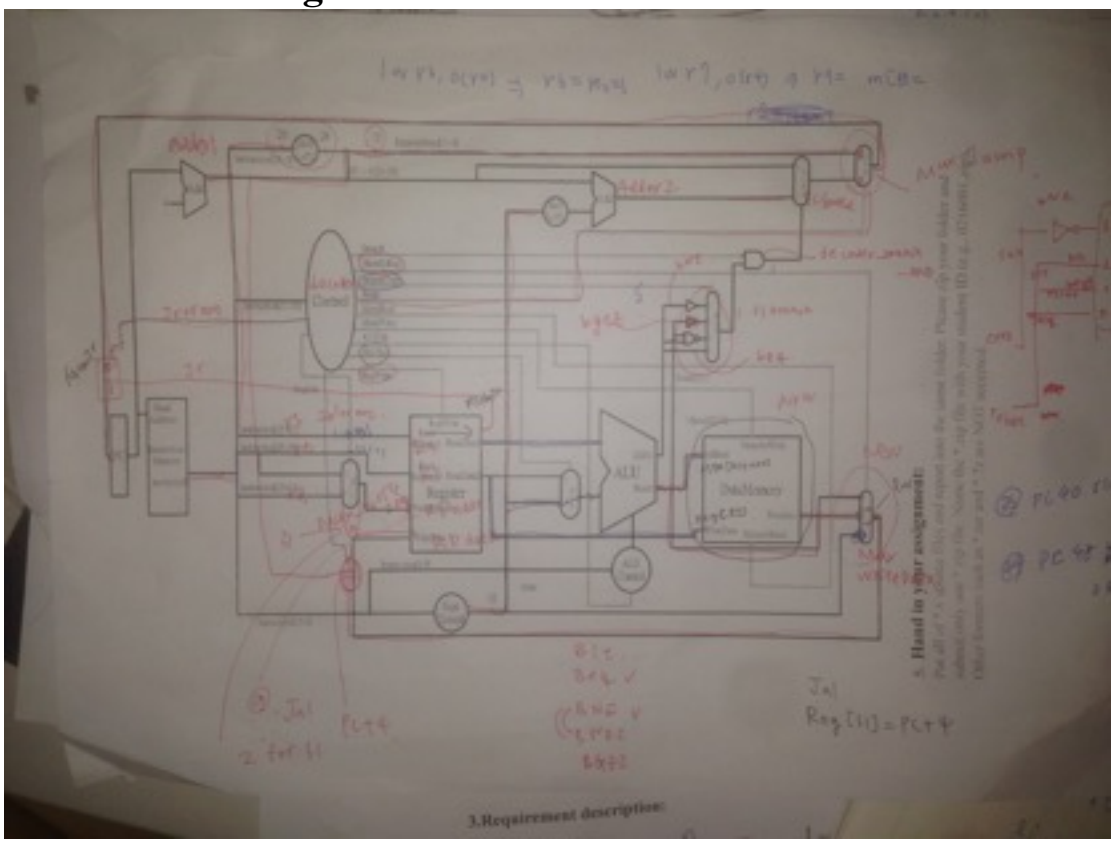
Computer Organization

Architecture diagram:



Detailed description of the implementation:
這次作業好難，好難debug。
幾乎是依照老師給的photo做最右邊datamemory出來的mux我只用了2 to 1 mux 最下面之前從sign extension 出來的wire我沒用。
另外多加了兩個mux 一個給jr 一個給jal 後來發現mux jr可以拿掉，但已經遲交，就趕快做好

test1 test2都有過 因為test3是自己打binary code就沒有測試有沒有過。

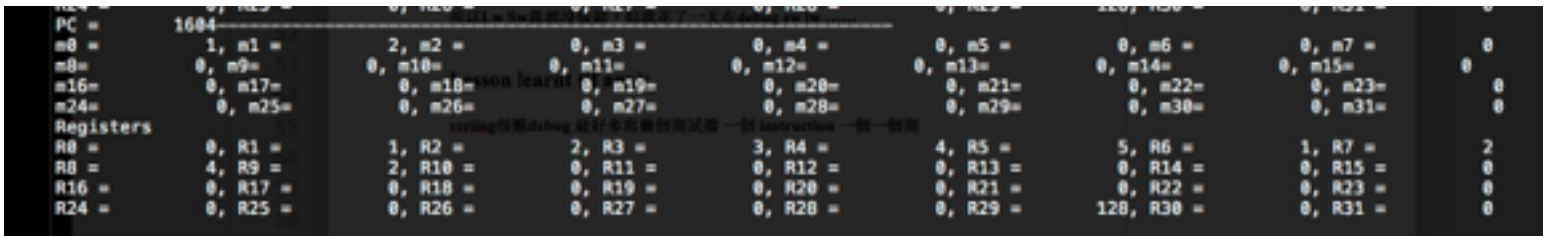
Problems encountered and solutions:
最大的麻煩就是我一直以為我lw sw 寫錯 但test1 都跑得出來，怎麼可能test2會錯。後來我一行一行看

0 add r0,r0,r0 **000000 00000 00000 00000 00000 100000**
4 addi a0,zero,4 **001000 00000 00100 00000 00000000100 r4 = 4**
8 addi t1,zero,1 **001000 00000 01001 00000 00000000001 r9 = 1**
12 jal fib **000011 00000000000000000000000101 r31 = fib:**
10100 20

