**Федеральное агентство связи**

**Государственное бюджетное образовательное учреждение высшего образование**

**Ордена Трудового Красного Знамени**

**«Московский технический университет связи и информатики»**

**Кафедра «Математическая кибернетика и информационные технологии»**

**дисциплина «Микропроцессорные системы»**

Отчет по лабораторной работе №1

Подготовили студенты

группы БВТ1901

Атаев Реджеп

Москва, 2020

**Цель работы:** ознакомиться с технологией написания и отладки программ на языке ассемблера. Ознакомиться со структурой программы и директивами языка. Ознакомиться с операциями пересылки данных, передачи управления и прерываниями ввода-вывода.

**Ход работы:**

Задача №1

.MODEL small ; *модель памяти, используемая для EXE*

.STACK 100h ; *выделение 256 Б для стака*

**.DATA** ; *сегмент данных*

msg1 DB "Enter string: $" ; *строка "Enter string: $" перед вводом строки для программы*

msg3 DB 0Ah, 0Dh, "Result: $" ; *перенос в начало следующей строки и строка "Result: $" перед выводом результата*

str1ml DB 200 ; *переменная str1m1 длиной 200 символов*

str1l DB '$'; *переменная str1l инициализированная символом '$'*

str1 DB 200 dup('$') ; *переменная str1 длиной 200 символов, заполненная символами '$'*

str2ml DB 200 ; *переменная str2m1 длиной 200 символов*

str2l DB '$' ; *переменная str2l инициализированная символом '$'*

str2 DB 200 dup('$') ; *переменная str2 длиной 200 символов, заполненная символами '$'*

max dw ? ; *неинициализированная переменная max*

maxfirst dw ? ; *неинициализированная переменная maxfirst*

**.CODE** ; *сегмент кода*

begin: ; *начало блока begin*

mov ax, @data ; *помещает адрес сегмента data в регистр ax*

mov ds, ax ; *помещает адрес сегмента data из регистра ax в регистр ds*

mov es,ax ; *помещает адрес сегмента data из регистра ax в регистр es*

xor ax,ax ; *происходит очистка регистра ax*

lea dx, msg1 ; *вывод приглашения msg1*

**call strout** ; *вызов процедуры strout*

lea dx, str1ml ; *ввод строки strl для последующих операций*

**call strin** ; *вызов процедуры strin*

mov maxfirst,0 ; *присвоение переменной maxfirst значения 0*

maxlen: ; *метка maxlen*

;-------------------------------------------------- Поиск слова

; *нахождение самого длинного слова*

mov ch,0 ; *старший байт = 0*

mov cl, str1l ; *количество байт (символов) для сканирования*

lea di,str1 ; *адрес строки*

mov al,' ' ; *искомый символ (пробел)*

mov dx,di ; *начало самого длинного слова на данный момент*

mov max,0 ; *длина самого длинного слова на данный момент*

find\_space: ; *метка find\_space*

mov si,di ; *запомнить адрес начала текущего слова*

repne scasb ; *сканировать пока не равно пробелу*

; *вычисление длины слова*

mov bx,di ; *указывает на следующий символ после найденного*

sub bx,si ; *вычесть адрес начала слова*

**jz no\_min**  ; *перейти если длина=0 (подряд 2 пробела)*

cmp cx,0 ; *если просканирована вся строка*

**jz end\_str** ; *перейти на метку*

dec bx ; *реальная длина слова, т.к. di указывал на символ, следующий за пробелом*

**jz no\_min** ; *перейти если длина=0 (подряд 2 пробела)*

end\_str: ; *метка end\_str*

cmp bx,max ; *сравнение длины слова (ВХ) с самым длинным на данный момент*

**jbe no\_min** ; *если меньше максимального, то перейти*

mov max,bx ; *присвоить переменной max (длина самого длинного слова) длину слова*

mov dx,si ; *поместить адрес начала слова в регистр dx*

no\_min: ; *метка no\_min*

cmp cx,0 ; *если просканированна не вся строка*

**jnz find\_space** ; *тогда перейти - продолжить сканирование*

mov si,dx ; *адрес начала слова*

mov bx, max ; *помещаем в регистр bx значение max (длину самого длинного слова)*

cmp bx, 0 ; *если длина самого длинного слова равна 0*

**je exit** ; *перейти к метке exit (завершение программы)*

cmp bx, maxfirst ; *Сравниваем максимальную длину этого витка цикла с первоначально найденной максимальной длиной*

**jb exit** ; *Если меньше, то программа завершает свою работу*

mov maxfirst, bx ; *иначе присваиваем новую максимальную длину (фактически, дублируем её)*

; *Искомое слово в Dx, длина самого длинного слова в Bx*

;(Space)*К удаляемому слову цепляем пробелы, стоящие после него*

space: ; *метка space*

mov cl, byte ptr [si+bx] ; *в регистр цикла cl помещается адрес конца слова*

cmp cl,' ' ; *если символ по адресу, записанному в cl,не равен пробелу*

**jne StartDelete** ; *перейти к метке StartDelete*

inc bx ; *увеличить значение, записанное в bx на 1*

**jmp space** ; *перейти к метке space*

;(space)

;------------------------------------------------- конец поиска слова

StartDelete: ; *метка StartDelete*

mov str2l, bl ; *Устанавливаем длину строки, равную длине найденного слова*

xor cx, cx; *очищаем регистр cx*

mov cl, str1l ; *устанавливается количествово повторений*

sub cl, str2l ; *вычитаем из cl длину строки str2l*

inc cl ; *увеличиваем значение, записанное в cl, на 1*

cld ; *обход вперёд*

mov di, dx ; *в di - строка, которую надо найти*

lea si, str1 ; *в si - строка, в которой ищем*

xor ax, ax ; *очищаем регистр ax*

all\_string: ; *повторяется для str1 до конца строки минус длина str2*

**call sub\_search** ; *вызов процедуры sub\_search*

inc si ; *увеличиваем значение, записанное в si, на 1*

loop all\_string ; *повтор цикла all\_string (уменьшение значения cl и переход на начало блока)*

success: ; *метка success*

;(len) *Делаем значение длины главной строки соответствующее действительности, отнимая длину удалённой строки*

mov bl, str1l ; *передаем регистру bl длину строки str1l*

sub bl, str2l ; *вычитаем из регистра bl длину строки str2l*

mov str1l, bl ; *устанавливаем длину строки strl1, равную bl*

;(len)

**jmp maxlen**; *Переход к дальнейшему поиску (условие выхода в поиске)*

exit: ; *метка exit*

**call nextstr** ; *переход на новую строку*

;(result) *Вывод результата*

lea dx, msg3 ; *помещает значение строки msg3 в регистр dx*

**call strout** ; *вызов процедуры strout*

lea dx, str1 ; *помещает строку str1 в регистр dx*

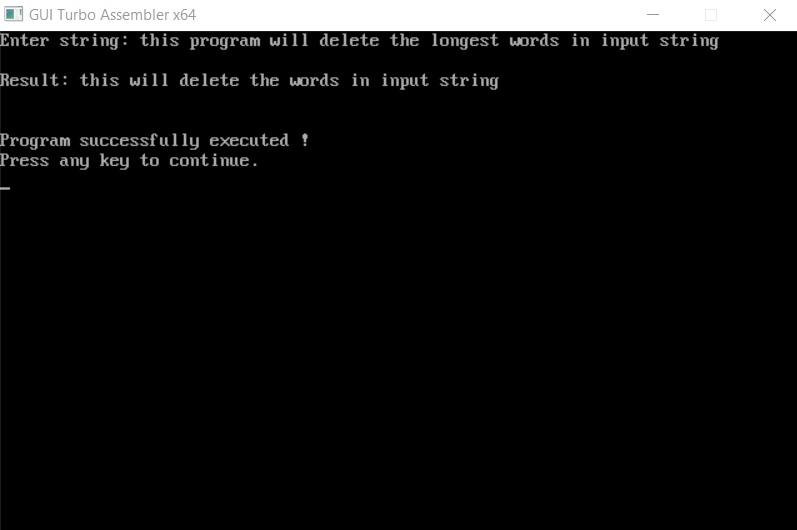
**call strout** ; *вызов процедуры strout*

