

# Лекция 15

---

- ❑ Команды программных прерываний
  - ❑ Механизм прерываний
  - ❑ Характеристика системного сервиса
  - ❑ Экранный системный сервис
  - ❑ Алгоритмы преобразования числовых кодов в символы
-

## Команда прерывания: INT n

---

- «Прерывания» - это механизм вызова процедур операционной системы, в котором участвуют процессор и операционная система. Через прерывания (аппаратные и программные по команде Int) исполняемым программам доступен «системный сервис» ОС. Процедуры, вызываемые через механизм прерываний называют «обработчики прерываний».
- Команда программного прерывания:
- INT n ; где n – номер прерывания



# Таблица прерываний

- ❑ ОС создает в памяти **Таблицу прерываний** с адресам своих процедур («указатель сегмента» и «внутрисегментное смещение»).
- Один элемент Таблицы (вектор) содержит адрес одной процедуры.
- ❑ **Номер прерывания** в команде INT– это номер элемента (вектора) Таблицы прерываний (0, 1, 2, ....n), в котором записан адрес процедуры.

Таблица прерываний (для 16-разр.программ)

*Пример:*

Указатель сегмента	Внутрисегм. смещение	
<b>0060</b>	<b>0000</b>	<b>Вектор прерывания 0</b>
<b>1700</b>	<b>0000</b>	<b>Вектор прерывания 1</b>
<b>ff00</b>	<b>0080</b>	<b>Вектор прерывания 2</b>
...	...	

По команде Int 1 процессор начнет исполнять процедуру, размещенную в памяти с адреса **1700:0000**, по команде Int 2 - процедуру, размещенную с адреса **ff00 : 0080** и т.д.

# Механизм выполнения команды **INT n** процессором

---

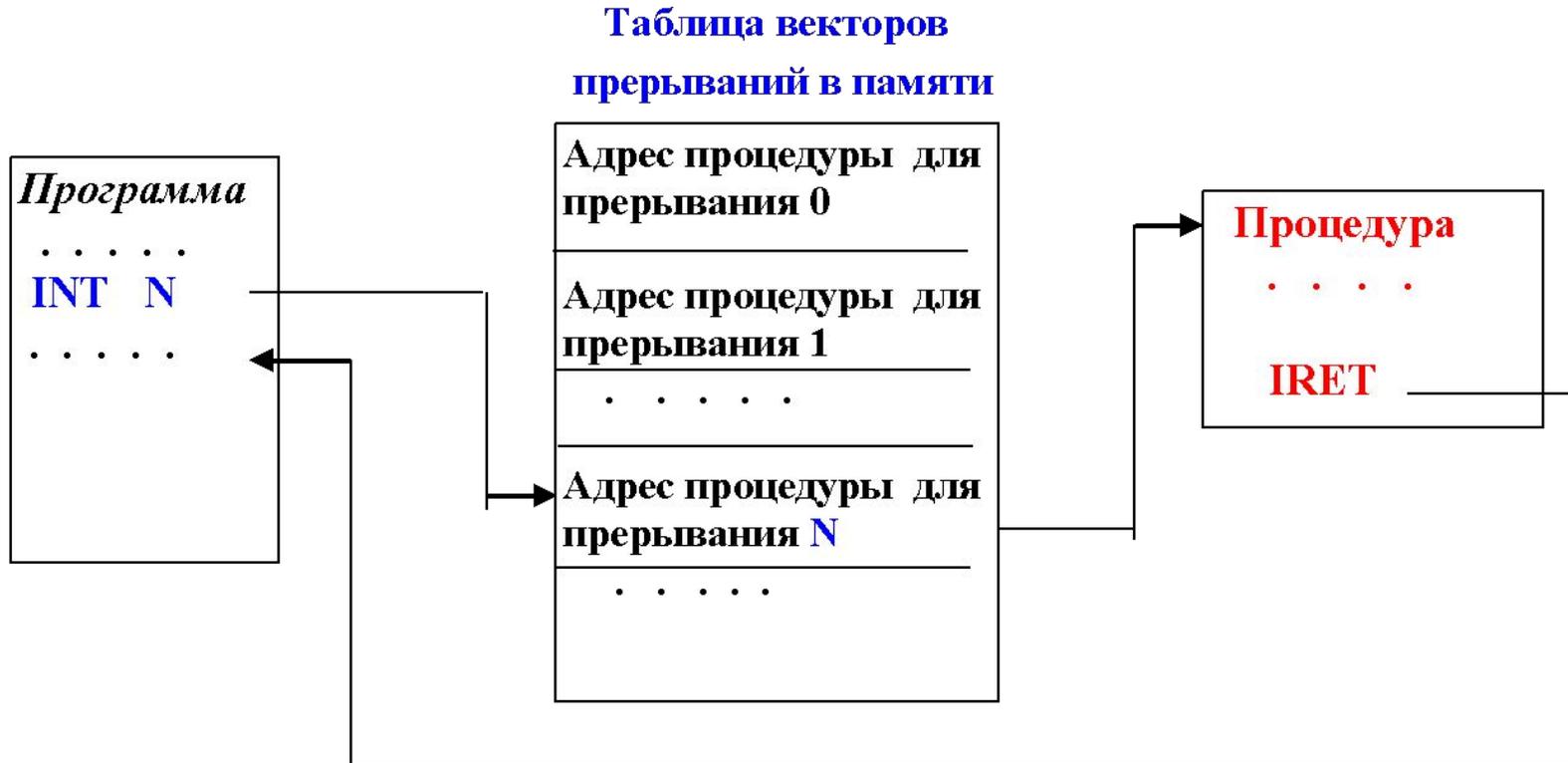
1. **Сохраняет в стеке** текущие значения CS и IP
2. По номеру прерывания **n** вычисляет смещение от начала Таблицы прерываний до **n-го вектора (в байтах):  $n * 4$** .

