Final Project

106學年度第2學期

Pipelined CPU Design

老師: 朱守禮 老師

學生: 10527130 陳少洋

10527132 林亞吟

10527135 張智欽

10527140 初 元

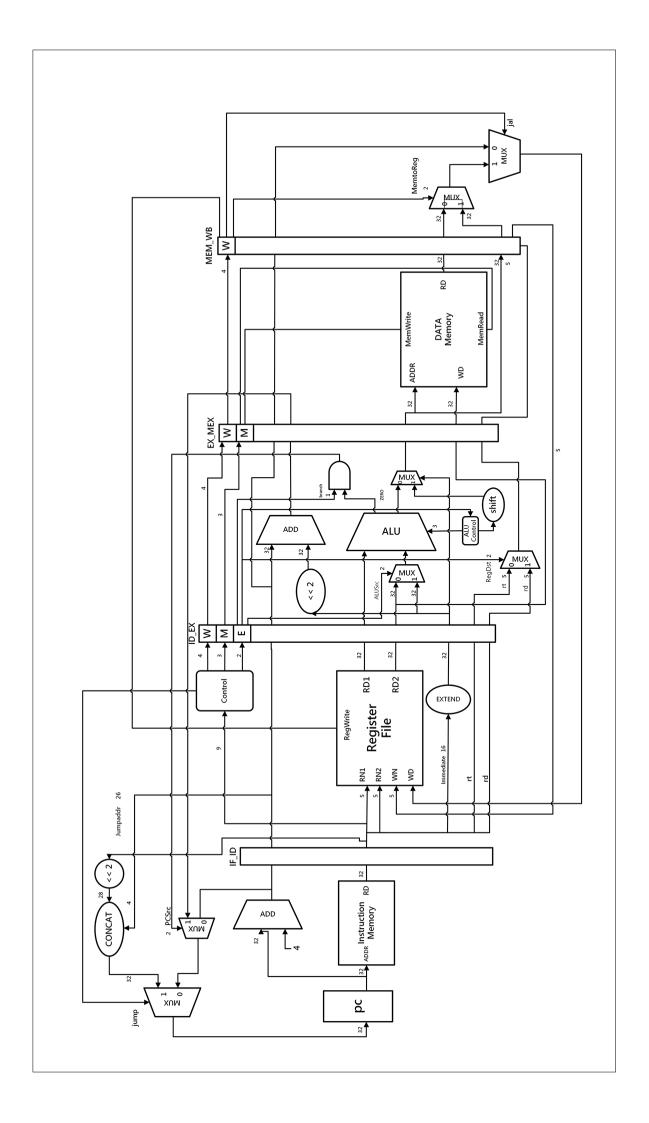
一、背景

本 Final Project 主要為設計一個 Pipelined CPU、第三版除法器,其目的 主要為實現十六項功能:AND、OR、ADD、SUB、SLT、SLL、SLTI、LW、 SW、BEQ、J、JAL、DIVU、MFHI、MFLO、NOP。

本次 Project 我們使用 ModelSim 來 compiler 以及執行,透過模擬 waveform 來檢查是否與 Testbench 的計算結果相同。

本次 Project 須符合以下規範:

- (1) 一個 Module 一個檔案,同時檔案名稱需與 Module 名稱相同。
- (2) 須放置以 Visio、Word 或 PowerPoint 繪製的 Datapath 架構圖。
- (3) Testbench 須依助教所提供之參考設計。
- (4) 設計均以 Verilog 完成,且須通過 ModelSim 模擬執行。
- (5) Verilog 設計只能包含規定的十六項功能。



(一) ALU(組合邏輯):

