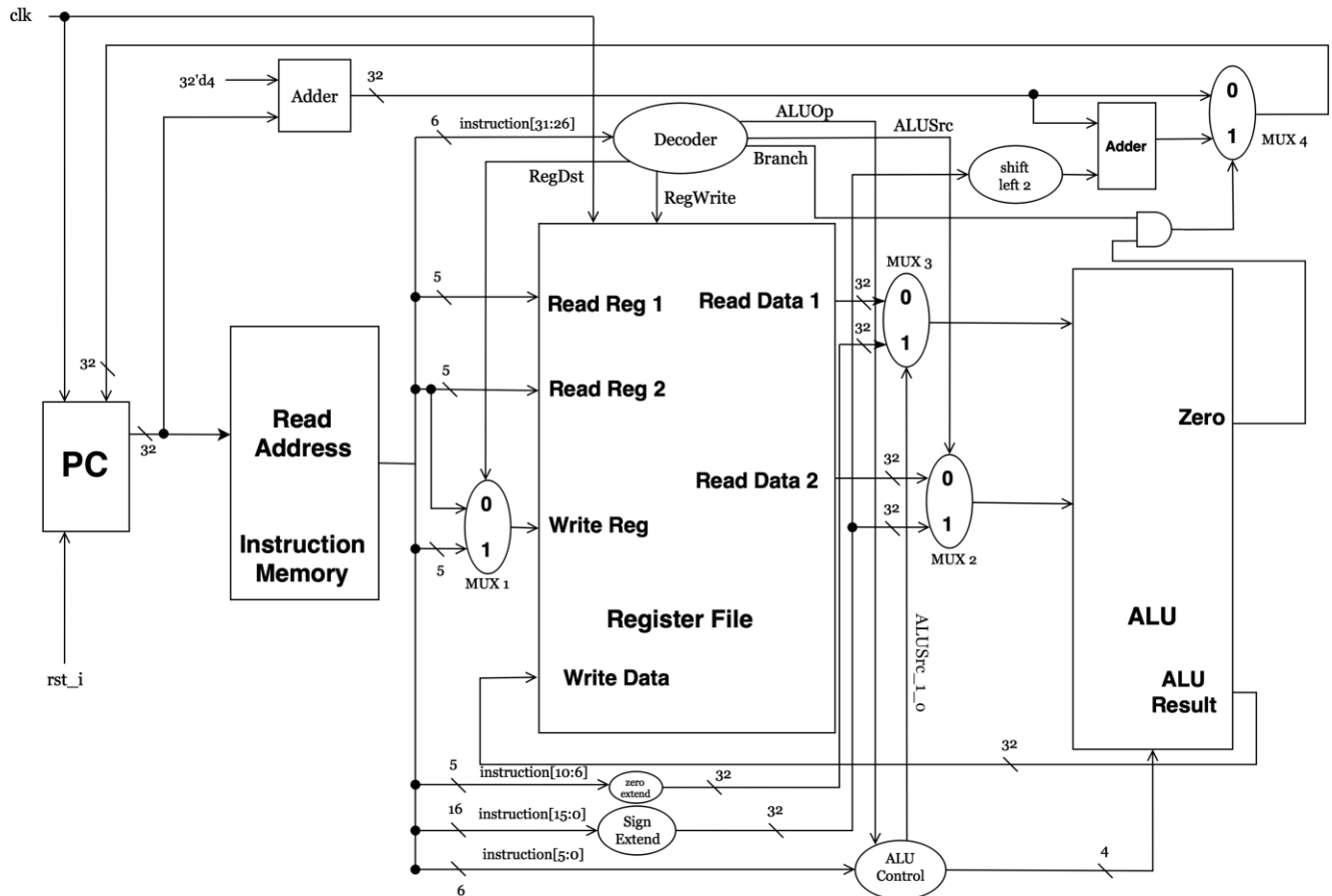


Computer Organization

Architecture diagram:



Detailed description of the implementation:

一、以 Simple_Single_CPU 為主要的模組，會去呼叫這個 simple CPU 所需要的所有模組，包含 PC、IM、RF、Decoder、ALU_Ctrl、Sign_Extend、MUX，ALU 等模組。此 CPU 為一個 clock cycle 執行一個指令。

二、Decoder 模組：

1. 我是根據傳入的 Op filed 去 map 那些輸出的值，以下是我採用的對應表：

| instr_op_i | ALU_op_o | Type | RegDst_o | RegWrite_o | ALUSrc_o | Branch_o |
|------------|----------|------|----------|------------|----------|----------|
| 6'd0 | 3'b000 | R | 1'b1 | 1'b1 | 1'b0 | 1'b0 |
| 6'd4 | 3'b001 | BEQ | 1'b0 | 1'b0 | 1'b0 | 1'b1 |
| 6'd5 | 3'b010 | BNE | 1'b0 | 1'b0 | 1'b0 | 1'b1 |
| 6'd8 | 3'b011 | I | 1'b0 | 1'b1 | 1'b1 | 1'b0 |
| 6'd13 | 3'b100 | ORI | 1'b0 | 1'b1 | 1'b1 | 1'b0 |
| 6'd15 | 3'b101 | LUI | 1'b0 | 1'b1 | 1'b1 | 1'b0 |

我們的 ALU_op_o 與 instr_op_i 之間並沒有什麼特殊的關係，純粹只是循序的指

定。

三、ALU_Control 模組：

1. 因為有 shamt 要送進 ALU 的緣故，我在這個模組有多加一個輸出序號 ALUSrc_1_o，此訊號會傳到 Read_Data1 後方的 mux 當作選擇線。
2. 只有 R-type 的指令需要再次利用 function filed 判斷要給 ALU 什麼值。
3. 以下為我們採用的對應表：

