



IMAGINATION大學計劃

# RVfpga實驗14

## 結構冒險

## 1. 簡介

在接下來的三個實驗（實驗14-16）中，我們將討論**管線冒險**。正如D. Patterson和J. Hennessy在其新書《電腦組織結構和設計》（RISC-V版本，Patterson & Hennessy，© Morgan Kaufmann 2017）[PaHe]的第4.5節中所述：管線中存在下一條指令無法在下一個時鐘週期執行的情況。這些事件稱為冒險。有三種不同類型的冒險：**結構冒險**、**資料冒險**和**控制冒險**。

正如Patterson和Hennessy在[PaHe]中所述，因硬體不支援設定為執行的指令組合而導致指令無法執行時，將發生**結構冒險**。在本實驗中，我們將分析SweRV EH1處理器中的結構冒險。

在實驗15中，我們將分析SweRV EH1處理器中的第二種冒險（**資料冒險**）。正如Hennessy和Patterson在其《電腦體系結構：量化研究方法》[HePa]一書的第6版中所述：當管線變更運算元讀/寫存取順序而使此順序與在非管線處理器上順序執行指令的順序不同時，將發生資料冒險。

最後，第三類冒險稱為**控制冒險**。正如S. Harris和D. Harris在其《數位設計和電腦體系結構：RISC-V版本》（我們稱之為DDCARV）一書的第7.5.3節中所述，如果在擷取發生時尚未決定接下來對哪條指令進行擷取，則將發生**控制冒險**。在實驗16中，我們將分析SweRV EH1處理器中的控制冒險。

## 2. SweRV EH1中的結構冒險

在本部分中，我們將說明SweRV EH1處理器中可能發生的結構冒險的兩種情況。每種結構冒險能夠以不同的方式解除，進而做出不同的效能-成本權衡。

為了說明第一種情況，我們將在第2.A部分中建立一個基於整數乘法指令（mul）的範例，同時會介紹此指令在SweRV EH1中的執行情況，之前的實驗中尚未對此進行過分析。回想一下GSG的第3部分和第4部分，此指令屬於SweRV EH1中支援的RISC-V M延伸（整數乘法和除法的標準延伸）。為了執行此指令，SweRV EH1處理器在乘法管道（參見實驗11的圖4）中實作了一個管線式多週期乘法單元（即需要多個週期來計算結果的管線乘法器），共分為三個階段：M1、M2和M3。

具體來說，在本例中，兩條mul指令在同一週期到達解碼階段。由於SweRV EH1只有一個乘法單元，因此當處理器「不支援目前設定為執行的指令組合」[PaHe]時，將發生結構冒險。在使用非管線式多週期乘法器的處理器中，第二條mul指令必須等待第一條指令執行完成，這將對效能產生重要影響。不過（如上文所述），SweRV EH1中使用的是管線式乘法器，因此第二條mul指令僅延遲一個週期，在第一條乘法指令完成乘法的第一階段（M1）、繼續第二階段（M2）後立即開始執行。這種解決方案對硬體成本的影響適中（管線結構比非管線結構更昂貴），但能夠在對效能影響很小（僅一個週期）的情況下解除結構冒險。

在第二個範例（第2.B部分）中，三條指令在同一週期內到達寫回階段，其中一條是在幾個週期之前執行的非阻塞載入。原則上，由於SweRV EH1是2路超標量核心，因此不可能在同一個週期內完成3條指令；不過（正如實驗11中所展示的內容），SweRV EH1的暫存器檔案具有第三個寫入連接埠，因此可避免這種情況下的結構冒險。由於需要額外的暫存器檔案連接埠，因此這種解決方案對硬體成本的影響較大，但能夠在不造成效能損失的情況下解除這種結構冒險。

**附錄A – 解碼階段兩條同時進行的div指令：**除了上述兩個範例外，在本實驗末尾的附錄中，我們將說明另一個基於除法指令的範例。儘管這一範例沒有嚴格說明結構冒險，但仍然非常值得關注，我們建議您也進行分析。

## A. 解碼階段兩條同時進行的mul指令

RISC-V M延伸還包括mul指令。此指令將rs1與rs2相乘，並將低位元置於目標暫存器（rd）中。mul的機器語言指令如下（參見[DDCARV]的附錄B）：

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

**任務：**可以對mul指令進行與實驗12中對算術-邏輯指令進行的研究類似的研究：查看透過各管線階段的指令流，分析控制位元（根據SweRVref的第4部分，mul指令有一個稱為mul\_pkt\_t的特定結構類型，並且模組dec\_decode\_ctl中定義了一個名為mul\_p的訊號）等。

乘法單元在模組exu\_mul\_ctl

（[RVfpgaPath]/RVfpga/src/SweRVolfSoC/SweRVEh1CoreComplex/exu/exu\_mul\_ctl.sv）

中實作。如前文所述，此單元是管線式的，需要3個週期來計算結果。使用管線式（與非管線式相反）乘法器可減少由於結構冒險造成的效能損失。

**任務：**檢查來自exu\_mul\_ctl的Verilog程式碼，瞭解乘法如何計算。請記住，RISC-V包括4條乘法指令（mul、mulh、mulhsu和mulhu），並且所有這些指令均必須受硬體支援。

作為一項選用練習，可以用自己的乘法單元或網際網路上的乘法單元來替換此乘法單元。

圖1中的範例執行兩條mul指令，這兩條指令包含在重複0xFFFF次迭代（即十進位65,535）的迴圈中。mul指令在圖中以紅色強調顯示。本例中，mul指令的前後存在幾條nop指令，用於將各迭代彼此隔離。通常，程式不執行任何有用的操作，僅用於說明mul指令導致的結構冒險。

```

.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

