



南开大学

NANKAI UNIVERSITY, P.R. CHINA 1919

允公允能 日新月异



汇编与逆向技术基础

第4章 数据传送与寻址

王志

zwang@nankai.edu.cn

updated on 24th Mar. 2020

College of Cyber Science
Nankai University
2019/2020



上 一 章 课 件 的 反 馈

- MASM内部以数据位的个数定义了多种数据类型
 - BYTE, db, 8位
 - var1 db 10h, 20h
 - var1 BYTE 10h, 20h
 - WORD, dw, 16位
 - DWORD, dd, 32位
 - QWORD , dq, 64





允公允能 日新月异

本章知识点

1. 数据传送指令
2. 加法和减法
3. 和数据相关的操作符和伪指令
4. 间接寻址





南开大学

NANKAI UNIVERSITY, P.R. CHINA 1919

允公允能 日新月异



1. 数据传输指令



汇编语言与高级语言

- 汇编语言与高级语言最根本的不同之处在于，程序员必须掌握**内存中的数据**存储和**机器与系统相关的大量细节**
- 汇编语言给了程序员极大的**自由**，可以直接与机器对话，不需要依靠各种“翻译人员”





允公允能 日新月异

指令

- 一条汇编语句
 - 标号 (identifier)
 - 指令助记符 (操作码opcode)
 - 操作数 (operand)
 - 注释 (comment)





操作数类型

- 立即数 (immediate)

- `mov eax, 10h`

- 寄存器 (register)

- `inc eax`

- 内存 (memory)

- `mov eax, [ebp+8]`





允公允能 日新月异

寄存器操作数的简写符号

- **r8**, 8位通用寄存器
- **r16**, 16位通用寄存器
- **r32**, 32位通用寄存器
- **reg**, 任意的通用寄存器
- **sreg**, 16位段寄存器





允公允能 日新月异

立即数操作数的简写符号

- **imm**, 8位、16位或32位立即数
- **imm8**, 8位立即数
- **imm16**, 16位立即数
- **imm32**, 32位立即数



南开大学
Nankai University

内存操作数的简写符号

- **r/m8**, 8位通用寄存器或内存操作数
- **r/m16**, 16位通用寄存器或内存操作数
- **r/m32**, 32位通用寄存器或内存操作数
- **mem**, 8位、16位、32位内存操作数





允公允能 日新月异

MOV指令

- mov指令从源操作数向目的操作数复制数据
 - mov destination, source
 - C++中, destination = source





允公允能 日新月异

MOV指令

- 两个操作数的尺寸必须一致
- 两个操作数不能同时为内存操作数
- 目的操作数不能是CS、EIP和IP
- 立即数不能直接送至段寄存器





允公允能 日新月异

mov指令

- **mov** — Move (Opcodes: 88, 89, 8A, 8B, 8C, 8E, ...)
- 语法
 - mov <reg>, <reg>
 - mov <reg>, <mem>
 - mov <mem>, <reg>
 - mov <reg>, <imm>
 - mov <mem>, <imm>
- 例子
 - mov byte ptr [var], 5





直接内存操作数

.data

var1 DWORD 1000h ; 内存位置偏移00403000h

.code

mov EAX, var1 ; 机器指令 A100304000

- 变量名（数据标号）
 - 数据段内偏移地址





内存寻址操作

- masm32使用方括号表示内存寻址操作
 - `mov eax, [var1]`
- 通常，直接内存操作数不使用中括号
 - `mov eax, var1`
- 涉及到算术表达式时，使用中括号
 - `mov eax, [var1+5]`





允公允能 日新月异

invalid instruction operands

```
.data
    var1 DWORD 1000h
    var2 DWORD 2000h

.code

start:
    mov eax, var1
    mov var1, var2
    invoke ExitProcess, 0

end start
```

```
D:\>\masm32\bin\ml /c /coff hello.asm
Microsoft (R) Macro Assembler Version 6.14.8444
Copyright (C) Microsoft Corp 1981-1997. All rights reserved.

Assembling: hello.asm

*****
ASCII build
*****

hello.asm(29) : error A2070: invalid instruction operands
```





