

## 任務

**任務：**檢查圖1的Verilog程式碼中包含的處理器元素，並解釋其工作原理。

- 解碼階段顯示的元素（暫存器檔案、指令暫存器和控制單元）位於模組**dec**、**dec\_decode\_ctl**和**dec\_gpr\_ctl**中。
- EX1階段顯示的元素位於模組**exu**和**exu\_alu\_ctl**中。
- FC1階段顯示的元素位於模組**ifu**和**ifu\_ifc\_ctl**中。

### FC1階段：

- 2:1多路開關：模組**ifu\_ifc\_ctl**

```
278     assign ifc_fetch_addr_f1[31:1] = ( ({31{exu_flush_final}} & exu_flush_path_final[31:1]) |  
279     ({31{~exu_flush_final}} & ifc_fetch_addr_f1_raw[31:1]));
```

- 5:1多路開關：模組**ifu\_ifc\_ctl**

```
150     assign fetch_addr_bf[31:1] = ( ({31{miss_sel_flush}} & exu_flush_path_final[31:1]) | // FLUSH path  
151     ({31{sel_miss_addr_bf}} & miss_addr[31:1]) | // MISS path  
152     ({31{sel_btb_addr_bf}} & ifu_bp_btb_target_f2[31:1]) | // BTB target  
153     ({31{sel_last_addr_bf}} & {ifc_fetch_addr_f1[31:1]}) | // Last cycle  
154     ({31{sel_next_addr_bf}} & {fetch_addr_next[31:1]}); // SEQ path  
155
```

- 序列位址加法器：模組**ifu\_ifc\_ctl**

```
185     assign {overflow_nc, fetch_addr_next[31:1]} = ({(1'b0, ifc_fetch_addr_f1[31:4]) + 29'b1}, 3'b0);
```

### EX1階段：

- 比較器：模組**exu\_alu\_ctl**

```
145     assign eq = a_ff[31:0] == b_ff[31:0];
```

比較器會比較兩個運算元：

- 如果兩個運算元相等：eq = 1。
- 如果兩個運算元不相等：eq = 0。

- 分支目標位址加法器：模組**exu\_alu\_ctl**

```
211     rvbradder ibradder (  
212         .pc(pc_ff[31:1]),  
213         .offset(brimm_ff[12:1]),  
214         .dout(pcout[31:1])  
215     );
```

加法器會計算PC與偏移量之和。

- 邏輯：模組**exu\_alu\_ctl**

```

202      assign actual_taken = (ap.beq & eq) |
203      (ap.bne & ne) |
204      (ap.bltn & lt) |
205      (ap.bge & ge) |
206      (any_jal);
207

```

actual\_taken中包含分支方向的結果：如果其值為1，則分支必須跳轉；如果其值為0，則分支不得跳轉。例如：

- 如果指令為beq指令 (ap.beq==1) 且兩個運算元相等 (eq==1) → actual\_taken = 1
- 如果指令為bne指令 (ap.bne==1) 且兩個運算元不相等 (ne==1) → actual\_taken = 1
- 如果指令為jal指令 (any\_jal==1)，則分支必須跳轉 → actual\_taken = 1

```

230      assign cond_mispredict = (ap.predict_t & ~actual_taken) |
231      (ap.predict_nt & actual_taken);
232

```

如果預測分支跳轉 (ap.predict\_t = 1) 但實際不跳轉 (actual\_taken = 0)，或者預測分支不跳轉 (ap.predict\_nt = 1) 但實際跳轉 (actual\_taken = 1)，則分支預測錯誤 (cond\_mispredict = 1)

```

237      assign flush_upper = ( ap.jal | cond_mispredict | target_mispredict) & valid_ff & ~flush & ~freeze;
238

