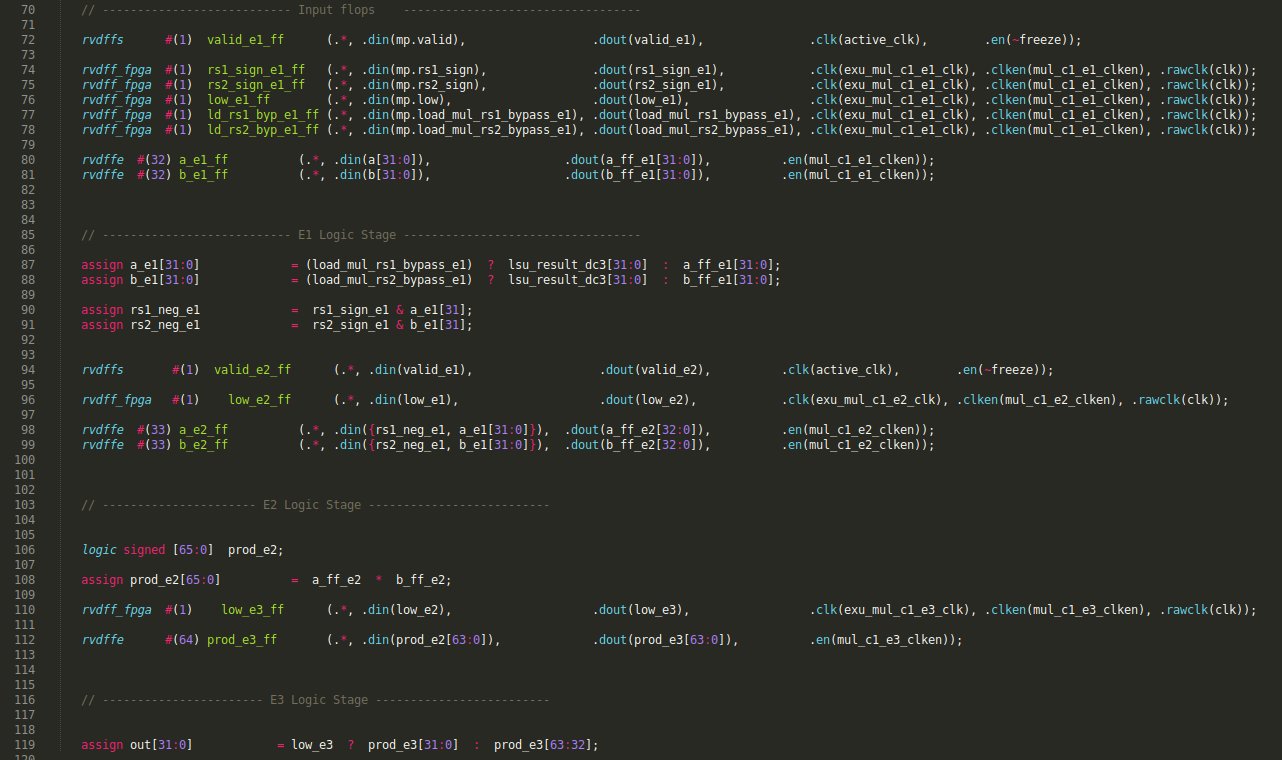
# TAREFAS

**TAREFA:** Pode realizar um estudo semelhante para a instrução mul como o realizado no Lab 12 para as instruções aritméticas-lógicas: veja o fluxo da instrução passando pelos andares do pipeline, analise os bits de controle (lembre-se do Apêndice D do Lab 11 que há um tipo de estrutura específica para a instrução mul chamada mul\_pkt\_t e há um sinal definido no módulo **dec\_decode\_ctl** chamado mul\_p) etc.

Solução não fornecida.

**TAREFA:** Inspecione o código Verilog do **exu\_mul\_ctl** e veja como a multiplicação é computada. Lembre-se de que o RISC-V inclui 4 instruções de multiplicação (mul, mulh, mulhsu e mulhu), e todas elas devem ser compatíveis com o hardware.

Como um exercício opcional, substitua a Multiply Unit por sua própria unidade ou por uma da Internet.



* As entradas e os bits de controle produzidos no andar Decode são registrados nas linhas 72-81.

M1:

* No caso de uma dependência de dados entre a multiplicação e uma leitura anterior, ocorre um forwarding nas linhas 87 a 88.
* Além disso, o tratamento do sinal dos operandos de entrada é determinado nas linhas 90-91. Lembre-se de que o RISC-V inclui três versões da operação "multiply high": mulh, mulhsu e mulhu.
* Esses valores são propagados para M2.

M2:

* A multiplicação real é realizada na linha 108.

M3:

* A parte baixa/alta é retornada em out[31:0] na linha 119. A parte baixa é selecionada no caso de uma instrução mulh, enquanto a parte alta é selecionada no caso de qualquer uma das três instruções mulh.

**TAREFA:** Verificar se esse par de 32 bits (0x03de02b3 e 0x03ff0333) corresponde às instruções mul t0,t3,t4 e mul t1,t5,t6 na arquitetura RISC-V.

**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**

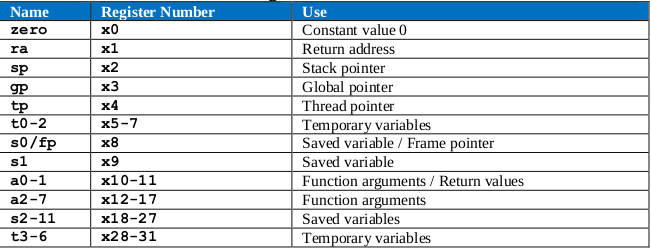
**rd = 00110 = x6 (t1)**

**op = 0110011**

Do Apêndice B do DDCARV:







**TAREFA:** Replique a simulação da Figura 2 no seu computador e analise-a mais detalhadamente.

Solução fornecida no documento principal do Lab 14.

**TAREFA:** Compare a ilustração da Figura 3 com a simulação da Figura 2, concentrando-se nas duas instruções mul. Especificamente, analise como as duas instruções são atribuídas às duas vias nos andares Align e Decode.

- No módulo **ifu\_aln\_ctl** (andar Align), as duas instruções são atribuídas a:

- Via 0: ifu\_i0\_instr

- Via 1: ifu\_i1\_instr

- No módulo **dec\_ib\_ctl,** as duas instruções são armazenadas no buffer de Align para Decode:

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

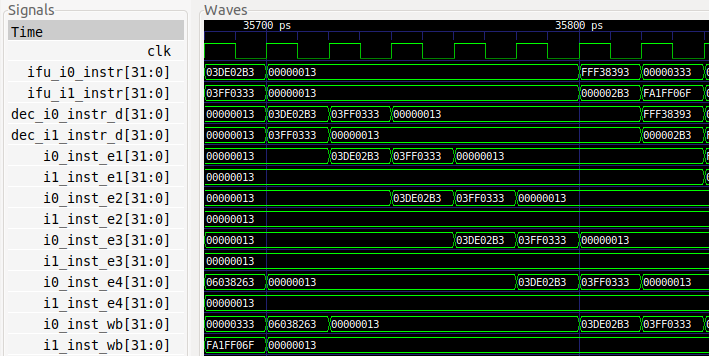
- Via 1: ifu\_i1\_instr🡪 dec\_i1\_instr\_d

- No módulo **dec\_decode\_ctl** (andar Decode), as duas instruções são programadas para os pipes correspondentes, se possível. Depois de enviadas, elas continuam pelos três andares de execução, Commit e Writeback:

- Via 0: i0\_inst\_e1 - i0\_inst\_e2 - i0\_inst\_e3 - i0\_inst\_e4 - i0\_inst\_wb

- Via 1: i1\_inst\_e1 - i1\_inst\_e2 - i1\_inst\_e3 - i1\_inst\_e4 - i1\_inst\_wb

Fornecemos um ficheiro *.tcl* chamado *[RVfpgaPath]/RVfpga/Labs/Lab14/MUL\_Instruction/test\_AssignmentWays.tcl* que inclui todos esses sinais.



