

МИНИСТЕРСТВО СЕЛЬСКОГО ХОЗЯЙСТВА РОССИЙСКОЙ ФЕДЕРАЦИИ  
ФГБОУ ВПО «Кубанский государственный аграрный университет»

Факультет прикладной информатики  
*Кафедра компьютерных технологий и систем*

**А. В. Параскевов, Д. Ю. Жмурко, В. И. Лойко, С. А. Курносков**

## **МИКРОПРОЦЕССОРЫ**

### **Лабораторный практикум**

По специальности «Прикладная информатика» и  
«Информационные системы и технологии»

Краснодар  
2013

**УДК 004.31 (076.5)**  
**ББК 32.973.2**  
**П 18**

**Р е ц е н з е н т ы:**

**Барановская Т. П.** – доктор экономических наук, профессор  
(Кубанского государственного аграрного университета)

**Параскевов А. В., Жмурко Д. Ю., Лойко В. И., Курносов С. А.**

**П 18** Микропроцессоры: лабораторный практикум (по специальности «Прикладная информатика» и «Информационные системы и технологии»). / А. В. Параскевов, Д. Ю. Жмурко, В. И. Лойко, С. А. Курносов – Краснодар: КубГАУ, 2013. – 71 с.

Лабораторный практикум по дисциплине "Микропроцессоры" подготовлен для специальности «Прикладная информатика» и «Информационные системы и технологии» разработан на кафедре компьютерных технологий и систем (КТС) факультета прикладной информатики (ФПИ) КубГАУ. Он соответствует требованиями государственного образовательного стандарта (ГОС) высшего профессионального образования по направлению «Прикладная информатика», утвержденного 14.03.2000г. (регистрационный № 52 мжд/сп) Министерством образования РФ.

Рассмотрено и рекомендовано к изданию на заседании кафедры компьютерных технологий и систем КубГАУ 12 февраля 2013 года, протокол № 2.

Рекомендован к печати Советом факультета прикладной информатики Кубанского государственного аграрного университета 24 февраля 2013 года, протокол № 18.

Практикум представляет собой великолепное практическое руководство по основам программирования на языке ассемблера. Изложение сопровождается подробно откомментированными примерами, что способствует наилучшему пониманию и усвоению материала. Доходчиво объясняются все основные вопросы программирования на этом языке.

Вы узнаете, как писать ассемблерные программы под разные операционные системы (Windows, DOS), как создавать резидентные программы, как писать ассемблерные вставки в программы на языках высокого уровня и многое другое. Попутно вам будут разъяснены основные моменты работы процессора, операционных систем, управления памятью и взаимодействия программ с аппаратными устройствами ПК – то есть все то, без знания чего нельзя обойтись при программировании на языке низкого уровня, которым и является ассемблер.

**УДК 004.31 (076.5)**  
**ББК 32.973.2**

© А. В. Параскевов, Д. Ю. Жмурко, 2013  
© ФГБОУ «Кубанский государственный аграрный университет», 2013

## ОГЛАВЛЕНИЕ

I ЦЕЛИ И ЗАДАЧИ ДИСЦИПЛИНЫ .....	4
II ТРЕБОВАНИЯ К ФОРМИРУЕМЫМ КОМПЕТЕНЦИЯМ .....	4
III ПЛАН ЛАБОРАТОРНЫХ ЗАНЯТИЙ .....	6
IV ПРОГРАММА САМОСТОЯТЕЛЬНОЙ РАБОТЫ СТУДЕНТОВ .....	7
V ЛАБОРАТОРНЫЕ ЗАДАНИЯ .....	8
Лабораторная работа №1 .....	8
Лабораторная работа №2 .....	13
Лабораторная работа №3 .....	21
Лабораторная работа №4 .....	25
Лабораторная работа №5 .....	30
Лабораторная работа №6 .....	35
Лабораторная работа №7 .....	38
Лабораторная работа №8 .....	42
Лабораторная работа №9 .....	45
Лабораторная работа №10 .....	47
VI ТЕСТОВЫЕ ЗАДАНИЯ.....	48
VII ВОПРОСЫ ДЛЯ ПОДГОТОВКИ К ЗАЧЕТУ.....	62
ГЛОССАРИЙ.....	64
СПИСОК ЛИТЕРАТУРЫ.....	68
П Р И Л О Ж Е Н И Я .....	69

## I ЦЕЛИ И ЗАДАЧИ ДИСЦИПЛИНЫ

Цель дисциплины – обеспечить базовую подготовку студентов в области применения языка ассемблера для процессоров семейства x86; основы функционирования микропроцессоров; применения микропроцессоров для построения вычислительных систем; дать студентам представление о структуре и работе микропроцессора, изучить основы языка ассемблера для процессоров семейства x86, подготовить студентов к использованию языков низкого уровня в системном программировании.

В результате изучения дисциплины студент должен:

Знать:

- архитектуру микропроцессоров семейства x86;
- организацию памяти микропроцессора;
- жизненный цикл программ на языке ассемблера.

Уметь:

- применять команды языка ассемблера для реализации алгоритмов работы со сложными структурами данных;
- реализовывать Windows-приложения на ассемблере.

Владеть:

- навыками практической работы с транслятором, компоновщиком и отладчиком.

Иметь представление:

- реализации команд работы с микропроцессором в защищенном режиме.

Виды и задачи профессиональной деятельности по дисциплине:

разработка средств реализации информационных технологий;

сборка программной системы из готовых компонентов;

адаптация приложений к изменяющимся условиям функционирования

Данная дисциплина является вариативной частью профессионального учебного цикла БЗ ООП.

Для успешного освоения дисциплины необходимы знания по следующим дисциплинам и разделам ООП:

- Информатика.

## II ТРЕБОВАНИЯ К ФОРМИРУЕМЫМ КОМПЕТЕНЦИЯМ

Процесс изучения дисциплины направлен на формирование следующих компетенций:

б) профессиональные (ПК):

- способность разрабатывать средства реализации информационных технологий (методические, информационные, математические, алгоритмические, технические и программные) (ПК-12);

- способность использовать технологии разработки объектов профессиональной деятельности, в областях: машиностроение, приборостроение, наука, техника, образование, медицина, административное управление, юриспруденция, бизнес, предпринимательство, коммерция, менеджмент, банковские системы, безопасность информационных систем, управление технологическими процессами, механика, техническая физика, энергетика, ядерная энергетика, силовая электроника, металлургия, строительство, транспорт, железнодорожный транспорт, связь, телекоммуникации, управление инфокоммуникациями, почтовая связь, химическая промышленность, сельское хозяйство, текстильная и легкая промышленность, пищевая промышленность, медицинские и биотехнологии, горное дело, обеспечение безопасности подземных предприятий и производств, геология, нефтегазовая отрасль, геодезия и картография, геоинформационные системы, лесной комплекс, химико-лесной комплекс, экология, сфера сервиса, системы массовой информации, дизайн, медиаиндустрия, а также предприятия различного профиля и все виды деятельности в условиях экономики информационного общества (ПК-18);

- способность осуществлять организацию рабочих мест, их техническое оснащение, размещение компьютерного оборудования (ПК-19);

- способность участвовать в постановке и проведении экспериментальных исследований (ПК-24);

- способность к инсталляции, отладке программных и настройке технических средств для ввода информационных систем в опытную эксплуатацию (ПК-29);

- готовность проводить сборку информационной системы из готовых компонентов (ПК-30);

- способность к осуществлению инсталляции, отладки программных и настройки технических средств для ввода информационных систем в промышленную эксплуатацию (ПК-31).

### III ПЛАН ЛАБОРАТОРНЫХ ЗАНЯТИЙ

Таблица 1

№ темы лекции	Наименование и № лабораторной работы
1	Лабораторная работа №1. Этапы разработки программы на языке ассемблера. Основы работы с отладчиком OllyDbg. Форматы исполняемых файлов.
1	Лабораторная работа №1. Этапы разработки программы на языке ассемблера. Основы работы с отладчиком OllyDbg. Форматы исполняемых файлов.
3	Лабораторная работа №2. Директивы определения данных и представления данных в памяти. Команды пересылки данных. Основы вызова функций WinAPI.
3	Лабораторная работа №2. Директивы определения данных и представления данных в памяти. Команды пересылки данных. Основы вызова функций WinAPI.
5	Лабораторная работа №3. Функция форматирования вывода. Команды передачи управления (условные и безусловные переходы).
5	Лабораторная работа №3. Функция форматирования вывода. Команды передачи управления (условные и безусловные переходы).
8	Лабораторная работа №4. Команды двоичной арифметики. Форматы целых чисел в ассемблере. BCD-числа.
8	Лабораторная работа №4. Команды двоичной арифметики. Форматы целых чисел в ассемблере. BCD-числа.
9	Лабораторная работа №5. Основы работы с числами с плавающей точкой (ЧПТ). Работа с математическим сопроцессором.
9	Лабораторная работа №5. Основы работы с числами с плавающей точкой (ЧПТ). Работа с математическим сопроцессором.
10	Лабораторная работа №6. Директивы управления потоком. Основы работы с массивами.
10	Лабораторная работа №7. Директивы управления потоком. Основы работы с массивами.
10	Лабораторная работа №7. Директивы управления потоком. Основы работы с массивами.
14	Лабораторная работа №8. Основы работы с процедурами. Использование процедур.

№ темы лекции	Наименование и № лабораторной работы
14	Лабораторная работа №8. Основы работы с процедурами. Использование процедур.
14	Лабораторная работа №9. Основы работы с процедурами. Использование процедур.
14	Лабораторная работа №10. Основы работы с процедурами. Использование процедур.

#### IV ПРОГРАММА САМОСТОЯТЕЛЬНОЙ РАБОТЫ СТУДЕНТОВ

Таблица 2

№ темы лекции	Форма самостоятельной работы	Форма контроля
1, 5, 13, 14	проработка вопросов, вынесенных на самостоятельное изучение, с использованием основной и дополнительной литературы	устный опрос на лабораторных занятиях
2, 3, 4, 6	подготовка рефератов	доклад
15,17	конспектирование материалов, работа со справочной литературой	ответ во время устного или письменного опроса

## V ЛАБОРАТОРНЫЕ ЗАДАНИЯ

### *Лабораторная работа №1*

#### **Основные команды Ассемблера**

##### *Цель работы:*

1. Изучить этапы разработки программы на языке АССЕМБЛЕРА<sup>1</sup>;
2. Изучить основы работы с отладчиком OllyDbg.

##### *Описание работы:*

- I. Разработка программы на языке ассемблера состоит из трех этапов:
- а) разработка алгоритма программы и запись его на языке ассемблера;
  - б) трансляция исходного текста в машинный код и компоновка;
  - в) отладка программы.

Текст программы на языке ассемблера может быть набран в любом текстовом редакторе.

Важно использовать формат файла «.TXT» или подобный ему, без специального форматирования. Нельзя использовать форматы «.DOC» или «.RTF». Предпочтительно набирать текст программы во встроенных редакторах NC, FAR, Total Commander и пр. Файл должен быть сохранен с расширением «.ASM».

В листинге 1 приведен исходный текст программы с комментариями. Наберите программу в редакторе и сохраните ее под именем lab01\_1.asm в C:\WORK.

Листинг 1.

---

<sup>1</sup> Ассемблер (от англ. *assembler* — сборщик) — компьютерная программа, компилятор исходного текста программы, написанной на языке ассемблера, в программу на машинном языке. Как и сам язык (ассемблера), ассемблеры, как правило, специфичны для конкретной архитектуры, операционной системы и варианта синтаксиса языка. Вместе с тем существуют мультиплатформенные или вовсе универсальные (точнее, ограниченно-универсальные, потому что на языке низкого уровня нельзя написать аппаратно-независимые программы) ассемблеры, которые могут работать на разных платформах и операционных системах. Среди последних можно также выделить группу *кросс-ассемблеров*, способных собирать машинный код и исполняемые модули (файлы) для других архитектур и ОС.



```

.386                ; используются регистры и
                    ; команды i386

.model flat,stdcall ; плоская модель памяти
.code              ; начало сегмента кода
start:            ; точка входа в программу
mov eax, 2        ; занести 2 в eax
add eax, 20       ; добавить к eax 20
ret               ; вернуться в ОС
end start          ; завершение программы

```

Транслируйте программу командой `ml /c /coff lab01_1.asm`. Если текст программы набран правильно, вы увидите примерно следующее сообщение:

```

Microsoft (R) Macro Assembler Version 6.14.8444
Copyright (C) Microsoft Corp 1981-1997.
All rights reserved.

```

```

Assembling: lab01_1.asm

```

Если в программе есть ошибки, они будут указаны в формате «имя\_файла.asm(номер\_строки): номер\_ошибки: Описание\_ошибки».

