汇编语言与逆向技术

第4章 数据传送、寻址和算术运算

王志
zwang@nankai.edu.cn

南开大学 网络空间安全学院

2024-2025学年

本章知识点

- 数据传送指令
  - 重点：mov、movzx、movsx
- 算术运算指令
- 伪指令和操作符
  - 难点：offset、ptr、label
- 循环语句
- 内存操作数与寻址方式

数据传送指令

投票 最多可选1项

设置

汇编语言中用的最多的指令是?

A 数据传送指令

B 算术运算指令

C 跳转指令

D 函数调用指令

提交

南開大學 Nankai University

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

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

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

主观题 10分

设置

思考题：为什么mov指令的操作码有很多个？
88, 89, 8A, 8B, 8C, 8E, ...

作答

直接内存操作数

```
.data
    var1 DWORD 1000h ;
.code
    mov EAX, var1 ;
```

- 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]

多选题 10分

设置

以下哪种MOV指令格式不符合规则

A mov mem, reg

B mov mem, imm

C mov mem, mem

D mov reg, mem

提交

主观题 10分

设置

```
.data
    var1 DWORD 0
    var2 DWORD 100h
.code
    ;?
    ; 如何将var2的数值赋值给var1
```

正常使用主观题需2.0以上版本雨课堂

作答

南開大學 Nankai University

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 reserve

 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

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

填空题 20分

设置

```
.data
    var1 BYTE 10h
.code
    movzx eax, var1
    movsx ebx, var1
```

eax寄存器的十六进制值是 [填空1]

ebx寄存器的十六进制值是 [填空2]

正常使用填空题需3.0以上版本雨课堂

作答

填空题 20分

设置

```
.data
    var1 BYTE 0A0h
.code
    movzx eax, var1
    movsx ebx, var1
```

eax寄存器的十六进制值是 [填空1]

ebx寄存器的十六进制值是 [填空2]

正常使用填空题需3.0以上版本雨课堂

作答

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)

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

主观题 10分

设置

```
.data
    var1 DWORD 100h
    var2 DWORD 200h
.code
    ;?
    ; 如何交换var1和var2的值？
```

正常使用主观题需2.0以上版本雨课堂

作答

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
```

算术运算指令

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]

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]

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

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

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

填空题 10分

设置

.data

var1 DWORD 1000h

.code

neg var1; NEG操作之后，var1的十六进制值是

[填空1]

正常使用填空题需3.0以上版本雨课堂

作答

南開大學 Nankai University

伪指令和操作符

数据相关的伪指令

- BYTE、WORD、DWORD
- ALIGN伪指令
- LABEL伪指令

Directives Reference

08/04/2021 • 2 minutes to read • +2

Data Allocation

ALIGN	FWORD	REAL8
BYTE	LABEL	REAL10
SBYTE	ORG	TBYTE
DWORD	QWORD	WORD
SDWORD	REAL4	SWORD
EVEN		

数据相关的操作符（Operator）

- PTR操作符
- TYPE操作符
- LENGTHOF操作符
- SIZEOF操作符
- OFFSET操作符

MASM Operators reference

08/04/2021 • 2 minutes to read • +3

Type

HIGH (high 8 bits of lowest 16 bits)
HIGH32 (high 32 bits of 64 bits)
HIGHWORD (high 16 bits of lowest 32 bits)
LENGTH (number of elements in array)
LENGTHOF (number of elements in array)
LOW (low 8 bits)
LOW32 (low 32 bits)
LOWWORD (low 16 bits)
OPATTR (get argument type info)
PTR (pointer to or as type)
SHORT (mark short label type)
SIZE (size of type or variable)
SIZEOF (size of type or variable)
THIS (current location)
TYPE (get expression type)
.TYPE (get argument type info)

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>
```

