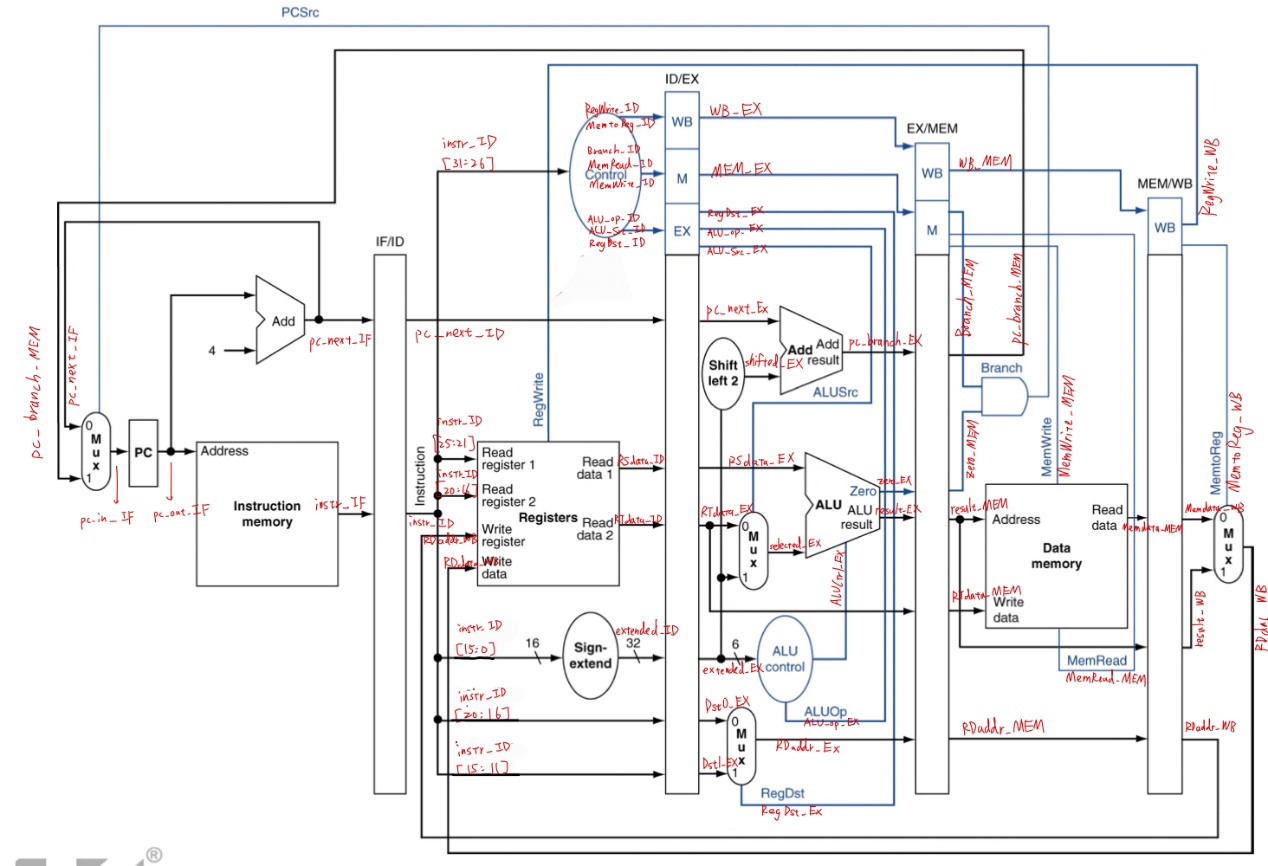
**Computer Organization Lab4**

**Name: 陳子祈**

**ID: 0819823**

**Architecture diagrams:**



|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **R type** | | | | | | |
| **Instruction set** | **Op code** | **rs** | **rt** | **rd** | **shamt** | **funct** |
| **Instr location** | [31:26] | [25:21] | [20:16] | [15:11] | [10:6] | [5:0] |
| add$rd,$rs,$rt | 000000(0) |  |  |  | 00000 | 100000(32) |
| sub$rd,$rs,$rt | 000000(0) |  |  |  | 00000 | 100010(34) |
| and $rd,$rs,$rt | 000000(0) |  |  |  | 00000 | 100100(36) |
| or$rd,$rs,$rt | 000000(0) |  |  |  | 00000 | 100101(37) |
| slt$rd,$rs,$rt | 000000(0) |  |  |  | 00000 | 101010(42) |
| mult$rd,$rs,$rt | 000000(0) |  |  |  | 00000 | 011000(24) |
|  | | | | | | |
| **I type** | | | | | | |
| **Instruction set** | **Op code** | **rs** | **rt** | **immediate** | | |
| **Instr location** | [31:26] | [25:21] | [20:16] | [15:0] | | |
| addi$rt,$rs,imm | 001000(8) |  |  |  | | |
| slti$rt,$rs,imm | 001010(10) |  |  |  | | |
| beq$rt,$rs,imm | 000100(4) |  |  |  | | |

**Hardware module analysis:**

**(explain how the design work and its pros and cons)**

Pipeline MIPS CPU每過一段pipeline clock period就做PC與暫存器的運算，pipeline clock period取大於所有stage執行指令cycle time的時間，因為總共有5個stage，所以會有5個指令同時在CPU不同stage被執行，每個stage的間隔會有buffer存前一個stage的output與下一個stage的input，如此設計可以讓每個stage之間的訊號不會直接互通，而是等一段pipeline clock period才將前一個stage的output與下一個stage的input。PC的運算只有分為Sequential的運算與beq跳行的運算。五個stage分別如下:

1. IF: Instruction fetch from memory 從memory請求指令
2. ID: Instruction decode & register read 解碼指令、產生控制訊號，並把暫存器的資料讀出來
3. EX: Execution operation or calculate address 執行指令，這次實驗使用到lw、sw存取記憶體的指令，因此需計算地址
4. MEM: Access memory operand 存取記憶體資料，若執行sw指令，記憶體的資料就要被更改，若執行lw指令，則讀取記憶體的資料
5. WB: Write result back to register 依據不同指令，將ALU計算出來的結果、記憶體的資料、常數或PC+4寫回去Write register，PC也會依據branch更新PC

Single cycle MIPS CPU優點就是不會發生hazards; 缺點就是以Longest delay的指令當作clock period，導致執行大部分指令都有很多CPU idle的時間。不過Pipeline MIPS CPU則剛好相反，優點是以Longest delay of a stage的指令當作clock period，減少很多CPU idle的時間; 不過這次Lab沒有處理hazards，所以缺點是萬一發生data hazards或load use hazards的話最後結果就會出錯。

