Институт управления и информационных технологий

Кафедра «Вычислительные системы и сети»

Т.Б. ЛАРИНА

Технология подготовки и отладки программ в Microsoft Macroassembler и Borland TurboAssembler

Рекомендовано редакционно-издательским советом университета в качестве методических указаний к лабораторным работам для студентов информационных специальностей ИУИТ

Москва – 2006

УДК 681.3

Л-25

Ларина Т.Б. Технология подготовки и отладки программ в Microsoft Macroassembler и Borland TurboAssembler.

Методические указания. -М.: МИИТ, 2006, - 39 с.

Методические указания предназначены для лабораторных работ по дисциплине «Программирование на ассемблере» для студентов информационных специальностей ИУИТ.

Содержат начальные сведения о языке ассемблера и структуре ассемблерной программы. Изложены вопросы создания, трансляции, компоновки и отладки исполняемой программы средствами Microsoft Macroassembler и Borland TurboAssembler.

Содержание

Тема 1. Основные понятия ассемблера

Цель работы: знакомство с понятиями «машинный язык» и «язык ассемблера», начальные представления о синтаксисе ассемблера, структуре ассемблерной программы и размещении исполняемой программы в памяти.

Задание: изучить содержательную часть темы и ответить на контрольные вопросы

Машинный язык процессора и язык ассемблера

Независимо от того, на каком языке программирования написана исходная программы, её исполняемый модуль, загружаемый в оперативную память, представляет собой ***последовательность команд процессора***, реализующий некоторый алгоритм действий.

Для выполнения каждой команды процессор предварительно считывает ее из памяти в свои регистры, считывает из памяти данные - операнды команды (если они есть) и затем исполняет команду .

Регистр процессора – это запоминающее устройство малой разрядности (от одного до нескольких байтов) внутри процессора. Часть регистров использует сам процессор, например, для размещения команд, часть – доступны для программиста.

Данные можно разместить в памяти и (или) в программно-доступных регистрах процессора. И команды и данные записаны в памяти в двоичном виде. Процессор «понимает» только двоичный код.

Машинным кодом команды называют собственно двоичный код команды. Совокупность всех команд конкретного процессора составляет понятие «машинный язык» данного процессора.

Например, машинный код команды сложения данных, размещенных в регистрах AL и BL, для процессоров Intel x86 таков:

$10001110\ 11011000_2$ (или 8ED8 в hex-виде)

Язык ассемблера - это символическая форма записи машинного языка. Каждой машинной команде соответствует определенная символическая команда ассемблера.

Так, приведенная запись машинной команды сложения содержимого регистров AL и BL на языке ассемблера для семейства процессоров Intel x86 будет выглядеть следующим образом:

ADD AL, BL , где ADD - мнемоника операции сложения.

Программа, записанная на языке ассемблера, гораздо понятнее для восприятия, чем машинный код. Однако, для получения исполняемой программы в машинных кодах придется выполнить преобразование каждой символической команды в соответствующий машинный код.

Преобразование символически записанных команд и данных в машинные коды называется ***ассемблированием (трансляцией)***. Трансляцию с языка ассемблера на машинный язык выполняет специальная программа - ***ассемблер (транслятор)***. Собственно язык ассемблера – это символический входной язык программы-транслятора.

Кроме команд процессора, язык ассемблера включает специальные служебные операторы - ***директивы***. В целом, они используются для описания структуры программы, данных, управления процессом трансляции и многого другого.

Трансляторы (ассемблеры) и компиляторы с языков высокого уровня выполняют одну задачу – получить машинный код из исходного символического текста программы. Их принципиальное отличие состоит в следующем.

Транслятор преобразует одну символическую команду языка ассемблера в однозначный машинный код команды, то есть выполняет преобразование «один к одному».

Компилятор с языков высокого уровня преобразует один символический оператор языка в последовательность из множества машинных команд, то есть выполняет преобразование «один к нескольким». Чем более абстрактен язык программирования, тем к большей последовательности команд процессора будут преобразовываться его операторы.

В случае языков высокого уровня эффективность машинного кода (объем, быстродействие) в целом зависит не от программиста, а от интеллектуальности компилятора. В ассемблере программист полностью определяет эффективность своей программы, конструируя ее непосредственно на уровне команд процессора.

Языки ассемблера называют «машинно-зависимыми». Получаемый после трансляции машинный код уникален для каждого процессора или семейства совместимых процессоров.

Синтаксис записи команды и данных

Символическая запись команды в ассемблере в общем случае состоит из четырех полей. Строка заканчивается нажатием Enter.

[Метка] Мнемоника операции Операнды [Комментарий]

Метка позволяет обозначить символический адрес команды в кодовом сегменте. Не является обязательной.

Необязательный комментарий должен начинаться с символа ";". Если комментарий занимает несколько строк, каждая из них должна начинаться с двоеточия.

Пример. Команда, прибавляющая содержимое регистра BL к содержимому регистра AL

```
A1:   ADD   AL, BL   ; сложение AL и BL
```

Здесь A1 – метка команды, ADD- мнемоника команды сложения.. Регистры BL и AL содержат операнды для команды.

Описание данных выполняется специальными директивами ассемблера. Строка описания данных (заканчивается нажатием Enter):

[Символическое имя] Формат данных Значение (несколько значений)

Формат данных задает размер ячейки памяти, выделяемый для размещения данного значения или последовательности значений.

Наиболее распространенными являются следующие форматы:

db - байты, **dw** – слова (2-байтные величины) , **dd** – двойные слова (4-байтные величины).

Указанное значение данных будет размещено в памяти в выбранном формате. При необходимости только «резервировать место» для переменной в поле значения записывают символ "**?**"

Примеры

```
X       db  5      ; однобайтная переменная с символическим именем X и значением 5
Mass    db   1, -2, 13, 4,  0  ; массив из пяти однобайтных величин с заданными значениями
        dw  ?      ; двухбайтная переменная  без  конкретного значения
```

Структура исходной программы на языке ассемблера

Для размещения загружаемой в оперативную память исполняемой программы операционная система выделяет один или несколько участков памяти - ***«сегментов».***

Структура ассемблерной программы в точности определяет будущее размещение команд и данных программы в сегментах памяти.

Исходный текст ассемблерной программы представляет собой «бумажное» описание содержимого сегментов памяти. Все, что записано в описании сегмента, именно в такой последовательности и будет размещено после загрузки программы в реальном сегменте памяти.

Сегмент памяти, в котором размещают команды, называют ***«кодовый сегмент».*** Если для размещения данных в ассемблерной программе планируется отдельный сегмент, его называют ***«сегмент данных».***

Количество и назначение сегментов (для команд или данных) определяет сам программист. В большинстве случаев достаточными являются следующие структуры исходной программы:

- два сегмента – кодовый и отдельный сегмент данных;
- один сегмент - кодовый, где размещают и команды и данные.

Пример.

Определить наибольшее из двух однобайтных переменных, размещенных в памяти. Найденное значение поместить в регистр процессора.

