String Operations

==========

String Primitive Instructions

==========

String primitive instructions are a set of instructions that are designed to manipulate strings of data. They are particularly useful for moving, comparing, loading, and storing blocks of data.

The x86 instruction set has five groups of string primitive instructions:

- **Move string data: MOVSB, MOVSW, MOVSD**
- **Compare strings: CMPSB, CMPSW, CMPSD**
- **Scan string: SCASB, SCASW, SCASD**
- **Store string data: STOSB, STOSW, STOSD**
- **Load accumulator from string: LODSB, LODSW, LODSD Image Description**

The image below shows a table of string primitive instructions, including their descriptions and mnemonics.

Table 9-1 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.

Using a Repeat Prefix

By itself, a string primitive instruction only processes a single memory value or pair of values.

However, you can use a repeat prefix to cause the instruction to repeat until a certain condition is met.

The most common repeat prefix is REP, which causes the instruction to repeat while the ECX register is greater than zero.

For example, the following code would copy 10 bytes from the string1 buffer to the string2 buffer:

```
001 cld ; clear Direction flag
002 mov esi, OFFSET string1 ; ESI points to source
003 mov edi, OFFSET string2 ; EDI points to target
004 mov ecx, 10 ; set counter to 10
005 rep movsb ; move 10 bytes
```

Direction Flag

String primitive instructions increment or decrement the ESI and EDI registers based on the state of the Direction flag. The Direction flag can be explicitly modified using the CLD and STD instructions:

CLD clears the Direction flag, causing the ESI and EDI registers to increment.

STD sets the Direction flag, causing the ESI and EDI registers to decrement.

TABLE 9-2 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

When the Direction flag is clear, the ESI and EDI registers are incremented after each operation. This means that the string operation will move from a lower address to a higher address.

When the Direction flag is set, the ESI and EDI registers are decremented after each operation. This means that the string operation will move from a higher address to a lower address.

It is important to set the Direction flag correctly before using a string primitive instruction, as otherwise the ESI and EDI registers may not increment or decrement as intended.

For example, if you want to copy a string from one buffer to another, you would need to clear the Direction flag first. Otherwise, the ESI and EDI registers would be decremented after each operation, and the string would be copied in reverse order.

It is important to set the Direction flag correctly before using a string primitive instruction, as otherwise the ESI and EDI registers may not increment or decrement as intended.

The following example shows how to use the MOVSB instruction to copy a string from one buffer to another:

```
009 ; Copy string1 to string2
010 cld ; clear Direction flag
011 mov esi, OFFSET string1 ; ESI points to source
012 mov edi, OFFSET string2 ; EDI points to target
013 mov ecx, 10 ; set counter to 10
014 rep movsb ; move 10 bytes
```

MOVSB, MOVSW, and MOVSD

The MOVSB, MOVSW, and MOVSD instructions are used to copy data from one memory location to another. They differ in the size of the data that they copy:

- **MOVSB copies bytes.**
- **MOVSW copies words.**

- **MOVSD copies doublewords.**

All three instructions use the ESI and EDI registers to address the source and destination memory locations, respectively.

The Direction flag determines whether the ESI and EDI registers are incremented or decremented after each operation.

The following code shows how to use the MOVSD instruction to copy a doubleword array from one buffer to another:

```
026 ; Copy source to target
027 cld ; clear Direction flag
028 mov ecx, LENGTHOF source ; set REP counter
029 mov esi, OFFSET source ; ESI points to source
030 mov edi, OFFSET target ; EDI points to target
031 rep movsd ; copy doublewords
```

The **CLD instruction** clears the Direction flag, causing the ESI and EDI registers to be incremented after each operation.

The **REP prefix** causes the MOVSD instruction to repeat while the ECX register is greater than zero.

After the code is executed, the ESI and EDI registers will point one position (4 bytes) beyond the end of each array.

This is because the MOVSD instruction copies 4 bytes of data each time it executes.

