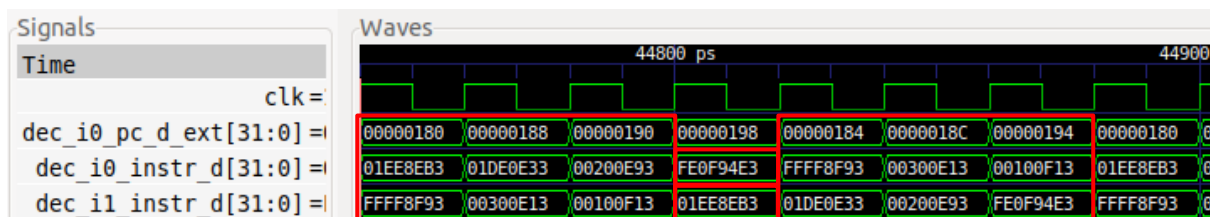


任務

任務：在自己的電腦上重複圖5中的模擬過程。可以使用以下位置提供的.tcl檔案：
[RVfpgaPath]/RVfpga/Labs/Lab15/DataHazards_AL-AL/test_Basic.tcl。

解答請參見實驗15的主文件。

任務：刪除圖2範例中的所有nop指令。對於迴圈的兩次連續迭代，繪製與圖3類似的圖，然後透過將其與Verilator模擬進行比較來分析並確認此圖是否正確，最後在板上執行程式時使用效能計數器計算IPC。



每次迭代需要3.5個週期來執行並且沒有暫停。

```
src > C Test.c x Test_Assembly.S startup.S
27
28 Test_Assembly();
29
30 cyc_end = pspPerformanceCounterGet(D_PSP_COUNTER0);
31 instr_end = pspPerformanceCounterGet(D_PSP_COUNTER1);
32
33 printfNexys("Cycles = %d", cyc_end-cyc_beg);
34 printfNexys("Instructions = %d", instr_end-instr_beg);
35
36 while(1);
37
```

```
> Executing task: platformio device monitor <
--- Available filters and text transformations: colorize, debug, default, direct, hexlify, log2fi
--- More details at http://bit.ly/pio-monitor-filters
--- Miniterm on /dev/ttyUSB1 115200,8,N,1 ---
--- Quit: Ctrl+C | Menu: Ctrl+T | Help: Ctrl+T followed by Ctrl+H ---
Cycles = 229619
Instructions = 458787
```

IPC是理想IPC： $IPC = 458 / 229 = 2$ 。

任務：在圖2的範例中，刪除所有nop指令並將add t6,t6,-1指令移至add t3,t3,t4指令之後，然後重新檢查模擬中以及板上的程式。在這個調整順序的程式中，兩條相關add指令（add t4,t4,t5和add t3,t3,t4）在同一週期到達解碼階段，這將影響效能。透過模擬和板上執行來說明這些變化的影響。

測試將相關add指令替換為其他相關指令時的類似情況，例如：

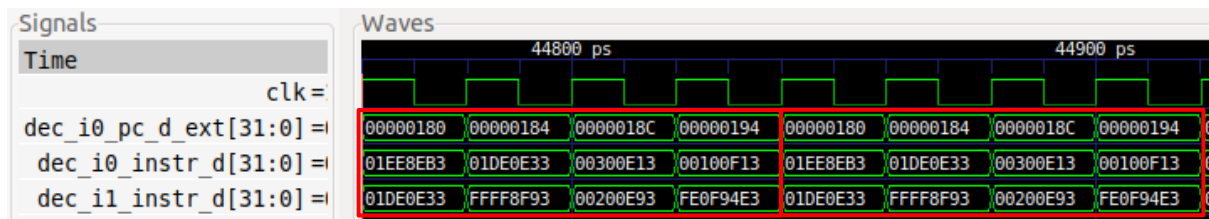
- add t4,t4,t5

```

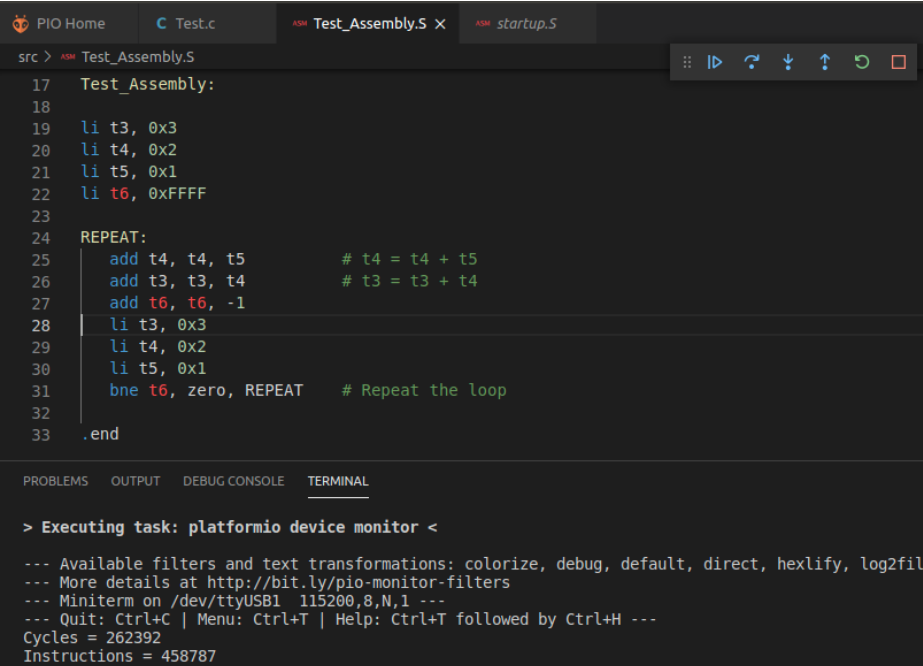
mul t3, t3, t4
-   add t4, t4, t5
    div t3, t3, t4
-   add t4, t4, t5
    lw  t3, 0(t4)

```

- 兩條add指令：



現在每次迭代需要4個週期來執行，因為相關add指令必須暫停1個週期，原因在於其輸入運算元之一在第一條add指令於EX1階段執行並轉送結果之前不可用。



```

PIO Home  C Test.c  ASM Test_Assembly.S  startup.S
src > ASM Test_Assembly.S
17 Test_Assembly:
18
19 li t3, 0x3
20 li t4, 0x2
21 li t5, 0x1
22 li t6, 0xFFFF
23
24 REPEAT:
25     add t4, t4, t5      # t4 = t4 + t5
26     add t3, t3, t4      # t3 = t3 + t4
27     add t6, t6, -1
28     li t3, 0x3
29     li t4, 0x2
30     li t5, 0x1
31     bne t6, zero, REPEAT # Repeat the loop
32
33 .end

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

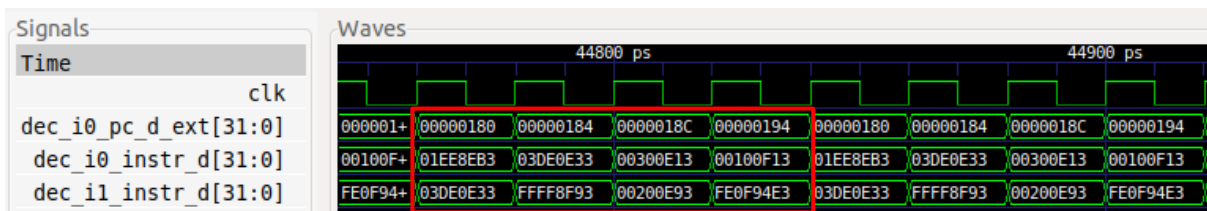
> Executing task: platformio device monitor <

--- Available filters and text transformations: colorize, debug, default, direct, hexlify, log2fil
--- More details at http://bit.ly/pio-monitor-filters
--- Miniterm on /dev/ttyUSB1 115200,8,N,1 ---
--- Quit: Ctrl+C | Menu: Ctrl+T | Help: Ctrl+T followed by Ctrl+H ---
Cycles = 262392
Instructions = 458787

```

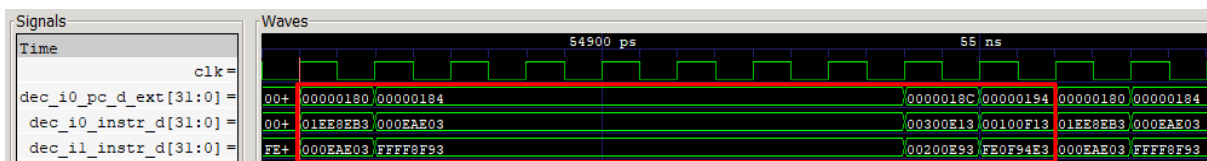
現在的IPC並非理想IPC：IPC = 458 / 262 = 1.75

- add指令後跟mul指令：



像前文一樣，相關mul指令必須暫停1個週期，原因在於其輸入運算元之一在第一條add指令於EX1階段執行並轉送結果之前不可用。

- add指令後跟lw指令：



像前文一樣，相關lw指令必須暫停1個週期，原因在於其輸入運算元之一在第一條add指令於EX1階段執行並轉發結果之前不可用。

任務：將前面的公式與DDCARV中管線處理器部分所述的公式進行比較。

DDCARV中管線處理器的公式：

```
if      ((Rs1E == RdM) & RegWriteM) & (Rs1E != 0) then // Forward from Memory stage
    ForwardAE = 10
else if ((Rs1E == RdW) & RegWriteW) & (Rs1E != 0) then // Forward from Writeback stage
    ForwardAE = 01
else      ForwardAE = 00                                // No forwarding (use RF output)
```

