微机原理与接口技术

姓名：陈致蓬

单位：中南大学自动化学院

电话： 15200328617

Email ： ZP.Chen@csu.edu.cn

Homepage:

https://www.scholarmate.com/psnweb/homepage

QQ ： 315566683

第9章 汇编（二）

9.1 寄存器介绍

8.2 寻址模式

8.3 汇编程序编写

9.1 寄存器介绍
9.1.1 X64 寄存器命名

- 前缀 R

表示 64 位寄存器。例如 RAX 。

- 前缀 E

表示 32 位寄存器。例如 EAX

- 后缀 H

表示寄存器的低 8 位

- 后缀 L

表示寄存器的 9~16 位

9.1 寄存器介绍
9.1.2 专用寄存器

- cs ds ss es fs gs 等段寄存器在 64 位平台几乎没什么用；
- ip 指令指针寄存器，指向下一条指令的地址；
- flags 标志位寄存器：

ZF 零标志；

CF 进位 / 借位标志；

SF 符号标志，正为 0 负为 1 ；

OF 溢出标志，加减乘运算时结果超出能表示的范围时为 1 ；

DF 方向标志，用于串指令，向高地址方向为 0 ，向低地址方向为 1 ；

9.1 寄存器介绍
9.1.3 其他寄存器

- mmx 寄存器

mm0~mm7 ，用于 mmx 指令集。

- xmm 寄存器 xmm0~xmm7 ， 128 位寄存器；

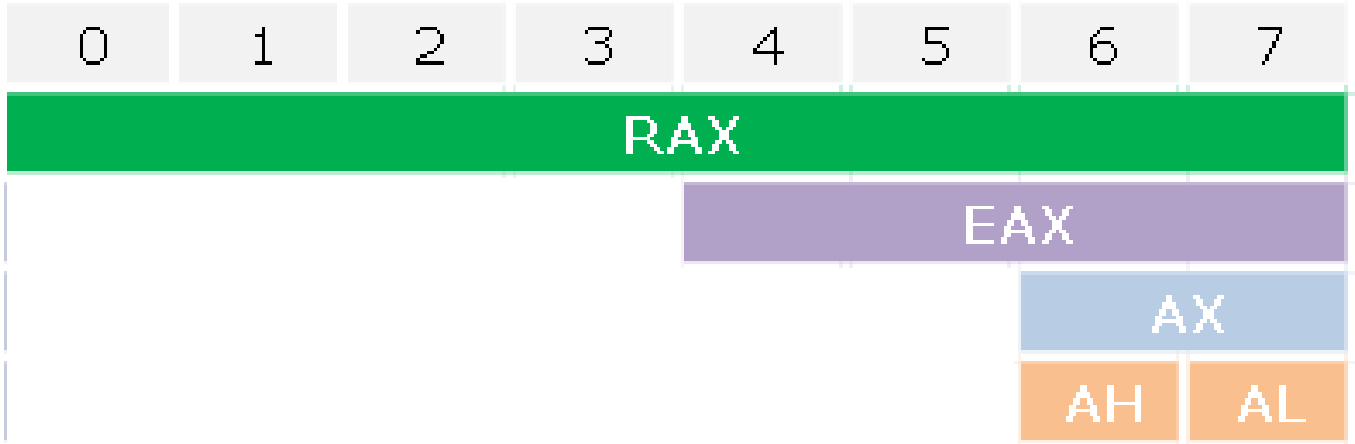
ymm0`ymm15` ， 256 位寄存器， `ymm0`ymm7 的低 128 位是 xmm 的映射； zmm0`zmm31` ， 512 位寄存器， `zmm0`zmm15 的低 256 位是 ymm 的映射。

