Лабораторная работа №1

Разработка программ на ассемблере.

Работа с отладчиками и пакетом masm32

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

**Теоретические сведения**

Masm32 (Microsoft Assembler) – низкоуровневая среда разработки   
32-разрядных приложений на языке ассемблер для операционной системы Windows. Masm32 включает в себя большое количество библиотек, в том числе для работы с консолью и оконными приложениями.

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 переменных, стека, динамического выделения памяти).

Регистры процессора являются наиболее «быстрой» памятью, т.к. расположены непосредственно на самом кристалле процессора. Но их количество очень мало и поэтому регистры используется в основном только при выполнении 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 | | | Также используется для работы со стеком |

Таблица 1. Описание регистров центрального процессора

EAX, EBX, ECX, EDX – регистры общего назначения. ESP (Stack Pointer) – регистр указатель на вершину стека. EBP (Base Pointer) можно использовать по собственному усмотрению, но обычно он используется для работы со стеком. ESI, EDI – индексные регистры, которые используются при работе с массивами. EIP (Instruction Pointer) – указатель на текущую выполняемую инструкцию.

Программа на ассемблере состоит из следующих сегментов (областей памяти): сегмента данных, сегмента кода, сегмента стека. Ниже приведён пример простой программы на ассемблере:

.386 ; Тип процессора

.model flat, stdcall ; Модель памяти и стиль вызова подпрограмм

option casemap: none ; Чувствительность к регистру

; --- Подключение файлов с кодом, макросами, константами, прототипами функций и т.д.

include d:\masm32\include\windows.inc

include d:\masm32\include\kernel32.inc

include d:\masm32\include\user32.inc

; --- Подключаемые библиотеки ---

includelib d:\masm32\lib\user32.lib

includelib d:\masm32\lib\kernel32.lib

; --- Сегмент данных ---

.data

; --- Сегмент кода ---

.code

start:

MOV EAX, 77 ; Поместить в регистр EAX число 77

MOV EBX, 34

ADD EAX, EBX ; Сложение чисел 77 + 34

MOV EDX, 4

MUL EDX ; Умножение EDX = (77 + 34)\*4

push NULL

call ExitProcess ; Выход из программы

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. Аргументы для данного стиля передаются 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 NULL» и «call ExitProcess» равнозначно вызову ExitProcess(0), например, на языке C++.

Чувствительность к регистру в данном случае присутствует, т.е. «FIRE» и «fire» будут восприниматься компилятором по-разному.

В приведённом выше примере сегмент данных ничего не содержит. 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 ?

.code

…

**a** – массив из 5 однобайтовых чисел

**b** – массив из трёх двухбайтовых чисел

**d1** – 4-байтовая переменная

**f** – массив из пяти 4-байтовых 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.   
Строка «**d2** dd ?» объявляет переменную **d2**, но не инициализирует её.

Для задания размеров переменным в сегменте данных используются следующие директивы:

|  |  |  |
| --- | --- | --- |
| Директива | Размер | |
| db | 1 | Байт (BYTE) |
| dw | 2 | Слово (WORD) |
| dd | 4 | Двойное слово (DOUBLE WORD) |
| dq | 8 | Учетверённое слово (QUAD WORD) |
| df | 6 | 6 байт |
| dt | 10 | 10 байт (TEN BYTE) |

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 obj-файл (объектного формата COFF или OMF),

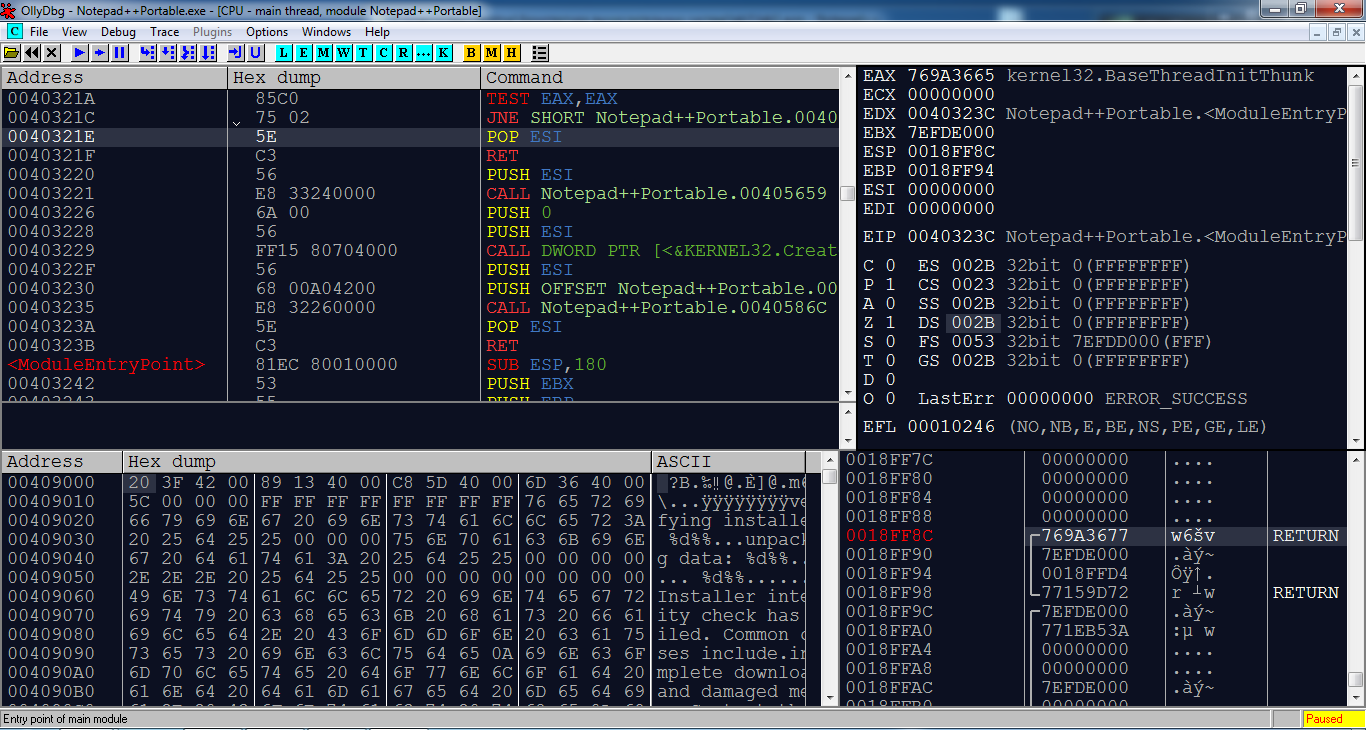
**link.exe** – линковщик. Создаёт готовый исполняемый exe или dll модуль.

