國 立 台 灣 科 技 大 學

計 算 機 組 織

指 導 教 授：陳雅淑 教 授

**------------------------------------------------------------------------**

計算機組織作業PA2報告

班 級 ： 四電機三甲

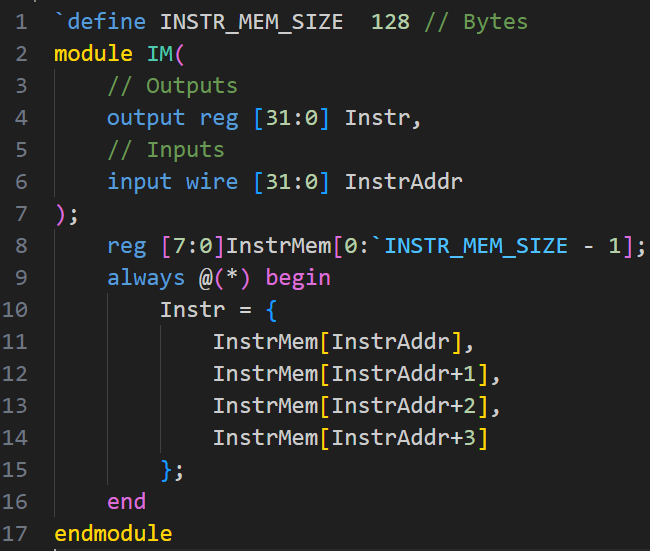
學 生 ： 呂佳祐

學 號 ： B11107055

SimpleCPU ： Area: 33892.922、Slack: 2.0369

1. **Instruction Memory**

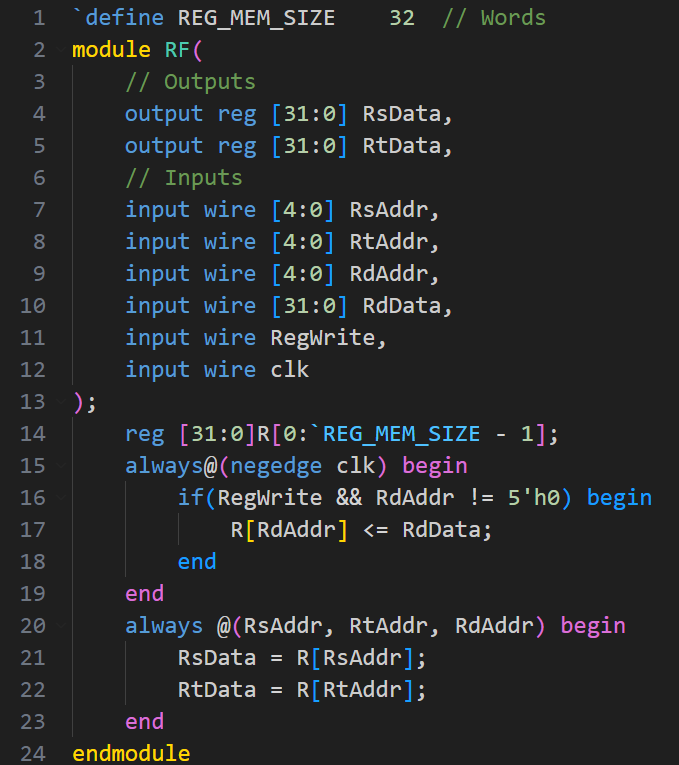
指令記憶體用於儲存程式的指令，由總共128格、大小為1byte的空間組成。



IM.v程式碼

記憶體透過輸入指定的記憶體位置InstrAddr，輸出從該位址算起的後四個空間組成的指令，總共4個bytes。本作業使用的是MIPS架構，資料排序為Big endian，所以在輸出的值中，最高位元會是最上層位址空間的資料，而最低位元則是最下層位址空間的資料。

1. **Register File**



RF.v程式碼

暫存器堆由32個大小為4個bytes的暫存器組成，為運算資料暫時保存的地方。若要寫入資料，則資料會在時鐘上緣時被存入由RdAddr指定位址的暫存器儲存。若要讀取資料，則透過輸入的RsAddr及RtAddr，來讀取該位分別對應的Source Register及Target Register，讀取資料與時鐘無關。