9.1 寄存器介绍
9.1.4 通用寄存器

- rax 累加寄存器，通常用于传递函数返回值；
- rcx 计数寄存器，通常用于循环计数和函数传参；
- rbx 基址寄存器，现在的 x86 架构中可以随意使用；
- rdx 数据寄存器，可以随意使用，也用于函数传参；
- rsi 源变址寄存器，用于串指令和函数传参；
- rdi 目的变址寄存器，用于串指令和函数传参；
- rbp 栈底指针；
- rsp 栈顶指针；
- r8 r9 用于函数传参；
- r10~r15 随意使用；

9.1.5 RAX 寄存器

accumulator register ，累加寄存器，通常用于存储函数的返回值。其实也可以用于存储其他值，只是通过 RAX 存储函数返回值属于惯例。



这个寄存器分为 8 个字节。 RAX 是 64 位寄存器的称呼，但寄存器是可拆分的。例如操作 EAX ，就是在对 RAX 的低 32 位进行操作。以此类推， AX 表示 RAX 的低 16 位， AH 表示 RAX 低 16 位中的高 8 位， AL 表示 RAX 低 16 位中的低 8 位。除了 RIP 之外，其余的寄存器都可以做类似的拆分

9.1.6 RBX、 RCX、 RDX 寄存器

- **RBX**

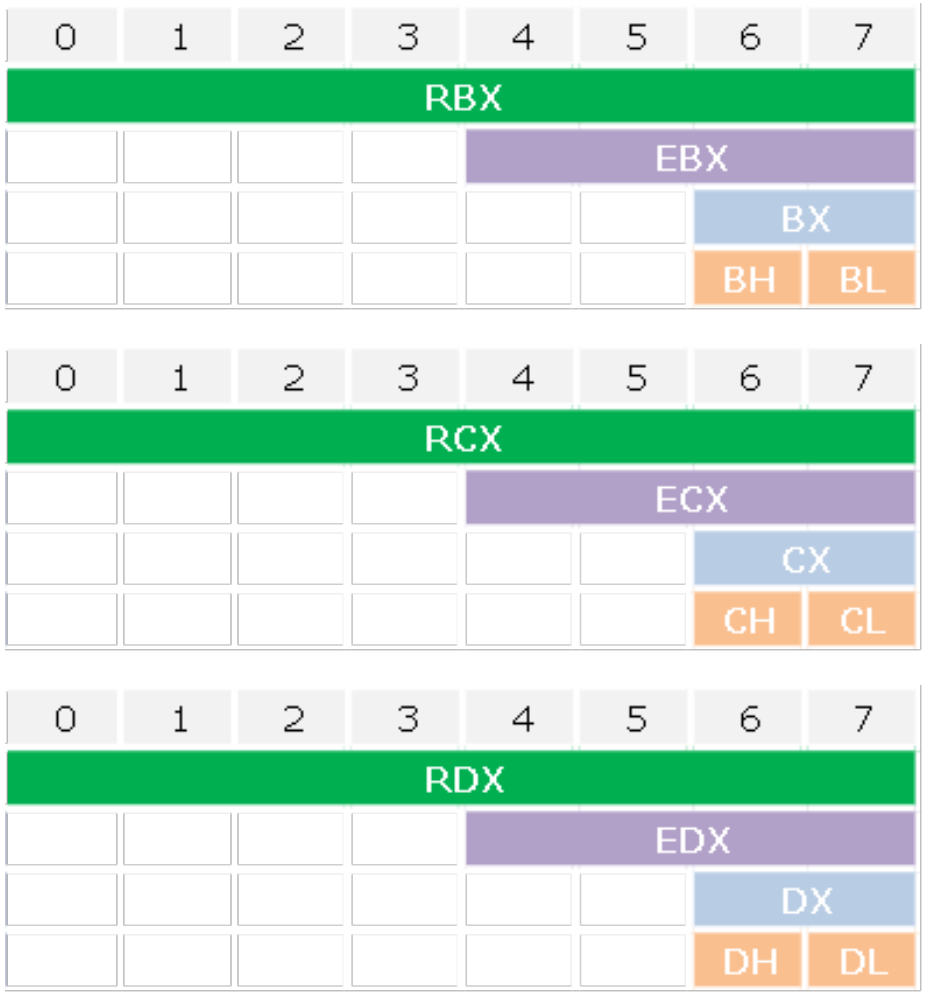
base register ，基址寄存器，一般用于访问内存的基址。

- **RCX**

counter register ，计数寄存器。一般用于循环计数。

- **RDX**

data register ，数据寄存器。



9.1.7 RSI、 RDI、 RSP 寄存器

- **RSI**

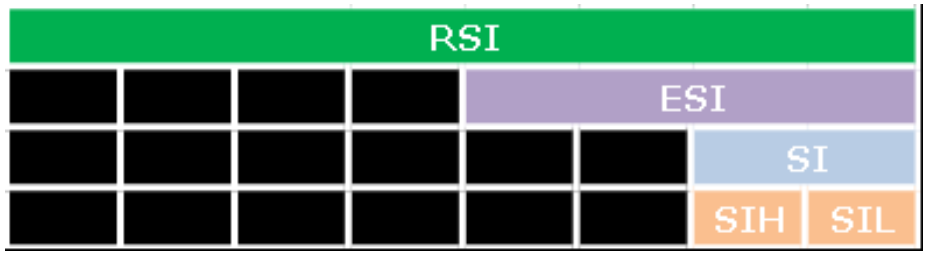
source index ，源变址寄存器，字符串运算时常应用于源指针。

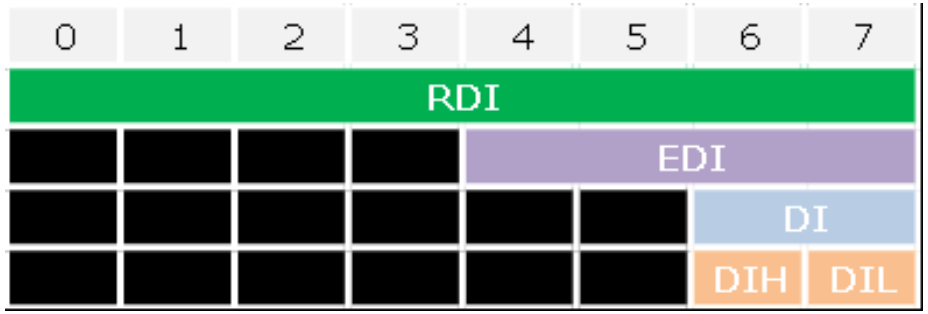
- **RDI**

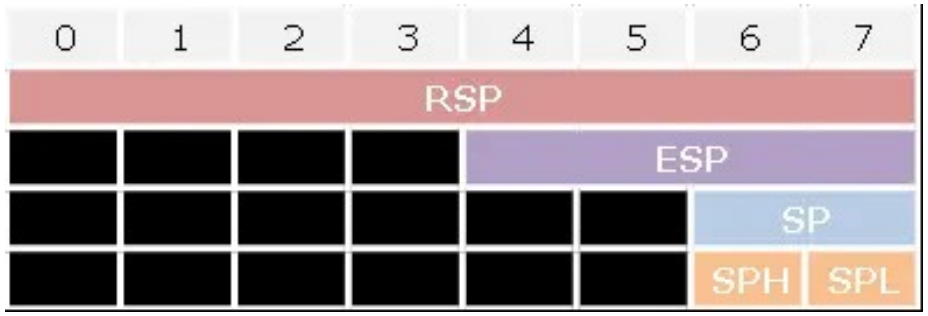
destination index ，目标变址寄存器，字符串运算时常用于目标指针。

