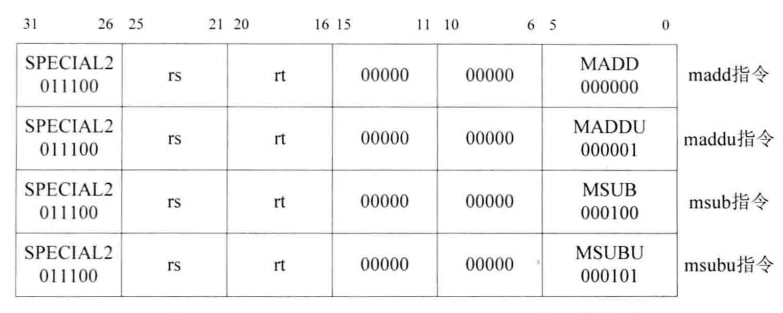
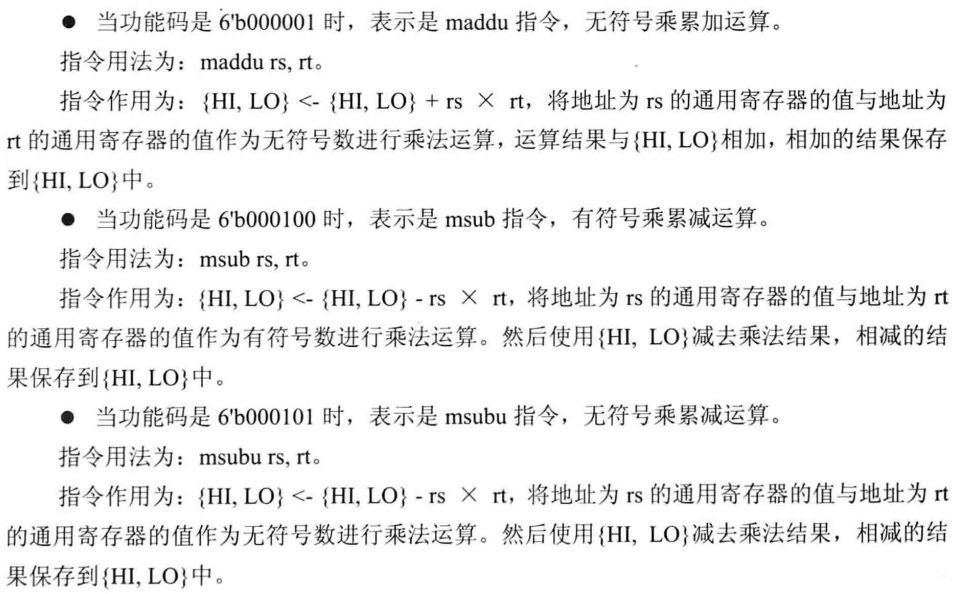
**乘累加和乘累减指令的实现**

# 指令说明

乘累加和乘累减指令包括四条：madd、maddu、msub、msubu，各指令格式如下表所示：





# 实现思路

我们在之前将CPU增加了流水线暂停机制，因此我们计划在流水线执行阶段采用两个时钟周期完成运算，第一个时钟周期执行乘法运算，第二个周期将乘法结果与HILO寄存器的结果相加/相减。

要实现乘累加和乘累减，要保存两个信息：当前的时钟周期，以及乘法结果。因此我们在EX/MEM模块中添加两个寄存器cnt、hilo，分别保存上述信息。修改后的系统结构如图所示：

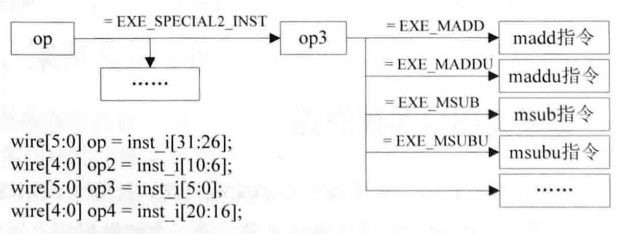
执行阶段EX模块的输出hilo\_temp\_o是乘法结果，传递到EX/MEM模块，并在下一个时钟周期送回EX模块，参与第二个时钟周期的加法/减法运算。

执行阶段EX模块的输出cnt\_o代表当前是第几个时钟周期，传递到EX/MEM模块，并在下一个时钟周期送回EX模块，后者据此判断当前处于乘累加、乘累减指令的第几个执行周期。

