บทที่ 11

การประมวลผลข้อมูลสตริง

STRING PROCESSING

วัตถุประสงค์

หลังจากที่ท่านศึกษาบทนี้ท่านจะมีความเข้าใจดังต่อไปนี้

- การใช้คำสั่งเกี่ยวกับการจัดการสตริง
- คำสั่ง MOVS , LODS , STOS , CMPS , SCAS

CT 215 279

บทที่ 11
การประมวลผลข้อมูลสตริง
STRING PROCESSING

บทนำ

ในบทนี้เป็นการทำงานของคำสั่งสตริง เป็นการทำงานที่เกี่ยวกับข้อมูลขนาดไบท์ เวิร์ดหรือ 2 เวิร์ดในหนึ่งเวลา การใช้งานจะใช้ในการเคลื่อนย้ายข้อมูล หรือเปรียบเทียบข้อมูลตามยาวของฟิลด์ ตัวอย่าง ถ้าท่านต้องการเปรียบเทียบรายการหรือชื่อในใบสั่ง เพื่อเรียงตามลำดับ รายการเหล่านี้จะอยู่ในรูปแบบของสตริง ที่เป็นตัวอักขระหรือตัวเลข สำหรับรายการประมวลผลข้อมูลที่เป็นสตริง ภาษาแอสเซมบลีจะมีคำสั่งเกี่ยวกับสตริง 5 คำสั่ง ดังนี้

MOVS เป็นการเคลื่อนย้ายข้อมูลขนาดไบท์ เวิร์ดหรือ 2 เวิร์ดจากตำแหน่งหนึ่งไปอีกตำแหน่งหนึ่งในหน่วยความจำ

LODS เป็นการโหลดข้อมูลจากหน่วยความจำในรูปของไบท์ ไปที่ AL เวิร์ดไปที่ AX และ 2เวิร์ดไปที่ EAX

STOS เป็นการเก็บข้อมูลของ AL.AX or EAX ไปไว้ในหน่วยความจำ

CMPS เป็นการเปรียบเทียบข้อมูลขนิดไบท์ เวิร์ดหรือ 2 เวิร์ดตามตำแหน่งในหน่วยความจำ

SCAS เป็นการเปรียบเทียบข้อมูลของ AL.AX or EAX กับข้อมูลในหน่วยความจำกลุ่มคำสั่งเหล่านี้จะมีคำนำหน้า (prefix) คือ REP เพื่อให้คำสั่งเหล่านี้ทำงานตามรอบที่กำหนด

การทำงานของคำสั่งสตริง

คำสั่งสตริงเป็นคำสั่งที่กำหนดให้ทำงานซ้ำๆกัน สำหรับขนาดไบท์ เวิร์ดหรือ 2 เวิร์ด (80386/80486) ในหนึ่งเวลา รีจิสเตอร์ที่จะใช้กำหนดแอดเดรสข้อมูล สำหรับการทำงาน

ของคำสั่งสตริงแต่ละคำสั่ง และในส่วนของไบท์ เวิร์ดและ 2 เวิร์ด โดยใช้รีจิสเตอร์ DI and SI กำหนดแอดเดรส

INSTRUCTION	IMPLIED OPERANDS	BYTE	WORD	DOUBLEWORD
MOVS	DI,SI	MOVSB	MOVSW	MOVSD
LODS	AL,SI or AX,SI	LODSB	LODSW	LODSD
STOS	DI,AL or DI,AX	STOSB	STOSW	STOSD
CMPS	SI,DI	CMPSB	CMPSW	SMPSD
SCAS	DI,AL or DI,AX	SCASB	SCASW	SCASD

รหัสคำสั่งที่กำหนดของคำสั่ง MOVS จะต้องมีโอเปอรานด์ สำหรับแสดงการกระทำ เช่น MOVS BYTE 1,BYTE2 โอเปอรานด์จะแสดงความยาวของการเคลื่อนย้าย การปฏิบัติจะทำการโหลดแอดเดรสของโอเปอรานด์ใน DI and SI และรหัสคำสั่งใช้ MOVSB,MOVSW and MOVSD คำสั่งเหล่านี้ไม่ต้องมีโอเปอรานด์

คำสั่งสตริงจะสมมติข้อมูลของ DI and SI เป็นค่า offset ที่อ้างอิงในหน่วยความจำ รีจิสเตอร์ SI ปกติจะใช้กับ DS(data segment) เช่น DS:SI รีจิสเตอร์ DI ใช้กับ ES (extra segment) เช่น ES:DI คำสั่ง MOVS,STOS,CMPS and SCAS ใช้ใน EXE โปรแกรมที่เซ็ทค่าเริ่มแรกใน ES ปกติใช้กับ DS (ใช้คำสั่ง LEA ในการโหลดแอดเดรส)

รูปแบบของคำสั่งสตริงที่ไม่ได้กำหนดขนาดของโอเปอร์รานด์ มีรูปแบบดังต่อไปนี้

```
MOVS    DEST,SOURCE     ;MOVE SOURCE TO DESTINATION
CMPS    DEST,SOURCE     ;COMPARE SOURCE TO DESTINATION
SCAS    DEST            ;SCAN DESTINATION STRING
STOS    DEST            ;STORE ACCUMALULATOR TO DESTINATION
LODS    SOURCE          ;LOAD ACCUMULATOR FROM SOURCE
```

ตัวอย่าง 11.1 Implied operands

```
MOV     SI,OFFSET SOURCE     ;POINT SI TO SOURCE
MOV     DI,OFFSET DEST       ;POINT DI TO DESTINATION
MOVSB                        ;MOVE FROM SOURCE TO DESTINATION
```

ตัวอย่าง 11.2 Explicit operands

```
MOV     SI,OFFSET SOURCE     ;POINT SI TO SOURCE
MOV     DI,OFFSET DEST       ;POINT DI TO DESTINATION
MOVSB   ES:DEST,SOURCE       ;MOVE FROM SOURCE TO DESTINATION
```

REP : REPEAT STRING PREFIX

```
REP           Repeat while CX >0
REPZ,REPE     Rpeat while the zero flag is set and CX >0
REPNZ,REPNE   Rpeat while the zero flag is clear and CX >0
```

คำนำหน้า REP ใช้กับคำสั่งสตริงเช่น REP MOVSB, เป็นการกำหนดให้คำสั่ง MOVSB ทำงานซ้ำๆกันตามข้อมูลที่อยู่ใน CX REP จะทำให้คำสั่งสตริง execute และ ลดค่าใน CX และทำงานจนกระทั่ง CX=0

แฟลกทิศทาง (direction flag , DF) เป็นตัวกำหนดทิศทางของการทำงานซ้ำๆ
- การทำงานปกติจะเลื่อนจากซ้ายไปขวา โดยใช้ CLD ไปเคลียร์แฟลก DF=0
- การเลื่อนจากขวาไปซ้าย ใช้ STD ในการเซ็ทค่า DF=1

ตัวอย่าง 11.3 เป็นการเคลื่อนย้ายข้อมูลขนาด 20 ไบท์ ของ STRING1 ไปที่ STRING2 สมมติค่า DS and ES ซึ่งเซกเมนต์ทั้งสองเก็บค่าแอดเดรสของ DS

```
STRING1  DB     20 DUP('*')
STRING2  DB     20 DUP(' ')
         ----
         CLD                ; CLEAR DF
         MOV    CX,20       ; SET CX=20
         LEA    DI,STRING2  ; DESTINATION NAME
         LEA    SI,STRING1  ; SENDING ADDRESS
REP      MOVSB              ; MOVE STRING1 TO STRING2
```

ตัวอย่าง 11.4 จงเขียนคำสั่งในการทำสำเนาข้อมูล STRING1 ไปยังสตริง 2 ในลักษณะตรงกันข้าม โดยให้ SI เป็นตัวชี้ตัวสุดท้ายของ STRING1 และ DI เป็นตัวชี้เริ่มต้นของ STRING2 และให้เคลื่อนย้ายตัวอักขระที่ SI กำหนดไปทางซ้ายของ STRING1

```
         LEA  SI,STRING1+4      ;SET PTS TO END OF STRING1
         LEA  DI,STRING2        ;DI PTS TO BEGINNING OF STR2
         STD                    ;RIGHT TO LEFT PROCESSING
         MOV  CX,5
MOVE:
         MOVSB                  ;MOVE A BYTE
         ADD  DI,2
         LOOP MOVE
```

