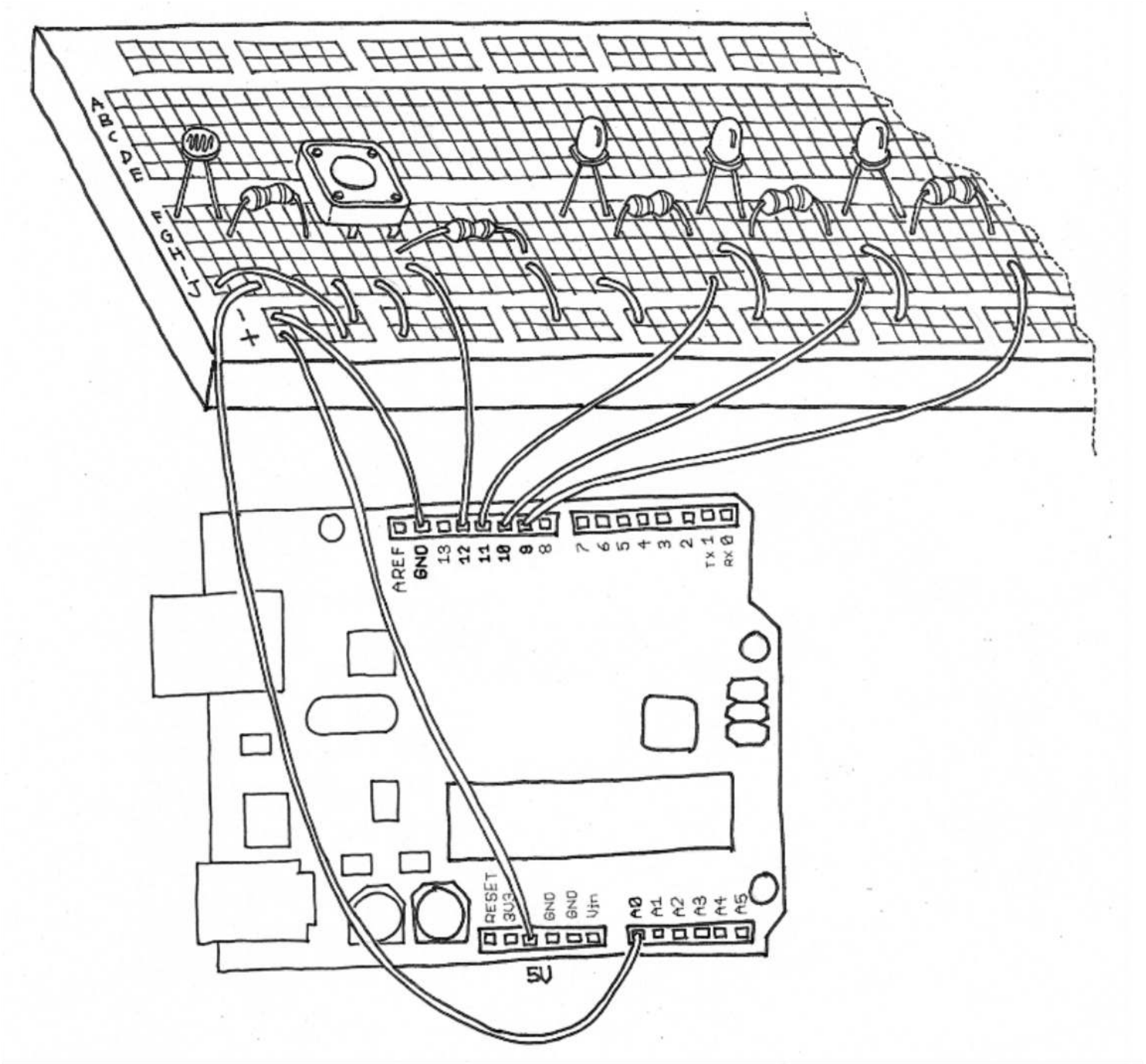
CS/EEE/INSTR F241
Microprocessor Programming and Interfacing

Lab 4 - String Operations



Dr. Vinay Chamola and Anubhav Elhence

String Operations

What are LODSB, LODSW and LODSD instructions?

