Computer Organization Project

Lab3 – Single Cycle CPU

1. 目標 Goal

利用 Lab2 中所定義的算術邏輯單元(Arithmetic Logic Unit, ALU)與移位單元(Shifter)，實作一個 Single-Cycle CPU。CPU 為電腦架構中重要的元件，本次 LAB 需要組出 CPU 之 Datapath，並加上控制電路，讓 CPU 能存取暫存器與記憶體，來執行 arithmetic、logic、shift、load、store、branch、jump 等類指令。(本實驗單元各模組可用 RTL-level 的 Verilog 語法來寫，不限制只能用 gate-level 的方式來實作)

Implement a Single Cycle CPU with the ALU and shifter defined in Lab1. CPU is the most important unit in computer system. In Lab3, construct the datapath of the CPU and add the control circuit to access the Register File in CPU and memory and execute the arithmetic, logic, shift, load, store, branch, and jump instructions. (In Lab3, you can design your modules with RTL-level in Verilog besides gate-level.)

2. 實驗說明 Lab Description

A. 附加檔案 Attached files

本單元所附的 CO-Lab3.zip 中，包含 Simple_Single_CPU.v、Adder.v、ALU.v、ALU_Ctrl.v、Decoder.v、Data_Memory.v、Instr_Memory.v、Mux2to1.v、Mux3to1.v、Program_Counter.v、Reg_File.v、Shifter.v、Sign_Extend.v、Zero_Filled.v 等電路模組，及一個測試模組 TestBench.v。本次 Lab 中，會提供 Data_Memory.v、Instr_Memory.v、Program_Counter.v 及 Reg_File.v 等內容完整的檔案，同學需完成其他模組，讓 CPU 可以運作。在實作中，如有需要，可新增額外的模組(.v 檔)來完成。

The attached file “CO-Lab3.zip” is composed of “Simple_Single_CPU.v”, “Adder.v”, “ALU.v”, “ALU_Ctrl.v”, “Decoder.v”, “Instr_Memory.v”, “Mux2to1.v”, “Mux3to1.v”, “Program_Counter.v”, “Reg_File.v”, “Shifter.v”, “Sign_Extend.v”, “Zero_Filled.v” and “TestBench.v”. Please use these modules to accomplish the design of your CPU. In Lab3, the complete modules of “Data_Memory.v”, “Instr_Memory.v”, “Program_Counter.v”, and “Reg_File.v” are provided. You should accomplish other modules to make this CPU work. You may create additional module (.v file) for your design, if necessary.

B. 基本 Single-Cycle CPU 設計 Basic Single-Cycle CPU Design (100%)

此 CPU 的指令為 16-bit 的長度，指令有 arithmetic、logic、shift、load、store、branch、及 jump 的類型，各指令請參照下列的格式和功能來實作。

Each instruction of this CPU is 16 bits. There are arithmetic、logic、shift、load、store、branch、and jump instructions in this CPU. Follow the below formats to implement Each instruction.

(a) 指令格式 Instruction formats

R-type

op (3 bits)	rs (3 bits)	rt (3 bits)	rd (3 bits)	funct (4 bits)

當指令 OP code 為 R-type (3’d000)的時候，對應的運算指令須參考 function field。
When OP code of an instruction is R-type(3’d0), you should use the function field to decode the corresponding operation.

I-type

op (3 bits)	rs (3 bits)	rt (3 bits)	constant/address (7 bits)

J-type

op (3 bits)	address(13 bits)

(b) 基本指令集 Basic instructions

所設計的 CPU 必須能夠執行以下指令，依指令格式分別說明於下表中：
The CPU should be able to execute the below instructions.

表一：R-type 指令定義與範例
Table 1: The definition and format of R-type instructions

指令名稱	Meaning	R-type 指令格式				
		Op field	rs	rt	rd	funct
add (addition)	rd = rs + rt	0 (3’b000)	rs[12:10]	rt[9:7]	rd[6:4]	0 (4'b0000)
sub (subtraction)	rd = rs – rt	0 (3’b000)	rs[12:10]	rt[9:7]	rd[6:4]	1 (4'b0001)
and (logic and)	rd = rs&rt	0 (3’b000)	rs[12:10]	rt[9:7]	rd[6:4]	2 (4'b0010)
or (logic or)	rd = rs|rt	0 (3’b000)	rs[12:10]	rt[9:7]	rd[6:4]	3 (4'b0011)
nor (logic nor)	rd = ~(rs|rt)	0 (3’b000)	rs[12:10]	rt[9:7]	rd[6:4]	4 (4'b0100)
slt (set on less than)	if (rs<rt) rd = 1 else rd = 0	0 (3’b000)	rs[12:10]	rt[9:7]	rd[6:4]	5 (4'b0101)
sll (shift left logic)	rd = rt<<1	0 (3’b000)	rs[12:10]	rt[9:7]	rd[6:4]	6 (4'b0110)
srl (shift right logic)	rd = rt>>1	0 (3’b000)	rs[12:10]	rt[9:7]	rd[6:4]	7 (4'b0111)

