**微算機系統**

期末專案

組別： 16

班級、姓名與學號：

資工二 陸詠涵 108590451

資工二 陳佳均 108830035

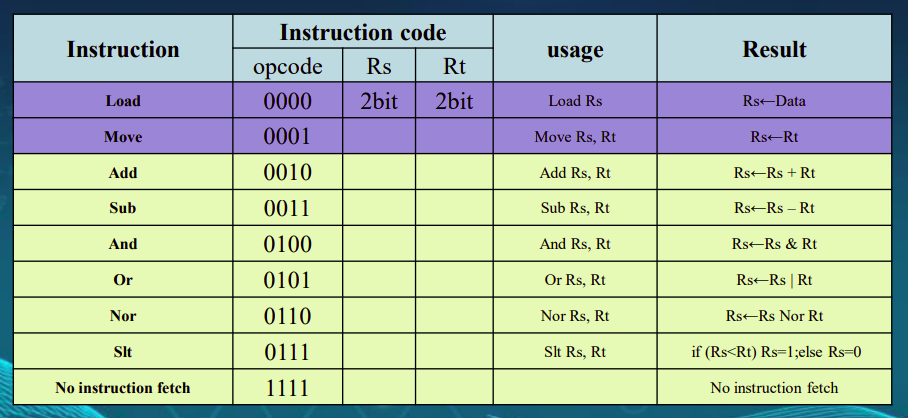
日期：2021.01.14

1. 實驗內容：

利用上課所學，設計出一個pipeline功能的簡易CPU。總實驗有四暫存器可用來儲存DATA,利用Rs和Rt來決定要存在哪個暫存器中，然後分別讓DATA,Rs和Rt接上七段顯示器，顯示在燒錄板子上。利用綠色LED來表示程式已執行到的步驟，再利用紅色LED來確保Hazard正在執行=>檢查Hazard是否發生。

1. 實驗過程及結果：

我們要透過簡易CPU實現的功能有Load、Move、Add、Sub、And、Or、Nor、Slt



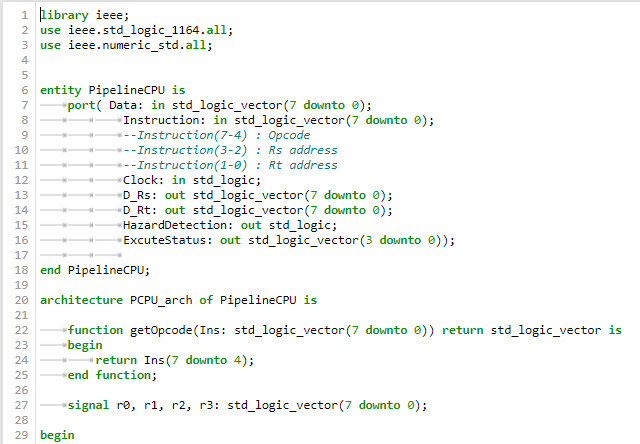
其中Add、Sub、And、Or、Nor、Slt需要測試是否有用到Hazard

<key>Hazard是一種用來偵測連續執行指令很重要的部分，將第一次執行的結果馬上write back然後再繼續執行指令。

這次pipeline的設計比較上一次的CPU多了四個variable，其功能是為了記錄每一階段所產生的結果。而且，為了解決Hazard的問題，我們將整個程式放入process中，並且利用倒敘的方式將variable依序往後放(EX:variable3=>variable4 再來是 variable2=>variable3)，如此便能不搞混順序。

