

ПРИДНЕСТРОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
им. Т.Г. Шевченко

Физико-математический факультет

Кафедра прикладной математики и информатики

**Программирование на языке ассемблера
Часть I**

Лабораторный практикум

г. Тирасполь, 2017

УДК 004.4`424 (075.8)
ББК 3 973.21я73
П78

Составитель:

А. В. Бугаенко, ст. преп. каф. прикладной математики и информатики

Рецензенты:

Е. В. Калинин, ст. преподаватель кафедры прикладной математики и информатики физико-математического факультета ПГУ им. Т.Г. Шевченко

Д. А. Марков, начальник УИР ПГУ им. Т.Г. Шевченко, к.ф.-м.н

Программирование на языке ассемблера Ч.I: Лабораторный практикум / Сост.: Бугаенко А. В., – Тирасполь, 2017 г. – 100 стр.

Лабораторный практикум содержит описание лабораторных работ, теоретическую часть, практическую часть, десять вариантов индивидуальных заданий в каждой лабораторной работе и методические рекомендации к их выполнению по курсам «Физические основы построения ЭВМ» и «Архитектура компьютеров».

Предназначено для студентов очного и заочного отделений физико-математического факультета направлений подготовки «Прикладная математика и информатика», «Педагогическое образование с двумя профилями подготовки» и смежных направлений, а также всем желающим научиться программировать на языке ассемблера.

УДК 004.4`424 (075.8)
ББК 3 973.21я73
П78

Рекомендовано Научно-методическим советом ПГУ им. Т. Г. Шевченко

© Составление:
Бугаенко А. В., 2017 г.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ	4
ЛАБОРАТОРНАЯ РАБОТА №1	5
ЛАБОРАТОРНАЯ РАБОТА №2	38
ЛАБОРАТОРНАЯ РАБОТА №3	50
ЛАБОРАТОРНАЯ РАБОТА №4	63
ЛАБОРАТОРНАЯ РАБОТА №5	74
ЛАБОРАТОРНАЯ РАБОТА №6	86
ЛИТЕРАТУРА.....	97
ПРИЛОЖЕНИЕ 1	98
ПРИЛОЖЕНИЕ 2	99

ВВЕДЕНИЕ

«Только язык ассемблера позволяет использовать мощность вашего ПК в полном объеме...»

Питер Нортон

Язык ассемблера является символическим представлением машинного языка, он неразрывно связан с архитектурой самого процессора. По мере внесения изменений в архитектуру процессора совершенствуется и сам язык ассемблера. По существу, ассемблер является языком, на котором "говорит" процессор! Таким образом, ассемблер имеет и всегда будет иметь одно фундаментальное преимущество перед языками высокого уровня: программы на ассемблере — самые быстрые и самые маленькие.

В ходе выполнения лабораторных работ постепенно происходит овладение технологиями программирования на языке ассемблера. Изложение материала сопровождается примерами и способствует более глубокому его пониманию. С целью самоконтроля в конце каждой лабораторной работы имеется десять вариантов индивидуальных заданий. Лабораторный практикум посвящен работе с целочисленной арифметикой, то есть целочисленным арифметическим командам языка, командам пересылки данных, процедурам в ассемблере, а также работе с массивами в ассемблере. Материал снабжен многочисленными иллюстрациями и сведениями справочного характера. Во многих случаях приводятся рекомендации практического характера по оптимизации выполнения часто встречающихся действий. Лабораторный практикум выполнен на требуемом уровне и соответствует государственному образовательному стандарту высшего образования по направлениям подготовки «Прикладная математика и информатика», «Педагогическое образование с двумя профилями подготовки» и может быть использован при обучении студентов этих и смежных направлений.

ЛАБОРАТОРНАЯ РАБОТА №1

ЯЗЫК АССЕМБЛЕРА. АРИФМЕТИЧЕСКИЕ ОПЕРАЦИИ В АССЕМБЛЕРЕ

ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

ЯЗЫК АССЕМБЛЕРА

Компьютер понимает только один язык – язык машинных команд. Еще в 50-е годы программисты стали использовать для программирования символический аналог машинного языка, который называли *язык ассемблера*. Язык ассемблера фактически представляет собой машинный язык (язык процессора), где коды команд заменены их мнемоническими обозначениями, вместо адресов памяти могут записываться имена переменных и символьные метки, а числовые константы пишутся в десятичной форме. Преобразование программы с языка ассемблера в машинный код (т.е. компиляцию) выполняет сам компьютер с помощью программы-ассемблера, откуда и происходит название языка (*assembler* – «сборщик» с англ. яз.). По структуре и размеру ассемблерная программа близка к машинной: каждый из ее операторов преобразуется в одну машинную команду (это не относится к макросам). Человек лучше ориентируется в именах, чем в числах, поэтому язык ассемблера проще для понимания, чем машинный язык. Таким образом, можно сказать, что ассемблер — это удобная человеку запись процессорных команд.

Алгоритмические языки (первые - 1960-е г.) дают более высокую степень автоматизации программирования. Они служат для записи непосредственно алгоритмов и не зависят от системы команд, т.е. от архитектуры процессора конкретного типа. Один оператор алгоритмического языка транслируется с помощью компилятора в среднем в десяток машинных команд. Трудоемкость программирования на них существенно ниже, чем при использовании языка ассемблера, и поэтому они называются языками высокого уровня. На рис.1 приведена классификация языков программирования, показывающая место языка ассемблера. Основным инструментом программиста служат

языки высокого уровня, а к языку ассемблера программисты обращаются при необходимости увеличить быстродействие программы или уменьшить объем занимаемой памяти, то есть при программировании на языке ассемблера доступны все возможности архитектуры процессора, и программист может их использовать для реализации приемов ускорения программы.

На языке ассемблера можно написать столь же эффективную по размеру и времени выполнения программу, как и программу на языке машинных команд. Это преимущество отсутствует у языков высокого уровня.

Языки программирования	Машинно-ориентированные (низкого уровня)	Машинные коды
		Язык ассемблера
Алгоритмические (высокого уровня)		Объектные (C++, C#, Delphi)
		Декларативные
		Скриптовые (Shell, PHP, Python)
		Процедурные (Фортран, Паскаль, C)

Рисунок 1. Классификация языков программирования

ОПТИМИЗАЦИЯ АССЕМБЛЕРНОГО КОДА

Одно из наиболее действенных средств оптимизации программ – применение языка ассемблера. Во-первых, использование языка ассемблера решает проблему избыточности программного кода, так как ассемблерный код более компактен, чем его аналог на языке высокого уровня. А также язык ассемблера позволяет разрабатывать короткий и эффективный код. Во-вторых, программный модуль на ассемблере обладает более высоким быстродействием, чем написанный на языке

высокого уровня, так как это связано с меньшим числом команд, которые требуются для реализации фрагмента кода. Меньшее число команд быстрее выполняется центральным процессором и, следовательно, повышает производительность программы.

Отметим, что из-за кажущейся сложности языка немногие программисты применяют его на практике.

ТИПЫ ДАННЫХ И ИХ ПРЕДСТАВЛЕНИЕ В КОМПЬЮТЕРЕ

Главный принцип хранения данных в компьютере – намагничивание и размагничивание одной дорожки. Одна микросхема памяти – это огромное количество дорожек. Предположим, что ноль будет обозначаться как 0000 (четыре нуля), 1 – 0001, 2 – 0010, и так далее. «Нули» и «единицы» – это биты. Один бит может иметь только два значения – 0 или 1, то есть, размагничена или намагничена та или иная дорожка. Компьютер хранит данные в памяти именно так. Для обозначения какого-нибудь символа (цифры, буквы, запятой, точки) компьютер использует определенное количество бит. Компьютер «распознает» 256 (от 0 до 255) различных символов по коду. Для представления символа с максимально возможным кодом (255) нужно 8 бит – один байт. Таким образом, байт – это минимальный объем данных, который реально может использовать компьютерная программа. Даже для изменения значения одного бита в памяти надо сначала считать байт, содержащий его. Биты в байте нумеруют справа налево, от нуля до семи, нулевой бит часто называют младшим битом, а седьмой – старшим (см. рис. 2).



7 6 5 4 3 2 1 0

Рисунок 2. Байт

Большие группы битов называются словом (word) или двойным словом (dword — double word). Относительно PC-совместимых компьютеров мы можем сказать следующее:

1 байт = 8 бит

1 слово (word) = 2 байта = 16 бит

1 двойное слово (dword) = 4 байта = 32 бит

1 учетверенное слово (qword) = 8 байт = 64 бита

Байт может содержать число в диапазоне 0 — 255 (то есть $2^8=256$ различных чисел). В большинстве случаев этого значения недостаточно, поэтому используется следующая единица данных — слово. Слово может содержать число в диапазоне 0 — 65 535 (то есть $2^{16} = 65\,536$ различных значений). Двойное слово имеет диапазон значений 0 — 4 294 967 295 ($2^{32} = 4\,294\,967\,296$ значений).

Байт используют для представления целых чисел от 0 до 255 (тип unsigned char в C), целых чисел со знаком от -128 до +127 (тип signed char в C), набора символов ASCII (тип char в C) или переменных, принимающих менее 256 значений, например для представления десятичных чисел от 0 до 99.

Слова используют для представления целых чисел без знака со значениями 0 – 65535 (тип unsigned short в C), целых чисел со знаком от -32 768 до +32 767 (тип short int в C), адресов сегментов и смещений при 16-битной адресации.

Память с точки зрения процессора представляет собой последовательность байтов, каждому из которых присвоен уникальный адрес со значениями от 0 до $2^{32}-1$ (4 Гб). Программы же могут работать с памятью как с одним непрерывным массивом (модель памяти flat) или как с несколькими массивами (сегментированные модели памяти). Во втором случае для задания адреса любого байта требуется два числа - адрес начала массива и адрес искомого байта внутри массива. Помимо основной памяти программы могут использовать регистры – специальные ячейки памяти, расположенные физически внутри процессора, доступ к которым осуществляется не по адресам, а по именам.

РЕГИСТРЫ ПРОЦЕССОРА

Регистры общего назначения

К регистрам общего назначения относят 32-битные регистры EAX, EBX, ECX и EDX (Аккумулятор, База, Счетчик, Регистр данных). Данные регистры могут использоваться без

ограничений для любых целей - временного хранения данных, аргументов или результатов различных операций. Аккумулятор необходим для хранения результата действий, выполняемых над двумя операндами, регистр данных в этих случаях получает старшую часть результата, если он не уместится в аккумулятор, регистр-счетчик работает как счетчик в циклах и строковых операциях, а регистр-база - при так называемой адресации по базе.

Регистры EAX, EBX, ECX и EDX могут быть разделены на две части – 16-разрядные регистры AX, BX, CX, DX соответственно и верхние 16 битов, которые никак не называются. В свою очередь, регистры AX, BX, CX, DX могут быть разделены на два 8-битных регистра – AH и AL, BH и BL, CH и CL, DH и DL соответственно. Например, если занести в регистр значение 0×12345678, то регистр AX будет содержать значение 0×5678, то есть 0×56 в AH и 0×78 в AL, а значение 0×1234 будет помещено в верхнюю часть регистра EAX.

Индексные регистры

К индексным регистрам процессора относят **ESI** (индекс источника (Source Index)), **EDI** (индекс приемника (Destination Index)) или SI и DI для 16-разрядных действий. Обычно эти регистры используются для адресации памяти: обращения к массивам, индексирования и т.д., но регистры ESI и EDI могут содержать и произвольные данные, то есть они программно доступны и их содержание может быть изменено программистом. У регистров ESI, EDI и EBP существуют только в 16-разрядная и 32-разрядная версии.

Важно! Другие регистры лучше «руками не трогать».

Регистр базового указателя

Регистр **EBP** – указатель базы (Base Pointer), облегчает доступ к параметрам: данным и адресам, переданным через стек. Может использоваться для расширенной индексной адресации и в операциях сложения и вычитания.

Сегментные регистры

Данную группу регистров относят к регистрам состояния. Регистры из этой группы используются при вычислении реального адреса (адреса, который будет передан на шину адреса). Процесс вычисления реального адреса зависит от режима процессора (реальный или защищенный). Сегментные регистры только 16-разрядные. Названия этих регистров соответствуют выполняемым функциям:

- CS (Code Segment, сегмент кода) вместе с EIP (IP) определяют адрес памяти, откуда нужно прочитать следующую инструкцию;
- SS (Stack Segment, сегмент стека) в паре с ESP (SS:SP) указывают на вершину стека.
- Сегментные регистры DS, ES, FS, и GS (Data, Extra, F и G сегменты) используются для адресации данных в памяти.

Регистры состояния и управления

Регистр ESP (SP) — это указатель памяти, который указывает на вершину стека. Также программно не может быть изменен регистр EIP (IP, Instruction Pointer) — указатель команд. Этот регистр указывает на инструкцию, которая будет выполнена следующей. Значение этого регистра изменяется непосредственно контроллером процессора согласно инструкциям, полученным из памяти.

Регистр флагов (регистр признаков) — FLAGS, он сигнализирует процессору о его состоянии или о том, как выполнялась та или иная арифметическая или логическая команда. Он состоит из одnorазрядных флагов. Флаг – это бит, принимающий значение, если он установлен, и 0, если он сброшен. За битами регистров флагов закреплены соответствующие имена:

Таблица 1
Регистры флагов

Регистр флагов (FLAGS)															
Биты															
5	4	3	2	1	0										
				F	F	F	F	F	F		F		F		F

Наиболее важные из них:

- Признак нуля ZF (Zero Flag) — 1, если результат предыдущей операции равен нулю.
- Признак знака SF (Sign Flag) — 1, если результат предыдущей операции отрицательный.
- Признак переполнения OF (Overflow Flag) — 1, если при выполнении предыдущей операции произошло переполнение (overflow), то есть результат операции больше, чем зарезервированная для него память.
- Признак переноса CF (Carry Flag) — 1, если бит был «перенесен» и стал битом более высокого порядка.
- Признак прерывания IF (Interrupt Flag) — 1, если прерывания процессора разрешены.
- Признак направления DF (Direction Flag) — используется для обработки строк, мы рассмотрим подробнее этот регистр в шестой главе.

Другие регистры процессора относятся к работе в защищенном режиме.

СПОСОБЫ АДРЕСАЦИИ

Адрес, как и команда – это число. Адресам присваивают символические обозначения, которые называют переменными (или указателями), чтобы не запоминать адреса всех «переменных», используемых в программе.

Есть команды, и есть операнды. Операнд – тот объект, над которым будет производиться операция. В команде мы указываем адрес.

При использовании косвенного операнда адрес в памяти, по которому находится нужное значение, записывается в квадратных скобках: [адрес]. Если мы используем указатель, то есть символическое представление адреса, то в листинге машинного кода мы увидим, что указатель был заменен реальным значением адреса. Можно также указать точный адрес памяти, например, [0x594F]. Процессор поддерживает сложные способы адресации. Далее рассмотрены все существующие способы задания адреса хранения операндов - *способы адресации*.

Регистровая адресация

Операнды могут располагаться в любых регистрах общего назначения и сегментных регистрах. Для этого в тексте программы указывается название соответствующего регистра, например: команда, копирующая в регистр AX содержимое регистра BX, записывается как *mov ax, bx*.

Непосредственная адресация

Некоторые команды (все арифметические, кроме деления) позволяют указывать один из операндов непосредственно в тексте программы (он может иметь любой смысл (число, адрес, код ASCII), а также быть представлен в виде символического обозначения). Например: команда *mov ax, 2* помещает в регистр AX число 2.

Прямая адресация

При прямой адресации эффективный адрес содержится в самой команде и для его формирования не используется никаких дополнительных источников или регистров. Эффективный адрес берется непосредственно из поля смещения машинной команды.

При прямой адресации исполнительный адрес является составной частью команды. Обычно прямая адресация применяется, если операндом служит метка. Например, *mov ax, table*.

Существует абсолютная и относительная прямая адресация.

- Относительная прямая адресация. Используется в командах перехода
- Абсолютная прямая адресация. В этом случае эффективный адрес является частью машинной команды, но формируется этот адрес только из значения поля смещения в команде

Косвенная адресация

По аналогии с регистровыми и непосредственными операндами адрес операнда в памяти также можно не указывать, а хранить в любом регистре. При косвенной адресации

исполнительный адрес операнда содержится в базовом регистре BX, в регистре указателя BP или индексном регистре (SI или DI).

Косвенные регистровые операнды надо заключать в квадратные скобки, чтобы отличить их от регистровых операндов. Например, `mov ax,[bx]`.

Для загрузки адреса в BX используется операция OFFSET(смещение) к адресу ячейки памяти.

```
mov bx, offset table  
mov ax, [bx]
```

Адресация по базе со сдвигом

Теперь скомбинируем два предыдущих метода адресации. Следующая команда `mov ax,[bx+2]` помещает в регистр AX слово, которое есть в сегменте, указанном в DS, со смещением на два больше, чем число из BX. Так как слово занимает ровно 2 байта, эта команда поместила в AX слово, непосредственно следующее за тем, которое было в предыдущем примере. Такая форма адресации используется в тех случаях, когда в регистре находится адрес начала структуры данных, а доступ надо осуществить к какому-нибудь ее элементу. Еще один вариант применения адресации по базе со сдвигом — доступ из подпрограммы к параметрам, переданным в стеке, используя регистр BP (EBP) в качестве базы и номер параметра в качестве смещения. Другие допустимые формы записи этого способа адресации:

```
mov ax, [bp]+2  
mov ax, 2[bp]
```

Важно!!! С помощью этого метода разрешается организовывать доступ к одномерным массивам байтов: смещение соответствует адресу начала массива, а число в регистре - индексу элемента массива, который надо считать. Очевидно, что, если массив состоит не из байтов, а из слов, придется умножить базовый регистр на два, а если из двойных слов - на четыре. Для этого предусмотрен специальный метод - косвенная адресация.

