

# 計算機組織 Lab3

---

2016/12/11

助教:李令竹

# Outline

---

- 實驗目的
- 實驗工具
- 實驗介紹
- Gtkwave教學
- 課堂練習
- 作業說明
- 參考資料

# 實驗目的

---

- 使用 Verilog 實作 RISC Processor-瞭解各指令在 RISC 運作方式

# 實驗工具

---

■ iVerilog

■ Gtkwave

# 實驗介紹-規劃各級硬體

---

- RISC架構下的指令，Data path可拆解成five 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

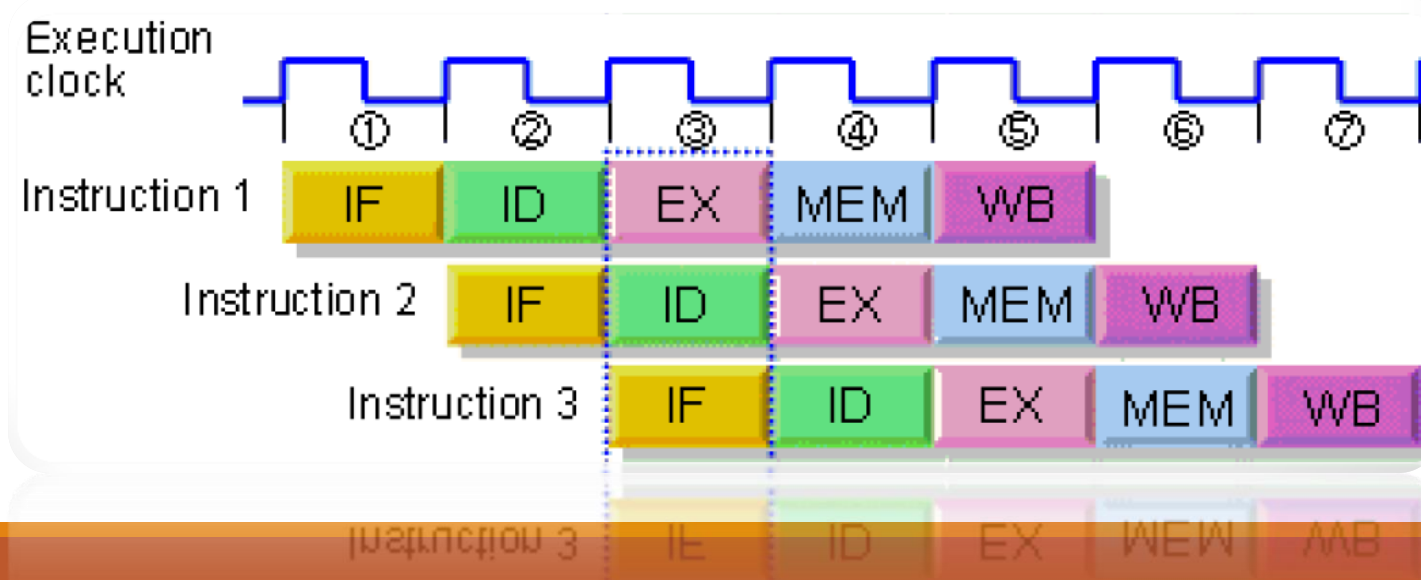


- Five stage：IF(instruction fetch)、ID(instruction decode)、EX(execution)、MEM(memory access)WB(write back)

