

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。

```

70 // ----- Input flops -----
71
72 rvdffs    #(1) valid_e1_ff    (*, .din(mp.valid),          .dout(valid_e1),          .clk(active_clk),          .en(~freeze));
73
74 rvdff_fpga #(1) rs1_sign_e1_ff (*, .din(mp.rs1_sign),      .dout(rs1_sign_e1),      .clk(exu_mul_c1_e1_clk), .clken(mul_c1_e1_clken), .rawclk(clk));
75 rvdff_fpga #(1) rs2_sign_e1_ff (*, .din(mp.rs2_sign),      .dout(rs2_sign_e1),      .clk(exu_mul_c1_e1_clk), .clken(mul_c1_e1_clken), .rawclk(clk));
76 rvdff_fpga #(1) low_e1_ff     (*, .din(mp.low),            .dout(low_e1),           .clk(exu_mul_c1_e1_clk), .clken(mul_c1_e1_clken), .rawclk(clk));
77 rvdff_fpga #(1) ld_rs1_byp_e1_ff (*, .din(mp.load_mul_rs1_bypass_e1), .dout(load_mul_rs1_bypass_e1), .clk(exu_mul_c1_e1_clk), .clken(mul_c1_e1_clken), .rawclk(clk));
78 rvdff_fpga #(1) ld_rs2_byp_e1_ff (*, .din(mp.load_mul_rs2_bypass_e1), .dout(load_mul_rs2_bypass_e1), .clk(exu_mul_c1_e1_clk), .clken(mul_c1_e1_clken), .rawclk(clk));
79
80 rvdffe #(32) a_e1_ff          (*, .din(a[31:0]),            .dout(a_ff_e1[31:0]),      .en(mul_c1_e1_clken));
81 rvdffe #(32) b_e1_ff          (*, .din(b[31:0]),            .dout(b_ff_e1[31:0]),      .en(mul_c1_e1_clken));
82
83
84
85 // ----- E1 Logic Stage -----
86
87 assign a_e1[31:0]              = (load_mul_rs1_bypass_e1 ? lsu_result_dc3[31:0] : a_ff_e1[31:0]);
88 assign b_e1[31:0]              = (load_mul_rs2_bypass_e1 ? lsu_result_dc3[31:0] : b_ff_e1[31:0]);
89
90 assign rs1_neg_e1              = rs1_sign_e1 & a_e1[31];
91 assign rs2_neg_e1              = rs2_sign_e1 & b_e1[31];
92
93
94 rvdffs    #(1) valid_e2_ff    (*, .din(valid_e1),          .dout(valid_e2),          .clk(active_clk),          .en(~freeze));
95
96 rvdff_fpga #(1) low_e2_ff     (*, .din(low_e1),            .dout(low_e2),           .clk(exu_mul_c1_e2_clk), .clken(mul_c1_e2_clken), .rawclk(clk));
97
98 rvdffe #(33) a_e2_ff          (*, .din({rs1_neg_e1, a_e1[31:0]}), .dout(a_ff_e2[32:0]),      .en(mul_c1_e2_clken));
99 rvdffe #(33) b_e2_ff          (*, .din({rs2_neg_e1, b_e1[31:0]}), .dout(b_ff_e2[32:0]),      .en(mul_c1_e2_clken));
100
101
102
103 // ----- E2 Logic Stage -----
104
105 logic signed [65:0] prod_e2;
106
107 assign prod_e2[65:0]           = a_ff_e2 * b_ff_e2;
108
109 rvdff_fpga #(1) low_e3_ff     (*, .din(low_e2),            .dout(low_e3),           .clk(exu_mul_c1_e3_clk), .clken(mul_c1_e3_clken), .rawclk(clk));
110
111 rvdffe #(64) prod_e3_ff       (*, .din(prod_e2[63:0]),      .dout(prod_e3[63:0]),      .en(mul_c1_e3_clken));
112
113
114
115 // ----- E3 Logic Stage -----
116
117
118
119 assign out[31:0]              = low_e3 ? prod_e3[31:0] : prod_e3[63:32];
120

```

- 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。

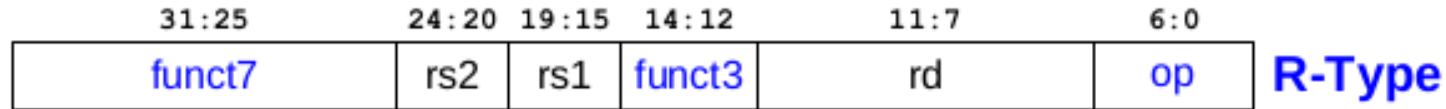
0x03de02b3 → 0000001 11101 11100 000 00101 0110011

**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:



op	funct3	funct7	Type	Instruction	Description	Operation
0110011 (51)	000	0000001	R	mul rd, rs1, rs2	multiply	rd = (rs1 x rs2) _{31:0}

Name	Register Number	Use
zero	x0	Constant value 0
ra	x1	Return address
sp	x2	Stack pointer
gp	x3	Global pointer
tp	x4	Thread pointer
t0-2	x5-7	Temporary variables
s0/fp	x8	Saved variable / Frame pointer
s1	x9	Saved variable
a0-1	x10-11	Function arguments / Return values
a2-7	x12-17	Function arguments
s2-11	x18-27	Saved variables
t3-6	x28-31	Temporary variables

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。

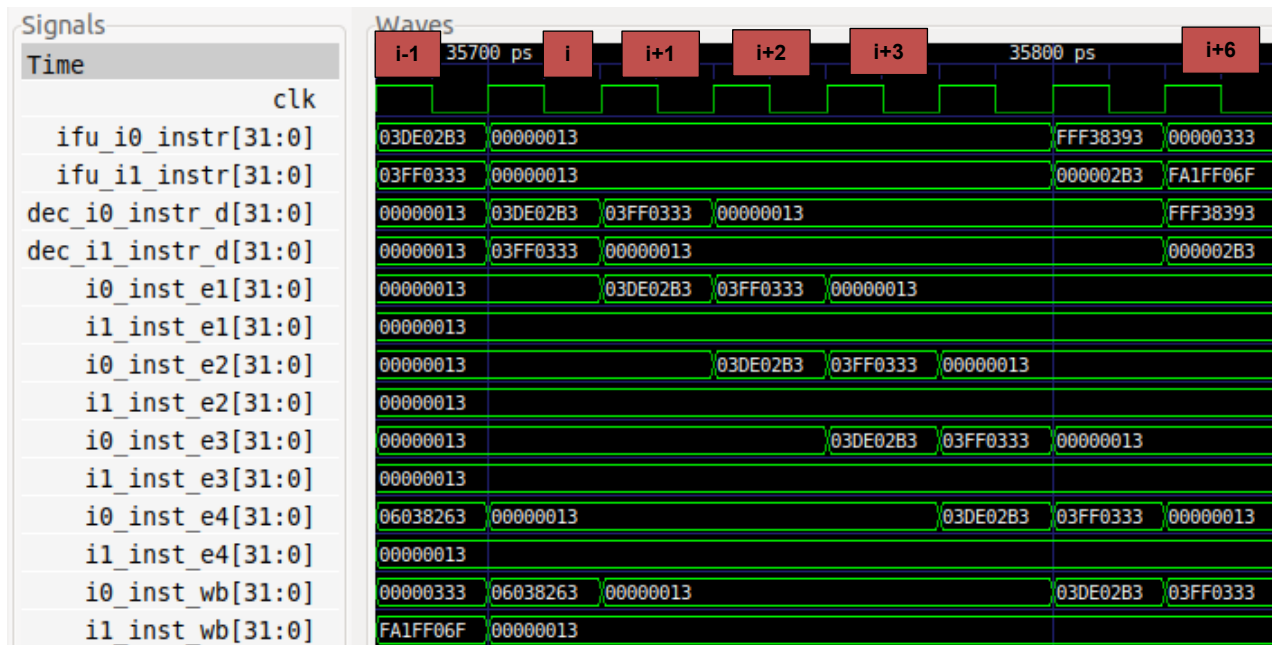
解答请参见实验14的主文档。

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)。

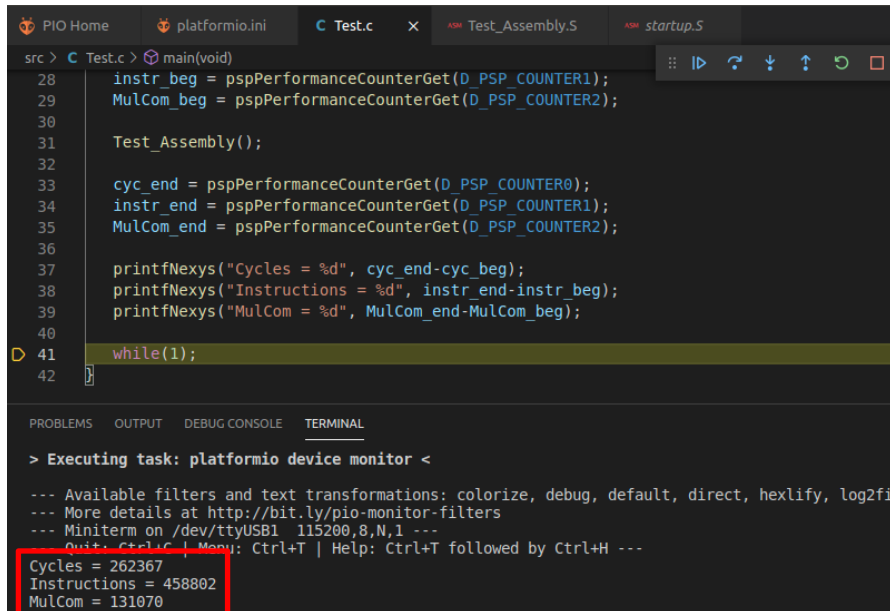


- 在周期*i-1* (图2和图3中未显示) 中, 两条mul指令处于对齐阶段: 第一条指令分配给模块**ifu_aln_ctl**的通路0 (`ifu_i0_instr = 0x03de02b3`), 第二条指令分配给该模块的通路1 (`ifu_i1_instr = 0x03ff0333`)。

