Министерство образования Республики Беларусь
Учреждение образования «Витебский государственный
университет имени П.М. Машерова»
Кафедра прикладного
и системного программирования

**С.В. Сергеенко, С.А. Ермоченко**

# АРХИТЕКТУРА ПРОЦЕССОРОВ INTEL И ЯЗЫК АССЕМБЛЕРА

***Методические рекомендации***

*Витебск*
*ВГУ имени П.М. Машерова*
*2022*

УДК 004.431.1:004.23(075.8)
ББК 32.973.21я73+32.971.32я73
С32

Печатается по решению научно-методического совета учреждения образования «Витебский государственный университет имени П.М. Машерова». Протокол № 3 от 03.03.2022.

Авторы: старший преподаватель кафедры прикладного и системного программирования ВГУ имени П.М. Машерова **С.В. Сергеенко;** заведующий кафедрой прикладного и системного программирования ВГУ имени П.М. Машерова, кандидат физико-математических наук, доцент **С.А. Ермоченко**

Рецензент:
заведующий кафедрой информационных систем автоматизации производства УО «ВГТУ», кандидат технических наук, доцент *В.Е. Казаков*

**Сергеенко, С.В.**

**С32** Архитектура процессоров Intel и язык Ассемблера / С.В. Сергеенко, С.А. Ермоченко. – Витебск : ВГУ имени П.М. Машерова, 2022. – 46 с.

В методических рекомендациях представлен основной теоретический материал, описан ход выполнения лабораторных работ по различным темам, связанным с изучением 32-битной архитектуры процессоров Intel и совместимых с ними. Практически по всем темам приводятся 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 «Управление информационными ресурсами» (дисциплина «Архитектура вычислительных систем»), «Программное обеспечение информационных технологий» (дисциплины «Архитектура компьютерной техники и операционных систем», «Низкоуровневое программирование и трансляторы»), направлений специальностей «Прикладная информатика (программное обеспечение информационных систем)» (дисциплина «Машинно-ориентированное программирование»), «Прикладная математика (научно-педагогическая деятельность)» (дисциплина «Машинно-ориентированное программирование»).

**УДК 004.431.1:004.23(075.8)**
**ББК 32.973.21я73+32.971.32я73**

© Сергеенко С.В., Ермоченко С.А., 2022
© ВГУ имени П.М. Машерова, 2022

## СОДЕРЖАНИЕ

Введение .... 4
1. Архитектура 32-битных Intel-совместимых микропроцессоров .... 7
 1.1. Основные характеристики процессора .... 7
 1.2. Регистры общего назначения .... 8
 1.3. Организация памяти .... 9
 1.4. Сегментные регистры .... 10
 1.5. Указатель инструкций .... 11
 1.6. Регистр флагов EFLAGS .... 11
 1.7. Основные режимы работы процессора .... 12
2. Структура программы на языке Ассемблера .... 14
 2.1. Синтаксис языка Ассемблера .... 14
 2.2. Пример программы на языке Ассемблера .... 15
 2.3. Описание системных функций .... 16
 2.4. Самостоятельная работа .... 19
3. Арифметические операции в языке Ассемблера .... 23
 3.1. Пример ввода-вывода целых чисел в языке Ассемблера .... 23
 3.2. Описание стандартных функций Ассемблера MASM 32 .... 24
 3.3. Самостоятельная работа «Простое арифметическое выражение» .... 26
 3.4. Самостоятельная работа «Учёт флага CF» .... 26
4. Ветвления и циклы в языке Ассемблера .... 27
 4.1. Команды условного и безусловного перехода .... 27
 4.2. Самостоятельная работа «Ветвления в языке Ассемблера» .... 29
 4.3. Самостоятельная работа «Циклы в языке Ассемблера» .... 29
5. Подпрограммы и массивы в языке Ассемблера .... 30
 5.1. Пример использования подпрограмм .... 30
 5.2. Описание функции wsprintf .... 31
 5.3. Самостоятельная работа .... 33
6. Работа с сопроцессором на языке Ассемблера .... 34
 6.1. Пример работы с сопроцессором .... 34
 6.2. Самостоятельная работа «Арифметическое выражение» .... 36
 6.3. Самостоятельная работа «Массивы вещественных чисел» .... 37
7. Рекурсия в языке Ассемблера .... 38
 7.1. Самостоятельная работа .... 38
8. Косвенный вызов подпрограмм в языке Ассемблера .... 39
 8.1. Пример косвенного вызова .... 39
 8.2. Описание работы со структурами на языке Ассемблера .... 43
 8.3. Самостоятельная работа .... 44
Список литературы .... 45