1. **Data Memory**

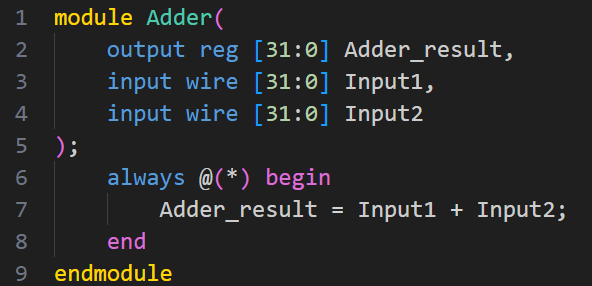


DM.v程式碼

資料記憶體用於儲存相比於暫存器，更不須被立馬運算，且資料保存時間較長。其由128格大小為1個Byte的空間組成。

資料記憶體在時鐘上緣時輸出資料，且在時鐘下緣時讀取輸入資料。不管是輸出還是輸入資料，都是Big endian。

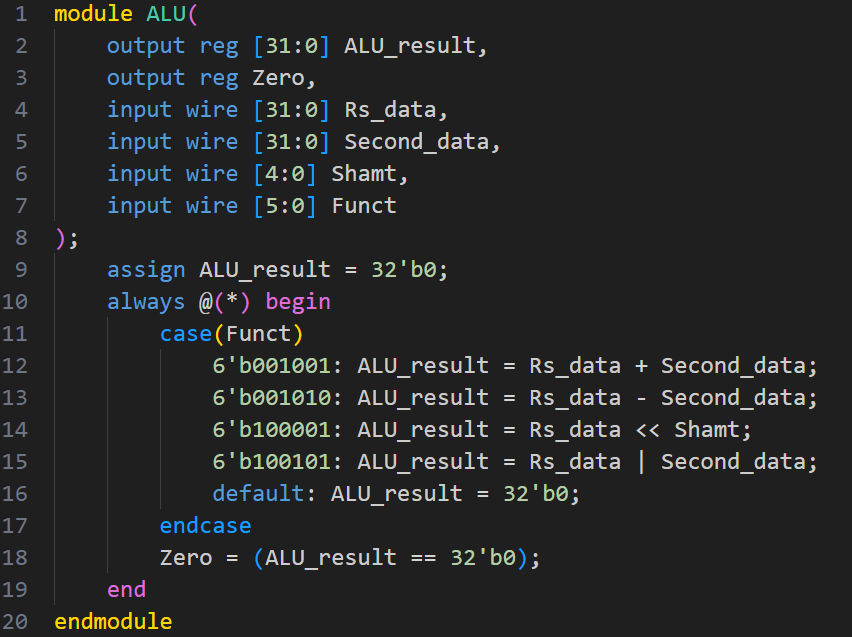
1. **Adder**



Adder.v程式碼

加法器用於計算下一個指令或是beq成立跳躍的位址。

1. **ALU**

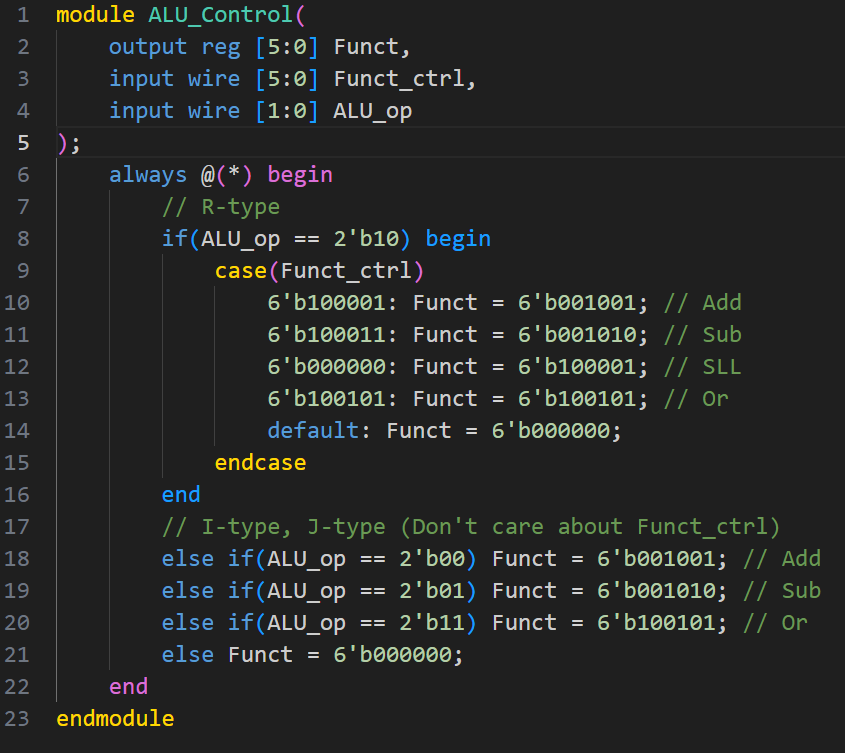


ALU.v程式碼

ALU負責處理兩輸入資料的運算，且運算功能會由輸入的Funct指定。若當運算結果為0時，Zero會被設為1，用於判斷beq指令是否成立。各Funct對應功能如下：

|  |  |
| --- | --- |
| Funct Code | Corresponding Function |
| 001001 | Addition |
| 001010 | Substraction |
| 100001 | Shift Left Logical |
| 100101 | Or |

