ЛАБОРАТОРНА РОБОТА № 2.

Дослідження способів представлення даних в пам'яті комп'ютера з архітектурою x86.

Мета: вивчити способи задання констант та змінних в Асемблері, набути навики інтерпретування даних в пам'яті комп'ютера з архітектурою x86.

Визначення даних в мові Assembler.

Асемблер допускає два основні методи визначення даних: через вказування довжини даних та за їх значенням. В першому випадку використовуються директиви визначення даних, а у другому безпосереднє значення або директива EQU.

Розглянемо основний формат визначення даних:

[ім'я] Dn вираз

- Ім'я елемента даних не є обов'язковим але якщо в програмі є посилання на деякий елемент, то це робиться за допомогою імені.
- Для визначення елементів даних є наступні директиви:

DB (байт)
DW (слово – 2 байти)
DD (подвійне слово – 4 байти),
DQ (8 байт)
DT (10 байт).

- Вираз може містити константу, наприклад: FLD1 DB 25,
- або знак питання для невизначеного значення, наприклад: FLDB DB ?

Вираз може містити кілька констант, розділених комами й обмеженими тільки довжиною рядка:

FLD3 DB 11, 12, 13, 14, 15, 16, ...

Асемблер визначає ці константи у вигляді послідовності сусідніх байт. Звертання за FLD3 вказує на першу константу 11, за FLD3+1 - на другу 12. (FLD3 можна представити як FLD3+0).

Наприклад команда MOV AL,FLD3+3 завантажує в регістр AL значення 14 (0Eh).

Вираз допускає також повторення константи в наступному форматі:

[ім'я] Dn кількість повторень DUP (вираз) ...

Символьні рядки

Символьний рядок використаються для опису даних, таких як, наприклад, імена людей або заголовки сторінок. Вміст рядка виділяється одинарними лапками, наприклад, 'PC' або подвійними лапками - "PC". Асемблер переводить символьні рядки в об'єктний код у звичайному форматі ASCII.

Символьний рядок визначається тільки директивою DB, в якій вказується більше двох символів у нормальній послідовності зліва направо. Отже, директива DB є єдиним можливим форматом для визначення символьних даних.

Числові константи

Числові константи використаються для арифметичних величин і для адрес пам'яті. Для опису константи лапки не ставляться. Асемблер перетворить всі числові константи в шістнадцятковий формат і записує байти в об'єктному коді в зворотної послідовності - справа наліво.

Десятковий формат. Десятковий формат допускає десяткові цифри від 0 до 9 і позначається останньою буквою D, що можна не вказувати, наприклад, 125 або 125D. Незважаючи на те, що асемблер дозволяє кодування в десятковому форматі, він перетворить ці значення в шістнадцятковий об'єктний код. Наприклад, десяткове число 125 перетвориться в. 7Dh.

Шістнадцятковий формат. Шістнадцятковий формат допускає шістнадцяткові цифри від 0 до F і позначається останньою буквою H. Оскільки асемблер думає, що з букви починаються ідентифікатори, те першою цифрою шістнадцяткової константи повинна бути цифра від 0 до 9. Наприклад, 2EH або 0FFFH, які асемблер перетворить відповідно в 2E і FF0F (байти в другому прикладі записуються в об'єктний код у зворотній послідовності).

Двійковий формат. Двійковий формат допускає двійкові цифри 0 і 1 і позначається останньою буквою B. Двійковий формат звичайно використається для більше чіткого подання бітових значень у логічних командах AND, OR, XOR і TEST. Десяткове 12, шістнадцяткове C і двійкове 1100B усі генерують один і той же код:

*Вісімковий формат. В*ісімковий формат допускає вісімкові цифри від 0 до 7 і позначається останньою буквою Q або O, наприклад, 253Q.

