

TASK: Remove all `nop` instructions in the example from **Error! Reference source not found.** Generate the trace with the RVfpga-Trace simulator, analyse the simulation on RVfpga-Pipeline, and then compute the IPC by using the Performance Counters while executing the program on the board (remember that you must uncomment all instructions in the main function, in file *Test.c*).

<code>clk=</code>							
<code>dec_i0_pc_d_ext[31:0] =</code>	00000424	00000428	0000042C	00000430	00000434	00000438	0000043C
<code>dec_i0_instr_d[31:0] =</code>	01EE8EB3	01DE0E33	FFFF8F93	00300E13	00200E93	00100F13	FE0F94E3
<code>dec_i0_rs2_bypass_en_d[3:0] =</code>	1	4	0				1
<code>dec_i0_result_r[31:0] =</code>	00000001	00000003	00000006	0000FFF0	00000003	00000002	00000001
<code>lsu_result_m[31:0] =</code>	00000000						
<code>exu_i0_result_x[31:0] =</code>	0000FFF1	00000003	00000006	0000FFF0	00000003	00000002	00000001
<code>lsu_nonblock_load_data[31:0] =</code>	00000000						
<code>i0_rs2_bypass_data_d[31:0] =</code>	00000001	00000003	00000000				00000001
<code>i0_rs1_d[31:0] =</code>	00000002	00000003	0000FFF1	00000000		0000FFF0	00000002
<code>i0_rs2_d[31:0] =</code>	00000001	00000003	FFFFFFFF	00000003	00000002	00000001	00000000
<code>result[31:0] =</code>	00000003	00000006	0000FFF0	00000003	00000002	00000001	0000FFF0
<code>i0_inst_x[31:0] =</code>	FE0F94E3	01EE8EB3	01DE0E33	FFFF8F93	00300E13	00200E93	00100F13
<code>exu_i0_result_x[31:0] =</code>	0000FFF1	00000003	00000006	0000FFF0	00000003	00000002	00000001
<code>i0_inst_r[31:0] =</code>	00100F13	FE0F94E3	01EE8EB3	01DE0E33	FFFF8F93	00300E13	00200E93
<code>wen0 =</code>							
<code>waddr0[4:0] =</code>	1E	09	1D	1C	1F	1C	1D
<code>wd0[31:0] =</code>	00000001	00000003	00000006	0000FFF0	00000003	00000002	00000001

```

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  SERIAL TERMINAL

Connected success !

Cycles = 460767
Instructions = 459360
BrCom = 65623
BrMis = 22

```

- The IPC = 1, thanks to the forwarding logic that allows the dependent instruction to not stall.
- The number of cycles is as expected $0xffff * 7 = 458745$

Exercise 1: In the example from **Error! Reference source not found.**, analyse and explain similar situations where you replace the dependent `add` instruction for other dependent instructions, such as:

- `add t4, t4, t5`
`mul t3, t3, t4`
- `add t4, t4, t5`
`div t3, t3, t4`
- `add t4, t4, t5`
`lw t3, 0(t4)`

Solution not provided for this exercise.

