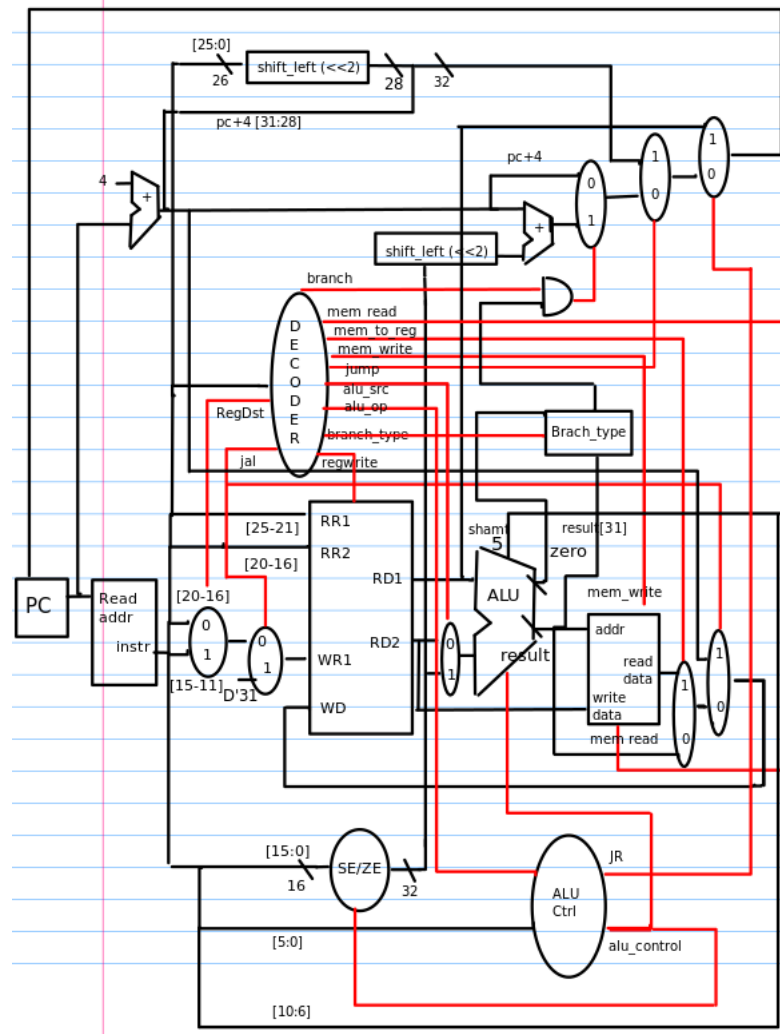


Computer Architecture LAB3

0711282 邱碩霖


Architecture Diagram


此次與LAB2皆使用Linux下的Xournal++繪製Architecture Diagram，
背景線條無法隱藏請助教見諒QQ



Implementation Details

首先，先將Decoder的部份由LAB2做延伸

Decoder 	R-type	addi	sltiu	beq	ori	bne
Reg_Write	1	1	1	0	1	0
ALU_op	000	001	010	110	111	110
ALUSrc_o	0	1	1	1	1	0
RegDst_o	1	0	0	0	0	0
Branch_o	0	0	0	0	0	1
mem_wrtie	0	0	0	0	0	0
mem_read	0	0	0	0	0	0
mem_to_reg	0	0	0	0	0	0
jump_o	0	0	0	0	0	0
branch_type(2bit)	0	0	0	0	0	1
jal_o	0	0	0	0	0	0

Decoder 	lw	sw	blez	bgtz	jump	jal
Reg_Write	1	0	0	0	0	1
ALU_op	101	101	110	110	000	000
ALUSrc_o	1	1	0	0	0	0
RegDst_o	0	0	0	0	0	0
Branch_o	0	0	1	1	0	1
mem_wrtie	0	1	0	0	0	0
mem_read	1	0	0	0	0	0
mem_to_reg	1	0	0	0	0	0
jump_o	0	0	0	0	1	1
branch_type(2bit)	0	0	2	3	0	0
jal_o	0	0	0	0	0	1