The notes you provided are unclear and hard to understand because they are missing some key information.

For example, the notes do not explain what the LENGTHOF macro is or how it is used. The notes also do not explain why the ESI and EDI registers are incremented or decremented after each operation.

CMPSB, CMPSW, and CMPSD

The CMPSB, CMPSW, and CMPSD instructions are used to compare two

memory operands. They differ in the size of the operands that they compare:

- **CMPSB compares bytes**
- **CMPSW compares words**
- **CMPSD compares doublewords**

All three instructions use the ESI and EDI registers to address the source and destination operands, respectively.

The Direction flag determines whether the ESI and EDI registers are incremented or decremented after each operation.

Example: Comparing Doublewords:

```
035 ; Compare source and target
036 mov esi, OFFSET source
037 mov edi, OFFSET target
038 cmpsd
039 ; compare doublewords
```

If the two doubleword values are equal, the Zero flag is set. If the source doubleword is greater than the target doubleword, the Carry flag is set. Otherwise, the Carry flag is cleared.

To compare multiple doublewords, you can use a repeat prefix with the CMPSD instruction:

```
042 mov esi, OFFSET source
043 mov edi, OFFSET target
044 cld ; clear Direction flag
045 mov ecx, LENGTHOF source ; repetition counter
046 repe cmpsd ; repeat while equal
```

REPE (Repeat While Equal). It's used to indicate that a repetitive operation should continue while a certain condition holds, in this case, while the comparison with CMPSD yields equality.

The REPE prefix repeats the comparison until ECX equals zero or a pair of doublewords is found to be different.

The notes you provided are missing some key information, such as an explanation of the LENGTHOF macro and the Direction flag.

The notes also do not provide a complete example of how to compare multiple doublewords using a repeat prefix.

The CMPSB, CMPSW, and CMPSD instructions can be used to implement a variety of string operations, such as searching for a specific character in a string or comparing two strings.

For example, the following code shows how to use the CMPSB instruction to search for the letter 'A' in a string:

```
050 mov esi, OFFSET string
051 mov edi, OFFSET 'A'
052 repe cmpsb ; repeat while equal
053 jne not_found ; jump if not equal
```

If the letter 'A' is found in the string, the jne instruction will jump to the not_found label. Otherwise, the jne instruction will not jump.

The CMPSB, CMPSW, and CMPSD instructions are powerful tools for comparing memory operands.

SCASB, SCASW, and SCASD

The SCASB, SCASW, and SCASD instructions compare the value in the AL, AX, or EAX register, respectively, to the byte, word, or doubleword, respectively, addressed by the EDI register.

These instructions are useful for searching 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 the AL, AX, or EAX register matches each subsequent value in memory.

The REPNE prefix scans until either the value in the AL, AX, or EAX register matches a value in memory or ECX = 0.

Example: Scan for a Matching Character. The following code shows how to use the SCASB instruction to search for the letter 'F' in the string alpha:

```
.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

    ; Test if the loop stopped because ECX = 0 and the character in AL was not found
    jnz quit

    ; Found: back up EDI
    dec edi

    ; Otherwise, the letter F was found
```

The REPNE prefix causes the SCASB instruction to repeat while the Zero flag is cleared and ECX is greater than zero.

The Zero flag is cleared when the value in the AL register does not match the byte at the memory address pointed to by the EDI register.

The jnz instruction jumps to the quit label if the Zero flag is set, meaning that the letter 'F' was not found.

Otherwise, the letter 'F' was found and the dec edi instruction is executed to back up the EDI register one position.

The jnz instruction is needed to test for the possibility that the loop stopped because ECX = 0 and the character in AL was not found.

If the loop stopped because ECX = 0, then the Zero flag will be set, and the jnz instruction will jump to the quit label.

Otherwise, the letter 'F' was found and the dec edi instruction is executed to back up the EDI register one position.