```

圖1. 具有兩條連續mul指令的範例

資料夾[RVfpgaPath]/RVfpga/Labs/Lab14/MUL\_Instruction提供PlatformIO專案，以便可以根據需要分析、模擬和修改程式。此專案的結構基於實驗11中供使用效能計數器的結構：它包含一個用於初始化、停止和輸出所需計數器值的.c檔案和一個包含從.c檔案叫用的待測試組合語言程式（本例中為包含兩條mul指令的迴圈）的.S檔案。

在PlatformIO中開啟、編譯專案，然後開啟反組譯檔案（位於[RVfpgaPath]/RVfpga/Labs/Lab14/MUL\_Instruction/.pio/build/swervolf\_nexys/firmware.dis）。請注意，mul指令位於位址0x000001e8和0x000001ec處。

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

**任務：**驗證這對32位元值（0x03de02b3和0x03ff0333）是否對應RISC-V架構中的指令mul t0, t3, t4和mul t1, t5, t6。

圖2顯示了迴圈第二次迭代期間圖1中程式的模擬情況。

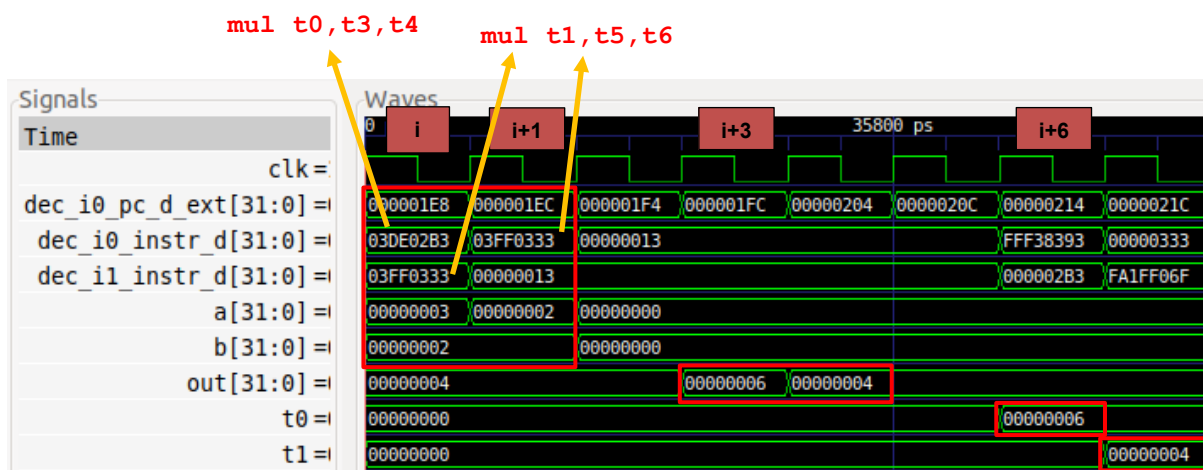


圖2. 圖1中範例的Verilog模擬

**任務：**在自己的電腦上重複圖2中的模擬過程，以進行更深入的分析。可以使用.tcl檔案 [RVfpgaPath]/RVfpga/Labs/Lab14/MUL\_Instruction/test.tcl

分析圖2中的波形。以紅色強調顯示的值對應於與遍歷管線的mul指令相關的不同訊號。

- **週期i：**兩條mul指令在同一週期到達解碼階段。結構冒險阻止第二條mul指令（dec\_i1\_instr\_d = 0x03ff0333）進入下一階段，前提是第一條mul指令（dec\_i0\_instr\_d = 0x03de02b3）已調度到相應單元。
- **週期i+1：**第一條mul指令在管線乘法器的第一階段（M1）執行，而第二條mul指令在解碼階段等待。
- **週期i+2：**第一條mul指令在管線乘法器的第二階段（M2）執行，而第二條mul指令在第一階段（M1）執行。
- **週期i+3：**當乘法的結果產生時（第一條mul指令為out = 0x6），第一條mul指令到達EX3。
- **週期i+4：**當乘法的結果產生時（第二條mul指令為out = 0x4），第二條mul指令到達EX3。
- **週期i+6：**暫存器檔案使用第一條mul指令（t0 = 0x6）的結果更新。
- **週期i+7：**暫存器檔案使用第二條mul指令（t1 = 0x4）的結果更新。

圖3所示為圖1的範例中通過SweRV EH1管線的指令流。**D**代表解碼階段，**A**代表對齊階段，**C**代表提交階段，**WB**代表寫回階段。當第一條mul指令解碼時（週期i），大多數後續指令在其目前階段暫停（圖中用字尾st標記）並插入泡泡（bubble）。在下一個週期（i+1）中，指令將恢復（請注意，第二條mul指令已從通路1移至通路0，並且通路1包含下一條指令（nop）。在週期i+2中，第一條mul指令處於執行的第二階段（M2），而第二條mul指令處

