<n26112437>_<劉兆軒> AIAS 2023 Lab 8 HW Submission

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 到你創建的檔案做修改。
2. 請修改你的學號與姓名在上面的 title, 以避免TA 修改作業時把檔案跟人弄錯了
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。

記得在文件標題上修改你的 <學號> <姓名>

Gitlab code link

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

- Gitlab link - https://playlab.computing.ncku.edu.tw:4001/kevin1217/lab08 (https://playlab.computing.ncku.edu.tw:4001/kevin1217/lab08)

Bonus link

- Bonus link - https://playlab.computing.ncku.edu.tw:3001/cF-pgeMeThCWzcADjfuZMA?view (https://playlab.computing.ncku.edu.tw:3001/cF-pgeMeThCWzcADjfuZMA?view)

HW8-1 Conv2D Assembly Code Implementation

Assembly Code - `test_data_1.S`

> 請放上你的程式碼並加上註解，讓 TA明白你是如何完成的。

```
.data
## input data size = 2x8x8
input_data:
.byte 0 1 2 3 4 5 6 7
.byte 0 1 2 3 4 5 6 7
.byte 0 1 2 3 4 5 6 7
.byte 0 1 2 3 4 5 6 7
.byte 0 1 2 3 4 5 6 7
.byte 0 1 2 3 4 5 6 7
.byte 0 1 2 3 4 5 6 7
.byte 0 1 2 3 4 5 6 7

.byte 0 1 2 3 4 5 6 7
.byte 0 1 2 3 4 5 6 7
.byte 0 1 2 3 4 5 6 7
.byte 0 1 2 3 4 5 6 7
.byte 0 1 2 3 4 5 6 7
.byte 0 1 2 3 4 5 6 7
.byte 0 1 2 3 4 5 6 7
.byte 0 1 2 3 4 5 6 7

## kernel size 2x3x3
kernel_data:
.byte 0 1 2
.byte 0 1 2
.byte 0 1 2

.byte 0 1 2
.byte 0 1 2
.byte 0 1 2

## output data size 1x6x6
output_data:
.byte 0 0 0 0 0 0 0 0
.byte 0 0 0 0 0 0 0 0
.byte 0 0 0 0 0 0 0 0
.byte 0 0 0 0 0 0 0 0
.byte 0 0 0 0 0 0 0 0
.byte 0 0 0 0 0 0 0 0

.text

## N -> matrix size CHANNEL_SIZE x INPUT_SIZE x INPUT_SIZE (input)
## M -> matrix size CHANNEL_SIZE x KERNEL_SIZE x KERNEL_SIZE (input)
## P -> matrix size CHANNEL_SIZE x OUTPUT_SIZE (output result)

main:
prologue:
li sp,0x10000

## testing parameter//
## s0 -> base address of N
## s1 -> base address of M
```

```
54  ## s2 -> base address of P
55  la s0,input_data
56  la s1,kernel_data
57  la s2,output_data
58
59  ## s3 -> INPUT_SIZE
60  ## s4 -> OUTPUT_SIZE
61  ## s5 -> KERNEL_SIZE
62  ## s6 -> CHANNEL_SIZE
63  li s3,8
64  li s4,6
65  li s5,3
66  li s6,2
67
68  ## parameter initialize
69  ## s8 -> cc
70  ## s9 -> ii
71  ## s10 -> jj
72  ## t0 -> i
73  ## t1 -> j
74  ## t2 -> m
75  ## t3 -> n
76  li s8,0
77  li s9,0
78  li s10,0
79  li t0,0
80  li t1,0
81  li t2,0
82  li t3,0
83
84
85  ## other 未用t4
86  ## t5 -> kernel_center =  KERNEL_SIZE / 2
87  ## s7 -> N[index]
88  ## s11 -> M[index]
89  ## t6-> P[index]
90  ## t4 -> buffer
91  srli t5,s5,1
92  li s7,0
93  li s11,0
94  li t6,0
95  li t4,0
96
97  # OUTPUT_SIZE_row for loop
98  OUTPUT_SIZE_row:
99  bge t0,s4,epilogue
100 li t1,0
101 jal OUTPUT_SIZE_col
102 addi t0,t0,1
103 j OUTPUT_SIZE_row
104
105 ## OUTPUT_SIZE_col for loop
106 OUTPUT_SIZE_col:
```

```
## 超過範圍跳出迴圈
bge t1,s4,return
## 每次跳都要先存原本for的addr
addi sp,sp,-4
sw ra,0(sp)
## 下個for的index重置為0
li t2,0
## 跳到下個for
jal KERNEL_SIZE_row
## 下個for結束後，return回來
lw ra,0(sp)
## 刪除該回stack內的addr
addi sp,sp,4
## 該for迴圈的index+1
addi t1,t1,1
j OUTPUT_SIZE_col

# KERNEL_SIZE_row for loop
KERNEL_SIZE_row:
bge t2,s5,return
addi sp,sp,-4
sw ra,0(sp)
li t3,0
jal KERNEL_SIZE_col
lw ra,0(sp)
addi sp,sp,4
addi t2,t2,1
j KERNEL_SIZE_row

## KERNEL_SIZE_col for loop
KERNEL_SIZE_col:
bge t3,s5,return
addi sp,sp,-4
sw ra,0(sp)
li s8,0
jal CHANNEL_SIZE
lw ra,0(sp)
addi sp,sp,4
addi t3,t3,1
j KERNEL_SIZE_col


## CHANNEL_SIZE for loop
CHANNEL_SIZE:
bge s8,s6,return
addi sp,sp,-4
sw ra,0(sp)
jal mult
lw ra,0(sp)
addi sp,sp,4
addi s8,s8,1
j CHANNEL_SIZE

