实验 5-3 报告

学号：2016K8009929011
姓名：段江飞

一、实验任务（10%）

本实验要求将 CPU 顶层的对外接口信号修改为一个 AXI 接口，CPU 通过 AXI 接口取指令和数据。在实验 5-1 的基础上，利用完成的类 SRAM-AXI 转接桥，将 CPU_CORE 通过类 SRAM 接口连接到转换桥，然后转换桥仲裁指令和数据请求，从 AXI 接口的 RAM 中取指令和取数据。5-3 要求实现的 CPU 能从**随机延迟**的 RAM 中正常取指令和取数据，并通过之前的全部功能点测试。

二、实验设计（30%）

NOTE：实验设计和之前的一样诶，最关键的导致上板过不了的 bug 在错误中有写。

（一）CPU 结构的改变

之前写的流水线控制信号，一直都是当前级的信号由上一级的流水线控制信号控制，就是下面这种情况：

```
always@(posedge clk)
begin
    if(ID_to_EX_valid & EX_allowin)
        ResM <= alu_result;
end
```

这是由于之前设计的 CPU，这里的每一级的流水线信号(valid, readygo, allowin)表示的其实是当本级有新的有效信号进来时，才进行流水，这样每一级的有效与否并不是像讲义上标准写法那么显而易见，而且阻塞不好处理，当时这么写是因为对那个标准写法没完全理解，自以为得其意，就按自己理解写了代码，导致在一段时间内，对别人的流水线信号产生了迷惑，但终究还是一条路走到了黑，一直按这样写了下去。然而情况总是朝着最坏的方向发展，这样写一直没出问题，我觉得也是对的，但实际上只是问题还不够大，还不足以给我一记暴击，等到了总线，我想了半天感觉这个阻塞、等待实在是有些难以处理，再加上以前瞎写的阻塞，所有的问题爆发了，于是我想修改流水线的结构，任重而道远。

简单来讲也就是将之前的那样错位的控制信号修改为标准的当前级的流水线信号控制当前级的信号，也就是如下：

```
always@(posedge clk)
begin
    if(EX_to_MA_valid & MA_allowin)
        ResM <= alu_result;
end
```

这样每一级的 valid 就表示当前级是否有效，然后阻塞的时候就很好阻塞了。我最初准备先修改，然后通过

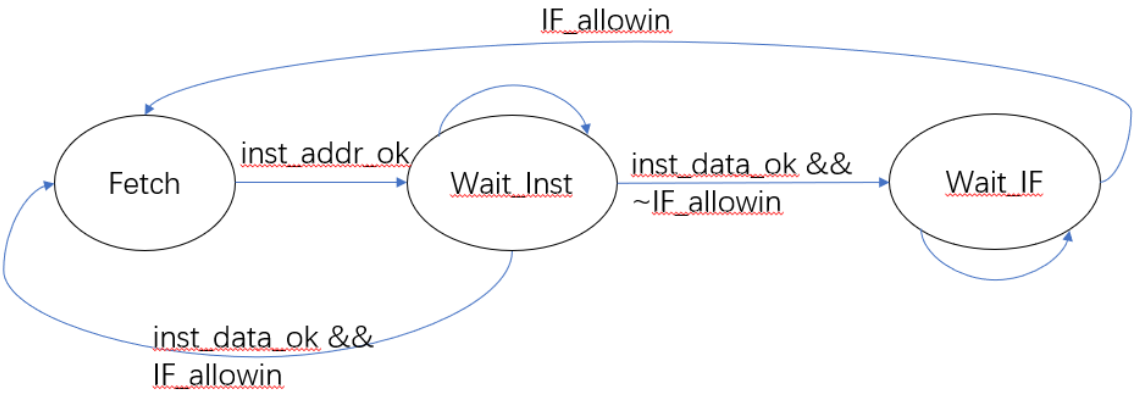
lab4 的测试，然后再去写总线，但是现实总是让人头疼，改起来是那么的复杂，中间出现的数据相关也不是很好处理（或许是太晚了，脑子不清楚了），于是决定放弃，不改了，就按原来的结构写总线，神挡杀神，佛挡杀佛。

开始写总线的时候，在开头写个状态机，这个状态机最初也没完全想清楚，然后又给访存加了一个状态机，。最初进行仿真的时候，刚开始还挺好，后来在阻塞的时候遇到了问题，我去检查我的逻辑，感觉很难处理，考虑到下周随机延迟，即便现在过了，下周可能也会爆炸，就又决定修改结构，不过这次是直接在总线实验里修改，遇到问题在直接 debug。流水线就是利用那个标准的流水线写法去写的了。