Косвенная адресация с масштабированием

Этот метод адресации полностью идентичен предыдущему, однако с его помощью можно прочитать элемент массива слов, двойных слов или учетверенных слов, просто поместив номер элемента в регистр:

```
mov ax, [esi*2]+2
```

Множитель, который равен 1, 2, 4 или 8, соответствует размеру элемента массива — байту, слову, двойному или учетверенному слову. Из регистров в этом варианте адресации можно использовать только EAX, EBX, ECX, EDX, ESI, EDI, EBP, ESP, но не SI, DI, BP или SP.

Адресация по базе с индексированием

В этом методе адресации смещение операнда в памяти вычисляется как сумма чисел, содержащихся в двух регистрах, и смещения, если оно указано. Все перечисленные ниже команды представляют собой разные формы записи одного и того же действия:

```
mov ax, [bx+si+2]
mov ax, [bx][si]+2
mov ax, [bx+2][si]
mov ax, [bx][si+2]
mov ax, 2[bx][si]
```

В регистр AX помещается слово из ячейки памяти со смещением, равным сумме чисел, содержащихся в BX, SI, и числа 2. Из 16-битных регистров так можно складывать только BX + SI, BX + DI, BP + SI и BP + DI, а из 32-битных — все восемь регистров. Как и для прямой адресации, вместо непосредственного указания числа разрешено использовать имя переменной, заданной одной из директив определения данных.

Адресация по базе с индексированием и масштабированием

Это самая полная схема адресации, в которую входят все случаи, рассмотренные ранее как частные. Полный адрес операнда можно записать как выражение, представленное на рис. 3.

	EAX		EAX		
CS	EBX		EBX		
SS	ECX		ECX	1	
DS	EDX	+	EDX	*	2 + смещение
ES	EBP		EBP	3	
FS	ESP			4	
OS	EDI		ESI		
	ESI		EDI		

Рисунок 3. Полная форма адресации

Смещение может быть байтом или двойным словом. Если ESP или EBP используются в роли базового регистра, селектор сегмента операнда берется по умолчанию из регистра SS, во всех остальных случаях - из DS.

ОСНОВНЫЕ КОМАНДЫ ЦЕЛОЧИСЛЕННОЙ АРИФМЕТИКИ

Для описания команд языка ассемблер используется следующий общий формат:

имя_команды [подсказка] операнды

Необязательная подсказка указывает компилятору требуемый размер операнда. Ее значением может быть слово BYTE (8-битный операнд), WORD (16-битный) или DWORD (32-битный).

Операнд может быть как один, так и два. Команды, содержащие два операнда, имеют следующий формат:

имя_команды приемник, источник

Причем, если команда содержит два операнда, они почти всегда интерпретируются так: первый операнд – приемник, а второй – источник.

Важно!!! В данной работе приведены не все команды, относящиеся к различным категориям, а только основные или те которые будут нами в дальнейшем использоваться.

Команда пересылки данных

Синтаксис:

MOV приемник, источник

Базовая команда пересылки данных. Копирует содержимое источника в приемник, источник не изменяется. Команда MOV действует аналогично операторам присваивания из языков высокого уровня, то есть команда

mov ax,bx

эквивалентна выражению

ax := bx;

языка Pascal или

ax = bx;

языка C, за исключением того, что команда ассемблера позволяет работать не только с переменными в памяти, но и со всеми регистрами процессора.

В качестве источника для *MOV* могут использоваться:

- число (непосредственный операнд),
- регистр общего назначения;
- сегментный регистр;
- переменная (то есть операнд, находящийся в памяти).

В качестве приемника:

- регистр общего назначения;
- сегментный регистр (кроме CS);
- переменная.

Важно!!! Оба операнда должны быть одного и того же размера - байт, слово или двойное слово.

Важно!!! Нельзя выполнять пересылку данных с помощью *MOV* из одной переменной в другую, из одного сегментного регистра в другой и нельзя помещать в сегментный регистр непосредственный операнд - эти операции выполняют двумя командами *MOV* (из сегментного регистра в обычный и уже из него в другой сегментный) или парой команд *PUSH/POP* (см. ниже).

Пример

```
mov eax, 5      ;в регистр eax помещается число 5
mov ebx, eax    ;в регистр ebx помещается
                ;значение регистра eax
mov ecx, [a]    ;в регистр ecx помещается
                ;значение переменной a
mov [a], eax    ;переменная a получает значение
                ;регистра eax
```

Важно!!! Другие варианты недопустимы!

Важно!!! В языке ассемблер комментарий обозначается знаком «;».

Команда обмена данными

Синтаксис:

XCHG приемник, источник

Данная команда меняет местами содержимое двух операндов. XCHG можно выполнять над двумя регистрами или над регистром и переменной.

Пример

```
mov esx, [a] ;в регистр esx помещается  
;значение переменной а и наоборот  
mov [a], eax ;переменная а получает значение  
;регистра eax и наоборот
```

Команды работы со стеком

Синтаксис:

PUSH источник

POP приемник

Команда PUSH помещает содержимое источника в стек.

В качестве источника для PUSH могут использоваться:

- регистр;
- сегментный регистр;
- непосредственный операнд;
- переменная.

Команда POP помещает в приемник слово или двойное слово, находящееся в вершине стека, увеличивая ESP на 2 или 4 соответственно. POP выполняет действие, полностью обратное PUSH.

В качестве приемника для POP могут использоваться:

- регистр общего назначения;
- сегментный регистр, кроме CS;
- переменная.

Синтаксис:

PUSHA

POPA

Команда PUSHA запоминает все регистры в стеке строго в следующей последовательности: AX, CX, DX, BX, SP, BP, SI, DI.

Команда POPA, наоборот, извлекает все эти регистры из стека: DI, SI, BP, SP, BX, DX, CX, AX.

Важно!!! Данным командам операнды не нужны.

Арифметические команды

Команды сложения

Синтаксис:

ADD приемник, источник

Команда ADD выполняет арифметическое сложение приемника и источника, помещает сумму в приемник, предыдущее значение которого теряется, не изменяя содержимое источника.

В качестве источника для ADD могут использоваться:

- число (непосредственный операнд),
- регистр;
- переменная.

В качестве приемника для ADD могут использоваться:

- регистр;
- переменная.

Важно!!! Нельзя использовать переменную одновременно и для источника, и для приемника.

Пример

```
mov eax, 5      ;помещаем число 5 в регистр eax
mov ebx, 6      ;помещаем число 6 в регистр ebx
add ebx, eax    ;складываем значение ebx и eax, а
                ;результат помещаем в ebx, то
                ;есть ebx будет равен 11, а eax
                ;не изменится
add eax, ebx    ;складываем значение eax (5) и
                ;ebx (11), а результат
                ;помещаем в eax, то есть eax
                ;будет равен 16, а ebx не
                ;изменится - 11
```

Для того чтобы избежать уничтожения исходных значений в регистре ebx, необходимо пересохранить значение данного регистра в другом месте.

Пример

```
mov eax, 5  
mov ebx, 6  
mov ecx, ebx  
add ecx, eax  
add ecx, eax
```

В результате получим значение 16, но в регистре ecx.

Важно!!! Команда *ADD* никак не различает числа со знаком и без знака, но, употребляя значения флагов *CF*, *OF* и *SF*, разрешается применять ее и для тех, и для других.

Синтаксис:

ADC приемник, источник

Назначение данной команды – сложение с переносом. Команда *ADC* аналогична *ADD*, но при этом выполняет арифметическое сложение приемника, источника и флага *CF*.

Синтаксис:

XADD приемник, источник

Выполняет сложение, помещает содержимое приемника в источник, а сумму операндов - в приемник.

В качестве источника для *XADD* может использоваться:

- регистр.

В качестве приемника для *XADD* могут использоваться:

- регистр;
- переменная.

Пример

```
mov eax, 5  
mov ebx, 6  
xadd eax, ebx
```

В результате в регистре *eax* находится значение 11, а в регистре *ebx* – 5.

Команды вычитания

Синтаксис:

SUB приемник, источник

Вычитает источник из приемника и помещает разность в приёмник.

В качестве приемника для *SUB* могут использоваться:

- регистр;
- переменная.

В качестве источника для *SUB* могут использоваться:

- число (непосредственный операнд),
- регистр;
- переменная.

Важно!!! Точно так же, как и команда *ADD*, *SUB* не делает разницы между числами со знаком и без знака, но флаги позволяют использовать ее и для тех, и для других.

Пример

```
mov eax, 5      ; помещаем число 5 в регистр eax
mov ebx, 6      ; помещаем число 6 в регистр ebx
sub ebx, eax    ; вычитаем значение eax из ebx,
                ; а результат помещаем в ebx, то
                ; есть ebx будет равен 1, а eax
                ; не изменится
```

Синтаксис:

SBX приемник, источник

Эта команда аналогична *SUB*, но она вычитает из приемника значение источника и дополнительно вычитает значение флага *CF*.

Важно!!! Нельзя использовать переменную одновременно и для источника, и для приемника.

Команда инкрементирования

Синтаксис:

INC приемник

Увеличивает приемник на 1.

В качестве приемника для *INC* могут использоваться:

- регистр;
- переменная.

Важно!!! Единственное отличие этой команды от *ADD* приемник, 1, состоит в том, что флаг *CF* не затрагивается. Остальные арифметические флаги (*OF*, *SF*, *ZF*, *AF*, *PF*) устанавливаются в соответствии с результатом сложения.

Команда декрементирования

Синтаксис:

DEC приемник

Уменьшает приемник на 1.

В качестве приемника для *DEC* могут использоваться:

- регистр;
- переменная.

Важно!!! Единственное отличие этой команды от *SUB* приемник, 1, состоит в том, что флаг *CF* не затрагивается. Остальные арифметические флаги (*OF, SF, ZF, AF, PF*) устанавливаются в соответствии с результатом сложения.

Команда изменения знака

Синтаксис:

NEG приемник

Выполняет над числом, содержащимся в приемнике, операцию дополнения до двух. Эта операция эквивалентна обращению знака операнда, если рассматривать его как число со знаком. Если приемник равен нулю, флаг *CF* устанавливается в 0, иначе - в 1. Остальные флаги (*OF, SF, ZF, AF, PF*) назначаются в соответствии с результатом операции.

В качестве приемника для *NEG* могут использоваться:

- регистр;
- переменная.

Пример

```
mov eax, 5; помещаем число 5 в регистр eax  
neg eax ; значение регистра eax изменилось на -5
```

Команда сравнения

Синтаксис:

CMR приемник, источник

Данная команда аналогична команде *SUB*, но в отличии от неё приемник не изменяется, а зато получают флаги. Другими словами, команда сравнивает приемник и источник и устанавливает флаги. Действие осуществляется путем вычитания источника из приемника, причем результат вычитания никуда не записывается. Единственным следствием работы этой команды оказывается изменение флагов *CF, OF, SF, ZF, AF* и *PF*.

Обычно команду CMP используют вместе с командами условного перехода, условной пересылки данных или условной установки байтов, которые позволяют применить результат сравнения, не обращая внимания на детальное значение каждого флага.

Важно!!! Нельзя использовать переменную одновременно и для источника, и для приемника.

Команды распространения знака

Синтаксис:

CWD

CDQ

Назначение CWD – конвертирование слова в двойное слово, CDQ – конвертирование двойного слова в учетверенное. Команда CWD превращает слово в AX в двойное слово, младшая половина которого (биты 0-15) остается в AX, а старшая (биты 16-31) располагается в DX.

Команда CDQ выполняет аналогичное действие по отношению к двойному слову в EAX, расширяя его до учетверенного слова в EDX:EAX.

Эти команды лишь устанавливают все биты регистра DX или EDX в значение, равное величине старшего бита регистра AX или EAX, сохраняя, таким образом, его знак.

Команды умножения

Синтаксис:

MUL источник

IMUL источник

Команда MUL – беззнаковое умножение, а IMUL – знаковое целочисленное умножение. Источник умножается на AL, AX или EAX (в зависимости от размера операнда), и результат располагается в AX, DX:AX или EDX:EAX соответственно.

В качестве источника для MUL (IMUL) могут использоваться:

- регистр;
- переменная.

Пример

```
mov eax, 5 ; помещаем число 5 в регистр eax
mov ebx, 6 ; помещаем число 6 в регистр ebx
mul ebx ; значение регистра eax стало равно
; произведению eax и ebx, т. е. eax=30
```

Синтаксис:

IMUL приемник, источник

Источник умножается на приемник и результат заносится в приемник

В качестве приемника для *IMUL* могут использоваться:

- регистр.

В качестве источника для *IMUL* могут использоваться:

- число (непосредственный операнд),
- регистр;
- переменная.

Синтаксис:

IMUL приемник, источник_1, источник_2

Источник_1 умножается на источник_2 и результат заносится в приемник.

В качестве приемника для *IMUL* могут использоваться:

- регистр.

В качестве источника_1 для *IMUL* могут использоваться:

- регистр;
- переменная.

В качестве источника_2 для *IMUL* могут использоваться:

- число.

Пример

```
mov eax, 6 ; помещаем число 6 в регистр eax
imul ebx, eax, 2 ; значение регистра eax
; умножили на 2 и результат
; поместили в ebx, то есть в
; ebx - 12, а eax не изменился
```

Команды деления

Синтаксис:

DIV источник

IDIV источник

Назначение команды *DIV* – целочисленное деление без знака, *IDIV* – целочисленное деление со знаком. Выполняет целочисленное деление без знака *AL*, *AX* или *EAX* (в зависимости от размера источника) на источник и помещает результат в *AL*, *AX* или *EAX*, а остаток - в *DH*, *DX* или *EDX* соответственно. Результат всегда округляется в сторону нуля, абсолютное значение остатка меньше абсолютного значения делителя. Флаги *CF*, *OF*, *SF*, *ZF*, *AF* и *PF* после этой команды не определены, а переполнение или деление на ноль вызывает исключение *#DE* (ошибка при делении) в защищенном режиме и прерывание 0 – в реальном.

В качестве источника для *DIV* (*IDIV*) могут использоваться:

- регистр;
- переменная.

Пример

```
mov eax, 13    ; помещаем число 13 в регистр eax
mov ebx, 6     ; помещаем число 6 в регистр ebx
cdq           ; конвертирование двойного слова в
              ; учетверенное
div ebx       ; значение регистра eax разделилось
              ; на значение регистра ebx, целая
              ; часть от деления записалась в
              ; регистр eax, а остаток от деления
              ; в edx (значения eax и edx
              ; перезаписались, а ebx – не
              ; изменился)
```

Компилятор FASM

Для того, чтобы начать программировать, нам понадобится компилятор. *Компилятор* — это программа, которая переводит исходный текст, написанный программистом, в исполняемый процессором машинный код. Мы познакомимся с молодым, стремительно набирающим популярность компилятором *FASM* (*Flat Assembler*).

Flat assembler - это быстрый компилятор ассемблера для процессоров с архитектурой x86, который делает множественные проходы для оптимизации размера сгенерированного машинного кода. Он способен скомпилировать сам себя и существуют версии

для разных операционных систем. Все версии созданы для использования с помощью системной командной строкой и в обращении с ними нет разницы. Этот компилятор достаточно прост в установке и использовании, отличается компактностью и быстротой работы, имеет богатый и емкий макросинтаксис, позволяющий автоматизировать множество рутинных задач, к тому же написан на себе самом. Отметим, что есть версии под DOS, Windows и Linux.

Для работы всех версий требуется 32-битный процессор с архитектурой x86 (как минимум 80386), хотя также он должен обрабатывать программы для 16-битных процессоров с архитектурой x86. DOS-версия требует ОС, совместимую с MS DOS 2.0, Windows – версия требует консоль Win32, совместимую с версией 3.1.

Рассмотрим каркас нашей программы:

```
format PE GUI 4.0
include "d:\include\win32ax.inc"
.data
<инициализируемые данные>
...
.code
<метка>:
    <код программы>
...
.end <метка>
```

Первая строчка указывает формат исполняемых файлов, то есть после директивы *format* указывается тип получаемой программы, будь то программы для ОС DOS (format MZ), для Windows Vista (format PE64 GUI 4.0) или в случае создания собственной библиотеки DLL (format PE CUI 4.0 DLL). Данную строку необязательно включать.

Важно!!! Пустые строки ничего не обозначают и чаще всего используются для более легкого прочтения кода.

Вторая строка начинается с директивы *include*, которая включает другой ассемблерный файл в текст. Таким образом, мы подключили к программе текст из *win32ax.inc*, *macro/if.inc*, множество макроинструкций и общий набор библиотек функций

Windows. При помощи подключаемых файлов мы организуем некое подобие языка высокого уровня, чтобы избежать множественного описания каждой функции вручную, мы подключаем целые библиотеки описания стандартных функций Windows. В кавычках необходимо указать точный путь к файлу *win32ax.inc* и его обязательно нужно включить в программу, которая предназначена для Windows. Также при помощи этой директивы можно разбить проект на несколько частей.

Директивы *.DATA* и *.CODE* – это то, что называется секциями. Вы можете поделить адресное пространство на логические секции. Начало одной секции отмечает конец предыдущей. Есть две группы секций: данных и кода.

.DATA — Эта секция содержит инициализированные данные вашей программы.

.CODE - секция для кода; там содержится весь код.

Важно!!! *<метка>* — любая произвольная метка, устанавливающая границы кода. Обе метки должны быть идентичны. Весь код должен располагаться между ними.

Ресурсы Windows

Windows предоставляет огромное количество ресурсов Windows-программам через Windows API (Application Programming Interface).

Windows API — это большая коллекция очень полезных функций, располагающихся непосредственно в операционной системе и готовых для использования программами. Эти функции находятся в нескольких динамически подгружаемых библиотеках (DLLs), таких как *kernel32.dll*, *user32.dll* и *gdi32.dll*.

Windows программы динамически подсоединяется к этим библиотекам, то есть код API-функций не включается в исполняемый файл. Информация находится в библиотеках импорта. Вы должны слинковать ваши программы с правильными библиотеками импорта, иначе они не смогут найти эти функции. Когда Windows программа загружается в память, Windows читает информацию, сохраненную в программе. Эта информация включает имена функций, которые программа использует и DLL, в которых эти функции располагаются. Когда

Windows находит подобную информацию в программе, она вызывает библиотеки и исправляет в программе вызовы этих функций, так что контроль всегда будет передаваться по правильному адресу.

