<n26112437>_<劉兆軒> AIAS 2022 Lab 5 HW Submission

注意事項

1. 請不要用這份template 交作業, 建立一個新的codimd 檔案, 然後copy & paste 這個 template 到你創建的文件做修改。
2. 在上面的title修改你的學號與姓名 , 避免TA修改作業時把檔案跟人弄錯。
   - Ex: E24062133_徐韋凱 AIAS 2022 Lab5 HW Submission
3. 在Playlab 作業中心繳交作業時, 請用你創建的檔案鏈結繳交, 其他相關的資料與鏈結請依照 Template 規定的格式記載於codimd 上。

Gitlab 連結

Please paste the link to your private Gitlab repository for this homework submission here.

- Gitlab link (https://playlab.computing.ncku.edu.tw:4001/kevin1217/Lab05) <- 將連結填入括號中。

Hw5-1 Mix Adder

Scala Code

> 請放上你的程式碼並加上註解(中英文不限)，讓 TA明白你是如何完成的。

```scala
## scala code & comment
## make sure your have pass the MixAdderTest.scala
package aias_lab5.Hw1

import chisel3._
import aias_lab5.Lab._

class MixAdder (n:Int) extends Module{
  val io = IO(new Bundle{
      val Cin = Input(UInt(1.W))
      val in1 = Input(UInt((4*n).W))
      val in2 = Input(UInt((4*n).W))
      val Sum = Output(UInt((4*n).W))
      val Cout = Output(UInt(1.W))
  })

  //please implement your code below

  val CLA_Array = Array.fill(n)(Module(new CLAdder()).io)
  val carry = Wire(Vec(n+1, UInt(1.W)))
  val sum = Wire(Vec(n, UInt(4.W)))
  //val sum   = Wire(Vec(n, Bool()))

  carry(0) := io.Cin
  //對應adder的輸入輸出
  for (i <- 0 until n) {
    //FA:CLA的
    CLA_Array(i).in1 := io.in1(4*i+3,4*i)
    CLA_Array(i).in2 := io.in2(4*i+3,4*i)
    CLA_Array(i).Cin := carry(i)
        //wire的
    carry(i+1) := CLA_Array(i).Cout
    sum(i) := CLA_Array(i).Sum
  }
  //io的
  io.Sum := sum.asUInt
  io.Cout := carry(n)

  //io.Sum := 0.U
  //io.Cout := 0.U
}
```

Test Result

```
Elaborating design...
Done elaborating.
[info] [0.003] SEED 1680531725197
[info] [0.088] MixAdder test completed!!!!!
test MixAdder Success: 1000 tests passed in 1005 cycles in 0.118840 seconds 8456.76 Hz
[info] [0.089] RAN 1000 CYCLES PASSED
```

Hw5-2 Add-Suber

Scala Code

> 請放上你的程式碼並加上註解(中英文不限)，讓 TA明白你是如何完成的。

```
## scala code & comment
## make sure your have pass the MixAdderTest.scala
    package aias_lab5.Hw2

import chisel3._
import chisel3.util._
import aias_lab5.Lab._

class Add_Suber extends Module{
    val io = IO(new Bundle{
    val in_1 = Input(UInt(4.W))
        val in_2 = Input(UInt(4.W))
        val op = Input(Bool()) // 0:ADD 1:SUB
        val out = Output(UInt(4.W))
        val o_f = Output(Bool())
  })

  //please implement your code below
  //這個array存了n個fulladder
  val FA_Array = Array.fill(4)(Module(new FullAdder()).io)
  val carry = Wire(Vec(5, Bool()))
  val sum  = Wire(Vec(4, Bool()))
  carry(0) := io.op

  for (i <- 0 until 4) {
    FA_Array(i).A := io.in_1(i)
    FA_Array(i).B := carry(0)^io.in_2(i)
    FA_Array(i).Cin := carry(i)
    carry(i+1) := FA_Array(i).Cout
    sum(i) := FA_Array(i).Sum
  }

  io.out := sum.asUInt
  io.o_f := sum(3)^carry(4)
}

