計算機結構 lab2

生機四 詹育晟 b10605023

一、 各模組實作說明

ALU_Control.v

負責根據 ALUOp、funct7 和 funct3 產生 ALU 的控制訊號。簡單來說，它就是把不同型態的指令（像是 R-type、I-type、load/store、branch）對應到要執行的運算種類，例如加法、減法、乘法等等。

我們會先把 funct7 和 funct3 合併起來當作 key，再透過 case 判斷對應的操作，例如：

- R-type 會根據 funct7|funct3 來分辨 ADD、SUB、MUL、AND 等；
- I-type 就看 funct3 是哪一種；
- branch 或 load/store 則固定對應 SUB 或 ADD。

如果遇到不支援的組合，會輸出 xxx，代表未定義。

ALU.v

ALU 是整個 CPU 的運算核心，主要負責執行算術與邏輯運算。這個模組的輸入是兩個操作數（src1_i 和 src2_i），加上一個來自 ALU_Control 的控制訊號（ALUCtr_i），根據這個控制碼決定要執行哪一種操作。

支援的操作有：

- 基本邏輯：AND、XOR
- 位移：SLL（左移）、SRAI（有號右移）
- 算術運算：ADD、SUB、MUL、ADDI

BranchUnit.v

負責判斷我們要不要跳轉。它會根據目前的程式計數器（pc_i）和位移（offset_i）來計算跳轉目的地（target_pc_o）。offset 會左移 1 位，因為 RISC-V 分支是以 2 bytes 為單位。

而是否真的跳轉則看 branch_en_i 和兩個操作數 op1_i, op2_i 是否相等。如果 branch_en_i 為真，而且兩個操作數相等，就輸出 take_branch_o = 1，表示要分支。

Control.v

這個模組的功能是根據指令的 opcode 產生對應的控制信號。整個 CPU pipeline 的資料流基本上都靠這裡控制。

它會根據 Op_i（opcode）來分辨指令類型，比如：

- R_TYPE: 就是一般的算術邏輯運算（像 add、sub），會啟用 RegWrite_o。
- I_TYPE_ALU: I-type 的運算（像 addi、srai），會用立即值、寫回 register。
- I_TYPE_LW: Load word，要開啟 MemRead_o，並從 memory 存回 register。
- S_TYPE: Store word，要開啟 MemWrite_o。
- SB_TYPE: 分支跳躍的指令，會打開 Branch_o。

有一個小設計是 NoOp_i，如果 pipeline 插入 bubble 的話，就會讓所有控制訊號都變成 0（等於這一拍沒做事）。

CPU.v

這個模組負責把所有模組整合起來，讓資料能夠在不同階段順利流動。

它的功能是：

- **整合與連接所有模組**：像是 ALU、Control、Register、Memory、BranchUnit、各種 MUX、以及各級 Pipeline Register。
- **模擬 RISC-V 指令的執行流程**：從抓指令（IF）、解碼（ID）、執行（EX）、存取記憶體（MEM）、最後寫回（WB）全部都涵蓋。
- **處理資料冒險與控制冒險**：透過 Forwarder 和 Hazard Detection 模組避免錯誤結果。
- **實作分支與暫停邏輯**：遇到分支或需要 stall 的情況時會自動插入 bubble 或停住某些部分。

把所有「功能零件」排成一個有效率的組裝線，讓每個 clock cycle 都有不同指令在不同階段被處理，實現 pipeline CPU。

EX_MEM.v

這個模組是 EX 階段與 MEM 階段之間的流水線暫存器，負責在時脈上升沿時把 EX 的輸出資料與控制訊號穩定地傳遞給下一個階段。裡面處理的內容包含：

- ALU 計算結果（ALUout）
- 要寫入記憶體的資料（WriteData）
- 寫入目的暫存器位置（Rd）
- 一連串控制訊號：RegWrite, MemtoReg, MemRead, MemWrite

若 rst_i 為 low，則所有輸出會被清成預設值，這樣可以避免剛 reset 時 pipeline 有錯誤訊號流動。這個模組看起來簡單，但確保資料穩定傳遞是 pipeline 正確運作的核心。

Forwarder.v

這個模組是處理 **forwarding（資料轉送）** 的關鍵元件，用來解決資料冒險（data hazard）問題。它會比對目前 EX 階段用到的暫存器（EX_Rs1_i, EX_Rs2_i）是否剛好是前面 MEM 或 WB 階段即將寫回的 register（MEM_Rd_i, WB_Rd_i）。

如果比對成功，代表結果還沒寫回，但已經被後面的指令需要，我們就透過 **多工器（MUX）** 轉送資料，避免 pipeline stall。

轉送邏輯如下：

- 優先從 MEM 階段轉送，因為離 EX 比較近。
- 若 MEM 沒有對應，再從 WB 階段拿。
- 兩個 output 分別控制第一與第二個 operand：Forward_A_o 與 Forward_B_o。

這樣的 forwarding 機制可以讓不需要 insert bubble，可以提高效能。

Hazard_Detection.v

這個模組負責偵測 load-use hazard，也就是當 EX 階段正在從記憶體讀資料（EX_MemRead_i = 1），而這筆資料剛好是 ID 階段的指令要用的 register（ID_Rs1_i 或 ID_Rs2_i）時，這時 pipeline 就需要 **插入 bubble** 來避免資料還沒準備好就被使用的問題。

