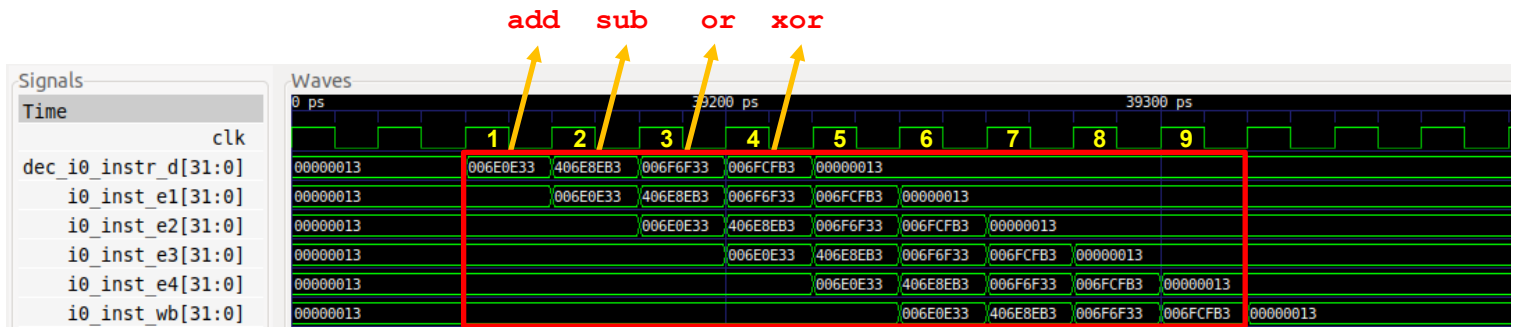
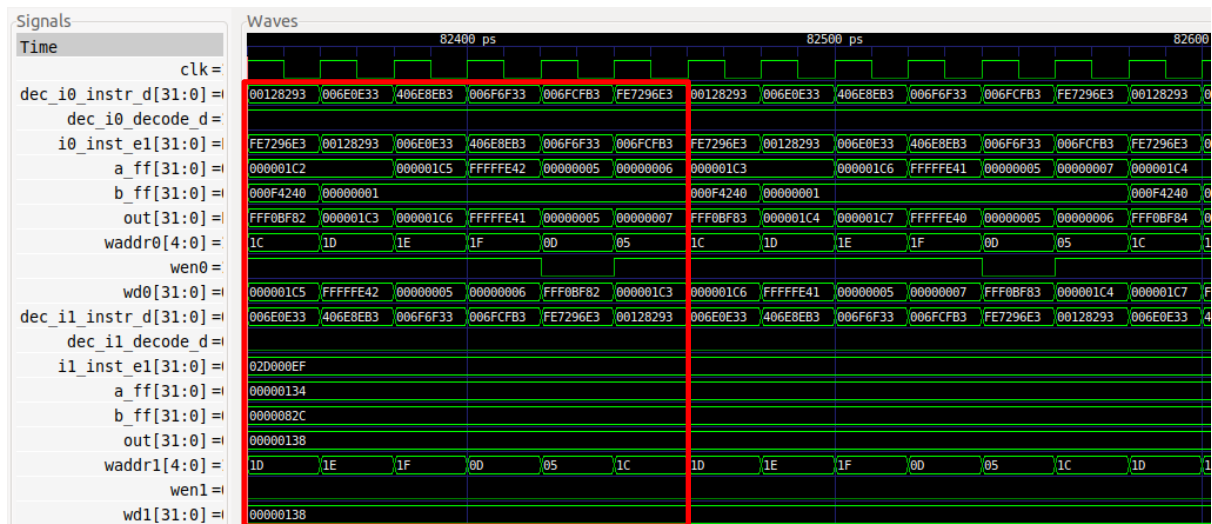


## TAREFAS

**TAREFA:** Na simulação da Figura 3, inclua os sinais de trace e destaque as instruções à medida que passam pelo pipeline desde o andar Decode até ao andar Writeback, à semelhança da Figura 4. Pode utilizar o ficheiro .tcl fornecido em: `[RVfpgaPath]/RVfpga/Labs/Lab17/Four_AL_Instructions/test_TAREFA1.tcl`.



**TAREFA:** Remova todas as instruções `nop` dentro do corpo do ciclo da Figura 2. Repita a simulação da Figura 3. Qual é o IPC esperado para este programa? Execute o programa na placa e verifique se o IPC obtido é o esperado.



As 6 instruções do ciclo necessitam de 6 ciclos de relógio para serem executadas. Assim, o IPC esperado é de 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 <

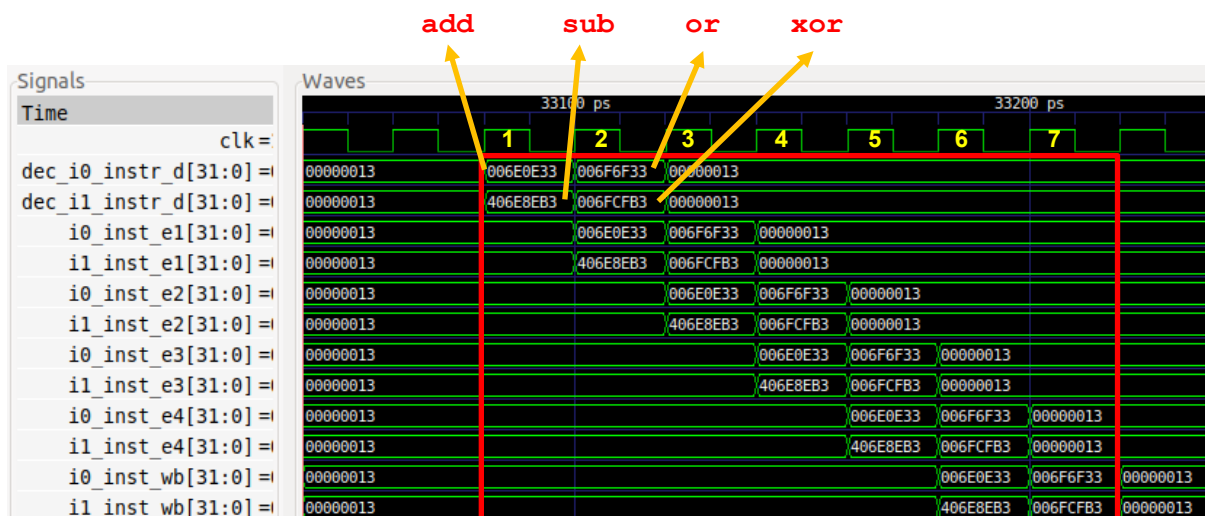
--- Available filters and text transformations: colorize, debug, ...  
 --- More details at <http://bit.ly/pio-monitor-filters>  
 --- Miniterm on /dev/ttyUSB1 115200,8,N,1 ---  
 --- Quit: Ctrl+C | Menu: Ctrl+T | Help: Ctrl+T followed by Ctrl+H

Cycles = 6000276  
 Instructions = 6000048

Como esperado:

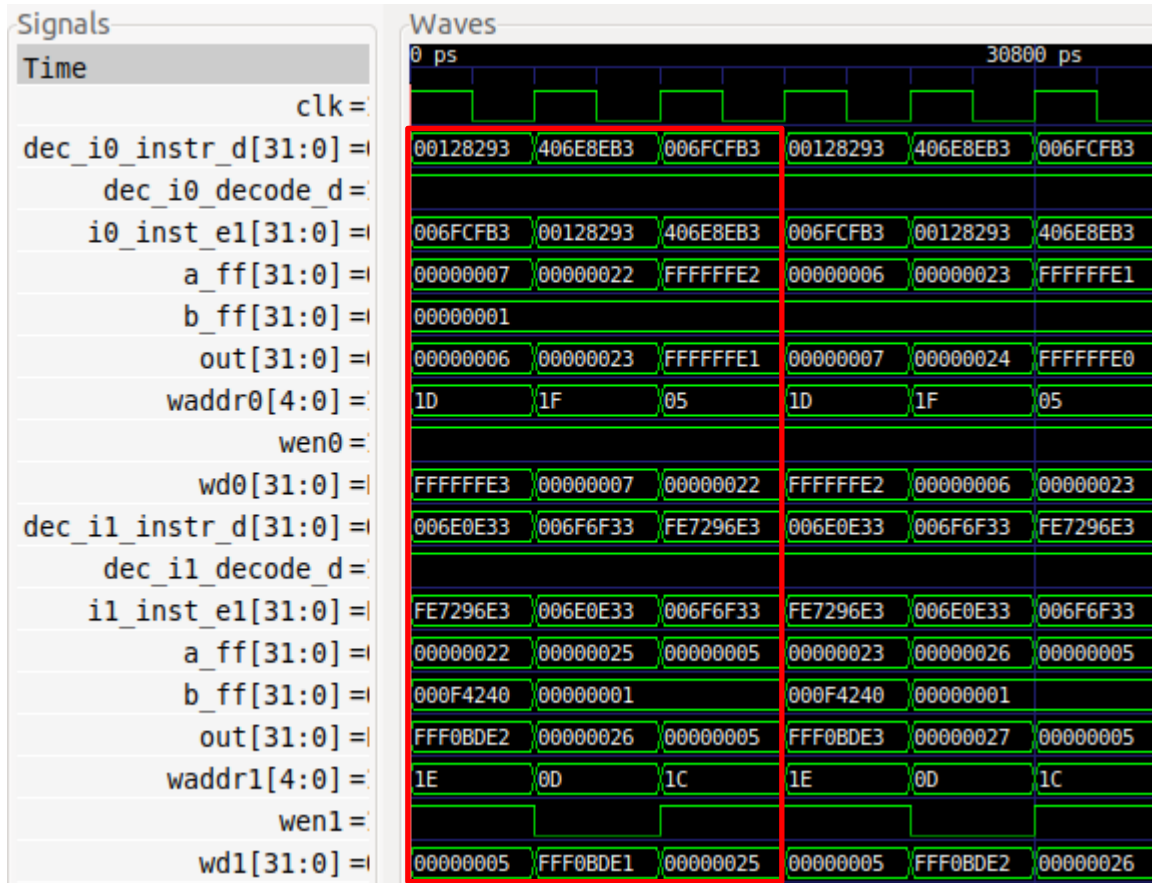
$$IPC = 6000000 \text{ instruções} / 6000000 \text{ ciclos} = 1$$