ในช่วงของการ execute CMPS and SCAS จะต้องเซ็ทแฟลกสถานะ เพื่อการทำงานในการหาเงื่อนไขที่กำหนด ตัวแปร REP สำหรับใช้ในวัตถุประสงค์ต่อไปนี้

REP	การทำงานซ้ำๆจนค่า CX ลดลงเป็น 0
REPZ,REPE	การทำงานซ้ำๆจนค่าแฟลก ZF เป็นตัวชี้ว่า equal/zero แฟลก ZF จะเป็นตัวชี้ถึงการสิ้นสุดการทำงาน not equal/zero หรือ ค่า CX=0
REPNE or,REPNZ	การทำงานซ้ำๆขณะที่ ZF เป็นตัวชี้ว่า not equal/zero จะสิ้นสุดการทำงานเมื่อ ZF equal/zero or CX=0

MOVS : MOVE STRING

จากตัวอย่างบทที่ 7 รูป 7.4 แสดงการเคลื่อนย้ายข้อมูลขนาด 9 ไบท์ การทำงานจะมี 3 คำสั่ง สำหรับเซ็ทค่าเริ่มแรก และ 5 คำสั่งสำหรับการวนรอบ คำสั่ง MOVS พร้อมกับคำนำหน้า REP จะความยาวของการเคลื่อนย้ายใน CX จะใช้แทนมีรูปแบบคำสั่งดังนี้

```
REP     MOVSB   [ES:DI,DS:SI]
```

ในส่วนของการรับสตริง เซกเมนต์รีจิสเตอร์คือ ES และoffset register คือ DI สำหรับในการส่งสตริง เซกเมนต์รีจิสเตอร์คือ DS และ offset register คือ SI ผลลัพธ์ของการเริ่มต้น execute โปรแกรม การเซ็ทค่าของ ES and DS ก่อนการ execute คำสั่ง MOVS โดยใช้คำสั่ง LEA ในการเซ็ทค่า DI and SI ขึ้นอยู่กับทิศทางของ DF คำสั่ง MOVS จะเพิ่มขึ้นหรือลดลงในข้อมูลของ DI and SI ทีละ 1 สำหรับข้อมูลชนิดไบท์ ทีละ 2 สำหรับข้อมูลชนิดเวิร์ด ทีละ 4 สำหรับข้อมูลชนิด 2 เวิร์ด

คำสั่งเทียบเท่าคำสั่ง REP MOVSB คือ

```
        JCXZ    LABEL       ; JUMP IF CX ZERO
LABEL1: MOV     AL,[SI]     ; GET CHARACTER
        MOV     [DI],AL     ; STORE CHARACTER
        INC/DEC DI          ; INCREMENT/DECREMENT
        INC/DEC SI          ; INCREMENT/DECREMENT
        LOOP    LABEL1
LABEL2: ---
```

ในรูป 11-1 ในส่วนของ procedure C10MVSB ใช้คำสั่ง MOVSB ในการเคลื่อนย้ายข้อมูลขนาด 10 ไบท์ ในฟิลด์ NAME1 การเคลื่อนย้ายข้อมูลครั้งละ 1 ไบท์ ไปยังฟิลด์ NAME2 คำสั่งแรก คำสั่ง CLD ทำหน้าที่เคลียร์ DF=0 ดังนั้นคำสั่ง MOVSB จะย้ายข้อมูลจากซ้ายไปขวา แฟลกทิศทางปกติจะมีค่าเป็น 0 ที่จุดเริ่มต้นของการ execute แต่คำสั่ง CLD เป็นการป้องกันในการเปลี่ยนค่า

คำสั่ง LEA 2 คำสั่งเป็นการโหลดค่าให้กับ SI and DI ซึ่งเป็นค่า offset address ของ NAME1 and NAME2 การทำงานของ DOS จะทำการโหลดค่าลงใน DS and ES โดยอัตโนมัติ สำหรับโปรแกรม COM เซกเมนต์แอดเดรสและค่า offset address จะกำหนดด้วย DS:SI คำสั่ง MOV เซ็ทค่า CX=10

คำสั่ง REP MOVSB มีรูปแบบการทำงานดังต่อไปนี้

- เคลื่อนย้ายไบท์ซ้ายสุดของ NAME1 (แอดเดรสกำหนดโดยDS:SI) ไปที่บิตซ้ายสุดของ NAME2 (แอดเดรสกำหนดโดย ES:DI)
- เพิ่มค่าของ DI and SI ขึ้นอีก 1 สำหรับไบท์ต่อไปทางขวา
- ลดค่า CX ลง 1
- ทำงานซ้ำๆ 10 รอบ จนกระทั่ง Cx=0

เนื่องจากทิศทางของ DF=0 และ MOVSB จะเพิ่มค่าของ DI and SI มันจะเคลื่อนย้ายครั้งละ 1 ไบท์เพิ่มไปทางขวา ที่แอดเดรส NAME1+1 ไป NAME2+1 และย้ายต่อไปเรื่อยๆ จนจุดสุดท้ายของการ execute เมื่อ CX=00 ข้อมูลใน DI เป็นแอดเดรสของ NAME2+10 และข้อมูลใน SI เป็นข้อมูลของ NAME1+10

ถ้า DF=1 คำสั่ง MOVSB จะลดค่า SI and DI การทำงานจะเคลื่อนย้ายข้อมูลจากขวาไปซ้าย แต่ในกรณีนั้น ท่านจะต้องเซ็ทค่าเริ่มแรกของ SI ด้วยค่า NAME+9 และ DI ด้วยค่า NAME2+9

จาก procedure ในรูป 11-1 D10MVSW ใช้คำสั่ง MOVSW ในการเคลื่อนย้ายข้อมูล 5 เวิร์ด จาก NAME2 ไป NAME3 จนจุดสุดท้ายของการ execute เมื่อ CX=0 DI จะมีข้อมูลแอดเดรสของ NAME3+10 และ SI จะมีข้อมูลแอดเดรสของ NAME2+10

คำสั่ง MOVSW จะเพิ่มค่า DI and SI ทีละ 2 การทำงานต้องการเพียง 5 รอบ สำหรับการทำงานจากขวาไปซ้าย การทำงานเซ็ทค่าเริ่มแรกของ SI ด้วยค่า NAME1+8 และ DI ด้วยค่า NAME2+8

ตัวอย่าง 11.5 การใช้คำสั่ง MOVSW ในการแทรกค่า 30 ในช่วงระหว่าง 20 กับ 40 สมมุติค่า DS และ ES ในการกำหนดค่าใน DATA SEGMENT ซึ่งมีข้อมูลดังต่อไปนี้

```
ARR  DW   10,20,40,50,60,?
```

```
        STD                     ;RIGHT TO LEFT PROCESSING
        LEA  SI,ARR+8H          ;SI PTS TO 60
        LEA  DI,ARR+AH          ;DI PTS TO ?
        MOV  CX,3               ;3 ELTS TO MOVE
REP     MOVSW                   ;MOVE 40,50,60
        MOV  WORD PTR [DI],3;INSERT 30
```

ตัวอย่าง 11.6 การประยุกต์การใช้คำสั่ง MOVE ในการเคลื่อนย้ายข้อมูล

```
TITLE   STRING1  (COM)  USE OF MOVS STRING OPERATION
        .MODEL   SMALL
        .CODE
        ORG      100H
BEGIN:  JMP      SHORT  MAIN
; ------------------------------------------------
NAME1   DB       'Assemblers'        ; DATA ITEMS
NAME2   DB       10 DUP(' ')
NMAE3   DB       10 DUP(' ')
; ------------------------------------------------
MAIN    PROC     NEAR                ; MAIN PROCEDURE
        CALL     C10MVSB             ; MVSB SUBROUTINE
        CALL     D10MVSW             ; MVSW SUBROUTINE
        MOV      AH,4CH              ; EXIT
        INT      21H
MAIN    ENDP
;                USE OF MOVSB:
;                ------------
C10MVSB PROC     NEAR
        CLD                          ; LEFT TO RIGHT
        MOV      CX,10               ; MOVE 10 BYTES,
        LEA      DI,NAME1            ;  NAME1 TO NAME2
        LEA      SI,NAME2
```

```
        REP MOVSB
        RET
C10MVSB ENDP
;               USE OF MOVSW:
;               ------------
D10MVSW PROC    NEAR
        CLD                         ; LEFT TO RIGHT
        MOV     CX,05               ; MOVE 5 WORDS,
        LEA     DI,NAME3            ;  NAME2 TO NAME3
        LEA     SI,NAME2
        REP     MOVSW
        RET
D10MVSW ENDP
        END     BEGIN
```

