Научись говорить с компьютером на языке процессора

АССЕМБЛЕР? ЭТО ПРОСТО!

УЧИМСЯ ПРОГРАММИРОВАТЬ

ОЛЕГ КАЛАШНИКОВ

+CD

bhv

Олег Калашников

Ассемблер! Это просто. Учимся программировать

Санкт-Петербург
«БХВ-Петербург»
2005

УДК 681.3.068+800.92Ассемблер
ББК 32.973.26-018.1
К17

Калашников О. А.

К17 Ассемблер? Это просто! Учимся программировать. — СПб.: БХВ-Петербург, 2005. — 384 с.: ил.

ISBN 5-94157-709-5

Подробно и доходчиво объясняются все основные вопросы программирования на ассемблере. Рассмотрены команды процессоров Intel, 16- и 32-разрядные регистры, основы работы с сопроцессором, сегментация памяти в реальном масштабе времени, управление клавиатурой и последовательным портом, работа с дисками и многое другое. Описано, как разработать безобидный нерезидентный вирус и антивирус против этого вируса, как написать файловую оболочку (типа Norton Commander или FAR Manager) и как писать резидентные программы. Каждая глава состоит из объяснения новой темы, описания алгоритмов программ, многочисленных примеров и ответов на часто задаваемые вопросы. Компакт-диск содержит исходные коды всех примеров, приведенных в книге, с подробными описаниями.

Для программистов

УДК 681.3.068+800.92Ассемблер
ББК 32.973.26-018.1

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Игорь Шишигин*
Зав. редакцией	*Григорий Добин*
Редактор	*Алия Амирова*
Компьютерная верстка	*Татьяны Олоновой*
Корректор	*Зинаида Дмитриева*
Дизайн обложки	*Игоря Цырульникова*
Зав. производством	*Николай Тверских*

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 24.06.05.
Формат 70×100$^{1}/_{16}$. Печать офсетная. Усл. печ. л. 30,96.
Тираж 3000 экз. Заказ №
"БХВ-Петербург", 194354, Санкт-Петербург, ул. Есенина, 5Б.

Санитарно-эпидемиологическое заключение на продукцию
№ 77.99.02.953.Д.006421.11.04 от 11.11.2004 г. выдано Федеральной службой
по надзору в сфере защиты прав потребителей и благополучия человека.

Отпечатано с готовых диапозитивов
в ГУП "Типография "Наука"
199034, Санкт-Петербург, 9 линия, 12

ISBN 5-94157-709-5

Оглавление

Предисловие

Итак, вы решили начать изучение языка ассемблера. Возможно, вы уже пробовали его изучать, но так и не смогли освоить до конца, поскольку он показался вам очень трудным. Обилие новых, неизвестных читателю терминов и сложность языка, которым написаны многие книги, делают их трудными для начинающих программистов. В настоящей книге автор старался излагать материал так, чтобы он был понятен любому пользователю: и начинающему программисту, и человеку, который ни разу не сталкивался ни с каким языком программирования.

Основой для книги послужили материалы разработанной автором рассылки "Ассемблер? Это просто! Учимся программировать". Используя данную рассылку, *более 18 000* подписчиков научились писать такие программы на ассемблере, которые казались им раньше чрезвычайно сложными и недоступными для понимания или написания. Большая часть подписчиков пыталась раньше изучать язык ассемблера, но так и не смогла пройти полный курс (прочитать ту или иную книгу до конца). Материал рассылки помог им понять ассемблер и научил писать довольно-таки сложные программы под операционными системами MS-DOS и Windows.

Автор постарался учесть все недоработки и ошибки, допущенные в рассылке, а также добавил много нового материала, который поможет вам изучить ассемблер за короткое время. Более того, автор попытался сделать обучение как можно более интересным для вас, перейдя с первой же главы к практической части.

Автор не претендует на то, что материал, изложенный в данной книге, поможет вам освоить ассемблер во всех его проявлениях и покажет все возможности языка. Ассемблер настолько многогранен, что просто невозможно подробно описать все его операторы, команды, алгоритмы, области применения в одной книге. Тем не менее, прочитав книгу до конца, вы сможете научиться писать собственные программы, разбирать чужие, а также поймете, как в целом работает компьютер.

Уникальность настоящей книги заключается в следующем:

- каждая глава соответствует одному занятию, в конце главы приводится файл для практического изучения;

- материал изложен на простом языке, все новые термины подробно объясняются;
- в процессе изучения ассемблера, начиная с *главы 11*, рассматриваются четыре программы:
  - безобидный нерезидентный вирус;
  - резидентный антивирус против написанного нами вируса;
  - файловая оболочка для DOS (типа Norton Commander®, FAR Manager®, DOS Navigator® и т. п.) с поддержкой длинных имен файлов и использованием XMS-памяти;
  - несколько видов резидентных программ (программ, которые постоянно находятся в памяти).

