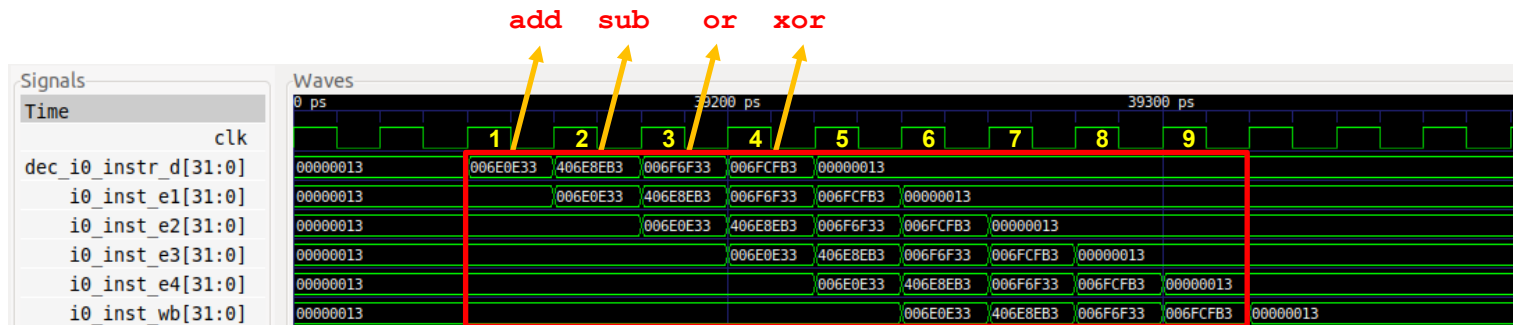


## 任務

**任務：**在圖3的模擬中加入追蹤訊號，並仿照圖4的形式強調顯示管線中從解碼階段到寫回階段的指令流。可以使用以下位置提供的.tcl檔案：

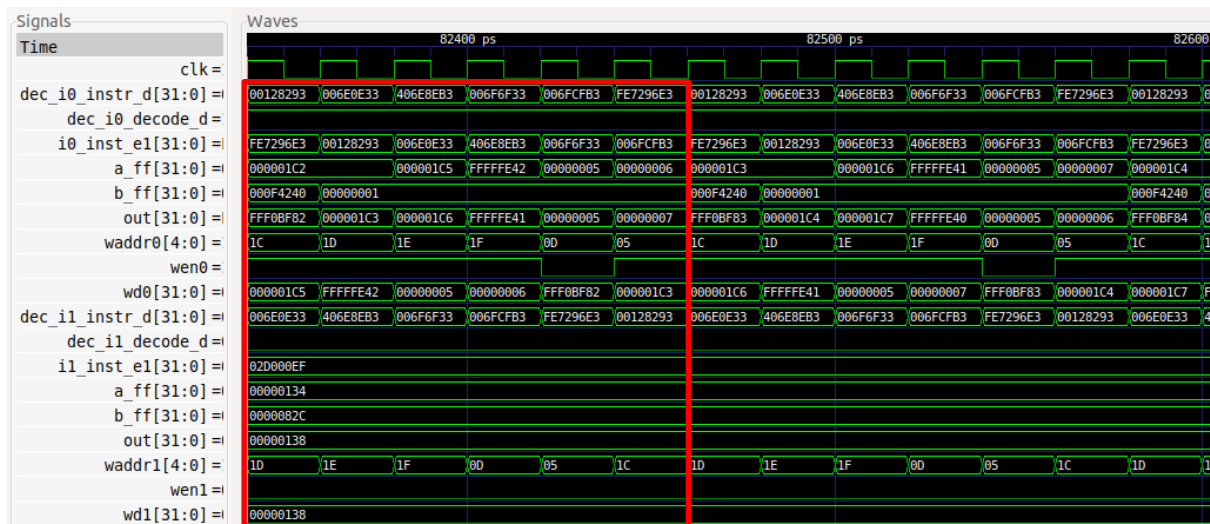
[RVfpgaPath]/RVfpga/Labs/Lab17/Four\_AL\_Instructions/test\_task1.tcl。



**任務：**刪除圖2所示的迴圈主體中的所有nop指令。

重複圖3中的模擬。該程式的預期IPC是多少？

在開發板上執行程式，驗證獲得的IPC是否符合您的預期。



迴圈中的6條指令需要透過6個週期執行。因此，IPC的預期值為1。

```

src > Test_Assembly.S
17
18 Test_Assembly:
19
20 li t2, 0x400          # Disable Dual-Issue Execution
21 csrrs t1, 0x7F9, t2
22
23 li t0, 0x0
24 li t1, 0x1
25 li t2, 0x1
26 li t3, 0x3
27 li t4, 0x4
28 li t5, 0x5
29 li t6, 0x6
30
31 lui t2, 0xF4
32 add t2, t2, 0x240
33
34 REPEAT:
35     add t0, t0, 1
36     add t3, t3, t1
37     sub t4, t4, t1
38     or  t5, t5, t1
39     xor t6, t6, t1
40     bne t0, t2, REPEAT # Repeat the loop
41
42 .end

```

PROBLEMS OUTPUT DEBUG CONSOLE **TERMINAL**

> Executing task: platformio device monitor <

```

--- Available filters and text transformations: colorize, debug,
--- 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 = 6000276
Instructions = 6000048

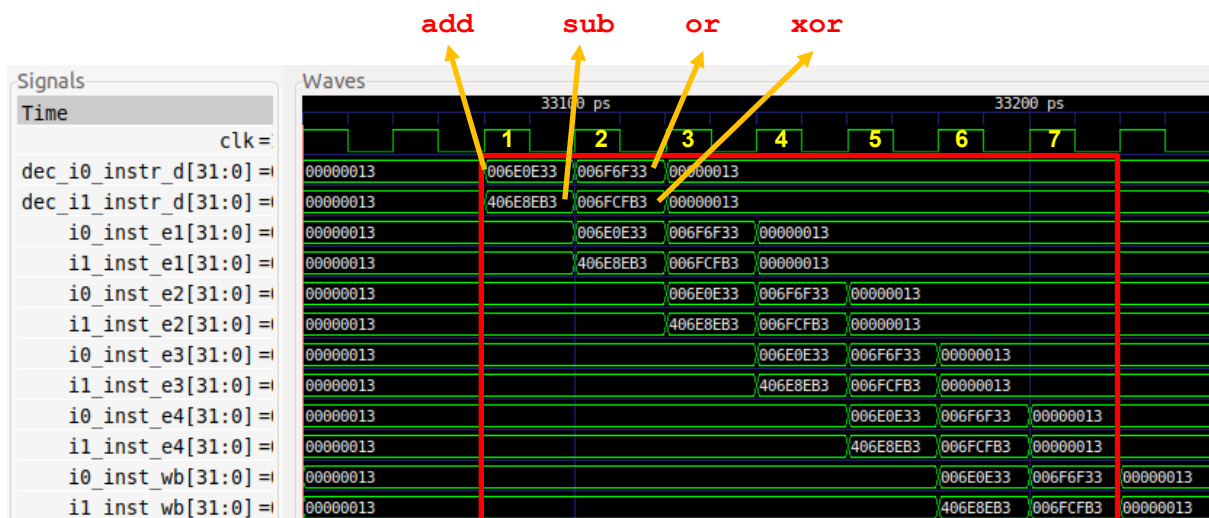
```