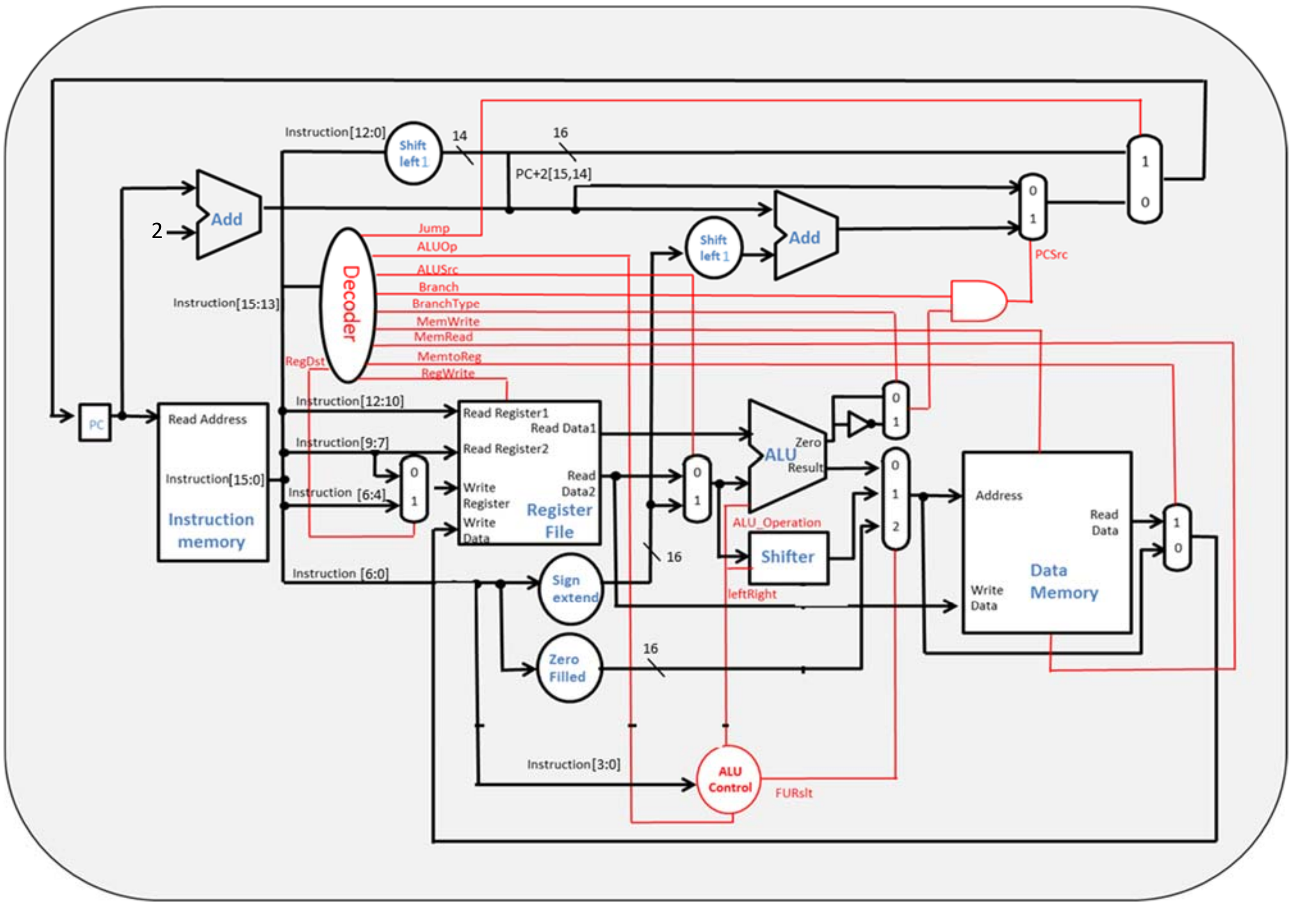
表二：I-type 與 J-type 指令定義與範例

Table 2: The definition and format of I-type and J-type instructions

指令名稱	Meaning	指令格式			
		Op field	Rs	rt	constant/address
addi	rt = rs + se_imm	1 (3'b001)	rs[12:10]	rt[9:7]	imm[6:0]
lui	rt = {0, imm, 8'b0}	2 (3'b010)	rs[12:10]	rt[9:7]	imm[6:0]
lw	rt = Mem[rs + se_imm]	3 (3'b011)	rs[12:10]	rt[9:7]	imm[6:0]
sw	Mem[rs + se_imm] = rt	4 (3'b100)	rs[12:10]	rt[9:7]	imm[6:0]
beq	If (rs==rt) then PC = PC + 2 + (se_imm << 1)	5 (3'b101)	rs[12:10]	rt[9:7]	imm[6:0]
bne	If (rs!=rt) then PC = PC + 2 + (se_imm << 1)	6 (3'b110)	rs[12:10]	rt[9:7]	imm[6:0]
jump	PC = {PC[15,14], address[12:0] << 1}	7 (3'b111)	address[12:0]		

