INTRODUCTION TO RISC-V

RISC-V入门教程

ISA ｜ 汇编指令 ｜ 系统编程 ｜ 组成原理 ｜ 嵌入式应用

User Level编程练习

主讲 邢建国

中国开放指令生态（RISC-V）联盟｜浙江中心
China RICS-V Alliance | Zhejiang Center

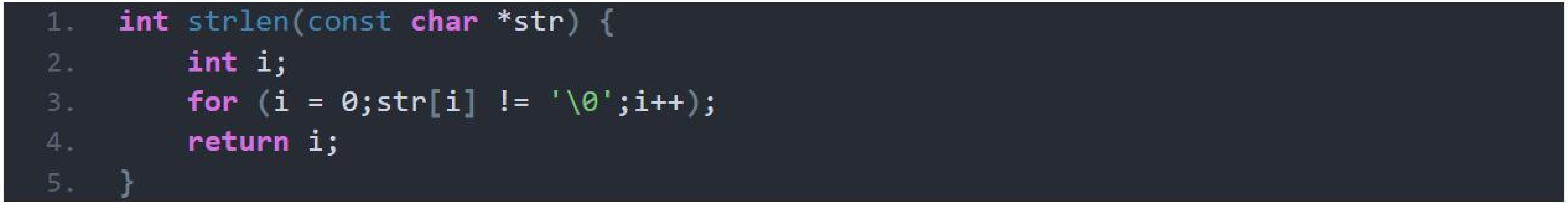
浙江图灵算力研究院
ZHEJIANG TURING INSTITUTE

RISC-V®

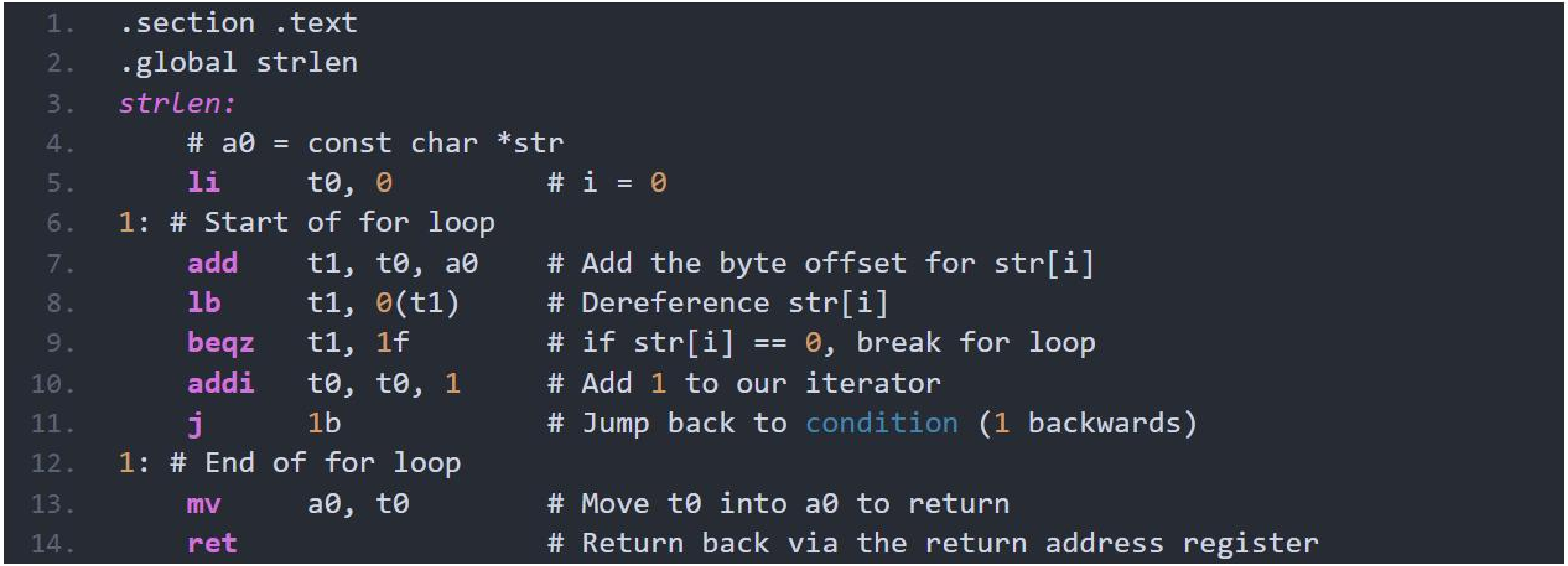
User Level编程练习

- 字符串操作
- 数组求和
- 冒泡排序
- 链表
- 调用C函数

字符串长度

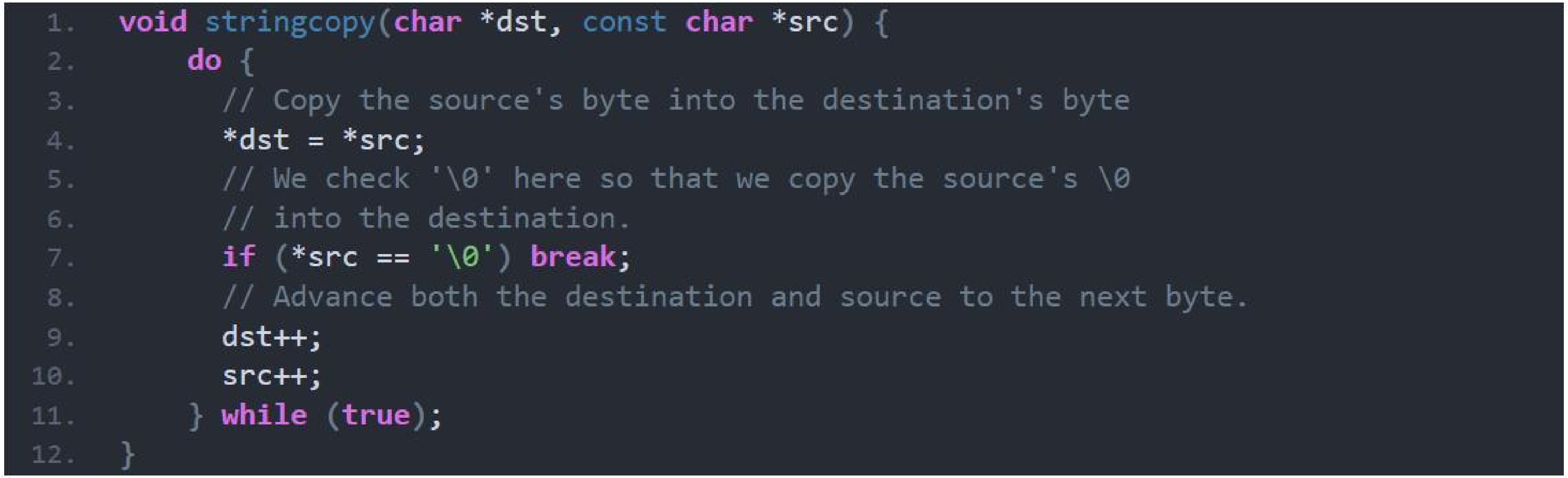


```c
int strlen(const char *str) {
    int i;
    for (i = 0;str[i] != '\0';i++);
    return i;
}
```

