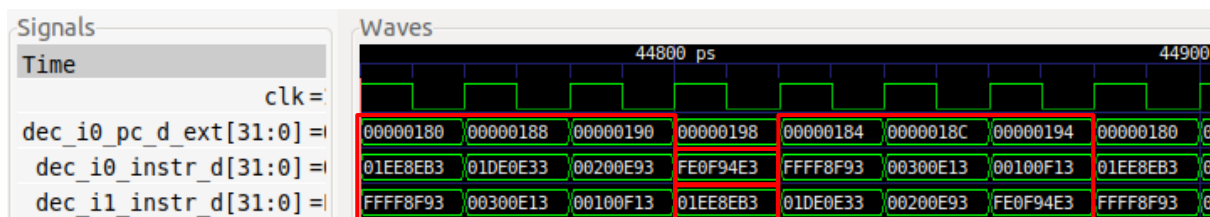


任务

任务： 在自己的计算机上重复图5中的仿真过程。可以使用以下位置提供的.tcl文件：
[RVfpgaPath]/RVfpga/Labs/Lab15/DataHazards_AL-AL/test_Basic.tcl。

解答请参见实验15的主文档。

任务： 删除图2示例中的所有nop指令。对于循环的两次连续迭代，绘制与图3类似的图，然后通过将其与Verilator仿真进行比较来分析并确认此图是否正确，最后在板上执行程序时使用性能计数器计算IPC。



每次迭代需要3.5个周期来执行并且没有暂停。

```
src > C Test.c x Test_Assembly.S startup.S
src > C Test.c > main(void)
27   pspPerformanceCounterGet(D_PSP_COUNTER0);
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 = 229619
Instructions = 458787
```

IPC是理想IPC: $IPC = 458 / 229 = 2$.

任务： 在图2的示例中，删除所有nop指令并将add t6,t6,-1指令移至add t3,t3,t4指令之后，然后重新检查仿真中以及板上的程序。在这个调整顺序的程序中，两条相关add指令（add t4,t4,t5和add t3,t3,t4）在同一周期到达译码阶段，这将影响性能。通过仿真和板上执行来说明这些变化的影响。

测试将相关add指令替换为其他相关指令时的类似情况，例如：

- add t4,t4,t5

```

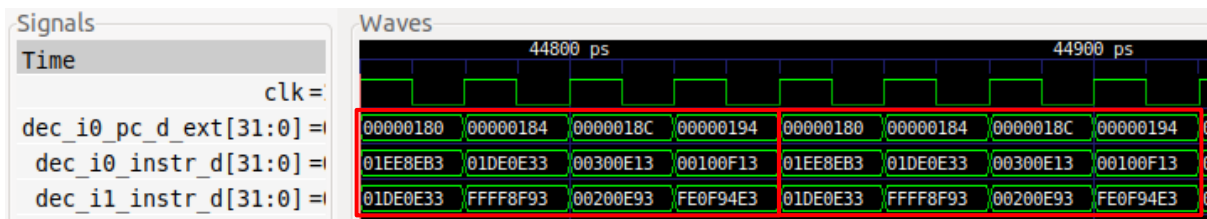
mul t3,t3,t4

-   add t4,t4,t5
    div t3,t3,t4

-   add t4,t4,t5
    lw  t3, 0(t4)

```

- 两条add指令：



现在每次迭代需要4个周期来执行，因为相关add指令必须暂停1个周期，原因在于其输入操作数之一在第一条add指令于EX1阶段执行并转发结果之前不可用。

```

PIO Home  C Test.c  Test_Assembly.S x  startup.S
src > Test_Assembly.S
17 Test_Assembly:
18
19 li t3, 0x3
20 li t4, 0x2
21 li t5, 0x1
22 li t6, 0xFFFF
23
24 REPEAT:
25     add t4, t4, t5      # t4 = t4 + t5
26     add t3, t3, t4      # t3 = t3 + t4
27     add t6, t6, -1
28     li t3, 0x3
29     li t4, 0x2
30     li t5, 0x1
31     bne t6, zero, REPEAT # Repeat the loop
32
33 .end

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

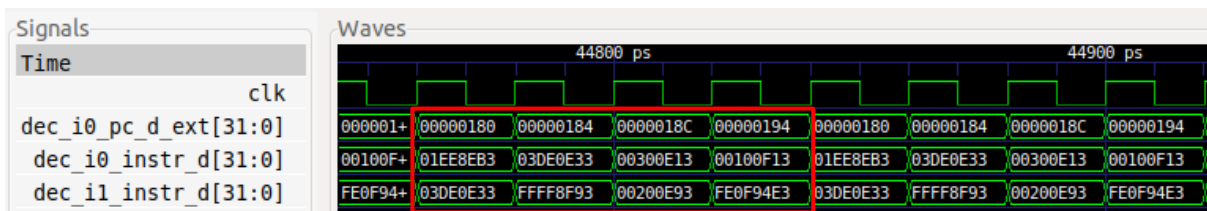
> Executing task: platformio device monitor <

--- 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 = 262392
Instructions = 458787

```

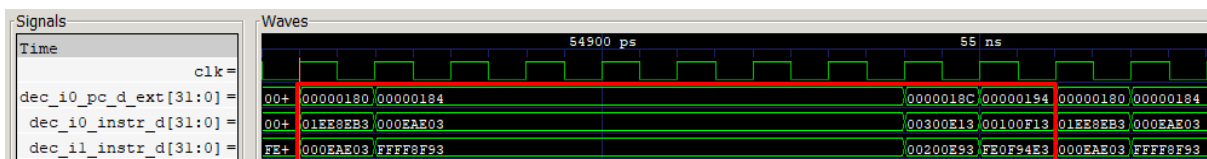
现在的IPC并非理想IPC: $IPC = 458 / 262 = 1.75$

- add指令后跟mul指令：



像前文一样，相关mul指令必须暂停1个周期，原因在于其输入操作数之一在第一条add指令于EX1阶段执行并转发结果之前不可用。

- add指令后跟lw指令：



像前文一样，相关lw指令必须暂停1个周期，原因在于其输入操作数之一在第一条add指令于EX1阶段执行并转发结果之前不可用。

任务：将前面的公式与DDCARV中流水线处理器部分所述的公式进行比较。

DDCARV中流水线处理器的公式：

```

if      ((Rs1E == RdM) & RegWriteM) & (Rs1E != 0) then // Forward from Memory stage
    ForwardAE = 10
else if ((Rs1E == RdW) & RegWriteW) & (Rs1E != 0) then // Forward from Writeback stage
    ForwardAE = 01
else      ForwardAE = 00                                // No forwarding (use RF output)
    
```

任务：分析Verilog代码，说明前一个公式如何执行计算。必须检查模块dec_decode_ctl的以下行。

不提供解答。

任务：写出i0_rs2bypass[9:0]、i0_rs1bypass[9:0]、i1_rs2bypass[9:0]和i1_rs1bypass[9:0]的其他控制位的公式（与上述公式类似）。

