实验 5 报告

第 28 小组
付琳晴、李舒博

一、实验任务（10%）

第一阶段：

（1）为 myCPU 添加例外支持，即添加例外指令 SYSCALL，该指令使程序跳到 0xbfc00380 地址处的中断处理函数，并且实现的是精确例外，即要清除 SYSCALL 指令之后的指令的执行效果（即写寄存器或写内存指令不写进去）。（2）添加 MTC0、MFC0、ERET 指令，相应的要添加一系列特殊寄存器：STATUS、CAUSE、EPC.

第二阶段：

（1）添加剩下的几种例外和中断支持：整形溢出例外、load 和 store 指令访存地址错误例外、取址指令错误例外、保留指令例外、软硬件中断（包括时钟中断）。（2）添加 COUNT、COMPARE、BADVADDR 三个寄存器，实现 COUNT 寄存器自动两个周期加 1。（3）中断或例外发生时，cp0 寄存器硬件自动置上相应位，并在例外结束后相应位取消。

二、实验设计（30%）

（一）特殊寄存器的读写

第一阶段添加了 STATUS、CAUSE、EPC 三个寄存器，第二阶段添加了 COUNT、COMPARE、BADVADDR 三个寄存器，来满足例外时对协处理器的寄存器们一系列读写操作，从而记录一些例外的关键信息。这些特殊寄存器在指令中出现是对应的 5 位序号，STATUS 对应 12 号，CAUSE 对应 13 号，EPC 对应 14 号。在 CPU 内部需要访问这些特殊寄存器时也采用根据序号访问的方式：当指令是对 cp0 寄存器做的操作（MTC0）且操作号是对应的，可以对该 cp0 寄存器进行写操作。例如 STATUS 寄存器的写操作部分代码如图 1.

```
end
else if (pipe2_data[194] && pipe2_data[199:195] == 5'd12) begin
    cp0_status <= alu_result1;
end
```

图 1. STATUS 寄存器写操作

SYSCALL 信号引起例外时，需要对 STATUS 寄存器 EXL 位（第 1 位）置 1，用来屏蔽所有硬件中断和软件中断。需要根据 SYSCALL 指令是否在延迟槽中对 CAUSE 寄存器 BD 位（32 位）写入 0 或 1，由 SYSCALL 引起的例外还要把 CAUSE 寄存器 ExcCode 位中系统调用例外位置 1. 这些操作都是对寄存器某一位进行写操作，因此特殊寄存器还要满足这些对单独位的写操作，通过非阻塞赋值的方式。

理论上来说，对寄存器的写操作都是在写回级进行的，但是在中断这种情况下，一个中断发生后就要立即屏蔽中断，要立即把 EXL 置 1,，因此在代码中对 cp0 寄存器们的写操作都发生在 exe 级，这一点有别于通用寄存器和 HI、LO 寄存器的写操作（因为 HI 和 LO 寄存器线路之前规划好不敢改了）。因此我们需要把 decode 级判断出来的 SYSCALL 信号和 ERET 信号传到 exe 级，在 exe 级判断指令是不是这两种情况，如果是则进入 SYSCALL 时 EXL 位置 1，退出中断程序时将 EXL 位置 0.

判断 SYSCALL 是否在延迟槽中需要看其前一条指令是否是 Branch 指令。需要将 decode 级译码出的指令是否是分支跳转指令的判断信号传到 exe 级，当 exe 级是一条分支跳转指令并且 decode 级此时是 SYSCALL 指令时，需要对 CAUSE 寄存器 BD 位（32 位）写入 1. 以上两段的操作对应的硬件控制信号线路如图 2.

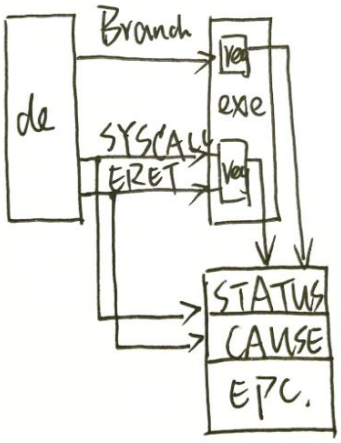

图 2. 特殊寄存器写数据控制信号线路

（二）精确例外

精确例外即要求例外处后面的指令都没有产生执行效果，代码中实现为把例外处后面的指令产生的写信号都置 0. 事实上只会多执行一条延迟槽的指令，只要消除这条指令的执行效果即可。选择在 decode 级发现 exe 级是 SYSCALL 指令时，将 MemWrite 和 RegWrite 置 0. 此处判断需要将 exe 级的信号前递到 decode 级，数据前递一般都会出现初始时是高阻态的情况，利用 valid 信号可以保证在后面流水级里数据还是高阻态的时候，valid = 0，可将数据信号与上 valid 信号作为前面流水级的控制信号。然而事实上实现起来有困难，尤其是在多条 lw 指令导致流水需要阻塞的时候，会导致控制信号一直为 0 又来不及恢复。因此代码中我并没有用 valid 信号与上数据信号作为前递的控制信号，而是采用了第三个实验时数据前递的解决办法，使用一个最初为 0 后来一直为 1 的信号保证前递数据信号在数据还没传到后级流水的时候有确切的值并且之后不干涉数据值。