Также исследуется работа отладчиков и способы обойти отладку программы.

В ассемблере, как и в любом другом языке программирования, очень важна практика и опыт. На компакт-диске, прилагаемом к настоящей книге, приводятся готовые ассемблерные файлы в формате DOS с подробными описаниями для практического изучения курса. Несомненно, они облегчат понимание самого языка.

Несколько советов

- Обязательно скачайте файлы-приложения для практического изучения курса с сайта **http://www.Kalashnikoff.ru** (если таковых нет). Без практики данная книга вряд ли поможет вам научиться программировать на ассемблере.
- Чаще пользуйтесь отладчиком.
- Изменяйте код программ (файлов-приложений), больше экспериментируйте.
- Пробуйте написать свою собственную программу на основе изученного материала.
- Так как вначале будет довольно сложно ориентироваться в обилии инструкций, директив, прерываний, процедур ассемблера, то пробуйте вставлять в ваши собственные программы выдержки, процедуры, алгоритмы из файлов-приложений. Помните, что опыт приходит со временем!
- Внимательно следите за ходом мысли автора, за его логикой. Это особенно актуально при чтении второй и третей части.
- *Не спешите!* Внимательно и досконально изучайте каждую главу, выполняйте все, что автор просит сделать с прилагаемыми программами (запускать их под отладчиком, изменять код, думать над тем, что делает та или иная процедура и пр.).

Ответы на некоторые вопросы

1. Под управлением каких операционных систем будут работать файлы-приложения?

Компания Microsoft придерживается политики поддержки работоспособности программ, написанных в более ранних версиях собственных операционных систем. Если программа разработана, например, для MS-DOS 3.30, то она будет выполняться и в более поздних версиях этой системы, если, конечно, в самой программе не установлены ограничения, или она не привязана в работе именно к той версии, для которой написана. Появление на рынке ОС Windows 95/98/ME продолжило эту традицию, оставив возможность загружать и выполнять программы, написанные под операционную систему MS-DOS, и даже запуск DOS-приложений, требующих загруженной "чистой" дисковой системы, к которым относятся, как правило, сложные графические игры, работающие с расширителями, например, DOS4GW. Тем не менее большинство программ прекрасно запускаются напрямую из Проводника (Windows Explorer), не требуя перезагрузки системы. Например, Norton Commander, DOS Navigator, Far Manager, а также необходимые для изучения настоящего курса средства разработки и отладки, перечисленные ниже.

Все примеры протестированы на работоспособность под управлением следующих операционных систем компании Microsoft на IBM-совместимых компьютерах:

- MS-DOS 5.00 и выше;
- Windows 95;
- Windows 98;
- Windows ME;
- Windows 2000 Pro и Server;
- Windows XP Home Edition и Pro.

Стоит также отметить, что программы, использующие в своей работе расширенную память (XMS), требуют наличия драйвера himem.sys. Если такая программа запускается в Windows, то нет необходимости заботиться о предварительной загрузке этого драйвера, т. к. по умолчанию он всегда загружается при старте системы.

2. Какое программное обеспечение нужно для того, чтобы создать программу на ассемблере, и где его можно достать?

- Прежде всего — это *текстовый редактор*, как отдельный, например, edit.com, входящий в состав MS-DOS, так и встроенный в какую-нибудь оболочку (например, Norton Commander, Volkov Commander и т. п.).

Мы рекомендуем пользоваться встроенным редактором DOS Navigator (<F4>), указав **on** в меню **Опции | Подсветка синтаксиса**. Так удобнее смотреть ассемблерный код. Думаем, что не следует заострять внимание на том, как пользоваться данными программами, тем более, что это выходит за рамки настоящей книги.

- Потребуется *ассемблер* — программа, которая переводит ассемблерные инструкции в машинный код. Это может быть MASM.EXE® (ML.EXE) компании Microsoft, TASM.EXE® компании Borland или другие. В принципе, большой разницы для наших примеров это пока не имеет, за исключением передачи параметров в командной строке при ассемблировании. Мы будем использовать MASM 6.11 — Macro Assembler® от Microsoft версии 6.11, что и вам советуем. Если вы используете другую программу-ассемблер, и она при ассемблировании примера выдает ошибки, то обращайтесь к *приложению 2 "Ошибки при ассемблировании"*.
- Настоятельно рекомендуем иметь *отладчик* (AFD®, SoftIce®, CodeView®). Он необходим для отладки программы и в целом для демонстрации ее работы. Предпочтительно использовать AFD или CodeView для начинающих и SoftIce для уже имеющих опыт программирования.
- В будущем вам, безусловно, понадобится *дизассемблер*, который необходим для перевода машинного кода на язык ассемблера. Мы предпочитаем IDA®, как один из самых мощных и удобных в пользовании.