(*se: sign extension)

(b) 架構圖 Architecture diagram



圖一：Top module (Single_Cycle_CPU) 之方塊圖

Figure 1: Block diagram of the top module Simple_Single_CPU

(a) 已提供完整內容之模組的說明 Description of the completed module

i. Data_Memory.v：data memory unit 為讀取及寫入記憶體的模組，是循序電路； data memory 的大小為 128 bytes。以下為 Data_memory.v 之輸入輸出介紹：

- clk_i：CPU 的時脈(clock)輸入，為正緣觸發(positive-edge triggered)。
- addr_i：為 16-bit 的輸入資料，作為要寫入值的目的記憶體位址。
- data_i：為 16-bit 的輸入資料，作為要寫入 Data Memory 中的內容。
- MemRead_i：為 1-bit 的輸入控制訊號，決定是否要從 Data Memory 中讀取資料。
- MemWrite_i：為 1-bit 的輸入控制訊號，決定運算結果是否要寫入 Data Memory 中。
- data_o：為 16-bit 的輸出資料，讀取相對應的記憶體所儲存的資料。

Data_Memory.v: Data memory unit is the module for reading/writing data from/into data memory. It is a sequential circuit which may store 128 bytes of data.

The input and output of Data_Memory.v are described as follows:

- clk_i: the clock input of CPU; is positive-edge triggered.
- addr_i: is a 16-bit input data used as the address, i.e., the destination memory number, of the Data Memory.
- data_i: is a 16-bit input data used as the data to be written into the Data Memory.
- MemRead_i: is a 1-bit control input signal used to decide whether to read data from Data Memory or not.
- MemWrite_i: is a 1-bit control input signal used to decide whether to write data into Data Memory or not.
- data_o: is a 16-bit output data which is the data reading from the Data Memory corresponding to the address.

ii. Instr_Memory.v：用以儲存指令，大小為 128 bytes，可根據 Program Counter 值讀取(fetch)對應的指令。

- pc_addr_i：為 16-bit 的輸入資料，作為要讀取值的目的記憶體位址。
- instr_o：為 16-bit 的輸出資料，讀取相對應的記憶體所儲存的指令。

Instr_Memory.v: Use to save instructions. Its capacity is 128 bytes. Fetch corresponding instructions by Program Counter.

- pc_addr_i：is a 16-bit input data used as the target address to read memory。
- instr_o：is a 16-bit output data which is the data reading from instruction memory.。

iii. **Program_Counter.v**：Program Counter (PC)為位址計數器，是循序電路。每道指令為 2 個 bytes，利用 PC = PC + 2 讀取下一道指令。

- clk_i：CPU 的時脈(clock)輸入，為正緣觸發(positive-edge triggered)。
- rst_n ：為 1-bit 的輸入資料，判斷 PC 值是否重置。
- pc_in_i ：為 16-bit 的輸入資料，為下一道欲執行指令的 address。
- pc_out_o：為 16-bit 的輸出資料，為目前欲執行指令的 address。

Program_Counter.v Program Counter (PC) is an address counter pointed to an instruction and is a sequential circuit. Since each instruction is 2 bytes, use "PC=PC+2" to point to the next instruction.

- clk_i：the clock input of CPU; is positive-edge triggered.
- rst_n ：is a 1-bit input data used to decide whether to reset PC.
- pc_in_i ：is a 16-bit input data which is the address of the next instruction。
- pc_out_o：is a 16-bit input data which is the address of the current instruction to be executed.

iv. **Reg_File.v**：Register File 為讀取及寫入暫存器的模組，是循序電路；共有 8 個 registers，皆為一般用途(general-purpose)，並無特定用途者(如：常數 0 的暫存器)。以下為 Reg_File.v 之輸入輸出介紹：

- clk_i：CPU 的時脈(clock)輸入，為正緣觸發(positive-edge triggered)。
- rst_n：reset signal 的控制輸入，並將 register file 初始值設定為 0 (active LOW)。
- RSaddr_i：為 3-bit 的輸入資料，作為要讀取值的第一個來源暫存器編號(圖一之 Register File 中的 Read register 1)。
- RTaddr_i：為 3-bit 的輸入資料，作為要讀取值的第二個來源暫存器編號(圖一之 Register File 中的 Read register 2)。
- RDaddr_i：為 3-bit 的輸入資料，作為要寫入值的目的暫存器編號(圖一之 Register File 中的 Write Register)。
- RDdata_i：為 16-bit 的輸入資料，作為要寫入 Register File 中之的內容(如：要寫入暫存器之運算結果)。
- RegWrite_i：為 1-bit 的輸入控制訊號，決定運算結果是否寫入 Register File。
- RSdata_o：為 16-bit 的輸出資料，第一個來源暫存器(Read register 1)裡儲存的資料。
- RTdata_o：為 16-bit 的輸出資料，第二個來源暫存器(Read register 2)裡儲存的資料。

Reg_File.v: is the module for reading and writing the registers in register file. It is a sequential circuit with 8 general-purpose registers. Assume that there is no register for special purpose (e.g., $zero).