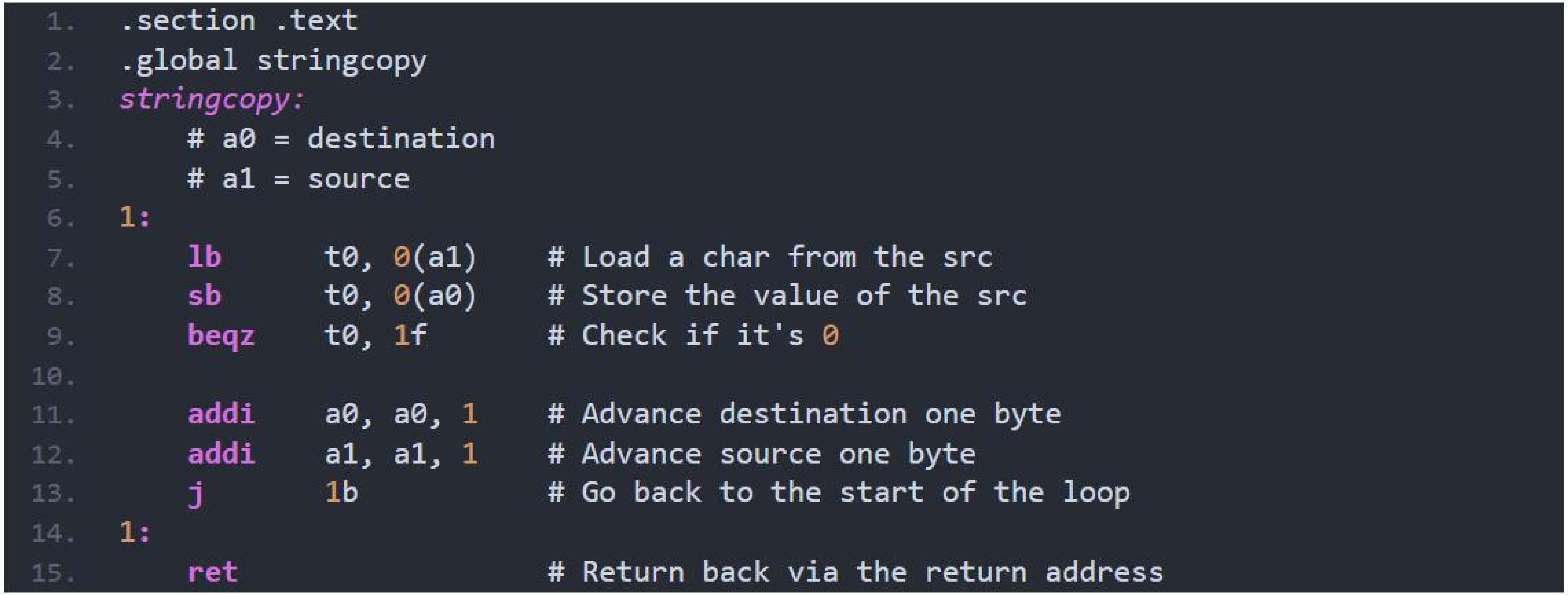


```asm
.section .text
.global strlen
strlen:
    # a0 = const char *str
    li     t0, 0         # i = 0
1: # Start of for loop
    add    t1, t0, a0    # Add the byte offset for str[i]
    lb     t1, 0(t1)     # Dereference str[i]
    beqz   t1, 1f        # if str[i] == 0, break for loop
    addi   t0, t0, 1     # Add 1 to our iterator
    j      1b            # Jump back to condition (1 backwards)
1: # End of for loop
    mv     a0, t0        # Move t0 into a0 to return
    ret                  # Return back via the return address register
```

■ 复制字符串



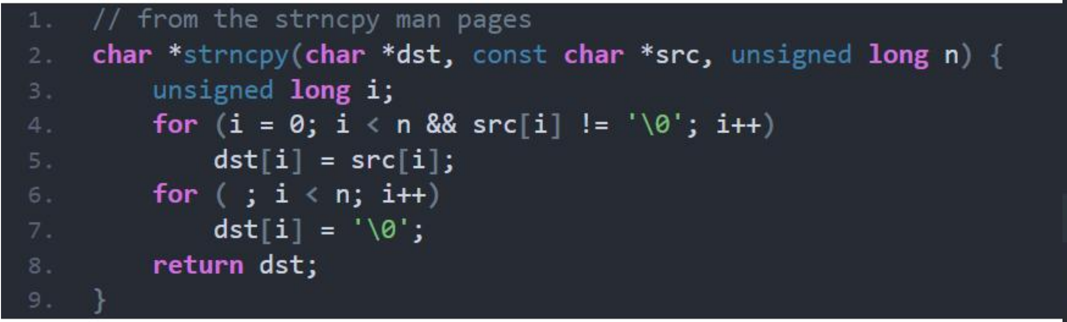
```c
void stringcopy(char *dst, const char *src) {
    do {
      // Copy the source's byte into the destination's byte
      *dst = *src;
      // We check '\0' here so that we copy the source's \0
      // into the destination.
      if (*src == '\0') break;
      // Advance both the destination and source to the next byte.
      dst++;
      src++;
    } while (true);
}
```