Можно также скачать минимальный, но достаточный для изучения настоящего курса набор программного обеспечения по следующему адресу: **http://www.Kalashnikoff.ru**. Стоит отметить, что информация на указанном сайте постоянно пополняется. В перспективе: периодическое проведение голосований, горячие обсуждения, чат с автором, обзоры новых ресурсов по программированию, реальные встречи с читателями и многое другое.

3. Как построены главы книги?

- Ответы на часто задаваемые вопросы.
- Заметки, дополнительные примеры и алгоритмы.
- Объяснение новой темы (теория).
- Примеры программ на ассемблере (практика).

Вы сможете самостоятельно написать простую программу уже после прочтения *главы 1*. Надеемся, что изучать язык будет интересней, если мы сразу перейдем к практической части, изучая параллельно теорию. Попутно отметим, что данная книга рассчитана, в первую очередь, на людей, которые ни разу не писали программы ни на ассемблере, ни на каком другом языке программирования. Конечно, если вы уже знакомы с Basic, Pascal, C или

каким-либо иным языком, то это только на пользу вам. Тем не менее все новые термины будут подробно объясняться.

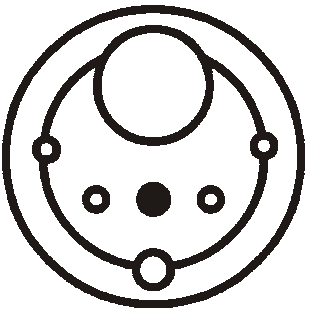
Также следует отметить, что для полного изучения курса необходимы минимальные пользовательские знания операционной системы MS-DOS, т. к. ассемблирование программ из данной книги следует выполнять именно в этой ОС. Однако вы также можете перегружаться в режим эмуляции MS-DOS или даже ассемблировать непосредственно в Windows с помощью программы Far Manager, но мы в таком случае не гарантируем корректную работу наших программ из практической части книги.

4. Какие темы будут рассмотрены в настоящей книге?

- Двоичная и шестнадцатеричная системы счисления.
- Основные команды процессоров Intel 8086, 80286, 80386, 80486.
- 16- и 32-разрядные регистры.
- Основы работы с сопроцессором.
- Сегментация памяти в реальном режиме.
- Расширенная память (XMS-память).
- Прямая работа с видеоадаптером.
- Режимы CGA, EGA, VGA, SVGA (кратко).
- Управление клавиатурой на уровне прерываний.
- Основные функции BIOS (ПЗУ) и MS-DOS.
- Работа с дисками, каталогами и файлами (как с короткими именами, так и с длинными).
- Вывод символов на принтер.
- Управление последовательным портом.
- Высокоуровневая оптимизация программ.
- Не обойдем стороной и технический английский язык, т. к. операторы ассемблера образованы от английских слов.

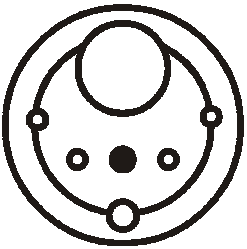
В *главе 1* мы рассмотрим шестнадцатеричную систему счисления и пример простейшей программы на ассемблере, традиционно называемой "Hello, world!".

Приятного вам изучения!



ЧАСТЬ I

Знакомьтесь: ассемблер



Глава 1

Первая программа

1.1. Шестнадцатеричная система счисления

Для написания программ на ассемблере необходимо разобраться с шестнадцатеричной системой счисления. Ничего сложного в ней нет. Мы используем в жизни десятичную систему. Не сомневаемся, что вы с ней знакомы, поэтому попробуем объяснить шестнадцатеричную систему, проводя аналогию с десятичной.

Итак, в десятичной системе, если мы к какому-нибудь числу справа добавим нуль, то это число увеличится в 10 раз. Например:

```
1 x 10 = 10
10 x 10 = 100
100 x 10 = 1000
```

и т. д.

В этой системе мы используем цифры от 0 до 9, т. е. десять разных цифр (собственно, поэтому она и называется десятичной).

В шестнадцатеричной системе мы используем, соответственно, шестнадцать "цифр". Слово "цифр" специально написано в кавычках, т. к. в этой системе используются не только цифры. От 0 до 9 мы считаем так же, как и в десятичной, а вот дальше таким образом: A, B, C, D, E, F. Число F, как не трудно посчитать, будет равно 15 в десятичной системе (табл. 1.1).

Таким образом, если мы к какому-нибудь числу в шестнадцатеричной системе добавим справа нуль, то это число увеличится в 16 раз (пример 1.1).

Пример 1.1

```
1 x 16 = 10
10 x 16 = 100
100 x 16 = 1000
```

и т. д.