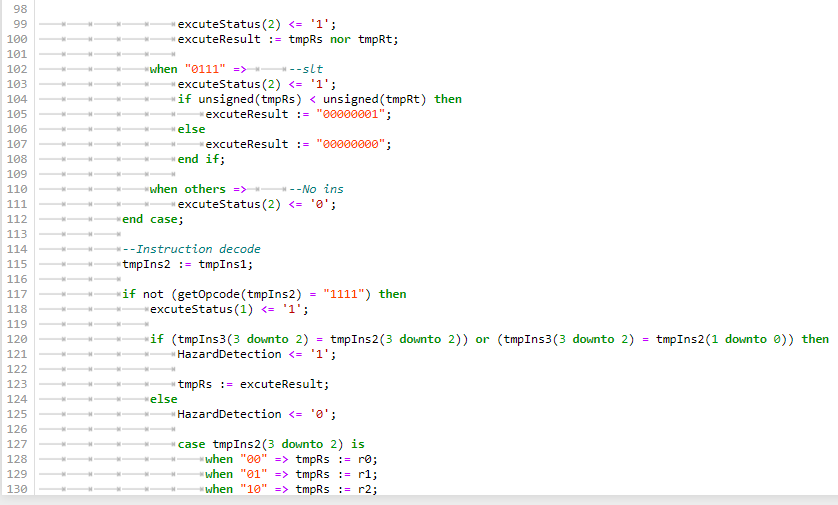
Step1設計程式碼

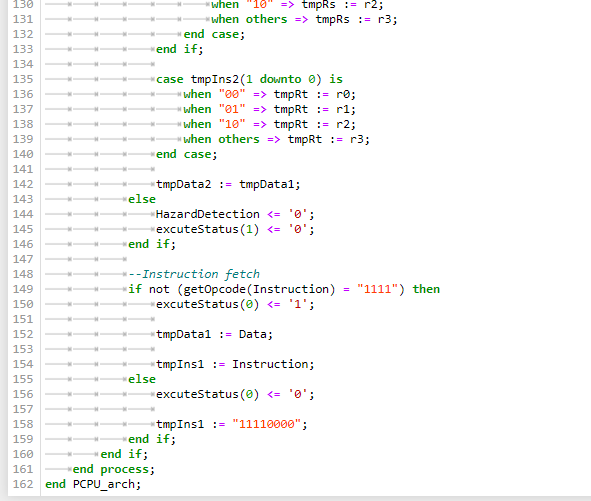
(pipeline\_CPU)



