

**Федеральное государственное бюджетное образовательное
учреждение высшего образования "Белгородский государственный
технологический университет им. В.Г. Шухова"**

Кафедра программного обеспечения вычислительной техники и
автоматизированных систем.

Лабораторная работа № 1

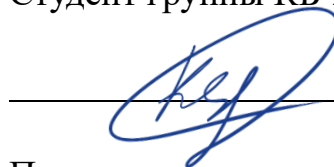
Разработка программ на ассемблере.

Работа с отладчиком x32dbg, пакетом masm32.

Вариант 13

Выполнил:

Студент группы КБ-211



Коренев Д.Н.

Принял:



Осипов О.В.

Цель работы: получить навыки создания простейших ассемблерных программ с использованием пакета `masm32` и научиться пользоваться отладчиком `x32dbg`.

Задание

1. Ознакомиться со средой `x32dbg` и компилятором `masm32`.
2. Создать и скомпилировать программу в соответствии с вариантом задания. В программу включить комментарии с описанием, что делает каждая инструкция. Подробное описание каждой команды можно найти в приложении учебника В.И. Юрова «Assembler», начиная со стр. 511. Комментарии следует выравнивать по левому краю (как в примере).

Выполнение задания

```
.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 windows.inc
include kernel32.inc
include user32.inc
include msvcrt.inc

; --- Подключаемые библиотеки ---
includelib user32.lib
includelib kernel32.lib
includelib msvcrt.lib

; --- Сегмент данных ---
.data
    strd DB "Division", 0
    a DD 10500000h, 1200h
    b DD 10000
    m DD ?
    mas DW 8 DUP(1)
    ten DT 30000, -30000
    f DQ 1, 1.0, -1.0
    h DF -3, -2, -1, 0, 1, 2, 3, 4, 40000h, 40000

; --- Сегмент кода ---
.code
start:
    MOV EDX, a[4] ; EDX = a[4] = 0x1200
    MOV EAX, a[0] ; EAX = a[0] = 0x10500000
    DIV b ; EAX = EAX / b
    MOV m, EDX ; m = 0x120010500000 mod b
```

```

ADD EAX, m      ; EAX = EAX + m
IMUL EAX, 3     ; EAX = EAX * 3

push NULL
call ExitProcess ; Выход из программы
end start

```

3. С помощью отладчика определить местонахождение переменных, строк и массивов в сегменте данных, а также их размер. Составить таблицу и подробное описание ячеек сегмента данных (как в примере).

Выполнение задания

Адрес	Шестнадцатеричное	ASCII
00403000	44 69 76 69 73 69 6F 6E 00 00 00 50 10 00 12 00	Division...P...
00403010	00 10 27 00 00 00 00 00 00 01 00 01 00 01 01Ou.....
00403020	00 01 00 01 00 01 00 01 00 30 75 00 00 00 00 00D.uyuyuyuy.....
00403030	00 00 00 D0 8A FF FF FF FF FF FF FF FF 01 00 00d?uyuyuyuy.....
00403040	00 00 00 00 00 00 00 00 00 00 00 F0 3F 00 00 00uyuyuyuyuyuy.....
00403050	00 00 00 F0 BF FD FF FF FF FF FF FE FF FF FF FFuyuyuyuyuyuy.....
00403060	FF FF FF FF FF FF FF 00 00 00 00 00 00 01 00 00	uyuyuyuy.....
00403070	00 00 00 02 00 00 00 00 00 03 00 00 00 00 00 04@.....
00403080	00 00 00 00 00 00 00 04 00 00 00 40 9C 00 00 00@.....
00403090	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

