Assembly Language for Intel-Based Computers, 5th Edition

Kip Irvine

Capítulo 4 – Parte B:

Operadores e Endereçamento de Dados; Instruções Loop e Jump;

Slides prepared by the author

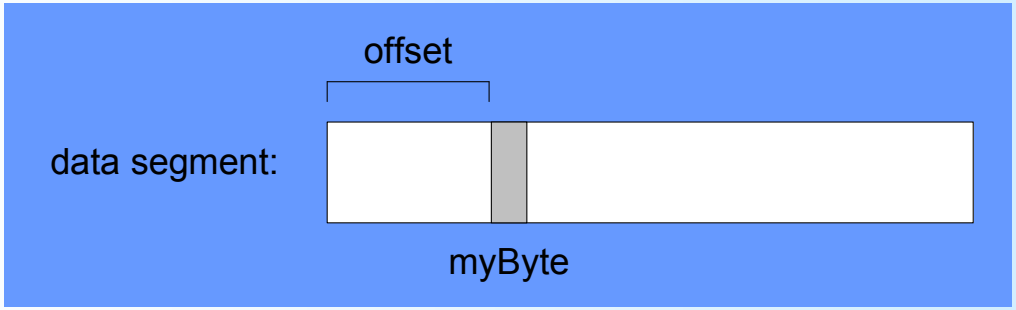
Revision date: June 4, 2006

Operadores e diretivas relacionados a dados

- Operador OFFSET
- Operador PTR
- Operador TYPE
- Operador LENGTHOF
- Operador SIZEOF
- Diretiva LABEL

Operador OFFSET

- OFFSET retorna a distância em bytes, de um label, do início do segmento
  - Modo Protegido: 32 bits
    - *Na prática, offset = endereço de um dado na memória*



Os programas em modo protegido tem apenas um segmento (usando o modelo de memória flat).

Exemplos de OFFSET

Assumindo que o segmento de dados começa em 00404000h:

```
.data
bVal BYTE ?
wVal WORD ?
dVal DWORD ?
dVal2 DWORD ?

.code
mov esi,OFFSET bVal          ; ESI = 00404000h
mov esi,OFFSET wVal          ; ESI = 00404001h
mov esi,OFFSET dVal          ; ESI = 00404003h
mov esi,OFFSET dVal2         ; ESI = 00404007h
```

Relacionando a C/C++

O valor retornado pelo OFFSET é um ponteiro. Comparar o seguinte código escrito em C++ e linguagem assembly:

```
; C++ version:
char array[1000];
char * p = array;
```

```
.data
array BYTE 1000 DUP(?)
.code
mov  esi,OFFSET array          ; ESI is p
```

Operador PTR

Sobrepõe o tipo default de um label (variável).

Provê a flexibilidade de acesso à parte de uma variável.

```
.data
myDouble DWORD 12345678h
.code
mov ax,myDouble                    ; error – why?

mov ax,WORD PTR myDouble           ; loads 5678h

mov WORD PTR myDouble,4321h        ; saves 4321h
```

Lembrar a ordem de armazenamento little endian.

Ordenamento Little Endian

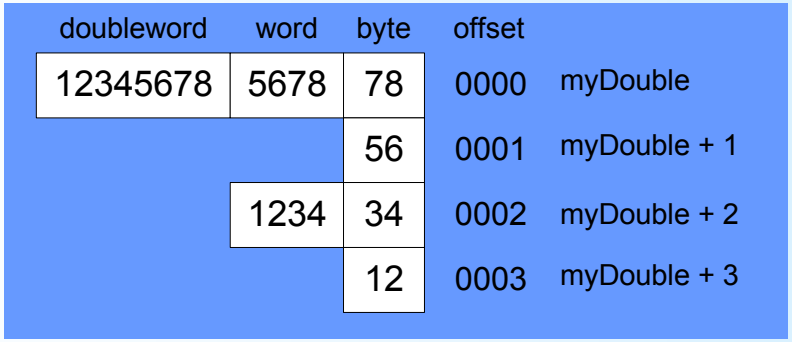
- O ordenamento Little endian refere-se a forma com que IA-32 guarda dados na memória.
- Por exemplo, o doubleword 12345678h é guardado na forma:

byte	offset
78	0000
56	0001
34	0002
12	0003

Quando os inteiros são carregados da memória para registradores, os bytes são automaticamente reordenados nas posições corretas.

