String instructions:

**String** - a byte or word array.

**Operations** that can be performed with string instructions:

* copy a string into another string
* search a string for a particular byte or word
* store characters in a string
* compare strings of characters alphanumerically

**String Primitive Instructions:**

The x86 instruction set has five groups of instructions for processing arrays of bytes, words, and doublewords. Although they are called *string primitives.*

String primitives execute efficiently because they automatically repeat and increment array indexes.



**Direction Flag**

* one of 8086 processor control flags.
* controls the direction of string operations:
* DF = 0 => forward (left to right) processing
* DF = 1 => backward (right to left) processing

**CLD** - clears the DF; sets DF = 0

**STD** - sets DF = 1

**Moving (Copying) Strings**

Instructions:

**MOVSB** - copies contents of BYTE given by **DS:SI** into **ES:DI**  
    **MOVSW** - copies contents of WORD given by **DS:SI** into **ES:DI  
    MOVSD** - copies contenst of DOUBLE WORD given by **DS:SI** into **ES:DI**

Notes:

the string instructions use **DS:SI** as the Source string and **ES:DI** as the destination string. (For the string instructions, DI references an offset in the ES by default).

MOVSD requires the target processor be at least .386 (32-bit processor)

**EXAMPLE**:

INCLUDE IRVINE32.INC

.DATA

STRING1 DB 'HELLO'

STRING2 DB 5 DUP (0)

.CODE

main proc

mov edx,offset STRING1

call writestring

call crlf

LEA SI, STRING1 ; source

LEA DI, STRING2 ; destination

CLD ; DF = 0

MOVSB ; mov 1st byte

MOVSB ; mov 2nd byte

mov edx,offset STRING1

call writestring

call crlf

mov edx,offset STRING2

call writestring

call crlf

exit

main endp

END MAIN

**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 following repeat prefixes are used:



Note: MOVSB moves only 1 byte at a time. Set CX = count and use the REP prefix to move a specified number of bytes.

**Example:**

       INCLUDE IRVINE32.INC

.DATA

STRING1 DB 'HELLO'

STRING2 DB 5 DUP (0)

.CODE

main proc

mov edx,offset STRING1

call writestring

call crlf

LEA SI, STRING1 ; source

LEA DI, STRING2 ; destination

CLD ; DF = 0

mov ecx,5

rep MOVSB

mov edx,offset STRING1

call writestring

call crlf

mov edx,offset STRING2

call writestring

call crlf

exit

main endp

END MAIN

**Using stack**

**Example:**

        INCLUDE IRVINE32.INC

.DATA

STRING1 DB 'HELLO'

STRING2 DB 5 DUP (0)

.CODE

main proc

mov edx,offset STRING1

call writestring

call crlf

CLD ; forward direction

LEA SI, STRING1 ; set SI to source

LEA DI, STRING2 ; set DI to destination

MOV eCX, 5

REP MOVSB ; copies 5 chars

mov edx,offset STRING1

call writestring

call crlf

mov edx,offset STRING2

call writestring

call crlf

exit

main endp

END MAIN

**Reverse:**

 mov edx,offset STRING1

call writestring

call crlf

std ; forward direction

LEA SI, STRING1 + 4 ; set SI to source

LEA DI, STRING2 +4 ; set DI to destination

MOV eCX, 5

REP MOVSB ; copies 5 chars

mov edx,offset STRING1

call writestring

call crlf

mov edx,offset STRING2

call writestring

call crlf

Note: both SI and DI are decremented by one for each byte that is copied from SI to DI (in the reverse direction).

* MOVSW works the same way as MOVSB and moves one word (2 bytes) at a time. Consequently, SI/DI will be incremented or decremented by 2 bytes for each word copied.
* MOVSD moves one double word (4 bytes) at a time.   Consequently, SI and DI will be incremented or decremented by 4 for each word copied.
* Note: take reverse byte ordering into account when moving WORD or DOUBLE WORD strings.

**Storing Strings**

Instructions:

**STOSB** - copies contents of AL to BYTE address given by ES:DI.  DI is incremented/decremented by 1.  
    **STOSW** - copies the contents of AX to the WORD address given by ES:DI.   DI is incremented/decremented by 2.  
    **STOSD** - copies contents of EAX to the DOUBLE WORD address given by ES:DI.  DI is incremented/decremented by 4.