R-type 以外：

| ALUOp_i | Instruction | ALUSrc_1_o | ALUCtrl_o |
|---------|-------------|------------|-----------|
| 3'b001 | BEQ | 1'b0 | 4'b0110 |
| 3'b010 | BNE | 1'b0 | 4'b1110 |
| 3'b011 | Addi | 1'b0 | 4'b0010 |
| 3'b100 | LUI | 1'b0 | 4'b0100 |
| 3'b101 | ORI | 1'b0 | 4'b0001 |

這五個指令除了 BEQ、Addi 的 ALUCtrl_o 是拿 lab1 中減法、加法做對應，其它是拿尚未被佔用的指令去指派的。

R-type：

| ALUOp_i | Instruction | ALUSrc_1_o | ALUCtrl_o |
|---------|-------------|------------|-----------|
| 3'b000 | ADD | 1'b0 | 4'b0010 |
| | SUB | 1'b0 | 4'b0110 |
| | AND | 1'b0 | 4'b0000 |
| | OR | 1'b0 | 4'b0001 |
| | SLT | 1'b0 | 4'b0111 |
| | SLTU | 1'b0 | 4'b1111 |
| | SLL | 1'b1 | 4'b0011 |
| | SLLV | 1'b0 | 4'b0011 |

SLL 指令因為會用到 shamt，所以它對應的 ALUSrc_1_o 會是 1。

R-type 中前五個指令是照 lab1 去指定的，而後面三個指令是拿尚未被佔用的值去指定的。

四、MUX 3：

MUX 3 的出現主要是要處理 SLL 及 SLLV 運算，因為在這兩個運算中，要被 shift 的運算元皆由 Read data2 來，而 shift amount 則是從 Read data1 或是 shamt 來，為了不要多接一條線進去 ALU 單獨處理 shamt，我們選擇在 Read data1 後方加一個 2-to-1 mux，輸入為 Read data1 及 shamt，利用 ALU_Control 送出的 ALUSrc_1_o 作為選擇線，ALUSrc_1_o 為 0 選 Read data1 作為輸出，反之則選 zero-signed shamt 作為輸出。

五、ALU 模組：

1. 各種運算與對應的 ctrl_i 表：

| Operation | ctrl_i |
|-----------|---------|
| AND | 4'b0000 |
| OR | 4'b0001 |
| ADD | 4'b0010 |
| SLL, SLLV | 4'b0011 |
| LUI | 4'b0100 |
| SUB, BEQ | 4'b0110 |

| | |
|------|---------|
| SLT | 4'b0111 |
| BNE | 4'b1110 |
| SLTU | 4'b1111 |

- 基本上跟上一次 lab 的 ALU 模組很像，不一樣的地方在於內部我們實作的是 33-bit ALU，且判斷 A_invert 的方法有稍微改變。
- 為了實作 SLTU，我們改實作 33-bit 的 ALU，第 33bit 是根據 ctrl_i 的值去指派，當 ctrl_i 代表的 SLTU 時我們將第 33bit 皆設為 0，其他狀況第 33bit 跟第 32bit 為同值，以此來實作 SLTU。
- 判斷 A_invert 的方法我們改為若 ctrl_i 最高位為 1 則 A_invert 的訊號為 1，反之 A_invert 的訊號值跟 ctrl_i 最高位相同，這樣改是因為要配合 1-bit ALU 的 operation 訊號(原本只用 ctrl_i 最高位會導致 SLTU 跟 BNE 運算結果錯誤)。
- 在 Decoder 把 BEQ、BNE 分開是因為 ALU 只有一個 zero 輸出，為了讓 BEQ 跟 BNE 能順利執行，在 BEQ 指令下，result 為 0 則 zero 輸出為 1，反之為 0；在 BNE 指令下，result 為 0 則 zero 輸出為 0，反之為 1，藉由這種方法讓 Branch 可以順利執行。

Problems encountered and solutions:

- SLTU 的實作方法，原本不太知道這個方法該怎麼利用 lab1 的程式碼去實作，想了一段時間後看到這次 lab 要用到的 sign-extension，想到可以使用 sign-extension 的方法去把原本 32-bit 的運算元改成 33bit 且第 33bit 是根據不同的運算而決定的，不是 SLTU 的運算第 33bit 接跟第 32bit 一樣，而在 SLTU 運算下第 33bit 皆為 0。
- 原本想說在實作 BEQ 跟 BNE 的時候是不是要從 ALU 多接一條線出來，因為如果不接的話 BNE 好像無法實作，最後想到可以讓 zero 不是只在 result 為 0 的時候為 1，在 BNE 的運算下，zero 在 result 不為 0 時要為 1，BEQ 時則維持原狀，利用 ctrl 訊號去控制，這樣就可不必從 ALU 多接線出來另外做 BNE 的 branch 判斷。
- SLTU 及 BNE 原本的 ctrl_i 是跟 SLT 及 BEQ 一樣，但這樣做的結果會導致結果錯誤，原因在第 33bit 的值及 zero 的輸出，故我們將 SLTU 及 BNE 單獨指派 ctrl_i，以利設定 1-bit ALU 的輸入訊號及 zero 輸出。

Lesson learnt (if any):

- 如何適當的配對現有的運算與尚未被使用到的 opcode。
- SLTU 的實作方法。