```

Test Result

```
Elaborating design...
Done elaborating.
[info] [0.004] SEED 1680531895784
[info] [0.045] Add_Suber test completed!!!
test Add_Suber Success: 128 tests passed in 517 cycles in 0.073242 seconds 7058.80 Hz
[info] [0.046] RAN 512 CYCLES PASSED
```

Hw5-3 Booth Multiplier

Scala Code

> 請放上你的程式碼並加上註解(中英文不限)，讓 TA明白你是如何完成的。

```
## scala code & comment
## make sure your have pass the Booth_MulTest.scala
    package aias_lab5.Hw3

import chisel3._
import chisel3.util._
import scala.annotation.switch

//------------------Radix 4---------------------
class Booth_Mul(width:Int) extends Module {
  val io = IO(new Bundle{
    val in1 = Input(UInt(width.W))      //Multiplicand 上
    val in2 = Input(UInt(width.W))      //Multiplier 下
    val out = Output(UInt((2*width).W)) //product
  })
  //please implement your code below
  val partial = Wire(Vec(8, SInt((2*width+1).W)))
  val multiplier= Wire(SInt((width+1).W))
  val multiplicand= Wire(SInt((2*width).W))
  val n_multiplicand= Wire(SInt((2*width).W))
  val product = Wire(SInt((2*width+1).W))


  multiplicand := (io.in1.asSInt)
  n_multiplicand := (((~(io.in1.asSInt))+1.S).asSInt).asSInt
  multiplier := Cat(io.in2,0.U).asSInt
  //operation
  //每次shift2，16bit共8次
  for (i <- 0 until (width/2)) {
  partial(i) := MuxLookup(multiplier(2*i+2,2*i),0.S,Seq(
  "b000".U -> 0.S, //0
  "b001".U -> multiplicand, //1
  "b010".U -> multiplicand ,//1
  "b011".U -> (multiplicand + multiplicand), //2
  "b100".U -> (n_multiplicand + n_multiplicand), //-2
  "b101".U -> n_multiplicand, //-1
  "b110".U -> n_multiplicand, //-1
  "b111".U -> 0.S //0
  ))
  }
  //partial product
  product := (partial(0)<<0) + (partial(1)<<2) + (partial(2)<<4) + (partial(3)<<

  //Sign_extend , Shift
  io.out := product(2*width,0).asUInt
}

```

Test Result

```
Elaborating design...
Done elaborating.
[info] [0.003] SEED 1680531975210
[info] [0.014] Booth_Mul test completed!!!!!
test Booth_Mul Success: 0 tests passed in 30 cycles in 0.040773 seconds 735.79 Hz
[info] [0.015] RAN 25 CYCLES PASSED
```

Hw5-4 Datapath Implementation for (3、4、7)

PC

Scala Code

請放上你的程式碼並加上註解(中英文不限)，讓 TA明白你是如何完成的。

```
## scala code & comment
## make sure your have pass the PCTest.scala
package aias_lab5.Hw4

import chisel3._
import chisel3.util._

class PC extends Module {
    val io = IO(new Bundle{
        val brtaken = Input(Bool())
        val jmptaken =  Input(Bool())
        val offset = Input(UInt(32.W))
        val pc = Output(UInt(32.W))
    })

    val pcReg = RegInit(0.U(32.W))
    //pcReg := pcReg + 4.U
        when(io.brtaken){
                pcReg := ((io.offset>>2.U)<<2.U)
        }.elsewhen(io.jmptaken){
                pcReg := ((io.offset>>2.U)<<2.U)
        }.otherwise{
                pcReg := pcReg + 4.U
        }
    io.pc := pcReg
}