ALU對應控制訊號要做的功能：

ALU control(4bit)	功能
0000	AND
0001	OR
1000	ADD(signed)
0010	ADD
1010	SUB(signed)
0110	SUB
0111	SLT
0101	SLTiu
1100	NOR
1110	SRA
1111	SRAV
1011	Lui
0011	beq
1101	SLL
0100	MUL
1001	bne

R type功能

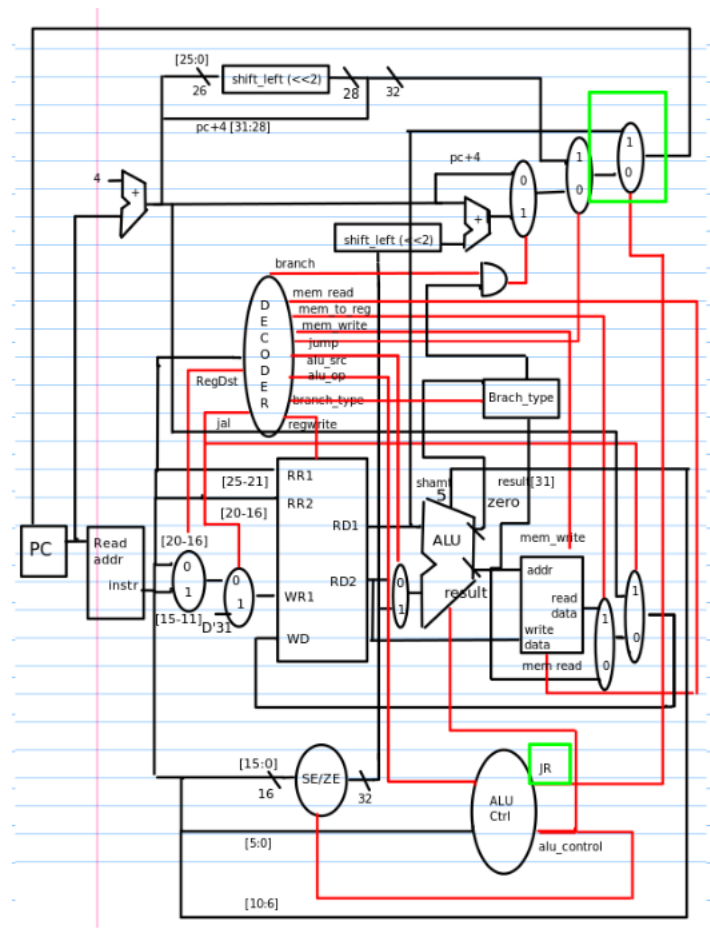
SLL MUL

在ALU_Control.v ALU.v內擴充SLL和MUL功能即可，由於PDF內說明MUL結果並不會超過32bit，因此我們並不用特別處理它

Jr

由Jr為R-type，Decoder無法直接由opcode知道現在要執行jr這項功能，因此在LAB3裡我選擇在ALU_control新增了Jr_o來控制MUX，此MUX的目的為選擇上一個MUX的結果(jump/not jump address)或是RS_data 並傳回給PC

下圖為此MUX之位置(綠色區為控制信號與MUX位置)



