# Введение

Целью данного курса является практическое освоение x86-архитектуры современных процессоров. Вся система команд процессора разделена на группы, на лабораторных занятиях по отдельности рассматривается какая-то часть системы команд ЭВМ и выполняются практические задачи по программированию. После выполнения всех работ студент должен получить целостное представление о работе команд центрального процессора, сопроцессора, механизмах адресации, хранении машинного кода и данных в оперативной памяти. Курс включает приобретение навыков разработки   
32-разрядных консольных приложений на ассемблере для ОС Windows с использованием пакета masm32. Отдельное внимание уделяется разработке динамических dll-библиотек на ассемблере, стилям вызова подпрограмм, способам соединения ассемблерного кода с другими программами, вопросам отладки и оптимизации кода.

Каждая лабораторная работа включает: краткие теоретические сведения, необходимые для выполнения работы, 25 вариантов заданий для выполнения к работе, примеры выполнения заданий, контрольные вопросы к защите.

**Порядок проведения и защиты лабораторных работ.** Лабораторные занятия проводятся в компьютерных залах. Каждый студент выполняет вариант задания с тем номером, который он имеет в журнале посещаемости группы, ведомый старостой.

После выполнения лабораторной работы требуется составить по ней отчёт. Если отчёт удовлетворяет требованиям, предъявляемым к лабораторным работам, то работа допускается к защите. Защита заключается в проверке преподавателем теоретических знаний и практических навыков по выполненной лабораторной работе.

**Требования к отчётам лабораторных работ**. Отчёт к работе составляется в печатном виде на листах формата A4. Отчёт обязательно должен включать:

титульный лист с номером работы, её названием, ФИО выполнившего работу студента, ФИО преподавателя, дату сдачи и другие необходимые данные;

со второй страницы отчёт должен содержать номер работы, её название, цель работы, номер варианта задания и само задание, порядок выполнения работы, исходный код программ для решения задачи, экранные снимки полученных программ, набор тестовых данных для проверки работы программы. Исходный код должен сопровождаться комментариями.

Если работа содержит пример выполнения задания, порядок выполнения должен соответствовать данному примеру. К отчётам некоторых лабораторных работ прилагаются отдельные требования. В этом случае они даны в разделе «Задания для выполнения к работе».

# Лабораторная работа №1

Разработка программ на ассемблере.

Работа с отладчиком OllyDbg и пакетом masm32

**Цель работы:** получить навыки создания и отладки простейших программ на ассемблере с использованием пакета masm32 и отладчика OllyDbg.

**Теоретические сведения**

Masm32 (Microsoft Assembler) – низкоуровневая среда разработки   
32-разрядных приложений на языке *Ассемблер* для операционной системы Windows. masm32 включает в себя средства для компиляции программ и большое количество библиотек для работы с различными объектами – строками, файлами, графикой, ресурсами операционной системы и т.д.   
С использованием masm32 возможно написание полноценных консольных и оконных приложений. При грамотном подходе, программы, написанные на ассемблере, могут работать эффективнее и быстрее, чем на языках программирования высокого уровня. С другой стороны, создавать на ассемблере большие программы очень сложно, т.к. требуется большая внимательность. Поэтому данный язык целесообразно использовать при написании оптимизированных подпрограмм, от которых требуется высокая скорость выполнения.

Программист, использующий ассемблер имеет в своём распоряжении следующие системные ресурсы:

1. 32-разрядные регистры центрального процессора;
2. Регистры сопроцессора для выполнения операций с вещественными числами;
3. 2 Гб оперативной памяти (предназначена для размещения команд, глобальных и локальных переменных, стека, динамического выделения памяти в куче).

Регистры процессора являются наиболее «быстрой» памятью, т.к. расположены непосредственно на самом кристалле процессора. Но их количество очень малó и поэтому регистры используется непосредственно в момент выполнения операций с данными, а пока они (данные) не нужны, они хранятся в оперативной памяти.