The input and output of Reg_File.v are described as follows:

- clk_i: the clock input of CPU; is positive-edge triggered.
- rst_n: an active-LOW input signal for reset. It will initialize the registers in register file to be 0 when being activated.
- RSaddr_i: is a 3-bit input data used as the 1st source register number ("Read register 1" of the register file in Figure 1) of the register file.
- RTaddr_i i: is a 3-bit input data used as the 2nd source register number ("Read register 2" of the register file in Figure 1) of the register file.
- RDaddr_i i: is a 3-bit input data used as the destination register number ("Write register" of the register file in Figure 1) of the register file.
- RDdata_i: is a 16-bit input data used as the data to be written into the register file ("Write data" of the register file in Figure 1).
- RegWrite_i: is a 1-bit control input signal used to decide whether to write the result into Register File or not.
- RSdata_o: is a 16-bit output data which is the 1st source operands reading from the register file.
- RTdata_o: is a 16-bit output data which is the 2nd source operands reading from the register file.

(b) 學生需完成之模組說明　Description of the modules need to be completed

i. Simple_Single_CPU.v：Top module，同學須完成模組中的資料及訊號接線，請參考圖一。

Simple_Single_CPU.v: Top module, you should connect the data and control signals between components in this module. Refer to Figure 1.

ii. Decoder.v：產生 CPU 中的控制訊號，為組合電路。

以下為 Decoder.v 之輸入輸出介紹：

- instr_op_i：為 3-bit 的輸入資料，即為指令中的 Op field。
- RegWrite_o：為 1-bit 的輸出控制訊號，決定運算結果是否寫入 Register File。
- ALUOp_o：為 2-bit 的輸出控制訊號，將為 ALU_Ctrl.v 的輸入。ALUOp 請參照表三。
- ALUSrc_o：為 1-bit 的輸出控制訊號，決定 aluSrc2 的來源。
- RegDst_o：為 1-bit 的輸出控制訊號，決定 Register File 中的 Write Register 的來源。
- Branch_o：為 1-bit 的輸出控制訊號，判斷是否為 branch 指令。
- BranchType_o：為 1-bit 的輸出控制訊號，判斷為何種 branch 指令(beq 或 bne)。
- Jump_o：為 1-bit 的輸出控制訊號，判斷是否為 jump 指令。

- MemRead_o：為 1-bit 的輸出控制訊號，判斷是否讀取 Data Memory 所儲存的資料。
- MemWrite_o：為 1-bit 的輸出控制訊號，判斷是否將資料寫入 Data Memory。
- MemtoReg_o：為 1-bit 的輸出控制訊號，判斷應將讀取自 Data Memory 的資料或 Function unit 的運算結果寫入暫存器。

表三：Instruction Opcode 對應之 ALUOp

Table 3: Corresponding ALUOp for different instructions

Instruction	ALUOp
R-type	10
addi	00
lui	11
lw、sw	00
beq、bne	01
jump	X (don’t care)

Decoder.v: is a combination circuit used to generate the control signals of the CPU.The input and output of Decoder.v are described as follows:

- instr_op_i: a 3-bit input data; represents the Op field of the instruction.
- RegWrite_o: a 1-bit output control signal; is used to decide whether to write the result into Register File or not.
- ALUOp_o: a 2-bit output control signal; is used as the input of ALU_Ctrl.v. ALUOp is defined in Table 3.
- ALUSrc_o: a 1-bit output control signal; is used to decide the 2nd source operand of the ALU (aluSrc2).
- RegDst_o: a 1-bit output control signal; is used to decide the source of the destination register number of the Register File (Write Register of the Register File in Figure 1).
- Branch_o: a 1-bit output control signal; is used to decide whether the instruction is a branch or not.
- BranchType_o：a 1-bit output control signal; is used to indicate the type of the branch instruction (beq or bne).
- Jump_o: a 1-bit output control signal; is used to decide whether the instruction is a jump or not.
- MemRead_o: a 1-bit output control signal; is used to decide whether to read the data from Data Memory or not.
- MemWrite_o: a 1-bit output control signal; is used to decide whether to write the data into Data Memory or not.

