

.586 ; подключение набора команд

Реntium

.MODEL flat, stdcall ; модель памяти и

; конвенция о передаче параметров

OPTION CASEMAP:NONE ; опция различия строчных

; и прописных букв

Include kernel32.inc ; подключение описаний процедур

и

Include masm32.inc ; констант

IncludeLib kernel32.lib ; подключение библиотек

IncludeLib masm32.lib

.CONST ; начало раздела констант

MsgExit DB "Press Enter to Exit",0AH,0DH,0

.DATA ;раздел инициализированных переменных

.DATA? ;раздел неинициализированных переменных

inbuf DB 100 DUP (?)

.CODE ; начало сегмента кода

Start:

;

; Add you statements

;

XOR EAX,EAX

Invoke StdOut,ADDR MsgExit

; вывод сообщения

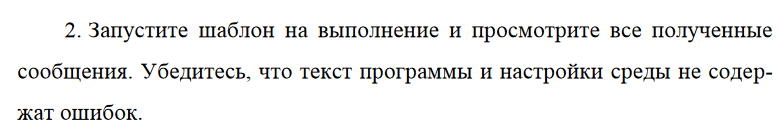
Invoke StdIn,ADDR inbuf,LengthOf inbuf

; ввод строки

Invoke ExitProcess,0

End Start

; завершение программы



**Сообщения среды после ассемблирования:**

C:\Masm32\Bin\ML.EXE /c /coff /Cp /nologo /I"C:\Masm32\Include" "lab1.asm"

Assembling: lab1.asm

Make finished.

Total compile time 125 ms

**Сообщения среды после компоновки:**

C:\Masm32\Bin\LINK.EXE /SUBSYSTEM:CONSOLE /RELEASE /VERSION:4.0 /LIBPATH:"C:\Masm32\Lib" /OUT:"lab1.exe" "lab1.obj"

Microsoft (R) Incremental Linker Version 5.12.8078

Copyright (C) Microsoft Corp 1992-1998. All rights reserved.

Make finished.

Total compile time 125 ms

**Сообщения среды после запуска программы:**

Executing:

"C:\Users\Trickster2038\Desktop\BmstuLabs4\lab1\lab1\lab1.exe"

Make finished.

Total compile time 110 ms

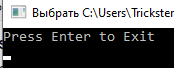
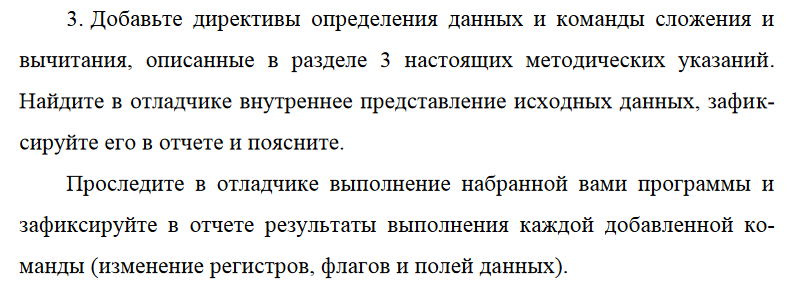


Рисунок 1 — информационное сообщение консоли

Все этапы запуска шаблона завершены успешно - ошибок нет, программа запускается, о чем свидетельствует сообщение на рисунке 1.



**Исходный код:**

; Template for console application

.586

.MODEL flat, stdcall

OPTION CASEMAP:NONE

Include kernel32.inc

Include masm32.inc

IncludeLib kernel32.lib

IncludeLib masm32.lib

.CONST

MsgExit DB "Press Enter to Exit",0AH,0DH,0

.DATA

A SDWORD -30

B SDWORD 21

.DATA?

inbuf DB 100 DUP (?)

X SDWORD ?

.CODE

Start:

;

; Add you statements

;

mov EAX, A

add EAX, 5

sub EAX, B

mov X, EAX;

; XOR EAX,EAX

; Invoke StdOut,ADDR MsgExit

;Invoke StdIn,ADDR inbuf,LengthOf inbuf

; Invoke ExitProcess,0

End Start

**Коды команд во внутреннем представлении:**

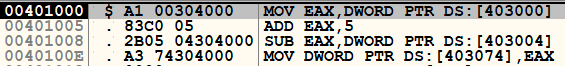


Рисунок 2 — коды машинных команд

На рисунках 3-8 представлено содержимое памяти и регистра EAX во время исполнения программы в режиме пошаговой отладки. Данные проанализированы в поясняющем тексте ниже.

