

## Парсинг программы

### Пример lab\_3\_1

#### 1.asm

// Объявили, что идентификатор output\_X с типом near (метка ближнего перехода) будет объявлен в другом модуле

```
EXTRN output_X: near
```

// Объявили сегмент с названием STK, выравниванием по параграфам (/ 16), типом stack и классом 'STACK'

```
STK SEGMENT PARA STACK 'STACK'
```

```
    db 100 dup(0)
```

// выделили 100 байт, заполнили 0

```
STK ENDS
```

// Объявили сегмент с названием DSEG, выравниванием по параграфам (/ 16), типом PUBLIC<sup>1</sup> и классом 'DATA'

```
DSEG SEGMENT PARA PUBLIC 'DATA'
```

```
    X db 'R'
```

// Выделили 1 байт, положили туда R и связали с этой ячейкой метку X

```
DSEG ENDS
```

// Объявили сегмент с названием CSEG, выравниванием по параграфам (/ 16), типом PUBLIC и классом 'CODE'

```
CSEG SEGMENT PARA PUBLIC 'CODE'
```

```
    assume CS:CSEG, DS:DSEG, SS:STK // связка регистров с сегментами
```

```
main:
```

```
    mov ax, DSEG
```

```
    mov ds, ax
```

// связали регистр ds с сегментом DSEG через регистр ax

```
    call output_X
```

// вызвали метку другого модуля

```
    mov ax, 4c00h2
```

// вызвали завершение

```
    int 21h
```

```
CSEG ENDS
```

```
PUBLIC X
```

// Сделали X доступной для других модулей

```
END main
```

// Закрыли сегмент и объявили, что выполнение программы начнется с метки main

---

<sup>1</sup> PUBLIC – сегменты с одним именем располагаются в памяти друг за другом, независимо от того, в каком порядке они были объявлены в исходном коде)

<sup>2</sup> ax, 4c00h – в результате мы в ah запишем 4c, в al – 00. 4c в ah нужно для прерывания и завершения программы, а 0 в al будет “кодом возврата” (типа EXIT\_SUCCESS)

## 2.asm

```
// Сделали метку output_X доступной для других модулей
PUBLIC output_X
// Объявили, что метка X с типом byte будет взята из другого модуля
EXTRN X: byte

// Объявили сегмент DS2, с выравниванием PARA (по умолчанию), и типом AT (то есть этот сегмент всегда
будет начинаться в памяти с фиксированного адреса 0b800h)
DS2 SEGMENT AT 0b800h
    CA LABEL byte      // Объявили метку CA с размером памяти byte (память не выделяется)
                        // с этого момента отступ в сегменте будет идти с переданного значения
    ORG 80 * 2 * 2 + 2 * 2
    SYMB LABEL word    // Объявили метку SYMB с размером памяти word (2 байта)
DS2 ENDS

// Объявили сегмент кода
CSEG SEGMENT PARA PUBLIC 'CODE'
    assume CS:CSEG, ES:DS2 // Связали соответствующие регистры с сегментами

// Объявили процедуру output_X
output_X proc near
    mov ax, DS2
    mov es, ax           // Связали регистр es и сегмент DS2
    mov ah, 10           // Хотим считать строку в буфер
    mov al, X            // Видимо X – это подготовленный буфер, вот мы его в al и кладем
    mov symb, ax         // Зачем-то положили содержимое ax в буфер symb
    ret                  // Передача управления вызывающей стороне
output_X endp           // Конец функции
CSEG ENDS               // Конец сегмента
END                      // Конец модуля
```

## Вывод

Выполнение программы начнется с метки `main` (см файл 1). Внутри этой метки будет вызвана процедура `output_X`, которая предназначена для подготовки считывания в `X` строки символов с клавиатуры (она только раскидывает нужные значения по регистрам, но не вызывает прерываний. еще один интересный факт, что судя по тому, что мы выделяли под `X` один байт, то больше чем один символ мы не считаем). После возврата в `main` будет вызвано завершение программы.

Соответственно в результате у нас не изменится значение `X`.

```
C:\>LR03_1_1.EXE
```

Никакого взаимодействия с пользователем нет, что подтверждается на практике.

## Пример lab\_3\_2

### 1.asm

// Выделили под стек 100 байт, заполненных 0

```
STK SEGMENT para STACK 'STACK'
```

```
    db 100 dup(0)
```

```
STK ENDS
```

// В сегменте данных объявили переменную W размером 2 байта, в которую положили два символа 4 и D (так как 34 в ASCII – 4, а 44 – D)

// Сегмент будет иметь типа common, то есть сегменты с таким именем будут накладываться друг на друга.

```
SD1 SEGMENT para common 'DATA'
```

```
    W dw 3444h
```

```
SD1 ENDS
```

```
END
```

### 2.asm

// Объявили еще один сегмент с данными

```
SD1 SEGMENT para common 'DATA'
```

```
    C1 LABEL byte           // Объявили метку C1
```

```
    ORG 1h                  // Сделали выравнивание 1
```

```
    C2 LABEL byte           // Объявили метку C2
```

```
SD1 ENDS
```

// Пока я думаю, что сегмент из верхнего файла будет главнее (то есть в размер объединения будет определяться по нему, так как только в нем есть директива выделения памяти)

// Сегмент кода со связыванием регистров и меткой main

```
CSEG SEGMENT para 'CODE'
```

```
    ASSUME CS:CSEG, DS:SD1
```

```
main:
```

```
    mov ax, SD1
```

```
    mov ds, ax             // связка регистра ds и сегмента SD1
```

```
    mov ah, 2
```

```
    mov dl, C1             // пытаемся вывести то, что лежит в метке C1
```

```
    int 21h
```

```
    mov dl, C2             // пытаемся вывести то, что лежит в метке C2
```

```
    int 21h
```

```
    mov ax, 4c00h         // завершаем программу
```

```
    int 21h
```

```
CSEG ENDS                 // закрыли сегмент
```

```
END main                  // закрыли модуль с объявлением начала программы с метки main
```

## Вывод

Программа должна будет вывести значения C1 и C2. Возможно из-за `common` там окажется то, что мы положили в переменную `w`.