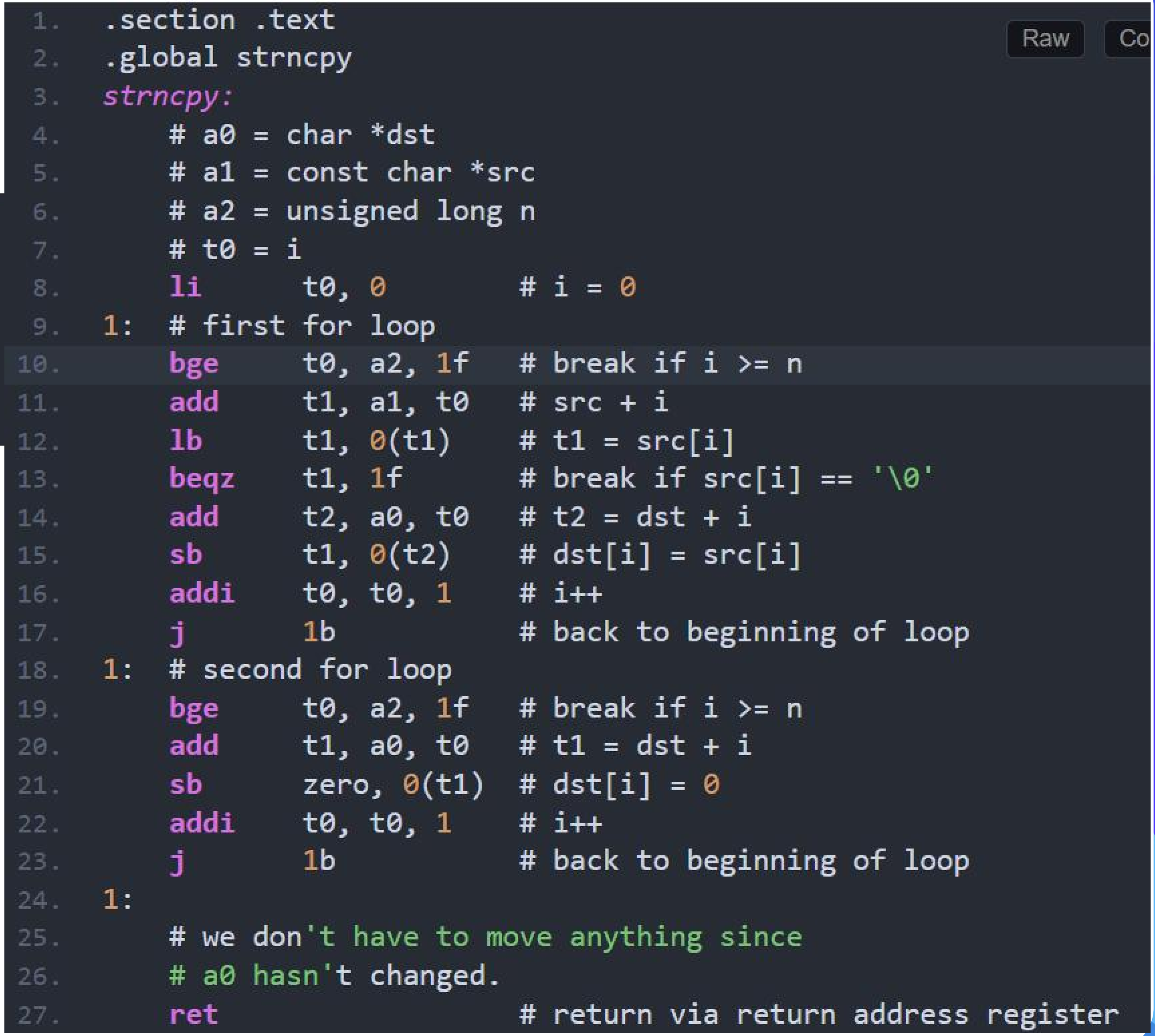
```asm
.section .text
.global stringcopy
stringcopy:
    # a0 = destination
    # a1 = source
1:
    lb      t0, 0(a1)    # Load a char from the src
    sb      t0, 0(a0)    # Store the value of the src
    beqz    t0, 1f       # Check if it's 0

    addi    a0, a0, 1    # Advance destination one byte
    addi    a1, a1, 1    # Advance source one byte
    j       1b           # Go back to the start of the loop
1:
    ret                  # Return back via the return address
```

复制n个字符

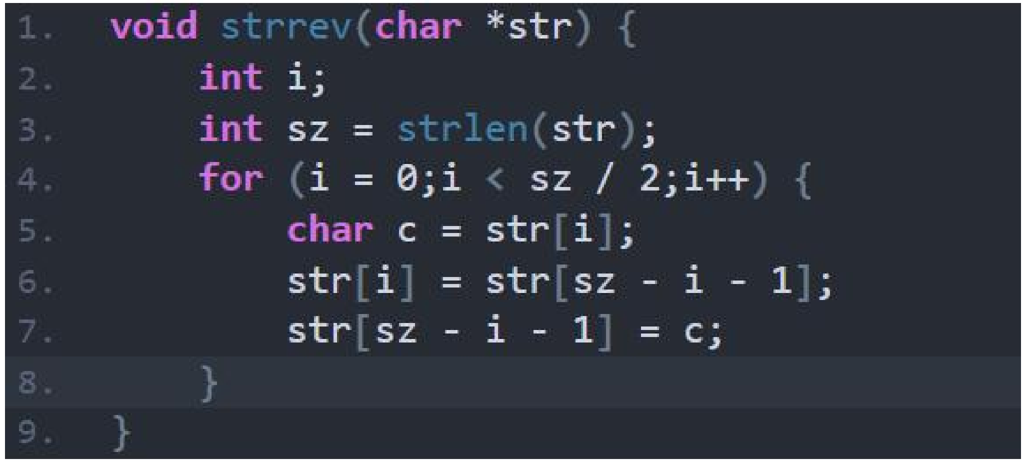


```c
// from the strncpy man pages
char *strncpy(char *dst, const char *src, unsigned long n) {
    unsigned long i;
    for (i = 0; i < n && src[i] != '\0'; i++)
        dst[i] = src[i];
    for ( ; i < n; i++)
        dst[i] = '\0';
    return dst;
}
```

