**MIPS-C 指令集(共计55条)**

**一、指令分类**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **指令类型** | **Instr** | **读寄存器** | **写寄存器** | **use(stage)** | **new(REG)** |
| cal\_R(13) | addu\subu\add\sub\and\nor\or\slt\sltu\sllv\srav\srlv\xor | rs, rt | rd | [rs: E] [rt: E] | [rd: M] |
| cal\_Ist(7) | ori\addi\addiu\andi\slti\sltiu\xori | rs | rt | [rs: E] | [rt: M] |
| cal\_It(1) | lui | --- | rt | --- | [rt: M] |
| j\_addr(1) | j | --- | --- | --- | --- |
| j\_l(1) | jal | --- | 31 | --- | [31: E(PC+8)] |
| j\_r(1) | jr | rs | --- | [rs: D] | --- |
| j\_lr(1) | jalr | rs | rd | [rs: D] | [rd: E(PC+8)] |
| store(3) | sw\sh\sb | rs, rt | --- | [rs: E] [rt: M] | --- |
| load(5) | lw\lh\lhu\lb\lbh | rs | rt | [rs: E] | [rt: W] |
| b\_st(2) | beq\bne | rs, rt | --- | [rs: D] [rt: D] | --- |
| b\_s(4) | blez\bgez\bltz\bgtz | rs | --- | [rs: D] | --- |
| syscall(5) | syscall\break\eret\mtc0\mfc0 | --- | --- | --- | --- |
| shift\_R(3) | sll\sra\srl | rt | rd | [rt: E] | [rd: M] |
| hilo\_cal | mult\multu\div\divu | rs, rt | ---(hilo) | [rs: E] [rt: E] | --- |
| hilo\_mf | mflo\mfhi | ---(hilo) | rd | --- | [rd: M] |
| hilo\_mt | mtlo\mthi | rs | ---(hilo) | [rs: E] | --- |

**二、控制信号赋值**

**A. cal\_R(13) \ shift\_R(3)**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Instr | ALUControl[3:0] | JumpOp[2:0] | DMType[2:0] | BType[2:0] | EXTOp[2:0] | ALUSrc[1:0] | MemtoReg | RegWrite | MemWrite | RegDst |
| addu | `ALU\_add | `JumpOp\_notj | `DMType\_null | `BType\_notb | `EXTOp\_null | `ALUSrc\_rt | 0 | 1 | 0 | rd |
| subu | `ALU\_sub |
| add | `ALU\_add |
| sub | `ALU\_sub |
| and | `ALU\_and |
| nor | `ALU\_nor |
| or | `ALU\_or |
| slt | `ALU\_slt |
| sltu | `ALU\_sltu |
| sllv | `ALU\_sll |
| srlv | `ALU\_srl |
| srav | `ALU\_sra |
| xor | `ALU\_xor |
| sll | `ALU\_sll | `EXTOp\_shamt | `ALUSrc\_Imm |
| sra | `ALU\_sra |
| srl | `ALU\_srl |

\*(slti/sltui均为SignExtended)

\*(更改EXTOp，增加输入端口shamt，添加控制信号`EXTOp\_shamt, [2:0])

\*(更改ALUSrc控制信号，添加`ALUSrc\_shamt, [2:0]) 🡪 错误！！！ 因为shamt扩展已在EXT中完成，ALU\_B端口输入仍为EXT的输出，因此不必增加控制信号，

\*但为防止上机时新指令需要增加扩展信号，提前将信号位数增加为2.

