

Tiny MIPS core in Verilog

CO Lab3



Department of Electrical Engineering and SoC Research Center
National Chung Cheng University

Outline

- 實驗目的
- 實驗環境
- 實驗介紹
- 範例教學
- 課堂練習
- 作業說明
- 參考資料
- 附錄

實驗目的

在之前課程已經學習如何使用Verilog實作一個乘法器，
本次課程將運用前一次課程所學，使用Verilog實做RISC
Processor中的MIPS CPU，並瞭解各指令在RISC運作方式。

實驗環境

在本實驗中同學將使用Icarus Verilog的 iverilog、vvp 兩個指令，進行編譯及模擬，並透過Gtkwave觀察波形。

規劃各級硬體

- RISC架構下的指令，Datapath可拆解成5 stage完成，並於pipeline中執行
- 每個stage完成的動作，可視為一組micro-operation，各有其對應的micro-architecture

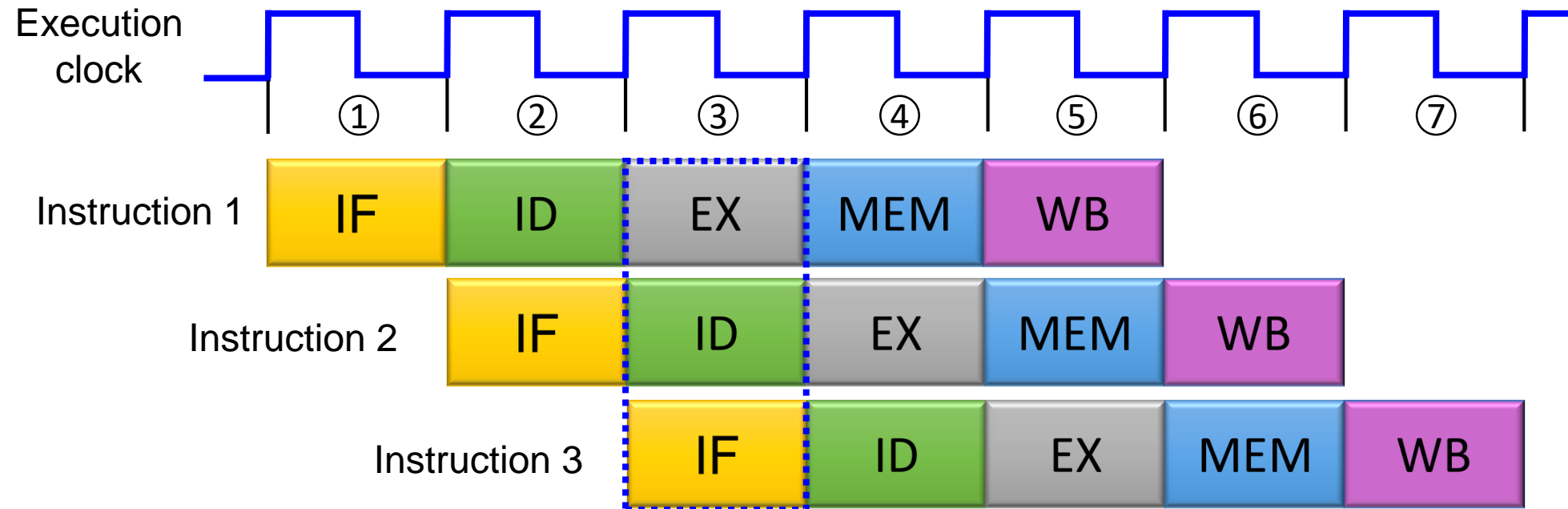


- 5 stage : IF(instruction fetch) 、 ID(instruction decode) 、 EX(execution) 、 MEM(memory access) 、 WB(write back)