Приведенный исходный текст ассемблерной программы описывает два сегмента памяти – кодовый и данных. Им даны произвольные символические имена (адреса): Dseg – для сегмента данных и Cseg – для кодового сегмента.

В сегменте данных описаны (а следовательно и будут размещены) две однобайтные переменные с конкретными значениями 5 и -3. Им даны символические имена - A и B.

В кодовом сегменте размещены команды, реализующие алгоритм нахождения наибольшего из A и B и занесения его в регистр DL.

Описание каждого сегмента начинается директивой SEGMENT (начало сегмента) и заканчивается директивой ENDS (конец сегмента).

После описания сегментов следует директива END, означающая для транслятора конец исходного текста и указывающая также на команду, с которой должно начаться исполнение программы. Это последняя строчка ассемблерной программы. Не забывать завершать ее нажатием Enter !

Жирным курсивом записаны директивы ассемблера.

По умолчанию ассемблер не различает регистр букв в исходном тексте. Поэтому символические имена, мнемоники команд, директивы и т.д. можно писать и прописными и строчными буквами.

Исходная программа

```
Dseg    segment     ; начало описания сегмента данных
  a     db   5             ; переменная со значением  5
  b     db  -3             ; переменная со значением  -3
Dseg    ends        ; конец описания сегмента

Cseg    segment     ; начало описания кодового сегмента
    assume ds: dseg, cs: cseg       ; описание, какие сегментные
;регистры используем для указания адресов сегментов Dseg и Cseg
; занести адрес сегмента данных в регистр DS
m1:     mov    ax, dseg
        mov    ds, ax
; определение наибольшего значения из A и B
        mov    dl, ds:a         ; занести значение A в регистр DL
        cmp    dl, ds:b         ; сравнить DL и  значение B
        jg     m2               ; переход (jmp) на команду с меткой m2,
; если  результат сравнения «больше», т.е.  DL >B. Условие перехода
;по «больше» заложено в мнемонику самой команды (g – greater)
        mov    dl, ds:b         ; занести значение B в регистр  DL
;  завершение исполнения программы  с выгрузкой из памяти
m2:     mov    ah, 4ch
        int 21h
Cseg    ends        ; конец описания кодового сегмента
        end    m1   ; конец исходного текста программы с указанием
;метки первой исполняемой команды
```

Важные комментарии, которые надо постараться понять (или пока принять на веру):

1. Символические имена данных и метки команд, используемые в каждом сегменте – a, b, m1, m2 - являются ***внутрисегментными адресами, то есть смещениями относительно начала своего сегмента в байтах.***

Самый первый байт в любом сегменте имеет смещение 0, следующий – 1, следующий - 2 и так далее.

2. Внутрисегментные адреса (смещения) данных или команд не зависят от места размещения самого сегмента в памяти. Следовательно, для однозначного определения адреса программной переменной или команды в памяти нужны две координаты – ***«указатель сегмента» и «внутрисегментное смещение».***

В качестве указателей сегментов служат специальные регистры процессора – сегментные регистры.

Для кодового сегмента – это всегда сегментный регистр CS, для сегмента данных – любой из свободных сегментных регистров. Обычно, это сегментный регистр DS.

Директивой Assume ds: dseg, cs: cseg мы «назначили» каждому сегменту свой регистр- указатель.

Теперь должно быть понятно, почему в команде mov dl, ds:a обращение к переменной A в сегменте Dseg делается как ***ds:a.*** Справедливо будет и обращение ***ds:0***, так как внутрисегментное смещение этой переменной в сегменте данных равно нулю. В этом случае, мы могли не использовать символическое имя при описании переменной.

3. Физические адреса команд и данных в памяти при исполнении программы ***процессор вычисляет аппаратно:***

адрес сегмента + внутрисегментное смещение

Адрес сегмента процессор определяет по значению, находящемуся в назначенном сегментном регистре.

Поэтому, изначально нужно программно позаботиться о занесении адресов сегментов в соответствующие сегментные регистры. Это делается программистом для всех своих программных сегментов, кроме кодового (регистр CS загружает операционная система).

Первыми командами в нашем исходном тексте были команды, заносящие адрес сегмента данных Dseg в сегментный регистр DS:

```
m1:     mov   ax, dseg
```

```
mov     ds, ax
```

Замечание. Строго говоря, заносимое в сегментный регистр значение еще не является адресом сегмента. Настоящий адрес сегмента процессор также вычислит аппаратно на основе значения сегментного регистра. Способ аппаратного определения адреса сегмента зависит от режима работы процессора.

4. При чтении команды из памяти для исполнения ее внутрисегментный адрес задает для процессора ***регистр IP.***

Таким образом, «координатой» в памяти очередной исполняемой команды для процессора являются значения ***CS:IP***.

Контрольные вопросы

1. Что такое машинный язык ?
2. Какова функция транслятора?
3. Чем отличается язык ассемблера от языков высокого уровня?
4. Что такое регистр процессора? Являются ли регистры программно доступными?
5. Как выглядит символическая запись команды?
6. Где можно размещать данные для программы?
7. Как определить конкретное значение байта, слова, двойного слова в памяти?
8. Что называют сегментом памяти?
9. Какова структура исходной программы на ассемблере?
10. Каково минимальное количество описываемых в программе сегментов?
11. Каково назначение сегментных регистров?
12. Поясните назначение директивы ASSUME
13. Что называют внутрисегментным смещением?
14. Как процессор определяет физические адреса команд и данных в памяти?
15. Какова для процессора координата в памяти команды, подлежащей исполнению?
16. Зачем нужно программно заносить адреса сегментов в назначенные им сегментные регистры?
17. Поясните любую строку в примере исходной программы

Тема 2. Подготовка исполняемой программы

Цель работы: практическое освоение всех этапов создания исполняемого модуля ассемблерной программы в системах разработки Turbo Assembler (TASM) и Macro Assembler (MASM)

Задание:

1. Создать исходный модуль приведенного примера ассемблерной программы.
2. Освоить трансляцию, получение объектного модуля и файла листинга, получение исполняемого модуля программы в системах разработки TASM и MASM.
3. Понимать содержимое файла листинга
4. Ответить на контрольные вопросы в конце темы.

Разработка ассемблерной программы состоит из следующих этапов:

a) разработка алгоритма и написание исходного текста программы (исходного модуля);

b) трансляция (ассемблирование) исходного текста программы в объектный модуль - машинные коды команд и данных;

c) компоновка объектного модуля в исполняемый модуль программы;

d) отладка исполняемого модуля.

Интегрированные пакеты Microsoft Macro Assembler (MASM) и Borland Turbo Assembler (TASM) включают в себя все необходимые средства для редактирования, ассемблирования, компоновки и отладки ассемблерных программ для процессоров Intel и их клонов.

Ассемблеры Tasm.exe и Masm.exe транслируют исходный текст ассемблерной программы в машинные коды.

Ассемблер Masm поддерживает классический синтаксис ассемблерной программы, называемый .masm. Пример программы в Теме 1 соответствует этому синтаксису.