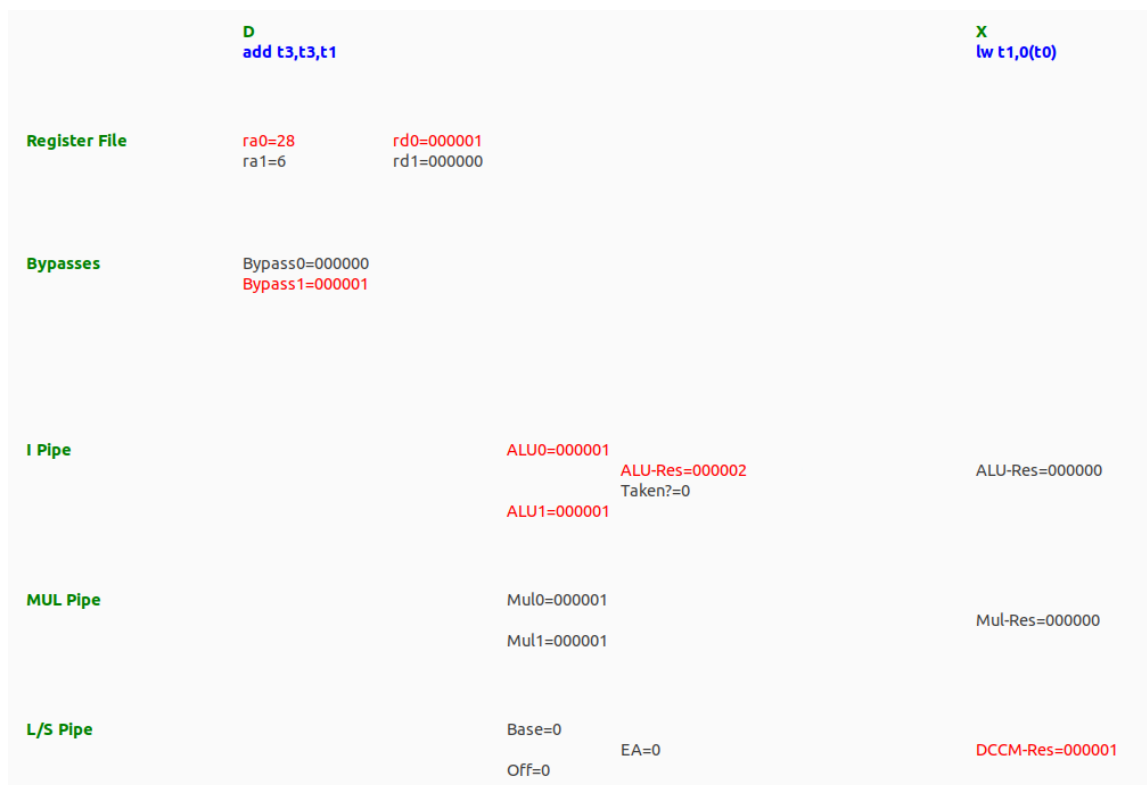
Exercise 2: Use the project called *DataHazards_LW-AL* to analyse a hazard between a `load` and an `add` instruction. Analyse the following two situations:

- **DCCM**: The load reads from the DCCM in a single cycle. You can use the tcl script called *test_DCCM.tcl*. For mapping the data to the DCCM, uncomment, in file *Test_Assembly.S*, line: `.section .midccm`, and comment line: `.section .ram`