- MemtoReg_o: a 1-bit output control signal; is used to decide the source (data reading from the Data Memory or the result of the function unit) of the data for the destination register in the Register File.

iii. **ALU_Ctrl.v**：產生 ALU_operation 控制訊號，為組合電路。
以下為 ALU_Ctrl.v 之輸入輸出介紹：

- funct_i：為 4-bit 的輸入資料，即為指令中的 function field。
- ALUOp_i：為 2-bit 的輸入資料，根據這個資料產生 function units (ALU&shifter)的控制訊號。
- ALU_operation_o：為 4-bit 的控制訊號線，負責控制 function units (ALU 與 shifter) (function units 之輸入控制請參考 Lab2，依序為 invertA、invertB、及 aluOp)；sll 及 srl 之控制訊號可自行定義，可利用尚未使用到的 ALU_operation 編碼，當作 Shifter.v 的控制訊號 leftRight。
- FURslt_o：為 2-bit 的輸出控制訊號；在本次 Lab 中，根據此控制訊號從 ALU、shifter、及 Zero_Filled 所產生的結果中，擇一作為寫入 register file 的 RDdata 或 data memory 的 address。

ALU_Ctrl.v: is a combination circuit used to generate the control signal of ALU_operation.
The input and output of ALU_Ctrl.v are described as follows:

- funct_i: a 4-bit input data, represents the function field of the instruction.
- ALUOp_i: a 2-bit input control signal. ALU_Ctl will generate proper control signals for the function units (ALU and Shifter) according to this input and funct_i.
- ALU_operation_o: a 4-bit output control signal used to control the function units (ALU and shifter). The control inputs of function units are defined in Lab2, which contain invertA, invertB, and aluOp. Define the control signals for sll and srl instruction by yourself. You may use the ALU_operation encoding which has not been defined in Lab2 as the control signal "leftRight" in Shifter.v.
- FUResult_o: a 2-bit output control signal. In Lab3, according to these control signals, one of the results from ALU, shifter, and Zero_Filled will be selected to be connected to RDdata of the register file or the address of the data memory.

iv. **ALU.v**：算術邏輯單元，為組合電路，必須能執行 Lab2 中要求的基本指令。可以用 RTL-level 語法來設計此單元，也可直接套用 Lab2 設計的模組(請參考加分設計)。
以下為 ALU.v 之輸入輸出介紹：

- aluSrc1：為 16-bit 的輸入資料，作為運算的第一個運算子。

- aluSrc2：為 16-bit 的輸入資料，作為運算的第二個運算子。
- ALU_operation：為 4-bit 的輸入控制訊號線；輸入控制請參考 Lab2。(依序為 invertA、invertB 及 aluOp)
- result：為 16-bit 的輸出資料，ALU 的運算結果。
- zero：為 1-bit 的輸出資料，判斷運算結果是否為 0。
- overflow：為 1-bit 的輸出訊號；當 ALU 中的加減法運算結果產生溢位時為 1，反之為 0 。(若沿用 Lab2 所設計的 ALU.v，會有這條線；如果是另用 RTL-level 來設計此 ALU，則可以忽略這項輸出)

ALU.v: Arithmetic Logic Unit, a combination circuit. ALU has to execute the basic operations in Lab2. You can design this module in RTL-level, or use the previous module designed in Lab2 (described in Bonus).
The input and output of ALU.v are described as follows:
- aluSrc1: a 16-bit input data, which represents the 1st operand of the ALU.
- aluSrc2: a 16-bit input data, which represents the 2nd operand of the ALU.
- ALU_operation: is a 4-bit input control signal used to indicate the proper operation of the ALU. The control inputs include invertA, invertB, and aluOp, and are defined in Lab2.
- result: a 16-bit output data, which represents the computation result of the ALU
- zero: a 1-bit output control signal. It is set to 1 when the computing result of ALU, “result” , is 0; otherwise, is clear to 0.
- overflow: a 1-bit output control signal. It is set to 1 when the computing result of ALU is overflow; otherwise, is clear to 0. (There is an overflow signal if you use your ALU.v designed in Lab2. You can ignore this output if you use RTL-level Verilog code to design the ALU.)

v. **Shifter.v**：移位器，為組合電路，必須能處理表一中所定義的 sll 與 srl 位移指令，亦即邏輯左或右移一位。
以下為 Shifter.v 之輸入輸出介紹：
- sftSrc：為 16-bit 的輸入資料，作為被移動的資料。
- leftRight：為 1-bit 的輸入控制訊號；其值為 1 時，執行左移，反之則右移。
- result：為 16-bit 的輸出資料，位移器的運算結果。

Shifter.v：shifter, a combination circuit. This shifter has to be able to execute the sll and srl instructions defined in Table 1, i.e., logically shift left/right one bit position.
The input and output of Shifter.v are described as follows:
- sftSrc: a 16-bit input data, is the source data of the shifter.

- leftRight: a 1-bit input control signal. When it is set to 1, the shifter perform logical left shift; else, does logical right shift.
- result: a 16-bit output data, which represents the shifting result of the shifter.

vi. **Sign_Extend.v**：將立即值以有號的方式從 7 bits 擴展為 16 bits，為組合電路。以 4-bit 有號擴展到 8-bit 為例：1100 -> 11111100。

Sign_Extend.v: is a combination circuit used to extend the immediate value from 7-bit to 16-bit with sign bit (e.g., 4-bit to 8-bit, 1100 -> 11111100).

vii. **Zero_Filled.v**：將立即值前方補一個 0，後方補八個 0，從 7 bits 擴展為 16 bits，為組合電路。例如：1100110 -> 0110011000000000。

Zero_Filled.v: is a combination circuit used to concatenate one zero in front of the immediate value and eight zeros behind the immediate value to extend the value from 7 bits to 16 bits (e.g. 1100110 -> 0110011000000000).

