|  |  |  |
| --- | --- | --- |
|  | STRING MANIPULATIONS |  |
| Exp No.: 3 |  | **Name:** S Vishakan |
| Date: 09-09-2020 |  | **Reg. No:** 18 5001 196 |

**AIM:**

To write assembly language programs to perform the following basic string operations.

1. To move a string of bytes.
2. To compare two strings of bytes.
3. To search a byte in a string of bytes.
4. To move a string without using string instructions.

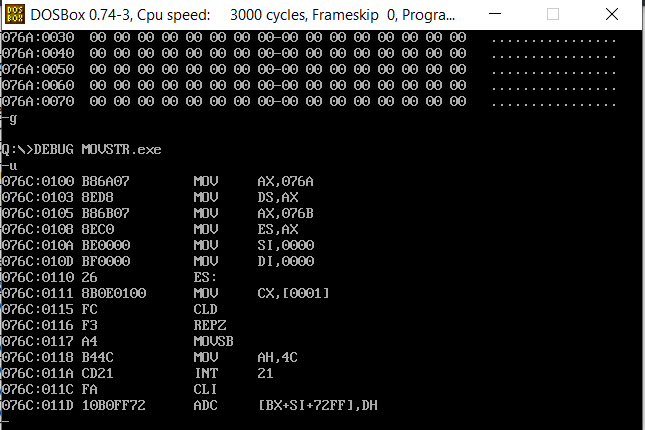
**PROGRAM – 1: MOVING A STRING OF BYTES:**

**ALGORITHM:**

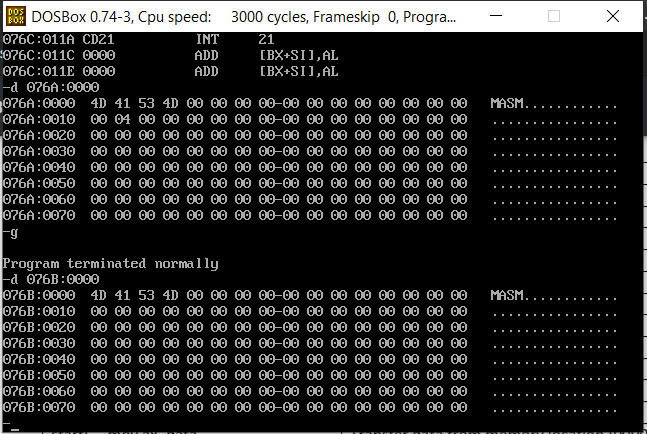
1. Begin.
2. Declare data segment.
3. Initialize data segment with variables for storing the source string and its length.
4. Close the data segment.
5. Declare extra segment.
6. Initialize extra segment with a variable to store the destination string.
7. Close the extra segment.
8. Declare code segment.
9. Set a preferred offset (preferably 100h)
10. Load the data segment content into AX register.
11. Transfer the contents of AX register to DS register.
12. Load the extra segment content into AX register.
13. Transfer the contents of AX register to ES register.
14. Transfer to CX the length of source string.
15. Have SI point to source string and DI to destination string.
16. Clear the direction flag.
17. Repeat until CX is zero:
18. Transfer data from SI to DI using MOVSB.
19. Safely exit the program using an interrupt signal.
20. Close the code segment.
21. End.

|  |  |
| --- | --- |
| **PROGRAM** | **COMMENTS** |
| assume cs:code, ds:data, es:extra | Declare code and data segment. |
|  |  |
| data segment | Initialize data segment with values. |
| str1 db "MASM" | Stores a string STR1. |
| data ends |  |
|  |  |
| extra segment | Initialize extra segment with values. |
| str2 db ? | Declaring a string with no preset value. |
| strlen dw 0004h | Declaring the length of the string STR1. |
| extra ends |  |
|  |  |
| code segment | Start the code segment. |
| org 0100h | Initialize an offset address. |
| start: mov ax, data | Transfer data from “data” to AX. |
| mov ds, ax | Transfer data from memory location AX to DS. |
| mov ax, extra | Transfer the data from “extra” to AX. |
| mov es, ax | Transfer the data AX to ES. |
| mov si, offset str1 | Store the starting offset address of STR1 in SI. |
| mov di, offset str2 | Store the starting offset address of STR2 in DI. |
| mov cx, strlen | Store the length of STR1 in CX. |
| cld | Clear directional flag value. |
| rep movsb | Repeat MOVSB instruction till CX ≠ 0.  MOVSB copies bytes from DS to ES. |
| mov ah, 4ch |  |
| int 21h | Interrupt the process with return code and exit. |
| code ends |  |
| end start |  |