Существует две категории API функций: одни работают с ANSI-строками, а другие с Unicode-строками. ANSI строки – массивы символов, оканчивающиеся NULL-ом достаточны для европейских языков и не поддерживает некоторые восточные языки. Размер ANSI-символа — 1 байт. Размер символа UNICODE — 2 байта, и поэтому может поддерживать 65536 различных символов. Чаще будем использовать include-файл, который может определить и выбрать подходящую для вашей платформы функцию. Просто обращайтесь к именам API-функций без постфикса.

Для того, чтобы создать окно с сообщением нужно воспользоваться функцией MessageBox. Прототип функции:

```
MessageBox PROTO hwnd: DWORD, lpText: DWORD,  
lpCaption: DWORD, uType: DWORD
```

- `hwnd` — это handle родительского окна, дескриптор. Это может быть числом, представляющим окно, к которому вы обращаетесь. Его значение для вас не важно. Вы только должны знать, что оно представляет окно. Когда вы захотите сделать что-нибудь с окном, вы должны обратиться к нему, используя его хэндл. NULL или 0 в примере ниже означает, что у окна нет владельца, оно само по себе и не зависит ни от каких других окон.

- `lpText` — это указатель на текст, который вы хотите отобразить в клиентской части окна сообщения. Указатель — это адрес чего-либо. Указатель на текстовую строку равен адресу этой строки.

- `lpCaption` — это указатель на заголовок окна сообщения.

- `uType` — устанавливает иконку, число и вид кнопок окна.

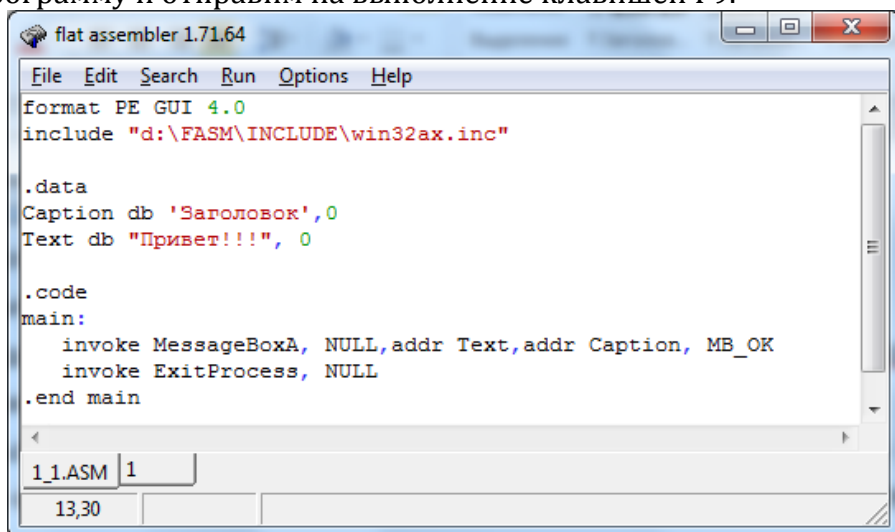
Когда программа выходит в Windows, ей следует вызвать API-функцию `ExitProcess`.

Большинство прототипов для API-функций содержатся в include-файлах. Они находятся в директории `FASM\INCLUDE`. Файлы подключения имеют расширение `.inc` и прототипы

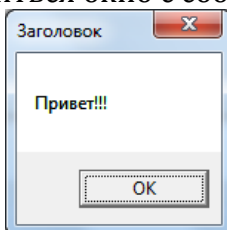
функций DLL находятся в .inc файле с таким же именем, как и у этой DLL. Например, ExitProcess экспортируется из kernel32.lib, так что прототип ExitProcess находится в kernel32.inc.

В строке `invoke ExitProcess, 0` параметр 0 - это значение, которое программа вернет Windows после окончания программы. Поместив эту строку непосредственно после стартовой метки, вы получите Win32-программу, немедленно выходящую в Windows, но тем не менее полнофункциональную.

Запустим Fasm и наберем текст программы. Скомпилируем программу и отправим на выполнение клавишей F9.



В результате появится окно с сообщением "Привет".



В секции данных определены две оканчивающиеся NULL строки.

Важно!!! Каждая ANSI строка в Windows должна оканчиваться NULL'ом (0 в шестнадцатеричной системе).

Далее используются две константы, NULL и MB_OK. Эти константы прописаны в win32ax.inc, так что вы можете обратиться к ним, указав их имя, а не значение. Это улучшает читабельность кода.

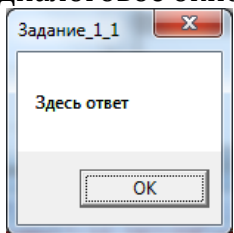
Оператор addr используется для передачи адреса метки (и не только) функции. Он действителен только в контексте директивы invoke. В данном примере вы можете использовать offset вместо addr. Тем не менее, есть некоторые различия между ними. Addr не может быть использован с метками, которые определены впереди, а offset может. Например, если метка определена где-то дальше в коде, чем строка с invoke, addr не будет работать.

Выполнение начинается с первой инструкции, следующей за меткой, установленной после конца директив. В вышеприведенном каркасе выполнение начинается непосредственно после метки 'start'.

ПРАКТИЧЕСКАЯ ЧАСТЬ

Пример №1

Вывести на экран диалоговое окно следующего вида:



Решение

```
format PE GUI 4.0
include 'd:\FASM\INCLUDE\win32ax.inc'
.data
Caption db "Задание_1_1", 0
Text db "Здесь ответ", 0
.code
start:
    invoke MessageBox, 0, Text, Caption, 0
    invoke ExitProcess, 0
```

.end start

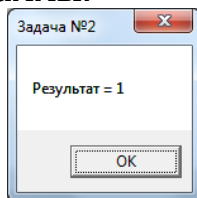
Пример №2

Найти значение выражения $y = \frac{ab}{a+b}$, где $a=2$, $b=3$.

Решение:

```
format PE GUI 4.0
include 'd:\FASM\INCLUDE\win32ax.inc'
.data
Caption db "Задача №2", 0
Text db "Результат = %d", 0
buf rd 255
a dd 2
b dd 3
.code
start:
push eax                ; помещаем данные,
                        ; находящиеся в регистрах
push ebx                ; eax, ebx в стек
    mov ebx, [a]         ; пересылает значение
                        ; переменной a в регистр ebx
    add ebx, [b]         ; производится сложение
                        ; регистра ebx и переменной b
    mov eax, [a]         ; пересылает значение
                        ; переменной a в регистр eax
    imul eax, [b]        ; производится умножение
    cdq                  ; конвертирование двойного
                        ; слова в учетверённое
    idiv ebx             ; производится целочисленное
                        ; деление значения регистра
                        ; eax на источник ebx
invoke wsprintf, addr buf, addr Text, eax
invoke MessageBox, 0, addr buf, addr Caption, MB_OK
pop ebx
pop eax
invoke ExitProcess, 0
.end start
```

Результат работы программы:



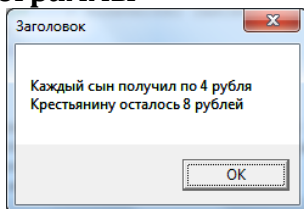
Пример №3

Составить программу для решения задачи: «У крестьянина N сыновей и K рублей. Он разделил все деньги поровну между сыновьями. Сколько рублей получил каждый сын, и сколько осталось у крестьянина?».

Решение

```
format PE GUI 4.0
include 'd:\FASM\INCLUDE\win32ax.inc'
.data
Caption db "Заголовок", 0
Text db 'Каждый сын получит по %d рубля',
      13, 10, 'Крестьянину осталось %d рублей', 0
buf rd 255
K dd 100      ; количество денег
N dd 23       ; количество сыновей
.code
start:
mov eax, [K]
cdq
div [N]
invoke wsprintf, addr buf, addr Text, eax
invoke MessageBox, 0, addr buf, addr Caption, MB_OK
invoke ExitProcess, 0
.end start
```

Результат работы программы



Пример №4

Составить программу для нахождения значения выражения в целых числах $y = \frac{a+b}{a \cdot b} - 2 \cdot a \cdot b$.

Решение

Решение задачи начнем с анализа выражения значение которого необходимо найти. Разобьём выражение на два: первое $\frac{a+b}{a \cdot b}$ и второе $2 \cdot a \cdot b$. В результате найдем разность.

Рассматривая первое выражение $\frac{a+b}{a \cdot b}$, делаем вывод, что нам необходимо будет находить частное от деления, а следовательно, использовать команду `div`. Отметим, что данная команда использует два регистра, причем делимое записывается в регистр `eax`, а делитель в регистр или переменную. Поэтому фрагмент кода программы имеет вид:

```
mov eax, [a]    ; формируем
add eax, [b]    ; числитель
mov ebx, [b]    ; формируем
imul ebx, [a]   ; знаменатель
cdq             ; конвертируем двойное слово в учетверенное
div ebx         ; производим деление
```

После произведенного деления содержимое регистров следующее: `eax` – целая часть от деления, `ebx` – значение делителя, `edx` – остаток от деления.

Так как значение регистра `ebx` не изменилось, следовательно, чтобы получить $2 \cdot a \cdot b$, необходимо значение регистра `ebx` умножить на 2:

```
imul ebx, 2
```

Осталось найти разность. В результате имеем:

```
format PE GUI 4.0
include 'd:\FASM\INCLUDE\win32ax.inc'
.data
Caption db 'Заголовок',0
Text db 'Результат %d',0
buf rd 255
a dd 5
b dd 1
```

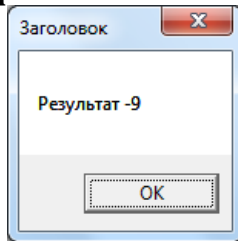


```

.code
start:
    mov eax, [a]
    add eax, [b]
    mov ebx, [b]
    imul ebx, [a]
    cdq
    div ebx
    imul ebx, 2
    sub eax, ebx
    invoke wsprintf, addr buf, addr Text, eax
    invoke MessageBox, 0, addr buf, addr Caption, MB_OK
    invoke ExitProcess, 0
.end start

```

Результат работы программы:



Пример №5

Составить программу для решения задачи: «Дано трехзначное натуральное число. Получить новое натуральное число Y, которое являлось бы перевёртышем числа X».

Решение

```

format PE GUI 4.0
include 'd:\FASM\INCLUDE\win32ax.inc'
.data
Caption db 'Заголовок', 0
Text db 'Старое число x=%d', 13, 10, 'Новое число y=%d', 0
buf db 100 dup(?)
x dd 123 ; трехзначное число
.code
start:

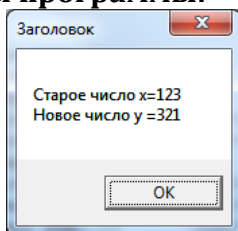
```

```

mov eax, [x]
mov ecx, 10
cdq
div ecx           ;eax=12, edx=3
mov ebx, edx      ;eax=12, edx=ebx=3
cdq
div ecx           ;eax=1, edx=2 , ebx=3
imul ebx, 100     ;ebx=300
imul edx, 10      ;edx=20
add eax, ebx      ;eax=301
add eax, edx      ;eax=321
invoke sprintf, addr buf, addr Text, [x], eax
invoke MessageBox, 0, addr buf, addr Caption, MB_OK
invoke ExitProcess, 0
end start

```

Результат работы программы:



ЗАДАНИЯ ДЛЯ САМОСТОЯТЕЛЬНОЙ РАБОТЫ

1. Ознакомьтесь с теоретическим материалом.
2. Разберите все примеры из практической части лабораторной работы, т.е. наберите и просмотрите их работу.
3. Выведите фразу «Лучший предмет на первом курсе это - «Математический анализ».
4. Выведите значение регистра eax, предварительно поместив в него значение 4.
5. Выведите значение переменной a=5.
6. Выполните индивидуальные задания.

ИНДИВИДУАЛЬНЫЕ ЗАДАНИЯ

Вариант 1

№1. Даны стороны прямоугольника. Найти его периметр.

№2. Найдите значение выражения

$$\bullet \quad y = \frac{a^3 + 5}{8a} - ab; \quad \bullet \quad y = \frac{4x^2 + a^2}{3(a-b)} + bx^3 - ax^2.$$

№3. Дано трехзначное натуральное число В. Найти сумму его цифр.

Вариант 2

№1. Дано расстояние в сантиметрах. Найти число полных метров в нем.

№2. Найдите значение выражения

$$\bullet \quad y = \frac{b^2}{5+a} - \frac{(b+a)^2}{3b}; \quad \bullet \quad y = a^2b^2c^2 + \frac{x^2 + c^2}{3+bc} - 2x.$$

№3. Дано трехзначное натуральное число В. Найти произведение его цифр.

Вариант 3

№1. Дана масса в килограммах. Найти число полных центнеров в ней.

№2. Найдите значение выражения

$$\bullet \quad y = \frac{(b-a)^2}{4+b} + ab; \quad \bullet \quad y = \frac{x^2 + ax}{3b-c} + 2xa^3 - \frac{x^3}{2}.$$

№3. Дано трехзначное число. В нем зачеркнули последнюю справа цифру и приписали ее в начале. Найти полученное число.

Вариант 4

№1. Дана масса в килограммах. Найти число полных тонн в ней.

№2. Найдите значение выражения

$$\bullet \quad y = \frac{a}{a+b} - \frac{1}{2}(a^3 + 3); \quad \bullet \quad y = \frac{3a^2 - b}{cx} - 5ax + \frac{4ab}{x^2 + 1}.$$

№3. Дано трехзначное натуральное число А. Получить новое натуральное число В, в котором цифра числа А, обозначающая десятки (меньше 5), была бы удвоена.

Вариант 5

№1. Дано расстояние в метрах. Найти число полных километров в нем.

№2. Найдите значение выражения

• $y = \frac{a^3 + 5}{8a} - ab$;

• $y = \frac{x^2 - ax}{b + 2} - \frac{x^3}{3} + a^2x$.

№3. Дано трехзначное натуральное число А. Получить новое натуральное число В, в котором поменялись бы местами 1 и 3 цифры числа А.

Вариант 6

№1. С некоторого момента прошло 234 дня. Сколько полных недель прошло за этот период?

№2. Найдите значение выражения

• $y = (a + b)a + \frac{ab}{5}$;

• $y = \frac{a^2x^2}{b - 2} - 3a^3x^3 + \frac{x + a^3}{5}$.

№3. Дано трехзначное число. Найти число, полученное при перестановке второй и третьей цифр заданного числа.

Вариант 7

№1. Дан прямоугольник с размерами 543×130 мм. Сколько квадратов со стороной 130 мм можно отрезать от него?

№2. Найдите значение выражения

• $y = \frac{3a^2 - b}{a + b} - 3(a^2 - 1)$;

• $y = \frac{x^3 + a^3}{8b + 1} + \frac{ax^3}{3} - ax$.

№3. Дано трехзначное натуральное число В. Получить новое натуральное число А, в котором поменялись бы местами 2 и 3 цифры числа В.

Вариант 8

№1. С начала суток прошло n секунд. Определить сколько полных часов прошло с начала суток.

№2. Найдите значение выражения

• $y = \frac{ab}{2} + 2ab^2$;

• $y = 2(x^2 + 3b) - \frac{x + a^2}{3b} - \frac{x^2}{2}$.

№3. Дано трехзначное натуральное число В. Найти удвоенное произведение его цифр.

Вариант 9

№1. С начала суток прошло n секунд. Определить сколько полных минут прошло с начала очередного часа.

№2. Найдите значение выражения

$$\bullet \quad y = \frac{4a^2 - b}{2ab} - ab^2; \quad \bullet \quad y = a^3 b^3 c^4 + \frac{x + c}{5 + bc} + 2x^2$$

№3. Дано трехзначное натуральное число B . Найти сумму произведений его цифр. (Например, из числа 123 получим $1 \cdot 2 + 1 \cdot 3 + 2 \cdot 3$).

Вариант 10

№1. С начала суток прошло n секунд. Определить сколько полных секунд прошло с начала очередной минуты.

№2. Найдите значение выражения

$$\bullet \quad y = 4ab^2 - \frac{a + 4}{2b + 1}; \quad \bullet \quad y = \frac{x + 2x^2}{3a} + \frac{x^3}{2 - b} - ax^2$$

№3. Дано трехзначное натуральное число A . Получить новое натуральное число B , в котором поменялись бы местами 1 и 3 цифры числа A .

ЛАБОРАТОРНАЯ РАБОТА №2

КОМАНДЫ ПЕРЕДАЧИ УПРАВЛЕНИЯ. ВЕТВЛЕНИЯ

ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

Чаще всего программы не являются линейными и в программах есть точки, в которых нужно принять решение о том, какая команда будет выполняться следующей. Это решение может быть:

- безусловным — в данной точке необходимо передать управление не следующей команде, а другой, которая находится на некотором удалении от текущей;
- условным — решение о том, какая команда будет выполняться следующей, принимается на основе анализа некоторых условий или данных

Программа представляет собой последовательность команд и данных, занимающих определенное пространство оперативной памяти. Это пространство памяти может либо быть непрерывным, либо состоять из нескольких фрагментов. Какая команда программы должна выполняться следующей, процессор узнает по содержимому пары регистров CS:(E)IP, в которой:

- CS — регистр сегмента кода, в котором находится физический (базовый) адрес текущего сегмента кода;
- EIP/IP — регистр указателя команды, в котором находится значение, представляющее собой смещение в памяти следующей выполняемой команды относительно начала текущего сегмента кода.

Команды передачи управления изменяют содержимое регистров CS и EIP, в результате чего процессор выбирает для выполнения не следующую по порядку команду программы, а команду в некотором другом участке программы. Конвейер внутри процессора при этом сбрасывается.

По принципу действия команды процессора, обеспечивающие организацию переходов в программе, можно разделить на три группы.

1. Команды безусловной передачи управления:
 - безусловного перехода;
 - вызова процедуры и возврата из процедуры;

- вызова программных прерываний и возврата из программных прерываний.

