# Solution of Discussion04

DrinkLessMilkTea

2025 年 2 月 27 日

## 1 Review: RISC-V Memory Access

### 1.1

t0: 0x00FF0004
t1: 36
t2: 0x0000000C
s0: 0xDEADB33F
s1: 0xFFFFFFC5

### 1.2

0xF9120504: 0xABADCAF8
0xBEEFDAB0: 0x00000000
0xABADCAFC: 0x0504DAB0
0xABADCAF8: 0xB0000400

## 2 RISC-V Calling Convention

### 2.1

(a) main 和 foo 函数之间满足调用约定, 因为 main 函数保存了返回地址, 同时 foo 函数保存了需要使用的保存寄存器

(b) 5, 4, 5

(c) 5, 4, 4, 这会导致原本应该在 main 中保持不变的 s0 值发生了改变

### 2.2

(a) ra s0

(b) t1

(c) s0 ra

## 3 Recursive Calling Convention

### 3.1

```
sum_squares:
beq a0 x0 zero_case
# To be implemented in the next question
zero_case:
mv a0 a1
jr ra
```

### 3.2

```
sum_squares:
beq a0 x0 zero_case
# To be implemented in the next question
mv t0 a0
jal ra square
add a1 a0 a1
addi a0 t0 -1
jal ra sum_squares
jr ra

zero_case:
mv a0 a1
jr ra
```

### 3.3

在调用 square 函数之后, a0 中是原先 a0 中的值的平方, a1 中的值不确定, a1 可能导致错误, 因为在调用 square 之前没有保存 a1 的值, 类似的 t0 也可能产生错误

### 3.4

在调用 sum_squares 后, a0 是最终累加的值, a1 的值不确定

### 3.5

```
sum_squares:
beq a0 x0 zero_case
mv t0 a0
addi sp sp -12
sw a1 0(sp)
sw t0 4(sp)
sw ra 8(sp)
jal ra square
lw a1 0(sp)
lw t0 4(sp)
lw ra 8(sp)
addi sp sp 12
add a1 a0 a1
addi a0 t0 -1
addi sp sp -4
sw ra 0(sp)
jal ra sum_squares
lw ra 0(sp)
addi sp sp 4
jr ra

zero_case:
mv a0 a1
```

```
jr ra
```

### 3.6

没有, 由于没有使用 s 寄存器, 所以被调用者不需要保存任何寄存器, 但是最好在跳转到 zero_case 之前保存 ra 的值