沿用 MidTerm Project 設計的 ALU,新增 SLL 指令。

1. 設計重點

我們先設計 1-bit 的 ALU, instance 32 次, 達到 Ripple Carry 的效果。

2. 說明

● ADD、SUB、SLT 不可直接使用"+"、"-",必須以邏輯 閘撰寫設計。

• AND : 000 / 36 / 100100

OR : 001 / 37 / 100101

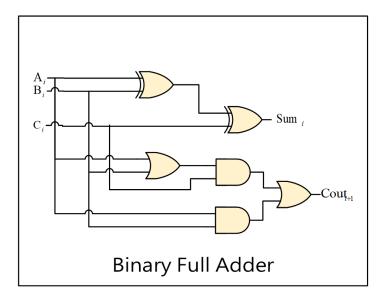
ADD : 010 / 32 / 100000

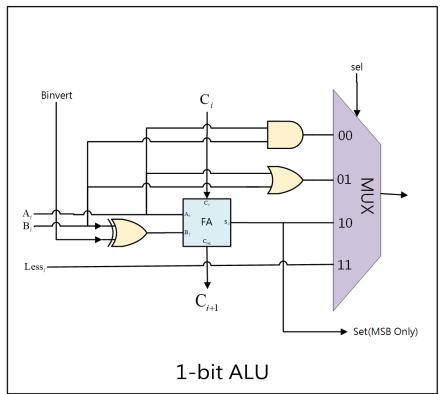
SUB : 110 / 34 / 100010

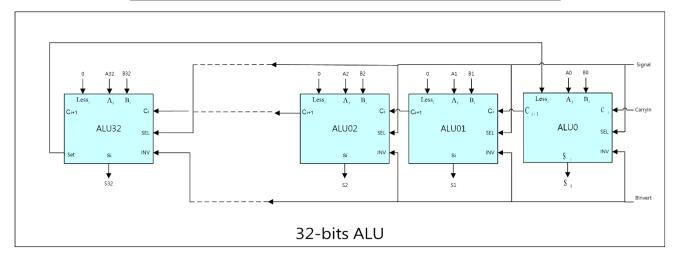
SLT : 111 / 42 / 101010

SLL : 011 / 0 / 000000

- AND:000 第一位元為 Bit Invert,二、三位元為 Selection,36 則為指令代碼,100100 則為 6-bits Signal,為 36 轉 binary後的值,我們讓他 1 bit 1 bit 做 AND 直到 32bit 結束,OR 也是一樣的操作。
- ADD: 010 第一位元為 Bit Invert,二、三位元為 Selection, 32 則為指令代碼,100000 則為 6-bits Signal,為 32 轉 binary 後的值,設計 ADD、SUB 時我們都是使用 Full Adder,差別 在於 Binvert,從 AND、OR、ADD、SUB、SLT 的 Signal 可以發現,只有 SUB、SLT 的 Signal[1]為 1,因此我們利用 Signal[1]來做判斷,決定 Binvert 是 0 或 1。
- 使用 SUB、SLT 時,兩指令都會有相減的動作,因此我們設計了一個 BinvertCarry,用來當成第一 bit 的 ALU 的輸入,若指令為 SUB、SLT,BinvertCarry 就設定為 1,其餘指令為 0。
- SLT 會利用 SUB 判斷兩數的關係,若兩數相減大於等於 0, SUB 結果的最高位元為 0,反之,兩數相減小於 0為 1。最後 會將此最高位元設定成輸出值的最低位元。
- SLL 是使用 Midterm Project 寫好的 Shifter 放入 ALU,實現邏輯左移的功能,ALU 再依據輸入指令來決定是否要輸出 SLL 結果。







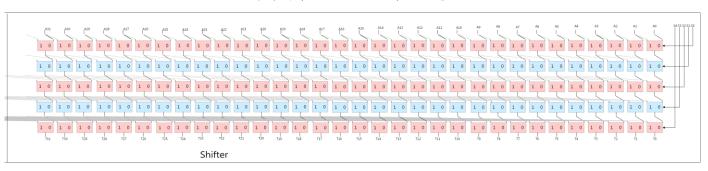
(二) Shifter(組合邏輯):

1. 設計重點

沿用 MidTerm Project 的 Shifters,主要目的還是實現 SLL,邏輯 左移的運算,用 Shamt 位移 in,將結果紀錄在 out。

2. 說明

- 用 5-bits 控制位移量,因為 2 的 5 次方最多可位移 31 bits,故以 5-bits 做為控制位移的,若要位移 32 bits 則全部補 0。
- 以一個 5bits 的線接收 Shamt 的前 5bits 判斷位移量,從第 1 個 bits 判斷到第 5 個 bits 依序可位移 1、2、4、8、16。



(三) Register_File:

1. 設計重點

Register_File 主要在實現 R-type、I-type 各指令時,用來存放的暫存器。

2. 說明

- 我們依照講義上給的圖下去設計 Register_File, R-type 指令,需要用到 RN1、RN2、WN 三個 input, RN1、RN2 分別讀入 rs、rt 暫存器的值,並將 RN1值->RD1, RN2值->RD2,之後進入 ALU 進行運算,之後運算結果傳回 WD, WD 在寫回 WN 暫存器(rd)。
- I-type 指令則有分三類,因為 I-type 指令只有兩個暫存器,若是 lw 指令,RegWrite 會設為 1,需要 RN1、WN 兩個 input,RN1 讀入 rs 暫存器的值,並將 RN1->RD1,然後 offset 進行有號數擴充,與 RD1 一起放入 ALU 運算,WD則接收 Data Memory 的 RD,然後將 WD->WN(rt)。
- 若是 sw 指令, RegWrite 會設為 0, 需要 RN1、RN2 兩個 input, RN1 讀入 rs 暫存器的值, 並將 RN1->RD1, RN2 讀 入 rt 暫存器的值, 並將 RN2->RD2, 然後 offset 進行有號數 擴充,與 RD1 一起放入 ALU 運算, 然後 RD2 將值給 Data

Memory 的 WD,然後 ALU 將運算結果的位址給 Data Memory 的 ADDR。

● 若是 beq 指令, RegWrite 會設為 0, 需要 RN1、RN2 兩個 input, RN1 讀入 rs 暫存器的值,並將 RN1 值->RD1, RN2 讀入 rt 暫存器的值,並將 RN2 值->RD2,之後進入 ALU 運算, offset 進行有號數擴充,並在左移兩位元達到乘以 4 的效果,若 RD1 = RD2,則將 PC+4 與有號數擴充後的值相加回傳給 PC,RD1!= RD2,則只將 PC+4 的值回傳給 PC。

(四) Signal_Extend:

1. 設計重點

Signal_Extend 主要在執行 I-type 指令時,將其 16 bits 的 offset 進行有號數擴充成 32 bits。

(五) Data Memory:

1. 設計重點

Data Memory 主要在於判斷 I-type 的 lw、sw,是否需要寫入記憶體或從記憶體中讀取資料。

2. 說明

- 我們照著講義給的架構圖下去設計,如果是 lw 指令, MemWrite 會設為 0, MemRead 會設為 1,從 MemRead = 1 我們可以判斷,這時我們需要將位址讀入暫存器,因此將從 ALU 讀到 ADDR 的位址(rs + offset),透過 RD 寫回 Register File 的 WD。
- 如果是 sw 指令,MemWrite 會設為 1,MemRead 會設為 0,從 MemWrite = 1 我們可以判斷,這時我們需要將位址寫入記憶體,因此我們將從 ALU 讀到 ADDR 的位址(rs + offset),並把 RD2 的值給 WD,之後 WD 將值寫入 ADDR 的位址。

(六) Mips_Pipeline:

1. 設計重點

將各項 module 建立並執行, Pipeline 主要將 Datapath 分成 5-stage, 分別為 IF、ID、EX、MEM、WB, 我們設計 4個暫存器, 分別為 IF_ID、ID_EX、EX_MEM、MEM_WB, 利用這 4個暫存器達到永遠可讀,特定時間可寫的效果。

2. 說明

我們依照講義給的 Datapath 下去設計,第一階段為 IF,主要在做的事為 Instruction Fetch,從 Instruction Memory 中讀取指令,並將 PC+4,將 IF 階段的結果存在 IF_ID 暫存器裡。下一階段為ID,主要在進行 Instruction decode,從 ID_IF 暫存器取出 IF 的結

果,將指令解碼,判斷指令是 I-type、R-type、J-type,將 ID 階段的結果存在 ID_EX 暫存器裡。下一階段為 EX,主要在做的事為 Execute,從 ID_EX 暫存器取出 ID 的結果,執行指令的運算,若 指令為 BEQ,則在此階段會判斷結果,若是相等,Zero = 1,則 會在此階段將 PC+4 與 offset 擴充偏移完的值相加回傳給 PC,若 不相等,則回傳 PC+4 給 PC,運算完後將 EX 階段的結果存在 EX_MEM 暫存器裡,下一階段為 MEM,主要在做的事 Memory Access,從 EX_MEM 暫存器裡取出 EX 的結果,看是否要進行存取,將結果存入 MEM_WB 暫存器裡,下一階段 WB,主要在進行 Write Back,從 MEM_WB 取出結果,利用多工器判斷是否要 寫回。

(七) ALU Control

ALU Control 主要在控制 ALU,以輸入的 input 決定訊號,再依訊號決定執行哪項功能。

(八) Control Unit

Control Unit 會進行解碼,判斷是 R-type 指令還是 I-type 指令,決定各個多工器的訊號為 1 還是 0、以及給 ALU Control 一個 ALU OP。

ALUOP / Ctl / Ctl 轉二進位

10 / 0 / 000000 R-type: 11 / 18 / 010010 SLTI: LW 00 / 35 / 100011 SW 00 / 43 / 101011 BEQ : 01 / 4 / 000100 J 01 / 2 / 000010 3 / 000011 JAL 01 /

(九) 2-1 MUX(組合邏輯)

接收 Control Unit 給的訊號。

(十) NOP 指令

- 為了避免 hazard,在遇到 J-type 指令、BEQ、連續指令使用到相同的暫存器,這時需要給 NOP,避免 hazard 的情況。
- 指令輸入 00000000, CPU 不做任何事。

(+-)TB_Pipeline

1. 設計重點

TB_Pipeline 主要目的為一個測試平台,從 Testbench 輸入,驗證 各功能的運算結果是否正確。

三、結果

1. add \$s1(r17,2), \$s0(r16,1), \$s1, 0

(rs) (rt) (rd)

20

88

30

02

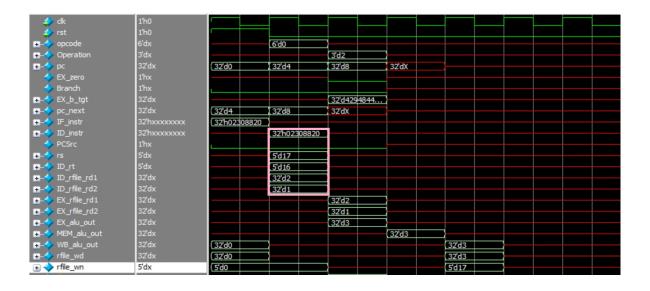
A -0.	1'h0								
∳ dk	1'h0				-			\vdash	
rst .	[E300]			 				_	
	6'dx		6'd0						
+	3'dx			3'd2	}				
₽ − ♦ pc	32'dx	(32'd0	32'd4	32'd8	32'dX				
EX_zero	1'hx								
Branch	1'hx	<u> </u>							
₽- ∜> EX_b_tgt	32'dx			32'd4294844	}				
	32'dx	32'd4	32'd8	32'dX					
🙌 IF_instr	32'hxxxxxxxxx	32'h02308820							
📭 🥎 ID_instr	32'hxxxxxxxxx		32'h02308820	} 					
PCSrc	1'hx				_				
+⊸∳rs	5'dx		5'd17						
₽-∜ ID_rt	5'dx		5'd16						
+	32'dx		32'd2						
+	32'dx		32'd1	<u> </u>					
+	32'dx			32'd2					
+	32'dx			32'd1					
₽-∜ EX_alu_out	32'dx			32'd3					
#	32'dx				32'd3				
	32'dx	32'd0				32'd3			
	32'dx	32'd0				32'd3			
+ 🔷 rfile_wn	5'dx	(5'd0		} 		5'd17			

電腦讀取會以 Little Endian,因此會顯示 02308820。

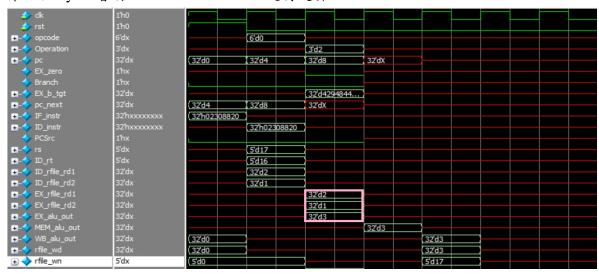
第一個 cycle 先 Fetch 指令(粉紅框框的位置)。



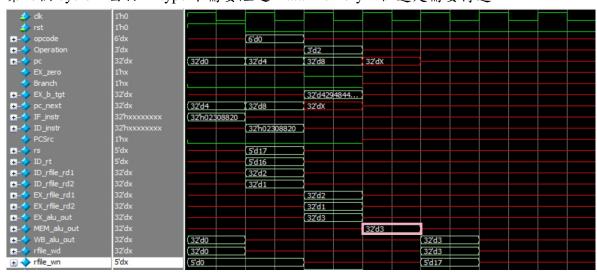
第二個 cycle 會將指令解碼,找到 $rs = 17 \cdot rt = 16$ 兩者為暫存器的位置。 $rd1 = 2 \cdot rd2 = 1$,兩者為暫存器的值。