於執行的第一階段（M1）。在週期i+5和i+6中，兩條mul指令將其結果寫回到暫存器檔案（可以看到在圖2的週期i+6和i+7中更新）。

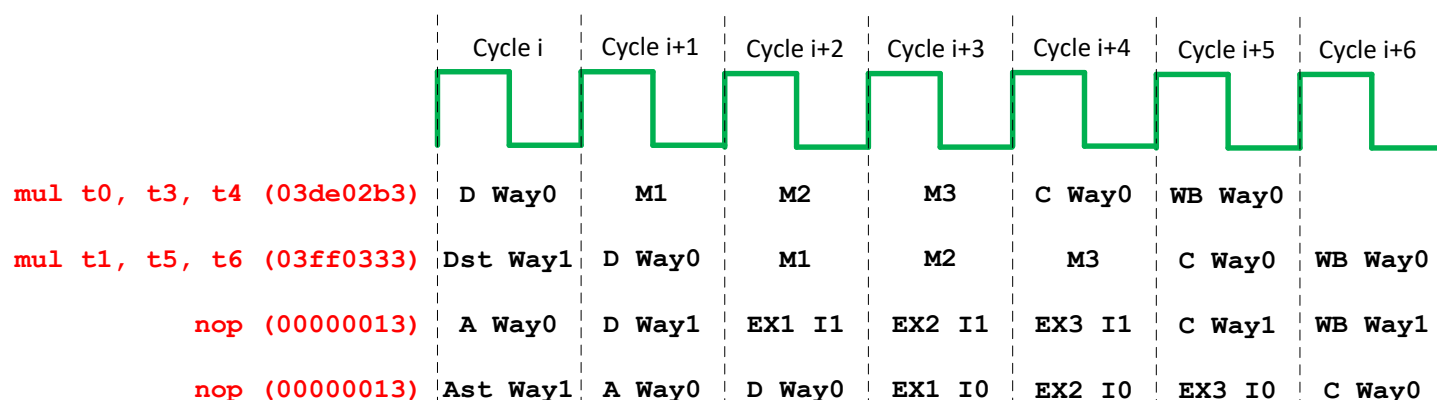


圖3. 圖1範例程式碼的執行

**任務：**將圖3中的說明與圖2中的模擬進行比較（重點關注兩條mul指令）。具體來說，分析這兩條指令如何在對齊和解碼階段指定給兩個通路，及其如何通過管線。

- 在模組ifu\_aln\_ctl（對齊階段）中，這兩條指令盡可能指定給以下訊號：
  - 通路0：ifu\_i0\_instr
  - 通路1：ifu\_i1\_instr
- 在模組dec\_ib\_ctl中，這兩條指令進行緩衝（對齊到解碼階段）。請注意，在某些情況下，指令可暫停在這些緩衝區中並重新指定給不同的通路：
  - 通路0：ifu\_i0\_instr → dec\_i0\_instr\_d
  - 通路1：ifu\_i1\_instr → dec\_i1\_instr\_d
- 在模組dec\_decode\_ctl（解碼階段）中，這兩條指令盡可能調度給相應管道。傳送後，它們將繼續進行三個執行階段、提交階段和寫回階段：
  - 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

我們提供包括所有這些訊號的.tcl檔案

（[RVfpgaPath]/RVfpga/Labs/Lab14/MUL\_Instruction/test\_AssignmentWays.tcl）。

**任務：**從圖1中刪除迴圈中包含的nop指令並使用SweRV EH1中提供的效能計數器測量不同的事件（執行週期、已提交的指令/乘法數等），如實驗11中所述。在分析圖2中的模擬後，週期數是否符合預期？證明您的答案。

現在調整迴圈內的程式碼順序，以達到理想的吞吐量。解釋在原始程式碼和調整順序的程式碼中取得的結果。

**任務：**資料夾[RVfpgaPath]/RVfpga/Labs/Lab14/MUL\_Instr\_Accumul\_C-Lang提供了C程式的PlatformIO專案，此C程式用於累加迴圈中兩次乘法的減法結果。

- 分析C程式。
- 執行模擬並檢查迴圈的隨機迭代。請注意，C程式在未經過最佳化的情況下編譯。
- 使用SweRV EH1中提供的效能計數器測量不同的事件（週期、已提交的指令/乘法數等），如實驗11中所述。  
在分析圖2中的模擬後，週期數是否符合預期？證明您的答案。

- 用RISC-V組合語言建立一個相似的程式並將其與C版本進行比較。調整指令順序以獲得最佳IPC。

- 停用C程式中的M RISC-V延伸功能並將結果與原始程式進行比較。為此，請將platformio.ini中的以下行：

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

修改為：

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

這樣便可避免使用M RISC-V延伸功能中的指令，而是使用其他指令進行模擬。

**任務：**修改圖1中的程式，將兩條mul指令替換為兩條lw指令（針對DCCM）。應觀察到與本部分中分析的結構冒險類似並以相似方式解除的結構冒險。

## B. 在寫回階段同時執行的三條指令

SweRV EH1是一個2路超標量處理器（GSG和之前的實驗中已簡要討論了此功能，實驗17中將更詳細地分析）。這意味著每個週期可以在此處理器中執行兩條指令。在三條指令在同一週期到達同一階段的情況下，可能會發生結構冒險。鑒於SweRV EH1的結構，這種情況或許看起來不可能，但是，有一種特殊情形可能會發生這種冒險：

- 外部DDR2記憶體具有中等延遲，會強制載入指令暫停。當載入最終從記憶體接收到資料時，它將進入寫回階段，在此階段將讀取值寫入暫存器檔案（假設此寫回發生在週期*i*）。
- 如果載入是非阻塞的（即載入等待記憶體中的資料到達時，處理器繼續執行不依賴於此資料的指令），則可能發生以下情況：另外兩條指令在週期*i*到達寫回階段，並且還需要寫入暫存器檔案（例如兩條add指令）。
- 在這種情況下，三條指令將嘗試在同一個週期（週期*i*）寫入暫存器檔案。

如果暫存器檔案只有兩個寫入連接埠，則將發生結構冒險，並且嘗試寫入的三條指令之一將必須等待暫存器檔案空間。但是，由於SweRV EH1中（如實驗11所示）實作了第三個寫入連接埠，因此可以在不暫停（因而沒有效能損失）的情況下解除這種結構冒險。

