Computer Organization Project

Lab4 – Pipelined Processor

1. 目標 (Goal)

在這次 Lab 中，我們希望同學們可以了解 pipeline 處理器的原理並實作之。請根據 Lab3 所設計的 single-cycle CPU，將其改成五個 stages 的 pipeline 處理器，並應仍可正確執行之前實作過的指令。

The main goal of this lab unit is to make students understand the principle of pipelined processor and implement it. Please modify the single cycle processor designed in Lab3 by you to a 5-stage pipelined processor and execute the instruction defined correctly.

2. 實驗說明 (Lab Description)

A. 附加檔案 (Attached files)

附檔為一個負緣觸發的 Reg_File.v、測試用的 TestBench.v 以及四組測資 txt 檔。請設計四個 pipeline stage registers 的 Verilog modules，修改你在 Lab3 所設計的 Single-cycle CPU (Simple_Single_CPU.v) 中的模組及連線，建構出 5-stage pipelined processor (Pipeline_CPU.v)。請注意：各 pipeline stage register 中，應包含儲存資料及控制訊號的欄位(fields)。

Attached files are a “negtive-edge triggered” Reg_File.v, TestBench.v, and four txt files for testing. Please design four modules for the pipeline stage registers, rewire the Single-cycle CPU **(**Simple_Single_CPU.v**),** and modify or add new modules to construct a 5-stage pipelined processor (Pipeline_CPU.v). Note that for each pipeline register, it should contain the fields for data and control signals.

B. 實作指令 (Implementation of instructions)

這次 Lab 將測試之前實作過的指令包含： add、sub、and、or、nor、slt、sll、srl、addi、lui、lw、sw、beq、bne 及 jump，各個指令的定義請參考 Lab3 之說明文件。

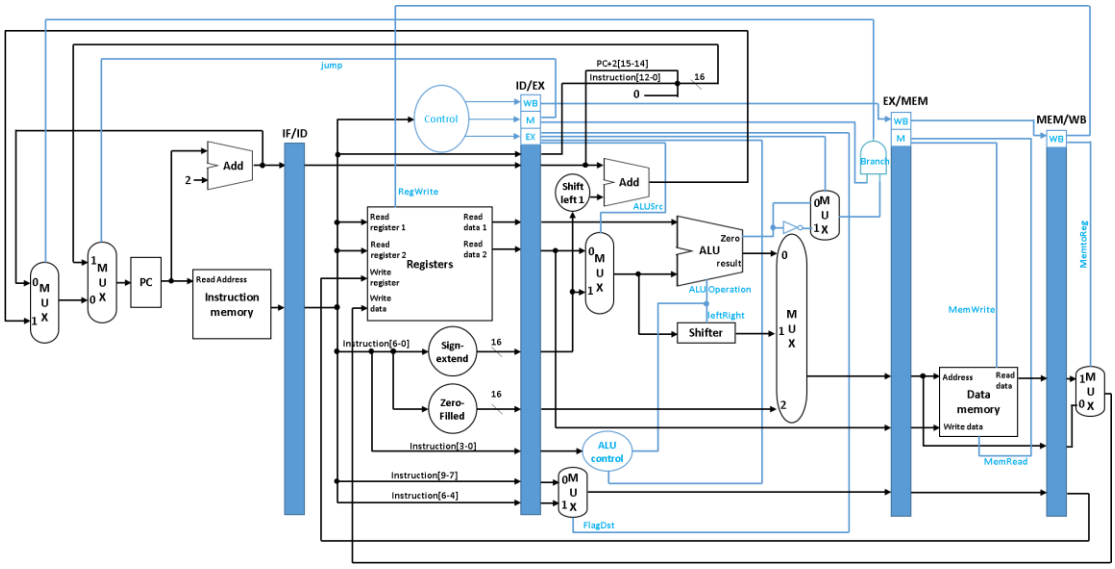
In this Lab, we will test the instructions defined and implemented in previous Lab3 : add, sub, and, or, nor, slt, sll, srl, addi, lui, lw, sw, beq, bne, and jump.

