ОНЛАЙН-ОБРАЗОВАНИЕ

Меня хорошо слышно && видно?

Напишите в чат, если есть проблемы!

Ставьте + если все хорошо

OTUS

Прерывания

Прерывания [BIOS]

Пакулов Артур

1

Прерывания

Прерывания - Interrupt

Прерывание - это временное прекращение основного процесса вычислений для выполнения некоторых запланированных или незапланированных действий, вызванных работой устройств или программы

Аппаратные	Программные	Исключения
• Процессор реагирует на внешний сигнал от устройств (ассинхронные, т.к возникают в случайные моменты)	• Возникают в заранее запланированный момент времени. Синхронные.	• Реакция процессора на нестандартную ситуацию

Вектор прерывания

Адрес процедуры обработки прерываний. Адреса хранятся в таблице по адресу 0000:0000. Каждай адрес хранится в полном виде – сегмент и смещение.

Старшее слово – сегмент, младшее 0 смещение

Вызов прерывания осуществляется командой **INT №**

Диапазон	Описание
00h – 1Fh	прерывания BIOS
20h – 3Fh	прерывания DOS
40h – 5Fh	зарезервировано
60h – 7Fh	прерывания пользователя

Порядок выплнение

IRET - Выход из прерывания

1. Сохраняется в стек регистр флагов
2. Сохраняется в стек регистр CS
3. Сохраняется в стек регистр IP
4. TF = IF = 0
5. Считается полный адрес - 4*№ = xxxxyyyy
6. Cs = xxxx
7. Ip = yyyy

Изменим адрес вектора 0-го прерывания

Пусть вектор будет по адресу 0x1111:0x2222

```
Mov ax, 0x1111
Mov bx, 0x2222
Xor si, si
Mov [si+2], ax
Mov [si+0], bx
Int 0
```

➔ **Старшее слово DWORD – сегмент**

➔ **Младшее слово DWORD - смещение**

Пример – ввод с клавиатуры

```
0x16 – прерывание работы с клавиатурой

Mov ah, 0          ➔ в AH заносим номер нужной функции прерывания


                   AL – ASCII символ нажатой клавиши или 0
Int 16h
                   AH – скан код = расширенный код SSCII
```

Пример – вывод на экран

```
0x10 – прерывание для работы с видеосервисом

Mov ah, 0xE          ➔ функция 10 прерывания

Mov al, 'a'          ➔ в AL заносим выводимый ASCII символ

Int 10h
```