RISC Processor in Pipeline Design

■ What is Pipeline

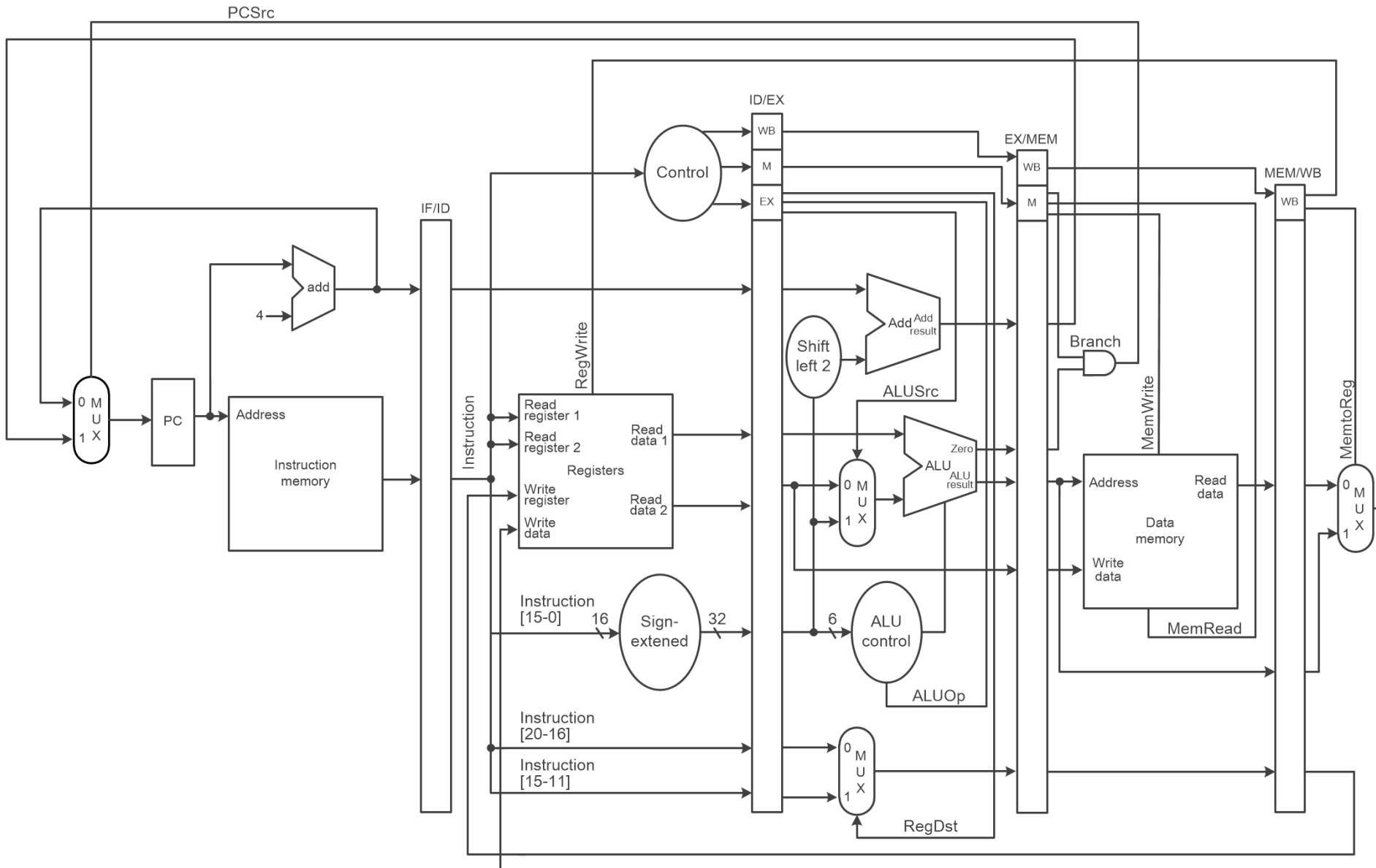
- 將每道指令切成多個stage
- 在同個clock cycle，讓多道指令於不同stage中執行



RISC Processor in Pipeline Design

- 設計「管線間暫存器」，保存指令於不同stage執行之值。
- 基本RISC pipeline架構下，管線間暫存器有四個：
 1. IF/ID
 2. ID/EX
 3. EX/MEM
 4. MEM/WB
- 各stage之I/O相關性：管線執行過程中，會將訊號於管線暫存器中逐級傳送，故上一級之output，通常為下一級的input。
- 定義好各stage之input及output，即完成初步pipeline硬體架構規劃。