三個輸出：

- NoOp_o：通知 Control 單元要產生 nop 指令（等於 bubble）
- PCWrite_o：停住 PC 不要更新
- Stall_o：停住 IF/ID 階段的寫入

ID_EX.v

ID_EX 是 ID 階段與 EX 階段之間的暫存器，負責將解碼後的資訊傳遞到執行階段。這個模組會把所有從 ID 階段輸出的資料（像是暫存器資料、立即數、控制訊號等）全部保存在 flip-flop 中，並在每個 clock cycle 更新。

如果 rst_i 被拉低，所有輸出都會被清空；否則就正常更新。這樣設計是為了支援 pipeline 架構，確保每一階段的資料都正確對應。

IF_ID.v

IF_ID 是 Instruction Fetch（IF）和 Instruction Decode（ID）之間的 pipeline 暫存器，負責保存當前指令與其對應的 PC 值。

在時脈上升沿時，如果 rst_i 被拉低，表示進行 reset，所有輸出會清為 0。 若是 Stall_i 為 1，代表遇到 hazard，則不做更新（維持原值）。
而 Flush_i 用來處理 branch 指令錯誤預測的情況，會將暫存器清為 0 插入泡沫（bubble）。在一般情況下，模組會把來自 IF 階段的指令與 PC 傳到 ID 階段。

Imm.v

Imm 模組是用來根據不同的指令格式（I-type、S-type、SB-type）去擷取出 12 位元的 immediate 值。這個值會在後續階段（像是 ALU 運算或記憶體存取）當作操作數使用。

根據 opcode 判斷是哪種類型的指令，然後就把指令中對應的欄位擷取出來給 immed_o。
像是：

- I-type 直接取 bits[31:20]；
- S-type 把 bits[31:25] 和 bits[11:7] 接起來；
- SB-type 比較特別，要把 sign bit 和其餘 bits 拼起來。

MEM_WB.v

這個模組負責從 Memory 階段把資料傳到 Write Back 階段。它會把 memory load 出來的資料、ALU 結果、目的暫存器號碼（Rd）以及控制訊號（是否寫回暫存器、是否來自 memory）一併存起來，並在下一個 clock cycle 傳給 WB 階段。

如果有 reset，全部會清成 0；否則就正常更新。這個模組本身邏輯很單純，但少了它，整個 WB 階段就沒東西能用，算是 pipeline 中的搬運工。

MUX32_Double.v

這個模組是個 4 選 1 的多工器（Multiplexer），用來選擇 ALU 的輸入資料來源，主要應用在資料轉發（Forwarding）機制上。輸入是四個 32-bit 資料和一個 2-bit 的控制訊號 select_i，根據這個控制訊號來決定要輸出哪一組資料，像是從 EX、MEM 或 WB 階段來的結果。

MUX32_Double.v

MUX32 是個簡單的 2-to-1 多工器，用來在兩個 32-bit 資料中選擇一個作為輸出。它透過 select_i 控制信號來決定輸出 src0_i 還是 src1_i，如果為 0 則選擇 src0_i，否則選擇 src1_i。這個模組在很多地方都用得到，例如控制 ALU 的輸入來源、寫回階段的資料來源等，是處理資料流的一個基礎元件。

Sign_Extend.v

Sign_Extend 模組的功能是將 12 位元的立即數（data_i）進行符號擴展成 32 位元。實作方式是根據第 11 位（最高位）是否為 1，來決定要補 0 還是補 1，讓結果符合帶符號數的格式。這個模組在處理像是 load、store、立即數運算時會用到，是資料解碼中的基本步驟。

Simple_Adder.v

這個模組就是一個簡單的 32-bit 加法器，負責計算下一個 PC（通常是 pc + 4），讓指令可以往下執行。輸入是兩個 32-bit 的整數，輸出就是它們的總和，邏輯非常直覺，整體就是用在程式計數器的更新上。

二、實作困難與解法

1. **Hazard 與 流水線清除 (Flush)**
    - **困難**：同時處理 load-use stall、分支 flush 邏輯較易衝突
    - **解法**：將 NoOp、PCWrite、IFID_Write 與清除控制訊號分開，並在 ID/EX 暫存器中 clear 控制位元，確保分支和 load stall 狀態互不干涉。
2. **轉發 (Forwarding) 實作**
    - **困難**：兩層轉發條件（MEM → EX、WB → EX）序要正確
    - **解法**：先檢查 MEM RegWrite，再檢查 WB；用 if/else 結構保證優先順序。
3. **TestBench 維度對齊**
    - **困難**：TestBench 會直接操作內部記憶體與暫存器陣列，需要正確命名與層級
    - **解法**：保持綁定名稱與 TestBench 一致，並在最外層 CPU.v expose Instruction_Memory.memory 與 Registers.register。

三、開發環境

- **作業系統**：Ubuntu 22.04
- **模擬器**：iverilog 11.0
- **自動化測試框架**：PECO (NTU ECLab)
- **編輯器**：VS Code + Verilog 插件