

任務

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（根據實驗11的附錄D，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。

不提供解答。

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。

```

70 // ----- Input flops -----
71
72 rvdffs    #(1) valid_e1_ff    (*, .din(mp.valid),          .dout(valid_e1),          .clk(active_clk),          .en(~freeze));
73
74 rvdff_fpga #(1) rs1_sign_e1_ff (*, .din(mp.rs1_sign),      .dout(rs1_sign_e1),      .clk(exu_mul_c1_e1_clk), .clken(mul_c1_e1_clken), .rawclk(clk));
75 rvdff_fpga #(1) rs2_sign_e1_ff (*, .din(mp.rs2_sign),      .dout(rs2_sign_e1),      .clk(exu_mul_c1_e1_clk), .clken(mul_c1_e1_clken), .rawclk(clk));
76 rvdff_fpga #(1) low_e1_ff     (*, .din(mp.low),            .dout(low_e1),           .clk(exu_mul_c1_e1_clk), .clken(mul_c1_e1_clken), .rawclk(clk));
77 rvdff_fpga #(1) ld_rs1_byp_e1_ff (*, .din(mp.load_mul_rs1_bypass_e1), .dout(load_mul_rs1_bypass_e1), .clk(exu_mul_c1_e1_clk), .clken(mul_c1_e1_clken), .rawclk(clk));
78 rvdff_fpga #(1) ld_rs2_byp_e1_ff (*, .din(mp.load_mul_rs2_bypass_e1), .dout(load_mul_rs2_bypass_e1), .clk(exu_mul_c1_e1_clk), .clken(mul_c1_e1_clken), .rawclk(clk));
79
80 rvdffe    #(32) a_e1_ff      (*, .din(a[31:0]),            .dout(a_ff_e1[31:0]),    .en(mul_c1_e1_clken));
81 rvdffe    #(32) b_e1_ff      (*, .din(b[31:0]),            .dout(b_ff_e1[31:0]),    .en(mul_c1_e1_clken));
82
83
84
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]);
89
90 assign rs1_neg_e1      = rs1_sign_e1 & a_e1[31];
91 assign rs2_neg_e1      = rs2_sign_e1 & b_e1[31];
92
93
94 rvdffs    #(1) valid_e2_ff    (*, .din(valid_e1),          .dout(valid_e2),          .clk(active_clk),          .en(~freeze));
95
96 rvdff_fpga #(1) low_e2_ff     (*, .din(low_e1),            .dout(low_e2),           .clk(exu_mul_c1_e2_clk), .clken(mul_c1_e2_clken), .rawclk(clk));
97
98 rvdffe    #(33) a_e2_ff      (*, .din({rs1_neg_e1, a_e1[31:0]}), .dout(a_ff_e2[32:0]),    .en(mul_c1_e2_clken));
99 rvdffe    #(33) b_e2_ff      (*, .din({rs2_neg_e1, b_e1[31:0]}), .dout(b_ff_e2[32:0]),    .en(mul_c1_e2_clken));
100
101
102
103 // ----- E2 Logic Stage -----
104
105 logic signed [65:0] prod_e2;
106
107 assign prod_e2[65:0]    = a_ff_e2 * b_ff_e2;
108
109 rvdff_fpga #(1) low_e3_ff     (*, .din(low_e2),            .dout(low_e3),           .clk(exu_mul_c1_e3_clk), .clken(mul_c1_e3_clken), .rawclk(clk));
110
111 rvdffe    #(64) prod_e3_ff    (*, .din(prod_e2[63:0]),      .dout(prod_e3[63:0]),    .en(mul_c1_e3_clken));
112
113
114
115 // ----- E3 Logic Stage -----
116
117
118
119 assign out[31:0]        = low_e3 ? prod_e3[31:0] : prod_e3[63:32];
120

```

- 解碼階段產生的輸入和控制位元在第72-81行記錄。

M1 :

- 如果乘法和先前載入之間存在資料相關性，則在第87-88行進行轉送。
- 此外，輸入運算元符號的處理在第90-91行確定。請記住，RISC-V包括三個版本的「高位乘」運算：mulh、mulhsu和mulhu。

- 這些值傳播到M2。

M2：

- 實際的乘法在第108行執行。

M3：

- 低/高位部分在第119行傳回out[31:0]。執行mul指令時選擇低位部分，而執行三條mulh指令中的任何一條時選擇高位部分。

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。

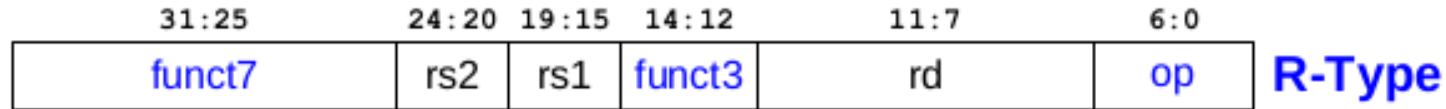
0x03de02b3 → 0000001 11101 11100 000 00101 0110011

**funct7 = 0000001
rs2 = 11101 = x29 (t4)
rs1 = 11100 = x28 (t3)
funct3 = 000
rd = 00101 = x5 (t0)
op = 0110011**

0x03ff0333 → 0000001 11111 11110 000 00110 0110011

**funct7 = 0000001
rs2 = 11111 = x31 (t6)
rs1 = 11110 = x30 (t5)
funct3 = 000
rd = 00110 = x6 (t1)
op = 0110011**

來自DDCARV的附錄B：



op	funct3	funct7	Type	Instruction	Description	Operation
0110011 (51)	000	0000001	R	mul rd, rs1, rs2	multiply	rd = (rs1 x rs2) _{31:0}

Name	Register Number	Use
zero	x0	Constant value 0
ra	x1	Return address
sp	x2	Stack pointer
gp	x3	Global pointer
tp	x4	Thread pointer
t0-2	x5-7	Temporary variables
s0/fp	x8	Saved variable / Frame pointer
s1	x9	Saved variable
a0-1	x10-11	Function arguments / Return values
a2-7	x12-17	Function arguments
s2-11	x18-27	Saved variables
t3-6	x28-31	Temporary variables