```

如果分支預測錯誤 (cond\_mispredict = 1)、指令有效 (valid\_ff = 1) 並且管線尚未排清或凍結，則必須排清管線。

**任務：**解釋如何在模組exu\_alu\_ctl中透過訊號eq、控制訊號ap.beq、ap.predict\_t和ap.predict\_nt以及部分其他訊號產生訊號flush\_upper。

- 邏輯：模組exu\_alu\_ctl

```

202      assign actual_taken = (ap.beq & eq) |
203      (ap.bne & ne) |
204      (ap.bltn & lt) |
205      (ap.bge & ge) |
206      (any_jal);
207

```

actual\_taken中包含分支方向的結果：如果其值為1，則分支必須跳轉；如果其值為0，則分支不得跳轉。例如：

- 如果指令為beq指令且兩個運算元相等 → actual\_taken = 1
- 如果指令為bne指令且兩個運算元不相等 → actual\_taken = 1
- 如果指令為jal指令，則分支必須跳轉 → actual\_taken = 1

```
230    assign cond_mispredict = (ap.predict_t & ~actual_taken) |
231    (ap.predict_nt & actual_taken);
```

如果預測分支跳轉 (ap.predict\_t = 1) 但實際未跳轉 (actual\_taken = 0)，或者預測分支不跳轉 (ap.predict\_nt = 1) 但實際跳轉 (actual\_taken = 1)，則分支預測錯誤 (cond\_mispredict = 1)

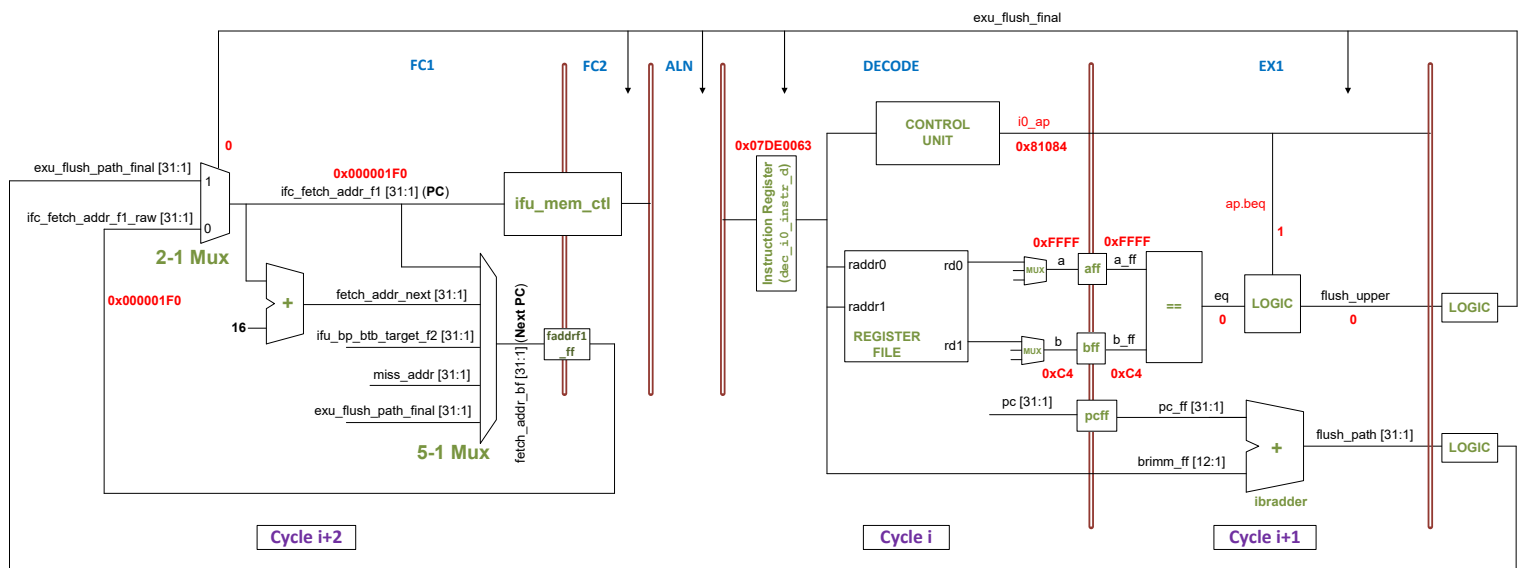
```
237    assign flush_upper = ( ap.jal | cond_mispredict | target_mispredict) & valid_ff & ~flush & ~freeze;
```