- 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。



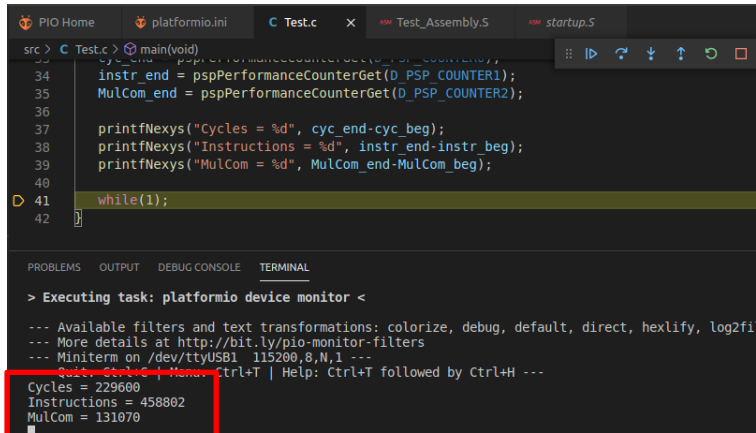
```
src > C Test.c > main(void)
28 instr_beg = pspPerformanceCounterGet(D_PSP_COUNTER1);
29 MulCom_beg = pspPerformanceCounterGet(D_PSP_COUNTER2);
30
31 Test_Assembly();
32
33 cyc_end = pspPerformanceCounterGet(D_PSP_COUNTER0);
34 instr_end = pspPerformanceCounterGet(D_PSP_COUNTER1);
35 MulCom_end = pspPerformanceCounterGet(D_PSP_COUNTER2);
36
37 printfNexys("Cycles = %d", cyc_end-cyc_beg);
38 printfNexys("Instructions = %d", instr_end-instr_beg);
39 printfNexys("MulCom = %d", MulCom_end-MulCom_beg);
40
41 while(1);
42
```

```
> 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 = 262367
Instructions = 458802
MulCom = 131070
```

$IPC = 458000 / 262000 = 1.748$ 。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。

```
22 REPEAT:
23     beq t2, zero, OUT    # Stay in the loop?
24     mul t0, t3, t4       # t0 = t3 * t4
25     add t2, t2, -1
26     mul t1, t5, t6       # t1 = t5 * t6
27     add t0, zero, zero
28     add t1, zero, zero
29     j REPEAT
30 OUT:
```



```
src > C Test.c > main(void)
34     instr_end = pspPerformanceCounterGet(D_PSP_COUNTER1);
35     MulCom_end = pspPerformanceCounterGet(D_PSP_COUNTER2);
36
37     printfNexys("Cycles = %d", cyc_end-cyc_beg);
38     printfNexys("Instructions = %d", instr_end-instr_beg);
39     printfNexys("MulCom = %d", MulCom_end-MulCom_beg);
40
41     while(1);
42 }
```

```
> 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 = 229600
Instructions = 458802
MulCom = 131070
```

$$\text{IPC} = 458000 / 229000 = 2$$

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=rv32ima -march=rv32ima`
 修改为：
`build_flags = -Wa,-march=rv32ia -march=rv32ia`
 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）：

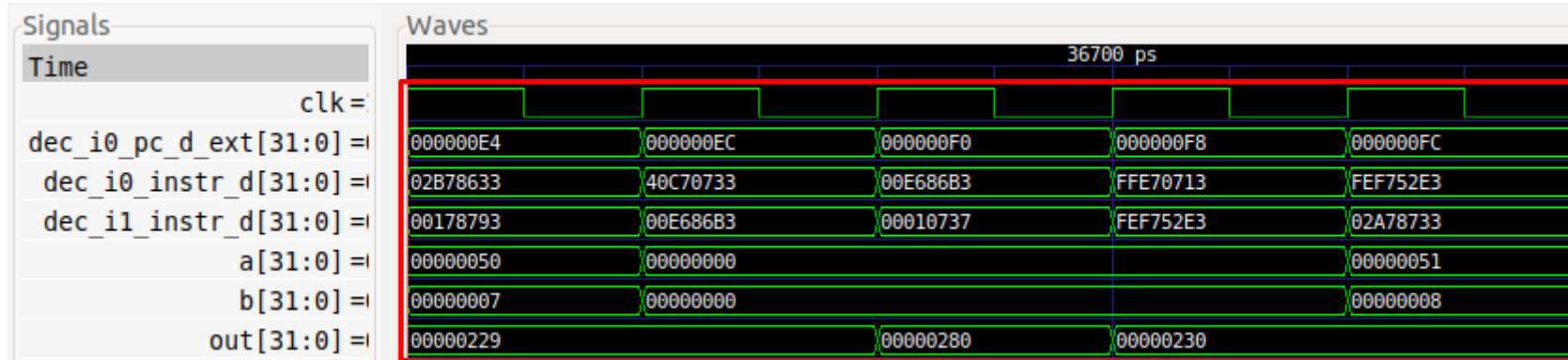
```
11  int Test_C(int a, int d)
12  {
13      int b, c, e=0, i=1;
14      do {
15          b = a*i;
16          c = d*i;
17          i = i+1;
18          e = e + (b-c);
19      } while(i<65535);
20      return(e);
21  }
```

```

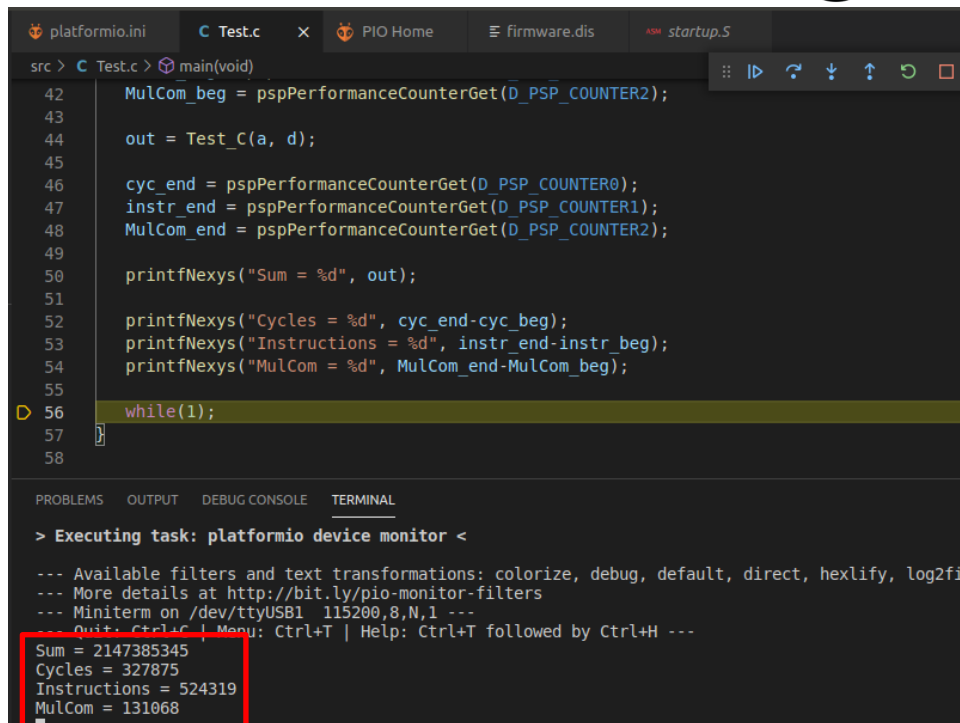
66 000000d8 <Test_C>:
67      d8: 00100793      li  a5,1
68      dc: 00000693      li  a3,0
69      e0: 02a78733      mul a4,a5,a0
70      e4: 02b78633      mul a2,a5,a1
71      e8: 00178793      addi a5,a5,1
72      ec: 40c70733      sub a4,a4,a2
73      f0: 00e686b3      add a3,a3,a4
74      f4: 00010737      lui a4,0x10
75      f8: ffe70713      addi a4,a4,-2 # fffe <_sp+0xc386>
76      fc: fef752e3      bge a4,a5,e0 <Test_C+0x8>
77      100: 00068513      mv  a0,a3
78      104: 00008067      ret

