Final Project: Optimized CPU Design

姓名：張峻瑋

學號：110511194

Server：DCS064

1. The optimized design for CPU

   I. The method in your design

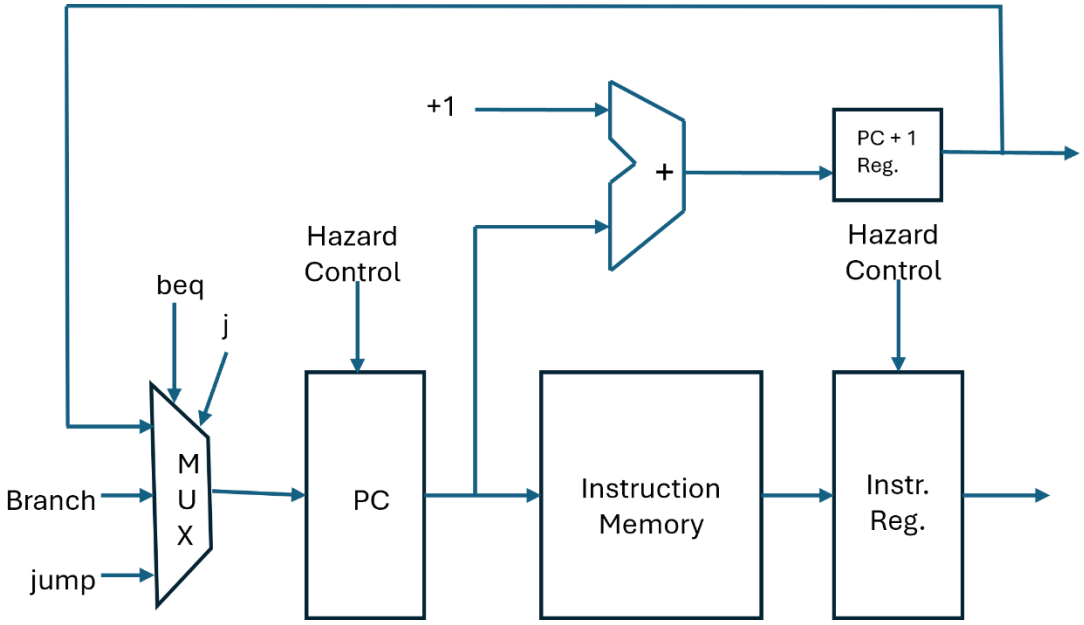

圖 1：IF 階段的設計

在此期末專題中，採用 Pipeline 的方式設計 CPU，可在 latency 不增加太多的情況下，犧牲一點面積，換取極低的 cycle time。在我的設計中，參採計算機組織課本的設計，將 CPU 的流水線分為五個階段：instruction fetch from memory (IF), instruction decode and register read (ID), execute operation or calculate address (EX), access memory operand (MEM), write result back to register (WB).

在 IF 階段中，Program counter (PC) 是一個暫存器，其輸出為指令的 index，送到 instruction memory 去抓取相對應的指令並存到 instruction register 中。又，其輸出會送到加法器裡加 1，並送到暫存器中。倘若指令有 branch 或 jump，則會在多工器中做出適當的選擇並採用（後面會詳述），若無，則 PC 輸入為

PC+1。

若遇到 Hazard 需要塞 bubble，會藉由 hazard control 進行判斷，並送到 PC 跟 instruction register 中。PC 不加 1，instruction register 則維持原樣。

