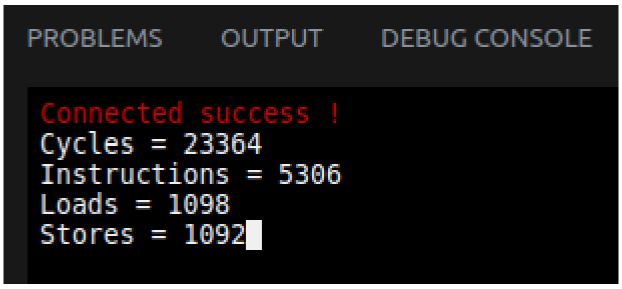
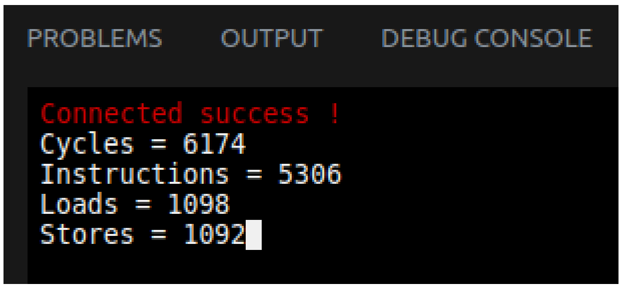
TASK: Using the HW Counters, measure the number of cycles, instructions, loads and stores in the program from **Error! Reference source not found.**. How much time in total (both for reading and writing) does it take to access Main Memory? You can compare the execution when using the DDR memory as in Figure 3 and when using the DCCM (another PlatformIO project is provided at *[RVfpgaEL2NexysA7DDRPath]/Labs/Lab19/LW-SW_Instruction_DCCM/*, which contains the same program prepared for reading from / writing to the DCCM).



```
Connected success !
Cycles = 23364
Instructions = 5306
Loads = 1098
Stores = 1092
```

- The loop contains 5 instructions.
- Ideally, IPC could be up to 1.
- However, we miss several cycles per iteration due to the read/write latency to Main Memory.

If we now execute the program that uses the DCCM, we obtain:

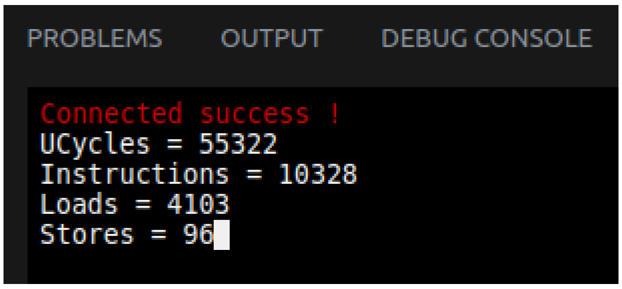


```
Connected success !
Cycles = 6174
Instructions = 5306
Loads = 1098
Stores = 1092
```

- Now the IPC is closer to 1, as the DCCM has 1 cycle read latency.

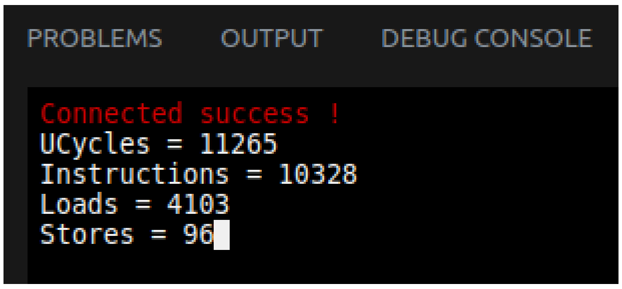
TASK: Use the example from *[RVfpgaEL2NexysA7DDRPath]/Labs/Lab19/LW_Instruction_MainMemory* to estimate the Main Memory read latency using the HW Counters. As in the previous task, you can use the example from *[RVfpgaEL2NexysA7DDRPath]/Labs/Lab19/LW_Instruction_DCCM* to compare with a program with no stalls due to the memory accesses.

Execution in Main Memory:


PROBLEMS OUTPUT DEBUG CONSOLE

```
Connected success !
UCycles = 55322
Instructions = 10328
Loads = 4103
Stores = 96
```

Execution in DCCM:

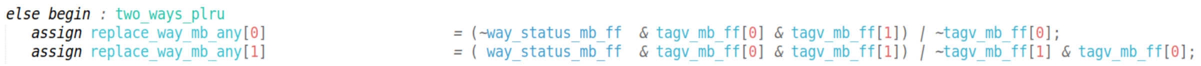

PROBLEMS OUTPUT DEBUG CONSOLE

```
Connected success !
UCycles = 11265
Instructions = 10328
Loads = 4103
Stores = 96
```

<u>TASK</u>: Analyse module **ifu_ic_mem** and the parameters of file *el2_param.vh* to understand how the elements in **Error! Reference source not found.** are implemented.

