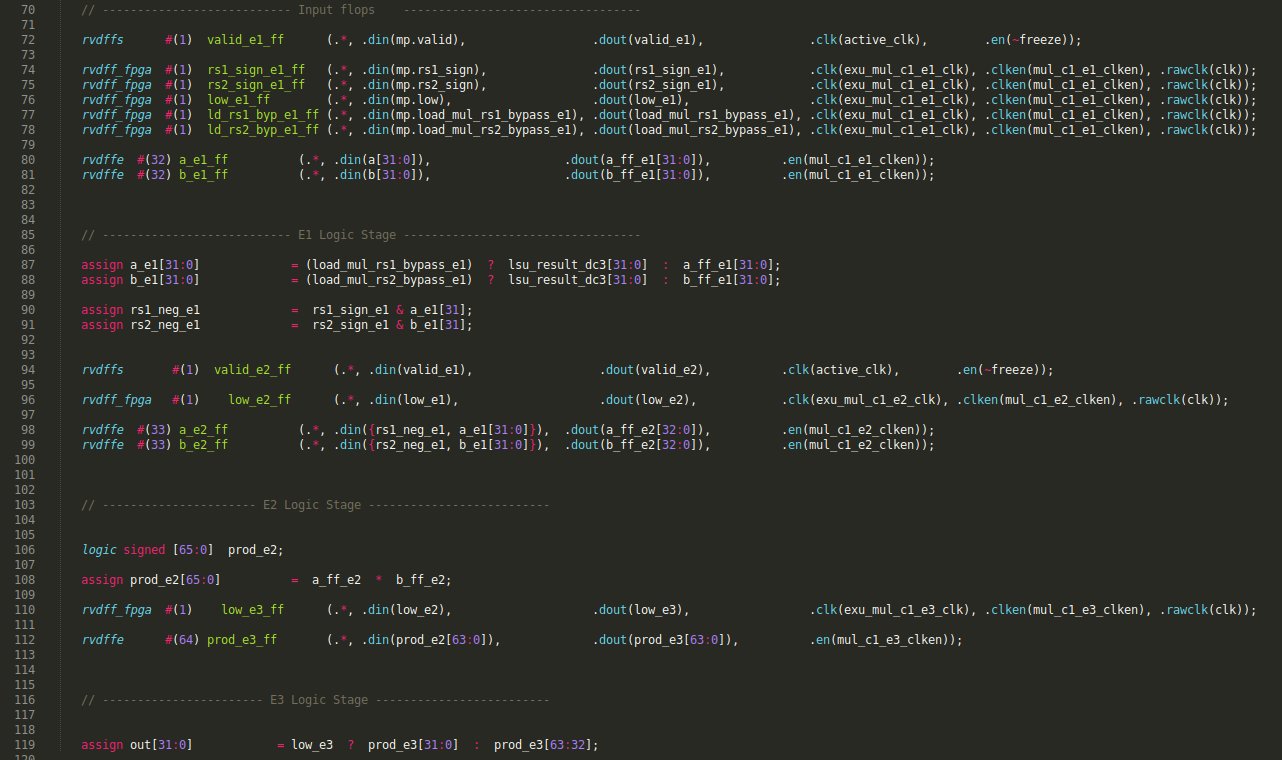
# 任務

**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。



* 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**

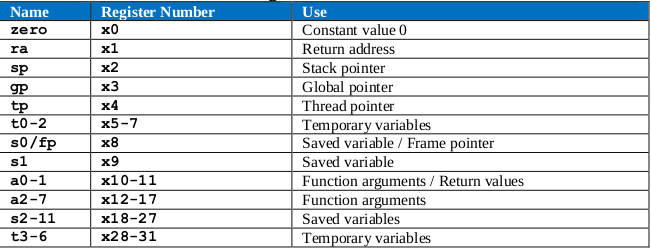
**rd = 00110 = x6 (t1)**

**op = 0110011**

來自DDCARV的附錄B：







**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

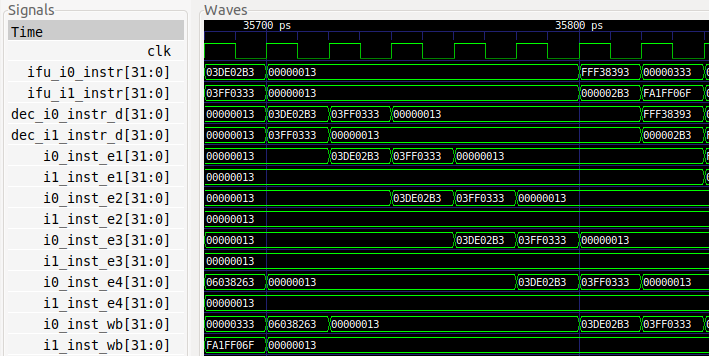
- 通路1：ifu\_i1\_instr 🡪 dec\_i1\_instr\_d