Ассемблер Tasm по умолчанию поддерживает синтаксис .masm. Дополнительно он имеет режим синтаксиса - .ideal, который отличается

от .masm синтаксисом ряда директив, связанных с описанием программных сегментов.

Для универсальности следует пользоваться синтаксисом классического режима .masm. Кроме того, он более логичен и понятен.

Компоновщики Tlink.exe и Link.exe преобразуют созданный транслятором объектный модуль (или несколько объектных модулей) в ***исполняемый модуль***. Формат исполняемого модуля включает дополнительную информацию, необходимую операционной системе для загрузки программы в память и в дальнейшем освобождения памяти.

Отладчики предоставляют средства для отладки исполняемой программы. Наиболее удобным для отладки представляется отладчик пакета TASM Turbo Debugger (**Td.exe**)

Для работы с транслятором и компоновщиком пакетов MASM и TASM требуется режим командной строки. Использование режима командной строки операционной системы требует хорошего знания командного интерфейса. Гораздо удобнее использовать любые файловые оболочки типа FAR, Norton Commander или другие с режимом командной строки и имеющие встроенные редакторы текста.

Создание исходного модуля

Исходный файл ассемблерной программы должен иметь расширение **.asm**. Имя файлу следует давать длиной не более 8 символов – классический синтаксис командной строки. Например: prog.asm.

Исходный текст готовится простым текстовым редактором, например, встроенными редакторами файловых оболочек FAR, NC. Редактирование нового файла в них выполняется нажатием Shift+F4. Нельзя использовать текстовый процессор Microsoft Word или подобные, создающие специфические коды управляющих символов.

Для правильного представления русоязычных комментариев в программе текстовый редактор должен поддерживать кодировку DOS/OEM (кодовая страница 866).

Любая строчка в тексте исходной программы должна заканчиваться нажатием «Enter».

Дальнейшие этапы работы с исходным модулем описаны на примере программы, приведенной в Теме 1. Пусть файл с ее исходным текстом называется ***primer.asm***

Трансляция в машинный код

При трансляции исходного модуля программы ассемблер последовательно выполняет:

- ***синтаксическую проверку исходного текста*** с выводом диагностических сообщений;
- ***трансляцию строк программы***. Символические команды и данные преобразуются в машинные коды, создается ***объектный модуль***.

Результатом трансляции являются три выходных файла (рис.1). По умолчанию, если не указаны пути к этим файлам, они создаются в каталоге транслятора.

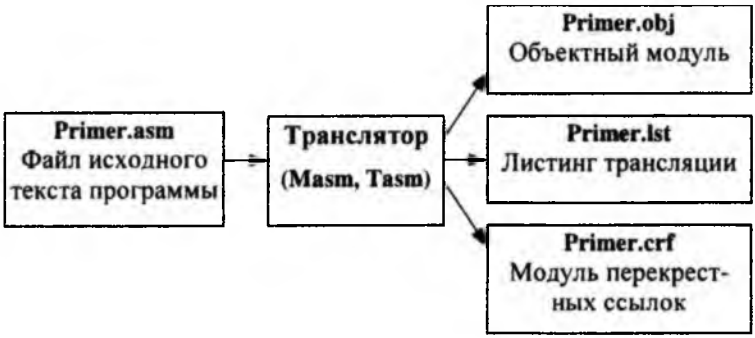


Рис.1 Модули, создаваемые транслятором

Объектный модуль - ***primer.obj*** содержит машинные коды команд и коды данных. Создается транслятором всегда.

Файл листинга - ***primer.lst*** по умолчанию не создается, однако крайне желателен. Информация, содержащаяся в файле листинга, очень полезна для эффективной отладки программы. Термин "листинг" в ассемблерной среде обозначает именно этот продукт деятельности транслятора, а не распечатку на бумаге исходного текста программы.

Текстовый файл перекрестных ссылок - ***primer.crf*** является информационным. Содержит сведения об используемых в программе символических именах. Не является обязательным. От создания этого файла можно отказаться.

После завершения трансляции диагностические сообщения ассемблера можно увидеть на экране по нажатию клавиш «Ctrl+O». Они же сохраняются в файле листинга.

Транслятор TASM

Запуск транслятора **Tasm.exe** делается в командной строке с указанием имен необходимых модулей и ключей трансляции.

tasm.exe [/ключи] primer.asm [primer.obj] [primer.lst] [primer.crf]

В скобках указаны необязательные имена модулей. Разделительное поле между именами – не менее одного пробела.

По умолчанию, объектный модуль (.obj) создается всегда и размещается в каталоге транслятора, файлы листинга и перекрестных ссылок – не создаются.

Некоторые полезные ключи транслятора:

/L - создать файл листинга ;

/N – не создавать в листинге таблицу описания символических имен (для экономии при печати листинга);

Справку по всем ключам трансляции можно получить запуском справочной системы транслятора:

tasm.exe /?

Используемые ключи можно включить в конфигурационный файл транслятора – Tasm.cfg, который должен находиться в каталоге ассемблера. В этом случае задавать ключи в командной строке не понадобится.

Примеры запуска транслятора

- ***tasm.exe /L /N primer.asm***

Исходный модуль ассемблерной программы находится в каталоге транслятора. Здесь же будут созданы: объектный модуль primer.obj и файл листинга primer.lst без таблицы символических имен (так как использован ключ /N).

- ***d:\TASM\tasm.exe /L a:\primer.asm a:\ primer.obj a:\primer.lst***

Исходный модуль ассемблерной программы находится не в каталоге транслятора. Поэтому, обязательно указываем пути к исходному модулю и создаваемым файлам. В корневом каталоге флоппи-диска будут созданы объектный модуль primer.obj и файл листинга primer.lst.

Транслятор MASM

Ассемблер **Masm.exe** может работать в двух режимах: запуск в командной строке с параметрами или диалоговом режиме.

Запуск в командной строке аналогичен транслятору Tasm. Запуск Masm.exe в режиме диалога делается без параметров:

- Masm.exe

Далее транслятор последовательно запрашивает:

- *Source filename* - путь к исходному модулю,
- *Object filename* - путь к размещению объектного модуля,
- *Source listing [nul.lst]* - путь к размещению файла листинга (по умолчанию не создается)
- *Cross reference [nul.crf]* - путь к размещению файла перекрестных ссылок (по умолчанию не создается).

Предлагаемое транслятором имя NUL для файла листинга и файла перекрестных ссылок означает, что эти файлы по умолчанию созданы не будут. Для их создания надо обязательно дать путь к файлу.

Пример диалога

Исходный модуль ассемблерной программы находится в корневом каталоге флоппи-диска (путь к нему - a:\primer.asm) . С помощью Masm.exe транслируем его в объектный модуль и получим файл листинга – оба файла в корне флоппи-диска.

