Практико-ориентированное задание № 1

Этапы создания программы на ASSEMBLER

Цель работы

Изучение процесса разработки программы на ассемблере, создание исходного файла, объектного и загрузочного модулей программы. Изучение основных возможностей, отладчика TDEBUG.EXE.

Порядок выполнения работы

1. Создать рабочую папку для текстов программ на ассемблере и записать в нее файлы tasm.exe, tlink.exe, rtm.exe и td.exe. из пакета tasm, а также файл с исходным текстом программы на ассемблере, который необходимо сохранить с именем prog1.asm.
2. Создать загрузочный модуль, загрузить его в отладчик и выполнить программу в пошаговом режиме.
3. Изучить возможности отладчика tdebug, режимы просмотра регистров, файлов, памяти.

Теоретическая часть

1. Перед тем, как познакомиться с программированием на Турбо Ассемблере, вам нужно сделать следующее. Возьмите дистрибутивные диски Турбо Ассемблера и сделайте для каждого из них (с помощью утилиты DOS) рабочую копию.

2. Ввод текста программы

На этом этапе можно использовать любой текстовый редактор для ввода кода программы. При выборе редактора должно учитываться то, что он не должен вставлять специальных символов форматирования. Созданный с помощью текстового редактора файл должен иметь расширение **.asm.**

3. Общая структура программы на языке Ассемблер. Сегмент кода. Сегмент данных.

Программа, написанная на ассемблере TASM, может состоять из нескольких частей, называемых модулями, в каждом из которых могут быть определены один или несколько сегментов данных, стека и кода.

Любая законченная программа на ассемблере должна включать один главный, или основной (main), модуль, с которого начинается ее выполнение. Основной модуль может содержать программные сегменты, сегменты данных и стека, объявленные при помощи упрощенных директив. Кроме того, перед объявлением сегментов нужно указать модель памяти при помощи директивы .MODEL.

```
.model small      ; эта директива указывается до объявления сегментов
.stack 100h       ; размер стека 256 байт
.data             ; начало сегмента данных

    ;данные

.code             ; начало сегмент программ
    ;команды ассемблера
end main
```

При помощи следующих команд в сегментный регистр DS помещается адрес сегмента данных, указанного директивой .data:

mov AX, @data
mov DS, AX

“Классический” шаблон 32-разрядного приложения содержит область данных (определяемую директивой .data), область стека (директива .stack) и область программного кода (директива .code). Может случиться так, что 32-разрядному приложению на ассемблере потребуется несколько отдельных сегментов данных и/или кода.

.DATA (.data) определяет сегмент данных, в котором располагаются неинициализированные данные. Преимуществом директивы .DATA является то, что при ее использовании уменьшается размер исполняемого файла и, кроме того, обеспечивается лучшая совместимость с другими языками. Этой директиве должна предшествовать директива .MODEL.

.STACK (.stack) [размер] — определяет начало сегмента стека с указанным размером памяти, который должен быть выделен под область стека. Если параметр не указан, размер стека предполагается равным 1 Кбайт. При наличии предыдущего сегмента новый сегмент завершает его. Этой директиве должна предшествовать директива .MODEL.

