National University of Computer and Emerging Sciences



**Laboratory Manual**

*for*

*Computer Organization and Assembly Language*

Course Instructors

Lab Instructor(s)

Section

Semester

**Department of Computer Science**

# COAL Lab 13 Manual

|  |
| --- |
| **Objectives:**     * String Instructions * Direction Flag and REP Prefix * Problems & Assignments |

**13.1 String Instructions:**

String instructions perform following functions:

|  |  |
| --- | --- |
| **Instruction** | **Description** |
| **MOVSB, MOVSW, MOVSD** | **Move string data:** copy data from memory addressed by **ESI** to memory addressed by **EDI**. |
| **LODSB, LODSW, LODSD** | **Load accumulator from string:** Load memory addressed by **ESI** into the accumulator. |
| **STOSB, STOSW, STOSD** | **Store string data:** Store the accumulator contents into memory addressed by **EDI**. |
| **SCASB, SCASW, SCASD** | Subtract the contents of accumulator register into memory location addressed by **DI** |
| **CMPSB, CMPSW** | Compare the contents of memory addressed by **SI** to memory addressed by **DI** |

**Table 13.1: String Instructions**

**Direction Flag:**

It is a control flag and works in accordance with following instructions:

|  |  |  |  |
| --- | --- | --- | --- |
| **Format** | **Description** | **Mode** | **Effect** |
| **CLD** | Clear DF, i.e DF=0 | Auto Increment | SI→ SI+1  DI → DI +1 |
| **STD** | Set DF, i.e: DF=1 | Auto Decrement | SI →SI − 1  DI → DI − 1 |

**Table 13.2: Direction Flag**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Instruction** | **Mnemonic** | **DF** | **Operation** | | **Flag affected** | **Example** |
| **Comments** |
| Move string | MOVSB | 0  1 | ES:EDI ← DS:ESI | ; ESI ← ESI + 1  ; EDI ← EDI + 1 | None | **EX # 1: Moving first two bytes of string** |
| **.**DATA  String1 BYTE ‘FAST-NU’  String2 BYTE 5 DUP(?) **.**CODE  **LEA** E**SI**,String1  **LEA** E**DI**,String2  **CLD** ; DF = 0  **MOVSB** ;move 1st byte i.e. H  **MOVSB** ;move 2nd byte i.e. E  ;String1 = FAST-NU ;String2 = FA????? |
| ES:EDI ← DS:ESI | ; ESI ← ESI - 1  ; EDI ← EDI - 1 |
| MOVSW | 0  1 | ES:EDI ← DS:ESI | ; ESI ← ESI + 2  ; EDI ← EDI + 2 |
| ES:EDI ← DS:ESI | ; ESI ← ESI - 2  ; EDI ← EDI - 2 |
| MOVSD | 0  1 | ES:EDI ← DS:ESI | ; ESI ← ESI + 4  ; EDI ← EDI + 4 |
| ES:EDI ← DS:ESI | ; ESI ← ESI - 4  ; EDI ← EDI - 4 |
| Load String | LODSB | 0  1  0  1 0 1 | AL ← DS:ESI | ; ESI ← EDI + 1  ; EDI ← EDI -1 | None | **EX # 2: Loading 1st two bytes of string in**  **AL** |
| **LEA** E**SI**,String1  **CLD** ; DF = 0  **LODSB** ; AL = F  **LODSB** ; AL = A |
| LODSW | AX ← DS:ESI | ; ESI ← EDI + 2  ; EDI ← EDI - 2 |
| LODSD | EAX ← DS:ESI | ; ESI ← EDI + 4  ; EDI ← EDI - 4 |
| Store String | STOSB | 0  1  0  1  0  1 | DS:ESI ← AL | ; ESI ← EDI + 1  ; EDI ← EDI -1 | None | **EX # 3: Storing two L’s in String1** |
| **LEA** E**DI**,String1  **CLD** ; DF = 0  **MOV AL**,‘L’  **STOSB** ; String1 = LAST-  NU  **STOSB** ; String1 = LLST-  NU |
| STOSW | DS:ESI ← AX | ; ESI ← EDI + 2  ; EDI ← EDI - 2 |
| STOSD | DS:ESI← EAX | ; ESI ← EDI + 4  ; EDI ← EDI - 4 |
| Scan String | SCASB | 0  1  0  1  0  1 | DS:ESI -- AL | ; ESI ← EDI + 1  ; EDI ← EDI -1 | CF, AF,  PF, ZF,  SF, OF | **EX # 4: Search ‘N’ in 1st two bytes of String1** |
| **LEA** **DI**,String2-1  **STD** ; DF = 1  **MOV AL**,‘N’  **SCASB** ;Scan 1st byte  ZF=0  **SCASB** ;Scan 2nd byte  ZF=1 |
| SCASW | DS:ESI -- AX | ; ESI ← EDI + 2  ; EDI ← EDI - 2 |
| SCASD | DS:ESI -- EAX | ; ESI ← EDI + 4  ; EDI ← EDI - 4 |
| Compare  String | CMPSB | 0  1  0 | ES:EDI -- DS:ESI | ; ESI ← ESI + 1  ; EDI ← EDI + 1 | CF, AF,  PF, ZF,  SF, OF | **EX # 5: Compare 1st two bytes of given Strings** |
| **.**DATA  String3 DB ‘ACD’  String4 DB ‘ABC’  **.**CODE  **LEA** **SI**,String3 |
| ES:EDI -- DS:ESI | ; ESI ← ESI - 1  ; EDI ← EDI - 1 |
| CMPSW | ES:EDI -- DS:ESI | ; ESI ← ESI + 2  ; EDI ← EDI + 2 |
|  |  | 1  0  1 | ES:EDI -- DS:ESI | ; ESI ← ESI - 2  ; EDI ← EDI - 2 |  | **LEA** **DI**,String4  **CLD** ; DF = 0  **CMPSB** ;Comp 1st byte ZF=1  **CMPSB** ;Comp 2nd byte ZF=0 |
| CMPSD | ES:EDI -- DS:ESI | ; ESI ← ESI + 4  ; EDI ← EDI + 4 |
| ES:EDI -- DS:ESI | ; ESI ← ESI - 4  ; EDI ← EDI - 4 |