# RISC Processor in Pipeline Design

## ■ 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



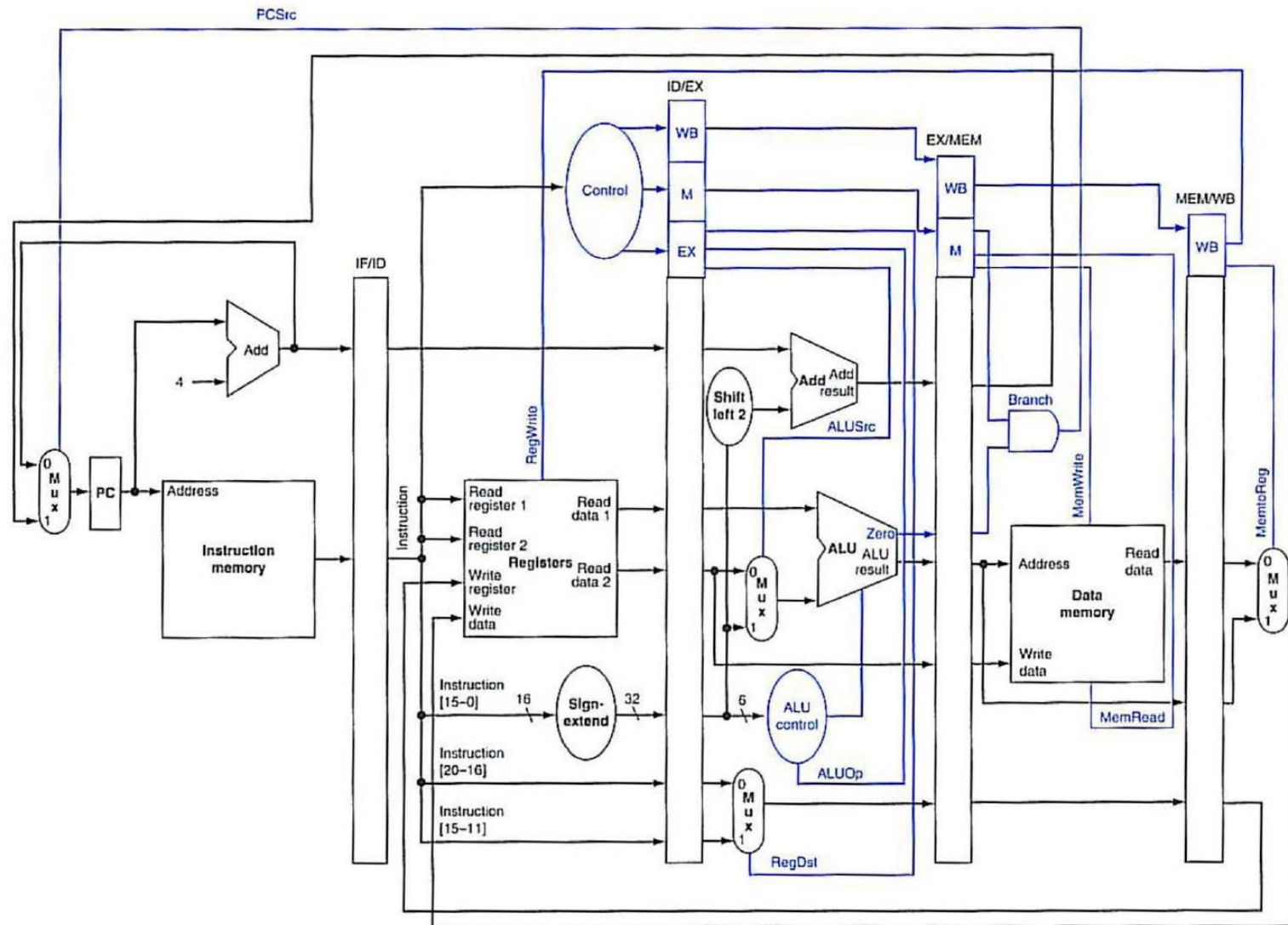
# RISC Processor in Pipeline Design

---

## ■ Pipeline設計

- 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





# RISC Processor Module in verilog RTL

---

Testbench.v

CPU.v

IF.v

ID.v

EXE.v

MEM.v

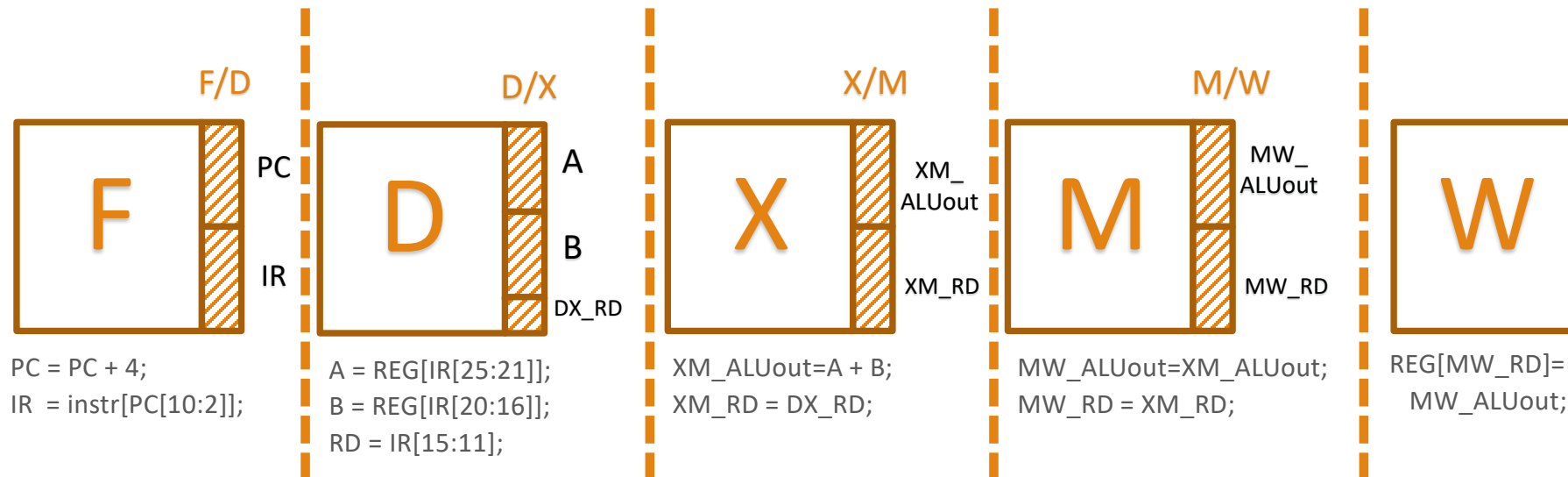


# MOP Design

■ add rs,rs,rt

>reg(rd) = reg(rs) + 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



# 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

格式：“xxx(yyy)”

當xxx在.v為input後的yyy即為其他檔案output

當xxx在.v為onput後的yyy即為其他檔案input

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
```

# 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
    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=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;

    //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

Initialilation data memory

Initialilation register file

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;
    // Finish when excute the 24-th instruction (End label).