**i-1**

**i+6**

**i+3**

**i+2**

**i+1**

**i**

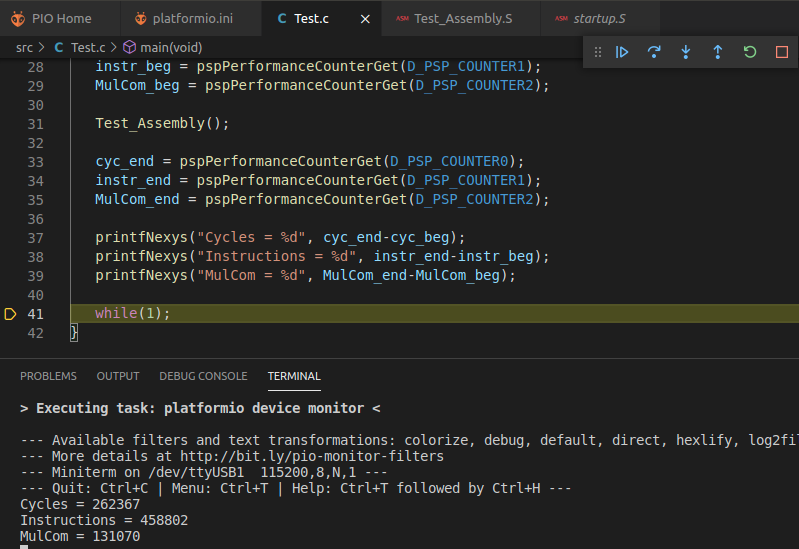
* No ciclo i-1 (não mostrado na Figura 2 nem na Figura 3), as duas instruções mul estão no andar Align: a primeira é atribuída à Via 0 (ifu\_i0\_instr = 0x03de02b3) e a segunda é atribuída à Via 1 (ifu\_i1\_instr = 0x03ff0333) no módulo ifu\_aln\_ctl.
* No ciclo i, as duas instruções foram propagadas para o andar Decode no módulo dec\_ib\_ctl: a primeira continua na Via 0 (dec\_i0\_instr\_d = 0x03de02b3) e a segunda continua na Via 1 (dec\_i1\_instr\_d = 0x03ff0333).
* No ciclo i+1, a primeira instrução mul foi propagada para o andar M1 no módulo dec\_decode\_ctl (i0\_inst\_e1 = 0x03de02b3). Entretanto, a segunda instrução mul não pôde ser propagada devido ao conflito estrutural analisado no Lab e, portanto, uma bolha foi inserida no primeiro andar de execução da Via 1: i1\_inst\_e1 = 0x00000013.

Além disso, como a via 0 foi libertada no andar Decode, o segundo mul foi reatribuído a essa via: dec\_i0\_instr\_d = 0x03ff0333.

* No ciclo i+2, a segunda instrução mul é propagada para o andar M1, que agora está livre (i0\_inst\_e1 = 0x03ff0333), e a primeira instrução mul é propagada para o andar M2.
* Nos ciclos i+3 a i+6, as duas instruções mul progridem pelo pipeline sem paradas até o andar Writeback.

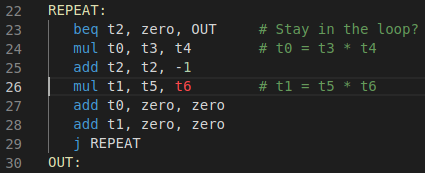
**TAREFA:** Remova as instruções nop incluídas no ciclo e meça os diferentes eventos (ciclos, instruções/multiplicações completadas, etc.) usando os contadores de desempenho disponíveis no SweRV EH1, conforme explicado no Lab 11. O número de ciclos está de acordo com o esperado depois de analisar a simulação da Figura 2? Justifique sua resposta.

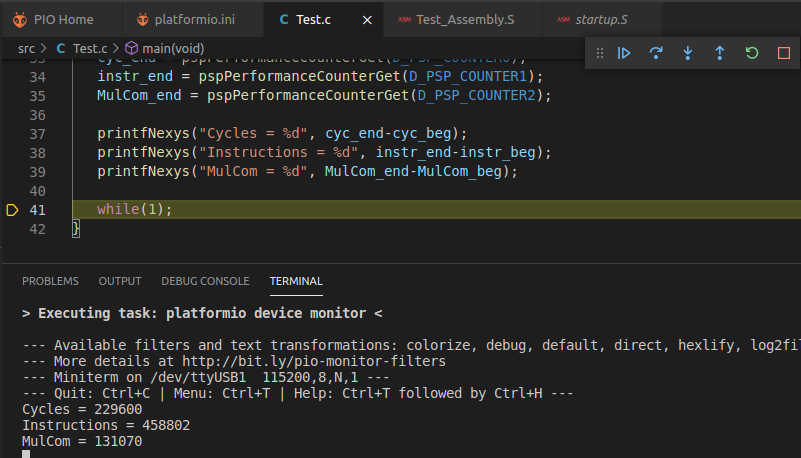
Agora, reordene o código dentro do ciclo tentando alcançar a taxa de transferência ideal. Justifique os resultados obtidos no código original e no código reordenado.



IPC = 458000 / 262000 = 1,748. O IPC é um pouco menor do que o ideal porque a segunda instrução mul deve esperar um ciclo devido ao conflito estrutural, conforme explicado no Lab.

Se reordenarmos o código, inserindo entre as duas instruções mul a atualização do índice do ciclo, obteremos o IPC ideal, pois preenchemos a bolha introduzida pelo conflito estrutural com uma instrução útil.





IPC = 458000 / 229000 = 2

**TAREFA:** A pasta *[RVfpgaPath]/RVfpga/Labs/Lab14/MUL\_Instr\_Accumul\_C-Lang* fornece o projeto PlatformIO de um programa em C que acumula a subtração de duas multiplicações num ciclo.

* Analisar o programa C.
* Execute uma simulação e inspecione uma iteração aleatória do ciclo. Observe que o programa em C é compilado sem otimizações.
* Meça diferentes eventos (ciclos, instruções/multiplicações completadas, etc.) usando os contadores de desempenho disponíveis no Swe RV EH1, conforme explicado no Lab 11.

O número de ciclos está de acordo com o esperado após a análise da simulação da Figura 2? Justifique sua resposta.

