第4季

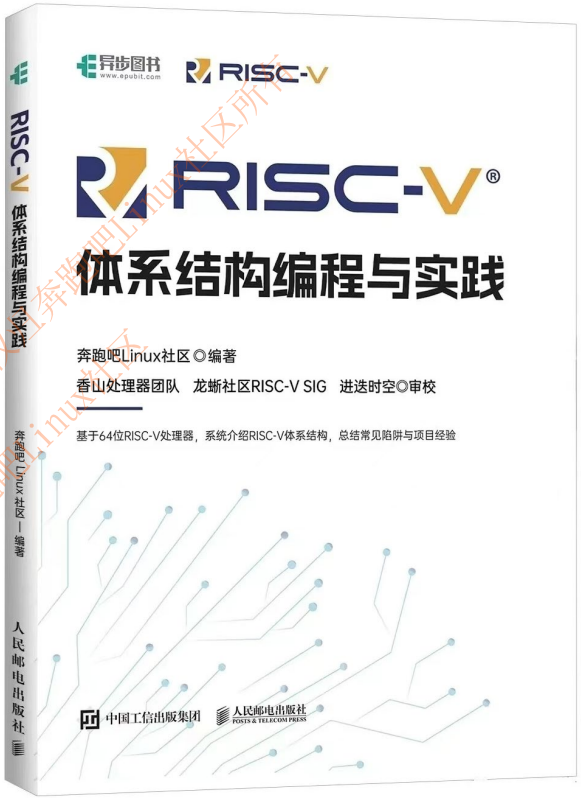
GNU AS汇编器介绍

本节课主要内容

- 本章主要内容
  - 编译过程和ELF文件
  - AS汇编语法
  - 伪指令
  - 汇编宏
  - 汇编实验

技术手册：
1. Using AS, the GNU Assembler, v2.34

扫码订阅RISC-V视频课程



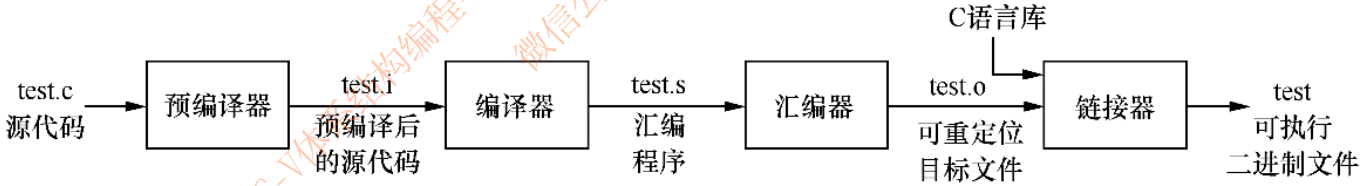
本节课主要讲解书上第5章内容

编译流程介绍

- 预处理（pre-process）
  - ✓ gcc -E test.c -o test.i
- 编译（compile）
  - ✓ gcc -S test.i -o test.s
- 汇编（assemble）
  - ✓ as test.s -o test.o
- 链接（link）
  - ✓ ld -o test test.o –lc

```
<test.c>
#include <stdio.h>
int data = 10;
int main(void)
{
    printf("%d\n", data);
    return 0;
}
```

gcc test.c -o test



ELF文件

- 可执行与可链接格式（Executable and Linkable Format，ELF），文件格式一种
- 常见的段：
  - ✓ 代码段：存放程序源代码编译后生成的机器指令。
  - ✓ 只读数据段：存储只能读取不能写入的数据。
  - ✓ 数据段：存放已初始化的全局变量和已初始化的局部静态变量。
  - ✓ 未初始化的数据段：存放未初始化的全局变量以及未初始化的局部静态变量。
  - ✓ 符号表（.symtab）段：存放函数和全局变量的符号表信息。
  - ✓ 可重定位代码（.rel.text）段：存储代码段的重定位信息。
  - ✓ 可重定位数据（.rel.data）段：存储数据段的重定位信息。
  - ✓ 调试符号表（.debug）段：存储调试使用的符号表信息。

