STRING INSTRUCTIONS

We will see:

That the 80x86 family of processors contains nine instructions specifically to deal with string operations.

String operations simplify programming whenever a program must interact with a user.

User commands and responses are usually saved as ASCII strings of characters

The Nine String instructions are:

REP	Repeat
REPE (REPZ)	Repeat while equal (zero)
REPNE (REPNZ)	Repeat while not equal (NZ)
MOVS	Move byte or word string
MOVSB (MOVSW)	Move byte string (word string)
CMPS	Compare byte or word string
SCAS	Scan byte or word string
LODS	Load byte or word string
STOS	Store byte or word string

String instructions con’t

A String is a collection of bytes or words.

No matter what kind of information is stored in a String, there are a number of common operations that we tend to perform on them.

These operations generally consist of copying, comparing, or scanning.

Strings con’t

A special instruction called **repeat prefix** can be used to repeat the copy, compare or scan operations.

Register ECX has an assigned role in this process. It contains the repeat count necessary for the repeat prefix.

ECX is decremented during the string operation, which terminates when ECX reaches zero.

ESI and EDI registers

The ESI and EDI registers are a vital part of all string operations.

The processor assumes that the ESI register points to the first element of the source string. Similarly, it assumes EDI points to the first element of the Destination.

A special flag called the **Direction Flag** is used to control the way ESI and EDI are adjusted during a string instruction. i.e. incremented or decremented based on the condition of the flag.

ESI and EDI registers con’t

In the original Real-Mode addressing the string primitives used SI and DI in conjunction with the segment registers SI is an offset of DS and DI was an offset of ES.

However, from our viewpoint using the flat memory model (Protected Mode) We do not need to worry about this and use ESI and EDI registers that are both fixed.

Example of Real Mode NO LONGER USED in 32-BIT

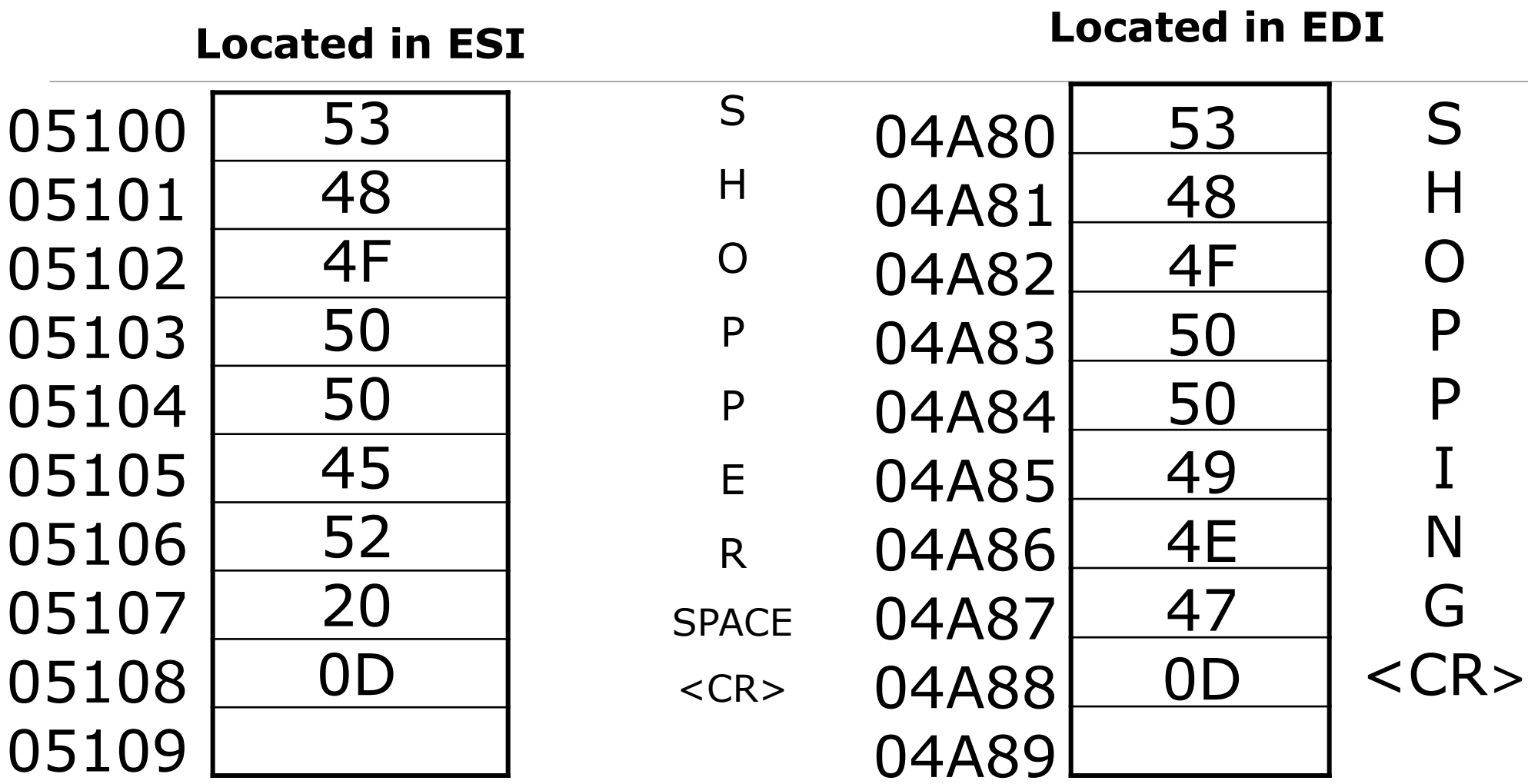
You may see this structure in the books but we NO longer need to worry about DS and ES.

```
main PROC
        mov ax,@data ; point AX of DATA
        mov  ds,ax ; initialise the DS segment
        mov  es,ax ; initialise the ES segment
```

