0416315 王定偉、0416005 張彧豪

**Computer Organization Lab3**

**Architecture diagram:**

**lalalala**

**Detailed description of the implementation:**

1. 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、Data\_memory等模組。此CPU為一個clock cycle執行一個指令。
2. Decoder模組：
3. 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 | MemToReg\_o  MemRead\_o  MemWrite\_o | RegDst\_o  RegWrite\_o | ALUSrc\_o | Branch\_o  Brach\_type\_o  Jump\_o |
| 6’d0 | 3’b000 | R | 2’b00  1’b0  1’b0 | 2’b01  1’b1 | 1’b0 | 1’b0  2’b00  1’b0 |
| 6’d2 | 3’b111 | J | 2’b00  1’b0  1’b0 | 2’b00  1’b0 | 1’b0 | 1’b0  2’b00  1’b1 |
| 6’d3 | 3’b111 | JAL | 2’b11  1’b0  1’b0 | 2’b10  1’b0 | 1’b0 | 1’b0  2’b00  1’b1 |
| 6’d4 | 3’b001 | BEQ | 2’b00  1’b0  1’b0 | 2’b00  1’b0 | 1’b0 | 1’b1  2’b00  1’b0 |
| 6’d5 | 3’b010 | BNE,  BNEZ | 2’b00  1’b0  1’b0 | 2’b00  1’b0 | 1’b0 | 1’b1  2’b11  1’b0 |
| 6’d6 | 3’b001 | BLT | 2’b00  1’b0  1’b0 | 2’b00  1’b0 | 1’b0 | 1’b1  2’b10  1’b0 |
| 6’d7 | 3’b001 | BLE | 2’b00  1’b0  1’b0 | 2’b00  1’b0 | 1’b0 | 1’b1  2’b01  1’b0 |
| 6’d8 | 3’b011 | ADDI | 2’b00  1’b0  1’b0 | 2’b00  1’b1 | 1’b1 | 1’b0  2’b00  1’b0 |
| 6’d13 | 3’b100 | ORI | 2’b00  1’b0  1’b0 | 2’b00  1’b1 | 1’b1 | 1’b0  2’b00  1’b0 |
| 6’d15 | 3’b101 | LI | 2’b10  1’b0  1’b0 | 2’b00  1’b1 | 1’b1 | 1’b0  2’b00  1’b0 |
| 6’d35 | 3’b011 | LW | 2’b01  1’b1  1’b0 | 2’b00  1’b1 | 1’b1 | 1’b0  2’b00  1’b0 |
| 6’d43 | 3’b011 | SW | 2’b00  1’b0  1’b1 | 2’b01  1’b0 | 1’b1 | 1’b0  2’b00  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。

1. ALU\_Control模組：
2. 這個ALU\_Control模組跟Lab2得非常相似，差異在於這個ALU\_Control有多一條輸出線為Junp\_type，這條輸出線的用意是要幫助我們實作的JR指令用的。
3. 以下為我們採用的對應表：

R-type 以外：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ALUOp\_i | Instruction | ALUSrc\_1\_o | ALUCtrl\_o | Jump\_type |
| 3’b001 | BEQ,BLT,BLE | 1’b0 | 4’b0110 | 1’b0 |
| 3’b010 | BNE,BNEZ | 1’b0 | 4’b1110 | 1’b0 |
| 3’b011 | Addi,LW,SW | 1’b0 | 4’b0010 | 1’b0 |
| 3’b100 | LI | 1’b0 | 4’b0100 | 1’b0 |
| 3’b101 | ORI | 1’b0 | 4’b0001 | 1’b0 |

這五個指令除了BEQ、BLT、BLE、BNE、BNEZ、Addi、LW、SW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 |
| MUL | 1’b0 | 4’b0101 |
| JR | 1’b0 | 4’b1000 |
| SLL | 1’b1 | 4’b0011 |
| SLLV | 1’b0 | 4’b0011 |