После запуска Masm.exe:

Source filename [.asm]: ***a:\primer***
Object filename [primer.obj]: ***a:***
Source listing [nul.lst]: ***a:\primer***
Cross reference [nul.crf]:

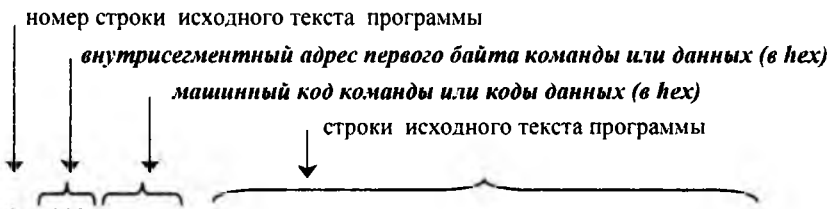
Слева – запросы транслятора, справа (жирным курсивом) - наши ответы. В скобках транслятор предлагает свои предложения по умолчанию: имена файлов, расширение или не создавать файл (имя NUL).

В результате этого диалога после трансляции файла a:\primer.asm будут созданы: объектный модуль a:\primer.obj и файл листинга a:\primer.lst.

Листинг трансляции

Разберем содержимое файла листинга на примере файла ***primer.lst*** - листинга трансляции исходной программы primer.asm..

Turbo Assembler Version 4.1 11/09/06 00:08:12 Page 1 primer.asm




```
1   0000                Dseg  segment     ; начало описания сегмента данных
2   0000 05                   a    db  5          ; переменная со значением 5
3   0001 FD                   b    db  -3         ; переменная со значением -3
4   0002                Dseg  ends        : конец описания сегмента
5   0000                Cseg  segment     : начало описания кодового сегмента
6                       assume ds: dseg, cs: cseg   : описание, какие сегментные
7 ; регистры используем для указания адресов сегментов Dseg и Cseg
8 ; занести адрес сегмета данных в регистр DS
9   0000 B8 0000 s      m1:   mov   ax, dseg
10  0003 8E D8                mov   ds, ax    ; определение наибольшего из A и B
11  0005 8A 16 0000 r         mov   dl, ds:a  ; занести значение A в регистр DL
12  0009 3A 16 0001 r         cmp   dl, ds:b  ; сравнить DL и значение B
13  000D 7F 04                jg    m2        ; переход (jmp) на команду с меткой m2.
14 : если результат сравнения "больше". т.е. DL >B. Условие перехода
15 : по "больше" заложено в мнемонику самой команды (g - greater)
16  000F 8A 16 0001 r         mov   dl, ds:b  : занести значение B в регистр DL
17 : завершение исполнения программы с выгрузкой из памяти
18  0013 B4 4C        m2:     mov   ah, 4ch
19  0015 CD 21                int 21h
20  0017                Cseg  ends        : конец описания кодового сегмента
21                            end   m1    : конец исходного текста программы
```

Turbo Assembler Version 4.1 11/09/06 00:08:12 Page 2

Symbol Table (Таблица символических имен)

Symbol Name	Type	Value
??DATE	Text	"11/09/06"
??FILENAME	Text	"primer "
??TIME	Text	"00:08:12"
??VERSION	Number	040A
@CPU	Text	0101H
@CURSEG	Text	CSEG
@FILENAME	Text	PRIMER
@WORDSIZE	Text	2
A	**Byte**	**DSEG:0000**
B	**Byte**	**DSEG:0001**
M1	**Near**	**CSEG:0000**
M2	**Near**	**CSEG:0013**

Groups & Segments (Характеристики сегментов)	**Bit**	**Size**	Align	Combine	Class
CSEG	**16**	**0017**	Para	none	
DSEG	**16**	**0002**	Para	none	

Листинг трансляции представляет собой текстовый файл, содержащий массу полезной информации для будущей отладки программы.

Листинг повторяет транслируемые ***строки исходного текста*** программы, нумеруя их в десятичном виде. Когда транслятор обнаруживает синтаксические ошибки в исходном тексте, диагностические сообщения на экране будут содержать номер строки исходного текста и характеристику ошибки. Эта информация допишется также к соответствующей строке в файл листинга.

Во второй колонке листинг показывает ***внутрисегментные адреса (смещения) команд и данных*** после трансляции в шестнадцатиричном коде (hex-коде). Внутрисегментный адрес является 16-разрядным значением, поэтому представляется четырьмя hex – цифрами.

В третьей колонке содержится главный продукт деятельности транслятора - ***машинные коды команд или коды данных*** в hex-коде. Один байт представляется двумя hex-цифрами.

Последовательность байтов машинных кодов размещается в сегменте, начиная с указанного внутрисегментного адреса. То есть, приведенный слева внутрисегментный адрес – это адрес первого байта для многобайтных команд или данных. Для каждого последующего байта машинного кода его внутрисегментный адрес больше на 1.

Обратите внимание, что служебные директивы транслятора не порождают каких-либо кодов.

В конце листинга находится ***служебная информация транслятора***: таблица символических имен и характеристики программных сегментов.

Таблица символических имен содержит:

1. Символические имена и метки исходной программы и их конкретное значение - внутрисегментный адрес - после трансляции. Обратите внимание на сведения о сегменте и внутрисегментном адресе переменных A, B и меток команд M1 и M2. Сравните эти сведениями с листингом .

2. Значения служебных переменных ассемблера, содержащих текущую дату, время, имя исходного файла и многое другое.

Главные характеристики сегментов - это фактический размер программных сегментов в байтах после трансляции (сравните по размеру машинного кода) и разрядность внутрисегментного адреса. Для программ реального режима эти адреса 16 -разрядные.

Служебная информация ассемблеров TASM и MASM может отличаться только по форме. По желанию от служебной информации в файле листинга можно отказаться, добавив при трансляции к ключу /L еще ключ /N.

Компоновка объектного модуля в исполняемый

Из объектного модуля нашей программы - ***primer.obj***, созданного транслятором, компоновщик создаст два выходных файла: исполняемый модуль и карту связи (рис.2). В общем случае, на входе компоновщика может быть несколько объектных модулей.

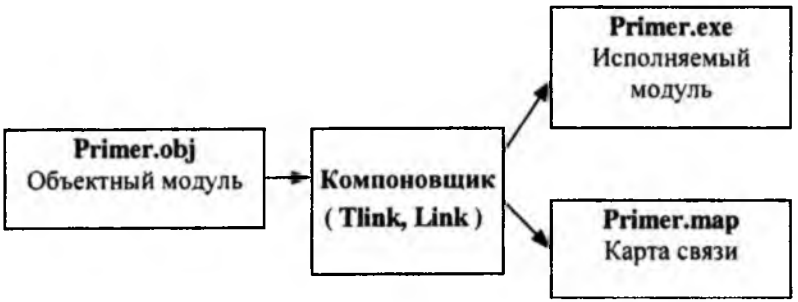


Рис. 2 Модули, создаваемые компоновщиком

Модуль ***primer.exe*** представляет собой исполняемую программу, пригодную для загрузки операционной системой в оперативную память. При определенных условиях из объектного модуля компоновщик может создать исполняемый модуль типа **.com**.

Файл карты связи ***primer.map*** является информационным и не обязателен. Он содержит имена, загрузочные адреса и длины всех сегментов программы. Эта информация может быть использована отладчиком.

