HW2 Report

王柏舜, 0716089

*Abstract*—這次作業對aquila 的 branch predictor 進行優化，並對原始的branch predictor 進行分析。在第一階段只更改 branch predictor 的 table size 能將 DMIPS 提升至約0.97

Keywords—component; formatting; style; styling; insert (key words)

# Introduction

這次作業是對aquila 的 branch predictor 進行優化來達到提升 DMIPS 的目的，第一階段是透過修改branch predictor 的table size 來討論table size 和 branch statistics 、 DMIPS 之間的關係 。並且在分析完原始的branch predictor 後嘗試設計出two level branch predictor 來改善 DMIPS。

# 改變TABLE SIZE 的影響

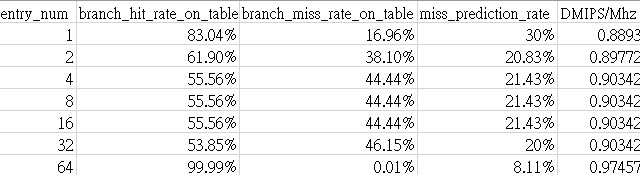
## 如何改變 TABLE SIZE

原始BPU.v中的設計只能讓 table size 最大到32，要讓table size 可以到 64 需要更改always block 中對addr\_hit\_PCU 和addr\_hit\_DEC 中的case 數量，以及更改在aquila\_top.v 中對BPU 宣告的 entry 數目，這樣才能正確更改 BPU 中的table size。

## 如何分析 BRANCH STSTISTICS

我在BPU.v 中新增加了一個 always block 以及 三個 32bits 的 registor 來分別計算 (1)branch prediction 失敗的數量我命名為bpu\_miss\_count (2)pc\_i form exe 有hit 到 BPU 的 table 我命名為total\_count (3)pc\_i from decode 有 hit 到 BPU 的table 我命名為 miss\_count 。這三個registor 數值會增加一的條件依序分別為(1)當 BPU 的 input : branch\_misprediction\_i 為 1 (2) 當BPU 的output : branch\_hit\_o 為1 (3) 當BPU 的 internal signal : we 為 1 。 .

## 結果

最終的統計結果為下圖所呈現的樣子，除了DMIPS 是透過uart跑出來的結果，其餘的結果都為simulation 上訊號的值，為了避免DMIPS 差異不大我將輸出結果增加到小數點後第五位，entry number 由最低 1 至 64。

可以由以上的結果明顯的發現，除了entry number 為 1 和 64 外的數據差異其實不太明顯，entry number 1 的結果原因非常明顯，因為table size 太小了 branch 在 table 上的miss 太長發生讓 branch predictor 沒有太好的猜測結果，而entry number 2至 32 的結果幾乎沒有相異我認為這是Dhry benchmark 的關係，benchmark 所呼叫的branch種類太少了以至於只有某些比較少出現的 instruction 會不常駐於 table 上，對於這個的驗證我從BPU中抓了訊號來計算branch 和 jump 指令的數量來看，最後發現前者出現的數量為後者的兩倍以上，這就說明了branch 會常駐於table 上使的猜測的結果會比較準確但是後者很常被前者覆蓋過去以至於每次出現後者的指令時幾乎都是從initial state 開始猜，這樣會使的猜測結果不準確，而 對於entry number 64想法的印證，我透過在 simulation 中觀察計算 branch miss on table 的 registor 的數值會停留在 60，這 證實了benchmark 的 branch 種類無法填滿長度為64 的 branch predictor table。此外我也將 branch predictor 關掉來進行比較，實作方式為將 pipeline\_control.v 和 program\_counter.v 中加入 `define disable\_branch\_prediction 1 來關掉 BPU，而關掉後再UART 上跑出來的DMIPS 為 0.87832。最後我抓了fetch 中的 pc 和 flush 的訊號想要找出BPU 甚麼時候容易發生猜錯的情況，藉由比對發現在strcpy 和 strcmp 中當輸入不滿足一個 word時fetch unit 中會頻繁的發生對於pc的flush。

# DESIGN BENCHMARK PROGRAM

這部分我並沒有真的實做，但跟據我在不同entry number 所觀察到的結果，我認為要能顯現出aquila 的 branch predictor 的弱點，我們需要在 benchmark 中加入更多複雜fucntion call 或者是更多種類的branch 種類，又或是同一種bracch 種類的出現頻率要在更多，這是因為在可以填滿branch prediction table 的情況下可以讓更多常用的branch 種類在branch table 上發生更多對entry 的競爭。如此一來才能對branch predictor 有更晚整個檢視。

# 計畫如何使用 two-level 的 branch predictor

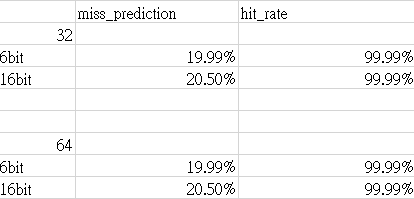
自在了解對two-level branch predictor 的了解後我目前是想要以一個global branch history table 和pattern history table 來執行。Global branch history table 是一個shift registor 用來記錄最近branch 的執行結果，因為在我們寫出來的程式碼中常常會遇到有for loop的情形這時global branch history就很適合來判斷for loop 執行到哪裡下一次branch 的執行結果，另外pattern history table 是以branch 的pc 為 index 的table而他會記錄branch likelihood 這個table 我還會想要希望能加入一個紀錄branch 出現頻率的功能，目的是要在branch miss on table 時被取代的那一個entry 是最少在使用的branch ，我想有這樣的功能應該能夠讓branch predictor 在處理這種miss 的能力上有更好的提升，主要有這樣的想法產生是我透過simulation 時比對 fetch unit 中的 pc 和 flush 訊號關西發現可以利用紀錄是否take branch 的 pattern 來試圖解決當load 進來的資料不滿足一個 word時會有頻繁對於pc 的 flush。

# TWO LEVEL BRANCH PREDICTOR

## Implement:

我新增加了一個global\_pattern registor 用來記錄每一次的branch 是否有 take 並且重新設計table 的 index我利用 address[31,6] 和 global\_pattern[5,0] 為table 的新index，global\_pattern 是利用shift 來記錄最近16 次branch 的情況

## RESULT OF SIMULATION

跑完模擬後的數據如下，我分別對table entry 32 and 64 和 global\_pattern 6 bit 和 16 bit 四種組合來觀察miss\_prediction rate 和 table hit rate 。

對於16 bit 比較容易猜錯的原因我想是因為global\_pattern 取太多位讓 branch address 的特徵下降太多有太多的不同的branch 都對應到同一個table entry 上，在entry 數 64則是因為DHRYSTONE 的branch 種類太少，原本table 可以記所有的 branch address 但是和global address 結合後會讓不同的branch address 對應到同一個branch 讓branch predictor 的準確度下降。而DIMPS 的數值並沒有特別的提升有些甚至下降了entry 數32 global\_pattern 6 bit 為 0.90 DMIPS/Mhz ，entry 數32 global\_pattern 16 bit 為 0.89 DMIPS/Mhz ，entry 數64 global\_pattern 6 bit 為 0.94 DMIPS/Mhz ，entry 數64 global\_pattern 16 bit 為 0.93 DMIPS/Mhz