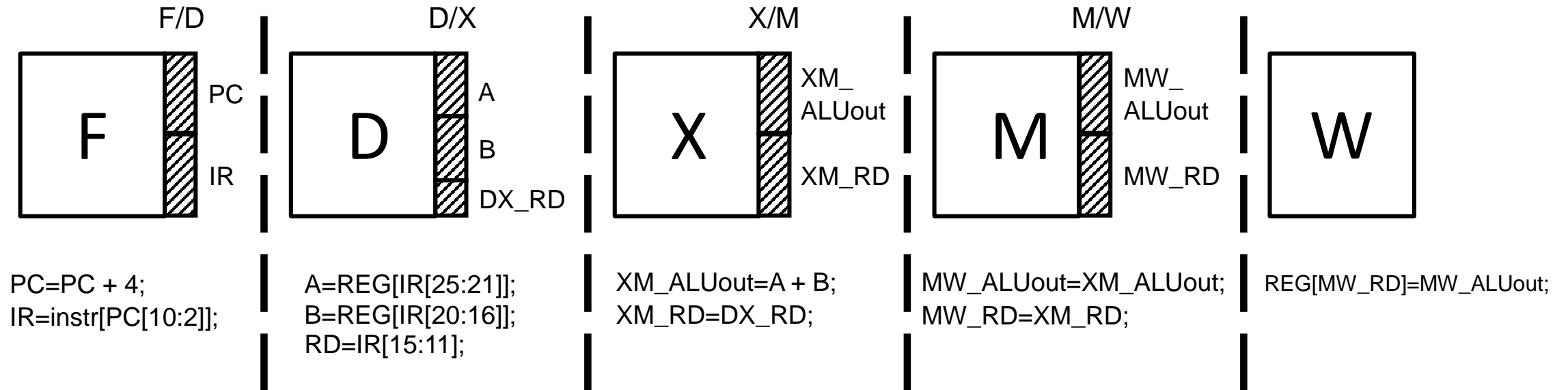
Pipeline Structure



Pipeline Design (EX. add)

■ add rd,rs,rt
 $\text{reg(rd)} = \text{reg(rs)} + \text{reg(rt)}$

Description	Adds two registers and stores the result in a register
Operation	$\$d = \$s + \$t$; advance_pc (4);
Syntax	add \$d, \$s, \$t
Encoding	0000 00ss ssst tttt dddd d000 0010 0000



Modularization (EX. add)

F/D

```
module INSTRUCTION_FETCH(
    clk,
    rst,

    PC,
    IR
);
input clk, rst;
output reg [31:0] PC, IR;
// instructions
reg [31:0] instr [127:0];
always @(posedge clk)
begin
    if(rst)begin
        PC <= 32'd0;
        IR <= 32'd0;
    end else begin
        PC <= PC+4;
        IR <= instr[PC[10:2]];
    end
end
endmodule
```

D/X

```
module INSTRUCTION_DECODE(
    clk,
    rst,
    PC,
    IR,
    MW_RD,
    MW_ALUout,

    A, B, RD
);
input clk, rst;
input [31:0] IR, PC, MW_ALUout;
input [4:0] MW_RD;

output reg [31:0] A, B;
output reg [4:0] RD;

// register files
reg [31:0] REG [0:31];

always @(posedge clk)
    REG[MW_RD] <= MW_ALUout;

always @(posedge clk)
begin
    A <= REG[IR[25:21]];
    B <= REG[IR[20:16]];
    RD <= IR[15:11];
end
endmodule
```

X/M

```
module EXECUTION(
    clk,
    rst,
    A,
    B,
    DX_RD,

    ALUout,
    XM_RD,

);
input clk, rst;
input [31:0] A, B;
input [4:0] DX_RD;

output reg [31:0] ALUout;
output reg [4:0] XM_RD;

always @(posedge clk)
begin
    ALUout <= A + B;
    XM_RD <= DX_RD;
end
endmodule
```

M/W

```
module MEMORY(
    clk,
    rst,
    ALUout,
    XM_RD,

    MW_ALUout,
    MW_RD
);
input clk, rst;
input [31:0] ALUout;
input [4:0] XM_RD;

output reg [31:0] MW_ALUout;
output reg [4:0] MW_RD;

// data memory
reg [31:0] Mem [0:127];

always @(posedge clk)
begin
    MW_ALUout <= ALUout;
    MW_RD <= XM_RD;
end
endmodule
```

