Introduction

• We already know that assembly code will execute significantly faster than code written in a higher-level language such as BASIC, C++ or Java.
• We’ll see that this can also apply to strings and array processing
String Primitive Instructions

- There are five groups of instructions for processing arrays of bytes, words and doublewords.
  - Although they are called string primitives they are not necessarily used just for strings.
- The groups are
  - Moving string data
  - Comparing strings
  - Scanning Strings - comparing an integer to memory contents.
  - Storing string data
  - Loading the accumulator form a string

Setting the ESI and EDI In Protected Mode

- ESI is automatically an offset in the segment to which DS points and EDI is automatically an offset in the segment to which ES points.
- DS and ES are set to the same values are cannot be changed.
Setting the ESI and EDI In Real Mode

- String primitives use SI and DI to address memory
- SI is an offset from DS and DI is an offset from ES.
- The programmer set these at the beginning of the main procedure:

```
main    proc
    mov ax, @data
    mov ds, ax
    mov es, ax
```

Using `rep`

- The string primitives by themselves only act on a single memory location.
- The rep prefix will make an instruction repeat:

```
rep        Repeat while ECX > 0
repz, repe Repeat while ZF is set
              and ECX > 0
repnz, repne Repeat while ZF is cleared and ECX > 0
```
Using \texttt{rep}: An Example

- To act on an entire string, set the ECX with the counter and use the rep prefix before the instruction:
  
  \begin{verbatim}
  cld ; clear direction flag
  mov esi, OFFSET string1 ; ESI pts to source
  mov edi, OFFSET string2 ; EDI pts to target
  mov ecx, 10 ; set counter to 10
  rep movsb ; move 10 bytes
  \end{verbatim}

Direction Flag

- The direction Flag is used to determine whether the ESI and EDI registers are incremented or decremented by the string primitives:

  \begin{tabular}{|c|c|c|}
  \hline
  Value of Direction Flag & Effect on EDI and ESI & Address Sequence \\
  \hline
  Clear & Incremented & Low to high \\
  \hline
  Set & Decremented & High to Low \\
  \hline
  \end{tabular}

- The CLD and STD instructions can explicitly change the Direction flag:
  
  \begin{verbatim}
  cld ; clear Direction flag
  std ; set Direction flag
  \end{verbatim}
### MOVSB, MOVSW, MOVSD Instructions

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Meaning</th>
<th>Value added/subtracted from ESI/EDI</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOVSB</td>
<td>Move Bytes</td>
<td>1</td>
</tr>
<tr>
<td>MOVSW</td>
<td>Move Words</td>
<td>2</td>
</tr>
<tr>
<td>MOVSD</td>
<td>Move Doublewords</td>
<td>4</td>
</tr>
</tbody>
</table>

### MOVSD: An Example

```assembly
.data
source DWORD 20 dup (0FFFFFFFFh)
target DWORD 20 dup (?)
.code
  cld ; direction = forward
  mov ecx, LENGTHOF source ; set REP counter
  mov esi, OFFSET source ; ESI pts to source
  mov edi, OFFSET target ; EDI pts to target
  rep movsd ; copy doublewords
```
CMPSB, CMPSW, CMPSD

- CMPSB, CMPSW and CMPSD each compare a memory operand pointed to by ESI to one pointed to by EDI.

- You can use a repeat prefix with CMPSB, CMPSW and CMPSD. The direction flag increments or decrements ESI and EDI.

<table>
<thead>
<tr>
<th>CMPSB</th>
<th>Compare Byte</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMPSW</td>
<td>Compare Words</td>
</tr>
<tr>
<td>CMPSD</td>
<td>Compare Doublewords</td>
</tr>
</tbody>
</table>

- You can use a repeat prefix with CMPSB, CMPSW and CMPSD. The direction flag increments or decrements ESI and EDI.
CMPSB, CMPSW, CMPSD: An Example

.data
source DWORD 1234h
target DWORD 5678h
.code
    mov esi, OFFSET source
    mov edi, OFFSET target
    cmpsd ; compare doublewords
    ja L1 ; jump if source > target
    jmp L2 ; jump, since source <= target