任務：分析Verilog程式碼，說明前一個公式如何執行計算。必須檢查模組dec_decode_ctl的以下行。

不提供解答。

任務：寫出i0_rs2bypass[9:0]、i0_rs1bypass[9:0]、i1_rs2bypass[9:0]和i1_rs1bypass[9:0]的其他控制位元的公式（與上述公式類似）。

可以從模組dec_decode_ctl中的以下幾行取得公式：

- 2372 – 2417
- 1721 – 1767
- 1497 – 1544

- 1130 – 1131和1255 – 1256

任務：在自己的電腦上重複圖8中的模擬過程。可以使用以下位置提供的.tcl檔案：
[RVfpgaPath]/RVfpga/Labs/Lab15/DataHazards_AL-AL/test_Advanced.tcl。

解答請參見實驗15的主文件。

任務：對於圖2中的程序，針對兩條相互依賴的指令彼此之間距離不同的情況執行與圖8中相同的分析。可以透過變更兩條相關add指令之間的nop數量來控制距離。

此外，需建立第一個輸入運算元接收轉送資料的其他範例。

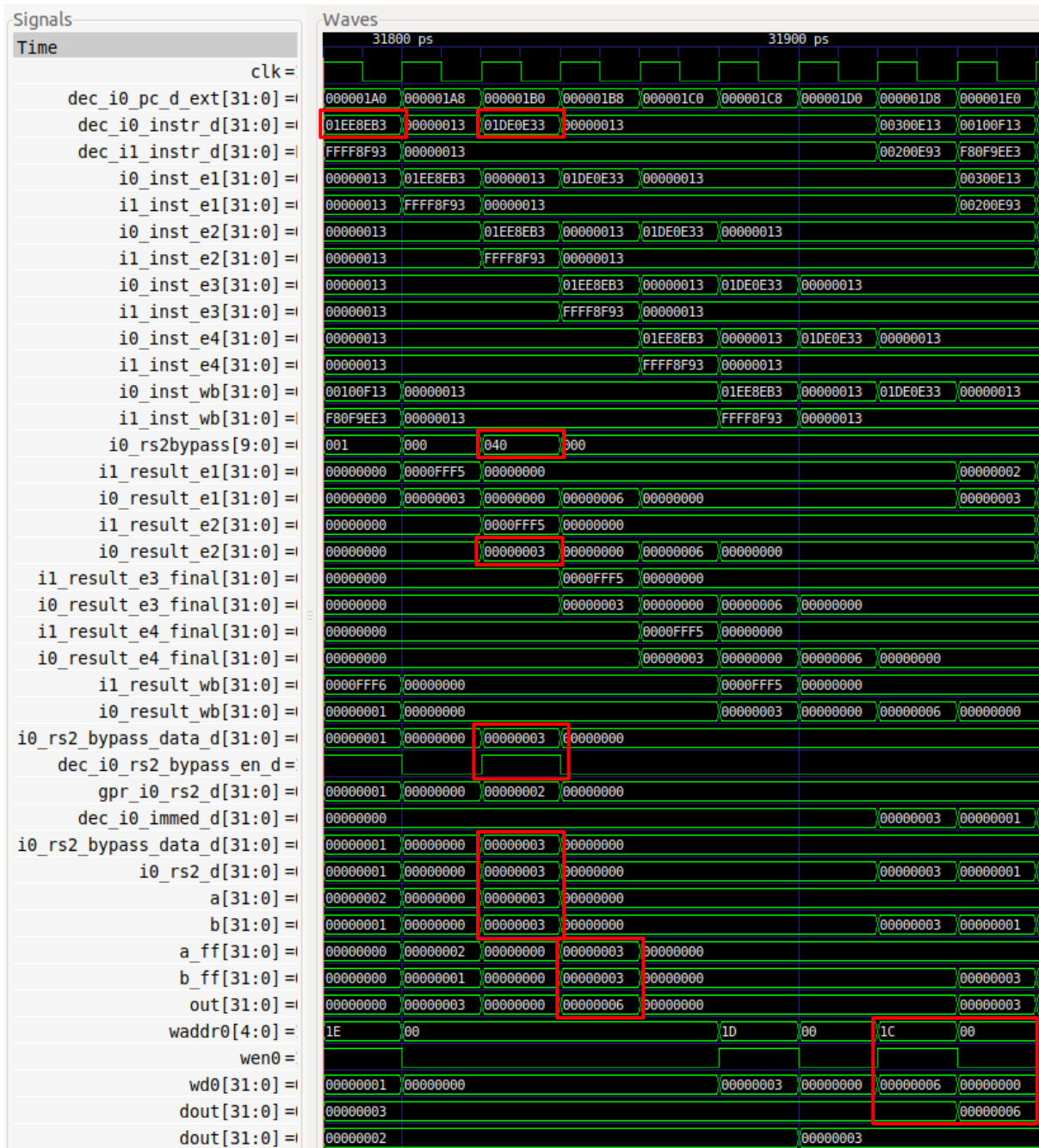
還可以建立兩條add指令透過l1管道執行的其他範例，但需確認行為相同。

最後，將相關add指令（add t3,t3,t4）替換為透過其他管道執行的其他相關指令並分析模擬結果。例如，可以包含以下指令之一來代替第二條add指令：

- lw t3, (t4)（強制讀取值來自DCCM，如實驗13所述）
- mul t3, t3, t4
- div t3, t3, t4

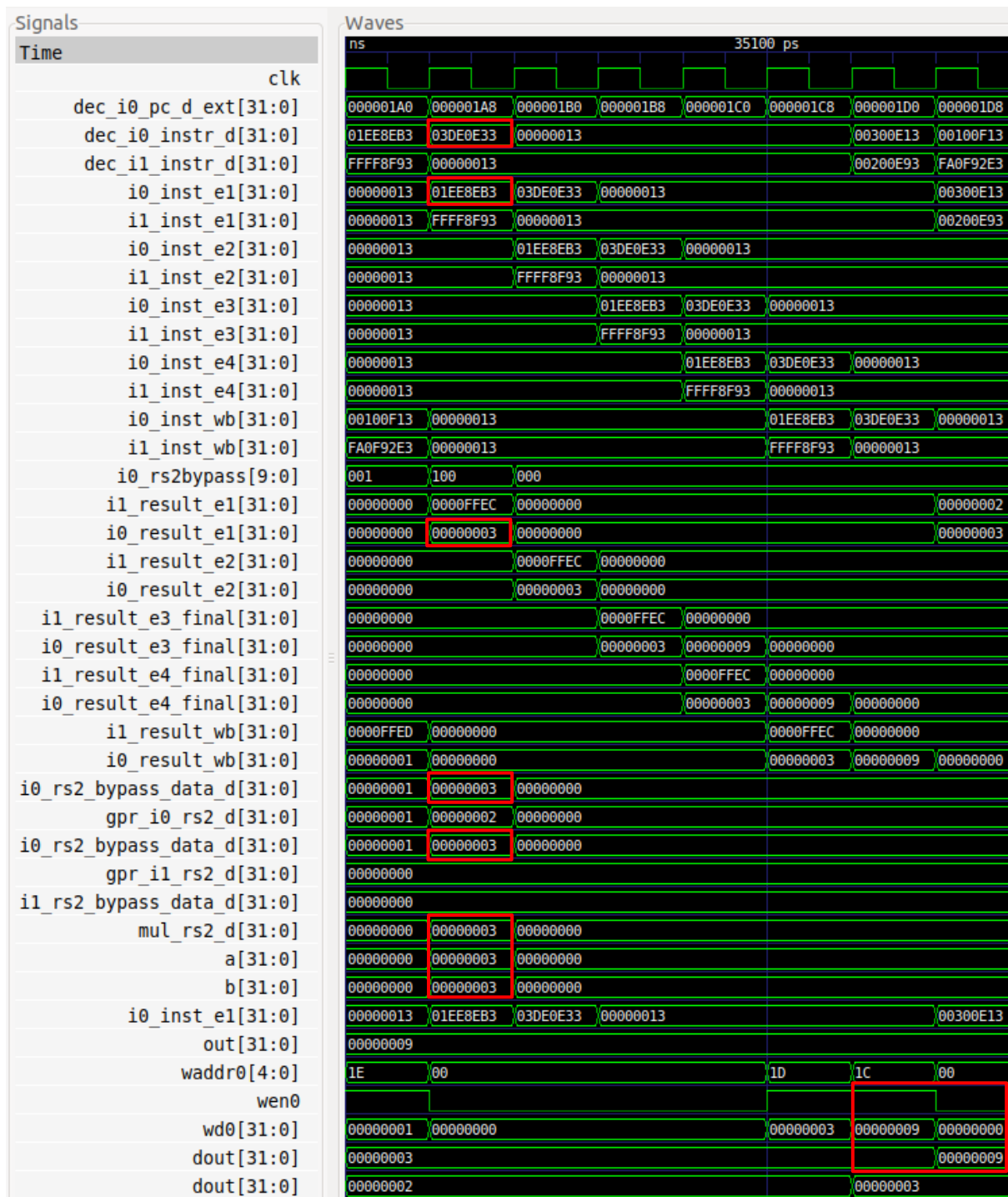
新模擬程式的範例：從EX2階段旁路到解碼階段：

1a0: 01ee8eb3	add t4,t4,t5
1a4: ffff8f93	addi t6,t6,-1
1a8: 00000013	nop
1ac: 00000013	nop
1b0: 01de0e33	add t3,t3,t4
1b4: 00000013	nop



