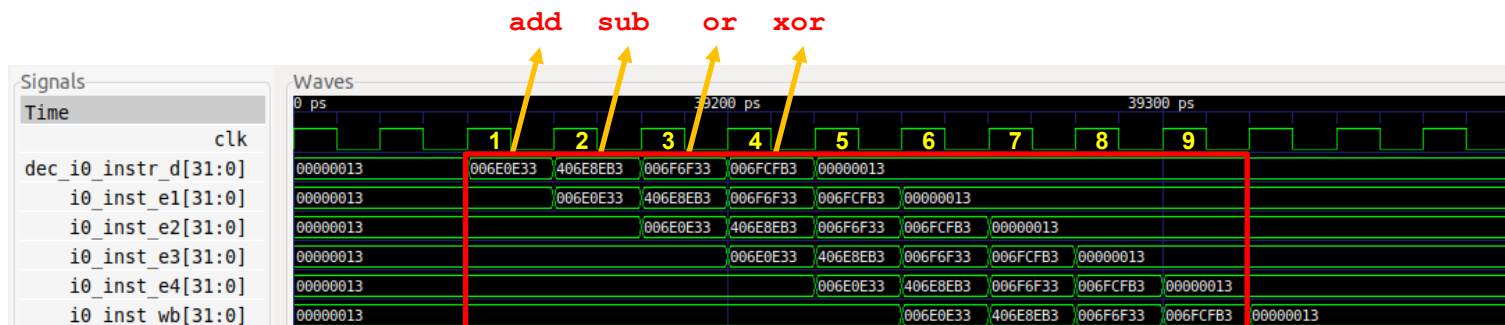


任务

任务：在图3的仿真中加入跟踪信号，并仿照图4的形式高亮显示流水线中从译码阶段到回写阶段的指令流。可以使用以下位置提供的.tcl文件：

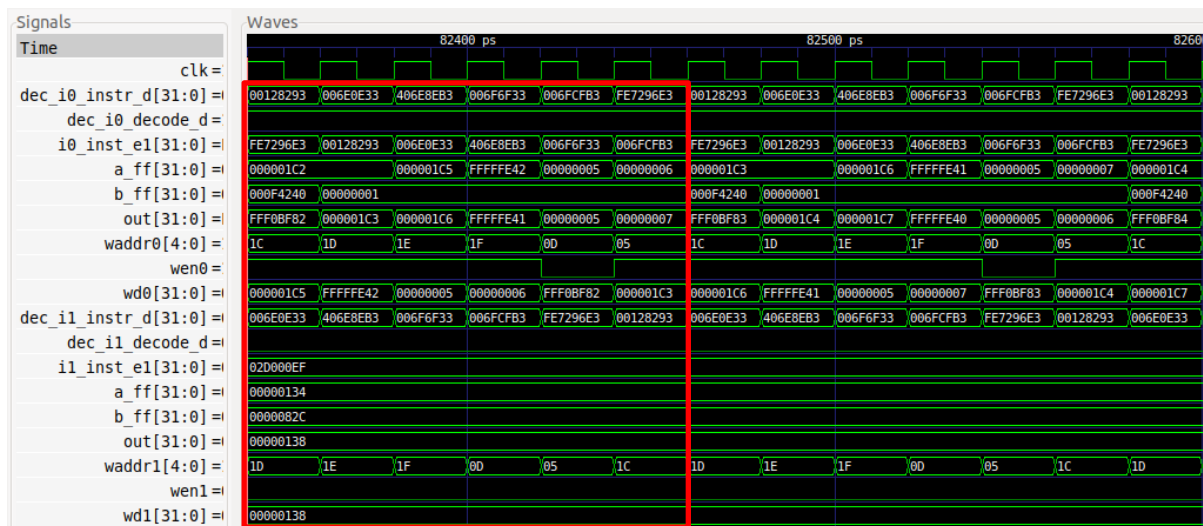
[RVfpgaPath]/RVfpga/Labs/Lab17/Four_AL_Instructions/test_task1.tcl。



任务：删除图2所示的循环主体中的所有nop指令。

重复图3中的仿真。该程序的预期IPC是多少？

在开发板上执行程序，验证获得的IPC是否符合您的预期。



循环中的6条指令需要通过6个周期执行。因此，IPC的预期值为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

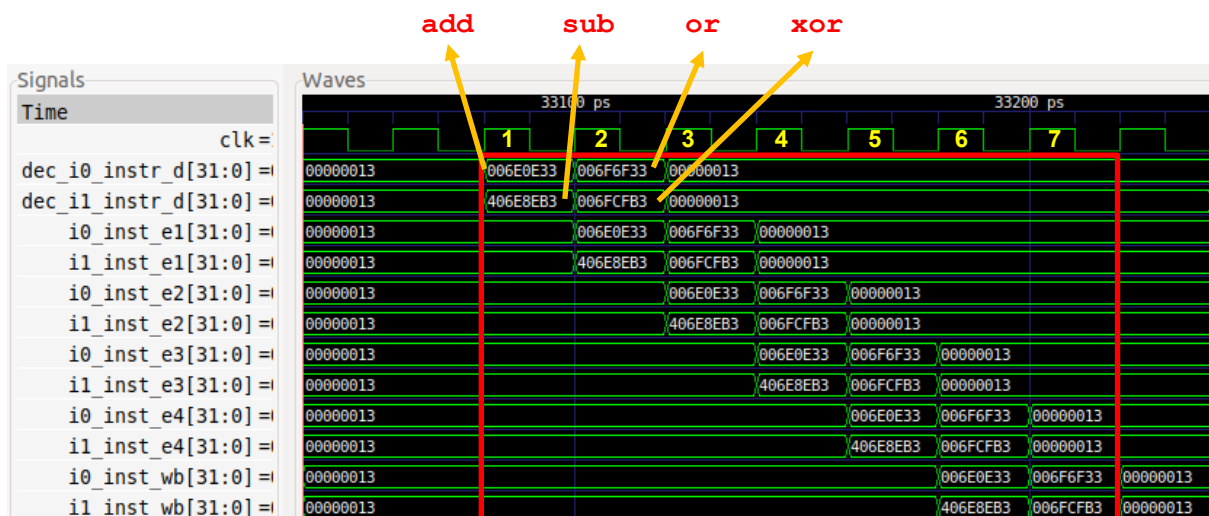
Cycles = 6000276
 Instructions = 6000048

正如预期:

$$IPC = 6000000 \text{ instructions} / 6000000 \text{ cycles} = 1$$

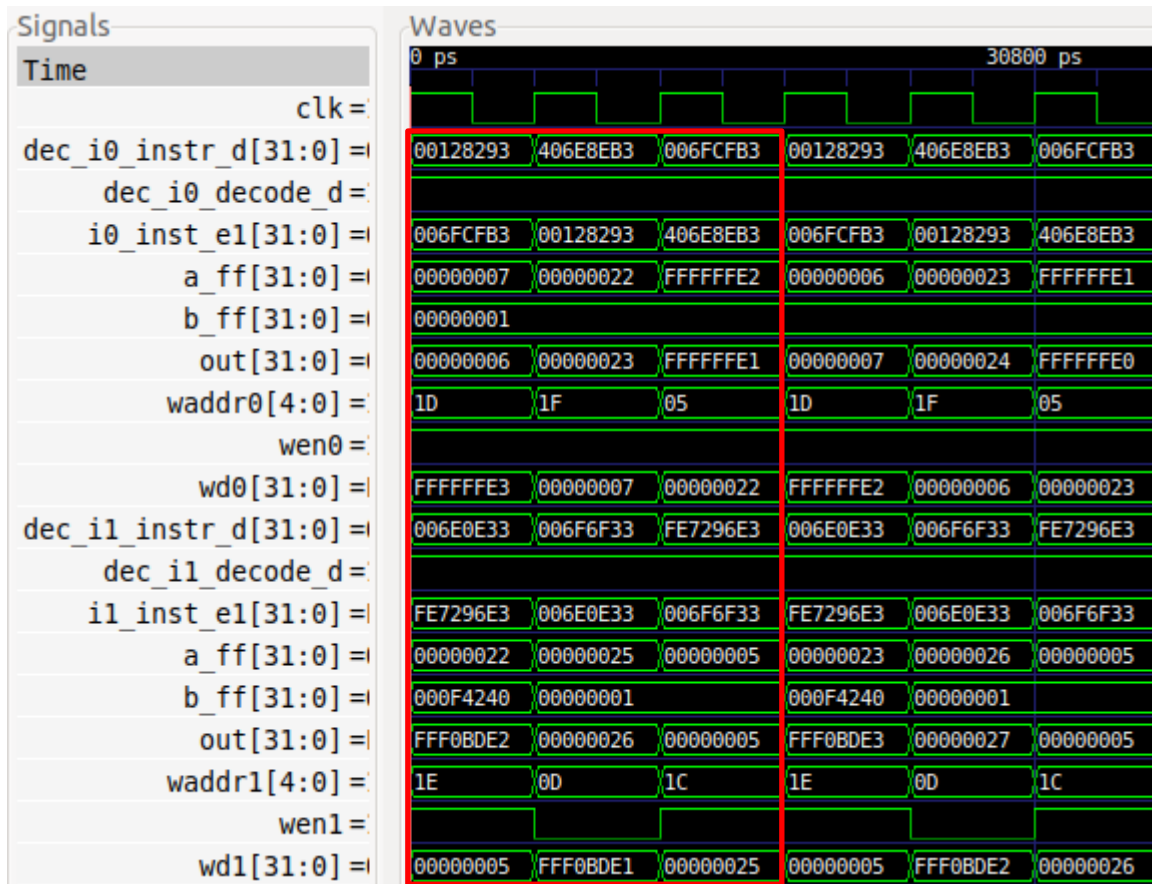
任务: 在图5的仿真中加入跟踪信号，并仿照图6的形式高亮显示流水线中从译码阶段到回写阶段的指令流。可以使用以下位置提供的.tcl文件:

[RVfpgaPath]/RVfpga/Labs/Lab17/Four_AL_Instructions/test_task2.tcl。



任务: 删除图2所示的循环主体中的所有nop指令。

重复图5中的仿真。该程序的预期IPC是多少？
在开发板上执行程序，验证获得的IPC是否符合您的预期。



循环中的6条指令需要通过3个周期执行。因此，IPC的预期值为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

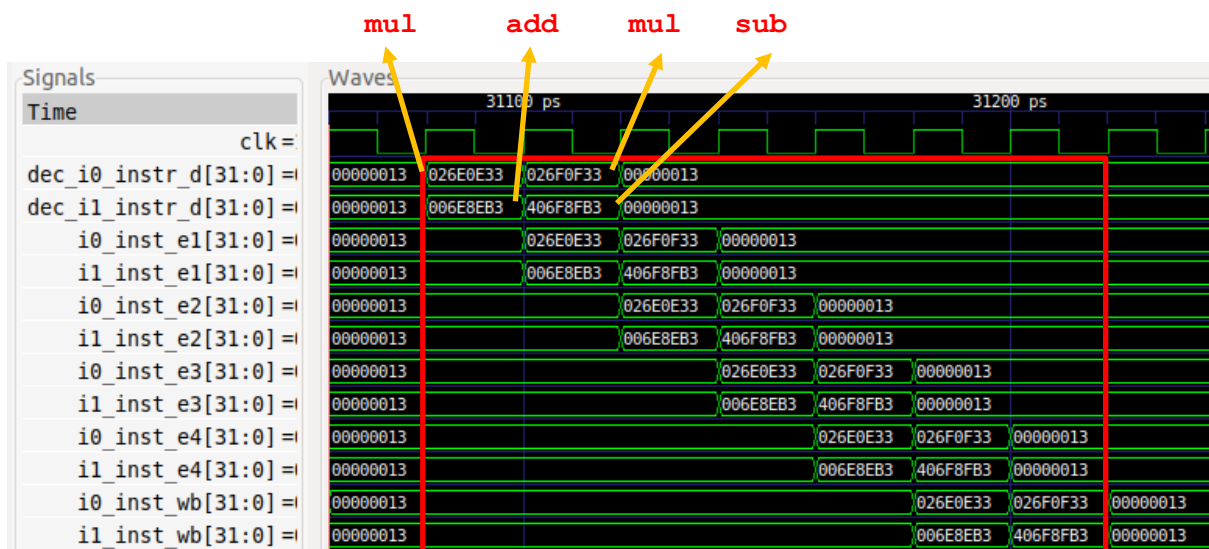
```

正如预期:

$$IPC = 6000000 \text{ instructions} / 3000000 \text{ cycles} = 2$$

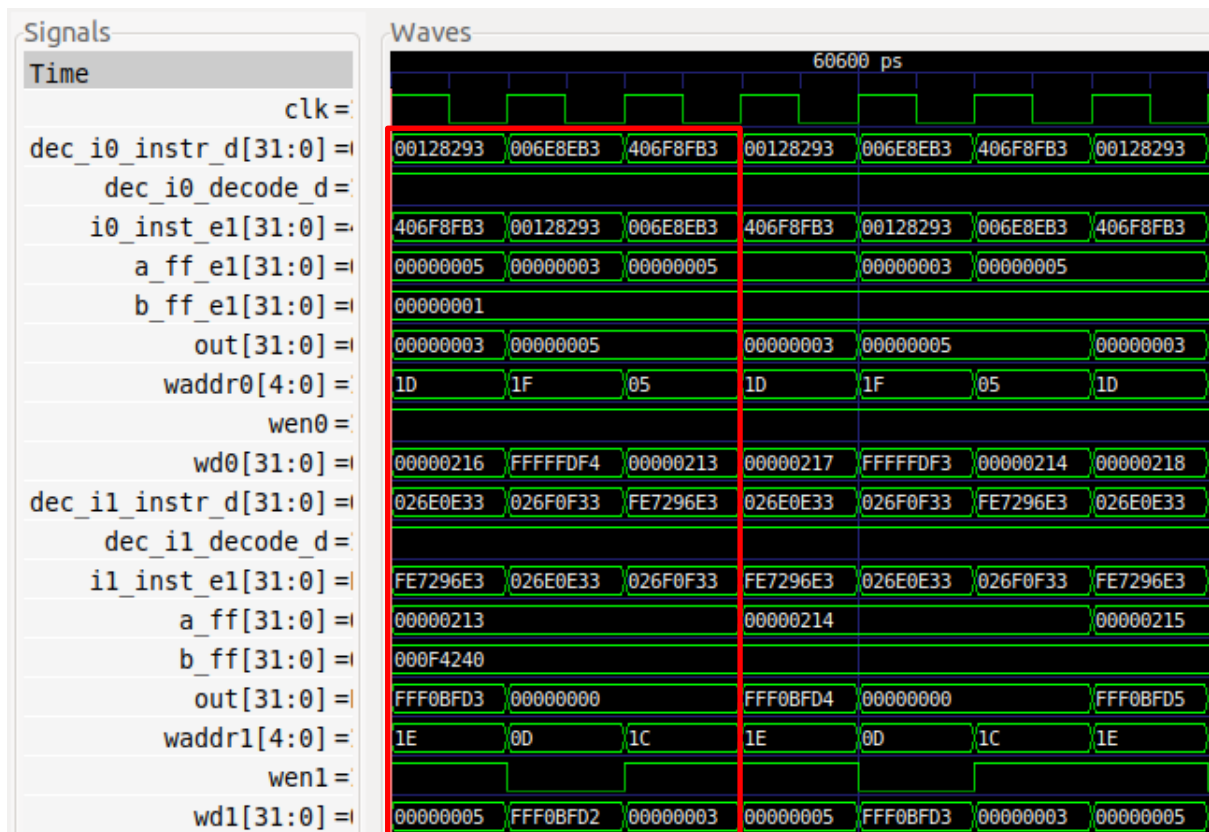
任务: 在图8的仿真中加入跟踪信号，并高亮显示流水线中从译码阶段到回写阶段的指令流。可以使用以下位置提供的.tcl文件:

[RVfpgaPath]/RVfpga/Labs/Lab17/TwoAL_TwoMUL_Instructions/test_taskMuls.tcl。

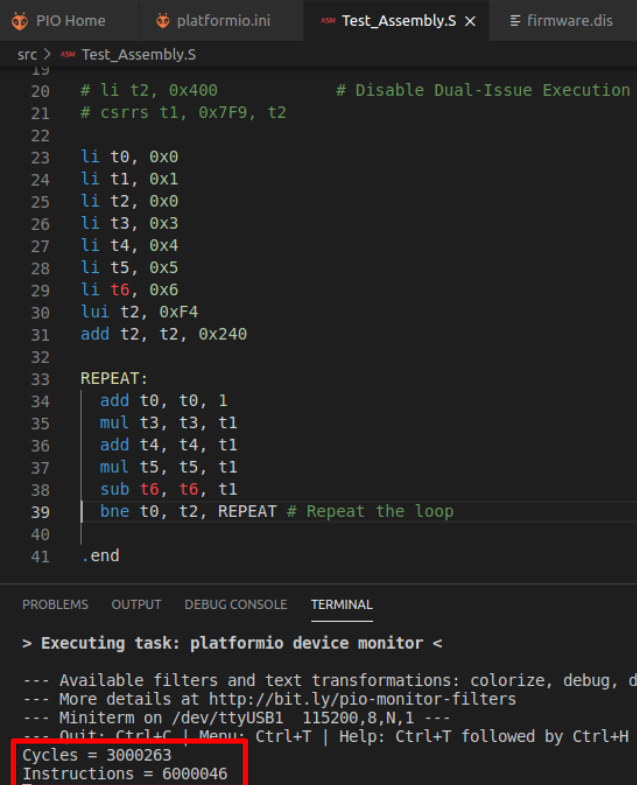


任务：删除图7所示的循环主体中的所有nop指令。
 重复图8中的仿真。该程序的预期IPC是多少？
 在开发板上执行程序，验证获得的IPC是否符合您的预期。
 使用单发射配置重复实验，并对比实验结果。

双发射：



循环中的6条指令需要通过3个周期执行。因此，IPC的预期值为2。



