計算機組織 Project 3

Part I ：Implement a 5-stage pipelined processor with R-format instruction supported.

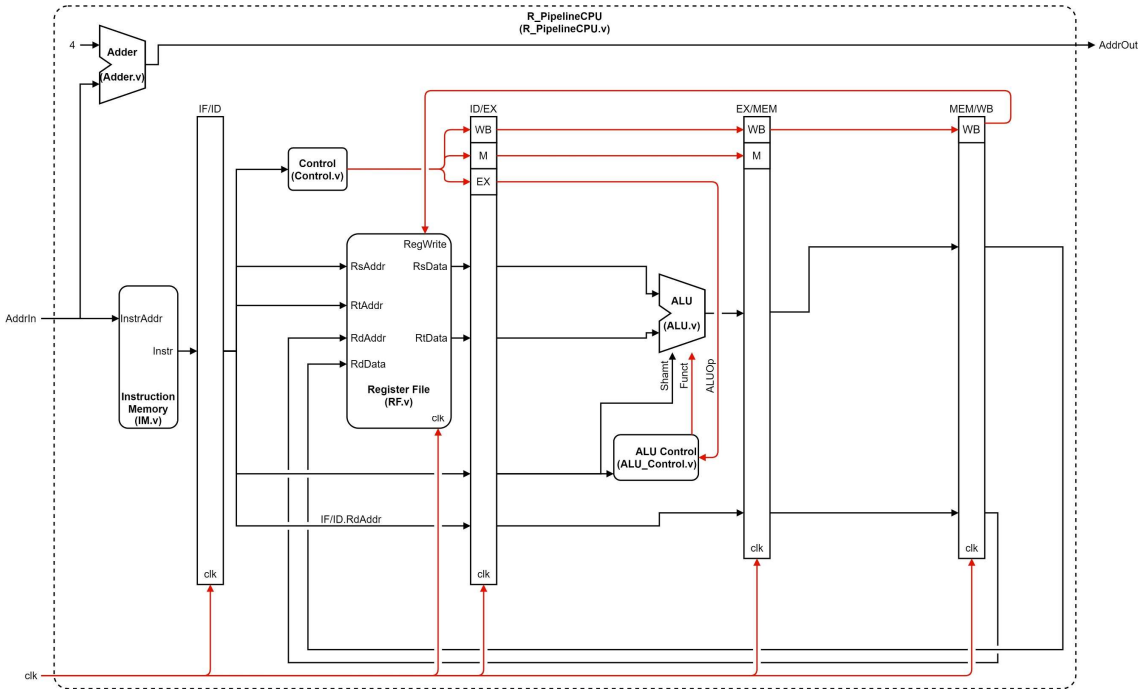


圖 1：支援 R-format 指令處理器架構圖

實作一 32 位元處理器並支援如下 R-format 指令。

Instruction	Example	Meaning	OpCode	funct
Add unsigned	addu $Rd, $Rs, $Rt	$Rd = $Rs + $Rt	000100	001011
Sub. unsigned	subu $Rd, $Rs, $Rt	$Rd = $Rs – $Rt	000100	001101
And	and $Rd, $Rs, $Rt	$Rd = $Rs & $Rt	000100	010010
Shift left logical	sll $Rd, $Rs, shamt	$Rd = $Rs << shamt	000100	100110

註：文字指令轉成 32 位元執行碼之方式請參考 HW1。

註：執行 R-format 指令時，可將 ALUOp 編碼為“10”，使得 ALU Control 辨識 funct 轉換出對應的 ALU 功能代碼 Funct。

I/O Interface

```
module R_PipelineCPU (
        output  wire    [31:0]  AddrOut,
        input   wire    [31:0]  AddrIn,
        input   wire            clk
);
```

Part II ：Implement a 5-stage pipelined processor with I-format instruction supported.

