# 第18章 x86-64位汇编程序设计

# 18.1 x86-64位汇编程序设计基础



- x86-64：Intel 64
  x64（64-bit eXtented），
  是x86架构的64位拓展。
- x86-64的指令集与x86-32指令集兼容。

- 在此之前，Intel公司推出了IA-64架构
- IA-64架构采用的是全新的指令集
- 与x86-32不兼容

Intel CPU 广泛使用是x86-64指令集

## 通用寄存器

- 16个64位的通用寄存器

  rax、rbx、rcx、rdx、rbp、rsi、rdi、rsp、r8、r9、r10、r11、r12、r13、r14、r15

- 这些寄存器的低双字、最低字、最低字节都可以被独立的访问，各自都有自己的名字

- 低双字寄存器

  eax、ebx、ecx、edx、ebp、esi、edi、esp、r8d、r9d、r10d -- r15d

# 18.1 x86-64位汇编程序设计基础



## 通用寄存器

- 低字寄存器

  ax、bx、cx、dx、bp、si、di、sp、

  r8w、r9w、r10w -- r15w

- 最低字节寄存器

  al、bl、cl、dl、bpl、sil、dil、spl、

  r8b、r9b、r10b -- r15b


## 标志寄存器 rflags

- 64位的寄存器，是标志寄存器eflags的扩展
- 低32位与eflags相同，目前其高32位并未使用

## 指令指针rip

- rip的作用与eip是相同的，用来保存当前将要执行的指令的偏移地址
- 与eip一样，不允许直接使用rip的名字
- rip的值由CPU自动维护，不论是顺序执行的指令，还是转移等指令，都会自动的更新rip。
- 16位的段寄存器cs、ds、es、ss、fs、gs仍保持不变。

## 浮点及多媒体寄存器

- 对于x87 FPU而言，仍然使用 32环境下的浮点寄存器st(0)-st(7)。
- 对于MMX，使用原来的8个64位寄存器mm0-mm7。
- 对于SSE，使用128位的寄存器xmm0-xmm7。
- 在SSE2中，新增了8个寄存器 xmm8-xmm15。
- 在AVX中，除了原来的8个256位寄存器ymm0-ymm7外，增加了8个寄存器 ymm8-ymm15。

## 寻址方式

- 立即寻址、寄存器寻址、直接寻址、寄存器间接寻址、变址寻址、基址加变址寻址。
- 内存的地址是64位的，对应内存寻址的四种方式中，计算出的地址是64位的。
- 内存寻址中，不能使用32位的寄存器，要使用64位的寄存器。
- 比例因子仍为 1、2、4、8
- 偏移量为 8位、16位、32位的有符号常量值，而不能使用64位的偏移量

## 指令系统

- x86-64指令基本上向下兼容x86-32的指令集。
- 对大多数的x86-32指令进行了升级。
- 指令默认使用的寄存器升级到64位的寄存器。
- 增加了一些新的指令
- 删除了几条指令

## 指令系统

- x86-64指令基本上向下兼容x86-32的指令集。

```
mov     al, 8          ; 一般数据传送指令
xchg    eax, ebx       ; 数据交换指令
add     ah, 10h        ; 加法指令
imul    bx, 2          ; 有符号乘法指令
shl     ax, 2          ; 逻辑左移指令
jmp     l1             ; 转移指令
```

## 指令系统

- 对大多数的x86-32指令进行了升级。

```
mov     rax, 1234567887654321h
add     rax, rbx
sub     rax, 1234h
lea     rax,  x
cmp     rcx, [rax]
and     [r10+5], dl
shl     rax, 4
movsx   rax, x
movzx   rbx, y
push    r10
pop     r11
```

## 指令系统

- 指令默认使用的寄存器升级到64位的寄存器

push和pop ： rsp 指向栈顶；

loop ： rcx 控制循环次数；

rep/repe/repne前缀：rcx来控制循环次数；

串操作指令：rsi、rdi指向操作数的地址。

# 18.2 64位的程序设计



- 从机器语言的角度来看，x86-32与x86-64 CPU程序的差别不大；
- 使用 VS2019开发汇编语言程序时，32位和64位编译链接器不同，分别是ml.exe 和ml64.exe。