LODS ; LOAD STRING

```
LODS    DEST
LODSB
LODSW
```

คำสั่ง LODS เป็นการโหลดข้อมูลจากหน่วยความจำขนาดไบท์ไว้ใน AL ข้อมูลเวิร์ดไว้ใน AX และ 2 เวิร์ดไว้ใน EAX แอดเดรสของหน่วยความจำถูกกำหนดโดยรีจิสเตอร์ SI(DS:SI) ถึงแม้ว่ารีจิสเตอร์ SI ขึ้นอยู่กับการทำงานของแฟลกทิศทาง(DF) โดยการเพิ่มหรือลดค่า SI ทีละ 1 สำหรับข้อมูลไบท์ ทีละ 2 สำหรับข้อมูลเวิร์ด และทีละ 4 สำหรับข้อมูล 2 เวิร์ด

การทำงานของคำสั่ง LODS แต่ละครั้งจะเก็บค่าไว้ในรีจิสเตอร์ ไม่เหมาะที่จะใช้กับคำสั่ง REP การทำงานส่วนมากจะใช้คำสั่ง MOV ก็เพียงพอ แต่คำสั่ง MOV จะใช้รหัสภาษาเครื่อง 3 ไบท์ ซึ่งคำสั่ง LODS จะใช้เพียง 1 ไบท์ ถึงแม้ว่าจะมีการเซ็ทค่าเริ่มแรกของ SI ท่านก็ใช้คำสั่ง LODS สำหรับการเคลื่อนย้ายข้อมูลไบท์,เวิร์ด,2 เวิร์ด ใน 1 เวลา

คำสั่งต่อไปนี้เทียบเท่าคำสั่ง LODSB

```
        MOV     AL,[SI]             ; LOAD CHARACTER IN AL
        INC     SI                  ; INCREMENT SI FOR NEXT
        ---------------------------------------------------

        CLD                         ;DIRECTION UP
        MOV     SI,OFFSET BUFFER    ;SOURCE BUFFER
        MOV     DI,OFFSET OUTPUT    ;DESTINATION BUFFER
        MOV     CX,10               ;BUFFER LENGTH
L1:
        LODSB                       ;COPY DS:[SI] INTO AL
        AND     AL,7FH              ;CLEAR HIGH BIT
        STOSB                       ;STORE AL AT ES:[DI]
        LOOP    L1
        ---
BUFFER  DB      0C8H,0FBH,0F5H,0CAH,41H,42H,43H,64H,87H,8CH
OUTPUT  DB      10 DUP(?)
```

คำสั่ง LODSW ในรูป 11-2 เป็นการทำงานเพียง 1 เวิร์ด โดยการนำไบท์แรกของNAME1 ไว้ใน AL และไบท์ที่ 2 ไว้ใน AH ดังนั้นค่า AX จะเก็บข้อมูล 1 เวิร์ด ค่า SI เพิ่มขึ้น 2

```
TITLE   STRING2  (COM)  USE OF LODS STRING OPERATION
        .MODEL   SMALL
        .CODE
        ORG      100H
BEGIN:  JMP      SHORT MAIN
; ---------------------------------------------------
NAME1   DB       'Assemblers'        ; DATA ITEM
; ---------------------------------------------------
```

```
MAIN    PROC    NEAR                ; MAIN PROCEDURE
        CLD                         ; LEFT TO RIGHT
        LEA     SI,NAME1            ; LOAD 1st WORD OF NAME1
        LODSW                       ;  INTO AX REGISTER
        MOV     AH,4CH              ; EXIT
        INT     21H
MAIN    ENDP
        END     BEGIN
```

รูปที่ 11-2 Use of LODSW string operation

ตัวอย่าง 11.7 การแสดงผลของสตริงโดยใช้คำสั่ง LOAD โดยให้ SI กำหนดตำแหน่งของสตริง และจำนวนตัวอักขระเก็บไว้ใน BX โดยมีอัลกอริทึมดังต่อไปนี้

```
FOR COUNT TIMES DO /*COUNT = NO. OF CHARACTER DISPLAY*/
 LOAD STRING CHARACTER INTO AL
 MOVE IT TO DL
 OUTPUT CHARACTER
END_FOR
```

สามารถเขียนโปรแกรมภาษาแอสแซมบลีได้ดังนี้

```
 DISP_STR   PROC
;DISPLAY A STRING
;INPUT: SI = OFFSET OF STRING
;       BX = NO. CHAR TO DISPLAY
;OUTPUT : NONE
           PUSH AX
           PUSH BX
           PUSH CX
           PUSH DX
           PUSH SI
           MOV  CX,BX          ;NO. OF CHARACTER
```

```
            JCXZ F_EXIT          ;EXIT IF NONE
            CLD                  ;PROCESS LEFT TO RIGHT
            MOV  AH,2            ;PREPARE TO PRINT
TOP:
            LODSB                ;CHARACTER IN AL
            MOV  DL,AL           ;MOVE IT IN DL
            INT  21H             ;PRINT CHARACTER
            LOOP TOP             ;LOOP UNTIL DONE
P_EXIT:
            POP  SI
            POP  DX
            POP  CX
            POP  BX
            POP  AX
            RET
DISP_STR    ENDP
```

STOS : STROE STRING

```
STOS        DEST
STOSB
STOSW
```

คำสั่ง STOS เป็นการเก็บข้อมูลของ AL,AX or EAX เก็บไว้ในหน่วยความจำในรูปของไบท์ เวิร์ด หรือ 2 เวิร์ด แอดเดรสของหน่วยความจำใช้รีจิสเตอร์ DI(ES:DI) การทำง่ายขึ้นอยู่กับ DF โดยคำสั่งของ STOS จะเพิ่มหรือลดค่า DI ทีละ 1,2 หรือ 4 ขึ้นอยู่กับชนิดข้อมูล

```
CLD                          ;DIRECTION UP
MOV     AL,0FFH              ;VALUE TO BE STORED
MOV     DL,OFFSET STRING1    ;ES:DI POINTS TO DESTINATION
```

```
        MOV     CX,100          ;CHARACTER COUNT
        REP     STOSB           ;FILL WITH COUNTERS OF AL
        ---
STRING1 DB      100 DUP (?)
```

ในทางปฏิบัติ การใช้คำสั่ง STOS กับ REP ในการเซ็ทค่าที่กำหนดในพื้นที่ของข้อมูลทุกค่า เช่น การเคลียร์พื้นที่แสดงผลด้วยค่า blank จำนวนข้อมูลที่เคลียร์อยู่ใน CX

คำสั่งที่เทียบเท่าคำสั่ง REP STOS คือ

```
        JCXZ    LABEL2          ; JUMP IF CX ZERO
LABEL1: MOV     [DI],AL         ; STORE AL IN MEMORY
        INC/DEC DI              ; INCREMENT OR DECREMENT
        LOOP    LABEL1
LABEL2: ---
```

คำสั่ง STOSW ในรูป 11-3 เป็นการทำงานซ้ำในการเก็บค่า 2020H(blanks) 5 ครั้งใน NAME1 การทำงานจะเก็บค่า AL ในไบท์แรกและ AH ในไบท์ต่อไป ส่วนข้อมูลใน CX=0 จะหยุดทำงานและข้อมูลใน DI ซึ่งเป็นแอดเดรสมีค่า NAME+10

```
TITLE   STRING3  (COM)  USE OF STOSW STRING OPERATION
        .MODEL   SMALL
        .CODE
        ORG      100H
BEGIN:  JMP      SHORT MAIN
; ---------------------------------------------------
NAME1   DB       'Assemblers'       ; DATA ITEM
; ---------------------------------------------------
MAIN    PROC     NEAR               ; MAIN PROCEDURE
        CLD                         ; LEFT TO RIGHT
        MOV      AX,2020H           ; MOVE
        MOV      CX,05              ;  5 BLANKS
```

```
        LEA     DI,NAME1            ;  TO NAME1
        REP STOSW
        MOV     AH,4CH              ; EXIT
        INT     21H
MAIN    ENDP
        END     BEGIN
```

