Topic 9
Lecture 9a
Strings and Arrays

CSCI 150

Assembly Language / Machine Architecture

Prof. Dominick Atanasio

Chapter Overview

- **String Primitive Instructions**
- 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

(Move String …) 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.

```
section .bss
        target:   resd 1
Section: .data
        source dd 0FFFFFFFFh
section .text
        mov esi,  source
        mov edi,  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

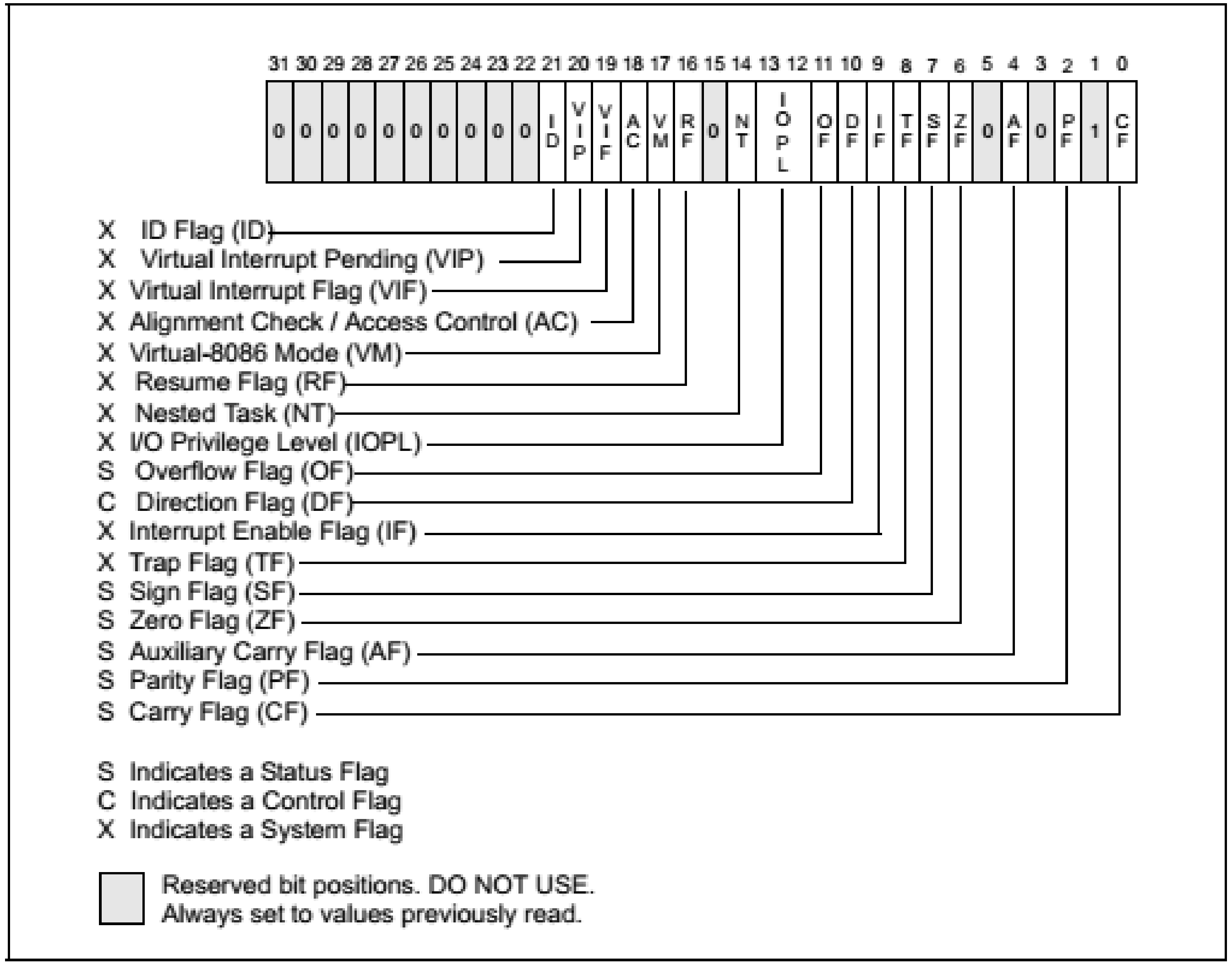
Direction Flag

- 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
```

eFlags Register



31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 ID VIP VIF AC VM RF 0 NT IOPL OF DF IF TF SF ZF 0 AF 0 PF 1 CF
X ID Flag (ID)
X Virtual Interrupt Pending (VIP)
X Virtual Interrupt Flag (VIF)
X Alignment Check / Access Control (AC)
X Virtual-8086 Mode (VM)
X Resume Flag (RF)
X Nested Task (NT)
X I/O Privilege Level (IOPL)
S Overflow Flag (OF)
C Direction Flag (DF)
X Interrupt Enable Flag (IF)
X Trap Flag (TF)
S Sign Flag (SF)
S Zero Flag (ZF)
S Auxiliary Carry Flag (AF)
S Parity Flag (PF)
S Carry Flag (CF)
S Indicates a Status Flag
C Indicates a Control Flag
X Indicates a System Flag
Reserved bit positions. DO NOT USE. Always set to values previously read.