Компоновщик TLINK

Запуск компоновщика **Tlink.exe** делается в командной строке с указанием имен необходимых модулей и ключей.

tlink.exe [/ключи] primer.obj [primer.exe] [primer.map]

В скобках указаны необязательные имена модулей. Разделительное поле между именами – не менее одного пробела.

По умолчанию исполняемый модуль .exe всегда создается в каталоге объектного модуля.

Некоторые полезные ключи компоновщика:

/X - не создавать файл карты связи (.map);

/3 – разрешить компоновку объектного кода, в котором используются 32-разрядные регистры процессора;

/T - создать исполняемый файл типа .com (только для исходных модулей определенной структуры).

Исполняемый модуль типа .com можно получить только из исходного модуля, удовлетворяющего следующим требованиям:

- состоит только из кодового сегмента;
- команды в кодовом сегменте размещаются, начиная с внутрисегментного адреса 100h;
- исполнение программы завершается командой int 20h.

Примеры запуска компоновщика

ξ ***tlink.exe primer.obj***

Исполняемый модуль primer.exe и карта связи primer.map будут созданы в каталоге объектного модуля.

ξ ***tlink.exe /x a:\primer.obj***

Из объектного модуля, находящегося в корневом каталоге флоппи-диска - a:\primer.obj, будет создан исполняемый модуль и размещен там же - a:\primer.exe. Файл primer.map создан не будет.

Компоновщик LINK

Компоновщик **Link.exe** может работать в двух режимах:

- запуск в командной строке с параметрами;
- диалоговый режим.

Запуск в командной строке аналогичен компоновщику Tlink.exe.

Запуск в режиме диалога делается без параметров:

- Link.exe

Далее компоновщик последовательно запрашивает:

- *Object modules [.obj]* - путь к объектному модулю,
- *Run file* - путь к размещению исполняемого модуля,

- *List file [nul.map]* - путь к размещению файла карты связи (по умолчанию не создается)
- *Libraries [.lib]* - путь к размещению модуля бибилотеки, который будет объединен с объектным модулем (по умолчанию не используется).

Пример диалога

Объектный модуль находится в корневом каталоге флоппидиска (a:\primer.obj) . С помощью компоновщика Link.exe хотим получить исполняемый модуль a:\primer.exe без создания карты связи.

После запуска Link.exe:

Object modules [.obj] : ***a:\primer***
Run file [primer.exe] : ***a:***
List file [nul.map] :
Libraries [.lib] :

Слева – запросы компоновщика, справа (жирным курсивом) - наши ответы. В скобках компоновщик предлагает свои предложения по умолчанию: имена файлов, расширение или не создавать файл (имя NUL).

В результате, после обработки объектного модуля a:\primer.obj компоновщик создаст исполняемый модуль a:\primer.exe.

После завершения работы компоновщика его сообщения на экране можно видеть по «Ctrl+O».

Сообщение компоновщика ***"Warning: No Stack"*** не является ошибкой. Это констатация факта, что исходный текст не содержал описания стекового сегмента.

Контрольные вопросы

1. Каковы этапы получения исполняемой ассемблерной программы?
2. Каковы требования к имени файла исходной ассемблерной программы?
3. Что такое трансляция (ассемблирование)?
4. Чем отличается транслятор TASM от MASM?
5. Какие модули создает транслятор? Какие из них являются необязательными? От каких можно отказаться?
6. Что такое объектный модуль?
7. Где размещает транслятор файл объектного модуля по умолчанию?
8. Как получить файл листинга трансляции?
9. Какую информацию содержит файл листинга?
10. От какого рода информации в файле листинга можно отказаться и каким образом?
11. Что такое исполняемый модуль?
12. Как получить исполняемый модуль типа .exe, .com?
13. Чем отличаются компоновщики TLINK и LINK?
14. Как просмотреть сообщения компоновщика, когда его окно закрылось после выполнения?
15. Является ли ошибкой его сообщение "Warning: No Stack"?
16. Где размещает компоновщик файл исполняемого модуля по умолчанию?
17. Поясните назначение ключей компоновщика: /X и /3.

Тема 3. Отладка ассемблерных программ

Цель работы: освоение технологии отладки ассемблерных программ в отладчике системы разработки TASM.

Задание:

1. Изучить содержательную часть темы.

2. Освоить на примере модуля primer.exe технологию отладки программы:

- исполнение программы в различных режимах (пошаговое, с точками останова), изменение команд во время отладки, повторное исполнение команд программы без ее перезагрузки;

- контроль и/или изменение данных различных форматов;

- контроль и/или изменение содержимого регистров, контроль/установку флагов;

- использование файла листинга и подготовку отладочных примеров для ассемблерной программы;

Отладчик Turbo Debugger

Отладчик **Td.exe**, входящий в пакет Borland TASM, предоставляет обширные возможности контроля исполнения программы: трассировку и пошаговое выполнение программы, просмотр, изменение и отслеживание значений переменных.

Трассировка - это выполнение программы по одной команде (для ассемблерных программ) или по одному оператору для языка высокого уровня.

Пошаговое выполнение - это выполнение программы по одной команде (одному оператору), но с пропуском вызовов процедур и функций, что увеличивает скорость отладки.

После загрузки Отладчика автоматически открывается многооконный ***«экран процессора» (CPU),*** который используется при отладке ассемблерных программ (рис.3).

«Экран процессора» состоит из пяти окон.

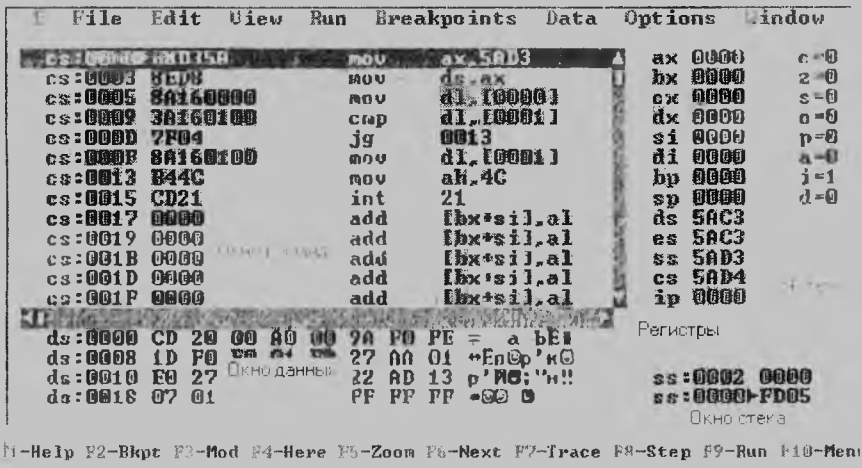


Рис 3. «Экран процессора» (CPU)

В *Окне команд (кода)* происходит собственно исполнение команд программы.

Окно данных позволяет просмотреть или изменить содержимое любой области памяти.

Окно стека используют для работы с областью данных, организованной в виде стека.

Окно регистров показывает и позволяет изменять содержимое регистров процессора.

Окно флагов показывает и позволяет изменить состояние флагов процессора.

