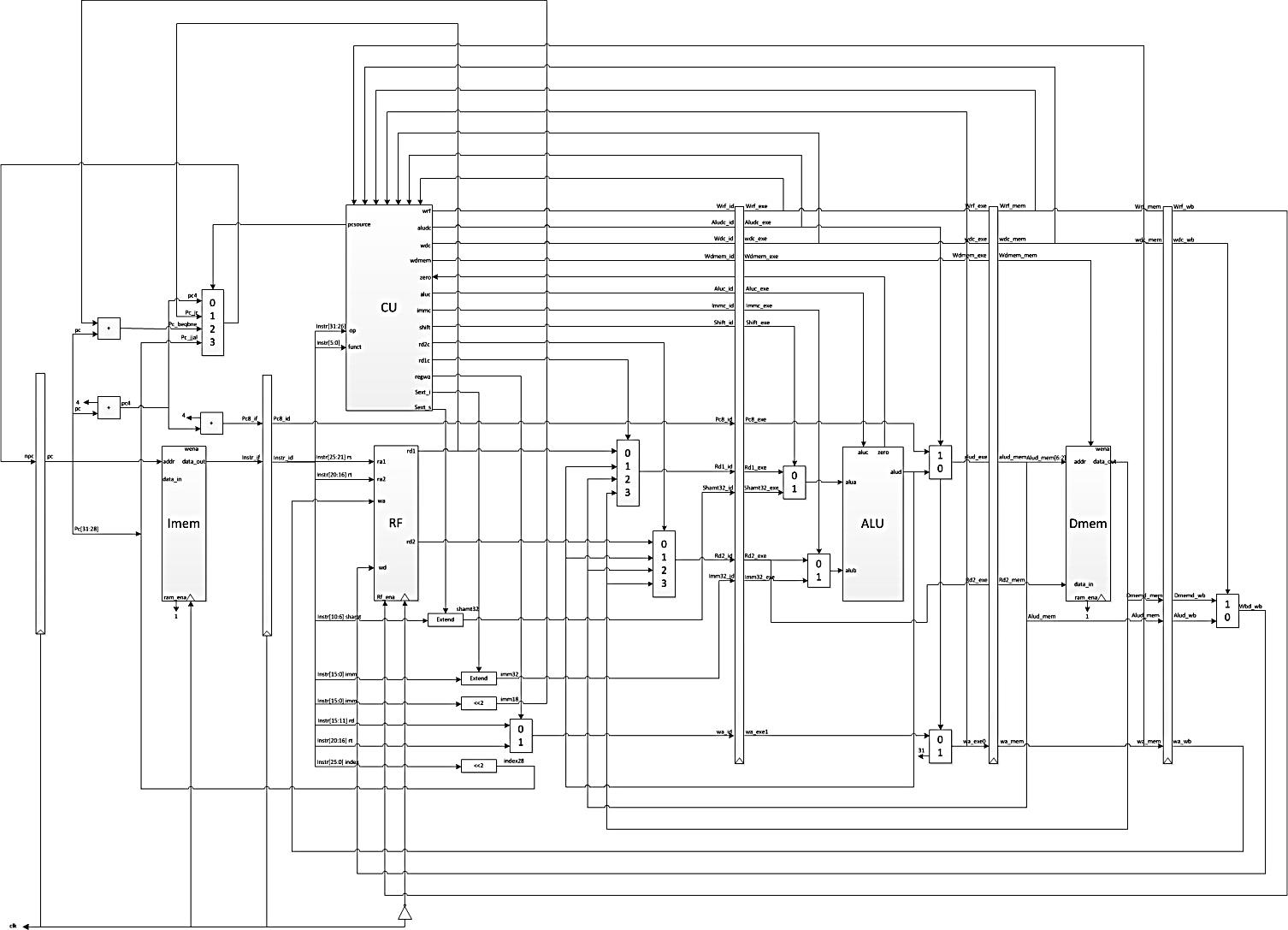
No Data Hazard CPU

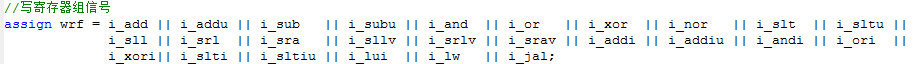
# 概念

数据相关性：如果第一条指令写入的目的寄存器和后续指令的源寄存器的地址是相同的，在后续指令对这个寄存器进行取值的过程中，第一条指令还来不及修改目的寄存器的值，就会造成读错误，这就是数据相关性问题

