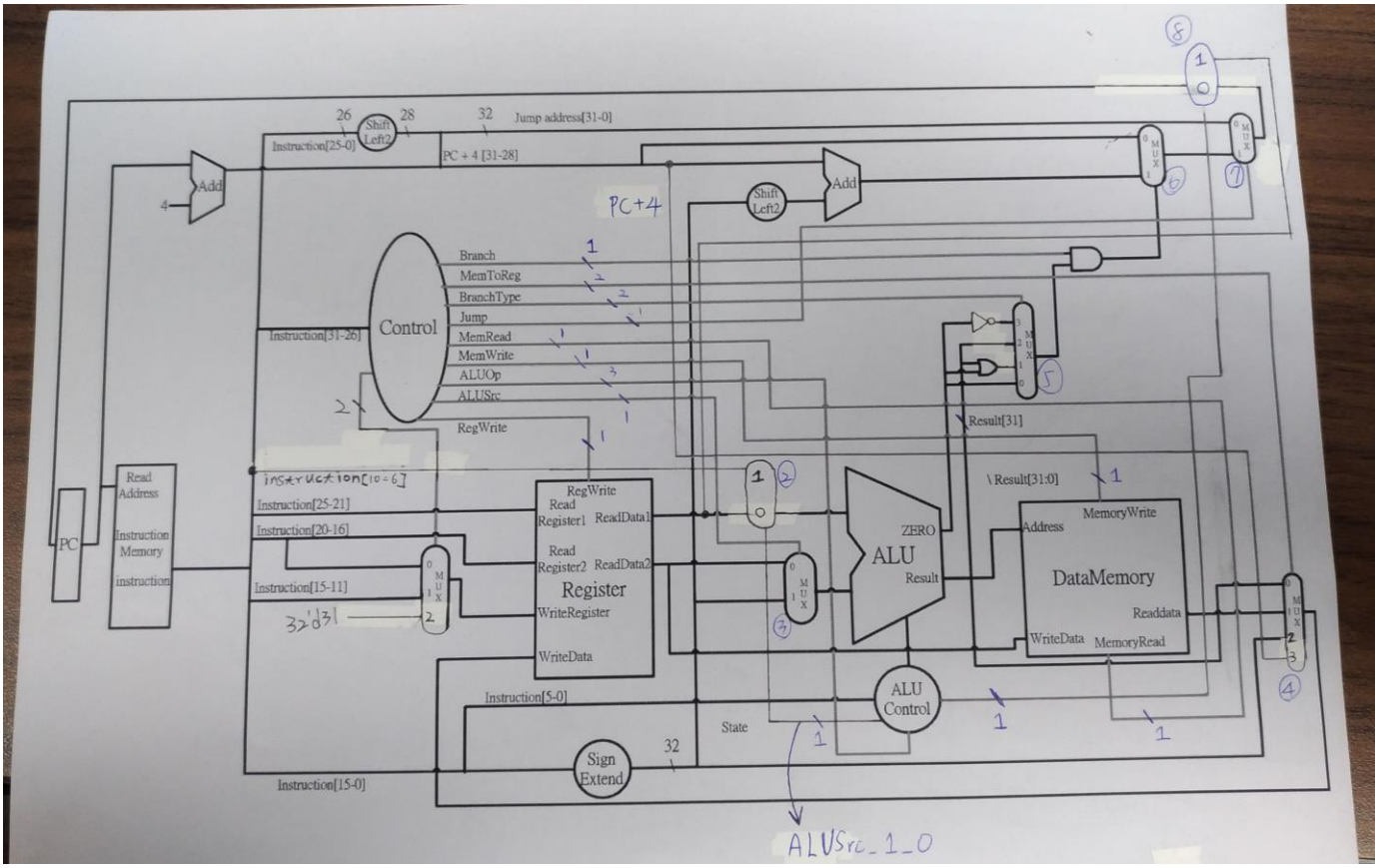
0416315 王定偉、0416005 張彧豪

Computer Organization Lab3

Architecture diagram:



Detailed description of the implementation:

一、 以 Simple_Single_CPU 為主要的模組，會去呼叫這個 simple CPU 所需要的所有模組，包含 PC、IM、RF、Decoder、ALU_Ctrl、Sign_Extend、MUX，ALU、Data_memory 等模組。此 CPU 為一個 clock cycle 執行一個指令。

二、Decoder 模組：

1. 我是根據傳入的 Op filed 去 map 那些輸出的值，以下是我採用的對應表：

instr_op_i	ALU_op_o	Type	MemToReg_o MemRead_o MemWrite_o	RegDst_o RegWrite_o	ALUSrc_o	Branch_o Brach_type_o Jump_o
6’d0	3’b000	R	2’b00 1’b0 1’b0	2’b01 1’b1	1’b0	1’b0 2’b00 1’b0
6’d2	3’b111	J	2’b00 1’b0 1’b0	2’b00 1’b0	1’b0	1’b0 2’b00 1’b1
6’d3	3’b111	JAL	2’b11 1’b0 1’b0	2’b10 1’b0	1’b0	1’b0 2’b00 1’b1
6’d4	3’b001	BEQ	2’b00 1’b0 1’b0	2’b00 1’b0	1’b0	1’b1 2’b00 1’b0
6’d5	3’b010	BNE, BNEZ	2’b00 1’b0	2’b00 1’b0	1’b0	1’b1 2’b11

			1’b0			1’b0
6’d6	3’b001	BLT	2’b00 1’b0 1’b0	2’b00 1’b0	1’b0	1’b1 2’b10 1’b0
6’d7	3’b001	BLE	2’b00 1’b0 1’b0	2’b00 1’b0	1’b0	1’b1 2’b01 1’b0
6’d8	3’b011	ADDI	2’b00 1’b0 1’b0	2’b00 1’b1	1’b1	1’b0 2’b00 1’b0
6’d13	3’b100	ORI	2’b00 1’b0 1’b0	2’b00 1’b1	1’b1	1’b0 2’b00 1’b0
6’d15	3’b101	LI	2’b10 1’b0 1’b0	2’b00 1’b1	1’b1	1’b0 2’b00 1’b0
6’d35	3’b011	LW	2’b01 1’b1 1’b0	2’b00 1’b1	1’b1	1’b0 2’b00 1’b0
6’d43	3’b011	SW	2’b00 1’b0 1’b1	2’b01 1’b0	1’b1	1’b0 2’b00 1’b0

我們的 ALU_op_o 與 instr_op_i 之間並沒有什麼特殊的關係，純粹只是循序的指定。

一、ALU_Control 模組：

1. 這個 ALU_Control 模組跟 Lab2 得非常相似，差異在於這個 ALU_Control 有多一條輸出線為 Junp_type，這條輸出線的用意是要幫助我們實作的 JR 指令用的。
2. 以下為我們採用的對應表：

R-type 以外：

ALUOp_i	Instruction	ALUSrc_1_o	ALUCtrl_o	Jump_type
3’b001	BEQ,BLT,BLE	1’b0	4’b0110	1’b0
3’b010	BNE,BNEZ	1’b0	4’b1110	1’b0
3’b011	Addi,LW,SW	1’b0	4’b0010	1’b0
3’b100	LI	1’b0	4’b0100	1’b0
3’b101	ORI	1’b0	4’b0001	1’b0

這五個指令除了 BEQ、BLT、BLE、BNE、BNEZ、Addi、LW、SW 的 ALUCtrl_o 是拿 lab1 中減法、加法做對應，其它是拿尚未被佔用的指令去指派的。

R-type：

ALUOp_i	Instruction	ALUSrc_1_o	ALUCtrl_o
3’b000	ADD	1’b0	4’b0010
	SUB	1’b0	4’b0110
	AND	1’b0	4’b0000
	OR	1’b0	4’b0001
	SLT	1’b0	4’b0111
	SLTU	1’b0	4’b1111
	MUL	1’b0	4’b0101
	JR	1’b0	4’b1000
	SLL	1’b1	4’b0011
	SLLV	1’b0	4’b0011