# 18.2 64位的程序设计



ml64.exe 对于很多高级用法不支持

- 不支持处理器选择伪指令
  无 “.686P”、“.xmm”的用法；
- 不支持存储模型说明伪指令
  无“.model”的用法；
- 不支持 invoke 伪指令，函数调用的参数传递由编程者自己掌控；
- 不支持 条件流控制伪指令
- 不持 “end 表达式”的用法，要在项目属性中设置程序的入口点；

# 18.2 64位的程序设计



ml64.exe 对于很多高级用法不支持

- 不支持在C语言程序中内嵌汇编；
- 不支持 “.stack”定义堆栈段。

32位与64位平台下，在C语言函数、Windows API函数调用方面也出现了较大的变化。

# 18.2 64位的程序设计



```
.data
MessageBoxA  proto
  lpContent db  'Hello x86-64',0
  lpTitle   db  'My first x86-64 Application',0
.code
start proc
    sub  rsp, 28h
    xor  r9d, r9d
    lea  r8,  lpTitle
    lea  rdx, lpContent
    xor  rcx,rcx
    call MessageBoxA
    add  rsp,28h
    ret
start endp
end
```

弹出一个消息框

# 18.2 64位的程序设计



程序自我修改，在程序的运行中，修改了机器码，使得程序运行结果发生了变化。

```
VirtualProtect 变更认可页面区域上的保护。
BOOL VirtualProtect(
LPVOID lpAddress, // 变更属性的内存起始地址
DWORD dwSize, //要改变属性的内存区域大小
DWORD flNewProtect, //内存新的属性类型，设置为
//PAGE_EXECUTE_READWRITE（0x40）时该内存页为可读可写可
//执行。
PDWORD lpflOldProtect //内存原始属性类型保存地址。
);
```

# 18.2 64位的程序设计



程序自我修改，在程序的运行中，修改了机器码，使得程序运行结果发生了变化。

```
 extern  MessageBoxA : proc
 extern  ExitProcess : proc
 extern  VirtualProtect : proc
.data
 szMsg1    db 'before Modify : Hello ',0
 szMsg2    db 'After  Modify : Interesting ',0
 szTitle   db 'Modify Program Self',0
 oldprotect    dd ?
```

# 18.2 64位的程序设计



```
.code
 mainp proc
 sub   rsp, 28H
 lea   r9,   oldprotect
       ; 参数 pfloldProtect 指向前一个内存保护值
 mov   r8d, 40H
     ; 参数 flNewProtect   要应用的内存保护的类型
 mov   rdx, 1
     ; 参数 dwsize,要更改的内存页面区域的大小
 lea   rcx, ModifyHere
     ; lpAddress，要更改保护特性的虚拟内存的基址
 call VirtualProtect          允许程序修改
 add   rsp, 28H
```

# 18.2 64位的程序设计



```
  lea   rax, ModifyHere
  inc   byte ptr [rax]          修改程序的核心语句
      ; jz,jnz的机器码分别为 74H，75H
      ; 可比较有此语句和无此语句程序运行结果的差异

  lea   rdx, szMsg1
  xor   eax, eax
ModifyHere:
  jz    next
  lea   rdx, szMsg2
next:
```

# 18.2 64位的程序设计



```
next:
   sub   rsp,  28H
   mov   r9d,  0
   lea   r8,   szTitle
   mov   rcx,  0
   call MessageBoxA
   add   rsp,  28h
   mov   rcx,  0
   call ExitProcess
mainp endp
end
```

显示对话框

# 第18章 x86-64位汇编程序设计



- x86-64位 CPU中的寄存器、寻址方式、指令系统
- 指令系统、编码规则与 32位 CPU 兼容
- VS2019提供的64位、32位编译器有差异

64位的编译器对很多伪指令尚不支持，使得用汇编语言写源程序比32位程序的开发要麻烦一些。