(WB=ID)

CPU.v

```
`timescale 1ns/1ps

`include "INSTRUCTION_FETCH.v"
`include "INSTRUCTION_DECODE.v"
`include "EXECUTION.v"
`include "MEMORY.v"

module CPU(
    clk,
    rst
);
input clk, rst;
/*===== Wire =====*/
// INSTRUCTION_FETCH wires
wire [31:0] FD_PC, FD_IR;
// INSTRUCTION_DECODE wires
wire [31:0] A, B;
wire [4:0] DX_RD;
wire [2:0] ALUctr;
// EXECUTION wires
wire [31:0] XM_ALUout;
wire [4:0] XM_RD;
// DATA_MEMORY wires
wire [31:0] MW_ALUout;
wire [4:0] MW_RD;

/*===== INSTRUCTION_FETCH =====*/
INSTRUCTION_FETCH IF(
    .clk(clk),
    .rst(rst),

    .PC(FD_PC),
    .IR(FD_IR)
);
```

宣告各Stage之前傳值所需要的連接線

接續

```
/*===== INSTRUCTION_DECODE =====*/
INSTRUCTION_DECODE ID(
    .clk(clk),
    .rst(rst),
    .PC(FD_PC),
    .IR(FD_IR),
    .MW_RD(MW_RD),
    .MW_ALUout(MW_ALUout),

    .A(A),
    .B(B),
    .RD(DX_RD),
    .ALUctr(ALUctr)
);

/*===== EXECUTION =====*/
EXECUTION EXE(
    .clk(clk),
    .rst(rst),
    .A(A),
    .B(B),
    .DX_RD(DX_RD),
    .ALUctr(ALUctr),

    .ALUout(XM_ALUout),
    .XM_RD(XM_RD)
);

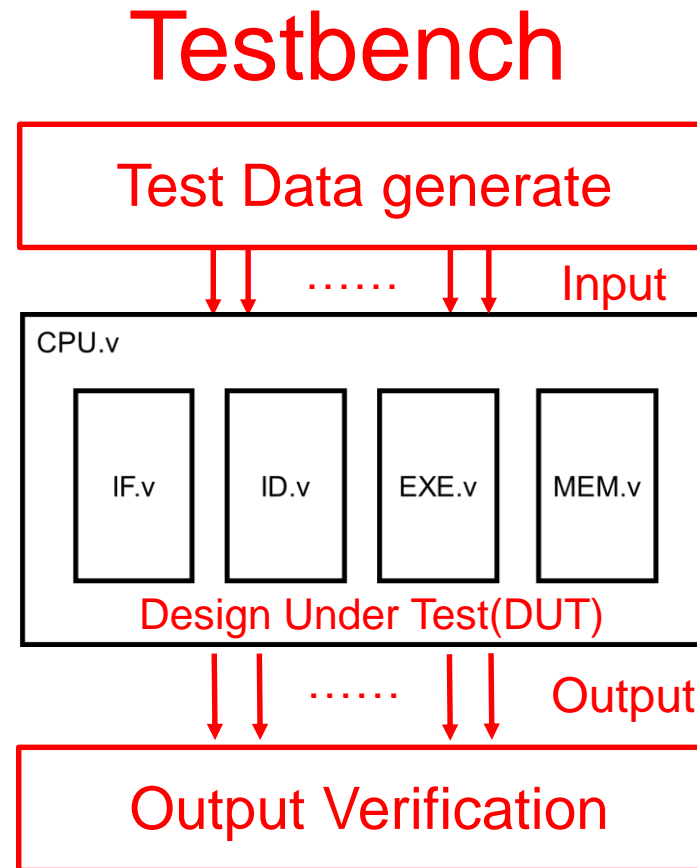
/*===== DATA_MEMORY =====*/
MEMORY MEM(
    .clk(clk),
    .rst(rst),
    .ALUout(XM_ALUout),
    .XM_RD(XM_RD),

    .MW_ALUout(MW_ALUout),
    .MW_RD(MW_RD)
);

endmodule
```