可以从模块dec_decode_ctl中的以下几行获得公式：

- 2372 – 2417
- 1721 – 1767
- 1497 – 1544

- 1130 – 1131和1255 – 1256

任务： 在自己的计算机上重复图8中的仿真过程。可以使用以下位置提供的.tcl文件：
`[RVfpgaPath]/RVfpga/Labs/Lab15/DataHazards_AL-AL/test_Advanced.tcl`。

解答请参见实验15的主文档。

任务： 对于图2中的程序，针对两条相互依赖的指令彼此之间距离不同的情况执行与图8中相同的分析。可以通过更改两条相关add指令之间的nop数量来控制距离。

此外，需创建第一个输入操作数接收转发数据的其他示例。

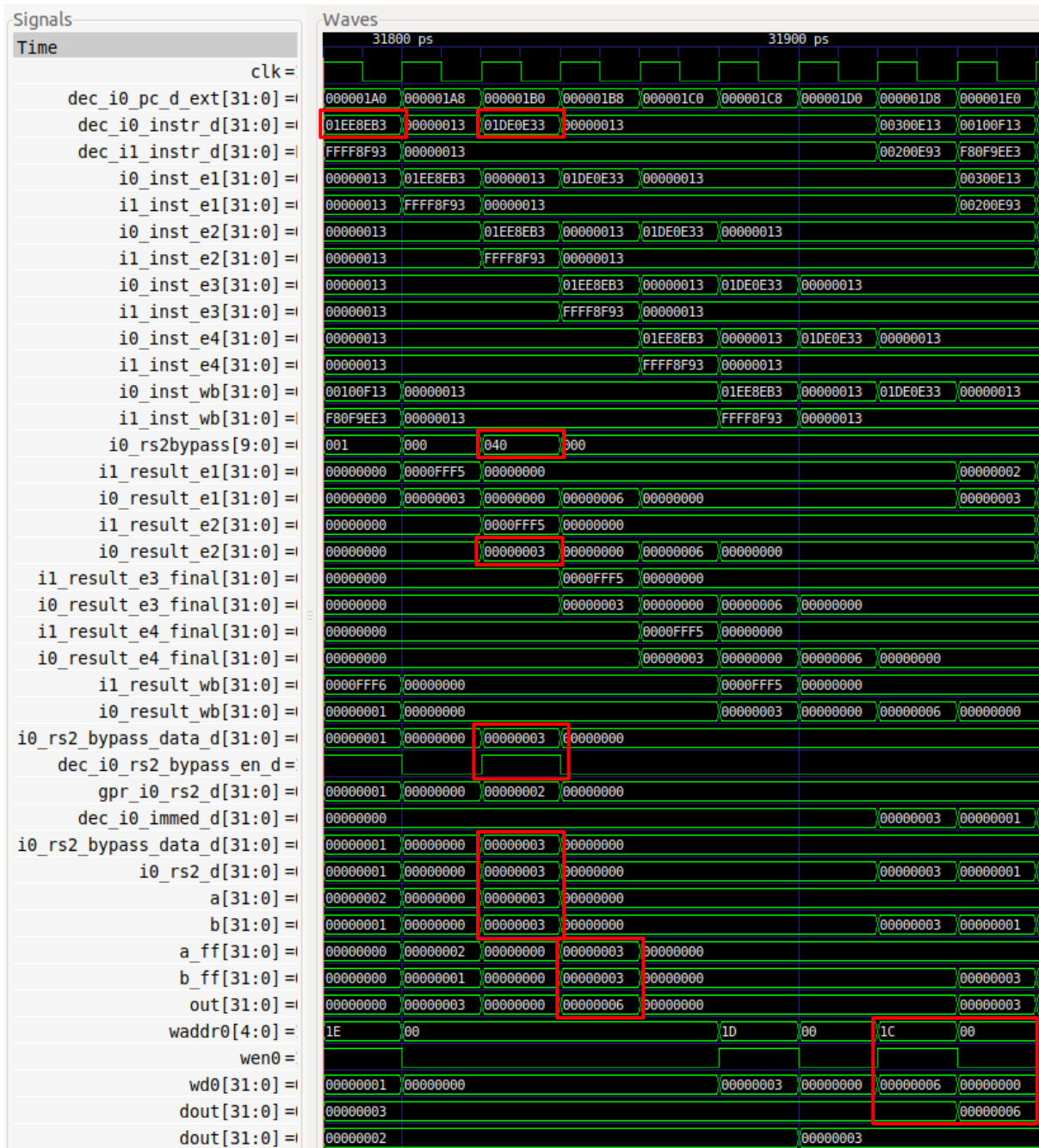
还可以创建两条add指令通过I1管道执行的其他示例，但需确认行为相同。

最后，将相关add指令（`add t3,t3,t4`）替换为通过其他管道执行的其他相关指令并分析仿真结果。例如，可以包含以下指令之一来代替第二条add指令：

- `lw t3, (t4)`（强制读取值来自DCCM，如实验13所述）
- `mul t3, t3, t4`
- `div t3, t3, t4`

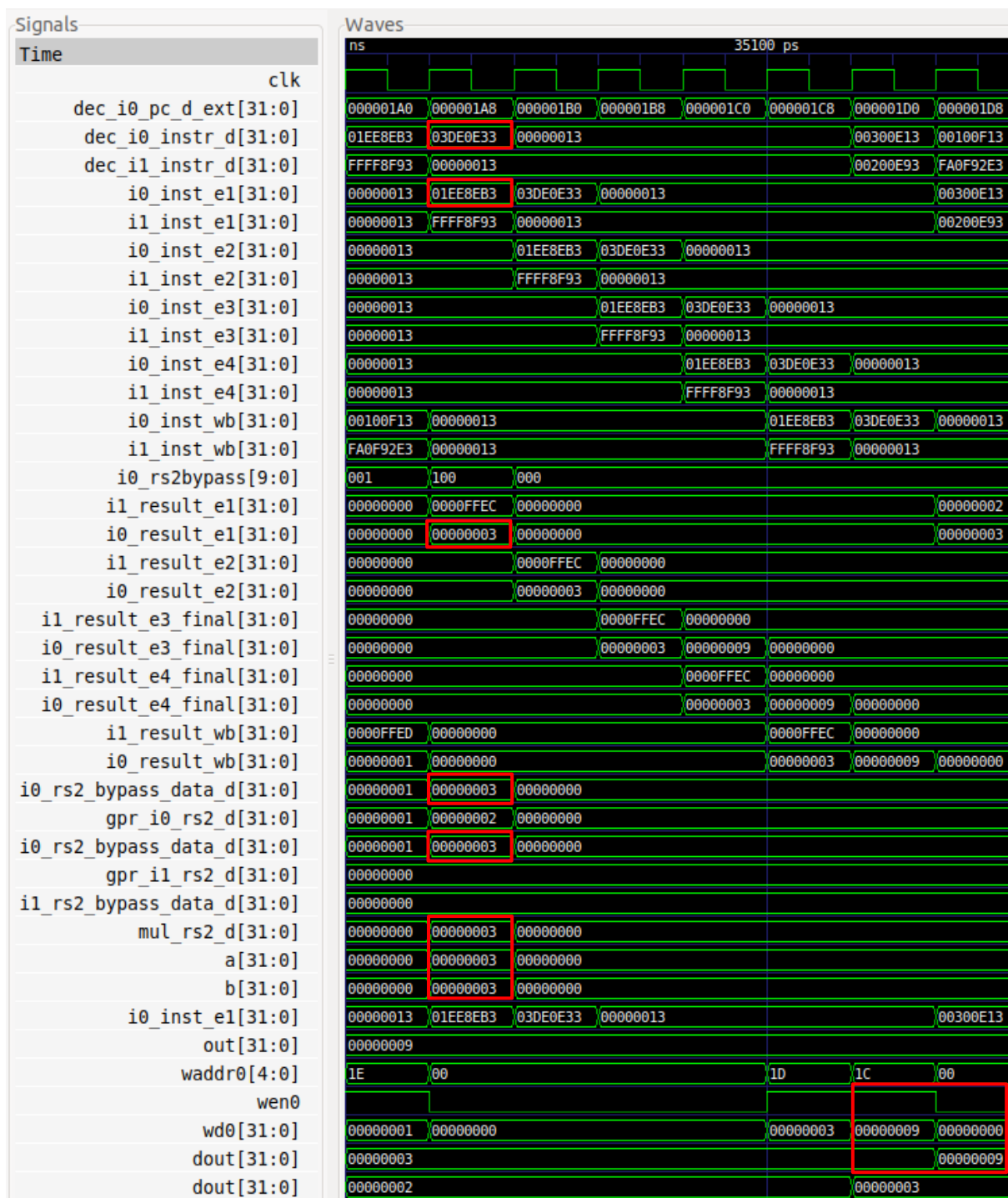
新仿真程序的示例：从EX2阶段旁路到译码阶段：

<code>1a0: 01ee8eb3</code>	<code>add t4,t4,t5</code>
<code>1a4: ffff8f93</code>	<code>addi t6,t6,-1</code>
