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

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

- 输入输出
- 宏
- 变量
- 表达式
- If/If-else
- 循环
- 函数调用和返回

用户软件和系统软件

许多计算机系统将软件分为用户软件和系统软件。

系统软件（操作系统内核和设备驱动程序）是负责保护和管理整个系统的软件，包括与外围设备交互以执行输入和输出操作以及加载和调度用户应用程序以供执行。

用户软件通常仅限于对位于寄存器和主存储器上的数据执行操作。

每当用户软件需要执行需要与系统其他部分交互的过程时，例如从文件中读取数据或在计算机显示器上显示信息，它就会调用系统软件代表用户软件执行该过程。

本章开始，主要介绍用户级别编程

输入输出

计算机系统通常提供了一组基本的输入输出系统调用，用户通过这些调用与系统交互

RARS系统调用提供：
输出：在终端上打印整数、字符、字符串、浮点数等
输入：从终端上读入整数、字符、字符串、浮点数等

Hello, world!

示例：在终端上打印”hello world!”


```
1   .data
2   hello: .asciz "hello, world!"
3
4   .text
5   la a0, hello
6   li a7, 4
7   ecall
```

字符串地址（符号）

要打印的字符串（常量）

要打印的字符串地址装入a0

RARS中打印字符串系统调用功能号为4

系统调用，产生一个异常

RARS宏

宏：为一种模式匹配和替换工具，它提供了一个简单的机制来为频繁使用的指令序列命名。

这允许程序员通过调用宏来指定指令序列。这只需要在每次使用时输入一行代码，而不是每次都重复输入整个指令序列。

“一次定义，多次使用”，这不仅减少了出错的机会，也便于程序维护。

宏在这方面类似于过程（子程序、函数），但与过程的操作方式不同。
汇编语言中的程序遵循特定的协议来进行过程定义、调用和返回。
宏通过在汇编时将宏体替换为每次使用来进行操作(这种替换称为宏扩展)，不需要过程调用时的协议和执行开销。

RARS宏

宏：.macro .... .end_macro

.eqv


```
.text
.macro print_str
        li a7, 4
        ecall
.end_macro

la a0, hello
#li a7, 4
#ecall
print_str
```

```
.eqv x t0
```

RARS宏

练习：

打印一个整数（功能号1）
读入一个整数（功能号5）

打印字符（功能号11）
打印换行

RARS宏

练习：

打印一个整数（功能号1）
读入一个整数（功能号5）

打印字符（功能号11）
打印换行

```
.macro print_int
        li a7,1
        ecall
        newline
.end_macro

.macro read_int
        li a7,5
        ecall
.end_macro
```

```
.macro print_char
        li a7, 11
        ecall
.end_macro

.macro newline
        li a0, '\n'
        print_char
.end_macro
```

变量

在汇编语言中，变量可以放在寄存器中，也可以放在内存中

寄存器数量有限，只能放整数
内存可以存放更多的变量，支持复杂的数据结构（数组、结构体）

```
.data
X:   .word 10

A: .word 10,20,30, 40
```

引用变量和赋值

只能通常load和store指令来访问内存变量

引用一个变量需要先装入寄存器
修改一个变量通过store来写入

```
.data
X:   .word 10
Y:   .word 20
Z:   .word 0

.text
...
la t0, X
lw t1, 0(t0)      #或者使用 lw t1, X

la t0, X
sw t1, 0(t0)     #或者使用：sw t1, X, t0
```

复合数据

如何实现高级语言中的结构体：

```
struct {
        char *name,
        int age,
        …
} p;

print(p.name );
p.age = p.age + 5;
```

```
p: .word name   #name
   .word 5      #age

name: .asciz "张三"
```