```

- C程序的仿真:



- 硬件计数器:



```

src > C Test.c > main(void)
42 MulCom_beg = pspPerformanceCounterGet(D_PSP_COUNTER2);
43
44 out = Test_C(a, d);
45
46 cyc_end = pspPerformanceCounterGet(D_PSP_COUNTER0);
47 instr_end = pspPerformanceCounterGet(D_PSP_COUNTER1);
48 MulCom_end = pspPerformanceCounterGet(D_PSP_COUNTER2);
49
50 printfNexys("Sum = %d", out);
51
52 printfNexys("Cycles = %d", cyc_end-cyc_beg);
53 printfNexys("Instructions = %d", instr_end-instr_beg);
54 printfNexys("MulCom = %d", MulCom_end-MulCom_beg);
55
56 while(1);
57
58

```

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 ---

Sum = 2147385345
Cycles = 327875
Instructions = 524319
MulCom = 131068

```

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。

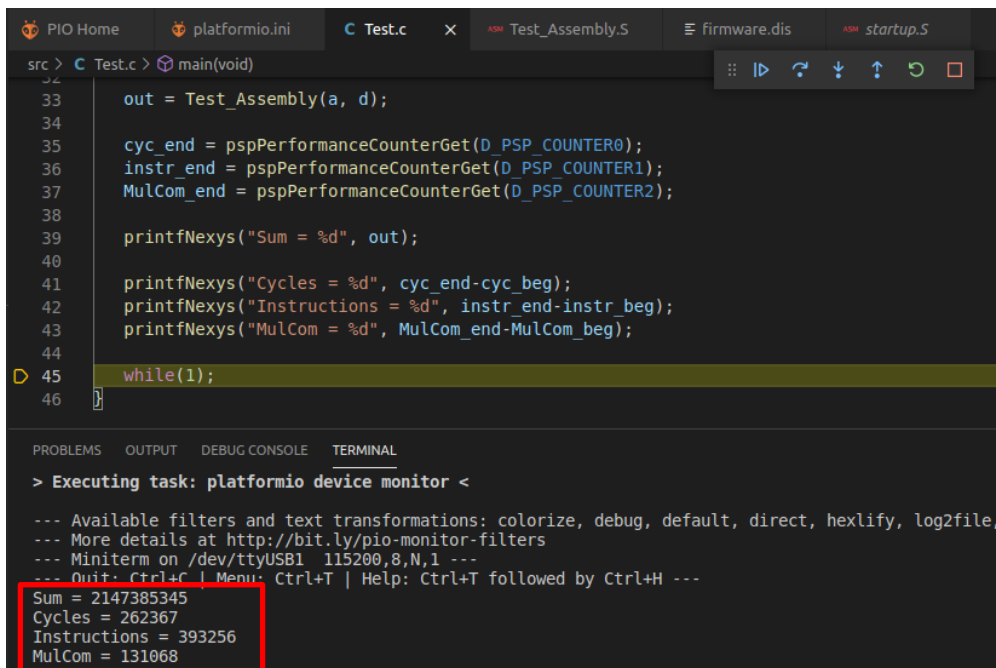
- 汇编程序位于：

[RVfpgaPath]/RVfpga/Labs/RVfpgaLabsSolutions/Programs_Solutions/Lab14/MUL_Instr_Accumul_Assembly

```

127 000001c4 <Test_Assembly>:
128 1c4: 00100393      li t2,1
129 1c8: 00000e93      li t4,0
130 1cc: 00000f93      li t6,0
131 1d0: 00010637      lui a2,0x10
132 1d4: fff60613      addi a2,a2,-1 # ffff <_sp+0xc567>
133 1d8: 00050e33      add t3,a0,zero
134 1dc: 00058f33      add t5,a1,zero
135
136 000001e0 <REPEAT>:
137 1e0: 027e02b3      mul t0,t3,t2
138 1e4: 027f0333      mul t1,t5,t2
139 1e8: 00138393      addi t2,t2,1
140 1ec: 40628eb3      sub t4,t0,t1
141 1f0: 01df8fb3      add t6,t6,t4
142 1f4: fec396e3      bne t2,a2,1e0 <REPEAT>
143 1f8: 00018533      add a0,t6,zero
144 1fc: 00008067      ret

```



The screenshot shows the PlatformIO IDE with the following components:

- Editor:** Displays the C code in `Test.c`. The code includes headers, defines performance counters, and calls the assembly test function. A `while(1);` loop is at the bottom.
- Terminal:** Shows the output of the program execution. The output is:


```

Sum = 2147385345
Cycles = 262367
Instructions = 393256
MulCom = 131068

```

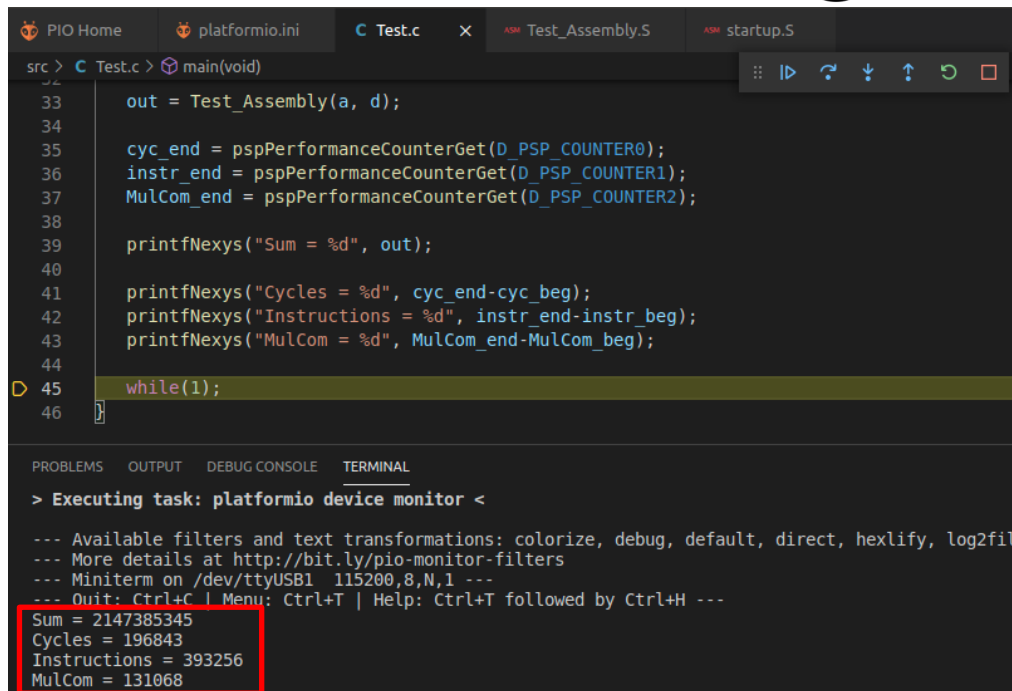
求和的结果相同，因为程序相同。

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：

```
15  li  t2, 0x1
16  li  t4, 0x0
17  li  t6, 0x0
18  li  a2, 0xFFFF
19  add t3, a0, zero
20  add t5, a1, zero
21
22  REPEAT:
23      mul t0, t3, t2      # t0 = t3 * t2
24      mul t1, t5, t2      # t1 = t5 * t2
25      sub t4, t0, t1
26      add t2, t2, 1
27      add t6, t6, t4
28      bne t2, a2, REPEAT  # Repeat the loop
29
30  add a0, t6, zero
```



```

src > C Test.c > main(void)
33     out = Test_Assembly(a, d);
34
35     cyc_end = pspPerformanceCounterGet(D_PSP_COUNTER0);
36     instr_end = pspPerformanceCounterGet(D_PSP_COUNTER1);
37     MulCom_end = pspPerformanceCounterGet(D_PSP_COUNTER2);
38
39     printfNexys("Sum = %d", out);
40
41     printfNexys("Cycles = %d", cyc_end-cyc_beg);
42     printfNexys("Instructions = %d", instr_end-instr_beg);
43     printfNexys("MulCom = %d", MulCom_end-MulCom_beg);
44
45     while(1);
46

```