<code>1a8: 00000013</code>	<code>nop</code>
<code>1ac: 00000013</code>	<code>nop</code>
<code>1b0: 01de0e33</code>	<code>add t3,t3,t4</code>
<code>1b4: 00000013</code>	<code>nop</code>



新仿真程序的示例：执行mul指令而不是第二条add指令：

```
1a0: 01ee8eb3      add    t4, t4, t5
1a4: ffff8f93      addi   t6, t6, -1
1a8: 03de0e33      mul    t3, t3, t4
```



任务： 向图10添加相应逻辑以生成I0管道中的辅助ALU的第一个输入操作数（a）。

不提供解答。

任务： 在自己的计算机上重复图12中的仿真过程。可以使用以下位置提供的.tcl文件：
[RVfpgaPath]/RVfpga/Labs/Lab15/DataHazards_Close-LW-AL/scriptLoad.tcl

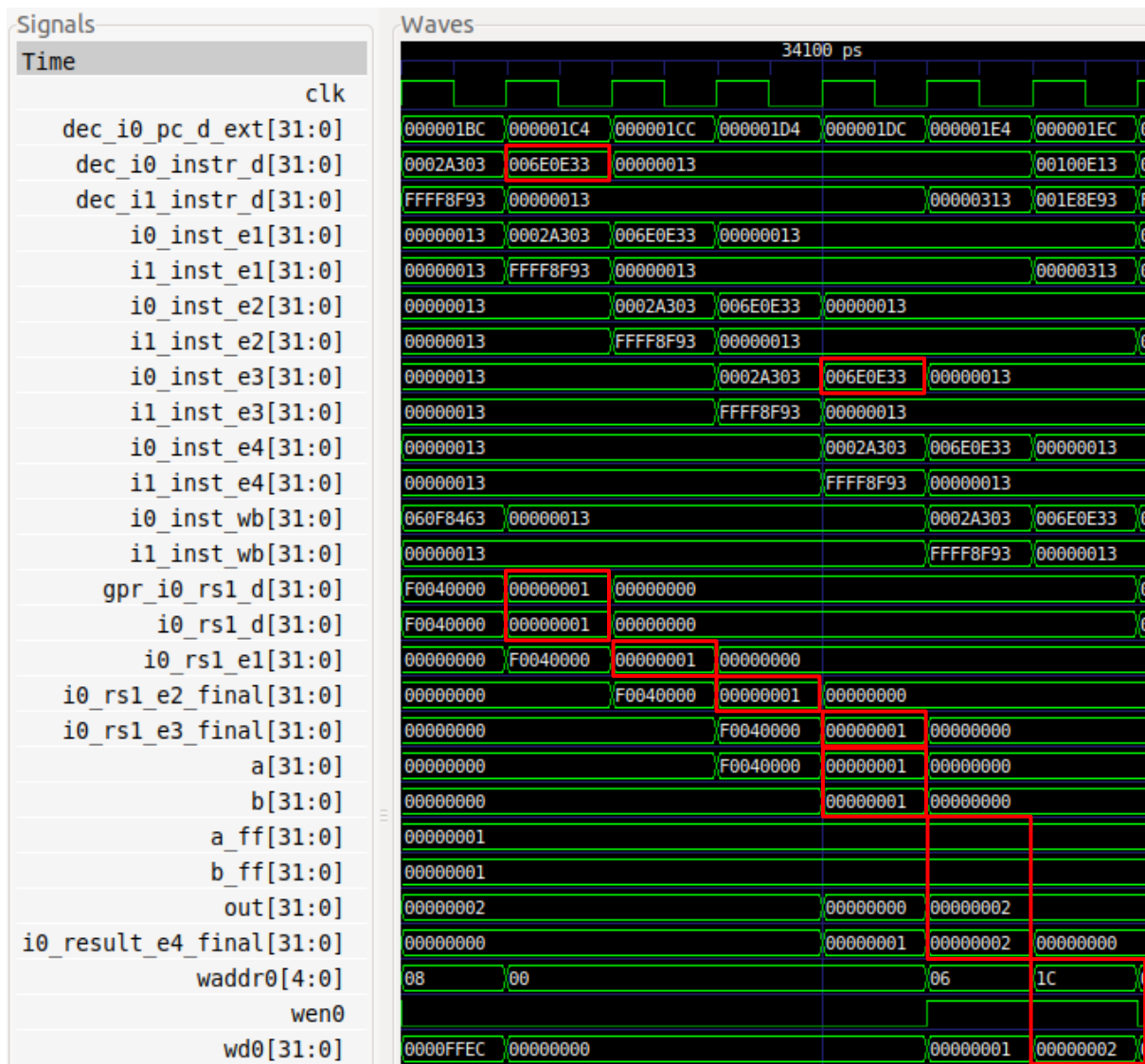
解答请参见实验15的主文档。

任务：为图11的示例绘制与图3类似的图。

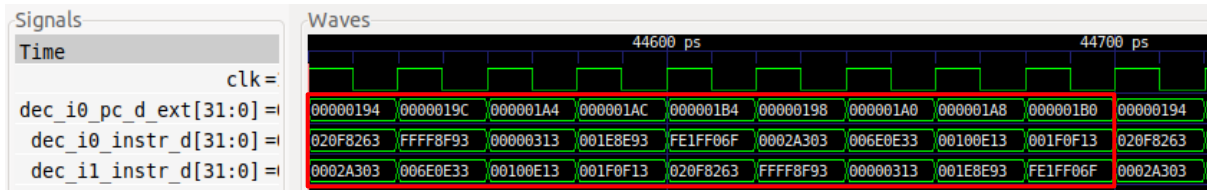
不提供解答。

任务：在前一个示例中，分析如何获取add t3, t3, t1指令的第一个操作数（t3）。可以使用以下位置提供的.tcl文件：*[RVfpgaPath]/RVfpga/Labs/Lab15/DataHazards_Close-LW-AL/scriptLoad_FirstOperand.tcl*