单选题 10分

设置

OFFSET操作符获得的偏移地址占用几个字节？

A 1字节

B 2字节

C 3字节

D 4字节

提交

南開大學 Nankai University

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

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

单选题 10分

设置

判断题：ALIGN伪指令可以对代码段的指令进行对齐操作。

A 正确

B 错误

提交

南開大學 Nankai University

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
```

填空题 10分

设置

```
.data
        var1 DWORD 12345678h
.code
        movzx eax, BYTE PTR var1
; 寄存器eax的十六进制值是 [填空1]
```

正常使用填空题需3.0以上版本雨课堂

作答

南開大學 Nankai University

填空题 10分

设置

```
.data
        var1 WORD 1234h
        var2 WORD 5678h
.code
        mov eax, DWORD PTR var1
; 寄存器eax的十六进制值是 [填空1]
```

正常使用填空题需3.0以上版本雨课堂

作答

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

填空题 10分

设置

```
.data
        w_var LABEL WORD
        var1 DWORD 12345678h
.code
    movzx eax, w_var
    ; eax的十六进制值是 [填空1]
```

正常使用填空题需3.0以上版本雨课堂

作答

南開大學 Nankai University

循环语句

主观题 10分

设置

思考：CPU的控制流跳转是如何实现的?

作答

控制转移

- 控制转移（transfer of control）是一种改变汇编语句执行顺序的方法。
  - 无条件转移
  - 条件转移

无条件转移 Unconditional Jump

- 将CPU控制权直接转移到指定的汇编语句
  - 修改EIP为指定的内存地址
  - CPU从EIP指定的内存地址读取下一条机器指令

JMP指令

- JMP 目的地址
- JMP指令实现CPU控制权的无条件跳转
- 目的地址是代码标号
  - 代码标号被汇编器翻译成内存地址
  - CPU看到的是内存地址，不是代码标号

循环

top：

… …

… …

JMP top

无限循环

主观题 10分

设置

条件跳转指令的判断条件存储在哪里?

作答

南開大學 Nankai University

条件跳转指令Conditional Jump

- 条件指令j<condition>，通过条件判断来修改CPU控制流

有符号数的条件跳转指令
Conditional jump instructions used on signed data

Instruction	Description	Flags tested
JE/JZ	Jump Equal or Jump Zero	ZF
JNE/JNZ	Jump not Equal or Jump Not Zero	ZF
JG/JNLE	Jump Greater or Jump Not Less/Equal	OF, SF, ZF
JGE/JNL	Jump Greater/Equal or Jump Not Less	OF, SF
JL/JNGE	Jump Less or Jump Not Greater/Equal	OF, SF
JLE/JNG	Jump Less/Equal or Jump Not Greater	OF, SF, ZF

条件跳转指令 Conditional Jump

无符号数的条件跳转指令

Conditional jump instructions used on unsigned data

Instruction	Description	Flags tested
JE/JZ	Jump Equal or Jump Zero	ZF
JNE/JNZ	Jump not Equal or Jump Not Zero	ZF
JA/JNBE	Jump Above or Jump Not Below/Equal	CF, ZF
JAE/JNB	Jump Above/Equal or Jump Not Below	CF
JB/JNAE	Jump Below or Jump Not Above/Equal	CF
JBE/JNA	Jump Below/Equal or Jump Not Above	AF, CF

条件跳转指令 Conditional Jump

标志位和特殊用途的条件跳转指令

conditional jump instructions have special uses and check the value of flags

Instruction	Description	Flags tested
JXCZ	Jump if CX is Zero	none
JC	Jump If Carry	CF
JNC	Jump If No Carry	CF
JO	Jump If Overflow	OF
JNO	Jump If No Overflow	OF
JP/JPE	Jump Parity or Jump Parity Even	PF
JNP/JPO	Jump No Parity or Jump Parity Odd	PF
JS	Jump Sign (negative value)	SF
JNS	Jump No Sign (positive value)	SF

主观题 10分

设置

如何进行跳转条件的判断?

作答

CMP指令

- CMP指令，比较目的操作数和源操作数
  - CMP reg，reg
  - CMP reg，imm
  - CMP mem，reg
  - CMP mem，imm
  - CMP reg，mem

CMP指令

- 执行从源操作数中减掉目的操作数的减法操作
  - 用于条件跳转指令的条件判断
  - 不改变目的操作数和源操作数，只影响eflags的标志位
- 设置相应的标志位
- 标志位：OF、SF、ZF、AF、PF、CF