# ВВЕДЕНИЕ

В данных методических рекомендациях приведены краткие теоретические сведения по языку Ассемблера и задания для Intel-процессоров 32-битной архитектуры и совместимых с ними на лабораторные работы по соответствующим темам. Краткие теоретические сведения даются 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 изучения 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 учебных программ по дисциплинам: «Архитектура вычислительных систем» специальности «Управление информационными ресурсами», «Архитектура компьютерной техники и операционных систем» специальности «Программное обеспечение информационных технологий», «Низкоуровневое программирование и трансляторы» специальности «Программное обеспечение информационных технологий», «Машинно-ориентированное программирование» направлений специальностей «Прикладная информатика (программное обеспечение информационных систем)» и «Прикладная математика (научно-педагогическая деятельность)».

Целями указанных дисциплин являются:

- изучение форматов представления различных данных в памяти компьютера, а также основ функционирования процессора и языка Ассемблера;
- систематизированная подготовка специалиста, владеющего технологиями создания и использования формализованных языков, предназначенных для описания программ и алгоритмов решения задач на ЭВМ;
- изучение основ составления программ на языках программирования низкого уровня для решения прикладных задач.

К задачам относятся:

- изучение форматов представления целых и вещественных чисел в памяти компьютера;
- изучение способов представления символов и строк в памяти;
- изучение способов представления массивов и структур в памяти компьютера;
- знакомство с основными принципами и алгоритмом работы процессора;
- изучение архитектуры процессора, включая понятие регистров и виды адресации;
- изучение основ синтаксиса языка Ассемблера и основных команд процессора;
- изучение принципов функционирования операционных систем;
- 
- приобретение знаний о концепциях, положенных в основу современных языков программирования;
- формирование навыков применения языков программирования низкого уровня для решения различных задач;
- изучение принципов формализации задач и их перевода на искусственный язык с целью реализации на ЭВМ;
- овладение методами разработки, тестирования, отладки и документирования программ с использованием современных инструментальных средств;
- изучение инструментов, используемых при написании и отладки кода на языке программирования низкого уровня.
- изучение основного состава архитектуры набора команд 32-битных Intel-совместимых процессоров;
- изучение способов представления основных алгоритмических конструкций средствами низкоуровневого языка программирования.

В результате изучения этих дисциплин формируются следующие компетенции:

– анализировать технические решения для формирования аппаратной и программной составляющих информационной инфраструктуры с учетом методов дискретной математики;

– строить и конфигурировать архитектуры вычислительных средств, основываясь на принципах функционирования операционных систем;

– проектировать компиляторы языков программирования;

– применять навыки построения, анализа и тестирования алгоритмов и программ для решения типовых задач прикладной математики, применять при проектировании приложений такие парадигмы программирования как структурное, объектно-ориентированное и функциональное программирование, а также иные парадигмы; разрабатывать программное обеспечение в интегрированных средах разработки, решать стандартные задачи профессиональной деятельности на основе применения информационно-коммуникационных технологий;

1. строить, анализировать и тестировать алгоритмы и программы решения типовых задач обработки информации с использованием структурного, объектно-ориентированного и иных парадигм программирования.

# 1. АРХИТЕКТУРА 32-БИТНЫХ INTEL-СОВМЕСТИМЫХ МИКРОПРОЦЕССОРОВ

## 1.1. Основные характеристики процессора

Любому приложению, выполняющемуся на 32-битном Intel-совместимом процессоре предоставляется набор ресурсов для выполнения инструкций и хранения кода, данных и информации о состоянии. Все эти ресурсы вместе образуют базовую среду выполнения.

Эта среда используется совместно как прикладными программами, так и операционной системой и (или) управляющей программой.

Базовая среда выполнения включает:

**Адресное пространство** – набор доступных адресов ячеек памяти. Адресуются отдельные байты. Программе может адресовать вплоть до 4 ГиБ ($2^{32}$ байт) памяти.