C. 基本 Pipelined CPU 設計 Basic Design of a Pipelined CPU (100%)

a. 電路圖 (the circuit diagram)：

本次 Lab 要實作之 pipelined processor 的方塊圖如下圖所示:

The block diagram of the pipelined processor to be implemented in this lab is given below:



如圖所示，這次 lab 所實作的 pipelined processor 有五個 Stages，分別為 **IF stage**、**ID stage**、**EX stage**、**MEM stage** 以及 **WB stage**。而每一個 stage 之間都需安插一個 pipeline register；每一個 pipeline register 所存的內容須包含此指令要傳遞到下一個 stage 的資料及控制訊號。請設定各 pipeline register 在時脈(clk_i)上升(positive-edge)時寫入。

According to the above diagram, in this lab you should implement a five-stage pipelined processor with **IF**, **ID**, **EX**, **MEM**, and **WB** stages. You should insert a pipeline register between two adjacent stages. Each pipeline register should contain the fields for data and control signals. The pipeline registers are written when the positive edge of clock (clk_i) occurs.

b. Pipeline stage 介紹 (Introduction of pipeline stages)：

以下介紹各 stage 所要執行的動作：

The function of each stage is described as follows:

IF stage：在這個 stage 中，處理器必須將指令從 instruction memory 中取出並且執行 PC+2 的動作。

IF stage: In this stage, the processor fetches an instruction from the instruction memory and performs PC + 2.

ID stage：在這個 stage 中，處理器必須依照 opcode 欄位產生出相對應的控

制訊號、讀出兩個來源暫存器的值、及產生出擴充(sign-extended 與 zero-filled)後的立即值。

ID stage: In this stage, the processor decodes the instruction to generate the control signals, reads two source registers, and generates the sign-extended and zero-filled immediate values.

EX stage：在這個 stage 中，處理器會將 ID stage 所傳過來的控制訊號(ALUOp)送至 ALU_Ctrl 中，以產生控制 ALU 與 Shifter 的控制訊號，執行正確運算。此外，在這一 stage 中也會選擇 Write Register (RDaddr_i)，以及算出 branch target 與 jump target 的值。

EX stage: In this stage, ALU_Ctrl generates control signals for function units according to ALUOp. At the same time, Write Register (RDaddr_i), branch target and condition, and jump target are also determined in this stage.

MEM stage：在這個 stage 中，處理器將會根據控制訊號對資料記憶體作存取。

MEM stage: In this stage, the processor accesses data memory according to the control signals.

WB stage：在這個 stage 中，處理器將會根據控制訊號決定是否要在時脈下降時對暫存器作寫入，以及寫入的 data 是從 Data Memory 讀出的資料或是 ALU 運算的結果值。

WB stage: In this stage, the processor will determine whether to write the value into register file when negative clock edge occurs or not and which of the data, data reading from data memory or calculating by ALU, should be written into register file according to the control signals.

c. Pipeline Register 說明 (Description of pipeline registers)：

請設計四個 pipeline stage registers 的 Verilog modules，其中每個 pipeline register 都必須是正緣觸發(clk_i)的循序電路，且初始值皆為 0。其重置訊號(rst_n)為 active Low，因此 pipeline register 在 rst_n 為 0 時會將內容設為 0，而在 rst_n 為 1 時才會執行寫入的動作。在設計完成後，請將四個 pipeline registers 安插到你在 Lab3 中所設計的 single-cycle CPU 中，並重新連線，以完成本實驗要求的 pipelined CPU。

Please design four pipeline registers. Each pipeline register must be “positive-edge triggered”, and has default value 0 initially. The reset control signal, rst_n, of a pipeline register is active LOW. Therefore, a pipeline register can only be written when “rst_n” is set to 1. Insert these pipeline registers into your single-cycle CPU designed in Lab3 and reconnect the modules to

accomplish the pipelined CPU required in this lab.