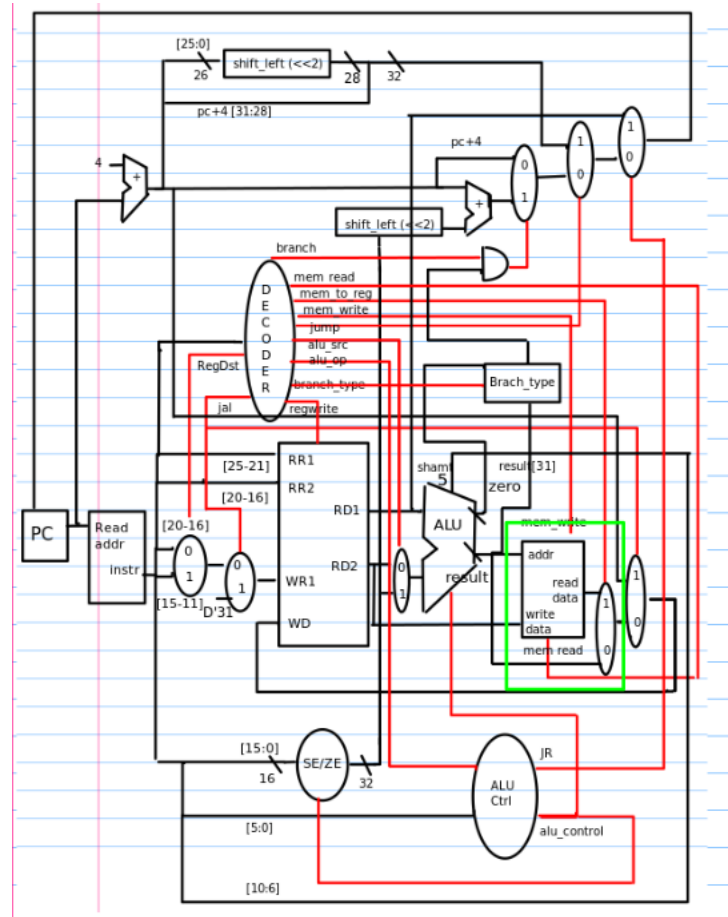
I type

lw sw

由ALU_result再延伸至Data_Memory，若為lw我們要將記憶體資料讀出來並寫入rt，將mem_read設為1；若為sw我們則要寫入記憶體中，則將mem_write設為1。

並在外部再多設立一個MUX決定要傳回Reg_file的write_data為ALU_result還是mem_read_data(為了lw)

下圖綠色光處為為了lw sw所新增的架構



bgtz blez

由於此次新增了bgtz blez兩項功能，按照LAB2的寫法(在ALU中判斷bne與beq的結果)將會變得過於複雜，因此此次將捨棄LAB2在ALU中所使用的bne與beq，

若讀到bne, beq, blez, bgtr任一指令，則在ALU中做signed sub，並將**result的最高位元(MSB)** (為了來判斷相減的結果為正或負數)以及**Zero(ALU_result是否為0)** 以及**branch_type** 傳入新的元件中Branch_type.v中

經由Branch_type.v內判斷後傳出branch_type_result_o，並與branch_o作AND之後結果傳給MUX控制PC是否為branch address，

至於為何還需要branch_o是因為branch_type僅2bit，因此其他非Branch的I-type也會傳到Branch_type.v裡頭並有可能結果為要Branch，因此要與Branch_o作AND防止此情況發生。

Branch_type與對應的type:

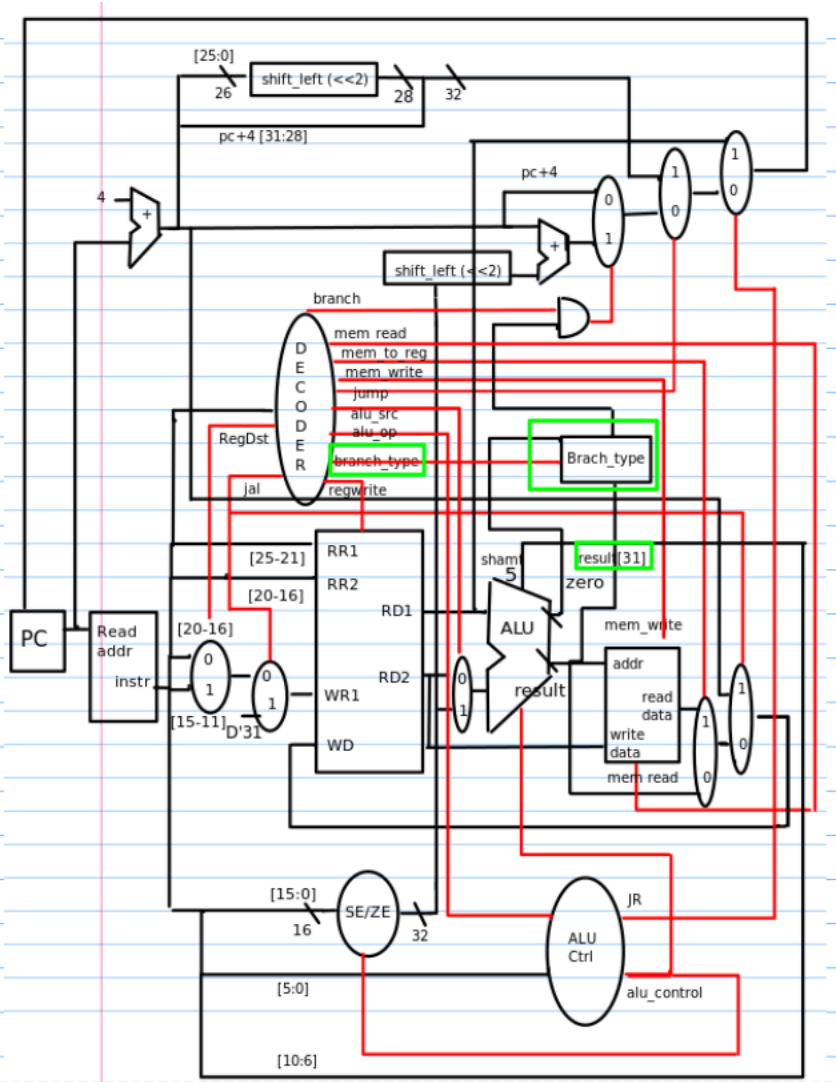
type	beq與其他非branch的I type	bne	blez	bgtz
value	0	1	2	3