В результате работы ассемблера будет получен объектный файл `lab01_1.obj`. Для того чтобы получить исполняемый файл нужно использовать компоновщик `link`. Результатом работы компоновщика является создание исполняемого файла с расширением «.EXE» или «.COM». Для создания исполняемого файла выполните команду «`link /SUBSYSTEM:CONSOLE lab01_1.obj`». Вы увидите примерно следующее сообщение:

```

Microsoft (R) Incremental Linker Version 5.12.8078 Copyright (C) Mi-
crosoft Corp 1992-1998.
All rights reserved.

```

В каталоге появится файл `lab01_1.exe`. Запустите полученный исполняемый файл командой `lab01_1`. Если все сделано правильно, на экране ничего не будет отображено и вы снова увидите приглашение командной строки.

II. Для отладки программы будет использоваться отладчик OllyDbg<sup>2</sup> (см. рисунок 1). Загрузите программу в отладчик командой `ollydbg lab01_1`.

1 – окно с исходной программой в дизассемблированном виде. Пошаговую отладку можно производить прямо в этом окне; строка с текущей командой подсвечивается;

2 – окно регистров микропроцессора, отражающее текущее содержимое регистров. Обратите внимание на регистр флагов (сверху вниз под регистрами общего назначения);

3 – окно дампа оперативной памяти, отражающее содержимое области памяти по адресу, который формируется из компонентов, указанных в левой части окна. В окне можно увидеть содержимое произвольной области памяти, а также выбрать тип отображения содержимого памяти;

4 – окно стека, отражающее содержимое памяти, выделенной для стека. Адрес области стека определяется содержимым регистров SS и ESP.

Для пошагового выполнения программы используйте клавишу «F8». Если в программе встречаются команды перехода в процедуры или прерывания и нужно проследить их выполнение по шагам, используется клавиша «F7». В нашем случае удобнее использовать «F8».

Для консольного приложения сразу создается окно, в котором вы сможете увидеть результаты работы (если используется вывод) или ввести данные. Выполните программу в пошаговом режиме и убедитесь в правильности ее работы (проконтролировать по состоянию регистров).

Наберите программу из листинга 2 и сохраните ее под именем `lab01_2.asm` в `C:\WORK`.

---

<sup>2</sup> OllyDbg — бесплатный проприетарный 32-битный отладчик уровня ассемблера для операционных систем Windows, предназначенный для анализа и модификации откомпилированных исполняемых файлов и библиотек, работающих в режиме пользователя (ring-3).

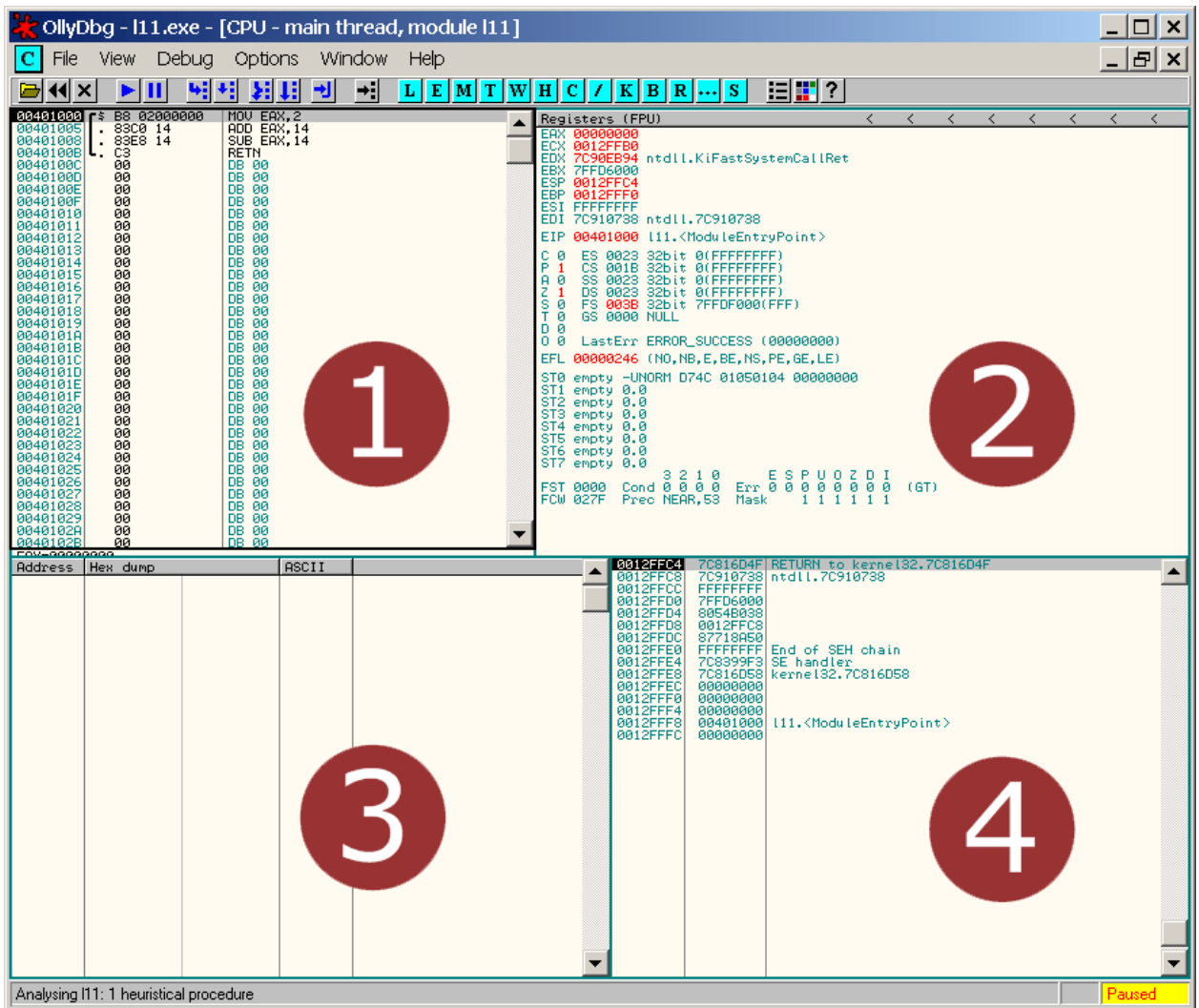


Рисунок 1 – Вид программы OllyDbg

## Листинг 2

```
.386

.model flat,stdcall
option casemap:none
include c:\windows\include\kernel32.inc
includelib c:\windows\lib\kernel32.lib

.data
stdout dd ?
msg db "Hello, world!",0dh,0ah
cWritten dd ?

.code
start:
invoke GetStdHandle, -11
mov stdout,eax
```

```
invoke WriteConsoleA, stdout, ADDR msg,\ ; символ\ мож-  
но использовать для переноса строки  
    sizeof msg, ADDR cWritten, 0  
    invoke ExitProcess, 0  
end start
```

Создайте исполняемый файл и загрузите его в отладчик. Выполните программу в отладчике. Вернитесь к началу, выполнив сброс (Ctrl+F2).

Выполните программу в пошаговом режиме.

### **Индивидуальные задания**

1. Проанализируйте принцип работы программы.
2. Прокомментируйте каждую строку листинга 2.

### **Указания к оформлению отчета**

Отчет оформляется в виде текстового файла и должен содержать:

- · фамилию, имя, группу и вариант студента;
- · номер задания, ответ на задание (если требуется);
- · комментарий к выполнению задания (если требуется).

## Лабораторная работа №2

### Арифметические операции

*Цель работы:*

1. Изучить арифметические операции ассемблера.

Программирование арифметических выражений в языке Ассемблер происходит через некоторые команды, такие как: mul, div, sub, add. Эти команды называются командами арифметических операций.

Mul – команда умножения. Она умножает регистр ax на тот операнд, что стоит после него. Результат заносится в ax.

Div – команда деления. Делит регистр ax на операнд, находящийся после него. Результат заносится в ax.

Add – команда сложения. Складывает два числа. Результат заносится в первый регистр.

Sub – команда вычитания. Вычитает два числа. Результат заносится в первый регистр.

#### **Пример:**

Написать программу на ассемблере вычисления выражения:

$$a - e / b - d \cdot e$$

где:

$$a = 5$$

$$b = 27$$

$$c = 86$$

$$e = 1986$$

$$d = 1112$$

Листинг программы:

```
.686          ; директива определения системы команд микропроцессора  
.model flat,stdcall ; задание линейной модели памяти
```

```
.data          ; директива определения данных  
_a dw 5 ;запись в 16-разрядную константу памяти с именем _a числа 5  
_b dw 27  
_c dw 86  
_e dw 1986
```

```

_d dw 1112
res dw 0      ; резервирование памяти для сохранения переменной res

.code        ; директива начала сегмента команд
start:
mov edx,0    ; очистка регистров
mov ebx,0    ; очистка регистров
mov ecx,0    ; очистка регистров
mov ax,_e    ; в регистр ax заносим число _e
mul _d       ; множим _e и _d
SHL edx,16   ; делаем сдвиг на 16
mov dx,ax
push edx     ; бросаем значение в стек
mov edx,0
mov ax,_e
mov cx,_b
div cx       ; делим ax на cx
pop ecx      ; достаем из стека значения
sub ecx,eax  ; отнимаем
mov ax,_a
sub eax,ecx
mov res, eax
ret          ; возвращение управление ОС
end start    ; окончание программы с именем _start

```

Результат работы программы показаны на рисунке 2.

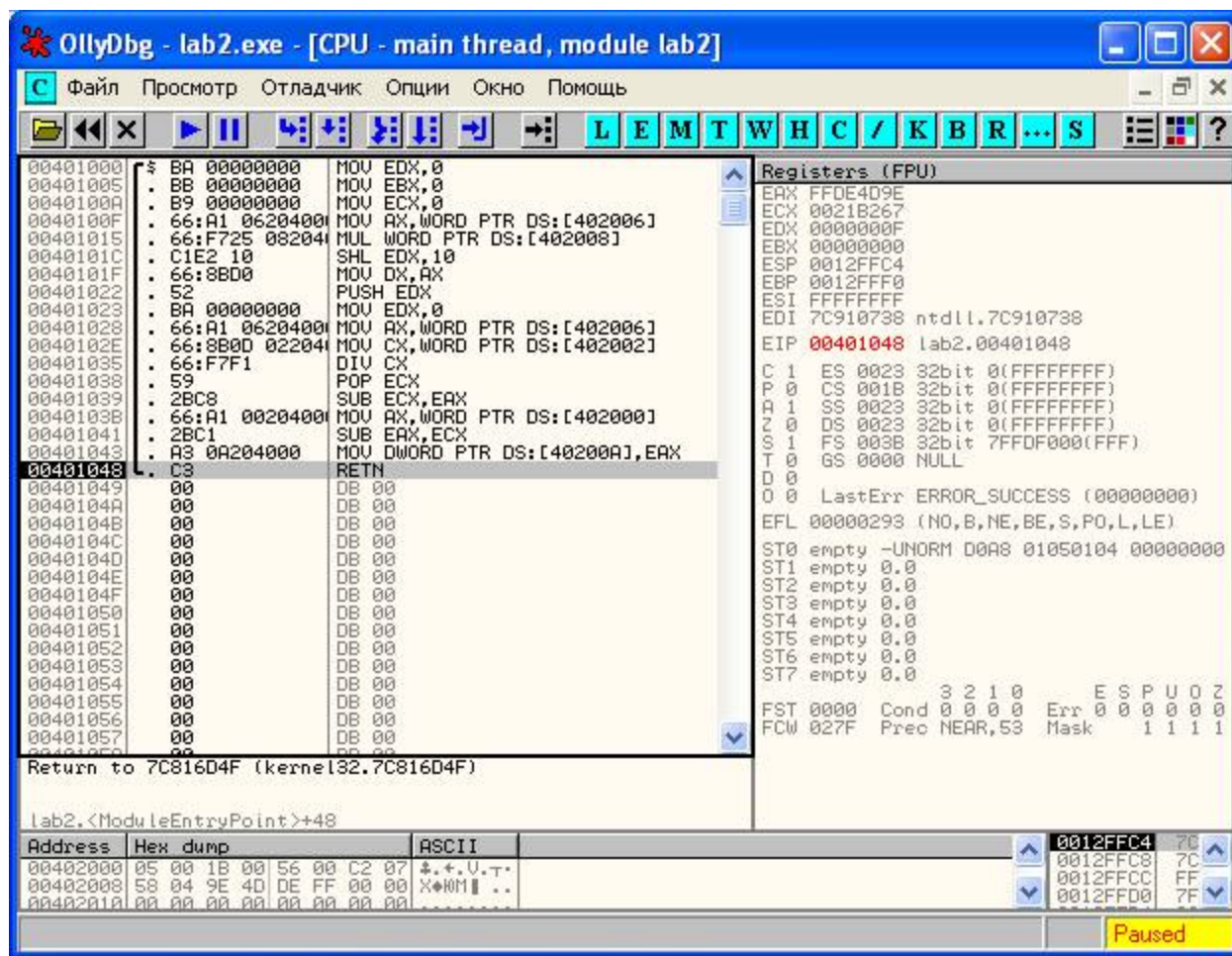


Рисунок 2 – Реализация программы в OllyDbg

### Индивидуальные задания

1.	$y = \begin{cases} ax^2 + bx +  c , & \text{если }  x  > 5; \\ ax + b, & \text{если }  x  = 5; \\ x^2(a - b) -  c , & \text{если }  x  < 5. \end{cases}$
2.	$y = \begin{cases} ax + bz, & \text{если } \min(a, b) < 0; \\ \frac{a}{x} + \frac{b}{z}, & \text{если } \min(a, b) \geq 0. \end{cases}$
3.	$y = \begin{cases} \frac{ax+5}{ax-5}, & \text{если }  ax  > 5; \\ ax^2 + 5x, & \text{если }  ax  = 5; \\ 5x^2 + ax, & \text{если }  ax  < 5; \end{cases}$