新模擬程式的範例：執行mul指令而不是第二條add指令：

```
1a0: 01ee8eb3      add    t4, t4, t5
1a4: ffff8f93      addi   t6, t6, -1
1a8: 03de0e33      mul    t3, t3, t4
```



任務：向圖10新增相應邏輯以產生I0管道中的輔助ALU的第一個輸入運算元（a）。

不提供解答。

任務：在自己的電腦上重複圖12中的模擬過程。可以使用以下位置提供的.tcl檔案：
`[RVfpgaPath]/RVfpga/Labs/Lab15/DataHazards_Close-LW-AL/scriptLoad.tcl`

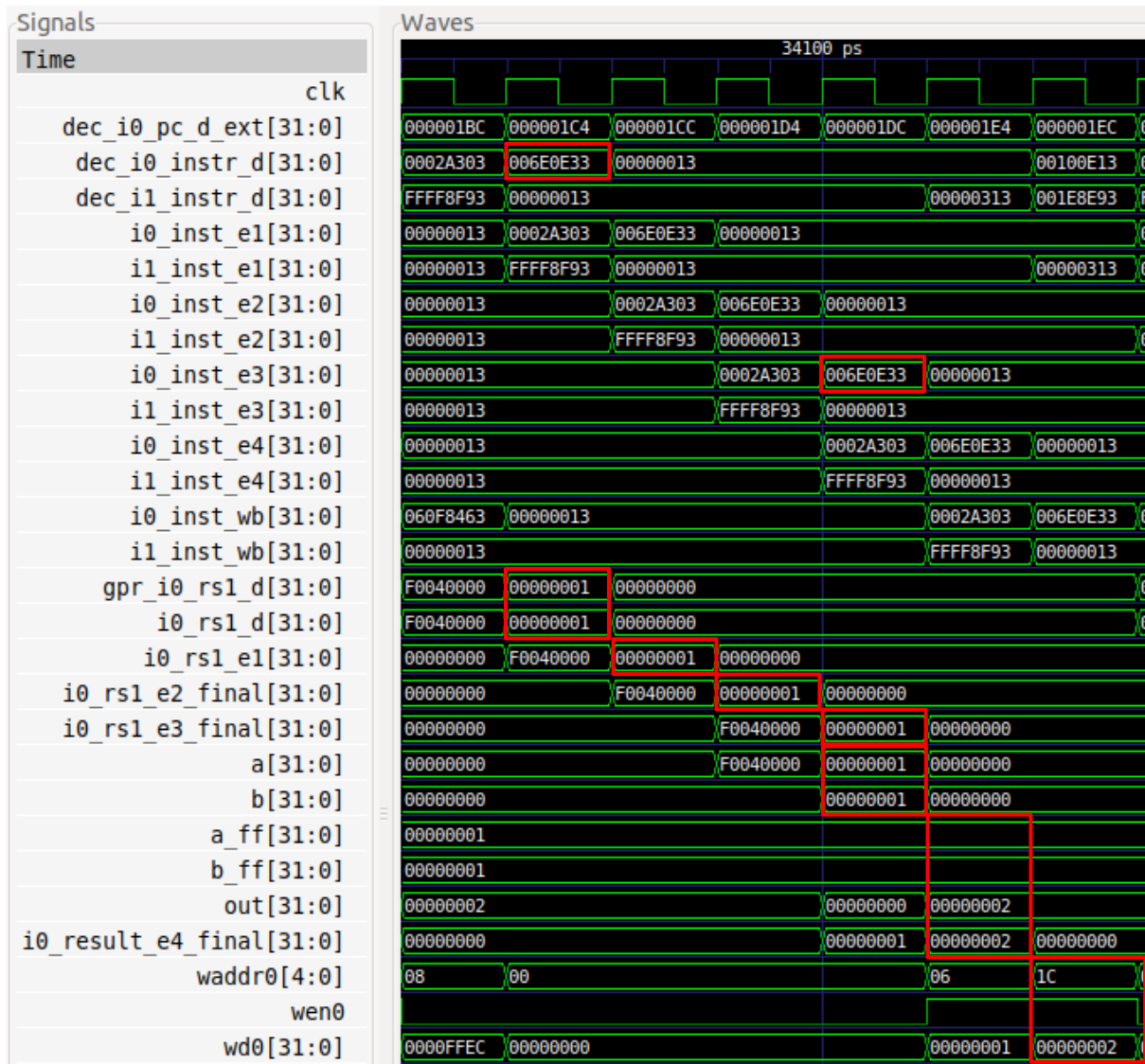
解答請參見實驗15的主文件。

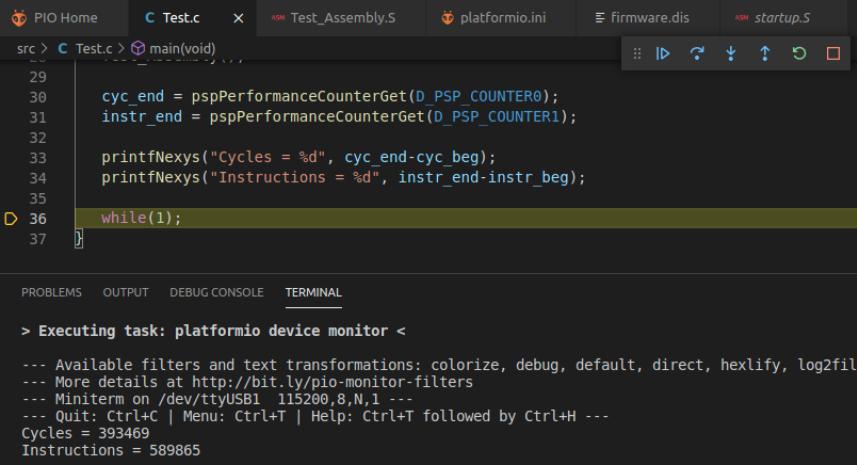
任務：為圖11的範例繪製與圖3類似的圖。

不提供解答。

任務：在前一個範例中，分析如何取得add t3, t3, t1指令的第一個運算元（t3）。可以使用以下位置提供的.tcl檔案：`[RVfpgaPath]/RVfpga/Labs/Lab15/DataHazards_Close-LW-AL/scriptLoad_FirstOperand.tcl`

第一個運算元與先前指令不相關，因此它直接從暫存器檔案中取得。





```

src > C Test.c x Test_Assembly.S platformio.ini firmware.dis startup.S
29
30     cyc_end = pspPerformanceCounterGet(D_PSP_COUNTER0);
31     instr_end = pspPerformanceCounterGet(D_PSP_COUNTER1);
32
33     printfNexys("Cycles = %d", cyc_end-cyc_beg);
34     printfNexys("Instructions = %d", instr_end-instr_beg);
35
36     while(1);
37
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
> Executing task: platformio device monitor <
--- Available filters and text transformations: colorize, debug, default, direct, hexlify, log2fil
--- More details at http://bit.ly/pio-monitor-filters
--- Miniterm on /dev/ttyUSB1 115200,8,N,1 ---
--- Quit: Ctrl+C | Menu: Ctrl+T | Help: Ctrl+T followed by Ctrl+H ---
Cycles = 393469
Instructions = 589865