**Начальные значения:**

EAX 0019FFCC



Рисунок 3 — содержимое памяти в начале работы программы

Значения при адресе команды 00401005:

EAX FF FF FF E2



Рисунок 4 — содержимое памяти во время работы программы

Значения при адресе команды 00401008:

EAX FF FF FF E7



Рисунок 5 — содержимое памяти во время работы программы

Значения при адресе команды 0040100E:

EAX FF FF FF D2



Рисунок 6 — содержимое памяти во время работы программы

Значения при адресе команды 00401013:

EAX FF FF FF D2



Рисунок 7 — содержимое памяти во время работы программы

Значение переменной Х = FF FF FF D216



Рисунок 8 — содержимое памяти вконце работы программы

**Пояснения:**

Выражение и значения из задания преобразуются в 16-ричную СС следующим образом:

А = -3010 = FFE216

В = 2110 = 1516

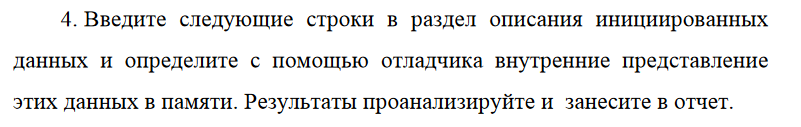
Х = А+5-В = -3010+510-2110 = -4610 = FFE216 + 516 - 1516 = FFD216

Примечание: FF FF FF E716 = -2510 = -3010 + 510

В соответствии с особенностями процессора IA-32 байты числа хранятся в памяти в обратном порядке, а в регистре - в прямом.

Также в соответствии с описанием числа имеют тип двойного слова со знаком => занимают в памяти по 4 байта.

Отрицательные числа хранятся в дополнительном коде.



A SDWORD -30

B SDWORD 21

val1 BYTE 255

chart WORD 256

lue3 SWORD -128

v5 BYTE 10h

v BYTE 100101B

beta BYTE 23,23h,0ch

sdk BYTE "Hello",0

min SWORD -32767

ar DWORD 12345678h

valar BYTE 5 DUP (1, 2, 8)

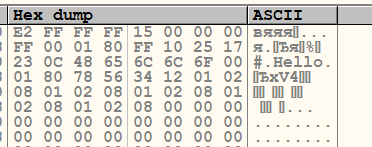


Рисунок 9 — содержимое памяти после объявления переменных

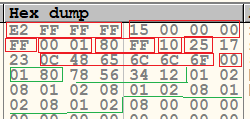


Рисунок 10 — выделенные значения отдельных переменных в памяти

**Пояснения:**

Каждой строке объявления переменных соответствует выделенное рамкой значение в памяти.

Байты числа/символов строки хранятся в обратном порядке.

Запись beta BYTE 23, 23h, 0ch объявляет в памяти значения 3 байт подряд(2310 = 1716, 23, 0c - уже в 16-ричной системе так заканчиваются буквой h обозначающей 16-ричный литерал).

Запись valar BYTE 5 DUP (1,2,8) дублирует 5 раз в памяти последовательность из 3 байт со значениями 1,2,8.

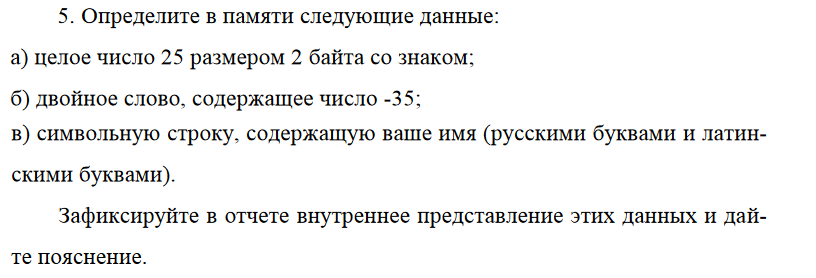
Строка Hello записана по байтам в кодировке ASCII.

Числа со отрицательным знаком представлены в дополнительном коде, число -128 занимает два байта т.к. имеет тип SWORD

12810 = 00000000 100000002

-12810 = 11111111 011111112 + 12 = 11111111 100000002 = FF 8016

Остальные объявленные данные записываются по аналогичным алгоритмам.