\*(更改ALUOp控制信号，添加`ALU\_shift\_sieries, [3:0])

**B. cal\_Ist(7) \ cal\_It(1)**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Instr | ALUControl[3:0] | JumpOp[2:0] | DMType[2:0] | BType[2:0] | EXTOp[1:0] | ALUSrc[1:0] | MemtoReg | RegWrite | MemWrite | RegDst |
| ori | `ALU\_or | `JumpOp\_notj | `DMType\_null | `BType\_notb | `EXTOp\_zero | `ALUSrc\_Imm | 0 | 1 | 0 | rt |
| addi | `ALU\_add | `EXTOp\_sign |
| addiu | `ALU\_add | `EXTOp\_sign |
| andi | `ALU\_and | `EXTOp\_zero |
| slti | `ALU\_slt | `EXTOp\_sign |
| sltiu | `ALU\_sltu | `EXTOp\_sign |
| xori | `ALU\_xor | `EXTOp\_zero |
| lui | `ALU\_add | `EXTOp\_lui |

**C. j(4) (略)**

**D. store(3) (略) \ load(5) (略)**

**E. b\_st(2) (略) \ b\_s(4) (略)**

**F. hilo(8)**

**G. syscall(5) (略)**

**(32)“ALU.v”: A, B, D**

**(25)“EXT.v”: A\_shift\_3, B, D, E**

**(50) “ALUSrc”(MUX):**

**三、P5\P6新增部件**

**1. \_Decoder(CU + Decoder)**

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 功能描述 |
| [31:0]Instr | I |  |
| [4:0]rs | O | Operand |
| [4:0]rt | O | Operand |
| [4:0]rd | O | Operand |
| [4:0]shamt | O | Operand |
| [15:0]Imm | O | Operand |
| [4:0]HILOType | O | Control\_Unit |
| [3:0]ALUControl | O | Control\_Unit |
| [2:0]JumpOp | O | Control\_Unit |
| [2:0]DMType | O | Control\_Unit |
| [2:0]BType | O | Control\_Unit |
| [2:0]EXTOp | O | Control\_Unit |
| [1:0]ALUSrcA | O | Control\_Unit |
| [1:0]ALUSrcB | O | Control\_Unit |
| [3:0]ALUControl | O | Control\_Unit |
| MemtoReg | O | Control\_Unit |
| RegWrite | O | Control\_Unit |
| MemWrite | O | Control\_Unit |
| jal | O | MUX\_Sel  1: \*\_PC8  0: M\_ALUResult\W\_WD3 |
| [5:0] InstrType | O | Instr\_Type(\_FSU) |
| [4:0] RegDst | O | mips\\_FSU |

**重大失误：**

**Shift\_R系列指令操作数为[rt]而不是[rs]，所以需要大改：**

**1. \_Decoder需要增加信号ALUSrcA\_Sel，并将原信号ALUSrc名称改为ALUSrcB\_Sel**

**2. MIPS.v 全面整改**

**(其实ALU新增端口改起来会更方便些，但是这样不够优雅)**

**<本次整改采用模拟考试的方式，即有条理不慌乱地按次序改正，纸笔记录全部改正过程，每一步操作务必留下证据，动手之前务必想清楚>**

\*

**2. \_FSU (将Decoder在\_FSU中实例化)**

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 功能描述 |
| FSU\_DInstr[31:0] | I | D\_Instr |
| FSU\_Einstr[31:0] | I | E\_Instr |
| FSU\_Minstr[31:0] | I | M\_Instr |
| FSU\_Winstr[31:0] | I | W\_Instr |
| E\_HILObusy | I | start | busy |
| stall | O |  |
| FWMux\_rsE\_Sel | O |  |
| FWMux\_rtE\_Sel | O |  |
| FWMux\_rsD\_Sel | O |  |
| FWMux\_rtD\_Sel | O |  |
| FWMux\_rtM\_Sel | O |  |

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 功能描述 |
| Tuse\_rs[3:0] | O | cal\_R -> 1  cal\_I -> 1  j\_r -> 0  store -> 1  load -> 1  b\_st -> 0  b\_s -> 0  hilo\_cal -> 1  hilo\_mt -> 1  else -> 5(用不到) |
| Tuse\_rt[3:0] | O | cal\_R -> 1  store -> 2  b\_st -> 0  hilo\_cal -> 1  else -> 5(用不到) |
| E\_Tnew[3:0] | O | cal\_R -> 1  cal\_I -> 1  j\_l -> 0  load -> 2  cal\_mf -> 1  else -> 0 |
| M\_Tnew[3:0] | O | cal\_R -> 0  cal\_I -> 0  j\_l -> 0  load -> 1  else -> 0 |

