



Imagination大学计划

RVfpga实验14

结构冒险

1. 简介

在接下来的三个实验（实验14-16）中，我们将讨论**流水线冒险**。正如D. Patterson和J. Hennessy在其新书《计算机组织结构和设计》（RISC-V版本，Patterson & Hennessy, © Morgan Kaufmann 2017）[PaHe]的第4.5节中所述：流水线中存在下一条指令无法在下一个时钟周期执行的情况。这些事件称为冒险。有三种不同类型的冒险：**结构冒险**、**数据冒险**和**控制冒险**。

正如Patterson和Hennessy在[PaHe]中所述，因硬件不支持设置为执行的指令组合而导致指令无法执行时，将发生**结构冒险**。在本实验中，我们将分析SweRV EH1处理器中的结构冒险。

在实验15中，我们将分析SweRV EH1处理器中的第二种冒险（**数据冒险**）。正如Hennessy和Patterson在其《计算机体系结构：量化研究方法》[HePa]一书的第6版中所述：当流水线更改操作数读/写访问顺序而使此顺序与在非流水线处理器上顺序执行指令的顺序不同时，将发生数据冒险。

最后，第三类冒险称为**控制冒险**。正如S. Harris和D. Harris在其《数字设计和计算机体系结构：RISC-V版本》（我们称之为DDCARV）一书的第7.5.3节中所述，如果在取指发生时尚未决定接下来对哪条指令进行取指，则将发生**控制冒险**。在实验16中，我们将分析SweRV EH1处理器中的控制冒险。

2. SweRV EH1中的结构冒险

在本部分中，我们将说明SweRV EH1处理器中可能发生的结构冒险的两种情况。每种结构冒险能够以不同的方式解除，进而做出不同的性能-成本权衡。

为了说明第一种情况，我们将在第2.A部分中创建一个基于整数乘法指令（mul）的示例，同时会介绍此指令在SweRV EH1中的执行情况，之前的实验中尚未对此进行过分析。回想一下GSG的第3部分和第4部分，此指令属于SweRV EH1中支持的RISC-V M扩展（整数乘法和除法的标准扩展）。为了执行此指令，SweRV EH1处理器在乘法管道（参见实验11的图4）中实现了一个流水线式多周期乘法单元（即需要多个周期来计算结果的流水线乘法器），共分为三个阶段：M1、M2和M3。

具体来说，在本例中，两条mul指令在同一周期到达译码阶段。由于SweRV EH1只有一个乘法单元，因此当处理器“不支持当前设置为执行的指令组合”[PaHe]时，将发生结构冒险。在使用非流水线式多周期乘法器的处理器中，第二条mul指令必须等待第一条指令执行完成，这将对性能产生重要影响。不过（如上文所述），SweRV EH1中使用的是流水线式乘法器，因此第二条mul指令仅延迟一个周期，在第一条乘法指令完成乘法的第一阶段（M1）、继续第二阶段（M2）后立即开始执行。这种解决方案对硬件成本的影响适中（流水线结构比非流水线结构更昂贵），但能够在对性能影响很小（仅一个周期）的情况下解除结构冒险。

在第二个示例（第2.B部分）中，三条指令在同一周期内到达回写阶段，其中一条是在几个周期之前执行的非阻塞装载。原则上，由于SweRV EH1是2路超标量内核，因此不可能在一个周期内完成3条指令；不过（正如实验11中所展示的内容），SweRV EH1的寄存器文件具有第三个写端口，因此可避免这种情况下的结构冒险。由于需要额外的寄存器文件端口，因此这种解决方案对硬件成本的影响较大，但能够在不造成性能损失的情况下解除这种结构冒险。

附录A – 译码阶段两条同时进行的div指令：除了上述两个示例外，在本实验末尾的附录中，我们将说明另一个基于除法指令的示例。尽管这一示例没有严格说明结构冒险，但仍然非常值得关注，我们建议您也进行分析。

A. 译码阶段两条同时进行的mul指令

RISC-V M扩展还包括mul指令。此指令将rs1与rs2相乘，并将低位置于目标寄存器（rd）中。mul的机器语言指令如下（参见[DDCARV]的附录B）：

```
0000001 | rs2 | rs1 | 000 | rd | 0110011
```

任务：可以对mul指令进行与实验12中对算术-逻辑指令进行的研究类似的研究：查看通过各流水线阶段的指令流，分析控制位（根据SweRVref的第4部分，mul指令有一个称为mul_pkt_t的特定结构类型，并且模块dec_decode_ctl中定义了一个名为mul_p的信号）等。

乘法单元在模块exu_mul_ctl

（[RVfpgaPath]/RVfpga/src/SweRVolfSoC/SweRVEh1CoreComplex/exu/exu_mul_ctl.sv）

中实现。如前文所述，此单元是流水线式的，需要3个周期来计算结果。使用流水线式（与非流水线式相反）乘法器可减少由于结构冒险造成的性能损失。

任务：检查来自exu_mul_ctl的Verilog代码，了解乘法如何计算。请记住，RISC-V包括4条乘法指令（mul、mulh、mulhsu和mulhu），并且所有这些指令均必须受硬件支持。

