实 验 报 告

学号：2014K8009929022　姓名：孔静　专业：计算机科学与技术

实验序号：2　实验名称：单周期处理器设计

一、　代码以及波形图

testbench 部分先设置两个外部存储器，存储好所需数据，其中 Instruction 的存储器总共存储了 12 道指令，10ns 执行一道指令（5#clk=~clk），指令依次为 load，load，add，store，sub，slt，store，and，store，or，store。
以下图片分别为主程序，alu，regfiles，Generating_ALU_Control，Control_Truth_Table 的相应数据信号截图。

图 1

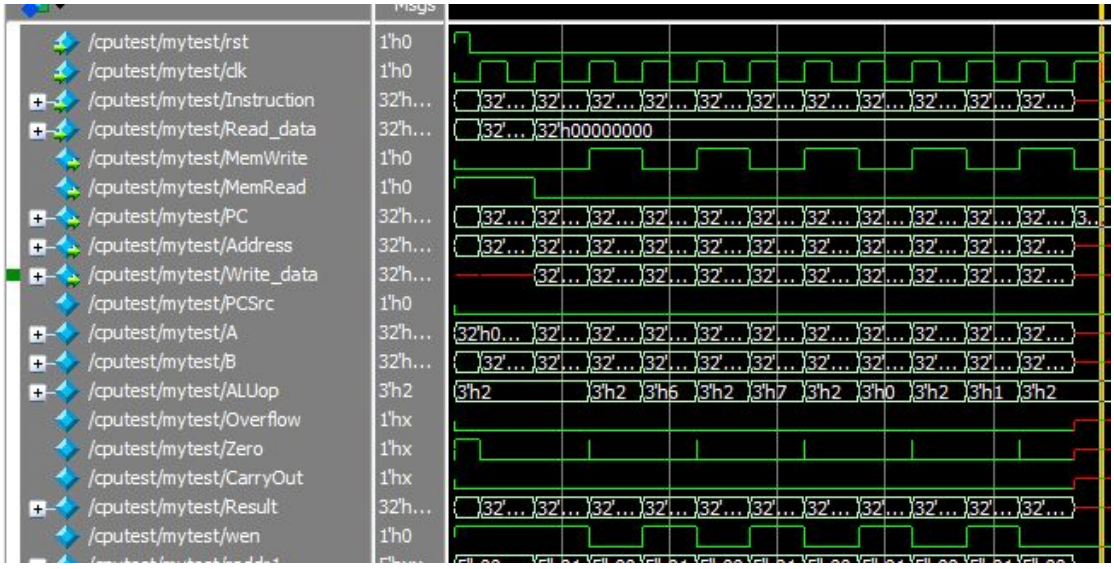

图 2

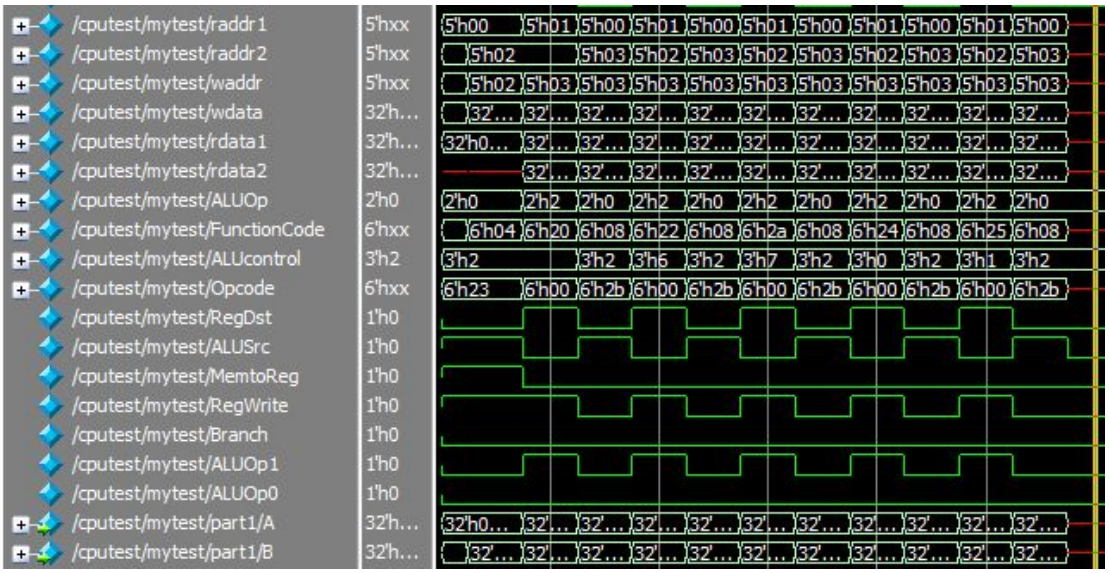

图 1 和图 2，都是 single_cycle_cpu 里面的数据信号。
可以发现除了图 1 的 Write_data、图 2 的 rdata2、以及图 4 寄存器的 rdata2，这三个信号初始没有信号为红色，因为初始时除了 regfiles 地址 5’b0 处默认为 0 有数外，其他部分都没有数据，而我 rdata1 等信号初始读取的都是 regfiles 地址 5’b0 处，然后随着不同操作的进行，regfiles 内部开始有数据存入，所以这三个信号开始有所变化变为绿色。
single_cycle_cpu 连线代码如下：

```
always @(posedge clk or posedge rst)
begin
  if(rst)
    PC<=32'b0;
  else
    PC<=(PCSrc)? PC+4+((Instruction[15])? {14'h3fff,Instruction[15:0],2'b0}:{14'b0,Instruction[15:0],2'b0}):PC+4;
end

assign PCSrc=Branch&Zero;
assign Address=Result;
assign Write_data=rdata2;
//cpu
assign A=rdata1;
assign B= (ALUSrc)? ((Instruction[15])? {16'hffff,Instruction[15:0]}:{16'b0,Instruction[15:0]}) : rdata2;
assign ALUop=ALUcontrol;
//alu
assign wen=RegWrite;
assign raddr1=Instruction[25:21];
assign raddr2=Instruction[20:16];
assign waddr= (RegDst)? Instruction[15:11]:Instruction[20:16];
assign wdata= (MemtoReg)? Read_data:Result;
//regfiles
assign ALUOp[1]=ALUOp1;
assign ALUOp[0]=ALUOp0;
assign FunctionCode=Instruction[5:0];
//Generating_ALU_Control
assign Opcode=Instruction[31:26];
//Control_Truth_Table
```