**Базовые регистры** – к ним относятся 8 регистров общего назначения, 6 сегментных регистра, регистр флагов и указатель инструкции. Регистры представляют собой набор внутренних высокопроизводительных ячеек памяти, предназначенных для хранения промежуточных результатов обработки данных или для управления работой процессора. Доступ к регистрам осуществляется быстрее чем к ячейкам основной (адресуемой) памяти.

**Регистры сопроцессора** – к ним относятся 8 регистров данных, регистры тегов, управления, состояния, указателя на инструкцию, указателя на данные и регистр кода операции. Эти регистры предоставляются для работы данными в формате числа с плавающей точкой одинарной, двойной и расширенной точности, 16-, 32- и 64-битными целыми числами, а также двоично-десятичными числами.

**Регистры расширений**.

**Стек** – предназначен для организации выполнения подпрограмм.

**Система команд процессора** – набор команд, которые может выполнять процессор.

В дополнение к ресурсом базовой среды выполнения архитектура 32-битных Intel-совместимых процессоров предоставляет дополнительные ресурсы системного уровня, предназначенные для поддержки операционных систем и системных программ. К системным ресурсам относятся:

Порты ввода-вывода.

Регистры управления.

Регистры управления памятью.

Отладочные регистры.

Диапазонные регистры типа памяти.

Специфичные для машины регистры.

Регистры проверки машины.

Счётчики производительности.

## 1.2. Регистры общего назначения

32-битные регистры общего назначения EAX, EBX, ECX, EDX, ESI, EDI, EBP, ESP предназначены для хранения операндов арифметических и логических операций, операндов вычисления адресов и адресов памяти.

Хотя все эти регистры доступны для хранения операндов, результатов и адресов, регистр ESP следует использовать с осторожностью, так как он содержит указатель на вершину стека и, как правило, не должен использоваться для хранения чего-либо ещё.

Кроме общего использования, каждый регистр имеет дополнительные особенности.

Регистр **EAX** – аккумулятор:

– обеспечивает наиболее эффективное выполнение большинства арифметических и логических операций;

– обязательно используется в некоторых арифметических командах (умножение, деление);

– обязательно используется при выполнении операций ввода-вывода;

– неявно используется при последовательной обработке цепочек элементов (ввод, вывод, поиск, заполнение и т. п.).

Регистр **EBX** – базовый регистр:

– обеспечивает наиболее эффективное вычисление адреса операнда при расширенной (базовой) адресации, например, при работе с массивами.

Регистр **EDX** – регистр данных:

– используется при организации ввода-вывода;

– обязательно используется в некоторых арифметических командах (умножение, деление).

Регистры **ESI**, **EDI** – индексные регистры:

– используются при выполнении цепочечных (последовательных) операций: копирование, заполнение, сравнение и т.п.

ESI – индекс источника

EDI – индекс назначения

При выполнении цепочечных операции регистры ESI, EDI автоматически изменяются на ±1.

Регистры **ESP**, **EBP** – указатели стека:

– используются при обращении к стеку

ESP – указатель вершины стека

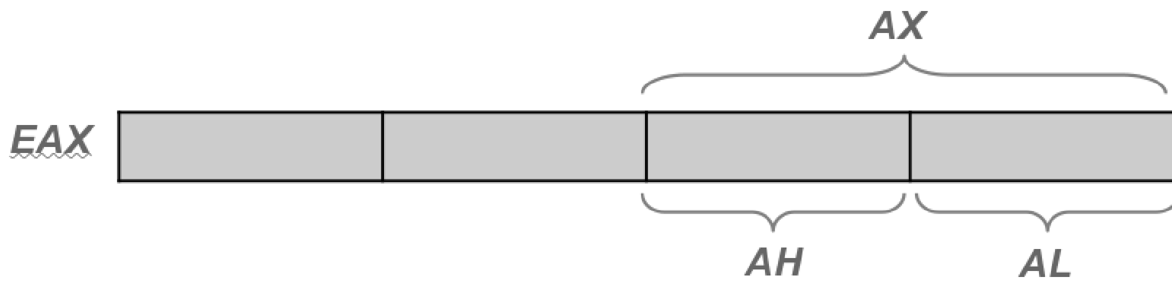
EBP – база записи активации.

