

Ю.В. Кольцов  
О.В. Гаркуша  
Н.Ю. Добровольская  
А.В. Харченко

# ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ АСЕМБЛЕРА IA-32 В СРЕДЕ RADAsm

Учебное пособие

A decorative graphic featuring several overlapping snippets of assembly code in a light blue, monospaced font. The snippets include instructions like MOV, ADD, and LOOP, along with register names such as EAX, ECX, and EBX. Some snippets are partially obscured by others, creating a layered effect. The code is arranged in a way that suggests a loop or a sequence of operations.

MOV EAX, 0  
MOV ECX, ECX  
L: ADD EAX, ECX  
LOOP L  
...

Краснодар  
2014

Министерство образования и науки Российской Федерации  
КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Ю.В. КОЛЬЦОВ, О.В. ГАРКУША,  
Н.Ю. ДОБРОВОЛЬСКАЯ, А.В. ХАРЧЕНКО

ПРОГРАММИРОВАНИЕ  
НА ЯЗЫКЕ АССЕМБЛЕРА IA-32  
В СРЕДЕ RADAsm

Учебное пособие

Краснодар  
2014

УДК 004.43(075.8)  
ББК 32.973-018.1я73  
П 691

Рецензенты:  
Доктор технических наук, профессор  
*В.Н. Марков*  
Кандидат физико-математических наук, доцент  
*М.Е. Бегларян*

**Кольцов, Ю.В., Гаркуша, О.В., Добровольская, Н.Ю.,  
Харченко, А.В.**

П 691 Программирование на языке ассемблера IA-32 в среде  
RADAsm: учеб. пособие / Ю.В. Кольцов, О.В. Гаркуша,  
Н.Ю. Добровольская, А.В. Харченко. — Краснодар:  
Кубанский гос. ун-т, 2014. — 38 с.  
ISBN 978-5-8209-1055-5

Содержит сведения по основам программирования на языке  
ассемблера IA-32 в среде RADAsm. Приведены многочисленные  
примеры, иллюстрирующие материал.

Адресуется студентам факультета компьютерных технологий  
и прикладной математики, изучающим основы  
программирования.

УДК 004.43(075.8)  
ББК 32.973-018.1я73

ISBN 978-5-8209-1055-5

© Кубанский государственный  
университет, 2014

## ПРЕДИСЛОВИЕ

Изучение архитектуры современных ПК, программирование на машинно-ориентированном языке является необходимой частью подготовки профессиональных программистов. Знание языка ассемблера позволяет лучше понять принципы работы ЭВМ, операционных систем и трансляторов с языков высокого уровня, разрабатывать высокоэффективные программы.

Masm32 – специализированный пакет для программирования на языке ассемблера IA-32. Являясь продуктом фирмы Microsoft, он максимально приспособлен для создания Windows-приложений на ассемблере. Кроме транслятора, компоновщика и необходимых библиотек пакет Masm32 включает сравнительно простой текстовый редактор и некоторые инструменты, предназначенные для облегчения программирования на ассемблере. Однако набор инструментов не содержит 32-разрядного отладчика и предполагает работу в командном режиме, что не очень удобно.

Для создания программ можно использовать специализированную интегрированную среду RADAsm, которая помимо других ассемблеров позволяет использовать Masm32. Точнее используется специально настроенная среда – «сборка» RADAsm + OlleDBG, где OlleDBG – 32-разрядный отладчик.

В учебном пособии рассматривается последовательность действий при разработке приложений на ассемблере в среде RADAsm, кроме того, указываются особенности архитектуры процессоров семейства IA-32.

## 1. НАЧАЛО РАБОТЫ СО СРЕДОЙ

Программная среда иницируется запуском программы RADAsm.exe.

После вызова на экране появляется окно среды RADAsm, где обычно высвечивается последняя программа, с которой работал пользователь в предыдущий сеанс (рис. 1).

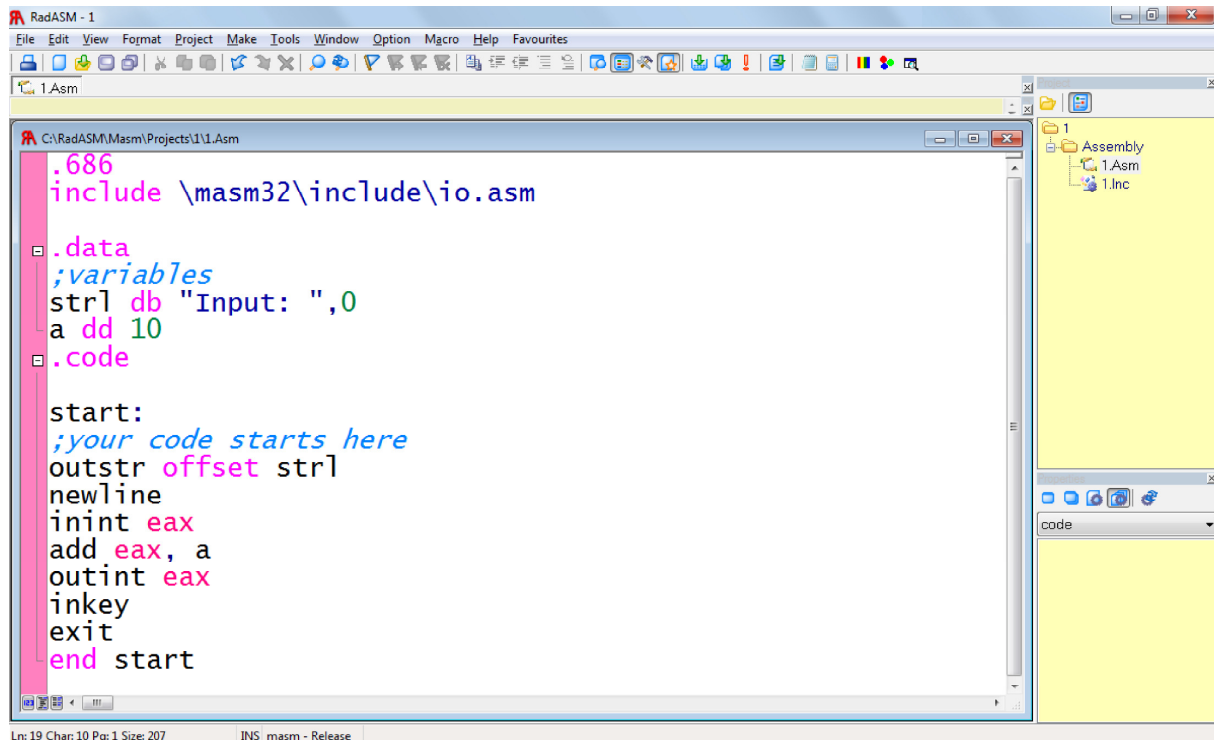


Рис. 1. Окно интегрированной среды RADAsm

Для создания нового проекта необходимо выбрать пункт меню **Файл (File)/Новый проект (New Project)**, после чего на экране появится первое окно четырехоконного Мастера создания проекта (рис. 2).

В этом окне необходимо выбрать тип проекта – в нашем случае Console App (консольное приложение), а также ввести его имя, например, Ex01, описание, например, «Пример № 1», и путь к создаваемой средой новой папке с именем проекта.

В следующем окне Мастера выбирается шаблон проекта (conapp.tpl), специально созданный для лабораторных работ шаблон консольного приложения.