第一段其实是在写第一阶段的时候对消除指令效果的线路规划，写第二阶段的时候，由于需要把前三级流水中产生的例外和中断全部汇集到一个地方，意味着发生取址错误等例外时，并不是立即就开始执行中断程序的，

所以被中断或例外标记的指令后续的指令变多了，需要消除执行效果的指令也多了。在我的代码中，将所有例外和中断全部汇聚到 exe 级，每一个流水级可能发生的例外或中断以及数据信号传递线路如图 3. 中断例外类型全部汇聚到 ex_valid 信号上，一起报出，进行修改寄存器和跳转到中断处理程序的操作。

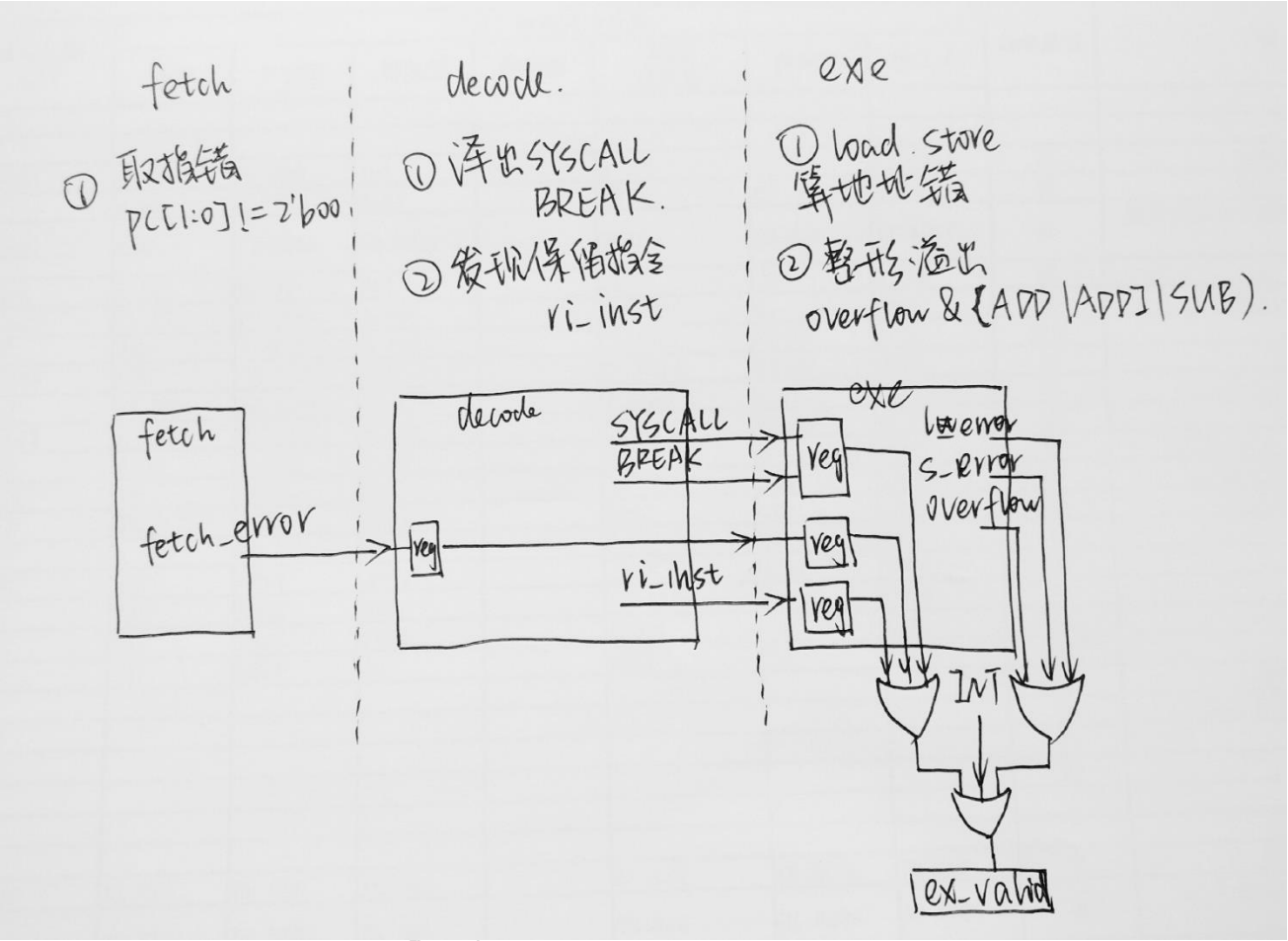

图 3. 中断和例外报出所在流水级和线路规划

将 exe 级的例外中断汇集点 ex_valid 信号连到 nextPC 处，从而影响 nextPC 的取值。按照这样的规划，在跳转到地址 0xbfc00380 之前，例外标记的指令之后，将多执行两条指令，我们需要消除这两条指令的执行效果。消除执行效果主要分为以下几类：（1）写寄存器。（2）sw 指令写内存。（3）跳转。（4）乘除法指令写 HI 和 LO 寄存器。（5）发生数据相关的两条指令产生阻塞。其中，（1）和（2）沿流水线把写使能置 0 即可，（3）的情况分为两种：第一种，当跳转指令（例如 JR）是发生例外的指令（例如 SYSCALL）的下一条时，JR 指令在 decode 级，SYSCALL 在 exe 级时，两者都会对 nextPC 做出影响，产生冲突，这时需要以例外产生的影响为先。第二种情况，当跳转指令（例如 JR）是发生例外的指令（例如 SYSCALL）的后面第两条时，bne 会影响 bfc00384 地址指令的取址。为消除这种情况带来的问题，我们需要消除这条跳转指令的执行效果，用 mem 级被例外标记的指令来消除跳转指令效果，其实就是将 exe 级汇聚点向后传入 mem 级流水级，作为控制信号修正跳转指令的结果，使 PC 正常加 4 而不是加 offset。

不仅如此，我们还需要消除被例外标记的指令的执行效果，例如：产生整形溢出例外的指令，不应该把计算的错误结果写到寄存器中，因此要清除这条指令的执行效果，即不写寄存器。

（三）数据相关

