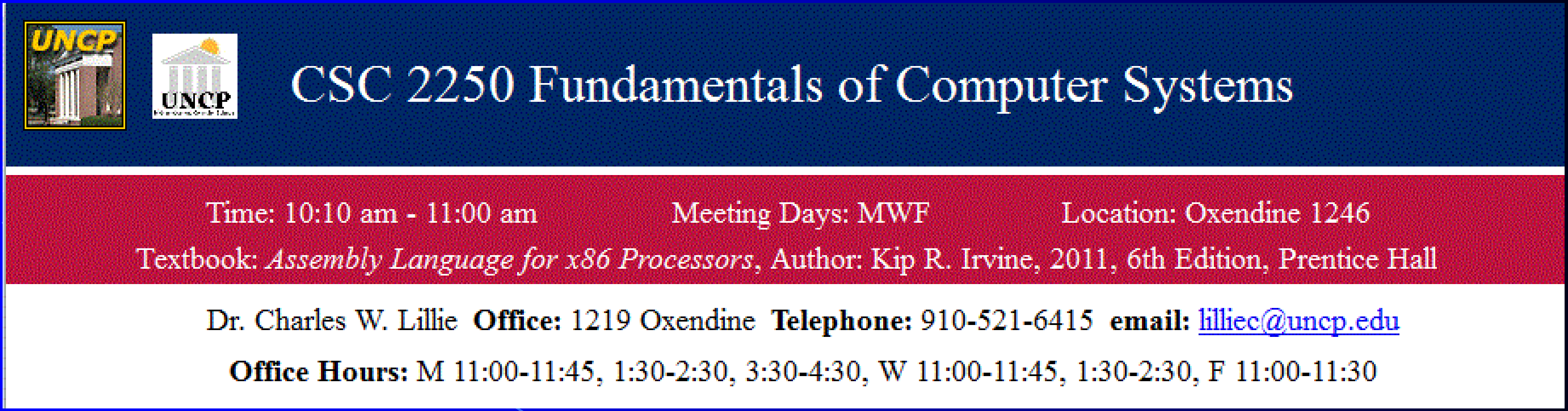


CSC 2250 Fundamentals of Computer Systems

Time: 10:10 am - 11:00 am Meeting Days: MWF Location: Oxendine 1246

Textbook: *Assembly Language for x86 Processors*, Author: Kip R. Irvine, 2011, 6th Edition, Prentice Hall

Dr. Charles W. Lillie **Office:** 1219 Oxendine **Telephone:** 910-521-6415 **email:** lilliec@uncp.edu

Office Hours: M 11:00-11:45, 1:30-2:30, 3:30-4:30, W 11:00-11:45, 1:30-2:30, F 11:00-11:30

Chapter 9: Strings and Arrays

Fall 2012

Chapter Overview

- **String Primitive Instructions**
- Selected String Procedures
- Two-Dimensional Arrays
- Searching and Sorting Integer Arrays

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
```

Irvine, Kip R. Assembly Language for x86 Processors 6/e, 2010.

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
```

Irvine, Kip R. Assembly Language for x86 Processors 6/e, 2010.

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
```

Irvine, Kip R. Assembly Language for x86 Processors 6/e, 2010.

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
```

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:

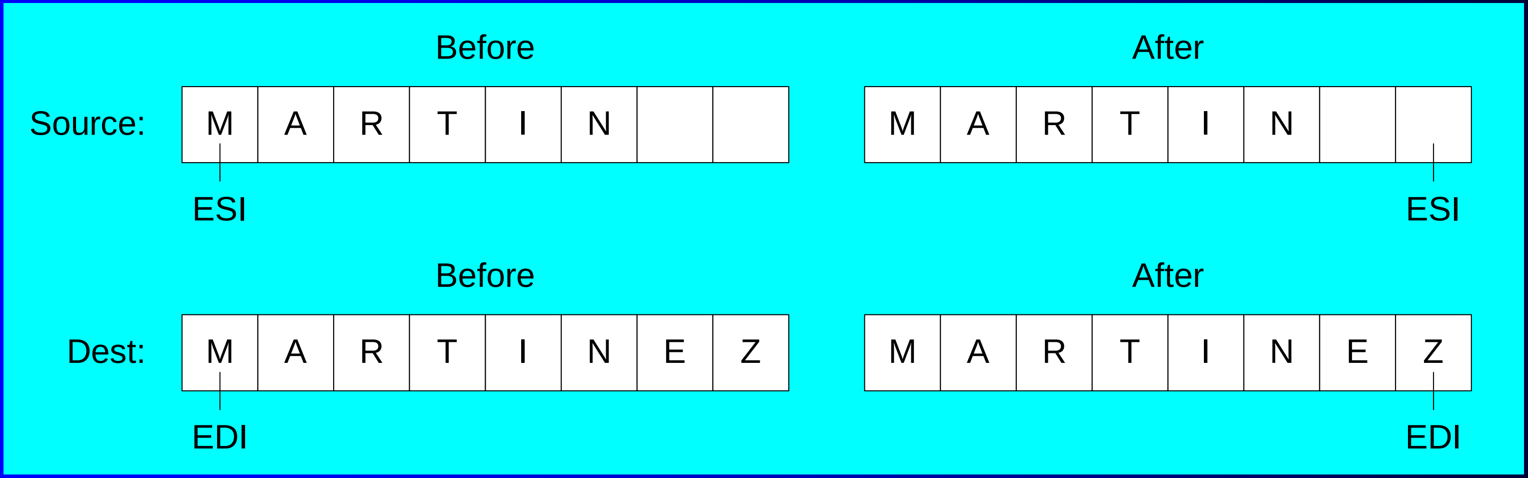
```
Source is smaller
```

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:



Before

Source: M A R T I N

ESI

After

M A R T I N

ESI

Before

Dest: M A R T I N E Z

EDI

After

M A R T I N E Z

EDI

Irvine, Kip R. Assembly Language for x86 Processors 6/e, 2010.

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?

Irvine, Kip R. Assembly Language for x86 Processors 6/e, 2010.

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
```

Irvine, Kip R. Assembly Language for x86 Processors 6/e, 2010.

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
```

Irvine, Kip R. Assembly Language for x86 Processors 6/e, 2010.

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
```

Irvine, Kip R. Assembly Language for x86 Processors 6/e, 2010.

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
```

What's Next

- String Primitive Instructions
- Selected String Procedures
- **Two-Dimensional Arrays**
- Searching and Sorting Integer Arrays

Two-Dimensional Arrays

- Base-Index Operands
- Base-Index Displacement

Irvine, Kip R. Assembly Language for x86 Processors 6/e, 2010.

Base-Index Operand

- A base-index operand adds the values of two registers (called base and index), producing an effective address. Any two 32-bit general-purpose registers may be used. *(Note: esp is not a general-purpose register)*
- Base-index operands are great for accessing arrays of structures. (A structure groups together data under a single name.)

Structure Application

A common application of base-index addressing has to do with addressing arrays of structures (Chapter 10). The following defines a structure named COORD containing X and Y screen coordinates:

```
COORD STRUCT
  X WORD ?          ; offset 00
  Y WORD ?          ; offset 02
COORD ENDS
```

Then we can define an array of COORD objects:

```
.data
setOfCoordinates COORD 10 DUP(<>)
```

Structure Application

The following code loops through the array and displays each Y-coordinate:

```
   mov  ebx,OFFSET setOfCoordinates
   mov  esi,2            ; offset of Y value
   mov  eax,0
L1:mov  ax,[ebx+esi]
   call WriteDec
   add  ebx,SIZEOF COORD
   loop L1
```

Base-Index-Displacement Operand

- A base-index-displacement operand adds base and index registers to a constant, producing an effective address. Any two 32-bit general-purpose register can be used.
- Common formats:

[*base* + *index* + *displacement*]

displacement [*base* + *index*]

Two-Dimensional Table Example

Imagine a table with three rows and five columns. The data can be arranged in any format on the page:

```
table  BYTE  10h,  20h,  30h,  40h,  50h
        BYTE  60h,  70h,  80h,  90h, 0A0h
        BYTE 0B0h, 0C0h, 0D0h, 0E0h, 0F0h
NumCols = 5
```

Alternative format:

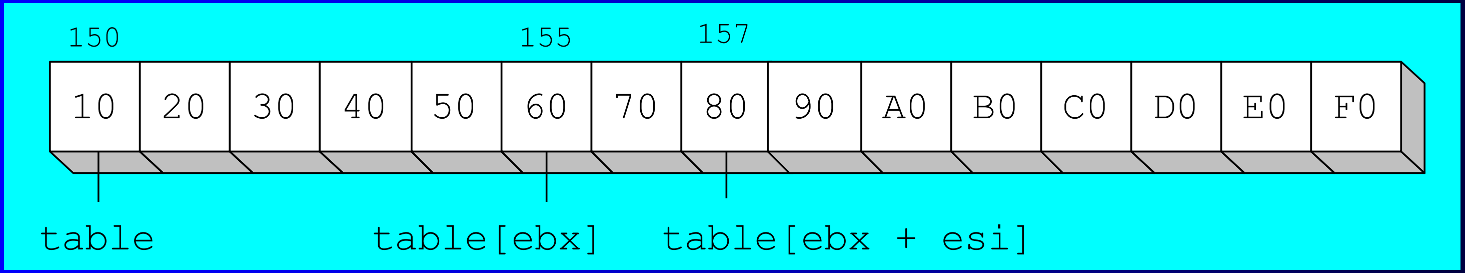
```
table  BYTE  10h,20h,30h,40h,50h,60h,70h,
        80h,90h,0A0h,
        0B0h,0C0h,0D0h,
        0E0h,0F0h
NumCols = 5
```

Two-Dimensional Table Example

The following code loads the table element stored in row 1, column 2:

```
RowNumber = 1
ColumnNumber = 2

mov ebx,NumCols * RowNumber
mov esi,ColumnNumber
mov al,table[ebx + esi]
```



Summary

- String primitives are optimized for efficiency
- Strings and arrays are essentially the same
- Keep code inside loops simple
- Use base-index operands with two-dimensional arrays
- Avoid the bubble sort for large arrays
- Use binary search for large sequentially ordered arrays