- 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*）。



**i-1**

**i+6**

**i+3**

**i+2**

**i+1**

**i**

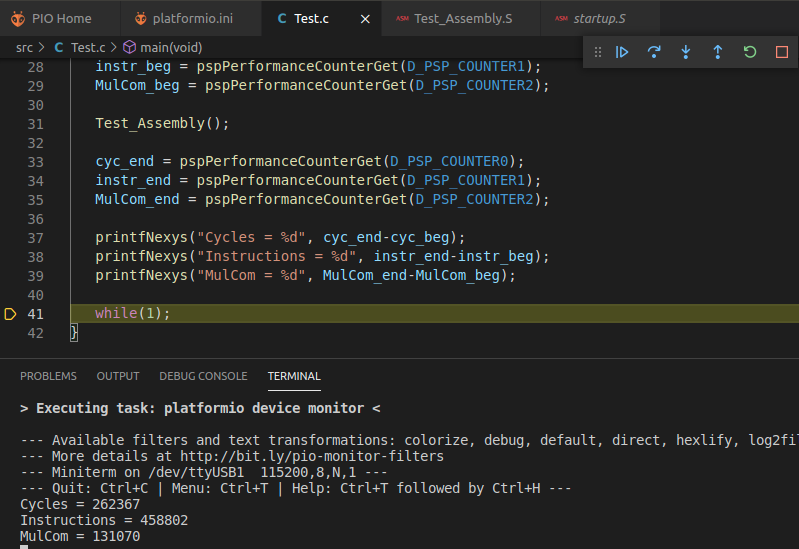
* 在週期i-1中（不顯示在圖2及圖3中），兩條mul指令位於對齊階段：在模組ifu\_aln\_ctl上，第一條分配給通路 0（ifu\_i0\_instr = 0x03de02b3），第二條分配給通路1（ifu\_i1\_instr = 0x03ff0333）。
* 在週期i中，兩條指令已傳播到模組dec\_ib\_ctl的解碼階段：第一條指令在通路0中繼續執行（dec\_i0\_instr\_d = 0x03de02b3），第二條指令在通路1中繼續執行（dec\_i1\_instr\_d = 0x03ff0333）。
* 在週期i+1中，第一條mul指令已傳播到dec\_decode\_ctl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。

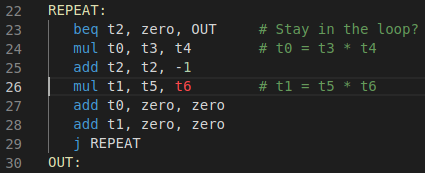
**任務：**刪除迴圈中包含的nop指令並使用SweRV EH1中提供的效能計數器測量不同的事件（執行週期、已提交的指令/乘法數等），如實驗11中所述。在分析圖2中的模擬後，週期數是否符合預期？證明您的答案。

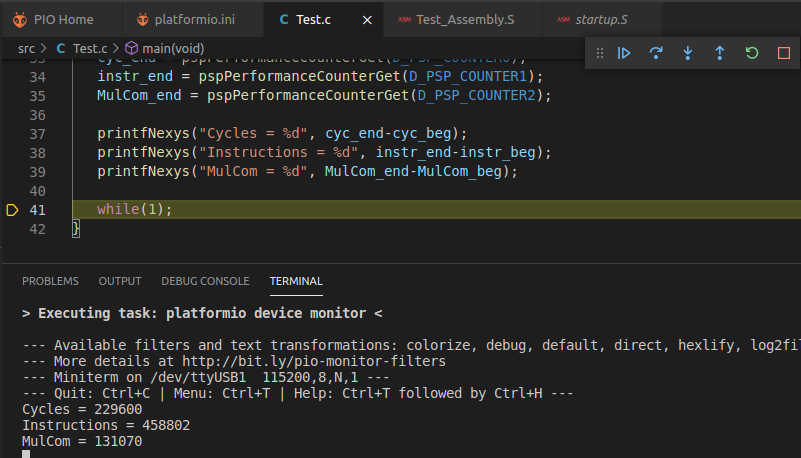
現在調整迴圈內的程式碼順序，以達到理想的吞吐量。解釋在原始程式碼和調整順序的程式碼中取得的結果。



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。





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：

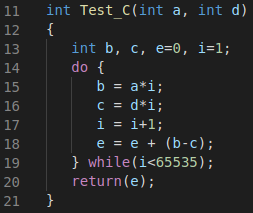
build\_flags = -Wa,-march=rv32**ima** -march=rv32**ima**