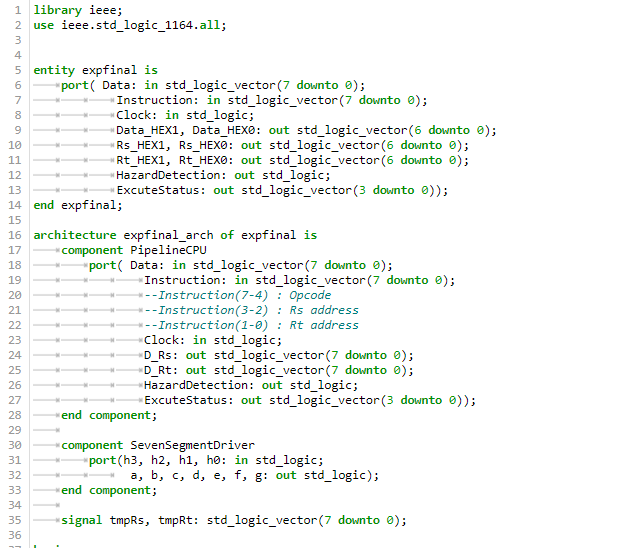


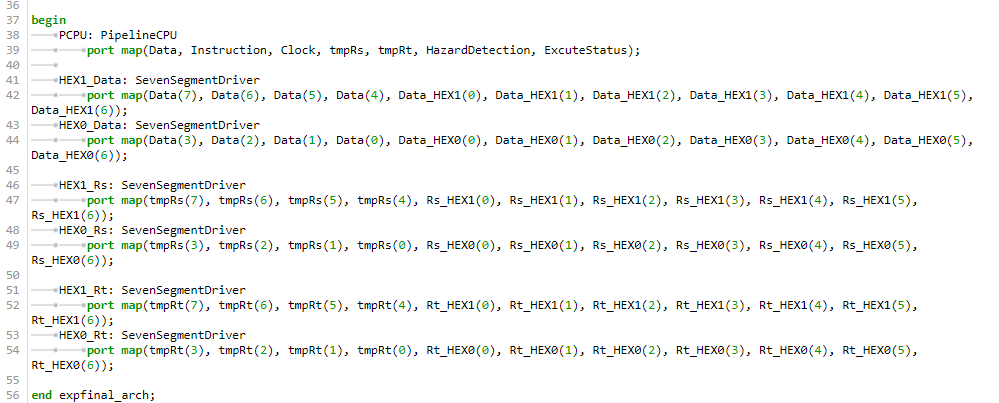




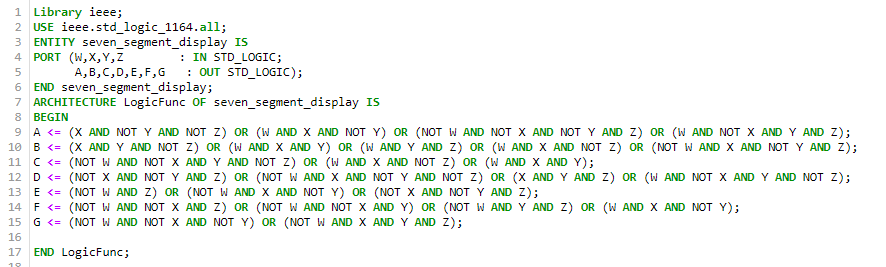


(主程式)