|  |  |  |  |
| --- | --- | --- | --- |
| EAX | | | Регистр аккумулятор. Размер – 32 бита. |
|  | AX | | Младшие 16 бит регистра EAX |
|  | AH | AL | Старший (AH) и младший (AL) байт регистра AX |
| EBX | | | Базовый регистр. Размер – 32 бита. |
|  | BX | | Младшие 16 бит регистра EBX |
|  | BH | BL | Старший (BH) и младший (BL) байт регистра BX |
| ECX | | | Регистр счётчик. Размер – 32 бита. |
|  | CX | | Младшие 16 бит регистра ECX |
|  | CH | CL | Старший (CH) и младший (CL) байт регистра CX |
| EDX | | | Размер – 32 бита. |
|  | DX | | Младшие 16 бит регистра EDX |
|  | DH | DL | Старший (DH) и младший (DL) байт регистра DX |
| ESI | | | Индексный регистр |
|  | SI | | Младшие 16 бит регистра ESI |
| EDI | | | Индексный регистр |
|  | DI | | Младшие 16 бит регистра EDI |
| ESP | | | Указатель на вершину стека |
| EBP | | | Указатель базы кадра стека |
| EIP | | | Указатель на текущую инструкцию |

Таблица 1. Регистры центрального процессора

**EAX**, **EBX**, **ECX**, **EDX** – регистры общего назначения. Их можно изменять и хранить в них свои данные. **EBP** (Base Pointer) можно использовать по собственному усмотрению, но в подпрограммах он обычно используется для обращения к аргументам подпрограмм в стеке. **ESI**, **EDI** – индексные регистры, которые обычно используются при работе с массивами, но их также можно использовать как регистры общего назначения.   
**EIP** (Instruction Pointer) – указатель на текущую выполняемую инструкцию. **ESP** (Stack Pointer) – регистр-указатель на вершину стека.

Программа на ассемблере состоит из следующих сегментов (областей памяти): сегмента данных, сегмента кода, сегмента стека. Ниже приведён пример простой программы на ассемблере, которая выполняет арифметические действия и выводит на экран строку:

.386

; Модель памяти и стиль вызова подпрограмм

.model flat, stdcall

; Чувствительность к регистру

option casemap: none

; --- Подключаемые заголовочные файлы и библиотеки

include d:\masm32\include\kernel32.inc

include d:\masm32\include\msvcrt.inc

includelib d:\masm32\lib\kernel32.lib

includelib d:\masm32\lib\msvcrt.lib

; --- Сегмент данных ---

.data

a db 5 ; Переменная размером 1 байт

b dw 10 ; Переменная размером 2 байта

d dd 77 ; Переменная размером 4 байта

array1 db 100 dup(1) ; Массив из 100 целых чисел размером один байт

str1 db "Моя первая программа на ассемблере", 0 ; Строка

; --- Сегмент кода ---

.code

start:

; Поместить в регистр EAX значение переменной d

MOV EAX, d ; EAX = d

MOV EBX, 34 ; EBX = 34

ADD EAX, EBX ; EAX = EAX + EBX | Сложение чисел d + 34

MOV ECX, 4 ; ECX = 4

MUL ECX ; Умножение: EDX:EAX = EAX\*ECX = (d + 34)\*4

; Вызов функции crt\_printf(str1) для вывода строки на экран

push offset str1 ; Поместить в стек адрес строки str1

call crt\_printf

; Вызов функции crt\_\_getch() для задержки ввода с клавиатуры

call crt \_\_getch

; Вызов функции ExitProcess(0)

push 0 ; Поместить аргумент функции в стек

call ExitProcess ; Выход из программы и возврат в ОС

end start

.386 – директива, которая указывает компилятору, что в программе будут использоваться команды процессора i80x386.

.model flat – директива, указывающая модель памяти. При разработке программ для Windows используется сплошная (flat) модель памяти.

stdcall – это стиль вызова подпрограмм. Подпрограммы в данном стиле принимают аргументы через стек в порядке справа налево, а вызываемая функция сама освобождает стек. В программах на ассемблере подпрограммы вызываются с помощью команды call. Аргументы при этом передаются в стек с помощью команды push. Выполнение двух последних команд «push 0» и «call ExitProcess» равнозначно вызову ExitProcess(0), например, на языке C++.

option casemap: none – включение чувствительности к регистру, т.е. идентификаторы «FIRE» и «fire» будут восприниматься компилятором по-разному. Всё, что записывается после точки с запятой «;» является комментарием.

