


```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
> Executing task: platformio device mon

--- Available filters and text transfo
--- More details at http://bit.ly/pio-
--- Miniterm on /dev/ttyUSB1 115200,8
--- Quit: Ctrl+C | Menu: Ctrl+T | Help:

Cycles = 30245
Instructions = 50051

```

Ciclos por iteração = 3

DDR Memory:

Execução na placa:

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
> Executing task: platformio device mon

--- Available filters and text transfo
--- More details at http://bit.ly/pio-m
--- Miniterm on /dev/ttyUSB1 115200,8,
--- Quit: Ctrl+C | Menu: Ctrl+T | Help:

Cycles = 357774
Instructions = 50051

```

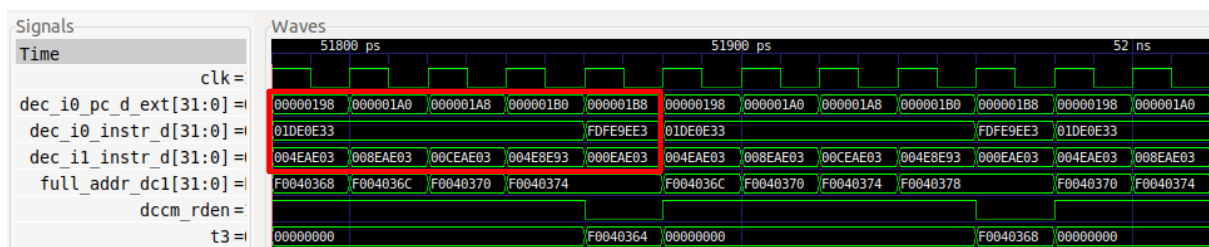
O número de instruções é o mesmo, uma vez que o programa é o mesmo. No entanto, agora são necessários cerca de 358000 ciclos para executar todas as iterações, portanto:

Número de ciclos gastos a aceder à memória por iteração $\approx (358000 - 30000) / 10000 \approx 33$

TAREFA: Use o exemplo de `[RVfpgaPath]/RVfpga/Labs/Lab19/LW_Instruction_ExtMem` para estimar a latência de leitura da memória externa DDR usando os contadores HW. Tal como na tarefa anterior, pode utilizar o exemplo de `[RVfpgaPath]/RVfpga/Labs/Lab19/LW_Instruction_DCCM` para comparar com um programa sem paragens devido aos acessos à memória. Lembre-se de que a memória simulada não é a mesma que a memória DDR real na placa Nexys A7.

DCCM:

Simulação no Verilator:



Cada iteração executa 10 instruções em 5 ciclos, pelo que executa com o IPC ideal.

Execução na placa:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

> Executing task: platformio device mon

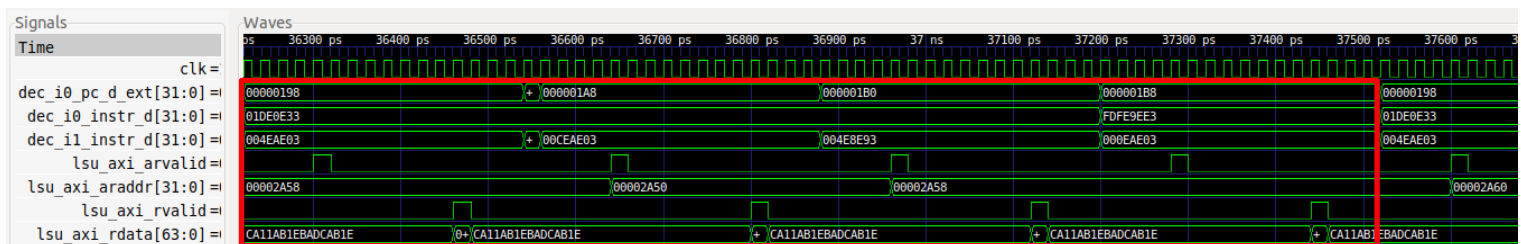
--- Available filters and text transform
--- More details at http://bit.ly/pio-m
--- Miniterm on /dev/ttyUSB1 115200,8,1
--- Quit: Ctrl+C | Menu: Ctrl+T | Help:

Cycles = 50237
Instructions = 100051
```