LODSB, LODSW, and LODSD are three x86 assembly language instructions used to load a byte (8 bits), a word (16 bits), or a doubleword (32 bits) from memory into the AL, AX, or EAX register, respectively. These instructions are part of the string operations category of instructions and are used to read data from a string of bytes, words, or doublewords in memory.

Here's a brief description of each instruction:

1. LODSB (Load String Byte): This instruction reads a byte from memory pointed to by the DS:(E)SI register pair into the AL register. It then increments or decrements the (E)SI register depending on the direction flag (DF) bit in the flags register. If the DF bit is clear, (E)SI is incremented. If the DF bit is set, (E)SI is decremented. This allows the instruction to read bytes from a string in either direction.
2. LODSW (Load String Word): This instruction reads a 16-bit word from memory pointed to by the DS:(E)SI register pair into the AX register. It then increments or decrements the (E)SI register in the same way as LODSB.
3. LODSD (Load String Doubleword): This instruction reads a 32-bit doubleword from memory pointed to by the DS:(E)SI register pair into the EAX register. It then increments or decrements the (E)SI register in the same way as LODSB.

These instructions are often used in conjunction with other string operations, such as STOSB (store string byte), STOSW (store string word), and STOSD (store string doubleword), to manipulate strings of bytes, words, or doublewords in memory.

What are STOSB, STOSW and STOSD instructions?

STOSB, STOSW, and STOSD are three x86 assembly language instructions used to store a byte (8 bits), a word (16 bits), or a doubleword (32 bits) from a register into memory. These instructions are part of the string operations category of instructions and are used to write data to a string of bytes, words, or doublewords in memory.

Here's a brief description of each instruction:

1. STOSB (Store String Byte): This instruction stores the byte in the AL register into the memory location pointed to by the ES:(E)DI register pair. It then increments or decrements the (E)DI register depending on the direction flag (DF) bit in the flags register. If the DF bit is clear, (E)DI is incremented. If the DF bit is set, (E)DI is decremented. This allows the instruction to store bytes into a string in either direction.
2. STOSW (Store String Word): This instruction stores the 16-bit word in the AX register into the memory location pointed to by the ES:(E)DI register pair. It then increments or decrements the (E)DI register in the same way as STOSB.
3. STOSD (Store String Doubleword): This instruction stores the 32-bit doubleword in the EAX register into the memory location pointed to by the ES:(E)DI register pair. It then increments or decrements the (E)DI register in the same way as STOSB.

These instructions are often used in conjunction with other string operations, such as LODSB (load string byte), LODSW (load string word), and LODSD (load string doubleword), to manipulate strings of bytes, words, or doublewords in memory.

What are SCASB, SCASW and SCASD instructions?

SCASB, SCASW, and SCASD are three x86 assembly language instructions used to compare a byte (8 bits), a word (16 bits), or a doubleword (32 bits) in memory with the AL, AX, or EAX register, respectively. These instructions are part of the string operations category of instructions and are used to search for a byte, word, or doubleword in a string of bytes, words, or doublewords in memory.

