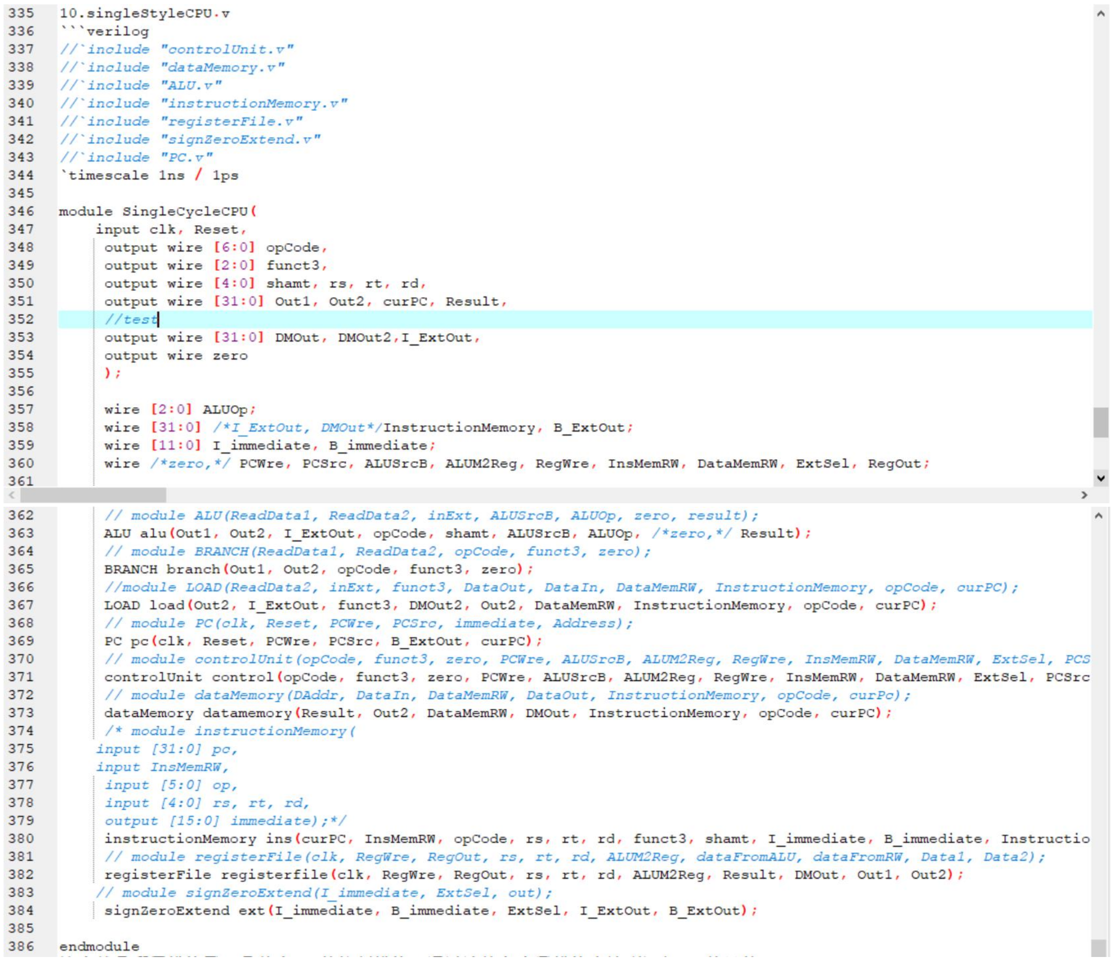


~~~
registerFile.v
```verilog
`timescale 1ns / 1ps

module registerFile(clk, RegWre, RegOut, rs, rt, rd, ALUM2Reg, dataFromALU, dataFromRW, Data1, Data2);
    input clk, RegOut, RegWre, ALUM2Reg;
     input [4:0] rs, rt, rd;
     input [31:0] dataFromALU, dataFromRW;
     output [31:0] Data1, Data2;

     wire [4:0] writeReg;
     wire [31:0] writeData;
     assign writeReg = RegOut? rd : rt;
     assign writeData = ALUM2Reg? dataFromRW : dataFromALU;

     reg [31:0] register[0:31];
     integer i;
     initial begin
         for (i = 0; i < 32; i = i+1) register[i] <= 1;
     end

     // output
     assign Data1 = register[rs];
     assign Data2 = register[rt];

     // Write Reg
     always @(posedge clk)
~~~



```verilog
    begin
        if (RegWre && writeReg) register[writeReg] = writeData;  // 防止数据写入0号寄存器
    end

endmodule
```

该单元模块基本功能就是将 rs、rt 寄存器里面的指赋值给输出信号。除此之外，根据控制信号判断存储到的寄存器编号是根据 rt 还是 rd，存储的内容是根据 ALU 算法还是基于读算法来写入数据。



