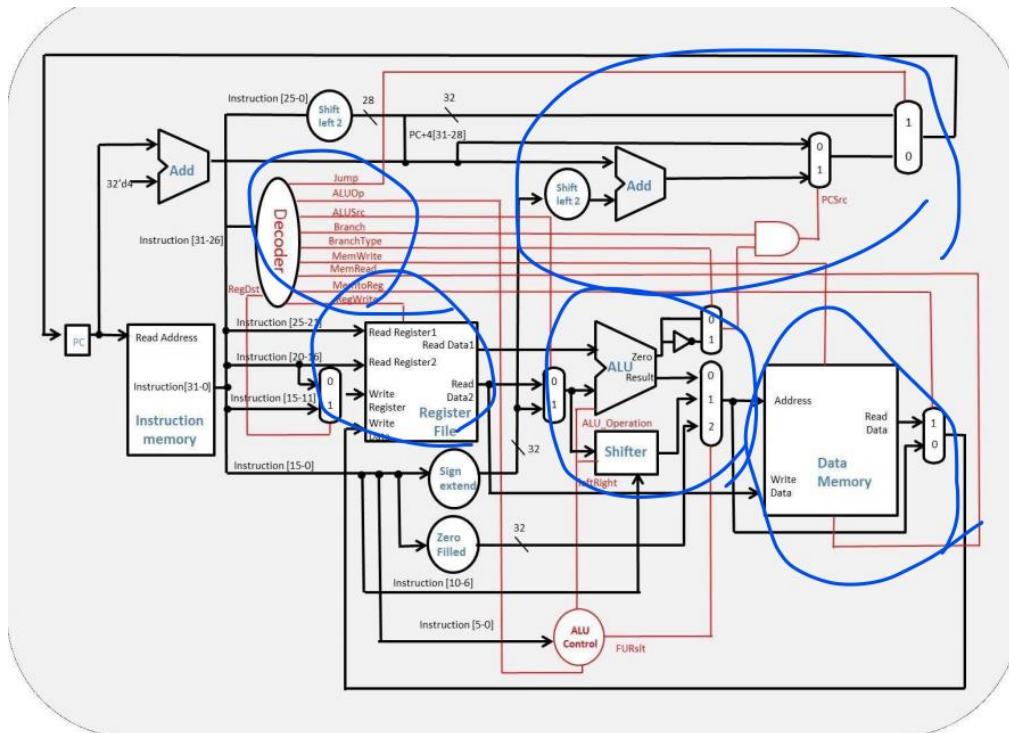


LAB3

1. Requirement

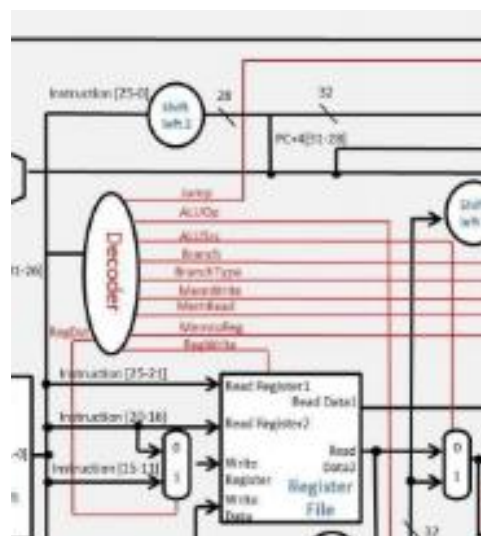
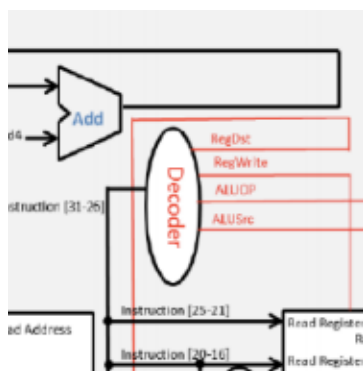
0613304 工工系 10 高宗霖