正如預期：

$$IPC = 6000000 \text{ instructions} / 6000000 \text{ cycles} = 1$$

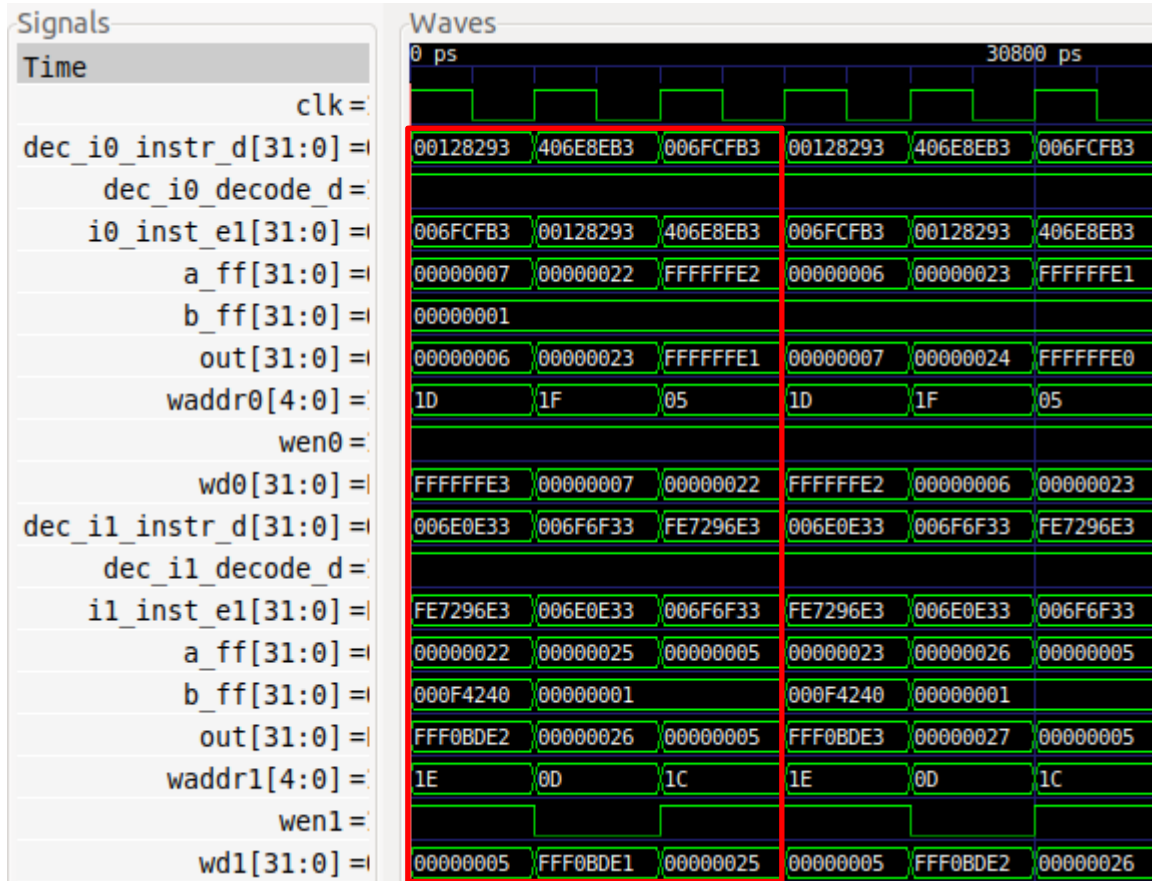
**任務：**在圖5的模擬中加入追蹤訊號，並仿照圖6的形式強調顯示管線中從解碼階段到寫回階段的指令流。可以使用以下位置提供的.tcl檔案：

[RVfpgaPath]/RVfpga/Labs/Lab17/Four\_AL\_Instructions/test\_task2.tcl。



**任務：**刪除圖2所示的迴圈主體中的所有nop指令。

重複圖5中的模擬。該程式的預期IPC是多少？  
在開發板上執行程式，驗證獲得的IPC是否符合您的預期。



迴圈中的6條指令需要透過3個週期執行。因此，IPC的預期值為2。

```

src > ASM Test_Assembly.S
1/
18 Test_Assembly:
19
20 //li t2, 0x400          # Disable Dual-Issue Execution
21 //csrrs t1, 0x7F9, t2
22
23 li t0, 0x0
24 li t1, 0x1
25 li t2, 0x1
26 li t3, 0x3
27 li t4, 0x4
28 li t5, 0x5
29 li t6, 0x6
30
31 lui t2, 0xF4
32 add t2, t2, 0x240
33
34 REPEAT:
35 add t0, t0, 1
36 add t3, t3, t1
37 sub t4, t4, t1
38 or t5, t5, t1
39 xor t6, t6, t1
40 bne t0, t2, REPEAT # Repeat the loop
41
42 .end

```

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

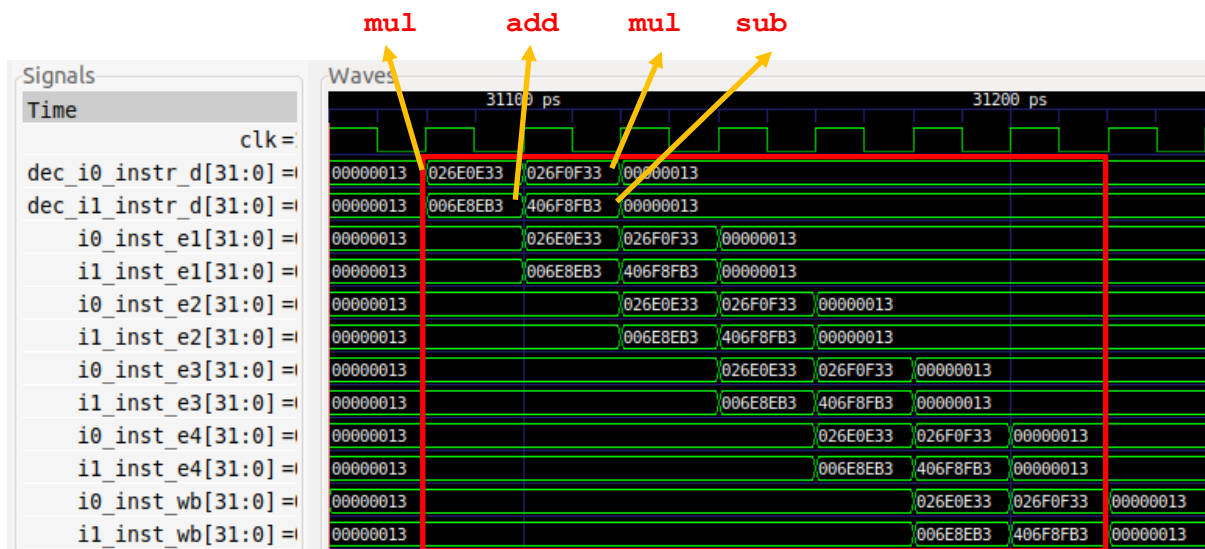
```

Cycles = 3000253  
Instructions = 6000046

正如預期：

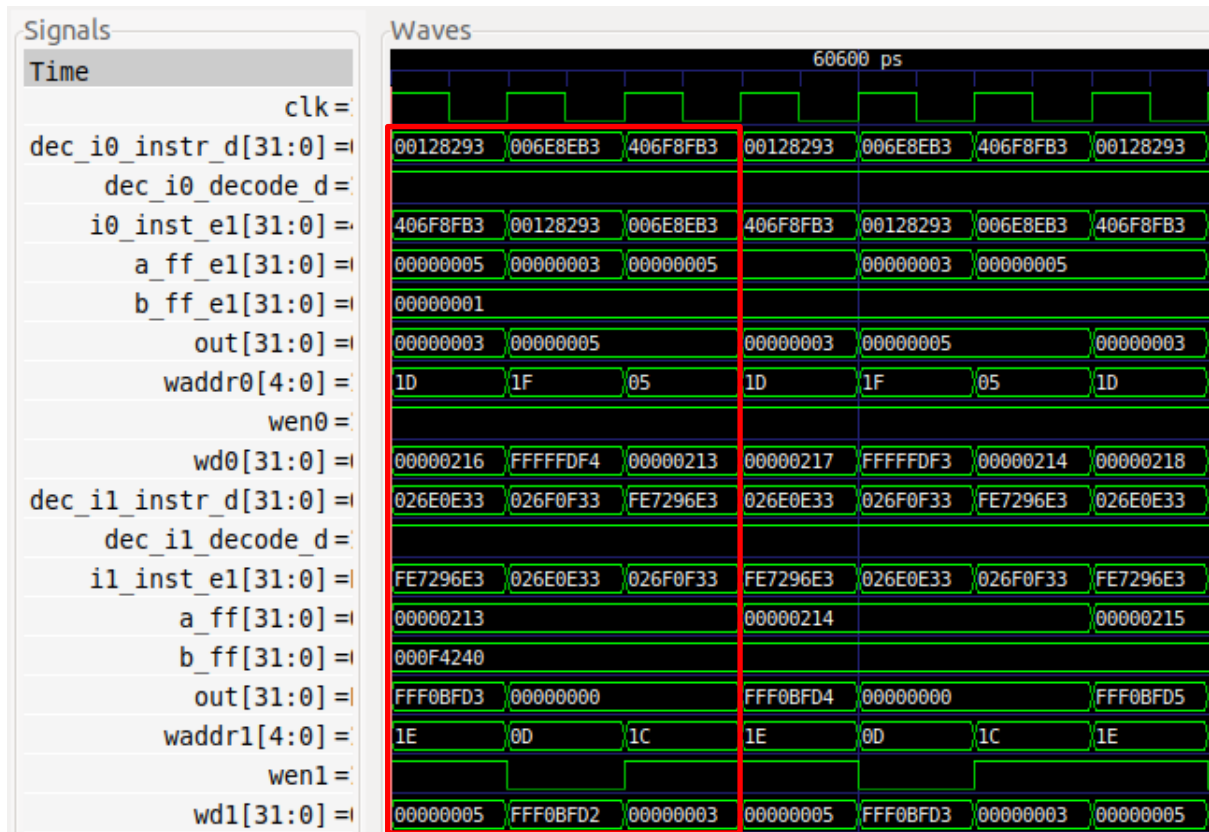
$$IPC = 6000000 \text{ instructions} / 3000000 \text{ cycles} = 2$$

**任務：**在圖8的模擬中加入追蹤訊號，並強調顯示管線中從解碼階段到寫回階段的指令流。  
可以使用以下位置提供的.tcl檔案：  
[RVfpgaPath]/RVfpga/Labs/Lab17/TwoAL\_TwoMUL\_Instructions/test\_taskMuls.tcl。



**任務：**刪除圖7所示的迴圈主體中的所有nop指令。  
 重複圖8中的模擬。該程式的預期IPC是多少？  
 在開發板上執行程式，驗證獲得的IPC是否符合您的預期。  
 使用單指令組態重複實驗，並比較實驗結果。

雙指令：



迴圈中的6條指令需要透過3個週期執行。因此，IPC的預期值為2。