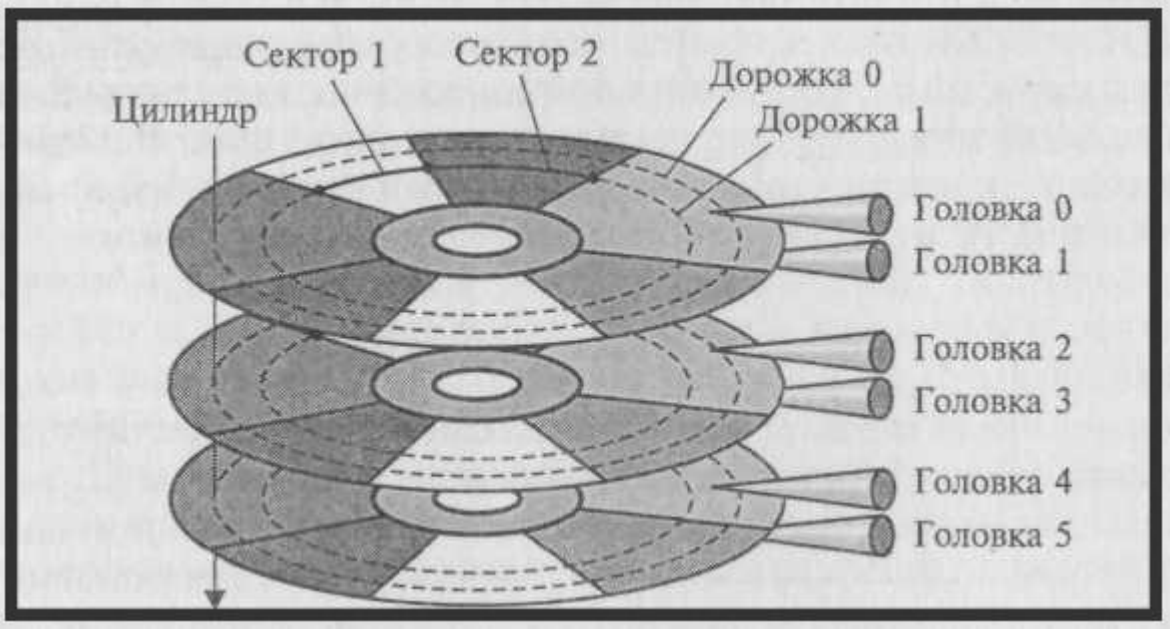
Чтение/Запись

13h прерывание BIOS. 2-я функция - чтение, 3-я - запись

```
02H читать секторы
     вход: DL = номер диска (0=диск A...; 80H=тв.диск 0; 81H=тв.диск 1)
           DH = номер головки чтения/записи
           CH = номер дорожки (цилиндра)(0-n) =¬
           CL = номер сектора (1-n) ===========¦== См. замечание ниже.
           AL = число секторов (в сумме не больше чем один цилиндр)
           ES:BX => адрес буфера вызывающей программы
           0:0078 => таблица параметров дискеты (для гибких дисков)
           0:0104 => таблица параметров тв.диска (для твердых дисков)
   выход: Carry-флаг=1 при ошибке и код ошибки диска в AH.
          ES:BX буфер содержит данные, прочитанные с диска
          замечание: на сектор и цилиндр отводится соответственно 6 и 10 бит:
                 1 1 1 1 1 1
               +5-4-3-2-1-0-9-8-7-6-5-4-3-2-1-0+
            CX: ¦c c c c c c c c C c S s s s s s¦
               +-+-+-+-+-+-+-+-¦-+-+-+-+-+-+-+-+
                               +======> исп. как старшие биты номера цилиндра
```

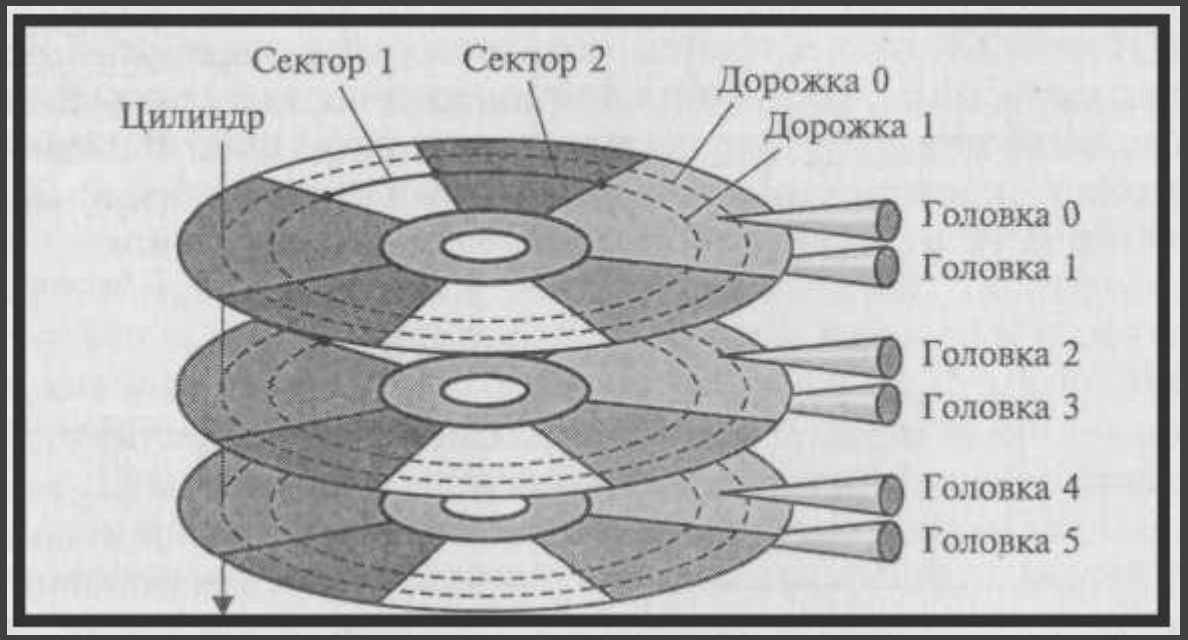
CHS адресация

Цилиндр - **совокупность круговых дорожек одинакового радиуса на всех магнитных поверхностях пластин одного накопителя**



