浙江大学

本科实验报告

课程名称：计算机体系结构

姓　　名：官泽隆

学　　院：计算机科学与技术学院

系：

专　　业：计算机科学与技术

学　　号：3180103008

指导教师：翁恺

2020 年 11 月 29 日

浙江大学实验报告

课程名称：计算机体系结构 实验类型：综合

实验项目 **PCPU with stall and forwarding**

学生姓名：官泽隆 专业：计算机科学与技术 学号：3180103008

同组学生姓名：/ 指导老师：翁恺

实验地点：曹光彪二期-301 实验日期：2020 年 11 月 29 日

一、 实验目的和要求

Objective:

1. better understanding about how to eliminate/mitigate hazards using stall and forwarding.

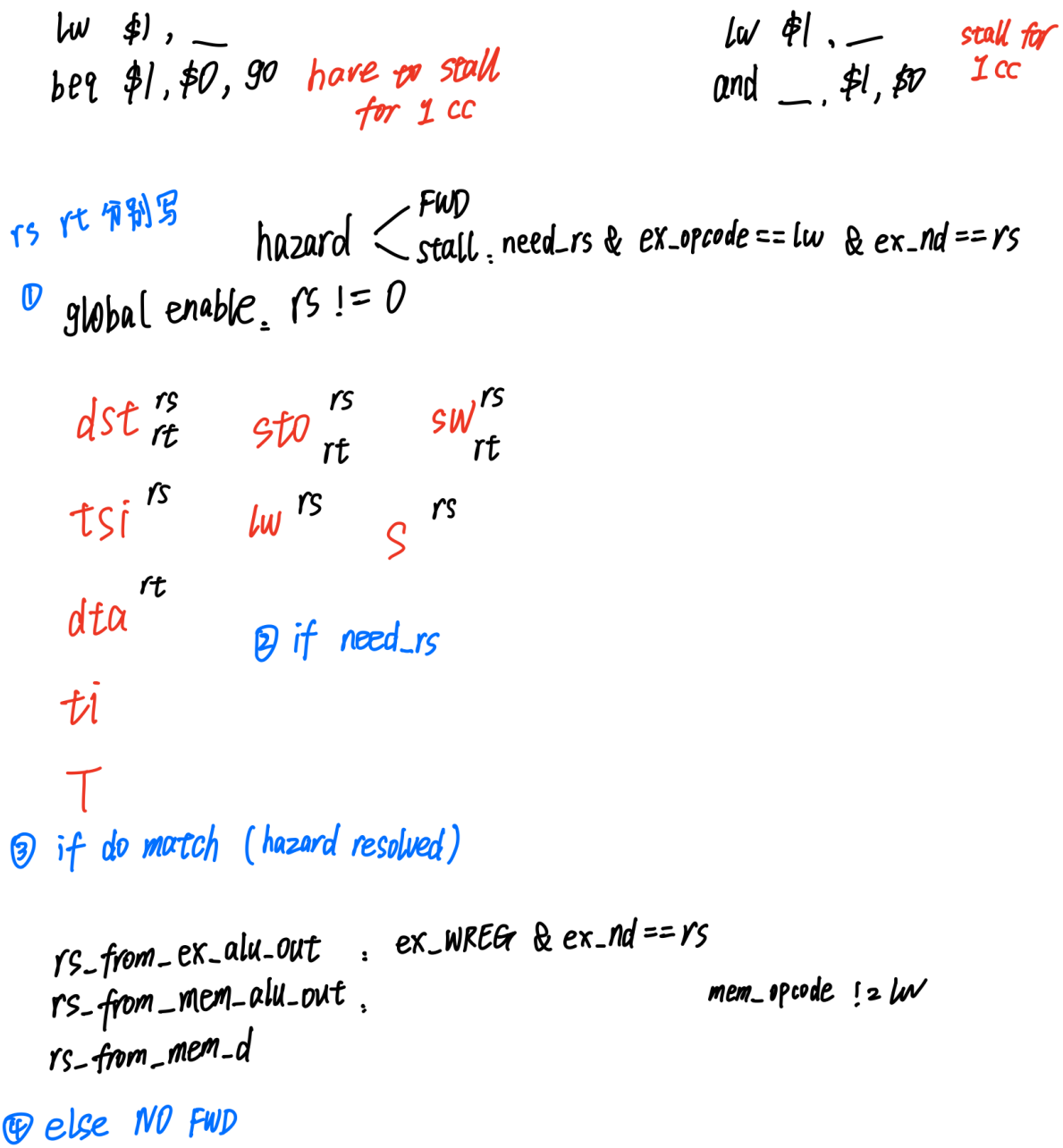
Task:

1. eliminate 3 kinds of data hazards with forwarding
2. modify stall logic, improving efficiency.
3. In the VGABased-Debugger project, replace module *mips* with your own PCPU
4. Validate the design of your own PCPU on SWORD Board

二、实验内容和原理

原理：

1. forwarding



①中，我们按指令的 operands 给指令分了个类，

```
enum Opr_T
{
    dst, // rd, rs, rt
    tsi, // rt, rs, imm
    dta, // rd, rt, sa
    S,   // rs
    tob, // rt, offset(base)
    ti,  // rt, imm
    sto, // rs, rt, offset
    T    //target
};      // oprand contained in one instruction
```

例如（不全，还有andi, xori, jalr这种也有实现）：

```
unordered_map<string, Opr_T> op_opr{
    {"add", dst}, {"sub", dst}, {"and", dst}, {"or", dst}, {"slt", dst},
    {"sll", dta}, {"srl", dta},
    {"jr", S}, // R-type
    {"addi", tsi}, {"ori", tsi}, {"slti", tsi},
    {"lw", tob}, {"sw", tob},
    {"lui", ti},
    {"beq", sto}, {"bne", sto}, // I-type
    {"j", T}, {"jal", T}, // J-type
};
```

并据此得出源操作数的依赖关系②

```
assign need_rs = dst | sto | sw | tsi | lw | S;
assign need_rt = dst | sto | sw | dta;
```

如③，可以依据 pipelined 的控制信号，区分源操作数在流水线级中的位置

以 rs 为例的转发选择信号 FWDA，stall 控制信号 FQ1_stall：

```
always @(*) begin
    FWDA = 0;
    FQ1_stall = 0;
    if (rs && need_rs) begin
        if (ex_WREG && ex_nd == rs) begin
            if (ex_M2REG) FQ1_stall = 1;
            else FWDA = 1;
        end
        else if (mem_WREG && mem_nd == rs) begin
            FWDA = mem_M2REG ? 3 : 2;
        end
    end
```

