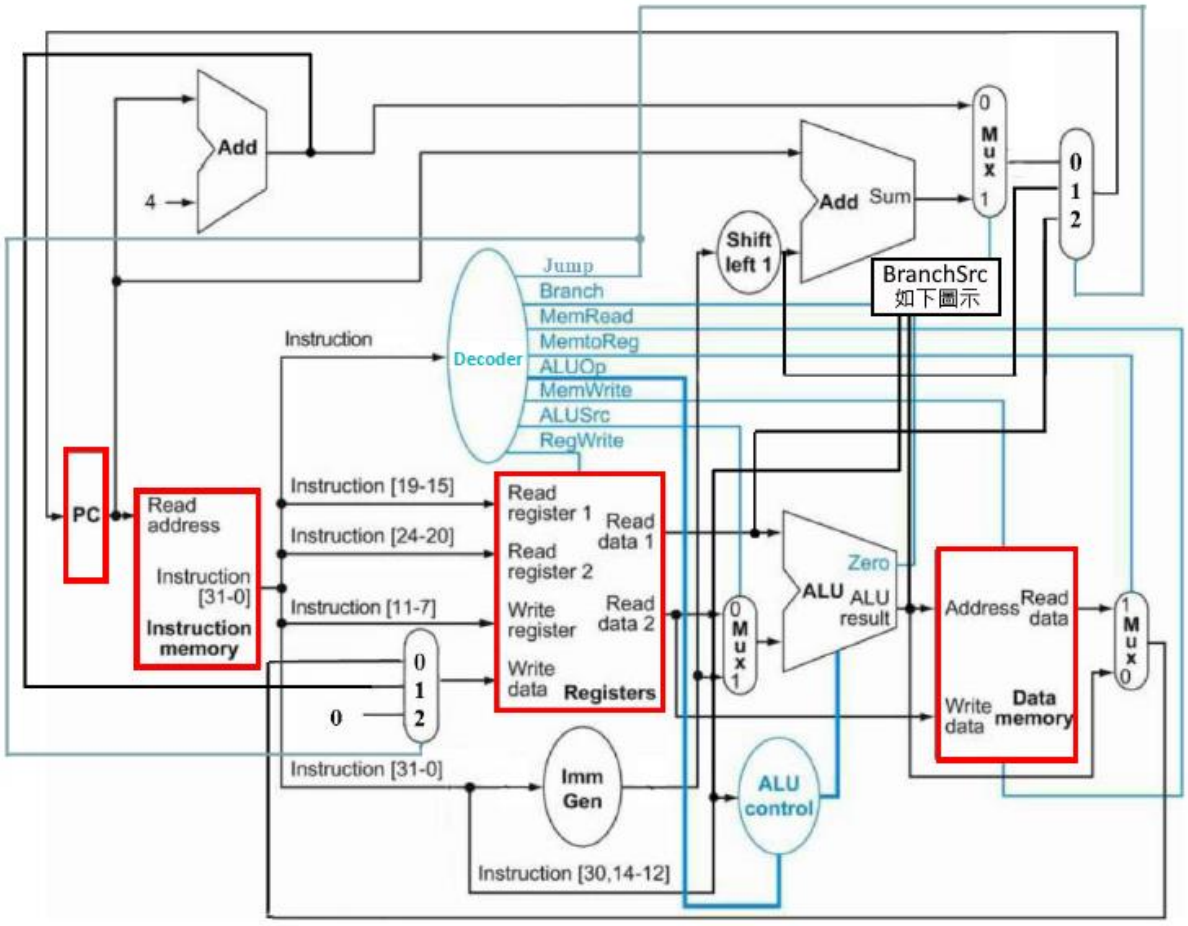
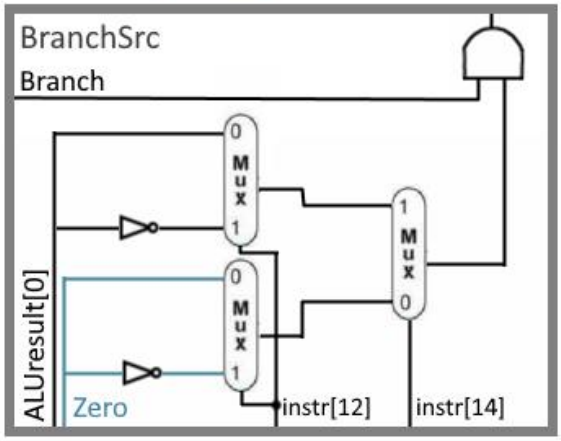
# Computer Organization