```

```
## calculate
mult:
## ii = (m - kernel_center)
sub s9,t2,t5
## ii = i + (m - kernel_center)
add s9,s9,t0
# ii = i + (m - kernel_center) +1;
addi s9,s9,1

## jj = (n - kernel_center)
sub s10,t3,t5
## jj = j + (n - kernel_center)
add s10,s10,t1
## jj = j + (n - kernel_center) +1;
addi s10,s10,1

## P[i][j] += N[cc][ii][jj] * M[cc][m][n];
# s7         s11             t6
## N[cc][ii][jj] = N長 * N寬 * cc + ii * N寬 + jj
## s11             s3     s3    s8   s9   s3    s10
mul t4,s3,s3
mul t4,t4,s8
mul s11,s9,s3
add s11,s11,t4
add s11,s11,s10
add s11,s0,s11
lb s11,0(s11)

## M[cc][m][n] = M長 * M寬 * cc + m * M寬 + n
## t6            s5    s5    s8   t2  s5    t3
mul t4,s5,s5
mul t4,t4,s8
mul t6,t2,s5
add t6,t6,t4
add t6,t6,t3
add t6,s1,t6
lb t6,0(t6)

mul t4,s11,t6

## P[i][j] = P寬 * i + j
## s2        s4    t0  t1
mul s7,s4,t0
add s7,s7,t1
add s7,s2,s7
lb s11,0(s7)
add s11,t4,s11
sb s11,0(s7)
ret

return:
ret

.eil
```

```
213 epilogue:
214 hcf
215 ## Terminate
```

Simulation Result

請複製上你在 Venus上的模擬結果 (screenshot), 驗證程式碼的正確性。

```
1 ## Your screenshot
```

0x100000c0	00	00	00	00
0x100000bc	00	00	00	00
0x100000b8	00	00	00	00
0x100000b4	66	78	00	00
0x100000b0	1e	30	42	54
0x100000ac	42	54	66	78
0x100000a8	66	78	1e	30
0x100000a4	1e	30	42	54
0x100000a0	42	54	66	78
0x1000009c	66	78	1e	30
0x10000098	1e	30	42	54
0x10000094	42	54	66	78
0x10000090	01	02	1e	30