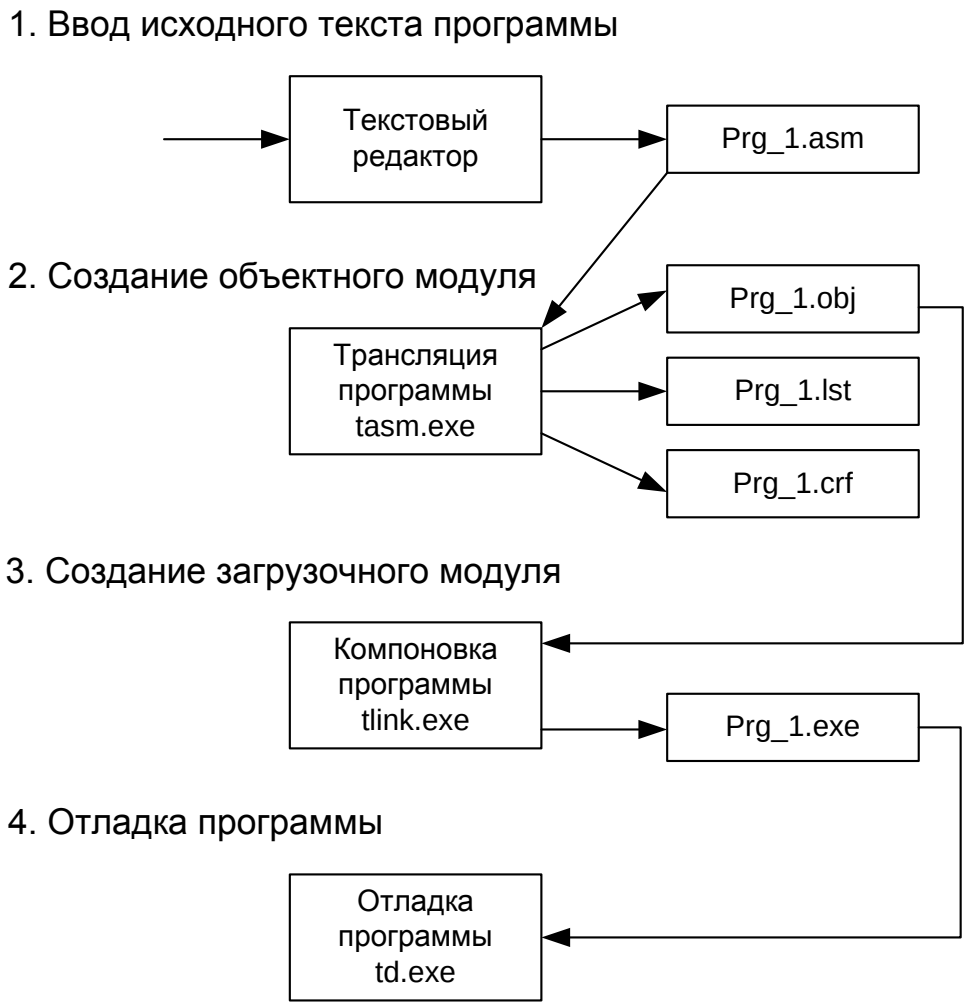
.CODE (.code) [имя] — определяет сегмент программного кода и заканчивает предыдущий сегмент, если таковой имеется. Необязательный параметр имя замещает имя _TEXT, заданное по умолчанию. Если имя не определено, ассемблер создает сегмент с именем _TEXT для моделей памяти tiny, small, compact и flat или сегмент с именем имя_модуля_TEXT для моделей памяти medium, large и huge. Этой директиве должна предшествовать директива .MODEL, указывающая модель памяти, используемую программой.

Prg_1.asm. Пример использования директив резервирования и инициализации данных

```
.model small
.stack 100h
.data
message db 'Запустите эту программу в отладчике','$'
perem_1    db    0ffh
perem_2    dw    3a7fh
perem_3    dd    0f54d567ah
mas        db    10 dup (' ')
pole_1     db    5 dup (?)
adr        dw    perem_3
adr_full   dd    perem_3
numbers    db    11, 34, 56, 23
fin        db    'Конец сегмента данных программы $'
.code
start:
      mov   ax,@data
      mov   ds,ax
      mov   ah,09h
      mov   dx,offset message
      int   21h
      mov   ah, 7h
      int   21h
      mov   ax,4c00h
      int   21h
end   start
```

4. Процесс разработки программы на Ассемблере

После того, как вы сохранили файл .ASM, вы захотите запустить программу. Однако, перед тем, как вы сможете ее запустить, вам потребуется преобразовать программу в выполняемый вид. Как показано на Рис. 1, где изображен полный цикл создания программы (редактирование, ассемблирование, компоновка и выполнение), это потребует двух дополнительных шагов - ассемблирования и компоновки.



5. Ассемблирование программы

На этапе **ассемблирования** ваш исходный код (текст программы) превращается в промежуточную форму, которая называется объектным модулем, а на этапе компоновки один или несколько модулей комбинируются в выполняемую программу. Ассемблирование и компоновку вы можете выполнять с помощью командной строки.

Для ассемблирования файла **имя файла.ASM** наберите команду:

tasm.exe [ключи] имя_исходного_файла [, имя_объектного_файла] [, имя_файла_листинга] [, имя_файла_перекрестных_ссылок]

Для получения справки по этой команде достаточно указать в командной строке tasm.exe без параметров. Квадратные скобки указывают на то, что параметр необязательный.
Нажмите клавишу **ENTER.**
Если вы не задали другое имя, файл **PRG_1.ASM** будет ассемблирован в файл **PRG_1.OBJ**. (Заметим, что расширение имени файла вводить не требуется. Турбо Ассемблер подразумевает в этом случае, что файл имеет расширение .ASM).