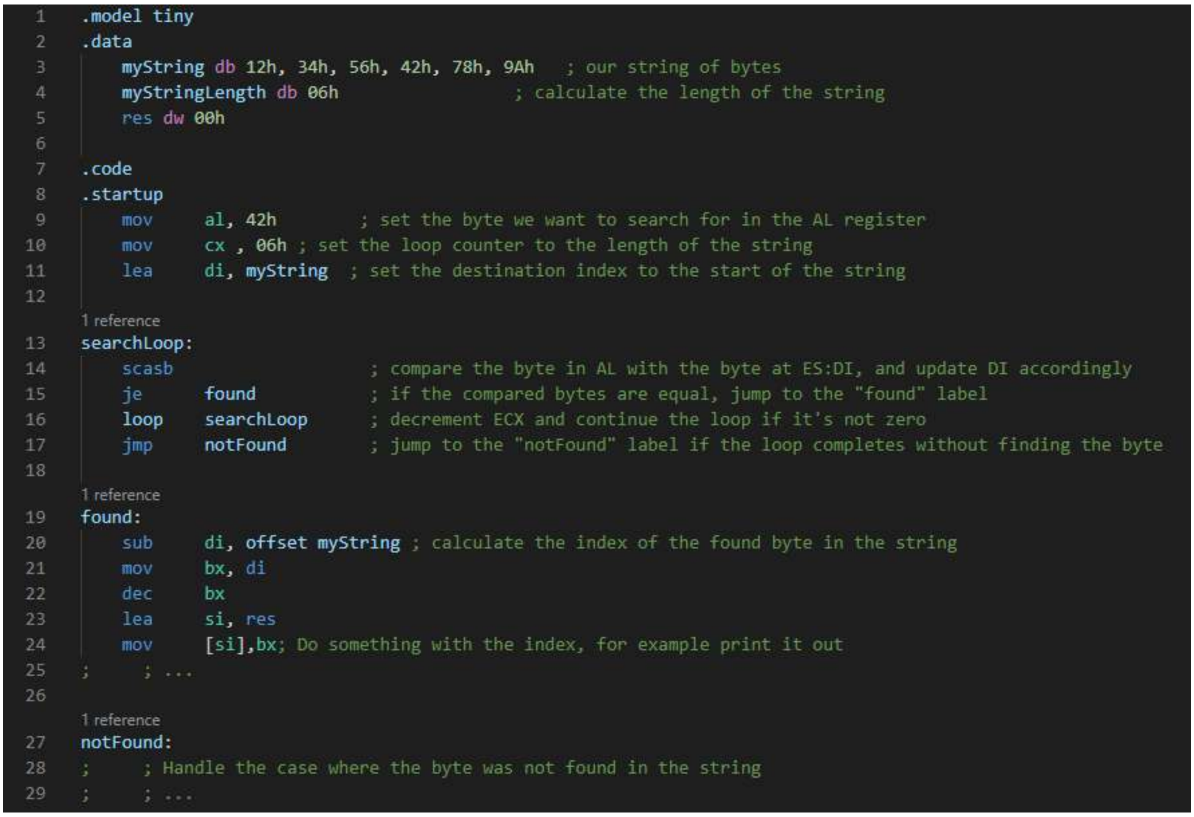
Here's a brief description of each instruction:

1. SCASB (Scan String Byte): This instruction compares the byte in the AL register with the byte at the memory location pointed to by the ES:(E)DI register pair. It then increments or decrements the (E)DI register depending on the direction flag (DF) bit in the flags register. If the DF bit is clear, (E)DI is incremented. If the DF bit is set, (E)DI is decremented. This allows the instruction to search for bytes in a string in either direction.
2. SCASW (Scan String Word): This instruction compares the 16-bit word in the AX register with the word at the memory location pointed to by the ES:(E)DI register pair. It then increments or decrements the (E)DI register in the same way as SCASB.
3. SCASD (Scan String Doubleword): This instruction compares the 32-bit doubleword in the EAX register with the doubleword at the memory location pointed to by the ES:(E)DI register pair. It then increments or decrements the (E)DI register in the same way as SCASB.

These instructions are often used in conjunction with other string operations, such as LODSB (load string byte), LODSW (load string word), and LODSD (load string doubleword), to manipulate and search strings of bytes, words, or doublewords in memory. After the comparison is made, the zero flag (ZF) is set if the compared values are equal, and the carry flag (CF) and the sign flag (SF) are set according to the result of the subtraction operation.