**UNASSEMBLED CODE:**



**SAMPLE I/O SNAPSHOT:**



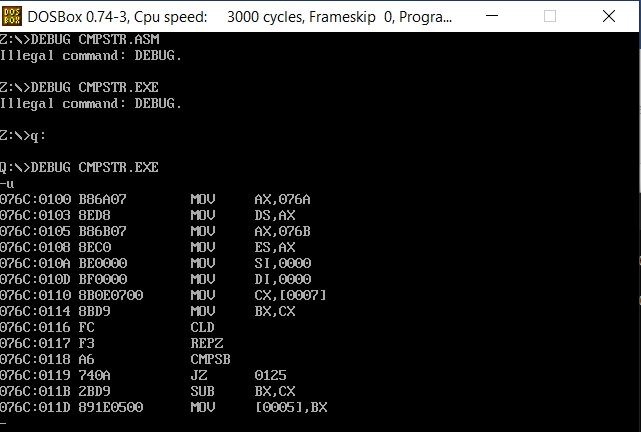
**PROGRAM – 2: COMPARING 2 STRINGS OF BYTES:**

**ALGORITHM:**

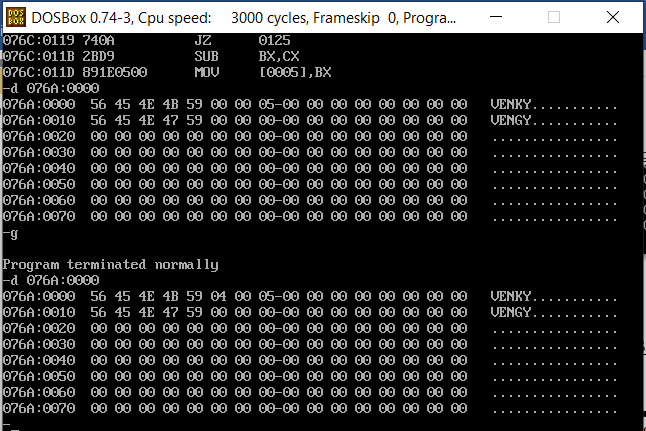
1. Begin.
2. Declare data segment.
3. Initialize data segment with variables for storing the source string and its length.
4. Close the data segment.
5. Declare extra segment.
6. Initialize extra segment with variables for storing the destination and its string.
7. Close the extra segment.
8. Declare code segment.
9. Set a preferred offset (preferably 100h)
10. Load the data segment content into AX register.
11. Transfer the contents of AX register to DS register.
12. Load the extra segment content into AX register.
13. Transfer the contents of AX register to ES register.
14. Transfer to CX the length of source string.
15. Have SI point to source string and DI to destination string.
16. Clear the direction flag.
17. Repeat until CX is zero or a mismatch is found:
18. Compare data in SI and DI using CMPSB, and increment the pointers.
19. If a mismatch is found, find the index of it by subtracting it with the source string’s length.
20. Else, store zero as result. (String equality)
21. Safely exit the program using an interrupt signal.
22. Close the code segment.
23. End.

|  |  |
| --- | --- |
| **PROGRAM** | **COMMENTS** |
| assume cs:code, ds:data, es:extra | Declare code and data segment. |
|  |  |
| data segment | Initialize data segment with values. |
| str1 db "VENKY" | Stores a string STR1. |
| streq dw 0000h | Variable to store the result of the comparison. |
| strlen dw 0005h | Variable to hold the length of STR1. |
| data ends |  |
|  |  |
| extra segment | Initialize extra segment with values. |
| str2 db "VENGY" | Declaring a string STR2. |
| extra ends |  |
|  |  |
| code segment | Start the code segment. |
| org 0100h | Initialize an offset address. |
| start: mov ax, data | Transfer data from “data” to AX. |
| mov ds, ax | Transfer data from memory location AX to DS. |
| mov ax, extra | Transfer the data from “extra” to AX. |
| mov es, ax | Transfer the data AX to ES. |
| mov si, offset str1 | Store the starting offset address of STR1 in SI. |
| mov di, offset str2 | Store the starting offset address of STR2 in DI. |
| mov cx, strlen | Store the length of STR1 in CX. |
| mov bx, cx | Copy the value of CX to BX. |
| cld | Clear directional flag value. |
| repe cmpsb | Repeat CMPSB instruction till ZF = 1.  CMPSB compares bytes of ES to corresponding bytes of DS. |
| jz equstr | Jump to “EQUSTR” if ZF = 0. |
| sub bx, cx | Subtract value of CX from value of BX. |
| mov streq, bx | Transfer the value of BX to variable STREQ. |
| mov ah, 4ch |  |
| int 21h | Interrupt the process with return code and exit. |
|  |  |
| equstr: mov streq, 0000h | Transfer 0 to STREQ indicating STR1 and STR2 are equal. |
| mov ah, 4ch |  |
| int 21h | Interrupt the process with return code and exit. |
| code ends |  |
| end start |  |

