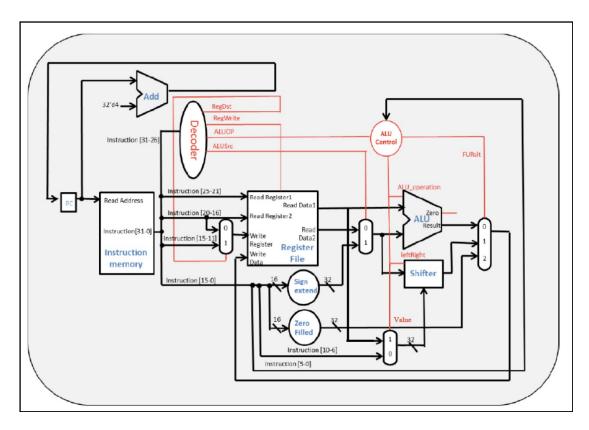
Computer Organization

Architecture diagrams:



Hardware module analysis:

- 1. Adder 功能 : 計算下一個 PC = PC+4 ,因為這次沒有 branch 指令,所以所有的指令完成後下一個指令的位置一律是 PC+4。
- 2. Decoder 功能: 讀取 opcode 並發送各種跟資料輸入有關控制訊號
 - I. Rtype opcode:

```
i. write = 1; // 1 => 要儲存運算結果
```

ii. op = ALUOP_RTYPE;

iii. src = 0; // 0 => 用 Reg 當第二個數入來源

iv. Dst = 1; // 1 => 用 Rd 來儲存

II. Addi opcode:

i. write = 1; // 1 => 要儲存運算結果

ii. op = ALUOP_ADDI;

iii. src = 1; // 1 => 用 Imm 當第二個輸入來源

iv. Dst = 0; // 0 => 用 Rt 來儲存

III. Lui opcode:

i. write = 1; // 1 => 要儲存運算結果

ii. op = ALUOP_LUI;

iii. src = 1; // 1 => 用 Imm 當第二個輸入來源

iv. Dst = 0; // 0 => 用 Rt 來儲存

- 3. ALU Controle 功能:接收 ALUOP 控制訊號並進一步轉換成與資料運算有關的訊號
 - I. Alu operation: 給定每種指令一個對應的 alu 運算方式。
 - II. LeftRight: 1 向左 shift(sll, sllv), 0 向右 shift(srl, srlv), 設計用 alu operation[0]來判斷
 - III. Value :0 用 Imm 來 shift(sll, srl), 1 用\$rs 來 shift(sllv, srlv), 设計用 alu operation[1]來判斷
- 4. Sign extended 功能:複製原訊號的最高位 bit(MSB)到前 16bit
- 5. Zero filled 功能 : 用於不需要計算的 lui,把輸入訊號移至前 16bit,並 將後 16bit 填 0
- 6. Shifter 功能: 將\$rt 依照給定的位移量跟位移方向做位移

Finished part:

Test data1

√ /TestBench/CLK	-1'd1									
<pre>// /TestBench/RST</pre>	-1'd1									
+	32'd25	32'd0		32'd1	32'd2	32'd3	32'd4	32'd5	32'd25	
+	32'd2	32'd2								
+	32'd25	32'd25								
+	-32'd56612843	32'd0	-32'd6039	-32'd6038	-32'd6487	-32'd6487	,-32'd6487	-32'd5661284	3	
	32'd24		32'd4	32'd8	32'd12	32'd16	32'd20	32'd24		
≖ - ∜ /TestBench/rs	5'd5		5'd0		5'd1			5'd5		
II — ♦ /TestBench/rt	5'd0		5'd1	5'd2				5'd0		
II — ♦ /TestBench/rd	5'd5				5'd3	5'd4	, 5'd5			
→ /TestBench/i	32'd31	32'd32		32'd31						

Test data2

√ /TestBench/CLK	1'd0										
/TestBench/RST	1'd0										
- → /TestBench/count	32'd0	32'd0		32'd1	32'd2	32'd3	32'd4	32'd5	32'd6	32'd25	
+	32'd2	32'd2									
∓ - / /TestBench/end_co	32'd25	32'd25									
≖ - /TestBench/instruction	32'd0	32'd0	-32'd6035	-32'd6035	-32'd5405	-32'd5404	-32'd6649	-32'd1073	-32'd6636307)	
-	32'dx		32'd4	32'd8	32'd12	32'd16	32'd20	32'd24	32'd28		
-/ /TestBench/rs	5'dx		5'd0		5'd6		5'd0				
≖ - / /TestBench/rt	5'dx		5'd6	5'd7			5'd9	5'd11			
≖ – ∜ /TestBench/rd	5'dx				5'd8	5'd9	5'd10		5'd12		
I —	32'd32	32'd32		32'd31							

Test data3

/TestBench/CLK	-1'd1										
/TestBench/RST	-1'd1										
+	32'd25	32'd0		32'd1	32'd2	32'd3	32'd4	32'd5	32'd6	32'd25	
II	32'd2	32'd2									
II	32'd25	32'd25									
+-/>/TestBench/instruction	-32'd 10732	32'd0	-32'd603	,-32'd6038	-32'd648	-32'd628	,-32'd6487	-32'd648	-32'd1073217	541	
II	32'd28		32'd4	32'd8	32'd12	32'd16	32'd20	32'd24	32'd28		
II - / /TestBench/rs	5'd0		5'd0		5'd1	5'd2	5'd1		, 5'd0		
II - / /TestBench/rt	5'd7		5'd1	, 5'd2		5'd1	5'd2		5'd7		
II - / /TestBench/rd	5'd6				5'd3	5'd4	, 5'd5	5'd6			
II - ∜ /TestBench/i	32'd31	32'd32		32'd31							

Problems you met and solutions:

其實這次的作業寫 verilog 沒什麼很困難的地方,比較蠢的是我一開始做加減法沒有加上\$signed(),所以算到負數就會錯,主要是要花很多時間搞清楚每個 module 負責的功能、cpu 運作的每一個過程。從解讀指令到控制 alu 運算每一個控制訊號都要了解,花了很多時間。

Summary:

經過這次的作業我認識了一個簡易 Cpu 完整的運作過程,從指令讀取到解碼, 從發送控制訊號到運算,每一個步驟都有更深入的了解。