Set A. 完成基本指令



根據題目要求，我們需要修改的地方為以上畫圈的部分。雖然看似只是多了 data_memory 的部分，但 Decoder、許多 multiplexer、jump 就多了許多要該改的部分。

首先看 Decoder



其中 Decoder.V 中，我們要根據不同的 Opcode 來做相對應 control Line 的設定。

```
File Diagnostics Desktop/Labs/reference_code/Decoder.v - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help

+ Program_Counter x ALU_Orb x Mult32 x Mo... x Mo...

25
26
27
28 //main function
29 assign RegWrite_o = 0;
30
31 (instr_op_1 == 0) ? 0x000000 : 0;
32 (instr_op_1 == 0) ? 0x000000 : 0;
33 (instr_op_1 == 0) ? 0x000000 : 0;
34
35 always @(*)
36 begin
37     case(instr_op_1)
38         0: //NOP
39             begin
40                 temp1 <= 0x0000;
41                 temp2 <= 0x01;
42                 temp3 <= 0x00;
43             end
44         1: //ADD
45             begin
46                 temp1 <= 0x0000;
47                 temp2 <= 0x00;
48                 temp3 <= 0x00;
49             end
50         2: //SUB
51             begin
52                 temp1 <= 0x0000;
53                 temp2 <= 0x00;
54                 temp3 <= 0x00;
55             end
56         3: //MUL
57             begin
58                 temp1 <= 0x0000;
59                 temp2 <= 0x00;
60                 temp3 <= 0x00;
61             end
62         4: //DIV
63             begin
64                 temp1 <= 0x0000;
65                 temp2 <= 0x00;
66                 temp3 <= 0x00;
67             end
68         5: //AND
69             begin
70                 temp1 <= 0x0000;
71                 temp2 <= 0x00;
72                 temp3 <= 0x00;
73             end
74         6: //OR
75             begin
76                 temp1 <= 0x0000;
77                 temp2 <= 0x00;
78                 temp3 <= 0x00;
79             end
80         7: //XOR
81             begin
82                 temp1 <= 0x0000;
83                 temp2 <= 0x00;
84                 temp3 <= 0x00;
85             end
86         8: //SHL
87             begin
88                 temp1 <= 0x0000;
89                 temp2 <= 0x00;
90                 temp3 <= 0x00;
91             end
92         9: //SHR
93             begin
94                 temp1 <= 0x0000;
95                 temp2 <= 0x00;
96                 temp3 <= 0x00;
97             end
98         10: //ROL
99             begin
100                 temp1 <= 0x0000;
101                 temp2 <= 0x00;
102                 temp3 <= 0x00;
103             end
104         11: //ROR
105             begin
106                 temp1 <= 0x0000;
107                 temp2 <= 0x00;
108                 temp3 <= 0x00;
109             end
110         12: //LUI
111             begin
112                 temp1 <= 0x0000;
113                 temp2 <= 0x00;
114                 temp3 <= 0x00;
115             end
116         13: //SLL
117             begin
118                 temp1 <= 0x0000;
119                 temp2 <= 0x00;
120                 temp3 <= 0x00;
121             end
122         14: //SRL
123             begin
124                 temp1 <= 0x0000;
125                 temp2 <= 0x00;
126                 temp3 <= 0x00;
127             end
128         15: //SLLV
129             begin
130                 temp1 <= 0x0000;
131                 temp2 <= 0x00;
132                 temp3 <= 0x00;
133             end
134         16: //SRLV
135             begin
136                 temp1 <= 0x0000;
137                 temp2 <= 0x00;
138                 temp3 <= 0x00;
139             end
140         17: //ANDI
141             begin
142                 temp1 <= 0x0000;
143                 temp2 <= 0x00;
144                 temp3 <= 0x00;
145             end
146         18: //ANDI
147             begin
148                 temp1 <= 0x0000;
149                 temp2 <= 0x00;
150                 temp3 <= 0x00;
151             end
152         19: //ANDI
153             begin
154                 temp1 <= 0x0000;
155                 temp2 <= 0x00;
156                 temp3 <= 0x00;
157             end
158         20: //ANDI
159             begin
160                 temp1 <= 0x0000;
161                 temp2 <= 0x00;
162                 temp3 <= 0x00;
163             end
164         21: //ANDI
165             begin
166                 temp1 <= 0x0000;
167                 temp2 <= 0x00;
168                 temp3 <= 0x00;
169             end
170         22: //ANDI
171             begin
172                 temp1 <= 0x0000;
173                 temp2 <= 0x00;
174                 temp3 <= 0x00;
175             end
176         23: //ANDI
177             begin
178                 temp1 <= 0x0000;
179                 temp2 <= 0x00;
180                 temp3 <= 0x00;
181             end
182         24: //ANDI
183             begin