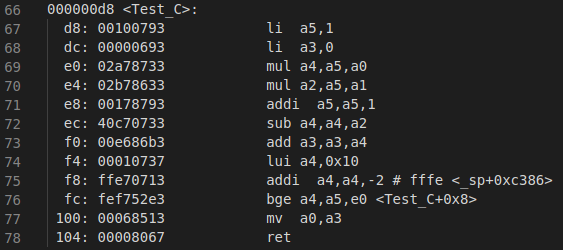
修改為：

build\_flags = -Wa,-march=rv32**ia** -march=rv32**ia**

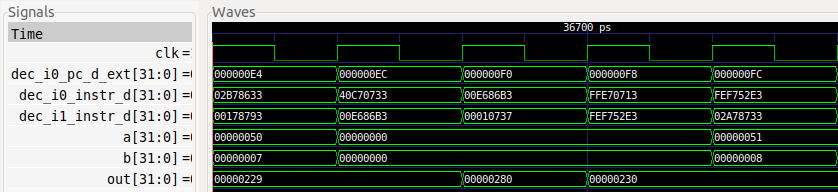
這樣便可避免使用M RISC-V延伸功能中的指令，而是使用其他指令進行模擬。

* C程式（原始和反組譯）：

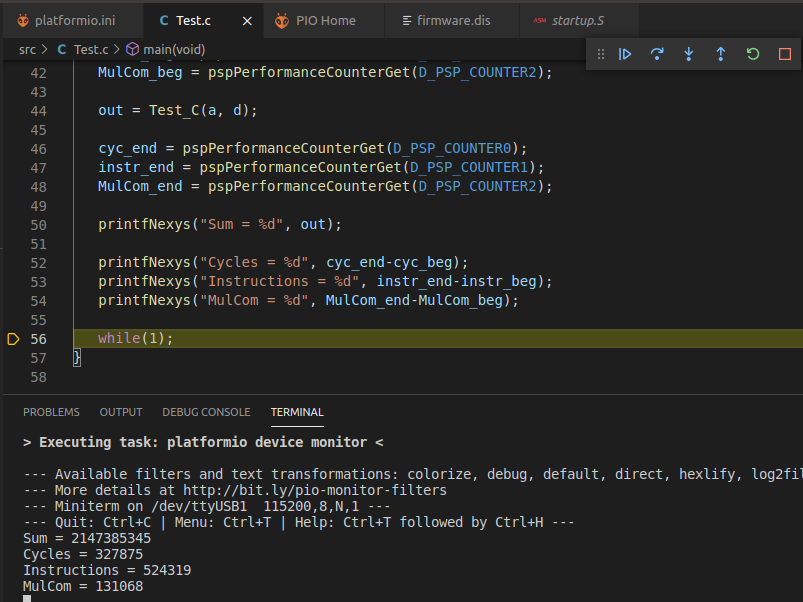




* C程式的模擬：



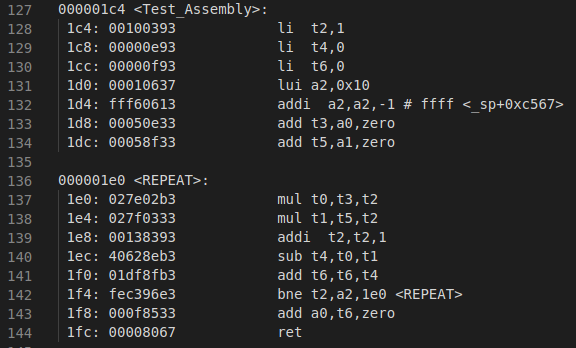
* 硬體計數器：

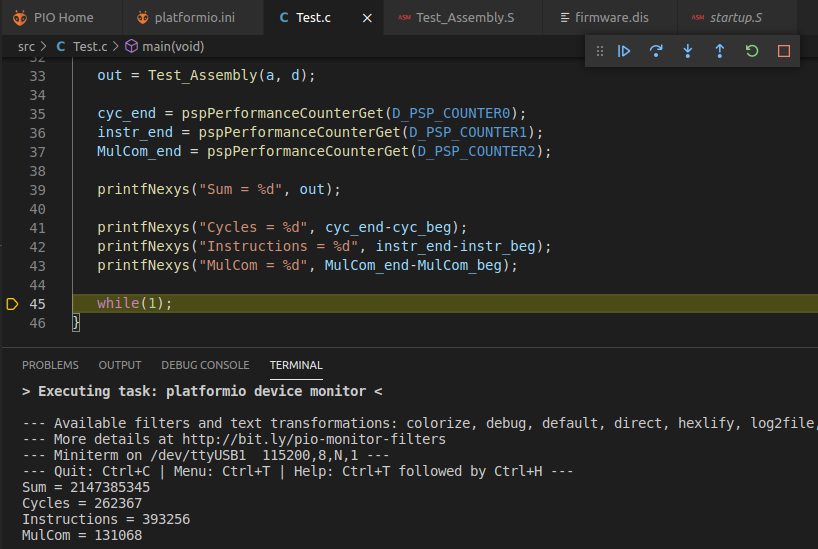


IPC = 524000 / 327000 = 1.6。由於實驗15中將分析的RAW資料冒險，因此遺失了一些週期。

* Assembly 程式位於：

*[RVfpgaPath]/RVfpga/Labs/RVfpgaLabsSolutions/Programs\_Solutions/Lab14/MUL\_Instr\_Accumul\_Assembly*



