Министерство образования Республики Беларусь
Учреждение образования «Витебский государственный
университет имени П.М. Машерова»
Кафедра прикладной математики и механики

С.А. Ермоченко, С.В. Сергеенко

ЯЗЫК АССЕМБЛЕРА ДЛЯ INTEL-СОВМЕСТИМЫХ ПРОЦЕССОРОВ С 32-БИТНОЙ АРХИТЕКТУРОЙ

Методические рекомендации к выполнению лабораторных работ

Витебск
ВГУ имени П.М. Машерова
2015

УДК 004.431.4(075.8)
ББК 32.973.2я73
Е74

Печатается по решению научно-методического совета учреждения образования «Витебский государственный университет имени П.М. Машерова». Протокол № 1 от 23.10.2015 г.

Авторы: заведующий кафедрой прикладной математики и механики ВГУ имени П.М. Машерова, кандидат физико-математических наук **С.А. Ермоченко;** старший преподаватель кафедры прикладной математики и механики ВГУ имени П.М. Машерова **С.В. Сергеенко**

Р е ц е н з е н т :
доцент кафедры автоматизации технологических процессов и производства УО «ВГТУ», кандидат технических наук, доцент *В.Е. Казаков*

Ермоченко, С.А.

Е74 Язык Ассемблера для Intel-совместимых процессоров с 32-битной архитектурой : методические рекомендации к выполнению лабораторных работ / С.А. Ермоченко, С.В. Сергеенко. – Витебск : ВГУ имени П.М. Машерова, 2015. – 39 с.

В методических рекомендациях изложен ход выполнения лабораторных работ по различным темам, связанным с изучением 32-битной архитектуры процессоров Intel и совместимых с ними. Практически по всем темам приводятся различные примеры, демонстрирующие те или иные особенности работы процессора или языка Ассемблера для него. Кроме того, дается описание полезных при выполнении лабораторных работ функций операционной системы Windows.

Предназначается для студентов специальностей «Прикладная информатика» (дисциплины «Архитектура компьютеров»), «Программное обеспечение информационных технологий» (дисциплины «Языки программирования»).

УДК 004.431.4(075.8)
ББК 32.973.2я73

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В данных методических рекомендациях приведены задания на лабораторные работы по языку Ассемблера для Intel-процессоров 32-битной архитектуры и совместимых с ними. По каждой лабораторной работе приводятся также краткие теоретические сведения как по самому языку Ассемблера, так и по функциям операционной системы Windows, которые могут быть полезны студентами при выполнении лабораторной работы. Теоретические сведения иллюстрируются примерами программ на языке Ассемблера.

Для изучении языка Ассемблера в данных методических рекомендациях приводятся задания как по изучению базового синтаксиста языка (объявления переменных, использованию различных инструкций), так и по изучению архитектурных особенностей процессора (работа с памятью, с вызовом подпрограмм, работе с сопроцессором).

Однако, в настоящих методических рекомендациях не затрагивается несколько важных для низкоуровневого программирования тем, таким как использование макросов в языке Ассемблера, создание многомодульных приложений и др. Сделано это в первую очередь потому, что подробное рассмотрение таких тем отвлекает внимание студентов от изучение основных архитектурных принципов Intel-совместимых процессоров.

Особое внимание уделяется работе с подпрограммами, механизму передачи управления подпрограммам, передачи параметров в подпрограммы, использованию стека, в том числе и для хранения локальных переменных, в том числе и при рекурсивных вызовах. Рассматривается на низком уровне концепция косвенных вызовов. Изучаются особенности работы с сопроцессором, различные способы адресации при обработке массивов.

Несколько заданий специально подобраны для закрепления студентами знаний по представлению чисел в памяти компьютера и пониманию механизмов выполнения действий с такими числами.

Материал соответствует отдельным темам рабочих программ курсов: «Архитектура компьютеров» (специальность «Прикладная информатика»), «Языки программирования» (специальность «Программное обеспечение информационных технологий»).

ЛАБОРАТОРНАЯ РАБОТА № 1.
СТРУКТУРА ПРОГРАММЫ НА ЯЗЫКЕ АССЕМБЛЕРА

1.1. Пример программы на языке Ассемблера

Ознакомитесь с приведённым ниже примером программы на языке Ассемблера. Наберите её в редакторе ASM Editor, скомпилируйте и запустите программу, убедитесь в её работоспособности:

```
.486
.model flat, stdcall
option casemap :none ; чувствительность к регистру букв
                     ; в идентификаторах
include windows.inc
include kernel32.inc
includelib kernel32.lib

.data
    messageString db "Hello, World!!!"
    inputBuffer db 0

.data?
    inputHandle dd ?
    outputHandle dd ?
    numberOfChars dd ?

.code
entryPoint:

    push STD_INPUT_HANDLE      ; передача параметра в функцию
    call GetStdHandle          ; вызов системной функции
    mov inputHandle, EAX       ; сохранение результата функции

    push STD_OUTPUT_HANDLE
    call GetStdHandle
    mov outputHandle, EAX

    push NULL                  ; 5-ый параметр функции,
                               ; системная константа
    push offset numberOfChars  ; 4-ый параметр функции,
                               ; адрес переменной
    push 15                    ; 3-ий параметр функции,
                               ; целочисленная константа
    push offset messageString  ; 2-ой параметр функций,
                               ; адрес массива символов
    push outputHandle          ; 1-ый параметр функции,
                               ; дескриптор системного объекта
    call WriteConsole
```

```
    push NULL
    push offset numberOfChars
    push 1
    push offset inputBuffer
    push inputHandle
    call ReadConsole

    push 0
    call ExitProcess

end entryPoint
```

1.2. Описание системных функций

Ознакомьтесь с приведённым ниже описанием системных функций, используемых в примере выше, и с общими правилами вызова системных функций в языке Ассемблера. На основе полученных знаний модифицируйте приведённый выше пример таким образом, чтобы программа считывала введённую пользователем строку и выводила её в фигурных скобках.

В приведённом примере используется три вида инструкций языка Ассемблера, это команды `push`, `call` и `mov`. Команда `push` используется для работы с сегментом стека приложения, подробнее действие данной команды будет рассмотрено позже, в данном приложении эта команда используется для предварительной передачи параметров в функцию ***перед*** вызовом самой функции, который осуществляется командой `call`. При этом параметры функции, описанные в прототипе этой функции, записанном на языке C++, при вызове на языке Ассемблера передаются в ***обратном*** порядке. То есть первым с помощью команды `push` передаётся последний параметр функции (последний в её записи на языке C++). Команда `mov` в данном примере присваивает переменной, которая является первым операндом команды, значение регистра `EAX`, который является вторым операндом этой команды. Делается это для получения значения, возвращаемого системной функцией, так как все системные функции операционной системы Windows возвращаемое значение помещают именно в этот регистр.

Рассмотрим функции, используемые в приведённом примере:

Функция `GetStdHandle`

Возвращает дескриптор стандартного потока (ввода, вывода или ошибок) консоли. Дескриптор - это 32-битное целое беззнаковое число, по которому операционная система находит системные объекты, к которым относятся и потоки ввода/вывода консоли.