Solution not provided for this exercise.

<u>TASK</u>: Analyse the Verilog code from **Error! Reference source not found.** and explain how it operates based on the above explanations.


```
else begin : two_ways_plru
   assign replace_way_mb_any[0]                    = (~way_status_mb_ff  & tagv_mb_ff[0] & tagv_mb_ff[1]) | ~tagv_mb_ff[0];
   assign replace_way_mb_any[1]                    = ( way_status_mb_ff  & tagv_mb_ff[0] & tagv_mb_ff[1]) | ~tagv_mb_ff[1] & tagv_mb_ff[0];
```

If both ways are invalid (i.e. **`tagv_mb_ff = 00`**), way 0 must be replaced first:

- replace_way_mb_any[0] = 1, as the second operand of the OR, which is ~tagv_mb_ff[0], is 1.
- replace_way_mb_any[1] = 0, as the two operands of the OR are 0.

If there is one invalid way, this is the one replaced.

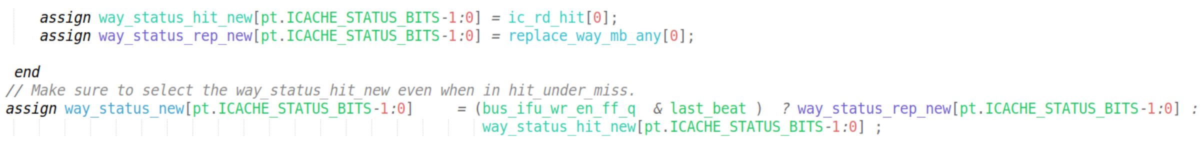
- If way 0 is invalid, the second operand of the OR, which is ~tagv_mb_ff[0], is 1.
- If way 1 is invalid and way 0 is valid, the second operand of the OR, which is ~tagv_mb_ff[1] & tagv_mb_ff[0], is 1.

If both ways are valid, signal **`way_status_mb_ff`** holds the LRU state (thus, the least recently used way, or the way to replace first) of the selected set, determines the way to replace.

<u>TASK</u>: Analyse the Verilog code that performs the same functionality on a 4-way I$.

Solution not provided for this exercise.

TASK: Analyse the Verilog code from **Error! Reference source not found.** and explain how it operates based on the above explanations.



```
    assign way_status_hit_new[pt.ICACHE_STATUS_BITS-1:0] = ic_rd_hit[0];
    assign way_status_rep_new[pt.ICACHE_STATUS_BITS-1:0] = replace_way_mb_any[0];

 end
// Make sure to select the way_status_hit_new even when in hit_under_miss.
assign way_status_new[pt.ICACHE_STATUS_BITS-1:0]     = (bus_ifu_wr_en_ff_q  & last_beat )  ? way_status_rep_new[pt.ICACHE_STATUS_BITS-1:0] :
                                                        way_status_hit_new[pt.ICACHE_STATUS_BITS-1:0] ;
```

The new value of the LRU state is determined by signal **`way_status_new`**.

- If there was a hit, signal **`ic_rd_hit`** determines the new value, as it holds the way where the hit has taken place.
- If there was a replacement, signal **`replace_way_mb_any`** determines the new value, as it holds the way that has been replaced.

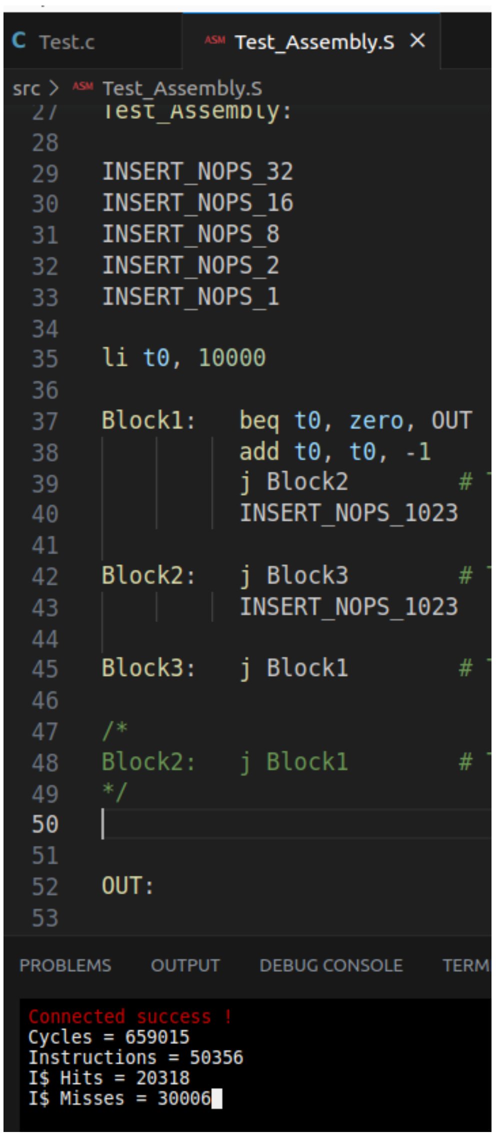
TASK: Analyse the Verilog code that performs the same functionality on a 4-way I$.