請注意，在你所設計的循序電路中，不可以自行加上延遲時間。如果造成測試時產生錯誤或是誤差，則請自行負責。
DO NOT set any delay time for the sequential circuits of the pipelined registers designed by you.

d. Register File (Reg_File.v)補充說明 (Supplement of Register File, Reg_File.v)：

為了避免在 WB stage 時寫入的新值無法被正在 ID stage 的指令所取得，Register file 的寫入時間改為在時脈下降(negative-edge)時；亦即在時脈下降時在 register file 中寫入值，以便可以在後半個 cycle 中讀出。以下列指令為例：

```
i1   add $s1, $s2, $s3   ; in WB stage
i2   …                   ; in MEM stage
i3   …                   ; in EX stage
i4   add $s5, $s1, $s4   ; in ID stage
```

在這一個 cycle 的時脈下降時我們先將 WB 中 i1 的結果存入 register $s1 中(此時 register 中的值是最新的)，i4 則可在後半個 cycle 將$s1 的值讀出；由於 pipeline registers 的更新是在時脈上升時，因此在下一個時脈上升時，可將 i4 讀出的$s1 資料存到 ID/EX pipeline register 中。

To achieve that the new register value (written in WB stage) can be read in ID stage in the same clock cycle, the Register File should be designed as negative-edge triggered, i.e., the register file should be written when negtive clock edge occurs. It means that the register file may be written value in the first half cycle and be read value from it file in the second half cycle. For example:

```
i1   add $s1, $s2, $s3   ; in WB stage
i2   …                   ; in MEM stage
i3   …                   ; in EX stage
i4   add $s5, $s1, $s4   ; in ID stage
```

The instruction in WB stage (i1) should write value into register file at the negtive clock edge. As a result, i4 will be able to read the correct register value in the second half of the cycle and store the data into the ID/EX pipeline register when the positive-edge of the next clock cycle occurs.

e. Data Hazard 處理機制 (Data Hazard Handling)：

當先行指令(Predecessor)之目標暫存器(rd 或 rt)與後行指令(Succeessor)的來源暫存器(rs 或 rt)之一有 true data dependency，且兩道指令間隔太近時，則會發生 data hazard。若在 ID stage 偵測出 ID stage 的指令與 EX stage 或 MEM stage 中的指令有相依性時，則需插入一個 stall；意即將下個 clock edge 來時要存入 ID/EX pipeline register 的控制訊號設為零，並保持 PC 與 IF/ID pipeline register 不變。

When the destination register of a predecessor instruction is the same as one of the source registers of its succeesor, data hazard may occurr. If the data hazard detection circuit in ID stage detect that the instuction in ID stage has data dependancy with the instuction in EX stage or in MEM stage, insert a stall into pipeline, i.e., set control signals for ID/EX pipeline register to 0s, preserve the PC and IF/ID pipeline register when the next positive-edge of clock occurs.

f. Control Hazard 處理機制 (Control Hazard Handling)：

在我們的 pipelined processor 設計中，beq、bne、及 jump 指令改變 PC 值的行為是在 EX stage 執行。也就是說，一旦 EX stage 發生 branch taken 或是 jump 指令，則必須清除(flush) branch 或 jump 後面的兩道指令，亦即目前在 IF 與 ID stages 中的指令；換言之，在 taken branch 或 jump 指令後會有兩個 stalls (bubbles)。之後，再由正確的指令位址，branch or jump target，開始 fetch 指令。

In our pipelined processor design, *beq*, *bne*, and *jump* instructions may change the PC value, i.e., change the control flow, in EX stage. Once a branch instruction is taken or a jump instruction is performed, you should flush the IF/ID and ID/EX pipeline registers, i.e., inserts two bubbles, and then fetch the correct instruction from the branch or jump target.