作为一项可选练习，可以用自己的乘法单元或互联网上的乘法单元来替换此乘法单元。

图1中的示例执行两条mul指令，这两条指令包含在重复0xFFFF次迭代（即十进制65,535）的循环中。mul指令在图中以红色突出显示。本例中，mul指令的前后存在几条nop指令，用于将各迭代彼此隔离。通常，程序不执行任何有用的操作，仅用于说明mul指令导致的结构冒险。

```

.globl Test_Assembly
Test_Assembly:

li t2, 0xFFFF

li t3, 0x3
li t4, 0x2
li t5, 0x2
li t6, 0x2

REPEAT:
    beq t2, zero, OUT    # Stay in the loop?
    INSERT_NOPS_9
    mul t0, t3, t4        # t0 = t3 * t4
    mul t1, t5, t6        # t1 = t5 * t6
    INSERT_NOPS_9
    add t2, t2, -1
    add t0, zero, zero
    add t1, zero, zero
    j REPEAT
OUT:
.end

```

图1. 具有两条连续mul指令的示例

文件夹[RVfpgaPath]/RVfpga/Labs/Lab14/MUL_Instruction提供PlatformIO项目，以便可以根据需要分析、仿真和修改程序。此项目的结构基于实验11中供使用性能计数器的结构：它包含一个用于初始化、停止和打印所需计数器值的.c文件和一个包含从.c文件调用的待测试汇编程序（本例中为包含两条mul指令的循环）的.S文件。

在PlatformIO中打开、编译项目，然后打开反汇编文件（位于[RVfpgaPath]/RVfpga/Labs/Lab14/MUL_Instruction/.pio/build/swervolf_nexys/firmware.dis）。请注意，mul指令位于地址0x000001e8和0x000001ec处。

| | | | |
|-------------|----------|-----|----------|
| 0x000001e8: | 03de02b3 | mul | t0,t3,t4 |
| 0x000001ec: | 03ff0333 | mul | t1,t5,t6 |

任务：验证这对32位值（0x03de02b3和0x03ff0333）是否对应RISC-V架构中的指令mul t0,t3,t4和mul t1,t5,t6。

图2给出了循环第二次迭代期间图1中程序的仿真情况。

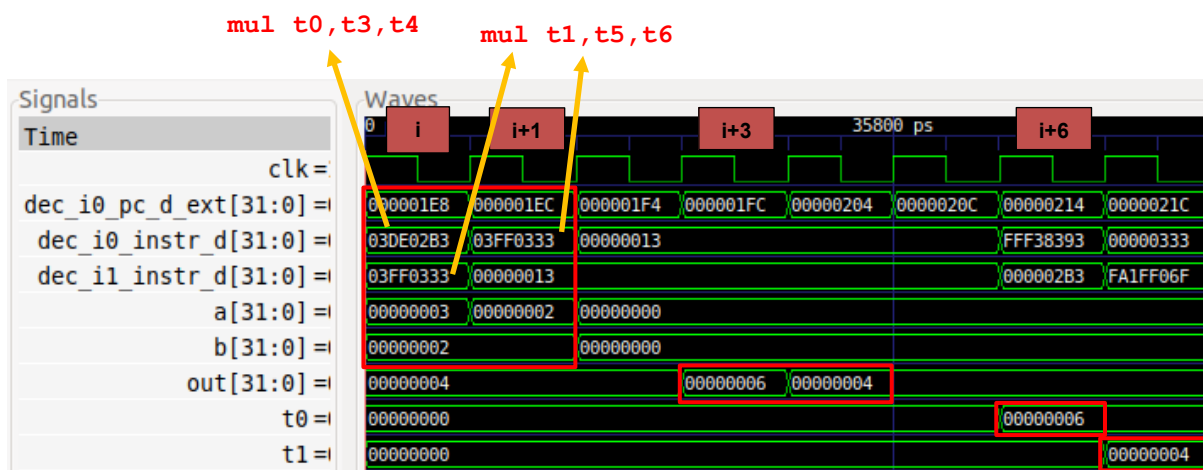


图2. 图1中示例的Verilator仿真

任务： 在自己的计算机上重复图2中的仿真过程，以进行更深入的分析。可以使用.tcl文件 [RVfpgaPath]/RVfpga/Labs/Lab14/MUL_Instruction/test.tcl

分析图2中的波形。以红色突出显示的值对应于与遍历流水线的mul指令相关的不同信号。

- **周期i：** 两条mul指令在同一周期到达译码阶段。结构冒险阻止第二条mul指令（dec_i1_instr_d = 0x03ff0333）进入下一阶段，前提是第一条mul指令（dec_i0_instr_d = 0x03de02b3）已调度到相应单元。
- **周期i+1：** 第一条mul指令在流水线乘法器的第一阶段（M1）执行，而第二条mul指令在译码阶段等待。
- **周期i+2：** 第一条mul指令在流水线乘法器的第二阶段（M2）执行，而第二条mul指令在第一个阶段（M1）执行。
- **周期i+3：** 当乘法的结果产生时（第一条mul指令为out = 0x6），第一条mul指令到达EX3。
- **周期i+4：** 当乘法的结果产生时（第二条mul指令为out = 0x4），第二条mul指令到达EX3。
- **周期i+6：** 寄存器文件使用第一条mul指令（t0 = 0x6）的结果更新。
- **周期i+7：** 寄存器文件使用第二条mul指令（t1 = 0x4）的结果更新。

图3所示为图1的示例中通过SweRV EH1流水线的指令流。**D**代表译码阶段，**A**代表对齐阶段，**C**代表提交阶段，**WB**代表回写阶段。当第一条mul指令译码时（周期i），大多数后续指令在其当前阶段暂停（图中用后缀st标记）并插入气泡（bubble）。在下一个周期（i+1）中，指令将恢复（请注意，第二条mul指令已从通路1移至通路0，并且通路1包含下一条指令（nop）。在周期i+2中，第一条mul指令处于执行的第二阶段（M2），而第二条mul指令处