```

PIO Home platformio.ini Test_Assembly.S x firmware.dis
src > Test_Assembly.S
19
20 # li t2, 0x400          # Disable Dual-Issue Execution
21 # csrrs t1, 0x7F9, t2
22
23 li t0, 0x0
24 li t1, 0x1
25 li t2, 0x0
26 li t3, 0x3
27 li t4, 0x4
28 li t5, 0x5
29 li t6, 0x6
30 lui t2, 0xF4
31 add t2, t2, 0x240
32
33 REPEAT:
34   add t0, t0, 1
35   mul t3, t3, t1
36   add t4, t4, t1
37   mul t5, t5, t1
38   sub t6, t6, t1
39   bne t0, t2, REPEAT # Repeat the loop
40
41 .end

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
> Executing task: platformio device monitor <

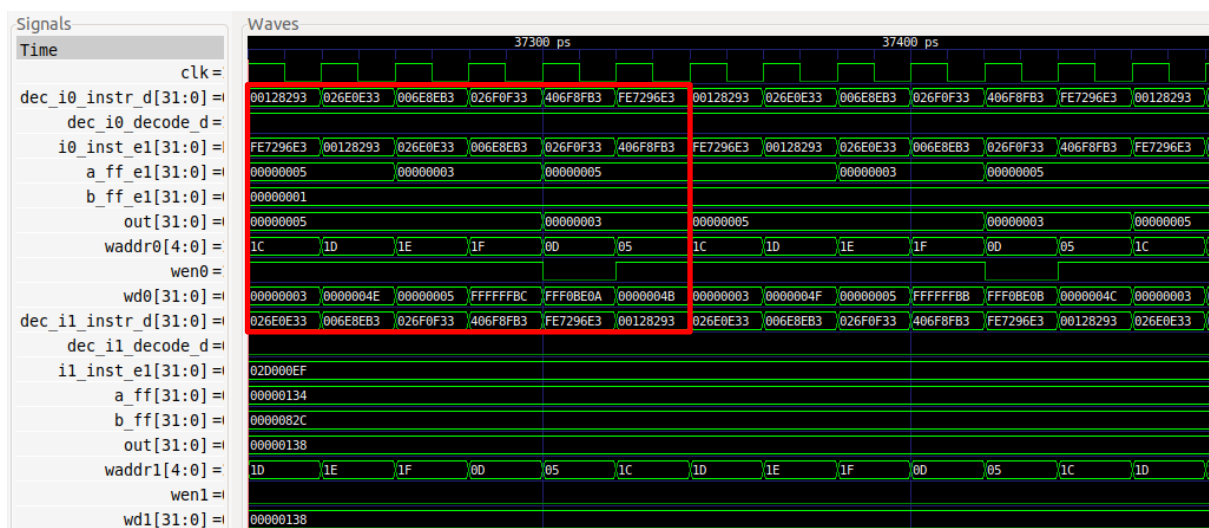
--- Available filters and text transformations: colorize, debug, d
--- More details at http://bit.ly/pio-monitor-filters
--- Miniterm on /dev/ttyUSB1 115200,8,N,1 ---
--- Quit: Ctrl+C | Menu: Ctrl+T | Help: Ctrl+T followed by Ctrl+H
Cycles = 3000263
Instructions = 6000046

```

正如預期：

$$IPC = 6000000 \text{ instructions} / 3000000 \text{ cycles} = 2$$

單指令：



迴圈中的6條指令需要透過6個週期執行。因此，IPC的預期值為1。

```

PIO Home platformio.ini Test_Assembly.S x Test.c
src > Test_Assembly.S
19
20 li t2, 0x400 # Disable Dual-Issue Execution
21 csrrs t1, 0x7F9, t2
22
23 li t0, 0x0
24 li t1, 0x1
25 li t2, 0x0
26 li t3, 0x3
27 li t4, 0x4
28 li t5, 0x5
29 li t6, 0x6
30 lui t2, 0xF4
31 add t2, t2, 0x240
32
33 REPEAT:
34 add t0, t0, 1
35 mul t3, t3, t1
36 add t4, t4, t1
37 mul t5, t5, t1
38 sub t6, t6, t1
39 bne t0, t2, REPEAT # Repeat the loop
40
41 .end

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
> Executing task: platformio device monitor <
--- Available filters and text transformations: colorize, debug, default
--- 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 = 6000277
Instructions = 6000048

```

正如預期：

$$\text{IPC} = 6000000 \text{ instructions} / 6000000 \text{ cycles} = 1$$

## 練習

- 1) 使用能夠展示雙指令執行相關新情境的指令組合，建立與圖2和圖7所示程式類似的程式。

不提供解答。

- 2) 分析（雙指令）SweRV EH1處理器與S.Harris和D.Harris所著教材《數位設計和電腦體系結構》（RISC-V版本，簡稱[DDCARV]）第7.7.4節中所示的超標量處理器（為方便起見，該處理器已在圖1中提供）之間的差異。

不提供解答。

- 3) 分析DDCARV第7.7.4節的圖7.70中的程式，該程式位於資料夾 `[RVfpgaPath]/RVfpga/Labs/Lab17/DDCARV_SuperscalarExample` 下的PlatformIO專案中。使用模擬器和開發板兩種方式，在SweRV EH1上執行該程式（對於後一種方式，請刪除nop指令）。解釋得到的結果。如有必要，可調整程式以獲得最理想的IPC。

接下來，根據本實驗及SweRVref.docx（第2部分）中所述，停用雙指令執行。將模擬結果和開發板上的執行結果與啟用雙指令功能時的相應結果進行比較。



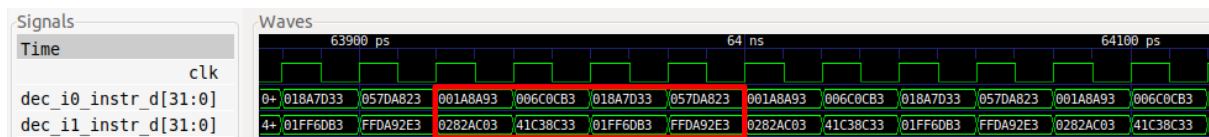
000001c0 <REPEAT>:

```

1c0: 001a8a93      addi    s5,s5,1
1c4: 0282ac03      lw      s8,40(t0)
1c8: 006c0cb3      add     s9,s8,t1
1cc: 41c38c33      sub     s8,t2,t3
1d0: 018a7d33      and     s10,s4,s8
1d4: 01ff6db3     or      s11,t5,t6
1d8: 057da823     sw      s7,80(s11)
1dc: ffd9a2e3     bne     s5,t4,1c0 <REPEAT>

```

模擬 – 雙指令：



迴圈不存在暫停，並且每個週期始終執行2條指令。系統會執行一些旁路，在某些情況下還會使用輔助ALU，如之前的實驗中所述。可以進一步分析這些情況。

開發板上的執行結果 – 雙指令：

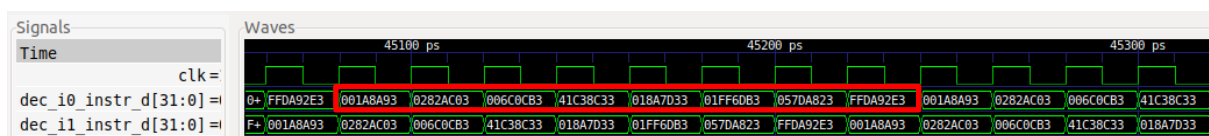
```

src > C Test.c > main(void)
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, log2file
... 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 = 4000258
Instructions = 8000057

```

IPC等於2，無需調整程式碼順序。

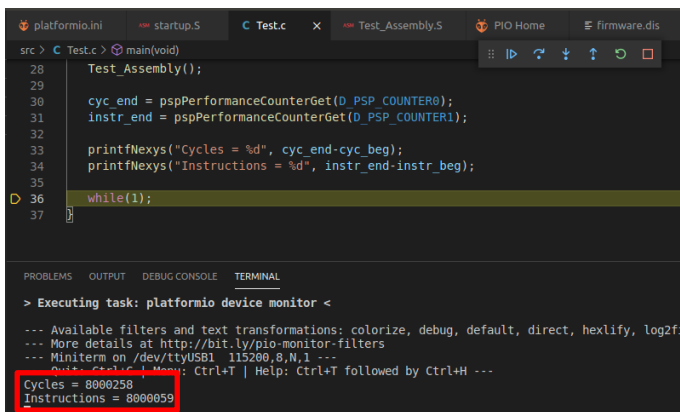
模擬 – 單指令：



迴圈不存在暫停，並且每個週期始終執行1條指令。

開發板上的執行結果 – 單指令：