;(result)

;Конец

mov ah, 4ch ; *помещает команду завершения программы в регистр ah*

int 21h ; *прерывание команды завершения программы*



Данная программа находит длину самого длинного слова в строке, сканируя символы в строке до пробелов (repne scasb) и сравнивая длину полученного слова с максимальной на данный момент (cmp bx,max). Пройдя по всей строке, программа сравнивает длину самого длинного слова в данном витке программы с максимальным по длине (cmp bx, maxfirst), и, если она больше или равна, удаляет данное слово из строки в блоке StartDelete, после чего возвращается на метку maxlen для поиска самого длинного слова в строке после удаления. Если самое длинное слово в текущем витке по длине меньше исходного самого длинного слова, программа выводит новую строку и завершается. Основная часть:

mov ch,0 ; *старший байт = 0*

mov cl, str1l ; *количество байт (символов) для сканирования*

lea di,str1 ; *адрес строки*

mov al,' ' ; *искомый символ (пробел)*

mov dx,di ; *начало самого длинного слова на данный момент*

mov max,0 ; *длина самого длинного слова на данный момент*

find\_space: ; *метка find\_space*

mov si,di ; *запомнить адрес начала текущего слова*

repne scasb ; *сканировать пока не равно пробелу*

; *вычисление длины слова*

mov bx,di ; *указывает на следующий символ после найденного*

sub bx,si ; *вычесть адрес начала слова*

**jz no\_min**  ; *перейти если длина=0 (подряд 2 пробела)*

cmp cx,0 ; *если просканирована вся строка*

**jz end\_str** ; *перейти на метку*

dec bx ; *реальная длина слова, т.к. di указывал на символ, следующий за пробелом*

**jz no\_min** ; *перейти если длина=0 (подряд 2 пробела)*

end\_str: ; *метка end\_str*

cmp bx,max ; *сравнение длины слова (ВХ) с самым длинным на данный момент*

**jbe no\_min** ; *если меньше максимального, то перейти*

mov max,bx ; *присвоить переменной max (длина самого длинного слова) длину слова*

mov dx,si ; *поместить адрес начала слова в регистр dx*

no\_min: ; *метка no\_min*

cmp cx,0 ; *если просканированна не вся строка*

**jnz find\_space** ; *тогда перейти - продолжить сканирование*

mov si,dx ; *адрес начала слова*

mov bx, max ; *помещаем в регистр bx значение max (длину самого длинного слова*

cmp bx, 0 ; *если длина самого длинного слова равна 0*

**je exit** ; *перейти к метке exit (завершение программы)*

cmp bx, maxfirst ; *Сравниваем максимальную длину этого витка цикла с первоначально найденной максимальной длиной*

**jb exit** ; *Если меньше, то программа завершает свою работу*

mov maxfirst, bx ; *иначе присваиваем новую максимальную длину (фактически, дублируем её)*

Задача №2

.MODEL small модель памяти, используемая для EXE

.STACK 100h выделение 256 Б для стака

.DATA сегмент данных

;обьявление строк

msg1 DB "Enter string: $"

msg2 DB 0Ah, 0Dh, "Enter substring to delete: $"

msg3 DB 0Ah, 0Dh, "Result: $"

str1ml DB 200 ;переменная str1m1 длиной 200 символов

str1l DB '$' ;переменная str1l инициализированная символом '$'

str1 DB 200 dup('$') ;переменная str1 длиной 200 символов, заполненная символами '$'

str2ml DB 200 ;переменная str2m1 длиной 200 символов

str2l DB '$' ;переменная str2l инициализированная символом '$'

str2 DB 200 dup('$') ;переменная str2 длиной 200 символов, заполненная символами '$'

.CODE

begin:

mov ax, @data ;помещает адрес сегмента data в регистр ax

mov ds, ax ;помещает адрес сегмента data из регистра ax в регистр ds

mov es,ax ;помещает адрес сегмента data из регистра ax в регистр es

xor ax,ax ;происходит очистка регистра ax

lea dx, msg1 ;вывод приглашения msg1

call strout ;вызов процедуры strout

lea dx, str1ml ;ввод строки str1

call strin ;вызов процедуры strin

lea dx, msg2 ;вывод приглашения msg2

call strout ;вызов процедуры strout

lea dx, str2ml ;ввод строки str2

call strin вызов процедуры strin

xor cx, cx ;происходит очистка регистра сx

mov cl, str1l ;устанавливается кол-во повторений

sub cl, str2l ;вычитание str2l из регистра cl

inc cl ;увеличивает значение cl на 1

cld ;обход вперёд

lea di, str2 ;в di - строка, которую надо найти

lea si, str1 ;в si - строка, в которой ищем

xor ax, ax ;очищаем ax

all\_string: ;повторяется для str1 до конца строки минус длина str2

call sub\_search ;вызов процедуры sub\_search

inc si ;увеличивает значение si на 1

loop all\_string ;Повторить цикл

call nextstr ;вызов процедуры nextstr

;xor dx, dx ;очистить dx

;mov dl, al ;поместить в dl al

;add dl, 30h ;вывести ее на экран

;mov ah, 02h ; устанавливает позицию курсора

;int 21h ;прерывание DOS

lea dx, msg3 ;адрес выводимой строки msg3

call strout ;вызов процедуры strout

lea dx, str1 ;адрес выводимой строки str1

call strout ;вызов процедуры strout

\_end:

mov ah, 4ch ;конец программы

int 21h ;прерывание DOS

;\*\*\*\*\*\*Процедуры\*\*\*\*\*\*\*\*  
nextstr proc ;Начало процедуры  
push dx ;добавляет dx в стек  
push ax ;добавляет ax в стек  
mov dl, 0Dh ;перевод на другую строку  
mov ah, 02h устанавливает позицию курсора  
int 21h ;прерывание  
mov dl, 0Ah ;ввод строки в буфер   
mov ah, 02h устанавливает позицию курсора  
int 21h ;прерывание  
pop ax ; извлекает ax из стека  
pop dx ; извлекает dx из стека  
ret ;возврат из подпрограммы  
nextstr endp ;конец процедуры  
;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
strin proc ;Начало процедуры  
mov ah, 0Ah ;вызов функции ввода с клав

int 2эхээж1h ;прерывание  
ret ;возврат из подпрограммы  
strin endp ;конец процедуры  
;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
strout proc ;Начало процедуры  
mov ah, 09h Вывод строки  
int 21h ;прерывание  
ret ;возврат из подпрограммы  
strout endp ;конец процедуры  
;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
sub\_search proc ;Начало процедуры  
push cx ;добавляет cx в стек  
push di ;добавляет di в стек  
push si ;добавляет si в стек  
mov bx, si поместить в dx si  
mov cl, str2l ;поместить в cl str2l  
repe cmpsb ; последовательное сравнение в цикле символов двух строк до первых несовпадающих символов или до конца строк  
je \_eq ;перейти к метке \_eq  
jne \_neq ;перейти к метке \_neq  
\_eq:  
;di указывает на строку 2, si указывает на конец слова, которое надо удалить  
call delete вызов процедуры delete  
inc al ;увеличить значение al на 1  
\_neq:  
pop si ; извлекает si из стека  
pop di ; извлекает di из стека  
pop cx ; извлекает сx из стека  
ret ;возврат из подпрограммы  
sub\_search endp ;конец процедуры  
;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
delete proc ;Начало процедуры  
push bx ;добавляет bx в стек  
push di ;добавляет di в стек  
push si ;добавляет si в стек  
mov di, bx ;в di - адрес начала удаляемого слова  
xor cx, cx ;обнуляет  
mov cl, str1l помещает в cl str1l  
repe movsb вставка 6 символов в строку  
pop si ; извлекает si из стека  
pop di ; извлекает di из стека  
pop bx ; извлекает bx из стека  
ret ;возврат из подпрограммы  
delete endp ;конец процедуры

end begin



Программа в цикле начиная с первого символа до конца строки минус длина str2 сравнивает строку str2 последовательно сравнивает имволы строк, при полнм совпадении удаляя набор символов str2 из строки str1  
Основной блок программы:

all\_string: ;повторяется для str1 до конца строки минус длина str2  
call sub\_search ;вызов процедуры sub\_search (основное действие)  
inc si ;увеличивает значение si на 1  
loop all\_string ;Повторить цикл