**Фрагмент кода программы:**

A1 SWORD 25

A2 DWORD -35

B1 BYTE "Sergey"

B2 BYTE "Сергей"

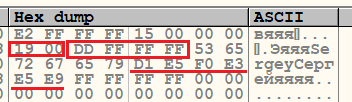


Рисунок 11 — содержимое памяти после объявления переменных

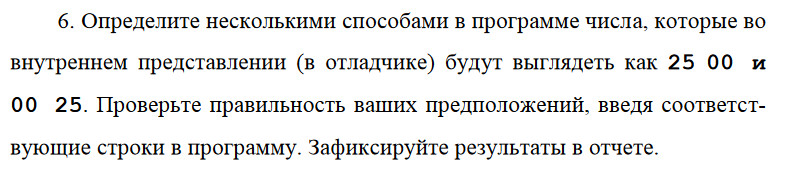
Положительное число 25 представлено в прямом коде в 16 С/С, байты записаны в обратном порядке, число занимает 2 байта согласно описанию типа SWORD

2510 = 1916

Отрицательное число -35 представлено в дополнительном коде(внутреннее представление не зависит от объявленного типа)

3510 = 00000000 00000000 00000000 001000112

-3510 = 11111111 11111111 11111111 110111002 + 12 = 11111111 11111111 11111111 1101110116 = FF FF FF DD16

Строки независимо от раскладки записываются по байтам в кодировке ASCII (расшифровку можно видеть справа).

**Фрагмент кода программы:**

X1 WORD 25h ; байты в обратном порядке

X2 BYTE 25h,00 ; байты в порядке перечисления

X3 SWORD 100101B ; байты в обратном порядке

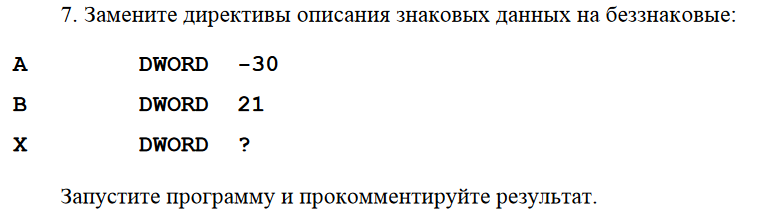
X4 WORD 2500h ; 16-ричный литерал сохранен в исходной форме

X5 SWORD 9472 ; байты в обратном порядке

На рисунке 12 видно что все переменные объявлены верно и соответсвуют требованиям задания:



Рисунок 12 - содержимое памяти после объявления переменных



На рисунках 13-15 приведено содержимое памяти в начале и по завершении работы программы. Полученные результаты проанализированы в пояснении ниже.

**Начальные значения:**

EAX 0019FFCC



Рисунок 13 - содержимое памяти после объявления переменных

**Конечные значения:**

EAX FF FF FF D2



Рисунок 14 - содержимое памяти после завершения работы программы

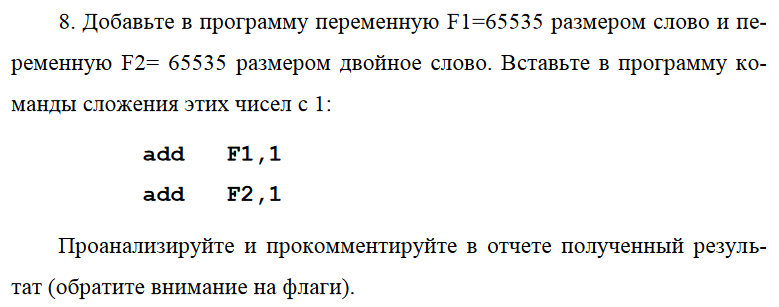
Значение переменной Х:



Рисунок 15 - содержимое памяти после завершения работы программы

**Пояснение:**

Числа знаковых и беззнаковых типов хранятся одинаково во внутреннем представлении => результат вычислений для исходного выражения не изменится при смене типа, изменится лишь его интерпретация при выводе в консоль.



F1 WORD 65535

F2 DWORD 65535

Флаги после выполнения add F1, 1:

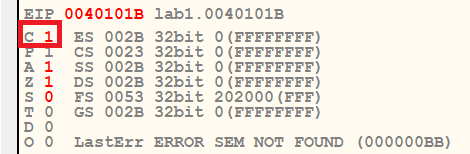


Рисунок 16 — значения флагов после операции сложния с перменной F1

В данном случае, как видно на рисунке 16, активировался флаг переноса С и флаг нуля Z так как в 1 байт невозможно записать значение больше 65535 => происходит переполнение разрядной сетки и возвращение 0 как результата операции.