Branch_type.v的內部判斷：

	beq	bne
branch_type	0	1
zero_i	1	0
alu_result_i	x	x
branch_type_result	1	1

	blez	bgtz
branch_type	2	3
zero_i	1	0
alu_result_i	1(相減之結果小於0小於成立)	0
branch_type_result_o	zero_i==1或 alu_result_i ==1	alu_result_i==0且zero_i!=1

下圖綠色區為為了bne，beq，blez，bgtz所更動的架構

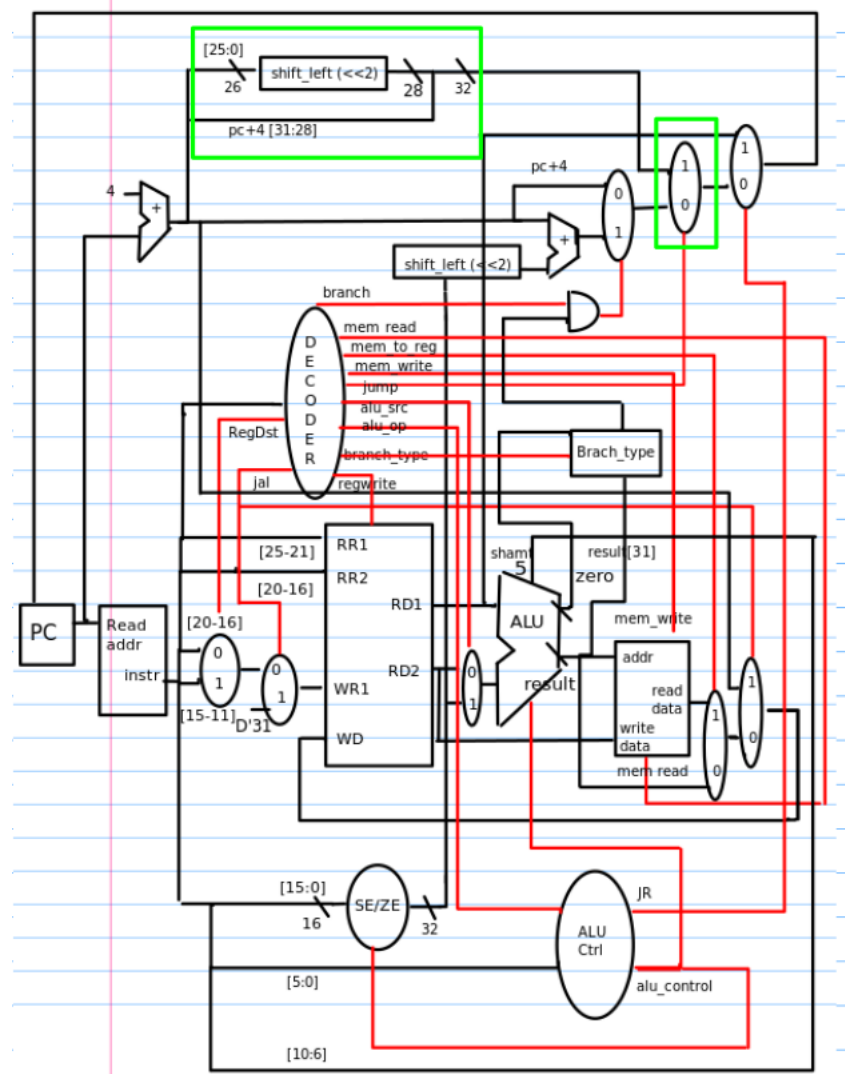


J type

Jump

首先要獲得jump address : concat{pc+4[31:28],instruction[25:0],2'b0}即可，