184                 temp1 <= 0x0000;
185                 temp2 <= 0x00;
186                 temp3 <= 0x00;
187             end
188         25: //ANDI
189             begin
190                 temp1 <= 0x0000;
191                 temp2 <= 0x00;
192                 temp3 <= 0x00;
193             end
194         26: //ANDI
195             begin
196                 temp1 <= 0x0000;
197                 temp2 <= 0x00;
198                 temp3 <= 0x00;
199             end
200         27: //ANDI
201             begin
202                 temp1 <= 0x0000;
203                 temp2 <= 0x00;
204                 temp3 <= 0x00;
205             end
206         28: //ANDI
207             begin
208                 temp1 <= 0x0000;
209                 temp2 <= 0x00;
210                 temp3 <= 0x00;
211             end
212         29: //ANDI
213             begin
214                 temp1 <= 0x0000;
215                 temp2 <= 0x00;
216                 temp3 <= 0x00;
217             end
218         30: //ANDI
219             begin
220                 temp1 <= 0x0000;
221                 temp2 <= 0x00;
222                 temp3 <= 0x00;
223             end
224         31: //ANDI
225             begin
226                 temp1 <= 0x0000;
227                 temp2 <= 0x00;
228                 temp3 <= 0x00;
229             end
230         32: //ANDI
231             begin
232                 temp1 <= 0x0000;
233                 temp2 <= 0x00;
234                 temp3 <= 0x00;
235             end
236         33: //ANDI
237             begin
238                 temp1 <= 0x0000;
239                 temp2 <= 0x00;
240                 temp3 <= 0x00;
241             end
242         34: //ANDI
243             begin
244                 temp1 <= 0x0000;
245                 temp2 <= 0x00;
246                 temp3 <= 0x00;
247             end
248         35: //ANDI
249             begin
250                 temp1 <= 0x0000;
251                 temp2 <= 0x00;
252                 temp3 <= 0x00;
253             end
254         36: //ANDI
255             begin
256                 temp1 <= 0x0000;
257                 temp2 <= 0x00;
258                 temp3 <= 0x00;
259             end
260         37: //ANDI
261             begin
262                 temp1 <= 0x0000;
263                 temp2 <= 0x00;
264                 temp3 <= 0x00;
265             end
266         38: //ANDI
267             begin
268                 temp1 <= 0x0000;
269                 temp2 <= 0x00;
270                 temp3 <= 0x00;
271             end
272         39: //ANDI
273             begin
274                 temp1 <= 0x0000;
275                 temp2 <= 0x00;
276                 temp3 <= 0x00;
277             end
278         40: //ANDI
279             begin
280                 temp1 <= 0x0000;
281                 temp2 <= 0x00;
282                 temp3 <= 0x00;
283             end
284         41: //ANDI
285             begin
286                 temp1 <= 0x0000;
287                 temp2 <= 0x00;
288                 temp3 <= 0x00;
289             end
290         42: //ANDI
291             begin
292                 temp1 <= 0x0000;
293                 temp2 <= 0x00;
294                 temp3 <= 0x00;
295             end
296         43: //ANDI
297             begin
298                 temp1 <= 0x0000;
299                 temp2 <= 0x00;
300                 temp3 <= 0x00;
301             end
302         44: //ANDI
303             begin
304                 temp1 <= 0x0000;
305                 temp2 <= 0x00;
306                 temp3 <= 0x00;
307             end
308         45: //ANDI
309             begin
310                 temp1 <= 0x0000;
311                 temp2 <= 0x00;
312                 temp3 <= 0x00;
313             end
314         46: //ANDI
315             begin
316                 temp1 <= 0x0000;
317                 temp2 <= 0x00;
318                 temp3 <= 0x00;
319             end
320         47: //ANDI
321             begin
322                 temp1 <= 0x0000;
323                 temp2 <= 0x00;
324                 temp3 <= 0x00;
325             end
326         48: //ANDI
327             begin
328                 temp1 <= 0x0000;
329                 temp2 <= 0x00;
330                 temp3 <= 0x00;
331             end
332         49: //ANDI
333             begin
334                 temp1 <= 0x0000;
335                 temp2 <= 0x00;
336                 temp3 <= 0x00;
337             end
338         50: //ANDI
339             begin
340                 temp1 <= 0x0000;
341                 temp2 <= 0x00;
342                 temp3 <= 0x00;
343             end
344         51: //ANDI
345             begin
346                 temp1 <= 0x00
```