此外，也請完成 Adder.v、Mux2to1.v、Mux3to1.v 等模組。如果有需要，同學可自行增加模組來完成 CPU 的設計。但請注意，實作時不要改動到上述給定的模組以及 top module，以免助教測試時發生錯誤。
Besides, you should also accomplish “Adder.v”, “Mux2to1.v”, and “Mux3to1.v” modules. You can add any module needed to accomplish your CPU design. DO NOT modify the provided modules and the top module in order to avoid the testing errors.

C. 加分部分設計 Bonus (15%)

請確認 CPU 基本要求部分(B.)之電路正確無誤後，再作加分題(C.)。請務必保留 B 部分之完整資料(.v files)，以便繳交。如完成進階部分加分設計，可得到額外的分數。
Please confirm that the circuit designed for the basic CPU requirements (B.) is correct before you take the advanced part (C.), and please do preserve the complete .v files of Part B for submission. If you complete the advanced part of the bonus design, you can get additional points.

(a) Lab2 ALU 延續使用 Applying ALU designed in Lab2 (5%)

若是能將 Lab2 設計好的 ALU 模組延續使用到此次 Lab，可以獲得額外的分數。
Apply the ALU module designed in Lab2 by you as the ALU in this single-cycle CPU.

(b) 實作進階 Shift 指令 Implementing Advanced Shift Instructions (10%)

將 sllv 以及 srlv 指令加入原基本 single-cycle CPU 設計，指令定義於下表；可修改或新增其他模組與控制訊號來完成這部份的設計。

Add sllv and srlv instructions into the design of the single-cycle CPU. These two instructions are defined in the following table. You may modify or add modules or control signals for this advanced design.

Instruction	Example	Meaning	Op field	Function field
SLLV	sllv rd,rt,rs	rd = rt<<rs	0	8 (4'b1000)
SRLV	srlv rd,rt,rs	rd = rt>>rs	0	9 (4'b1001)

3. 測試方式 Test Method

作業繳交後，助教會將 TestBench.v 編譯出的模組(TestBench)與 Simple_Single_CPU.v 編譯出的模組連結，來進行模擬測試。模擬中，會讀入不同的測試數據以驗證同學們程式的正確性。

After you hand in your code, TA will use the same TestBench module and different test data to verify the correctness of your design of Simple_Single_CPU.

同學如果有新增其他的檔案，請確保其編譯出的模組不會影響 TestBench 對 Simple_Single_CPU 的訊號輸入與結果顯示。

If you have attached additional modules for your design, do ensure that these modules would not affect our testing.

本次 lab 提供 3 筆基本測試資料(binary code)，分別存於 CO_Lab3_testdata1.txt ~ CO_Lab3_testdata3.txt;預設的測資為第一筆，如果想測試第二筆資料，請將檔案“ TestBench.v”中第 147 行指令改成

```
$readmemb("CO_Lab3_testdata2.txt", cpu.IM.Instr_Mem);
```

第三筆測試資料的修改方式也類似，但改用 CO_Lab3_testdata3.txt。此外，同學也可以自己設計測試資料去確認 Simple_Single_CPU 的正確性。

In Lab3, three basic test data (binary code), stored in “CO_Lab3_testdata1.txt” ~ “CO_Lab3_testdata3.txt”, are provided. The default test data is the first one. If you would like to use second test data, modify line 147 of “ TestBench.v” as follows:

```
$readmemb("CO_Lab3_testdata2.txt", cpu.IM.Instr_Mem);
```

Use the similar way to check the third test data “CO_Lab3_testdata3.txt”. You may design other test data by yourself to verify the correctness of your design.

以下為測試資料的組合語言指令片段：

The Assembly codes of the test data are given as follows:

testdata1	testdata2	testdata3 (加分題用) (for bonus)
addi r1,r0,10 addi r2,r0,17 addi r3,r0,-2 add r4,r3,r2 sub r5,r2,r1 and r6,r1,r2 or r7,r1,r2 nor r0,r6,r7	addi r0,r2,3 addi r1,r0,7 addi r6,r0,6 addi r7,r0,12 sw r6,0(r7) slt r2,r0,r1 beq r2,r3,2 lw r5,0(r7) jump L1 nor r5,r5,r5 L1: addi r4,r4,-2 sub r3,r3,r4	addi r0,r0,6 addi r1,r0,2 addi r2,r0,-3 nor r7,r4,r5 sll r3,r2 sllv r4,r7,r2 srlv r5,r7,r1 srl r6,r0

執行 TestBench 後，會產生 "CO_Lab3_result.txt"，同學可將結果與提供的答案進行比對以驗證是否正確。
After the simulation of Testbench, Verilog will generate the file "CO_Lab3_result.txt". You can verify the result by comparing to provided ans.txt file.