Прототип данной функции на языке C++ выглядит так:

```
HANDLE WINAPI GetStdHandle(DWORD nStdHandle);
```

Параметры функции:

`nStdHandle`

Входной обязательный параметр, который определяет, дескриптор какого именно потока необходимо получить. Может принимать значение одной из целочисленных констант:

`STD_INPUT_HANDLE = -10`	Поток ввода, по умолчанию ввод с клавиатуры в консоли.
`STD_OUTPUT_HANDLE = -11`	Поток вывода, по умолчанию вывод на экран в консоль.
`STD_ERROR_HANDLE = -12`	Поток ошибок, по умолчанию вывод на экран в консоль.

Возвращаемое значение:

В случае успеха возвращается дескриптор запрошенного потока, который затем может использоваться для ввода или вывода данных, используя функции `ReadConsole` или `WriteConsole`. В случае ошибки функция возвращает специальное значение, равное константе `INVALID_HANDLE_VALUE`. В случае, если ошибки не произошло, но приложение не может иметь доступ к консоли, возвращается нулевое значение, равное константе `NULL`.

Функция `WriteConsole`

Записывает символы строкового буфера в указанный поток вывода.

Прототип данной функции на языке C++ выглядит так:

```
BOOL WINAPI WriteConsole(
    HANDLE hConsoleOutput,
    const VOID *lpBuffer,
    DWORD nNumberOfCharsToWrite,
    LPDWORD lpNumberOfCharsWritten,
    LPVOID lpReserved
);
```

Параметры функции:

`hConsoleOutput`

Входной обязательный параметр – дескриптор потока вывода, в который будут выведены символы.

`lpBuffer`

Входной обязательный параметр – адрес начала строкового буфера, содержимое которого будет выведено в поток.

`nNumberOfCharsToWrite`

Входной обязательный параметр – количество символов в буфере, которое необходимо вывести в поток.

`lpNumberOfCharsWritten`

Выходной обязательный параметр – адрес 32-битной ячейки памяти, в которую функция запишет количество фактически выведенных в поток символов.

`lpReserved`

Зарезервированный параметр. Должен быть равен `NULL`.
Возвращаемое значение:
В случае успеха возвращается значение `true`, иначе `false`.

Функция `ReadConsole`

Считывает символы из указанного потока ввода в строковый буфер. При этом считанные символы удаляются из потока ввода.

Прототип данной функции на языке C++ выглядит так:

```
BOOL WINAPI ReadConsole(
    HANDLE hConsoleInput,
    LPVOID lpBuffer,
    DWORD nNumberOfCharsToRead,
    LPDWORD lpNumberOfCharsRead,
    LPVOID pInputControl
);
```

Параметры функции:

`hConsoleInput`

Входной обязательный параметр – дескриптор потока ввода, из которого будут прочитаны символы.

`lpBuffer`

Входной обязательный параметр – адрес начала строкового буфера, в который будут введены символы, прочитанные из потока.

`nNumberOfCharsToRead`

Входной обязательный параметр – количество символов, которое необходимо прочитать из потока. Данное количество должно быть не больше размера самого буфера. При этом если пользователь ввёл с клавиатуры большее количество символов, чем указанное данным параметром, то введённая строка будет урезана до `nNumberOfCharsToRead` символов, остальные символы в буфер помещены не будут.

`lpNumberOfCharsRead`

Выходной обязательный параметр – адрес 32-битной ячейки памяти, в которую функция запишет количество фактически прочитанных из потока символов.

`pInputControl`

Входной необязательный параметр – адрес структуры, в которой можно указать дополнительные параметры потока, такие как символ окончания операции чтения. Данный параметр указывается для чтения символов в кодировке Unicode. В нашем случае можно всегда оставлять его равным `NULL`.

Возвращаемое значение:

В случае успеха возвращается значение `true`, иначе `false`.

Функция `ExitProcess`

Завершает приложение, в котором она вызывается.

Прототип данной функции на языке C++ выглядит так:

```
VOID WINAPI ExitProcess(UINT uExitCode);
```

Параметры функции:

`uExitCode`

Код завершения приложения, возвращаемый операционной системе.

Возвращаемое значение:

Функция не возвращает никакого значения.

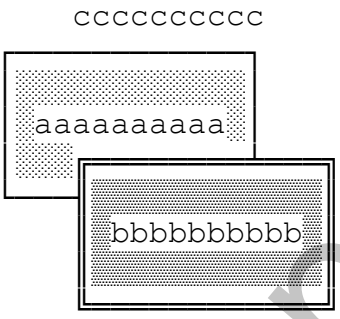
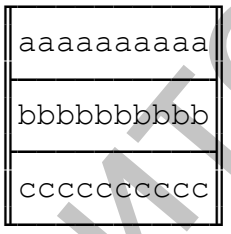
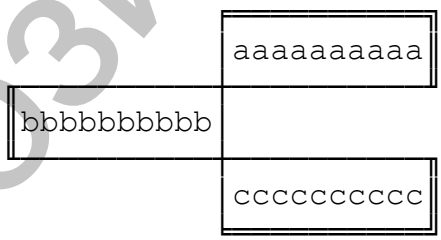
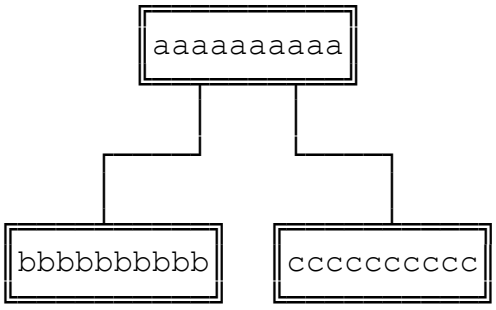
Примечание:

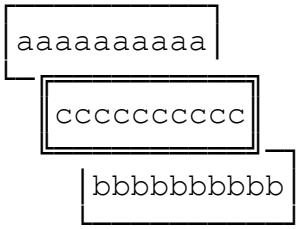
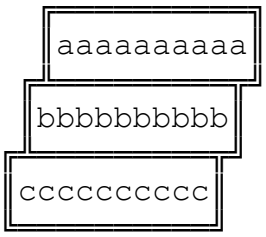
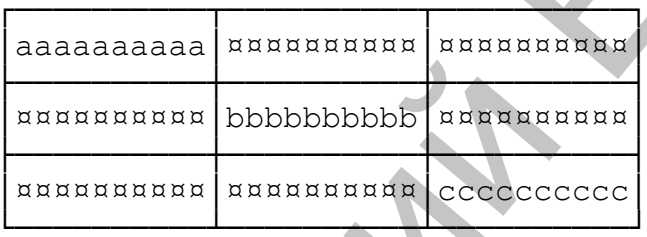
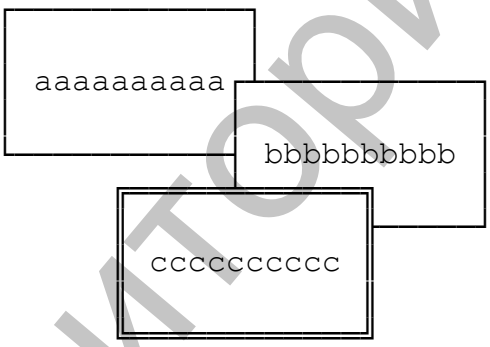
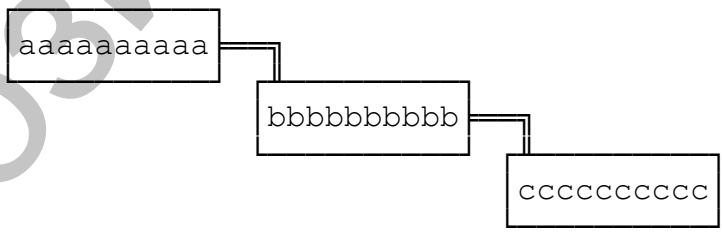
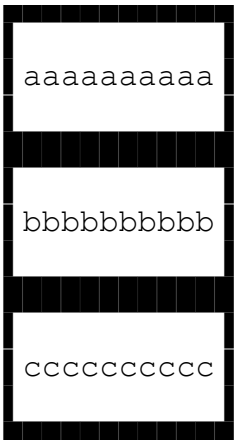
В рассмотренных функциях `WriteConsole` и `ReadConsole` используется явное указание размера буфера, то есть не используются строки с завершающим нулевым символом. Это нужно учитывать при операциях чтения строк из консоли. Если дальнейшая обработка прочитанной строки будет требовать завершающего нуля, то после операции чтения из потока в ячейку памяти с адресом `lpBuffer+[lpNumberOfCharsRead]` необходимо будет явно занести значение 0. Здесь квадратные скобки обозначают обращение к