1. **ALU Controller**



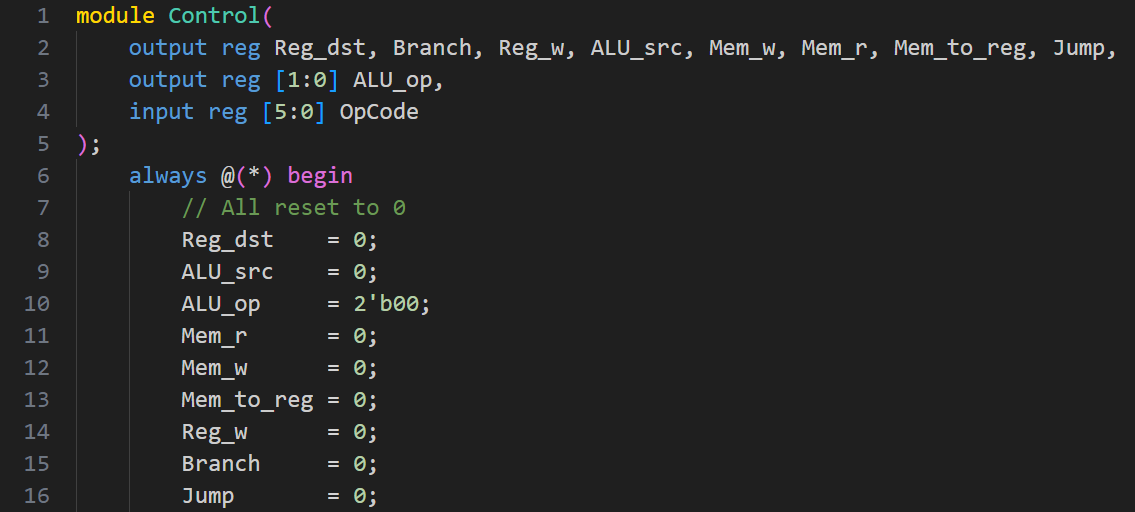
ALU\_Control.v程式碼

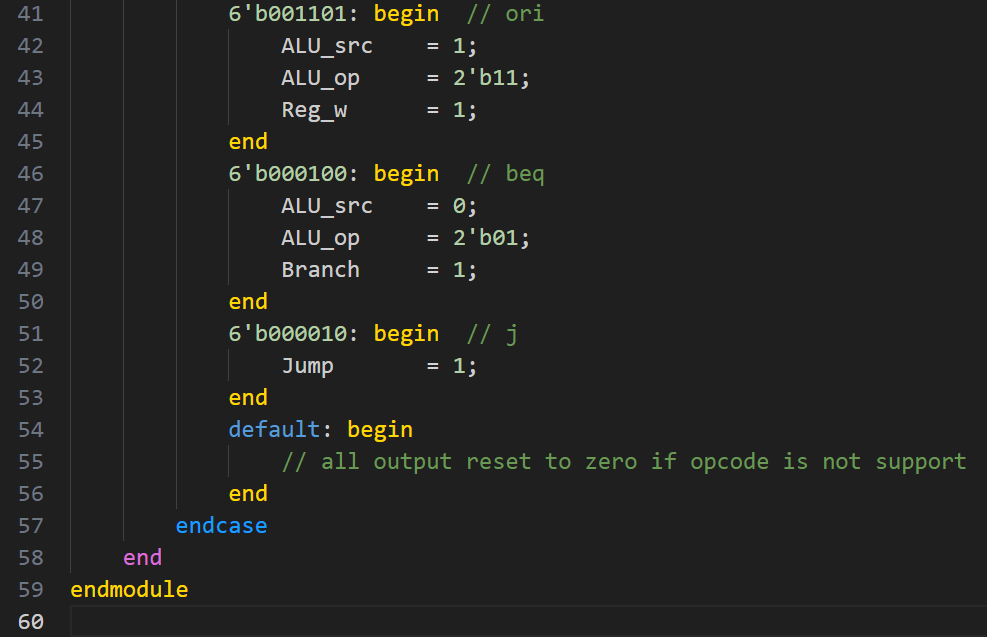
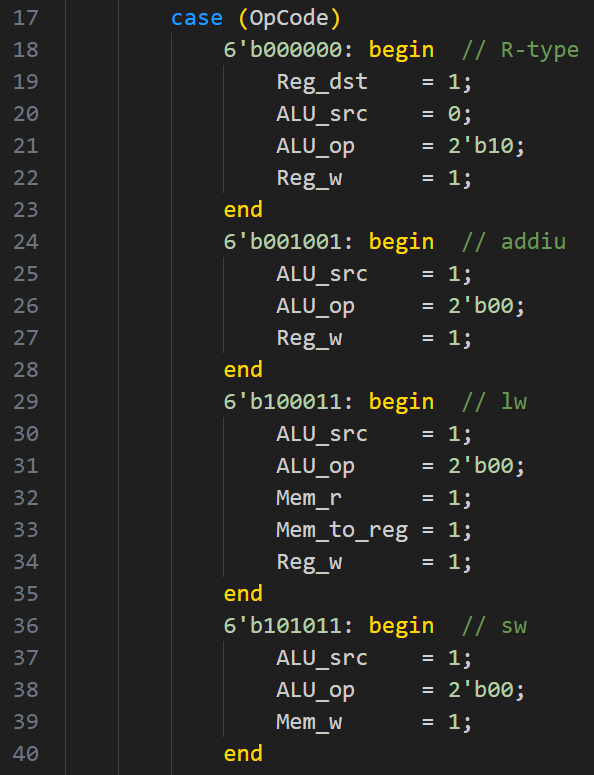
MIPS架構中，程式指令分為三個類型，分別是R-type、I-type及J-type，為使ALU功能簡單，所以處理分別指令類型的邏輯判斷，由ALU\_Control及Control負責。

在ALU\_Control中，決定要使用什麼運算功能由兩個輸入資料決定，分別是在R-type的Instruction裡的Funct\_ctrl，以及由Control輸出的ALU\_op。對應功能如下：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Instruction Type | ALU\_op | Funct\_ctrl | Correspoding function | MIPS Instruction |
| R-type | 10 | 100001 | Add unsigned | addu |
| 100011 | Sub unsigned | subu |
| 000000 | Shift Left Logic | sll |
| 100101 | Or | or |
| I-type、J-type | 00 | Don’t care | Add imm unsigned | addiu, sw, lw |
| 01 | Sub imm unsigned | beq, j |
| 11 | Or imm | ori |