```

src > C Test.c > main(void)
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 = 8000258
Instructions = 8000059

```

IPC等於1，這是單指令SweRV EH1中所能達到的最佳值。

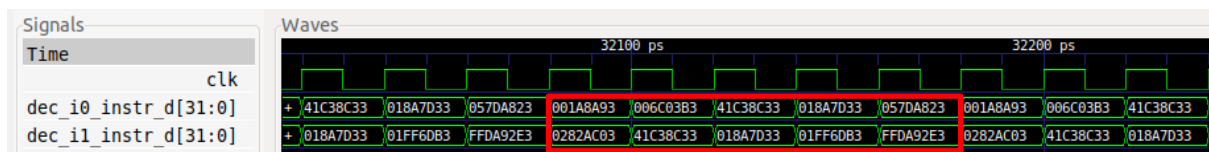
- 4) 修改練習3中的程式，將指令add s9, s8, t1替換為add t2, s8, t1。解釋得到的結果。如有必要，可調整程式以獲得最理想的IPC。

然後，根據本實驗及SweRVref.docx（第2節）中所述，停用雙指令執行。將模擬結果和開發板上的執行結果與啟用雙指令功能時的相應結果進行比較。

000001c0 <REPEAT>:

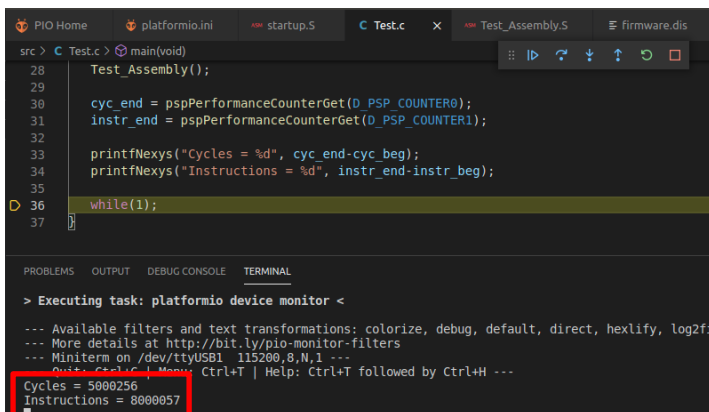
1c0:	001a8a93	addi	s5,s5,1
1c4:	0282ac03	lw	s8,40(t0)
1c8:	006c03b3	add	t2,s8,t1
1cc:	41c38c33	sub	s8,t2,t3
1d0:	018a7d33	and	s10,s4,s8
1d4:	01ff6db3	or	s11,t5,t6
1d8:	057da823	sw	s7,80(s11)
1dc:	ffda92e3	bne	s5,t4,1c0 <REPEAT>

模擬 – 雙指令：



由於AL指令之間的相關性，迴圈中存在2個暫停。

開發板上的執行結果 – 雙指令：



```

src > C Test.c > main(void)
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, 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 = 5000256
Instructions = 8000057

```

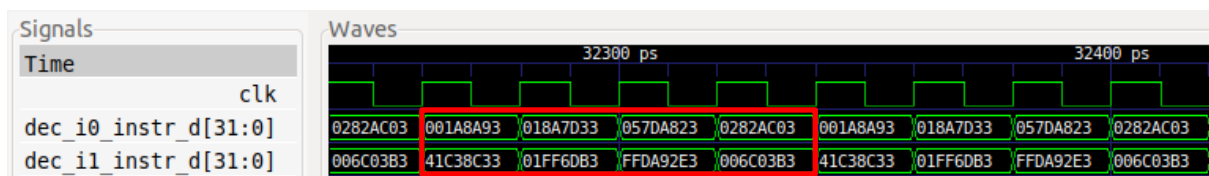
由於存在暫停，IPC小於2。

調整程式碼順序：

000001c0 <REPEAT>:

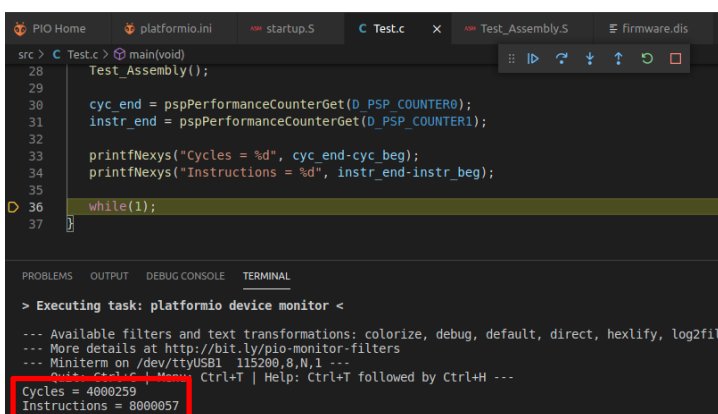
1c0:	0282ac03	lw	s8,40(t0)
1c4:	006c03b3	add	t2,s8,t1
1c8:	001a8a93	addi	s5,s5,1
1cc:	41c38c33	sub	s8,t2,t3
1d0:	018a7d33	and	s10,s4,s8
1d4:	01ff6db3	or	s11,t5,t6
1d8:	057da823	sw	s7,80(s11)
1dc:	ffda92e3	bne	s5,t4,1c0 <REPEAT>

模擬 – 雙指令：



迴圈不存在暫停。

開發板上的執行結果 – 雙指令：



```

src > C Test.c > main(void)
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,N,1 ---
--- Quit: Ctrl+C | Menu: Ctrl+T | Help: Ctrl+T followed by Ctrl+H ---
Cycles = 4000259
Instructions = 8000057

```

IPC等於2。

回到原始程式：

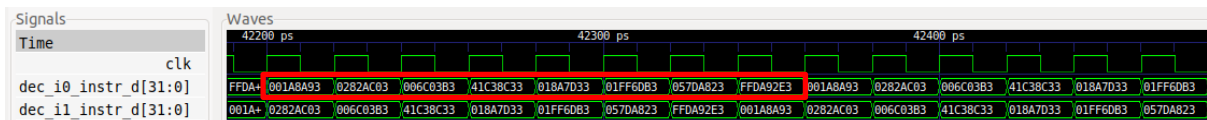
000001c0 <REPEAT>:

```

1c0: 001a8a93      addi    s5,s5,1
1c4: 0282ac03      lw      s8,40(t0)
1c8: 006c03b3      add     t2,s8,t1
1cc: 41c38c33      sub     s8,t2,t3
1d0: 018a7d33      and     s10,s4,s8
1d4: 01ff6db3      or      s11,t5,t6
1d8: 057da823      sw      s7,80(s11)
1dc: ffd92e3       bne     s5,t4,1c0 <REPEAT>

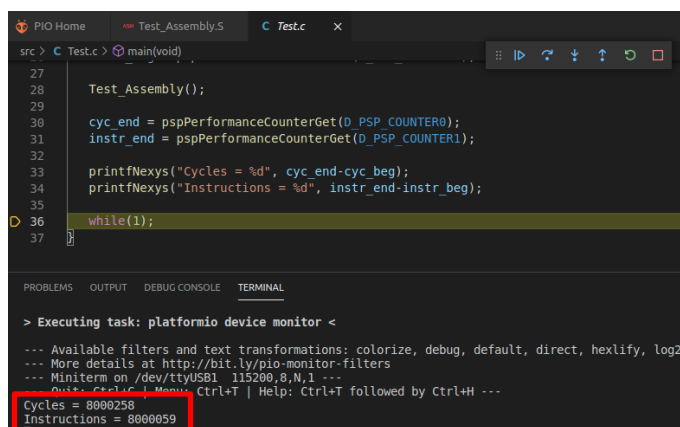
```

模擬 – 單指令：



迴圈不存在暫停。

開發板上的執行結果 – 單指令：



```

src > Test.c > main(void)
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, log2
--- 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 = 8000258
Instructions = 8000059

```

IPC等於1。

### 5) (以下練習基於《電腦組織結構和設計》(RISC-V版本, Patterson & Hennessy ([HePa])) 中的練習4.31。)

在本練習中，我們將比較單指令處理器和雙指令處理器的效能，並考慮可最佳化雙指令執行的程式修改。本練習中的問題涉及以下迴圈（用C語言編寫）：

```
for(i=0; i!=j; i+=2) b[i]=a[i]-a[i+1];
```

僅經過少量最佳化或未經最佳化的編譯器可能會產生以下RISC-V組合語言程式碼：