實際上先將instruction[25:0]接到shift_left_2就可以得到後28bit最後再與pc+4 concat便完成
Jump_address，
最後再接一個MUX至Branch的MUX後判斷要得到Branch_address或是Jump_address

下圖綠色部份為此次做更動的架構部份



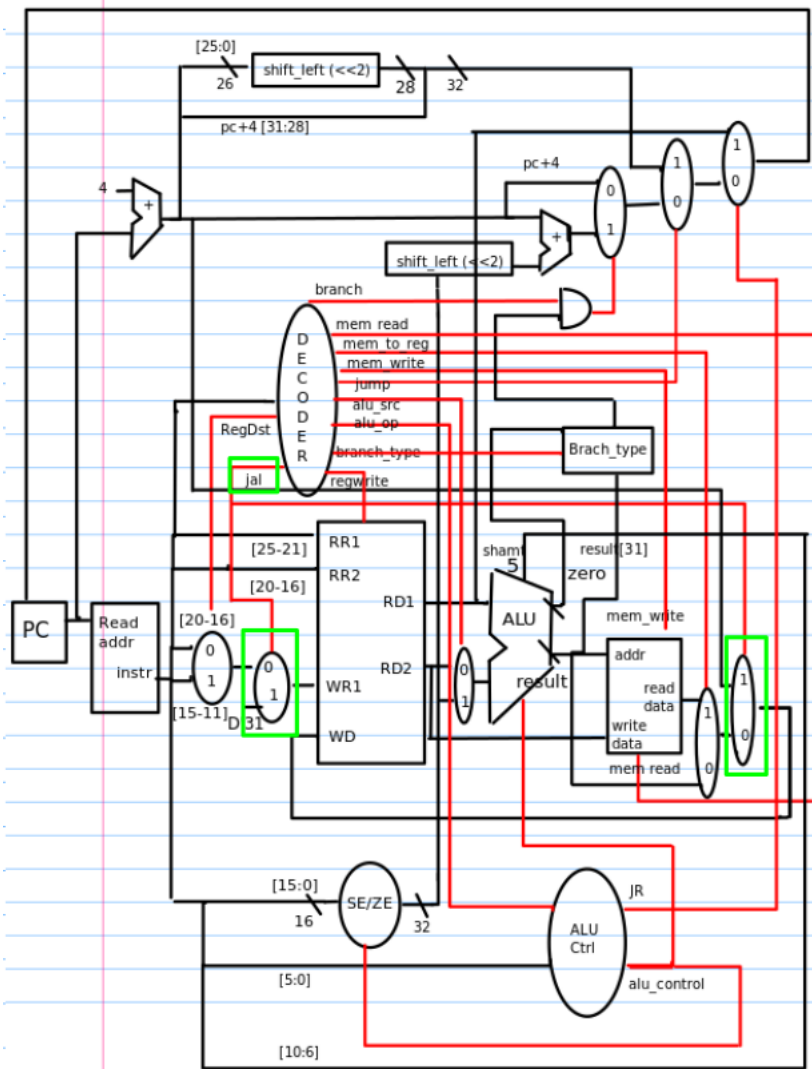
Jump and link

jal實現方式與jump大同小異，首先要先在decoder內將jump_o設為確保pc會接到jump_address，

在RD_addr前再設置一個MUX，一個接收來自RegDst的結果，一個設置D'31(R31)，由jal控制，

在ALU_result或mem_read_data的MUX之後再設置一個MUX決定要寫入的資料為ALU_result或mem_read_data的結果或是pc+4，控制信號則由jal控制