Части регистров EAX, EBX, ECX, EDX имеют собственные имена, например,

AX – младшее слово регистра EAX;

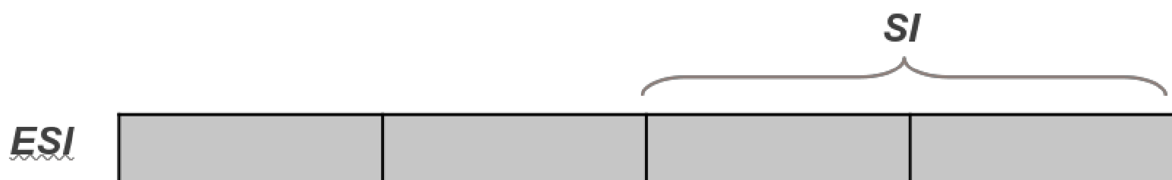
AL – младший байт слова AX;

AH – старший байт слова AX.




Части регистров ESI, EDI, ESP, EBP имеют собственные имена:

SI – младшее слово регистра ESI;
DI – младшее слово регистра EDI;
SP – младшее слово регистра ESP;
BP – младшее слово регистра EBP.




### 1.3. Организация памяти

Память к которой микропроцессор может обращаться задавая определенные сигналы на соответствующую шину называется *физической памятью*. Физическая память представляет собой последовательность 8-битных байтов. Каждому байте присвоен уникальный адрес, называемый *физическим адресом*. Физические адреса лежат в диапазоне от 0 до $2^{36}$–1.

Можно считать, что любая операционная система или управляющая программа, спроектированная для работы на 32-битных Intel-совместимых процессорах будет использовать для доступа к памяти соответствующие возможности такого процессора. К таким возможностям относятся сегментная и страничная организация памяти.

Когда задействованы возможности процессора по управлению памятью, программы не имеют прямого доступа к адресам физической памяти. Вместо этого они получают доступ к памяти с помощью одной из трёх моделей памяти:

**Плоская модель памяти** – память представляется программе как единое непрерывное адресное пространство. Оно называется *линейным адресным пространством*. Код, данные и стек находятся в этом пространстве. В этом пространстве каждый байт имеет свой адрес из диапазона от 0 до $2^{32}$–1, который называется *линейным адресом*.

**Сегментная модель памяти** – Память представляется для программы как набор независимых адресных пространств, называемых *сегментами*.

Код, данные и стек обычно содержаться в отдельных сегментах. При адресовании байта в сегменте программа использует *логический адрес*. Он состоит из селектора сегмента и смещения. Логические адреса часто называют *дальними указателями* (far pointer). Селектор сегмента задает сегмент, к которому будет осуществляться доступ, а смещение задает байт в адресном пространстве выбранного сегмента. Программы, работающие на 32-битном процессоре могут использовать до 16 383 сегментов различных типов и размеров, а каждый сегмент может быть до $2^{32}$ байт. Все определенные для системы сегменты отображаются на линейное адресное пространство процессора. Таким образом, для доступа к памяти процессор переводит логические адреса в линейные. Это преобразование происходит незаметно для программы.

**Модель памяти режима реальных адресов** – данная модель предоставляется для совместимости с процессорами Intel 8086.

При использовании плоской и сегментной моделей памяти линейное адресное пространство отображается на физическое адресное пространство процессора либо непосредственно, либо с помощью механизма страничной организации памяти. При прямом отображении каждый линейный адрес взаимно однозначно соответствует физическому. В этом случае линейные адреса устанавливаются на адресной шине процессора без преобразований.

При использовании страничной организации памяти линейные адреса разделяются на страницы, которые отображаются в виртуальную память. Страницы виртуальной памяти отображаются по мере надобности в физическую память. При использовании страничной организации памяти операционной системой или управляющей программой, этот механизм будет прозрачным для прикладных программ. Все что видит приложение – линейное адресное пространство.

### 1.4. Сегментные регистры

32-битные Intel-совместимыме процессоры имеют 6 сегментных регистров: CS, DS, ES, FS, GS, SS.

Эти регистры предназначены для организации хранения данных в различных сегментах памяти:

CS – содержит указание на сегмент кода (code segment);

DS – содержит указание на сегмент данных (data segment);

SS – содержит указание на сегмент стека (stack segment);

ES, FS, GS – содержат указания на дополнительные сегменты данных (обозначены буквами латинского алфавита, следующими после D).