```

src > ASM 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

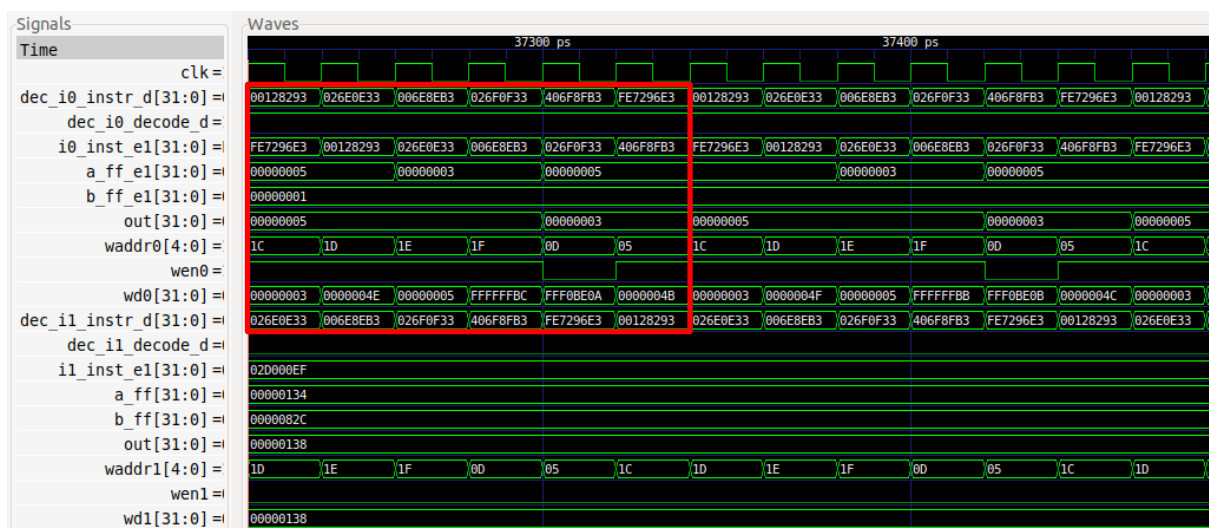
Cycles = 3000263
Instructions = 6000046

```

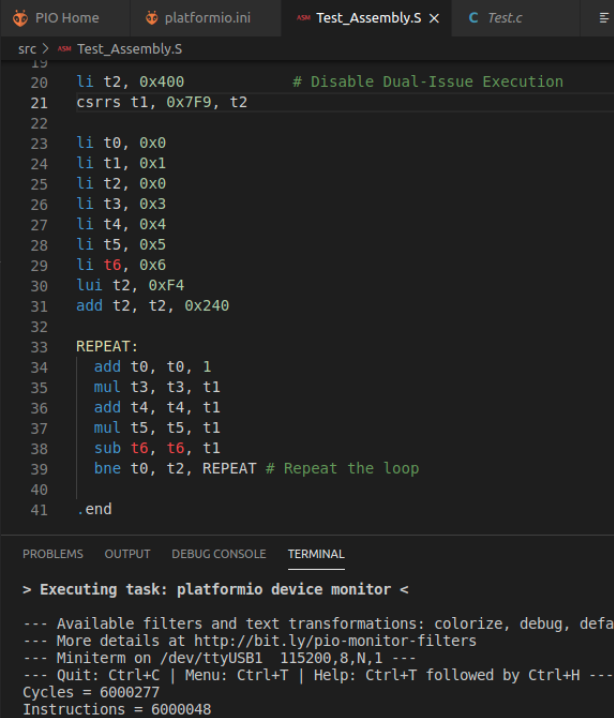
正如预期:

$$IPC = 6000000 \text{ instructions} / 3000000 \text{ cycles} = 2$$

单发射:



循环中的6条指令需要通过6个周期执行。因此，IPC的预期值为1。



```

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

```

正如预期:

$$\text{IPC} = 6000000 \text{ instructions} / 6000000 \text{ cycles} = 1$$

练习

- 1) 使用能够展示双发射执行相关新情境的指令组合，创建与图2和图7所示程序类似的程序。

不提供解答。

- 2) 分析（双发射）SweRV EH1处理器与S.Harris和D.Harris所著教材《数字设计和计算机体系结构》（RISC-V版本，简称[DDCARV]）第7.7.4节中所示的超标量处理器（为方便起见，该处理器已在图1中给出）之间的差异。

不提供解答。

- 3) 分析DDCARV第7.7.4节的图7.70中的程序，该程序位于文件夹 `[RVfpgaPath]/RVfpga/Labs/Lab17/DDCARV_SuperscalarExample` 下的PlatformIO项目中。使用仿真器和开发板两种方式，在SweRV EH1上运行该程序（对于后一种方式，请删除nop指令）。解释得到的结果。如有必要，可调整程序以获得最理想的IPC。

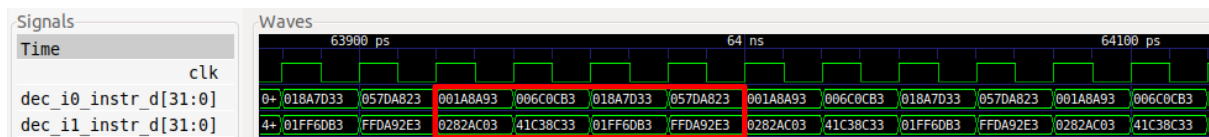
接下来，根据本实验及SweRVref.docx（第2部分）中所述，禁止双发射执行。将仿真结果和开发板上的执行结果与使能双发射功能时的相应结果进行比较。

000001c0 <REPEAT>:

```

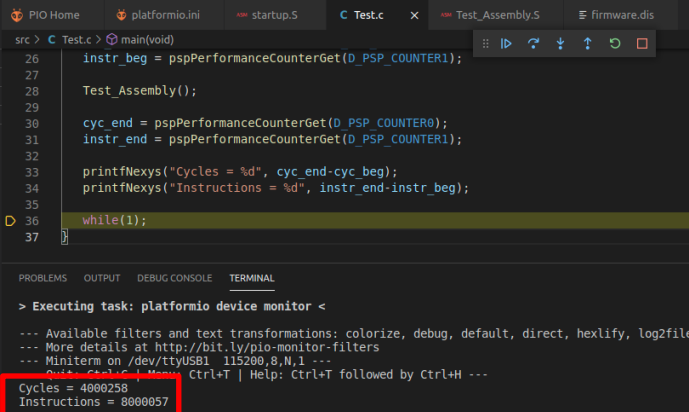
1c0: 001a8a93      addi    s5,s5,1
1c4: 0282ac03      lw      s8,40(t0)
1c8: 006c0cb3      add     s9,s8,t1
1cc: 41c38c33      sub     s8,t2,t3
1d0: 018a7d33      and     s10,s4,s8
1d4: 01ff6db3     or      s11,t5,t6
1d8: 057da823     sw      s7,80(s11)
1dc: ffd92e3      bne     s5,t4,1c0 <REPEAT>
  
```

仿真 – 双发射:



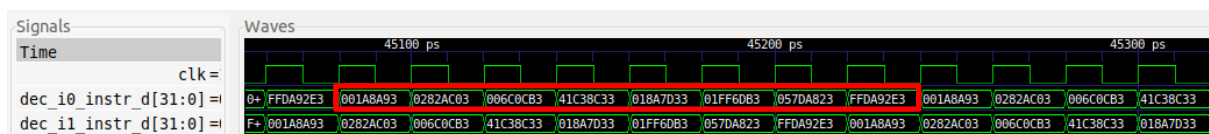
循环不存在暂停，并且每个周期始终执行2条指令。系统会执行一些旁路，在某些情况下还会使用辅助ALU，如之前的实验中所述。可以进一步分析这些情况。

开发板上的执行结果 – 双发射:



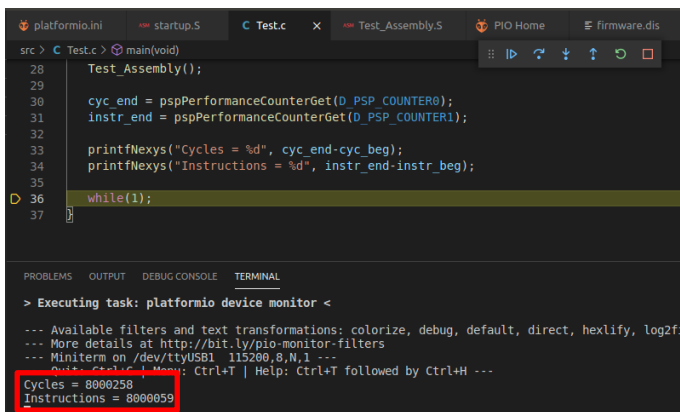
IPC等于2，无需调整代码顺序。

仿真 – 单发射:



循环不存在暂停，并且每个周期始终执行1条指令。

开发板上的执行结果 – 单发射:



```

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

```

```

> 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

```

IPC等于1，这是单发射SweRV EH1中所能达到的最优值。

- 4) 修改练习3中的程序，将指令add s9, s8, t1替换为add t2, s8, t1。解释得到的结果。如有必要，可调整程序以获得最理想的IPC。

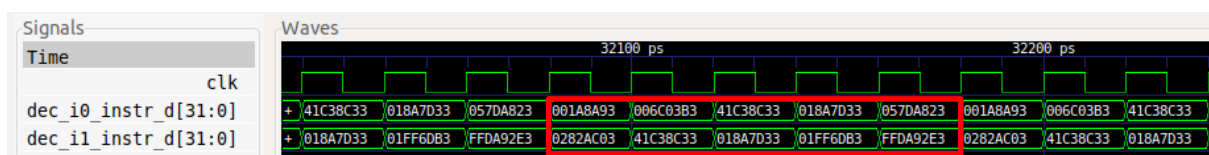
然后，根据本实验及SweRVref.docx

（第2节）中所述，禁止双发射执行。将仿真结果和开发板上的执行结果与使能双发射功能时的相应结果进行比较。

000001c0 <REPEAT>:

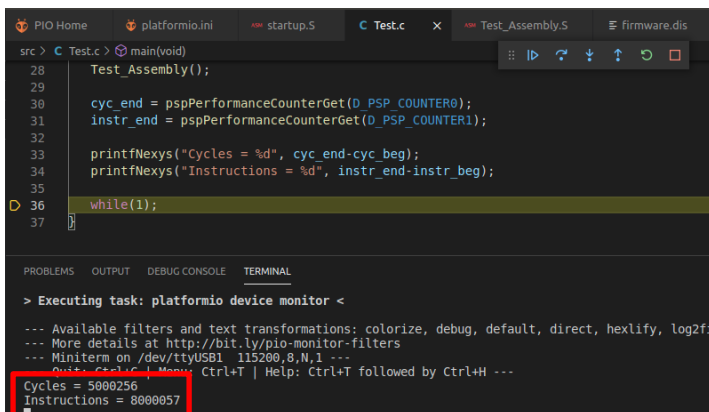
1c0:	001a8a93	addi	s5,s5,1
1c4:	0282ac03	lw	s8,40(t0)
1c8:	006c03b3	add	t2,s8,t1
1cc:	41c38c33	sub	s8,t2,t3
1d0:	018a7d33	and	s10,s4,s8
1d4:	01ff6db3	or	s11,t5,t6
1d8:	057da823	sw	s7,80(s11)
1dc:	ffda92e3	bne	s5,t4,1c0 <REPEAT>

仿真 – 双发射:



由于AL指令之间的相关性，循环中存在2个暂停。

开发板上的执行结果 – 双发射:



```

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

```

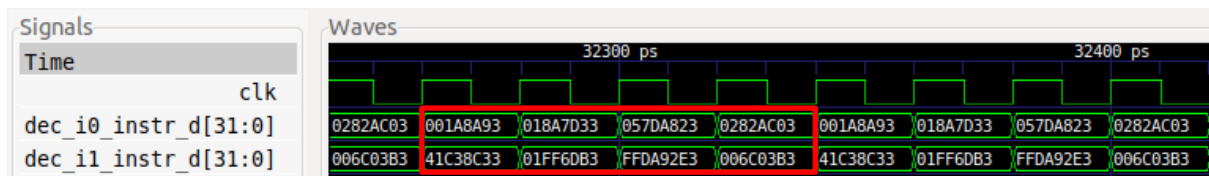
由于存在暂停，IPC小于2。

调整代码顺序：

000001c0 <REPEAT>:

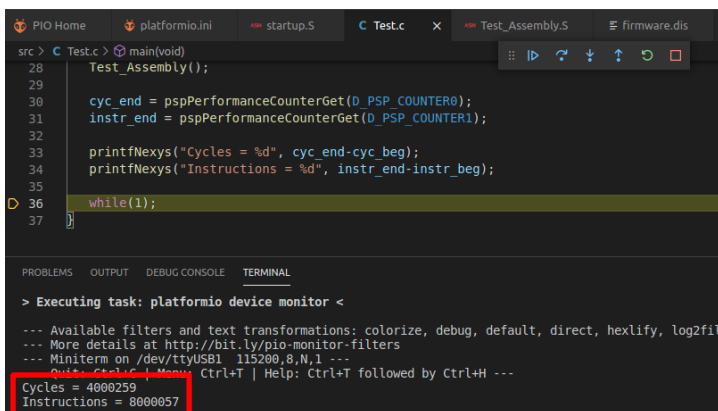
1c0:	0282ac03	lw	s8,40(t0)
1c4:	006c03b3	add	t2,s8,t1
1c8:	001a8a93	addi	s5,s5,1
1cc:	41c38c33	sub	s8,t2,t3
1d0:	018a7d33	and	s10,s4,s8
1d4:	01ff6db3	or	s11,t5,t6
1d8:	057da823	sw	s7,80(s11)
1dc:	ffda92e3	bne	s5,t4,1c0 <REPEAT>

仿真 – 双发射：



循环不存在暂停。

开发板上的执行结果 – 双发射：