содержимому ячейки памяти с адресом `lpNumberOfCharsRead`. Записанное здесь выражение в таком виде на языке Ассемблера записано быть не может, правильная запись будет ясна после рассмотрения методов адресации.

1.3. Самостоятельная работа

На языке Ассемблера создать консольное приложение, которое считывает три введённых пользователем строки, усекает или дополняет пробелами каждую строку до 10 символов, после чего выводит текст в соответствии с заданием Вашего варианта:

№ варианта	Шаблон
1.	
2.	
3.	
4.	

5.	
6.	
7.	
8.	
9.	
10.	

Во всех вариантах первая введённая пользователем строка обозначена как "aaaaaaaaaa", вторая – как "bbbbbbbbbb", и третья – как "cccccccccc".

Коды символов, с помощью которых необходимо строить предложенные схемы, можно найти в следующей таблице:

	_0	_1	_2	_3	_4	_5	_6	_7	_8	_9	_A	_B	_C	_D	_E	_F
0_				♥	♦	♣	♠					♂	♀		♫	☼
1_	►	◄	↕	‼	¶	§	▬	↨	↑	↓	→		∟	↔	▲	▼
2_		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3_	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4_	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5_	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6_	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7_	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	
8_	А	Б	В	Г	Д	Е	Ж	З	И	Й	К	Л	М	Н	О	П
9_	Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ъ	Ы	Ь	Э	Ю	Я
A_	а	б	в	г	д	е	ж	з	и	й	к	л	м	н	о	п
B_	░	▒	▓	│	┤	╡	╢	╖	╕	╣	║	╗	╝	╜	┐	
C_	└	┴	┬	├	─	┼	╞	╟	╚	╔	╩	╦	╠	═	╬	╧
D_	╨	╤	╥	╙	╘	╒	╓	╫	╪	┘	┌	▀	▄	▌	▐	■
E_	р	с	т	у	ф	х	ц	ч	ш	щ	ъ	ы	ь	э	ю	я
F_	Ё	ё	Є	є	Ї	ї	Ў	ў	°	∙	·	√	№	¤	■	

В данной таблице крайний левый столбец содержит старшие цифры шестнадцатеричного кода символа, верхняя строка содержит младшие цифры шестнадцатеричного кода символа, а на пересечении соответствующих строки и столбца находится сам символ.

Замечание: рассмотрим операцию чтения строки из консоли

```
push NULL
push offset numberOfChars
push 1000
push offset inputBuffer
push inputHandle
call ReadConsole
```

где переменные описаны следующим образом

```
inputBuffer db 1000 dup (" ")
inputHandle dd ?
numberOfChars dd ?
```

При таком чтении если с клавиатуры будет прочитано более 10 символов, то при использовании вывода с помощью функции

WriteConsole можно будет ограничить количество выводимых символов третьим параметром. Однако при вводе трёх символов, допустим, строки "abc" пользователь нажимает клавишу Enter для завершения ввода, и в строку на 4-ую и 5-ую позицию будут записаны коды символов, соответствующие этой клавише (символ с кодом 13 и символ с кодом 10). И хоть при объявлении строки inputBuffer мы резервируем 1000 пробельных символов (с кодом 32), при выводе первых 10 символов такой строки будут выведены 3 символа, введённых пользователем ("abc"), затем символ перехода на новую строку (код 13), затем символ возврата каретки (код 10), после чего оставшиеся 5 пробельных символов (код 32). Для того чтобы избежать вывода символов с кодами 13 и 10, нужно на их место записать пробельные символы (с кодом 32). Добиться этого можно с помощью переменной numberOfChars. Для этого можно занести в регистр EBX адрес введённой строки inputBuffer, в регистр EAX занести количество прочитанных символов (фактически, длину строки inputBuffer). А затем в ячейки памяти с адресами EBX + EAX - 1 и EBX + EAX - 2 занести код пробельного символа

```
mov EBX, offset inputBuffer
mov EAX, numberOfChars
mov byte ptr [ EBX + EAX - 1 ], " "
mov byte ptr [ EBX + EAX - 2 ], " "
```

ЛАБОРАТОРНАЯ РАБОТА № 2. АРИФМЕТИЧЕСКИЕ ОПЕРАЦИИ В ЯЗЫКЕ АССЕМБЛЕРА

2.1. Пример ввода-вывода целых чисел в языке Ассемблера

Ознакомьтесь с приведёнными ниже фрагментами программы на языке Ассемблера и на основе этих фрагментов составьте программу, считывающую с клавиатуры два числа и выводящую их сумму, разность, произведение и частное.

Вывод в консоль строки с завершающим нулём:

```
push offset message         ; указатель на строку, описанную,
                            ; например, так:
                            ; message db "hello", 0
call lstrlen                ; вычисление длины строки,
                            ; завершающейся нулём, при этом
                            ; длина возвращается в регистре EAX
push NULL
push offset numberOfChars
push EAX                    ; ограничение количества выводимых
                            ; символов вычисленной ранее длиной
                            ; строки
push offset message
push outputHandle
call WriteConsole
```

Ввод строки из консоли с добавлением в конец строки завершающего нуля:

```
push NULL
push offset numberOfChars
push 1000
push offset buffer
push inputHandle
call ReadConsole                    ; чтение с клавиатуры
                                    ; строки, при этом длина
                                    ; прочитанной строки
                                    ; записывается в переменную
                                    ; numbersOfChars
mov EDX, offset buffer              ; сохранение адреса
                                    ; прочитанной строки
mov EAX, numberOfChars              ; сохранение длины
                                    ; прочитанной строки
mov byte ptr [ EDX + EAX - 2 ], 0   ; запись завершающего нуля в
                                    ; конец строки, при этом
                                    ; вычитание 2 из адреса
                                    ; необходимо для
```

```
                    ; отбрасывания символов с
                    ; кодами 13 и 10 в конце
                    ; строки, помещаемых в буфер
                    ; после нажатия
                    ; пользователем клавиши
                    ; Enter
```

