

# 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：3 实验名称：多周期处理器设计（上）

## 一、代码以及波形图

testbench 部分先设置一个外部存储器，存储好所需数据，前 48 个存储了 12 条指令 Instruction，100 后存储了 2 个准备好的数据并提供写入数据的地址，10ns 执行一道指令（5#clk=~clk），指令依次为 load, load, add, store, sub, store, slt, store, and, store, or, store。

下图（图 0），为 multi\_cycle\_cpu 中的变量定义，之后有图片对应主程序，Outputs Control, ALU control, alu, regfiles, 以及 Mem 外部存储器的相应数据信号截图。

图 0

```
module multi_cycle_cpu(rst,clk,Address,MemWrite,Write_data,MemRead,Read_data);
    input rst,clk;
    input [31:0]Read_data;
    output MemWrite,MemRead;
    output [31:0]Address,Write_data;

    reg [31:0]PC,ALUOut,A,B,Instruction,MemWdata;
    wire [31:0]pc;
    //Outputs Control
    wire PCWrite,PCWriteCond,IorD,MemRead,MemWrite,IRWrite,MemtoReg,ALUSrcA,RegWrite,RegDst;
    wire [1:0]PCSource,ALUOp,ALUSrcB;
    reg [3:0]NState,State;
    wire [5:0]Op;
    wire [9:0]state;
    //ALU control
    wire [5:0]FunctionCode;
    reg [2:0]ALUcontrol;
    //alu
    wire [31:0]C,D,Result;
    wire [2:0]ALUOp;
    wire Overflow,Zero,CarryOut;
    wire [31:0]Signextend,Signextend2;
    //regfiles
    wire wen;
    wire [4:0]raddr1,raddr2,waddr;
    wire [31:0]rdata1,rdata2,wdata;
```

图 0，信号定义，如图所示，先是输入输出部分，然后是一些寄存器用于随着时钟信号变化的数据，小写 pc 其实没多大用处，只是在 always 模块里面简写一下，个人代码写法==。然后是 Outputs Control 对各类信号的控制以及 State 和 NState 的处理，ALU control 跟单周期中一样（其实就是复制黏贴过来的），alu, regfiles。

图 1

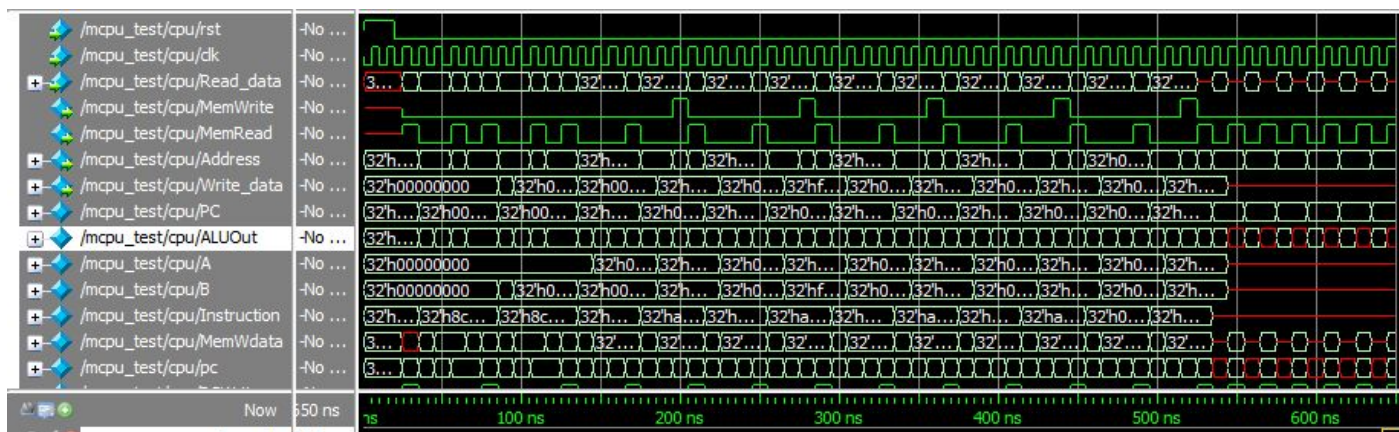


图 1，为 multi\_cycle\_cpu 基础的数据信号。

开头 20ns, rst 为 1, 没有读入, 红线部分, 然后执行设置好的 12 条指令。红线部分为, 未涉及数据所以没有信号位红色。

multi\_cycle\_cpu 连线代码如下:

```
//multi_cycle_cpu
always @(posedge clk or posedge rst)
begin
    if(rst)
        begin
            PC<=32'b0;
            A<=32'b0;
            Instruction<=32'b0;
            B<=32'b0;
            ALUOut<=32'b0;
            MemWdata<=32'b0;
            NState<=4'b0;
        end
    else
        begin
            if (PCWrite | (PCWriteCond & Zero))
                PC<=pc;
            else PC<=PC;
            if (IRWrite)
                Instruction<=Read_data;
            else
                Instruction<=Instruction;
            ALUOut<=Result;
            A<=rdatal;
            B<=rdata2;
            MemWdata<=Read_data;
            State<=NState;
        end
    end
end

assign Address= (IorD)? ALUOut:PC;
assign Write_data=B;
assign pc= (PCSource[1])? {PC[31:28], Instruction[25:0], 2'b00}: ( (PCSource[0])? ALUOut:Result );
```