**UNASSEMBLED CODE:**



**SAMPLE I/O SNAPSHOT:**



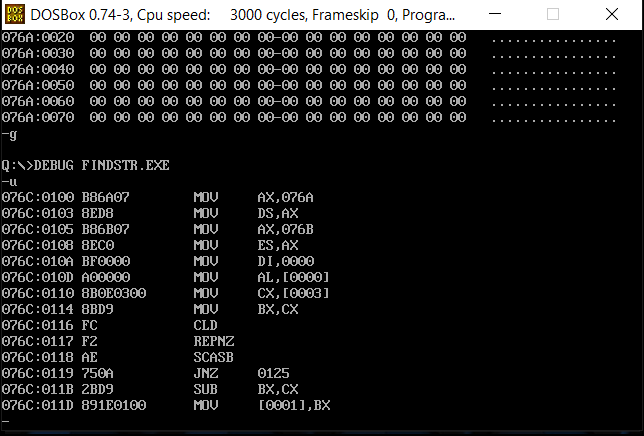
**PROGRAM – 3: SEARCHING A BYTE IN A STRING:**

**ALGORITHM:**

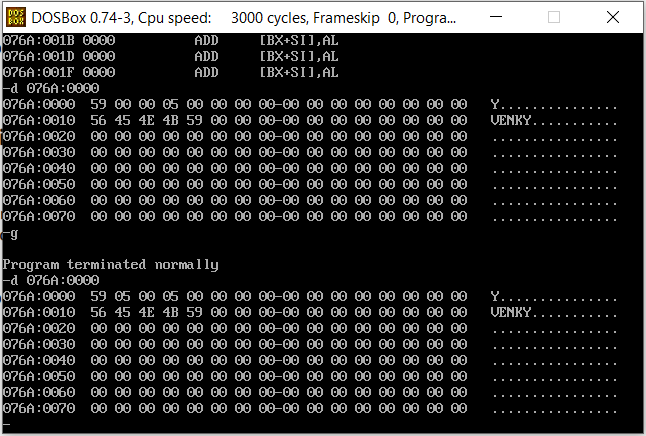
1. Begin.
2. Declare data segment.
3. Initialize data segment with a variable for the byte to be searched and variables to store the location and source string’s length.
4. Close the data segment.
5. Declare extra segment.
6. Initialize extra segment with a variable to store the source string.
7. Close the extra segment.
8. Declare code segment.
9. Set a preferred offset (preferably 100h)
10. Load the data segment content into AX register.
11. Transfer the contents of AX register to DS register.
12. Load the extra segment content into AX register.
13. Transfer the contents of AX register to ES register.
14. Transfer to CX the length of source string.
15. Store in AL the byte to be searched for and in DI the source string.
16. Clear the direction flag.
17. Repeat until CX is zero or until a match is found:
18. Compare data in AL and DI using SCASB, and increment the pointers.
19. If a match is found, find its position by subtracting it with the source string’s length.
20. Else, store zero as result. (Byte not found)
21. Close the code segment.
22. End

|  |  |
| --- | --- |
| **PROGRAM** | **COMMENTS** |
| assume cs:code, ds:data, es:extra | Declare code and data segment. |
|  |  |
| data segment | Initialize data segment with values. |
| str2 db "Y" | Stores a string STR1. |
| strloc dw 0000h | Variable to store the index location. |
| strlen dw 0005h | Variable to hold the length of STR1. |
| data ends |  |
|  |  |
| extra segment | Initialize extra segment with values. |
| str1 db "VENKY" | Declaring a string STR2. |
| extra ends |  |
|  |  |
| code segment | Start the code segment. |
| org 0100h | Initialize an offset address. |
| start: mov ax, data | Transfer data from “data” to AX. |
| mov ds, ax | Transfer data from memory location AX to DS. |
| mov ax, extra | Transfer the data from “extra” to AX. |
| mov es, ax | Transfer the data AX to ES. |
| mov di, offset str1 | Store the starting offset address of STR1 in DI. |
| mov al, str2 | Store STR2 in AL. |
| mov cx, strlen | Store the length of STR1 in CX. |
| mov bx, cx | Copy the value of CX to BX. |
| cld | Clear directional flag value. |
| repne scasb | Repeat SCASB instruction till ZF = 0.  SCASB scans for the occurrence of the byte in AL in ES. |
| jnz notfnd | Jump to “NOTFND” if ZF = 1. |
| sub bx, cx | Subtract value of CX from value of BX. |
| mov strloc, bx | Transfer the value of BX to variable STRLOC. |
| mov ah, 4ch |  |
| int 21h | Interrupt the process with return code and exit. |
|  |  |
| notfnd: mov strloc, 0000h | Transfer 0 to STRLOC indicating STR2 was not found in STR1. |
| mov ah, 4ch |  |
| int 21h | Interrupt the process with return code and exit. |
| code ends |  |
| end start |  |

