

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

ОТЧЕТ
по лабораторной работе №7
по дисциплине «Организация ЭВМ и систем»
Тема: Преобразование целых чисел. Использование процедур в
Ассемблере.

Студент гр. 1381

Мамин Р.А.

Преподаватель

Ефремов М.А.

Санкт-Петербург

2022

Цель работы.

Научится создавать процедуры на языке Ассемблер и использовать их для решения различных задач. Научится преобразовывать целые числа из одних форматов хранения данных в другие, используя механику процедур в языке Ассемблер.

Задание.

Вариант 1

Шифр задания: 1.1.2.1.В

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.

Пример для однобайтовых чисел:

Десятичное число в символьном виде. Двоично-десят. упаков.число

	в ДК	в ПК
+ 35	00110101	00110101
- 35	11001011	10110101

Вариант выполнения преобразования определяется шифром, состоящим из 4-х цифр:

- 1-я цифра задает длину целого числа:
1- 16 бит, 2- 32 бита;
- 2-я цифра задает вид представления числа:
1- с учетом знака, 2- без учета знака;
- 3-я цифра задает систему счисления для символьного изображения числа:
1- двоичная, 2- восьмеричная, 3- десятичная, 4- шестнадцатеричная.
- 4-я цифра задает способ вызова процедур:
1- near (ближнего вызова), 2 - far (дальнего вызова);

Написать простейшую головную программу для иллюстрации корректности выполнения заданных преобразований.

Связь по данным между основной программой и подпрограммами может осуществляться следующими способами:

А - через РОНы; В - через кадр стека.

Выполнение работы.

В регистр AX записывается число, которое необходимо перевести в строку. В программе реализовано две процедуры AX_TO_STR и STR_TO_AX.

Первая процедура AX_TO_STR начинает с проверки знака числа. Если число отрицательное, то оно инвертируется и инкрементируется для того, чтобы корректно был произведен перевод в строку. В начале строки записывается знак. Если число равно нулю, то сразу записывается нуль, как ответ. Ранее была объявлена переменная, нужная для того, чтобы проследить, нужно ли в строку записывать спереди идущие нули. Программа из числа берет цифру и записывает в символьном виде в строку, проверяя переменную, кото-

рая была упомянута в прошлом предложении. В конец строки добавляется символ конца строки, и строка выводится.

Вторая процедура STR_TO_AX начинает со знака. Если это минус, то в конце нужно будет проинвертировать число и инкрементировать. Для проверки этого условия была объявлена переменная is_neg. Далее происходит считывание количества цифр и проход по строке. Расстояние в таблице ASCII между цифрами и буквами, используемыми в 16-ичной СС равно 7, поэтому случаи с буквами нужно рассматривать немного по-другому. Результат записывается в другой регистр для удобной работы, затем возвращается в AX.

В основной процедуре сначала вызывается AX_TO_STR и выводится строка, которая является числом в 16-ичной СС. Затем вызывается STR_TO_AX, для проверки корректности вызывается заново AX_TO_STR, если строки совпадают, то их корректность очевидна.

Тестирование.

№ п/п	Исходные данные	Выходные данные	Комментарии
1.	AX = 0h	Перевод из регистра AX в строку: +0 Перевод из строки в регистр AX и обратно: +0	Программа работает корректно
2.	AX = FFFFh	Перевод из регистра AX в строку: -1 Перевод из строки в регистр AX и обратно: -1	Программа работает корректно
3.	AX = 8000h	Перевод из регистра AX в строку: -8000 Перевод из строки в регистр AX и обратно: -8000	Программа работает корректно

Выводы.

Было освоено умение создавать процедуры на языке Ассемблер и использовать их для решения различных задач.