第一个操作数与先前指令不相关，因此它直接从寄存器文件中获取。



任务：删除图11的示例中的nop指令并使用硬件计数器获取IPC。



9个周期内执行2次迭代。每次迭代包含9条指令。因此：IPC = 18 / 9 = 2

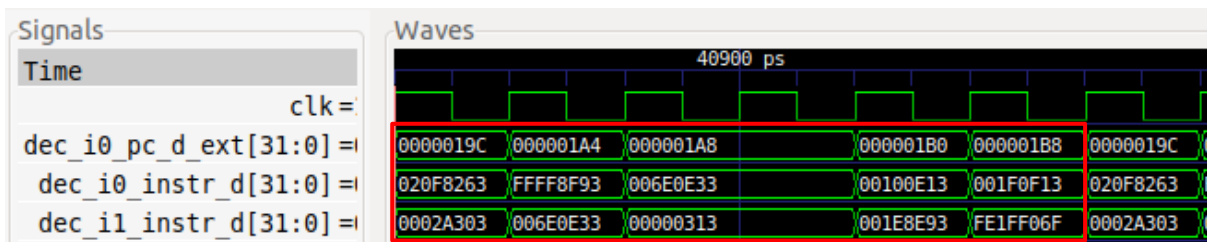
```

src > C Test.c x Test_Assembly.S platformio.ini startup.S
src > C Test.c > main(void)
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 = 295144
Instructions = 589863

```

由于转发逻辑和辅助ALU的原因，没有暂停，IPC是理想IPC：IPC = 5898 / 2951 = 1.998

任务：禁止实验11所述的辅助ALU，并通过Verilator仿真和板上执行的方式分析图11中的示例。



在1w（0x0002a303）之后，相关add指令（0x006e0e33）暂停一些周期。1次迭代需要6个周期。


```

PIO Home  C Test.c  x  Test_Assembly.S  platformio.ini  Firmware.dis  startup.S
src > C Test.c > main(void)
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 = 393469
Instructions = 589865

```

受lw-add数据冒险引起的暂停影响，IPC现与理想IPC相差很大：IPC = 5898 / 3934 = 1.499

任务：在图11的示例中，将add t6,t6,-1指令移至add t3,t3,t1指令之后，然后重新检查仿真真中以及板上的程序。

可以使用以下位置提供的程序：

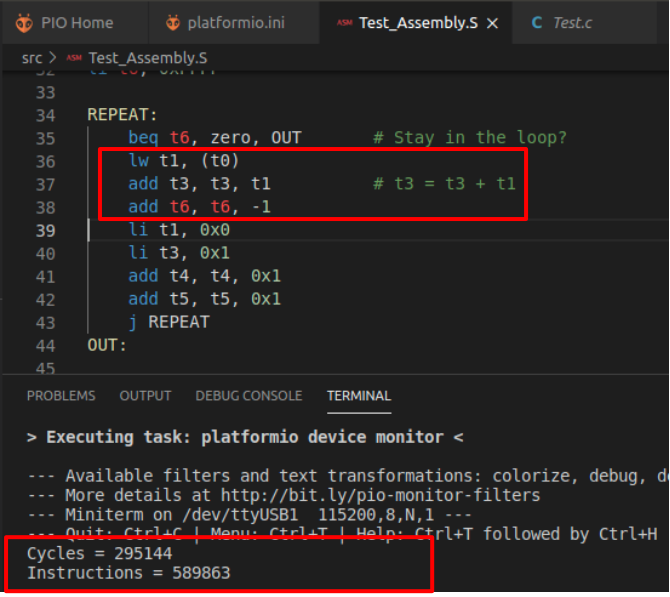
[RVfpgaPath]/RVfpga/Labs/RVfpgaLabsSolutions/Programs_Solutions/Lab15/DataHazards_SameCycle-LW-AL

add t3,t3,t1指令与装载指令相关，二者均通过两个通路并行执行。在这种情况下，add指令会在辅助ALU中重新执行。请注意，通路1可以从其他管道接收输入操作数，因此在这种情况下不会丢失任何周期。

```

1688     assign i0_rs1_bypass_data_e3[31:0] = ({32{e3d.i0rs1bype3[3]}} & i1_result_e4_eff[31:0]) |
1689                                         ({32{e3d.i0rs1bype3[2]}} & i0_result_e4_eff[31:0]) |
1690                                         ({32{e3d.i0rs1bype3[1]}} & i1_result_wb_eff[31:0]) |
1691                                         ({32{e3d.i0rs1bype3[0]}} & i0_result_wb_eff[31:0]);
1692
1693     assign i0_rs2_bypass_data_e3[31:0] = ({32{e3d.i0rs2bype3[3]}} & i1_result_e4_eff[31:0]) |
1694                                         ({32{e3d.i0rs2bype3[2]}} & i0_result_e4_eff[31:0]) |
1695                                         ({32{e3d.i0rs2bype3[1]}} & i1_result_wb_eff[31:0]) |
1696                                         ({32{e3d.i0rs2bype3[0]}} & i0_result_wb_eff[31:0]);
1697
1698     assign i1_rs1_bypass_data_e3[31:0] = ({32{e3d.i1rs1bype3[6]}} & i0_result_e3[31:0]) |
1699                                         ({32{e3d.i1rs1bype3[5]}} & exu_mul_result_e3[31:0]) |
1700                                         ({32{e3d.i1rs1bype3[4]}} & lsu_result_dc3[31:0]) |
1701                                         ({32{e3d.i1rs1bype3[3]}} & i1_result_e4_eff[31:0]) |
1702                                         ({32{e3d.i1rs1bype3[2]}} & i0_result_e4_eff[31:0]) |
1703                                         ({32{e3d.i1rs1bype3[1]}} & i1_result_wb_eff[31:0]) |
1704                                         ({32{e3d.i1rs1bype3[0]}} & i0_result_wb_eff[31:0]);
1705
1706
1707     assign i1_rs2_bypass_data_e3[31:0] = ({32{e3d.i1rs2bype3[6]}} & i0_result_e3[31:0]) |
1708                                         ({32{e3d.i1rs2bype3[5]}} & exu_mul_result_e3[31:0]) |
1709                                         ({32{e3d.i1rs2bype3[4]}} & lsu_result_dc3[31:0]) |
1710                                         ({32{e3d.i1rs2bype3[3]}} & i1_result_e4_eff[31:0]) |
1711                                         ({32{e3d.i1rs2bype3[2]}} & i0_result_e4_eff[31:0]) |
1712                                         ({32{e3d.i1rs2bype3[1]}} & i1_result_wb_eff[31:0]) |
1713                                         ({32{e3d.i1rs2bype3[0]}} & i0_result_wb_eff[31:0]);
1714