訂正：在emulator上顯示out的結果
由於暫存器數量不多，無法將所有值顯現出來，
故在x19~x27放入memory的前幾個數值，
1e 30 52 54 66 78 30 42.....，
會發現是循環的，如venus上所示

```
Reached Halt and Catch Fire instruction!
inst:  29042 pc:    484 src line: 223
x00:0x00000000 x01:0x000000ac x02:0x00010000 x03:0x00000000 x04:0x00000000 x05:0x00000006 x06:0x00000006 x07:0x00000003
x08:0x00008000 x09:0x00008080 x10:0x00000000 x11:0x00000000 x12:0x00000000 x13:0x00000000 x14:0x00000000 x15:0x00000000
x16:0x00000000 x17:0x00000000 x18:0x00008092 x19:0x0000001e x20:0x00000030 x21:0x00000042 x22:0x00000054 x23:0x00000066
x24:0x00000078 x25:0x0000001e x26:0x00000030 x27:0x00000042 x28:0x00000003 x29:0x0000000e x30:0x00000001 x31:0x00000002
```

HW8-2 Conv2D Assembly Code Acceleration

Assembly Code - `test_data_2.S`

> 請放上你的程式碼並加上註解，讓 TA明白你是如何完成的。

```
.data
## input data size = 2x8x8
input_data:
.byte 0 1 2 3 4 5 6 7
.byte 0 1 2 3 4 5 6 7
.byte 0 1 2 3 4 5 6 7
.byte 0 1 2 3 4 5 6 7
.byte 0 1 2 3 4 5 6 7
.byte 0 1 2 3 4 5 6 7
.byte 0 1 2 3 4 5 6 7
.byte 0 1 2 3 4 5 6 7

.byte 0 1 2 3 4 5 6 7
.byte 0 1 2 3 4 5 6 7
.byte 0 1 2 3 4 5 6 7
.byte 0 1 2 3 4 5 6 7
.byte 0 1 2 3 4 5 6 7
.byte 0 1 2 3 4 5 6 7
.byte 0 1 2 3 4 5 6 7
.byte 0 1 2 3 4 5 6 7

## kernel size 2x3x3
kernel_data:
.byte 0 1 2
.byte 0 1 2
.byte 0 1 2

.byte 0 1 2
.byte 0 1 2
.byte 0 1 2

## output data size 1x6x6
output_data:
.byte 0 0 0 0 0 0 0 0
.byte 0 0 0 0 0 0 0 0
.byte 0 0 0 0 0 0 0 0
.byte 0 0 0 0 0 0 0 0
.byte 0 0 0 0 0 0 0 0
.byte 0 0 0 0 0 0 0 0

.text

## N -> matrix size CHANNEL_SIZE x INPUT_SIZE x INPUT_SIZE (input)
## M -> matrix size CHANNEL_SIZE x KERNEL_SIZE x KERNEL_SIZE (input)
## P -> matrix size CHANNEL_SIZE x OUTPUT_SIZE (output result)

main:
prologue:
li sp,0x10000
## testing parameter//
## s0 -> base address of N
## s1 -> base address of M
## s2 -> base address of P
```

```
54  la s0,input_data
55  la s1,kernel_data
56  la s2,output_data
57
58  ## s3 -> INPUT_SIZE
59  ## s4 -> OUTPUT_SIZE
60  ## s5 -> KERNEL_SIZE
61  ## s6 -> CHANNEL_SIZE
62  li s3,8
63  li s4,6
64  li s5,3
65  li s6,2
66
67  ## parameter initialize
68  ## s8 -> cc
69  ## t0 -> i
70  ## t1 -> j
71  ## t2 -> m
72  ## t3 -> n
73  li s8,0
74  li t0,0
75  li t1,0
76  li t2,0
77  li t3,0
78
79
80  ## other 未用t5
81  ## t4 -> N[index]
82  ## t6 -> M[index]
83  ## s7-> P[index]
84  ## s10 -> buffer
85  ## s9 -> 8
86  li s7,0
87  li s11,0
88  li t6,0
89  li t4,0
90  li s9,8
91  li s10,0
92
93  # loop_i for loop
94  loop_i:
95  bge t0,s4,epilogue
96  li t2,0
97  jal loop_m
98  addi t0,t0,1
99  j loop_i
100
101 # loop_m for loop
102 loop_m:
103 bge t2,s5,return
104 addi sp,sp,-4
105 sw ra,0(sp)
106 li t3,0
```

```
jal loop_n
lw ra,0(sp)
addi sp,sp,4
addi t2,t2,1
j loop_m

## loop_n for loop
loop_n:
bge t3,s5,return
addi sp,sp,-4
sw ra,0(sp)
li s8,0
jal loop_cc
lw ra,0(sp)
addi sp,sp,4
addi t3,t3,1
j loop_n


## loop_cc for loop
loop_cc:
bge s8,s6,return

## N[cc][i+m][n+x] = N[cc][i+m][n開始](vec) = N長 * cc + (i+m) + n
## t4                                          s3   s8      t0t2   t3
mul t4,s3,s8
add t4,t4,t0
add t4,t4,t2
mul t4,t4,s9
add t4,s0,t4
add t4,t4,t3
vle8.v v1,(t4),0

## M[cc][m][n] = M長 * M寬 * cc + m * M寬 + n
## t6            s5    s5    s8   t2  s5    t3
mul s10,s5,s5
mul s10,s10,s8
mul t6,t2,s5
add t6,t6,s10
add s10,t6,t3
add t6,s1,s10
lb t6,0(t6)


## P += N * M
vmul.vx v1,v1,t6,0

## P[i](vec) = i
## s7          t0
mul s7,t0,s9
add s7,s7,s2
vle8.v v2,(s7),0
vadd.vv v2,v1,v2,0
```

```
vse8.v v2,(s7),0

## cc++
addi s8,s8,1
j loop_cc

return:
ret

epilogue:
vle8.v v20,(s2),0
addi s2,s2,8
vle8.v v21,(s2),0
addi s2,s2,8
vle8.v v22,(s2),0
addi s2,s2,8
vle8.v v23,(s2),0
addi s2,s2,8
vle8.v v24,(s2),0
addi s2,s2,8
vle8.v v25,(s2),0
addi s2,s2,8
vle8.v v26,(s2),0
addi s2,s2,-48
hcf
## Terminate
```