Название переменной	Начальный адрес	Конечный адрес	Размер данных, байт	Описание
strd	00403000	00403007	8	строка «Division»
-	00403008	00403008	1	символ окончания строки (0)
a	00403009	00403010	8	два четырехбайтовых целых числа 10500000 ₁₆ и 1200 ₁₆
b	00403011	00403014	4	одно четырехбайтовое число 10000 ₁₀
m	00403015	00403018	4	неинициализированная 4-байтовая переменная
mas	00403019	00403028	16	массив из 8 двухбайтовых чисел 1
ten	00403029	0040303C	20	два 10-байтовых числа 30000 ₁₀ и -30000 ₁₀
f	0040303D	00403054	24	три 8-байтовых числа 1, 1.0, -1.0
h	00403055	00403090	60	10 6-байтовых чисел -3, -2, -1, 0, 1, 2, 3, 4, 40000 ₁₆ , 40000
Общий размер сегмента данных:			145	

Ячейки памяти с адресами от 0x00403000 до 0x00403007 содержат ASCII-коды символов строки «Division».

Строка заканчивается ноль-символом, расположенным по адресу 0x00403008.

Массив a начинается с адреса 0x00403009 и содержит два числа: 10500000₁₆, и 1200₁₆.

Переменная *b* начинается с адреса 0x00403011 и содержит 4-байтовое число 10000 (2710 в дополнительном коде).

По адресу 0x00403015 располагается неинициализированная 4-байтовая переменная *m*, но в сегменте данных данные ячейки заполнены нулями.

С адреса 0x00403019 начинается массив из 8 2-байтовых чисел 1.

Массив *ten* начинается с адреса 0x00403029 и содержит два 10-байтовых числа $30000_{10} = 7530_{16}$ и $-30000_{10} = -7530_{16}$ (FFFFFFFF8AD0 в дополнительном коде).

Массив *f* начинается с адреса 0x0040303D и содержит три 8-байтовых числа 1, 1.0, -1.0 (дробные числа хранятся в памяти в формате FP*, в данном случае FP64 (знак, экспонента и дробная часть).

Массив *h* начинается с адреса 0x00403055 и содержит 10 6-байтовых чисел $-3 = -03_{16} = \text{FFFFFFFFFD}_{16}$, $-2 = -02_{16} = \text{FFFFFFFE}_{16}$, $-1 = -01_{16} = \text{FFFFFFF}_{16}$, 0, 1, 2, 3, 4, 40000_{16} , $40000 = \text{0FA0}_{16}$ (отрицательные числа хранятся в дополнительном коде).

4. Выполнить пошаговую трассировку программы. Определить какие регистры, флаги и ячейки памяти изменяют свои значения в процессе выполнения команд. Обеспечить корректное завершение программы вызовом системной функции *ExitProcess* с кодом завершения 0. Если в сегменте данных есть строки, то вывести её в консоль. Трассировку требуется выполнить до команды «*call ExitProcess*» включительно. Составить для каждой инструкции таблицу трассировки (как в примере).

Выполнение задания

00401000	8B15 0D304000	mov edx,dword ptr ds:[40300D]
00401006	A1 09304000	mov eax,dword ptr ds:[403009]
0040100B	F735 11304000	div dword ptr ds:[403011]
00401011	8915 15304000	mov dword ptr ds:[403015],edx
00401017	0305 15304000	add eax,dword ptr ds:[403015]
0040101D	68C0 03	imul eax,eax,3
00401020	6A 00	push 0
00401022	E8 01000000	call <JMP.&ExitProcess>
00401027	CC	int3
00401028	FF25 00204000	jmp dword ptr ds:[<&ExitProcess>]
0040102E	0000	add byte ptr ds:[eax],al

EAX=	0019FFCC	EBX=	00217000	ECX=	00401000	EDX=	00401000
ESP=	0019FF78	EBP=	00401000	ESI=	00401000	EDI=	00401000
EIP=	00401000						
ZF=	1	PF=	1	AF=	0		
OF=	0	SF=	0	DF=	0		
CF=	0	TF=	0	IF=	1		

mov edx, dword ptr ds:[0x0040300D]	КОП:	8B15 0D304000
------------------------------------	------	---------------