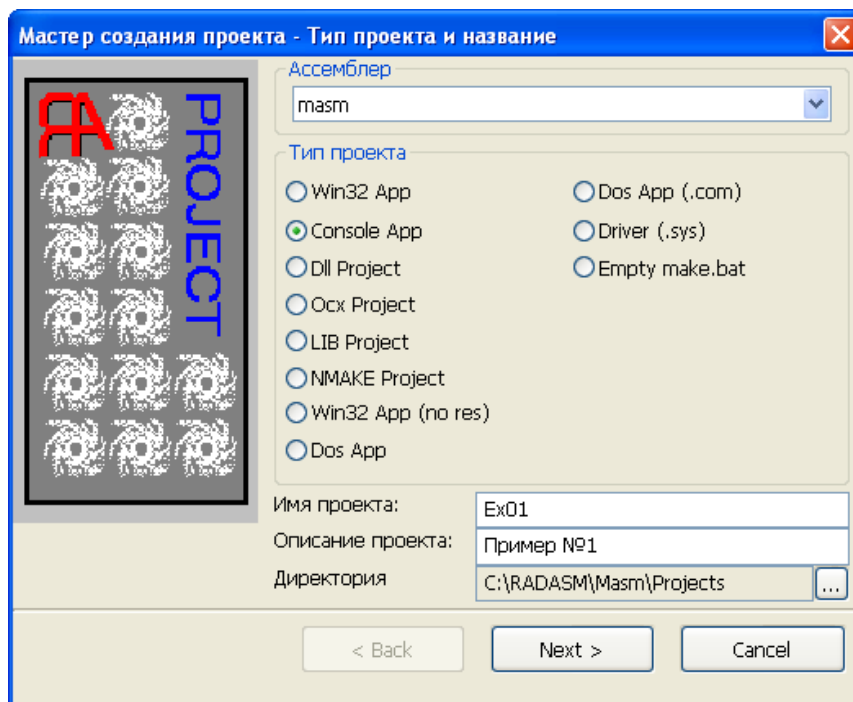


Рис. 2. Окно мастера создания нового проекта

Предпоследнее окно Мастера предлагает выбрать типы создаваемых файлов – выбираем Asm (исходные файлы ассемблера), и папки – выбираем папку Bak, используемую для размещения предыдущих копий файлов.

Последнее окно Мастера определяет доступные для работы с проектом пункты меню запуска приложения **Создать** и выполняемые команды. В данной сборке выполняемые команды уже настроены, поэтому в нем можно ничего не менять, хотя использоваться будут не все пункты созданного меню, а только следующие:

- **Assemble** (транслировать, точнее, ассемблировать);
- **Link** (компоновать);
- **Run** (выполнить);
- **Run w/Debug** (выполнить в подключенном отладчике).

В результате будет получен шаблон консольного приложения Windows. Просмотреть этот шаблон можно, дважды щелкнув левой клавишей мыши по файлу Ex01.asm в окне навигатора Project, расположенном справа вверху.

Шаблон содержит:

- директивы, определяющие набор команд и модель памяти;
- директивы подключения библиотек;

- разделы констант, инициализированных данных с минимально необходимыми директивами определения данных;
- раздел кода, обеспечивающий выход из программы.

Добавим в шаблон описание строки, команду вывода этой строки на экран и команду ожидания нажатия любого символа клавиатуры. Последнее сделано для того, чтобы задержать автоматическое закрытие окна консоли при завершении программы.

```
; Template for console application – комментарий
.686 ; подключение набора команд Pentium
; Подключение библиотеки ввода-вывода
INCLUDE \MASM32\INCLUDE\IO.ASM
.DATA ; раздел инициализации переменных
STR1 db "Hello World",0
.CODE ; начало сегмента кода
START:
; начало кода
outstrln OFFSET STR1
inkey ; ожидание нажатия клавиши
exit ; выход из программы
END START; конец модуля
```

### 1.1. ЗАПУСК ШАБЛОНА ПРИЛОЖЕНИЯ

Для запуска шаблона необходимо выполнить:

- трансляцию **Создать (Make)/Assemble**;
- компоновку **Создать (Make)/Link**;
- запуск на выполнение **Создать (Make)/Run**.

В процессе трансляции (ассемблирования) исходная программа на ассемблере преобразуется в двоичный эквивалент. Если трансляция проходит без ошибок, то в окне Output, которое появляется под окном программы, выводится текст:

```
C:\Masm32\Bin\ML.EXE /c /coff /Cp /nologo
/I"C:\Masm32\Include" "Ex01.asm"
Assembling: Ex01.asm
Make finished.
Total compile time 78 ms
```

*Примечание.* Окно Output появляется и закрывается. Чтобы повторно посмотреть результаты, необходимо установить курсор мыши на верхнюю часть строки состояния под окном текста программы (нижнюю рамку окна Output).

Первая строка сообщения об ассемблировании – вызов ассемблера:

C:\Masm32\Bin\ML.EXE – полное имя файла транслятора ассемблера masm32 (путь + имя), за которым следуют опции:

/c – заказывает ассемблирование без автоматической компоновки;

/coff – определяет формат объектного модуля Microsoft (coff);

/Cr – означает сохранение регистра строчных и прописных букв всех идентификаторов программы;

/nologo – осуществляет подавление вывода сообщений на экран в случае успешного завершения ассемблирования;

/I"C:\Masm32\Include" – определяет местонахождение вставляемых (.inc) файлов;

"Ex01.asm" – имя обрабатываемого файла.

Остальные строки – сообщение о начале и завершении процесса ассемблирования и времени выполнения этого процесса.

Результатом нормального завершения ассемблирования является создание файла, содержащего объектный модуль программы, – файла Ex01.obj.

Если при ассемблировании обнаружены ошибки, то объектный модуль не создается и после сообщения о начале ассемблирования идут сообщения об ошибках, например:

**Ex01.asm(26) : error A2006: undefined symbol : EAX**

В сообщении указывается:

- номер строки исходного текста (в скобках);
- номер ошибки, под которым она описана в документации;
- возможная причина.



После исправления ошибок процесс ассемблирования повторяют.

Следующий этап – компоновка программы. На этом этапе к объектному (двоичному) коду программы добавляются объектные коды используемых подпрограмм. При этом в тех местах программы, где происходит вызов процедур, указывается их относительный адрес в модуле. Сведения о компоновке также выводятся в окно Output:

```
C:\Masm32\Bin\LINK.EXE /SUBSYSTEM:CONSOLE /RELEASE  
/VERSION:4.0 /LIBPATH:"C:\Masm32\Lib"  
/OUT:"Ex01.exe" "Ex01.obj"
```

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

```
Make finished.
```

```
Total compile time 109 ms
```

Первая строка вывода также является командной строкой вызова компоновщика

**C:\Masm32\Bin\LINK.EXE** – полное имя компоновщика, за которым следуют опции:

**/SUBSYSTEM:CONSOLE** – подключить стандартное окно консоли;

**/RELEASE** – создать реализацию (а не отладочный вариант);

**/VERSION:4.0** – минимальная версия компоновщика;

**/LIBPATH:"C:\Masm32\Lib"** – путь к файлам библиотек;

**/OUT:"Ex01.exe"** – имя результата компоновки – загрузочного файла и параметр **"Ex01.obj"** – имя объектного файла.

После устранения ошибки программу необходимо перетранслировать и заново скомпоновать.

Если процессы трансляции и компоновки прошли нормально, то ее можно запустить на выполнение. При этом открывается окно консоли, в которое выводится строка запроса (рис. 3).



Рис. 3. Окно консоли

Окно закрывается при нажатии любого символа.

## 1.2. СОЗДАНИЕ ПРОСТЕЙШЕЙ ПРОГРАММЫ

Для изучения возможностей отладчика необходимо ввести программу, которая выполняет несколько простых команд, например, вычисляет результат следующего выражения:

$$A + 5 - B.$$

Данные для программы зададим константами, поместив их описание в раздел инициированных данных.

```
.DATA
A DWORD 30h
B DWORD 21h
```

Фрагмент кода программы (рис. 4), выполняющей сложение и вычитание, необходимо поместить в сегменте кодов после метки Start.

```
Start:
mov EAX,A ;    поместить число в регистр EAX
add EAX,5h;    сложить EAX и 5, результат в EAX
sub EAX,B ;    вычесть B, результат в EAX
```

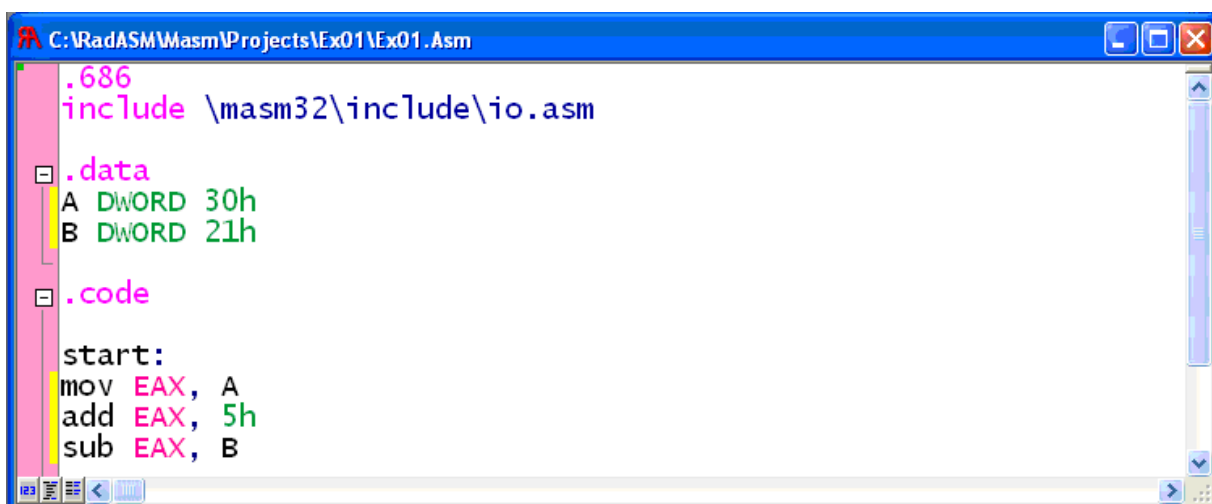


Рис. 4. Фрагмент исходного текста программы  $A + 5 - B$

Поскольку программа ничего не выводит, результат операции следует посмотреть в отладчике.