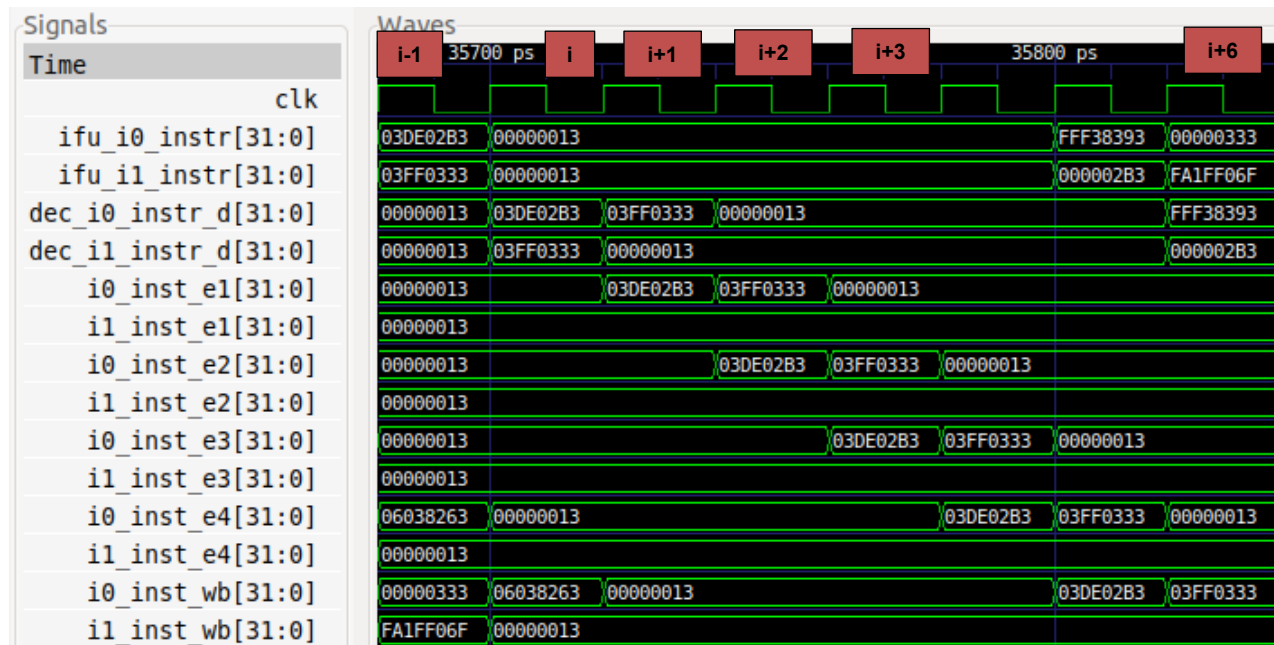
任務：在自己的電腦上重複圖2中的模擬過程，以進行更深入的分析。

解答請參見實驗14的主文件。

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分配給兩個通路。

- 在模組ifu_aln_ctl（對齊階段）中，這兩條指令分配給以下訊號：
- 通路0: ifu_i0_instr

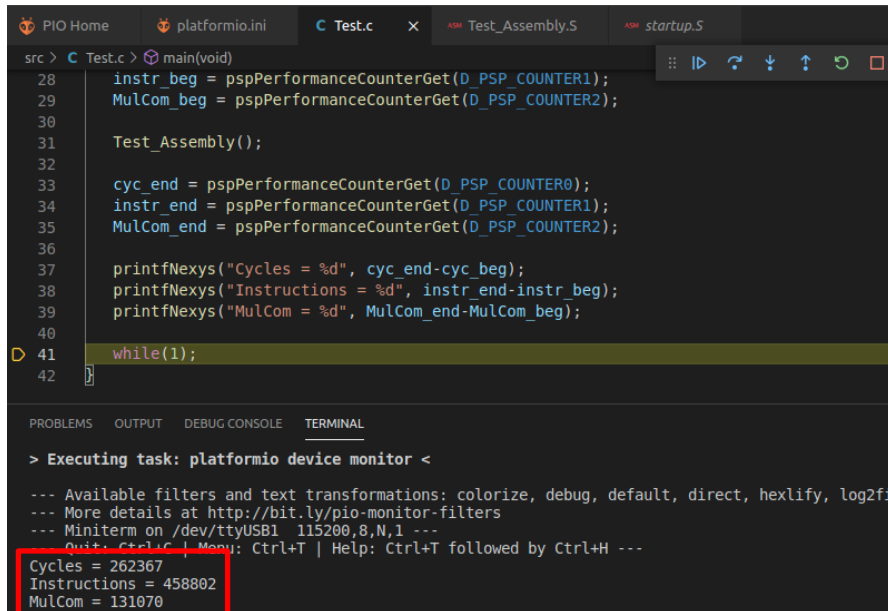
- 通路1: ifu_i1_instr
 - 在模組 **dec_ib_ctl** 中，這兩條指令進行緩衝（對齊到解碼階段）：
 - 通路0: ifu_i0_instr → dec_i0_instr_d
 - 通路1: ifu_i1_instr → dec_i1_instr_d
 - 在模組 **dec_decode_ctl**（解碼階段）中，這兩條指令盡可能調度給相應管道。傳送後，它們將繼續進行三個執行階段、提交階段和寫回階段：
 - 通路0: i0_inst_e1 - i0_inst_e2 - i0_inst_e3 - i0_inst_e4 - i0_inst_wb
 - 通路1: i1_inst_e1 - 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`）。



- 在週期 `i-1` 中（不顯示在圖2及圖3中），兩條 `mul` 指令位於對齊階段：在模組 `ifu_aln_ctl` 上，第一條分配給通路 0（`ifu_i0_instr = 0x03de02b3`），第二條分配給通路1（`ifu_i1_instr = 0x03ff0333`）。

- 在週期*i*中，兩條指令已傳播到模組**dec_ib_ctl**的解碼階段：第一條指令在通路0中繼續執行（`dec_i0_instr_d = 0x03de02b3`），第二條指令在通路1中繼續執行（`dec_i1_instr_d = 0x03ff0333`）。
- 在週期*i+1*中，第一條mul指令已傳播到**dec_decode_ctl**模組的M1階段（`i0_inst_e1 = 0x03de02b3`）。但第二條mul指令由於實驗中分析的結構冒險而無法傳播，因此在通路1的第一個執行階段插入了一個氣泡：`i1_inst_e1 = 0x00000013`。此外，鑒於通路0已在解碼階段釋放，第二條mul重新分配到了該通路：`dec_i0_instr_d = 0x03ff0333`。
- 在週期*i+2*中，第二條mul指令傳播到目前空閒的M1階段（`i0_inst_e1 = 0x03ff0333`），第一條mul指令傳播到M2階段。
- 在週期*i+3*至*i+6*中，兩條mul指令通過管線，直到寫回階段才暫停。

任務：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。



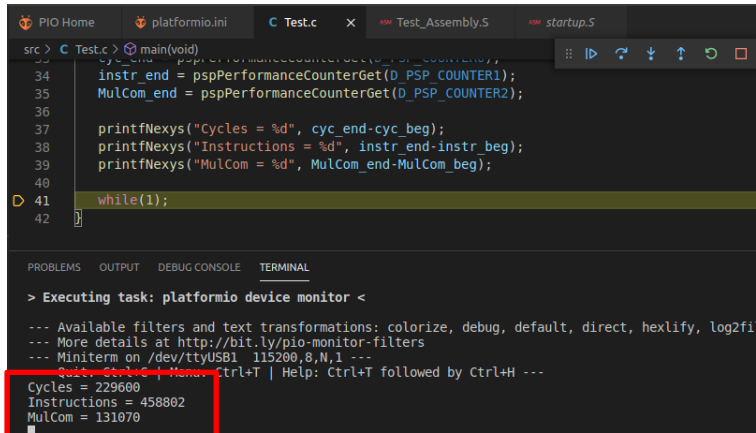
```
src > C Test.c > main(void)
28 instr_beg = pspPerformanceCounterGet(D_PSP_COUNTER1);
29 MulCom_beg = pspPerformanceCounterGet(D_PSP_COUNTER2);
30
31 Test_Assembly();
32
33 cyc_end = pspPerformanceCounterGet(D_PSP_COUNTER0);
34 instr_end = pspPerformanceCounterGet(D_PSP_COUNTER1);
35 MulCom_end = pspPerformanceCounterGet(D_PSP_COUNTER2);
36
37 printfNexys("Cycles = %d", cyc_end-cyc_beg);
38 printfNexys("Instructions = %d", instr_end-instr_beg);
39 printfNexys("MulCom = %d", MulCom_end-MulCom_beg);
40
41 while(1);
42
```

```
> 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 = 262367
Instructions = 458802
MulCom = 131070
```

$IPC = 458000 / 262000 = 1.748$ 。IPC比理想值小一點，因為第二條mul指令由於結構冒險必須等待一個週期，如實驗中所述。

如果我們調整程式碼順序，在兩條mul指令之間插入迴圈索引的更新，將獲得理想IPC，因為我們使用有用的指令填充了結構冒險引入的氣泡。

```
22 REPEAT:
23     beq t2, zero, OUT    # Stay in the loop?
24     mul t0, t3, t4       # t0 = t3 * t4
25     add t2, t2, -1
26     mul t1, t5, t6       # t1 = t5 * t6
27     add t0, zero, zero
28     add t1, zero, zero
29     j REPEAT
30 OUT:
```



```

src > C Test.c > main(void)
34     cyc_end = pspPerformanceCounterGet(D_PSP_COUNTER1);
35     instr_end = pspPerformanceCounterGet(D_PSP_COUNTER1);
36     MulCom_end = pspPerformanceCounterGet(D_PSP_COUNTER2);
37
38     printfNexys("Cycles = %d", cyc_end-cyc_beg);
39     printfNexys("Instructions = %d", instr_end-instr_beg);
40     printfNexys("MulCom = %d", MulCom_end-MulCom_beg);
41
42     while(1);
43 }

```

```

> 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 = 229600
Instructions = 458802
MulCom = 131070

```

$$\text{IPC} = 458000 / 229000 = 2$$

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（已提交的週期、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。

- C程式（原始和反組譯）：

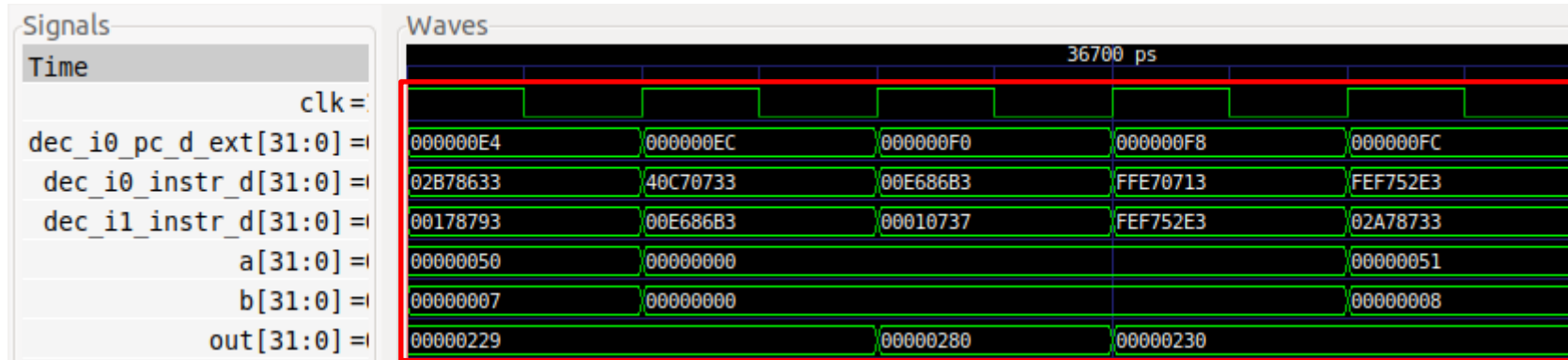
```
11  int Test_C(int a, int d)
12  {
13      int b, c, e=0, i=1;
14      do {
15          b = a*i;
16          c = d*i;
17          i = i+1;
18          e = e + (b-c);
19      } while(i<65535);
20      return(e);
21  }
```

```

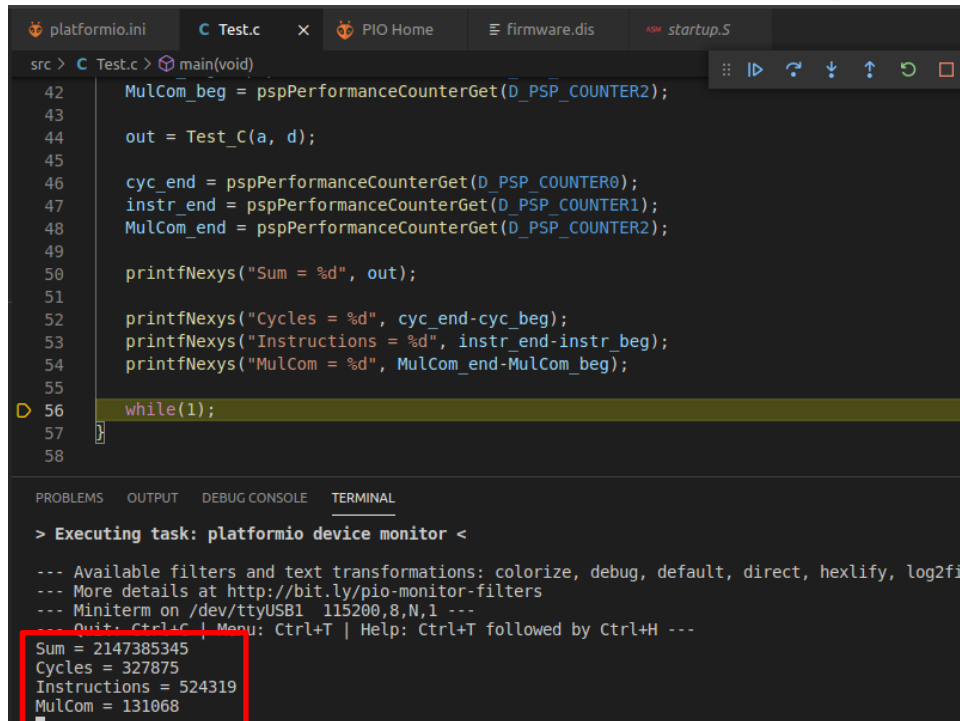
66 000000d8 <Test_C>:
67 d8: 00100793      li a5,1
68 dc: 00000693      li a3,0
69 e0: 02a78733      mul a4,a5,a0
70 e4: 02b78633      mul a2,a5,a1
71 e8: 00178793      addi a5,a5,1
72 ec: 40c70733      sub a4,a4,a2
73 f0: 00e686b3      add a3,a3,a4
74 f4: 00010737      lui a4,0x10
75 f8: ffe70713      addi a4,a4,-2 # fffe <_sp+0xc386>
76 fc: fef752e3      bge a4,a5,e0 <Test_C+0x8>
77 100: 00068513      mv a0,a3
78 104: 00008067      ret