Формат з рухомою комою. Числові константи з рухомою комою задаються у форматі *ціла_частина.дробова_частина E степінь*. Де *E* – основа системи лічби (10). Стандарт IEEE 754-2008 визначає кілька форматів представлень чисел з рухомою комою. Розглянемо 3 з них:

- одинарної точності (32 біти): біт 31 – знак мантиси, біти 30-23 – 8-ми бітна зміщена на +127 експонента (порядок), біти 22-0 – 23-бітна мантиса без першої цифри. Для представлення у 10-тковій системі лічби використовується формула: $F = (-1)^S 2^{(E-127)}(1+M/2^{23})$, де S–знак, E – зміщена експонента, M – нормалізована мантиса.
- подвійної точності (64 біти): біт 63 – знак мантиси, біти 62-52 – 11-ти бітна зміщена на +1023 експонента (порядок), біти 51-0 – 52-бітна мантиса без першої цифри (нормалізована мантиса). Для представлення у 10-тковій системі лічби використовується формула: $F = (-1)^S 2^{(E-1023)}(1+M/2^{52})$, де S–знак, E – зміщена експонента, M – нормалізована мантиса.
- розширеної точності (80 біт): біт 79 – знак мантиси, біти 78-64 – 15-ти бітна зміщена на +16383 експонента (порядок), біти 63-0 – 64-бітна мантиса з першою цифрою (біт 63 рівний 1). Для представлення у 10-тковій системі лічби використовується формула: $F = (-1)^S 2^{(E-16383)}(M/2^{64})$, де S–знак, E – зміщена експонента, M – мантиса.

Беззнакові й знакові дані.

Числові поля можуть трактуватися як знакові, або як беззнакові. Це контролюється при написанні програми **самостійно**. Для беззнакових величин всі біти є бітами даних Для знакових величин лівий біт є знаковим.

Директива визначення байта (DB)

Один байт виражається двома шістнадцятковими цифрами. Для беззнакових величин за допомогою цієї директиви можна представити числа від 0 до 255 (FFh). Для знакових даних найбільше додатне число в одному байті це 7F, всі "більші" числа від 80 до FF представляють від'ємні значення. У десятковому виразі ці межі визначаються числами +127 і -128.

Директива визначення слова (DW)

Директива DW визначає елементи, які мають довжину в одне слово (два байти). . Два байти представляються чотирма шістнадцятковими цифрами. Для беззнакових величин за допомогою цієї директиви можна представити числа від 0 до 65535 (FFh). Для знакових даних найбільше додатне шістнадцяткове число у двох байтах це 7FFF; всі "більші" числа від 8000 до FFFF представляють від'ємні значення. У десятковому форматі ці межі визначаються числами +32767 і -32768.

Символьний вираз в DW обмежений двома символами, які асемблер представляє в об'єктному коді так, що, наприклад, 'PC' стає 'CP'. Для визначення символьних рядків директива DW має обмежене застосування.

Директиви визначення DD,DQ та **DT** є аналогічними до DW з відповідними довжинами даних.

Директива EQU

Директива EQU не визначає елемент даних, але визначає значення, що може бути використане для підстановки в інших командах. Припустимо, що в сегменті даних закодовано наступну директиву EQU:

TIMES EQU 10

Ім'я, у цьому випадку TIMES, може бути представлено будь-яким припустимим в асемблері ім'ям. Тепер, у якій би команді або директиві не використалося слово TIMES асемблер підставить значення 10. Наприклад, асемблер перетворить директиву

FIELDA DB TIMES DUP (?)

в директиву

FIELDA DB 10 DUP (?)

Ім'я, пов'язане з деяким значенням за допомогою директиви EQU, може використатися в командах, наприклад:

COUNTR EQU 05

.....
MOV CX,COUNTR

Асемблер заміняє ім'я COUNTR у команді MOV на значення 05, створюючи операнд з безпосереднім значенням, так, ніби було закодовано MOV CX,05 ;Асемблер підставляє 05.