如果分支預測錯誤 (cond\_mispredict = 1)、指令有效 (valid\_ff = 1) 並且管線尚未排清或凍結，則必須排清管線。

**任務：**在 Verilog 程式碼中分析訊號 exu\_flush\_final、exu\_flush\_upper\_e2、exu\_i0\_flush\_final 和 exu\_i1\_flush\_final 對 EX1 及其之前各階段 (FC1、FC2、對齊和解碼) 的影響。對於該分析，第 2.B 部分的模擬非常有用，您可以在其中加入所需的訊號。

不提供解答。

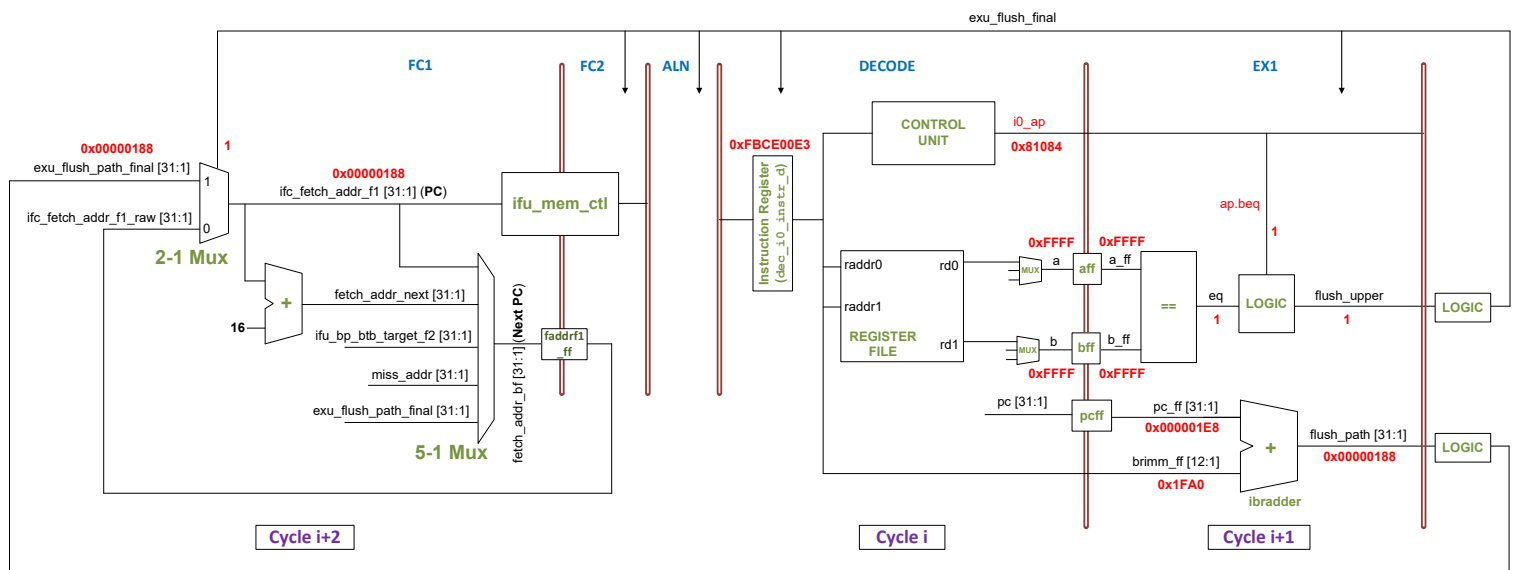
**任務：**修改圖1，將圖3的週期  $i$ 、 $i+1$  和  $i+2$  中所示的每個訊號的值包含在內。