ELF文件头
程序头表
.text段
.rodata段
.data段
.bss段
.symtab段
⋮
段头表
⋮

```
rlk@master:~$ readelf --help
Usage: readelf <option(s)> elf-file(s)
 Display information about the contents of ELF format files
 Options are:
  -a --all               Equivalent to: -h -l -S -s -r -d -V -A -I
  -h --file-header       Display the ELF file header
  -l --program-headers   Display the program headers
     --segments          An alias for --program-headers
  -S --section-headers   Display the sections' header
     --sections          An alias for --section-headers
  -g --section-groups    Display the section groups
  -t --section-details   Display the section details
  -e --headers           Equivalent to: -h -l -S
  -s --syms              Display the symbol table
     --symbols           An alias for --syms
  --dyn-syms             Display the dynamic symbol table
```

```
root:riscv# readelf -h test
ELF Header:
  Magic:   7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF64
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              DYN (Position-Independent Executable file)
  Machine:                           RISC-V
  Version:                           0x1
  Entry point address:               0x560
  Start of program headers:          64 (bytes into file)
  Start of section headers:          6688 (bytes into file)
  Flags:                             0x5, RVC, double-float ABI
  Size of this header:               64 (bytes)
  Size of program headers:           56 (bytes)
  Number of program headers:         9
  Size of section headers:           64 (bytes)
  Number of section headers:         27
  Section header string table index: 26
```

test文件是一个ELF64类型的可执行文件（Executable file）。test程序的入口地址为0x560。段头信息（section header）数量是27个，程序头（program header）信息数量是9个。

可重定位目标文件与可执行目标文件

- 可重定位目标文件（Relocatable Object Files）：在链接阶段与其他可重定位目标文件合并成一个可执行目标文件
  - ✓ 包含代码和数据
  - ✓ 所有段的起始地址都不确定，暂定为0
  - ✓ 可重定位代码（.rel.text）段
  - ✓ 可重定位数据（.rel.data）段

- 可执行目标文件（excuteable Object Files）：可执行的目标文件
  - ✓ 包含代码和数据
  - ✓ 所有段的起始地址都有了确定的地址

例子：一个简单的RISC-V汇编程序

```
1    # 测试程序：往终端中输出my_data1数据与my_data2数据之和
2     .section .data
3     .align 3
4
5     my_data1:
6         .word  100
7
8    my_data2:
9         .word  50
10
11   print_data:
12        .string "data: %d\n"
13
14   .align  3
15   .section .text
16
17   .global main
18   main:
19        addi sp, sp, -16
20        sd ra, 8(sp)
21
22        lw t0, my_data1
23        lw t1, my_data2
24        add a1, t0, t1
25
26        la a0, print_data
27        call printf
28
29        li a0, 0
30
31        ld ra, 8(sp)
32        addi sp, sp, 16
33        ret
```

编译和运行

在笨叔提供的runninglinuxkernel_5.15下编译。

https://github.com/runninglinuxkernel/runninglinuxkernel_5.15

```
# as test.S -o test.o

# ld test.o -o test -Map test.map -lc --dynamic-linker /lib/ld-linux-riscv64-lp64d.so.1

# ./test
data: 150
```

```
root:riscv# readelf -s test

Symbol table '.symtab' contains 37 entries:
   Num:    Value          Size Type    Bind   Vis      Ndx Name
    26: 0000000000002040     0 NOTYPE  GLOBAL DEFAULT   14 __BSS_END__
    27: 0000000000002040     0 NOTYPE  GLOBAL DEFAULT   14 _edata
    28: 0000000000002040     0 NOTYPE  GLOBAL DEFAULT   14 __SDATA_BEGIN__
    29: 0000000000002000     0 NOTYPE  GLOBAL DEFAULT   13 __DATA_BEGIN__
    30: 0000000000002000     0 NOTYPE  GLOBAL DEFAULT   13 my_data1
    31: 0000000000002040     0 NOTYPE  GLOBAL DEFAULT   14 _end
    32: 0000000000000320     0 NOTYPE  GLOBAL DEFAULT   11 main
    33: 0000000000002800     0 NOTYPE  GLOBAL DEFAULT  ABS __global_pointer$
    34: 0000000000002040     0 NOTYPE  GLOBAL DEFAULT   14 __bss_start
    35: 0000000000002004     0 NOTYPE  GLOBAL DEFAULT   13 my_data2
```

my_data1标签的地址为0x2000，my_data2标签的地址为0x2004，而main符号的地址为0x320

汇编语法－注释

- label: 任何以冒号结尾的标识符都被认为是一个标号
- 注释：
  - ✓ “//” 表示注释
  - ✓ “#”：在一行的开始，表示注释整行
  - ✓ /* */
- 指令，伪指令，可以全部是大写或者小写，GNU风格默认是小写
- 寄存器必须小写