圖4中的範例說明了這種情況，此範例依次執行1條非阻塞lw指令和36條add指令，這些指令包含在重複0xFFFF次迭代（即65,535）的迴圈中。lw指令在圖中以紅色強調顯示。與lw指令在同一週期（週期*i*）到達寫回階段的兩條add指令也進行強調顯示。本例中不包括nop指令。通常，程式不執行任何有用的操作，僅用於說明本部分的範例。

```
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
```

圖4. 依次執行非阻塞lw指令和36條A-L指令的範例

資料夾[RVfpgaPath]/RVfpga/Labs/Lab14/LW\_Instruction\_ExtMemory提供PlatformIO專案，以便可以根據需要分析、模擬和修改程式。此專案的結構基於實驗11中供使用效能計數器的結構：它包含一個用於初始化、停止和輸出所需計數器值的.c檔案和一個包含從.c檔案叫用的待測試組合語言程式（本例中為包含非阻塞lw指令的迴圈）的.S檔案。



如圖5所示，在lsu\_bus\_intf模組（匯流排介面）中取得的32位透元資料過訊號lsu\_nonblock\_load\_data[31:0]提供給暫存器檔案。此外，在解碼階段產生、經管線暫存器傳播的控制訊號分別透過訊號dec\_nonblock\_load\_waddr[4:0]和dec\_nonblock\_load\_wen提供給暫存器檔案，向其告知何時在何處寫入資料。這三個訊號透過此結構中提供的第三個寫入連接埠（waddr2、wen2和wd2）進入暫存器檔案，如圖所示。請記住，在實驗11的圖6中，我們詳細說明了暫存器檔案。

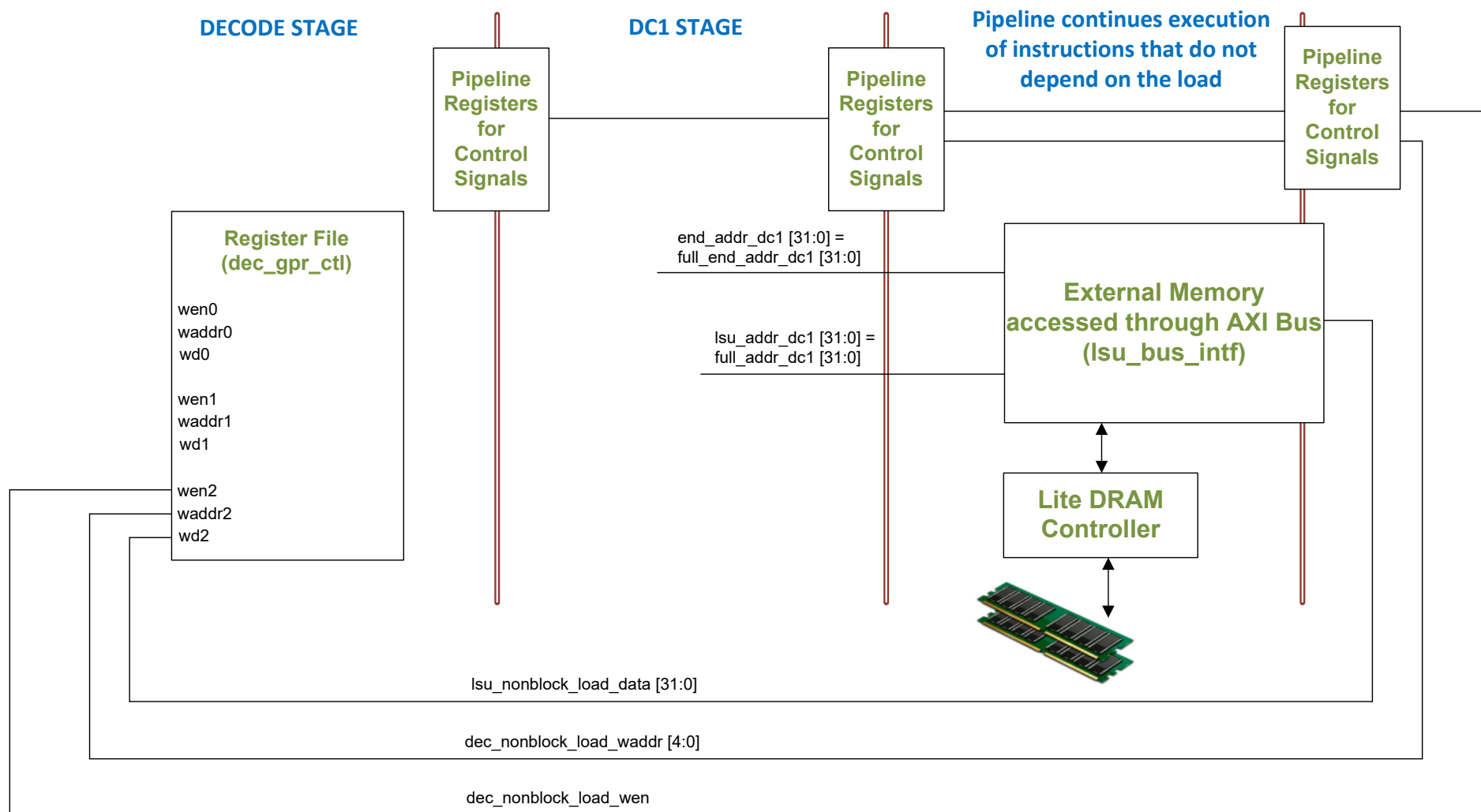
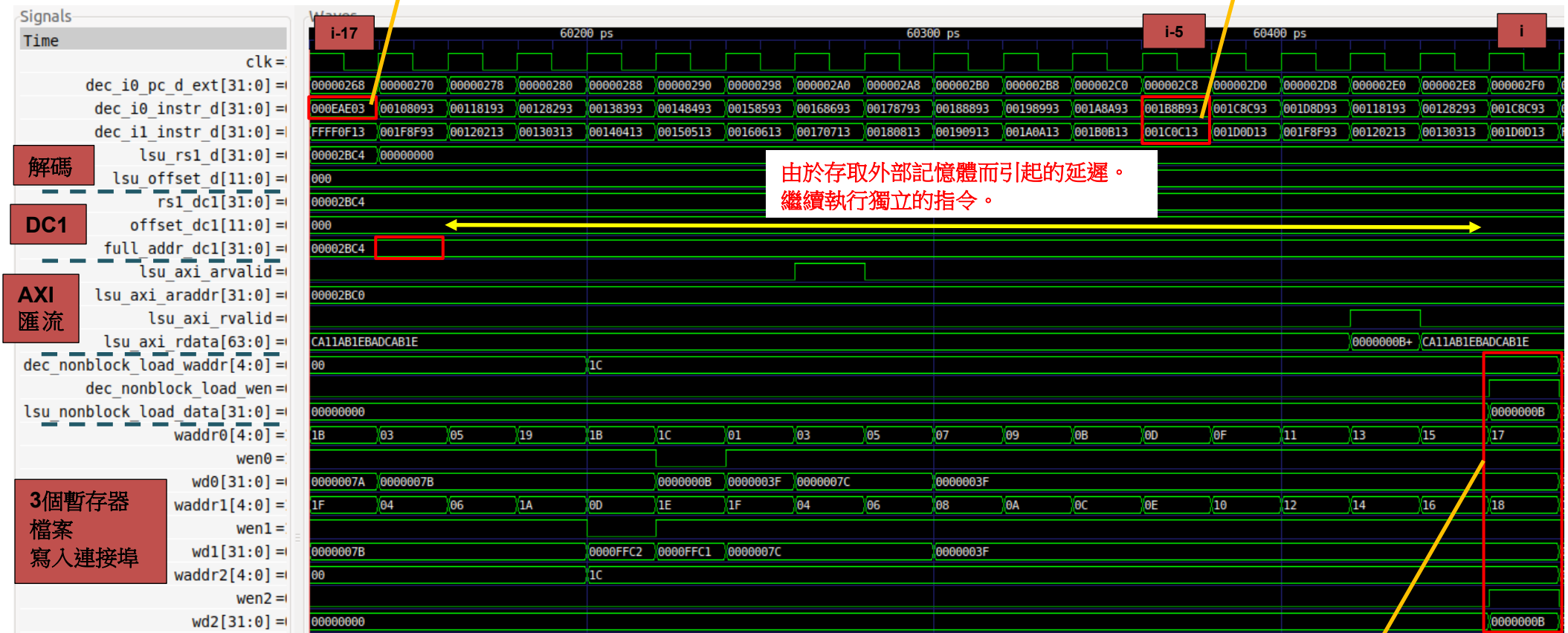