```

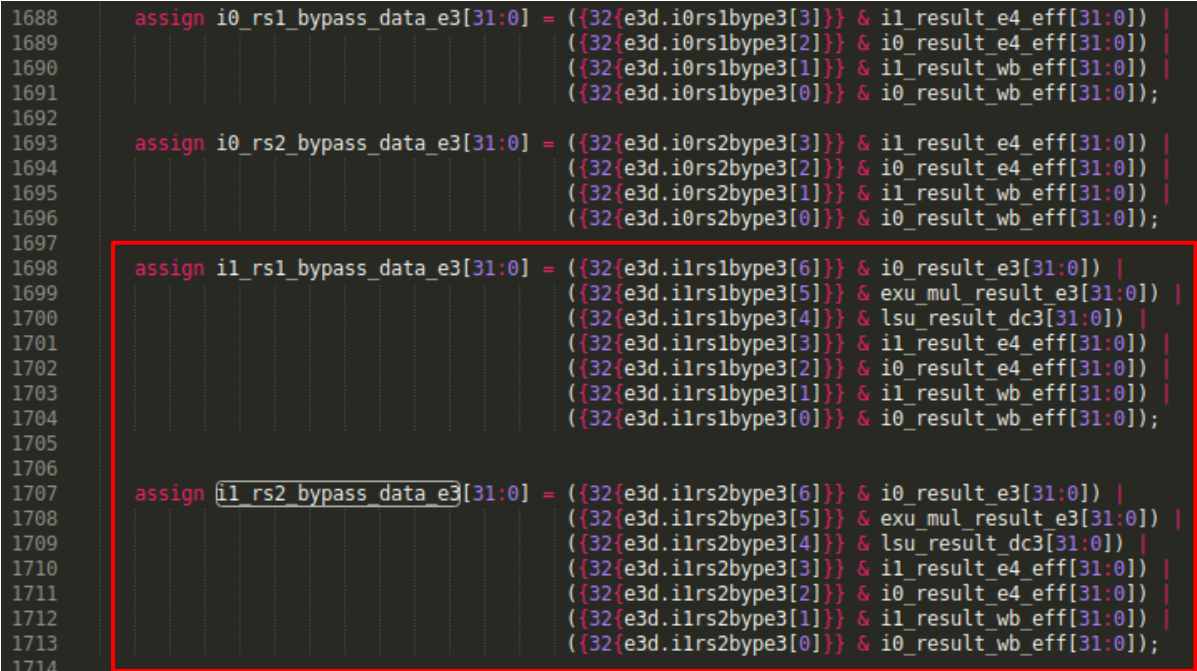
受lw-add資料冒險引起的暫停影響，IPC現與理想IPC相差很大： $IPC = 5898 / 3934 = 1.499$

任務：在圖11的範例中，將add t6,t6,-1指令移至add t3,t3,t1指令之後，然後重新檢查模擬中以及板上的程式。

可以使用以下位置提供的程式：

[RVfpgaPath]/RVfpga/Labs/RVfpgaLabsSolutions/Programs_Solutions/Lab15/DataHazards_SameCycle-LW-AL

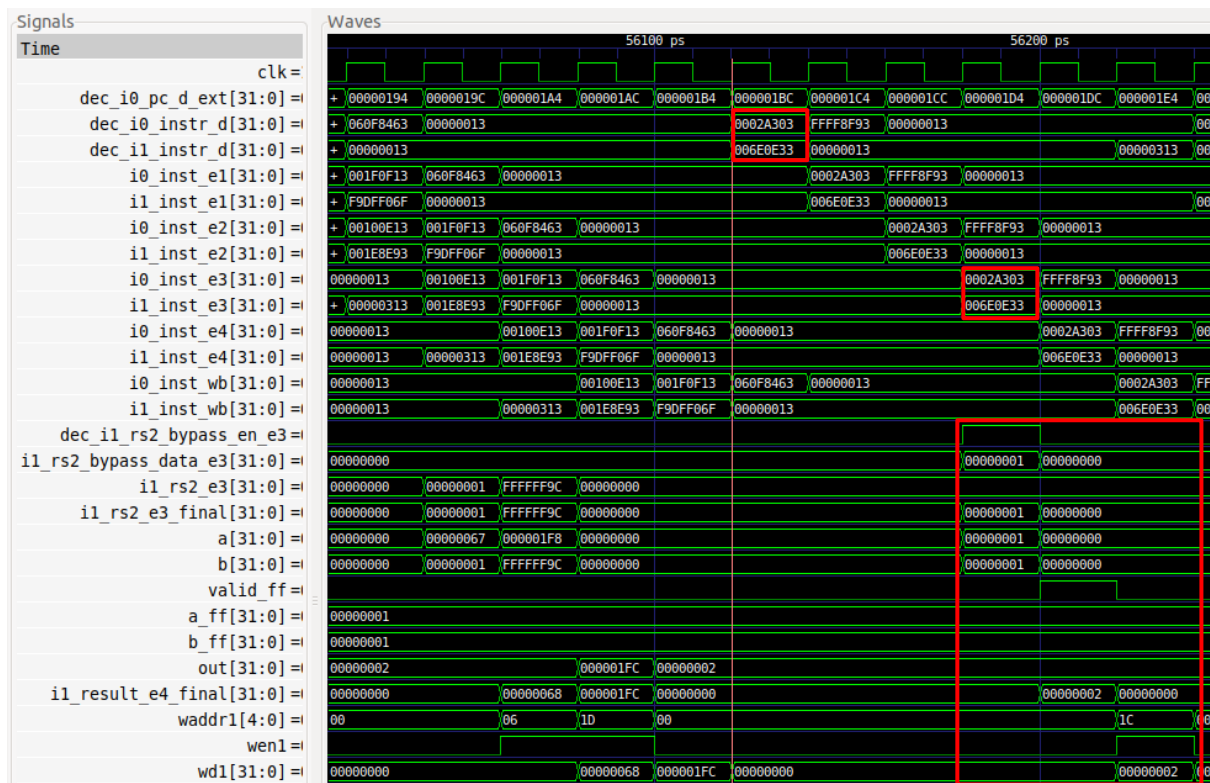
add t3,t3,t1指令與載入指令相關，兩者均透過兩個通路平行執行。在這種情況下，add指令會在輔助ALU中重新執行。請注意，通路1可以從其他管道接收輸入運算元，因此在這種情況下不會遺失任何週期。



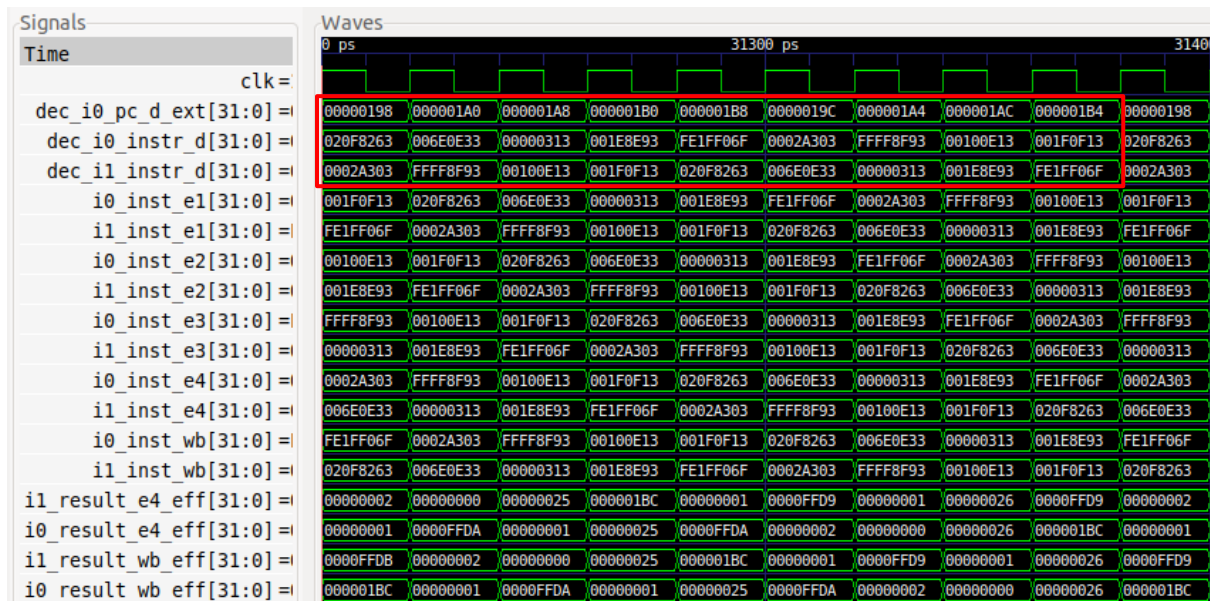
```

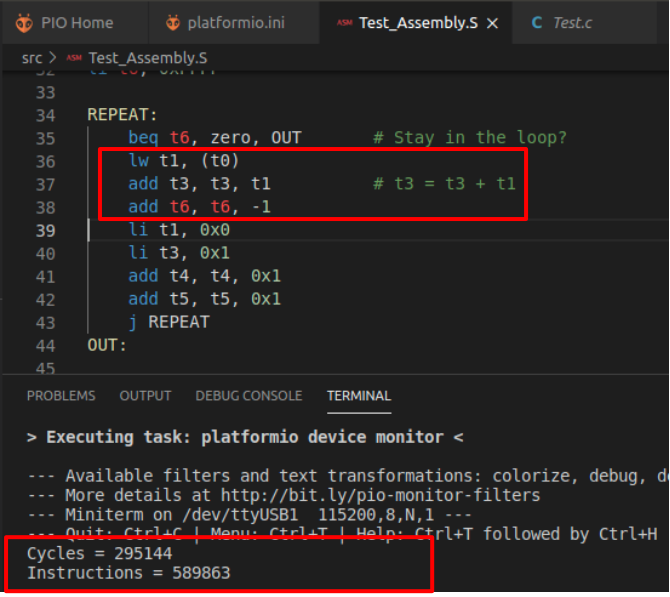
1688     assign i0_rs1_bypass_data_e3[31:0] = ({32{e3d.i0rs1bype3[3]}} & i0_result_e4_eff[31:0]) |
1689                                           ({32{e3d.i0rs1bype3[2]}} & i0_result_e4_eff[31:0]) |
1690                                           ({32{e3d.i0rs1bype3[1]}} & i1_result_wb_eff[31:0]) |
1691                                           ({32{e3d.i0rs1bype3[0]}} & i0_result_wb_eff[31:0]);
1692
1693     assign i0_rs2_bypass_data_e3[31:0] = ({32{e3d.i0rs2bype3[3]}} & i1_result_e4_eff[31:0]) |
1694                                           ({32{e3d.i0rs2bype3[2]}} & i0_result_e4_eff[31:0]) |
1695                                           ({32{e3d.i0rs2bype3[1]}} & i1_result_wb_eff[31:0]) |
1696                                           ({32{e3d.i0rs2bype3[0]}} & i0_result_wb_eff[31:0]);
1697
1698     assign i1_rs1_bypass_data_e3[31:0] = ({32{e3d.i1rs1bype3[6]}} & i0_result_e3[31:0]) |
1699                                           ({32{e3d.i1rs1bype3[5]}} & exu_mul_result_e3[31:0]) |
1700                                           ({32{e3d.i1rs1bype3[4]}} & lsu_result_dc3[31:0]) |
1701                                           ({32{e3d.i1rs1bype3[3]}} & i1_result_e4_eff[31:0]) |
1702                                           ({32{e3d.i1rs1bype3[2]}} & i0_result_e4_eff[31:0]) |
1703                                           ({32{e3d.i1rs1bype3[1]}} & i1_result_wb_eff[31:0]) |
1704                                           ({32{e3d.i1rs1bype3[0]}} & i0_result_wb_eff[31:0]);
1705
1706
1707     assign i1_rs2_bypass_data_e3[31:0] = ({32{e3d.i1rs2bype3[6]}} & i0_result_e3[31:0]) |
1708                                           ({32{e3d.i1rs2bype3[5]}} & exu_mul_result_e3[31:0]) |
1709                                           ({32{e3d.i1rs2bype3[4]}} & lsu_result_dc3[31:0]) |
1710                                           ({32{e3d.i1rs2bype3[3]}} & i1_result_e4_eff[31:0]) |
1711                                           ({32{e3d.i1rs2bype3[2]}} & i0_result_e4_eff[31:0]) |
1712                                           ({32{e3d.i1rs2bype3[1]}} & i1_result_wb_eff[31:0]) |
1713                                           ({32{e3d.i1rs2bype3[0]}} & i0_result_wb_eff[31:0]);
1714