Задача №3

.model small ;модель памяти, используемая для EXE

.386 ; Директива для того, чтобы ассемблер правильно транслировал команды с 32-разрядными операндами

dos\_inp macro lbl, sz ; оформление структуры текста для ввода, как макрос

ib struc ; начало структуры

; поля структуры

max db sz ; наибольшая длина строки

len db ? ; сюда вернется настоящая длина

bf db sz dup(?) ; тут будет то, что ввели

ib ends ; конец структуры

lbl ib <> ; создание экземпляра структуры

endm ; конец

.stack 100 выделение 256 Б

.data ; сегмент данных

req db 'Enter string:$' ;строка "Enter string: $" перед вводом строки для программы

dos\_inp buf, 80 ; сюда введется строка

.code ; код программы

start: ; начало

mov ax, @data ; помещает идентификатор сегмента data в регистр ax

mov ds, ax ; помещает идентификатор сегмента data из регистра ax в регистр ds

mov es,ax ; помещает идентификатор сегмента data из регистра ax в регистр es

; вывод на экран строки "Enter string: $"

mov ah, 09h ;Функция DOS вывод строки

lea dx, req ;адрес выводимой строки

int 21h ; обращение к функции DOS

; ввод строки

mov ah, 0Ah ;Функция DOS ввод строки

lea dx, buf ;адрес буфера для ввода

int 21h ; обращение к функции DOS

movzx cx, byte ptr buf.len ; Копирует длину byte ptr buf в cx c дополнением нулевыми битами

lea si, buf.bf ;помещает адрес строки в si

mov di, si ; помещает si в di

@loop: lodsb ;основной цикл

cmp al, '0' ; Сравнение al и '0'

jb @non\_num ; если меньше то выполняется @non\_num

cmp al, '9' ; Сравнение al и '9'

ja @non\_num ; если меньше то выполняется @non\_num

jmp @next ; Безусловная передача управления циклу @next

@non\_num: ;цикл @non\_num

stosb ; Запись байта в строку

@next: loop @loop ;цикл @next

mov al, '$' ; перемещение $ в регистр al

stosb ; Запись байта в строку

mov bx, dx ; перемещение значения из регистра dx в bx

mov word ptr [bx], 0D0Ah ;будет записываться в слово, на которое указывает bx

; Вывод на экран

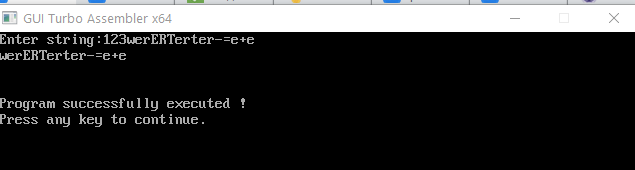
mov ah, 09h ;Функция DOS вывод строки

int 21h ; обращение к функции DOS

mov ah,04Ch ;Функция DOS закрытия программы

int 21h ; обращение к функции DOS

end start ; конец программы



Данная программа отвечает за удаление цифр из строки, это выполняется с помощью цикла, в котором проверяется, является ли символ цифрой, если нет, то записывается в результат. Основной блок кода:

movzx cx, byte ptr buf.len ; Копирует длину byte ptr buf в cx c дополнением нулевыми битами

lea si, buf.bf ;помещает адрес строки в si

mov di, si ; помещает si в di

@loop: lodsb ;основной цикл

cmp al, '0' ; Сравнение al и '0'

jb @non\_num ; если меньше то выполняется @non\_num

cmp al, '9' ; Сравнение al и '9'

ja @non\_num ; если меньше то выполняется @non\_num

jmp @next ; Безусловная передача управления циклу @next

@non\_num: ;цикл @non\_num

stosb ; Запись байта в строку

@next: loop @loop ;цикл @next

mov al, '$' ; перемещение $ в регистр al

stosb ; Запись байта в строку

mov bx, dx ; перемещение значения из регистра dx в bx

mov word ptr [bx], 0D0Ah ;будет записываться в слово, на которое указывает bx

Задача №4

.model small ;определяем модель памяти записи в один стек

.stack 100h ; задаем размер стека в 256 байт для хранения данных.

.data ; указываем начало сегмента даты, в котором размещаем переменные памяти.

string db 'edcbabbffccfgaabrrzz', 0AH, '$' ; строка, над которой производятся операции.

len dw 20 ; длина строки.

.code ; указываем начала сегмента кода, где требуется поместить инструкции.

start: ; метка начала выполнения программы.

mov ax, DGROUP ; Загрузить AX с непосредственным значением.

mov ds, ax ; перенос строки в Data Segment.

mov dx, [len] , записываем количество элементов регистра DX

mov cx, dx ; перенос длины строки в регистр cx (регистр для циклов).

jmp lb3 ; произвести переход на цикл lb3.

lb2: ; меняет символы местами.

dec di ; уменьшаем значение в di на 1.

z ;

inc di ;увеличиваем занчение в регистре di на 1.

mov [string[di]], al ;

loop lb ; вернуться к циклу lb.

lb3: ; основной цикл.

mov di, 0 ;указываем регистром di на 0 ячейку памяти

mov dx, cx ;записываем в регистр dx значение из регистра cx.

mov cx, [len] ; количество элементов массива = число повторений цикла.

sub cx, 1 ; вычитаем из регистра cx 1.

lb: ; цикл сравнивает поэлементно n и n + 1.

mov al, string[di] ; запись элемента n в регистр al.

inc di ; увеличиваем занчение в регистре di на 1.

mov bl, string[di] ; запись элемента n+1 в регистр bl.

cmp al, bl ; сравнение al и bl.

jg lb2 ; если al > bl по ASCII произвести переход на lb2.

loop lb ; снова уменьшаем значение в регистре cx и возвращаемся в точку lb.

mov cx, dx ; складываем данные из dx с данными в cx, cx будет перезаписана результатом.

loop lb3 ; вывод возвращаемся к началу цикла lb3.

mov ah, 9 ; записываем в регистр ah число 9.

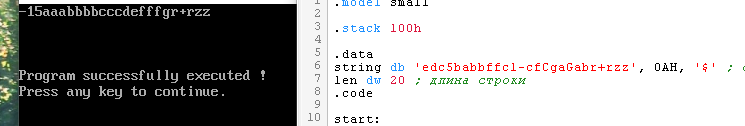
mov dx, offset string

int 21h ;прерывание DOS

mov ax,4C00h ; DOS функция выхода из программы.

int 21h ; операция прерывания для завершения программы.

end start ; конец программы и сообщение программе что начинать выполнение программы надо с метки START.



