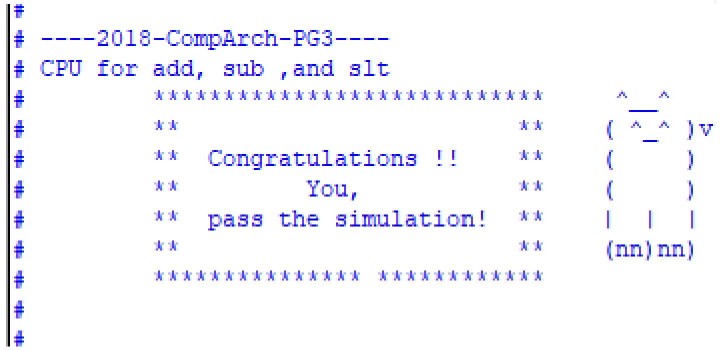
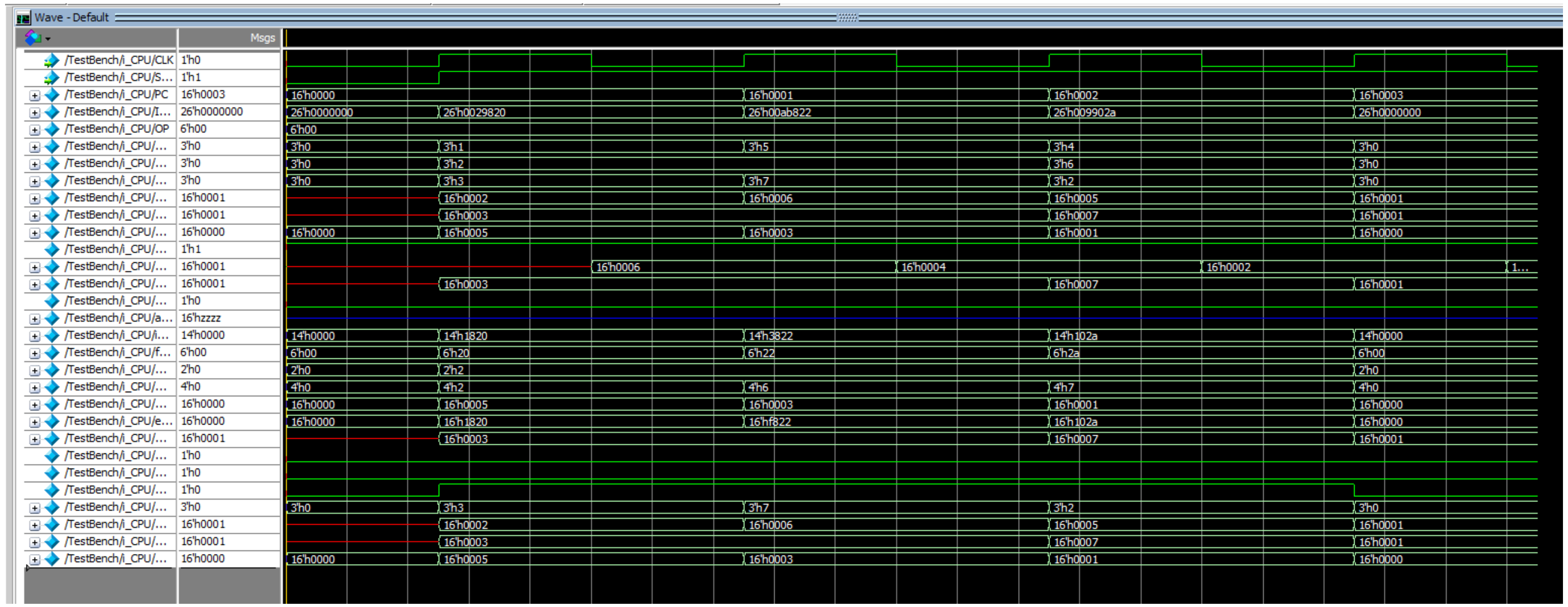
Programming Assignment 4
(due：01/04 13：00)

作業説明

1. 實作一個Single CPU可以執行MIPS ISA的add，sub，slt的指令
2. 用測試資料驗證CPU的正確性並將通過結果圖截圖（如右圖）
3. 需要用波形圖去驗證結果是否正確（如下圖）



```
#
# ----2018-CompArch-PG3----
# CPU for add, sub ,and slt
#           ****************************      ^__^
#           **                        **     ( ^_^ )v
#           **  Congratulations !!    **     (      )
#           **          You,          **     (      )
#           **  pass the simulation!  **     |  |   |
#           **                        **     (nn)nn)
#           *************** ************
#
#
```



