实验 1 — 流水线 CPU

姓名：汤尧 学号：3200106252 学院：计算机学院

课程名称：计算机系统 II 同组学生姓名：无

实验时间：周四 3，4，5 节课 实验地点：紫金港机房 指导老师：卢立

一、 实验目的和要求

1. 理解流水线的基本概念与思想
2. 基于在单周期 CPU 中已经实现的模块，实现 5 级流水线框架
3. 理解流水线设计在提高 CPU 的吞吐率，提升整体性能上的作用和优越性
4. 拓展指令，理解不同类型指令在流水线中的运作差异
5. 实现功能基本完善的流水线 CPU

二、 实验内容和原理

2.1 实验内容

1-1 实现 addi 和 nop 指令，能正常运行给出的测试程序，仿真结果正确。

1-2 在流水线 CPU 中加入以下的 RV32I 指令：lui, jal, jalr, beq, bne, lw, sw, addi, slti, xori, ori, andi, srli, srai,。实现指令。通过仿真测试和上板验收。

2.2 设计模块

2.2.1 流水线与单周期 CPU 设计对比

关于流水线的基本设计思想已经在计算机体系结构课程的理论部分中提及，这里不再赘述。但实际落实到电路设计，与此前计算机组成中设计的单周期 CPU 相比还是有很多不同之处。

在同一时刻下，流水线的五个流水级运行的指令实际各不相同。因此，我们一般将一条指令所需的控制信号一起存在流水段间寄存器中，随指令其他信息一起顺着流水线传递，这里的段间寄存器是指：IF/ID 寄存器、ID/EX 寄存器、EX/MEM 寄存器、MEM/WB 寄存器这四个用于流水段间传递指令信息的寄存器组。例如 add 指令，在 ID

阶段对指令解码，得到所使用的寄存器 rd, rs, rt 需要等待到 WB 阶段写回，就需要由段间寄存器存储 rd, rs, rt 的值，一级一级地传递到 WB 阶段。

原则上每一拍过后，流水线都会向前流动一段。第 x 拍 IF 段内取出指令，第 x+1 拍该指令流入 ID 段进行译码，例如译码得到的是 lw 指令，那么第 x+2 拍流入 EX 段内指令 lw 完成地址计算，第 x+3 拍 lw 流入 MEM 段完成访存的读取，第 x+4 拍流入 WB 阶段将访存中的值写回寄存器。一般而言段内任务需要当拍完成，因为在下一拍时，段内的信号就会变为下一条流入指令需要的信号，在使用段间寄存器以外的寄存器时，务必要注意可能产生的时序问题。

	Single-cycle	Pipeline
模块复用	结构冲突	结构冲突
街段寄存器	无	有
状态机	无	无
时钟频率	取决于最长指令	取决于最长阶段
CPI	1	1（极限值）

表1：2种 CPU 设计对比

其他关于单周期、流水线 CPU 的特点对比如表 1 所示。

2.2.2 流水级划分

流水级划分的一般经验准则：

1. **按功能划分**。不同的功能最好属于不同的流水级，因为一般情况下，单个流水级功能越少，组合逻辑越简单，延迟越低，时钟频率越高。
2. **按时间划分**。在有些组件中（例如译码器），可能会出现较长连续的组合电路；它们往往会成为系统性能的瓶颈，但拆分组合电路、在中间插入阶段寄存器，并不会对电路的功能造成影响。由于流水线时钟频率取决于最长阶段，一般情况下，阶段越平均，CPU 的整体利用率越高，性能越高。
3. **按空间划分**。空间上远离的组件应该尽可能属于不同的流水级，使用寄存器隔断往返的长距离通信带来的时间开销。在 FPGA 板上，由于各类硬件资源已经固定而无法任意挪动，按空间划分显得更为重要。