图 3

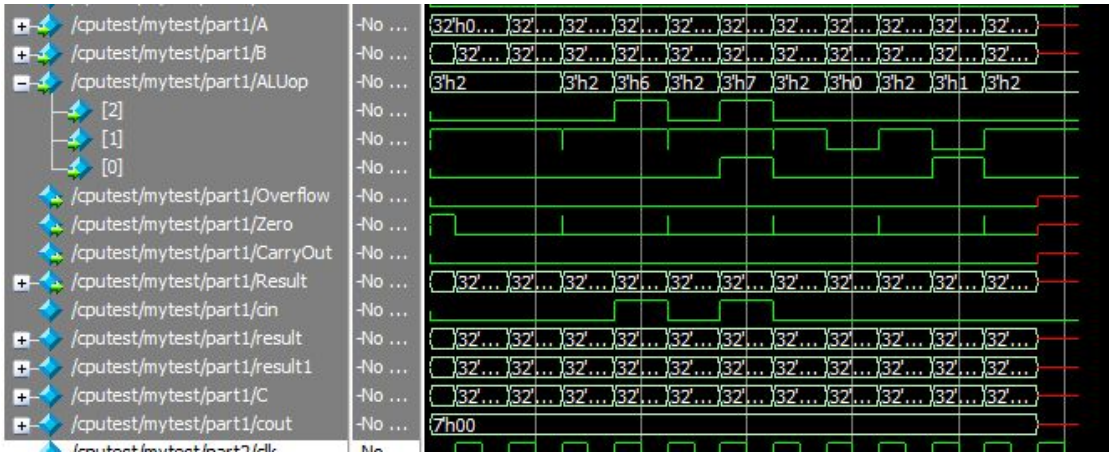

图 3 为 alu 的信号部分。

alu 代码略。

图 4

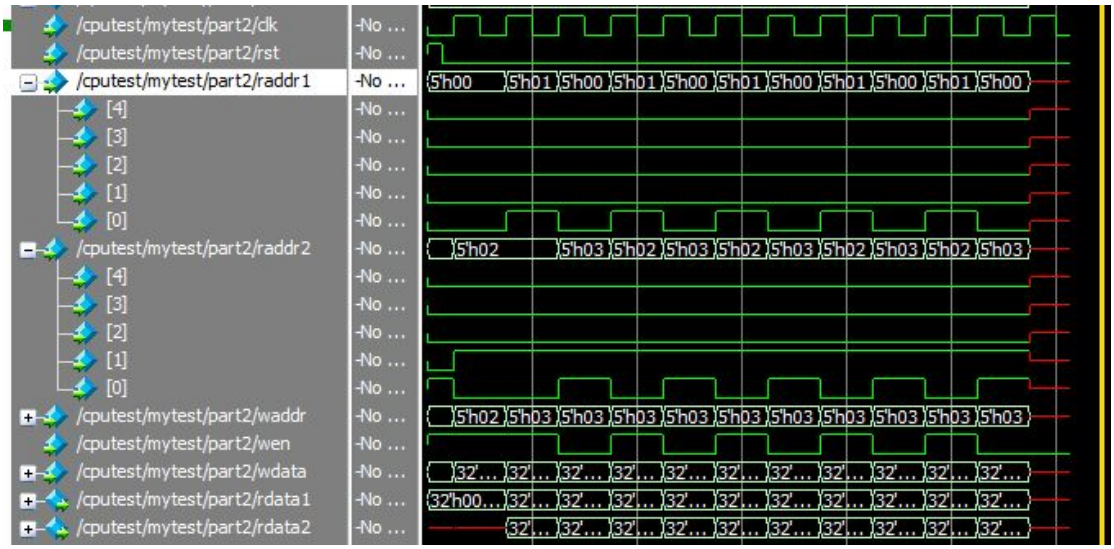

图 4 为 regfiles 的信号部分。
regfiles 代码略。

图 5

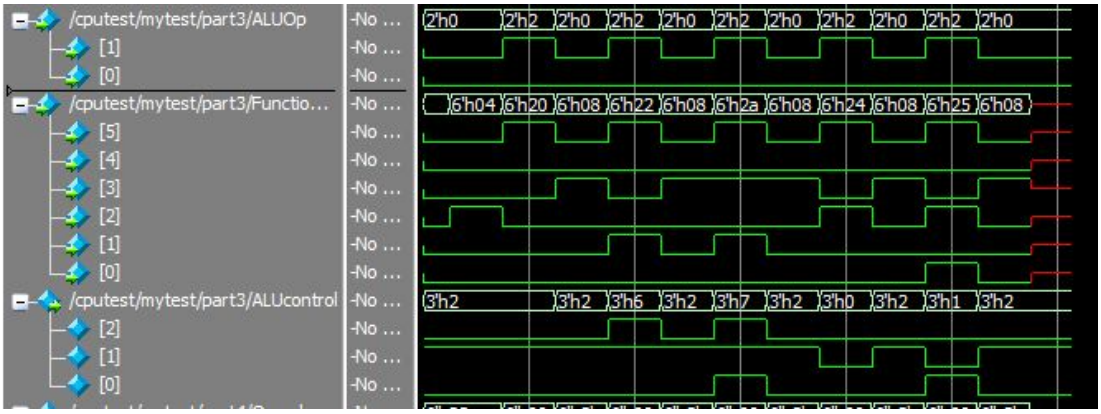

图 5 为 Generating_ALU_Control 的信号部分，根据 ALUOp（由 Instruction[31:26]给出）FunctionCode（由 Instruction[5:0]给出），定义 ALUcontrol（连线 alu 中的 ALUop）来控制 alu 进行相应运算。
Generating_ALU_Control 代码如下：

```
module Generating_ALU_Control(ALUOp,FunctionCode,ALUcontrol);
  input [1:0]ALUOp;
  input [5:0]FunctionCode;
  output [2:0]ALUcontrol;
  reg [2:0]ALUcontrol;

  always @(ALUOp or FunctionCode)
  begin
    case(ALUOp)
      2'b00:ALUcontrol=3'b010;//add
      2'b01:ALUcontrol=3'b110;//sub
      2'b10:
      begin
        case(FunctionCode)
          6'b100000:ALUcontrol=3'b010;//add
          6'b100010:ALUcontrol=3'b110;//sub
          6'b100100:ALUcontrol=3'b000;//and
          6'b100101:ALUcontrol=3'b001;//or
          6'b101010:ALUcontrol=3'b111;//slt
          default:ALUcontrol=3'b000;
        endcase
      end
      default:ALUcontrol=3'b000;
    endcase
  end

endmodule
```