当 MFC0 指令前面有一条 MTC0 指令时会产生数据相关，有可能导致 MTC0 还没有存进去数就被读，因此要考虑从流水线中每级传下去的 alu 结果中得到要写入 cp0 寄存器的值并使用。这个和之前的数据相关类似，不再过多描述。

三、实验过程（60%）

（一）实验流水账

付琳晴：

11.12 上午：读任务书，查看 gettrace 代码相应部分，思考借鉴地方，规划线路。

下午：规划好线路后开始改代码。

晚上：debug，留了一个比较关键的 bug 回宿舍和室友讨论。

11.13 下午：debug 完成第一阶段，上板成功。

11.18 上午：重新规划线路，将各个例外情况与 SYSCALL 融合到一起并依旧拿 func1 测试，修改线路。

下午：de 了一个 load 类型指令的以前残留的没发现的 bug。

11.19 下午：func1 测试的目测可以了，开始拿 func2 测试，狂改 bug。

晚上：狂 debug，一直卡在 n69，被 div 和 mul 折磨一晚上。

11.20 上午：解决 div 和 mul 阻塞带来的各种问题，终于跑过前 80 个用例。

下午：跑过 func2，开始上板。

晚上：解决了 lab4 时电子表遗留的一个问题，继续尝试上板并 debug，上板不行听从助教建议先进行仿真，仿真跑过了之后电子表的上板也正确了。然后写实验报告。

李舒博：

11.12 看实验 5-1pdf，2h 左右，开始糊代码

11.13 接着糊代码，知道队友搞定之后更慢地糊代码

11.14 磨蹭地接着糊代码

11.19 接着糊实验 5-1 的代码，debug，看实验 5-2pdf，2h 左右

11.20 终于把 5-1 的代码 de 过去开始糊 5-2 的代码，糊不下去开始撸实验报告

（二）错误记录

第一阶段：

1、错误 1

（1）错误现象

SYSCALL 指令的下一条指令会被执行并且向寄存器中写值了，导致和 trace 不符。

（2）分析定位过程

考虑到 SYSCALL 事实上是一条类似于 J 的指令，一定会跳转前执行延迟槽里的指令，之前没有考虑消除这条指令的执行效果，发现这个错误之后修改了代码，MemWrite 和 RegWrite 会根据上一条指令是 SYSCALL 而赋值 0.