RISC Processor RTL Simulation

利用 Structural modeling技巧進行Verilog模擬，將需要驗證之設計(Design under verification, DUV)包在top-module下，並以high-level語法產生測試pattern及觀察結果



Testbench.v

```
`define CYCLE_TIME 20
`define INSTRUCTION_NUMBERS 20
`timescale 1ns/1ps
`include "CPU.v"

module testbench;
  reg Clk, Rst;
  reg [31:0] cycles, i;

  // Instruction DM initialilation
  initial
  begin
    /*===== 連加 =====*/
    cpu.IF.instruction[ 0] = 32'b000000 00001 00010 00011 00000 100000; //add $3, $1, $2      $3 = 1 + 2 = 3
    cpu.IF.instruction[ 1] = 32'b000000_00000_00000_00000_00000_100000; //NOP(add $0, $0, $0)
    cpu.IF.instruction[ 2] = 32'b000000_00000_00000_00000_00000_100000; //NOP(add $0, $0, $0)
    cpu.IF.instruction[ 3] = 32'b000000_00000_00000_00000_00000_100000; //NOP(add $0, $0, $0)
    cpu.IF.PC = 0;

  end

  // Data Memory & Register Files initialilation
  initial
  begin
    cpu.MEM.DM[0] = 32'd9;
    cpu.MEM.DM[1] = 32'd3;
    for (i=2; i<128; i=i+1) cpu.MEM.DM[i] = 32'b0;

    cpu.ID.REG[0] = 32'd0;
    cpu.ID.REG[1] = 32'd1;
    cpu.ID.REG[2] = 32'd2;

    for (i=3; i<32; i=i+1) cpu.ID.REG[i] = 32'b0;

  end

  //clock cycle time is 20ns, inverse Clk value per 10ns
  initial Clk = 1'b1;
  always #(`CYCLE_TIME/2) Clk = ~Clk;
end
```

輸入code的機械碼

插入NOP處理Hazard問題

Initialization data memory

Initialization register file

因為CPU沒有做任何處理Hazard的硬體，故只能透過插入NOP指令或是調整指令順序的方式節省cycle數。

什麼時候插入NOP?

EX. add \$3, \$1, \$2

add \$5, \$3, \$4

第一行的\$1+\$2還未寫回\$3，故下一行的\$3內並非預期的值，故插入3個NOP等待

Lab1程式一個輸入請放在DM[0]中，兩個結果放在DM[1]、DM[2]

Testbench.v

```
//Rst signal
initial begin
    cycles = 32'b0;
    Rst = 1'b1;
    #12 Rst = 1'b0;
end

CPU cpu(
    .clk(Clk),
    .rst(Rst)
);

//display all Register value and Data memory content
always @(posedge Clk) begin
    cycles <= cycles + 1;
    if (cycles == `INSTRUCTION_NUMBERS) $finish; // Finish when ex
    $display("PC: %d cycles: %d", cpu.FD_PC>>2, cycles);
    $display(" R00-R07: %08x %08x %08x %08x %08x %08x %08x %08x",
    $display(" R08-R15: %08x %08x %08x %08x %08x %08x %08x %08x",
    $display(" R16-R23: %08x %08x %08x %08x %08x %08x %08x %08x",
    $display(" R24-R31: %08x %08x %08x %08x %08x %08x %08x %08x",
    $display(" 0x00 : %08x %08x %08x %08x %08x %08x %08x %08x",
    $display(" 0x08 : %08x %08x %08x %08x %08x %08x %08x %08x",
end

//generate wave file, it can use gtkwave to display
initial begin
    $dumpfile("cpu_hw.vcd");
    $dumpvars;
end
endmodule
```

