

TASK: Measure different events (cycles, instructions/loads committed, etc.) using the Performance Counters available in VeeR EL2, as explained in Lab 11. Remember that you must uncomment the code that configures and uses the Performance Counters. Is the number of cycles as expected after analysing the simulation from **Error! Reference source not found.**? Justify your answer.

EXECUTION ON RVfpgaEL2-Nexys (physical board):

```

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  SERIAL TERMINAL

Connected success !
Cycles = 525196
Instructions = 524581
BrCom = 65563
BrMis = 15
  
```

EXECUTION ON RVfpgaEL2-ViDBo (virtual board):

```

Cycles = 525205

Instructions = 524581

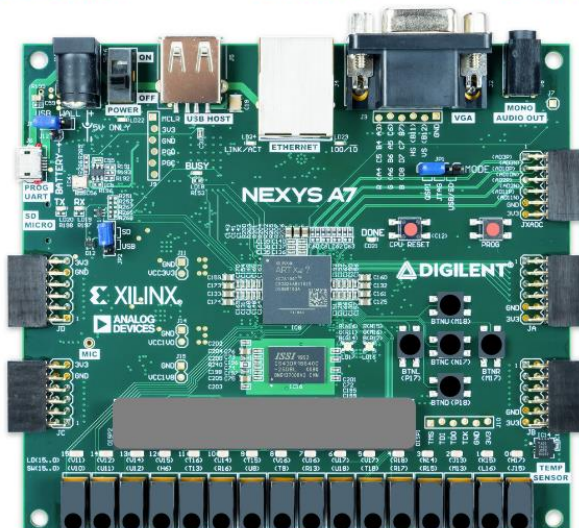
BrCom = 65563

BrMis = 15
  
```

Disconnect Clear UART output

7 SEGMENT DISPLAYS:

B value: G value: R value: Color:



EXPLANATION:

- IPC=1, which is reasonable as there are no stalls.
- The loop contains 8 instructions and the number of iterations is 65563. Thus, in theory it takes $65563 \times 8 = 524504$. The number of cycles obtained is almost the same.

TASK: Measure different events (cycles, instructions/loads committed, etc.) using the Performance Counters available in VeeR EL2, as explained in Lab 11. Remember that you must uncomment the code that configures and uses the Performance Counters. Is the number of cycles as expected after analysing the simulation from **Error! Reference source not found.**? Justify your answer.

EXECUTION ON RVfpgaEL2-Nexys (physical board):

```

PROBLEMS  OUTPUT  DEB

Connected success !
Cycles = 1115254
Instructions = 655654
BrCom = 65563
BrMis = 15

```

EXECUTION ON RVfpgaEL2-ViDBo (virtual board):

```

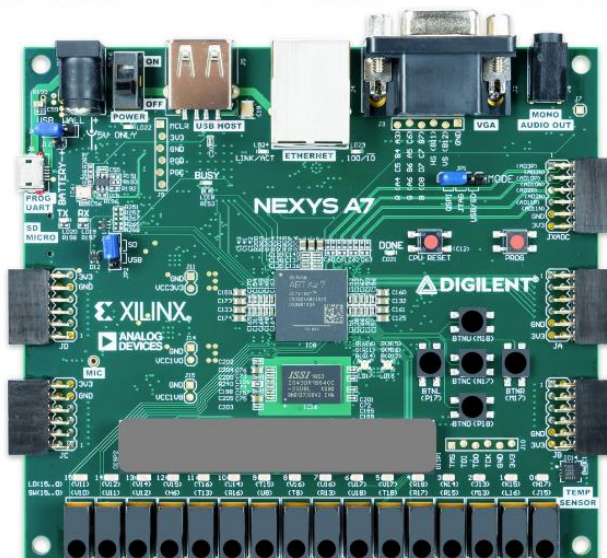
Cycles = 1115006
Instructions = 655654
BrCom = 65563
BrMis = 16

```

Disconnect Clear UART output

7 SEGMENT DISPLAYS:

B value: G value: R value: Color:



EXPLANATION:

- IPC is around 0.5. The poor performance is caused by the stalls introduced due to the structural hazard between the two `div` instructions.

