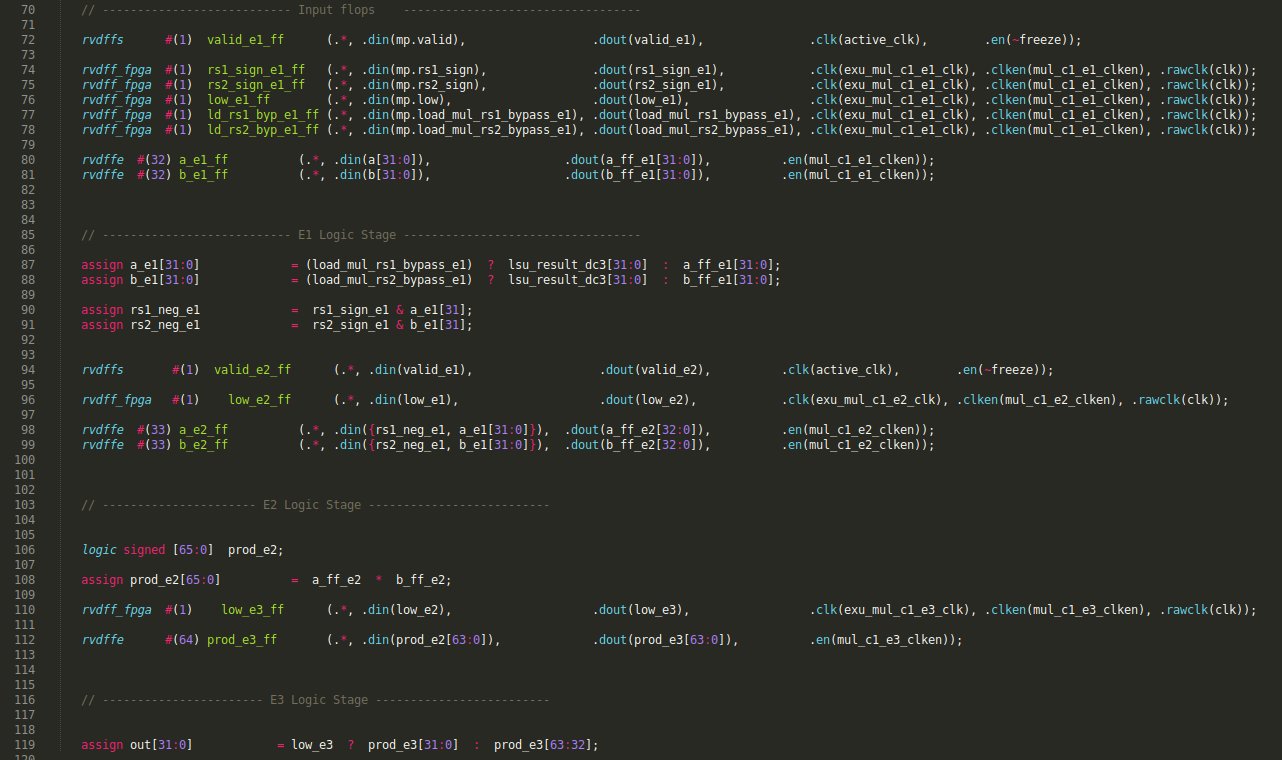
# 任务

**任务：**可以对mul指令进行与实验12中对算术-逻辑指令进行的研究类似的研究：查看通过各流水线阶段的指令流，分析控制位（根据实验11的附录D，mul指令有一个称为mul\_pkt\_t的特定结构类型，并且模块**dec\_decode\_ctl**中定义了一个名为mul\_p的信号）等。

不提供解答。

**任务：**检查来自**exu\_mul\_ctl**的Verilog代码，了解乘法如何计算。请记住，RISC-V包括4条乘法指令（mul、mulh、mulhsu和mulhu），并且所有这些指令均必须受硬件支持。

作为一项可选练习，可以用自己的乘法单元或互联网上的乘法单元来替换此乘法单元。



* 译码阶段产生的输入和控制位在第72-81行记录。

M1：

* 如果乘法和先前装载之间存在数据相关性，则在第87-88行进行转发。
* 此外，输入操作数符号的处理在第90-91行确定。请记住，RISC-V包括三个版本的“高位乘”运算：mulh、mulhsu和mulhu。
* 这些值传播到M2。

M2：

* 实际的乘法在第108行执行。

M3：

* 低/高位部分在第119行返回out[31:0]。执行mul指令时选择低位部分，而执行三条mulh指令中的任何一条时选择高位部分。

**任务：**验证这对32位值（0x03de02b3和0x03ff0333）是否对应RISC-V架构中的指令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**

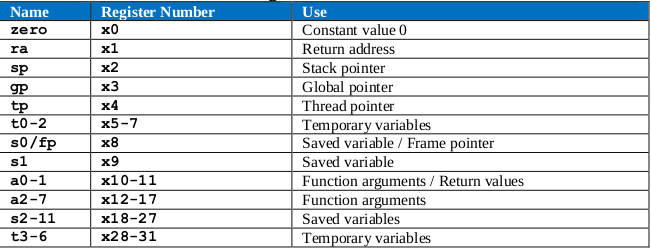
**rd = 00110 = x6 (t1)**

**op = 0110011**

来自DDCARV的附录B：







**任务：**在自己的计算机上重复图2中的仿真过程，以进行更深入的分析。

解答请参见实验14的主文档。

**任务：**将图3中的说明与图2中的仿真进行比较（重点关注两条mul指令）。具体来说，分析这两条指令如何在对齐和译码阶段分配给两个通路。

- 在模块**ifu\_aln\_ctl**（对齐阶段）中，这两条指令分配给以下信号：

- 通路0：ifu\_i0\_instr

- 通路1：ifu\_i1\_instr

- 在模块**dec\_ib\_ctl**中，这两条指令进行缓存（对齐到译码阶段）：

- 通路0：ifu\_i0\_instr 🡪 dec\_i0\_instr\_d

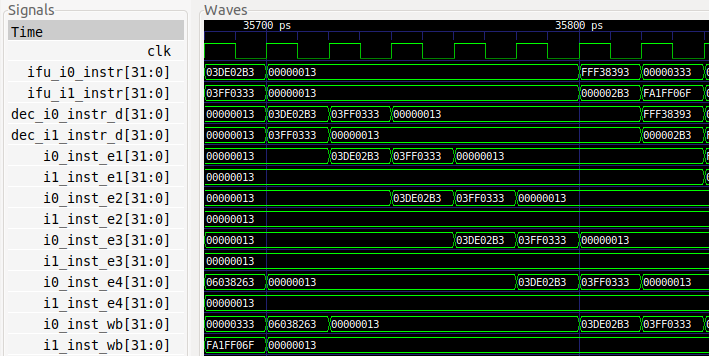
- 通路1：ifu\_i1\_instr 🡪 dec\_i1\_instr\_d

