Весна 2021

Системное программное обеспечение

Онлайн-лекции

Лекция №11: Работа с файлами. Вывод информации на дисплей.

Доцент, к.т.н. ГОЛЬЦОВ Александр Геннадьевич

Операции с каталогами

- Функции работают по имени, хэндлы не нужны:
  - 41h - удалить файл
  - 43h - прочитать/установить атрибуты файла
  - 36h - узнать свободное место на диске
  - 39h - создать подкаталог
  - 3Ah - удалить подкаталог
  - 3Bh - установить текущий каталог
  - 47h - узнать текущий каталог
  - 4Eh - найти первый файл по образцу
  - 4Fh - найти следующие файлы по образцу

Поиск файла

- Поиск надо инициировать (найти первый), потом продолжать (искать следующий).
- Поиск производится в текущем каталоге.
- Задается шаблон имени файла, включающий * и ?, а также атрибуты.
- Если биты атрибута не заданы - находятся только файлы с атрибутом 0. Если заданы - еще и с заданными атрибутами.
  ➔ Если задать атрибут "подкаталог", найдутся обычные файлы и подкаталоги.
- Используется буфер DTA (data transfer area), который желаттельно зарегистрировать предварительно (ф-ция 1Ah), иначе он в PSP по смещению 80h

Начать поиск

```
INT 21 - DOS 2+         - FIND FIRST ASCIZ (FIND FIRST)
                AH = 4Eh
                CX = search attributes
                DS:DX =     pointer       to ASCIZ filename
Return:         CF = 1 if error
                   AX = Error Code if any
                [DTA] =      data block
                     undocumented fields
                                PC-DOS    3.10
                                  byte         00h:          drive        letter
                                  bytes 01h-0Bh: search template
                                  byte         0Ch:          search attributes
                                  bytes 0Dh-0Eh: offset of entry within directory
                                  bytes 0Fh-14h: ???
                                DOS 2.x (and DOS 3.x except 3.1???)
                                  byte         00h:          search attributes
                                  byte         01h:          drive        letter
                                  bytes 02h-0Ch: search template
                                  bytes 0Dh-0Eh: entrycount within directory
                                  bytes 0Fh-12h: reserved
                                  bytes 13h-14h: cluster number of parent directory
                          byte  15h:     attribute of file found
                          bytes 16h-17h: file time
                          bytes 18h-19h: file date
                          bytes 1Ah-1Dh: file size
                          bytes 1Eh-3Ah: ASCIZ filename+extension
```

Продолжить поиск

```
INT 21 - DOS 2+         - FIND NEXT ASCIZ (FIND          NEXT)
                AH = 4Fh
                [DTA] =      data block from last AH = 4Eh/4Fh call
Return:         CF = 1 if error
                   AX = Error Code
                [DTA] =      data block, see AH = 4Eh above
```

Если поиск выполняется рекурсивно (с заходом в подкаталоги), то для каждого входа в подкаталог нужно настраивать свой DTA, а при возврате к перебору вышестоящего каталога - восстанавливать DTA.

Пример поиска

```
file_spec db "*.*", 0
DTA db 128h dup(0)
.....................................................
mov ax, @Data
mov ds, ax
mov dx,offset DTA
mov ah,1Ah
int 21h

mov ah,4Eh
mov dx,offset file_spec
xor cx, cx
int 21h
jc  quit
```

```
Srch:

; DTA+1Eh = начало имени
; имя кончается 0
; что-то делать

mov ah, 4Fh
int 21h
jc quit
jmp Srch

quit: ; поиск завершен
.............................
```

Буферизация

- В современных ОС при работе с дисками информация эффективно кэшируется.
- В ДОС тоже имелась возможность кэшировать диск (драйвер SmartDrive).
- Несмотря на это, если программа обрабатывает файл последовательно и только читает или только пишет в файл - она будет работать эффективнее, если обмен с файлом будет производиться большими блоками, а "мелкие" чтения или записи будут производиться из или в буфер в памяти.
- Яркий пример такой разновидности файлов - текстовые файлы.