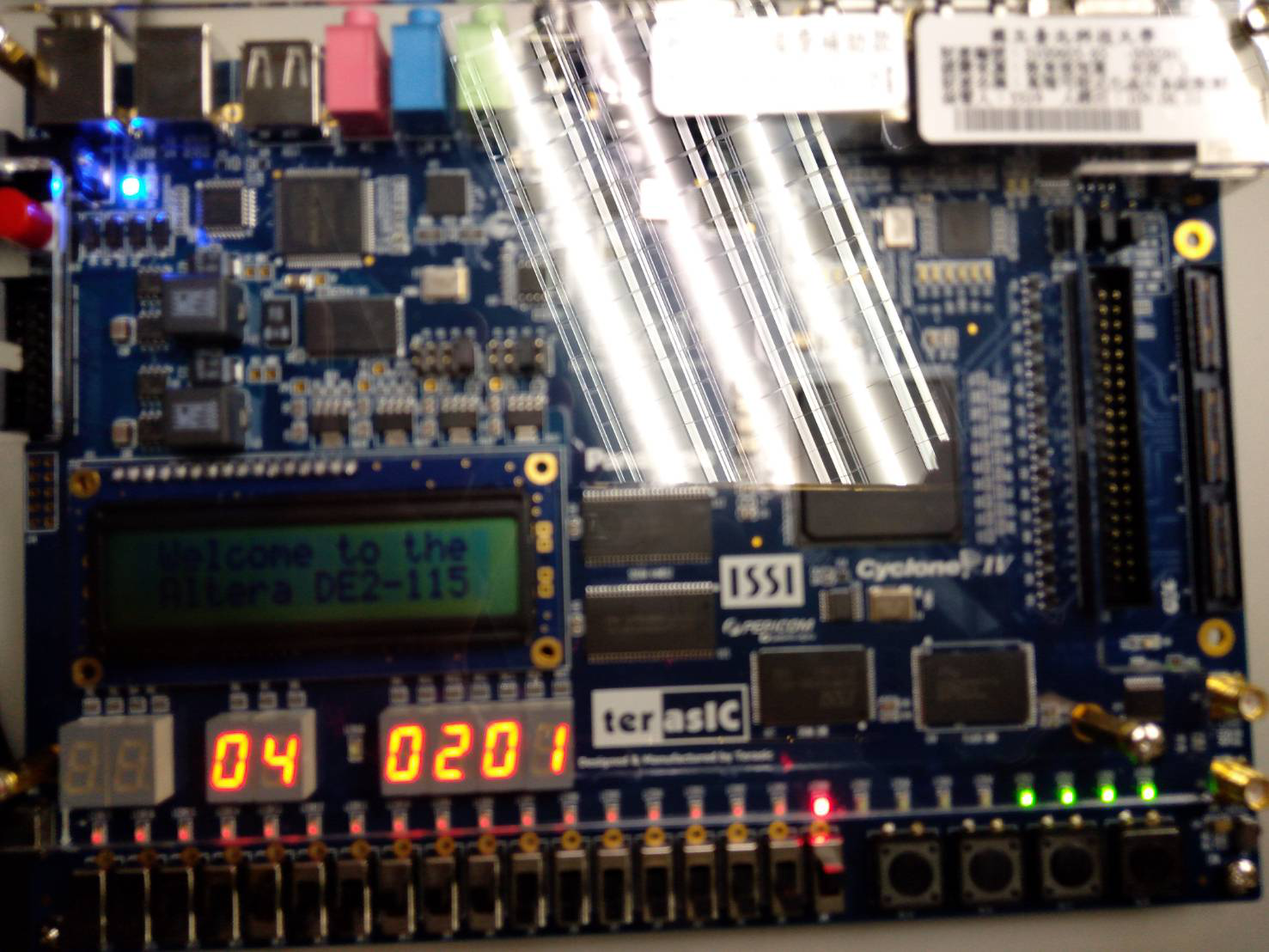




(seven segment)



Step2燒錄結果



1. 程式碼

|  |
| --- |
| 實驗 |
| library ieee;  use ieee.std\_logic\_1164.all;  use ieee.numeric\_std.all;  entity PipelineCPU is  port( Data: in std\_logic\_vector(7 downto 0);  Instruction: in std\_logic\_vector(7 downto 0);  --Instruction(7-4) : Opcode  --Instruction(3-2) : Rs address  --Instruction(1-0) : Rt address  Clock: in std\_logic;  D\_Rs: out std\_logic\_vector(7 downto 0);  D\_Rt: out std\_logic\_vector(7 downto 0);  HazardDetection: out std\_logic;  ExcuteStatus: out std\_logic\_vector(3 downto 0));    end PipelineCPU;  architecture PCPU\_arch of PipelineCPU is  function getOpcode(Ins: std\_logic\_vector(7 downto 0)) return std\_logic\_vector is  begin  return Ins(7 downto 4);  end function;  signal r0, r1, r2, r3: std\_logic\_vector(7 downto 0);  begin    D\_Rs <= r0 when Instruction(3 downto 2) = "00" else  r1 when Instruction(3 downto 2) = "01" else  r2 when Instruction(3 downto 2) = "10" else  r3;  D\_Rt <= r0 when Instruction(1 downto 0) = "00" else  r1 when Instruction(1 downto 0) = "01" else  r2 when Instruction(1 downto 0) = "10" else  r3;    process(Clock)  variable tmpIns1: std\_logic\_vector(7 downto 0) := "11110000"; --Instruction fetch  variable tmpIns2: std\_logic\_vector(7 downto 0) := "11110000"; --Instruction decode  variable tmpIns3: std\_logic\_vector(7 downto 0) := "11110000"; --Excution  variable tmpIns4: std\_logic\_vector(7 downto 0) := "11110000"; --Write back    variable excuteResult: std\_logic\_vector(7 downto 0);    variable tmpRs, tmpRt, tmpData1, tmpData2: std\_logic\_vector(7 downto 0);    begin  if rising\_edge(Clock) then  --Write back  tmpIns4 := tmpIns3;    if not (getOpcode(tmpIns4) = "1111") then  ExcuteStatus(3) <= '1';    case tmpIns4(3 downto 2) is --Rs address  when "00" => r0 <= excuteResult;  when "01" => r1 <= excuteResult;  when "10" => r2 <= excuteResult;  when others => r3 <= excuteResult;  end case;  else  ExcuteStatus(3) <= '0';  end if;    --Excution  tmpIns3 := tmpIns2;      case getOpcode(tmpIns3) is  when "0000" => --Load  excuteStatus(2) <= '1';  excuteResult := tmpData2;    when "0001" => --Move  excuteStatus(2) <= '1';  excuteResult := tmpRt;    when "0010" => --Add  excuteStatus(2) <= '1';  excuteResult := std\_logic\_vector(signed(tmpRs) + signed(tmpRt));    when "0011" => --Sub  excuteStatus(2) <= '1';  excuteResult := std\_logic\_vector(signed(tmpRs) - signed(tmpRt));    when "0100" => --And  excuteStatus(2) <= '1';  excuteResult := tmpRs and tmpRt;    when "0101" => --Or  excuteStatus(2) <= '1';  excuteResult := tmpRs or tmpRt;    when "0110" => --Nor  excuteStatus(2) <= '1';  excuteResult := tmpRs nor tmpRt;    when "0111" => --slt  excuteStatus(2) <= '1';  if unsigned(tmpRs) < unsigned(tmpRt) then  excuteResult := "00000001";  else  excuteResult := "00000000";  end if;    when others => --No ins  excuteStatus(2) <= '0';  end case;    --Instruction decode  tmpIns2 := tmpIns1;    if not (getOpcode(tmpIns2) = "1111") then  excuteStatus(1) <= '1';    if (tmpIns3(3 downto 2) = tmpIns2(3 downto 2)) or (tmpIns3(3 downto 2) = tmpIns2(1 downto 0)) then  HazardDetection <= '1';    tmpRs := excuteResult;  else  HazardDetection <= '0';    case tmpIns2(3 downto 2) is  when "00" => tmpRs := r0;  when "01" => tmpRs := r1;  when "10" => tmpRs := r2;  when others => tmpRs := r3;  end case;  end if;    case tmpIns2(1 downto 0) is  when "00" => tmpRt := r0;  when "01" => tmpRt := r1;  when "10" => tmpRt := r2;  when others => tmpRt := r3;  end case;    tmpData2 := tmpData1;  else  HazardDetection <= '0';  excuteStatus(1) <= '0';  end if;    --Instruction fetch  if not (getOpcode(Instruction) = "1111") then  excuteStatus(0) <= '1';    tmpData1 := Data;    tmpIns1 := Instruction;  else  excuteStatus(0) <= '0';    tmpIns1 := "11110000";  end if;  end if;  end process;  end PCPU\_arch; |