Таблица 1.1. *Десятичная и шестнадцатеричная системы*

Десятичное число	Шестнадцатеричное число	Десятичное число	Шестнадцатеричное число
0	0	26	1A
1	1	27	1B
2	2	28	1C
3	3	29	1D
4	4	30	1E
...	...	...	...
8	8	158	9E
9	9	159	9F
10	A	160	A0
11	B	161	A1
12	C	162	A2
13	D	...	...
14	E	254	FE
15	F	255	FF
16	10	256	100
17	11	257	101
...	...	...	...

Вы смогли отличить в примере 1.1 шестнадцатеричные числа от десятичных? А из этого ряда: 10, 12, 45, 64, 12, 8, 19? Это могут быть как шестнадцатеричные числа, так и десятичные. Для того чтобы не было путаницы, а компьютер и программист смогли бы однозначно отличить одни числа от других, в ассемблере принято после шестнадцатеричного числа ставить символ h или H — сокращение от англ. "hexadecimal" ("шестнадцатеричное"), который для краткости часто называют просто hex. После десятичного числа, как правило, ничего не ставят. Так как числа от 0 до 9 в обеих системах имеют одинаковые значения, то числа, записанные как 5 и 5h, — одно и то же. Таким образом, корректная запись чисел из примера 1 будет следующей (примеры 1.2 и 1.3).

Пример 1.2. Корректная форма записи чисел

```
1 x 16 = 10h
10h x 16 = 100h
100h x 16 = 1000h
```

Пример 1.3. Другой вариант записи чисел

```
1h x 10h = 10h
10h x 10h = 100h
100h x 10h = 1000h
```

Для чего нужна шестнадцатеричная система и в каких случаях она применяется — мы рассмотрим в следующих главах. А в данный момент для нашего примера программы, который будет рассмотрен ниже, нам необходимо знать о существовании шестнадцатеричных чисел.

Итак, настала пора подвести промежуточный итог. Шестнадцатеричная система счисления состоит из 10 цифр (от 0 до 9) и 6 букв латинского алфавита (A, B, C, D, E, F). Если к какому-нибудь числу в шестнадцатеричной системе добавим справа нуль, то это число увеличится в 16 раз. Очень важно уяснить принцип шестнадцатеричной системы счисления, так как мы будем постоянно использовать ее при написании наших программ на ассемблере.

Теперь немного о том, как будут строиться примеры на ассемблере в данной книге. Не совсем удобно приводить их сплошным текстом, поэтому сперва будет идти сам код программы с пронумерованными строками, а сразу же после него — объяснения и примечания. Примерно так, как показано в листинге 1.1.

Листинг 1.1. Пример записи ассемблерных инструкций, применяемой в книге

```
        …
(01)    mov ah,9
(02)    mov al,8
…
(15)    mov dl,5Ah
…
```

Обратите внимание, что номера строк ставятся только в настоящей книге, и при наборе программ в текстовом редакторе эти номера ставить НЕ нужно! Номера строк ставятся для того, чтобы удобно было давать объяснения к каждой строке: в строке (01) мы делаем то-то, а в строке (15) — то-то.

Несмотря на то, что на компакт-диске, прилагаемом к книге, имеются набранные и готовые для ассемблирования программы, мы рекомендуем все-таки первое время набирать их самостоятельно. Это ускорит запоминание операторов, а также облегчит привыкание к самому языку.

И еще момент. Строчные и ПРОПИСНЫЕ символы программой-ассемблером не различаются. Записи вида:

```
mov ah,9
```

и

```
MOV AH,9
```

воспринимаются одинаково. Можно, конечно, заставить ассемблер различать регистр, но мы пока этого делать не будем. Для удобства чтения программы лучше всего операторы печатать строчными буквами, а названия подпрограмм и меток начинать с прописной.

1.2. Наша первая программа

Итак, переходим к нашей первой программе (/001/prog01.asm) (листинг 1.2).

Листинг 1.2. Наша первая программа на ассемблере

```
(01) CSEG segment
(02) org 100h
(03)
(04) Begin:
(05)
(06)   mov ah,9
(07)   mov dx,offset Message
(08)   int 21h
(09)
(10)   int 20h
(11)
(12) Message db 'Hello, world!$'
(13) CSEG ends
(14) end Begin
```

Еще раз обратим внимание: когда вы будете перепечатывать примеры программ, то номера строк ставить не нужно!

В скобках указывается имя файла из архива файлов-приложений (в данном случае — /001/prog01.asm, где 001 — каталог, prog01.asm — имя ассемблерного файла в DOS-формате).

Прежде чем пытаться ассемблировать, прочтите данную главу до конца!