于执行的第一阶段（M1）。在周期i+5和i+6中，两条mul指令将其结果写回到寄存器文件（可以看到在图2的周期i+6和i+7中更新）。

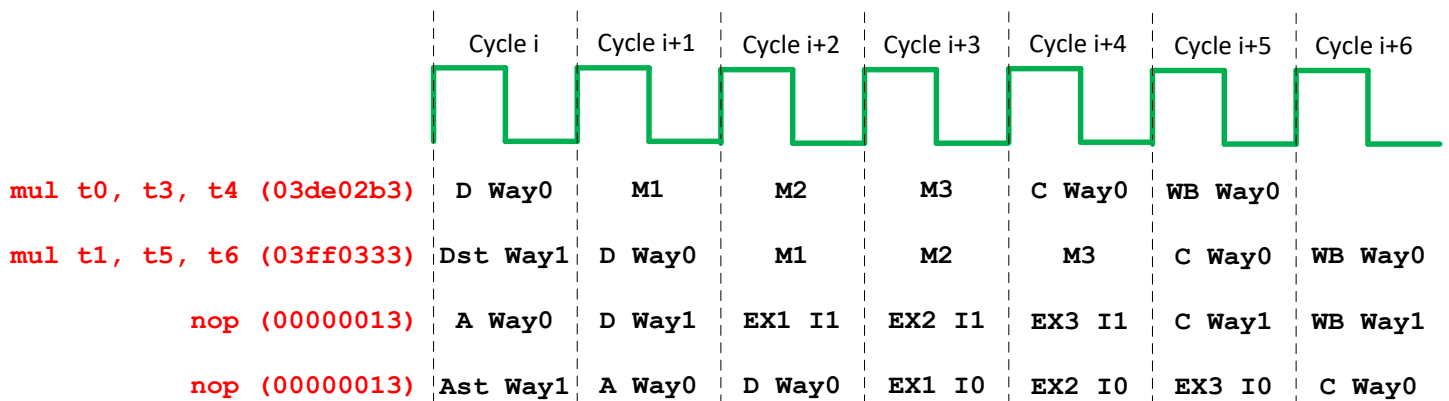


图3. 图1示例代码的执行

任务：将图3中的说明与图2中的仿真进行比较（重点关注两条mul指令）。具体来说，分析这两条指令如何在对齐和译码阶段分配给两个通路，及其如何通过流水线。

- 在模块ifu_aln_ctl（对齐阶段）中，这两条指令尽可能分配给以下信号：

- 通路0: ifu_i0_instr
- 通路1: ifu_i1_instr

- 在模块dec_ib_ctl中，这两条指令进行缓存（对齐到译码阶段）。请注意，在某些情况下，指令可暂停在这些缓冲区中并重新分配给不同的通路：

- 通路0: ifu_i0_instr → dec_i0_instr_d
- 通路1: ifu_i1_instr → dec_i1_instr_d

- 在模块dec_decode_ctl（译码阶段）中，这两条指令尽可能调度给相应管道。发送后，它们将继续进行三个执行阶段、提交阶段和回写阶段：

- 通路0: i0_inst_e1 → i0_inst_e2 → i0_inst_e3 → i0_inst_e4 → i0_inst_wb
- 通路1: i1_inst_e1 → i1_inst_e2 → i1_inst_e3 → i1_inst_e4 → i1_inst_wb

我们提供包括所有这些信号的.tcl文件

（[RVfpgaPath]/RVfpga/Labs/Lab14/MUL_Instruction/test_AssignmentWays.tcl）。

任务：从图1中删除循环中包含的nop指令并使用SweRV EH1中提供的性能计数器测量不同的事件（执行周期、已提交的指令/乘法数等），如实验11中所述。在分析图2中的仿真后，周期数是否符合预期？解释您的答案。

现在调整循环内的代码顺序，以达到理想的吞吐量。解释在原始代码和调整顺序的代码中获得的结果。

任务：文件夹[RVfpgaPath]/RVfpga/Labs/Lab14/MUL_Instr_Accumul_C-Lang提供了C程序的PlatformIO项目，此C程序用于累加循环中两次乘法的减法结果。

- 分析C程序。
- 执行仿真并检查循环的随机迭代。请注意，C程序在未经过优化的情况下编译。
- 使用SweRV EH1中提供的性能计数器测量不同的事件（周期、已提交的指令/乘法数等），如实验11中所述。
在分析图2中的仿真后，周期数是否符合预期？解释您的答案。
- 用RISC-V汇编语言创建一个相似的程序并将其与C版本进行比较。调整指令顺序以获得最佳IPC。
- 禁止C程序中的M RISC-V扩展并将结果与原始程序进行比较。为此，请将platformio.ini中的以下行：
 build_flags = -Wa,-march=rv32ima -march=rv32ima
修改为：
 build_flags = -Wa,-march=rv32ia -march=rv32ia
这样便可避免使用M RISC-V扩展中的指令，而是使用其他指令进行仿真。

任务：修改图1中的程序，将两条mul指令替换为两条lw指令（针对DCCM）。应观察到与本部分中分析的结构冒险类似并以相似方式解除的结构冒险。

B. 在回写阶段同时执行的三条指令

SweRV EH1是一个2路超标量处理器（GSG和之前的实验中已简要讨论了此功能，实验17中将更详细地分析）。这意味着每个周期可以在此处理器中执行两条指令。在三条指令在同一周期到达同一阶段的情况下，可能会发生结构冒险。鉴于SweRV EH1的结构，这种情况或许看起来不可能，但是，有一种特殊情形可能会发生这种冒险：