Wave - Default

```
                              Msgs
/TestBench/i_CPU/CLK          1'h0
/TestBench/i_CPU/S...         1'h1
/TestBench/i_CPU/PC           16'h0003       16'h0000   | 16'h0001   | 16'h0002   | 16'h0003
/TestBench/i_CPU/I...         26'h0000000    26'h0000000 | 26'h0029820 | 26'h00ab822 | 26'h009902a | 26'h0000000
/TestBench/i_CPU/OP           6'h00          6'h00
/TestBench/i_CPU/...          3'h0           3'h0 | 3'h1 | 3'h5 | 3'h4 | 3'h0
/TestBench/i_CPU/...          3'h0           3'h0 | 3'h2 | 3'h6 | 3'h0
/TestBench/i_CPU/...          3'h0           3'h0 | 3'h3 | 3'h7 | 3'h2 | 3'h0
/TestBench/i_CPU/...          16'h0001       16'h0002 | 16'h0006 | 16'h0005 | 16'h0001
/TestBench/i_CPU/...          16'h0001       16'h0003 | 16'h0007 | 16'h0001
/TestBench/i_CPU/...          16'h0000       16'h0000 | 16'h0005 | 16'h0003 | 16'h0001 | 16'h0000
/TestBench/i_CPU/...          1'h1
/TestBench/i_CPU/...          16'h0001       16'h0006 | 16'h0004 | 16'h0002 | 1...
/TestBench/i_CPU/...          16'h0001       16'h0003 | 16'h0007 | 16'h0001
/TestBench/i_CPU/...          1'h0
/TestBench/i_CPU/a...         16'hzzzz
/TestBench/i_CPU/i...         14'h0000       14'h0000 | 14'h1820 | 14'h3822 | 14'h102a | 14'h0000
/TestBench/i_CPU/f...         6'h00          6'h00 | 6'h20 | 6'h22 | 6'h2a | 6'h00
/TestBench/i_CPU/...          2'h0           2'h0 | 2'h2 | 2'h0
/TestBench/i_CPU/...          4'h0           4'h0 | 4'h2 | 4'h6 | 4'h7 | 4'h0
/TestBench/i_CPU/...          16'h0000       16'h0000 | 16'h0005 | 16'h0003 | 16'h0001 | 16'h0000
/TestBench/i_CPU/e...         16'h0000       16'h0000 | 16'h1820 | 16'hf822 | 16'h102a | 16'h0000
/TestBench/i_CPU/...          16'h0001       16'h0003 | 16'h0007 | 16'h0001
/TestBench/i_CPU/...          1'h0
/TestBench/i_CPU/...          1'h0
/TestBench/i_CPU/...          1'h0
/TestBench/i_CPU/...          3'h0           3'h0 | 3'h3 | 3'h7 | 3'h2 | 3'h0
/TestBench/i_CPU/...          16'h0001       16'h0002 | 16'h0006 | 16'h0005 | 16'h0001
/TestBench/i_CPU/...          16'h0001       16'h0003 | 16'h0007 | 16'h0001
/TestBench/i_CPU/...          16'h0000       16'h0000 | 16'h0005 | 16'h0003 | 16'h0001 | 16'h0000
```

指令格式

Instruction format:

instr	op [25:20]	rs [19:17]	rt [16:14]	rd[13:11]	Shamt[10:6]	Func[5:0]
add	6’b000000	SSS	TTT	DDD	5’b00000	6’b100000
sub	6’0000000	SSS	TTT	DDD	5’b00000	6’b100010
slt	6’b000000	SSS	TTT	DDD	5’b00000	6’b101010