```

- C程式的模擬：



- 硬體計數器：



The screenshot shows the PlatformIO IDE with a C file named `Test.c` open. The code defines a `main` function that uses Pico Performance Counters to measure execution time. It calculates the sum of a series, the number of cycles, instructions, and multi-ported accesses. A `while(1);` loop is present at the end of the function. The terminal window at the bottom shows the output of the program, which is highlighted with a red box:

```
> 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 ---

Sum = 2147385345
Cycles = 327875
Instructions = 524319
MulCom = 131068
```

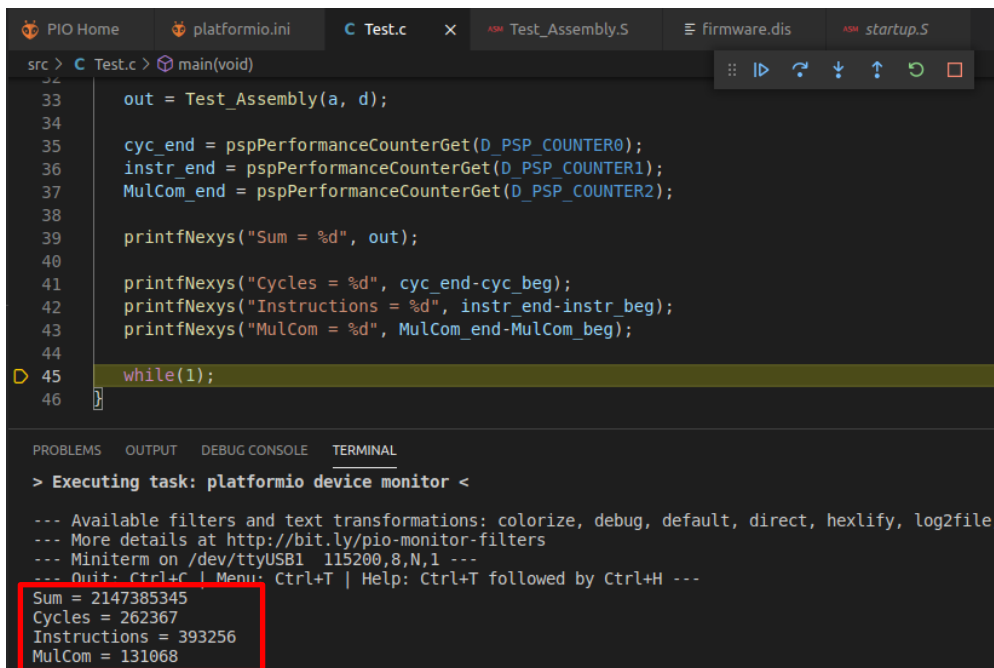
$IPC = 524000 / 327000 = 1.6$ 。由於實驗15中將分析的RAW資料冒險，因此遺失了一些週期。

- Assembly 程式位於：
`[RVfpgaPath]/RVfpga/Labs/RVfpgaLabsSolutions/Programs_Solutions/Lab14/MUL_Instr_Accumul_Assembly`

```

127 000001c4 <Test_Assembly>:
128 1c4: 00100393      li t2,1
129 1c8: 00000e93      li t4,0
130 1cc: 00000f93      li t6,0
131 1d0: 00010637      lui a2,0x10
132 1d4: fff60613      addi a2,a2,-1 # ffff <_sp+0xc567>
133 1d8: 00050e33      add t3,a0,zero
134 1dc: 00058f33      add t5,a1,zero
135
136 000001e0 <REPEAT>:
137 1e0: 027e02b3      mul t0,t3,t2
138 1e4: 027f0333      mul t1,t5,t2
139 1e8: 00138393      addi t2,t2,1
140 1ec: 40628eb3      sub t4,t0,t1
141 1f0: 01df8fb3      add t6,t6,t4
142 1f4: fec396e3      bne t2,a2,1e0 <REPEAT>
143 1f8: 00018533      add a0,t6,zero
144 1fc: 00008067      ret

```



The screenshot shows the PlatformIO IDE with the following components:

- Editor:** Displays the `Test.c` file. The `while(1);` loop is highlighted in green. The code includes performance counter initialization and printing of sum, cycles, instructions, and multiplication counts.
- Terminal:** Shows the output of the program execution. The results are:


```

Sum = 2147385345
Cycles = 262367
Instructions = 393256
MulCom = 131068

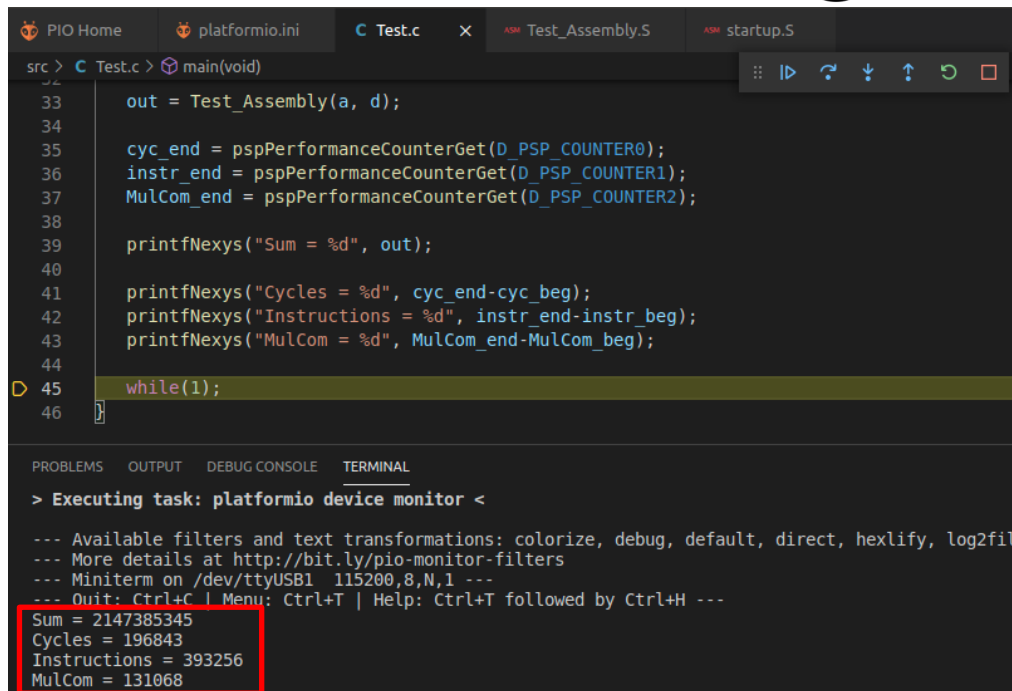
```

求和的結果相同，因為程式相同。

週期數略少一點，因為手動編寫的組合語言版本比未經最佳化的編譯器獲得的組合語言版本效率更高。
指令數也更少一點。

我們按如下方式調整迴圈順序：

```
15  li  t2, 0x1
16  li  t4, 0x0
17  li  t6, 0x0
18  li  a2, 0xFFFF
19  add t3, a0, zero
20  add t5, a1, zero
21
22  REPEAT:
23      mul t0, t3, t2      # t0 = t3 * t2
24      mul t1, t5, t2      # t1 = t5 * t2
25      sub t4, t0, t1
26      add t2, t2, 1
27      add t6, t6, t4
28      bne t2, a2, REPEAT  # Repeat the loop
29
30  add a0, t6, zero
```



```

src > C Test.c > main(void)
33 out = Test_Assembly(a, d);
34
35 cyc_end = pspPerformanceCounterGet(D_PSP_COUNTER0);
36 instr_end = pspPerformanceCounterGet(D_PSP_COUNTER1);
37 MulCom_end = pspPerformanceCounterGet(D_PSP_COUNTER2);
38
39 printfNexys("Sum = %d", out);
40
41 printfNexys("Cycles = %d", cyc_end-cyc_beg);
42 printfNexys("Instructions = %d", instr_end-instr_beg);
43 printfNexys("MulCom = %d", MulCom_end-MulCom_beg);
44
45 while(1);
46

```

```

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 ---

Sum = 2147385345
Cycles = 196843
Instructions = 393256
MulCom = 131068

