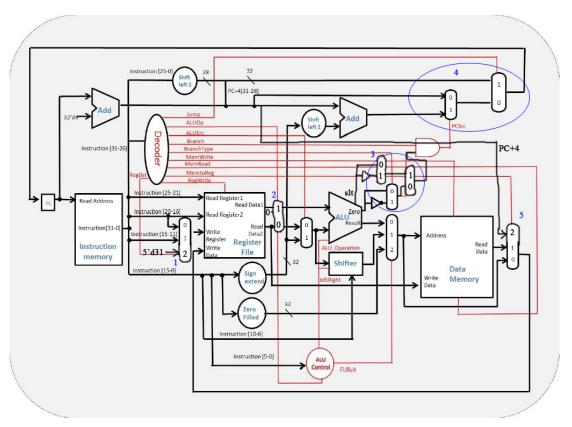
Computer Organization

Architecture diagrams:



Hardware module analysis:

1. 新增的硬體

1:預設 5'd31 的暫存器位置用來讓 jar 寫入\$r31

2: 預設 rt = 0, 用來給 blt, bgez 判斷的 0(防止 r0 被改掉時功能錯誤)

3: 用 2 個 flag(zero, slt)來判斷,再用一個 mux 來選要用哪種 flag 的結果做 branch 判斷

4: 左邊的 mux 是用來判斷在 bne, beq 時有沒有要 branch,如果 PCsrc=1 代表符合條件需要 branch。右邊的 mux 是用來判斷是直接 jump 還是條件判斷後 branch

5: 用於 jar 將 pc+4 傳到 Reg File 寫入

2. 新增的 control signal

Select_Zero_branch = rs=rt 時為 1, rs!=rt 時為 0

Select_Slt_branch = rs<rt 時為 1, rs>=rt 時為 0

Select_branchType: bne, beq 用==判斷的指令為 0, blt, bgez 等用<或>=判斷

的指令為1

RegDst: 新增2用來選擇 5'd31

MemtoReg: Rtype 指令輸出=0, Ltype 指令輸出=1, PC+4輸出=2(jar)

MemRead: 只有 1w=1,其餘為 0

MemWrite: 只有 sw=1,其餘為 0

Finished part:

Test data 1

≨ -	Msgs									
→ /TestBench/CLK	-1'd1									
→ /TestBench/RST	-1'd1									
→ /TestBench/count	32'd600	32'	32'd18	32'd19	32'd20	32'd21	32'd22	32'd23	32'd600	
→ /TestBench/handle	32'd2	32'd2								
→ /TestBench/end_co	32'dx									
→ /TestBench/instruction	-32'd1945894876	-32	-32'd1807	-32'd6031	-32'd1945	-32'd2013	-32'd5639	-32'd1945894	876	
→ /TestBench/pc	32'd124	32'	32'd92	32'd96	32'd100	32'd116	32'd120	32'd124		
→ /TestBench/rs	5'd0	5'd0	5'd2	5'd0			5'd5	5'd0		
→ /TestBench/rt	5'd4	5'd4	5'd5	5'd13	5'd4		5'd3	5'd4		
→ /TestBench/rd	5'd15	5'd8					5'd15			
→ /TestBench/addr	32'd36	32'd20			32'd28			32'd36		
→ /TestBench/data	32'd4	32'd4								
→ /TestBench/i	32'd16	32'd12	3							
→ /TestBench/j	32'd9									
→ /TestBench/k	4'dx									
→ /TestBench/mode	32'd0	32'd0								
→ /TestBench/score	32'd65									
→ /TestBench/initail_e	32'd0									
± → /TestBench/memory	32'd2 -32'd2 32'd	32'd2 -	32'd2 32'd1 32'	d0 32'd1 3	32'd2 -32'd2 3	2'd1 32'd0 32'd	1 32'd4 32'	32'd2 -32'd2 3	2'd1 32'd0 32'd	1 32'd4 32'

Test data 2

<u>)</u>	Msgs														
→ /TestBench/CLK	-1'd1														
√ /TestBench/RST	-1'd1														
→ /TestBench/count	32'd600	32'd600													
± → /TestBench/handle	32'd2	32'd2													
→ /TestBench/end_co	32'dx														
→ /TestBench/instruction	-32'd2013265890	-32'd2013	265890												
→ /TestBench/pc	32'd120	32'd120													
→ /TestBench/rs	-5'd6	-5'd6													
± → /TestBench/rt	5'd0	5'd0													
→ /TestBench/rd ———————————————————————————————————	5'd0	5'd0													
→ /TestBench/addr	32'd20	32'd20													
→ /TestBench/data	32'd1	32'd1													
→ /TestBench/i	32'd4	32'd128													
± → /TestBench/j	32'dx														
± → /TestBench/k	4'dx														
± → /TestBench/mode	32'd1	32'd1													
± → /TestBench/score	32'd10														
± ♦ /TestBench/initail_e	32'dx														
TestBench/memory	32'd8 32'd5 32'd3	32'd8 32'd	5 32'd3 32'd	2 32'd1 32'	1 32'd0 3	2'd0 32'd	0 32'd0 3	2'd0 32'd0	32'd0 32	'd0 32'd0	32'd0 32	d0 32'd88	32'd2 32	'd1 32'd8	8 32'

Test data 2_2

\$1 ▼	Msgs															
→ /TestBench/CLK	-1'd1															
→ /TestBench/RST	-1'd1															
+ /TestBench/count	32'd600	32'd	500													
+ /TestBench/handle	32'd2	32'd	2													
→ /TestBench/end_co	32'dx	\rightarrow														
+ /TestBench/instruction	-32'd1004470271	-32'c	1004470	271												
→ /TestBench/pc	32'd80	32'd	30													
→ /TestBench/rs	5'd1	5'd1														
	5'd0	5'd0														
→ /TestBench/rd	5'd4	5'd4														
→ /TestBench/addr	32'd8	32'd	3													
→ /TestBench/data	32'd2	32'd	2													
	32'd5	32'd	128													
→ /TestBench/j	32'dx															
	4'dx															
+ /TestBench/mode	32'd2	32'd	2													
+ /TestBench/score	32'd5															
→ /TestBench/initail_e	32'dx															
+ /TestBench/memory	32'd2 32'd2 32'd2	32'd	2 32'd2 3	2'd2 32'd	32'd0 32	'd0 32'd0	32'd0 32'd	10 32'd0 :	2'd0 32'd	0 32'd0 3	'd0 32'd0	32'd0 32	d0 32'd0	32'd0 32'd	0 32'd0 3	2'd0 3.

Problems you met and solutions:

經過上次的 Lab,這次上手就比較快了,但還是在一些小地方容易出錯。 像是我的 decoder 為了要做出 jr 的功能所以把輸入訊號多加了一個 instr_func_i,結果 always@()裡面忘了加進去判斷,導致每次做到 add 的時候 都會直接 jump,找個很久才發現錯誤。

Summary:

經過這次的 lab 我對指令的架構更加了解了,也更熟悉 branch 指令的執行過程。