Для того чтобы объяснить все операторы из листинга 1.2, нам потребуется несколько глав. Поэтому описание некоторых команд мы на данном этапе опустим. Просто считайте, что так должно быть. В самое ближайшее время мы рассмотрим эти операторы подробно. Итак, строки с номерами (01),

(02) и (13) вы игнорируете. Строки (03), (05), (09) и (11) остаются пустыми. Это делается для наглядности и удобства программиста при просмотре и анализе кода. Программа-ассемблер пустые строки опускает.

Теперь перейдем к рассмотрению остальных операторов. Со строки (04) начинается код программы. Это метка, указывающая ассемблеру на начало кода. В строке (14) стоят операторы `end Begin` (`Begin` — "начало"; `end` — "конец"). Это конец программы. Вообще вместо слова `Begin` можно было бы использовать любое другое. Например, `Start`. В таком случае, нам пришлось бы и завершать программу `End Start` (14).

Строки (06)—(08) выводят на экран сообщение `Hello, world!`. Здесь придется вкратце рассказать о регистрах процессора (более подробно эту тему мы рассмотрим в последующих главах).

Регистр процессора — это специально отведенная память для хранения какого-нибудь числа. Например, если мы хотим сложить два числа, то в математике запишем так:

```
A=5
B=8
C=A+B.
```

`A`, `B` и `C` — это своего рода регистры (если говорить о компьютере), в которых могут храниться некоторые данные. `A=5` следует читать как: *"Присваиваем A число 5"*.

Для присвоения регистру какого-нибудь значения в ассемблере существует оператор `mov` (от англ. `move` — в данном случае "загрузить"). Строку (06) следует читать так: *"Загружаем в регистр `ah` число 9"* (проще говоря, присваиваем `ah` число 9). Ниже рассмотрим, зачем это необходимо. В строке (07) загружаем в регистр `dx` адрес сообщения для вывода (в данном примере это будет строка "`Hello, world!$`"). Далее, в строке (08) вызываем прерывание MS-DOS, которое и выведет нашу строку на экран. Прерывания будут подробно рассматриваться в последующих главах, мы же пока коснемся только самых элементарных вещей.

1.3. Введение в прерывания

Прерывание MS-DOS — это своего рода подпрограмма (часть MS-DOS), которая находится постоянно в памяти и может вызываться в любое время из любой программы. Рассмотрим вышесказанное на примере (листинг 1.3).

Сразу стоит отметить, что в ассемблере после точки с запятой располагаются *комментарии*. Комментарии будут опускаться MASM/TASM при ассемблировании. Примеры комментариев:

```
;это комментарий
mov ah,9        ;это комментарий
```

В комментарии программист вставляет замечания по программе, которые помогают сориентироваться в коде.

Листинг 1.3. Программа (алгоритм) сложения двух чисел

```
НачалоПрограммы
A=5           ;в переменную A заносим значение 5
B=8           ;в переменную B значение 8
ВызовПодпрограммы Addition
;теперь C равно 13

A=10          ;то же самое, только другие числа
B=25
ВызовПодпрограммы Addition
;теперь C равно 35

КонецПрограммы
;выходим из программы

…
Подпрограмма Addition
C = A + B
ВозвратИзПодпрограммы
;возвращаемся в то место, откуда вызывали
КонецПодпрограммы
```

В данном примере мы дважды вызвали подпрограмму (процедуру) `Addition`, которая произвела сложение двух чисел, переданных ей в переменных `A` и `B`. Результат математического действия сохраняется в переменной `C`. Когда вызывается подпрограмма, компьютер запоминает, с какого места она была вызвана, и после того, как процедура отработала, возвращается в то место, откуда она вызывалась. Таким образом можно вызывать подпрограммы неопределенное количество раз с любого участка основной программы.

При выполнении строки (08) (см. листинг 1.2) мы вызываем подпрограмму (в данном случае это называется прерывание), которая выводит на экран строку. Для этого мы, собственно, и помещаем необходимые значения в регистры, т. е. готовим для прерывания необходимые параметры. Всю работу (вывод строки, перемещение курсора) берет на себя эта процедура. Строку (08) следует читать так: *"Вызываем двадцать первое прерывание"* (`int` — от англ. "interrupt" — "прерывание"). Обратите внимание, что после числа 21 стоит буква h. Это, как мы уже знаем, шестнадцатеричное число (33 в деся-

тичной системе). Конечно, нам ничего не мешает заменить строку `int 21h` на `int 33`. Программа будет работать корректно. Но в ассемблере принято указывать номера прерываний в шестнадцатеричной системе, да и все отладчики работают с этой системой.

В строке (10) мы, как вы уже догадались, вызываем прерывание `20h`. Для его вызова не нужно указывать какие-либо значения в регистрах. Оно выполняет только одну задачу: выход из программы (выход в DOS). В результате выполнения прерывания `20h` программа вернется туда, откуда ее запускали (загружали, вызывали). Например, в Norton Commander или DOS Navigator. Это что-то вроде оператора `exit` в некоторых языках высокого уровня.

