中国科学院大学计算机组成原理实验课

实 验 报 告

学号： 2018K8009929021 姓名：袁欣怡 专业：计算机科学与技术

实验序号：prj3 实验名称：内存及外设通路设计与处理器与处理器新功能评估

一、 逻辑电路结构与仿真波形的截图及说明（比如关键 RTL 代码段{包含注释}及其对应的逻辑电路结构、相应信号的仿真波形和信号变化的说明等）

1．三段式状态机

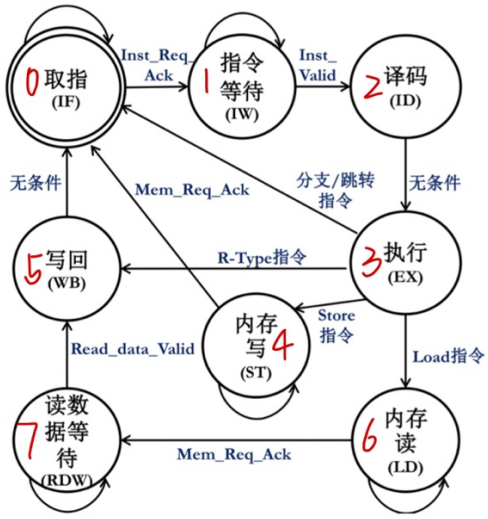
第一段：时序逻辑控制状态跳转

```
always@(posedge clk)
begin
        if (rst) state<=3'd0;
        else state<=nextstate;
end
```

在时钟上升沿触发跳转。

第二段：组合逻辑控制下一状态

```
always@(*)
begin
        case(state)
        3'd0: //IF
                if (Inst_Req_Ack) nextstate=3'd1;
                else nextstate=3'd0;
        3'd1: //IW
                if (Inst_Valid) nextstate=3'd2;
                else nextstate=3'd1;
        3'd2: //ID
                nextstate=3'd3;
        3'd3: //EX
                if (Branch||Jump) nextstate=3'd0;
                else if (Load) nextstate=3'd6;
                else if (Store) nextstate=3'd4;
                else nextstate=3'd5;
        3'd4: //ST
                if (Mem_Req_Ack) nextstate=3'd0;
                else nextstate=3'd4;
        3'd5: //WB
                nextstate=3'd0;
        3'd6: //LD
                if (Mem_Req_Ack) nextstate=3'd7;
                else nextstate=3'd6;
        3'd7: //RDW
                if (Read_data_Valid) nextstate=3'd5;
                else nextstate=3'd7;
        default:
                nextstate=3'd5;
        endcase
end
```



共有八个状态，因此令 state 和 nextstate 位宽均为 3 位，分别给每一个状态赋值为 0～7。

nextstate 的定义遵循状态机的变化规律，当对应路径的 Valid 和 Ack 信号同时拉高时握手成功，给 nextstate 赋值。

第三段：给各个信号赋值

```
always@(posedge clk)
begin
        if (rst) Inst_Req_Valid<=0;
        else if (state==3'd0) Inst_Req_Valid<=1;
        else if (state==3'd3 && (Branch||Jump)) Inst_Req_Valid<=1;
        else if (state==3'd4 && Mem_Req_Ack) Inst_Req_Valid<=1;
        else if (state==3'd5) Inst_Req_Valid<=1;
        else Inst_Req_Valid<=0;
end

always@(posedge clk)
begin
        if (rst) Inst_Ack<=1;
        else if (state==3'd1) Inst_Ack<=1;
        else Inst_Ack<=0;
end

assign MemRead=(state==3'd6)?1:0;
assign MemWrite=(state==3'd4)?1:0;
assign Read_data_Ack=(state==3'd7)?1:0;
```