图 2

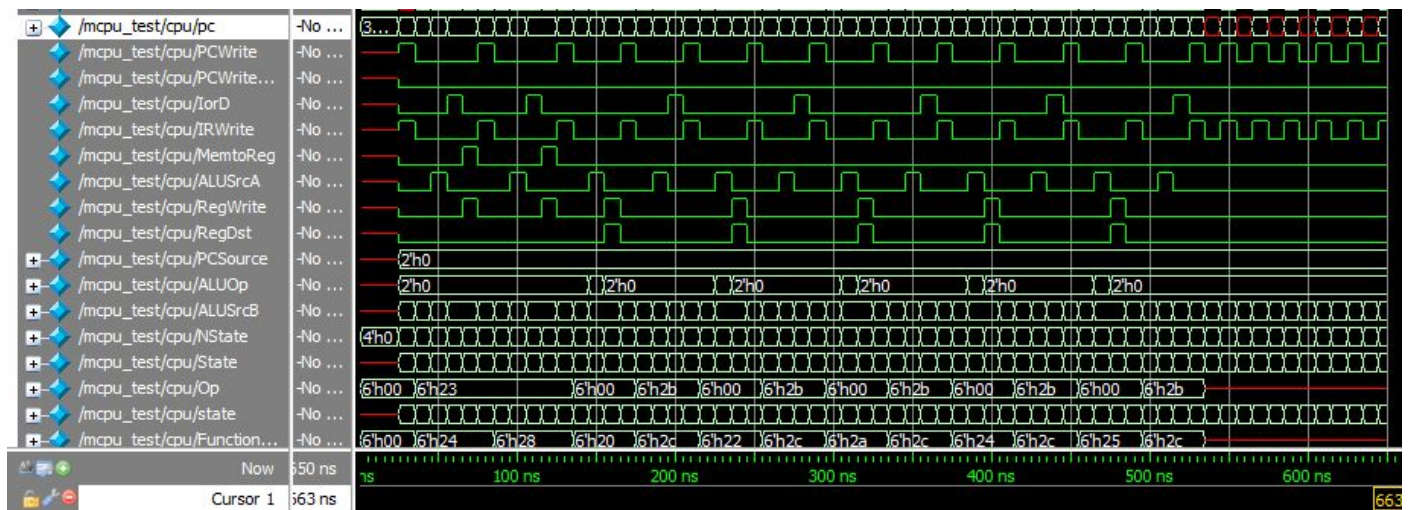


图 2 为 Outputs Control 的信号部分。本来是想再写一个 module 然后在 multi 里面调用这个，老师说没必要多写一个，然后这次只有一个 module，其中除了[9:0]state 是一个保存 State 当前状态的自己添加的变量，即 State 是 x(十进制)，那么 state[x]=1 其他都是 0

Outputs Control 代码如下。

```

assign state[0]=(State==4'd0)? 1:0;
assign state[1]=(State==4'd1)? 1:0;
assign state[2]=(State==4'd2)? 1:0;
assign state[3]=(State==4'd3)? 1:0;
assign state[4]=(State==4'd4)? 1:0;
assign state[5]=(State==4'd5)? 1:0;
assign state[6]=(State==4'd6)? 1:0;
assign state[7]=(State==4'd7)? 1:0;
assign state[8]=(State==4'd8)? 1:0;
assign state[9]=(State==4'd9)? 1:0;
assign Op=Instruction[31:26];
assign PCWrite=state[0] | state[9];
assign PCWriteCond=state[8];
assign IorD=state[3] | state[5];
assign MemRead=state[0] | state[3];
assign MemWrite=state[5];
assign IRWrite=state[0];
assign MemtoReg=state[4];
assign PCSource[1]=state[9];
assign PCSource[0]=state[8];
assign ALUOp[1]=state[6];
assign ALUOp[0]=state[8];
assign ALUSrcB[1]=state[1] | state[2];
assign ALUSrcB[0]=state[0] | state[1];
assign ALUSrcA=state[2] | state[6] | state[8];
assign RegWrite=state[4] | state[7];
assign RegDst=state[7];

always @ (Op or State)
begin
    case (State)
        4'b0000:NState=4'b0001;
        4'b0001:
            begin
                case (Op)
                    6'b000000:NState=4'b0110;//R-type
                    6'b100011:NState=4'b0010;//lw
                    6'b101011:NState=4'b0010;//sw
                    6'b000100:NState=4'b1000;//beq
                    default:NState=4'b0000;
                endcase
            end
        4'b0010:
            begin
                case (Op)
                    6'b100011:NState=4'b0011;//lw
                    6'b101011:NState=4'b0101;//sw
                    default:NState=4'b0000;
                endcase
            end
        4'b0011:NState=4'b0100;
        4'b0110:NState=4'b0111;
        default:NState=4'b0000;
    endcase
end

```

Timing diagram for the mcpu\_test module. The diagram shows various signals over a 600 ns period. Signals include ALUOp, ALUSrcB, NSState, State, Op, cpu/state, Function..., ALUcontrol, C, D, Result, ALUOp, Overflow, Zero, CarryOut, Signextend, Signexte..., and wen. The diagram shows the state of these signals during the execution of the test program, with values changing at specific time intervals.