**UNASSEMBLED CODE:**



**SAMPLE I/O SNAPSHOT:**



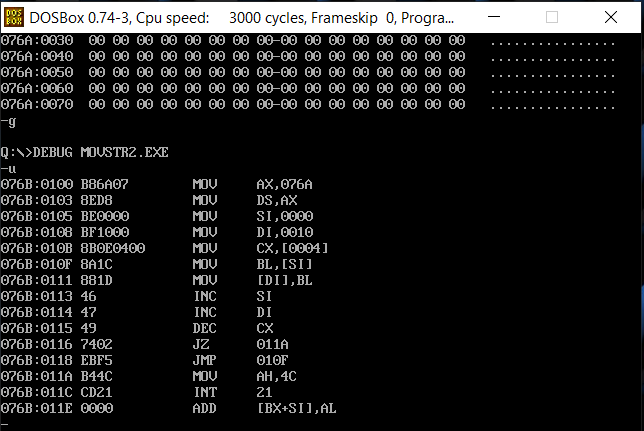
**PROGRAM – 4: MOVING A STRING WITHOUT USING STRING INSTRUCTIONS:**

**ALGORITHM:**

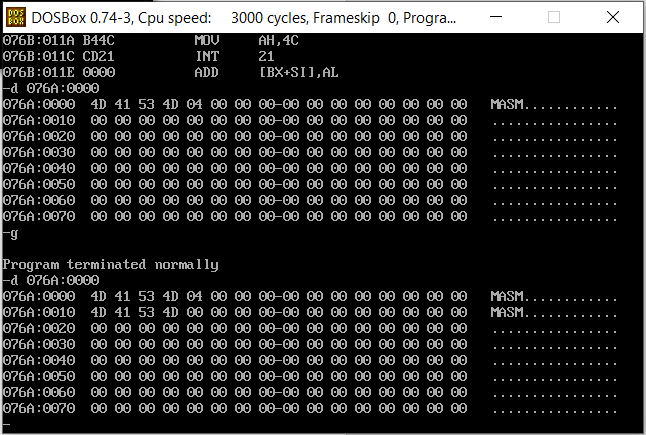
1. Begin.
2. Declare data segment.
3. Initialize data segment with variables for storing the source string and its length.
4. Close the data segment.
5. Declare extra segment.
6. Initialize extra segment with a variable to store the destination string.
7. Close the extra segment.
8. Declare code segment.
9. Set a preferred offset (preferably 100h)
10. Load the data segment content into AX register.
11. Transfer the contents of AX register to DS register.
12. Load the extra segment content into AX register.
13. Transfer the contents of AX register to ES register.
14. Store in CX the length of the source string.
15. Have SI point to the source string and DI to the destination string.
16. Clear the direction flag.
17. Repeat until CX is zero:
    * 1. Transfer data from SI to DI
      2. Increment SI and DI
      3. Decrement CX
18. Safely exit the program using an interrupt signal.
19. Close the code segment.
20. End

|  |  |
| --- | --- |
| **PROGRAM** | **COMMENTS** |
| assume cs:code, ds:data, es:extra | Declare code and data segment. |
|  |  |
| data segment | Initialize data segment with values. |
| str1 db "MASM" | Stores a string STR1. |
| strlen dw 0004h | Variable to hold the length of STR1. |
| str2 db ? | Declaring a string with no preset value. |
| data ends |  |
|  |  |
| code segment | Start the code segment. |
| org 0100h | Initialize an offset address. |
| start: mov ax, data | Transfer data from “data” to AX. |
| mov ds, ax | Transfer data from memory location AX to DS. |
| mov si, offset str1 | Store the starting offset address of STR1 in SI. |
| mov di, offset str2 + 000Ah | Store the starting offset address of STR2 + 000Ah in DI. |
| mov cx, strlen | Store the length of STR1 in CX. |
|  |  |
| looper: mov bl, [si] | Copy the value at SI’s address location to BL. |
| mov [di], bl | Copy the value in BL to DI’s address location. |
| inc si | Increment SI. |
| inc di | Increment DI. |
| dec cx | Decrement CX. |
| jz break | Jump to “BREAK” if CX = 0. |
| jmp looper | Unconditionally jump back to “LOOPER”. |
|  |  |
| break: mov ah, 4ch |  |
| int 21h | Interrupt the process with return code and exit. |
| code ends |  |
| end start |  |

**UNASSEMBLED CODE:**



**SAMPLE I/O SNAPSHOT:**



**RESULT:**

The assembly level programs were written to perform the above specified basic string operations and the output was verified.