Computer Architecture
Fall, 2019
Week 13
2019.12.02

組別：__________ 簽名：____

[group12] (對抗賽)

1. 下列關於 Multiple issue 的敘述何者正確，請修正錯誤答案？

a. 使用 Multiple issue 方法有機會使 CPI<1

b. Static multiple issue resolves hazards at runtime

c. Dynamic multiple issue resolves hazards by Compiler

d. Speculation guess what to do with an instruction

A:

a. True

b. 非在 runtime

c. Dynamic multiple issue resolves hazards by CPU

d. True

[group11] (對抗賽)

2. Given that compilers can also schedule code around data dependences, but why a superscalar processor would use dynamic scheduling? (give 3 reasons)

ANS:

(1) Not all stalls are predictable.

(2) Can't always schedule around branches. If the processor speculates on branch outcomes using dynamic branch prediction, it cannot know the exact order of instructions at compile time.

(3) Different implementations of an ISA have different latencies and hazards. As the pipeline latency and issue width change from one implementation to another, the best way to compile a code sequence also changes.

[group10] (對抗賽)

3.

(1)關於以下 MIPS code 使用 1bit (taken/not taken) Dynamic Branch Prediction 需要 flush pipeline 幾次 假設 branch history table 的值為 taken？

```
Start :
            addi        $t0,        $0,         5
            add         $t1,        $0,         $0
Loop :      beq         $t0,        $0,         Continue
            add         $t1,        $t1,        $t0
            addi        $t0,        $t0,        -1
            j           Loop
Continue:   //Other code
             :
             :
             :
```

(2)承上題，若 predict branch always taken 則以上 MIPS code 需要 flush pipeline 幾次?

ANS

(1)2 次

第一次為進 Loop 第二次為出 Loop 跳到 continue 時

(2)5 次

當 to 為 5, 4, 3, 2, 1, 時 beq 皆不跳，但猜測為跳所以共五次。

[group5]

4. When handling Branch Hazard, if pipelines are deep, how can we solve the problem of significant branch penalty? How?

A：Dynamic prediction.

Ans:
根據過去歷史推測(branch prediction buffer, branch history table)，如果過去為
taken : predict taken
not taken : predict not taken
預測錯誤 : flush pipeline and flip prediction

[group7] (對抗賽)

5. Consider a 5-stage pipeline like MIPS. The finish time of branch instructions can be moved early from MEM to ID. What are the costs behind that? Is it possible to move earlier to the IF stage?

Ans: 需要增加額外硬體 (XOR array) 於 ID stage 來比較兩個暫存器是否相等。
不可能將 branch 完成時間移至 IF stage。因為此時尚無法擷取出暫存器內容來進行比較。

[group2] (對抗賽)

6. Consider the following three branch prediction mechanism: Predict not taken, Predict taken, Dynamic prediction. Assume they won't be punished under correct prediction, while they will be punished for two clock cycles if predicting wrong. The average accuracy of predictor is 90%. What kind of predictor is capable for the following branch?
   1. Branch establishment ratio = 5%
   2. Branch establishment ratio = 95%
   3. Branch establishment ratio = 70%

A1:

	Predict not taken	Predict taken	Dynamic prediction
1	0.95	0.05	0.9
2	0.05	0.95	0.9
3	0.3	0.7	0.9

[group9] (對抗賽)

7. 試比較 branch hazard 以及 exception 的相同及相異之處。

Ans.
發生原因:
branch hazard: 因為預測 branch equality 錯誤，導致 instruction 執行的次序出現錯誤，因此在做 branch decision 前 fetch 入的指令都不能被執行。
exception: 因為在程式運行中途發生 overflow 或碰到不合法 opcode，故當前及接下來的指令都不能被執行。
處理過程:
相同處:
兩者都有把不該執行的指令"flush"掉的過程，且沖掉指令後，空出來的位置都會變成 bubble。
處理過程都會產生 performance penalty。
相異處:
Exception 須把錯誤資訊存到 cause register 及 EPC 內，branch hazard 則無。
branch hazard 指令 flush 完後，至多就是改變指令執行的次序(看是要跳還是 sequentially 執行)，因此程式還是保證可以繼續運行下去。
但 exception 指令沖完後需跳回 OS，確認是否有辦法補救，若無法補救就會回報錯誤，程式也就直接被中斷了。
如何預防或降低 penalty:
branch hazard:
利用 dynamic prediction，即可大幅增加在 loop 中的預測準確率。(預防)
把 branch decision 的工作挪到較前面 pipeline stage 執行。即使最後 prediction 錯誤，所被沖掉的 stage 數量也會下降許多。(降低 penalty)
Exception:
確實做好 debug 動作，檢查指令字元有沒有打錯、或是數值會不會太大導致 overflow。(預防)
升級 OS，讓 OS 可以以更快的速度完成 recover 。(降低 penalty)