```

Test Result

```
Elaborating design...
Done elaborating.
[info] [0.004] SEED 1680532539700
[info] [0.011] PC test completed!!!!!
test PC Success: 4 tests passed in 17 cycles in 0.040168 seconds 423.22 Hz
[info] [0.012] RAN 12 CYCLES PASSED
```

Decoder

Scala Code

> 請放上你的程式碼並加上註解(中英文不限)，讓 TA明白你是如何完成的。

```
## scala code & comment
## make sure your have pass the DecoderTest.scala
package aias_lab5.Hw4

import chisel3._
import chisel3.util._

object opcode_map {
    val LOAD      = "b0000011".U
    val STORE     = "b0100011".U
    val BRANCH    = "b1100011".U
    val JALR      = "b1100111".U
    val JAL       = "b1101111".U
    val OP_IMM    = "b0010011".U
    val OP        = "b0110011".U
    val AUIPC     = "b0010111".U
    val LUI       = "b0110111".U
}

import opcode_map._

class Decoder extends Module{
    val io = IO(new Bundle{
        val inst = Input(UInt(32.W))

        //Please fill in the blanks by yourself
        val funct3 = Output(UInt(3.W))
               val funct7 = Output(UInt(7.W))
        val rs1 = Output(UInt(5.W))
               val rs2 = Output(UInt(5.W))
        val rd = Output(UInt(5.W))
        val opcode = Output(UInt(7.W))
        val imm = Output(SInt(32.W))

        val ctrl_RegWEn = Output(Bool()) // for Reg write back
        val ctrl_ASel = Output(Bool()) // for alu src1
        val ctrl_BSel = Output(Bool()) // for alu src2
        val ctrl_Br = Output(Bool()) // for branch inst.
        val ctrl_Jmp = Output(Bool()) // for jump inst.
        val ctrl_Lui = Output(Bool()) // for lui inst.
        val ctrl_MemRW = Output(Bool()) // for L/S inst
        val ctrl_WBSel = Output(Bool())
    })

    //Please fill in the blanks by yourself
    io.funct3 := io.inst(14,12)
        io.funct7 := io.inst(31,25)
    io.rs1 := io.inst(19,15)
        io.rs2 :=io.inst(24,20)
    io.rd := io.inst(11,7)
    io.opcode := io.inst(6,0)

    //ImmGen
```

```
54      io.imm := MuxLookup(io.opcode,0.S,Seq(
55          //R-type
56          //Please fill in the blanks by yourself
57
58          //I-type
59          OP_IMM -> io.inst(31,20).asSInt,
60          //Please fill in the blanks by yourself
61          JALR -> io.inst(31,20).asSInt,
62          LOAD -> io.inst(31,20).asSInt,
63
64          //B-type
65          //Please fill in the blanks by yourself
66          BRANCH -> Cat(io.inst(31),io.inst(7),io.inst(30,25),io.inst(11,8),0.U).a
67
68          //S-type
69          //Please fill in the blanks by yourself
70                 STORE -> Cat(io.inst(31,25),io.inst(11,7)).asSInt,
71
72          //U-type
73          //Please fill in the blanks by yourself
74          LUI -> (io.inst(31,12)<<12.U).asSInt,
75                 AUIPC -> (io.inst(31,12).asSInt<<12.U).asSInt,
76
77          //J-type
78          //Please fill in the blanks by yourself
79                 JAL -> Cat(io.inst(31),io.inst(19,12),io.inst(20),io.inst(30,21),
80      ))
81
82      //Controller
83      io.ctrl_RegWEn := Mux(io.opcode===OP,true.B,false.B)
84      io.ctrl_ASel := Mux(io.opcode===BRANCH|| io.opcode===JAL,true.B,false.B)
85      io.ctrl_BSel := Mux(io.opcode===OP,false.B,true.B)
86      io.ctrl_Br := Mux(io.opcode===BRANCH,true.B,false.B)
87      io.ctrl_Jmp := Mux(io.opcode===JAL||io.opcode===JALR,true.B,false.B)
88      io.ctrl_Lui := Mux(io.opcode===LUI,true.B,false.B)
89      io.ctrl_MemRW := Mux(io.opcode===STORE,true.B,false.B)
90      io.ctrl_WBSel := Mux(io.opcode===LOAD,false.B,true.B)
91
92          //true: from alu , false: from dm , another source?
93  }
94
```