CHS адресация

Максимальный объём диска – 508 Mb



LBA адресация

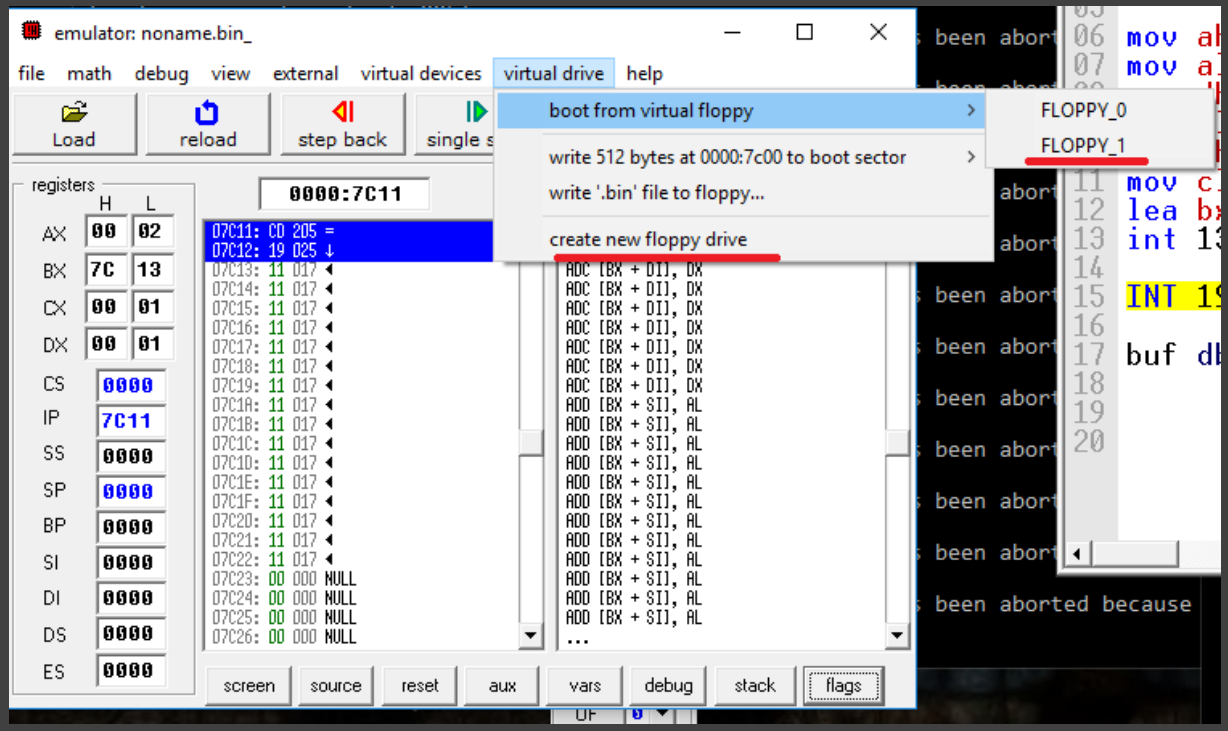
Logical block addressing

Максимальный объём диска – 128 Пиб

Каждый сектор нумеруется по порядку, начиная с 1-ого (нулевого)

Настройка Emu8086

```
Emulator->virtual drive->create new floppy drive
```



Настройка Emu8086

Пример чтения сектора FLOPPY_1

```
#make_boot#

org 7c00h

mov ah, 2    ;read sector
mov al, 1    ;one sector
mov dh, 0
mov dl, 1    ;floppy 1
mov ch, 0
mov cl, 1    ;sector numbe
lea bx, buf
int 13h

INT 19h

buf db 0xFF dup(0)

```

Int 13h

```
Расширенное чтение – 0x42 функция

mov ah, 0x42

lds si, DAP

int 13h
```

Offset	Size	Description
00h	BYTE	Размер этой структуры (0x10)
01h	BYTE	Зарезервировано (0)
02h	WORD	Кол-во секторов
04h	DWORD	Адрес буфера-приёмника
08h	QWORD	Начальный номер сектора (LBA)