Exemplo de operador PTR

```
.data
myDouble DWORD 12345678h
```




```
mov al,BYTE PTR  myDouble          ; AL = 78h
mov al,BYTE PTR [myDouble+1]       ; AL = 56h
mov al,BYTE PTR [myDouble+2]       ; AL = 34h
mov ax,WORD PTR  myDouble          ; AX = 5678h
mov ax,WORD PTR [myDouble+2]       ; AX = 1234h
```

Operador PTR (cont)

PTR pode também ser usado para combinar elementos de um tipo de dados menor, movendo-o a um operando maior . O CPU reverte automaticamente os bytes.

```
.data
myBytes BYTE 12h,34h,56h,78h

.code
mov ax,WORD PTR [myBytes]          ; AX = 3412h
mov ax,WORD PTR [myBytes+2]        ; AX = 7856h
mov eax,DWORD PTR myBytes          ; EAX = 78563412h
```

Sua vez . . .

Escrever o valor de cada operando destino:

```
.data
varB BYTE 65h,31h,02h,05h
varW WORD 6543h,1202h
varD DWORD 12345678h

.code
mov ax,WORD PTR [varB+2]          ; a.
mov bl,BYTE PTR varD              ; b.
mov bl,BYTE PTR [varW+2]          ; c.
mov ax,WORD PTR [varD+2]          ; d.
mov eax,DWORD PTR varW            ; e.
```

Sua vez . . .

Escrever o valor de cada operando destino:

```
.data
varB BYTE 65h,31h,02h,05h
varW WORD 6543h,1202h
varD DWORD 12345678h

.code
mov ax,WORD PTR [varB+2]          ; a. 0502h
mov bl,BYTE PTR varD              ; b. 78h
mov bl,BYTE PTR [varW+2]          ; c. 02h
mov ax,WORD PTR [varD+2]          ; d. 1234h
mov eax,DWORD PTR varW            ; e. 12026543h
```

Operador TYPE

O operador TYPE retorna o tamanho em bytes de um único elemento de uma declaração de dados.

```
.data
var1 BYTE ?
var2 WORD ?
var3 DWORD ?
var4 QWORD ?

.code
mov eax,TYPE var1          ; 1
mov eax,TYPE var2          ; 2
mov eax,TYPE var3          ; 4
mov eax,TYPE var4          ; 8
```

Operador LENGTHOF

O operador LENGTHOF conta o número de elementos numa declaração única de dado.

```
.data                                          LENGTHOF
byte1  BYTE 10,20,30                           ; 3
array1 WORD 30 DUP(?),0,0                      ; 32
array2 WORD 5 DUP(3 DUP(?))                    ; 15
array3 DWORD 1,2,3,4                           ; 4
digitStr BYTE "12345678",0                     ; 9

.code
mov ecx,LENGTHOF array1                        ; 32
```

Operador SIZEOF

O operador SIZEOF retorna um valor que é equivalente a multiplicar LENGTHOF por TYPE.

```
.data                                          SIZEOF
byte1  BYTE 10,20,30                           ; 3
array1 WORD 30 DUP(?),0,0                      ; 64
array2 WORD 5 DUP(3 DUP(?))                    ; 30
array3 DWORD 1,2,3,4                           ; 16
digitStr BYTE "12345678",0                     ; 9

.code
mov ecx,SIZEOF array1                          ; 64
```

Usando Múltiplas Linhas (1 de 2)

Uma declaração de dados usa múltiplas linhas se cada linha (exceto a última) termina com vírgula. Os operadores LENGTHOF e SIZEOF incluem todas as linhas pertencentes à declaração:

```
.data
array WORD 10,20,
    30,40,
    50,60

.code
mov eax,LENGTHOF array              ; 6
mov ebx,SIZEOF array                ; 12
```

Usando Múltiplas Linhas (2 de 2)