Преобразование строки с завершающим нулём, содержащей десятичную запись целого беззнакового числа, в 32-битное число (для использования данной функции необходимо дополнительно подключить заголовочный файл `masm32.inc` и библиотеку `masm32.lib`):

```
push offset buffer ; адрес строки, содержащей запись числа
call atodw         ; преобразование строки в число, результат
                   ; возвращается в регистре EAX
```

Преобразование 32-битного знакового числа в строку с завершающим нулём, содержащую запись этого числа в 10-ричной системе счисления (для использования данной функции необходимо дополнительно подключить заголовочный файл `masm32.inc` и библиотеку `masm32.lib`):

```
push offset buffer ; адрес строки, в которую будет записано
                   ; число
push EBX           ; преобразуемое число
call dwtoa
```

2.2. Описание стандартных функций Ассемблера MASM 32

Функции операционной системы Windows, как используемые в данной лабораторной и описанные ниже, так и все остальные, ***не модифицируют*** регистры `EBX`, `ESI`, `EDI`, `EBP`, и ***могут модифицировать*** регистры `EAX`, `ECX`, `EDX`. Возвращаемые системными функциями значения передаются через регистры `AL` (1-байтовый результат), `AX` (2-байтовый результат), `EAX` (4-байтовый результат) или в паре регистров `EDX:EAX` (8-байтовый результат).

Рассмотрим функции, используемые в приведённом примере:

Функция `lstrlen`

Возвращает длину строки, завершающуюся нулевым символом (сам нулевой символ в длине не учитывается).

Прототип данной функции на языке C++ выглядит так:

```
int WINAPI lstrlen(LPCTSTR lpString);
```

Параметры функции:

`lpString`
Входной обязательный параметр, адрес первого символа строки.
Возвращаемое значение:
Длина строки или 0, в случае, если адрес строки равен `NULL`.

Функция `atodw`

Преобразует строку с завершающим нулевым символом, содержащую запись целого беззнакового числа в 10-тичной системе счисления, в 32-битное целое число.

Прототип данной функции на языке C++ выглядит так:

```
DWORD atodw(LPCTSTR lpString);
```

Параметры функции:

`lpString`

Входной обязательный параметр, адрес первого символа строки.
Возвращаемое значение:
Целое число, полученное в результате преобразования.

Функция `dwtoa`

Записывает в строку представление 32-битного целого знакового числа в 10-тичной системе счисления.

Прототип данной функции на языке C++ выглядит так:

```
void dwtoa(
    DWORD nValue,
    LPSTR lpString
);
```

Параметры функции:

`nValue`

Входной обязательный параметр – 32-битное целое знаковое число, преобразуемое в строковое представление.

`lpString`

Входной обязательный параметр – адрес начала строкового буфера, в который будет записанное строковое представление числа в 10-тичной системе счисления.
Возвращаемое значение:
Функция не возвращает никакого значения.

2.3. Самостоятельная работа «Простое арифметическое выражение»

Разработайте приложение для вычисления значения арифметического выражения, соответствующего вашему варианту.

Варианты задания:

№ варианта	выражение	№ варианта	выражение
1.	$3 \cdot a - \frac{a+b}{2}$	2.	$b^2 - 4 \cdot a \cdot c$
3.	$\frac{(a+b)^2}{a-b}$	4.	$\frac{-b+c}{2 \cdot a}$
5.	$a + \frac{(b-c)^2}{3}$	6.	$\frac{(a-b) \cdot (a+b)}{4}$
7.	$a^2 - \frac{b^2}{5}$	8.	$\frac{a}{b} + 3 \cdot (a-b)$
9.	$\frac{a+b}{a} - 4 \cdot a$	10.	$\left(a - \frac{b}{2}\right) \cdot \left(a + \frac{b}{2}\right)$

Указания: Не преобразовывайте выражения (с целью упрощения и/или изменения порядка действий).

2.4. Самостоятельная работа «Учёт флага CF»

Разработайте приложение, позволяющее ввести с клавиатуры 3 целых 32-битных беззнаковых числа `a`, `b` и `c`, после чего вычисляющее значение выражения, соответствующее варианту. При возведении в степень числа `a` циклы не использовать и выполнить возведение в степень не более чем за `N` операций умножения. Считать, что степень числа `a` не превосходит 64-битного беззнакового числа.

Варианты задания:

№	Выражение	N	Тестовые данные			Ответ
			a	b	c	
1.	$\left(a^7 + b/32\right)/c + \left(a^7 + b/32\right)\%\, c$	4	35	3 000 000 000	1 000 000	111 308
2.	$\left(a^9 + b/16\right)/c + \left(a^9 + b/16\right)\%\, c$	4	15	4 000 000 000	1 000 000	398 068
3.	$\left(a^{10} + 128 \cdot b\right)/c + \left(a^{10} + 128 \cdot b\right)\%\, c$	4	12	20 000 000	1 000 000	428 701
4.	$\left(a^{11} + 64 \cdot b\right)/c + \left(a^{11} + 64 \cdot b\right)\%\, c$	5	11	40 000 000	1 000 000	958 482
5.	$\left(a^{12} + 8 \cdot b\right)/c + \left(a^{12} + 8 \cdot b\right)\%\, c$	4	12	50 000 000	1 000 000	9 364 756

6.	$\left(a^{13}+b/8\right)/c+\left(a^{13}+b/8\right)\%c$	5	11	2 000 000 000	1 000 000	34 666 893
7.	$\left(a^{14}+16\cdot b\right)/c+\left(a^{14}+16\cdot b\right)\%c$	5	7	25 000 000	1 000 000	751 472
8.	$\left(a^{15}+512\cdot b\right)/c+\left(a^{15}+512\cdot b\right)\%c$	6	6	5 000 000	1 000 000	1 457 320
9.	$\left(a^{21}+256\cdot b\right)/c+\left(a^{21}+256\cdot b\right)\%c$	6	3	10 000 000	1 000 000	366 223
10.	$\left(a^{24}+32\cdot b\right)/c+\left(a^{24}+32\cdot b\right)\%c$	5	3	50 000 000	1 000 000	820 510

Указания: при разработке приложения необходимо помнить, что операция деления может приводить к исключительным ситуациям в двух случаях: во-первых, при делении на 0; во-вторых, если частное слишком велико для размещения в регистре `EAX`.

ЛАБОРАТОРНАЯ РАБОТА № 3.
ВЕТВЛЕНИЯ И ЦИКЛЫ В ЯЗЫКЕ АССЕМБЛЕРА

3.1. Самостоятельная работа «Ветвления в языке Ассемблера»

Разработайте приложение, вычисляющее значение выражения, соответствующего варианту.