- 在模块**dec\_decode\_ctl**（译码阶段）中，这两条指令尽可能调度给相应管道。发送后，它们将继续进行三个执行阶段、提交阶段和回写阶段：

- 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

我们提供包括所有这些信号的*.tcl*文件（*[RVfpgaPath]/RVfpga/Labs/Lab14/MUL\_Instruction/test\_AssignmentWays.tcl*）。



**i-1**

**i+6**

**i+3**

**i+2**

**i+1**

**i**

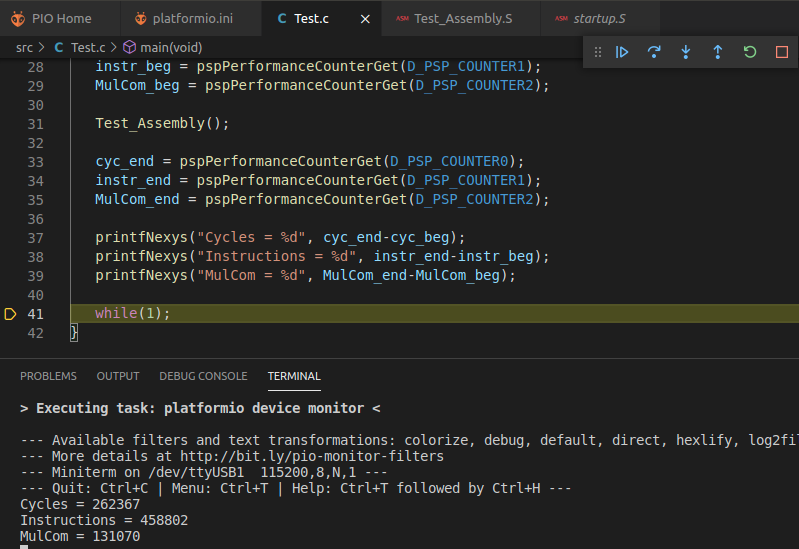
* 在周期i-1（图2和图3中未显示）中，两条mul指令处于对齐阶段：第一条指令分配给模块ifu\_aln\_ctl的通路0（ifu\_i0\_instr = 0x03de02b3），第二条指令分配给该模块的通路1（ifu\_i1\_instr = 0x03ff0333）。
* 在周期i中，两条指令已传播到模块dec\_ib\_ctl的译码阶段：第一条指令在通路0中继续执行（dec\_i0\_instr\_d = 0x03de02b3），第二条指令在通路1中继续执行（dec\_i1\_instr\_d = 0x03ff0333）。
* 在周期i+1中，第一条mul指令已传播到dec\_decode\_ctl模块的M1阶段（i0\_inst\_e1 = 0x03de02b3）。但第二条mul指令由于实验中分析的结构冒险而无法传播，因此在通路1的第一个执行阶段插入了一个气泡：i1\_inst\_e1 = 0x00000013。

此外，鉴于通路0已在译码阶段释放，第二条mul重新分配到了该通路：dec\_i0\_instr\_d = 0x03ff0333。

* 在周期i+2中，第二条mul指令传播到当前空闲的M1阶段（i0\_inst\_e1 = 0x03ff0333），第一条mul指令传播到M2阶段。
* 在周期i+3至i+6中，两条mul指令通过流水线，直到回写阶段才暂停。

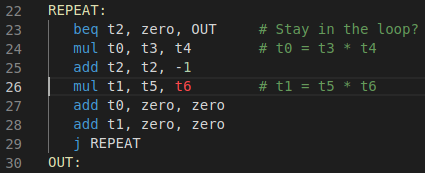
**任务：**删除循环中包含的nop指令并使用SweRV EH1中提供的性能计数器测量不同的事件（执行周期、已提交的指令/乘法数等），如实验11中所述。在分析图2中的仿真后，周期数是否符合预期？解释您的答案。

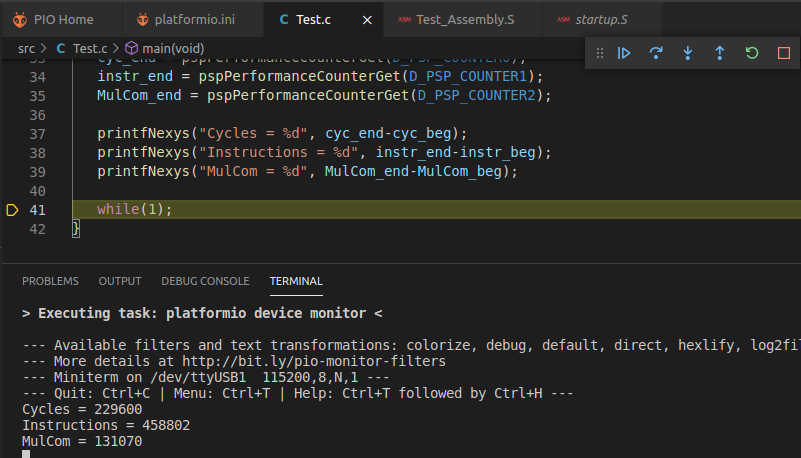
现在调整循环内的代码顺序，以达到理想的吞吐量。解释在原始代码和调整顺序的代码中获得的结果。



IPC = 458000 / 262000 = 1.748。IPC比理想值小一点，因为第二条mul指令由于结构冒险必须等待一个周期，如实验中所述。

如果我们调整代码顺序，在两条mul指令之间插入循环索引的更新，将获得理想IPC，因为我们使用有用的指令填充了结构冒险引入的气泡。





IPC = 458000 / 229000 = 2

**任务：**文件夹*[RVfpgaPath]/RVfpga/Labs/Lab14/MUL\_Instr\_Accumul\_C-Lang*提供了C程序的PlatformIO项目，此C程序用于累加循环中两次乘法的减法结果。

* 分析C程序。
* 执行仿真并检查循环的随机迭代。请注意，C程序在未经过优化的情况下编译。
* 使用SweRV EH1中提供的性能计数器测量不同的事件（周期、已提交的指令/乘法数等），如实验11中所述。

在分析图2中的仿真后，周期数是否符合预期？解释您的答案。

* 用RISC-V汇编语言创建一个相似的程序并将其与C版本进行比较。调整指令顺序以获得最佳IPC。
* 禁止C程序中的**M** RISC-V扩展并将结果与原始程序进行比较。为此，请将*platformio.ini*中的以下行：

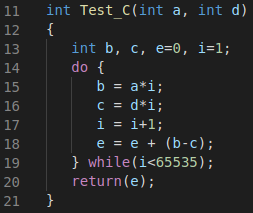
build\_flags = -Wa,-march=rv32**ima** -march=rv32**ima**