Содержимое сегментов данных, регистров, машинные коды команд Отладчик показывает в hex-коде.

Для смещения изображения в окнах (прокрутки) используют клавиши «↑», «↓», «*PgUp*» или «*PgDn*». Для перехода между окнами клавишу табуляции «*Tab*» .

Отладчик имеет традиционную систему главного меню, поскольку он используется и для отладки программ на языках высокого уровня.

Нижняя строка экрана содержит перечень служебных и функциональных клавиш: от F1 до F10.

Служебные клавиши:

F1 –справка,

F3 и F6 - выбор модуля и переключение окон в режиме символической отладки;

F5 – увеличение/уменьшение окна;

F10 – главное меню отладчика.

Функциональные клавиши F2, F7, F8, F4 и F9 определяют режим исполнения команд программы.

Большинство операций, используемых при отладке ассемблерных программ, предусмотрено в «экране процессора» в виде ***локальных меню***.

Находясь в любом из окон, нажав и удерживая клавишу *«Ctrl»*, можно получить локальное меню возможных действий в этом окне. Оно появится в нижней строке экрана. Другой вариант – воспользоваться правой кнопкой мыши.

Загрузку в память исполняемой программы для отладки можно выполнить через пункт главного меню Отладчика *Fileo Open* или вместе с загрузкой Отладчика:

- Td.exe a:\primer.exe

Завершение работы Отладчика выполняется через пункт *Fileo Quit* главного меню.

Окно команд. Режимы исполнения команд. Встроенный транслятор

После загрузки в память исполняемой программы, при трансляции и компоновке которой не предусматривалось создание символьной информации для Отладчика, *в окне команд появится фрагмент содержимого кодового сегмента программы* (рис.4).

Содержимое сегмента «прокручивается» в окне по мере выполнения команд или принудительного сдвига изображения.

Рис.4 Окно команд

Каждая строчка окна команд содержит:

- ***адрес команды*** в виде пары значений – указатель сегмента CS и внутрисегментное смещение ;

- ***машинный код команды*** в hex-коде:

- ***символическую команду*** программы, в которой вместо символических имен (адресов) используются фактические числовые *внутрисегментные смещения.*

Все числовые значения Отладчик показывает *в hex-коде.*

Команда, которая *будет выполняться следующей*, помечена в окне команд символом ▯ . Будем называть ее текущей командой.

Режимы исполнения команд

Выбираются функциональной клавишей. Выбор режима определяется стратегией отладки и здравым смыслом.

F7 (трассировка) – исполнение одной текущей команды. После исполнения текущей команды Отладчик приостанавливает процессор, дожидаясь следующего действия пользователя. Это самый эффективный способ начала отладки любой программы, позволяющий контролировать результаты выполнения каждой команды.

F8 (пошаговое исполнение) – исполнение одной текущей команды. Отличие от трассировки в том, что исполнение процедур происходит без остановок.

F4 (исполнение до курсора) - непрерывное исполнение последовательности команд от текущей до той, на который пользователь установил курсор. Исполнение до курсора используется, если вы полностью уверены в правильности работы этого программного фрагмента.

F9 (исполнение) - исполнение команд, начиная с текущей, происходит без останова и неконтролируемо. Последствия такого исполнения:

отлаженная программа завершится и будет выгружена из памяти. Судить о ее работоспособности можно только, когда в программе будет предусмотрен ввод данных и вывод результатов на экран;

не отлаженная логика программы не приведет к завершению и выгрузке. Результат - «зависание» программы.

Пользоваться режимом F9 при отладке можно только в сочетании с использованием точек останова. При расставленных точках останова исполнение по F9 будет происходить от текущей команды до команды, помеченной точкой останова (исполняться она не будет).

Точки останова расставляются клавишей F2. Подведите курсор к команде и нажмите F2. Команда будет помечена красной строкой. Повторное нажатие F2 снимет эту точку останова.

На рис.5 видно, что точка останова поставлена на команде, с которой начинается завершение и выгрузка программы из памяти. Это делается для того, чтобы не перезагружать постоянно свою программу после ее завершения.

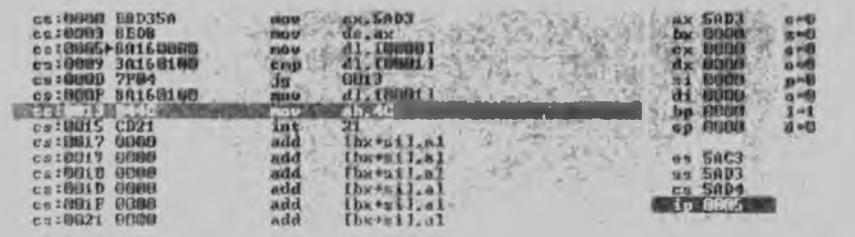


Рис. 5 Точка останова установлена на команде mov ah,4ch.

Для того, чтобы после этой остановки продолжить отладку с начала, достаточно *занести в регистр IP значение внутрисегментного адреса команды*. Для процессора текущей командой всегда является та, координаты которой находятся в CS:IP.

В нашем примере, для повторения отладки программы не имеет смысла возвращаться к первым двум командам (загрузка адреса сегмента данных в регистр DS), так как программа не выгружалась из памяти. Начнем исполнять ее сразу с команды по адресу CS:0005.

На рисунке видно, что в регистр IP занесено значение 0005. Команда со смешением 0005 в кодовом сегменте автоматически стала текущей – на ней установился символ □ .

Отладка продолжится с начала программы.

Встроенный транслятор

Отладчик позволяет в упрощенном виде транслировать символическую команду непосредственно в памяти, размещая машинный код по заданному адресу.

Для этого надо установить курсор в кодовом сегменте на адрес, с которого следует занести машинный код, и нажать любую клавишу. Всплывающая строка с приглашением *«Enter instruction to assemble»* предназначена для ввода символической команды.

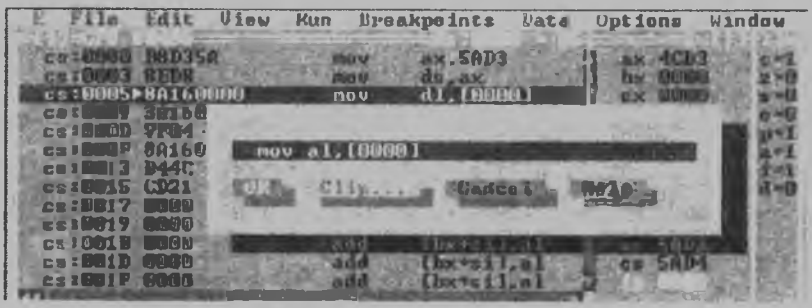


Рис.6 Трансляция команды в Отладчике

Трансляция средствами Отладчика имеет следующее важное ограничение:

- *внутрисегментные адреса команд или данных, числовые или символьные константы должны задаваться только в hex-коде.*

На рис.6 машинный код команды mov al, [0000] будет размещен с адреса CS:0005. Прежнее содержимое байтов памяти, которые займет новая команда, *будет потеряно.*

Следует иметь в виду - длина машинного кода новой команды не должна превысить отведенного для нее места в кодовом сегменте. Иначе, будет испорчен код последующей команды.