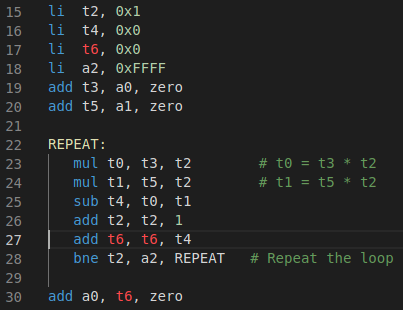


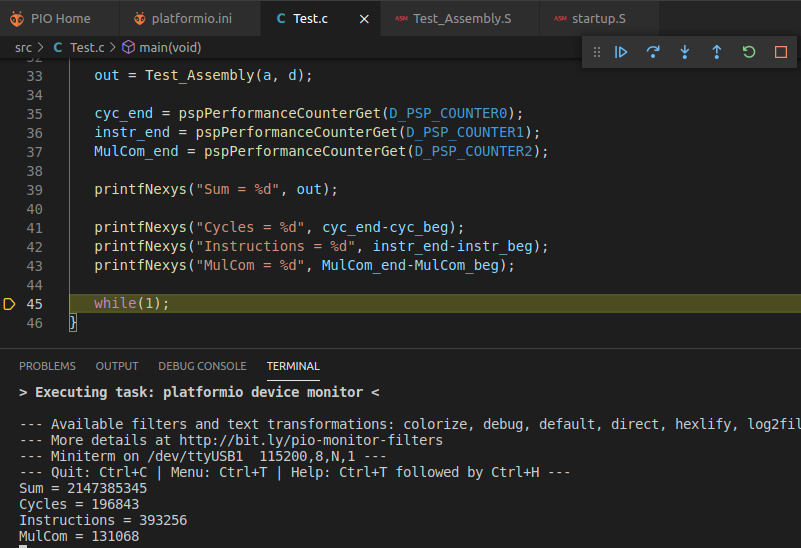
求和的結果相同，因為程式相同。

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：





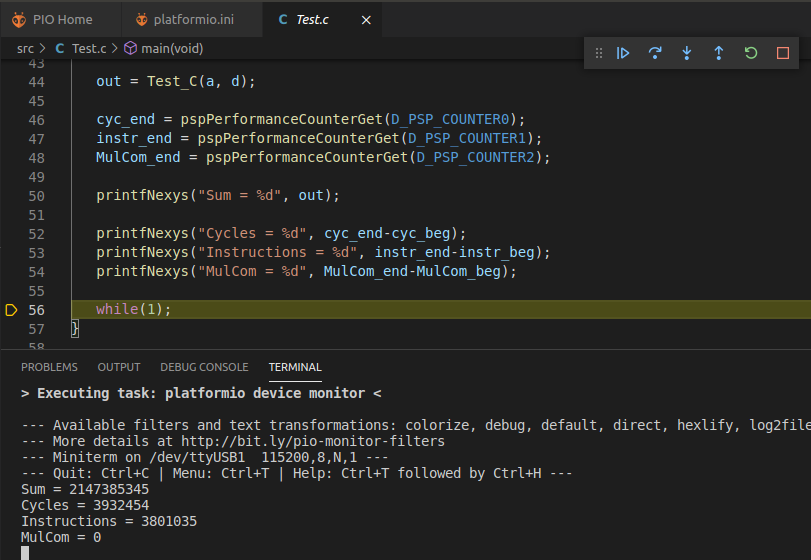
求和的結果相同，因為程式相同。

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：



求和的結果相同，因為程式相同。

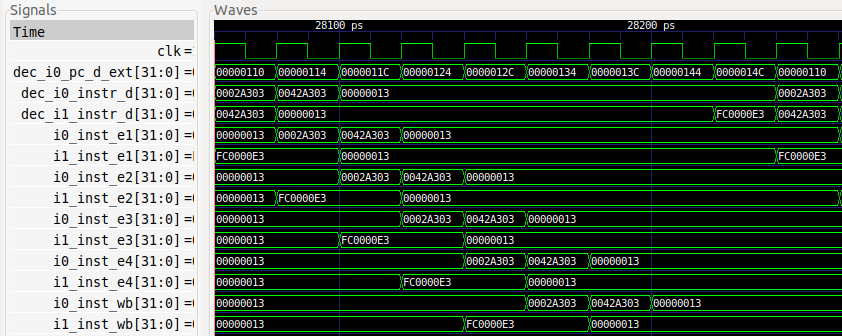
週期數多很多：約4M與約0.3M。

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。

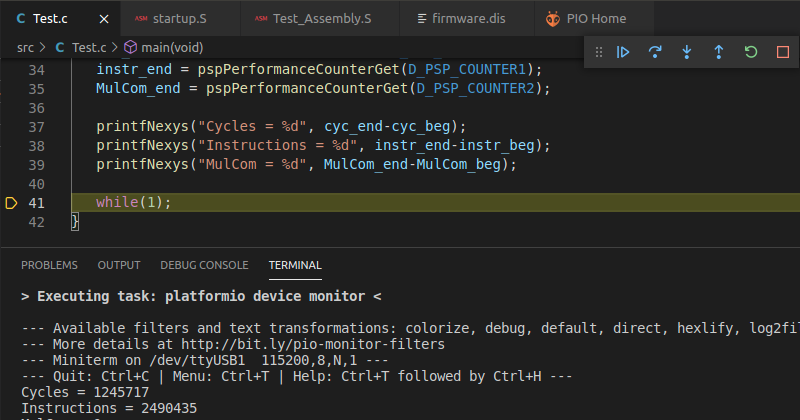
**任務：**將圖7中的說明與已複製到電腦上的圖6中的模擬進行比較。根據需要新增訊號以擴展模擬並加深理解。