D. 加分部分之設計 Bonus (30%)

請確認 pipelined CPU 基本要求部分(C.)之電路正確無誤後，再作加分題(D.)。請務必保留 C 部分之完整資料(.v files)，以便繳交。如完成進階部分加分設計，則可得到額外的分數。

Please confirm that the circuit designed for the basic pipelined CPU requirements (C.) is correct before you take the advanced part (D.), and please do preserve the complete .v files of Part C for submission.

本次實驗中可藉由 data forwarding 方式處理的指令相依性類型如下：
(表中之 R-type 表示 add、sub、and、or、nor、sll、srl、slt 這八道指令，FU source1 表示 EX stage 中 ALU 的第一個 operand，FU source2 則表示 EX stage 中 ALU 的第二個 operand 或是 shifter 的 operand，MEM data 則是要傳到 MEM stage 作為寫入 Data Memory 的資料。)

The types of true data depeandencies which may be resolved by data forwarding in this lab are given below:
(*R-type* represents the set of *add*, *sub*, *and*, *or*, *nor*, *sll*, *srl*, and *slt* instructions, *FU source1* represents the 1st source operand of ALU, *FU source2* represents the 2nd source operand of ALU or the source operand of shifter in EX stage, and *MEM data* represents the data to be written to Data Memory in MEM stage.)

表一： 可靠 Forwarding 方式處理的 Data Hazard 基本類型
Table 1: Types of data hazards resolving by forwarding

Forwarding Type		Predecessor	Succeessor
MEM ↓ EX	FU source1 (rs)	R-type (rd) addi、lui (rt)	R-type (excluding sll, srl)、 addi、lw、sw、beq、bne
	FU source2 (rt)	R-type (rd) addi、lui (rt)	R-type、beq、bne
	MEM data (rt)	R-type (rd) addi、lui (rt)	sw
WB ↓ EX	FU source1 (rs)	R-type (rd) addi、lui、lw (rt)	R-type (excluding sll, srl)、 addi、lw、sw、beq、bne
	FU source2 (rt)	R-type (rd) addi、lui、lw (rt)	R-type、beq、bne
	MEM data (rt)	R-type (rd) addi、lui、lw (rt)	sw
WB ↓ MEM	MEM data (rt)	lw (rt)	sw

如上表所示，請同學在實作 data forwarding unit 時考慮以上指令之間的相依性並且將資料傳到正確的位置。針對 lw 指令，則需設計 load-use hazard detection unit，以便在 load 與 use 之間加入一個 stall (bubble)，而後再利用 data forwarding 將資料傳到正確的位置。

Please design two Data Forwarding Unit, one in EX stage and another in MEM stage, to resolving the data hazards according to the table given above, , i.e.,

forward data from a predecessor to its successor with true data dependency. As for *lw* instruction, you should design a Load-Use Hazard Detection Unit to insert a bubble between *lw* and the use instruction (except *sw* of which the rt depends on the rt of *lw*) followed right after.

(a) Load-Use Hazard Detection Unit (in ID stage)： (10%)

在處理 load-use 類型的 data hazard 時，因為 *lw* 指令(指令 *i*)要到 MEM stage 方能得到要存入 destination register 的值，因此與 *lw* 指令有 true data dependency 的第 *i+1* 道指令(除了 *sw* 指令之 rt 外)，必須先 stall 一個 cycle，而後再利用 forwarding 於 EX stage 時取得正確的值。因此，同學們必須設計一個 Hazard detection unit 來處理這種狀況，並且在之前設計的 pipelined processor 模組中作一些修改，使得 CPU 可以正確表現出 stall 一個 cycle 的行為。請將 **Load-Use Hazard Detection Unit** 置於 ID stage 中。

For resolving load-use data hazard, since the *lw* instruction may obtain its loading data until MEM stage, you should stall the "use" instruction (except the sw which has true data dependecy of its rt with the *lw* instuction) for a cycle and then apply data forwarding to get the correct value in EX stage. Therefore, modify the pipelined processor and design a **Load-Use Hazard Detection Unit** in ID stage to detect the load-use hazard and stall for a cycle.

