How to Translate

Java Byte Code into RISC-V Assembly Code

看這份文件的前提是，必須對於 Java bytecode 的運作模式要有一定程度的了解，且對於 RISC-V 的指令也需要一點認識。

首先，先認識一下 RISC-V 的一些暫存器：

sp : stack 指標暫存器

這跟我們後面用的 stack 是不一樣的，**接下來這份文件所講的 stack 都是透過 RISC-V 的指令去模擬 Java bytecode 的 stack**，這個暫存器所指的 stack 是 RISC-V 架構本身的機制，有興趣可再去了解是如何運作的，跟課程比較無關，便不再多加贅述，但程式開始跑之後，此暫存器裡面就會存放一個記憶體的位置，我們接下來的實作皆會根據此暫存器開始。

a0~a7： 一般暫存器

通常 RISC-V 在傳遞參數皆是使用 a 開頭的暫存器，不過就算你要用別的也是沒差啦，但這就是所謂的 calling convention，也就是使用習慣，所以等等的範例中也都是透過這些 a 開頭的暫存器。
a0： 第一個參數、a1： 第二個參數，以此類推。
回傳值通常會存在 a0 裡面。

t1~t5： 就是拿來隨便用的暫存器

沒什麼特色，但常會用到，運算的結果之類的常常會存到這類暫存器中。

s1~s11： 存取記憶體用的暫存器

其實我也不是很清楚這類暫存器本來的目的，但我都是拿來放一些記憶體位置，以便存取記憶體。

再來，在正式開始轉換 Java bytecode 之前，會需要配置一些東西。

```
.text
.section    .rodata
.align  3
```

執行 main function 的上面一定要有這些。

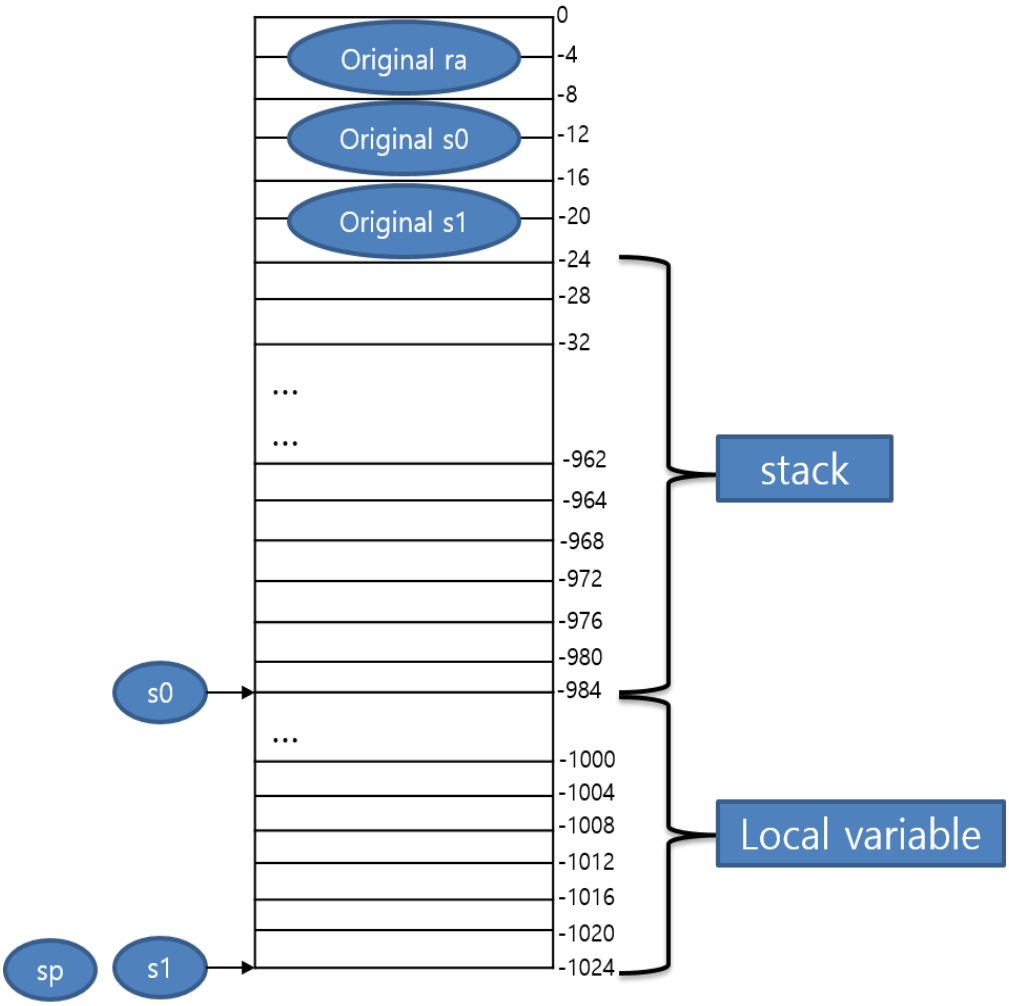
接著是每個 function 前都必須要有的東西：

```
.text
.align  1
.globl  #name
.type   #name, @function
```