Создадим сегмент данных, который включает строки, массивы и переменные:

.data

**a** db 1, 2, 3, 4, 5

**b** dw 1, 2, 3

d1 dd 5

f dd 5 dup (10)

str db "some string", 13, 10, 0

dq 1, 3, 5, 7

d2 dd ?

d3 dd 3.5

d4 dq 5.0

…

**a** – массив из 5 однобайтовых целых чисел;

**b** – массив из трёх двухбайтовых целых чисел;

**d1** – 4-байтовая целочисленная переменная;

d3 – вещественная переменная размером 4 байта (соответствует типу float на языке C++);

d4 – вещественная переменная размером 8 байт (тип double на C++);

**f** – массив из пяти 4-байтовых целых чисел. Каждый элемент массива имеет значение 10, т.е. записи;

**«f** dd 5 dup (10)» и

**«f** dd 10, 10, 10, 10, 10» эквивалентны;

str – строка, заканчивающаяся переходом на новую.

Следующая строка «**dq** 1, 3, 5, 7» просто заполняет память числами, при этом данная последовательность не ассоциируется с какой-либо переменной. Отметим, что целые числа хранятся в памяти в обратном порядке, т.е. младший байт числа храниться по младшему адресу. При этом адресом числа считается адрес его младшего байта. Такой способ хранения обеспечивает одинаковый способ обращения к числам, вне зависимости от их размера.

Строка «d2 dd ?» объявляет переменную d2, но не инициализирует её. Для объявления и задания размеров переменным и массивам в сегменте данных используются следующие директивы:

|  |  |  |  |
| --- | --- | --- | --- |
| Директива | Размер (байт) | | Соответствие типа на языке C++ |
| db | 1 | Байт (BYTE) | char, unsigned char |
| dw | 2 | Слово (WORD) | short, unsigned short |
| dd | 4 | Двойное слово (DOUBLE WORD) | int, unsigned int,  float |
| dq | 8 | Учетверённое слово (QUAD WORD) | long long,  double |
| df | 6 | 6 байт |  |
| dt | 10 | 10 байт (TEN BYTE) |  |

Данные, размещаемые в сегменте данных, являются глобальными данными. Они доступны в любом месте программы.

Компиляция программ, написанных на языке *Ассемблер*, осуществляется с помощью командной строки в 2 этапа. Для этого необходимы следующие исполняемые файлы, которые находятся в каталоге masm32\bin:

ml.exe – транслятор. Он преобразует исходный текст в объектный (объектного формата COFF или OMF),

link.exe – компоновщик. Собирает из объектных файлов готовый исполняемый (exe) файл или dll модуль.

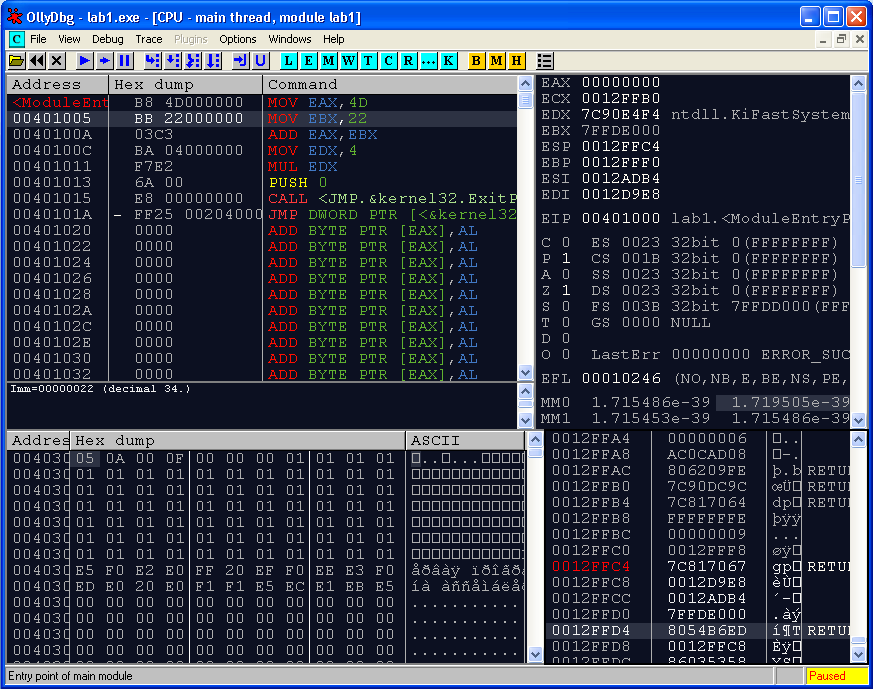
Для компиляции нужно создать каталог, например, с именем lab1 и скопировать в него файл с исходным кодом lab1.asm. Далее в командной строке (вызывается комбинацией клавиш WIN+R; открыть: cmd) нужно сделать текущим каталог lab1 и выполнить следующие команды:

D:\lab1**>**c:\masm32\bin\ml /c /coff lab1.asm

D:\lab1**>**c:\masm32\bin\link /SUBSYSTEM:CONSOLE /LIBPATH:c:\masm32\lib lab1.obj

Исходный код при этом необходимо предварительно сохранить в файле lab1.asm. Для того чтобы каждый раз не открывать командную строку для компиляции программ удобно создать командный файл с расширением bat и поместить в него вышеперечисленные команды. Для написания программ на ассемблере можно использовать простой блокнот с подсветкой синтаксиса, например, Notepad++.

После компиляции программы и получения файла lab1.exe его необходимо открыть в OllyDbg (F3).



OllyDbg – программа, предназначенная для анализа и отладки машинного кода. Окно отладчика состоит из 4 основных частей. В левом верхнем углу отображаются команды. В трёх колонках содержится информация об адресах команд (Address), их коды (Hex dump) в шестнадцатеричном представлении, и мнемоническое описание (Command). В окне, содержащем команды, можно с помощью EIP найти по адресу текущую команду.

Правая верхняя часть отображает содержимое регистров и флагов процессора. С (CF) – флаг переноса, P (PZ) – флаг чётности,   
A (AF) – вспомогательный флаг переноса, Z (ZF) – флаг нуля, S (SF) – флаг знака, T (TF) – флаг трассировки, D (DF) – флаг направления, O (OF) – флаг переполнения. Флаг представляет собой регистр, который может иметь состояние 0 или 1. Флаги отражают текущее состояние процессора и устанавливаются командами в зависимости от результата их выполнения.

Нижняя левая часть изначально содержит информацию о *глобальных данных*, а именно о глобальных переменных, массивах, строках и т.д. Но в общем, здесь можно увидеть содержимое участка памяти по любому интересующему адресу (комбинация клавиш Ctrl+G). Первая колонка (Address) – начальный адрес блока размером 16 байт. Следующие четыре колонки (Hex dump) содержат значения этой последовательности из 16 байтов. Вся информация выводится в шестнадцатеричном представлении. Последняя колонка (ASCII) содержит символьное представление последовательности байтов из предшествующих колонок.

В правой нижней части окна изображён стек. Стек предназначен для передачи аргументов в процедуры и функции, хранения адресов возврата из подпрограмм и для хранения локальных переменных и массивов. При вызове подпрограмм их аргументы помещаются в стек. Содержимое стека отображается с адреса, находящегося в ESP.

OllyDbg содержит следующие основные команды для отладки программ:

F7 – выполнить одну команду с заходом в подпрограмму;

F8 – выполнить одну команду с обходом подпрограммы;

F2 – создать точку останова;

Ctrl+F2 – перезагрузка программы (возврат в начало);

Alt+F2 – закрыть отлаживаемую программу.

**Задания для выполнения к работе**

1. Ознакомиться со средой OllyDbg и компилятором masm32.
2. Создать и скомпилировать программу в соответствии с вариантом задания. Добавить вызов функции ExitProcess(0) в конце для корректного завершения программы.
3. Отладить программу.
4. С помощью OllyDbg определить местонахождение переменных в сегменте данных, а также их размер. Оформить результат в виде таблицы.
5. Проверить на практике с использованием OllyDbg тот факт, что байты целых чисел хранятся в оперативной памяти в обратном порядке.
6. Выполнить пошаговую трассировку программы. Определить какие регистры и флаги изменяют свои значения в процессе выполнения команд. Включить в отчёт подробное описание действия каждой команды и снимки содержимого регистров и флагов.