На экране вы увидите следующее:

```
Turbo Assembler Version 2.0 Copyright (C) 1990        (1)
by Borland International Inc.
 Assembling file: PRG_1.ASM                           (2)
 Error messages: None                                 (3)
 Warning messages: None                               (4)
 Remaining memory: 266K                               (5)
```

1 - Турбо Ассемблер, версия 2.0; авторские права фирмы Borland,1990 г.;
2 - ассемблирован файл **PRG_1.ASM**;
3 - сообщения об ошибках: **нет**;
4 - предупреждающие сообщения: **нет**;
5 - остается памяти: **266К**

Если вы введете файл **PRG_1.ASM** в точности так, как показано, то вы не получите никаких предупреждающих сообщений или сообщений об ошибках.

Если вы получаете такие сообщения, они появляются на экране наряду с номерами строк, указывающими строки, где содержатся ошибки.

При получении сообщений об ошибках проверьте исходный код (текст) программы и убедитесь, исправьте допущенные ошибки, а затем снова ассемблируйте программу.
Примечание. Воспользуйтесь файлом **«Ошибки ассемблера».**

6. Компоновка программы

После ассемблирования файла **PRG_1.ASM** вы продвинулись только на один шаг в процессе создания программы. Теперь, если вы скомпонуете только что полученный объектный код в выполняемый вид, вы сможете запустить программу.

Для компоновки программы используется программа **TLINK**, представляющая собой поставляемый вместе с Турбо Ассемблером компоновщик. Введите командную строку:

```
tlink.exe [ключи] список_объектных_файлов [, имя_загрузочного_модуля]
[, имя_файла_карты] [, имя_файла_библиотеки]
[, имя_файла_определений] [, имя_ресурсного_файла]
```

Список объектных файлов – обязательный параметр, содержащий список компонуемых файлов (или один файл) с расширением .obj. Файлы в списке должны быть разделены пробелами или знаком +.

Примечание. Здесь опять не требуется вводить расширение имени файла. TLINK по умолчанию предполагает, что этим расширением является расширение **.OBJ.** Когда компоновка завершится, компоновщик автоматически присвоит файлу с расширением **.EXE** имя, совпадающее с именем вашего объектного файла (если вы не определили другое имя). При успешной компоновке на экране появляется сообщение:

Turbo Linker Version 3.0 Copyright (c) 1987, 1990 by by Borland International Inc.

В процессе компоновки могут возникнуть ошибки (в данной программе это маловероятно). Если вы получили сообщения об ошибках компоновки (они выводятся на экран), измените исходный код программы, а затем снова выполните ассемблирование и компоновку.

7. Запуск вашей первой программы

Теперь программу можно запустить на выполнение. Для этого в ответ на подсказку операционной системы DOS введите **PRG_1** и нажмите **ENTER.** На экран выведется сообщение.

8. Особенности работы с Ассемблером для MS DOS

На компьютерах с 64-х разрядными операционными системами Windows 7, 8, 10 не удается запустить программы реального режима **MS DOS** из-за несовместимости. В этом случае используем эмулятор системы DOS – **DOSBox.** **DOSBox** является свободно распространяемой программой с открытым исходным кодом.

Для работы через **DOSBox** нужно использовать некоторые простые приемы, описанные ниже.

Предположим, что Ваши файлы хранятся на диске **D:\UCHEBA\ASM**.

Для работы с использованием DOSBox нужно выполнить следующее:

– запустить DOSBox с рабочего стола компьютера. На экране появятся два окна, одно из которых является окном состояния и служит для вывода служебных сообщений, а второе является рабочим. Окно состояния можно свернуть.