内存之间的数据移动

.data

var1 DWORD 0

var2 DWORD 100h

.code

mov eax, var2

mov var1, eax





整数的零扩展

- 复制尺寸较小的操作数到尺寸较大的操作数
- **MOVZX指令**(move with zero-extend)
 - `movzx r32, r/m8`
 - `movzx r32, r/m16`
 - `movzx r16, r/m8`





允公允能 日新月异

MOVSX

- MOVSX (move with sign-extend) 符号扩展传送指令，最高位循环填充所有扩展位
 - 有符号整数的存储空间扩展
 - movsx r32, r/m8
 - movsx r32, r/m16
 - movsx r16, r/m8





允公允能 日新月异

LAHF指令

- **LAHF** (load status flags into AH) 指令把EFLAGS寄存器的低字节复制到AH寄存器
 - 符号标志 (SF)
 - 零标志(ZF)
 - 辅助进位标志(AF)
 - 奇偶标志(PF)
 - 进位标志(CF)





允公允能 日新月异

SAHF指令

- SAHF (store AH into status flags) 指令复制AH寄存器的值至EFLAGS寄存器的低字节
 - 修改CPU的符号标志 (SF)、零标志(ZF)、辅助进位标志(AF)、奇偶标志(PF)、进位标志(CF)





XCHG指令

- **XCHG** (exchange data) 指令交换两个操作数的内容
 - XCHG reg, reg
 - XCHG reg, mem
 - XCHG mem, reg





交换两个内存的值

.data

var1 DWORD 100h

var2 DWORD 200h

.code

mov eax, var1

xchg eax, var2

mov var1, eax





直接偏移操作数

- 在变量名后面加上一个偏移值，可以创建直接偏移（**direct-offset**）操作数
- 访问没有显式标号的内存





直接偏移操作数

```
.data  
    var1 DWORD 1000h, 2000h, 3000h, 4000h
```

```
.code
```

```
start:
```

```
    mov eax, var1  
    mov eax, [var1+1]  
    mov eax, [var1+2]  
    invoke ExitProcess, 0
```

窗口 - 主线程, 模块 hello		
00	A1 00304000	MOV EAX, DWORD PTR DS:[403000]
05	A1 01304000	MOV EAX, DWORD PTR DS:[403001]
0A	A1 02304000	MOV EAX, DWORD PTR DS:[403002]
0F	6A 00	PUSH 0
11	E8 00000000	CALL <JMP.&kernel32.ExitProcess>

地址	十六进制数据	多
00403000	00 10 00 00 00 20 00 00 00 30 00 00 00 40 00 00	
00403010	00 20 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
00403020	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
00403030	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	





南開大學

NANKAI UNIVERSITY, P.R. CHINA 1919

允公允能 日新月異



2. 加法和減法



允公允能 日新月异

INC指令

- INC (increment) 指令从操作数中加1
- 语法
 - `inc <reg>`
 - `inc <mem>`
- 例子
 - `inc eax`
 - `inc [var1+4]`





允公允能 日新月异

DEC指令

- **DEC** (decrement) 指令从操作数中减1

- 语法

dec <reg>

dec <mem>

- 例子

inc eax

inc [var1+4]



南开大学
Nankai University



允公允能 日新月异

ADD指令

- ADD指令将同尺寸的源操作数和目的操作数相加

add <reg>,<reg>

add <reg>,<mem>

add <mem>,<reg>

add <reg>,<imm>

add <mem>,<imm>

- 相加的结果存储在目的操作数中
 - ADD 目的操作数，源操作数
 - 影响标志位CF、ZF、SF、OF、AF、PF





允公允能 日新月异

SUB指令

- SUB指令将源操作数从目的操作数中减掉

sub <reg>,<reg>

sub <reg>,<mem>

sub <mem>,<reg>

sub <reg>,<imm>

sub <mem>,<imm>

- SUB 目的操作数， 源操作数
- 影响的标志位有CF、ZF、SF、OF、AF、PF





允公允能 日新月异

NEG指令

- NEG (negate) 指令通过将数字转换为对应的补码而求得其相反数
 - neg <reg>
 - neg <mem>
- 影响的标志位: CF、ZF、SF、OF、AF、PF