Идея буферизации чтения из текстового файла

- В программе организуется и поддерживается:
  - BufSize - размер буфера, например, 4096 байт
  - Buffer - буфер размером BufSize
  - BufCount - текущее количество байт в буфере, изначально 0
  - BufPos - текущее смещение в буфере, изначально 0
- При считывании информации указывается, сколько байт необходимо прочитать.
- Поскольку файл текстовый - считывается указанное или меньшее количество байт, если вдруг встретится конец строки (символы chr(13), chr(10)). Сами концы строк могут не читаться.
- Байты, начиная с BufPos, копируются в приемный буфер в заданном количестве, пока не встретится конец строки.
- Если случится, что BufPos >= BufCount, то происходит считывание из файла в буфер следующих BufSize байт, при этом вполне может быть, что BufCount<BufSize.

Данные с побитовой организацией

- Иногда необходимо читать или писать в файл произвольное количество не байтов, а битов, размер разовой порции битов произволен и не кратен 8.
- Пример: эффективное кодирование по Хаффману или Шеннону-Фано, UUE-кодирование.
- Для этих целей также удобно организовать буфер в памяти, но дополнительно нужно контролировать битовое смещение в буфере при чтении/записи.

Буферизованная запись в файл битов

- В программе организуется и поддерживается:
  - BufSize - размер буфера, например, 4096 байт
  - Buffer - буфер размером BufSize
  - BufPos - текущее битовое смещение в буфере, изначально 0
  - BufCount - количество байтов, заполненных в буфере
  - Изначально все байты в буфере обнулены.
- Пусть в файл пишется от 1 до 32 бит за раз, тогда передается:
  - Bits - 32-разрядное целое
  - BitCount - количество значащих (младших) бит в Bits
- В ходе записи BufPos и BufCount увеличиваются.
- Если вдруг BufCount превысит BufSize - весь буфер нужно записать в файл, обнулить и начать заполнять сначала.
- Дополнительно должно быть реализовано действие "записать текущий буфер как есть" (flush) - для последней порции.

Добавка порции битов в буфер


```
.model small
.stack 100h
.data
BufSize equ 4096
BufBitSize equ BufSize*8

Buffer db BufSize+4 dup(0)
BufPos dw 0
BufCount dw 0

Bits dd ?,0
BitCount db ?
.code

start:
        mov     ax, @data
        mov     ds,ax

        mov     word ptr Bits+2,0
        mov     word ptr Bits, 701h
        mov     BitCount, 11
        call    WriteBits

        mov     ax,4C00h
        int 21h
```

```
WriteBits:
        mov     cl,byte ptr BufPos
        and     cl,7    ; на сколько бит сдвигать

        mov     bx,0
        mov     di,BufCount
cnt:
        mov     ax,word ptr Bits+bx
        shl     ax,cl
        or      word ptr Buffer[di],ax
        inc     di
        inc     bx
        cmp     bx,4
        je      fin
        jmp     cnt

fin:    mov     ax, BufPos
        add     al,BitCount
        adc     ah,0
        mov     BufPos,ax
        shr     ax,3
        mov     BufCount, ax

        cmp     ax,bufsize
        jb      fin2
        call    WriteDisk
fin2:   ret
```

Комментарии к примеру

- Алгоритм подразумевает "нечеткий размер" буфера: выделено BufSize+4 байт, а "порог срабатывания" для записи на диск - ровно BufSize.
- В ходе записи на диск (WriteDisk):
  - должны быть записаны все BufCount полных байтов
  - буфер должен быть обнулен, кроме того
  - неполный байт, лежащий по смещению BufCount, должен быть записан в начало буфера
  - BufCount = 0; BufPos = BufPos and 7;
- Bits должен содержать в старших не записываемых в буфер битах нули
- Можно предложить алгоритм, производящий не 4, а "сколько нужно" операций пересылки в память буфера, но будет ли он эффективнее - вопрос.