汇编语法 – 符号symbol

- 符号：代表它所在的地址, 也可以当作变量或者函数来使用。

- 符号可以由下面几种字符组合而成：
  - ✓ 所有字母（包括大写和小写）；
  - ✓ 数字；
  - ✓ “_”“.”以及“$”这三个字符。

- 符号类型
  - ✓ 全局符号，可以使用.global来声明
  - ✓ 本地符号：在本地汇编代码中引用。通常使用“.L”前缀来定义一个本地符号
  - ✓ 本地便签，主要在局部范围内使用，开头以0-99直接的数字为标号名，通常和b指令结合使用
    - f：指示编译器向前搜索
    - b：指示编译器向后搜索

```
1:
    j 1f
2:
    j 1b
1:
    j 2f
2:
    j 1b
```

```
label_1:
        j label_3
label_2:
        j label_1
label_3:
        j label_4
label_4:
        j label_3
```

伪指令

- 伪指令是对汇编器发出的命令，它在源程序汇编期间由汇编器处理。
- 伪操作可以实现如下功能：
  - ✓ 符号定义；
  - ✓ 数据定义和对齐；
  - ✓ 汇编控制；
  - ✓ 汇编宏；
  - ✓ 段描述。

对齐伪指令

```
print_data:
    .string "data: %d\n"

main:
    addi sp, sp, -16
```

```
print_data:
    .string "data: %d\n“

.align 2
main:
    addi sp, sp, -16
```

- .align 对齐，填充数据来实现对齐。可以填充0或者使用nop指令。
  - 告诉汇编程序, align后面的汇编必须从下一个能被2^n整除的地址开始分配
  - RISC-V系统中，第一个参数表示2^n大小。

7.3 .align [*abs-expr*[, *abs-expr*[, *abs-expr*]]]

Pad the location counter (in the current subsection) to a particular storage boundary. The first expression (which must be absolute) is the alignment required, as described below. If this expression is omitted then a default value of 0 is used, effectively disabling alignment requirements.

例子1:

```
.align 2
```

例子2:

```
.align 5,0,100

.align 5,0,8
```

数据定义伪指令

- .byte: 把8位数当成数据插入到汇编中
- .hword: 把16位数当成数据插入到汇编中
- .long 和 .int：把32位数当成数据插入到汇编中
- .quad: 把64位数当成数据插入到汇编中
- .float: 把浮点数当成数据插入到汇编中
- .ascii “string” -> 把string当作数据插入到汇编中，ascii伪操作定义的字符串需要自行添加结尾字符'\0'。
- .asciz “string” -> 类似ascii，在string后面插入一个结尾字符’\0’。

- .rept：重复定义

```
.rept 3
.long 0
.endr
```

```
.long 0
.long 0
.long 0
```

- .equ: 赋值操作
- .set : 赋值操作

```
.equ  abcd,  0x45   //让abcd 等于 0x45
```

```
.equ my_data1, 100   #为 my_data1 符号赋值 100
.equ my_data2, 50    #为 my_data2 符号赋值 50

.global main
main:
    ...
    li x2, =my_data1
    li x3, =my_data2

    add x1, x2, x3
    ...
```

函数相关的伪操作

- .global：定义一个全局的符号
- .include: 引用头文件
- .if，.else, .endif 控制语句

If语句

- .ifdef symbol　　判断symbol是否定义
- .ifndef symbol　　判断symbol是否 没有定义
- .ifc string1,string2　　字符串string1和string2是否相等
- .ifeq expression　　判断expression的值是否为0
- .ifeqs string1,string2　　等同于.ifc
- .ifge expression　　判断expression的值是否大于等于0
- .ifle expression　　判断expression的值是否小于等于0
- .ifne expression　　判断expression的值是否不为0

与段相关的伪操作

- .section：表示接下来的汇编会链接到哪个段里，例如代码段，数据段等
- 每一个段以段名为开始，以下一个段名或者文件结尾为结束

.section name, “flags”

后面可以添加flags，表示段的属性。

a	section is allocatable
d	section is a GNU_MBIND section
e	section is excluded from executable and shared library.
w	section is writable
x	section is executable
M	section is mergeable
S	section contains zero terminated strings
G	section is a member of a section group
T	section is used for thread-local-storage
?	section is a member of the previously-current section’s group, if any

```
470 /*
471  * end early head section, begin head code that is also used for
472  * hotplug and needs to have the same protections as the text region
473  */
474         .section ".idmap.text","awx"
```