– создать локальный рабочий каталог с именем, совпадающим с именем диска, на котором находятся Ваши файлы. Для этого в командной строке рабочего окна DOSBox набираем**:**

z:\>mount D D:

После сообщения о том, что рабочий каталог создан, работаем из командной строки **DOSBox** как из обычной командной строки **MSDOS**:

z:\>D: ; перейти в корневой каталог диска D;

D:\>cd UCHEBA\ASM ; перейти в папку ASM, сделав ее текущей;

D:\UCHEBA\ASM>tasm/z/zi/n PRG_1 PRG_1 PRG_1; запустить; программу tasm.exe

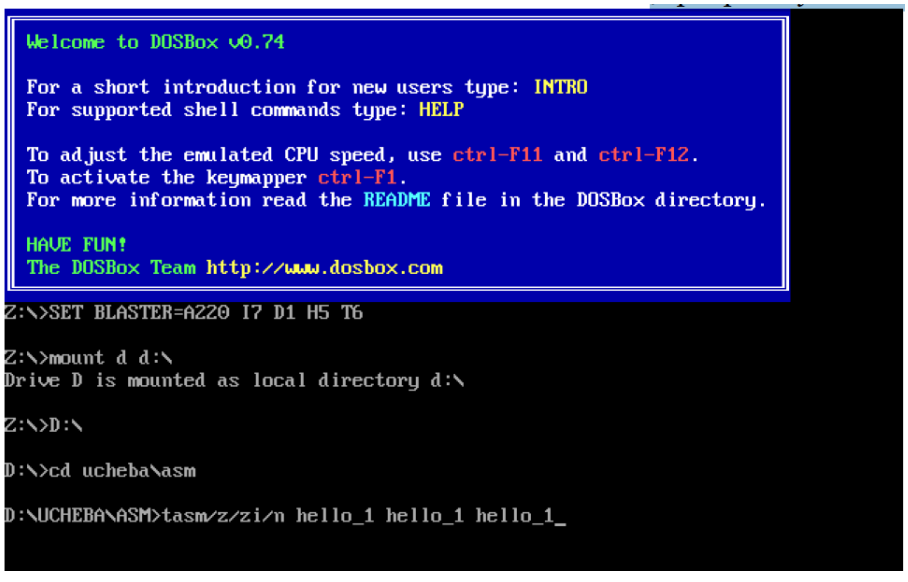


Рис. 1. – Начальное окно DOSBox и отображаемые в нем команды

DOSBox обладает достаточно большим набором возможностей, которые можно узнать набрав в командной строке

z:>intro.

Для получения сведений о командах DOS можно набрать **z:\>help**.

Следующие комбинации нажатий клавиш позволяют управлять режимами **DOSBox**:

ALT+Enter – переключение полный/уменьшенный экран;

CTRL+F1 – переназначение кнопок клавиатуры;

CTRL+F5 – сделать снимок экрана в виде файла *.png;

CTRL+F9 – закрыть программу;

Escape –закрыть графическое окно.

9. Работа с отладчиком Turbo Debugger (TD)

Tasm умеет ассемблировать синтаксически правильные программы, но не понимает, что, собственно, эта программа делает. Часто программа работает не так, как, по вашему мнению, должна была бы работать. В такой ситуации может помочь **TD-программа**, разработанная для поиска и исправления логических ошибок. Подобно всем отладчикам TD может работать в режиме супервизора, беря на себя управление программой в режиме пошагового исполнения, кода программы. Можно при этом изменять значения операндов в памяти, а также значения регистров и флагов. TD используется и в качестве учителя при изучении форматов машинных команд процессора в различных режимах адресации операндов.

Чтобы показать, как использовать TD при изучении языка ассемблера, исследуем программу **PRG_1** под управлением отладчика. Произведём заново ассемблирование и компоновку программы с опциями, которые добавляют отладочную информацию в obj- и exe-файлы:

tasm/zi[1] **PRG_1**.asm

tlink/v **PRG**[2]**_1**.obj

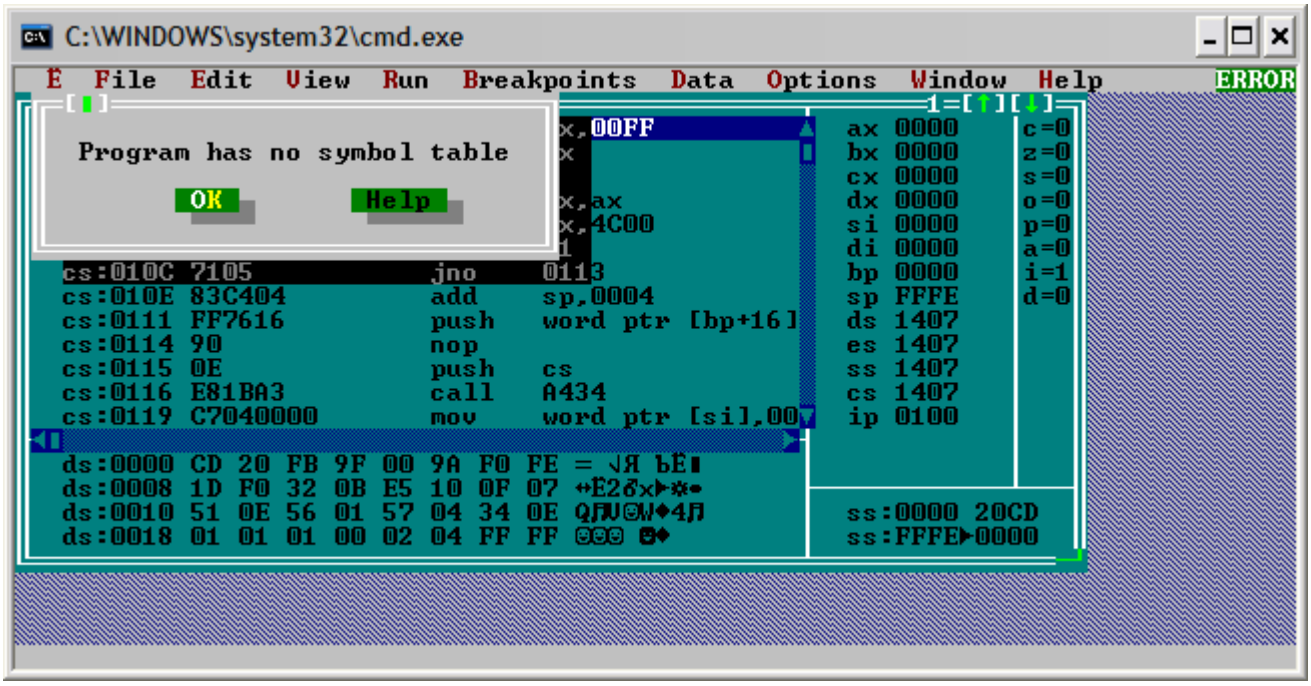
td **PRG_1**.exe

После запуска отладчика вы увидите примерно такое окно:

---

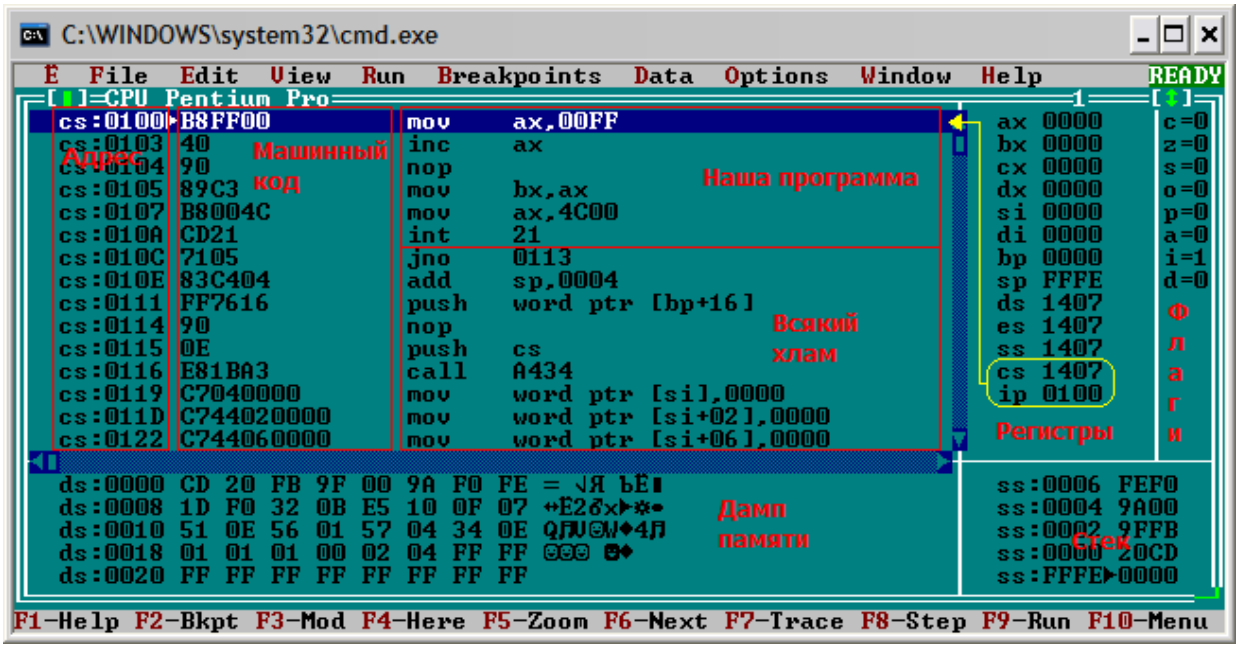
[1] *tasm/zi PRG_1* – команда добавляет в PRG_1.*obj* информацию, необходимую для использования отладчика *Turbo Debugger*