- 外部DDR2存储器具有中等延时，会强制装载指令暂停。当装载最终从存储器接收到数据时，它将进入回写阶段，在此阶段将读取值写入寄存器文件（假设此回写发生在周期*i*）。
- 如果装载是非阻塞的（即装载等待存储器中的数据到达时，处理器继续执行不依赖于此数据的指令），则可能发生以下情况：另外两条指令在周期*i*到达回写阶段，并且还需要写入寄存器文件（例如两条add指令）。
- 在这种情况下，三条指令将尝试在同一个周期（周期*i*）写入寄存器文件。

如果寄存器文件只有两个写端口，则将发生结构冒险，并且尝试写入的三条指令之一将必须等待寄存器文件空闲。但是，由于SweRV EH1中（如实验11所示）实现了第三个写端口，因此可以在不暂停（因而没有性能损失）的情况下解除这种结构冒险。

图4中的示例说明了这种情况，此示例依次执行1条非阻塞lw指令和36条add指令，这些指令包含在重复0xFFFF次迭代（即65,535）的循环中。lw指令在图中以红色突出显示。与lw指令在同一周期（周期*i*）到达回写阶段的两条add指令也进行突出显示。本例中不包括nop指令。通常，程序不执行任何有用的操作，仅用于说明本部分的示例。

```
REPEAT:
    lw x28, (x29)
    add x30, x30, -1
    add x1, x1, 1
    add x31, x31, 1
    add x3, x3, 1
    add x4, x4, 1
    add x5, x5, 1
    add x6, x6, 1
    add x7, x7, 1
    add x8, x8, 1
    add x9, x9, 1
    add x10, x10, 1
    add x11, x11, 1
    add x12, x12, 1
    add x13, x13, 1
    add x14, x14, 1
    add x15, x15, 1
    add x16, x16, 1
    add x17, x17, 1
    add x18, x18, 1
    add x19, x19, 1
    add x20, x20, 1
    add x21, x21, 1
    add x22, x22, 1
    add x23, x23, 1
    add x24, x24, 1
    add x25, x25, 1
    add x26, x26, 1
    add x27, x27, 1
    add x31, x31, 1
    add x3, x3, 1
    add x4, x4, 1
    add x5, x5, 1
    add x6, x6, 1
    add x25, x25, 1
    add x26, x26, 1
    add x27, x27, 1
    bne x30, zero, REPEAT    # Repeat the loop
```

图4. 依次执行非阻塞lw指令和36条A-L指令的示例

文件夹[RVfpgaPath]/RVfpga/Labs/Lab14/LW_Instruction_ExtMemory提供PlatformIO项目，以便可以根据需要分析、仿真和修改程序。此项目的结构基于实验11中供使用性能计数器的结构：它包含一个用于初始化、停止和打印所需计数器值的.c文件和一个包含从.c文件调用的待测试汇编程序（本例中为包含非阻塞lw指令的循环）的.S文件。

如图5所示，在`lsu_bus_intf`模块（总线接口）中获得的32位数据通过信号`lsu_nonblock_load_data[31:0]`提供给寄存器文件。此外，在译码阶段生成、经流水线寄存器传播的控制信号分别通过信号`dec_nonblock_load_waddr[4:0]`和`dec_nonblock_load_wen`提供给寄存器文件，向其告知何时在何处写入数据。这三个信号通过此结构中提供的第三个写端口（`waddr2`、`wen2`和`wd2`）进入寄存器文件，如图所示。请记住，在实验11的图6中，我们详细说明了寄存器文件。