Test Result

```
Elaborating design...
Done elaborating.
[info] [0.004] SEED 1680532345861
[info] [0.031] Decoder test completed!!!!!
test Decoder Success: 180 tests passed in 185 cycles in 0.058090 seconds 3184.69 Hz
[info] [0.032] RAN 180 CYCLES PASSED
```

BranchComp

Scala Code

> 請放上你的程式碼並加上註解(中英文不限)，讓 TA明白你是如何完成的。

```
## scala code & comment
## make sure your have pass the BranchCompTest.scala
package aias_lab5.Hw4

import chisel3._
import chisel3.util._

object condition{
  val EQ = "b000".U
  val NE = "b001".U
  val LT = "b100".U
  val GE = "b101".U
  val LTU = "b110".U
  val GEU = "b111".U
}

import condition._

class BranchComp extends Module{
    val io = IO(new Bundle{
        val en = Input(Bool())
        val funct3 = Input(UInt(3.W))
        val src1 = Input(UInt(32.W))
        val src2 = Input(UInt(32.W))

        val brtaken = Output(Bool()) //for pc.io.brtaken
    })

    //please implement your code below
        io.brtaken := false.B
        when(io.en){
        io.brtaken := MuxLookup(io.funct3,0.U,Seq(
        EQ -> (io.src1 === io.src2),
        NE -> (io.src1 =/= io.src2),
        LT -> (io.src1.asSInt < io.src2.asSInt),
        GE -> (io.src1.asSInt >= io.src2.asSInt),
        LTU -> (io.src1 < io.src2),
        GEU -> (io.src1 >= io.src2),
        ))
        }
}
```

Test Result

```
Elaborating design...
Done elaborating.
[info] [0.004] SEED 1680532478387
[info] [0.025] BranchComp test completed!!!!!
test BranchComp Success: 140 tests passed in 145 cycles in 0.049936 seconds 2903.74 Hz
[info] [0.026] RAN 140 CYCLES PASSED
```

Hw5-4 Datapath Implementation for (1、2、6、8、9)

InstMem_wc.txt

> 請同學放上自行準備的測資，需含有(R、I、S、B、J type測資)

```
//assembly code
addi x2 x2 -1
slli x0 x4 1
add x1 x2 x1
sub x1 x6 x1
lb x1 x0 0
lh x1 x0 1
beq x1 x1 8
blt x1 x2 4
jal x0 12
jalr x1 x6 9
sb x1 x2 2
sh x1 x2 4

//machine code
11111111111100010000000100010011
00000000000100100001000000010011
00000000001000001000000010110011
01000000000100110000000000110011
00000000000000000000000010000011
00000000001000000001000010000011
00000000000100001000010001100011
00000000001000001100001001100011
00000001100000000000000001101111
00000000100100110000000011100111
00000000000100010000000100100011
00000000000100010001001000100011
```