# 修改实现

## 3.1 修改ID模块

确定指令过程如图：



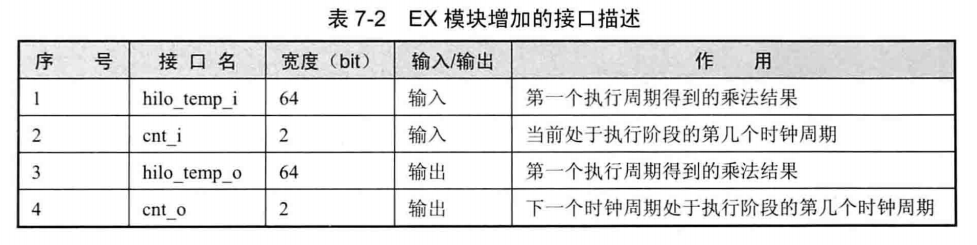
具体代码见**附录一**

值得注意的是：因为最终将结果是写入HILO寄存器，而不是写入通用寄存器，所以设置wreg\_o为WriteDisable。但是我发现了一个问题，将MADD和MADDU类型设置成MUL类型，在执行阶段这是保存到通用寄存器的操作，为什么这里MUL可以执行保存到HILO寄存器呢？

往后继续学习可能会有答案。

## 3.2 修改EX模块

增加的接口描述如下表所示：



具体代码见**附录二**

我们首先计算乘法结果，取出乘法操作的被乘数，指令、都是有符号乘法，我们用进行正负数转换，，同理第二个数也是一样，目的是全部转换成乘法运算，再根据两个乘数的符号来判断最后结果的正负。

然后我们得到了临时乘法结果，根据以下的几种情况来判断正负：

1. 若为有符号乘法、
   1. 乘数和被乘数为一正一负，则对最终结果求补码；
   2. 若两者同号，则最终结果不变；
2. 若为无符号乘法运算，最终结果也不变

另外，在乘累加乘累减指令中的流水暂停中，我们引入了新的变量，用来记录周期数，当处于第一个周期时，将设置为，当进入第二个周期时，此时EX模块驶入的就是上一个时钟周期得到的乘法结果，所以将与寄存器相加，保存到中，同时设置为，乘累加指令结束，设置为，而不是直接设置为，目的是，如果是其他原因导致流水线保持暂停，那么由于为，所以EX阶段不在计算，从而方式乘累加指令重复运行。

最后给出信号的值，目前只有乘累加乘累减指令会导致流水线暂停，所以就直接等于变量的值。

## 3.3 修改EX/MEM模块

增加的接口如下表所示：



具体代码见**附录三**

## 3.4 修改OpenMIPS模块

这部分没什么好说的，就是细心一点，变量的值不要弄混淆了，类似于接口的值是等等，具体代码见**附录四**。

# 测试及结果

我使用了下面的测试程序进行测试：

.org 0x0

.set noat

.global \_start

\_start:

ori $1,$0,0xffff

sll $1,$1,16

ori $1,$1,0xfffb # $1 = -5

ori $2,$0,6 # $2 = 6

mult $1,$2 # hi = 0xffffffff

# lo = 0xffffffe2

madd $1,$2 # hi = 0xffffffff

# lo = 0xffffffc4

maddu $1,$2 # hi = 0x5

# lo = 0xffffffa6

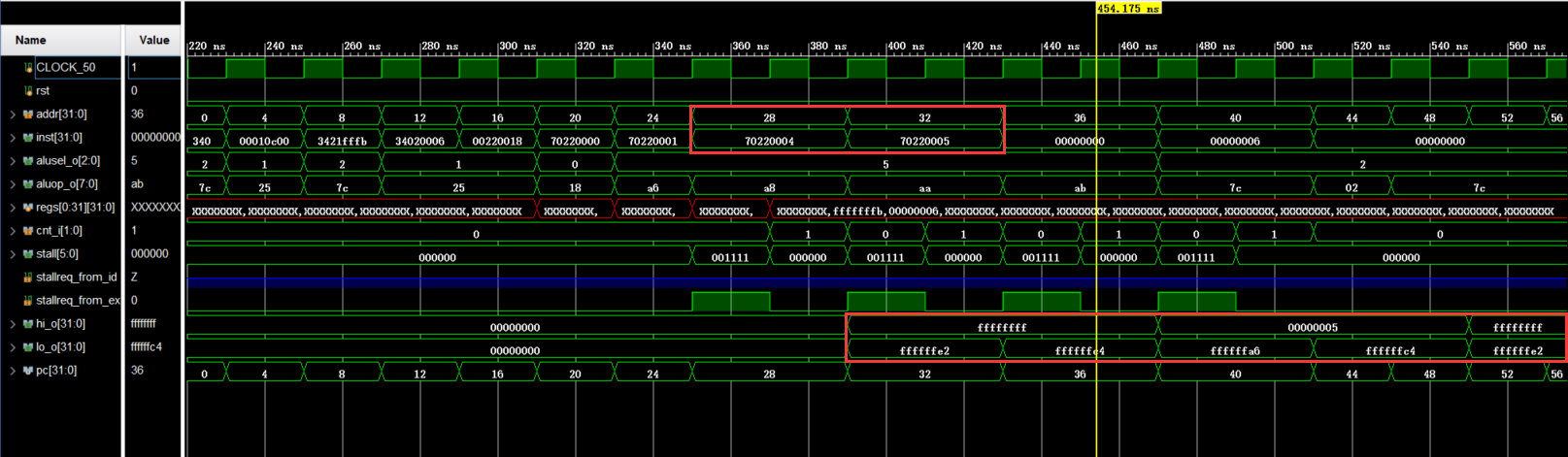
msub $1,$2 # hi = 0x5

# lo = 0xffffffc4

msubu $1,$2 # hi = 0xffffffff

# lo = 0xffffffe2

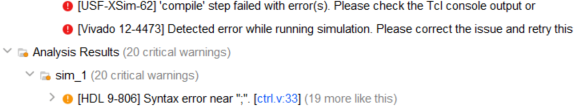
最终仿真结果为：

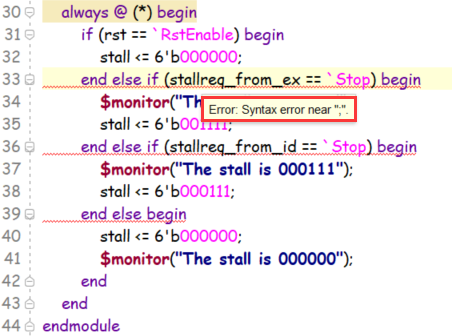


结果正确。

另外，我在给实验班编了几个错误中，包含了一些需要理解源程序以及整个工作原理才能找到的错误（当然一行行对照源码也能改出来）。

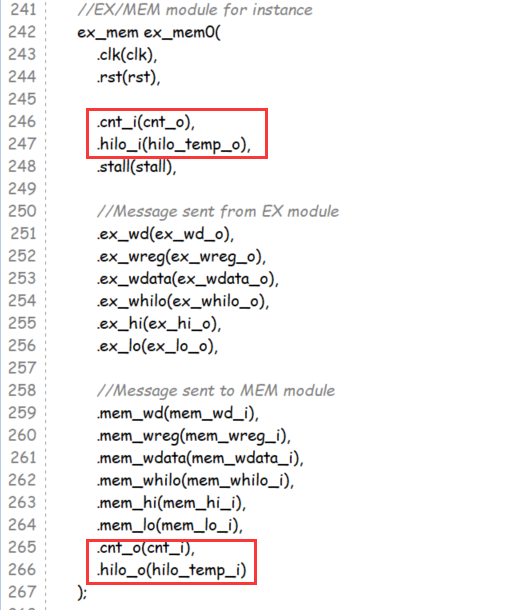
1. 首先是一个语法错误，就是在define宏定义后面加上了分号，这是在预处理的时候就执行的语句，本质上并不是一个语句，所以不能加分号，最终加上分号的结果是编译错误，提示这样的信息：