其中需要注意的是，为了防止出现死锁，当 rst 信号为高时，Inst_Req_Valid 需为低，Inst_Ack 需为高。

2. PC 时序逻辑

```
always@(posedge clk)
begin
    if (rst) PC<=32'b0;
    else if (state==3'd3) PC<=PCnext;
end
```

rst 为高时 PC 保持为 0，否则在 EX 状态时跳转到新 PC。

3. 对 Instruction 和 Read_data 的处理

```
always@(posedge clk)
begin
    if (rst) instruction<=32'b0;
    else if (Inst_Valid && Inst_Ack) instruction<=Instruction;
end


always@(posedge clk)
begin
    if (state==3'd7) datal_load<=data_load;
end
```

和单周期处理器相比，Instruction 和 Read_data 都只保持一个时钟周期，导致我之前设计的 data_load 也只能保持一个周期，但是这两个数据可能在执行周期中被调用，所以用两个新的寄存器在存放它们的值。

4. Puts 函数

```
int
puts(const char *s)
{
        //TODO: Add your driver code here
        int i=0;

        while(s[i]!='\0'){
                while(UART_TX_FIFO_FULL & (*(uart+UART_STATUS/4)))
                        ;
                *(uart+UART_TX_FIFO/4)=s[i];
                i++;
        }

        return i;
}
```

5. 性能计数器和运行时间统计

```
assign mips_perf_cnt_0=cycle_cnt;
always@ (posedge clk)
begin
        if (rst) cycle_cnt<=32'd0;
        else cycle_cnt<=cycle_cnt+32'd1;
end
```

```
unsigned long _uptime() {
  // TODO [COD]
  //   You can use this function to access performance counter related with time or cycle.
  unsigned long *addr=(void *) 0x40020000;
  unsigned long val=*addr;

  return val;
}

static void bench_prepare(Result *res) {
  // TODO [COD]
  //   Add preprocess code, record performance counters' initial states.
  //   You can communicate between bench_prepare() and bench_done() through
  //   static variables or add additional fields in `struct Result`
  res->msec = _uptime();
}

static void bench_done(Result *res) {
  // TODO [COD]
  //  Add postprocess code, record performance counters' current states.
  res->msec = _uptime() - res->msec;
}
```

二、 实验过程中遇到的问题、对问题的思考过程及解决方法（比如 RTL 代码中出现的逻辑 bug，仿真、本地上板及云平台调试过程中的难点等）

1. 和单周期处理器相比，将一些寄存器的赋值从组合逻辑转化成了时序逻辑，这会导致波形上的变化比原先晚一个周期。当别的信号的赋值语句跟它们有关时，有可能导致赋值失败。
2. 在实验中遇到了本地仿真波形正确，但是上板 hit bad trap 的情况。跟同学讨论后认为，可能是因为在 always 逻辑块中添加了自赋值语句，在上板时产生了错误。以后要注意语法的规范性。

三、 对讲义中思考题（如有）的理解和回答

volatile 关键字的作用：确保该指令不会被编译器忽略，而且每次都需要直接读值（该语句的值可能会发生改变）。

可能会用到 volatile 关键字的地方：

1．可能会访问非自动变量的中断服务子程序

2．并行设备的硬件寄存器

3．多线程应用中几个任务共享的变量

本次涉及的 printf 属于其中的第一项，其本质上为一次系统中断服务。假如用宏定义，因为宏可以控制全局，所以可能会导致问题。这里用 volatile 可以保证在 puts 函数调用时，调用值不是外部调用值，而是在程序内声明的值。

四、 在课后，你花费了大约 20 小时完成此次实验。

五、 对于此次实验的心得、感受和建议（比如实验是否过于简单或复杂，是否缺少了某些你认为重要的信息或参考资料，对实验项目的建议，对提供帮助的同学的感谢，以及其他想与任课老师交流的内容等）

实验本身并不困难，在理解了实验意图之后，很快就可以构建基本的框架。难点在于没有了 prj2 时候的错误提醒，一开始并不知道要怎么确认错误在哪。即便后来知道要对照反汇编文件来差错，debug 时间仍大大延长。最后还遇到了 bf 无法通过的情况，也花了很多时间尝试各种方法。

感谢张爱瑶同学在我 debug 时帮助我，解决了我不少困惑。