**附：Tuse/E\M\_Tnew赋值**

单周期与流水线在周期内与上升沿工作状态不同：

单周期在第n个上升沿到来之际开始执行第(n+1)条指令；

流水线在第n个上升沿到来之际第n条指令进入D级

**四、相较于P5的重大改动**

\*1. 新增E\_HILO模块

\*\*2. 更改E-M级数据通路

\*a. 因为E级新增HILO部件，M\_REG输入源要发生改变；

\*b. 除了增加多路选择器外，\*名称\*也要发生变化：E\_ALUResult, E\_HILOResult, E\_Result,

\*c. 转发源不要改；

\*3. \_FSU[ -> mips]

\*4. const

\*5. \_Decoder [ -> mips(ok)]

\*6. W\_REG\M\_REG[ -> mips]

/\*\*\*/

观察roife\_HILOUnit暂停转发的行为。

在执行乘除法运算时是一律暂停，还是只有某些特定功能会受影响的指令暂停？

答：只有hilo\_cal\mf\mt暂停，暂停时只是将值写入内置寄存器中，\*\*不存在指令继续向前传递的问题\*\*。

**五、具体部件**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Global | F | D | E | M | W |
| \_Decoder |  | D\_REG | E\_REG | M\_REG | W\_REG |
| \_FSU | F\_IFU | D\_GRF | E\_ALU | M\_DM | \*D\_GRF\* |
|  |  | D\_EXT | E\_HILO |  |  |
|  |  | D\_NPC |  |  |  |
|  |  | D\_Branchif |  |  |  |

**六、思考题：**

1. 为什么需要有单独的乘除法部件而不是整合进 ALU？为何需要有独立的 HI、LO 寄存器？

因为乘除法运算耗时较长，如果单独隔离出来，在HILO处于busy状态时，其他仅仅使用ALU其他运算功能的指令仍可正常运行，可以起到节省时间的作用。如果将HI\LO合并到GRF中，由于乘除法执行存在延时，在乘除法指令开始执行到产生结果这段时间内所有需要读取寄存器的指令都可能会出错。为解决这一问题，肯定要涉及出两个单独存储乘除法运算结果的寄存器；与其将两个寄存器放在W\D级的GRF中，倒不如直接归入HILO部件，计算出结果后直接存储。

2. 参照你对延迟槽的理解，试解释 “乘除槽”。

在乘(除)法运算指令后分别加入5(10)条无关指令，保证CPU正常流水，使效率最大化。

3. 举例说明并分析何时按字节访问内存相对于按字访问内存性能上更有优势。（Hint： 考虑 C 语言中字符串的情况）

因为一个字符占内存大小为1个字节，对字符个数不为4的整数倍字符串or单个字符进行存取操作时，按字节访问更为方便。

4. 在本实验中你遇到了哪些不同指令类型组合产生的冲突？你又是如何解决的？相应的测试样例是什么样的？

见后方测试板块。主要问题在于转发处理。

**七、测试**

***1. store***

li  $t0, 0x89abcdef

sw  $t0, 0($0)  # 0(0)

sb  $t0, 5($0)  # 1(4)

sh  $t0, 10($0) # 2(8)

sb  $t0, 15($0) # 3(12)

li  $t0, 0x89abcdef

sw  $t0, 16($0) # 0(16)

sb  $t0, 21($0) # 1(20)