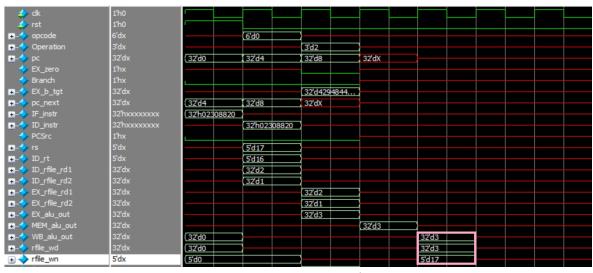
第三個 cycle 會將 rd1、rd2 放入 ALU 進行運算。



第四個 cycle,由於 R-type 不需要經過 Data Memory,但還是需要傳遞。



第五個 cycle,要將 ALU 計算完的值 Write Back 給 rd 暫存器,這時 rd 暫存器的位置為 17,值為 3。



因為 R-type 指令的運作幾乎相同,只差在 ALU 運算的結果,因此我們以 add 為例。

```
2. lw $s1(17,2), $t7(15,21), 0
```

00

00

2F

8E

// nop

00

00

00

00

sw \$zero, \$s2, 24

18

00

12

AC

🥎 opcode	6'dx		6'd35	(6'd0	6'd43	3			
Operation	3'bxxx			3'b010	3'b011	3ъ010			
- ♦ pc	32'd0	(32'd0	, 32'd4	32'd8	32'd12	32'd16	32'dX		
⊢ <mark>-∜></mark> EX_b_tgt	32'dx			32'd4	, 32'd8	32'd108			
pc_next	32'd4	32'd4	32'd8	32'd12	32'd16	32'dX			
-🥎 IF_instr	32'h8e2f0000	32'h8-2f0000	32'h000000000	32'hac120018					
→ ID_instr	32'hxxxxxxxxx		32'h8e2f0000	32'h000000000	32'hac120018	3			
♦ PCSrc	1'h0								
- ∜ rs	5'dx		5'd17	5'd0					
◆ ID_rt	5'dx		(5'd15	5'd0	, 5'd 18				
-🧇 ID_rfile_rd1	32'dx		32'd2	32'd0		3			
-🥎 ID_rfile_rd2	32'dx		32'd21	32'd0	, 32'd3	3			
<pre>EX_rfile_rd1</pre>	32'dx			32'd2	32'd0				
- /> EX_rfile_rd2	32'dx			32'd21	, 32'd0	32'd3			
EX_extend_immed	32'dx			32'd0		32'd24			
-🧇 EX_alu_out	32'dx			32'd2	32'd0	32'd24	=		
	32'dx				(32'd2	32'd0	32'd24	\Rightarrow	
-MEM_dmem_rdata	32'dx				(32'd256	3			
- ∜ rfile_wd	32'd0	(32'd0	3			32'd256	32'd0	\Rightarrow	
⊢ ∜ rfile_wn	5'd0	(5'd0				(5'd15	5'd0	5'dX	

我們把這三個指令的 cycle 一起分解。

第一個 cycle 先 fetch 指令(LW、粉紅框框的位置)。

opcode	6'dx		6'd35	(6'd0	6'd43	3			
Operation	3'bxxx			3'b010	(3b011	3'b010	\Rightarrow ——		
p c	32'd0	(32'd0	32'd4	32'd8	32'd12	32'd16	32'dX		
EX_b_tgt	32'dx			32'd4	32'd8	32'd108	\Rightarrow		
pc_next	32'd4	32'd4	32'd8	32'd12	32'd16	32'dX			
> IF_instr	32'h8e2f0000	32'h8e 2f0000	32'h000000000	32'hac 1200 18	}—				
ID_instr	32'hxxxxxxxxx		32'h8e2f0000	32'h000000000	32'hac120018	3			
> PCSrc	1'h0								
rs	5'dx		5'd17	5'd0					
D_rt	5'dx		5'd15	, 5'd0	, 5'd 18	1			
ID_rfile_rd1	32'dx		32'd2	32'd0					
ID_rfile_rd2	32'dx		32'd21	32'd0	32'd3				
> EX_rfile_rd1	32'dx			32'd2	32'd0				
> EX_rfile_rd2	32'dx			32'd21	32'd0	32'd3	=		
> EX_extend_immed	32'dx			(32'd0		32'd24			
> EX_alu_out	32'dx			32'd2	32'd0	32'd24			
MEM_alu_out	32'dx				32'd2	32'd0	32'd24		
MEM_dmem_rdata	32'dx				32'd256				
rfile_wd	32'd0	(32'd0				32'd256	32'd0		
🔷 rfile_wn	5'd0	(5'd0		} 		(5'd15	1 5'd0	5'dX	

第二個 cycle 將 LW 指令解碼,並 fetch NOP 指令,因為 I-type 只有兩個暫存器,而 LW 只用到 rdl,因此我們這時只需要看 rs=17 的位置,rdl=2 的值。

