

任務

任務：在自己的電腦上重複圖3中的模擬過程。為此，請按照以下步驟操作（在GSG的第7部分中詳述）：

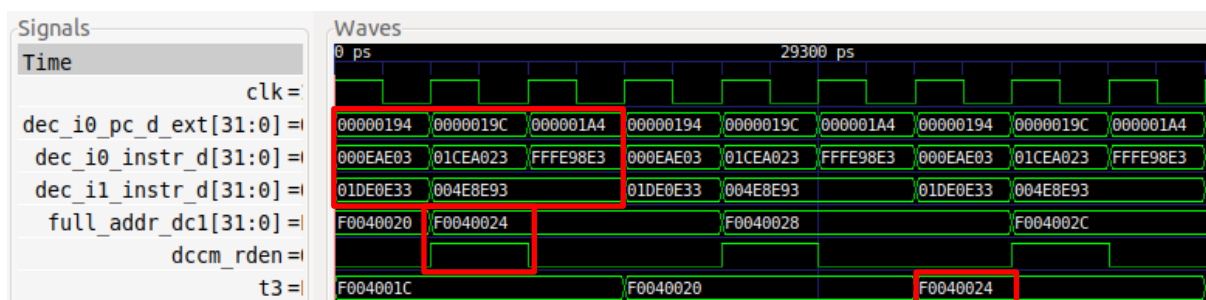
- 必要時產生模擬二進位檔案（*Vrvfpgasim*）。
- 在PlatformIO中，開啟在以下位置提供的專案：*[RVfpgaPath]/RVfpga/Labs/Lab19/LW-SW_Instruction_ExtMemory*。
- 在檔案*platformio.ini*中建立到RVfpga模擬二進位檔案（*Vrvfpgasim*）的正確路徑。
- 使用Verilator產生模擬軌跡（產生軌跡）。
- 在GTKWave上開啟軌跡。
- 使用檔案*test_Blocking_Extended.tcl*（在*[RVfpgaPath]/RVfpga/Labs/Lab19/LW-SW_Instruction_ExtMemory*中提供）開啟與圖6所示訊號相同的訊號。為此，在GTKWave上，按一下「File → Read Tcl Script File」（檔案 → 讀取Tcl指令碼檔案）並選擇*test_Blocking_Extended.tcl*檔案。
- 按幾次「Zoom In」（放大）（），然後分析自42500 ps起的區域。

解答請參見實驗19的主文件。

任務：使用硬體計數器測量圖2中程式的週期數、指令數、載入次數和儲存次數。存取DDR外部記憶體所用的總時間是多少（包括讀取存取和寫入存取）？可以比較使用圖3中的DDR記憶體與使用DCCM時的執行情況（*[RVfpgaPath]/RVfpga/Labs/Lab19/LW-SW_Instruction_DCCM*下提供另一個PlatformIO專案，其中包含用於對DCCM進行讀/寫操作的相同程式）。請注意，用於模擬的記憶體不是Nexys A7開發板上實際使用的DDR記憶體。

DCCM：

在Verilator中模擬：



每次迭代需要3個週期，執行5條指令。每次迭代僅遺失半個週期。

在開發板上執行：

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

> Executing task: platformio device mon

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

Cycles = 30245
Instructions = 50051

```

每次迭代的週期數 = 3

DDR記憶體：

在開發板上執行：

```

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,
--- Quit: Ctrl+C | Menu: Ctrl+T | Help:

