BITS Pilani

Microprocessors & Interfacing

MASM & INSTRUCTION SET

Dr. Gargi Prabhu
Department of CS & IS

String Comparisons

- String instructions are very powerful because they allow the programmer to manipulate large blocks of data with relative ease.
- Block data manipulation occurs with the string instructions MOVS, LODS, STOS.
- Instructions that allow a section of memory to be tested against a constant or against another section of memory:
  - SCAS (string scan)
  - CMPS (string compare)

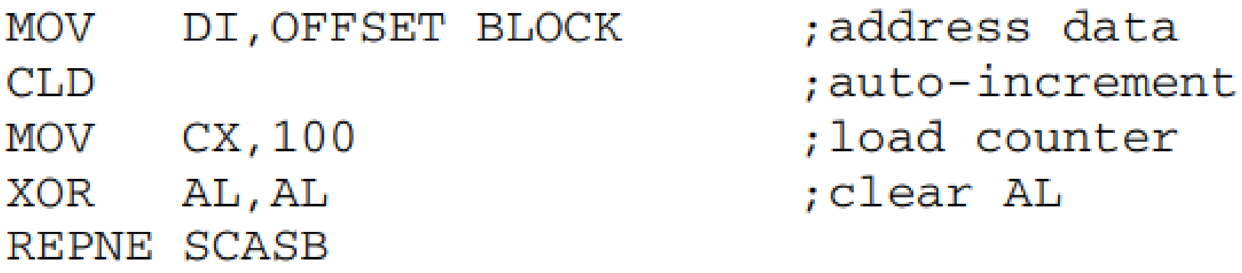
SCAS (string scan instruction)

- Compares the AL register with a byte block of memory, the AX register with a word block of memory, or the EAX register (80386-Core2) with a doubleword block of memory.
- Subtracts memory from AL, AX, or EAX without affecting either the register or the memory location
- SCASB – Byte Comparison
- SCASW – Word Comparison
- SCASD – Double Word Comparison

- Contents of the extra segment memory location addressed by DI is compared with AL, AX, or EAX

SCASB Example

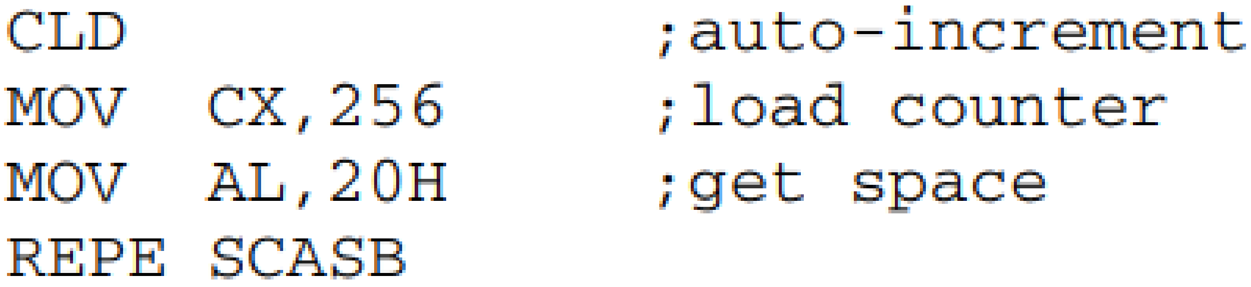
Search part of memory for 00H



```
MOV    DI,OFFSET BLOCK      ;address data
CLD                         ;auto-increment
MOV    CX,100               ;load counter
XOR    AL,AL                ;clear AL
REPNE SCASB
```

SCASB Example

Skip ASCII-coded spaces

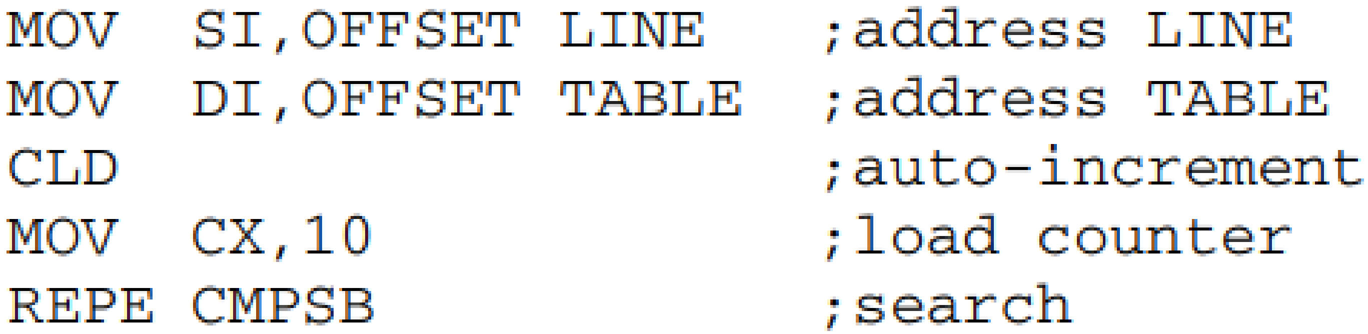


```
CLD                     ;auto-increment
MOV   CX,256            ;load counter
MOV   AL,20H            ;get space
REPE  SCASB
```

CMPS: String Compare

- The CMPS (compare strings instruction) always compares two sections of memory data as bytes (CMPSB), words (CMPSW), or doublewords (CMPSD)
- The contents of the data segment memory location addressed by SI are compared with the contents of the extra segment memory location addressed by DI.
- The CMPS instruction increments or decrements both SI and DI.
- CMPS instruction is normally used with either the REPE or REPNE prefix.