**TAREFA:** Na simulação da Figura 5, adicione sinais trace e destaque as instruções à medida que passam pelo pipeline, desde o andar Decode até ao Writeback, à semelhança do que é mostrado na Figura 6. Pode utilizar o ficheiro `.tcl` fornecido em: `[RVfpgaPath]/RVfpga/Labs/Lab17/Four_AL_Instructions/test_TAREFA2.tcl`.



**TAREFA:** Remova todas as instruções `nop` dentro do corpo do ciclo da Figura 2.

Repita a simulação da Figura 5. Qual é o IPC esperado para este programa?  
Execute o programa na placa e verifique se o IPC obtido é o esperado.



As 6 instruções dentro do ciclo precisam de 3 ciclos para serem executadas. Assim, o IPC esperado é de 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 <

```

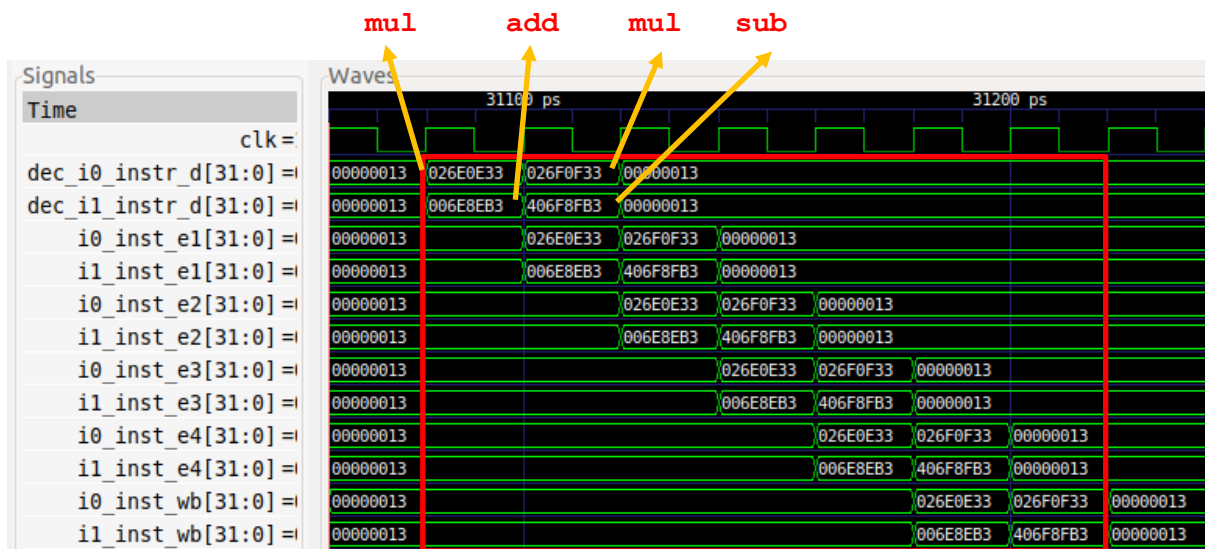
--- Available filters and text transformations: colorize, debug, d
--- More details at http://bit.ly/pio-monitor-filters
--- Miniterm on /dev/ttyUSB1 115200,8,N,1 ---
--- Quit: Ctrl+C | Menu: Ctrl+T | Help: Ctrl+T followed by Ctrl+H
Cycles = 3000253
Instructions = 6000046

```

Como esperado:

$$IPC = 6000000 \text{ instruções} / 3000000 \text{ ciclos} = 2$$

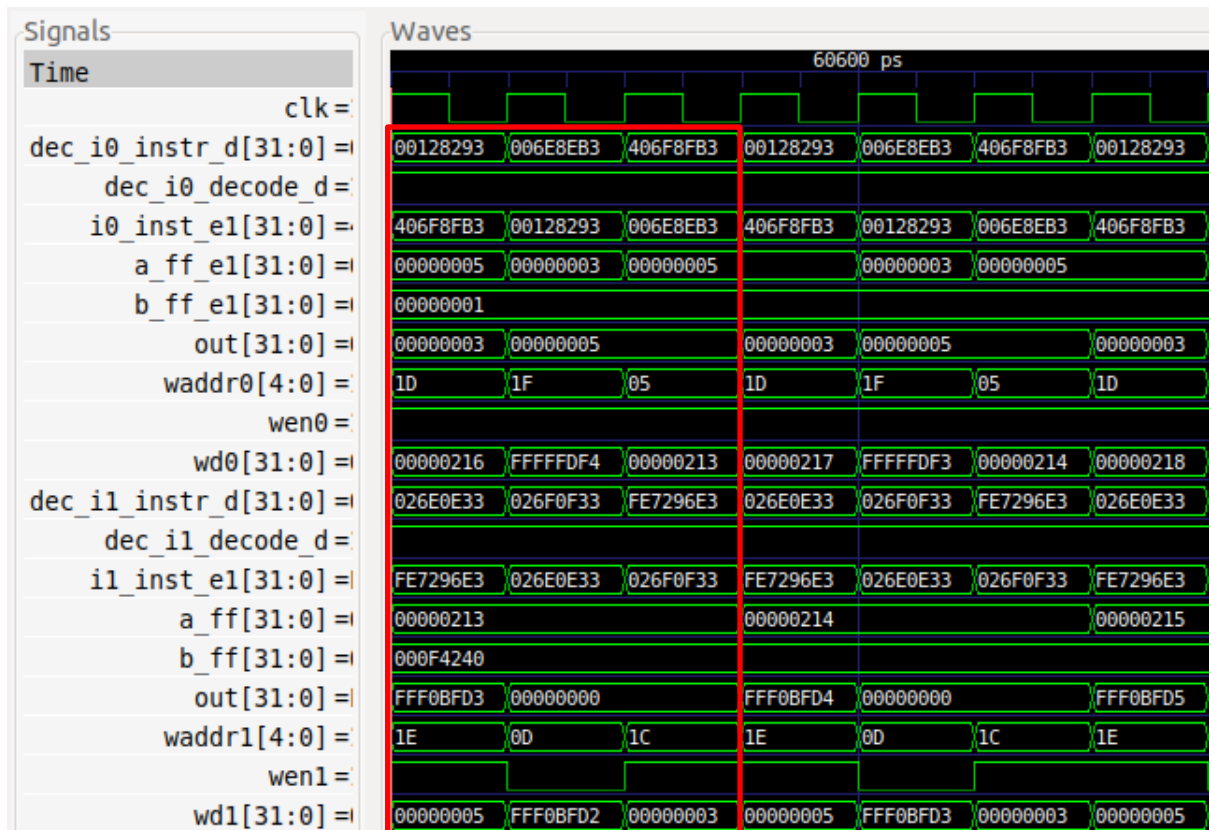
**TAREFA:** Na simulação da Figura 8, adicione sinais trace e destaque as instruções à medida que elas passam pelo pipeline, desde os andares de Decode até Writeback. Pode utilizar o ficheiro .tcl file fornecido em:  
`[RVfpgaPath]/RVfpga/Labs/Lab17/TwoAL_TwoMUL_Instructions/test_TAREFAMuls.tcl`.



**TAREFA:** Remova todas as instruções `nop` dentro do corpo do ciclo da Figura 7. Repita a simulação da Figura 8. Qual é o IPC esperado para este programa? Execute o programa na placa e verifique se o IPC obtido é o esperado.

Repita as mesmas experiências para a configuração single-issue e compare os resultados.