### 1.3. ПРОСМОТР ВЫПОЛНЕНИЯ ПРОГРАММЫ В ОТЛАДЧИКЕ

Для запуска программы в режиме отладки следует последовательно инициировать следующие пункты меню:

- трансляцию **Создать (Make)/Assemble**;
- компоновку **Создать (Make)/Link**;
- запуск на выполнение в отладчике **Создать (Make)/Run w/Debug**.

После запуска на экране появляется окно отладчика OllyDBG (рис. 5).

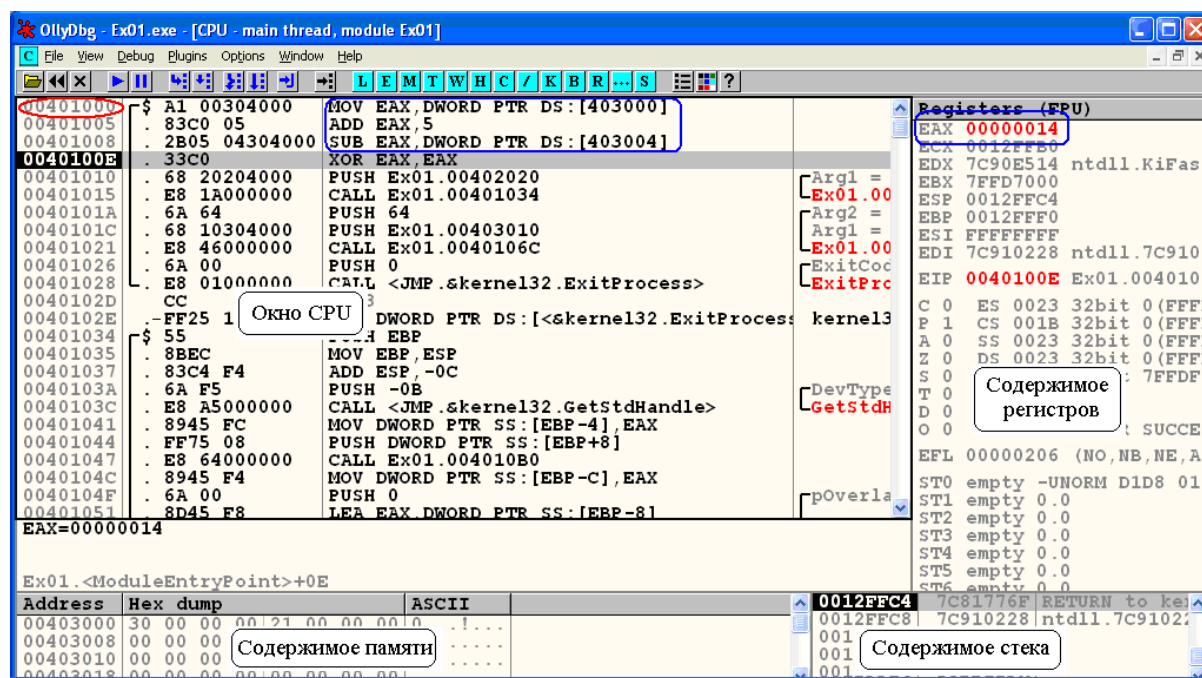


Рис. 5. Окно OllyDBG

В окне можно выделить четыре области:

- дочернее окно **CPU** – окно процессора, в котором высвечиваются адреса команд, их шестнадцатеричные коды, результаты дисассемблирования и результаты выделения параметров процедур;
- область **Registers** – окно содержимого регистров процессора;
- окно данных, в котором высвечиваются адреса памяти (**Address**) и ее шестнадцатеричный (**Hex dump**) и символьный (**ASCII**) дампы;
- окно стека, в котором показывается содержимое вершины стека.

В исходный момент времени курсор находится в окне CPU, т.е. программа готова к выполнению. Для выполнения команды отладчика используют следующие клавиши:

- F7 – выполнить шаг с заходом в тело процедуры;
- F8 – выполнить шаг, не заходя в тело процедуры.

Адрес начала программы **00401000h** выделен на рис. 4 овалом. Сам фрагмент кода содержит три строки в центре.

Адрес начала раздела инициированных данных – **00403000h**. Каждое значение занимает столько байт, сколько резервируется директивой. В отладчике каждый байт представлен двумя шестнадцатеричными цифрами. Кроме того, используется обратный порядок байт, т.е. младший байт числа находится в младших адресах памяти (перед старшим). Если шестнадцатеричная комбинация соответствует коду символа, то он высвечивается в следующем столбце (ASCII), иначе в нем высвечивается точка. Не инициированные данные располагаются после инициированных данных, начиная с адреса, кратного 16.

Для удобства отслеживания команд все числа в примере записаны в шестнадцатеричной системе счисления (например, 30h).

При каждом выполнении команды мы можем наблюдать изменение данных в памяти и/или в регистрах, отслеживая процесс выполнения программы и контролируя правильность промежуточных результатов.

Так, после выполнения первой команды число A копируется в регистр EAX, который при этом подсвечивается красным. При записи в регистр порядок байт меняется на прямой, при котором первым записан старший байт. После выполнения фрагмента в регистре EAX находится ответ выражения 00000014h.

## 2. ОПИСАНИЕ ДАННЫХ

Все данные, используемые в программах на ассемблере, обязательно должны быть объявлены с использованием соответствующих директив, которые определяют тип данных и количество байт, необходимое для размещения этих данных в памяти:

[<Имя>] <Директива> [<Константа> DUP ( [<Список инициализаторов> ] ) ]

где: <Имя> – имя поля данных, которое может не присваиваться;

<Директива> – команда, объявляющая тип описываемых данных (табл. 1);

<Константа> DUP – используется при описании повторяющихся данных, тогда константа определяет количество повторений;

<Список инициализаторов> – последовательность инициализирующих констант через запятую или символ «?», если инициализирующее значение не определяется.

Таблица 1

Директивы определения данных

Директива	Описание типа данных
BYTE / SBYTE	8-разрядное целое без знака / со знаком
WORD / SWORD	16-разрядное целое без знака / со знаком
DWORD / SDWORD	32-разрядное целое без знака / со знаком или ближний указатель
FWORD	48-разрядное целое или дальний указатель
QWORD	64-разрядное целое
TBYTE	80-разрядное целое
REAL4	32-разрядное короткое вещественное
REAL8	64-разрядное длинное вещественное
REAL10	80-разрядное расширенное вещественное

*Примечание.* В качестве директив также могут применяться:

- **DB** – определить байт;
- **DW** – определить слово;
- **DD** – определить двойное слово (4 байта);
- **DQ** – определить четыре слова (8 байт);
- **DT** – определить 10 байт.

В качестве инициализаторов при описании данных применяются:

1) целые константы

[<знак>]<целое>[<основание системы счисления>],

например:

а) –43236, 236d – целые десятичные числа;

б) 23h, 0ADh – целые шестнадцатеричные числа (если шестнадцатеричная константа начинается с буквы, то перед ней указывается 0);

в) 0111010b – целое двоичное;

2) вещественные константы

[<знак>] <целое> . [Ee [<знак>] <целое>],

например: –2., 34E–28;

3) символы в кодировке ASCII (MS DOS) или ANSI (Windows) в апострофах или кавычках, например: 'A' или “A” ;

4) строковые константы в апострофах или кавычках, например, 'ABCD' или “ABCD”.