Перевага директиви EQU полягає в тому, що багато команд можуть використати значення, визначене іменем COUNTR. Якщо це значення повинне бути змінене, то зміні підлягає лише одна директива EQU. Природно, що використання директиви EQU доцільне лише там, де підстановка має сенс для асемблера. У директиві EQU можна використати символічні імена:

1. TP EQU TOTALPAY
2. MPY EQU MUL

Перший приклад припускає, що в сегменті дані програми визначене ім'я TOTALPAY. Для будь-якої команди, що містить операнд TP, асемблер замінить його на адресу TOTALPAY. Другий приклад показує можливість використання в програмі слова MPY замість звичайного мнемокоду MUL.

Директива LABEL

Директива LABEL створює мітку, яка не займає місця в пам'яті, а просто вказує на адресу команди або дані, що слідують за нею. Має наступний формат:

мітка LABEL тип

Мітка – це назва містки, параметр *тип* може приймати одне з значень: BYTE (1 байт), WORD (2 байти), DWORD (4 байти), FWORD (6 байт), QWORD (8 байт), TBYTE (10 байт), NEAR (ближня мітка), FAR (дальня мітка). Мітка отримує значення рівне зміщенню в сегменті адрес наступної за нею команди або зміщенню в сегменті даних наступного за нею операнду і тип вказаний явно. За допомогою міток з різними типами зручно організовувати доступ до одних і тих самих даних, що інтерпретуватимуться по різному в залежності від типу мітки. Наприклад, як байти чи як слова. Або здійснювати ближні чи дальні переходи.

Контрольні запитання:

1. Як визначаються в Асемблері дані довжиною в 2 байти? Як вони записуються у пам'ять?
2. Чим відрізняються директива EQU від DW ?
3. Як у пам'яті визначається число з рухомою комою?
4. Як інтерпретувати число з рухомою комою з розширеною точністю?

Література:

1. Р.Джордейн.Справочник програмиста персональных компъютеров типа IBM PC XT и AT. - М."Финансы и статистика",1992,стор.13-31.
2. Л.О.Березко,В.В.Троценко. Особливості програмування в турбо-асемблері. -Київ,НМК ВО,1992.
3. Л.Дао. Программирование микропроцессора 8088.Пер.с англ.-М."Мир",1988.
4. П.Абель.Язык ассемблера для IBM PC и программирования. Пер. з англ.-М.,"Высшая школа",1992.
5. Зубков С.В. Assembler для DOS, WINDOWS и UNIX. – М.: ДМК Пресс, 2000. с.106-126.

ЗАВДАННЯ

1. Створити *.exe програму, яка розміщує в пам’яті даних комп’ютера, операнди, що задані варіантом. Вхідні операнди A, B, C, D, E, F з індексом u вважати без знаковими і довжиною в байтах, згідно з індексу, з індексом fs вважати з рухомою комою одинарної точності (32 біти), з індексом fd вважати з рухомою комою подвійної точності (64 біти), з індексом fe вважати з рухомою комою розширеної точності (80 біт); крім цього операнд A є масивом з 3-ох елементів. При оголошені призначити операндам початкові значення використовуючи всі можливі системи лічби. K – константа, довжина якої визначається значенням(згідно варіанту), а значення задане в шістнадцятковому форматі. Для її опису слід використати директиву EQU. Задати одну мітку в довільному місці сегменту даних. Задати в сегменті даних змінну Message db '*Прізвище*',13,10, , де '*Прізвище*' – прізвище виконавця роботи, яке вивести на екран.
2. За допомогою меню Debug середовища Visual Studio 2019, дослідити представлення даних в пам’яті комп’ютера (продемонструвати розміщення даних та здійснити інтерпретацію).
3. Скласти звіт про виконану роботу з приведенням тексту програми з коментарями, дампу пам’яті та аналітично інтерпретувати дані для кожної з змінних.
4. Дати відповідь на контрольні запитання.