EAX=	0019FFCC		EBX=	00217000		ECX=	00401000		EDX=	00001200
ESP=	0019FF78		EBP=	00401000		ESI=	00401000		EDI=	00401000
EIP=	00401006									
ZF=	1	PF=	1	AF=	0					
OF=	0	SF=	0	DF=	0					
CF=	0	TF=	0	IF=	1					
Пересылает из ячейки памяти с адресом 0x0040300D в регистр EDX 4 байта. Увеличивает значение в регистре EIP на 6 (размер кода 8B15 0D304000).										

mov eax, dword ptr ds:[0x00403009]						КОП:		A1 09304000							
EAX=		10500000		EBX=		00217000		ECX=		00401000		EDX=		00001200	
ESP=		0019FF78		EBP=		00401000		ESI=		00401000		EDI=		00401000	
EIP=		0040100B													
ZF=		1		PF=		1		AF=		0					
OF=		0		SF=		0		DF=		0					
CF=		0		TF=		0		IF=		1					
Пересылает из ячейки памяти с адресом 0x00403009 в регистр EAX 4 байта. Увеличивает значение в регистре EIP на 5 (размер кода A1 09304000).															

div dword ptr ds:[0x00403011]

КОП: F735 11304000

EAX=	75F76809	EBX=	00217000	ECX=	00401000	EDX=	00002070
ESP=	0019FF78	EBP=	00401000	ESI=	00401000	EDI=	00401000
EIP=	00401011						
ZF=	1	PF=	1	AF=	0		
OF=	0	SF=	0	DF=	0		
CF=	0	TF=	0	IF=	1		

Выполняет целочисленное деление числа в регистре EAX на число в ячейке памяти с адресом 0x00403011, результат помещается в регистр EAX, остаток от деления в регистр EDX. Увеличивает значение в регистре EIP на 6.

mov dword ptr ds:[0x00403015], edx						КОП:		8915 15304000			
EAX=	75F76809		EBX=	00217000		ECX=	00401000		EDX=	00002070	
ESP=	0019FF78		EBP=	00401000		ESI=	00401000		EDI=	00401000	
EIP=	00401017										
ZF=	1	PF=	1	AF=	0						
OF=	0	SF=	0	DF=	0						
CF=	0	TF=	0	IF=	1						
Пересылает 4 байта из регистра EAX в ячейку памяти с адресом 0x00403015. Увеличивает значение в регистре EIP на 6.											

add eax, dword ptr ds:[0x00403015]				КОП:	0305 15304000		
EAX=	75F78879	EBX=	00217000	ECX=	00401000	EDX=	00002070

ESP=	0019FF78	EBP=	00401000	ESI=	00401000	EDI=	00401000
EIP=	0040101D						
ZF=	0	PF=	0	AF=	0		
OF=	0	SF=	0	DF=	0		
CF=	0	TF=	0	IF=	1		
Суммирует значение в регистре EAX и число в ячейке памяти с адресом 0x00403015. Увеличивает значение в регистре EIP на 6. Сбрасывает флаги ZF, PF.							

imul eax, eax, 0x3				КОП:		6BC0 03	
EAX=	61E6996B	EBX=	00217000	ECX=	00401000	EDX=	00002070
ESP=	0019FF78	EBP=	00401000	ESI=	00401000	EDI=	00401000
EIP=	00401020						
ZF=	0	PF=	0	AF=	0		
OF=	1	SF=	0	DF=	0		
CF=	1	TF=	0	IF=	1		
Умножает значение регистра EAX на число 0x3. Записывает результат в регистр EAX, устанавливает флаги OF, CF. Увеличивает значение в регистре EIP на 3.							

push 0x0					КОП:	6A 00		
EAX=	61E6996B	EBX=	00217000	ECX=	00401000	EDX=	00002070	
ESP=	0019FF74	EBP=	00401000	ESI=	00401000	EDI=	00401000	
EIP=	00401022							
ZF=	0	PF=	0	AF=	0			
OF=	1	SF=	0	DF=	0			
CF=	1	TF=	0	IF=	1			
Помещает число 0 на стек. Уменьшает указатель на вершину стека в регистре ESP на 4. Увеличивает значение в регистре EIP на 2.								