รูปที่ 11-3 การใช้คำสั่ง STOWS การทำงานของสตริง

ตัวอย่าง 11.8 การอ่านข้อมูลและเก็บข้อมูลสตริงโดยใช้คำสั่ง STORE ดังอัลกอริทึมต่อไปนี้

```
CHAR_READ  =  0
     READ  A CHAR
          WHILE CHAR IS NOT A CARRIAGE RETURN DO
          IF CHAR IS A BACKSPACE
     THEN
          CHAR_ READ = CHAR_READ - 1
          REMOVE  PREVIOUS CHAR FROM STRING
     ELSE
          STORE CHAR IN STRING
     CHAR_READ = CHAR_READ + 1
     END_IF
          READ A CHAR
     END_WHILE
```

โปรแกรมภาษาแอสแซมบลี

```
READ_STR   PROC   NEAR
;READS AND STORE A STRING
; INPUT DI OFFSET OF STRING
;OUTPUT DI OFFSET OF STRING
;BX NO. OF CHAR READ
```

```
        PUSH AX
        PUSH DI
        CLD                     ;PROCESS FROM LEFT
        XOR  BX,BX              ;NO.OF CHAR READ
        MOV  AH,1               ;INPUT FUNCTION
        INT  21H                ;READ CHAR INTO AL
WHILE1:
        CMP  AL,0DH             ;CR?
        JE   END_WHILE1         ;YES EXIT
;IF CHAR IS BACKSPACE
        CMP  AL,08H             ;BS?
        JNE  ELSE1              ;NO. STORE IN STRING
;THEN
        DEC  DI                 ;YES, MOVE STRING PTR BACK
        DEC  BX                 ;DECREMENT CHAR COUNTER
        JMP  READ               ;AND GOTO READ ANOTHER CHAR
ELSE1:
        STOSB                   ;STORE CHAR IN STRING
        INC  BX                 ;INCREMENT CHAR COUT
READ:
        INT  21H                ;READ CHAR INTO AL
        JMP  WHILE1
END_WHILE1:
        POP  DI
        POP  AX
        RET
READ_STR  ENDP
```

จากตัวอย่างของโปรแกรมย่อยทั้งสอง เราสามารถนำมารวมกันในการเขียนโปรแกรมในการอ่านตัวอักขระและแสดงผลการพิมพ์ของตัวอักขระดังต่อไปนี้

```
                TITLE     READ+STR AND PRINT_STR
                .MODEL    SMALL
                .STACK
                .DATA
STRING          DB        80 DUP(0)
CRLF            DB        0DH,0AH,'$'
                .CODE
     MAIN       PROC
                MOV  AX,@DATA
                MOV  DS,AX
                MOV  ES,AX
          ;READ A STRING
                LEA  DI,STRING          ;DI PTS TO STRING
                CALL READ_STR
          ; GOTO A NEW LINE
                LEA  DX,CRLF
                MOV  AH,9
                INT  21H
          ;PRINT STRING
                LEA  SI,STRING          ;SI PTS TO STRING
                MOV  BX,10              ;DISPLAY 10 CHARACTER
                CALL DISP_STR
          ;DOES EXIT
                MOV  AH,4CH
                INT  21H
          MAIN ENDP
                END  MAIN
```

CMPS : COMPARE STRING

คำสั่งCMPS เป็นการเปรียบเทียบข้อมูลในหน่วยความจำ ที่แอดเดรสหนึ่งที่แอดเดรสกำหนด

โดย SI(DS:SI) กับข้อมูลในหน่วยความจำตำแหน่งอื่นๆ แอดเดรสกำหนดโดย ES:DI การเปรียบเทียบข้อมูลขึ้นอยู่กับทิศทาง(DF) คำสั่ง CMPS จะเพิ่มหรือลดค่าของ SI and DI ทีละ 1,2 หรือ 4 ขึ้นอยู่กับข้อมูล การทำงานจะเป็นการเซ็ทค่าแฟลก AF,CF,OF,PF,SF and ZF โดยใช้คำนำหน้า REP และความยาวใน CX คำสั่ง CMPS นี้จะใช้กับสตริงได้ทุกๆความยาว

พิจารณาจากการเปรียบเทียบสตริง 2 ชุด ของข้อมูล JEAN และ JOAN การเปรียบเทียบจากซ้ายไปขวาทีละไบท์ ดังนี้

```
J : J  equal
E : O  unequal
A : A  equal
N : N  equal
```

การเปรียบเทียบข้อมูล 4 ไบท์จนถึงค่า N เราจะเห็นว่าชื่อทั้งสองไม่เท่ากัน (not equal) การทำงานจะสิ้นสุดตั้งแต่ค่าของ E:O ซึ่งเป็น unequal สำหรับวัตถุประสงค์ของคำสั่ง REP จะมีตัวแปร REPE จะทำงานซ้ำตามความยาวของการเปรียบเทียบที่เท่ากัน หรือทำงานจนกระทั่ง CX=0 รหัสการทำงานซ้ำขนาด 1 ไบท์คือ REPE CMPSB

ตามรูป 11-4 การเปรียบเทียบข้อมูล 2 ตัว โดยใช้คำสั่ง CMPSB ตัวอย่างแรกเป็นการเปรียบเทียบ NAME1 กับ NAME2 ซึ่งข้อมูลมีค่าเหมือนกัน คำสั่ง CMPSB จะทำงานต่อไปจนกระทั่ง 10 ไบท์ ในการสิ้นสุดการทำงานค่า cx=0 และ DI=NAME2+10 แอดเดรสใน SI เท่ากับ NAME1+10 , ค่า SF=บวกและค่า ZF จะมีค่าเท่ากับหรือศูนย์

ในตัวอย่างที่ 2 เป็นการเปรียบเทียบ NAME2 and NAME3 ซึ่งมีข้อมูลที่แตกต่างกัน คำสั่ง CMPSB จะสิ้นสุดการทำงานหลังจากการเปรียบเทียบไบท์แรก และผลลัพธ์จะเท่ากับ high /unequal ค่าของ CX=9 ค่าของ DI=NAME3+1 ค่าของ SI=NAME2+1 SF เป็นบวกและ ZF บอกว่า unequal

ตัวอย่างแรก ผลลัพธ์ใน equal/zero และเคลื่อนค่า 1 ใน BH ตัวอย่าง 2 ผลลัพธ์ unequal และเคลื่อนค่า 2 ใน BL ถ้าท่านใช้ debug ตรวจสอบรีจิสเตอร์ BX=0102 หลังจากหยุดการ execute

```
TITLE  STRING4  (COM)  USE OF CMPS STRING OPERATION
       .MODEL   SMALL
       .CODE
       ORG      100H
BEGIN: JMP      SHORT MAIN
; ---------------------------------------------------
NAME1  DB       'Assemblers'          ; DATA ITEM
NAME2  DB       'Assemblers'
NAME3  DB       10 DUP(' ')
; ---------------------------------------------------
MAIN   PROC     NEAR                  ; MAIN PROCEDURE
       CLD                            ; LEFT TO RIGHT
       MOV      CX,10                 ; INITIALIZE FOR 10 BYTES
       LEA      DI,NAME2
       LEA      SI,NAME1
       REPE CMPSB                     ; COMPARE NAME1:NAME2
       JNE      G20                   ; NOT EQUAL -- BYPASS
       MOV      BH,01                 ; EQUAL -- SET BH
G20:
       MOV      CX,10                 ; INITIALIZE FO 10 BYTES
       LEA      DI,NAME3
       LEA      SI,NAME2
       REPE CMPSB                     ; COMPARE NAME2:NAME3
       JE       G30                   ; EQUAL -- EXIT
       MOV      BL,02                 ; NOT EQUAL -- SET BL
G30:
       MOV      AH,4CH                ; EXIT
       INT      21H
MAIN   ENDP
       END      BEGIN
```

รูปที่ 11-4 Use of CMPS string operations

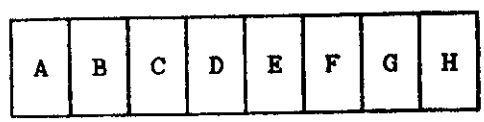
SCAS : SCAN STRING

```
SCAS    DEST
SCASB
SCASW
```