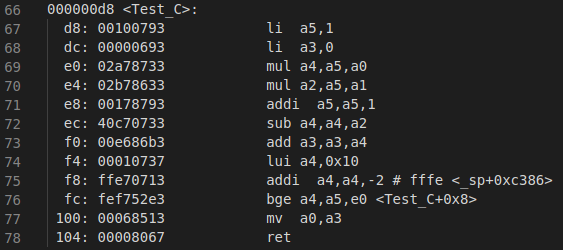
修改为：

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

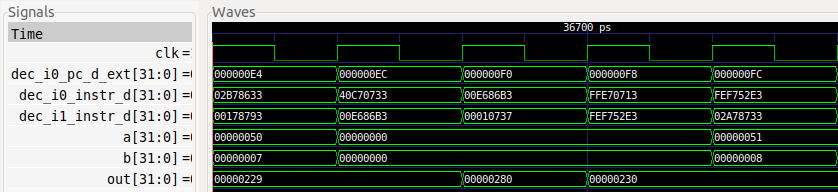
这样便可避免使用M RISC-V扩展中的指令，而是使用其他指令进行仿真。

* C程序（原版和反汇编）：

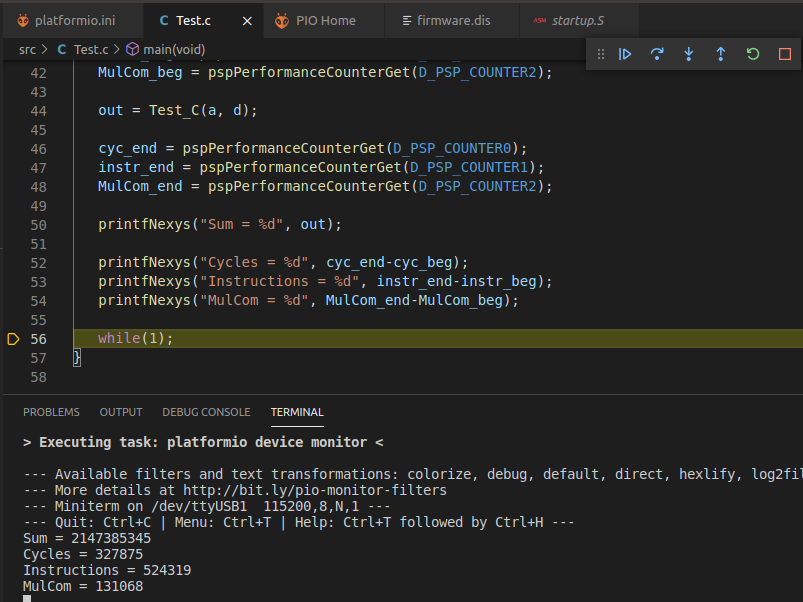




* C程序的仿真：



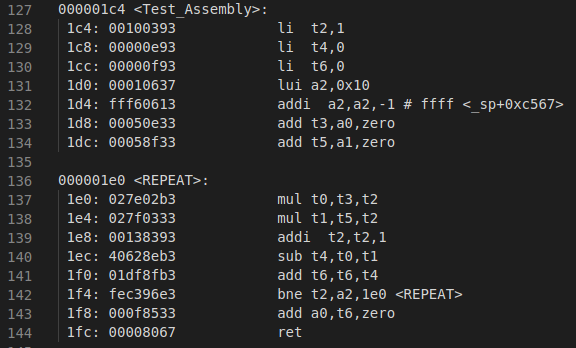
* 硬件计数器：

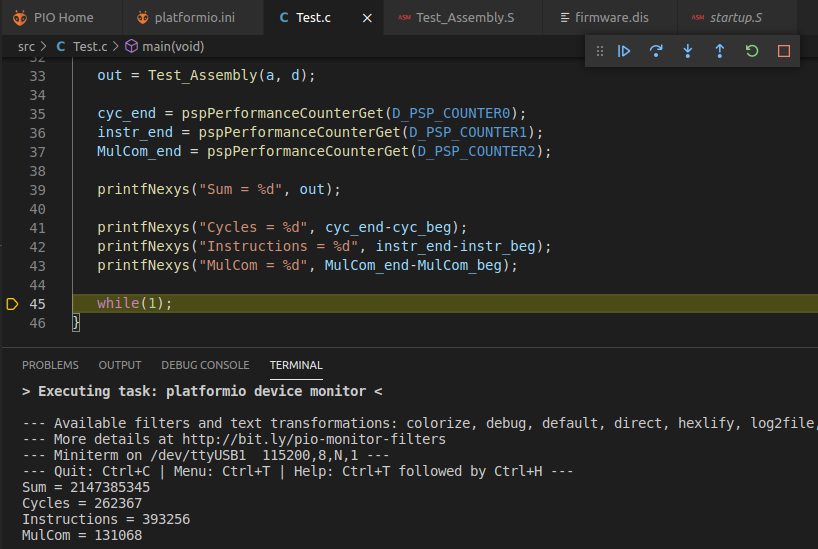


IPC = 524000 / 327000 = 1.6.由于实验15中将分析的RAW数据冒险，一些周期丢失。

* 汇编程序位于：

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



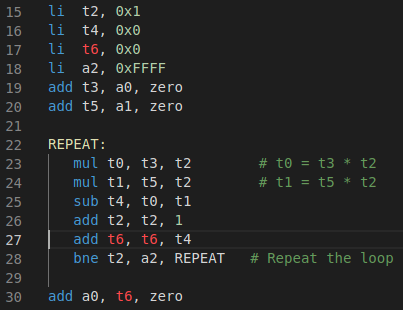


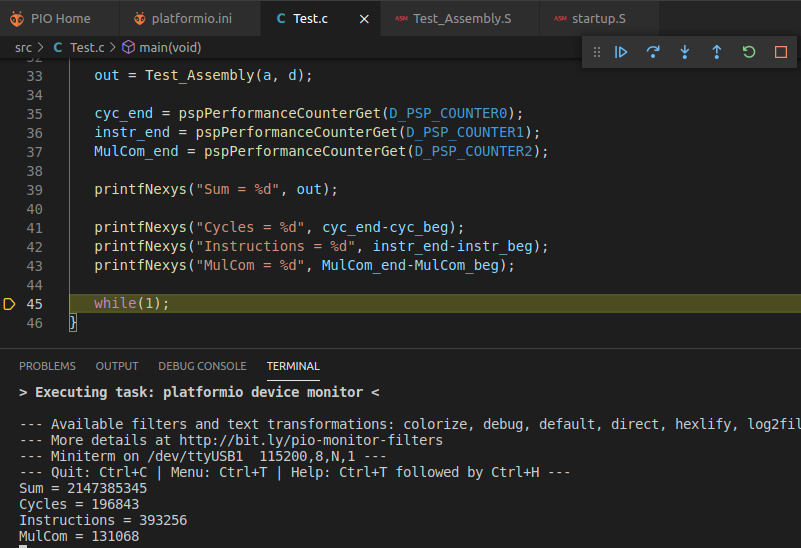
求和的结果相同，因为程序相同。

周期数略少一点，因为手动编程的汇编版本比未经优化的编译器获得的汇编版本效率更高。

指令数也更少一点。

我们按如下方式调整循环顺序：





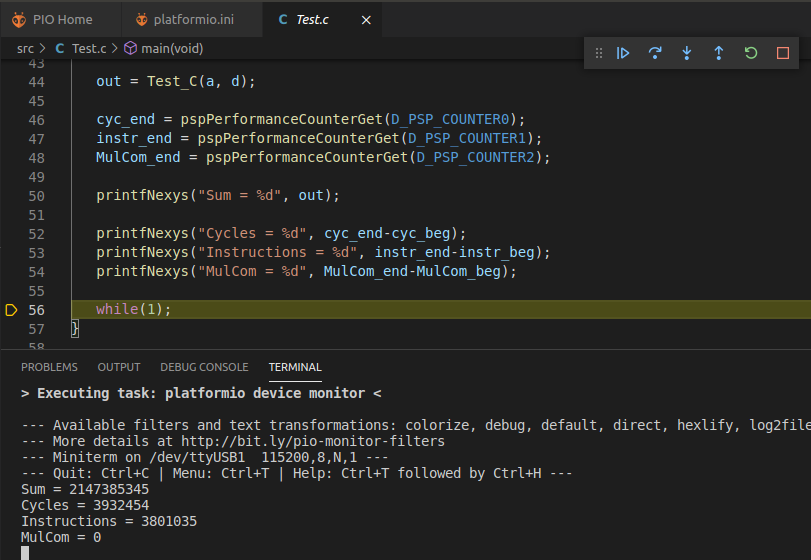
求和的结果相同，因为程序相同。