Cycles = 357774
Instructions = 50051

```

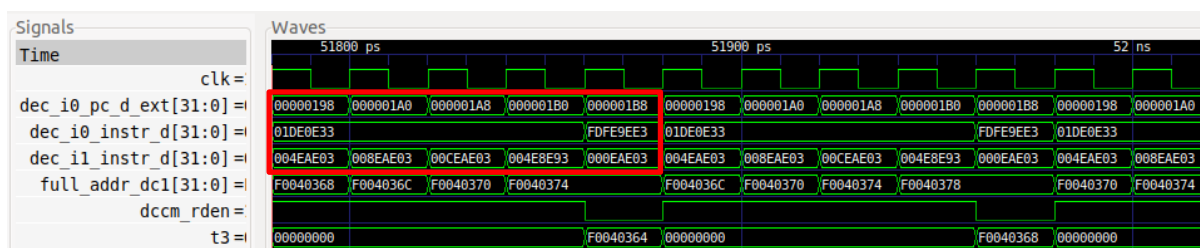
因為程式未發生變化，所以指令數相同。但此時執行所有迭代需要大約358000個週期，因此：

每次迭代中存取記憶體所用的週期數 $\approx (358000 - 30000) / 10000 \approx 33$

任務：使用[RVfpgaPath]/RVfpga/Labs/Lab19/LW_Instruction_ExtMem中提供的範例，藉助硬體計數器估算DDR外部記憶體的讀取延遲。與上一任務一樣，可以使用[RVfpgaPath]/RVfpga/Labs/Lab19/LW_Instruction_DCCM中的範例，將現有程式與因記憶體存取而不存在暫停的程式進行比較。請注意，用於模擬的記憶體不是Nexys A7開發板上實際使用的DDR記憶體。

DCCM：

在Verilator中模擬：



每次迭代需要5個週期，執行10條指令，因此IPC為理想值。

在開發板上執行：

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

> Executing task: platformio device mon

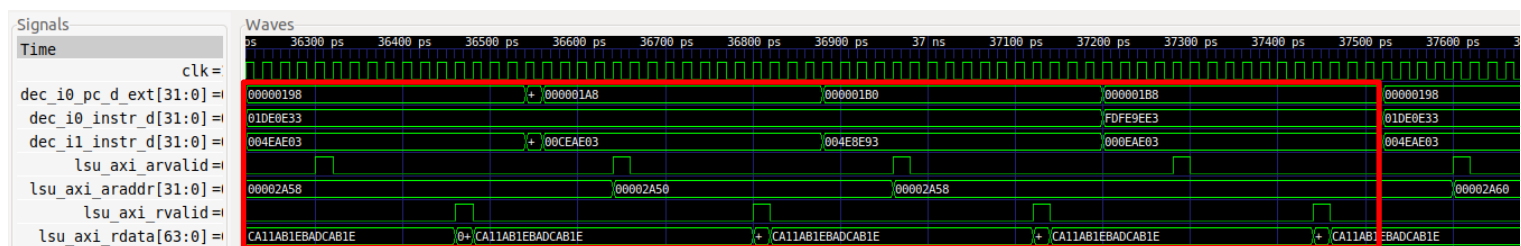
--- Available filters and text transform
--- More details at http://bit.ly/pio-m
--- Miniterm on /dev/ttyUSB1 115200,8,
--- Quit: Ctrl+C | Menu: Ctrl+T | Help:

Cycles = 50237
Instructions = 100051
```

每次迭代的週期數 = 5

DDR記憶體：

在Verilator中模擬：



在開發板上執行：

```
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

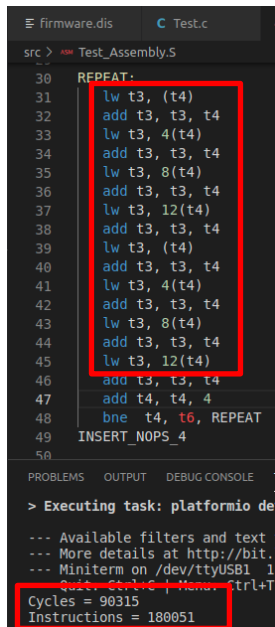
Cycles = 938723
Instructions = 100051
```

因為程式未發生變化，所以指令數相同。但此時執行所有迭代需要大約939000個週期，因此：

DDR記憶體的讀取延遲 $\approx (939000 - 50000) / (10000 * 4) \approx 22$

為了檢查該值是否正確，我們將載入指令的數量加倍，然後再次執行程式：

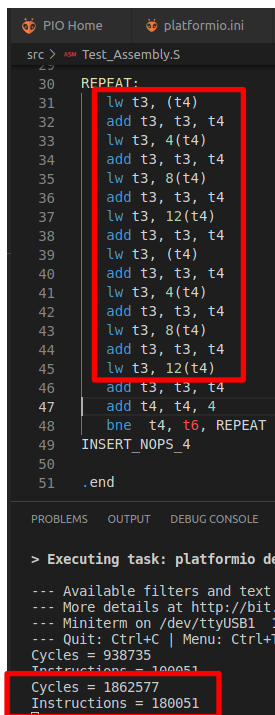
DCCM：



The screenshot shows the PlatformIO IDE interface. The top panel displays the assembly code for 'Test_Assembly.S'. A red box highlights a loop labeled 'REPEAT' containing instructions from line 31 to 45. The bottom panel shows the execution results for the task 'platformio dev'. A red box highlights the execution statistics: 'Cycles = 90315' and 'Instructions = 180051'.

```
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
> Executing task: platformio dev
--- Available filters and text
--- More details at http://bit.ly
--- Miniterm on /dev/ttyUSB1 115200
Cycles = 90315
Instructions = 180051
```

