**Міністерство освіти і науки України Національний технічний університет України «Київський Політехнічний Інститут імені Ігоря Сікорського» Кафедра конструювання електронно-обчислювальної апаратури**

Звіт з виконання лабораторної роботи №5

з дисципліни “Основи мікропроцесорної техніки”

Виконав: студент групи ДК-82

Рудюк Б. Б.

Перевірив: доц.

Корнєв В. П.

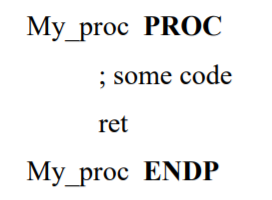
**Київ – 2020**

**Тема.** Ознайомлення з командами які дозволяють проводити операції над стеком. Ознайомлення з процедурами у мові асемблеру та способами передачі даних у них.

**Мета:** Вивчення команд операцій над стеком та створення власних процедур.

**Теоретична частина**

**Створення та використання процедур у мові асемблеру**

Для ініціалізації процедури використовують відповідно директиви PROC та ENDP, приклад синтаксу:

Команда ret потрібна для того, щоб по завершенню виконання процедури виконання програми відбувалося далі з того місця, на якому була викликана процедура, тобто повертається адреса на ділянку коду в якій потрібно продовжити виконання програми.

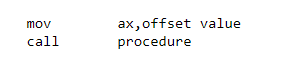
Для виклику процедури необхідно використати команду Call після чого вказати ім’я процедури. Проводячи аналогії з мовами програмування, наприклад С, С++, можна вважати, що процедури є своєрідним аналогом функцій. Як і в функціях, до процедур необхідно надсилати аргументи, або дані, з якими потім в середині процедури будуть проводитися операції.

**Способи передачі параметрів процедурам**

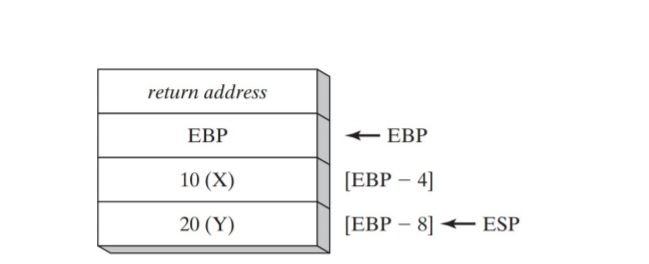
* Передача параметрів за значеннями. Процедурі передається власне значення параметра. При цьому фактично значення параметра копіюється, і процедура використовує його копію, тому модифікація вихідного параметра виявляється неможливою. Цей механізм застосовується для передачі невеликих параметрів, таких як байти або слова.

Приклад даної передачі:

* Через регістри. Перед викликом процедури необхідні параметри зберігаються в регістрах. Такий метод є найшвидшим, але є істотний недолік: кількість регістрів обмежена.
* Через стек. Використовуючи стек до процедур можна передавати як значення даних, так і адреси на пам’ять, в якій зберігаються дані. Доступ до даних у стеку відбувається за допомогою регістрів покажчику стеку EBP, ESP. Регістр вказівник на стек вказує на адресу останнього доданого елементу до стеку. Перед закінченням процедури, ESP вказує на сегмент стеку де зберігається адреса повернення з процедури, тобто адреса комірки у сегменті коду, до якої буде передано управління по завершенню процедури. Саме через цю особливість, використання регістру вказівника на стек для доступу до даних, що знаходяться в стеку, не є правильним, оскільки, в такому разі може змінитися вміст регістру ESP, що в свою чергу може призвести до помилок у роботі програми в майбутньому. Для доступу до елементів стеку використовують регістр вказівник на базу — EBP, в який записується вміст регістру ESP.
* Передача параметрів за посиланням. Процедурі передається не значення змінної, а її адресу, за якою процедура повинна сама прочитати значення параметра. Цей механізм зручний для передачі великих масивів даних і для тих випадків, коли процедура повинна модифікувати параметри, хоча він і повільніше через те, що процедура буде виконувати додаткові дії для отримання значень параметрів.

