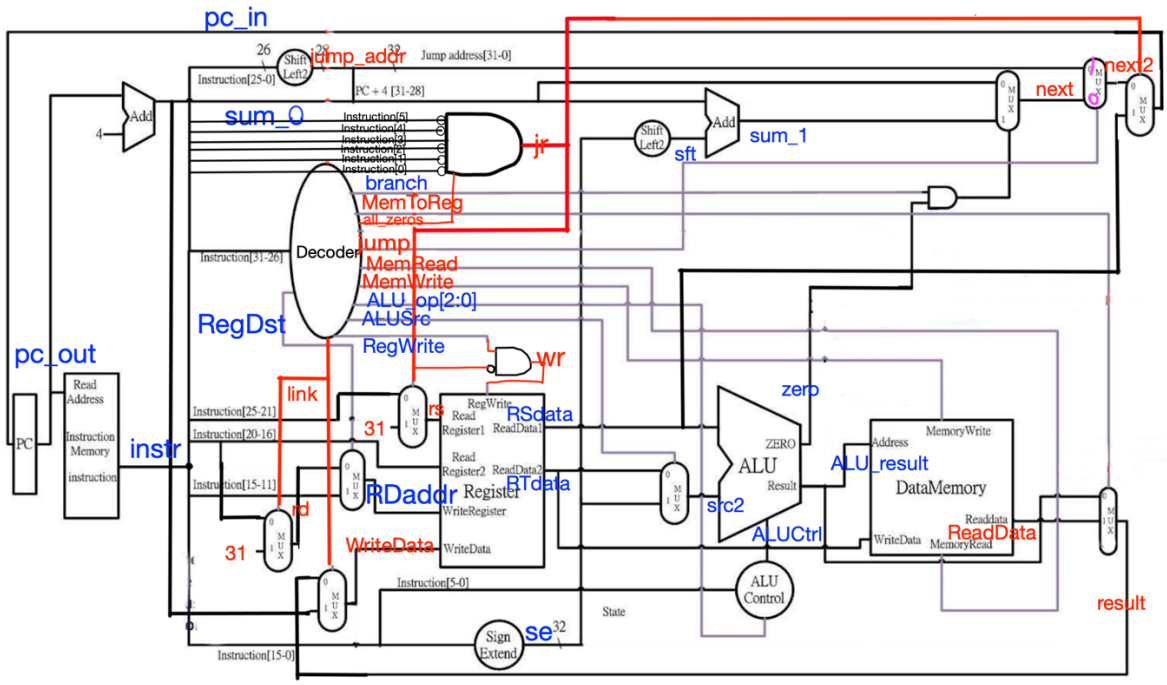
Computer Organization Lab3

Name: 陳芷萱

ID: 110550029

//以 iverilog 編譯

- Architecture diagrams



jr：是否為 jr 指令

wr：是否寫入暫存器，jr 指令 Decoder 會輸出 RegWrite=1 但此時不將資料寫入暫存器

link：是否將下一個指令位置寫入暫存器

jump_addr：jump 目的地位置，為 PC 前 4 bit + jump 右移 2 bit 共 32 bit 組成

all_zeros：是否全部 Opcode 皆為 0

- Hardware module analysis

  此次電路設計以 Lab2 為基礎，僅更動 Decoder.v 以及 Simple_Single_CPU.v 兩子模組。

  - Decoder.v

    Lab3 之 Decoder 新增 Link_o、jump_o、MemToReg_o、MemRead_o、MemWrite_o、all_zeros_o（標記 Opcode 是否為 0）共六個控制訊號，並

新增 100011、101011、000010、000011 共四種 Opcode 之判讀。

- R-format、beq、addi、slti

  除 R-format 之 all_zeros_o 需設為 1 外，其餘舊有之控制訊號輸出不變；由於此類指令皆不牽涉記憶體操作與 jump，新增之控制訊號輸出 0。

- lw – 100011

  需進行加法運算以得出讀取記憶體位置，因此 ALU_op_o 輸出表加法的 010；加數來源為指令後 16 bit 代表之數字，ALUSrc_o 輸出 1；第二個暫存器為寫入位置，RegDst_o 輸出 0；需將讀取記憶體並將讀取結果寫入暫存器，MemRead_o、MemToReg_o 與 RegWrite_o 輸出 1；不需進行 branch、link、jump 與記憶體寫入之操作，Branch_o、Link_o、jump_o 與 MemWrite_o 輸出 0。