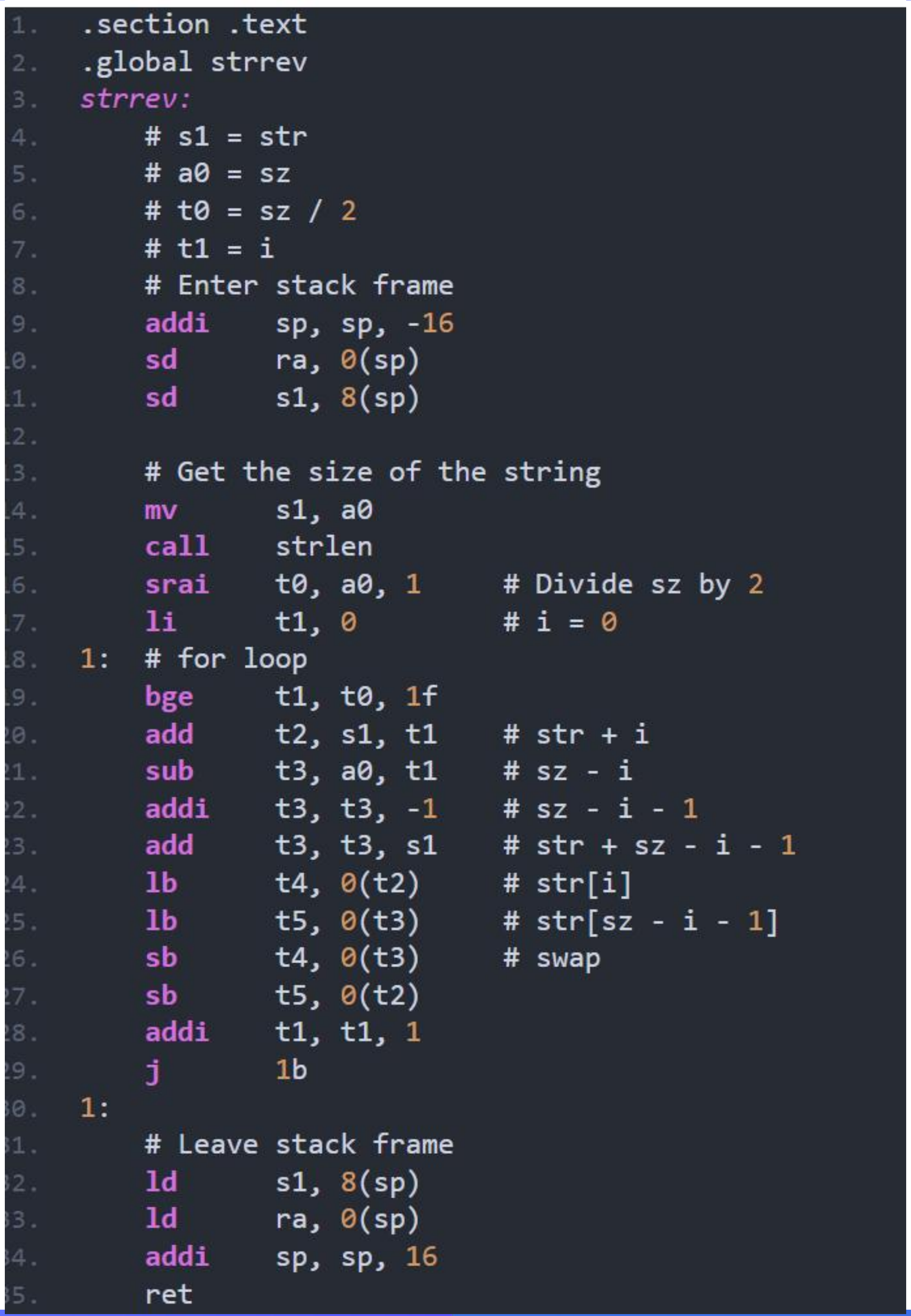


```asm
.section .text
.global strncpy
strncpy:
    # a0 = char *dst
    # a1 = const char *src
    # a2 = unsigned long n
    # t0 = i
    li      t0, 0          # i = 0
1:  # first for loop
    bge     t0, a2, 1f     # break if i >= n
    add     t1, a1, t0     # src + i
    lb      t1, 0(t1)      # t1 = src[i]
    beqz    t1, 1f         # break if src[i] == '\0'
    add     t2, a0, t0     # t2 = dst + i
    sb      t1, 0(t2)      # dst[i] = src[i]
    addi    t0, t0, 1      # i++
    j       1b             # back to beginning of loop
1:  # second for loop
    bge     t0, a2, 1f     # break if i >= n
    add     t1, a0, t0     # t1 = dst + i
    sb      zero, 0(t1)    # dst[i] = 0
    addi    t0, t0, 1      # i++
    j       1b             # back to beginning of loop
1:
    # we don't have to move anything since
    # a0 hasn't changed.
    ret                    # return via return address register
```

反转一个字符串



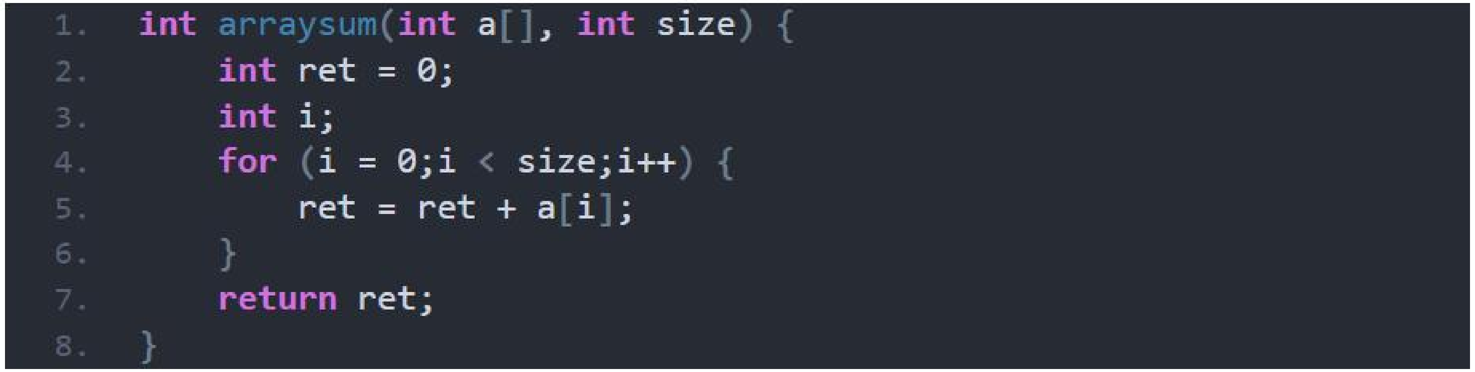
```c
void strrev(char *str) {
    int i;
    int sz = strlen(str);
    for (i = 0;i < sz / 2;i++) {
        char c = str[i];
        str[i] = str[sz - i - 1];
        str[sz - i - 1] = c;
    }
}
```