Данная программа отвечает за сортировку символов строки по ASCII, происходит поиск по регистру и сравнение между собой, затем передача по абсолютному значению. Основной блок программы :

jmp lb3 ; произвести переход на цикл lb3.

lb2: ; меняет символы местами.

dec di ; уменьшаем значение в di на 1.

inc di ;увеличиваем занчение в регистре di на 1.

mov [string[di]], al ;

loop lb ; вернуться к циклу lb.

lb3: ; основной цикл.

mov di, 0 ;указываем регистром di на 0 ячейку памяти

mov dx, cx ;записываем в регистр dx значение из регистра cx.

mov cx, [len] ; количество элементов массива = число повторений цикла.

sub cx, 1 ; вычитаем из регистра cx 1.

lb: ; цикл сравнивает поэлементно n и n + 1.

mov al, string[di] ; запись элемента n в регистр al.

inc di ; увеличиваем занчение в регистре di на 1.

mov bl, string[di] ; запись элемента n+1 в регистр bl.

cmp al, bl ; сравнение al и bl.

jg lb2 ; если al > bl по ASCII произвести переход на lb2.

loop lb ; снова уменьшаем значение в регистре cx и возвращаемся в точку lb.

mov cx, dx ; складываем данные из dx с данными в cx, cx будет перезаписана результатом.

loop lb3 ; вывод возвращаемся к началу цикла lb3.

Задача №5

locals - директива, разрешающая локальные идентификаторы

DOSSEG - разрешает переопределение сегментов DOS во время компоновки

.model small - сообщает ассемблеру, что в данной программе нужно исп. не более 64 Кбайт кодов и не более 64 Кбайт данных

.stack 100h - объявляем сегмент Стека размером 100h

.data - объявляем сегмент данных

len db 200 - длина строки

res db 0 - резервный байт (для чтения)

string db 200 dup('$')- сама строка

number db 'number$' - строка 'number' с '$' на конце - определяет конец строки

.code ;процедура сдвига строки на 6 байтов вправо (для того чтобы вставить на это место 'number') в si - адрес строки с нужного места

shift proc near - задаем процедуру shift типа near

cld - прямой порядок работы со строковыми инструкциями (типа stosb)

mov di, si - содержимое si перемещается в di

xor cx, cx - операция искл. ИЛИ

@@1: - метка

mov bl, byte ptr [si] - bl получит значение si

cmp bl, '$' - сравнение bl и '$'

je @@2 - если просканирована вся строка: переход к метке @@2

inc cx - cx++

inc si - si++

jmp @@1 - перейти на следующий идентификатор @@

@@2:- метка

inc cx - cx++

mov di, si - содержимое si перемещается в di

add di, 6 - прибавление

@@3: - метка

mov bl, byte ptr [si] - bl получит значение si

mov byte ptr [di], bl - di получит значение bl

dec si - уменьшает si на единицу

dec di - уменьшает di на единицу

loop @@3 - повторение цикла

ret - прерывание команды

shift endp - конец процедуры shift

copy proc near - процедура copy

cld - очищает флаг в направлении

lea si, number - в di - адрес строки с нужного места

mov cx, 6 - число 6 перемещается в регистр cx

rep movsb - копирование строк байтов

ret - прерывание команды

copy endp - конец процедуры copy

find proc near - начало процедуры find

mov di, si - содержимое si перемещается в di

mov bl, 1 - число 1 перемещается в регистр bl

@@start: - метка

mov ah, byte ptr[si] - ah получит значение si

cmp ah, 13 - сравниваем ah и 13

je @@4 - если просканирована вся строка: переход к метке @@4

cmp ah, ' ' - сравниваем ah и ' '

je @@1 - если просканирована вся строка: переход к метке @@1

jmp @@3 - перейти на следующий идентификатор @@

@@1:

cmp bl, 1 - сравниваем bl и 1

je @@2 - если просканирована вся строка: переход к метке @@2

mov bl, 1 - число 1 перемещается в регистр bl

inc si - si++

mov di, si - содержимое si перемещается в di

jmp @@3 - перейти на следующий идентификатор @@

@@2: - метка

mov ax, di - содержимое di перемещается в ax

ret - прерывание команды

@@4: - метка

cmp bl, 1 - сравниваем bl и 1

je @@2 - если просканирована вся строка: переход к метке @@2

mov ax, 0 - число 0 перемещается в регистр ax

ret - прерывание команды

@@3: - метка

mov ah, byte ptr[si] - ah получит значение si

cmp ah, '0' - сравниваем ah и '0'

jl @@5 - выполняет короткий переход, если первый операнд МЕНЬШЕ второго операнда при выполнении операции

сравнения с помощью команды CMP.

cmp ah, '9' - сравниваем ah и '9'

jg @@5 - выполняет короткий переход, если первый операнд БОЛЬШЕ второго операнда при выполнении операции

сравнения с помощью команды CMP.

inc si -si++

jmp @@start - перейти на следующий идентификатор @@

@@5: - метка

mov bl, 0 - число 0 перемещается в регистр bl

inc si -si++

jmp @@start - перейти на следующий идентификатор @@

find endp - конец процедуры find

newline proc near - процедура «newline»

mov dl,10 - число 10 перемещается в регистр dl

mov ah,2 - номер функции DOS в АН

int 21h - вывод символа

ret - прерывание команды

newline endp - ввод строки

input proc near - процедура input

mov ah, 0Ah

int 21h - считать строку символов в буфер

ret - прерывание команды ввода строки

input endp - конец процедуры input

output proc near - процедура output

mov ah, 9 - число 9 перемещается в регистр ah

int 21h - вывести приглашение ко вводу message1

ret - прерывание команды вывода строки

output endp - конец процедуры output

do proc near - основной алгоритм

lea si, string - в si - адрес строки

@@1: - метка

push si - кладем si в стек

call find - вызываем функцию find - ищем первую позицию числа

pop si - забираем si из стека

cmp ax, 0 - сравниваем ax с 0 (ax - возвращаемое значение функции find) если ноль - завершаем операцию

je @@2 - если просканирована вся строка: переход к метке @@2

mov si, di - содержимое si перемещается в di

push di - кладем di в стек

call shift - вызов процедуры shift

pop di - забираем di из стека

call copy - копируем 'number' в нужную часть строки

lea si, string - в si - адрес строки

jmp @@1 - перейти на следующий идентификатор @@

@@2: - метка

ret - прерывание команды

do endp - конец процедуры do

start proc near - процедура «start»

mov ax, @data - помещает адрес сегмента @data в регистр ax

mov ds, ax - помещает адрес сегмента data из регистра ax в регистр ds

mov es, ax - помещает адрес сегмента data из регистра ax в регистр es

xor ax, ax - искл. ИЛИ

mov dx, offset len

call input - вызов процедуры input

call do - вызов основного алгоритма

call newline - вызов процедуры newline

mov dx, offset string - вывод полученной строки

call output- ;ожидание ввода символа

mov ah, 0 - перемещает число 0 в регистр ah

int 16h - прерывание команды

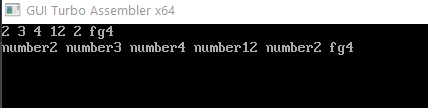
mov ah, 4Ch - помещает команду завершения программы в регистр ah

mov al, 0 - перемещает число 0 в регистр al

int 21h - прерывание команды завершения программы

start endp - конец кода процедуры «start»

end start - конец программы «start»



Программа ищет слова в строке, и если это число, сдвигает стоку из даной позиции на 6 байтов и вставляет туда «number» Основное действие программы:

mov dx, offset len ; ввод строки  
call input ; вызов процедуры input - ввод строки  
call do ; вызов процедуры do - основной алгоритм  
call newline ; вызов процедуры newline - перевод каретки на новую строку  
mov dx, offset string ; вывод полученной строки