Трансляцию команды непосредственно в Отладчике имеет смысл делать только в случае обнаружения единичных ошибок в программе. Чтобы не прерывать отладку для внесения изменений в исходный текст, трансляции, компоновки и загрузки обновленного исполняемого модуля.

Изменения в машинных кодах команд, сделанные в Отладчике, не сохраняются в исполняемом модуле после выгрузке из памяти. Все равно придется откорректировать исходный текст и получить новый исполняемый модуль.

Окна регистров и флагов

В окне флагов показано текущее состояние арифметических и управляющих флагов процессора. Любой флаг имеет значение 0 или 1. Находясь в окне флагов, можно переключить значение флага нажатием клавиши *«Пробел»*.

Арифметические флаги позволяют судить о свойствах результата арифметической или логической команды: **C** - перенос, **A** - дополнительный перенос, **O** - переполнение, **Z** - ноль, **S** - знак, **P** - четность.

Управляющие флаги устанавливаются программистом: **I** - разрешение прерываний, **D** - флаг направления.

Окно регистров показывает содержимое программно-доступных регистров процессора.

Меню возможных действий с регистрами можно получить, находясь в окне регистров и нажав клавишу «*Ctrl*» или правую кнопку мыши (рис.7). Нажатие «*Ctrl* + *первая буква действия*» дает прямой доступ к этому действию.

Рис.7 Меню действий с регистрами

Содержимое регистра можно увеличить на 1 (*Increment*), уменьшить на 1 (*Decrement*), обнулить (*Zero*), изменить (*Change*).

Рис.8 Изменение значения регистра

Переключатель *Registers 32-bit* позволяет выбрать для отображения регистры 16-разрядных процессоров (по умолчанию) или 32-разрядных. На рис.9 окно регистров показывает состояние программно-доступных регистров 32-разрядного процессора Intel.

Рис.9 Регистры 32-разрядного процессора

Однобайтные регистры входят в состав двухбайтных регистров. Например, регистр DH (high) является старшим байтом регистра DX, а регистр DL (low) – его младшим байтом. В свою очередь, 16-разрядные регистры являются младшими половинами 32-разрядных регистров. Например, регистр DX - младшая половина регистра EDX.

Окно данных

В окне данных можно наблюдать или изменять содержимое байтов в любом сегменте памяти. Меню возможных действий для окна данных можно получить, находясь в нем и нажав клавишу «*Ctrl*» или правую кнопку мыши (рис.10).

Основные действия в окне данных:

- настройка окна на адрес в памяти – *Goto*;
- изменение содержимое байтов памяти – *Change*;
- выбор формата отображения данных – *Display as*;
- работа с блоком данных – *Block*;
- поиск кодовой комбинации в сегменте памяти – *Search*;

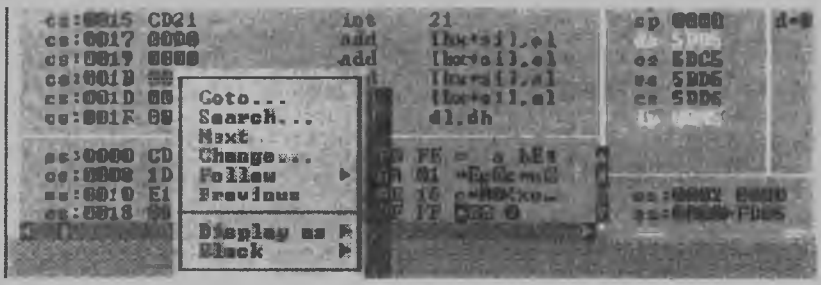


Рис.10 Меню действий в окне данных

Настройку окна на нужный сегмент и смещение следует делать только *после выполнения команд загрузки сегментных регистров*. Иначе, неправильный адрес сегмента приведет вас в другое место памяти, где вы не увидите ожидаемых значений.

На рис.11 показана настройка окна данных. После выбора опции *Goto* запрашивается адрес начала области памяти. В нашем примере байты переменных A и B находятся в начале сегмента данных. Соответственно, указываем координату – DS:0000.

Обратите внимание, что перед настройкой окна данных команды загрузки сегментного регистра DS уже выполнены – указатель текущей команды стоит по адресу CS:0005.

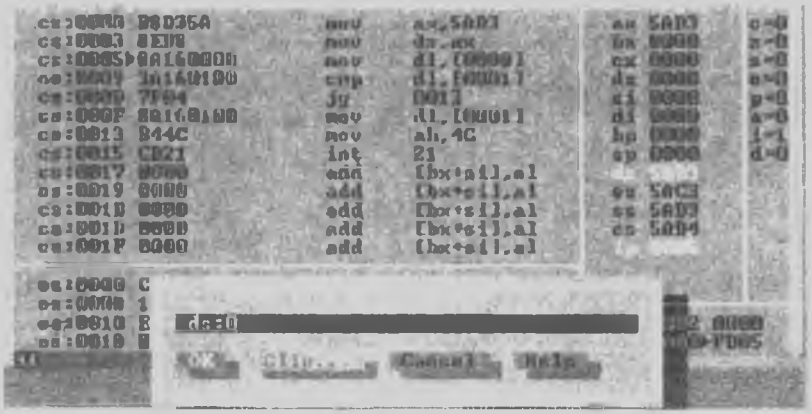


Рис.11 Настройка окна данных на область в сегменте данных

После настройки окна на сегмент данных мы увидим исходные значения байтов данных в памяти в hex-коде: 05 (5) и FD (–3) (рис.12).

Рис.12 Размещение исходных данных

По умолчанию, Отладчик показывает область данных, как *последовательность байтов*. В правой части окна данных выводятся ASCII-символы, соответствующие каждому байту.

Если числовые данные являются многобайтными (например, слово), обратите внимание на последовательность размещения в памяти старшего и младшего байтов числа.

Правила размещения многобайтных числовых величин в памяти: старший байт располагается по старшему адресу. Так, если к первым двум байтам сегмента данных относиться как к числу формата слова, то его значение – FD05h.

Отладчик дает возможность установить любой формат отображения данных с помощью опции *Display As* . Для целых чисел это –

байт, слово, двойное слово или учетверенное слово. Рис.13 демонстрирует отображение числовых данных в формате слова.

Рис.13 Отображение данных в формате слова

Изменение данных можно делать обычным набором через пробел последовательности новых значений, начиная байта, на который установлен курсор, или через опцию *Change*.

Вводимые значения должны задаваться в hex-коде. Величины, начинающиеся с буквы, должны иметь первой цифрой 0 и заканчиваться признаком h. Например: 25, 7Ah, 8Ch, но (!) - **0AAh, 0F3h**.

Рис.14 Изменение данных

Рис.14 показывает изменение значений перемнных A и B на другие значения: –6 и 1.

Опция *Block* меню окна данных позволяет работать с блоками памяти и предлагает следующий набор действий: обнуление области данных - Clear, копирование блока памяти - Move, установка каждого байта области - Set, чтение из файла в область памяти -Read, запись из области памяти в файл - Write.