Варианты задания:

№	выражение
1.	$z = \frac{\max(x^2; \max(y;10))}{\min(x;y)}$
2.	$z = \frac{\min(\min(10;y); \max(x;y))}{\max(x\%10; y\%10)}$
3.	$z = \frac{\min(\max(x;5); \max(y;5))}{\min(10; \lvert x-y \rvert)}$
4.	$z = \frac{\min(\max(x;5); \max(y;5))}{\max(\min(x;10); \min(y; x\%5))}$
5.	$z = \frac{\min(\max(x\%10;5); \max(y\%10;5))}{\min(x\%4; y\%4)}$
6.	$z = \frac{\min(\max(x^2;10); \min(y^4;10))}{\min(x\%y; y\%x)}$
7.	$z = \frac{\min(\max(10; \lvert x-y \rvert); \max(x;y))}{\min(x;y)}$
8.	$z = \frac{\min(\max(x+y; x\cdot y); \max(2x+y; 2y\%x))}{\max(2x-y; 2y-x)}$
9.	$z = \frac{\min(\max(2x-y; y); \max(2y-x; x))}{\max(x\%10; y\%10)}$
10.	$z = \frac{\min(\max(x\cdot y; x+y); \max((x+y)\%10; (x\cdot y)\%10))}{\max(\lvert x \rvert; \lvert y \rvert)+1}$

3.2. Самостоятельная работа «Циклы в языке Ассемблера»

Разработайте приложение для решения задачи вашего варианта.

Варианты задания:

1. Не используя команд `mul` и `imul` найдите произведение двух целых чисел.

2. Возведите заданное целое число в целую неотрицательную степень, используя минимальное число операций умножения.
3. Найдите сумму первых `N` элементов последовательности Фибоначчи (`a[0] = a[1] = 1; a[i] = a[i-1] + a[i-2], i = 2, 3, 4, ...`).
4. Не используя команды `div` и `idiv` найдите неполное частное от деления двух целых чисел.
5. Найдите сумму цифр заданного целого числа.
6. Найдите количество делителей заданного натурального числа.
7. Найдите количество цифр заданного целого числа.
8. Не используя команды `div` и `idiv` найдите остаток от деления двух целых чисел.
9. Найдите наибольший общий делитель двух целых чисел, используя алгоритм Евклида.
10. Найдите факториал целого неотрицательного числа.

ЛАБОРАТОРНАЯ РАБОТА № 4.
ПОДПРОГРАММЫ И МАССИВЫ В ЯЗЫКЕ АССЕМБЛЕРА

4.1. Пример использования подпрограмм

Внимательно изучите структуру подпрограммы `arrayToStr`, предназначенной для формирования строки, содержащей все элементы массива. Подготовьте программу, демонстрирующую использование данной подпрограммы.

```
; ------------- Подпрограмма ArrayToStr --------------------
; void arrayToStr(char* buffer, int* array, int size)
;
; Формирует строковое представление целочисленного массива
;
; Входные параметры:
; buffer ( [EBP + 8] )  - указатель на строку, в которой будет
;                         формироваться представление массива
; array  ( [EBP + 12] ) - указатель на массив
; size   ( [EBP + 16] ) - количество элементов в массиве

.data
  template db "%d ", 0 ; Образец строки для целого числа

.code

arrayToStr:
  ; Стандартный пролог функции
  push EBP
  mov EBP, ESP
  ; начало цикла - пока есть необработанные элементы
  cycle:
    cmp dword ptr [ EBP + 16 ], 0
    je endFunction
    ; Формирование текстового представления целого числа
    mov EAX, [ EBP + 12 ] ; указатель на очередное число
    push [ EAX ]          ; целое число (элемент массива),
                          ; преобразуемое в строку
    push offset template  ; шаблон строки, в который
                          ; подставляются значения
    push [ EBP + 8 ]      ; адрес буфера для размещения
                          ; итоговой строки
    call wsprintf         ; в EAX записывается число символов,
                          ; записанных в буфер
    add ESP, 12           ; выровняем стек
    ; Подготовка к следующему числу
    add [ EBP + 8 ], EAX  ; рассчитаем адрес строки для
                          ; следующего числа
```

```
    add dword ptr [ EBP + 12 ], 4 ; перемещаем указатель на
                                  ; следующий элемент массива
    dec dword ptr [ EBP + 16 ]    ; уменьшаем счетчик
  ; конец цикла
  jmp cycle
  endFunction:
  ; Стандартный эпилог функции
  pop EBP
; Выйти и выровнять стек
ret 12
```

4.2. Описание функции wsprintf

Функция `wsprintf`

Записывает форматированные данные в строковый буфер. Все аргументы конвертируются в строку и копируются в выходной буфер согласно шаблону, указанного в строке форматирования. Функция дописывает завершающий строку нулевой символ в конец строкового буфера, но не учитывает этот символ, когда возвращает общее количество записанных в выходную строку символов.

Прототип данной функции на языке C++ выглядит так:

```
int wsprintf(
    LPTSTR lpOut,
    LPCTSTR lpFmt,
    ...
);
```

Параметры функции:

`lpOut`

Выходной обязательный параметр – адрес начала строкового буфера, в который будут скопированы форматированные данные. Максимальный объём буфера 1024 байта.

`lpFmt`

Входной обязательный параметр – адрес начала строки форматирования, содержащей шаблон, определяющий, каким образом будут форматироваться данные, копируемые в выходную строку. Подробнее смотрите в пояснении.

`...`

Входные необязательные параметры – данные – переменные различных типов, форматирование которых будет производиться в соответствии с указанными в строке форматирования шаблонами.
Возвращаемое значение:
Количество символов, скопированных функцией в выходной буфер (в этом количестве не учитывается завершающий нулевой символ, добавляемый в конец строки).
Пояснение:
Функция последовательно читает строку форматирования, и если очередной символ это строки не равен символу `'%'`, то этот символ копируется в выходной буфер. Если же функция из строки форматирования читает символ `'%'`, то функция считывает шаблон, имеющий вид:

```
%тип
```

Далее функция читает очередной аргумент из списка аргументов-данных, преобразует значение этого аргумента в строковое представление в соответствии с типом, указанным в шаблоне, и полученную строку вместо самого шаблона копирует в выходную строку. Процесс продолжается до тех пор, пока строка форматирования не закончится. Количество шаблонов в строке форматирования не должно быть больше количества аргументов-данных.
Некоторые возможные значения типа, используемого в шаблоне:

тип	**описание**
`d`	целое знаковое число в десятичной системе счисления
`u`	целое беззнаковое число в десятичной системе счисления
`x`	целое беззнаковое число в шестнадцатеричной системе счисления (цифры от A до F записываются в нижнем регистре)
`X`	целое беззнаковое число в шестнадцатеричной системе счисления (цифры от A до F записываются в верхнем регистре)
`c`	одиночный символ
`s`	строка

Функция использует переменное число параметров, поэтому она не выравнивает стек самостоятельно. Это нужно сделать после вызова функции вручную, удалив из стека все параметры, переданные в функцию через стек.

4.3. Самостоятельная работа

Составьте программу, которая обрабатывает целочисленный массив из нескольких (не менее 10) элементов следующим образом:

– заполняет массив некоторыми числами (согласно пункту *a* соответствующего варианта задания);

– выводит сформированный массив на экран;

– подсчитывает и выводит на экран сумму элементов массива;

– изменяет элементы массива по некоторому правилу (согласно пункту *b* варианта задания);

– выводит полученный массив на экран;

– подсчитывает и выводит на экран новую сумму элементов массива.