**任務：**修改圖2中的程式，讓第一條分支指令透過轉送的方式取得其輸入運算元。

不提供解答。

**任務：**修改圖1，將圖4的週期  $i$ 、 $i+1$  和  $i+2$  中所示的每個訊號的值包含在內。



**任務：**根據圖2中的範例，檢查不同情況下的訊號，分析FC1中兩個多路開關的操作。

例如，分析在按順序執行指令（即一組沒有分支的指令）的情況下如何完成擷取。在這種情況下，您將看到SweRV EH1處理器進行如下操作：

- 在偶數週期中，使用5:1多路開關選擇fetch\_addr\_next，該多路開關包含的值為目前擷取位址 (ifc\_fetch\_addr\_f1) + 16，因此會讀取下一個連續的128位元指令束（請記住，IS讀取操作提供128位元）。
- 在奇數週期中，使用5:1多路開關選擇ifc\_fetch\_addr\_f1，因此不會擷取新的指令。這樣，每2個週期會擷取4條32位元指令，這與解碼階段所需的擷取速率相同（每個週期2條指令）。

注意，在DDCARV所述的處理器中，每個週期只需將PC的值加4（適用於按順序執行指令的情況），進而在每個週期擷取一條指令。

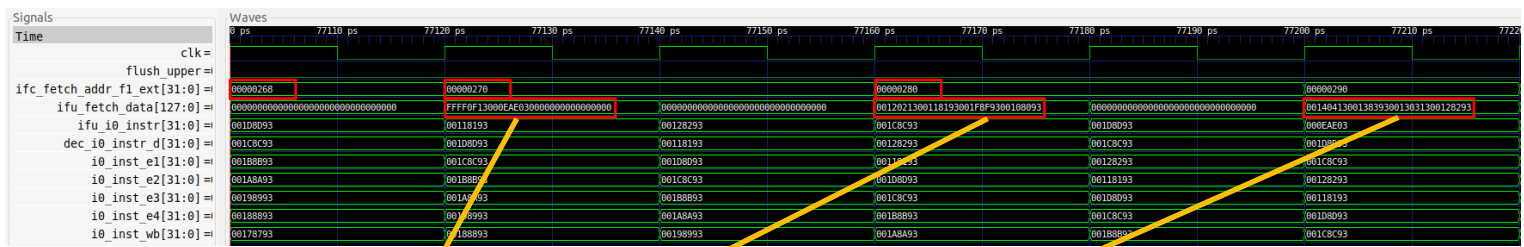
還可以修改圖2中的程式以建立新的場景。例如，可以在跳轉的分支後新增一些A-L指令，查看如何在重新導向後排清這些指令。

## 按順序執行:

使用以下來源：

- 程式路徑：`[RVfpgaPath]/RVfpga/Labs/Lab14/LW_Instruction_ExtMemory`
- Tcl指令碼路徑：`[RVfpgaPath]/RVfpga/Labs/RVfpgaLabsSolutions/Programs_Solutions/Lab16/test_SequentialExecution.tcl`

在Verilator中可得到如下模擬結果：



268:	000eae03	lw	t3,0(t4)
26c:	ffff0f13	addi	t5,t5,-1
270:	00108093	addi	ra,ra,1
274:	001f8f93	addi	t6,t6,1
278:	00118193	addi	gp,gp,1
27c:	00120213	addi	tp,tp,1
280:	00128293	addi	t0,t0,1
284:	00130313	addi	t1,t1,1
288:	00138393	addi	t2,t2,1
28c:	00140413	addi	s0,s0,1