The SCASB, SCASW, and SCASD instructions are powerful tools for searching for a single value in a string or array.

STOSB, STOSW, and STOSD

The STOSB, STOSW, and STOSD instructions store the contents of the AL, AX, or EAX register, respectively, in memory at the offset pointed to by the EDI register.

- **STOSB: Store byte.**
- **STOSW: Store word.**
- **STOSD: Store doubleword.**

The EDI register 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: Initialize an Array. The following code shows how to use the STOSB instruction to initialize each byte in the string1 array to 0xFFh:

```
080 ; Initialize each byte in string1 to 0xFFh
081
082 .data
083     Count = 100
084     string1 BYTE Count DUP(?)
085 .code
086     mov al, 0xFFh ; value to be stored
087     mov edi, OFFSET string1 ; EDI points to target
088     mov ecx, Count ; character count
089     cld ; direction = forward
090     rep stosb ; fill with contents of AL
```

The **REP prefix** causes the STOSB instruction to repeat while ECX is greater than zero.

The **CLD instruction** clears the Direction flag, causing the EDI register to be incremented after each operation.

After the code is executed, each byte in the string1 array will be set to 0xFFh.

The original notes are unclear and hard to understand because they do not explain why the CLD instruction is needed.

The CLD instruction is needed to clear the Direction flag.

If the Direction flag is set, then the EDI register will be decremented after each operation, and the STOSB instruction will fill the string1 array in reverse order.

The STOSB, STOSW, and STOSD instructions are powerful tools for filling all elements of a string or array with a single value.

```
094 mov al, 0xFFh
095 mov edi, OFFSET string1
096 mov ecx, LENGTHOF string1
097 cld
098 rep stosb
```

This code would first move the value 0xFFh into the AL register.

It would then move the value of the EDI register into the ECX register.

Next, it would clear the Direction flag.

Finally, it would repeat the STOSB instruction while ECX is greater than zero.

After this code is executed, each byte in the string1 array would be set to 0xFFh.

LODSB, LODSW, and LODSD

The LODSB, LODSW, and LODSD instructions load a byte, word, or doubleword, respectively, from memory at the offset pointed to by the ESI register into the AL, AX, or EAX register, respectively.

The ESI register 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: Load a Single Value. The following code shows how to use the LODSB instruction to load a single byte from memory into the AL register:

```
101 ; Load a single byte from memory into the AL register
102 lod sb
```

This instruction will load the byte at the memory address pointed to by the ESI register into the AL register. The ESI register will then be incremented by one.

Array Multiplication Example. The following code shows how to use the LODSD and STOSD instructions to multiply each element of an array by a constant value:

```
; 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
    ; 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
    exit
    main ENDP
    END main
```

This code first clears the Direction flag. It then moves the offset of the array variable to the ESI and EDI registers.

The ESI register will be used to address the source array, and the EDI register will be used to address the destination array.

The code then moves the length of the array variable to the ECX register. This will be used as the loop counter.

The code then enters a loop. On each iteration of the loop, the LODSD instruction is used to load a doubleword from the source array into the EAX register.

The EAX register is then multiplied by the value of the multiplier variable. The result is then stored in the destination array using the STOSD instruction.

The loop is repeated until the ECX register is equal to zero. After the loop is finished, the code exits.

The original notes are unclear and hard to understand because they do not explain why the Direction flag is cleared and why the ECX register is used as a loop counter.

The Direction flag is cleared so that the ESI and EDI registers will be incremented after each iteration of the loop.

This ensures that the LODSD and STOSD instructions will access the next element of the source and destination arrays, respectively.

The ECX register is used as a loop counter to ensure that the loop will repeat the correct number of times.

The loop will repeat until the ECX register is equal to zero. This means that the loop will repeat the same number of times as the length of the array variable.

The LODSB, LODSW, and LODSD instructions are powerful tools for loading data from memory into the accumulator.