```

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

```

IPC等于2。

回到原程序：

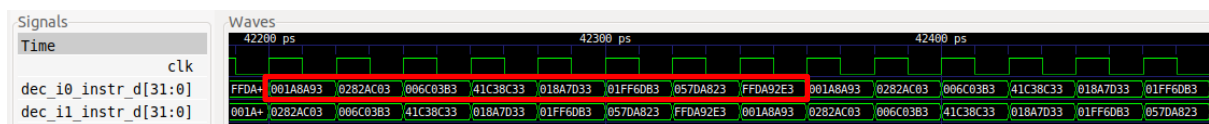
000001c0 <REPEAT>:

```

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>

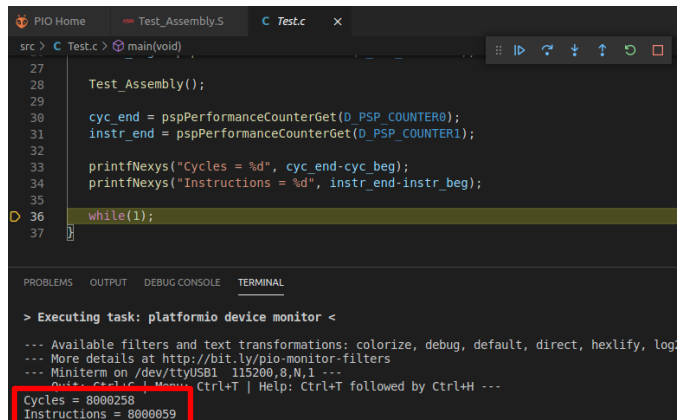
```

仿真 – 单发射：



循环不存在暂停。

开发板上的执行结果 – 单发射：



```

src > C Test.c > main(void)
27 Test_Assembly();
28
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+Enter to send | Ctrl+T | Help: Ctrl+T followed by Ctrl+H ---
Cycles = 8000258
Instructions = 8000059

```

IPC等于1。

5) （以下练习基于《计算机组织结构和设计》（RISC-V版本，Patterson & Hennessy ([HePa])）中的练习4.31。）

在本练习中，我们将比较单发射处理器和双发射处理器的性能，并考虑可优化双发射执行的程序修改。本练习中的问题涉及以下循环（用C语言编写）：

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

仅经过少量优化或未经优化的编译器可能会生成以下RISC-V汇编代码：

```

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
```

该代码使用以下寄存器：

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

该代码位于 `[RVfpgaPath]/RVfpga/Labs/Lab17/PaHe_SuperscalarExample` 路径下，与本书练习中提供的代码相比，存在一些小的修改，但不会影响程序的行为：

- 将寄存器x13初始化为8，以使程序执行4次迭代。
- 删除了jal指令。
- 将ld和sd指令替换为lw和sw指令。这表示访问从4个字节宽变为8个字节宽。

假设有一个具备以下属性的双发射静态调度处理器：

1. 一条指令必须是访存操作；另一条必须是算术/逻辑指令或分支指令。
2. 处理器的各阶段之间支持所有可能的转发路径。
3. 处理器可实现完美分支预测。
4. 如果两条指令相关，则可能无法同时发出。
5. 如果某一阶段需要暂停指令，则必须同时暂停两条指令。

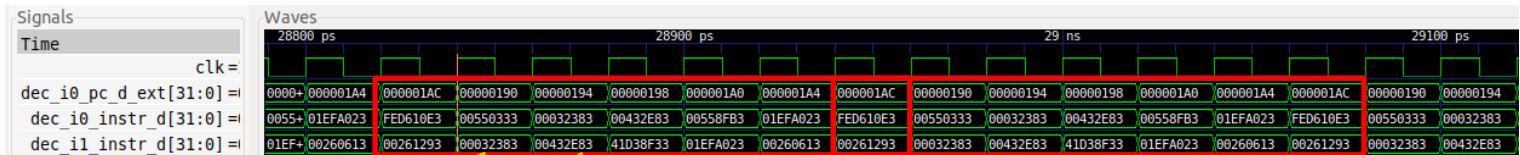
- a) 比较该示例处理器与SweRV EH1处理器的属性。
- b) 绘制流水线图和仿真图，显示上述RISC-V代码的任意一次循环迭代（第一次迭代除外）在双发射SweRV EH1处理器上的执行情况。假设在进行4000次迭代后退出循环（上述代码即为如此）。
- c) 从单发射SweRV EH1处理器变为双发射SweRV EH1处理器时，加速比为多少？解释得到的结果。在开发板上测试程序并允许/禁止双发射执行。
- d) 重新排列/重写上述RISC-V代码，以在双发射SweRV EH1处理器上实现更高的性能。（但请勿展开循环。）
- e) 接下来展开RISC-V代码，使展开后循环的每次迭代等同于原循环的两次迭代。然后，重新排列/重写展开的代码，以在双发射SweRV EH1处理器上实现更高的性能。

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

```



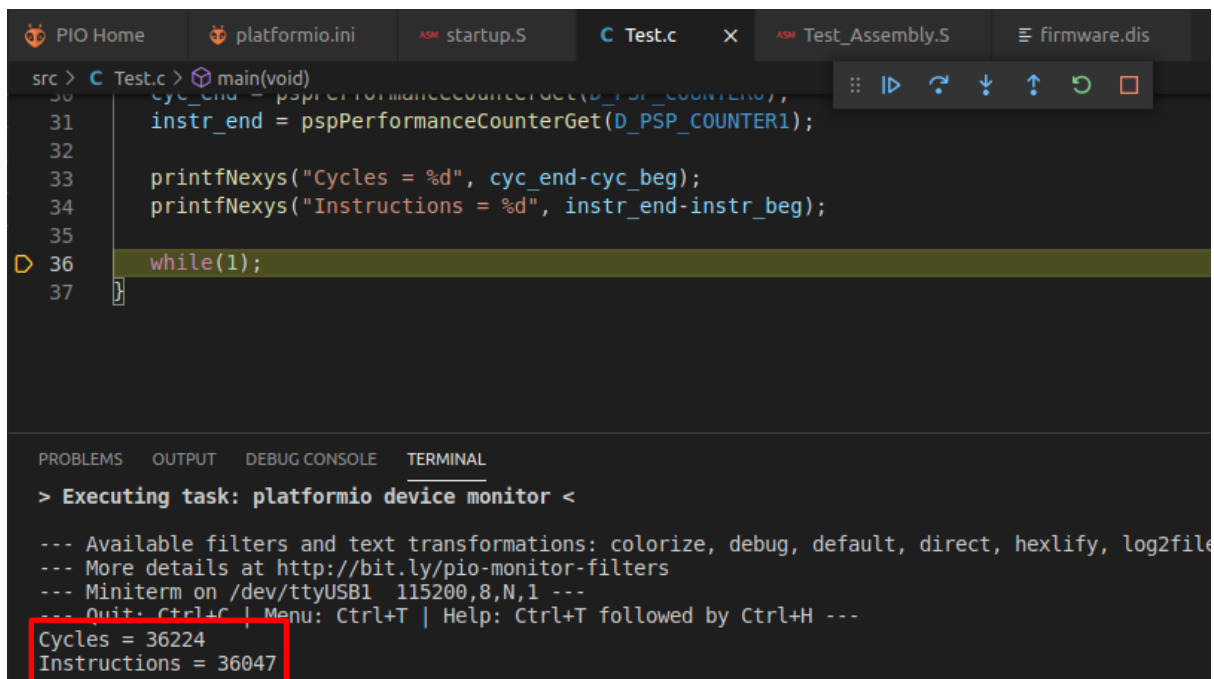
由于add t1,a0,t0存在写后读（RAW）数据冒险，lw t2,0(t1)被延时。实验15的最后一项任务（第2.A部分）中已分析过此情况。