**Пример выполнения работы**

1. Создать файл lab1.asm со следующим содержимым:

.386

.MODEL FLAT, STDCALL

OPTION CASEMAP: NONE

.DATA

**str1** DB "Hello, World!", 13, 10, 0

**a** DB 50

**arr** DW 10, 15, 20

**b** DD 10

**c1** DD 3500

**d** DQ 15, 16, 17

.CODE

START:

MOV AL, a

MOV ESI, 0

XOR EBX, EBX

MOV BX, arr[0]

ADD BX, arr[2]

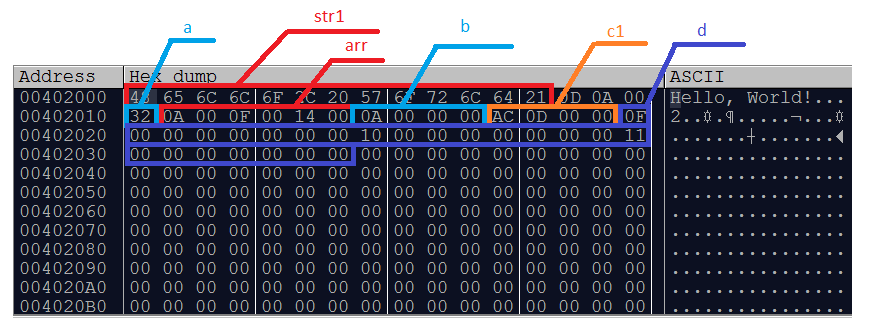
ADD BX, arr[4]

MOV EAX, b

MOV EDX, c1

END START

1. Скомпилировать программу и получить исполняемый файл lab1.exe.
2. Открыть файл lab1.exe в отладчике OllyDbg.
3. Сегмент данных содержит одну строку, заканчивающуюся переходом на новую, два массива и три переменные:



| Название переменной | Начальный адрес | Конечный адрес | Размер данных, байт | Тип |
| --- | --- | --- | --- | --- |
| **str1** | 00402000 | 0040200С | 13 | строка |
| **–** | 0040200D | 0040200F | 3 | символы окончания строки (0) и возврата каретки (13, 10) |
| **a** | 00402010 | 00402010 | 1 | однобайтовое целое |
| **arr** | 00402011 | 00402016 | 6 | массив из трёх двухбайтовых целых чисел |
| **b** | 00402017 | 0040202A | 4 | 4-байтовое целое |
| **c1** | 0040202B | 0040202E | 4 | 4-байтовое целое |
| **d** | 0040202F | 00402037 | 24 | массив из трёх 8-байтовых целых чисел |
| Общий размер сегмента данных: | | | **55** |  |

1. Пошаговое выполнение программы

|  |  |
| --- | --- |
| Исходное состояние регистров: | **5.1.** MOV AL, a  Код команды – **A0 10204000h**. Команда пересылает в регистр AL (младшую часть EAX) байт из ячейки памяти с адресом 00402010h (переменная **a**). Размер пересылаемых данных – байт. Увеличивает EIP на 5 (размер кода команды). |
|  | **5.2. MOV ESI, 0**  Код команды – BE 00000000h. Команда помещает в регистр ESI ноль и увеличивает EIP на 5. Размер помещаемых данных – двойное слово. |
|  | **5.3.** XOR EBX, EBX  Код команды – 33DB. Команда выполняет побитовое исключающее ИЛИ над регистром EBX, тем самым обнуляя его. Содержимое регистра EIP увеличивается на 2. |
|  | **5.4.** MOV BX, arr[0]  Код команды – 66:8B1D 11204000. Команда пересылает в регистр BX слово из памяти по адресу 00402011h. По данному адресу находится первый элемент массива arr – число 10. Размер пересылаемых данных – слово. EIP увеличивается на 7. |
|  | **5.5.** ADD BX, arr[2]  Код команды – 66:031D 13204000. Команда прибавляет к регистру BX число из памяти по адресу 00402013h. По данному адресу хранится второй элемент массива arr – число 15. Результат – BX равен 25 (19h).  EIP увеличивается на 7. Изменяются флаги PF, AF, ZF. |

**Варианты заданий**

1. .DATA

DB "MASM32", 0

DB 250, 251, 252, 254

a DW 500

b DW 2

cc DW 250

float1 DD 13.5

float2 DD 26.5

dmas DQ 5 DUP (10)

.CODE

START:

MOV AX, a

MOV BX, b

MOV CX, cc

DIV BX

ADD BX, CX

END START

1. .DATA

strt DB "Some String", 0

DW 400

a DF 900

mas DD 4\*8 DUP (3)

s DQ 15.7

.CODE

START:

MOV EAX, mas[0]

MOV EBX, DWORD PTR [a]

SUB EBX, EAX

MUL EBX

MOV ECX, offset strt

END START

1. .DATA

m DB 20 DUP (4)

n DB 20 DUP (8)

a DW 500

b DD ?

cc DQ 15.5

d DD 7.5

.CODE

START:

MOV AL, m[0]

MOV AH, n[0]

MOV m[0], AH

MOV n[0], AL

AND AL, n[2]

END START

1. .DATA

scanf\_s DB "%d", 0

printf\_s DB "Result: %d", 13, 10, 0

v DD 4\*4 DUP(5)

cc DW 2

double DQ 2.5

.CODE

START:

XOR EAX, EAX

MOV ESI, 0

MOV EAX, v[ESI]

MUL cc

ADD ESI

MOV v[ESI], EAX

END START

1. .DATA

s DB "Length: ", 0

len DW 0

mas DD 8 DUP(1)

ten DT 300000000

.CODE

START:

MOV ESI, 0

MOV EAX, mas[ESI]

INC len

INC ESI

SUB EAX, mas[ESI+8]

END START

1. .DATA

s DB "some string", 0

d DD 255

sm DW 500

result DD ?

p DQ 17.5

ten DT 180000.5

.CODE

START:

MOV EAX, d

MUL sm

MOV result, EAX

ADD d, EDX

LEA ECX, sEND START

END START

1. .DATA

Sstr DB 13, 10, "Strings", 0

p DB 5

k DW 500

s DD 5000

ten DT 88.38

.CODE

START:

XOR EAX, EAX

MOV AX, k

MOV BX, WORD PTR [p]

MUL BX

MOV k, AX

END START

1. .DATA

a DB 5

b DD a, 6

DF 15

DD 13.5, 15.6, 18.9

DQ 17890.65432

.CODE

START:

XOR EAX, EAX

MOV EAX, b[0]

ADD EAX, DWORD PTR b[8]

MOV EBX, 4

MUL EBX

END START

1. .DATA

as DB 10, 13, "a-s", 0

h DW 20

w DW 100

s DD ?

DT 17834.55

.CODE

START:

XOR EAX, EAX

MOV AX, h

MUL w

MOV s, EAX

SUB s, 20

END START

1. .DATA

result DB "Result: ", 0

a DD ?

v1 DW 8

v2 DW 16

t DW 4

dop DF 5 DUP(16536)

.CODE

START:

XOR EAX, EAX

MOV AX, v2

SUB AX, v1

CWD

DIV t

MOV a, EAX

END START

1. .DATA

strs DB 13, 10, "Summ:", 0

sum DD ?

mas DW 4 DUP(5)

longFloat DQ 154.5

.CODE

START:

MOV EAX, sum

MOV EBX, 4

MOV ESI, 0

ADD AX, mas[ESI]

ADD ESI, 2

MUL EBX

MOV sum, EAX

END START

1. .DATA

s\_per DB "perimetr", 0

per DD ?

a DW 400

b DW 5

c1 DW 15

DF 15789

longFloat DQ 154.142423

.CODE

START:

XOR EAX, EAX

ADD AX, a

SUB AX, b

DIV c1

MOV per, EAX

END START

1. .DATA

scanf\_s DB "%d", 0

d DW 500

mas DD 8 DUP(1)

ten DT 30000000

.CODE

START:

XOR EAX, EAX

MOV AX, d

MUL mas[0]

MOV EBX, mas[0]

MOV d, AX

END START

1. .DATA

s1 DB 13, 10, "\_\_\_\_", 0

d DW 500

result DD ?

p DQ 17.5

ten DT 183333.5

.CODE

START:

XOR ECX, ECX

MOV CX, d

ADD CX, 5

MOV AX, 6

IDIV AX

MOV result, EAX

END START

1. .DATA

name DB 10, 13, "ассемблер и ollydbg", 0

a DW ?

b DD ?

c1 DB 16

DF 15, 150

.CODE

START:

MOV AL, c1

XOR EBX, EBX

MOV BL, AL

ADD a, BX

MOV BH, AL

DEC b

END START

1. .DATA

s DB 13, 10, "string", 0

d DW 555

subs DB "str", 0

DD 15, 5

a DQ ?

