Computer Organization HW3

110550088 李杰穎

May 4, 2023

1 Architecture Diagrams

本次作業的架構圖與 spec 相同，可參照 Figure 1。比較特別的是 ALU Control 輸出的 `leftRight` 實際上是 `ALU_operation[3]`。

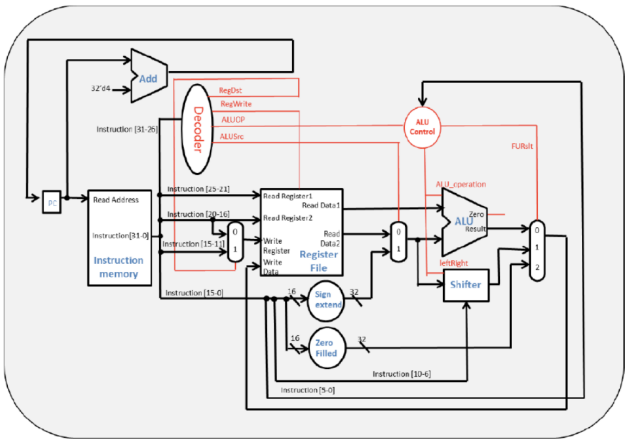

Figure 1: 本次作業的整體架構圖

2 Hardware Module Analysis

本次作業共需要完成以下 11 個 modules：

1. `Adder`
2. `Mux2to1`
3. `Mux3to1`
4. `ALU_1bit`
5. `ALU`
6. `Shifter`
7. `Sign_Extend`
8. `Zero_Filled`
9. `ALU_Ctrl`
10. `Decoder`
11. `Simple_Single_CPU`

接下來會介紹每個 module 的功能及實作細節。

2.1 `Adder`

此 module 功能為相加兩個 32 bit 的數字，故實作細節相當簡單。即利用 `assign` 去將相加後的結果 assign 到 output wire 上。

2.2 `Mux2to1`

此 module 為一個 2 to 1 MUX，利用一個 1 bit 的 select 去選擇輸出的數值是哪一個輸入。值得注意的是，此 module 使用 `parameter size` 去控制輸入及輸出位元的數量。

實作主要是利用三元運算子去進行類似於 if-else 的輸出控制。

2.3 `Mux3to1`

此 module 為一個 3 to 1 MUX，利用一個 2 bit 的 select 去選擇輸出的數值是哪一個輸入。其餘細節皆與 `Mux2to1` 類似。

2.4 `ALU_1bit`

本 module 是使用前次作業的程式碼，主要就是根據 `operation, invertA, invertB, carryIn, less` 去輸出 1 bit 的 ALU 計算結果。實作細節就不再贅述。

2.5 `ALU`

本 module 是利用 32 個 `ALU_1bit` module 去計算 32 bit 的 ALU 結果，值得注意的是最低位元 bit 的 `set` input 是最高位元 ALU 中加法器的計算結果。且為了讓中間 30 個 ALU 可以更簡單的被產生，使用了 `generate, for` 去重複產生 1-bit ALU module。

2.6 Shifter

本 module 為輸入一 32 bit 的數字，將其根據 `shamt`, `leftRight`，進行左移或右移的操作。實作細節就是同樣利用三元運算子去進行 `leftRight` 的判斷，再利用 «, » 兩個 operator 去進行左移和右移的操作。值得注意的是因為我們是實作 `sll`, `srl` 這兩個 instruction，所以我們並不需要根據最高位元去進行補 1 或 0 的判斷。

2.7 Sign_Extend

本 module 為輸入一 16 bit 的數字，利用其最高位元判斷正負性，再將其 extend 成 32 bit 的數字。這邊主要的細節是利用 `{16{data_i[15]}}` 這個語法去做到對於最高的 16 bits 填入第 16 bit 的值。

2.8 Zero_Filled

本 module 在此次作業並沒有用到，但其功能與 `Sign_Extend` 類似，都是將 16 bit 的數字擴展到 32 bit，不同的是本 module 不會根據 16 bit 的最高位判斷最高的 16 bit 要填上 1 或是 0，而是統一填上 0。

2.9 Decoder

本 module 是根據 instruction 中的最高 6 bit 的 `opcode`，輸出用於控制 ALU 的 `ALUOp` 訊號、控制 ALU 輸入的 `ALUSrc`、控制 register memory 是否啟用寫入的 `RegWrite` 及寫入 register 的 address `RegDst`。

2.10 ALU_Ctrl

本 module 會根據 `Decoder` 的輸出，`ALUOp` 及 instruction 最低的 6 bit，也就是 `funct` 生成用於控制 ALU 的 `ALU_operation` 及控制 write data 的輸入的 MUX 的 select `FURSlt`。

實作部分也是利用三元運算子，根據 `ALU_operation` 及 `funct` 去計算出正確的輸出。

2.11 Simple_Single_CPU

本 module 為本次作業的核心 module，其將以上的 module 組合起來，構造出本次作業要求的 Simple Single Cycle CPU。實作細節與 spec 中描述的相同，基本上就是生成出不同的 wire，再將 wire 連接到正確的 module 上。

3 Finished Part

本次作業我完成了所有需要完成的 module，並也通過助教提供的三個測資。

4 Problems and Solutions

在第一次接完線之後，發現並不能輸出正確的答案，且輸出的結果為 ×。利用在 test bench 中增加 `$display` 進行 debug 後發現問題出在 ALU 中，最後才發現原來是在將 `ALU_operation_i` 分解出 `invertA`、`invertB` 及 `operation` 時，錯誤的將 wire 取名為 `invert_A`，導致無法執行出正確的結果，導致最終結果變成 ×。

5 Summary

在本次作業中，我完成了簡單的 Single Cycle CPU，此 CPU 僅支援有限的 ISA，如 MIPS 中的 `lw`, `sw`, `beq` 等 instruction 都尚不能在本次實作的 CPU 中使用，且本次作業的 CPU 也沒有加上 Data Memory，這也使得支援的指令有一定的受限。但是在本次作業中，我還是大致理解 CPU 的實作流程，並能夠正確的使用 verilog 實作出可以正確運作的 CPU。