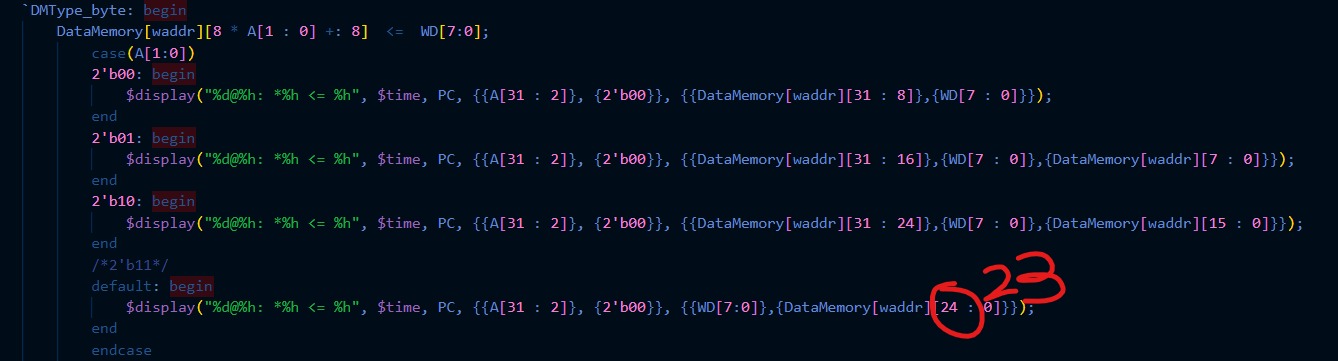
sh  $t0, 26($0) # 2(24)

sb  $t0, 31($0) # 3(28)

li  $t1, 0x55555555

sw  $t1, 24($0) # 0(24)

sh  $t0, 26($0) # 2(24)

BUG\_1\_1：

***2.load***

li  $t0, 0x89abcdef

sw  $t0, 0($0)  # 0(0)

sb  $t0, 5($0)  # 1(4)

sh  $t0, 10($0) # 2(8)

sb  $t0, 15($0) # 3(12)

li  $t0, 0x89abcdef

sw  $t0, 16($0) # 0(16)

sb  $t0, 21($0) # 1(20)

sh  $t0, 26($0) # 2(24)

sb  $t0, 31($0) # 3(28)

li  $t1, 0x55555555

sw  $t1, 24($0) # 0(24)

sh  $t0, 26($0) # 2(24)

lw  $t0, 0($0)

addu    $t0, $t0, $0    #stall (load, cal\_R)

