Tratamento de sequências

João Canas Ferreira

Abril 2014

U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Instruções para tratamento de sequências

- IA-32 inclui *instruções dedicadas para tratamento de sequências* de bytes, words ou doublewords em memória:
  - copiar entre posições de memória — `movsb`, `movsw`, `movsd`
  - comparar duas posições de memória — `cmpsb`, `cmpsw`, `cmpsd`
  - comparar AL/AX/EAX com memória — `scasb`, `scasw`, `scasd`
  - guardar AL/AX/EAX em memória — `stosb`, `stosw`, `stosd`
  - carregar AL/AX/EAX de memória — `lodsb`, `lodsw`, `lodsd`
- Memória indexada por registos: fonte ESI, destino EDI
- Registos de índice ajustados automaticamente segundo:
  - tamanho dos dados: bytes ± 1, words ± 2, doublewords ± 4
  - valor do indicador de direção D:
    0 - incremento
    1 - decremento
- Instrução STD coloca D=1
- Instrução CLD coloca D=0
- *stdcall:* indicador D=0 à entrada e à saída de sub-rotinas

Instruções MOVSx

- As instruções MOVSB, MOVSW e MOVSD copiam o conteúdo da posição de memória apontada por ESI para a posição de memória apontada por EDI

```
        .data
origem  DWORD  20 dup(0FFFFFFFFh)
destino DWORD  20 dup(?)
        .code
        mov    ecx,LENGTHOF origem
        mov    esi,OFFSET origem
        mov    edi,OFFSET destino
; ciclo de cópia de origem para destino
@@:     movsd
        loop   @B
        ...
```


Prefixo de repetição REP

- Prefixo REP antes de uma instrução provoca a repetição da instrução ECX vezes
- REP deve ser usado com as instruções de tratamento de sequências
- REP instrução é equivalente a:

```
enquanto (ECX != 0) {
     executar instrução
     (inclui ajustar ESI e/ou EDI)
     ECX = ECX-1 (sem alterar indicadores)
}
```

```
        .data
origem  DWORD  20 dup(0FFFFFFFFh)
destino DWORD  20 dup(?)
        .code
        mov    ecx, LENGTHOF origem
        mov    esi, OFFSET origem
        mov    edi, OFFSET destino
        rep movsd
```

Instruções STOSx

- As instruções STOSB, STOSW e STOSD copiam AL/AX/EAX para a posição de memória apontada por EDI

```
            .data
NElem =     100
vec         WORD    NElem dup(?)
            .code
            cld
            mov     ax, 0FFFFh
            mov     edi, OFFSET vec
            mov     ecx, NElem
            ; preencher vec com 0FFFFh
            rep stosw
```


Instruções LODSx

- As instruções LODSB, LODSW e LODSD copiam a posição de memória apontada por ESI para AL/AX/EAX

```
            .data ; Exemplo:  multiplicar vec por factor
vec         DWORD  1, 2, 3, 4, 5
factor      DWORD  12
            .code
            mov    esi, OFFSET vec
            mov    edi, esi
            mov    ecx, LENGTHOF vec
            cld
@@:         lodsd                       ; ESI = ESI +4
            mul    factor
            ; ignorar EDX
            stosd                       ; EDI = EDI +4
            loop   @B
```

Instruções CMPSx

- CMPSB, CMPSW e CMPSD comparam o conteúdo da memória apontada por ESI com o da memória apontada por EDI
- Estas instruções afetam os indicadores *(flags)* como a instrução CMP

```
        .data
var1    BYTE   1
var2    BYTE   12
        .code
        mov    esi, OFFSET var1
        mov    edi, OFFSET var2
        cmpsb                      ; ESI=ESI+1 e EDI =EDI+1
        je     iguais
        ; Valores são diferentes ...
        jmp    seg
iguais: ...; Valores são iguais
seg:    ...
```