### DUAL ISSUE:



As 6 instruções dentro do ciclo precisam de 3 ciclos de relógio para serem executadas. Assim, o IPC esperado é de 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 <

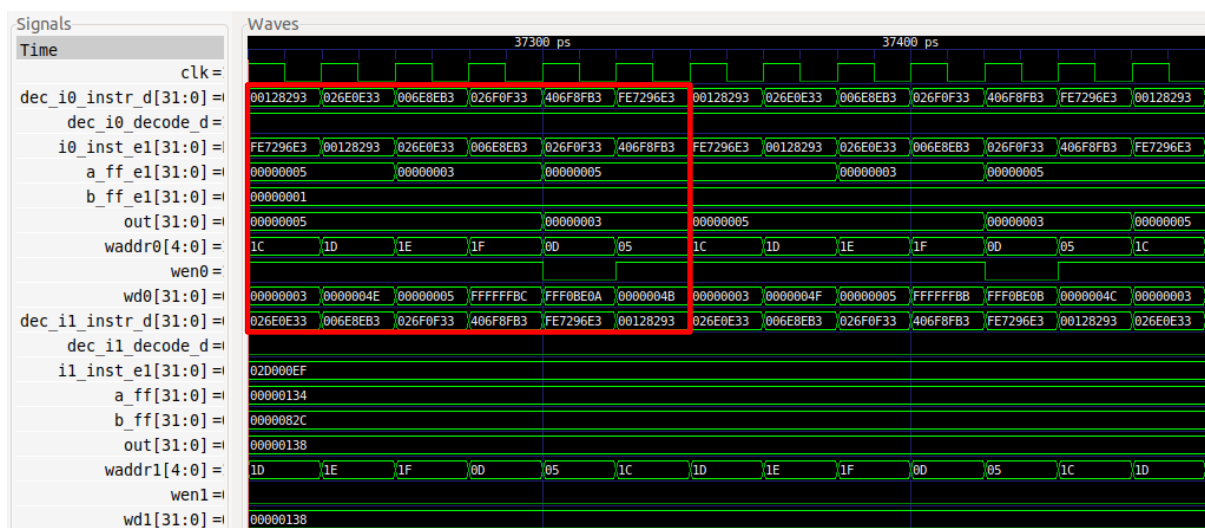
--- Available filters and text transformations: colorize, debug, d
--- More details at http://bit.ly/pio-monitor-filters
--- Miniterm on /dev/ttyUSB1 115200,8,N,1 ---
--- Quit: Ctrl+C | Menu: Ctrl+T | Help: Ctrl+T followed by Ctrl+H
Cycles = 3000263
Instructions = 6000046

```

Como esperado:

$$\text{IPC} = 6000000 \text{ instruções} / 3000000 \text{ ciclos} = 2$$

## SINGLE ISSUE:



As 6 instruções do ciclo necessitam de 6 ciclos de relógio para serem executadas. Assim, o IPC esperado é de 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

```

Como esperado:

$$\text{IPC} = 6000000 \text{ instruções} / 6000000 \text{ ciclos} = 1$$

## EXERCÍCIOS

- 1) Crie programas semelhantes aos das figuras 2 e 7 usando combinações de instruções que mostrem novas situações relacionadas com a execução dual-issue.

Solução não fornecida.

- 2) Analise as diferenças entre o processador SweRV EH1 (dual-issue) e o exemplo de processador superescalar proposto na Secção 7.7.4 do livro de S. Harris e D. Harris, “Digital Design and Computer Architecture: RISC-V Edition” [DDCARV] (mostrado na Figura 1 por conveniência).

Solução não fornecida.

- 3) Analise o programa da Figura 7.70 na Secção 7.7.4 do DDCARV, que é fornecido num projeto PlatformIO na pasta *[RVfpgaPath]/RVfpga/Labs/Lab17/DDCARV\_SuperscalarExample*. Execute o programa no SweRV EH1, tanto em simulação como na placa (para esta última, remova as instruções nop). Explique os resultados. Se necessário, reordene o programa tentando obter o IPC ótimo.

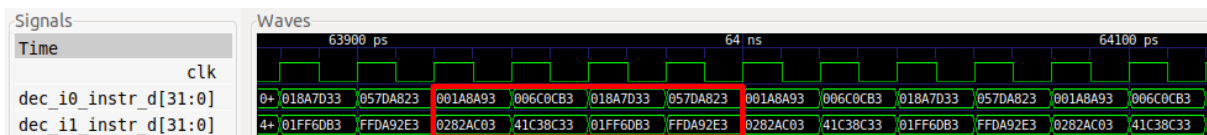
A seguir, desactive a execução dual-issue como explicado neste laboratório - e no *SweRVref.docx* (Secção 2). Compare a simulação e os resultados obtidos na placa quando comparados a quando o recurso dual-issue está ativado.

Em seguida, desactive a execução dual-issue conforme explicado neste laboratório – e no SweRVref.docx (Secção 2). Compare a simulação e os resultados obtidos na placa com quando a funcionalidade dual-issue está activada.

000001c0 <REPEAT>:

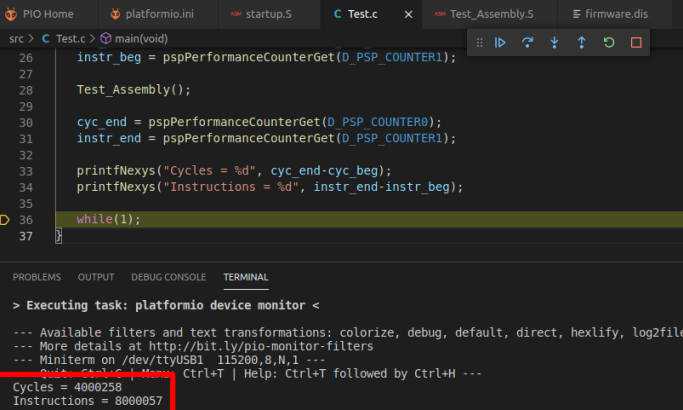
```
1c0: 001a8a93      addi    s5,s5,1
1c4: 0282ac03      lw      s8,40(t0)
1c8: 006c0cb3      add     s9,s8,t1
1cc: 41c38c33      sub     s8,t2,t3
1d0: 018a7d33      and     s10,s4,s8
1d4: 01ff6db3     or      s11,t5,t6
1d8: 057da823     sw      s7,80(s11)
1dc: ffd9a92e3     bne     s5,t4,1c0 <REPEAT>
```

### Simulação – Dual-Issue:



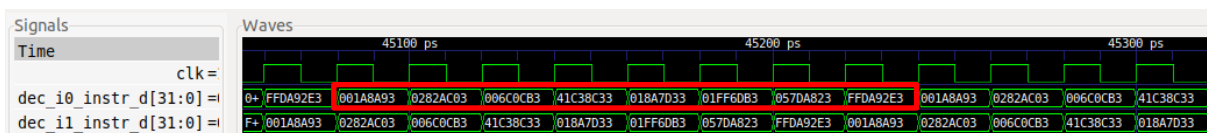
Não há paragens no ciclo e são sempre executadas 2 instruções por ciclo. São efetuados alguns bypasses e, em alguns casos, é utilizada a ALU secundária, tal como explicado em laboratórios anteriores. Pode analisar melhor estas situações.

### Resultados na placa – Dual-Issue:



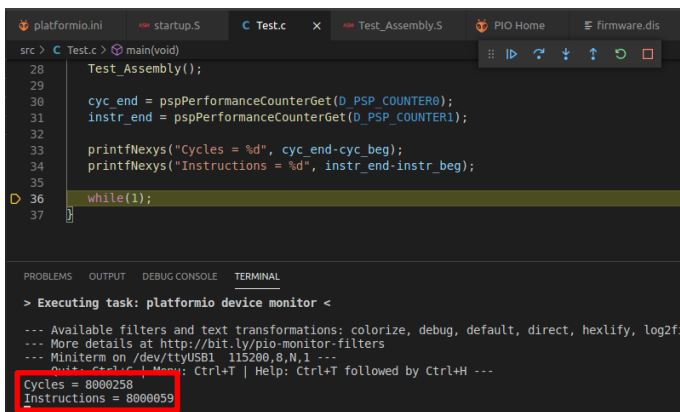
O IPC é igual a 2, sem necessidade de reordenar o código.

### Simulação – Single-Issue:



Não há paragens no ciclo e é sempre executada uma instrução por ciclo de relógio.

### Resultados na placa – Single-Issue:



```

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 = 8000258
Instructions = 8000059

```

O IPC é igual a 1, que é o melhor que pode obter num SweRV EH1 de single-issue.

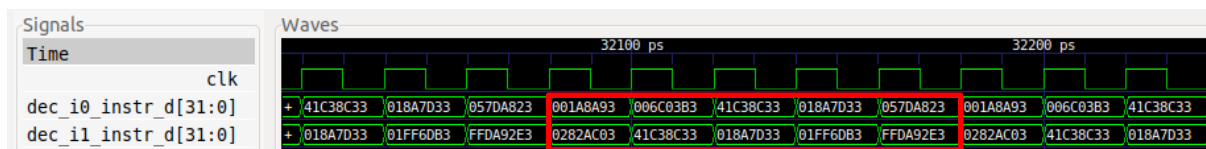
- 4) Modifique o programa do EXERCÍCIO 3 substituindo a instrução `add s9, s8, t1` pela instrução `add t2, s8, t1`. Explique os resultados. Se necessário, reordene o programa para tentar obter o IPC ótimo.

Em seguida, desactive a execução dual-issue conforme explicado neste laboratório e em *SweRVref.docx* (Secção 2). Compare a simulação e os resultados obtidos na placa quando comparados com quando a funcionalidade dual-issue está activada.

