HW2 REPORT

謝翔丞, 109550025

***Abstract*—本篇主要以兩個部分實行，首先是針對 branch history table 進行分析，嘗試不同 entry size number 以及 branch predict unit 的有無對於程式執行之影響，同時針對不同 branch 狀況也進行分析統計; 第二部分是自行實作一個 two-level predictor ，並嘗試利用此 predictor 來優化整體的 performance。**

Keywords—branch predictor , BHT, branch history table (keywords)

I. INTRODUCTION

本次作業可拆分為兩階段，第一階段先針對現有 branch predict unit 進行嘗試開啟/關閉以及調整不同 entry size number (也就是 branch predict table 大小) 並觀察、分析現象以及不同 type 的 branch 之 hit / miss rate; 第二階段則是自行實作出一個 two-level 的 branch predictor ，嘗試以此實現整體效能的優化.

II. BRANCH ANALYSIS 前置作業與構思

A. 初步想法

觀察不同 entry size 導致的 performance 變化，我打算利用 coremark 執行後計算出的 Iteration / sec 來做比較，照我初步的推論，執行 performance 應會隨著 entry size 下降而隨之下降，因為這表示 table 能儲存的組數越少，也就代表預測上更容易遇到 table 上不存在的案例，造成需要不斷置換 table content、耗費操作及時間;反之也會隨著 entry size 上升而促使 performance 優化，但是達到某個上限之後表現會趨緩甚至可能持平，而這個上限就是每次 Iteration 內會遇到的 branch 指令次數，一旦全部會遇到的案例都能夠被放入 table 內部，那麼就不會有需要置換或是查找不到預測值的問題，也就表示假如 (branch instruction / iteration) = x 而 entry size = y 且 x < y 的話，對於任何 entry size z > y，其表現都與 entry size = y 相差無幾。以上為我個人初期的推論，詳細結果與驗證會在後續提及與說明。

B. 電路實作

要調整 entry size 有幾個地方要修改，除了在 bpu.v 內將以 ENTRY__NUM 設置的 addr_hit_CPU case 選項都更改成希望的 entry size 例如 32 等等之外，還要將 auila_top 中呼叫 core_top module 以及 core_top 呼叫 bpu module 時的參數也一併更改 .

```
// In auila_top.v
core_top#( .HART_ID( HART_ID ),  .XLEN( XLEN ),  .BPU_ENTRY_NUM( 32 ))
// In core_top.v
module core_top #(
    parameter HART_ID             = 0,
    parameter XLEN                = 32,
    parameter BPU_ENTRY_NUM  = 32  // <= here
// In bpu.v
case (addr_hit_PCU)
    32'h0000_0002: read_addr <= 1 ;
```

另外一部分是欲關閉 BPU 有幾個值要設定，分別是先在 core_top.v 設定代表 disable 的參數為 0，並在 program_counter.v 與 pipeline_control.v 將原先的 `ifdef ENABLE_BRANCH_PREDICTION 修改為該參數，以我為例便是更改如下圖:

```
// In core_top.v
`define disable_branch_prediction 0;
// In program_counter.v
`ifndef disable_branch_prediction
// In pipeline_control.v
`ifndef disable_branch_prediction
```

III. ANALYSIS OF DIFFERENT ENTRY SIZE

以下是我針對不同 BPU 情形進行的分析結果，可以分為以下幾點:

A. *BPU 的使用與否對 Performance 之影響*

由 Figure 1.Iteration / sec Performance Comparison of With and Without BPU 可見，.一旦關閉 BPU 功能，每單位時間內能完成的 iteration 次數會大幅下降，也就表示無法藉由預測下步執行位置來達到加速的效果; 相較之下，一旦使用 BPU 可以有效地提升單位時間內的 Iteration 次數，這是因為藉由 BHT 的紀錄我們可以先行猜測該指令要跳到何處，即使猜錯目的地，頂多也是變成和沒使用 BPU 一樣的執行過程，因此並不會造成增加 cycle 的反效果。

