



IMAGINATION大學計劃

# RVfpga實驗16

控制冒險：分支指令

## 1. 簡介

在本實驗中，我們將完成冒險分析。在之前兩個實驗中，我們研究了SweRV EH1處理器存在的**結構冒險**和**資料冒險**，本實驗將專門分析**控制冒險**。正如S.Harris和D.Harris在《數位設計和電腦體系結構》（RISC-V版本，簡稱DDCARV）中所述，當未能在規定時間內決定下一條要擷取的指令時，就會發生**控制冒險**。

**附註：**在分析SweRV EH1控制冒險邏輯之前，建議使用者先閱讀DDCARV第7.5節，瞭解如何在管線處理器中執行beq指令和解除控制冒險。有關控制冒險的具體章節為第7.5.3節。另外，在開始本實驗的第3部分之前，建議使用者先閱讀第7.7.3節有關分支預測的內容。

控制冒險是由分支和跳轉指令引起的，因為這些指令必須計算下一個擷取的位址。分支指令還必須計算是否要發生分支。相比之下，對於所有其他類型的指令，下一個擷取的位址均為PC + 4，無需計算。

一些處理器永遠不會發生控制冒險。舉例來說，在處理器中，如果指定指令在下一次擷取前便完全執行，則不會發生控制冒險。DDCARV中的單週期和多週期處理器均符合這一條件。具體而言，由於分支指令能夠完全執行，因此在下一次擷取前，處理器便能夠確定是否發生分支並確定下一條要擷取的指令。與之相反，管線處理器在做出這些決定之前便要進行下一次擷取。

控制冒險的一種處理方法是暫停管線，直至確定分支後要擷取的指令。由於決定是在SweRV EH1的EX1階段做出的（將在第2部分進行介紹），因此，管線在每條分支處必須暫停四個週期（參見實驗11的圖1中所示的管線）。在實際程式中，分支往往會頻繁出現，導致系統效能嚴重降低，因此SweRV EH1中並未實作此解決方案。

另一種方法是預測是否會發生分支，並從預測的路徑中擷取。分支確定後，如果與預測相反，處理器可以排清擷取的指令（在這種情況下，必須承擔分支預測錯誤的損失）；如果與預測相同，則繼續執行擷取的指令（在這種情況下不存在效能損失）。在本實驗中，我們分析了SweRV EH1提供的兩種分支預測器（Branch Predictor, BP）：一種是**簡單分支預測器**，該預測器始終假定不會發生分支，雖然無需花費硬體成本，但會導致系統效能低下；另一種是**Gshare分支預測器**，該預測器需要花費額外的硬體成本，能夠提供更高的效能。

在第2部分中，我們將介紹如何在SweRV中執行beq指令，然後會使用簡單BP進行一些範例模擬（採用DDCARV等教材中假定的典型場景）。然後，在第3部分，我們將展示SweRV EH1如何使用Gshare分支預測器更高效地處理控制冒險。

## 2. beq指令執行和PC計算

在本部分中，我們將分析SweRV EH1中beq指令的執行情況。首先，在第2.A部分，我們將介紹如何在EX1階段執行beq指令，以及如何在FC1階段計算擷取位址和下次擷取位址（對實驗11的第2.B.i部分中有關FC1階段的說明進行補充）。隨附的圖示（圖1）和大多數描述適用於所有指令，但我們重點關注beq指令在使用簡單BP的處理器上的執行情況，採用該組態時，預測結果始終為不發生分支（類似於DDCARV或PaHe中設定的情景）。在接下來的第2.B部分，我們會進行一些實驗，透過實例來闡明相關概念。對於這些實驗，我們同樣不會使用真正的分支預測器，而是將所有條件分支預測為不發生（即採用所謂的簡單BP）。

### A. 理論說明

圖1所示為FC1階段中用於確定擷取位址（程式計數器（Program Counter，PC）中的值，在DDCARV中定義為保存目前指令記憶體位址的暫存器）和下次擷取位址（用於在每個週期結束時更新PC的值）的主要結構。圖中還展示了在EX1階段執行beq指令所需的結構（圖中所示的大多數硬體也用於執行其他分支指令）。與其他實驗一樣，圖中所示的訊號名稱均為SweRV EH1處理器的Verilog模組所使用的實際名稱。

#### i. 擷取位址計算

如圖1所示，FC1階段包含兩個多路開關：一個是2:1多路開關，用於在ifc\_fetch\_addr\_f1[31:1]中產生擷取位址；另一個是5:1多路開關，用於計算下次擷取位址並將該位址放入訊號fetch\_addr\_bf[31:1]。

- **2:1多路開關**：產生訊號ifc\_fetch\_address\_f1[31:1]，即目前週期中擷取的指令的記憶體位址，正如實驗11的圖3中的分析結果，該位址將提供給記憶體控制器，用於從指令快取中讀取128位元指令束。該多路開關的兩個輸入為：
  - 分支目標位址（exu\_flush\_path\_final[31:1]），在EX1階段計算，相關分析如下文所述。
  - 下次擷取位址（ifc\_fetch\_addr\_f1\_raw[31:1]），在上一個週期中計算並暫存，用作本階段中5:1多路開關的輸出，相關分析如下文所述（fetch\_addr\_bf[31:1]）。

該多路開關的控制訊號稱為exu\_flush\_final，在執行階段提供。如果必須從分支目標位址擷取，則exu\_flush\_final = 1，使用exu\_flush\_path\_final[31:1]作為擷取位址；否則exu\_flush\_final = 0，使用ifc\_fetch\_addr\_f1\_raw[31:1]作為擷取位址。