Simulation Result

請複製上你在 Emulator上的模擬結果 (擷取 scalar register + vector register結果), 驗證程式碼的正確性。最後詳細說明所使用參數 & 寫回 vector register中的格式。

```
## Your simulation result & comment
## Please describe you store your output data in which register

## testing parameter

s0 -> base address of N
s1 -> base address of M
s2 -> base address of P

s3 -> INPUT_SIZE
s4 -> OUTPUT_SIZE
s5 -> KERNEL_SIZE
s6 -> CHANNEL_SIZE

s8 -> cc
t0 -> i
t1 -> j
t2 -> m
t3 -> n

t4 -> N[index]
t6 -> M[index]
s7 -> P[index]
s10 -> buffer
s9 -> 8

vector20~25存放的是output_data，即從s2的初始位址開始存，值得注意的是，
由於output_data是6*6的矩陣，但一個vec會讀到8byte，
所以會有2個byte是不用理的，即紅框外的數字，這裡只要關注紅框內的數值即可。
```

```
Reached Halt and Catch Fire instruction!
inst:   3450 pc:    408 src line: 184
x00:0x00000000 x01:0x000000a8 x02:0x00010000 x03:0x00000000 x04:0x00000000 x05:0x00000006 x06:0x00000000 x07:0x00000003
x08:0x00008000 x09:0x00008080 x10:0x00000000 x11:0x00000000 x12:0x00000000 x13:0x00000000 x14:0x00000000 x15:0x00000000
x16:0x00000000 x17:0x00000000 x18:0x00008092 x19:0x00000008 x20:0x00000006 x21:0x00000003 x22:0x00000002 x23:0x000080ba
x24:0x00000002 x25:0x00000008 x26:0x00000011 x27:0x00000000 x28:0x00000003 x29:0x0000807a x30:0x00000000 x31:0x00000002

Vector register 0 : 0000000000000000
Vector register 1 : 0x02000e0c0a080604
Vector register 2 : 0x0c2a78665442301e
Vector register 3 : 0000000000000000
Vector register 4 : 0000000000000000
Vector register 5 : 0000000000000000
Vector register 6 : 0000000000000000
Vector register 7 : 0000000000000000
Vector register 8 : 0000000000000000
Vector register 9 : 0000000000000000
Vector register 10 : 0000000000000000
Vector register 11 : 0000000000000000
Vector register 12 : 0000000000000000
Vector register 13 : 0000000000000000
Vector register 14 : 0000000000000000
Vector register 15 : 0000000000000000
Vector register 16 : 0000000000000000
Vector register 17 : 0000000000000000
Vector register 18 : 0000000000000000
Vector register 19 : 0000000000000000
Vector register 20 : 0x0c2a78665442301e
Vector register 21 : 0x0c2a78665442301e
Vector register 22 : 0x0c2a78665442301e
Vector register 23 : 0x0c2a78665442301e
Vector register 24 : 0x0c2a78665442301e
Vector register 25 : 0x0c2a78665442301e
Vector register 26 : 0000000000000000
Vector register 27 : 0000000000000000
Vector register 28 : 0000000000000000
Vector register 29 : 0000000000000000
Vector register 30 : 0000000000000000
Vector register 31 : 0000000000000000
Cache read 0/0 Cache write 0/0
Cache flush words: 0
Execution done!
translation done!
```