（3）错误原因

没考虑 SYSCALL 后面一条指令会执行，要清除其执行效果。

（4）修正效果

修改过后与 trace 比对正确。

2、错误 2

（1）错误现象

CAUSE 寄存器读出的值和 ref 比对错误。

（2）分析定位过程

CAUSE 寄存器在 n69 测试里进入 SYSCALL 之前并没有写操作，进入 SYSCALL 里也没有做写操作，说明一定是自己写代码少考虑了情况，又仔细看了讲义发现果然是没有看到约定的硬件自动置 1.

（3）错误原因

讲义里寄存器在例外时硬件要自动置的位没有看仔细，导致代码里并没有将 SYSCALL 指令发生时把 CAUSE 相应位置 1.

（4）修正效果

加了判断，如果 decode 级是 SYSCALL 则置第 6 位和 32 位为 1，与 trace 比对正确。

3、错误 3

（1）错误现象

即使考虑了进入 SYSCALL 时要对硬件做的赋值操作，仍然没有赋值进去，导致 MFC0 取出的 STATUS 寄存器值不对。

（2）分析定位过程

查看 cp0_status 的赋值代码，发现出错的地方在对 EXL 位赋值 1 的时候没有赋值进去，进一步查看 test.s 的代

码发现在进入 SYSCALL 前有一步对 STATUS 赋值的操作，而紧接着又发生了 SYSCALL。而我代码中原本使用的对 STATUS 寄存器赋值的控制信号使用的分别是来自 decode 级的 SYSCALL 和来自 wb 级的 cp0_waddr 信号，会导致一个问题，在一个时钟周期，有可能这两个信号同时有效，即 MTC0 想往 EXL 位写,0，SYSCALL 想往 EXL 写 1，导致 if 分支只能判定一种情况，导致写入数据不对。原本错误代码如图 4. 应该使用同一级的信号作为控制信号，这样每一级每一刻都只会有一条指令，就不会导致写数据冲突。因此修改过后，统一使用 exe 级的控制信号作为写操作控制信号。

```
always @(posedge clk) begin
    if (rst) begin
        cp0_status <= 32'b0001_0000_0100_0000_0000_0000_0000_0000;

    end
    else if (pipe4_data[67] && pipe4_data[72:68] == 5'd12 ) begin
        cp0_status <= pipe4_data[119:88];
    end
    else if (SYSCALL) begin
        cp0_status[1] <= 1'b1;
    end
```

图 4. 错误的赋值代码

（3）错误原因

分别使用了不同流水级的控制信号，导致某周期两个信号同时有效，写入数据冲突，导致某个数据没有写入。

（4）修正效果

将 decode 级译码出的 SYSCALL 指令传到 exe 级，cp0_waddr 也传到 exe 级，统一在 exe 级判断这些信号并赋值，不会产生冲突，与 trace 比对正确。

（5）归纳总结（可选）

写代码时要有全局观，尤其是在流水线中，不同级的流水线执行的不同的指令操作，各级流水线的控制信号放在一起做控制信号会导致线路混乱，寄存器不知道用哪个 if 分支赋值，导致赋值出现差错。

第二阶段：

4、错误 4

（1）错误现象

Lab4 遗留问题！设置时分秒的时候数字蹦的过快，即使加了一堆空循环企图放慢过程，数码管仍然跳的快到看不到。

（2）分析定位过程

重新看了一遍汇编代码，尝试把空循环加在其他地方，并用 lab4 时 gettrace 代码进行仿真，查看波形，发现居然在空循环中跳不出来，仔细看了一眼空循环部分，发现是代码写错，犯了极蠢的错误，错误代码如图 5. 居然把赋值放进了循环内部，导致每次循环都重新开始计数，因此一直在循环里跳不出来。

```
hour_nop_loop:
    LI(a0, 600000)
    addiu a0, a0, -1
    bne zero, a0, hour_nop_loop
    nop
```

图 5. 错误代码

（3）错误原因

Lab4 当时时间紧张，脑袋短路蠢到写了一个死循环。

（4）修正效果

把循环中第一句提到括号外面，能够放慢设置时间向上加的速度。

5、错误 5

（1）错误现象

COUNT 寄存器不能自动加 1

（2）分析定位过程

查看 COUNT 寄存器对应代码，被控制信号 count_add 控制，count_add 信号每个周期取反，COUNT 在 count_add 为 1 时加 1，代码看起来没有问题，查看波形发现 count_add 最开始复位之后变成高阻态，因此每次取反都没有用，导致 COUNT 寄存器不能自动加 1.

（3）错误原因

控制 COUNT 寄存器的控制信号 count_add 由于还受其他信号的控制，初始化后变成高阻态，导致 COUNT 无法在其控制下自动加 1.

