Assembly Programming

Chapter 9: Strings and Arrays

CSE3030

Prof. Youngjae Kim

Distributed Computing and Operating Systems Laboratory (DISCOS)

https://discos.sogang.ac.kr

Introduction

- String Primitive Instructions
  - Chapter 9.2 and 9.3
- Two-Dimensional Arrays
  - We will show how to manipulate two-dimensional arrays, using advanced indirect addressing modes: base-index and base-index-displacement.
- Searching and Sorting Integer Arrays
  - Second, we will see how to implement bubble sort and binary search.

Ordering of Rows and Columns

- A two-dimensional array is a high-level abstraction of a one-dimensional array.
- Two mentions to arrange the rows and columns in memory
  - Row-major order
  - Column-major order

Logical Arrangement:

10	20	30	40	50
60	70	80	90	A0
B0	C0	D0	E0	F0

Row-major order

10	20	30	40	50	60	70	80	90	A0	B0	C0	D0	E0	F0

Column-major order

10	60	B0	20	70	C0	30	80	D0	40	90	E0	50	A0	F0

Base-Index Operand

- A base-index operand adds the values of two registers (called *base* and *index*), producing an offset address:

[base + index]

```
tableB BYTE 10h, 20h, 30h, 40h, 50h
Rowsize = ($ - tableB)
    BYTE    60h, 70h, 80h, 90h,0A0h
     BYTE   0B0h,0c0h,0D0h,0E0h,0F0h

row_index = 1
column_index = 2

mov ebx, OFFSET tableB          ; table offset
add ebx, RowSize * row_index    ; row offset
mov esi, column_index
mov al, [ebx + esi]             ; AL = 80h
```

Row-major
row offset -> base register
column offset -> index register

```
;-----------------------------------------------------
; calc_row_sum
; Calculates the sum of a row in a byte matrix.
; Receives: EBX = table offset, EAX = row index,
; ECX = row size, in bytes.
; Returns: EAX holds the sum
;-----------------------------------------------------
 calc_row_sum PROC USES ebx ecx edx esi
        mul ecx             ; row index *row size
        add ebx, eax        ; row offset
        mov eax, 0          ; accumulator
        mov esi, 0          ; column index
 L1:    movzx edx, BYTE PTR[ebx+esi]   ; get a byte
        add   eax, edx                 ; add to accumulator
        inc   esi                      ; next byte in row
        loop L1
 calc_row_sum ENDP
```

BYTE PTR was needed to clarify the operand size in the MOVZ instruction.

Base-Index-Displacement Operands

- A base-index-displacement operand combines a displacement, a base register, and index register, and an optional scale factor to produce an effective address. Format:

- Example

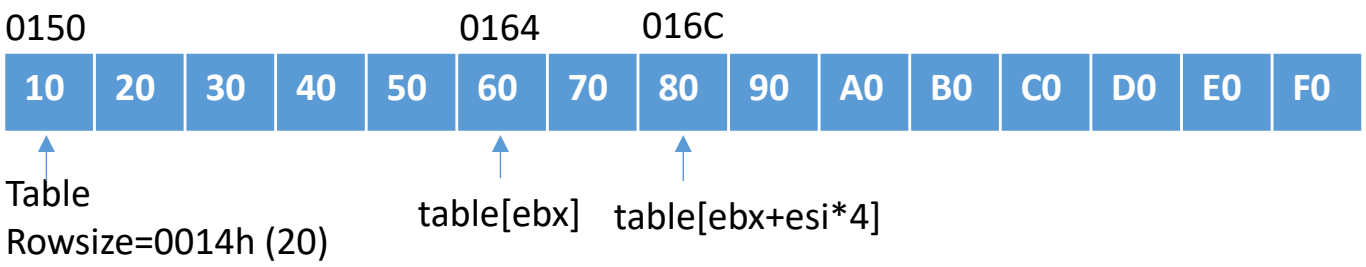
```
[base+index+displacement]
Displacement[base+index]
```

```
tableD DWORD  10h, 20h, 30h, 40h, 50h
Rowsize = ($ - tableD)
       DWORD   60h, 70h, 80h, 90h,0A0h
       DWORD  0B0h,0c0h,0D0h,0E0h,0F0h

mov ebx, Rowsize                    ; row index
mov esi, 2                          ; column index
mov eax, tableD[ebx + esi*TYPE tableD]
```

