**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 №4

з дисципліни “Основи мікропроцесорної техніки”

Виконав: студент групи ДК-82

Рудюк Б. Б.

Перевірила: ст.вик.

Бондаренко Н. О.

**Київ – 2020**

**Тема.** Ознайомлення з системою команд мови асемблера, командами обробки даних, умовні переходи.

**Мета:** Вивчення способів представлення і обробки різних типів числових даних, що використовуються в програмах на асемблері, а також способів розгалуження у програмах із застосуванням команд умовних переходів.

**Теоретична частина**

**Додавання і віднімання двійкових чисел**

Для додавання та віднімання двійкових чисел із знаком і без знаку використовують одні і ті ж команди ADD або відповідно SUB (integer SUBtraction):

**ADD dst, scr ; dst = dst+src**

**SUB dst, scr ; dst = dst−src**

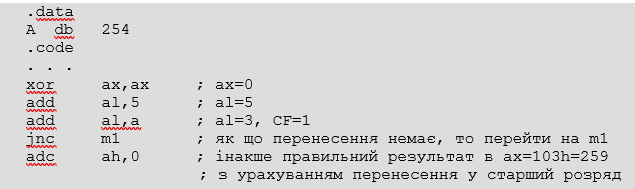
Операнди **dst** і **scr** можуть бути в регістрах (**r**) або пам’яті (**m**), а **scr** до того ж може бути числом, заданим безпосередньо (**imm**).

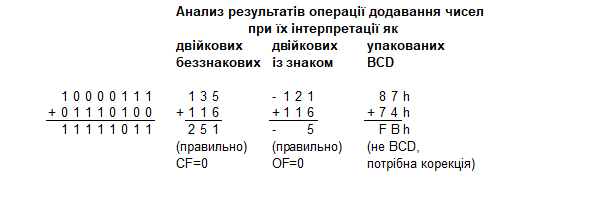
Для операцій багатократної точності, то б то для операцій з багаторозрядними числами, використовуються ще і команди, що враховують перенесення при складанні ADC (ADd with Carry) або позику SBB (integer SuBtraction with Borrow) при відніманні:

**ADC dst, scr ; dst = dst+src+CF**

**SUB dst, scr ; dst = dst−src−CF**

При додаванні чисел будь якої розмірності, як що значення результату перевищує розмірність поля результату (dst), то результат буде невірним. Наприклад, виконаємо додавання: 254 + 5 = 259 у двійковому виді 111111110 + 00000101 = 1 00000011. Правильний результат укладається у 9 біт, а у 8-бітном полі операнда dst8 осталось значення 3, що звичайно є невірним. Ознакою виходу результату за розрядну сітку у даному випадку є прапорець перенесення СF.

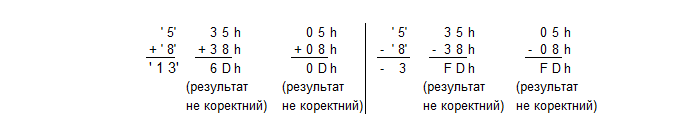
Програма має перевіряти прапорець CF після команди додавання і корегувати результат:

При складанні або відніманні двійкових чисел із знаком, представлених ланцюжком слів або байтів, схема виконання операції та ж, тільки ознакою виходу результату за межі допустимого діапазону служить прапорець OF = 1 після складання або віднімання самих старших розрядів. У наступному прикладі розглянуто додавання двох чисел командою ADD:

Оскільки при виконанні операцій ADD i SUB процесор «не знає» про існування чисел із знаком і без знака, то вся відповідальність за правильність інтерпретації отриманих результатів лежить на програмісті.

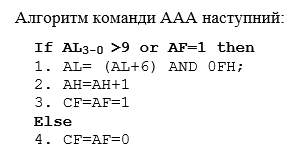
**Додавання і віднімання ASCII чисел (неупакований BCD формат)**

Як що данні вводяться з клавіатури, то вони будуть надані у ASCII форматі. Наприклад, число 1234 буде у пам’яті мати вид 31h 32h 33h 34h. Команди ADD i SUB виконують операції над такими числами як над звичайними двійковими числами, то б то враховують старші півбайти як значущі цифри. В наслідок цього результат може бути неправильний.

