

МИНОБРНАУКИ РОССИИ САНКТ-
ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МОЭВМ

ОТЧЕТ

по лабораторной работе №7

по дисциплине «Организация ЭВМ и систем»

ТЕМА: ПРЕОБРАЗОВАНИЕ ЦЕЛЫХ ЧИСЕЛ. ИСПОЛЬЗОВАНИЕ ПРОЦЕДУР
В АССЕМБЛЕРЕ.

Студент гр. 1303

Насонов Я.К.

Преподаватель

Ефремов М.А.

Санкт-Петербург

2022

Цель работы.

Получить навыки программирования на языке Ассемблера. Изучить работу с целыми числами с использованием процедур на языке Ассемблера.

Задание.

Разработать на языке Ассемблер IntelX86 две процедуры: одна - прямого и другая - обратного преобразования целого числа, заданного в регистре AX или в паре регистров DX:AX, в строку, представляющую его символьное изображение в заданной системе счисления (с учетом или без учета знака в зависимости от варианта задания).

Строка должна храниться в памяти, а также выводиться на экран для индикации.

Отрицательные числа при представлении с учетом знака должны в памяти храниться в дополнительном коде, а на экране изображаться в прямом коде с явным указанием знака или в символьном виде со знаком.

Вариант 13:

16-битное число, с учетом знака, десятичная система счисления, способ вызова процедур – far, связь по данным между основной программой и подпрограммами через РОН.

Выполнение работы.

В главной процедуре MAIN происходит запись в регистрах исходного числа (number в сегменте данных). Далее проверяется знак числа, если число положительное, то в sign кладется знак '+', если отрицательное, то '-'.

Это необходимо для вывода символьного представления числа. Затем с помощью процедуры Number_to_String преобразовываем исходное число в строку (в десятичной системе счисления) и записываем это значение в number_string. С помощью процедуры WriteMsg выводим. После того, как получено строковое представление числа, обнулим регистр AX. В него запишем число после обратного преобразования. Затем с помощью процедуры

String_to_Number получаем из строки число. Тот результат, который мы записали в регистр AX должен совпадать со значением, находящемся в number.

Процедура Number_to_String: число делится на 10, до тех пор, пока будет не 0, а к остатку от деления добавляется код символа '0', полученное значение кладется на стек, затем элементы из стека записываются в строку (таким образом мы получим нужный порядок цифр без дополнительных обработок).

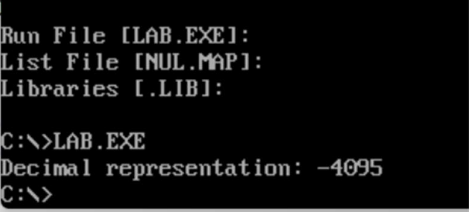
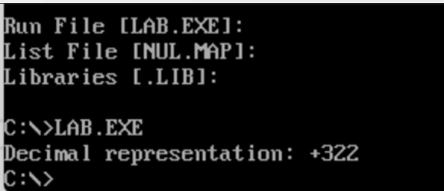
Процедура String_to_Number: циклом подсчитываем длину числа. Затем значение, лежащее в регистре AX (в начале 0, т.к. мы обнулили его в MAIN), умножается на 10, и к нему добавляется разность кодов символов из записи числа и '0'. Данные действия осуществляются в цикле, пока не пройдем всю строку. В конце мы проверяем, было ли исходное число отрицательным, в случае, если да, применяем команду neg для регистра AX.

Исходный код программы см. в приложении А.

Тестирование.

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии
1	-offf	<pre> number_output DB '0' N DW 0 number_string DB ' ' sign DB ' ', '\$' number DW -0ffffh DATA ENDS </pre> 	
2	142	<pre> 6 number_output DB '0' 7 N DW 0 8 number_string DB ' ' 9 sign DB ' ', '\$' 10 number DW 142h 11 DATA ENDS 12 13 </pre> 	

Выводы.

В ходе выполнения лабораторной работы были получены навыки программирования на языке Ассемблера. Была разработана программа переводящая число в 10-ричную систему в строковом виде и обратно.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: lab7.asm

```
AStack SEGMENT STACK
    DB 1024 DUP(?)
AStack ENDS
```

```
DATA SEGMENT
    number_output DB 'Decimal representation: ', '$'
    N DW 0
    number_string DB ' ', '$'
    sign DB ' ', '$'
    number DW 142h
DATA ENDS
```

```
CODE SEGMENT
    ASSUME CS:CODE, DS:DATA, SS:AStack
```

```
WriteMsg PROC NEAR
    mov ah, 9
    int 21h
    ret
WriteMsg ENDP
```

```
Number_to_String proc FAR
    push ax
    push bx
    push cx
    push dx

    xor cx, cx
    mov bx, 10 ; делитель 10 для десятичной с.с.
    mov di, offset number_string
```

```
division_mod:
    xor dx, dx
    div bx ; ax = (dx:ax) / bx, остаток в dx
    add dl, '0'
    push dx
    inc cx
    test ax, ax
    jnz division_mod
```

```
symbols_from_stack:
    pop dx
    mov [di], dl
    inc di
    loop symbols_from_stack
```

```
    mov bx, '$'
    mov [di], bx
```

```
    pop dx
    pop cx
    pop bx
    pop ax
```

```
    ret
Number_to_String ENDP
```

```
String_to_Number proc FAR
    push di
    push cx
    push bx
    push dx

    mov di, offset number_string
    mov dx, '$'

    xor bx, bx

find_len:
    cmp [di+bx], dx
    je len_found
    inc bx
    jmp find_len

len_found:
    mov cx, bx

    mov bx, 10
    mov dx, 0

mul_numbers:
    mul bx
    mov dl, [di]
    sub dl, '0'
    add al, dl
    inc di
    loop mul_numbers

    mov di, offset N
    mov dx, [di]
    cmp dx, 0
    je positive_num

    neg ax

positive_num:
    pop dx
    pop bx
    pop cx
    pop di
    ret
String_to_Number endp
```

```
MAIN PROC FAR
    push DS
    xor ax, ax
    push ax
    mov ax, DATA
    mov ds, ax

    mov dx, offset number_output
    call WriteMsg
```

```

    mov ax, number
    mov di, offset sign
    mov bx, "+"
    cmp ax, 0
    jge set_sign
    mov bx, "-"
    neg ax
    push bx
    mov bx, 1
    mov N, bx
    pop bx

set_sign:
    mov [di], bx
    inc di
    mov bx, '$'
    mov [di], bx

    push ax
    mov dx, offset sign
    call WriteMsg
    pop ax

    call Number_to_String
    push ax
    mov dx, offset number_string
    call WriteMsg
    pop ax

    xor ax, ax
    call String_to_Number
    ret
MAIN ENDP
CODE ENDS
    END MAIN

```