```

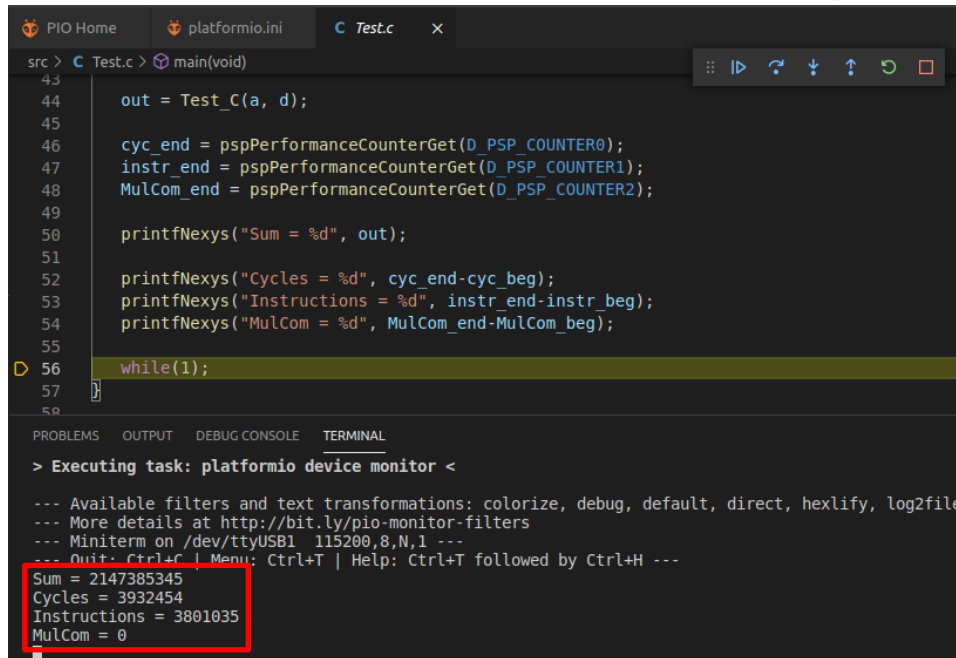
求和的結果相同，因為程式相同。

每次迭代的週期數 = $196800 / 65500 = 3$

指令數相同。每次迭代的指令數 = $393000 / 65500 = 6$

IPC = $393 / 197 = 1.994$ 。我們獲得了最佳IPC。

- 停用M延伸功能：



The screenshot shows a code editor with a C program in `Test.c`. The program calculates the sum of two numbers, `a` and `d`, and prints the result. It also measures the execution time in cycles, instructions, and multipliers. The output shows that the sum is 2147385345, the number of cycles is 3932454, the number of instructions is 3801035, and the number of multipliers is 0.

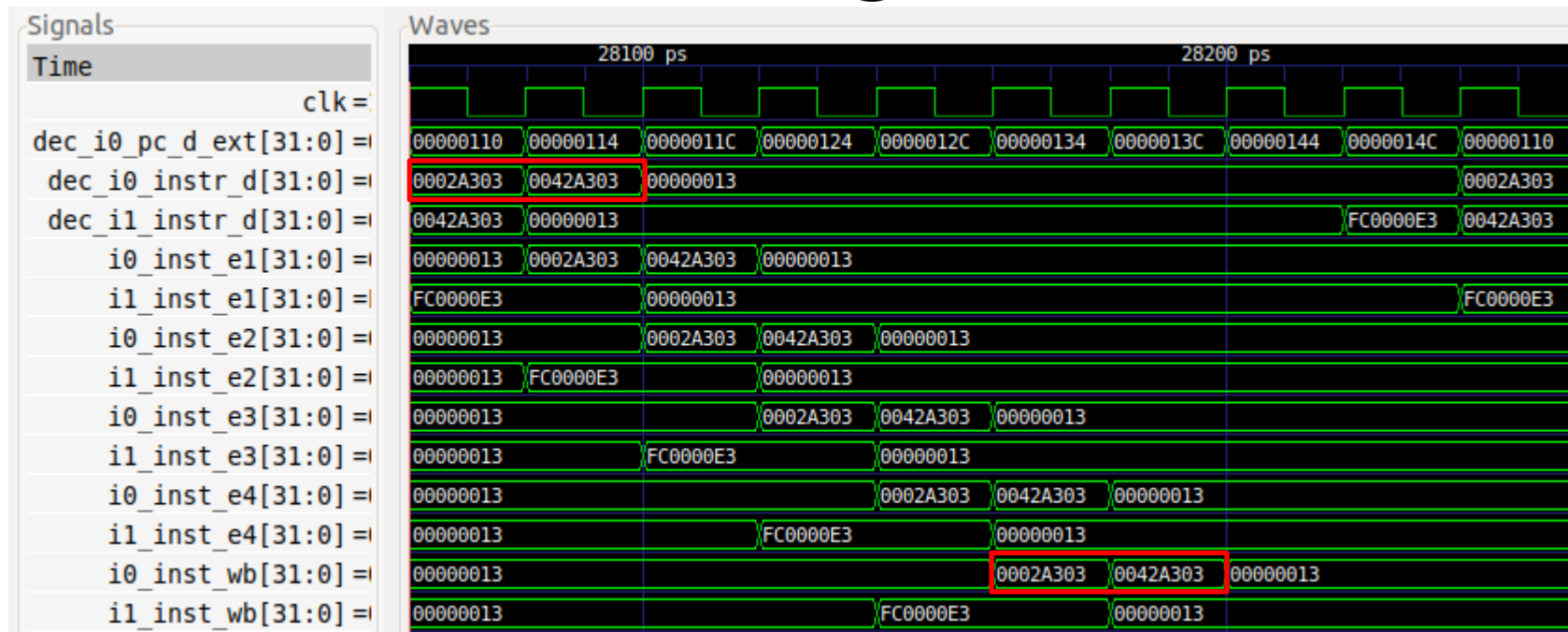
```

src > C Test.c > main(void)
43
44     out = Test_C(a, d);
45
46     cyc_end = pspPerformanceCounterGet(D_PSP_COUNTER0);
47     instr_end = pspPerformanceCounterGet(D_PSP_COUNTER1);
48     MulCom_end = pspPerformanceCounterGet(D_PSP_COUNTER2);
49
50     printfNexys("Sum = %d", out);
51
52     printfNexys("Cycles = %d", cyc_end-cyc_beg);
53     printfNexys("Instructions = %d", instr_end-instr_beg);
54     printfNexys("MulCom = %d", MulCom_end-MulCom_beg);
55
56     while(1);
57
58
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 ---
Sum = 2147385345
Cycles = 3932454
Instructions = 3801035
MulCom = 0


```

求和的結果相同，因為程式相同。
週期數多很多：約4M與約0.3M。
指令數也多很多：約3M與約0.5M。
CPI現在更好。
未提交乘法。

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。



正如我們在模擬中看到的那樣，兩次連續載入的行為與兩條連續mul指令的行為完全相同。

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。

解答請參見實驗14的主文件。

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。

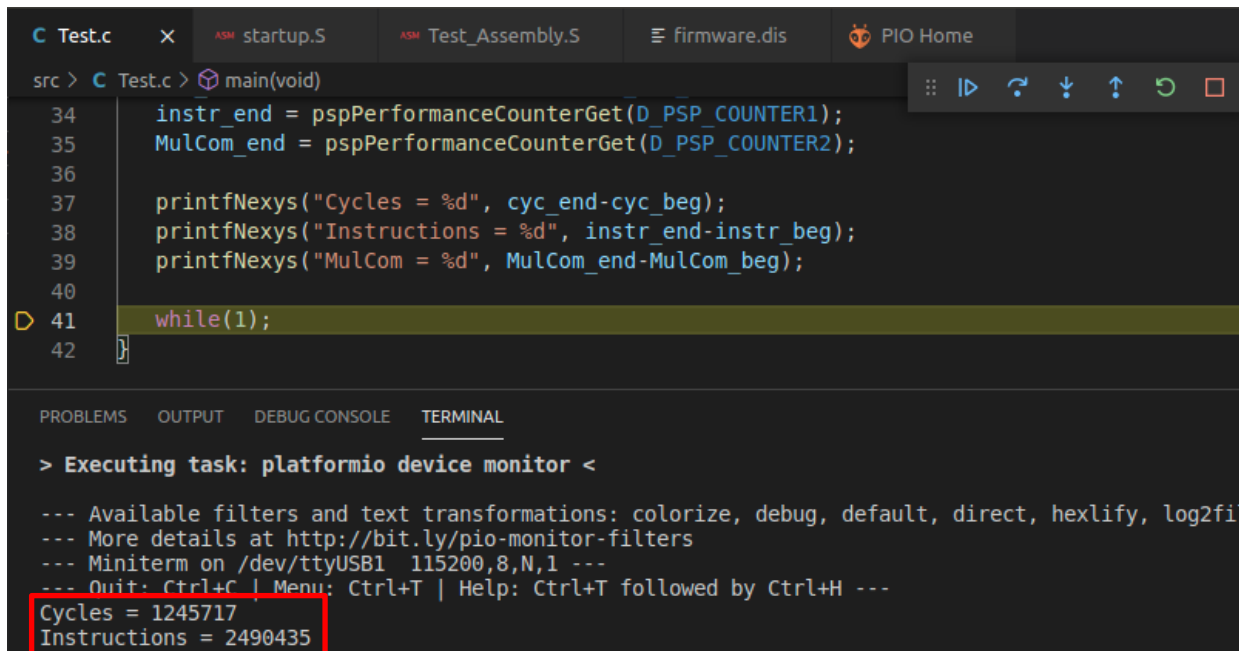
不提供解答。

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。

不提供解答。

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？證明您的答案。
 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。