Ciclos por iteração = 5

Memória DDR:

Simulação no Verilator:



Execução na placa:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

> Executing task: platformio device monitor <

--- Available filters and text transformations
--- More details at http://bit.ly/pio-monitor-
--- Miniterm on /dev/ttyUSB1 115200,8,N,1 ---
--- Quit: Ctrl+C | Menu: Ctrl+T | Help: Ctrl+T

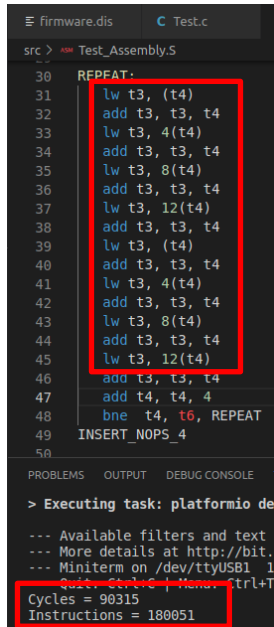
Cycles = 938723
Instructions = 100051
```

O número de instruções é o mesmo, uma vez que o programa é o mesmo. No entanto, agora são necessários cerca de 939000 ciclos para executar todas as iterações, portanto:

Latência de uma leitura da memória DDR $\approx (939000 - 50000) / (10000 * 4) \approx 22$

Para verificar se está correto, duplicamos o número de instruções de leitura e executamos o programa novamente:

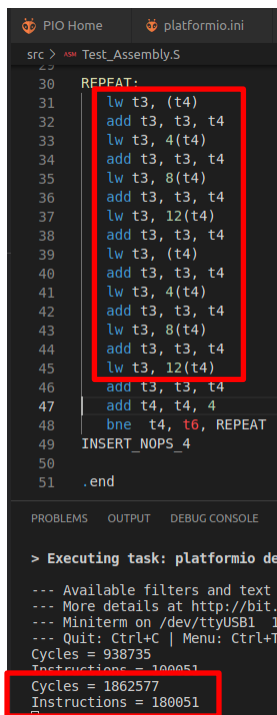
DCCM:



The screenshot shows the RVfpga IDE with the assembly file `Test_Assembly.S` open. The code contains a loop labeled `REPEAT` that performs a series of load (`lw`) and add (`add`) instructions on registers `t3` and `t4`. The instructions are highlighted with a red box. Below the code, the execution results are displayed in the `DEBUG CONSOLE` tab, showing `Cycles = 90315` and `Instructions = 180051`, both highlighted with a red box.

```
src > Test_Assembly.S
30 REPEAT:
31 lw t3, (t4)
32 add t3, t3, t4
33 lw t3, 4(t4)
34 add t3, t3, t4
35 lw t3, 8(t4)
36 add t3, t3, t4
37 lw t3, 12(t4)
38 add t3, t3, t4
39 lw t3, (t4)
40 add t3, t3, t4
41 lw t3, 4(t4)
42 add t3, t3, t4
43 lw t3, 8(t4)
44 add t3, t3, t4
45 lw t3, 12(t4)
46 add t3, t3, t4
47 add t4, t4, 4
48 bne t4, t6, REPEAT
49 INSERT_NOPS_4
50
> Executing task: platformio dev
--- Available filters and text
--- More details at http://bit.
--- Miniterm on /dev/ttyUSB1 1
--- Quit: Ctrl+C | Menu: Ctrl+T
Cycles = 90315
Instructions = 180051
```

Memória DDR:



The screenshot shows the RVfpga IDE with the assembly file `Test_Assembly.S` open. The code is identical to the one in the DCCM test, with a loop labeled `REPEAT` performing load and add instructions. The instructions are highlighted with a red box. Below the code, the execution results are displayed in the `DEBUG CONSOLE` tab, showing `Cycles = 938735` and `Instructions = 180051`, both highlighted with a red box.