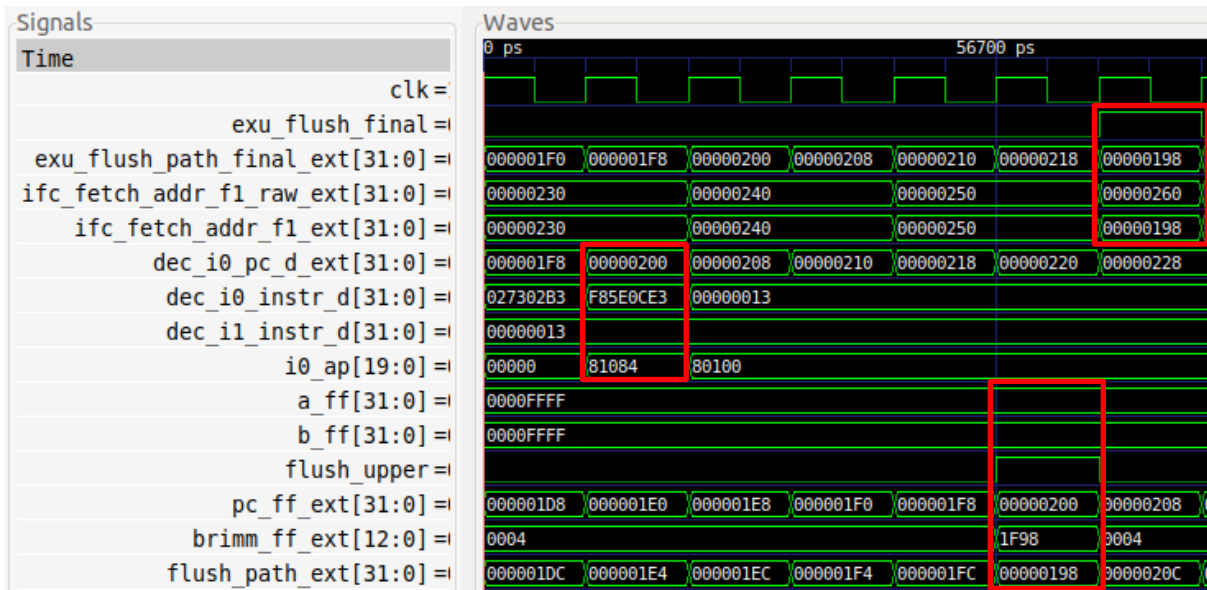
可以看出，每兩個週期將擷取一個新的128位元指令束。

**任務：**在實驗15中，我們已分析過如何在提交階段透過輔助ALU消除寫後讀RAW資料冒險。與該實驗探討的A-L指令類似，如果先前執行過多週期操作，則條件分支指令可能產生寫後讀RAW資料冒險，必須在提交階段消除冒險。如果確定分支預測錯誤，則必須在提交階段排清管線並重新導向。請使用[RVfpgaPath]/RVfpga/Labs/Lab16/BEQ\_Instruction\_HazardCommit資料夾中的程式（對圖2中的程式進行了少許修改）和.tcl檔案分析該情況。

產生的程式碼：

```
00000198 <LOOP>:
198: 001f0f13      addi   t5,t5,1
19c: 00000013      nop
1a0: 00000013      nop
1a4: 00000013      nop
1a8: 00000013      nop
1ac: 00000013      nop
1b0: 00000013      nop
1b4: 00000013      nop
1b8: 07de0463     beq    t3,t4,220 <OUT>
1bc: 00000013      nop
1c0: 00000013      nop
1c4: 00000013      nop
1c8: 00000013      nop
1cc: 00000013      nop
1d0: 00000013      nop
1d4: 00000013      nop
1d8: 001e8e93     addi   t4,t4,1
1dc: 00000013      nop
1e0: 00000013      nop
1e4: 00000013      nop
1e8: 00000013      nop
1ec: 00000013      nop
1f0: 00000013      nop
1f4: 00000013      nop
1f8: 027302b3     mul    t0,t1,t2
1fc: 00000013      nop
200: f85e0ce3     beq    t3,t0,198 <LOOP>
204: 00000013      nop
208: 00000013      nop
20c: 00000013      nop
210: 00000013      nop
214: 00000013      nop
218: 00000013      nop
21c: 00000013      nop
```