Comparison between Scalar & Vector Operation

> 比較一下所完成的兩組 Assembly code, 所使用的 instruction數量各為多少。

```
## Your comparison result & comment
指令數:
hw8-1:29033
hw8-2:3450
透過SIMD，整體減少了將近10倍的指令數量，差非常多!
```

hw8-1

```
Reached Halt and Catch Fire instruction!
inst:  29033 pc:    448 src line: 214
```

hw8-2

```
Reached Halt and Catch Fire instruction!
inst:   3450 pc:    408 src line: 184
```

HW8-3 Single Cycle CPU with Vector Extension

Scala Code

請放上你的程式碼並加上註解，讓 TA明白你是如何完成的。除了下方兩個 Module外同學應該還有新增的部分 (e.g., DataMem), 這些部分先不用放上來, 在更改作業時 TA會直接利用 Gitlab查看。

- `Vector_ALU.scala`

```
// Create a module for vector reg arithmetic
package aias_lab8.Single_Cycle.Datapath

import chisel3._
import chisel3.util._

object vector_ALU_op{
    val VADD_VV = 0.U
        val VMUL_VX = 1.U
}

import vector_ALU_op._



class VECTOR_ALUIO extends Bundle{
  val vector_src1    = Input(UInt(64.W))
  val vector_src2    = Input(UInt(64.W))
  val vector_ALUSel  = Input(UInt(4.W))
  val vector_out     = Output(UInt(64.W))
}

class Vector_ALU extends Module{
  val io = IO(new VECTOR_ALUIO)

  val wire_set = Wire(Vec(8,UInt(8.W)))

  //Default Value of wire_set
  wire_set.foreach{wire=>
    wire := 0.U
  }
  printf("io.vector_ALUSel  = %x\n",io.vector_ALUSel)
  switch(io.vector_ALUSel){

    // Improved version
    is(VADD_VV){
      //choise 1
      //wire_set(0) := io.vector_src1(7,0)   + io.vector_src2(7,0)
      //wire_set(1) := io.vector_src1(15,8)  + io.vector_src2(15,8)
      //wire_set(2) := io.vector_src1(23,16) + io.vector_src2(23,16)
      //wire_set(3) := io.vector_src1(31,24) + io.vector_src2(31,24)
      //wire_set(4) := io.vector_src1(39,32) + io.vector_src2(39,32)
      //wire_set(5) := io.vector_src1(47,40) + io.vector_src2(47,40)
      //wire_set(6) := io.vector_src1(55,48) + io.vector_src2(55,48)
      //wire_set(7) := io.vector_src1(63,56) + io.vector_src2(63,56)


      //choise 2
     //64bit = 8個8bit的elem
      val src1_unit = io.vector_src1.asTypeOf(Vec(8,UInt(8.W)))
      val src2_unit = io.vector_src2.asTypeOf(Vec(8,UInt(8.W)))

      wire_set.zip(src1_unit zip src2_unit).map{
```

```
                case(wire,(src1,src2)) => wire := src1 + src2

        }
      }

      is(VMUL_VX){
        val src1_unit = io.vector_src1.asTypeOf(Vec(8,UInt(8.W)))
        val src2_unit = io.vector_src2.asTypeOf(Vec(8,UInt(8.W)))
        wire_set.zip(src1_unit zip src2_unit).map{
                case(wire,(src1,src2)) => wire := src1 * src2
        }
      }

    }

    //test
    //val out = io.vector_src1 * io.vector_src2
    //printf("out  = %x\n",out)

    io.vector_out := wire_set.asUInt
    //printf("vout = %x\n",io.vector_out)

  }