```

```
src > Test_Assembly.S
32  t0, 0x1ff
33
34  REPEAT:
35      beq t6, zero, OUT      # Stay in the loop?
36      lw t1, (t0)
37      add t3, t3, t1          # t3 = t3 + t1
38      add t6, t6, -1
39      li t1, 0x0
40      li t3, 0x1
41      add t4, t4, 0x1
42      add t5, t5, 0x1
43      j REPEAT
44  OUT:
45
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 = 295144
Instructions = 589863
```

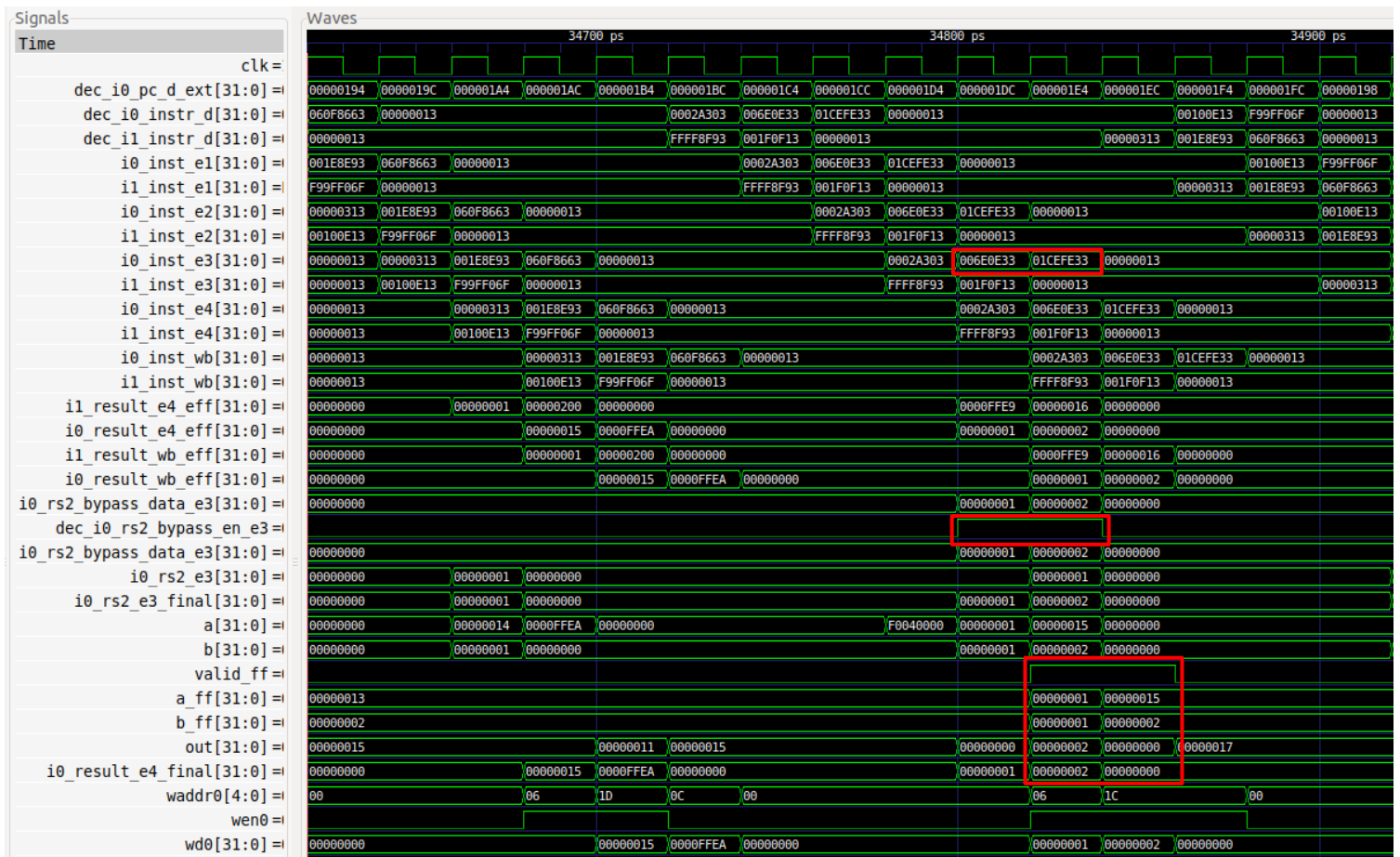
由于转发逻辑和辅助ALU的原因，没有暂停，IPC是理想IPC：IPC = 5898 / 2951 = 1.998

1. 练习

- 1) 通过添加与add指令结果相关的额外算术-逻辑指令，修改第3部分中使用的程序。例如，可以将图11中的循环替换为以下代码，其中包含一条新的AND指令（**and t3, t4, t3**），并通过前移指令**add t5, t5, 0x1**对代码顺序稍作调整：

```
REPEAT:
    beq t6, zero, OUT
    INSERT_NOPS_9
    lw t1, (t0)
    add t6, t6, -1
    add t3, t3, t1
    add t5, t5, 0x1
    and t3, t4, t3
    INSERT_NOPS_8
    li t1, 0x0
    li t3, 0x1
    add t4, t4, 0x1
    j REPEAT
OUT:
```

分析Verilator仿真并说明如何为新的A-L指令处理数据冒险。然后删除所有nop指令并分析硬件计数器提供的结果。



相关add和and指令使用辅助ALU重新计算结果。请注意，and指令的第二个输入操作数在EX3阶段旁路。

```

src > C Test.c > main(void)
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, 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 = 327910
Instructions = 655398

```

$$IPC = 6553 / 3279 = 1.998$$

$$\text{每次迭代执行的指令: } 655398 / 65535 = 10$$

每次迭代的周期数：327910 / 65535 = 5

由于转发和辅助ALU的原因，在该程序中实现了理想IPC。

- 2) 分析与第2.C部分所述情况相同的情况：mul指令后跟使用乘法结果的add指令。在图11的程序中，只需将lw替换为写入寄存器t1的mul。

不提供解答。

- 3) 分析lw指令后跟与装载读取的值相关的mul指令这种情况。在图11的程序中，只需将相关add指令替换为mul指令。

可以使用以下位置提供的程序：

[RVfpgaPath]/RVfpga/Labs/RVfpgaLabsSolutions/Programs_Solutions/Lab15/DataHazards_Close-LW-MUL