🛶 opcode	6'dx		6'd35	(6'd0	6'd43	1 -			
	3'bxxx			3'b010	(3'b011	3'b010			
- ∜ pc	32'd0	(32'd0	32'd4	32'd8	32'd12	32'd16	32'dX		
-🥎 EX_b_tgt	32'dx			32'd4	32'd8	32'd108			
-🥎 pc_next	32'd4	(32'd4	32'd8	32'd12	32'd16	32'dX			
-∜ IF_instr	32'h8e2f0000	32'h8e2f0000	32'h000000000	32'hac120018					
-🧇 ID_instr	32'hxxxxxxxxx		32'h8e2f0000	32'h000000000	32'hac120018				
♦ PCSrc	1'h0								
- √ rs	5'dx		5'd17	5'd0					
◆ ID_rt	5'dx		5'd15	5'd0	5'd18	3			
ID_rfile_rd1	32'dx		32'd2	32'd0					
	32'dx		32'd21	32'd0	32'd3				
	32'dx			32'd2	32'd0				
	32'dx			32'd21	32'd0	32'd3			
EX_extend_immed	32'dx			32'd0		32'd24			
	32'dx			32'd2	32'd0	32'd24			
✓ MEM_alu_out	32'dx				32'd2	32'd0	32'd24		
- MEM_dmem_rdata	32'dx				32'd256				
→ rfile_wd	32'd0	(32'd0				32'd256	32'd0		
🛶 rfile_wn	5'd0	5'd0				5'd15	5'd0	5'dX	

第三個 cycle LW 進行運算,並 fetch SW 指令,我們給的立即值為 0,因此 ALU 進行運算後的位置為 2 。



第四個 cycle SW 進行解碼,SW 用到的是 rs、rt 暫存器,rs = 0、rt = 18,rd1 = 0、rd2 = 3,這時 LW 正在進行 Memeory Access,我們從 ADDR 讀到暫存器的位置 = 2,得到的記憶體的資料 = 256。

🥎 opcode	6'dx		6'd35	(6'd0	6'd43	-			
Operation	3'bxxx			3'b010	3'b011	3'b010	=		
> pc	32'd0	(32'd0	32'd4	32'd8	32'd12	32'd16	32'dX		
EX_b_tgt	32'dx			32'd4	32'd8	32'd108			
pc_next	32'd4	32'd4	32'd8	32'd12	32'd16	32'dX			
F_instr	32'h8e2f0000	32'h8-2f0000	32'h000000000	32'hac120018					
ID_instr	32'hxxxxxxxxx		32'h8e2f0000	32'h000000000	32'hac120018	1——			
PCSrc PCSrc	1'h0								
rs	5'dx		(5'd17	5'd0					
ID_rt	5'dx		5'd15	5'd0	5'd18				
ID_rfile_rd1	32'dx		32'd2	32'd0		1			
ID_rfile_rd2	32'dx		32'd21	32'd0	32'd3				
EX_rfile_rd1	32'dx			32'd2	32'd0				
EX_rfile_rd2	32'dx			32'd21	32'd0	32'd3			
EX_extend_immed	32'dx			(32'd0		32'd24			
EX_alu_out	32'dx			32'd2	32'd0	32'd24			
MEM_alu_out	32'dx				32'd2	32'd0	32'd24	=	
MEM_dmem_rdata	32'dx				32'd256				
rfile_wd	32'd0	32'd0	3			32'd256	32'd0	=	
rfile_wn	5'd0	5'd0				(5'd15	5'd0	5'dX	

第五個 cycle SW 進行運算, SW 的立即值為 24, 因此 ALU 計算後的位置為 24, LW 將計算後暫存器 Write Back 回 Register file 的 WN(rt)。

		_								
4	opcode	6'dx		6'd35	(6'd0	(6'd43	}——			
4	Operation	3'bxxx			3'b010	3'b011	3'b010			
4	рс	32'd0	(32'd0	32'd4	32'd8	32'd12	32'd16	32'dX		
4	EX_b_tgt	32'dx			32'd4	32'd8	32'd108			
4	pc_next	32'd4	32'd4	32'd8	32'd12	32'd16	32'dX			
H	IF_instr	32'h8e2f0000	32'h8e 2f0000	32'h000000000	32'hac120018					
4	ID_instr	32'hxxxxxxxxx		32'h8e2f0000	32'h00000000	32'hac120018				
4	PCSrc	1'h0								
-	rs	5'dx		(5'd17	5'd0					
4	ID_rt	5'dx		5'd15	5'd0	5'd18				
4	ID_rfile_rd1	32'dx		32'd2	32'd0					
H	ID_rfile_rd2	32'dx		32'd21	32'd0	32'd3				
4	EX_rfile_rd1	32'dx			32'd2	32'd0				
4	EX_rfile_rd2	32'dx			32'd21	32'd0	32'd3			
4	EX_extend_immed	32'dx			32'd0		32'd24			
4	EX_alu_out	32'dx			32'd2	32'd0	32'd24			
4	MEM_alu_out	32'dx				(32'd2	32'd0	32'd24		
4	MEM_dmem_rdata	32'dx				32'd256				
4	rfile_wd	32'd0	(32'd0				32'd256	32'd0	\Rightarrow	
4	rfile_wn	5'd0	(5'd0				5'd15	5'd0	5'dX	