圖5. 存取外部記憶體的非阻塞載入指令



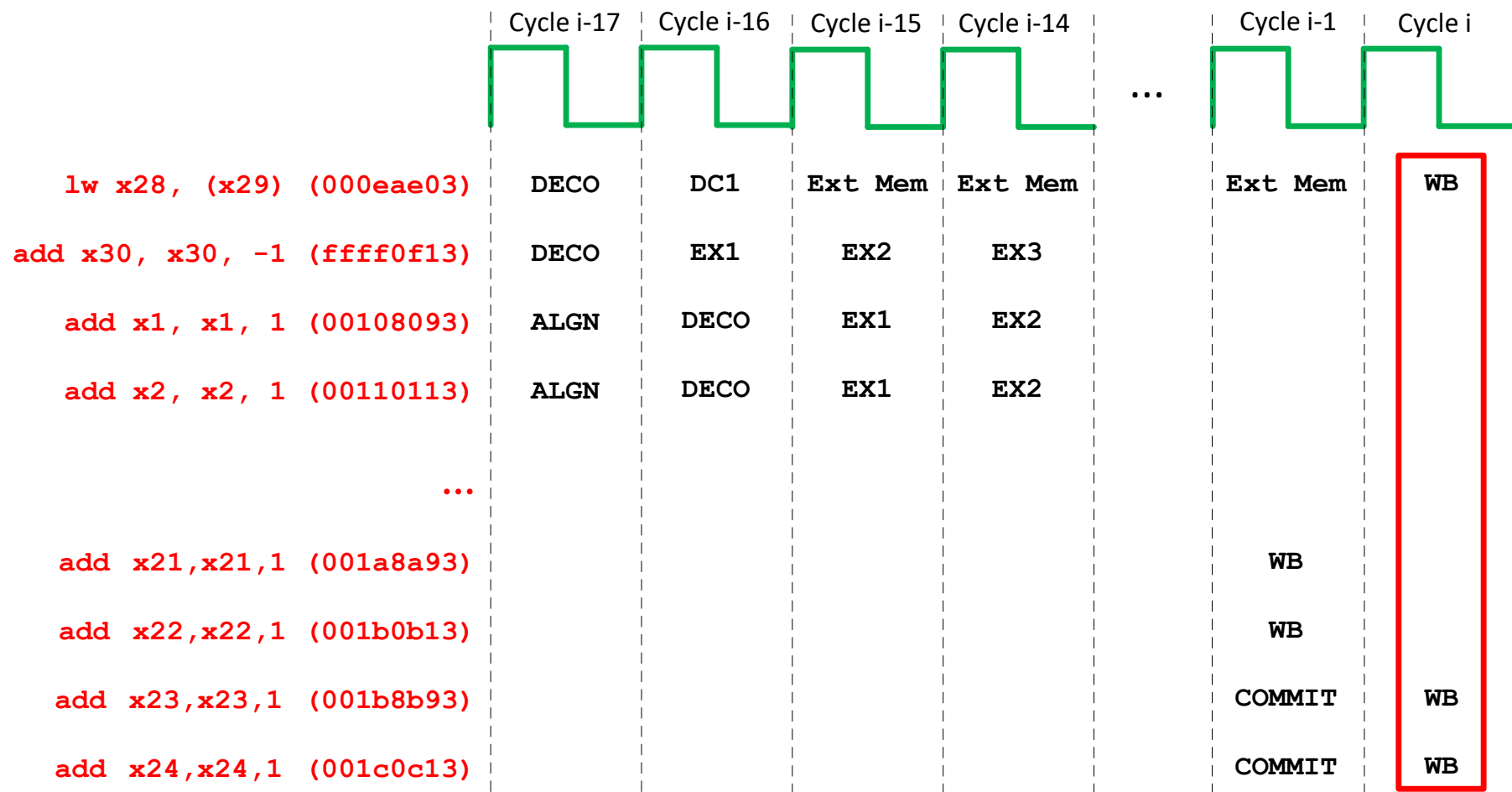


圖7. 圖4中範例程式碼的執行

圖6和圖7所示為圖4中程式的Verilator模擬以及此程式針對迴圈的隨機迭代的執行情況說明圖。

**任務：**在自己的電腦上重複圖6中的模擬過程。使用檔案`test_NonBlocking.tcl`（在`[RVfpgaPath]/RVfpga/Labs/Lab14/LW_Instruction_ExtMemory`中提供）。按幾次「Zoom In」（放大）（）移動至60120 ps。

分析圖6中的波形以及圖7中的圖。

- **週期i-17：**lw指令處於解碼階段。
- **週期i-16：**計算有效記憶體位址並透過AXI匯流排傳送到外部記憶體。外部記憶體的延遲會強制載入指令等待幾個週期，以便資料到達核心。
- **週期i-5：**對兩條衝突的add指令進行解碼。
- **週期i：**lw指令和兩條衝突的add指令進入寫回階段，必須全部寫入暫存器檔案。這可透過SweRV EH1暫存器檔案中的三個寫入連接埠實作。請注意，暫存器編號在模擬中以十六進位顯示。將寫入x23、x24和x28（暫存器0x17、0x18和0x1c）。

**任務：**將圖6所示的模擬（非阻塞載入）與實驗13的圖14所示的模擬（阻塞載入）進行比較。新增比較需要的所有訊號。

**任務：**將圖7中的說明與已複製到電腦上的圖6中的模擬進行比較。根據需要新增訊號以擴展模擬並加深理解。

**任務：**使用SweRV EH1中提供的效能計數器測量不同的事件（週期、已提交的指令/載入等），如實驗11中所述。在分析圖6中的模擬後，週期數是否符合預期？解釋您的答案。將這些結果與載入設定為阻塞載入時取得的結果進行比較。

### 3. 練習

1. 在模擬中以及在開發板上分析在同一週期到達L/S管道的兩條連續記憶體存取指令（可以分析載入和儲存等兩條連續記憶體存取指令的任意組合）之間發生的結構冒險。可以使用以下位置提供的PlatformIO專案：

**[RVfpgaPath]/RVfpga/Labs/Lab14/TwoConsecutiveLW\_Instructions**。

2. （以下練習基於《電腦組織結構和設計》（RISC-V版本，Patterson & Hennessy ([PaHe])）中的練習4.22。）

請看下面的RISC-V組合語言片段：

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

假設我們將SweRV EH1處理器修改為只有一個記憶體（處理指令和資料）。在這種情況下，每次程式需要在一條指令存取資料的同一週期內擷取另一條指令時，均存在結構冒險。

- a. 繪製管線圖以顯示上述程式碼將在SweRV EH1處理器的這一假想版本中的哪個位置暫停。
- b. 通常而言，能否透過調整程式碼順序來減少暫停/nop的數量？
- c. 是否必須在硬體中處理這種結構冒險？可以看到，可透過在程式碼中新增nop來消除資料冒險。能否對這種結構冒險執行相同的操作？如果可以，請說明方法。如果不能，請說明原因。

## 附錄A - 解碼階段兩條同時進行的DIV指令

這一額外範例基於整數除法指令（div）。與mul指令一樣，div指令屬於SweRV EH1中支援的RISC-V M延伸（整數乘法和除法的標準延伸）。

div指令對rs1和rs2執行有符號整數除法並將結果儲存在rd中。div的機器語言指令如下（參見[DDCARV]的附錄B）：

```
00000001 | rs2 | rs1 | 100 | rd | 0110011
```

**任務：**可以對div指令進行與實驗12中對算術-邏輯指令進行的研究類似的研究：查看透過各管線階段的指令流，分析控制位元（根據SweRVref的第4部分，div指令有一個稱為div\_pkt\_t的特定結構類型，並且模組dec\_decode\_ctl中定義了一個名為div\_p的訊號）等。