4.	$y = \begin{cases} \frac{abc}{x}, & \text{если }  x  > 7; \\ 7a^2 + bc, & \text{если }  x  = 7; \\ \frac{a+b+c}{x+4}, & \text{если }  x  < 7. \end{cases}$
5.	$y = \begin{cases} \frac{3b+b^2-5}{b+3}, & \text{если } b \in (-\infty; -5]; \\ 3, & \text{если } b \in (-5; 0]; \\ \frac{3b-b^2+5}{b+3}, & \text{если } b \in (0; \infty). \end{cases}$
6.	$y = \begin{cases} -x^2 + 4, & \text{если } x < 0; \\ x^2 + 4, & \text{если } 0 \leq x < 2; \\ 8, & \text{если } x \geq 2. \end{cases}$
7.	$y = \begin{cases} \frac{a^2 + 4x + 2}{b}, & \text{если }  x  < 5; \\ (a^2 + 17), & \text{если } x \geq 5; \\ (a^2 + 4x - 2)b, & \text{если } x \leq 5. \end{cases}$
8.	$y = \begin{cases} \left[ \frac{a^2}{b-2} \right] - 14(c+4x), & \text{если }  b  \neq  c ; \\ 0, & \text{если }  b  =  c . \end{cases}$
9.	$y = \begin{cases} \left[ \frac{a+2}{c} \right], & \text{если } x < -10; \\ 3(a^2 + c), & \text{если } -10 \leq x < 13; \\ a^3, & \text{если } x \geq 13. \end{cases}$
10.	$y = \begin{cases} \frac{100}{x^2 + ab}, & \text{если } x \leq -10; \\ (x^2 + ab)^2 - \frac{a}{b}, & \text{если } -10 < x < 0; \\ \frac{x^2 + ab - b^3}{a + bx}, & \text{если } x \geq 0. \end{cases}$
11.	$y = \begin{cases} \left[ \frac{ay + bz}{ x } \right], & \text{если } x < 0; \\ a^3 + 4b, & \text{если } x = 0; \\ \left[ \frac{a}{x} \right] + \left\{ \frac{y}{b} \right\}, & \text{если } x > 0. \end{cases}$



12.	Переменной $k$ присвоить номер четверти плоскости, в которой расположена точка с координатами $(x, y)$ . ( $xy \neq 0$ )
13.	$y = \begin{cases} a^2 + ax - bc, & \text{если } \max(a, b, c) > 5; \\ abc + \frac{(b+c)x}{a}, & \text{если } \max(a, b, c) \leq 5. \end{cases}$
14.	$y = \begin{cases} \left\lceil \frac{ax}{b} \right\rceil, & \text{если } x < -5; \\ ax^2 + b, & \text{если }  x  \leq 5; \\ \left\lfloor \frac{ax^3}{b} \right\rfloor, & \text{если } x > 5. \end{cases}$
15.	$y = \begin{cases} x^2, & \text{если } (x-2)^2 < 4; \\ 16 + x, & \text{если } (x-2)^2 = 4; \\ ax^2 + 5x + 4, & \text{если } (x-2)^2 > 4. \end{cases}$
16.	$y = \begin{cases} \frac{(x+a)^2}{b} - \frac{c}{a^2}, & \text{если }  x  \leq 10; \\ \frac{(x+b)^2}{a} + \frac{c}{a^2}, & \text{если }  x  > 10. \end{cases}$
17.	$y = \begin{cases} ( a  -  b )^2 x, & \text{если } x^3 < 64; \\ 3x^3, & \text{если } x^3 = 64; \\ \left\lfloor \frac{ab+3}{2} \right\rfloor, & \text{если } x^3 > 64. \end{cases}$
18.	$y = \begin{cases} \max(a, b, c), & \text{если }  a+b  > c; \\ \frac{abc}{ a+b +2}, & \text{если }  a+b  = c; \\ \min(a, b, c), & \text{если }  a+b  < c. \end{cases}$
19.	$y = \begin{cases} \frac{mx+n}{2}, & \text{если } x < m, \\ n^2 + m^2, & \text{если } x = m, \\ \frac{(n-m)^2}{x}, & \text{если } x > m \end{cases}$

20.	$y = \begin{cases} mx + nz, & \text{если }  x + z  > 5; \\ 25xz, & \text{если }  x + z  = 5; \\ \frac{mx - nz}{ x + z  + 1}, & \text{если }  x + z  < 5. \end{cases}$
21.	$y = \begin{cases} ax^2 + bx + c, & \text{если }  x  \leq 2; \\ a + b + c, & \text{если } x < -2; \\ ax + bc, & \text{если } x > 2. \end{cases}$
22.	$y = \begin{cases} \max(x, y), & \text{если } xy > 0; \\ 5 +  x + y , & \text{если } xy = 0; \\ \min(x, y), & \text{если } xy < 0. \end{cases}$
23.	$y = \begin{cases} \frac{mx + n}{mx - n}, & \text{если } mx > n; \\ (mx + n)^2, & \text{если } mx = n; \\ mx^2 - nx, & \text{если } mx < n. \end{cases}$
24.	$y = \begin{cases} \min(a, b, c), & \text{если } x \leq 0; \\ (a + b + c)x, & \text{если } x > 0. \end{cases}$
25.	$y = \begin{cases} \left[ \frac{ab}{c} \right], & \text{если }  a  +  b  >  c ; \\ abc, & \text{если }  a  +  b  =  c ; \\ \left\{ \frac{a}{bc} \right\}, & \text{если }  a  +  b  <  c . \end{cases}$
26.	$y = \begin{cases} \left[ \frac{a^3}{2b} \right], & \text{если } x < -7; \\ 2(a - b^2), & \text{если } -7 \leq x < 10; \\ \left[ \frac{a + b}{3} \right], & \text{если } x \geq 10. \end{cases}$
27.	$y = \begin{cases} a + b - ab, & \text{если }  a + b  > 10; \\ a^2 + b^2, & \text{если }  a + b  = 10; \\ b^2 + ab - a^2, & \text{если }  a + b  < 10. \end{cases}$
28.	$y = \begin{cases} ax - b, & \text{если } x < -5; \\ a + 2x, & \text{если } -5 \leq x \leq 1; \\ \left[ \frac{a}{x} \right] - 3b, & \text{если } x > 1. \end{cases}$

29.	$y = \begin{cases} \frac{7ab}{x^2 + a^2}, & \text{если }  x  >  a ; \\ \frac{ax+b}{x^2 + a^2}, & \text{если }  x  =  a ; \\ 25ab + x^2, & \text{если }  x  <  a . \end{cases}$
30.	$y = \begin{cases} \frac{100}{x^2 + ab}, & \text{если } x \leq -10; \\ (x^2 + ab)^2 - \frac{a}{b}, & \text{если } -10 < x < 0; \\ \frac{x^2 + ab - b^3}{a + bx}, & \text{если } x \geq 0. \end{cases}$
31.	$y = \begin{cases} \left[ \frac{ax}{3+b} \right], & \text{если }  x  +  z  < 13; \\ 0, & \text{если }  x  +  z  = 13; \\ a^3 - 7, & \text{если }  x  +  z  > 13. \end{cases}$
32.	$y = \begin{cases} ax + b^2, & \text{если }  a + b  > 10; \\ \frac{ab}{x}, & \text{если }  a + b  = 10; \\ \frac{bx + a^2}{ a + b  + 1}, & \text{если }  a + b  < 10. \end{cases}$
33.	$y = \begin{cases} \frac{x+a}{ x+a } + bc, & \text{если } \frac{bc}{a} > 10; \\ bx - ca, & \text{если } \frac{bc}{a} < 10; \\ abc, & \text{если } \frac{bc}{a} = 10. \end{cases}$
34.	$y = \begin{cases} x, & \text{если } 5a^2 + b^2 -  c  < 0; \\ \left[ \frac{a}{b+c} \right], & \text{если } 5a^2 + b^2 -  c  = 0; \\ -x, & \text{если } 5a^2 + b^2 -  c  > 0. \end{cases}$
35.	$y = \begin{cases} \left[ \frac{a}{2} \right], & \text{если } x^2 + y^2 < r^2; \\ -a, & \text{если } x^2 + y^2 = r^2; \\ \left\{ \frac{7a}{ x-y } \right\}, & \text{если } x^2 + y^2 > r^2. \end{cases}$

36.	$y = \begin{cases} ax^2 + bx + c, & \text{если }  x  < 7; \\ 0, & \text{если } x \leq -7; \\ \frac{ab}{c} - x, & \text{если } x \geq 7. \end{cases}$
37.	$y = \begin{cases} mx^2 + n, & \text{если } x > 10; \\ (m+n)x, & \text{если } 5 < x \leq 10; \\ m^2 + n^2, & \text{если } x \leq 5. \end{cases}$
38.	$y = \begin{cases} \left[ \frac{a+b+c}{c} \right], & \text{если }  a+c  >  b ; \\ ac^2, & \text{если }  a+c  =  b ; \\ \frac{abc}{ a+c }, & \text{если }  a+c  <  b . \end{cases}$
39.	$y = \begin{cases} -1, & \text{если } \left[ \frac{a}{b} \right]^2 < x; \\ 0, & \text{если } \left[ \frac{a}{b} \right]^2 = x; \\ 1, & \text{если } \left[ \frac{a}{b} \right]^2 > x. \end{cases}$
40.	$y = \begin{cases} \frac{ ax+b }{ ax +5}, & \text{если } a > b; \\ \frac{ ax-b }{ ax +3}, & \text{если } a = b; \\ \frac{abx}{a+b}, & \text{если } a < b. \end{cases}$
41.	$y = \begin{cases} \frac{\min(c, x)}{(a-b)+5x}, & \text{если }  a - b  > 5; \\ \frac{\min(a, x)}{(a-b)+5x}, & \text{если }  a - b  = 5; \\ \frac{\min(b, x)}{(a-b)+5x}, & \text{если }  a - b  < 5. \end{cases}$

### Указания к оформлению отчета

Отчет оформляется в виде текстового файла и должен содержать:

- · фамилию, имя, группу и вариант студента;
- · номер задания, ответ на задание (если требуется);
- · комментарий к выполнению задания (если требуется).

## Лабораторная работа №3

### Форматирование вывода и команды передачи управления

*Цель работы:*

1. Изучить функцию форматирования вывода.
2. Изучить команды передачи управления (безусловные и условные переходы).

1. Для форматирования вывода числовых данных используется функция `wsprintf`. У функции переменное число параметров. Рассмотрим первые три.

`wsprintf ADDR buffer_for_string, ADDR szformat, number`

`ADDR buffer_for_string` - адрес начала строки для хранения результата форматирования;

`ADDR szformat` - адрес начала строки формата;

`Number` - параметр, который подставляется в строку формата.

#### Листинг 1

```
.386
.model flat, stdcall
option casemap:none
include c:\windows\include\windows.inc
include c:\windows\include\user32.inc
include c:\windows\include\kernel32.inc
includelib c:\windows\lib\user32.lib
includelib c:\windows\lib\kernel32.lib
bufsize equ 12

.data
fmt db "Number = %2d",0
buf db bufsize dup(?)
crlf db 0dh,0ah
stdout dd ?
cWritten dd ?
.code
start:
invoke GetStdHandle, STD_OUTPUT_HANDLE
mov stdout, eax
```

```

mov edx,1
mov ecx,10
nxt:
push  ecx
push  edx
invoke sprintf, ADDR buf, ADDR fmt,edx
invoke WriteConsoleA, stdout, ADDR buf,\
bufsize, ADDR cWritten, NULL
invoke WriteConsoleA, stdout, ADDR crlf,\
2, ADDR cWritten, NULL
pop   edx
inc   edx
pop   ecx
loop  nxt
invoke ExitProcess, 0
end start

```

Для форматирования используются стандартные форматы (аналогично используемым в функции printf языка C). Числа с плавающей точкой форматировать таким образом нельзя.