2. Команды условной передачи управления:

- перехода по результату команды сравнения;
- перехода по состоянию определенного флага;
- перехода по содержимому регистра ECX/CX.

3. Команды управления циклом:

- организации цикла со счетчиком ECX/CX;
- организации цикла со счетчиком ECX/CX с возможностью досрочного выхода из цикла по дополнительному условию.

Место, куда необходимо передать управление в языке ассемблера обозначается с помощью меток. **Метка** — это символическое имя, обозначающее определенную ячейку памяти и предназначенное для использования в качестве операнда в командах передачи управления. Метку можно определить двумя способами: оператором «:» (можно определить метку только ближнего типа — near (команды Jcc и JMP); директивой LABEL (тип метки может иметь значение near или far).

Команда безусловного перехода

Синтаксис:

JMP адрес перехода

Назначение данной команды – безусловный переход без сохранения информации о точке возврата. Здесь *адрес перехода* представляет метку или адрес области памяти, в которой находится указатель перехода.

В зависимости от типа перехода различают:

- переход типа short (короткий переход) - если адрес перехода находится в пределах -128...+ 127 байт от команды JMP;
- переход типа near (ближний переход) - если адрес перехода находится в том же сегменте памяти, что и команда JMP;
- переход типа far (дальний переход) - если адрес перехода находится в другом сегменте. Дальний переход может выполняться и в тот же самый сегмент при условии, что в сегментной части операнда указано число, совпадающее с текущим значением CS;

- переход с переключением задачи - передача управления другой задаче в многозадачной среде.

Пример

```
mov eax, 100
mov ecx, 10
jmp l1
invoke wsprintf, addr buf, addr Text, eax
invoke MessageBox, 0, addr buf, addr Caption, MB_OK
l1:
invoke ExitProcess, 0
```

В результате строки вывода выполняться не будут. Программа в регистр `eax` поместит значение 100, в регистр `ecx` – 10, а затем инструкция `jmp` заставит программу перейти на строку `invoke ExitProcess, 0`.

Команды условного перехода

Другой способ изменения последовательности выполнения команд заключается в использовании команды условного перехода. Имена этих команд различаются в зависимости от условия перехода. Условие состоит из значений одного или нескольких флагов в регистре признаков. Работают эти команды одинаково: если условие истинно, выполняется переход на указанную метку, если нет, то процессор продолжит выполнять программу со следующей команды. Общий формат команд условного перехода следующий:

Синтаксис:

Jcc метка_перехода

Это набор команд, выполняющих переход (типа `short` или `near`), если удовлетворяется соответствующее условие, которым в каждом случае реально является состояние тех или иных флагов. Но, когда команда из набора `Jcc` используется сразу после `CMR`, условия приобретают формулировки, соответствующие отношениям между операндами `CMR` (см. табл. 1). Например, если операнды `CMR` были равны, то команда `JE`, выполненная сразу после `CMR`, осуществит переход.

Операнд для всех команд из набора `Jcc` - 8-битное или 32-битное смещение относительно текущей команды.

Таблица 2
Варианты команды Jcc

Код команды	Реальное условие	Условие для СМР
JA JBE	CF=0 и ZF=0	Если выше Если не ниже и не равно
JAЕ JNB	CF=0	Если выше или равно Если не ниже
JB LNAЕ	CF=1	Если ниже Если не выше и не равно
JBE JNA	CF=1 или ZF=1	Если ниже или равно Если не выше
JE JZ	ZF=1	Если равно Если ноль
JG JNLE	ZF=0 и SF=OF	Если больше Если не меньше и не равно
JGE JNL	SF=OF	Если больше или равно Если не меньше
JL JNGE	SF=OF	Если меньше Если не больше и не равно
JLE JNG	ZF=1 или SF=OF	Если меньше или равно Если не больше
JNE JNZ	ZF=0	Если не равно Если не ноль

Важно!!! Слова «выше» и «ниже» в таблице относятся к сравнению чисел без знака; слова «больше» и «меньше» учитывают знак.

Пример

```
mov eax, 5 ; в регистр eax помещается число 5
mov ebx, -5 ; в регистр ebx помещается число -5
cmp eax, ebx; команда cmp меняет флаги (ZF = 0)
jne 11 ; в случае если eax - ebx ≠ 0, то
; происходит прыжок на метку
add eax, ebx
jmp 12
11:
```

```
imul eax, ebx
l2: nop
```

В результате значение регистра `eax` будет равно -25.

Важно!!! Таким образом, совокупность двух команд `str` и `Jcc` дают нам возможность организовать ветвление в языке ассемблер.

Команды условного перехода не изменяют регистры флагов, а значит, могут использоваться несколько штук одновременно после команды `str`.

Пример

```
mov eax, [x]
cmp eax, 0
jl l1
je l2
jg l3
l1:  imul eax, eax
     jmp q1
l2:  add eax, 1
     jmp q1
l3:  sub eax, 10
q1:  invoke wsprintf, addr buf, addr Text, eax
     invoke MessageBox, 0, addr buf, addr Caption, MB_OK
     invoke ExitProcess, 0
```

В результате выполнения данного фрагмента программы мы имеем, что в зависимости от значения переменной `x` будет произведен прыжок или на метку `l1` или на метку `l2` или на метку `l3`.

Допустим, значение `x` будет равно 10, тогда произведётся прыжок на метку `l3` и выполнится команда `sub`, вследствие чего значение регистра `eax` уменьшится на 10 и произойдет вывод значения этого регистра.

Допустим, значение `x` будет равно -10, тогда произведётся прыжок на метку `l1` и выполнится команда `imul`, вследствие чего значение регистра `eax` увеличится на `eax` и произойдет принудительный прыжок на метку `q1` командой `jmp` и значения этого регистра `eax` выведется на экран.

Допустим, значение `x` будет равно 0, тогда произведётся прыжок на метку `l2` и выполнится команда `add`, вследствие чего значение регистра `eax` увеличится вдвое и произойдет принудительный прыжок на метку `q1` командой `jmp` и значения этого регистра `eax` выведется на экран.

Если мы не предусмотрим обход командой `jmp`, то выполнятся все строки кода последовательно и если `x = -10`, то в регистре `eax` будет значение 91.

ПРАКТИЧЕСКАЯ ЧАСТЬ

Пример №1

Даны три целых числа. Вывести на экран те из них, которые кратны пяти.

Решение

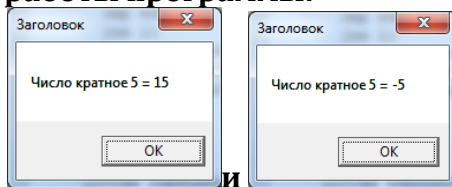
```
format PE GUI 4.0
include 'd:\FASM\INCLUDE\win32ax.inc'
.data
Caption db 'Заголовок', 0
Text db 'Число кратное 5 = %d', 0
buf db 100 dup(?)
x1 dd 123
x2 dd 15
x3 dd -5
.code
start:
    mov ebx, 5
    mov eax, [x1]
    cdq
    idiv ebx
    cmp edx, 0
    jne l1
invoke wsprintf, addr buf, addr Text, [x1]
invoke MessageBox, 0, addr buf, addr Caption, MB_OK
l1:  mov eax, [x2]
    cdq
    idiv ebx
    cmp edx, 0
```

```

    jne l2
invoke sprintf, addr buf, addr Text, [x2]
invoke MessageBox, 0, addr buf, addr Caption, MB_OK
l2:  mov eax, [x3]
     cdq
     idiv ebx
     cmp edx, 0
     jne l3
invoke sprintf, addr buf, addr Text, [x3]
invoke MessageBox, 0, addr buf, addr Caption, MB_OK
l3:
invoke ExitProcess, 0
.end start

```

Результат работы программы:



Пример №2

Рассчитать значение y при заданном значении x :

$$y = \begin{cases} x^2 + 1, & \text{если } x < 0 \\ 0, & \text{если } x = 0 \\ x^3, & \text{если } x > 0 \end{cases}$$

Решение

В данной задаче реализовано последовательное сравнение.

```

format PE GUI 4.0
include 'd:\FASM\INCLUDE\win32ax.inc'
.data
Caption db 'Заголовок', 0
Text db 'y = %d', 0
buf db 100 dup(?)
x dd 0
.code
start:
    mov eax, [x]
    cmp eax, 0

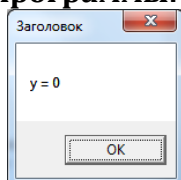
```

```

    jge l1
    imul eax, eax
    add eax, 1
    jmp l3
l1:  cmp eax, 0
    je l2
    imul eax, eax
    imul eax, [x]
    jmp l3
l2:  mov eax, 0
l3:
invoke wsprintf, addr buf, addr Text, eax
invoke MessageBox, 0, addr buf, addr Caption, MB_OK
invoke ExitProcess, 0
.end start

```

Результат работы программы:



Просмотрите работу программы для различных значений x .

ЗАДАНИЯ ДЛЯ САМОСТОЯТЕЛЬНОЙ РАБОТЫ

1. Ознакомьтесь с теоретическим материалом.
2. Разберите все примеры из практической части лабораторной работы, т.е. наберите и просмотрите их работу.
3. Выполните индивидуальные задания.

ИНДИВИДУАЛЬНЫЕ ЗАДАНИЯ

Вариант 1

№1. Составить программу, которая уменьшает первое введенное число в два раза, если оно больше второго введенного числа.

№2. Даны три различных целых числа. Определить, какое из них (первое, второе или третье) самое большое.

№3. Вывести на экран номер четверти координатной плоскости, которой принадлежит точка с координатами (x, y) , если они не нулевые.

№4. Даны три натуральных числа. Определить какие из них оканчиваются цифрой 1?

№5. Если целое число m делится нацело на целое число n , то вывести на экран частное от деления, в противном случае вывести сообщение " m на n нацело не делится"

Вариант 2

№1. Составить программу, которая увеличивает первое введенное число в два раза, если оно больше второго введенного числа.

№2. Дано трехзначное число. Определить является ли сумма его цифр двузначным числом?

№3. Определить пройдет ли сундук со сторонами a , b и c сквозь круглое отверстие с радиусом q ?

№4. Даны три натуральных числа. Определить какие из них оканчиваются цифрой 2?

№5. Дано двузначное число. Определить какая из его цифр больше: первая или вторая?

Вариант 3

№1. Составить программу, которая увеличивает первое введенное число в три раза, если оно меньше второго введенного числа.

№2. Даны три натуральных числа. Определить, имеется ли среди них хотя бы одна пара равных между собой чисел.

№3. Даны натуральные положительные числа a , b , c , d . Выяснить, можно ли прямоугольник со сторонами a , b уместить внутри прямоугольника со сторонами c , d так, чтобы каждая из сторон одного прямоугольника была параллельна или перпендикулярна каждой стороне другого прямоугольника.

№4. Даны три натуральных числа. Определить какие из них оканчиваются цифрой 3?

№5. Дано двузначное число. Определить, равен ли квадрат этого числа учетверенной сумме кубов его цифр. Например, для числа 48 ответ положительный, для числа 52 — отрицательный.

Вариант 4

№1. Даны два числа. Если второе число в квадрате меньше первого числа, то увеличить второе число в пять раз.

№2. Определить является ли треугольник со сторонами a , b , c равносторонним?

№3. Даны три угла. Проверить могут ли они быть углами треугольника. Если да, то проверить, будет ли этот треугольник остроугольным.

№4. Даны три натуральных числа. Определить какие из них оканчиваются цифрой 4?

№5. Дано двузначное число. Определить кратна ли трем сумма его цифр?

Вариант 5

№1. Даны два числа. Если второе число в квадрате больше первого числа, то увеличить второе число в семь раз.

№2. Даны три целых числа. Определить сумму тех чисел, которые кратны трем.

№3. Даны натуральные положительные числа a , b , c , x , y . Выяснить, пройдет ли кирпич с ребрами a , b , c в прямоугольное отверстие со сторонами x , y . Просовывать кирпич в отверстие разрешается только так, чтобы каждое из ребер было параллельно или перпендикулярно каждой из сторон отверстия.

№4. Даны три натуральных числа. Определить какие из них оканчиваются цифрой 5?

№5. Дано двузначное число. Определить кратна ли пяти сумма его цифр?

Вариант 6

№1. Даны два числа. Если второе число в квадрате больше первого числа, то увеличить второе число в пять раз.

№2. Дано трехзначное число. Определить является ли сумма его цифр однозначным числом?

№3. Даны две тройки чисел. В каждой тройке все числа различные. Найти сумму средних чисел каждой тройки.

№4. Даны три натуральных числа. Определить какие из них оканчиваются цифрой 6?

№5. Дано двузначное число. Определить кратно ли семи произведение его цифр?

Вариант 7

№1. Даны два числа. Если второе число в квадрате меньше первого числа, то увеличить второе число в шесть раз.

№2. Даны три целых числа. Определить, сколько из них четных.

№3. В чемпионате по футболу команде за выигрыш дается 3 очка, за проигрыш – 0, за ничью – 1. Известно количество очков, полученных командой за игру. Определить словесный результат игры.

№4. Даны три натуральных числа. Определить какие из них оканчиваются цифрой 7?

№5. Дано трехзначное число. Определить, равен ли квадрат этого числа сумме кубов его цифр.

Вариант 8

№1. Составить программу, которая уменьшает первое введенное число в десять раз, если его квадрат больше второго введенного числа.

№2. Определить максимальное и минимальное значение из трех различных натуральных чисел.

№3. Пройдет ли кирпич со сторонами a , b и c сквозь прямоугольное отверстие со сторонами p и q ? Стороны отверстия должны быть параллельны граням кирпича.

№4. Даны три натуральных числа. Определить какие из них оканчиваются цифрой 8?

№5. Определить, является ли число a делителем числа b ? А наоборот? (Получить два ответа.)

Вариант 9

№1. Составить программу, которая уменьшает первое введенное число в десять раз, если его куб больше второго введенного числа.

№2. Даны три целых числа. Найти сумму тех чисел, которые больше пяти.

№3. Даны три угла. Проверить могут ли они быть углами треугольника. Если да, то проверить, будет ли этот треугольник прямоугольным.

№4. Даны три натуральных числа. Определить какие из них оканчиваются цифрой 9?

№5. Дано натуральное число. Верно ли, что оно заканчивается четной цифрой?

Вариант 10

№1. Составить программу, которая уменьшает первое введенное число в три раза, если его куб больше второго введенного числа.

№2. Даны три целых числа. Вывести на экран те из них, которые являются нечетными.

№3. Определить является ли треугольник со сторонами a , b , c равнобедренным?

№4. Даны три натуральных числа. Определить какие из них оканчиваются цифрой 0?

№5. Дано натуральное число. Верно ли, что оно заканчивается нечетной цифрой?

ЛАБОРАТОРНАЯ РАБОТА №3

КОМАНДЫ ПЕРЕДАЧИ УПРАВЛЕНИЯ. ЦИКЛЫ.

ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

Цикл представляет собой важную алгоритмическую структуру. Циклом называется многократное повторение последовательности команд до наступления указанного условия.

Организовать циклическое выполнение некоторого фрагмента программы можно, используя команды условной передачи управления, или команду безусловного перехода JMP.

Пример

```
mov eax, 1          ; помещаем в регистр eax
                    ; значение начала цикла
mov ebx, 0          ; обнуляем сумму
q1:
  add ebx, eax       ; суммируем
  inc eax            ; увеличиваем счетчик на единицу
  cmp eax, 10        ; сравниваем результат с
                    ; последним значением суммы
  jle q1             ; если меньше или равно, то
                    ; производим прыжок
```

Напомним, что регистр ECX/CX имеет определенное функциональное назначение — он выполняет функции счетчика в командах управления циклами и при работе с цепочками символов.

Синтаксис:

JCXZ метка_перехода

Назначение команды JCXZ (Jump if ex is Zero) — переход, если CX ноль.

Синтаксис:

JECXZ метка_перехода

Назначение команды JECXZ (Jump Equal ecx Zero) — переход, если ECX ноль.

Эти команды очень удобно использовать при организации цикла и при работе с цепочками символов. В отличие от других команд условной передачи управления, команды JCXZ/JECXZ

могут адресовать только короткие переходы — на -128 байт или на +127 байт от следующей за ней команды.

Пример

Организуем цикл, используя команду сравнения и соответствующую команду передачи управления.

```
mov ecx, 0
q1:
    ; тело цикла
    sub ecx, 1
    cmp ecx, 0
    jg q1
    invoke wsprintf, addr buf, addr Text, ecx
    invoke MessageBox, 0, addr buf, addr Caption, MB_OK
```

Анализируя данный код программы увидим что тело цикла выполнилось один раз и следовательно регистр ecx после выполнения данных строк кода будет иметь значение равное (-1). Для того чтобы избежать данной ситуации, можно воспользоваться командой передачи управления JECXZ, которая предотвращает выполнение «пустого» цикла (когда счетчик цикла в ECX равен нулю). Решение задачи с учетом вышесказанного приведено ниже.

```
mov ecx, 0
jcxz exit
q1:
    ; тело цикла
    sub ecx, 1
    cmp ecx, 0
    jg q1
exit:
    invoke wsprintf, addr buf, addr Text, ecx
    invoke MessageBox, 0, addr buf, addr Caption, MB_OK
```

Важно!!! Команды передачи управления и команда сравнения позволяет организовать циклы, подобные циклам *while* в языках высокого уровня.

Команды LOOP

Команда LOOP позволяет организовать циклы, подобные циклам for в языках высокого уровня с автоматическим уменьшением счетчика цикла.

Синтаксис:

loop метка_перехода

Команда реализует описанные далее действия.

1. Декремент регистра ECX/CX.

2. Сравнение регистра ECX/CX с нулем:

- если $(ECX/CX) > 0$, то управление передается на метку перехода;
- если $(ECX/CX) = 0$, то управление передается на следующую после LOOP команду.

Пример

Реализуем решение предыдущего примера, используя команду Loop метка_перехода

```
mov ecx, 10
```

```
jcxz exit
```

```
q1:
```

```
;тело цикла
```

```
loop q1
```

```
exit:
```

```
invoke wsprintf, addr buf, addr Text, ecx
```

```
invoke MessageBox, 0, addr buf, addr Caption, MB_OK
```

Отметим, что в данном примере тело цикла выполняется 10 раз, так как значение регистра ecx равно 10.