Example:

Let's say we have a string of bytes stored in memory, and we want to search for the first occurrence of the byte 0x42 (hexadecimal representation of the decimal number 66) in the string. We can use the SCASB instruction to do this search.



```
.model tiny
.data
    myString db 12h, 34h, 56h, 42h, 78h, 9Ah   ; our string of bytes
    myStringLength db 06h                  ; calculate the length of the string
    res dw 00h

.code
.startup
    mov     al, 42h         ; set the byte we want to search for in the AL register
    mov     cx , 06h ; set the loop counter to the length of the string
    lea     di, myString  ; set the destination index to the start of the string

searchLoop:
    scasb                   ; compare the byte in AL with the byte at ES:DI, and update DI accordingly
    je      found           ; if the compared bytes are equal, jump to the "found" label
    loop    searchLoop      ; decrement ECX and continue the loop if it's not zero
    jmp     notFound        ; jump to the "notFound" label if the loop completes without finding the byte

found:
    sub     di, offset myString ; calculate the index of the found byte in the string
    mov     bx, di
    dec     bx
    lea     si, res
    mov     [si],bx; Do something with the index, for example print it out
;     ; ...

notFound:
;     ; Handle the case where the byte was not found in the string
;     ; ...
```

In this example code, we first set the AL register to the byte we want to search for, then we set the loop counter to the length of the string and the destination index to the start of the string.

We then enter a loop where we use the SCASB instruction to compare the byte in AL with the byte at ES:DI, and update DI accordingly. If the compared bytes are equal (i.e., the ZF flag is set), we jump to the "found" label. If the loop completes without finding the byte, we jump to the "notFound" label.

In the "found" label, we calculate the index of the found byte in the string by subtracting the offset of the start of the string from the value of DI. We can then do something with this index, for example print it out. In the "notFound" label, we can handle the case where the byte was not found in the string.

A simpler way to do this is by using the REPNE instruction.

What is REPNE and REPE instruction in 8086?

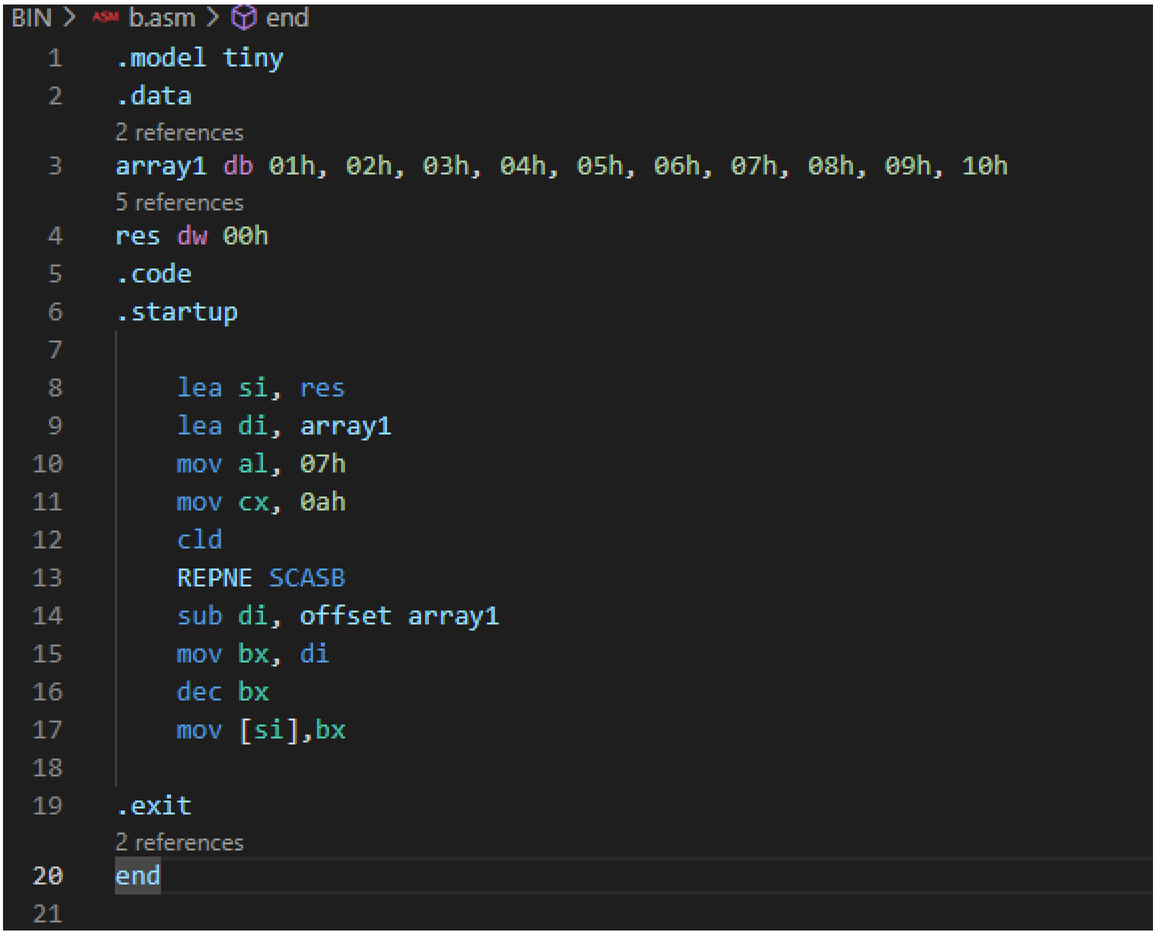
REPNE (repeat not equal) and REPE (repeat equal) are prefix instructions in the x86 assembly language used to repeat string operations with certain conditions.

