**НАЦІОНАЛЬНИЙ технічний університет України**

**“Київський політехнічний інститут імені ігоря Сікорського”**

Інститут прикладного системного аналізу

Кафедра Системного проектування

**Лабораторна робота №5**

«Архітектура обчислювальних систем»

# «Дослідження адресації команд, стеку, віртуальної адресації»

Виконав:

студент групи ДА-51

Болобан Олег

Київ 2017

***Завдання:*** Дослідити адресацію команд, стеку, віртуальну адресацію.

Частина 1. Дослідження способів адресації команд.

1. Перерахувати способи адресації команд і пояснити їх особливості.

2. Скласти схеми обчислення адреси наступної команди для різних режимів адресації команд.

3. Пояснити призначення регістрів, використовуваних для адресації команд.

4. Пояснити використання стеку підчас виклику функцій.

Частина 2. Дослідження способів адресації стека.

1. Пояснити принципи організації стека. Визначте склад регістрів і пояснити призначення засобів адресації стека.

2. Оформити програму обробки масиву у вигляді головної програми і процедури. Зафіксувати зміни вмісту регістрів і стека при виклику підпрограми і при поверненні в основну програму. Пояснити ці зміни.

Частина 3. Дослідження організації віртуальної адресації.

1. Описати засоби віртуальної адресації пам'яті х86 в 32-розрядному захищеному режимі. Скласти схеми обчислення адрес.

***Дослідження способів адресації команд.***

В процесорах x86 за адресацію команд відповідає два регістри:

* CS - сегмент коду
* IP – регістр зміщення коду.

Зміна їх вмісту через звичайні команди (типу mov) неможлива, але існують спеціальні команди, які дозволяють це зробити неявно: JMP, CALL, LOOP і т.п.

*Існують наступні способи адресації команд:*

* Абсолютна - використовується для переходу по відомій адресі (напр. jmp 100);
* Відносна - вказується лише зміщення відносно поточної адреси команди;
* Регістрова - адреса наступної команди знаходиться в регістрі (jmp ax);
* Непряма - адреса наступної команди знаходиться в пам'яті (jmp [x]).

Для виклику власноруч написаних процедур зазвичай використовують команду CALL. Відмінність від простої команди переходу полягає в тому, що при виклику процедури в стек заноситься адреса команди, яка йде після її виклику.

Після виконання процедури використовують команду RET для виштовхування вершини стеку в IP і продовження виконання основної програми.

***Дослідження способів адресації стека.***

Стек — це спеціальним чином організована ділянка пам'яті, що використовується для: тимчасового зберігання змінних, передачі параметрів викликаним функціям та для зберігання адреси повернення при виклику процедур та переривань. Доступ до даної структури даних організований по принципу “останнім прийшов — першим пішов”.

*Для адресації стеку використовуються наступні регістри:*

* SS – сегмент стеку
* SP — зміщення вершини стеку. Особливість зміщення вершини стеку в тому, що при запису в стек його значення зменшується, тобто стек росте згори вниз.

*Для роботи з вершиною стеку призначені наступні команди:*

* push - зменшує зміщення вершини стеку на 2 та копіює значення вказанного регістру загального призначення;
* pop - копіює значення з вершини стеку у вказаний регістр загального призначення та збільшує зміщення вершини стеку на 2.
* pusha – завантаження в стек значень всіх 16 бітних регістрів.
* popa – вивантаження з стеку значення всіх 16 бітних регістрів.
* pushad – завантаження в стек значень всіх 32 бітних регістрів.
* popad – вивантаження з стеку значення всіх 32 бітних регістрів.
* pushaf(f) – завантаження в стек регістру признаків для 16(32) бітних процесорів.
* popad(f) – вивантаження з стеку регістру признаків для 16(32) бітних процесорів.