CMPSB, CMPSW, CMPSD: Another Example

• We can compare multiple doublewords by initializing the ECX register and using a repeat prefix:
  mov esi, OFFSET source
  mov edi, OFFSET target
  cld ; direction = up
  mov ecx, count ; repetition counter
  repe cmpsd ; repeat until equal

• The repe prefix repeat the comparison until either ECX = 0 or a pair of doublewords are different.
Example: Comparing Two Strings

TITLE Comparing String (Cmpsb.asm)
; This program uses CMPSB to compare two strings
; of equal length
INCLUDE Irvine32.inc
.data
source BYTE "MARTIN ", 0,
dest BYTE "MARTINEZ ", 0,
str1 BYTE "Source is smaller", 0dh, 0ah, 0
str2 BYTE "Source is not smaller", 0dh, 0ah, 0
.code
main PROC
  cld ; direction = up
  mov esi, OFFSET source
  mov edi, OFFSET dest
  mov cx, LENGTHOF source
  repe cmpsb
  jb source_smaller
  mov edx, OFFSET str2
  jmp done
source_smaller:
  mov edx, OFFSET str1
done: call WriteString
  exit
main ENDP
END main
**The Strings Before and After**

**Before**

<table>
<thead>
<tr>
<th>Source</th>
<th>Dest</th>
</tr>
</thead>
<tbody>
<tr>
<td>Martin</td>
<td>Martinez</td>
</tr>
<tr>
<td>ESI</td>
<td>EDI</td>
</tr>
</tbody>
</table>

**Before**

<table>
<thead>
<tr>
<th>Source</th>
<th>Dest</th>
</tr>
</thead>
<tbody>
<tr>
<td>Martin</td>
<td>Martinez</td>
</tr>
<tr>
<td>ESI</td>
<td>EDI</td>
</tr>
</tbody>
</table>

**SCASB, SCASW, SCASD**

- **SCASB, SCASW, SCASD** compare a value in the AL (or AX or EAX) register to a byte (or word or doubleword) addressed by the EDI register.
- This is useful when you’re for a single value in a long string or array.
- **REPE SCASB** will scan a string while ECX > 0 and the value in AL matches each subsequent value in memory.
- **REPNE SCASB** will scan a string **until** ECX > 0 and the value in AL matches each subsequent value in memory.
Scanning For a Matching Character

.data
alpha BYTE "ABCDEFGH", 0
.code
    mov edi, OFFSET alpha ; EDI points to string
    mov al, 'F' ; search for the letter F
    mov ecx, LENGTHOF alpha
    cld ; direction = up
    repne scasb ; repeat while not equal
    jnz quit ; quit if letter not found
    dec edi ; found: back up EDI

STOSB, STOSW, STOSD

- **STOSB, STOSW, STOSD** store the contents of the AL (or AX or EAX) in memory at the offset pointed to by EDI.

- EDI is incremented or decremented based on the direction flag.

- Using the rep prefix, an entire string can be filled with a single byte (or word or doubleword).
STOSB, STOSW, STOSD: An Example

.data
Count = 100
,string1 BYTE Count DUP (?)

.code
mov al, 0FFh ; value to be stored
mov edi, OFFSET string1
; ES;DI points to target
mov ecx, Count ; character count
cld ; direction = forward
rep stosb ; fill with contents of AL

LODSB, LODSW, LODSD

- **LODSB, LODSW, LODSD** loads a byte (or word or doubleword) from memory at ESI into the AL (or AX or EAX) and the ESI is incremented or decremented based on the direction flag.
- rep is rarely used with **LODSB, LODSW, LODSD** because it would overwrite the same data.
- **LODSB** replaces the following:
  
  mov al, [esi] ; move byte into AL
  inc esi ; point to the next byte
Example: Multiply an Array

TITLE Multiply an Array (Mult.asm)
; This program multiplies each element of an array
; 32-bit integers by a constant value.
INCLUDE Irvine32.inc
.data
array DWORD 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
multiplier DWORD 10
.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
Selected String Procedures

- Str_compare
- Str_length
- Str_copy
- Str_trim
- Str_ucase

Two-Dimensional Arrays

- Base-Index operands
- Base-Index Displacement
Searching and Sorting Integer Arrays

- Bubble Sort
- Binary Search