图 6

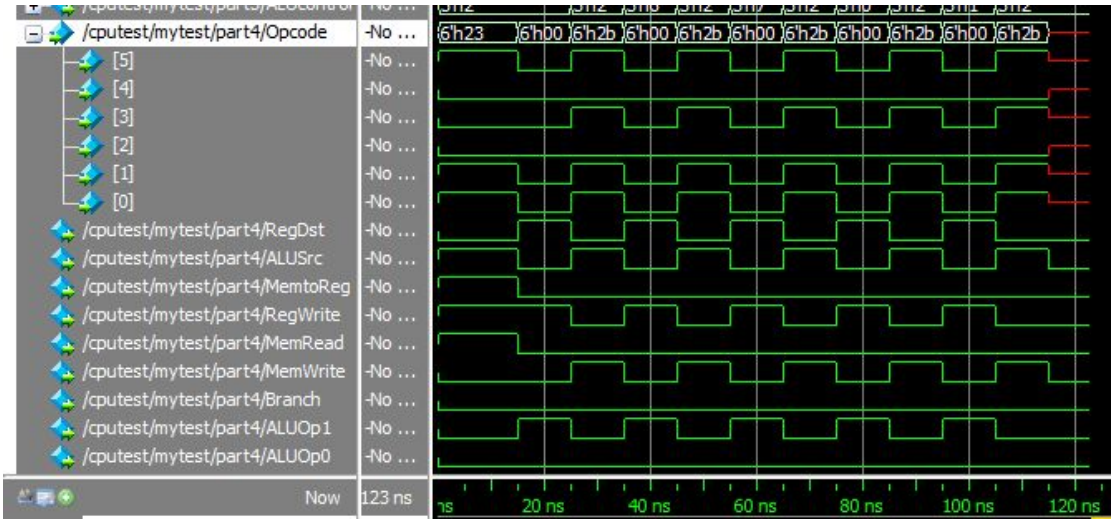

图 6 为 Control_Truth_Table 的信号部分，根据 Opcode（由 Instruction[31:26]给出）定义 RegDst，ALUSrc，MemtoReg，RegWrite，MemRead，MemWrite，Branch，ALUOp1，ALUOp0。

Control_Truth_Table 代码如下：

```
module Control_Truth_Table(Opcode,RegDst,ALUSrc,MemtoReg,RegWrite,MemRead,MemWrite,Branch,ALUOp1,ALUOp0);
  input [5:0]Opcode;
  output RegDst,ALUSrc,MemtoReg,RegWrite,MemRead,MemWrite,Branch,ALUOp1,ALUOp0;
  reg RegDst,ALUSrc,MemtoReg,RegWrite,MemRead,MemWrite,Branch,ALUOp1,ALUOp0;

  always @(Opcode)
  begin
    case(Opcode)
      6'b000000://waddr=Instruction[15:11],B=rdata2,wdata=Result,wen=1,MemRead=0,MemWrite=0,PC=PC+4,case(FunctionCode)
      begin
        RegDst=1'b1;ALUSrc=1'b0;MemtoReg=1'b0;RegWrite=1'b1;MemRead=1'b0;MemWrite=1'b0;Branch=1'b0;ALUOp1=1'b1;ALUOp0=1'b0;
      end
      6'b100011://waddr=Instruction[20:16],B=Instruction[15:0],wdata=Read_data,wen=1,MemRead=1,MemWrite=0,PC=PC+4,add
      begin
        RegDst=1'b0;ALUSrc=1'b1;MemtoReg=1'b1;RegWrite=1'b1;MemRead=1'b1;MemWrite=1'b0;Branch=1'b0;ALUOp1=1'b0;ALUOp0=1'b0;
      end
      6'b101011://B=Instruction[15:0],wen=0,MemRead=0,MemWrite=1,PC=PC+4,add
      begin
        RegDst=1'b0;ALUSrc=1'b1;MemtoReg=1'b0;RegWrite=1'b0;MemRead=1'b0;MemWrite=1'b1;Branch=1'b0;ALUOp1=1'b0;ALUOp0=1'b0;
      end
      6'b000100://B=rdata2,wen=0,MemRead=0,MemWrite=0,PC=PC+4+Insturction[15:0],sub
      begin
        RegDst=1'b0;ALUSrc=1'b0;MemtoReg=1'b0;RegWrite=1'b0;MemRead=1'b0;MemWrite=1'b0;Branch=1'b1;ALUOp1=1'b0;ALUOp0=1'b1;
      end
      default:
      begin
        RegDst=1'b0;ALUSrc=1'b0;MemtoReg=1'b0;RegWrite=1'b0;MemRead=1'b0;MemWrite=1'b0;Branch=1'b0;ALUOp1=1'b0;ALUOp0=1'b0;
      end
```