Команды LOOPE и LOOPZ (Loop still cx \neq 0 or Zero flag = 0 — повторить цикл пока CX \neq 0 или ZF = 0) — абсолютные синонимы, поэтому используйте ту команду, которая вам больше нравится.

Синтаксис:

loope/loopz метка_перехода

Команды реализуют описанные далее действия.

1. Декремент регистра ECX/CX.

2. Сравнение регистра ECX/CX с нулем и анализ состояния флага нуля ZF:

- если $(ECX/CX) > 0$ и ZF = 1, управление передается на метку перехода;

- если $(ECX/CX) = 0$ или $ZF=0$, управление передается на следующую после `LOOPe(z)` команду.

Команды `LOOPNE` и `LOOPNZ` (Loop still $ex \neq 0$ or NonZero flag = 0 — повторить цикл, пока $CX \neq 0$ или $ZF = 1$) также абсолютные синонимы.

Синтаксис:

`loopne/loopnz` метка_перехода

Команды реализуют описанные далее действия.

1. Декремент регистра `ECX/CX`.

2. Сравнение регистра `ECX/CX` с нулем и анализ состояния флага нуля `ZF`:

- если $(ECX/CX) > 0$ и $ZF = 0$, управление передается на метку перехода;

- если $(ECX/CX) = 0$ или $ZF = 1$, управление передается на следующую после `LOOP` команду.

Команды `LOOPE/LOOPZ` и `LOOPNE/LOOPNZ` по принципу своей работы являются взаимнообратными. Они расширяют действие команды `LOOP` тем, что дополнительно анализируют флаг `ZF`. Это дает возможность организовать досрочный выход из цикла, используя этот флаг в качестве индикатора. Типичное применение этих команд связано с операцией поиска определенного значения в последовательности или со сравнением двух чисел. Недостаток команд организации цикла `LOOP`, `LOOPE/LOOPZ` и `LOOPNE/LOOPNZ` заключается в том, что они реализуют только короткие переходы (от -128 до +127 байт). Для работы с длинными циклами придется использовать команды условного перехода и команду `JMP`.

Пример

```
mov eax, 0      ; обнуляем сумму
mov ecx, 10     ; счетчик цикла
q1:
add eax, ecx    ; тело цикла
loop q1
```

В результате значение регистра `eax` равно 55, в регистра `ecx` – 0.

ПРАКТИЧЕСКАЯ ЧАСТЬ

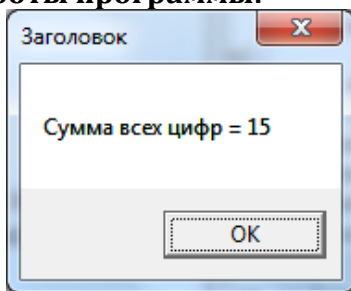
Пример №1

Вычислить сумму $\sum_{k=1}^5 k$.

Решение

```
format PE GUI 4.0
include 'd:\FASM\INCLUDE\win32ax.inc'
.data
Caption db 'Заголовок', 0
Text db 'Сумма всех цифр = %d', 0
buf db 100 dup(?)
.code
start:
    mov eax, 0
    mov ecx, 5
l1:  add eax, ecx
    loop l1
    invoke wsprintf, addr buf, addr Text, eax
    invoke MessageBox, 0, addr buf, addr Caption, MB_OK
    invoke ExitProcess, 0
.end start
```

Результат работы программы:



Пример №2

Определить сколько раз цифра А входит в натуральное число n, если вхождения нет, то выдать словесный ответ.

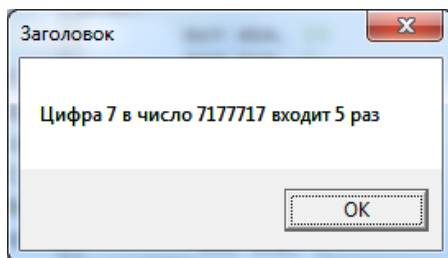
Решение

Решение задачи осложняется тем, что мы не знаем, сколько цифр в числе n. Следовательно, нам необходимо организовать цикл аналогичный циклу while из языков высокого уровня. Для

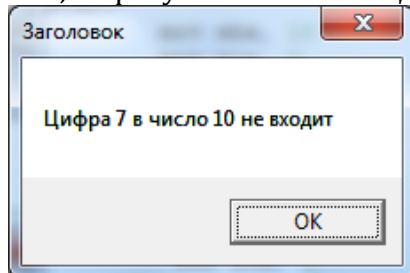
этого нам понадобятся команда сравнения `cmp` и команда передачи управления `jne`. Таким образом, мы в теле цикла будем производить деление на 10 в регистре `edx` получать последнюю цифру числа, до тех пор пока не получим первую цифру.

```
format PE GUI 4.0
include 'd:\FASM\INCLUDE\win32ax.inc'
.data
Caption db 'Заголовок',0
Text1 db 'Цифра %d в число %d входит %d раз',0
Text2 db 'Цифра %d в число %d не входит',0
buf db 100 dup(?)
A dd 7
N dd 7177717
.code
start:
    mov ebx,10
    mov ecx, 0
l1:
    cdq
    idiv ebx
    cmp edx, [A]
    jne l2
    add ecx, 1
l2: nop
    cmp eax, 0
    jne l1
    cmp ecx, 0
    je l3
    invoke wsprintf, addr buf, addr Text,[A],[n], ecx
    invoke MessageBox,0, addr buf, addr Caption, MB_OK
    invoke ExitProcess, 0
l3:
    invoke wsprintf, addr buf, addr Text,[A],[n]
    invoke MessageBox,0, addr buf, addr Caption, MB_OK
    invoke ExitProcess, 0
.end start
```

Результат работы программы:



Если число $n=10$, то результат имеет вид:



Пример №3

Напечатать все целые числа от a до b .

Решение

Реализуем ввод и вывод в данной задаче двумя способами:

1 способ. Воспользуемся стандартным выводом, используя диалоговое окно.

```
format PE GUI 4.0
include 'd:\FASM\INCLUDE\win32ax.inc'
.data
Caption db 'Заголовок', 0
Text db '%d', 0
buf db 100 dup(?)
A dd -3
B dd 3
.code
start:
    mov ebx, [A]
q1:
    invoke wsprintf, addr buf, addr Text, ebx
    invoke MessageBox, 0, addr buf, addr Caption, MB_OK
    add ebx, 1
```

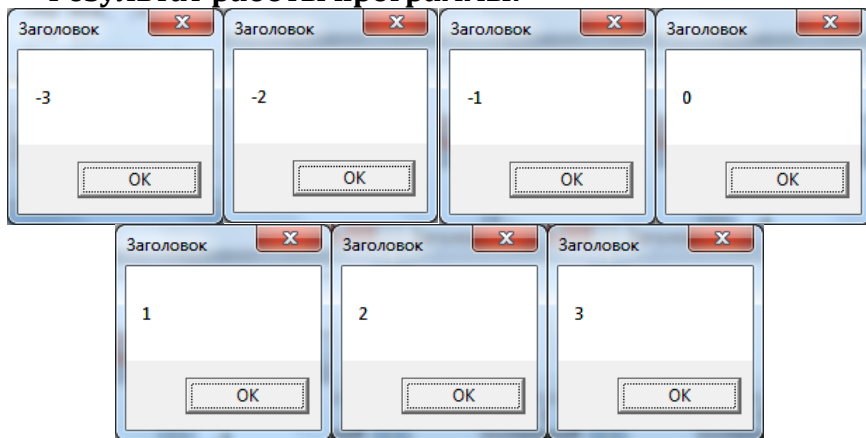


```

cmp ebx, [B]
jle q1
invoke ExitProcess, 0
.end start

```

Результат работы программы:



2 способ. В данной задаче организуем ввод числа с клавиатуры и множественные вывод с помощью функций `printf` и `scanf` соответственно. Отметим, что секция `«.idata»` подключает необходимые библиотеки, в которых находятся соответствующие функции.

В результате откроется консольное приложение, в котором будет отображена одна строка с просьбой ввести значение A. После нажатия клавиши Enter приложение продолжит работу и отобразится новая строка с просьбой ввести значение B. После нажатия клавиши Enter приложение выведет столбец значений и по истечении 5 секунд завершит свою работу, то есть закроет консольное приложение.

```

format PE console 4.0
entry start
include 'd:\FASM\INCLUDE\win32ax.inc'
section '.data' data readable writable
Text1 db 'Input a',0
Text2 db 'Input b',0
tpt db '%d',0
rez db 13,10,'%d',0

```

```

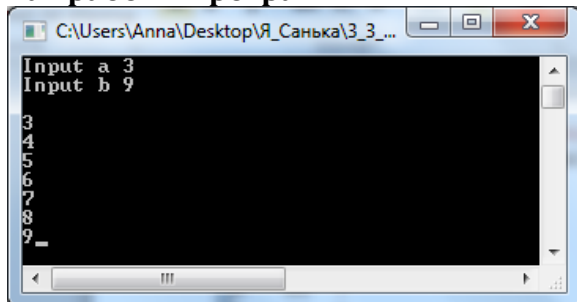
buf rd 255
A dd ?
B dd ?
section '.code' code readable executable
start:
    cinvoke printf, Text1 ;Вывод сообщения с
                           ;просьбой ввести A
    cinvoke scanf, tpt, A ;Ввод A
    cinvoke printf,       ;Вывод сообщения с
                           ;просьбой ввести B
    cinvoke scanf, tpt, B ;Ввод B

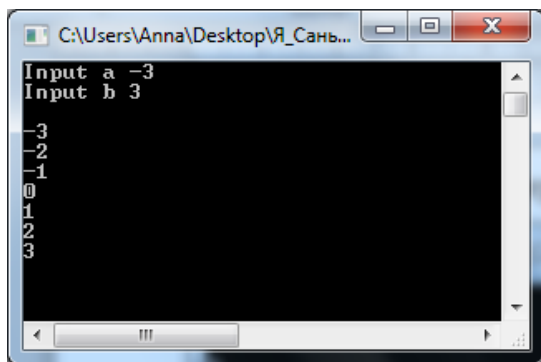
    mov ebx,[A]           ;левый край отрезка копируем
                           ;в регистр ebx
l1:
    cinvoke printf, rez, ebx ;функция вывода
    inc ebx                 ;увеличиваем ebx на 1
    cmp ebx, [B]           ;проверяем ebx и правый
                           ;край отрезка

    jle l1
    invoke sleep, 5000 ;приостанавливаем
;выполнение функции exit на 5 секунд
    invoke exit, 0
section '.idata' import data readable writeable
library user32,'USER32.DLL', msvcrt, 'MSVCRT.DLL',
kernel32,'KERNEL32.DLL'
import kernel32,sleep,'Sleep'
import msvcrt, puts,'puts', scanf,'scanf',
printf,'printf', exit,'exit'

```

Результат работы программы:





Важно!!! Отметим, что функции, используемые для вывода и в первом и во втором способе, используют регистры *eax*, *edx* и *ecx*, следовательно, данные регистры для организации циклов с использованием вывода применять нельзя!!!

ЗАДАНИЯ ДЛЯ САМОСТОЯТЕЛЬНОЙ РАБОТЫ

1. Ознакомьтесь с теоретическим материалом.
2. Разберите все примеры из практической части лабораторной работы, т.е. наберите и просмотрите их работу.
3. Выполните индивидуальные задания.

ИНДИВИДУАЛЬНЫЕ ЗАДАНИЯ

Вариант 1

№1. Вычислить сумму ряда $\sum_{k=5}^{10} k^2$.

№2. Даны натуральные числа x и y . Вычислить произведение $x \cdot y$, используя лишь операцию сложения.

№3. Дано натуральное число. Найти сумму его четных делителей.

№4. Найти все простые числа из диапазона от a до b .

№5. Найти натуральное число из интервала от a до b , у которого количество делителей максимально. Если таких чисел несколько, то должно быть найдено максимальное из них.

Вариант 2

№1. Найти сумму $n^2 + (n - 2)^2 + \dots + 1^2$, где n – нечетное число.

№2. Дано натуральное число. Найти произведение его цифр, меньших семи.

№3. Дано натуральное число. Получить все его делители.

№4. Найти количество делителей каждого из целых чисел от 120 до 140.

№5. Найти натуральное число из интервала от a до b , у которого количество делителей максимально. Если таких чисел несколько, то должно быть найдено минимальное из них.

Вариант 3

№1. Найти сумму $-1^2 + 2^2 - 3^2 + \dots + 10^2$.

№2. Дано натуральное число. Определить, есть ли в нем цифра A . Дать словесный ответ.

№3. Найти сумму целых положительных чисел, больших 30 и меньших 100, кратных трем и оканчивающихся на 2.

№4. Найти все целые числа из промежутка от 200 до 500, у которых ровно шесть делителей.

№5. Два натуральных числа называются дружественными, если каждое из них равно сумме всех делителей другого (само другое число в качестве делителя не рассматривается). Найти все пары натуральных дружественных чисел, меньших 50 000.

Вариант 4

№1. Дано натуральное число $n \geq 2$. Найти сумму $n^2 + (n-1)^2 + \dots + 1^2$.

№2. Дано натуральное число. Найти первую и последнюю цифры числа.

№3. Вывести на экран все целые числа от a до b , кратные некоторому числу c .

№4. Найти 100 первых простых чисел.

№5. Составить программу для нахождения всех натуральных решений $(x$ и $y)$ уравнения $x^2 + y^2 = k^2$, где x , y и k лежат в интервале от 1 до 30. Решения, которые получаются перестановкой x и y , считать совпадающими.

Вариант 5

№1. Дано натуральное число n . Найти сумму $n^2 + (n+2)^2 + \dots + (4n)^2$.

№2. Дано натуральное число. Сколько раз данная цифра А встречается в данном числе.

№3. Найти сумму целых положительных чисел из промежутка от а до b, кратных четырем.

№4. Определить количество трехзначных натуральных чисел, сумма цифр которых равна целому числу n ($0 < n \leq 27$).

№5. Даны натуральные числа m и n . Вычислить $1^n + 2^n + \dots + m^n$.

Вариант 6

№1. Вычислить сумму $1 + 1^2 + 2 + 2^2 + \dots + n + n^2$, где $n \geq 5$.

№2. Дано натуральное число. Найти сумму четных цифр числа.

№3. Найти сумму целых положительных чисел из промежутка от а до b, кратных пяти.

№4. Найти все целые числа из промежутка от 100 до 300, у которых сумма делителей равна 50.

№5. Дано натуральное число n. Вычислить $1^1 + 2^2 + \dots + n^n$.

Вариант 7

№1. Дано натуральное число n. Найти сумму $n^2 + (n + 1)^2 + \dots + (2n)^2$.

№2. Дано натуральное число. Найти его минимальную цифру.

№3. Определить количество натуральных чисел из интервала от 100 до 500, сумма цифр которых равна 15.

№4. Найти все целые числа из промежутка от 1 до 300, у которых ровно пять делителей.

№5. Даны n натуральных чисел. Найти их наибольший общий делитель, используя алгоритм Евклида и учитывая, что $\text{НОД}(a, b, c) = \text{НОД}(\text{НОД}(a, b), c)$.

Вариант 8

№1. Вычислить сумму $1! + 2! + 3! + \dots + n!$. Значение $1 < n \leq 5$.

№2. Дано натуральное число. Найти его максимальную цифру.

№3. Дано натуральное число. Определить количество четных делителей.

№4. Найти все двузначные простые числа.

№5. Найти натуральное число из интервала от a до b , у которого количество делителей максимально. Если таких чисел несколько, то должно быть найдено максимальное из них.

Вариант 9

№1. Найти сумму $2^2 + 2^3 + \dots + 2^{10}$.

№2. Дано натуральное число. Найти сумму нечетных цифр числа.

№3. Дано натуральное число. Определить количество нечетных делителей.

№4. Найти сумму делителей каждого из целых чисел от 50 до 70.

№5. Даны натуральные числа m и n . Получить все натуральные числа, меньшие n , квадрат суммы цифр которых равен m .

Вариант 10

№1. Найти сумму $1^2 + 3^2 + \dots + n^2$.

№2. Дано натуральное число. Найти количество значащих нулей в числе.

№3. Найти 15 первых натуральных чисел, делящихся нацело на 19 и находящихся в интервале, левая граница которого равна 100.

№4. Найти все трехзначные простые числа.

№5. Составить программу нахождения цифрового корня натурального числа. Цифровой корень данного числа получается следующим образом. Если сложить все цифры этого числа, затем все цифры найденной суммы и повторять этот процесс, то в результате будет получено однозначное число (цифра), которая и называется цифровым корнем данного числа.

ЛАБОРАТОРНАЯ РАБОТА №4

ОДНОМЕРНЫЕ МАССИВЫ В АССЕМБЛЕРЕ

ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

Массив – структурированный тип данных, состоящий из некоторого числа элементов одного типа. Массив характеризуется типом элементов, числом элементов и способом их нумерации. Число элементов называется размером, или длиной, массива; способ нумерации описывается одномерным целочисленным массивом, называемым формой массива.

Массив — это последовательность элементов, доступ к которым осуществляется при помощи целочисленного индекса. Индексы всегда следуют по порядку, и поэтому очевидным является использование “циклов” для работы с массивами.

Массивы, доступ к элементам которых осуществляется при помощи одного индекса, называются одномерными массивами или векторами.

Описание и инициализация массива в программе

Специальных средств описания массивов в программах ассемблера, конечно, нет. Чтобы использовать массив в программе, его нужно смоделировать. Можно перечислить элементы массива в поле операндов одной из директив описания данных. При перечислении элементы разделяются запятыми. Например, массив из 5 элементов. Размер каждого элемента 4 байта: `mas dd 1, 2, 3, 4, 5`

Доступ к элементам массива

При работе с массивами необходимо четко представлять себе, что все элементы массива располагаются в памяти компьютера последовательно. Только программист с, помощью составленного им алгоритма обработки определяет, как нужно трактовать последовательность байтов, составляющих массив. Так, одну и ту же область памяти можно трактовать одновременно и как одномерный, и как двухмерный массив. Все зависит только от алгоритма обработки этих данных в конкретной программе.