คำสั่ง SCAS จะแตกต่างจากคำสั่ง CMPS เพราะว่ามันจะสแกนสตริงสำหรับไบท์ เวิร์ด หรือ 2 เวิร์ดที่กำหนด คำสั่ง SCAN จะทำการเปรียบเทียบข้อมูลของหน่วยความจำ (แอดเดรสกำหนดโดย ES:DI) กับค่า AL,AX or EAX ขึ้นอยู่กับแฟลกทิศทาง คำสั่ง SCAS จะเพิ่มหรือลดค่ารีจิสเตอร์ DI ทีละ 1,2 หรือ 4ขึ้นอยู่กับข้อมูลหลังจากการ execute คำสั่ง SCAS จะเซ็ทค่าแฟลก AF,CF,OF,PF,AF and ZF เมื่อทำงานร่วมกับคำสั่ง REP รีจิสเตอร์ CX เก็บค่าความยาว คำสั่ง SCAS จะใช้ได้กับสตริงทุกๆความยาว

```
        CLD                     ;DIRECTION UP
        MOV   DI,OFFSET ALPHA   ;ES:DI POINTS TO THE STRING
        MOV   AL,'F'            ;SEARCH FOR THE LETTER 'F'
        MOV   CX,8              ;SET THE SEARCH COUNT
        REPNE SCASB             ;REPEAT WHILE NOT EQUAL
        JNZ   EXIT              ;QUIT IF LETTER IS NOT FOUND
        DEC   DI                ;FOUND: BACK UP DI ONE CHARACTER
        ---
ALPHA   DB    'ABCDEFGH',0
```

After the search



DI

คำสั่ง SCAS จะประยุกต์ใช้กับการแก้ไข text ซึ่งโปรแกรมจะสแกนสำหรับหาเครื่องหมาย เช่น periods,commas and blanks

ตัวอย่างรูป 11-5 จะสแกน NAME1 สำหรับตัวอักษร 'm' โดยใช้คำสั่ง SCASB ในการสแกนอย่างต่อเนื่อง ขณะที่การเปรียบเทียบไม่เท่ากับหรือจนกระทั่ง CX=0 โดยใช้คำสั่ง REPNE SCASB

```
TITLE   STRING5  (COM)  USE OF SCAS STRING OPERATION
        .MODEL   SMALL
        .CODE
        ORG      100H
BEGIN:  JMP      SHORT MAIN
; ---------------------------------------------------
NAME1   DB       'Assemblers'          ; DATA ITEM
; ---------------------------------------------------
MAIN    PROC     NEAR                  ; MAIN PROCEDURE
        CLD                            ; LEFT TO RIGHT
        MOV      AL,'m'
        MOV      CX,10                 ; SCAN NAME1
        LEA      DI,NAME1              ;  FOR 'm'
        REPE SCASB
        JNE      H20                   ; IF FOUND,
        MOV      AH,30                 ;  STORE 30 IN AH
H20:
        MOV      AH,4CH                ; EXIT
        INT      21H
MAIN    ENDP
        END      BEGIN
```

Figure 11-5 Use of SCASB string operation

ข้อมูลใน NAME1 คือ "Assemblers" คำสั่ง SCASB จะที่เท่ากับตัวที่ 5 ถ้าท่านใช้ debug ใน execute ทีละคำสั่ง จุดสุดท้ายของการ execute คำสั่ง REP SCASB แฟลก ZF=0 ค่าของ CX ลดลง 5 และ DI จะเพิ่มขึ้น 5

CT 215

โปรแกรมจะเก็บค่า 03 ใน AH เพื่อชี้ว่า 'm' ได้ตรวจพบ ถ้าไม่เท่ากันแสดงว่าไม่พบ การทำงานจะเซ็ทค่าแฟลกศูนย์(ZF)เป็นค่า 1

คำสั่ง SCASW เป็นการสแกนสำหรับค่าเวิร์ด ในหน่วยความจำที่ matches กับเวิร์ดใน AX ถ้าใช้คำสั่ง LODSW หรือ MOV ในการเคลื่อนย้ายเวิร์ดลงใน AX ไบท์แรกจะเก็บใน AL และไบท์ที่ 2 จะเก็บใน AH

SCAN AND REPLACE

ท่านสามาถแทนที่ตัวอักขระพิเศษด้วยค่าตัวอักขระอื่นๆ สำหรับตัวอย่างในการเคลียร์อักขระที่แก้ไข เช่น paragraph และสัญลักษณ์สุดท้ายของหน้ากระดาษ โปรแกรมจะสแกนสตริงดังต่อไปนี้ สำหรับ &(ampersand) แทนที่ด้วยค่า blank ถ้าคำสั่ง SCASB พบ ampersand มันจะสิ้นสุดการทำงาน ในตัวอย่างนี้มีตัวอักขระ ampersand อยู่ที่ STRING+8 ขณะที่ตัวอักขระ blank จะถูกนำมาแทนที่ การทำงานของคำสั่ง SCASB จะเพิ่มค่า DI ทีละ 1 เป็น STRING+9 การลดค่า DI ทีละ1 จะเป็นการกำหนดแอดเดรสสำหรับ blank

```
SRLEN   EQU     15              ; LENGTH OF STRING
STRING  DB      'THE TIME& IS NOW'
        ---
        CLD                     ; LEFT TO RIGHT
        MOV     AL,'&'          ; SEARCH CHARACTER
        MOV     CX,STRLEN       ; LENGTH OF STRING
        LEA     DI,STRING       ; ADDRESS OF STRING
        REPNE SCASB             ; SCAN
        JNZ     K20             ; FOUND
        DEC     DI              ; YES -- ADJUST ADDRESS
        MOV     BYTE PTR[DI],20H    ; REPLACE WITH BLANK
K20:    ---
```

ALTERNATE CODING FOR STRING INSTRUCTIONS

ถ้าท่านกำหนดรหัสคำสั่งเป็นไบท์ เวิร์ด หรือ 2 เวิร์ด เช่น MOVSB, MOVSW or MOVSP assemblerจะกำหนดความยาวได้ถูกต้อง และไม่ต้องมีโอเปอรานด์ เช่นคำสั่ง MOVS ซึ่ง

ไม่มีตัวต่อท้าย (suffix) ในการกำหนดไบท์ เวิร์ดหรือ 2 เวิร์ด ท่านจะต้องแสดงความยาวของโอเปอรานด์ สำหรับตัวอย่าง ถ้า FLDA and FLDB กำหนดเป็นไบท์(DB)

```
REP     MOVS    FLDA,FLDB
```

คำสั่งจะเคลื่อนย้ายข้อมูลขนาดไบท์ซ้ำๆกัน เริ่มจาก FLDA ไปยัง FLDB ถ้าท่านโหลด DI and SI ด้วยแอดเดรส FLDA and FLDB ท่านก็กำหนดรหัสคำสั่งได้ดังนี้

```
REP     MOVS    ES:BYTE PTR[DI],DS:[SI]
```

DUPLICATING A PATTERN

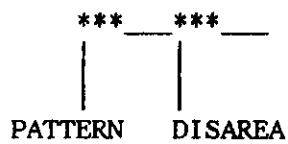
คำสั่ง STOS ใช้ประโยชน์สำหรับการเซ็ทพื้นที่ตามไบท์ที่กำหนด เวิร์ดและ 2 เวิร์ดที่กำหนด สำหรับรูปแบบการทำงานซ้ำๆ ท่านสามารถใช้คำสั่ง MOVS ดัดแปลง เพื่อให้แสดงรูปแบบของบรรทัดต่อไปนี้

```
***___***___***___***___***___...
```

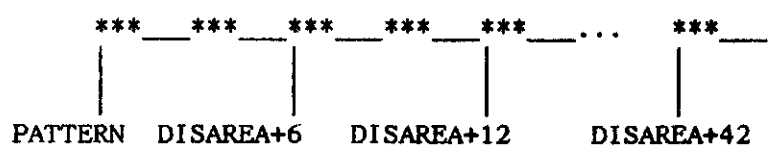
การทำงานซ้ำๆและรวดเร็วได้โดยท่านต้องกำหนด 6 ไบท์แรกทันที ก่อนที่จะแสดงผลในบรรทัด ตามรหัสคำสั่งต่อไปนี้

```
PATTERN  DB      '***___'
DISAREA  DB      42 DUP(?)
         ---
         CLD
         MOV     CX,21
         LEA     DI,DISAREA
         LEA     SI,PATTERN
         REP     MOVSW
```