Вывод информации на дисплей

Вывод текста на дисплей

- Через стандартный вывод - функциями DOS (int 21h) по выводу в файл, указав хэндл StdOut.
- Функциями DOS (int 21h) для вывода на экран.
- Функциями BIOS (int 10h).
- Напрямую в видеопамять в текстовом режиме.

- Функции DOS и BIOS работают и в графическом режиме!
- Функции DOS, как правило, интерпретируют и не выводят управляющие символы, функции BIOS выводят изображения для всех символов.
- Вывод функциями DOS может быть перенаправлен: MyLab.exe > a.txt
- При работе функций DOS проверяется нажатие Ctrl-Break

Консольный вывод

```
.model small
.stack 100h
.data
s db 'Hello world'
.code
start:
        mov     ax, @data
        mov     ds,ax

        mov     ah, 40h
        mov     bx,1    ; stdout
        mov     cx, 10
        mov     dx, offset s
        int     21h

        mov     ax, 4c00h
        int 21h

        end start
```

(Остальные функции вывода опробованы на ЛР1.)

Текстовый и графический режим дисплея

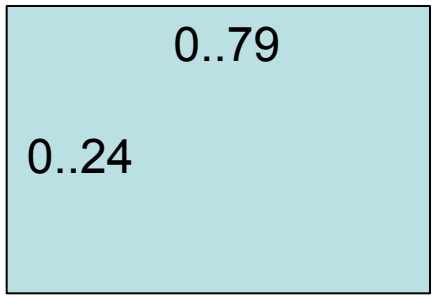
- До сих пор большинство графических адаптеров обратно совместимы с MDA-CGA-EGA-VGA-VESA SVGA и на аппаратном уровне поддерживают все эти режимы.
- Изображение строится в видеопамяти - это сегмент в пределах 1M памяти, расположенный по фиксированному адресу.
- В графическом режиме в видеопамяти хранятся цвета точек (пикселей) изображения (pixel = picture element / picture cell).
- В текстовом режиме - коды и цвет символов, расположенных в знакоместах текстового дисплея.

Переключение видеорежима

- Функция AH=0 прерывания 10h
- AL = номер режима
- Основные (обычно использовавшиеся) режимы:
  - 03h - текстовый цветной 80x25
  - 12h - VGA 640x480 точек 16 цветов
  - 13h - VGA 320x200 точек 256 цветов
  - VESA SVGA, более чем 640x480, 256 и более цветов, выбор режима другой функцией
- Переключение видеорежима - способ очистить экран

Текстовый режим

- Адрес начала видеопамяти B800:0000 для цветных режимов и B000:0000 для монохромных (наследие MDA).
- Видеопамять разбита на 8 страниц (образов экрана) по 4к.
- Можно переключать видимые страницы, по умолчанию видна 0-я.
- Размер экрана - 80x25 или 40x25 знакомест.
- Каждое знакоместо - слово:
  - младший байт - код символа,
  - старший байт - атрибут
- Изменение ячейки видеопамяти немедленно аппаратно приводит к изменению текста на экране.
- На CGA писать в видеопамять следовало только во время обратного хода луча развертки, начиная с EGA - память двухпортовая, писать можно смело всегда.



Атрибут символа

- Атрибут - байт, кодирующий цвет символа и фона для каждого знакоместа.
- Младшая тетрада - символ, старшая - фон.
- Формат байта: krgb irgb
- Старший бит цвета символа i = повышенная яркость
- Старший бит цвета фона k = мигание или повышенная яркость (настраивается, см. int 10h, ax=1003h).

- Пример кодировки цвета символа:
  F = 1111 = белый = яркий синий+зеленый+красный
  0 = 0000 = черный
  6 = 0110 = коричневый = красный + зеленый
  E = 1110 = желтый = яркий красный + зеленый
  8 = 1000 = темно-серый = светло-черный = яркий + ничего