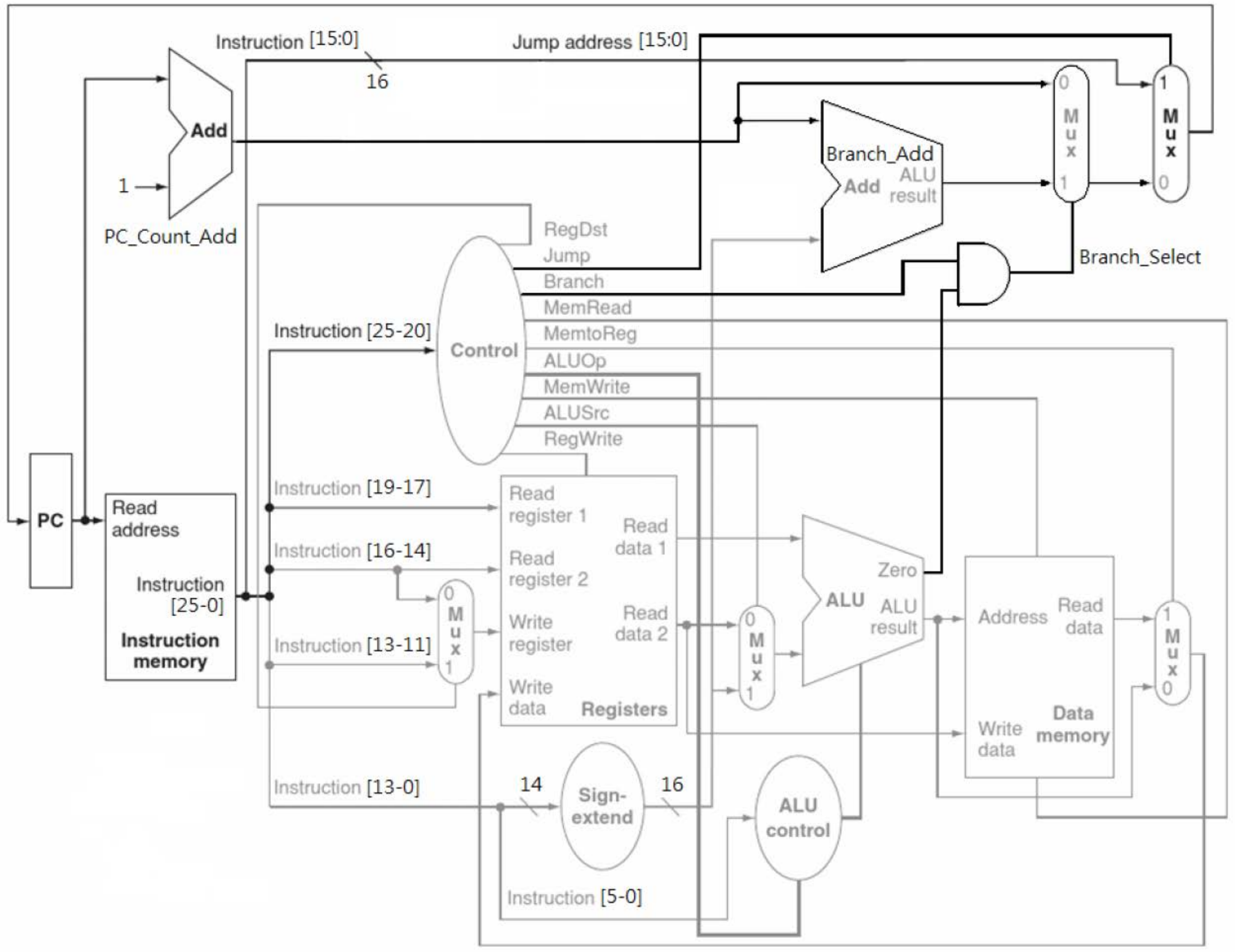
add: do R[SSS]+R[TTT] and store in R[DDD]

sub: do R[SSS]-R[TTT] and store in R[DDD]

slt: do if(R[SSS]<R[TTT]) then R[DDD]=1 else R[DDD]=0

參考CPU圖

reference graph



Module介紹

TestBench module（不需要修改）

TestBench不是CPU的一部分，為CPU的測資，並幫助同學檢查輸出資料是否符合作業要求。

- TestBench有以下主要功能
  - 輸出周期性時脈CLK訊號給CPU
  - 設定Reg（Register）與DM （Data memory）的初始值
  - 將每次clock cycle所更變的value寫到對應的Reg（ Register ）與DM（Data memory）
  - 比對每個clock cycle的值是否正確

CPU module（需要修改）

以下是線路說明

- *Instr* represents the instruction to be executed in this cycle.
- *OP* represents the OP column of the instruction.
- *Reg_WE* represents the signal whether the data should be wrote in Reg.
- *RS_ID* represents RS column in MIPS instruction.
- *RT_ID* represents RT column in MIPS instruction.
- *Reg_RData1* represents the data read from Reg.
- *Reg_RData2* represents the data read from Reg.
- *Reg_WData* represents the data ready to be wrote in Reg.
- *DM_RData* represents the data read from DM.
- *DM_WData* represents the data ready to be wrote in DM.
- DM_WE represents the signal whether the data should be wrote in DM.
- *address* represents the value out of Adder.
- *immediate_in* represents the immediate value read from instruction.
- *func* represents the Func in MIPS instruction.
- *ALU_OP* represent the control signal from Decoder module to ALU_ctrl module.