ВАРІАНТИ ЗАВДАНЬ:

№	Операнди	K
1	A_2,B_{1u},C_4,D_{fd},E_{10},F_3, K	1254021
2	A_2,B_{4u},C_{fs},D_{8u},E_{10},F_5, K	202
3	A_4,B_{3u},C_4,D_{fd},E_{10},F_8, K	37788663
4	A_1,B_{4u},C_{fs},D_{10u},E_1,F_8, K	45694
5	A_8,B_{1u},C_4,D_{fd},E_{10},F_5, K	505
6	A_2,B_{4u},C_{fs},D_{8u},E_{10},F_7, K	6DD02316
7	A_4,B_{5u},C_4,D_{fs},E_{10},F_8, K	717
8	A_4,B_{2u},C_{ds},D_{10u},E_1,F_8, K	88
9	A_1,B_{3u},C_4,D_{fd},E_8,F_2, K	29
10	A_2,B_{5u},C_{fs},D_{8u},E_{10},F_4, K	2310
11	A_4,B_{3u},C_4,D_{fd},E_8,F_8, K	311
12	A_2,B_{4u},C_{fd},D_{10u},E_1,F_8, K	7055E0AC
13	A_8,B_{1u},C_4,D_{fe},E_2,F_5, K	2513
14	A_2,B_{3u},C_{fd},D_{8u},E_{10},F_1, K	614
15	A_4,B_{5u},C_4,D_{fe},E_{10},F_8, K	4569600F
16	A_1,B_{3u},C_{fe},D_{10u},E_4,F_8, K	616
17	A_1,B_{1u},C_3,D_{fe},E_{10},F_4, K	1017
18	A_2,B_{4u},C_{fs},D_{8u},E_{10},F_5, K	56987018
19	A_4,B_{3u},C_4,D_{fd},E_{10},F_8, K	4019
20	A_8,B_{4u},C_{fe},D_{10u},E_1,F_8, K	18932020
21	A_{10},B_{1u},C_4,D_{fs},E_8,F_5, K	21
22	A_8,B_{4u},C_{fd},D_{8u},E_{10},F_7, K	45781022
23	A_4,B_{5u},C_4,D_{fe},E_{10},F_8, K	7AA02023
24	A_2,B_{2u},C_{fs},D_{10u},E_1,F_8, K	74569024
25	A_1,B_{3u},C_4,D_{fd},E_8,F_4, K	2B05025
26	A_2,B_{6u},C_{fs},D_{8u},E_{10},F_4, K	6C26
27	A_4,B_{3u},C_2,D_{fs},E_8,F_8, K	A77627
28	A_8,B_{4u},C_{fd},D_{10u},E_1,F_2, K	3FF28
29	A_{10},B_{1u},C_4,D_{fe},E_2,F_5, K	12A0C029
30	A_2,B_{6u},C_{fd},D_{8u},E_{10},F_1, K	25630

Приклад виконання

Завдання: Написати програму, яка розміщує в пам'яті даних комп'ютера, операнди, що задані варіантом.
Вхідні операнди A, B, C, D, E, F з індексом u вважати без знаковими і довжиною в байтах, згідно з індексу, з індексом fs вважати з рухомою комою одинарної точності (32 біти), з індексом fd вважати з рухомою комою подвійної точності (64 біти), з індексом fe вважати з рухомою комою розширеної точності (80 біт); операнд A є масивом з 3-ох елементів. При оголошені призначити операндам початкові значення використовуючи всі можливі системи лічби. K – константа, довжина якої визначається значенням(згідно варіанту), а значення задане в шістнадцятковому форматі. Для її опису слід використати директиву EQU. Задати одну мітку в довільному місці сегменту даних. В кінці розташувати змінну HelloMessage db 'Hello, world',13,10, яку вивести на екран.