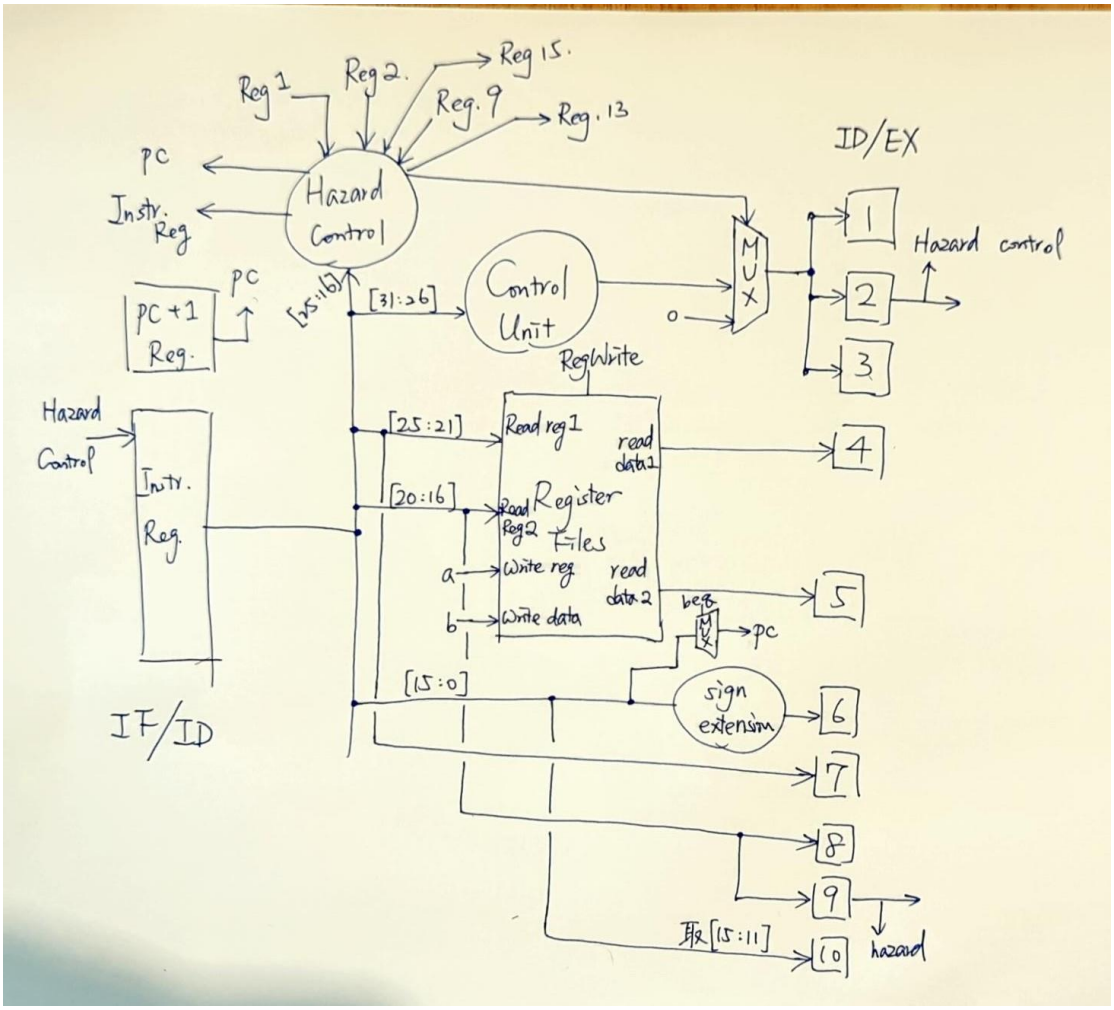

圖 2：ID 階段的設計

在 ID 階段中，從 instruction register 輸出的指令，依序被解碼。其中 31-26 bit 為 opcode，為 control unit 的輸入，依照其指令的類型將 RegDst, branch, MemRead, MemtoReg, ALUOp, MemWrite, ALUSrc, RegWrite, jump 填入適當的值。

Hazard control 會判斷是否要插 bubble，若要，則多工器輸出全為 0，若不需要，則將 MemtoReg、RegWrite 填入 Reg 1，並於 WB 階段使用。MemRead、