call 0x00401028				КОП:		E8 01000000	
EAX=	61E6996B	EBX=	00217000	ECX=	00401000	EDX=	00002070
ESP=	0019FF70	EBP=	00401000	ESI=	00401000	EDI=	00401000
EIP=	00401028						
ZF=	0	PF=	0	AF=	0		
OF=	1	SF=	0	DF=	0		
CF=	1	TF=	0	IF=	1		
Вызывает ExitProcess(). Уменьшает указатель на вершину стека в регистре ESP на 4. Увеличивает значение в регистре EIP на 6.							

5. Добавить в программу вывод названий переменных, адреса их начала и конца, их размер.

Ошибка			
var	addr	end	size
strd	00403000	00403008	9
a	00403009	00403010	8
b	00403011	00403014	4
m	00403015	00403018	4
mas	00403019	00403028	16
ten	00403029	0040303C	20
f	0040303D	00403054	24
h	00403055	00403090	60

OK

```
.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 windows.inc
include kernel32.inc
include user32.inc
include msvcrt.inc

; --- Подключаемые библиотеки ---
includelib user32.lib
includelib kernel32.lib
includelib msvcrt.lib

; --- Сегмент данных ---
.data
    strd DB "Division", 0
    a DD 10500000h, 1200h
    b DD 10000
    m DD ?
    mas DW 8 DUP(1)
    ten DT 30000, -30000
    f DQ 1, 1.0, -1.0
    h DF -3, -2, -1, 0, 1, 2, 3, 4, 40000h, 40000

    format DB 8 DUP ("%s", 9, "%p", 9, "%p", 9, "%d", 13, 10), 0
    header DB "var", 9, "addr", 9, 9, "end", 9, 9, "size", 13, 10,
        "-----",
        13, 10, 0

    arg1_name DB "strd", 0
    arg2_name DB "a", 0
    arg3_name DB "b", 0
    arg4_name DB "m", 0
```

```

arg5_name DB "mas", 0
arg6_name DB "ten", 0
arg7_name DB "f", 0
arg8_name DB "h", 0

buffer DB 1000 DUP (?)

; --- Сегмент кода ---
.code
start:
    push offset header
    push offset buffer
    call crt_sprintf
    add esp, 3*4

    push (offset format) - offset h
    push ((offset format) - 1)
    push offset h
    push offset arg8_name

    push (offset h) - offset f
    push ((offset h) - 1)
    push offset f
    push offset arg7_name

    push (offset f) - offset ten
    push (offset f - 1)
    push offset ten
    push offset arg6_name

    push (offset ten) - offset mas
    push ((offset ten) - 1)
    push offset mas
    push offset arg5_name

    push (offset mas) - offset m
    push ((offset mas) - 1)
    push offset m
    push offset arg4_name

    push (offset m) - offset b
    push ((offset m) - 1)
    push offset b
    push offset arg3_name

    push (offset b) - offset a
    push ((offset b) - 1)
    push offset a
    push offset arg2_name

```



```

push (offset a) - offset strd
push ((offset a) - 1)
push offset strd
push offset arg1_name

push offset format
push offset buffer[92]
call crt_sprintf
add esp, 31*4

push NULL
push NULL
push offset buffer
push NULL
call MessageBoxA
add esp, 5*4

push NULL
call ExitProcess ; Выход из программы
end start

```

6. Сделать выводы о проделанной работе.

Вывод: В ходе лабораторной работы мы получили навыки создания простейших ассемблерных программ с использованием пакета `masm32` и научились пользоваться отладчиком `x32dbg`. Создали и скомпилировали программу в соответствии с вариантом задания. С помощью отладчика определили местонахождение переменных, строк и массивов в сегменте данных, а также их размер. Составили таблицу и подробное описание ячеек сегмента данных. Выполнили пошаговую трассировку программы, определили какие регистры, флаги и ячейки памяти изменяют свои значения в процессе выполнения команд.