[2] *tlink/v PRG_1* – опция */v* включает в *PRG_1.exe* информацию, необходимую для последующего открытия этого файла в отладчике *Turbo Debugger*;



Сообщение означает, что в исполняемом файле нет специальных данных для отладки. Нажимаем ОК.

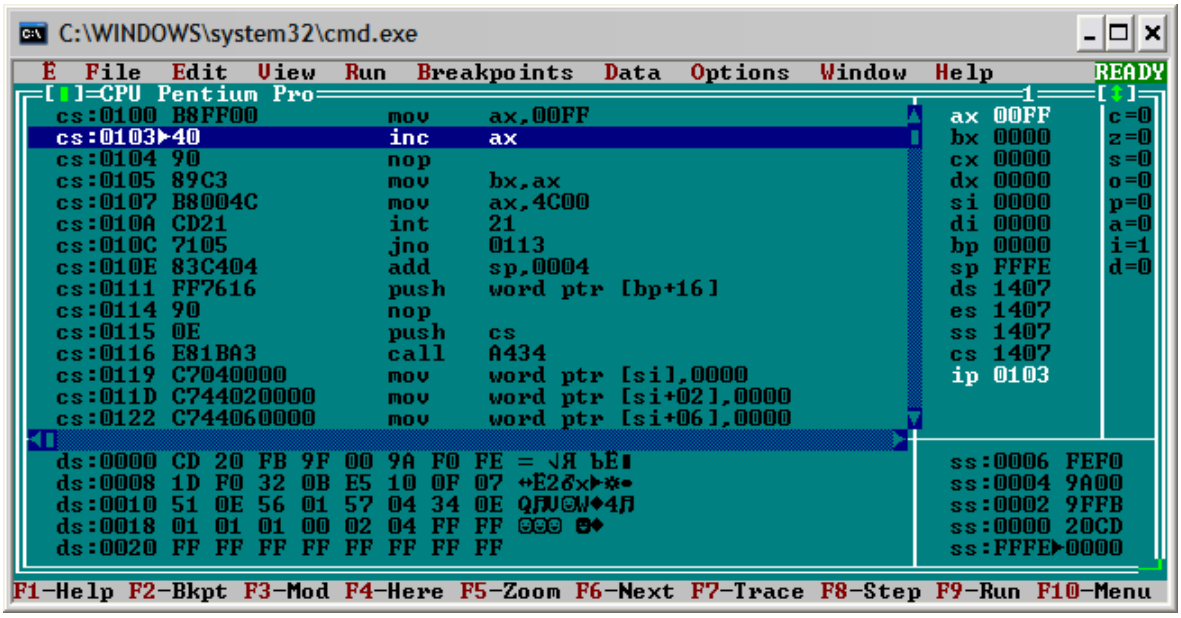
Turbo Debugger отображает окно CPU, в котором можно увидеть, как выполняется программа.