```



如果我們刪除nop指令，進行模擬並在板上執行，則：





```

src > Test_Assembly.S
32  t0, 0x1ff
33
34  REPEAT:
35      beq t6, zero, OUT      # Stay in the loop?
36      lw t1, (t0)
37      add t3, t3, t1          # t3 = t3 + t1
38      add t6, t6, -1
39      li t1, 0x0
40      li t3, 0x1
41      add t4, t4, 0x1
42      add t5, t5, 0x1
43      j REPEAT
44  OUT:
45
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL
> Executing task: platformio device monitor <
--- Available filters and text transformations: colorize, debug, d
--- More details at http://bit.ly/pio-monitor-filters
--- Miniterm on /dev/ttyUSB1 115200,8,N,1 ---
--- Quit: Ctrl+C | Menu: Ctrl+T | Help: Ctrl+T followed by Ctrl+H
Cycles = 295144
Instructions = 589863

```

由於轉送邏輯和輔助ALU的原因，沒有暫停，IPC是理想IPC： $IPC = 5898 / 2951 = 1.998$

1. 練習

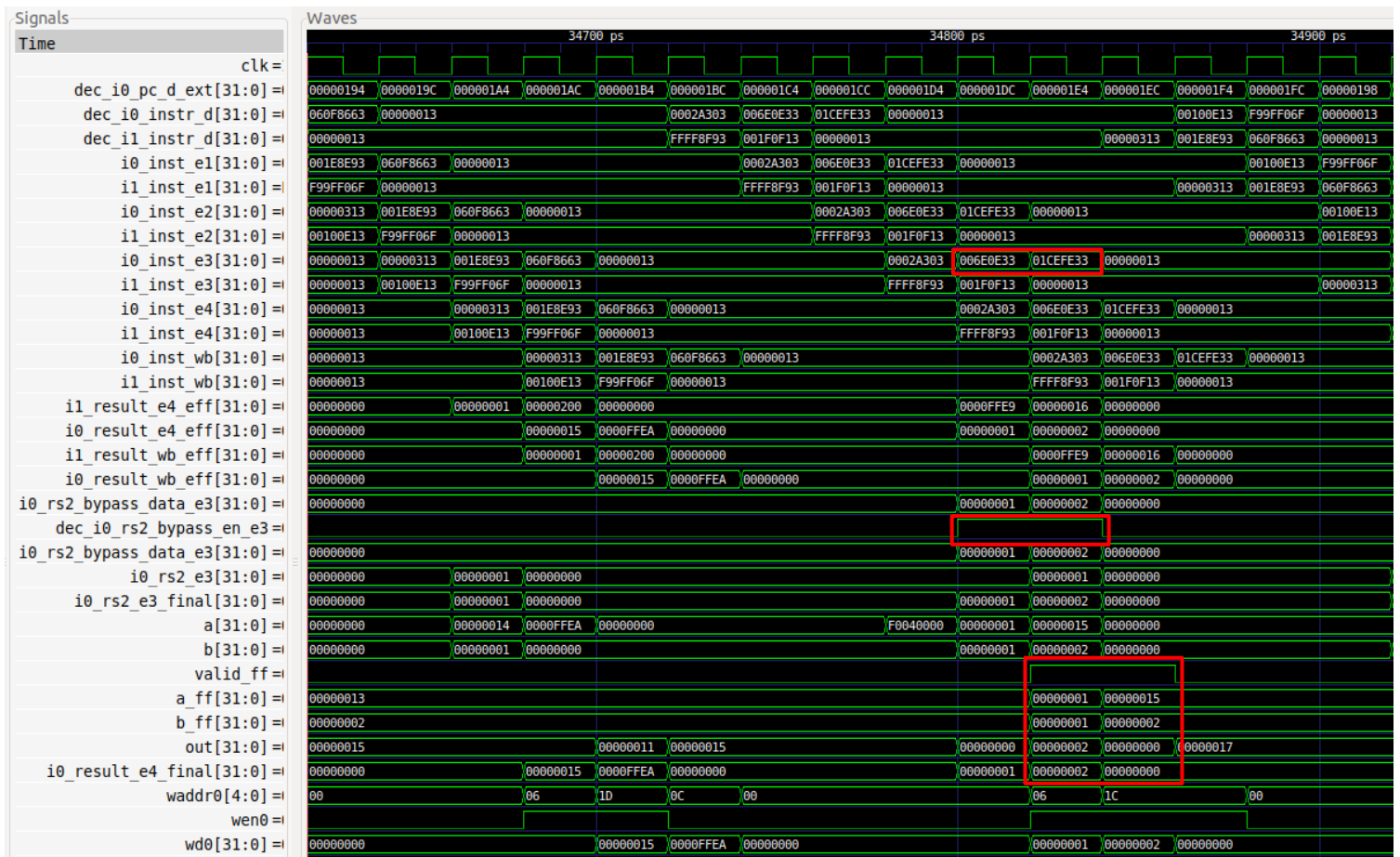
- 1) 透過新增與add指令結果相關的額外算術-邏輯指令，修改第3部分中使用的程式。例如，可以將圖11中的迴圈替換為以下程式碼，其中包含一條新的AND指令（**and t3, t4, t3**），並透過前移指令**add t5, t5, 0x1**對程式碼順序稍作調整：

```

REPEAT:
    beq t6, zero, OUT
    INSERT_NOPS_9
    lw t1, (t0)
    add t6, t6, -1
    add t3, t3, t1
    add t5, t5, 0x1
    and t3, t4, t3
    INSERT_NOPS_8
    li t1, 0x0
    li t3, 0x1
    add t4, t4, 0x1
    j REPEAT
OUT:

```

分析Verilator模擬並說明如何為新的A-L指令處理資料冒險。然後刪除所有nop指令並分析硬體計數器提供的結果。



相關add和and指令使用輔助ALU重新計算結果。請注意，and指令的第二個輸入運算元在EX3階段旁路。

```

src > C Test.c > main(void)
22  pspPerformanceCounterSet(D_PSP_COUNTER0, E_CYCLES_CLOCKS_ACTIVE);
23  pspPerformanceCounterSet(D_PSP_COUNTER1, E_INSTR_COMMITTED_ALL);
24
25  cyc_beg = pspPerformanceCounterGet(D_PSP_COUNTER0);
26  instr_beg = pspPerformanceCounterGet(D_PSP_COUNTER1);
27
28  Test_Assembly();
29
30  cyc_end = pspPerformanceCounterGet(D_PSP_COUNTER0);
31  instr_end = pspPerformanceCounterGet(D_PSP_COUNTER1);
32
33  printfNexys("Cycles = %d", cyc_end-cyc_beg);
34  printfNexys("Instructions = %d", instr_end-instr_beg);
35
36  while(1);
37

```

PROBLEMS OUTPUT DEBUG CONSOLE **TERMINAL**

```

> Executing task: platformio device monitor <

-- Available filters and text transformations: colorize, debug, default, direct, hexlify, log2fil
-- More details at http://bit.ly/pio-monitor-filters
-- Miniterm on /dev/ttyUSB1 115200,8,M,1 ---
-- Quit: Ctrl+C | Menu: Ctrl+T | Help: Ctrl+T followed by Ctrl+H ---
Cycles = 327910
Instructions = 655398

```

$$IPC = 6553 / 3279 = 1.998$$

$$\text{每次迭代執行的指令} : 655398 / 65535 = 10$$

每次迭代的週期數：327910 / 65535 = 5

由於轉發和輔助ALU的原因，在該程式中實現了理想IPC。

2) 分析與第2.C部分所述情況相同的情況：mul指令後跟使用乘法結果的add指令。在圖11的程式中，只需將lw替換為寫入暫存器t1的mul。

不提供解答。

3) 分析lw指令後跟與載入讀取的值相關的mul指令這種情況。在圖11的程式中，只需將相關add指令替換為mul指令。

可以使用以下位置提供的程式：

[RVfpgaPath]/RVfpga/Labs/RVfpgaLabsSolutions/Programs_Solutions/Lab15/DataHazards_Close-LW-MUL

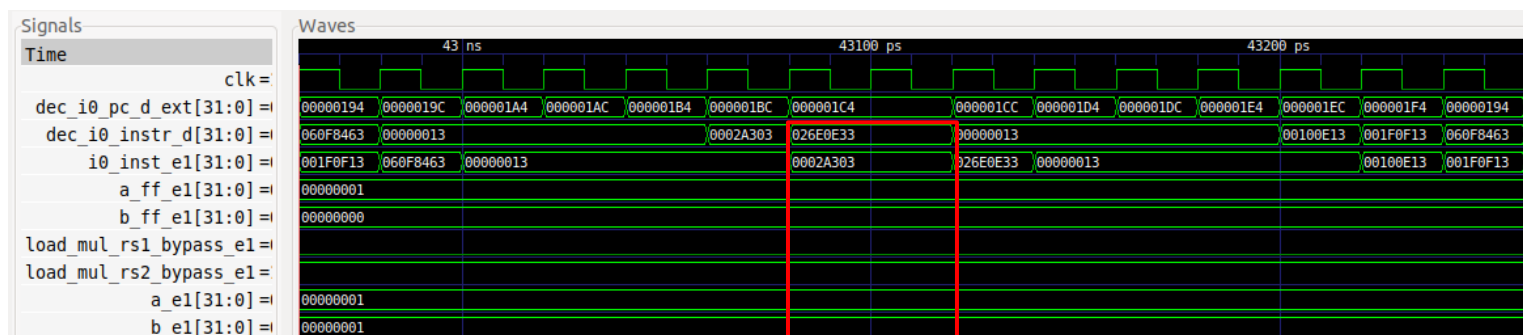
輔助ALU無法執行mul指令。乘法器（模組exu_mul_ctl）內部實作了一個新的旁路路徑，用於將載入讀取的值轉送到M1階段。

```

85 // ----- E1 Logic Stage -----
86
87 assign a_e1[31:0] = (load_mul_rs1_bypass_e1) ? lsu_result_dc3[31:0] : a_ff_e1[31:0];
88 assign b_e1[31:0] = (load_mul_rs2_bypass_e1) ? lsu_result_dc3[31:0] : b_ff_e1[31:0];

```

這樣一來，由於這種RAW相關性，只有1個週期遺失。



第二個運算元從載入讀取的值旁路。這樣一來，由於相關性的原因，只有1個週期遺失。

4) （以下練習基於[HePa]的練習4.18、4.19、4.20和4.26。）

假設在不處理資料冒險的SweRV EH1處理器版本上執行了以下程式碼（即，程式設計師負責透過在必要時插入nop指令來處理資料冒險）。向程式碼中新增nop指令以使其正確執行。

```

addi x11, x12, 5
add x13, x11, x12

```



```
addi x14, x11, 15
add x15, x13, x12
```

然後組成包含至少三個組合語言程式碼片段的序列，這些片段顯示不同類型的RAW資料冒險。RAW資料相關性的類型由產生結果的階段和使用結果的下一條指令標識。

對於每個序列，要使程式碼在沒有轉送或冒險偵測的SweRV EH1處理器上正確執行，需要在何處插入多少條nop？如果我們使用SweRV EH1中提供的轉送而不插入nop，那麼CPI是多少？

不提供解答。

5) 在實驗14第2.C部分的程式（位於 `[RVfpgaPath]/RVfpga/Labs/Lab14/LW_Instruction_ExtMemory`）中，將指令add **x1**, x1, 1替換為add **x28**, x1, 1。這將在修改後的add指令和迴圈開頭的非封鎖載入（lw x28, (x29)）之間引入WAW冒險。利用模擬分析SweRV EH1中如何處理此冒險，可以在暫存器檔案中查看訊號wen2的值。嘗試瞭解控制單元（模組dec）中如何計算此訊號。

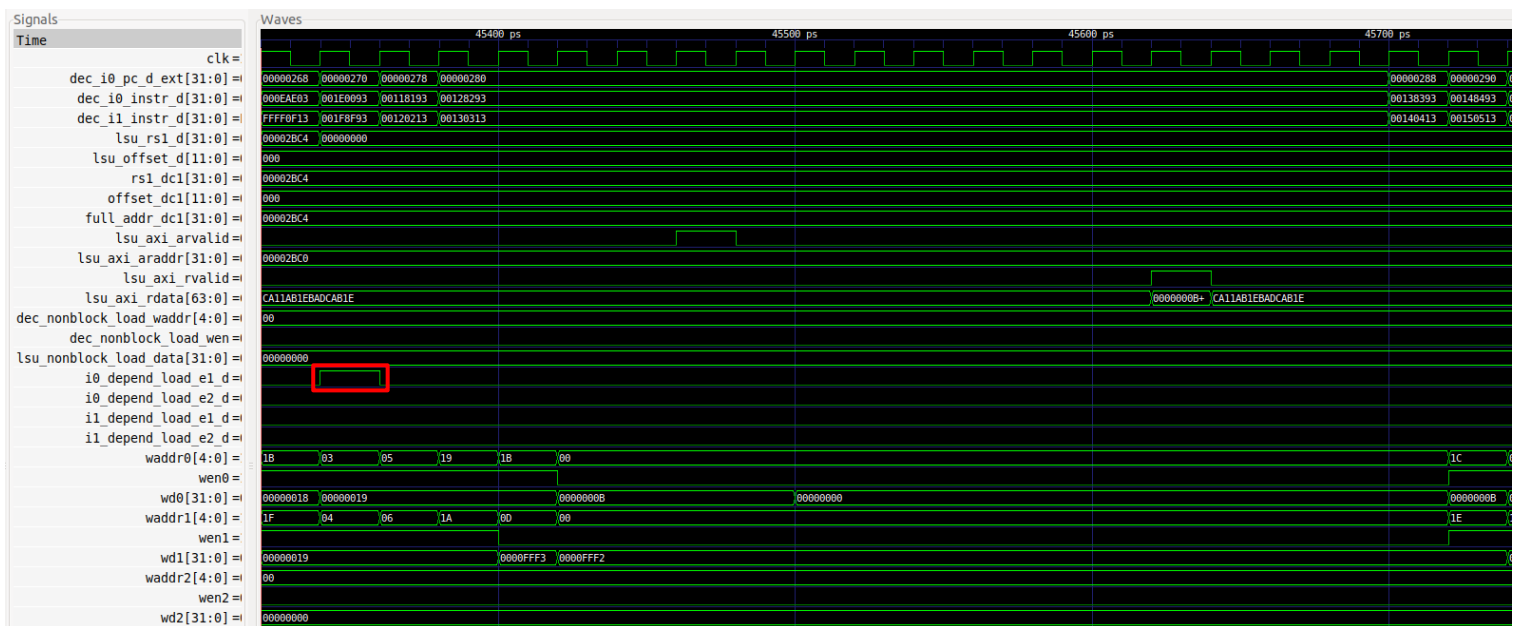
原始程式的模擬如下。正如我們在實驗14中分析的那樣，當對暫存器檔案同時執行3個寫入操作（2條add指令和1條非封鎖load指令）。



