```
        LEA     SI,SEND_STR         ;  sending address (DS:SI)
        REP     MOVSB               ;Copy SEND_STR to RECV_STR
```

MOV, LODS, and STOS always fully repeat the specified number of times. However, CMPS and SCAS make comparisons that set status flags so that the operations can end immediately on finding a specified condition. The variations of REP that CMPS and SCAS use for this purpose are the following:

- REP: Repeat the operation until CX is decremented to zero.
- REPE or REPZ: Repeat the operation while the Zero Flag (ZF) indicates equal/zero. Stop when ZF indicates not equal/zero or when CX is decremented to zero.
- REPNE or REPNZ: Repeat the operation while ZF indicates not equal/zero. Stop when ZF indicates equal or zero or when CX is decremented to zero.

The following sections examine each string operation in detail.

MOVS: MOVE STRING INSTRUCTION

MOVSB, MOVSW, and MOVSD combined with a REP prefix and a length in CX can move a specified number of characters. The segment:offset registers are ES:DI for the receiving string and DS:SI for the sending string. As a result, at the start of an .EXE program, be sure to initialize ES along with DS, and prior to executing the MOVS, also initialize DI and SI. Depending on the Direction Flag, MOVS increments or decrements DI and SI by 1 for byte, 2 for word, and 4 for doubleword. The following example illustrates moving 12 words:

```
MOV     CX,12               ;Number of words
LEA     DI,RECV_STR         ;Address of RECV_STR (ES:DI)
LEA     SI,SEND_STR         ;Address of SEND_STR (DS:SI)
REP   MOVSW                 ;Move 12 words
```

The instructions equivalent to the REP MOVSW operation are:

```
        JCXZ    L40         ;Bypass if CX initially zero
L30:    MOV     AX, [SI]    ;Get word from SEND_STR
        MOV     [DI],AX     ;Store word in RECV_STR
        ADD     DI,2        ;Increment for next word
        ADD     SI,2        ;
        LOOP    L30         ;Decrement CX and repeat
L40:    ...
```

Earlier, Figure 6-2 illustrated moving a 9-byte field. The program could also have used MOVSB for this purpose. In the partial program in Figure 8-2, ES is initialized because it is required by the MOVS instructions. The program uses MOVSB to move a 12-byte field, STRING1, one byte at a time to STRING2. The first instruction, CLD, clears the Direction Flag so that the MOVSB processes data from left to right. The Direction Flag is normally 0 at the start of execution, but CLD is coded here as a precaution.

A MOV instruction initializes CX with 12 (the length of STRING1 and of STRING2). Two LEA instructions load SI and DI with the offset addresses of STRING1 and STRING2, respectively. REP MOVSB now performs the following:

- Moves the leftmost byte of STRING1 (addressed by DS.SI) to the leftmost byte of STRING2 (addressed by ES.DI);

```
STRING1 DB     'Interstellar'      ;Data items
STRING2 DB     12 DUP(' ')
STRING3 DB     12 DUP(' ')
        ...
        MOV    AX,@data            ;Initialize
        MOV    DS,AX               ;  segment
        MOV    ES,AX               ;  registers
                                   ;Use of MOVSB:
        CLD                        ;Left to right
        MOV    CX,12               ;Move 12 bytes,
        LEA    DI,STRING2          ;  STRING1 to STRING2
        LEA    SI,STRING1
        REP MOVSB
                                   ;Use Of MOVSW:
        CLD                        ;Left to right
        MOV    CX,06               ;Move 6 words,
        LEA    DI,STRING3          ;. STRING2 to STRING3
        LEA    SI,STRING2
        REP MOVSW
        ...
```

Figure 8-2 Using MOVS String Operations

- Increments DI and SI by 1 for the next bytes to the right;
- Decrements CX by 1;
- Repeats this operation 12 times until CX becomes 0.

Because the Direction Flag is 0 and MOVSB increments DI and SI, each iteration processes one byte farther to the right, as STRING1+1 to STRING2+1, and so on. At the end of execution, CX contains 0, DI contains the address of STRING2+12, and SI contains the address of STRING1+12—both one byte past the end of the name.

To process from *right to left,* set the Direction Flag to 1. MOVSB then decrements DI and SI, but to move the contents correctly, you have to initialize SI with STRINGl+11 and DI with STRING2+11.

The program next uses MOVSW to move six words from STRING2 to STRING3. At the end of execution, CX contains 0, DI contains the address of STRING3+12, and SI contains the address of STRING2+12.

Because MOVSW increments DI and SI by 2, the operation requires only six loops. For processing right to left, set the Direction Flag and initialize SI with STRING1+10 and DI with STRING2+10.