***Дослідження організації віртуальної адресації:***

Адресація в 32-розрядному режимі дворівнева:

* Сегментна:

Механізм організації віртуальної пам'яті, при якому віртуальний простір ділиться на частини довільного розміру - сегменти. Цей механізм дозволяє, наприклад, розбити дані процесу на логічні блоки. [2] Для кожного сегмента, як і для сторінки, можуть бути призначені права доступу до нього користувача та його процесів. При завантаженні процесу частина сегментів міститься в ОП (при цьому для кожного з цих сегментів ОС підшукує придатний ділянку вільної пам'яті), а частина сегментів розміщається в дисковій пам'яті. Сегменти однієї програми можуть займати в ОП несуміжні ділянки. Під час завантаження система створює таблицю сегментів процесу (аналогічну таблиці сторінок), в якій для кожного сегмента вказується початкова фізична адреса сегмента в оперативній пам'яті, розмір сегмента, правила доступу, ознака модифікації, ознака звертання до даного сегмента за останній інтервал часу і деяка інша інформація . Якщо віртуальні адресні простори декількох процесів включають той самий сегмент, то в таблицях сегментів цих процесів робляться посилання на один і той же ділянку оперативної пам'яті, в який даний сегмент завантажується в єдиному екземплярі. Система із сегментною організацією функціонує аналогічно системі зі сторінкової організацією: час від часу відбуваються переривання, пов'язані з відсутністю потрібних сегментів у пам'яті, при необхідності звільнення пам'яті деякі сегменти вивантажуються, при кожному зверненні до оперативної пам'яті виконується перетворення віртуальної адреси у фізичний. Крім того, при зверненні до пам'яті перевіряється, чи дозволений доступ необхідного типу до даного сегмента.  
  
Віртуальну адресу при сегментній організації пам'яті може бути представлено парою (g, s), де g - номер сегмента, а s - зсув у сегменті. Фізична адреса виходить шляхом додавання початкової фізичної адреси сегмента, знайденого в таблиці сегментів по номері g, і зсуву s.  
  
Недоліком даного методу розподілу пам'яті є фрагментація на рівні сегментів і більш повільне в порівнянні з сторінковою організацією перетворення адреси.

* Сторінкова :

У більшості сучасних операційних систем віртуальна пам'ять організовується за допомогою сторінкової адресації. Оперативна пам'ять ділиться на сторінки: області пам'яті фіксованої довжини (наприклад, 4096 байт), які є мінімальною одиницею виділеної пам'яті (тобто навіть запит на 1 байт від додатка призведе до виділення йому сторінки пам'яті). Процес звертається до пам'яті за допомогою адреси віртуальної пам'яті, який містить в собі номер сторінки і зміщення всередині сторінки. Процесор перетворить номер віртуальної сторінки на адресу відповідної їй фізичної сторінки за **допомогою буфера асоціативної трансляці**ї. Якщо йому не вдалося це зробити, то потрібно звернення до таблиці сторінок (так званий Page Walk), що може зробити або сам процесор, або операційна система (в залежності від архітектури) [1]. Якщо сторінку вивантажено з оперативної пам'яті, то операційна система підкачує сторінку з жорсткого диска (див. свопінг). При запиті на виділення пам'яті операційна система може «скинути» на жорсткий диск сторінки, до яких давно не було звернень. Критичні дані (наприклад, код запущених і працюючих програм, код і пам'ять ядра системи) зазвичай знаходяться в оперативній пам'яті (виключення існують, однак вони не стосуються тих частин, які відповідають за обробку апаратних переривань, роботу з таблицею сторінок і використання файлу підкачки).

Аналогічно реальному режиму, адреса складається з адреси початку сегменту та відносного зміщення. Але в захищеному режимі в сегментних регістрах замість адреси початку сегмента знаходяться спеціальні 16-розрядні структури — селектори, що мають наступний вигляд:

* біти 15...3 — номер дескриптору в таблиці;
* біт 2 — індикатор таблиці (0 — GDT, 1 – LDT);
* біти 1...0 — рівень привілегій запиту (RPL).

GDT та LDT – це, відповідно, таблиці глобальних та локальних дескрипторів.