**0716214 江岳勳**
**0716222 黃偉傑**

## Architecture diagram:



## Detailed description of the implementation:

有許多 module 都跟 Lab3 的一模一樣，所以這裡只提到有更動過的實作細節

- ALU_Ctrl.v

  因為我們上次都已經把 lw 跟 sw 等指令分配完成了，所以這次要新增處理的只有 blt 跟 bge 而已。blt 和 bge 都是執行 slt 指令，再由外部 top module 來區分兩者。

  因此這裡的 ALU_Ctrl_o 為 4'b0111：

```
(ALUOp==2'b01 && instr[2]==1'b0)?       4'b0110:(       // beq, bne
(ALUOp==2'b01 && instr[2]==1'b1)?       4'b0111:(       // blt, bge
```

- Decoder.v

  多判斷指令是否為 jal，若是的話 Instr_field 為 4。

  查閱真值表，逐一設定指令的 MemtoReg, RegWrite, MemRead, MemWrite, Jump (參見後表)

- MUX_3to1.v

```
assign data_o = (select_i == 2'b00) ? data0_i : ((select_i == 2'b01 ? data1_i : data2_i));
```

- Simple_Single_CPU.v

  將多新增的 module 彼此之間的 wire 接上：Data_Memory, Mux_MemToRegSrc, Mux_WBSrc, Mux_PCJumpSrc

  處理新的 Branch 指令：原本的 PC_Src 只有考慮 beq 和 bne(只處理 zero)，新增了 blt 和 bge 後，查看真值表可以發現這四個可以先用 instr[14]分類，再進行處理。所以新的 PC_Src 為：

```
assign PC_SRC = (instr[14] == 1'b1) ?
            ((instr[12] == 1'b0 ? ALUresult[0] : ~ALUresult[0]) & Branch):
            ((instr[12] == 1'b0 ? ZERO : ~ZERO) & Branch);
```

  下面是我們建立的真值表：

ALU_Ctrl (不會進 ALU 的 instruction : jal, jalr)

| instruction | ALUOp[1 : 0] | instr [3 : 0] | ALU action | ALU_Ctrl_o |
|---|---|---|---|---|
| lw | 00 | 010X | add | 0010 |
| sw | 00 | 010X | add | 0010 |
| add | 10 | 0000 | add | 0010 |

| | | | | |
|---|---|---|---|---|
| sub | 10 | 0001 | sub | 0110 |
| and | 10 | 1110 | and | 0000 |
| or | 10 | 1100 | or | 0001 |
| xor | 10 | 1000 | xor | 0011 |
| slt | 10 | 0100 | slt | 0111 |
| sll | 10 | 0010 | sll | 0100 |
| sra | 10 | 1011 | sra | 0101 |
| beq | 01 | 000X | sub | 0110 |
| bne | 01 | 001X | sub | 0110 |
| blt | 01 | 100X | slt | 0111 |
| bge | 01 | 101X | slt | 0111 |
| addi | 11 | 000X | add | 0010 |
| andi | 11 | 111X | and | 0000 |
| ori | 11 | 110X | or | 0001 |
| xori | 11 | 100X | xor | 0011 |
| slti | 11 | 010X | slt | 0111 |
| slli | 11 | 0010 | sll | 0100 |
| srli | 11 | 1010 | srli | 1111 |

