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: Использование арифметических операций над целыми числами и процедур в Ассемблере.

Студентка гр. 9382 _____ Пя С.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург

2020

Цель работы.

Изучить арифметические операции над целыми числами и процедуры в Ассемблере.

Теоретические сведения:

Разработать на языке Ассемблер процессора IntelX86 две процедуры:

• одна – выполняет прямое преобразование 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);

• другая - обратное преобразование строки, представляющей символьное изображение числа в заданной системе счисления в целое число, помещаемое в регистр AX (или в пару регистров DX:AX)

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.

Задание:

Вариант 1.1.4 – 16 бит, с учетом знака, 16-ичная СС

Ход работы:

В регистр записывается число, которое необходимо перевести в строку. В программе реализовано две процедуры DEC_TO_HEX и HEX_TO_DEC.

Первая процедура DEC_TO_HEX 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, так как в последующем коде такой случай был бы исключительным. 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, которая была упомянута в прошлом предложении. В конец строки добавляется символ конца строки, и строка выводится.

Вторая процедура HEX_TO_DEC 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, использующимися в 16-ирчной СС равно 7, поэтому случаи с буквами нужно рассматривать немного по-другому. Результат записывается в другой регист для удобной работы, затем возвращается в AX.

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

Тестирование.

№	Входные данные	Результат
1.	AX = 0h	Transformation to string: +0 Transformation from string to digit and back: +0
2.	AX = 8000h	Transformation to string: -8000 Transformation from string to digit and back: -8000
3.	AX = FFFFh	Transformation to string: -1 Transformation from string to digit and back: -1
4.	AX = 8001h	Transformation to string: -7FFF Transformation from string to digit and back: -7FFF

Выводы.

В результате выполнения лабораторной работы был разработан код, использующий арифметические операции над целыми числами и процедуры в ассемблере, написаны две процедуры по преобразованию числа в строку и обратно.

Приложение.

Текст файла 7lab.asm

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

DATASG  SEGMENT  PARA 'Data'; SEG DATA
       KEEP_CS DW 0 ;
        MESSAGE1 DB 'Transformation to string: $'
        MESSAGE2 DB 'Transformation from string to digit and back: $'
       STRING DB 35 DUP('0')
DATASG ENDS; ENDS DATA


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

DEC_TO_HEX PROC NEAR
       jmp start_1
       delete_nul DW 0
start_1:
       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
       jne 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
DEC_TO_HEX ENDP

HEX_TO_DEC 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 ; декреминтим DI
       cmp DI,0 ; сравниваем DI с 0
       jle done ; DI <= 0
       lodsb
       cmp al, 'A'
       jge bukva

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

bukva:
        sub al, 7
        jmp continue

check_negative:
        not ax
        add ax, 1

end_2:
        ret
HEX_TO_DEC ENDP


Main PROC FAR
        mov  ax, DATASG
        mov  ds, ax

         mov DX, offset MESSAGE1
         mov ah,09h;
        int 21h;
        mov AX, 08001h
        call DEC_TO_HEX
        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 HEX_TO_DEC
        call DEC_TO_HEX

        mov ah,09h
        int 21h

        mov ah,4Ch;
        int 21h;

Main      ENDP
CODE      ENDS
        END Main
```