再來，是配置模擬 JAVA 行為的 stack 以及 local variable：

```
addi    sp, sp, -1024
sd      ra, 1016(sp)
sd      s0, 1008(sp)
sd      s1, 1000(sp)
addi    s0, sp, 40      #stack point(240 stack entries)
addi    s1, sp, 0       #define 12 local variable
```

第一行為預留一些空間，數字可以自訂，但通常預留個 1000 多個的空間應該是非常足夠的。
第二到四行，將原本在這些記憶體中的資料先儲存起來，因為之後我們會對這些暫存器做存取，但程式結束之後，**也別忘了要把這些暫存器裡原本的值給 load 回來，以及回復 sp 的值。**
第五行為對 stack 設定大小，第六行是對 local variable 設定數量，同樣的這些數字大小也可以自行更改。**每個 function 都需要重新配置一次。**



- **整體記憶體配置示意圖(數字皆是以原本的 sp 為基準)**

接下來，是關於 function 參數傳遞的方式：

由於 java 在你呼叫 function 時，會自動將參數壓到 local variable 中，但這些過程並不會寫在 bytecode 裡面，因此，當進到 function 時，他要拿參數就會直接使用 load 指令，範例請見下面的費氏數列，因此我們若要實作出這樣子的概念，就必須自己手動將參數壓到 local variable 中。

在 RISC-V 中，參數通常都是透過 a 系列的暫存器來傳遞，因此在使用 call function 指令之前，需要多一道將參數壓入 a 系列暫存器的指令，至於要如何得知哪些是參數，看你需要幾個參數，在執行到 call function 之前參數都應該已經在你的 stack 中了，就只要把他 pop 出來就好，詳細也可以看下面的費氏數列範例。

基本上有幾個參數，可以看 java bytecode，看他在 call function 前 load 了幾次，也就是 push 了幾次到 stack。

假設現在你有三個參數在 stack 中：

```
    lw   t0,0(s0)
    addi s0,s0,-4 #pop first argument
    lw   t1,0(s0)
    addi s0,s0,-4 #pop second argument
    lw   t2,0(s0)
    addi s0,s0,-4 #pop third argument

    mv a0,t0
    mv a1,t1
    mv a2,t2

    call function
    …
    …

function:
    addi     sp,sp,-1024
    sd   ra,1016(sp)
    sd   s0,1008(sp)
    sd   s1,1000(sp)
    addi     s0,sp,40     #stack point(240 stack entries)
    addi     s1,sp,0      #define 12 local variable

    sw a0,0(s1)
    sw a1,4(s1)
    sw a2,8(s1)
    …
```