- **RSP**

stack pointer ，栈指针寄存器，正常情况下存放栈顶地址，如果用于其他事务，使用完成之后需要恢复原先的数据。



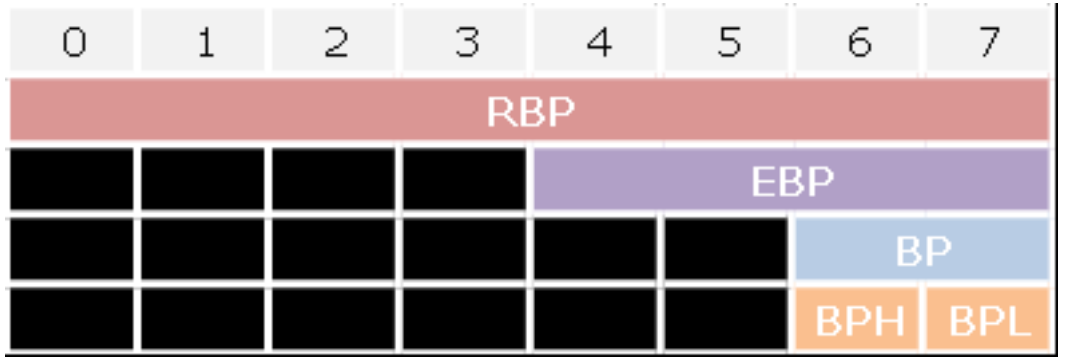




9.1.8 RBP、 RIP 寄存器

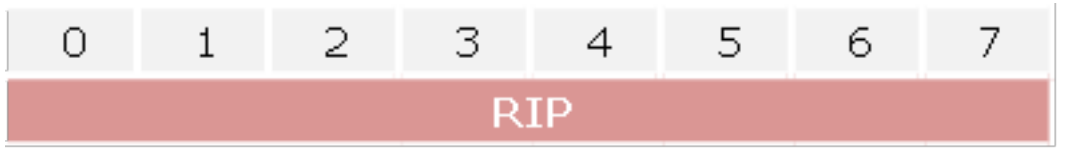
- **RBP**

base pointer ，基址寄存器，正常情况用于访问栈底地址。与 RBP 类似，如果用于其他事务，使用完成之后需要恢复原先的数据。



- **RIP**

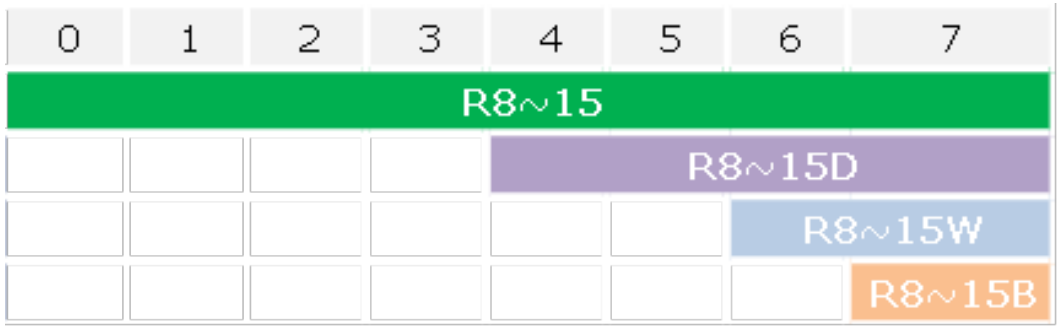
instruction pointer ，指令指针。只读，且不可拆分。指向下一条需要执行的指令地址。