南開大學

NANKAI UNIVERSITY, P.R. CHINA 1919

允公允能 日新月異



3. 和数据相关的操作符和伪指令



允公允能 日新月异

和数据相关的操作符和伪指令

- OFFSET操作符
- ALIGN伪指令
- PTR操作符
- TYPE操作符
- LENGTHOF操作符
- SIZEOF操作符
- LABEL伪指令





OFFSET操作符

- OFFSET操作符返回数据标号的偏移地址
- 偏移地址表示标号距离数据段开始的距离
 - CS的值一般是0
 - CS为零的时候，OFFSET等同内存虚拟地址





OFFSET操作符

```
.data
    var1 DWORD 1000h
    var2 DWORD 2000h
.code
start:
    mov eax, OFFSET var1
    mov ebx, OFFSET var2
    invoke ExitProcess, 0

end start
```

B8 00304000	MOV EAX, OFFSET 00403000
BB 04304000	MOV EBX, OFFSET 00403004
6A 00	PUSH 0
E8 01000000	CALL <JMP. &kernel32.ExitProcess>





ALIGN伪指令

- ALIGN指令将变量的位置按BYTE、WORD、DWORD边界对齐
 - ALIGN 边界值
 - 边界值可以是1、2、4、8或16（ a power of 2 ）
 - “Aligned data can improve **performance**, at the expense of wasted **space** between data elements.”





ALIGN伪指令

```
.data  
var1 BYTE 10h, 20h  
var2 DWORD 0AAAAAAAAh  
ALIGN 4  
var3 DWORD 0BBBBBBBBh  
.
```

地址	十六进制数据															
00403000	10	20	AA	AA	AA	AA	00	00	BB	BB	BB	BB	00	00	00	00
00403010	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00403020	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00403030	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00403040	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00403050	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00





PTR操作符

- PTR操作符可以重载操作数声明的默认尺寸

```
.data
    var1 DWORD 12345678h

.code
start:
    movzx eax, BYTE PTR var1
    movzx ebx, BYTE PTR [var1+1]
    invoke ExitProcess, 0
```





ALIGN

- When data is aligned, the skipped space is **padded with zeroes**. When instructions are aligned, the skipped space is **filled with appropriately-sized NOP instructions**.
- <https://docs.microsoft.com/en-us/cpp/assembler/masm/align-masm?view=msvc-160>





TYPE操作符

- TYPE操作符返回变量的字节数

.data

var1 BYTE 0

var2 WORD 0

var3 DWORD 0

.code

mov eax, TYPE var2





LENGTHOF操作符

- LENGTHOF操作符计算数组中元素的数目，元素由出现在同一行的值定义

.data

```
var1 DWORD 0, 1, 2, 3
```

.code

```
mov eax, LENGTHOF var1
```





LENGTHOF操作符

.data

```
var1  DWORD  0,  1,  2,  3  
      DWORD  4,  5,  6,  7
```

.code

```
mov eax, LENGTHOF var1
```





LENGTHOF操作符

.data

```
var1 DWORD 0, 1, 2, 3,  
         4, 5, 6, 7
```

.code

```
mov eax, LENGTHOF var1
```

- 第一行的最后加一个逗号，连接下一行的初始值





SIZEOF操作符

- SIZEOF操作符的返回值等于LENGTHOF和TYPE返回值的乘积

.data

```
var1 DWORD 0, 1, 2, 3,  
         4, 5, 6, 7
```

.code

```
mov eax, SIZEOF var1
```





LABEL伪指令

- LABEL伪指令允许插入一个标号，并赋予其尺寸属性而**无须分配任何实际的存储空间**。
- 为数据段内其后定义的变量提供一个**别名**





Label 伪指令

.data

dw_var LABEL DWORD

var1 WORD 1234h

var2 WORD 5678h

.code

mov eax, dw_var

eax 等于 56781234h





南开大学

NANKAI UNIVERSITY, P.R. CHINA 1919

允公允能 日新月异



4. 间接寻址



允公允能 日新月异

间接寻址

- 用寄存器作为指针并控制该寄存器的值称为间接寻址（indirect addressing）
- 如果一个操作数使用的是间接寻址，就称之为间接操作数（indirect operand）。