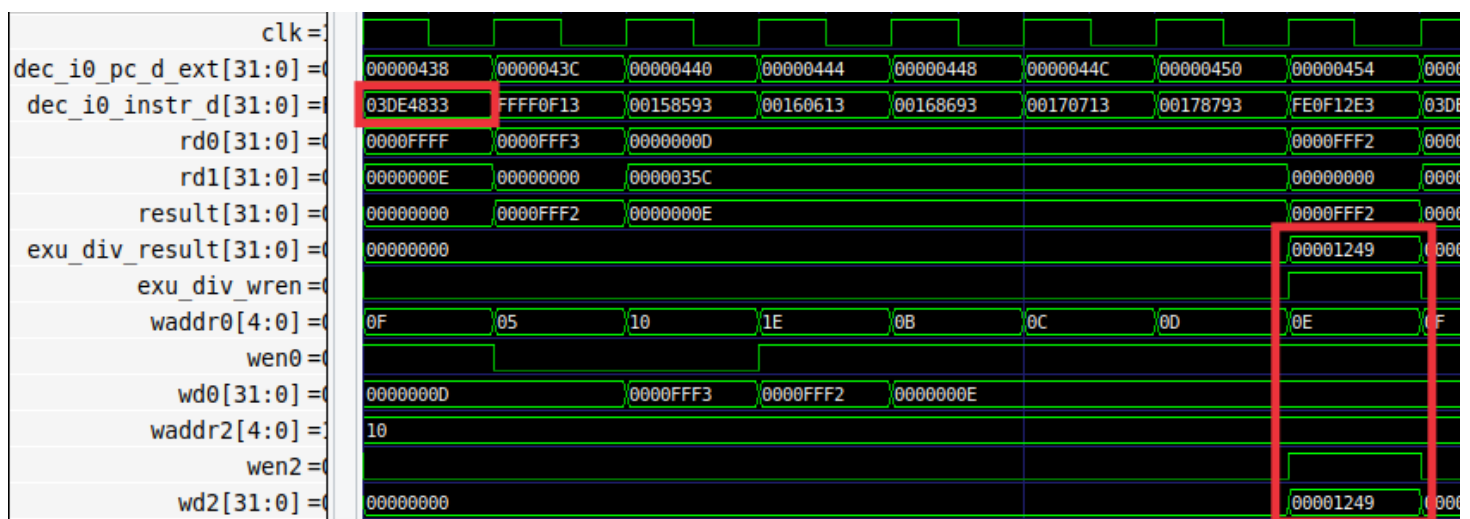
1) Like loads, `div` instructions are non-blocking, thus independent instructions can continue executing while the `div` is being computed in the divisor. Also like in the case of loads, it can happen that when the division finishes and progresses to the R Stage, another instruction is also at this stage and needs to write its result to the Register File. The solution is again the same as the one used for `load` instructions: a third write port (port 2, see Lab 11) is included in the Register File so that the two writes can happen in the same cycle. Illustrate this situation simulating the program provided in folder *[RVfpgaEL2NexysA7NoDDRPath]/Labs/Lab14/Div_Instruction*.

The example below illustrates this situation. It executes a `div` instruction followed by several `add` instructions contained within a loop that repeats for 0xFFFF iterations (i.e. 65,535). The `div` instruction is highlighted in red. The `add` instruction that arrives at the Writeback Stage in the same cycle as the `div` instruction is also highlighted. As usual, the program does nothing useful and is only intended to illustrate the example of this lab.

```

REPEAT:
    div a6, x28, x29
    add x30, x30, -1
    add a1, a1, 1
    add a2, a2, 1
    add a3, a3, 1
    add a4, a4, 1
    add a5, a5, 1
    bne x30, zero, REPEAT # Repeat the loop
  
```

The following figure shows the RVfpgaEL2-Trace simulation for the previous example program for a random iteration of the loop.



In the final cycle, the `div` instruction and the conflicting `add` instruction arrive at the R Stage, where they must write the register file. This is possible thanks to the three write ports available in VeeR EL2's register file.

Imagination
university programme

- 2) Create a program, similar to the one from Section 2.B, where two sequential independent load instructions are executed. How is this scenario handled in VeeR EL2? Is it equal to or different from the solution used for the two sequential divisions.