```
src > Test_Assembly.S
30 REPEAT:
31 lw t3, (t4)
32 add t3, t3, t4
33 lw t3, 4(t4)
34 add t3, t3, t4
35 lw t3, 8(t4)
36 add t3, t3, t4
37 lw t3, 12(t4)
38 add t3, t3, t4
39 lw t3, (t4)
40 add t3, t3, t4
41 lw t3, 4(t4)
42 add t3, t3, t4
43 lw t3, 8(t4)
44 add t3, t3, t4
45 lw t3, 12(t4)
46 add t3, t3, t4
47 add t4, t4, 4
48 bne t4, t6, REPEAT
49 INSERT_NOPS_4
50
51 .end

> Executing task: platformio dev
--- Available filters and text
--- More details at http://bit.
--- Miniterm on /dev/ttyUSB1 1
--- Quit: Ctrl+C | Menu: Ctrl+T
Cycles = 938735
Instructions = 180051
Cycles = 1862577
Instructions = 180051
```

Latência de uma leitura da memória DDR $\approx (1862000 - 90000) / (10000 * 8) \approx 22$

TAREFA: Um exercício bastante complexo mas muito interessante é analisar o Controlador de Memória utilizado no Sistema RVfpga. Lembre-se que pode encontrar os módulos que compõem este controlador na pasta *[RVfpgaPath]/RVfpga/src/LiteDRAM*, e que o módulo de topo está implementado no ficheiro *litedram_top.v* dentro dessa pasta. Pode começar com a simulação da Figura 3 e adicionar e analisar alguns sinais do controlador da LiteDRAM.

Solução não fornecida.

TAREFA: Analise o módulo **ifu_ic_mem** para compreender como os elementos da Figura 4 estão implementados.

Módulo **ifu_ic_mem**:

Instanciação do Data Array e do Tag Array:

```
IC_TAG #( .ICACHE_TAG_HIGH(ICACHE_TAG_HIGH) ,
          .ICACHE_TAG_LOW(ICACHE_TAG_LOW) ,
          .ICACHE_TAG_DEPTH(ICACHE_TAG_DEPTH)
        ) ic_tag_inst
    (
        .*,
        .ic_wr_en      (ic_wr_en[3:0]),
        .ic_debug_addr (ic_debug_addr[ICACHE_TAG_HIGH-1:2]),
        .ic_rw_addr    (ic_rw_addr[31:3])
    ) ;

IC_DATA #( .ICACHE_TAG_HIGH(ICACHE_TAG_HIGH) ,
          .ICACHE_TAG_LOW(ICACHE_TAG_LOW) ,
          .ICACHE_IC_DEPTH(ICACHE_IC_DEPTH)
        ) ic_data_inst
    (
        .*,
        .ic_wr_en      (ic_wr_en[3:0]),
        .ic_debug_addr (ic_debug_addr[ICACHE_TAG_HIGH-1:2]),
        .ic_rw_addr    (ic_rw_addr[ICACHE_TAG_HIGH-1:2])
    ) ;
```

Data Array e bits Parity (No nosso caso, o RV_ICACHE_ECC não está definido):

```

for (genvar i=0; i<NUM_WAYS; i++) begin: WAYS

    for (genvar k=0; k<NUM_SUBBANKS; k++) begin: SUBBANKS // 16B subbank

        // way3-bank3, way3-bank2, ... way0-bank0
        assign ic_bank_way_clken[i][k] = ic_bank_read[k] | ic_b_sb_wren[k][i];

        rvoclkhdr bank_way_bank_c1_cgic ( .en(ic_bank_way_clken[i][k] | clk_override), .l1clk(ic_bank_way_clk[i][k]), .* );