* Crie um programa análogo em Assembly RISC-V e compare-o com a versão em C. Reordene as instruções tentando obter o melhor IPC possível.
* Desative a extensão **M** RISC-V no programa C e compare os resultados com o programa original. Para fazer isso, modifique a seguinte linha no ficheiro *platformio.ini* de:

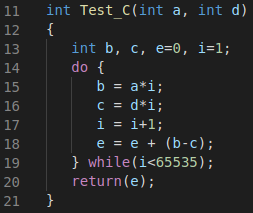
build\_flags = -Wa,**-march=rv32ima -march=rv32ima**

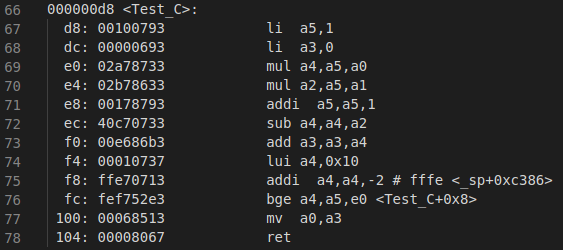
Para:

build\_flags = -Wa,**-march=rv32ia -march=rv32ia**

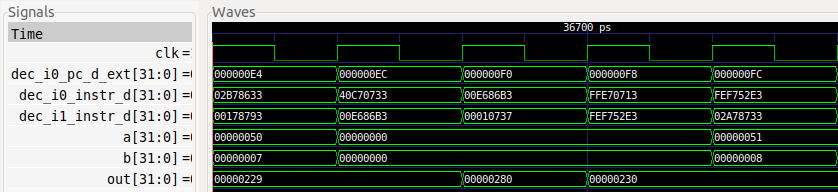
Isso evita o uso das instruções da extensão M RISC-V e emula-as usando outras instruções.

* Programa C (original e Disassembly):

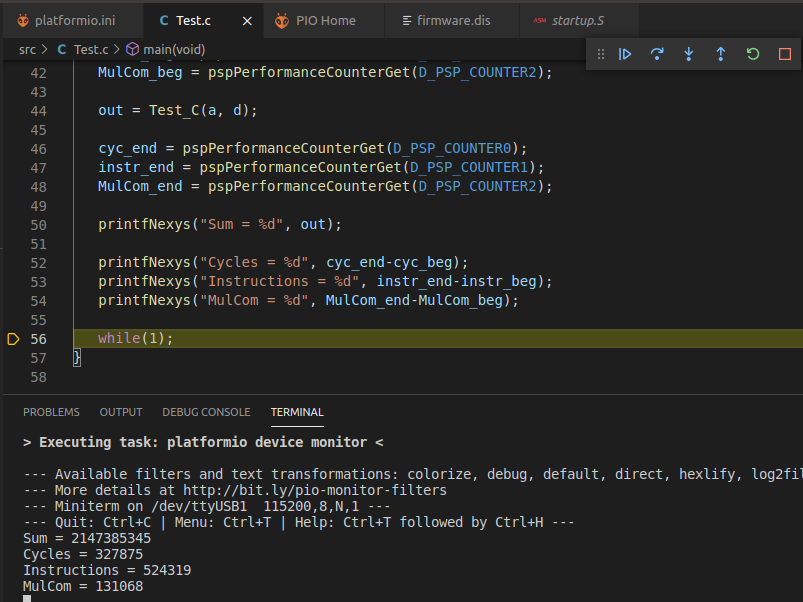




* Simulação do programa em C:



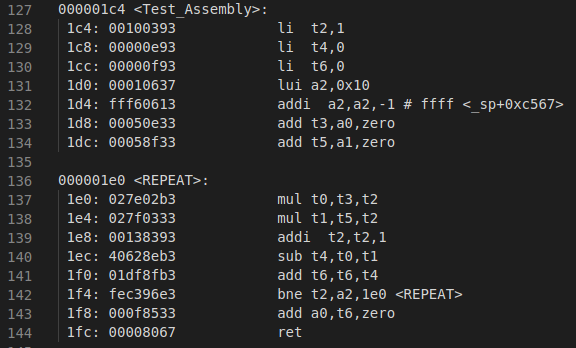
* Contadores HW:

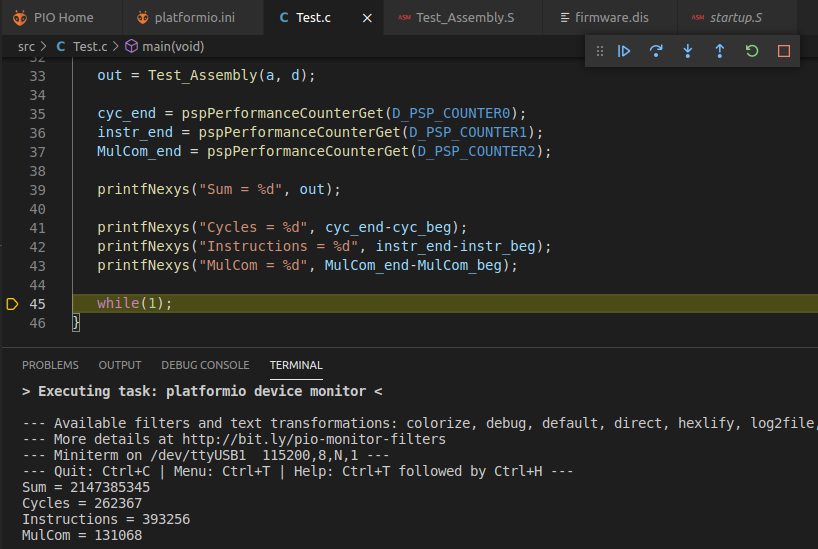


IPC = 524000 / 327000 = 1,6. Alguns ciclos são perdidos devido aos conflitos de dados RAW, que analisaremos no Lab 15.

* O programa Assembly pode ser encontrado em:

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



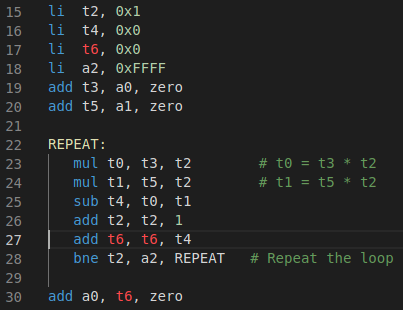


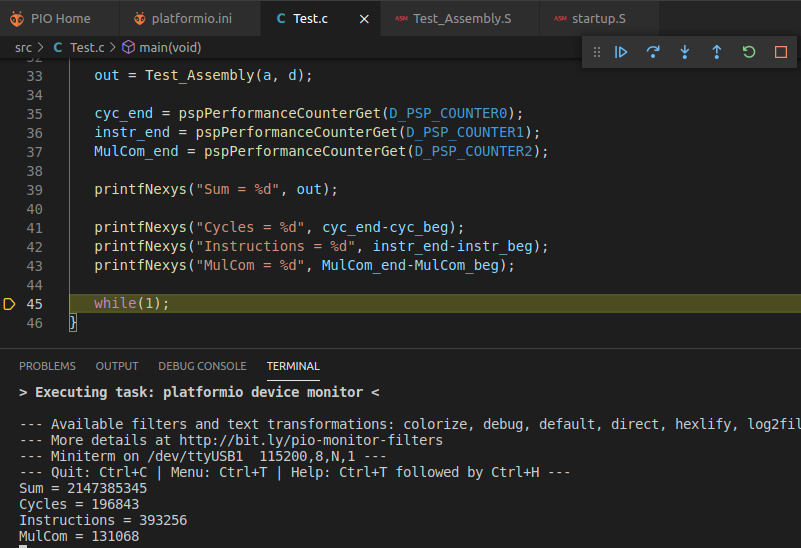
O resultado da soma é o mesmo, pois o programa é o mesmo.