由于两条lw指令间存在结构冒险而延时。

由于add t6,a1,t0存在写后读（RAW）数据冒险，sw t5,0(t6)被延时。

c)

单发射：



```

src > C Test.c > main(void)
30  cyc_beg = pspPerformanceCounterGet(D_PSP_COUNTER0);
31  instr_beg = 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

```

```

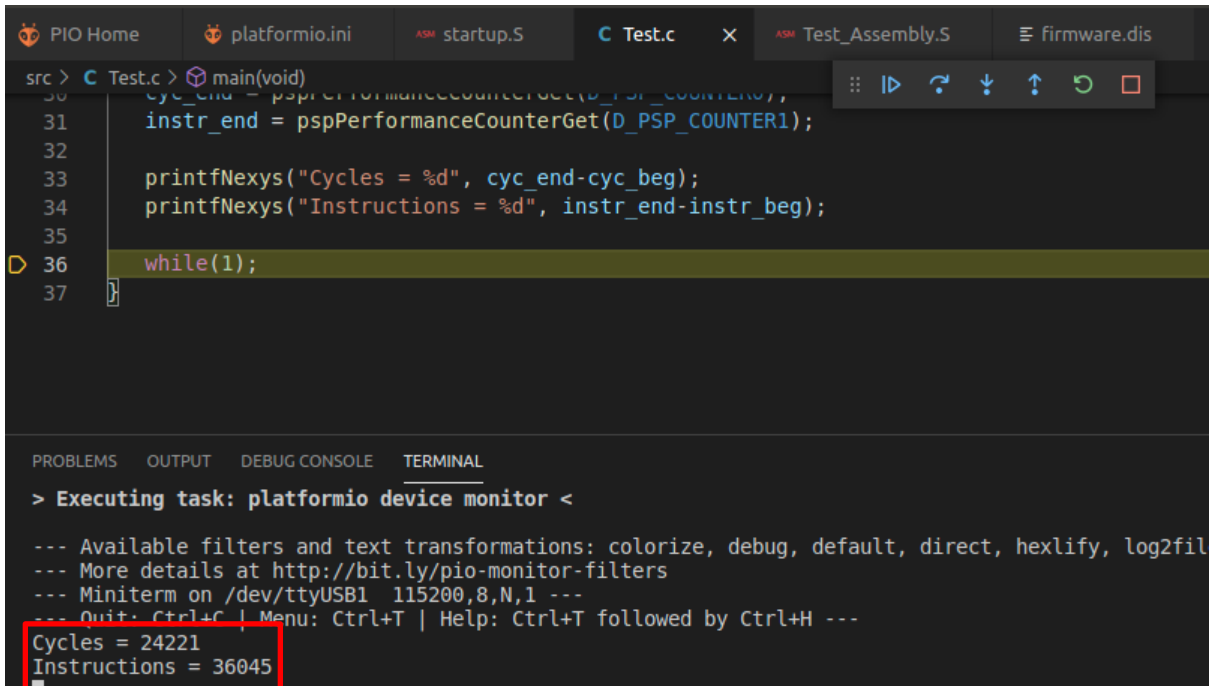
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL
> Executing task: platformio device monitor <

--- Available filters and text transformations: colorize, debug, default, direct, hexlify, 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

```

双发射：



```
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
```

加速比 = 单发射时间/双发射时间 = $36224/24221 \approx 1.5$ (理想情况下加速比应为2, 但根据(b)中的分析, 由于存在暂停, 实际加速比仅为1.5)

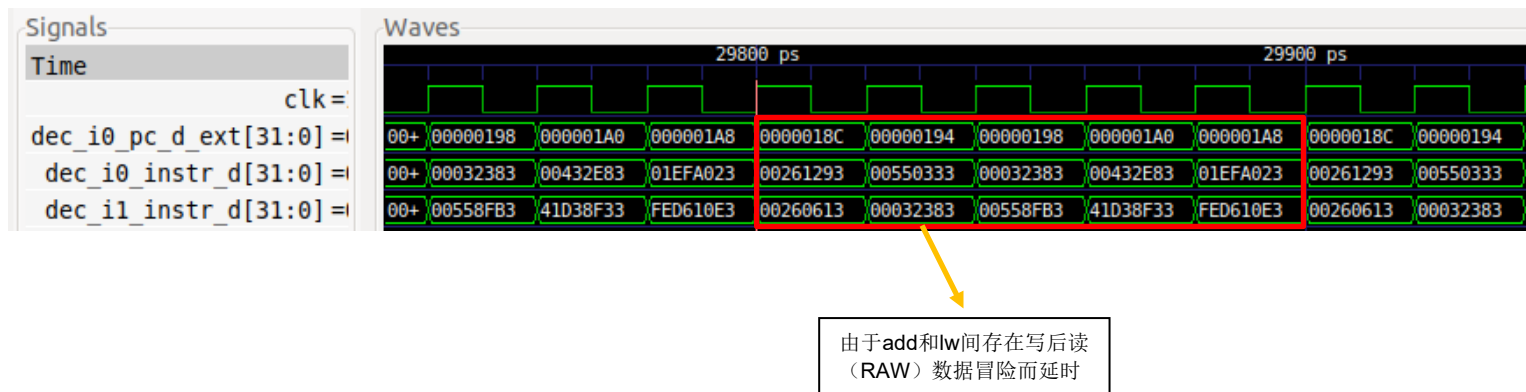
d)

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)

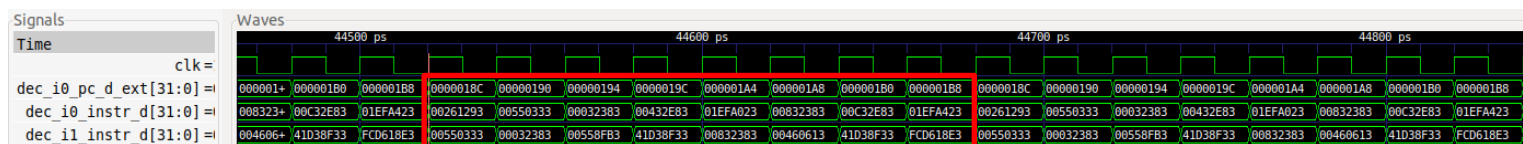
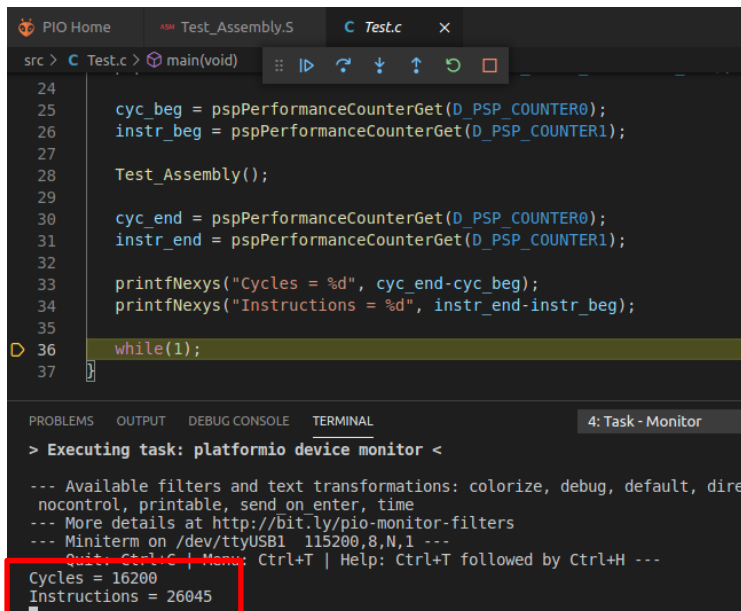
TOP:

```
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
```

得益于代码的展开和重写，性能进一步提升。

6) (以下练习基于DDCARV第7章的练习7.30、7.32和7.34。)

假设SweRV EH1处理器正在运行以下代码片段。回想一下，SweRV EH1带有冒险单元。可以假设一个能够在一个周期内返回结果的存储器系统（为此，我们使用DCCM，将代码片段插入一个循环中，并避免使用第一次迭代，以免发生IS\$未命中）。

