计算机体系结构实验课程第三次综合实验报告

实验名称	静态5级流水线CPU优化			班级	李雨森老师
学生姓名	郭裕彬	学号	2114052	指导老师	董前琨
实验地点	A304		实验时间	2023.10.30、11.6	

实验目的

请参考实验指导书的课程设计扩展的内容，分析五级流水线处理器的阻塞情况，实现功能或性能优化，包括但不限于以下几方面：

1、使用前递技术减少阻塞情况；

2、针对分支跳转指令，实现不阻塞后续指令执行；

3、针对分支调整指令，考虑加入分支预测，提升流水线性能；

4、修改流水线结构，按照自己的想法设计流水线。

5、补充MIPS指令，扩展CPU功能。

注意：以上5点挑1~2点完成，在实验报告中详细说明自己的优化思路和流程，如果一条也没有完成，在实验报告中说明做了哪些尝试，发现了哪些问题即可。

实验内容：使用前递技术减少阻塞情况

背景

当流水线出现数据相关时，可能的情况是：

- 相邻指令间存在数据相关，即流水线译码、执行阶段存在数据相关；
- 相隔一条指令的指令间存在数据相关，即流水线译码、访存阶段存在数据相关；

- 相隔两条指令的指令间存在数据相关，即流水线译码、写回阶段存在数据相关。

因此，我们需要考虑到这三种情况，进而对代码进行修改。在之前的实验中，我们统一使用气泡阻塞的方式，增加等待周期从而消除这些数据相关的影响，这大大加剧的流水线的阻塞情况，因此我们尝试使用数据前递，也就是将计算结果从其产生处直接送到后续指令需要的地方参与指令执行，以避免流水线阻塞。

实现

decode.v

在译码模块上添加输入信号，分别对应上述三种情况下对应阶段的写使能、写数据。

```
    input       wire EXE_write,
    input       wire[ 31:0]EXE_wdata,
    input       wire MEM_write,
    input       wire[ 31:0]MEM_wdata,
    input       wire WB_write,
    input       wire[ 31:0]WB_wdata
```

数据前递意味着参与ALU计算，因此在alu操作数的赋值语句部分修改如下：

```
assign alu_operand1 =
(~inst_no_rs&&rs!=5'd0&&rs==EXE_wdest&&EXE_write) ? EXE_wdata:

 (~inst_no_rs&&rs!=5'd0&&rs==MEM_wdest&&MEM_write) ? MEM_wdata:

 (~inst_no_rs&&rs!=5'd0&&rs==WB_wdest&&WB_write) ? WB_wdata:
                          inst_j_link ? pc :
                          inst_shf_sa ? {27'd0,sa} :
                          rs_value;
assign alu_operand2 =     inst_imm_zero ? {16'd0, imm} :
                          inst_imm_sign ?  {{16{imm[15]}}, imm}:

 (~inst_no_rt&&rt!=5'd0&&rt==EXE_wdest&&EXE_write) ? EXE_wdata:

 (~inst_no_rt&&rt!=5'd0&&rt==MEM_wdest&&MEM_write) ? MEM_wdata:

 (~inst_no_rt&&rt!=5'd0&&rt==WB_wdest&&WB_write) ? WB_wdata:
                          inst_j_link ? 32'd8 :
                          rt_value;
```

这段赋值语句的作用是判断当前译码的指令需要的rs和rt寄存器是否就是当前EXE/MEM/WB阶段中要写入的寄存器，如果是，则将数据直接传递成为alu操作数，否则正常赋值。

同样地，我们也需要修改SW指令相关的取值语句：

```
assign store_data =
(~inst_no_rt&&rt!=5'd0&&rt==EXE_wdest&&EXE_write) ? EXE_wdata:

 (~inst_no_rt&&rt!=5'd0&&rt==MEM_wdest&&MEM_write) ? MEM_wdata:

 (~inst_no_rt&&rt!=5'd0&&rt==WB_wdest&&WB_write) ?
WB_wdata:rt_value;
```

此外，我们还要将原先设计的延迟等待的代码删除，并将ID_over信号的赋值语句进行修改：

```
//    wire rs_wait;
//    wire rt_wait;
//    assign rs_wait = ~inst_no_rs & (rs!=5'd0)
//                   & ( (rs==EXE_wdest) | (rs==MEM_wdest) |
(rs==WB_wdest) );
//    assign rt_wait = ~inst_no_rt & (rt!=5'd0)
//                   & ( (rt==EXE_wdest) | (rt==MEM_wdest) |
(rt==WB_wdest) );

assign ID_over = ID_valid & (~inst_jbr | IF_over);
```