c:\masm32\bin\ml /c /coff lab1.asm

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

Исходный код при этом необходимо предварительно сохранить в файле lab1.asm. 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). Вместо OllyDbg можно использовать отладчик IDA или x64dbg.



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.

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 её аргументы помещаются в стек. Содержимое стека отображается с адреса, находящегося в ESP.

С, P, A, Z, S, T, D, O – флаги переноса, чётности, нуля, переполнения и т.д. Флаг представляет собой регистр, который может иметь состояние 0 или 1. Флаги устанавливаются командами в зависимости от результата их выполнения.

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).

**Задания для выполнения к работе**

1. Ознакомиться со средой OllyDbg, x64dbg, IDA и компилятором masm32.
2. Создать и скомпилировать программу в соответствии с вариантом задания.
3. Отладить программу.
4. С помощью OllyDbg (x64dbg, IDA) определить местонахождение переменных в сегменте данных, а также их размер.
5. Выполнить пошаговую трассировку программы в трёх отладчиках (OllyDbg, x64dbg, IDA). Определить какие регистры изменяют свои значения в процессе выполнения команд.
6. Декомпилировать программу в IDA HexRay. Включить в отчёт результаты декомпиляции.
7. Сделать выводы о проделанной работе.

**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]

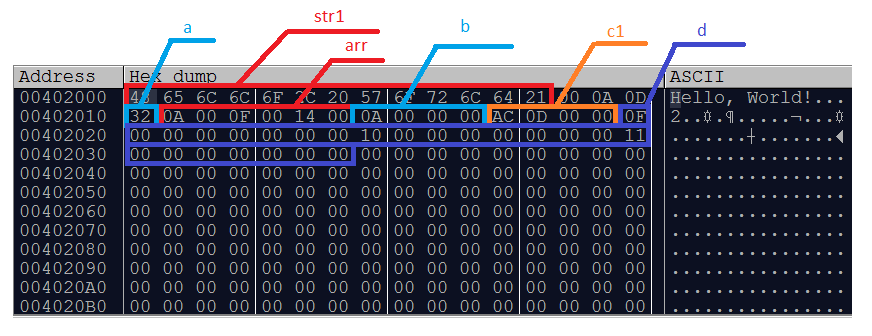
ADD BX, arr[4]

MOV EAX, b

MOV EDX, c1

END START

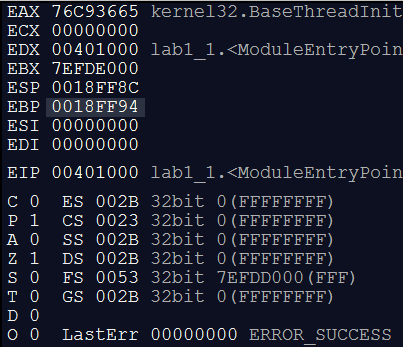
1. Скомпилировать программу и получить исполняемый файл lab1.exe.
2. Открыть файл lab1.exe в OllyDbg.
3. Сегмент данных содержит одну строку, два массива и три переменные:



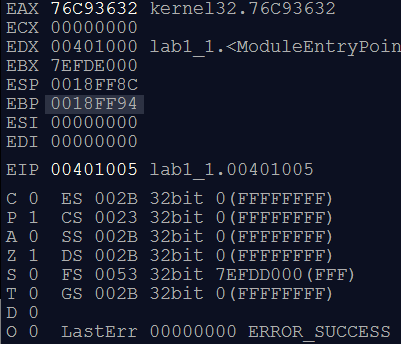
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Название переменной | Начальный адрес | Конечный адрес | Размер данных, байт | Тип |
| **str1** | 00402000 | 0040200С | 13 | строка |
| **–** | 0040200D | 0040200F | 3 | символы окончания строки (0) и возврата каретки (10, 13) |
| **a** | 00402010 | 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

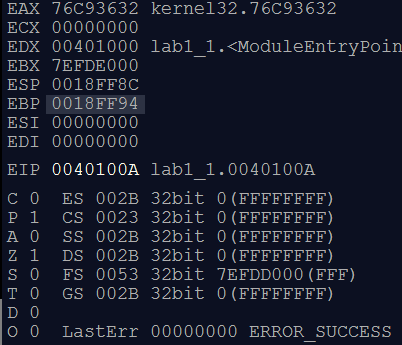
Исходное состояние регистров:



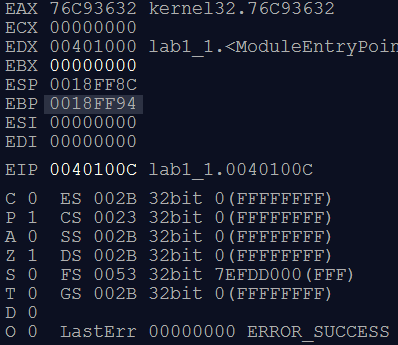
* 1. MOV AL, a

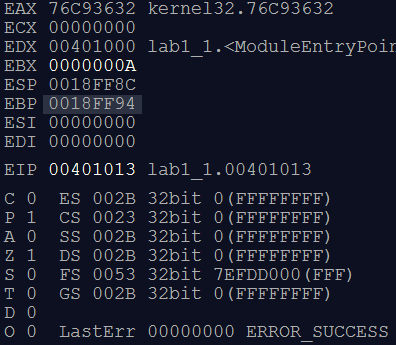
Код команды – **A0 10204000h**. Команда пересылает в регистр **AL** (младшую часть **EAX**) байт из ячейки памяти с адресом 00402010h (переменная **a**). Размер пересылаемых данных – байт. Увеличивает **EIP** на 5 (размер кода команды).

* 1. MOV ESI, 0

Код команды – **BE 00000000h**. Команда помещает в регистр **ESI** ноль и увеличивает **EIP** на 5. Размер помещаемых данных – двойное слово.

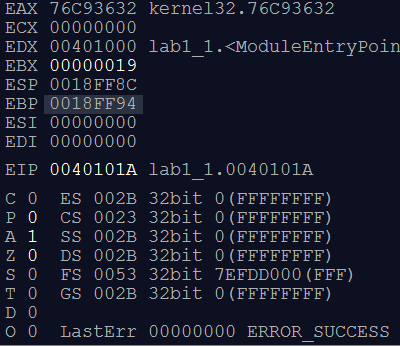
* 1. XOR EBX, EBX

Код команды – **33DB**. Команда выполняет побитовое исключающее **ИЛИ** над регистром **EBX**, тем самым обнуляя его. Содержимое регистра **EIP** увеличивается на 2.

* 1. 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.

* 1. 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 P, A, Z.

**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 (5)

.CODE

START:

MOV AX, a

MOV BX, b

MOV CX, cc

DIV BX

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

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

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

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

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

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]

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

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

11. .DATA

strs DB 10, 13, "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

12. .DATA

sper DB "perimeter", 0

per DD ?

a DW 4

b DW 5

c DW 15

DF 15789

longFloat DQ 154.142423

.CODE

START:

XOR EAX, EAX

ADD EAX, a

ADD EAX, b

ADD EAX, c

MOV per, EAX

END START

13. .DATA

scanf\_s DB "%d", 0

das DW 500

mas DD 8 DUP(1)

ten DT 30000000

.CODE

START:

XOR EAX, EAX

MOV AX, das

MUL 5

MOV EBX mas[0]

MOV das, AX

END START

14. .DATA

String1 DB 13, 10, "\_\_\_\_", 0

ds DW 5

result DD ?

p DQ 17.5

ten DT 183333.5

.CODE

START:

XOR ECX, ECX

MOV CX, ds

ADD CX, 5

MOV EAX, ECX

DIV 2

MOV result, EAX

END START

15. .DATA

name DB 13, 10, "Andrey", 0

a DW ?

b DD ?

c1 DB 16

DF 15, 150

.CODE

START:

MOV AL, c1

DIV 4

XOR EBX, EBX

MOV BL, AL

MOV a, BX

MOV BH, AL

MOV b, EBX

END START

16. .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 CX, d

MOV EAX, ECX

DIV 5

MUL 3

MOV d, AX

END START

17. .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

18. .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

MOV b, Mas[4]

MOV AX, Mas[10]

MOV c1, EAX

END START

19. .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

20. .DATA

enter 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 EAX, a

MUL CX

MOV result, EAX

END START

21. .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

22. .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

23. .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

END START

24. .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

25. .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

END START