Знакогенератор

- Начиная с EGA, начертание символов можно изменять (символы отображаются аппаратно!).
- Знакоместо - это прямоугольник шириной 8 точек и высотой 14 или 16 строк.
- Начертание одного символа - 14 или 16 байт, 1 = красить цветом символа, 0 = цветом фона в пределах знакоместа.
- Код символа - номер ячейки знакогенератора, используемой для раскраски знакоместа, 0..255.
- Знакогенератор лежит в сегменте памяти A000:0000
- Как им управлять - утратило актуальность, не рассматриваем.
- A = 00 18 3C 3C 7E FF C3 C3 C3 C3 00 00 00 00 ➔

```
00000000
00011000
00111100
01100110
01100110
01111110
11111111
11000011
11000011
11000011
11000011
00000000
00000000
00000000
00000000
```

4-битный цвет

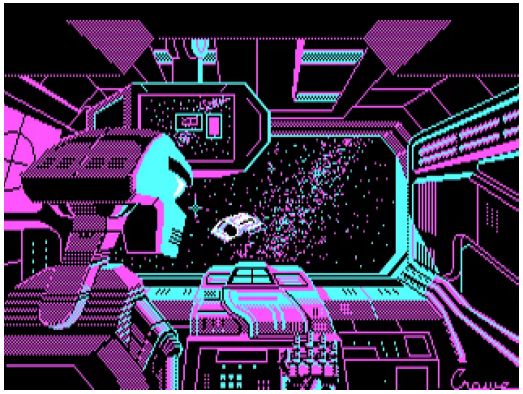
- Начиная с VGA, 4-битный цвет (например, атрибут символа) тоже не используется сам по себе! Он трактуется как ссылка на один из 16 элементов палитры, в которых цвет закодирован в 24-битовом формате, т.е. интенсивность красного, зеленого и синего луча задана в диапазоне от 0 до 255.
- Палитра находится в служебной области памяти и аппаратно интерпретируется видеоадаптером.

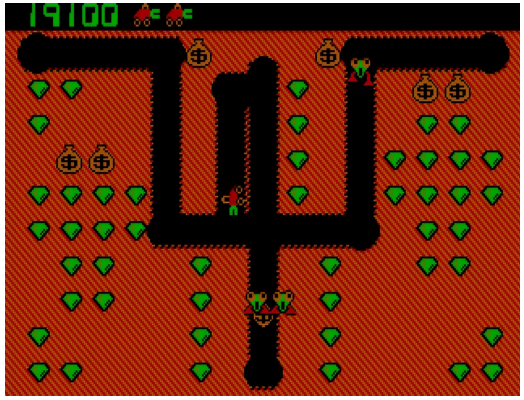
Графический режим

- Эволюционировал от экзотических по современным меркам решений до привычного правила: число в видеопамяти (бит, байт или несколько байт) кодируют цвет точки.
- Причина "экзотических решений" - острая нехватка памяти и прочих аппаратных ресурсов для отрисовки графики.

Графика CGA

- CGA = Color Graphics Adapter
- Режим т. наз. "высокого разрешения" = 320x200, 4 цвета.
- Цвета фиксированные: либо красный-желтый-зеленый, либо (+синий луч ко всем) пурпурный-белый-голубой, 0=цвет фона - один из 8 (3-битная кодировка, задавался отдельно).
- Видеопамять по адресу B800:0000
- 1 пиксель = 4 бита
- Два экрана умещаются в 1 сегмент (32 000 байт на экран)





Графика EGA

- EGA = Enhanced Graphics Adapter
- Полноценная 16-цветная графика 640x350.
- Адрес видеопамяти A000:0000h
- В один сегмент умещалось две видеостраницы 640x350x16 - НО!
- 640*350 / 2 = 112 000 > 64k, это много, это не влезает в один сегмент ➔ "битовые цветовые плоскости" (см. далее)
- Геометрия картинки такова, что эллипс с равными осями, нарисованный по точкам на экране, весьма вытянут по вертикали и не похож на круг.

Битовые плоскости в 16-цветных режимах

- Цель - уместить доступ к информации об изображении 640x350 или 640x480 16 цветов в один сегмент.
- Каждому пикселю соответствует один бит, начиная с адреса A000:0000.
- Но по этому адресу идет доступ не к одному, а сразу к 4 байтам памяти, отвечающим за компоненты 4-битного цвета (iRGB).
- Чтобы нарисовать точку - нужно выбрать плоскости для записи и установить маску (в какие биты байта писать), после чего - произвести запись байта по адресу, соответствующему 8 точкам в "окрестности" нужной точки.