```

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 ---

Sum = 2147385345
Cycles = 196843
Instructions = 393256
MulCom = 131068

```

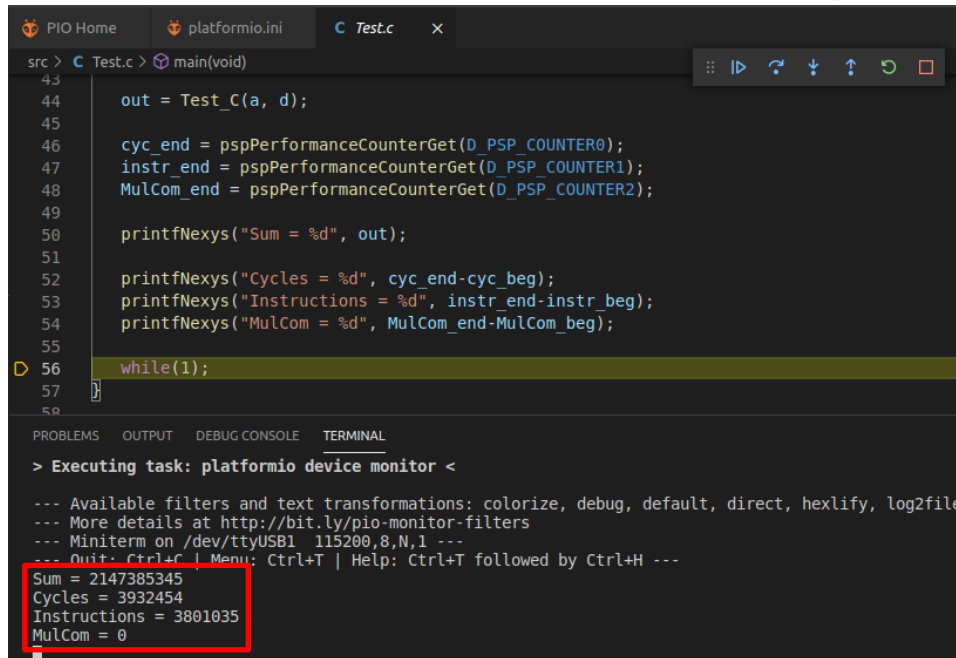
求和的结果相同，因为程序相同。

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:



The screenshot shows a code editor with a C program in `Test.c`. The program calculates the sum of two numbers, `a` and `d`, and prints the result. It also uses performance counters to measure the number of cycles, instructions, and multiplies completed. The output in the terminal shows the sum is 2147385345, cycles are 3932454, instructions are 3801035, and multiplies completed are 0.

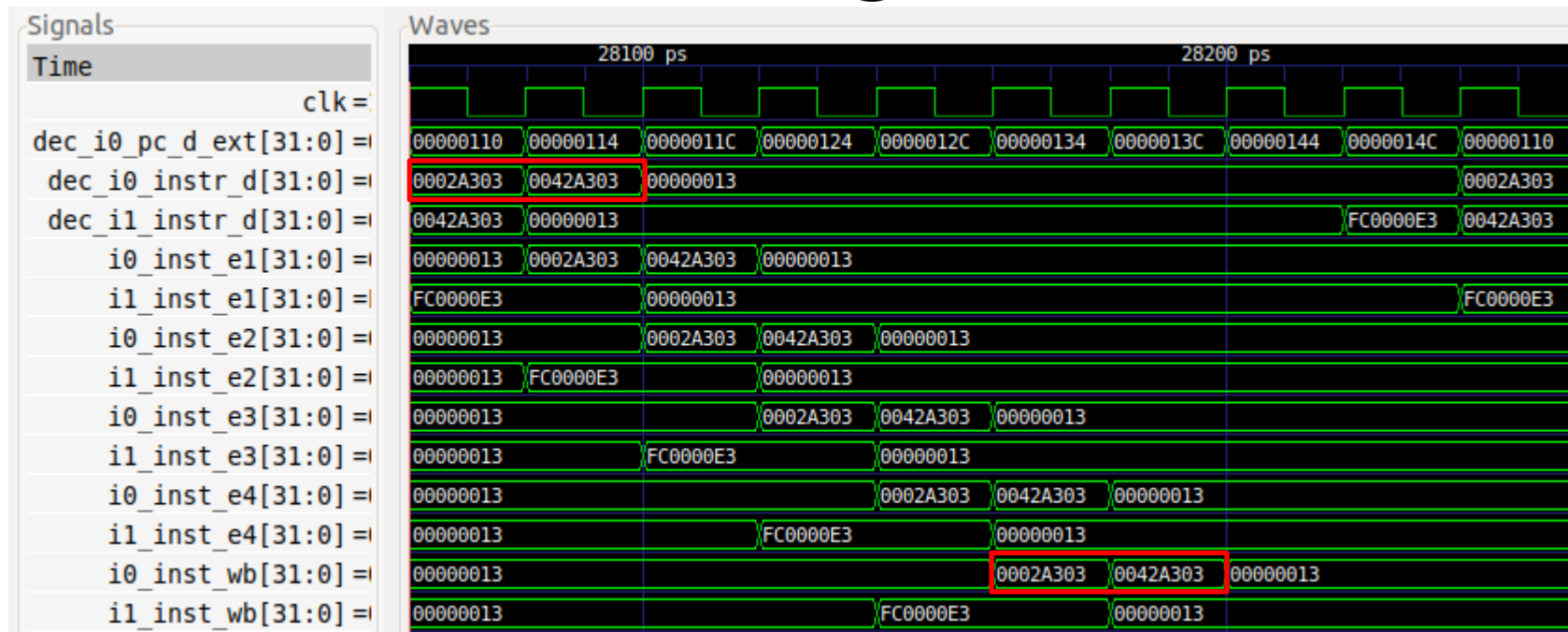
```

src > C Test.c > main(void)
43
44     out = Test_C(a, d);
45
46     cyc_end = pspPerformanceCounterGet(D_PSP_COUNTER0);
47     instr_end = pspPerformanceCounterGet(D_PSP_COUNTER1);
48     MulCom_end = pspPerformanceCounterGet(D_PSP_COUNTER2);
49
50     printfNexys("Sum = %d", out);
51
52     printfNexys("Cycles = %d", cyc_end-cyc_beg);
53     printfNexys("Instructions = %d", instr_end-instr_beg);
54     printfNexys("MulCom = %d", MulCom_end-MulCom_beg);
55
56     while(1);
57
58
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 ---
Sum = 2147385345
Cycles = 3932454
Instructions = 3801035
MulCom = 0


```

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。

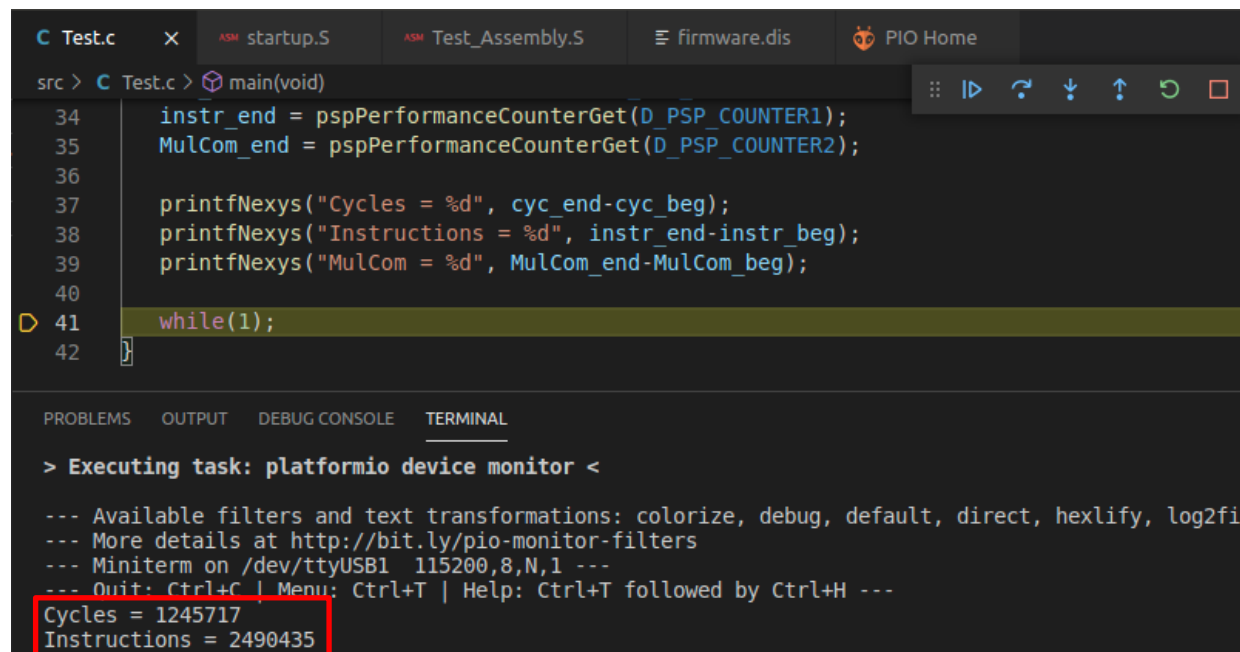
不提供解答。

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：