The following example moves 10 bytes from string1 to string2

```
cld                          ;clear direction flag
mov esi, OFFSET string1      ;ESI points to source
mov edi, OFFSET string2      ;EDI points to destination
mov ecx, 10                  ;set counter to 10
rep movsb                    ;mov 10 bytes
```

String example



Example con’t

In the last slide two text strings are stored in memory. The first spells SHOPPER and is followed by a space and carriage return.

The second spells SHOPPING and is followed only by a carriage return.

Although both strings are of the same length they differ after the fifth character. We shall use these two strings to investigate 80x86 string instructions.

1.Initializing the String pointers

Before we can use any string we must set up the ESI and EDI registers.

```
MOV     ESI, OFFSET shopper
MOV     EDI, OFFSET shopping
```

DIRECTION FLAG

We have previously mentioned the use of the direction flag.

This flag is one of the bits in the flags register.

Its function is to select the direction in which the various string functions operate on the EDI and ESI registers.

If it is set to 1 then the operation is auto decremented.

If set to 0 then the operation is auto-incremented

Direction Flag con’t

There are two instructions used to set this flag.

CLD (Clear direction flag) Clears the direction flag i.e. sets it to 0.

SLD (Set direction flag) Sets the direction flag i.e. set it to 1.

CLD selects auto-increment.

SLD selects auto-decrement.

REP/REPE/REPZ/REPNE/REPNZ

These five mnemonics are available for use by the programmer to control the way a string operation is repeated.

All five instructions are recognised by the processor as prefix instructions for string operations. MOVS (move string) and STOS (store string) make use of the REP prefix.

When preceded by REP these string instructions repeat the operation until ECX reaches zero

REP/REPE/REPZ/REPNE/REPNZ

REPE and REPZ operate in the same way but are used for SCAS (scan string) and CMPS (compare string).

Here an additional condition is needed to continue (i.e. the zero flag) The zero flag is set if string found or match.

This gives three ways to repeat string operations:

- Repeat while ECX does not equal 0
- Repeat while ECX does not equal 0 and zero flag set
- Repeat while ECX does not equal 0 and zero flag cleared.

REP REPE/REPZ con’t

The REP instructions can be used with any combination of the string primitives.

MOVSB, MOVSW, STOSB, STOSW, LODSB, LODSW, CMPSB, CMPSW, SCASB and SCASW.

Since MOVS, LODS and STOS have no effect on the flags it is unlikely one would want to use REPZ or REPNZ in conjunction with these instructions, but it is not forbidden.

MOVS

Use: Destination-String, Source-String (Move String)

This instruction is used to make a copy of the source string in the destination string.

The names of the strings must be used in the operand fields so that the processor knows if they are byte strings or word strings i.e.

- STRINGX DB 'SHOPPER', 0DH
- STRINGY DW 1000H, 2000H, 3000H,4000H

The assembler will associate DB or DW in the source line to set the type of string being defined.

MOVS con’t

When MOVS executes, it assumes that the strings are pointed to by ESI and EDI and if any of the REP prefixes are used that ECX is initialized to the proper count

The state of the direction flag will determine which way the strings are copied.

If cleared ESI and EDI will auto increment.

If set ESI and EDI auto-decrement.

The direction flag can be cleared with CLD and set with STD

MOVSB/MOVSW

These two mnemonics can be used in place of MOVS and cause identical execution.