4. 作業報告 Report

請回答以下問題：

Q1: 請簡略說明你所新增的檔案內容以及其功能。倘若你沒有新增檔案，則請簡略說明你的 Simple_Single_CPU.v 中設計了哪些電路。
Describe the additional module designed by you, if any, and its circuit briefly. If you have not attached additional file for CPU design, describe your Simple_Single_CPU.v.

Q2: 根據你設計的電路模組，請解釋在 Simple_Single_CPU 中，如何決定 arithmetic、logic、shift、load、store、branch 或 jump 指令。此外，請列出各個指令在 Decoder 的控制訊號輸出值，格式如下表所示。
According to the circuit modules designed by you, explain how to determine the type of instruction being executed as arithmetic, logic, shift, load, store, branch or jump in Simple_Single_CPU module. Moreover, list the control signals generated by the decoder for each instruction as the following table.

	Inputs	Outputs									
Instruction	instr_op (3 bits)	Branch	Branch Type	Jump	ALUSrc	ALUOp (2 bits)	MemWrite	MemRead	MemtoReg	RegDst	RegWrite
R-type	000										
addi	001										
…	…										

Q3: 根據你設計的 **ALU_Ctrl.v** 電路模組，列出其輸入與輸出的關係，如下表所示。
According to the **ALU_Ctrl.v** circuit module designed by you, list the relationship between its inputs and outputs as the following table.

	Inputs		Outputs	
Instruction	ALUOp (2 bits)	funct (4 bits)	ALU_operation (4 bits)	FUResult (2 bits)

Q4: 根據你設計的電路模組，請解釋在 Simple_Single_CPU 中，如何決定下一個 PC 的數值，並且說明各 PC 值是從何處計算出來。
According to the circuit modules designed by you, explain how to determine next PC, and how to calculate next PC.

Q5: 請寫出你在 Lab3 中所碰到的困難，以及實作完成後的心得。
Write down your comment for Lab3. If you have any difficulty while implementing the function units in this lab, express it also in your report.

Q6: (若有實作加分題，請回答此題) 請分別說明為了實作加分題 C(a) 與 C(b) 修改了原基本 single-cycle CPU 設計中的那些模組、或新增了甚麼模組。
(Answer Q6 if you have implemented the Bonus Part) Please explain respectively which modules in the original basic single-cycle CPU design have been modified, or which modules have been added, in order to implement bonus part C(a) and C(b).

Q7: (若有實作 Barrel shift instructions 加分部分，請回答此題) 請簡單說明你是作何修改來完成這個部分的設計(如：增加的模組或是控制訊號)。列出 ALU_Ctrl.v 中，針對 sllv 與 srlv 這兩道指令，其輸入與輸出的關係，如下表所示。並請畫出修改後的架構方塊圖。
(Answer Q7 if you have implemented the advanced shift instructions for bonus) Describe the advanced shifter (for examples, new modules and control signals) designed by you, list the relationship between the input and output of the ALU_Ctrl.v for these two instructions (sllv, srlv), and draw the block diagram of the modified circuit.

	Inputs		Outputs	
Instruction	ALUOp (2 bits)	funct (4 bits)	ALU_operation (4 bits)	FUResult (2 bits)
sllv				
srlv				

5. 注意事項 Notice

a. 請用 Vivado 2020.3(推薦)或 iverilog 做為開發環境。(硬體描述語言模擬器之安裝與設定請參閱 Lab2-0-Vivado_iverilog_gtkwave 說明文件)
Develop your lab in Vivado 2020.3 (recommended) or iverilog. (Refer to Lab2-0-Vivado_iverilog_gtkwave for the simulator of hardware description language, if necessary.)

b. 請務必使用附件提供的模組來完成作業，如用自己新增的模組，請確保「TestBench.v」能正確模擬。
Use modules provided to implement your single cycle CPU. If you have designed any extra module, please ensure that TestBench.v can simulate correctly.

c. **除非有特別說明，否則請勿更改**任何現有的程式碼 (TestBench 中的測資檔名除外)。若造成助教無法測試，後果請自行負責。
Do not modify any existing code except the input filename in TestBench.v.

d. 請將你的程式碼寫在 module 中標示「/*your code here*/」的位置。
Write your code in “/*your code here*/” in each module.

e. 這次的 Lab 中，除了 Simple_Single_CPU.v 外，其餘所有要求學生完成的模組請確實採用**組合電路**之設計(**禁止採用循序邏輯**，違者 0 分計)。
Do not design the incomplete modules left for students, except Simple_Single_CPU.v, as sequential circuit!! Zero score will be given for sequential model.

f. 本實驗單元各模組可用 behavioral-level 的 Verilog 語法來寫，不限制只能用 gate-level 的方式來實作。
In Lab3, you can design your modules with behavioral-level in Verilog as well as gate-level.