## 1.5. Указатель инструкций

Регистр **EIP** называется *указателем инструкций*. Этот регистр содержит смещение в текущем сегменте кода очередной инструкции подлежащей выполнению. После выполнения одной инструкции автоматически заносится адрес следующей инструкции. Напрямую программе не доступен; для изменения последовательности выполнения инструкций используются специальные команды перехода. Единственный способ получить значение регистра EIP – выполнить инструкцию CALL и потом прочитать значение адреса возврата из стека.

## 1.6. Регистр флагов EFLAGS

32-битный регистр флагов EFLAGS содержит группу регистров состояния, управляющий флаг и группу системных флагов. После инициализации этот регистр содержит значение 0x00000002. Биты 1, 3, 5, 15 и с 22 до 31 зарезервированы. Программное обеспечение не должно использовать или полагаться на значения этих битов.

Некоторые из флагов в регистре EFLAGS могут быть изменены с использованием специальных инструкций. Однако, не существует инструкций которые бы позволяли изменять или изучать непосредственно весь регистр.

Флаги состояния (биты 0, 2, 4, 6, 7 и 11) регистра EFLAGS характеризуют результаты арифметических операций (например ADD, SUB, MUL, DIV). Эти флаги имеют следующий смысл:

Флаг переноса **CF** (Carry flag) хранится в бите 0 регистра EFLAGS – данный флаг устанавливается в значение 1, если операция привела к переносу или заимствованию за пределы наиболее значимого бита результата. В противном случае сбрасывается в значение 0. Данные флаг указывает на условия переполнения целочисленной арифметики без знака.

Флаг четности **PF** (Parity flag) хранится в бите 2 регистра EFLAGS – устанавливается, если младший (наименее значимый) байт результата содержит четное количество бит со значением 1. Сбрасывается в противном случае.

Флаг вспомогательного переноса **AF** (Auxiliary Carry flag) хранится в бите 4 регистра ELFAGS – устанавливается, если операция привела к переносу из или заимствованию в 3-й бит результата. Сбрасывается в противном случае. Данный флаг используется при работе с двоично-десятичными числами.

Флаг нуля **ZF** (Zero flag) ххранится в бите 6 регистра EFLAGS – ранится в бите 6 регистра EFLAGS – устанавливается в случае получения нулевого результата (значения всех бит равны 0). Сбрасывается в противном случае.

Флаг знака **SF** (Sign flag) хранится в бите 7 регистра EFLAGS – равен значению старшего (наиболее значимого) бита результата, который является знаковым битом числа со знаком.

Флаг переполнения **OF** (Overflow flag) хранится в бите 11 регистра EFLAGS – устанавливается в случае, если целочисленный результат будет слишком большим положительным или слишком маленьким отрицательным для хранения в операнде назначении числом. Сбрасывается в противном случае. Данный флаг указывает на состояние переполнения для арифметике целых чисел со знаком в дополнительном коде.

Из всех этих флагов состояния только для флага CF существуют инструкции для его непосредственного изменения.

Наличие флагов состояния позволяет одной и той же арифметической операции получать результаты трёх различных типов: числа без знака, числа со знаком и двоично-десятичные числа.

Управляющий флаг направления **DF** (Direction flag) хранится в бите 10 регистра EFLAGS влияет на действие цепочечных инструкций. Установка этого флага приводит к автоматическому уменьшению индексов операндов при выполнении цепочечных инструкций. Сброс флага приводит к тому, что эти инструкции будут увеличивать индексы операндов (значения регистров ESI и EDI).

### 1.7. Основные режимы работы процессора

32-битные Intel-совместимые процессоры поддерживают три основных режима работы:

– защищенный режим;

– режим реальных адресов;

– режим системного управления.

**Защищенный режим** является «родным» состоянием для процессора. Этот режим

– реализован в процессорах, начиная с i80286 (1982 год);

– позволяет использовать большее количество физической памяти – 16 Мб (i80286), 4 Гб (i80386 – Pentium), 64 Гб (Pentium Pro – …) и т.д.;

– поддерживает многозадачность (одновременное выполнение нескольких программ);

– позволяет защитить исполняемые процессором программы от взаимного влияния;

и т. д.

В защищенном режиме прямой доступ к физической памяти возможен только для программ, имеющих особые привилегии (например, операционная система).