再來，若是需要配置 array，一樣需要先配置記憶體，就從 s2 開始使用，可看以下的例子。

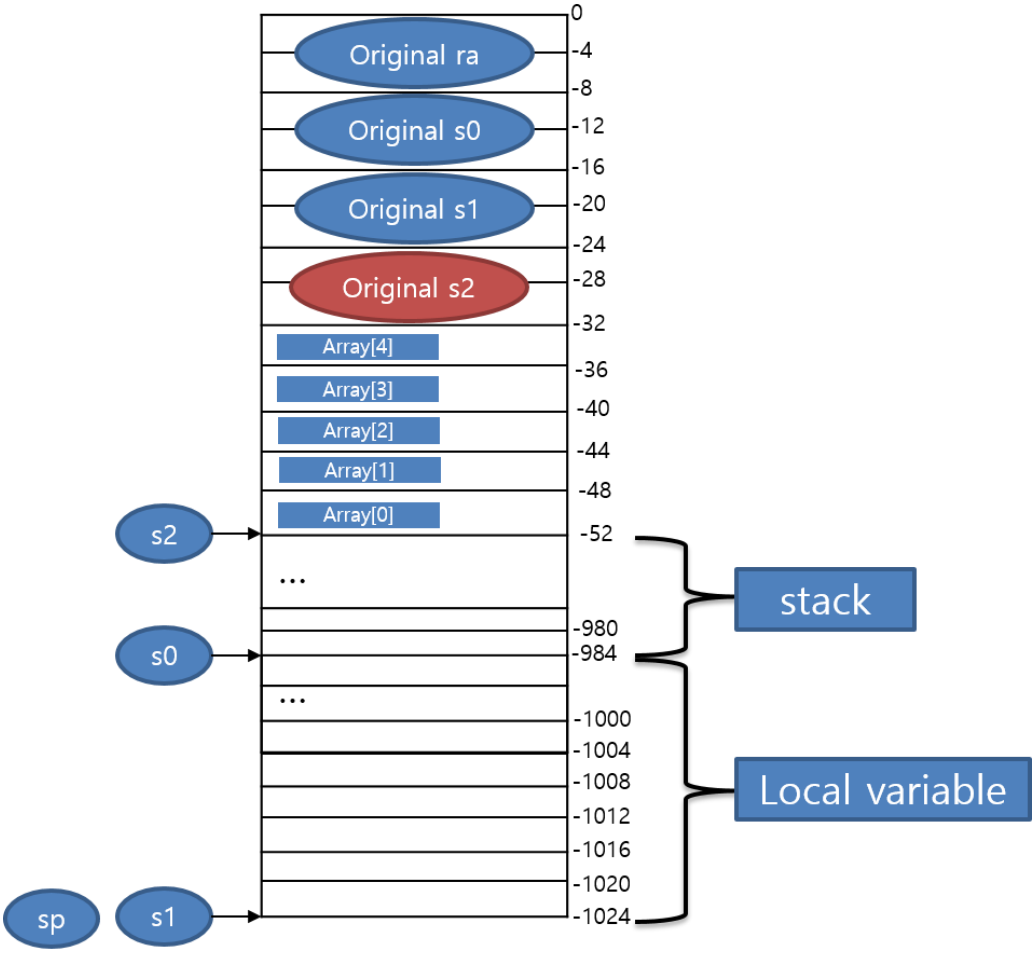
在 java bytecode 中，當看到 **newarray #type** 指令，極為配置型別為#type 的陣列，陣列的大小為目前 stack 頂端的值，因此通常在配置 array 時，此兩道指令會是一起出現的。

```
iconst_5
newarray int
```

此兩道指令即是**配製一個大小為 5 的 int 陣列**。若以上述指令為例子，對應出來的 riscv asm 為：

```
#iconst_5
li t1,5
addi s0,s0,4
sw t1,0(s0) #push constant 5 onto the stack
#newarray int
sd s2,992(sp) #反正這邊就是需要 8 的空間
lw t1,0(s0)
addi s0,s0,-4 #pop form the top of stack
imul t1,t1,4
li t2,992
sub t1,t2,t1
add s2,sp,t1
```