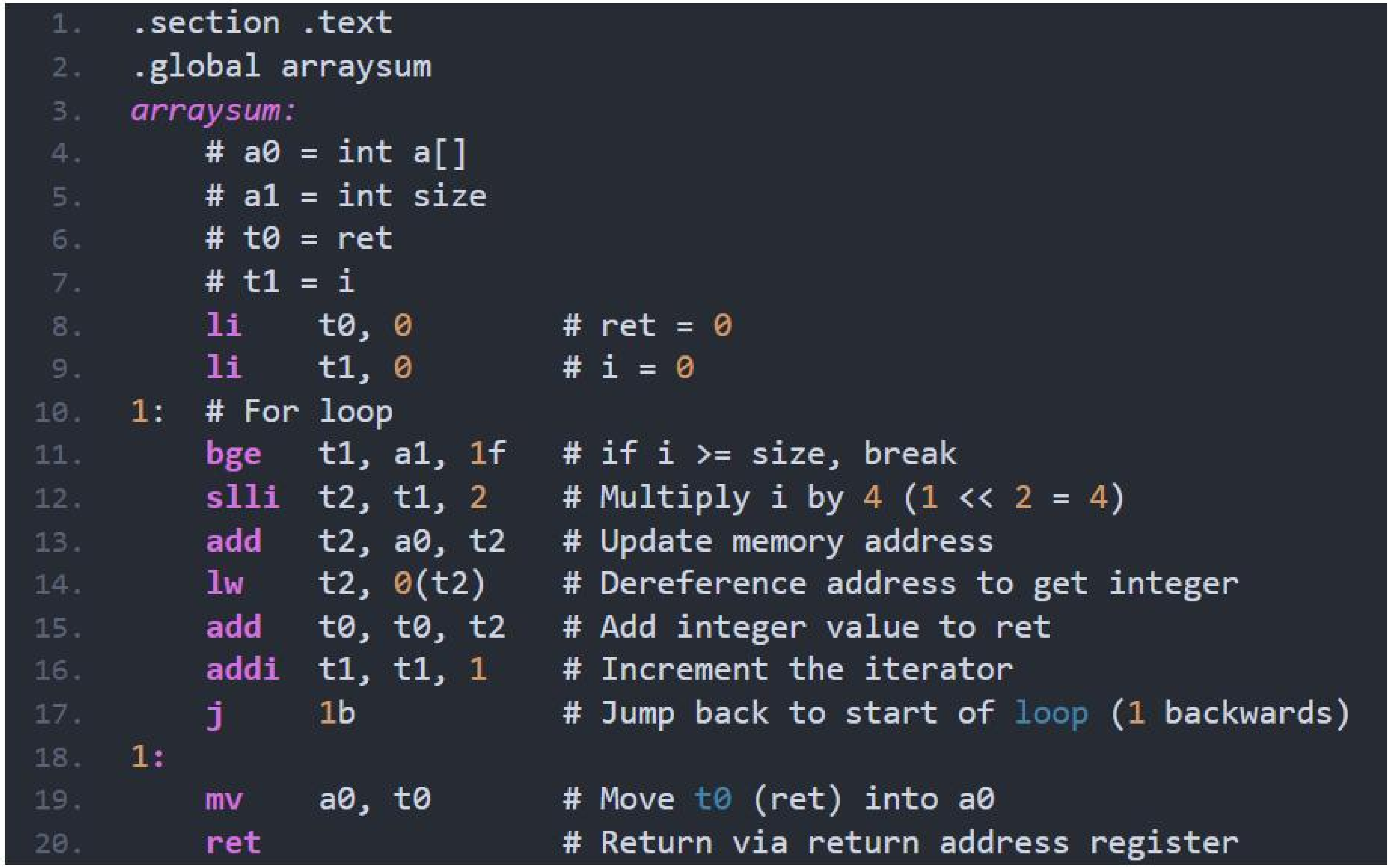
```
.section .text
.global strrev
strrev:
    # s1 = str
    # a0 = sz
    # t0 = sz / 2
    # t1 = i
    # Enter stack frame
    addi    sp, sp, -16
    sd      ra, 0(sp)
    sd      s1, 8(sp)

    # Get the size of the string
    mv      s1, a0
    call    strlen
    srai    t0, a0, 1     # Divide sz by 2
    li      t1, 0         # i = 0
1:  # for loop
    bge     t1, t0, 1f
    add     t2, s1, t1    # str + i
    sub     t3, a0, t1    # sz - i
    addi    t3, t3, -1    # sz - i - 1
    add     t3, t3, s1    # str + sz - i - 1
    lb      t4, 0(t2)     # str[i]
    lb      t5, 0(t3)     # str[sz - i - 1]
    sb      t4, 0(t3)     # swap
    sb      t5, 0(t2)
    addi    t1, t1, 1
    j       1b
1:
    # Leave stack frame
    ld      s1, 8(sp)
    ld      ra, 0(sp)
    addi    sp, sp, 16
    ret
```