Задані операнди:

$$A_2, B_{3u}, C_4, D_{fe}, E_1, F_8,\ K = 52A8h$$

Операнд **B** є трибайтним, його можна представити трьома однобайтними, або двома двобайтними числами, або одним 4 байтним з старшим байтом рівним 0, оскільки він є беззнаковим. Представлення двобайтними числами є оптимальним варіантом з точки зору ефективності проведення обчислень з цим числом.

Отже, сегмент даних може буде описаний таким чином:

```
A       dw    0123Q, 0ABCh, -9874
B       dw    00A5h, 0AB3Ch
Cc      dd    512h
LBL     LABEL BYTE
D       dt    3.14e8
E       db    11111010b
F       dq    12356789ABCDEFh
K       equ   52A8h
HelloMessage db 'Hello, world',13,10
```

Дамп пам'яті відображено на рис.1, де кожна змінна виділена окремим кольором. Константа K не заноситься в пам'ять, а заміняється числом, що їй відповідає, у тексті програми на етапі розбору коду препроцесором.

Щоб переглянути дамп пам'яті необхідно запустити програму в покроковому режимі (F10), а далі вибрати в меню Debug\Windows\Memory одне з чотирьох вікон відображення вмістимого пам'яті.

Щоб знайти адресу, з якої починається розміщення даних у пам'яті, можна скористатись такою хитрістю – дивимось вмістиме регістру `EIP = 01141000` на початку виконання програми, додаємо до нього значення 3000 і там шукаємо дані програми.