```

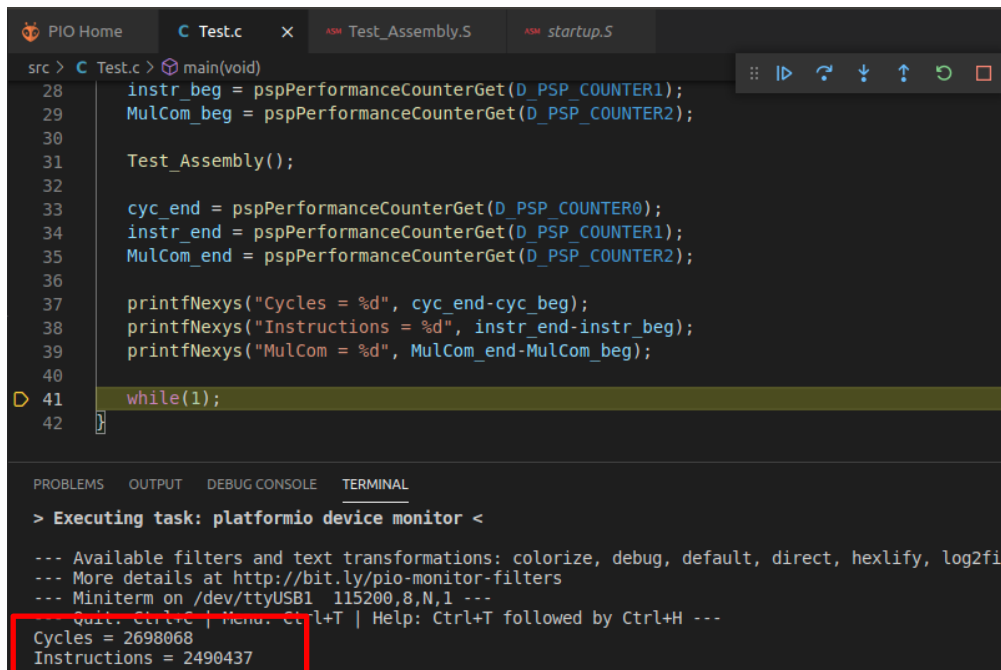
C Test.c x ASM startup.S ASM Test_Assembly.S firmware.dis PIO Home
src > C Test.c > main(void)
34 instr_end = pspPerformanceCounterGet(D_PSP_COUNTER1);
35 MulCom_end = pspPerformanceCounterGet(D_PSP_COUNTER2);
36
37 printfNexys("Cycles = %d", cyc_end-cyc_beg);
38 printfNexys("Instructions = %d", instr_end-instr_beg);
39 printfNexys("MulCom = %d", MulCom_end-MulCom_beg);
40
41 while(1);
42

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 = 1245717
Instructions = 2490435
  
```

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

阻塞装载：



```

src > C Test.c > main(void)
28   instr_beg = pspPerformanceCounterGet(D_PSP_COUNTER1);
29   MulCom_beg = pspPerformanceCounterGet(D_PSP_COUNTER2);
30
31   Test_Assembly();
32
33   cyc_end = pspPerformanceCounterGet(D_PSP_COUNTER0);
34   instr_end = pspPerformanceCounterGet(D_PSP_COUNTER1);
35   MulCom_end = pspPerformanceCounterGet(D_PSP_COUNTER2);
36
37   printfNexys("Cycles = %d", cyc_end-cyc_beg);
38   printfNexys("Instructions = %d", instr_end-instr_beg);
39   printfNexys("MulCom = %d", MulCom_end-MulCom_beg);
40
41   while(1);
42

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

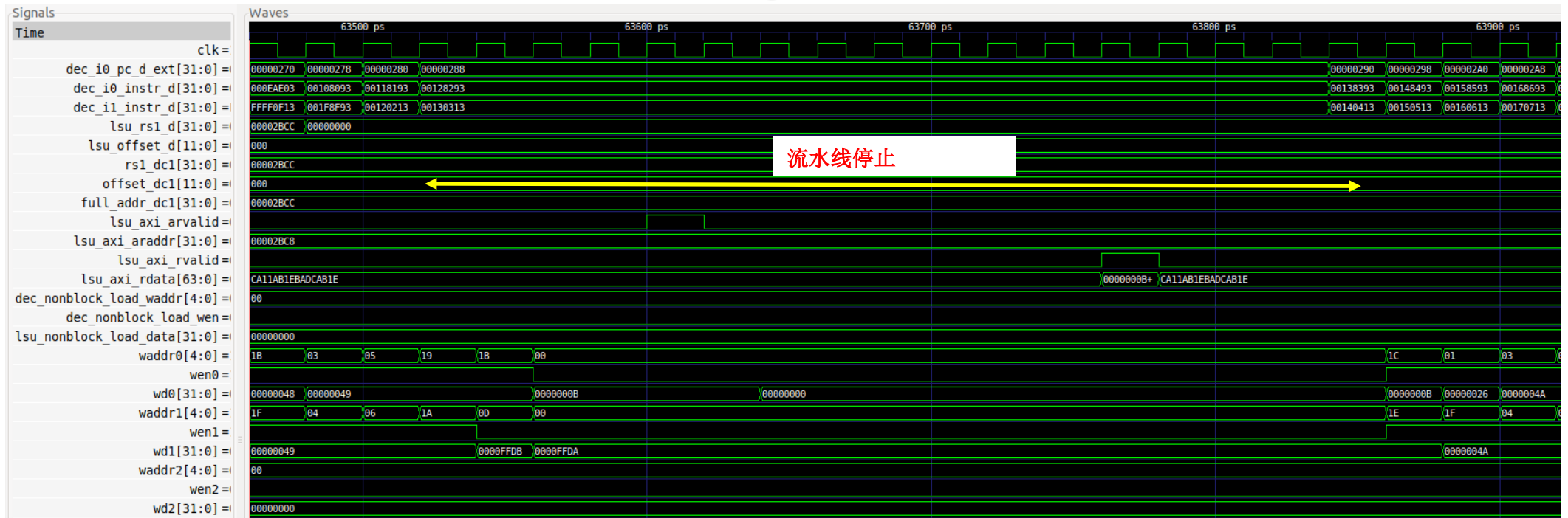
```

> 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 = 2698068
Instructions = 2490437