請注意，DDCARV中所述的處理器使用類似的2:1多路開關在每個週期更新PC。

- **5:1多路開關**：產生訊號`fetch_addr_bf[31:1]`，位址來自以下五個來源之一：
  - 擷取位址 (`ifc_fetch_addr_f1`)，用於PC在下一週期保持不變的情況。
  - 序列中的下一個位址 (`fetch_addr_next`)，計算方式為擷取位址 (`ifc_fetch_addr_f1`) + 16，指向下一個128位元指令束。
  - 分支目標緩衝區 (`ifc_bp_btb_target_f2`) 預測的位址，分支目標緩衝區是分支預測器的主要結構之一，如果預測結果為發生分支，則該位址將用作擷取位址。
  - 分別與`miss path`和`flush path`相對應的另外兩個輸入訊號 (`miss_addr`和`exu_flush_path_final`)，但我們在本實驗中不會分析這些訊號。

該多路開關提供的訊號 (`fetch_addr_bf[31:1]`) 將被暫存，並在下一個週期中用作上述2:1多路開關的輸入。

請注意，DDCARV所述的處理器設計更為簡單，不包含此5:1多路開關。

## ii. `beq`指令的執行

條件分支必須計算分支目標位址，並測試是否滿足條件。具體以SweRV EH1為例（參見圖1）：

- **分支目標位址計算**：EX1使用一個新的加法器來計算分支目標位址，並將位址放入訊號`flush_path[31:1]`中。該訊號將透過一些邏輯和暫存器，作為輸入 (`exu_flush_path_final[31:1]`) 提供給FC1中的2:1多路開關。
- **條件解析**：EX1使用`exu_alu_ctl`模組內部的一個新模組來檢查兩個運算元是否相等 (`eq = 1`表示相等，`eq = 0`表示不相等)。系統會根據`eq`訊號（以及一些其他訊號，如您將在擬定的任務中分析的`ap.beq`訊號）計算`flush_upper`和`exu_flush_final`訊號，並將兩個訊號提供給FC1階段，後者將用作2:1多路開關的控制訊號。分支預測錯誤時，該控制訊號 (`exu_flush_final`) 為1，其他情況下則為0。

具體地說，使用`beq`指令以及上文所述的簡單BP（預測所有分支均不發生）時，如果分支的兩個運算元不相等，則不得發生分支，此時預測正確：`exu_flush_final = flush_upper = eq = 0`。在這種情況下，處理器可以繼續按順序擷取並執行指令，不會出現效能損失。我們將在第2.B.i部分分析該情況。

相反的，如果兩個運算元相等，則必須發生分支，對於預測不發生分支的簡單BP而言，其預測是錯誤的：`exu_flush_final = flush_upper = eq = 1`。在這種情況下，如下文第2.B.ii部分所述，SweRV EH1管線中將觸發以下操作（參見圖1）。

- 當 `exu_flush_final = 1` 時，透過選擇 **FC1** 中 **2:1** 多路開關的輸入 **1**（其中包含在上述 **EX1** 階段中計算的分支目標位址）（`ifc_fetch_addr_f1[31:1] = exu_flush_path_final[31:1]`），擷取位址將重新導向到分支的目標位址。
- **EX1** 之前的管線階段將被排清。為此，將向先前的階段提供多個訊號（`exu_flush_final`、`exu_flush_upper_e2`、`exu_i0_flush_final` 和 `exu_i1_flush_final`）（圖1中未標明這些訊號的用途）。