因此，若加入了一個陣列，整體記憶體配置的情況就變成：



●加入陣列後，原本的 stack 空間會被稍稍壓縮

最後，是 Global Variable 的部分，在本次 project 中，直接將 global variable 視為是 main function 的 local variable，在呼叫 function 的時候可將其透過類似傳遞參數的方法丟過去，接著在 function 中再繼續配置一塊記憶體空間，專門接收這樣的變數。

此外，多層次 procedure 的部分，只需要將最外層的變數傳遞下去就夠了。比如第二層的變數就不需要傳到第三層去，這樣比較好處理。

Java Bytecode to RISC-V Asm Instruction Mapping Table	
Java bytecode	RISC-V asm
Arithmetic ISA	

iadd	lw a1, 0(s0) #pop from the top of stack
	addi s0,s0,-4 #move the pointer of stack
	lw a2, 0(s0)
	addi s0,s0,-4
	add a3, a1, a2
	sw a3, 0(s0) #store the result to stack
	addi s0,4

isub	lw a1, 0(s0)
	addi s0,s0,-4
	lw a2, 0(s0)
	addi s0,s0,-4
	sub a3, a1, a2
	sw a3, 0(s0)
	addi s0,4

imul	lw a1, 0(s0)
	addi s0,s0,-4
	lw a2, 0(s0)
	addi s0,s0,-4
	mul a3, a1, a2
	sw a3, 0(s0)
	addi s0,4

idiv	lw a1, 0(s0)
	addi s0,s0,-4
	lw a2, 0(s0)
	addi s0,s0,-4
	div a3, a1, a2
	sw a3, 0(s0)
	addi s0,4

irem	lw a1, 0(s0)
	addi s0,s0,-4
	lw a2, 0(s0)
	addi s0,s0,-4
	rem a3, a1, a2
	sw a3, 0(s0)
	addi s0,4

Load/Store ISA	
iload_#index	lw a3, 4*[#index](s1) #load int value from local variable table **sw a3, 0(s0)** #store the int value to the top of stack

	addi s0,s0,-4	#move the pointer of stack

istore_#index	lw a3, 0(s0)
	addi s0,s0,-4
	sw a3, 4*[#index](s1)
	table
		#pop from the top of stack
		#move the pointer of stack
		#store the popped value to local variable

Jump ISA		
goto LABEL	**j LABEL**	

ifeq LABEL	lw a2, 0(s0)
	addi s0,s0,-4
	beq a2,zero,LABEL
		#pop from the top of stack
		#move the pointer of stack

ifge LABEL	lw a2, 0(s0)
	addi s0,s0,-4
	bge a2, zero,LABEL	

ifgt LABEL	lw a2, 0(s0)
	addi s0,s0,-4
	bgt a2, zero,LABEL	

ifle LABEL	lw a2, 0(s0)
	addi s0,s0,-4
	ble a2, zero,LABEL	

iflt LABEL	lw a2, 0(s0)
	addi s0,s0,-4
	blt a2, zero,LABEL	

ifne LABEL	lw a2, 0(s0)
	addi s0,s0,-4
	bne a2, zero,LABEL	

Additional		

iconst_#num	#push constant #num onto the stack
	li t1,#num
	addi s0,s0,4
	sw t1,0(s0)	

Example : Simple print

```
#include <stdio.h>

int main(){
    int x,y,z;
    x=1;
    y=2;
    z=x+y;

    if(z>=0){
        printf(“z=%d\n”,z);
    }
}
```