The REPNE prefix is used to repeat a string operation as long as the condition for not being equal is met. It can be used with string operations such as SCASB, CMPSB, SCASW, CMPSW, SCASD, and CMPSD. For example, the instruction sequence "REPNE SCASB" can be used to search for a byte in a string until the byte is found or the end of the string is reached.

The REPE prefix is used to repeat a string operation as long as the condition for being equal is met. It can also be used with string operations such as SCASB, CMPSB, SCASW, CMPSW, SCASD, and CMPSD. For example, the instruction sequence "REPE CMPSW" can be used to compare two strings of words until a difference is found or the end of the strings is reached.

Both REPNE and REPE instructions use the CX register as a counter for the number of repetitions, and they decrement CX by one after each repetition. If CX becomes zero, the string operation is terminated.

The above example using REPNE:



```
.model tiny
.data
array1 db 01h, 02h, 03h, 04h, 05h, 06h, 07h, 08h, 09h, 10h
res dw 00h
.code
.startup

    lea si, res
    lea di, array1
    mov al, 07h
    mov cx, 0ah
    cld
    REPNE SCASB
    sub di, offset array1
    mov bx, di
    dec bx
    mov [si],bx

.exit
end
```

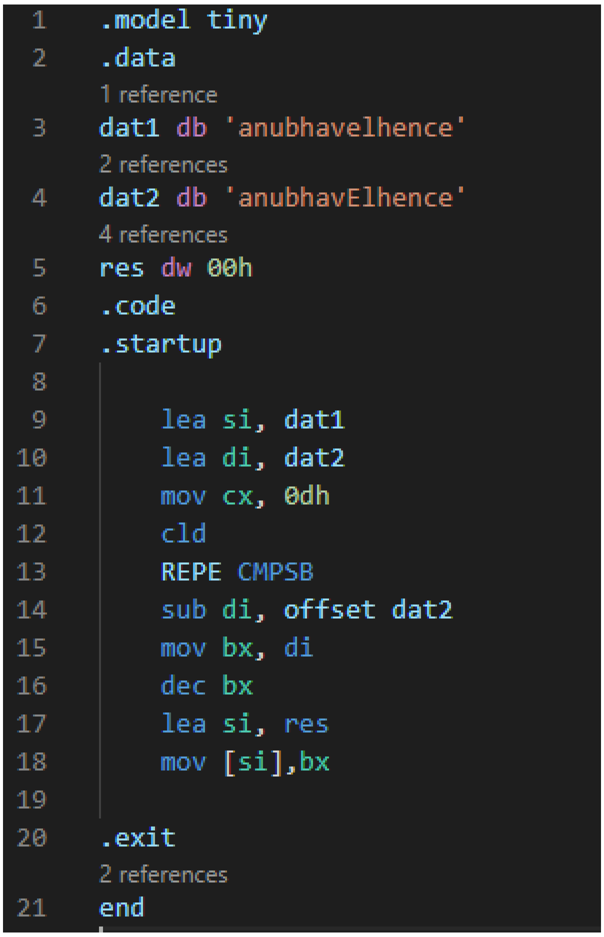
What are CMPSB, CMPSW and CMPSD instructions?

CMPSB, CMPSW, and CMPSD are x86 assembly language instructions used to compare a byte (8 bits), a word (16 bits), or a doubleword (32 bits) in memory at two locations pointed to by the source and destination index registers, SI and DI, respectively. These instructions are part of the string operations category of instructions and are used to compare two strings of bytes, words, or doublewords in memory.

Here's a brief description of each instruction:

1. CMPSB (Compare String Byte): This instruction compares the byte at the memory location pointed to by the DS:SI register pair with the byte at the memory location pointed to by the ES:DI register pair. It then increments or decrements the SI and DI registers depending on the direction flag (DF) bit in the flags register. If the DF bit is clear, both registers are incremented. If the DF bit is set, both registers are decremented. This allows the instruction to compare bytes in two strings in either direction.