```
    MOV EAX，100h
    MOV EBX, 200h
    CMP EAX, EBX
    JA L1
    INVOKE StdOut，ADDR str1
    JMP L2
L1：
    INVOKE StdOut，ADDR str2 ;
L2：
    INVOKE ExitProcess，0
```

LOOP指令

- LOOP 目的地址
- LOOP指令可以指定循环执行的次数 （loop count）
  - ECX寄存器作为循环计数器
  - LOOP指令执行时，ECX减1
  - 如果ECX不等于0，跳转到目的地址
  - 如果ECX等于0，不跳转，顺序执行

LOOP指令

MOV EAX 10h

MOV ECX 10h

L1：

INC EAX

LOOP L1

设置

```
MOV EAX 10h
MOV ECX 10h
L1:
    INC EAX
    LOOP L1
```

LOOP循环结束后，EAX寄存器的值为 [填空1]

正常使用填空题需3.0以上版本雨课堂

作答

LOOP指令

- LOOP指令先ecx减1，然后判断ecx是否为0.
- LOOP is exactly like dec ecx / jnz

填空题 10分

设置

```
MOV EAX 10h
MOV ECX 0
L1：
    INC EAX
    LOOP L1
```

如果ECX的初始值为0，LOOP会循环执行 [填空1] 次。

正常使用填空题需3.0以上版本雨课堂

作答

南開大學 Nankai University

循环的嵌套

```
.data
    count DWORD 0
.code
    MOV ECX, 100；  L1 循环100次
L1：
    MOV count, ECX
    MOV ECX, 10 ; L2 循环10次
L2:
    … ….
    LOOP L2
    MOV ECX，  count
    LOOP L1
```

主观题 10分

设置

```
.data
    array DWORD 100h, 200h, 300h, 400h
```

编写汇编代码，计算数组array的和

正常使用主观题需2.0以上版本雨课堂

作答

数组求和

```
.data
    array DWORD 100h, 200h, 300h, 400h
.code
    MOV ECX, LENGTHOF array ; 循环次数
    MOV EDI，OFFSET array；索引
    MOV EAX，0；和
L1：
    ADD EAX，[EDI]
    ADD EDI, TYPE array
    LOOP L1
```

主观题 10分

设置

```
.data
    src BYTE “Hello World”, 0Dh, 0Ah, 0
    dst BYTE SIZEOF src DUP(0), 0
```

使用LOOP指令，将字符串src复制到dst

正常使用主观题需2.0以上版本雨课堂

作答

南開大學 Nankai University

字符串赋值