根据经验准则，我们可以构造最基础的 5 段流水线设计（取指、译码、执行、访存、写回），在此基础上，我们可以根据实际布线中观察到的时间开销有针对性地进行优化。

三、 主要仪器设备

- HDL：Verilog、SystemVerilog
- IDE：Vivado
- 开发板：NEXYS A7

四、 操作方法与实验步骤

4.1 操作方法

完成单周期 CPU 设计，并将其通过 verilog 电路实现，Run Simulation 进行仿真并观察其波动情况。

4.2 实验步骤

1 使用开发工具建立工程，推荐在 Vivado 2019.2 以上版本完成。

2 根据参考设计图或自己设计的设计图搭建完整的流水线加法机,继续使用 Nexys7 对应的外设进行实验，实现 lab1 的流水加法机替换 lab0 中的 CPU 模块。请注意 Memory 必须使用 Vivado 提供的 Block RAM IP 核完成，以免 LUT 资源不够支持 CPU 设计。

3 在本实验中不要求对数据通路进行专门的封装，推荐直接将 Control Unit (Decoder) 视为译码模块放在 ID 段中。

4 进行仿真测试，以检验 CPU 基本功能。

5 调整时钟频率，确保时钟周期长度不小于最长流水段的信号延迟。

6 进行上板测试，以检验 CPU 设计规范，上板测试调试工具相关内容请参考 lab0。

五、 实验结果与分析

5.1 主要代码片段

状态寄存器相关连线

```
//----------------ID define start line----------------
wire [31:0] id_pc, id_inst;
wire [3:0] id_alu_op;
wire [1:0] id_mem_reg, id_pc_src;
wire id_alu_src, id_b_type, id_reg_write;
wire id_mem_read, id_mem_write;
wire [31:0] id_read_data_1, id_read_data_2;
wire [31:0] id_extend;
wire [4:0] id_write_register;

//----------------EX Define start line----------------
wire [3:0] ex_alu_op;
wire [1:0] ex_mem_reg, ex_pc_src;
wire ex_alu_src, ex_b_type, ex_reg_write;
wire ex_mem_read,  ex_mem_write;
wire [31:0] ex_read_data_1, ex_read_data_2, ex_pc;
wire [31:0] ex_extend;
wire [4:0] ex_write_register;
wire [31:0] ex_alu2res,ex_ALUres,ex_mux1_out;
wire ex_ALUzero;
```

将状态寄存器封装成模块

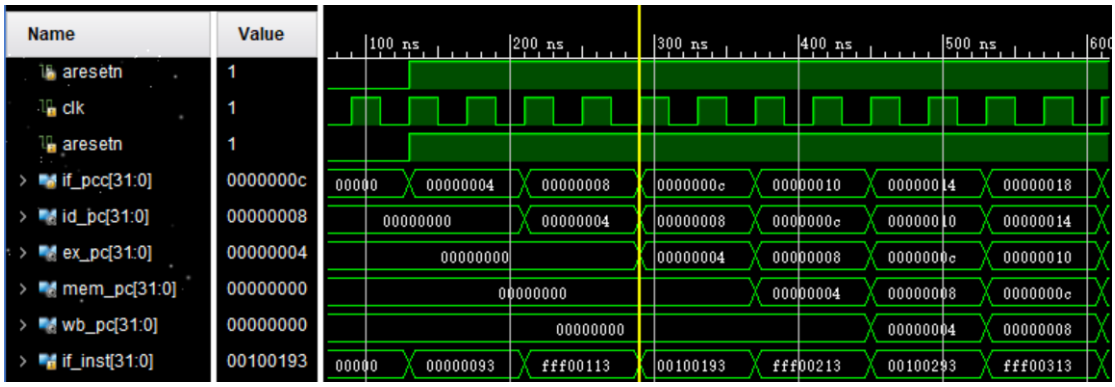
```
MEM_WB mem_wb(
    .clk(clk),
    .rst(rst),
    .mem_pc(mem_pc),
    .mem_reg_write(mem_reg_write),
    .mem_write_register(mem_write_register),
    .mem_read_data(mem_read_data),
    .mem_ALUres(mem_ALUres),
    .mem_mem_reg(mem_mem_reg),

    .wb_pc(wb_pc),
    .wb_reg_write(wb_reg_write),
    .wb_write_register(wb_write_register),
    .wb_read_data(wb_read_data),
    .wb_ALUres(wb_ALUres),
    .wb_mem_reg(wb_mem_reg)
);
```