Ассемблер не подозревает ни об их существовании индексов элементов массива, ни об их численных смысловых значениях. Для того чтобы локализовать определенный элемент массива, к его имени нужно добавить индекс. Так как мы моделируем массив, то должны позаботиться и о моделировании индекса. В языке ассемблера индексы массивов — это обычные адреса, но с ними работают особым образом. Другими словами, когда при программировании на ассемблере мы говорим об индексе, то, скорее, подразумеваем под этим не номер элемента в массиве, а некоторый адрес. Нумерация элементов массива в ассемблере начинается с нуля.

В общем случае для получения адреса элемента в массиве необходимо начальный (базовый) адрес массива сложить с произведением индекса (номер элемента минус единица) этого элемента на размер элемента массива:

база + (индекс • размер элемента) .

Архитектура процессора предоставляет довольно удобные программно-аппаратные средства для работы с массивами. К ним относятся базовые и индексные регистры, позволяющие реализовать несколько режимов адресации данных. Используя данные режимы адресации, можно организовать эффективную работу с массивами в памяти.

Индексная адресация со смещением — режим адресации, при котором эффективный адрес формируется из двух компонентов:

- постоянный (базовый) компонент формируется указанием прямого адреса массива в виде имени идентификатора, обозначающего начало массива;
- переменный (индексный) компонент формируется указанием имени индексного регистра.

Пример

`mov eax, [mas+esi]` ; mas – базовый компонент; esi – индексный компонент

Базовая индексная адресация со смещением — режим адресации, при котором эффективный адрес формируется максимум из трех компонентов:

- в качестве постоянного (необязательного) компонента может выступать прямой адрес массива в виде имени идентификатора, обозначающего начало массива, или непосредственного значения;

- переменный (базовый) компонент формируется указанием имени базового регистра;

- переменный (индексный) компонент формируется указанием имени индексного регистра.

Важно!!! В качестве базового регистра может использоваться любой из восьми регистров общего назначения. В качестве индексного регистра также можно использовать любой регистр общего назначения, за исключением ESP/SP.

Процессор позволяет масштабировать индекс. Это означает, что если указать после имени индексного регистра символ звездочки (*) с последующей цифрой 2, 4 или 8, то содержимое индексного регистра будет умножаться на 2, 4 или 8, то есть масштабироваться. Применение масштабирования облегчает работу с массивами, которые имеют размер элементов, равный 2, 4 или 8 байт, так как процессор сам производит коррекцию индекса для получения адреса очередного элемента массива. Нам нужно лишь загрузить в индексный регистр значение требуемого индекса (считая от 0).

ПРАКТИЧЕСКАЯ ЧАСТЬ

Пример №1

Найти сумму элементов одномерного массива.

Решение

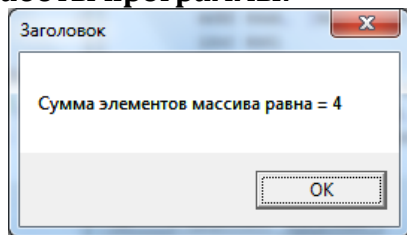
```
format PE GUI 4.0
entry start
include 'd:\FASM\INCLUDE\win32ax.inc'
.data
Caption db 'Заголовок', 0
Text db 'Сумма элементов массива равна = %d', 0
buf db 100 dup(?)
n dd 5 ; количество элементов массива
mass dd -3, 1, 2, 1, 3
```

```

.code
start:
    mov ecx, [n]          ;n раз выполнится цикл
    mov esi, 0            ;обнулили индексный регистр
    mov eax, 0            ;обнулили сумму
l1:
    add eax, [mass+esi*4] ;суммируем
                           ;элементы массива
    inc esi               ;переходим
                           ;к следующему элементу массива
loop l1
invoke wsprintf, addr buf, addr Text, eax
invoke MessageBox, 0, addr buf, addr Caption, MB_OK
invoke ExitProcess, 0
.end start

```

Результат работы программы:



Пример №2

Расположить элементы массива по возрастанию.

Решение

```

format PE GUI 4.0
entry start
include 'd:\FASM\INCLUDE\win32ax.inc'
.data
Caption db 'Заголовок', 0
Text db '%d', 0
buf db 100 dup(?)
n dd 5          ;количество элементов массива
mass dd 3, -1, 2, 0, 5
.code
start:
    mov ecx, 0

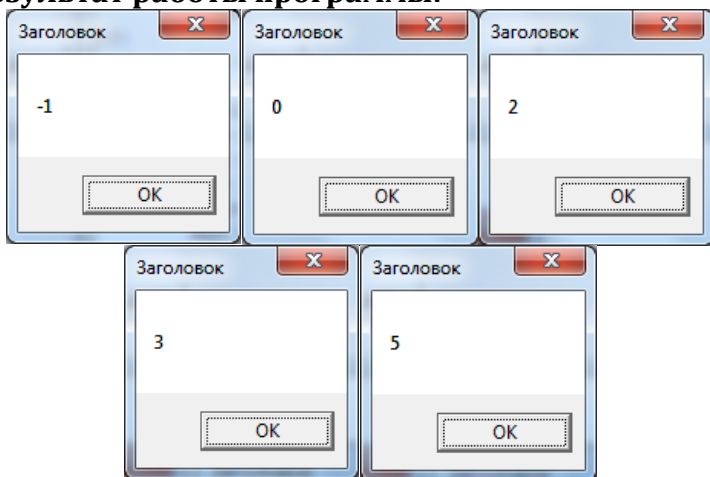
```

```

l1:  mov esi, 0
l2:  mov eax, [mass+esi*4]
     cmp eax, [mass+esi*4+4]
     jl  l3
     mov edx, [mass+esi*4+4]
     mov [mass+esi*4+4], eax
     mov [mass+esi*4], edx
l3:
     inc esi
     cmp esi, 3
     jl  l2
     add ecx, 1
     cmp ecx, [n]
     jne l1
;ВЫВОД ЭЛЕМЕНТОВ МАССИВА
     mov esi, 0
l4:
     invoke sprintf, addr buf, addr Text, [mass+esi*4]
     invoke MessageBox, 0, addr buf, addr Caption, MB_OK
     inc esi
     cmp esi, [n]
     jne l4
     invoke ExitProcess, 0
.end start

```

Результат работы программы:

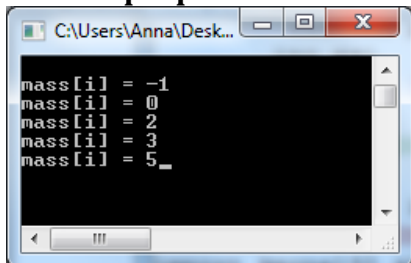


Далее показана реализация вывода используя консольное приложение.

```
format PE console 4.0
entry start
include 'd:\FASM\INCLUDE\win32ax.inc'
section '.data' data readable writable
tpt db '%d',0
rez db 13,10,'mass[i] = %d',0
n dd 5 ;количество элементов массива
mass dd 3, -1, 2, 0, 5
buf rd 255
section '.code' code readable executable
start:
    mov ecx, 0
l1:    mov esi, 0
l2:    mov eax, [mass+esi*4]
        cmp eax, [mass+esi*4+4]
        jl l3
            mov edx, [mass+esi*4+4]
            mov [mass+esi*4+4], eax
            mov [mass+esi*4], edx
l3:
        inc esi
        cmp esi, 3
        jl l2
        add ecx, 1
        cmp ecx, [n]
        jne l1
;Вывод элементов массива
        mov esi, 0
l4:
        cinvoke printf, rez, [mass+esi*4]
        inc esi
        cmp esi, [n]
        jne l4
invoke sleep, 10000
invoke exit, 0
section '.idata' import data readable writeable
```

```
library user32, 'USER32.DLL', msvcrt, 'MSVCRT.DLL',
kernel32, 'KERNEL32.DLL'
import kernel32, sleep, 'Sleep'
import msvcrt, puts, 'puts', scanf, 'scanf',
printf, 'printf', exit, 'exit'
```

Результат работы программы:



Пример №3

Заменить элементы массива с нечетными номерами на квадраты их номеров, а элементы с четными номерами удвоить.

Решение

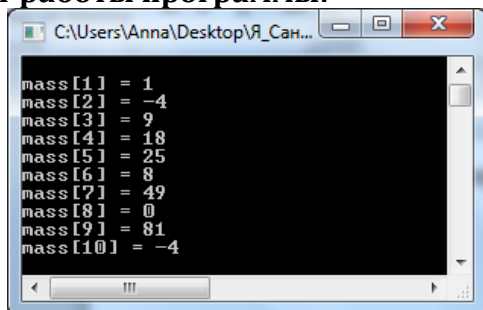
```
format PE console 4.0
entry start
include 'd:\FASM\INCLUDE\win32ax.inc'
section '.data' data readable writable
tpt db '%d', 0
rez db 13, 10, 'mass[%d] = %d', 0
n dd 10 ; количество элементов массива
mass dd 3, -2, 7, 9, -3, 4, -1, 0, 4, -2
buf rd 255
section '.code' code readable executable
start:
    mov esi, 0
l1:
    mov eax, esi
    inc eax
    imul eax, eax
    mov [mass+esi*4], eax
    mov eax, [mass+esi*4+4]
    imul eax, 2
    mov [mass+esi*4+4], eax
```

```

    add esi, 2
    cmp esi, [n]
    jl l1
;Вывод элементов массива
    mov esi, 0
l4    inc esi
    cinvoke printf, rez, esi, [mass+(esi-1)*4]
    cmp esi, [n]
    jne l4
invoke sleep, 10000
invoke exit, 0
section '.idata' import data readable writeable
library user32, 'USER32.DLL', msvcrt, 'MSVCRT.DLL',
kernel32, 'KERNEL32.DLL'
import kernel32, sleep, 'Sleep'
import msvcrt, puts, 'puts', scanf, 'scanf',
printf, 'printf', exit, 'exit'

```

Результат работы программы:



Важно!!! Отметим, что функции, используемые для вывода и в первом и во втором способе, используют регистры *eax*, *edx* и *ecx*, следовательно, данные регистры для организации вывода элементов массива применять нельзя!!!

ЗАДАНИЯ ДЛЯ САМОСТОЯТЕЛЬНОЙ РАБОТЫ

1. Ознакомьтесь с теоретическим материалом.
2. Разберите все примеры из практической части лабораторной работы.
3. Выполните индивидуальные задания.

ИНДИВИДУАЛЬНЫЕ ЗАДАНИЯ

Вариант 1

№1. Дан массив целых чисел, состоящий из 10 элементов. Найти произведение нечетных элементов.

№2. Дан массив целых чисел, состоящий из 10 элементов. Определить количество пар соседних элементов с одинаковыми знаками.

№3. Заменить максимальные элементы массива на минимальные.

№4. В массиве имеется только два одинаковых элемента. Найдите их.

Вариант 2

№1. Дан массив целых чисел, состоящий из 15 элементов. Найти сумму элементов, кратных 3.

№2. Дан массив целых чисел, состоящий из 10 элементов. Вывести индексы нулевых элементов.

№3. Заменить нулями элементы массива между минимальными и максимальными, кроме их самих.

№4. Найти количество различных элементов в массиве.

Вариант 3

№1. Дан массив целых чисел, состоящий из 15 элементов. Найти среднее арифметическое отрицательных элементов;

№2. Дан массив целых чисел, состоящий из 10 элементов. Найти номер первого нулевого элемента.

№3. Перенести в начало массива максимальный элемент

№4. Дан одномерный массив из 20 элементов. Переставить первые три и последние три элемента, сохранив порядок их следования.

Вариант 4

№1. Дан массив целых чисел, состоящий из 10 элементов. Вывести количество отрицательных элементов.

№2. Дан массив целых чисел, состоящий из 10 элементов. Определить, есть ли в данном массиве нулевые элементы и вывести номер первого нулевого элемента массива.

№3. Заменить последний положительный элемент массива на первый элемент массива.

№4. Дан массив не нулевых целых чисел. Определить, сколько раз элементы массива при просмотре от его начала меняют знак. Например, в массиве 10, -4, 12, 56, -4, -89 знак меняется 3 раза.

Вариант 5

№1. Дан массив целых чисел, состоящий из 10 элементов. Вывести количество тех элементов, значения которых больше значения предыдущего элемента (начиная со второго).

№2. Дан массив целых чисел, состоящий из 10 элементов. Найти номер элемента массива, значение которого положительно и не превосходит заданного числа В.

№3. Заменить минимальный положительный элемент массива нулем.

№4. Найти элемент массива, наиболее близкий к среднему значению всех элементов массива.

Вариант 6

№1. Дан массив целых чисел, состоящий из 10 элементов. Найти произведение четных элементов;

№2. Дан массив целых чисел, состоящий из 10 элементов. Определить, есть ли в данном массиве отрицательные элементы и найти номер последнего отрицательного элемента.

№3. Заменить первый отрицательный элемент массива на последний элемент массива.

№4. Дан массив. Определить количество элементов, больших суммы всех элементов массива и напечатать их номера.

Вариант 7

№1. Дан массив целых чисел, состоящий из 15 элементов. Найти среднее арифметическое положительных элементов;

№2. Дан массив целых чисел, состоящий из 10 элементов. Определить номер последнего положительного элемента, кратного k.

№3. Заменить первые k элементов массива на максимальный элемент.

№4. Дан массив из 20 элементов. Найти пять соседних элементов, сумма значений которых максимальна.

Вариант 8

№1. Дан массив целых чисел, состоящий из 15 элементов. Найти сумму отрицательных элементов, значения которых больше заданного числа А.

№2. Дан массив целых чисел, состоящий из 10 элементов. Номер последней пары соседних элементов с разными знаками.

№3. Найти максимальный элемент среди отрицательных элементов и поменять его местами с минимальным положительным.

№4. Найти число пар соседних элементов массива, оканчивающихся нулем.

Вариант 9

№1. Дан массив целых чисел, состоящий из 10 элементов. Найти сумму отрицательных элементов, модуль которых не превосходит 5;

№2. Дан массив целых чисел, состоящий из 10 элементов. Найти номер первого отрицательного элемента.

№3. Перенести в начало массива все положительные элементы.

№4. Определить сумму второго, четвертого, шестого и т.д. элементов массива.

Вариант 10

№1. Дан массив целых чисел, состоящий из 10 элементов. Найти сумму элементов, значения которых больше заданного числа А.

№2. Дан массив целых чисел, состоящий из 10 элементов. Найти сумму четных элементов, имеющих нечетные индексы;

№3. Заменить нулями все элементы массива, равные максимальному.

№4. Дан массив из 20 элементов. Найти пять соседних элементов, сумма значений которых минимальна.

ЛАБОРАТОРНАЯ РАБОТА №5

ДВУХМЕРНЫЕ МАССИВЫ В АССЕМБЛЕРЕ

ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

Описание и инициализация массива в программе

Специальных средств описания двумерных массивов в программах ассемблера нет. Чтобы использовать массив в программе, его нужно смоделировать. Можно перечислить элементы массива в поле операндов одной из директив описания данных. При перечислении элементы разделяются запятыми. Например, массив состоит из 2 строк и 2 столбов, то есть из четырех элементов. Размер каждого элемента 4 байта: `mas dd 1, 2, 3, 4`

Доступ к элементам массива

Специальных средств для описания двумерного массива в ассемблере нет. Его нужно моделировать. На описании самих данных это почти никак не отражается — память под массив выделяется с помощью директив резервирования и инициализации памяти. Непосредственно моделирование обработки массива производится в сегменте кода, где программист, описывая алгоритм обработки на ассемблере, определяет, что некоторую область памяти необходимо трактовать как двумерный массив. При этом вы вольны в выборе того, как понимать расположение элементов двумерного массива в памяти: по строкам или по столбцам. Таким образом, структура хранения остается прежней – вектор. Наиболее естественен порядок расположения элементов массива – по строкам. При этом наиболее быстро изменяется последний элемент последнего элемента индекса. Например, рассмотрим представление на логическом уровне двумерного массива A_{ij} размерностью $n \times m$, где $0 \leq i \leq n-1, 0 \leq j \leq m-1$.

$$\begin{array}{cccc}
 a_{00} & a_{01} & a_{02} & a_{03} \\
 a_{10} & a_{11} & a_{12} & a_{13} \\
 a_{20} & a_{21} & a_{22} & a_{23} \\
 a_{30} & a_{31} & a_{32} & a_{33}
 \end{array}$$

Соответственно этому массиву физическое представление в памяти – вектор будет выглядеть так:

$$a_{00}a_{01}a_{02}a_{03}a_{10}a_{11}a_{12}a_{13}a_{20}a_{21}a_{22}a_{23}a_{30}a_{31}a_{32}a_{33}.$$

Номер конкретного элемента массива в этой, уже ставшей линейной, последовательности определяется адресной функцией, которая устанавливает положение (адрес) в памяти этого элемента исходя из значения его индексов: $a_{ij} = n \cdot i + j$.

Для получения адреса элемента массива в памяти необходимо полученное знание умножить на размер элемента и сложить с базовым адресом массива.

Если последовательность однотипных элементов в памяти трактуется как двухмерный массив, расположенный по строкам, то адрес элемента (i, j) вычисляется по формуле:

$$(\text{база} + (\text{количество_элементов_в_строке} \cdot i + j) \cdot \text{размер_элемента})$$

Здесь $i = 0 \dots (n-1)$ — номер строки, а $j = 0 \dots (m-1)$ — номер столбца. Например, пусть имеется массив чисел (размером в 2 байта) $\text{mas}(i, j)$ с размерностью $4 \cdot 4$ ($i = 0 \dots 3, j = 0 \dots 3$):

```

23 04 05 67
05 06 07 99
67 08 09 23
87 09 00 08

```

В памяти элементы этого массива будут расположены в следующей последовательности:

```

23 04 05 67 05 06 07 99 67 08 09 23 87 09 00 08

```

Если мы хотим трактовать эту последовательность как двухмерный массив, приведенный раньше, и извлечь, например, элемент $\text{mas}(2, 3) = 23$, то, проведя нехитрый подсчет, убедимся в правильности наших рассуждений:

Эффективный адрес $\text{mas}(2, 3) = \text{mas} + (4 \cdot 2 + 3) \cdot 2 = \text{mas} + 22$.