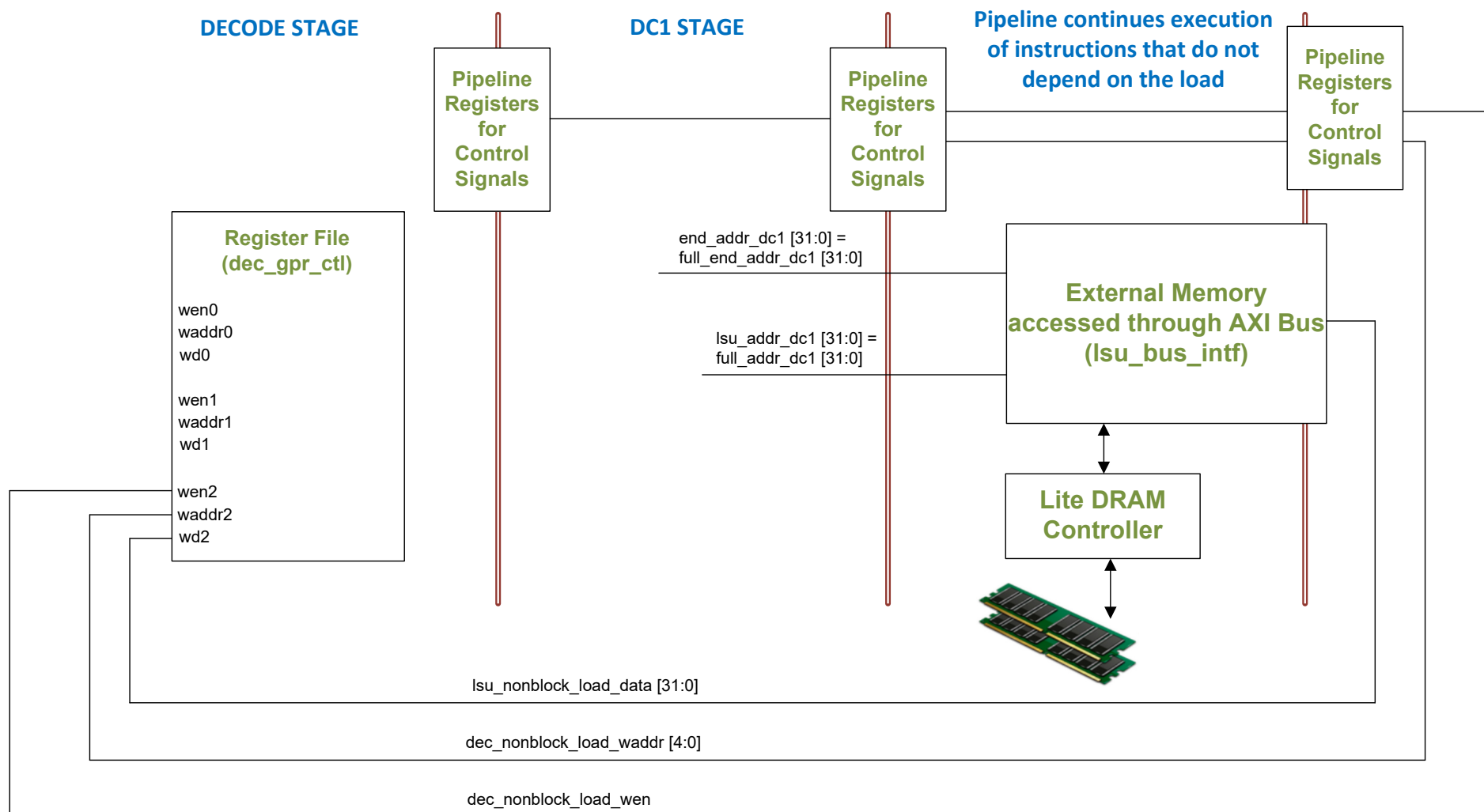


图5. 访问外部存储器的非阻塞装载指令



图6. 图4中示例的Verilator仿真

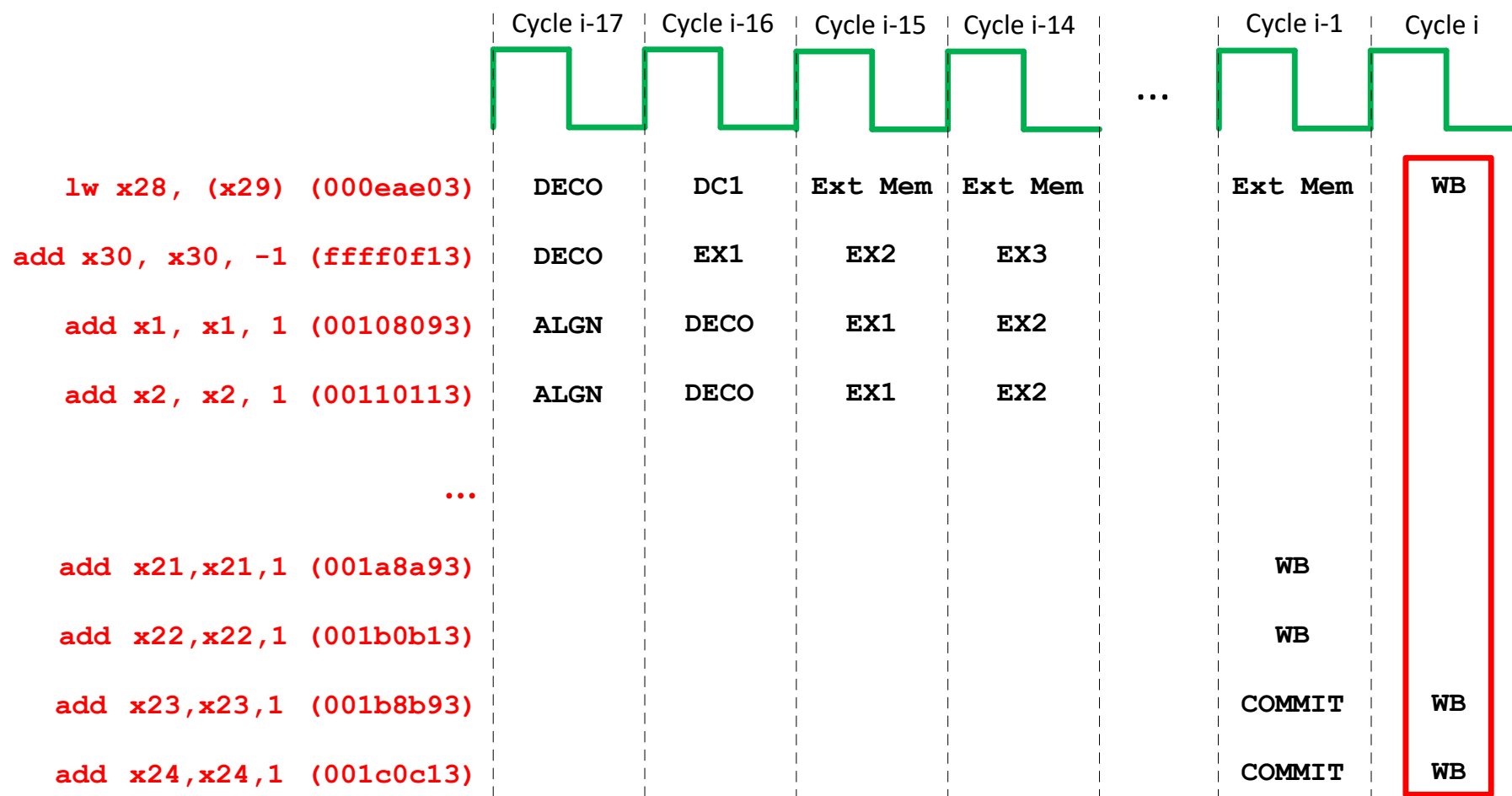



图7. 图4中示例代码的执行

图6和图7所示为图4中程序的Verilator仿真以及此程序针对循环的随机迭代的执行情况说明图。

任务： 在自己的计算机上重复图6中的仿真过程。使用文件 `test_NonBlocking.tcl`（在 `[RVfpgaPath]/RVfpga/Labs/Lab14/LW_Instruction_ExtMemory` 中提供）。单击几次 “Zoom In”（放大）（）移动至60120 ps。

分析图6中的波形以及图7中的图。

- **周期i-17：** lw指令处于译码阶段。
- **周期i-16：** 计算有效存储器地址并通过AXI总线发送到外部存储器。外部存储器的延时会强制装载指令等待几个周期，以便数据到达内核。
- **周期i-5：** 对两条冲突的add指令进行译码。
- **周期i：** lw指令和两条冲突的add指令进入回写阶段，必须全部写入寄存器文件。这可通过SweRV EH1寄存器文件中的三个写端口实现。请注意，寄存器编号在仿真中以十六进制显示。将写入x23、x24和x28（寄存器0x17、0x18和0x1c）。

任务： 将图6所示的仿真（非阻塞装载）与实验13的图14所示的仿真（阻塞装载）进行比较。添加比较需要的所有信号。

任务： 将图7中的说明与已复制到计算机上的图6中的仿真进行比较。根据需要添加信号以扩展仿真并加深理解。

任务： 使用SweRV EH1中提供的性能计数器测量不同的事件（周期、已提交的指令/装载等），如实验11中所述。在分析图6中的仿真后，周期数是否符合预期？解释您的答案。将这些结果与装载配置为阻塞装载时获得的结果进行比较。

3. 练习

1. 在仿真中以及在开发板上分析在同一周期到达L/S管道的两条连续访存指令（可以分析装载和存储等两条连续访存指令的任意组合）之间发生的结构冒险。可以使用以下位置提供的PlatformIO项目：

[RVfpgaPath]/RVfpga/Labs/Lab14/TwoConsecutiveLW_Instructions。

2. （以下练习基于《计算机组织结构和设计》（RISC-V版本，Patterson & Hennessy [PaHe]））中的练习4.22。）