在如下所示的新程式中（將指令add **x1**, x1, 1替換為指令add **x28**, x1, 1），可以偵測到程式順序中的後面一條指令修改了同一個暫存器，因此停用了載入的寫入操作（wen2訊號永遠不會變為高電平），這樣便可解除WAW資料冒險。



6) 在實驗14第2.C部分的程式（位於 `[RVfpgaPath]/RVfpga/Labs/Lab14/LW_Instruction_ExtMemory`）中，將指令 `add x1, x1, 1` 替換為 `add x1, x28, 1`。這將在修改後的 `add` 指令和迴圈開頭的非封鎖載入（`lw x28, (x29)`）之間引入RAW冒險。透過模擬分析SweRV EH1中如何處理此冒險。



偵測到RAW冒險，管線暫停並按照實驗中的說明進行轉送。

7) 最後，在實驗14第2.C部分的程式（位於 `[RVfpgaPath]/RVfpga/Labs/Lab14/LW_Instruction_ExtMemory`）中，將指令 `add x1, x1, 1` 替換為 `add x1, x28, 1`，將指令 `add x7, x7, 1` 替換為 `add x28, x7, 1`。這將導致RAW和WAW冒險。利用模擬分析SweRV EH1中如何處理這兩種冒險。

不提供解答。

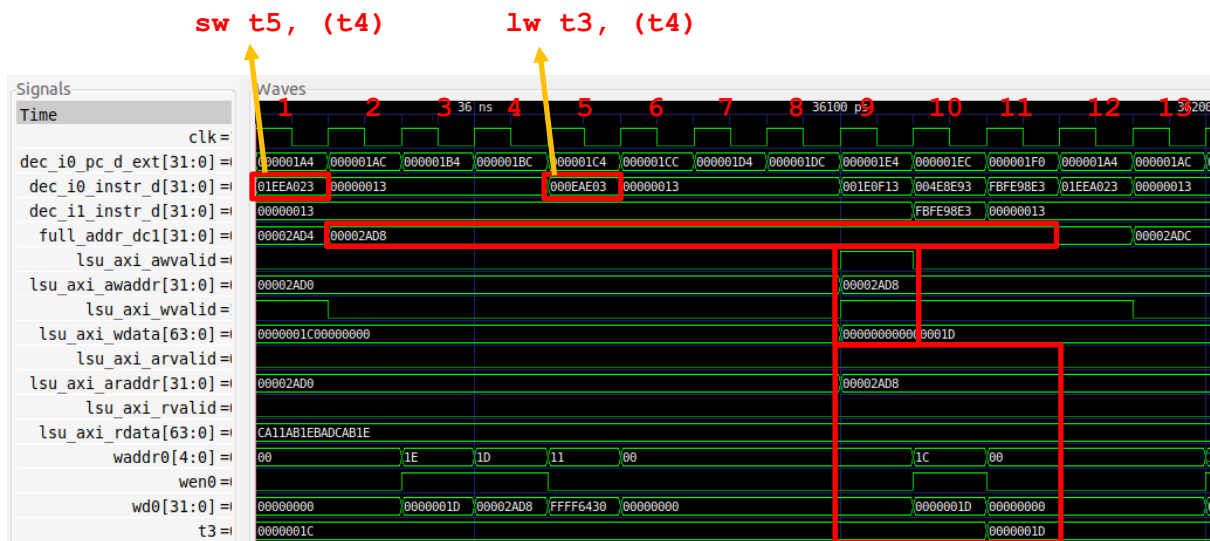
8) 儲存-載入轉送

這是一種非常值得關注的情況，我們沒有在本實驗中分析，但您可在本練習中進行分析。當儲存以及隨後的載入存取相同位址時，可以在核心內將資料從儲存轉送到載入，無需讀取DDR外部記憶體，進而節省時間和功耗。

實作這種轉送的邏輯包含在LSU中，具體來說是包含在模組lsu_bus_intf和lsu_bus_buffer中，必須在本練習中進行檢查。

[RVfpgaPath]/RVfpga/Labs/Lab15/Sw-Lw-Forwarding中的PlatformIO專案用於說明儲存-載入轉送。此資料夾中提供.tc指令碼，您可以使用它來分析迴圈的任意一次迭代以及瞭解如何執行轉送。

Verilator模擬：



分析模擬：

- **週期1：**sw指令進行解碼。
- **週期5：**lw指令進行解碼。
- **週期2至11：**在整個迭代過程中，訊號full_addr_dc1 = 0x00002AD8。發生這種情況是因為儲存位址和載入位址相同。
- **週期9：**儲存透過AXI匯流排的寫入訊號寫入DDR外部記憶體。

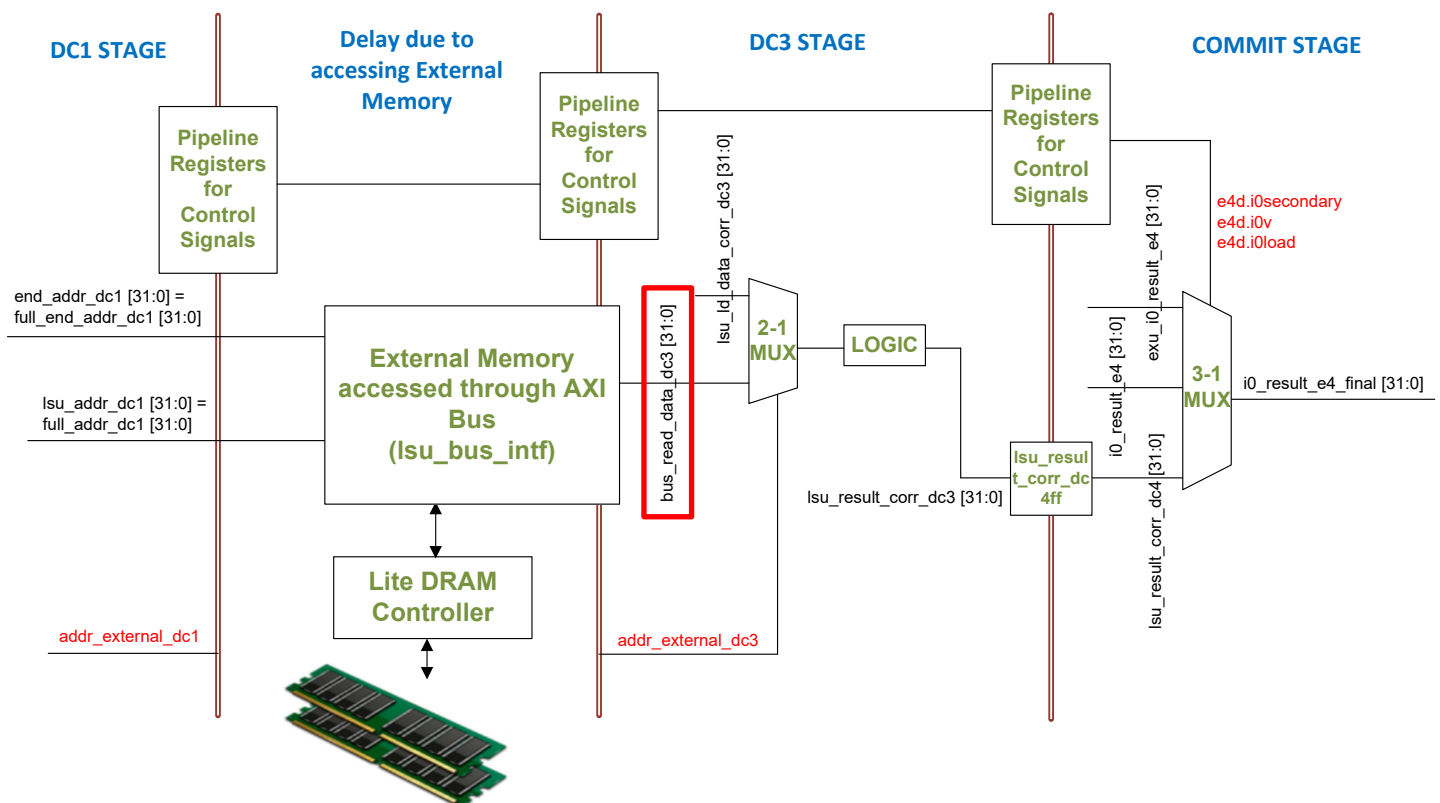
- o `lsu_axi_awvalid = 1`
- o `lsu_axi_awaddr = 0x00002AD8`
- o `lsu_axi_wvalid = 1`
- o `lsu_axi_wdata = 0x000000000000001D`