1. **Controller**



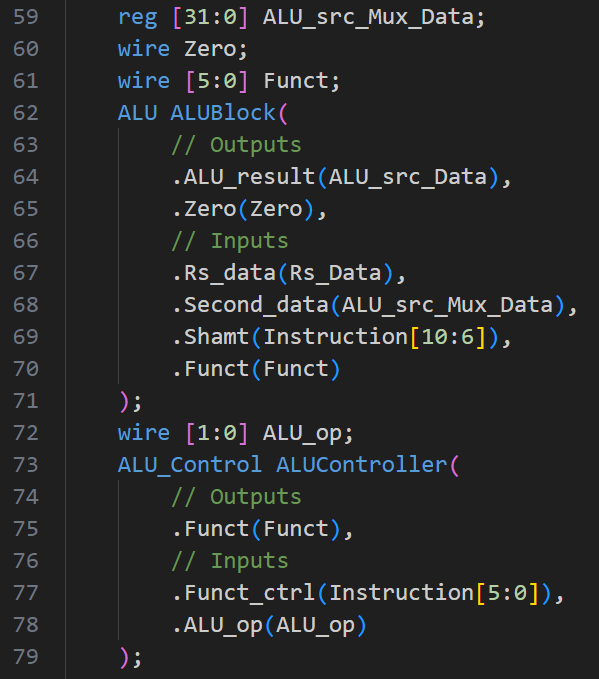
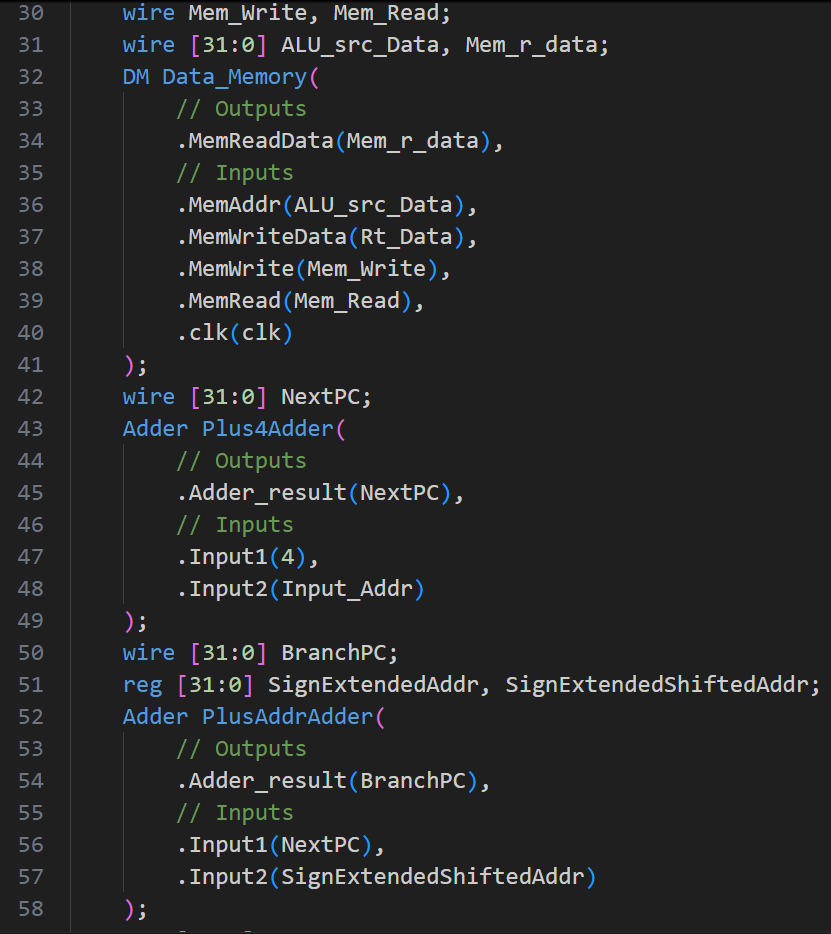
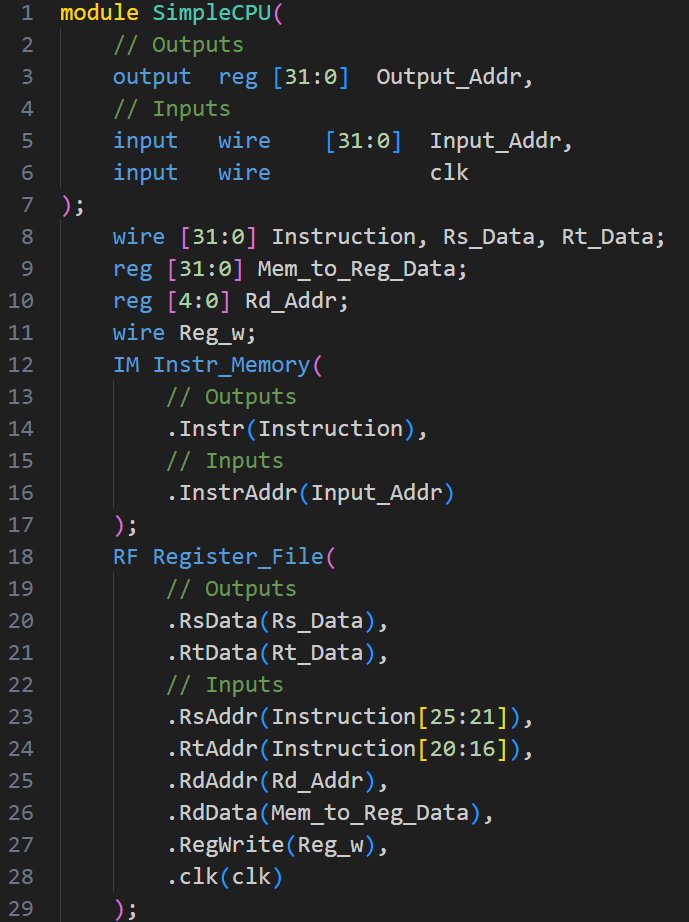