MemWrite 填入 Reg 2，並於 MEM 階段使用。ALUSrc、ALUOp、RegDst 填入 Reg 3，並於 EX 階段使用。

Instruction [25:21] 以及[20:16]為暫存器的 index，該 index 所存的值會在 read data 1 以及 read data 2 輸出，並存到 Reg 4 和 Reg 5。另外，由於 R-type 是讀 2 寫 1，I-type 是讀 1 寫 1，branch 是讀 2，故我們將 instruction[25:21]、[20:16]、[15:11]各分別儲存到 Reg 6 到 Reg10，以利下個階段的處理。

Instruction [15:0] 會經過 sign extension 存到 Reg 6。或者是在這之前，已知指令是 beq 且必須要跳，則將此值傳至 pc。

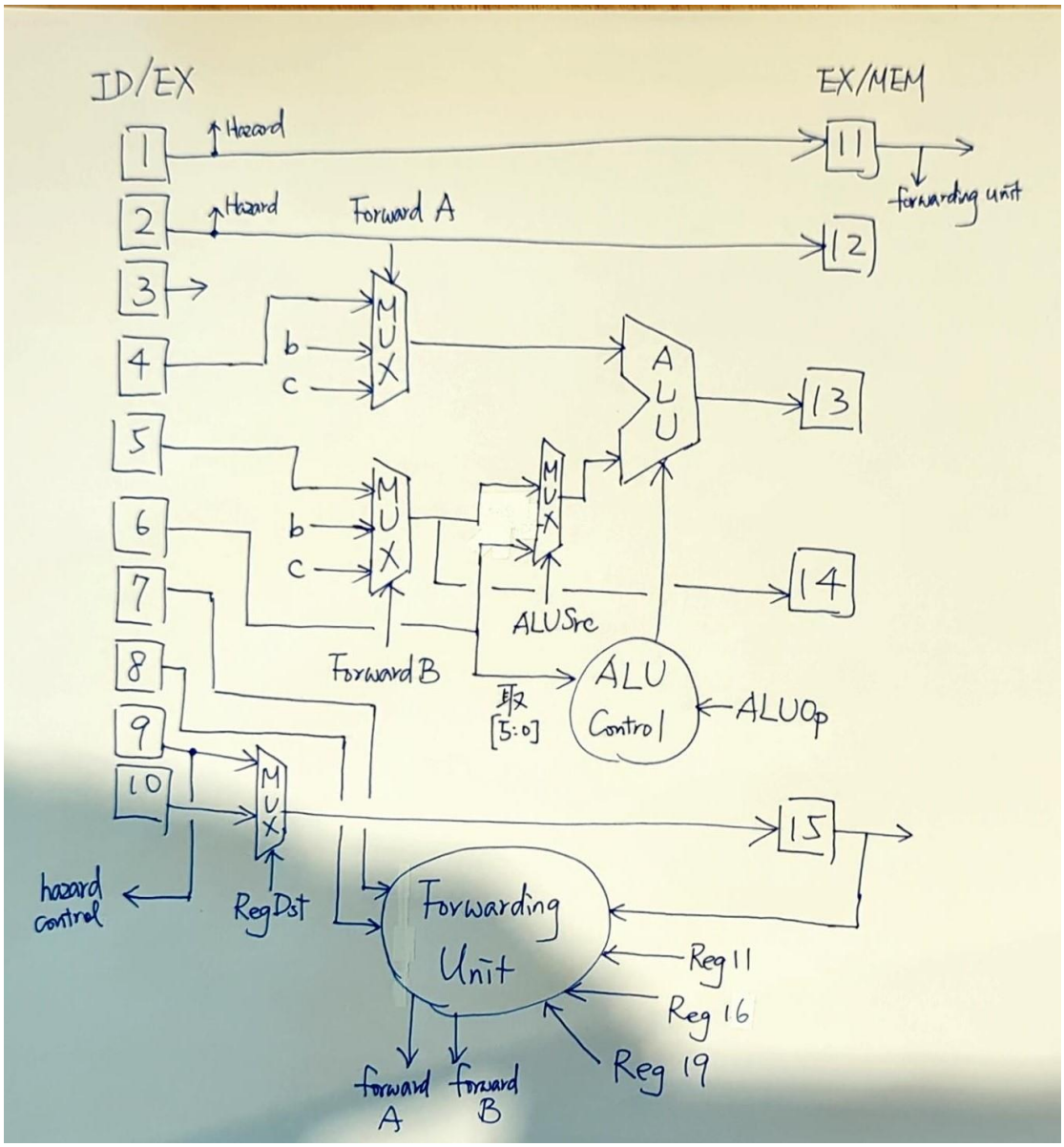

圖 3：EX 階段的設計

Reg 1 裡的 flags 會直接送到 Reg 11，而 Reg 2 的 flags 在送到 Reg 12 的同時「有部分會送到 hazard control。Reg 3 裡的 flags 都是在這個階段要使用的，RegDst 控制 Reg 9, 10 為輸入的多工器，若為 R-type，則寫入暫存器的為 Rd，其他則為 Rt。ALUOp 控制著 ALU control，若為 R-type，則必須要看 instruction [5:0] 來判斷需使用怎麼樣的運算，若非 R-type，則依 opcode 做判斷要做什麼運算，不需要 funct。ALUSrc 則是決定 ALU 的第二個輸入是來自暫存器的內容，也就是 Reg 5，還是從 instruction [15:0] sign extension 而來的值。ALU 運算結果將存入 Reg 13。

另外，為避免 hazard，設有 forwarding unit 來控制。而 forwarding unit 的結果會分到 ALU 兩個輸入之前的兩個多工器當判斷依據。詳細會在下面的小節敘述。而 forward B 所控制的多工器，其輸出結果會送至 Reg 14。RegDst 所控制的多工器，其輸出會送到 Reg 15。

