Laboratory 7

Title of the Laboratory Exercise: String manipulation

1. Introduction and Purpose of Experiment

   Students will be able to perform all string manipulations in assembly language

2. Aim and Objectives

   Aim

   To develop assembly language program to perform all string operations like inserting a byte, deleting a byte and copying a string as a sub-string

   Objectives

   At the end of this lab, the student will be able to

   - Identify instructions for performing string manipulation
   - Use indexed addressing mode
   - Apply looping instructions in assembly language
   - Use data segment to represent arrays

3. Experimental Procedure

   1. Write algorithm to solve the given problem
   2. Translate the algorithm to assembly language code
   3. Run the assembly code in GNU assembler
   4. Create a laboratory report documenting the work

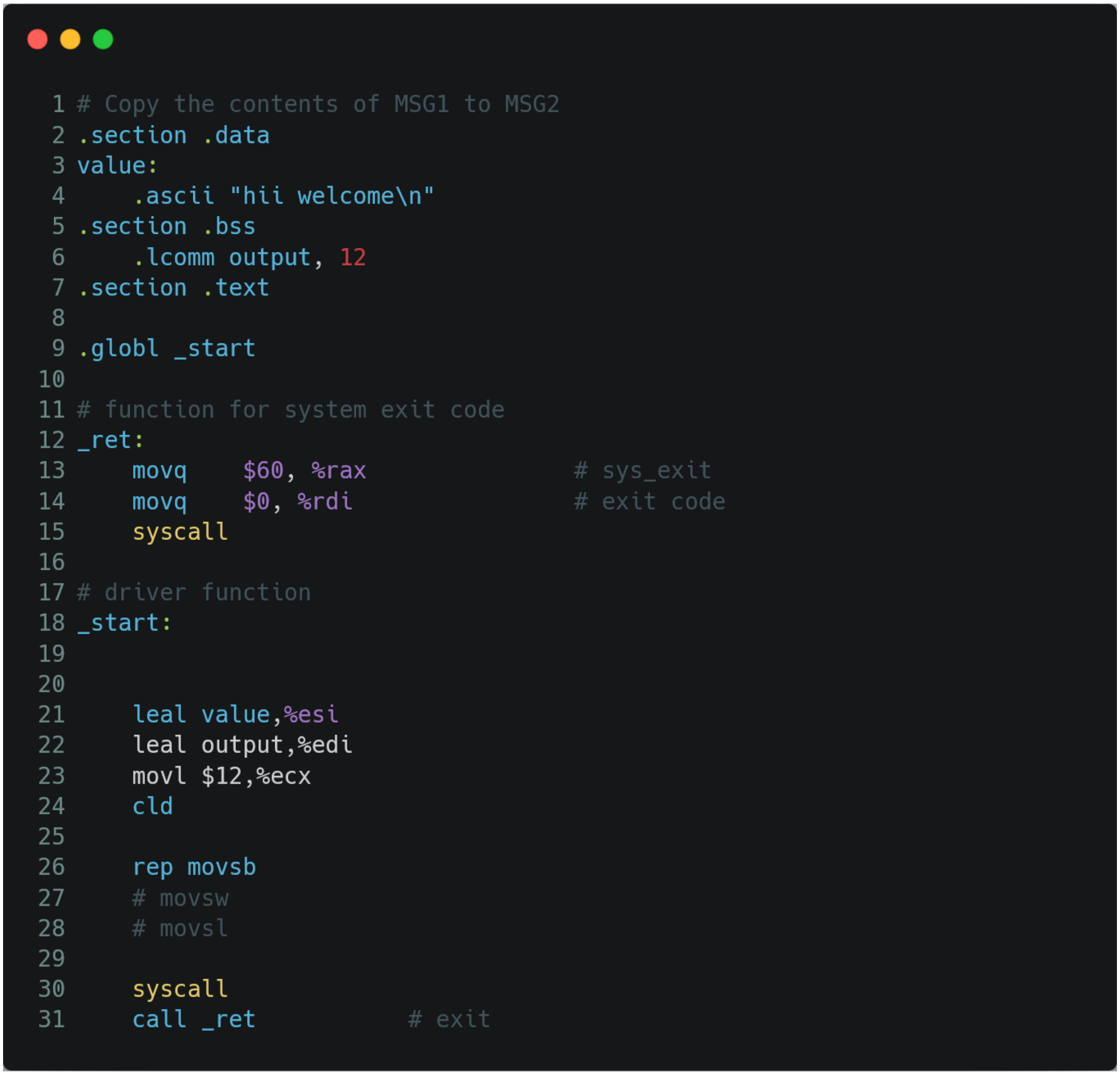
4. Questions

   Develop an assembly language program to perform the following

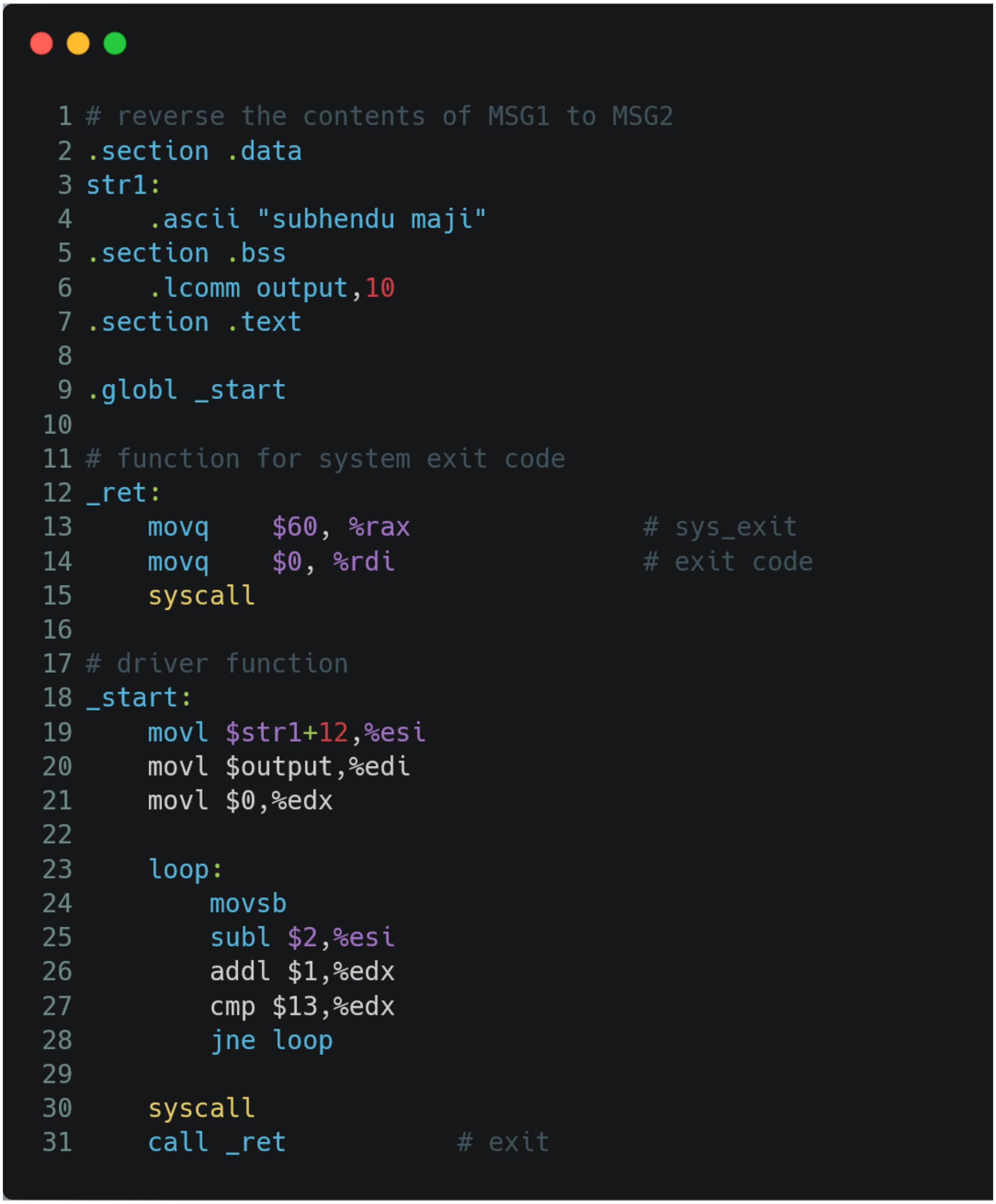
   1. Copy the contents of MSG1 to MSG2
   2. Copy the contents of MSG1 to MSG3 in reverse order

3. Develop an assembly language program to compare two strings and print a message “Equal” if they are equal, “Not Equal” if they are not equal.