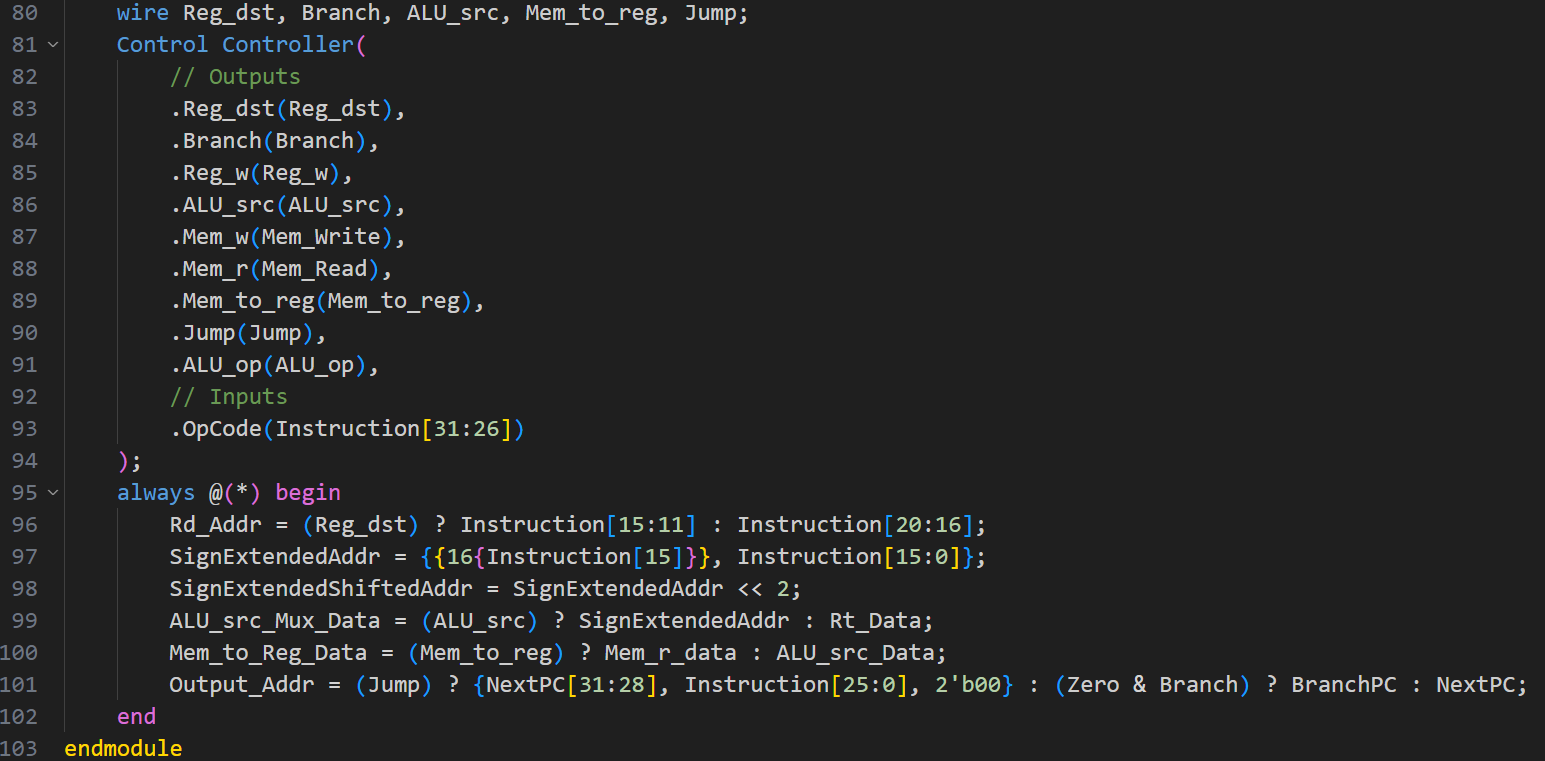
Control.v程式碼

控制器猶如整顆CPU的指揮官，它會根據Instruction的Opcode，判斷指令是什麼類型，控制各多工器的輸出、記憶體的讀寫狀態以及向ALU Control輸出對應的ALU\_op。

控制器會先將所有輸出設為0，再根據Instruction，將對應的輸出設為1；如果Opcode不是規定的值，控制器會將所有輸出預設為0。各Opcode對應的輸出可以參考圖上的程式碼。

1. **Simple CPU**

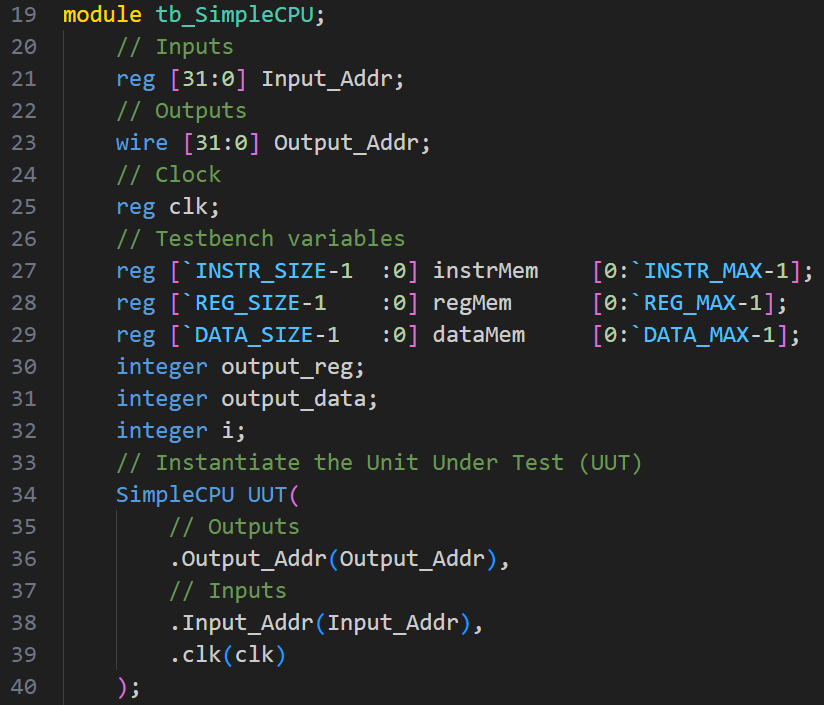
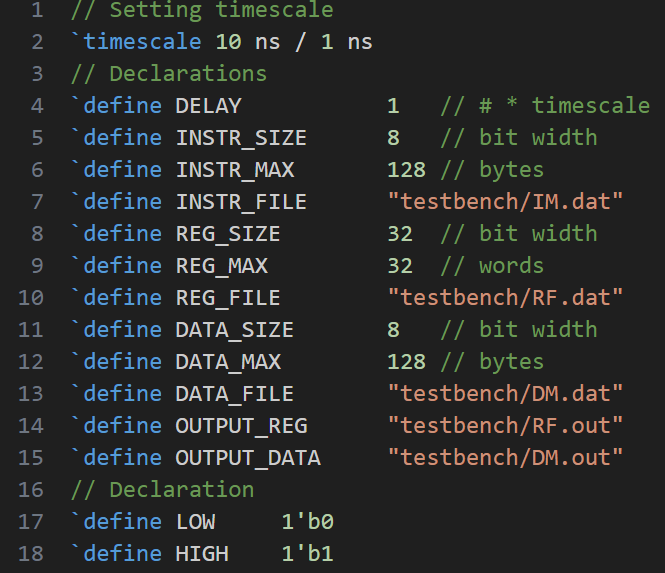