LODS: LOAD STRING INSTRUCTION

LODS simply loads AL with a byte, AX with a word, EAX with a doubleword from memory. The memory address is subject to DS:SI registers, although you can override SI. Depending on the Direction Flag, the operation also increments or decrements SI by 1 for byte, 2 for word, and 4 for doubleword.

Because one LODS operation fills the register, there is no practical reason to use the REP prefix with it. For most purposes, a simple MOV instruction is adequate. But MOV generates three bytes of machine code, whereas LODS generates only one, although you have to initialize SI. You could use LODS to step through a string one byte, word, or doubleword at a time, examining successively for a particular character.

The instructions equivalent to LODSB are

```
MOV AL, [SI]     ;Transfer byte to AL
INC SI           ;Increment SI for next byte
```

The following example defines a 12-byte field named STRING1 containing the value "Interstellar" and another 12-byte field named STRING2. The objective is to transfer the bytes from STRING1 to STRING2 in

reverse sequence, so that STRING2 contains "rallet-sretnI." LODSB accesses one byte at a time from STRING1 into AL, and MOV [DI],AL transfers the bytes to STRING2, from right to left.

```
STRING1   DB      'Interstellar'   ;Data items
STRING2   DB      12 DUP(20H)
          ...
          CLD
          MOV     CX,12           ;Left to right
          LEA     SI,STRING1
          LEA     DI,STRING2+11   ;Address of STRING1 (DI:SI)
L20:      LODS                    ;Address of STRING2+11 (ES:DI)
          MOV     [DI],AL         ;Get character in AL,
          DEC     DI              ;  store in STRING2,
          LOOP    L20             ;  right to left
          ...                     ;12 characters?
```

STOS: STORE STRING INSTRUCTION

STOS stores the contents of AL, AX, or EAX into a byte, word, or doubleword in memory. The memory address is always subject to ES.DI. Depending on the Direction Flag, STOS also increments or decrements DI by 1 for byte, 2 for word, and 4 for doubleword.

A practical use of STOS with a REP prefix is to initialize a data area to any specified value, such as clearing an area to blanks. You set the number of bytes, words or doublewords in CX. The instructions equivalent to REP STOSB are:

```
          JCXZ L30          ;Jump if CX zero
L20:      MOV [DI],AL       ;Store AL in memory
          INC/DEC DI        ;Increment or decrement (sets flags)
          LOOP L20          ;Decrement CX and repeat
L30:      ...               ;Operation complete
```

The STOSW instruction in the following example repeatedly stores a word containing 2020H (blanks) six times through STRING1. The operation stores AL in the first byte and AH in the next byte (that is, reversed). At the end, all of STRING1 is blank, CX contains 00, and DI contains the address of STRING1+12.

```
CLD                     ;Left to right
MOV     AX,2020H        ;Move
MOV     CX,06           ; 6 blank words
LEA     DI,STRING1      ; to STRING1 (ES:DI)
REP STOSW
```

PROGRAM: USING LODS AND STOS TO EDIT DATA

The program in Figure 8-3 illustrates the use of both the LODS and STOS instructions. Refer chapter 12 for details on video and keyboard processing. Its purpose is to allow a user to edit a string of characters. To reduce the space required and the complexity, this is a bare-bones editor. Basically, the program displays a string of 30 characters. The more relevant procedures perform the following:

- A10MAIN initializes addressability, calls Q30DISPLY to display the string, and calls B10KEYBRD to request a keyboard character. The program ends when the user presses <Esc>.

```
TITLE     A11EDIT (EXE)  Editing Features
          .MODEL SMALL
          .STACK 64
          .DATA
INDENT    EQU    24                 ;Screen indent
LEFTLIM   EQU    00                 ;Left limit of data
RIGHTLIM  EQU    29                 ;Right limit of data
NOCHARS   EQU    30                 ;Length of data
COL       DB     00                 ;Screen column
ROW       DB     10                 ;Screen row
DATASTR   DB     'abcdefghijklmno' ;Area for editing data
          DB     'pqrstuvwxyzABCD', 2OH
.386    ; ------------------------------------------------
          .CODE
A10MAIN   PROC   FAR
          MOV    AX,@data           ;Initialize segment
          MOV    DS,AX              ;  registers
          MOV    ES,AX
          CALL   Q10CLEAR           ;Clear screen
          CALL   Q20CURSOR          ;Set cursor start
          CALL   Q30DISPLY          ;Display string
A30:
          CALL   Q20CURSOR          ;Reset cursor start
          CALL   B10KEYBRD          ;Get KB character
          CMP    AH,01H             ;Escape key?
          JNE    A30                ;  no, continue
          MOV    AX,0600H           ;  yes, quit
          CALL   Q10CLEAR           ;Clear screen
          MOV    AX,4C00H           ;End of processing
          INT    21H
A10MAIN   ENDP

;         Get keyboard character and determine action to take:
;         ----------------------------------------------------
B10KEYBRD PROC   NEAR                ;Uses AX only
          MOV    AH,10H             ;Get
          INT    16H                ;  character
          CMP    AL,00H             ;Function/direction key?
          JE     B20                ;  yes
          CMP    AL,0E0H            ;Function/direction key?
          JE     B20                ;  yes
          CALL   H10CHARS           ;Other character
          JMP    B90                ;Exit
B20:      CMP    AH,4DH             ;Right arrow?
          JNE    B30                ;  no
          CALL   C10RTARRW          ;  yes, process
          JMP    B90
B30:      CMP    AH,4BH             ;Left arrow?
          JNE    B40                ;  no
          CALL   D10LFARRW          ;  yes, process
          JMP    B90
B40:      CMP    AH,53H             ;Delete key?
          JNE    B50                ;  no
          CALL   E10DELETE          ;  yes, process
          JMP    B90
B50:      CMP    AH,47H             ;Home key?
          JNE    B60                ;  no
          CALL   F10HOME            ;  yes, process
          JMP    B90
B60:      CMP    AH,4FH             ;End key?
          JNE    B90                ;  no
          CALL   G10END             ;  yes, process
B90:      RET
B10KEYBRD ENDP

;         Right arrow.  If at right edge, set cursor
;         to left edge, else increment column:
;         ------------------------------------------
C10RTARRW PROC   NEAR
          CMP    COL,RIGHTLIM       ;At rightmost edge?
          JAE    C20                ;  yes,
```

Figure 8-3 Simple Editing Instructions

```
            INC     COL                     ;  no, increment col
            JMP     C90                     ;  exit
C20:        CALL    F10HOME                 ;  cursor to left edge
C90:        RET                             ;
C10RTARRW   ENDP

;           Left arrow.  If at left edge, set cursor
;           to right edge, else decrement column:
;           -------------------------------------
D10LFARRW   PROC    NEAR
            CMP     COL,LEFTLIM             ;At leftmost edge?
            JBE     D20                     ;  yes,
            DEC     COL                     ;  no, decrement col
            JMP     D90                     ;  exit
D20:        CALL    G10END                  ;  cursor to right edge
D90:        RET
D10LFARRW   ENDP

;           Delete key.  Replace current character with one
;           to right, shift rightmost characters to left:
;           ---------------------------------------------
E10DELETE   PROC    NEAR                    ;Uses BX, DI, SI
            MOVZX   BX,COL                  ;Get column
            PUSH    BX                      ;Save for later
            LEA     DI,[DATASTR+BX]         ;Init. present col
            LEA     SI,[DATASTR+BX+1]       ;  and adjacent col
E20:
            LODSB                           ;Store adjacent char
            STOSB                           ;  in present col
            CALL    Q40DISCHR               ;Display the char
            INC     COL                     ;Increment next col
            CALL    Q20CURSOR               ;Set cursor
            CMP     COL,RIGHTLIM            ;At right edge?
            JBE     E20                     ;  no, repeat
            POP     BX                      ;Get saved original
            MOV     COL,BL                  ;  column
            RET
E10DELETE   ENDP
;           Home key.  Set cursor to left column:
;           -------------------------------------
F10HOME     PROC    NEAR
            MOV     COL,LEFTLIM             ;Set cursor
            CALL    Q20CURSOR               ;  at left edge
            RET
F10HOME     ENDP

;           End key.  Set cursor to right column:
;           -------------------------------------
G10END      PROC    NEAR
            MOV     COL,RIGHTLIM            ;Set cursor
            CALL    Q20CURSOR               ;  at right edge
            RET
G10END      ENDP

;           All other characters.  Bypass characters below
;           20H and above 7EH, else insert at cursor:
;           ---------------------------------------------
H10CHARS    PROC    NEAR                    ;Uses BX, DI
            CMP     AL,20H                  ;ASCII char below 20H?
            JB      H90                     ;  yes, bypass
            CMP     AL,7EH                  ;Above 7EH?
            JA      H90                     ;  yes, bypass
            MOVZX   BX,COL                  ;Use COL as index
            LEA     DI,DATASTR              ;Move character to
            MOV     [DI+BX],AL              ;  data string
            CALL    Q40DISCHR               ;Display the character
            CMP     COL,RIGHTLIM            ;At right edge?
            JAE     H90                     ;  yes, exit
            INC     COL                     ;  no, increment column
H90:        RET
```

Figure 8-3 *Continued*