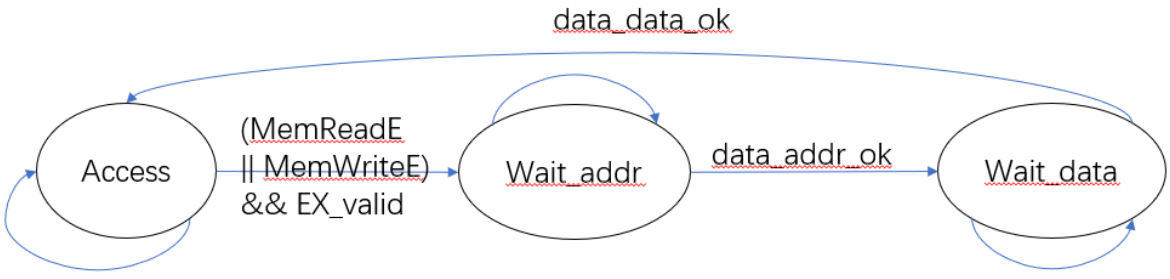
（二）总线

1、状态机实现

首先是取指和访存利用状态机控制发请求，这两个状态机主要的不同的在于起始状态不一样。指令的起始状态就是发取指请求，然后等待指令返回，如果指令返回了，这时候可以进行下一次取指令，就发取下一条指令的请求，否则，就等待可以发取下一条指令的请求。访存的状态机的其实状态是等待发访存请求，如果不需要，就直接流水，如果需要访存，就发访存请求，等待地址接受之后，进入等待数据返回的阶段，数据返回之后，就回到起始状态。



图一



图二

2、IR

由于利用了总线，取指和访存延迟都比较长，这样导致译码的时候，没有指令寄存器用起来不是很方便，而

且对译码级的阻塞也不是很好处理，于是添加了指令寄存器，译码利用指令寄存器中的指令进行译码。

3、例外处理

例外大概是加上总线之后，最难处理的一部分了。首先，考虑到例外提交时在访存级提交，例外发生时，一定不会进行访存，这时候可能正在进行取指，于是，例外的提交有两种处理方式：

一是例外直接提交，提交之后请流水线，这时候如果可以立即发取指请求，就将 PC 修改为例外处理入口，如果有一条指令正在取指，就再另外设置一个状态机，标记发生了例外提交了，等到下一次取指的时候，将 PC 修改为例外处理入口，同时这条取出来的指令也清除。

二是例外先不提交，而是阻塞在访存级，对访存的状态机进行修改，等待正在取指的指令取指完毕，将例外提交，PC 修改为例外处理入口地址。

eret 提交的处理和例外提交处理基本一致，而且更加简单。

对于时钟中断和软中断，我采取的是第一种方式，其他的例外，我所采用的是第二种方式，对访存级的状态机做了修改，添加了例外等待状态，因为例外发生时，到了访存级是不会访存的，这时候除了就比较简单了，直接进入一个等待状态，等待例外提交。

好吧，例外处理部分截止到此处，之前的是最初的设计，确实通过了仿真，但是…综合爆炸，出现了组合环，我又修改了设计，全部的例外处理都是利用第一种方式，直接提交，附加在访存级指令上，如果访存级指令无效，就等到访存级指令有效在提交。

（三）验证

1、仿真验证

在 vivado 上仿真，测试 94 个功能点，通过测试。

2、上板验证

数码管一侧从 1 加到 5E，高低 8 位同步累加，拨动拨码，数码管都能加到 0x5E，随机延迟上板通过测试。

（四）其他

1、电子表与记忆游戏

在 CPU 上试了电子表和记忆游戏，均能正常运行。

三、实验过程（60%）

（一）实验流水账

1、12 月 14 日下午 8 点到 9 点

写的 CPU 在 5-2 里上板过不了，感谢老师给我找出来 bug，修改了这个 bug 之后，上板直接过了随机延迟，然后写了实验报告，非常舒适。

2、12 月 14 日下午 9 点 30 到 11 点 30

试了电子表和记忆游戏，都能运行。

（二）错误记录

1、错误 1

（1）错误现象

仿真通过，上板不过。

（2）分析定位过程

我没找出来呀，老师找出来的隐藏 bug。

（3）错误原因

还是因为总线的原因，总线上的返回的数据都不会保持，我在代码里面利用了返回的信号 rid，把它当做一个会保持的值来用了，因为返回的时数据是 rid 为 1，但是 rid 不会保持，导致等到数据迟一点取走时，rid 就变成了 0，这样所以的 lw，lb，lh 之类的之类执行都会出现错误，同时，测试 store 之类的指令时，会用到 load，所以 store 之类的指令也会出错，这很符合上板现象，在 load 和 store 的功能点全错了，但是取指能正常。

另外，在 wait1s 里面，会 lw 设定的一个值，来判断循环此时，load 的出错，导致判断不能和预期一致，循环次数大大增加，导致每次数码管加 1 会需要很久很久。

（4）修正效果

通过了随机延迟测试，美滋滋。

四、实验总结

这次试验最关键的就改了 rid，非常感谢老师！！！