**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。

**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`。

**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。

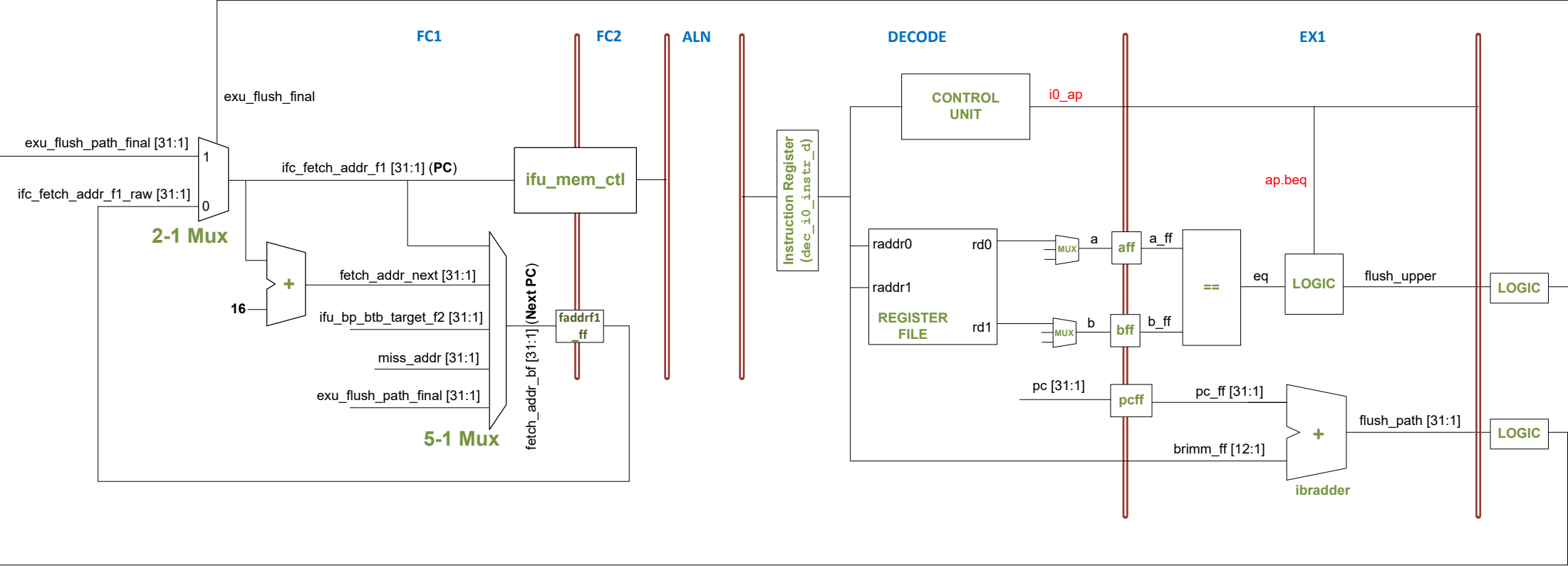


圖1. 透過SweRV EH1執行的beq指令的高階檢視

## B. 實驗

我們已經介紹了在EX1階段執行beq指令以及在FC1階段計算擷取位址和下次擷取位址時涉及的主要概念，接下來我們將進行一些模擬展示，以加強對於這些概念的理解。

在本部分中，我們將使用圖2中所示的範例，該範例會執行一個重複0xFFFF迭代的迴圈（十進位形式為65,535），並包含兩條beq指令：第一條beq指令始終不發生（迴圈的最後一次迭代除外），第二條指令將始終發生。與往常一樣，我們要分析的指令（在本例中，beq指令以紅色強調顯示）使用多個nop圈住，以便與前面和後面的指令分隔開來。資料夾

[RVfpgaPath]/RVfpga/Labs/Lab16/BEQ\_Instruction中提供PlatformIO專案，可根據需要對其進行分析、模擬和修改。

```
Test_Assembly:

li t2, 0x008                                # Disable Branch Predictor
csrrs t1, 0x7F9, t2

li t3, 0xFFFF
li t4, 0x1
li t5, 0x0
li t6, 0x0

LOOP:
    add t5, t5, 1
    INSERT_NOPS_7
    beq t3, t4, OUT
    INSERT_NOPS_7
    add t4, t4, 1
    INSERT_NOPS_7
    beq t3, t3, LOOP
    INSERT_NOPS_7
OUT:
    INSERT_NOPS_8

.end
```

圖2. 包含beq指令的程式

在實驗中，我們停用壓縮指令的使用。此外，如上所述，在本部分中，SweRV EH1中提供的Gshare分支預測器被停用，並且分支預測結果始終為不發生（簡單BP）。上述設置需透過加入兩條指令實現，這兩條指令允許使用者在執行期間配置處理器。如實驗11的附錄B所述，必須在程式碼中包含以下兩條指令，進而停用分支預測器並預測不發生每條分支。

```
li t2, 0x008
csrrs t1, 0x7F9, t2
```

在該組態中，程式（圖2）中的第一條分支始終能夠預測正確（迴圈的最後一次迭代除外，我們暫不分析該情況），而第二條分支始終會預測錯誤，這將導致前面四個階段被排清並重新導向。接下來，我們將分析兩條beq指令的執行。

### i. 執行第一條分支：`beq t3, t4, OUT`

在本部分中，我們將分析圖2中第一條分支指令的執行情況，該分支始終能夠預測正確（迴圈的最後一次迭代除外，我們暫不分析該情況）。在PlatformIO中開啟、編譯專案，然後開啟反組譯檔案（位於

`[RVfpgaPath]/RVfpga/Labs/Lab16/BEQ_Instruction/.pio/build/swervolf_nexys/firmware.dis`）。

請注意，第一條`beq`指令位於位址`0x000001a8`處：

```
0x000001a8:      07de0063      beq    t3,t4,208 <OUT>
```

接下來，我們按照GSG中的說明，在Verilator中模擬圖2所示的程式，然後在GTKWave上開啟模擬器產生的波形檔案。圖3為迴圈中隨機某次迭代的放大圖示（應避免採用第一次迭代，因為該次迭代包含`!$`未命中，會增加分析難度，也應避免最後一次迭代，該次迭代將預測錯誤），其中重點展示第一條`beq`指令的執行。


圖中包含的大多數訊號為圖1的框圖中所示的訊號。但必須考慮到，為了清楚起見，在模擬時，包含指令位址的訊號（帶有字尾`_ext`）均在右邊增加了一個數值為0的位元（注意，Verilog程式碼中原始的未延伸訊號不包括最低有效位元，因為該位元始終為0）；具體訊號為：

Verilog程式碼： <code>exu_flush_path_final[31:1]</code>	→	模擬： <code>exu_flush_path_final_ext[31:0]</code>
Verilog程式碼： <code>ifc_fetch_addr_f1_raw[31:1]</code>	→	模擬： <code>ifc_fetch_addr_f1_raw_ext[31:0]</code>
Verilog程式碼： <code>ifc_fetch_addr_f1[31:1]</code>	→	模擬： <code>ifc_fetch_addr_f1_ext[31:0]</code>
Verilog程式碼： <code>pc_ff[31:1]</code>	→	模擬： <code>pc_ff_ext[31:0]</code>
Verilog程式碼： <code>brim_ff[12:1]</code>	→	模擬： <code>brim_ff_ext[12:0]</code>
Verilog程式碼： <code>flush_path[31:1]</code>	→	模擬： <code>flush_path_ext[31:0]</code>