การ execute ใช้คำสั่ง MOVSW จะเป็นการเคลื่อนย้ายเวิร์ดแรกของ PATTERN(**) ไปยังเวิร์ดแรกของ DISAREA และเคลื่อนย้ายไบท์ที่ 2 (*_) และไบท์ที่ 3 (__)



ที่จุดนี้ข้อมูลของ DI เป็นแอดเดรสของ DISAREA+6 และ SI มีข้อมูลแอดเดรสของ PATTERN+6 ซึ่งเป็นแอดเดรสของ DISAREA การทำงานจะทำสำเนาอัตโนมัติ ในรูปแบบของการเคลื่อนเวิร์ดแรกของ DISAREA ไป DISAREA+6 DISAREA+2 ไป DISAREA+8 DISAREA+4 ไป DISAREA+10 และต่อไปเรื่อยๆ



RIGHT-ADJUSTING ON THE SCREEN

โปรแกรมในรูป 11-6 เป็นการแสดงการใช้คำสั่งที่อธิบายในบทนี้ มี procedure ดังต่อไปนี้

B10INPT รับข้อมูล 30 ตัวอักขระ ในส่วนบนจอภาพ

D10SCAS ใช้คำสั่ง SCASB สแกนชื่อและผ่านทุกตัวมีข้อมูล asterisk

E10RGHT ใช้คำสั่ง MOVSB สำหรับการปรับชื่อที่ป้อนเข้ามาไปทางขวาของจอภาพ ความยาวของ ACTNLEN จะอยู่ในส่วนที่ป้อนเข้ามาของพารามิเตอร์ คือใช้คำนวณสำหรับตัวอักขระขวาสุดของชื่อ ดังนี้

JEROME KERN
RICHARD RODGERS
IRVING BERLIN

F10CLNM ใช้คำสั่ง STOSW ในการเคลียร์ชื่อในหน่วยความจำ

```
TITLE     EXRIGHT (EXE)  Right-adjust displayed names
          .MODEL  SMALL
          .STACK  96
; ------------------------------------------------------
          .DATA
NAMEPAR   LABEL   BYTE              ;Name parameter list
MAXNLEN   DB      31                ;Maximum length
ACTNLEN   DB      ?                 ;No. of chars entered
NAMEFLD   DB      31 DUP(' ')       ;Name

PROMPT    DB      'Name?', '$'
NAMEDSP   DB      31 DUP(' '), 13, 10, '$'
ROW       DB      00
; ------------------------------------------------------
          .CODE
BEGIN     PROC    FAR               ;Main procedure
          MOV     AX,@data          ;Initialize
          MOV     DS,AX             ;  data segment
          MOV     ES,AX
          MOV     AX,0600H
          CALL    Q10SCR            ;Clear screen
          SUB     DX,DX             ;Set cursor 00,00
          CALL    Q20CURS
A10LOOP:
          CALL    B10INPT           ;Request input of name
          TEST    ACTNLEN,0FFH      ;No name? (indicates end)
          JZ      A90               ;  yes - exit
          CALL    D10SCAS           ;Scan for asterisk
          CMP     AL,'*'            ;Found?
          JE      A10LOOP           ;  yes - bypass
          CALL    E10RGHT           ;Right-adjust name
```

```
            CALL    F10CLNM                 ;Clear name
            JMP         A10LOOP
A90:        MOV         AH,4CH              ;Return to DOS
            INT         21H
BEGIN       ENDP
;Prompt for input:
;-----------------
B10INPT      PROC
            MOV         AH,09
            LEA         DX,PROMPT           ;Display prompt
            INT         21H
            MOV         AH,0AH
            LEA         DX,NAMEPAR          ;Accept input
            INT         21H
            RET
B10INPT      ENDP
;                   Scan name for asterisk:
;                   -----------------------
D10SCAS      PROC
            CLD                             ;Left to right
            MOV         AL,'*'
            MOV         CX,30               ;Set 30-byte scan
            LEA         DI,NAMEFLD
            REPNE SCASB                     ;Asterisk found?
            JE          D20                 ;  yes - exit
            MOV         AL,20H              ;  no  - clear * in AL
D20:        RET
D10SCAS      ENDP

;Right-adjust & display name:
;----------------------------
```

```
E10RGHT     PROC
            STD                         ;Right to left
            MOV         CH,00
            MOV         CL,ACTNLEN      ;Length in CX for REP
            LEA         SI,NAMEFLD      ;Calculate rightmost
            ADD         SI,CX           ;  position
            DEC         SI              ;  of input name
            LEA         DI,NAMEDSP+30   ;Right pos'n of display name
            REP MOVSB                   ;Move string right to left
            MOV         DH,ROW
            MOV         DL,48
            CALL    Q20CURS             ;Set cursor
            MOV         AH,09
            LEA         DX,NAMEDSP      ;Display name
            INT         21H

            CMP         ROW,20          ;Bottom of screen?
            JAE         E20             ;  no --
            INC         ROW             ;  increment row
            JMP         E90
E20:
            MOV         AX,0601H        ;  yes --
            CALL    Q10SCR              ;  scroll and
            MOV         DH,ROW          ;  set cursor
            MOV         DL,00
            CALL    Q20CURS
E90:        RET
E10RGHT     ENDP
;Clear name:
;----------
```

```
F10CLNM     PROC
            CLD                         ;Left to right
            MOV         AX,2020H
            MOV         CX,15           ;Clear 15 words
            LEA         DI,NAMEDSP
            REP STOSW
            RET
F10CLNM     ENDP
;                       Scroll screen:
;                       -------------
Q10SCR      PROC                        ;AX set on entry
            MOV         BH,30           ;Color (07 for BW)
            MOV         CX,00
            MOV         DX,184FH
            INT         10H
            RET
Q10SCR      ENDP
;Set cursor row/col:
;-------------------
Q20CURS     PROC                        ;DX set on entry
            MOV         AH,02
            SUB         BH,BH
            INT         10H
            RET
Q20CURS     ENDP

            END         BEGIN
```

ตัวอย่างการทำงานของสตริง

The READSTRING Procedure เป็นการอ่านการทำงานของสตริงรหัส ASCII จากคอนโซล เมื่อมีการเรียก Procedure DS:DX เป็นตัวชี้บัฟเฟอร์รับข้อมูล และตัวนับเก็บใน CX กำหนดค่าสูงสุดของตัวอักขระในการรับ การสร้างบัฟเฟอร์การรับข้อมูลต้องใหญ่กว่าข้อมูลที่รับเข้ามาจากคีย์บอร์ด ดังตัวอย่าง

```
                MOV   CX,20
                MOV   DX,OFFSET INPUT_BUFFER
                CALL READSTRING
;-------------------------------------------------------------
READSTRING      PROC                    ;DS:DX POINTS TO STRING
                PUSH    BX
                PUSH    CX
                PUSH    DX
                ADD     CX,2            ;ADD 2 TO COUNT FOR  CR = LF
                MOV     AH,3FH          ;FUNCTION:READ FROM DEVICE
                MOV     BX,0            ;DEVICED = KEYBOARD
                INT     21H
                MOV     BX,DX           ;GET OFFSET OF STRING
                SUB     AX,2            ;SUBTRACT CR/LF FROM LENGTH
                ADD     BX,AX           ;ADD NUMBER OF CHARACTERS ENTERED
                MOV     BYTE PTR[BX],0      ;INSERT BINARY 0
                POP     DX
                POP     CX
                POP     BX
                RET                     ;AX = SIZE OF INPUT STRING
READSTRING      ENDP

;-------------------------------------------------------------
```