Disk Address Packet

Перехват прерывания

1. Бекап оригинального вектора

2. Подмена

3. Реализация нового обработчика

Бекап

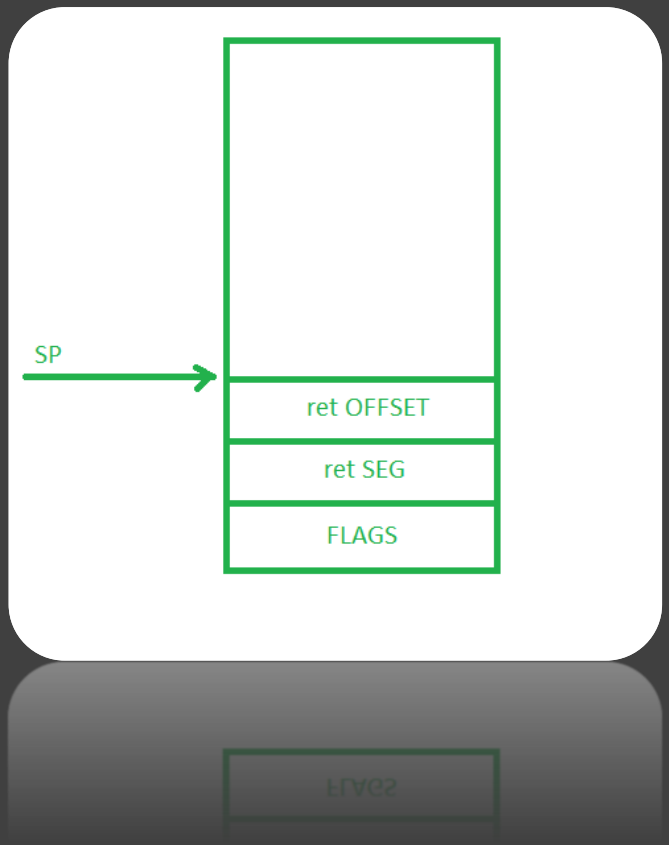
```
org 7c00h

start:

    les ax, [13h * 4] ;assume we hook int 13h

    mov bx, es

    mov [cs:old_vector+0], ax   ;save offset

    mov [cs:old_vector+2], bx   ;save segment

    ...

old_vector:

    dw 0, 0
```

Состояние стека

Сохраняется в стек регистр флагов

Сохраняется в стек регистр CS

Сохраняется в стек регистр IP



Вызов оригинального вектора

После выполнения оригинального обработчика прерывания:

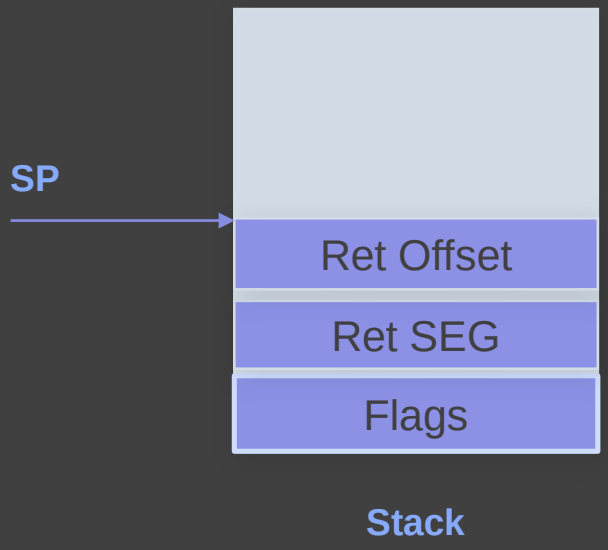
Сразу вернуться в основную программу

Вернуться в перехваченный вектор для обработки результата

Сразу вернуться в основную программу

1. На вершине стека должен быть полный адрес возврата + бекап флагов

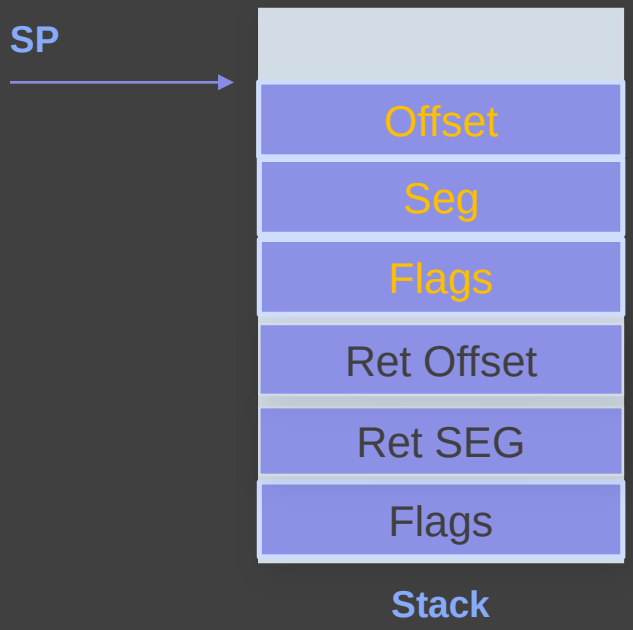
2. Передачу на ООП осуществить: jmp far xxxx:yyyy



Вернуться в перехваченный вектор прерывания

1. Имитируем инструкцию int n: сами сохраняем флаги и длинный адрес возврата

2. Передаём управление на ООП



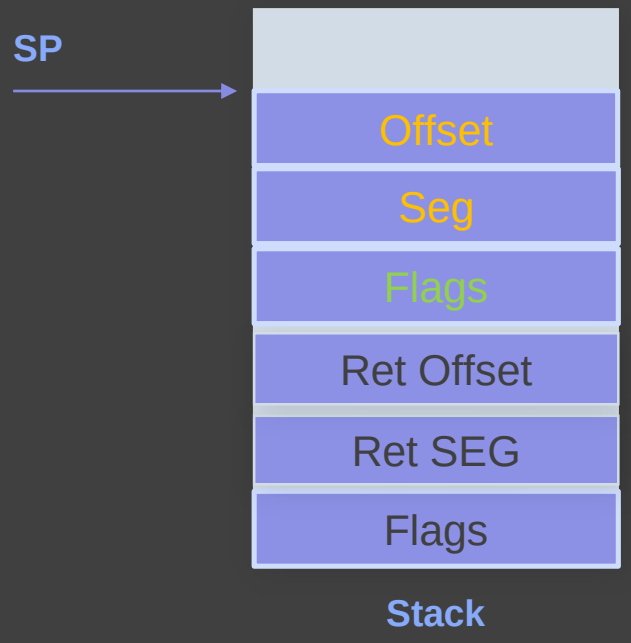
Вернуться в перехваченный вектор прерывания

1. Имитируем инструкцию int n: сами сохраняем флаги и длинный адрес возврата

2. Передаём управление на ООП

1. Pushf

2. Call far xxxx:yyyy



Пример

```
Подменяем результат чтения секторов. Поксорим первый байт прочитанных
данных на 0x90


Перехват int 13h
```

```
cli
mov ax, [4Eh]              ;read segment int 13h
mov [cs:Int_13+2], ax      ;save segment
mov ax, [4Ch]              ;read offset int 13h
mov [cs:Int_13], ax        ;save offset
mov [4eh], cs              ;segment of new IV
mov ax, newInt13           ;offset of new IV
mov [4Ch], ax
sti
```

Пример

```
Подменяем результат чтения секторов. Поксорим первый байт прочитанных
данных на 0x90


New int 13h
                        newInt13:
                                cmp ah, 2
                                jz exec

                                db 0xea          ;EA 0000 0000 | jmp far xxxx:yyyy
                        Int_13:
                                dw 0, 0
                        exec:
Формируем возврат в текущий     pushf
обработчик и вызываем ООП       call far [Int_13]

Сохраняем все результаты        pusha
работы ООП                      pushf

Подменяем результат ООП         xor byte[es:bx], 0x90

Восстанавливаем результаты      popf
работы ООП                      popa

Возврат в программу     iret
```

OTUS

Вопросы???

ДЗ

Написать программу, принимающую на вход два слагаемых в виде hex цифр, считает сумму и выводит её не экран тоже, в hex виде

Пакулов Артур

A.Pakulov.Otus@Gmail.com

Спасибо
за внимание!