O número de ciclos é um pouco menor, pois a versão Assembly programada manualmente é mais eficiente do que a obtida pelo compilador sem otimizações.

O número de instruções também é um pouco menor.

Reordenamos o ciclo da seguinte forma:





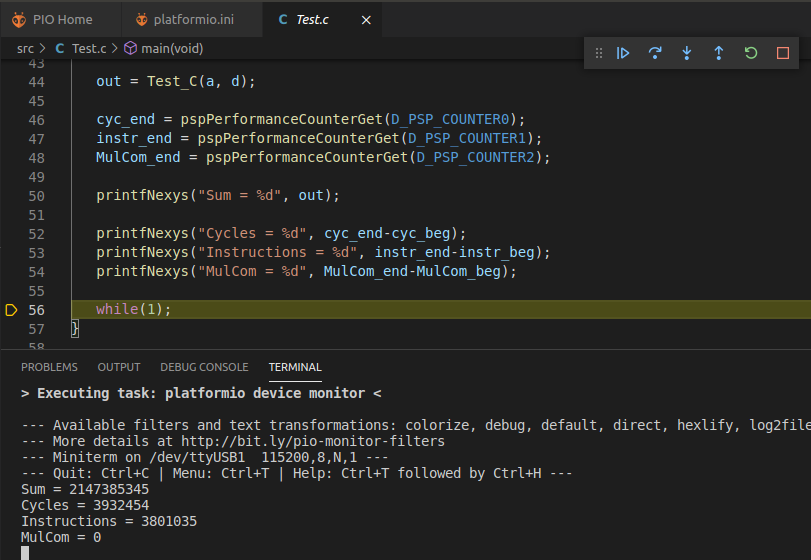
O resultado da soma é o mesmo, pois o programa é o mesmo.

Número de ciclos por iteração = 196800 / 65500 = 3

O número de instruções é o mesmo. Número de instruções por iterações = 393000 / 65500 = 6

IPC = 393 / 197 = 1,994. Obtemos o IPC ideal.

* Desativar a extensão M:



O resultado da soma é o mesmo, pois o programa é o mesmo.

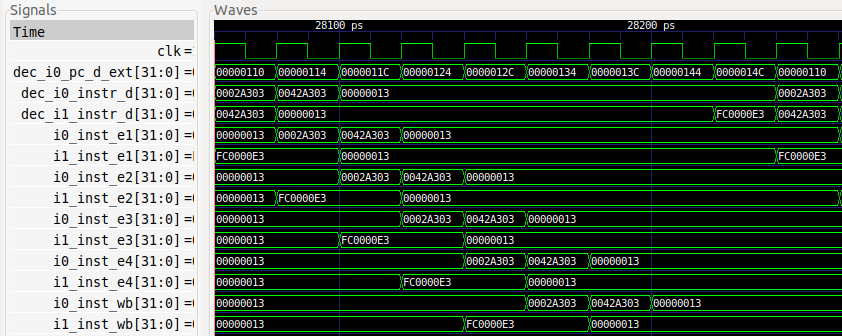
O número de ciclos é muito maior: Cerca de 4 milhões contra cerca de 0,3 milhão.

O número de instruções também é muito maior: Cerca de 3 milhões contra cerca de 0,5 milhão.

O IPC está melhor agora.

Não há multiplicações completadas.

**TAREFA:** Modifique o programa da Figura 1, substituindo as duas instruções mul por duas instruções lw para a DCCM. Deve observar um conflito estrutural análogo ao analisado nesta seção e resolvido por uma via semelhante.



Como podemos ver na simulação, o comportamento para duas leituras consecutivas é exatamente o mesmo que no caso de duas instruções mul consecutivas.

**TAREFA:** Replique a simulação da Figura 6 no seu computador. Use o ficheiro *test\_NonBlocking.tcl* (fornecido em *[RVfpgaPath]/RVfpga/Labs/Lab14/LW\_Instruction\_ExtMemory*). Aumente o *zoom* ( ) várias vezes e vá para 60120ps.

Solução fornecida no documento principal do Lab 14.

**TAREFA:** Compare a simulação mostrada na Figura 6 (leitura não-bloqueante) com a simulação mostrada na Figura 14 do Lab 13 (leitura bloqueante). Adicione todos os sinais necessários para a comparação.

Solução não fornecida.

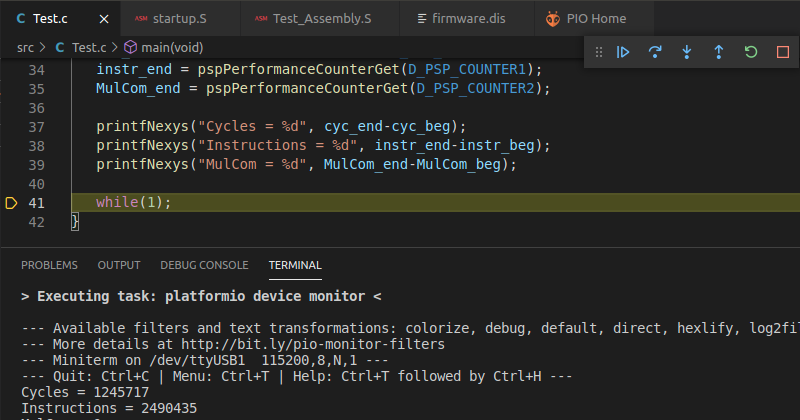
**TAREFA:** Compare a ilustração da Figura 7 com a simulação da Figura 6 que reproduziu no seu computador. Adicione sinais para ampliar a simulação e aprofundar a compreensão, conforme desejar.

Solução não fornecida.

**TAREFA:** Meça diferentes eventos (ciclos, instruções/leituras completadas, etc.) usando os contadores de desempenho disponíveis no SweRV EH1, conforme explicado no Lab 11. O número de ciclos está de acordo com o esperado após a análise da simulação da Figura 6? Justifique sua resposta.

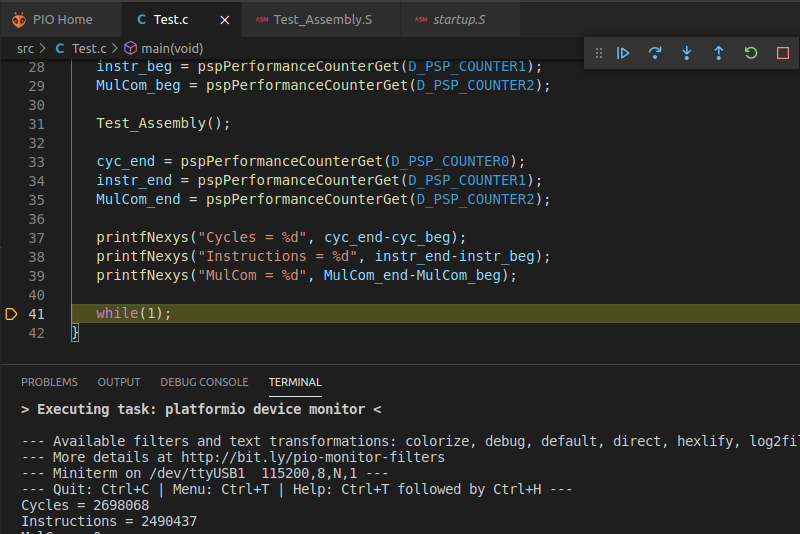
Compare esses resultados com os obtidos quando as leituras são configuradas como leituras bloqueantes.