二、 问题合集

这次基本没啥问题，根据 Single+Cycle+CPU.pdf Single+Cycle+CPU.pdf 文件中 29 页的电路图给数据连线，33 页 Generating_ALU_Control 表格以及 35 页 Control_Truth_Table 表格 case 不同情况给数据赋值之后，直接写 testbench 就可以跑了=。=，顺利的比较让人意外 O(∩_∩)O~~。

Problem1

```
//always @(posedge clk)
begin
  if(rst)
    PC<=32'b0;
  else
    PC<=(PCSrc)? PC+4+((Instruction[15])? {14'h3fff,Instruction[15:0],2'b0}:{14'b0,Instruction[15:0],2'b0}):PC+4;
end
```

问题：在 always 部分没有将 rst 写入，导致 testbench，rst=1 初始化时，若 clk 没有上升信号，则初始化失败，导致 wave 波形图大片红线。

解决：后将 posedge rst 补在 always 条件里了。

Problem2

问题：给老师检查的时候，老师说 rst 在高阻态的时候，最好保持 clk 不要动，如果我的程序有漏洞，那么有可能在 rst 是高阻态的时候，由于 clk 在变化，仍然执行了程序导致错误。

解决：选择老师提供的第二种解决办法，将 rst 为高阻态的时间变短，短到 clk 还没有上升信号即可。

Problem3

```
6'b000100:
begin
  RegDst=1'b0;ALUSrc=1'b0;MemtoReg=1'b0;RegWrite=1'b0;MemRead=1'b0;MemWrite=1'b0;Branch=1'b1;ALUOp1=1'b0;ALUOp0=1'b1;
  //RegDst=1'bx;ALUSrc=1'b0;MemtoReg=1'bx;RegWrite=1'b0;MemRead=1'b0;MemWrite=1'b0;Branch=1'b1;ALUOp1=1'b0;ALUOp0=1'b1;
end
```

问题：傻傻的根据表格，在赋值 x 即任意都可的情况下，真的给数据赋值了 x。

解决：将所有 x 改成了 0。

三、 对于此次实验的心得、感受和建议

(⊙v⊙)嗯，由于此次作业主要部分都是连线，所以没怎么花时间，除了在 testbench 部分想了一下具体操作指令所对应的操作。听说有的同学在 Simulate 的时候发现自己漏了一些连线，有点庆幸自己根据那个电路图按 single_cycle_cpu，alu，regfiles，Generating_ALU_Control，Control_Truth_Table 五个函数涉及数据的顺序，一条条连线，以及将变量命名保持了高度一致，所以一次性过了。

在写 testbench 的时候，本来想要按照以前一样，一边给里面数据一边跑程序，但是谈清扬 mvp 同学及时阻止了我即将要犯的错误，告诉我应该先设置外部存储器将数据放完之后，让它自己跑，表示对谈 mvp 同学的感谢。还有最开始写 single_cycle_cpu 的时候，感谢张传奇 carry 同学的指引，告诉我那两张表格在哪里，省去了我寻找的麻烦。感谢以上两位同学的帮助，以及助教老师的修正。

建议嘛，这次没太多问题，所以也没啥想法，因此也没啥建议。