# 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)***

## I. 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。

## II. 改變 TABLE SIZE 的影響

### *A. 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。

### *B. 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 。 .

### *C. 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。

| entry_num | branch_hit_rate_on_table | branch_miss_rate_on_table | miss_prediction_rate | DMIPS/Mhz |
|---|---|---|---|---|
| 1 | 83.04% | 16.96% | 30% | 0.8893 |
| 2 | 61.90% | 38.10% | 20.83% | 0.89772 |
| 4 | 55.56% | 44.44% | 21.43% | 0.90342 |
| 8 | 55.56% | 44.44% | 21.43% | 0.90342 |
| 16 | 55.56% | 44.44% | 21.43% | 0.90342 |
| 32 | 53.85% | 46.15% | 20% | 0.90342 |
| 64 | 99.99% | 0.01% | 8.11% | 0.97457 |

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。

## III. 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。

## IV. 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。

## V. TWO LEVEL BRANCH PREDICTOR

### A. *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

### B. *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。

| | miss_prediction | hit_rate |
|---|---|---|
| 32 | | |
| 6bit | 19.99% | 99.99% |
| 16bit | 20.50% | 99.99% |
| | | |
| | | |
| 64 | | |
| 6bit | 19.99% | 99.99% |
| 16bit | 20.50% | 99.99% |

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