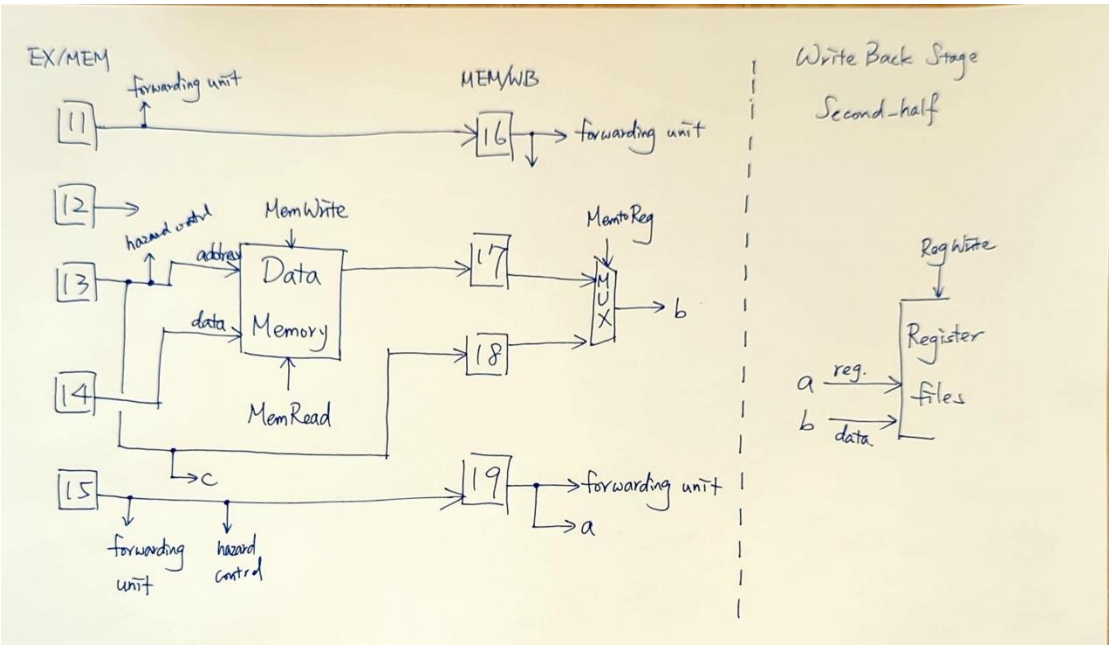

圖 4：MEM 及 WB 兩階段的設計

在 MEM 階段中，Reg 11 的 flags 會直接送到 Reg 16 以利 WB 階段使用，而 Reg

12 的則會於本階段使用。MemWrite 與 MemRead 皆控制著 Data Memory，決定是要從記憶體中讀資料、寫資料還是沒有動作。在 lw 或 sw 中，Reg 13 所存的是遇接觸的位址，若是 lw 則將其對應到的位址的值讀出來，存到 Reg 17，而若為 sw 則是將 Reg 14 的 data 寫入 Reg 13 所對應的記憶體位址中。若為 R-type，Reg 13 記錄的是運算結果，會直接送到 Reg 18，並在 WB 階段中由 MemtoReg 判斷要拿 Reg 17 還是 18 的值送下去。Reg 15 的值會直接傳到 Reg 19。

在 WB 階段中，若要將資料寫入 Register files 中，也就是 RegWrite = 1，Reg 19 存的是欲存入的位址，而經 MemtoReg 判斷的多工器輸出則為欲寫入的 data。

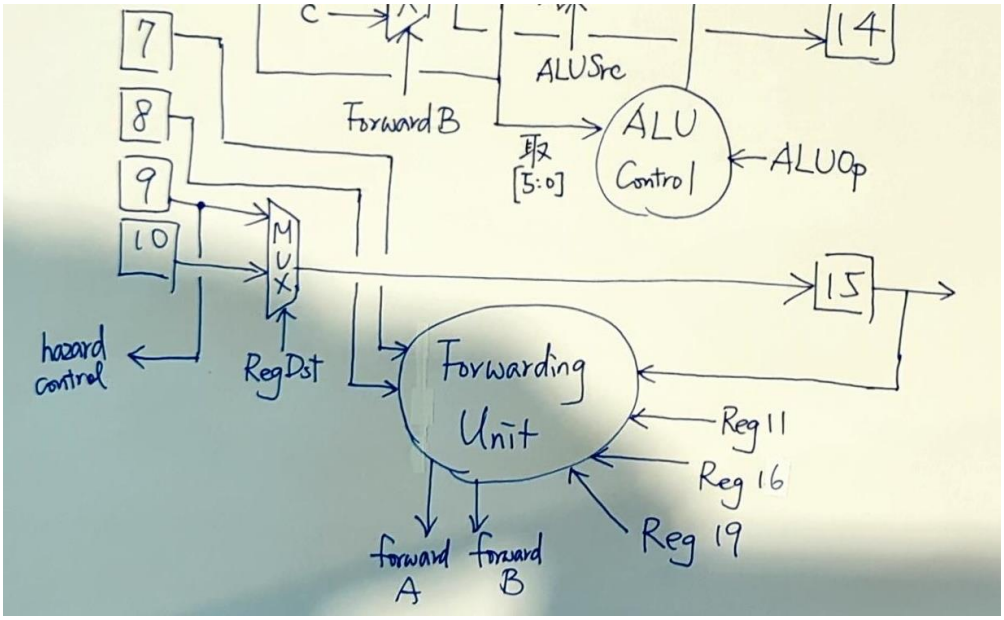

圖 5：Forwarding unit 局部放大圖

在計算機結構中，將 hazard 分為 Read after Write (RAW)、Write after Write (WAW)、Write after Read (WAR)。以下主要將重點放在 RAW。RAW 所造成的 data hazard 可分成兩種情況：