# 设计步骤

设计流程

数据相关性造成的原因是由于前几条指令需要写寄存器（如下表红色所示），按照这个原因结合wrf信号（如下图所示）主要分成三大类，第一类为最终写入到寄存器的是经过alu运算得到的结果，第二类为最终写入到寄存器的是从dmem中取出的值，第三类为最终写入到寄存器的是非alu非dmem中的值。下面分别对这三类数据相关性进行分析。



|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Mnemonic Symbol** | **Format** | | | | | | **Sample** |
| **Bit #** | **31..26** | **25..21** | **20..16** | **15..11** | **10..6** | **5..0** |  |
| **R-type** | **op** | **rs** | **rt** | **rd** | **shamt** | **func** |  |
| **add** | **000000** | **rs** | **rt** | **rd** | **0** | **100000** | **add $1,$2,$3** |
| **addu** | **000000** | **rs** | **rt** | **rd** | **0** | **100001** | **addu $1,$2,$3** |
| **sub** | **000000** | **rs** | **rt** | **rd** | **0** | **100010** | **sub $1,$2,$3** |
| **subu** | **000000** | **rs** | **rt** | **rd** | **0** | **100011** | **subu $1,$2,$3** |
| **and** | **000000** | **rs** | **rt** | **rd** | **0** | **100100** | **and $1,$2,$3** |
| **or** | **000000** | **rs** | **rt** | **rd** | **0** | **100101** | **or $1,$2,$3** |
| **xor** | **000000** | **rs** | **rt** | **rd** | **0** | **100110** | **xor $1,$2,$3** |
| **nor** | **000000** | **rs** | **rt** | **rd** | **0** | **100111** | **nor $1,$2,$3** |
| **slt** | **000000** | **rs** | **rt** | **rd** | **0** | **101010** | **slt $1,$2,$3** |
| **sltu** | **000000** | **rs** | **rt** | **rd** | **0** | **101011** | **sltu $1,$2,$3** |
| **sll** | **000000** | **0** | **rt** | **rd** | **shamt** | **000000** | **sll $1,$2,10** |
| **srl** | **000000** | **0** | **rt** | **rd** | **shamt** | **000010** | **srl $1,$2,10** |
| **sra** | **000000** | **0** | **rt** | **rd** | **shamt** | **000011** | **sra $1,$2,10** |
| **sllv** | **000000** | **rs** | **rt** | **rd** | **0** | **000100** | **sllv $1,$2,$3** |
| **srlv** | **000000** | **rs** | **rt** | **rd** | **0** | **000110** | **srlv $1,$2,$3** |
| **srav** | **000000** | **rs** | **rt** | **rd** | **0** | **000111** | **srav $1,$2,$3** |
| **jr** | **000000** | **rs** | **0** | **0** | **0** | **001000** | **jr $31** |
|  | | | | | | | |
| **Bit #** | **31..26** | **25..21** | **20..16** | **15..0** | | |  |
| **I-type** | **op** | **rs** | **rt** | **immediate** | | |  |
| **addi** | **001000** | **rs** | **rt** | **immediate** | | | **addi $1,$2,100** |
| **addiu** | **001001** | **rs** | **rt** | **immediate** | | | **addiu $1,$2,100** |
| **andi** | **001100** | **rs** | **rt** | **immediate** | | | **andi $1,$2,10** |
| **ori** | **001101** | **rs** | **rt** | **immediate** | | | **andi $1,$2,10** |
| **xori** | **001110** | **rs** | **rt** | **immediate** | | | **andi $1,$2,10** |
| **lw** | **100011** | **rs** | **rt** | **immediate** | | | **lw $1,10($2)** |
| **sw** | **101011** | **rs** | **rt** | **immediate** | | | **sw $1,10($2)** |
| **beq** | **000100** | **rs** | **rt** | **immediate** | | | **beq $1,$2,10** |
| **bne** | **000101** | **rs** | **rt** | **immediate** | | | **bne $1,$2,10** |
| **slti** | **001010** | **rs** | **rt** | **immediate** | | | **slti $1,$2,10** |
| **sltiu** | **001011** | **rs** | **rt** | **immediate** | | | **sltiu $1,$2,10** |
| **lui** | **001111** | **00000** | **rt** | **immediate** | | | **Lui $1, 10** |
|  | | | | | | | |
| **Bit #** | **31..26** | **25..0** | | | | |  |
| **J-type** | **op** | **Index** | | | | |  |
| **j** | **000010** | **address** | | | | | **j 10000** |
| **jal** | **000011** | **address** | | | | | **jal 10000** |