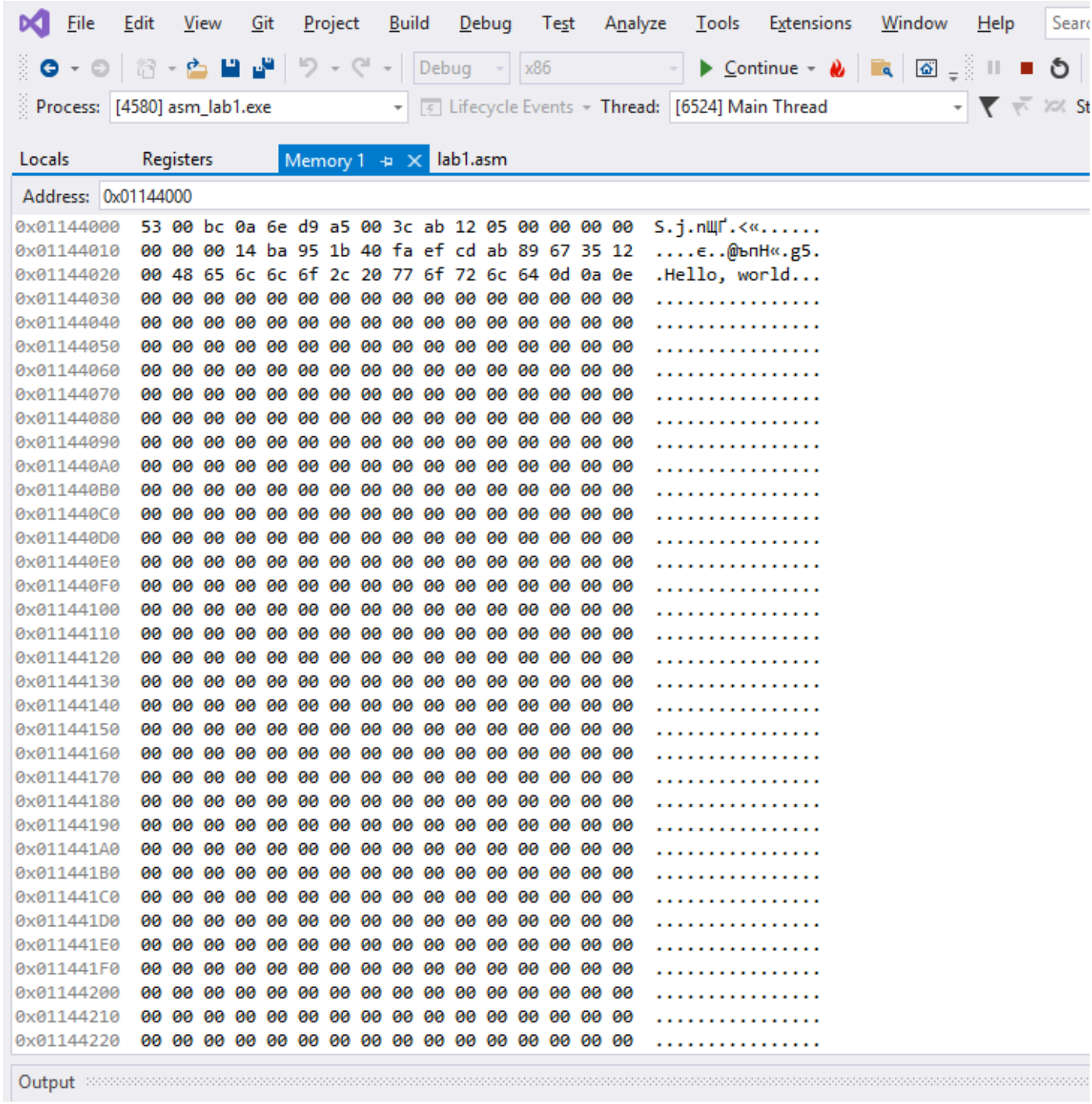

Рис.1. Вмістиме дампу пам’яті.

Далі залишиться проаналізувати і проінтерпретувати отримані дані.

```
0x01144000  53 00 bc 0a 6e d9 a5 00 3c ab 12 05 00 00 00 00  S.j.nЩґ.<«.....
0x01144010  00 00 00 14 ba 95 1b 40 fa ef cd ab 89 67 35 12  ....є..@ъпН«.g5.
0x01144020  00 48 65 6c 6c 6f 2c 20 77 6f 72 6c 64 0d 0a 0e  .Hello, world...
```

```
A       dw  0123Q, 0ABCh, -9874
B       dw      00A5h, 0AB3Ch
Cc      dd      512h
LBL     LABEL   BYTE
D       dt      3.14e8
E       db      11111010b
F       dq      12356789ABCDEFh
K       equ 52A8h
HelloMessage db 'Hello, world',13,10
```

Перевіримо число з плаваючою комою:

3.14e8 = 3.14 * 10^8 = 314000000 = $10010101101110100001010000000_2$ =
$1.00101011011101000010100000000*2^{28}$
Знак числа = 0
Порядок = 28 + 16383 = 16411 = 100000000011011
Мантиса = 10010101101110100001010000000…0
Внутрішнє представлення числа = 0 100000000011011 10010101101110100001010000000…0 або
0100 0000 0001 1011 1001 0101 1011 1010 0001 0100 0000 … 0000 або
40 1b 95 ba 14 00 00 00 00 00

Лістинг програми

```
.686
.model flat, stdcall
option casemap:none
include \masm32\include\windows.inc
include \masm32\include\kernel32.inc
includelib \masm32\lib\kernel32.lib

.data
A      dw  0123Q, 0ABCh, -9874
Bdw    00A5h, 0AB3Ch
Cc     dd   512h
LBL    LABEL BYTE
Ddt    3.14e8
Edb    11111010b
Fdq    12356789ABCDEFh
K      equ 52A8h
HelloMessage db 'Hello, world',13,10
NumberOfCharsToWrite dd $-HelloMessage
hConsoleOutput dd 0
NumberOfCharsWritten dd 0

.code
start:
push -11
call GetStdHandle
mov hConsoleOutput, eax
push 0
push offset NumberOfCharsWritten
push NumberOfCharsToWrite
push offset HelloMessage
push hConsoleOutput
call WriteConsoleA
push 0
call ExitProcess
end start
```