**Table 13.3: String Instructions with Examples**

**REP Prefix:**

|  |  |  |
| --- | --- | --- |
| **Prefix** | **Description** | **Examples** |
| REP | Repeat while CX>0 | **EX # 6: Comparing two given Strings** |
| **.**DATA  String5 DB ‘HELLO’  St5\_Len = $ - String5  String6 DB ‘WORLD’  **.**CODE  **LEA** **SI**,String5  **LEA** **DI**,String6  **MOV** **CX**,St5\_Len  **CLD** ; DF = 0  **REP CMPSB** ; Compare string bytes while CX > 0 |
| REPZ, REPE | Repeat while ZF=1 and CX>0 | See Yourself |
| REPNZ, REPNE | Repeat while ZF=0 and CX>0 | See Yourself |

**Table 13.4: REP Instruction with example**

# Problem(s) / Assignment(s)

|  |  |  |
| --- | --- | --- |
| **Discussion & Practice** |  | **Estimated completion time: 1 hr, 30 mins** |

|  |  |
| --- | --- |
| **PROBLEM 13.1: *SEARCHING***  Write a program that performs following tasks using String instructions only:   1. Prompt the user to enter a string terminated by Enter using **STR\_IN** procedure. 2. Search for vowels and consonants in the string using **SEARCH** procedure. 3. Display vowels and consonants in alphabetical order using | **Estimated completion time:30mins** |

**DIS\_STR.**

**Solution:**

include Irvine32.inc

.data

num1 byte 20 dup(0)

num2 byte 'a,e,i,o,u',0

vo byte " are vowels",0dh,0ah,0

co byte "constant",0dh,0ah,0

num3 DWORD ?

num4 byte (0)

num5 DWORD ?

count DWORD ?

.code

main PROC

call fun

call search

mov eax,num3

sub al,num4

mov edx,offset vo

call writestring

mov edx,offset co

call writestring

call writedec

EXIT

fun proc

mov edx,offset num1

mov ecx,sizeof num2

call readstring

ret

fun endp

search proc

mov count,eax

mov esi,offset num2

mov ecx,eax

mov num3,eax

l1:

mov al,[esi]

mov ebx,ecx

cld

mov ecx,num3

mov edi,offset num1

repnz scasb

jz found

jmp next

found:

inc num4

call writechar

next:

add esi,2

mov ecx,ebx

loop l1

ret

search endp

main ENDp

END main

|  |  |
| --- | --- |
| **PROBLEM 13.2:** *Finding relative prime numbers*  Harry wants to check the relative primality of 2 numbers. For this purpose, he checks the GCD (Greatest Common Divisor) of the numbers. If GCD comes out 1 then numbers are relative prime to each other. Harry requirements are as follows:   1. Procedure **DEC\_IN** should load two registers (BX and DX) with two numbers. Numbers should be a 2 - digit decimal ranging from (01 -   99).   1. Procedure **GCD\_AB** apply the logic for GCD of two numbers. GCD of two numbers is performed by dividing the greater number (in BX) by the smaller number (in DX) till the remainder is zero. If it is zero, the divisor is the GCD if not the remainder and the divisor of the previous division are the new set of two numbers. The process is repeated by dividing greater of the two numbers by the smaller number till the remainder is zero and GCD is found. 2. Also check if,    1. The numbers are equal then GCD would be BX,    2. BX<DX then exchange the contents of 2 register. 3. Procedure **DEC\_OUT** should display the GCD on screen in decimal.       **Sample 1:**  Enter 1st Number: **20** Enter 2nd Number: **09**  GCD is: **1**  **Numbers are relative prime**  **Sample 2:**  Enter 1st Number: **09** Enter 2nd Number: **03**  GCD is: **03**  **Numbers are not relative prime**  Solution:  include Irvine32.inc  .data  str1 byte "enter the first number x ",0  str2 byte "enter the second number y ", 0  x dword 0  y dword 0  .code  main proc  mov edx,offset str1  call writestring  call readdec  mov x,eax  mov edx,offset str2  call writestring  call readdec  mov y, eax  mov eax,x  mov ebx,y  call great\_commen\_diviser  mov x,eax  call writedec  call crlf  CALL WAITMSG  exit  main endp  great\_commen\_diviser proc  mov ecx,-1  start:  mov edx, 0  idiv ebx  mov eax,ebx  mov ebx,edx  cmp ebx,0  jbe jump  loop start  jump:  ret  great\_commen\_diviser endp  end main | **Estimated completion time:30mins** |