9.1.9 普通寄存器

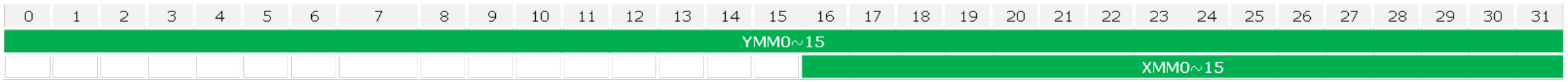
- **R8 ~ R15**

R8, R9, R10, … , R15 属于普通寄存器，一般是可以任意使用，不指定特定用途。



- **YMM0~15**

YMM0~15 专门用于存储浮点数（包括 float 和 double ）。单个寄存器可以存放多个浮点数。



9.2 寻址模式

9.2.1 介绍

什么是寻址模式？

数据在内存器和寄存器之间进行移动时，取得数据地址的不同表达方式。最常用的寻址的汇编指令是 mov 。 x86-64 使用的是复杂指令集(cisc) ，因此 mov 有许多不同的变体，可以在不同单元之间移动不同类型的据。 mov 与大多数指令一样，具有单字母后缀，用于确定要移动的数据量。下表用于描述各种大小的数据值：

前缀	全称	Size
B	BYTE	1 byte (8 bits)
W	WORD	2 bytes (16 bits)
L	LONG	4 bytes (32 bits)
Q	QUADWORD	8 bytes (64 bits)

9.2.2 MOV 指令

前缀	全称	Size
B	BYTE	1 byte (8 bits)
W	WORD	2 bytes (16 bits)
L	LONG	4 bytes (32 bits)
Q	QUADWORD	8 bytes (64 bits)

MOVB 移动一个字节， MOVW 移动一个字， MOVL 移动一个长整形， MOVQ 移动一个四字。一般来说， MOV 指令移动数据的大小必须与后缀匹配。虽然可以忽略后缀，汇编器将尝试根据参数选择合适的 MOV 指令。但是，不推荐这样做，因为它可能会造成预料之外的结果。

9.2.3 寻址方式

对于 AT&T 语法使用 MOV 寻址时需要两个参数，第一个参数是源地址，第二个参数是目标地址。原地址的表达方式不一样那么寻址的方式也就不一样。以下是使用各种寻址模式将 64 位值加载到 % rax 的示例：

寻址模式	示例
全局符号寻址(Global Symbol)	MOVQ x, %rax
直接寻址(Immediate)	MOVQ $56, %rax
寄存器寻址(Register)	MOVQ %rbx, %rax
间接寻址(Indirect)	MOVQ (%rsp), %rax
相对基址寻址(Base-Relative)	MOVQ -8(%rbp), %rax
相对基址偏移缩放寻址(Offset-Scaled-Base-Relative)	MOVQ -16(%rbx,%rcx,8), %rax

9.2 寻址模式
9.2.4 寻址方式

- 直接寻址

使用绝对在指令格式的地址字段中直接指出操作数在内存地址，称这种寻址方式为直接寻址方式。

- 全局符号寻址

访问全局变量，使用一个简单变量的名称，比如 x 代替绝对地址，将其作为源地址，我们称之为全局符号寻址。

寻址模式	示例
全局符号寻址(Global Symbol)	MOVQ x, %rax
直接寻址(Immediate)	MOVQ $56, %rax
寄存器寻址(Register)	MOVQ %rbx, %rax
间接寻址(Indirect)	MOVQ (%rsp), %rax
相对基址寻址(Base-Relative)	MOVQ -8(%rbp), %rax
相对基址偏移缩放寻址(Offset-Scaled-Base-Relative)	MOVQ -16(%rbx,%rcx,8), %rax

9.2.4 寻址方式

- 寄存器寻址

直接使用寄存器的名字如 %RBX 来直接访问寄存器的值，我们称之为寄存器寻址。

- 间接寻址

如果寄存器中存放的是一个地址，访问这个地址中的数据时需要在寄存器外面加上括号如 (%rbx) ，我们称之为间接寻址。

寻址模式	示例
全局符号寻址(Global Symbol)	MOVQ x, %rax
直接寻址(Immediate)	MOVQ $56, %rax
寄存器寻址(Register)	MOVQ %rbx, %rax
间接寻址(Indirect)	MOVQ (%rsp), %rax
相对基址寻址(Base-Relative)	MOVQ -8(%rbp), %rax
相对基址偏移缩放寻址(Offset-Scaled-Base-Relative)	MOVQ -16(%rbx,%rcx,8), %rax