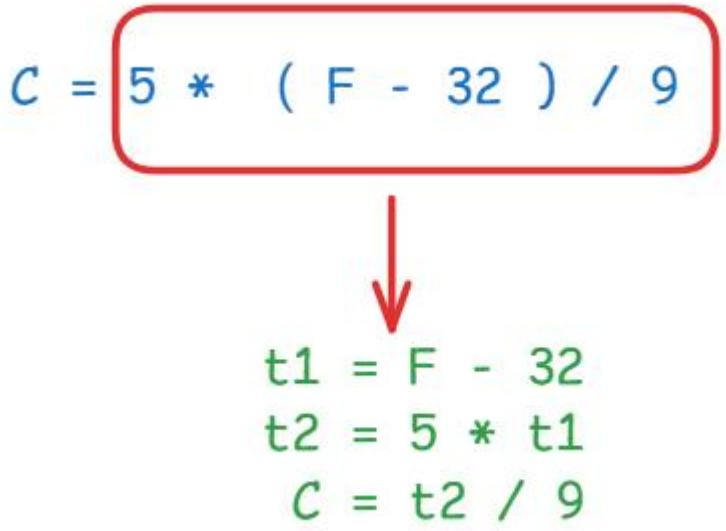
表达式

a + b * c
⇒ t1 = b * c
⇒ t2 = t1 + a

表达式

练习

温度转换：celsius = 5 * (fahr - 32) / 9



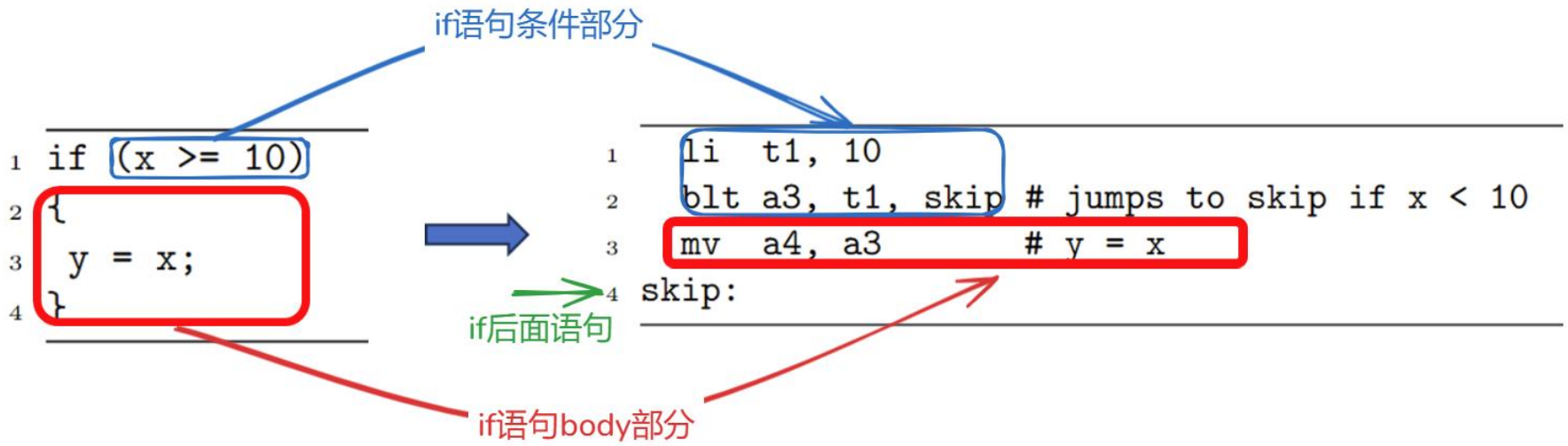
控制流语句

在RV32I汇编语言中，只有条件转移指令（beq, …）和无条件转移指令(jal、jalr)