1. 實驗心得：

**陸詠涵：**

本次實驗，因為和實驗七的執行步驟很像，所以我們利用實驗七的程式來改。我們的想法是，將4個步驟各自設立不同的process，再利用rising\_edge clock來達成每按一次，執行到下一個步驟的結果。但是在設計的過程中，我們遇到了在不同的process重複使用到相同的變數，所以在compile的過程中，當然不會通過。然後還有Hazard的部分

，他需要先將上一個指令write back然後再繼續執行下一個指令。所以我們回去討論之後，發現可以將所以指令放在同一個process裡，這樣不但更快速，且不會因為時間差而有錯誤。最後，我們粗心的將腳位接錯了，所以debug了一陣子。謝謝老師和助教這學期的用心指導，隨然每次過程都卡卡的，但是期末的收穫真的很多。

**陳佳均：**

這次的期末實驗本來想用VHDL程式，也就是同時運行的思維來寫的。但是，當我們在設變數的時候卻發現有些變數的功能好像重複了(沒錯，就是設了一些沒意義的變數)，有些功能卻還需要其他變數，搞得我頭實在很痛。於是我們最後毅然決然地放棄了寫了一個禮拜的程式，重新在寫了一遍process的版本，也就是類似於python這種有順序的程式的思維方式。

再來就是最後在檢查的時候，真的沒想到會檢查那麼久，對不起助教了，不過還是好好笑，哈哈。因為我們怕沒檢查通過要留時間改進，所以我們程式一跑過就教助教了，完全沒發現我們的opcode跟rs,rt指標順序錯了，超糗的。

**最後，感謝助教每一堂課的幫忙，謝謝!!**