请看下面的RISC-V汇编语言片段：

```
sw x29, 12(x16)
lw x29, 8(x16)
sub x17, x15, x14
beqz x17, label
add x15, x11, x14
sub x15, x30, x14
```

假设我们将SweRV EH1处理器修改为只有一个存储器（处理指令和数据）。在这种情况下，每次程序需要在一条指令访问数据的同一周期内对另一条指令取指时，均存在结构冒险。

- a. 绘制流水线图以显示上述代码将在SweRV EH1处理器的这一假想版本中的哪个位置暂停。
- b. 通常而言，能否通过调整代码顺序来减少暂停/nop的数量？
- c. 是否必须在硬件中处理这种结构冒险？可以看到，可通过在代码中添加nop来消除数据冒险。能否对这种结构冒险执行相同的操作？如果可以，请说明方法。如果不能，请说明原因。

附录A - 译码阶段两条同时进行的div指令

这一额外示例基于整数除法指令（div）。与mul指令一样，div指令属于SweRV EH1中支持的RISC-V M扩展（整数乘法和除法的标准扩展）。

div指令对rs1和rs2执行有符号整数除法并将结果存储在rd中。div的机器语言指令如下（参见[DDCARV]的附录B）：

```
0000001 | rs2 | rs1 | 100 | rd | 0110011
```

任务：可以对div指令进行与实验12中对算术-逻辑指令进行的研究类似的研究：查看通过各流水线阶段的指令流，分析控制位（根据SweRVref的第4部分，div指令有一个称为div_pkt_t的特定结构类型，并且模块dec_decode_ctl中定义了一个名为div_p的信号）等。

为了执行这条指令，SweRV EH1处理器在模块exu_div_ctl

（[RVfpgaPath]/RVfpga/src/SweRVolfSoC/SweRVEh1CoreComplex/exu/exu_div_ctl.sv）中实现了一个非流水线式阻塞多周期除法单元。此单元最多需要34个周期来计算结果，但根据输入的情况，也可能远小于此值。此除法单元向处理器输出几个信号（div_stall、finish_early和finish）来指示除法指令的状态。

任务：检查来自exu_div_ctl的Verilog代码，了解如何计算除法。此外，还要分析信号div_stall、finish_early和finish的影响。作为一项可选练习，可以用自己的除法单元或互联网上的除法单元来替换此除法单元。

