String Instructions

A string is simply an array of bytes or words
Here are some operations which may 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 alphabetically

The Direction Flag

One of the control flags in the FLAGS register is the *direction flag* (DF)
It determines the direction in which string operations will proceed
The string operations are implemented by the two index registers SI and DI
If DF = 0, SI and DI proceed in the direction of increasing memory addresses
If DF = 1, they proceed in decreasing direction

CLD and STD

To make DF = 0, use the cld instruction

```
cld ;clear direction flag
```

To make DF = 1, use the std instruction

```
std ;set direction flag
```

cld and std have no effect on the other flags

Moving a String

Suppose we have defined two strings

```
DATASEG
string1 DB "HELLO"
string2 DB 5 DUP (?)
```

The movsb instruction

```
movsb ;move string byte
```

copies the contents of the byte addressed by DS:SI to the byte addressed by ES:DI
after the byte is moved, both SI and DI are incremented if DF=0; if DF=1, they are decremented

MOVSB example

To copy the first two bytes of str1 to str2, we use the following instructions:

```
mov ax,@data
mov ds,ax ;initialize ds
mov es,ax ; and es
lea si,[str1] ;si points to source string
lea di,[str2] ;di points to dest string
cld ;set df=0 (increasing)
movsb ;move first byte
movsb ;move second byte
```

The REP Prefix

movsb moves only a single byte from the source string to the destination
To move the entire string, first initialize cx to the number *N* of bytes in the source string and execute rep movsb
The rep prefix causes movsb to be executed *N* times
After each movsb, cx is decremented until it becomes 0

REP Example

```
mov ax,@data
mov ds,ax ;initialize ds
```

```
mov es,ax ; and es
lea si,[str1] ;si points to source string
lea di,[str2] ;di points to dest string
cld ;set df=0 (increasing)
mov cx,5 ;# of chars in string1
rep movsb ;copy the string
```

MOVSW

The word form of movsb is movsw

`movsw ;move string word`

movsw moves words rather than bytes

After the string word has been moved, both SI and DI are incremented (or decremented) by 2

Neither movsb nor movsw have any effect on the flags

The STOSB and STOSW Instructions

`stosb ;store string byte`

Moves the contents of the AL register to the byte addressed by ES:DI

DI is incremented if DF=0 or decremented if DF=1

Similarly,

`stosw ;store string word`

Moves the contents of AX register to the word addressed by ES:DI

DI is incremented or decremented by 2

Neither stosb nor stosw have any effect on the flags

Code using STOSB

```
mov ax,@data
mov es, ax ;initialize es
lea di,[str] ;di points to str
cld ;process to the right
mov al,'A' ;al has char to store
stosb ;store an 'A'
stosb ;store another one
```

Reading and Storing a Character String

Int 21h, function 1 reads a character from the keyboard into AL

Use interrupt with stosb to read a character string

Pseudocode:

chars_read = 0
read a character (using int 21h, fcn 1)
while *character is not* CR *do*
if char is BS *then*
chars_read = chars_read - 1
back up in string
else
store char in string
chars_read = chars_read + 1
endif
read another character
endwhile

Code to Read a String

```
cld ;process from left
```

```
xor bx,bx ;BX holds no. of chars read
mov ah,1 ;input char function
int 21h ;read a char into AL
WHILE1: cmp al,0Dh ;<CR>?
je ENDWHLE1 ;yes, exit
;if char is backspace
cmp al,08h ;is char a backspace?
jne ELSE1 ;no, store in string
dec di ;yes, move string ptr back
dec bx ;decrement char counter
jmp read ;and go to read another char
ELSE1: stosb ;store char in string
inc bx ;increment char count
READ: int 21h ;read a char into AL
jmp WHILE1 ;and continue loop
ENDWHLE1:
```

See READSTR.ASM for a complete procedure

The LODSB Instruction

`lodsb` ;*load string byte*

Moves the byte addressed by DS:SI into the AL register
SI is incremented if DF=0 or decremented if DF=1
Similarly,

`lodsw` ;*store string word*

Moves the word addressed by DS:SI into the AX register
SI is incremented or decremented by 2
Neither lodsb nor lodsw have any effect on the flags

Code using LODSB

```
DATASEG
str DB 'ABC' ;define string
CODESEG
mov ax,@data
mov ds, ax ;initialize ds
lea si,[str] ;si points to str
cld ;process left to right
lodsb ;load first byte in al
lodsb ;load second byte in al
```

Displaying a Character String

Int 21h, function 2 displays the character in dl
Use interrupt with lodsb to display a character string
Pseudocode:
for *count* times do
load string character into `al`
move it to `dl`
output the character
endfor

Code to Display a String

```
cld ;process from left
mov cx,number ;cx holds no. of chars
jcxz ENDFOR ;exit if none
mov ah,2 ;display char function
TOP:
lodsb ;char in al
mov dl,al ;move it to dl
int 21h ;display character
loop TOP ;loop until done
ENDFOR:
```

Scan String

`scasb ;scan string byte`
examines a string for a target byte (contained in al)
subtracts the string byte pointed to by es:di from al and sets the flags
the result is not stored
di is incremented if df = 0 or decremented if df = 1

SCASW

scasw is the word form of scan string
The target word is in ax
di is incremented or decremented by 2 depending on the value of df
All the status flags are affected by scasb and scasw

SCASB Example

```
DATASEG
str DB 'ABC' ;define string
CODESEG
mov ax,@data
mov es,ax ;initialize es
cld ;process left to right
lea di,[str] ;di points to str
mov al,'B' ;target character
scasb ;scan first byte
scasb ;scan second byte
```

REPNE, REPNZ, REPE, and REPZ

In looking for a target byte, the string is traversed until a match is found or the string ends
As with rep, cx is initialized to the length of the string
`repne scansb` (*repeat while not equal*) will repeatedly subtract each string byte from al, update di, and decrement cx
until either the target is found (zf = 1) or cx = 0
repnz is a synonym for repne
repe (*repeat while equal*) repeats a string instruction until zf = 0 or cx = 0
repz is a synonym for repe

Comparing Strings

The cmpsb instruction
`cmpsb` *`;compare string byte`*
subtracts the byte addressed by DS:SI from the byte addressed by ES:DI, sets the flags, and throws the result away
afterward, both SI and DI are incremented if DF=0; if DF=1, they are decremented

The word version of cmpsb is
`cmpsw` *;compare string word*
All status flags are affected by cmpsb and cmpsw

Example of CMPSB

```
mov ax,@data
mov ds,ax ;initialize ds
mov es,ax ; and es
lea si,[string1] ;si points to first string
lea di,[string2] ;di points to second string
cld ;left to right processing
mov cx,10 ;# of chars in strings
repe cmpsb ;compare string bytes
jl S1_1st ;string1 precedes string2
jg S2_1st ;string2 precedes string1
mov ax, 0 ;put 0 in ax, string1=string2
jmp EXIT ;and exit
S1_1st:
mov ax, 1 ;put 1 in ax, string1>string2
jmp EXIT ;and exit
S2_1st:
mov ax, -1 ;put -1 in ax, string1<string2
EXIT:
```