Вони містять восьмибайтні структури, які називаються дескрипторами сегментів, в яких і знаходяться початкові адреси сегментів разом з іншою важливою інформацією:

1. Слово 3 (старше):

* біти 15...8: біти 31...24 бази;
* біт 7: біт гранулярності (0 — ліміт у байтах, 1 — у 4-кілобайтних одиницях);
* біт 6: біт розрядності (0/1 – 16/32-бітний сегмент);
* біт 5: 0;
* біт 4: зарезервовано для операційної системи;
* біти 3...0: біти 19...16 ліміта.

1. Слово 2:

* біт 15: біт присутності сегменту;
* біти 14...13: біти привілегії дескриптора (DPL);
* біт 12: тип дескриптору (0 — системний (глобальний), 1 - звичайний (користувацький));
* біти 11...8: тип сегменту;
* біти 7...0: біти 23...16 бази;

1. Слово 1: біти 15...0 бази;
2. Слово 0: біти 15...0 ліміту.

Основні поля в цій структурі — база та ліміт сегменту. База — це 32-бітна адреса початку сегменту, а ліміт — це 20-бітне число, який дорівнює розміру сегменту в байтах (залежить від біту гранулярності: 0 — 1б...1Мб, 1 — 4Кб...4Гб)

Інші елементи дескриптору виконують наступні функції:

Біт розрядності: для сегменту коду цей біт вказує на розрядність операндів і адрес по замовчуванню.

Поле DPL показує рівень привілеїв сегменту.

Біт присутності показує, чи є сегмент в пам'яті.

Біт типу дескриптора: 1 – звичайний сегмент коду чи даних, 0 — тип сегменту визначає поле типу сегмента.

Тип сегменту: для системних регістрів тут знаходиться число від 0 до 15, що визначає тип сегменту. Для звичайних сегментів коду та даних ці 4 біти виконують наступні функції:

біт 11: 0 — сегмент даних, 1 — сегмент коду;

біт 10: для даних — біт напрямку росту сегменту;

для коду — біт підпорядкування;

біт 9: для даних — біт дозволу запису;

для коду — біт дозволу читання;

біт 8: біт звертання.

Біт звертання встановлюється в 1 при завантаженні селектору цього сегменту в регістр.

Таблиця глобальних дескрипторів (GDT) — може бути лише одна, таблиці локальних дескрипторів (LDT) – одна на кожну задачу.

***Лістінг коду програми***

.386

.MODEL flat, stdcall

.data

array db 45,16,85,42,14,123,26,11,2

arraylen dd 9

sum dd 0

.code

evensum PROC a:PTR BYTE, alen:DWORD

LOCAL ssum:DWORD

mov ssum, 0

mov ecx, alen

LoopArray:

lea eax, a

mov eax, [eax]

xor ebx, ebx

mov bl, [eax+ecx]

test ebx, 1

.if zero?

add ebx, ssum

mov ssum, ebx

.endif

loop LoopArray

push ssum

ret

evensum ENDP

main PROC

push arraylen

push offset array

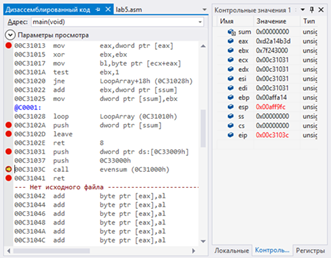
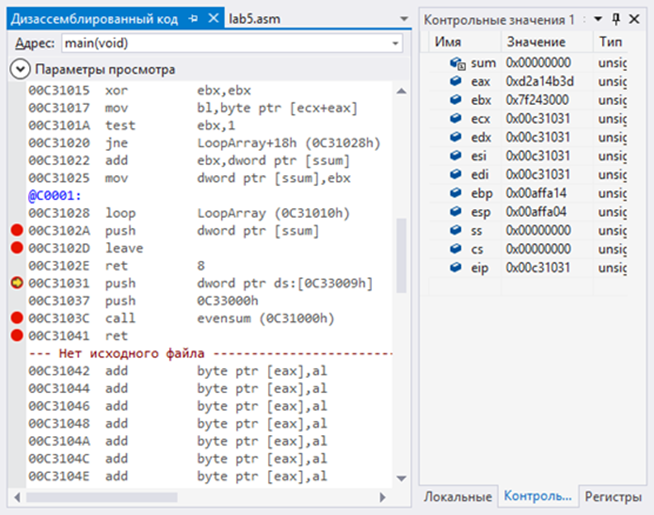
call evensum