```

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。



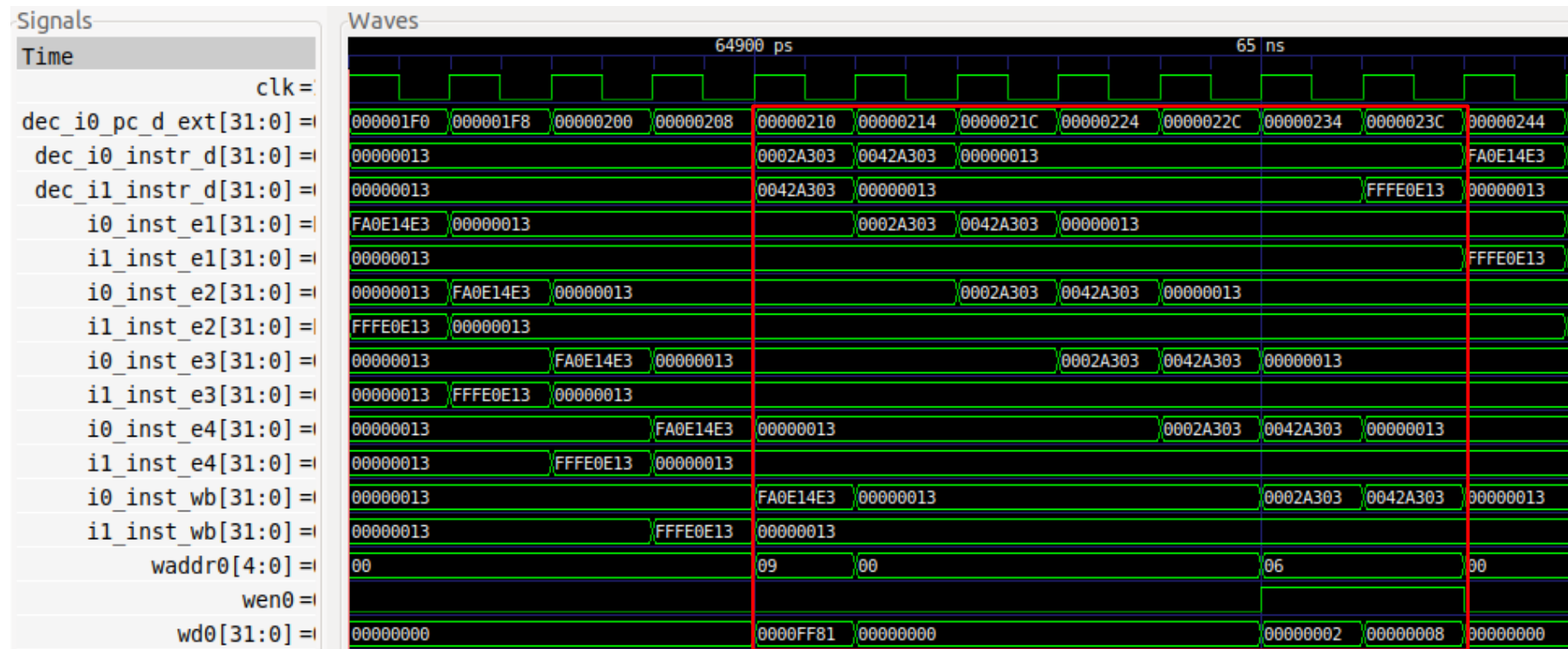
练习

- 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：

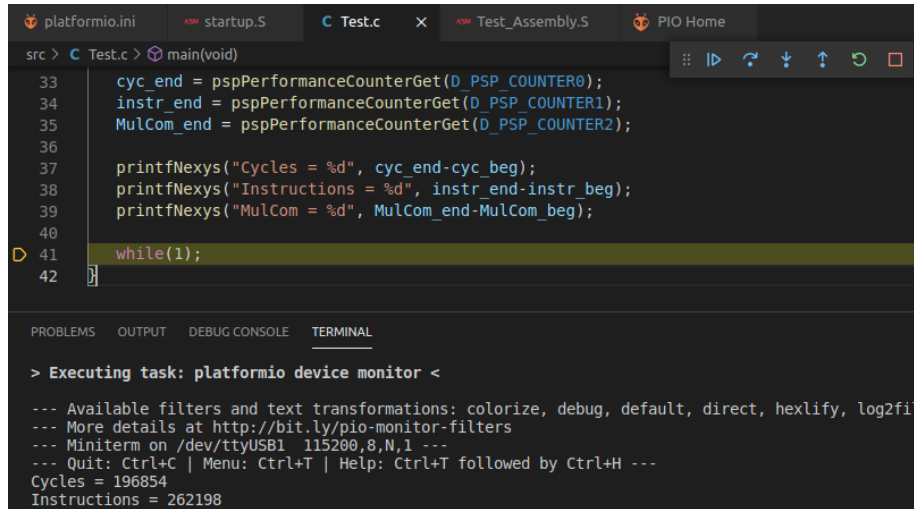
```
210: 0002a303      lw    t1,0(t0)
214: 0042a303      lw    t1,4(t0)
```

- 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:



```

src > C Test.c > main(void)
33     cyc_end = pspPerformanceCounterGet(D_PSP_COUNTER0);
34     instr_end = pspPerformanceCounterGet(D_PSP_COUNTER1);
35     MulCom_end = pspPerformanceCounterGet(D_PSP_COUNTER2);
36
37     printfNexys("Cycles = %d", cyc_end-cyc_beg);
38     printfNexys("Instructions = %d", instr_end-instr_beg);
39     printfNexys("MulCom = %d", MulCom_end-MulCom_beg);
40
41     while(1);
42
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 = 196854
Instructions = 262198
  
```

$$\text{IPC} = 262 / 196 = 1.33$$

两次连续存储:

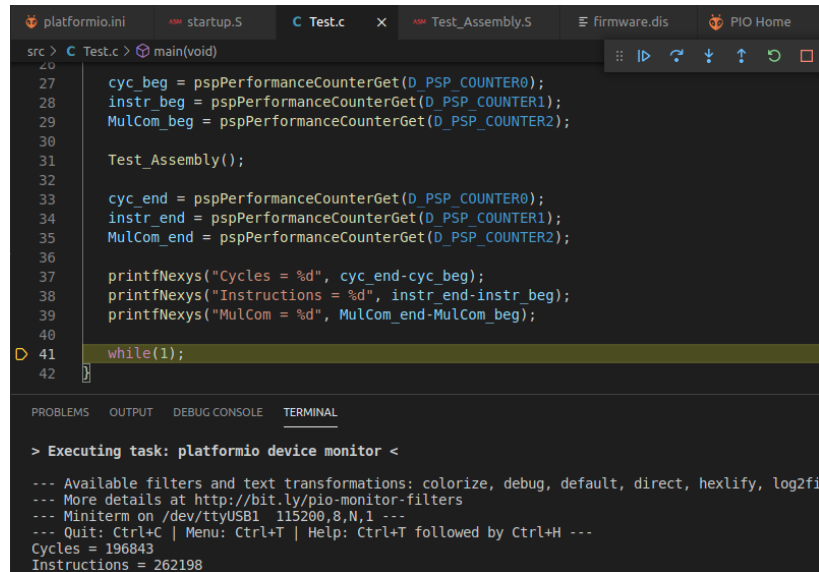
210:	0062a023	sw	t1,0(t0)
214:	0062a223	sw	t1,4(t0)

- 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：



```

src > C Test.c > main(void)
27   cyc_beg = pspPerformanceCounterGet(D_PSP_COUNTER0);
28   instr_beg = pspPerformanceCounterGet(D_PSP_COUNTER1);
29   MulCom_beg = pspPerformanceCounterGet(D_PSP_COUNTER2);
30
31   Test_Assembly();
32
33   cyc_end = pspPerformanceCounterGet(D_PSP_COUNTER0);
34   instr_end = pspPerformanceCounterGet(D_PSP_COUNTER1);
35   MulCom_end = pspPerformanceCounterGet(D_PSP_COUNTER2);
36
37   printfNexys("Cycles = %d", cyc_end-cyc_beg);
38   printfNexys("Instructions = %d", instr_end-instr_beg);
39   printfNexys("MulCom = %d", MulCom_end-MulCom_beg);
40
41   while(1);
42

```

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 = 196843
 Instructions = 262198

$$\text{IPC} = 262 / 196 = 1.33$$

2. （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。

- 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。

- b. 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？
- c. 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。

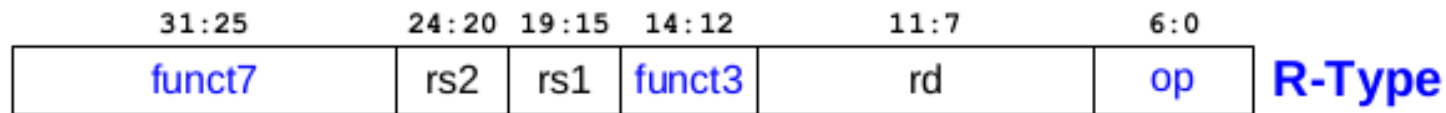
0x03de42b3 → 0000001 11101 11100 100 00101 0110011

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:



op	funct3	funct7	Type	Instruction	Description	Operation
0110011 (51)	100	0000001	R	div rd, rs1, rs2	divide (signed)	rd = rs1 / rs2

Name	Register Number	Use
zero	x0	Constant value 0
ra	x1	Return address
sp	x2	Stack pointer
gp	x3	Global pointer
tp	x4	Thread pointer
t0-2	x5-7	Temporary variables
s0/fp	x8	Saved variable / Frame pointer
s1	x9	Saved variable
a0-1	x10-11	Function arguments / Return values
a2-7	x12-17	Function arguments
s2-11	x18-27	Saved variables
t3-6	x28-31	Temporary variables

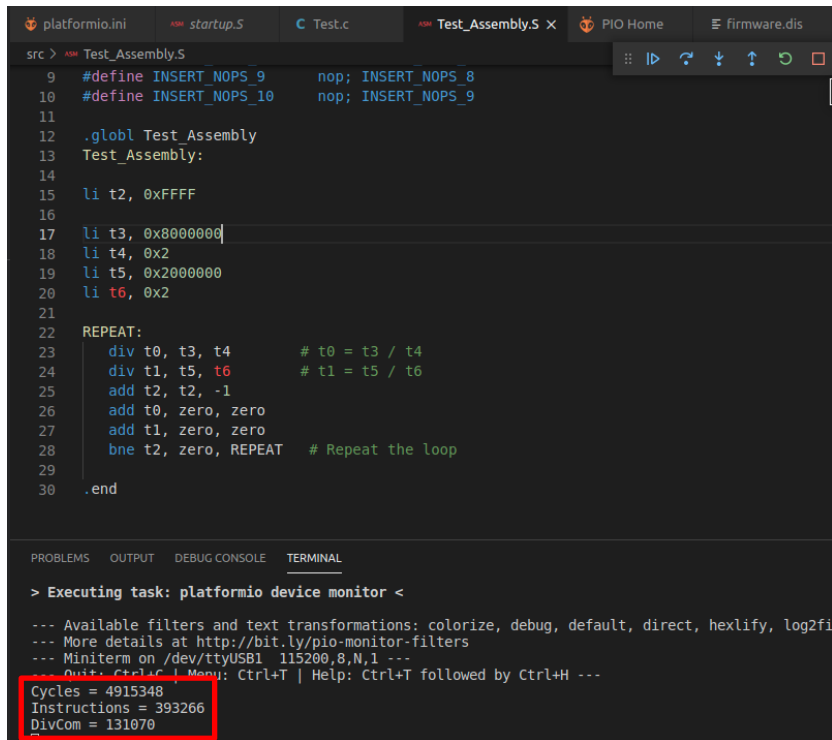
任务： 在自己的计算机上重复图9中的仿真过程，以进行详细的分析。

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。



```

src > Test_Assembly.S
9  #define INSERT_NOPS_9    nop; INSERT_NOPS_8
10 #define INSERT_NOPS_10   nop; INSERT_NOPS_9
11
12 .globl Test_Assembly
13 Test_Assembly:
14
15     li t2, 0xFFFF
16
17     li t3, 0x80000000
18     li t4, 0x2
19     li t5, 0x20000000
20     li t6, 0x2
21
22 REPEAT:
23     div t0, t3, t4        # t0 = t3 / t4
24     div t1, t5, t6        # t1 = t5 / t6
25     add t2, t2, -1
26     add t0, zero, zero
27     add t1, zero, zero
28     bne t2, zero, REPEAT  # Repeat the loop
29
30 .end