該專案隨附檔案`test_1.tcl`。要在GTKWave中使用該檔案，請按一下「**File**」（檔案）→

「**Read Tcl Script File**」（讀取Tcl指令碼檔案），並開啟檔案

`[RVfpgaPath]/RVfpga/Labs/Lab13/BEQ_Instruction/test_1.tcl`。然後按幾次「**Zoom In**」（放

大）（）移動至迴圈的任意一次迭代（第一次或最後一次除外）。可以看到兩條`beq`指令的執行情況；圖3所示為執行第一條分支指令時應觀察到的內容。



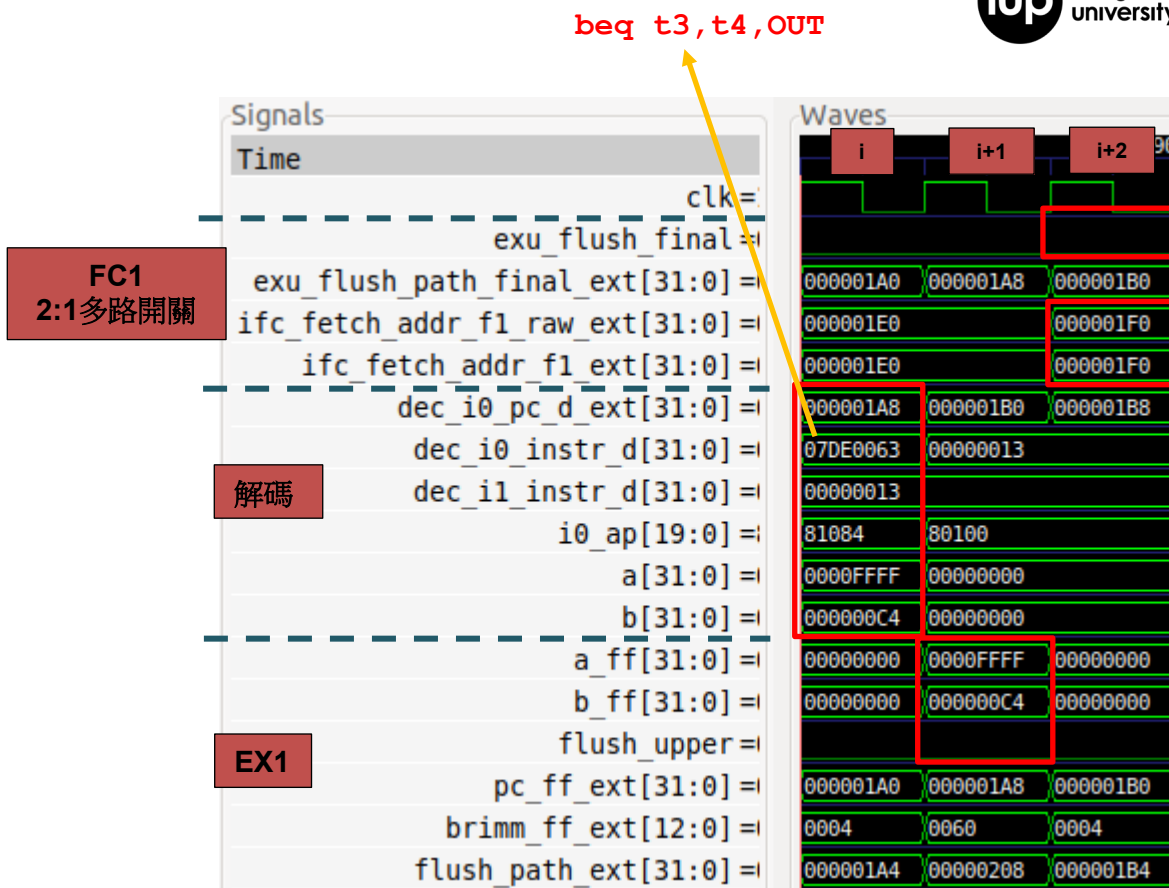


圖3. 執行圖2中第一條beq指令時的Verilator模擬

同時分析圖3中的波形以及圖1中的圖。圖3顯示了三個連續的週期：beq的解碼階段（週期*i*）、beq的EX1階段（週期*i+1*）、在解析beq後的FC1階段選擇下一PC（週期*i+2*）。

- **週期*i* - beq指令的解碼階段**：訊號dec\_i0\_pc\_d\_ext包含解碼階段中指令的位址（位於路徑0中），第一個beq的位址為0x000001A8，訊號dec\_i0\_instr\_d（在教材中通常稱為指令暫存器（Instruction Register，IR））包含32位元機器指令，第一個beq的位址為0x07DE0063（二進位形式為：0000 0111 1101 1110 0000 0000 0110 0011）。

在RISC-V中，beq指令的操作碼如下（參見[DDCARV]的附錄B）：

imm<sub>12,10:5</sub> | rs2 | rs1 | 000 | imm<sub>4,1,11</sub> | 1100011

因此，您可以驗證0x07DE0063對應於：beq t3, t4, OUT（imm<sub>12:0</sub> = 0x060）。請記住，立即數提供了目標位址目前PC的偏移量。目標位址（由標籤「OUT:」指示）是目前PC（即beq t3, t4, OUT）之後的24條指令（7條nop + 1條add + 7條nop + 1條beq + 7條nop + 1條nop = 24條指令）。目前PC之後共有24\*4 = 96（0x60）個位元組。

在此階段，將產生管線控制訊號。對於第一條beq指令，i0\_ap（對於該指令，其值為0x81084 – 參見SweRVref.docx）的以下位置1：

- o valid：指示該指令為使用ALU的有效指令。
- o beq：指示該指令為如相等則分支指令。

- `sub` : 指示ALU必須執行減法操作。一些分支指令會使用減法運算的結果來進行比較（然而，如下文所示，`beq`的情況並非如此）。
- `predict_nt` : 指示分支預測結果為不發生。