.CODE

START:

XOR ECX, ECX

MOV AX, d

MOV CX, 20

CWDE

DIV CX

MOV d, AX

END START

1. .DATA

t1 DW 14, 15

t2 DW 2 DUP(2)

k DD ?

b DD ?

ddd DT 155000

.CODE

START:

XOR EAX, EAX

MOV AX, t1[0]

MOV BX, t2[0]

SUB AX, BX

MOV k, EAX

MOV b, EBX

END START

1. .DATA

String DB 13, 10, "RESULT: ", 0

Mas DW 15, 16, 17, 18, 19, 20

a DD ?

b DD ?

c1 DD ?

DQ 1500.0

.CODE

START:

XOR EAX, EAX

MOV AX, Mas[0]

MOV a, EAX

ADD b, 4

MOV AX, Mas[10]

INC c1

END START

1. .DATA

str1 DB "Some\_S", 0

DD 4\*15 DUP (60)

mas DD 20 DUP(20)

dff DF ?

DQ 15.5

.CODE

START:

XOR ECX, ECX

MOV ECX, mas[16]

ADD CX, 5

MOV AL, CL

MOV mas[12], EAX

END START

1. .DATA

new\_line DB 13, 10, 0

a DW 5

b DW 5

result DD ?

p DF 17.5

ten DT 15.5

.CODE

START:

XOR ECX, ECX

MOV CX, b

XOR EAX, EAX

MOV AX, a

IMUL CX, 5

MOV result, EAX

END START

1. .DATA

hello DB "Hello", 0

mas DW 232, 443, 567, 197

bigMas DD 4 DUP(?)

p DQ 156.43

ten DT ?

.CODE

START:

XOR EAX, EAX

MOV ESI, 0

MOV AX, mas[ESI]

MOV bigMas[ESI], EAX

ADD ESI, 2

MOV AX, mas[ESI]  
ADD ESI, 2

MOV bigMas[ESI], EAX

END START

1. .DATA

printf DB 13, 10, "Result: %d", 0

a DW 5

b DD 60

r DD ?

dqq DQ 171.233

.CODE

START:

XOR EAX, EAX

MOV AX, a

MOV EBX, b

ADD EAX, EBX

MOV r, EAX

END START

1. .DATA

strt DB "Some String", 0

v DD 4\*4 DUP(5)

DW 387

r DQ ?

ten DT 183333.423

.CODE

START:

XOR EAX, EAX

MOV EAX, v[0]

MOV EBX, v[4]

ADD EAX, EBX

INC EAX

END START

1. .DATA

scanf\_s DB "%d", 0

arr DW 4\*5 DUP(?)

con DD 5

DD ?

pi DF 3.14

.CODE

START:

XOR EAX, EAX

MOV ESI, 0

MOV EBX, con

ADD EAX, EBX

MOV arr[ESI], AX

ADD ESI, 2

END START

1. .DATA

scanf\_s DB "%d", 0

printf\_s DB 13, 10, "Result: %d", 0

arr1 DW 4\*5 DUP(?)

arr2 DW 5\*4 DUP(2)

DD ?

e DT 2.75

.CODE

START:

XOR EAX, EAX

MOV AX, arr2[0]

ADD AX, 15

MOV arr1[0], AX

IMUL AX, 5

END START

Контрольные вопросы

1. Из каких частей состоит программа на ассемблере?
2. Каково назначение стека?
3. Для чего предназначена программа OllyDbg?
4. Какие директивы используются для задания размеров и объявления массивов и переменных на ассемблере?
5. Какая информация хранится в регистрах EIP и ESP?
6. Какая информация отображается в окнах отладчика OllyDbg?
7. Что такое регистры и флаги центрального процессора?
8. Какие регистры процессора можно использовать как регистры общего назначения?
9. Как объявить массив длиной 100 вещественных чисел типа float, состоящий из одних единиц? Какой будет его размер в байтах?
10. Из каких этапов состоит компиляция программ на ассемблере?
11. Как соответствуют директивы объявления переменных типам языка C++?
12. Как хранятся в памяти целые числа?
13. Как вызываются подпрограммы на ассемблере?
14. Где хранятся локальные и глобальные данные программы?