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。

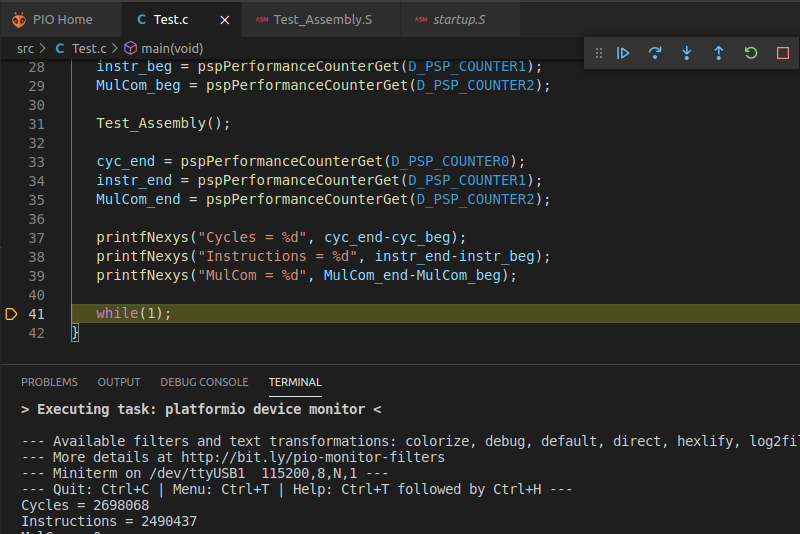
將這些結果與載入設定為阻塞載入時取得的結果進行比較。

**非阻塞載入：**

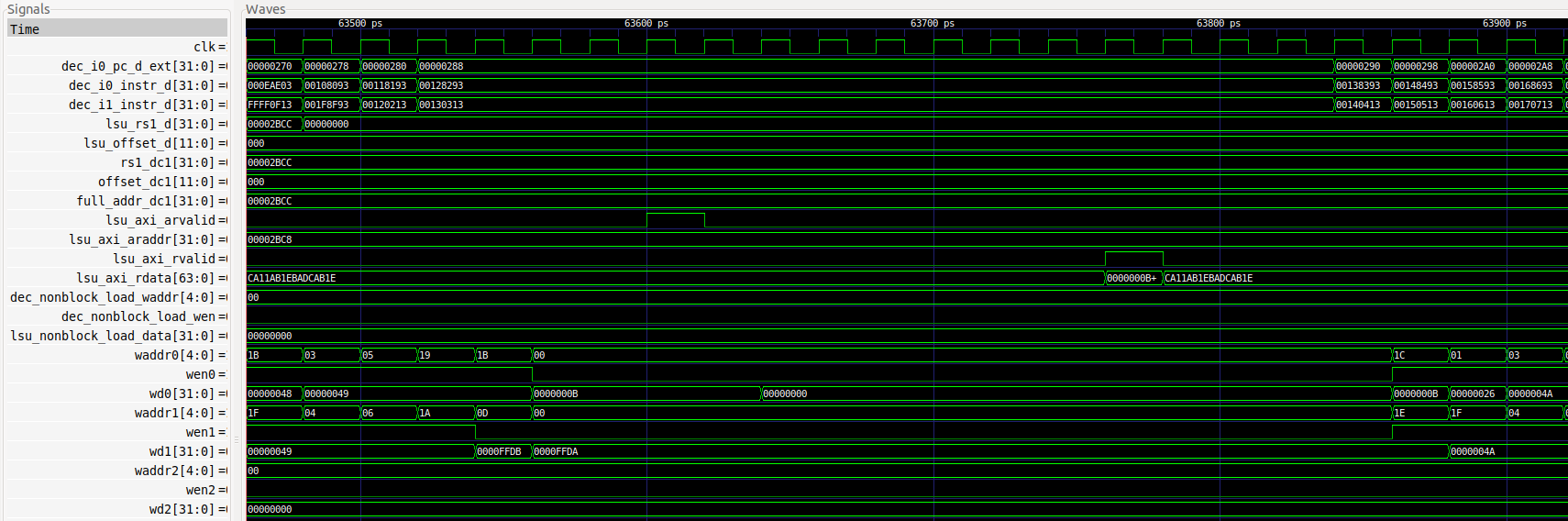


由於非阻塞載入，獲得的IPC（IPC = 2490 / 1245 = 2）為理想IPC。

**阻塞載入：**



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資料。模擬更清楚地說明了這一點。



**管線停止**

# 練習

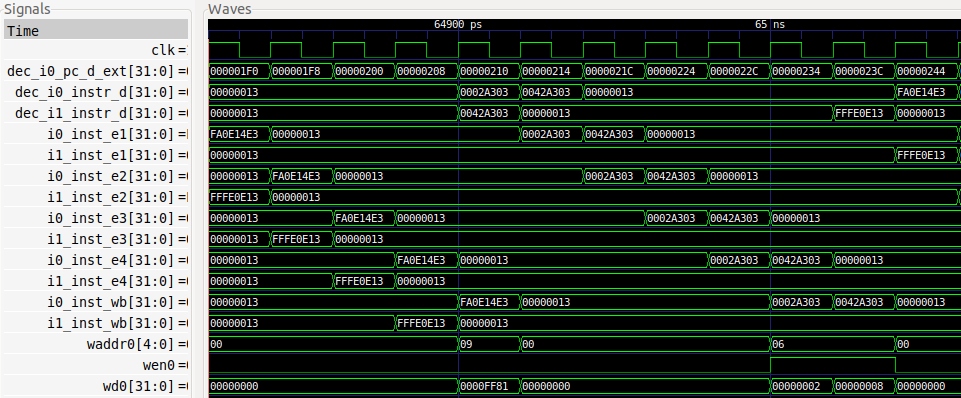
1. 在模擬中以及在開發板上分析在同一週期到達L/S管道的兩條連續記憶體存取指令（可以分析載入和儲存等兩條連續記憶體存取指令的任意組合）之間發生的結構冒險。測試非阻塞載入和阻塞載入。可以使用以下位置提供的PlatformIO專案：*[RVfpgaPath]/RVfpga/Labs/Lab14/TwoConsecutiveLW\_Instructions*。

兩次連續載入：

210: 0002a303 lw t1,0(t0)

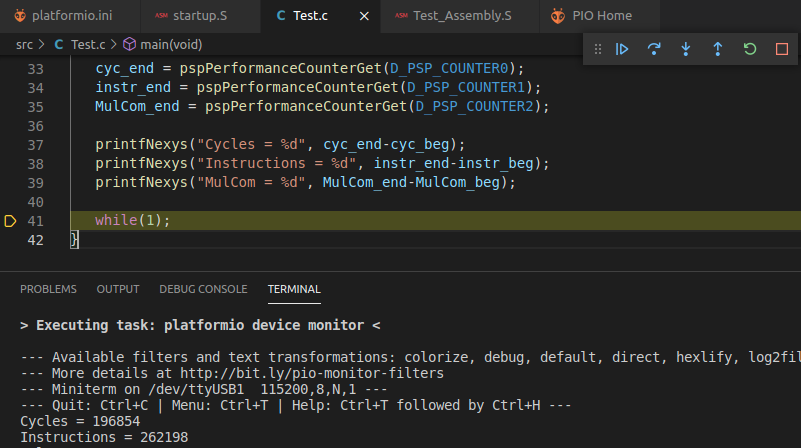
214: 0042a303 lw t1,4(t0)