Задача №6

.model small ; программа будет скомпилирована в .exe

.data ; сегмент данных

string db 250,?, 256 dup ('$') ; строка для хранения данных

.code ; сегмент кода

start: ; процедура start

mov ax,@data ; помещение в регистр DS сегмента данных, но так как мы не можем напрямую обратится к регистру DS, то мы делаем это через регистр AX

mov ds,ax ; помещение в регистр DS сегмента данных

mov ax,0b800h ; загружаем в 16-битный регистр данных, соответствующий сегменту дисплея в тестовом режиме

mov es,ax ; загрузка числа напрямую в сегментный регистр

mov ax,3 ; смена цвета символа на красный для итоговой строки

int 10h ; установление регистра палитры

mov ah,10 ; вызов функции ввода

lea dx,string ; записываем введённые данные в string

int 21h ; прерывание функции dos

lea di,string+1 ; записываем в di адрес начала слова в строке

mov si,160 ; перевод каретки на указанную позицию

xor cx,cx ; обнуляем cx

mov cl,byte ptr [di] ; помещаем в cl значение di с указанием участия однобайтовой ячейки памяти

add di,cx ; прибавляем к значению di значение cx

inc cx ; прибавляем к cx 1

mov dx,cx ; заносим в dx значение cx

mov al,' ' ; помещаем в al пробел, т.е. разделитель слов

deg: ; метка deg

push ds ; помещаем значение ds в стек

pop es ; извлекаем значение из стека в es

std ; устанавливаем флаг направления DF в единицу

repnz scasb ; поиск ячейки с заданным по адресу es:di байтом

push di ; заносим значение di в стек

xchg dx,cx ; заносим в dx значение cx, и наоборот - заносим в cx значение dx

sub cx,dx ; вычитаем из значения cx значение dx и заносим его в cx

add di,2 ; прибавляем 2 к значению di

dec cx ; уменьшение cx на единицу

mov ax,0b800h ; загружаем в 16-битный регистр данных, соответствующий сегменту дисплея в тестовом режиме

mov es,ax ; загрузка числа напрямую в сегментный регистр

cld ; сброс флага направления DF

mov ah,12 ; установка красного цвета в качестве выходного

xchg si,di ; заносим в di значение si, и наоборот - заносим в si значение di

begs: ; метка begs

lodsb ; копируем один байт из памяти по адресу DS:SI в регистр AL (копируем символ слова из исходной строки)

stosw ; сохраняем регистр AX в ячейке памяти по адресу ES:DI (охраняем символ в новой строке)

loop begs ; если значение cx не равно нулю (т.е. мы находимся не в конце слова), то переходим к началу метки begs

mov al,' ' ; помещаем в al пробел, т.е. разделитель слов

stosw ; сохраняем регистр AX в ячейке памяти по адресу ES:DI (записываем пробел в новой строке)

mov si,di ; помещаем значение di в si

pop di ; помещаем в di значение из стека (находим начало следующего слова для записи в новую строку)

mov cx,dx ; помещаем значение dx в cx

or cx,cx ; если значение cx не равно нулю,

jnz deg ; то перейти к метке deg

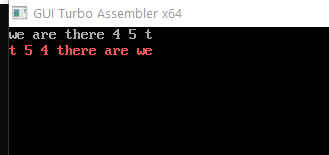
mov ah,10h ; вызов функции ввода с клавиатуры (для того, чтобы мы могли увидеть результат в консоли)

int 16h ; прерывание DOS

mov ah,4ch ; вызов функции завершения программы

int 21h ; прерывание DOS

end start ; окончание процедуры start



В данной программе происходит реверс слов в строке, программа сканирует исходную строку с конца (std) до пробела (repnz scasb), после чего выводит в новую строку слово, начиная с найденного индекса, посимвольно с помощью алгоритма  
begs:  
lodsb  
stosw  
loop begs  
далее, если выведены не все слова исходной строки, сканирование продолжается с конца следующего в обратном порядке слова, пока не будет выведена вся новая строка.

Задача №7

locals ; директива для объявления локальных переменных в процедурах

DOSSEG ; сегменты будут сгруппированы в соответствии с соглашениями по упорядочиванию сегментов фирмы Microsoft

.model small ; программа будет скомпилирована в .exe

.stack 100h ; выделение памяти для программы (256 байт)

.data ; начало сегмента данных

len db 200 ; длина строки

res db 0 ; резервный байт (для чтения)

string db 200 dup('$') ; переменная для хранения строки

number db 'number$' ; подстрока "number" для вставки в строку

.code

shift proc near ; процедура сдвига строки на 6 байтов вправо (для того чтобы вставить на жто место 'number')

cld ; очистить флаг DF

mov di, si ; поиск адреса нового слова в строке

xor cx, cx ; обнуление cx

@@1: ; метка @@1 (в нашем случае выполняет роль цикла while)

mov bl, byte ptr [si] ; переопределение типа данных bl в si

cmp bl, '$' ; если символ bl обозначает конец строки

je @@2 ; то переходим к метке @@2

inc cx ; иначе инкрементируем значение cx

inc si ; инкрементируем значение ci

jmp @@1 ; переходим к метке @@1 (т.е. начинаем новый проход по циклу)

@@2: ; метка @@2

inc cx ; инкерементируем значение cx

mov di, si ; поиск адреса нового слова в строке

add di, 6 ; адресация к адресу di+6

@@3: ; метка @@3

mov bl, byte ptr [si] ; переопределение типа данных bl в si

mov byte ptr [di], bl ; занесение щначения bl в di с переопределением типа данных

dec si ; декремент si

dec di ; декремент di

loop @@3 ; продолжаем цикл метки

ret ; операнд возврата

shift endp ; окочание процедуры shift

copy proc near ; процедура копирования 'number' в строку (в di - адрес строки с нужного места)

cld ; очистить флаг DF

lea si, number ; подготовка вставки 'number' в строку

mov cx, 6 ; определение 6 байтов для вставки в строку

rep movsb ; вставка 6 символов в строку

ret ; операнд возврата

copy endp ; окончание процедуры copy

Iurii Nesterov, [15.11.20 23:35]

find proc near ; процедура поиска числа

mov di, si ; перенос в di адреса строки с нужного места

mov bl, 1 ; занесение единицы в bl

@@start: ; метка @@start

mov ah, byte ptr[si] ; переопределение типа данных ah в si

cmp ah, 13 ; если ah = 13,

je @@4 ; то переходим к метке @@4

cmp ah, ' ' ; если ah = ' ',

je @@1 ; то переходим к метке @@1

jmp @@3 ; иначе переходим к метке @@3

@@1: ; метка @@1

cmp bl, 1 ; если bl = 1,

je @@2 ; то переходим к метке @@2

mov bl, 1 ; заносим единицу на bl

inc si ; инкрементируем si

mov di, si ; заносим знаение si в di

jmp @@3 ; безусловный переход к метке @@3

@@2: ; метка @@2

mov ax, di ; заносим значение di в ax

ret ; операнд возвращения

@@4: ; метка @@4

cmp bl, 1 ; если bl = 1,

je @@2 ; то переходим к метке @@2

mov ax, 0 ; обнялуем регистр ax

ret ; операнд возвращения

@@3:

mov ah, byte ptr[si] ; переопределение типа данных ah в si

cmp ah, '0' ; если в ah лежит символ, чей ASCII меньше, чем '0'

jl @@5 ; то переходим к метке @@5

cmp ah, '9' ; если в ah лежит символ, чей ASCII больше, чем '9'

jg @@5 ; то переходим к метке @@5

inc si ; инкрементируем si

jmp @@start ; безусловный переход к метке @@start

@@5: ; метка @@5

mov bl, 0 ; обнуляем bl

inc si ; инкрементируем si

jmp @@start ; безусловный переход к @start

find endp ; окончание процедуры find

newline proc near ; процедура перехода на новую линию

mov dl,10 ; заносим значение 10 в dl

mov ah,2 ; заносим значение 2 в регистр ah

int 21h ; обращение к функции DOS

ret ; операнд возвращения

newline endp ; окончание процедуры

input proc near ; процедура ввода строки

mov ah, 0Ah ; вызов функции ввода с к лавиатуры

int 21h ; обращение к функции DOS

ret ; операнд возвращения

input endp ; окончание процедуры

Iurii Nesterov, [15.11.20 23:35]

output proc near ; процедура вывода строки

mov ah, 9 ; номер функции DOS для вывода строки

int 21h ; обращение к функции DOS

ret ; операнд возвращения

output endp ; окончание процедуры

do proc near ; процедура с основным алгоритмом программы

lea si, string ; заносим адрес строки в si

@@1: ; метка @@1

push si ; заносим si в стек

call find ; вызываем функцию find - ищем первую позицию числа

pop si ; извлекаем значение si из стека

cmp ax, 0 ; если ax == 0 (ax - возвращаемое значение функции find)

je @@2 ; то переходим к метке @@2

mov si, di ; иначе сдвигаем строку с позиции ax на 6 символов вправо (с помощью функции shift)

push di ; заносим di в стек

call shift ; вызываем функцию shift

pop di ; извлекаем значение di из стека

call copy ; вывзываем функцию copy - копируем 'number' в нужную часть строки

lea si, string ; заносим адрес строки в si

jmp @@1 ; начинаем сначала метки (цикл)