Результат превзошел все мои ожидания, конечно..

```
C:\>LR03_2_1.EXE
D4
```

Если не ставить выравнивание org 1h

```
C:\>LR03_2_1.EXE
DD
```

То есть отсюда понятно, откуда берется 4. Мы поставили выравнивание с 1 и он взял первое значение.

Вопрос откуда D?

То ли пишет он в обратном порядке...

```

SS:0000  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  .....
SS:0070  44 34 00 00 00 00 00 00  00 00 00 00 00 00 00 00  D4..

```

Вот это надо у Лехи спросить

Судя по этой прекрасной строчке значения в память у нас пишутся в обратном порядке

```
W dd 33344344h SS:0000 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
SS:0070 44 43 34 33 00 00 00 00 00 00 00 00 00 00 00 00 DC43...
```

### Пример lab\_3\_3

// Объявили сегмент с данными

```
SD1 SEGMENT para public 'DATA'
```

```
    S1 db 'Y' // Создали переменную 1 байт, значение Y
```

```
    db 65535 - 23 dup (0) // Выделили 65533 байт заполненных 0
```

```
SD1 ENDS
```

// Объявили еще один сегмент с данными

```
SD2 SEGMENT para public 'DATA'
```

```
    S2 db 'E' // Создали переменную 1 байт, значение E
```

```
    db 65535 - 2 dup (0) // Выделили 65533 байт заполненных 0
```

```
SD2 ENDS
```

// Где-то я это уже видела

```
SD3 SEGMENT para public 'DATA'
```

```
    S3 db 'S'
```

```
    db 65535 - 2 dup (0)
```

```
SD3 ENDS
```

// Сегмент кода со связыванием регистров

```
CSEG SEGMENT para public 'CODE'
```

```
    assume CS:CSEG, DS:SD1
```

output:

```
    mov ah, 2
```

```
    int 21h // Вывели то, что сейчас лежит в регистре dl
```

```
    mov dl, 13
```

```
    int 21h // Вывели символ с номером 13 (поместить курсор в начало)
```

```
    mov dl, 10
```

```
    int 21h // Вывели символ с номером 10 (перевод на новую строку)
```

```
    ret // Вернули управление вызывающей стороне
```

main:

```
    mov ax, SD1
```

```
    mov ds, ax // Связка регистров
```

```
    mov dl, S1 // Положили в dl значение S1
```

```
    call output // Вызывали подпрограмму
```

```
assume DS:SD2 // Объявили о новом связывании
```

```
    mov ax, SD2
```

```
    mov ds, ax
```

```
    mov dl, S2 // Положили в dl значение S2
```

```
    call output // Вызывали подпрограмму
```

---

<sup>3</sup> Вообще максимальный размер сегмента 64 Кб, что равно 65536 байт. Здесь мы вроде не переходим за пределы сегмента (то есть выделяем 65533 + 1 для переменной)

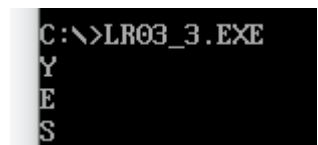
```

assume DS:SD3
    mov ax, SD3
    mov ds, ax
    mov dl, S3
    call output           // То же самое
    mov ax, 4c00h         // Завершились
    int 21h
CSEG ENDS
END main                 // Выполнение программы начнется с main

```

### Вывод

Программа выведет на экран значения Y\nE\nS\n.



C:\>LR03\_3.EXE

Y  
E  
S

ЧТД

## Пример lab\_3\_4

1.asm

// X будет доступен другим модулям

PUBLIC X

// Тырим из других модулей метку exit с далеким связыванием

EXTRN exit: far

// Выделили под стек 100 байт

SSTK SEGMENT para STACK 'STACK'

db 100 dup(0)

SSTK ENDS

// Объявили переменную X в один байт, записали в нее X

SD1 SEGMENT para public 'DATA'

X db 'X'

SD1 ENDS

// Объявили сегмент кода, соответствующее связывание для сегментов и регистров

SC1 SEGMENT para public 'CODE'

assume CS:SC1, DS:SD1

main:

jmp exit // переход на метку exit

SC1 ENDS

END main // Выполнение программы начнется с метки main

2.asm

EXTRN X: byte

PUBLIC exit

SD2 SEGMENT para 'DATA'

Y db 'Y'

SD2 ENDS

SC2 SEGMENT para public 'CODE'

assume CS:SC2, DS:SD2

exit:

mov ax, seg X // команда seg вернет адрес сегмента, где расположена переменная

mov es, ax // Связка сегмента и регистра

mov bh, es:X // такой вызов переменной, так как мы работаем без assume

mov ax, SD2

mov ds, ax

```

    xchg ah, Y      // Поменять значения местами
    xchg ah, ES:X
    xchg ah, Y      // В итоге вроде как в X будет лежать 'Y', а в Y будет лежать 'X'

    mov ah, 2
    mov dl, Y
    int 21h         // Вывели Y

    mov ax, 4c00h
    int 21h
SC2 ENDS
END

```

### Вывод

Программа поменяет местами значения переменных X и Y и выведет 'X'.

ЧТД

```

C:\>LR03_4_1.EXE
X

```