微处理器与智能芯片 lab1 report

钟作奇 2000012741

1 设计思路

RAM与ALU完全按照助教给出的图示进行设计，与助教给出的代码不同的点是，将ram和alu的output口声明为reg。对于top模块，添加wire型的data_in1 作为ram的out和alu的in1之间的连线。在`define.v`中，对alu的各操作进行了宏定义编码。

2 Debug 经验

难点攻克

本次lab的难点在于**地址的处理**与**符号数的处理**。对于前者，我在了解大端法的定义后，对`addr`变量进行处理，削去最后两位成为`address`作为数组中的索引，削去的两位`pos`保留了写byte、half word的位置信息。在这里特别要注意的是（以byte的write为例）`pos`为00对应`mem_core`中一个word（32位）的31-24位，01对应23-16位，以此类推。这也就是大端法书写过程中尤其需要注意的地方。因为声明时每个words编号是[31:0]，而左边的index对应的总是地址小端，所以出现了前面pos和index的这种对应关系。

对于后者，符号数的处理，主要在alu的符号数大小比较与符号数算数右移中（因为给的都是没有符号的一些2进制比特数？不知道称呼是否准确但先这么称呼）。对于符号数的大小比较，我先提取最高位的符号信息，然后分情况进行比较；对于符号数的算数右移，我先在符号数左边扩展了32位符号位，然后进行逻辑右移就ok了，因为如果此时符号位为0那么在前面补的就是0，符号位为1那么在前面补的就是1（实际上平台给的testbench无脑补1就行，但这是错误的）。在与同学的讨论中，我还发现了另外一种可行的写法：用`$signed()`进行类型转换后，比较与算数位移就不用我们自己去实现了。

经验教训

这是第二次写verilog程序+第一次写多文件的verilog项目，在磕磕绊绊的debug过程中，除了给了我一点小小的veribug震撼，我还是有了如下收获：

- tb中include了top和define那么iverilog给的指令的arg里只用写tb了（top里include了俩子模块），命令行需要敲的东西就会少一些。要注意的是不能循环include（你~~中有我我中有你~~），否则会报奇怪的错误（合理）

- `{32{alu_in1[31]}, alu_in1}`报错很久，后面发现改成`{{32{alu_in1[31]}}, alu_in1}`就没事了，一些奇怪的平级判断……
- 我对verilog里何时使用reg和何时使用wire可以说是通过测试出错后修改进行的，希望这段时间抽空看看书进行一下系统学习……
- begin和end，module和endmodule这些的匹配别忘了，分号别忘了（写python写久了……）
- 注意区分一般逻辑操作与按位逻辑操作，这在C中其实也有出现（比如!是整体，~是按位）
- 看波形图debug的技能有所提高，gtkwave对mac的适配可谓糟糕，vscode内置的波形图查看器勉强能用~~（只是最多八个波形多了要加钱）~~

3 提交结果

以上是测试结果，提交的id为1805a99a64fb。