表示接下来的代码是是在”.idmap.text”段里，具有可分配，可写，和可执行的属性

书上图5.4

- .pushsection: 把下面的代码push到指定的section中
- .popsection: 结束push
- 成对使用，仅仅是把 pushsection和popsection的圈出来的代码 加入到指定的section中，其他代码还是在原来的section

```
1    .section .text
2    .global my_memcpy_test
3    my_memcpy_test:
4        ...
5        ret
6
7    .pushsection ".my.text", "awx"
8
9    .global compare_and_return
10   compare_and_return:
11       bltu    a0,a1,.L2
12       li  a5,0
13       j   .L3
14   .L2:
15       li  a5,-1
16   .L3:
17       mv  a0, a5
18       ret
19
20   .popsection
21
22   ...
```

书上例5.8

- .section和.previous两个伪指令是配对一起使用，用来把一段汇编代码链接到特定的段中。
- .section伪指令表示开始一个新的段，.previous伪指令表示恢复到.section定义之前的那个段作为当前段。

```
1    .section ".text.boot"
2
3    .globl _start
4    _start:
5        /* 关闭中断 */
6        csrw sie, zero
7
8        /* 设置栈，栈的大小为4KB */
9        la sp, stacks_start
10       li t0, 4096
11       add sp, sp, t0
12
13       .section .fixup, "ax"
14       .balign 4
15       li a2, -1
16       li a1, 0
17       .previous
18
19       .section .init_boot,"ax"
20       li a0, -1
21       mv a1, a2
22       .previous
23
24       call kernel_main
```

```
$ riscv64-linux-gnu-objdump -d benos.elf

benos.elf:     file format elf64-littleriscv


Disassembly of section .text.boot:

0000000080200000 <_start>:
    80200000:	10401073          	csrw    sie,zero
    80200004:	00001117          	auipc   sp,0x1
    80200008:	ffc10113          	addi    sp,sp,-4 # 80201000 <stacks_start>
    8020000c:	000012b7          	lui t0,0x1
    80200010:	00510133          	add sp,sp,t0
    80200014:	174000ef          	jal ra,80200188 <kernel_main>
Disassembly of section .fixup:

00000000802001a4 <.fixup>:
    802001a4:	fff00613          	li  a2,-1
    802001a8:	00000593          	li  a1,0

Disassembly of section .init_boot:

00000000802001ac <.init_boot>:
    802001ac:	fff00513          	li  a0,-1
    802001b0:	00060593          	mv  a1,a2
```

书上例5.9

实验1：使用汇编伪操作来实现一张表

目的：熟悉常用的汇编伪指令

1. 使用汇编的数据定义伪指令，可以实现 表的定义，例如Linux内核使用.quad和.asciz来定义一个kallsyms的表，地址和函数名的对应关系。

0x800800 -> func_a
0x800860 -> func_b
0x800880 -> func_c

请在汇编里定义一个类似这样表，然后在C语言中 根据函数的地址来 查找表，并且正确打印函数的名称。

```
rlk@master:benos$ make run
qemu-system-riscv64 -nographic -machine virt
Booting at asm
Welcome RISC-V!
lab3-5: compare_and_return ok
lab3-5: compare_and_return ok
lab3-7: branch test ok
func_c
func_b
func_a
```

汇编宏 （难点）

- .macro和.endm组成一个 宏
- .macro后面跟着的是 宏的名称，在后面是宏的参数
- 在宏里使用参数，需要添加前缀”\”

```
.macro add_1 p1 p2
add x0, \p1, \p2
.endm
```

定义了一个名为add_1的宏，有两个参数p1，和p2。
在宏里使用参数需要前缀，“\p1” 表示第一个参数，
“\p2”表示第二个参数

- 宏参数定义的时候可以 设置一个初始化值

```
.macro reserve_str p1=0 p2
```

第一个参数p1有一个初始化的值，0。这时候你可以使用
reserve_str a,b
reserve_str ,b

来调用这个宏

例子

```
1    .macro add_data p1=0 p2
2    mv a5, \p1
3    mv a6, \p2
4    add a1, a5, a6
5    .endm
6
7    .globl main
8    main:
9        mv a2, #3
10       mv a3, #3
11
12       add_data a2, a3
13       add_data , a3
```

```
1    .macro add_data_1 p1:req p2
2    mv a5, \p1
3    mv a6, \p2
4    add a1, a5, a6
5    .endm
6
7    .globl main
8    main:
9        add_data_1 , a3
```