i. write 的指令在 EX 階段而 read 指令在 ID 階段

ii. write 的指令在 MEM 階段而 read 指令在 ID 階段

i 發生的條件為：EX/MEN 的 RegWrite 為 1（表示要寫），EX/MEM 的 Reg 15 不

等於 0 且等於 ID/EX 的 Reg 7, 8（要寫的位置等於後面指令要讀的位置），這種狀況下就必須要 forwarding 直接拉線過去。這就是為什麼 forwarding unit 的輸入會需要 Reg 7, 8, 11, 15。此時 EX 階段 ALU 前面的多工器輸入為 Reg 13 拉線過來。

ii 發生的條件為：不符合 i 的條件，MEM/WB 的 RegWrite 為 1（表示要寫），MEM/WB 的 Reg 16 不為 0 且等於 ID/EX 的 Reg 7, 8（要寫的位置等於後面指令要讀的位置）這種狀況下就必須要 forwarding 直接拉線過去。這就是為什麼 forwarding unit 的輸入會需要 Reg 7, 8, 16, 19。此時 EX 階段 ALU 前面的多工器輸入為 WB 的多工器輸出拉線過來。

然而 lw 指令要在 MEM/WB 才得到值，若下一個指令的 EX 階段就需要對其在 ID 階段欲存取的資料進行運算，無法使用 forwarding，必須要在其後插入 bubble。故需要 hazard control。

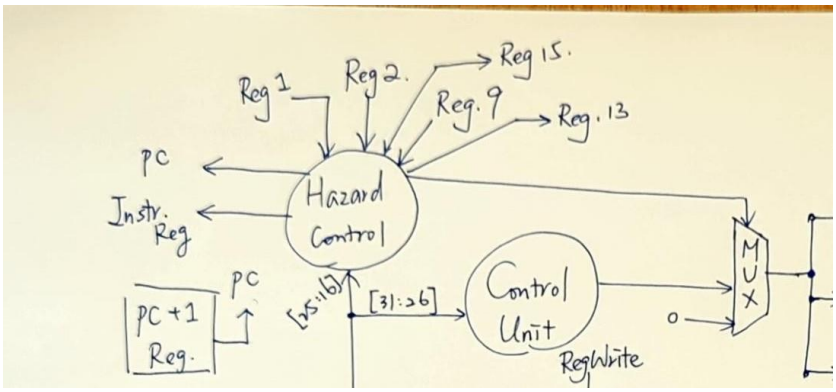

圖 6：hazard control 局部放大

有兩種狀況需要插 bubble，即 data hazard 的 lw，以及 control hazard 的 beq。Load-use hazard 成立狀況如下：ID/EX 的 MemRead = 1 （也就是 lw，存在 Reg 2）且 ID/EX 的 RegisterRt（Reg 9，也就是 lw 將寫入的位置）等於 IF/ID 的 RegisterRs 或 Rt（下一個指令可能要 access 的位置）。條件成立的話要 stall 且插入 bubble。故 hazard control 的輸入需要 instruction[25:21], [20:16]以及 Reg 2, 9。

Branch 在此次期末專題中，其判斷依據涉及前一項指令 slt 的結果，故必須儘早將 slt 的結果拉過來讓 branch 在 ID 階段即可判斷要不要跳，減少 branch delay 的時間。然而 slt 的結果要在 EX/MEM 才知道，來不及與下一個指令的 ID 同步，故需插入 bubble。在這個部分我分成 2 個部分：第一個部分是，如果 opcode 為 beq 且 Reg 1 中的 ID/EX 的 RegWrite 為 1（表示上一個指令要寫入），則先插一個 bubble。第二部分，插過 bubble 後，若 Reg 15（也就是 slt 要寫入的暫存器位址）與 beq 中其中一個比較對象相同，且另一個比較對象之值與 Reg 13 中 ALU 運算（slt）的結果相等，則跳到 instruction [15:0]的位置，若否，則繼續下一個指令。

Jump 的邏輯也類似。雖然 jump 與其他指令無關，但由於有一些 bug，我選擇在 j 前面插一個 bubble。由於 jump 在 ID 階段時，會有下一個指令進入 IF 階段，故 pattern 中終止條件設在指令 18。