lhu $t2, 10($0) # 0x0000cdef

lh  $t3, 10($0) # 0xffffcdef

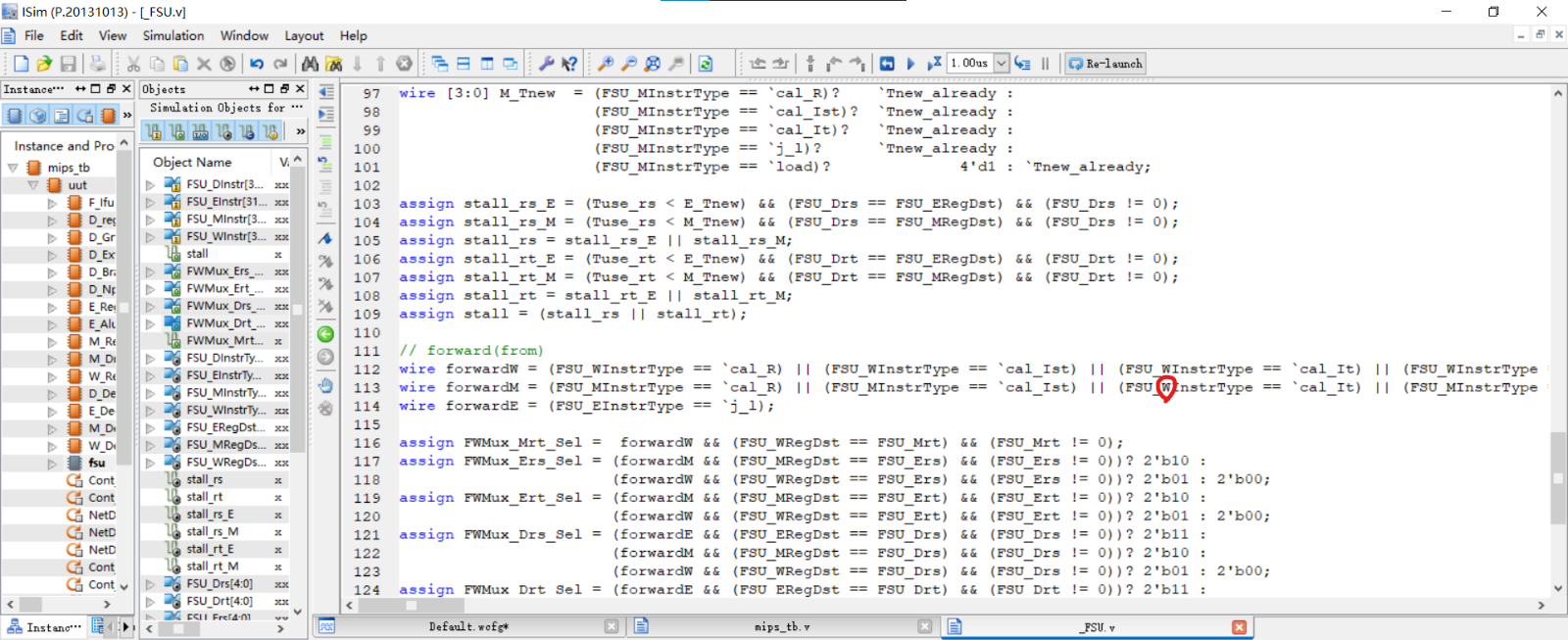
lb  $t4, 31($0) # 0xffffffef

lbu $t5, 31($0) # 0x000000ef

Result:

215@00003040: $ 8 <= 89abcdef

235@00003044: $ 8 <= 89abcdef

BUG\_2\_1:

***3. bne\_jal\_j/ 4. lw***

BUG\_3\_0:

155@0000302c: \*0000001c <= ef000000

175@00003030: $ 1 <= 55550000

Q: Why stall?

A: 并没有暂停。

0x0000302c对应指令在M级完成并输出（store）

0x00003030对应指令在W级完成并输出（cal\_It）

BUG\_4\_1:

输出结果：

 55@00003000: $31 <= 00003008

 65@00003004: $ 1 <= 89ab0000

 75@0000301c: $ 1 <= 89ab0000

 85@00003020: $ 8 <= 89abcdef

 85@00003024: \*00000010 <= 89abcdef

 95@00003028: \*00000014 <= 0000ef00

105@0000302c: \*00000018 <= cdef0000

115@00003030: \*0000001c <= ef000000

145@00003038: $ 1 <= 55550000

155@00003048: $ 8 <= 00000000

175@0000304c: $ 8 <= 00000000

185@00003050: $10 <= 00000000

195@00003054: $11 <= 00000000

205@00003058: $12 <= 00000000

215@0000305c: $13 <= 00000000

ISim>

期待结果：

 55@00003000: $31 <= 00003008

 65@00003004: $ 1 <= 89ab0000

 75@0000301c: $ 1 <= 89ab0000

 85@00003020: $ 8 <= 89abcdef

 85@00003024: \*00000010 <= 89abcdef

 95@00003028: \*00000014 <= 0000ef00

105@0000302c: \*00000018 <= cdef0000

115@00003030: \*0000001c <= ef000000

155@0000304c: $ 8 <= 00000000

175@00003050: $ 8 <= 00000000

185@00003054: $10 <= 00000000

195@00003058: $11 <= 00000000

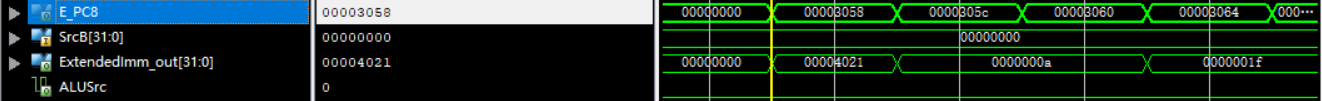
205@0000305c: $12 <= ffffffef

215@00003060: $13 <= 000000ef

ISim>

暂停功能正常，但最后两条load指令输出有误。

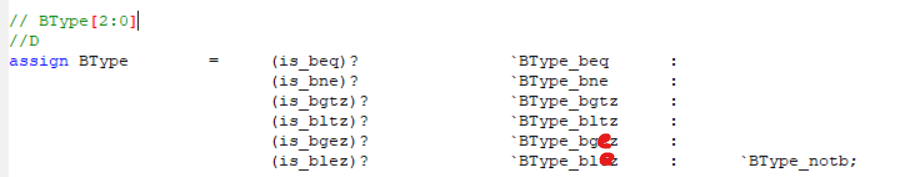
问题定位：E级Imm\_or\_rt 选择信号ALUSrc 出了问题。

最后发现，在添加lh/lb/lhu/lbu等一系列指令时没有对ALUSrc赋值。。。

Jal指令Jump&Link功能均无误。

Bne指令顺序跳转无误。

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 跳转方向 | Beq | Bne | Bgtz | Bltz | Bgez | Blez |
| 顺序 | 1 | 1 | 1 | 1 | 0 | 0 |
| 逆序 | 1 | 1 | 1 | 1 | 0 | 0 |

***5. bgez/blez***

”bgez/blez $0, label”无法正常跳转

Assign BType时出现小笔误。。。

***6. Jalr***

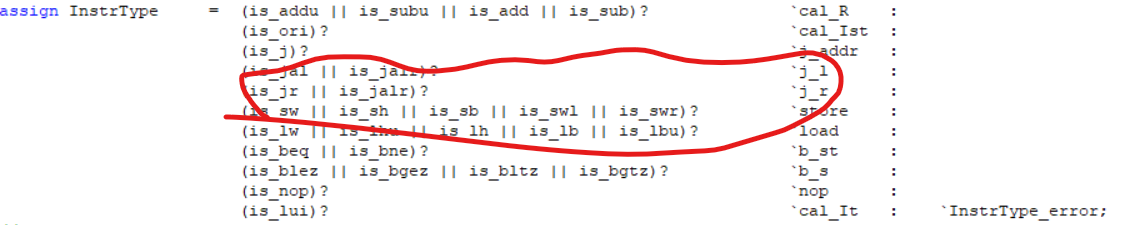
将信号内聚时应注意信号的赋值独立性。

如采用多个单位信号耦合的赋值方式，则可以采用信号复用的方式，即同一指令可以符合set多个信号的特征，

e.g. jalr可以将j\_r 与 j\_l 两个信号同时置1

然而若采用将全部信息内聚在单个多位信号的赋值方式的话，则需要将信号拆开，保证一条指令具有的全部特征仅对应其中一个信号。

e.g. `j\_l `j\_r `j\_lr 三状态需分开设立。



***7. HILO*系列指令**

**在执行0x00003014指令时出现问题**

**0x00003004: lui $1, 8 # $1 <= 0x00080000**

**0x00003008: xor # $3 <= 0**

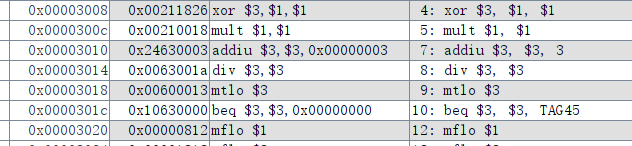
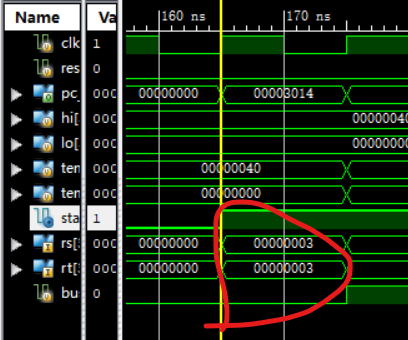
**0x0000300c: mult # hi <= 0x00000040; lo <= 0x00000000**