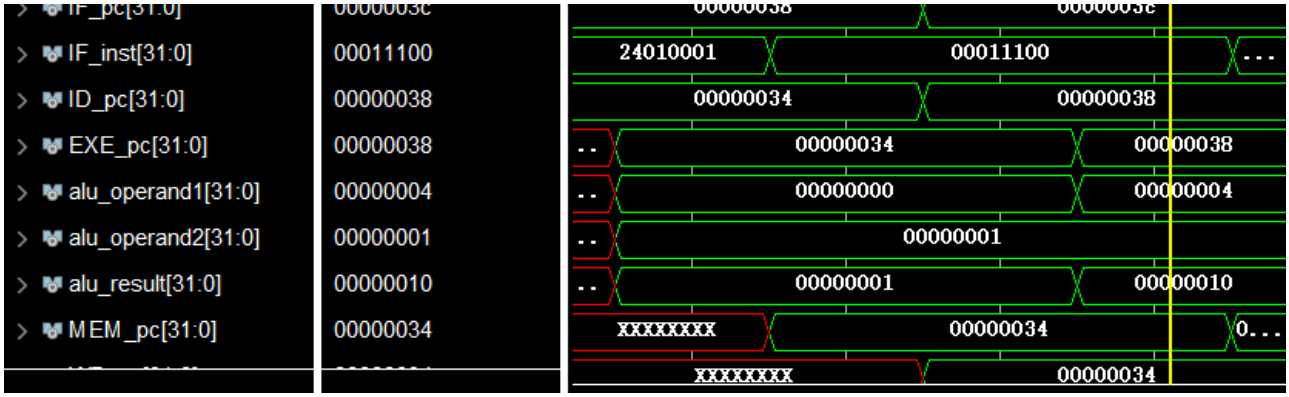
pipeline_cpu.v

主要是将对应的接口进行关联。

```
decode ID_module(                // 译码级
        .ID_valid   (ID_valid   ),  // I, 1
        .IF_ID_bus_r(IF_ID_bus_r),  // I, 64
        .rs_value   (rs_value   ),  // I, 32
        .rt_value   (rt_value   ),  // I, 32
        .rs         (rs         ),  // O, 5
        .rt         (rt         ),  // O, 5
        .jbr_bus    (jbr_bus    ),  // O, 33
//        .inst_jbr   (inst_jbr   ),  // O, 1
        .ID_over    (ID_over    ),  // O, 1
        .ID_EXE_bus (ID_EXE_bus ),  // O, 167

        //5级流水新增
        .IF_over     (IF_over     ),// I, 1
        .EXE_wdest   (EXE_wdest   ),// I, 5
        .MEM_wdest   (MEM_wdest   ),// I, 5
        .WB_wdest    (WB_wdest    ),// I, 5

        //展示PC
        .ID_pc       (ID_pc       ), // O, 32

        .EXE_write    (EXE_MEM_bus[116]),
        .EXE_wdata    (EXE_MEM_bus[67:36]),
```

```
        .MEM_write      (MEM_WB_bus[0]),
        .MEM_wdata      (MEM_WB_bus[37:6]),
        .WB_write       (rf_wen),
        .WB_wdata       (rf_wdata)
    );
```

经过仿真模拟测试，模拟的流水线CPU能够正常运行，且通过对对应值的监听可以看到，前递正常实现。如：



34H：addiu $1, $0,#1　　38H：sll　$2, $1,#4
可以看到，在34H还未进入到WB阶段时，执行38H的ALU模块已经获取到了即将写入1号寄存器的值1，以准备进行后续的计算。

需要注意的是，在之前的实验中为了避免同步CPU和指令阶段周期数造成的异常问题，我们直接令IF阶段固定执行三个周期，这使得本次实验中实现的前递技术并没有真正的减少阻塞，可以看到上图中38H进入EXE阶段迟滞于34H进入WB阶段。因此我们只是实现了这个功能，真正起到作用需要对本实验代码框架的取指阶段进行时钟周期数的优化。

实验内容：针对实验二发现的138H指令异常状况的分析

如图所示，根据

```
wire [31:0] mem_result; //MEM传到WB的result为load结果或EXE结果
assign mem_result = inst_load ? load_result : exe_result;
```

wire类型的mem_result在inst_load=1的时候，其取值应该是load_result对应的12345678，但图中竖直黄线所指时间mem_result的值却是exe_result的0x08000000。

进一步检查发现，mem_result值的更改发生在MEM_valid置位后的下一个周期，如图，当cpu_5_valid为0xff0f0后的下一个周期，mem_result从0x3c变为0x08000000。

所以我们分析，这也是由于上一次实验中单纯添加周期产生的valid信号、over信号紊乱的问题，但暂时没有想到好的解决方法。

实验内容：针对分支调整指令，考虑加入分支预测，提升流水线性能

针对这个实现思路，我们尝试了

```
module branch_predictor(
    input istaken,
    output prediction
);
reg [1:0] counter = 2'b10;
always @(posedge clk)
    begin
        if(istaken)
            begin
                if(counter<2'b11)
                    begin counter=counter+1'b1;
                    end
                else
                    begin if(counter>2'b00)
```

```
                        begin counter=counter-1'b1;
                        end
                    end
            end
    end
assign prediction = counter[1]?1'b1:1'b0
```

为原型的2位预测器，但并不能很好的与实验中的代码结构相契合，最终没有能够实现这个功能。