II. The problem encountered in this project and how the problem was solved

第一個問題是不太好 debug。在討論區上有看到學長說這 17 個指令應該是將記憶體的內容做 bubble sort 後就比較有感覺在做什麼。Debug 的一些過程會在下面提到。

因為沒有實際上過計算機組織，只有看交大 OCW，故在實作 single cycle 時不知道 address 及各運算階為 signed。實作 single cycle 主要就這個問題比較大。

在實作 pipeline 時，有遇到過以下幾個問題：lw 沒辦法適當 load 進來，sw 沒辦法存進去，beq 跳的點不正確，j 會跳到不對的地方，跳的時候前一個 addi 沒有實作到，slt 會跟不對的 index 比大小，比完的結果存到不對的暫存器，pc + 1 的時機點錯誤，塞 bubble 卻沒有效果，塞錯位置等等。

解決方法直有一個，先把 17 條指令展開成組合語言，然後在看 nWave 的時候把線跟暫存器都攤開來，一條一條追。先 0 到 16 條指令沒問題，前 32 組沒問

題，再到最後全部 Pass。沒有什麼特別的技巧，土法煉鋼。

2. Please make a comparison between the CPU in the HW04 and the one in this final project.

	Single cycle	Pipeline
Area	137106.850417	180582.187839
Cycle time	7.10	6.00
Latency	5143	7969
performance	5006497775.031	8634356729.333

表 1：single cycle 與 pipeline 實作成果比較

3. Platform Architecture (Bonus)

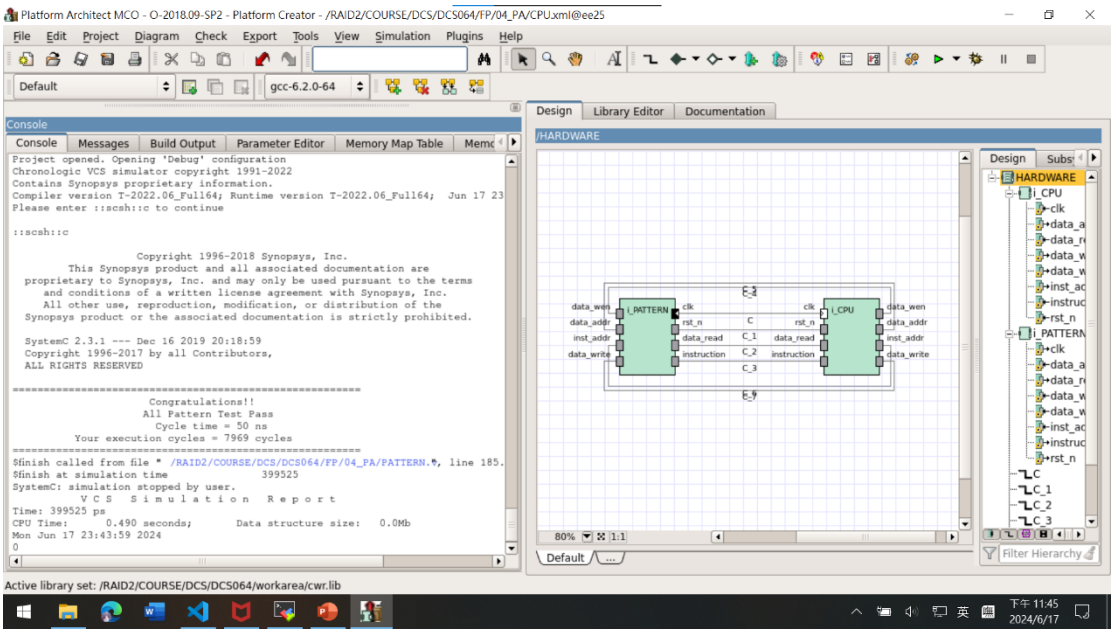

圖 7：PA 結果截圖