(b) Data Forwarding Unit (in EX stage)：(15%)

Data forwarding unit 是負責決定在 EX stage 中之 function units (ALU 與 Shifter)的來源、及之後要傳到 MEM stage 作為寫入 Data Memory 的資料 (sw 之 rt 值)是否要用 forwarding 的值還是原本由暫存器檔案讀取的值。其 Forwarding 種類包含上表中之 MEM → EX 與 WB → EX 兩部份，請同學們參考上課時所學到的判斷式來設計本次實驗的 forwarding unit。請將 **Data Forwarding Unit** 置於 EX stage 中。

Data forwarding unit is used to determine whether the source operands of the function units (ALU or Shifter) in EX stage and the data (rt of *sw*) to be written to Data Memory in MEM stage are the original values obtained from register file or the forwarded data. The forwarding types designed in this unit should contain MEM → EX and WB → EX listed in the table given above. Please attach this data forwarding unit in EX stage.

(c) Load-Store Forwarding Unit (in MEM stage)：(5%)

在“D. (a)”所設計的 Load-Use Hazard Detection Unit 中，有 load-use 關係的指令，除 rt 與其有 true data depencency 之 *sw* 指令外，在 ID stage 偵測出後，要先 stall 一個 cycle 再作 forwarding 的方式處理。但若是 *sw* 指令要存入 Data Memory 的值(rt)與 *lw* 的 destination 值(rt)有相依關係，則應在 MEM stage 中增加 forwarding 電路，將 WB stage 中 *lw* 的 destination 值傳給 MEM stage 中 Data Memory 的 write data，而無需 stall。請同學們針對這個部分(上表中 WB → MEM 的部份)，在 MEM stage 中加入 Load-store forwarding unit。

For the Load-Use Detection Unit designed in “D. (a)”, the use instruction (except the sw which has true data dependecy of its rt with the *lw* instuction) will be stall for one cycle when load-use hazard is detected in ID stage. However, if the load-use dependency is between an *lw* instruction and the write data (rt) of an *sw* instruction, i.e., the destination register (rt) of *lw* is the same as the register of the write data (rt) of *sw*, then you sould forward the data read by *lw* to the write data of *sw* in MEM stage without any stall. Please attach the “Load-Store Forwarding Unit” in MEM stage.

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 Pipeline_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 Pipeline_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 Pipeline_CPU 的訊號輸入(clk_i 和 rst_n)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 4 筆測試資料(binary code)，分別存於 CO_Lab4_testdata1.txt~ CO_Lab4_testdata4.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 35 行指令改成

```
$readmemb("CO_Lab4_testdata2.txt", cpu.IM.Instr_Mem);
```

第三、四筆測試資料的修改方式也相同，但用“CO_Lab4_testdata3.txt”、“CO_Lab4_testdata4.txt”；同學可以自己設計測試資料去確認 Pipeline_CPU 的正確性。
In Lab4, four test data (binary code), stored in “CO_Lab4_testdata1.txt”~

“CO_P4_testdata4.txt”, are provided. The default test data is the first one. If you would like to use second test data, modify line 35 of “ TestBench.v” as follows:

```
$readmemb("CO_Lab4_testdata2.txt", cpu.IM.Instr_Mem);
```

Use the similar way to check the third test data “CO_Lab4_testdata3.txt”, and “CO_Lab4_testdata4.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:

CO_Lab4_testdata1.txt	CO_Lab4_testdata2.txt	CO_Lab4_testdata3.txt (加分題用)	CO_Lab4_testdata4.txt (加分題用)

addi r1,r0,31
addi r2,r0,-2
add r3,r1,r2
nor r4,r5,r6
and r5,r1,r0
slt r6,r2,r1
or r7,r1,r2
	addi r2,r0,6
	addi r3,r0,8
	sub r5,r3,r2
	L1: addi r5,r5,1
	addi r4,r4,3
	beq r4,r2,1
	jump L1
	sw r3,0(r0)
	sw r5,0(r5)
		addi r2,r0,5
		sw r2,0(r0)
		nor r4,r3,r2
		and r1,r4,r3
		lw r5,0(r0)
		add r7,r5,r1
			addi r2,r0,3
			sw r2,0(r0)
			slt r4,r3,r2
			or r1,r4,r3
			lw r5,0(r0)
			sw r5,2(r0)