但後來查講義的時後意外發現其實可以用 **gateway** 的方式比較輕鬆好看。所以我也試著把它打出來了(結果也正確開動~)

而以上實作為查表，並作對應，在此不多做說明

比照原本的範例，現在題目要求只多了 lw、sw、branch 等要求。因此，將其對應的 ALUOP 寫上就完成了。

```

12 wire [2:1:0] FURslt_o;
13
14 //Main function
15 assign ALU_operation_o = ((ALUop_1,funcnt_1) == 9'b010010011 || ALUop_1 == 3'b000) ? 4'b0010 : //add add1
16 ((ALUop_1,funcnt_1) == 9'b010010001 || ALUop_1 == 3'b001) ? 4'b0110 : //sub
17 ((ALUop_1,funcnt_1) == 9'b010010100) ? 4'b0000 : //and
18 ((ALUop_1,funcnt_1) == 9'b010010110) ? 4'b0001 : //or
19 ((ALUop_1,funcnt_1) == 9'b010010101) ? 4'b1100 : //nor
20 ((ALUop_1,funcnt_1) == 9'b010110000) ? 4'b0111 : //slt
21 ((ALUop_1,funcnt_1) == 9'b010000000) ? 4'b0000 : //sll
22 ((ALUop_1,funcnt_1) == 9'b010000010) ? 4'b0001 : //sr1
23 ((ALUop_1,funcnt_1) == 9'b010000110) ? 4'b0010 : //slrv
24 ((ALUop_1,funcnt_1) == 9'b010000100) ? 4'b0011 : //sr1
25 ((ALUop_1 == 3'b000)
26 ALUop_1 == 3'b001)
27 ALUop_1 == 3'b110) ? 4'b0010 : // 1n / sw
28 ALUop_1 == 3'b001) ? 4'b0110 : //branch
29 assign FURslt_o = ((ALUop_1,funcnt_1) == 9'b010010011 || ALUop_1 == 3'b011) ? 3'b00 : //bne others
30 ((ALUop_1,funcnt_1) == 9'b010010001) ? 2'b00 : //sub
31 ((ALUop_1,funcnt_1) == 9'b010010100) ? 2'b00 : //and
32 ((ALUop_1,funcnt_1) == 9'b010010110) ? 2'b00 : //or
33 ((ALUop_1,funcnt_1) == 9'b010010101) ? 2'b00 : //nor
34 ((ALUop_1,funcnt_1) == 9'b010110000) ? 2'b00 : //slt
35 ((ALUop_1,funcnt_1) == 9'b010000000) ? 2'b01 : //sll
36 ((ALUop_1,funcnt_1) == 9'b010000010) ? 2'b01 : //sr1
37 ((ALUop_1,funcnt_1) == 9'b010000110) ? 2'b01 : //slrv
38 ((ALUop_1,funcnt_1) == 9'b010000100) ? 2'b01 : 2'b00 : //sr1v others
39
40 endmodule