Посмотрите на представление массива в памяти и убедитесь, что по этому смещению действительно находится нужный элемент массива.

Логично организовать адресацию двухмерного массива, используя базово-индексную адресацию. При этом возможны два основных варианта выбора компонентов для формирования эффективного адреса:

- сочетание прямого адреса как базового компонента адреса и двух индексных регистров для хранения индексов: `mov ax, [mas+ebx+esi]`
- сочетание двух индексных регистров, один из которых является и базовым, и индексным одновременно, а другой — только индексным: `mov ax, [ebx+esi]`

Таким образом, необходимо использовать для работы с двумерными массивами адресацию по базе с индексированием и самую полную адресацию - адресация по базе с индексированием и масштабированием.

ПРАКТИЧЕСКАЯ ЧАСТЬ

Пример №1

Найти сумму элементов двухмерного массива.

Решение

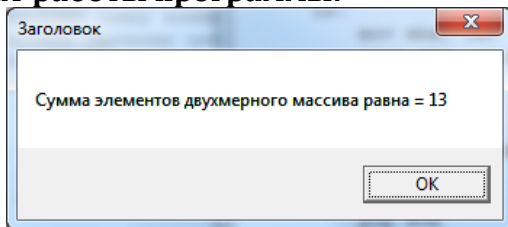
```
format PE GUI 4.0
include 'd:\FASM\INCLUDE\win32ax.inc'
.data
Caption db 'Заголовок', 0
Text db 'Сумма элементов двухмерного массива равна = %d', 0
buf db 100 dup(?)
n dd 3 ; кол-во строк
m dd 4 ; кол-во столбцов
mass dd 3, -2, 7, 9, -3, 4, -1, 0, 4, -2, 1, -7
.code
start:
```

```

mov esi, 0
mov eax, 0
mov ecx, [n]
11:  push ecx
    mov ecx, [m]
    mov edi, 0
    12:
        mov ebx, [m]                ; Форми-
        imul ebx, esi                ; руем
        add ebx, edi                ; смеще-
        imul ebx, 4                 ; ние
        add eax, [mass+ebx]         ; сумма
        add edi, 1
    loop 12
    pop ecx
    add esi, 1
    loop 11
invoke wsprintf, addr buf, addr Text, eax
invoke MessageBox, 0, addr buf, addr Caption, MB_OK
invoke ExitProcess, 0
.end start

```

Результат работы программы:



Пример №2

Дан двумерный массив целых чисел размером $m \times n$.

- Посчитать сумму положительных элементов массива.
- Заменить отрицательные элементы нулем.
- Поменять местами первый и последний столбцы.
- Выведите массив на экран после изменения.

Решение:

format PE console 4.0

```

entry start
include 'd:\FASM\INCLUDE\win32a.inc'
section '.data' data readable writable
tpt db '%d',0
rez1 db 'Summa mass[i]>0 = %d',0
rez2 db 13,10,'mass[%d,%d] = %d',0
n dd 3 ;количество строк
m dd 4 ;количество столбцов
mass dd 3,-2, 7, 9, -3, 4,-1, 0, 4, -2, 1, -7
buf rd 255
section '.code' code readable executable
start:
    mov eax, 0
    mov esi, 0
    mov ecx, [n]
l1:    push ecx
    mov ecx, [m]
    mov edi, 0
    12:
        mov ebx, [m] ;Форми-
        imul ebx, esi ;руем
        add ebx, edi ;смеще-
        imul ebx, 4 ;ние
;находим сумму положительных эл-тов массива
        cmp [mass+ebx], 0
        jl p1
        add eax, [mass+ebx]
        p1: nop
;заменяем отриц. эл-ты массива на нули
        cmp [mass+ebx], 0
        jge p2
        mov [mass+ebx], 0
        p2:nop
    add edi, 1
    loop 12
    pop ecx
    add esi, 1
    loop l1

```

```

cinvoke printf, rez1, eax
;меняем местами 1-йи последний столбцы
mov ecx, [m]
mov ebx, 3 ;номер последнего столбца
mov edx, 0 ;номер первого столбца
q1:
    mov esi, [m]
    imul esi, edx
    imul esi, 4
    mov edi, ebx
    imul edi, 4
    add edi, esi
    push eax
    push ecx
        mov eax, [mass+esi]
        mov ecx, [mass+edi]
        mov [mass+esi], ecx
        mov [mass+edi], eax
    pop ecx
    pop eax
    add edx, 1
    loop q1
;Вывод эл-тов двухмерного массива
    mov esi, 0
    mov ebx, 0
    mov edi, 0
q2:
    mov edi, 0
    q3:
        mov ebx, [m] ;Форми-
        imul ebx, esi ;руем
        add ebx, edi ;смеще-
        imul ebx, 4 ;ние
cinvoke printf, rez2, esi, edi, [mass+ebx]
    inc edi
    cmp edi, [m]
    jne q3
    inc esi

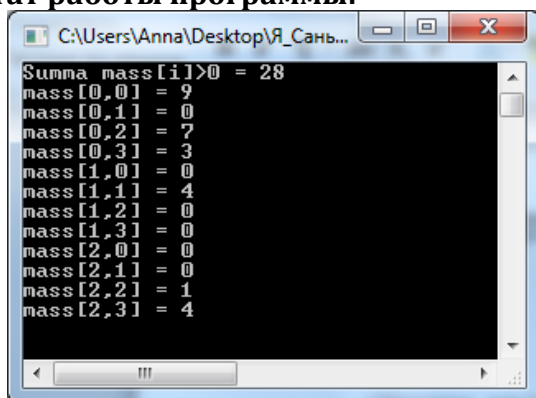
```

```

    cmp esi, [n]
    jne q2
invoke sleep, 10000
invoke exit, 0
section '.idata' import data readable writeable
library user32, 'USER32.DLL', msvcrt, 'MSVCRT.DLL',
kernel32, 'KERNEL32.DLL'
import kernel32, sleep, 'Sleep'
import msvcrt, puts, 'puts', scanf, 'scanf',
printf, 'printf', exit, 'exit'

```

Результат работы программы:



```

Summa mass[i][j]>0 = 28
mass[0,0] = 9
mass[0,1] = 0
mass[0,2] = 7
mass[0,3] = 3
mass[1,0] = 0
mass[1,1] = 4
mass[1,2] = 0
mass[1,3] = 0
mass[2,0] = 0
mass[2,1] = 0
mass[2,2] = 1
mass[2,3] = 4

```

Далее показана фрагмент программы, где реализован вывод, используя функцию MessageBox.

; вывод эл-тов двухмерного массива

```

    mov esi, 0
    mov ebx, 0
    mov edi, 0
q2:
    mov edi, 0
q3:
    mov ebx, [m]                ; Форми-
    imul ebx, esi                ; руем
    add ebx, edi                 ; смеще-
    imul ebx, 4                  ; ние
invoke wsprintf, addr buf, addr rez2, esi, edi, [mass+ebx]
invoke MessageBox, 0, addr buf, addr Caption, MB_OK
invoke ExitProcess, 0

```

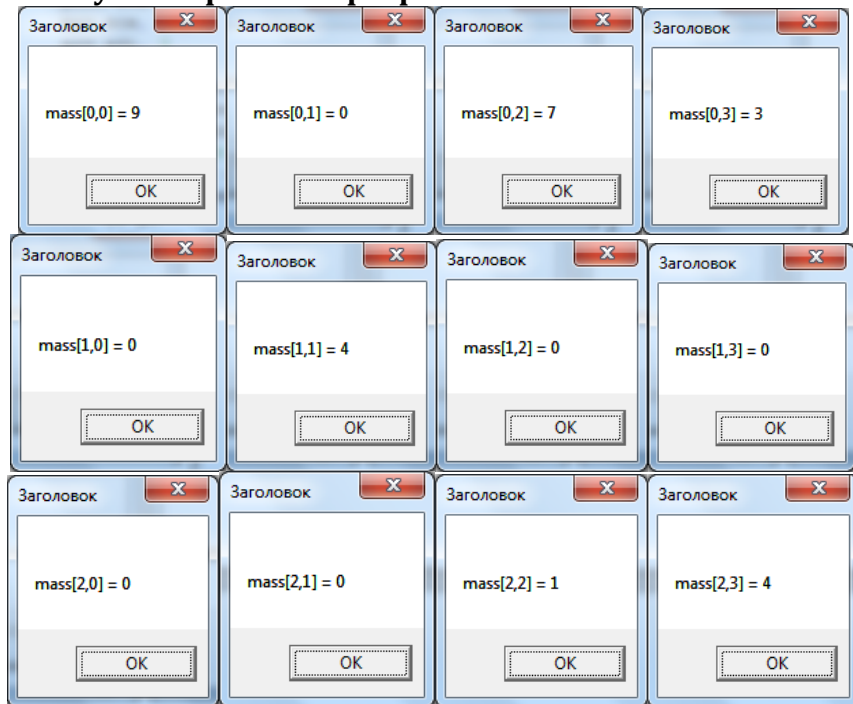


```

    inc edi
    cmp edi, [m]
    jne q3
    inc esi
    cmp esi, [n]
    jne q2
invoke ExitProcess, 0
.end start

```

Результат работы программы:



ЗАДАНИЯ ДЛЯ САМОСТОЯТЕЛЬНОЙ РАБОТЫ

1. Ознакомьтесь с теоретическим материалом.
2. Разберите все примеры из практической части лабораторной работы, т.е. наберите и просмотрите их работу.
3. Выполните индивидуальные задания.

ИНДИВИДУАЛЬНЫЕ ЗАДАНИЯ

Вариант 1

№1. Дан двумерный массив целых чисел (3 строки, 4 столбца).

- поменяйте местами 3-й элемент 2-й строки и 2-й элемент 4-го столбца;
- подсчитайте сумму всех положительных элементов массива;
- подсчитайте количество отрицательных элементов массива;
- выведите массив на экран после изменения.

№2. Дан двумерный массив целых чисел размером $n \times m$.

- Заменить максимальный элемент каждой строки на противоположный по знаку.
- Поменять местами первый и последний столбцы.
- Выведите массив на экран после изменения.

Вариант 2

№1. Дан двумерный массив целых чисел (3 строки, 5 столбцов).

- поменяйте местами 2-й элемент 3-й строки и 3-й элемент 5-го столбца;
- все нечетные элементы замените нулем;
- подсчитайте сумму всех отрицательных элементов массива;
- выведите массив на экран после изменения.

№2. Дан двумерный массив целых чисел размером $n \times m$.

- Заменить минимальный элемент каждого столбца нулем.
- Поменять местами первую и предпоследнюю строки.
- Выведите массив на экран после изменения.

Вариант 3

№1. Дан двумерный массив целых чисел (4 строки, 3 столбца).

- поменяйте местами 2-й элемент 4-й строки и 3-й элемент 2-го столбца;
- подсчитайте сумму элементов 3-го столбца;
- все отрицательные элементы увеличьте в 3 раза;

- выведите массив на экран после изменения.
- №2. Дан двумерный массив целых чисел размером $n \times m$.
- Заменить максимальный элемент каждой строки нулем.
 - Поменять местами второй и предпоследний столбцы.
 - Выведите массив на экран после изменения.

Вариант 4

№1. Дан двумерный массив целых чисел (3 строки, 4 столбца).

- поменяйте местами 2-й элемент 3-й строки и 1-й элемент 4-го столбца;
- подсчитайте произведение элементов 3-го столбца;
- все положительные элементы увеличьте в 2 раза;
- выведите массив на экран после изменения.

№2. Дан двумерный массив целых чисел размером $n \times m$.

- Заменить минимальный по модулю элемент каждого столбца на противоположный.
- Поменять местами первую и последнюю строки.
- Выведите массив на экран после изменения.

Вариант 5

№1. Дан двумерный массив целых чисел (2 строк, 5 столбцов).

- поменяйте местами 2-й элемент 1-й строки и 3-й элемент 4-го столбца;
- подсчитайте произведение элементов 2-го столбца;
- все положительные элементы увеличьте на 3;
- выведите массив на экран после изменения.

№2. Дан двумерный массив целых чисел размером $n \times m$.

- Заменить все элементы первых трех столбцов на их квадраты.
- Поменять местами средние строки с первой и последней.
- Выведите массив на экран после изменения.

Вариант 6

№1. Дан двумерный массив целых чисел (5 строк, 3 столбца).

- поменяйте местами 1-й элемент 3-го столбца и 2-й элемент 3-й строки;

- подсчитайте произведение элементов 2-й строки;
- подсчитайте количество нечетных элементов;
- выведите массив на экран после изменения.

№2. Дан двумерный массив целых чисел размером $n \times m$.

- Заменить первый отрицательный элемент каждого столбца нулем.
- Поменять местами второй и последний столбцы.
- Выведите массив на экран после изменения.

Вариант 7

№1. Дан двумерный массив целых чисел (3 строки, 5 столбцов).

- поменяйте местами 3-й элемент 4-го столбца и 1-й элемент 2-й строки;
- подсчитайте сумму элементов 2-го столбца;
- все положительные элементы увеличьте на 3;
- выведите массив на экран после изменения.

№2. Дан двумерный массив целых чисел размером $n \times m$.

- Заменить максимальный по модулю элемент каждой строки на противоположный по знаку.
- Поменять местами средние столбцы.
- Выведите массив на экран после изменения.

Вариант 8

№1. Дан двумерный массив целых чисел (2 строки, 5 столбцов).

- поменяйте местами 2-й элемент 5-го столбца и 3-й элемент 2-й строки;
- подсчитайте сумму всех четных элементов массива;
- подсчитайте количество элементов, не превосходящих 5;
- выведите массив на экран после изменения.

№2. Дан двумерный массив целых чисел размером $n \times m$.

- Заменить последний нулевой элемент каждого столбца максимальным по модулю элементом массива.
- Поменять местами вторую и предпоследнюю строки.
- Выведите массив на экран после изменения.

Вариант 9

№1. Дан двумерный массив целых чисел (4 строки, 5 столбцов).

- поменяйте местами 2-й элемент 3-й строки и 4-й элемент 5-го столбца;
- подсчитайте произведение элементов 3-го строки;
- подсчитайте количество четных элементов;
- выведите массив на экран после изменения.

№2. Дан двумерный массив целых чисел размером $n \times m$.

- Заменить минимальный по модулю элемент каждого столбца нулем.
- Поменять местами первую и последнюю строки.
- Выведите массив на экран после изменения.

Вариант 10

№1. Дан двумерный массив целых чисел (2 строки, 5 столбцов).

- поменяйте местами 2-й элемент 4-го столбца и 3-й элемент 1-й строки;
- подсчитайте количество отрицательных элементов массива;
- все положительные элементы массива обнулите;
- выведите массив на экран после изменения.

№2. Дан двумерный массив целых чисел размером $n \times m$.

- Заменить максимальный элемент каждого столбца нулем.
- Поменять местами последний и предпоследний столбцы.
- Выведите массив на экран после изменения.

ЛАБОРАТОРНАЯ РАБОТА №6

ПРОЦЕДУРЫ В АССЕМБЛЕРЕ

ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

Процедура, или *подпрограмма*, — это основная функциональная единица декомпозиции (разделения на части) некоторой задачи. Процедура представляет собой группу команд для решения конкретной подзадачи и обладает средствами получения управления из точки вызова задачи более высокого уровня и возврата управления в эту точку. В простейшем случае программа может состоять из одной процедуры. Другими словами, процедуру можно определить как правильным образом оформленную совокупность команд, которая, будучи однократно описана, при необходимости может быть вызвана в любом месте программы. Для описания последовательности команд в виде процедуры в языке ассемблера используются две директивы: PROC и ENDP.

Синтаксис описания процедуры таков:

ИМЯ_ПРОЦЕДУРЫ PROC [[МОДИФИКАТОР_ЯЗЫКА] ЯЗЫК] [РАССТОЯНИЕ]	}	Заголовок процедуры
[ARG СПИСОК_АРГУМЕНТОВ]		
[RETURN СПИСОК_ЭЛЕМЕНТОВ]	}	Тело процедуры
...		
команды, директивы ассемблера		
...		
[ИМЯ_ПРОЦЕДУРЫ] ENDP	}	Конец процедуры

Рисунок 3. Синтаксис описания процедуры в программе

Из рисунка видно, что в заголовке процедуры (директиве PROC) обязательным является только задание имени процедуры. Среди большого количества операндов директивы PROC следует особо выделить [расстояние]. Этот атрибут может принимать значения NEAR или FAR и характеризует возможность обращения к процедуре из другого сегмента кода. По умолчанию атрибут [расстояние] принимает значение NEAR. Процедура может размещаться в любом месте программы, но так, чтобы на нее случайным образом не попало управление. Если процедуру просто вставить в общий поток команд, то процессор воспримет

команды процедуры как часть этого потока и, соответственно, начнет выполнять эти команды. Учитывая это обстоятельство, есть следующие варианты размещения процедуры в программе:

- в начале программы (до первой исполняемой команды);
- в конце программы (после команды, возвращающей управление операционной системе);
- промежуточный вариант — внутри другой процедуры или основной программы (в этом случае необходимо предусмотреть обход процедуры с помощью команды безусловного перехода JMP);
- в другом модуле (библиотеке DLL).

Размещение процедуры в начале сегмента кода предполагает, что последовательность команд, ограниченная парой директив PROC и ENDP, будет размещена до метки, обозначающей первую команду, с которой начинается выполнение программы. Эта метка должна быть указана как параметр директивы END, обозначающей конец программы:

Объявление имени процедуры в программе равнозначно объявлению метки, поэтому директиву PROC в частном случае можно рассматривать как завуалированную форму определения программной метки. Поэтому сама исполняемая программа также может быть оформлена в виде процедуры, что довольно часто и делается с целью пометить первую команду программы, с которой должно начаться выполнение. При этом не забывайте, что имя этой процедуры нужно обязательно указывать в заключительной директиве END.