Флаги после выполнения add F2, 2:

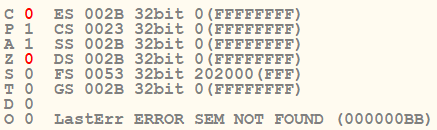
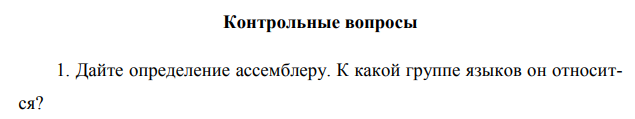


Рисунок 17 — значения флагов после операции сложния с перменной F2

В данном случае, как видно на рисунке 17, число хранится в 2 байтах памяти => переполнения не происходит и флаги C и Z остаются равными 0.

Язык ассемблера - язык низкого уровня, команды которого обычно соответствуют командам процессора. Относится к группе машинно-зависимых языков.



Для создания заготовки программы в RadASM необходимо создать новый проект, выбрать ассемблер, тип и шаблон проекта, типы создаваемых файлов и пункты меню необходимые для работы с проектом.

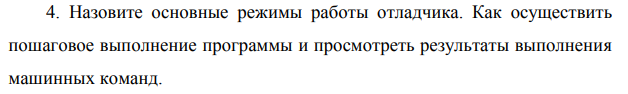
Заготовка содержит:

* указание настроек для транслятора, подключение описаний процедур и библиотек
* разделы объявления констант и переменных
* сегмент кода, завершающийся вызовом ExitProcess



Чтобы запустить программу, необходимо пройти следующие этапы обработки:

* Трансляцию(ассемблирование) - программа преобразуется из мнемонических (словесных) команд в машинные (двоичные)Компоновка - к двоичному коду основной программы добавляются объектные коды используемых подпрограмм
* Запустить программу/ запустить программу в режиме отладки

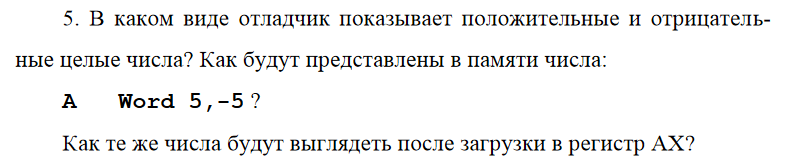


Основные режимы работы отладчика - с заходом и без захода в тело процедуры.

Для начала отладки необходимо транслировать и скомпоновать программу, затем выбрать опцию Run w debug.

Далее для выполнения шага с заходом в процедуру необходимо нажимать F7, без захода - F8.

Коды машинных команд видны в левом верхнем углу, содержимое памяти - в левом нижнем, содержимое регистров и флагов - в правом верхнем, стека - в правом нижнем.



5 => 05 00

-5 => FB FF

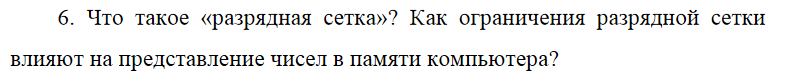
В регистре AX:

510 => 00 0516

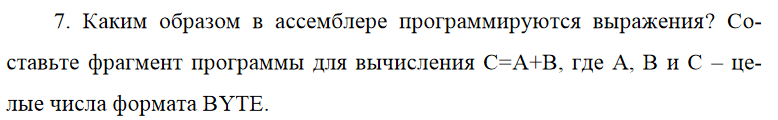
-510 => FF FB16 = 11111111 11111111 11111111 111110112 - дополнительный код -5

В памяти байты чисел представлены в обратном порядке, отрицательные числа хранятся в дополнительном коде.

В регистре байты становятся в прямой порядок.



Под разрядной сеткой понимают количество разрядов, выделенное в ЭВМ под запись 1 числа. Разрядная сетка определяет диапазон значений для целых чисел (причем для чисел со знаком он в 2 раза меньше чем для чисел без знака той же разрядности) и точность для дробных чисел.



Любое математическое выражение в ассемблере имеет не более двух операндов, поэтому любое сложное выражение необходимо разбивать на последовательность простых.

Фрагмент программы:

.Data

A BYTE 1

B BYTE 4

.Data?

C BYTE ?

.CODE

Start:

mov AX, A

add AX, B

mov C, AX

Вывод: в ходе работы были изучены основы работы со средой RadAsm, отладчиком OllyDbg, основы программирования на языке ассемблера(объявление переменных и констант, команда MOV, запуск программы), особенности внутреннего представления данных.