Scala Code

> 請放上你的程式碼並加上註解(中英文不限)，讓 TA明白你是如何完成的。

- top

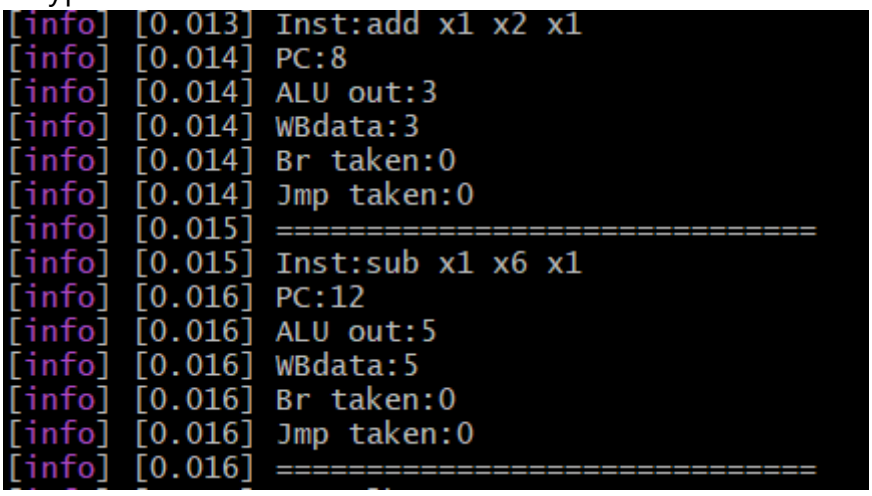
```
## scala code & comment
## make sure you have comfirm the correctness of every type of instructions by y
package aias_lab5.Hw4

import chisel3._
import chisel3.util._

class top extends Module {
    val io = IO(new Bundle{
    val pc_out = Output(UInt(32.W))
    val alu_out = Output(UInt(32.W))
    val rf_wdata_out = Output(UInt(32.W))
    val brtaken_out = Output(Bool())
    val jmptaken_out = Output(Bool())
    val dm_rdata_out = Output(SInt(32.W))
    val dm_wdata_out = Output(UInt(32.W))
    })

    val pc = Module(new PC())
    val im = Module(new InstMem())
    val dc = Module(new Decoder())
    val rf = Module(new RegFile(2))
    val alu = Module(new ALU())
    val bc = Module(new BranchComp())
    val dm = Module(new DataMem())

    //PC
    pc.io.jmptaken :=  false.B // Don't modify //dc.io.ctrl_Jmp
    pc.io.brtaken :=   false.B // Don't modify //dc.io.ctrl_Br
    pc.io.offset :=   0.U // Don't modify //dc.io.imm.asUInt

    //Insruction Memory
    im.io.raddr := pc.io.pc

    //Decoder
    dc.io.inst := im.io.rdata

    //RegFile
    rf.io.raddr(0) := dc.io.rs1
    rf.io.raddr(1) := dc.io.rs2
    rf.io.wdata := Mux(dc.io.ctrl_WBSel,Mux(dc.io.opcode==="b1101111".U||dc.io.c
    rf.io.waddr := 0.U  // Don't modify
    rf.io.wen := false.B // Don't modify

    //ALU
    val rdata_or_zero = WireDefault(0.U(32.W))
    alu.io.src1 := Mux(dc.io.ctrl_ASel,pc.io.pc,rf.io.rdata(0))
    alu.io.src2 := Mux(dc.io.ctrl_BSel
                    ,Mux(dc.io.ctrl_Br
                    ,Mux(bc.io.brtaken,dc.io.imm.asUInt,0.U)
                    ,dc.io.imm.asUInt)
                    ,rf.io.rdata(1))

```

```
54        alu.io.funct3 := dc.io.funct3
55        alu.io.funct7 := dc.io.funct7
56        alu.io.opcode := dc.io.opcode
57
58        //Data Memory
59        dm.io.funct3 := dc.io.funct3
60        dm.io.raddr := alu.io.out
61        dm.io.wen   := dc.io.ctrl_MemRW
62        dm.io.waddr := alu.io.out
63        dm.io.wdata := rf.io.rdata(1)
64
65        //Branch Comparator
66        bc.io.en := dc.io.ctrl_Br // need to be changed
67        bc.io.funct3 := dc.io.funct3
68        bc.io.src1 := rf.io.rdata(0)
69        bc.io.src2 := rf.io.rdata(1)
70
71
72        //Check Ports
73        io.pc_out := pc.io.pc
74        io.alu_out := alu.io.out
75        io.rf_wdata_out := rf.io.wdata
76        io.brtaken_out := bc.io.brtaken
77        io.jmptaken_out := dc.io.ctrl_Jmp//false.B // need to be changed
78        io.dm_rdata_out := dm.io.rdata
79        io.dm_wdata_out := dm.io.wdata
80    }
```