非阻塞載入：



```

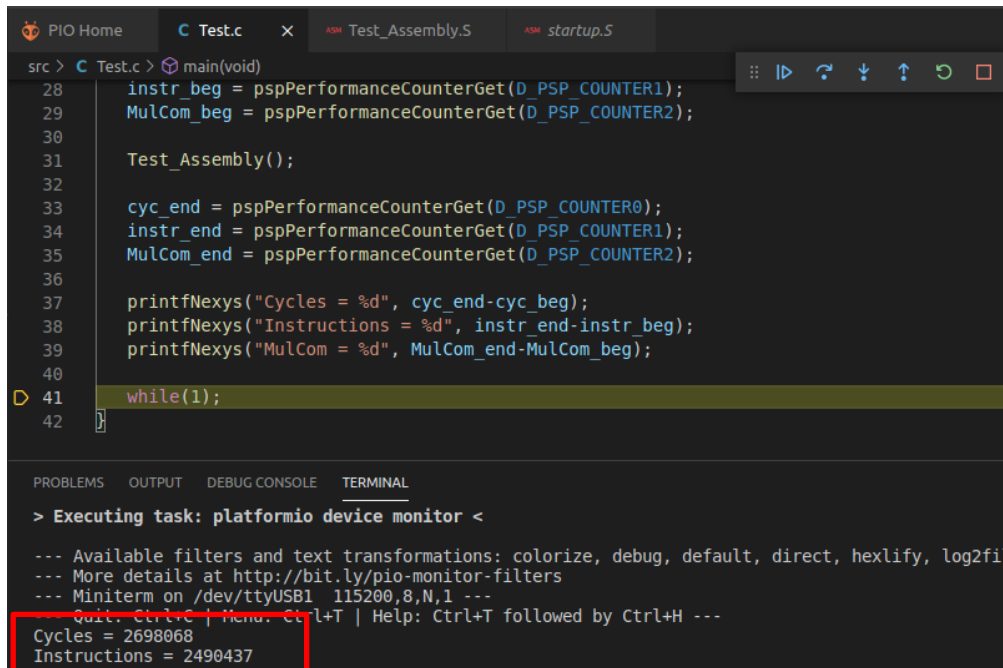
C Test.c x ASM startup.S ASM Test_Assembly.S firmware.dis PIO Home
src > C Test.c > main(void)
34 instr_end = pspPerformanceCounterGet(D_PSP_COUNTER1);
35 MulCom_end = pspPerformanceCounterGet(D_PSP_COUNTER2);
36
37 printfNexys("Cycles = %d", cyc_end-cyc_beg);
38 printfNexys("Instructions = %d", instr_end-instr_beg);
39 printfNexys("MulCom = %d", MulCom_end-MulCom_beg);
40
41 while(1);
42

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 = 1245717
Instructions = 2490435
  
```

由於非阻塞載入，獲得的IPC（ $IPC = 2490 / 1245 = 2$ ）為理想IPC。

阻塞載入：



The screenshot shows the PIO Home IDE with a C file named `Test.c` open. The code defines performance counters and a `while(1);` loop. The terminal output shows the execution results:

```
src > C Test.c > main(void)
28 instr_beg = pspPerformanceCounterGet(D_PSP_COUNTER1);
29 MulCom_beg = pspPerformanceCounterGet(D_PSP_COUNTER2);
30
31 Test_Assembly();
32
33 cyc_end = pspPerformanceCounterGet(D_PSP_COUNTER0);
34 instr_end = pspPerformanceCounterGet(D_PSP_COUNTER1);
35 MulCom_end = pspPerformanceCounterGet(D_PSP_COUNTER2);
36
37 printfNexys("Cycles = %d", cyc_end-cyc_beg);
38 printfNexys("Instructions = %d", instr_end-instr_beg);
39 printfNexys("MulCom = %d", MulCom_end-MulCom_beg);
40
41 while(1);
42

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

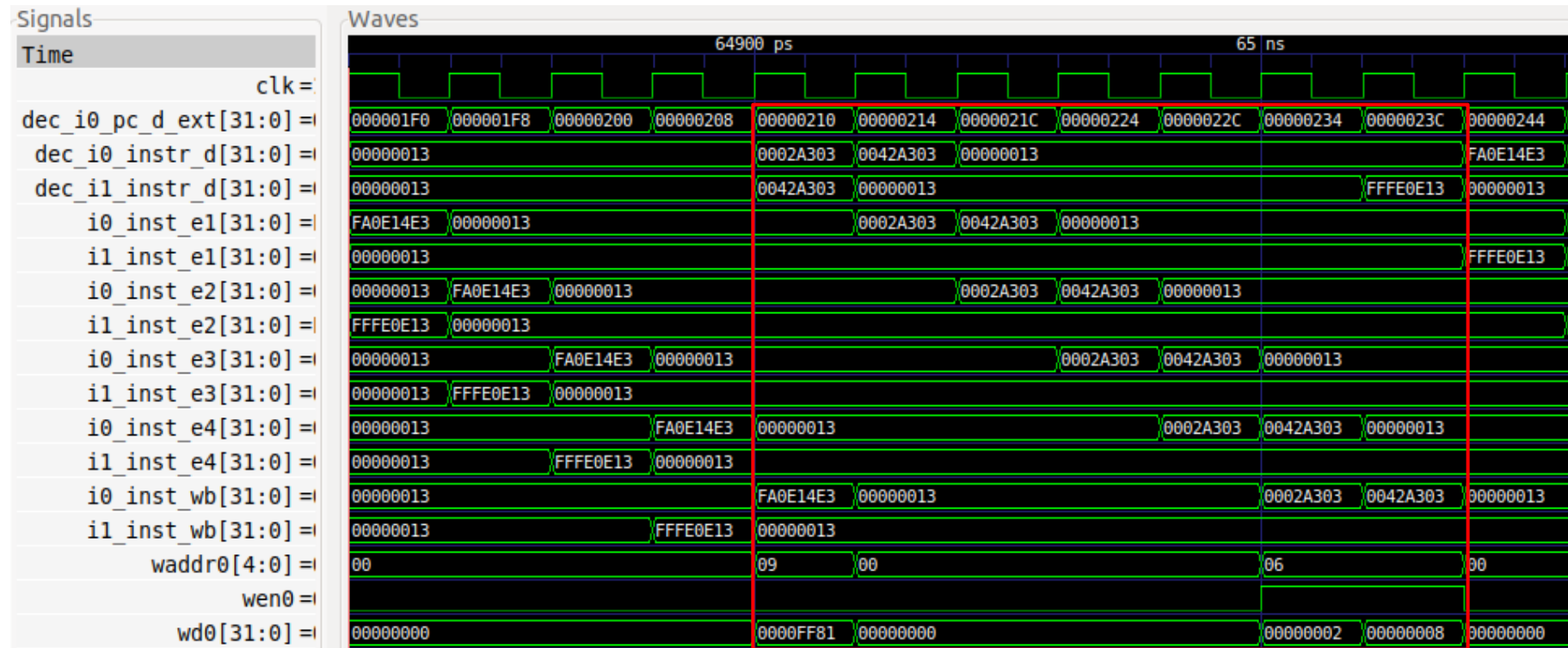
--- Available filters and text transformations: colorize, debug, default, direct, hexlify, log2fi
--- More details at http://bit.ly/pio-monitor-filters
--- Miniterm on /dev/ttyUSB1 115200,8,N,1 ---
--- Quit: Ctrl+C | Menu: Ctrl+T | Help: Ctrl+T followed by Ctrl+H ---
Cycles = 2698068
Instructions = 2490437
```

指令數相同，但現在需要更多的週期執行迴圈，因為載入使後續指令暫停以等待來自記憶體的资料。模擬更清楚地說明了這一點。

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。測試非阻塞載入和阻塞載入。可以使用以下位置提供的PlatformIO專案：
[\[RVfpgaPath\]/RVfpga/Labs/Lab14/TwoConsecutiveLW_Instructions](#)。

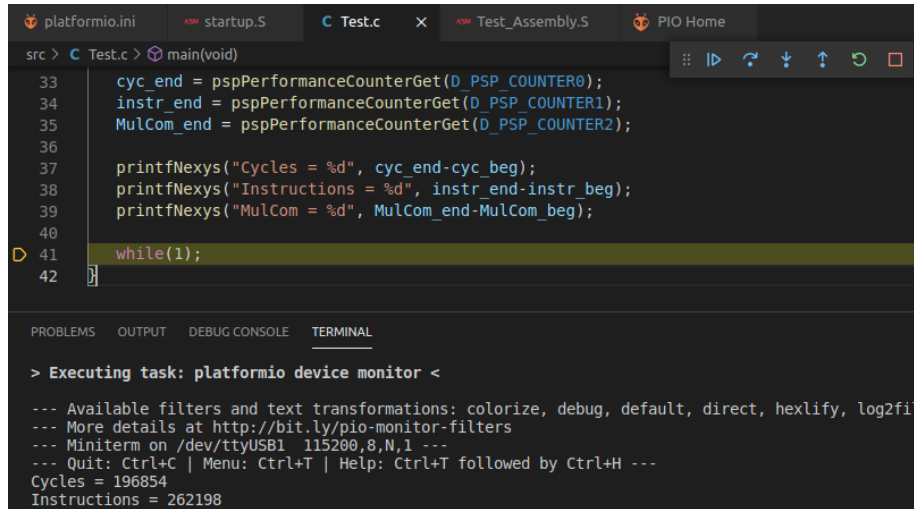
```
210: 0002a303          lw      t1,0(t0)
214: 0042a303          lw      t1,4(t0)
```