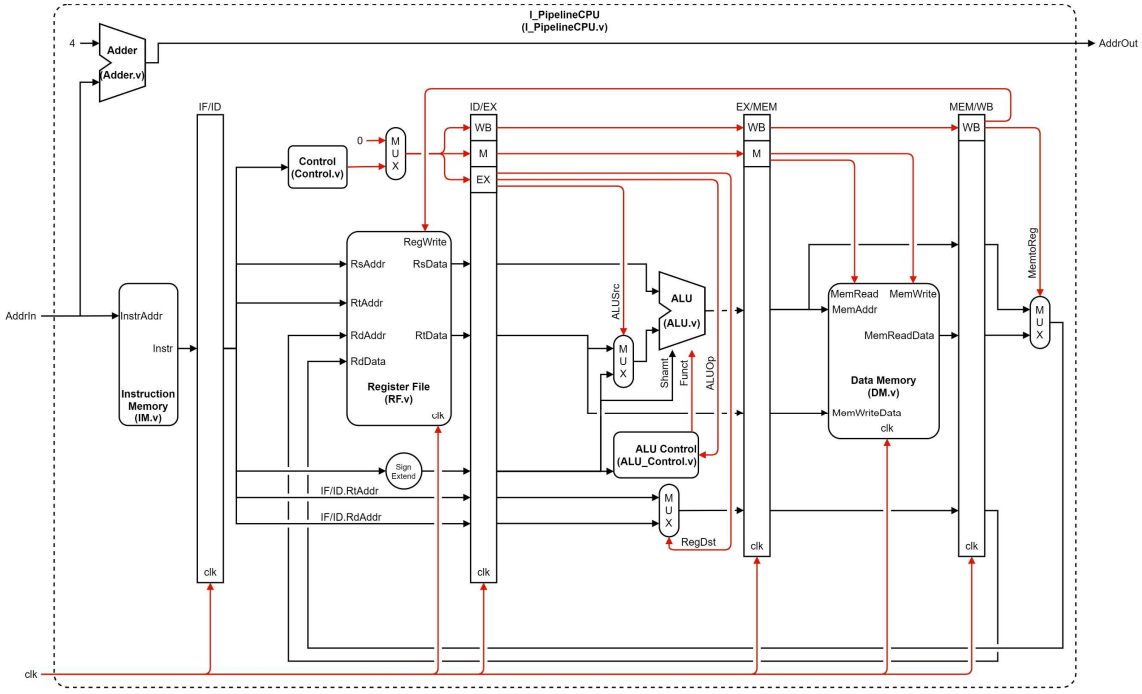


圖 2：支援 I-format 指令處理器架構圖

實作一 32 位元處理器除支援前一 part 的 R-format 且支援如下 I-format 指令。

Instruction	Example	Meaning	OpCode
Add imm. unsigned	addiu $Rt, $Rs, Imm.	$Rt = $Rs + Imm.	001100
Sub. imm. unsigned	subiu $Rt, $Rs, Imm.	$Rt = $Rs – Imm.	001101
Store word	sw $Rt, Imm. ($Rs)	Mem.[$Rs+Imm.] =$Rt	010000
Load word	lw $Rt, Imm. ($Rs)	$Rt =Mem.[$Rs+Imm.]	010001

註：執行 I-format 指令時，可將 ALUOp 編碼為“00”或“01”，使得 ALU Control 不理會 funct，控制 ALU 為“加法”或“減法”並輸出對應的 Funct。

I/O Interface

```
module I_PipelineCPU (
        output  wire    [31:0]  AddrOut,
        input   wire    [31:0]  AddrIn,
        input   wire            clk
);
```

Part III ：Implement a 5-stage pipelined processor with forwarding and hazard detection.

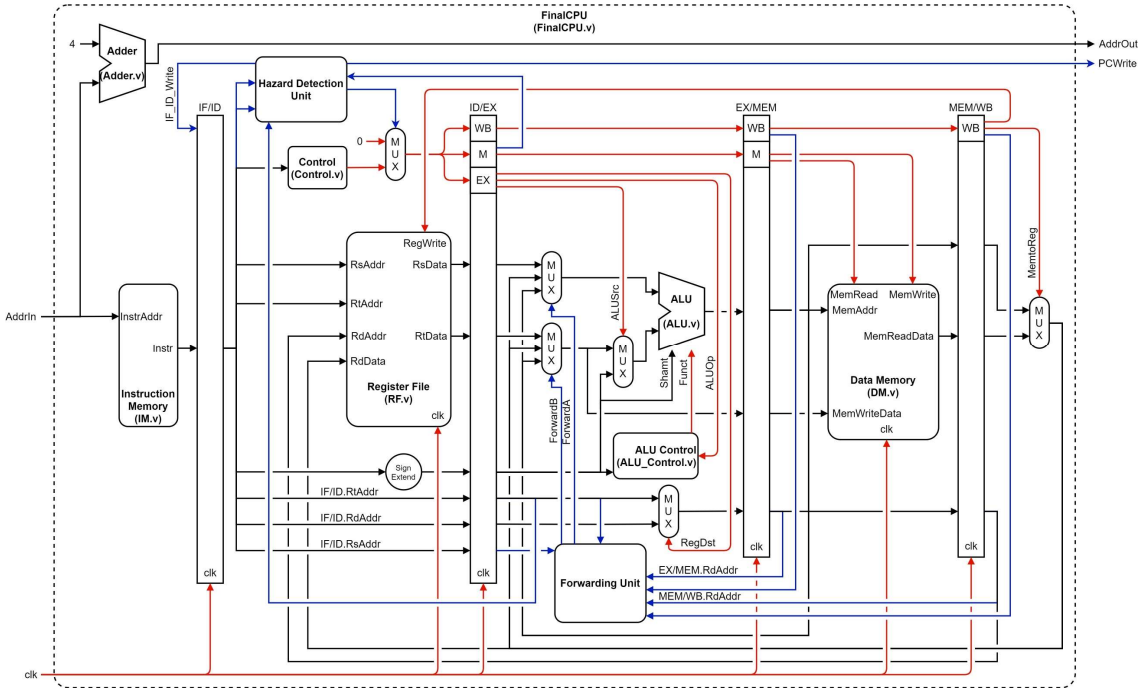


圖 3：支援 Forwarding 和 Hazard Detection 處理器架構圖

實作一 32 位元處理器除支援前二 part 的 R-format 和 I-format，並支援 Forwarding 和 Hazard Detection。

I/O Interface

```
module FinalCPU (
        output  wire            PCWrite,
        output  wire    [31:0]  AddrOut,
        input   wire    [31:0]  AddrIn,
        input   wire            clk
);
```