```

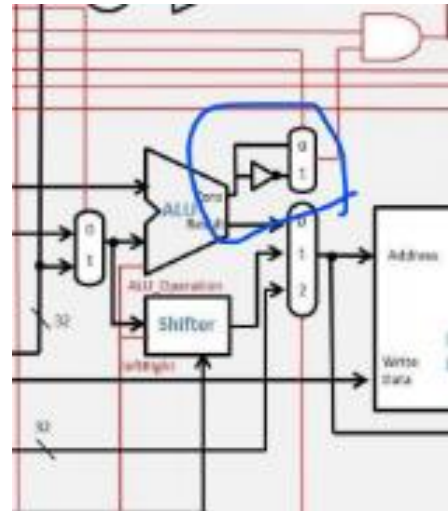
其中，Addi 的 ALUop 有改，要改為 1' b100 才有符合題目的設定。

接下來，因為我們有 Bne 的指令。因此要多一個 multiplexer 接在 ALU Zero 後面。

此 Multiplexer 的控制線為 Branch_Type

當指令為 Bne 時 Branch_Type 設為 1，其餘指令皆設為 0。

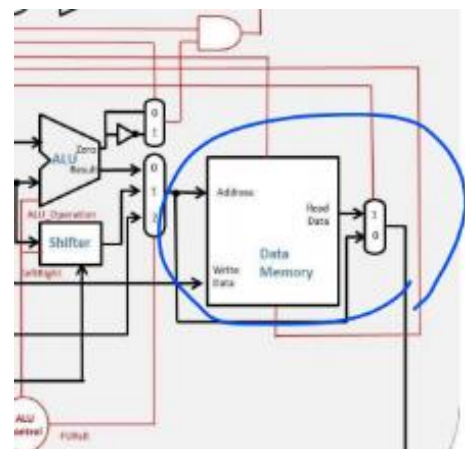
因此，當 ALU 比對結果為不一樣，Zero 會跑 0，接 NOT 後變為 1 接在 multiplexer 的 1 端。



接下來放入，data_Memory。該 module 不需要更改，只需要在 Simple_Single_CPU 內做對應的設定即可。

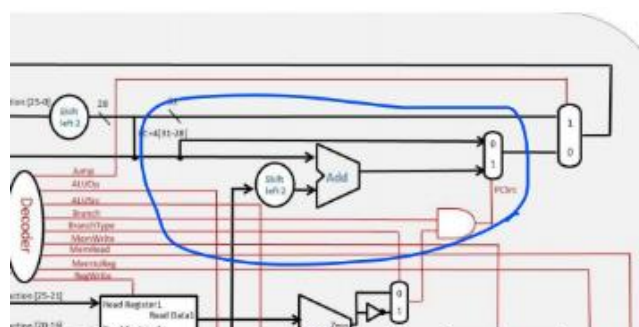
另外，我們要再加一個 multiplexer 控制輸出資料來源

最後，輸出的 wire 接回 register，基本上一個 cycle 就完成了。



接下來為 Branch

Branch 為 PC+4 再加上指令後面的 16bit 位址，因此我們要多加一個 Adder。



將 16bit sign extension 的結果與 PC+4 放入新增的加法器，output 即為 branch 的位址。

我們必須再新增一個 multiplexer 做為控制 Branch 或 PC+4 的路徑。而此 multiplexer 的控制線為下面 AND GATE 的結果。

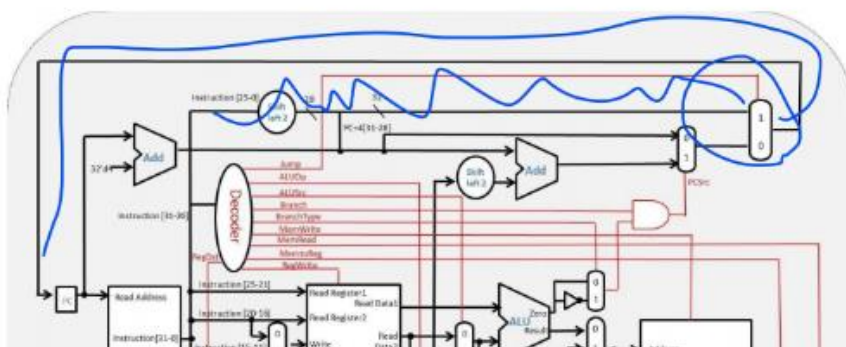
```
137
138   assign PCsrc = BRANCH & Branch_col;
139
140
141   Mux2to1 #(.size(32)) GO_branch(
142     .data0_1(add_pc),
143     .data1_1(add_branch),
144     .select_1(PCsrc),
145     .data_o(select_branch)
146   );
147
```