下圖綠色部份為此次做更動的架構部份



3. Assume there is an instruction “beqi \$rt, imm, offset” (branch if equal to immediate). Although it is not supported by our CPU, it can be achieved by the combination of two instructions “ori \$ra, \$zero, imm” and “beq \$rt, \$ra, offset”. Can the instruction “bge \$rs, \$rt, offset” (branch if greater than or equal, branch if \$rs >= \$rt) be replaced with instructions implemented in lab2/3? (Hint: slt)

ANSWER:

Yes, 可以利用 `slt $t0,rs,rt` 與 `beq $t0, $ZERO,offset` 組合
因為若 `t0=1` 表示 `rs<rt`，若 `t0=0` 則表示 `rs>=rt`，符合 `bge`

4. Following the previous question, can we reduce some of the instructions in lab2/3 without reducing the capability of CPU? If possible, what is the minimum instruction set, and what are the advantages and disadvantages of this reduction?

ANSWER:

首先，先列出所有目前的指令功能：

R-type:

`addu` , `subu` , `and` , `or` , `sra` , `srav` , `slt` , `sll` , `mul` , `jr`

I-type:

`addi` , `sltiu` , `beq` , `lui` , `ori` , `bne` , `lw` , `sw` , `blez` , `bgtz`

J-type:

`j` , `jal`

1. `srav` 可替代 `sra`:

利用 `addi $t0, $ZERO,imm` 將 `imm` 的值存入 `t0` 之中
再利用 `srav $rd, $rt, $t0(rd=rt>>t0)`

2. `sltiu`:

原指令: `sltiu r1,r2,imm` (`r2<imm` 的話 `r1=1`，其餘為0)
利用 `addi $t0, $ZERO,imm`
(經 Sign Extension 使其滿足32bit)
`slt $r1, $r2, $t0`

3. `lui`:

原指令: `lui r1,imm` (`r1=imm<<16`)
利用 `addi $t0, $ZERO,imm`
再往左 shift , `sll $r1, $t0,16`

4. `ori`:

原指令: `ori $r1, $r2,imm` (`r1=r2|imm` (Zero Extension))
利用 `addi $t0, $ZERO,imm`
(經 Zero_Extension 使其滿足32bit)
再 `or $r1, $r2, $t0`

5. `blez`:(`<=0`)

原指令: `blez $rs,imm`
利用: `slt $t0, $ZERO, $rs` (`t0=1` if `$zero<rs`; `t0=0` `$zero>=rs`)
再 `beq $t0, $ZERO,imm`

6. `bgtz` (`>0`)

原指令: `bgtz rs,imm`
利用: `slt $t0, $ZERO, $rs` (`t0=1` if `0<rs`; `t0=0` `0>=rs`)
再 `bne $t0, $ZERO, imm`

最終的指令:

R-type:

addu , subu , and , or , srav , slt , sll , mul , jr

I-type:

addi , beq , bne , lw , sw

J-type:

j , jal

優缺點

優點	缺點
簡化指令集	因為使用較多指令集去組合成一個功能 耗費較多執行時間與系統資源

Contribution

架構： 0711282邱頌霖

CODE: 0711282邱頌霖

確認結果: 0716077 柯柏揚

DEBUG: 0711282邱頌霖 0716077 柯柏揚

生測資: 0716077 柯柏揚

Discussions, problem encountered and miscellaneous

1. 這次遇到最大的Bug 就是在Single_Cycle_Cpu.v中Data_Memory的module name沒有跟testbench.v中呼叫的相同...看了錯誤訊息一直以為是自己其他部份線沒有接好，反覆看了好幾遍覺得都是對的，東改西改都還是有錯誤訊息，花了好多時間DEBUG最後才發現是module name的問題 QQ
2. LAB2的時候不知道LAB3要接續著，因此架構圖沒有留白，所以又重畫了，不過這次畫出來的比LAB2好看多了 😊