Testbench 動作說明

a. 初始化

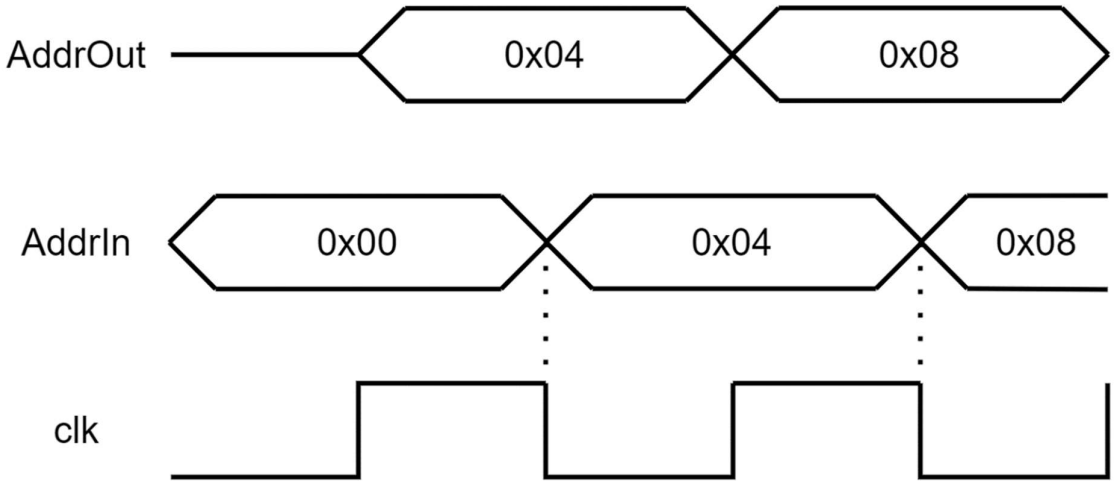
本次 Testbench（『tb_R_PipelineCPU.v』、『tb_I_PipelineCPU.v』、『tb_FinalCPU.v』）開始執行時，會根據『/testbench/IM.v』、『/testbench/RF.v』、『/testbench/DM.v』（Part I 除外）分別初始化 Instruction Memory、Register File、Data Memory。

b. 時脈

本次 testbench 會產生週期時脈（clk），用於驅動 CPU 模組。

c. 定址及結束

對於 AddrIn 訊號，Testbench 會初始化為 0，並在每次 clk 正緣前將 AddrOut 賦予 AddrIn（Part III 會以 PCWrite 作為控制訊號，只有其為"1"時，AddrIn 才會更新）。直至 AddrIn 大於或等於本次作業指令的最大定址空間（0x7D），則 Testbench 結束執行，並輸出當下暫存器及記憶體內容（『/testbench/RF.out』、『/testbench/DM.out』）用於分析程式正確性。下圖為 Testbench 動作基本波形：



注意：由於 Testbench 會根據 AddrIn 作為結束判斷，故當系統 AddrOut 失效或程式為無窮迴圈，則模擬將持續進行，需手動終止模擬並判斷問題點。

Submission

報告書（B10YDDXXX.pdf）：

a. 封面。
b. 各新模組程式碼截圖並說明（延用的模組無須截圖，但須簡要說明其功能）。
c. 各章節範例程式（『/testbench/IM.dat』）執行結果（『/testbench/RF.out』，『/testbench/DM.out』）截圖並說明。
d. 作業總結與心得。
e. 報告書以 30 頁為限，請妥善排版。
f. 匯出為 PDF 檔，並以學號命名——"B10XXXXXX.pdf"。

壓縮檔（B10YDDXXX.zip）：

- 報告書（B10YDDXXX.pdf）
- Part I
    a. R_PipelineCPU.v
    b. IM.v
    c. RF.v
    d. 其他必要的.v 檔案
- Part II
    a. I_PipelineCPU.v
    b. IM.v
    c. RF.v
    d. DM.v
    e. 其他必要的.v 檔案
- Part III
    a. FinalCPU.v
    b. IM.v
    c. RF.v
    d. DM.v
    e. 其他必要的.v 檔案

評分：

a. 主程式：使用另外產生的 testbench 測試，Part I（30%）、Part II（30%）、Part III（10%）。
b. 各程式截圖，並描述流程及作法（15%）。
c. 各章節範例程式執行結果，截圖並說明（10%）。
d. 格式、心得、完整性（5%）。
e. 抄襲則以零分計算。

繳交時間：110/06/17　　13:00 前上傳至 Moodle。