Decoder

| Instruction | Field(Opcode) | M2R | RegW | MemR | MemW | Jump[2] | ALUSrc | Branch | ALUOp[2] |
|---|---|---|---|---|---|---|---|---|---|
| R-type | 0 | 0 | 1 | 0 | 0 | 00 | 0 | 0 | 10 |
| LW | 1(0000011) | 1 | 1 | 1 | 0 | 00 | 1 | 0 | 00 |
| JALR | 1(1100111) | 0 | 1 | 0 | 0 | 10 | 1 | 0 | 11 |
| I-type | 1(else) | 0 | 1 | 0 | 0 | 00 | 1 | 0 | 11 |
| S-type | 2 | 0 | 0 | 0 | 1 | 00 | 1 | 0 | 00 |
| B-type | 3 | 0 | 0 | 0 | 0 | 00 | 0 | 1 | 01 |
| JAL | 4 | 0 | 1 | 0 | 0 | 01 | 0 | 0 | 00 |

## Implementation results:

CO_test_data1.txt：

```
# PC =         76
# Data Memory =          0,          0,          0,          0,          0,          0,          0,          0
# Data Memory =          0,          0,          0,          0,          0,          0,          0,          0
# Data Memory =          0,          0,          0,          0,          0,          0,          0,          0
# Data Memory =          0,          0,          0,          0,          0,          1,          2,          3
# Registers
# R0 =          0, R1 =         16, R2 =        128, R3 =          0, R4 =          0, R5 =          4, R6 =          5, R7 =          6
# R8 =          1, R9 =          2, R10 =          3, R11 =          0, R12 =          0, R13 =          0, R14 =          0, R15 =          0
# R16 =          0, R17 =          0, R18 =          0, R19 =          0, R20 =          0, R21 =          0, R22 =          0, R23 =          0
# R24 =          0, R25 =          0, R26 =          0, R27 =          0, R28 =          0, R29 =          0, R30 =          0, R31 =          0
```

CO_test_data2.txt：

```
# PC =         44
# Data Memory =          0,          0,          0,          0,          0,          0,          0,          0
# Data Memory =          0,          0,          0,          0,          0,          0,          0,          0
# Data Memory =          0,          0,          0,          0,          0,          0,          0,          0
# Data Memory =          0,          0,          0,          0,          0,          0,          0,         28
# Registers
# R0 =          0, R1 =         16, R2 =        128, R3 =          0, R4 =          0, R5 =          1, R6 =          0, R7 =         28
# R8 =          0, R9 =          0, R10 =          0, R11 =          0, R12 =          0, R13 =          0, R14 =          0, R15 =          0
# R16 =          0, R17 =          0, R18 =          0, R19 =          0, R20 =          0, R21 =          0, R22 =          0, R23 =          0
# R24 =          0, R25 =          0, R26 =          0, R27 =          0, R28 =          0, R29 =          0, R30 =          0, R31 =          0
```

## Problems encountered and solutions:

1. 在做 blt 和 bge 的時候，一開始不是很清楚要怎麼讓 Branch 的 PC_Src 知道現在示做 blt/bge/beq/bne
   - 將真值表建出來之後，很輕鬆地就可以看出當 instr[14]為 1 時是 blt/bge，了解之後再改動 PC_src 便沒遇到什麼困難。
2. 在測試時 sw 怎麼樣都不會動，解讀了一下之後發現 sw 的目標為位置為-12 的記憶體，想了很久怎麼想都覺得不合理
   - 東看西看找很久才發現 Lab3 和 Lab4 的 Reg_File 不一樣，Lab4 的 Reg_Filer2 的初始值為 128，因為我們是直接使用 Lab3 的結果下去時做 Lab4 的需求，所以沒改到這裡。

## Comment:

0716214 江岳勳：

我覺得這次的作業其實非常簡單，因為我們上次都已經把 DataMemory 有關的功能都寫好了，只需要將線接上，再讓 ALU_Ctrl 能夠多判斷 blt 和 bge 就結束了，是一份相對輕鬆的作業。

0716222 黃偉傑：

有了上次 Lab3 協作與建構 Simplfied CPU 的經驗，我認為對降低這次 Lab 的難度有顯著的幫助。感覺得出來上次跟這次 Lab 應該原本要合在一起，只是這樣一來會讓 loading 更繁重，感謝助教的貼心哈哈。

經過這次經驗，讓我也對於 Single Cycle 的架構更加了解了，讓我覺得挑戰 pipelined CPU 或許也不是件難事，希望我不要被打臉太嚴重。t