（4）修正效果

将 count_add 的控制信号进行修改，保证了该信号在初始时都是有示数的，而非高阻态。COUNT 可以正常加 1.

6、错误 6

（1）错误现象

Load 指令地址计算错误例外判断有问题。

（2）分析定位过程

发现会莫名其妙的认为其他指令是 load 指令，查看 load 指令译码环节，发现并没有问题，进而查看与 load 指令相关代码，发现在 decode 级有一个向后传的信号 load_type 有大问题！在之前 load 相关指令测试时没有发现，可能是侥幸脱逃，错误代码如图 6. 乍一看似乎没有问题，是很正常的赋值方式，但是仔细想想之后意识到当指令不是这几种 Load 指令的时候，load_type 依然会被赋值 3’b000，于是就被后面的流水级认作 load 指令做处理了。发现这个错误之后我很怀疑自己先前测试为什么没有发现，可能是因为当时后面流水级还用到了其他的控制信号做判断，导致侥幸 Pass 吧。

```
assign load_type =  ({3{LW}} & 3'b000)
                  | ({3{LB}} & 3'b001) | ({3{LBU}} & 3'b010)
                  | ({3{LH}} & 3'b011) | ({3{LHU}} & 3'b100)
                  | ({3{LWL}} & 3'b101) | ({3{LWR}} & 3'b110);
```

图 6. 错误的赋值方式

（3）错误原因

赋值信号考虑不完善，写错。

（4）修正效果

能够正确识别 load 指令地址计算错误例外。

（5）归纳总结

发现这个错误之后查看了自己其他类似的赋值方式，幸亏其他的控制信号这样赋值由于 0 代表无效就不会出错。这个问题很重要，幸亏今日发现了，不然以后不留意还是要错。

7、错误 7

（1）错误现象

当 SYSCALL 指令后面是 div 指令时不能正确的进入中断处理程序，流水线被阻塞。

（2）分析定位过程

发现报错是这个问题之后一声叹息，真是怕什么来什么，之前对 div 的阻塞一直不太清楚阻塞的过程，跑过且过。然后开始找错。

出错的原因是没有消除 div 的执行效果，在这里 div 执行效果分为两部分，第一部分是计算得的结果会写入 HI 和 LO 寄存器，第二部分是 div 指令本身在 decode 级译码出来之后会导致流水线阻塞。消除计算结果只需要将写信号与上非 ex_valid 就可以置 0，在这里我是将 div 等在译码级得到的指令沿流水线向后传，作为 HI 和 LO 的写信号。对于 div 会引起流水线阻塞的问题，同理，在 pipe1_readygo 的判断信号中对于 div 的判断与上非 ex_valid，为了防止 div 出现在例外指令的后面第二条指令位置，将 ex_valid 向后传一级流水，可以保证当例外指令后面第二条指令在 decode 级的时候正好可以用传到 mem 级的 ex_valid 消除效果。示意图如图 7.

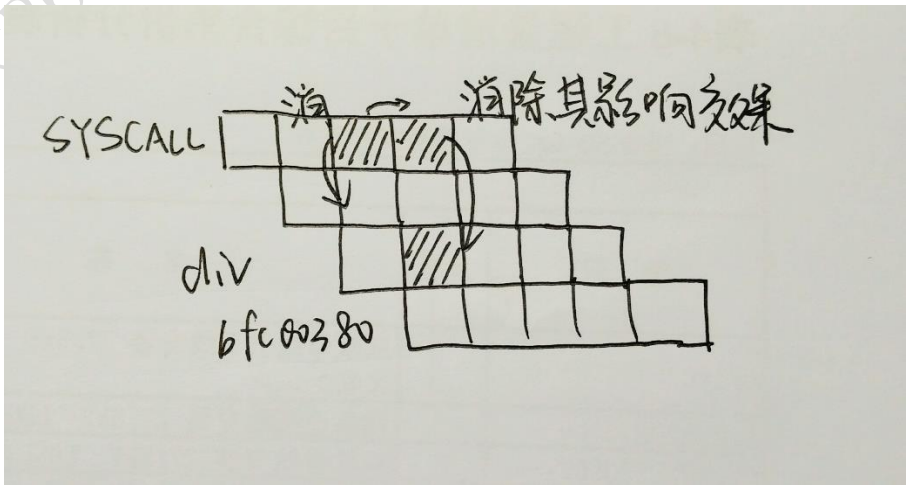

图 7. 用汇聚点消除 div 阻塞影响的解决方法

这个错误解决之后相应的 div、divu、mul、mulu 都可以采用同样的方式解决，一举两得，一劳永逸。

（3）错误原因

没有完全消除被中断标记的指令后面的 div 指令执行效果，使 PC 没有正确跳转到 0xbfc00380.