*Замечание. Адрес самой Таблицы прерываний процессору известен.*

3. **Считывает из вектора** координаты процедуры в регистры CS и IP. Таким образом, процессор себя перенаправил на исполнение кода процедуры.

Процедура, вызываемая по прерыванию, должна заканчиваться командой **IRET**. По команде IRET процессор **восстанавливает из стека** в CS и IP сохраненный адрес возврата в основную программу.

# Иллюстрация программного прерывания



# Характеристика системного сервиса ОС

---

- Операционная система предоставляет исполняемым программам возможность обращаться к своим программным сервисам (функциям) для:
  - **взаимодействия с операционной системой**: запрос на выделение дополнительной памяти для исполняемого кода, сообщение о своем завершении, получение сведений о системе и т.д....
  - **программного обращения к внешним устройствам**: клавиатура, экран, дисковые файлы и другим устройствам системы.
  
- **За каждой системной процедурой ОС закрепляет определенный номер прерывания.**

*Замечание. Подробнее см. учебное пособие Т.Б.Ларина «Использование системного сервиса в ассемблерных программах»*

## «Функции» системного сервиса

---

Системные процедуры, доступные через программные прерывания, в большинстве своем состоят из множества мелких внутренних процедур – «функций».

Чтобы их различать, за ними закреплены номера: 0, 1, 2 . . .

Номер «функции» перед вызовом прерывания надо задать в регистре **АН** в качестве входного параметра процедуры

Разные «функции» могут требовать различные входные параметры в определенных регистрах или памяти. Это справочная информация.

### Пример:

Системная процедура для выгрузки исполняемой программы из памяти – это функция **4ch** прерывания номер **21h**. Она без дополнительных параметров.