    if (cycles == `INSTRUCTION_NUMBERS) $finish;
    $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;
```

秀出所有register及Data memory內容

產生波型檔

# 計算程式耗費Cycle數

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
```

※執行Testbench輸出結果

# Gtkwave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。

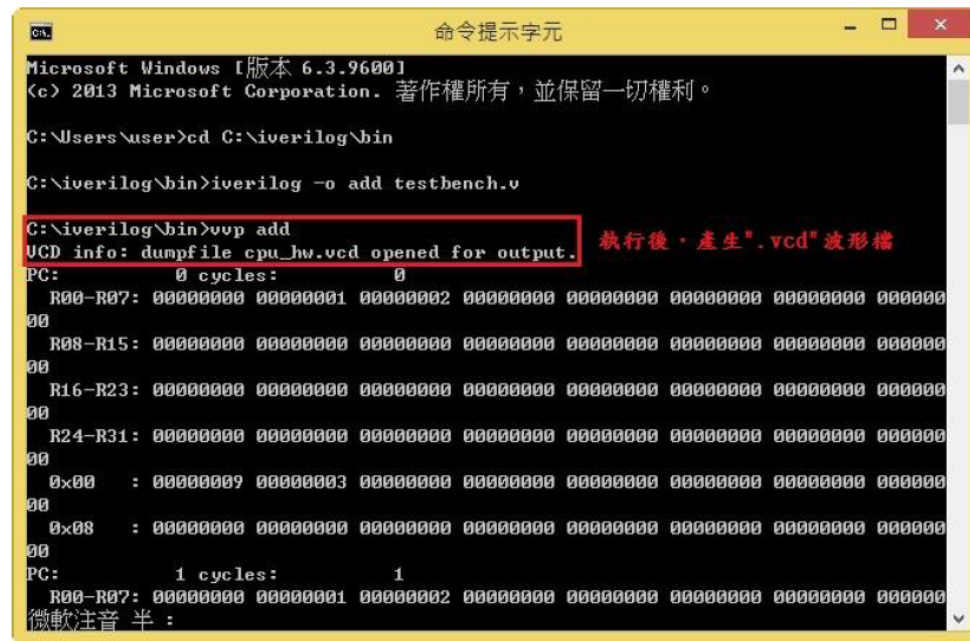
C:\Users\user>cd C:\iverilog\bin

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. 執行後，產生".vcd"波形檔
PC:          0 cycles:          0
R00-R07: 00000000 00000001 00000002 00000000 00000000 00000000 00000000 000000
00
R08-R15: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 000000
00
R16-R23: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 000000
00
R24-R31: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 000000
00
0x00 : 00000000 00000003 00000000 00000000 00000000 00000000 00000000 000000
00
0x08 : 00000000 00000000 00000000 00000000 00000000 00000000 00000000 000000
00
PC:          1 cycles:          1
R00-R07: 00000000 00000001 00000002 00000000 00000000 00000000 00000000 000000
00
微軟注音 半：
```