R-type 中前五個指令是照 lab1 去指定的，而後面三個指令是拿尚未被佔用的值去指定的。

3. Branch mux 詳細說明：

我們將 branch 指令分成四個種類，第一為 beg 類，beg 指令為此類；第二為 ble 類，ble 指令為此類；第三為 blt 類，blt 指令為此類；最後一個為 bne 類，bne、bnez 指令為此類

的。而圖中第五個 mux 即是從這四類中為一類出來作為最後是否要 branch 的訊號，詳細對應如下：

BrancgType(選擇線)	輸出	對應的指令
2'b00	zero_o	beq
2'b01	zero_o|result_o[31]	ble
2'b10	result_o[31]	blt
2'b11	zero_o	bne,bnez

- zero_o 在 bne、bnez 的情況下輸出 1 代表不為 0，反之代表為 0(在 ALU 內控制)；其餘狀況則是輸出 1 代表為 0，反之代表不為 0。
- ble、blt 用 result 的 MSB 則是要看 src_1 是否 src_2，若小於成立，則 result 的 MSB 必為 1，故可用這種方法來決定 ble、blt 是否要 branch。

4. JAL 實作方法：

JAL 因為要將 PC+4 存回 reg31，故我們將圖中第四個 mux 變成 4-1 的，多的那個輸入室已經算出來的 PC+4，並以原本即為 2bit 的 MemToReg_o 當作選擇線，這樣就可以把 PC+4 傳回 WriteData；同時我們也有修改第一個 mux，將它變成 3-to-1 的，多的輸入為常數 31，且須將原本 1bit 的 RegDst_o 改成 2bit，此舉是為了 WriteRegister 能有正確的寫入位址。第四個 mux 詳細對應如下：

MemToReg_o(選擇線)	輸出
2'b00	result_o
2'b01	MEM_Read_data_o
2'b10	SE_data_o
2'b11	pc_plus_four

第一個 mux 詳細對應如下：

RegDst_o(選擇線，2bit)	輸出
2'b00	instruction_o[20:16]
2'b01	instruction_o[15:11]
2'b10	5'd31

5. JR 實作方法：

JR 的實作我們是多加一個 mux，多加的這個 mux 為第八個 mux，放的位置是在圖中第七個 mux 後面，會放在這麼後面是因為 Jump 是在很後面才決定的，為了避免混亂，放在最後面是最好的選擇，此 mux 為 2-to-1 的 mux，輸入分別是 RSdata_o 及原本沒有 JR 指令時的下一個 PC number，選擇線為 Jump_type，此選擇線只有在 JR 指令下為 1，其餘指令下為 0，此選擇線由 ALU_ctrl 計算，因為 JR 指令為 R-type 的關係且不破壞整體架構的情況下，由 ALU_ctrl 計算它是最好的選擇。

此 mux 詳細對應如下：

Jump_type(選擇線)	輸出
1'b0	pc_number_next
1'b1	RSdata_o

- pc_number_next 為原本沒有 JR 指令時的下一個 PC number。

Problems encountered and solutions:

1. 其實一開始不太知道該怎麼實作 JR 這個指令，因為它是 R-type 而且跟其他 R-type 指令都好像沒有什麼關係，最後看了一下指令的結構以及參考的架構圖，發現可以將 RSdata_o 直接接出來送進一個 mux 做選擇。在實作這個指令時遇到的另外一個問題是那個 mux 的選擇線該從哪裡來，因為它是 R-type 指令的關係，故無法在 Decoder 的部分就產出選擇線的訊號，我們選擇在 ALU_ctrl 那部分產生選擇線 Jump_type 的訊號。最後一個要注意的地方是，因為 JR 指令為 R-type 指令，為了避免寫入垃圾值到 reg0，在 JR 指令下，ALU 的輸出強制為 0，以此避免此狀況的發生。
2. 在測試的 bubble sort 那題的測資時，發現就算按了 run all 結果仍然不對，以一次增加 1000ns 模擬時間的方式 debug，最後發現是 testbench 的強制結束時間太短了，將結束時間改長一點即可看到正確的結果。

Lesson learnt (if any):

1. 四種不同類型 branch 指令是否要 branch 的判斷方法。
2. JR 指令的實作方法。