（4）修正效果

能够先执行中断处理程序，执行结束之后再返回执行 div 指令。

（5）归纳总结

Div 的阻塞问题是门学问…叹气。

8、错误 8

（1）错误现象

当 SYSCALL 指令后第二条指令是跳转指令时，会导致 PC 指向 0xbfc00380 后的下一条指令受跳转指令影响，而不能正常加 4.

（2）分析定位过程

分析了出错原因，示意图如图 8. 这个问题的出现还是因为没有做到精确例外的后果，即没能消除 bne 指令的执行效果。解决方法是修改 PC_next 赋值代码，当出现如下情况时，令 PC_next 等于 PC 加 4，而不受跳转指令的影响。

需要注意的一点是，跳转指令跳转地址的计算分为两种，一种是 bne 类型的 PC + (offset << 2), 一种是 J 类型的 PC 高四位与 Index << 2 的 28 位拼接。我最开始采用消除跳转指令影响的方法是让 offset 为 4，只考虑了第一种跳转指令的情况，在 func2 用例测试时也没有出现问题，因此没发现，结果在电子表上板时有问题，发现没办法中断，无奈重新看仿真波形，才发现是因为自己忽略了处理第二种情况。看波形费了很大的功夫，实在不应该。

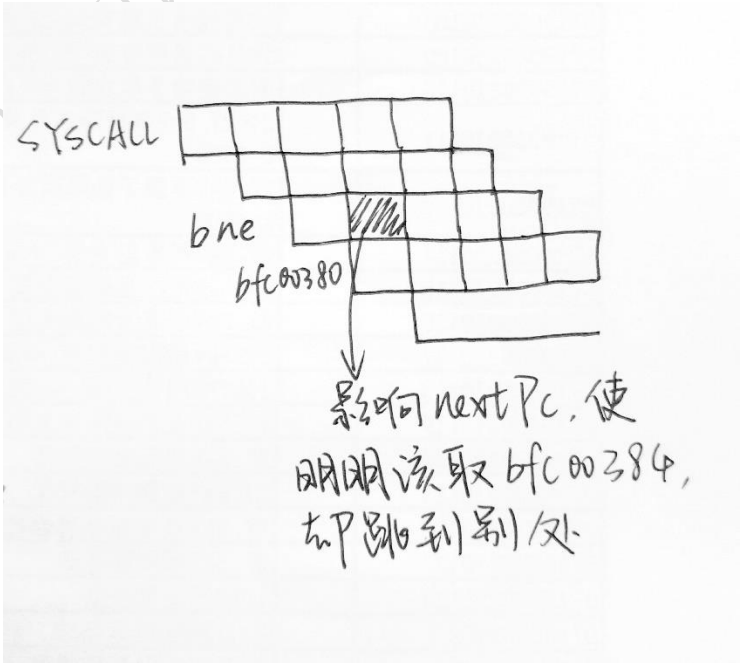

图 8. 取址错误的情况

（3）错误原因

没有完全消除被中断标记的指令后面的 bne 指令执行效果，使 PC 没有正确跳转到 0xbfc00384.

（4）修正效果

能够消除跳转指令带来的影响，PC 在取过 0xbfc00380 后将取 0xbfc00384.

9、错误 9

（1）错误现象

写寄存器指令和 trace 比对正好错开，发生错误。

（2）分析定位过程

查看波形后发现前面发生了整数溢出例外，trace 并没有把整数溢出的结果写到寄存器中，但是我的代码里没有考虑到这一点。我只是把发生例外的指令的后续指令执行结果消掉，因此出错。修改很简单，写寄存器的信号要传到 mem 级流水才会使用，在前面的任意一级流水中将其置 0 即可.

举一反三，相应的 load 和 store 指令取数地址计算错误时，也不应该将数存入错误地址，也应该消除执行效果。

（3）错误原因

指令发生整数溢出例外时，该指令计算的结果不应该写入寄存器。而我的代码没有消除这一计算结果，导致和 trace 比对正好错开，发生错误。

（4）修正效果

与 trace 比对正确。

10、错误 10

（1）错误现象

Mfc0 指令取出 BADVADDR 寄存器值时与 trace 比对错误。报错如图 9.

```
[ 193906 ns] Error!!!
    reference: PC = 0xbfc87ff0, wb_rf_wnum = 0x16, wb_rf_wdata = 0x8001fde1
    mycpu    : PC = 0xbfc87ff0, wb_rf_wnum = 0x16, wb_rf_wdata = 0xbfc87fdc
```