000001c0 <REPEAT>:

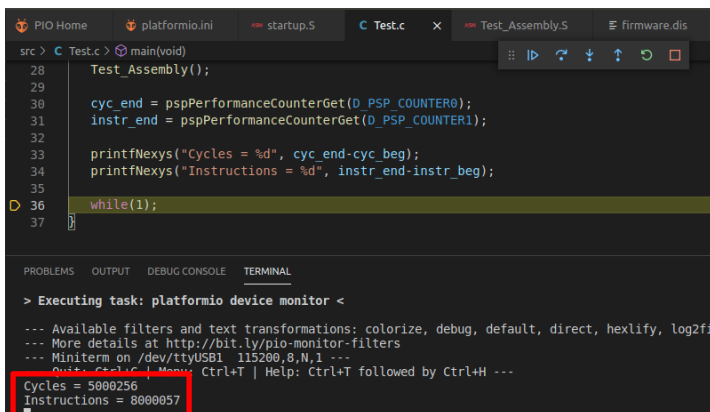
1c0:	001a8a93	addi	s5,s5,1
1c4:	0282ac03	lw	s8,40(t0)
1c8:	006c03b3	add	t2,s8,t1
1cc:	41c38c33	sub	s8,t2,t3
1d0:	018a7d33	and	s10,s4,s8
1d4:	01ff6db3	or	s11,t5,t6
1d8:	057da823	sw	s7,80(s11)
1dc:	ffda92e3	bne	s5,t4,1c0 <REPEAT>

### Simulação – Dual-Issue:



Existem 2 paragens no ciclo devido às dependências entre as instruções AL.

### Resultados na placa – Dual-Issue:



```

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 = 5000256
Instructions = 8000057

```

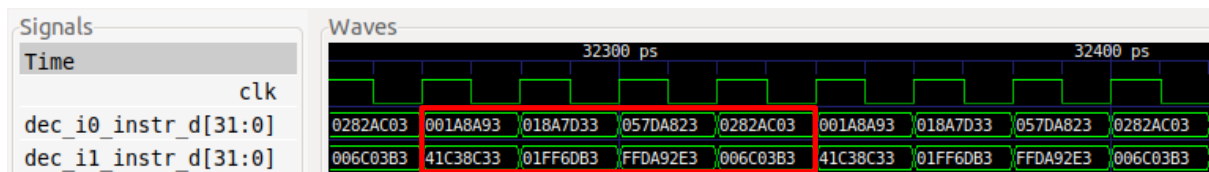
O IPC é inferior a 2 devido aos impasses.

### Código reordenado:

000001c0 <REPEAT>:

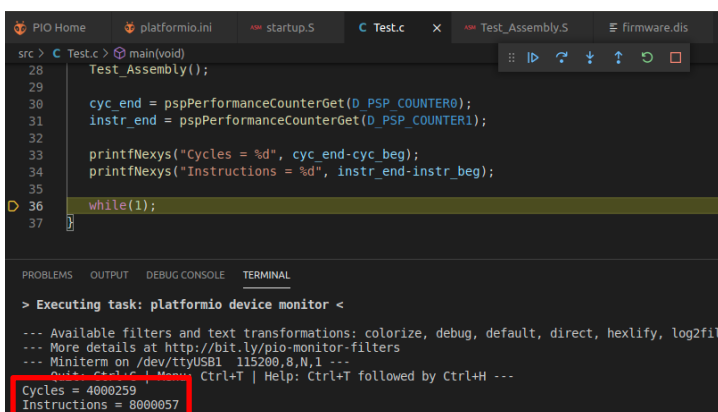
1c0:	0282ac03	lw	s8,40(t0)
1c4:	006c03b3	add	t2,s8,t1
1c8:	001a8a93	addi	s5,s5,1
1cc:	41c38c33	sub	s8,t2,t3
1d0:	018a7d33	and	s10,s4,s8
1d4:	01ff6db3	or	s11,t5,t6
1d8:	057da823	sw	s7,80(s11)
1dc:	ffda92e3	bne	s5,t4,1c0 <REPEAT>

### Simulação – Dual-Issue:



Não existem paragens no ciclo.

### Resultados na placa – Dual-Issue:



```

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

```

O IPC é 2.

## De volta ao programa original:

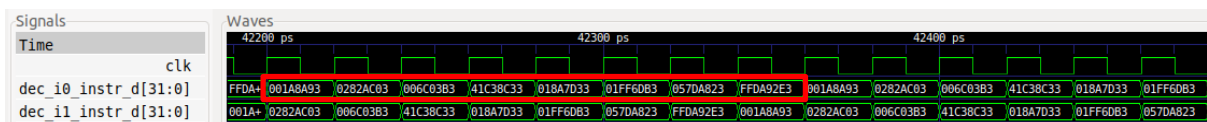
000001c0 <REPEAT>:

```

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>

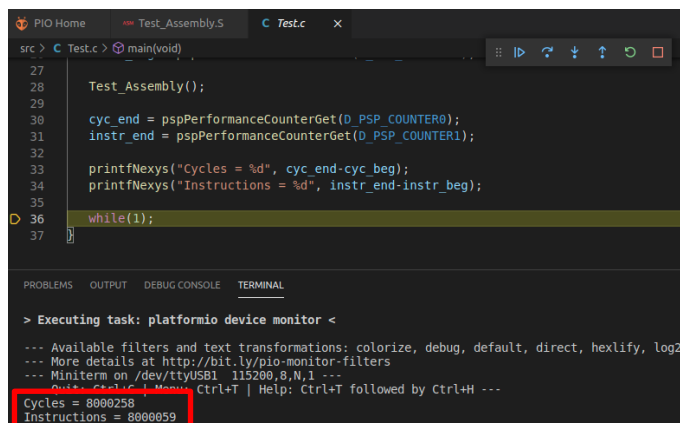
```

## Simulação – Single-Issue:



Não existem paragens no ciclo.

## Results na placa – Single-Issue:



```

src > C 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 ---
Ctrl+Q to send | Ctrl+Z to stop | Ctrl+C to quit | Help: Ctrl+T followed by Ctrl+H ---
Cycles = 8000258
Instructions = 8000059

```

O IPC é 1.

5) (O seguinte exercício é baseado no exercício 4.31 do livro “Computer Organization and Design – RISC-V Edition”, by Patterson & Hennessy ([HePa]).)

Neste exercício comparamos o desempenho de processadores single-issue com dual-issue, tendo em conta as transformações de programa que podem ser feitas para otimizar a execução dual-issue. Os problemas deste exercício referem-se ao seguinte ciclo (escrito em C):

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

Um compilador com pouca ou nenhuma otimização pode produzir o seguinte código Assembly RISC-V:

```
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

```

Este código utiliza os seguintes registos:

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

Este código é dado em *[RVfpgaPath]/RVfpga/Labs/Lab17/PaHe\_SuperscalarExample* com algumas pequenas modificações em relação ao código fornecido pelo exercício do livro que não afetam o comportamento do programa:

- O registo `x13` é inicializado em 8, para que o ciclo efectue 4 iterações.
- A instrução `jal` é removida.
- As instruções `ld` e `sd` são substituídas pelas instruções `lw` e `sw`. Isto implica a alteração dos acessos de 4 para 8 bytes de largura.

Suponha um processador dual-issue, com agendamento estático, que tem as seguintes propriedades:

1. Uma instrução deve ser uma operação de memória; a outra deve ser uma instrução aritmética/lógica ou um branch.
2. O processador tem todos os caminhos de forwarding possíveis entre andares.
3. O processador tem uma previsão de branch perfeita.
4. Duas instruções não podem ser emitidas (issued) em conjunto se uma depender da outra.
5. Se for necessário um stall, ambas as instruções num andar têm de parar.

- a) Compare as propriedades deste processador de exemplo e as propriedades do processador SweRV EH1.
- b) Desenhe um diagrama do pipeline e uma simulação mostrando como uma iteração aleatória do ciclo (exceto a primeira) do código RISC-V dado acima é executada no processador SweRV EH1 dual-issue. Suponha que o ciclo termina ao fim de quatro mil iterações (é o que acontece no código acima).
- c) Qual é o aumento de velocidade ao passar de um processador SweRV EH1 de single-issue para um processador SweRV EH1 de dual-issue? Explique os resultados. Teste o programa na placa e ative/desative a execução dual-issue.
- d) Reorganize/reescreva o código RISC-V dado acima para obter um melhor desempenho no processador SweRV EH1 de duas emissões. (No entanto, não desenrole o ciclo).
- e) Agora, desenrole o código RISC-V de forma que cada iteração do ciclo desenrolado manipule duas iterações do ciclo original. Em seguida, reorganize/reescreva o seu código desenrolado para obter um melhor desempenho no processador SweRV EH1

de dual-issue.