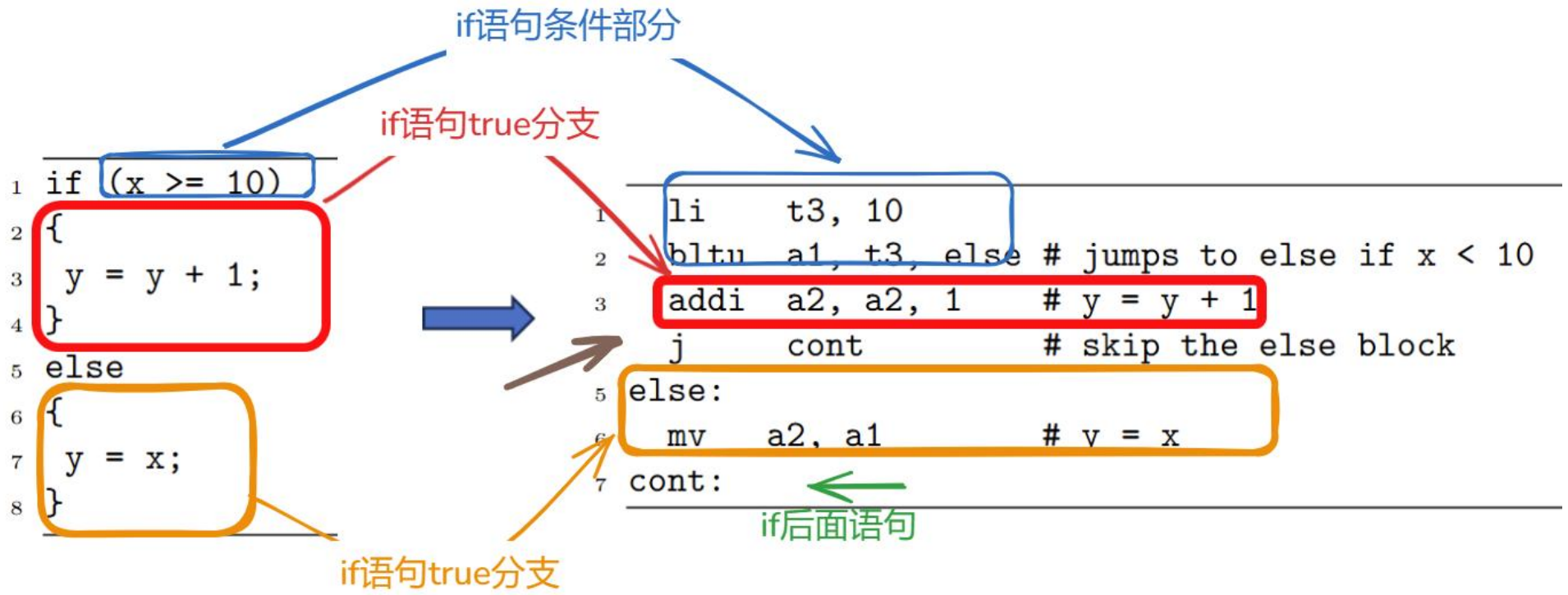
如何在汇编程序中实现高级语言的：

条件语句 if 、if else
循环语句 while、for、do while
函数调用和返回return

条件语句：if … then


if语句条件部分
1 if (x >= 10)
2 {
3 y = x;
4 }
1 li t1, 10
2 blt a3, t1, skip # jumps to skip if x < 10
3 mv a4, a3 # y = x
4 skip:
if后面语句
if语句body部分

■ 条件语句：if … then … else



复合条件

条件1　　条件2

```
1 if ((x>=10) && (y<20))
  {
    x = y;
  }
```

```
1   li   t1, 10
2   blt  a1, t1, skip # jumps to skip if x < 10
3   li   t1, 20
4   bge  a2, t1, skip # jumps to skip if y >= 20
5   mv   a1, a2       # x = y
6 skip:
```

条件1

条件2

if语句true分支

条件1 && 条件2
条件1不成立，则不用测试条件2，结果为false
条件1成立，再测试条件2，如果条件2成立，则结果为true，否则为false

条件1 || 条件2
条件1成立，则不用测试条件2，结果为true
条件1不成立，再测试条件2，如果条件2成立，则结果为true，否则为false

复合条件

条件1　　条件2

```
1 if ((x>=10) || (y<20))
2 {
3   x = y;
4 }
```

```
1  li   t1, 10
2  bge  a1, t1, then # jumps to then if x >= 10
3  li   t1, 20
4  bge  a2, t1, skip # jumps to skip if y >= 20
5 then:
6  mv   a1, a2       # x = y
7 skip:
```

条件1

条件2

if语句true分支

条件1 && 条件2

条件1不成立，则不用测试条件2，结果为false

条件1成立，再测试条件2，如果条件2成立，则结果为true，否则为false

条件1 || 条件2

条件1成立，则不用测试条件2，结果为true

条件1不成立，再测试条件2，如果条件2成立，则结果为true，否则为false

嵌套if

外层if条件

```
1 if (x == 10)
2 {
3   x = 5;
4   if (y == 20)
5   {
6     x = 0;
7   }
8 }
```

外层if条件

```
1 li   t1, 10
2 bne  a1, t1, skip # jumps to skip if x != 10
3                   # <= Insert the code for the then block
4 skip:
```

内层if条件

```
3 li   a1, 5                # x = 5
4 li   t1, 20
5 bne  a2, 20, skip_inner # jumps to skip_inner if y != 20
6 li   a1, 0                # x = 0
7 skip_inner:
```

```
1 li   t1, 10
2 bne  a1, t1, skip         # jumps to skip if x != 10
3 li   a1, 5                # x = 5
4 li   t1, 20
5 bne  a2, 20, skip_inner # jumps to skip_inner if y != 20
6 li   a1, 0                # x = 0
7 skip_inner:
8 skip:
```

循环：while

```
1 int i=0; while条件
2 while (i < 20)
3 {
4    y = y+3;
5    i = i+1;
6 }
```

while条件

```
1    li   a1, 0          # i=0
2 while:
3    li   t1, 20         # if i>=20
4    bge  a1, t1, skip   # jump to skip to leave the loop
5    addi a2, a2, 3      # y = y+3
6    addi a1, a1, 1      # i = i+1
7    j    while          # loop back
8 skip:
```