## 2.1. РЕГИСТРЫ ПРОЦЕССОРОВ СЕМЕЙСТВА IA-32

Регистры данных (32 разряда)

	AH	AL	EAX
	BH	BL	EBX
	CH	CL	ECX
	DH	DL	EDX
	SI		ESI
	DI		EDI
	BP		EBP
	SP		ESP

Селекторы (16 разрядов)

CS
SS
DS
ES
FS
GS

Регистр указатель команд (32)

	IP	EIP
--	----	-----

Слово системных флагов (32)

	FLAGS	EFLAGS
--	-------	--------

Управляющие регистры: CR0..CR3.

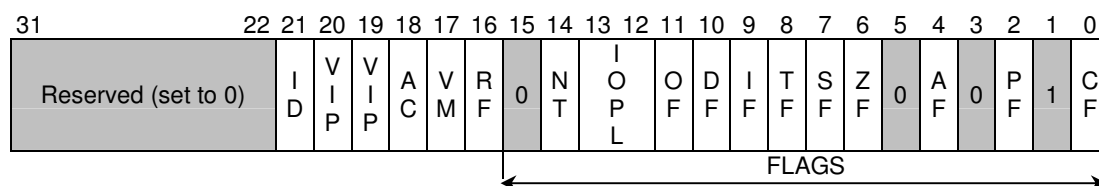
Регистры системных адресов:

- GDTR – регистр адреса таблицы глобальных дескрипторов;
- LDTR – регистр адреса таблицы локальных дескрипторов;
- IDTR – регистр адреса таблицы дескрипторов прерываний;
- TR – регистр состояния задачи.

Отладочные регистры.

Тестовые регистры.

Системные флаги:



Флаги статуса CF, PF, AF, ZF, SF и OF отражают статус выполнения арифметических инструкций (таких как ADD, SUB, MUL, DIV):

- CF – флаг переноса (Carry Flag);
- PF – флаг четности (Parity Flag);
- AF – флаг вспомогательного переноса (Adjust Flag);
- ZF – флаг нуля (Zero Flag);
- SF – флаг знака (Sign Flag);
- OF – флаг переполнения (Overflow Flag);
- DF – флаг направления (Direction Flag).

Системные флаги и поле IOPL влияют на процесс исполнения задачи и поэтому не должны изменяться прикладной программой.

## 2.2. ФОРМАТЫ МАШИННЫХ КОМАНД IA-32

Размер машинной команды процессора IA-32 — от 1 до 15 байт. Структура команды представлена на рис. 6, она содержит:

- **префикс повторения** – используется только для строковых команд;
- **префикс размера адреса** (67h) – применяется для изменения размера смещения: 16 бит при 32-разрядной адресации;
- **префикс размера операнда** (66h) – указывается, если вместо 32-разрядного регистра для хранения операнда используется 16-разрядный;
- префикс замены сегмента – используется при адресации данных любым сегментом, кроме DS;
- d – направление обработки, например, пересылки данных: 1 – в регистр, 0 – из регистра;
- w – размер операнда: 1 – операнды - двойные слова, 0 – операнды-байты;
- mod – режим:
  - 00 – Disp=0 – смещение в команде 0 байт;
  - 01 – Disp=1 – смещение в команде 1 байт;
  - 10 – Disp=2 – смещение в команде 2 байта;
  - 11 – операнды-регистры.



Рис. 6. Структура процессора IA-32



Регистры кодируются в зависимости от размера операнда

	reg (r)	w=1		w=0	
		000	EAX	000	AL
		001	ECX	001	CL
		010	EDX	010	DL
		011	EBX	011	BL
		100	ESP	100	AH
		101	EBP	101	CH
		110	ESI	110	DH
		111	EDI	111	BH

Если в команде используется двухбайтовый регистр (например, AX), то перед командой добавляется префикс изменения длины операнда (66h).

Различают два вида команд, обрабатывающих операнд в памяти:

- команды без байта sib (табл. 2);
- команды, содержащие байт sib (табл. 3).

Различаются эти команды по содержимому поля m (r/m): если  $m \neq 100$ , то байт sib в команде отсутствует и используется табл. 2.

Таблица 2

Схемы адресации памяти в отсутствии байта Sib

Поле r/m	Эффективный адрес второго операнда		
	mod = 00B	mod = 01B	mod = 10B
000B	EAX	EAX+Disp8	EAX+Disp32
001B	ECX	ECX+Disp8	ECX+Disp32
010B	EDX	EDX+Disp8	EDX+Disp32
011B	EBX	EBX+Disp8	EBX+Disp32
<b>100B</b>	<b>Определяется Sib</b>	<b>Определяется Sib</b>	<b>Определяется Sib</b>
101B	Disp32	SS:[EBP+Disp8]	SS:[EBP+Disp32]
110B	ESI	ESI+Disp8	ESI+Disp32
111B	EDI	EDI+Disp8	EDI+Disp32

Таблица 3

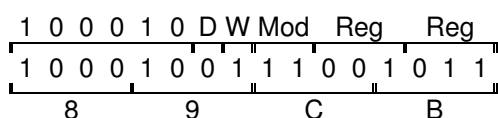
Схемы адресации памяти при наличии байта Sib

Поле base	Эффективный адрес второго операнда		
	mod = 00B	mod = 01B	mod = 10B
000B	EAX+ ss*index	EAX+ ss*index +Disp8	EAX+ ss*index +Disp32
001B	ECX+ ss*index	ECX+ ss*index +Disp8	ECX+ ss*index +Disp32
010B	EDX+ ss*index	EDX+ ss*index +Disp8	EDX+ ss*index +Disp32
011B	EBX+ ss*index	EBX+ ss*index +Disp8	EBX+ ss*index +Disp32
100B	SS:[ESP+ ss*index]	SS:[ESP+ ss*index]+Disp8	SS:[ESP+ ss*index] +Disp32
101B	Disp32+ ss*index	SS:[EBP+ ss*index +Disp8]	SS:[EBP+ ss*index +Disp32]
110B	ESI+ ss*index	ESI+ ss*index +Disp8	ESI+ ss*index +Disp32
111B	EDI+ ss*index	EDI+ ss*index +Disp8	EDI+ ss*index +Disp32

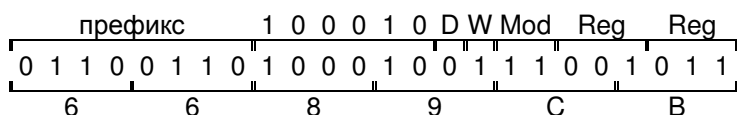
В табл. 3: **ss** – масштаб; **Index** – индексный регистр; **Base** – базовый регистр.

### Примеры

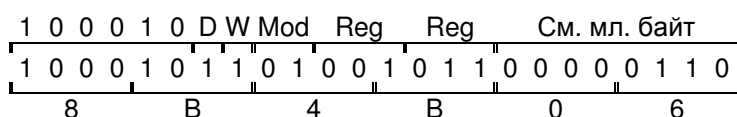
1) `mov EBX, ECX`



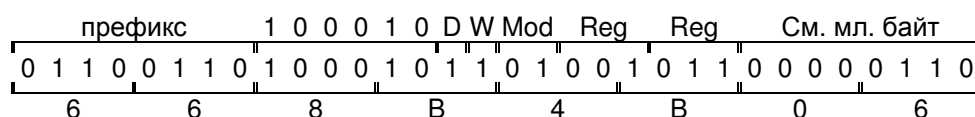
2) `mov BX, CX`



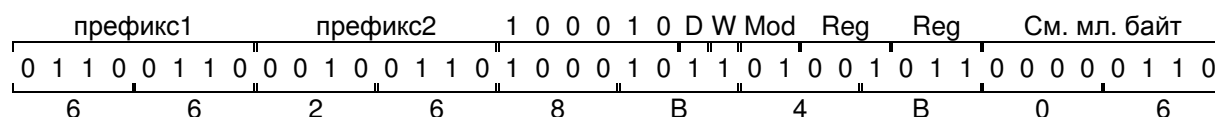
3) `mov ECX, DS:6[EBX]`



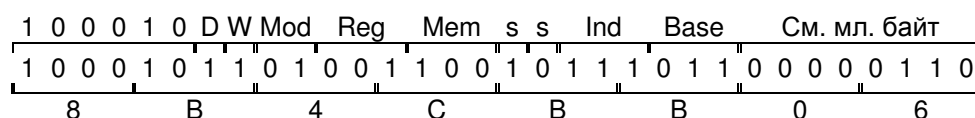
4) `mov CX, DS:6[EBX]`



5) `mov CX, ES:6[EBX]`



6) `mov ECX, 6[EBX+EDI*4]`



## 2.3. КОМАНДЫ ЦЕЛОЧИСЛЕННОЙ АРИФМЕТИКИ IA-32

Процессоры семейства IA-32 поддерживают арифметические операции над однобайтовыми, двухбайтовыми и четырехбайтовыми целыми числами.