The program is provided at [Labs/RVfpgaLabsSolutions/Lab14/Lw_Sequential_Instructions](#). This is the code:

```
.data
D: .word 11, 10, 9, 8, 7, 6

.text
Test_Assembly:

la x29, D

li x30, 0xFFFF

add a1, zero, 1
add a2, zero, 1
add a3, zero, 1
add a4, zero, 1
add a5, zero, 1

REPEAT:
    lw x28, (x29)
    lw x31, 20(x29)
    add x30, x30, -1
    add a1, a1, 1
    add a2, a2, 1
    add a3, a3, 1
    add a4, a4, 1
    add a5, a5, 1
    bne x30, zero, REPEAT        # Repeat the loop
```

clk=												
dec_i0_pc_d_ext[31:0]=	00000430	00000434	00000438	0000043C	00000440	00000444	00000448	0000044C	00000450	00000430	00000434	0
dec_i0_instr_d[31:0]=	000EAE03	014EAF83	FFFF0F13	00158593	00160613	00168693	00170713	00178793	FE0F10E3	000EAE03	014EAF83	F
lsu_axi_arvalid=												
lsu_axi_araddr[31:0]=	00003368		00003350	00003368								
lsu_axi_rvalid=												
lsu_axi_rdata[63:0]=	0000000000000+	CALLAB1EBADCAB1E						0000000B00000+	CALLAB1EBADCA+	00000000000000+	CALLAB1EBADCAB1E	
waddr0[4:0]=	0F	01	1C	1F	1E	0B	0C	0D	0E	0F	01	1
wen0=												
wd0[31:0]=	0000001A		00000000		0000FFE5	0000001B						0
waddr1[4:0]=	00	1F	1C						00		1F	1
wen1=												
wd1[31:0]=	00000000	00000006	00000000						0000000B	00000000	00000006	0

The AXI bus allows requests to be pipelined, thus no stalls occur in this case.

- 3) Analyse a scenario where three instructions arrive at the R Stage at the same time: `add`, `lw` and `div`. Is it necessary to stall the processor? Explain it theoretically and demonstrate it with an example program. Simulate the program on RVfpgaEL2-Pipeline.

The program is provided at
Labs/RVfpgaLabsSolutions/Lab14/3InstructionsRstage_Instructions. This is the code:

```
.data
D: .word 11, 10, 9, 8, 7, 6

.text
Test_Assembly:

la x29, D

li x30, 0xFFFF

add a1, zero, 1
add a2, zero, 1
add a3, zero, 1
add a4, zero, 1
add a5, zero, 1
li x30, 0xFF

REPEAT:
    lw x28, (x29)
    add x30, x30, -1
    add a1, a1, 1
    div x31, x29, x30
    add a2, a2, 1
    add a3, a3, 1
    add a4, a4, 1
    add a5, a5, 1
    bne x30, zero, REPEAT    # Repeat the loop
```

clk=									
dec_i0_pc_d_ext[31:0]=	00000434	00000438	0000043C	00000440	00000444	00000448	0000044C	00000450	00000454
dec_i0_instr_d[31:0]=	000EAE03	FFFF0F13	00158593	03EECFB3	00160613	00168693	00170713	00178793	FE0F10E3
lsu_axi_arvalid=									
lsu_axi_araddr[31:0]=	00003350								
lsu_axi_rvalid=									
lsu_axi_rdata[63:0]=	CA11AB1EBADCAB1E						0000000B0000+	CA11AB1EBADCAB1E	
waddr0[4:0]=	0F	01	1C	1E	0B	1F	0C	0D	0E
wen0=									
wd0[31:0]=	0000000A	00000000	000000F5	0000000B					
waddr1[4:0]=	00	1C							03
wen1=									
wd1[31:0]=	00000000							0000000B	03
waddr2[4:0]=	1F								
wen2=									
wd2[31:0]=	00000000							00000035	03

The three writes can be performed in the same cycle as the Register File contains 3 write ports.

This is the simulation on RVfpgaEL2-Pipeline in the cycle when the three instructions write the RF:

	D and zero,t4,t5		X addi a5,a5,1	R addi a4,a4,1
Register File	ra0=29 ra1=30	rd0=013140 rd1=000252		wa0=14 we0=1 wd0=4 wa1=28 we1=1 wd1=11 wa2=31 we2=1 wd2=52
Bypasses	Bypass0=000000 Bypass1=000000			

- 4) You can perform a similar study for the `div` instruction as the one performed in Lab 12 for arithmetic-logic instructions: view the flow of the instruction through the pipeline stages, analyse the control bits, etc.

Solution not provided for this exercise.

- 5) Analyse `mul` instructions in VeeR EL2, both theoretically and practically with example programs.

Solution not provided for this exercise.

- 6) Replace the divide unit, implemented in module `el2_exu_div_ctl`, with your own unit or an open-source unit downloaded from the Internet.

Solution not provided for this exercise.