

Tigger 文档说明

1 概述

Tigger /'tɪgə(r)/ 是面向 RISC-V 的一种中间表示，用作寄存器分配的输出格式。

为了让同学们快速熟悉 Tigger 语法，Tigger 遵循一贯的简洁易读风格，被设计得与 Eeyore 很像。

2 语法描述

2.1 寄存器

Tigger 共有 28 个可用的寄存器，这些寄存器的名称与 RISC-V 保持一致（相比 RISC-V，删去了一些不需要编译器管理寄存器）。

- x0: 该寄存器恒等于 0，不可更改
- s0-s11: 没什么特殊之处，被调用者保存。
- t0-t6: 没什么特殊之处，调用者保存。
- a0-a7: 用来传递函数参数，调用者保存。其中 a0-a1 也被用作传递函数返回值，但因为 MiniC 中所有函数返回值都是 int，所以实际上只有 a0 被用作传递返回值。

可以看出，最多只能通过寄存器传递 8 个参数。简单起见，限定所有函数参数个数不超过 8 个。

2.2 表达式、标号、跳转语句

- 所有的表达式计算都在寄存器上进行。
- 所有在 Eeyore 中支持的运算符，在 Tigger 中都支持。
- 注意！因为 MiniC 里只有 int 和 int 数组类型，所以形似数组赋值语句的赋值语句中括号内的数是 4 的倍数。
- 注意！由于 RISC-V 某些规则的原因，Tigger 中只有 '+' 和 '<' 运算符允许作为 `Reg = Reg OP2 <INTEGER>` 语句中的 OP2。
- 标号与跳转语句和 Eeyore 中的语法相同，标号是全局的。

2.3 函数

- 函数定义语句形如 `f_xxx [2] [3]`，第一个中括号内是参数个数，第二个是该函数需要用到的栈空间的大小（除以 4 之后）。
- 函数结束语句和 Eeyore 中的一样，形如 `end f_xxx`。
- 函数必须以返回语句返回。返回值通过寄存器传递。
- 函数调用语句形如 `call f_xxx`。

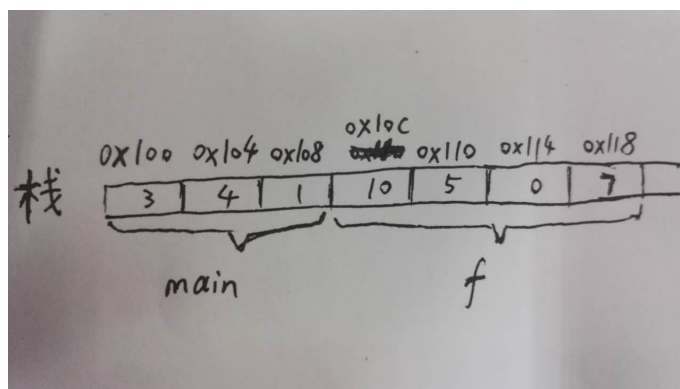
2.4 栈内存操作

程序运行时，每个被调用的函数都会维护一个连续的栈空间，大小为函数定义语句中的第二个参数。

局部变量都可以在栈中找到，因此 Tigger 中不再有局部变量了。

- `store Reg <INTEGER>` 语句中，`<INTEGER>` 是一个小于函数定义语句第二个系数的非负整数。该语句会把寄存器 `<Reg>` 的值存入当前函数栈空间第 `<INTEGER>` 个位置。
- `load <INTEGER> Reg` 语句中，`<INTEGER>` 是一个小于函数定义语句第二个系数的非负整数。该语句会把当前函数栈空间第 `<INTEGER>` 个整数存入寄存器 `<Reg>`。
- `loadaddr <INTEGER> Reg` 语句中，`<INTEGER>` 是一个小于函数定义语句第二个系数的非负整数。该语句会把当前函数栈空间第 `<INTEGER>` 个位置的内存地址存到寄存器 `<Reg>`。

举个例子，假设某个时刻函数调用关系是 `main[0][3] -> f[0][4]`，正在执行函数 `f`，假设此时的栈如下图所示：



此时语句 `load 2 s0`，会使 `s0 = 5`；语句 `loadaddr 2 s0`，会使 `s0 = 0x110`；语句 `store s0 2` 会把图中的 5 改成 `s0` 的值。

2.5 全局变量

- 全局变量名称以 `v` 开头，后接一个整数编号，编号从 0 开始，比如 `v0, v1`。
- `<VARIABLE> = <INTEGER>` 用来声明一个初始值为 `<INTEGER>` 的全局变量 `<VARIABLE>`，即 `<VARIABLE>` 这个名称表示的内存地址上 4 字节的内容为 `<INTEGER>`。

- `<VARIABLE> = malloc <INTEGER>` 用来声明数组, `<VARIABLE>` 这个名称表示的内存地址之后的 `<INTEGER>` 字节的内容为一个数组。注意! `<INTEGER>` 是 4 的倍数。
- `load <VARIABLE> Reg` 表示把 `<VARIABLE>` 这个全局变量对应内存地址上 4 字节的内容加载到寄存器 `Reg`。
- `loadaddr <VARIABLE> Reg` 表示把 `<VARIABLE>` 这个全局变量对应内存地址加载到寄存器 `Reg`。
- 注意! 由于 RISC-V 汇编的原因, 没有 `store Reg <VARIABLE>` 语句。该语句可以通过 `loadaddr` 语句与数组访问语句结合来完成。