JAVA bytecode

```
public class test {
        public test();
        public static void main(java.lang.String[])
            Code:
              0: iconst_1
              1: istore_1
              2: iconst_2
              3: istore_2
              4: iload_1
              5: iload_2
              6: iadd
              7: istore_3
              8: iload_3
              9: iflt 24
              12: getstatic #7 # Field java/lang/System.out:Ljava/io/PrintStream;
              15: iload_3
              16: invokedynamic #13, 0
              21: invokevirtual #17 # print
              24: return
}
```

RISC-V asm

```
    .text
    .section    .rodata
    .align  3
.LC0:
    .string "z=%d\n"
    .text
    .align  1
    .globl  main
    .type   main, @function
main:
    addi    sp,sp,-1024
    sd  ra,1016(sp)
    sd  s0,1008(sp)
    sd  s1,1000(sp)
    addi    s0,sp,40    #stack point(240 stack entries)
    addi    s1,sp,0     #define 12 local variable

    #load 1 into local variable_0
    li  t1,1
    addi s0,s0,4
    sw  t1,0(s0) #push constant 1 onto the stack

    lw  t1,0(s0) #pop from the top of the stack
    addi s0,s0,-4
    sw  t1,0(s1) #store the value into local variable_1

    #load 2 into local variable_1
    li  t1,2
    addi s0,s0,4
    sw  t1,0(s0)

    lw  t1,0(s0)
    addi s0,s0,-4
    sw  t1,4(s1)

    #push 2 local variable onto the stack
    lw  t1,0(s1)
    addi s0,s0,4
    sw  t1,0(s0)

    lw  t1,4(s1)
```

```
    addi s0,s0,4
    sw  t1,0(s0)

    #add the top 2 numbers of the stack
    lw  t1,0(s0)
    addi s0,s0,-4
    lw  t2,0(s0)
    addi s0,s0,-4
    add t3,t1,t2

    #store the result into local variable_2
    sw  t3,8(s0)

    #push local variable_2 onto the stack
    lw  t1,8(s0)
    addi s0,s0,4
    sw  t1,0(s0)

    # "jump" if the top value on the stack is smaller then zero
    lw  t1,0(s0)
    addi s0,s0,-4
    blt t1,zero,.L2

    #if didn't jump, the print the local variable_2
    lw  a5,8(s0)
    mv  a1,a5
    lui a5,%hi(.LC0)
    addi    a0,a5,%lo(.LC0)
    call    printf

.L2:
    #recover some register setting
    li  a5,0
    mv  a0,a5
    ld  ra,1016(sp)
    ld  s0,1008(sp)
    ld  s1,1000(sp)
    addi    sp,sp,1024
    jr  ra
    .size   main, .-main
```

Example : Fibonacci

```
# Fibonacci Series using Recursion

#include <stdio.h>

int fib(int n)
{
    if (n <= 1)
        return n;
    return fib(n - 1) + fib(n - 2);
}

int main()
{
    int n = 9;
    printf("fib(9)=%d", fib(n));
    return 0;
}
```

JAVA bytecode

```
public class test {

  public static int fib(int);
    Code:
       0: iload_0
       1: iconst_1                          #push const “1” into stack

       2: if_icmpgt     7
       5: iload_0
       6: ireturn

       7: iload_0
       8: iconst_1
       9: isub
      10: invokestatic  #2                  # Method fib:(I)I

      13: iload_0
      14: iconst_2
      15: isub
      16: invokestatic  #2                  # Method fib:(I)I

      19: iadd
      20: ireturn

  public static void main(java.lang.String[]);
    Code:
       0: bipush        9
       2: istore_1
       6: iload_1
       7: invokestatic  #2                  # Method fib:(I)I
      10: invokevirtual #4                  # Method print java/io/PrintStream.println:(I)V
      13: return
}
```