Розглянемо приклади додавання і віднімання ASCII чисел і не упакованих BCD чисел:

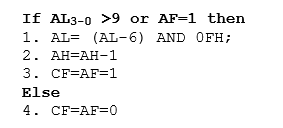
Для корекції результатів додавання і віднімання ASCII (або не упакованих BCD) чисел використовують інструкції:

* **AAA** (ASCII Adjust after Addition )

****Команда ААА корегує значення двійкової суми, отриманої після виконання команд ADD або ADC над десятковими неупакованими числами. В результаті сума, що знаходиться в регістрі AL, буде завжди відповідати представленню числа в форматі ASCII. Алгоритм команди ААА наступний:

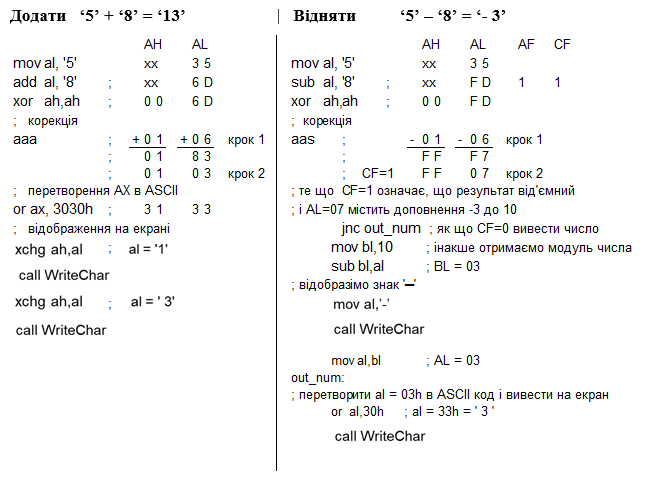
Команда аналізує вміст регістра AL і флагу АF. Прапори AF і CF змінюються в залежності від результату виконання команди. Решта прапорів мають невизначений стан.

* **AAS** (ASCII Adjust after Subtraction)

Команда ААS використовується після команд SUB або SBB, за допомогою яких одне неупаковане десяткове число віднімається від другого чилса і результат переміщується в регіст AL. В результаті різниця, яка знаходиться в регістрі AL, буде завжди відповідати представленню числа в форматі ASC. Алгоритм команди ААS наступний:

Виправлення різниці за допомогою інструкції AAS потрібні тільки в тому випадку, коли результатом є негативне число.

Приклади реалізації на асемблері додавання та віднімання десяткових чисел 5 і 8 у символьному поданні:



При додаванні не яких проблем не виникає. Але при відніманні, як що «*зменшуване*» (5) менш ніж «*від’ємник*» (8), результат ( «різниця» ) повинен бути від’ємним (−3). Однак він буде уявляти собою «десяткове» доповнення (07), то б то −10 + 7= − 3.

**Хід роботи**

**Завдання**

Додавання чисел в неупакованому BCD форматі. З клавіатури вводиться перший та другий доданок, а їхня сума виводиться на екран.

**Програмний код**

INCLUDE Irvine32.inc

.386

.model flat,stdcall

.stack 4096

ExitProcess proto,dwExitCode:dword

.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

lea esi, numb1 ;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, numb2 ;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

call WriteChar ;виводимо значення регістру al

xchg ah,al ;обмінюємо значення регістрів al і ah

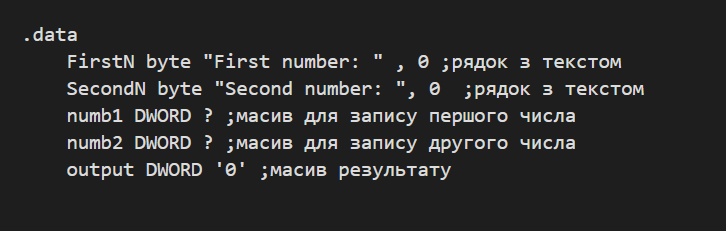
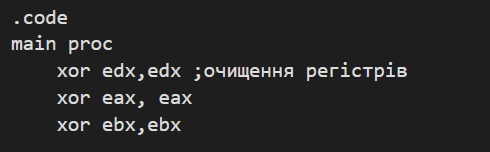
call WriteChar ;виводимо значення регістру al

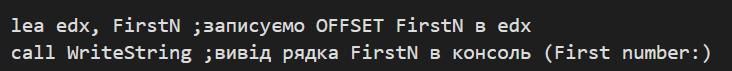
invoke ExitProcess,0

main endp

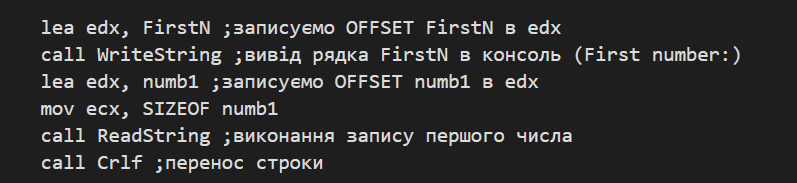
end main

**Опис програми**

1. У даному сегменті коду, зберігаються дані в пам’яті.
2. Перед початком вводу даних, відбувається очищення регістрів.
3. 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.