Solution not provided for this exercise.

1. EXERCISES

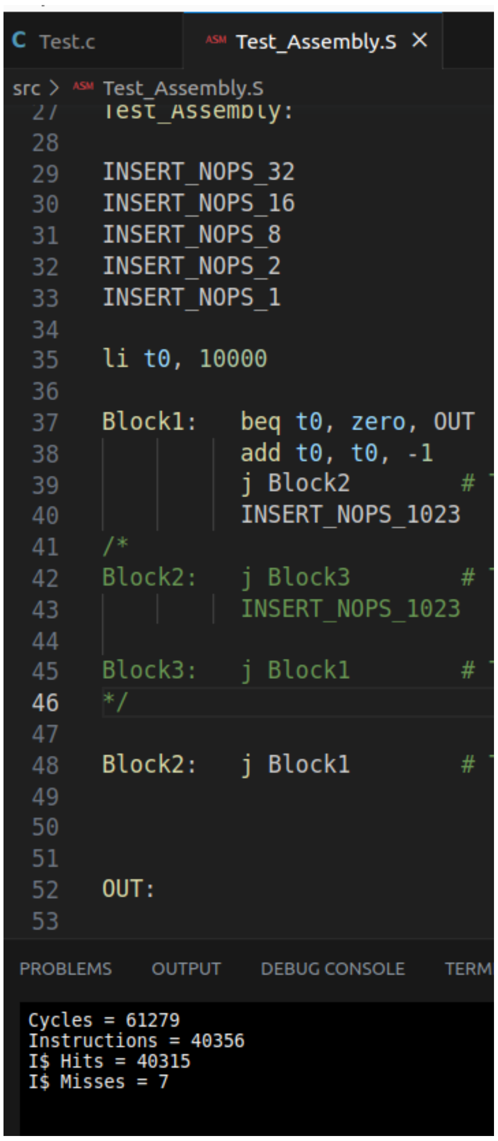
1) Transform the infinite loop from **Error! Reference source not found.** into a loop with 10000 iterations, but keep the `j` instructions at the same addresses. Measure the number of cycles and I$ hits and misses. Then remove one of the `j` instructions and measure the same metrics. Compare and explain the results.

   A Catapult project is provided at: *[RVfpgaEL2NexysA7DDRPath]/Labs/RVfpgaLabsSolutions/Lab19/InstructionMemory_LRU_Example_FiniteLoop*



```
Test_Assembly:

INSERT_NOPS_32
INSERT_NOPS_16
INSERT_NOPS_8
INSERT_NOPS_2
INSERT_NOPS_1

li t0, 10000

Block1:   beq t0, zero, OUT
          add t0, t0, -1
          j Block2          #
          INSERT_NOPS_1023

Block2:   j Block3          #
          INSERT_NOPS_1023

Block3:   j Block1          #

/*
Block2:   j Block1          #
*/


OUT:
```

```
Connected success !
Cycles = 659015
Instructions = 50356
I$ Hits = 20318
I$ Misses = 30006
```



```
Test_Assembly:

INSERT_NOPS_32
INSERT_NOPS_16
INSERT_NOPS_8
INSERT_NOPS_2
INSERT_NOPS_1

li t0, 10000

Block1:   beq t0, zero, OUT
          add t0, t0, -1
          j Block2          #
          INSERT_NOPS_1023
/*
Block2:   j Block3          #
          INSERT_NOPS_1023

Block3:   j Block1          #
*/

Block2:   j Block1          #



OUT:
```

```
Cycles = 61279
Instructions = 40356
I$ Hits = 40315
I$ Misses = 7
```

- The number of I$ misses in the code with 3 jump instructions is 3 per iteration (30000 / 10000 = 3).
- There are no I$ misses in the code with 2 jump instructions, except for the first iteration. This dramatically decreases the number of cycles.

2) Extend **Error! Reference source not found.** to analyse in detail how each 64-bit chunk is written in the I$.

Solution not provided for this exercise.

3) Analyse in simulation and on the board other I$ configurations. For example, it can be very interesting to analyse a 4-way I$.

Solution not provided for this exercise.

You can find a useful study in RVfpga v2.2 (provided at: https://university.imgtec.com/rvfpga-download-page-en/), Lab 19, where a 4-way I$ is used in SweRV EH1.

4) Analyse the logic that checks the correctness of the parity information.

Solution not provided for this exercise.