2. CMPSW (Compare String Word): This instruction compares the 16-bit word at the memory location pointed to by the DS:SI register pair with the 16-bit word at the memory location pointed to by the ES:DI register pair. It then increments or decrements the SI and DI registers in the same way as CMPSB.
3. CMPSD (Compare String Doubleword): This instruction compares the 32-bit doubleword at the memory location pointed to by the DS:SI register pair with the 32-bit doubleword at the memory location pointed to by the ES:DI register pair. It then increments or decrements the SI and DI registers in the same way as CMPSB.

Take a look at the example, where we try to find out the index where the two string' start to mismatch.



```
.model tiny
.data
1 reference
dat1 db 'anubhavelhence'
2 references
dat2 db 'anubhavElhence'
4 references
res dw 00h
.code
.startup

    lea si, dat1
    lea di, dat2
    mov cx, 0dh
    cld
    REPE CMPSB
    sub di, offset dat2
    mov bx, di
    dec bx
    lea si, res
    mov [si],bx

.exit
2 references
end
```

Lab Task:

Task 1

Write an 8086 program to check whether a given string is palindrome or not. If it is a palindrome, store ‘01h’ in RES or else ‘00h’.

Input String: "wasitcatisaw"

Output: RES = 01h

Go to below link to download starter code: https://github.com/anubhavelhence/Microprocessor-Programming-and-Interfacing-MuP-Lab-Session/blob/week-4/q1.asm

Task 2

Write an 8086 program to replace a substring S1 of a string S with “*”

Input: S = “BITSIOTLAB”, S1 = “IOT”, S2 = “a”

Output: BITS*LAB

Explanation:

Change the substrings S[4,6] to string “*” modifies the string S to “BITS*LAB”

Go to below link to download starter code:

https://github.com/anubhavelhence/Microprocessor-Programming-and-Interfacing-MuP-Lab-Session/blob/week-4/q2.asm

BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI

ज्ञानं परमं बलम्