    `ifdef RV_ICACHE_ECC
    `RV_ICACHE_DATA_CELL ic_bank_sb_way_data (
        .CLK(ic_bank_way_clk[i][k]),
        .WE (ic_b_sb_wren[k][i]),
        .D (ic_sb_wr_data[k][41:0]),
        .ADR(ic_rw_addr_q[ICACHE_TAG_HIGH-1:4]),
        .Q (wb_dout[i][((k+1)*42-1:k*42])
    );
    `else
    `RV_ICACHE_DATA_CELL ic_bank_sb_way_data (
        .CLK(ic_bank_way_clk[i][k]),
        .WE (ic_b_sb_wren[k][i]),
        .D (ic_sb_wr_data[k][33:0]),
        .ADR(ic_rw_addr_q[ICACHE_TAG_HIGH-1:4]),
        .Q (wb_dout[i][((k+1)*34-1:k*34])
    );
    `endif
end // block: SUBBANKS

end

```

Multiplexer 4-1:

```

`else
    logic [135:0] ic_premux_data_ext;
    logic [3:0] [135:0] wb_dout_way;
    logic [3:0] [135:0] wb_dout_way_with_premux;

    assign ic_premux_data_ext[135:0] = {2'b0,ic_premux_data[127:96],2'b0,ic_premux_data[95:64],2'b0,ic_premux_data[63:32],2'b0,ic_premux_data[31:0]};
    assign wb_dout_way[0][135:0] = wb_dout[0][135:0] & { {34{ic_bank_read_ff[3]}}, {34{ic_bank_read_ff[2]}}, {34{ic_bank_read_ff[1]}}, {34{ic_bank_read_ff[0]}} };
    assign wb_dout_way[1][135:0] = wb_dout[1][135:0] & { {34{ic_bank_read_ff[3]}}, {34{ic_bank_read_ff[2]}}, {34{ic_bank_read_ff[1]}}, {34{ic_bank_read_ff[0]}} };
    assign wb_dout_way[2][135:0] = wb_dout[2][135:0] & { {34{ic_bank_read_ff[3]}}, {34{ic_bank_read_ff[2]}}, {34{ic_bank_read_ff[1]}}, {34{ic_bank_read_ff[0]}} };
    assign wb_dout_way[3][135:0] = wb_dout[3][135:0] & { {34{ic_bank_read_ff[3]}}, {34{ic_bank_read_ff[2]}}, {34{ic_bank_read_ff[1]}}, {34{ic_bank_read_ff[0]}} };

    assign wb_dout_way_with_premux[0][135:0] = ic_sel_premux_data ? ic_premux_data_ext[135:0] : wb_dout_way[0][135:0] ;
    assign wb_dout_way_with_premux[1][135:0] = ic_sel_premux_data ? ic_premux_data_ext[135:0] : wb_dout_way[1][135:0] ;
    assign wb_dout_way_with_premux[2][135:0] = ic_sel_premux_data ? ic_premux_data_ext[135:0] : wb_dout_way[2][135:0] ;
    assign wb_dout_way_with_premux[3][135:0] = ic_sel_premux_data ? ic_premux_data_ext[135:0] : wb_dout_way[3][135:0] ;

    assign ic_rd_data[135:0] = ((136{ic_rd_hit_q[0] | ic_sel_premux_data}) & wb_dout_way_with_premux[0][135:0]) |
        ((136{ic_rd_hit_q[1] | ic_sel_premux_data}) & wb_dout_way_with_premux[1][135:0]) |
        ((136{ic_rd_hit_q[2] | ic_sel_premux_data}) & wb_dout_way_with_premux[2][135:0]) |
        ((136{ic_rd_hit_q[3] | ic_sel_premux_data}) & wb_dout_way_with_premux[3][135:0]) ;

`endif

```

Tag Array e bits Parity (No nosso caso, RV_ICACHE_ECC não está definido):

```

for (genvar i=0; i<NUM_WAYS; i++) begin: WAYS

    rvoclkhdr ic_tag_clk_cgc ( .en(ic_tag_clken[i]), .clk(ic_tag_clk[i]), .* );

    if (ICACHE_TAG_DEPTH == 64 ) begin : ICACHE_SZ_16
        `ifdef RV_ICACHE_ECC
            ram_64x25 ic_way_tag (
                .CLK(ic_tag_clk[i]),
                .WE (ic_tag_wren_q[i]),
                .D (ic_tag_wr_data[24:0]),
                .ADR(ic_rw_addr_q[ICACHE_TAG_HIGH-1:ICACHE_TAG_LOW]),
                .Q (ic_tag_data_raw[i][24:0])
            );