Пример на рис.15 показывает использование действия Set для установки первых пяти байтов в сегменте данных в значение FF.

Рис.15 Установка последовательности байтов в одинаковое значение

Окно стека

Окно стека показывает несколько последних *слов данных*, записанных в память командой процессора PUSH.

Локальное меню области стека вызывается обычным образом - нажатием правой кнопки мыши или клавишей «*Ctrl*» (рис.16).

Рис.16 Меню окна стека

Goto настраивает окно стека на указываемый адрес.

Origin позиционирует окно по текущему состоянию SS: SP.

Follow позиционирует окно на слово, выделенное курсором.

Previous восстанавливает настройку окна стека в соответствии с установленным адресом до последнего действия, которое явно его изменило.

Change позволяет ввести новое значение для выделенного курсором слова. То же можно выполнить простым занесением нового значения.

Подготовка отладочных примеров

Ассемблерную программу нельзя отладить «умозрительно». Расстановка точек останова, просмотр значений переменных и другое связано с *внутрисегментными адресами размещения* команд и данных в сегменте, которые определяются по листингу трансляции.

Для каждого из возможных вариантов исполнения (их количество зависит от логики программы) следует подготовить отладочный пример.

Отладочный пример должен содержать:

- *сведения о размещении исходных данных*: имена регистра и/или адреса программных переменных в памяти в виде «сегмент: смещение»;
- *значения исходных и промежуточных (если это важно) переменных*: в удобной форме (например, в десятичном виде) и обязательно в форме представления Отладчиком – в hex-коде;
- *сведения о размещении и ожидаемые значения результатов*, аналогично предыдущим.

После загрузки исполняемая программа тщательно тестируется в пошаговом режиме по всем отладочным примерам. Только после того, как все варианты исполнения дадут *ожидаемые результаты*, можно говорить о работоспособности программы.

Важность хорошо подготовленных вручную отладочных примеров очень велика. Не стоит лениться потратить на них время – иначе реально потерянное на отладке программы время может быть неоправдано велико.

Иногда в желании сэкономить на подготовке отладочных примеров возникает соблазн «принять желаемое за действительное», то есть сначала увидеть некоторые результаты исполнения, а затем зафиксировать их в качестве своих ожидаемых прогнозов. Однако чудес не бывает. Процессор исполняет лишь то, что написано вами же. Так что, это прямой путь в полную безисходность.

Форма подготовки отладочных примеров может быть произвольной, но для экономии времени при отладке максимально четкой и ясной, например, в форме таблицы.

Для программы определения наибольшего из двух однобайтных переменных - ***primer.exe*** достаточно двух отладочных примеров.

Сведения для двух вариантов отладки сведены в таблицу.

		исходные		результат
	Переменная	**A**	**B**	**наибольшее**
	формат	*байт*	*байт*	*байт*
Размещение	в исходной программе	ds:a	ds:b	DH
	размещение в памяти	**ds:0000**	**ds:0001**	
1) A < B	dec	-6	1	1
	hex	**FA**	**01**	**01**
2) A > B	dec	5	-3	5
	hex	**05**	**FD**	**05**

Отладка с символическими адресами

Отладчик поддерживает работу с символическими именами, если информация о символических именах исходной программы была включена в исполняемый модуль.

Для этого трансляция исходного текста программы должна быть выполнена с ключем **/zi**, а компоновка – с ключом /v.

Включение отладочной информации существенно увеличивает объем исполняемого модуля. Например, исполняемый файл primer.exe без символьной информации для Отладчика имел размер 551 байт, при наличии такой информации – 993 байта.

Использовать символическую информацию в исполняемом модуле следует *только на этапе отладки*. После того, как программа отлажена, обязательно следует ее перетранслировать и перекомпоновать без указанных ключей.

Поскольку символическая отладка используется в основном для программ на языках высокого уровня, после загрузки исполняемого модуля на экране появляется окно с исходным текстом программы и окно отслеживания значений символьных переменных – *Watches* (рис.17).

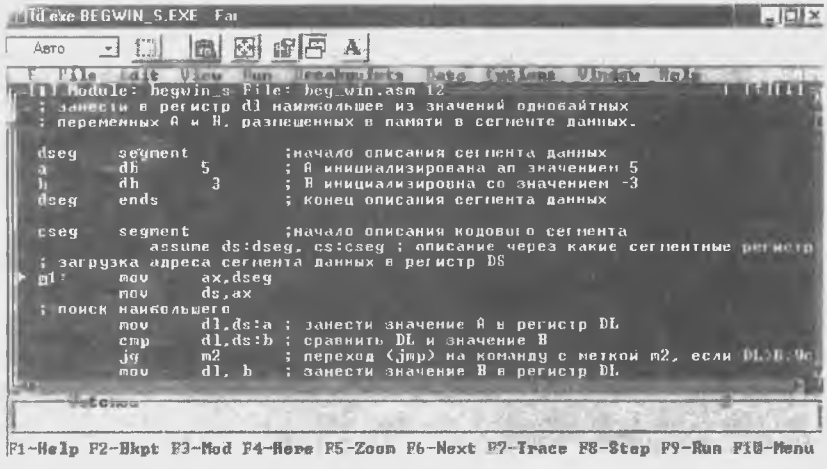


Рис.17 Окно исходного текста и окно переменных

Экран процессора, содержащий машинные коды команд и данных, значения регистров и флагов, открывается через пункт главного меню *View o CPU* (рис.18).

Рис.18 Экран процессора при символической отладке

Обратите внимание, что в командах окна *CPU* сохранены символические адреса. Текущая команда отмечена символом ▯ одновременно в обоих окнах - окне команд и в исходном тексте.

Переключение между окнами исходного текста, экраном процессора и окном переменных делается служебной клавишей F6 или мышью.

Для того, чтобы в исходном тексте Отладчик корректно отображал комментарии на русском языке, исходный текст программы должен создаваться в кодировке Win.

Перечень возможных манипуляций с переменными в окне *Watch* можно получить, находясь в этом окне, нажатием правой кнопки мыши или клавишей «*Ctrl*» (рис.19).

Основные действия - добавление переменных (*Watch*), редактирование (*Edit*), удаление имени из окна просмотра (*Remove*).

Рис.19 Меню окна *Watch*

Рис.20 Значение переменной A в окне *Watch*

На рис.20 показано отображение переменной A в окне просмотра значений переменных. Показан тип переменной – байт, ее значение в десятичном виде - 5 и hex-коде - 05h. Показан также символ, соответствующий ASCII-коду 05h.

Учебно-методическое издание

Ларина Татьяна Борисовна

Технология подготовки и отладки программ в Microsoft Macroassembler и Borland TurboAssembler

Методические указания

Подписано в печать - 19.09.06. Формат - 60×84/16 Тираж - 150.

Усл.печ.л. - 2,5. Заказ - 387.

Изд.№ 202-06.

127994, Москва, ул. Образцова, 15.

Типография МИИТ