```

- Vector_RegFile.scala

```
package aias_lab8.Single_Cycle.Datapath

import chisel3._
import chisel3.stage.ChiselStage


class Vector_RegFile(readPorts:Int) extends Module {
   val io    = IO(new Bundle{
   val vector_wen   = Input(Bool())
   val vector_waddr = Input(UInt(5.W))
   val vector_wdata = Input(UInt(64.W))
   val vector_raddr = Input(Vec(readPorts, UInt(5.W)))
   val vector_rdata = Output(Vec(readPorts, UInt(64.W)))
   val vector_regs  = Output(Vec(32,UInt(64.W)))
 })

  // 32 * 64 RegFile
  val init_value = Seq.fill(32)(0.U(64.W))

  //for Lab8 vadd_vv demonstration
  //val init_value = Seq(0.U(64.W)) ++ Seq("h0001020304050607".U(64.W),"h0001020

  val vector_regs = RegInit(VecInit(init_value))

  //Wiring================================================================
  //Read
  (io.vector_rdata.zip(io.vector_raddr)).map{case(data,addr)=>data:=vector_regs(

  //Write
  when(io.vector_wen) {vector_regs(io.vector_waddr) := io.vector_wdata}
  vector_regs(0) := 0.U(64.W)

  io.vector_regs := vector_regs
}
```

Simulation Result

請複製上你在使用 Chisel完成的 CPU模擬結果 (擷取 scalar register + vector register結果), 確認和 Emulator的輸出相同。

```
## Your simulation result & comment
```

