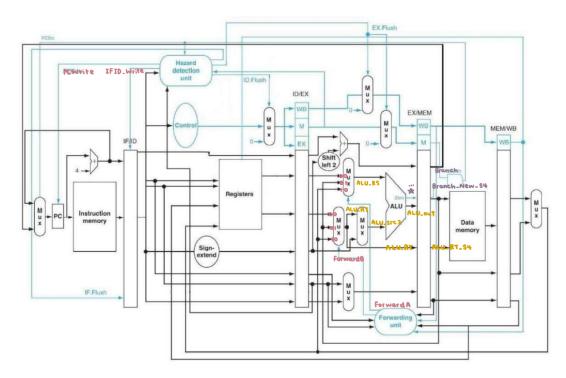
# **Computer Organization Lab5**

Name: 莊婕妤 ID: 109550182

# **Architecture Diagram**



# **Hardware Module Analysis**

這次的 Lab 我以上次較簡化的 Pipelined CPU 程式碼為基礎來實作, 本次實作的重點在於需要設計處理 Data Hazards 以及 Branch Hazards 的相關機制, 因此我增加了 Hazard Detection Unit 以及 Forwarding Unit。整體電路的架構細節可以參考上面的圖(有用顏色標記的是與上次 Lab 較不同的設計, 以及新加的訊號名稱)。

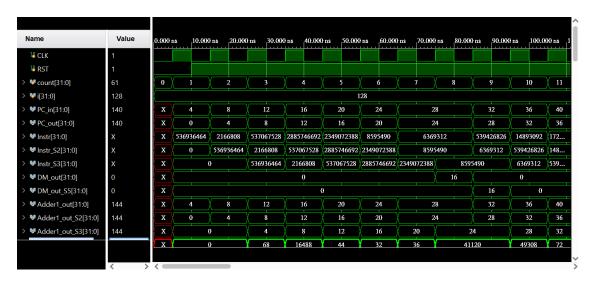
因為這次的 Lab 需要多處理三種 branch operation, 因此在圖上標記紫色星星的地方我做了與實作圖稍加不同的設計, 增加了一個 4 to 1 的 Multiplexer, 並且用 branchType 作為 control signal 來 select 該 multiplexer 的 output。

## 輸入及輸出如下表所示:

	BranchType	Output
BEQ	00	Zero
BGT	01	~ (Zero OR ALU_out [31])
BGE	10	~ ALU_out [31]
BNE	11	~ Zero

## **Finished Part**

#### Test Data 1



######	##########	###########	###### clk_				!#########	####
r0 =	0, r1 =	16, r2 =	256, r3 =				24, r7 =	26
r8 =	8, r9 =	1, r10=	0, r11=	0, r12=	0, r13=	0, r14=	0, r15=	0
r16=	0, r17=	0, r18=	0, r19=	0, r20=	0, r21=	0, r22=	0, r23=	0
r24=	0, r25=	0, r26=	0, r27=	0, r28=	0, r29=	0, r30=	0, r31=	0
				Nama av				
mO =	0, m1 =		0, m3 =				0, m7 =	0
m8 =	0, m9 =	0, m10=	0, m11=	0, m12=	0, m13=	0, m14=	0, m15=	0
m16=	0, m17=	0, m18=	0, m19=	0, m20=	0, m21=	0, m22=	0, m23=	0
m24=	0, m25=	0, m26=	0, m27=	0, m28=	0, m29=	0, m30=	0, m31=	0

# **Explanation**

為了方便觀察每一個訊號在各個 clock cycle 下變動的情形,我將所有的訊號都設為 Pipe\_CPU Module 的 output,將他們顯示在波形圖上(有將輸出形式改為 unsigned decimal),也方便我去做 debug。而上面 Test Data 1 波形圖以及跑出的結果都和我自己 trace 過整個 assembly code 的結果相同,因此我認為我的電路實作是正確的。