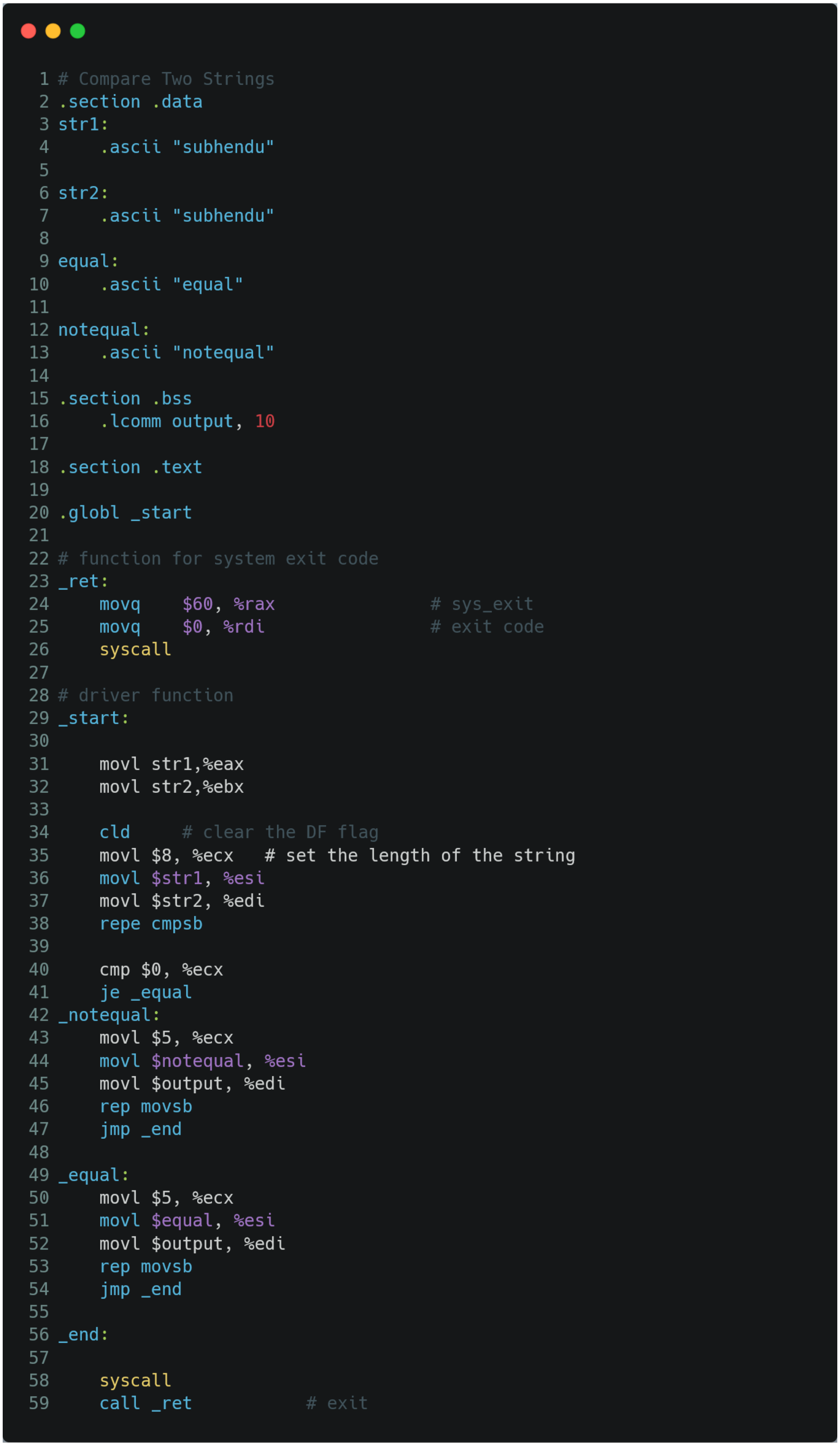
5. Calculations/Computations/Algorithms



```
# Copy the contents of MSG1 to MSG2
.section .data
value:
    .ascii "hii welcome\n"
.section .bss
    .lcomm output, 12
.section .text

.globl _start

# function for system exit code
_ret:
    movq    $60, %rax               # sys_exit
    movq    $0, %rdi                # exit code
    syscall

# driver function
_start:


    leal value,%esi
    leal output,%edi
    movl $12,%ecx
    cld

    rep movsb
    # movsw
    # movsl

    syscall
    call _ret           # exit
```