Строка (12) содержит сообщение для вывода. Первое слово (`message` — сообщение) — название этого сообщения. Оно может быть любым (например, `mess` или `string` и пр.). Обратите внимание на строку (07), в которой мы загружаем в регистр `dx` адрес этого сообщения.

Можно создать еще одну строку, которую назовем `Mess2`. Затем, начиная со строки (09), вставим в нашу программу следующие команды:

```
...
(09)    mov ah,9
(10)    mov dx,offset Mess2
(11)    int 21h
(12)    int 20h
(13) Message db 'Hello, world!$'
(14) Mess2 db 'Это Я!$'
(15) CSEG ends
(16) end Begin
```

Уверены, вы поняли, что именно произойдет.

Обратите внимание на последний символ в строках `Message` и `Mess2` — `$`. Он указывает на конец выводимой строки. Если мы его уберем, то прерывание `21h` продолжит вывод до тех пор, пока не встретится где-нибудь в памяти тот самый символ `$`. На экране, помимо нашей строки, мы увидим "мусор" — разные символы, которых в строке вовсе нет.

Теперь ассемблируйте программу. Как это сделать — написано в *приложении 1*. Заметьте, что мы создаем пока только COM-файлы, а не EXE! Для того чтобы получить COM-файл, нужно указать определенные параметры ассемблеру (MASM/TASM) в командной строке. Пример получения COM-файла с помощью Macro Assembler версии 6.11 и результат выполнения программы приведен на рис. 1.1. При возникновении ошибок в процессе ассемблирования — обращайтесь к *приложению 2*, где рассматриваются типичные ошибки при ассемблировании программ.

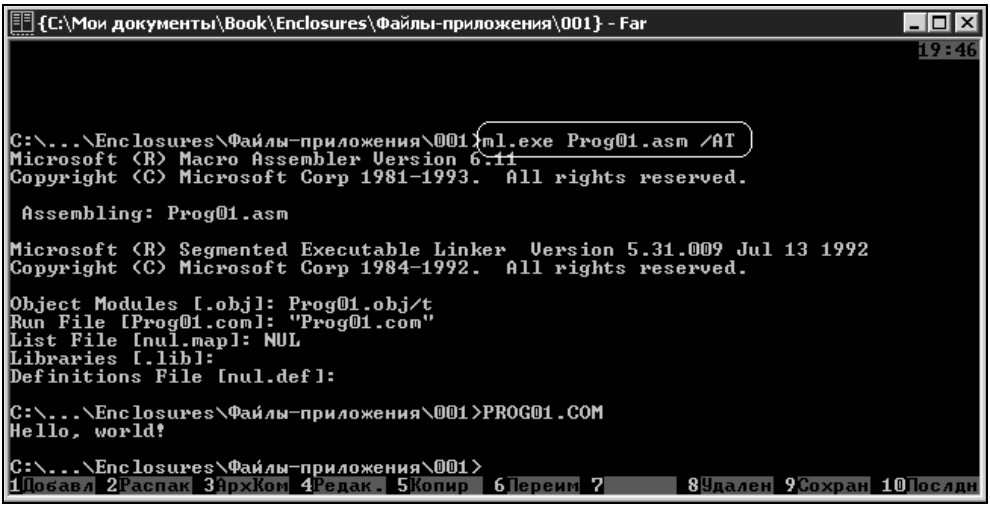


Рис. 1.1. Ассемблирование и результат выполнения программы Prog01.com

Если у вас есть отладчик (AFD, CodeView), то можно (и даже нужно!) запустить эту программу под его управлением. Это поможет вам лучше понять структуру и принцип работы ассемблера, а также продемонстрирует реальную работу написанной нами программы.

На рис. 1.2 показано, как эта программа выглядит в отладчике AFD Pro. Пока не обращайте особого внимания на различие между реальным кодом, набранным руками, и тем, как эта программа отображается в отладчике. Подробно работу отладчика мы рассмотрим в последующих главах.