Для остальных программ используется более сложный аппаратный механизм, осуществляющий вычисление адреса на основании специальных таблиц дескрипторов сегментов, содержащих информацию о том, какие сегменты находятся в распоряжении программы. Адрес вычисляется следующим образом:

– сегментный регистр содержит селектор – указатель на элемент таблицы дескрипторов;

– из данных этой таблицы извлекается адрес начала сегмента;

– к адресу начала сегмента прибавляется заданное в смещение.

**Режим реальных адресов (реальный режим)** реализует для программ окружение процессора Intel 8086 с некоторыми расширениями (например, возможностью переключиться в защищенный режим и режим системного управления). При включении компьютера процессор изначально находится в реальном режиме. В реальном режиме программа может напрямую обращаться к физической памяти, но при этом ей будем доступно не более 1 мегабайта ОЗУ.

В реальном режиме адрес вычисляется по следующей формуле

адрес = сегмент*16 + смещение

Поскольку величины сегмент и смещение могут принимать значения от 0 до 65535, то максимальный адрес ячейки памяти может быть 1114095.

Подобная схема расчета позволяла 16-разрядному процессору Intel 8086 использовать 20-разрядные адреса.

Величина сегмент берется из соответствующего сегментного регистра (код – CS, стек – SS, данные – DS, ES, FS, GS).

Величина смещение указывается непосредственно в команде или берется из какого-нибудь регистра.

**Режим системного управления** предоставляет операционной системе или управляющей программе прозрачный механизм реализации специфичных для конкретной платформы функций таких, как управление питанием и безопасность системы.

Среди возможностей защищенного режима следует отметить способность выполнять в этом режиме программный код предназначенный для режима реальных адресов процессора Intel 8086 в многозадачном окружении. Такая возможность называется **режимом виртуального процессора 8086**, хотя при этом не является отдельным режимом работы процессора. На самом деле это атрибут защищенного режима, который может быть включен на уровне отдельной задачи.

# 2. СТРУКТУРА ПРОГРАММЫ НА ЯЗЫКЕ АССЕМБЛЕРА

## 2.1. Синтаксис языка Ассемблера

*Ассемблер* – программа, используемая для преобразования исходной программы на языке Ассемблера в машинный код.

*Язык Ассемблера* – система обозначений, используемая для представления в удобочитаемой форме программ, записанных в машинном коде.

Инструкции на языке ассемблера разделены разрывами строк, и ожидается, что одна инструкция заполнит одну строку текста. Если строка содержит

точка с запятой, за исключением точек с запятой внутри строк в кавычках, остальная часть этой строки является комментарием, и компилятор его игнорирует. Если строка заканчивается символом "\" (в конце концов, за ней может следовать точка с запятой и комментарий), в этом месте присоединяется следующая строка.

Каждая строка в исходном коде представляет собой последовательность элементов, которые могут быть одного из трех типов. Один тип – это символы-символы, представляющие собой специальные символы, которые представляют собой отдельные элементы, даже если они не отделены друг от друга. Любой из "+-*/=<>()[]{}:,|&~#`" является символом символа. Последовательность других символов, отделенных от других элементов либо пробелами, либо символами, является символом. Если первым символом символа является либо одинарная, либо двойная кавычка, то он объединяет любую последовательность следующих за ней символов, даже специальных, в строку в кавычках, которая должна заканчиваться тем же символом, с которого она начиналась (одинарная или двойная цитата) – однако, если есть два таких символа подряд (без каких-либо других символов между ними), они интегрируются в строку в кавычках как только один из них, и тогда строка в кавычках продолжается. Символы, отличные от символов символов и строк в кавычках, могут использоваться в качестве имен, поэтому их также называют символами имени.

Каждая инструкция состоит из мнемоники и различного количества операндов, разделенных запятыми. Операнд может быть регистром, немедленным значением или данными, адресованными в памяти, ему также может предшествовать оператор размера, чтобы определить или переопределить его размер.

