CAIC Lab4 Report

1 设计思路

这次lab的难度比之前大了不少，而且提示也少了一些，所以在此记录一些设计思路：

关于每个Reg存什么

~~每个reg需要存的东西就是这个reg需要存的东西。~~ 基本上是图上带有`*_if/id_ex/mem/wb`的信号都通过例化的模板寄存器进行存储与传输。这里的一个经验是，如果要写这样的重复的东西可以用python来生成……

数据前递与冒险处理

Rs1/2_fwd_ex信号的判断条件与课件上所示一致。

```
    //EX_MEM hazard && MEM_WB hazard
    if((reg_write_mem)&&(rd_mem!=5'b0)&&(rd_mem == rs1_ex))
        rs1_fwd_ex = `FWD_MEM;//From Slides P47
    else if((reg_write_wb)&&(rd_mem!=rs1_ex)&&(rd_wb == rs1_ex))
        rs1_fwd_ex = `FWD_WB;//From Slides P50
    else rs1_fwd_ex = `NO_FWD;
    if((reg_write_mem)&&(rd_mem!=5'b0)&&(rd_mem == rs2_ex))
        rs2_fwd_ex = `FWD_MEM;
    else if((reg_write_wb)&&(rd_mem!=rs2_ex)&&(rd_wb == rs2_ex))
        rs2_fwd_ex = `FWD_WB;
    else rs2_fwd_ex = `NO_FWD;
```

对于stall，由于暂停流水线仅需在取指令与译指令两阶段stall住就行，所以后面三个阶段的stall直接置0。而stall信号的判断是这样考虑的：（以`branch`为例，`jalr`类似）如果需要判断的数据在前一条指令中被rtype、itype等指令进行修改，那么需要在if，if_id插入`stall`指令，且在id_ex插入`bubble`指令进行冲刷，当`branch`的前一条指令得到的结果返回id进行判断后，运行`branch`后pc+4的指令或者是`branch`使跳转后的指令；较为麻烦的是`branch`前有load指令对需要比较的数据进行修改（也就是`mem_read_mem&&((rd_mem == rs1_id)||(rd_mem == rs2_id))`对应的情形），若是如此，则需要在ex进行stall后继续stall。

当出现需要跳转的指令(`branch`或`jalr`或`jal`)时，在if_id寄存器插入bubble，对后面的流水线进行冲刷。

```
    assign stall = (branch_id || jalr_id) && ((reg_write_ex&&((rd_ex == rs1_id )||
(rd_ex == rs2_id)))||(mem_read_mem&&((rd_mem == rs1_id)||(rd_mem == rs2_id))));
    assign bubble = (branch_id || jalr_id || jal_id);

    assign stall_if = stall;
    assign stall_id = stall;
    assign stall_ex = 1'b0;
    assign stall_mem = 1'b0;
    assign stall_wb = 1'b0;

    assign bubble_if = 1'b0;
```

```
    assign bubble_id = bubble;
    assign bubble_ex = stall;
    assign bubble_mem = 1'b0;
    assign bubble_wb = 1'b0;
```

数据通路的变化&分支前递

与助教给的结构图大致一致。将分支预测提前到id阶段来执行，需要借助`rs1/2_fwd_id`的判断在操作数与`reg_write_data_mem`之间进行选择。这里注意了零寄存器不可被改写的特质导致涉及它的一些冒险不是真正的冒险。

```
    if(reg_write_mem&&branch_id&&(rd_mem!=5'b0)&&(rd_mem == rs1_id))
        rs1_fwd_id = `FWD_MEM;
    else if(reg_write_mem&&jalr_id&&(rd_mem!=5'b0)&&(rd_mem == rs1_id))
        rs1_fwd_id = `FWD_MEM;
    else rs1_fwd_id = `NO_FWD;

    if(reg_write_mem&&branch_id&&(rd_mem!=5'b0)&&(rd_mem == rs2_id))
        rs2_fwd_id = `FWD_MEM;
    else rs2_fwd_id = `NO_FWD;
```

但要注意的是对regfile进行写的只能是`reg_write_data_wb`，信号`reg_write_data_mem`只是在分支前递阶段使用，不会替换掉id阶段向ex阶段传递的`rs1/2_data`信号（踩过坑）。

另外一些模块的设计

增加了id_control.v文件，通过`rs1/2_fwd_id`信号的控制在regfile读出数据与前馈数据之间进行选择，并对id阶段判断跳转的信号`zero`,`less_than`等进行生成；

增加了forward_mux.v文件，于op_selector.v文件中实例化，通过`rs1/2_fwd_ex`信号的控制，对ex阶段中的操作数进行选择。

除此之外，借鉴课件中的思路，让寄存器在时钟下降沿写入，避免一些读写冲突。

在riscv.v中对reg_write_data_mem进行赋值，类似于wb模块中对reg_write_data_wb的选择赋值对其进行赋值。

2 痛苦Debug

这个lab的debug模块异常痛苦。总的来说，先要知道一步步在干什么，instr应该是什么、其他信号应该是什么，再看各种信号出了什么问题。实际上因为信号过多，连线的bug就断断续续找了挺久。以下是一些***意识流***的记录……

test 0

pc rst记得

Data_in出错->rs2_data_new出错->rs2_data出错（橙色是idmodule的）

在control unit里的`reg_write_addr = instr[11:7];`一句进行提前，实时更新

尤其需要注意的是id的reg_write，有一个out是往后传，有一个in是后面送过来的。一开始弄混了，debug了好久啊啊啊！找到这个bug之后除了branch（test3）都过了

test 1

由于add的指令比起一堆lw lb什么的好读一些，所以就先从test1开始。

发现了rd_mem存在的问题（在不应该变化时进行了变化），原来是一开始的设计中instr没经过reg直接去id了，，，

现在是14ns的时候rs2_data的选择出错，在ex阶段

test 3

极度痛苦

能用assign就不用时序，，

无论是否跳转，只要指令是branch类型，均将reg_src置为1。

test 2

调通branch之后 发现是没有SUBI却给了一个SUB的alu_type，这就是对itype不了解而妄下推断的结果

3 运行正确截图

提交id：5ff967926c3c