Figure 3-8. EFLAGS Register

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 10 doublewords from source to target

```
section .data
        source: dd 1,2,3,4,5,6,7,8,9,10
section .bss
        target:   resd 10
        len:        equ $ - target
section .text
        cld                                         ; direction = forward
        mov ecx, len                                ; set REP counter
        mov esi, source
        mov edi, target
        rep movsd
```

Your turn . . . (1 of 3)

- 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: dd 1,1,2,3,4,5,6,7,8,9,10
```

```
section .data
        array: dd 1,1,2,3,4,5,6,7,8,9,10
        len:     equ ($ - array) / 4
section .text
        cld
        mov ecx,  len - 1
        lea   esi, [array+4]
        mov edi, 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 (Compare String Byte)
  - CMPSW compares words (Compare String Word)
  - CMPSD compares doublewords (Compare String DWord)
- 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

```
section .data
        source: dd 1234h
        target:   dd 5678h

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

Your turn . . . (2 of 3)

- 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. Assume there are two arrays called *source* and *target* with a constant called *len* that is their size.

```
section .text
    mov ecx, len                ; repetition count
    mov esi, source
    mov edi, 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.

```
section .data
        source: db “MARTIN”
        src_len: equ $ - source
        dest:     db “MARTINEZ”
        str1:      db “Source is smaller”, 0ah, 0
        str2:      db “Source is not smaller”, 0ah, 0
```

Example: Comparing Two Strings (2 of 3)

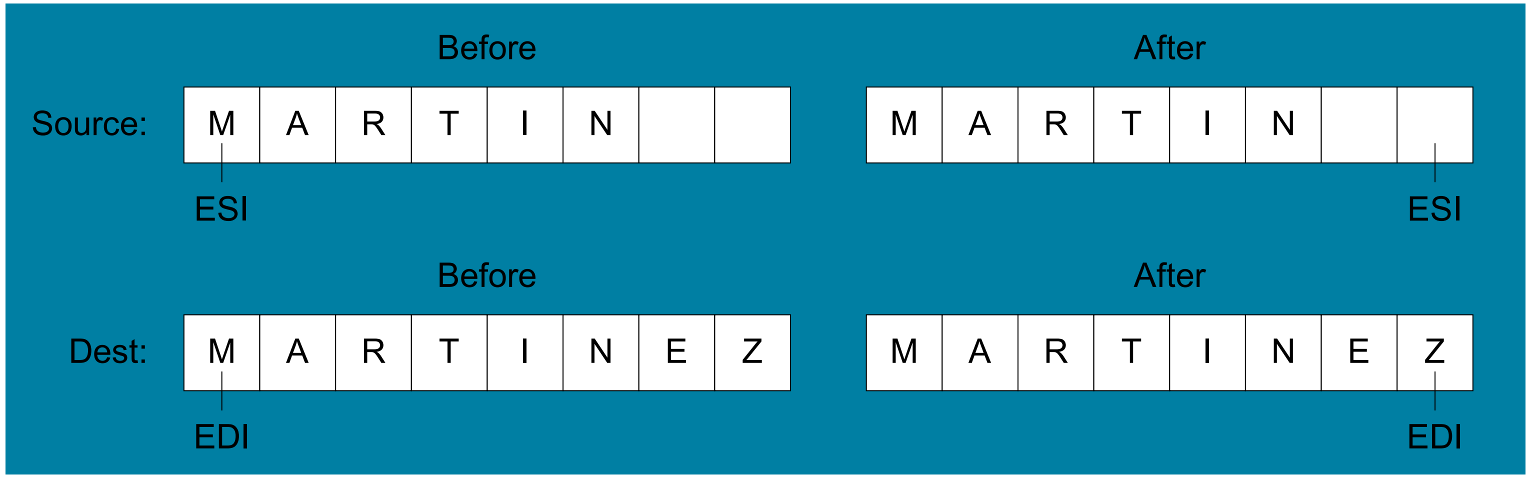
Screen Output

```
Source is smaller
```

```
section .text
_start:
        cld                                   ; direction = forward
        mov esi, source
        mov edi, dest
        mov ecx, src_len
        repe cmpsb
        jb source_smaller
        mov edx, str2                         ; “source is not smaller”
        jmp done
source_smaller:
        mov edx, str1                         ; “source is smaller”
done:
        call print_string2
        exit
…
```

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 (Scan String as …)

- 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:

```
section .data
        alpha: db "ABCDEFGH",0
        alpha_sz: equ $ - alpha
section .text
        mov edi, alpha
        mov al, 'F'                         ; search for 'F’
        mov ecx, alpha_sz
        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

```
section .bss
        count equ 100
        string1: resb count
section .text
        mov al,0FFh                          ; value to be stored
        mov edi, 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:

```
section .data
array: db 1,2,3,4,5,6,7,8,9
array_sz: $ - array
section .text
        mov esi, array
        mov ecx, array
        cld
L1:     lodsb                   ; load byte into AL
        or al,30h               ; convert to ASCII
        push eax
        call print_char         ; display it
        add esp, 4
        loop L1
```

Array Multiplication Example

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

```
section .data
array dd 1,2,3,4,5,6,7,8,9,10
array_sz: $ - array
multiplier dd 10
section .text
        cld                         ; direction = up
        mov esi, array              ; source index
        mov edi, esi                ; destination index
        mov ecx, array_sz           ; loop counter

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

Your turn . . . (3 of 3)

- 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.

```
section .data
      array: db 1,2,3,4,5,6,7,8,9
      array_sz: equ $ - array
section .bss
      dest: resb array_sz
```

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

45 6E 64 65