為了執行這條指令，SweRV EH1處理器在模組exu\_div\_ctl（*[RVfpgaPath]/RVfpga/src/SweRVolfSoC/SweRVEh1CoreComplex/exu/exu\_div\_ctl.sv*）中實作了一個非管線式阻塞多週期除法單元。此單元最多需要34個週期來計算結果，但根據輸入的情況，也可能遠小於此值。此除法單元向處理器輸出幾個訊號（div\_stall、finish\_early和finish）來指示除法指令的狀態。

**任務：**檢查來自exu\_div\_ctl的Verilog程式碼，瞭解如何計算除法。此外，還要分析訊號div\_stall、finish\_early和finish的影響。作為一項選用練習，可以用自己的除法單元或網際網路上的除法單元來替換此除法單元。

圖8中的範例執行兩條div指令，這兩條指令包含在重複0xFFFF次迭代（即十進位65,535）的迴圈中。div指令在圖中以紅色強調顯示。與許多其他範例相反，在本例中，nop並非必要，因為div指令已因除法單元的高延遲而與所有其他指令隔離。與我們之前使用的範例程式一樣，此程式不執行任何有用的操作。

```
.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
```

圖8. 兩條連續div指令的範例

資料夾[RVfpgaPath]/RVfpga/Labs/Lab14/DIV\_Instruction提供PlatformIO專案，以便可以根據需要分析、模擬和變更程式。此專案的結構與mul指令使用的結構一樣，基於實驗11中供使用效能計數器的結構。

如果在PlatformIO中開啟、編譯專案，然後開啟反組譯檔案（位於[RVfpgaPath]/RVfpga/Labs/Lab14/DIV\_Instruction/.pio/build/swervolf\_nexys/firmware.dis中），則會發現div指令位於位址0x000001c0和0x000001c4處。

0x000001c0:	03de42b3	div	t0,t3,t4
0x000001c4:	03ff4333	div	t1,t5,t6

**任務：**驗證這對32位元值（0x03de42b3和0x03ff4333）是否對應指令RISC-V架構中的指令div t0,t3,t4和div t1,t5,t6。

圖9顯示了迴圈隨機迭代期間圖8中程式的模擬情況。



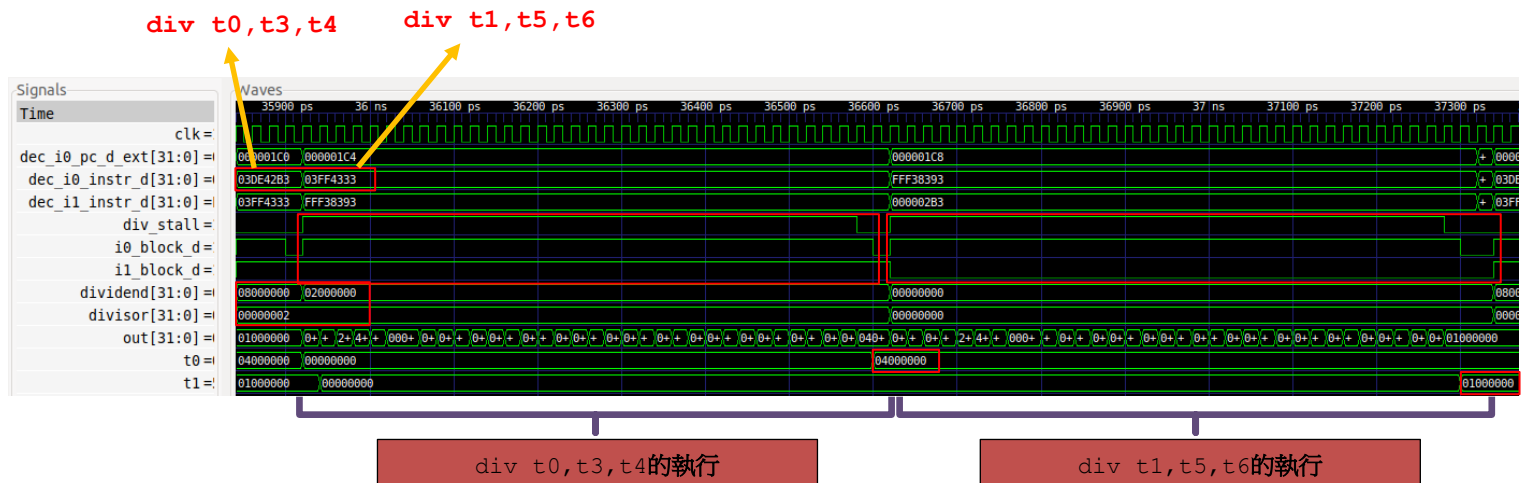


圖9. 圖8中範例的Verilator模擬

**任務：**在自己的電腦上重複圖9中的模擬過程，以進行詳細的分析。

分析圖9中的波形。以紅色強調顯示的值是與遍歷管線的兩條div指令相關的訊號。

- 兩條div指令在同一週期到達解碼階段（dec\_i0\_pc\_d\_ext = 0x0000001c0，即第一條div的指令位址）。第一條div指令（0x03de42b3）調度為在除法單元中執行，因此將被除數和除數（被除數 = 0x08000000，除數 = 0x00000002）傳送到此單元。請注意，我們為被除數選擇了較大的值，以使除法計算時間接近最大值（34個週期）。

鑒於SweRV EH1中的除法是阻塞的，div後的任何其他指令都將暫停。但是請注意，即使除法是非阻塞的，由於只有一個除數而造成的結構冒險也會使第二條div指令暫停。如第2部分中所述，還有其他方法可以提高效能，例如將除法器管線化或包含另一個除法器。不過，考慮到除法不是頻繁操作，這種情況下主要考慮降低硬體成本。