Test Result

> 請仿效文件上的截圖，放上各個type的指令驗證結果(每種type至少兩個，且與測資呼應。)

- R-type



- I-type

```
[info] [0.007] Inst:addi x2 x2 -1
[info] [0.010] PC:0
[info] [0.011] ALU out:1
[info] [0.011] WBdata:1
[info] [0.011] Br taken:0
[info] [0.011] Jmp taken:0
[info] [0.011] ============================
[info] [0.012] Inst:slli x0 x4 1
[info] [0.012] PC:4
[info] [0.012] ALU out:8
[info] [0.013] WBdata:8
[info] [0.013] Br taken:0
[info] [0.013] Jmp taken:0
[info] [0.013] ============================
```

- S-type

```
[info] [0.025] Inst:sb x1 x2 2
[info] [0.025] PC:40
[info] [0.026] ALU out:4
[info] [0.026] WBdata:4
[info] [0.026] Br taken:0
[info] [0.026] Jmp taken:0
[info] [0.026] ============================
[info] [0.027] Inst:sh x1 x2 4
[info] [0.027] PC:44
[info] [0.027] ALU out:6
[info] [0.027] WBdata:6
[info] [0.028] Br taken:0
[info] [0.028] Jmp taken:0
```

- B-type

```
[info] [0.021] Inst:beq x1 x1 8
[info] [0.022] PC:24
[info] [0.022] ALU out:32
[info] [0.022] WBdata:32
[info] [0.022] Br taken:1
[info] [0.022] Jmp taken:0
[info] [0.023] ============================
[info] [0.023] Inst:blt x1 x2 4
[info] [0.024] PC:28
[info] [0.024] ALU out:32
[info] [0.024] WBdata:32
[info] [0.024] Br taken:1
[info] [0.024] Jmp taken:0
```

- J-type

```
[info] [0.025] Inst:jal x0 12
[info] [0.025] PC:32
[info] [0.026] ALU out:44
[info] [0.026] WBdata:36
[info] [0.026] Br taken:0
[info] [0.026] Jmp taken:1
[info] [0.026] ============================
[info] [0.027] Inst:jalr x1 x6 9
[info] [0.028] PC:36
[info] [0.028] ALU out:15
[info] [0.028] WBdata:40
[info] [0.028] Br taken:0
[info] [0.028] Jmp taken:1
[info] [0.028] ============================
```

文件中的問答題

- Q1:哪兩種type的指令需要仰賴ALU最多的功能呢？對於其他type的指令而言，ALU的功能是？
  - Ans1:R type和I type指令需要ALU最多功能，而其餘指令都只要ALU的相加功能而已。
- Q2:Branch情況發生時，**pc+offset**會從哪裡傳回pc呢？Beq x0 x0 imm 可以完全取代 jal x0 imm嗎？如果不行，為什麼?
  - Ans2:從alu.io.out回傳到PC；不行，因為jal可以跳更遠，beq有些地方跳不到。
- Q3:假設我用int存取src1和src2，由於int是signed的方式存取，比大小(-1<3)自然能夠成立。有沒有什麼方式是**不需透過轉換Dtype**，就能夠實現Unsigned的比較方式呢?比如說，該如何驗證BGEU呢？(可以用pseudo code或者你認為你能表達清楚你的想法，用文字呈現也行。)
  - Ans3:若先比較src1和src2的最高位找正負數，若不同則最高位為1的大於最高位為0的；若相同則分成最高位為1(正數)，後面數值越大表示整體值越大；最高位為0(負數)，後面數值越大表示整體值越小。

意見回饋和心得(可填可不填)

好難哦，希望後面lab手下留情。