How to Translate

Java Byte Code into RISC-V Assembly Code

You need to understand Java bytecode’s behavior before you read this.

First of all, let’s check some RISC-V registers：

sp : stack pointer register

We need use this register to imitate the Java stack behavior.

a0~a7 : normal registers

Use the registers to pass the parameter, but if you want to use other registers to pass them that is still fine. But we usually use a-reg to pass them, this is called “calling convention”.

a1 : first parameter
a2 : second parameter
, and so on

t1~t5 : temp registers

We usually store the result of the calculate into these registers.

s1~s11 : accessing memory register

I use these register to access the memory.

Next, we need to configure something before translate Java bytecode.

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

You need to write this before main function.

You need to write this before any functions (including main)：

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

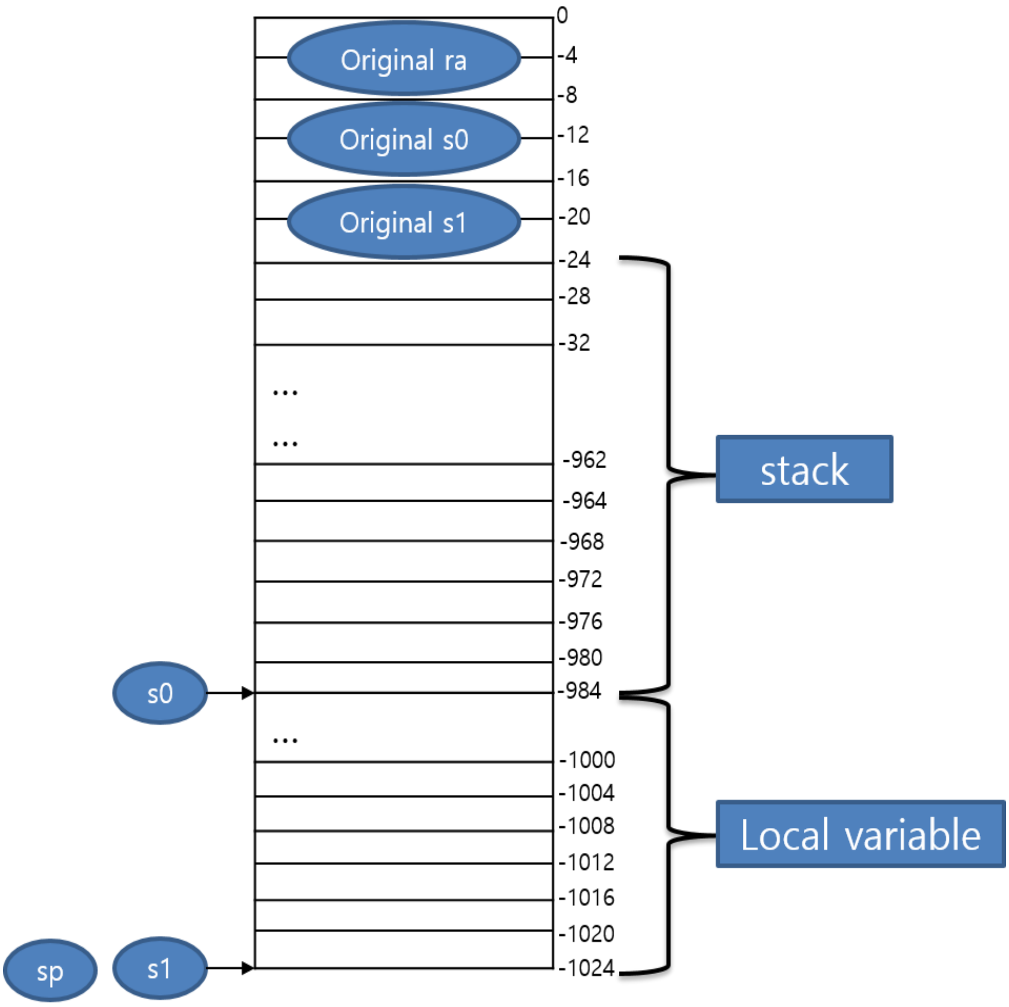
Next, this is used to configure the imitation of java stack and java local variable：

```
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
```

First Line is used to reserve some memory space.

From second to forth lines, storing the original values in these registers cause we need to access these register later. When the program end, **do not forget to load the values back.**

Fifth line is setting the stack. **Sixth line** is setting the local variable. You can adjust the size of them. **You need to do this in every function.**



- Memory configuration

Next, it’ s about pass the function arguments：

You can observe how many times that Java “load” the value onto the stack right before calling function. That’ s the number of the arguments.