每次迭代的周期数 = 196800 / 65500 = 3

指令数相同。每次迭代的指令数 = 393000 / 65500 = 6

IPC = 393 / 197 = 1.994。我们获得了最佳IPC。

* 禁止M扩展：



求和的结果相同，因为程序相同。

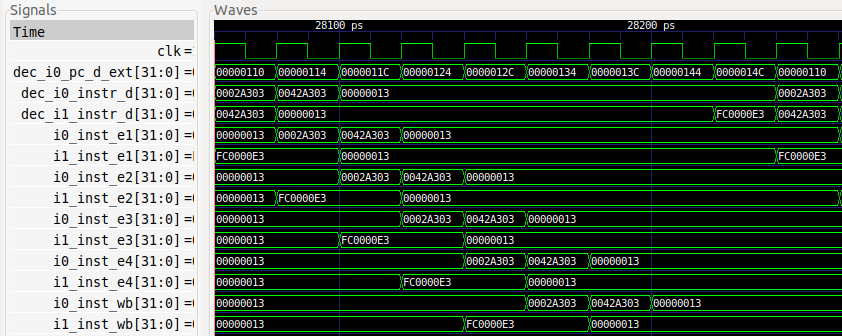
周期数多很多：约4M与约0.3M。

指令数也多很多：约3M与约0.5M。

CPI现在更好。

未提交乘法。

**任务：**修改图1中的程序，将两条mul指令替换为两条lw指令（针对DCCM）。应观察到与本部分中分析的结构冒险类似并以相似方式解除的结构冒险。



正如我们在仿真中看到的那样，两次连续装载的行为与两条连续mul指令的行为完全相同。

**任务：**在自己的计算机上重复图6中的仿真过程。使用文件*test\_NonBlocking.tcl*（在*[RVfpgaPath]/RVfpga/Labs/Lab14/LW\_Instruction\_ExtMemory*中提供）。单击几次“*Zoom In*”（放大）（）移动至60120 ps。

解答请参见实验14的主文档。

**任务：**将图6所示的仿真（非阻塞装载）与实验13的图14所示的仿真（阻塞装载）进行比较。添加比较需要的所有信号。

不提供解答。

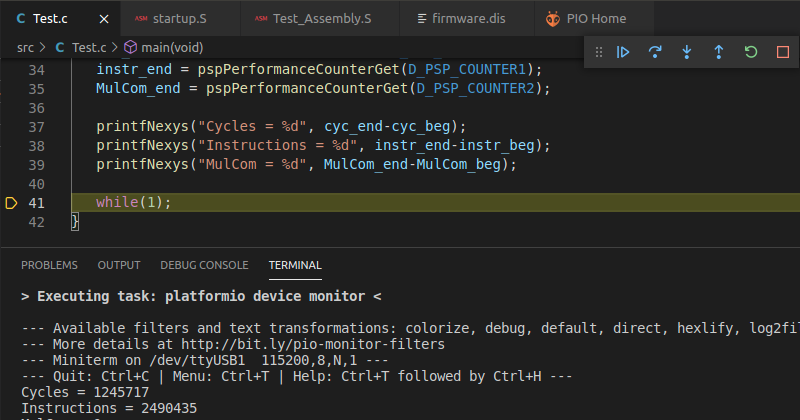
**任务：**将图7中的说明与已复制到计算机上的图6中的仿真进行比较。根据需要添加信号以扩展仿真并加深理解。