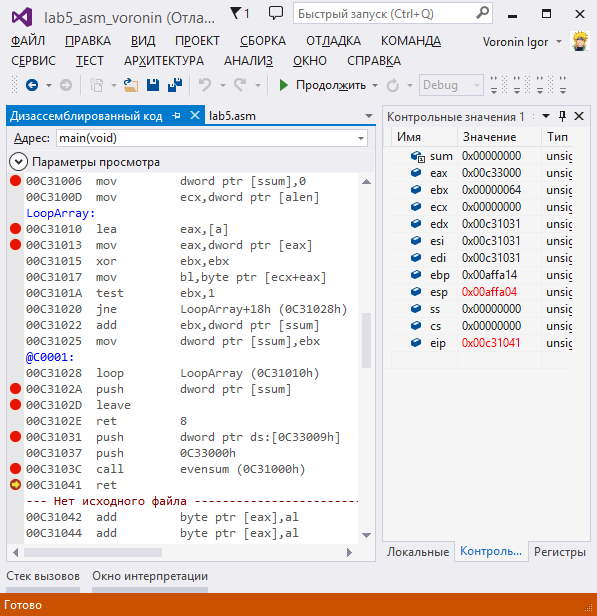
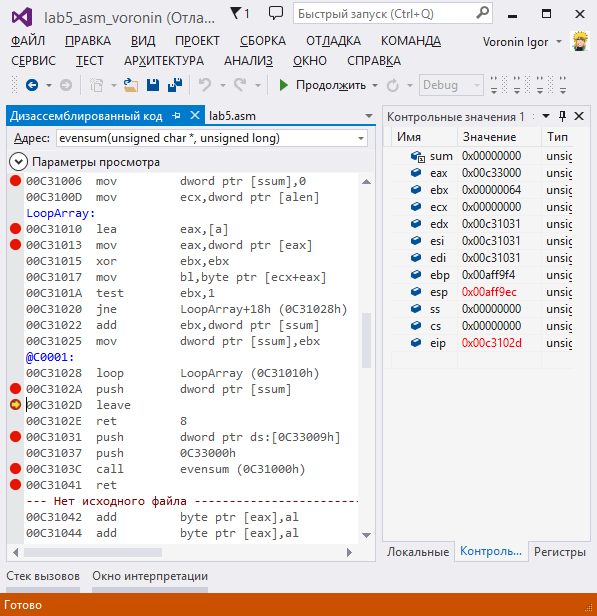
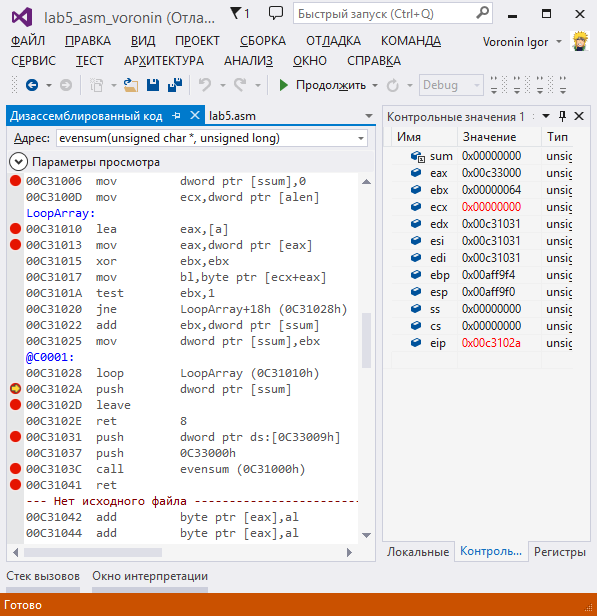
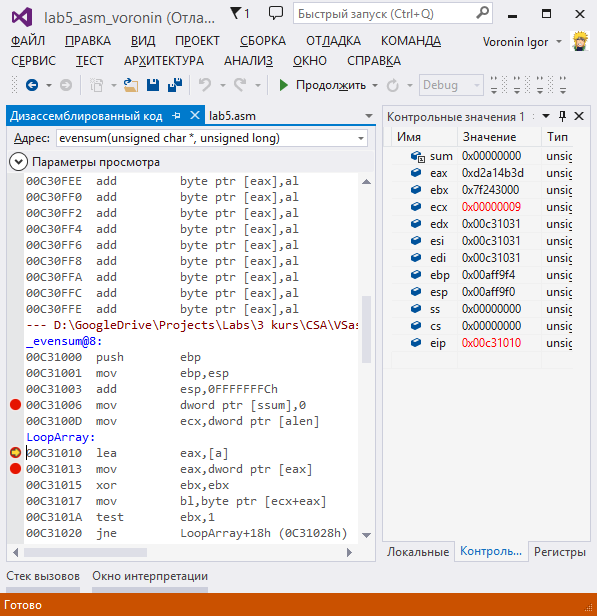
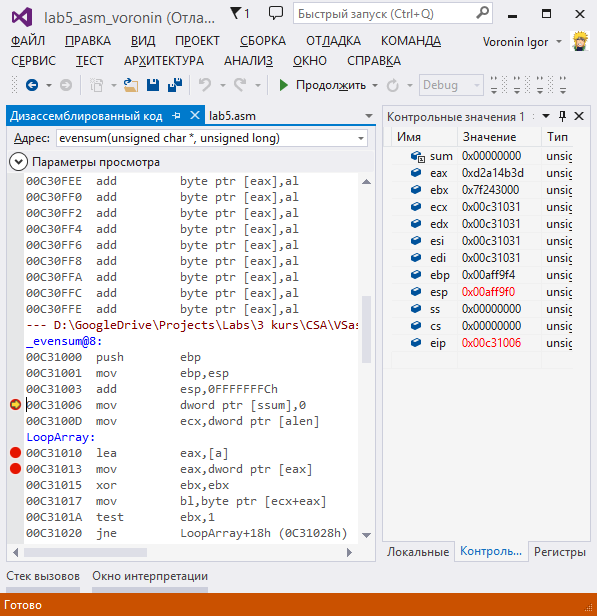
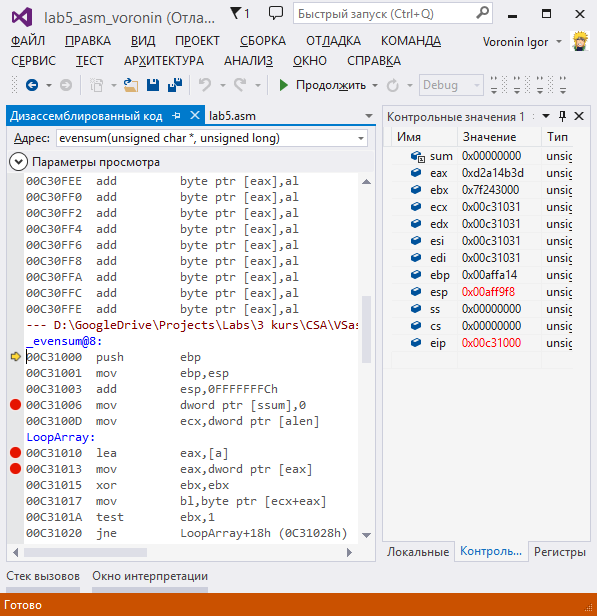
ret

main ENDP

END main

***Зміна регістрів і стеку при виклику і завершенні процедури з лабораторної роботи #4:***

******



На даному прикладі ми можемо спостерігати зміну значення регістру EIP з 0х00с3103с на 0х00с31000, до виконання процедури і під час неї. Також бачимо, що до поміщення даних до стеку змінює адрес верхівки стека з ESP = 0х00affa04 на ESP = 0х00aff9fс.

**Висновок**

В ході виконання лабораторної роботи були досліджені способи адресації команд, стеку, віртуальної адресації.