图 9. BADVADDR 寄存器存错数据

（2）分析定位过程

一定是 BADVADDR 赋值代码有问题，赋错了值。仔细理解了讲义后，如果是 fetch 指令时错了地址，要存 fetch 错的地址，load 和 store 指令访存时出错要存的是错误的访存地址。因为自己对讲义没有读全导致的存错数据。

（3）错误原因

BADVADDR 存错数据。

（4）修正效果

与 trace 比对正确。

11、错误 11

（1）错误现象

Mfc0 指令取出 CAUSE 寄存器最高位 BD 位应该是 1，myCPU 显示 0，如图 9。

```
——[ 198748 ns] Number 8'd79 Functional Test Point PASS!!!
——[ 199505 ns] Number 8'd80 Functional Test Point PASS!!!
——[ 201586 ns] Number 8'd81 Functional Test Point PASS!!!
       [ 202000 ns] Test is running, debug_wb_pc = 0xbfc8be88
——[ 202466 ns] Number 8'd82 Functional Test Point PASS!!!
--------------------------------------------------------------
[ 202529 ns] Error!!!
    reference: PC = 0xbfc00400, wb_rf_wnum = 0x1a, wb_rf_wdata = 0x80000020
    mycpu    : PC = 0xbfc00400, wb_rf_wnum = 0x1a, wb_rf_wdata = 0x00000020
--------------------------------------------------------------
```

图 9. CAUSE 寄存器值错误

（2）分析定位过程

查看 BD 位赋值是正确的，而且波形中 BD 位也是正确的，怀疑其他地方出错。因为我的 cp0 寄存器们赋值采用和 gettrace 一样方法，将各个位拼起来形成 32 位 CAUSE 数值，并赋值给一个 wire 类型。去查看了位拼接的部分，数了数位数发现怎么数都是 33 位，很奇怪，明明按照讲义写的，最后发现讲义一处错误如图 10. 被误导了。

表 6-5 Cause 寄存器域描述

域名称	位	功能描述
BD	31	标识最近发生例外的指令是否处于分支延迟槽。1：在延迟槽中；0：不在延迟槽中
TI	30	计时器中断指示。1：有待处理的计时器中断；0：没有计时器中断。
0	30..16	只读恒为 0。
IP7..IP2	15..10	待处理硬件中断标识。每一位对应一个中断线，IP7~IP2 依次对应硬件中断 5~0。 1：该中断线上由待处理的中断；0：该中断线上无中断。
IP1..IP0	9..8	待处理软件中断标识。每一位对应一个软件中断，IP1~IP0 依次对应软件中断 1~0。 软件中断标识位可由软件设置和清除。
0	7	只读恒为 0。
ExcCode	6..2	例外编码。详细描述请见表 6-6。
0	1..0	只读恒为 0。

图 10. 讲义错误

（3）错误原因

位拼接成 33 位，导致最高位被忽略。

（4）修正效果

与 trace 比对正确。

12、错误 12

（1）错误现象

发生地址错例外的 lw 指令后面紧跟 bne 指令，使后面指令执行出现混乱。

（2）分析定位过程

万万没有想到，这种看起来我明明已经考虑了的情况又出现错误，一定是发生了新的没有考虑到的情况。查看了一下指令发现，原来 lw 和 bne 发生了数据相关，就是这么巧！就是这么神奇！由于 lw 带来的数据相关没法用数据前递进行，只能阻塞流水，因此整个流水线被阻塞了，没有按时跳到中断处理程序处。 但事实上这里的阻塞毫无用处，反正 bne 指令也不会真的执行，因此在流水阻塞的控制信号 pipe1_readygo 的 判断数据相关的条件中再次与上非 ex_valid。即这种指令相关直接无视，反正数据也不会用到。

（3）错误原因

lw 和 bne 发生了数据相关，导致地址错例外没有及时跳到中断处理程序。

（4）修正效果

正确跳到 0xbfc00380，与 trace 比对正确。

13、错误 13

（1）错误现象

电子表上板时钟自动加一过一会会卡住。

（2）分析定位过程

上板出错真的令人崩溃！完全不知道哪里错了，汇编代码在 gettrace 上试过是可以完成正常功能的。用自己的 CPU 就死掉了，很崩溃，然后助教建议我用仿真波形看。然后我取消了 gettrace 比对的代码部分，开始看仿真波形，果然 num_data 的数据跑到一定程度就一直保持一个值不变了，极其诡异。考虑到自己汇编代码里有一个等待时间中断打断的死循环，num_data 不变一定是陷入这个死循环了，于是就继续查看波形。发现了死循环里有一步 J 指令，有一次恰好和时间中断一起发生了，结果没有跳到时间中断里面去，而是继续陷入死循环，而 COUNT 和 COMPARE 寄存器之后不会再相等了，因此陷入死循环且数码管不加数。