**Leituras não-bloqueantes:**

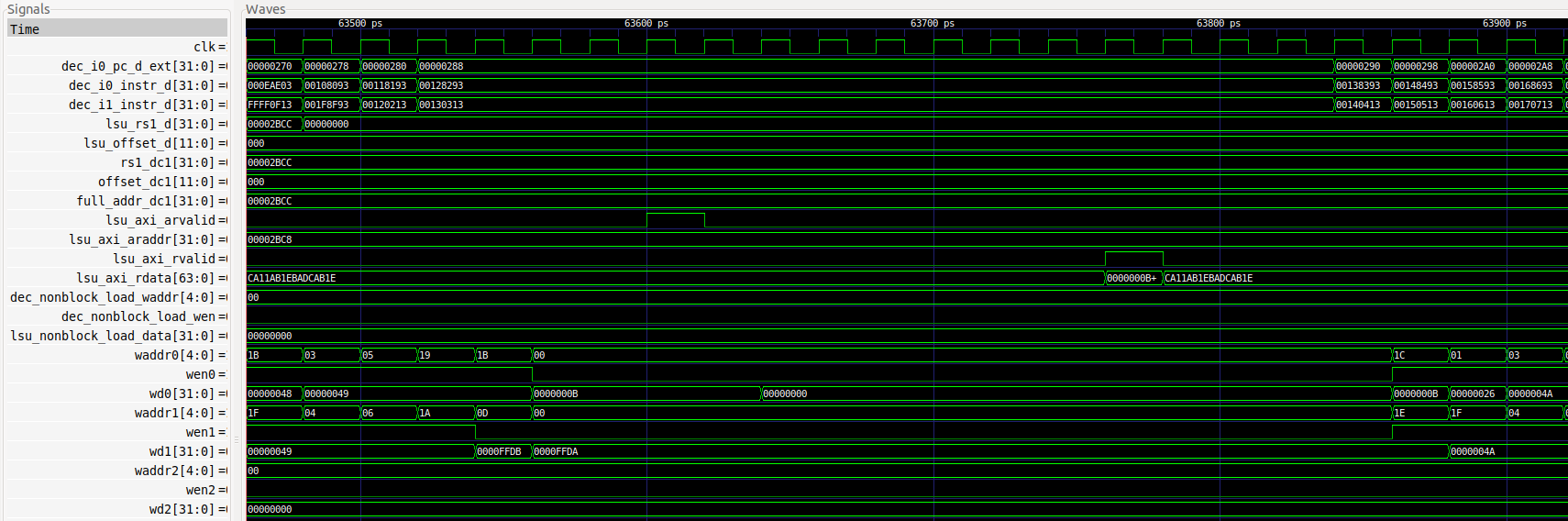


O IPC obtido (IPC = 2490 / 1245 = 2) é o ideal graças à carga sem bloqueio.

Leituras bloqueantes**:**



O número de instruções é o mesmo, mas agora são necessários muito mais ciclos para executar o ciclo, já que as instruções de leitura fazem com que as instruções subsequentes fiquem paradas para que os dados venham da memória. A simulação demonstra isso com mais clareza.



**Pipeline Parado**

# EXERCÍCIOS

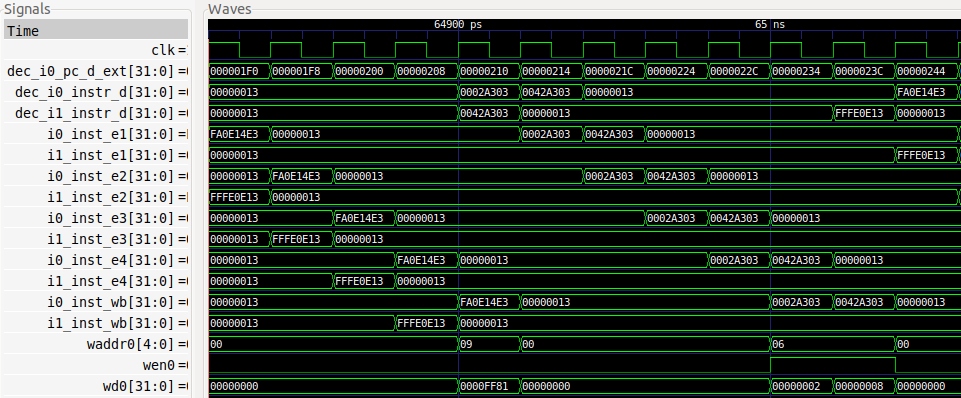
1. Analise, tanto na simulação como na placa, o conflito estrutural que ocorre entre duas instruções de memória consecutivas (pode analisar qualquer combinação de duas instruções de memória consecutivas, como leituras e escritas) que chegam ao L/S Pipe no mesmo ciclo. Teste tanto para leituras não-bloqueantes quanto para leituras bloqueantes. Pode usar o projeto PlatformIO fornecido em: *[RVfpgaPath]/RVfpga/Labs/Lab14/TwoConsecutiveLW\_Instructions*.

Duas leituras consecutivas:

210: 0002a303 lw t1,0(t0)

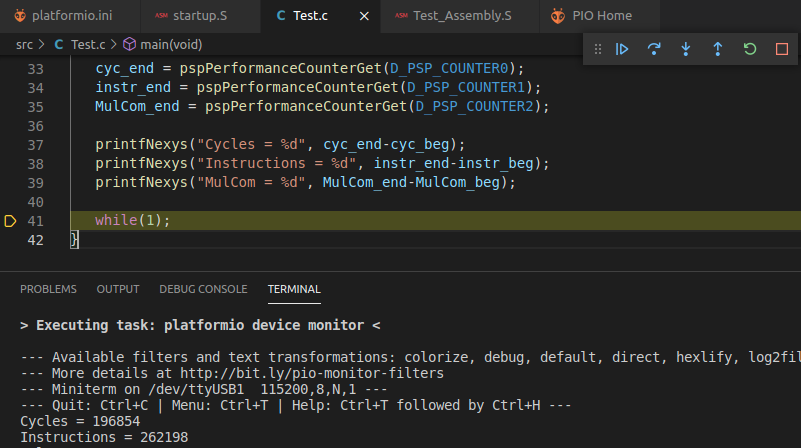
214: 0042a303 lw t1,4(t0)

* Simulação:



Devido ao conflito estrutural no Pipe L/S, o segundo lw deve ficar parado por 1 ciclo, de forma semelhante ao Pipe Mult que lida com duas instruções mul consecutivas.