## Verilator模擬：



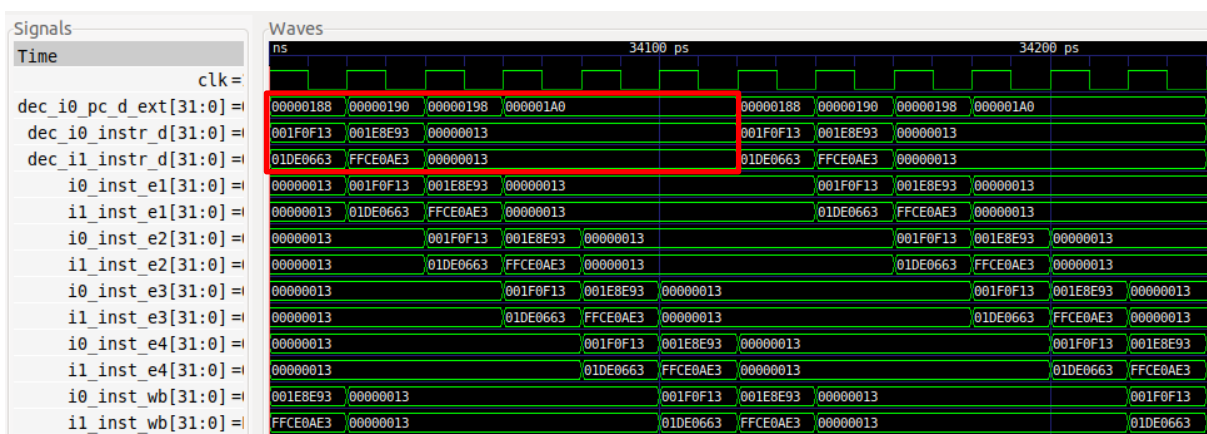
對beq指令（0xf85e0ce3）進行解碼，經過EX1（以錯誤的運算元執行指令）、EX2和EX3階段，最後進入提交階段，該階段將觸發排清和重新導向（flush\_upper = exu\_flush\_final = 1），進而以正確的運算元再次執行指令。