此外，將讀取暫存器檔案並將分支指令傳送至I0管道。訊號a和b（本例中分別為0xFFFF和0xC4）中包含下一階段所使用的比較器的輸入，在本例中，輸入為從暫存器檔案讀取的值（在其他情況下，運算元可透過轉送的形式提供，如實驗15的分析所示）。

- **週期*i+1* - `beq`指令的EX1階段**：在下一週期中，將執行`beq`指令。比較訊號a\_ff和b\_ff。如果兩個數（0xFFFF和0xC4）不同，則不發生分支。如上文所述，在此組態中，所有分支的預測結果均為不發生（`i0_ap.predict_nt = 1`）。因此，分支預測結果正確（`flush_upper = 0`），可以正常執行。
- **週期*i+2* - FC1 階段**：在下一週期中，假設已預測分支並確定不發生分支，則正常按順序擷取。在圖3中，應注意`exu_flush_final = 0`，`ifc_fetch_addr_f1_ext[31:0] = ifc_fetch_addr_f1_raw_ext[31:0] = 0x000001F0`。該位址指向下一個連續的128位元指令束。可以看出，在前兩個週期中，之前的128位元指令束已擷取（`ifc_fetch_addr_f1_ext[31:0] = 0x000001E0`）。

**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形式擷取其輸入運算元。

## ii. 第二條分支的執行：`beq t3, t3, OUT`

接下來我們分析第二條分支，系統總是錯誤預測為不發生該分支，但實際會發生該分支。開啟反組譯檔案（位於

`[RVfpgaPath]/RVfpga/Labs/Lab16/BEQ_Instruction/.pio/build/swervolf_nexys/firmware.dis`）

。請注意，第二條`beq`指令位於位址0x000001E8處：

```
0x000001e8:      fbce00e3          beq    t3,t3,188 <LOOP>
```

圖4所示為迴圈中隨機某次迭代期間的訊號（應避免採用第一次迭代，因為該次迭代包含指令快取（I\$）未命中）。

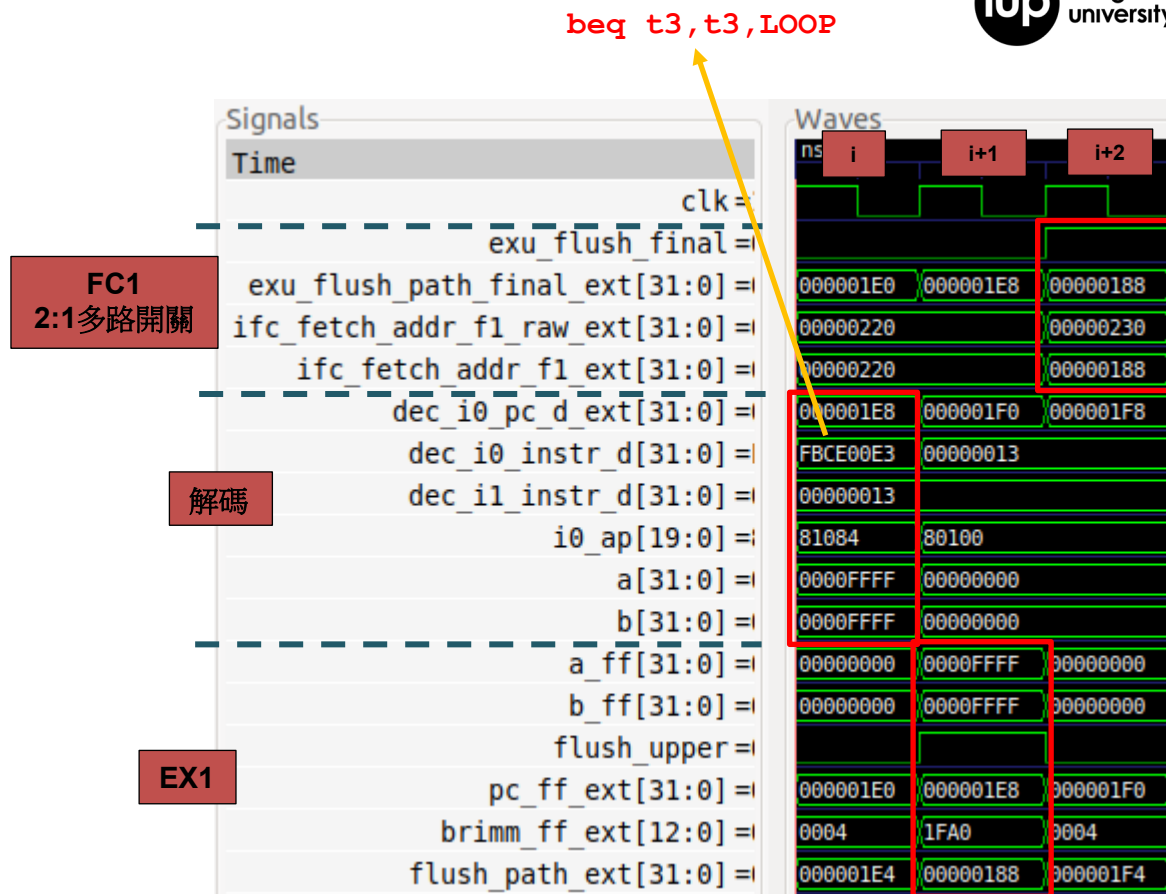


圖4. 圖2的範例中第二條分支的Verilator模擬

同時分析圖4中的波形以及圖1中的圖。以紅色強調顯示的值表示執行第二條beq指令期間三個連續的週期：beq的解碼階段（週期*i*）、beq的EX1階段（週期*i+1*）、在解析beq後的FC1階段選擇下一PC（週期*i+2*）。