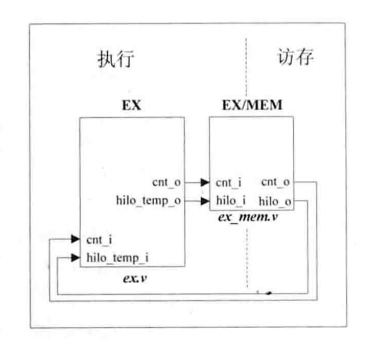


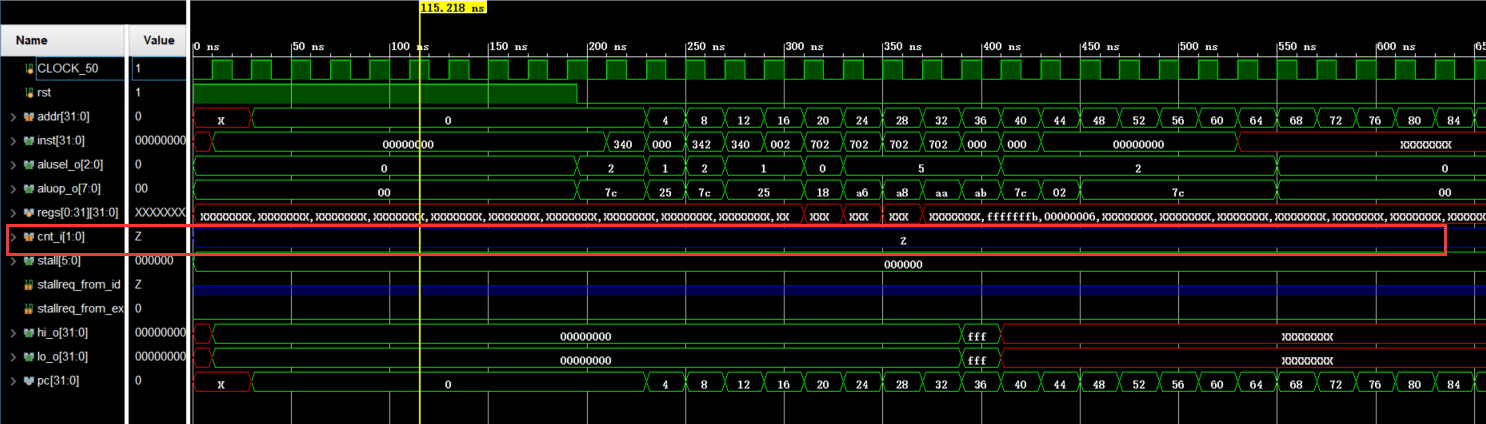


只是提示在分号处有语法错误，这样其实很难发现是宏定义出错~~~

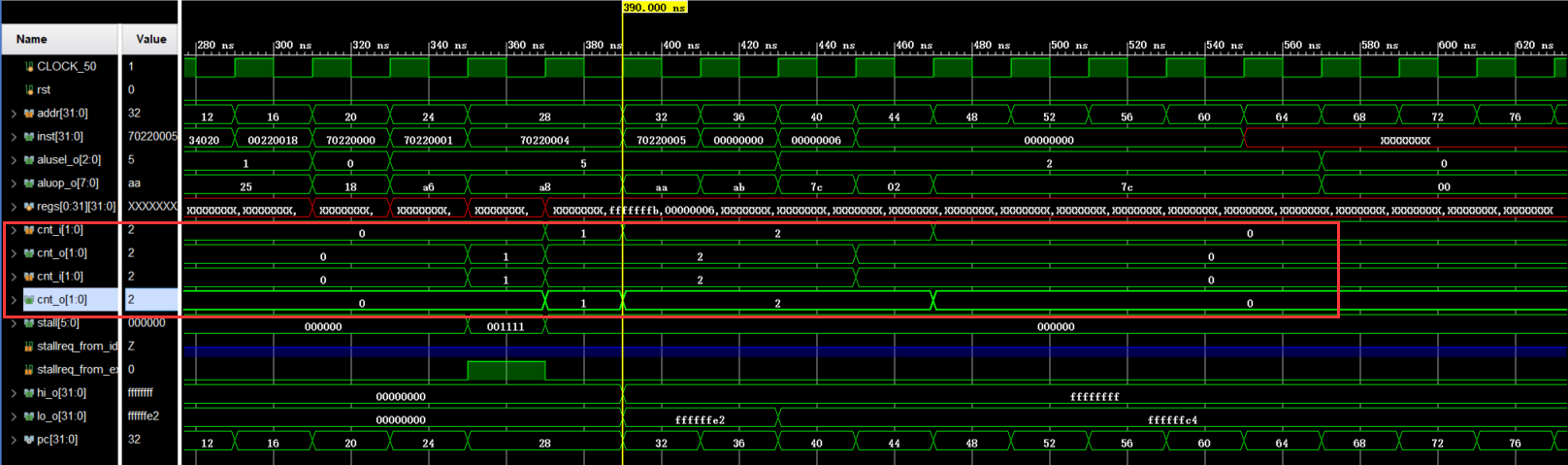
1. 然后就是在OpenMIPS模块中实例化参数错误：



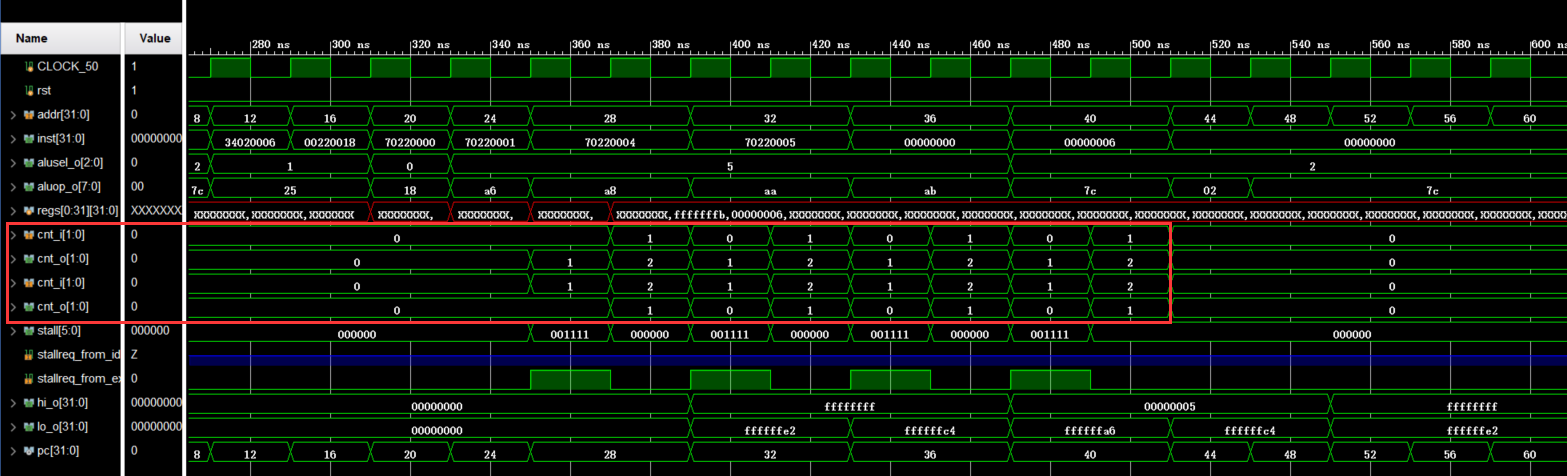
我将这两个参数互换了一下，如果熟悉右边这张系统模块图的话，根据最终仿真结果为高阻态，很容易知道是接口实例化出错。看右边的图我们知道，EX模块的和EX/MEM模块的的值应该是一样的，而变量是EX模块的接口的值，它应该和EX/MEM模块中的值一致。检查接口实例化，应该会发现EX/MEM模块中那两个并没有正确传值；

仿真发现为高阻态：