В таблице 1 приведены операторы условного перехода, зависящие от флагов. Их можно использовать для определения фактов переноса, переполнения и т.д.

Таблица 1– Команды условного перехода и флаги

Типы операндов	Мнемокод команды условного перехода	Критерий условного перехода	Значения флагов для осуществления перехода
Любые	je	операнд_1 = операнд_2	zf = 1
Любые	jne	операнд_1 $\neq$ операнд_2	zf = 0
Со знаком	jl/jnge	операнд_1 < операнд_2	sf $\neq$ of
Со знаком	jle/jng	операнд_1 $\leq$ операнд_2	sf $\neq$ of or zf = 1
Со знаком	jg/jnle	операнд_1 > операнд_2	sf = of and zf = 0
Со знаком	jge/jnl	операнд_1 $\geq$ операнд_2	sf = of
Без знака	jb/jnae	операнд_1 < операнд_2	cf = 1
Без знака	jbe/jna	операнд_1 $\leq$ операнд_2	cf = 1 or zf=1
Без знака	ja/jnbe	операнд_1 > операнд_2	cf = 0 and zf = 0
Без знака	jae/jnb	операнд_1 $\geq$ операнд_2	cf = 0

В листинге 2 приведена программа, определяющая и выводящая на экран большее из двух чисел. Числа заданы переменными в сегменте данных.

## ЛИСТИНГ 2

.386

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

bufsize equ 8

.data
fmt db "Max=%d",0
buf db bufsize dup(?)
crlf db 0dh,0ah
stdout dd ?
cWritten dd ?
a dd 100
b dd 210

.code
start:
invoke GetStdHandle, STD_OUTPUT_HANDLE
mov stdout, eax
mov eax, a
cmp eax, b
jb _b
mov edx, eax
jmp print
_b:
mov edx, b
print:
invoke wsprintf, ADDR buf, ADDR fmt,edx
invoke WriteConsoleA, stdout, ADDR buf,\
bufsize, ADDR cWritten, NULL
invoke WriteConsoleA, stdout, ADDR crlf,\
```

2, ADDR cWritten, NULL

invoke ExitProcess, 0  
end start

### **Индивидуальные задания**

1. Довести программу до логического завершения из листинга 2 так, чтобы она определяла наибольшее число и выводила в зависимости от результата проверки одно из трех сообщений: « $a > b$ », « $a < b$ », « $a = b$ ».
2. Написать программу, выводящую минимальное значение из 4 чисел (числа задаются переменными в сегменте данных).

### **Указания к оформлению отчета**

Отчет оформляется в виде текстового файла и должен содержать:

- · фамилию, имя, группу и вариант студента;
- · номер задания, ответ на задание (если требуется);
- · комментарий к выполнению задания (если требуется).



## Лабораторная работа №4.

### Основы работы с числами с плавающей точкой

#### *Цель работы:*

1. Научиться работать с вещественными числами (числа с плавающей запятой).

#### *Краткая теория:*

Чтобы повысить точность и максимально компактно расположить вещественное число в памяти компьютера, были придуманы числа с плавающей точкой. В старшие биты стали записывать порядок числа (Нормализованная запись числа). Размер порядка числа известен заранее (зависит от типа данных) и занимает намного меньше места, чем могла бы занять целая часть числа.

В младшие биты стали записывать мантиссу – нормализованную экспоненциальную форму числа без запятой. Таким образом, в пределах мантиссы точка может как бы «плавать», то есть её расположение зависит от порядка числа.

Но как заставить эту точку плавать? Этим занимается процессор, то есть аппаратная часть компьютера. Во многих современных процессорах даже есть специальные команды для операций над числами с плавающей точкой (но об этом позже). В большинстве компьютеров используются именно числа с плавающей точкой (а не с фиксированной), потому что это позволяет экономить память и получать большую точность.

Вспомним алгоритм представления вещественного числа в памяти компьютера:

1. Перевести число из  $P$ -ичной системы в двоичную
2. Представить двоичное число в нормализованной экспоненциальной форме
3. Рассчитать смещённый порядок числа
4. Разместить знак, порядок и мантиссу в соответствующие разряды

В разделе Представление вещественных чисел в памяти компьютера первый шаг мы уже сделали и получили двоичное представление целой и дробной части числа 3,14:

$$3 = 11b$$

$$0,14 = 0,00100011b$$

То есть число 3,14 в двоичном виде равно:

$$3,14 = 11,00100011b$$

Теперь преобразуем это число в нормализованную экспоненциальную форму:

$$11,00100011b = 1,100100011b \times 2^1$$

Теперь рассчитаем смещённый порядок (предположим, что для хранения порядка у нас используется 5 бит). Тогда исходные данные:

$$ИП = 1 \text{ (у нас } 2 \text{ в степени } 1)$$

$$k = 5$$

$$СП = ИП + 2^{k-1} - 1 = 1 + 2^{5-1} - 1 = 1 + 16 - 1 = 16$$

Записываем знак числа, порядок и мантиссу в соответствующие разряды:

Знак	Порядок	Мантисса
0	10000	0010001100

Как видите, в мантиссе у нас младшие два разряда – это нули. Эти разряды нами не используются, но при желании мы бы могли их использовать и тем самым повысить точность.

А теперь давайте спустимся с небес на землю. Все приведённые нами примеры являются упрощёнными. В реальных машинах обычно для чисел с плавающей точкой используются числа с большим количеством разрядов. Но реальные числа мы здесь рассматривать не будем, тем более что представление данных может отличаться в зависимости от процессора. Для первого знакомства информации достаточно. Возможно, что эту тему я расширю в будущих изданиях книги. А пока, если хотите знать больше – изучайте стандарт **IEEE 754**, который реализован во всех x86-совместимых процессорах. Переходим непосредственно к Ассемблеру.

Микропроцессор для работы с ЧПТ использует математический сопроцессор (МС). В МС есть 8 регистров для хранения чисел, обозначаемых ST0-ST7. Эти регистры образуют стек. Таким образом, при загрузке числа в регистр командой **fld** оно помещается на вершину стека в регистр ST0 (или просто ST).

*Команды пересылки данных*

**fld**– загрузка в регистр ЧПТ

**fild** – загрузка в регистр целого числа

**fstp** – извлечение из стека и запись в переменную ЧПТ

**fistp** - извлечение из стека и запись в переменную целого числа

### *Арифметические команды*

Сопроцессор использует шесть основных типов арифметических команд:

- Fxxx – первый операнд берется из верхушки стека (источник), второй - следующий элемент стека. Результат выполнения команды записывается в стек

- Fxxx память – источник берется из памяти, приемником является верхушка стека ST(0). Указатель стека ST не изменяется, команда действительна только для операндов с одинарной и двойной точностью

- Fixxx память – аналогично предыдущему типу команды, но операндами могут быть 16- или 32-разрядные целые числа

- Fxxx ST, ST(i) – для этого типа регистр ST(i) является источником, а ST(0) - верхушка стека - приемником. Указатель стека не изменяется

- Fxxx ST(i), ST – для этого типа регистр ST(0) является источником, а ST(i) - приемником. Указатель стека не изменяется

- FxxxP ST(i), ST – регистр ST(i) - приемник, регистр ST(0) - источник. После выполнения команды источник ST(0) извлекается из стека

Строка "xxx" может принимать следующие значения:

- ADD - Сложение
- SUB - Вычитание
- SUBR - Обратное вычитание, уменьшаемое и вычитаемое меняются местами

- MUL - Умножение

- DIV - Деление

- DIVR - Обратное деление, делимое и делитель меняются местами

Кроме основных арифметических команд имеются дополнительные арифметические команды:

- FSQRT - Извлечение квадратного корня

- FRNDINT - Округление до целого

- FABS - Вычисление абсолютной величины числа

- FCHS - Изменение знака числа

По команде FSQRT вычисленное значение квадратного корня записывается в верхушку стека ST(0).

Команда RNDINT округляет ST(0) в соответствии с содержимым поля RC управляющего регистра.

Команда FABS вычисляет абсолютное значение ST(0). Аналогично, команда FCHS изменяет знак ST(0) на противоположный.

Для подготовки к выводу на экран ЧПТ можно использовать функцию FpuFLtoA, преобразующую ЧПТ в строку.

Формат функции следующий: FpuFLtoA Num, DIGS, BUF, константы

Num – адрес числа;

DIGS – число знаков после запятой;

BUF – адрес буфера для хранения строки;

Константы – одна или несколько констант, указывающих вид числа (SRC1\_REAL - вещественное)

*Трансцендентные команды математического сопроцессора*

•FCOS Вычисление  $\cos(x)$

•FSIN Вычисление  $\sin(x)$

Аргумент функций должен находиться в st(0), результат сохраняется там же.

## Индивидуальные задания

### Задание 1. Вычислите значение выражения

1)  $(x - y) \times z + \frac{z}{x}$

2)  $(y - x) \times z + z \times x$

3)  $\frac{(z - x)}{y} - \frac{z}{x}$

4)  $(x + z) \times y + \frac{z}{x}$

5)  $(x + y) \times z + y \times z$

6)  $(y - z) \times x + y \times x$

7)  $\frac{(z + x)}{z} \times z - \frac{y}{x}$

8)  $\frac{(z + x)}{y} \times x + x \times y$

9)  $(y + z) \times z + x \times x$

10)  $\frac{(y + y)}{z} - z \times x$

при  $x = 749.05$ ;  $y = 900.40$ ;  $z = 355.31$ .

**Задание 2. Вычислите значение выражения**

$$1) \sqrt{\left| \frac{\frac{x}{y}}{\sin(\frac{y}{y \times x})} \right|}$$

$$2) \sqrt{\left| \frac{\sin(\frac{x \times y}{\left| \cos(\frac{y}{x}) \right|})}{\left| \cos(\frac{y}{x}) \right|} \right|}$$

$$3) \frac{\sqrt{|\cos(x \times y)|}}{y \times x}$$

$$4) \sqrt{\left| \frac{\cos(\frac{x}{y})}{\left| \sin(y \times x) \right|} \right|}$$

$$5) \sqrt{\left| \frac{\sin(x \times y)}{y \times x} \right|}$$

$$6) \sqrt{\left| \sin(\frac{x}{y}) \right| \times |\cos(y \times x)|}$$

$$7) \sqrt{|\cos(x \times y)| - \frac{y}{x}}$$

$$8) \sqrt{|\cos(x \times y)| \times |\sin(y - x)|}$$

$$9) \sqrt{\left| \frac{\cos(y + x)}{x \times y} \right|}$$

$$10) \sqrt{\frac{|\sin(y \times x)|}{|\cos(x \times y)|}}$$

где  $x = 23.05$ ;  $y = 12.79$ .

**Указания к оформлению отчета**

Отчет оформляется в виде текстового файла и должен содержать:

- · фамилию, имя, группу и вариант студента;
- · номер задания, ответ на задание (если требуется);
- · комментарий к выполнению задания (если требуется).

## Лабораторная работа №5

### Директивы управления потоком. Основы работы с массивами.

#### Цель работы:

1. Изучить работу директив, управляющих потоками.
2. Изучить и применить на практике работу с массивами на языке низкого уровня программирования.

Для организации ветвления и циклов можно использовать специальные директивы ассемблера для управления потоком. При этом директивы при ассемблировании будут преобразованы в обычные команды проверки условий и переходов.

#### Пример 1 – Условный оператор

```
.IF условие  
операторы1  
.ELSE  
операторы2  
.ENDIF
```

Блок "операторы1" выполняется, если условие истинно, блок "операторы" - если ложно.

#### Листинг 1. Определить $a < b$ или $a > b$

```
.386  
.model flat, stdcall  
option casemap:none  
  
include c:\windows\include\windows.inc  
include c:\windows\include\user32.inc  
include c:\windows\include\kernel32.inc  
include c:\windows\include\fpw.inc  
includelib c:\windows\lib\user32.lib  
includelib c:\windows\lib\kernel32.lib  
includelib c:\windows\lib\fpw.lib
```

```
BSIZE equ 30
```

```
.data
```

```
a dd 10
```

```
b dd 20
```

```
stdout dd ?
```

```
cWritten dd ?
```

```
buf db BSIZE dup (?)
```

```
fmt db "a %c b",13,10,0
```

```
.code
```

```
start:
```

```
invoke GetStdHandle, STD_OUTPUT_HANDLE
```

```
mov stdout, eax
```

```
mov eax,a
```

```
.IF eax < b
```

```
mov edi,"<"
```

```
.ELSE
```

```
mov edi,">"
```

```
.ENDIF
```

```
invoke sprintf, ADDR buf, ADDR fmt,edi
```

```
invoke WriteConsoleA, stdout, ADDR buf,\  
BSIZE, ADDR cWritten, NULL
```

```
invoke ExitProcess, 0
```

```
end start
```

```
invoke ExitProcess, 0
```

```
end start
```

*Пример 2 - Циклы типа WHILE .. DO и REPEAT ... UNTIL*

```
.WHILE условие
```

```
операторы
```

```
.ENDW
```

```
.REPEAT
```

операторы  
.UNTIL условие

2. Массив в ассемблере можно определить, например, следующим образом

m1 dd 10 dup(0) ; массив из 10 элементов (32 бита каждый), инициализированных нулями  
m2 dd 10,20,30,50,100 ; массив из 5 элементов, инициализированных явно

## Листинг 2. Работа с массивами

```
.386
.model flat, stdcall
option casemap:none

include c:\windows\include\windows.inc
include c:\windows\include\user32.inc
include c:\windows\include\kernel32.inc
include c:\windows\include\fpu.inc
includelib c:\windows\lib\user32.lib
includelib c:\windows\lib\kernel32.lib
includelib c:\windows\lib\fpu.lib

BSIZE equ 7

.data
m1 dd 10 dup(0)
m2 dd 10,20,30,50,100
stdout dd ?
cWritten dd ?
buf db BSIZE dup (?)
fmt db "%4d",13,10,0

.code
start:
invoke GetStdHandle, STD_OUTPUT_HANDLE
```



```

mov stdout, eax

mov edi,0 ; в - индекс массива
mov eax,1 ; всего 10 элементов, счетчик цикла

;заполним массив m1 квадратами чисел от 1 до 10.