SimpleCPU.v程式碼

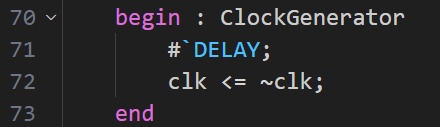
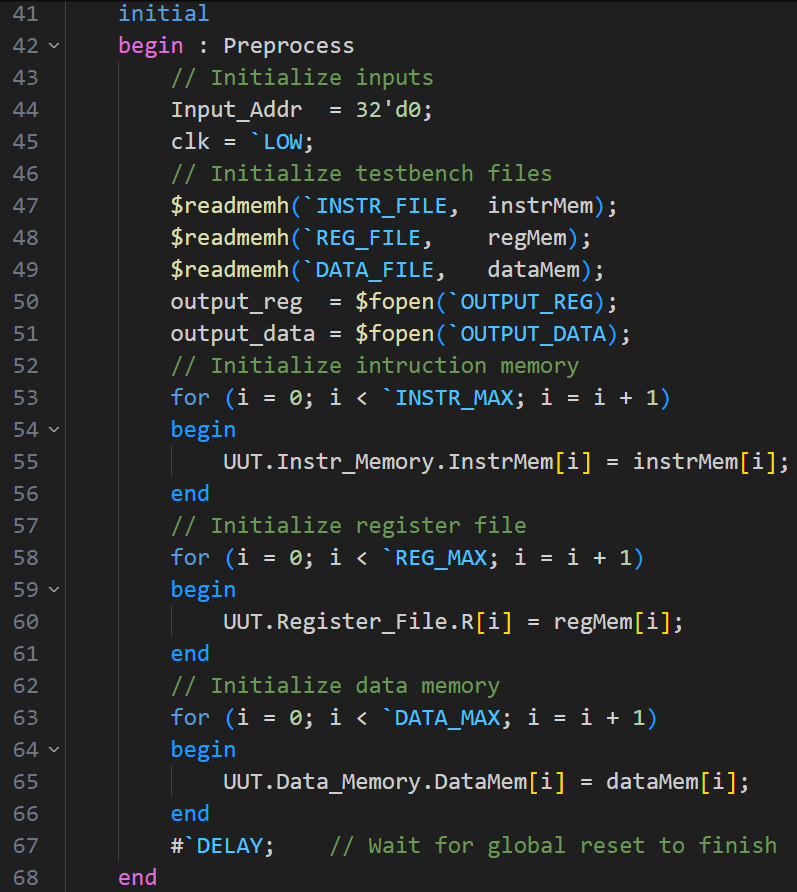
SimpleCPU將所有模組整合，連接個模組對應腳位，並且將所有多工器、Sign Extend、位址左移兩位元在此實作。若是單純連接兩模組的導線就以wire宣告；若是在SimpleCPU裡會被賦值的導線，就會以reg宣告，例如多工器的輸出。

1. **Testbench and Simulation**
2. Program of Testbench of SimpleCPU



tb\_SimpleCPU.v程式碼－宣告變數與連接模組等前置動作

設定時間單位為10 ns、時間精度1 ns。定義所有常數、宣告變數並連接對應SimpleCPU模組的腳位。

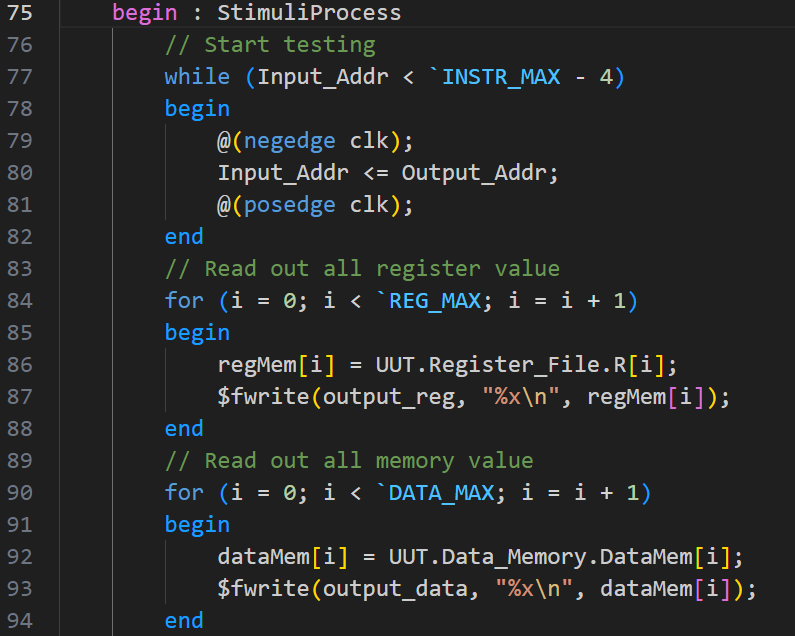


tb\_SimpleCPU.v程式碼－預處理與時鐘設定

Preprocess區塊負責開啟IM.dat、RF.dat、DM.dat這三個輸入檔，並讀取其中資料，將資料分別存入Instruction Memory、Register File、Data Memory中。

ClockGenerator區塊負責產生週期為2個時間單位的時鐘訊號。且為使等等從IM的第一個指令開始運作，所以在Preprocess中有定義時鐘訊號會先從低位元開始(Line 45: clk = `LOW;)。