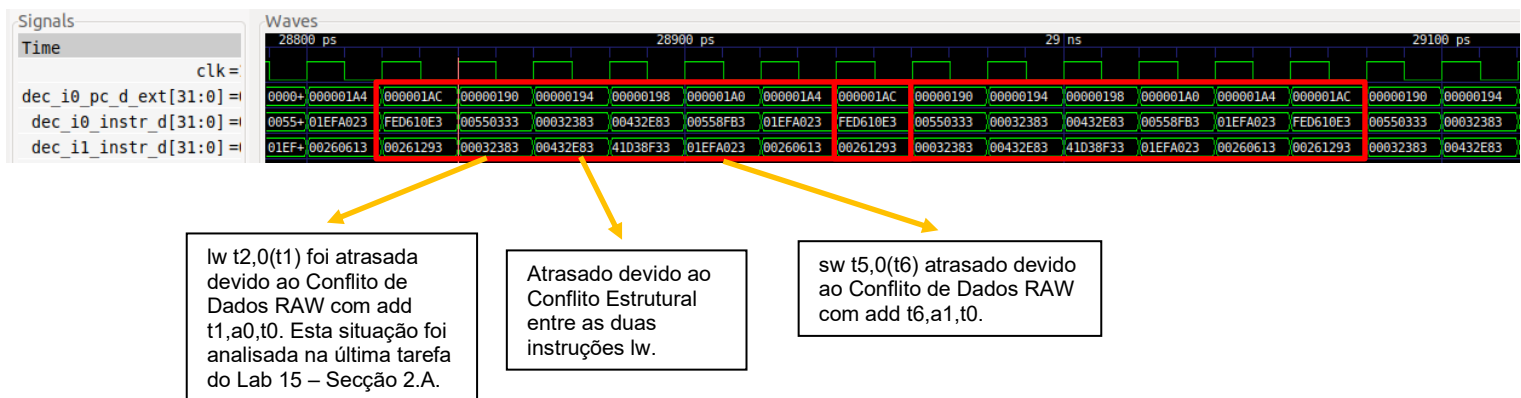
b)

0000018c <TOP>:

```

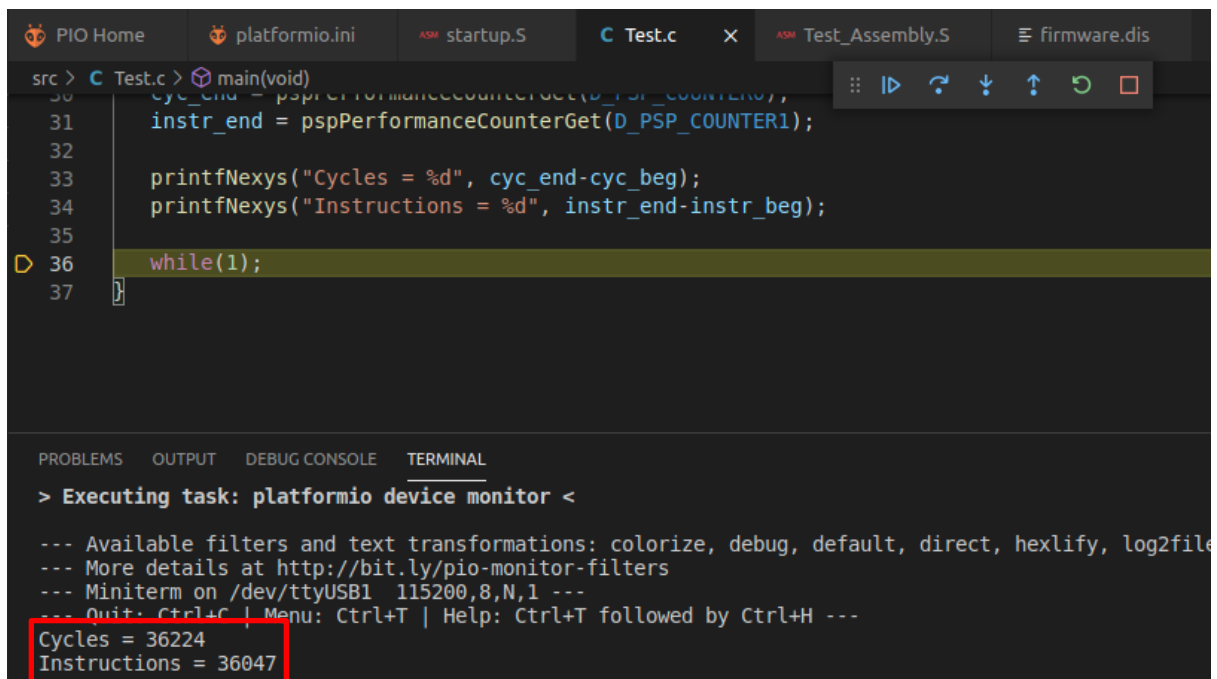
18c: 00261293      slli    t0,a2,0x2
190: 00550333      add     t1,a0,t0
194: 00032383      lw      t2,0(t1)
198: 00432e83      lw      t4,4(t1)
19c: 41d38f33      sub     t5,t2,t4
1a0: 00558fb3      add     t6,a1,t0
1a4: 01efa023      sw      t5,0(t6)
1a8: 00260613      addi    a2,a2,2

```



c)

Single Issue:



The screenshot shows a code editor with a C program and a terminal window. The C program is as follows:

```

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

```

The terminal window shows the output of the program:

```

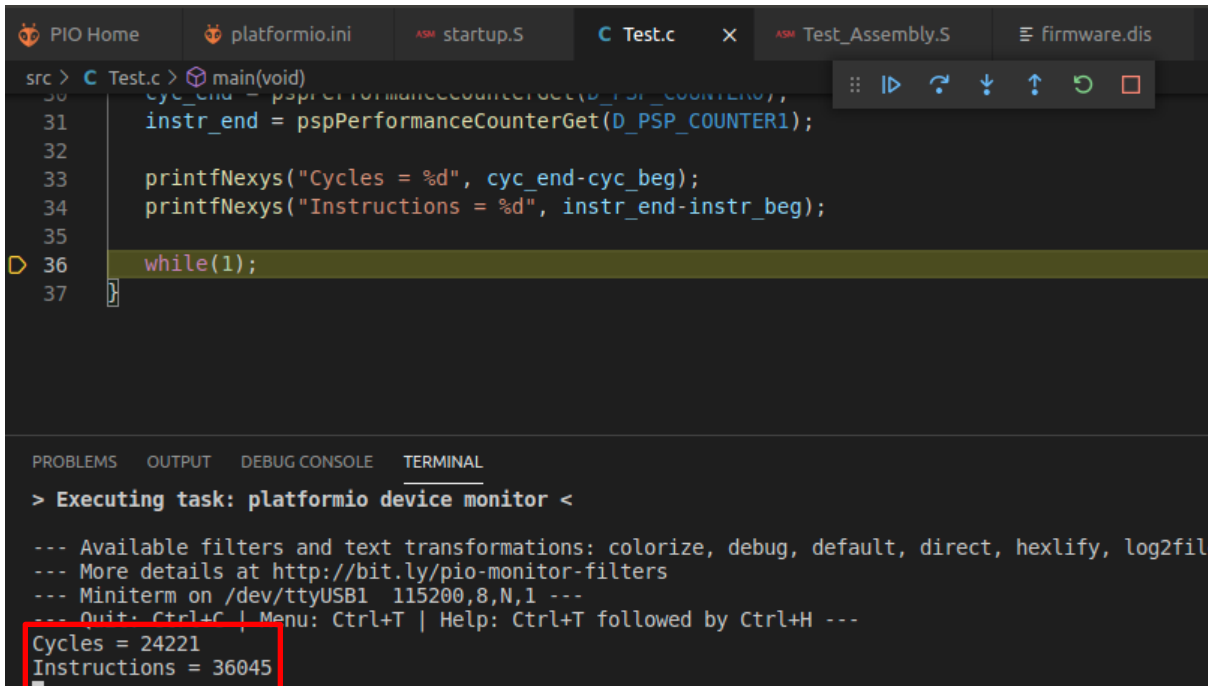
> Executing task: platformio device monitor <

--- 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

```

## Dual Issue:



The screenshot shows the PlatformIO IDE with a C file named `Test.c` open. The code defines a `main` function that uses the `pspPerformanceCounterGet` function to measure execution time and instructions. It prints the results using `printfNexys` and enters an infinite `while(1);` loop. The bottom panel shows the 'TERMINAL' output, which displays the results of the execution: 'Cycles = 24221' and 'Instructions = 36045'. These values are highlighted with a red box in the original image.

```
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, 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
```

$SPEEDUP = \text{Time\_Single} / \text{Time\_Dual} = 36224 / 24221 \approx 1.5$  (o speedup ideal seria 2, mas devido aos stalls analisados em (b), é apenas de 1.5)

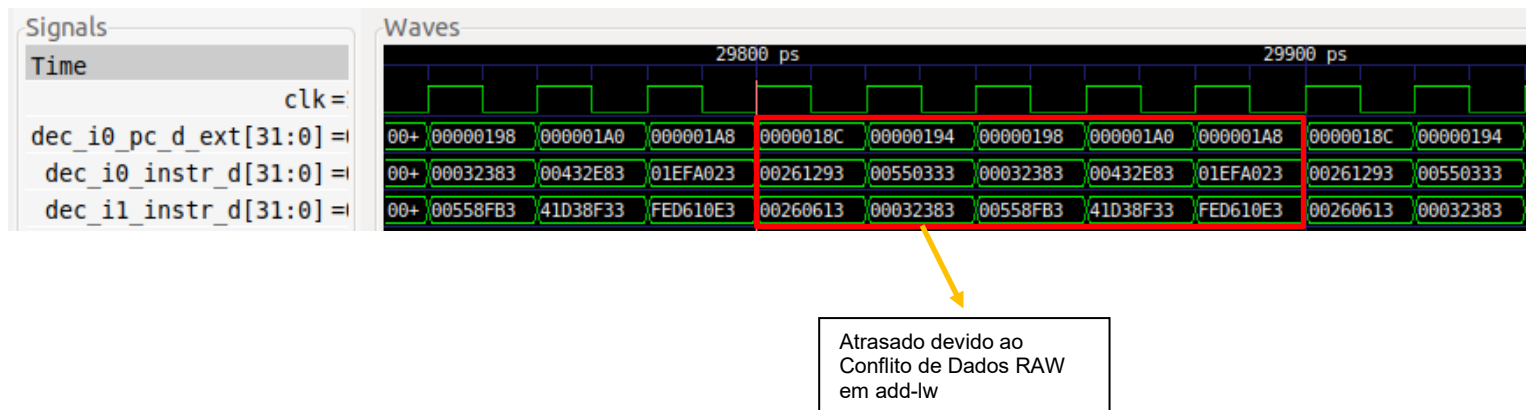
d)