Заполнение, изменение и подсчет суммы элементов массива необходимо оформить в виде отдельных подпрограмм.

Варианты задания:

№	Задание *A*	Задание *B*
1.	арифметическая прогрессия: $a_0 = 18$, $d = 43$	элементы, кратные четырём, уменьшить в четыре раза
2.	геометрическая прогрессия: $a_0 = 3$, $q = -3$	каждый отрицательный элемент уменьшить в 3 раза
3.	числа Фибоначчи: $a_0 = a_1 = 1$, $a_2 = a_1 + a_0$, $a_3 = a_2 + a_1$, ...	чётные элементы возвести в квадрат
4.	последовательность квадратов натуральных чисел: 1, 4, 9, ...	поменять знак у нечётных чисел
5.	последовательность кубов чисел, начиная от -5: -125, -64, -27, ...	чётные числа возвести в квадрат
6.	последовательность чисел, кратных 7: 7, 14, 21, ...	чётные числа уменьшить в 2 раза
7.	последовательность остатков от деления числа 101 на 1, 2, 3, ...	поменять знак у чётных чисел
8.	последовательность квадратов чисел, начиная от -10: 100, 81, ...	элементы, заканчивающиеся на 6, увеличить в 3 раза
9.	последовательность степеней тройки: 3, 9, 27, ...	элементы, заканчивающиеся на 9, увеличить на 1
10.	факториалы чисел от 1 до N	от каждого элемента массива отнять среднее арифметическое всех элементов массива

ЛАБОРАТОРНАЯ РАБОТА № 5.
РАБОТА С СОПРОЦЕССОРОМ НА ЯЗЫКЕ АССЕМБЛЕРА

5.1. Пример работы с сопроцессором

Изучите пример использования сопроцессора для вычисления значения функции:

$$y(x) = \sqrt{|\cos x + \sin x|}$$

```
.486
.model flat, stdcall
option casemap :none
include windows.inc
include kernel32.inc
include masm32.inc
include user32.inc
includelib kernel32.lib
includelib masm32.lib
includelib user32.lib

.data
    template db "sqrt(|cos(%s) + sin(%s)|) = %s", 0
    inputMessage db "input floating point number > ", 0

.data?
    inputHandle dd ?
    outputHandle dd ?
    numberOfChars dd ?
    inputBuffer db ?
    numberX db 1000 dup (?)
    numberY db 1000 dup (?)
    answer db 100 dup (?)
    x dq ?
    y dq ?

.code

entryPoint:
    ; получение дескриптора потоков
    push STD_INPUT_HANDLE
    call GetStdHandle
    mov inputHandle, EAX
    push STD_OUTPUT_HANDLE
    call GetStdHandle
    mov outputHandle, EAX
```

```
; вывод приглашения к вводу
push offset inputMessage
call lstrlen
push NULL
push offset numberOfChars
push EAX
push offset inputMessage
push outputHandle
call WriteConsole

; прочитать строку с числом x
push NULL
push offset numberOfChars
push 1000
push offset numberX
push inputHandle
call ReadConsole
mov EDX, offset numberX
mov EAX, numberOfChars
mov byte ptr [ EDX + EAX - 2 ], 0

; преобразование строки в дробное число
push offset x
push offset numberX
call StrToFloat

; вычисление выражения с помощью сопроцессора
finit
fld x
fcos
fld x
fsin
fadd ST (0), ST (1)
fabs
fsqrt
fstp y

; преобразование числа-результата в строку
push offset numberY
push dword ptr y + 4
push dword ptr y
call FloatToStr

; формирование строки-результата для вывода
push offset numberY
push offset numberX
push offset numberX
push offset template
push offset answer
call wsprintf
add ESP, 20
```

```
    ; вывод строки-результата
    push offset answer
    call lstrlen
    push NULL
    push offset numberOfChars
    push EAX
    push offset answer
    push outputHandle
    call WriteConsole

    ; задержка закрытия окна
    push NULL
    push offset numberOfChars
    push 1
    push offset inputBuffer
    push inputHandle
    call ReadConsole

    ; выход из программы
    push 0
    call ExitProcess
END entryPoint
```

Подготовьте программу, демонстрирующую использование данного примера.

5.2. Самостоятельная работа «Арифметическое выражение»

Разработайте программу для вычисления значения арифметического выражения (согласно варианту) с использованием сопроцессора.

Варианты задания:

№ варианта	выражение	№ варианта	выражение
1.	$3 \cdot a - \frac{a+b}{2}$	2.	$b^2 - 4 \cdot a \cdot c$
3.	$\frac{(a+b)^2}{a-b}$	4.	$\frac{-b+c}{2 \cdot a}$
5.	$a + \frac{(b-c)^2}{3}$	6.	$\frac{(a-b) \cdot (a+b)}{4}$
7.	$a^2 - \frac{b^2}{5}$	8.	$\frac{a}{b} + 3 \cdot (a-b)$
9.	$\frac{a+b}{a} - 4 \cdot a$	10.	$\left(a - \frac{b}{2}\right) \cdot \left(a + \frac{b}{2}\right)$

Указание. Значения выражений требуется вычислить без использования вспомогательных переменных (то есть для хранения промежуточных результатов разрешается использовать только стек сопроцессора). Для этого тщательно продумайте последовательность выполнения действий. Не забывайте, что для целочисленных операндов требуются специальные команды.

5.3. Самостоятельная работа «Массивы вещественных чисел»

Постройте таблицу значений функции (см. свой вариант) на отрезке $[x_1; x_2]$ с шагом Δx при заданных пользователем значениях переменных x_1, x_2, Δx, a, b:

Варианты задания:

№ варианта	Функция
1.	$y = \begin{cases} \sqrt{-ax + a}, & x < 1 \\ b \ln x, & x \geq 1 \end{cases}$
2.	$y = \begin{cases} \dfrac{b}{1 + x^2}, & x \leq 0 \\ b \cos ax, & x > 0 \end{cases}$
3.	$y = \begin{cases} ax^2 + b, & x \leq 0 \\ b - a^x + 1, & x > 0 \end{cases}$
4.	$y = \begin{cases} \dfrac{1}{a^x}, & x \leq 0 \\ \cos bx, & x > 0 \end{cases}$
5.	$y = \begin{cases} a\sqrt{1 - x}, & x \leq 1 \\ b \log_3 x, & x > 1 \end{cases}$
6.	$y = \begin{cases} \ln\left(\sin ax + \dfrac{\pi}{2}\right), & x \leq 0 \\ \ln\left(\pi + bx^2\right) - \ln \dfrac{2}{1 + x}, & x > 0 \end{cases}$
7.	$y = \begin{cases} a \cos x, & x \leq 0 \\ a - b^x, & x > 0 \end{cases}$

8.	$y=\begin{cases}\sqrt{a\lvert x+1\rvert}, & x\le -1\\ \dfrac{b}{\sqrt{2}}-\sqrt{\dfrac{b^2}{1+x^2}}, & x>-1\end{cases}$
9.	$y=\begin{cases}\ln\left(x^2+a\right), & x\le 0\\ \sqrt{x^2-bx+\ln^2 a}, & x>0\end{cases}$
10.	$y=\begin{cases}ax-bx^2, & x<1\\ \dfrac{a-b}{\ln\left(e+\sqrt{x^2-1}\right)}, & x\ge 1\end{cases}$