■ 循环：do ••• while

```
1 int i=0;
2 do
3 {
4   y = y+2;
5   i = i+1;
6 } while (i < 10);
```

do while条件

```
1   li   a1, 0              # i=0
2 dowhile:
3   addi a2, a2, 2          # y = y+2
4   addi a1, a1, 1          # i = i+1
5   li   t1, 10
6   blt  a1, t1, dowhile # jumps back to dowhile if i < 10
```

循环：for

```
1 for (i=0; i<10; i=i+1)
2 {
3   y = y+2;
4 }
```

```
1   li   a1, 0          # i=0
2 for:
3   li   t1, 10         # if i >= 10 then jumps
4   bge  a1, t1, skip # to skip to leave the loop
5   addi a2, a2, 2      # y = y+2
6   addi a1, a1, 1      # i = i+1
7   j    for
8 skip:
```

循环不变量外提

```
1   li   a1, 0          # i=0
2 for:
3   li   t1, 10         # if i >= 10 then jumps
4   bge  a1, t1, skip # to skip to leave the loop
5   addi a2, a2, 2     # y = y+2
6   addi a1, a1, 1     # i = i+1
7   j    for
8 skip:
```

```
1   li   a1, 0          # i=0
2   li   t1, 10         # t1=10
3 for:
4   bge  a1, t1, skip # if i >= 10 then jumps to skip to leave the loop
5   addi a2, a2, 2     # y = y+2
6   addi a1, a1, 1     # i = i+1
7   j    for
8 skip:
```

函数调用和返回

函数在汇编语言中由一个标号和一段代码定义，标号定义了函数的入口点。

调用函数只要跳转到其入口点即可。但是，在调用（跳转到）函数之前，保存返回地址，以便函数在执行后可以返回调用点。在RISC-V中，可以使用JAL指令来实现保存返回地址和跳转，通常返回地址放在寄存器ra中：JAL ra, func

函数定义

```
1 # The update_x routine
2 update_x:
3   la   t1, x
4   sw   a0, (t1)
5   ret
```

函数调用

```
1 .data
2 x: .skip 4
3
4   li  a0, 42     # loads 42 into a0
5   jal update_x   # invoke the update_x routine
```

注意：返回地址由指令自动存储在寄存器ra中。此操作破坏了寄存器ra先前的值，因此，在调用函数之前，可能需要保存此寄存器的内容，以便以后可以恢复。

函数调用和返回

如何从函数中返回值?

从函数返回值是一个约定问题，通常由应用应用二进制接口或ABI定义。RISC-V ABI定义函数必须通过将返回值存储在寄存器a0中。

示例：搜索数组的最大值

```
1  /* Global array */
2  int numbers[10];
3
4  /* Returns the largest value from array numbers. */
5  int get_largest_number()
6  {
7      int largest = numbers[0];
8      for (int i=1; i<10; i++) {
9          if (numbers[i] > largest)
10             largest = numbers[i];
11     }
12     return largest;
13 }
```

示例：搜索数组的最大值(解法1)

```
1  .data
2  # Allocate the numbers array (10 integers = 40 bytes)
3  numbers: .skip 40
4
5  .text
6  get_largest_number:
7    la a5, numbers      # a5 = &numbers
8    lw a0, (a5)         # a0 (largest) = numbers[0]
9    li a1, 1            # a1 (i) = 1
10   li t4, 10
11   for:
12    bge  a1, t4, end: # if i >= 10, then end loop
13    slli t1, a1, 2    # t1 = i * 4
14    add  t2, a5, t1   # t2 = &numbers + i*4
15    lw   t3, (t2)     # t3 = numbers[i]
16    blt  t3, a0       # if numbers[i] < largest, then skip
17    mv   a0, t3       # Update largest
18    skip:
19    addi a1, a1, 1  # i = i+1
20    j    for
21   end:
22   ret                # Return
```

示例：搜索数组的最大值（解法2）

```
1  get_largest_number:
2    la   a5, numbers       # a5: pointer to current element
3    lw   a0, (a5)          # Load first element (number[0])
4    addi a5, a5, 4         # Advance pointer to next element
5    addi a6, a5, 40        # a6 <= address after last element
6  do_while:
7    lw   a4, (a5)          # Load current element
8    bge  a0, a4, skip
9    mv   a0, a4            # Update largest
10 skip:
11   addi a5, a5, 4         # Advance pointer to next element
12   bne  a5, a3, do_while # do while a5 != a3
13   ret
```

RISC-V®

控制流

- 输入输出
- 宏
- 变量
- 表达式
- If/If-else
- 循环
- 函数调用和返回