Prefixos de repetição com teste adicional

- Prefixo REPE: a repetição da instrução ECX vezes enquanto ZF=1 .
- `REPE instrução` é equivalente a:

```
enquanto (ECX != 0) {
        executar instrução
        (inclui ajustar EDI e/ou ESI)
        ECX = ECX-1 (sem alterar indicadores)
        se ZF=0 terminar ciclo
}
```

- `REPNE instrução` é equivalente a:

```
enquanto (ECX != 0) {
        executar instrução
        (inclui ajustar EDI e/ou ESI)
        ECX = ECX-1 (sem alterar indicadores)
        se ZF=1 terminar ciclo
}
```

- REPE/REPNE usados apenas com instruções CMPSx ou SCASx
- Sinónimos: REPE = REPZ e REPNE = REPNZ

Exemplo: ordem alfabética de 2 cadeias de carateres

```
.data
str1   BYTE "contar",0
str2   BYTE "contratar",0
.code
cld
mov    esi, OFFSET str1
mov    edi, OFFSET str2
mov    ecx, LENGTHOF str1
repe cmpsb
jb     str1_antes
;; STR1 não vem antes de STR2
jmp    seg
str1_antes:
;; STR1 vem antes de STR2
seg:  ...
```

- Varrimento pode terminar:
  1. ZF=0 e ECX≠0
     ⇒ cadeias são diferentes
  2. ZF=1 e ECX=0
     ⇒ cadeias são iguais
  3. ZF=0 e ECX=0
     ⇒ cadeias são diferentes
- Resultado da última comparação determina a ordem:
  - Se ECX≠0 e [ESI]<[EDI]
    str1 < str2
  - Se ECX=0, então tem-se sempre [ESI]=0 e [ESI]<[EDI]
    Logo: str1 < str2
  - Basta um teste! [ESI]<[EDI]
    usando o salto condicional jb

Instruções SCASx

- As instruções SCASB, SCASW e SCASD comparam AL/AX/EAX com o conteúdo da posição de memória apontada por EDI
- Estas instruções afetam os indicadores *(flags)* como a instrução CMP
- Também podem ser usadas com REPE/REPNE

```
        .data
letras  BYTE   "ABCDEFGHIJ",0
        .code
        mov    edi, OFFSET letras
        mov    al, 'F'
        mov    ecx, LENGTHOF letras
        repne scasb
        jnz    seg                ; Para quê?
        dec    edi                ; Para quê?
        ; EDI aponta para F em letras
seg:    ...
```

Endereçamento com dois registos

- Em muitas operações de tratamento de sequências é útil usar dois registos para endereçar a memória:
  1. o registo de *base* define o início da sequência
  2. o registo de *índice* define a distância (em bytes) a partir da base
- O registo ESP não pode ser usado neste tipo de endereçamento
- O endereçamento pode usar ainda uma constante adicional

```
.data ; exemplo sem funcionalidade útil
string1 BYTE 100 dup('A')
.code
mov esi, OFFSET string1
mov edx, 0
mov al, [esi+edx]; posição inicial porque edx=0
inc edx
mov al, [esi+edx] ; 2º byte de string1
mov bl, [esi+edx+2] ; 2 posições à frente da anterior
mov bl, string1[edx] ; equivale a [edx+string1]
```


Endereçamento indexado com escala

- Quando os vetores têm elementos de 2, 4 ou 8 bytes é útil multiplicar o índice pelo tamanho do elemento.
- O fator de escala pode ser apenas 1,2,4 ou 8.
- Também se pode usar uma constante no endereçamento

```
.data ; exemplo sem funcionalidade útil
vect    WORD 20 dup(?)
.code
mov edi, OFFSET vect
...
mov ax, [edi+2*ebx]
mov dx, [edi+2*ebx+10]
mov cx, [vect+2*ebx]
mov cx, vect[2*ebx] ; o mesmo que o anterior
```

Referências

- Mais informação sobre estes tópicos pode ser encontrada no capítulo 9 de:

Irvine03 K. R. Irvine, *Assembly Language for Intel-Based Computers*, 4ª edição, Prentice Hall, 2003.