在模块中添加时序逻辑电路，实现不同模块间状态寄存器的功能。

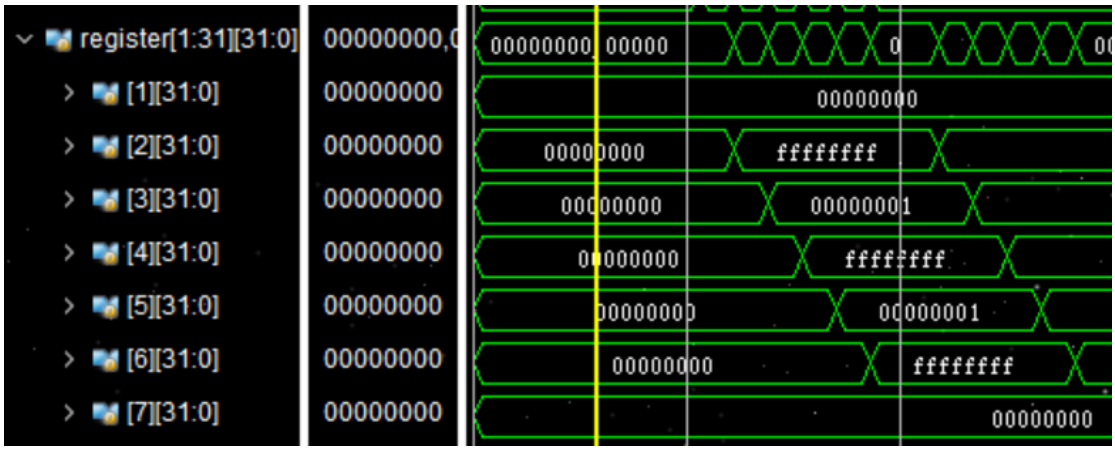
```
module IF_ID(
    input clk,
    input rst,
    input wire [31:0] if_pc,
    input wire [31:0] if_inst,
    output reg [31:0]  id_pc,
    output reg [31:0]  id_inst
    );

      always @(posedge clk or posedge rst) begin
         if(rst) begin
         id_pc <= 32'b0;
         id_inst <= 32'b0;
         end
         else begin
         id_pc <= if_pc;
         id_inst <= if_inst;
         end
      end
endmodule
```

5.2 仿真模拟

不同模块的 pc 随时钟周期依次传递。



使用测试代码，寄存器变化和预想结果一致。



六、 思考题

1-1 流水加法机

1. 对于 part1 (2-14 行)，请计算你的 CPU 的 CPI，再用 lab0 的单周期 CPU 运行 part1，对比二者的 CPI。

lab1 CPI = 1370ns / 12 = 114 ns

lab0 CPI = 1140ns / 12 = 95 ns

流水线的 CPI 比单周期的 CPI 大，单周期一个时钟周期执行一个指令，而流水线一个时钟周期走一个阶段，一条指令被分为 5 个阶段，故总运行时间较长。出现此情况的原因是时钟信号周期过长，cpuclk 调为最小时仍可以在一个时钟周期内执行完成一条完整指令，而流水线中每个模块运行所用时间其实比一个周期短很多，故会出现此结果。在实际运行的 cpu 中，流水线 cpu 的时钟周期会比单周期短很多。

2. 对于 part2 (24-39 行)，请计算你的 CPU 的 CPI（假设 nop 不计入指令条数），再用 lab0 的单周期 CPU 运行 part2，对比二者的 CPI。试解释为何需要添加 nop 指令（提示：如果不添加，会导致什么问题？）。

lab1 3050 – 1530 = 1520 ns

CPI = 1520/16 = 95 ns

lab0 2900 – 1620 = 1280 ns

CPI = 1280/15 = 80 ns

不添加 nop 时下一条指令需要读取的寄存器值在读取前还未改变，即数据冲突，会导致取值错误使得运算错误。

1-2 指令拓展

没有出现因 nop 不足而存在冲突的情况，部分指令 nop 数量多于所需必须数量。如因在我设计的 CPU 里 pc 在 mem 阶段写回，bne 和 jal 后只需 3 个 nop 即可跳转成功，不需要 5 个 nop。注意：改写时需要将跳转的偏移地址一并改掉，删去 nop 会导致跳转中指令条数减少，偏移地址改变。