StimuliProcess區塊開始執行IM裡的程式，並將本次的輸出位址設成下次執行指令的位址。程式結束後將Register File及Data Memory資料，存到RF.out及DM.out檔案中，最後在終端輸出可愛的顏文字。



tb\_SimpleCPU.v程式碼－執行IM指令與輸出結果

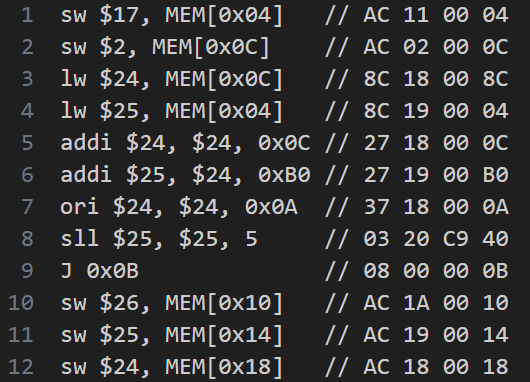
/**\\**\_\_/**\\** /**\\**\_\_/**\\** /**\\**\_\_/**\\**

(O . O) (–.–) ( ^ **\_** ^ )

> ^ < > ^ < > ^ <

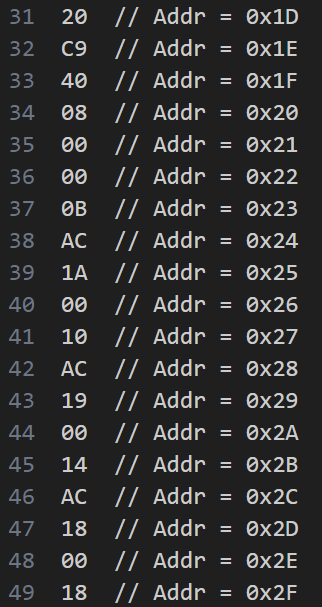
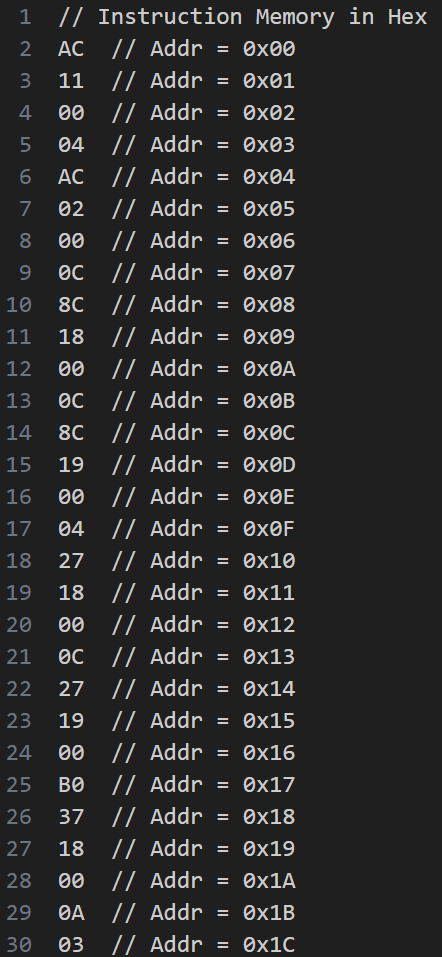
1. Test Pattern

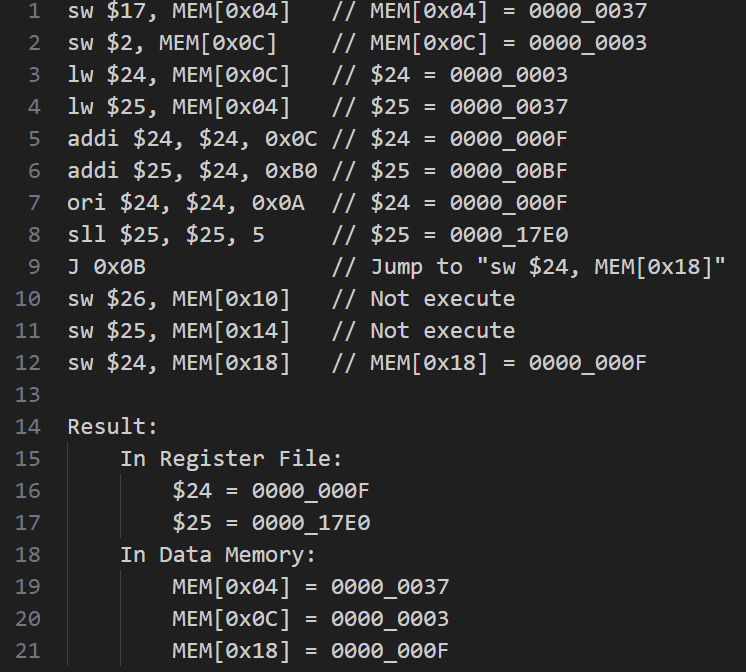
要測試的Instruction，以組合語言表達，且其轉換成十六進制的指令，分別註解在每行程式後：



