Lab 9: *String Handling Instructions*	**Session:** Fall 2019
Instructor(s): Sumaiyah Zahid & Fahim Ahmed	

Primitive String Instructions

- The x86 instruction set has five groups of instructions for processing arrays of bytes, words, and double words.
- Each instruction implicitly uses ESI, EDI, or both registers to address memory.
- References to the accumulator imply the use of AL, AX, or EAX, depending on the instruction data size. String primitives execute efficiently because they automatically repeat and increment array indexes.

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.

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. The two registers are either incremented or decremented automatically (based on the value of the Direction flag).

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

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. The following repeat prefixes are used:

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

In the following example, MOVSB moves 10 bytes from string1 to string2. The repeat prefix first tests ECX> 0 before executing the MOVSB instruction. If ECX = 0, the instruction is ignored and control passes to the next line in the program. If ECX > 0, ECX is decremented and the instruction repeats:

```
.data
string1 BYTE 'this is first string',0
string2 BYTE 'this is second string',0
.code
cld                             ; clear direction flag
mov esi,OFFSET string1          ; ESI points to source
mov edi,OFFSET string2          ; EDI points to target
mov ecx,sizeof string1          ; set counter to 15
rep movsb                       ; move bytes
mov edx,offset string2
call writestring
```

EDI are automatically incremented when MOVSB repeats. This behavior is controlled by the CPU's Direction flag.

Direction Flag

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

Direction Flag Usage in String Primitive Instructions.

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

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: You can use a repeat prefix with CMPSB, CMPSW, and CMPSD. The Direction flag determines the incrementing or decrementing of ESI and EDI.

CMPSB	Compare bytes
CMPSW	Compare words
CMPSD	Compare doublewords

Example: Comparing Doublewords

Suppose you want to compare a pair of double words using CMPSD. In the following example, source has a smaller value than target , so the JA instruction will not jump to label L1.

```
.data
        greater BYTE 'source > target',0
        lessOrEqual BYTE 'source < target',0
        source BYTE 'abcd',0
        target BYTE 'abc',0
.code
        mov esi,OFFSET source
        mov edi,OFFSET target
        cmpsd                       ; compare doublewords
        ja L1                       ; jump if source > target
        mov edx,offset lessOrEqual  ;else print source <= target
        call writestring
        jmp endd
L1:
        mov edx,offset greater
        call writestring
endd:
```

SCASB, SCASW and SCASD

The SCASB, SCASW, and SCASD instructions compare a value in AL/AX/EAX to a byte, word, or double word, respectively, addressed by EDI. The instructions are useful when looking for a single value in a string or array. Combined with the REPE (or REPZ) prefix, the string or array is scanned while ECX > 0 and the value in AL/ AX/ EAX match each subsequent value in memory. The REPNE prefix scans until either AL/AX/EAX matches a value in memory or ECX = 0.

Example: Scan for Matching Character

In the following example we search the string alpha, looking for the letter F. If the letter is found, EDI points one position beyond the matching character. If the letter is not found, JNZ exits:

```
.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
quit:
        mov al,[edi]
        call writechar
```

JNZ was added after the loop to test for the possibility that the loop stopped because ECX = 0 and the character in AL was not found.

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. For example, the following code initializes each byte in string1 to 0FFh:

Example:

```
.data
        Count = 100
        string1 BYTE Count DUP(?)
.code
        mov al,0FFh                 ; value to be stored
        mov edi,OFFSET string1      ; EDI 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 or word 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. In the next example, LODSB substitutes for the following two instructions (assuming the Direction flag is clear):

Example: Array multiplication

The following program multiplies each element of a doubleword array by a constant value. LODSD and STOSD work together:

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

String Procedures

STR_CPY

The Str_copy procedure copies a null-terminated string from a source location to a target location.

Syntax: INVOKE Str_copy, ADDR source, ADDR target

STR_LENGTH

The Str_length procedure returns the length of a string in the EAX register. When you call it, pass the string's offset.

Syntax: INVOKE Str_length, ADDR myString

STR_ COMPARE

The Str_compare procedure compares two strings. It affects the CF and ZF as shown in the following table.

Syntax: INVOKE Str_compare, ADDR string1, ADDR string2

Relation	Carry Flag	Zero Flag	Branch If True
string1 < string2	1	0	JB
string1 = string2	0	1	JE
string1 > string2	0	0	JA

Activity:

1. Create a procedure named Scan_String to find the index of the first occurrence of the character '#' in the given string.
    i. Str1 BYTE '127&j~3#^&*#*#45^',0
2. Modify the above procedure to take offset of string1 and the character to be searched as argument.
3. Create IsCompare procedure to compare two strings.
4. Create Move procedure to perform move operation.
5. Create a Str_Reverse procedure to reverse strings.
6. Create a procedure that Loads an array of integer by multiplying it with 3. Load(offset array, byte no)
7. Rewrite the binary search procedure shown in this chapter by using registers for mid, first, and last. Add comments to clarify the registers' usage.
8. Write the procedure to get_frequency Find the frequency of characters:

```
.data
        target BYTE "AAEBDCFBBC",0
        freqTable DWORD 256 DUP(0)
 .code
        INVOKE Get_frequencies, ADDR target, ADDR freqTable
```

Target string:	A	A	E	B	D	C	F	B	B	C	0
ASCII code:	41	41	45	42	44	43	46	42	42	43	0

Frequency table:	2	3	2	1	1	1	0	0	0	0	0	
Index:	41	42	43	44	45	46	47	48	49	4A	4B	etc.