6. 評分 Grading

程式正確性與 Demo (80%)：助教會以多筆測資測試你的程式，並檢查你的程式是否符合作業要求。若有答錯或未符合作業要求，則視程式完成度斟酌給分。助教在 demo 時會要求你解釋你自己的程式碼，還有解釋你的結果是否正確。未上傳報告與程式者，不予 Demo。有上傳報告與程式但未 Demo 者，不予評分。

報告(20%)：回答作業報告中所列的 Q1~Q5 的問題。請依照自己的想法回答，回答只需正確或合乎邏輯即可得到分數。

加分(15%)：完成進階電路，並回答作業報告中的 Q6~Q7。根據電路與問題回答情形，斟酌給分。未 Demo 者，不予評分。

Program & Demo (80%): We will test many test cases for grading. We will request you to explain how you complete your code and your result. If you didn't upload your code and report, you cannot demo your work. If you have uploaded your code but doesn't join the Demo, you will get a zero point.

Report (20%): You should answer the four questions (Q1~Q5) mentioned at least, and you will get the points if your answer is correct or reasonable.

Bonus (15%): If you have implemented the advanced designed for bonus, please answer questions (Q6~Q7) in your report. You will get bonus if your circuit design and question answer are correct.

7. 作業繳交 Deadline

a. 本實驗單元為一人一組，請將實驗報告及相關 Verilog 檔案上傳至 e3 平台。
This lab unit is one student per group. Please upload your Lab Report (word file) and the corresponding HDL code (.v files) onto e-Campus platform.

- **實驗報告**：pdf 檔，請命名為 **Lab3_學號_姓名**，如：「Lab3_110551600_王大明」。
Hand in the lab report as a word file, named **Lab3_StudentID_Name**, for example, "Lab3_110551600_Kent Chang".
- **Verilog modules 檔案**：14 .v files at least
Adder.v、ALU.v、ALU_Ctrl.v、Decoder.v、Data_Memory.v、Instr_Memory.v、Mux2to1.v、Mux3to1.v、Program_Counter.v、Reg_File.v、Shifter.v、Sign_Extend.v、Zero_Filled.v、Simple_Single_CPU.v
- 請將上述實驗報告 pdf 檔及 Verilog 電路模組與測試模組檔案(.v)全部壓縮成一個 zip 檔 (禁止上傳 rar 檔或是其他檔案格式)，並以「Lab3_學號_姓名」

的方式命名，如：「Lab3_110551600_王大明」。若有實作加分題部分，則請將基本 Single-Cycle CPU 設計的.v 檔案放在壓縮檔中 basic 資料夾中，加分題部分的.v 檔案(須包含原基本設計的功能)的檔案放在壓縮檔中 bonus 資料夾中，並請確保兩個資料夾中的程式可以分別執行；壓縮檔結構可參考附圖。

Please compress the pdf file of lab report, the Verilog circuit modules and testbench described above all into one zip file (rar file or other format is not accepted), and name the zip file as "Lab3_StudentID_Name", for example, "Lab3_110551600_KentChang". If you have implemented the bonus part, please store the .v files for the basic single-cycle CPU design under the "basic" folder and the .v files for the bonus (which should also include the functions of the basic part) under the "bonus" folder in the zip file. Make sure that the modules in the two folders can be executed separately.

```
─Lab3_110551600_王大明.zip
    │  Lab3_110551600_王大明.pdf
    ├─basic
    │    └─ 14(or more) files for basic CPU
    └─bonus
         └─ 14(or more) files for bonus CPU
```

b. 繳交日期為 2022/8/1 (一) 23:59，準時繳交者可獲得 10%的額外加分；最後繳交期限為 2022/8/15 (一) 23:59，之後不接受逾期繳交。
The due date is 2022/8/1 (Monday) 23:59 and you may get extra 10% points for on-time award. Deadline is 2022/8/15 (Monday) 23:59, and no late hand-in is allowed.

c. Demo 日期：

- 2022/8/1 (一) 23:59 前繳交的 demo 日期暫定為 2022/8/3 (三) 至 2022/8/4 (四)；如期 demo 者，方可得 10% on-time award。
  For those who hand in this lab unit before 2022/8/1 (Monday) 23:59, the demo dates are arranged on 2022/8/3 (Wednesday) and 2022/8/4 (Thursday) tentatively.
- 2022/8/15 (一) 23:59 前繳交的 demo 日期暫定為 2022/8/17 (三)。
  For those who hand in this lab unit after the due date but before 2022/8/15 (Monday) 23:59, the demo date is arranged on 2022/8/17 (Wednesday) tentatively.
- 時間表會在之後公告，請至線上填寫時間。如有需要在其他時間 demo，請盡快與助教連絡討論。
  The time table for demo will be announced later. Please fill the time table on line. If you have any question about demo time, please contact TA ASAP.

d. 禁止抄襲，違者(抄襲者與被抄襲者)以 0 分計算。
Any assignment work by fraud will get a zero point.