1. Alu结果回写寄存器

如下图所示，将alu的结果写入到寄存器和其指令顺序存在一定关系，由于是在WB级的下降沿将数据写入到寄存器，所以存在如下三种情况：

1. 第二条指令的源寄存器如果用到第一条指令的目的寄存器中的值（如图中黑线1所示），则会造成数据的读错误，那么对于这种情况下的数据相关性的解决方案是在第一条指令的EXE级，即第二条指令的ID级，将要到WB级才能读到的数据提前到第一条指令的EXE级（如图中红线1所示）。
2. 第三条指令的源寄存器如果用到第一条指令的目的寄存器中的值（如图中黑线2所示），则会造成数据的读错误，那么对于这种情况下的数据相关性的解决方案是在第一条指令的MEM级，即第三条指令的ID级，将要到WB级才能读到的数据提前到第一条指令的MEM级（如图中红线2所示）。
3. 第四条指令的源寄存器如果用到第一条指令的目的寄存器中的值（如图中黑线3所示），由于第一条指令是在WB级的clk下降沿将数据写入到寄存器组，而第四条指令的读源寄存器是网线类型，只要在这个时钟周期内有正确的数据来修改寄存器组，那么最终译码得到的源操作数的值就是正确的值。



1. Dmem结果回写寄存器

如下图所示，将dmem的结果写入到寄存器和相关性指令的位置存在一定关系，由于是在WB级的下降沿将数据写入到寄存器，所以存在如下两种情况：

1. 第一条读存储器指令的目的寄存器地址和第三条的指令的读源寄存器地址相同（如图中黑线所示），则会造成数据的读错误，那么对于这种情况下的数据相关性的解决方案是在第一条指令的MEM级，即第三条指令的ID级，将要到WB级才能读到的数据提前到第一条指令的MEM级（如图中红线所示）。



1. 第一条读存储器指令的目的寄存器地址和第二条的指令的读源寄存器地址相同（如图中黑线1所示），则会造成数据的读错误，那么对于这种情况下的数据相关性的解决方案是在对流水线进行一个时钟周期的暂停，即第二条指令执行两遍，但是为了让第一次执行的时候不对结果产生影响，需要对第一次执行进行废弃处理（如图灰色斜线部分），具体的处理方法如下代码所示。主要是通过禁止写寄存器和写存储器来废弃第一次执行时的影响。

经过了一个废弃指令的填充，第一条读存储器指令的目的寄存器地址和第二条的指令的读源寄存器地址相同（如图中黑线2所示）而引起的读错误也得到了解决。



1. 非alu结果非dmem值回写寄存器

这种指令为jal指令，将pc+8的结果写回到寄存器。对于第二条指令，其处理过程如下图所示，由于MIPS对jal指令的特性，存在一个时钟周期的气泡（见No Control Hazard CPU）的填充（如下图灰色斜杠所示），并且jal所写的31#寄存器是系统级寄存器，只允许存放中断返回地址，所以除了jal和jr指令之外不允许访问31#寄存器，所以第三天指令（即中断返回地址所指向的指令）是一条和jal无关的的指令，那么作为中断入口地址的指令最早出现在如下图所示的第四条指令处，如果这条中断是一个没有意义的简单返回，那么最快处理中断返回的指令就是第四条指令。综上所述，最快要在第四条指令处才会读31#寄存器，由于第一条指令在WB的下降沿已经把正确的值写入到31#寄存器，所以第四条指令不会读到一个错误的值，即不需要对第一条指令是jal的指令进行数据相关性控制。



|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 数据相关第一条指令真值表 | | | | | | | | | |
| 指令 | 控制信号 | | | | | | | | |
|  | Sext\_s | Sex\_i | shift | regwa | immc | wena | wdc | aludc | Wrf |
| 第一条为alu结果写寄存器指令 | | | | | | | | | |
| Add | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| Addu | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| Sub | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| Subu | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| And | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| or | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| Xor | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| Nor | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| Slt | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| Sltu | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| Sll | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| Srl | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| Sra | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| Sllv | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| Srav | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| Addi | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 1 |
| Addiu | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 1 |
| Andi | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 |
| Ori | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 |
| Xori | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 |
| Slti | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 1 |
| Sltiu | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 1 |
| Lui | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 |
| 第一条指令为dmem结果写寄存器指令 | | | | | | | | | |
| Lw | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 |
| 第一条指令为非alu非dmem结果写寄存器指令 | | | | | | | | | |
| Jal | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |

后续相关指令（读寄存器指令）

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Mnemonic Symbol** | **Format** | | | | | | **Sample** |
| **Bit #** | **31..26** | **25..21** | **20..16** | **15..11** | **10..6** | **5..0** |  |
| **R-type** | **op** | **rs** | **rt** | **rd** | **shamt** | **func** |  |
| **add** | **000000** | **rs** | **rt** | **rd** | **0** | **100000** | **add $1,$2,$3** |
| **addu** | **000000** | **rs** | **rt** | **rd** | **0** | **100001** | **addu $1,$2,$3** |
| **sub** | **000000** | **rs** | **rt** | **rd** | **0** | **100010** | **sub $1,$2,$3** |
| **subu** | **000000** | **rs** | **rt** | **rd** | **0** | **100011** | **subu $1,$2,$3** |
| **and** | **000000** | **rs** | **rt** | **rd** | **0** | **100100** | **and $1,$2,$3** |
| **or** | **000000** | **rs** | **rt** | **rd** | **0** | **100101** | **or $1,$2,$3** |
| **xor** | **000000** | **rs** | **rt** | **rd** | **0** | **100110** | **xor $1,$2,$3** |
| **nor** | **000000** | **rs** | **rt** | **rd** | **0** | **100111** | **nor $1,$2,$3** |
| **slt** | **000000** | **rs** | **rt** | **rd** | **0** | **101010** | **slt $1,$2,$3** |
| **sltu** | **000000** | **rs** | **rt** | **rd** | **0** | **101011** | **sltu $1,$2,$3** |
| **sll** | **000000** | **0** | **rt** | **rd** | **shamt** | **000000** | **sll $1,$2,10** |
| **srl** | **000000** | **0** | **rt** | **rd** | **shamt** | **000010** | **srl $1,$2,10** |
| **sra** | **000000** | **0** | **rt** | **rd** | **shamt** | **000011** | **sra $1,$2,10** |
| **sllv** | **000000** | **rs** | **rt** | **rd** | **0** | **000100** | **sllv $1,$2,$3** |
| **srlv** | **000000** | **rs** | **rt** | **rd** | **0** | **000110** | **srlv $1,$2,$3** |
| **srav** | **000000** | **rs** | **rt** | **rd** | **0** | **000111** | **srav $1,$2,$3** |
| **jr** | **000000** | **rs** | **0** | **0** | **0** | **001000** | **jr $31** |
|  | | | | | | | |
| **Bit #** | **31..26** | **25..21** | **20..16** | **15..0** | | |  |
| **I-type** | **op** | **rs** | **rt** | **immediate** | | |  |
| **addi** | **001000** | **rs** | **rt** | **immediate** | | | **addi $1,$2,100** |
| **addiu** | **001001** | **rs** | **rt** | **immediate** | | | **addiu $1,$2,100** |
| **andi** | **001100** | **rs** | **rt** | **immediate** | | | **andi $1,$2,10** |
| **ori** | **001101** | **rs** | **rt** | **immediate** | | | **andi $1,$2,10** |
| **xori** | **001110** | **rs** | **rt** | **immediate** | | | **andi $1,$2,10** |
| **lw** | **100011** | **rs** | **rt** | **immediate** | | | **lw $1,10($2)** |
| **sw** | **101011** | **rs** | **rt** | **immediate** | | | **sw $1,10($2)** |
| **beq** | **000100** | **rs** | **rt** | **immediate** | | | **beq $1,$2,10** |
| **bne** | **000101** | **rs** | **rt** | **immediate** | | | **bne $1,$2,10** |
| **slti** | **001010** | **rs** | **rt** | **immediate** | | | **slti $1,$2,10** |
| **sltiu** | **001011** | **rs** | **rt** | **immediate** | | | **sltiu $1,$2,10** |
| **lui** | **001111** | **00000** | **rt** | **immediate** | | | **Lui $1, 10** |