Рисование точки в 16 цветах

- Пусть надо нарисовать голубую точку с координатами (1,1).
- Весь экран имеет координаты (0,0) - (639, 349).
- Голубой цвет = яркий синий+зеленый.
- Одна строка экрана - это 640/8 = 80 байт.
- Маска для записи будет 01000000
- Маска плоскостей будет соответствовать 1011 (iRGB)
- Байт для записи будет по адресу A000:0050h.
- Наконец, выставив через порты все маски в соответствующих регистрах адаптера, можно писать по этому адресу 40h или FFh (нужный бит чтобы был задействован)

Функции рисования BIOS

- Функция AH=0Ch прерывания 10h: нарисовать точку Проделывает все нужные фокусы и рисует точку заданного цвета в нужных координатах в любом графическом режиме.
- Работает очень долго, поскольку для каждой точки устанавливаются все-все настройки и вычисляется адрес обращения к памяти по двумерным координатам.

Графика VGA

- VGA = Video Graphics Array
- 640x480 16 цветов с правильной геометрией.
- 320x200 256 цветов - наконец можно рисовать что-то хоть слегка реалистичное.
- Использование палитры (все цвета из 16 или 256 настраиваемые).
- CGA и EGA были цифровыми (код цвета передавался в кабеле 4 или 8 битами) - VGA стал аналоговым (интенсивность RGB задается при помощи АЦП).