@@2: ; метка @@2

ret ; операнд возвращения

do endp ; окончание процедуры

start proc near ; процедура start. Главная функция (точка входа)

mov ax, @data ; пересылка адреса сегмента данных в регистр AX

mov ds, ax ; установка регистра DS на сегмент данных

mov es, ax ; установка регистра ES на сегмент данных

xor ax, ax ; обнуление ax

mov dx, offset len ; ввод строки

call input ; вызов процедуры input - ввод строки

call do ; вызов процедуры do - основной алгоритм

call newline ; вызов процедуры newline - перевод каретки на новую строку

mov dx, offset string ; вывод полученной строки

call output ; вызов процедуры output

mov ah, 0 ; ожидание ввода символа

int 16h ; прерывание

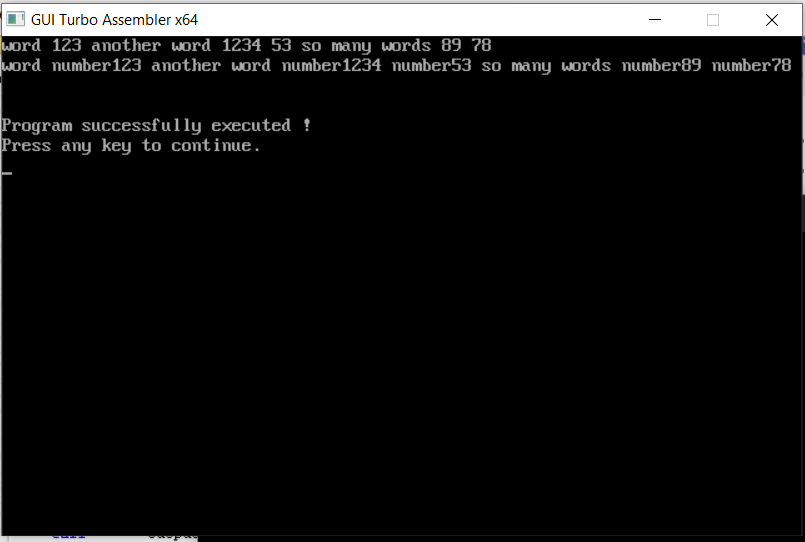
mov ah, 4Ch ; завершение программы

mov al, 0 ; номер функции DOS

int 21h ; обращение к функции DOS

start endp

end start



Программа ищет слова в строке, и если это число, сдвигает стоку из даной позиции на 6 байтов и вставляет туда «number» Основное действие программы:

mov dx, offset len ; ввод строки  
call input ; вызов процедуры input - ввод строки  
call do ; вызов процедуры do - основной алгоритм  
call newline ; вызов процедуры newline - перевод каретки на новую строку  
mov dx, offset string ; вывод полученной строки

Задача №8

LOCALS

.model small модель памяти, используемая для EXE

.stack 100h ; выделение 256 Б для стака

.data ; сегмент данных

kbdBuffer db 80, ?, 80 dup(0) ;буфер клавиатуры для ввода строки

CrLf db 0Dh, 0Ah, '$' ;символы перевода строки

Delimiter db ' ' ;разделитель слов в строке

String db ?, 80 dup(0) ;строка в формате длина, символы

WordForDelete db ?, 80 dup(0) ;строка в формате длина, символы

PromptStr db 'Enter string:', 0Dh, 0Ah, '$' ; перенос в начало следующей строки и строка "Enter string:" перед вводом строки для программы

PromptWord db 'Enter word:', 0Dh, 0Ah, '$' ; перенос в начало следующей строки и строка "Enter word:" перед вводом слова для программы

msgNewString db 'Result String: ', 0Dh, 0Ah, '$' ; перенос в начало следующей строки и строка "Result String: " перед выводом результата

.code ; сегмент кода

main proc ; процедура main

mov ax, @data ; помещает адрес сегмента data в регистр ax

mov ds, ax ; помещает адрес сегмента data из регистра ax в регистр ds

;ввод строки

mov ah, 09h ; помещает команду вывода строки на экран в регистр ah

lea dx, PromptStr ; помещает в dx строку PromptStr

int 21h ; прерывание команды ввода строки

lea dx, String ; помещает строку String в регистр dx

call GetStr ; вызов процедуры GetStr

;ввод слова для удаления из строки

mov ah, 09h ; помещает команду вывода строки на экран в регистр ah

lea dx, PromptWord ; помещает строку PromptWord в регистр dx

int 21h ; прерывание команды ввода слова

lea dx, WordForDelete ; помещает строку WordForDelete в регистр dx

call GetStr ; вызов процедуры GetStr

;цикл выделения слов и сравнения их с удаляемым словом

mov cx, 0 ;cx - длина строки

mov cl, String ; передаём регистру cl длину строки String

jcxz @@Break ;если строка пустая - завершить программу

mov dx, 0 ;dx - длина удаляемого слова

mov dl, WordForDelete ; передаём регистру dl длину строки WordForDelete

or dx, dx ; сравнивает

jz @@Break ; если равно - завершить программу

lea si, String+1 ;si - адрес первого символа строки

mov di, si ;di - адрес начала слова в строке

@@For: ; метка

lodsb ;чтение очередного символа в al, увеличение si на 1

cmp al, Delimiter ;очередной символ - разделитель слов

je @@NewWord ;да - перейти к строкам, выполняющим сравнение с образцом

loop @@For ; повторять до последнего слова

;здесь обработка последнего слова в строке

inc si ;для проверки последнего слова

@@NewWord: ; метка

pushf ; сохранить в стеке регистр FLAGS

cld ; очищение df

;сравнить длины слов

mov ax, si ; в ax помещаем si

sub ax, di ; вычитаем из ax адрес начала слова в строке

dec ax ; уменьшение значения, записанного в ax на 1

cmp ax, dx ; сравниваем ax и dx

jne @@Next ; если неравно переходим к @@Next

;при равенстве - сравнить слова

push si ; помещаем значение si в стек

push di ; помещаем значение di в стек

push cx ; помещаем значение cx в стек

push es ; помещаем значение es в стек

push ds ; помещаем значение ds в стек

pop es ; извлечение из стека значения es

mov cx, dx ; cx используется в repe

lea si, WordForDelete+1 ;si - адрес первого символа строки

repe cmpsb ; сравниваем значения на которые указывают di и si

pop es ; извлечение из стека значения es

pop cx ; извлечение из стека значения cx

pop di ; извлечение из стека значения di

pop si ; извлечение из стека значения si

jne @@Next ; если неравно переходим к @@Next

;при совпадении слов - удалить из строки слово

jcxz @@SkipCopy ; если cx = 0 то к метке скипкопи

push cx ; помещаем значение cx в стек

push si ; помещаем значение si в стек

push di ; помещаем значение di в стек

push es ; помещаем значение es в стек

push ds ; помещаем значение ds в стек

pop es ; извлечение из стека значения es

rep movsb ; ; пересылаем значения из si в di

pop es ; извлечение из стека значения es

pop di ; извлечение из стека значения di

pop si ; извлечение из стека значения si

pop cx ; извлечение из стека значения cx

;после переноса, просмотр продолжать с 1-го символа удалённого слова

mov si, di ; переместить в si значение di

dec [String] ;удаляется не только слово, но и разделитель

@@SkipCopy: ; метка

sub [String], dl ;коррекция длины строки

@@Next: ; метка

popf ; извлечение регистра флагов из стека

mov di, si ; переместить в si значение di

jcxz @@Break ; если cx = 0 то к метке @@Break

loop @@For ; повторение

@@Break: ; метка

;вывод результатов

mov ah, 09h ; помещает команду вывода строки на экран в регистр ah

lea dx, msgNewString ; помещает значение строки msgNewString в регистр dx

int 21h ; прерывание команды

call ShowString ; вызов процедуры ShowString

mov ah, 09h ; помещает команду вывода строки на экран в регистр ah

lea dx, CrLf ; помещает значение строки CrLf в регистр dx

int 21h ; прерывание команды

@@Exit: ; метка

mov ax, 4C00h ;завершение программы

int 21h ; прерывание команды

main endp ; конец процедуры

;Чтение строки с клавиатуры

;на входе:

; ds:dx - адрес строки

;на выходе:

; ds:dx - строка в Pascal формате (длина, символы)

GetStr proc ; процедура GetStr

push ax ; помещаем значение ax в стек

push bx ; помещаем значение bx в стек

push cx ; помещаем значение cx в стек

push dx ; помещаем значение dx в стек

push si ; помещаем значение si в стек

push di ; помещаем значение di в стек

pushf ; помещение регистра флагов в стек

push es ; помещаем значение es в стек

mov bx, dx ;сохранение адреса строки

;чтение в буфер из клавиатуры

mov ah, 0Ah ; Помещает команду ввода строки с клавиатуры в регистр ah

lea dx, kbdBuffer ; помещает значение строки kbdBuffer в регистр dx

int 21h ; прерывание команды

;перевод строки

mov ah, 09h ; Помещает команду вывода строки на экран в регистр ah

lea dx, CrLf ; помещает значение строки CrLf в регистр dx

int 21h ; прерывание команды

;копирование из буфера в переменную строки

mov cx, 0 ; cx = 0

mov cl, [kbdBuffer+1] ; в cl перемещается значение [kbdBuffer+1]

;jcxz @@SkipCopy

inc cx ; увеличивает значение в cx на 1

push ds ; помещаем значение ds в стек

pop es ; извлечение es из стека

lea si, kbdBuffer+1 ; помещает значение строки kbdBuffer в регистр si прибавив 1

mov di, bx ; перемещение в di значение bx

cld ; очищение df

rep movsb ; повтор копирования строк байтов

;добавление признака конца строки за последним символом

mov byte ptr [di], '$' ; в байтах записывается '$' в di

@@SkipCopy: ; метка

pop es ; извлечение из стека значения es

popf ; извлечение регистра флагов из стека

pop di ; извлечение из стека значения di

pop si ; извлечение из стека значения si

pop dx ; извлечение из стека значения dx

pop cx ; извлечение из стека значения cx

pop bx ; извлечение из стека значения bx

pop ax ; извлечение из стека значения ax

ret ; возврат из процедуры

GetStr endp ; конец кода процедуры GetStr

ShowString proc ; процедура ShowString

.386

pusha ; помещение регистров общего назначения в стек

mov cx, 0 ; cx = 0

mov cl, String ; помещает в cl длину String

lea si, String ; помещает значение строки String в регистр si

add si, cx ; прибавить к si значение cx

mov byte ptr [si+1],'$'

mov ah, 09h ; Помещает команду вывода строки на экран в регистр ah

lea dx, String+1 ; помещает значение строки String в регистр dx прибавив 1

int 21h ; прерывание команды

mov ah, 09h ; Помещает команду вывода строки на экран в регистр ah

lea dx, CrLf ; помещает значение строки CrLf в регистр dx

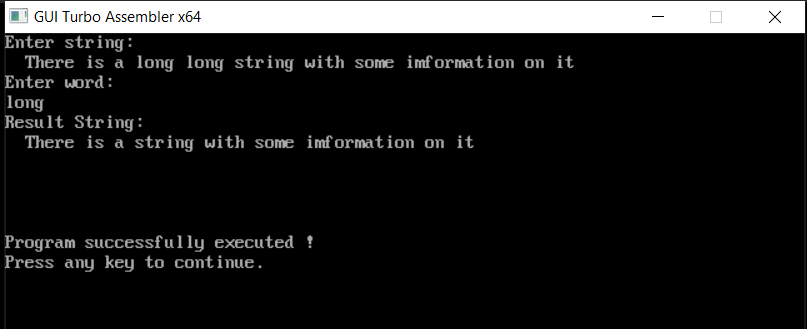
int 21h ; прерывание команды

popa ; извлечение регистров общего назначения из стека

ret ; возврат из процедуры

ShowString endp ; конец кода процедуры ShowString

end main ; конец программы main



Данная программа отвечает за удаление указанного пользователем слова из строки(удаляются все вхождения этог слова в строку), происходик перебор всех слов в цикле и сравнение каждого с удаляемым словом.

mov cx, 0 ;cx - длина строки

mov cl, String ; передаём регистру cl длину строки String

jcxz @@Break ;если строка пустая - завершить программу

mov dx, 0 ;dx - длина удаляемого слова

mov dl, WordForDelete ; передаём регистру dl длину строки WordForDelete

or dx, dx ; сравнивает

jz @@Break ; если равно - завершить программу

lea si, String+1 ;si - адрес первого символа строки

mov di, si ;di - адрес начала слова в строке

@@For: ; метка

lodsb ;чтение очередного символа в al, увеличение si на 1

cmp al, Delimiter ;очередной символ - разделитель слов

je @@NewWord ;да - перейти к строкам, выполняющим сравнение с образцом

loop @@For ; повторять до последнего слова

;здесь обработка последнего слова в строке

inc si ;для проверки последнего слова

@@NewWord: ; метка

pushf ; сохранить в стеке регистр FLAGS

cld ; очищение df

;сравнить длины слов

mov ax, si ; в ax помещаем si

sub ax, di ; вычитаем из ax адрес начала слова в строке

dec ax ; уменьшение значения, записанного в ax на 1

cmp ax, dx ; сравниваем ax и dx

jne @@Next ; если неравно переходим к @@Next

;при равенстве - сравнить слова

push si ; помещаем значение si в стек

push di ; помещаем значение di в стек

push cx ; помещаем значение cx в стек

push es ; помещаем значение es в стек

push ds ; помещаем значение ds в стек

pop es ; извлечение из стека значения es

mov cx, dx ; cx используется в repe

lea si, WordForDelete+1 ;si - адрес первого символа строки

repe cmpsb ; сравниваем значения на которые указывают di и si

pop es ; извлечение из стека значения es

pop cx ; извлечение из стека значения cx

pop di ; извлечение из стека значения di

pop si ; извлечение из стека значения si

jne @@Next ; если неравно переходим к @@Next

;при совпадении слов - удалить из строки слово

jcxz @@SkipCopy ; если cx = 0 то к метке скипкопи

push cx ; помещаем значение cx в стек

push si ; помещаем значение si в стек

push di ; помещаем значение di в стек

push es ; помещаем значение es в стек

push ds ; помещаем значение ds в стек

pop es ; извлечение из стека значения es

rep movsb ; ; пересылаем значения из si в di

pop es ; извлечение из стека значения es

pop di ; извлечение из стека значения di

pop si ; извлечение из стека значения si

pop cx ; извлечение из стека значения cx

;после переноса, просмотр продолжать с 1-го символа удалённого слова

mov si, di ; переместить в si значение di

dec [String] ;удаляется не только слово, но и разделитель

@@SkipCopy: ; метка

sub [String], dl ;коррекция длины строки

@@Next: ; метка

popf ; извлечение регистра флагов из стека

mov di, si ; переместить в si значение di

jcxz @@Break ; если cx = 0 то к метке @@Break

loop @@For ; повторение

@@Break: ; метка

Задача №9

.model small ; модель сегментации, используемая для компляции в .exe

.stack 100h ; выделение памяти для программы (256 байт)