Размер операндов при этом определяется:

- объемом регистра, хранящего число – если хотя бы один операнд находится в регистре;
- размером числа, заданным директивой определения данных;
- специальными описателями, например, BYTE PTR (байт), WORD PTR (слово) и DWORD PTR (двойное слово), если *ни один операнд не находится в регистре и размер операнда отличен от размера, определенного директивой определения данных.*

**1. Команда пересылки данных** – пересылает число размером 1, 2 или 4 байта из источника в приемник:

**`mov Приемник, Источник`**

Допустимые варианты

```
mov reg, reg
mov mem, reg
mov reg, mem
mov mem, imm
```

```
mov reg, imm
mov r/m16, sreg
mov sreg, r/m16
```

### ***Примеры***

- 1) `mov AX, BX`
- 2) `mov ESI, 1000`
- 3) `mov 0[DI], AL`
- 4) `mov AX, code`
- 5) `mov DS, AX`

**2. Команда перемещения и дополнения нулями** – при перемещении значение источника помещается в младшие разряды, а в старшие заносятся нули:

**`movzx Приемник, Источник`**

Допустимые варианты

```
movzx r16/r32, r/m8
movzx r32, r/m16
```

### ***Примеры***

- 1) `movzx EAX, BX`
- 2) `movzx SI, AH`

**3. Команда перемещения и дополнения знаковым разрядом** – команда выполняется аналогично, но в старшие разряды заносятся знаковые биты:

**`movsx Приемник, Источник`**

### ***4. Команда обмена данных.***

**`xchg Операнд1, Операнд2`**

Допустимые варианты

```
xchg reg, reg
xchg mem, reg
xchg reg, mem
```

**5. Команды записи слова или двойного слова в стек и извлечения из стека.**

**push imm16 / imm32 / r16 / r32 / m16 / m32**

**pop r16 / r32 / m16 / m32**

Если в стек помещается 16-разрядное значение, то значение  $ESP := ESP - 2$ , если помещается 32-разрядное значение, то  $ESP := ESP - 4$ .

Если из стека извлекается 16-разрядное значение, то значение  $ESP := ESP + 2$ , если помещается 32-разрядное значение, то  $ESP := ESP + 4$ .

### ***Примеры***

push SI

pop word ptr [EBX]

**6. Команды сложения** – складывает операнды, а результат помещает по адресу первого операнда. В отличие от ADD команда ADC добавляет к результату значение бита флага переноса CF.

**add Операнд1, Операнд2**

**adc Операнд1, Операнд2**

Допустимые варианты:

add reg, reg

add mem, reg

add reg, mem

add mem, imm

add reg, imm

**7. Команды вычитания** – вычитает из первого операнда второй и результат помещает по адресу первого операнда. В отличие от SUB команда SBB вычитает из результата значение бита флага переноса CF. Допустимые варианты те же, что и у сложения.

**sub Операнд1, Операнд2**

**sbb Операнд1, Операнд 2**

## **8. Команды добавления/вычитания единицы.**

**inc reg/mem**

**dec reg/mem**

### **Примеры**

**inc AX**

**dec byte ptr 8[EBX,EDI]**

## **9. Команды умножения.**

**mul <Операнд2>**

**imul <Операнд2>**

Допустимые варианты:

**mul/imul r|m8 ; AX= AL\*<Операнд2>**

**mul/imul r|m16 ; DX:AX= AX\*<Операнд2>**

**mul/imul r|m32 ; EDX:EAX= EAX\*<Операнд2>**

**Важно!** В качестве второго операнда нельзя указать непосредственное значение.

Регистры первого операнда в команде не указываются. Местонахождение и длина результата операции зависит от размера второго операнда.

### **Примеры**

**mov AX, 4**

**imul word ptr A ; DX:AX:=AX\*A**

**10. Команды «развертывания» чисел** – операнды в команде не указываются.

Операнд и его длина определяются кодом команды и не могут быть изменены. При выполнении команды происходит расширение записи числа до размера результата посредством размножения знакового разряда.

Команды часто используются при программировании деления чисел одинаковой размерности для обеспечения удвоенной длины делимого

**cbw ; байт в слово AL -> AX**

**cwd ; слово в двойное слово AX -> DX:AX**

**cdq ; двойное слово в учетверенное EAX -> EDX:EAX**

**cwde ; слово в двойное слово AX -> EAX**

## 11. Команды деления.

**div** <Операнд>

**idiv** <Операнд>

Допустимые варианты:

```
div/idiv r|m8      ; AL := AX div <Операнд>,
                   ; AH := AX mod <Операнд>
div/idiv r|m16     ; AX := (DX:AX) div <Операнд>,
                   ; DX := (DX:AX) mod <Операнд>
div/idiv r|m32     ; EAX := (EDX:EAX) div <Операнд>,
                   ; EDX := (EDX:EAX) mod <Операнд>
```

**Важно!** В качестве операнда нельзя указать непосредственное значение.

### Примеры

```
mov AX, 40
cld
idiv word ptr A; AX:=(DX:AX):A
```

## 3. ОРГАНИЗАЦИЯ ВВОДА-ВЫВОДА

Для лабораторных занятий разработана библиотека io.asm. Библиотека содержит специальные подпрограммы ввода-вывода консольного режима.

1. Процедура ввода числа.

**Inint x:DWORD**

Операнд – адрес буфера ввода.

2. Процедура вывода числа.

**Outint x:DWORD**

Операнд – адрес буфера вывода.

3. Процедура вывода символа.

**Outch x:BYTE**

Операнд – адрес буфера вывода.

4. Процедура вывода строки.

**Outstr x:BYTE**

Операнд – адрес буфера вывода.

5. Процедура перевода курсора на новую строку.

**Newline**

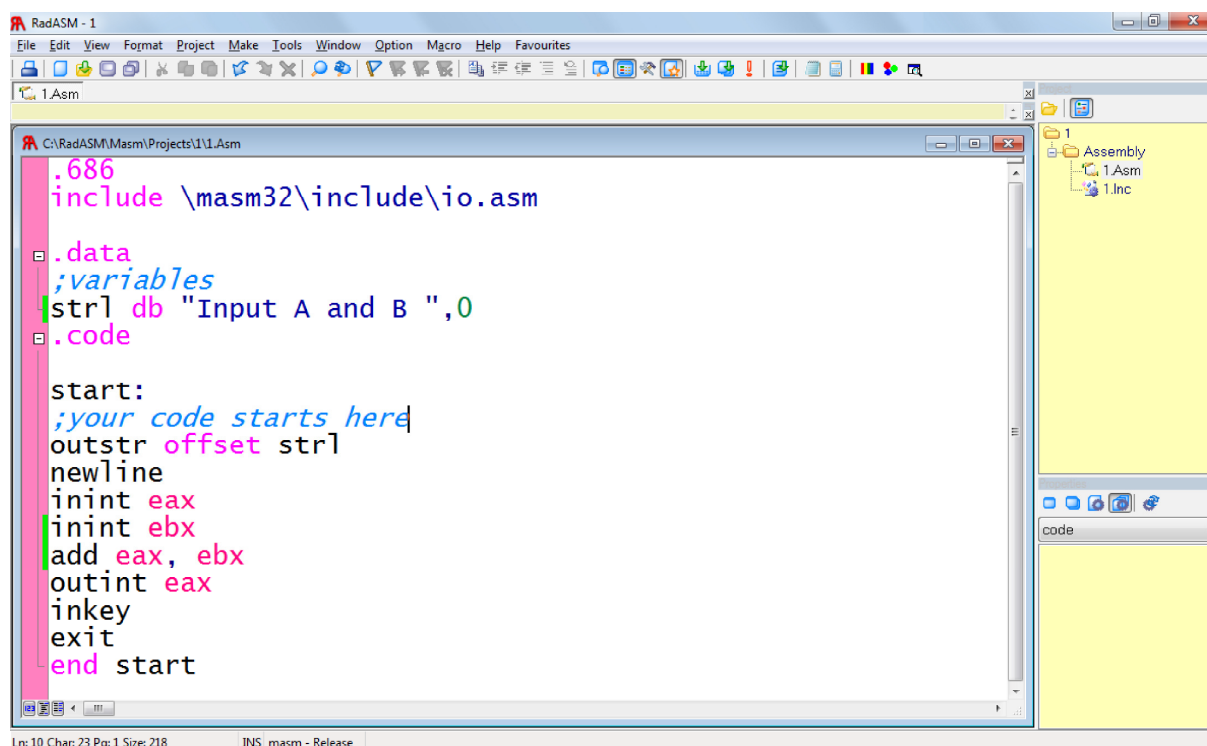
6. Процедура ожидания нажатия клавиши.