- **週期*i*** - beq指令的解碼階段：PC（訊號dec\_i0\_pc\_d\_ext）為0x000001E8，指令（訊號dec\_i0\_instr\_d）為0xFBCE00E3（二進位形式為：1111 1011 1100 1110 0000 0000 1110 0011）。

在RISC-V中，beq指令的操作碼如下（參見[DDCARV]的附錄B）：

imm<sub>12,10:5</sub> | rs2 | rs1 | 000 | imm<sub>4:1,11</sub> | 1100011

因此，您可以驗證0xFBCE00E3對應於：beq t3, t3, LOOP（Immediate<sub>12:0</sub> = 0x1FA0）。請記住，立即數提供了目標位址目前PC的偏移量。目標位址（由標籤「LOOP:」指示）是目前PC（即beq t3, t3, LOOP）之前的24條指令（7條nop + 1條add + 7條nop + 1條beq + 7條nop + 1條add = 24條指令）。目前PC之前共有24\*4 = 96個位元組。因此，立即數編碼為-96，即0x1FA0，以13位元二進位補碼形式寫入。

在此階段，將產生管線控制訊號。此beq指令的控制訊號與第一條beq指令的控制訊號相同（參見上一部分）。

此外，將讀取暫存器檔案並將分支指令傳送至I0管道。訊號a和b（均為0xFFFF）中包含下一階段所使用的比較器的輸入，在本例中，輸入為從暫存器檔案讀取的值。

- **週期*i*+1 - beq指令的EX1階段**：在下一週期中，將執行beq指令。在一側比較a\_ff和b\_ff訊號。如果兩個值相同，則必須發生分支。然而，如上文所述，在我們的組態中，所有分支的預測結果均為不發生（i0\_ap.predict\_nt = 1）。也就是說，分支預測結果錯誤（flush\_upper = 1）。因此，必須從分支目標位址擷取，並且必須排清初始管線階段。

在此階段，目標位址的計算方法為將pc\_ff\_ext (0x1E8) 與brim\_ff\_ext (0x1FA0) 相加。結果將放入訊號flush\_path\_ext (0x00000188)。

- **週期*i*+2 - FC1階段**：在下一週期中，指令必須在分支目標位址處繼續執行。如圖4中所示，exu\_flush\_final = 1, ifc\_fetch\_addr\_f1\_ext = exu\_flush\_path\_final\_ext = 0x00000188。該位址對應於分支目標位址，即迴圈的第一條指令的位址（請注意，這是一個向後分支）。

**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，可以在已發生的分支後新增一些A-L指令，查看如何在重新導向後排清這些指令。

**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解除寫後讀資料冒險。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RAW資料冒險，必須在提交階段解除冒險。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。

### 3. SweRV EH1使用的Gshare分支預測器

在第2部分，我們已討論了僅包含一個簡單分支預測器（預測結果始終為不發生）的SweRV EH1組態，本部分將繼續分析SweRV EH1中提供的Gshare分支預測器的操作。Gshare BP會對每條分支指令執行更聰明的預測，能夠提高效能，但需要使用額外的硬體。在描述Gshare BP如何在SweRV EH1中工作之前，我們先比較兩個BP的效能。

**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。

**附註：**Scott McFarling曾在1993年發表過一篇題為「Combining Branch Predictors」的經典論文（<https://www.hpl.hp.com/techreports/Compaq-DEC/WRL-TN-36.pdf>）。該論文的第7章介紹了Gshare分支預測器的工作原理。也可搜尋其他文件，如<https://people.engr.ncsu.edu/efg/521/f02/common/lectures/notes/lec16.pdf>。我們建議您在開始本部分之前先閱讀這些資料，瞭解Gshare BP的工作原理。

圖5所示為SweRV EH1中可用的Gshare BP的簡化檢視。所有BP結構均在模組ifu\_bp\_ctl（位於檔案[RVfpgaPath]/RVfpga/src/SweRVolfSoC/SweRVEh1CoreComplex/ifu/ifu\_bp\_ctl.sv中）內部實作。圖中，與Gshare BP相關的結構由藍色虛線框圈出。

該BP由分支歷史記錄表（Branch History Table，BHT）和分支目標緩衝區（Branch Target Buffer，BTB）組成，前者預測分支的方向（發生或不發生），後者預測發生分支時的目標位址。在預設組態中，BHT包含128個2位元項目。可在模組ifu\_bp\_ctl的第1615-1705行查看BHT。在預設組態中，BTB包含32個13位元項目。可在模組ifu\_bp\_ctl的第1439-1613行查看BTB。

預測分支時，每個週期中將發生以下事件（參見圖5）：

1. 透過模組ifu\_bp\_ctl內的幾個雜湊模組傳遞擷取位址（ifc\_fetch\_addr\_f1 [31:1]）和一些其他訊號：flhash、rdtagf1和fghrns等。所有這些雜湊模組均在檔案[RVfpgaPath]/RVfpga/src/SweRVolfSoC/SweRVEh1CoreComplex/beh/beh\_lib.sv中實作，具體將使用[RVfpgaPath]/RVfpga/src/SweRVolfSoC/SweRVEh1CoreComplex/include/common\_defines.vh中定義的巨集。