* 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：



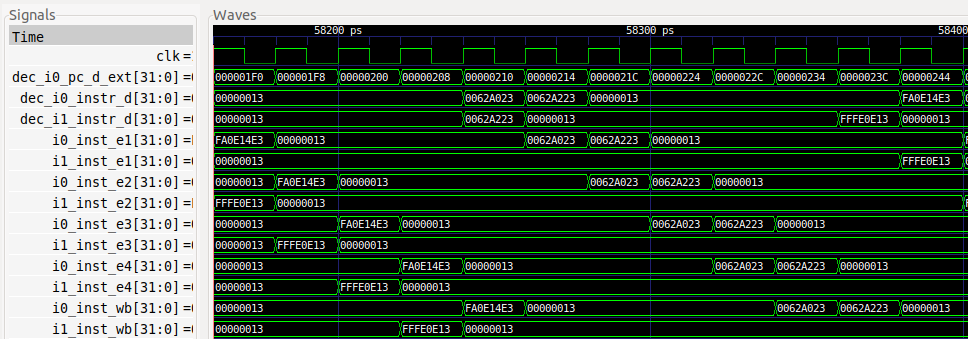
IPC = 262 / 196 = 1.33

兩次連續儲存：

210: 0062a023 sw t1,0(t0)

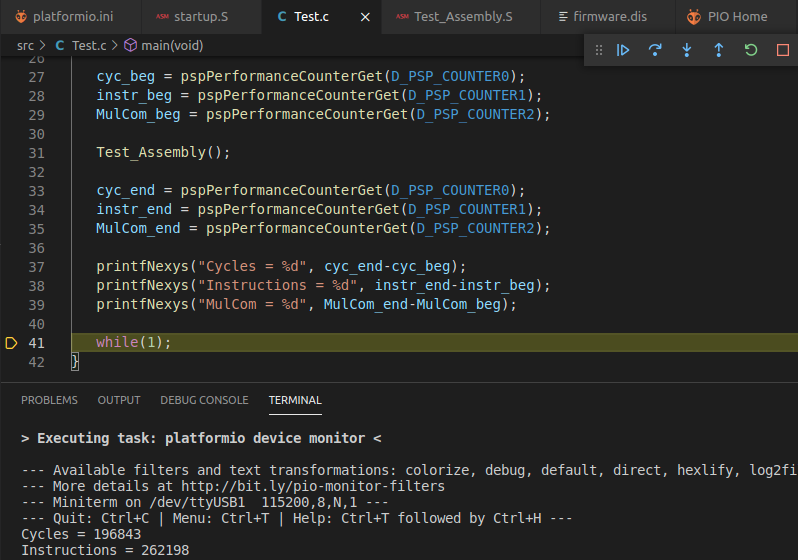
214: 0062a223 sw t1,4(t0)

* 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：



IPC = 262 / 196 = 1.33

1. （*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。

* 1. 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。
  2. 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？
  3. 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**