~~~verilog
10.singleStyleCPU.v
```verilog
//`include "controlUnit.v"
//`include "dataMemory.v"
//`include "ALU.v"
//`include "instructionMemory.v"
//`include "registerFile.v"
//`include "signZeroExtend.v"
//`include "PC.v"
`timescale 1ns / 1ps

module SingleCycleCPU(
    input clk, Reset,
     output wire [6:0] opCode,
     output wire [2:0] funct3,
     output wire [4:0] shamt, rs, rt, rd,
     output wire [31:0] Out1, Out2, curPC, Result,
     //test
     output wire [31:0] DMOut, DMOut2,I_ExtOut,
     output wire zero
     );

     wire [2:0] ALUOp;
     wire [31:0] /*I_ExtOut, DMOut*/InstructionMemory, B_ExtOut;
     wire [11:0] I_immediate, B_immediate;
     wire /*zero,*/ PCWre, PCSrc, ALUSrcB, ALUM2Reg, RegWre, InsMemRW, DataMemRW, ExtSel, RegOut;

     // module ALU(ReadData1, ReadData2, inExt, ALUSrcB, ALUOp, zero, result);
     ALU alu(Out1, Out2, I_ExtOut, opCode, shamt, ALUSrcB, ALUOp, /*zero,*/ Result);
     // module BRANCH(ReadData1, ReadData2, opCode, funct3, zero);
     BRANCH branch(Out1, Out2, opCode, funct3, zero);
     //module LOAD(ReadData2, inExt, funct3, DataOut, DataIn, DataMemRW, InstructionMemory, opCode, curPC);
     LOAD load(Out2, I_ExtOut, funct3, DMOut2, Out2, DataMemRW, InstructionMemory, opCode, curPC);
     // module PC(clk, Reset, PCWre, PCSrc, immediate, Address);
     PC pc(clk, Reset, PCWre, PCSrc, B_ExtOut, curPC);
     // module controlUnit(opCode, funct3, zero, PCWre, ALUSrcB, ALUM2Reg, RegWre, InsMemRW, DataMemRW, ExtSel, PCS
     controlUnit control(opCode, funct3, zero, PCWre, ALUSrcB, ALUM2Reg, RegWre, InsMemRW, DataMemRW, ExtSel, PCSrc
     // module dataMemory(DAddr, DataIn, DataMemRW, DataOut, InstructionMemory, opCode, curPc);
     dataMemory datamemory(Result, Out2, DataMemRW, DMOut, InstructionMemory, opCode, curPC);
     /* module instructionMemory(
    input [31:0] pc,
    input InsMemRW,
     input [5:0] op,
     input [4:0] rs, rt, rd,
     output [15:0] immediate);*/
     instructionMemory ins(curPC, InsMemRW, opCode, rs, rt, rd, funct3, shamt, I_immediate, B_immediate, Instructio
     // module registerFile(clk, RegWre, RegOut, rs, rt, rd, ALUM2Reg, dataFromALU, dataFromRW, Data1, Data2);
     registerFile registerfile(clk, RegWre, RegOut, rs, rt, rd, ALUM2Reg, Result, DMOut, Out1, Out2);
    // module signZeroExtend(I_immediate, ExtSel, out);
     signZeroExtend ext(I_immediate, B_immediate, ExtSel, I_ExtOut, B_ExtOut);

endmodule
~~~

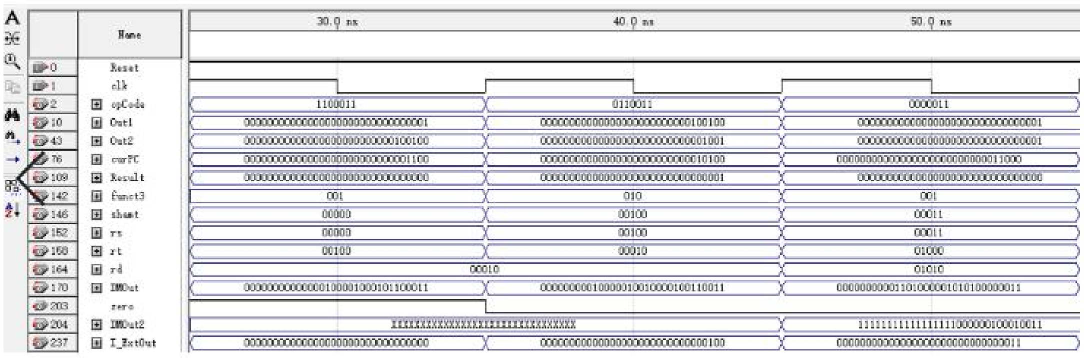
顶层模块，是整个 CPU 的控制模块，通过连接各个子模块来达到运行 CPU 的目的。

测试

测试平台

部件	配置	备注
CPU	core(TM) i3-6100U	
内存	DDR3-8GB	
操作系统	Windows10 家庭版	64 位操作系统

第一条指令 在第 2 个寄存器写入 0x66



```
                     30.0 ns                              40.0 ns                              50.0 ns
0    Reset
1    clk
2    opCode   1100011                            0110011                              0000011
10   Out1     00000000000000000000000000000001   00000000000000000000000000100100     00000000000000000000000000000001
43   Out2     00000000000000000000000000100100   00000000000000000000000000001001     00000000000000000000000000000001
76   curPC    00000000000000000000000000001100   00000000000000000000000000010100     00000000000000000000000000011000
109  Result   00000000000000000000000000000000   00000000000000000000000000000001     00000000000000000000000000000000
142  funct3   001                                010                                  001
146  shamt    00000                              00100                                00011
152  rs       00000                              00100                                00011
158  rt       00100                              00010                                01000
164  rd                       00010                                                   01010
170  IMOut    0000000000000100001000101100011    00000000010000010010000100110011     00000000001101000001010100000011
203  zero
204  DMOut2              XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX                             11111111111111111000000100010011
237  I_ExtOut 00000000000000000000000000000000   00000000000000000000000000000100     00000000000000000000000000000011
```

分析和结论

从测试记录来看，模拟器实现了对二进制指令文件的读入，指令功能的模拟，CPU 和存储器状态的输出。根据分析结果，可以认为编写的模拟器实现了所要求的功能，完成了实验目标。