

# Архитектура ЭВМ и систем

## *Лекция № 6*

### Введение в программирование на языке ассемблера

# План лекции

1. Структура программы на языке ассемблера.
  - 1.1. Общий формат ассемблерной команды.
  - 1.2. Определение данных.
3. Процесс ассемблирования и выполнения программы.
  - 3.1. Получение исполняемого файла.
  - 3.2. Работа в отладчике TD.

# Структура программы на языке ассемблера

Программа на языке ассемблера представляет собой совокупность блоков памяти – **сегментов**.

Каждый сегмент содержит совокупность предложений языка, каждое из которых занимает отдельную строку кода программы.

Можно выделить четыре типа предложений языка ассемблера:

1. *Команды или инструкции*, представляющие собой символические аналоги машинных команд.

В процессе трансляции инструкции ассемблера преобразуются в соответствующие команды системы команд микропроцессора.

# Структура программы на языке ассемблера

2. *Макрокоманды* – оформляемые определенным образом предложения текста программы, замещаемые во время трансляции другими предложениями.

3. *Директивы*, являющиеся указанием транслятору ассемблера на выполнение некоторых действий. У директив нет аналогов в машинном представлении.

4. *Комментарии*, содержащие любые символы, в том числе и буквы русского алфавита. Комментарии игнорируются транслятором.

# Общий формат ассемблерной команды

**Метка: Мнемоника Операнд, Операнд; Комментарий**

*Метка (символьный адрес команды в сегменте кода)* представляет собой идентификатор. Символы метки могут разделяться знаком подчеркивания. Символьный адрес команды является необязательным. Он указывается в команде тогда, когда на неё ссылаются в командах условного или безусловного переходов.

*Команда (мнемоника)* указывает транслятору с ассемблера, какое действие должен выполнить данный оператор.

# Общий формат ассемблерной команды

*Операнды* – это регистры, метки (адреса) данных или непосредственно данные.

*Комментарий* служит для пояснения действий команды или директивы ассемблера. После точки с запятой комментарий записывается на одной строке. Комментарий является необязательным.

*Примеры:*

**add ah, al;** Сложение содержимого регистров *ah* и *al*

**met: inc ah;** Увеличение содержимого регистра *ah* на 1

**mov ah, 4;** Пересылка в регистр *ah* значения 4

**inc cx;** Увеличение содержимого регистра *cx* на 1

# Определение данных

В сегменте данных можно зарезервировать определённое количество байтов для размещения исходных данных или результатов работы программы, а также указать их начальные значения.

**Метка Мнемоника Операнд,...,Операнд ; Комментарий**

*Метка (символьный адрес)* обозначает смещение (номер ячейки в сегменте данных) первого резервируемого байта. Метка в сегменте данных обладает теми же свойствами, что и метка в сегменте кода, но является обязательной.

# Определение данных

*Мнемоника* определяет длину каждого операнда:

1) **DB** (определить байт). Диапазон для целых чисел без знака: 0...255, для целых чисел со знаком: -128...127.

2) **DW** (определить слово – два байта). Диапазон для целых чисел без знака: 0...65535, для целых чисел со знаком: -32768...32767.

3) **DD** (определить двойное слово – четыре байта).  
Диапазон для целых чисел без знака: 0...4294967295, для целых чисел со знаком: -2147483648...2147483647.



# Определение данных

*Операнды* показывают инициализируемые данные или объем резервируемого пространства. Выражение может содержать константу или символ ? для неопределенного значения.

При определении большого числа ячеек можно применять оператор повторения

**DUP (Операнд,...,Операнд).**

Операнды могут задаваться в различных системах счисления: для двоичной системы счисления после значения операнда ставится символ *B*, для шестнадцатеричной – символ *H*, для десятичной – ничего.

# Определение данных

*Примеры:*

**Data\_word DW 100H** – по адресу *Data\_word* зарезервировано 2 байта с первоначальным значением *100* в шестнадцатеричной системе счисления (или *256* в десятичной).

**Arr\_DW DW 4 DUP(?)** – определяет по адресу *Arr\_DW* 4 двухбайтовые ячейки, содержащих произвольную информацию.

**Data\_byte DB -104** – по адресу *Data\_byte* зарезервирован 1 байт с первоначальным значением *-104* в десятичной системе счисления.