例如，可以看到fghrns模組會接收來自擷取位址

(f1hash(.pc(ifc\_fetch\_addr\_f1[31:1]),.hash(btb\_rd\_addr\_f1[RV\_BTBD\_ADDR\_HI:RV\_BTBD\_ADDR\_LO])))和fghr\_ns(全域歷史記錄暫存器)的雜湊的btb\_rd\_addr\_f1訊號，並輸出bht\_rd\_addr\_hashed\_f1訊號。

```
rvbtb_ghr_hash fghrns (.hashin(btb_rd_addr_f1[RV_BTBD_ADDR_HI:RV_BTBD_ADDR_LO]),.ghr(fghr_ns[RV_BHT_GHR_RANGE]),.hash(bht_rd_addr_hashed_f1[RV_BHT_ADDR_HI:RV_BHT_ADDR_LO]));
```

該訊號用於存取Gshare分支預測器的BHT表。

**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。

2. 使用所有這些雜湊訊號(btb\_rd\_addr\_f1、bht\_rd\_addr\_hashed\_f1和fetch\_rd\_tag\_f1等)存取Gshare BP的兩個主要結構：BHT和BTB。

**任務：**分析如何對這兩個結構進行存取。

3. 存取BHT後，可在訊號ifu\_bp\_kill\_next\_f2中取得方向預測結果，如果預測不發生分支，則訊號值為0；如果預測發生分支，則訊號值為1。使用該訊號與其他訊號(在此不具體介紹)共同計算FC1中5:1多路開關的控制訊號。

**任務：**分析如何計算5:1多路開關的選擇訊號。

4. 存取BTB後，可在訊號ifu\_bp\_btb\_target\_f2 [31:1]的加法器中取得所發生分支的預測目標位址。(注意，如果對ret指令進行預測，預測位址也可以來自傳回位址堆疊(Return Address Stack, RAS)。)該訊號為FC1中5:1多路開關的輸入之一。

**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)。

**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))。

5. 在FC1的5:1多路開關中，如果ifu\_bp\_kill\_next\_f2 = 1，則預測目標位址將用作下次擷取位址：fetch\_addr\_bf [31:1] = ifu\_bp\_btb\_target\_f2 [31:1](需確保未排清管線)。相反的，如果ifu\_bp\_kill\_next\_f2 = 0，則使用其他四個輸入之一作為下次擷取位址。