### 2.2. 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
```

### 2.3. 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 С++ 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.

## 2.4. Самостоятельная работа

На языке Ассемблера создать консольное приложение, которое считывает три введённых пользователем строки, усекает или дополняет про-

белами каждую строку до 10 символов, после чего выводит текст в соответствии с заданием Вашего варианта:

| № варианта | Шаблон |
|---|---|
| 1. | cccccccccc<br>aaaaaaaaaa<br>bbbbbbbbbb |
| 2. | aaaaaaaaaa<br>bbbbbbbbbb<br>cccccccccc |
| 3. | aaaaaaaaaa<br>bbbbbbbbbb<br>cccccccccc |
| 4. | aaaaaaaaaa<br>bbbbbbbbbb cccccccccc |
| 5. | aaaaaaaaaa<br>cccccccccc<br>bbbbbbbbbb |
| 6. | aaaaaaaaaa<br>bbbbbbbbbb<br>cccccccccc |

| | |
|---|---|
| 7. | aaaaaaaaaa ¤¤¤¤¤¤¤¤¤¤ ¤¤¤¤¤¤¤¤¤¤<br>¤¤¤¤¤¤¤¤¤¤ bbbbbbbbbb ¤¤¤¤¤¤¤¤¤¤<br>¤¤¤¤¤¤¤¤¤¤ ¤¤¤¤¤¤¤¤¤¤ cccccccccc |
| 8. | aaaaaaaaaa<br>bbbbbbbbbb<br>cccccccccc |
| 9. | aaaaaaaaaa<br>bbbbbbbbbb<br>cccccccccc |
| 10. | aaaaaaaaaa<br>bbbbbbbbbb<br>cccccccccc |

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 сим-

волы (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 ], " "
```

# 3. АРИФМЕТИЧЕСКИЕ ОПЕРАЦИИ В ЯЗЫКЕ АССЕМБЛЕРА

## 3.1. 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
```

### 3.2. 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 С++ 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 С++ 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 С++ 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.

### 3.3. 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).

### 3.4. 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`.

# 4. ВЕТВЛЕНИЯ И ЦИКЛЫ В ЯЗЫКЕ АССЕМБЛЕРА

## 4.1. Команды условного и безусловного перехода

«jmp» безоговорочно передает управление целевому местоположению. Адрес назначения может быть указан непосредственно в инструкции или косвенно через регистр или память, допустимый размер этого адреса зависит от того, ближний или дальний переход (его можно указать, предваряя операнд «ближним» или «дальним»). оператора) и является ли инструкция 16-битной или 32-битной. Операнд для ближнего перехода должен быть размером «слово» для 16-битной инструкции или размером «двойное слово» для 32-битной инструкции. Операнд для дальнего перехода должен иметь размер «dword» для 16-битной инструкции или размер «pword» для 32-битной инструкции. Прямая инструкция «jmp» включает в себя адрес назначения как часть инструкции (ей может предшествовать оператор «short», «near» или «far»), операнд, определяющий адрес, должен быть числовым выражением для ближнего или короткого перехода, или два числовых выражения, разделенных двоеточием для дальнего перехода, первое указывает селектор сегмента, второе – смещение внутри сегмента. Оператор «pword» может использоваться для принудительного 32-битного дальнего вызова, а «dword» – для принудительного 16-битного дальнего

вызова. Косвенная инструкция «jmp» получает адрес назначения косвенно через регистр или переменную-указатель, операндом должен быть общий регистр или память.

```
jmp 100h           ; прямой ближний прыжок
jmp 0FFFFh:0       ; прямой дальний прыжок
jmp ax             ; непрямой ближний прыжок
jmp pword [ebx]    ; непрямой дальний прыжок
```

Инструкции условной передачи – это переходы, которые могут передавать или не передавать управление в зависимости от состояния флагов ЦП во время выполнения инструкции. Мнемоника для условных переходов может быть получена путем присоединения мнемоники условия к мнемонике «j», например, инструкция «jc» передаст управление, когда установлен флаг CF. Условные переходы могут быть короткими или близкими, и только прямыми, и могут быть оптимизированы, операнд должен быть немедленным значением, указывающим целевой адрес.

Инструкции «loop» – это условные переходы, которые используют значение, помещенное в CX (или ECX), чтобы указать количество повторений программного цикла. Все инструкции «цикла» автоматически уменьшают CX (или ECX) и завершают цикл (не передают управление), когда CX (или ECX) равен нулю. Он использует CX или ECX независимо от того, является ли текущая настройка кода 16-битной или 32-битной, но его можно заставить использовать CX с мнемоникой «loopw» или использовать ECX с мнемоникой «loopd». «loope» и «loopz» – синонимы одной и той же инструкции, которая действует как стандартная «loop», но также завершает цикл, когда установлен флаг ZF. Мнемоники «loopew» и «loopzw» заставляют их использовать регистр CX, а «looped» и «loopzd» заставляют их использовать регистр ECX. «loopne» и «loopnz» – синонимы одной и той же инструкции, которая действует как стандартный «loop», но также завершает цикл, когда флаг ZF не установлен. Мнемоники «loopnew» и «loopnzw» заставляют их использовать регистр CX, а «loopned» и «loopnzd» заставляют их использовать регистр ECX. Каждая инструкция «loop» нуждается в операнде, представляющем собой непосредственное значение, задающее целевой адрес, это может быть только короткий переход (в диапазоне 128 байт назад и 127 байт вперед от адреса инструкции, следующей за инструкцией «loop»).

Инструкция «jcxz» переходит к метке, указанной в инструкции, если находит значение нуля в CX, «jecxz» делает то же самое, но проверяет значение ECX вместо CX. Правила для операндов такие же, как и для инструкции «loop».

## 4.2. Самостоятельная работа «Ветвления в языке Ассемблера»

Разработайте приложение, вычисляющее значение выражения, соответствующего варианту.

Варианты задания:

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

## 4.3. 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.

   Найдите факториал целого неотрицательного числа.

# 5. ПОДПРОГРАММЫ И МАССИВЫ В ЯЗЫКЕ АССЕМБЛЕРА

## 5.1. 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
```