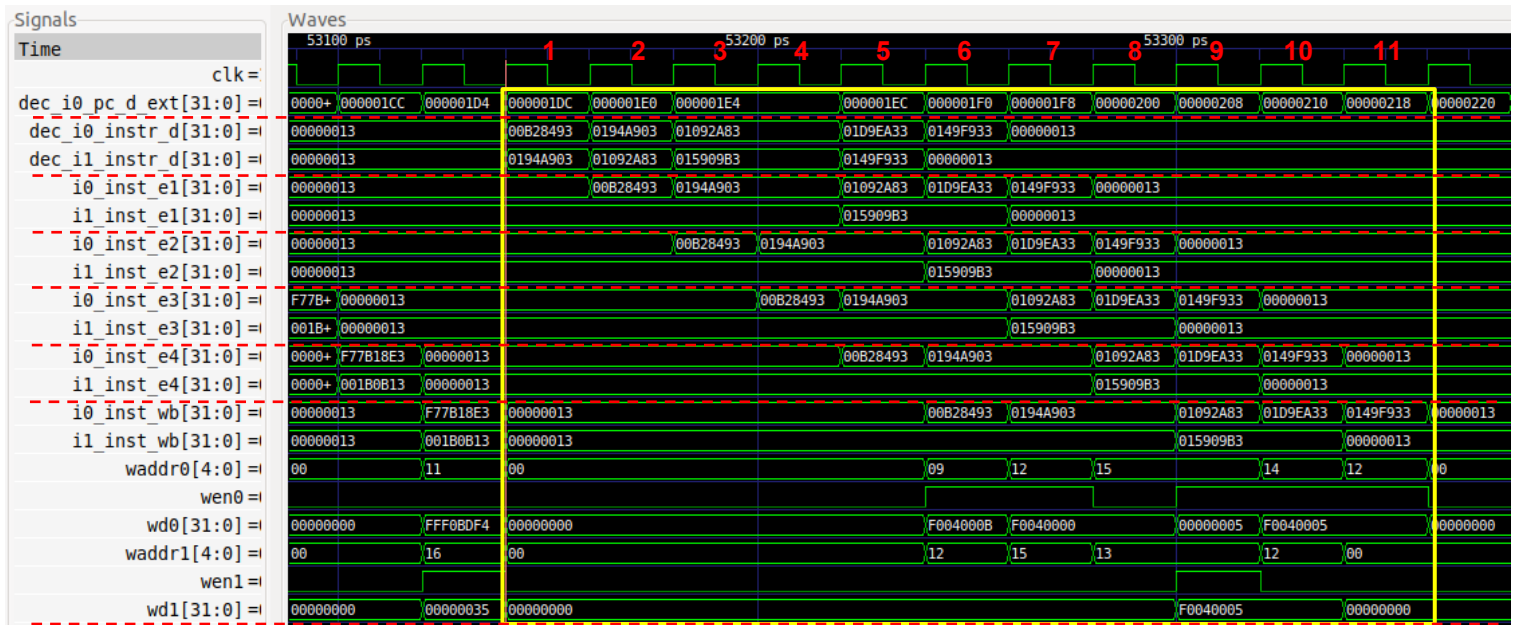
```
addi s1, t0, 11    # t0 contains the base address of the DCCM
lw   s2, 25(s1)
lw   s5, 16(s2)
add  s3, s2, s5
or   s4, s3, t4
and  s2, s3, s4
```

- a) 使用Verilator和GTKWave仿真该程序。分析结果，并针对每个周期指明以下内容：
 - * 对哪些指令进行译码？将哪些指令发送至执行阶段？提交哪些指令？
 - * 对哪些寄存器进行写操作和读操作？
 - * 发生哪些转发和暂停情况？
- b) 对于该程序，处理器的CPI是多少？先给出理论解答，然后在开发板上执行程序，确认您的答案。
- c) 在单发射处理器上执行相同的分析，将结果与使用双发射处理器时的结果进行比较。

PlatformIO项目的路径如下：[RVfpgaPath]/RVfpga/Labs/Lab17/DDCARV_Exercises-30-32-34。所分析的程序将插入一个循环中，以使仿真更易理解（可使用除第一次迭代外的任何迭代进行分析）并且可以使用性能计数器进行测定。

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

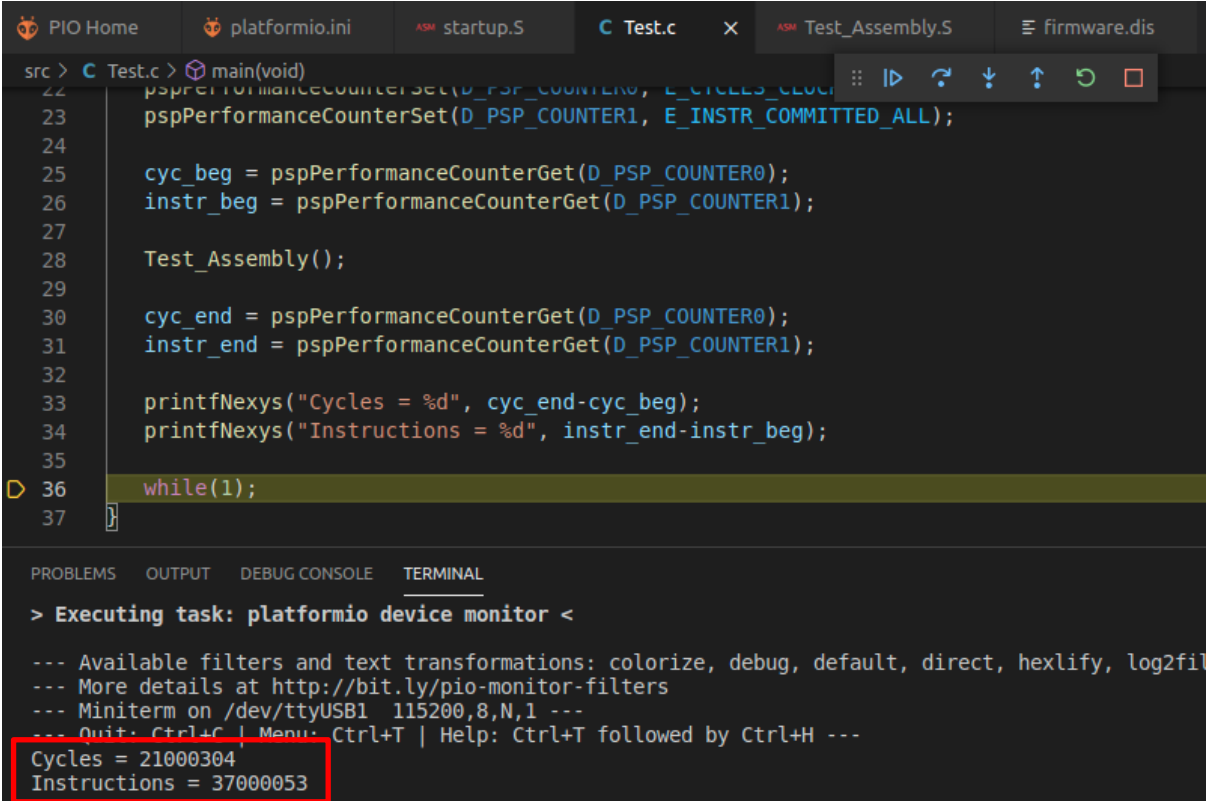


- a. 第1个周期:
 - i. 通路0:
 1. 指令addi (0x00b28493) 处于译码阶段。此周期结束时, 指令进入EX1阶段。
 - ii. 通路1:
 1. 指令lw (第一次装载, 0x0194a903) 处于译码阶段。此周期结束时, 指令仍留在译码阶段, 因为该指令的阶段取决于前一条addi指令的结果。
- b. 第2个周期:
 - i. 通路0:
 1. 指令lw (第一次装载) 再次处于译码阶段, 但从通路1移至通路0。有效地址的输入操作数取自addi指令。指令lw通过转发的方式从addi指令接收用于计算有效地址的输入操作数。此周期结束时, 指令lw进入DC1阶段。
 2. 指令addi处于EX1阶段。在此周期中, 该指令会获取相加后的结果并将其转发至第一次装载。
 - ii. 通路1:
 1. 指令lw (第二次装载, 0x01092a83) 处于译码阶段。此周期结束时, 由于与第一次装载之间存在结构冒险, 该指令无法进入下一阶段。
- c. 第3个周期:
 - i. 通路0:
 1. 指令lw (第二次装载) 再次处于译码阶段, 但从通路1移至通路0。由于第二次装载需要使用第一次装载读取的值来计算有效地址, 该指令无法进入下一阶段。在DC2阶段结束时可获得该值 (参见实验13)。