* Execução na placa:



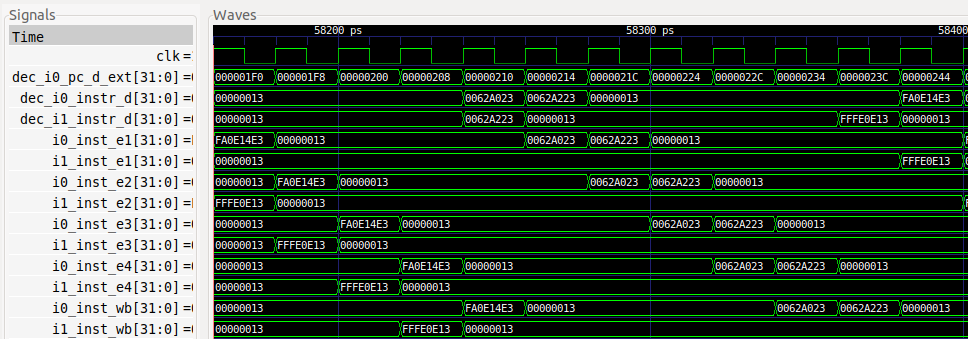
IPC = 262 / 196 = 1,33

Duas escritas consecutivas:

210: 0062a023 sw t1,0(t0)

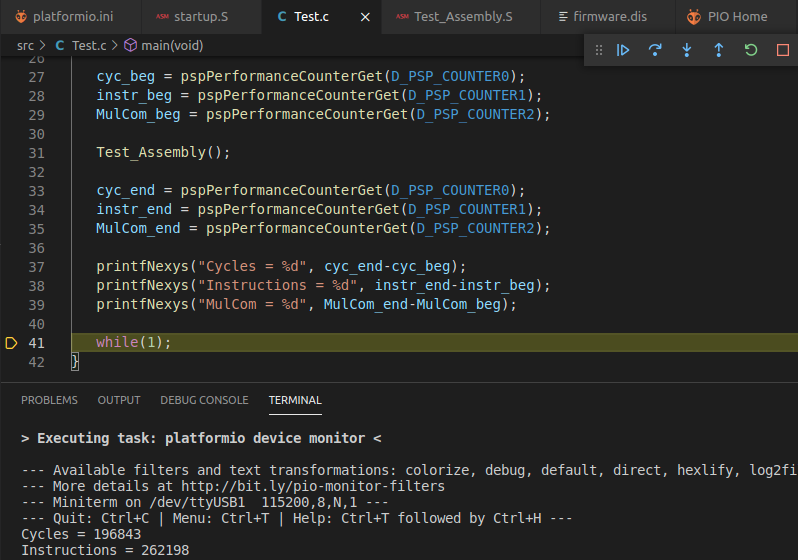
214: 0062a223 sw t1,4(t0)

* Simulação:



Devido ao conflito estrutural no L/S Pipe, o segundo sw deve ser parar por 1 ciclo, da mesma forma que o Pipe Mult lida com duas instruções mul consecutivas.

* Execução na placa:



IPC = 262 / 196 = 1,33

1. (*O exercício a seguir é baseado no exercício 4.22 do livro "Computer Organization and Design - RISC-V Edition", de Patterson & Hennessy ([HePa]*)*.*

Considere o fragmento do conjunto RISC-V abaixo:

sd x29, 12(x16)

ld x29, 8(x16)

sub x17, x15, x14

beqz x17, label

add x15, x11, x14

sub x15, x30, x14

Suponha que modifica o processador SweRV EH1 para que ele tenha apenas uma memória (que manipula instruções e dados). Nesse caso, haverá um conflito estrutural toda vez que um programa precisar carregar (Fetch) uma instrução durante o mesmo ciclo em que outra instrução acede a dados.

* 1. Desenhe um diagrama de pipeline para mostrar onde o código acima irá parar nessa versão imaginária do processador SweRV EH1.
  2. Em geral, é possível reduzir o número de paragens/nops resultantes desse conflito estrutural reordenando o código?
  3. Esse conflito estrutural deve ser tratado no hardware? Vimos que os conflitos de dados podem ser eliminados com a adição de nops ao código. Pode fazer o mesmo com esse conflito estrutural? Em caso afirmativo, explique como. Caso contrário, explique por que não.

Solução não fornecida.

# APÊNDICE A - DUAS INSTRUÇÕES DIV SIMULTÂNEAS NO ANDAR DECODE

**TAREFA:** Pode realizar um estudo semelhante para a instrução div como o realizado no Lab 12 para as instruções aritméticas-lógicas: veja o fluxo da instrução passando pelos andares do pipeline, analise os bits de controle (lembre-se do Apêndice D do Lab 11 que há um tipo de estrutura específica para a instrução div chamada div\_pkt\_t, e há um sinal definido no módulo **dec\_decode\_ctl** chamado div\_p), etc.

Solução não fornecida.

**TAREFA:** Inspecione o código Verilog do **exu\_div\_ctl** para entender como a divisão é computada. Analise também o efeito dos sinais div\_stall, finish\_early e finish. Como exercício opcional, substitua a Divide Unit por sua própria unidade ou por uma da Internet.

Solução não fornecida.

**TAREFA:** Verificar se esse par de 32 bits (0x03de42b3 e 0x03ff4333) corresponde às instruções div t0,t3,t4 e div t1,t5,t6 na arquitetura RISC-V.

**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**

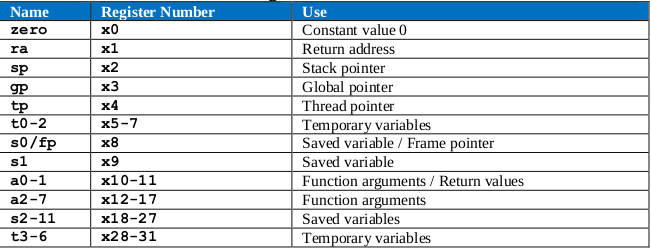
**rd = 00110 = x6 (t1)**

**op = 0110011**

Do Apêndice B do DDCARV:







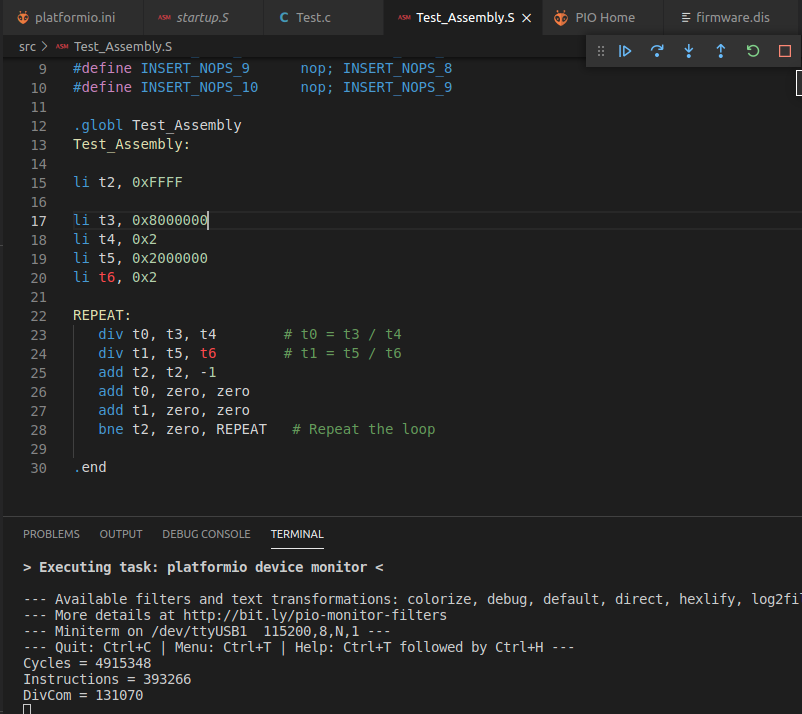
**TAREFA:** Replique a simulação da Figura 9 no seu computador e analise-a detalhadamente.