CPU module（需要修改）

以下是線路說明

- *ALU_CTRL* is the output signal of the ALU_ctrl module and the input signal of the ALU module in order to tell the ALU module which operation to do.
- *ALU_result* represent the result of the ALU module.
- *ALU_src* is used to choose which data should be the input of ALU(the sign_extend_module output or Reg_RData2).
- Mux_to_ALU is the MUX_2_to_1 module output. It is used to selected output from the sign_extend_module output or Reg_RData2 according to the input you provide.
- MEM_to_REG: If this signal is one then the data is from memory to register; if is zero, data fromALU will be written to register.
- REG_Dst is used to choose which register ID should be written(RT or RD) .
- REG_W_ID is the register ID that should be written.
- ALU_src1, ALU_src2 are the inputs of ALU.
- Mux_Mem_to_reg_out are the MUX output which should be write to register .

CPU module（需要修改）

需修改事項：

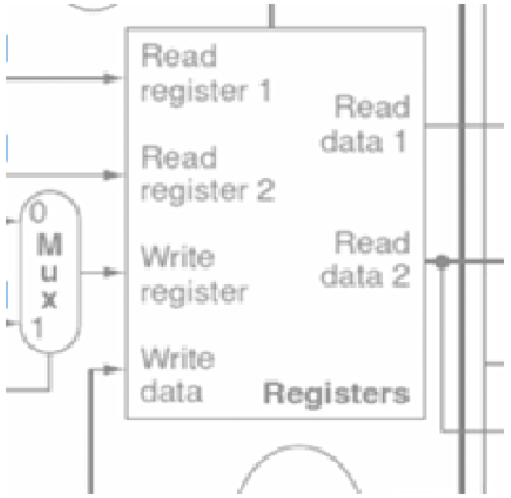
需要檢查並把剩下沒接上的線接上，以assign實作。

Reg module （需要修改）

Reg指的是Register module，他會根據input signals讀與寫入資料，線路的名稱是參照CPU module

需修改事項：

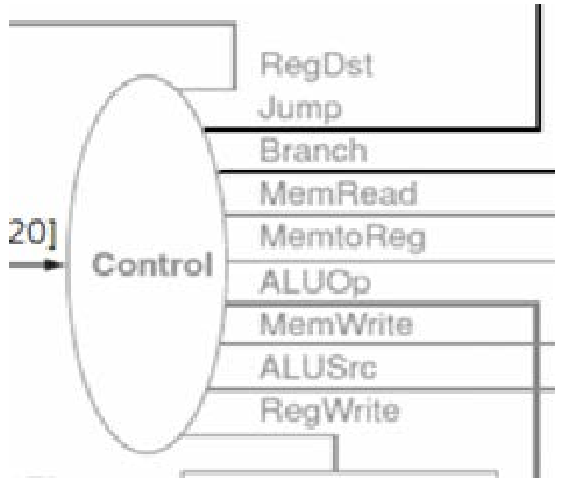
- 修改需要寫入的register的ID與輸出之Read data1,2資料
- 另外也需修改當Reg_WE signal 為1 or 0時register寫入的資料



Decoder module （需要修改）

Decoder module 依據輸入的OP 解碼訊號，並輸出解碼後的訊號，送至Reg或DM或各個組織結構使其判斷指令應該read或write。

例如6’b100011，可知此instruction 為R-type， Decoder 需要設定Reg_WE=1，因爲這個OP code 代表data會被寫進Reg。



Decoder module （需要修改）

Hint:

- ALU_OP 用來區隔instruction為R-type(add sub slt) 或I-type instruction (load store)
- MEM_to_REG 表示區隔data是從memory或ALU輸入
- REG_Dst 是用來區隔需要被寫入的register ID （RT或RD）
- ALU_src 用來選擇輸入ALU 的訊號線（sign_extend ouput 或 Reg_RData2）