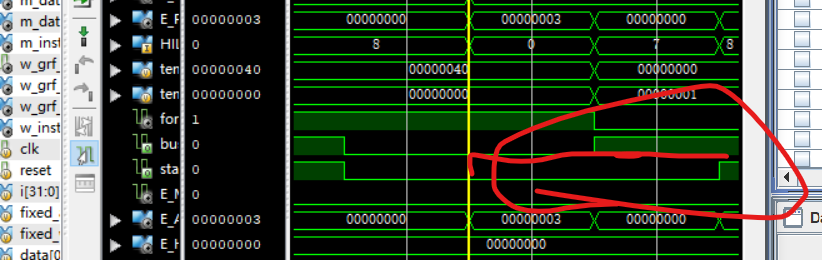
**0x00003010: addiu # $3 <= 3**

**0x00003014: div # 行为在这里出现了错误**

**首先，通过对冲突的分析发现div指令进入E级寄存器时，下一条指令mtlo应该进入D级寄存器，此时的冲突无法通过转发解决，因此下一周期上升沿应暂停，这就意味着此时stall信号应该已经通过组合逻辑被置为1。然而，实际上此时stall信号为0，成功找到问题所在，暂停并未顺利进行。与此同时，调出stall信号赋值逻辑来源之一busy的值，发现busy对应值正确。这时，通过上下两个波形图的对比，我们突然发现，正确的CPU中stall较busy信号早产生一个周期，突然意识到在HILO内部信号start值为1的周期，即busy信号产生的前一周期，HILO就已经投入使用，成功找到这只小臭虫！！！**

assign HILObusy = start | busy;





***8. Forward***

**刚刚搭建好CPU模块时，进行基本转发功能测试时，**

**addiu $1, $0, 5**

**mult $1, $1**

**mfhi $1**

**add $2, $1, $0**

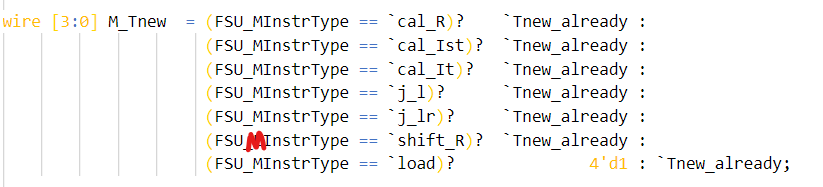
**以上测试程序导入CPU后测试结果有误。**

**原因是忘记进行乘除指令的转发调整，mfxx系列指令有对寄存器的写入操作，必要时需要进行转发。**

wire forwardW = (FSU\_WInstrType == `load)    || (FSU\_WInstrType == `shift\_R) || (FSU\_WInstrType == `cal\_R)   || (FSU\_WInstrType == `cal\_Ist) || (FSU\_WInstrType == `cal\_It) || (FSU\_WInstrType == `j\_l)  || (FSU\_WInstrType == `j\_lr) || (FSU\_WInstrType == `hilo\_mf);

wire forwardM = (FSU\_MInstrType == `shift\_R) || (FSU\_MInstrType == `cal\_R)   || (FSU\_MInstrType == `cal\_Ist) || (FSU\_MInstrType == `cal\_It)  || (FSU\_MInstrType == `j\_l)    || (FSU\_MInstrType == `j\_lr) || (FSU\_MInstrType == `hilo\_mf);

wire forwardE = (FSU\_EInstrType == `j\_l)     || (FSU\_EInstrType == `j\_lr);

***9. stall***

**这是迄今为止我遇到的最难找的臭虫之一，一条笔误导致M级指令为`load，E级指令恰好为nop时发生乱暂停的现象。**

**错误行为很明显，定位十分困难。**