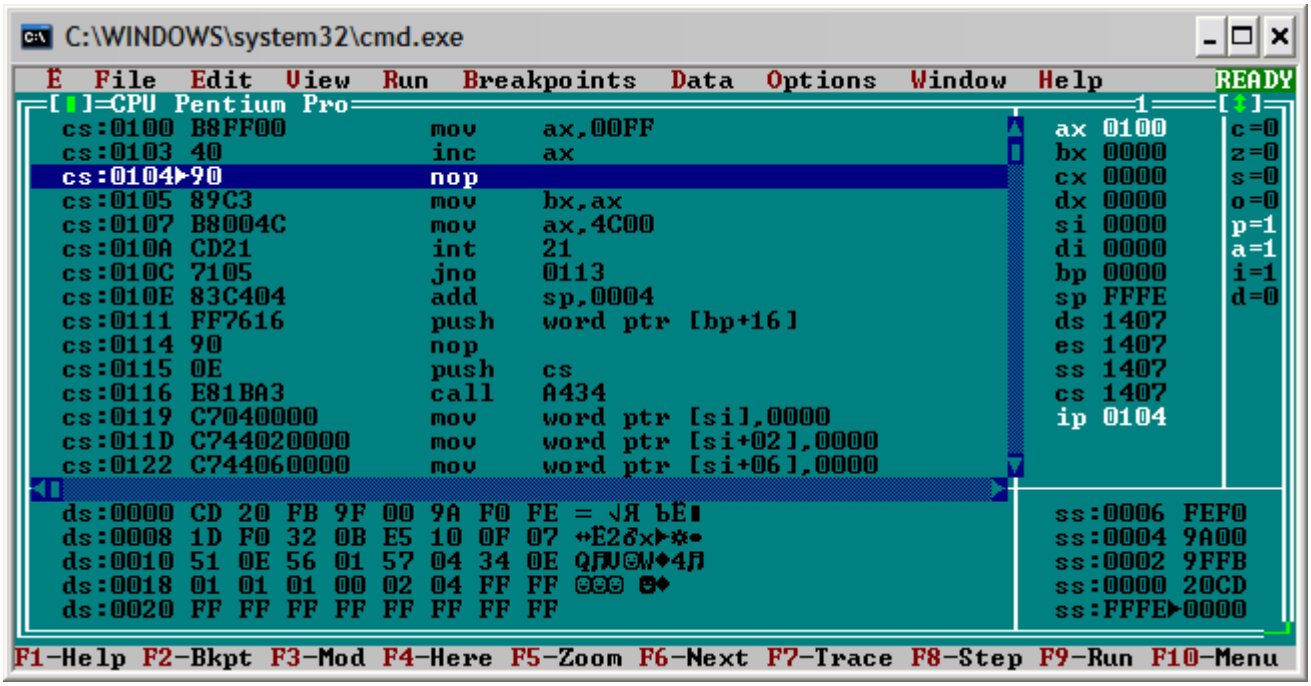
В большой области мы видим код нашей программы.

Самый левый столбец — Адреса, правее отображаются байты машинного кода, а ещё правее — символическое обозначение команд. Программа размещается в памяти, начиная с адреса 0100h в сегменте кода.
Сначала отражены машинные команды нашей программы, а за ними в памяти находится случайный мусор (точные значения неизвестны).
Обратите внимание, что отладчик показывает адреса и значения в шестнадцатеричном виде.
В правой части окна CPU отображаются регистры процессора и флаги.
В нижней части можно увидеть дамп области памяти и стек.

Стек — это специальная структура данных, с которой работают некоторые команды процессора.
Адрес текущей машинной команды определяется регистрами CS и IP, эта команда показана выделенной строкой и стрелкой.
Теперь нажмите **F8,** чтобы выполнить первую команду.



Теперь стрелка указывает на вторую команду. Изменившиеся регистры выделены белым цветом. Регистр AX теперь содержит значение 00FFh (то есть 255, чего мы и хотели от команды «mov ax,255»). Также изменилось значение регистра IP — оно увеличилось на размер выполненной машинной команды, а именно на 3. Теперь CS:IP указывает на следующую команду. Снова нажимаем F8.



Значение регистра AX увеличилось на 1 и стало равным 0100h (256). Значение IP тоже увеличилось на 1, потому что длина команды «inc ax» — 1 байт. Процессор выполняет программу последовательно, одну команду за другой. Ещё раз нажимаем F8.

После перехода со строки на строку с помощью нажатия F8 программа завершается.

Действия при работе с отладчиком:

1. В ответ на сообщение «Program has no symbol table» следует нажать клавишу **<Enter>;**
2. Загрузка программы командой **File-Open;**
3. Открыть "окно процессора" **View-CPU** или нажать **<Alt+V>;**
4. Переход из одной области в другую - **<Tab>;**
5. Увеличить размер кадра до размеров экрана, нажав **<F5>;**
6. Режим автоматического выполнения программы **<F9> (Run);**
7. Выполнение по шагам **<F8>** или **<F7>**. Отличие в назначении этих клавиш проявляется в том, что клавиша F7 используется для пошагового исполнения тела цикла, процедуры или программы обработки прерывания. Клавиша F8 исполняет эти процедуры как одну обычную команду и передает управление следующей команде программы;
8. Выполнение с установленными точками прерывания (***Breakpoints***). Перед исполнением программы необходимо установить эти точки, для чего следует перейти в нужную строку программы и нажать клавишу **F2** (***toggle***). Выбранная строка с контрольной точкой подсвечивается красным цветом. Чтобы убрать контрольную точку, надо повторить эту операцию снова. После установки точек прерывания программа запускается на исполнение клавишей **F9**. После первого нажатия клавиши *F9* программа остановится на первой точке прерывания, после второго – на второй точке и.т.д. Это очень удобный режим отладки программы, когда необходимо контролировать правильность её исполнения в некоторых характерных точках.
9. Если в процессе работы программа осуществляет вывод на экран, то просмотреть его можно, нажав **<Alt+F5>**. Возврат в окно отладчика осуществляется нажатием любой клавиши;
10. Просмотр памяти, нажать **<Ctrl+G>** и ввести в поле ввода начальный адрес интересуемой области памяти (сегмента данных) DS:0. В зоне появится содержимое сегмента данных;
11. Поиск необходимой инструкции или содержимого ячейки памяти <**Ctrl+S>;**
12. Завершение работы с отладчиком **<Alt+X>** или командой **File – Quit.**

10. Изменение содержимого регистров

Это дает возможность исправлять обнаруженные ошибки (если выяснилось, что в какой-то строке программы заполняется не тот регистр или не тем содержимым), а также динамически модифицировать программу с целью изучения ее работы.

11. Задание на выполнение работы

1. Используя текстовый редактор, создать и отредактировать исходный модуль программы **PRG_1.asm**, текст которого приведен ниже.

Prg_1.asm. Пример использования директив резервирования и инициализации данных

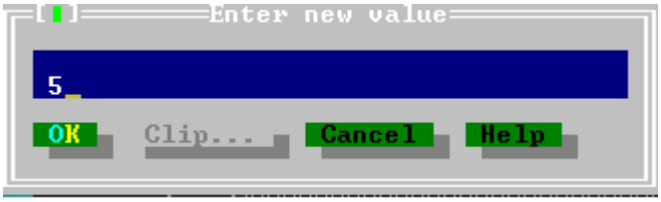
```
.model small
.stack 100h
.data
message db 'Запустите эту программу в отладчике','$'
perem_1    db    0ffh
perem_2    dw    3a7fh
perem_3    dd    0f54d567ah
mas        db    10 dup (' ')
pole_1     db    5 dup (?)
adr        dw    perem_3
adr_full   dd    perem_3
numbers    db    11, 34, 56, 23
fin        db    'Конец сегмента данных программы $'
.code
start:
     mov   ax,@data
     mov   ds,ax
     mov   ah,09h
     mov   dx,offset message
     int   21h
     mov   ah, 7h
     int   21h
     mov   ax,4c00h
     int   21h
end   start
```

2. Используя компилятор Турбо Ассемблера **tasm.exe** создать файл **PRG_1.obj**.
3. Убедиться в работоспособности программы **PRG_1**, запустив ее из командной строки
4. Запустите программу в отладчике. Выполните программу до команды **int 21h** и просмотрите содержимое регистров процессора.
   - Вы должны увидеть, что в старшей половине регистра AX находится число 09h - номер вызываемой функции DOS. Младшая половина регистра AX заполнена остатком от выполнения последней операции с регистром AX.
   - В регистре DX будет 0000h - относительный адрес первого байта строки message в сегменте команд.
   - Изменим этот относительный адрес. Для этого надо

     1) перейти в окно регистров, поместить курсор на строку, отображающую содержимое регистра DX, и ввести команду Alt+F10, открывающую внутреннее меню окна регистров

Increment, Decrement,- уменьшить, увеличить значение регистра на 1
Zero – обнулить регистр;
Change – заменить значение в регистре, на любое заданное значение

     2) Выбрав пункт Change, занесем **в регистр DX число 5**;



3) выполнить очередную команду (int 21h), DOS выведет на экран строку, начало которой расположено в байте 5 сегмента данных.

В нашей фразе "Запустите эту программу в отладчике» байт 5 приходится на букву **т** (нумерация байтов в строке, естественно, начинается с нуля).

В результате на экран будет выведена строка «тите эту программу в отладчике»;

5. Внести изменения в программу **PRG_1**, которые заставят ее выводить на экран еще две строки символов: «My name is Family name» и «My group IUK7». Для этого создайте новый исходный модуль **PRG_2.asm,** выполните ассемблирование и компоновку, после чего убедитесь в работоспособности программы.