图8中的示例执行两条div指令，这两条指令包含在重复0xFFFF次迭代（即十进制65,535）的循环中。div指令在图中以红色突出显示。与许多其他示例相反，在本例中，nop并非必需，因为div指令已因除法单元的高延时而与所有其他指令隔离。与我们之前使用的示例程序一样，此程序不执行任何有用的操作。

```
.globl Test_Assembly
Test_Assembly:

li t2, 0xFFFF

li t3, 0x8000000
li t4, 0x2
li t5, 0x2000000
li t6, 0x2

REPEAT:
    div t0, t3, t4          # t0 = t3 / t4
    div t1, t5, t6          # t1 = t5 / t6
    add t2, t2, -1
    add t0, zero, zero
    add t1, zero, zero
    bne t2, zero, REPEAT    # repeat the loop

.end
```

图8. 两条连续div指令的示例

文件夹`[RVfpgaPath]/RVfpga/Labs/Lab14/DIV_Instruction`提供PlatformIO项目，以便可以根据需要分析、仿真和更改程序。此项目的结构与`mul`指令使用的结构一样，基于实验11中供使用性能计数器的结构。

如果在PlatformIO中打开、编译项目，然后打开反汇编文件（位于`[RVfpgaPath]/RVfpga/Labs/Lab14/DIV_Instruction/.pio/build/swervolf_nexys/firmware.dis`中），则会发现`div`指令位于地址`0x000001c0`和`0x000001c4`处。

| | | | |
|--------------------------|-----------------------|------------------|-----------------------|
| <code>0x000001c0:</code> | <code>03de42b3</code> | <code>div</code> | <code>t0,t3,t4</code> |
| <code>0x000001c4:</code> | <code>03ff4333</code> | <code>div</code> | <code>t1,t5,t6</code> |

任务：验证这对32位值（`0x03de42b3`和`0x03ff4333`）是否对应指令RISC-V架构中的指令`div t0,t3,t4`和`div t1,t5,t6`。

图9给出了循环随机迭代期间图8中程序的仿真情况。

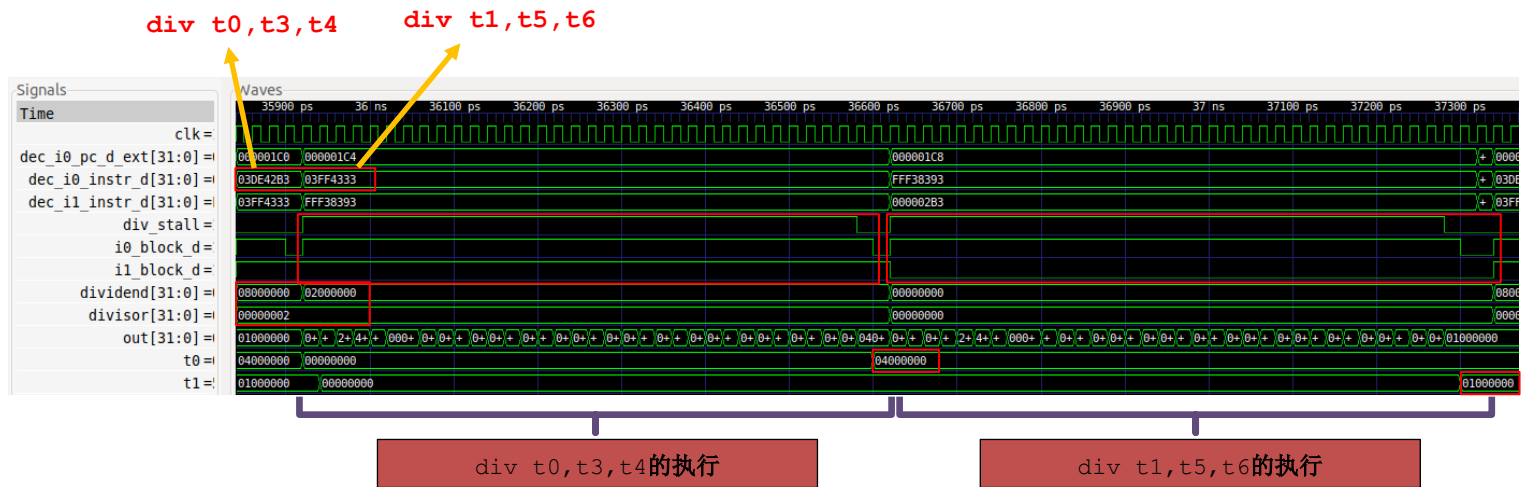


图9. 图8中示例的Verilator仿真

任务： 在自己的计算机上重复图9中的仿真过程，以进行详细的分析。

分析图9中的波形。以红色突出显示的值是与遍历流水线的两条div指令相关的信号。

- 两条div指令在同一周期到达译码阶段（`dec_i0_pc_d_ext = 0x000001c0`，即第一条div的指令地址）。第一条div指令（`0x03de42b3`）调度为在除法单元中执行，因此将被除数和除数（被除数 = `0x08000000`，除数 = `0x00000002`）发送到此单元。请注意，我们为被除数选择了较大的值，以使除法计算时间接近最大值（34个周期）。

鉴于SweRV EH1中的除法是阻塞的，div后的任何其他指令都将暂停。但是请注意，即使除法是非阻塞的，由于只有一个除数而造成的结构冒险也会使第二条div指令暂停。如第2部分中所述，还有其他方法可以提高性能，例如将除法器流水线化或包含另一个除法器。不过，考虑到除法不是频繁操作，这种情况下主要考虑降低硬件成本。