; выгрузка исполняемого кода из памяти

**mov ah, 4ch** ; задали номер функции

**int 21h** ; вызов системного сервиса по прерыванию 21h

---

# Системный сервис для вывода символьных данных на экран

---

Доступен через прерывания:

- **Прерывание 21h** - вывод на экран символа из регистра и вывод строки символов (из памяти);
- **Прерывание 10h** - множество детальных функций экранного сервиса базовой системы ввода-вывода (BIOS)

## Вывод символа на экран из регистра DL ( функция 2 прерывания 21h )

---

Вход: DL - Ascii-код символа

- Символ, код которого задан в регистре DL, будет выведен в текущую позицию экрана, которая затем увеличится на 1.

Пример: вывести на экран символ «?»

; подготовка входных параметров

```
mov ah, 2 ; номер функции
```

```
mov dl, '?' ; код символа
```

; вызов системного сервиса

```
int 21h
```

## Вывод на экран символьной строки из памяти ( функция 9 прерывания 21h )

---

Вход: DS:DX - адрес в памяти символьной строки, которая должна заканчиваться символом '\$'

- Содержимое байтов памяти, начиная с указанного адреса, будет выводиться на экран, пока не встретится символ '\$'.

Пример: вывод на экран текста «Мой текст!»

```
text    db    'Мой текст!$' ; строка символов в сегменте данных
        .
        .
        .
        mov  ah, 9
        lea  dx, text
        int  21h
```

---

## Перевод строки на экране

---

Для перевода строки на экране в символную строку включают байты **управляющих кодов**:

**0Dh** - Возврат каретки и **0Ah** - Перевод строки

### Примеры:

; текст, предназначенный для отображения с начала следующей строки экрана

```
text db 0Dh, 0Ah, 'Текст$'
```

; символная строка, после отображения которой на экране будет перевод позиции в начало следующей строки экрана

```
text db 'Текст', 0Dh, 0Ah, '$'
```

# Преобразование числовых кодов в символы

---

На низком уровне автор сам должен заботиться о преобразовании числовых кодов в коды символов (ASCII коды) перед обращением к сервису для вывода их на экран.

В какое символьное изображение можно преобразовать числовой код?  
в виде 2-ого кода, в виде hex-кода, в 10-м виде со знаком или без него

Пример: Каким должно быть символьное представление числового байта 1A перед выводом на экран?

- для двоичного вида '00011010' надо подготовить ASCII –строку кодов: 30 30 30 31 31 30 31 30 (в hex)
  
  - для hex-вида '1A' => ASCII-коды символов: 31 41
  - десятичное б/знака - '26' => ASCII-коды символов: 32 36
  - десятичное знаковое - '+26' => ASCII-коды: 2B 32 36
-

Алгоритмы преобразования базируются на зависимости между числовым байтом цифры: от 00 – 09 и от 0A - 0F и ее символьным кодом

Символ цифры	ASCII-код символа	Числовой байт	Разность между ASCII-кодом и числовым кодом цифры
'0'	30h	00h	30h
'1'	31h	01h	
'2'	32h	02h	
'3'	33h	03h	
...	...	...	
'9'	39h	09h	
'A' / 'a'	41h / 61h	0Ah	37h или / 57h 41h – 0Ah= 37h 61h – 0Ah= 57h
'B' / 'b'	42h / 62h	0Bh	
'C' / 'c'	43h / 63h	0Ch	
'D' / 'd'	44h / 64h	0Dh	
'E' / 'e'	45h / 65h	0Eh	
'F' / 'f'	46h / 66h	0Fh	

## Преобразование числового кода в символьный 2-й вид

---

### Алгоритм:

- циклически сдвигаем числовой код влево на 1 бит
- в зависимости от флага CF (0 или 1) записываем в массив символ - '0' или '1'
- повторяем столько раз, какова разрядность кода

### Отладочный пример:

Исходный числовой байт: **FAh (11111010<sub>2</sub>)**

Желаемый результат:            массив ASCII кодов :  
                                  **31 31 31 31 31 30 31 30h**  
                                  (символы '1 1 1 1 1 0 1 0')

# Преобразование числового кода в символный 16-й вид

---

## Алгоритм

- Из каждой тетрады числового кода создать **отдельный байт с цифрой**
- Байты цифр, значения которых  $\leq 9$ , преобразовать в символ прибавлением **30h**.
- Байты цифр, значения которых от 0A до 0F, преобразовать в hex-символ прибавлением **37h** (для получения большой буквы 'A' – 'F') или **57h** (для маленькой 'a' – 'f').