```

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 = 4915348
 Instructions = 393266
 DivCom = 131070

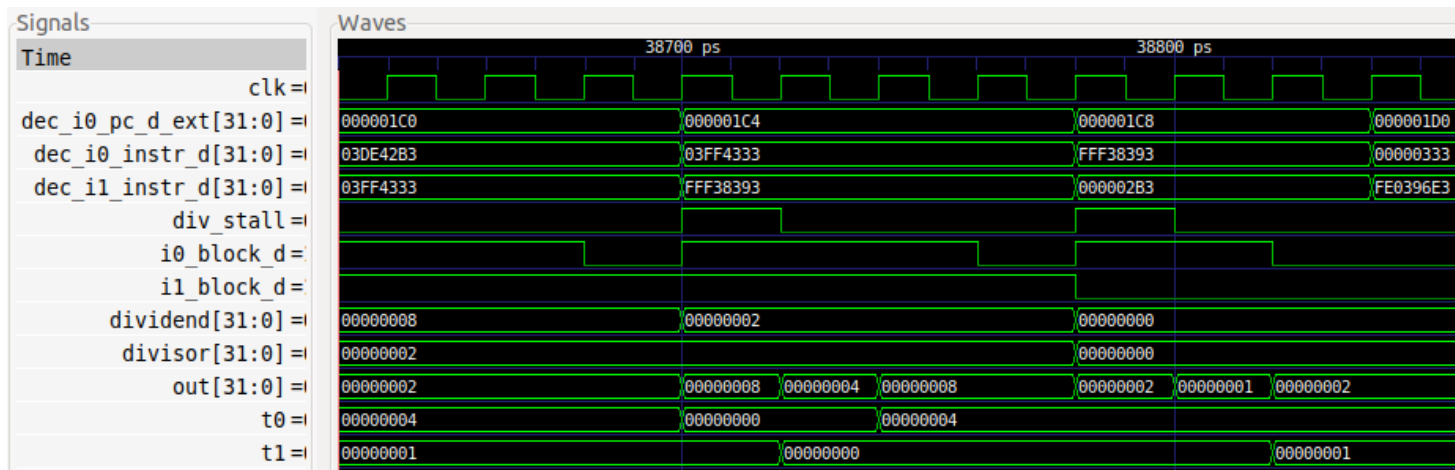
$CPI = 4910000 / 393000 = 12$. 考虑到每个除法运算大约需要34个周期来执行，而其他指令各需要 $\frac{1}{2}$ 个周期（上面是能够预测的近似情况），近似的理论计算如下：在 $34 + 34 + \frac{1}{2} + \frac{1}{2} + \frac{1}{2} + \frac{1}{2}$ 个周期内执行6条指令 $\rightarrow CPI = 70 / 6 = 11$

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。

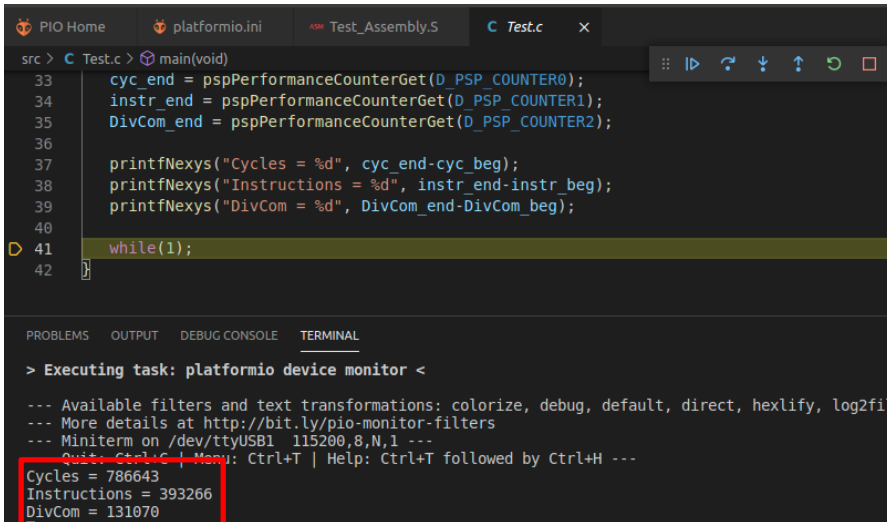
```

12 .globl Test_Assembly
13 Test_Assembly:
14
15 li t2, 0xFFFF
16
17 li t3, 0x8
18 li t4, 0x2
19 li t5, 0x2
20 li t6, 0x2
21
22 REPEAT:
23 div t0, t3, t4      # t0 = t3 / t4
24 div t1, t5, t6      # t1 = t5 / t6
25 add t2, t2, -1
26 add t0, zero, zero
27 add t1, zero, zero
28 bne t2, zero, REPEAT # Repeat the loop
29
30 .end

```



现在，除法计算仅需大约5个周期。



The screenshot shows the PlatformIO IDE with a C program in `Test.c`. The program calculates performance metrics using PSP performance counters. The output in the terminal shows the results of the execution:

```

> 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 = 786643
Instructions = 393266
DivCom = 131070
  
```

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=rv32ima -march=rv32ima`

修改为:

```
build_flags = -Wa,-march=rv32ia -march=rv32ia
```

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。

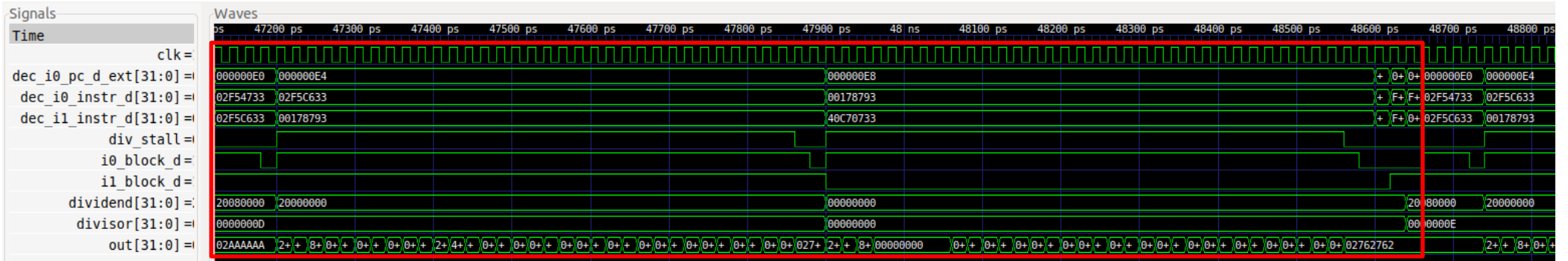
- C程序（原版和反汇编）：

```
11 int Test_C(int a, int d)
12 {
13     int b, c, e=0, i=1;
14     do {
15         b = a/i;
16         c = d/i;
17         i = i+1;
18         e = e + (b-c);
19     } while(i<65535);
20     return(e);
21 }
```

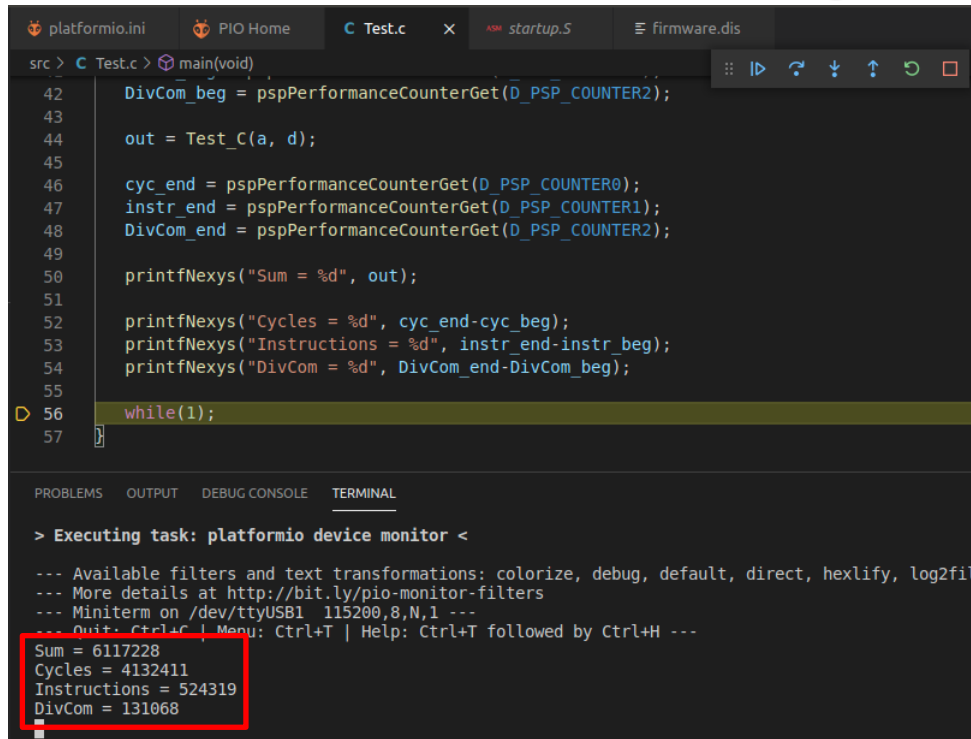
```
66 000000d8 <Test_C>:
67 d8: 00100793      li a5,1
68 dc: 00000693      li a3,0
69 e0: 02f54733      div a4,a0,a5
70 e4: 02f5c633      div a2,a1,a5
71 e8: 00178793      addi a5,a5,1
72 ec: 40c70733      sub a4,a4,a2
73 f0: 00e686b3      add a3,a3,a4
74 f4: 00010737      lui a4,0x10
75 f8: ffe70713      addi a4,a4,-2 # fffe <_sp+0xc386>
76 fc: fef752e3      bge a4,a5,e0 <Test_C+0x8>
77 100: 00068513      mv a0,a3
78 104: 00008067      ret
```

