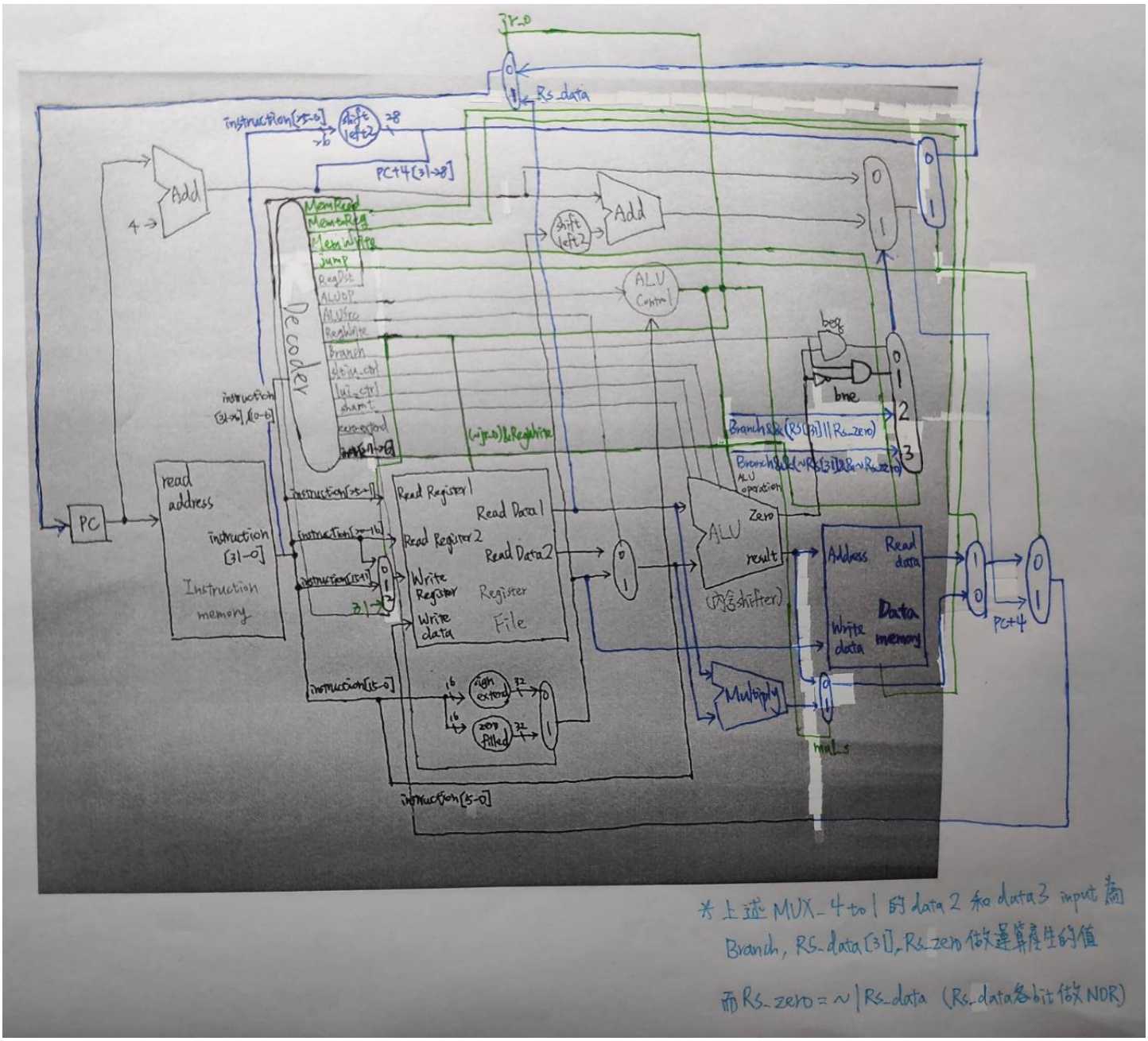
Term Project – Lab 3 Report

0716221 余忠旻

Architecture diagram:



Implementation details:

Decoder:增加 MemRead，MemToReg，MemWrite，Jump，Jal 的控制元，MemRead 決定要不要讀取 data_memory，MemToReg 決定要不要把 data_memory 的資料移入 register，Jump 決定要不要做 jump，Jal 決定要不要把 PC+4 寫入$ra (r31)，

Multiply:實作乘法指令

Data_memory:讀取 memory 資料或者把資料寫入 memory，並且處理 load 和 store 指令

MUX_4to1 Mux_Branch:決定 Branch 的哪種指令(beq，bne，blez，bgtz)

