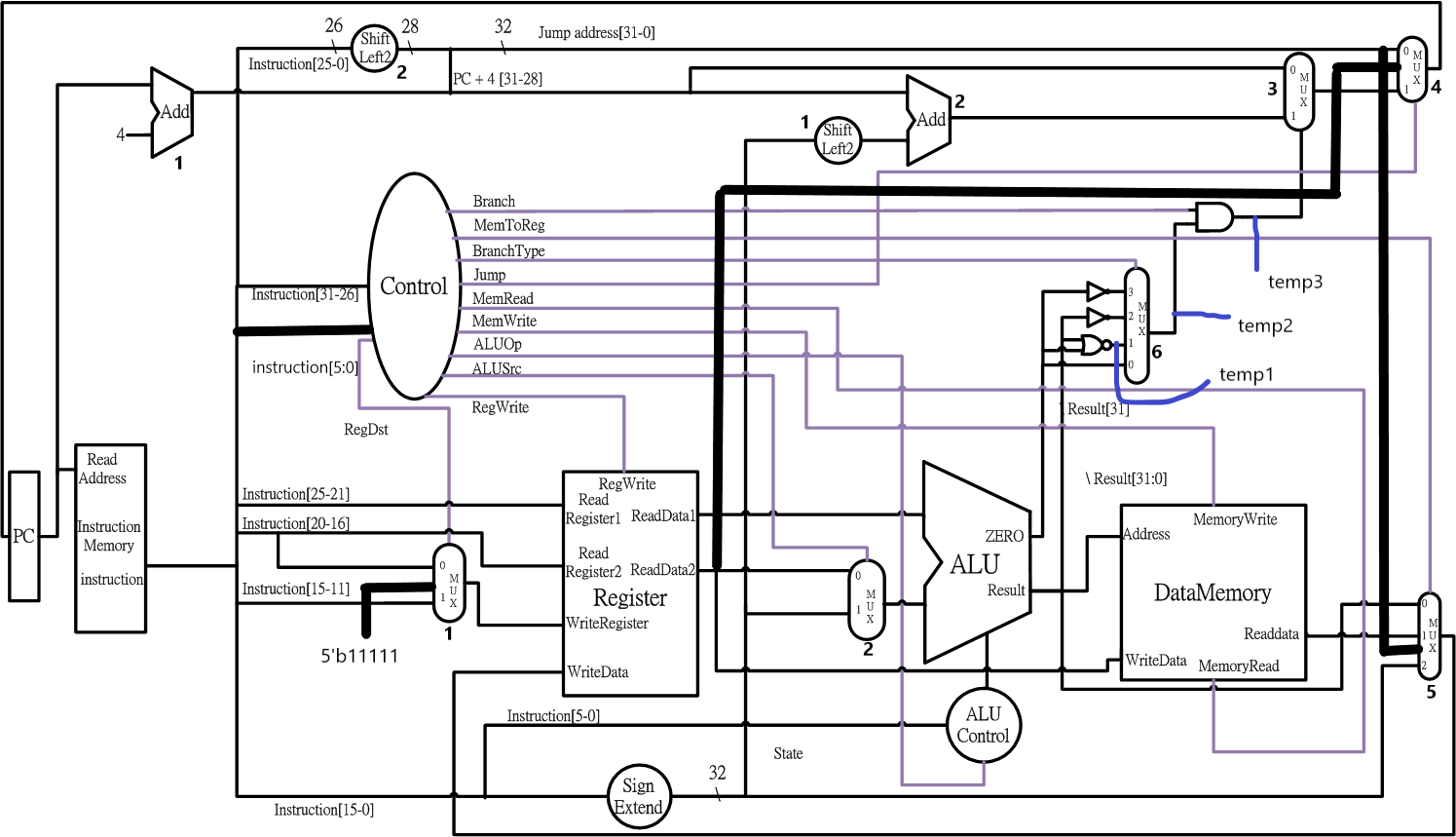
**Computer Organization**

**Architecture diagrams:**

****

**Hardware module analysis:**

PC :

控制本次輸入的instruction的位置，一種情況是上一次的instruction+4，另一種是根據branch所跳的位置來讀取insruction。

Adder unit:

本次作業中有兩個Adder，第一個是PC+4，第二個是PC+4 + Shift left 的output。

* Adder1:將目前的instruction位置加上4，成為下個instruction位置的選擇之一。
* Adder2:將PC+4 加上的Shift left 的output，成為下個instruction位置的選擇之一。

IM:

根據PC傳送的位置尋找instruction code，並將instruction code輸出到後續的Unit中

Mux Unit:

本次作業中有六個Mux unit。

* Mux1:控制register file RDaddr的輸入，控制指令是Decoder的RegDst，如果RegDst=00，output為instr\_out[20:16]

如果RegDst=01，output為instr\_out[15:11]

如果RegDst=10，output為5’11111( jal指令 )

* Mux2:控制ALU src2的輸入，控制指令是Decoder的RegSrc。

如果RegSrc=0，output為RT data output

如果RegSrc=1，output為Sign extention output

* Mux3:控制下一個instruction的位置，控制指令是temp2( Mux6的output 和 branch\_o的and )