CPU模擬

```
[info] [2.614] Inst:Hcf
[info] [2.614] This is the end of the program!!
[info] [2.614] ============================
[info] [2.614] Value in the RegFile
[info] [2.615] reg[00] : 00000000 reg[01] : 00000000 reg[02] : 00010000 reg[03] : 00000000 reg[04] : 00000000 reg[05] : 00000001 reg[06] : 000080ab reg[07] : 00008034
[info] [2.616] reg[08] : 00000000 reg[09] : 00000000 reg[10] : 00008000 reg[11] : 00008080 reg[12] : 00008092 reg[13] : 00000000 reg[14] : 00000000 reg[15] : 00000000
[info] [2.616] reg[16] : 00000000 reg[17] : 00000000 reg[18] : 00000000 reg[19] : 00000000 reg[20] : 00000000 reg[21] : 00000000 reg[22] : 00000000 reg[23] : 00000000
[info] [2.616] reg[24] : 00000000 reg[25] : 00000000 reg[26] : 00000000 reg[27] : 00000000 reg[28] : 00000000 reg[29] : 00000000 reg[30] : 00000000 reg[31] : 00000000
[info] [2.616]
[info] [2.617] Value in the Vector RegFile
[info] [2.617] vector_reg[00] : 0000000000000000 vector_reg[01] : 0302010007060504 vector_reg[02] : 210e2b786d5a4734 vector_reg[03] : 0000000000000000
[info] [2.617] vector_reg[04] : 0000000000000000 vector_reg[05] : 0000000000000000 vector_reg[06] : 0000000000000000 vector_reg[07] : 0000000000000000
[info] [2.617] vector_reg[08] : 0000000000000000 vector_reg[09] : 0000000000000000 vector_reg[10] : 0000000000000000 vector_reg[11] : 0000000000000000
[info] [2.618] vector_reg[12] : 0000000000000000 vector_reg[13] : 0000000000000000 vector_reg[14] : 0000000000000000 vector_reg[15] : 0000000000000000
[info] [2.618] vector_reg[16] : 0000000000000000 vector_reg[17] : 0000000000000000 vector_reg[18] : 0000000000000000 vector_reg[19] : 0000000000000000
[info] [2.618] vector_reg[20] : 0c2a78665442301e vector_reg[21] : 0c2a78665442301e vector_reg[22] : 0c2a78665442301e vector_reg[23] : 0c2a78665442301e
[info] [2.618] vector_reg[24] : 0c2a78665442301e vector_reg[25] : 0c2a78665442301e vector_reg[26] : 0000000000000000 vector_reg[27] : 0000000000000000
[info] [2.618] vector_reg[28] : 0000000000000000 vector_reg[29] : 0000000000000000 vector_reg[30] : 0000000000000000 vector_reg[31] : 0000000000000000
test top Success: 0 tests passed in 5843 cycles in 2.664268 seconds 2193.10 Hz
[info] [2.619] RAN 5838 CYCLES PASSED
```

Emulator模擬

```
Reached Halt and Catch Fire instruction!
inst:   3450 pc:    408 src line: 184
x00:0x00000000 x01:0x000000a8 x02:0x00010000 x03:0x00000000 x04:0x00000000 x05:0x00000006 x06:0x00000000 x07:0x00000003
x08:0x00008000 x09:0x00008080 x10:0x00000000 x11:0x00000000 x12:0x00000000 x13:0x00000000 x14:0x00000000 x15:0x00000000
x16:0x00000000 x17:0x00000000 x18:0x00008092 x19:0x00000008 x20:0x00000006 x21:0x00000003 x22:0x00000002 x23:0x000080ba
x24:0x00000002 x25:0x00000008 x26:0x00000011 x27:0x00000000 x28:0x00000003 x29:0x0000807a x30:0x00000000 x31:0x00000002

Vector register 0 : 0000000000000000
Vector register 1 : 0x02000e0c0a080604
Vector register 2 : 0x0c2a78665442301e
Vector register 3 : 0000000000000000
Vector register 4 : 0000000000000000
Vector register 5 : 0000000000000000
Vector register 6 : 0000000000000000
Vector register 7 : 0000000000000000
Vector register 8 : 0000000000000000
Vector register 9 : 0000000000000000
Vector register 10 : 0000000000000000
Vector register 11 : 0000000000000000
Vector register 12 : 0000000000000000
Vector register 13 : 0000000000000000
Vector register 14 : 0000000000000000
Vector register 15 : 0000000000000000
Vector register 16 : 0000000000000000
Vector register 17 : 0000000000000000
Vector register 18 : 0000000000000000
Vector register 19 : 0000000000000000
Vector register 20 : 0x0c2a78665442301e
Vector register 21 : 0x0c2a78665442301e
Vector register 22 : 0x0c2a78665442301e
Vector register 23 : 0x0c2a78665442301e
Vector register 24 : 0x0c2a78665442301e
Vector register 25 : 0x0c2a78665442301e
Vector register 26 : 0000000000000000
Vector register 27 : 0000000000000000
Vector register 28 : 0000000000000000
Vector register 29 : 0000000000000000
Vector register 30 : 0000000000000000
Vector register 31 : 0000000000000000
Cache read 0/0 Cache write 0/0
Cache flush words: 0
Execution done!
translation done!
```