**InKey**

## 4. ПРИМЕРЫ ПРОГРАММ

### 4.1. ПРИМЕР ЛИНЕЙНОЙ ПРОГРАММЫ

Рассмотрим пример программы, которая вводит с клавиатуры два числа и выводит на экран их сумму. Исходный текст приведен на рис. 7.



```
.686
include \masm32\include\io.asm

.data
;variables
str1 db "Input A and B ",0
.code

start:
;your code starts here
outstr offset str1
newline
inint eax
inint ebx
add eax, ebx
outint eax
inkey
exit
end start
```

Рис. 7. Исходный текст программы A + B

Результат работы приложения представлен на рис. 8.

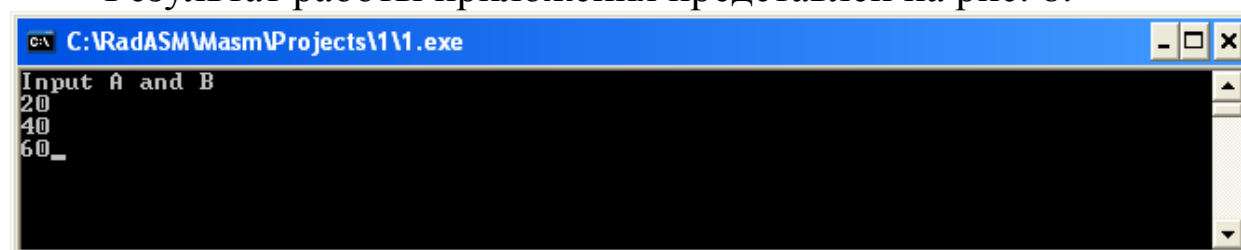


Рис. 8. Результат программы A + B



При программировании на ассемблере используется модель памяти Flat. В этой модели считается, что все сегменты программы (кодový, сегменты данных и стека) начинаются с нулевого адреса и имеют размер, равный доступной памяти компьютера. Таким образом, они как бы накладываются на общее пространство адресов. Реальное разделение адресного пространства между данными каждого сегмента осуществляется посредством размещения программы, данных и стека с различными смещениями относительно начала сегментов.

Та же модель используется при компиляции программ с языков высокого уровня в основных программных средах (например, Turbo Delphi и Visual C++), поскольку она существенно упрощает адресацию программы.

## 4.2. ОБРАБОТКА ЦЕЛЫХ ЧИСЕЛ

Рассмотрим примеры решения задач.

*Задача 1.* Найти наибольшее из двух целых чисел.

```
.686
include /masm32/include/io.asm

.data
MsgInput db "Введите 2 целых числа > ",0
MsgOutput db "Наибольшее число: ",0

.code
start:
    outstr    offset MsgInput;вывод сообщения
    inint     EAX              ;ввод первого числа
    inint     EBX              ;ввод второго числа

    cmp EAX, EBX              ;сравнение чисел
    jg V              ;переход, если
                        ;первое число больше
    mov EAX, EBX              ;иначе EAX=EBX

V:
    outstr    offset MsgOutput;вывод сообщения
    outint    EAX              ;вывод результата
exit
end start
```

### *Задача 2. Найти сумму цифр натурального числа.*

```
.686
include /masm32/include/io.asm

.data
MsgInput db "Введите положительное число > ",0
MsgOutput db "Сумма цифр равна ",0
ten dw 10

.code
start:
    outstr    offset MsgInput    ; вывод сообщения
    inint     EAX                ; ввод числа

    mov EBX, 0                  ; EBX=0
L:
    mov EDX, 0                  ; делим EDX:EAX
    div ten                      ; на 10
    add EBX, EDX                ; прибавляем остаток
                                ; (последнюю цифру)
    cmp EAX, 0                  ; если число не
                                ; равно нулю, то
    jne L                      ; возврат к метке L

    outstr    offset MsgOutput; вывод сообщения
    outint    EBX                ; вывод результата

exit
end start
```

### **Задачи для самостоятельного решения**

1. Дано натуральное число. Найти количество его цифр.
2. Дано натуральное число. Найти сумму его делителей.
3. Даны три целых числа. Найти наименьшее из них.
4. Даны два натуральных числа. Найти число с наибольшей суммой цифр.
5. Даны два натуральных числа. Проверить верно ли, что сумма цифр первого числа в два раза меньше произведения цифр второго числа.

### 4.3. ОБРАБОТКА ПОСЛЕДОВАТЕЛЬНОСТЕЙ ЧИСЕЛ

*Задача.* Дано натуральное число  $N$ . Дана последовательность, состоящая из  $N$  чисел. Найти количество чисел, кратных трем.

```
.686
include /masm32/include/io.asm

.data
MsgInput db "Введите количество элементов
           последовательности > ",0
MsgInput2 db "Введите элементы > ",0
MsgOutput db "Количество : ",0
three     dw 3
a         dd ?
N         dd ?
count     dd ?

.code
start:
    outstr    offset MsgInput;вывод сообщения
    inint     N              ;ввод количества (N)

    outstr    offset MsgInput2;вывод сообщения
    mov ECX, N              ;число повторений цикла
    mov count, 0            ;обнуляем счетчик
L:
    inint a                  ;ввод элемента
    mov EDX, 0
    mov EAX, a               ;деление элемента
    div three                ;на 3
    cmp EDX, 0               ;если не делится нацело,
    jne LL                  ;то переход в конец цикла
    inc count                ;иначе увеличиваем количество

    LL: Loop L               ;следующая итерация цикла
                                ;(возврат к метке L)

    outstr    offset MsgOutput;вывод сообщения
    outint    count          ;вывод результата
exit
end start
```

## Задачи для самостоятельного решения

1. Дано натуральное число  $N$ . Дана последовательность, состоящая из  $N$  чисел. Найти сумму четных элементов.
2. Дано натуральное число  $N$ . Дана последовательность, состоящая из  $N$  чисел. Найти количество элементов, кратных пяти.
3. Дано натуральное число  $N$ . Дана последовательность, состоящая из  $N$  чисел. Найти произведение модулей отрицательных элементов.
4. Дано натуральное число  $N$ . Дана последовательность, состоящая из  $N$  чисел. Найти сумму квадратов положительных элементов.
5. Дано натуральное число  $N$ . Дана последовательность, состоящая из  $N$  чисел. Найти произведение порядковых номеров элементов, значение которых не кратно семи.

### 4.4. ОБРАБОТКА ОДНОМЕРНЫХ МАССИВОВ

*Задача 1.* Дано натуральное число  $N$ . Дан одномерный массив. Найти сумму отрицательных элементов, расположенных на четных позициях.

```
.686
include /masm32/include/io.asm

.data
MsgInput db "Введите размерность массива > ",0
MsgInput2 db "Введите элементы массива > ",0
MsgOutput db "Результат: ",0
two dw 2
a dd 100 dup (?)
N dd ?
p dd ?
count dd ?

.code
start:
    outstr    offset MsgInput
    inint     N
    outstr    offset MsgInput2
```

```

    mov ECX, N      ; помещаем в регистр ECX
                    ; число повторений цикла
    mov EDI, 0      ; адрес первого элемента массива

L:
    inint a[EDI]    ; ввод элемента массива
    add EDI, 4      ; переход на следующий элемент
    Loop L

    mov ECX, N
    mov EDI, 0      ; адрес первого элемента
    mov p, 1        ; позиция первого элемента
    mov EBX, 0      ; начальное значение суммы

L2:
    cmp a[EDI], 0   ; проверка элемента на
                    ; отрицательность
    jg LL2          ; если неотрицательный, то
                    ; переход в конец цикла
    mov EAX, p      ; проверка позиции элемента
    mov EDX, 0      ; на четность
    div two
    cmp EDX, 0      ; если нечетная позиция, то
    jne LL2         ; переход в конец цикла

    add EBX, a[EDI] ; если выполнены условия, то
                    ; вычисляем сумму
LL2: add EDI, 4      ; переход на следующий элемент
    inc p
    Loop L2

    outstr    offset MsgOutput
    outint    EBX
newline

inkey        ; ожидание нажатия клавиши
exit
end start

```

*Задача 2.* Дан одномерный массив размерности *N*. Проверить является ли массив упорядоченным по возрастанию.