.WHILE eax <= 10
push eax
mul eax
mov m1[edi],eax
add edi,4
pop eax
inc eax
.ENDW

; выведем на экран массив m2
mov edi,0
mov ecx,0
.REPEAT
push ecx
mov eax,m2[edi]
invoke sprintf, ADDR buf, ADDR fmt,eax
invoke WriteConsoleA, stdout, ADDR buf,\
BSIZE, ADDR cWritten, NULL
add edi,4
pop ecx
inc ecx
.UNTIL ecx == 5

invoke ExitProcess, 0
end start

```

### **Индивидуальные задания**

Задание 1. Определите максимальное и минимальное значения в массиве из 25 элементов. Массив, найденные элементы и их номера выведите на экран. Массив задать явно.

Задание 2. Отсортируйте массив из 17 элементов по возрастанию. Исходный и отсортированный массив выведите на экран. Массив задать явно, использовать алгоритм пузырьковой сортировки.

### **Указания к оформлению отчета**

Отчет оформляется в виде текстового файла и должен содержать:

- · фамилию, имя, группу и вариант студента;
- · номер задания, ответ на задание (если требуется);
- · комментарий к выполнению задания (если требуется).

## Лабораторная работа №6

### Условные переходы

#### Цель работы:

Изучить работу команд условного перехода (команды CMP и TEST).

Для организации ветвлений и более сложных чем loop циклов в ассемблере используется следующая конструкция.

Сначала производится сравнение двух чисел с помощью команды cmp:

```
cmp eax, ebx
```

Операнды у команды cmp могут быть любые, кроме двух ячеек памяти. Результат сравнения заносится в регистр флагов.

Затем на основании этого результата осуществляется условный переход на указанную метку.

```
jb label ; jb - jump if below
```

Если  $eax < ebx$ , то будет произведен переход на метку label, иначе будет выполнен следующий после jb оператор.

Список операторов условного перехода:

ja - jump if above ( $>$ )

jb - jump if below ( $<$ )

jg - jump if greater ( $>$ )

jl - jump if less ( $<$ )

jae - jump if above or equal ( $\geq$ )

jbe - jump if below or equal ( $\leq$ )

jge - jump if greater or equal ( $\geq$ )

jle - jump if less or equal ( $\leq$ )

je - jump if equal ( $=$ )

jne - jump if not equal ( $\neq$ )

jmp - unconditional jump

above/below означают беззнаковое сравнение, greater/less - знаковое:

```
mov al, 0xff
```

```
cmp al, 1
```

ja x ; Беззнаковое сравнение:  $255 > 1$ , переход будет выполнен

```
mov al, 0xff
```

```
cmp al, 1
```

jg x ; Знаковое сравнение:  $-1 < 1$ , переход не будет выполнен

## Пример

Программа, проверяющая, является ли знаковое число "a" положительным.

```
char* istrue = "a > 0";
char* isfalse = "a <= 0";
char* answer;
int a; scanf("%i", &a);
__asm {
    cmp a, 0
    jg _greater;
    mov eax, isfalse;
    jmp _end;
_greater:
    mov eax, istrue;
_end:
    mov answer, eax;
}
printf("%s\n", answer);
```

## Индивидуальные задания

### Задание 1.

Дана точка (x, y) и прямоугольник (x1, y1), (x1, y2), (x2, y1), (x2, y2), известно, что  $x_1 < x_2$  и  $y_1 < y_2$ .

Проверить, находится ли точка внутри прямоугольника.

### Задание 2.

Определить количество корней у уравнения  $ax^2 + bx + c = 0$  (варианты: нет, 1, 2, бесконечно много).

### Задание 3.

Перевернуть число x задом наперёд и увеличить результат на 1. Числа  $x_1 = 1234$ ,  $x_2 = 987$ .

Примеры:

1234 -> 4321+1 = 4322

987 -> 789 + 1 = 790

### **Указания к оформлению отчета**

Отчет оформляется в виде текстового файла и должен содержать:

- · фамилию, имя, группу и вариант студента;
- · номер задания, ответ на задание (если требуется);
- · комментарий к выполнению задания (если требуется).

## Лабораторная работа №7

### Процедуры

*Цель работы:*

1. Работа с процедурами (синтаксис определения, Описание, Описание языка)

Ассемблер позволяет нам описывать процедуры несколькими способами. В данной работе описываются процедуры, объявление языка процедур, использование в процедурах аргументов и переменных, сохранение регистров и вложенные процедуры.

В упрощенном виде шаблон процедуры выглядит следующим образом.

*Имя\_процедуры proc [uses используемые\_регистры]*

*...*

*[код процедуры]*

*ret*

*имя\_процедуры endp*

*Пример 1.* Процедура, вычисляющая сумму трех целых чисел

; используются регистры: eax, ebx, ecx – значения суммируемых чисел

; eax – результат

sum2 proc

add eax,ebx

add eax,ecx

ret

sum2 endp

Листинг 1. Подсчет суммы элементов целочисленного массива

.386

.model flat, stdcall

option casemap:none

include c:\windows\include\windows.inc

include c:\windows\include\kernel32.inc

include c:\windows\include\fpw.inc

includelib c:\windows\lib\user32.lib

```

includelib c:\windows\lib\kernel32.lib
includelib c:\windows\lib\fpu.lib
BSIZE equ 10
.data
array dd 10, 15, 49, 30, 85, 100, 15, 94, 12, 34
sum dd 0
stdout dd ?
cWritten dd ?
buf db BSIZE dup (?)
fmt db "%4d",13,10,0
.code
; вычисляет сумму элементов массива 32-х разрядных целых чисел
; адрес массива – esi, количество элементов – ecx
; сумма – eax
arraySum proc
    push esi          ; сохраним используемые регистры
    push ecx
    mov eax, 0
L1:
    add eax, [esi]     ; добавим к сумме очередное значение из массива
    add esi, 4         ; установим указатель на следующий элемент
    loop L1
    pop ecx
    pop esi           ; восстановим используемые регистры
    ret
arraySum endp
start:
    invoke GetStdHandle, STD_OUTPUT_HANDLE
    mov stdout, eax
    mov esi, offset array ; указатель на начало массива
    mov ecx, lengthof array ; размер массива
    call arraySum
    mov sum, eax
    invoke SetConsoleTextAttribute, \
stdout, 00000101b ; установим цвет текста и фона
; старшая тетрада - фон, младшая тетрада - текст
    invoke wsprintf, ADDR buf, ADDR fmt, sum
    invoke WriteConsoleA, stdout, ADDR buf, \
BSIZE, ADDR cWritten, NULL

```

```
invoke SetConsoleTextAttribute, stdout, 7 ; вернем цвет по умолчанию – серый
invoke ExitProcess, 0
end start
```

Если в заголовок процедуры добавить ключевое слово `uses`, то сохранение используемых регистров и их восстановление ассемблер выполнит автоматически.

Заголовок должен выглядеть так:

```
arraySum proc uses esi ecx
```

Регистры перечисляются через пробел или символ табуляции.

### Индивидуальные задания

**Задание 1.** Доработать программу из листинга 1 таким образом, чтобы она выводила массив на экран. Каждый элемент отобразить разным цветом (используйте номера от 1 до 10).

**Задание 2.** Напишите процедуру вывода на экран целого числа заданным цветом. Параметры – число и номер цвета (номер цвета фона всегда 0 – черный). Доработайте программу из задания 1 так, чтобы в ней использовалась эта процедура.

**Задание 3.** Напишите программу для вычисления среднего значения элементов целочисленного массива. Для определения среднего используйте процедуру с параметрами: адрес массива, число элементов массива. Программа должна выводить исходный массив и найденное среднее значение.

**Задание 4.** Доработайте программу из задания 3 таким образом, чтобы при выводе элементов массива использовалось цветовое разделение значений: красным цветом элементы, большие, либо равные среднему (в этом случае используйте целое среднее); желтым – остальные.

**Задание 5.** Напишите программу преобразования температуры по шкале Цельсия в температуру по шкале Фаренгейта для значений от 15 до 36 градусов Цельсия (указание:  $1F = 32 + 1.8C$ ).



**Задание 6.** Определите сумму ряда, где  $x$  – номер вариант\*3,  $y$  – номер варианта. Промежуточные значения суммы вывести на экран. Для вывода и вычисления очередного элемента ряда используйте процедуры.

**Указания к оформлению отчета**

Отчет оформляется в виде текстового файла и должен содержать:

- · фамилию, имя, группу и вариант студента;
- · номер задания, ответ на задание (если требуется);
- · комментарий к выполнению задания (если требуется).

## Лабораторная работа №8

### Связь подпрограмм на ассемблере с программами на языке высокого уровня

*Цель работы:*

1. Изучение методов использования ассемблерных подпрограмм в программах на языках высокого уровня.

Комбинирование программ на языке высокого уровня с кодом на ассемблере используется в том случае, если присутствуют участки, которые невозможно реализовать без использования ассемблера, либо его применение значительно повысит эффективность работы.

Существуют следующие формы комбинирования программ на языках высокого уровня с ассемблером:

- Использование операторов типа `inline` и ассемблерных вставок. Эта форма сильно зависит от синтаксиса языка высокого уровня и конкретного компилятора. Она предполагает, что ассемблерные коды в виде команд ассемблера или прямо в машинных командах вставляются в текст программы на языке высокого уровня. Компилятор языка распознает их как ассемблера и без изменений включает в формируемый им объектный код. Эта форма удобна, если вставлять небольшой фрагмент.
- Использование внешних процедур и функций. Это более универсальная форма комбинирования. У нее есть ряд преимуществ:
  1. Написание и отладку программ можно производить независимо.
  2. Написанные программы можно использовать в других проектах.
  3. Облегчаются модификация и сопровождение подпрограмм в течение жизненного цикла проекта.

Возможны два вида связи – программа на языке высокого уровня вызывает процедуру на ассемблере и наоборот. Синтаксис директивы `имя_процедуры PROC [[модификатор_языка]язык] [расстояние]`

Таблица 2 – Значения операнда язык

Операнд «язык»	Язык	Направление передачи аргументов	Какая процедура очищает стек
NOLANGUAGE	Ассемблер	Слева направо	Вызываемая
BASIC	Basic	Слева направо	Вызываемая
PROLOG	Prolog	Справа налево	Вызывающая
FORTRAN	Fortran	Слева направо	Вызываемая
C	C	Справа налево	Вызывающая
C++	C++	Справа налево	Вызывающая
PASCAL	Pascal	Слева направо	Вызываемая
STDCALL	–	Справа налево	Вызываемая
SYSCALL	C++	Справа налево	Вызывающая

### Индивидуальные задания

0. Подсчитать количество элементов  $<x$  в массиве  $[4 \times 4]$ . Число  $x$  задано в сегменте данных.
1. Дано натуральное число. Найти сумму первой и последней цифры данного числа. Результат поместить в АХ. Число двухзначное.
2. Дано натуральное число. Найти сумму первой и последней цифры данного числа. Результат поместить в АХ. Число трехзначное.
3. Составить программу, которая находит количество отрицательных элементов массива  $[5 \times 5]$  и вывести их.
4. Разработать программу, которая при вводе с клавиатуры двух строк определяет, сколько раз вторая строка встречается в первой.
5. Напишите программу, которая запрашивает строку символов с клавиатуры, а затем на экран выводит все символы этой строки кроме знаков препинания.
6. Среднее арифметическое отрицательное чисел массива  $N \times M$  чисел (Сделать счетчик отрицательных чисел, разделить их, получить среднее арифметическое)
7. В матрице  $4 \times 4$  разделить все положительные элементы главной диагонали на 2.
8. Сформировать одномерный массив  $A=[i]$  ( $i=1...20$ ), где  $a_i$  вычисляется по формулам:  

$$A[i] = (2 * i + 10), \text{ где } i \quad 0 < i < 11$$

$$A[i] = (i * i / 3), \text{ где } i \quad 10 < i < 21$$
9. С клавиатуры вводится матрица  $N \times M$ . Найти максимальный элемент в каждой строке.

### **Указания к оформлению отчета**

Отчет оформляется в виде текстового файла и должен содержать:

- · фамилию, имя, группу и вариант студента;
- · номер задания, ответ на задание (если требуется);
- · комментарий к выполнению задания (если требуется).

## Лабораторная работа №9

### Изучение команд обработки строк

*Цель работы:*

1. Научиться использовать команды обработки строк при написании ассемблерных программ.

Замечания: реализовать работу со строками при помощи команд обработки строк

### Индивидуальные задания

1. Написать процедуру копирования строки.
2. Написать процедуру объединения двух строк.
3. Дана строка. Преобразовать строчные буквы в прописные. Рассмотреть только латинский алфавит
4. Дана строка. Преобразовать строчные буквы в прописные. Рассмотреть только русский алфавит
5. Дана строка. Преобразовать прописные буквы в строчные. Рассмотреть только латинский алфавит.
6. Дана строка. Преобразовать прописные буквы в строчные. Рассмотреть только русский алфавит.
7. Написать процедуру, осуществляющую сравнение строк. Рассмотреть только латинский алфавит.
8. Написать процедуру, осуществляющую сравнение строк. Рассмотреть только русский алфавит.
9. Написать процедуру, выводящую строку на экран путем прямого доступа к видеопамяти.
10. Зашифровать и расшифровать исходную строку.
11. Написать процедуру копирования массива типа `char`. Во входных параметрах необходимо задать количество элементов массива.
12. Написать процедуру копирования массива типа `long`. Во входных параметрах необходимо задать количество элементов массива.

13. Обменять между собой содержимое 0-й и 1-й страниц видеопамати.
14. Сохранить содержимое 0-й страницы видеопамати в файле на диске.
15. Очистить экран заданным цветом путем прямого доступа к видеопамати.

### **Указания к оформлению отчета**

Отчет оформляется в виде текстового файла и должен содержать:

- · фамилию, имя, группу и вариант студента;
- · номер задания, ответ на задание (если требуется);
- · комментарий к выполнению задания (если требуется).

## Лабораторная работа №10

### Написание собственного обработчика прерывания

*Цель работы:*

1. Научиться разрабатывать собственные обработчики аппаратных прерываний.

### Индивидуальные задания

Написать резидентную программу, которая содержит собственный ISR прерывания 9 (аппаратное прерывание клавиатуры). Обработчик должен выдавать на экран в заданную позицию экрана заданное сообщение, при нажатии на определенную клавишу. Позиция экрана, сообщение и нажатая клавиша задается согласно варианту.

**Таблица 1. Позиция экрана, куда выдается сообщение**

1-я цифра варианта	0	1	2	3	4	5	6	7	8	9
X	4	8	17	9	20	3	4	0	7	12
Y	65	78	2	4	5	26	47	54	18	0

**Таблица 2. Сообщение, которое выдается на экран**

2-я цифра варианта	0	1	2	3	4	5	6	7	8	9
Сообщение	Lat	Num	0	Hi	Good	P41	Pk	Mm	sk	ok

**Таблица 3. Клавиша, по которой выдается сообщение на экран**

3-я цифра варианта	0	1	2	3	4	5	6	7	8	9
Клавиша	Tab	F1	End	Ctrl	Alt	K	BKSP	F5	F8	Del

### Указания к оформлению отчета

Отчет оформляется в виде текстового файла и должен содержать:

- · фамилию, имя, группу и вариант студента;
- · номер задания, ответ на задание (если требуется);
- · комментарий к выполнению задания (если требуется).

## VI ТЕСТОВЫЕ ЗАДАНИЯ

### 1. Язык ассемблера – это...

+	язык программирования низкого уровня. В отличие от языка машинных кодов, позволяет использовать более удобные для человека мнемонические (символьные) обозначения команд. При этом для перевода программы с языка ассемблера в понимаемый процессором машинный код требуется специальная программа, называемая ассемблером.
-	язык программирования высокого уровня. В отличие от языка машинных кодов, позволяет использовать более удобные для человека мнемонические (символьные) обозначения команд. При этом для перевода программы с языка ассемблера в понимаемый процессором машинный код требуется специальная программа, называемая ассемблером.
-	язык программирования высокого уровня, основанный на pascal. В отличие от языка машинных кодов, позволяет использовать более удобные для человека мнемонические (символьные) обозначения команд. При этом для перевода программы с языка ассемблера в понимаемый процессором машинный код требуется специальная программа, называемая ассемблером.
-	язык программирования высокого уровня, основанный на C++. В отличие от языка машинных кодов, позволяет использовать более удобные для человека мнемонические (символьные) обозначения команд. При этом для перевода программы с языка ассемблера в понимаемый процессором машинный код требуется специальная программа, называемая ассемблером.

### 2. Какими достоинствами обладают программы на ассемблере?

-	легким восприятием.
+	компактностью кода.
+	высокой скоростью.
-	малым количеством операторов.

### 3. Область применения ассемблера?

-	веб-приложения.
+	драйвера устройств.
+	приложения для встраиваемых систем.
+	приложения, с высоким требованием к размеру кода.

### 4. CPU - это...

-	( <i>central processing unit, CPU</i> , дословно – центральное обрабатывающее
---	---



	устройство) – исполнитель машинных инструкций, часть программного обеспечения компьютера или программируемого логического контроллера; отвечает за выполнение операций, заданных программами.
+	( <i>central processing unit, CPU</i> , дословно – центральное обрабатывающее устройство) – исполнитель машинных инструкций, часть аппаратного обеспечения компьютера или программируемого логического контроллера; отвечает за выполнение операций, заданных программами.
-	Central position output.
-	( <i>central processing unit, CPU</i> , дословно – центральное обрабатывающее устройство) – исполнитель программных инструкций, часть аппаратного обеспечения компьютера или программируемого логического контроллера; отвечает за выполнение операций, заданных программами.

### 5. Основные функции МП - ...

+	исполнение инструкций программ.
-	исполнение инструкций пользователя.
+	управление и координация работы остальных компонентов ЭВМ.
-	исполнение инструкций операционной системы.

### 6. Архитектуры МП?

+	CISC (Complex Instruction Set Computing).
-	MRSC (Minimal Routing Set Computing)
+	RISC (Reduced Instruction Set Computing).
+	MISC (Minimum Instruction Set Computing)

### 7. Характеристики МП?

-	Скорость.
+	Разрядность.
+	Частота системной шины и памяти.
+	Система команд.

### 8. Из чего состоит МП?

-	Оперативная память.
+	Устройство управления.
+	Арифметико-логическое устройство.
+	Микропроцессорная память.

### 9. Наименьшая единица памяти?

-	1 байт.
-	8 бит.
-	1 кбит.
+	1 бит.

**10. Как происходит нумерация битов в байте?**

+	7,6,5,4,3,2,1,0.
-	0,1,2,3,4,5,6,7.
-	0,7,1,6,2,5,3,4.
-	15, ...,5,4,3,2,1.

**11. Как происходит нумерация битов в слове?**

-	1,2,3,4,5,...,15.
-	0,1,2,3,4,5...,15.
+	15, ...,5,4,3,2,1.
-	0,7,1,6,2,5,3,4.

**12. Какие модели памяти поддерживают МП аппаратно?**

-	Сегментскую.
+	Сегментарную.
-	Статичную.
+	Страничную.

**13. Как реализован комментарий в ассемблере?**

-	//комментарий//
-	/ комментарий/
+	“комментарий”
-	*/ комментарий/*

**14. Отметьте правильные сочетания операндов.**

+	Регистр – регистр.
-	Память-память.
+	Память – регистр.
+	Непосредственный операнд – регистр.

**15. Что обозначает сегментный регистр CS?**

+	сегмент кода.
-	сегмент данных.
-	сегмент стека.
-	дополнительный сегмент.

**16. Что обозначает сегментный регистр DS?**

-	сегмент кода.
+	сегмент данных.
-	сегмент стека.
-	дополнительный сегмент.

**17. Что обозначает сегментный регистр SS?**

-	сегмент кода.
-	сегмент данных.

+	сегмент стека.
-	дополнительный сегмент.

### 18. Что обозначает сегментный регистр ES?

-	-: сегмент кода.
-	-: сегмент данных.
-	-: сегмент стека.
+	+: дополнительный сегмент.

### 19. Что обозначает флаг CF?

-	Флаг четности. В рез. пред. операции четное число единиц.
-	Вспомогательный флаг переноса, для BCD-чисел.
-	Флаг переполнения. Результат превосходит доп. величину.
+	Флаг переноса. Пред. операции произвела перенос.

### 20. Что обозначает флаг PF?

+	Флаг четности. В рез. пред. операции четное число единиц.
-	Вспомогательный флаг переноса, для BCD-чисел.
-	Флаг переполнения. Результат превосходит доп. величину.
-	Флаг переноса. Пред. операции произвела перенос.

### 21. Что обозначает флаг AF?

-	Флаг четности. В рез. пред. операции четное число единиц.
+	Вспомогательный флаг переноса, для BCD чисел.