**Example**:

mov edx,offset STRING1

call writestring

call crlf

LEA DI, STRING1 ; assume BYTE string

CLD

MOV AL, 'A'

STOSB ; store 1st byte of A

STOSB ; store 2nd byte of A

mov edx,offset STRING1

call writestring

call crlf

**Load String**

Instructions:

**LODSB** - moves the BYTE at address DS:SI into AL. SI is incremented/decremented by 1.  
    **LODSW** - moves the WORD at address DS: SI into AX. SI is incremented/decremented by 2.  
    **LODSD** - moves the DOUBLE WORD at address DS:SI into EAX. SI is incremented/decremented by 4.

Loads two bytes from STRING1 into AL (second byte overwrites the first).

Example:

mov edx,offset STRING1

call writestring

call crlf

LEA SI, STRING1

CLD

LODSB

call writechar

call crlf

LODSB

call writechar

call crlf

**Scan String**

Instructions:

**SCASB** - compares **BYTE** at **ES:DI** with **AL** and sets flags according to result.  
    **SCASW** - compares **WORD** at **ES:DI** with **AX** and sets flags.  
    **SCASD** - compares **DOUBLE WORD** at **ES:DI** with **EAX** and sets flags.

Note: when the target ("L") is found, ZF = 1 and DI points to the byte following the target since DI is automatically incremented by SCASB.

Also, set CX = count and use:

        REPNE SCASB  
        REPNZ SCASB

to repeat the scan until the target byte is found, or until the entire string has been searched (i.e., CX = 0).

Example:

mov edx,offset STRING1

call writestring

call crlf

CLD ; left to right

LEA DI, STRING1

MOV AL, 'L' ; target character

MOV ECX,5

REPNZ SCASB

MOV EAX,ECX

CALL WRITEDEC

CALL CRLF

mov edx,offset STRING1

call writestring

call crlf

**Compare String**

Instructions:

**CMPSB** - compares BYTE at ES:DI with BYTE at DS:SI and sets flags.  
        **CMPSW** - compares WORD at ES:DI with WORD at DS:SI and sets flags.  
        **CMPSD** - compares DOUBLE WORD at ES:DI with WORD at DS:SI and sets flags.

Increments (or decrements) each string pointer and successively compares bytes until there is a mismatch between the bytes being compared, or until CX = 0.

CMPSB can be used to determine whether two strings match, or whether one string is a substring of another string.

Example:

INCLUDE Irvine32.inc

.data

greater BYTE 'source > target',0

lessOrEqual BYTE 'source <target',0

source BYTE 'abcd',0

target BYTE 'abc',0

.code

main PROC

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:

exit

main ENDP

END main

EXIT

main ENDP

END main

**String Procedures**

**1. 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

**2. 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

EXAMPLE:

INCLUDE Irvine32.inc

.data

STR1 BYTE "ABCDEF",0

.code

main PROC

INVOKE STR\_LENGTH,ADDR STR1

CALL DUMPREGS ;EAX=7

exit

main ENDP

END main

EXIT

**3. 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



**4. STR\_CPY**

REMOVE ALL OCCURENCES OF A SELECTED CHARACTER FROM A NULL TERMINATED STRING.

.data

STR1 BYTE "MY STRING ###",0

.code

main PROC

INVOKE STR\_TRIM,ADDR STR1,'#'

MOV EDX,OFFSET STR1

CALL WRITESTRING

exit

main ENDP

END main

EXIT

**Exercise**

1. Create a procedure named *Scan\_String* to find the index of the first occurrence of the character ‘#’ in the given string.

**Str1 BYTE ‘127&j~3#^&\*#\*#45^’,0**

1. Modify the above procedure to take *offset of string1* and the *character to be searched* as argument.
2. Create *IsCompare* procedure to compare two strings.
3. Create *Move* procedure to perform move operation.
4. Create a Str\_Reverse procedure to reverse strings.
5. Create a procedure that Loads an array of integer by multiplying it with 3. Load(offset array, byte no)
6. Rewrite the binary search procedure shown in this chapter by using registers for mid, ﬁrst, and last. Add comments to clarify the registers’ usage.
7. 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