**Data\_str DB 'H','E','L','L','O'** – по адресу *Data\_str* зарезервировано 5 однобайтовых ячеек, содержащих символы.

# Пример программы

**s\_s segment** stack "stack" ;адрес начала сегмента стека  
dw 12 dup(?) ;зарезервировано в памяти 24 ячейки с любым значением  
**s\_s ends** ;адрес конца сегмента стека  
**d\_s segment** ;адрес начала сегмента данных  
aa dw 7145h, 23h ;данные, т.е. числа 7145h и 23h записаны по адресу  
;aa и aa+2 соответственно, они определены как слова  
sum dw 0 ; данное, т.е. число 0 записано по адресу sum  
**d\_s ends** ;адрес конца сегмента данных  
**c\_s segment** ;начало сегмента кода  
**assume** ss:s\_s,ds:d\_s,cs:c\_s ;директива, определяющая, каким сегментным  
; регистрам соответствуют назначенные  
;метками адреса начала сегментов  
**begin:** ;начало программы  
mov ax,d\_s ;пересылка в регистр сегмента данных (ds) адреса  
;начала сегмента данных, т.е. метки начала сегмента данных  
mov ds,ax ;пересылку в ds можно сделать только через промежуточную  
;пересылку в рабочий регистр, например ax, т.к. из памяти в  
;сегментные регистры записывать нельзя

# Пример программы

<b>mov ax,aa</b>	;пересылка в регистр <u>ax</u> содержимого, находящегося ;по адресу <u>aa</u> , т.е. числа <u>7145h</u>
<b>add ax,aa+2</b>	;сложить число, которое находится в регистре <u>ax</u> с ;содержимым, находящимся по адресу <u>aa+2</u> ;т.е.числа <u>23h</u> и результат записывается в регистр <u>ax</u>
<b>mov sum,ax</b>	;переслать содержимое <u>ax</u> , т.е. результат в ячейку ;памяти по адресу <u>sum</u>
<b>mov ah,4ch</b>	;для правильного завершения программы ;необходимо переслать в регистр <u>ah</u> - байт <u>4ch</u>
<b>int 21h</b>	;и вызвать прерывание равное <u>21h</u>
<b>c_s ends</b>	;конец сегмента кода
<b>end begin</b>	; конец программы. ; Метки начала и конца программы должны совпадать.

**Примечание:** для операционных систем типа UNIX данная программа будет выглядеть точно также, только системный вызов будет *int 80h* вместо *int 21h*.

# Процесс ассемблирования и выполнения программы

Для работы удобно использовать:

- простой компилятор ассемблера – *TASM*;
- текстовый редактор *Блокнот*;
- виртуальную машину *DOSBox*.

Дистрибутив DOSBox устанавливается на компьютер.

Для Windows обычно скачивается дистрибутив в виде EXE. – файла и устанавливается стандартным образом.

Для UNIX, например, Ubuntu, дистрибутив можно найти в Центре приложений или установить командой:

**sudo apt install dosbox**

# Процесс ассемблирования и выполнения программы

Перечень файлов пакета TASM:

- 1) DPMILOAD.EXE;
- 2) DPMIMEM.DLL;
- 3) DPMI16BI.OVL;
- 4) TASM.EXE;
- 5) TLINK.EXE;
- 6) TD.EXE;
- 7) TDHELP.TDH – не обязательно.

# Процесс ассемблирования и выполнения программы

Для получения исполняемого файла (с расширением .EXE) необходимо выполнить следующие действия:

1. Создать в Блокноте исходную программу на языке ассемблера, и сохранить её как файл с расширением *.ASM* в рабочую директорию своего компьютера.

2. Запустить эмулятор DOSBox.

Для Windows: выбрать соответствующую команду в меню *Пуск* или ярлык на рабочем столе компьютера.

Для UNIX: выполнить команду **dosbox** в терминале.

# Процесс ассемблирования и выполнения программы

3. Связать диск *C:* или *D* в ОС MS DOS с папкой Windows (или другой ОС), в которой расположены файлы пакета TASM и исходная программа. Желательно, чтобы перечисленные выше семь файлов и исходный файл *.ASM* находились в одной папке. Например, если пакет TASM находится на *F:\OKS\ASM\*, то в строке приглашения необходимо указать следующее:

**MOUNT D: F:\OKS\ASM\**

Для UNIX: создаем папку в домашней директории и прописываем к ней путь. Например, если пакет TASM находится в папке *ASM* *домашней директории*, то в строке приглашения необходимо указать следующее:

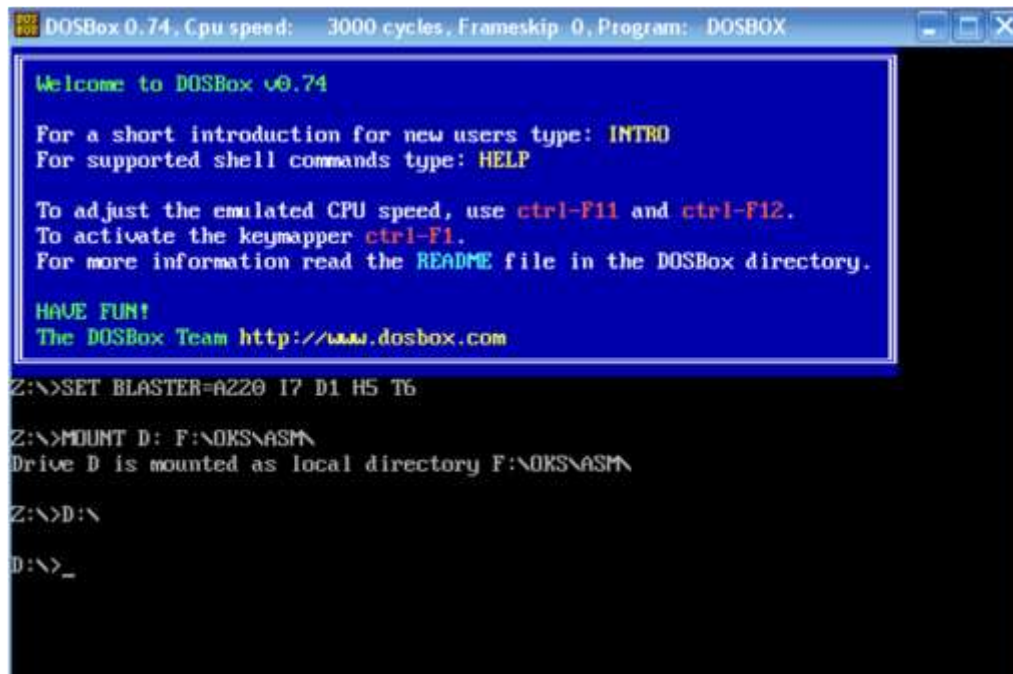
**MOUNT C: ~/ASM**



# Процесс ассемблирования и выполнения программы

4. Перейти на диск *C:* или *D:* в MS DOS, что равносильно переходу в соответствующую папку Windows или UNIX, указанную в команде *MOUNT*.

При необходимости с помощью команды **CD Имя\_папки** можно перейти к директории с нужными файлами.



```
DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip: 0, Program: DOSBOX

Welcome to DOSBox v0.74

For a short introduction for new users type: INTRO
For supported shell commands type: HELP

To adjust the emulated CPU speed, use ctrl-F11 and ctrl-F12.
To activate the keymapper ctrl-F1.
For more information read the README file in the DOSBox directory.

HAVE FUN!
The DOSBox Team http://www.dosbox.com

Z:\>SET BLASTER=A220 I7 D1 H5 T6

Z:\>MOUNT D: F:\DOS\ASM
Drive D is mounted as local directory F:\DOS\ASM

Z:\>D:\
D:\>_
```

Для ОС UNIX можно русифицировать консоль, чтобы русский язык нормально отображался. В командной строке необходимо ввести команду: **KEYB RU**.

# Процесс ассемблирования и выполнения программы

5. Странслировать исходный файл с расширением .ASM путём ввода в командной строке следующей команды:

**TASM \Путь\Имя файла.ASM /Z**

Например, если надо странслировать файл MY.ASM, то в команде указывается имя этого файла.



```
DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip: 0, Program: DOSBOX
To adjust the emulated CPU speed, use ctrl-F11 and ctrl-F12.
To activate the keymapper ctrl-F1.
For more information read the README file in the DOSBox directory.

HAVE FUN!
The DOSBox Team http://www.dosbox.com

Z:\>SET BLASTER=A220 I7 D1 H5 T6

Z:\>MOUNT D: F:\DOS\ASM\
Drive D is mounted as local directory F:\DOS\ASM\

Z:\>D:\>
D:\>TASM MY.ASM/Z
Turbo Assembler Version 3.2 Copyright (c) 1988, 1992 Borland International

Assembling file: MY.ASM
Error messages: None
Warning messages: None
Passes: 1
Remaining memory: 470k

D:\>
```