- sw – 101011

  需進行加法運算以得出寫入記憶體位置，因此 ALU_op_o 輸出表加法的 010；加數來源為指令後 16 bit 代表之數字，ALUSrc_o 輸出 1；不需將資料寫入暫存器，RegWrite_o 輸出 0，RegDst_o 可輸出任意值；需將資料寫入記憶體，MemWrite_o 輸出 1；不需進行 branch、link、jump 與記憶體讀取之操作，Branch_o、Link_o、jump_o、MemRead_o 與 MemToReg_o 輸出 0。

- jump – 000011

  不需進行 ALU 運算、讀寫等操作，因此除 jump_o 外其餘輸出 0。

- jal – 000011

  除 jump 外，需進行 link 將下一個指令位置寫入 r31 中，因此 jump_o、Link_o、RegWrite_o 輸出 1，其餘輸出 0。

```
always @(instr_op_i) begin
        case(instr_op_i)
                6'b000000: begin //R-format
                        ALU_op_o<=R; ALUSrc_o<=0; RegDst_o<=1; RegWrite_o<=1; Branch_o<=0;
                        Link_o<=0; jump_o<=0; MemToReg_o<=0; MemRead_o<=0; MemWrite_o<=0; all_zeros_o<=1;
                end
                6'b000100: begin //beq
                        ALU_op_o<=3'b110; ALUSrc_o<=0; RegDst_o<=0; RegWrite_o<=0; Branch_o<=1;
                        Link_o<=0; jump_o<=0; MemToReg_o<=0; MemRead_o<=0; MemWrite_o<=0; all_zeros_o<=0;
                end
                6'b001000: begin //addi
                        ALU_op_o<=3'b010; ALUSrc_o<=1; RegDst_o<=0; RegWrite_o<=1; Branch_o<=0;
                        Link_o<=0; jump_o<=0; MemToReg_o<=0; MemRead_o<=0; MemWrite_o<=0; all_zeros_o<=0;
                end
                6'b001010: begin //slti
                        ALU_op_o<=3'b111; ALUSrc_o<=1; RegDst_o<=0; RegWrite_o<=1; Branch_o<=0;
                        Link_o<=0; jump_o<=0; MemToReg_o<=0; MemRead_o<=0; MemWrite_o<=0; all_zeros_o<=0;
                end
                6'b100011: begin //lw
                        ALU_op_o<=3'b010; ALUSrc_o<=1; RegDst_o<=0; RegWrite_o<=1; Branch_o<=0;
                        Link_o<=0; jump_o<=0; MemToReg_o<=1; MemRead_o<=1; MemWrite_o<=0; all_zeros_o<=0;
                end
                6'b101011: begin //sw
                        ALU_op_o<=3'b010; ALUSrc_o<=1; RegDst_o<=0; RegWrite_o<=0; Branch_o<=0;
                        Link_o<=0; jump_o<=0; MemToReg_o<=0; MemRead_o<=0; MemWrite_o<=1; all_zeros_o<=0;
                end
                6'b000010: begin //jump
                        ALU_op_o<=3'b000; ALUSrc_o<=0; RegDst_o<=0; RegWrite_o<=0; Branch_o<=0;
                        Link_o<=0; jump_o<=1; MemToReg_o<=0; MemRead_o<=0; MemWrite_o<=0; all_zeros_o<=0;
                end
                6'b000011: begin //jal
                        ALU_op_o<=3'b000; ALUSrc_o<=0; RegDst_o<=0; RegWrite_o<=1; Branch_o<=0;
                        Link_o<=1; jump_o<=1; MemToReg_o<=0; MemRead_o<=0; MemWrite_o<=0; all_zeros_o<=0;
                end
                default: begin
                        ALU_op_o<=R; ALUSrc_o<=0; RegDst_o<=1; RegWrite_o<=1; Branch_o<=0;
                        Link_o<=0; jump_o<=0; MemToReg_o<=0; MemRead_o<=0; MemWrite_o<=0; all_zeros_o<=1;
                end
        endcase
end
```

- ■ Simple_Single_CPU.v

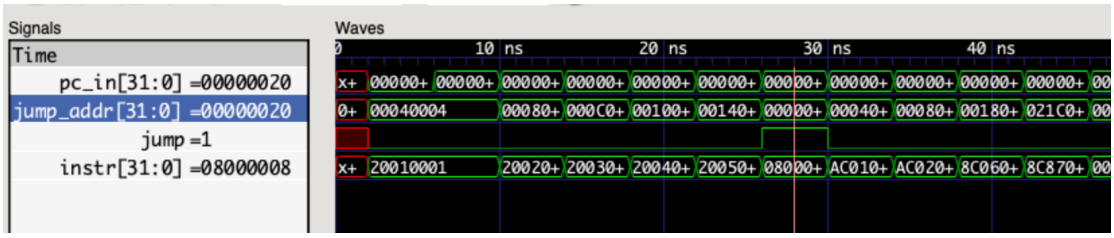
  將各元件依照 Architecture Diagram 連接，而輸出 jr 與 wr 之 AND 邏輯閘以 assign 方式撰寫。

```
assign jr= ~instr[5] & ~instr[4] & instr[3] & ~instr[2] & ~instr[1] & ~instr[0] & all_zeros;
assign wr=RegWrite & ~jr;
```