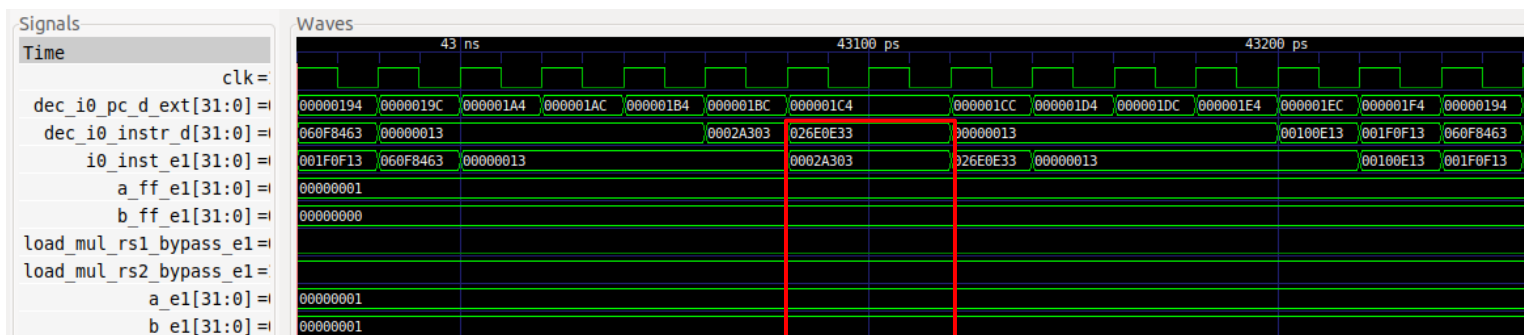
辅助ALU无法执行mul指令。乘法器（模块exu_mul_ctl）内部实现了一个新的旁路路径，用于将装载读取的值转发到M1阶段。

```

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];

```

这样一来，由于这种RAW相关性，只有1个周期丢失。



第二个操作数从装载读取的值旁路。这样一来，由于相关性的原因，只有1个周期丢失。

- 4) （以下练习基于[HePa]的练习4.18、4.19、4.20和4.26。）
 假设在不处理数据冒险的SweRV EH1处理器版本上执行了以下代码（即，编程器负责通过在必要时插入nop指令来处理数据冒险）。向代码中添加nop指令以使其正确运行。
- ```

 addi x11, x12, 5
 add x13, x11, x12

```



```
addi x14, x11, 15
add x15, x13, x12
```

然后组成包含至少三个汇编代码片段的序列，这些片段显示不同类型的RAW数据冒险。RAW数据相关性的类型由产生结果的阶段和使用结果的下一条指令标识。

对于每个序列，要使代码在没有转发或冒险检测的SweRV EH1处理器上正确运行，需要在何处插入多少条nop？如果我们使用SweRV EH1中提供的转发而不插入nop，那么CPI是多少？

不提供解答。

5) 在实验14第2.C部分的程序（位于 `[RVfpgaPath]/RVfpga/Labs/Lab14/LW_Instruction_ExtMemory`）中，将指令add **x1**, x1, 1替换为add **x28**, x1, 1。这将在修改后的add指令和循环开头的非阻塞装载（lw x28, (x29)）之间引入WAW冒险。利用仿真分析SweRV EH1中如何处理此冒险，可以在寄存器文件中查看信号wen2的值。尝试了解控制单元（模块dec）中如何计算此信号。

原始程序的仿真如下。正如我们在实验14中分析的那样，当对寄存器文件同时执行3个写操作（2条add指令和1条非阻塞load指令）。



在如下所示的新程序中（将指令add **x1**, x1, 1替换为指令add **x28**, x1, 1），可以检测到程序顺序中的后面一条指令修改了同一个寄存器，因此禁止了装载的写操作（wen2信号永远不会变为高电平），这样便可解除WAW数据冒险。

The figure displays a timing diagram with two main sections: 'Signals' on the left and 'Waves' on the right. The 'Signals' section lists various digital signals, including control signals like `dec_i0_pc_d_ext[31:0]`, `dec_i0_instr_d[31:0]`, `dec_i1_instr_d[31:0]`, `lsu_rsl_d[31:0]`, `lsu_offset_d[11:0]`, `rsl_dcl[31:0]`, `offset_dcl[11:0]`, `full_addr_dcl[31:0]`, `lsu_axi_arvalid`, `lsu_axi_araddr[31:0]`, `lsu_axi_rvalid`, `lsu_axi_rdata[63:0]`, `dec_nonblock_load_waddr[4:0]`, `dec_nonblock_load_wen`, `lsu_nonblock_load_data[31:0]`, `i0_depend_load_e1_d`, `i0_depend_load_e2_d`, `i1_depend_load_e1_d`, `i1_depend_load_e2_d`, `waddr0[4:0]`, `wen0`, `wd0[31:0]`, `waddr1[4:0]`, `wen1`, `wd1[31:0]`, `waddr2[4:0]`, `wen2`, and `wd2[31:0]`. The 'Waves' section shows the timing of these signals. A clock signal (`clk`) is shown at the top. Below it, various data signals are shown. A red box highlights a specific data signal (`lsu_nonblock_load_data[31:0]`) at approximately 4500 ps.

15



不提供解答。

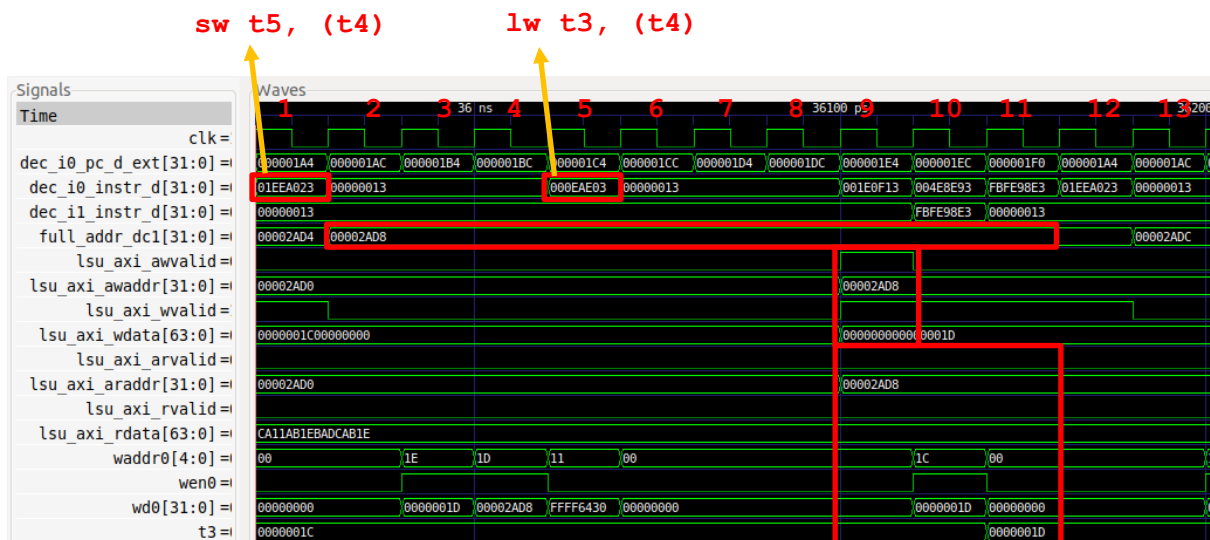
## 8) 存储-装载转发

这是一种非常值得关注的情况，我们没有在本实验中分析，但您可在本练习中进行分析。当存储以及随后的装载访问相同地址时，可以在内核内将数据从存储转发到装载，无需读取DDR外部存储器，从而节省时间和功耗。