CMPS Example



```
MOV   SI,OFFSET LINE    ;address LINE
MOV   DI,OFFSET TABLE   ;address TABLE
CLD                     ;auto-increment
MOV   CX,10             ;load counter
REPE CMPSB              ;search
```

MASM

- MASM, short for Microsoft Macro Assembler, is a low-level programming language used for programming in assembly language.
- Developed by Microsoft, MASM provides developers with a powerful toolset for writing efficient and optimized code for x86 and x64 architectures.
- Originally introduced in the early 1980s, MASM has since evolved and been updated to support modern computing platforms.
- MASM is widely used in various domains including system programming, device drivers, embedded systems, and performance-critical applications.

Basic Rules

- MASM follows a syntax that is based on mnemonic instructions and directives.
- Instructions are symbolic representations of machine-level operations, while directives are commands to the assembler itself.
- Examples of directives include .MODEL, .DATA, .CODE, etc.
- Comments in MASM start with a semicolon (;) and continue until the end of the line.
- Labels are used to mark specific locations in the code and are followed by a colon (:).
- MASM is not case-insensitive, meaning that upper and lower case letters are treated the same way.

Directives

TITLE line (optional)

- Contains a brief heading of the program and the disk file name

.MODEL directive

- Specifies the memory model configuration

TINY: Suitable for small programs that fit within a single code segment and don't require data or stack segments.

SMALL: Suitable for small to moderately sized programs with separate code, data, and stack segments. Code and data segments can be up to 64KB in size.

COMPACT, MEDIUM, LARGE, HUGE: These memory models are suitable for larger programs that require more memory segmentation. They provide increasing levels of memory management and segmentation capabilities.

Directives

.STACK directive

- Tells the assembler to define a runtime stack for the program
- The size of the stack can be optionally specified by this directive
- The runtime stack is required for procedure calls

.DATA directive

- Defines an area in memory for the program data
- The program's variables should be defined under this directive
- Assembler will allocate and initialize the storage of variables

.CODE directive

- Defines the code section of a program containing instructions
- Assembler will place the instructions in the code area in memory

Directives

INCLUDE directive

- Causes the assembler to include code from another file

PROC and ENDP directives

- Used to define procedures
- As a convention, you may define ***main*** as the first procedure
- Additional procedures can be defined after ***main***

END directive

- Marks the end of a program
- Identifies the name (***main***) of the program's startup procedure

Data Types

DB (Define Byte):
Used to declare one or more bytes.
Syntax: variable_name DB initial_value

DW (Define Word):
Used to declare one or more 16-bit words (2 bytes).
Syntax: variable_name DW initial_value

DD (Define Doubleword):
Used to declare one or more 32-bit double words (4 bytes).
Syntax: variable_name DD initial_value

MASM Example

```
MODEL SMALL
DATAS  SEGMENT
    STRING  DB  'Hello World! Its 2024!','$'
DATAS  ENDS

CODES  SEGMENT
    ASSUME    CS:CODES,DS:DATAS

START:
    MOV  AX,DATAS
    MOV  DS,AX

    LEA  DX,[STRING]

    MOV  AH,9
    INT  21H

    MOV  AH,4CH
    INT  21H
CODES  ENDS
END   START
```

This moves the value 09h into the AH register. This value is the DOS function number for "Display String" using the DOS interrupt 21h

MASM Example

```
MODEL SMALL
DATAS  SEGMENT
    STRING  DB  'Hello World! Its 2024!','$'
DATAS  ENDS

CODES  SEGMENT
    ASSUME    CS:CODES,DS:DATAS

START:
    MOV  AX,DATAS
    MOV  DS,AX

    LEA  DX,[STRING]

    MOV  AH,9
    INT  21H

    MOV  AH,4CH
    INT  21H
CODES  ENDS
END   START
```

This generates a software interrupt (interrupt 21h), which is a DOS interrupt. When AH contains 09h and DS:DX points to a $-terminated string, this function prints the string to the screen.

MASM Example: Using Model

```
.model small
.stack 100h

.data
msg        db          'Hello world!$'

.code
start:
    mov  ah, 09h lea          dx, msg
    int    21h
    mov  ax, 4C00h  ;
    int    21h

end start
```

This moves the value 4C00h into the AX register. This value is the DOS function number for "Terminate Program" using the DOS interrupt 21h.

MASM Example: Using Model

```
.model small
.stack 100h

.data
msg       db          'Hello world!$'

.code
start:
     mov  ah, 09h lea          dx, msg
     int     21h
     mov  ax, 4C00h  ;
     int     21h

end start
```

This generates another software interrupt (interrupt 21h), this time to terminate the program. When AH contains 4Ch, this function terminates the program.

Example 2

```
.model small
.stack 64
.data
a db 02h,02h,02h,02h,02h,02h,02h,02h,02h,02h

.code
 start: mov ax,@data
        mov ds,ax
        mov cl,10
        lea si,a
        mov ax,0000h
again: add al,[si]
        inc si
        dec cl
        jnz again
        mov cl,0ah
        div cl
        mov ah,4ch
        int 21h
        end start
        .end
```

Example 3

```
.model small
.stack 64
.data
.code
start:
mov al,00h
mov dl,00h
mov bl,01h
mov cl,05h
again: add al,bl
mov dl,al
mov al,bl
mov bl,dl
dec cl
jnz again
mov ah,4ch
int 21h
end start
.end
```

innovate achieve lead

BITS Pilani
Pilani Campus

Thank You