No exemplo seguinte, array identifica somente a primeira declaração de WORD. Comparar os valores que retornam pelo LENGTHOF e SIZEOF com o slide anterior:

```
.data
array  WORD 10,20
       WORD 30,40
       WORD 50,60

.code
mov eax,LENGTHOF array          ; 2
mov ebx,SIZEOF array            ; 4
```

Diretiva LABEL

- Atribui um nome e tipo de label alternativo a uma posição existente
- LABEL não aloca nenhum dado a si próprio
- Remove a necessidade do operador PTR

```
.data
dwList   LABEL DWORD
wordList LABEL WORD
intList  BYTE 00h,10h,00h,20h
.code
mov eax,dwList              ; 20001000h
mov cx,wordList             ; 1000h
mov dl,intList              ; 00h
```

Próxima seção

- Endereçamento Indireto

Endereçamento Indireto

- Operandos Indiretos
- Exemplo de soma de vetor
- Operandos Indexados
- Ponteiros

Operandos indiretos (1 de 2)

Um operando indireto refere-se ao endereço, geralmente, de um vetor ou cadeia. Funciona com um ponteiro.

```
.data
val1 BYTE 10h,20h,30h
.code
mov esi,OFFSET val1
mov al,[esi]                ; dereference ESI  (AL = 10h)

inc esi
mov al,[esi]                ; AL = 20h

inc esi
mov al,[esi]                ; AL = 30h
```

Operandos indiretos (2 de 2)

Usar PTR para determinar o tamanho de um operando de memória.

```
.data
myCount WORD 0

.code
mov esi,OFFSET myCount
inc [esi]                    ; error: ambiguous
inc WORD PTR [esi]           ; ok
```

PTR seria usado aqui?

```
add [esi],20
```

Operandos indiretos (2 de 2)

Usar PTR para determinar o tamanho de um operando de memória.

```
.data
myCount WORD 0

.code
mov esi,OFFSET myCount
inc [esi]                    ; error: ambiguous
inc WORD PTR [esi]           ; ok
```

PTR seria usado aqui?

```
add [esi],20
```

Sim, porque [esi] pode apontar para um byte, word, ou doubleword

Exemplo de soma de vetor

Operandos indiretos são ideais para vetores. Note que o registrador entre colchetes deve ser incrementado por um valor que coincide com o tipo do vetor.

```
.data
arrayW WORD 1000h,2000h,3000h
.code
   mov esi,OFFSET arrayW
   mov ax,[esi]
   add esi,2                  ; or: add esi,TYPE arrayW
   add ax,[esi]
   add esi,2
   add ax,[esi]               ; AX = sum of the array
```

Modificar este exemplo para um vetor de doublewords.

Operandos indexados

Um operando indexado soma uma constante a um registrador para gerar um endereço efetivo. Existem duas formas de notação:

[label + reg] ***label[reg]***

```
.data
arrayW WORD 1000h,2000h,3000h
.code
    mov esi,0
    mov ax,[arrayW + esi]          ; AX = 1000h
    mov ax,arrayW[esi]             ; alternate format
    add esi,2
    add ax,[arrayW + esi]
    etc.
```

Modificar o exemplo para um vetor de doublewords.

Índice com escala

Pode-se usar um fator de escala (tipo do vetor) ao endereço de um operando indireto ou indexado.

```
.data
arrayB BYTE  0,1,2,3,4,5
arrayW WORD  0,1,2,3,4,5
arrayD DWORD 0,1,2,3,4,5

.code
mov esi,4
mov al,arrayB[esi*TYPE arrayB]    ; 04
mov bx,arrayW[esi*TYPE arrayW]    ; 0004
mov edx,arrayD[esi*TYPE arrayD]   ; 00000004
```

Ponteiros

Pode-se declarar uma variável ponteiro que contem o offset de uma outra variável.

```
.data
arrayW WORD 1000h,2000h,3000h
ptrW DWORD arrayW
.code
    mov esi,ptrW
    mov ax,[esi]              ; AX = 1000h
```

Formato alternativo:

```
ptrW DWORD OFFSET arrayW
```

Próxima seção

- Instruções JMP e LOOP

Instruções JMP e LOOP

- Instrução JMP
- Instrução LOOP
- Exemplo de LOOP
- Somando um vetor de inteiros
- Copiando uma cadeia (String)

Instrução JMP

- JMP é um saldo incondicional a um label que é usualmente dentro de um mesmo procedimento.
- Sintaxe: JMP *target*
- Lógica: EIP ← *target*
- Exemplo:

```
top:
    .
    .
    jmp top
```

Um jump para fora do procedimento deve ser um tipo especial de label chamado de label global.