2. 指令lw（第一次装载）处于DC1阶段。
3. 指令addi处于EX2阶段。
- ii. 通路1:
 1. 指令add（0x015909b3）处于译码阶段。该指令无法进入下一阶段，因为两条装载指令存在数据冒险。
- d. 第4个周期:
 - i. 通路0:
 1. 指令lw（第二次装载）再次处于译码阶段。此时，由于LSU管道空闲且已通过转发获取其操作数，该指令可以进入下一阶段。
 2. 指令lw（第一次装载）处于DC2阶段。
 3. 指令addi处于EX3阶段。
 - ii. 通路1:
 1. 指令add再次处于译码阶段。该指令从两次装载中获得两个输入操作数，因此将在辅助ALU中执行（参见实验15）。
- e. 第5个周期:
 - i. 通路0:
 1. 指令or（0x01d9ea33）处于译码阶段。通过转发获得第一个操作数后，该指令可以进入下一阶段。
 2. 指令lw（第二次装载）处于DC1阶段。该指令会从第一次装载中接收用于计算有效地址的输入操作数，随后即可进入下一阶段。
 3. 指令lw（第一次装载）处于DC3阶段。
 4. 指令addi处于提交阶段（EX4）。
 - ii. 通路1:
 1. 指令and处于译码阶段。该指令无法进入下一阶段，因为通路0中的or指令在译码阶段存在数据冒险（参见实验15）。
 2. 指令add处于EX1阶段。
- f. 第6个周期:
 - i. 通路0:
 1. 指令and再次处于译码阶段，但从通路1移至通路0。该指令会通过转发的形式获得两个输入操作数，随后即可进入下一阶段。
 2. 指令or处于EX1阶段。该指令会将结果转发给and指令。
 3. 指令lw（第二次装载）处于DC2阶段。
 4. 指令lw（第一次装载）处于提交阶段。
 5. 指令addi处于回写阶段（EX5）。向寄存器x9写入0xF004000B。
 - ii. 通路1:
 1. 指令add处于EX2阶段。
- g. 第7个周期:
 - i. 通路0:
 1. 指令and（0x0149f933）处于EX1阶段。

2. 指令or处于EX2阶段。
 3. 指令lw（第二次装载）处于DC3阶段。
 4. 指令lw（第一次装载）处于回写阶段。向寄存器x12写入0xF0040000。
- ii. 通路1:
1. 指令add处于EX3阶段。
- a. 第8个周期:
- iii. 通路0:
1. 指令and处于EX2阶段。
 2. 指令or处于EX3阶段。
 3. 指令lw（第二次装载）处于提交阶段。
- iv. 通路1:
1. 指令add处于提交阶段。
- b. 第9个周期:
- v. 通路0:
1. 指令and处于EX3阶段。
 2. 指令or处于提交阶段。
 3. 指令lw（第二次装载）处于回写阶段。向寄存器x15写入0x00000005。
- vi. 通路1:
1. 指令add处于回写阶段。向寄存器x13写入0xF0040005。
- c. 第10个周期:
- vii. 通路0:
1. 指令and处于提交阶段。
 2. 指令or处于回写阶段。向寄存器x14写入0xF0040005。
- d. 第11个周期:
- viii. 通路0:
1. 指令and处于回写阶段。向寄存器x12写入0xF0040005。

b)



```

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

```

```

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 = 21000304
Instructions = 37000053

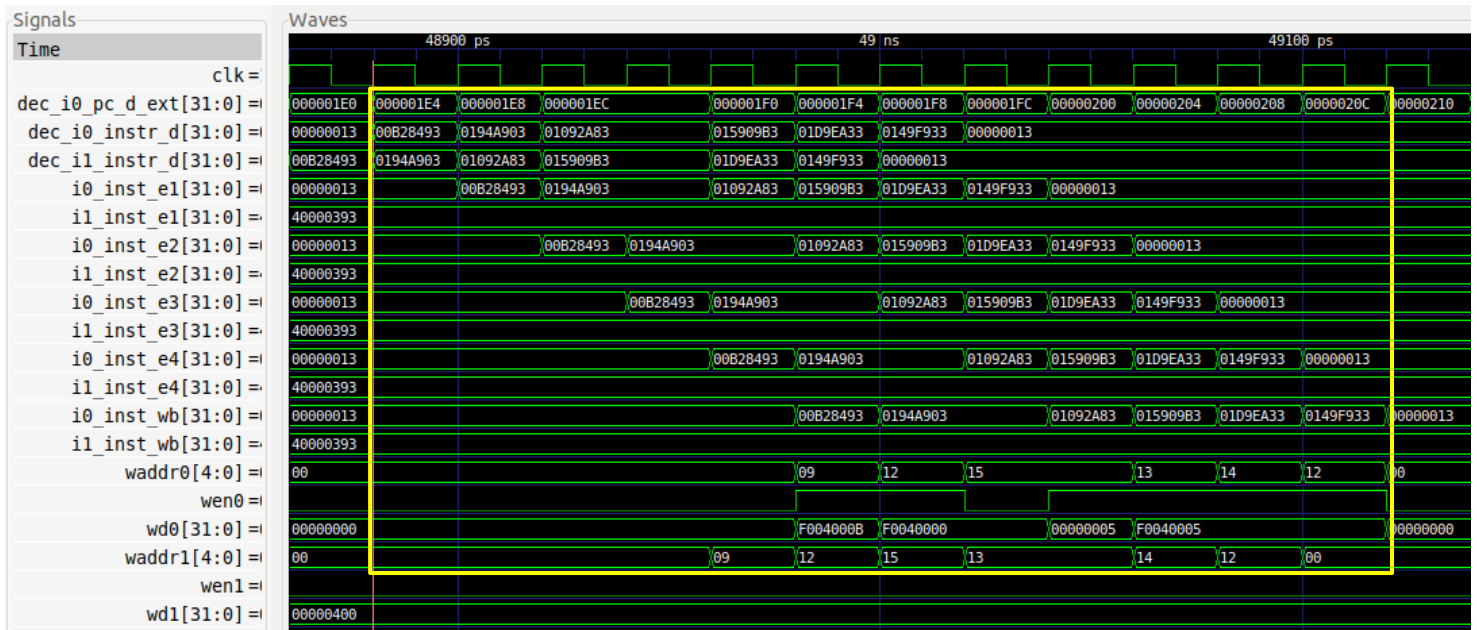
```

请注意，在该计算中，必须删除加入的所有额外指令：首先是addi指令，然后是29条nop指令，最后是bne指令。上述指令均可以在 $\frac{1}{2}$ 个周期内执行（在执行我们程序的最后一条指令时，会有一个空闲时隙）。

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

与我们在仿真中观察到的情况完全相同。

c)



每个周期执行1条指令，唯一的例外是当两次装载（0x0194a903和0x01092a83）之间存在数据冒险时，此时系统在流水线中插入一个气泡并丢弃1个周期。

```

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, log2
--- 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
  
```

和之前一样，为了计算IPC，必须删除31条额外的指令。这些指令均可在一个周期内执行。

因此： $IPC = (37-31) / (38-31) = 6 / 7$

与我们在仿真中观察到的情况完全相同。

7) (以下练习基于DDCARV第7章的练习7.31、7.33和7.35。)

使用以下代码片段重复练习7。

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

PlatformIO项目的路径如下：

[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