RISC-V asm

```
    .text
    .align  1
    .globl   fib
    .type    fib, @function
fib:
    addi sp,sp,-48
    sd   ra,40(sp)
    sd   s0,32(sp)
    sd   s1,24(sp)
    addi s0,sp,8 #stack point
    addi  s1,sp,0 #local variable

    #store the argument in local variable_0
     sw  a0,0(s1)

    #(iload_0)push the argument onto the stack
    lw   t1,0(s1)
    addi s0,s0,4
    sw   t1,0(s0)

    #(iconst_1)push const 1 onto the stack
    li t1,1
    addi s0,s0,4
    sw   t1,0(s0)

    #pop the top 2 value from the stack
    lw   t1,0(s0)
    addi s0,s0,-4
    lw   t2,0(s0)
    addi s0,s0,-4

    #compare 2 values
    bgt  t2,t1,.L2

    #(iload_0)push argument onto the stack
    mv   t1,t0
    addi s0,s0,4
    sw   t1,0(s0)
    j    .L3

.L2:
```

```
    #(iload_0)push argument onto the stack
    lw  t1,0(s1)
    addi s0,s0,4
    sw  t1,0(s0)

    #(iconst_1)push const 1 onto the stack
    li t1,1
    addi s0,s0,4
    sw  t1,0(s0)

    #pop the top 2 value from the stack
    lw  t1,0(s0)
    addi s0,s0,-4
    lw  t2,0(s0)
    addi s0,s0,-4

    #(isub)parse the result of sub as argument
    sub t3,t2,t1
    addi s0,s0,4
    sw  t3,0(s0)
    lw  t1,0(s0)
    addi s0,s0,-4
    mv  t0,t1
    call fib

    #push the return number onto the stack
    mv  t1,a0
    addi s0,s0,4
    sw  t1,0(s0)



    #(iload_0)push argument onto the stack
    lw  t1,0(s1)
    addi s0,s0,4
    sw  t1,0(s0)

    #(iconst_2)push const 2 onto the stack
    li t1,2
    addi s0,s0,4
    sw  t1,0(s0)

    #pop the top 2 value from the stack
```

```
    lw   t1,0(s0)
    addi s0,s0,-4
    lw   t2,0(s0)
    addi s0,s0,-4

    #(isub)parse the result of sub as argument
    sub t3,t2,t1
    addi s0,s0,4
    sw   t3,0(s0)
    lw   t1,0(s0)
    addi s0,s0,-4
    mv   t0,t1
    call fib

    #push the return number onto the stack
    mv   t1,a0
    addi s0,s0,4
    sw   t1,0(s0)

    #pop the top 2 value from the stack
    lw   t1,0(s0)
    addi s0,s0,-4
    lw   t2,0(s0)
    addi s0,s0,-4

    add t3,t2,t1
    addi s0,s0,4
    sw   t3,0(s0)

.L3:

    #return the stack value
    lw t1,0(s0)
    addi s0,s0,-4
    mv   a0,t1

    ld   ra,40(sp)
    ld   s0,32(sp)
    ld   s1,24(sp)
    addi sp,sp,48

    jr   ra
```

```
    .size   fib, .-fib
    .section .rodata
    .align  3


.LC0:
    .string "fib(9)=%d\n"
    .text
    .align  1
    .globl  main
    .type   main, @function
main:
    addi sp,sp,-32
    sd  ra,24(sp)
    sd  s0,16(sp)
    addi s0,sp,8 #stack point

    #push 9 onto the stack
    li  t0,9
    addi s0,s0,4
    sw  t0,0(s0)

    #pop the top stack value into local variable_1
    lw  t0,0(s0)
    addi s0,s0,-4

    #parse argument and call the function
    mv  a0,t0
    call fib

    #print the return value
    mv  a5,a0
    mv  a1,a5
    lui  a5,%hi(.LC0)
    addi a0,a5,%lo(.LC0)
    call printf
    li  a5,0

    mv  a0,a5
    ld  ra,24(sp)
    ld  s0,16(sp)
    addi sp,sp,32
```

```
    jr   ra


    .size    main, .-main
```