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，發現一旦table size大於單次iteration中branch指令出現次數，即會達到趨近表現優化的極限; 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。結果呈現two level predictor並非所有情況都能造成表現上升，反而可能因為考慮冗贅資訊導致index配對率大幅下降。

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

# 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.

# Branch Analysis 前置作業與構思

## 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。

## 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:

取代 `ifdef ENABLE\_BRANCH\_PREDICTION

// In core\_top.v

`define disable\_bpu 0;

// In program\_counter.v

`ifndef disable\_bpu

// In pipeline\_control.v

`ifndef disable\_bpu Analysis of Different Entry Size

以下是我針對不同BPU情形進行的分析結果，可以分為幾點:

## BPU的使用與否對Performance之影響

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

## 不同Entry Size 之BPU對Performance之影響

參照Figure 2，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。

Figure .Iteration / sec Performance Comparison of Different Entry Size

## 不同Entry Size之 hit / miss rate 比較

Figure 3.Hit / Miss Rate of Different Entry Size

## 不同Branch Type之 hit / miss rate 比較

Figure 4.Hit / Miss Rate of Different Branch Type

# Two-Level Branch Predictor

## 初步概念.

原始2 bit predictor有它的限制在於它只local branch prediction導致疏漏global的資訊，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Global Branch History與address做XOR來作為predictor的input 。我預計此次作業將使用講義第二種predictor的方式，將 dec\_pc\_i 的其中6 bit 與Global Branch History 做XOR，並且這個Global Branch History值是由每次的taken ( 1 bit )進行concatenate並取末6 bit的組合。上述概念化簡為圖可參考Figure 5

## 

Figure 5.Two level predictor (2nd)

## 電路實作

依照Figure 5.Two level predictor (2nd)，我另外設置6 bit的global\_branch\_history以及hash\_pc、hash\_dec\_pc。

Hash的方式是由如下:

hash\_dec\_pc <= (global\_branch\_history^dec\_pc\_i[8:3]);

(對於要取哪6 bit後續有詳細實驗)

而原先branch\_likelihood要放入的update\_addr，改成上述XOR過後的值:

// branch\_likelihood[update\_addr] <= 2'b01;

branch\_likelihood[hash\_dec\_pc] <= 2'b01;

## 實驗結果

我首先嘗試使用dec\_pc\_i[5:0] 作為XOR的對象。在我原先的猜測，two level predictor應該要比原始的saturating predictor擁有更好的performance憶及更高的iteration/sec ; 但結果Figure 6出乎意料的表現下降不少，因此我又嘗試各種不同dec\_pc bit的組合，而數據**錯誤! 找不到參照來源。**顯示即使這些組合裡最好的表現( 選用dec\_pc\_i [6:1]作為XOR operand )依舊遠遠不及原始predictor帶來的成果，因此我對這樣的數據進行一些分析與推論。

Figure 6.Performance Between Origin Predictor and Two Level Predictor

Figure 7.Performance Between Different Bit Combination

## 分析與討論

針對III.C現象的成因，我有以下推論:

Coremark執行過程較少或是不需要考慮Global Data。若是Coremark其實只需將local data納入考量，那麼我實作的Two Level Predictor反而是將不必要的資訊考慮進去，複雜化的index 導致branch table的match率大幅下降 。

當然，與Figure 1呈現without BPU的數據相比，該Two Level Predictor仍然有表現上的提升，代表此預測器仍有一定效力，只是相對單純使用local branch history的預測器反而考慮冗贅資訊index的hash致使配對率下降。

## 結論

在某些情況下，Two Level Predictor因為多考慮到Global Branch History，使table能將更多資訊納入範圍，同時避免likelihood輕易地被改變 ; 但在面對只須考慮local data的狀況下，這個做法反而造成累贅，導致index更難被配對table必須不斷更新，重複移除與塞入不同值作為index致使cycle數量往沒predict的表現靠攏。