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。

1. Branch mux詳細說明：

我們將branch指令分成四個種類，第一為beg類，beg指令為此類；第二為ble類，ble指令為此類；第三為blt類，blt指令為此類；最後一個為bne類，bne、bnez指令為此類的。而圖中第五個mux即是從這四類中為一類出來作為最後是否要branch的訊號，詳細對應如下：

|  |  |  |
| --- | --- | --- |
| BrancgType(選擇線) | 輸出 | 對應的指令 |
| 2’b00 | zero\_o | beq |
| 2’b01 | zero\_o|result\_o[31] | ble |
| 2’b10 | result\_o[31] | blt |
| 2’b11 | zero\_o | bne,bnez |

* zero\_o在bne、bnez的情況下輸出1代表不為0，反之代表為0(在ALU內控制)；其餘狀況則是輸出1代表為0，反之代表不為0。
* ble、blt用result的MSB則是要看src\_1是否src\_2，若小於成立，則result的MSB必為1，故可用這種方法來決定ble、blt是否要branch。

1. JAL實作方法：

JAL因為要將PC+4存回reg31，故我們將圖中第四個mux變成4-1的，多的那個輸入室已經算出來的PC+4，並以原本即為2bit的MemToReg\_o當作選擇線，這樣就可以把PC+4傳回WriteData；同時我們也有修改第一個mux，將它變成3-to-1的，多的輸入為常數31，且須將原本1bit的RegDst\_o改成2bit，此舉是為了WriteRegister能有正確的寫入位址。第四個mux詳細對應如下：

|  |  |
| --- | --- |
| MemToReg\_o(選擇線) | 輸出 |
| 2’b00 | result\_o |
| 2’b01 | MEM\_Read\_data\_o |
| 2’b10 | SE\_data\_o |
| 2’b11 | pc\_plus\_four |

第一個mux詳細對應如下：

|  |  |
| --- | --- |
| RegDst\_o(選擇線，2bit) | 輸出 |
| 2’b00 | instruction\_o[20:16] |
| 2’b01 | instruction\_o[15:11] |
| 2’b10 | 5’d31 |

1. JR實作方法：

JR的實作我們是多加一個mux，多加的這個mux為第八個mux，放的位置是在圖中第七個mux後面，會放在這麼後面是因為Jump是在很後面才決定的，為了避免混亂，放在最後面是最好的選擇，此mux為2-to-1的mux，輸入分別是RSdata\_o及原本沒有JR指令時的下一個PC number，選擇線為Jump\_type，此選擇線只有在JR指令下為1，其餘指令下為0，此選擇線由ALU\_ctrl計算，因為JR指令為R-type的關係且不破壞整體架構的情況下，由ALU\_ctrl計算它是最好的選擇。

此mux詳細對應如下：

|  |  |
| --- | --- |
| Jump\_type(選擇線) | 輸出 |
| 1’b0 | pc\_number\_next |
| 1’b1 | RSdata\_o |

* pc\_number\_next為原本沒有JR指令時的下一個PC number。

**Problems encountered and solutions:**

1. 其實一開始不太知道該怎麼實作JR這個指令，因為它是R-type而且跟其他R-type指令都好像沒有什麼關係，最後看了一下指令的結構以及參考的架構圖，發現可以將RSdata\_o直接接出來送進一個mux做選擇。在實作這個指令時遇到的另外一個問題是那個mux的選擇線該從哪裡來，因為它是R-type指令的關係，故無法在Decoder的部分就產出選擇線的訊號，我們選擇在ALU\_ctrl那部分產生選擇線Jump\_type的訊號。最後一個要注意的地方是，因為JR指令為R-type指令，為了避免寫入垃圾值到reg0，在JR指令下，ALU的輸出強制為0，以此避免此狀況的發生。
2. 在測試的bubble sort那題的測資時，發現就算按了run all結果仍然不對，以一次增加1000ns模擬時間的方式debug，最後發現是testbench的強制結束時間太短了，將結束時間改長一點即可看到正確的結果。

**Lesson learnt (if any):**

1. 四種不同類型branch指令是否要branch的判斷方法。
2. JR指令的實作方法。