The STRCHR Procedure

การทำงานของ Procedure เป็นการค้นหาสตริงที่ชี้โดย ES:DI สำหรับตัวอักขระใน AL ถ้าตรวจพบตัวอักขระ ข้อมูลของ DI จะเป็นค่าออฟเซ็ตและแฟลกตัวทดจะถูกเคลีย ส่วนกรณีอื่นๆ แฟลกตัวทดจะถูกเซ็ต และ DI จะเก็บค่าเดิม

```
STRCHR  PROC
        PUSH    AX
        PUSH    CX
        PUSH    DI          ;SAVE POINTER TO STRING
        PUSH    AX          ;SAVE THE CHARACTER
        CALL    STRLEN      ;GET STRING LENGTH
        MOV     CX,AX
        POP     AX          ;RETRIEVE THE CHARACTER
        CLD                 ;SET DIRECTION TO UP
REPNE   SCASB               ;CHARACTER FOUND?
        JNZ     L1          ;NOT:SET CARRY FLAG AND QUIT
        DEC     DI          ;YES: SAVE POSITION IN DI
        CLC                 ;CLEAR THE CARRY FLAG AND EXIT
        POP     AX          ;THROW AWAY OLD DI ON STACK
        JMP     L2
L1:     STC                 ;CHARACTER NOT FOUND:SET CARRY
        POP     DI
L2:     POP     CX
        POP     AX
        RET
STRCHR  ENDP
;---------------------------------------------------------------
```

The STRCOMP Procedure

การทำงานของ Procedure เป็นการเปรียบเทียบแหล่งส่งสตริงที่ DS:SI กำหนด กับแหล่งรับสตริงที่ ES:DI กำหนด และเซ็ตแฟลกตามการทำงาน ถ้าแหล่งส่งสตริงน้อยกว่าแหล่งรับสตริง แฟลกตัวทดจะถูกเซ็ต ถ้าสตริงมีค่าเท่ากัน แฟลกศูนย์ (ZERO FLAG)และแฟลกตัวทด = 0 ดังตัวอย่างของของ STRING1 กับ STRING2 ดังนี้

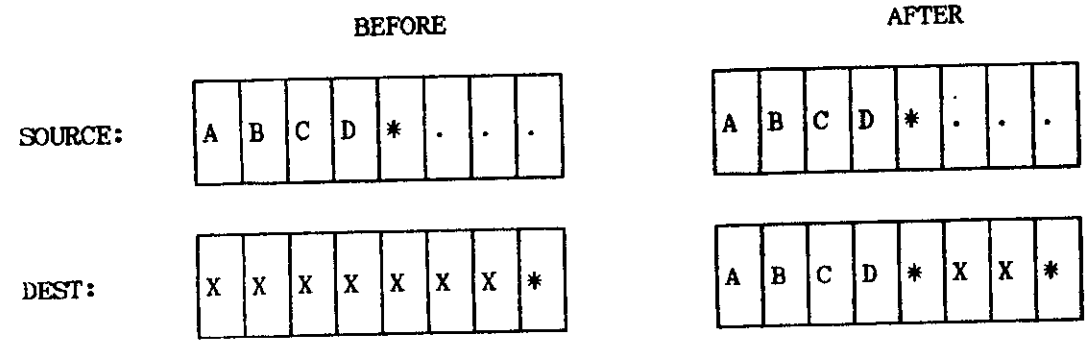
```
                MOV     SI,OFFSET STRING1    ;SOURCE IS STRING1
                MOV     DI,OFFSET STRING2    ;DESTINATION IS STRING2
                CALL    STRCOMP              ;COMPARE,SET THE FLAGS
                JB      STRING1_LESS         ;JUMP IF STRING1 IS LESS
                ...
STRING1_LESS:
```

ภายในการทำงานของ Procedure STRCOMP จะมีการเรียก STRLEN เพื่อหาความยาวของสตริงแหล่งส่ง ซึ่งใช้เป็นเคาน์เตอร์ เราจะเคลียแฟลกทิศทาง เป็น UP และเปรียบเทียบสตริงทั้งสอง

```
STRCOMP         PROC
                PUSH    AX
                PUSH    CX
                PUSH    SI
                PUSH    DI
                CALL    STRLEN          ;GET LENGTH OF DESTINATION
                MOV     CX,AX           ;STORE IN CX
                CLD                     ;CLEAR DIRECTION TO UP
        REP     CMPSB                   ;COMPARE AND SET THE FLAGS
                POP     DI
                POP     SI
                POP     CX
                POP     AX
                RET
        STRCOMP ENDP
```

The STRCOPY Procedure

การทำงานของ Procedure STRCOPY จะทำการทำสำเนาสตริง DS:SI ไปยังสตริงของตัวรับที่ ES:DI ตามไดอะแกรมที่แสดงในการทำสำเนาสตริงดังนี้



* = ZERO TERMINATOR BYTE.

```
STRCOPY     PROC
            PUSH    AX
            PUSH    CX
            PUSH    SI
            PUSH    DI
            PUSH    DI          ;SAVE DEST POINTER
            MOV     DI,SI       ;GET LENGTH OF SOURCE STRING
            CALL    STRLEN      ;RETURNS LENGTH IN AX
            POP     DI          ;RESTORE DEST POINTER
            INC     AX          ;ADD 1 FOR ZERO TERMINTOR
            MOV     CX,AX       ;SET CX TO LENGTH
            CLD                 ;CLEAR DIRECTION TO UP
      REP   MOVSB               ;COPY THE STRING
            POP     DI
            POP     SI
            POP     CX
            POP     AX
            RET
STRCOPY     ENDP
```

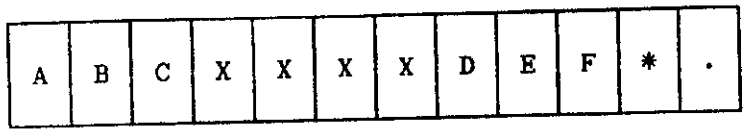
The STRDEL Procedure]

การทำงานของ Procedure STRDEL จะเป็นการลบตัวอักขระที่กำหนดจากทุกๆตำแหน่งของข้อมูลสตริง เมื่อมีการเรียกโปรแกรมย่อย STRDEL ตัวชี้ข้อมูลคือ ES:DI เป็นการชี้ตัวอักขระตัวแรก ที่จะทำการลบ และเซ็ตค่าของ CX เท่ากับจำนวนตัวอักขระที่จะลบ การลบตัวอักขระจากชุดสตริง โปรแกรม STRDEL จะทำสำเนาข้อมูลไปข้างหน้า ดังนั้นตัวอักขระจะถูกเขียนทับ รีจีสเตอร์ CX จะชี้จำนวนของตัวอักขระที่ลบ ดังนั้นเราจะบวก CX กับ DI ที่แหล่งของตัวชี้ ตำแหน่งนี้จะชี้ตัวอักขระตัวแรกที่ทำสำเนา

```
STRDEL      PROC                ;ES:DI POINTS TO FIRST CHARACTER
            PUSH    AX          ;TO BE DELETED.
            PUSH    CX          ;CX=NUMBER OF CHARS TO DELETE
            PUSH    SI
            PUSH    DI
            CALL    STRLEN      ;HOW MANY CHARS AVAILABLE?
            CMP     AX,CX       ;MORE THAN CX?
            JA      L1          ;YES:CX STAYS THE SAME
            MOV     CX,AX       ;NO:CX=NUMBER AVAIL CHARS
L1:         MOV     SI,DI       ;FIND POSITION TO COPY FROM:
            ADD     SI,CX       ;SI=DI+DI
            SUB     AX,CX       ;CALC NUMBER BYTES TO MOVE:
            INC     AX          ;CX = (AX-CX)+1
            MOV     CX,AX
            CLD                 ;SET DIRECTION TO UP
      REP   MOVSB               ;COPY THE BYTES
            POP     DI
            POP     SI
            POP     CX
            POP     AX
            RET
STRDEL      ENDP
```

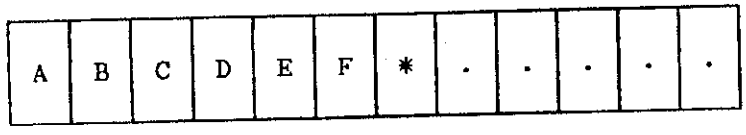
ตัวอย่าง สมมุติการลบตัวสตริง XXXX จากชุดสตริง "ABCXXXXDEF". ดังตัวอย่างที่แสดงในการทำงานก่อนการลบและหลังการลบสตริง