9.2.4 寻址方式

- 相对基址寻址

如果寄存器中存放的是一个数组的地址，我们需要访问数组中的元素时可能需要操作这个地址进行偏移，如 8(%rcx) 是指%rcx 中存放的的地址加 8 字节存储单元的值，我们称之为相对基址寻址（此模式对于操作堆栈，局部变量和函数参数非常重要）。间接寻址

寻址模式	示例
全局符号寻址(Global Symbol)	MOVQ x, %rax
直接寻址(Immediate)	MOVQ $56, %rax
寄存器寻址(Register)	MOVQ %rbx, %rax
间接寻址(Indirect)	MOVQ (%rsp), %rax
相对基址寻址(Base-Relative)	MOVQ -8(%rbp), %rax
相对基址偏移缩放寻址(Offset-Scaled-Base-Relative)	MOVQ -16(%rbx,%rcx,8), %rax

9.2.4 寻址方式

- 相对基址偏移缩放寻址

在相对基址寻址上有各种复杂的变化，例如 -16 （% rbx ，% rcx ，8）是指地址 -16 + % rbx + % rcx * 8 处的值。此模式对于访问排列在数组中的特殊大小的元素非常有用。

寻址模式	示例
全局符号寻址(Global Symbol)	MOVQ x, %rax
直接寻址(Immediate)	MOVQ $56, %rax
寄存器寻址(Register)	MOVQ %rbx, %rax
间接寻址(Indirect)	MOVQ (%rsp), %rax
相对基址寻址(Base-Relative)	MOVQ -8(%rbp), %rax
相对基址偏移缩放寻址(Offset-Scaled-Base-Relative)	MOVQ -16(%rbx,%rcx,8), %rax

9.3 汇编程序编写
9.3.1 汇编介绍

现在 x86_64 平台上有很多种不同的汇编语言，这些汇编语言一般只能由特定的汇编器汇编。

现在的 x86_64 汇编器主要有 nasm 、 gas 、微软 MASM 等。

汇编代码的编码格式也有些不同，分别是以 nasm 为代表的 intel 汇编格式和以 gas 为代表的 ATT 汇编格式。

9.3.2 如何获得汇编器

- **windows 平台**
  - nasm 下载 **nasm** 官网安装程序并安装即可。
  - gas 它是 **GCC** 的一部分，安装 **mingw-w64** ，并适当配置环境变量即可，运行它的命令是 **as** 。

- **linux 平台**
  - nasm 使用软件包管理器即可安装。
  - gas 是 **GCC** 的一部分，只要有 **GCC** 环境就一定有 **as** 命令。

9.3.3 程序格式

MASM 中的语句分为指令性语句和伪指令语句

指令性语句与机器指令相对应，汇编程序将它们翻译成目标代码。语句格式为：

标号： 指令助记符 操作数，操作数 ； 注释

伪指令语句没有对应的机器指令，可完成数据定义，存储区分配，段定义，段分配，指示程序结束等功能。

名字 伪指令指示符 操作数，操作数 ； 注释

汇编语言中常数，变量和标号是三种基本数据项

9.3.4 算数运算符

符号	名称	运算结果
ADD	加法	和
SUB	减法	差
MUL	乘法	积
DIV	除法	商
MOD	模除	余数
SHL	左移	左移后的二进制数
SHR	右移	右移后的二进制数