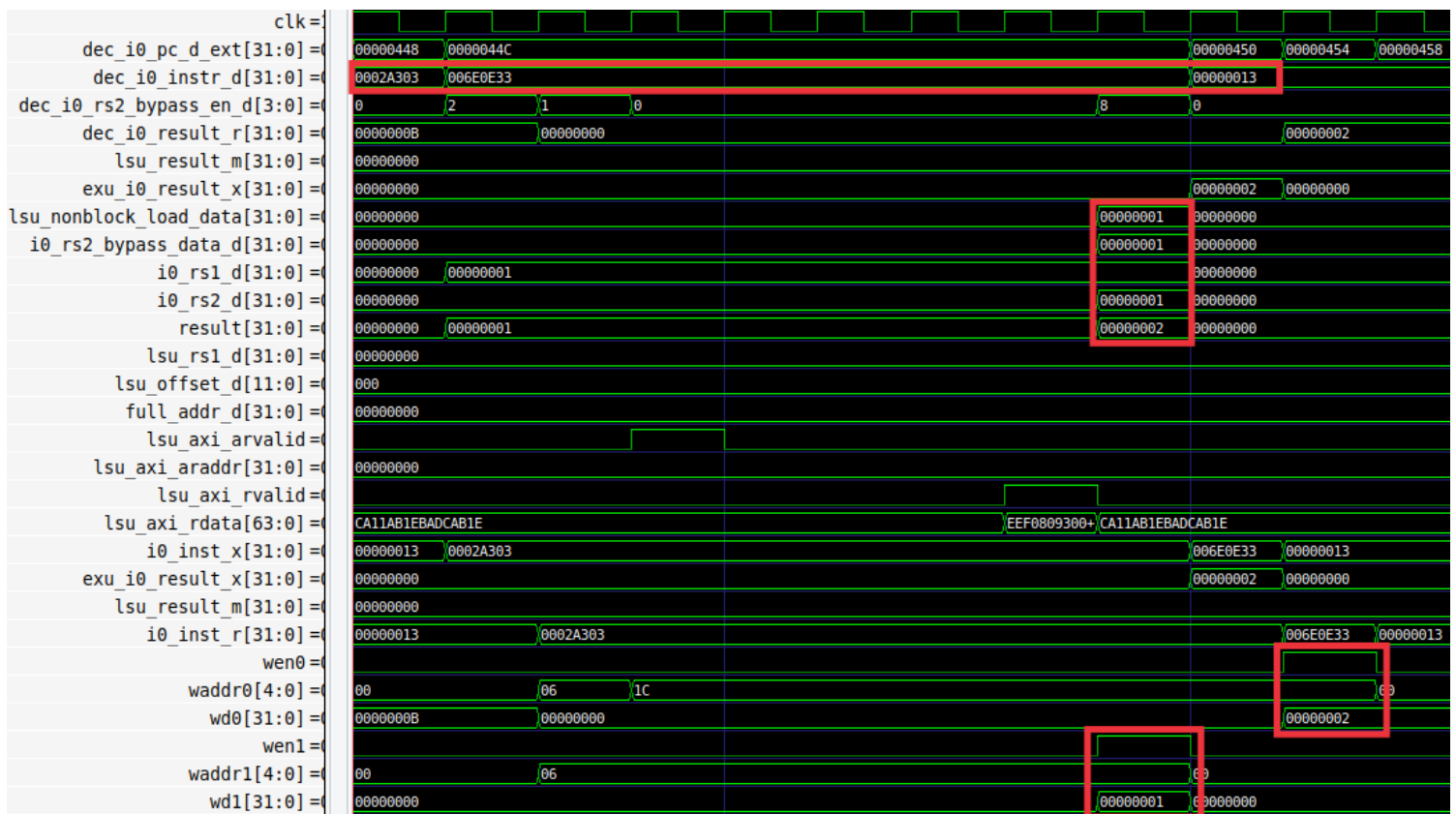
- **Main Memory:** The load reads from Main Memory in several cycles. You can use the tcl script called *test_MainMemory.tcl*. For mapping the data to Main Memory, uncomment, in file *Test_Assembly.S*, line: `.section .ram`, and comment line: `.section .midccm`

Simulate the program both in RVfpga-Trace and RVfpga-Pipeline.

[illegible]



- The DCCM provides the data in one cycle.
- That data (0x1) is forwarded to the add instruction, which uses it as its second operand instead of the data read from the Register File, which is incorrect.
- There are no stalls thanks to the DCCM low-latency.



	D	X	R
	add t3,t3,t1		
Register File	ra0=28 ra1=6	rd0=000001 rd1=000000	wa0=28 we0=0 wd0=0 wa1=6 we1=1 wd1=1 wa2=0 we2=0 wd2=0
Bypasses	Bypass0=000000 Bypass1=000001		
I Pipe	ALU0=000001 ALU1=000001	ALU-Res=000002 Taken?=0	ALU-Res=000000
MUL Pipe	Mul0=000001 Mul1=000001		Mul-Res=000000
L/S Pipe	Base=0 Off=0	EA=0	DCCM-Res=000000

- The Main Memory needs several cycles for providing the result, due to the latency of the memory itself and the AXI bus protocol.
- When data is available, it is written to the RF and provided to the add instruction through forwarding, so that it can start right away.