Указания:

1. Формирование массивов значений аргумента и функции оформить в виде отдельной функции.
2. Вывод таблицы значений можно организовать, используя функцию `wsprintf`. Для этого в строку шаблона можно включить две строки: `"| %s | %s |"`, – вместо первого значения подставлять строку, содержащую значение аргумента, вместо второго – значение функции. Для формирования строки, содержащей десятичное представление вещественного числа, необходимо использовать функцию `FloatToStr`. При этом следует помнить, что у функции `FloatToStr` нет возможностей по форматированию вещественного числа, поэтому для вывода чисел с одинаковой шириной (в символах) можно указывать ширину вывода строки в функции `wsprintf` с помощью специального синтаксиса строки-шаблона, например, строка-шаблон `"%4.6s"` выводит некоторую строку, используя не менее 4 символов (если их меньше, то слева от строки добавляется необходимое количество пробелов) и не более 6 символов (если их больше, то строка усекается). Но следует помнить, что если функция `FloatToStr` выводит, например, число в виде следующей строки `"1.23456e-10"`, а в функцию `wsprintf` она подставляется вместо шаблона `"%5.5s"`, то результатом будет строка `"1.234"`, что не соответствует исходному числу.

ЛАБОРАТОРНАЯ РАБОТА № 6.
РЕКУРСИЯ В ЯЗЫКЕ АССЕМБЛЕРА

6.1. Самостоятельная работа

Опишите рекурсивную функцию и с помощью этой функции (в зависимости от варианта):

1. посчитайте n-ый член последовательности Фибоначчи;
2. определите, содержится ли заданный ключ в полном поисковом бинарном дереве (ключи – целые числа);
3. вычислите факториал заданного целого числа;
4. вычислите a_n, где a и n – натуральные числа;
5. определите индекс заданного числа в отсортированном массиве целых чисел с применением алгоритма бинарного поиска;
6. вычислите A_n^k, где n и k – целые числа, удовлетворяющие неравенству $0 \le k \le n$, используя соотношения $A_n^0 = 1$, $A_{n+1}^{k+1} = (n+1)A_n^k$;
7. найдите наибольший общий делитель двух заданных натуральных чисел;
8. вычислите сумму всех ключей в полном бинарном дереве (ключи – целые числа);
9. вычислите C_n^k, где n и k – целые числа, удовлетворяющие неравенству $0 \le k \le n$, используя соотношения $C_n^0 = C_n^n = 1$, $C_{n+1}^{k+1} = C_n^{k+1} + C_n^k$;
10. выведите все перестановки натуральных чисел от 1 до n.

ЛАБОРАТОРНАЯ РАБОТА № 7. КОСВЕННЫЙ ВЫЗОВ ПОДПРОГРАММ В ЯЗЫКЕ АССЕМБЛЕРА

7.1. Пример косвенного вызова

Изучите примеры описания и использования структур, а также примеры косвенного вызова подпрограмм. Напишите программу, демонстрирующую эти примеры. Описание работы со структурами в языке Ассемблера смотри ниже.

Пример описания структуры и объявления массива:

```
; описание структуры для хранения комплексного числа
complex struct
    re dq ?
    im dq ?
complex ends

.data
    ; объявление массива из 5-ти комплексных чисел
    array complex < 1.0,  2.0>,
                  <-4.5,  1.25>,
                  < 0.0, -3.1>,
                  < 3.5, -1.5>,
                  < 2.0,  3.0>
```

Пример подпрограммы, сравнивающих два комплексных числа, переданных в подпрограмму по адресу соответствующих ячеек памяти. Подпрограмма сравнивает только вещественные части двух комплексных чисел и возвращает в регистре EAX число: -1, если первый аргумент меньше второго; +1, если первый аргумент больше второго; 0, если аргументы равны:

```
complexCompareByRealPart:
    ; получаем адрес первого аргумента
    mov EAX, [ ESP + 4 ]
    ; заносим в вершину стека сопроцессора
    ; действительную часть первого аргумента
    fld qword ptr [ EAX ]
    ; получаем адрес второго аргумента
    mov EAX, [ ESP + 8 ]
    ; сравниваем действительную часть первого
    ; аргумента из вершины стека сопроцессора
    ; с действительной частью второго аргумента,
    ; находящегося в памяти; при этом число из
    ; вершины стека сопроцессора удаляется
    fcomp qword ptr [ EAX ]
```

```
    ; копируем флаги сопроцессора в регистр
    ; флагов основного процессора
    fstsw AX
    sahf
    ; возвращаем необходимый целочисленный результат
    ja great1
    jb less1
        mov EAX, 0
        jmp return1
    great1:
        mov EAX, 1
        jmp return1
    less1:
        mov EAX, -1
    return1:
ret 8
```

Пример подпрограммы, сравнивающих два комплексных числа, переданных в подпрограмму по адресу соответствующих ячеек памяти. Подпрограмма сравнивает модули двух комплексных чисел и возвращает в регистре EAX число: -1, если первый аргумент меньше второго; +1, если первый аргумент больше второго; 0, если аргументы равны:

```
complexCompareByModulus:
    ; получаем адрес первого аргумента
    mov EAX, [ ESP + 4 ]
    ; заносим в вершину стека сопроцессора
    ; действительную часть первого аргумента
    fld qword ptr [ EAX ]
    ; возводим действительную часть первого
    ; аргумента в квадрат
    fmul ST (0), ST (0)
    ; заносим в вершину стека сопроцессора
    ; мнимую часть первого аргумента
    fld qword ptr [ EAX + 8 ]
    ; возводим мнимую часть первого аргумента
    ; в квадрат
    fmul ST (0), ST (0)
    ; складываем в регистре ST (1) квадрат
    ; действительной и мнимой части, после
    ; второй операнд ST (0) удаляется из стека,
    ; а результат подымается в вершину стека
    faddp ST (1), ST (0)
```

```
    ; производим вычисление модуля второго
    ; комплексного числа аналогично первому
    mov EAX, [ ESP + 8 ]
    fld qword ptr [ EAX ]
    fmul ST (0), ST (0)
    fld qword ptr [ EAX + 8 ]
    fmul ST (0), ST (0)
    faddp ST (1), ST (0)

    ; находим разность квадратов модулей
    ; комплексных чисел
    fsubp ST (1), ST (0)
    ; сравнение разности модулей с нулём
    ficomp zero
    fstsw AX
    sahf
    ; возвращаем целочисленный результат
    ja great2
    jb less2
        mov EAX, 0
        jmp return2
    great2:
        mov EAX, 1
        jmp return2
    less2:
        mov EAX, -1
    return2:
ret 8
```