Since they explicitly inform the assembler of the string size there is no need to include the string operands in the instruction.

See next slide.

Problem: Make a copy of the SHOPPER string to destination address copyLoc, and increment the index registers automatically during the string operation.

```
.data
        shopper     db "SHOPPER"
        copyLoc     db 100 DUP(0)

main    proc
        MOV         CX,LENGTHOF shopper     ;COUNT FOR REPEAT
        MOV         ESI, OFFSET shopper     ;SOURCE ADDRESS
        MOV         EDI, OFFSET copyLoc     ;DESTINATION ADDRESS
        CLD                                 ;AUTO-INCREMENT CLEAR
                                            ;DIRECTION FLAG
        REP                                 ;REPEAT WHILE CX <>0
        MOVSB                               ;COPY STRING
```

CMPS (Compare Byte or Word String)

This instruction is used to compare two strings. This is done by the processor doing an internal subtraction.

If the two strings are equal the zero flag is set. Otherwise its cleared.

The REPZ prefix will allow strings to be checked to see if they are identical. This is a handy tool when writing interactive programs.

CMPS example

```
MATCH   PROC
                MOV     ESI,LINE        ;Address of LINE
                MOV     EDI,TABLE       ;Address of TABLE
                CLD                     ;auto-increment
                MOV     ECX,10          ;counter
                REPE CMPSB              ;search
                RET
MATCH   ENDP
```

If ECX = zero when proc returns then strings are the same else they do not match.

SCAS (Scan byte or word string)

This instruction is used to scan a string by comparing each string element with the value saved in AL, AX or EAX.

AL is used for Byte Strings

AX is for Word strings.

EAX is for Double word strings.

The string element pointed to by EDI is internally subtracted from the accumulator and the flags adjusted accordingly.

Once again the zero flag is set if a match is found.

SCAS con’t

If REPNZ is used as the prefix to SCAS the string is effectively searched for the item contained in the accumulator.

Alternately, if REPZ is used, a string can be scanned until an element differs from the accumulator.

SCAS example

```
MOV    EDI,BLOCK          ;ADDRESS OF DATA
CLD                       ;AUTO-INCREMENT
MOV    ECX,100            ;COUNTER
MOV    AL,'A'
REPNE SCASB               ;SEARCH
```

On exit if ECX zero then no match else ECX contains how many bytes checked before match

LODS (Load Byte or Word String)

This instruction is used to load the current string element into the accumulator and automatically advance the ESI register to point to the next element.

It is assumed that ESI has already been set up prior to execution of LODS.

The direction flag determines whether ESI is incremented or decremented.

STOS (Store Byte or Word String)

This instruction is used to write elements of a string into memory.

The byte or word in AL or AX is written into memory at the address pointed to by ESI and then EDI is adjusted accordingly depending on the state of the direction flag and the size of the string elements.

Example: We wish to modify the SHOPPING string and add MALL to it using STOS the original CR needs to be replaced by a blank and a new CR placed at the end.

Solution: Since the modification represents 6 bytes to write into memory we will use the word version of STOS to write two codes into memory at a time.

First we will write the blank and letter ‘M’ then ‘A’ and ‘L’ and finally ‘L’ and ‘CR’.

The direction flag will be cleared to allow auto-increment of EDI which must be initialized to address of the “shopping” string to begin.

STOS example con’t

```
MOV     ESI,OFFSET FIRST_BLOCK      ;SOURCE ADDRESS
MOV     EDI,OFFSET FIRST_BLOCK +8   ;DESTINATION ADDRESS


CLD                                 ;AUTO-INCREMENT CLEAR DIRECTION
  FLAG
MOV     AX,4D20H                    ;CODE FOR BLANK 20H AND M 4DH
STOSW                               ;STORE WORD
MOV     AX,4C41H                    ;CODE FOR LETTERS AL 41H and 4CH
STOSW
MOV     AX,0D4CH                    ;CODE FOR L AND CR 4CH and 0DH
STOSW
```

Note: **

we loaded the AX register with each byte in reverse (Byte Swapped). Remember the processor will take care of reversing the values in memory

Using STOS to clear a block of memory

```
MOV EDI,BUFFER      ;get buffer address
MOV ECX,10          ;load counter
CLD                 ;select auto-increment
MOV AL,0            ;Clear AL
REP   STOSB         ;Clear buffer
```

Using STOSW to clear a buffer

```
MOV   EDI,BUFFER      ;get buffer address
MOV ECX,5             ;Load counter
CLD                   ;select auto-increment
MOV AX,0              ;clear AX
REP   STOSW           ;clear buffer
```