Приклад даної передачі:

**Використання локальних змінних у процедурах**

Іноді в процесі виконання тієї чи іншої операції в процедурах, необхідно зберігати проміжкові дані в змінних, якщо всі регістри вже застосовані. Щоб зайвий раз не створювати глобальну змінну в сегменті даних і тим самим не займати зайвої пам’яті, можна використовувати локальні змінні в процедурах. Локальні змінні створюються в стеку шляхом звернення до комірок стеку адреса яких менша за адресу на яку вказує EBP. Приклад створення двох локальних змінних:

При створенні локальних змінних слід пам’ятати, що після їх створення вказівник ESP вже вказуватиме на адрес комірки локальної змінної. Саме тому перед тим як видаляти значення EBP зі стеку, потрібно завантажити в регістр ESP адресу з регістру EBP, таким чином ESP вказуватиме на комірку з EBP і локальні змінні видаляться. Якщо ж не провести попередньо видалення локальних змінних то регістр ESP вказуватиме на невірний адрес і в результаті не відбудеться повернення з процедури.

**Спосіб виклику процедур (INVOKE)**

За допомогою директиви INVOKE можна легко передавати параметри у процедуру. Синтаксис директиви наступний: INVOKE procedureName [, argumentList] Тобто через кому можна задавати необхідну кількість параметрів. Ця директива робить push argument. Для того, щоб викликати процедуру за допомогою INVOKE, нам потрібно перед цим створити прототип за допомогою директиви PROTO.

Приклад виклику функції Dump Array:

INVOKE DumpArray, OFFSET array, LENGTHOF array, TYPE array

Інший варіант даного виклику:

push TYPE array

push LENGTHOF array

push OFFSET array

call DumpArray

**Xiд роботи**

**Завдання**

Зробити виконання 4-ї лабараторної роботи у процедурі.

**Програмний код**

INCLUDE Irvine32.inc

ExitProcess proto,dwExitCode:dword

addnumb PROTO, x1:DWORD, x2:DWORD ;протип для функції з параметрами

.data

FirstN byte "First number: " , 0 ;рядок з текстом

SecondN byte "Second number: ", 0 ;рядок з текстом

numb1 DWORD ? ;масив для запису першого числа

numb2 DWORD ? ;масив для запису другого числа

output DWORD '0' ;масив результату

.code

main proc

xor edx,edx ;очищення регістрів

xor eax, eax

xor ebx,ebx

lea edx, FirstN ;записуємо OFFSET FirstN в edx

call WriteString ;вивід рядка FirstN в консоль (First number:)

lea edx, numb1 ;записуємо OFFSET numb1 в edx

mov ecx, SIZEOF numb1

call ReadString ;виконання запису першого числа

call Crlf ;перенос строки

lea edx, SecondN ;записуємо OFFSET SecondN в edx

call WriteString ;вивід рядка SecondN в консоль (Second number:)

lea edx, numb2 ;записуємо OFFSET numb2 в edx

mov ecx, SIZEOF numb2

call ReadString ;виконання запису другого числа

call Crlf ;перенос строки

INVOKE addnumb, numb1, numb2 ;виклик створеної функції, яка буде виконувати додавання (передаємо параметри)

mov output, eax ;записуємо відповідь в масив output

lea edx, output ;записуємо OFFSET output в edx

call WriteString ;виводимо строку output з результатом

invoke ExitProcess,0

main endp

addnumb proc, x1: dword, ;описуємо ім'я процедури і список передаваємих їй параметрів

x2: dword

lea esi, x1 ;записуємо OFFSET numb1 в esi

mov al, byte ptr [esi] ;переводимо значення регістру esi з DWORD в BYTE (byte ptr) і записуємо в al

and al, 0fh ;перевод в неупаков ВСD формат (з ASCII)

lea esi, x2 ;записуємо OFFSET numb2 в esi

mov bl, byte ptr [esi] ;переводимо значення регістру esi з DWORD в BYTE (byte ptr) і записуємо в al

and bl, 0fh ;перевод в неупаков ВСD формат (з ASCII)

