بسم اللہ الرّحمٰن الرّحیم

EE213 COMPUTER ORGANIZATION AND ASSEMBLY LANGUAGE

FALL 2018

9

Strings and Arrays

OUTLINES

- String Primitive Instructions
- Two Dimensional Arrays

STRING PRIMITIVE INSTRUCTIONS

Instruction	Description
MOVSB, MOVSW, MOVSD	Move string data: Copy data from memory addressed by ESI to memory addressed by EDI.
CMPSB, CMPSW, CMPSD	Compare strings: Compare the contents of two memory locations addressed by ESI and EDI.
SCASB, SCASW, SCASD	Scan string: Compare the accumulator (AL, AX, or EAX) to the contents of memory addressed by EDI.
STOSB, STOSW, STOSD	Store string data: Store the accumulator contents into memory addressed by EDI.
LODSB, LODSW, LODSD	Load accumulator from string: Load memory addressed by ESI into the accumulator.

- Although they are called *string primitives*, they are not limited to character arrays.
- Each instruction implicitly uses **ESI**, **EDI**, or both registers to address memory.
- String primitives execute efficiently because they automatically repeat and increment array indexes.

MOVSB, MOVSW, AND MOVSD

- 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

MOVSB	Move (copy) bytes
MOVSW	Move (copy) words
MOVSD	Move (copy) doublewords

- Depending upon **Direction Flag, ESI** and **EDI** are automatically incremented or decremented.

Instruction	Value Added or Subtracted from ESI and EDI
MOVSB	1
MOVSW	2
MOVSD	4

USING A REPEAT PREFIX

- By itself, a string primitive instruction processes only a single memory value or pair of values.
- If you add a *repeat prefix*, the instruction repeats, using ECX as a counter.
  - The repeat prefix permits you to process an entire array using a single instruction.

REP	Repeat while ECX > 0
REPZ, REPE	Repeat while the Zero flag is set and ECX > 0
REPNZ, REPNE	Repeat while the Zero flag is clear and ECX > 0

EXAMPLE: COPY A STRING

```
cld                          ; clear direction flag
mov esi,OFFSET string1       ; ESI points to source
mov edi,OFFSET string2       ; EDI points to target
mov ecx,10                   ; set counter to 10
rep movsb                    ; move 10 bytes
```

- If **ECX > 0**, ECX is decremented and the instruction repeats; else the control is passed to the next line in the program.
- ESI and EDI are automatically incremented when MOVSB repeats.

DIRECTION FLAG

- String primitive instructions increment or decrement ESI and EDI based on the state of the Direction flag.

Value of the Direction Flag	Effect on ESI and EDI	Address Sequence
Clear	Incremented	Low-high
Set	Decremented	High-low

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

```
CLD    ; clear Direction flag (forward direction)
STD    ; set Direction flag (reverse direction)
```

EXAMPLE: COPY DOUBLEWORD ARRAY

```
.data
        source DWORD 20 DUP(0FFFFFFFFh)
        target DWORD 20 DUP(?)
.code
        cld
        mov ecx,LENGTHOF source
        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:

```
.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	Compare bytes
CMPSW	Compare words
CMPSD	Compare doublewords

- `Repeat` (**`rep`**, **`repe`**, **`repz`**, **`repne`**, **`repnz`**)can be used with CMPSB, CMPSW, and CMPSD.
- The Direction flag determines the incrementing or decrementing of ESI and EDI.

COMPARING A PAIR OF DOUBLEWORDS

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

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

COMPARING MULTIPLE DOUBLE DOUBLEWORDS

- Clear the Direction flag (forward direction), initialize ECX as a counter, and use a repeat prefix with CMPSD