这个例子会编译出错

```
root:riscv# as test.S -o test.o
test.S: Assembler messages:
test.S:27: Error: Missing value for required parameter `p1' of macro
`add_data_1'
test.S:27: Error: illegal operands `mv a5,'
```

- 宏参数后面加入“:req”表示在宏调用过程中必须传递一个值，否则会编译报错

宏 （难点）

```
.macro label l
\l:
.endm
```

```
.macro opcode base length
    \base.\length
.endm
```

这里并不能生成一个以参数l为名称的符号

上述宏会出错，或者有问题

这里并不能把两个参数以及“.”连接在一起

解决办法

- 使用空格 或者使用 altmacro+&

```
.macro label l
\l :
.endm
```

```
.altmacro
.macro label l
l&:
.endm
```

- 使用 “\()” 表示 **用来指示 字符串什么时候结束**

```
.macro opcode base length
        \base\() \length
.endm
```

如果没有”\()”， \base这个参数不知道 哪个字符算 参数的结束字符

宏中的连接符号的妙用

```
.macro  my_entry,  rv,  label
    j    rv\()\rv\()_\label
.endm
```

调用my_entry宏。

```
my_entry  1, irq
```

```
j rv1_irq
```

文件相关的伪指令

- .incbin伪指令可以把文件的二进制数据嵌入到当前位置。

```
    .section .payload, "ax", %progbits
    .globl payload_bin
payload_bin:
    .incbin "benos.bin"
```

把benos.bin的二进制数据嵌入到.payload段中

- .include伪指令可以在汇编代码中插入另外一个文件的汇编代码。

```
.include "sbi/sbi_payload.S"
```

RISC-V汇编编译选项

表 5.2 RISC-V 命令行选项

选项	说明
-fpic/-fPIC	生成位置无关的代码
-fno-pic	不生成位置无关的代码（as编译器默认配置）
-mabi=ABI	指定源代码使用哪个ABI。可识别的参数是：ilp32和lp64，它们分别决定生成ELF32或者ELF64格式的对象文件。
-march=ISA	用来指定目标体系结构，例如-march=rv32ima。如果没有指定这个选择，那么as编译器会读取默认的配置“–with-arch=ISA”。
-misa-spec=ISAspec	选择目标指令集的版本。
-mlittle-endian	生成小端的机器码。
-mbig-endian	生成大端的机器码。

汇编器实验2：汇编宏的使用

目的：熟悉汇编宏的使用

1. 在汇编文件里有 如下两个函数：

```
.global op_1
op_1:
        add a0, a0, a1
        ret

.global op_2
op_2:
        sub a0, a0, a1
        ret
```

写一个汇编宏来调用上述函数。

```
. .macro op_func op, a, b
        //这里调用op_1或者op_2函数，op等于1或者2
.endm
```

文字不如声音，声音不如视频

扫描订阅RISC-V视频课程

第4季 奔跑吧Linux社区 视频课程

RISC-V体系结构编程与实践

主讲：笨叔

课程名称	进度	时长（分钟）
第1课：课程介绍（免费）	完成	20
第2课：RISC-V体系结构介绍（免费）	完成	47
第3课：RISC-V处理器微架构（免费）	完成	48
第4课：搭建RISC-V开发环境（免费）	完成	30
第5课：RISC-V指令集（免费）	完成	128
第6课：RISC-V函数调用规范	完成	40
第7课：RISC-V GNU AS汇编器	完成	42
第8课：RISC-V GNU 链接脚本	完成	90
第9课：RISC-V GNU 内嵌汇编	完成	52
第10课：RISC-V异常处理	完成	80
第11课：RISC-V中断处理	完成	52
第12课：RISC-V内存管理	完成	116
第13课：内存管理实验讲解	完成	36
第14课：cache基础知识	完成	78
第15课：缓存一致性	完成	96
第16课：RISC-V TLB管理	完成	54
第17课：RISC-V原子操作	未录制	
第18课：RISC-V内存屏障	未录制	
第19课：BenOS操作系统相关知识	未录制	
第20课：RVV可伸缩矢量计算	未录制	
第21课：RISC-V压缩指令	未录制	
第22课：RISC-V虚拟化	未录制	
		总计17小时

更多精彩内容马上献上....
微信公众号：奔跑吧Linux社区

视频课程持续更新中...