## Test Data 2

Name	Value	0.000 n		00 ns 20.00	0 ns  30.00	0 ns  40.00	0 ns   50.000	0 ns   60.00k	0 ns 70.00	0 ns  80.000	ns  90.00	Ons 10	n 000.00	s
<sup>™</sup> CLK	0											<u> </u>		
<b>₩</b> RST	1													
> ♥ count[31:0]	10	0	1	2	3	4	5	6	7	8	9	(10	χ:	11
> <b>™</b> i[31:0]	128		128											
> <b>₩</b> PC_in[31:0]	40	x	4	8	12	16	20	24	28	32	36	<u>4</u> 9		44
> <b>W</b> PC_out[31:0]	36	X	0	4	8	12	16	20	24	28	32	36		40
> <b>₩</b> Instr[31:0]	537198608	x	0	537001987	2885812224	537001985	2885812228	2885681160	537001989	2885812236	537001984	53719 <mark>8</mark>	608 53	7
> <b>№</b> Instr_S2[31:0]	537001984	x		0	537001987	2885812224	537001985	2885812228	2885681160	537001989	2885812236	53700 <mark>1</mark>	984 53	7
> <b>№</b> Instr_S3[31:0]	2885812236	x	X 0 537001987 2885812224 537001985 2885812228 2885681160 537001989 2885						<mark>28858</mark> 12	236 53	7			
> <b>♥</b> DM_out[31:0]	0	x	X 0											
> <b>♥</b> DM_out_S5[31:0]	0	x	X 0											
> <b>V</b> Adder1_out[31:0]	40	x	4	8	12	16	20	24	28	32	36	49	(	44
> • Adder1_out_S2[31:0]	36	x	0	4	8	12	16	20	24	28	32	36		40
> • Adder1_out_S3[31:0]	32	x		0	4	8	12	16	20	24	28	32		36
		x		0	<b>4</b>	20	12	20	36	56	48	<b>∑</b> 80	Y	36

######			###### clk_				###########	###
r0 =			12, r3 =				0, r7 =	0
r8 =	2, r9 =	0, r10=	0, r11=	0, r12=	0, r13=	0, r14=	0, r15=	0
r16=	0, r17=	0, r18=	0, r19=	0, r20=	0, r21=	0, r22=	0, r23=	0
r24=	0, r25=	0, r26=	0, r27=	0, r28=	0, r29=	0, r30=	0, r31=	0
				Memory				
mO =	4, m1 =	1, m2 =	0, m3 =	_		0, m6 =	0, m7 =	0
m8 =	0, m9 =	0, m10=	0, m11=	0, m12=	0, m13=	0, m14=	0, m15=	0
m16=	0, m17=	0, m18=	0, m19=	0, m20=	0, m21=	0, m22=	0, m23=	0
m24=	0, m25=	0, m26=	0, m27=	0, m28=	0, m29=	0, m30=	0, m31=	0

# **Explanation**

與 Test Data 1 不同的地方是 Test Data 2 有需要處理 branch hazard 的問題,因此我在 trace 波形圖時有特別 focus 在 Branch 以及 BranchType 等 control signal 上。而上面的波形圖以及 register file、memory file 在每個 clock cycle 下輸出的結果都和我自己 trace 過整個 assembly code 的結果相同,因此我認為我的電路實作是正確的。

#### **Problems You Met and Solutions**

在實作得過程中,其中一個我遇到的問題是,對於如何處理 load-use data hazard 以及 branch hazard 沒有頭緒(有關於 Flush Register, Stall 相關的操作),而我也在詢問同學後得到了很大的幫助。另一個遇到的問題是,在設計完 Advanced Pipelined CPU 後,我的電路在 Vivado 上無法輸出正常的結果,因此就如同先前所述,我 output 了所有 CPU 會用到的訊號,再利用波形圖以及電路圖來 trace 哪裡出問題,後來才慢慢修改成現在的結果。

# Summary

就即使在課堂上、準備期末考的過程中,完全理解不同 hazard 的處置原理,但與如何透過 verilog 來實作還是相差甚遠。透過這次的 Lab,我更了解了整個 Advanced Pipelined CPU 的運作原理、中間的訊號是如何做傳遞的,以及 module 的設計來協助處理 hazard 的問題(像是如何做到 stall one cycle 等)。