■ 整数数组求和

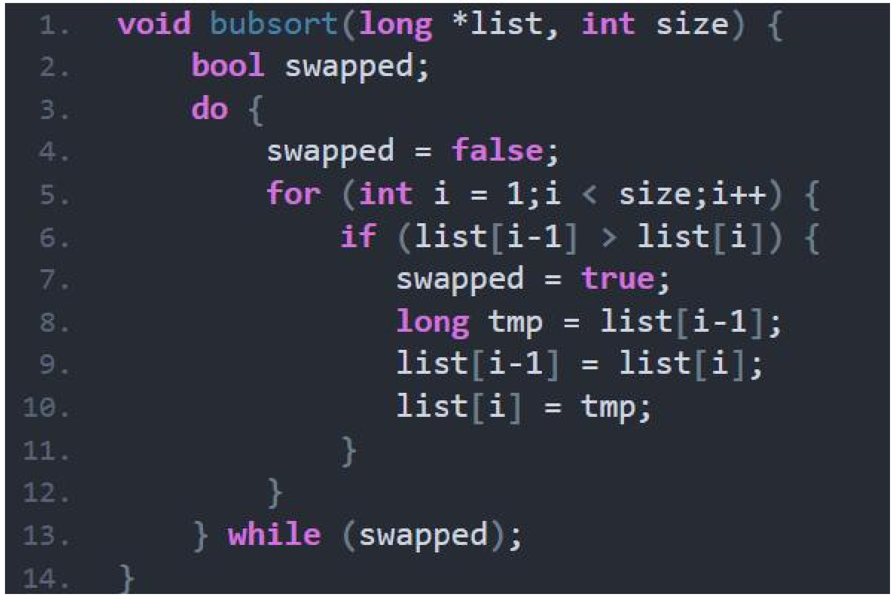


```c
int arraysum(int a[], int size) {
    int ret = 0;
    int i;
    for (i = 0;i < size;i++) {
        ret = ret + a[i];
    }
    return ret;
}
```

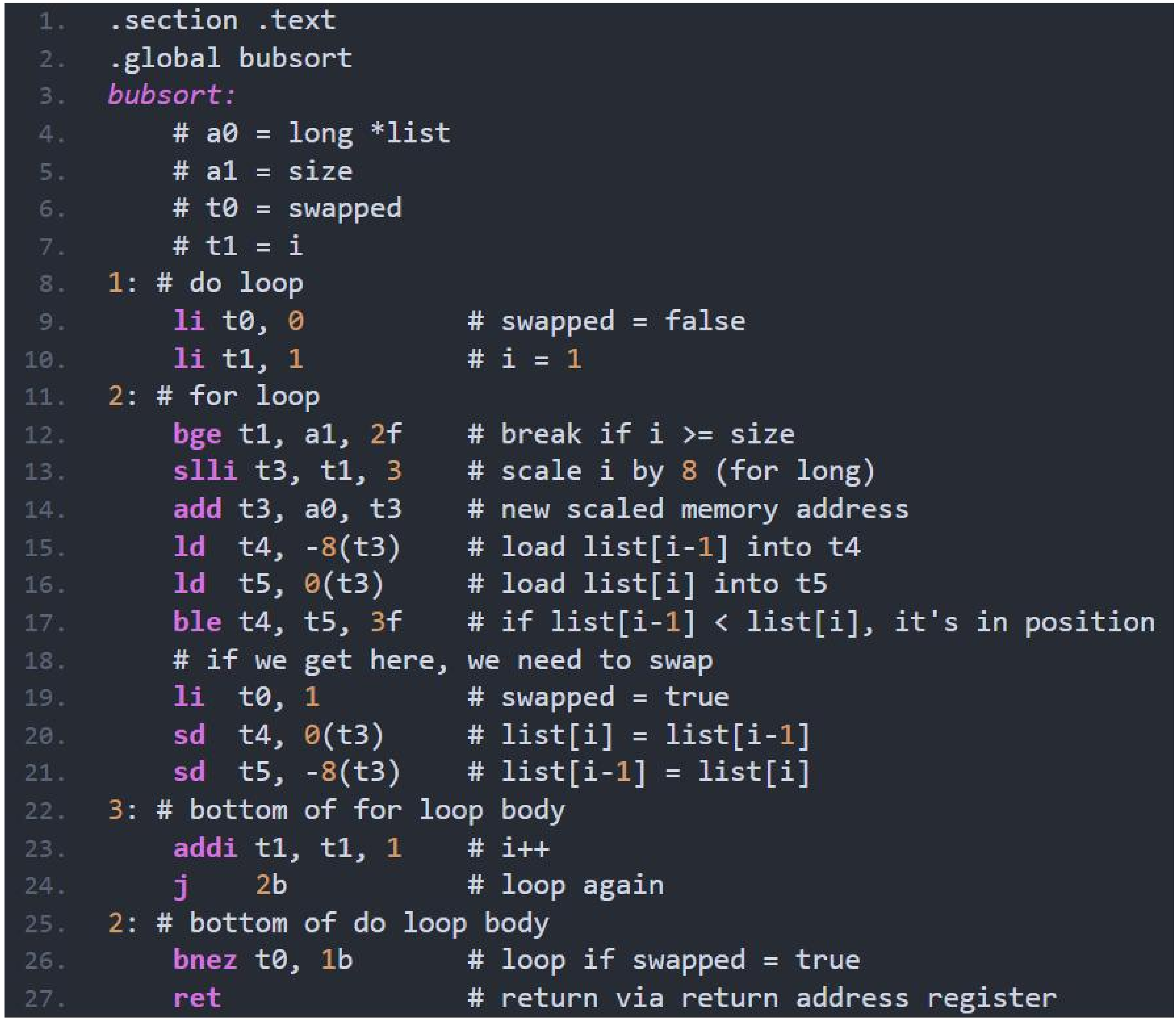


```asm
.section .text
.global arraysum
arraysum:
    # a0 = int a[]
    # a1 = int size
    # t0 = ret
    # t1 = i
    li    t0, 0        # ret = 0
    li    t1, 0        # i = 0
1:  # For loop
    bge   t1, a1, 1f   # if i >= size, break
    slli  t2, t1, 2    # Multiply i by 4 (1 << 2 = 4)
    add   t2, a0, t2   # Update memory address
    lw    t2, 0(t2)    # Dereference address to get integer
    add   t0, t0, t2   # Add integer value to ret
    addi  t1, t1, 1    # Increment the iterator
    j     1b           # Jump back to start of loop (1 backwards)
1:
    mv    a0, t0       # Move t0 (ret) into a0
    ret                # Return via return address register
```