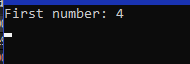


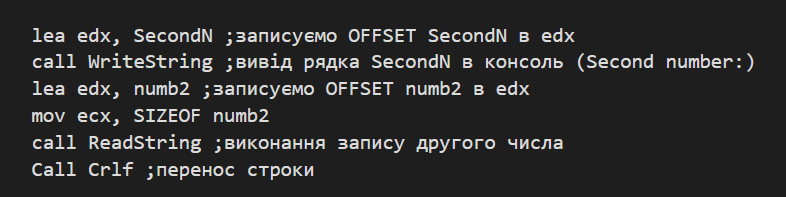
Далі відбувається підготовка регістрів для виклику функції call ReadString, щоб користувач міг записати дані використовуючи клавіатуру. У регістр edx записуємо OFFSET масива (numb1) куди будуть записуватися дані, а в регістр ecx передаємо SIZEOF (розмір) даного масива. Викликаємо функцію call ReadString.



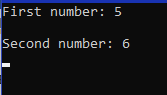
Після того, як користувач введе дані, використовуючи функцію call Crlf, перемістимо курсор на екрані монітора в першу позицію наступної строки.

Інформація, яку буде бачити користувач на екрані.

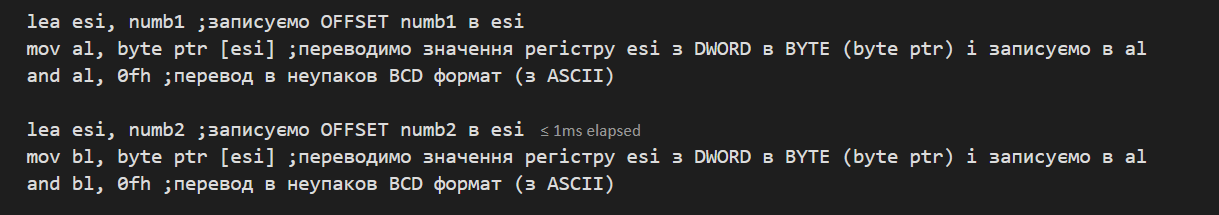


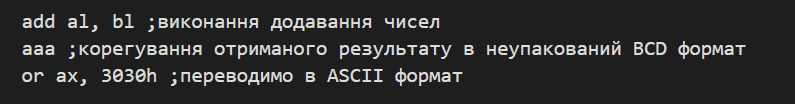
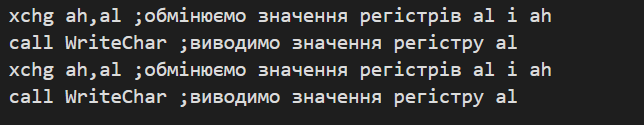
Так, як для додавання нам потрібно два числа, ще раз виконуємо описанні вище дії, але записуємо дані в вже в інший масив (numb2).

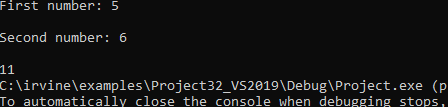
В підсумку, користувач запише два числа в масив numb1 і numb2. Екран користувача, матиме такий вигляд.



1. Далі відбувається збереження введених даних і перевод в неупаковані BCD формат. Спочатку, використовуючи раніше описану функцію lea, записуємо адресу numb 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 (numb2) і записуємо його в регіст bl.



1. У цьому сегменті коді, 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.
2. Кінець програми, це вивід результату. Спочатку виконуємо функцію xchg ah,al, якщо ця операція не буде виконуватися, то користувач побачить на екрані результат задом наперед. Для виводу використовуємо функцію call WriteChar, яка виводить значення, що записане в регістрі al.
3. Інформацію, яку буде бачити користувач на екрані, після завершення програми.



**Висновок**

В результаті виконання даної лабораторної роботи, було написано програму, яка додає два неупакованих BCD числа від 0 до 9. Також у ході виконання лабораторної роботи ознайомився з методами виконання арифметичних дій над упакованими та неупакованими BCD числами, та з командами, які корегують результати цих операцій.