如果temp3=0，output是PC+4

如果temp=1，output是 PC+4 + Shift left output

* Mux4: 控制下一個instruction的位置，控制指令是 jump\_o

如果jump\_o=00，output是temp3 = { pc\_plus4[31:28] , shift\_left2\_out}

如果jump\_o=01，output是Mux3的output

如果jump\_o=10，output是RSdata\_out( jr指令 )

* Mux5:控制回傳到register的值，控制指令是MemtoReg\_o

如果MemtoReg\_o=00，output是alu\_result

如果MemtoReg\_o=01，output是memory\_o

如果MemtoReg\_o=10，output是pc+4( jal指令 )

* Mux6:控制branch instruction的type，控制指令是branch\_type\_o

如果branch\_type\_o=00，output是alu\_zero

如果branch\_type\_o=01，output是temp1=~ (zero\_out & result\_out[31] )

如果branch\_type\_o=10，output是~result\_out[31]

如果branch\_type\_o=11，output是~alu\_zero

Decoder:

根據讀入的Op code，判斷出各指令，並根據目前指令控制RF、ALU control unit、Mux unit以及branch control。

* Branch\_o :000100(beq)
* MemtoReg\_o[1]: 000011(jal)
* MemtoReg\_o[0]: 100011(lw)
* jump\_o[1]:Opcode:000000，func\_code:001000 (jr)
* jump\_o[0]:001000(addi)、000000、100011(lw)、101011(sw)、000100(beq)
* MemRead\_o:100011(lw)
* MemWrite\_o:101011(sw)
* ALU\_op\_o [2]:001010(slti)
* ALU\_op\_o [1]:000000
* ALU\_op\_o [0]:000100 (beq)
* ALUSrc\_o: 001000(addi)、100011(lw)、101011(sw)、001010(slti)
* RegWrite\_o: 000000、001000(addi)、100011(lw)、001010(slti)、000011(jal)
* RegDst\_o[1]:000011(jal)
* RegDst\_o[0]:000000

RF:

暫存instruction中的各個register的值，輸出RS和RT的data，並根據Reg\_write來判斷是否寫入ALU result到 RD。

IM:

將資料寫入儲存，或是將儲存的資料讀到register中。

SE unit:

將instr[15:0]共16bit的data extend成32bit的data，而其數值並未改變。

Shift left unit

將SE output 往左shift 2bit，作為往後adder的input之一。

ALU control unit:

根據Decoder的 ALU\_op及funct\_code來判斷該指令，並輸入相對應的ALU\_control到ALU進行運算。

* ALU\_op=010(R-type)

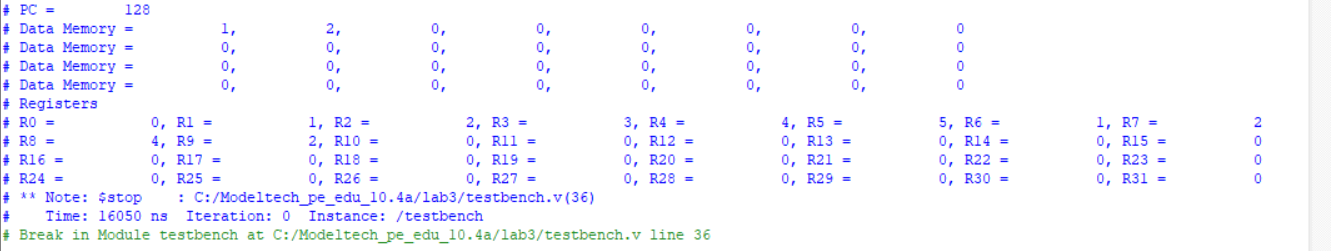
1. Funct\_code=100000(add): ALU\_control=0010
2. Funct\_code=100010(sub): ALU\_control=0110
3. Funct\_code=100100(and): ALU\_control=0000
4. Funct\_code=100001(or): ALU\_control=0001
5. Funct\_code=101010(slt): ALU\_control=0111

* ALU\_op=011(addi): ALU\_control=0010
* ALU\_op=100(slti): ALU\_control=0111
* ALU\_op=001(beq):ALU\_control=0110

ALU:

根據從ALU control unit輸入的ALU\_control，將src1和src2進行運算，並輸出zero及result。

**Finished part:**

****

**Problems you met and solutions:**

**Jal:**

**在test2中，執行Jal指令時會儲存20在R[31]，之後遇到相同指令時都會回到20繼續執行，可是我的R[31]會被之後執行Jal指令的位置覆寫，導致下一次執行Jal時不會回到20。**

**不知道如何解決這個問題QQ。**

**Summary:**

**Jal和Jr的datapath參考**

**https://www.youtube.com/watch?v=cZM\_tlDTakY&t=228s** [**https://www.youtube.com/watch?v=4p470WeycwU&t=260s**](https://www.youtube.com/watch?v=4p470WeycwU&t=260s)

**這次lab3是lab2的延伸，再加上lw、sw、j、jal、jr等指令，datapath較上次的作業複雜，更有挑戰性。**