DDR記憶體：



The screenshot shows the PlatformIO IDE interface. The top panel displays the assembly code for 'Test_Assembly.S'. A red box highlights a loop labeled 'REPEAT' containing instructions from line 31 to 45. The bottom panel shows the execution results for the task 'platformio dev'. A red box highlights the execution statistics: 'Cycles = 1862577' and 'Instructions = 180051'.

```
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.ly
--- Miniterm on /dev/ttyUSB1 115200
Cycles = 1862577
Instructions = 180051
```

DDR記憶體的讀取延遲 $\approx (1862000 - 90000) / (10000 * 8) \approx 22$

任務：分析RVfpga系統中使用的記憶體控制器，本練習頗為複雜但十分有趣。請記住，構成該控制器的模組位於[RVfpgaPath]/RVfpga/src/LiteDRAM中，頂層模組在該資料夾內的litedram_top.v檔案中實作。可以先進行圖3所示的模擬，然後新增並分析來自LiteDRAM控制器的一些訊號。

不提供解答。

任務：分析模組ifu_ic_mem，瞭解如何實作圖4中的元素。

模組ifu_ic_mem：

資料陣列和標籤陣列實例化：

```
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])  
);
```

資料陣列加同位檢查（在本例中未定義RV_ICACHE_ECC）：

```

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

```

4-1 多路開關：

```

`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

```

標籤陣列加同位檢查（在本例中未定義RV_ICACHE_ECC）：

```

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

    rvoclkhdr ic_tag_c1_cgic ( .en(ic_tag_clken[i]), .l1clk(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

```


比較器：

```

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];


```

任務：在自己的電腦上重複圖6中的模擬過程。為此，請按照以下步驟操作（在GSG的第7部分中詳述）：

- 必要時產生模擬二進位檔案（*Vrvfpgasim*）。
- 在PlatformIO中，開啟在以下位置提供的專案：
[RVfpgaPath]/RVfpga/Labs/Lab19/InstructionMemory_Example。
- 在檔案*platformio.ini*中更新到RVfpga模擬二進位檔案（*Vrvfpgasim*）的路徑。
- 使用Verilator產生模擬軌跡（產生軌跡）。
- 在GTKWave上開啟軌跡。
- 使用檔案*test1_Miss.tcl*（在*[RVfpgaPath]/RVfpga/Labs/Lab19/InstructionMemory_Example*中提供）開啟與圖6所示訊號相同的訊號。為此，在GTKWave上，按一下「File → Read Tcl Script File」（檔案 → 讀取Tcl指令碼檔案）並選擇*test1_Miss.tcl*檔案。
- 按幾次「Zoom In」（放大）（），然後分析28900 ps至30220 ps範圍內的區域。

還可以進行一些更深入的分析，例如對I\$的寫入操作或初始指令的旁路。

解答請參見實驗19的主文件。

任務：在自己的電腦上重複圖7中的模擬過程。使用檔案`test1_Hit.tcl`（在 `[RVfpgaPath]/RVfpga/Labs/Lab19/InstructionMemory_Example` 中提供）。按幾次「Zoom In」（放大）（）移動至34680 ps。

解答請參見實驗19的主文件。

任務：分析圖9中的Verilog程式碼，並基於上述說明解釋程式碼如何執行。

不提供解答。

任務：分析圖10中的Verilog程式碼，並基於上述說明解釋程式碼如何執行。

不提供解答。

1. 練習

- 1) 將圖11所示的迴圈轉換為0x10000次迭代的迴圈，但為j指令保持原有的位址。測量週期數以及I\$命中和未命中的次數。然後刪除其中一條j指令，再次測量上述指標。比較測量結果，並做出解釋。

5條跳轉指令：

4條跳轉指令：


```
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
```

在採用4條j指令的程式中，IS未命中數和週期數顯著減少，因為此時區塊之間彼此不會發生衝突。與此同時，IS命中數大幅增加。

2) 使用圖5中的程式，從IS替換策略的角度分析IS命中。

不提供解答。

3) 展開圖6，詳細分析每個64位元區塊如何寫入IS。

不提供解答。

4) 透過模擬器和開發板分析其他IS組態，例如具有不同區塊大小的IS。請注意，無法修改通路的數量。

不提供解答。

5) 分析用於檢查資料陣列和標籤陣列同位檢查資訊正確性的邏輯。

不提供解答。