BEFORE



DI SI

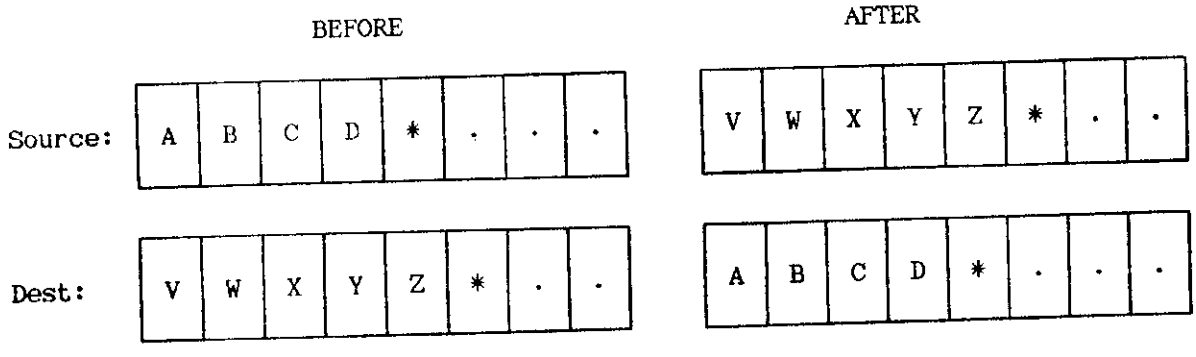
AFTER:



* = zero terminator byte.

The STREXCH Procedure

การทำงานของโปรแกรมย่อย STREXCH เป็นการแลกเปลี่ยนข้อมูลของสตริง 2 ชุด การเรียกโปรแกรมนี้แอดเดรสของสตริงอยู่ใน DS:SI และ ES:DI



* = zero terminator byte.

อัลกอริทึมการใช้ STREXCH ที่ใช้ตรวจสอบสำหรับการสิ้นสุดดังนี้

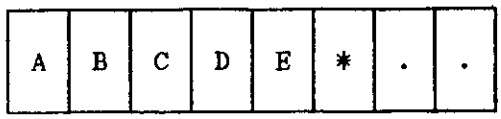
```
DO WHILE (source <> 0 ) and (dest <> 0)
    EXchang source , dest
ENDDO
```

STREXCH procedure มีดังต่อไปนี้

```
STREXCH     PROC
            PUSH    AX
            PUSH    SI
            PUSH    DI
L1:         CMP     BYTE PTR [SI],0     ;TERM BYTE IN SOURCE?
            JNE     L2                  ;NO:CONTINUE
            CMP     BYTE PTR [DI],0     ;TERM BYTE IN DESTINATION
            JNE     L2                  ;NO:CONTINUE
            JMP     L3                  ;YES:FINISHED
L2:         MOV     AL,[DI]             ;GET DESTINATION BYTE
            XCHG    AL,[SI]             ;EXCHANGE WITH SOURCE
            MOV     [DI],AL
            INC     SI                  ;MOVE TO NEXT POSITION
            INC     DI
            JMP     L1                  ;GET NEXT TWO CHARACTERS
L3:         POP     DI
            POP     SI
            POP     AX
            RET
STREXCH     ENDP
```

The STRLEN Procedure

การทำงานของโปรแกรมย่อย STRLEN จะทำการสแกนตัวสตริงที่ชี้โดย ES:DI สำหรับ ค่าของ zero terminator byte และความยาวของสตริงอยู่ใน AX โปรแกรมจะใช้คำสั่ง SCASB เพื่อหาค่าของ zero terminator จากตำแหน่งลบออกจากแอดเดรสของสตริง สมมุติว่าสตริงเริ่มต้นที่แอดเดรส 0004H และ zero terminator พบที่แอดเดรส 0009H ความยาวของสตริงคือ 5 ไบท์



ADDRESS: 0004H 0009H LENGTH=(0009H-0004H)

```
;-----------------------------------------------------------------
    STRLEN      PROC                ;ES:DI POINTS TO THE STRING
                PUSH    CX
                PUSH    DI          ;SAVE POINTER TO STRING
                MOV     CX,0FFFFH   ;SET CX TO MAXIMUM WORD VALUE
                MOV     AL,0        ;SCAN FOR ZERO TERMINATOR
                CLD                 ;DIRECTION UP
    REPNZ       SCASB               ;COMPARE AL TO ES:[DI]
                DEC     DI          ;BACK UP ONE POSITION
                MOV     AX,DI       ;GET ENDING POINTER
                POP     DI          ;RETRIEVE STARTING POINTER
                SUB     AX,DI       ;SUBTRACT START FROM END
                POP     CX
                RET                 ;AX = STARTING LENGTH
    STRLEN      ENDP
;-----------------------------------------------------------------
```

สรุป

- รูปแบบของหน่วยความจำสตริง ที่เก็บในหน่วยความจำมีมาตราฐาน 3 ชนิดคือ ความยาวสตริงคงที่ ความยาวที่เปลี่ยนค่าได้ ความยาวของสตริงที่ตามด้วย zero terminator byte
- คำสั่งสตริงมี 5 คือ MOVS , CMPS , SCAS , STOS , LODS

- รีจีสเตอร์ที่ใช้กับคำสั่งสตริงมีดังต่อไปนี้

รีจีสเตอร์	คำสั่ง
SI , DI	MOVS , CMPS
DI	SCAS , STOS
SI	LODS

- รีจีสเตอร์ DS:SI เป็นตัวชี้ของแหล่งส่งสตริง และรีจีสเตอร์ ES:DI เป็นตัวชี้ของแหล่งรับสตริง
- คำสั่งสตริงจะมีคำนำหน้าคือ REPEAT PREFIX ที่ใช้ร่วมกับรีจีสเตอร์ CX มีดังนี้

  REP

  REPZ , REPE

  REPNZ , REPNE

- แฟลกทิศทาง ใช้คำสั่ง CLD , STD กำหนดทิศทางการเคลื่อนย้ายของสตริง
- คำสั่ง CMPSW และ SCASW จะเปรียบเทียบข้อมูลสลับไบท์
- ขณะที่ท่านประมวลผลจากขวาไปซ้าย การกำหนดแอดเดรสเริ่มแรกอยู่ที่ไบท์ขวามสุดของฟิลด์ ถ้าฟิลด์ชื่อ NAME1 และมีความยาว 10 ไบท์ การประมวลผลไบท์แรกจะต้องโหลดที่แอดเดรส NAME + 9 ถ้าเป็นการประมวลผลแบบเวิร์ด จะกำหนดแอดเดรสที่ NAME + 8

แบบฝึกหัด

11.1 การทำงานของคำสั่งสตริงถ้าโอเปอร์รานด์มีความสัมพันธ์กับรีจีสเตอร์ DI หรือ SI จงกำหนดการทำงานของคำสั่งต่อไปนี้

a) MOVS (operand1 and 2)

b) CPMS (operand1 and 2)

c) SCAS (operand1)

11.2 การทำงานของคำสั่งสตริงในการใช้ REP ท่านจะกำหนดจำนวนการทำซ้ำอย่างไร

11.3 สำหรับการทำงานของคำสั่งสตริงในการทำงานซ้ำในการเซ็ตจากขวาไปซ้ายได้อย่างไร

11.4 จากข้อมูลที่กำหนดให้จงเขียนคำสั่งในการทำงานของสตริงจากข้อ a - f

```
DATASG    SEGMENT    PARA
CONAME    DB         'SPACEEXPLORERS INC.'
PRLINE    DB         20DUP(' ')
```

a) เคลื่อนย้ายข้อมูล CONAME ไปที่ PRLINE จากซ้ายไปขวา

b) เคลื่อนย้ายข้อมูล CONAME ไปที่ PRLINE จากขวาไปซ้าย

c) โหลดไบท์ที่ 3 และไบท์ที่ 4 ของ CONAME ไปที่ AX

d) เก็บค่าของ AX ที่แอดเดรส PRLINE+5

e) เปรียบเทียบว่าข้อมูลของ CONAME กับ PRLINE เท่ากันหรือไม่

f) สแกนค่าของ CONAME สำหรับค่า BLANK CHARACTER ถ้าพบเก็บใน BH

11.5 จงเขียนโปรแกรม PASSWORD ขนาด 3 ไบท์ โดยเขียนมีการตรวจสอบ 3 ครั้ง ถ้าไม่ถูกต้องให้กลับสู่ DOS