Размещение процедуры в конце программы предполагает, что последовательность команд, ограниченная директивами PROC и ENDP, находится следом за командой, возвращающей управление операционной системе.

Промежуточный вариант расположения тела процедуры предполагает ее размещение внутри другой процедуры или основной программы. В этом случае необходимо предусмотреть обход тела процедуры, ограниченного директивами PROC и ENDP, с помощью команды безусловного перехода JMP.

Последний вариант расположения описаний процедур — в *отдельном сегменте кода* — предполагает, что часто используемые процедуры выносятся в отдельный файл, который должен быть оформлен как обычный исходный файл и подвергнут трансляции для получения объектного кода. Впоследствии этот объектный файл с помощью утилиты `tlink` можно объединить с файлом, в котором данные процедуры используются. Этот способ предполагает наличие в исходном тексте программы еще некоторых элементов, связанных с особенностями реализации концепции модульного программирования в языке ассемблера.

Процедуры в FASM

Для создания процедуры используется следующий синтаксис:

```
proc <имя_процедуры>[, ] [<список_параметров>]  
    ...  
    ret  
endp
```

После `proc` указывается имя процедуры. Далее через запятую список параметров. Между именем процедуры и списком параметров запятую ставить не обязательно (можно просто поставить пробел). Для возврата из процедуры следует использовать команду `RET` без операндов. Завершается процедура макросом `endp`. Например, объявим процедуру с тремя параметрами:

```
proc myproc, a, b, c  
    mov ax, [b]  
    ...  
    ret  
endp
```

Внутри процедуры обращаться к параметрам можно как к простым переменным — с помощью квадратных скобок! При вызове процедуры параметры должны помещаться в стек, начиная с последнего.

Сохранение и восстановление используемых регистров

Макросы PROC и ENDP позволяют также организовать сохранение и восстановление регистров, используемых кодом процедуры. Для этого после имени процедуры нужно указать ключевое слово `uses` и список регистров через пробел. Регистры будут помещены в стек при входе в процедуру (в порядке их записи) и восстановлены перед возвратом. Если объявление процедуры получается слишком длинным, можно продолжить его на следующей строке, добавив символ «\» в конец первой строки (это работает и с любыми другими макросами).

Объявление локальных переменных

Для объявления локальных переменных существует 3 варианта синтаксиса:

1. local + директивы объявления данных

Макрос `local` предназначен для создания локальных переменных. После слова `local` локальные переменные объявляются обычными директивами. Можно использовать как инициализированные, так и неинициализированные переменные. Можно объявлять несколько переменных в одной строке через запятую (однако, не получится объявить массив, перечислив значения). После объявления обращаться к локальным переменным можно так же, как к параметрам и глобальным переменным.

2. Альтернативный синтаксис local

Альтернативный синтаксис похож на синтаксис размеров параметров. После имени переменной ставится двоеточие и оператор размера. Вместо оператора размера может быть также имя структуры (это удобно при программировании для Windows). Инициализация переменных в этом варианте синтаксиса не предусмотрена, её придётся делать вручную. Можно легко объявлять массивы — обозначение как в языках высокого уровня.

3. locals и endl

Третий вариант — объявление локальных переменных в виде блока `locals — endl`. Используются обычные директивы объявления данных.

Этот способ хорошо подходит, если в процедуре много локальных переменных.

Макросы для вызова процедур

Существуют макросы для удобного вызова процедур. Эти макросы избавляют от необходимости писать несколько команд PUSH для помещения параметров в стек. Вместо этого достаточно написать одну строку, в которой указывается имя процедуры и список параметров через запятую. Например:

```
stdcall <имя_процедуры>[, <список_параметров>]
```

Всего существует 4 разных макроса, они перечислены в таблице. Два последних макроса чаще используются в программировании для Windows. Они выполняют вызов процедуры, адрес которой находится в переменной.

Таблица 3
Макрос

Макрос	Описание
stdcall	Вызов процедуры с соглашениями вызова stdcall
ccall	Вызов процедуры с соглашениями вызова с
invoke	То же самое, что stdcall [<имя_переменной>]
cinvoke	То же самое, что ccall [<имя_переменной>]

Важно!!! Если вы объявили процедуру, но ни разу не вызывали, то её код не будет добавлен в исполняемый файл! Это позволяет создать свою библиотеку полезных процедур, сохранить их в отдельном файле и добавлять во все проекты, где они нужны. При этом в исполняемом файле окажутся только те процедуры, которые использует ваша программа.

ПРАКТИЧЕСКАЯ ЧАСТЬ

Пример №1

Определить сколько чисел являющихся палиндромами находятся в диапазоне от 100 до 300. (Определить процедуру нахождения палиндрома)

Решение

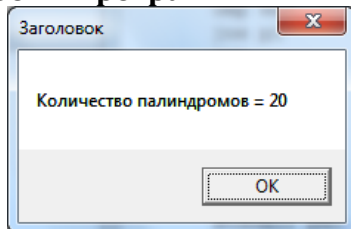
```
format PE GUI 4.0
include 'd:\FASM\INCLUDE\win32ax.inc'
.data
Caption db 'Заголовок',0
Text db 'Количество палиндромов = %d',0
```

```

buf db 100 dup(?)
proc pal uses eax ebx edx, x
    mov eax, [x]
    mov ebx, 10
    mov ecx, 0
p1:
    imul ecx, 10
    cdq
    div ebx
    add ecx, edx
    cmp eax, 0
    jne p1
    ret
endp
.code
start:
    mov ebx, 0
    mov eax, 100
l1:
    stdcall pal, eax
    cmp ecx, eax
    jne l2
    add ebx, 1
l2:
    inc eax
    cmp eax, 300
    jne l1
invoke wsprintf, addr buf, addr Text, ebx
invoke MessageBox, 0, addr buf, addr Caption, MB_OK
invoke ExitProcess, 0
.end start

```

Результат работы программы:



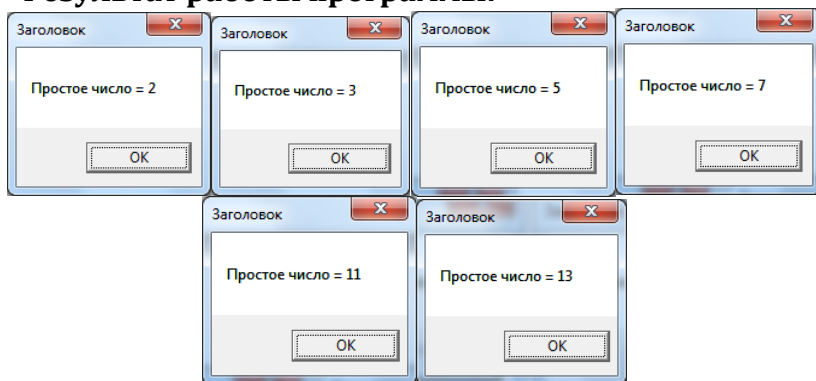
Пример №2

Вывести все простые числа из диапазона от а до b.
(Определить процедуру нахождения количества делителей
числа).

Решение

```
format PE GUI 4.0
include 'd:\FASM\INCLUDE\win32ax.inc'
.data
Caption db 'Заголовок',0
Text db 'Простое число = %d',0
buf db 100 dup(?)
proc delit_ch uses eax ebx edx, x
    mov ebx, [x]
    mov ecx, 0
p1:
    mov eax, [x]
    cdq
    div ebx
    cmp edx, 0
    jne p2
    inc ecx
p2:
    dec ebx
    cmp ebx, 0
    jne p1
ret
endp
.code
start:
    mov ebx, [a]
l1:
    stdcall delit_ch, ebx
    cmp ecx, 2
    jne l2
invoke sprintf, addr buf, addr Text, ebx
invoke MessageBox,0, addr buf, addr Caption, MB_OK
l2: inc ebx
    cmp ebx, [b]
    jne l1
invoke ExitProcess, 0
.end start
```

Результат работы программы:



ЗАДАНИЯ ДЛЯ САМОСТОЯТЕЛЬНОЙ РАБОТЫ

1. Ознакомьтесь с теоретическим материалом.
2. Разберите все примеры из практической части лабораторной работы.
3. Выполните индивидуальные задания.

ИНДИВИДУАЛЬНЫЕ ЗАДАНИЯ

Вариант 1

№1. Даны 3 натуральных числа. Выяснить, в каком из них больше цифр. (Определить процедуру для расчета количества цифр натурального числа.)

№2. Вывести сумму простых трехзначных чисел. (Определить процедуру, позволяющую распознавать простые числа.)

№3. Написать процедуру для вывода на экран цифр натурального числа в обратном порядке.

Вариант 2

№1. Даны 3 натуральных числа. Определить количество чисел являющихся степенями двойки. (Определить процедуру позволяющую распознать степени двойки.)

№2. Найти наибольший общий делитель трех натуральных чисел, имея в виду, что $\text{НОД}(a,b,c) = \text{НОД}(\text{НОД}(a,b),c)$. (Определить процедуру для расчета наибольшего общего делителя двух натуральных чисел, используя алгоритм Евклида).

№3. Написать процедуру перевода натурального числа из десятичной системы счисления в двоичную.

Вариант 3

$$\frac{2*5!+3*8!}{6!+4!}$$

№1. Найти значение выражения $\frac{2*5!+3*8!}{6!+4!}$. (Определить процедуру для вычисления факториала.)

№2. Даны n натуральных чисел. Найти НОД, используя алгоритм Евклида. (Определить процедуру для расчета наибольшего общего делителя двух натуральных чисел, используя алгоритм Евклида).

№3. Написать процедуру, определяющую, является ли симметричной часть числа, начиная с i -го элемента и кончая j -м.

Вариант 4

№1. Даны 3 натуральных числа. Выяснить, в каком из них сумма цифр больше. (Определить процедуру для расчета суммы цифр натурального числа.)

№2. Даны 2 натуральных числа a и b , обозначающих числитель и знаменатель дроби. Сократить дробь, т.е. найти такие натуральные числа p и q , не имеющих общих делителей, что $p/q=a/b$. Найти их наименьшее общее кратное. (Определить процедуру для расчета наибольшего общего делителя двух натуральных чисел, используя алгоритм Евклида).

№3. Для заданного натурального N определите N простых чисел.

Вариант 5

№1. Даны 3 натуральных числа. Определить количество чисел являющихся полными квадратами. (Определить процедуру позволяющую распознать полные квадраты.)

№2. Вывести сумму простых двузначных чисел. (Определить процедуру, позволяющую распознавать простые числа.)

№3. Дано n различных натуральных чисел. Напечатать все перестановки этих чисел.

Вариант 6

№1. Даны 3 натуральных числа. Определить количество чисел являющихся степенями шестерки. (Определить процедуру позволяющую распознать степени шестерки.)

№2. Найти наибольший общий делитель трех натуральных чисел, имея в виду, что $\text{НОД}(a,b,c)=\text{НОД}(\text{НОД}(a,b),c)$. (Определить процедуру для расчета наибольшего общего делителя двух натуральных чисел, используя алгоритм Евклида).

№3. Даны целое число A , натуральное число n . Организовать вычисление A^n с помощью процедуры.

Вариант 7

№1. Найти значение выражения $(2!)^2 + (3!)^2 + (4!)^2$. (Определить процедуру для вычисления факториала.)

№2. Даны 2 натуральных числа a и b , обозначающих числитель и знаменатель дроби. Сократить дробь, т.е. найти такие натуральные числа p и q , не имеющих общих делителей, что $p/q=a/b$. Найти их наименьшее общее кратное. (Определить процедуру для расчета наибольшего общего делителя двух натуральных чисел, используя алгоритм Евклида).

№3. Составить процедуру нахождения N -го числа Фибоначчи: 0, 1, 1, 2, 3, 5, 8, ..., то есть каждое последующее число равно сумме двух предыдущих.

Вариант 8

№1. Даны 3 натуральных числа. Выяснить, в каком из них произведение цифр больше. (Определить процедуру для расчета произведения цифр натурального числа.)

№2. Вывести сумму простых четырехзначных чисел. (Определить процедуру, позволяющую распознавать простые числа.)

№3. Написать процедуру перевода натурального числа из десятичной системы счисления в восьмеричную.

Вариант 9

№1. Даны 3 натуральных числа. Определить количество чисел являющихся степенями тройки. (Определить процедуру позволяющую распознать степени тройки.)

№2. Даны n натуральных чисел. Найти НОД, используя алгоритм Евклида. (Определить процедуру для расчета

наибольшего общего делителя двух натуральных чисел, используя алгоритм Евклида).

№3. Написать процедуру перевода натурального числа из десятичной системы счисления в семеричную.

Вариант 10

№1. Даны 3 натуральных числа. Определить количество чисел являющихся кубами чисел. (Определить процедуру позволяющую распознать куб числа.)

№2. Определить количество «близнецов» двузначных чисел. («Близнецами» называются простые числа отличающиеся друг от друга на 2, например, 41 и 43). Вывести сумму простых трехзначных чисел. (Определить функцию, позволяющую распознавать простые числа.)

№3. Написать процедуру перевода натурального числа из десятичной системы счисления в троичную.

ЛИТЕРАТУРА

1. Бурдаев О. В., Иванов М. А., Тетерин И. И., Ассемблер в задачах защиты информации / Под ред. И. Ю. Жукова - М.: КУДИЦ-ОБРАЗ, 2002. - 320 с.
2. Златопольский Д. М., Сборник задач по программированию. — 3-е изд., перераб. и доп. — СПб.: БХВ-Петербург, 2011. — 304 с.
3. Зубков С. В., Assembler для DOS, Windows и UNIX. 3-е издание., М.: ДМК Пресс; СПб.: Питер, 2004. 608 с.
4. Калашников О. А., Ассемблер? Это просто! Учимся программировать. – СПб.: БХВ-Петербург, 2006. – 384 с.
5. Крупник А. Б., Изучаем Ассемблер., СПб.: Питер, 2005. – 249 с.
6. Магда Ю. С., Использование ассемблера для оптимизации программ на C++. — СПб.: БХВ-Петербург, 2004. — 496 с:
7. Пирогов В. Ю., ASSEMBLER. Учебный курс. , М.: Издатель Молгачева С. В., Издательство Нолидж, 2001. – 848 с.
8. Юров В. И., Assembler. Учебник для вузов. 2-е издание., СПб.: Питер, 2003.- 637 с.

ПРИДНЕСТРОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
им. Т.Г. Шевченко

физико-математический факультет

кафедра прикладной математики и информатики

ОТЧЁТ

по лабораторной работе № __

по дисциплине Физические основы построения ЭВМ

по теме «_____»

Выполнил:

студент(ка) __ гр. ФМФ

Проверил:

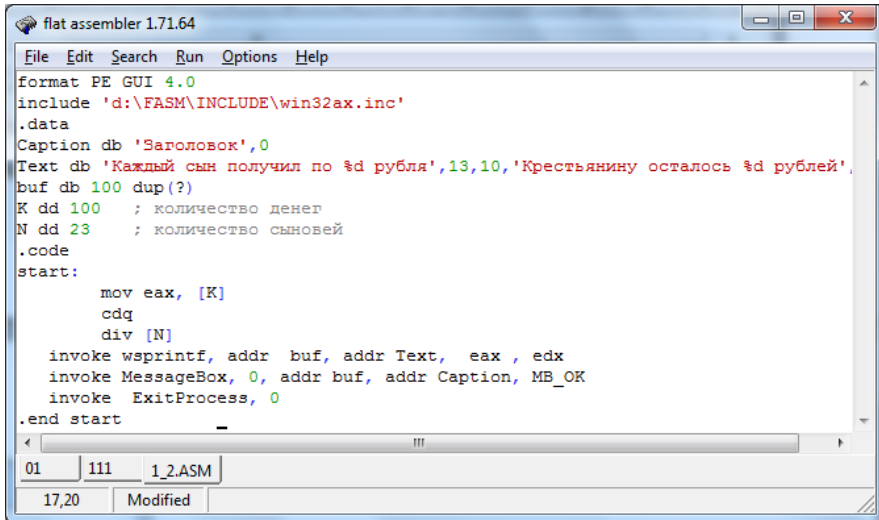
ст. преподаватель кафедры ПМий

г. Тирасполь, 201_г.

ПРИМЕР ОФОРМЛЕНИЯ ИНДИВИДУАЛЬНОГО ЗАДАНИЯ

Задание №1

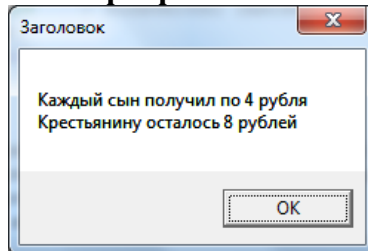
Составить программу для решения задачи: «У крестьянина N сыновей и K рублей. Он разделил все деньги поровну между сыновьями. Сколько рублей получил каждый сын, и сколько осталось у крестьянина?».

Решение


```

flat assembler 1.71.64
File Edit Search Run Options Help
format PE GUI 4.0
include 'd:\FASM\INCLUDE\win32ax.inc'
.data
Caption db 'Заголовок',0
Text db 'Каждый сын получил по %d рубля',13,10,'Крестьянину осталось %d рублей',
buf db 100 dup(?)
K dd 100 ; количество денег
N dd 23 ; количество сыновей
.code
start:
    mov eax, [K]
    cdq
    div [N]
    invoke wsprintf, addr buf, addr Text, eax, edx
    invoke MessageBox, 0, addr buf, addr Caption, MB_OK
    invoke ExitProcess, 0
.end start

```

Результат работы программы

Учебно-практическое издание

Программирование на языке ассемблера

Часть I

Лабораторный практикум

Издаётся в авторской редакции

Подписано в печать 08.12.17
Формат 60×84/16. Усл. печ. л. 5,0.
Заказ №91. Тираж 10 экз.

Отпечатано с готового оригинал-макета
Типографией ООО «РВТ»
3200, г. Бендеры, ул. Московская, д. 30