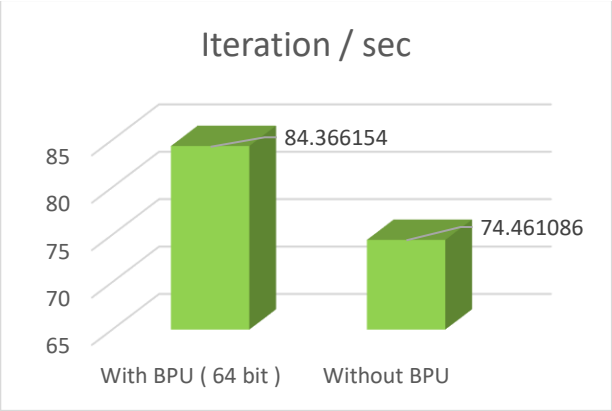

Figure 1.Iteration / sec Performance Comparison of With and Without BPU

B. *不同 Entry Size 之 BPU 對 Performance 之影響*

參照 Figure 2..Iteration / sec Performance Comparison of Different Entry Size，可以發現單位時間內能完成的 Iteration 數量自從 Entry.Size 超過 32 bit 之後就有逐漸緩和的趨勢，這符合在 II.A 初步想法 提及的猜測，一旦 Entry Size 增加到某個臨界值，整體表現就會因為 table 已能容納所有案例而逐漸趨緩; 反觀在 16 bit – 32 bit 這個 range 內有較大的落差，因此我猜測 coremark 在每次 iteration 內所使用到 branch instruction 的次數可能在 16-32 之間，詳細近一步的測試與證明會在下個版本的報告中提及。

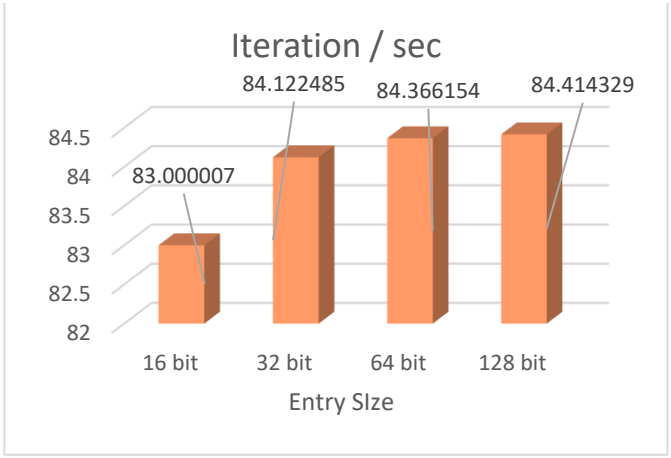

Figure 2..Iteration / sec Performance Comparison of Different Entry Size

C. *不同 Branch Type 之 hit / miss rate 比較*

此處數據尚有錯誤，下個版本會進行修正

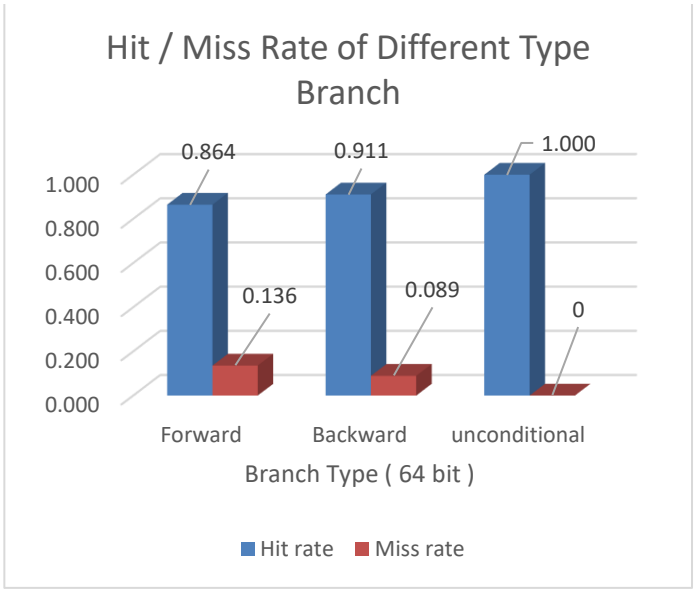

Figure 3.Hit / Miss Rate of Different Branch Type

IV. TWO-LEVEL BRANCH PREDICTOR

A. *初步概念.*

考慮到單純 2 bit predictor 有它的極限，我們可以嘗試實作 2 level branch predictor，差別在於並不是直接拿 address 當作 input，而是利用過往 history 與當前 address 做 XOR 來作為 predictor 的 input。

我預計此次作業將使用 8bit (dec_pc_i[8-1:0])與 history table 做 XOR，並且這個 table 儲存的 hash 值是由部分自身與 taken (2 bit)進行 concatenate 的組合。