不提供解答。

**任务：**使用SweRV EH1中提供的性能计数器测量不同的事件（周期、已提交的指令/装载等），如实验11中所述。在分析图6中的仿真后，周期数是否符合预期？解释您的答案。

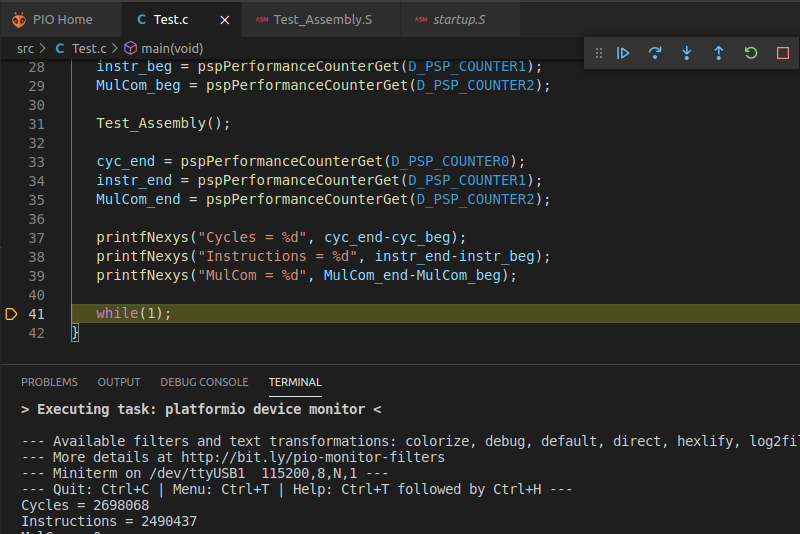
将这些结果与装载配置为阻塞装载时获得的结果进行比较。

**非阻塞装载：**

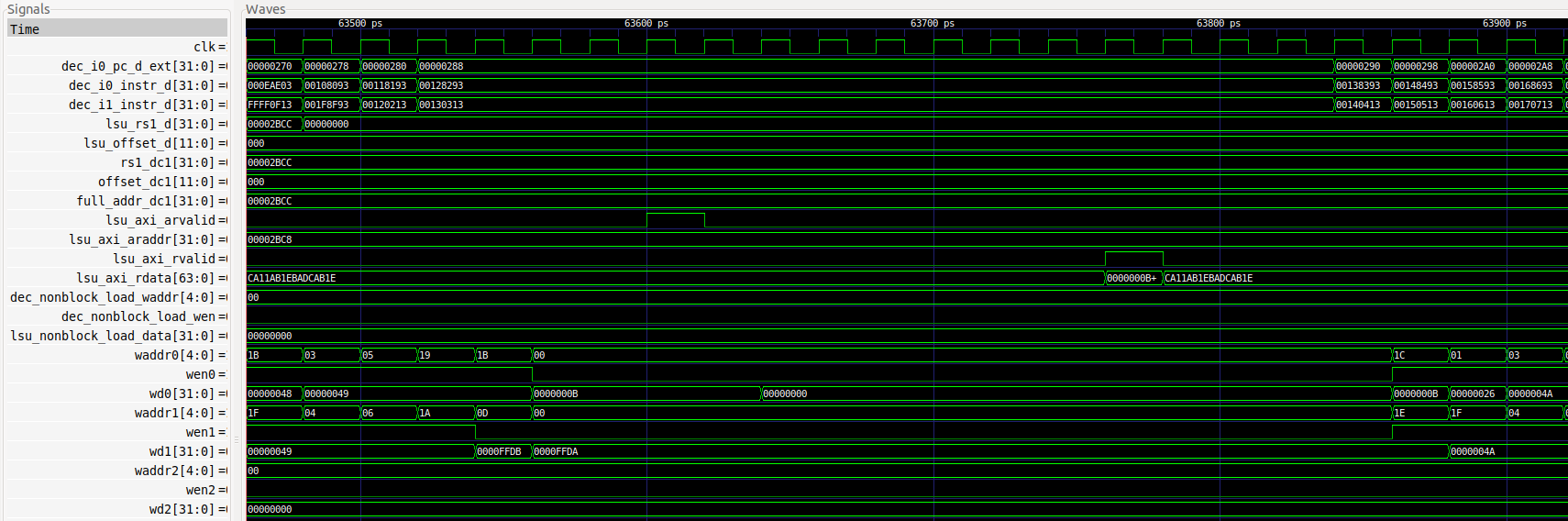


由于非阻塞装载，获得的IPC（IPC = 2490 / 1245 = 2）为理想IPC。

**阻塞装载：**



指令数相同，但现在需要更多的周期执行循环，因为装载使后续指令暂停以等待来自存储器的数据。仿真更清楚地说明了这一点。



**流水线停止**

# 练习

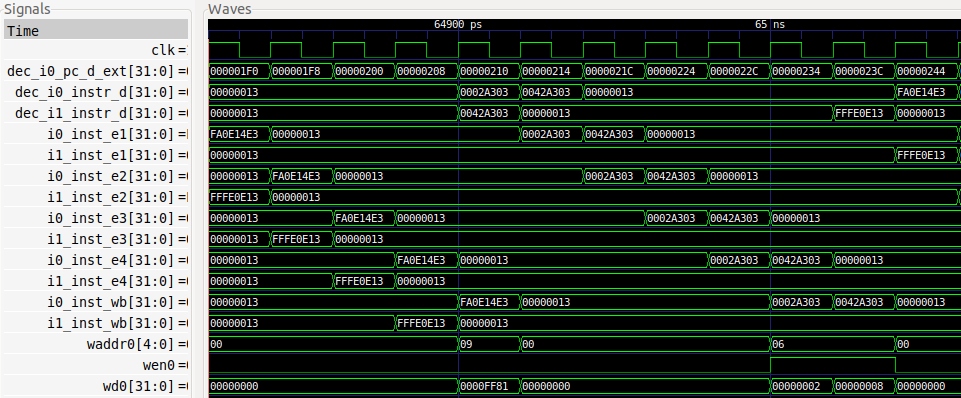
1. 在仿真中以及在开发板上分析在同一周期到达L/S管道的两条连续访存指令（可以分析装载和存储等两条连续访存指令的任意组合）之间发生的结构冒险。测试非阻塞装载和阻塞装载。可以使用以下位置提供的PlatformIO项目：*[RVfpgaPath]/RVfpga/Labs/Lab14/TwoConsecutiveLW\_Instructions*。

两次连续装载：

210: 0002a303 lw t1,0(t0)

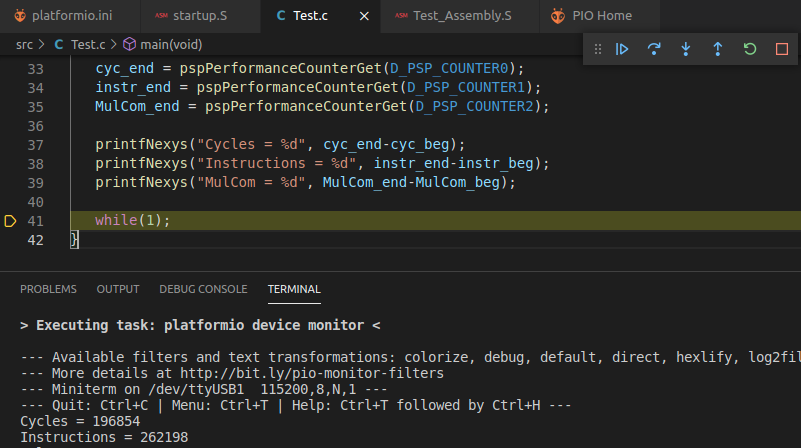
214: 0042a303 lw t1,4(t0)