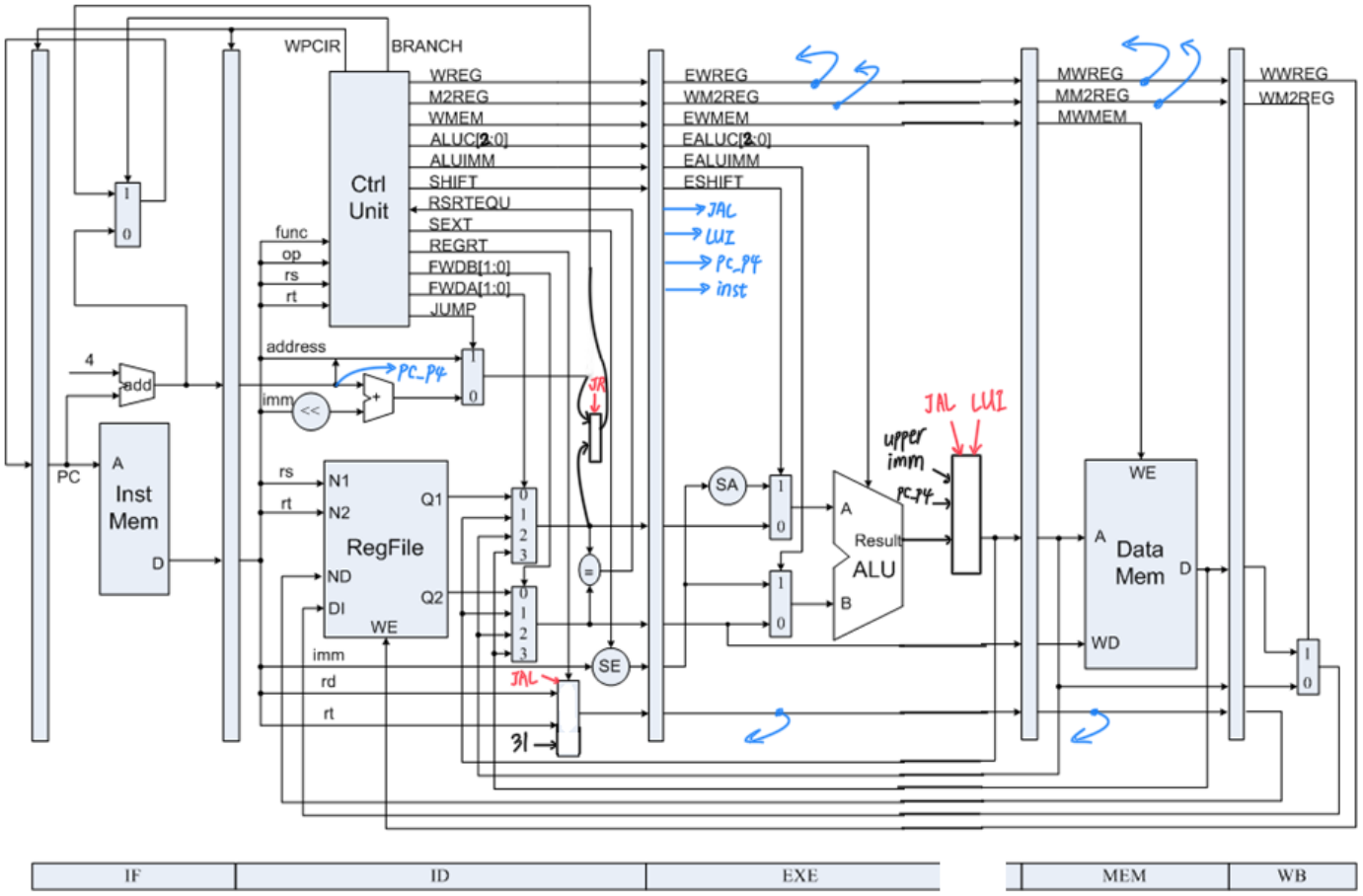
示意图见下一页：

注：新增的手绘线，黑色线表示数据，红色线表示控制，蓝色线表示向后流水的或是向前转发的信号

略去 ID.stall 的线

ID 级的 Critical Path 毫无疑问来自转发，因此我暂不处理 jal 的 pc+4, lui 的 upper_imm，而是流水到 EX 级。但又要尽量减少转发时源的数量，因此多路选择逻辑放在 MEM 级暂存器前，这样下来为 ID 级节省了一个四选一多路器的延迟

事实上，蓝色线对应的信号并不需要我去手动引线。得益于之前实验部署的自动生成脚本，只需更改了 IdStage 和 ExStage 两个模块的代码，把 fwd 需要的信号作为新增的 input；重新调用脚本生成，就可以保证级间寄存器的信号不多不少，以及 PCPU 例化时，级间接线正确。