Solução fornecida no documento principal do Lab 14.

**TAREFA:** Compare a ilustração da Figura 10 e a simulação da Figura 9 que reproduziu no seu computador. Adicione sinais para ampliar a simulação e aprofundar a compreensão, conforme desejado.

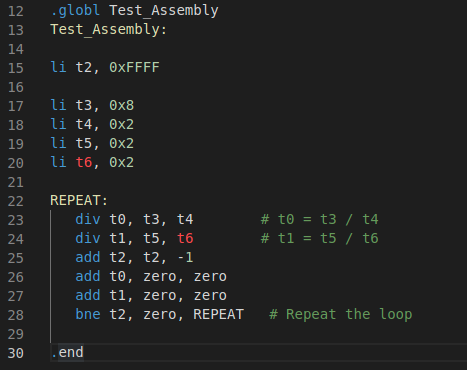
Solução não fornecida.

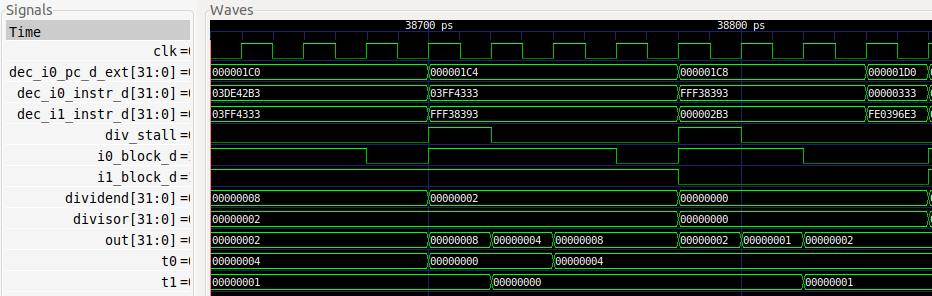
**TAREFA:** Meça diferentes eventos (ciclos, instruções/divisões completadas etc.) usando os contadores de desempenho disponíveis no SweRV EH1, conforme explicado no Lab 11. O número de ciclos está de acordo com o esperado após a análise da simulação da Figura 9? Justifique sua resposta.



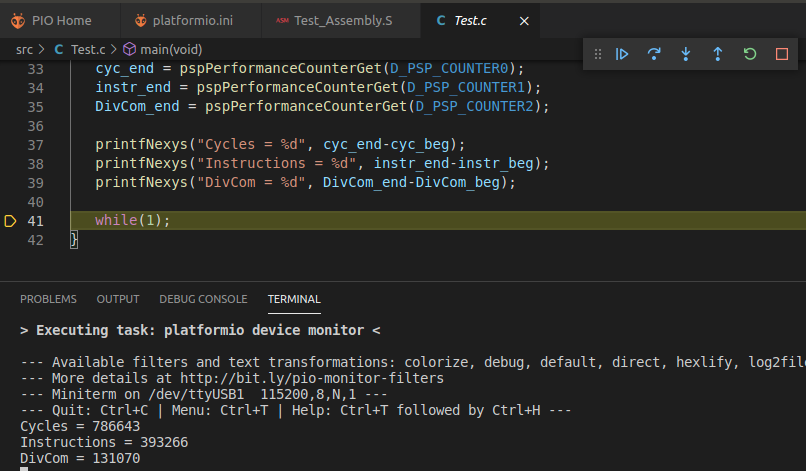
CPI = 4910000 / 393000 = 12. Levando em conta que cada divisão leva cerca de 34 ciclos para ser executada e que as outras instruções levam ½ ciclo cada, isso é aproximadamente o que poderíamos esperar: um cálculo teórico aproximado poderia ser: 6 instruções executadas em 34 + 34 + ½ + ½ + ½ + ½ + ½ ciclos🡪 CPI = 70 / 6 = 11

**TAREFA:** Experimente dividendos e divisores diferentes e veja como o número de ciclos para computar o resultado depende do valor deles. Veja o experimento na simulação e com os HW Counters.





Agora, as divisões são computadas em apenas cerca de 5 ciclos.



O CPI diminui muito (cerca de 2 por ciclo), já que o tempo para computar cada divisão também diminui muito.

**TAREFA:** A pasta *[RVfpgaPath]/RVfpga/Labs/Lab14/DIV\_Instr\_Accumul\_C-Lang* fornece o projeto PlatformIO de um programa em C que acumula a subtração de duas divisões num ciclo.

* Analisar o programa C.
* Execute uma simulação e inspecione uma iteração aleatória do ciclo. Observe que o programa em C é compilado sem otimizações.
* Meça diferentes eventos (ciclos, instruções/divisões completadas, etc.) usando os contadores de desempenho disponíveis no SweRV EH1, conforme explicado no Lab 11.

O número de ciclos está de acordo com o esperado após a análise da simulação da Figura 9? Justifique sua resposta.

* Crie um programa análogo em Assembly RISC-V e compare-o com a versão em C.
* Desative a extensão **M** RISC-V no programa C e compare os resultados com o programa original. Para fazer isso, modifique a seguinte linha no ficheiro *platformio.ini* de:

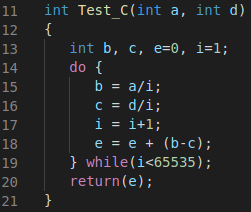
build\_flags = -Wa,-march=rv32**ima** -march=rv32**ima**

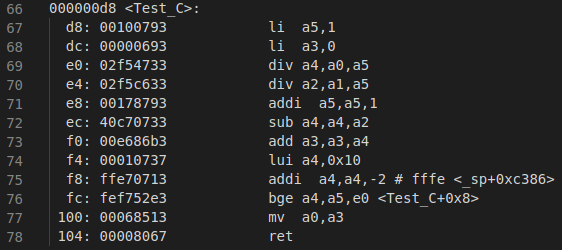
Para:

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

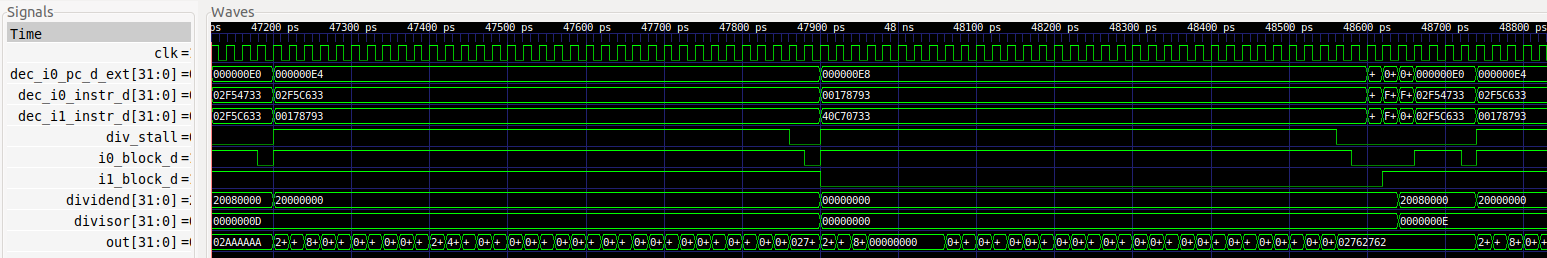
Isso evita o uso das instruções da extensão RISC-V M e as emula usando outras instruções.