- 模擬：



由於L/S管道中的結構冒險，第二條lw必須暫停1個週期，類似於乘法管道處理兩條連續mul指令的情形。

- 在開發板上執行：



```

src > C Test.c > main(void)
33   cyc_end = pspPerformanceCounterGet(D_PSP_COUNTER0);
34   instr_end = pspPerformanceCounterGet(D_PSP_COUNTER1);
35   MulCom_end = pspPerformanceCounterGet(D_PSP_COUNTER2);
36
37   printfNexys("Cycles = %d", cyc_end-cyc_beg);
38   printfNexys("Instructions = %d", instr_end-instr_beg);
39   printfNexys("MulCom = %d", MulCom_end-MulCom_beg);
40
41   while(1);
42
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 = 196854
Instructions = 262198

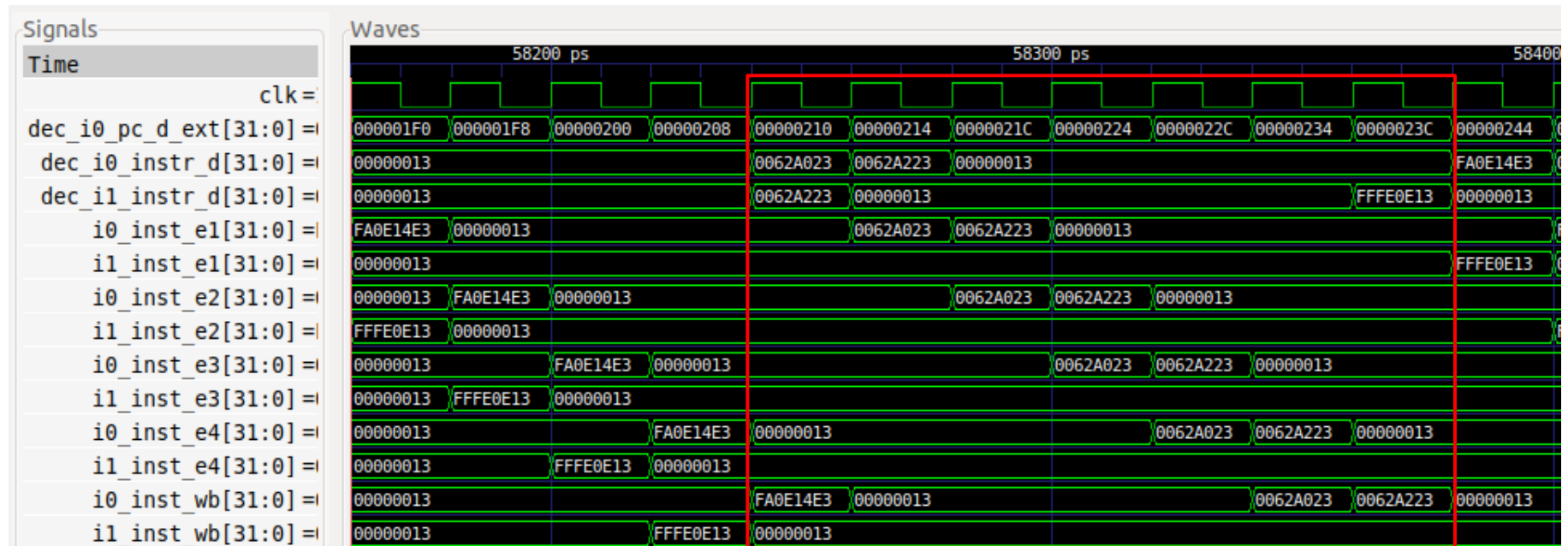
```

$$\text{IPC} = 262 / 196 = 1.33$$

兩次連續儲存：

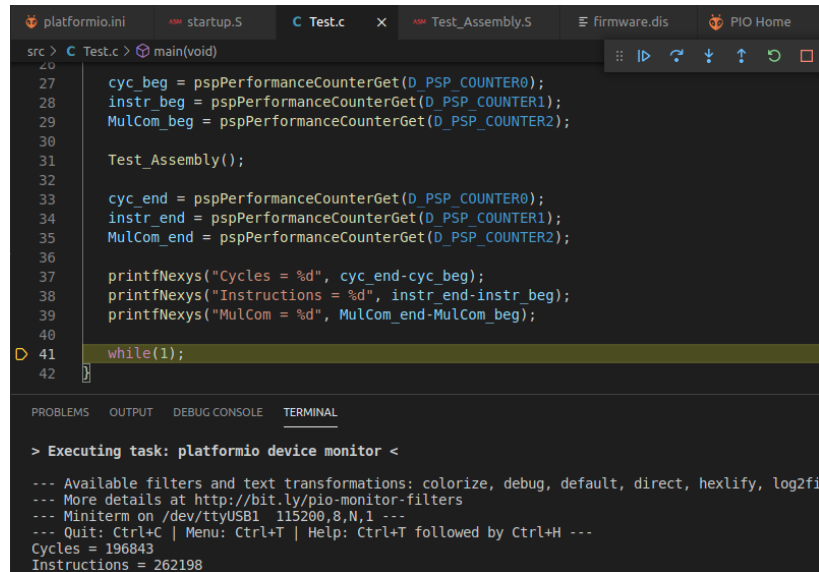
210:	0062a023	sw	t1,0(t0)
214:	0062a223	sw	t1,4(t0)

- 模擬：



由於L/S管道中的結構冒險，第二條sw必須暫停1個週期，類似於乘法管道處理兩條連續mul指令的情形。

- 在開發板上執行：



```

src > C Test.c > main(void)
27   cyc_beg = pspPerformanceCounterGet(D_PSP_COUNTER0);
28   instr_beg = pspPerformanceCounterGet(D_PSP_COUNTER1);
29   MulCom_beg = pspPerformanceCounterGet(D_PSP_COUNTER2);
30
31   Test_Assembly();
32
33   cyc_end = pspPerformanceCounterGet(D_PSP_COUNTER0);
34   instr_end = pspPerformanceCounterGet(D_PSP_COUNTER1);
35   MulCom_end = pspPerformanceCounterGet(D_PSP_COUNTER2);
36
37   printfNexys("Cycles = %d", cyc_end-cyc_beg);
38   printfNexys("Instructions = %d", instr_end-instr_beg);
39   printfNexys("MulCom = %d", MulCom_end-MulCom_beg);
40
41   while(1);
42

```

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 = 196843
 Instructions = 262198

$$\text{IPC} = 262 / 196 = 1.33$$

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

請看下面的RISC-V組合語言片段：

```

sd x29, 12(x16)
ld 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。

- 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此結構冒險引起的暫停/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。

不提供解答。

附錄A - 解碼階段兩條同時進行的DIV指令

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：查看通過各管線階段的指令流，分析控制位元（根據實驗11的附錄D，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。

不提供解答。

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。

不提供解答。

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

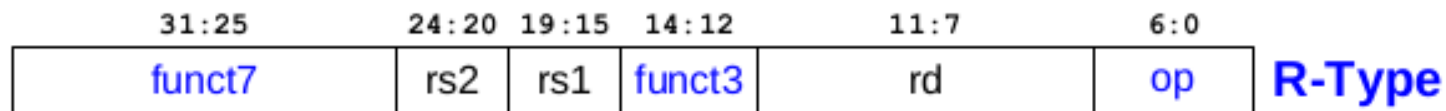
0x03de42b3 → 0000001 11101 11100 100 00101 0110011

funct7 = 0000001
rs2 = 11101 = x29 (t4)
rs1 = 11100 = x28 (t3)
funct3 = 100
rd = 00101 = x5 (t0)
op = 0110011

0x03ff4333 → 0000001 11111 11110 100 00110 0110011

funct7 = 0000001
 rs2 = 11111 = x31 (t6)
 rs1 = 11110 = x30 (t5)
 funct3 = 100
 rd = 00110 = x6 (t1)
 op = 0110011

來自DDCARV的附錄B：



op	funct3	funct7	Type	Instruction	Description	Operation
0110011 (51)	100	0000001	R	div rd, rs1, rs2	divide (signed)	rd = rs1 / rs2

Name	Register Number	Use
zero	x0	Constant value 0
ra	x1	Return address
sp	x2	Stack pointer
gp	x3	Global pointer
tp	x4	Thread pointer
t0-2	x5-7	Temporary variables
s0/fp	x8	Saved variable / Frame pointer
s1	x9	Saved variable
a0-1	x10-11	Function arguments / Return values
a2-7	x12-17	Function arguments
s2-11	x18-27	Saved variables
t3-6	x28-31	Temporary variables

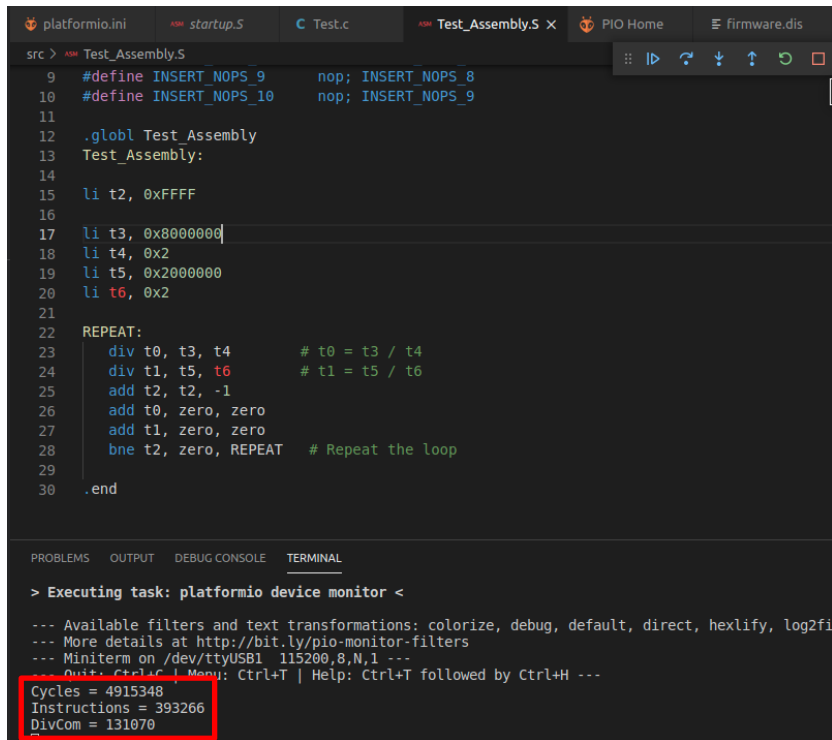
任務：在自己的電腦上重複圖9中的模擬過程，以進行詳細的分析。

解答請參見實驗14的主文件。

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。

不提供解答。

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。