.data ; сегмент данных

i dw 0h ; переменная для хранения слов

String db ': $' ; исходная строка, выводимая пользователю перед вводом

Stg db 100h dup(0h) ; массив для строки

.code ; сегмент кода

Start: ; метка Start

Sort proc ; процедура Sort

mov ax, @data ; помещение в регистр DS сегмента данных, но так как мы не можем напрямую обратится к регистру DS, то мы делаем это через регистр AX

mov ds, ax ; помещение в регистр DS сегмента данных

mov ah, 00h ; очистка консоли

mov al, 2h ; функция установки позиции курсора

int 10h ; прерывание DOS

mov ah, 09h ; функция вывода на экран

Lea dx, String ; вывод строки String на экран

int 21h ; прерывание DOS

mov ah, 1h ; функция ввода символа

mov si, 0h ; очистка экрана

mov bx, 0h ; установка полей BIOS

Input: ; метка Input. Ввод массива

int 21h ; прерывание DOS

mov cx, si ; запись значения si в cx

mov Stg[bx], cl ; формирование массива Stg длинной равной длине слова (cl)

cmp al, 32 ; проверка на пробел

jne Skip1 ; условный переход к метке Skip1, если пойманный символ - пробел

mov si, 0h ; установка полей BIOS

add bx, 10h ; начало следующего слова

jmp Input ; переход к метке Input

Skip1: ; метка Skip1

inc si ; приавляем 1 к значению si

mov Stg[bx+si], al ; помещение символа в массив

cmp al, 13 ; проверка на перенос строки

jne Input ; условный переход к метке Input, если был пойман перевод строки

mov Stg[bx+si], 0h; Удаление Enter'а

mov i, bx ; количество слов

mov bx, 0h ; установка полей BIOS

Sort1: ; метка Sort1. выборочная сортировка

mov di, bx ; заносим индекс минимальной длины в di

mov ax, bx ; заносим индекс минимальной длины в ax

add ax, 10h ; прибавляем к регистру ax число 10

Sort2: ; метка Sort2

mov si, ax ; заносим в si значение ax

mov cl, Stg[di] ; смещение в массиве по длине слова

cmp cl, Stg[si] ; сравнение внутри массива по длине слов

jae Skip2 ; условный переход, если слово больше или равно второму

mov di, si ; если меньше, то помещаем значение si в di

Skip2: ; метка Skip2

add ax, 10h ; прибавляем к регистру ax число 10

cmp ax, i ; Сравниваем значение регистра ax со значением данных i

jbe Sort2 ; сли значение регистра ax меньше или равно переходим к команде Sort2.

mov si, 0h ; установка полей BIOS

Sort3: ; метка Skip3

mov cl, Stg[bx+si] ; смена слов. Выделение дополнительной памяти под длину большего слова

mov al, Stg[di] ; установление меньшей памяти для меньшего слова

mov Stg[bx+si], al ; запись длины большего слова в al

mov Stg[di], cl ; запись длины меньшего слова в cl

inc si ; увеличение si на 1

inc di ; увеличение di на 1

cmp si, 10h ; если значение si меньше 10,

jb Sort3 ; то выполняем условный переход к метке Sort3

add bx, 10h ; прибавляем к регистру bx число 10

cmp bx, i ; если значение bx меньше i,

jb Sort1 ; то выполняем условный переход к метке Sort1

mov ah, 02h; функция установки позиции курсора:

mov bh, 0h; № Страницы

mov dh, 2h; № строки

mov dl, 0h; № столбца

int 10h ; прерывание DOS

mov bx, 0h ; установка на начало массива

mov si, 0h ; установка на начало первого элемента

mov ah, 2h ; функция вывода символа

Output: ; метка Output. Вывод массива

inc si ; прибавляем к si единицу

mov dx, word ptr Stg[bx+si] ; получение слова для вывода по указанному адресу

cmp dx, 0h ; если значение регистра dx не равно 0,

jne Skip3 ; то переходим к команде Skip3.

cmp bx, i ; если значение регистра bx равно i,

je Exit ; то переходим к команде Exit

mov si, 0h ; установка полей BIOS

add bx, 10h ; прибавляем 10 к bx

mov dx, ' ' ; записываем разделитель слов (пробел) в dx (для отделения слов друг от друга)

Skip3: ; метка Skip3

int 21h ; прерывание DOS

cmp bx, i ; если bx меньше или равно i,

jbe Output ; то выполняем условный переход к метке Output

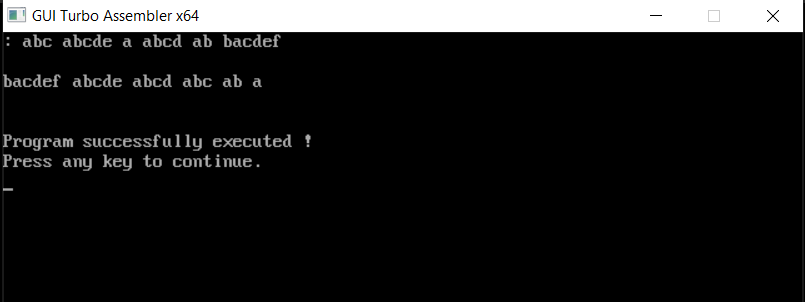
Exit: ; метка Exit

mov ah, 4ch ; вызов функции завершения программы

int 21h ; прерывание DOS

Sort endp ; завершение процедуры Sort

End Start ; завершение функции Start



Данная программа возвращает строку, слова которой упорядочены по убыванию длины слов, происходит перебор всех слов, измерение их длины и сравнение, слова переставляются пока не будет достигнут нужный результат.

Основной блок кода:

mov Stg[bx+si], 0h; Удаление Enter'а

mov i, bx ; количество слов

mov bx, 0h ; установка полей BIOS

Sort1: ; метка Sort1. выборочная сортировка

mov di, bx ; заносим индекс минимальной длины в di

mov ax, bx ; заносим индекс минимальной длины в ax

add ax, 10h ; прибавляем к регистру ax число 10

Sort2: ; метка Sort2

mov si, ax ; заносим в si значение ax

mov cl, Stg[di] ; смещение в массиве по длине слова

cmp cl, Stg[si] ; сравнение внутри массива по длине слов

jae Skip2 ; условный переход, если слово больше или равно второму

mov di, si ; если меньше, то помещаем значение si в di

Skip2: ; метка Skip2

add ax, 10h ; прибавляем к регистру ax число 10

cmp ax, i ; Сравниваем значение регистра ax со значением данных i

jbe Sort2 ; сли значение регистра ax меньше или равно переходим к команде Sort2.

mov si, 0h ; установка полей BIOS

Sort3: ; метка Skip3

mov cl, Stg[bx+si] ; смена слов. Выделение дополнительной памяти под длину большего слова

mov al, Stg[di] ; установление меньшей памяти для меньшего слова

mov Stg[bx+si], al ; запись длины большего слова в al

mov Stg[di], cl ; запись длины меньшего слова в cl

inc si ; увеличение si на 1

inc di ; увеличение di на 1

cmp si, 10h ; если значение si меньше 10,

jb Sort3 ; то выполняем условный переход к метке Sort3

add bx, 10h ; прибавляем к регистру bx число 10

cmp bx, i ; если значение bx меньше i,

jb Sort1 ; то выполняем условный переход к метке Sort1

**Вывод:** ознакомились с технологией написания и отладки программ на языке ассемблера. Ознакомились со структурой программы и директивами языка. Ознакомилась с операциями пересылки данных, передачи управления и прерываниями ввода-вывода.