顯示所有register及
Data memory內容

產生波形檔

Testbench輸出結果解說

目前執行cycle數

```
PC:      0 cycles:      0
R00-R07: 00000000 00000001 00000002 00000000 00000000 00000000 00000000 00000000
R08-R15: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
R16-R23: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
R24-R31: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x00 : 00000009 00000003 00000000 00000000 00000000 00000000 00000000 00000000
0x08 : 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
PC:      1 cycles:      1
R00-R07: 00000000 00000001 00000002 00000000 00000000 00000000 00000000 00000000
R08-R15: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
R16-R23: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
R24-R31: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x00 : 00000009 00000003 00000000 00000000 00000000 00000000 00000000 00000000
0x08 : 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
PC:      2 cycles:      2
R00-R07: 00000000 00000001 00000002 00000000 00000000 00000000 00000000 00000000
R08-R15: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
R16-R23: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
R24-R31: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x00 : 00000009 00000003 00000000 00000000 00000000 00000000 00000000 00000000
0x08 : 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
```

Register(R00~R31)內數值

Data memory內數值

課堂練習

- 修改課程壓縮檔內的testbench.v檔，使用已定義的加法功能，在Instruction DM initialization程式段中，加入適當指令，作連續加法，使得 $\$4 = 9$

➤ 初始化時，需給定暫存器初值： $\$0=0$ 、 $\$1=1$ 、 $\$2=2$

- 向助教Demo結果
- 佔Lab3成績30%

```
// Instruction DM initialilation
initial
begin
    cpu.IF.instruction[ 0] = 32'b000000_00001_00010_00011_00000_100000; //add $3, $1, $2
    cpu.IF.instruction[ 1] = 32'b000000_00000_00000_00000_00000_100000; //NOP(add $0, $0, $0)
    cpu.IF.instruction[ 2] = 32'b000000_00000_00000_00000_00000_100000; //NOP(add $0, $0, $0)
    cpu.IF.instruction[ 3] = 32'b000000_00000_00000_00000_00000_100000; //NOP(add $0, $0, $0)
    cpu.IF.PC = 0;
end

// Data Memory & Register Files initialilation
initial
begin
    cpu.MEM.DM[0] = 32'd9;
    cpu.MEM.DM[1] = 32'd3;
    for (i=0; i<128; i=i+1) cpu.MEM.DM[i] = 32'b0;

    cpu.ID.REG[0] = 32'd0;
    cpu.ID.REG[1] = 32'd1;
    cpu.ID.REG[2] = 32'd2;
    for (i=3; i<32; i=i+1) cpu.ID.REG[i] = 32'b0;
end
```


作業說明

1. 新增RISC指令(30%)

- R-type : add , sub , and , or , slt
- I-type : lw , sw , beq
- J-type : j

2. 修改 “testbench.v” ，使其能執行Lab1的程式(30%)

- 從MEM讀出(lw)一個給定的輸入值做運算，並將得出的兩個結果存回(sw)MEM。

3. 比較第2部分執行cycle數(10%)

- 第1名10分、2~5名6分、6~10名4分、11~15名2分、以下0分

4. 將六個 “.v” 檔壓縮後，上傳至E-course，壓縮檔使用 “學號_姓名 ” 命名

5. Deadline : 2019/11/20 23:59

參考資料

■ MIPS Instruction Reference

<http://www.mrc.uidaho.edu/mrc/people/jff/digital/MIPSir.html>

附錄

CO Lab3



Department of Electrical Engineering and SoC Research Center
National Chung Cheng University