```

src > Test_Assembly.S
9  #define INSERT_NOPs_9    nop; INSERT_NOPs_8
10 #define INSERT_NOPs_10   nop; INSERT_NOPs_9
11
12 .globl Test_Assembly
13 Test_Assembly:
14
15 li t2, 0xFFFF
16
17 li t3, 0x8000000
18 li t4, 0x2
19 li t5, 0x2000000
20 li t6, 0x2
21
22 REPEAT:
23 div t0, t3, t4      # t0 = t3 / t4
24 div t1, t5, t6      # t1 = t5 / t6
25 add t2, t2, -1
26 add t0, zero, zero
27 add t1, zero, zero
28 bne t2, zero, REPEAT # Repeat the loop
29
30 .end

```

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 = 4915348
 Instructions = 393266
 DivCom = 131070

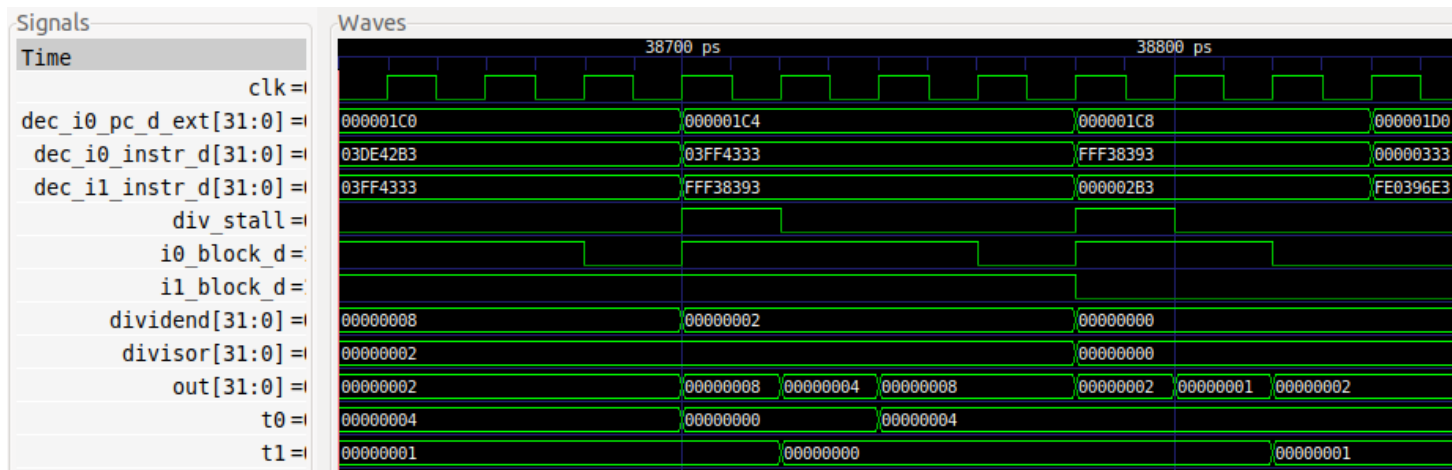
$CPI = 4910000 / 393000 = 12$. 考慮到每個除法運算大約需要34個週期來執行，而其他指令各需要 $\frac{1}{2}$ 個週期（上面是能夠預測的近似情況），近似的理論計算如下：在 $34 + 34 + \frac{1}{2} + \frac{1}{2} + \frac{1}{2} + \frac{1}{2}$ 個週期內執行6條指令 $\rightarrow CPI = 70 / 6 = 11$

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相關性。透過模擬和硬體計數器查看實驗。

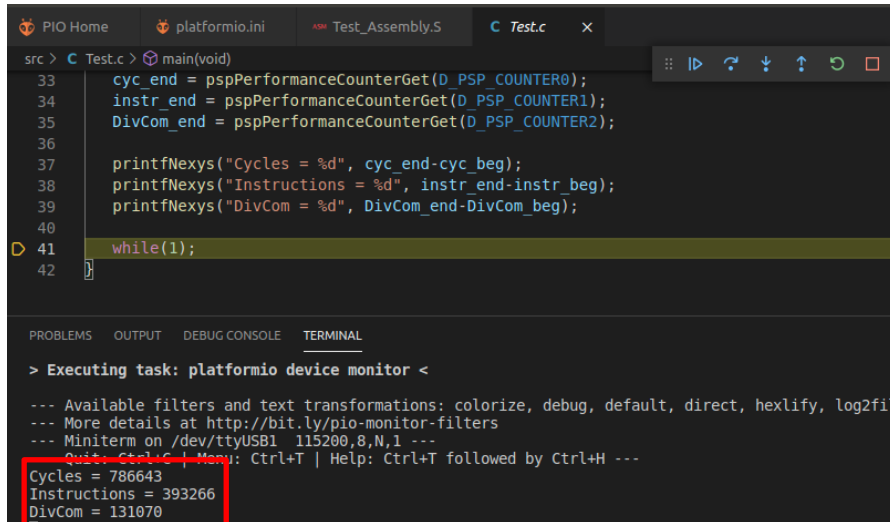
```

12 .globl Test_Assembly
13 Test_Assembly:
14
15 li t2, 0xFFFF
16
17 li t3, 0x8
18 li t4, 0x2
19 li t5, 0x2
20 li t6, 0x2
21
22 REPEAT:
23 div t0, t3, t4      # t0 = t3 / t4
24 div t1, t5, t6      # t1 = t5 / t6
25 add t2, t2, -1
26 add t0, zero, zero
27 add t1, zero, zero
28 bne t2, zero, REPEAT # Repeat the loop
29
30 .end

```



現在，除法計算僅需大約5個週期。



```
src > C Test.c > main(void)
33   cyc_end = pspPerformanceCounterGet(D_PSP_COUNTER0);
34   instr_end = pspPerformanceCounterGet(D_PSP_COUNTER1);
35   DivCom_end = pspPerformanceCounterGet(D_PSP_COUNTER2);
36
37   printfNexys("Cycles = %d", cyc_end-cyc_beg);
38   printfNexys("Instructions = %d", instr_end-instr_beg);
39   printfNexys("DivCom = %d", DivCom_end-DivCom_beg);
40
41   while(1);
42
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 = 786643
Instructions = 393266
DivCom = 131070
```

鑒於計算每個除法運算的時間縮短了很多，CPI也減小了很多（每個週期約為2）。

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。

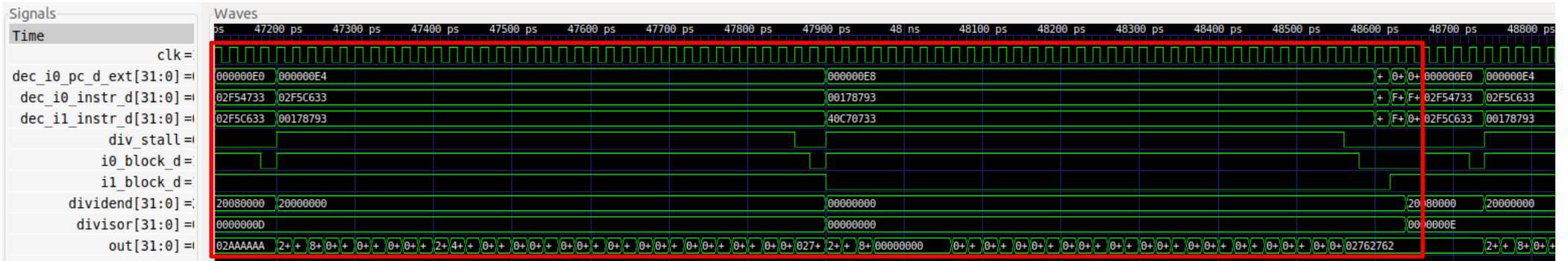
- C程式（原始和反組譯）：