執行 TestBench 後，同學可透過 display 的結果驗證是否正確。
After the simulation of TestBench, You can verify the result by the display.

4.作業報告 (Report)

Q1：請寫出在你的電路中，各 pipeline register 中所設計的欄位。
Write down the fields in each pipeline register designed by you.

Q2: (a) 請說明為了將 CPU 設計成 pipelined 形式，對原 single-cycle CPU 中那些模組做了甚麼修改、或新增了甚麼功能的模組。
Describe what modifications have been made to those modules in the original single-cycle CPU, or what function modules have been added in order to make the CPU become pipelined.

(b) 請說明為了以 stall 方式處理 data hazard 而新增或修改了哪些電路。
Describe what circuits have been added or modified by you to solve data hazard by stalling.

(c) 請說明為了以 stall 方式處理 control hazard 而新增或修改了哪些電路？

Describe what circuits have been added or modified by you to solve control hazard by stalling.

Q3：請拍下你在“CO_Lab4_testdata2.txt ”這個測資中的 Data Hazard Detection Unit 以及 Control Hazard Detection Unit 的波形圖，貼入報告中，並說明其波形(Hazard detect)是否正確。

Take a snapshot for the waveform diagrams of the Data Hazard Detection Unit and Control Hazard Detection Unit for “CO_Lab4_testdata2.txt ” and attach the diagrams into the report. Describe the correctness of your hazard detect waveform.

Q4：請寫出你在 Lab4 中所碰到的困難，以及實作完成後的心得。

Write down your comment for Lab4. If you have any difficulty while implementing this lab, express it also in your report (Please write in detail).

Q5: (若有實作加分題，請回答此題) 請分別說明為了實作加分題 D(a)、D(b)、及 D(c) 修改了原基本 pipelined CPU 設計中的那些模組、或新增了甚麼模組。

(Answer Q5 if you have implemented the Bonus Part) Please explain respectively which modules in the basic pipelined CPU design have been modified, or which modules have been added, in order to implement bonus part D(a), D(b), and D(c).

Q6 : (若有實作加分題，請回答此題) 請寫出在你的 Data Forwarding Unit 電路中，決定是否 forwarding 的判斷式。(e.g., if ID/EX.rs == EX/MEM.rd then …)

(Answer Q6 if you have implemented the Bonus Part) Write down the condition equations defined in you Data Forwarding Unit for determine whether forwarding should be performed or not. (e.g., if ID/EX.rs == EX/MEM.rd, then …)

Q7 ：(若有實作加分題 D(a)或 D(b)，請回答此題) 請拍下你的設計在執行“CO_Lab4_testdata3.txt”這個測資的 Load-Use Hazard Detection Unit，貼入報告中，並說明其波形(Hazard detect)是否正確。

(Answer Q7 if you have implemented D(a) or D(b) of the Bonus Part) Take a snapshot for the waveform diagrams of the Load-Use Hazard Detection Unit

for “CO_Lab4_testdata3.txt ” and attach the diagrams into the report. Describe the correctness of your hazard detect waveform.

Q8: (若有實作加分題 D(c)，請回答此題)請拍下你的設計在執行“CO_Lab4_testdata4.txt”這個測資的 Load-Store Forwarding Unit，貼入報告中，並說明其波形(Forwarding)是否正確。
(Answer Q8 you have implemented D(c) of the Bonus Part) Take a snapshot for the waveform diagrams of the Load-Store Forwarding Unit for “CO_Lab4_testdata4.txt ” and attach the diagrams into the report. Describe the correctness of your forwarding waveform.