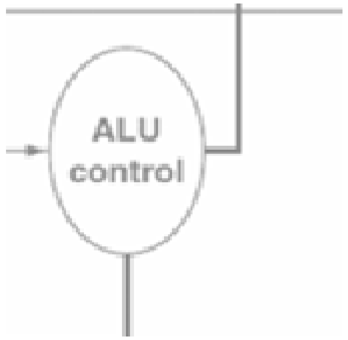
需修改事項：

依據輸入的funct與OP去設定輸出的control signal

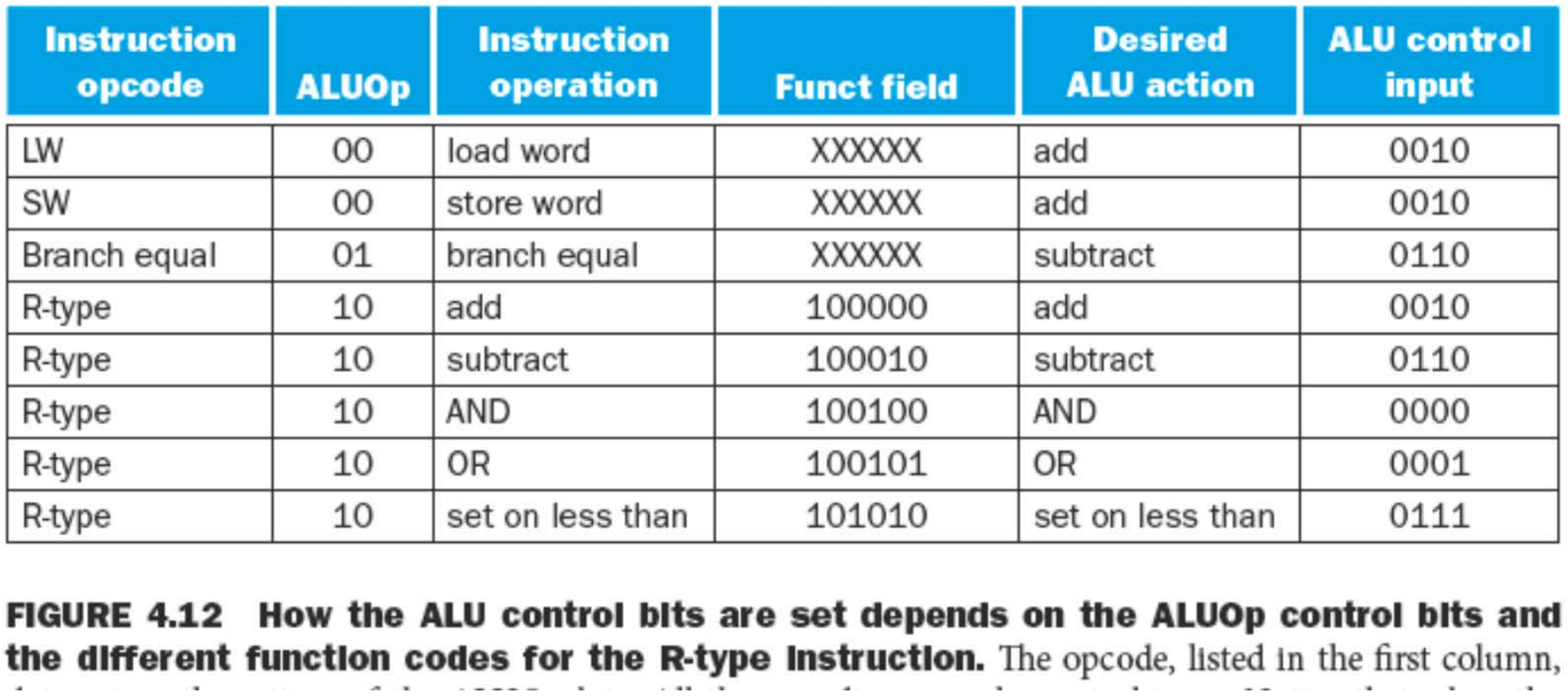
ALU_ctrl module （需要修改）

需要根據ALU_OP and funct去設定ALU_CTRL的訊號

需修改事項：
依據輸入的funct與ALU_OP去設定輸出的ALU_CTRL



參考ALU-Control圖(From testbook)

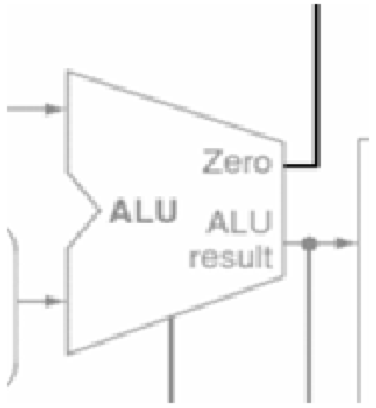


Instruction opcode	ALUOp	Instruction operation	Funct field	Desired ALU action	ALU control input
LW	00	load word	XXXXXX	add	0010
SW	00	store word	XXXXXX	add	0010
Branch equal	01	branch equal	XXXXXX	subtract	0110
R-type	10	add	100000	add	0010
R-type	10	subtract	100010	subtract	0110
R-type	10	AND	100100	AND	0000
R-type	10	OR	100101	OR	0001
R-type	10	set on less than	101010	set on less than	0111