For example that you have 3 arguments to pass：

```
    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)
    …
```

Next, it is about array.

You can see **newarray #type** instruction in java bytecode, it means configure a type “#type” array. The size of the array is in the top value of stack.

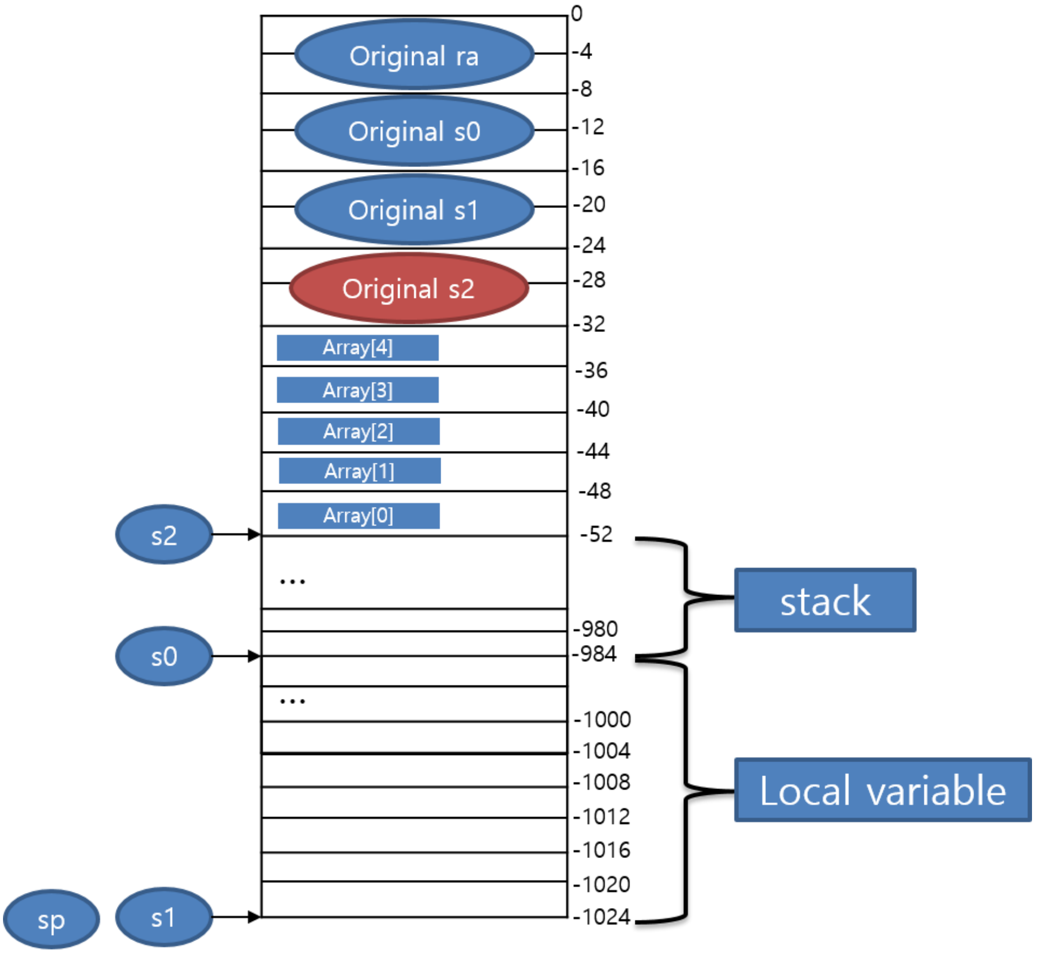
```
iconst_5
newarray int
```

These 2 instructions are meant to configure a int array, size is 5.

Its RISC-V code are like this：

```
#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) #You need some space to store the original value of s2
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
```

This is the memory configure after add an array：



Global variable, you can treat them as main function’s local variable. You can use the way to pass the global variable like the way to pass the function arguments.

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) #pop from the top of stack addi s0,s0,-4 #move the pointer of stack **sw a3, 4*[#index](s1)** #store the popped value to local variable table
Jump ISA	
goto LABEL	**j LABEL**
ifeq LABEL	lw a2, 0(s0) #pop from the top of stack addi s0,s0,-4 #move the pointer of stack **beq a2,zero,LABEL**
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
```