Topic 9
Lecture 9b
Strings and Arrays

CSCI 150

Assembly Language / Machine Architecture

Prof. Dominick Atanasio

What's Next (2 of 3)

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

Two-Dimensional Arrays

- Base-Index Operands
- Base-Index Displacement

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.
- REMINDER: *esp is not a general-purpose register*
  - In 64-bit mode, you use 64-bit registers for bases and indexes
- Base-index operands are great for accessing arrays of structures.
- A structure groups together data under a single label.

Structure Application (1 of 2)

A common application of base-index addressing has to do with addressing arrays of structures (Topic 10). In NASM, struc is a preprocessor macro. The following defines a structure named coord containing X and Y screen coordinates:

```
struc coord
 .x resw 1                ; offset 00
 .y resw 1                ; offset 02
endstruc
```

Then we can define an array of coord objects:

```
section .bss
        coords_sz:  equ 5
        coords: resb coord_size * coords_sz
```

Note: The preprocessor creates a constant (equ) called coord_ size

Using the local labels, we can refer to the offset of an element with *coord.x*

Structure Application (2 of 2)

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

```
     mov ebx, coords
     mov ecx, coords_sz
L1:  movzx eax, word [ebx + coord.y]
     call print_int
     add ebx, coord_size
     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 format:

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

Two-Dimensional Table Example (1 of 2)

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

```
table:    db    10h, 20h, 30h, 40h, 50h
          db    60h, 70h, 80h, 90h, 0A0h
          db    0B0h, 0C0h, 0D0h, 0E0h, 0F0h
col_qty: equu   5
```

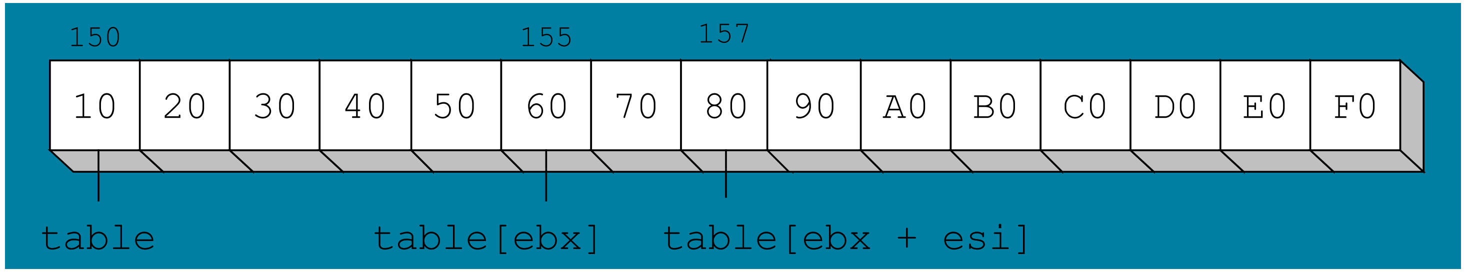
Alternative format:

```
table:      db 10h, 20h, 30h, 40h, 50,
               60h,70h, 80h,90h,0A0h,
               0B0h,0C0h,0D0h,0E0h,0F0h
col_qty: equ   5
```

Two-Dimensional Table Example (2 of 2)

The following 32-bit code loads the table element stored in row 1, column 2