* 仿真：



由于L/S管道中的结构冒险，第二条lw必须暂停1个周期，类似于乘法管道处理两条连续mul指令的情形。

* 在开发板上执行：



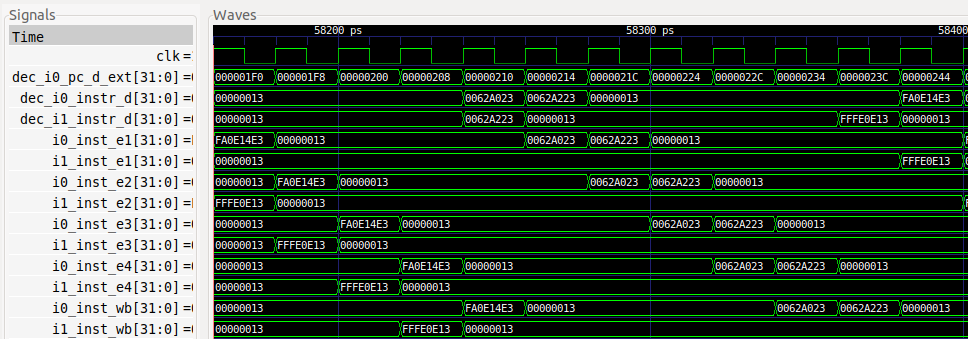
IPC = 262 / 196 = 1.33

两次连续存储：

210: 0062a023 sw t1,0(t0)

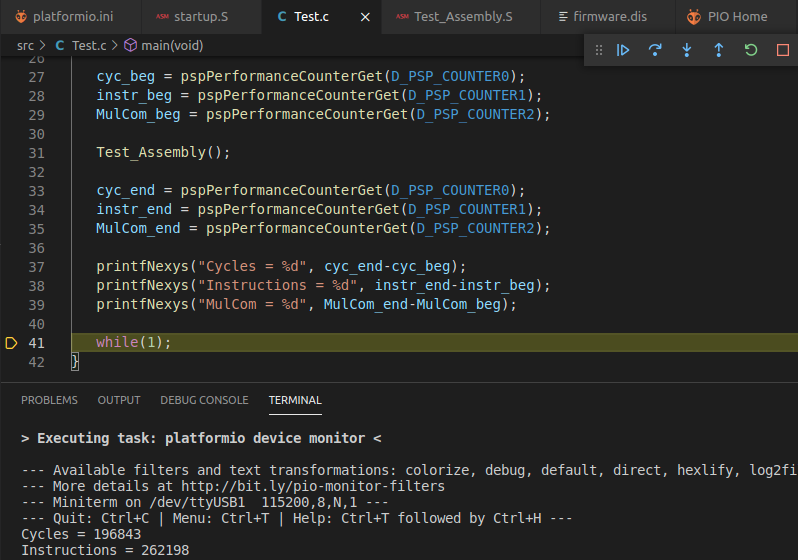
214: 0062a223 sw t1,4(t0)

* 仿真：



由于L/S管道中的结构冒险，第二条sw必须暂停1个周期，类似于乘法管道处理两条连续mul指令的情形。

* 在开发板上执行：



IPC = 262 / 196 = 1.33

1. （*以下练习基于《计算机组织结构和设计》（RISC-V版本，作者Patterson & Hennessy（[HePa]））中的练习4.22。*）

请看下面的RISC-V汇编语言片段：

sd x29, 12(x16)

ld x29, 8(x16)

sub x17, x15, x14

beqz x17, label

add x15, x11, x14

sub x15, x30, x14

假设我们将SweRV EH1处理器修改为只有一个存储器（处理指令和数据）。在这种情况下，每次程序需要在一条指令访问数据的同一周期内对另一条指令取指时，均存在结构冒险。

* 1. 绘制流水线图以显示上述代码将在SweRV EH1处理器的这一假想版本中的哪个位置暂停。
  2. 通常而言，能否通过调整代码顺序来减少此结构冒险引起的暂停/nop的数量？
  3. 是否必须在硬件中处理这种结构冒险？可以看到，可通过在代码中添加nop来消除数据冒险。能否对这种结构冒险执行相同的操作？如果可以，请说明方法。如果不能，请说明原因。

不提供解答。

# 附录A - 译码阶段两条同时进行的DIV指令

**任务：**可以对div指令进行与实验12中对算术-逻辑指令进行的研究类似的研究：查看通过各流水线阶段的指令流，分析控制位（根据实验11的附录D，div指令有一个称为div\_pkt\_t的特定结构类型，并且模块**dec\_decode\_ctl**中定义了一个名为div\_p的信号）等。

不提供解答。

**任务：**检查来自**exu\_div\_ctl**的Verilog代码，了解如何计算除法。此外，还要分析信号div\_stall、finish\_early和finish的影响。作为一项可选练习，可以用自己的除法单元或互联网上的除法单元来替换此除法单元。