# Gtkwave教學

---

- 執行Gtkwave，顯示波形檔

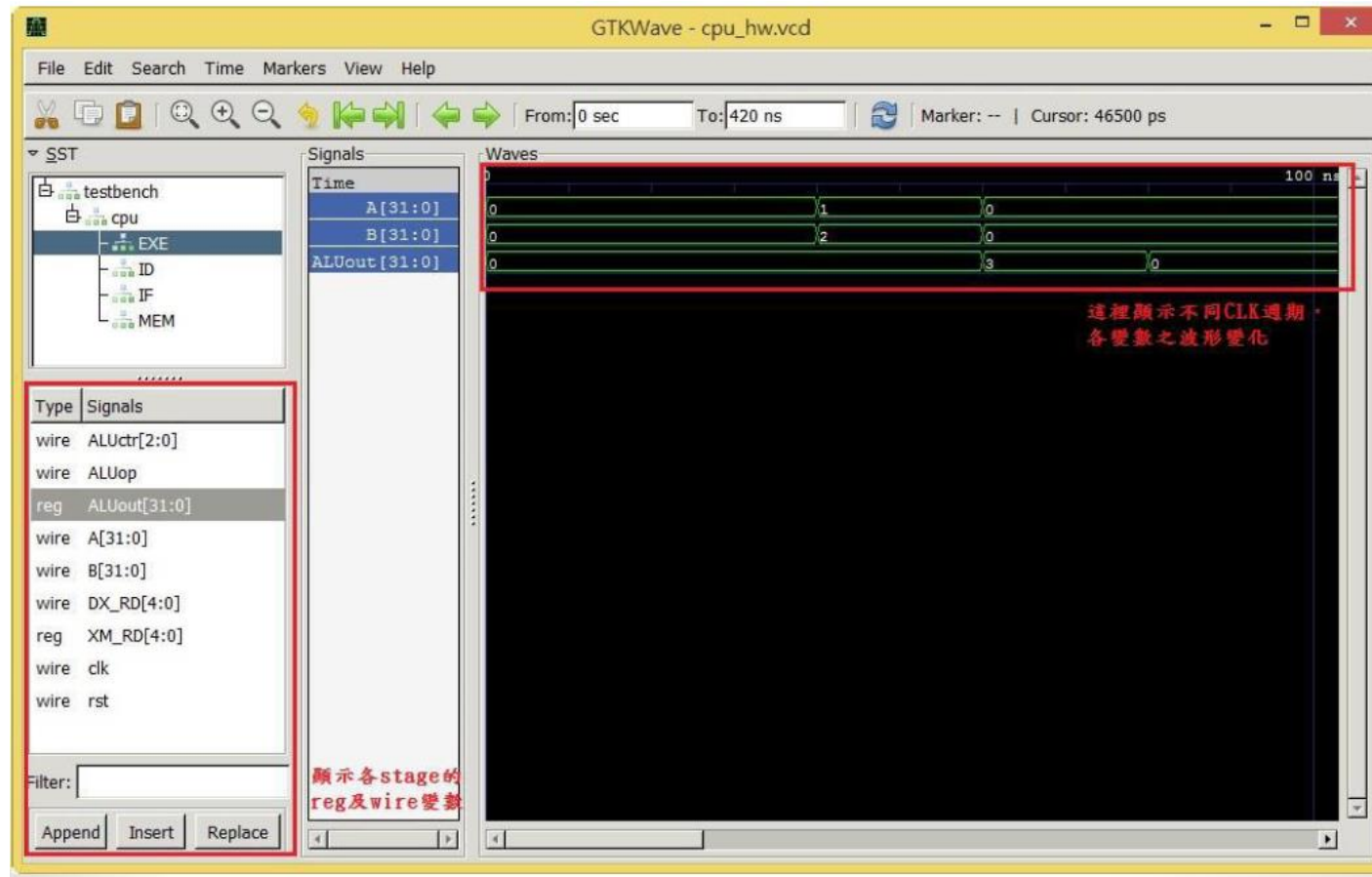


A screenshot of a Windows Command Prompt window titled "命令提示字元". The window has a yellow title bar and a black background. The command prompt shows the following text:

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

Below the command, there is a red text annotation: "使用gtkwave程式，顯示波形檔". At the bottom left of the window, there is a small text label: "微軟注音 半：".

# Gtkwave教學



# 課堂練習

---

- 修改課程壓縮檔內之“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，作連續加法後，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%

# 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”，使其能執行找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:12/18 23:59

# 參考資料

---

- MIPS Instruction Reference

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