第六個 cycle SW 進行 Memory Access, 我們在 Data Memory 裡將 rd2 的值 = 3(wd)寫入暫存器(addr)的位置。

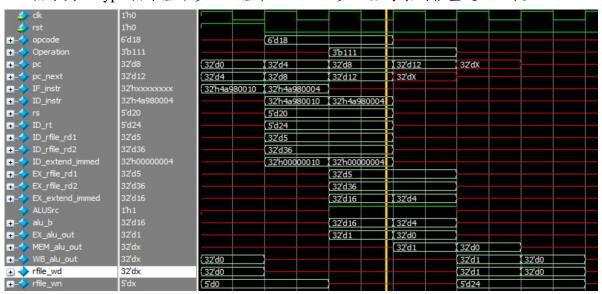
-	opcode	6'dx			6'd35		(6'd0		6'd43				
a -4	Operation	3'bxxx					3'b010		(3'b011	3ъ010			
u-	рс	32'd0	(32'd0		32'd4		32'd8		32'd12	32'd16	32'dX		
a -�	EX_b_tgt	32'dx					32'd4		32'd8	32'd108			
1 -4	pc_next	32'd4	(32'd4		32'd8		32'd12		32'd16	32'dX			
3 -4	IF_instr	32'h8e2f0000	(32'h8e	2f0000	32'h000	00000	32'hac12	0018					
n-4	ID_instr	32'hxxxxxxxxx			32'h8e2	f0000	32'h0000	0000	32'hac120018	3 ———			
4	PCSrc	1'h0											
1	rs	5'dx			5'd17		5'd0			3			
-	ID_rt	5'dx			5'd15		5'd0		5'd18	3			
1	ID_rfile_rd1	32'dx			32'd2		32'd0			3			
	ID_rfile_rd2	32'dx			32'd21		32'd0		32'd3	3———			
H-4	EX_rfile_rd1	32'dx					32'd2		32'd0		\Rightarrow		
3-4	EX_rfile_rd2	32'dx					32'd21		32'd0	32'd3	\Rightarrow		
H -�	EX_extend_immed	32'dx					32'd0			32'd24	\Rightarrow		
H -�	EX_alu_out	32'dx					32'd2		32'd0	32'd24	\Rightarrow		
e 🔷	MEM_alu_out	32'dx							32'd2	, 32'd0	32'd24		
-	MEM_dmem_rdata	32'dx							32'd256	3 ———			
B -4	rfile_wd	32'd0	32'd0		}					32'd256	32'd0		
e-4	rfile_wn	5'd0	(5'd0							(5'd15	5'd0	5'dX	

3. slti \$s4, \$t8 #16

4A

slti \$s4, \$t8 #1

Slti 指令與 R-type 指令差不多,只差在 ALUSrc 多工器時讓偏移量進入比較。



4. beq rs rt imm beq s1, s1,#5

nop

```
00
00
00
00
nop
00
00
00
00
add $s2(r18,3), $s0(r16,1), $s2(r18,4)
90
50
sub $s1(r17,1), $s0(r16,1), $s1(r17,0)
88
30
02
sub $s3(r19,4), $s0(r16,1), $s3(r19,3)
22
98
70
or s4(r20,5), s0(r16,1), s4(r20,5)
25
A0
90
02
add $s2(r18,4), $s0(r16,1), $s2(r20,5)
20
90
50
02
sub $$1(r17,0), $$0(r16,1), $$1(r17,-1)
88
30
sub $s3(r19,3), $s0(r16,1), $s3(r19,2)
22
98
70
02
```

BEQ指令第一階段先 fetch 指令(粉紅框框的部分)

	1'h0						
✓ rst	1'h1						
+	6'dx		(6'd4	(6'd0			
+	3'bxxx			(3'b110	3'b011		3'b001
⊨ -∲ pc	32'd0	32'd0	32'd4	32'd8	32'd24	32'd28	32'd32
├ ∳ pc_next	32'd4	32'd4	32'd8	32'd24	32'd28	32'd32	32'd36
►◆ IF_instr	32'h12310005	32'h12310005	32'h000000000		32'h0290a025	32'h02509020	32'h02308822
I → ID_instr	32'hxxxxxxxxx		32h12310005	32'h000000000		32'h0290a025	32'h02509020
⊢ ∳rs	5'dx		(5'd17	, 5'd0		5'd20	5'd18
⊢ ∳ ID_rt	5'dx		5'd17	, 5'd0		5'd16	
►◆ ID_rfile_rd1	32'dx		32'd2	32'd0		32'd5	32'd3
└─️ ID_rfile_rd2	32'dx		(32'd2	32'd0		32'd1	
	32'dx		32'd5	32'd0		32'd4294942	32'd4294938
	32'dx			32'd2	32'd0		32'd5
	32'dx			32'd2	32'd0		32'd1
	32'dx			32'd5	32'd0		32'd4294942
ALUSrc	1'hx						
- ∜ alu_b	32'dx			32'd2	X 32'd0		32'd1
⊢🧇 EX_alu_out	32'dx			32'd0	32'd0		32'd5
EX_zero	1'hx						
Branch	1'h0						
♦ PCSrc	1'h0						
🛶 b_offset	32'dX	32'dX		32'd20	【32'd0		32'd4294869
FA EX_pc_incr	32'dx			32'd4	32'd8	32'd12	32'd28
!∜ EX_b_tgt	32'dx			32'd24	32'd8	32'd12	32'd4294869