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)

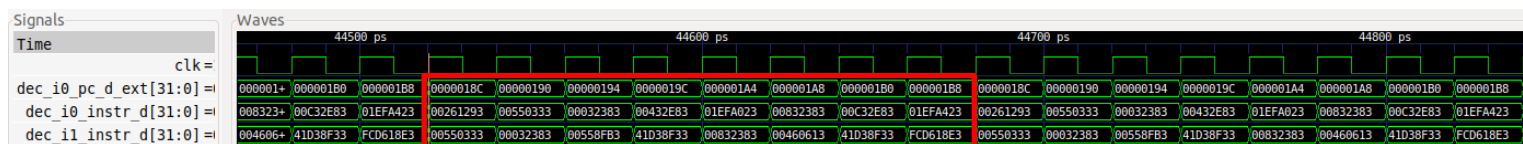
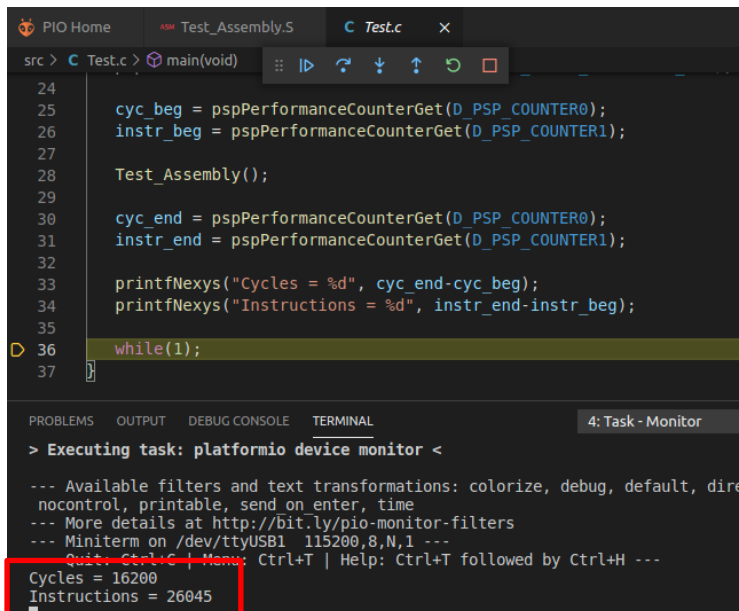
TOP:

```
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
```

Graças ao desenrolar e à reescrita, o desempenho aumenta ainda mais.

6) (O seguinte exercício é baseado em exercícios 7.30, 7.32 e 7.34 do Capítulo 7 de DDCARV.)

Suponha que o processador SweRV EH1 está a executar o seguinte trecho de código. Lembre-se que o SweRV EH1 tem uma Unidade de Conflito. Pode assumir um sistema de memória que devolve o resultado num ciclo de relógio (para esse efeito, utilizamos o DCCM e inserimos o trecho de código num ciclo e evitamos a primeira iteração para que não haja misses na I\$).

```
addi s1, t0, 11    # t0 contém o endereço de base da DCCM
lw   s2, 25(s1)
lw   s5, 16(s2)
add  s3, s2, s5
or   s4, s3, t4
and  s2, s3, s4
```

a) Simule o programa com o Verilator e o GTKWave. Analise os resultados e, para cada ciclo, especifique

- \* Que instruções são DECODED, ISSUED para execução e COMMITED?
- \* Quais os registos que estão a ser escritos e quais os que estão a ser lidos?
- \* Que forwarding e stalls acontecem?

a) Qual é o IPC do processador neste programa? Primeiro responda teoricamente e depois confirme a sua resposta executando o programa na placa.

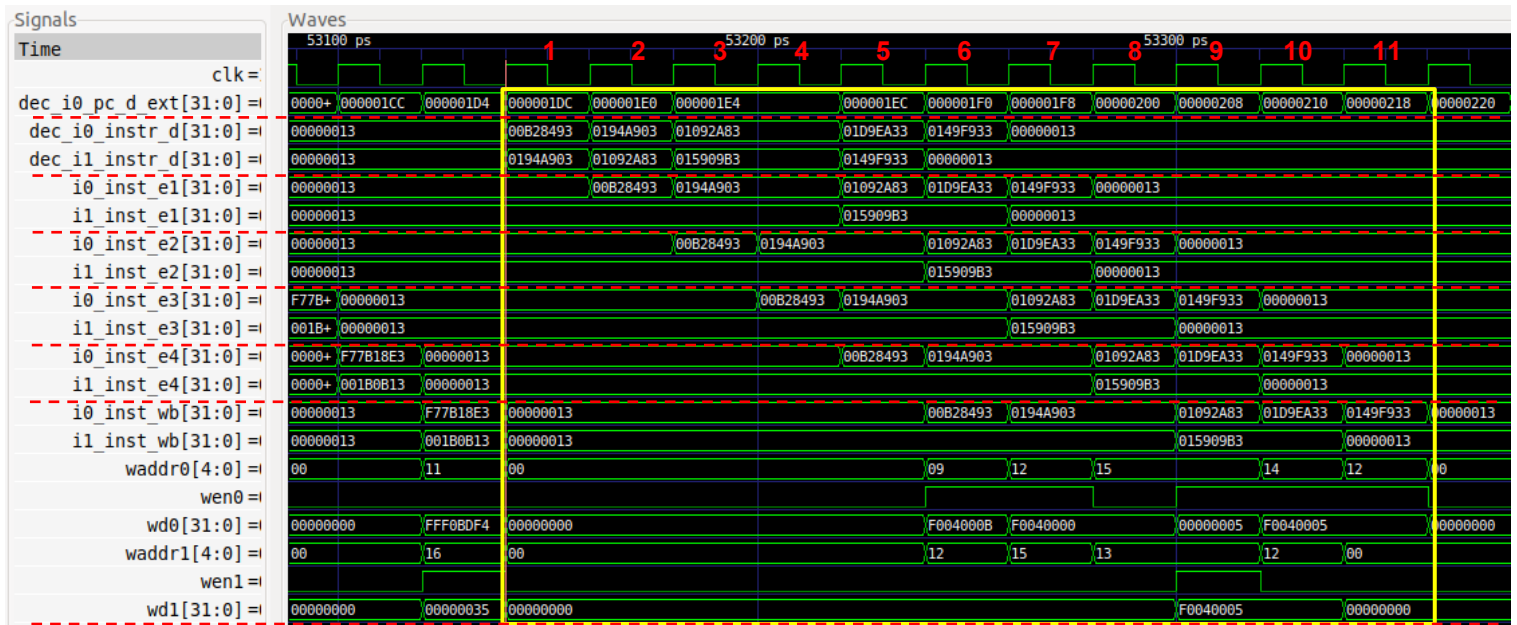
b) Faça a mesma análise no processador single-issue e compare os resultados com os resultados do processador dual-issue.

Um projeto PlatformIO é fornecido em:

*[RVfpgaPath]/RVfpga/Labs/Lab17/DDCARV\_EXERCÍCIOS-30-32-34*. O programa em análise é inserido num ciclo para que seja mais fácil de compreender na simulação (qualquer iteração, exceto a primeira, pode ser utilizada para análise) e pode ser medido utilizando contadores de desempenho.