**任務：**在圖2的範例中，刪除所有nop指令並分析模擬。然後透過在開發板上執行程式，用效能計數器計算IPC。

啟用SweRV EH1中使用的分支預測器（方法為註解掉圖2中的兩條初始化指令），並分析開發板上的模擬和執行情況。

比較兩個實驗並解釋結果。

## 簡單分支預測器：



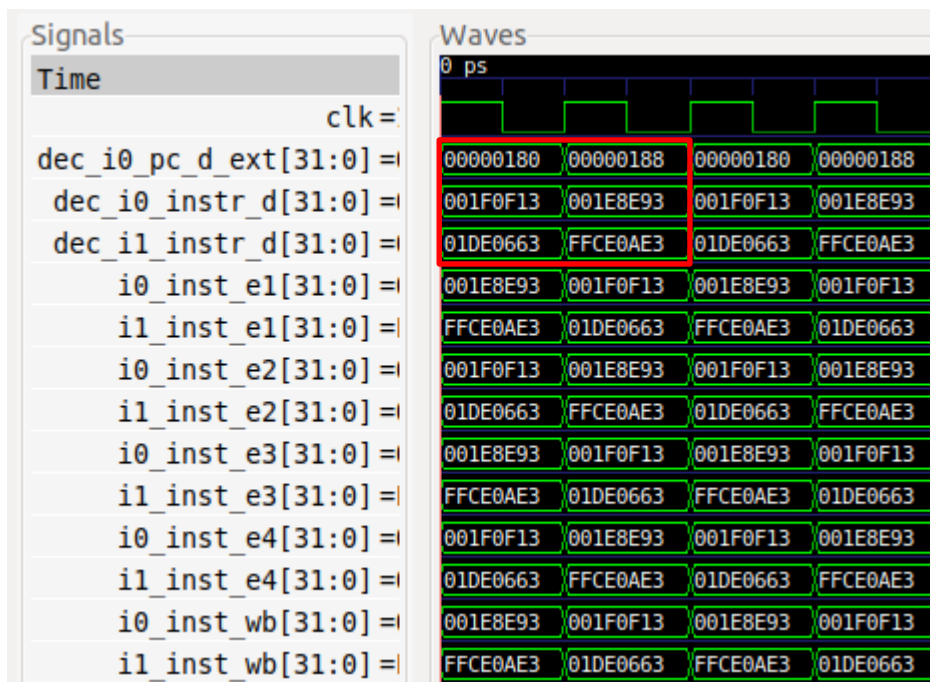
```

PIO Home  C Test.c  Test_Assembly.S x  startup.S
src > Test_Assembly.S
17 Test_Assembly:
18
19 li t2, 0x008           # Disable Branch Predictor
20 csrrs t1, 0x7F9, t2
21 //INSERT_NOPS_2
22
23 li t3, 0xFFFF
24 li t4, 0x1
25 li t5, 0x0
26 li t6, 0x0
27
28 LOOP:
29 add t5, t5, 1
30 beq t3, t4, OUT
31 add t4, t4, 1
32 beq t3, t3, LOOP
33 OUT:
34 INSERT_NOPS_8
35
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL
> Executing task: platformio device monitor <
--- Available filters and text transformations: colorize, debug, c
--- More details at http://bit.ly/pio-monitor-filters
--- Monitor on /dev/ttyUSB1 115200,8,N,1 ---
--- Quit: Ctrl+C | Menu: Ctrl+T | Help: Ctrl+T followed by Ctrl+H
Cycles = 393468
Instructions = 262190

```

$$\text{IPC} = 262 / 393 = 0.67$$

Gshare分支預測器：





```

PIO Home  C Test.c  Test_Assembly.S x  startup.S  File
src > Test_Assembly.S
17 Test_Assembly:
18
19 //li t2, 0x008           # Disable Branch Predictor
20 //csrrs t1, 0x7F9, t2
21
22 li t3, 0xFFFF
23 li t4, 0x1
24 li t5, 0x0
25 li t6, 0x0
26
27 LOOP:
28     add t5, t5, 1
29     beq t3, t4, OUT
30     add t4, t4, 1
31     beq t3, t3, LOOP
32 OUT:
33     INSERT_NOPS_8
34
35 .end

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL
> Executing task: platformio device monitor <
--- Available filters and text transformations: colorize, debug, def
--- More details at http://bit.ly/pio-monitor-filters
--- Miniterm on /dev/ttyUSB1 115200,8,N,1 ---
--- Quit: Ctrl+C | Menu: Ctrl+T | Help: Ctrl+T followed by Ctrl+H --
Cycles = 131322
Instructions = 262188