5.注意事項 (Notes)

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。
If you have designed any extra module, please ensure that TestBench.v can simulate correctly.

c. 除了 TestBench 中的測資檔名外，若更動到任何現有的模組程式碼(如 Program_counter.v)，請在報告中解釋。若造成助教無法測試，後果請自行負責。
If you have modified any existing code except the input filename in TestBench.v, please explain in your report.

d. 這次的 Lab 中，四個 pipeline registers 必須使用循序邏輯來設計。
In this Lab, ensure to develop the four pipeline registers by sequential circuit.

e. 本次 Lab 中，如果發現實作完成的並非 pipeline 處理器，則本次 lab 測試部分以 0 分計算
Zero score will be given for non-pipelined CPU.

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 或 dataflow-level 的方式來實作。

In this Lab, you can design your modules with behavioral-level in Verilog as well as gate-level and dataflow-level.

g. 請重新命名你的 Top module 為「Pipeline_CPU」。(原本為「Simple_Single_CPU」)
Please rename your top module into “Pipeline_CPU”.

6.評分 (Score)

程式與 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~Q4 四個問題。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(30%)：在完成進階電路後，請回答作業報告中的 Q5~Q8 之相關問題。根據電路與問題回答情形，斟酌給分。

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 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~Q4) mentioned at least, and you will get the points if your answer is correct or reasonable.

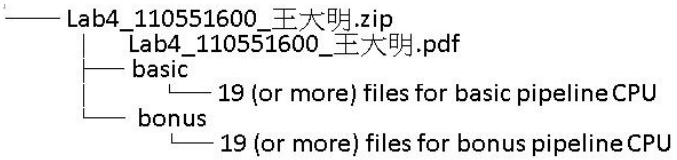
Bonus (30%): If you have implemented the advanced circuit for bonus, please answer questions Q5 to Q8 in your report. You will get bonus if your circuit design and question answer are correct.

7.作業繳交 (Hand in)

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 (pdf

file) and the corresponding HDL code (.v files) onto e-Campus platform.

- **實驗報告**：pdf 檔，請命名為 **Lab4_學號_姓名**，如：「Lab4_110551600_王大明」。
  Hand in the lab report as a pdf file, named **Lab4_StudentID_Name**, for example, "Lab4_110551600_Kent Chang".
- **Verilog modules 檔案**：
  (For basic pipelined CPU design) Pipeline_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、Pipe_IFID.v、Pipe_IDEX.v、Pipe_EXMEM.v、Pipe_MEMWB.v、HazardDetector.v、…
  (For bonus) Load_Use_HazardDetector.v、EX_ForwardingUnit.v、MEM_ForwardingUnit.v、…
- **若僅實作基本 pipelined CPU 部分**，請將實驗報告 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「Lab4_學號_姓名」的方式命名，如：「Lab4_110551600_王大明」。**若有實作加分題部分**，則請將基本 pipelined 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。
  If you have implemented the basic pipelined CPU part only, please compress the pdf file of lab report, the Verilog circuit modules and testbench 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 pipelined 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.



b. 繳交日期為 2022/8/8 (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 2022/8/8 (Monday) 23:59and 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：

- 2022/8/8 (一) 23:59 前繳交的 demo 日期暫定為 2022/8/10 (三)、2022/8/11 (四) ；如期 demo 者，方可得 10% on-time award。
  For those who hand in the homework before 2022/8/8 (Monday) 23:59, the demo dates are arranged on 2022/8/10 (Wednesday) and 2022/8/11 (Thursday) tentatively.
- 2022/8/15 (一) 23:59 前繳交的 demo 日期暫定為 2022/817 (三)。
  For those who hand in the homework after the due date but before 2022/8/15 (Monday) 23:59, the demo date is arranged on 2017/8/17 (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，請寄信與助教連絡討論。
  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 email 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.