不提供解答。

**任务：**验证这对32位值（0x03de42b3和0x03ff4333）是否对应RISC-V架构中的指令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**

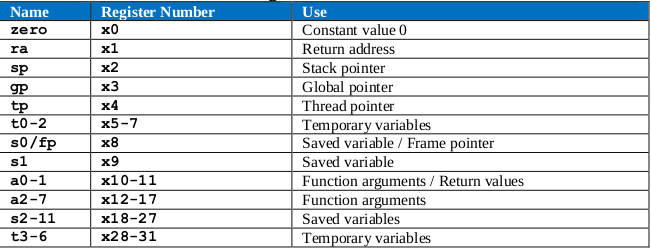
**rd = 00110 = x6 (t1)**

**op = 0110011**

来自DDCARV的附录B：







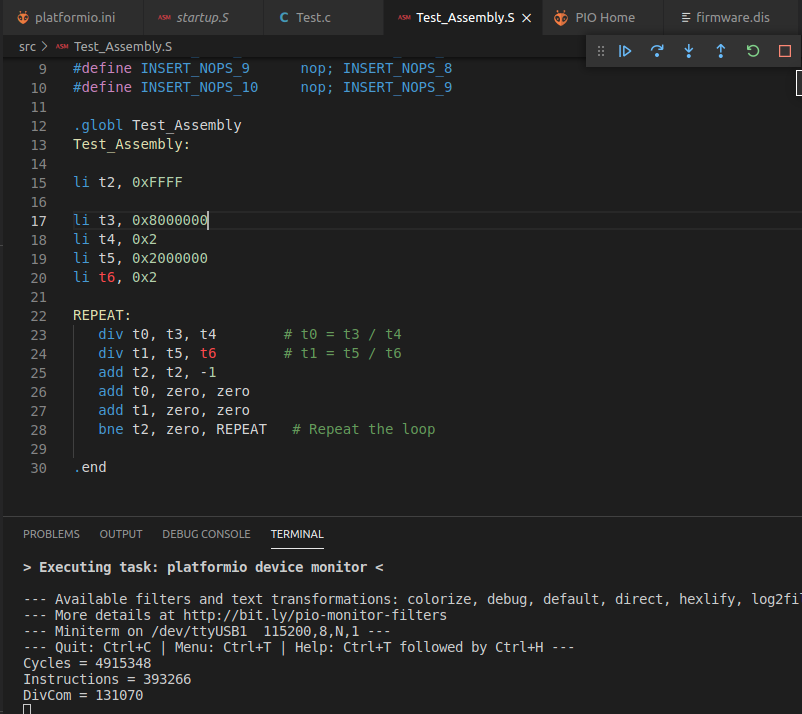
**任务：**在自己的计算机上重复图9中的仿真过程，以进行详细的分析。

解答请参见实验14的主文档。

**任务：**将图10中的说明与已复制到计算机上的图9中的仿真进行比较。根据需要添加信号以扩展仿真并加深理解。

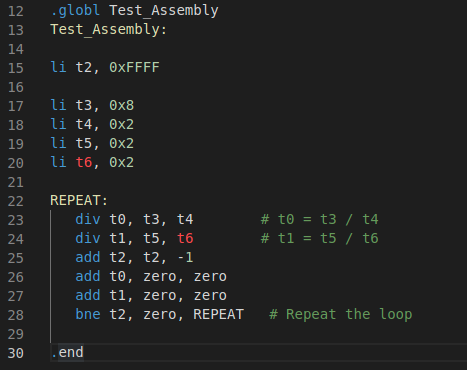
不提供解答。

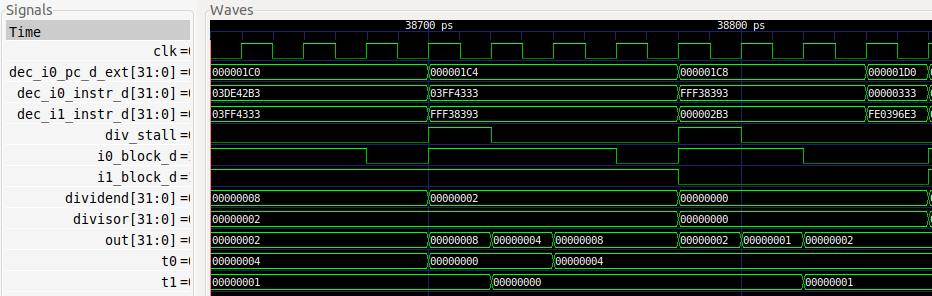
**任务：**使用SweRV EH1中提供的性能计数器测量不同的事件（周期、已提交的指令/除法等），如实验11中所述。在分析图9中的仿真后，周期数是否符合预期？解释您的答案。



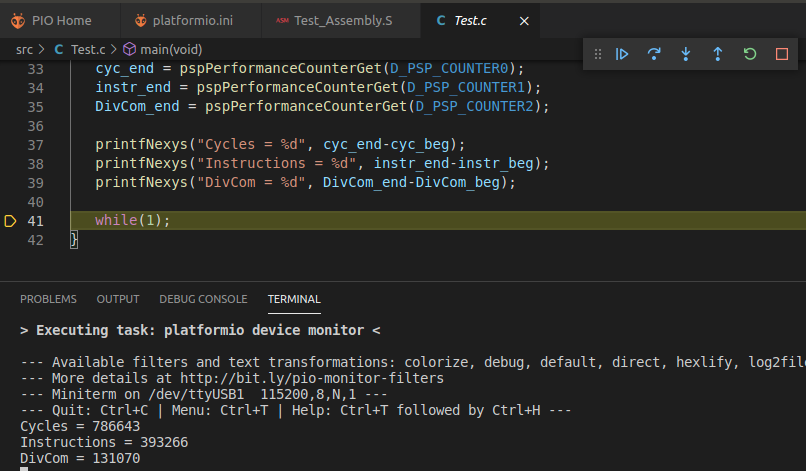
CPI = 4910000 / 393000 = 12.考虑到每个除法运算大约需要34个周期来执行，而其他指令各需要½个周期（上面是能够预测的近似情况），近似的理论计算如下：在34 + 34 + ½ + ½ + ½ + ½个周期内执行6条指令 🡪 CPI = 70 / 6 = 11

**任务：**尝试不同的被除数和除数，了解用于计算结果的周期数与其值的相关性。通过仿真和硬件计数器查看实验。





现在，除法计算仅需大约5个周期。



鉴于计算每个除法运算的时间缩短了很多，CPI也减小了很多（每个周期约为2）。

**任务：**文件夹*[RVfpgaPath]/RVfpga/Labs/Lab14/DIV\_Instr\_Accumul\_C-Lang*提供C程序的PlatformIO项目，此C程序用于累加循环中两次除法的减法结果。

* 分析C程序。
* 执行仿真并检查循环的随机迭代。请注意，C程序在未经过优化的情况下编译。
* 使用SweRV EH1中提供的性能计数器测量不同的事件（周期、已提交的指令/除法等），如实验11中所述。

在分析图9中的仿真后，周期数是否符合预期？解释您的答案。

* 用RISC-V汇编语言创建一个相似的程序并将其与C版本进行比较。
* 禁止C程序中的**M** RISC-V扩展并将结果与原始程序进行比较。为此，请将*platformio.ini*中的以下行：

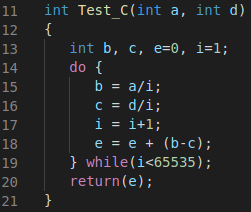
build\_flags = -Wa,-march=rv32**ima** -march=rv32**ima**

