机器级代码

机器语言由二进制代码构成，汇编语言由助记符构成，高级语言翻译成汇编后再翻译为机器语言。

指令的作用：

- 改变程序执行流
- 处理数据
- 指令格式：操作码+地址码
    - 操作码（负责如何处理）：
        - 算术运算：加减乘除、取负数、自增自减
        - 逻辑运算：与或非、异或、左移、右移
        - 其他
    - 地址码（负责寻找数据）：
        - 寄存器
            - 在指令中给出“寄存器名”
            - 通用寄存器：eax, ebx, ecx, edx
                - X=未知，E=Extended=32bit，
                - 去除E则是使用低16bit（变址和堆栈寄存器不行）
            - 变址寄存器：esi, edi
                - I=Inex，S=Sourse，D=Destination
            - 堆栈寄存器：ebp、esp
                - BP=BasePointer，SP=StackPointer
        - 主存
            - 在指令中给出度写长度、主存地址
            - dword ptr——双字，32bit
            - word ptr——单字，16bit
            - byte ptr——字节，8bit
        - 指令
            - 直接在指令中给出要操作的数，“立即寻址”
            - 直接给出常量

以mov指令为例：

`mod 目的操作数d，源操作数s` # 将s复制到d的位置

```
mov eax, ebx     # 将寄存器ebx的数值复制到寄存器eax
mov eax, 5       # 将立即数5复制到寄存器eax
mov eax, dword ptr[af996h]  # 将内存地址af996h所指的32bit值复制到寄存器eax
mov byte ptr[af996h], 5     # 将立即数5复制到内存地址af996h所指的一字节中
move eax, dword ptr[ebx]    # 将ebx所指主存地址的32bit复制到eax寄存器
mov dword ptr[ebx], eax     # 将eax的内容复制到ebx所指主存地址的32bit
mov eax, bytr ptr[ebx]      # 将ebx所指的主存地址的8bit复制到eax
mov eax, [ebx]              # 若未指明读写长度，默认32bit
mov [af996h], eax           # 将eax的内容复制到ad996h所指的地址（未指明长度默认
32bit)
mov eax, dword ptr[ebx+8]   # 将ebx+8所指的主存地址的32bit复制到eax寄存器中
mov eax, dword ptr[af996-12h]   # 将af996-12所指的主存地址的32bit复制到eax寄存器中
```

考试要求

- 只关注x86汇编语言（其他架构会详细注释）
- 结合C语言看懂汇编语言的关键语句
  - 常见指令
  - 选择结构
  - 循环结构
  - 函数调用
- 结合汇编语言分析机器语言指令的格式、寻址方式
- （不会考C翻译为汇编或机器语言）

常见指令

算术运算指令

功能	汇编指令	注释
加	add d, s	add，`d=d+s`
减	sub d, s	substract，`d=d-s`
乘	mul d, s imul d, s	multiply，integer。mul无符号数`d=d*s`；imul有符号数乘法
除	div s idiv s	divide，integer。div无符号除法`edx:eax/s，商存入eax，余数存入edx`；idiv有符号除法
取负数	neg d	negative，`d=-d`
自增++	inc d	increase，`d++`
自减--	dec c	decrease，`d--`

逻辑运算指令

功能	汇编指令	注释
与	and d, s	and，将d、s逐位相与，结果放回d
或	or d, s	or，将d、s逐位相或，结果放回d
非	not d	not，将d逐位取反，结果放回d
异或	xor d, s	exclusive or，将d、s逐位异或，结果放回d
左移	shl d, s	shift left，将d逻辑左移s位，结果放回d
右移	shr d, s	shift right，将d逻辑左移s位，结果放回d

其他指令

实现分支结构、循环结构:
cmp、test、jmp、jxxx

实现函数调用:
push、pop、call、ret

实现数据转移:
mov

AT&T格式汇编语言

通常用在Unix、Linux系统

Intel格式是Windows常用格式

-	AT&T格式	Intel格式
目的操作数d、源操作数s	op s, d	op d, s
寄存器的表示	mov %ebx, %eax	mov eax, ebx
立即数的表示	mov $985, %eax	mov eax, 985
主地址的表示	mov %eax, (af996h)	mov [af996h], eax
读写长度表示	movb $5, (af996) movw $5, (af996h) movl $5, (af996h) addb $4, (af996)	mov byte ptr[af996h], 5 mov word ptr[af996h], 5 mov dword ptr [af996h], 5 add byte ptr [af996h], 4
主存地址偏移量的表示	movl -8(%ebx), %eax movl 4(%ebx, %ecx, 32), %eax *偏移量(基址,变址,比例因子)*	mov eax, [ebx-8] mov eax, [ebx+ecx*32+4] *[基址+变址*比例因子+偏移量]*

实现选择语句

程序计数器PC在Intel x86处理器通常被称作IP

从jmp指令引入

无条件转移指令：`jmp <地址> # PC无条件转移至<地址>`

```
mov eax, 7
mov ebx, 6
jmp NEXT    # 用“标号”锚定位置，名字可自取。类似goto
mov ecx, ebx
NEXT:
mov ecx, eax
```