Icarus Verilog教學

■ 編譯RISC CPU檔案



```
Microsoft Windows [版本 6.3.9600]
(c) 2013 Microsoft Corporation. 著作權所有，並保留一切權利。

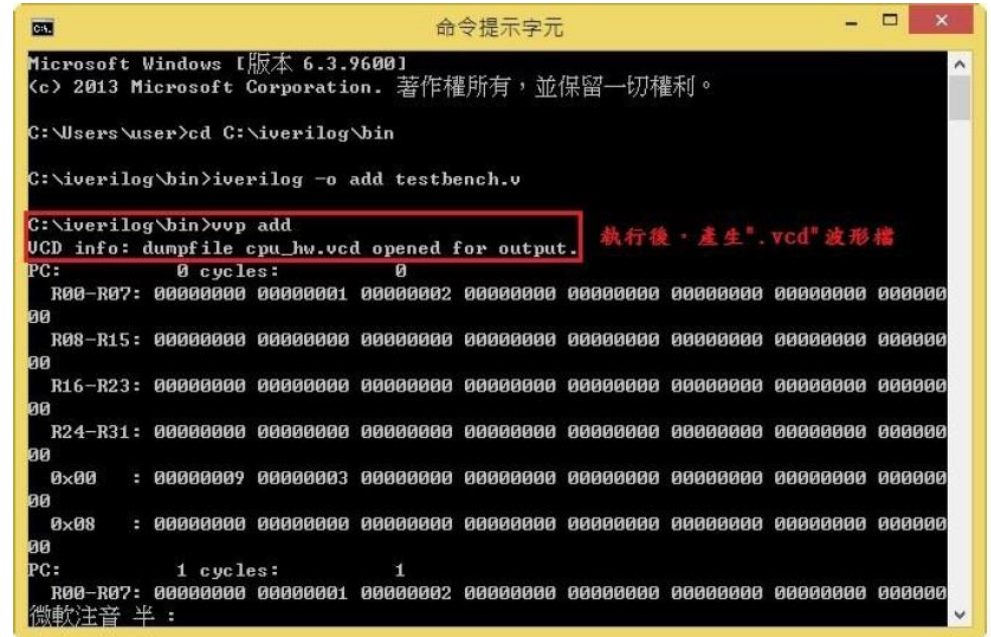
C:\Users\user>cd C:\iverilog\bin

C:\iverilog\bin>iverilog -o add testbench.v_
```

鍵入此段指令，編譯RISC CPU之testbench

微軟注音 半：

■ 執行後，產生波形檔(cpu_hw.vcd)



```
Microsoft Windows [版本 6.3.9600]
(c) 2013 Microsoft Corporation. 著作權所有，並保留一切權利。

C:\Users\user>cd C:\iverilog\bin

C:\iverilog\bin>iverilog -o add testbench.v

C:\iverilog\bin>vvp add
VCD info: dumpfile cpu_hw.vcd opened for output.
PC: 0 cycles: 0
R00-R07: 00000000 00000001 00000002 00000000 00000000 00000000 00000000 00000000
R08-R15: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
R16-R23: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
R24-R31: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x00 : 00000007 00000003 00000000 00000000 00000000 00000000 00000000 00000000
0x08 : 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
PC: 1 cycles: 1
R00-R07: 00000000 00000001 00000002 00000000 00000000 00000000 00000000 00000000
```

執行後，產生".vcd"波形檔

微軟注音 半：

Gtkwave教學

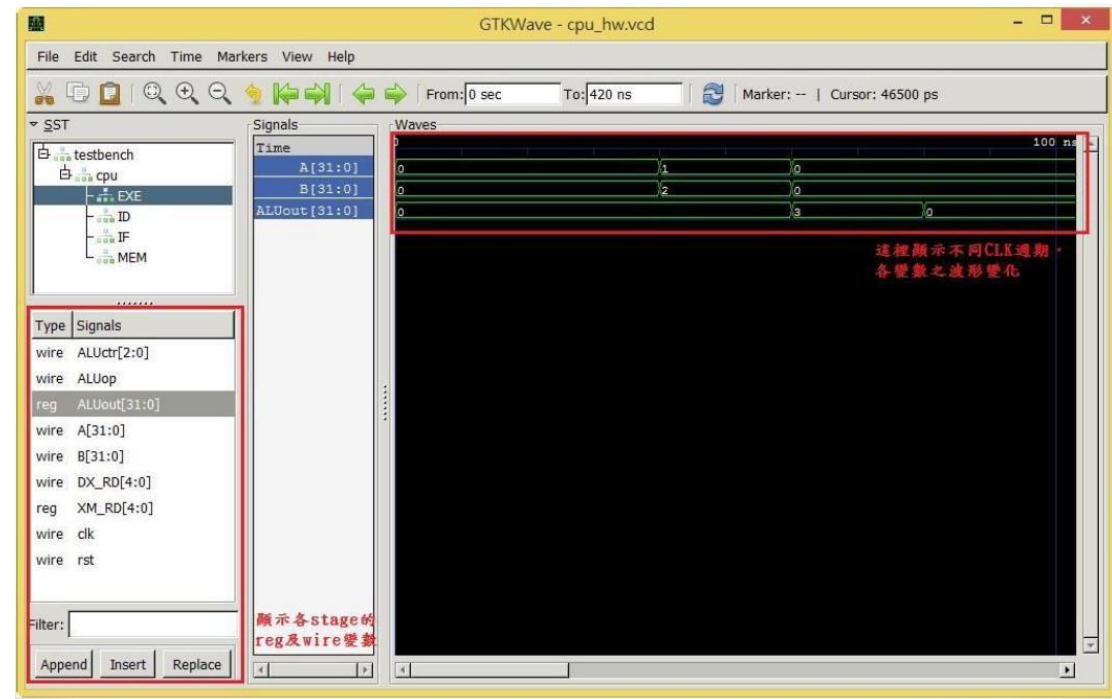
■ 執行Gtkwave，顯示波形檔



```
C:\iverilog\bin>cd C:\iverilog\gtkwave\bin
C:\iverilog\gtkwave\bin>gtkwave cpu_hw.vcd
```