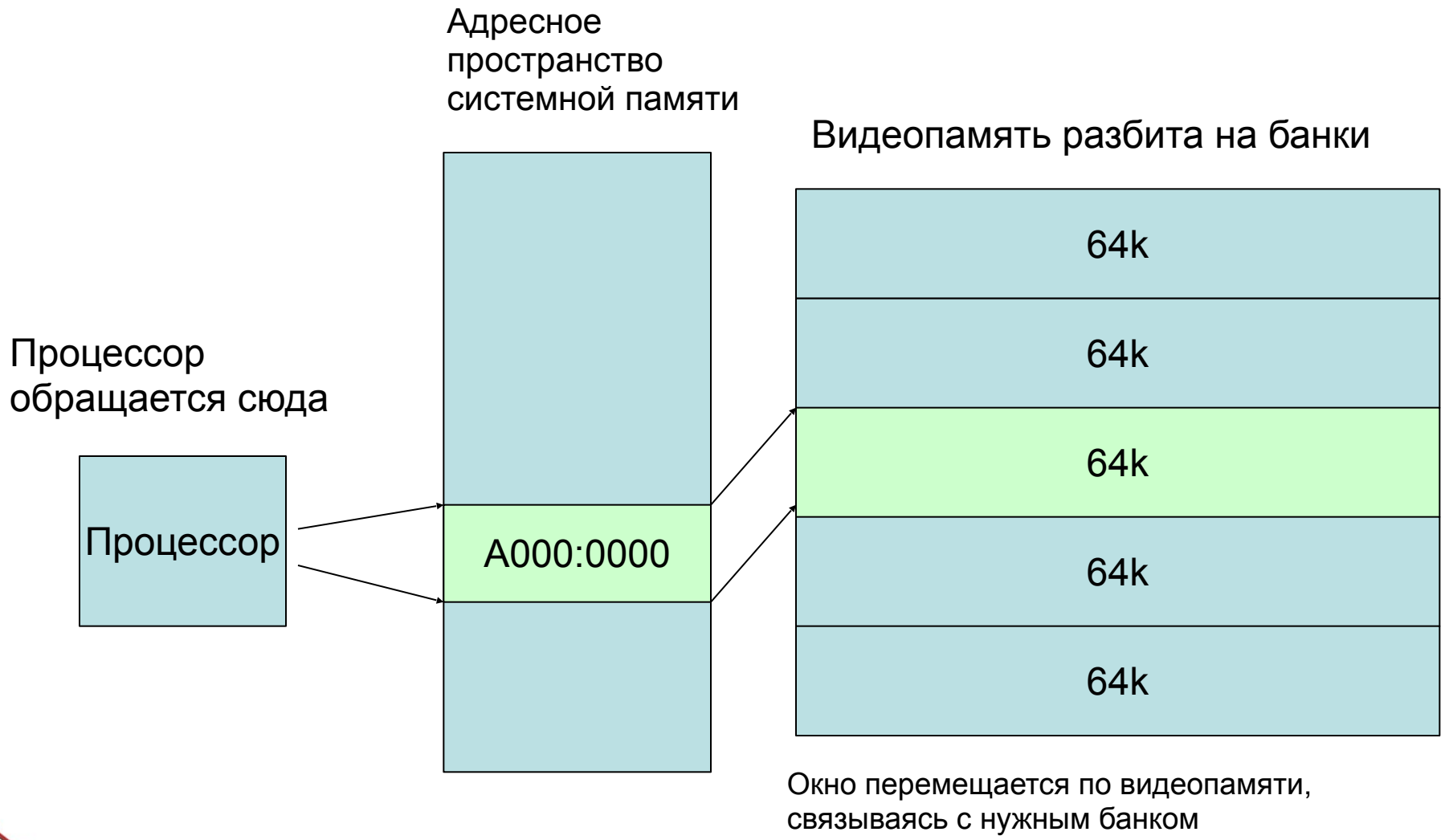
Режим 320x200 256 цветов

- Удобен программисту:
  - вся картинка помещается в 1 сегмент, начиная с адреса A000:0000
  - один байт задает цвет одной точки 0..255
- Для управления палитрой цветов служит функция 10h прерывания 10h

VESA SVGA

- Супер-VGA, попытка организовать и стандартизировать доступ к режимам высокого разрешения и цветности.
- Вся картинка уже никак не влезает в один сегмент (64 килобайта).
- Придуман механизм скользящего окна: изображение строится в видеопамяти на борту графического адаптера, для доступа к ней используется адрес A000:0000, но дополнительно настраивается, какая часть видеопамяти "видна" по этому адресу.
- Одной точке соответствует 1, 2, 3 или 4 байта (256 цетов с палитрой, 16-, 24- или 32-битный цвет без палитры).

Окно графической памяти



Управление VESA SVGA

- Функции управления реализованы как подфункции функции 4Fh прерывания 10h
- Выбор режима:
  AX = 4F02h
  BX = номер режима
- Переместить окно:
  AX = 4F03h
  BH = 0 (0=установка, 1=чтение)
  BL = 0/1 (номер окна)
  DX = номер банка, с которым связывается адрес A000:0000

Некоторые режимы SVGA

8-битные режимы (256 цветов):
- 013h: 320x200 (64 Кб) (standard VGA)
- 100h: 640x400 (256 Кб)
- 101h: 640x480 (320 Кб)
- 103h: 800x600 (512 Кб)
- 105h: 1024x768 (768 Кб)
- 107h: 1280x1024 (1,3 Мб)

- 15-битные режимы (32 К цветов):
- 10Dh: 320x200 (128 Кб)
- 110h: 640x480 (768 Кб)
- 113h: 800x600 (1 Мб)
- 116h: 1024x768 (1,5 Мб)
- 119h: 1280x1024 (2,5 Мб)

- 16-битные режимы (64 К цветов):
- 10Eh: 320x200 (128 Кб)
- 111h: 640x480 (768 Кб)
- 114h: 800x600 (1 Мб)
- 117h: 1024x768 (1,5 Мб)
- 11Ah: 1280x1024 (2,5 Мб)

24-битные 16 М цветов:
- 10Fh: 320x200 (192 Кб)
- 112h: 640x480 (1 Мб)
- 115h: 800x600 (1,4 Мб)
- 118h: 1024x768 (2,3 Мб)
- 11Bh: 1280x1024 (3,7 Мб)

Спасибо за внимание.