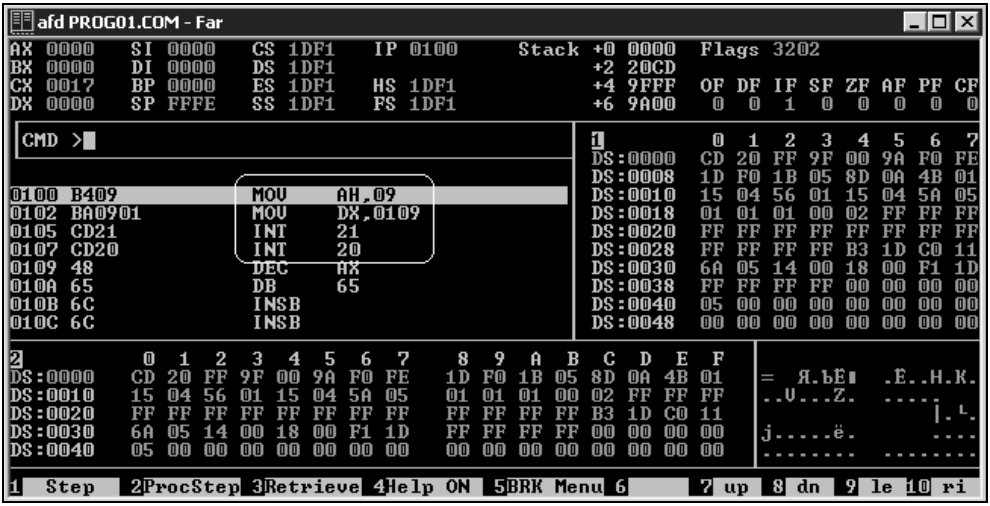


Рис. 1.2. Вид программы в отладчике AFD Pro

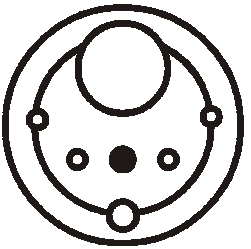
1.4. Послесловие

Целью настоящей главы не было разобраться подробно с каждым оператором. Это невозможно, если вы не обладаете базовыми знаниями. Но, прочитав 3—4 главы, вы поймете принцип и структуру программы на ассемблере.

Может быть, язык ассемблер вам показался чрезвычайно сложным, но это, поверьте, только с первого взгляда. Вы должны научиться строить алгоритм программы на ассемблере в голове, а для этого нужно будет самостоятельно написать несколько программ, опираясь на информацию из данной книги. Мы будем постепенно учить вас мыслить структурой ассемблера, составлять алгоритмы, программы, используя операторы языка. После изучения очередной главы вы будете чувствовать, что постепенно начинаете осваивать ассемблер, будет становиться все проще и проще.

Например, если вы знакомы с Бейсиком, то, ставя перед собой задачу написать программу, выводящую 10 слов "Привет", вы будете использовать операторы `FOR`, `NEXT`, `PRINT` и пр., которые тут же появятся в ваших мыслях. Вы строите определенный алгоритм программы из этих операторов, который в какой-то степени применим только к Бейсику. То же самое и с ассемблером. При постановке задачи написать ту или иную программу вы мысленно создаете алгоритм, который применим к ассемблеру и только, т. к. языков, похожих на ассемблер, просто не существует. Наша задача — научить вас создавать в уме алгоритмы, применимые к ассемблеру, т. е. образно говоря, научить "мыслить на ассемблере".

В *главе 2* мы подробно рассмотрим регистры процессора и напишем еще одну простую программу.



Глава 2

Регистры процессора

2.1. Введение в регистры микропроцессоров 8086—80186

Регистр, как мы уже выше говорили, — это специально отведенная память для временного хранения каких-то данных. Микропроцессоры 8086—80186 имеют 14 регистров. В *главе 1* мы познакомились с двумя из них: `ah` и `dx`. В табл. 2.1, 2.3 и 2.4 приведен перечень всех регистров, кроме `ip` и регистра флагов, которые будут рассмотрены отдельно.

2.1.1. Регистры данных

Таблица 2.1. *Регистры данных*

ax		**bx**		**cx**		**dx**	
ah	al	bh	bl	ch	cl	dh	dl
Аккумулятор		База		Счетчик		Регистр данных	

Регистры данных могут использоваться программистом по своему усмотрению (за исключением некоторых случаев). В них можно хранить любые данные: числа, адреса и пр. В верхнем ряду табл. 2.1 находятся шестнадцатиразрядные регистры, которые могут хранить числа от 0 до 65 535 или от 0h до FFFFh в шестнадцатеричной системе, что одно и то же.

В следующей строке расположен ряд восьмиразрядных регистров: `ah`, `al`, `bh`, `bl`, `ch`, `cl`, `dh`, `dl`. В эти регистры можно загружать максимальное число 255 (FFh). Это так называемые половинки (старшая или младшая) шестнадцатиразрядных регистров.

Мы уже изучили оператор `mov`, который предназначен для загрузки числа в регистр. Чтобы присвоить, к примеру, регистру `al` число 35h, нам необходимо записать так:

```
mov al,35h
```

а регистру `ax` — число 346Ah, так:

```
mov ax,346Ah
```

Если мы попытаемся загрузить большее число, чем может содержать регистр, то при ассемблировании программы произойдет ошибка. Например, следующие записи будут ошибочны:

~~`mov ah,123h`~~ → максимум FFh

~~`mov bx,12345h`~~ → максимум FFFFh

~~`mov dl,100h`~~ → максимум FFh

Здесь надо отметить, что если шестнадцатеричное число начинается не с цифры (12h), а с буквы (C5h), то перед таким числом ставится нуль: 0C5h. Это необходимо для того, чтобы программа-ассемблер могла отличить, где шестнадцатеричное число, а где название переменной или метки. Ниже мы рассмотрим это на примере.