add al, bl ;виконання додавання чисел

aaa ;корегування отриманого результату в неупакований BCD формат

or ax, 3030h ;переводимо в ASCII формат

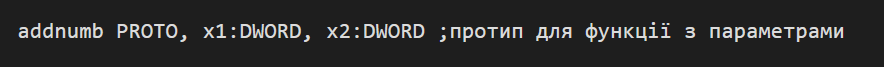
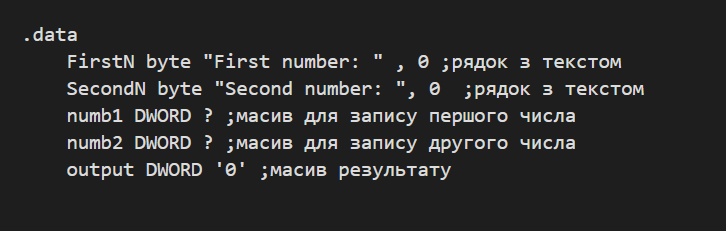
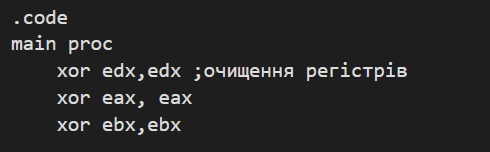
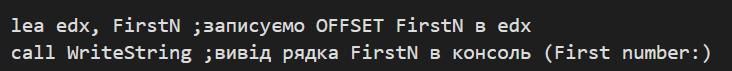
xchg ah,al ;обмінюємо значення регістрів al і ah

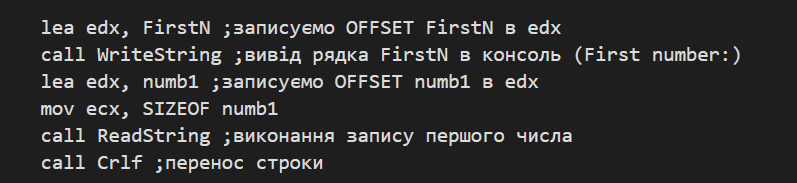
ret ;функція, яка поверне управління команді, наступній за тією, яка викликала цю процедуру

addnumb endp

end main

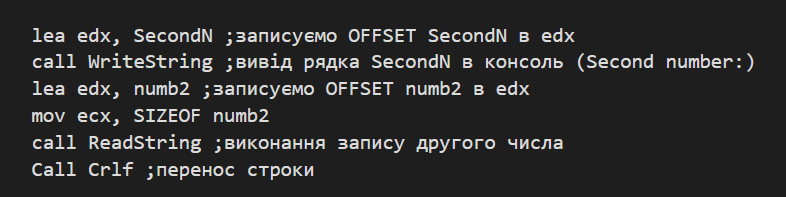
**Опис програми**

1. На початку програми створюємо прототип функції.
2. У даному сегменті коду, зберігаються дані в пам’яті.
3. Перед початком вводу даних, відбувається очищення регістрів.
4. Починається сегмент коду, в якому вводяться дані. Для створення певного інтерфейсу для користувача, спочатку виводиться строка, яка інформує які дані потрібно ввести користувачу. Виконується дана операція за допомогою функції call WriteString, але потрібно перед викликом цієї функції записати в edx, адресу строки, яку хочемо виводити. Виконуємо цю умову за допомогою команди lea edx, FirstN, ця команда еквівалентна mov edx, OFFSET FirstN.

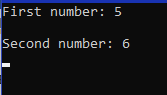
Далі відбувається підготовка регістрів для виклику функції call ReadString, щоб користувач міг записати дані використовуючи клавіатуру. У регістр edx записуємо OFFSET масива numb1 куди будуть записуватися дані, а в регістр ecx передаємо SIZEOF (розмір) даного масива. Викликаємо функцію call ReadString.

Після того, як користувач введе дані, використовуючи функцію call Crlf, перемістимо курсор на екрані монітора в першу позицію наступної строки.

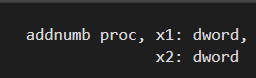
Інформація, яку буде бачити користувач на екрані.

Так, як для додавання нам потрібно два числа, ще раз виконуємо описанні вище дії, але записуємо дані в вже в інший масив (numb2).