-	Флаг переполнения. Результат превосходит доп. величину.
-	Флаг трассировки.

### 22. Что обозначает флаг ZF?

-	Флаг знака. В пред. операции отрицательный результат.
-	Вспомогательный флаг переноса, для BCD-чисел.
+	Флаг нуля. Результат пред. операции = 0.
-	Флаг трассировки.

### 23. Что обозначает флаг SF?

+	Флаг знака. В пред. операции отрицательный результат.
-	Вспомогательный флаг переноса, для BCD-чисел.
-	Флаг нуля. Результат пред. операции = 0.
-	Флаг трассировки.

### 24. Что обозначает флаг OF?

-	Флаг знака. В пред. операции отрицательный результат.
+	Флаг переполнения. Результат превосходит доп. величину
-	Флаг нуля. Результат пред. операции = 0.
-	Флаг прерываний.

### 25. Что обозначает флаг TF?

+	Флаг трассировки.
-	Флаг переполнения. Результат превосходит доп. величину.
-	Флаг нуля. Результат пред. операции = 0.
-	Флаг прерываний.

#### **26. Что обозначает флаг IF?**

+	Флаг трассировки.
-	Флаг прерываний.
-	Флаг нуля. Результат пред. операции = 0.
-	Флаг четности. В рез. пред. операции четное число единиц.

#### **27. Что обозначает флаг DF?**

-	Флаг знака. В пред. операции отрицательный результат.
+	Флаг переполнения. Результат превосходит доп. величину
-	Флаг нуля. Результат пред. операции = 0.
-	Флаг направления.

#### **28. Микроконтроллер (MCU) - это ...**

-	это аппаратно-программное устройство вычислительной техники, предназначенное для объединения аудио- и видеоконференции в многоточечный режим.
-	проект по разработке и практическому использованию универсальной компьютерной программы для численного моделирования процессов переноса различного вида излучений (нейтронов, гамма-квантов, электронов) в трёхмерных системах методом Монте-Карло.
+	микросхема, предназначенная для управления электронными устройствами. Типичный микроконтроллер сочетает в себе функции процессора и периферийных устройств, может содержать ОЗУ и ПЗУ. По сути, это однокристальный компьютер, способный выполнять простые задачи.
-	микросхема, предназначенная для управления электронными устройствами. типичный микроконтроллер сочетает в себе функции процессора, не содержит ОЗУ и ПЗУ.

#### **29. Как реализуется параллелизм ассемблерного уровня?**

+	в виде конвейера (конвейера команд, арифметического конвейера или конвейера смешанного типа).
-	в виде многопоточности.
-	в виде параллельности.
+	в виде многопроцессорной системы.

#### **30. Какие группы конвейеров существуют?**

-	Интегральные векторы.
---	-----------------------

+	Векторные конвейеры.
+	Скалярные конвейеры.
-	Производные конвейеры.

### 31. Какие ступени содержит целочисленный конвейер?

+	Ступень предвыборки.
-	Ступени перевыборки
+	Ступень декодирования полей команды D1.
+	Ступень исполнения EX.

### 32. Какой размер данных соответствует байту в ассемблере?

-	w(16).
-	d(64).
-	q(8 bait).
+	b(8 bit).

### 33. Какой размер данных соответствует слову в ассемблере?

+	w(16).
-	d(64).
-	q(8 bait).
-	t(10 bait).

### 34. Какой размер данных соответствует двойному слову в ассемблере?

-	w(16).
+	d(64).
-	q(8 bait).
-	t(10 bait).

### 35. Какие команды отвечают за пересылку данных?

+	MOV.
-	XCOG.
+	XCHG.
+	PTR.

### 36. По каким направлениям команда mov может пересылать данные?

+	Регистр → регистр.
+	Регистр → память.
-	Регистр или основная память → константа.
+	Память → регистр.

### 37. Какие модификаторы размера используются в ассемблере?

+	byte ptr.
+	word ptr.
+	dword ptr.
-	bit ptr.

### 38. Стэк - это...

-	это область памяти, специально выделяемая для постоянного хранения данных программы.
+	это область памяти, специально выделяемая для временного хранения данных программы.
-	это область памяти, выделяемая для постоянного хранения данных программы.
-	это область памяти, специально образующиеся для постоянного хранения данных программы.

### 39. SS - это...

-	регистр указателя стека.
-	регистр указателя базы стека.
+	сегментный регистр стека.
-	сегмент стека.

### 40. Sp/esp - это...

-	регистр указателя стека.
+	регистр указателя базы стека.
-	сегментный регистр стека.
-	сегмент стека.

### 41. Bp/ebp - это...

+	регистр указателя стека.
-	регистр указателя базы стека.
-	сегментный регистр стека.
-	сегмент стека.

### 42. Назначение команды PUSH:

-	извлекает операнд из стека и записывает его в указанный регистр.
+	записывает в стек содержимое указанного регистра.
-	команда групповой записи/чтения в стек.
-	команда группового извлечения из стека.

### 43. Назначение команды POP:

+	извлекает операнд из стека и записывает его в указанный регистр.
-	записывает в стек содержимое указанного регистра.
-	команда групповой записи/чтения в стек.
-	команда группового извлечения из стека.

### 44. Назначение команды PUSHA:

-	извлекает операнд из стека и записывает его в указанный регистр.
-	записывает в стек содержимое указанного регистра.
+	команда групповой записи/чтения в стек.

-	команда группового извлечения из стека.
---	---

#### 45. Процедура - это...

-	это дополнительная функциональная единица декомпозиции (разделения на несколько частей) некоторой задачи.
-	это дополнительная функциональная единица декомпозиции (разделения на несколько частей) некоторой задачи, предназначена для выполнения основных операций.
+	это основная функциональная единица декомпозиции (разделения на несколько частей) некоторой задачи.
-	это часть основной функции.

#### 46. Какие директивы используют для написания процедуры?

+	PROC.
-	PROD.
-	NEDP.
+	ENDP.

#### 47. Где можно размещать процедуру?

+	До точки входа;
+	После команды завершения основной программы;
-	После точки входа;
+	В отдельном сегменте кода.

#### 48. Какими способами можно передавать параметры процедуре?

+	Через регистры (непосредственно).
+	Через регистры (как адреса).
+	Через стек.
-	Через память.

#### 49. Какими способами можно вызвать процедуру?

+	Поместить параметры в стек.
+	Использовать стандартный вызов call.
-	Поместить параметры в регистр.
+	Использовать директиву invoke.

#### 50. Какие разновидности имеет команда jmp?

+	переход прямой короткий (в пределах -128... + 127 байтов)
+	переход прямой ближний (в пределах текущего программного сегмента)
-	переход прямой дальний (в другой программный сегмент)
+	переход косвенный ближний

#### 51. Что обозначает атрибутный оператор short?

-	прямой ближний переход.
---	-------------------------

+	прямой короткий переход.
-	прямой дальний переход.
-	косвенный ближний переход.

**52. Что обозначает атрибутивный оператор near ptr?**

-	прямой ближний переход.
-	прямой короткий переход.
-	прямой дальний переход.
+	косвенный ближний переход.

**53. Что обозначает атрибутивный оператор far ptr?**

-	прямой ближний переход.
-	прямой короткий переход.
+	прямой дальний переход.
-	косвенный ближний переход.

**54. Что обозначает атрибутивный оператор word ptr?**

-	прямой ближний переход.
-	прямой короткий переход.
-	прямой дальний переход.
+	косвенный ближний переход.

**55. Что обозначает атрибутивный оператор dword ptr?**

+	косвенный дальний переход.
-	прямой короткий переход.
-	прямой дальний переход.
-	прямой ближний переход.

**56. Что обозначает команда and?**

-	дизъюнкция.
-	строгая дизъюнкция.
+	конъюнкция.
-	команда сдвига.

**57. Что обозначает команда or?**

+	дизъюнкция.
-	строгая дизъюнкция.
-	конъюнкция.
-	команда сдвига.

**58. Что обозначает команда xor?**

-	дизъюнкция.
+	строгая дизъюнкция.
-	конъюнкция.
-	команда сдвига.



**59. Что обозначает команда cnt?**

-	сдвиг содержимого ячейки влево.
-	сдвиг содержимого ячейки вправо.
-	сдвиг содержимого ячейки в конец строки.
+	перемещают содержимое ячейки влево или вправо.

**60. Что обозначает команда Shr bx?**

-	арифметический сдвиг содержимого регистра вправо.
-	арифметический сдвиг содержимого регистра влево.
-	логический сдвиг содержимого регистра влево.
+	логический сдвиг содержимого регистра вправо.

**61. Что обозначает команда Sar ax?**

+	арифметический сдвиг содержимого регистра вправо.
-	арифметический сдвиг содержимого регистра влево.
-	логический сдвиг содержимого регистра влево.
-	логический сдвиг содержимого регистра вправо.

**62. Что обозначает команда Sal bx?**

-	арифметический сдвиг содержимого регистра вправо.
+	арифметический сдвиг содержимого регистра влево.
-	логический сдвиг содержимого регистра влево.
-	логический сдвиг содержимого регистра вправо.

**63. Что обозначает команда Shl ax?**

-	арифметический сдвиг содержимого регистра вправо.
-	арифметический сдвиг содержимого регистра влево.
+	логический сдвиг содержимого регистра влево.
-	логический сдвиг содержимого регистра вправо.

**64. Что обозначает команда INC, AX?**

+	увеличивает содержимое регистра на единицу.
-	уменьшает содержимое регистра на единицу.
-	увеличивает содержимое регистра на две единицы.
-	уменьшает содержимое регистра на две единицы.

**65. Что обозначает команда DEC, AX?**

-	увеличивает содержимое регистра на единицу.
+	уменьшает содержимое регистра на единицу.
-	увеличивает содержимое регистра на две единицы.
-	уменьшает содержимое регистра на две единицы.

**66. Что обозначает команда inc?**

-	Уменьшение операнда.
+	Увеличение операнда.

-	вычитание второго операнда из другого.
-	изменение знака операнда.

#### **67. Что обозначает команда add?**

-	добавления операнда.
+	сложение двух операндов с записью результата.
-	вычитание второго операнда из другого.
-	изменение знака операнда.

#### **68. Что обозначает команда adc?**

-	добавления операнда.
-	сложение двух операндов с записью результата.
-	вычитание второго операнда из другого.
+	сложения с учетом переноса.

#### **69. Что обозначает команда sub?**

+	вычитание второго операнда из первого.
-	сложение двух операндов с записью результата.
-	вычитание второго операнда из другого.
-	сложения с учетом переноса.

#### **70. Что обозначает команда sbc?**

-	вычитание второго операнда из первого.
-	сложение двух операндов с записью результата.
+	вычитания с учетом займа.
-	сложения с учетом переноса?

#### **71. Что обозначает команда neg?**

+	изменение знака операнда.
-	сложение двух операндов с записью результата.
-	вычитания с учетом займа.
-	сложения с учетом переноса?

#### **72. Что обозначает команда Mul?**

-	команда деления.
-	сложение двух операндов с записью результата.
+	команда умножения.
-	сложения с учетом переноса?

#### **73. Что обозначает команда div?**

+	команда деления.
-	сложение двух операндов с записью результата.
-	команда умножения.
-	сложения с учетом переноса?

#### **74. Что обозначает команда imul?**

-	команда деления.
-	знаковое целочисленное деление..
-	команда умножения.
+	знаковое целочисленное умножение.

#### **75. Что обозначает команда idiv?**

-	команда деления.
+	знаковое целочисленное деление..
-	команда умножения.
-	знаковое целочисленное умножение.

#### **76. Что обозначает команда CBW?**

+	преобразовывает байт содержащийся в регистре AL в слово, помещаемое в регистр AX.
-	преобразовывает слово содержащееся в регистре AX в двойное слово, помещаемое в регистры DX:AX. Старшая часть значения разместится в DX, а младшая – в AX.
-	преобразовывает слово содержащееся в регистре AX в двойное слово, помещаемое в регистр EAX.
-	преобразовывает двойное слово содержащееся в EAX, в учетверенное слово помещаемое в регистры EDX:EAX.

#### **77. Что обозначает команда CWD?**

-	преобразовывает байт содержащийся в регистре AL в слово, помещаемое в регистр AX.
+	преобразовывает слово содержащееся в регистре AX в двойное слово, помещаемое в регистры DX:AX. Старшая часть значения разместится в DX, а младшая – в AX.
-	преобразовывает слово содержащееся в регистре AX в двойное слово, помещаемое в регистр EAX.
-	преобразовывает двойное слово содержащееся в EAX, в учетверенное слово помещаемое в регистры EDX:EAX.

#### **78. Что обозначает команда CWDE?**

-	преобразовывает байт содержащийся в регистре AL в слово, помещаемое в регистр AX.
-	преобразовывает слово содержащееся в регистре AX в двойное слово, помещаемое в регистры DX:AX. Старшая часть значения разместится в DX, а младшая – в AX.
+	преобразовывает слово содержащееся в регистре AX в двойное слово, помещаемое в регистр EAX.
-	преобразовывает двойное слово содержащееся в EAX, в учетверенное

	слово помещаемое в регистры EDX:EAX.
--	--------------------------------------

#### 79. Что обозначает команда CDQ?

-	преобразовывает байт содержащийся в регистре AL в слово, помещаемое в регистр AX.
-	преобразовывает слово содержащееся в регистре AX в двойное слово, помещаемое в регистры DX:AX. Старшая часть значения разместится в DX, а младшая – в AX.
-	преобразовывает слово содержащееся в регистре AX в двойное слово, помещаемое в регистр EAX.
+	преобразовывает двойное слово содержащееся в EAX, в учетверенное слово помещаемое в регистры EDX:EAX.

#### 80. На какие группы можно разделить команды MC?

+	команды пересылки данных.
+	арифметические команды.
+	команды сравнений чисел.
-	команды кодировки данных.

#### 81. Команды пересылки данных предназначены для?

+	загрузки чисел из оперативной памяти в числовые регистры.
-	загрузка числе из регистров в оперативную память.
+	записи данных из числовых регистров в оперативную память.
+	копирования данных из одного числового регистра в другой.

#### 82. Что обозначает команда fstsw?

-	загружает ЧПТ в стек FPU.
-	копирует значение из вершины стека сопроцессора в операнд-адресат.
-	меняет местами содержимое регистра st0 и другого регистра FPU.
+	чтение слова состояния в регистр или переменную.

#### 83. Что обозначает команда fld?

+	загружает ЧПТ в стек FPU.