```
; assume esi holds row number
mov eax col_qty
mul  esi
add eax, 2  ; col number
mov al, [table + eax]
```



What's Next (3 of 3)

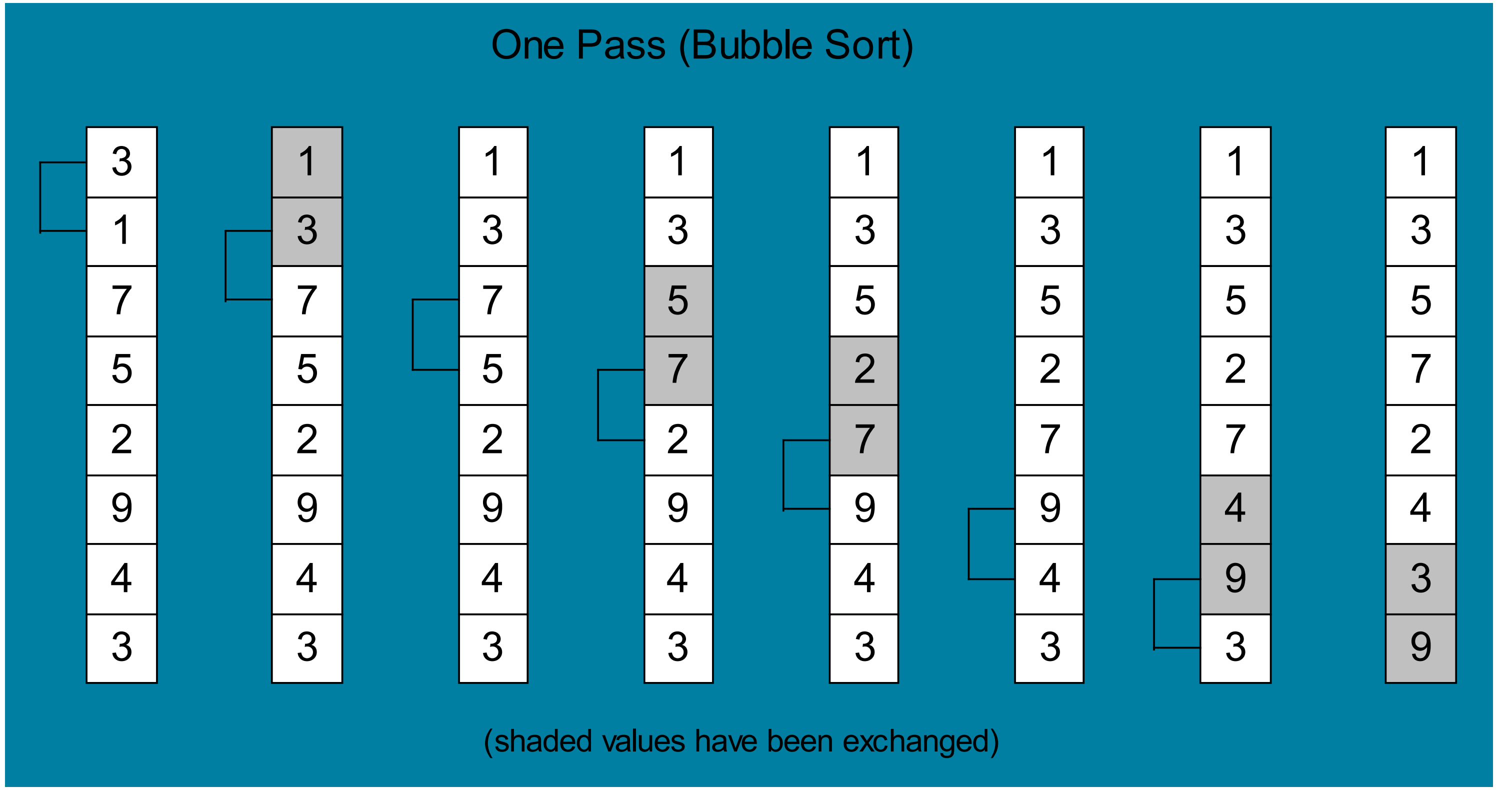
- String Primitive Instructions
- Two-Dimensional Arrays
- **Searching and Sorting Integer Arrays**

Searching and Sorting Integer Arrays

- Bubble Sort
  - A simple sorting algorithm that works well for small arrays. Its speed is indistinguishable from a faster sorting method for small arrays
- Binary Search
  - A simple searching algorithm that works well for large arrays of values that have been placed in either ascending or descending order

Bubble Sort

Each pair of adjacent values is compared, and exchanged if the values are not ordered correctly:



Bubble Sort Pseudocode

N = array size, cx1 = outer loop counter, cx2 = inner loop counter:

```
cx1 = N - 1
while( cx1 > 0 )
{
        esi = addr(array)
        cx2 = cx1
        while( cx2 > 0 )
        {
                if( array[esi] < array[esi+4] )
                   exchange( array[esi], array[esi+4] )
                add esi, 4
                dec cx2
        }
        dec cx1
}
```

Bubble Sort Implementation

Code shown in class

Binary Search

- Searching algorithm, well-suited to large ordered data sets
- Divide and conquer strategy
- Each "guess" divides the list in half
- Classified as an O(log n) algorithm:
  - As the number of array elements increases by a factor of *n*, the average search time increases by a factor of $\log_2 n$.

Sample Binary Search Estimates

Array Size (n)	Maximum Number of Comparisons: $(\log_2 n) + 1$
64	7
1,024	11
65,536	17
1,048,576	21
4,294,967,296	33

Binary Search Pseudocode

```
int bin_search( int values[], int count, const int searchVal )
{
    int first = 0;
    int last = count - 1;
    while( first <= last )
    {
        int mid = (last + first) / 2;
        if( values[mid] < searchVal )
            first = mid + 1;
        else if( values[mid] > searchVal )
            last = mid - 1;
        else
            return mid;    // success
    }

    return -1;             // not found
}
```