## 5.2. 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.

### 5.3. 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 |

# 6. РАБОТА С СОПРОЦЕССОРОМ НА ЯЗЫКЕ АССЕМБЛЕРА

## 6.1. 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.

## 6.2. 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.

## 6.3. Самостоятельная работа «Массивы вещественных чисел»

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

Варианты задания:

| № варианта | Функция |
|---|---|
| 1. | $y = \begin{cases} \sqrt{-ax + a}, & x < 1 \\ b \ln x, & x \ge 1 \end{cases}$ |
| 2. | $y = \begin{cases} \frac{b}{1 + x^2}, & x \le 0 \\ b \cos ax, & x > 0 \end{cases}$ |
| 3. | $y = \begin{cases} ax^2 + b, & x \le 0 \\ b - a^x + 1, & x > 0 \end{cases}$ |
| 4. | $y = \begin{cases} \frac{1}{a^x}, & x \le 0 \\ \cos bx, & x > 0 \end{cases}$ |
| 5. | $y = \begin{cases} a\sqrt{1 - x}, & x \le 1 \\ b \log_3 x, & x > 1 \end{cases}$ |
| 6. | $y = \begin{cases} \ln\left(\sin ax + \frac{\pi}{2}\right), & x \le 0 \\ \ln\left(\pi + bx^2\right) - \ln \frac{2}{1 + x}, & x > 0 \end{cases}$ |
| 7. | $y = \begin{cases} a \cos x, & x \le 0 \\ a - b^x, & x > 0 \end{cases}$ |
| 8. | $y = \begin{cases} \sqrt{a\lvert x + 1\rvert}, & x \le -1 \\ \frac{b}{\sqrt{2}} - \sqrt{\frac{b^2}{1 + x^2}}, & x > -1 \end{cases}$ |
| 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. | $y = \begin{cases} ax - bx^2, & x < 1 \\ \frac{a - b}{\ln\left(e + \sqrt{x^2 - 1}\right)}, & x \ge 1 \end{cases}$ |

*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, со-

держащую 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.

## 7. Рекурсия в языке Ассемблера

### 7.1. 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 $n$.

# 8. Косвенный вызов подпрограмм в языке Ассемблера

## 8.1. 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
```

## 8.2. 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).

### 8.3. 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.

## СПИСОК ЛИТЕРАТУРЫ

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

Учебное издание

**СЕРГЕЕНКО** Сергей Владимирович
**ЕРМОЧЕНКО** Сергей Александрович

# АРХИТЕКТУРА ПРОЦЕССОРОВ INTEL И ЯЗЫК АССЕМБЛЕРА

Методические рекомендации


| | |
|---|---|
| Технический редактор | *Г.В. Разбоева* |
| Компьютерный дизайн | *А.В. Табанюхова* |

Подписано в печать 19.08.2022. Формат 60x84 $^{1}/_{16}$ . Бумага офсетная.
Усл. печ. л. 2,67. Уч.-изд. л. 1,74. Тираж 40 экз. Заказ 139.

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.

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