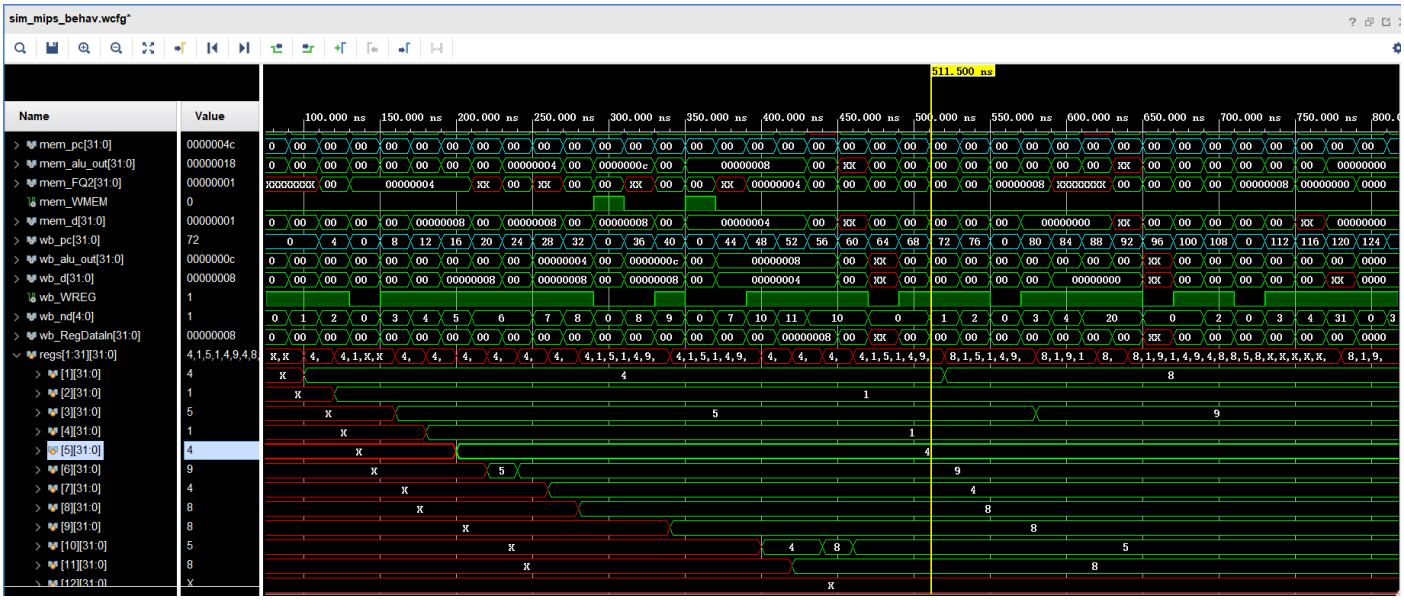


跳转为 delayed branch（长度为 1 个指令的延迟槽）模式，在 ID 解析完毕。

三、实验过程和数据记录

1. 行为仿真——助教给的测试程序

对应 sim_mips.v



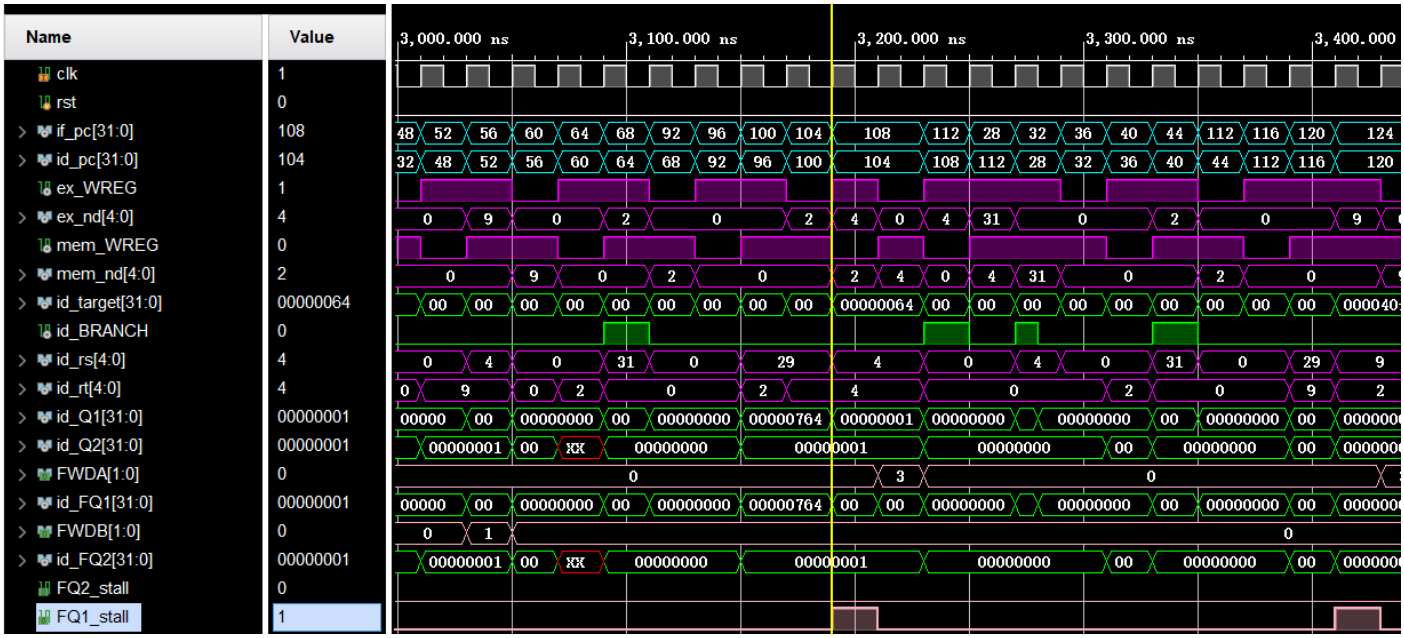
2. 行为仿真——黑盒测试 Fibonacci

路径：这一测试的程序欲计算 Fib(14)，设置入口环境，设置出口死循环。观察结果。

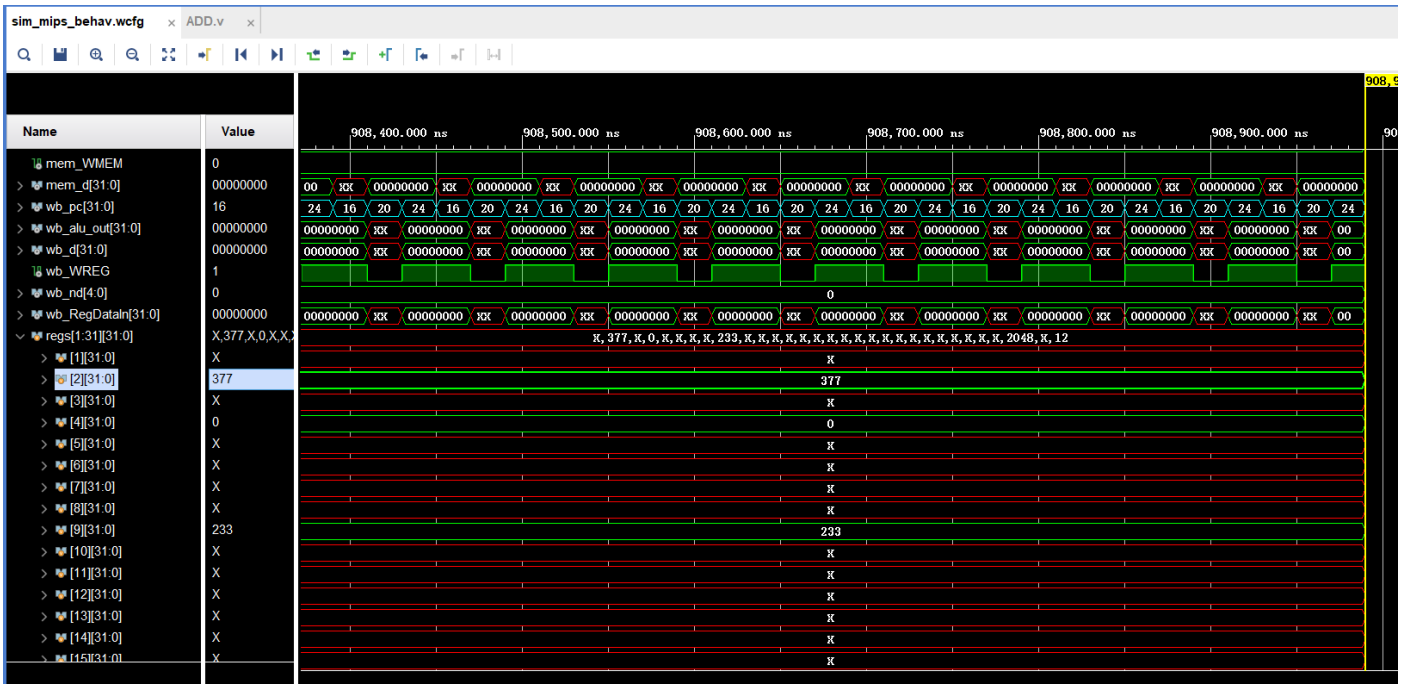
测试用程序 fib.s 随附。

测试程序 fib.s 验证了最容易出错的一些功能：sw, lw, j, jal, jr, bne.

lw 后接 ALU 指令导致的 stall+fwd



得出 Fib(14)=377 正确！



The Rule

The Fibonacci Sequence can be written as a "Rule" (see Sequences and Series).

First, the terms are numbered from 0 onwards like this:

n =	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	...
x_n =	0	1	1	2	3	5	8	13	21	34	55	89	144	233	377	...