.686

include /masm32/include/io.asm

.data

MsgInput db "Введите размерность массива > ",0

MsgInput2 db "Введите элементы массива > ",0

MsgOutput db "Результат: ",0

a dd 100 dup (?)

N dd ?

flag dd ?

.code

start:

outstr offset MsgInput

inint N

outstr offset MsgInput2

mov ECX, N

mov EDI, 0

L: inint a[EDI] ;ввод массива

add EDI, 4

Loop L

mov ECX, N ;помещаем в ECX число N-1, чтобы

dec ECX ;не выйти за границы массива

mov EDI, 0

mov flag,1 ;flag=1 (массив упорядочен)

L2: mov EAX, a[EDI] ;сравниваем два соседних

cmp EAX, a[EDI+4] ;элемента

jle LL2 ;если первый меньше или

;равен, то переход в

;конец цикла

mov flag,0 ;иначе flag=0

; (массив неупорядочен)

LL2: add EDI, 4

Loop L2

```

        outstr      offset MsgOutput
        outint      flag
newline
inkey          ; ожидание нажатия клавиши
exit
end start

```

### Задачи для самостоятельного решения

1. Дано натуральное число  $N$ . Дан одномерный массив. Найти максимальный элемент массива.
2. Дано натуральное число  $N$ . Дан одномерный массив. Проверить, является ли он знакопеременным.
3. Дано натуральное число  $N$ . Дан одномерный массив. Найти произведение положительных элементов, расположенных на позициях, не кратных пяти.
4. Дано натуральное число  $N$ . Дан одномерный массив. Найти целую часть среднего арифметического кратных трем элементов.
5. Дано натуральное число  $N$ . Дан одномерный массив. Заменить отрицательные элементы значением их порядковых номеров.

### 4.5. ОБРАБОТКА ДВУМЕРНЫХ МАССИВОВ (МАТРИЦ)

*Задача 1.* Дано натуральное число  $N$ . Дана матрица размером  $N \times N$ . Найти сумму положительных элементов, расположенных выше главной диагонали.

```

.686
include /masm32/include/io.asm
.data
MsgInput db "Введите размерность матрицы > ",0
MsgInput2 db "Введите элементы матрицы > ",0
MsgOutput db "Результат: ",0
a dd 100 dup (?)
N dd ?
step dd ?
sum dd ?
.code
start:
        outstr      offset MsgInput
        inint       N

```

```

    outstr      offset MsgInput

    mov EAX, N      ; помещаем в ECX значение nхn
    mul N          ; (количество элементов матрицы)
    mov ECX, EAX
    mov EBX, 0      ; адрес первого элемента матрицы
L:
    inint a[EBX]    ; ввод матрицы
    add EBX, 4
    Loop L

    mov EAX, N      ; вычисление шага
    mov EDX, 0      ; step=4n+4
    mul four        ; для обработки элементов,
    add EAX, 4       ; расположенных выше
    mov step, EAX    ; главной диагонали

    mov ECX, N      ; в ECX – количество строк
    mov EBX, 0      ; адрес первого элемента
    mov sum, 0       ; начальное значение суммы

L2:  push ECX        ; сохраняем значение ECX в стеке
     mov EDI, 0      ; адрес первого элемента текущей
                     ; строки

    LL:  cmp a[EBX+EDI], 0 ; если элемент
                                     ; неположительный, то
        jle LL2        ; переход в конец цикла
        mov EAX, a[EBX+EDI] ; иначе вычисляем сумму
        add sum, EAX

    LL2: add EDI, 4    ; переход к следующему
                     ; элементу текущей строки
        Loop LL

    pop ECX          ; восстанавливаем из стека
                     ; значение ECX
    add EBX, step     ; устанавливаем EBX на адрес
                     ; первого элемента следующей
                     ; строки

    Loop L2

```



```

        outstr      offset MsgOutput
        outint      sum
newline

inkey          ; ожидание нажатия клавиши
exit
end start

```

**Задача 2.** Дано натуральное число  $N$ . Дана матрица размера  $N \times N$ . Найти максимальный элемент, среди элементов, расположенных на главной диагонали.

```

.686
include /masm32/include/io.asm

.data
MsgInput db      "Введите размерность матрицы > ",0
MsgInput2 db     "Введите элементы матрицы > ",0
MsgOutput db     "Результат: ",0
a dd 100 dup (?)
N dd ?
four dd 4
step dd ?
max dd ?

.code
start:

        outstr      offset MsgInput
        inint       N
        outstr      offset MsgInput

        mov ECX, N      ; помещаем в ECX значение nхn
        mul N          ; (количество элементов матрицы)
        mov ECX, EAX
        mov EBX, 0      ; адрес первого элемента матрицы

L:      inint a[EBX]     ; ввод матрицы
        add EBX, 4
        Loop L

        mov EAX, N      ; вычисление шага
        mov EDX, 0      ; step=4n+4

```

```

mul four                ;для обработки элементов,
add EAX, 4              ;расположенных на
mov step, EAX           ;главной диагонали

mov ECX, N              ;в ECX – количество строк
mov EBX, 0              ;адрес первого элемента
mov EAX, a[EBX]         ;переменная max получает
mov max, EAX            ;значение первого элемента

L2: mov EAX, a[EBX]      ;сравнение текущего элемента
    cmp EAX, max         ;матрицы с переменной max
    jle LL2             ;если меньше или равен, то
                        ;переход в конец цикла
    mov max, EAX        ;иначе max получает новое
                        ;значение
LL2: add EBX, step       ;переход на следующий
                        ;элемент главной диагонали
    Loop L2

    outstr      offset MsgOutput
    outint     max
newline

inkey          ; ожидание нажатия клавиши
exit
end start

```

### Задачи для самостоятельного решения

1. Дано натуральное число  $N$ . Дана матрица размером  $N \times N$ . Заменить нулем элементы, расположенные ниже главной диагонали.
2. Дано натуральное число  $N$ . Дана матрица размером  $N \times N$ . Найти максимальный элемент среди элементов, не принадлежащих главной диагонали.
3. Дано натуральное число  $N$ . Дана матрица размером  $N \times N$ . Найти сумму нечетных элементов, расположенных выше побочной диагонали.

4. Дано натуральное число  $N$ . Дана матрица размером  $N \times N$ . Найти номера строк матрицы, содержащих только положительные элементы.

5. Дано натуральное число  $N$ . Дана матрица размером  $N \times N$ . Найти минимальный элемент среди элементов, расположенных на побочной диагонали.

## **РЕКОМЕНДУЕМАЯ ЛИТЕРАТУРА**

1. Кольцов Ю.В., Гаркуша О.В., Добровольская Н.Ю. Программирование на языке ассемблера: учеб. пособие. Краснодар: Кубанский гос. ун-т, 2011.

2. Ирвин К. Язык ассемблера для процессоров Intel. М.: «Вильямс», 2005.

3. Пильщиков В.Н. Assembler. Программирование на языке ассемблера IBM PC. М.:Диалог-МИФИ, 2005.

4. Юров В.И. Assembler. Практикум. 2-изд. СПб.: Питер, 2006.

## ПРИЛОЖЕНИЕ

### ОСНОВНЫЕ НАСТРОЙКИ СРЕДЫ RADASM

#### НАСТРОЙКА ПУТЕЙ

Для того чтобы настроить пути автовызова среды, используется пункт меню **Настройки/Установить пути** (рис. П1).

При этом:

\$A – путь к каталогу, содержащему Masm32;

\$R – путь к каталогу установки RADAsm;

\$E – путь к каталогу, содержащему папку отладчика OlleDBG;

\$B – путь к каталогу обрабатывающих программ Masm32;

\$H – путь к каталогу, содержащему файлы справок;

\$I – путь к каталогу подставляемых файлов Masm32 (.inc);

\$L – путь к каталогу автовызова двоичных образов стандартных подпрограмм (функций API и т.п.);

\$M – путь к каталогу, содержащему файлы макросов;

\$P – путь к каталогу, в котором по умолчанию создаются проекты;

\$T – путь к каталогу шаблонов новых проектов.