Отладочный пример: Преобразуем 2-байтный числовой код **F0 1A** в символы

- Сделаем из тетрад отдельные байты и запишем в память, как массив:  
**0F 00 01 0A**
- Преобразуем каждый полученный байт цифры в символ:  
**66 30 31 61h** (это 'f' '0' '1' 'a')

# Преобразование числового кода в символьный 10-й вид

---

**Важно!** Размер массива для записи символов выбрать с учетом символа знака и в расчете на максимальное количество символов, которое может получиться при переводе.

## Алгоритм:

- Привести числовой код к положительному числу и запомнить первый символ для массива: '+' или '-' .
- **Выполнять деление числового кода на 10** , пока частное => 10. К получаемым однобайтным остаткам прибавлять **30h** и записывать их с конца массива символов
- К последнему частному прибавить 30h и записать в массив
- Записать символ знака в массив

Отладочный пример: пусть однобайтное число = 82h (это  $-126_{10}$ )

Ожидаемый массив символов: '-' '1' '2' '6'

Пример. Подготовить к отображению на экране в символьном 10-м виде знаковый числовой код из регистра BL

- Резервируем массив в сегменте данных  
DS:Res : 4 нулевых байта для будущих символов и за ними '\$'.
- Запомним символ знака '+'.

Содержательный алгоритм с ручной отладкой. Пусть, BL содержит код 82h (-126)

- Сравним числовой код с 0. Если =0, запишем в массив символ '0'.  
Если код меньше нуля, запомним знак '-' и получим положительный код: 00 – 82h = 7Eh (126)
- Делим код на 10: остаток 06h и частное 0Ch. Прибавим к байту остатка 30h и запишем с конца массива  
DS:Res: 00 00 00 36h "\$"
- Делим частное на 10: остаток 02h и частное 01h. Прибавим к байту остатка 30h и запишем в массив  
DS:Res: 00 00 32 36h "\$"
- Т.к. частное 01h меньше 10, преобразуем его в символ и запишем в массив  
DS:Res: 00 31 32 36h "\$"
- Запишем в массив сохраненный символ знака '-'  
DS:Res: "-" 31 32 36h "\$"

## Размещение данных в памяти, использование регистров:

---

- ds:Res – адрес области памяти для записи символов
- BL – исходное число;
- DI – косвенный внутрисегментный адрес байта области символов;
- BH – делитель (число 10)
- AX – делимое; AL – частное; AH – остаток от деления (как требует команда DIV)

## Исходный текст программы:

```
data segment use16
res db 4 dup (0), 0dh,0ah,'$'
znak db '+' ; символ знака
data ends
cod segment
assume cs:cod, ds:data
f1: mov ax, data
    mov ds, ax
    lea di, Res+3 ;
    mov bh, 10 ; делитель
    mov al, bl ; al – исходный код
```

---

; проверим число на отрицательность

---

```
    cmp al, 0
    jg short cikl
    jz short Zero
    neg al
    mov ds:znak, '-' ; запомним '-'
; деление, преобразование остатков
cikl:  xor ah, ah
       div bh          ; в ah - остаток
       add ah, 30h
       mov ds:[di], ah
       dec di
       cmp al, 10     ; сравним частное
       jge short cikl
       add al, 30h
       mov ds:[di], al
       dec di
       mov al, ds:znak
       mov ds:[di], al
```

; продолжение справа →

; вызов сервиса вывода строки на экран

```
vyv:  mov ah, 9
      lea dx, Res
      int 21h
```

; выгрузка программы из памяти

```
      mov ah, 4ch
      int 21h
zero: mov ds: Res, 30h
      jmp short vyv
cod  ends
      end f1
```