ALU_Ctrl:增加 Shift_o，Jr_o，mul_s，Shift_o 決定 shift 的方向(右移還左移)，Jr_o 決定 Jr 指令，當有做 Jr 指令要將 Rs_data 寫入 PC，並且將 RegWrite 設為 0，防止數字被寫入而被覆蓋，mul_s 決定是否有做乘法

Jump 和 Jal:將 PC+4[31-28]和 instruction[25-0]左移 2 位做 concatenation 送回 PC，而 Jal 要多做把 PC+4 的值存回$ra (r31)

#上圖 architecture 有顏色部分是這次 lab3 新增加的部分#

Question:

1. Please list the machine code of each branch/jump (j, bne, beq and bgtz) instructions in "_CO_Lab3_test_data_bubble_sort.txt" on the report.

00000000000000000100000000100001
00100000000010010000000000001010
00100000000010100000000000001101
00000001001010010101100000011000
00001000000000000000000000011001
//bubble:
00100000000010000000000000001010
00100000000010010000000000000100
00000001000010010110000000011000
//outer:
00100000000011100000000000001000
00000001100011100100000000100011
00100000000010010000000000000000
//inner:
10001101000010100000000000000100
10001101000010110000000000000000
00000001011010100000100000100011
00011100001000000000000000000011
10101101000010100000000000000000
10101101000010110000000000000100
00100000000010010000000000000001
//no_swap:
00100000000011010000000000000100
00000001000011010100000000100011
00000001000000000000100000101010
00011100001000000000000000000001
00001000000000000000000000001011
//next_turn:
00010101001000001111111111110000
00001000000000000000000000011101
//Jump:
00000001010010010101000000100011
//Loop:
00000001011010100110000000100001
00010001001010101111111111111110
00001000000000000000000000000101
//End:
00000000000000000000000000000000

2. What is the main purpose of "jal" and "jr" instructions? Please write down a simple program which explains the scenario.

Jal:將 PC+4 存入$ra ,並且跳到指定的指令
r31($ra) = PC + 4
PC = {(PC + 4)[31:28], addr,2b'00}
Jr:將 rs 的 data 存入 PC 中
Pc = rs

```
int fact (int n)
{ if (n < 1) return 1;
else return n * fact(n -1);}

addi $sp, $sp, -8
sw $ra, 4($sp)
sw $a0, 0($sp)
$t0, $a0, 1
beq $t0, $zero, L1
addi $v0, $zero, 1
addi $sp, $sp, 8
jr $ra
L1: addi $a0, $a0, -1
jal fact
lw $a0, 0($sp)
lw $ra, 4($sp)
addi $sp, $sp, 8
mul $v0, $a0, $v0
jr $ra
```

上述 code 的 jal 會將下一個位址(PC+4)存到$ra，並且跳到 fact 的 Label
而 jr 則是將之前$ra 存的位址讀取出來並丟給 PC，也就是說 jr 會跳到原本 jal 的下個指令執行，就是 lw $a0, 0($sp)

3. 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)
slt $t0,$rs,$rt (If rs>=rt,then t0=0)
beq $t0,$zero,offset

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?

可以，雖然 reduce 一些 instruction 的時候會使 insturction count 的數量增加，但每一個的 cycle time 會減少，因此總體執行時間是較少的，而複雜的指令集每一個 cycle time 會較多，而簡單的指令會因為相同的 cycle time 而遭到有浪費時間的情況，也就是有 idle 的情形，pipeline design 也會增加難度，不好設計，所以 reduce some instruction 能提高整體效能 without reducing the capacity of CPU。

Minimum instruction set: addu，and，or，subu，slt，srav，sra，sll，addi，beq，bne，lw，sw，lui，ori，sltiu，j，jal，jr

blez 和 bgtz 可以用 slt，beq，bne 等指令完成

mul 可以用 addu，sll 等指令完成

Advantage:架構較為簡單，執行效率會比較好，在 pipeline design 較容易。

Disadvantage:有些複雜指令需要分為多個指令來執行，需要更加頻繁的讀取 register，且在 coding 上會較為困難。

Contribution:

各個module實作、接線， d-bug，共同寫bubble sort的machine code，共同討論答案

Discussions:

problem I encountered: Verilog 語法不熟悉，花費不少時間研究語法，有些的指令需要額外處理，bubble sort 的 machine code 要查詢各方資料才能寫出來