Instrução LOOP

- A instrução LOOP cria um loop com contador
- Sintaxe: LOOP *target*
- Lógica:
  - ECX ← ECX – 1
  - se ECX != 0, salta para *target*
- Implementação:
  - O assembler calcula a distância, em bytes, entre o offset da instrução seguinte e o offset do target, chamada offset relativo.
  - O offset relativo é somado ao EIP.

Exemplo de LOOP

O loop seguinte calcula a soma dos inteiros 5 + 4 + 3 +2 + 1:

```
offset      machine code          source code
00000000    66 B8 0000            mov   ax,0
00000004    B9 00000005           mov   ecx,5

00000009    66 03 C1          L1: add   ax,cx
0000000C    E2 FB                 loop  L1
0000000E
```

Quando LOOP é executado, a posição corrente = 0000000E (offset da instrução seguinte) é somado com –5 (FBh) causando um salto para a posição 00000009:

00000009 ← 0000000E + FB

Sua vez . . .

Se o offset relativo é codificado num byte com sinal,

(a) qual é o maior jump retroativo possível?

(b) qual é o maior jump progressivo possível?

(a)
(b)

Sua vez . . .

Se o offset relativo é codificado num byte com sinal,

(a) qual é o maior jump retroativo possível?

(b) qual é o maior jump progressivo possível?

(a) –128

(b) +127

Sua vez . . .

Qual o valor final de AX?

```
    mov ax,6
    mov ecx,4
L1:
    inc ax
    loop L1
```

Quantas vezes o loop executa?

```
    mov ecx,0
X2:
    inc ax
    loop X2
```

Sua vez . . .

Qual o valor final de AX?

10

```
    mov ax,6
    mov ecx,4
L1:
    inc ax
    loop L1
```

Quantas vezes o loop executa?

4,294,967,296 → 2^{32}

```
    mov ecx,0
X2:
    inc ax
    loop X2
```

Loop aninhado

Se necessitamos codificar um loop dentro de um loop, deve-se salvar o valor do contador do loop externo ECX.

No exemplo seguinte, o loop externo executa 100 vezes e o loop interno 20 vezes.

```
.data
count DWORD ?
.code
   mov ecx,100          ; set outer loop count
L1:
   mov count,ecx        ; save outer loop count
   mov ecx,20           ; set inner loop count
L2: .
   .
   loop L2              ; repeat the inner loop
   mov ecx,count        ; restore outer loop count
   loop L1              ; repeat the outer loop
```

Somando um vetor de inteiros

O seguinte código calcula a soma de um vetor de inteiros de 16 bits

```
.data
intarray WORD 100h,200h,300h,400h
.code
    mov edi,OFFSET intarray        ; address of intarray
    mov ecx,LENGTHOF intarray      ; loop counter
    mov ax,0                       ; zero the accumulator
L1:
    add ax,[edi]                   ; add an integer
    add edi,TYPE intarray          ; point to next integer
    loop L1                        ; repeat until ECX = 0
```

Sua vez . . .

Que mudança voce faria ao programa anterior se fosse somar um vetor de doublewords?

Copiando uma cadeia

O seguinte código copia uma cadeia de fonte para destino:

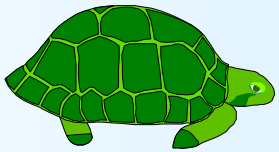
```
.data
source  BYTE  "This is the source string",0
target  BYTE  SIZEOF source DUP(0)

.code
    mov  esi,0                    ; index register
    mov  ecx,SIZEOF source        ; loop counter
L1:
    mov  al,source[esi]           ; get char from source
    mov  target[esi],al           ; store it in the target
    inc  esi                      ; move to next character
    loop L1                       ; repeat for entire string
```

good use of SIZEOF

Sua vez . . .

Reescrever o programa anterior usando endereçamento indireto ao invés de endereçamento indexado.



The End