冒泡排序

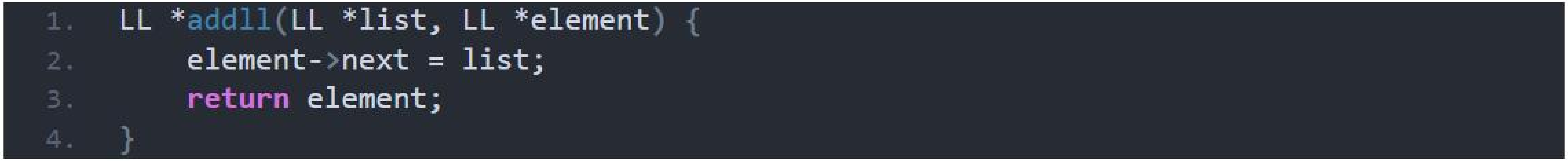


```c
void bubsort(long *list, int size) {
    bool swapped;
    do {
        swapped = false;
        for (int i = 1;i < size;i++) {
            if (list[i-1] > list[i]) {
                swapped = true;
                long tmp = list[i-1];
                list[i-1] = list[i];
                list[i] = tmp;
            }
        }
    } while (swapped);
}
```

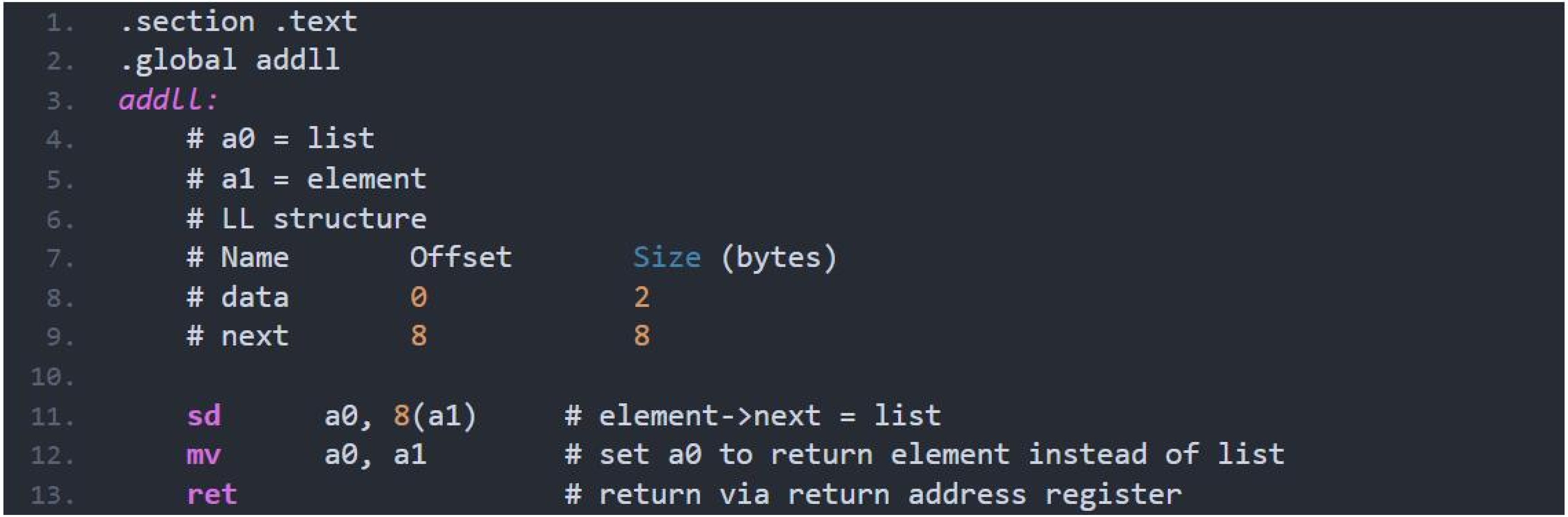


```asm
.section .text
.global bubsort
bubsort:
    # a0 = long *list
    # a1 = size
    # t0 = swapped
    # t1 = i
1: # do loop
    li t0, 0          # swapped = false
    li t1, 1          # i = 1
2: # for loop
    bge t1, a1, 2f    # break if i >= size
    slli t3, t1, 3    # scale i by 8 (for long)
    add t3, a0, t3    # new scaled memory address
    ld  t4, -8(t3)    # load list[i-1] into t4
    ld  t5, 0(t3)     # load list[i] into t5
    ble t4, t5, 3f    # if list[i-1] < list[i], it's in position
    # if we get here, we need to swap
    li  t0, 1         # swapped = true
    sd  t4, 0(t3)     # list[i] = list[i-1]
    sd  t5, -8(t3)    # list[i-1] = list[i]
3: # bottom of for loop body
    addi t1, t1, 1    # i++
    j    2b           # loop again
2: # bottom of do loop body
    bnez t0, 1b       # loop if swapped = true
    ret               # return via return address register
```

■ 单链表中插入元素



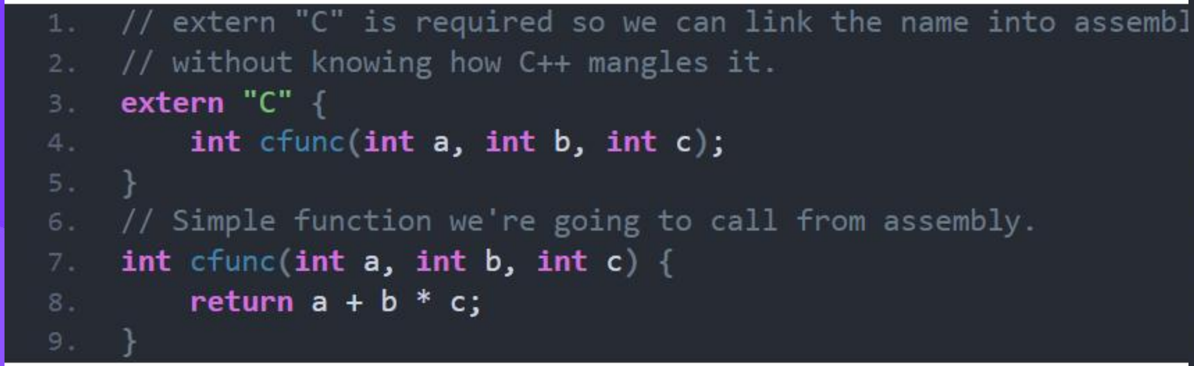
```c
LL *addll(LL *list, LL *element) {
    element->next = list;
    return element;
}
```