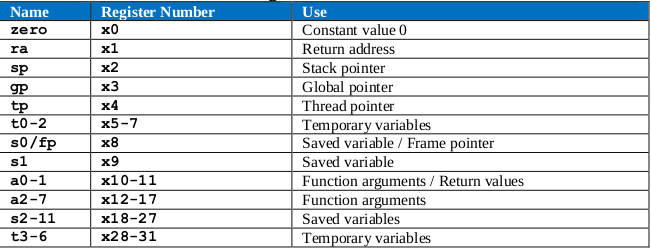
**rd = 00110 = x6 (t1)**

**op = 0110011**

來自DDCARV的附錄B：







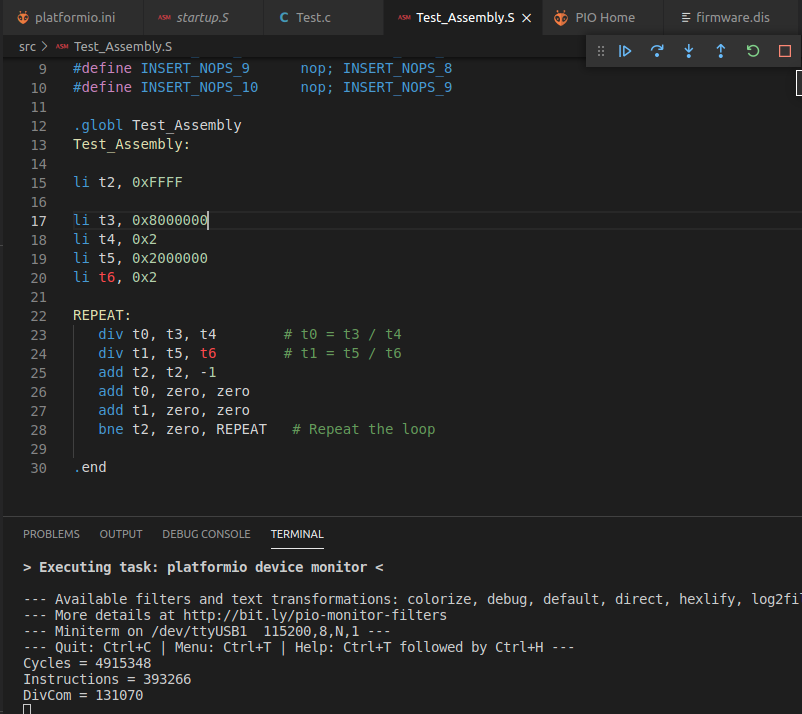
**任務：**在自己的電腦上重複圖9中的模擬過程，以進行詳細的分析。

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。

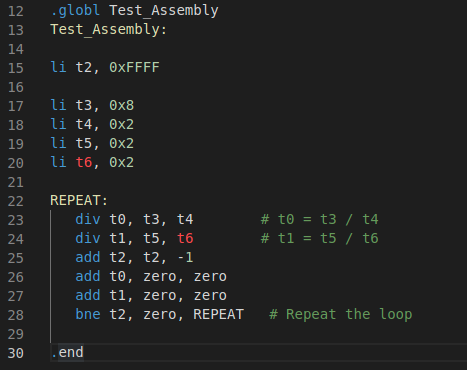
不提供解答。

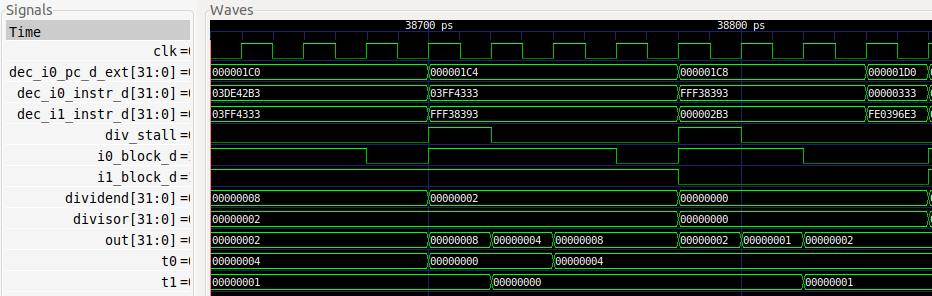
**任務：**使用SweRV EH1中提供的效能計數器測量不同的事件（週期、已提交的指令/除法等），如實驗11中所述。在分析圖9中的模擬後，週期數是否符合預期？證明您的答案。



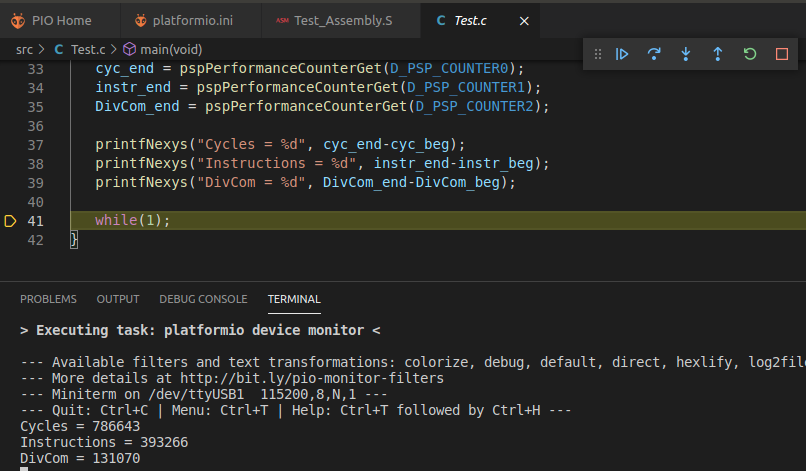
CPI = 4910000 / 393000 = 12.考慮到每個除法運算大約需要34個週期來執行，而其他指令各需要½個週期（上面是能夠預測的近似情況），近似的理論計算如下：在34 + 34 + ½ + ½ + ½ + ½個週期內執行6條指令 🡪 CPI = 70 / 6 = 11

**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。





現在，除法計算僅需大約5個週期。



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：

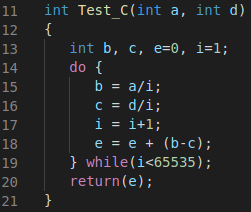
build\_flags = -Wa,-march=rv32**ima** -march=rv32**ima**