第二階段進行解碼。

另一 陷权连行牌								
	1'h0							
🚣 rst	1'h1		i					
- → opcode	6'dx			6'd4	(6'd0			
■ Operation	3'bxxx				3'b110	3'b011		3'b001
⊕ - ∲ pc	32'd0	32'd0		32'd4	32'd8	32'd24	32'd28	32'd32
-	32'd4	32'd4		32'd8	32'd24	32'd28	32'd32	32'd36
IF_instr □ □ □ □ □ □ □ □ □ □ □ □ □	32'h12310005	(32'h1231	0005	32'h00000000		32'h0290a025	32'h02509020	32'h02308822
ID_instr ID_instr	32'hxxxxxxxxx			32'h12310005	32'h000000000		32'h0290a025	32'h02509020
 ∕y rs	5'dx			5'd17	5'd0		5'd20	5'd18
 → ID_rt	5'dx			5'd17	5'd0		5'd16	
 → ID_rfile_rd1	32'dx			32'd2	32'd0		32'd5	32'd3
ID_rfile_rd2 □ □ □ □ □ □ □ □ □ □ □ □ □	32'dx	_		32'd2	32'd0		32'd1	
∓ -∜ ID_extend_immed	32'dx			32'd5	32'd0		32'd4294942	32'd4294938
EX_rfile_rd1	32'dx				32'd2	32'd0		32'd5
EX_rfile_rd2	32'dx				32'd2	32'd0		32'd1
EX_extend_immed	32'dx				32'd5	32'd0		32'd4294942
ALUSrc	1'hx		-					
📭 🔷 alu_b	32'dx				32'd2	32'd0		32'd1
→ EX_alu_out	32'dx	_			32'd0	32'd0		32'd5
EX_zero	1'hx							
Branch	1'h0							
PCSrc	1'h0		Ļ					
 - → b_offset	32'dX	32'dX			32'd20	32'd0		32'd4294869
	32'dx				(32'd4	32'd8	32'd12	32'd28
	32'dx				32'd24	32'd8	32'd12	32'd4294869

第三階段進行運算,判斷 rd1、rd2 的值是否相等,由圖可知相等,偏移量 = 5,因為 PC 計算 increment 單位為 4,所以 offset 為 20,原本 PC 為 0,將 offset 與 PC+4 相加,得出 24,回傳給 PC。

d dk	1'h0							
The state of the s	1'h1							
THE RESERVE TO THE PARTY OF THE	6'dx			(6'd4	Ĭ 6'd0			
The state of the s	3'bxxx			644	(3'b110	X3'b011		X 3'b001
+-4 pc	32'd0	(32'd0	-	I 32'd4	X 32'd8	32'd24	32'd28	32'd32
	32'd4	(32'd4		32'd8	32'd24	32'd28	32'd32	32'd36
THE RESERVE OF THE PARTY OF THE	32'h12310005	(32h1231	0005	1 32'h00000000	, 52021	32'h0290a025		32'h02308822
	32'hxxxxxxxxxx	SETTIEST	3003		X 32'h000000000	, 32110230d023	32'h0290a025	
- Company of the Comp	5'dx			(5'd17	X 5'd0		5'd20	5'd18
	5'dx			5'd17	X 5'd0		5'd16	7 3 4 10
	32'dx			32'd2	X 32'd0		1 32'd5	1 32'd3
	32'dx		_	32'd2	32'd0		32'd1	13233
	32'dx			(32'd5	X 32'd0		STATEMENT STATEM	32'd4294938
EX_rfile_rd1	32'dx			3233	32'd2	32'd0	, 32 d 123 13 12111	32'd5
The second secon	32'dx				32'd2	32'd0		32'd1
EX extend immed	32'dx		_		32'd5	32'd0		32'd4294942
→ ALUSrc	1'hx		_		3203	3230		320 123 13 12111
	32'dx				(32'd2	I 32'd0		32'd1
	32'dx				(32'd0	32'd0		32'd5
- Company of the Comp	1'hx		_		3230	32.00		10200
→ Branch	1'h0					1		
	1'h0		i i					
+- D_oπset	32'dX	32'dX			32'd20	32'd0		32'd4294869
	32'dx	DEGA			32'd4	32'd8	Ĭ 32'd12	32'd28
The state of the s	32'dx				32'd24	32'd8	32'd12	32'd4294869

```
5. sll t0(rd,r8,8), t1(rt,r9,9), 2 => ans = 36
80
40
09
00
lw $s1(17,2), $t7(15,), 0
00
00
2F
8E
sub $s2, $s0, $s2
22
90
50
02
12 add $s1, $s0, $s1, 0
20
88
30
02
jal 7
07
00
00
0c
nop
00
00
00
00
nop
00
00
00
add $s2(r18,3), $s0(r16,1), $s2(r18,4)
20
90
```

下面是J和Jal都會做的事。

當第二個 cycle 時,會把 instrc[25:0]左移兩位元得到 28 位元的 jumpoffset 再把 pc+4[31:28]與 jumpoffset 做 Concat 得到 jumpaddr 同時會把 control Unit 會傳出 Jump 指令給 JUMPMUX 去選擇 最後設定 PC_next。

	1h1												
🏄 rst	1'h0												الكام
⊕ → opcode	6'dx		(6'd0	6'd35	[6'd0		[6'd3	[6'd0					
⊞- pc	32'd36	(32'd0	32'd4	32'd8	32'd12	32'd16	32'd20	32'd28	32'd32	32d36	32'dX		
pc_next	32'dX	(32'd4	32'd8	32'd12	32'd16	32'd20	32'd28	32'd32	32'd36	32 dX			
■- IF_instr IF_instr IF instr IF inst	32 hxxxxxxxxx	32'h00094080	32'h8e2f0000	32h02509022	32'h02308820	32/h0c000007	32'h000000000	32'h02509020					
☐→ ID_instr	32*hxxxxxxxxxx		32'h00094080	32h8e2f0000	32h02509022	32/h02308820	32h0c000007	32'h000000000	32h02509020				
ump_addr jump_addr	32'dX	32'dX	32'd2425344	32'd12320768	32'd21119112	32'd12722304	32'd28	32'd0	32'd21119104	32 dX		32'dX	
⊕ - ∜ branch_addr	32'd40	32'd4	32'd8	32'd12	32'd16	32'd20	32'd24	32'd32	32'd36	32 d40			
🥠 Jump	1'hx												
₩B_pc_incr	32'd20	(32'd0		3——		32'd4	32'd8	32'd12	32'd16	32 d20	32'd24	32'd32	32'd36
🥠 Jal	1h1												
⊞ - ∜ rfile_wd	32'd20	(32'd0				32'd36	32'd256	32'd2	32'd3	32 d20	32'd0	32'd3	
→ rfile_wn	5'd31	(5'd0				5'd8	5'd15	5'd18	5'd17	5'd31	5'd0	5'd18	