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

EXAMPLE: COMPARING TWO STRINGS (1 OF 3)

```
.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:

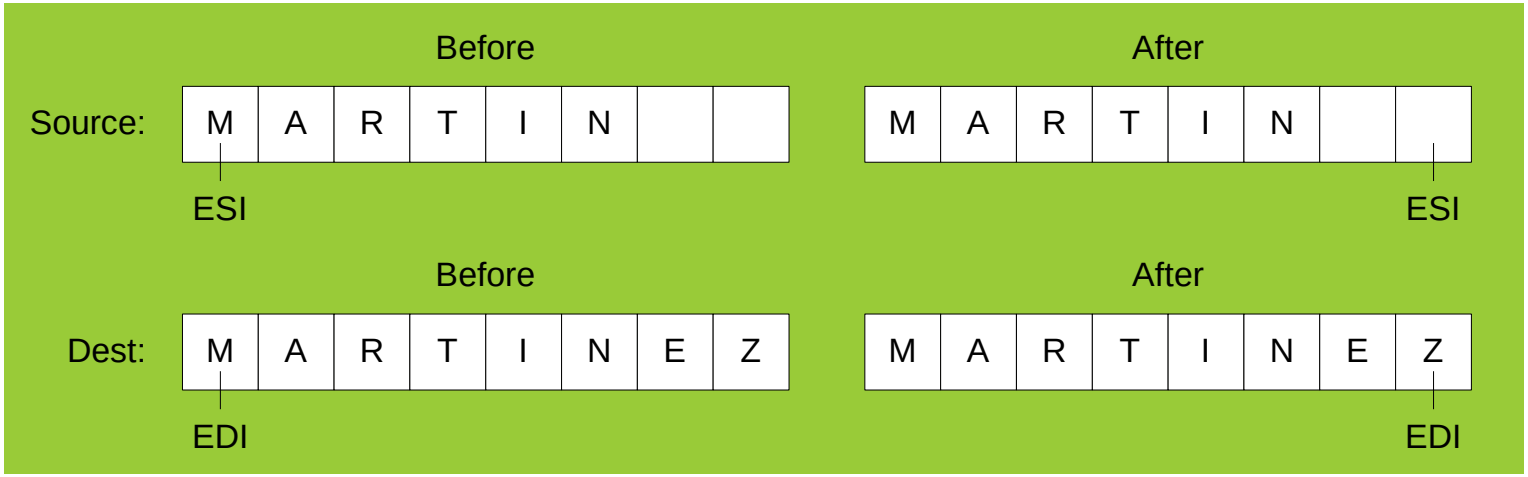
```
Source is smaller
```

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
```

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** instruction compares a value in AL to a byte addressed by EDI.
- **SCASW** instruction compares a value in AX to a word addressed by EDI.
- **SCASD** instruction compares a value in EAX to a doubleword addressed by EDI.
- The Direction flag determines the incrementing or decrementing of EDI.
- The instructions are useful when looking for a single value in a string or array.

EXAMPLE: SCAN FOR A MATCHING CHARACTER

```
.data
alpha BYTE "ABCDEFGH",0
.code
mov edi,OFFSET alpha      ; EDI points to the string
mov al,'F'                ; search for the letter F
mov ecx,LENGTHOF alpha    ; set the search count
Cld                       ; direction = forward
repne scasb               ; repeat while not equal
jnz quit                  ; quit if letter not found
dec edi                   ; found: back up EDI
```

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.
- EDI is incremented or decremented based on the state of the Direction flag.
- When used with the REP prefix, these instructions are useful for filling all elements of a string or array with a single value

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

- The **LODSB**, **LODSW**, and **LODSD** instructions load a byte/word/doubleword from memory at ESI into AL/AX/EAX, respectively.
- ESI is incremented or decremented based on the state of the Direction flag.
- The REP prefix is rarely used with LODS because each new value loaded into the accumulator overwrites its previous contents.
  - Instead, LODS is used to load a single value.

EXAMPLE: ARRAY MULTIPLICATION EXAMPLE

```
.data
  array DWORD 1,2,3,4,5,6,7,8,9,10  ; test data
  multiplier DWORD 10               ; test data
.code
 main PROC
  cld
  mov esi,OFFSET array
  mov edi,esi
  mov ecx,LENGTHOF array
  L1: lodsd                         ; load [ESI] into EAX
  mul multiplier                    ; multiply by a value
  stosd                             ; store EAX into [EDI]
  loop L1
```

TWO DIMENSIONAL ARRAY

- The two methods of arranging the rows and columns in memory: ***row-major order*** and ***column-major*** order.
- If you implement a two-dimensional array in assembly language, you can choose either ordering method.

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

THE TWO OPERAND TYPES

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

   `[base + index]`

   - Any 32-bit general-purpose registers may be used as base and index registers (esp is not a general-purpose register)

2. **Base-Index-Displacement Operands:** A base-index-displacement operand combines a displacement, a base register, an index register, and an optional scale factor to produce an effective address.