Imagination
university programme

- C程序的仿真:



- 硬件计数器:



The screenshot shows the PlatformIO IDE with a C program in `Test.c`. The program calculates the sum of integers from 1 to 100, measures the number of cycles, instructions, and divcom operations, and prints these values. The code is as follows:

```

src > C Test.c > main(void)
42 DivCom_beg = pspPerformanceCounterGet(D_PSP_COUNTER2);
43
44 out = Test_C(a, d);
45
46 cyc_end = pspPerformanceCounterGet(D_PSP_COUNTER0);
47 instr_end = pspPerformanceCounterGet(D_PSP_COUNTER1);
48 DivCom_end = pspPerformanceCounterGet(D_PSP_COUNTER2);
49
50 printfNexys("Sum = %d", out);
51
52 printfNexys("Cycles = %d", cyc_end-cyc_beg);
53 printfNexys("Instructions = %d", instr_end-instr_beg);
54 printfNexys("DivCom = %d", DivCom_end-DivCom_beg);
55
56 while(1);
57

```

The terminal output shows the results of the execution:

```

> 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 ---

Sum = 6117228
Cycles = 4132411
Instructions = 524319
DivCom = 131068

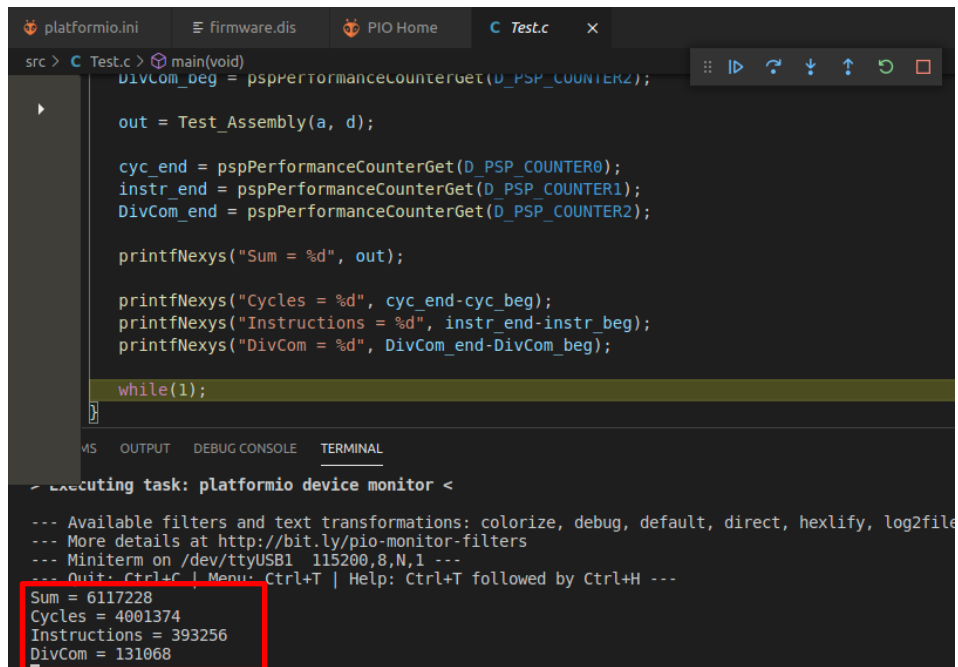
```

- 汇编程序位于:
[RVfpgaPath]/RVfpga/Labs/RVfpgaLabsSolutions/Programs_Solutions/Lab14/DIV_Instr_Accumul_Assembly


```

128 000001c8 <Test_Assembly>:
129 1c8: 00100393      li    t2,1
130 1cc: 00000e93      li    t4,0
131 1d0: 00000f93      li    t6,0
132 1d4: 00010637      lui   a2,0x10
133 1d8: fff60613      addi  a2,a2,-1 # ffff <_sp+0xc377>
134 1dc: 00050e33      add  t3,a0,zero
135 1e0: 00058f33      add  t5,a1,zero
136
137 000001e4 <REPEAT>:
138 1e4: 027e42b3      div  t0,t3,t2
139 1e8: 027f4333      div  t1,t5,t2
140 1ec: 00138393      addi  t2,t2,1
141 1f0: 40628eb3      sub  t4,t0,t1
142 1f4: 01df8fb3      add  t6,t6,t4
143 1f8: fec396e3      bne  t2,a2,1e4 <REPEAT>
144 1fc: 000f8533      add  a0,t6,zero
145 200: 00008067      ret

```



```

platformio.ini  firmware.dis  PIO Home  C Test.c  x
src > C Test.c > main(void)
    DivCom_beg = pspPerformanceCounterGet(D_PSP_COUNTER2);

    out = Test_Assembly(a, d);

    cyc_end = pspPerformanceCounterGet(D_PSP_COUNTER0);
    instr_end = pspPerformanceCounterGet(D_PSP_COUNTER1);
    DivCom_end = pspPerformanceCounterGet(D_PSP_COUNTER2);

    printfNexys("Sum = %d", out);

    printfNexys("Cycles = %d", cyc_end-cyc_beg);
    printfNexys("Instructions = %d", instr_end-instr_beg);
    printfNexys("DivCom = %d", DivCom_end-DivCom_beg);

    while(1);
}

MS  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 ---

Sum = 6117228
Cycles = 4001374
Instructions = 393256
DivCom = 131068

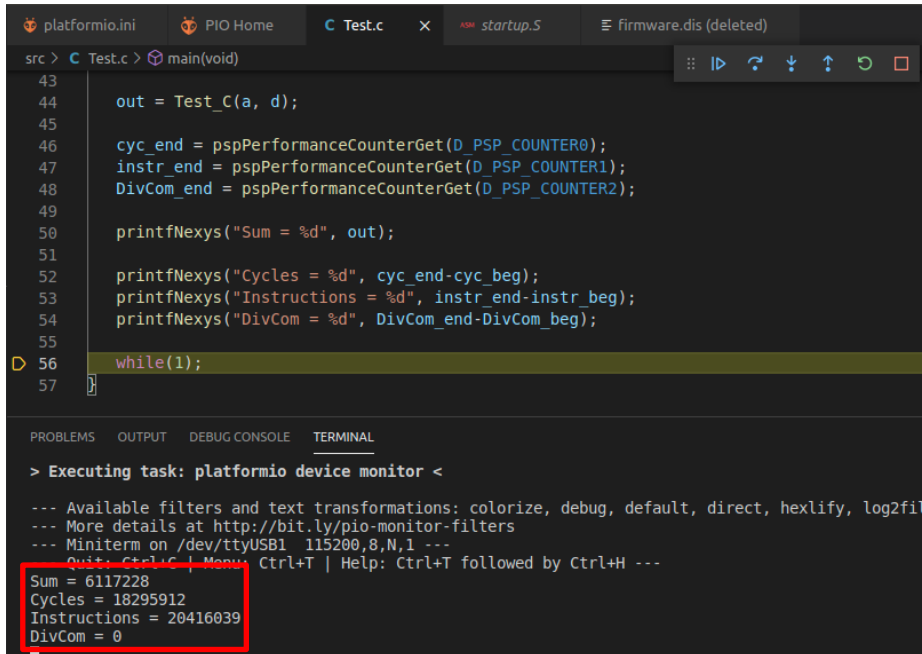
```

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：



The screenshot shows a code editor with a C program in `Test.c`. The code calculates a sum and uses performance counters to measure cycles, instructions, and divcom. A `while(1);` loop is present at line 56. Below the code, the terminal window shows the execution output:

```
> 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 ---

Sum = 6117228
Cycles = 18295912
Instructions = 20416039
DivCom = 0
```

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。