Пример поиска адреса максимального элемента массива, при этом сравнение элементов производится с помощью некоторой функции, адрес которой передаётся в данную подпрограмму:

```
max:
    ; первый параметр – адрес первого
    ; элемента массива
    mov EBX, [ ESP + 4 ]
    ; количество элементов массива
    mov ECX, [ ESP + 8 ]
    ; адрес функции, которая будет
    ; сравнивать два элемента массива
    mov EDX, [ ESP + 12 ]

    ; заносим в регистр ESI адрес
    ; первого элемента массива, в
    ; дальнейшем здесь будем хранить
    ; адрес максимального элемента
    mov ESI, EBX
```

```
    beginCycle:
        ; проверяем, достигнут ли
        ; конец массива
        cmp ECX, 0
        je endCycle
        ; передаём в функцию сравнения
        ; адрес текущего максимального
        ; элемента массива
        push ESI
        ; передаём в функцию сравнения
        ; адрес очередного элемента массива
        push EBX
        ; вызываем функцию сравнения,
        ; используя концепцию косвенного вызова
        call EDX
        ; сравниваем результат, возвращённый
        ; функцией сравнения, с нулём
        cmp EAX, 0
        ; если текущий максимальный элемент
        ; меньше очередного элемента массива
        jng skip
            ; запоминаем в качестве адреса
            ; максимального элемента массива
            ; адрес текущего элемента массива
            mov ESI, EBX
        skip:
        ; переходим к следующему элементу массива
        ; (используем смещение 16 байт - размер
        ; структуры complex)
        add EBX, 16
        ; уменьшаем количество элементов массива
        dec ECX
        jmp beginCycle
    endCycle:
    ; заносим в регистр EAX адрес максимально
    ; элемента массива
    mov EAX, ESI
ret 12
```

Пример вызова подпрограммы `max` для поиска максимального элемента в массиве комплексных чисел, используя сравнение действительных частей элементов:

```
push complexCompareByRealPart
push 5
push offset array
call max
```

Пример вызова подпрограммы `max` для поиска максимального элемента в массиве комплексных чисел, используя сравнение модулей элементов:

```
push complexCompareByModulus
push 5
push offset array
call max
```

7.2. Описание работы со структурами на языке Ассемблера

Для объявления структур в языке Ассемблера используются ключевые слова `struct` (начало объявление структуры) и `ends` (конец объявления структуры). Перед каждым из этих ключевых слов указывается имя структуры.

Пример:

```
имя_структуры struct
    ; описание полей структуры
имя_структуры ends
```

При описании полей структуры используются директивы объявления переменных, такие же, что и в сегментах инициализированных и неинициализированных данных. При этом те поля структуры, которые объявлены, как инициализированные данные, имеют некоторое значение по умолчанию.

Объявлять переменную типа структуры можно как в сегменте `.data`, так и в сегменте `.data?`. В обоих случаях структуру нужно инициализировать явно, используя скобки вида `<...>`

Пример:

```
имя_переменной имя_структуры <
    значение_первого_поля структуры,
    значение_второго_поля_структуры,
    ...,
    значение_N_го_поля структуры
>
```

При этом в случае описания неинициализированной структуры, вместо значения полей структуры используется знак `'?'`. Если же поле структуры при объявлении было проинициализировано, то при описании переменной соответствующего типа это поле примет значение по умолчанию (указанной в объявлении). В ином случае значение поля будет считаться не проинициализированным.

Для обращения к полям структуры можно использовать специальный оператор `'.'`.

Пример:

```
имя_переменной.имя_поля
```

или же используя адрес начала структуры и смещение полей структуры (зная размер каждой структуры в байтах).

7.3. Самостоятельная работа

Напишите программу, выводящую список некоторых элементов, отсортированный с использованием указанного алгоритма сортировки (см. свой вариант) по двум критериям (сначала вывести список, отсортированный по одному критерию; затем вывести тот же список, отсортированный по второму критерию). Алгоритм сортировки реализовать независимо от используемого критерия сортировки, используя концепцию косвенного вызова.

Варианты задания:

1. Отсортировать методом выбора список результатов замеров массы тела спортсмена (день, месяц, год взвешивания, результат взвешивания – дробное число) по массе и по дате взвешивания.
2. Отсортировать методом выбора список результатов замеров температуры тела больного (часы, минуты, температура – дробное число) по значению температуры и по времени измерения.
3. Отсортировать методом выбора список правильных многоугольников (количество сторон, длина стороны – дробное число) по площади и по периметру.
4. Отсортировать методом выбора список билетов в кинотеатр (ряд, место, стоимость – дробное число) по стоимости и по ряду с местом.
5. Отсортировать методом выбора список запросов к серверу (IP-адрес в формате XXX.XXX.XXX.XXX, где XXX – числа в диапазоне от 0 до 255; частота запросов в процентах – дробное число) по частоте запросов и по IP-адресу.
6. Отсортировать методом обмена список результатов замеров массы тела спортсмена (день, месяц, год взвешивания, результат взвешивания – дробное число) по массе и по дате взвешивания.
7. Отсортировать методом обмена список результатов замеров температуры тела больного (часы, минуты, температура – дробное число) по значению температуры и по времени измерения.
8. Отсортировать методом обмена список правильных многоугольников (количество сторон, длина стороны – дробное число) по площади и по периметру.

9. Отсортировать методом обмена список билетов в кинотеатр (ряд, место, стоимость – дробное число) по стоимости и по ряду с местом.
10. Отсортировать методом обмена список запросов к серверу (IP-адрес в формате XXX.XXX.XXX.XXX, где XXX – числа в диапазоне от 0 до 255; частота запросов в процентах – дробное число) по частоте запросов и по IP-адресу.

ЛИТЕРАТУРА

1. Буза, М.К. Архитектура компьютеров / М.К. Буза. – Минск: Новое знание, 2007. – 559 с.
2. Таненбаум, Э. Архитектура компьютеров: 5-е изд. / Э. Таненбаум. – СПб.: Питер, 2007. – 844 с.
3. Юров, В.И. Assembler. Учебник для вузов: 2-е изд. / В.И. Юров. – СПб.: Питер, 2003. – 624 с.
4. Юров, В.И. Assembler. Практикум: 2-е изд. / В.И. Юров. – СПб.: Питер, 2004. – 400 с.
5. Пирогов, В.Ю. Ассемблер. Учебный курс / В.Ю. Пирогов. – М.: Нолидж, 2001. – 848 с.
6. Patterson, D. Computer Architecture a Quantitative approach (3td edition) / D. Patterson, J. Hennesy. – San Francisco, 2003 – 978 с.

Учебное издание

ЕРМОЧЕНКО Сергей Александрович
СЕРГЕЕНКО Сергей Владимирович

ЯЗЫК АССЕМБЛЕРА ДЛЯ INTEL-СОВМЕСТИМЫХ ПРОЦЕССОРОВ С 32-БИТНОЙ АРХИТЕКТУРОЙ

Методические рекомендации
к выполнению лабораторных работ

Технический редактор *Г.В. Разбоева*
Компьютерный дизайн *Т.Е. Сафранкова*

Подписано в печать .2015. Формат 60x84 $^{1}/_{16}$. Бумага офсетная.
Усл. печ. л. 2,26. Уч.-изд. л. 1,16. Тираж экз. Заказ .

Издатель и полиграфическое исполнение – учреждение образования
«Витебский государственный университет имени П.М. Машерова».

Свидетельство о государственной регистрации в качестве издателя,
изготовителя, распространителя печатных изданий
№ 1/255 от 31.03.2014 г.

Отпечатано на ризографе учреждения образования
«Витебский государственный университет имени П.М. Машерова».
210038, г. Витебск, Московский проспект, 33.