-	копирует значение из вершины стека сопроцессора в операнд-адресат.
-	меняет местами содержимое регистра st0 и другого регистра FPU.
-	чтение слова состояния в регистр или переменную.

#### 84. Что обозначает команда fst?

-	загружает ЧПТ в стек FPU.
+	копирует значение из вершины стека сопроцессора в операнд адресат.
-	меняет местами содержимое регистра st0 и другого регистра FPU.
-	чтение слова состояния в регистр или переменную.

#### 85. Что обозначает команда fxch?

-	загружает ЧПТ в стек FPU.
---	---------------------------

-	копирует значение из вершины стека сопроцессора в операнд-адресат.
+	меняет местами содержимое регистра st0 и другого регистра FPU.
-	чтение слова состояния в регистр или переменную.

## **VII ВОПРОСЫ ДЛЯ ПОДГОТОВКИ К ЗАЧЕТУ**

по дисциплине "МИКРОПРОЦЕССОРЫ",  
по направлению подготовки 230700 – «Прикладная информатика» и  
230400 – «Информационные системы и технологии»

1. Характеристики микропроцессора. Структурная схема микропроцессора. Основные блоки.
2. Особенности реализации микропроцессоров Intel и AMD.
3. Режимы работы микропроцессора.
4. Чипсеты (наборы системной логики) для микропроцессоров Intel и AMD. Структура чипсета.
5. Программная модель микропроцессора.
6. Регистры общего назначения. Сегментные регистры.
7. Регистры состояния и управления. Флаги.
8. Организация памяти и режимы работы микропроцессора. Понятие сегментации. Сегментированная модель памяти.
9. Формирование физического адреса в реальном режиме.
10. Жизненный цикл программы на языке ассемблера.
11. Трансляция программы. Компоновка. Отладка.
12. Система команд микропроцессора.
13. Структура машинной команды. Способы задания операндов.
14. Функциональная классификация машинных команд.
15. Директивы сегментации: стандартные и упрощенные.
16. Модели памяти при использовании упрощенных директив сегментации.
17. Простые типы данных ассемблера. Директивы определения данных.
18. Команды обмена данными. Пересылка данных.
19. Ввод-вывод в порт.
20. Организация циклов.
21. Безусловные переходы.
22. Условные переходы.
23. Работа с адресами и указателями.
24. Преобразование данных.
25. Организация стека. Команды работы со стеком.
26. Арифметические команды. Форматы целых чисел в ассемблере. BCD-числа.
27. Сложение двоичных чисел без знака. Вычитание двоичных чисел без знака.

28. Сложение двоичных чисел со знаком. Вычитание двоичных чисел со знаком.
29. Умножение двоичных чисел без знака. Деление двоичных чисел без знака.
30. Умножение двоичных чисел со знаком. Деление двоичных чисел со знаком.
31. Сложение неупакованных BCD-чисел. Вычитание неупакованных BCD-чисел.
32. Умножение неупакованных BCD-чисел. Деление неупакованных BCD-чисел.
33. Логические команды. Условные и безусловные переходы.
34. Флаги и команды условных переходов. Организация циклов.
35. Процедуры. Основные понятия.
36. Организация процедур. Передача параметров в процедуру.
37. Концепция прерываний. Классификация прерываний.
38. Внешние прерывания.
39. Внутренние прерывания.
40. Программные прерывания.
41. Обработка прерываний в реальном режиме.
42. Обработка прерываний в защищенном режиме.
43. Сложные структуры данных. Способы организации массивов.
44. Цепочечные команды.
45. Макросредства языка ассемблера.
46. Директивы компиляции по условию. Директивы генерации ошибок.
47. Ассемблер и языки высокого уровня.
48. Интерфейс с языками высокого уровня.
49. Ассемблерные вставки на C и Pascal.
50. Использование процедур на ассемблере.
51. Защищенный режим работы микропроцессора.
52. Характеристики защищенного режима работы микропроцессоров Intel.
53. Сегментные регистры и структуры данных защищенного режима.
54. Перевод микропроцессора в защищенный режим.
55. Особенности работы в защищенном режиме.
56. Ассемблер для Windows. Структура Windows – программы на языке ассемблера.
57. Программирование на ассемблере с использованием Win32 API.
58. Система команд сопроцессора.
59. Исключения сопроцессора и их обработка.
60. MMX – расширение архитектуры микропроцессора.

## ГЛОССАРИЙ

*Адресация памяти (addressing mode)* – осуществление ссылки (обращение) к устройству или элементу данных по его адресу; установление соответствия между множеством однотипных объектов и множеством их адресов; метод идентификации местоположения объекта

*Активационная запись (activation record)* – область стека, заполняемая при вызове процедуры

*Ассемблер (assembly language)* – язык программирования низкого уровня

*Ассемблер (assembler)* – компилятор с языка ассемблера

*Байт (byte)* – тип данных, имеющий размер 8 бит, минимальная адресуемая единица памяти

*Бит (bit)* – минимальная единица измерения информации

*«Всплывающая» программа (popup program)* – резидентная программа, активирующаяся по нажатию определенной «горячей» клавиши

*«Горячая» клавиша (hotkey)* – клавиша или комбинация клавиш, используемая не для ввода символов, а для вызова программ и подобных необычных действий

*Двойное слово (double word)* – тип данных, имеющий размер 32 бита

*Дескриптор (descriptor)* – восьмибайтная структура, хранящаяся в одной из таблиц GDT, LDT или IDT и описывающая сегмент или шлюз

*Директива (directive)* – команда ассемблеру, которая не соответствует командам процессора

*Драйвер (driver)* – служебная программа, выполняющая функции посредника между операционной системой и внешним устройством

*Защищенный режим (protected mode)* – режим процессора, в котором действуют механизмы защиты, сегментная адресация с дескрипторами и селекторами и страничная адресация

*Задача (task)* – программа, модуль или другой участок кода программы, который можно запустить, выполнять, отложить и завершить

*Идентификатор (handle или identifier)* – число (если handle) или переменная другого типа, используемая для идентификации того или иного ресурса

*Исключение (exception)* – событие, при котором выполнение программы прекращается и управление передается обработчику исключения



*Итерация (iteration)* — организация обработки данных, при которой действия повторяются многократно, не приводя при этом к вызовам самих себя (не путать с рекурсией)

*Код (code)* – исполнимая часть программы (обычная программа состоит из кода, данных и стека)

*Команда перехода (branch)*– команда процессора, которая нарушает естественный порядок исполнения команд, вынуждая выбирать и исполнять последующие команды с произвольно заданного адреса

*Компилятор (compiler)* – программа, преобразующая текст, написанный на понятном человеку языке программирования, в исполнимый файл

*Конвейер (pipe)* – последовательность блоков процессора, которая задействуется при выполнении команды

*Конвенция (convention)* – договоренность о передаче параметров между процедурами

*Конечный автомат (finite state machine)* – программа, которая может переключаться между различными состояниями и выполнять в разных состояниях разные действия

*Кэш (cache)* – быстрая память, используемая для буферизации обращений к основной памяти

*Лимит (limit)* – поле дескриптора (равно размеру сегмента минус 1)

*Линейный адрес (linear address)* — адрес, получаемый сложением смещения и базы сегмента

*Ловушка (trap)* – исключение, происходящее после вызвавшей его команды

*Локальная метка (local label)* – это метка, которая известна только внутри того оператора Asm, где она была определена

*Метка (label)* – идентификатор, связанный с адресом в программе

*Нить (thread)* – процесс, данные и код которого совпадают с данными и кодом других процессов

*Нереальный режим (unreal mode)* – реальный режим с границами сегментов по 4 Гб

*Операнд (operand)* – параметр, передаваемый команде процессора

*Описатель носителя (media descriptor)* – байт, используемый DOS для идентификации типа носителя (обычно не используется)

*Останов (abort)* – исключение, происходящее асинхронно

*Отложенное вычисление (lazy evaluation)* – вычисление, которое выполняется, только если реально требуется его результат

*Очередь предвыборки (prefetch queue)* – буфер, из которого команды передаются на расшифровку и выполнение

*Ошибка (fault)* – исключение, происходящее перед вызвавшей его командой

*Пиксель (pixel)* – минимальный элемент растрового изображения

*Повторная входимость (reentrancy)* – возможность запуска процедуры из обработчика прерывания, прервавшего выполнение этой же процедуры

*Подчиненный сегмент (conforming segment)* – сегмент, на который можно передавать управление программам с более низким уровнем привилегий

*Препроцессор (preprocessor)* – это компьютерная программа, принимающая данные на входе и выдающая данные, предназначенные для входа другой программы (например, компилятора). О данных на выходе препроцессора говорят, что они находятся в препроцессированной форме, пригодной для обработки последующими программами (компилятор)

*Прерывание (interrupt)* – сигнал от внешнего устройства, приводящий к прерыванию выполнения текущей программы и передаче управления специальной программе-обработчику (см. ловушка)

*Разворачивание циклов (loop unrolling)* – превращение циклов, выполняющихся известное число раз, в линейный участок кода

*Реальный режим (real mode)* – режим, в котором процессор ведет себя идентично 8086 – адресация не выше одного мегабайта памяти, размер всех сегментов ограничен и равен 64 Кб, только 16-битный режим

*Резидентная программа (resident program)* – программа, остающаяся в памяти после возврата управления в DOS

*Сегмент (segment)* – элемент сегментной адресации в памяти или участок программы для DOS/Windows

*Селектор (selector)* – число, хранящееся в сегментном регистре

*Секция (section)* – участок программы для UNIX

*Скан-код (scan-code)* – любой код, посылаемый клавиатурой

*Слово (word)* – тип данных, имеющий размер 16 бит

*Смещение (offset)* – относительный адрес, отсчитываемый от начала сегмента

*Стековый кадр (stack frame)* – область стека, занимаемая параметрами процедуры, активационной записью и локальными переменными или только локальными переменными

*Страничная адресация (pagination)* – механизм адресации, в котором линейное адресное пространство разделяется на страницы, которые могут располагаться в разных областях памяти или вообще отсутствовать

*Таблица переходов (jumptable)* – массив адресов процедур для косвенного перехода на процедуру с известным номером

*Шлюз (gate)* – структура данных, позволяющая осуществлять передачу управления между разными уровнями привилегий в защищенном режиме

## СПИСОК ЛИТЕРАТУРЫ

### Основная литература:

1. Александров Е., Грушвицкий Р., Куприянов М. Микропроцессорные системы. – М.: – Политехника, 2008. – 543 с.
2. Пильщиков В. Язык макроассемблера IBM PC. – М.: – Академия, 2008. – 412 с.
3. Пирогов В. Ассемблер для Windows. – М.: – Вильямс, 2007. – 522 с.
4. Юров В. Ю. Ассемблер. – СПб.: Питер, 2007. – 624 с.
5. Григорьев В.Л. Архитектура и программирование арифметического сопроцессора. М., 1991. –326 с.
6. Лямлин Л.В. Макроассемблер MASM. М.,1994. – 254 с.

### Дополнительная литература:

1. К. Ирвин. Язык Ассемблера для процессоров Intel. М.: – Вильямс, 2006. 616 с.
2. Корнеев В., Киселев А. Современные микропроцессоры. – СПб.: – BHV-СПб, 2007. – 448 с.
3. Новиков Ю. Основы микропроцессорной техники: Курс лекций. – М.: – Интернет-университет информационных технологий, 2007. – 436 с.
4. Финогенов К.Г., Рудаков П.И. Язык Ассемблера: уроки программирования. М.: – Диалог-МИФИ, 2005. – 235 с.
5. Белов А.В. Самоучитель по микропроцессорной технике. – М.: – Наука и Техника, 2007. – 224 с.
6. Кузин А.В., Жаворонков М.А. Микропроцессорная техника. Учебник. – М.: – Академия, 2008. – 314 с.

# П Р И Л О Ж Е Н И Я

## СПИСОК ЧАСТО ИСПОЛЬЗУЕМЫХ КОМАНД

Группа команд	Назначение
Команды перемещения значений	MOV — скопировать
	XCHG — поменять местами значения
	PUSH — сохранить в стеке
	POP — извлечь из стека
Арифметические команды	ADD — сложение
	SUB — вычитание
	MUL — умножение
	DIV — деление
	INC — инкремент (увеличение на 1)
	DEC — декремент (уменьшение на 1)
Логические команды	AND — логическое И (логическое умножение)
	OR — логическое ИЛИ (логическая сумма)
	XOR — исключительное ИЛИ
	NOT — отрицание
Сравнение	CMP — сравнение
	TEST — поразрядное сравнение
Сдвиг и ротация	SHR — логический (поразрядный) сдвиг вправо
	SHL — логический (поразрядный) сдвиг влево
	RCR — ротация через флаг переноса вправо
	RCL — ротация через флаг переноса влево
Команды перехода	JMP — безусловный переход
	LOOP — переход, пока (E)CX не сравняется с 0
Условные переходы	JZ — если флаг нуля (ZF) установлен
	JC — если флаг переноса (CF) установлен
	JNZ — если флаг нуля (ZF) не установлен
	JNC — если флаг переноса (CF) не установлен
Подпрограммы	CALL — вызов подпрограммы
	RET — возврат из подпрограммы
	INT — вызов прерывания
Операции над строками	REP — повторять следующую команду, пока (E)CX не 0
	MOVSx — копирование строк
	CMPSx — сравнение строк
	SCASx — поиск в строке

Учебное издание

**Параскевов** Александр Владимирович  
старший преподаватель  
кафедрой компьютерных технологий и систем

**Жмурко** Даниил Юрьевич  
доцент кафедры компьютерных технологий  
и систем, к.э.н.

## МИКРОПРОЦЕССОРЫ

Лабораторный практикум

По специальности 230700.62 – «Прикладная информатика» и  
230400.62 – «Информационные системы и технологии»

В авторской редакции

Подписано в печать 16.03.2013 г. Формат  $60 \times 84 \frac{1}{16}$ .

Тираж 100 экз. Усл. печ. л. – 18. Уч.-изд. л. – 13.

Заказ № .

Редакционный отдел и типография  
Кубанского государственного аграрного университета,  
350044, г. Краснодар, ул. Калинина, 13