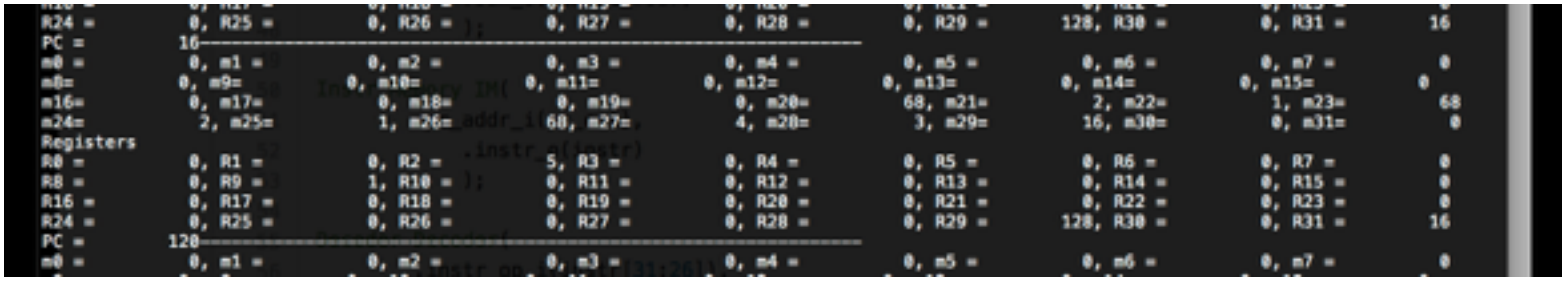
```
16 j       final            000010 00000000000000000000011110 11110<<2 1111000
120
20 fib:    addi sp,sp,-12           001000 11101 11101 1111111111110100 r29=r29 -
12 = 116
24 sw ra,0(sp)                 101011 11101 11111(31) 0000000000000000
m[r29=116/4=29+0] = r31
28 sw s0,4(sp)                     101011 11101 10000(16) 0000000000000100
m[r29+1] = r16
32 sw s1,8(sp)                     101011 11101 10001(17) 0000000000001000
m[r29+2] = r17
36 add s0,a0,zero              000000 00100 00000 10000 00000 100000 r16 = r4 +
r0
40 beq s0,zero,re1             000100 10000 00000 0000000000001100 r16 ?= r0
off=12 44+12*4 = 92
44 beq s0,t1,re1               000100 10000 01001 0000000000001011 r16 ?= r9
off=11 48+11*4 = 92
48 addi a0,s0,-1               001000 10000 00100 1111111111111111 r4 = r16 -1
52 jal fib                         000011 00000000000000000000000101 r31 =
add(56), 101<<2= 10100 =20
56 add s1,zero,v0              000000 00000 00010 10001 00000 100000 r17 = r0+r2
60 addi a0,s0,-2               001000 10000 00100 1111111111111110r4 = r16+ (-2)
64         jal fib                 000011 00000000000000000000000101 r31 =
exitfib(68), 101<<2 = 10100 =20
68         add v0,v0,s1        000000 00010 10001 00010 00000 100000 r2 = r2+
r17
72 exitfib:lw ra,0(sp)         100011 11101 11111 0000000000000000 r31 =
mem[r29]
76         lw s0,4(sp)         100011 11101 10000 0000000000000100 r16 =
mem[r29+1]
80         lw s1,8(sp)         100011 11101 10001 0000000000001000 r17 =
mem[r29+2]
84         addi sp,sp,12       001000 11101 11101 0000000000001100  r29 = r29+12
88         jr ra               000000 11111 00000 00000 000000 01000 jr $31
           //function call結束
92 re1:    addi v0,zero,1          001000 00000 00010 0000000000000001   r2 =
r0+1;
96         j exitfib           000010 00000000000000000000010010      10010<<2
1001000 =72
100 final:      nop
```

才發現是 jr 的opcode 也是000000 沒有辦法直接從decoder那邊去判斷pc要吃mux Jump出來的wire (最右上角的mux)還是吃register File出來的 RS data 最後我終於看懂原來 sw ra,0(sp) 101011 11101 11111(31) 0000000000000000 m[r29=116/4=29+0] = r31 是這樣運作的。
所以Lw Sw我都沒做錯，但我花了一天在debug sw lw ……

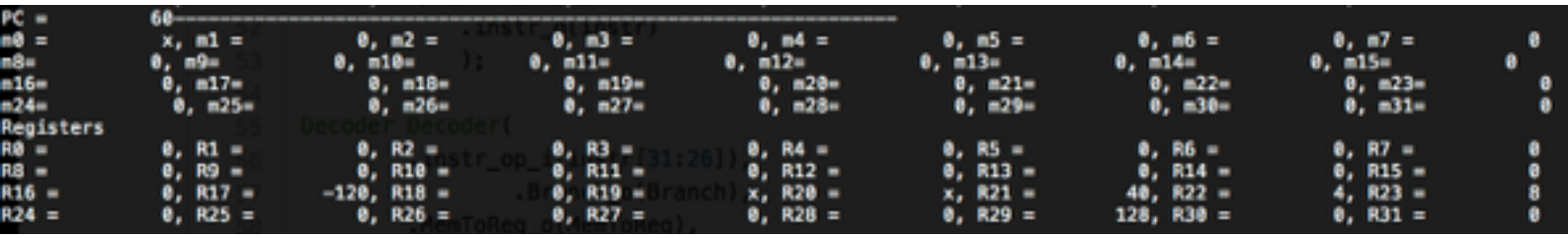
test1



test2



test3



Lesson learnt (if any):

verilog很難debug 最好多寫幾個測試擋 一個 instruction 一個一個測