Jal 在跳躍的同時會把 Jal 的下道指令 PC 值儲存到 r31 暫存器。 當第五個 cycle 時,MEM_WB 暫存器會傳給 WB Jal 和 PC+4,Jal 為 1 將寫回的 值設為 PC+4,同時將寫回的暫存器設為 31。

₫y ck	1h1												
✓ rst	1'h0												
⊞ - ∮ opcode	6'dx	_	(6'd0	6'd35	[6'd0		6'd3	[6'd0					
⊞- ∳ pc	32'd36	(32'd0	32'd4	32'd8	32'd12	32'd16	32'd20	32'd28	32'd32	32 d36	32'dX		
pc_next	32'dX	(32'd4	32'd8	32'd12	32'd16	32'd20	32'd28	32'd32	32'd36	32 dx			
■- IF_instr	32/hxxxxxxxxx	32'h00094080	32h8e2f0000	32h02509022	32h02308820	32h0c000007	32h00000000	32h02509020					
ID_instr ID_instr	32'hxxxxxxxxxx		32h00094080	32h8e2f0000	32h02509022	32h02308820	32h0c000007	32h00000000	32'h02509020				
⊕ -∜ jump_addr	32'dX	32'dX	32'd2425344	32'd12320768	32'd21119112	32'd12722304	32'd28	32'd0	32'd21119104	32 dX		32'dX	
→ branch_addr → branch_addr	32'd40	(32'd4	32'd8	32'd12	32'd16	32'd20	32'd24	32'd32	32'd36	32 d40	\Rightarrow		
🥎 Jump	1'hx							1					
₩B_pc_incr	32'd20	(32'd0				32'd4	32'd8	32'd12	32'd16	32 d20	32'd24	32'd32	32'd36
🥠 Jal	1h1												
→ rfile_wd	32'd20	32'd0	3——			32'd36	32'd256	32'd2	32'd3	32d20	32'd0	32'd3	
+ > rhie wn	5'd31	(5'd0				5'd8	5'd15	5'd18	5'd17	5'd31	[5'd0	5'd18	

四、討論

經過了 Midterm Project 的練習,這次在討論上思路就比較清晰,也比較有方向可以進行,Final Project 的部分元件,是利用的期中寫好的東西,而 Datapath、切 pipeline 等比較新的東西,就要從老師上課所教的、講義給的架構圖去慢慢討論、思考,一步一步完成 Final Project。

五、結論

我們的結論是相比於 Midterm Project,我們對於在遇到比較需要思考的問題,像是要建立在 clock 時序邏輯上來實現循序邏輯、blocking 跟 non-blocking 的使用時機、從軟體的角度轉變成從硬體的角度等等,都比較有概念可以去撰寫,但在實現循序邏輯這方面,還是會遇到些許的小瓶頸。而這次面對 Pipeline 這個新的東西、難度較高的問題,我們也是在花了許久時間,從上課吸收老師上課內容、理解講義上的架構流程等等…,然後嘗試著把 Final Project 的功能一一

實現。而遇到比較小的問題像是對於 Verilog 語法的不熟悉,可能接觸 C、C++的時間較長,有時候還是會不小心用到 C 的語法、還有在 Verilog 撰寫時接線的部分,有時沒接好,導致輸出的 waveform 出現紅線…,而隨著我們練習的次數增加,就能一步步改正上面提到的小錯誤。

心得

雖然經過了 Midterm Project 的磨練,但我們這次在寫 Final Project 時還是遇到許多多的難題,因為這次要寫的是一整個 CPU,因此要考慮到的東西變多、也變得比較複雜,老師在上課有說到,畫圖是設計硬體的第一步,也從這句話能了解到 Datapath 的重要性,幸好老師在上課時,對於 Datapath 這一段有一再地強調、重複,因此對於各元件功能的 Datapath,我們就比較沒有那麼不熟悉,比較好掌握,也比較好實現,然而像是 Pipeline 這東西就讓我們困擾了很久,在聽老師在上課時的講課內容時,雖然能理解 Pipeline 如何運作,還有怎麼實現的例子等等,但在實際撰寫 Verilog 程式碼時,還是遇到很多問題,像是實現 Pipeline 的概念,在不同 stage 同時做不同指令,節省時間以提升效率的時候,會因為沒有抓到前幾個 stage 的指令,然後就會發生一些像是在執行後面階段時,前面的可能無法 fetch 到指令,幸好後來與同學交流討論後解決了。最後,很開心這學期在這計組這門課,學到了非常豐富的知識,雖然遇到很多困難、錯誤,但也因為這些錯誤,讓我們有更多練習機會,也學習的更深入,再經過兩次的 Project 後,也讓我們更加理解這門課程的學習內容。

六、未來展望

終於完成這學期計組的兩次 Project,從一開始對這門課感到畏懼,到最後寫完、完全了解這門課的學習內容,經歷了很多挫折,解決了很多困難,雖然很辛苦,但這些辛苦都很值得,也透過計組這門課,點燃了我們對硬體領域的些許興趣,希望之後能在計組這領域繼續學習更多知識。雖然這學期了解的並沒有到很扎實,但還是希望計算機組織能在未來成為我們的一技之長。

七、分工

10527130 陳少洋:部分程式碼、Debug

10527140 初 元:架構圖繪製、Debug

10527132 林亞吟:程式碼、報告結果討論、Debug

10527135 張智欽:撰寫報告分析、報告結果討論、Debug