```
11 int Test_C(int a, int d)
12 {
13     int b, c, e=0, i=1;
14     do {
15         b = a/i;
16         c = d/i;
17         i = i+1;
18         e = e + (b-c);
19     } while(i<65535);
20     return(e);
21 }
```

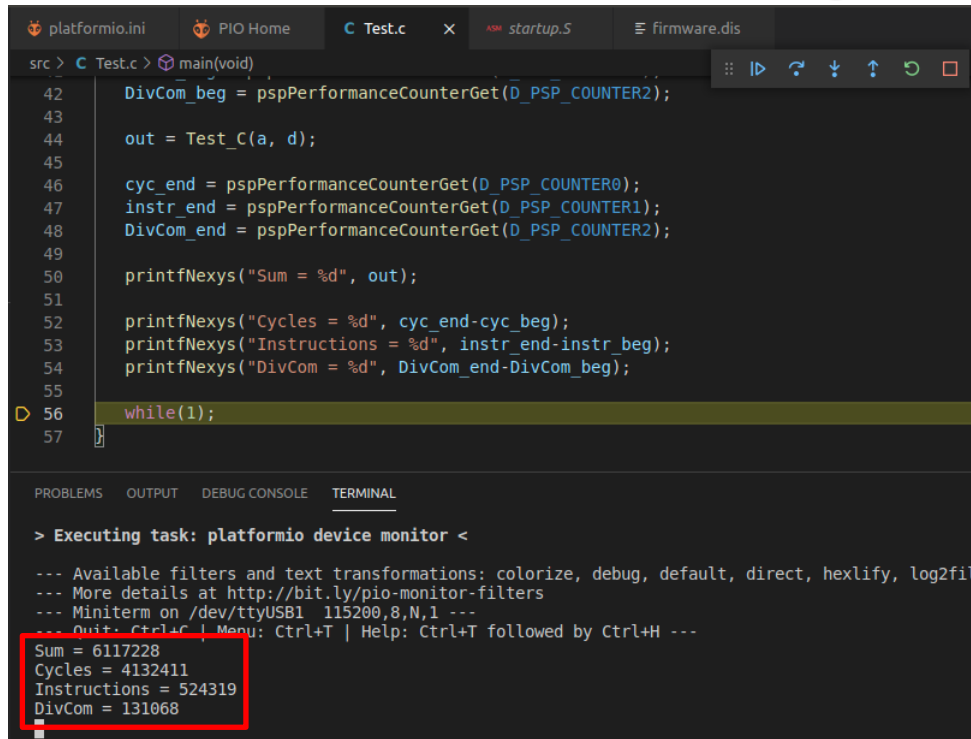
```
66 000000d8 <Test_C>:
67 d8: 00100793      li a5,1
68 dc: 00000693      li a3,0
69 e0: 02f54733      div a4,a0,a5
70 e4: 02f5c633      div a2,a1,a5
71 e8: 00178793      addi a5,a5,1
72 ec: 40c70733      sub a4,a4,a2
73 f0: 00e686b3      add a3,a3,a4
74 f4: 00010737      lui a4,0x10
75 f8: ffe70713      addi a4,a4,-2 # fffe <_sp+0xc386>
76 fc: fef752e3      bge a4,a5,e0 <Test C+0x8>
77 100: 00068513      mv a0,a3
78 104: 00008067      ret
```

Imagination
university programme

- C程式的模擬：



- 硬體計數器：



The screenshot shows a code editor with a C program in `Test.c`. The program calculates the sum of integers from 1 to 100, measures the number of cycles, instructions, and divcom operations, and prints these values. The code is as follows:

```

src > C Test.c > main(void)
42 DivCom_beg = pspPerformanceCounterGet(D_PSP_COUNTER2);
43
44 out = Test_C(a, d);
45
46 cyc_end = pspPerformanceCounterGet(D_PSP_COUNTER0);
47 instr_end = pspPerformanceCounterGet(D_PSP_COUNTER1);
48 DivCom_end = pspPerformanceCounterGet(D_PSP_COUNTER2);
49
50 printfNexys("Sum = %d", out);
51
52 printfNexys("Cycles = %d", cyc_end-cyc_beg);
53 printfNexys("Instructions = %d", instr_end-instr_beg);
54 printfNexys("DivCom = %d", DivCom_end-DivCom_beg);
55
56 while(1);
57

```

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 ---
Sum = 6117228
Cycles = 4132411
Instructions = 524319
DivCom = 131068

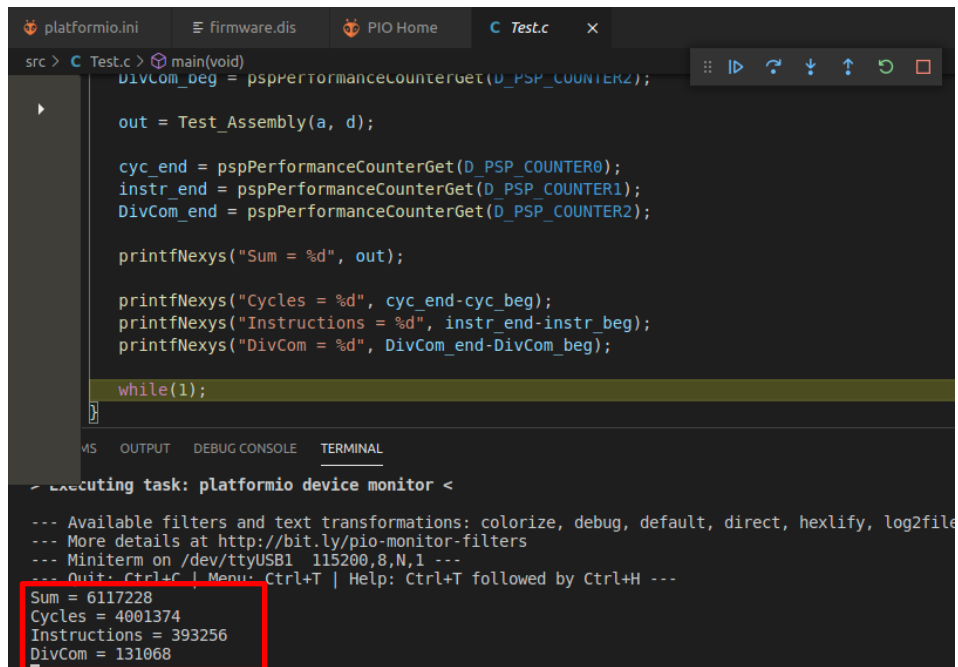
```

- Assembly 程式位於：
`[RVfpgaPath]/RVfpga/Labs/RVfpgaLabsSolutions/Programs_Solutions/Lab14/DIV_Instr_Accumul_Assembly`


```

128 000001c8 <Test_Assembly>:
129 1c8: 00100393      li    t2,1
130 1cc: 00000e93      li    t4,0
131 1d0: 00000f93      li    t6,0
132 1d4: 00010637      lui   a2,0x10
133 1d8: fff60613      addi  a2,a2,-1 # ffff <_sp+0xc377>
134 1dc: 00050e33      add  t3,a0,zero
135 1e0: 00058f33      add  t5,a1,zero
136
137 000001e4 <REPEAT>:
138 1e4: 027e42b3      div  t0,t3,t2
139 1e8: 027f4333      div  t1,t5,t2
140 1ec: 00138393      addi  t2,t2,1
141 1f0: 40628eb3      sub  t4,t0,t1
142 1f4: 01df8fb3      add  t6,t6,t4
143 1f8: fec396e3      bne  t2,a2,1e4 <REPEAT>
144 1fc: 000f8533      add  a0,t6,zero
145 200: 00008067      ret

```



```

src > C Test.c > main(void)
    DivCom_beg = pspPerformanceCounterGet(D_PSP_COUNTER2);

    out = Test_Assembly(a, d);

    cyc_end = pspPerformanceCounterGet(D_PSP_COUNTER0);
    instr_end = pspPerformanceCounterGet(D_PSP_COUNTER1);
    DivCom_end = pspPerformanceCounterGet(D_PSP_COUNTER2);

    printfNexys("Sum = %d", out);

    printfNexys("Cycles = %d", cyc_end-cyc_beg);
    printfNexys("Instructions = %d", instr_end-instr_beg);
    printfNexys("DivCom = %d", DivCom_end-DivCom_beg);

    while(1);
}

MS  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 ---

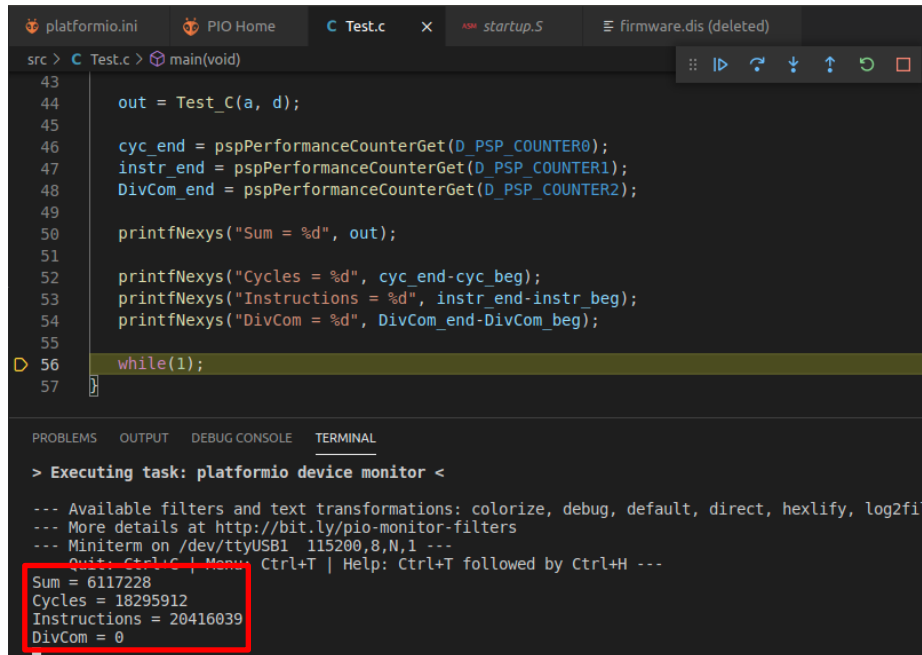
Sum = 6117228
Cycles = 4001374
Instructions = 393256
DivCom = 131068

```

求和的結果相同，因為程式相同。

週期數略少一點，因為手動編寫的組合語言版本比未經最佳化的編譯器獲得的組合語言版本效率更高。
指令數也更少一點。

- 停用M延伸功能：



The screenshot shows the PlatformIO IDE with a C program in `Test.c`. The program calculates a sum, cycles, instructions, and divcom. The output in the terminal is as follows:

```

> 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 ---
Sum = 6117228
Cycles = 18295912
Instructions = 20416039
DivCom = 0
  
```

求和的結果相同，因為程式相同。

週期數多很多：約18M與約4M。

指令數也多很多：約20M與約0.5M。

CPI現在更好。

未提交除法。

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的範例中的兩條div指令可以平行執行。

不提供解答。