使用gtkwave程式，顯示波形檔

微軟注音 半：



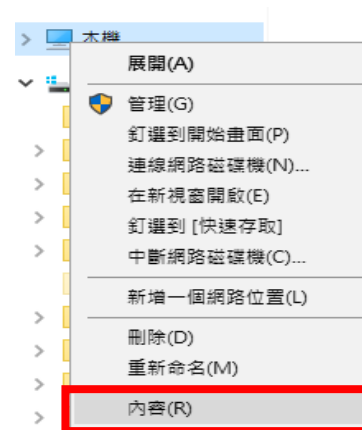
Icarus Verilog進階環境設定

■避免同學將程式全放在bin資料夾編譯、執行，請同學依照下面步驟操作：

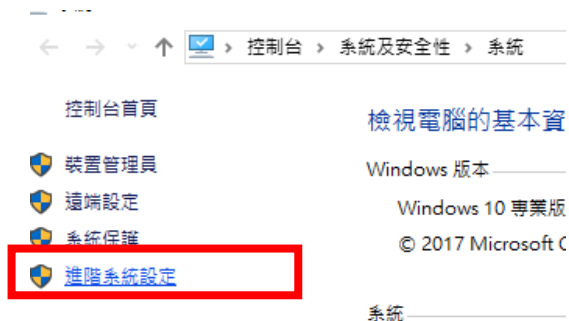
1.打開檔案總管



2.在本機圖示點擊右鍵，選擇內容



點擊進階系統設定

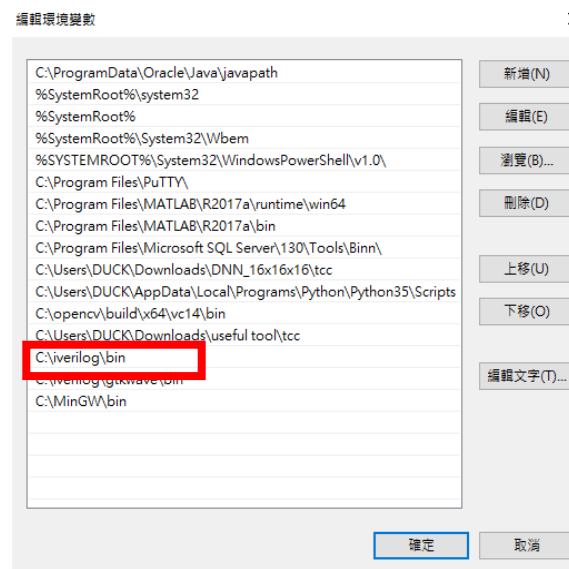
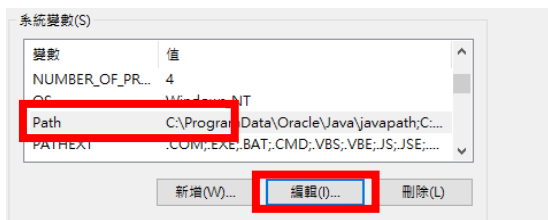


Icarus Verilog進階環境設定

4. 點擊環境變數



5. 點擊path並按下編輯



6. 新增並輸入bin資料夾路徑，按下確定

※路徑為iverilog與gtkwave下的bin資料夾，

已將資料放置同處，同學只需新增一個環境變數