```

$$\text{IPC} = 262 / 131 = 2$$

使用Gshare BP時，可以得到理想的IPC，但在使用簡單BP時，由於第二個分支指令引發的排清和重新導向，IPC遠遠達不到理想水平。

**任務：**分析上述所有雜湊模組，嘗試瞭解其工作原理及其在Gshare BP結構中的使用方式。

不提供解答。

**任務：**分析如何對這兩個結構進行存取。

不提供解答。

**任務：**分析如何計算5:1多路開關的選擇訊號。

不提供解答。

**任務：**分析如何透過BTB中讀取的值（btb\_rd\_tgt\_f2[11:0]）和FC2中的擷取位址（ifc\_fetch\_addr\_f2[31:4]）取得預測目標位址（ifu\_bp\_btb\_target\_f2）。

模組ifu\_bp\_ctl：



```

1115 // compute target
1116 // Form the fetch group offset based on the btb hit location and the location of the branch within the 4 byte chunk
1117 assign btb_fg_crossing_f2 = btb_sel_f2[0] & btb_rd_pc4_f2;
1118
1119 wire [2:0] btb_sel_f2_enc, btb_sel_f2_enc_shift;
1120 assign btb_sel_f2_enc[2:0] = encode8_3(btb_sel_f2[7:0]);
1121 assign btb_sel_f2_enc_shift[2:0] = encode8_3(1'b0, btb_sel_f2[7:1]);
1122
1123 assign bp_total_branch_offset_f2[3:1] = (((3{ btb_rd_pc4_f2 } & btb_sel_f2_enc_shift[2:0]) |
1124                                         ((3{~btb_rd_pc4_f2} & btb_sel_f2_enc[2:0]) |
1125                                         (3{btb_fg_crossing_f2})));
1126
1127
1128 logic [31:4] adder_pc_in_f2, ifc_fetch_addr_prior;
1129 rvdffe # (28) faddrf2_ff (., .en(ifc_fetch_req_f2 & ~ifu_bp_kill_next_f2 & ic_hit_f2), .din(ifc_fetch_addr_f2[31:4]), .dout(ifc_fetch_addr_prior[31:4]));
1130
1131 assign ifu_bp_poffset_f2[11:0] = btb_rd_tgt_f2[11:0];
1132
1133 assign adder_pc_in_f2[31:4] = ((28{ btb_fg_crossing_f2 } & ifc_fetch_addr_prior[31:4]) |
1134                               ((28{~btb_fg_crossing_f2} & ifc_fetch_addr_f2[31:4]));
1135
1136 logic [31:0] pc_ext = {adder_pc_in_f2[31:4], bp_total_branch_offset_f2[3:1], 1'b0};
1137 logic [12:0] offset_ext = {btb_rd_tgt_f2[11:0], 1'b0};
1138
1139
1140 rvbradder predtgt_addr (., p({adder_pc_in_f2[31:4], bp_total_branch_offset_f2[3:1]}),
1141                        .offset(btb_rd_tgt_f2[11:0]),
1142                        .dout(bp_btb_target_addr_f2[31:1]));
1143
1144 // mux the btb target address here for a predicted return
1145 assign ifu_bp_btb_target_f2[31:1] = btb_rd_ret_f2 & ~btb_rd_call_f2 ? rets_out[0][31:1] : bp_btb_target_addr_f2[31:1];
1146

```

**任務：**分析SweRV EH1處理器中實作的RAS。可透過搜尋網際網路獲得有關該結構的更多操作資訊（例如[http://www-classes.usc.edu/engr/ee-s/457/EE457\\_Classnotes/ee457\\_Branch\\_Prediction/EE560\\_05\\_Ras\\_Just\\_FYI.pdf](http://www-classes.usc.edu/engr/ee-s/457/EE457_Classnotes/ee457_Branch_Prediction/EE560_05_Ras_Just_FYI.pdf)）。

不提供解答。

**任務：**分析如何更新全域歷史記錄暫存器。

不提供解答。

## 練習

1) 實作一個雙模分支預測器，並將其效能與Gshare BP的效能進行比較。

不提供解答。

2) （以下練習基於《電腦組織結構和設計》（RISC-V版本，Patterson & Hennessy ([HePa])）中的練習4.25。）

請看下面的迴圈：

```

LOOP: lw x10, 0(x13)
      lw x11, 4(x13)
      add x12, x10, x11
      add x13, x13, -8
      bnez x12, LOOP

```

假設採用了完美分支預測（在SweRV EH1中，只需避免使用第一次迭代即可模擬該行為），管線具有完備的轉送支援（同樣是在SweRV EH1中），並且可在EX1階段確定分支執行情況。

a. 展示該迴圈的第二次和第三次迭代的模擬結果。解釋出現的行為。可使用[RVfpgaPath]/RVfpga/Labs/Lab16/HePa\_Exercise-4-25中提供的程式。