间接操作数

- 任何一个 32 位通用寄存器（EAX、EBX、ECX、EDX、ESI、EDI、EBP 和 ESP）加上方括号就能构成一个间接操作数





间接操作数

.data

val DWORD 12345678h

.code

mov esi, OFFSET val

mov eax, DWORD PTR [esi]





间接操作数

.data

array_dw DWORD 10000h, 20000h, 30000h

.code

mov esi, OFFSET array_dw

mov eax, [esi] ; (第一个数)

add esi, 4

add eax, [esi] ; (第二个数)

add esi, 4

add eax, [esi] ; (第三个数)





变址操作数

- 变址操作数（indexed operand）把常量和寄存器相加得到一个有效地址
- 任何32位通用寄存器都可以作为变址寄存器
 - $\text{constant}[\text{reg}]$
 - $[\text{constant}+\text{reg}]$





变址操作数

.data

array_dw DWORD 10000h, 20000h, 30000h

.code

mov esi, 0

mov eax, array_dw[esi] ; (第一个数)

add esi, 4

add eax, array_dw[esi] ; (第二个数)

add esi, 4

add eax, array_dw[esi] ; (第三个数)





变址操作数

.data

array_dw DWORD 10000h, 20000h, 30000h

.code

mov esi, OFFSET array_dw

mov eax, [esi] ; (第一个数)

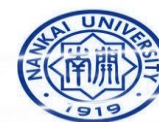
add eax, [esi+4] ; (第二个数)

add eax, [esi+8] ; (第三个数)



变址操作数的比例因子

```
.data  
array_dw DWORD 10000h, 20000h, 30000h  
  
.code  
mov esi, 0  
mov eax, array_dw[esi*TYPE array_dw] ;  
mov esi, 1  
add eax, array_dw[esi* TYPE array_dw]  
mov esi, 2  
add eax, array_dw[esi* TYPE array_dw]
```





允公允能 日新月异

指针

- 如果一个变量包含另一个变量的地址，则该变量称为指针





允公允能 日新月异

指针

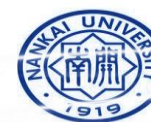
.data

array_b BYTE 10h, 20h, 30h, 40h

array_w WORD 1000h, 2000h, 3000h

ptr_b DWORD array_b

ptr_w DWORD array_w





允公允能 日新月异

指针

.data

array_b BYTE 10h, 20h, 30h, 40h

array_w WORD 1000h, 2000h, 3000h

ptr_b DWORD OFFSET array_b

ptr_w DWORD OFFSET array_w





允公允能 日新月异

指针

- 32位模式下的NEAR指针和FAR指针
- **NEAR指针（课程使用NEAR指针）**
 - 相对数据段开始的32位偏移地址
- FAR指针
 - 48位的段选择子-偏移地址





TYPEDEF操作符

- TYPEDEF操作符允许创建用户自定义的类型
 - PBYTE **TYPEDEF** PTR BYTE ;字节指针
 - PWORD **TYPEDEF** PTR WORD ;字指针
 - PDWORD **TYPEDEF** PTR DWORD ;双字指针





TYPEDEF操作符

PADWORD TYPEDEF PTR DWORD

.data

array1 DWORD 1000h, 2000h, 3000h, 4000h

ptr1 **PADWORD** array1





本章学习的知识点

1. 数据传送指令
2. 加法和减法
3. 和数据相关的操作符和伪指令
4. 间接寻址