- ● Finished part

  - ■ Testcase1

    輸出之 result 與解答相符。根據波形圖可發現 jump 訊號正確出現，且 jump 訊號為 1 時 PC 讀入之指令與 jump address 相符。



```
r0=          0
r1=          1
r2=          2
r3=          3
r4=          4
r5=          5
r6=          1
r7=          2
r8=          4
r9=          2
r10=         0
r11=         0
r12=         0
r13=         0
r14=         0
r15=         0
r16=         0
r17=         0
r18=         0
r19=         0
r20=         0
r21=         0
r22=         0
r23=         0
r24=         0
r25=         0
r26=         0
r27=         0
r28=         0
r29=       128
r30=         0
r31=         0
m0=          1
m1=          2
```

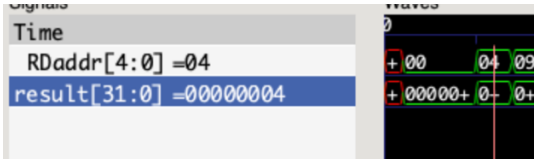
- Testcase2

此程式碼設定$a0 儲存費氏數列項數，$v0 儲存該項之值，即

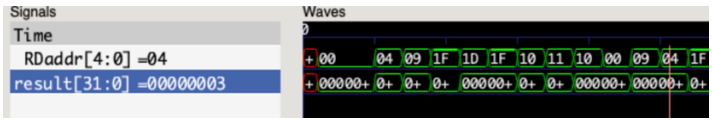
fib($a0)=$v0。又$a0 為$4，$v0 為$2，result 為寫入之數值。

此程式預計算 fib(4)，順序為：

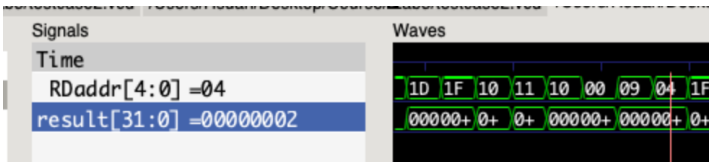
->計算 fib(4)



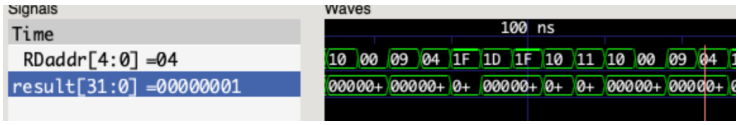
->計算 fib(3)



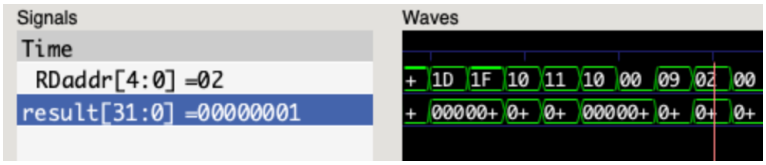
->計算 fib(2)



->計算 fib(1)

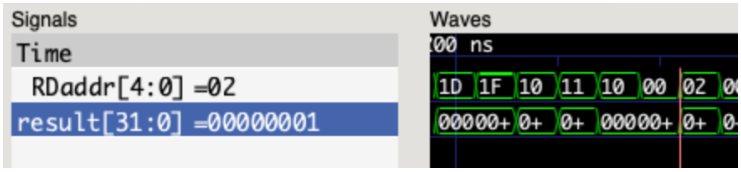


->得出 fib(1)=1



->計算 fib(0)

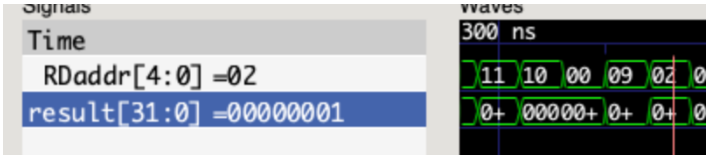
->得出 fib(0)=1



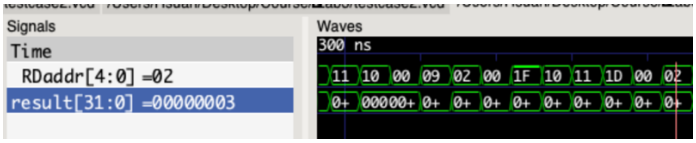
->得出 fib(2)=2

->計算 fib(1)