```

li x12, 0
li x13, 8000

```

```

li x14, 0
TOP:
    slli x5, x12, 2
    add x6, x10, x5
    lw x7, 0(x6)
    lw x29, 4(x6)
    sub x30, x7, x29
    add x31, x11, x5
    sw x30, 0(x31)
    addi x12, x12, 2
    ENT: bne x12, x13, TOP

```

該程式碼使用以下暫存器：

i	j	a	b	Temporary values
x12	x13	x10	x11	x5-x7, x29-x31

該程式碼位於 *[RVfpgaPath]/RVfpga/Labs/Lab17/PaHe\_SuperscalarExample* 路徑下，與本書練習中提供的程式碼相比，存在一些小的修改，但不會影響程式的行為：

- 將暫存器x13初始化為8，以使程式執行4次迭代。
- 刪除了jal指令。
- 將ld和sd指令替換為lw和sw指令。這表示存取從4個位元組寬變為8個位元組寬。

假設有一個具備以下屬性的雙指令靜態調度處理器：

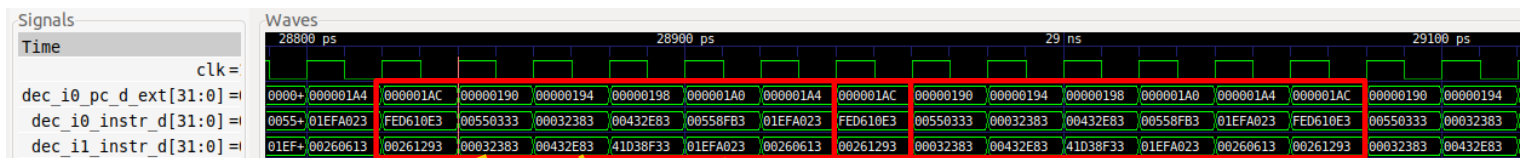
1. 一條指令必須是記憶體操作；另一條必須是算術/邏輯指令或分支指令。
2. 處理器的各階段之間支援所有可能的轉送路徑。
3. 處理器可實現完美分支預測。
4. 如果兩條指令相互依賴，則可能無法同時發出。
5. 如果某一階段需要暫停指令，則必須同時暫停兩條指令。

- a) 比較該範例處理器與SweRV EH1處理器的屬性。
- b) 繪製管線圖和模擬圖，顯示上述RISC-V程式碼的任意一次迴圈迭代（第一次迭代除外）在雙指令SweRV EH1處理器上的執行情況。假設在進行4000次迭代後退出迴圈（上述程式碼即為如此）。
- c) 從單指令SweRV EH1處理器變為雙指令SweRV EH1處理器時，加速比為多少？解釋得到的結果。在開發板上測試程式並啟用/停用雙指令執行。
- d) 重新排列/重寫上述RISC-V程式碼，以在雙指令SweRV EH1處理器上實現更高的效能。（但請勿展開迴圈。）
- e) 接下來展開RISC-V程式碼，使展開後迴圈的每次迭代等於原始迴圈的兩次迭代。然後，重新排列/重寫展開的程式碼，以在雙指令SweRV EH1處理器上實現更高的效能。

b)

0000018c <TOP>:

18c: 00261293	slli t0,a2,0x2
190: 00550333	add t1,a0,t0
194: 00032383	lw t2,0(t1)
198: 00432e83	lw t4,4(t1)
19c: 41d38f33	sub t5,t2,t4
1a0: 00558fb3	add t6,a1,t0
1a4: 01efa023	sw t5,0(t6)
1a8: 00260613	addi a2,a2,2



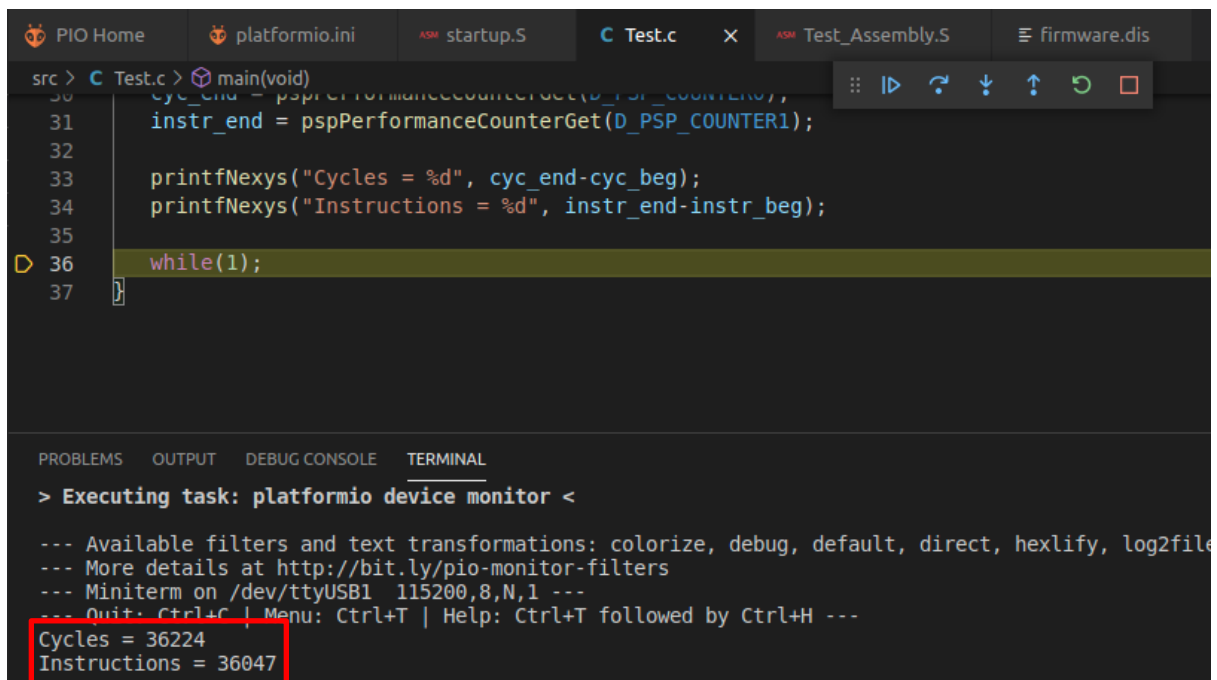
由於add t1,a0,t0存在寫後讀 (RAW) 資料冒險，lw t2,0(t1)被延遲。實驗15的最後一項任務 (第2.A部分) 中已分析過此情況。

由於兩條lw指令間存在結構冒險而延遲。

由於add t6,a1,t0存在寫後讀 (RAW) 資料冒險，sw t5,0(t6)被延遲。

c)

單指令：



```

src > C Test.c > main(void)
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

```

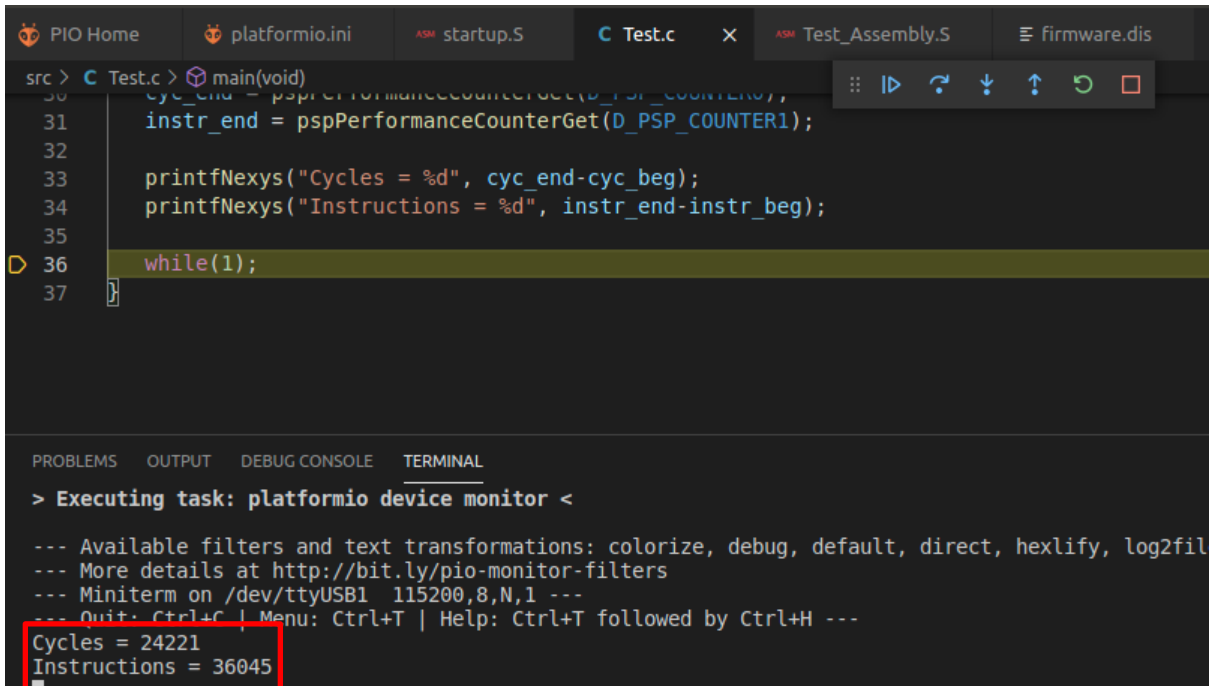
```

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL
> Executing task: platformio device monitor <

--- Available filters and text transformations: colorize, debug, default, direct, hexlify, log2file
--- 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 = 36224
Instructions = 36047

```

雙指令：



The screenshot shows the PlatformIO IDE with a C file named `Test.c` open. The code defines a `main` function that uses the `pspPerformanceCounter` to measure execution time. It prints the number of cycles and instructions. A `while(1);` loop is present at the bottom of the code. The terminal output shows the results of the execution:

```
> 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 = 24221
Instructions = 36045
```

加速比 = 單指令時間/雙指令時間 =  $36224/24221 \approx 1.5$  (理想情況下加速比應為2，但根據(b)中的分析，由於存在暫停，實際加速比僅為1.5)

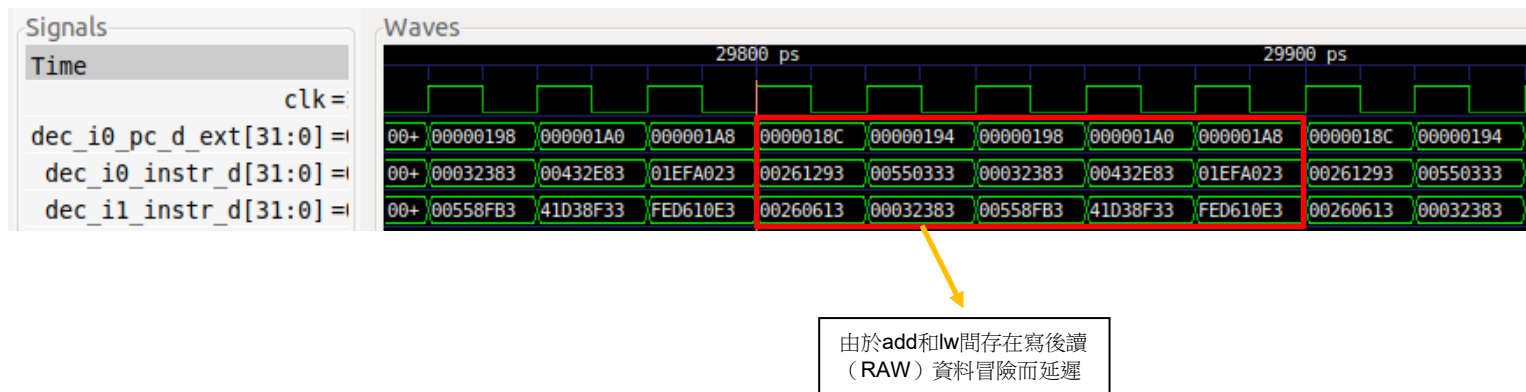
d)

TOP:

```
slli x5, x12, 2
addi x12, x12, 2
add x6, x10, x5
lw x7, 0(x6)
add x31, x11, x5
lw x29, 4(x6)
sub x30, x7, x29
sw x30, 0(x31)
ENT: bne x12, x13, TOP
```

0000018c <TOP>:

18c: 00261293	slli	t0,a2,0x2
190: 00260613	addi	a2,a2,2
194: 00550333	add	t1,a0,t0
198: 00032383	lw	t2,0(t1)
19c: 00558fb3	add	t6,a1,t0
1a0: 00432e83	lw	t4,4(t1)
1a4: 41d38f33	sub	t5,t2,t4
1a8: 01efa023	sw	t5,0(t6)



```
platformio.ini  ASM startup.S  C Test.c  x  PIO Home  ASM Test_Assembly.S  firmware.dis

src > C Test.c > main(void)
22  pspPerformanceCounterSet(D_PSP_COUNTER0, E_CYCLES_COUNT);
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,N,1 ---
--- Quit: Ctrl+C | Menu: Ctrl+T | Help: Ctrl+T followed by Ctrl+H ---
Cycles = 20224
Instructions = 36045
```

$$\text{SPEEDUP} = \text{Time\_Single} / \text{Time\_Dual} = 36224 / 20224 \approx 1.8$$



e)

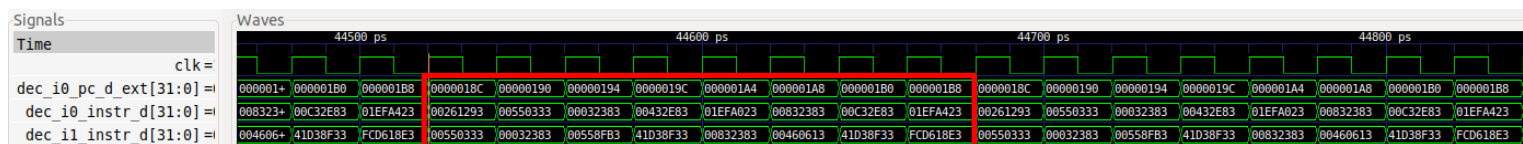
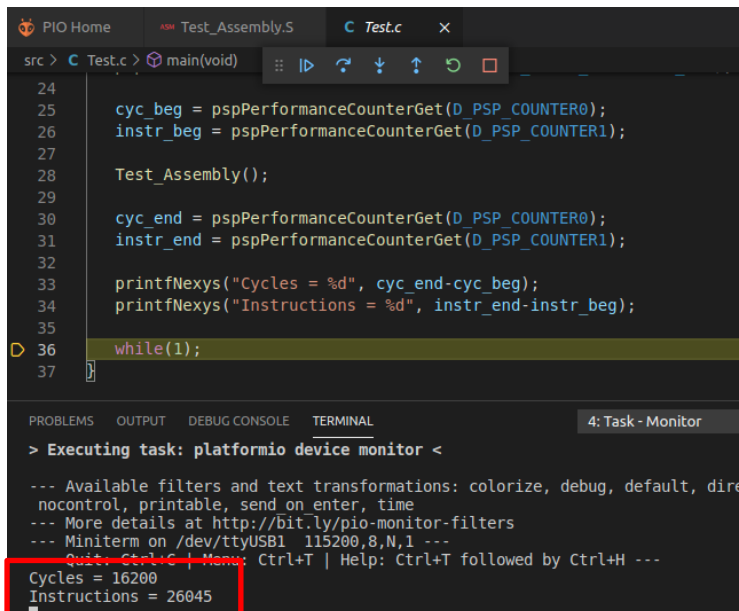
TOP:

```
slli x5, x12, 2
add x6, x10, x5
```

```
lw x7, 0(x6)
add x31, x11, x5
lw x29, 4(x6)
sub x30, x7, x29
sw x30, 0(x31)
```

```
lw x7, 8(x6)
addi x12, x12, 4
lw x29, 12(x6)
sub x30, x7, x29
sw x30, 8(x31)
```

```
ENT: bne x12, x13, TOP
```

The screenshot shows the PIO Home IDE with the Test.c file open. The code includes performance counter initialization and a while loop. The Task-Monitor output shows the execution results:

```
> Executing task: platformio device monitor <

--- Available filters and text transformations: colorize, debug, default, direct,
nocontrol, printable, send on enter, time
--- 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 = 16200
Instructions = 26045
```

得益於程式碼的展開和重寫，效能進一步提升。

6) (以下練習基於DDCARV第7章的練習7.30、7.32和7.34。)

假設SweRV EH1處理器正在執行以下程式碼片段。回想一下，SweRV EH1具有冒險單元。可以假設一個能夠在一個週期內傳回結果的記憶體系統（為此，我們使用DCCM，將程式碼片段插入一個迴圈中，並避免使用第一次迭代，以免發生IS未命中）。

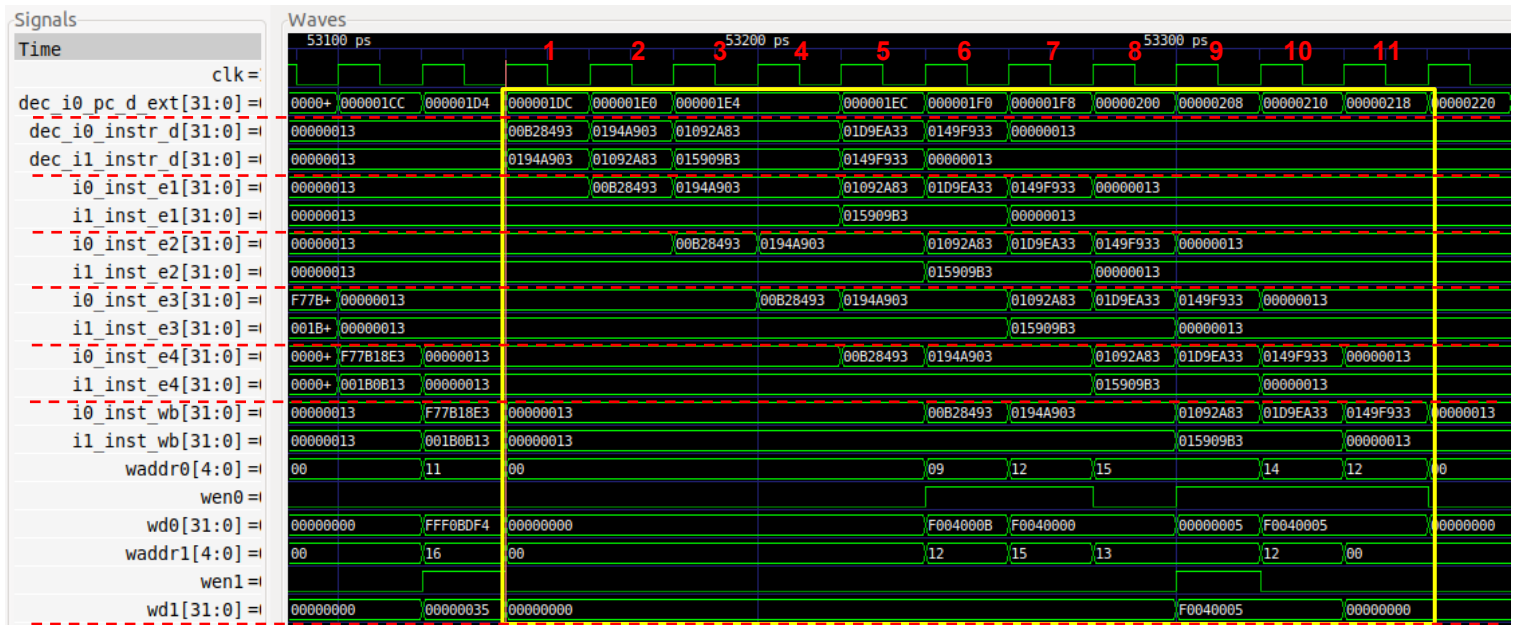
```
addi s1, t0, 11    # t0 contains the base address of the DCCM
lw   s2, 25(s1)
lw   s5, 16(s2)
add  s3, s2, s5
or   s4, s3, t4
and  s2, s3, s4
```

- a) 使用Verilator和GTKWave模擬該程式。分析結果，並針對每個週期指明以下內容：
  - \* 對哪些指令進行解碼？將哪些指令傳送至執行階段？提交哪些指令？
  - \* 對哪些暫存器進行寫入操作和讀取操作？
  - \* 發生哪些轉送和暫停情況？
- b) 對於該程式，處理器的CPI是多少？先做出理論解答，然後在開發板上執行程式，確認您的答案。
- c) 在單指令處理器上執行相同的分析，將結果與使用雙指令處理器時的結果進行比較。

PlatformIO專案的路徑如下：[\[RVfpgaPath\]/RVfpga/Labs/Lab17/DDCARV\\_Exercises-30-32-34](#)。所分析的程式將插入一個迴圈中，以使模擬更易理解（可使用除第一次迭代外的任何迭代進行分析）並且可以使用效能計數器進行測定。

a)

1dc:	00b28493	addi	s1,t0,11
1e0:	0194a903	lw	s2,25(s1)
1e4:	01092a83	lw	s5,16(s2)
1e8:	015909b3	add	s3,s2,s5
1ec:	01d9ea33	or	s4,s3,t4
1f0:	0149f933	and	s2,s3,s4



a. 第1個週期：

i. 通路0：

1. 指令addi（0x00b28493）處於解碼階段。此週期結束時，指令進入EX1階段。

ii. 通路1：

1. 指令lw（第一次載入，0x0194a903）處於解碼階段。此週期結束時，指令仍留在解碼階段，因為該指令的階段取決於前一條addi指令的結果。

b. 第2個週期：

i. 通路0：

1. 指令lw（第一次載入）再次處於解碼階段，但從通路1移至通路0。有效位址的輸入運算元取自addi指令。指令lw透過轉送的方式從addi指令接收用於計算有效位址的輸入運算元。此週期結束時，指令進入DC1階段。
2. 指令addi處於EX1階段。在此週期中，該指令會取得相加後的結果並將其轉送至第一次載入。

ii. 通路1：

1. 指令lw（第二次載入，0x01092a83）處於解碼階段。此週期結束時，由於與第一次載入之間存在結構冒險，該指令無法進入下一階段。

c. 第3個週期：

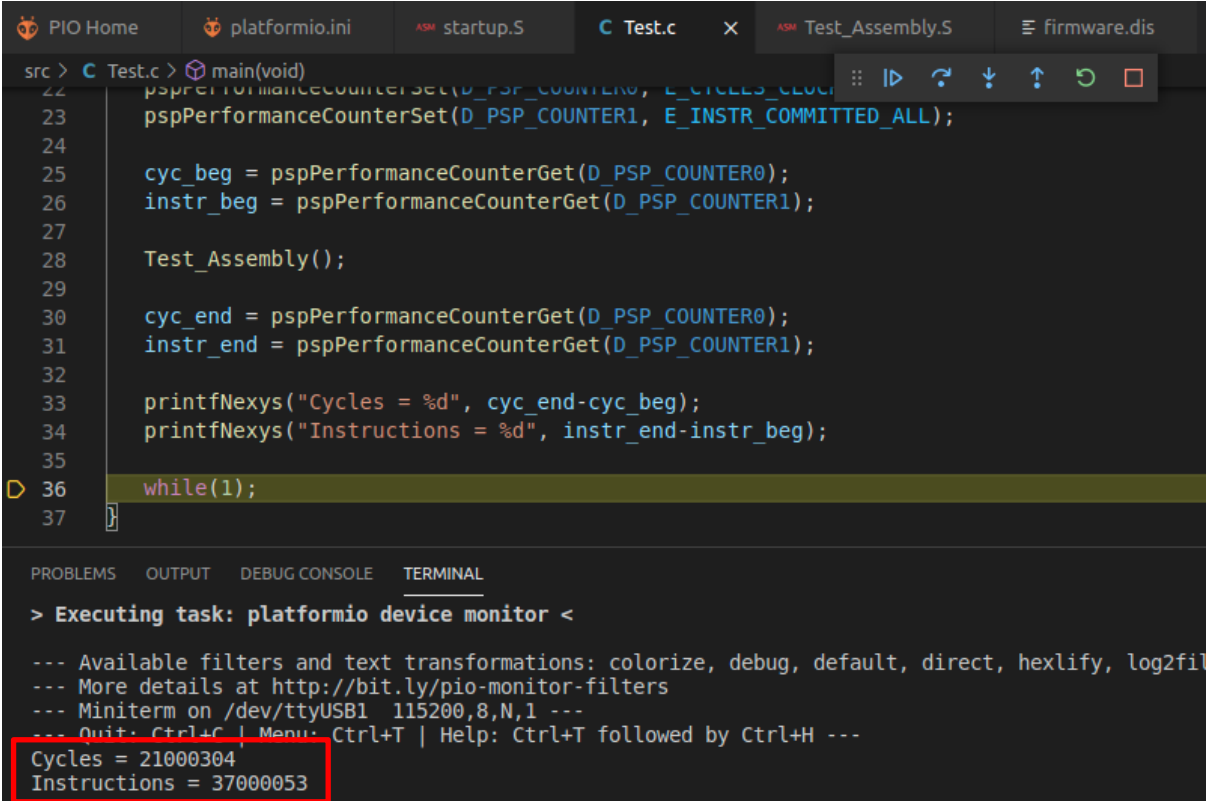
i. 通路0：

1. 指令lw（第二次載入）再次處於解碼階段，但從通路1移至通路0。由於第二次載入需要使用第一次載入讀取的值來計算有效位址，該指令無法進入下一階段。在DC2階段結束時可取得該值（參見實驗13）。

2. 指令lw（第一次載入）處於DC1階段。
3. 指令addi處於EX2階段。
- ii. 通路1：
  1. 指令add（0x015909b3）處於解碼階段。該指令無法進入下一階段，因為兩條載入指令存在資料冒險。
- d. 第4個週期：
  - i. 通路0：
    1. 指令lw（第二次載入）再次處於解碼階段。此時，由於LSU管道空閒且已透過轉送取得其運算元，該指令可以進入下一階段。
    2. 指令lw（第一次載入）處於DC2階段。
    3. 指令addi處於EX3階段。
  - ii. 通路1：
    1. 指令add再次處於解碼階段。該指令從兩次載入中取得兩個輸入運算元，因此將在輔助ALU中執行（參見實驗15）。
- e. 第5個週期：
  - i. 通路0：
    1. 指令or（0x01d9ea33）處於解碼階段。透過轉送取得第一個運算元後，該指令可以進入下一階段。
    2. 指令lw（第二次載入）處於DC1階段。該指令會從第一次載入中接收用於計算有效位址的輸入運算元，隨後即可進入下一階段。
    3. 指令lw（第一次載入）處於DC3階段。
    4. 指令addi處於提交階段（EX4）。
  - ii. 通路1：
    1. 指令and處於解碼階段。該指令無法進入下一階段，因為通路0中的or指令在解碼階段存在資料冒險（參見實驗15）。
    2. 指令add處於EX1階段。
- f. 第6個週期：
  - i. 通路0：
    1. 指令and再次處於解碼階段，但從通路1移至通路0。該指令會透過轉送的形式取得兩個輸入運算元，隨後即可進入下一階段。
    2. 指令or處於EX1階段。該指令會將結果轉送給and指令。
    3. 指令lw（第二次載入）處於DC2階段。
    4. 指令lw（第一次載入）處於提交階段。
    5. 指令addi處於寫回階段（EX5）。向暫存器x9寫入0xF004000B。
  - ii. 通路1：
    1. 指令add處於EX2階段。
- g. 第7個週期：
  - i. 通路0：
    1. 指令and（0x0149f933）處於EX1階段。

2. 指令or處於EX2階段。
  3. 指令lw（第二次載入）處於DC3階段。
  4. 指令lw（第一次載入）處於寫回階段。向暫存器x12寫入0xF0040000。
- ii. 通路1：
1. 指令add處於EX3階段。
- a. 第8個週期：
- iii. 通路0：
1. 指令and處於EX2階段。
  2. 指令or處於EX3階段。
  3. 指令lw（第二次載入）處於提交階段。
- iv. 通路1：
1. 指令add處於提交階段。
- b. 第9個週期：
- v. 通路0：
1. 指令and處於EX3階段。
  2. 指令or處於提交階段。
  3. 指令lw（第二次載入）處於寫回階段。向暫存器x15寫入0x00000005。
- vi. 通路1：
1. 指令add處於寫回階段。向暫存器x13寫入0xF0040005。
- c. 第10個週期：
- vii. 通路0：
1. 指令and處於提交階段。
  2. 指令or處於寫回階段。向暫存器x14寫入0xF0040005。
- d. 第11個週期：
- viii. 通路0：
1. 指令and處於寫回階段。向暫存器x12寫入0xF0040005。

b)