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

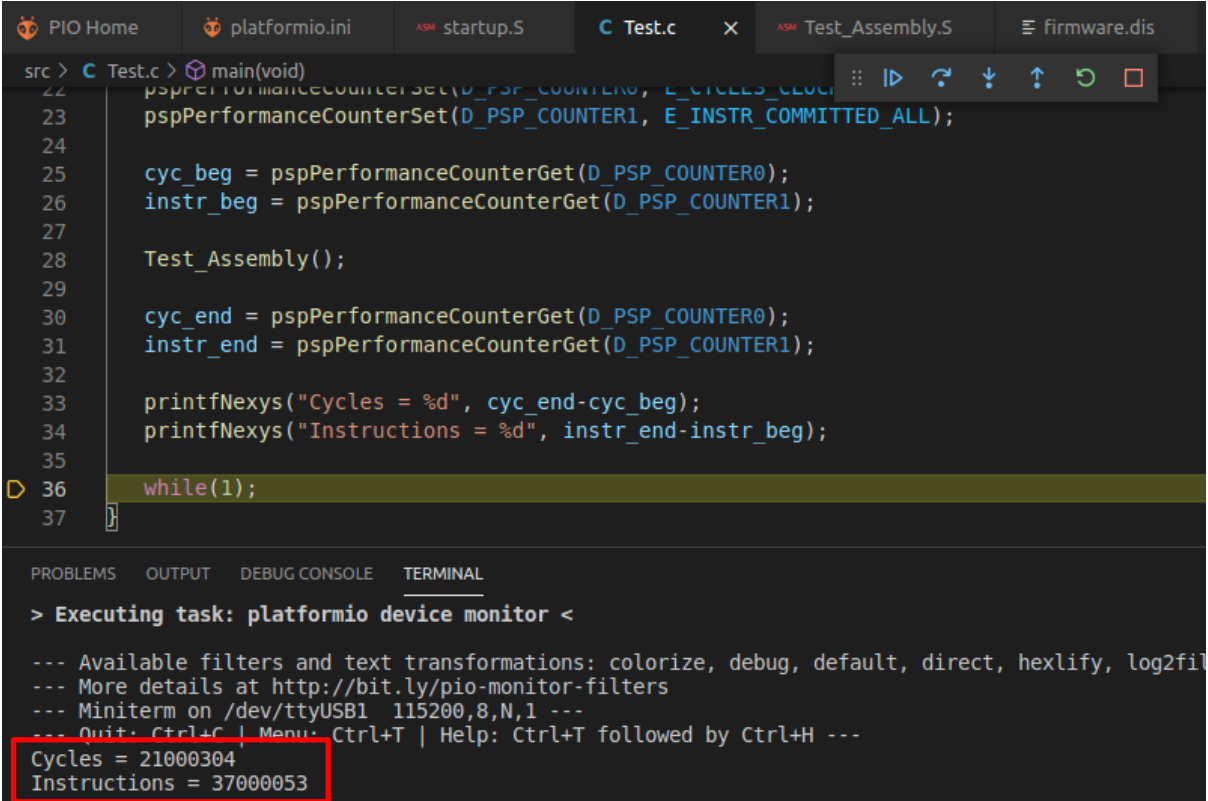


- a. 1º ciclo:
  - i. Via-0:
    1. A instrução `addi` (0x00b28493) está no andarcode. No final do ciclo, esta instrução progride para EX1.
  - ii. Via-1:
    1. A instrução `lw` (primeiro load, 0x0194a903) está no andarcode. No final do ciclo, esta instrução mantém-se na fase Decode, porque depende do resultado da instrução anterior.
- b. 2º ciclo:
  - i. Via-0:
    1. A instrução `lw` (primeiro load) está novamente no andarcode, mas agora passou da Via-1 para a Via-0. O operando de entrada para o endereço efetivo é obtido a partir da instrução aditiva. Recebe o operando de entrada para o cálculo do endereço efetivo através do reencaminhamento da instrução aditiva. No final do ciclo, esta instrução progride para DC1.
    2. A instrução `addi` está no andar EX1. Durante este ciclo, obtém o resultado da adição e envia-o para a primeiro load.
  - ii. Via-1:
    1. A instrução `lw` (segundo load, 0x01092a83) está no andarcode. No final deste ciclo não pode progredir devido a um conflito estrutural com a primeiro load.
- c. 3º ciclo:
  - i. Via-0:
    1. A instrução `lw` (segundo load) está novamente no andarcode, mas agora passou da Via-1 para a Via-0. Não poderá avançar, dado que o segundo load precisa do valor lido pelo primeiro load para calcular o endereço efetivo. Este valor é obtido no final de DC2 (ver Lab 13).
    2. A instrução `lw` (primeiro load) está no andar DC1.

3. A instrução `addi` está no andar EX2.
- ii. Via-1:
  1. A instrução `add` (0x015909b3) está no andarcod. Não poderá avançar devido a um conflito de dados com as duas instruções de leitura.
- d. 4º ciclo:
  - i. Via-0:
    1. A instrução `lw` (segundo load) está de novo no andarcod. Agora poderá progredir, pois o Pipe LSU está livre e o seu operando é obtido por forwarding.
    2. A instrução `lw` (primeiro load) está no andar DC2.
    3. A instrução `addi` está no andar EX3.
  - ii. Via-1:
    1. A instrução `add` está novamente no andarcod. Obtém os seus dois operandos de entrada a partir das duas leituras, pelo que será executado na ALU secundária (ver Lab 15).
- e. 5º ciclo:
  - i. Via-0:
    1. A instrução `or` (0x01d9ea33) A instrução está no andarcod. Poderá progredir, obtendo o primeiro operando através de forwarding.
    2. A instrução `lw` (segundo load) está no andar DC1. Recebe o operando de entrada para calcular o endereço efetivo a partir da primeira leitura, e poderá progredir.
    3. A instrução `lw` (primeiro load) está no andar DC3.
    4. A instrução `addi` está no andar Commit (EX4).
  - ii. Via-1:
    1. A instrução `and` está no andarcod. Não pode avançar porque tem um conflito de dados com a instrução `or` que se encontra no andarcod da Via-0 (ver Lab 15).
    2. A instrução `add` está no andar EX1.
- f. 6º ciclo:
  - i. Via-0:
    1. A instrução `and` está novamente no andarcod, mas passou de Via-1 para Via-0. Obtém os seus dois operandos de entrada através de forwarding e poderá progredir.
    2. A instrução `or` está no andar EX1. Reencaminha o resultado para a instrução `and`.
    3. A instrução `lw` (segundo load) está no andar DC2.
    4. A instrução `lw` (primeiro load) está no andar Commit.
    5. A instrução `addi` está no andar WriteBack (EX5). O registo `x9` é escrito em 0xF004000B.
  - ii. Via-1:
    1. A instrução `add` está no andar EX2.
- g. 7º ciclo:
  - i. Via-0:
    1. A instrução `and` (0x0149f933) está no andar EX1.

2. A instrução `or` está no andar EX2.
  3. A instrução `lw` (segundo load) está no andar DC3.
  4. A instrução `lw` (primeiro load) está no andar WB. O registo `x12` é escrito em `0xF0040000`.
- ii. Via-1:
1. A instrução `add` está no andar EX3.
- a. 8º ciclo:
- iii. Via-0:
1. A instrução `and` está no andar EX2.
  2. A instrução `or` está no andar EX3.
  3. A instrução `lw` (segundo load) está no andar Commit.
- iv. Via-1:
1. A instrução `add` está no andar Commit.
- b. 9º ciclo:
- v. Via-0:
1. A instrução `and` está no andar EX3.
  2. A instrução `or` está no andar Commit.
  3. A instrução `lw` (segundo load) está no andar WB. O registo `x15` é escrito em `0x00000005`.
- vi. Via-1:
1. A instrução `add` está no andar WB. O registo `x13` é escrito em `0xF0040005`.
- c. 10º ciclo:
- vii. Via-0:
1. A instrução `and` está no andar Commit.
  2. A instrução `or` está no andar WB. Register `x14` é escrito em `0xF0040005`.
- d. 11º ciclo:
- viii. Via-0:
1. A instrução `and` está no andar WB. Register `x12` é escrito em `0xF0040005`.

b)



The screenshot shows the PlatformIO IDE with the following code in `Test.c`:

```

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

```

The terminal output shows the execution results:

```

> 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

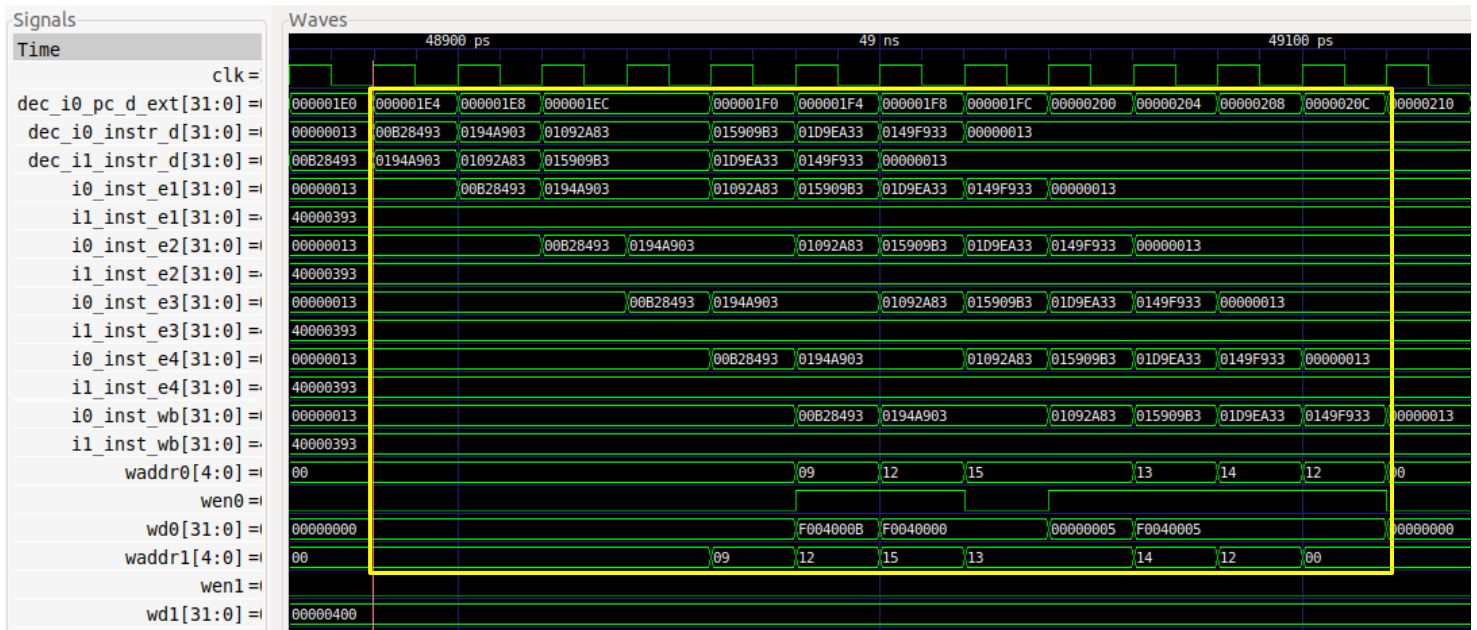
```

Note que temos de ter em conta que neste cálculo temos de remover todas as instruções extra que incluímos: primeiro `addi`, 29 `nops` e um `bne` final. Todas elas podem ser executadas em  $\frac{1}{2}$  ciclo (há um espaço livre na execução da última instrução do nosso programa).