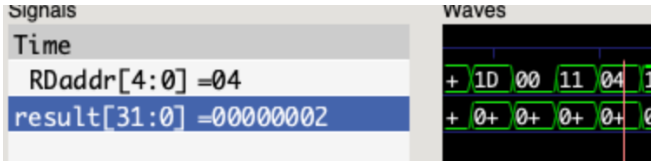
->得出 fib(1)=1



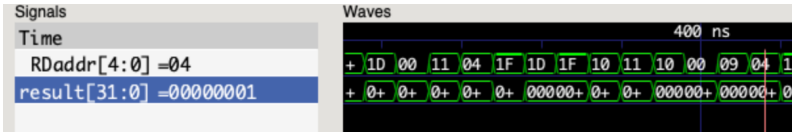
->得出 fib(3)=3



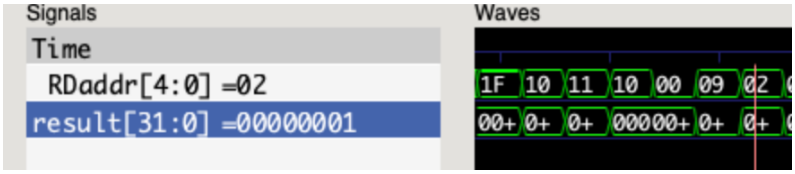
->計算 fib(2)



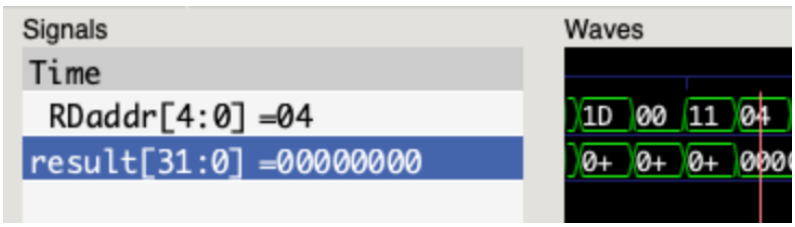
->計算 fib(1)



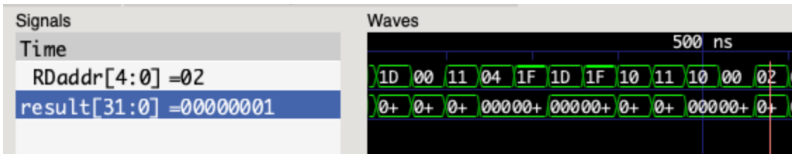
->得出 fib(1)=1



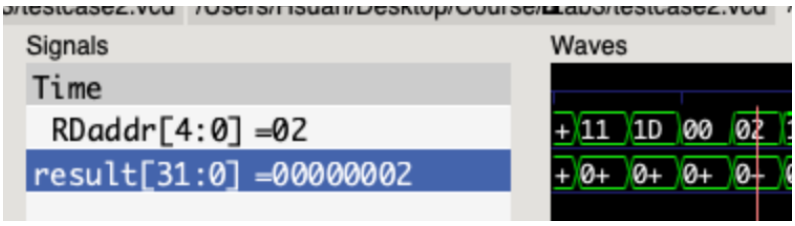
->計算 fib(0)



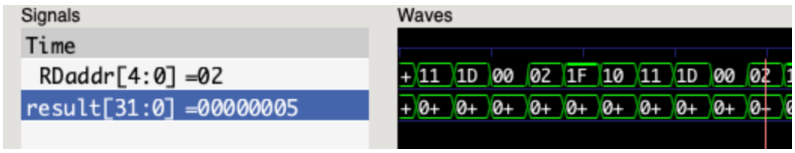
->得出 fib(0)=1



->得出 fib(2)=2



->得出 fub(4)=5



得出之 fib(4)結果正確。

- ## Problems you met and solutions

此次作業中覺得最為困難的是設計 jr 的判讀方式。由於後來才發現 jr 的 Opcode 與 R-format 一樣為全 0，無法直接透過 Decoder 判斷出。最後的解決方式是讓 Decoder 輸出一個 all_zeros 訊號判斷是否 Opcode 為全 0，再與 Funct Field 中的每個 bit 以 AND 閘接起，以判斷該指令是否為 jr。

- ## Summary

此次作業不同於 Lab1 與 Lab2，幾乎只要照著提供的電路圖連接即可，需要根據題目要求修改電路圖，因此在 trace 各訊號流向與意義、刪除不需要的電路、新增 jr 與 jal 功能設計時花了不少時間。不過對於未來需要進一步設計更複雜電路的我們而言，這是一項很好的練習。