实现这种转发的逻辑包含在LSU中，具体来说是在模块lsu\_bus\_intf和lsu\_bus\_buffer中，必须在本练习中进行检查。

[RVfpgaPath]/RVfpga/Labs/Lab15/Sw-Lw-Forwarding中的PlatformIO项目用于说明存储-装载转发。此文件夹中提供.tcl脚本，您可以使用它来分析循环的任意一次迭代以及了解如何执行转发。

## Verilator仿真：



## 分析仿真：

- **周期1：** sw指令进行译码。
- **周期5：** lw指令进行译码。
- **周期2至11：** 在整个迭代过程中，信号full\_addr\_dc1 = 0x00002AD8。发生这种情况是因为存储地址和装载地址相同。
- **周期9：** 存储通过AXI总线的写信号写入DDR外部存储器。

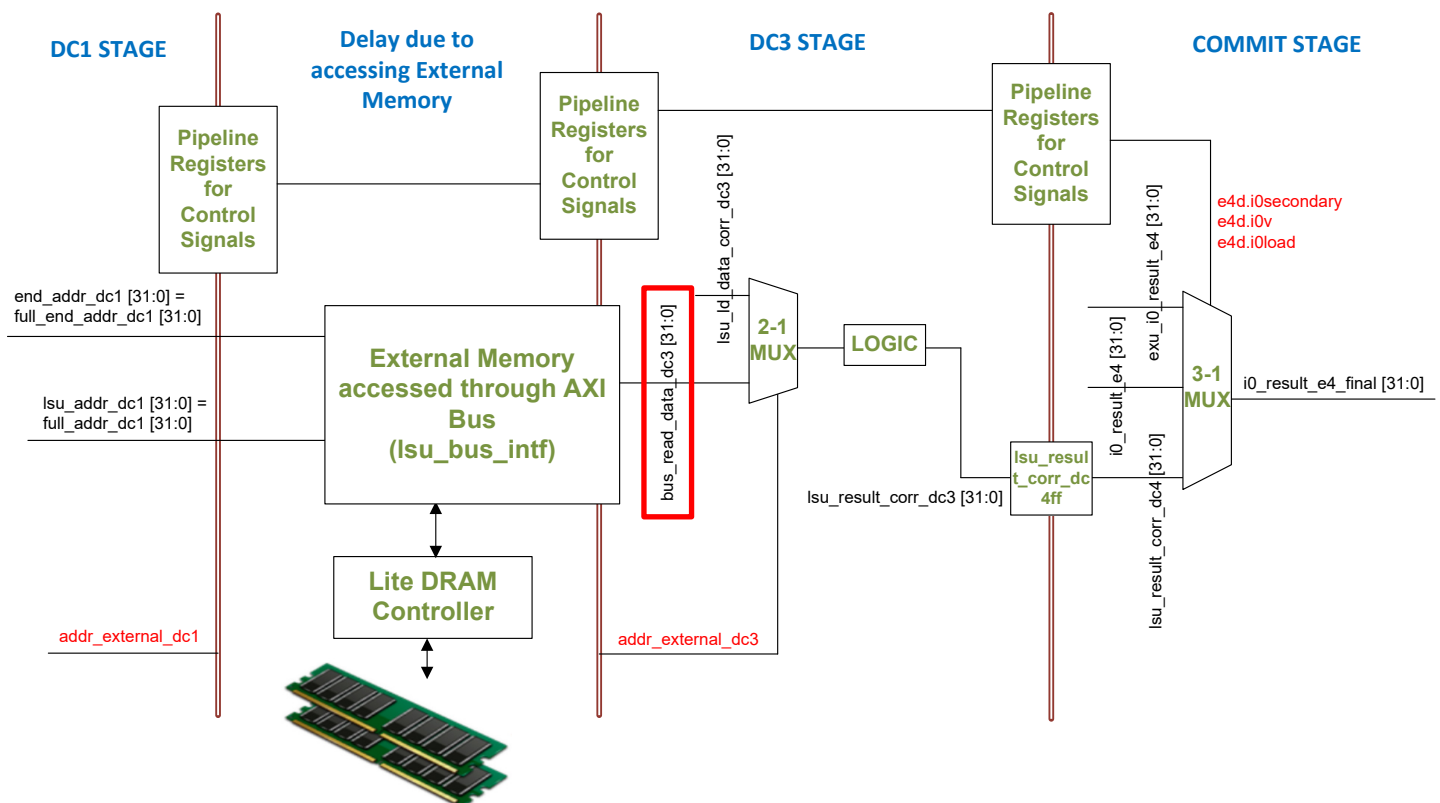
- o `lsu_axi_awvalid = 1`
- o `lsu_axi_awaddr = 0x00002AD8`
- o `lsu_axi_wvalid = 1`
- o `lsu_axi_wdata = 0x000000000000001D`

- **周期9、10和11:** 装载通过旁路逻辑立即接收其数据并将数据写入寄存器文件。读取永远不会发送到DDR存储器（参见AXI总线的读使能信号：`lsu_axi_arvalid = lsu_axi_rvalid = 1`）：
  - o `waddr0 = 0x1C`（即寄存器x28 = t3）
  - o `wen0 = 1`
  - o `wd0 = 0x0000001D`
  - o `t3 = 0x0000001D`