Допустим, процессор выполняет команду `mov ax,1234h`. В этом случае в регистр `ah` загружается число 12h, а в регистр `al` — 34h. То есть `ah`, `al`, `bh`, `bl`, `ch`, `cl`, `dh` и `dl` — это младшие (**L**ow) или старшие (**H**igh) половинки шестнадцатиразрядных регистров (табл. 2.2).

Таблица 2.2. *Результаты выполнения различных команд*

Команда	Результат
`mov ax,1234h`	`ax = 1234h, ah = 12h, al = 34h`
`mov bx,5678h`	`bx = 5678h, bh = 56h, bl = 78h`
`mov cx,9ABCh`	`cx = 9ABCh, ch = 9Ah, cl = 0BCh`
`mov dx,0DEF0h`	`dx = 0DEF0h, dh = 0DEh, dl = 0F0h`

2.1.2. Регистры-указатели

Таблица 2.3. *Регистры-указатели*

si	Di	bp	sp
Индекс источника	Индекс приемника	Регистры для работы со стеком	

Регистры `si` (индекс источника) и `di` (индекс приемника) используются в строковых операциях. Регистры `bp` и `sp` задействуются при работе со стеком. Мы подробно их рассмотрим на примерах в следующих главах.

2.1.3. Сегментные регистры

Сегментные регистры необходимы для обращения к тому или иному сегменту памяти (например, видеобуферу). Сегментация памяти — довольно сложная и объемная тема, которую также будем рассматривать в следующих главах.

Таблица 2.4. *Сегментные регистры*

CS	DS	ES	SS
Регистр кода	Регистр данных	Дополнительный регистр	Регистр стека

2.2. Команды сложения и вычитания

Для выполнения арифметических операций сложения и вычитания в ассемблере существуют следующие операторы: `add`, `sub`, `inc`, `dec`.

2.2.1. Оператор *add*

Формат оператора `add` показан в табл. 2.5. Впоследствии мы всегда будем оформлять новые команды в подобные таблицы. В столбце **Команда** будет описана новая команда и ее применение. В столбце **Назначение** — что выполняет или для чего служит данная команда, а в столбце **Процессор** — модель (тип) процессора, начиная с которой команда поддерживается. В столбце **Перевод** будет указано, от какого английского слова образовано название оператора и дан перевод этого слова.

Таблица 2.5. *Оператор* `add`

Команда	Перевод	Назначение	Процессор
`add приемник, источник`	**Add**ition — сложение	Сложение	8086

В данном примере оператор поддерживается процессором 8086, но работать команда будет, естественно, и на более современных процессорах (80286, 80386 и т. д.).

Команда `add` производит сложение двух чисел (листинг 2.1).

Листинг 2.1. Примеры использования оператора `add`

```
mov al,10       ;загружаем в регистр al число 10
add al,15       ;al = 25; al — приемник, 15 — источник
mov ax,25000    ;загружаем в регистр ax число 25000
```

```
add ax,10000    ;ax = 35000; ax — приемник, 10000 — источник
mov cx,200      ;загружаем в регистр cx число 200
mov bx,760      ;а в регистр bx — 760
add cx,bx       ;cx = 960, bx = 760 (bx не меняется); cx —  приемник,
                 bx — источник
```

2.2.2. Оператор *sub*

Команда `sub` производит вычитание двух чисел (табл. 2.6, листинг 2.2).

Таблица 2.6. *Оператор* `sub`

Команда	Перевод	Назначение	Процессор
`sub приемник, источник`	**Sub**traction — вычитание	Вычитание	8086

Листинг 2.2. Примеры использования оператора `sub`

```
mov al,10
sub al,7              ;al = 3; al — приемник, 7 — источник
mov ax,25000
sub ax,10000          ;ax = 15000; ax — приемник, 10000 — источник
mov cx,100
mov bx,15
sub cx,bx
```

Это интересно

Следует отметить, что ассемблер максимально быстрый язык. Можно посчитать, сколько раз за одну секунду процессор сможет сложить два любых числа от 0 до 65 535.

Каждая команда процессора выполняется определенное количество тактов. Когда говорят, что тактовая частота процессора 100 МГц, то это значит, что за секунду проходит 100 миллионов тактов. Чтобы компьютер сложил два числа, ему нужно выполнить следующие команды:

```
…
mov ax,2700
mov bx,15000
add ax,bx
…
```

В результате выполнения данных инструкций в регистре `ax` будет число 17 700, а в регистре `bx` — 15 000. Команда `add ax,bx` выполняется за один