- 流水线在第一条div指令的执行期间暂停（参见第一次除法计算期间的信号 `div_stall = 1`）。还可以看到，当信号 `i0_block_d` 和 `i1_block_d` 为1时阻塞，通路0和通路1被阻塞。此外，现在 `dec_i0_pc_d_ext = 0x000001c4`，这是在译码阶段暂停的第二条除法指令的地址。
- 除法单元的信号 `out` 在34个周期后提供结果，此结果写入目标寄存器（`t0 = 0x04000000`）。可以看到随着除法运算逐步得出最终结果，每个周期的输出值如何变化。
- 得到结果后，除数被释放，流水线继续执行（`div_stall = 0`），第二条div指令调度给除法单元。随后，在34个周期后，第二条div指令的结果写入寄存器文件（`t1 = 0x01000000`）。

与第一条div指令一样，由于阻塞除数的原因，第二条div指令之后的所有指令必须暂停。不过，在这种情况下，不依赖于t1的指令可以继续（如果是非阻塞除法）。

图10所示为图8的示例中通过SweRV EH1流水线的指令流。当第一条div指令进行译码时（周期i），由于SweRV EH1阻塞除法和除法单元中结构冒险的原因，第二条div指令和后续指令在其当前阶段暂停。（暂停的指令在图中用后缀-st标记。）之后，在34个周期后（周期i+34），第一条div指令完成执行，并通过2:1多路开关将结果写回到寄存器文件，如实验11的图4所示。在接下来的周期中（i+35），后续指令恢复。然后，在周期36中，第二条div指令开始执行，由于SweRV EH1阻塞除法的原因，后续指令再次暂停。

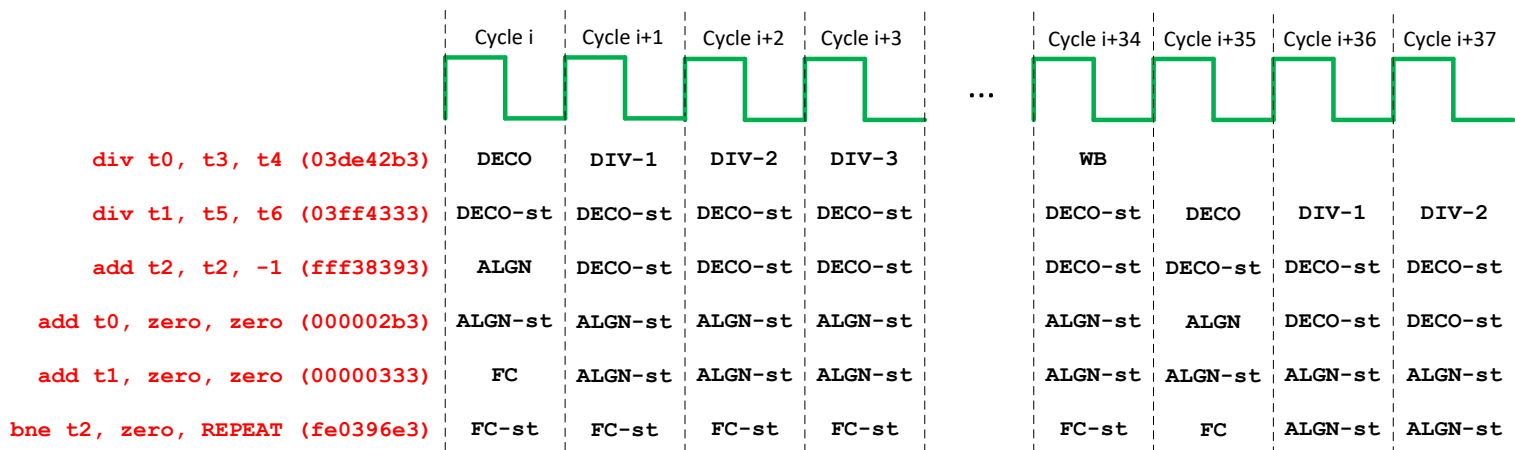


图10. 图8示例代码的执行（-st后缀表示暂停指令）

任务：将图10中的说明与已复制到计算机上的图9中的仿真进行比较。根据需要添加信号以扩展仿真并加深理解。

任务：使用SweRV EH1中提供的性能计数器测量不同的事件（周期、已提交的指令/除法），如实验11中所述。
在分析图9中的仿真后，周期数是否符合预期？解释您的答案。

任务：尝试不同的被除数和除数，了解用于计算结果的周期数与其值的相关性。通过仿真和硬件计数器查看实验。

任务： 文件夹[RVfpgaPath]/RVfpga/Labs/Lab14/DIV_Instr_Accumul_C-Lang提供C程序的PlatformIO项目，此C程序用于累加循环中两次除法的减法结果。

- 分析C程序。
- 执行仿真并检查循环的随机迭代。请注意，C程序在未经过优化的情况下编译。
- 使用SweRV EH1中提供的性能计数器测量不同的事件（周期、已提交的指令/除法等），如实验11中所述。

在分析图9中的仿真后，周期数是否符合预期？解释您的答案。

- 用RISC-V汇编语言创建一个相似的程序并将其与C版本进行比较。
- 禁止C程序中的M RISC-V扩展并将结果与原始程序进行比较。为此，请将platformio.ini中的以下行：

```
build_flags = -Wa,-march=rv32ima -march=rv32ima
```

修改为：

```
build_flags = -Wa,-march=rv32ia -march=rv32ia
```

这样便可避免使用RISC-V M扩展中的指令，而是使用其他指令进行仿真。

任务： 在SweRV EH1中，div指令是阻塞的。修改处理器以允许非阻塞div指令。

然后向SweRV EH1处理器添加第二个除法器，以便图8示例中的两条div指令可以并行执行。