每行程式碼執行後，某記憶體空間或暫存器的值如下圖每行程式碼後的註解所示，最後結果也統整在Result區塊裡：

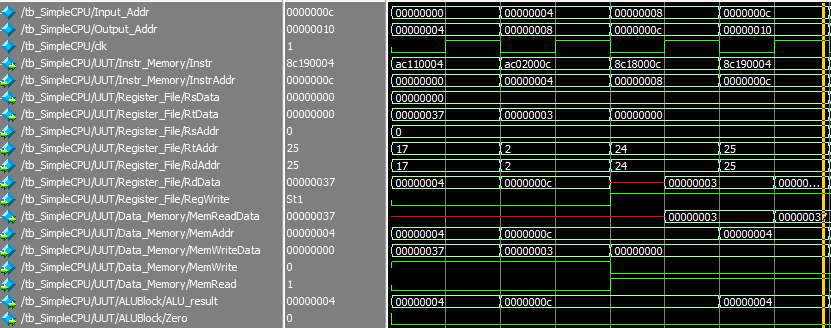
把事先模擬的指令存入IM.dat：

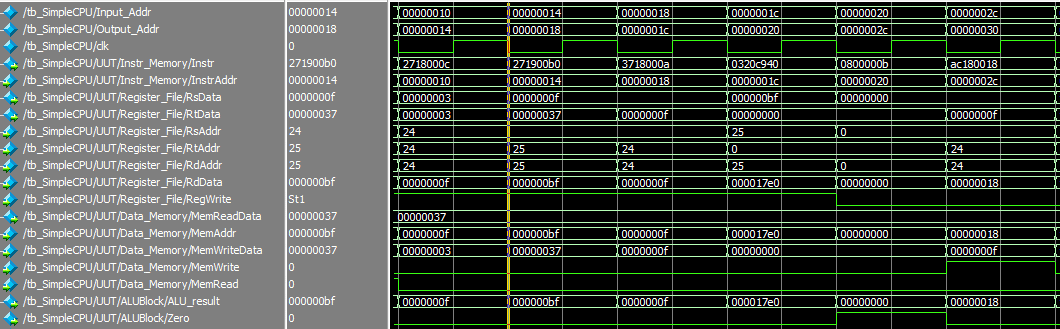




從Addr = 0x30後都是FF，所以就沒列出。

1. Simulation Result Waveform

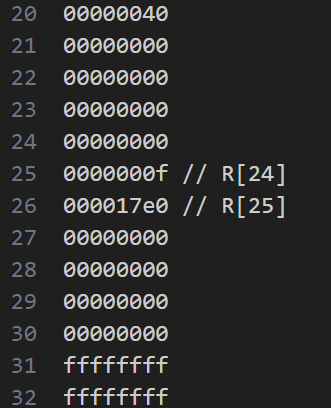




Simulation Waveform

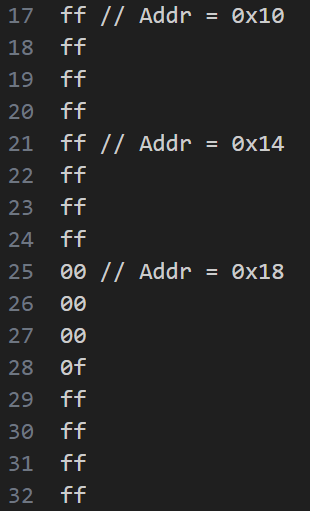
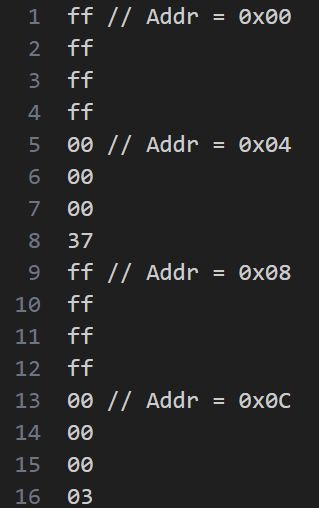
可以看到CPU的Input\_Addr及Output\_Addr是跟預期的一樣，Jump指令也有正確跳到指定位址，然後控制器的輸出也有在正確的時機輸出正確的值，整體看起來是沒問題的。

1. RF.out and DM.out after simulation



RF.out

可以看到在Register File裡，$24跟$25如預期存入了0000\_000F和0000\_17E0。



DM.out

可以看到Data Memory的MEM[0x04]是0000\_0037、MEM[0x0C]是0000\_0003、MEM[0x18]是0000\_000F，跟上面預期的一樣。

根據上面所有的測試，可以得知SimpCPU及其他模組的功能是正常的。

1. **Datapath Rethinking**

建立一個乘數與除數共用的暫存器A，其輸入來自ALU\_src多工器的輸出，並將ALU\_src的輸出與A的輸出，各自連接到一顆新多工器X的輸入，並將X的輸出連接到ALU的其中一個輸入。如果要使用乘除法運算就將X的輸出導向到A的輸出；如果不使用乘除法，就將X的輸出導向到ALU\_src的輸出。

建立一個64-bits的暫存器B，商數/餘數放置在左32-bits，被乘數/被除數放置在右32-bits。將RF的Rs\_data輸出到B的右半部，且將Rs\_data與B的左半部輸出到一顆新多工器Y，並將Y的輸出連接到ALU另一個輸入。若要使用乘除法運算，就將Y的輸出導向到B的左半部；若不使用乘除法運算，就將Y的輸出導向到Rs\_data。

Controller控制ALU Controller要不要執行乘除法運算，控制多工器X、Y。

ALU Controller控制暫存器A、B要不要從ALU\_src與Rs\_data接收資料，控制B的位元左右移，控制B的左半部要不要寫入ALU，控制ALU的運算功能，檢查是否乘除法運算結束。

1. **Conclusion and insights**

我覺得Single Cycle的CPU算是蠻好做的，也是個很適合新手入門的專案。未來後，在這基礎上加入pipeline，會更好實作。

在測試的時候我覺得比較麻煩的是指令的轉換，從組合語言轉成4-bytes十六進制的形式。如果像我一樣，一個一個位元寫出來，用計算機轉十六進制，再一個一個貼到IM.dat裡，那會有點花時間。但是當程式跑出來的結果，跟預期想的一樣，那就很有成就感。

這次的CPU雖來看起來datapath很複雜，但只要釐清之間的關係，加上每個模組的分工清晰，其實做起來很快，要找bug沒那麼難。總之，這次作業讓我更熟悉上課內容，從講義上抽象的概念，變成實際看的到的結果，我想這能讓我更了解課堂內容的運作邏輯，也可以從實作中學到課堂上沒教到的、更細節的內容。