## 内核内部如何执行转发？

要分析存储-装载转发，必须检查两个模块：`lsu_bus_intf`和`lsu_bus_buffer`。

- 1) 我们在实验13的第4部分中分析了对DDR外部存储器的读访问，在实验13的图16中说明了此访问涉及的SweRV EH1结构：



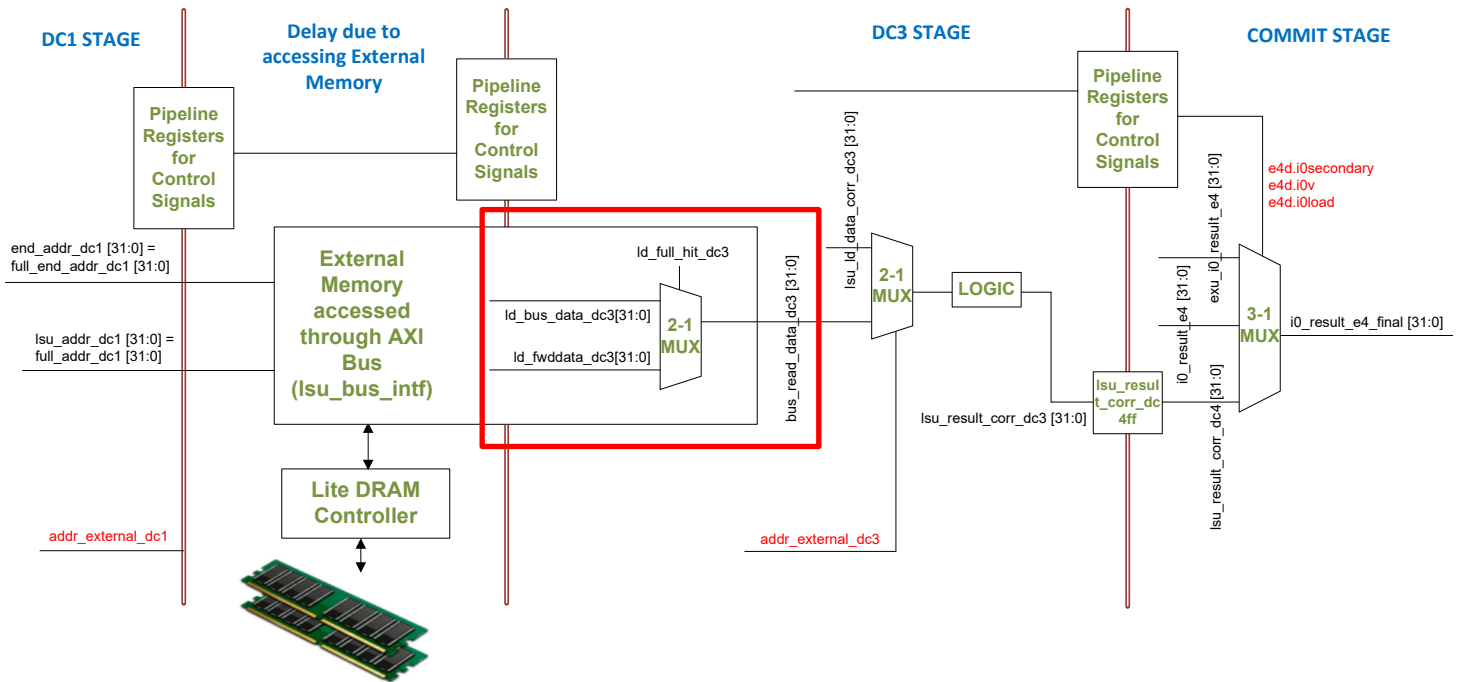
数据在信号`bus_read_data_dc3`中提供。

- 2) 分配给`bus_read_data_dc3`的值可以来自DDR存储器或转发逻辑。为此，模块`lsu_bus_intf`中包含一个2:1多路开关，用于在从DDR存储器读取的值

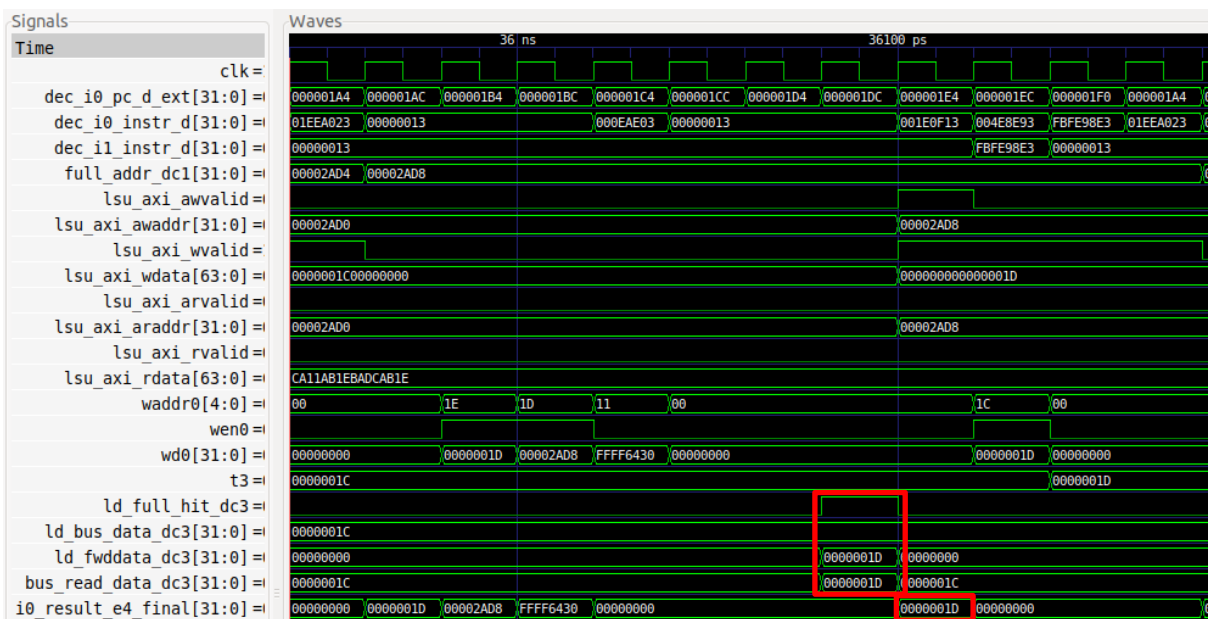
(ld\_bus\_data\_dc3) 和旁路值 (ld\_fwddata\_dc3) 之间进行选择。

```
387 assign bus_read_data_dc3[31:0] = ld_full_hit_dc3 ? ld_fwddata_dc3[31:0] : ld_bus_data_dc3[31:0];
```

接下来，我们将此2:1多路开关包含到上图中：



3) 如果在之前的仿真中添加此多路开关的输入/输出/控制信号，则：



可以看到提供给装载的数据从存储旁路。

- 4) 可以进一步分析模块`lsu_bus_intf`和`lsu_bus_buffer`中如何计算控制信号 (`ld_full_hit_dc3`) 和旁路值 (`ld_fwddata_dc3`)。

## 附录A

**任务：** 在自己的计算机上重复图15中的仿真过程。

解答请参见实验15的主文档。

**任务：** 比较上述场景在SweRV EH1和DDCARV所述流水线处理器中的处理方式。

不提供解答。

**任务：** 如果仔细比较图16和实验13的图6，将看到实验13的图6中lw指令读入寄存器文件的值（信号`lsu_ld_data_corr_dc3[31:0]`）与图16中lw转发的值（信号`lsu_ld_data_dc3[31:0]`）不同。两个值的区别在于前者经过模块`lsu_ecc`中的ECC逻辑校验，而后者没有。说明为什么lw转发的值未经过错误校验也没有问题。

如果在装载读取的数据中检测到错误，则流水线将停止并刷新。因此，装载指令和所有后续指令（其中一些指令接收不正确的转发值）均刷新，永不提交。

**任务：** 在图14的示例中，删除lw之前和add之后的所有nop指令。不要删除两条相关指令之间的5个nop。分析仿真，然后通过开发板上执行程序，用性能计数器计算IPC（在测量IPC时保留nop指令似乎并不合适，因为它们是无用指令；不过，程序本身就是无用的，这里我们的惟一目的是分析并了解数据冒险）。

不提供解答。