各module的description:

1) Decoder

功能：透過6bit的instruction operation code 決定各種控制訊號。

Port description：

instr\_op\_i ：6bit input instruction operation code

RegWrite\_o ：1bit output RegFile Write or not

ALU\_op\_o ：3bit output for ALU\_Ctrl to determine operation type

ALUSrc\_o ：1bit output determine ALU source

RegDst\_o ：1 bit output determine Read reg2 is rt or rd

Branch\_o ：1bit output the instruction is branch type or not

MemRead\_o : 1bit output for Data memory to determine read memory data or not

MemWrite\_o : 1bit output for Data memory to determine write memory data or not

MemtoReg\_o : 2bit output to determine where Register write data is from

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Instr\_op  [31:26] | | Instruction | RegDst | ALUSrc | Mem  toReg | Reg  Write | Mem  Read | Mem  Write | Branch | ALU  Op  [2:0] |
| [31:29] | [28:26] |  |  |  |  |  |  |  |  |  |
| 000 | 000 | R-type | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 000 |
| 100 | beq | X | 0 | X | 0 | 0 | 0 | 1 | 011 |
| I-type | | | | | | | | | | |
| 001 | 000 | addi | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 010 |
| 010 | slti | 111 |
| 100 |  | lw | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 010 |
| 101 |  | sw | 0 | 1 | X | 0 | 0 | 1 | 0 | 010 |

2) ALU\_Ctrl

功能：將ALU\_op及function code轉成ALU所需的ALUCtrl，決定ALU的動作及控制其他MUX、Shifter。

Port description：

funct\_i ：6bit input function code

ALUOp\_i ：3bit input for ALU\_Ctrl to determine operation type

ALUCtrl\_o ：4bit output to ALU control

|  |  |  |  |
| --- | --- | --- | --- |
| ALUOp\_i | funct\_i | operation | ALUCtrl |
| R-type | | | |
| 000 | 100000(32) | add | 0010 |
| 100010(34) | sub | 0110 |
| 100100(36) | and | 0000 |
| 100101(37) | or | 0001 |
| 101010(42) | slt | 0111 |
| 011000(24) | mult | 1111 |
| I-type | | | |
| 010 |  | addi  lw  sw | 0010 |
| 011 |  | beq | 0110 |
| 111 |  | slti | 0111 |