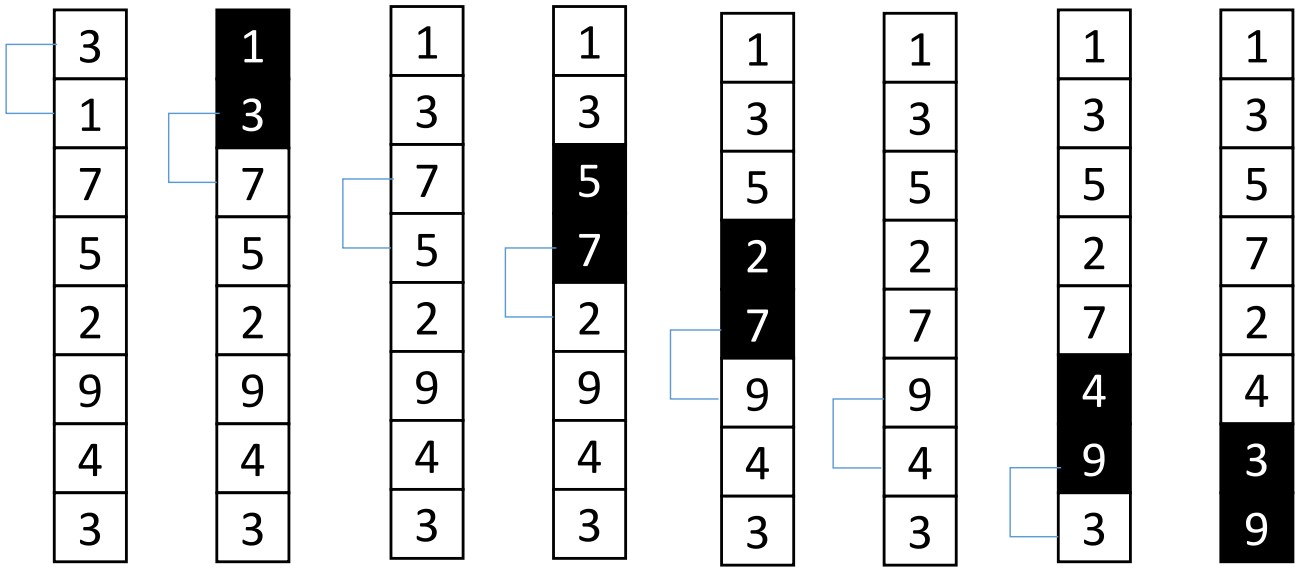
Row offset

Column offset



Sorting Integer Arrays

- Bubble Sort
  - The bubble sort compares pairs of array values, beginning in positions 0 and 1. If compared values are in reverse order, they are exchanged.
  - Time Complexity: O(n^2)



Bubble Sort

- Pseudocode
  - N: The size of the array
  - cx1: the outer loop counter
  - cx2: the inner loop counter

```
cx1 = N-1
while ( cx > 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 c2
  }
  dec cx1
}
```

Bubble Sort (Assembly Language)

```
;-----------------------------------------------------
; BubbleSort
; Sort an array of 32-bit signed integers in ascending
; order, using the bubble sort algorithm.
; Receives: pointer to array, array size
; Returns: nothing
;-----------------------------------------------------
BubbleSort PROC USES eax ecx esi
        pArray:PTR DWORD        ; pointer to array
        Count:DWORD             ; array size

        mov ecx, Count
        dec ecx
L1:     push ecx                ; decrement count by 1
        mov esi, pArray         ; point to first value
L2:     mov eax, [esi]          ; get array value
        cmp [esi+4], eax        ; compare a pair of values
        jg L3                   ; if[ESI]<=[ESI+4], no exchange
        xchg eax, [esi+4]       ; exchange the pair
        mov [esi], eax
L3:     add esi, 4              ; move both pointers forward
        loop L2                 ; inner loop

        pop ecx                 ; retrieve out loop count
        loop L1                 ; else repeat outer loop
L3:     ret
BubbleSort ENDP
```

Searching Integer Array

- Sequential Search
  - For any array of n elements, a sequential search requires an average of n/2 comparisons.
    - If the array consists of more than 1 million elements for example, it will require a more significant amount of processing time.

- Binary Search
  - Effective when searching for a single item in a large array.
  - Precondition: the array elements must be arranged in ascending or descending order.
  - Time Complexity: O(log n)

Binary Search

- C++ implementation of a binary search function

```
int BinSEarch( int values[], const int searchVal, int count)
{
    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] > search =Val)
            last = mid - 1;
         else
            return mid;              // success
    }
    return -1;                       // not found
}
```

```
;-----------------------------------------------------
; BinarySearch
; Searches an array of signed integers for a single value.
; Receives: Pointer to array, array size, search value.
; Return: If a match is found, EAX = the array position of
; the matching element; otherwise, EAX = -1;
;-----------------------------------------------------
BinarySearch PROC USES ebx edx esi edi,
        pArray:PTR DWORD,       ; pointer to array
        Count:DWORD,            ; array size
        searchVal:DWORD,        ; search value
        LOCAL first:DWORD,      ; first position
        last:DWORD,             ; last position
        mid:DWPRD               ; midpoint

        mov first, 0            ; first = 0
        mov eax, Count          ; last = (count-1)
        dec eax
        mov last, eax
        mov edi, searchVal      ; EDI = searchVal
        mov ebx, pArray         ; EBX points to the array
L1:                             ; while first <= last
        mov eax, first
        comp eax, last
        jg L5
                                ; mid = (last+first)/2
        mov eax, last
        add eax, first
        shr eax, 1
        mov mid, eax            ; EDX = values[mid]
```

```
        mov esi, mid
        shl esi, 2              ; scale mid value by 4 (double word)
        mov edx, [ebx+esi]      ; EDX = values[mid] (ebx: array pointer)
                                ; if (EDXX < searchval(EDI)
        cmp edx, edi            ; (edi: search value)
        jge L2
                                ; first = mid + 1
        mov eax, mid
        inc eax,
        mov first, eax
        jmp L4
                                ; else if (EDX > searchVal(EDI)
L2:     cmp edx, edi
        jle L3
                                ; last = mid - 1
        mod eax, mid
        dec eax
        mov last, eax
        jmp L4
                                ; else return mid
L3:     mov eax, mid            ; value found
        jmp L9                  ; return (mid)
L4:     jmp L1                  ; continue the loop
L5:     mov eax, -1             ; search failed
L9:     ret
BinarySearch ENDP
```