```

src > C Test.c > main(void)
22 pspPerformanceCounterSet(D_PSP_COUNTER0, E_CYCLES_CLOCK);
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,N,1 ---
--- Quit: Ctrl+C | Menu: Ctrl+T | Help: Ctrl+T followed by Ctrl+H ---
Cycles = 21000304
Instructions = 37000053

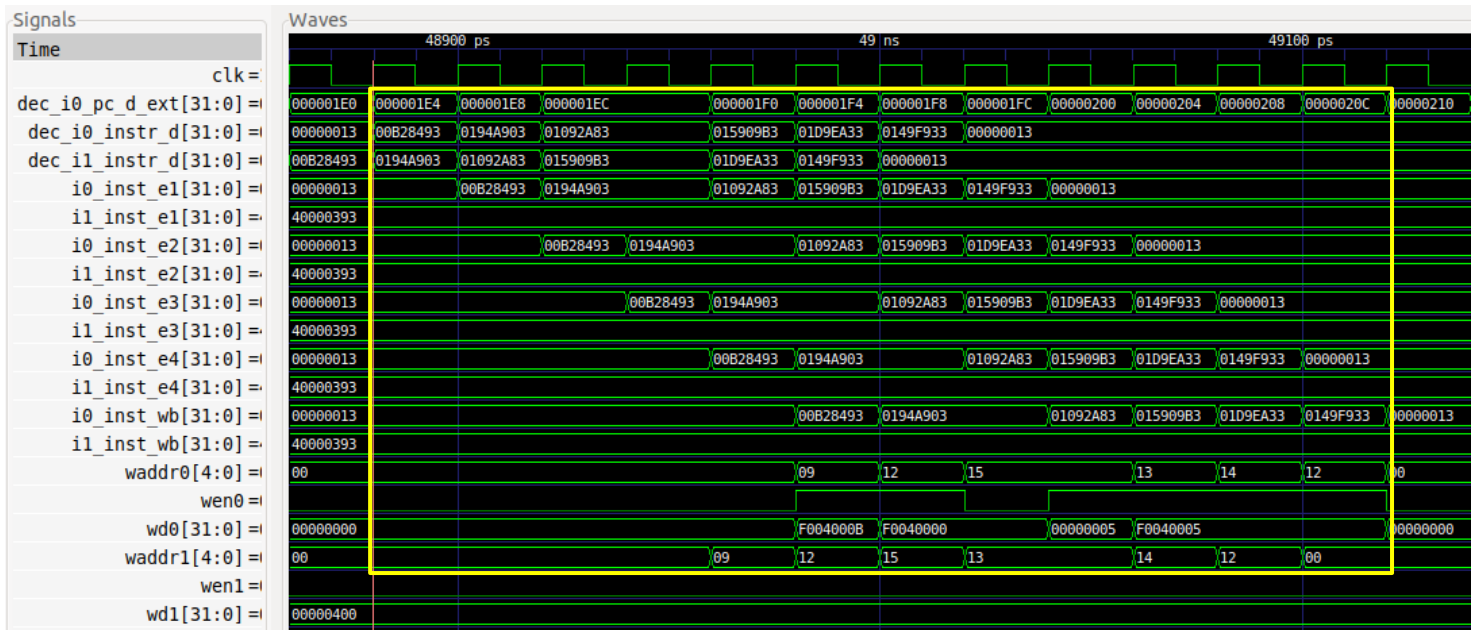
```

請注意，在該計算中，必須刪除加入的所有額外指令：首先是addi指令，然後是29條nop指令，最後是bne指令。上述指令均可以在 $\frac{1}{2}$ 個週期內執行（在執行我們程式的最後一條指令時，會有一個空閒時隙）。

因此： $IPC = (37-31) / (21-15.5-0.5) = 6 / 6 = 1$

與我們在模擬中觀察到的情況完全相同。

c)



每個週期執行1條指令，唯一的例外是當兩次載入（0x0194a903和0x01092a83）之間存在資料冒險時，此時系統在管線中插入一個氣泡並遺失1個週期。

```

firmware.dis  PIO Home  C Test.c  X  ASM Test_Assembly.S  ASM startup.S
src > C Test.c > main(void)
21
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, log2
--- 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 = 38000320
Instructions = 37000055

```

和之前一樣，為了計算IPC，必須刪除31條額外的指令。這些指令均可在一個週期內執行。



因此： $IPC = (37-31) / (38-31) = 6 / 7$

與我們在模擬中觀察到的情況完全相同。

7) (以下練習基於DDCARV第7章的練習7.31、7.33和7.35。)

使用以下程式碼片段重複練習7。

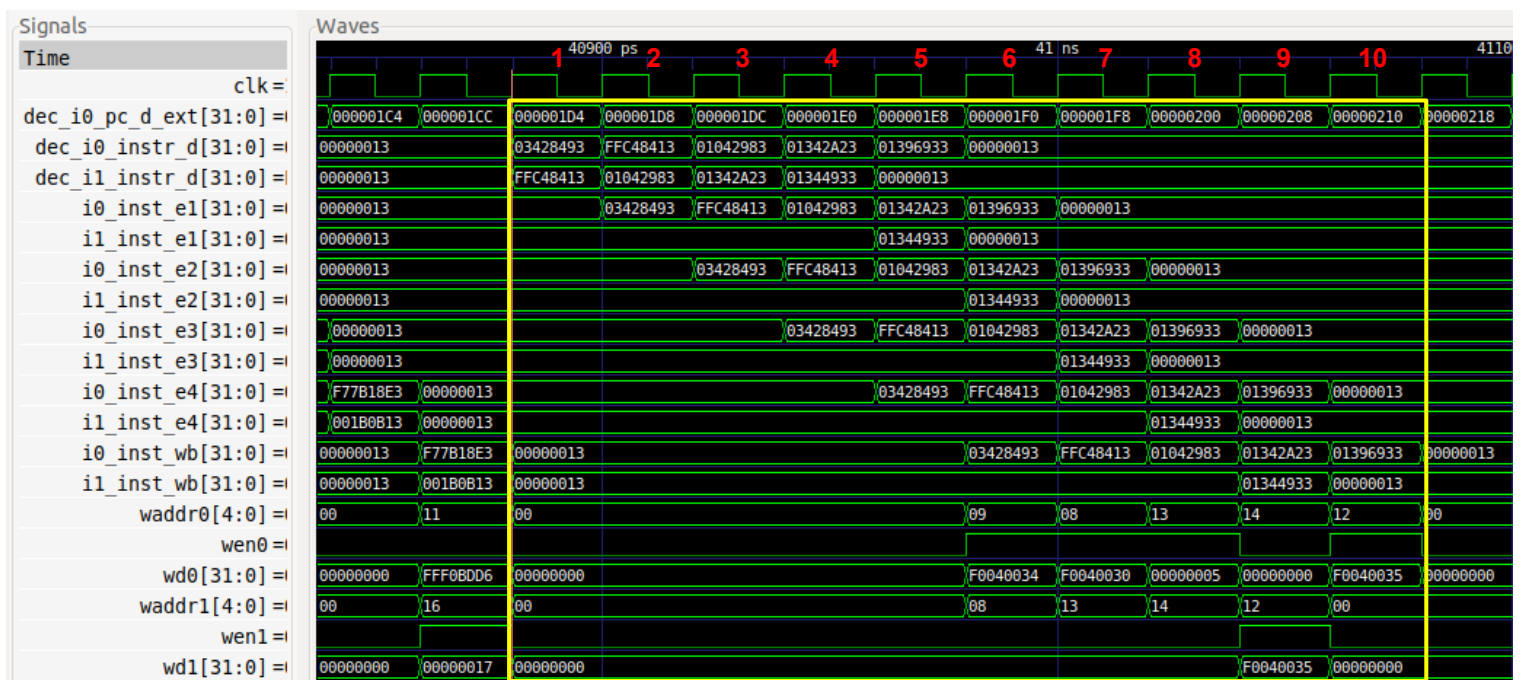
```
addi s1, t0, 52
addi s0, s1, -4
lw    s3, 16(s0)
sw    s3, 20(s0)
xor   s2, s0, s3
or    s2, s2, s3
```

PlatformIO專案的路徑如下：

[RVfpgaPath]/RVfpga/Labs/Lab17/DDCARV\_Exercises-31-33-35。

a)

1d4: 03428493	addi s1,t0,52
1d8: ffc48413	addi s0,s1,-4
1dc: 01042983	lw s3,16(s0)
1e0: 01342a23	sw s3,20(s0)
1e4: 01344933	xor s2,s0,s3
1e8: 01396933	or s2,s2,s3



除sw (0x01342a23) 和xor (0x01344933) 之外，所有連續指令之間都存在冒險。在這些連續指令中，lw (0x01042983) 和sw (0x01342a23) 之間存在結構冒險，其他連續指令之間均存在資料冒險。根據模擬結果，每個週期僅執行1條指令，但指令對sw-xor為例外，其中的xor指令由通路1執行。

b)

在該情況下： $IPC = 6 / 5$

c)



在該情況下： $IPC = 6 / 6 = 1$