3) ALU

功能：32bit運算邏輯單位，參考課本附錄程式，可做add、sub、or、and、slt、mult。

Port description：

src1\_i ：32bit input data

src2\_i ：32bit input data

ctrl\_i ：4bit ALU\_Control

result\_o ：32bit result for ALU

zero\_o ：1 bit when the output is 0, zero must be set

4) Adder

功能：輸入兩個data輸出其相加結果。

Port description：

src1\_i ：32bit input data

src2\_i ：32bit input data

sum\_o ：32bit output sum

5) Sign\_Extend

功能：將輸入data做Sign Extend，data\_i複製到data\_o低16位，data\_i最高位bit複製到data\_o高16位。

Port description：

data\_i ：16bit input data

data\_o ：32bit output data

6) Shift\_Left\_Two\_32

功能：將input data左移兩個bit。

Port description：

data\_i ：32bit input data

data\_o ：32bit output data

7) MUX\_2to1

功能：如果 select\_i = 0 則輸出 data0\_i；select\_i = 1 則輸出data1\_i。

Port description：

data0\_i ：32bit input data

data1\_i ：32bit input data

select\_i ：1bit select for MUX

data\_o ：32bit output data

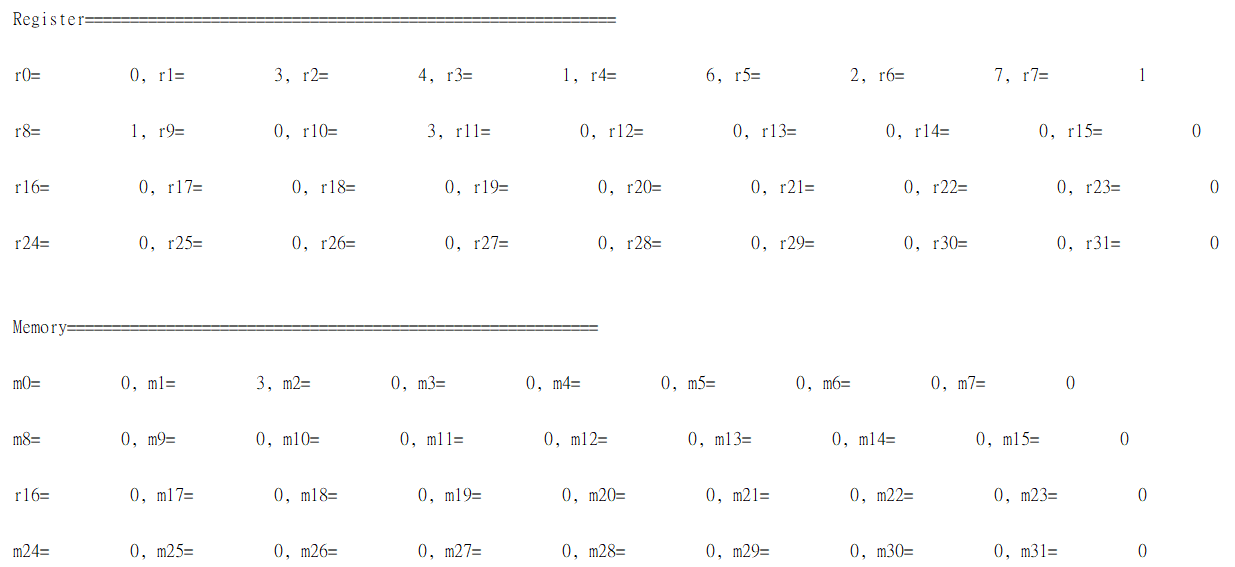
8) Pipe\_CPU\_1

功能：將上述所提到之Module依照Architecture diagram的附圖做連接，完成Pipeline CPU。

**Finished part:**

**(show the screenshot of the simulation result and waveform, and explain it)**

使用“CO\_P4\_test\_1.txt”作為指令輸入，最後各register與data memory結果如下:



此結果與答案相符。

使用“CO\_P4\_test\_2.txt”作為指令輸入，最後各register與data memory結果如下:

一張含有 桌 的圖片

自動產生的描述

因為這份指令輸入的不同指令之間有data dependency，會產生data hazards或load use hazards，所以此結果與答案不符。詳細來看，可以發現Instruction 1、Instruction 2發生data hazard，Instruction 2讀到的r1會是0，r2就會是0 + 4 = 4; 同樣的，Instruction 5、Instruction 6發生load use hazard，Instruction 6讀到的r4會是0，r5就會是0 – 8 = -8; 還有另外一個地方也是，Instruction 8、Instruction 9發生 data hazard，Instruction 8讀到的r7會是0，r8就會是0 & 8 = 0。

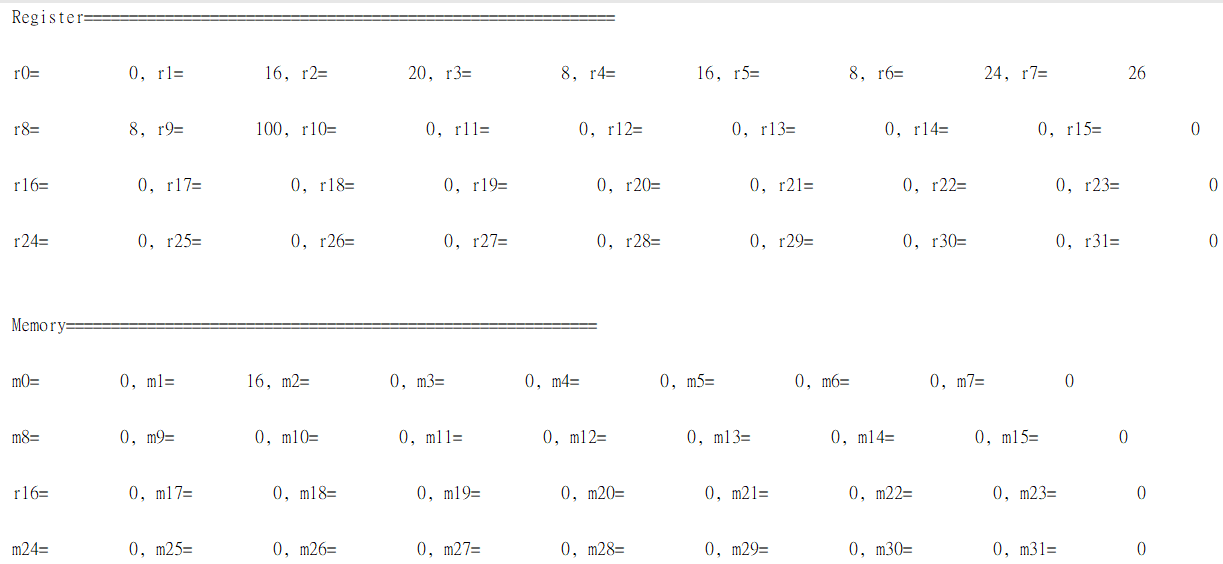
**Bonus (optional):**

將CO\_P4\_test\_2.txt指令重新編排如下:

一張含有 桌 的圖片

自動產生的描述

將此編排存到CO\_P4\_test\_3.txt(再請助教用這個檔案測試)，使用“CO\_P4\_test\_3.txt”作為指令輸入，最後各register與data memory結果如下:



重新編排指令可以手動解決hazards的問題，可發現r2、r5、r8的結果都是正確的。

**Problems you met and solutions:**

1. 暫存器bit總數數錯

這次實作Pipeline CPU需要在每個stage之間架好Pipeline register的模組，因為要依據每個stage的輸入多少bit輸出就要多少bit，所以要計算每個stage輸出的暫存器總bit數量，一不小心數錯讓我結果出錯，數對之後結果就與答案相符了。

**Summary:**

這次實作Pipeline MIPS CPU不會太困難，因為之前已經做過兩次Single cycle MIPS CPU，這次只要將之前的模組加入Pipeline register的設計，就可以實作出來簡易的Pipeline MIPS CPU。不過這次沒有處理hazards的部分，希望下次Lab可以實作有處理hazards的Pipeline MIPS CPU，感覺滿有挑戰性。