而該 multiplexer 的控制線輸出，我用 assign 的方式直接將 Decoder 輸出的 Branch 與先前 multiplexer 的輸出做 AND。

最後一步，將 Jump 接上

Jump 指令為 PC+4 前 4bit 接上 instruction 26bit left shift 2 bit

用一個 multiplexer 將 Jump 與剛剛 Branch multiplexer 輸出再做一次路徑控制，control selection 用 Decoder 輸出的 JUMP。將最後的輸出接回 PC，即大功告成。



Set B 增加 JAL、JR 指令

因為 JAL 需要將 PC+4 的 addr 再次存入 Register31 內。因此在線路設計需要稍微更改 Set A 的內容。

JAL Jump 的部分一樣不須多做設定，但我們需要拉一個線路，將 PC+4 拉到 Mem_to_Reg 這個 multiplexer 上面，使原本的分類器變為 3 to 1 multiplexer。然後將相對應的 control line 設定好成寫入模式(尤其是 RegisterWrite 一定要記得設成 1)。

JR 比較麻煩一點點，因為它要讀 Reg31 的位址並 Jump。

順應的 machine code，rs_i、rt_i 輸出 rs_o、rt_o。因為 rt_o 為 0(zero)，因此經過 ALU 相加後即為要跳的地址。接上 multiplexer (3 to 1)即可完成。

這裡要注意的是，RegWrite 要特別注意，不能把它當 don't care，不然它會寫東西進去 Reg 內(我一開始沒注意 Debug 了好久才抓到)

Set C 增加 BLT、BNEZ、BGEZ 指令

以上這三項指令 control line 與 Bne 差不多，需要更改的是在 ALU 內相對應的輸出。

BLT: 我在 ALU 內設了一個 sign bit，監看 result[31]是否為 1。若為 1 跳，反之亦然。

BNEZ 與 BNE 指令一樣，因此不需更改什麼。

BGEZ: 作法與 BLT 差不多，在 ALU 內設一個 sign bit 看 ALUsource1[31]是否為 1，為 1 不跳，反之亦然。

最後在 multiplexer 的部分我新增了一個 4 to 1 的 multiplexer，分別控制(Beq、Bne、Blt、Bgez)

```

16
17
18 module Mux4to1( data0_i, data1_i, data2_i, data3_i, select_i, data_o );
19
20 parameter size = 0;
21
22 //I/O ports
23 input wire [size-1:0] data0_i;
24 input wire [size-1:0] data1_i;
25 input wire [size-1:0] data2_i;
26 input wire [size-1:0] data3_i;
27 input wire [2-1:0] select_i;
28 output wire [size-1:0] data_o;
29
30 //Main function
31 assign data_o = (select_i[1] & select_i[0]) ? data3_i :
32                (select_i[1]) ? data2_i :
33                select_i[0] ? data1_i : data0_i;
34 endmodule
35

```

我把此 module 放在 Mux3to1.V 中與 3to1MUX 並行。

結束!!!

最後心得:

這次作業看起來很難，我一開始也看檔案看到眼花。但仔細把每一份 code 掃過一遍竟然發現該做的都被寫完了。我只需要甜甜的把 control line 設定就好(感謝助教 carry)

Set B，Set C 的部分有些筱困難，在想怎麼接的時候都因為要大幅度更改 Set A 的內容搞得我有點不想做(但最後還是硬著頭皮趕在 Deadline 前兩小時做完)，呼~好險有趕完。