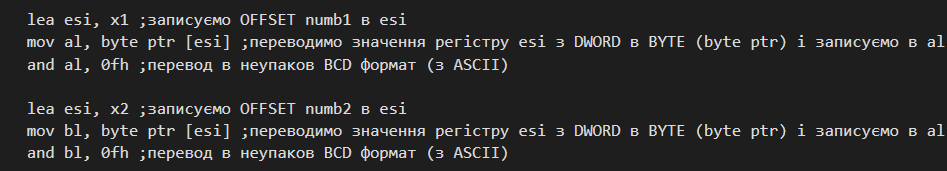
В підсумку, користувач запише два числа в масив numb1 і numb2. Екран користувача, матиме такий вигляд.

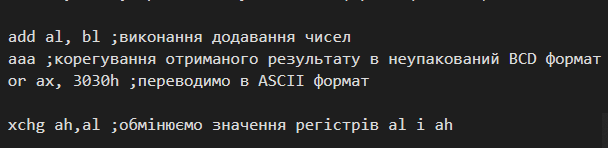


1. Далі використовуючи директиву INVOKE, відбувається виклик процедури addnumb. Передаємо параметри в процедуру numb1 і numb2.

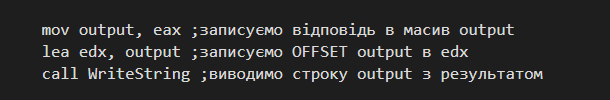
Відбувається перекид до частини коду, де описана процедура addnumb. Дана процедура має два параметри x1 і х2 типу DWORD, саме ці параметри будуть вказувати на сегменти стеку, де зберігаються значення (копії) змінних numb1 і numb2.

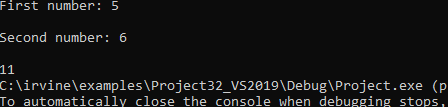
Cпочатку процедура addnumb виконує перевод чисел в неупаковані BCD формат. Використовуючи раніше описану функцію lea, записуємо адресу x1 (numb1) в регістр esi. За допомогою функції byte ptr [esi], переводимо значення регістру esi з типу DWORD в BYTE і записуємо в регіст al. Це потрібно, так як числа в BCD форматі мають тип ВYTE. Далі виконуючи логічне множення (опреація AND) переводимо значення, в регістрі al, з ASCII в неупакований BCD формат. Ще раз виконуємо описанні вище дії, але вже для другого числа x2 (numb2) і записуємо його в регіст bl.



Далі, відбувається операція додавання і корегування отриманого результату. Спочатку виконуємо додавання двух чисел, використовуючи функцію add al, bl, результат даної операції запишеться в регістр al в шістнадцятковому форматі. Так, як це було додавання, щоб скорегувати результат в неупакований BCD формат, використовуємо функцію aaa. Якщо результат додавання, буде більший ніж 9 (максимальне значення BCD числа це 9), або флаг переносу AF=1, то функція aaa, виконає додавання числа 6 до регістру al, щоб перевести це число в діапазон чисел від 0 до 9 і обнулить старші 4 біта регістру al, а до регістру ah додається 1. Отримане значення в регістрі ах буде записане в неупакованому BCD форматі. Використовуючи логічне додавання (операція OR), переводимо значення в регістрі ax, з ASCII в неупакований BCD формат. Під кінець процедури, використовуємо функцію xchg ah,al, це потрібно, щоб результат додавання коректно вивівся на екран (не задом наперед).

Так як, це кінець процедури використовуємо функцію ret, яка поверне управління команді, наступній за тією, яка викликала цю процедуру.

1. Кінець програми, це вивід результату. Cпочатку в масив результати output, закидується відповідь, що була обрахована за допомогою процедури addnumb. Вивід результату відбувається за допомогою функції call WriteString, але потрібно перед викликом цієї функції записати в edx, адресу строки, яку хочемо виводити.
2. Інформацію, яку буде бачити користувач на екрані, після завершення програми.



**Висновок**

В результаті виконання даної лабораторної роботи, було модифіковано програму з попередньої лабораторної роботи за допомогою використання процедур та операцій зі стеком. Як наслідок код програми тепер займає менше місця і став краще організованим та легшим для розуміння.