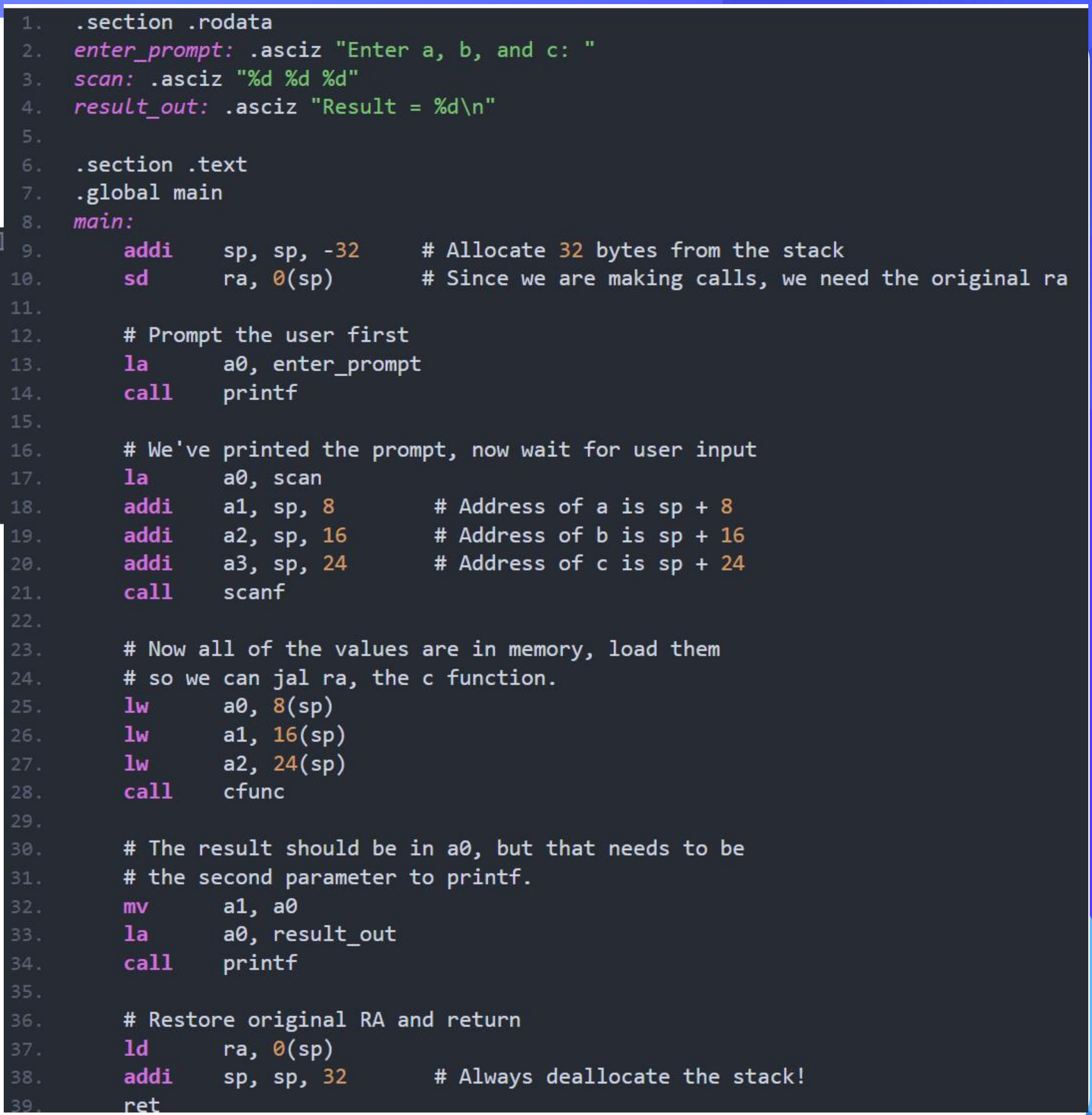
```asm
.section .text
.global addll
addll:
    # a0 = list
    # a1 = element
    # LL structure
    # Name       Offset       Size (bytes)
    # data       0            2
    # next       8            8

    sd      a0, 8(a1)     # element->next = list
    mv      a0, a1        # set a0 to return element instead of list
    ret                   # return via return address register
```

调用C函数



```cpp
// extern "C" is required so we can link the name into assembl
// without knowing how C++ mangles it.
extern "C" {
    int cfunc(int a, int b, int c);
}
// Simple function we're going to call from assembly.
int cfunc(int a, int b, int c) {
    return a + b * c;
}
```



```asm
.section .rodata
enter_prompt: .asciz "Enter a, b, and c: "
scan: .asciz "%d %d %d"
result_out: .asciz "Result = %d\n"

.section .text
.global main
main:
    addi    sp, sp, -32     # Allocate 32 bytes from the stack
    sd      ra, 0(sp)       # Since we are making calls, we need the original ra

    # Prompt the user first
    la      a0, enter_prompt
    call    printf

    # We've printed the prompt, now wait for user input
    la      a0, scan
    addi    a1, sp, 8        # Address of a is sp + 8
    addi    a2, sp, 16       # Address of b is sp + 16
    addi    a3, sp, 24       # Address of c is sp + 24
    call    scanf

    # Now all of the values are in memory, load them
    # so we can jal ra, the c function.
    lw      a0, 8(sp)
    lw      a1, 16(sp)
    lw      a2, 24(sp)
    call    cfunc

    # The result should be in a0, but that needs to be
    # the second parameter to printf.
    mv      a1, a0
    la      a0, result_out
    call    printf

    # Restore original RA and return
    ld      ra, 0(sp)
    addi    sp, sp, 32       # Always deallocate the stack!
    ret
```

User Level编程示例

- 字符串长度、复制、反转
- 数组求和
- 冒泡排序
- 链表
- 调用C函数