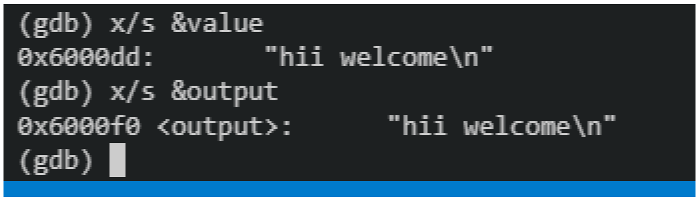


```
# reverse the contents of MSG1 to MSG2
.section .data
str1:
    .ascii "subhendu maji"
.section .bss
    .lcomm output,10
.section .text

.globl _start

# function for system exit code
_ret:
    movq    $60, %rax               # sys_exit
    movq    $0, %rdi                # exit code
    syscall

# driver function
_start:
    movl $str1+12,%esi
    movl $output,%edi
    movl $0,%edx

    loop:
        movsb
        subl $2,%esi
        addl $1,%edx
        cmp $13,%edx
        jne loop

    syscall
    call _ret           # exit
```



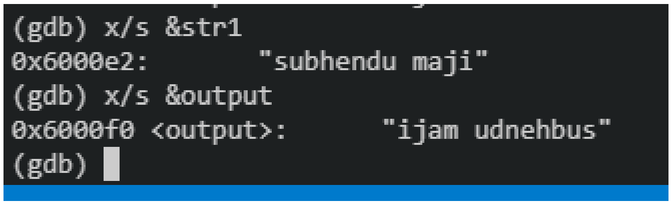
```
# Compare Two Strings
.section .data
str1:
    .ascii "subhendu"

str2:
    .ascii "subhendu"

equal:
    .ascii "equal"

notequal:
    .ascii "notequal"

.section .bss
    .lcomm output, 10

.section .text

.globl _start

# function for system exit code
_ret:
    movq    $60, %rax               # sys_exit
    movq    $0, %rdi                # exit code
    syscall

# driver function
_start:

    movl str1,%eax
    movl str2,%ebx

    cld     # clear the DF flag
    movl $8, %ecx   # set the length of the string
    movl $str1, %esi
    movl $str2, %edi
    repe cmpsb

    cmp $0, %ecx
    je _equal
_notequal:
    movl $5, %ecx
    movl $notequal, %esi
    movl $output, %edi
    rep movsb
    jmp _end

_equal:
    movl $5, %ecx
    movl $equal, %esi
    movl $output, %edi
    rep movsb
    jmp _end

_end:

    syscall
    call _ret           # exit
```