```
.data
    src BYTE “Hello World”, 0Dh, 0Ah, 0
    dst BYTE SIZEOF src DUP(0), 0
.code
    MOV ECX, SIZEOF src ；循环次数
    MOV ESI, 0 ；字符索引
L1:
    MOV AL, BYTE PTR src[ESI]
    MOV BYTE PTR dst[ESI], AL
    INC ESI
    LOOP L1
```

主观题 10分

设置

.data
    num BYTE 1, 2, 3, 4, 5, 6, 7, 8, 9, 0
.code
    将数字转换成对应的ASCII字符，输出到命令行窗口
ASCII 编码：
‘0’ 30h
‘1’31h
… …
‘9’39h

正常使用主观题需2.0以上版本雨课堂

作答

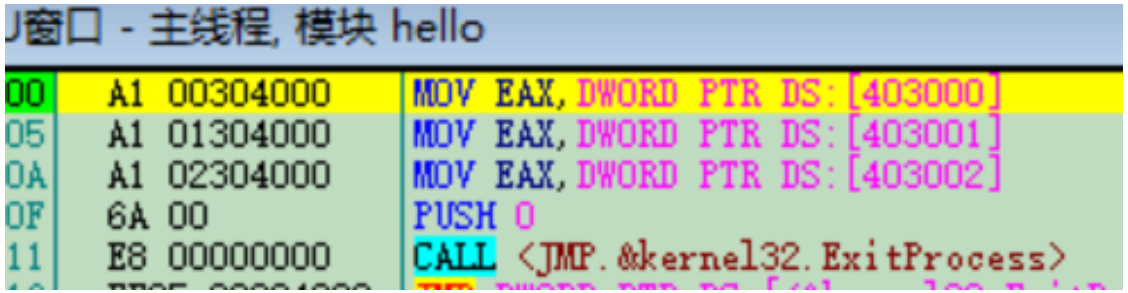
南開大學 Nankai University

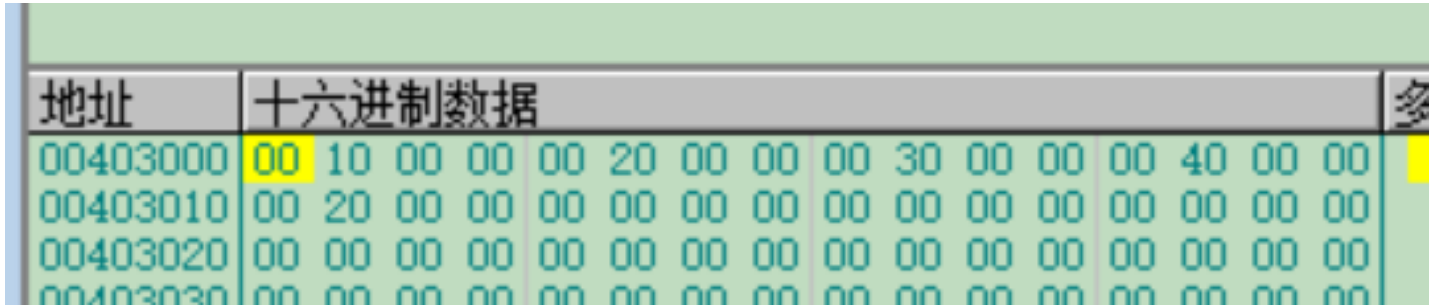
```
.data
    num BYTE 1， 2， 3， 4， 5， 6， 7， 8， 9， 0，0
.code
    MOV ECX, 10 ； 循环次数
    MOV ESI, 0 ； 索引
L1:
    MOV AL, BYTE PTR num[ESI]
    ADD AL，30h
    MOV BYTE PTR num[ESI], AL
    INC ESI
    LOOP L1
```

内存操作数与寻址方式

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
```





填空题 10分

设置

```
.data
    var1 DWORD 1000h, 2000h, 3000h
.code
    mov eax, [填空1] ; 如何访问没有显式标号的内存值2000h
```

正常使用填空题需3.0以上版本雨课堂

作答

南開大學 Nankai University

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

主观题 10分

设置

“inc [eax]”指令在编译的时候为什么会出错？

正常使用主观题需2.0以上版本雨课堂

作答

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以得到一个有效地址
- 任何32位通用寄存器都可以作为变址寄存器
  - constant[*reg*]
  - [constant+*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]
```

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

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
```

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
```

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
```

单选题 10分

设置

PBYTE TYPEDEF PTR BYTE
.data
var1 BYTE 10h
ptr1 PBYTE var1 ; 变量ptr1的尺寸？

A 1 BYTE

B 2 BYTE

C 3 BYTE

D 4 BYTE

提交

汇编语言与逆向技术

第4章 数据传送、寻址和算术运算

王志
zwang@nankai.edu.cn

南开大学 网络空间安全学院

2024-2025学年