- **週期9、10和11：**載入透過旁路邏輯立即接收其資料並將資料寫入暫存器檔案。讀取永遠不會傳送到DDR記憶體（參見AXI匯流排的讀取啟用訊號：`lsu_axi_arvalid = lsu_axi_rvalid = 1`）：
 - o `waddr0 = 0x1C`（即暫存器x28 = t3）
 - o `wen0 = 1`
 - o `wd0 = 0x0000001D`
 - o `t3 = 0x0000001D`

核心內部如何執行轉送？

要分析儲存-載入轉送，必須檢查兩個模組：`lsu_bus_intf`和`lsu_bus_buffer`。

- 1) 我們在實驗13的第4部分中分析了對DDR外部記憶體的讀取存取，在實驗13的圖16中說明了此存取涉及的SweRV EH1結構：



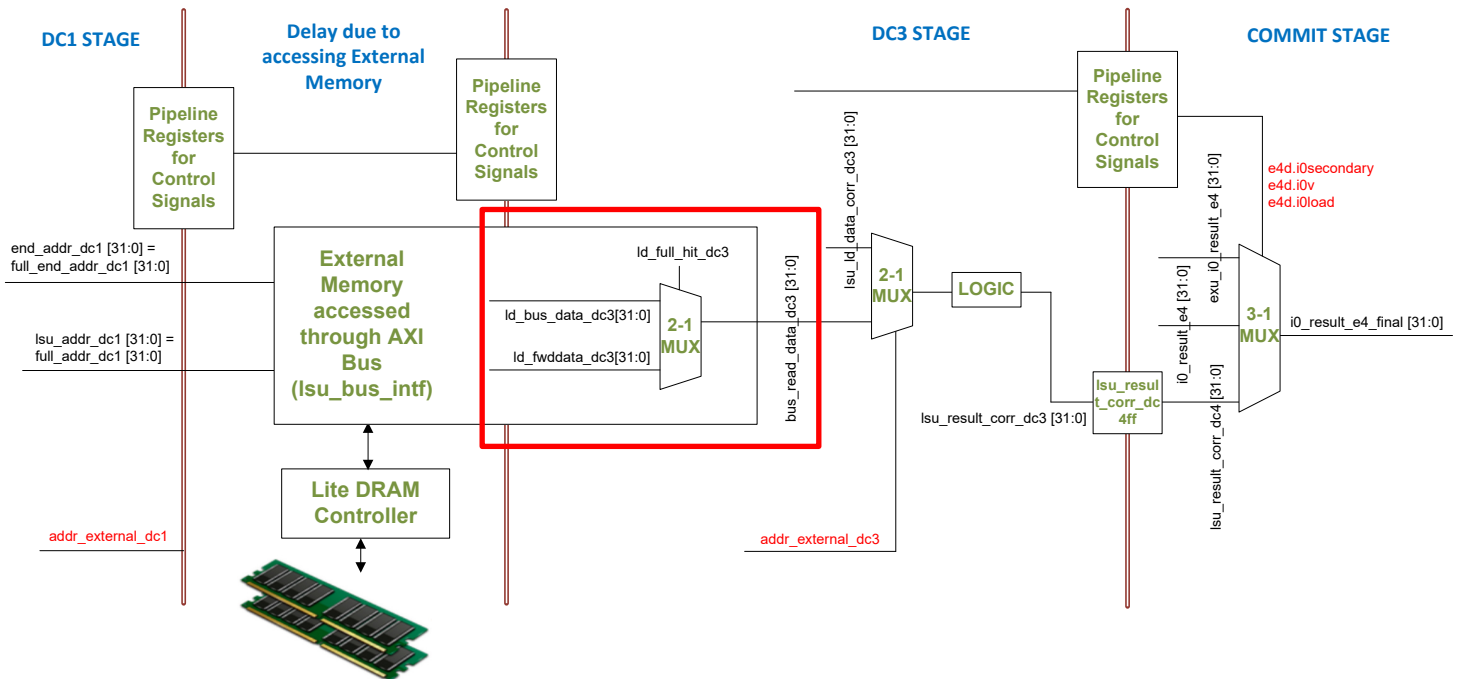
資料在訊號`bus_read_data_dc3`中提供。

- 2) 分配給`bus_read_data_dc3`的值可以來自DDR記憶體或轉送邏輯。為此，模組`lsu_bus_intf`中包含一個2:1多路開關，用於在從DDR記憶體讀取的值

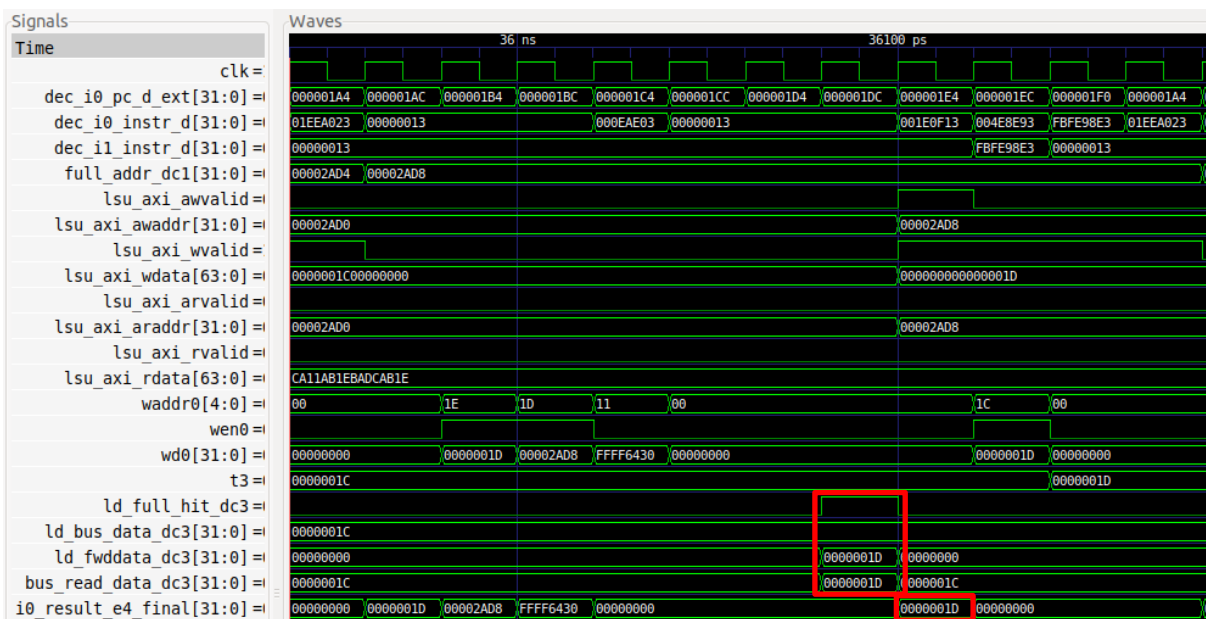
(ld_bus_data_dc3) 和旁路值 (ld_fwddata_dc3) 之間進行選擇。

```
387 assign bus_read_data_dc3[31:0] = ld_full_hit_dc3 ? ld_fwddata_dc3[31:0] : ld_bus_data_dc3[31:0];
```

接下來，我們將此2:1多路開關包含到上圖中：



3) 如果在之前的模擬中新增此多路開關的輸入/輸出/控制訊號，則：



可以看到提供給載入的資料從儲存旁路。

- 4) 可以進一步分析模組 **lsu_bus_intf** 和 **lsu_bus_buffer** 中如何計算控制訊號 (`ld_full_hit_dc3`) 和旁路值 (`ld_fwddata_dc3`)。

附錄A

任務：在自己的電腦上重複圖15中的模擬過程。

解答請參見實驗15的主文件。

任務：比較上述場景在 **SweRV EH1** 和 **DDCARV** 所述管線處理器中的處理方式。

不提供解答。

任務：如果仔細比較圖16和實驗13的圖6，將看到實驗13的圖6中 `lw` 指令讀入暫存器檔案的值（訊號 `lsu_ld_data_corr_dc3[31:0]`）與圖16中 `lw` 轉送的值（訊號 `lsu_ld_data_dc3[31:0]`）不同。兩個值的區別在於前者經過模組 **lsu_ecc** 中的 **ECC** 邏輯檢查，而後者沒有。說明為什麼 `lw` 轉送的值未經過錯誤檢查也沒有問題。

如果在載入讀取的資料中偵測到錯誤，則管線將停止並排清。因此，載入指令和所有後續指令（其中一些指令接收不正確的轉送值）均會排清，永不提交。

任務：在圖14的範例中，刪除 `lw` 之前和 `add` 之後的所有 `nop` 指令。不要刪除兩條相關指令之間的5個 `nop`。分析模擬，然後透過在開發板上執行程式，用效能計數器計算 **IPC**（在測量 **IPC** 時保留 `nop` 指令似乎並不適合，因為它們是無用指令；不過，程式本身就是無用的，這裡我們的唯一目的是分析並瞭解資料冒險）。

不提供解答。