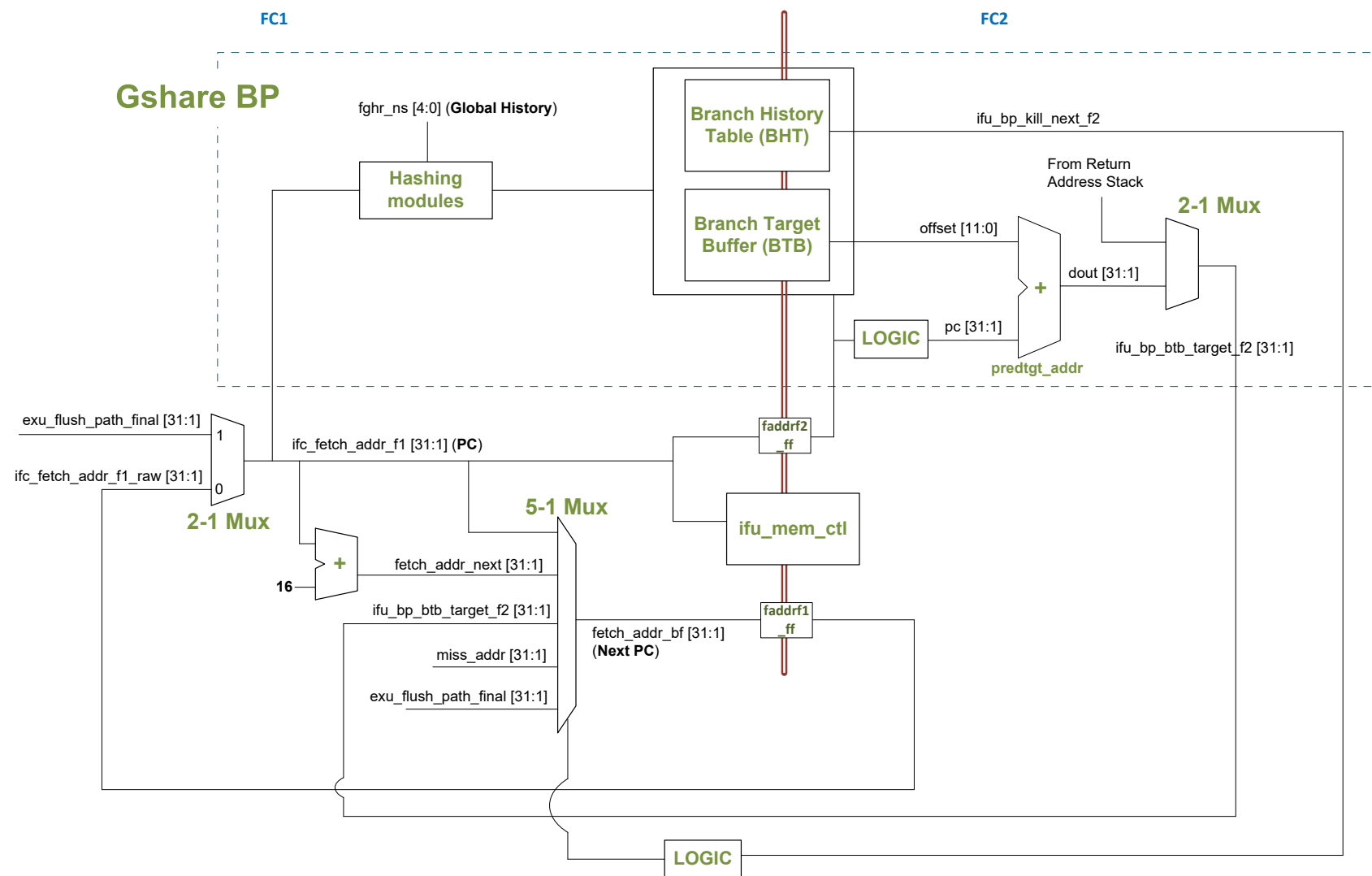



圖5. SweRV EH1中提供的Gshare分支預測器的主要結構（由藍色虛線框圈出）

在本部分中，我們將繼續使用圖2中的範例程式碼。本部分唯一的不同是，我們使用兩條nop指令替換原有的兩條指令（原有的指令會停用Gshare BP），以啟用Gshare分支預測器（插入兩條nop指令是為了確保指令位址與上一部分相同）。

接下來分析程式中第二條分支指令的執行情況，如第2.B.ii部分所述。請記住，在我們的程式中，第二條beq指令位於位址0x000001E8處，這意味著該指令包含在位址範圍0x1E0-0x1EF中所對映的128位原指令束中：

**0x000001e8:           fbce00e3                   beq    t3,t3,188 <LOOP>**

圖6為迴圈中隨機某次迭代的放大圖示。與往常一樣，應避免採用第一次迭代，因為該次迭代包含I\$未命中。此外，對於此分支指令的第一次迭代，預測將會出錯。圖中包含的大多數訊號為圖5中所示的訊號。該專案隨附檔案test\_1\_BP.tcl。要在GTKWave中使用該檔案，請按一下「File」（檔案）→「Read Tcl Script File」（讀取Tcl指令碼檔案），並開啟檔案[RVfpgaPath]/RVfpga/Labs/Lab16/BEQ\_Instruction/test\_1\_BP.tcl。然後按幾次「Zoom In」

（放大）（）移動至迴圈的任意一次迭代（第一次除外）。可以看到兩條beq指令的執行情況；圖6所示為執行第二條分支指令時應觀察到的內容。

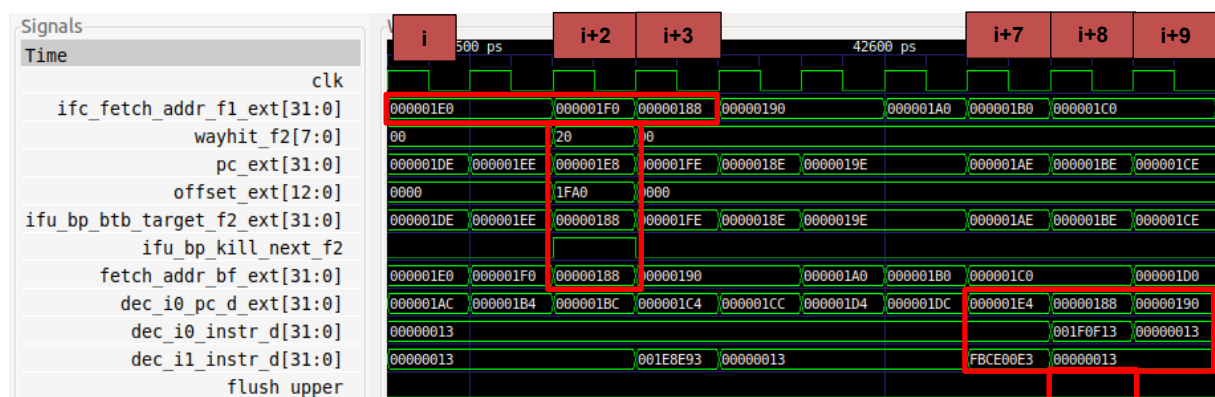


圖6. 圖2中範例的Verilator模擬

同時分析圖6中的波形以及圖5中的圖。以紅色強調顯示的值對應於第二條beq指令，因為該指令遍歷管線階段。

- **週期i**：包含第二條分支的指令束的位址將提供給指令快取：  
ifc\_fetch\_addr\_f1\_ext = 0x000001E0。系統使用該位址讀取分支目標緩衝區（BTB）。
- **週期i+2**：BTB中發生命中：wayhit\_f2 = 0x20（該訊號不包含在圖5中，當訊號不為零時，表示發生命中）。將分支位址（pc\_ext = 0x000001E8）與BTB提供的偏移量（offset\_ext = 0x1FA0，該值為負值）相加，即可得到預測目標位址（ifu\_bp\_btb\_target\_f2\_ext = 0x00000188）。如果BHT預測將發生分支（ifu\_bp\_kill\_next\_f2 = 1），則預測目標位址將用作下次擷取PC（fetch\_addr\_bf\_ext = 0x00000188）。



- **週期*i*+3**：擷取位址為上一週期中計算的分支預測目標位址：  
`ifc_fetch_addr_fl_ext = 0x00000188`。
- **週期*i*+7**：在通路1中對分支進行解碼（`dec_i1_instr_d = 0xFBCE00E3`）。
- **週期*i*+8**：執行分支。預測正確，因此無需觸發排清操作（`flush_upper = 0`）。
- **週期*i*+9**：如果預測正確，將透過分支目標位址正常繼續執行分支。

**任務**：解釋如何在模組 `ifu_bp_ctl` 處更新全域歷史記錄暫存器。

## 4. 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。
- 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。