   `[base + index + displacement]`

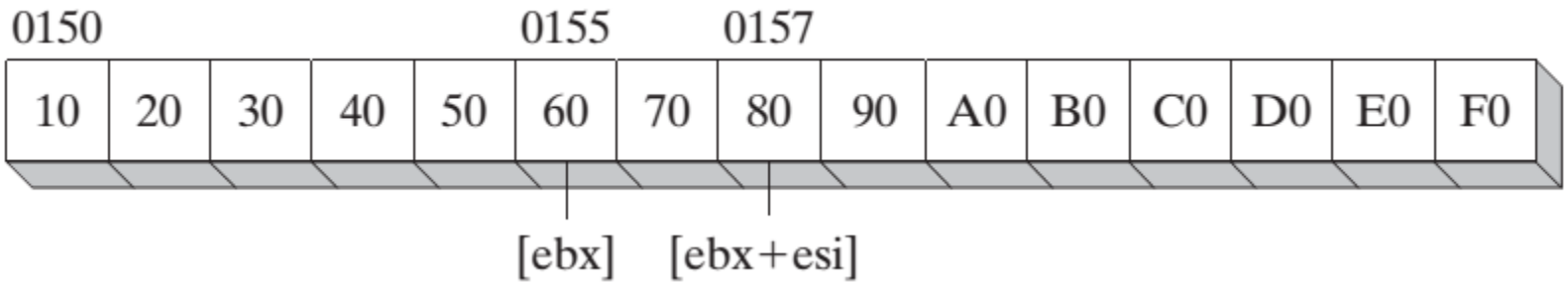
   `displacement[base + index]`

- *Displacement* can be the name of a variable or a constant expression.

BASE-INDEX OPERANDS

```
.data
array WORD 1000h,2000h,3000h
.code
mov   ebx,OFFSET array
mov   esi,2
mov   ax,[ebx+esi]                ; AX = 2000h

mov   edi,OFFSET array
mov   ecx,4
mov   ax,[edi+ecx]                ; AX = 3000h

mov   ebp,OFFSET array
mov   esi,0
mov   ax,[ebp+esi]                ; AX = 1000h
```

```
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
add ebx,RowSize * row_index
mov esi,column_index
mov al,[ebx + esi]
```



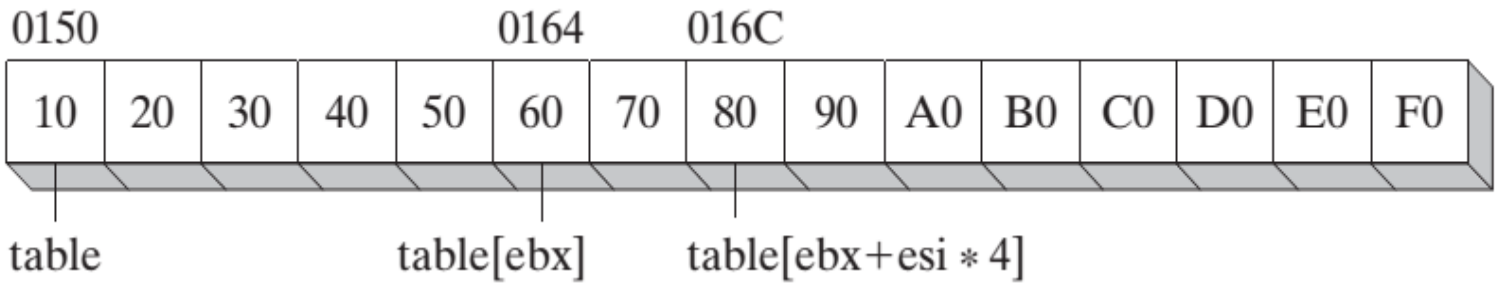
CALCULATING A ROW SUM

```
; 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]
 add eax,edx                ; add to accumulator
 inc esi                    ; next byte in row
 loop L1
 ret
calc_row_sum ENDP
```

Scale Factors: If you're writing code for an array of WORD, multiply the index operand by a scale factor of 2

BASE-INDEX-DISPLACEMENT OPERANDS

```
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]
```



SUMMARY

- String Primitive Instructions
  - MOVSB, MOVSW, MOVSD
  - CMPSB, CMPSW, SMPSD
  - SCASB, SCASW, SCASD
  - STOSB, STOSW, STOSD
  - LODSB, LODSW, LODSD

- Two Dimensional Arrays
  - Base-Index Operands
  - BASE-Index-Displacement Operands