Chapter Overview

- **String Primitive Instructions**

String Primitive Instructions

- MOVSB, MOVSW, and MOVSD
- CMPSB, CMPSW, and CMPSD
- SCASB, SCASW, and SCASD
- STOSB, STOSW, and STOSD
- LODSB, LODSW, and LODSD

MOVSB, MOVSW, and MOVSD (1 of 2)

- The MOVSB, MOVSW, and MOVSD instructions copy data from the memory location pointed to by ESI to the memory location pointed to by EDI.

```
.data
source DWORD 0FFFFFFFFh
target DWORD ?
.code
mov esi,OFFSET source
mov edi,OFFSET target
movsd
```

MOVSB, MOVSW, and MOVSD (2 of 2)

- ESI and EDI are automatically incremented or decremented:
  - MOVSB increments/decrements by 1
  - MOVSW increments/decrements by 2
  - MOVSD increments/decrements by 4

Direction Flag

- The Direction flag controls the incrementing or decrementing of ESI and EDI.
  - DF = clear (0): increment ESI and EDI
  - DF = set (1): decrement ESI and EDI

The Direction flag can be explicitly changed using the CLD and STD instructions:

```
CLD             ; clear Direction flag
STD             ; set Direction flag
```

Using a Repeat Prefix

- REP (a repeat prefix) can be inserted just before MOVSB, MOVSW, or MOVSD.
- ECX controls the number of repetitions
- Example: Copy 20 doublewords from source to target

```
.data
source DWORD 20 DUP(?)
target DWORD 20 DUP(?)
.code
cld                          ; direction = forward
mov ecx,LENGTHOF source      ; set REP counter
mov esi,OFFSET source
mov edi,OFFSET target
rep movsd
```

Your turn . . .

- Use MOVSD to delete the first element of the following doubleword array. All subsequent array values must be moved one position forward toward the beginning of the array:

```
array DWORD 1,1,2,3,4,5,6,7,8,9,10
```

```
.data
array DWORD 1,1,2,3,4,5,6,7,8,9,10
.code
cld
mov ecx,(LENGTHOF array) - 1
mov esi,OFFSET array+4
mov edi,OFFSET array
rep movsd
```

CMPSB, CMPSW, and CMPSD

- The CMPSB, CMPSW, and CMPSD instructions each compare a memory operand pointed to by ESI to a memory operand pointed to by EDI.
  - CMPSB compares bytes
  - CMPSW compares words
  - CMPSD compares doublewords
- Repeat prefix often used
  - REPE (REPZ)
  - REPNE (REPNZ)

Comparing a Pair of Doublewords

If source > target, the code jumps to label L1; otherwise, it jumps to label L2

```
.data
source DWORD 1234h
target DWORD 5678h

.code
mov esi,OFFSET source
mov edi,OFFSET target
cmpsd              ; compare doublewords
ja L1              ; jump if source > target
jmp L2             ; jump if source <= target
```

Your turn . . .

- Modify the program in the previous slide by declaring both source and target as WORD variables. Make any other necessary changes.

Comparing Arrays

Use a REPE (repeat while equal) prefix to compare corresponding elements of two arrays.

```
.data
source DWORD COUNT DUP(?)
target DWORD COUNT DUP(?)
.code
mov ecx,COUNT                ; repetition count
mov esi,OFFSET source
mov edi,OFFSET target
cld                          ; direction = forward
repe cmpsd                   ; repeat while equal
```

; The REPE prefix repeats the comparison, incrementing ESI and EDI automatically until ECX equals zero or a pair of doublewords is found to be different.

Example: Comparing Two Strings (1 of 3)

This program compares two strings (source and destination). It displays a message indicating whether the lexical value of the source string is less than the destination string.

```
.data
source BYTE "MARTIN  "
dest   BYTE "MARTINEZ"
str1 BYTE "Source is smaller",0dh,0ah,0
str2 BYTE "Source is not smaller",0dh,0ah,0
```

Screen output:

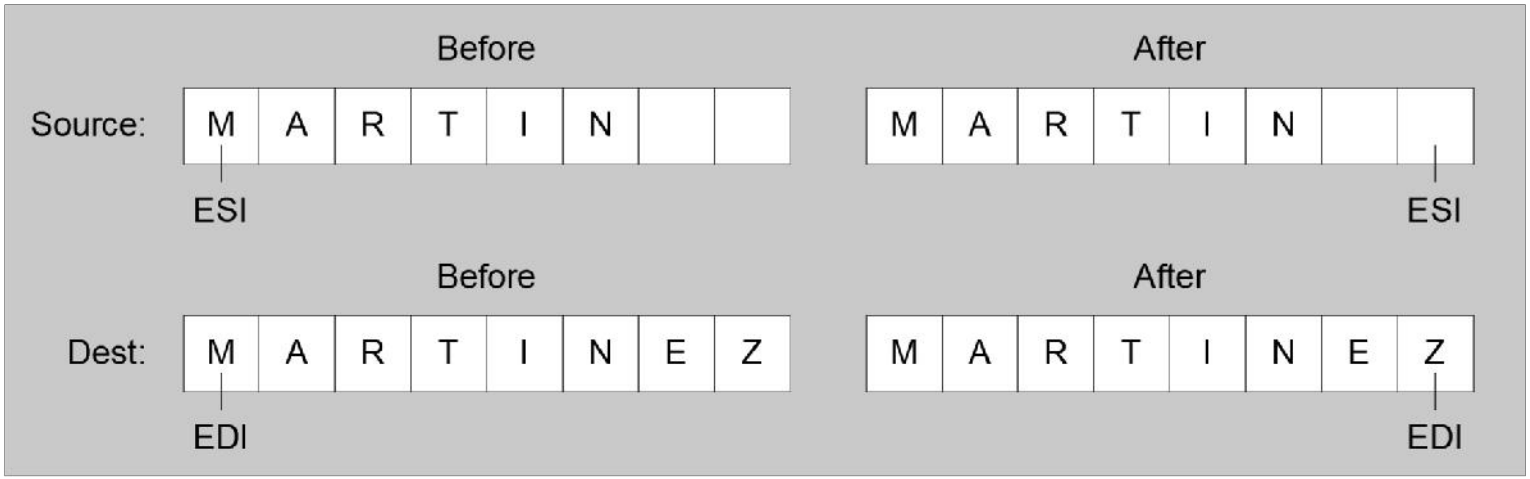
```
Source is smaller
```

Example: Comparing Two Strings (2 of 3)

```
.code
main PROC
    cld                     ; direction = forward
    mov  esi,OFFSET source
    mov  edi,OFFSET dest
    mov  ecx,LENGTHOF source
    repe cmpsb
    jb   source_smaller
    mov  edx,OFFSET str2    ; "source is not smaller"
    jmp  done
source_smaller:
    mov  edx,OFFSET str1    ; "source is smaller"
done:
    call WriteString
    exit
main ENDP
END main
```

Example: Comparing Two Strings (3 of 3)

- The following diagram shows the final values of ESI and EDI after comparing the strings:



SCASB, SCASW, and SCASD

- The SCASB, SCASW, and SCASD instructions compare a value in AL/AX/EAX to a byte, word, or doubleword, respectively, addressed by EDI.
- Useful types of searches:
  - Search for a specific element in a long string or array.
  - Search for the first element that does not match a given value.

SCASB Example

Search for the letter ’F’ in a string named alpha:

```
.data
alpha BYTE "ABCDEFGH",0
.code
mov edi,OFFSET alpha
mov al,'F'                    ; search for 'F'
mov ecx,LENGTHOF alpha
cld
repne scasb                   ; repeat while not equal
jnz quit
dec edi                       ; EDI points to 'F'
```

What is the purpose of the JNZ instruction?

STOSB, STOSW, and STOSD

- The STOSB, STOSW, and STOSD instructions store the contents of AL/AX/EAX, respectively, in memory at the offset pointed to by EDI.
- Example: fill an array with 0FFh

```
.data
Count = 100
string1 BYTE Count DUP(?)
.code
mov al,0FFh                 ; value to be stored
mov edi,OFFSET string1      ; ES:DI points to target
mov ecx,Count               ; character count
cld                         ; direction = forward
rep stosb                   ; fill with contents of AL
```

LODSB, LODSW, and LODSD

- LODSB, LODSW, and LODSD load a byte or word from memory at ESI into AL/AX/EAX, respectively.
- Example:

```
.data
array BYTE 1,2,3,4,5,6,7,8,9
.code
     mov esi,OFFSET array
     mov ecx,LENGTHOF array
     cld
L1:  lodsb                    ; load byte into AL
     or al,30h                ; convert to ASCII
     call WriteChar           ; display it
     loop L1
```

Array Multiplication Example

Multiply each element of a doubleword array by a constant value.

```
.data
array DWORD 1,2,3,4,5,6,7,8,9,10
multiplier DWORD 10
.code
   cld                      ; direction = up
   mov esi,OFFSET array     ; source index
   mov edi,esi              ; destination index
   mov ecx,LENGTHOF array   ; loop counter

L1:lodsd                    ; copy [ESI] into EAX
   mul multiplier           ; multiply by a value
   stosd                    ; store EAX at [EDI]
   loop L1
```

Your turn . . .

- Write a program that converts each unpacked binary-coded decimal byte belonging to an array into an ASCII decimal byte and copies it to a new array.

```
.data
array BYTE 1,2,3,4,5,6,7,8,9
dest  BYTE (LENGTHOF array) DUP(?)
```

```
   mov esi,OFFSET array
   mov edi,OFFSET dest
   mov ecx,LENGTHOF array
   cld
L1:lodsb                    ; load into AL
   or al,30h                ; convert to ASCII
   stosb                    ; store into memory
   loop L1
```