条件转移指令：jxxx

两数比较：`cmp a, b`

```
cmp a, b  # 两数比较
je <地址>     # jump when equal, a==b时跳转
jne <地址>    # jump when not equal, a!=b时跳转
jg <地址>     # jump when greater then, a>b时跳转
jge <地址>    # jump when greater then or equal, a>=b时跳转
jl <地址>     # jump when less then, a<b时跳转
jle <地址>    # jump when less then or equal, a<=b时跳转

"""配合标号的跳转例子"""
cmp eax, ebx
jg NEXT

"""将c翻译成汇编的例子"""
'''c
if(a>b){
  c = a;
} else {
  c = b;
}
'''
'''翻译后'''
cmp eax, ebx
jg NEXT       # 若满足a>b，跳转到NEXT。否则进入else（如下内容）
mov ecx, ebx
jmp END
NEXT:
mov ecx, eax
END:
```

用条件转移指令实现循环

```
"""将一个c语言的循环语句翻译为汇编"""
'''c
int result = 0;
for(int i=1; i<=100; i++){
  result += i;
}
'''
'''翻译后'''
move eax, 0     # eax保存result
mov edx, 1      # edx保存i
cmp edx, 100
jg L2           # 若i>100，跳转到L2
L1:             # 循环主体
add eax, edx    # 实现result += i
inc edx         # 自增指令，实现i++
cmp edx, 100
```

```
jle L1          # 若i<=100，跳转到L1
L2:             # 跳出循环
```

用loop实现循环

```
"""将一个c语言的循环语句翻译为汇编"""
'''c
for(int i=500; i>0; i--){
  <执行操作>
}
'''
'''翻译后'''
mov ecx, 500
Looptop:        # 循环开始
<执行操作>
loop Looptop    # ecx--，若exc!=0，跳转到Looptop
'''
loop Looptop等价于：
dec ecx
cmp ecx, 0
jne Looptop
'''
```

补充：loopnz、loopz

loopnz：loop not zero，当exc!=0且ZF==0时，继续循环

loopz：loop zero，当exc!=0且ZF==1时，继续循环

实现函数调用

函数调用指令：`call <函数名>`
函数返回指令：`ret`

```
caller:
... ...
call add    # 调用add
... ...
leave
ret

add:        # add的功能
... ...
leave
ret         # 返回call并继续运行
```

访问栈帧数据：push、pop指令

```
push eax        # 将寄存器eax的值压栈
push 986        # 将立即数985压栈
push [ebp+8]    # 将主地址[ebp+8]里的数据压栈

pop eax         # 栈顶元素出栈，写入寄存器eax
pop [ebp+8]     # 栈顶元素出栈，写入主存地址[ebp+8]
```

访问栈帧数据：mov指令

通过add和sub调整当前函数的栈帧范围，
用mov指令结合esp和ebp访问栈帧数据。

```
sub esp, 12         # 顶栈指针-12
mov [esp+8], eax    # 将eax的值复制到主存[esp+8]
mov [esp+4], 985    # 将985复制到主存[esp+4]
mov eax, [ebp+8]    # 将主存[ebp+8]的值复制到eax
mov [esp], eax      # 将eax的值复制到主存[esp]
add esp, 8          # 顶栈指针+8
```

切换栈帧

call指令的作用：

1. 将IP旧值压栈保存（相当于`push IP`）
2. 设置IP新值，无条件转移至被调用函数的第一条指令（效果相当于`jmp add`）

当一个函数return之前，只需要如下指令，即可让esp和ebp重新只会上一层函数的栈帧（**效果等价于leave**）：

```
mov esp, ebp    # 让esp指向当前栈顶的底部
pop ebp         # 将esp所指元素出栈，写入寄存器ebp
# 效果等价于leave
```

传递参数和返回值

将*局部变量*集中存储在栈帧*底部*区域；
将*调用参数*集中存储在栈帧*顶部*区域。

栈帧最*底部*一定是*上一层栈帧基址*（ebp旧值）；
栈帧最*顶部*一定是*返回地址*（当前函数的栈帧除外）。

汇编中调用函数的情况：

- **调用者**：
  - 保存必要的寄存器
    - eax、edx、ecx等
  - 将调用参数写入当前栈帧的顶部区域

        - push或mov可实现
    - 执行call指令
        - 返回之地亚茹顶栈、并跳转到被调用函数的第一条指令
    - 使用返回值
        - 通过eax寄存器
    - 回复必要的寄存器
- **被调用者**:
    - 保存上一层函数的栈帧，设置当前函数的栈顶
        - push ebp
          mov ebp, esp
        - 或enter指令
    - 初始化局部变量
        - [ebp-4]、[ebp-8]...
    - 一系列逻辑处理
    - 向上层函数传递返回值
        - 通过eax寄存器
    - 恢复上一层函数的栈帧
        - mov esp, ebp
          pop ebp
        - 或leave指令
    - 执行ret指令
        - 从栈顶找到返回地址，出栈并恢复IP值

CISC、RISC

CISC和RISC是指令系统的两种设计方向

CISC = Complex Instruction Set Computer复杂指令集计算机系统，
一条指令可以完成一个复杂的基本功能，
主要用于x86架构。

RISC = Reduced Instruction Set Computer精简指令集计算机系统，
一条指令完成一个基本操作，多条指令组成完成一个复杂的基本功能，
主要用于ARM架构。