发现这个之后我开始反思自己为什么没有消除跳转指令的影响，然后意识到如错误 8 提到的，跳转指令地址计算有两种，我最开始只消除了 BNE 之类指令的跳转效果，没有考虑 J 之类的跳转效果，是疏忽了，一声叹息。改过之后仿真正常了。

（3）错误原因

没有消除跳转指令 J 的执行效果，导致时间中断即使跳到 0xbfc00380 又被 J 指令影响跳了回来，开始死循环，数码管不加数。

（4）修正效果

仿真波形可以看到 num_data 固定的正常的加 1.

14、错误 14

（1）错误现象

电子表上板设置时间之后返回，数码管显示开始各种混乱，不能继续各一秒自动加 1，而是数字变化快到肉眼看不见。

（2）分析定位过程

于是又开始看仿真，在 testbanch 里模拟按键。发现设置完时间用 ERET 跳出中断处理程序之后，num_data 开始以极快的频率加 1. 非常奇怪，于是我仔细看了 ERET 的波形，发现当执行完 ERET，STATUS 的 EXL 位被归 0，而由于在设置时间的过程中，其实 COUNT 和 COMPARE 寄存器已经相等了，CAUSE 的 TI 位一直是 1，只是中断被屏蔽，没能发生时间中断。而 EXL 置 0 之后，立刻发生了时间中断，意味着还没能回到原来的程序里面，就又发生了时间中断，于是我的 EPC 里存了一个中断处理程序里面的一条指令的 PC！导致时间中断发生完又回到中断处理程序，就这样一直让 num_data 很快的加 1。

发现是这个错误之后陷入了沉思…在看起来软件和硬件都有错的情况下怎么办。宁可改汇编代码！因此，我在硬件中断处理程序快结束的部分加入了重写 COMPARE 寄存器和清 COUNT 寄存器的操作，即消除了 TI 位，阻止了时间中断立即发生的可能，且等 COMPARE 和 COUNT 相等之后可以发生时间中断。

（3）错误原因

设置完时间没能回到原来的程序里面，就又发生了时间中断，EPC 里存了一个中断处理程序里面的一条指令的 PC，导致时间中断发生完又回到中断处理程序，就这样一直让 num_data 很快的加 1。

（4）修正效果

上板终于终于对了！

四、实验总结

（一）组员：付琳晴

谁说的第一阶段完成了，第二阶段很简单…第一阶段太简单，导致第二阶段的时候重新规划了一遍线路…在写 cp0 寄存器们的时候借鉴了 gettrace 代码，觉得 gettrace 的代码看起来真是很整齐且有逻辑，虽然控制信号太多看不懂，但是对 cp0 赋值的情况大体是统一的，所以就按照 gettrace 改了自己的代码。第二阶段 debug 真的有毒，以前写 mul 和 div 得过且过的代码在第二阶段测试全部死掉，一个一个 de 啊真的是，debug 心得是千万千万不要乱改以前已经跑过前几十个测试案例的代码部分，尤其是 mul 和 div 的线路，改一下之后连之前跑过的都 error 了。不过 func2 测试最终还是一个个 de 出来了，电子表和记忆游戏的上板调试就难过的要死。

在硬件和软件都貌似有问题的时候，竟然不知道该怪哪个，反正都是自己写的，大概是怪自己叭。软件看起来应该是没有问题的，但我还是选择了，改汇编代码，唉，因为 CPU 不敢乱动。

不过很感谢助教提醒我上板测试的两个程序都可以先仿真，我就沉迷于看仿真波形，看自己电子表是怎么乱加数的。最后仿真跑对了再上板，就成功了。

（二）组员：李舒博

一期中考完松懈导致 lab5-1 用队友代码验收 orz。松懈完发现自己欠一篇论文和一篇巨大的 presentation…看完文献看了两遍实验指导就强行代码糊不出来。lab5-1 拖到好晚才糊出来 orz，lab5-2 写到一半开始撸实验报告…

lab4 的时候队友处理了中断例外相关的代码，导致在 lab5-1 的时候写 cp0 寄存器出了很多理解上有问题导致的 bug。这种 bug 巨大地难想怎么 de 了 就 惯性思维怎么都拧不回来。非常惨了。lab5 惨痛 orz。对不起队友。没搞懂原理就糊代码真是和上刑一样……（希望）再也不了。恩。