- 管線在第一條div指令的執行期間暫停（參見第一次除法計算期間的訊號div\_stall = 1）。還可以看到，當訊號i0\_block\_d和i1\_block\_d為1時阻塞，通路0和通路1被阻塞。此外，現在dec\_i0\_pc\_d\_ext = 0x0000001c4，這是在解碼階段暫停的第二條除法指令的位址。
- 除法單元的訊號out在34個週期後提供結果，此結果寫入目標暫存器（t0 = 0x04000000）。可以看到隨著除法運算逐步得出最終結果，每個週期的輸出值如何變化。
- 得到結果後，除數被釋放，管線繼續執行（div\_stall = 0），第二條div指令調度給除法單元。隨後，在34個週期後，第二條div指令的結果寫入暫存器檔案（t1 = 0x01000000）。

與第一條div指令一樣，由於阻塞除數的原因，第二條div指令之後的所有指令必須暫停。不過，在這種情況下，不依賴於t1的指令可以繼續（如果是非阻塞除法）。

圖10所示為圖8的範例中通過SweRV EH1管線的指令流。當第一條div指令進行解碼時（週期i），由於SweRV EH1阻塞除法和除法單元中結構冒險的原因，第二條div指令和後續指令在其目前階段暫停。（暫停的指令在圖中用字尾-st標記。）之後，在34個週期後（週期i+34），第一條div指令完成執行，並透過2:1多路開關將結果寫回到暫存器檔案，如實驗11的圖4所示。在接下來的週期中（i+35），後續指令恢復。然後，在週期36中，第二條div指令開始執行，由於SweRV EH1阻塞除法的原因，後續指令再次暫停。

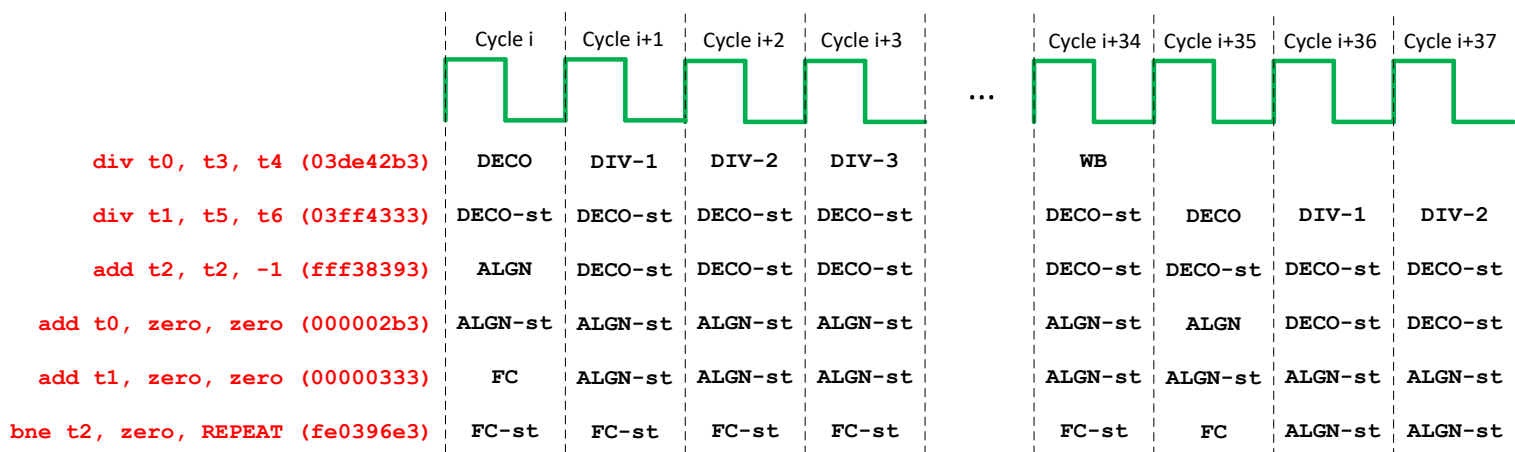


圖10. 圖8範例程式碼的執行（-st字尾表示暫停指令）

**任務：**將圖10中的說明與已複製到電腦上的圖9中的模擬進行比較。根據需要新增訊號以擴展模擬並加深理解。

**任務：**使用SweRV EH1中提供的效能計數器測量不同的事件（週期、已提交的指令/除法），如實驗11中所述。  
在分析圖9中的模擬後，週期數是否符合預期？證明您的答案。

**任務：**嘗試不同的被除數和除數，瞭解用於計算結果的週期數與其值的依賴關係。透過模擬和硬體計數器查看實驗。

**任務：**資料夾[RVfpgaPath]/RVfpga/Labs/Lab14/DIV\_Instr\_Accumul\_C-Lang提供C程式的PlatformIO專案，此C程式用於累加迴圈中兩次除法的減法結果。

- 分析C程式。
- 執行模擬並檢查迴圈的隨機迭代。請注意，C程式在未經過最佳化的情況下編譯。
- 使用SweRV EH1中提供的效能計數器測量不同的事件（週期、已提交的指令/除法等），如實驗11中所述。  
在分析圖9中的模擬後，週期數是否符合預期？證明您的答案。
- 用RISC-V組合語言建立一個相似的程式並將其與C版本進行比較。
- 停用C程式中的M RISC-V延伸功能並將結果與原始程式進行比較。為此，請將platformio.ini中的以下行：  

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

 修改為：  

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

 這樣便可避免使用RISC-V M延伸功能中的指令，而是使用其他指令進行模擬。

**任務：**在SweRV EH1中，div指令是阻塞的。修改處理器以允許非阻塞div指令。

然後向SweRV EH1處理器新增第二個除法器，以便圖8範例中的兩條div指令可以平行執行。