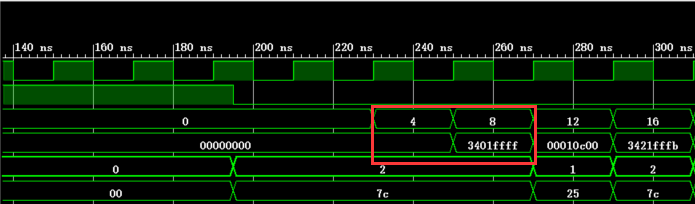
1. 在模块中，我用自行赋值了，最终仿真结果的便为我赋值的值，正确的应为高阻态（整个程序并为对其赋值）
2. 还有就是在EX/MEM模块中一段代码写错，当为0时，cnt\_o应为，我将它改为了，这样EX模块中的的值便一直为，程序便会在一次流水暂停后一直暂停下去，不会跳出暂停；



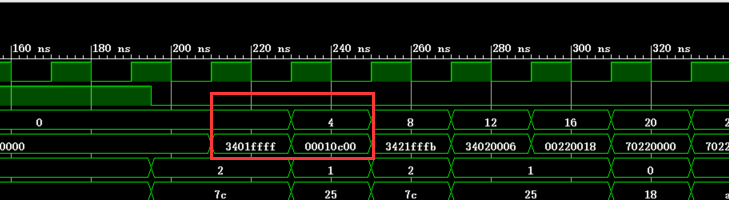
正确的应为：



1. 当然还有一个最难发现的错误，就是汇编代码的一个小问题，我将第一行的改为了，结果虽然正确，但是细心的人会发现，读取指令延后了两个周期：



修改之前正确的应为：



当然还有一些没有意义的改错，譬如故意将之类的指令码写错，少一个之类的，就没有出的必要了。我觉得能从错误中进一步理解代码和流程才是改错的核心所在。

# 附录

不写了，懒得贴代码了。