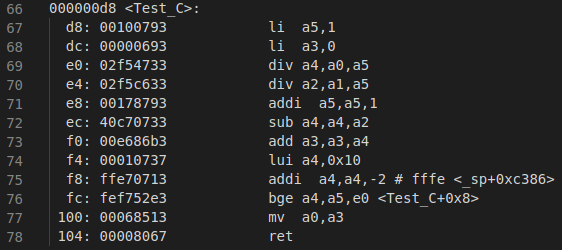
修改为：

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

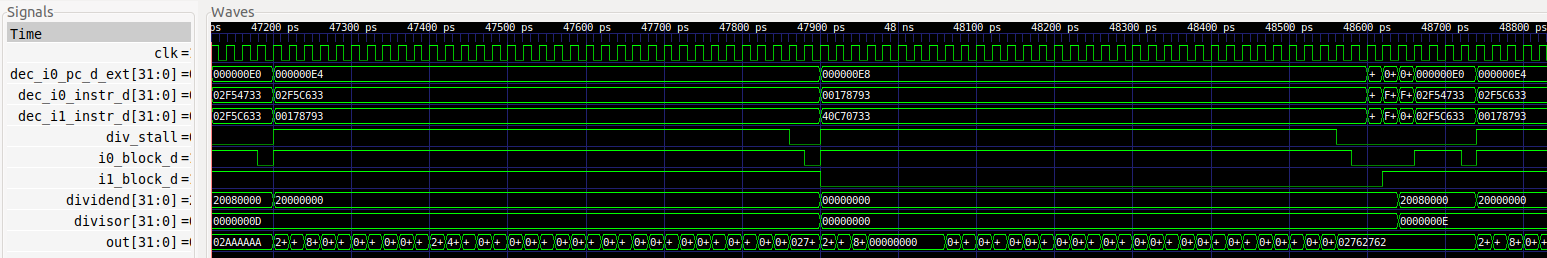
这样便可避免使用RISC-V M扩展中的指令，而是使用其他指令进行仿真。

* C程序（原版和反汇编）：

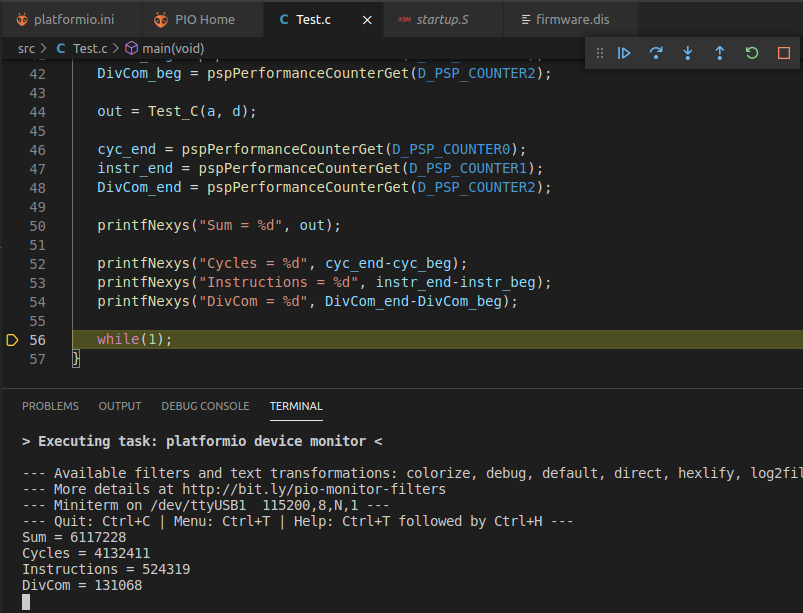




* C程序的仿真：

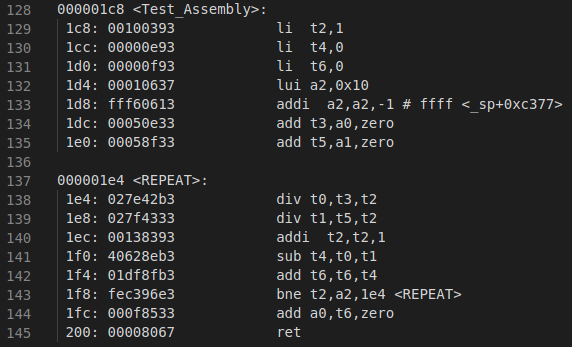


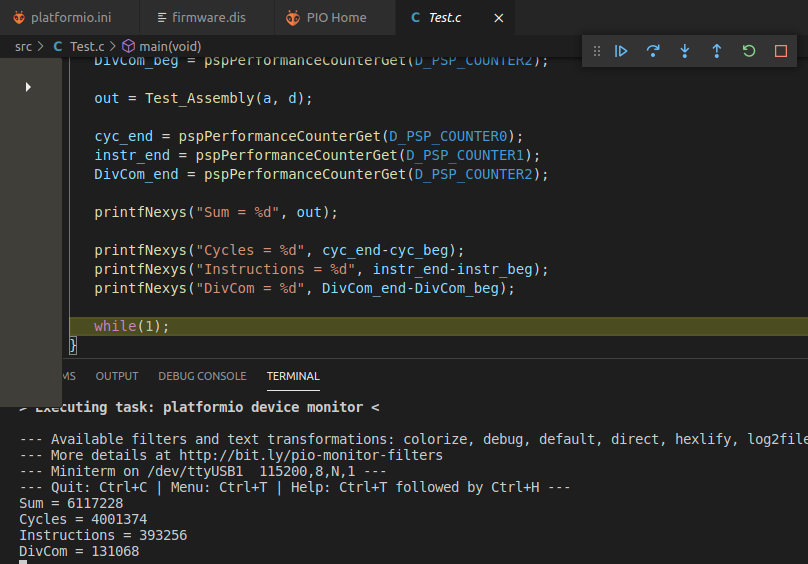
* 硬件计数器：



* 汇编程序位于：

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



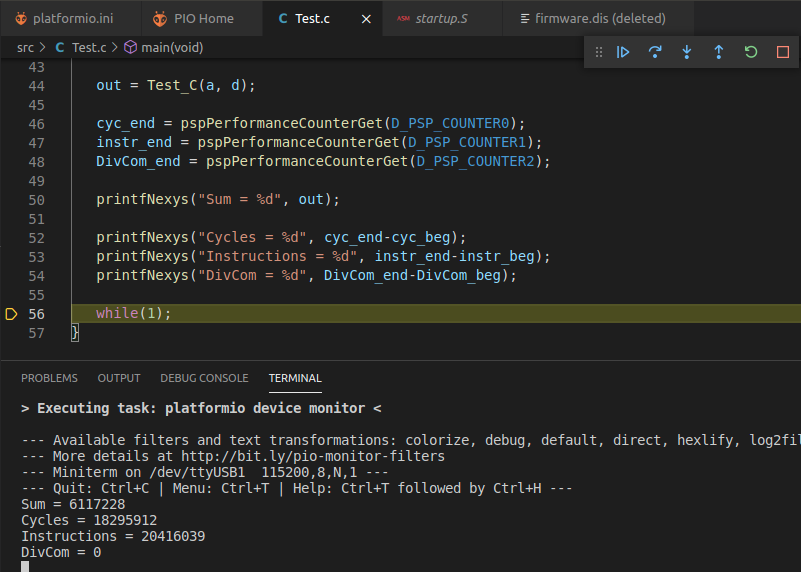


求和的结果相同，因为程序相同。

周期数略少一点，因为手动编程的汇编版本比未经优化的编译器获得的汇编版本效率更高。

指令数也更少一点。

* 禁止M扩展：



求和的结果相同，因为程序相同。

周期数多很多：约18M与约4M。

指令数也多很多：约20M与约0.5M。

CPI现在更好。

未提交除法。

**任务：**在SweRV EH1中，div指令是阻塞的。修改处理器以允许非阻塞div指令。

然后向SweRV EH1处理器添加第二个除法器，以便图8的示例中的两条div指令可以并行执行。

不提供解答。