計結 CH4 Part 2

107061218 謝霖泳

1.

假設原本的 CPI 為 1，計算後來的 CPI'。以下兩小題的算是均可看到乘以 1 的過程，那是我假設 misprediction 發生時，只需要 flush 掉一個 stage 的 instruction，理由是題目中說 Assume branch is calculated at ID stage，也就是我推測該 CPU 支援 early branch prediction，所以只需要 flush one instruction。

(1) CPI' = 1 + (1 - 0.55) * 0.25 * 1 = 1.1125。

(2) CPI' = 1 + (1 - 0.85) * 0.25 * 1 = 1.0375。

2.

首先，我用紅色來表示 ALU / branch instruction，藍色表示 load / store instruction。並用灰色與白色的底色分別 two-issue 中不同的 packet，每個 packet 中可容納上述兩種 type 的各一個 instruction。

(1)

在本小題中，可以看到幾乎所有的 packet 都只有一個 instruction 在運作，這是因為這題並沒有做指令重排，而是直接將它們丟到 two-issue packet 中，所以因為 instruction type 的不合或是 data dependency 的問題，使得大部分 packet 中都只有其中一邊的指令在運作。像是前面都是紅色的 ALU / branch type instruction，好不容易遇到藍色的 ld x7, 0(x6)指令，卻因為 x6 最快要等到前一個 add instruction EXE 結束之後才拿得到，所以又不能一起執行了。

另一點值得注意的是，在倒數第二個 packet 的地方，有先 stall 一個 cycle 才執行，其原因為 x29 為 sub 的 source，最快要等到前一個 packet 中的 ld 做完 MEM Stage 之後才有可能 forward 過來，所以必須停下一回合，以確保 x29 之正確性。

本小題最終以 16 個 cycle 完成，如下頁 pipeline diagram 所示。

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
li x12, 0	IF *	ID *	EX *	ME *	WB *											
jal ENT		IF *	ID *	EX *	ME *	WB *										
bne x12, x13, TOP			IF *	ID *	EX *	ME *	WB *									
slli x5, x12, 3				IF *	ID *	EX *	ME *	WB *								
add x6, x10, x5					IF *	ID *	EX *	ME *	WB *							
ld x7, 0(x6)						IF *	ID *	EX *	ME *	WB *						
ld x29, 8(x6)							IF *	ID *	EX *	ME *	WB *					
sub x30, x7, x29								* *	IF *	ID *	EX *	ME *	WB *			
add x31, x11, x5										IF *	ID *	EX *	ME *	WB		
sd x30, 0(x31)											IF	ID	EX	ME	WB	
addi x12, x12, 2											IF	ID	EX	ME	WB	
bne x12, x13, TOP												IF *	ID *	EX *	ME *	WB *

(2)

本小題要求將指令重排，以達到最高效率。所以我將 add x31, x11, x5 與 addi x12, x12, 2 往前移，分別移到兩個 ld 指令的 packet 之中，並將最後面的 bne x12, x13, TOP 順勢往前移，移至與 sd x30, 0(x31)平行的位置。

而最後一個 packet 中的 sd 不能再往上移動一個 packet 讓它與 sub 平行的原因也是因為 data dependency，因為 x30 是 sub 的 destination 又是我要存的東西，我必須要等它先算完有結果才能存，所以不能放在同一個 packet 中。此外，與上小題相同的是倒數第三個指令 sub 因為 load RAW 的關係，仍需 stall 一個 cycle。

本小題最終以 14 個 cycle 完成，如下頁 pipeline diagram 所示。

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
li x12, 0	IF	ID	EX	ME	WB									
	*	*	*	*	*									
jal ENT		IF	ID	EX	ME	WB								
		*	*	*	*	*								
bne x12, x13, TOP			IF	ID	EX	ME	WB							
			*	*	*	*	*							
slli x5, x12, 3				IF	ID	EX	ME	WB						
				*	*	*	*	*						
add x6, x10, x5					IF	ID	EX	ME	WB					
					*	*	*	*	*					
ld x7, 0(x6)						IF	ID	EX	ME	WB				
add x31, x11, x5						IF	ID	EX	ME	WB				
ld x29, 8(x6)							IF	ID	EX	ME	WB			
addi x12, x12, 2							IF	ID	EX	ME	WB			
sub x30, x7, x29								*	IF	ID	EX	ME	WB	
								*	*	*	*	*	*	
sd x30, 0(x31)										IF	ID	EX	ME	WB
bne x12, x13, TOP										IF	ID	EX	ME	WB

(3)

本小題需要將 loop 執行兩次的結果做 unroll 並重排且優化之。首先，因為 unroll 的關係，可以省去很多重複性的計算，像是 load 得部分，我不需要做兩次 slli x5, x12, 3 和 add x6, x10, x5，因為他們在兩次執行的過程是一樣的，所以只需依照 index 的不同去給不同的 offset 去取值，像第一次執行是取 0(x6)與 8(x6)作相減，第二次則是 16(x6)和 24(x6)相減。

第二，經過重排之後我將第一次 loop 的 sub 往下移了，所以免掉了前二小題都有遇到的 load RAW 要多等一個 cycle 的問題。但第二次執行 loop 時，這個情況又出現了，而且這次無可避免的一定要 stall 一個 cycle。

此外，因為已經確定執行兩次，所以第一次執行結束的 bne 也拿掉了。我也將第二次相減的 source 與結果都存在與第一次不同的 register 中，這樣才能在最後面的兩個 packet 終將它們都存回去，不會產生兩次運算的變數打架的問題。可以觀察到兩次將結果 store 回去的地方，我使用的方法也是給不同的 offset 去存回陣列對應的不同位置。

本小題最終以 16 個 cycle 完成，如下頁 pipeline diagram 所示。

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
li x12, 0	IF	ID	EX	ME	WB											
	*	*	*	*	*											
jal ENT		IF	ID	EX	ME	WB										
		*	*	*	*	*										
bne x12, x13, TOP			IF	ID	EX	ME	WB									
			*	*	*	*	*									
slli x5, x12, 3				IF	ID	EX	ME	WB								
				*	*	*	*	*								
add x6, x10, x5					IF	ID	EX	ME	WB							
					*	*	*	*	*							
ld x7, 0(x6)						IF	ID	EX	ME	WB						
add x31, x11, x5						IF	ID	EX	ME	WB						
ld x29, 8(x6)							IF	ID	EX	ME	WB					
addi x12, x12, 4							IF	ID	EX	ME	WB					
ld x8, 24(x6)								IF	ID	EX	ME	WB				
								*	*	*	*	*				
ld x28, 16(x6)									IF	ID	EX	ME	WB			
sub x30, x7, x29									IF	ID	EX	ME	WB			
sub x27, x8, x28										*	IF	ID	EX	ME	WB	
sd x30, 0(x31)										*	IF	ID	EX	ME	WB	
sd x27, 16(x31)												IF	ID	EX	ME	WB
bne x12, x13, TOP												IF	ID	EX	ME	WB