* Programa C (original e desmontagem):

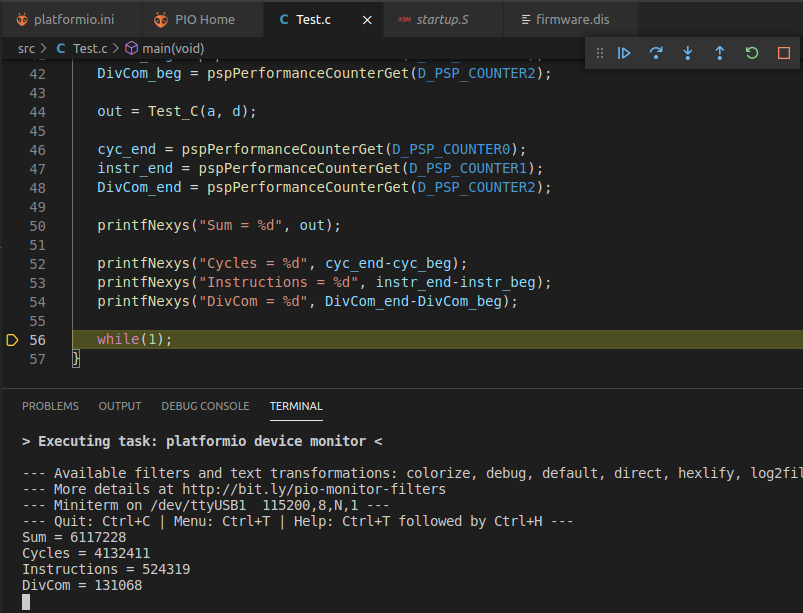




* Simulação do programa C:

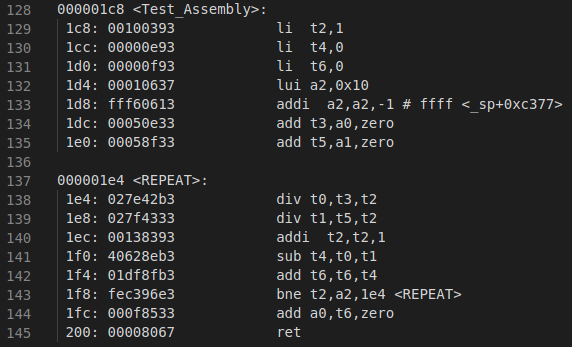


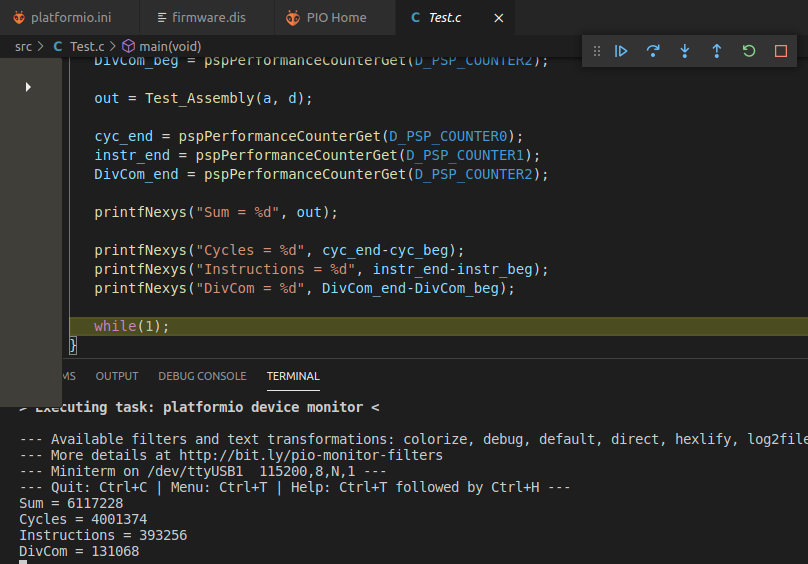
* Contadores HW:



* O programa Assembly pode ser encontrado em:

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



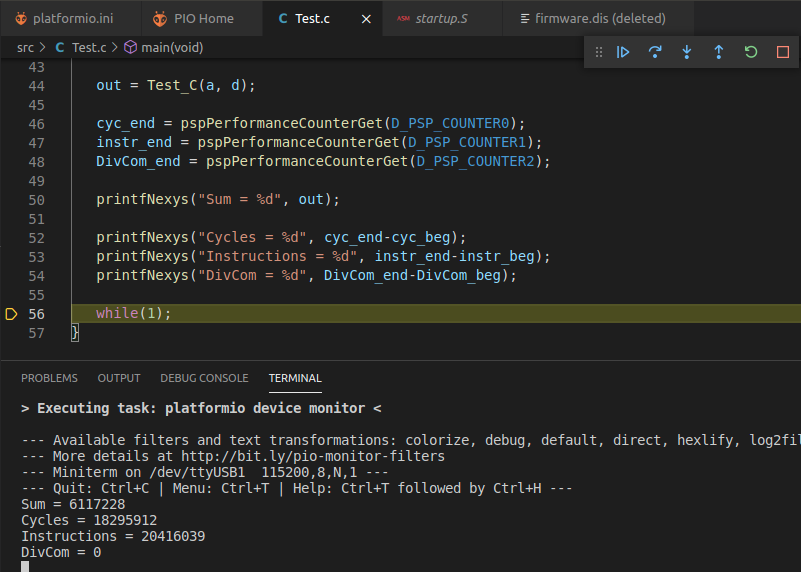


O resultado da soma é o mesmo, pois o programa é o mesmo.

O número de ciclos é um pouco menor, pois a versão Assembly programada manualmente é mais eficiente do que a obtida pelo compilador sem otimizações.

O número de instruções também é um pouco menor.

* Desativar a extensão M:



O resultado da soma é o mesmo, pois o programa é o mesmo.

O número de ciclos é muito maior: Cerca de 18 milhões contra cerca de 4 milhões.

O número de instruções também é muito maior: Cerca de 20 milhões contra cerca de 0,5 milhão.

O IPC está melhor agora.

Não há divisões completadas.

**TAREFA:** No SweRV EH1, as instruções div são bloqueantes. Modifique o processador para permitir instruções div não-bloqueantes.

Em seguida, adicione um segundo divisor ao processador SweRV EH1, de modo que duas instruções div do exemplo da Figura 8 possam ser executadas em paralelo.

Solução não fornecida.