9.3.4 算数运算符

- 例：
  - add eax,10
  - sub eax,10
  - mul eax , ecx , 4
  - div eax,ecx, 4

9.3.5 逻辑运算符

符号	名称	运算结果
AND	与运算	逻辑与结果
OR	或运算	逻辑或结果
XOR	异或运算	逻辑异或结果
NOT	非运算	逻辑非结果

9.3.5 逻辑运算符

- 例：
  - MOV EAX, NOT OFFH
  - MOV EBX, 8 AND 73
  - MOV EAX, 8 OR 73
  - MOV ECX, 8 XOR 73

9.3.6 关系运算符

符号	名称	运算结果为真输出全‘1’，为假输出全‘0’
EQ	相等	
NE	不等	
LT	小于	
LE	小于等于	
GT	大于	
GE	大于等于	

9.3.6 段定义语句

段定义语句 **SEGMENT** **ENDS**	定义段 段名 **segment** 定义类型 组合类型 ‘分类名’ 段名 **ends** ；指示段或者结构结束 功能：将一个逻辑段定义成一个整体
ASSUME	规定段所属的段寄存器 **assume cs:** 段名 , **ds:** 段名 , **ss:** 段名 ,**ES:** 段名 功能：定义 4 个逻辑段，指明段和段寄存器的关系

9.3.6 段定义语句

```
DSEG1 SEGMENT
    VARW DW 12
DSEG1 ENDS
DSEG2 SEGMENT
    XXX DW 0
DSEG2 ENDS
CSEG SEGMENT
    ASSUME CS:CSEG , DS: DSG1 , ES : DSG2
    MOV AX , DSEG1
    MOV DS , AX
    MOV AX , DSEG2
    MOV ES , AX
    ……
    ASSUME DS: DSG2 , ES :NOTHING    ;NOTHING 表示该 ES 寄存器不与任何段有对应关系
    MOV AX , DSEG2
    MOV DS , AX
    ……
DSEG ENDS
```

9.3.7 Hello World

```
1   global _start
2
3   section .text
4
5   _start:
6       endbr64
7       push rbp
8       mov rbp, rsp
9
10      ;系统调用sys_write(stdout, hellomsg, strlen)
11      mov rdi, 1              ;第一个参数, 文件描述符, stdout是1
12      mov rsi, hellomsg       ;第二个参数, 字符串地址
13      mov rdx, msglen         ;第三个参数, 字符串长度
14      mov rax, 1              ;系统调用号
15      syscall
16
17      leave
18
19      ;系统调用sys_exit(0)
20      xor rdi, rdi            ;参数, 返回值0
21      mov rax, 60             ;系统调用号
22      syscall
23
24      nop
25
26  section .data
27
28  hellomsg: db "Hello World!", 0xa
29  msglen: equ $-hellomsg
```

9.3.7 Hello World （1）

```
global _start
section .text
_start:
    endbr64
    push rbp
    mov rbp, rsp
```

_start:

汇编语言以 _start 为入口点。

endbr64

在 x86_64 平台中， endbr64 指令须在远跳转后立即出现。

9.3.7 Hello World （2）

```
; 系统调用 sys_write(stdout, hellomsg, strlen)
mov rdi, 1          ; 第一个参数，文件描述符， stdout 是 1
mov rsi, hellomsg   ; 第二个参数，字符串地址
mov rdx, msglen     ; 第三个参数，字符串长度
mov rax, 1          ; 系统调用号
syscall
```

此段程序通过调用 linux 系统调用
sys_write 向 stdout 文件写入字符串。

9.3.7 Hello World （3）

```
; 系统调用 sys_exit(0)
    xor rdi, rdi        ; 参数，返回值 0
    mov rax, 60         ; 系统调用号
    syscall
```

此段程序通过调用 linux 系统调用 sys_exit
退出进程。

9.3.7 Hello World （4）

```
hellomsg: db "Hello World!", 0xa
msglen: equ $-hellomsg
```

此处定义了一个字符串 "Hello World!\n" ，nasm 不支持转义字符，因此用 \n 的 16 进制码 0xa 代替。

9.3.7 Hello World （5）

```
section .text
section .data
```

这两个语句就是定义程序中的 .text 段和 .data 段的，一般程序分为 .text （程序代码） .data （全局变量） .rodata （全局常量） .bss （符号区）等段

谢谢大家！