            assign w_tout[i][31:ICACHE_TAG_HIGH] = ic_tag_data_raw[i][31-ICACHE_TAG_HIGH:0] ;
            assign w_tout[i][36:32] = ic_tag_data_raw[i][24:20] ;

            rvecc_decode ecc_decode (
                .en(~dec_tlu_core_ecc_disable),
                .sed_ded ( 1'b1 ), // 1 : means only detection
                .din({12'b0,ic_tag_data_raw[i][19:0]}),
                .ecc_in({2'b0, ic_tag_data_raw[i][24:20]}),
                .dout(ic_tag_corrected_data_unc[i][31:0]),
                .ecc_out(ic_tag_corrected_ecc_unc[i][6:0]),
                .single_ecc_error(ic_tag_single_ecc_error[i]),
                .double_ecc_error(ic_tag_double_ecc_error[i]));

            assign ic_tag_way_perr[i]= ic_tag_single_ecc_error[i] | ic_tag_double_ecc_error[i] ;
        `else
            ram_64x21 ic_way_tag (
                .CLK(ic_tag_clk[i]),
                .WE (ic_tag_wren_q[i]),
                .D (ic_tag_wr_data[20:0]),
                .ADR(ic_rw_addr_q[ICACHE_TAG_HIGH-1:ICACHE_TAG_LOW]),
                .Q (ic_tag_data_raw[i][20:0])
            );

            assign w_tout[i][31:ICACHE_TAG_HIGH] = ic_tag_data_raw[i][31-ICACHE_TAG_HIGH:0] ;
            assign w_tout[i][32] = ic_tag_data_raw[i][20] ;

            rveven_paritycheck #(32-ICACHE_TAG_HIGH) parcheck(.data_in (w_tout[i][31:ICACHE_TAG_HIGH]),
                .parity_in (w_tout[i][32]),
                .parity_err(ic_tag_way_perr[i]));
        `endif
    end // block: ICACHE_SZ_16

```

Comparadores:


```

assign ic_rd_hit[0] = (w_tout[0][31:ICACHE_TAG_HIGH] == ic_rw_addr_ff[31:ICACHE_TAG_HIGH]) & ic_tag_valid[0];
assign ic_rd_hit[1] = (w_tout[1][31:ICACHE_TAG_HIGH] == ic_rw_addr_ff[31:ICACHE_TAG_HIGH]) & ic_tag_valid[1];
assign ic_rd_hit[2] = (w_tout[2][31:ICACHE_TAG_HIGH] == ic_rw_addr_ff[31:ICACHE_TAG_HIGH]) & ic_tag_valid[2];
assign ic_rd_hit[3] = (w_tout[3][31:ICACHE_TAG_HIGH] == ic_rw_addr_ff[31:ICACHE_TAG_HIGH]) & ic_tag_valid[3];

```


TAREFA: Replique a simulação da Figura 6 no seu computador. Para o fazer, siga os passos seguintes (descritos em pormenor na Secção 7 do GSG):

- Se necessário, gere o binário de simulação (Vrvfpgasim).
- No PlatformIO, abra o projeto fornecido em:
[RVfpgaPath]/RVfpga/Labs/Lab19/InstructionMemory_Example.
- Atualize o caminho para o binário de simulação RVfpga (Vrvfpgasim) no ficheiro *platformio.ini*.
- Gere o trace da simulação com o Verilator (Generate Trace).
- Abra o trace no GTKWave.
- Use o ficheiro *test1_Miss.tcl* (fornecido em
[RVfpgaPath]/RVfpga/Labs/Lab19/InstructionMemory_Example) para abrir os mesmos sinais que os mostrados na Figura 6. Para isso, no GTKWave, clique em File → Read Tcl Script File e selecione o ficheiro *test1_Miss.tcl*.

- Clique em *Zoom In* () várias vezes e analise a região de 28900 ps a 30220 ps.

Também pode analisar algumas coisas com mais pormenor, como a escrita na I\$ ou o bypass das instruções iniciais.

Solução apresentada no documento principal do Lab 19.

TAREFA: Reproduza a simulação da Figura 7 no seu computador. Utilize o ficheiro *test1_Hit.tcl* (em *[RVfpgaPath]/RVfpga/Labs/Lab19/InstructionMemory_Example*). *Zoom In* () várias vezes e passe para 34680ps.

Solução apresentada no documento principal do Lab 19.

TAREFA: Analise o código Verilog da Figura 9 e explique o seu funcionamento com base nas explicações anteriores.

Solução não fornecida.

TAREFA: Analise o código Verilog da Figura 10 e explique o seu funcionamento com base nas explicações anteriores.

Solução não fornecida.

1. EXERCÍCIOS

- 1) Transforme o ciclo infinito da Figura 11 num ciclo com 0x10000 iterações, mas mantenha as instruções *j* nos mesmos endereços. Meça o número de ciclos e I\$ hits e misses. Em seguida, remova uma das instruções *j* e meça as mesmas métricas. Compare e explique os resultados.

5 instruções jump:

4 instruções jump:

```
PIO Home platformio.ini Test.c
src > Test_Assembly.S
27 Test_Assembly:
28
29 li t0, 0x10000
30 INSERT_NOPS_16
31 INSERT_NOPS_2
32
33 Set8_Block1: beq t0, zero, OUT
34 add t0, t0, -1
35 j Set8_Block2
36 INSERT_NOPS_1023
37
38 Set8_Block2: j Set8_Block3
39 INSERT_NOPS_1023
40
41 Set8_Block3: j Set8_Block4
42 INSERT_NOPS_1023
43
44 Set8_Block4: j Set8_Block5
45 INSERT_NOPS_1023
46
47 Set8_Block5: j Set8_Block1
48
49 OUT:
50
51 ret
52

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
> Executing task: platformio device monitor
--- Available filters and text transformations
--- More details at http://bit.ly/pio-monitor
--- Miniterm on /dev/ttyUSB1 115200,8,N,1
Quit: Ctrl+C | Menu: Ctrl+T | Help: Ctrl+?

Hits = 131142
Miss = 327679
Cycles = 11531154
```

```
PIO Home platformio.ini Test.c
src > Test_Assembly.S
25 .text
26
27 Test_Assembly:
28
29 li t0, 0x10000
30 INSERT_NOPS_16
31 INSERT_NOPS_2
32
33 Set8_Block1: beq t0, zero, OUT
34 add t0, t0, -1
35 j Set8_Block2
36 INSERT_NOPS_1023
37
38 Set8_Block2: j Set8_Block4
39 INSERT_NOPS_1023
40
41 Set8_Block4: j Set8_Block5
42 INSERT_NOPS_1023
43
44 Set8_Block5: j Set8_Block1
45
46 OUT:
47
48 ret
49
50 .end

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
> Executing task: platformio device monitor
--- Available filters and text transformations
--- More details at http://bit.ly/pio-monitor
--- Miniterm on /dev/ttyUSB1 115200,8,N,1
Quit: Ctrl+C | Menu: Ctrl+T | Help: Ctrl+?

Hits = 1179696
Miss = 6
Cycles = 1704291
```

No programa com 4 instruções `j`, o número de I\$ misses e o número de ciclos diminuem drasticamente, pois agora os blocos não entram em conflito uns com os outros. Ao mesmo tempo, o número de I\$ hits aumenta bastante.

2) Utilize o programa da Figura 5 para analisar um acerto I\$ do ponto de vista da Política de Substituição I\$.

Solução não fornecida.

3) Expanda a Figura 6 para analisar em pormenor a forma como cada pedaço de 64 bits é escrito na I\$.

Solução não fornecida.

4) Analise em simulação e na placa outras configurações da I\$, como uma I\$ com um tamanho de bloco diferente. Lembre-se que o número de vias não pode ser modificado.

Solução não fornecida.

5) Analise a lógica que verifica a correção da informação de paridade do Data Array e do Tag Array.

Solução não fornecida.