2.6 注释

Tigger 允许单行注释, 与 C 语言注释类似使用 `//`, 处理时自动忽略改行从 `//` 之后所有内容。

2.7 系统库支持

与 MiniC 和 Eeyore 中的输入输出函数原型相同。

四种输入输出函数都通过 `a0` 寄存器传递参数和返回值。

3 BNF

```
 $\langle Goal \rangle ::= (\text{FunctionDecl} \mid \text{GlobalVarDecl})^*$   
 $\langle GlobalVarDecl \rangle ::= \langle VARIABLE \rangle '=' \langle INTEGER \rangle$   
                   $\mid \langle VARIABLE \rangle '=' 'malloc' \langle INTEGER \rangle$   
 $\langle FunctionDecl \rangle ::= \text{Function } '[' \langle INTEGER \rangle ']' '[' \langle INTERGER \rangle ']' (\text{Expression})^* \text{'end'}$   
                  Function  
 $\langle Expression \rangle ::= \text{Reg } '=' \text{Reg OP2 Reg}$   
                   $\mid \text{Reg } '=' \text{Reg OP2 } \langle INTEGER \rangle$   
                   $\mid \text{Reg } '=' \text{OP1 Reg}$   
                   $\mid \text{Reg } '=' \text{Reg}$   
                   $\mid \text{Reg } '=' \langle INTEGER \rangle$   
                   $\mid \text{Reg } '[' \langle INTEGER \rangle ']' = \text{Reg}$   
                   $\mid \text{Reg} = \text{Reg } '[' \langle INTEGER \rangle ']'$   
                   $\mid \text{'if' Reg LogicalOP Reg 'goto' Label}$   
                   $\mid \text{'goto' Label}$   
                   $\mid \text{Label ':'}$   
                   $\mid \text{'call' Function}$   
                   $\mid \text{'store' Reg } \langle INTEGER \rangle$   
                   $\mid \text{'load' } \langle INTEGER \rangle \text{Reg}$   
                   $\mid \text{'load' } \langle VARIABLE \rangle \text{Reg}$   
                   $\mid \text{'loadaddr' } \langle INTEGER \rangle \text{Reg}$   
                   $\mid \text{'loadaddr' } \langle VARIABLE \rangle \text{Reg}$   
                   $\mid \text{'return'}$   
 $\langle Reg \rangle ::= \text{'x0' } \mid \text{'s0' } \mid \text{'s1' } \mid \text{'s2' } \mid \text{'s3' } \mid \text{'s4' } \mid \text{'s5' } \mid \text{'s6' } \mid \text{'s7' } \mid \text{'s8'}$   
                   $\mid \text{'s9' } \mid \text{'s10' } \mid \text{'s11' } \mid \text{'a0' } \mid \text{'a1' } \mid \text{'a2' } \mid \text{'a3' } \mid \text{'a4' } \mid \text{'a5'}$   
                   $\mid \text{'a6' } \mid \text{'a7' } \mid \text{'t0' } \mid \text{'t1' } \mid \text{'t2' } \mid \text{'t3' } \mid \text{'t4' } \mid \text{'t5' } \mid \text{'t6'}$   
 $\langle Label \rangle ::= \langle LABEL \rangle$   
 $\langle Function \rangle ::= \langle FUNCTION \rangle$ 
```

4 示例

```
f_fac [1] [3]
    a0 = a0 + -1
    if a0 <= x0 goto l1
    store a0 0
    call f_fac
    store a0 1
    store s0 2
    loadaddr 0 s0
    a0 = s0[0]
    a0 = a0 + -1
    call f_fac
    a1 = s0[4]
    load 2 s0
    a0 = a0 + a1
    goto l2
l1:
    a0 = 1
l2:
    return
end f_fac

f_main [0] [0]
    call f_getint
    call f_fac
    call f_putint
    a0 = 10
    call f_putchar
    a0 = 0
    return
end f_main
```

5 Tigger 模拟器使用方式

Usage ./Tigger [-d] <filename>

-d : enable debug mode

- e.g. ./Tigger -d test.in

出现">"提示符表示进入debug模式，支持如下指令：

- + l
 - Print current line number
- + n
 - Run one step
- + pr <Reg>
 - e.g. pr a0, pr s0
 - Print register value
- + prx <Reg>
 - e.g. prx a0, prx s0
 - Print register value as hexadecimal
- + ps <stacknum>
 - e.g. ps 0, ps 1
 - Print the value of stack memory
- + psx <stacknum>
 - e.g. psx 0, psx 1
 - Print the value of stack memeory as hexadecimal
- + pg <variable>
 - e.g. pg v0, pg v1
 - Print the value of global variable
- + b <number>
 - e.g. b 10
 - Set a breakpoint at a certain line
- + d <number>
 - e.g. d 10
 - Delete the breakpoint at a certain line
- + c
 - Run until meet a breakpoint
- + q
 - Quit Tigger simulator