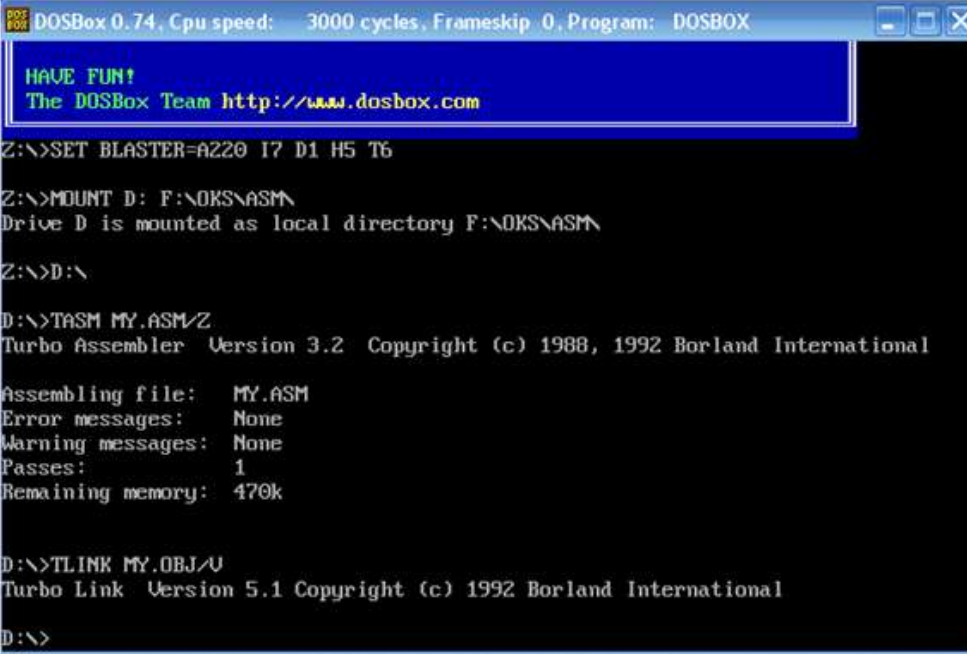
Результатом работы транслятора в случае отсутствия ошибок будет файл с расширением *.OBJ* – объектный модуль и тем же именем, что и исходный файл. Иначе на экране появится перечень ошибок с указанием их типа и местоположения.

*Пример трансляции файла MY.ASM*

# Процесс ассемблирования и выполнения программы

6. Странслированный без ошибок файл необходимо обработать компоновщиком (загрузчиком), т.е. набрать в командной строке следующую команду:

**TLINK \Путь\ Имя файла.OBJ /V**



```
DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: DOSBOX

HAVE FUN!
The DOSBox Team http://www.dosbox.com

Z:\>SET BLASTER=A220 I7 D1 H5 T6

Z:\>MOUNT D: F:\OKS\ASM\
Drive D is mounted as local directory F:\OKS\ASM\

Z:\>D:\>

D:\>TASM MY.ASM/Z
Turbo Assembler Version 3.2 Copyright (c) 1988, 1992 Borland International

Assembling file: MY.ASM
Error messages: None
Warning messages: None
Passes: 1
Remaining memory: 470k

D:\>TLINK MY.OBJ/V
Turbo Link Version 5.1 Copyright (c) 1992 Borland International

D:\>
```