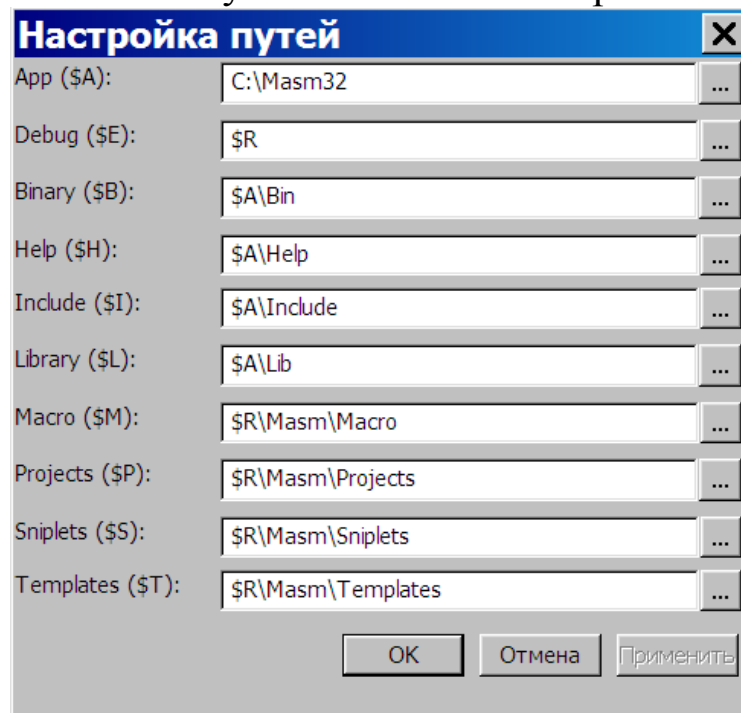


Рис. П1. Настройки путей

Если к проекту с помощью INCLUDE подключаются другие файлы, не заданные явно в проекте, то путь к ним указывается полностью.

## ФОРМИРОВАНИЕ КОМАНДНЫХ СТРОК ДЛЯ ВЫЗОВА АСЕМБЛЕРА, КОМПОНОВЩИКА ИЛИ ОТЛАДЧИКА

Для формирования командных строк используется специальный формат, который затем в среде распознается и транслируется в команду. Настройки выполняются при создании проекта (на последнем шаге) или при вызове пункта меню **Проект/Настройки проекта** (рис. П2).

Помеченные команды доступны для вызова.

**Compile RC** – компилятор ресурсов (окон, кнопок и т.п.).

**Assemble** – ассемблирование.

**Link** – компоновка.

**Build** – сборка.

**Go** – компиляция ресурсов, ассемблирование, сборка и запуск.

**Run** – выполнение.

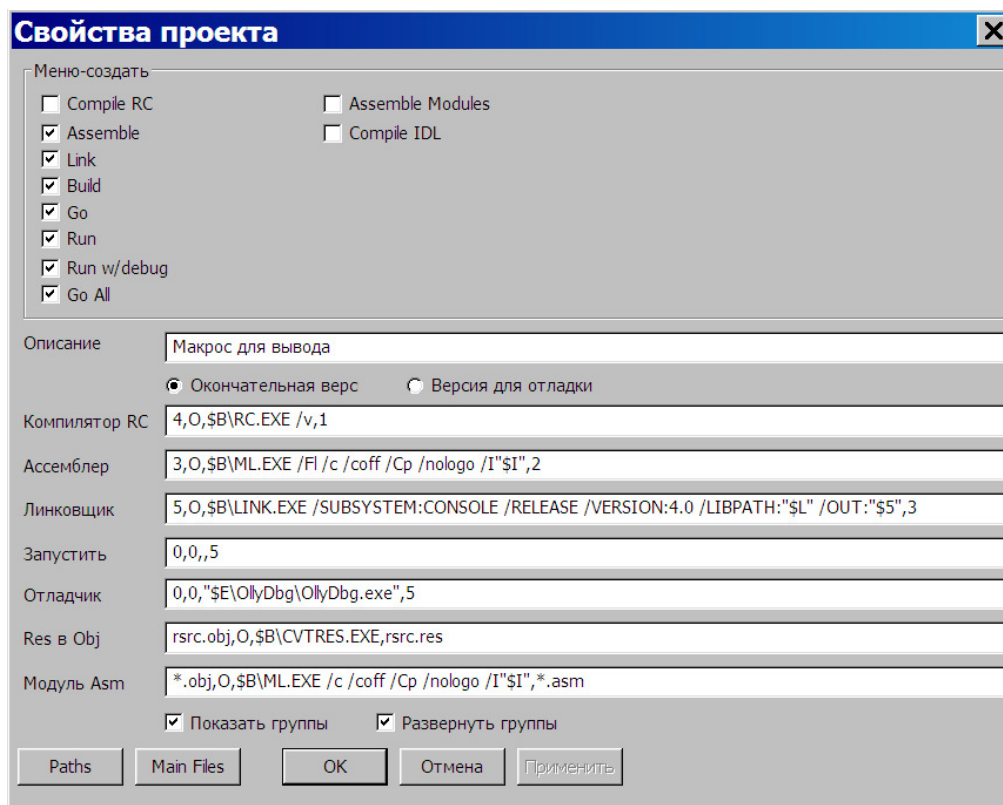


Рис. П2. Настройки командных строк

**Rub w/Debug** – выполнение в отладчике.

**Go all** – компиляция ресурсов, ассемблирование всех модулей, сборка и запуск.

**Assemble modules** – ассемблирование модулей, оформленных в отдельных файлах.

Для записи команды используется специальный формат:

**<Номер или имя создаваемого или пересоздаваемого файла>, <Указание окна вывода результата>, <Имя вызываемой программы с опциями>, <Номер или имя исходного файла>**

В соответствии с принятой автором среды системой все типы файлов, используемых в проекте, имеют номера, например:

0=.rap – файл проекта RADAsm;

1=.rc – файл ресурсов;

2=.asm – исходная программа на ассемблере;

3=.obj – объектный модуль (результат ассемблирования);

4=.res – результат компиляции файла ресурсов;

5=.exe – исполняемый файл;

7=.dll – файл динамически загружаемой библиотеки;

8=.txt – текстовый файл;

9=.lib – файл библиотеки;

10=.mak – командный файл;

11=.hla – файл ассемблера высокого уровня;

12=.com – исполняемый com файл.

Второй параметр – указание окна вывода результатов:

**O** – сообщения об ошибках выводятся в окно Output RADAsm (если указано OT, то туда же выводится командная строка);

**C** – сообщения об ошибках выводятся в окно Console.

Далее идет имя обрабатываемой программы с опциями и имя исходного модуля.

Более подробно правила записи командных строк приведены в справке среды RADAsm.

## ОГЛАВЛЕНИЕ

ПРЕДИСЛОВИЕ .....	3
1. НАЧАЛО РАБОТЫ СО СРЕДОЙ .....	4
1.1. Запуск шаблона приложения .....	6
1.2. Создание простейшей программы .....	9
1.3. Просмотр выполнения программы в отладчике .....	10
2. ОПИСАНИЕ ДАННЫХ .....	11
2.1. Регистры процессоров семейства IA-32 .....	13
2.2. Форматы машинных команд IA-32 .....	15
2.3. Команды целочисленной арифметики IA-32 .....	18
3. ОРГАНИЗАЦИЯ ВВОДА-ВЫВОДА .....	22
4. ПРИМЕРЫ ПРОГРАММ .....	23
4.1. Пример линейной программы .....	23
4.2. Обработка целых чисел .....	24
4.3. Обработка последовательностей чисел .....	26
4.4. Обработка одномерных массивов .....	27
4.5. Обработка двумерных массивов (матриц) .....	30
РЕКОМЕНДУЕМАЯ ЛИТЕРАТУРА .....	34
ПРИЛОЖЕНИЕ .....	35

*Учебное издание*

К о л ь ц о в Ю р и й В л а д и м и р о в и ч  
Г а р к у ш а О л е г В а с и л ь е в и ч  
Д о б р о в о л ь с к а я Н а т а л ья Ю р ь е в н а  
Х а р ч е н к о А н н а В л а д и м и р о в н а

ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ  
АССЕМБЛЕРА IA-32 В СРЕДЕ RADAsm

Учебное пособие

---

Подписано в печать 27.11.2014. Формат 60×84 1/16.  
Печать цифровая. Уч.-изд. л. 2,8. Тираж 100 экз. Заказ №

Кубанский государственный университет.  
350040, г. Краснодар, ул. Ставропольская, 149.

Издательско-полиграфический центр КубГУ  
350040, г. Краснодар, ул. Ставропольская, 149.