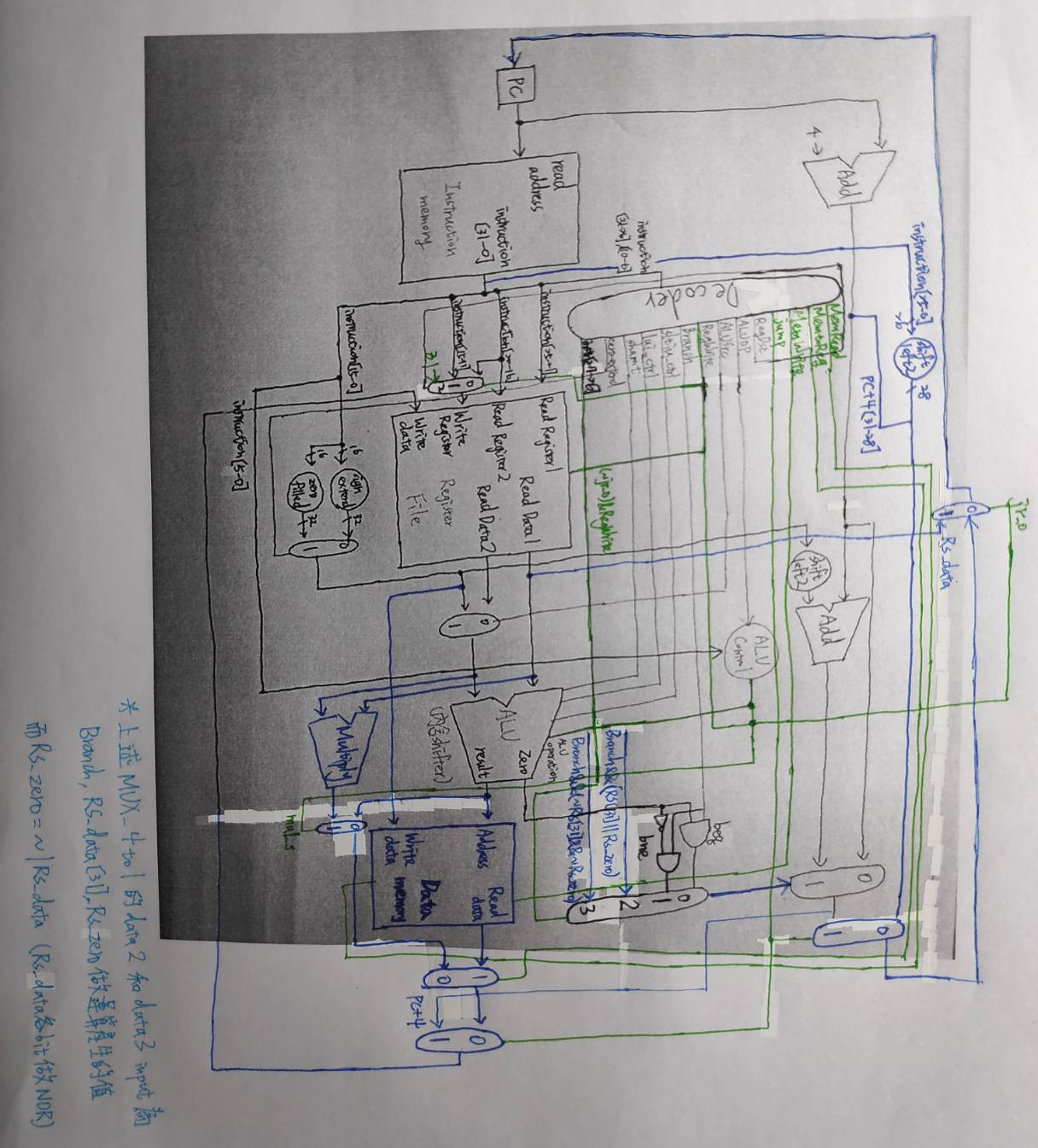
**Term Project - Lab 3 (Single cycle CPU with memory and jump) Report**

**Architecture diagram:**



\*有色部分為lab3增加之實作

**Implementation details:**

Decoder: 增加MemRead, MemToReg, MemWrite, Jump, Jal等上次未實作之部分。

Multiply: 實作兩數相乘指令。

Data\_memory: 讀取memory資料或者把資料寫入memory，並且處理load和store指令。此.v檔不須進行任何修改。

MUX\_4to1 Mux\_Branch: 決定要執行哪種Branch (beq, bne, blez, bgtz四選一)。

ALU\_Ctrl: 增加Shift\_o, Jr\_o, mul\_s。Shift\_o決定shift的方向(右移、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位串接後送回PC，而Jal要多做把PC+4的值存回$ra (r31)。

**Question:** (黑字為題目，紅字為答案)

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)。

Simple program:

int fact (int n){

if (n < 1) return 1;

else return n \* fact(n -1);

}

MIPS:

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

Explanation:

jal 會將(PC+4)存到$ra，並跳到fact的Label。而jr則是將之前$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，也就是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)

Yes.

slt $t0,$rs,$rt

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，雖然減少一些instruction會使instruction count增加，但每一個的cycle time 會減少，因此總體執行時間可能會較少。而複雜的指令雖然一個指令能做到較多事，但每一個cycle time也會較多，也不容易進行pipeline。所以這題的答案是yes。

Minimum instruction set in lab2/3: 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等指令組合實行。mul則可以用addu, sll兩指令組合完成。

Advantage: 架構簡單，執行效率較好，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pc 和register，且在coding上有可能會較為不便。

**Contribution:**

各個module實作、接線，debug，共同討論答案。

**Discussions:**

架構上越接越複雜，許多額外處理的mux和線路未經過整合，將來在pipeline上會是一大挑戰。bubble sort的machine code自己寫容易出現小錯誤，debug de了很久。