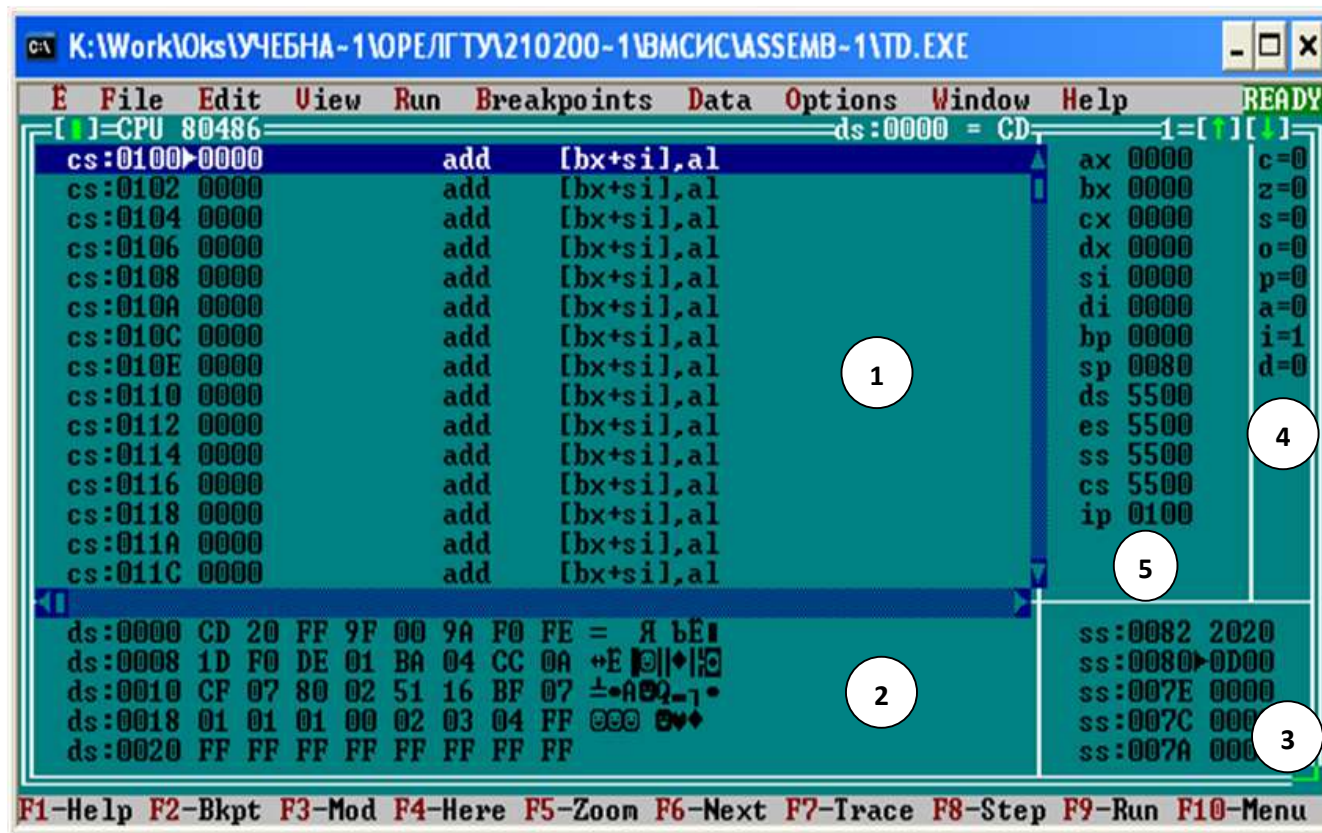
Результатом при отсутствии ошибок будет файл с расширением **.EXE** – исполняемый файл, и именем, совпадающим с именем исходного файла.

*Пример компоновки файла MY.OBJ*

# Процесс ассемблирования и выполнения программы

7. Для запуска отладчика необходимо набрать в командной строке команду:

**TD \Путь\ Имя файла.EXE**



Внешний вид окна TD

# Процесс ассемблирования и выполнения программы

Отладчик TD позволяет по шагам проследить процесс выполнения программы на уровне регистров процессора и ячеек памяти.

Значения некоторых функциональных клавиш:

1) **F7** – трассировка программы.

2) **F8** – выполнение программы по шагам т.е. по программе перемещается полоса выбора (синяя), и будет выполнена та команда, на которой эта полоса размещена.

После выполнения команды на экране появляется содержимое регистров, флагов и адрес следующей на очереди команды (соответствующие регистры подсвечиваются белым цветом).

3) **F10** – выход в главное, верхнее меню.

Запускаются команды или с помощью мыши или с помощью клавиш перемещения курсора на клавиатуре.

# Процесс ассемблирования и выполнения программы

Выбор группы верхнего меню также может выполняться с помощью мыши или с клавиатуры (ALT+ горячая клавиши соответствующей группы).

В верхнем меню по команде *FILE* можно открыть любой файл. По команде *VIEW* появляется еще меню, в котором находится команда *DUMP* – команда получения содержимого памяти по соответствующему адресу заданному в регистре *DS*, т.е. содержимое данных определенных в нашей программе. Данные начинаются с нулевого относительного адреса. Эти данные можно изменять.

*REGISTERS* – после запуска этой команды появляется окно с регистрами, и данные, находящиеся в этих регистрах, можно изменять.

Выход из отладчика по нажатию *ALT+X*.

Выход из любой команды по нажатию клавиши *ESC*.