6. Presentation of Results



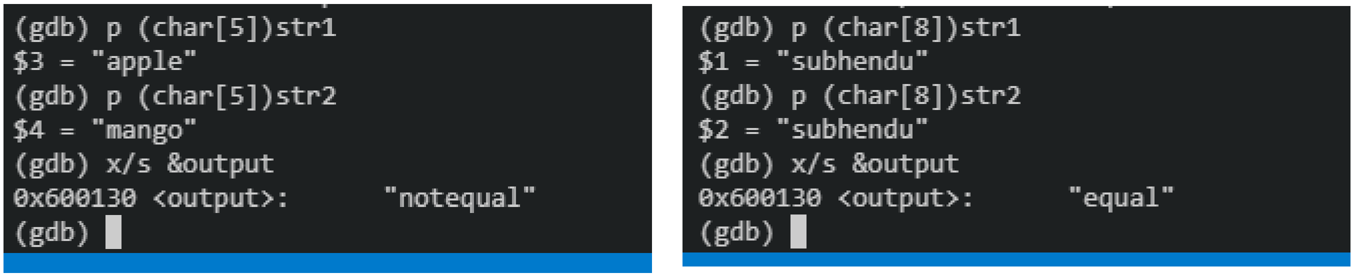
```
(gdb) x/s &value
0x6000dd:       "hii welcome\n"
(gdb) x/s &output
0x6000f0 <output>:      "hii welcome\n"
(gdb)
```

Figure 1 copying contents of msg1 to msg2



```
(gdb) x/s &str1
0x6000e2:       "subhendu maji"
(gdb) x/s &output
0x6000f0 <output>:      "ijam udnehbus"
(gdb)
```

Figure 2 Reversing the string from msg1



```
(gdb) p (char[5])str1
$3 = "apple"
(gdb) p (char[5])str2
$4 = "mango"
(gdb) x/s &output
0x600130 <output>:      "notequal"
(gdb)
```

```
(gdb) p (char[8])str1
$1 = "subhendu"
(gdb) p (char[8])str2
$2 = "subhendu"
(gdb) x/s &output
0x600130 <output>:      "equal"
(gdb)
```

Figure 3 comparing two strings and printing equal or not

7. Analysis and Discussions

8. Code	`movs`
Example	`movsb`
Explanation	Performs: Moves a byte from `esi` to `edi` Description: Moves the byte, word, or doubleword specified with the second operand (source operand) to the location specified with the first operand (destination operand). Both the source and destination operands are located in memory. The address of the source operand is read from the DS:ESI or the DS:SI registers (depending on the address-size attribute of the instruction, 32 or 16, respectively). The address of the destination operand is read from the ES:EDI or the ES:DI registers (again depending on the address-size attribute of the instruction). The

	DS segment may be overridden with a segment override prefix, but the ES segment cannot be overridden.

Code	`rep`
Example	`repe`
Explanation	Performs: Repeat string operation prefix Description: Repeats a string instruction the number of times specified in the count register ((E)CX) or until the indicated condition of the ZF flag is no longer met. The REP (repeat), REPE (repeat while equal), REPNE (repeat while not equal), REPZ (repeat while zero), and REPNZ (repeat while not zero) mnemonics are prefixes that can be added to one of the string instructions. The REP prefix can be added to the INS, OUTS, MOVS, LODS, and STOS instructions, and the REPE, REPNE, REPZ, and REPNZ prefixes can be added to the CMPS and SCAS instructions. (The REPZ and REPNZ prefixes are synonymous forms of the REPE and REPNE prefixes, respectively.) The behavior of the REP prefix is undefined when used with non-string instructions. The REP prefixes apply only to one string instruction at a time. To repeat a block of instructions, use the LOOP instruction or another looping construct.

9. Conclusions

Repeat Prefixes

Repeat Prefix	**Termination Condition 1**	**Termination Condition 2**
REP	ECX=0	None
REPE/REPZ	ECX=0	ZF=0
REPNE/REPNZ	ECX=0	ZF=1

Instruction such as movsb, movsl, are used to move bytes and words from source register to destination register, which are esi and edi respectively.

To repeat an instruction, rep instruction is used, this is used to make a loop like construct the copy strings, and also to compare strings.

10. Comments

1. Limitations of Experiments

The length of the string to be copied has to be known to know how many characters has to be copied.

2. Limitations of Results

The destination memory which is assigned in the uninitialized bss segment is fixed size, hence strings of larger sizes could overflow the memory.

3. Learning happened

The concept of strings and various string operations in assembly is learnt in this lab.

4. Recommendations

The source and destination registers should be carefully taken and the DF flag must be cleared using the cld instruction

Signature and date Marks