如表所示，对于后续读寄存器指令的rs和rt主要有两种情况，一种是rs/rt为0，另一种rs/rt为非零，对于为0的情况需要屏蔽。

# 实验验证

1．Alu结果回写寄存器

如图1所示，对于具有数据相关性的汇编指令，如下代码所示，在准pipeline CPU中由于数据相关性，会造成最终写入寄存器一个错误的值，如下图1的3#和4#寄存器所示，理论上应该写入的值是1，而实际上由于数据相关性问题，写入的值是0。当处理了数据相关性之后，跑想通的汇编指令，得到的结果如下图2所示，3#和4#寄存器的值和预期的一致。

add $2, $1, $0

add $3, $2, $0

add $4, $2, $0

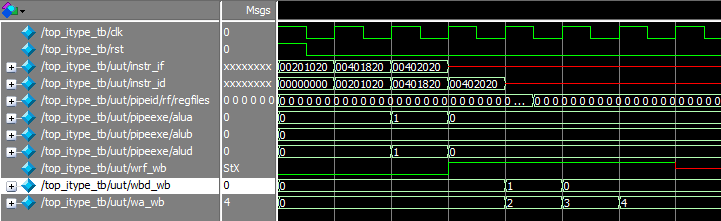


图1 alu结果没有处理数据相关性

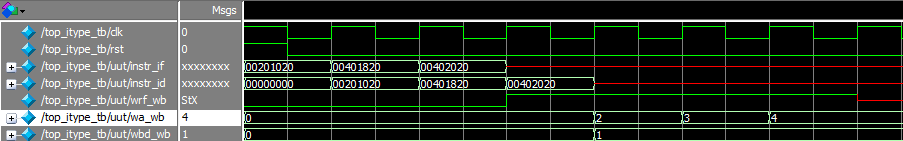


图2 alu结果处理数据相关性

2．Dmem结果回写寄存器

1. 非阻塞类型

如图1所示，对于具有数据相关性的汇编指令，如下代码所示，在准pipeline CPU中由于数据相关性，会造成最终写入寄存器一个错误的值，如下图1的4#寄存器所示，理论上应该写入的值是1，而实际上由于数据相关性问题，写入的值是0。当处理了数据相关性之后，跑想通的汇编指令，得到的结果如下图2所示，4#寄存器的值和预期的一致。

sw $1, 0($1)

lw $2, 0($1)

add $3, $1, $0

add $4, $2, $0

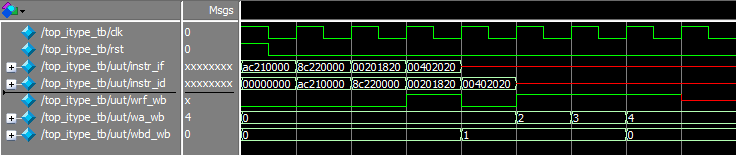


图1 lw结果不处理数据相关性

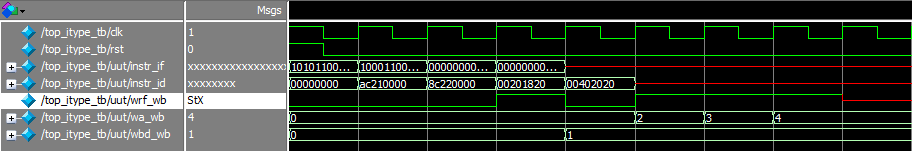


图2 lw结果处理数据相关性

1. 阻塞类型

如图1所示，对于具有数据相关性的汇编指令，如下代码所示，在准pipeline CPU中由于数据相关性，会造成最终写入寄存器一个错误的值，如下图1的3#寄存器所示，理论上应该写入的值是1，而实际上由于数据相关性问题，写入的值是0。当处理了数据相关性之后，让这条数据相关指令（add $3, $2, $0）暂停一个时钟周期，即在数据通路中跑两边，通过给出阻塞信号stall保证在第一次（废弃指令）跑的时候不会影响到寄存器和数据存储器的值，跑想通的汇编指令，得到的结果如下图2所示，3#寄存器的值和预期的一致。

sw $1, 0($1)

lw $2, 0($1)

add $3, $2, $0

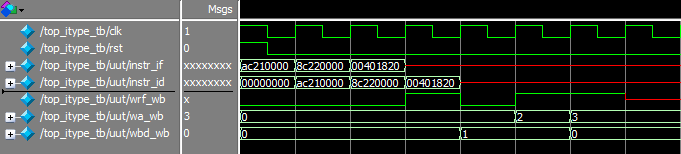
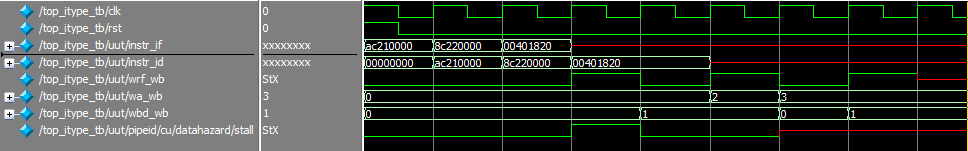


图1 lw不带阻塞功能结果不处理数据相关性

图2 lw带阻塞功能结果处理数据相关性