В ходе выполнения работы была разработана программа выполняющая поставленное задание, а именно, преобразовывать целые числа записанные в регистре в строку и обратно . Для решения этой задачи были применены полученные навыки использования процедур в языке Ассемблер.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: **main.asm**

```
STACKSG SEGMENT PARA STACK 'Stack'
        DW      512 DUP(?)
STACKSG      ENDS

DATASG SEGMENT PARA 'Data'; SEG DATA
        KEEP_CS DW 0 ;
        MESSAGE1 DB 'Перевод из регистра AX в строку: $'
        MESSAGE2 DB 'Перевод из строки в регистр AX и обратно: $'
        STRING DB 35 DUP('#')
DATASG      ENDS; ENDS DATA

CODE SEGMENT; SEG CODE
ASSUME DS:DataSG, CS:Code, SS:STACKSG
;-32 768:+32 767

AX_TO_STR PROC NEAR
        jmp start_1
        delete_nul DW 0
start_1:
        mov delete_nul, 0
        mov DI, 0h; DI - индекс текущего символа строки
        cmp AX, 0
        jge positive

negative:
        mov STRING[DI], '-'
        add DI, 1 ;инвертируем число и прибавляем единицу
        not AX
        add AX, 1
        jmp scan_ax

check_nul:
        cmp delete_nul, 0
        je skip_char
        jmp no_skip_char

positive:
        mov STRING[DI], '+'
        add DI, 1
        cmp AX, 0
        je case_nul

scan_ax:
        mov SI, AX ; записываем в si, ax
```

```

        mov     cx, 4          ; в слове 4 ниббла (полубайта)

next_char:
        rol     ax, 1          ; выдвигаем младшие 4 бита
        rol     ax, 1
        rol     ax, 1
        rol     ax, 1
        push    ax             ; сохраним AX
        and     al, 0Fh        ; оставляем 4 младших бита AL
        cmp     al, 0Ah        ; сравниваем AL со значением 10
        sbb     al, 69h        ; целочисленное вычитание с заёмом
        das                     ; BCD-коррекция после вычитания
        cmp     al, '0'
        je      check_nul
        mov     delete_nul, 1

no_skip_char:
        mov     STRING[DI], al
        add     DI, 1

skip_char:
        pop     ax             ; восстановим AX
        loop    next_char
        jmp     end_1

case_nul:
        mov     STRING[DI], '0'
        add     DI, 1

end_1: ; когда прошли все регистры
        mov     STRING[DI], '$' ; добавляем в конец строки символ конца
строки
        mov     DX, offset STRING ; записываем в dx сдвиг строки
        ret
AX_TO_STR ENDP

```

```

STR_TO_AX PROC FAR
        jmp     start_2
        IS_NEG DB 0; отвечает за знак числа

start_2:
        mov     AX, 0; обнуляем ax
        mov     CX, 0
        mov     SI, 0; за индекс строки будет отвечать si
        cmp     STRING[SI], '-' ; сравниваем первый элемент строки с
минусом
        jne     positive_parse; если не равен минусу, то число
положительное
        ;если равен то отрицательное
        mov     IS_NEG, 1; в is_neg записываем 1

```

```

positive_parse: ; если число положительно
                mov SI,0 ; кладем в SI 0

len_loop: ; считаем длину строки
            add SI,1
            cmp STRING[SI], '$' ; сравниваем элемент строки с $
            jne len_loop ; если не равен $ то возвращаемся в цикл

            mov DI, SI
            lea SI, STRING
            inc SI
            xor cx, cx
            cld

number_construct:
            xor AX, AX
            dec DI ; декрементам DI
            cmp DI,0 ; сравниваем DI с 0
            jle done ; DI <= 0
            lodsb
            cmp al, 'A'
            jge буква

continue:
            sub al, '0'
            xchg ax, cx
            mov dx, 10h
            mul dx
            add cx, ax
            jmp number_construct
done:
            mov ax, cx
            cmp IS_NEG, 1
            je check_negative
            jmp end_2

буква:
            sub al, 7
            jmp continue

check_negative:
            not ax
            add ax, 1

end_2:
            ret
STR_TO_AX ENDP

```

```

Main PROC FAR
        mov ax, DATASG

```



```

mov ds, ax

    mov DX, offset MESSAGE1
    mov ah, 09h;
int 21h;
mov AX, 0h ; задаем регистр AX
pushf
call AX_TO_STR
mov ah, 09h;
int 21h;

mov dl, 10
mov ah, 02h
int 21h
mov dl, 13
mov ah, 02h
int 21h

    mov DX, offset MESSAGE2
    mov ah, 09h;
int 21h;
mov ax, 0
call STR_TO_AX
popf
call AX_TO_STR

mov ah, 09h
int 21h

mov ah, 4Ch;
int 21h;

Main      ENDP
CODE      ENDS
END Main

```