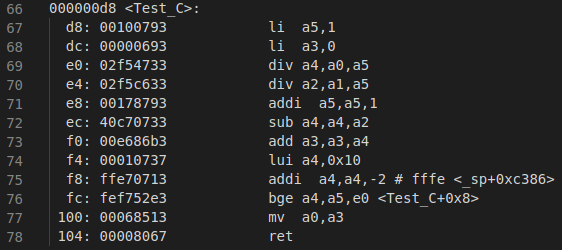
修改為：

build\_flags = -Wa,-march=rv32**ia** -march=rv32**ia**

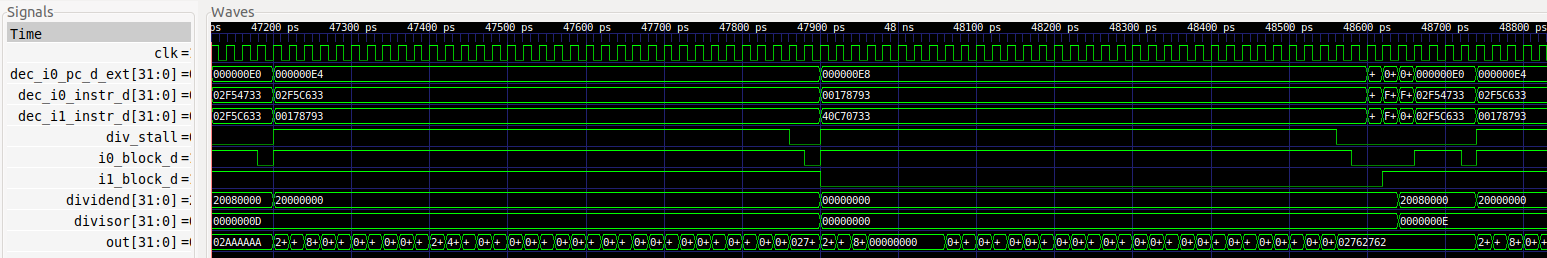
這樣便可避免使用RISC-V M延伸功能中的指令，而是使用其他指令進行模擬。

* C程式（原始和反組譯）：

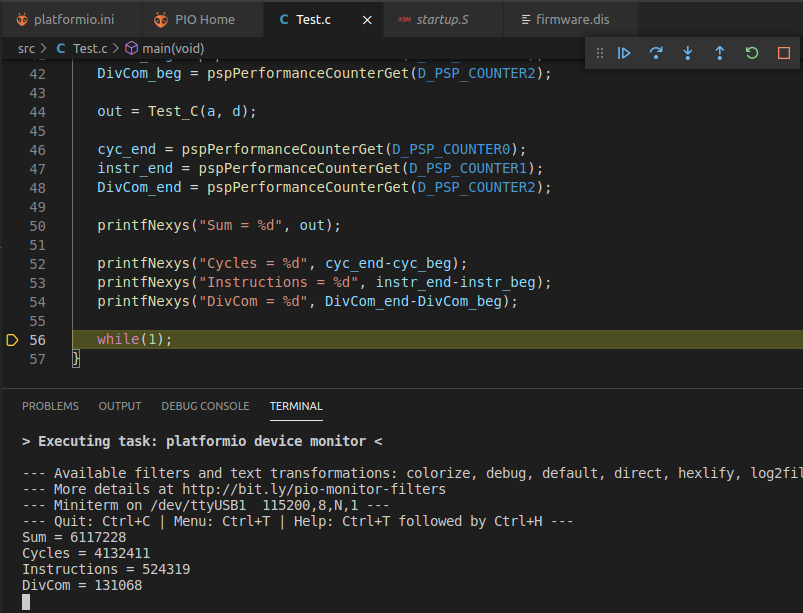




* C程式的模擬：

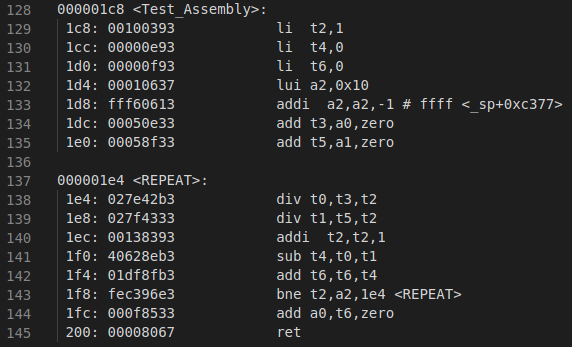


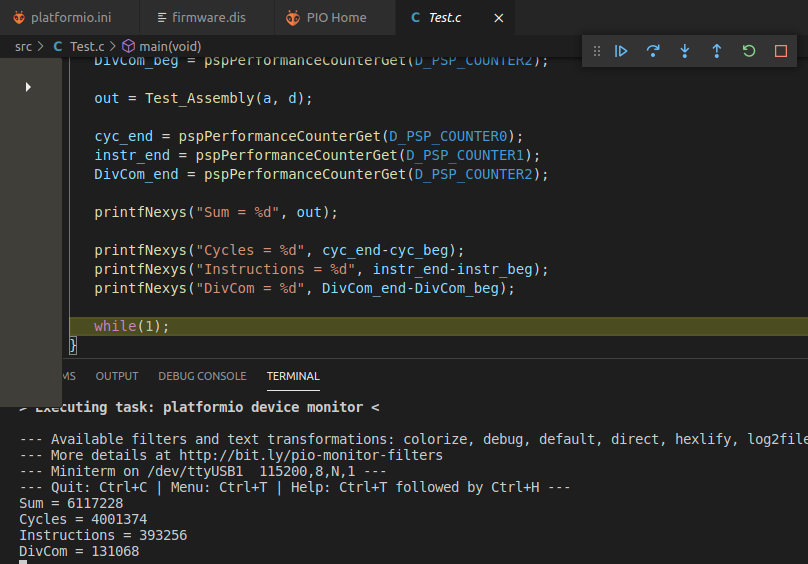
* 硬體計數器：



* Assembly 程式位於：

*[RVfpgaPath]/RVfpga/Labs/RVfpgaLabsSolutions/Programs\_Solutions/Lab14/DIV\_Instr\_Accumul\_Assembly*



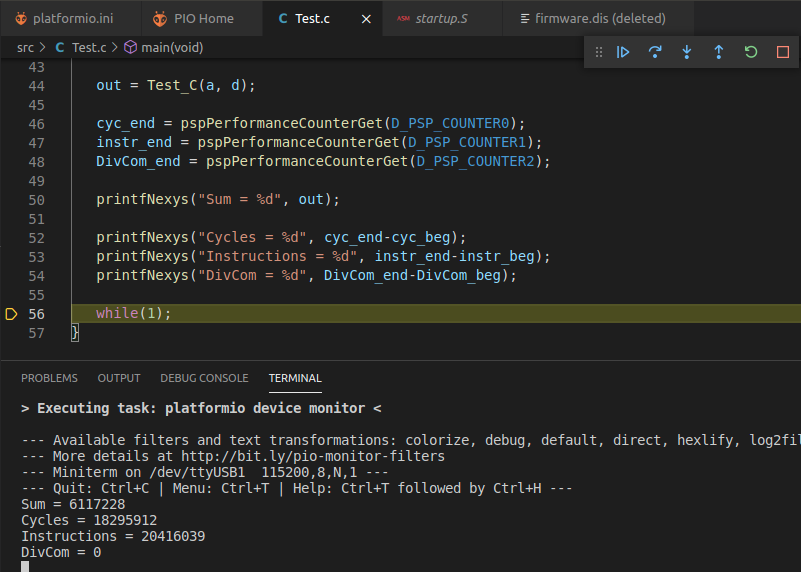


求和的結果相同，因為程式相同。

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：



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。