代码如下。

```
//ALU control
assign FunctionCode=Instruction[5:0];
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

//alu
assign ALUOp=ALUcontrol;
assign C= (ALUSrcA)? A:PC;
assign D= (ALUSrcB[1])? ( (ALUSrcB[0])? Signextend2:Signextend):( (ALUSrcB[0])? 32'd4:B);
assign Signextend= (Instruction[15])? {16'hffff,Instruction[15:0]}:{16'b0,Instruction[15:0]};
assign Signextend2= (Instruction[15])? {14'h3fff,Instruction[15:0],2'b0}:{14'b0,Instruction[15:0],2'b0};
```



[illegible]


regfiles 代码如下。

Figure 5 shows a logic analyzer interface displaying a memory dump. The left pane lists memory addresses from [120] down to [100]. The main pane displays the corresponding hex values. A cursor is positioned at address [111], which contains the value 8'h00. The time axis at the bottom ranges from 0 to 600 ns.

63

## 二、 问题合集

这次的问题比上次单周期的多，但其实也没啥大问题，都是不够仔细造成的。根据隋老师提供的

 MultiCycle+CPU.pdf 文件中 36 页的表格 Control truth table，以及单周期复制黏贴过来的 ALUcontrol 部分代码，以及计算机组成原理实验指导中所提供的连线图，连线无遗漏错误，偷个懒借大腿的 testbench 边跑边发现错误的连线改改之后，就跑成功了。

### Problem1

```
MemWdata<=Read_data,
State<=NState;
end
end

//assign State=NState;
```

问题：一开始没有将 State 写在 always 模块里，没有根据时钟信号，导致程序逻辑上出错。

解决：将 State 的变量类型从 wire 改成 reg，并改入 always@（posedge clk or posedge rst）中了。

### Problem2

```
always @(posedge clk)
begin
  if(rst)
  begin
    r[0]=32'b0;r[1]=32'b0;r[2]=32'b0;r[3]=32'b0;
    r[4]=32'b0;r[5]=32'b0;r[6]=32'b0;r[7]=32'b0;
    r[8]=32'b0;r[9]=32'b0;r[10]=32'b0;r[11]=32'b0;
    r[12]=32'b0;r[13]=32'b0;r[14]=32'b0;r[15]=32'b0;
    r[16]=32'b0;r[17]=32'b0;r[18]=32'b0;r[19]=32'b0;
    r[20]=32'b0;r[21]=32'b0;r[22]=32'b0;r[23]=32'b0;
    r[24]=32'b0;r[25]=32'b0;r[26]=32'b0;r[27]=32'b0;
    r[28]=32'b0;r[29]=32'b0;r[30]=32'b0;r[31]=32'b0;
    //r[0]=32'b0;
  end
end
```

问题：本来写 regfiles 的时候，我并没有写 rst，当时老师也说写不写问题不大，但多周期老师要求 regfiles 要有 rst。

解决：在 regfiles 里加入 rst 部分，不敢用 for，就一句句写了，\ (ノ▽ノ) 。

### Problem3

```
//multi_cycle_cpu
always @(posedge clk or posedge rst)
begin
  if(rst)
  begin
    PC<=32'b0;
    A<=32'b0;
    Instruction<=32'b0;
    B<=32'b0;
    ALUOut<=32'b0;
    MemWdata<=32'b0;
    NState<=4'b0;
    //PC<=32'b0;
  end
end
```

问题：本来 rst 为 1 的时候，只初始化了 PC，后经同学提醒，应该将所有 always 块的东西都重置

解决：补充了其他数据的 rst 部分

### Problem ?~?

此处略去一些粗心的大小写拼写错误，以及粗心遗漏连线等问题

## 三、 对于此次实验的心得、感受和建议

在写过单周期处理器设计的基础上，这次的多周期处理器设计，没有增加很高难度。当然主要是老师把 pdf 资料给的很齐全，写的时候看图即可，完全不用去想里面的逻辑问题。Testbench 部分偷懒借用了谈清扬大腿的 testbench，通过跑 testbench 也能检查出很多错误，不过老师跑的跟我们的程序不一样，所以有些因程序差异导致的错误没有办法修正。

感谢谈清扬大腿同学提供的 testbench，以及破布黄博文助教的意见。

有了助教的联系方式，这次就是彻底没啥意见了，因为可以及时反馈嘛。