FIGURE 4.12 How the ALU control bits are set depends on the ALUOp control bits and the different function codes for the R-type instruction. The opcode, listed in the first column,

ALU module（需要修改）

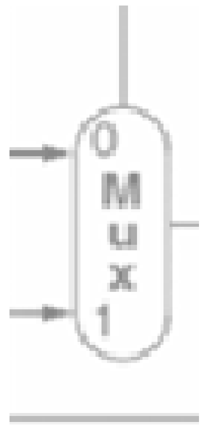
需修改事項：
依據ALU_ctrl module 輸入的ALU_CTRL實作執行運算



MUX_2_to_1 module （需要修改）

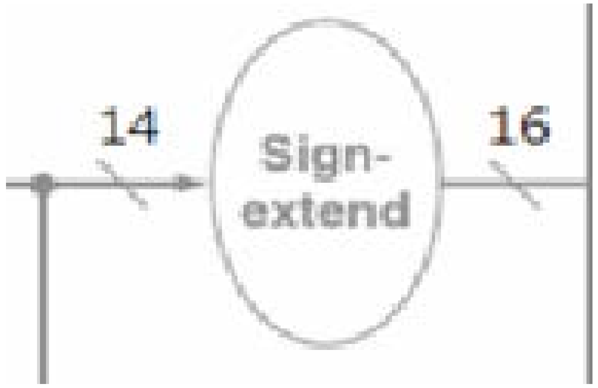
這次MUX有3個，分別爲
MUX_REG_Dst, MUX_ALUsrc, MUX_MEM_to_REG