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

O que é exatamente o que observamos na nossa simulação.

c)



É executada 1 Instrução por ciclo, exceto no conflito de dados entre os dois loads (0x0194a903 e 0x01092a83), que insere uma bolha no pipeline e 1 ciclo de relógio é perdido.

```

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, log
--- 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

```

Tal como anteriormente, existem 31 instruções extra que temos de remover para calcular o IPC. Todas elas podem ser executadas num 1 ciclo de relógio.

Assim: **IPC** =  $(37-31) / (38-31) = 6 / 7$

O que é exatamente o que observamos na nossa simulação.

7) (O seguinte exercício é baseado nos exercícios 7.31, 7.33 e 7.35 do Capítulo 7 de DDCARV.)

Repita o exercício 7 para o seguinte trecho de código.

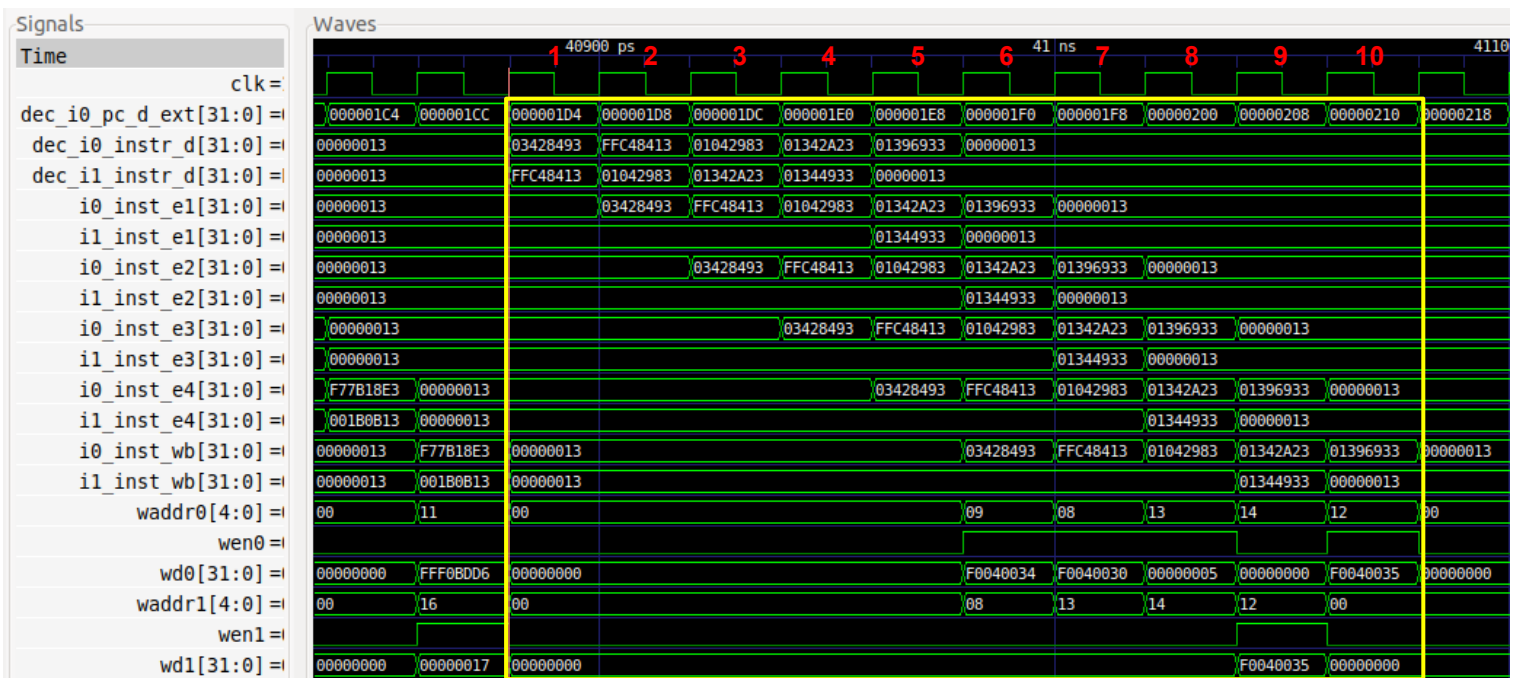
```
addi s1, t0, 52
addi s0, s1, -4
lw    s3, 16(s0)
sw    s3, 20(s0)
xor   s2, s0, s3
or    s2, s2, s3
```

Um projeto PlatformIO é fornecido em:

[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

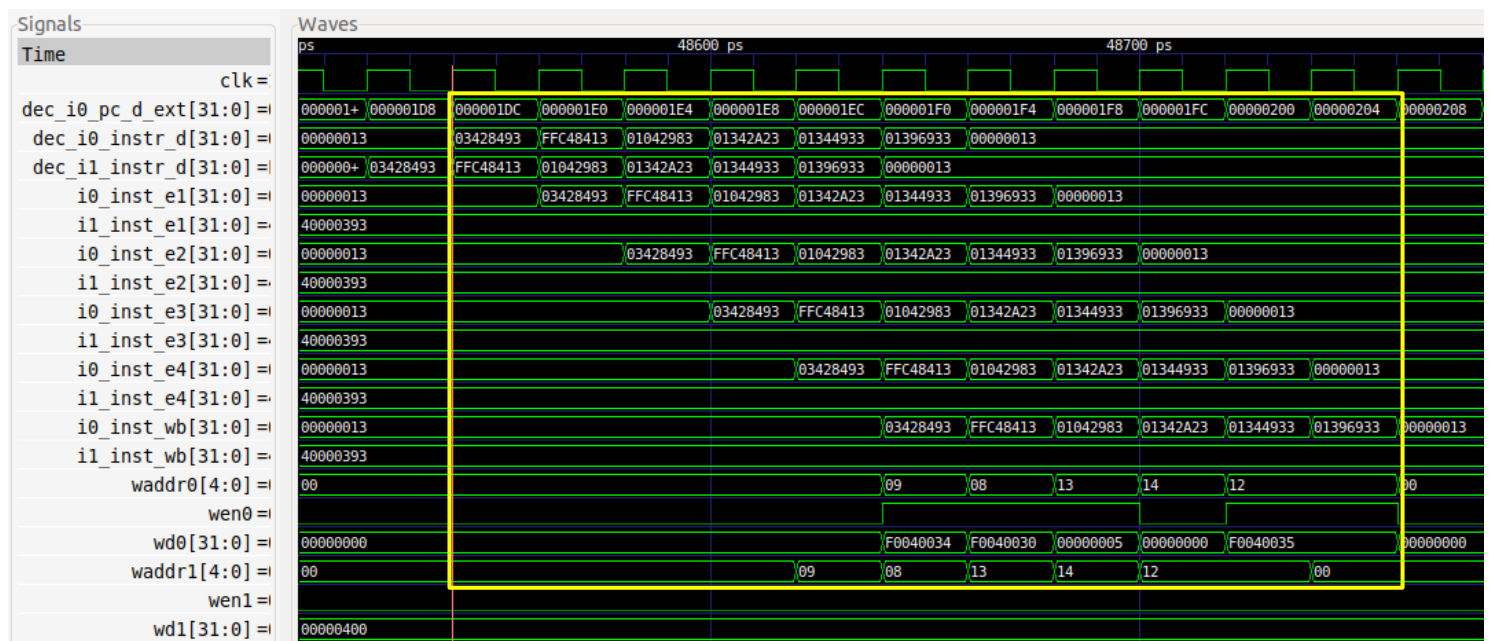


Existe um conflito entre todas as instruções consecutivas, exceto no caso do `sw` (0x01342a23) e do `xor` (0x01344933). Todas elas são conflitos de dados, exceto no caso do `lw` (0x01042983) e do `sw` (0x01342a23), que apresentam um conflito estrutural. De acordo com a simulação, apenas 1 instrução é executada por ciclo, exceto no caso do par `sw-xor`, para o qual a instrução `xor` é executada através da Via 1.

b)

Neste caso: **IPC = 6 / 5**

c)



Neste caso: **IPC = 6 / 6 = 1**