需修改事項：
根據輸入的訊號選擇需要輸出的訊號線



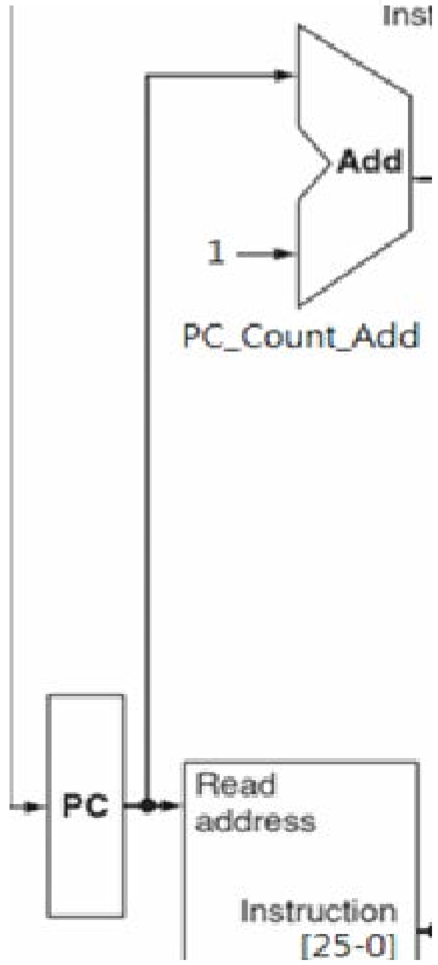
sign_extend module （需要修改）

需修改事項：
根據輸入的immediate_in(14 bits) extend成16bits



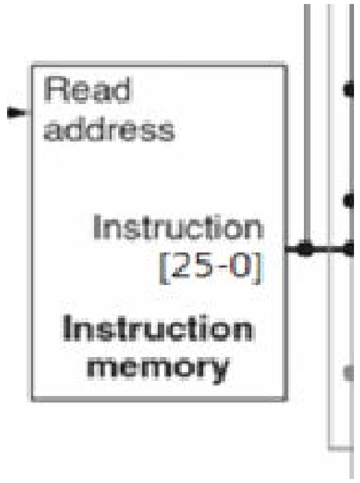
PC module（不需要修改）

PC為Programming Counter的縮寫，負責控制下一個要執行的指令，在此作業CPU中每個Clock cycle PC值會加一



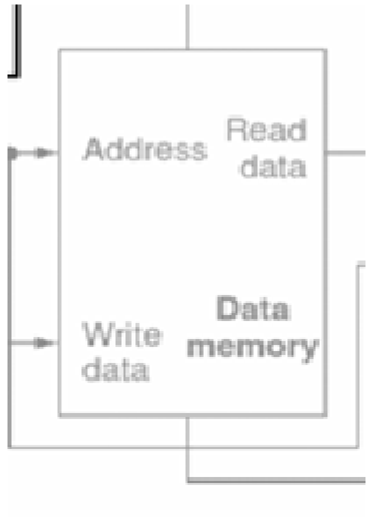
IM module （不需要修改）

IM為Instruction Memory的縮寫， IM module 會儲存所有指令並且依據PC將指令送至CPU



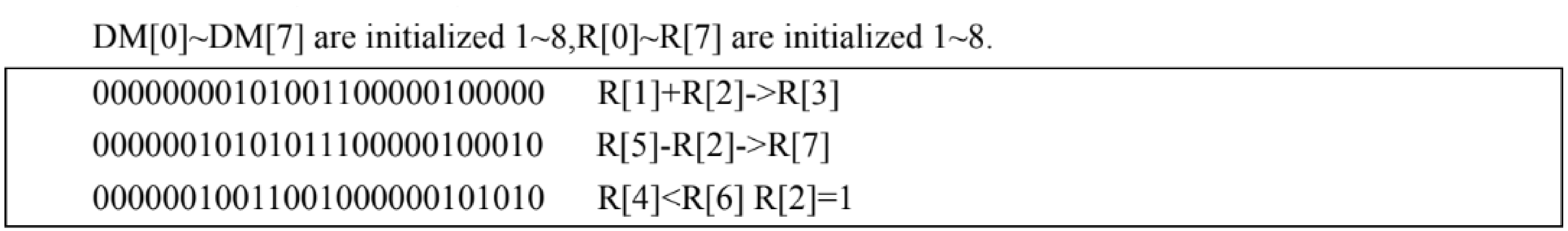
DM module （不需要修改）

功能與Reg一樣，但是目標在DM為Data Memory
實際上記憶體是以bytes為單位的連續空間配置，但爲了更輕易的實作這次的CPU作業，我們以Reg的實作方法去模擬DM，我們以16 bits爲單位去代表記憶體，另外以“grid” (例如DM[0] ，DM[1] 皆爲grid) 代表連續的空間



測試資料與測試結果

測試指令（26bits）


DM[0]~DM[7] are initialized 1~8,R[0]~R[7] are initialized 1~8.

00000000101001100000100000	R[1]+R[2]->R[3]
00000010101011100000100010	R[5]-R[2]->R[7]
00000100110010000000101010	R[4]<R[6] R[2]=1

3個cycle暫存器中value

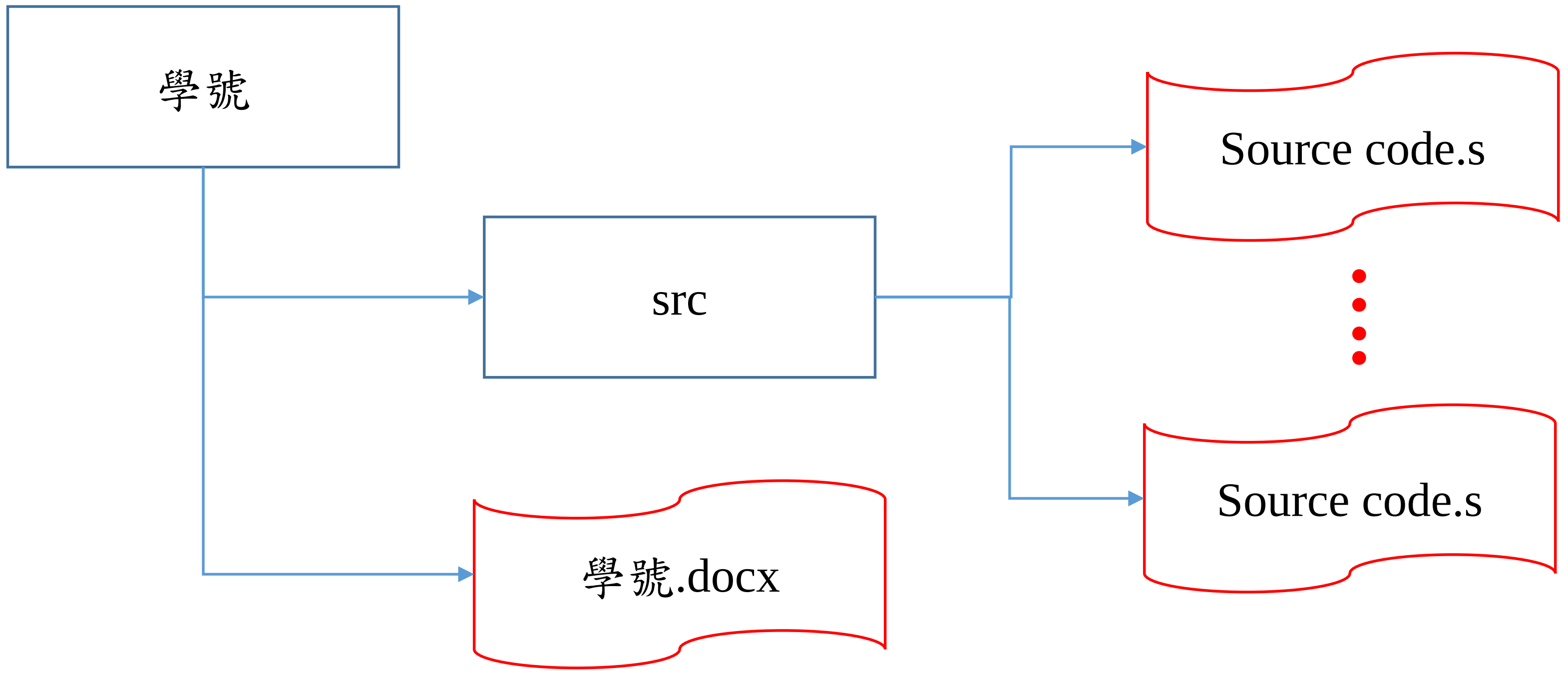
Simulation result:

After 1th cycle							
reg[0]	Reg[1]	Reg[2]	Reg[3]	Reg[4]	Reg[5]	Reg[6]	Reg[7]
1	2	**3**	5	5	6	7	8
DM[0]	DM[1]	DM[2]	DM[3]	DM[4]	DM[5]	DM[6]	DM[7]
1	2	3	4	5	6	7	8

After 2th cycle							
Reg[0]	Reg[1]	Reg[2]	Reg[3]	Reg[4]	Reg[5]	Reg[6]	Reg[7]
1	2	3	5	5	6	7	3
DM[0]	DM[1]	DM[2]	DM[3]	DM[4]	DM[5]	DM[6]	DM[7]
1	2	3	4	5	6	7	8

After 3th cycle							
Reg[0]	Reg[1]	Reg[2]	Reg[3]	Reg[4]	Reg[5]	Reg[6]	Reg[7]
1	2	1	5	5	6	7	3
DM[0]	DM[1]	DM[2]	DM[3]	DM[4]	DM[5]	DM[6]	DM[7]
1	2	3	**4**	5	6	7	8

作業繳交格式



請將檔案壓縮成.zip檔繳交
程式檔檔名需要按照作業提供格式不可更改

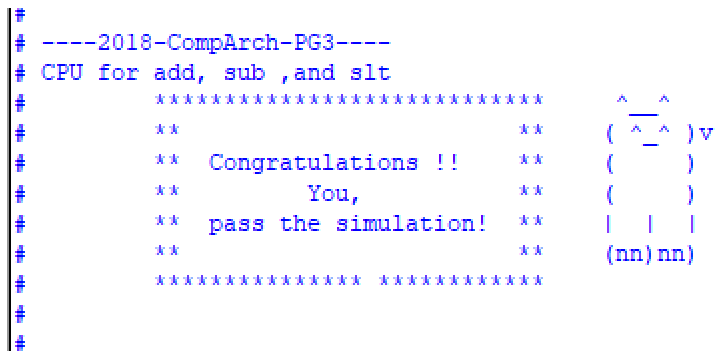
About Programming Assignment 3

1. 作業評分規則
   - 有依照作業要求繳交: 40分
   - 程式碼compile無誤：60分
   - 未按規定繳交作業：一項扣10分
   - 遲交0分

2. 程式作業請獨立完成，請勿抄襲同學之程式碼

About Programming Assignment 3

3. 繳交作業內容：
- 實驗結果圖與波形圖（如右圖）
- 解釋2條output波形運作流程
- 心得



```
#
# ----2018-CompArch-PG3----
# CPU for add, sub ,and slt
#         ****************************     ^__^
#         **                        **    ( ^_^ )v
#         **  Congratulations !!    **    (     )
#         **        You,            **    (     )
#         **  pass the simulation!  **    |  |  |
#         **                        **    (nn)nn)
#         *************** *************
#
#
```